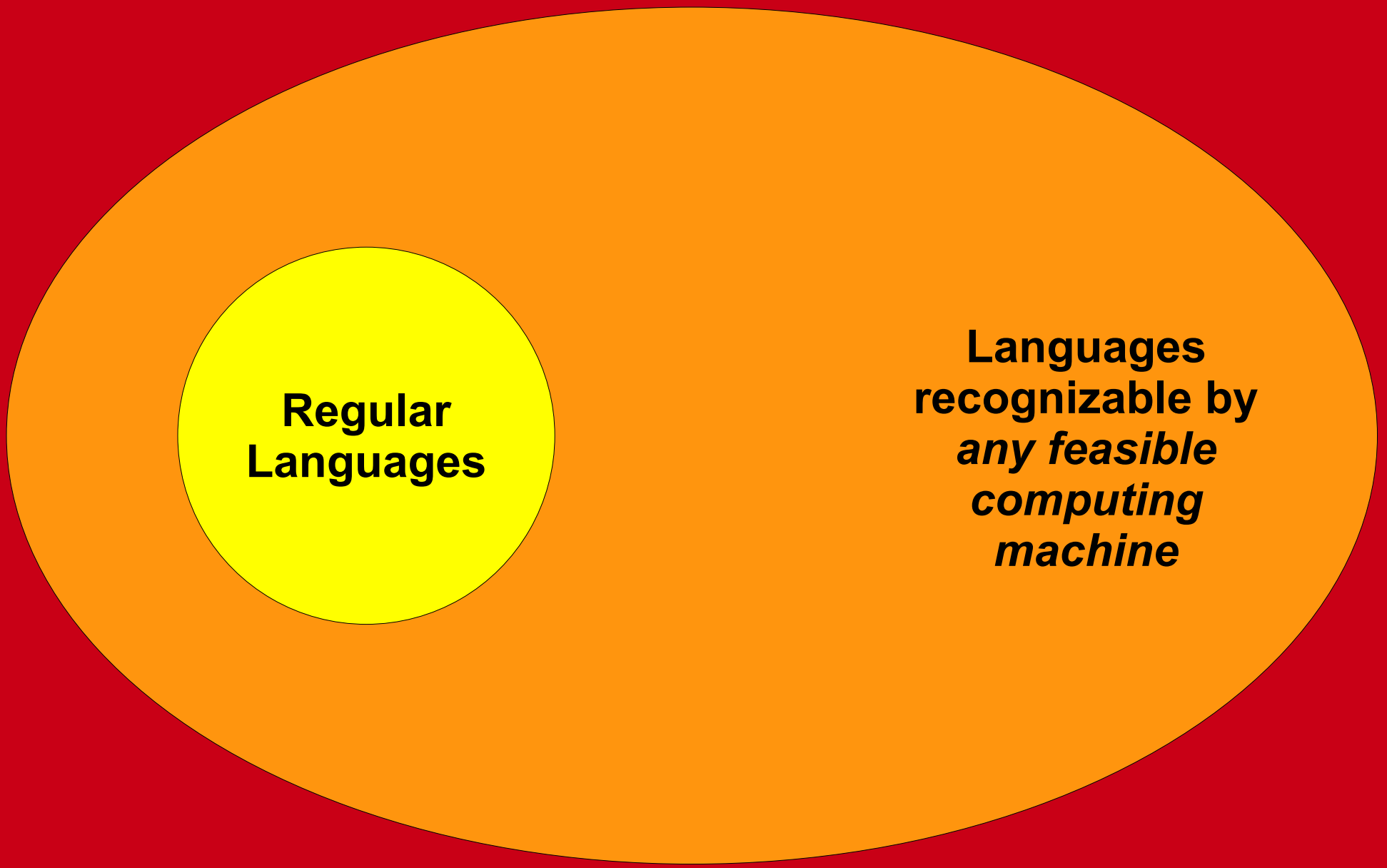
Turing Machines

Part One

What problems can we solve with a computer?


Regular Languages
Languages recognizable by *any feasible computing machine*
All Languages

That same drawing, to scale.

All Languages

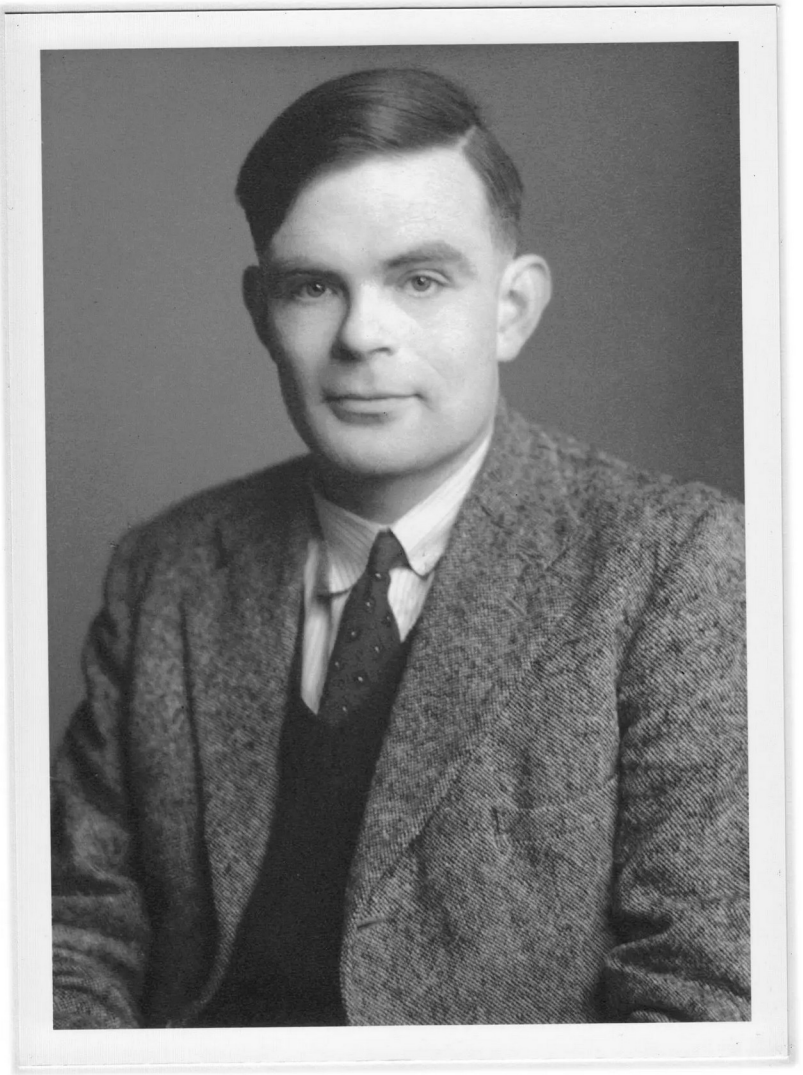
The Problem

- Finite automata accept precisely the regular languages.
- We've already seen cases where we may need unbounded memory to recognize languages.
  - e.g. $\{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$ requires unbounded counting.
- How do we model a computing device that has unbounded memory?

A Brief History Lesson

Turing Machines

- In March 1936, Alan Turing (aged 23!) published a paper detailing the ***a-machine*** (for ***automatic machine***), an automaton for computing on real numbers.
- They're now more popularly referred to as ***Turing machines*** in his honor.
- He also later made contributions to computational biology, artificial intelligence, cryptography, etc. Seriously, Google this guy.



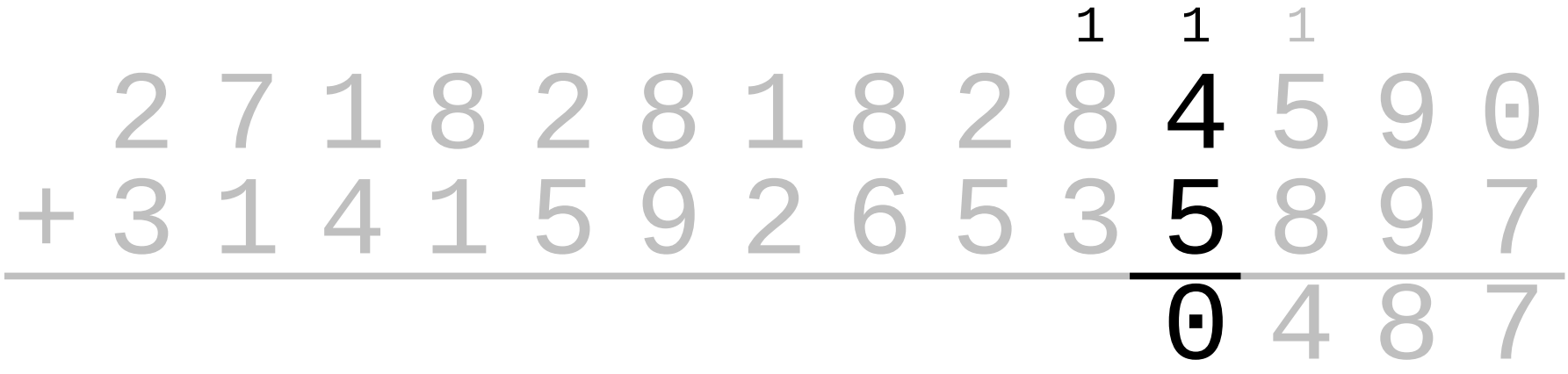
https://static01.nyt.com/images/2019/06/10/obituaries/10overlooked-alanturing-1/f86fc26c86b444c391fb412bb3dcec76-superJumbo.jpg?

$$
\begin{array}{r}
2\,7\,1\,8\,2\,8\,1\,8\,2\,8\,4\,5\,9\,0 \\
+\,3\,1\,4\,1\,5\,9\,2\,6\,5\,3\,5\,8\,9\,7 \\
\hline
\end{array}
$$

$$
\begin{array}{r}
2\,7\,1\,8\,2\,8\,1\,8\,2\,8\,4\,5\,9\,0 \\
+\,3\,1\,4\,1\,5\,9\,2\,6\,5\,3\,5\,8\,9\,7 \\
\hline
7
\end{array}
$$

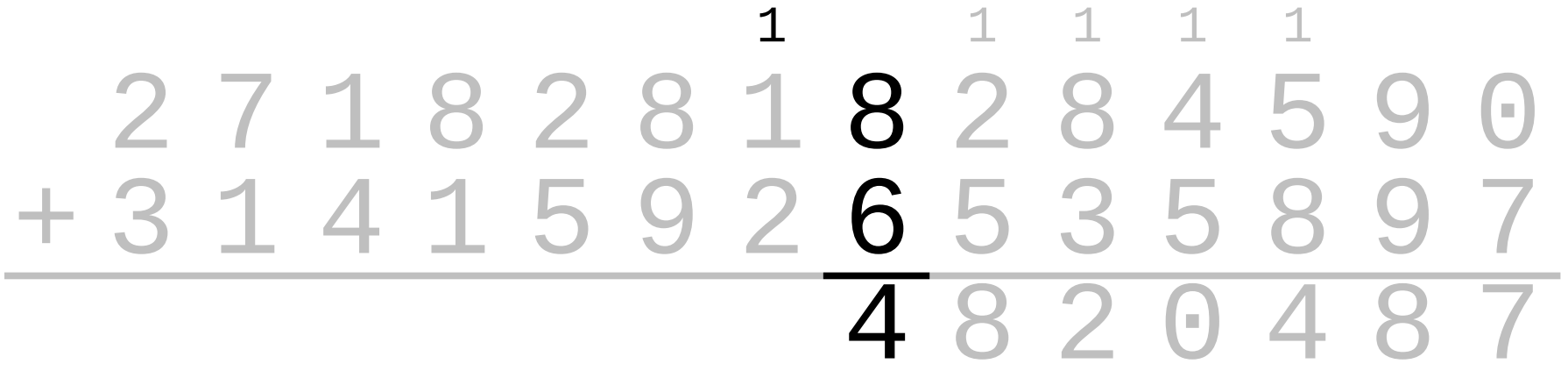
```
                  1
  2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
-----------------------------
                    8 7
```

```
    1  1
  2718281828459 0
+3141592653589 7
---------------
              487
```


1 1 1
2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
0 4 8 7

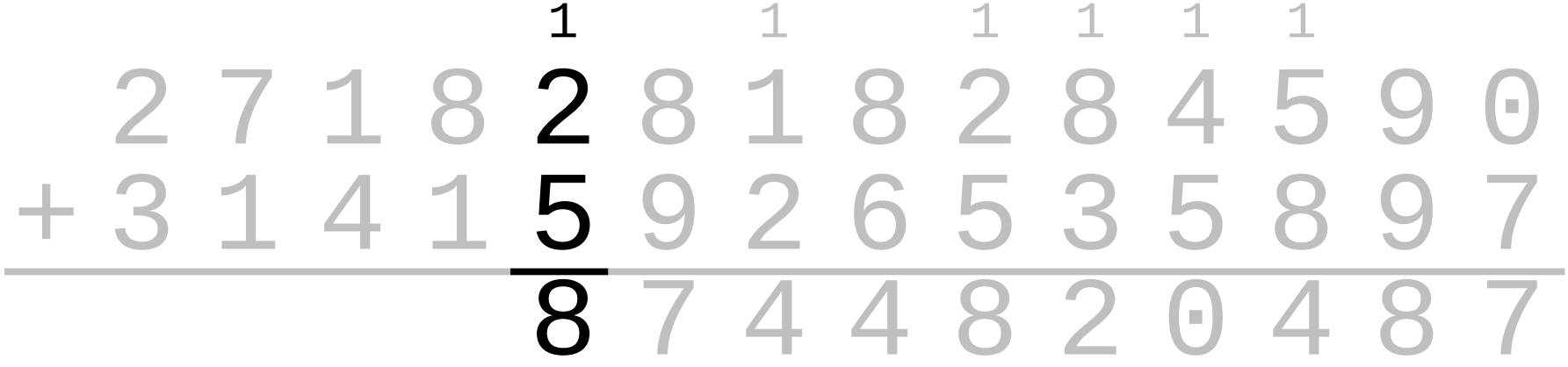
```
                    1  1  1  1
  2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
-----------------------------
                    2 0 4 8 7
```

```
                    1  1  1  1
  2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
-----------------------------
                  8 2 0 4 8 7
```


1 1 1 1 1
2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
4 8 2 0 4 8 7

```
      1     1 1 1 1
  2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
-----------------------------
      4 4 8 2 0 4 8 7
```

```
    1     1   1 1 1 1
  2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
-----------------------------
      7 4 4 8 2 0 4 8 7
```


1 1 1 1 1
2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
8 7 4 4 8 2 0 4 8 7

```
    1   1   1 1 1 1
  2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
-----------------------------
    9 8 7 4 4 8 2 0 4 8 7
```

```
    1   1   1 1 1 1
  2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
-----------------------------
      5 9 8 7 4 4 8 2 0 4 8 7
```

```
    2 7 1 8 2 8 1 8 2 8 4 5 9 0
  + 3 1 4 1 5 9 2 6 5 3 5 8 9 7
  -------------------------------
      8 5 9 8 7 4 4 8 2 0 4 8 7
```

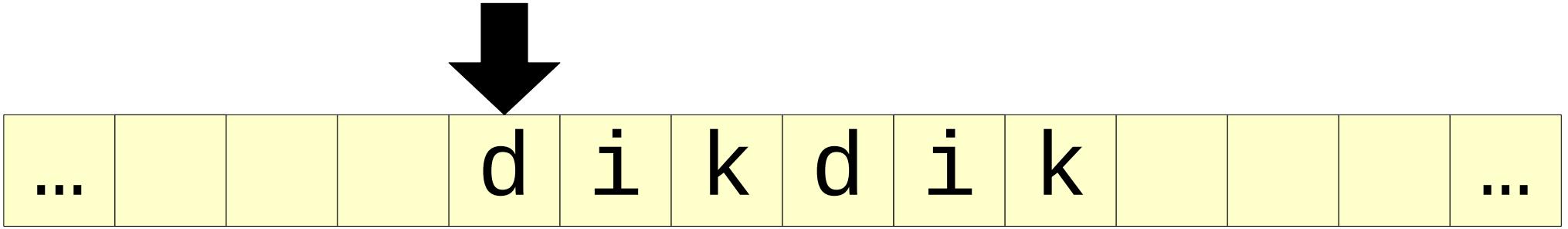
```
          1   1   1 1 1 1
  2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
-----------------------------
  5 8 5 9 8 7 4 4 8 2 0 4 8 7
```

```
          1   1   1 1 1 1
  2 7 1 8 2 8 1 8 2 8 4 5 9 0
+ 3 1 4 1 5 9 2 6 5 3 5 8 9 7
-----------------------------
  5 8 5 9 8 7 4 4 8 2 0 4 8 7
```

Key Idea: Even if you need huge amounts of scratch space to perform a calculation, at each point in the calculation you only need access to a small amount of that scratch space.

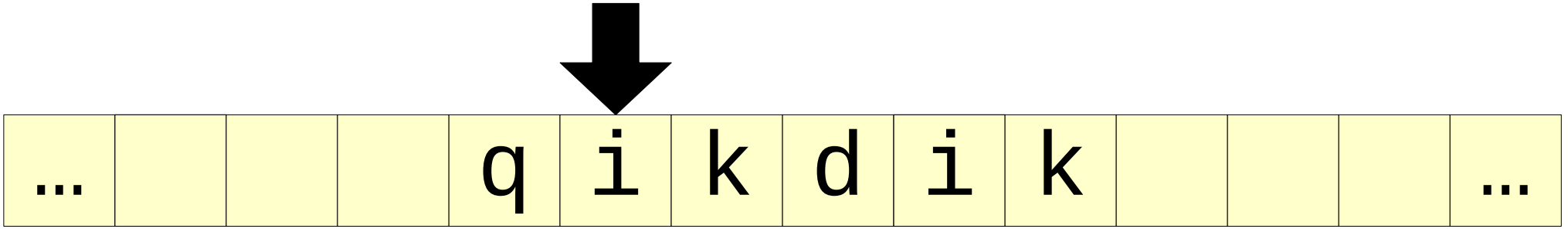
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



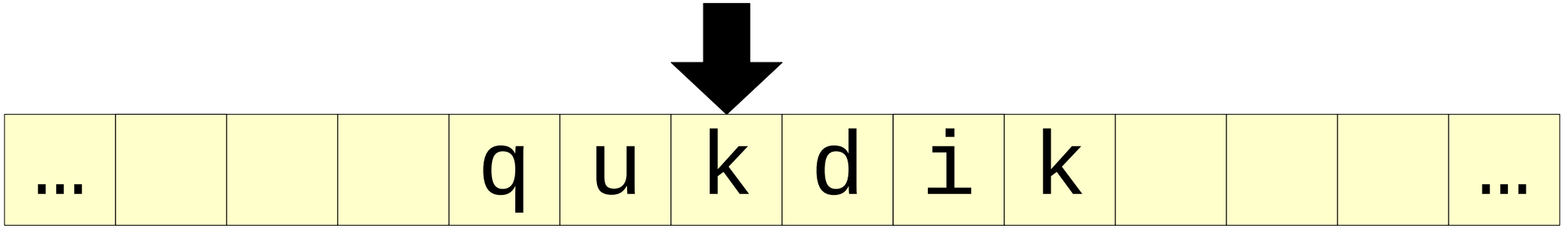
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



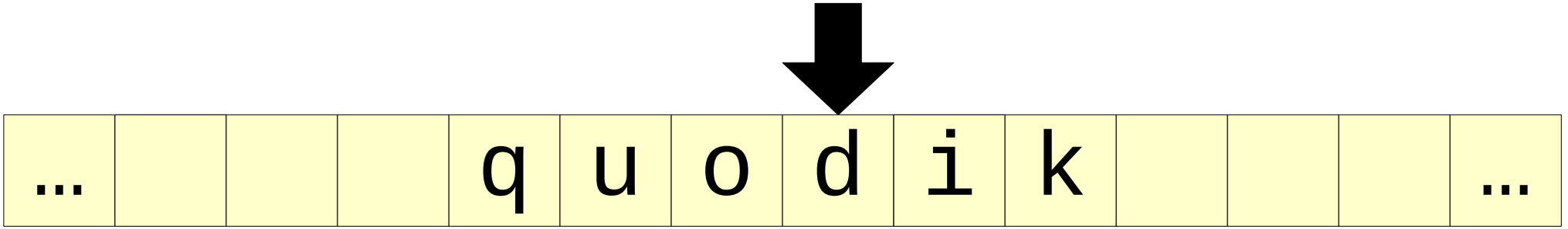
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



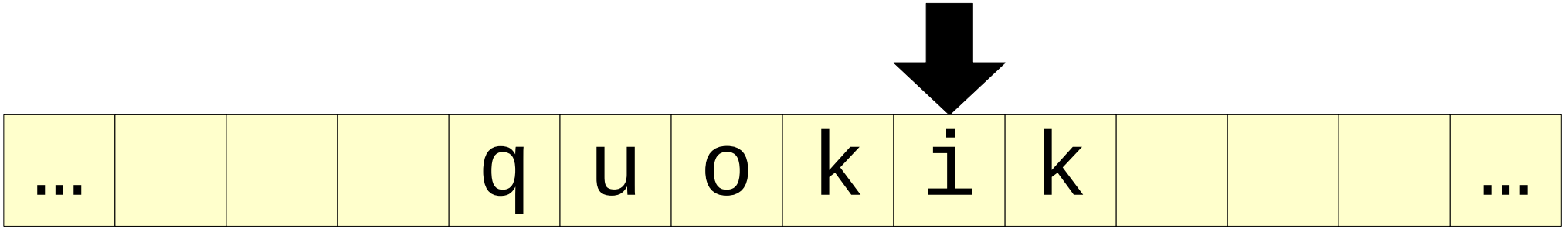
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



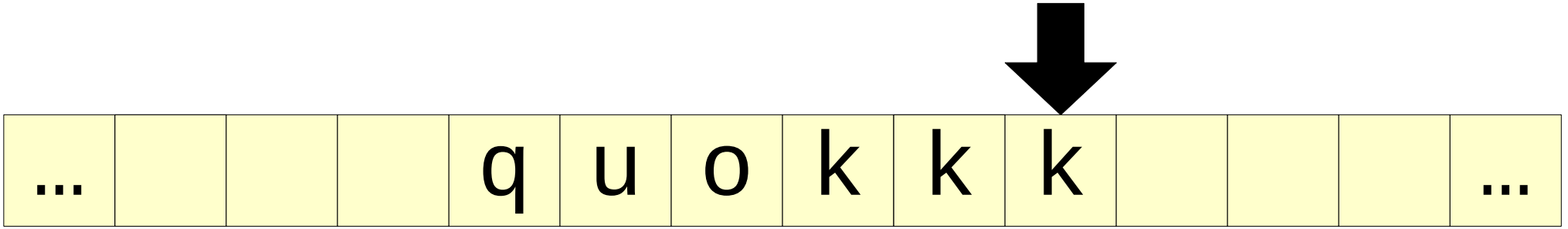
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



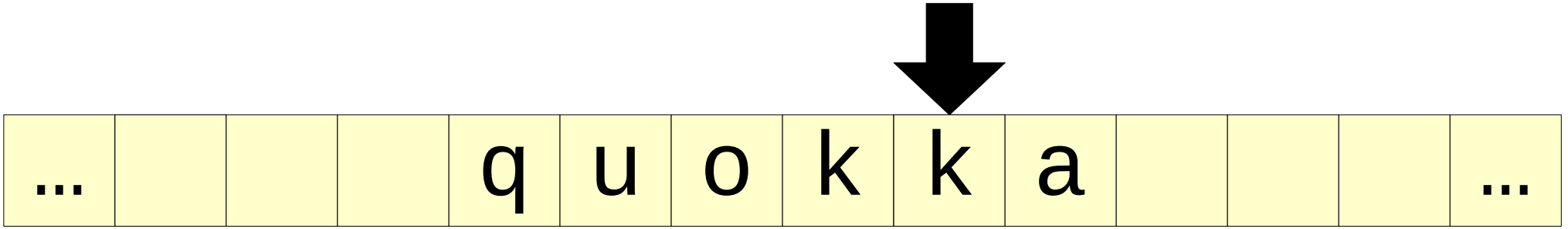
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



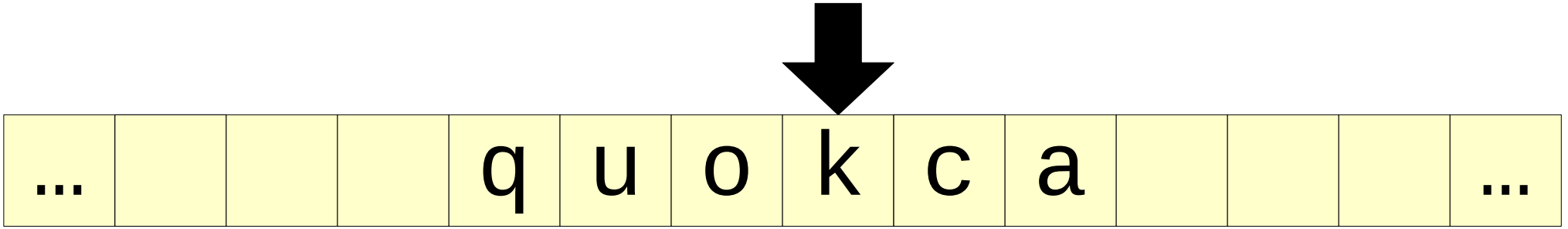
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



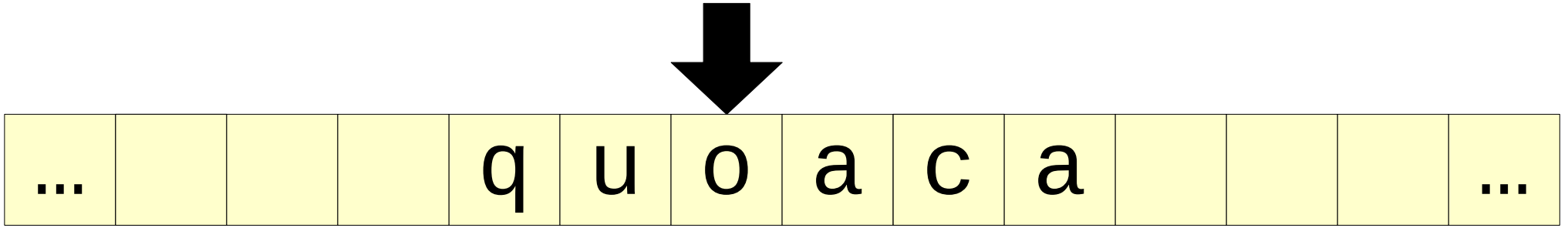
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



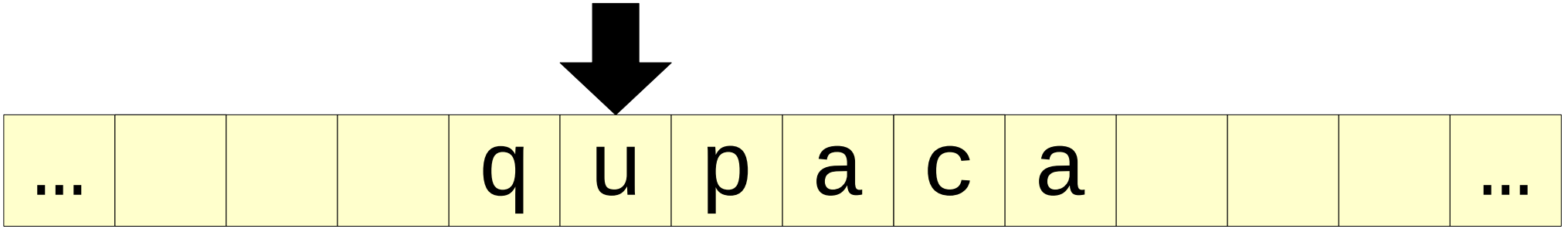
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



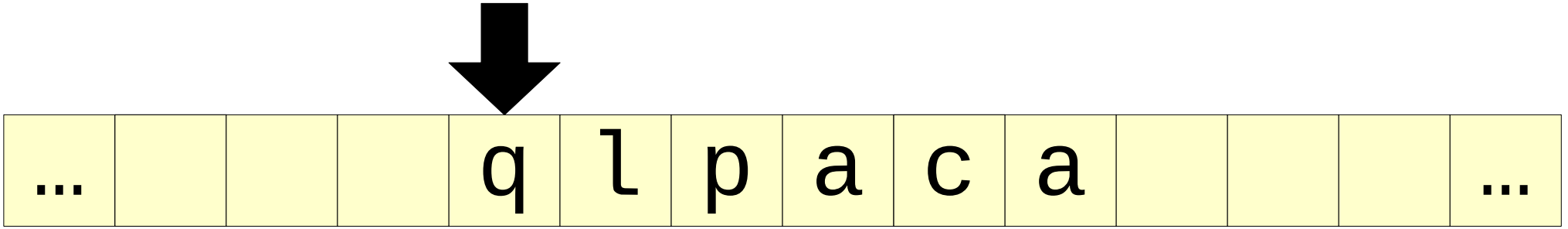
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



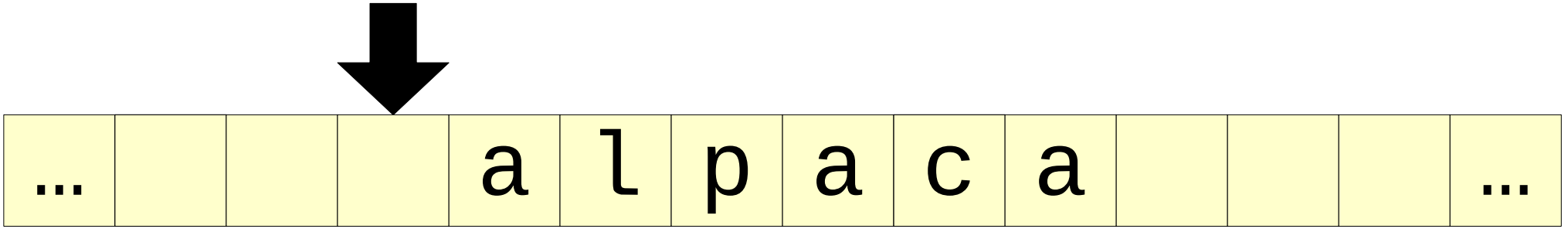
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an ***infinite tape*** subdivided into a number of ***tape cells***.
- A Turing machine can only see one tape cell at a time, the one pointed at by the ***tape head***.
- The Turing machine can
  - read the cell under the tape head,
  - (possibly) change which symbol was written under the tape head, and
  - move its tape head to the left or to the right.



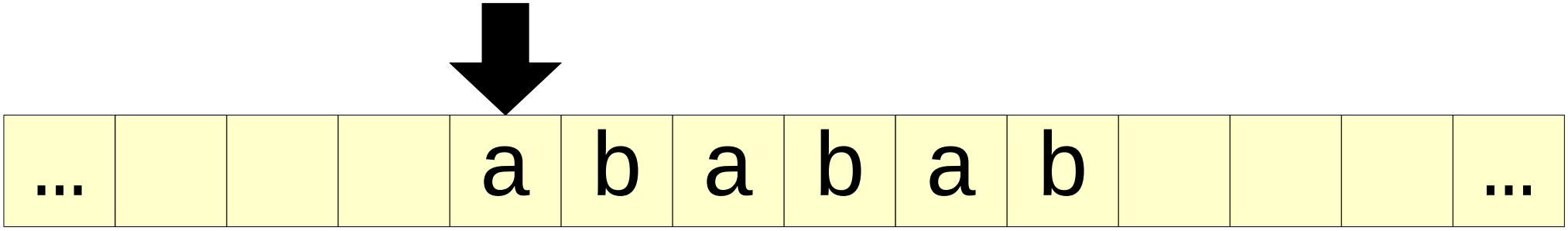
Turing Machines

- Over the years, there have been many simplifications and edits to Turing's original automata.
  - In practice, electronic computers are written in terms of individual instructions rather than states and transitions.
  - Turing's original paper deals with computing individual real numbers; we typically want to compute functions of inputs.
- What we're going to present as "Turing machines" in this class differ significantly from Turing's original description, while retaining the core essential ideas.
  - (Our model is closer to Emil Post's *Formulation 1* and Hao Wang's *Basic Machine B*, for those of you who are curious.)

Turing Machines

- A TM is a series of instructions that control a tape head as it moves across an infinite tape.
- The tape begins with the input string written somewhere, surrounded by infinitely many blank cells.
  - Rule: The input string cannot contain blank cells.
- The tape head begins above the first character of the input. (If the input is ε, the tape head points somewhere on a blank tape.)

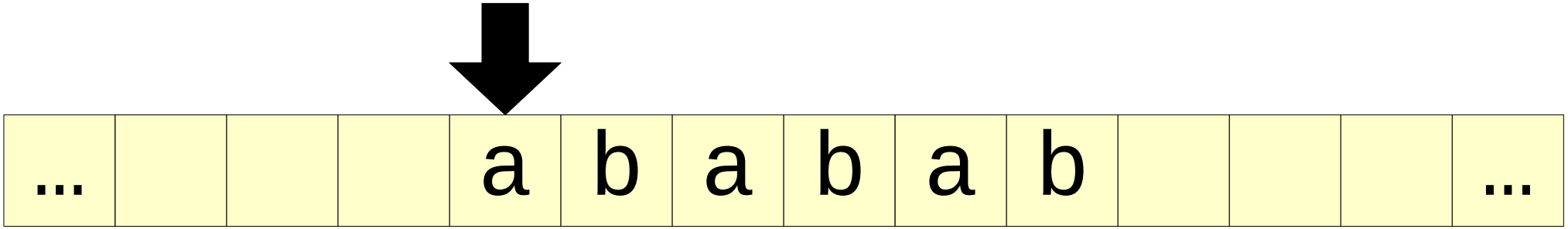
```
Start:
  If Blank Return True
  If 'b' Return False
  Write 'x'
  Move Right
  If Not 'b' Return Fals
  Write 'x'
  Move Right
  Goto Start
```



Turing Machines

- We begin at the `Start` label.
- Labels indicate different sections of code. The name `Start` is special and means “begin here.”
- Labels have no effect when executed. We just move to the next line.

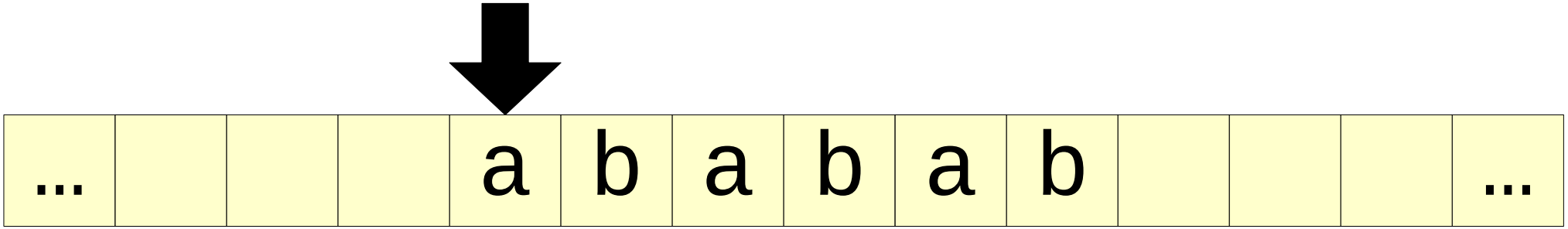
```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



Turing Machines

- We begin at the `Start` label.
- Labels indicate different sections of code. The name `Start` is special and means “begin here.”
- Labels have no effect when executed. We just move to the next line.

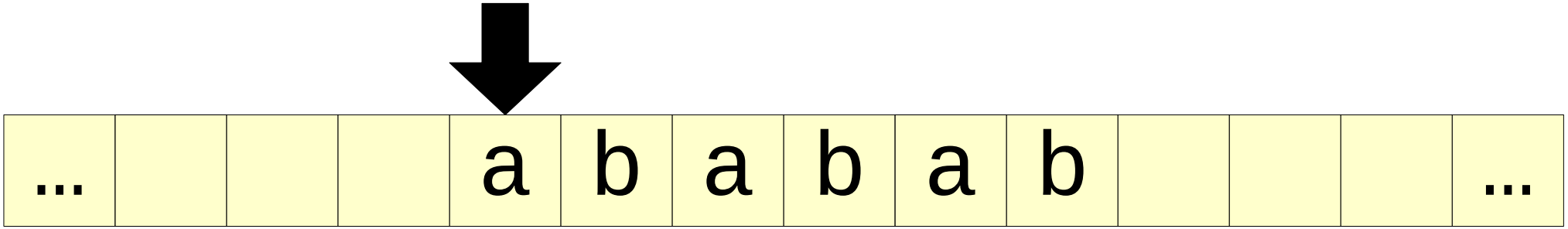
```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



Turing Machines

- We begin at the `Start` label.
- Labels indicate different sections of code. The name `Start` is special and means “begin here.”
- Labels have no effect when executed. We just move to the next line.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



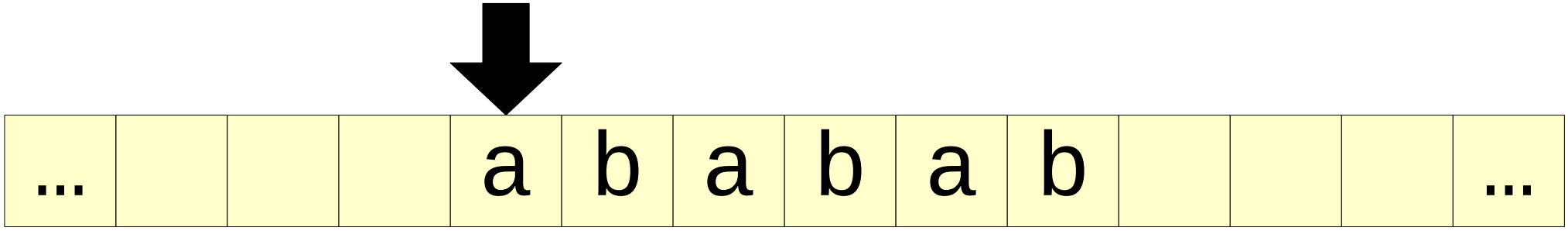
Turing Machines

- A statement of the form

  If *symbol command*

  checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



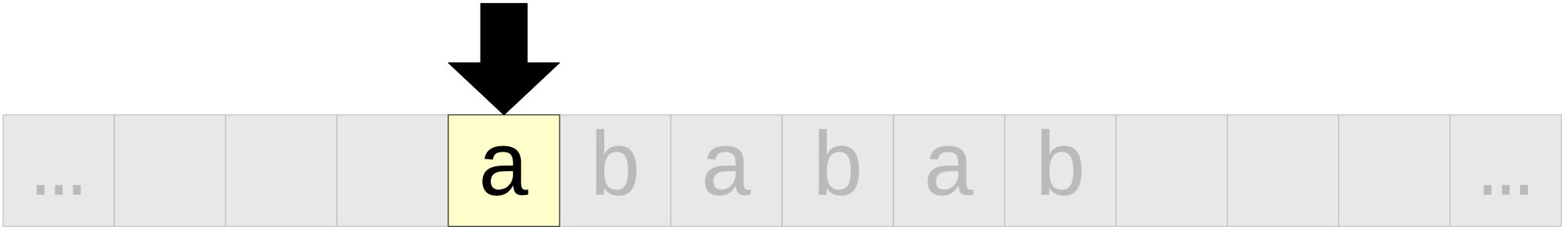
Turing Machines

- A statement of the form

  If *symbol command*

  checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



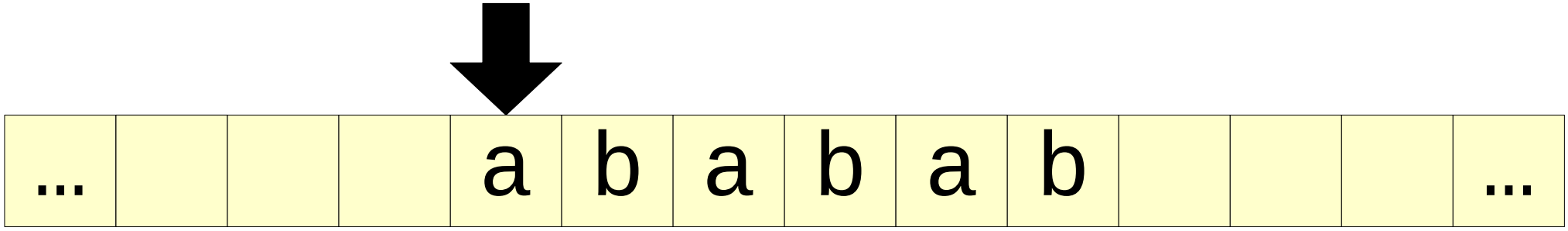
Turing Machines

- A statement of the form

  If *symbol command*

  checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



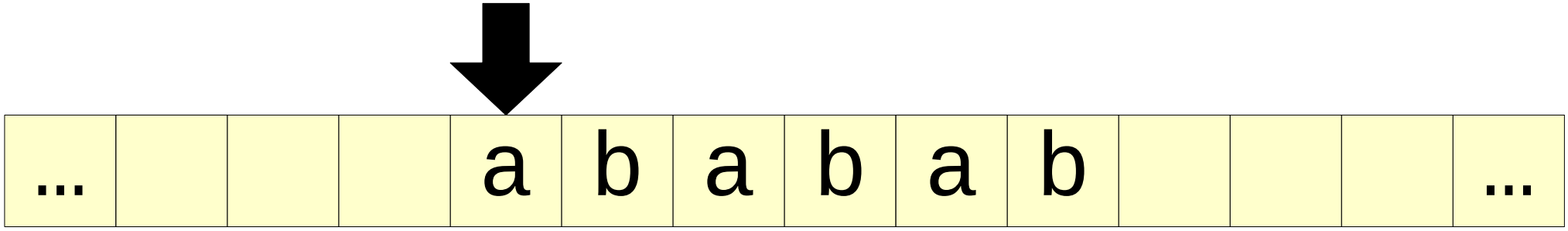
Turing Machines

- A statement of the form

  If *symbol command*

  checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



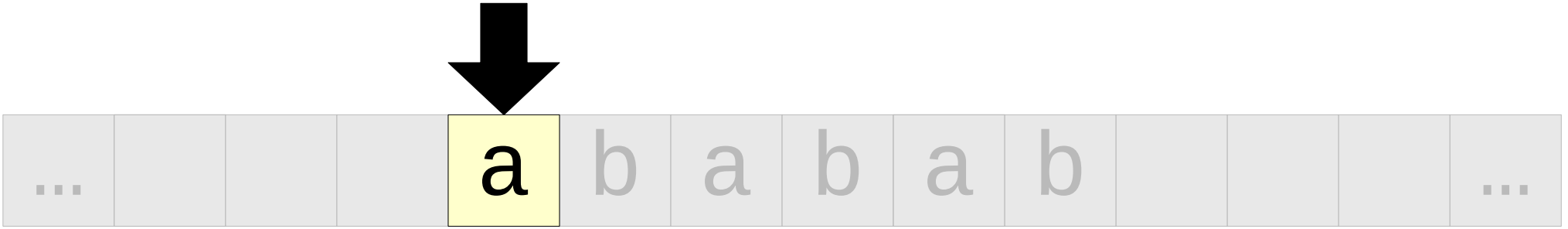
Turing Machines

- A statement of the form

  If *symbol command*

  checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



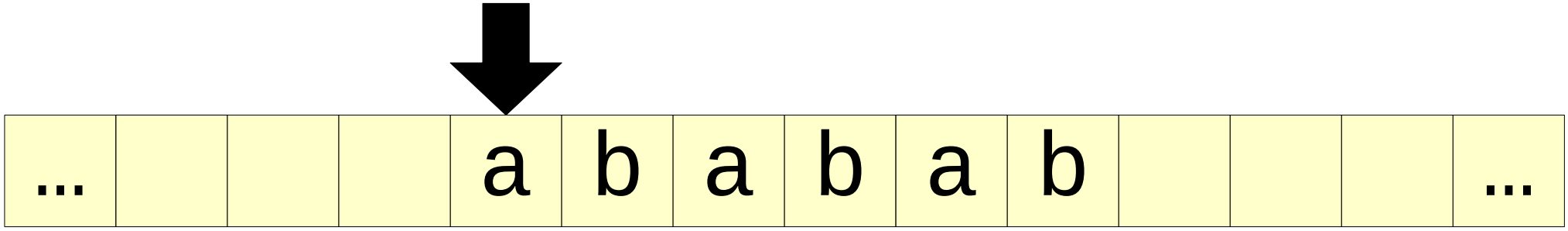
Turing Machines

- A statement of the form

  If *symbol command*

  checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



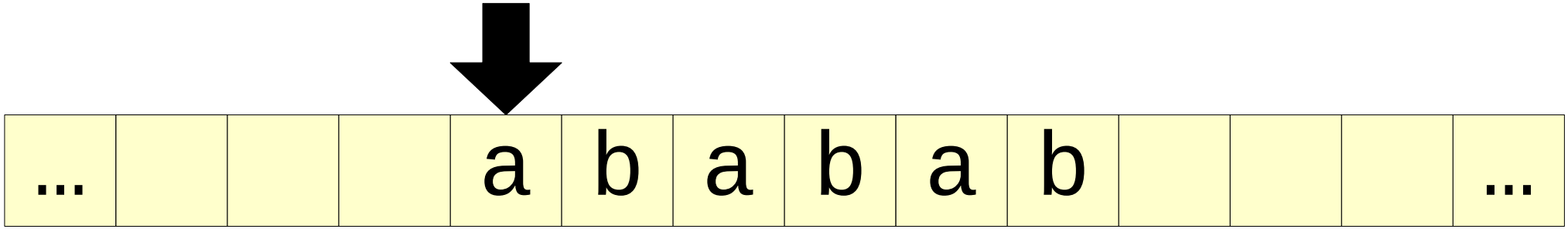
Turing Machines

- A statement of the form

  If *symbol command*

  checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



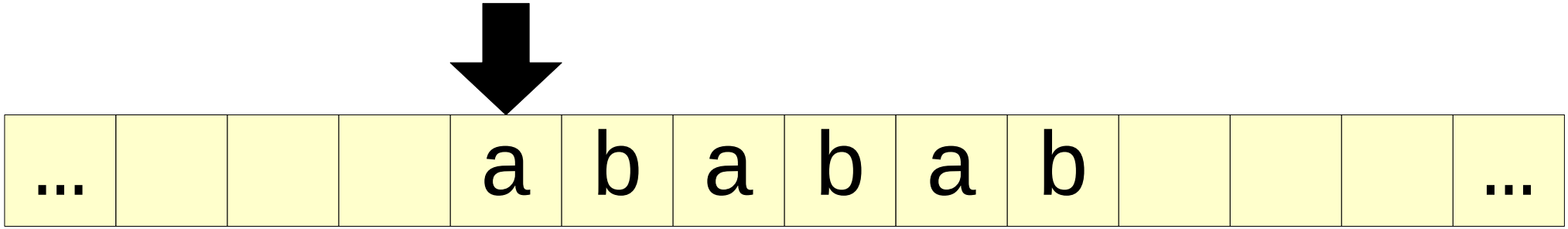
Turing Machines

- The statement

  `Write` *symbol*

  writes *symbol* to the cell under the tape head.
- The *symbol* can either be `Blank` or a character in quotes.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



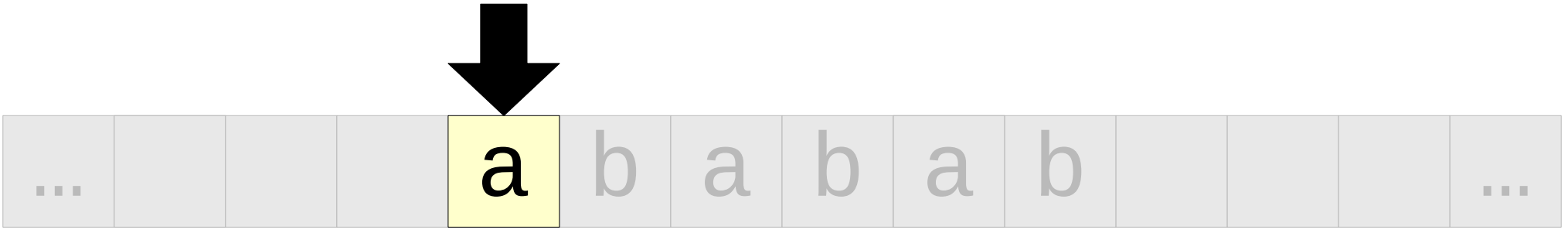
Turing Machines

- The statement

  `Write` *symbol*

  writes *symbol* to the cell under the tape head.
- The *symbol* can either be `Blank` or a character in quotes.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



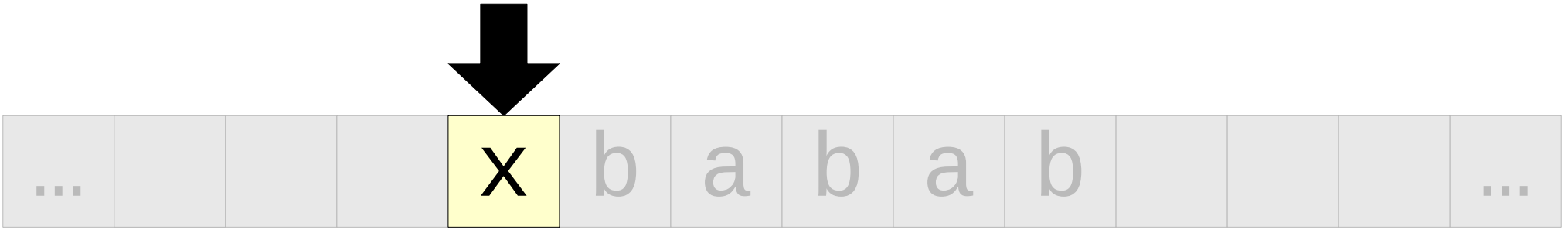
Turing Machines

- The statement

  `Write` *symbol*

  writes *symbol* to the cell under the tape head.
- The *symbol* can either be `Blank` or a character in quotes.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



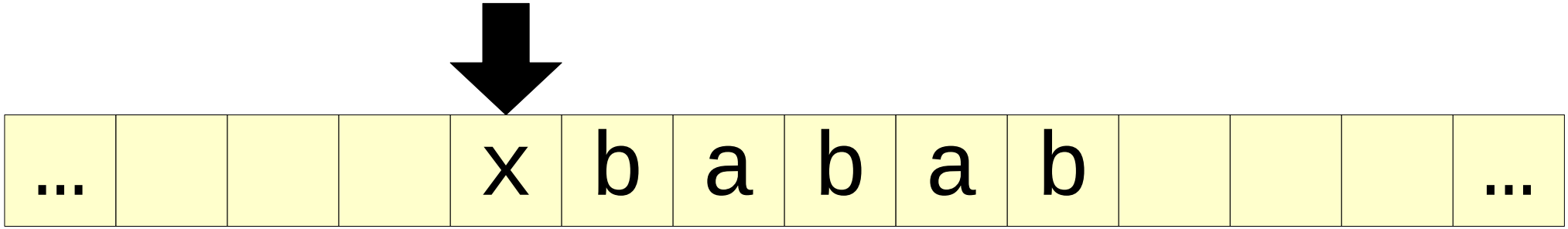
Turing Machines

- The statement

  `Write` *symbol*

  writes *symbol* to the cell under the tape head.
- The *symbol* can either be `Blank` or a character in quotes.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



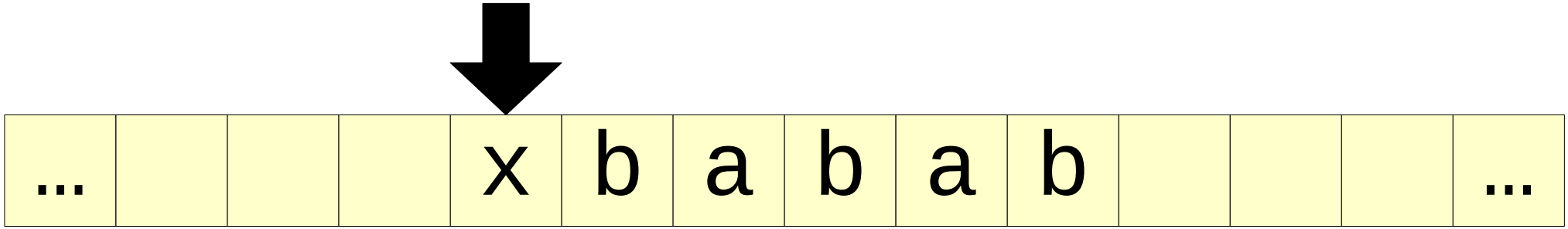
Turing Machines

- The statement

  `Write` *symbol*

  writes *symbol* to the cell under the tape head.
- The *symbol* can either be `Blank` or a character in quotes.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



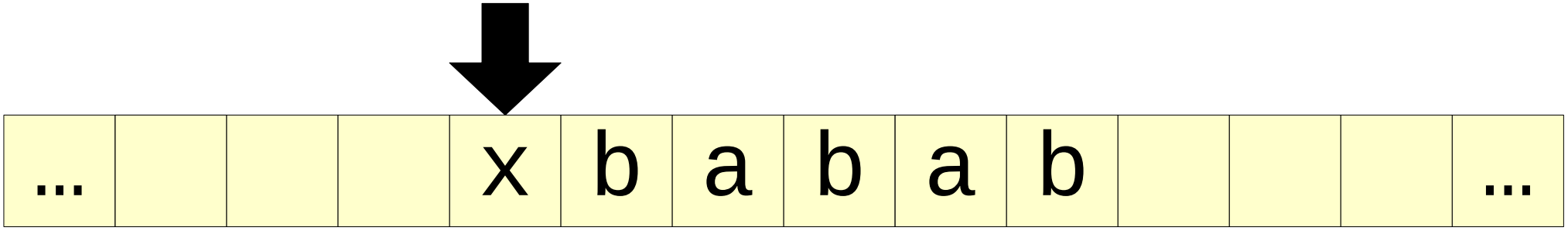
Turing Machines

- The command

  Move *direction*

  moves the tape head one step in the indicated direction (either `Left` or `Right`).

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



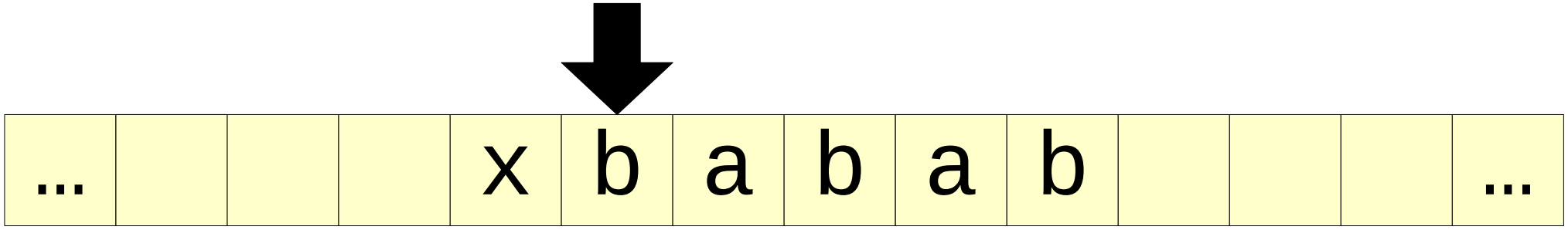
Turing Machines

- The command

  Move *direction*

  moves the tape head one step in the indicated direction (either `Left` or `Right`).

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



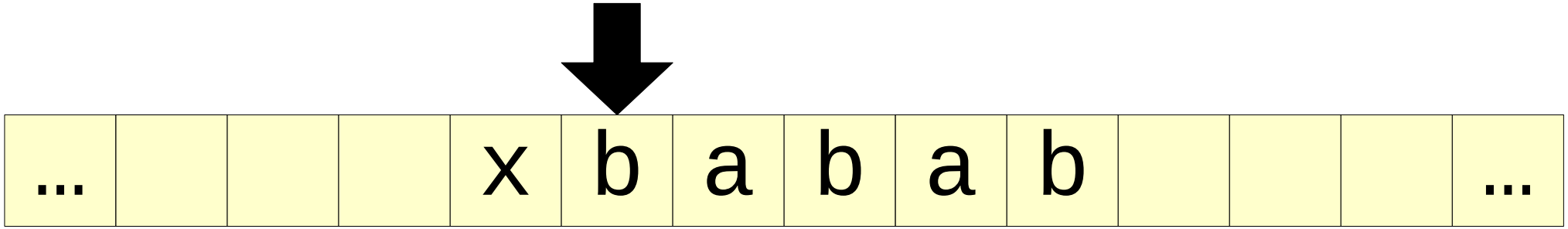
Turing Machines

- The command

  Move *direction*

  moves the tape head one step in the indicated direction (either `Left` or `Right`).

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



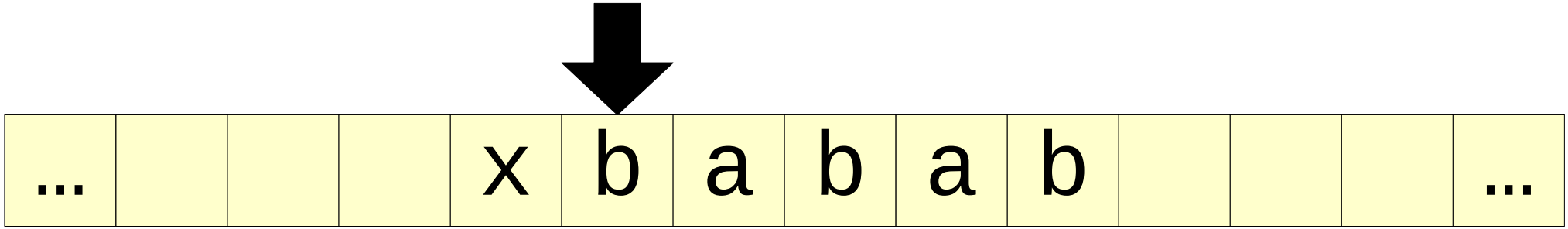
Turing Machines

- A statement of the form

  `If Not` *symbol command*

  sees if the cell under the tape head holds *symbol.*
- If so, nothing happens.
- If not, it executes *command*.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



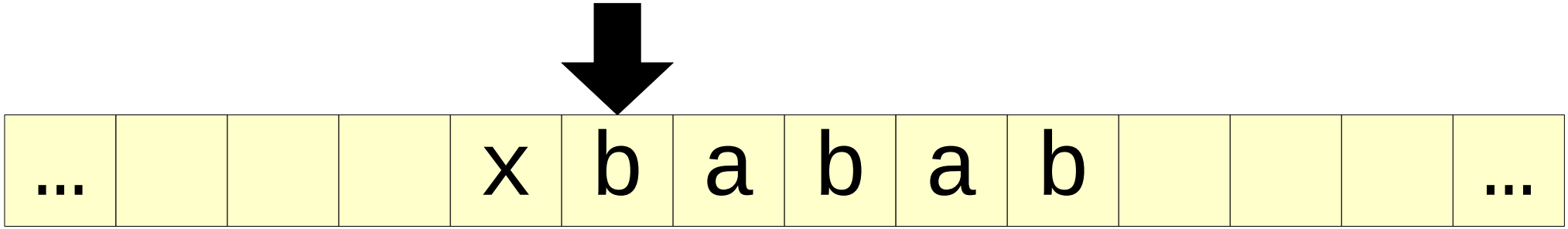
Turing Machines

- A statement of the form

  `If Not` *symbol command*

  sees if the cell under the tape head holds *symbol.*
- If so, nothing happens.
- If not, it executes *command*.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



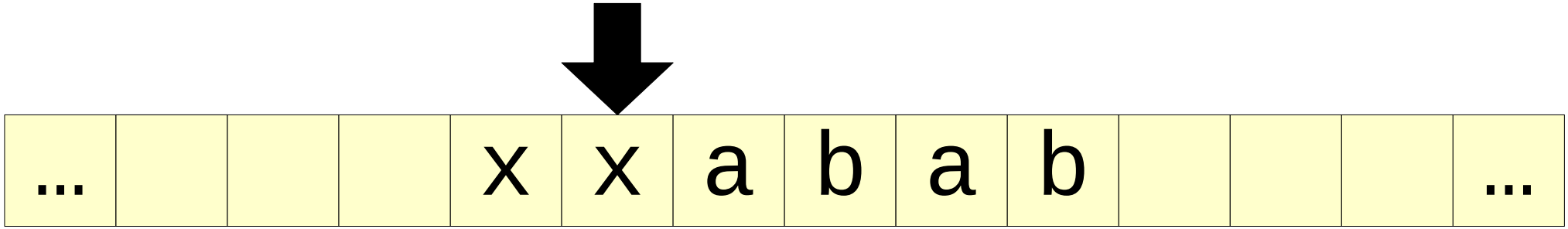
Turing Machines

- A statement of the form

  `If Not` *symbol command*

  sees if the cell under the tape head holds *symbol.*
- If so, nothing happens.
- If not, it executes *command*.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



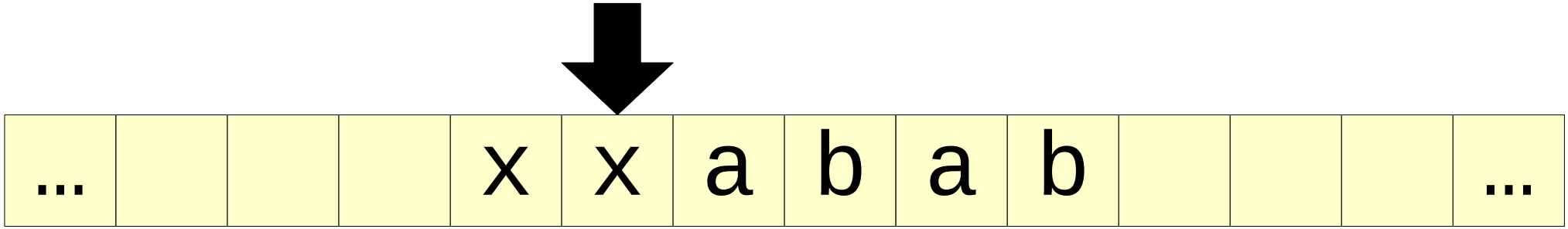
Turing Machines

- A statement of the form

  `If Not` *symbol command*

  sees if the cell under the tape head holds *symbol.*
- If so, nothing happens.
- If not, it executes *command*.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



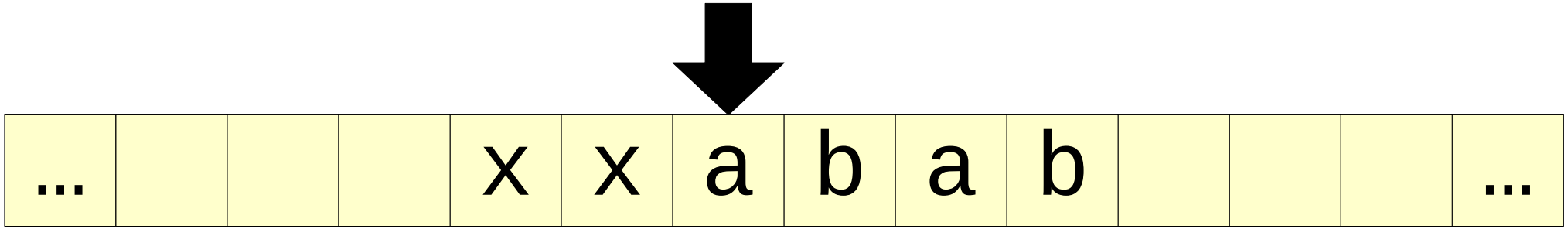
Turing Machines

- A statement of the form

  `If Not` *symbol command*

  sees if the cell under the tape head holds *symbol.*
- If so, nothing happens.
- If not, it executes *command*.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



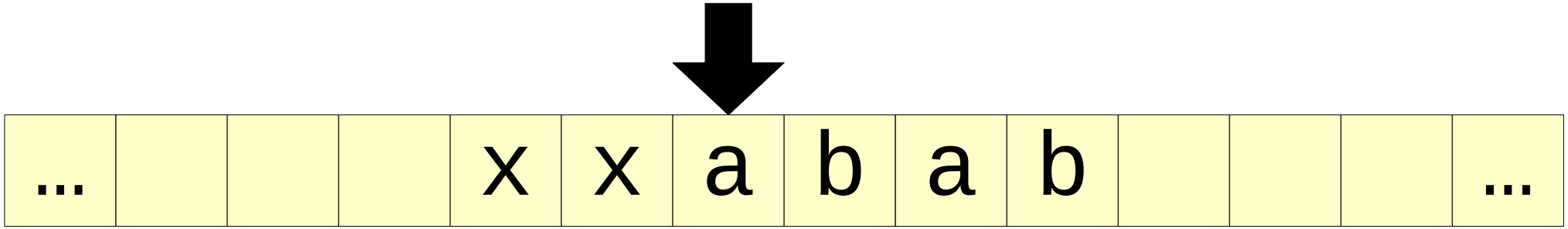
Turing Machines

- A statement of the form

  `If Not` *symbol command*

  sees if the cell under the tape head holds *symbol.*
- If so, nothing happens.
- If not, it executes *command*.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



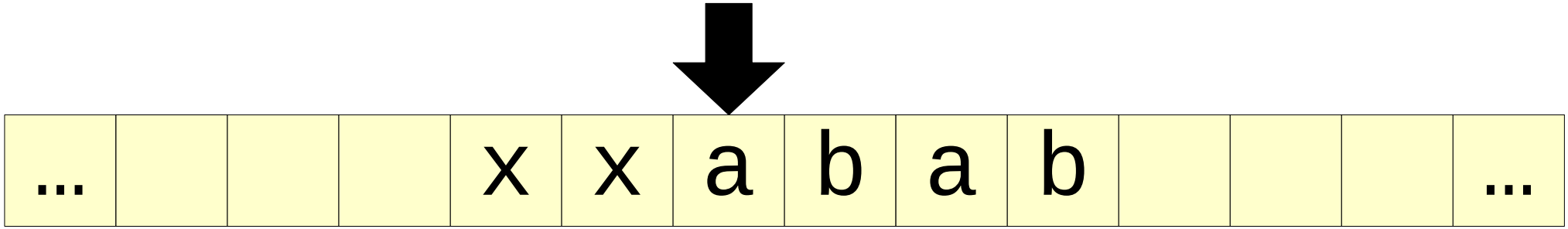
Turing Machines

- The command

  `Goto` *label*

  jumps to the indicated label.
- This program just has a `Start` label, but most interesting programs have other labels beyond this.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



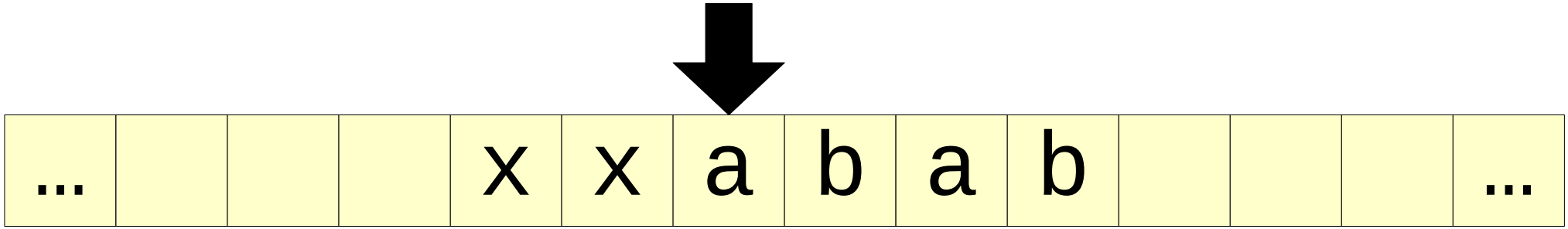
Turing Machines

- The command

  `Goto` *label*

  jumps to the indicated label.
- This program just has a `Start` label, but most interesting programs have other labels beyond this.

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



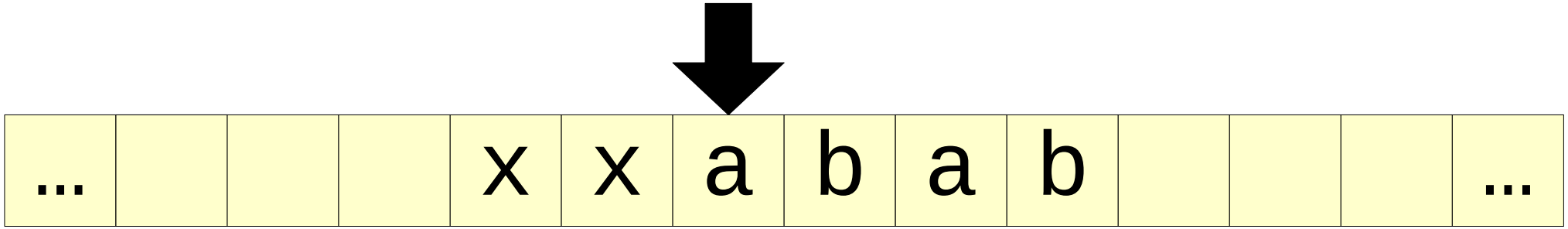
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



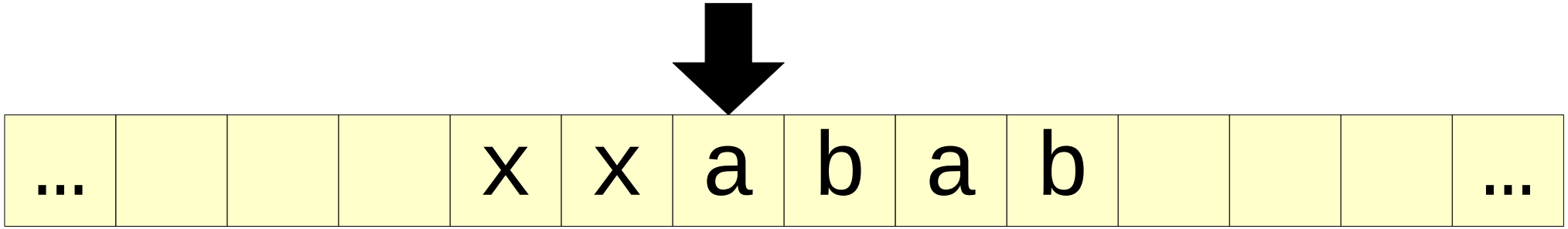
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



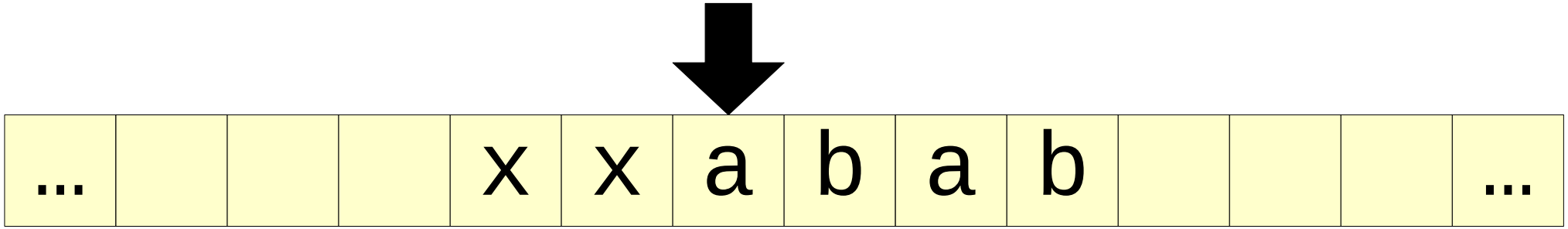
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



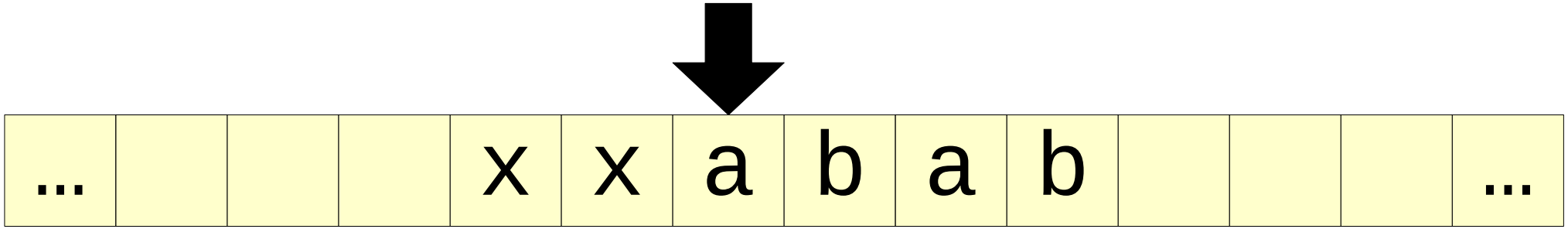
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



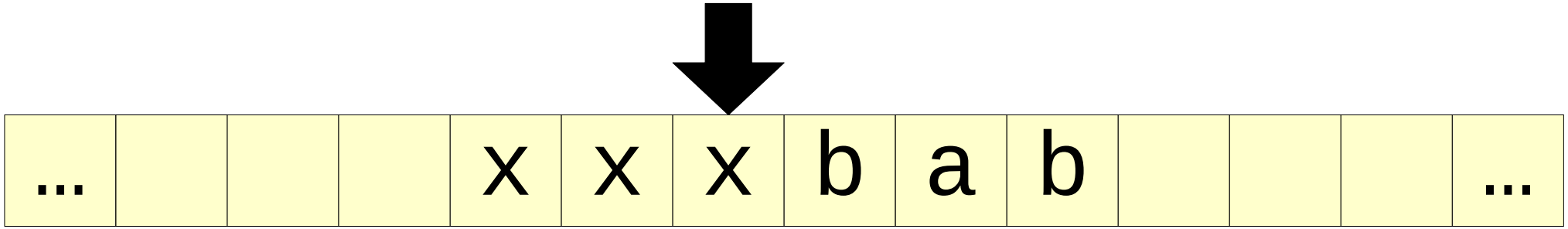
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



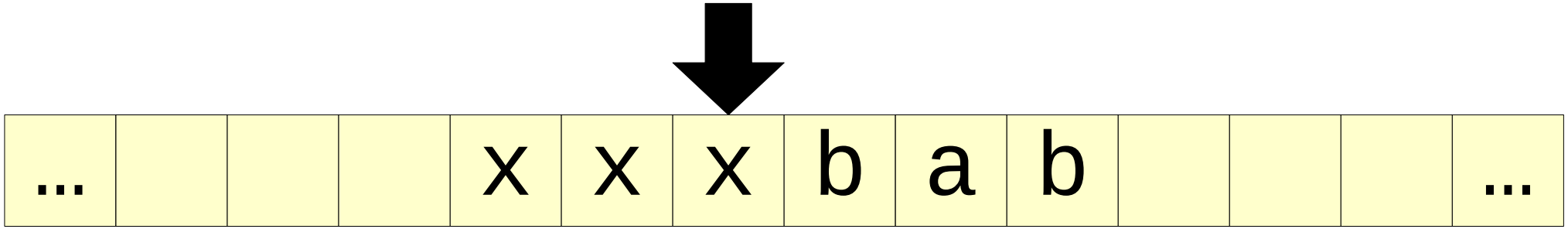
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



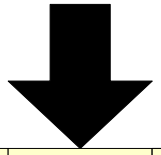
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



…				x	x	x	b	a	b				…

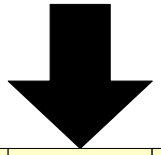
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



…				x	x	x	b	a	b				…

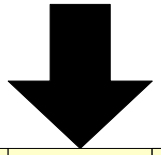
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



…				x	x	x	b	a	b				…

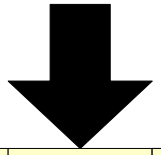
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



…				x	x	x	x	a	b				…

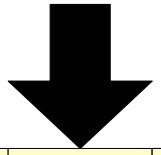
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



…				x	x	x	x	a	b				…

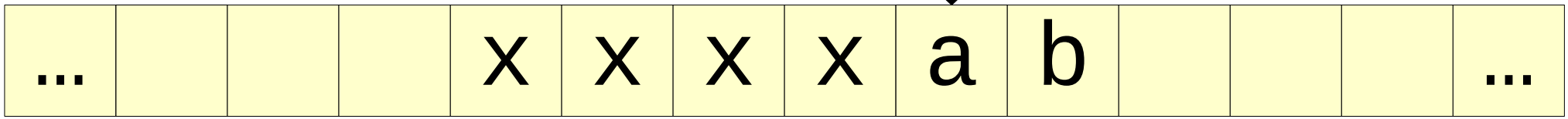
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



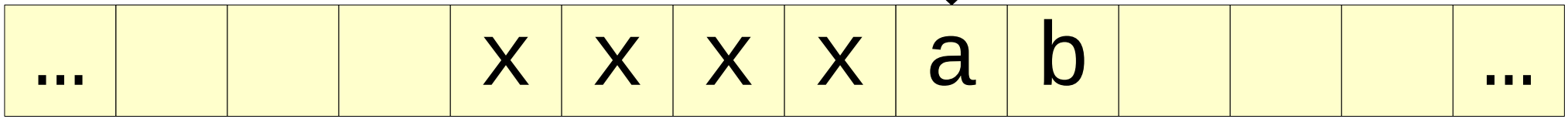
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



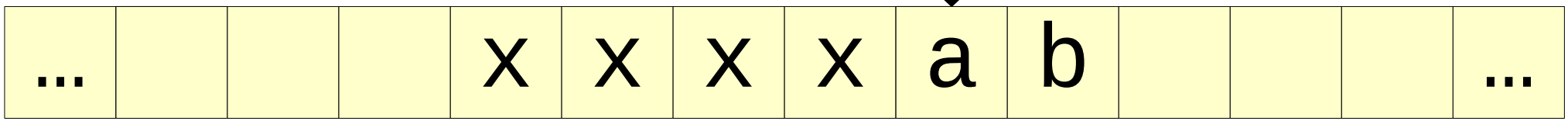
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



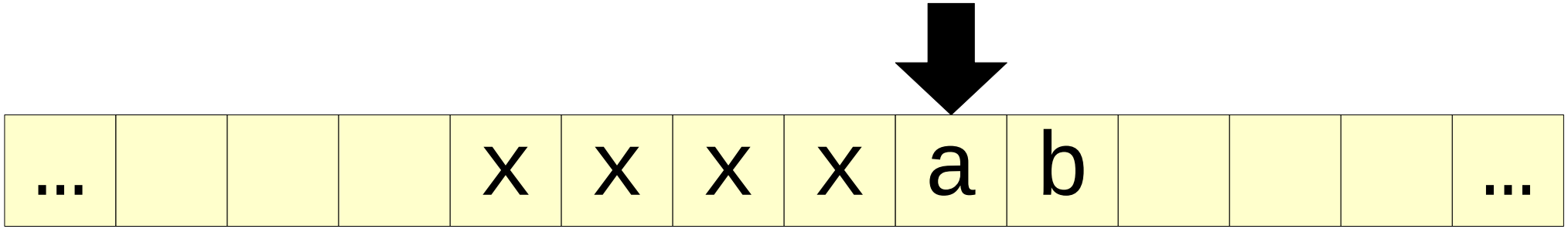
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



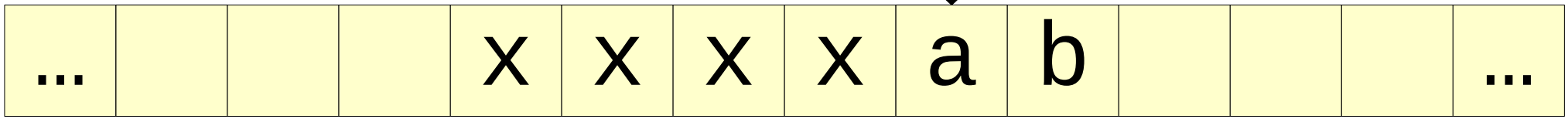
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



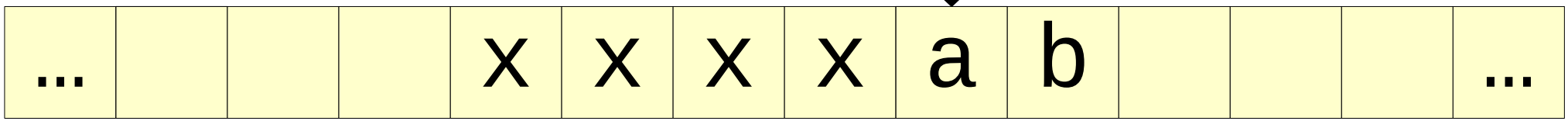
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



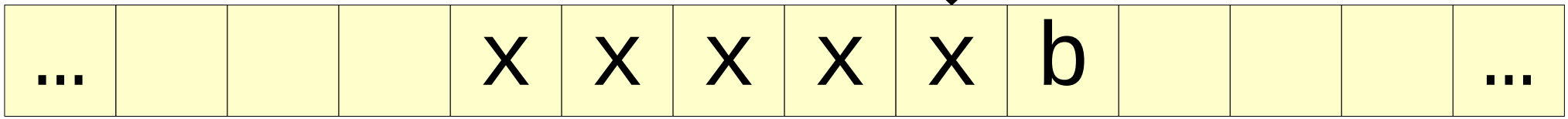
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



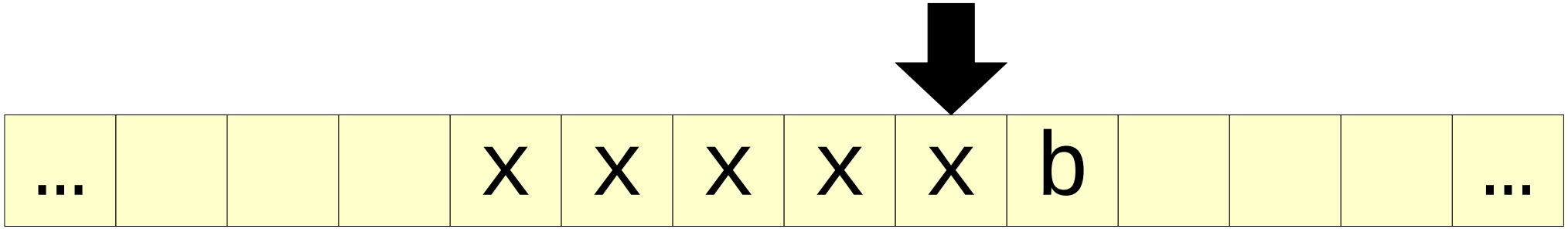
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



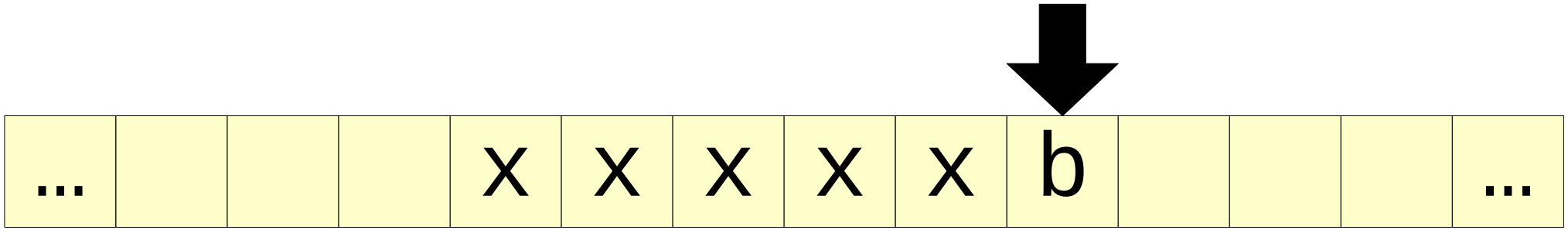
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



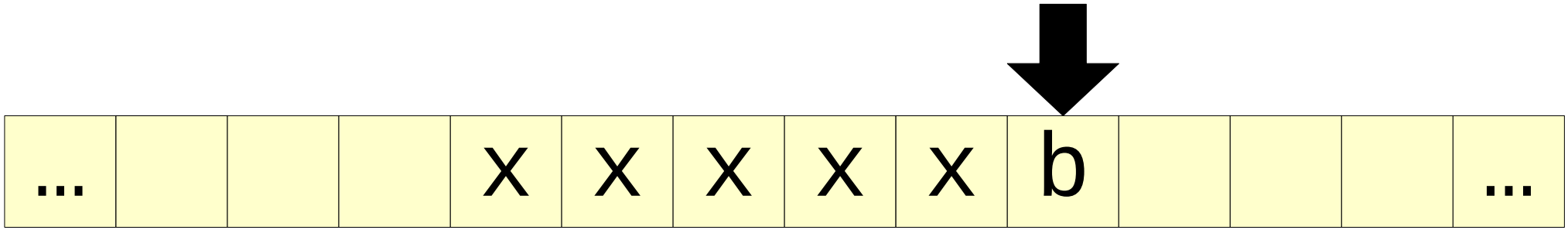
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



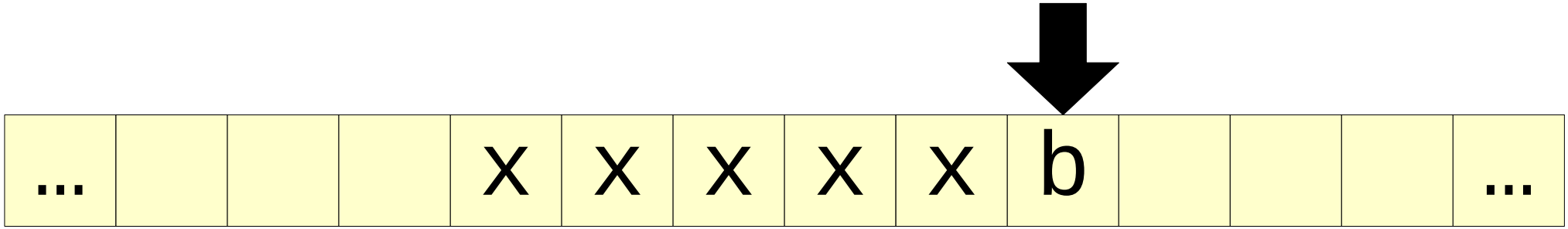
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



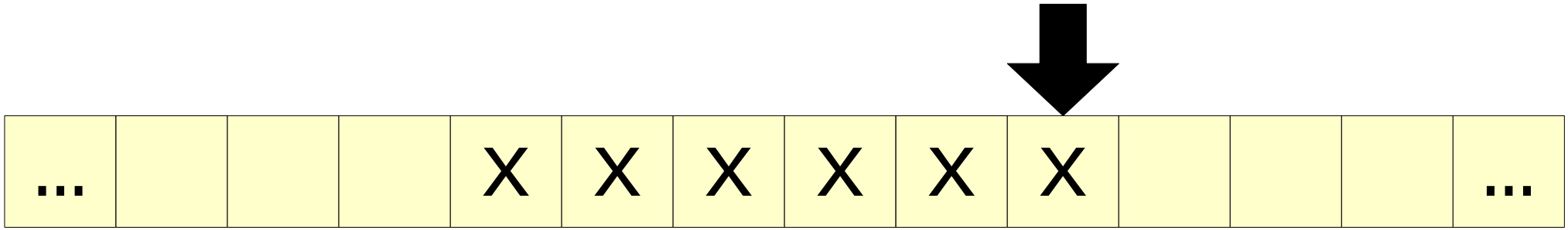
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
  If Blank Return True
  If 'b' Return False
  Write 'x'
  Move Right
  If Not 'b' Return Fals
  Write 'x'
  Move Right
  Goto Start
```



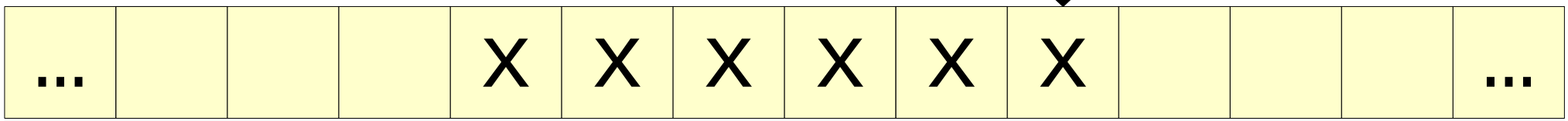
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
  If Blank Return True
  If 'b' Return False
  Write 'x'
  Move Right
  If Not 'b' Return Fals
  Write 'x'
  Move Right
  Goto Start
```



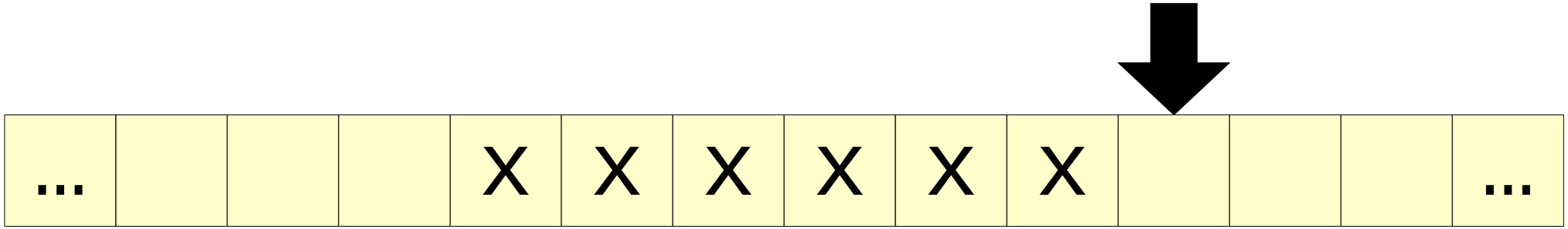
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



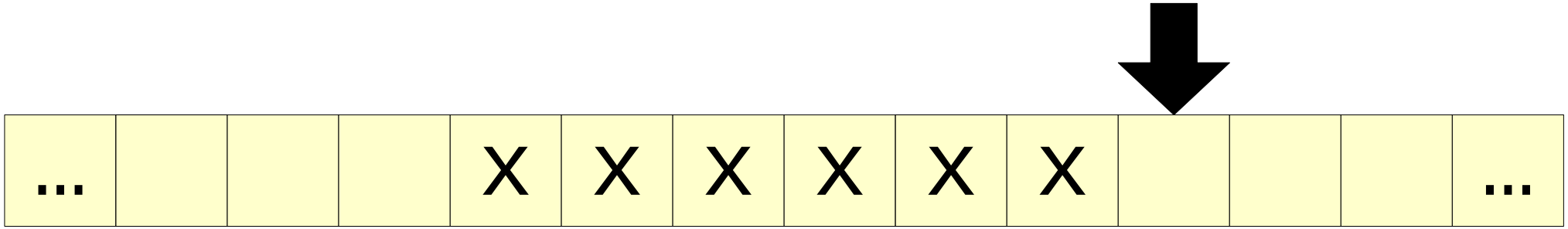
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



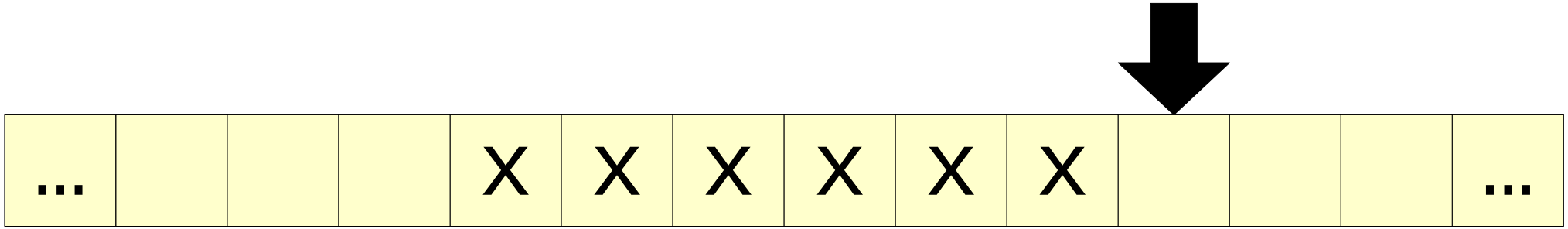
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we "fall off" the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



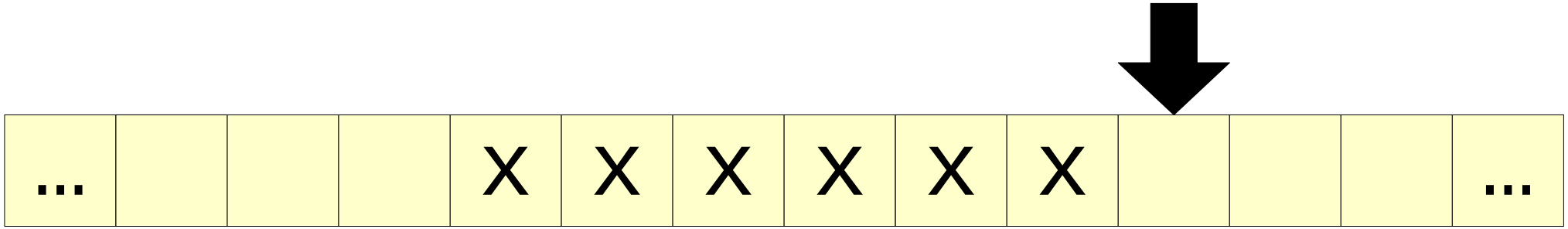
Turing Machines

- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



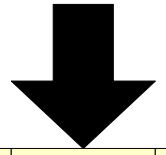
Turing Machines

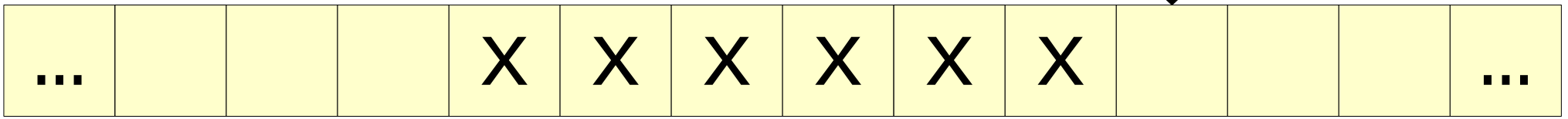
- A TM stops when executing the

  `Return` *result*

  command.
- Here, *result* can be either `True` or `False`.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the `Return False` command.)

```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```

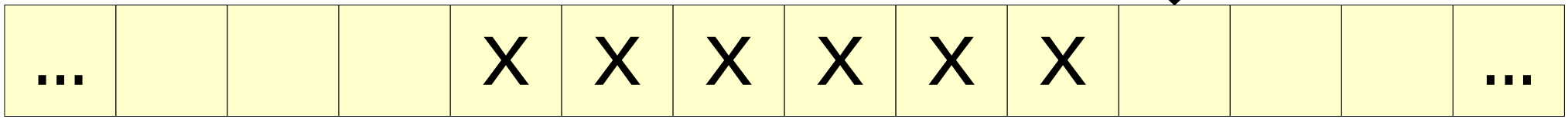




Turing Machines

- This TM initially started up with the string ababab on its tape, so this means that TM returns true on the input ababab, not xxxxxx.
- An intuition for this: we gave this program an input. It therefore returned true with respect to that input, not whatever internal data it generated in making its decision.

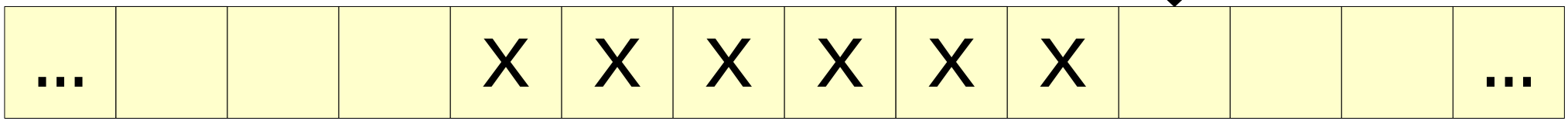
```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return Fals
   Write 'x'
   Move Right
   Goto Start
```



Turing Machines

- To summarize, we only have six commands:
  - `Move` *direction*
  - `Write` *symbol*
  - `Goto` *label*
  - `Return` *result*
  - `If` *symbol command*
  - `If Not` *symbol command*
- Despite their simplicitly, TMs are *surprisingly* powerful. The rest of this lecture explores why.

```
Start:
  If Blank Return True
  If 'b' Return False
  Write 'x'
  Move Right
  If Not 'b' Return Fals
  Write 'x'
  Move Right
  Goto Start
```



Programming Turing Machines

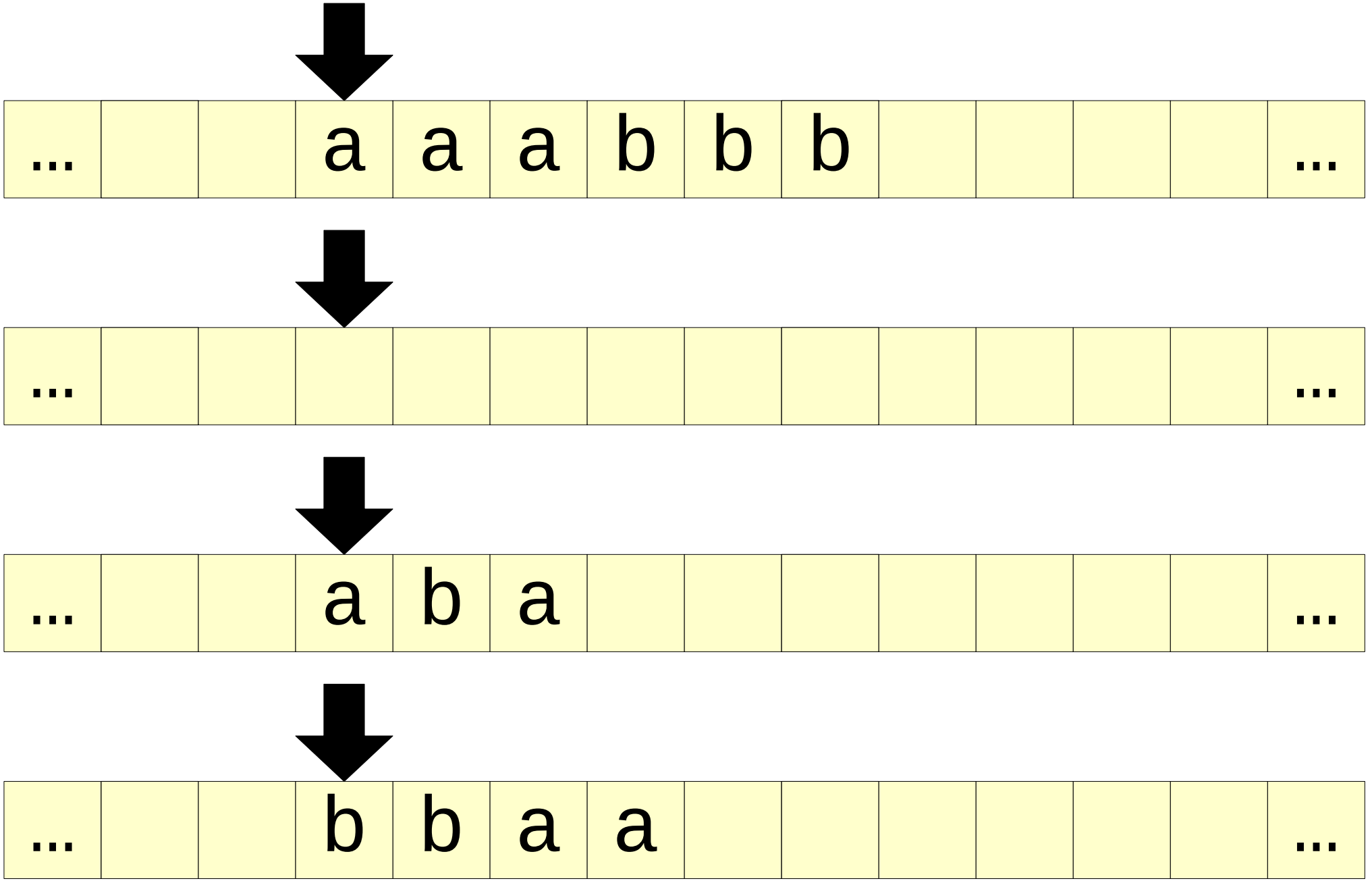
Our First Challenge

- The language

$$\{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$$

  is a canonical example of a nonregular language. It's not possible to check if a string is in this language given only finite memory.
- Turing machines, however, are powerful enough to do this. Let's see how.

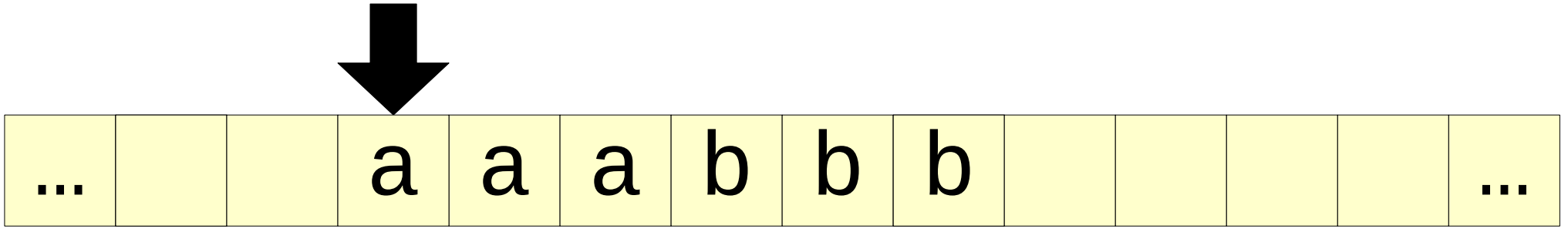
$$L = \{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$$



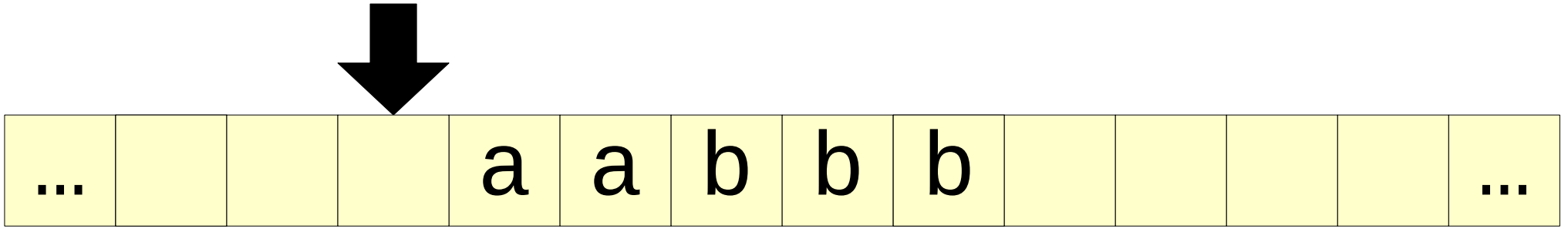
A Recursive Approach

- We can process our string using this recursive approach:
  - The string ε is in L.
  - The string **a***w***b** is in L if and only if w is in L.
  - Any string starting with **b** is not in L.
  - Any string ending with **a** is not in L.
- All that's left to do now is write a TM that implements this.

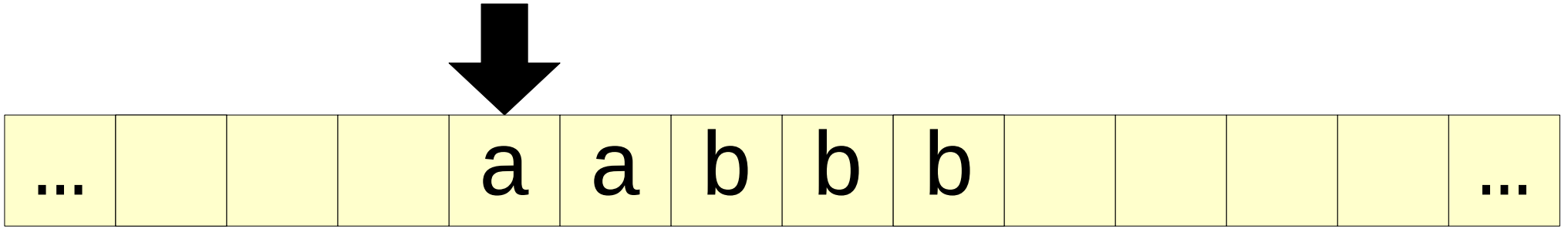
A Sketch of our TM



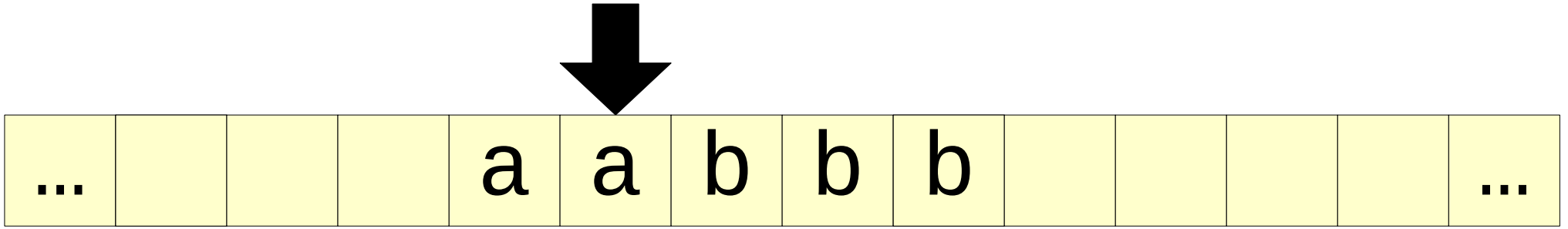
A Sketch of our TM



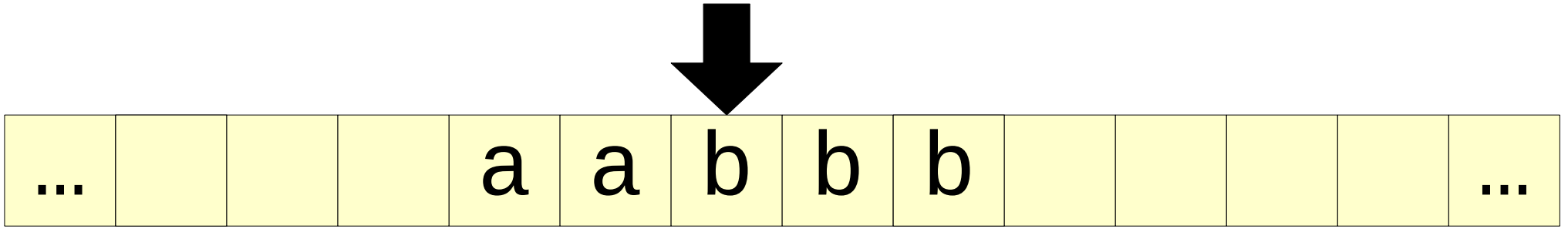
A Sketch of our TM



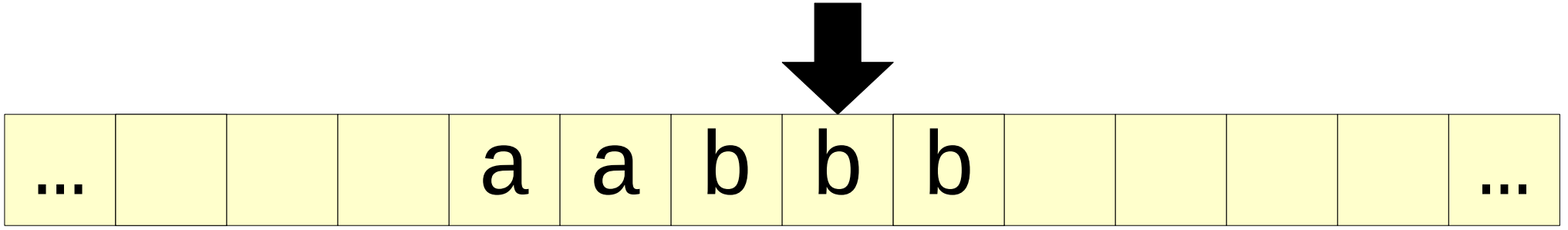
A Sketch of our TM



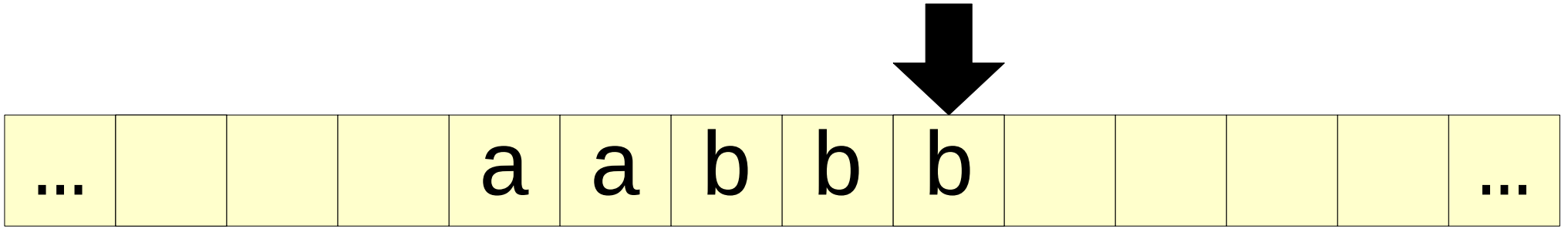
A Sketch of our TM



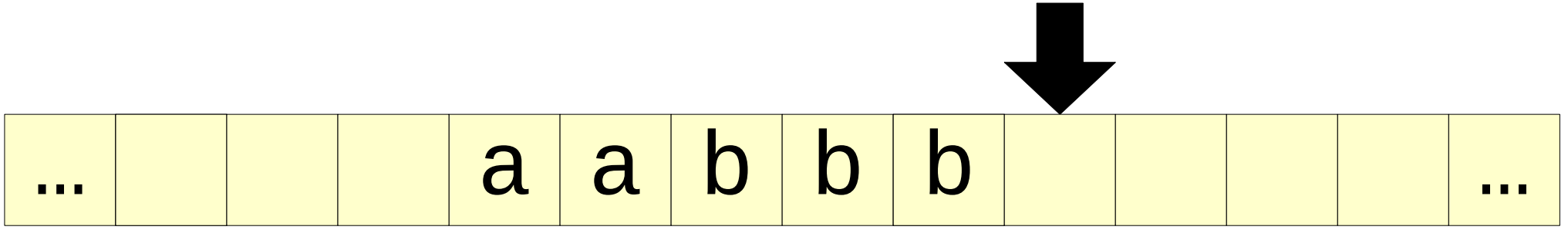
A Sketch of our TM



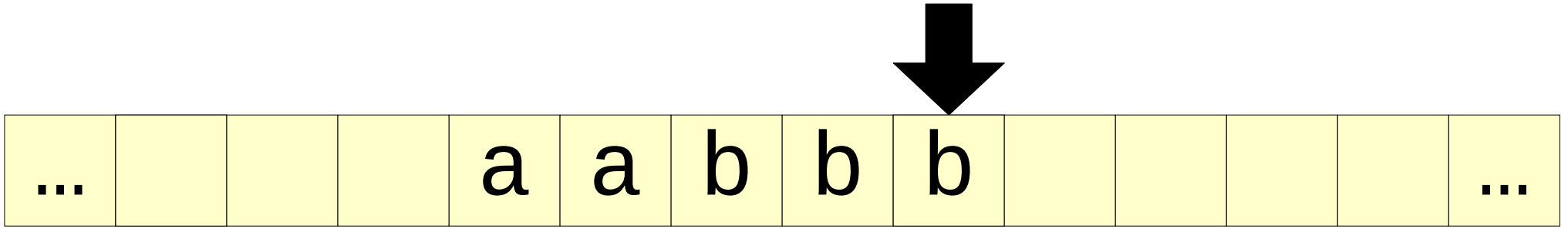
A Sketch of our TM



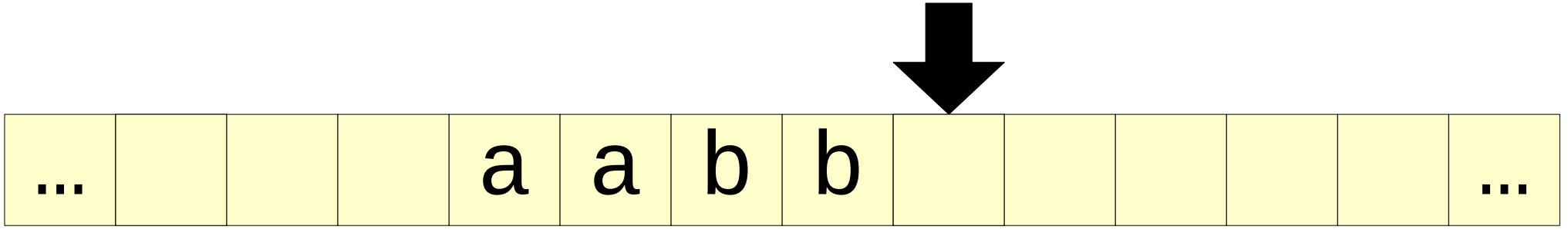
A Sketch of our TM



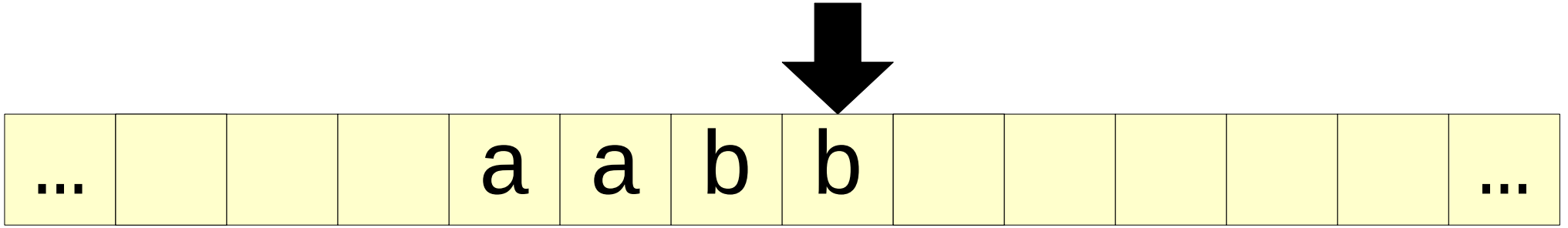
A Sketch of our TM



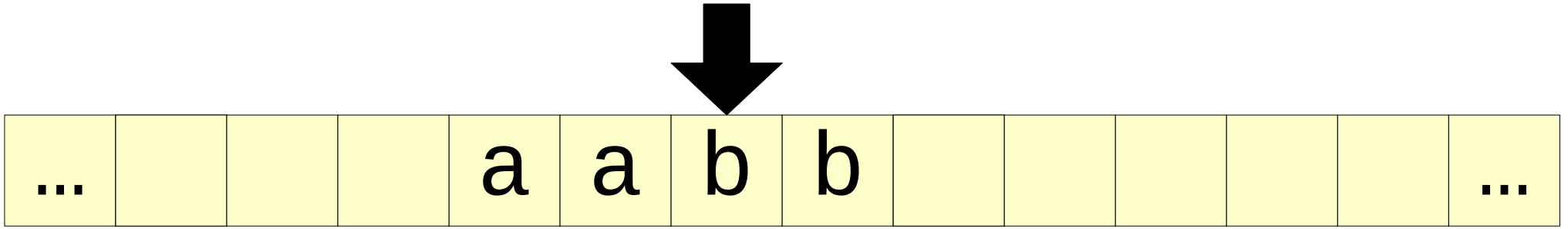
A Sketch of our TM



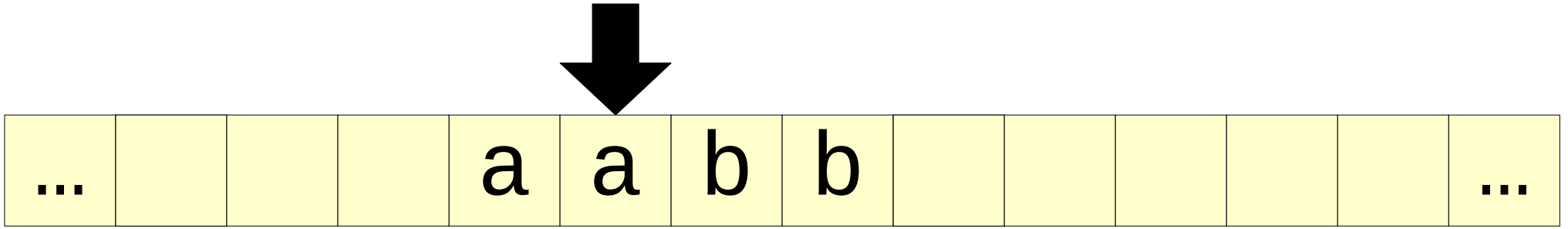
A Sketch of our TM



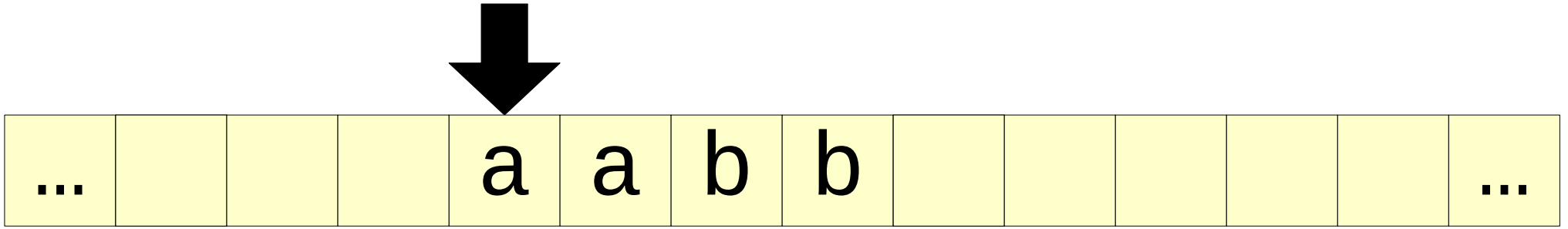
A Sketch of our TM



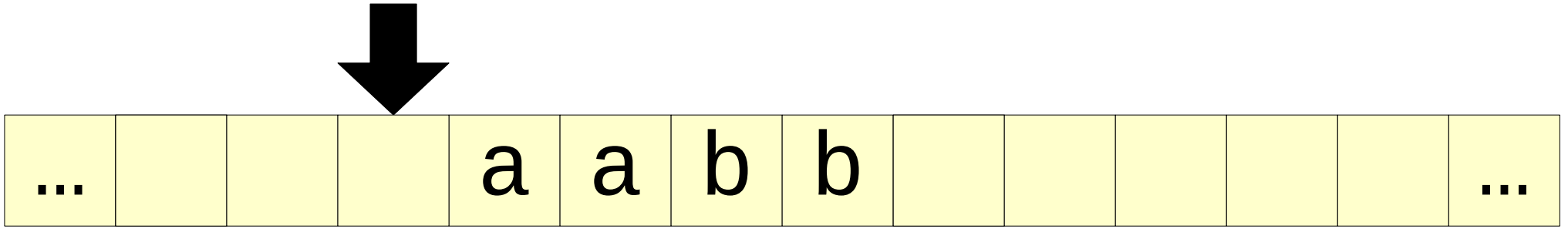
A Sketch of our TM



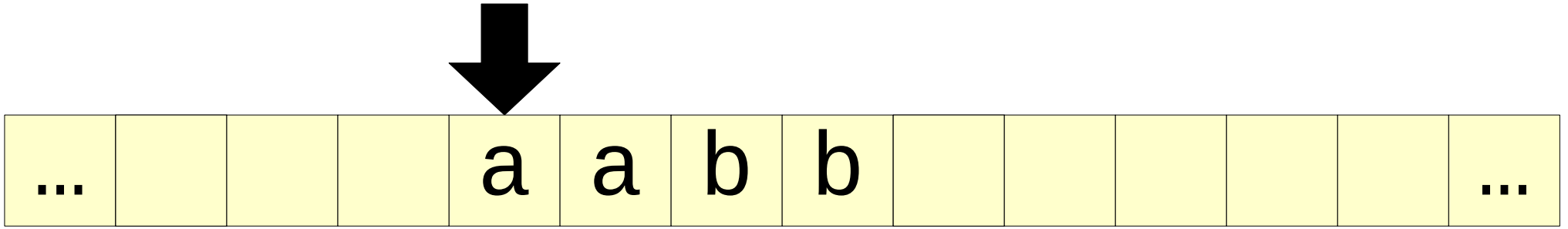
A Sketch of our TM



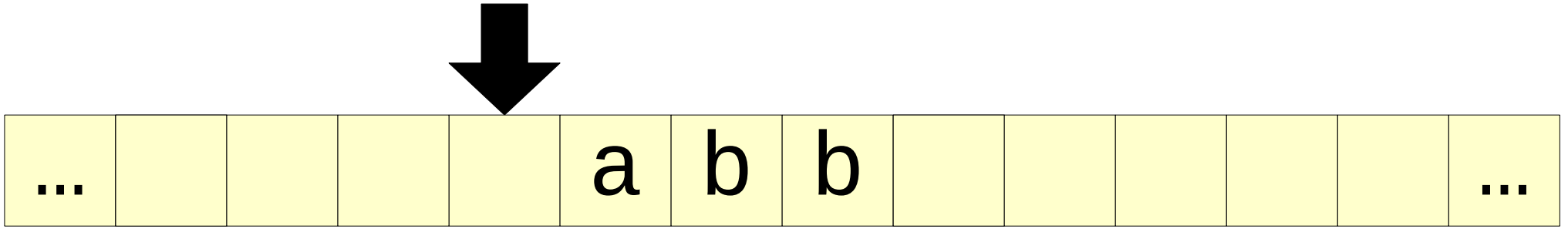
A Sketch of our TM



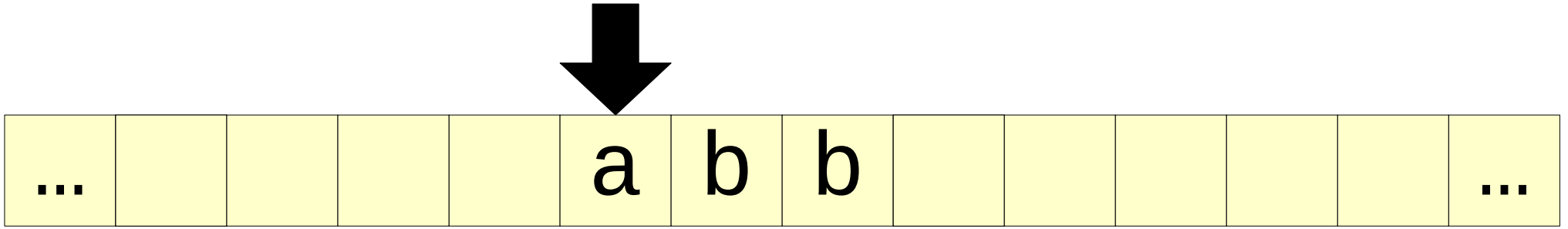
A Sketch of our TM



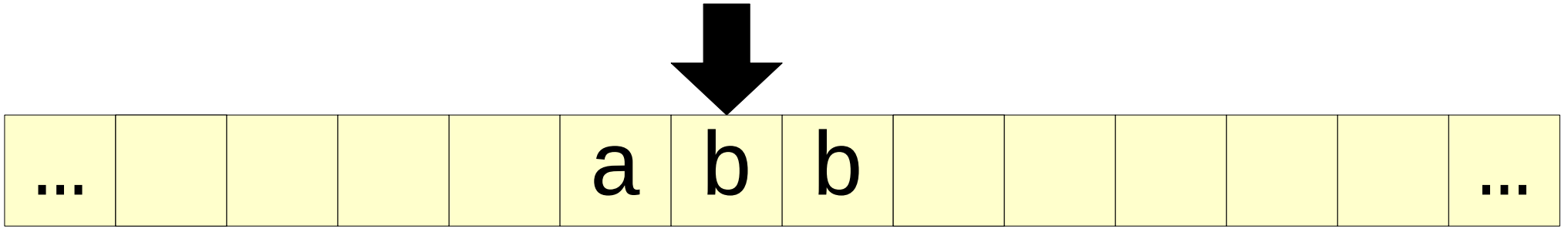
A Sketch of our TM



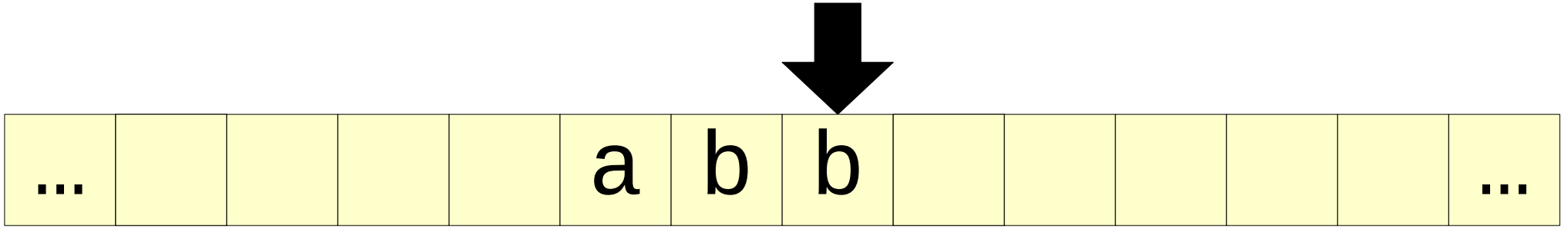
A Sketch of our TM



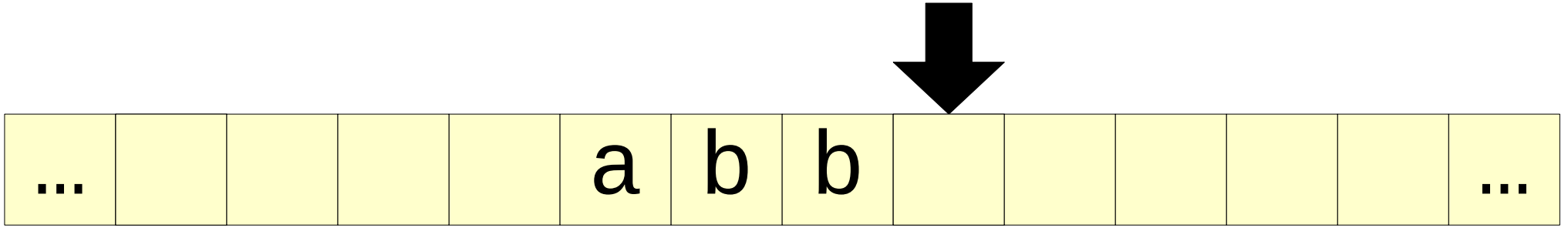
A Sketch of our TM



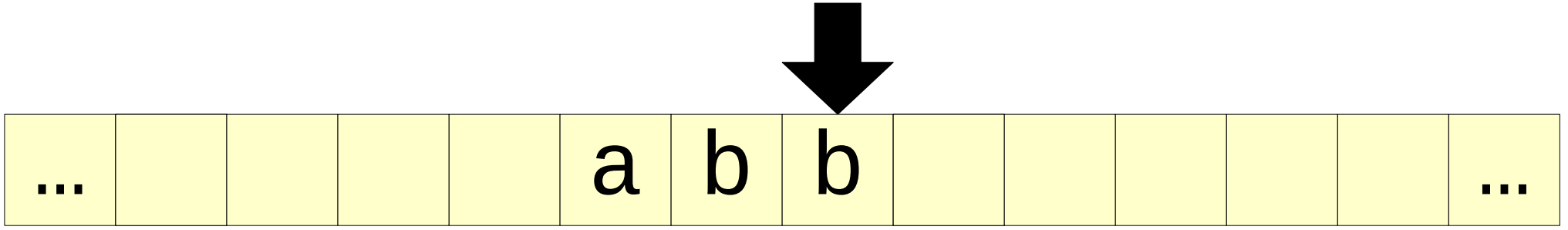
A Sketch of our TM



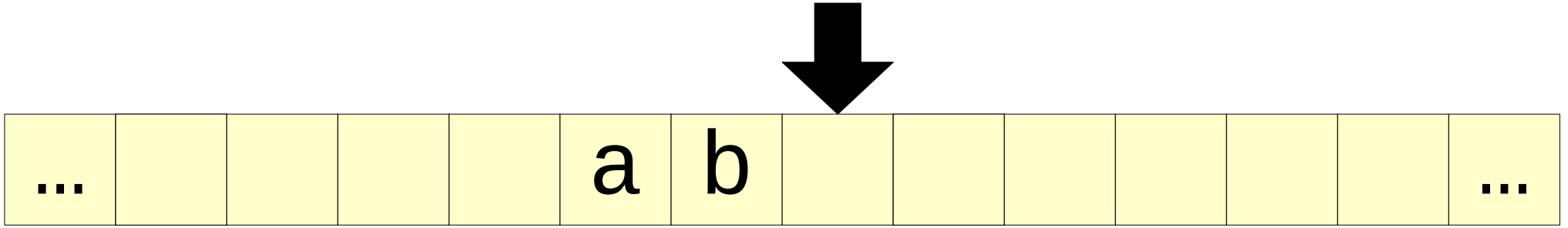
A Sketch of our TM



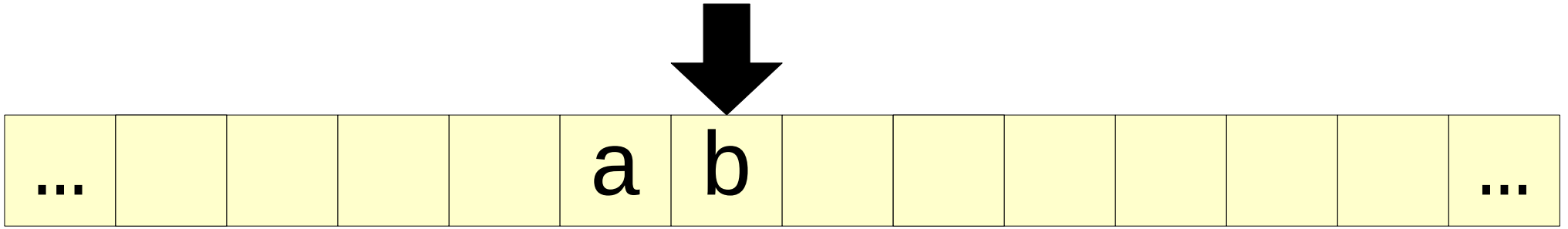
A Sketch of our TM



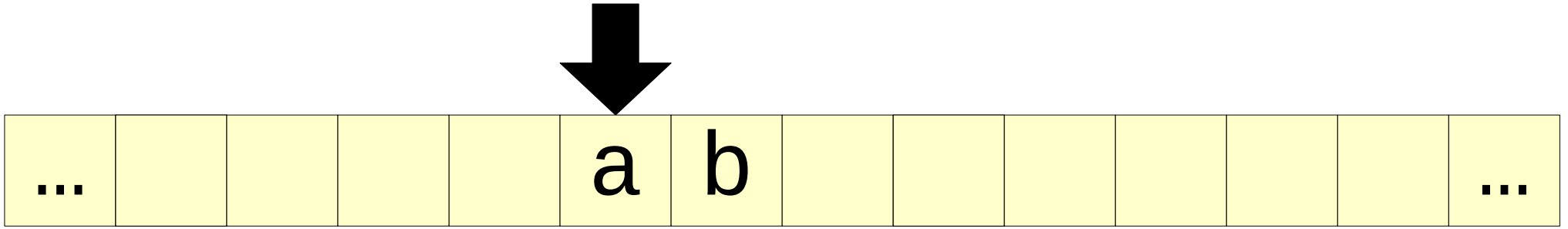
A Sketch of our TM



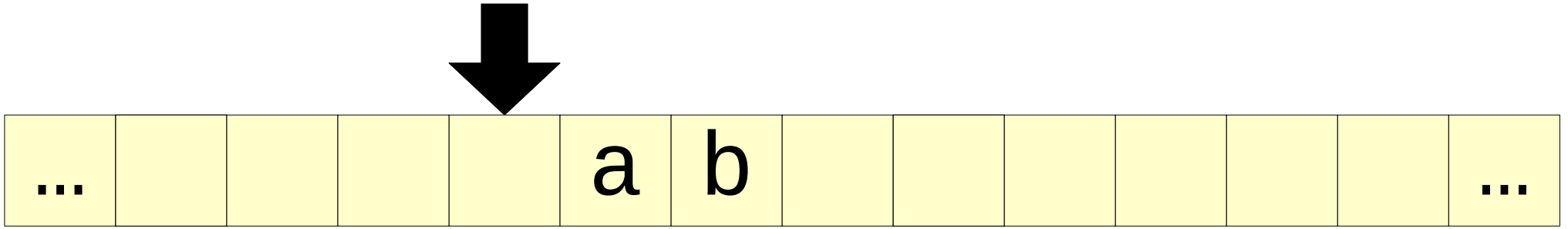
A Sketch of our TM



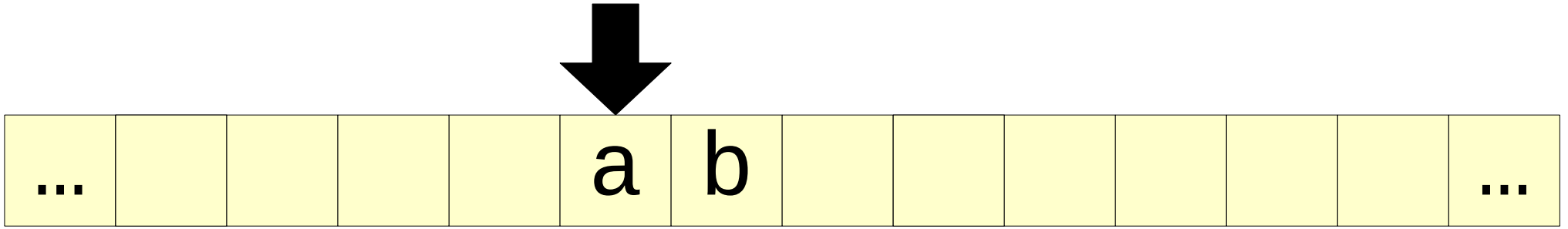
A Sketch of our TM



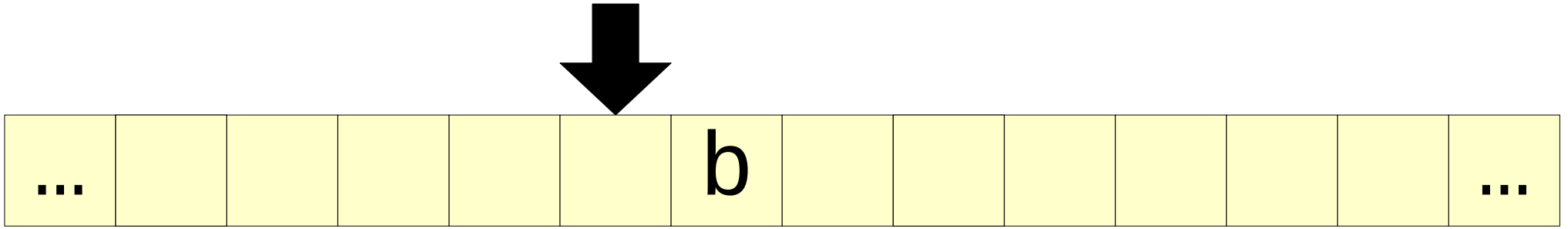
A Sketch of our TM



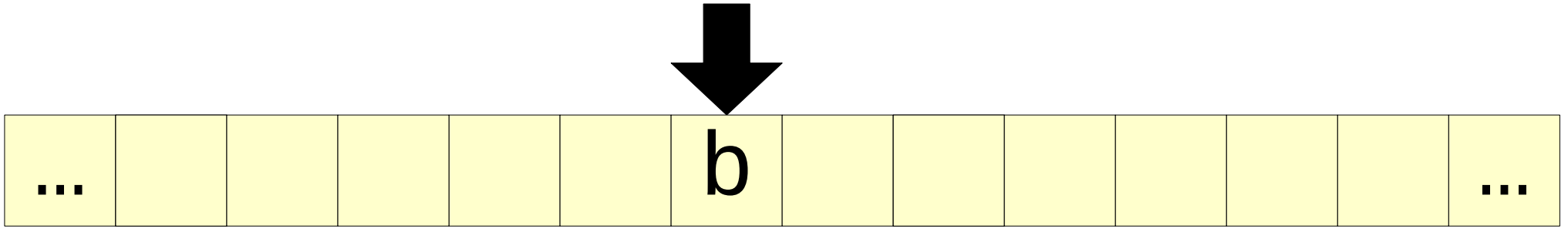
A Sketch of our TM



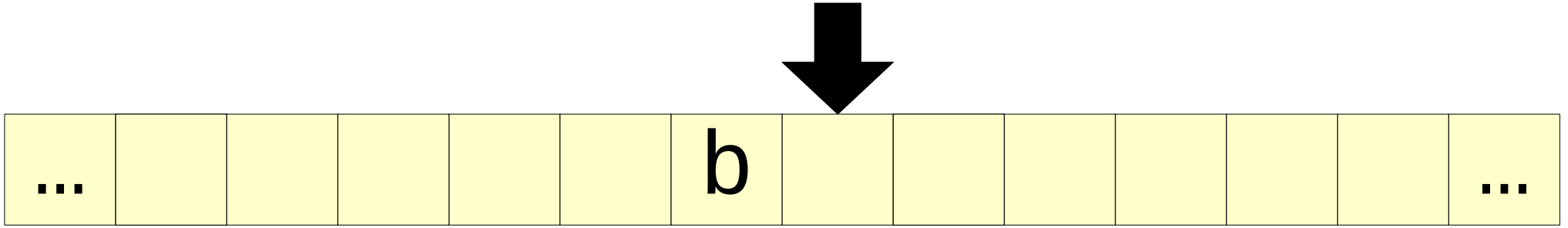
A Sketch of our TM



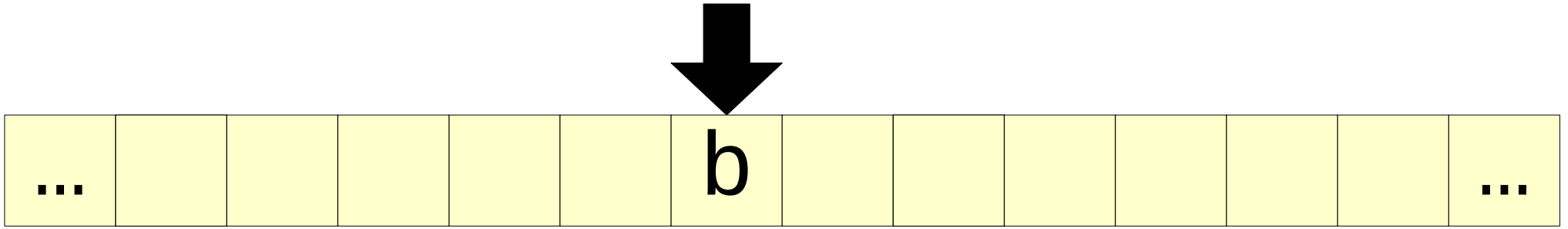
A Sketch of our TM



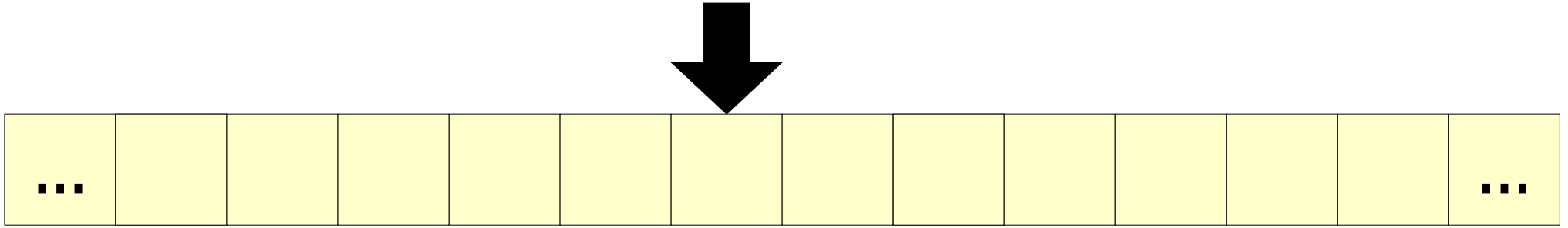
A Sketch of our TM

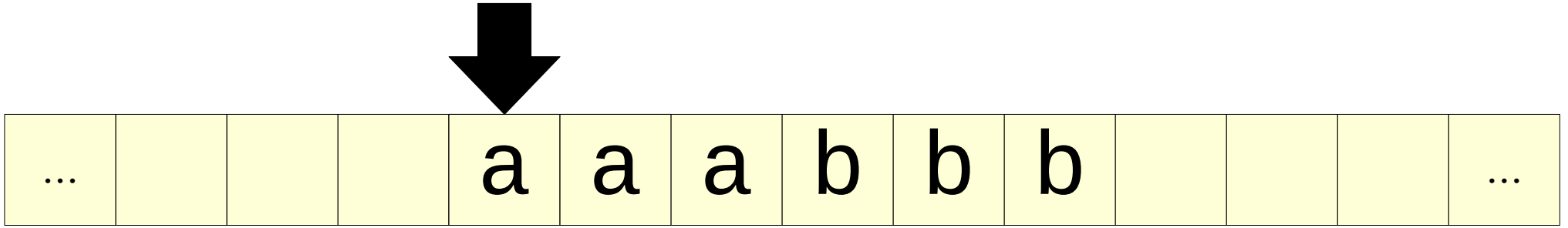


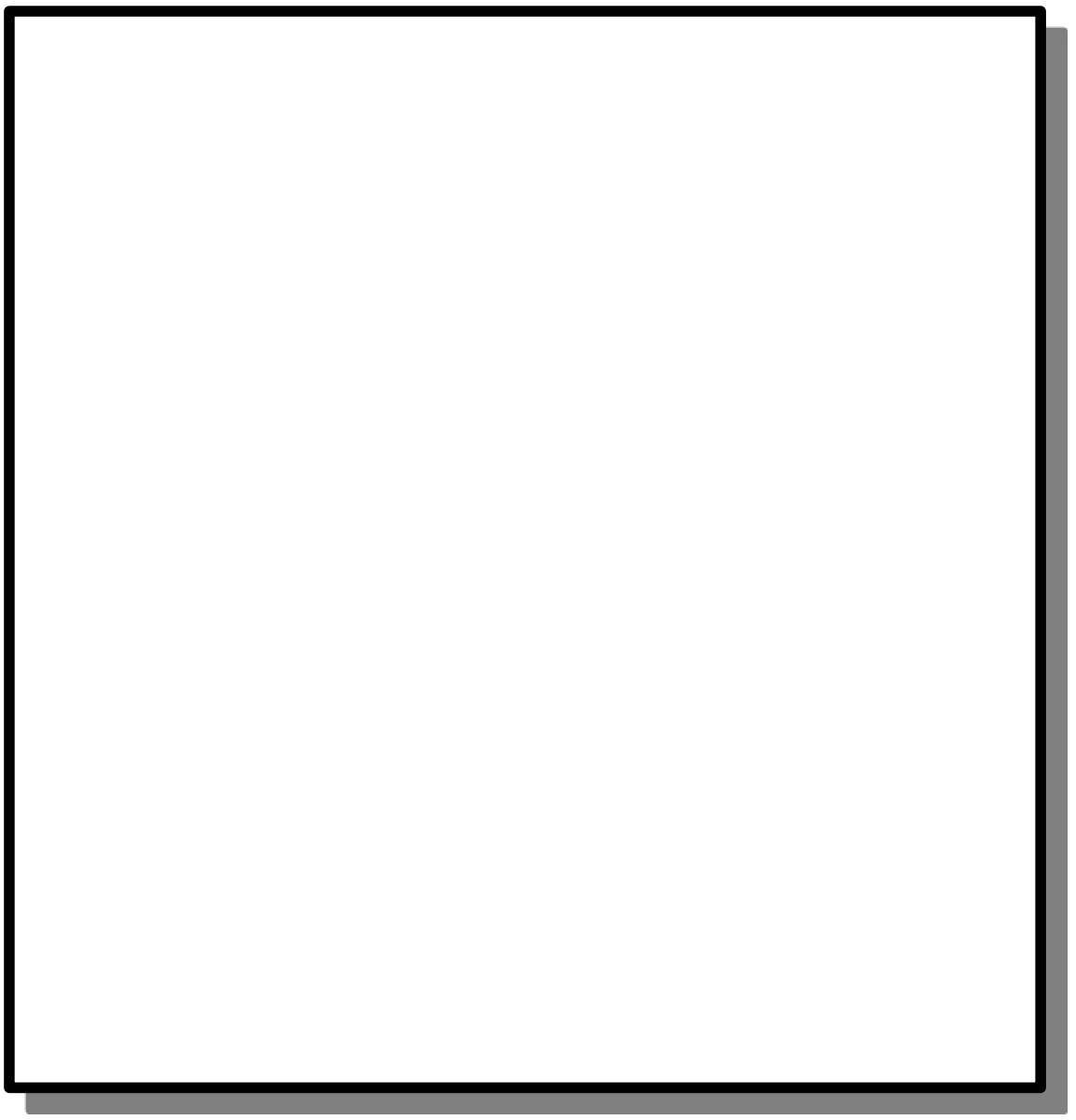
A Sketch of our TM



A Sketch of our TM

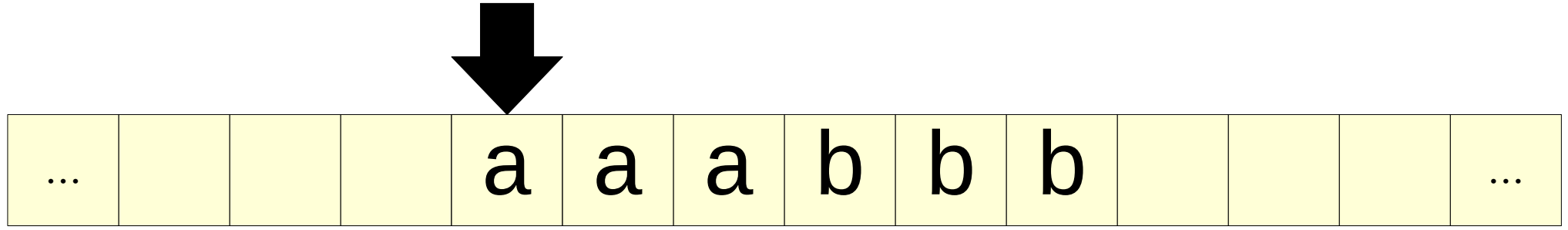



...
a a a b b b
...

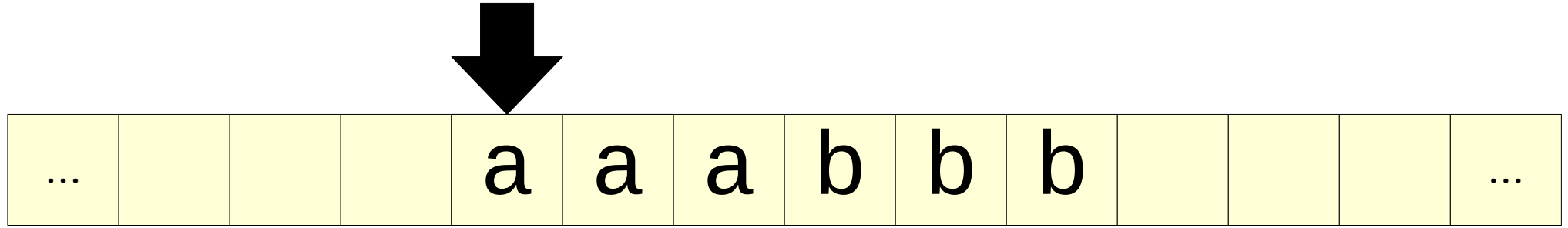


…				a	a	a	b	b	b				…

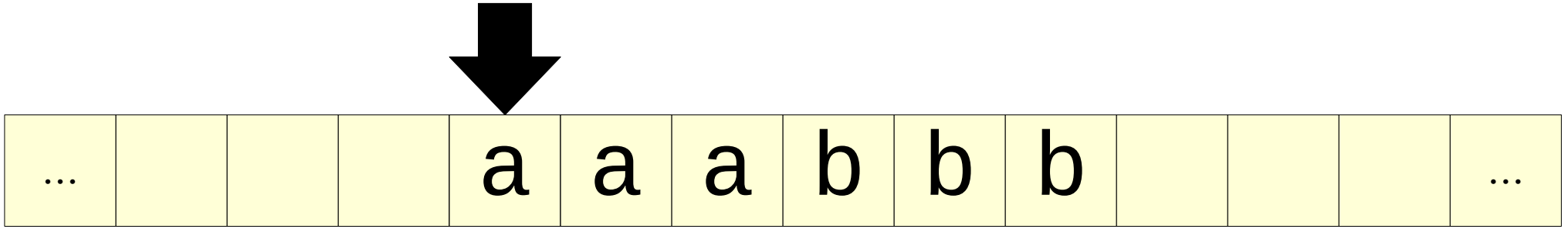
Start:



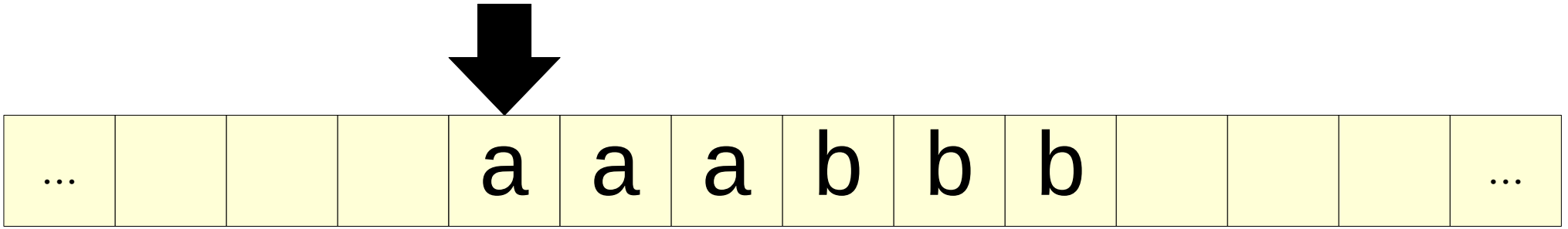
```
Start:
```



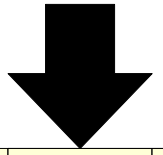
Start:



```
Start:
    If Blank Return True
```

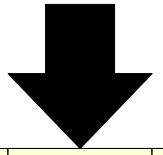


```
Start:
   If Blank Return True
```



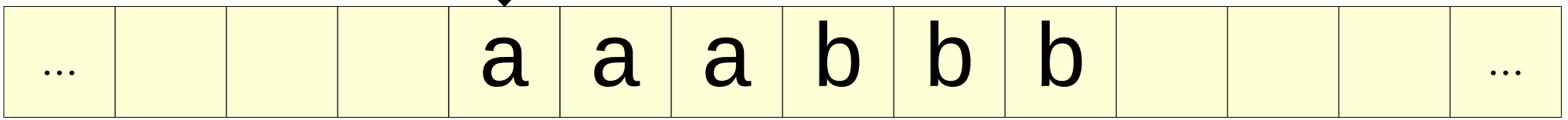
…				a	a	a	b	b	b				…

```
Start:
   If Blank Return True
   If 'b' Return False
```

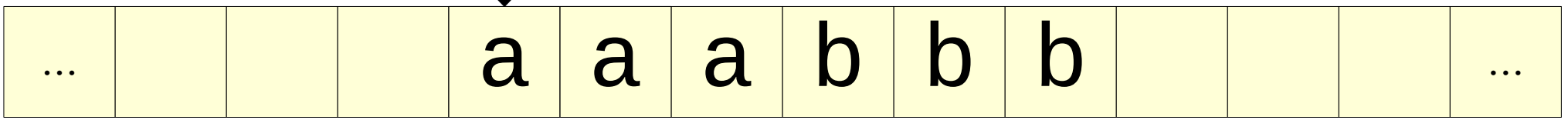


…				a	a	a	b	b	b				…

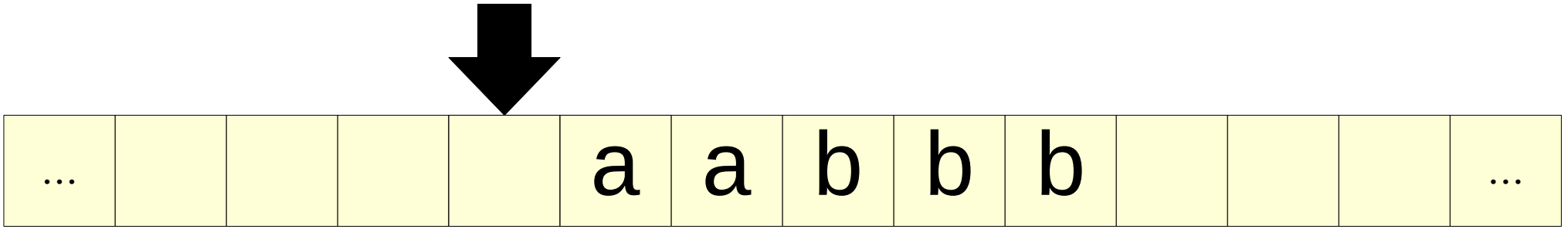
```
Start:
   If Blank Return True
   If 'b' Return False
```



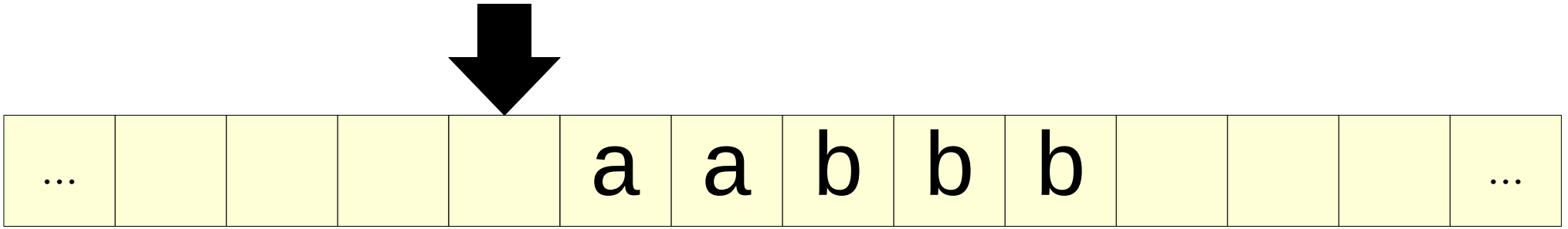
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank
```



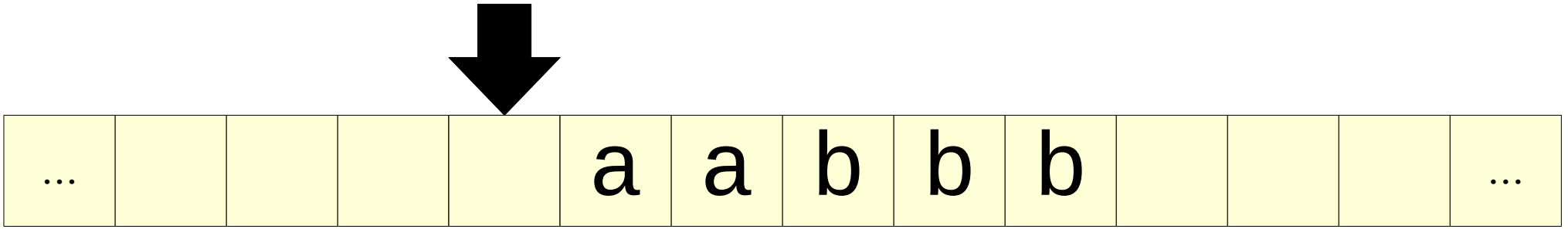
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank
```



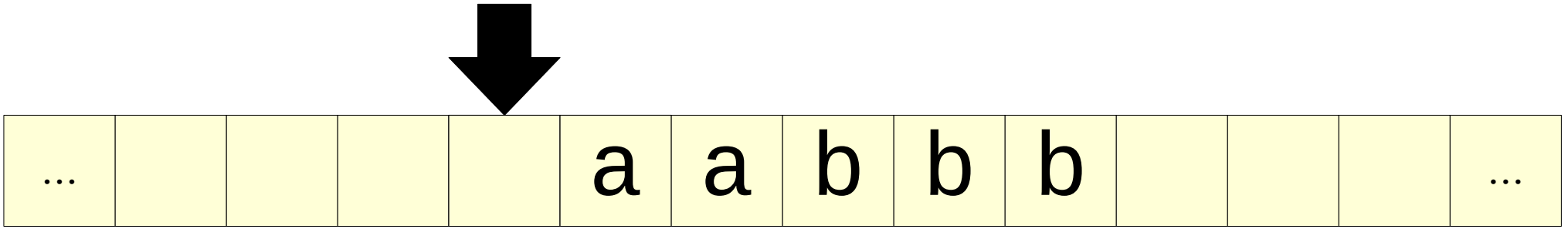
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank
```



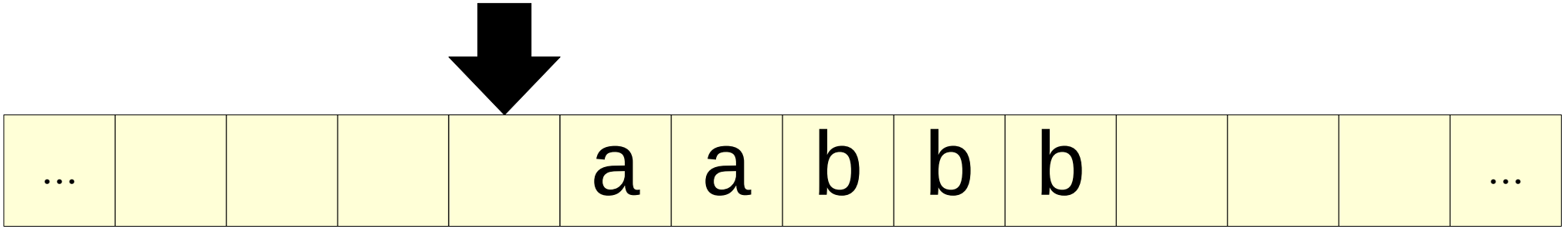
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
```



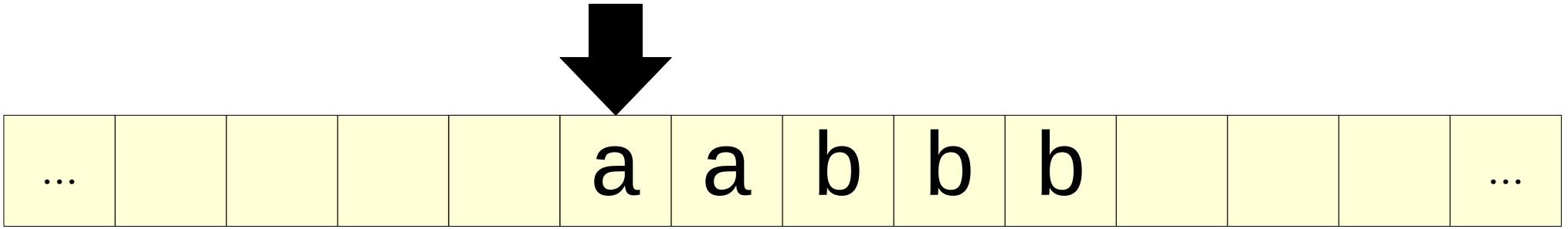
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
```



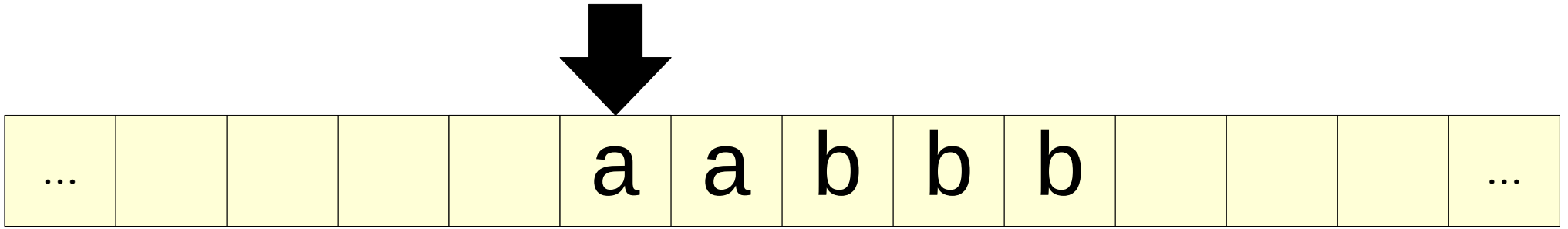
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
```



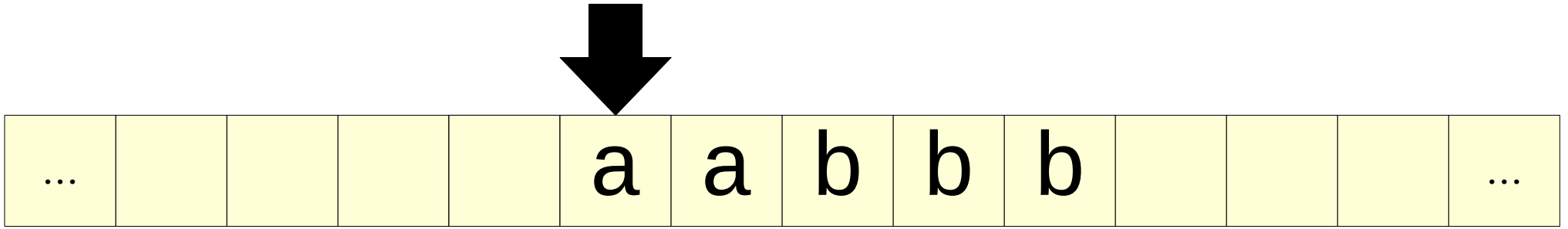
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
```



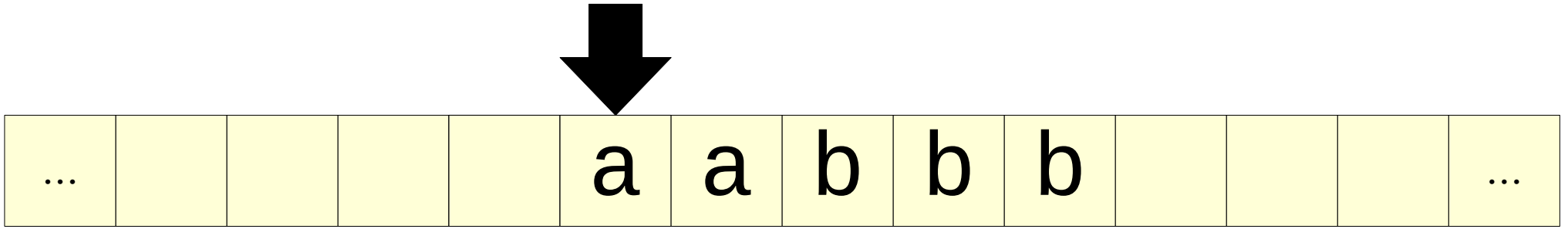
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
```



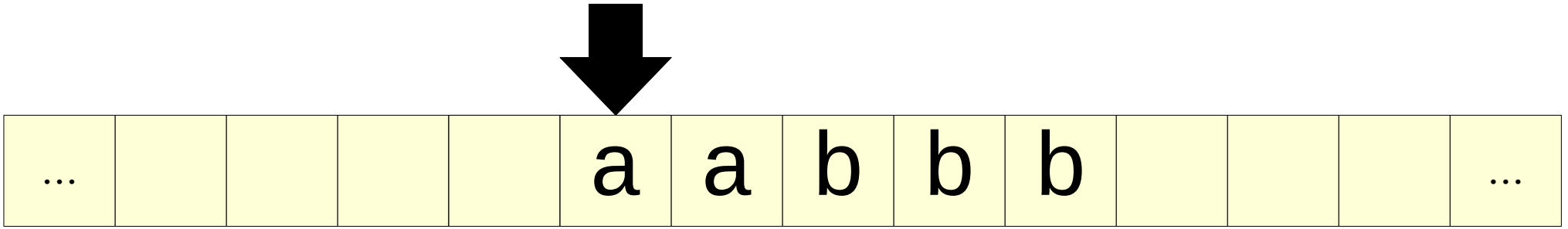
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



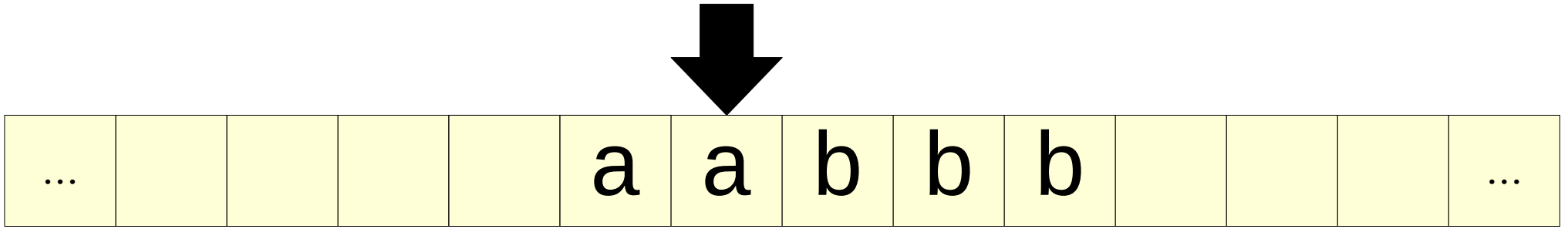
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



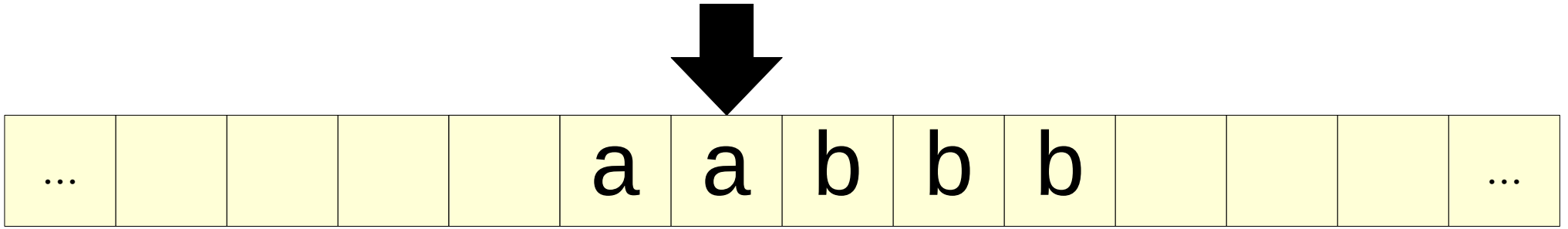
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



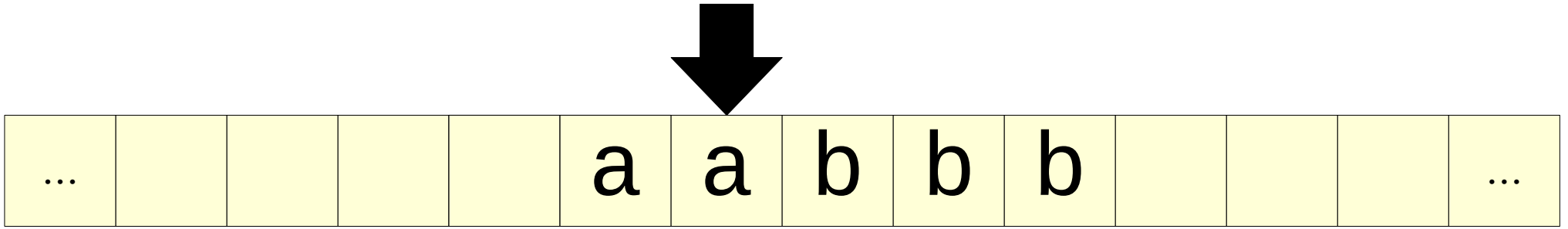
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



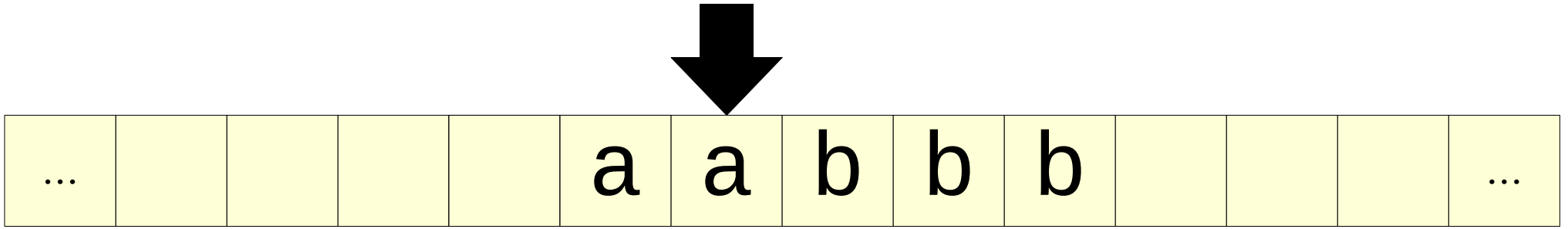
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



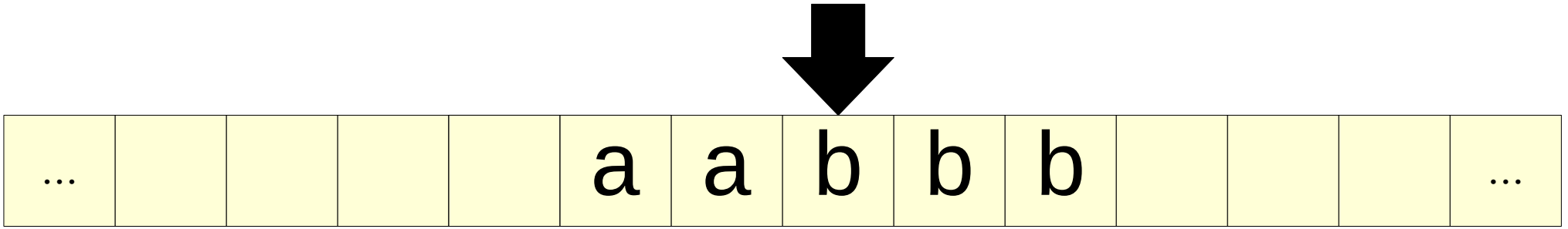
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



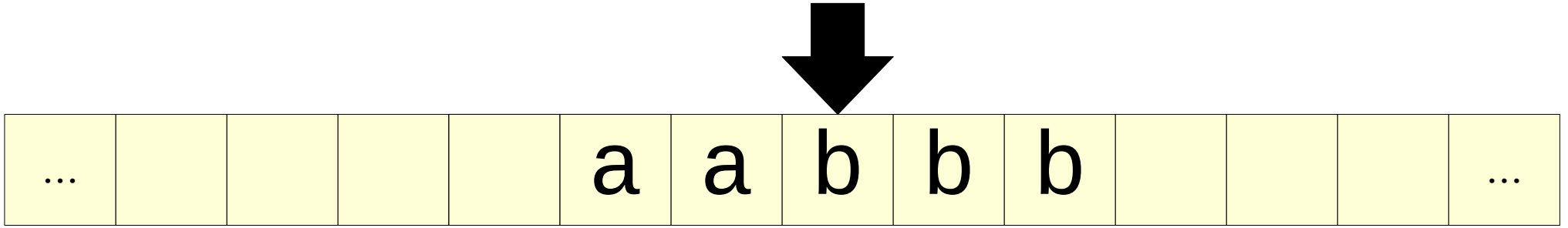
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



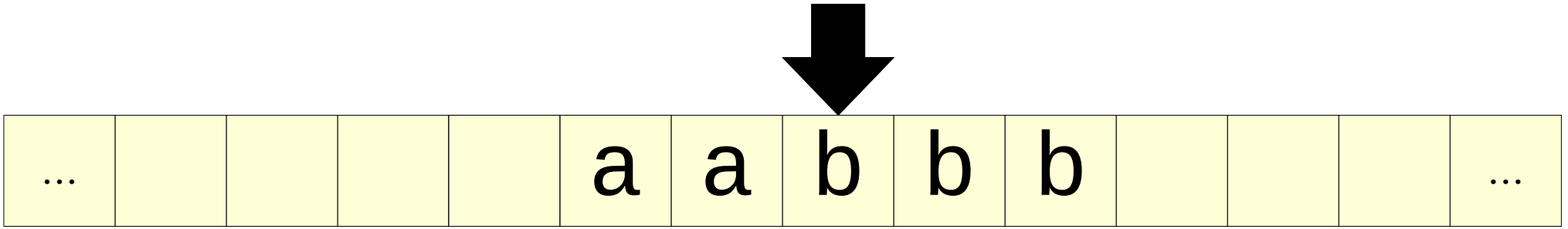
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



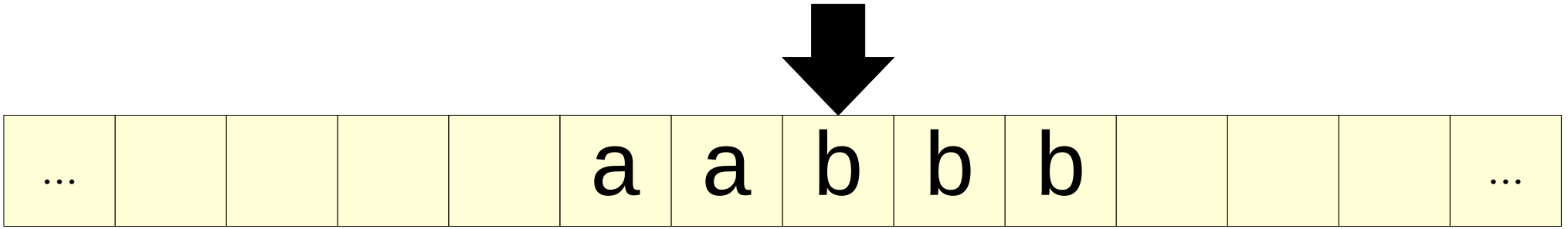
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



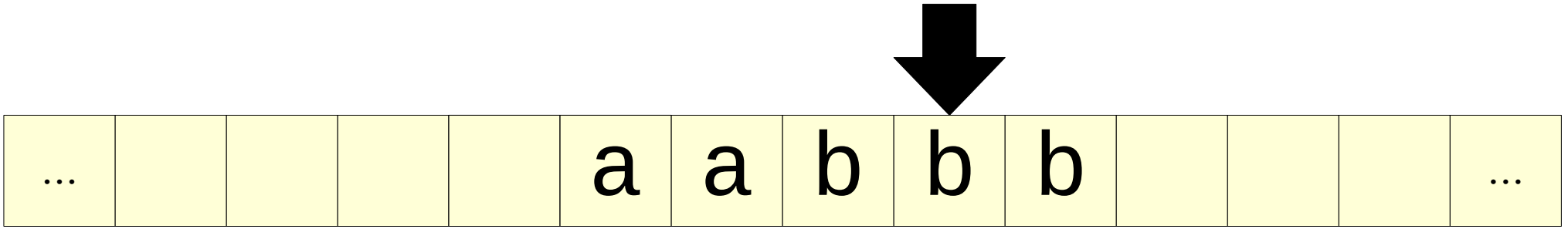
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



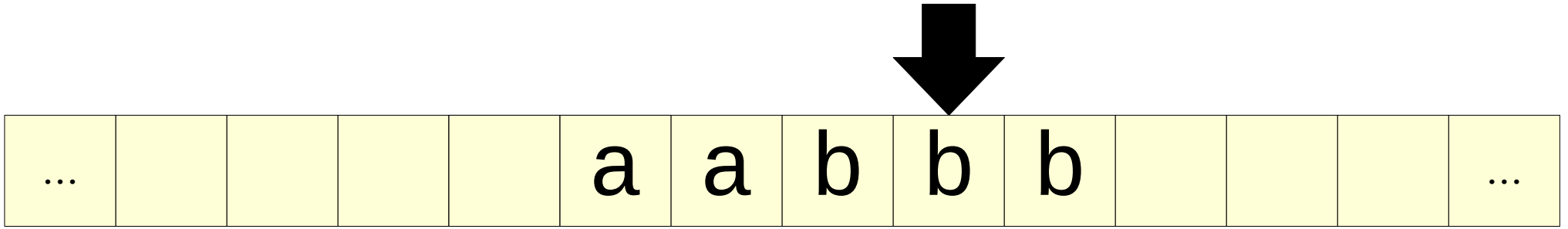
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



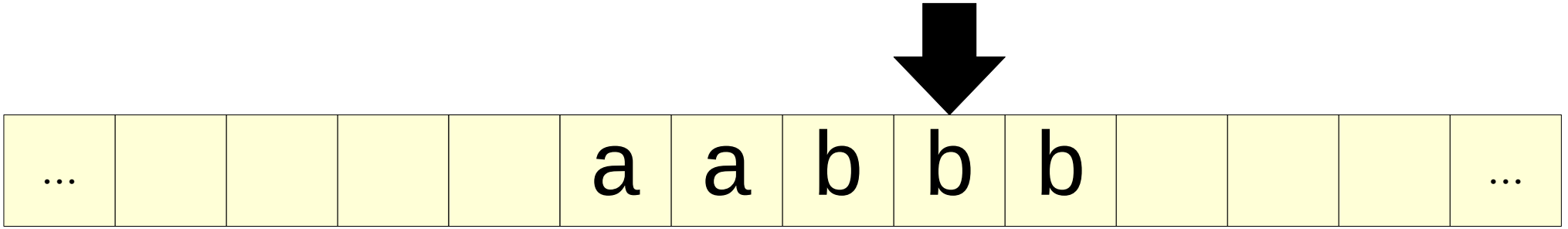
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



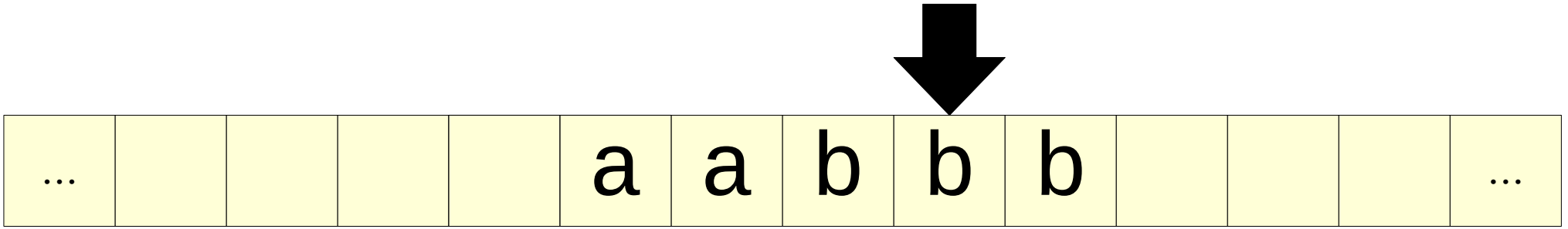
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



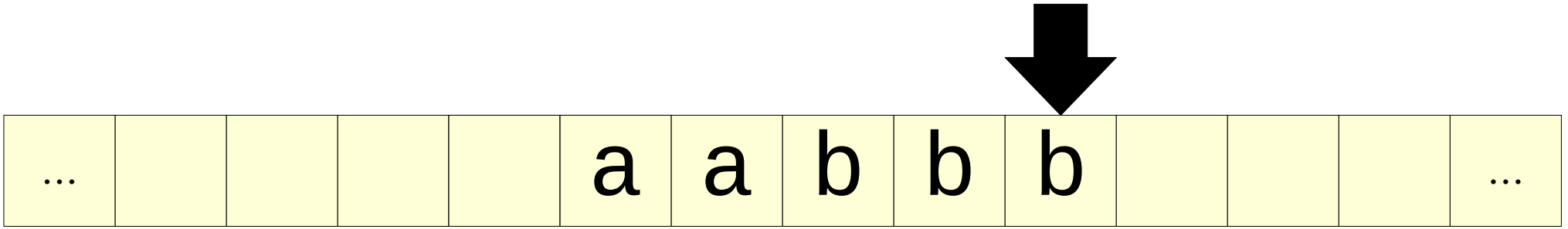
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



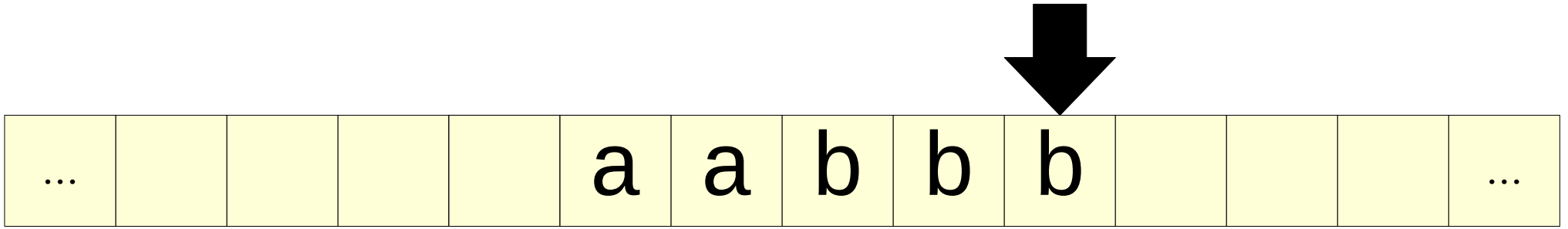
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



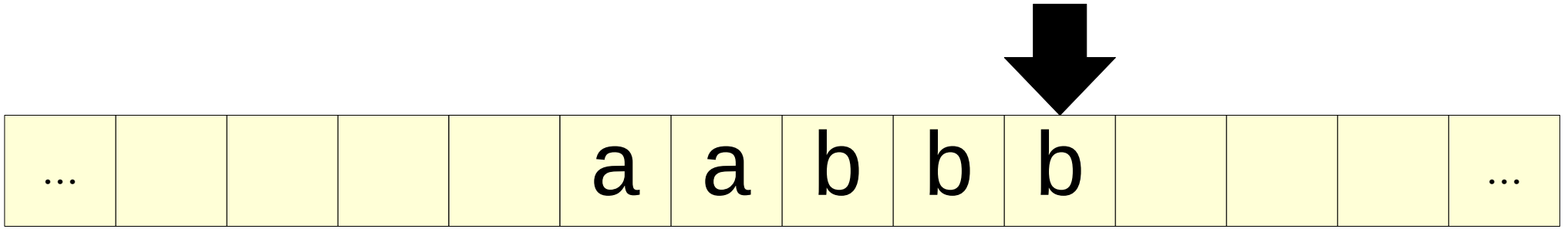
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



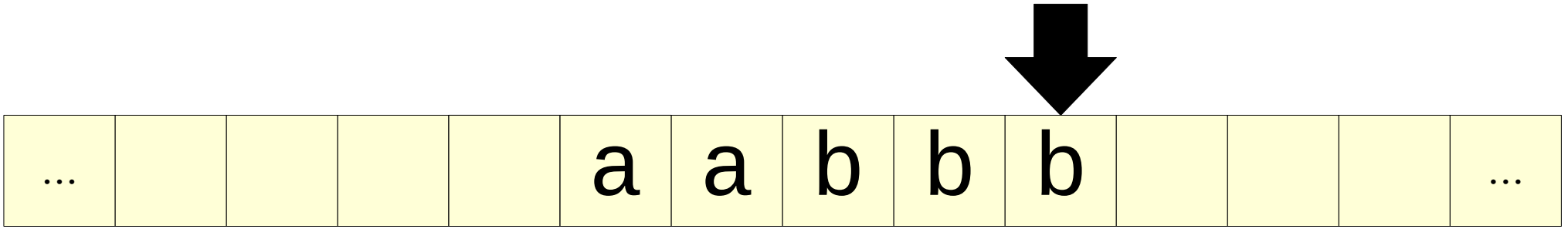
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



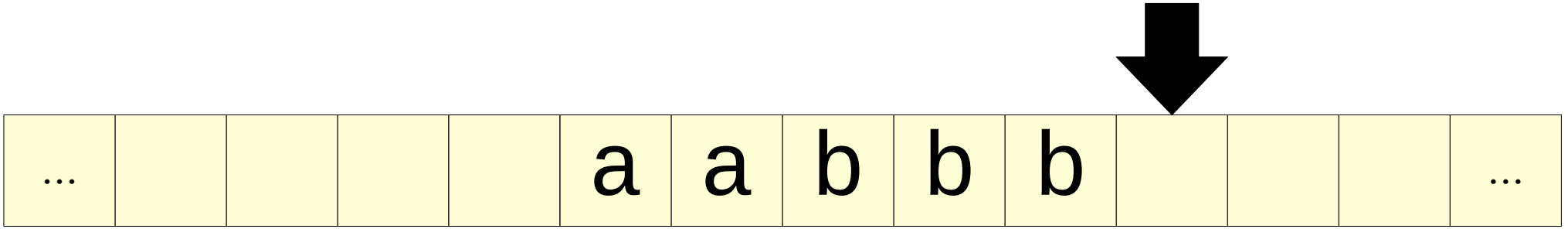
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



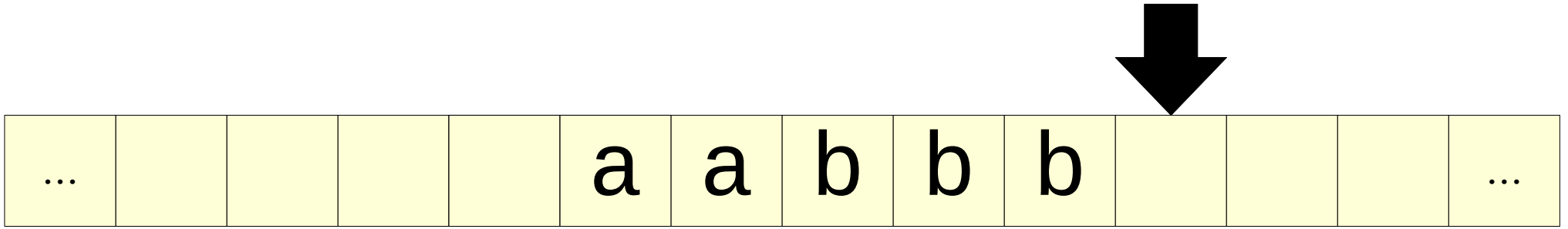
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



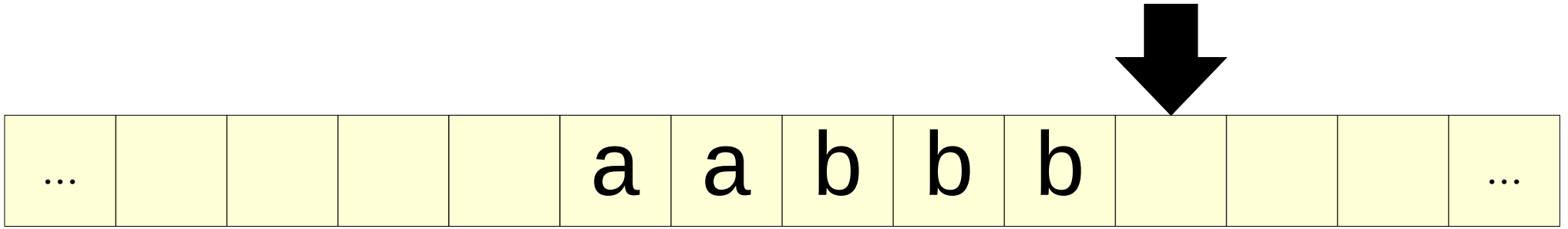
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



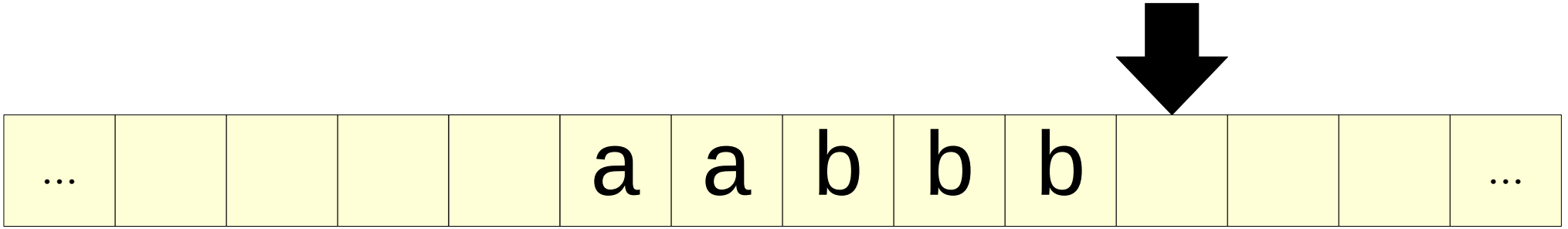
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



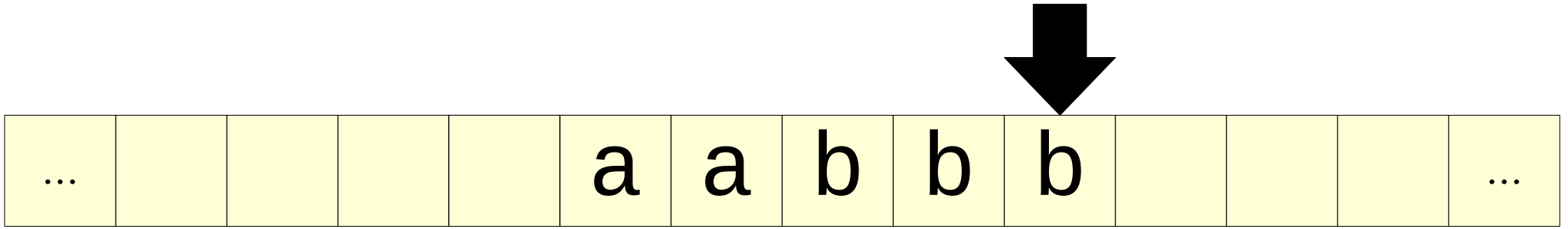
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
```



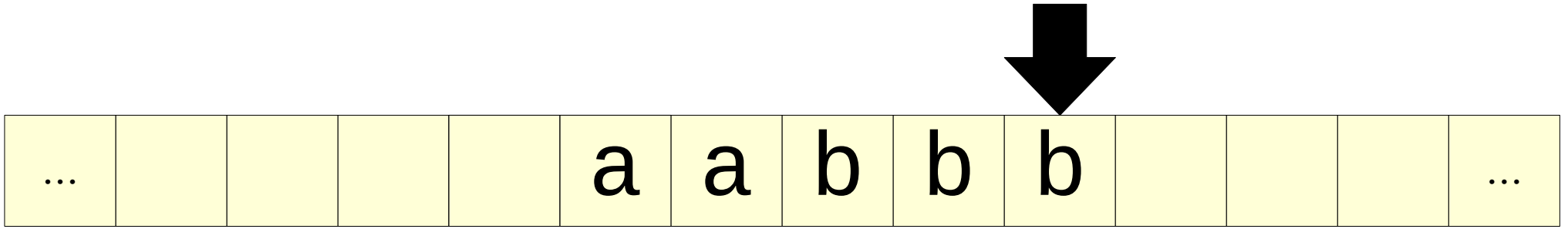
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
```



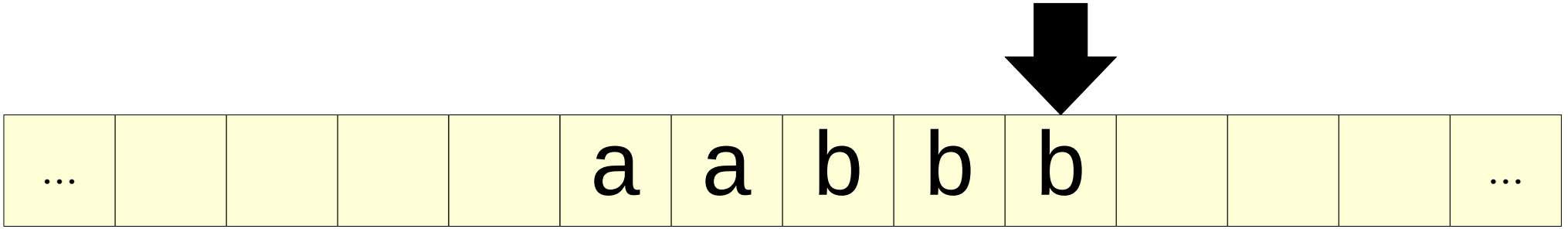
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
```



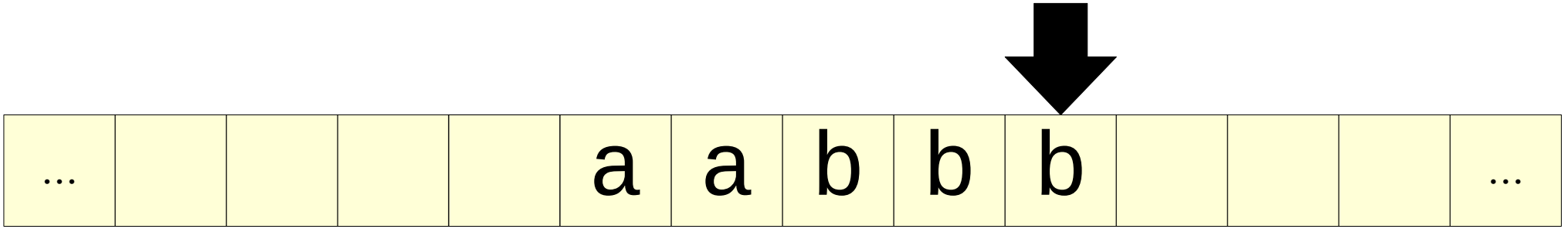
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
```



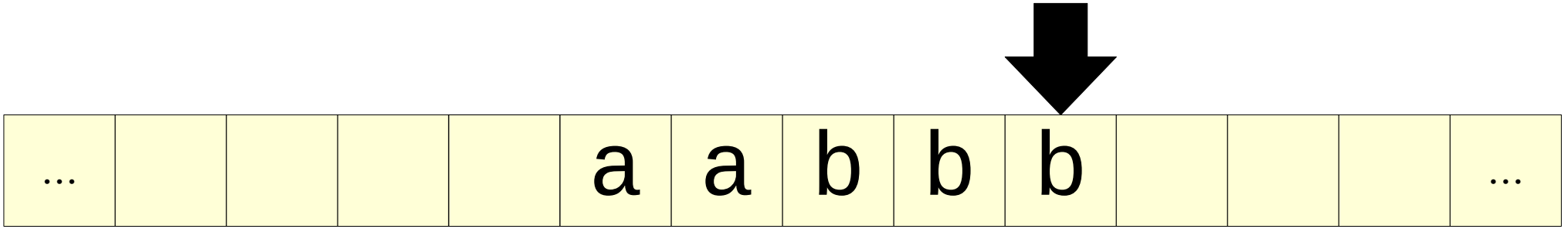
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
```



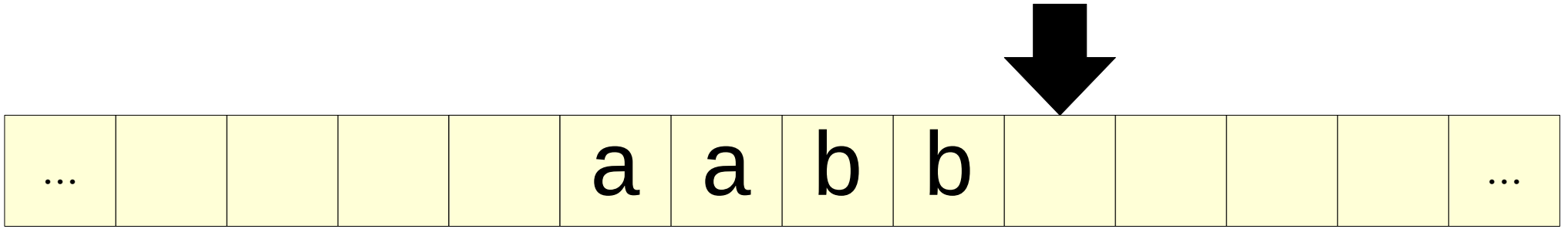
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
```



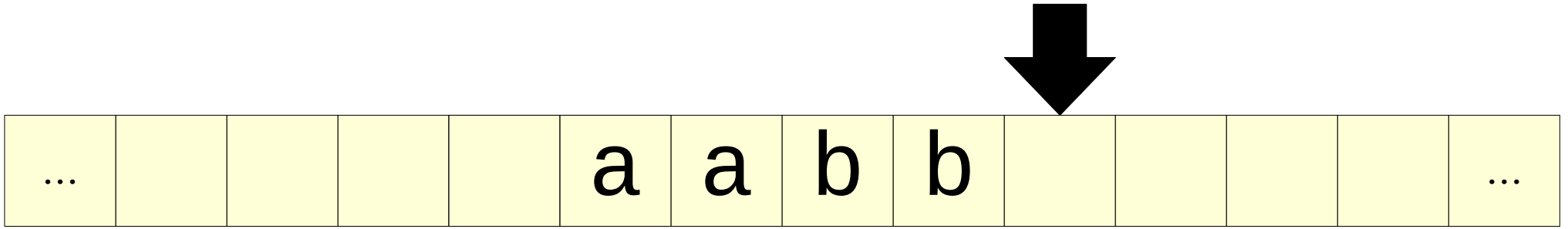
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank
```



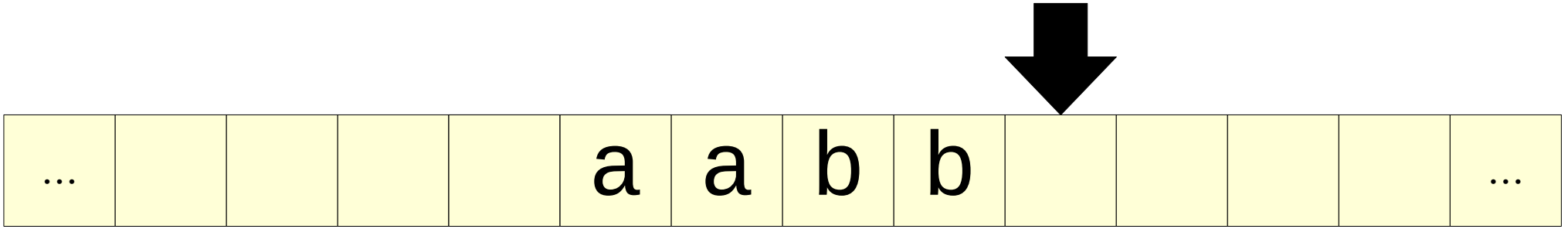
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank
```



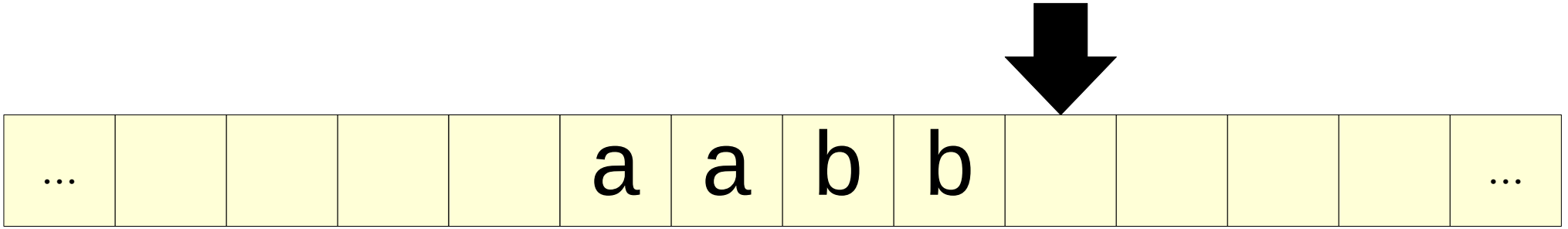
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank
```



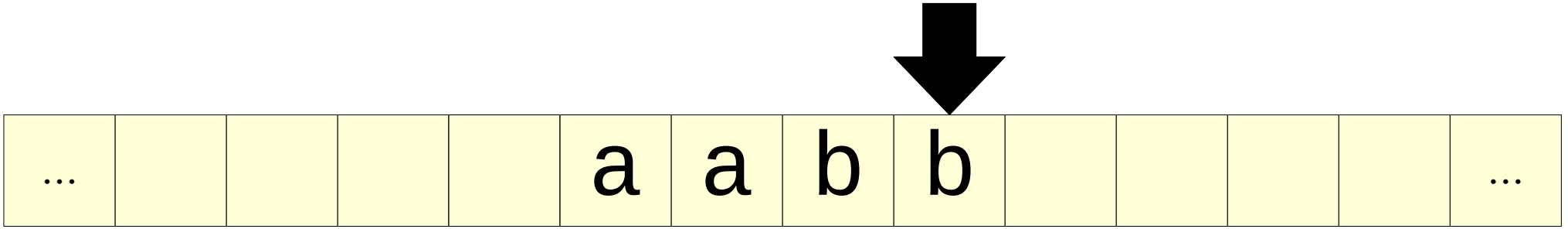
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



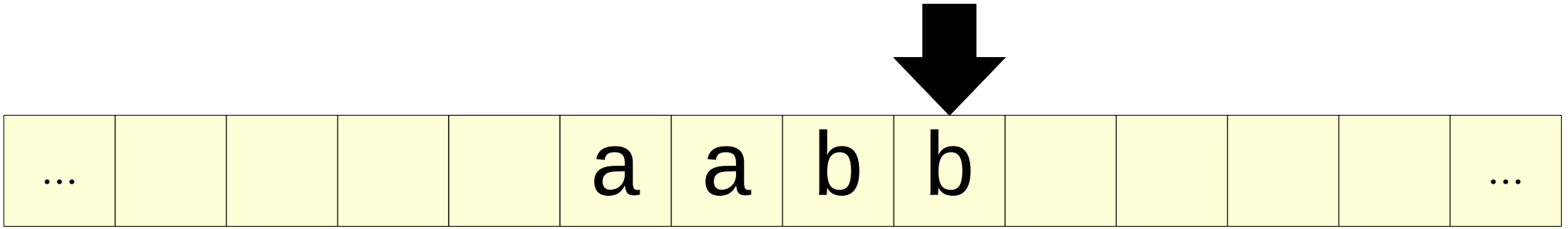
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



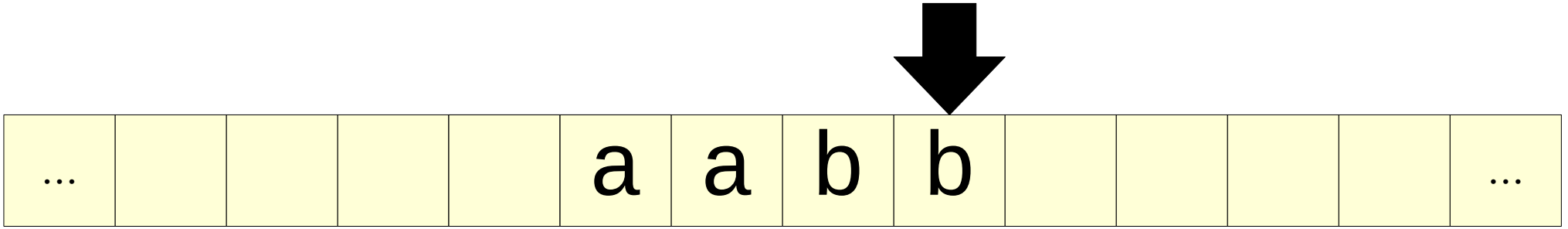
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



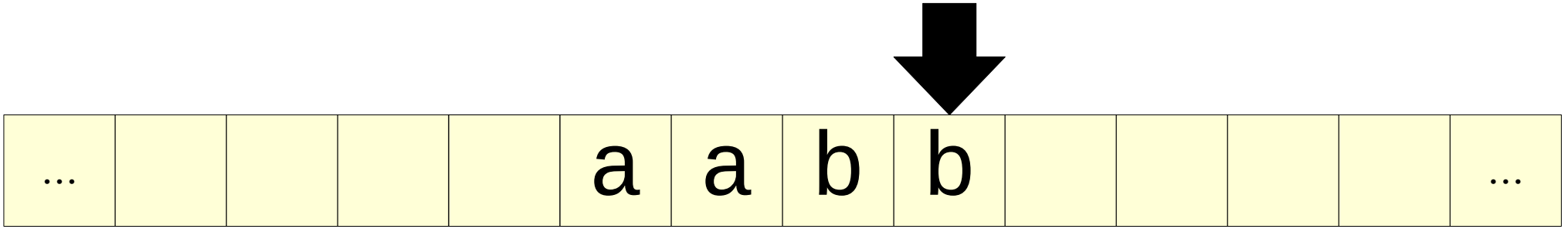
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



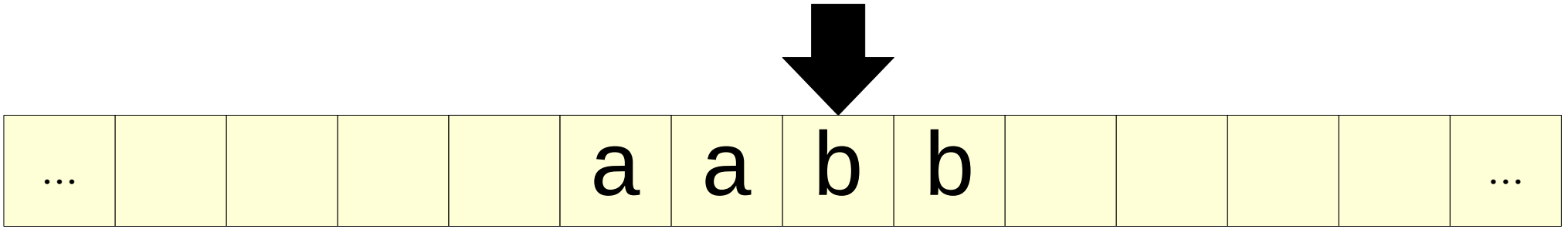
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



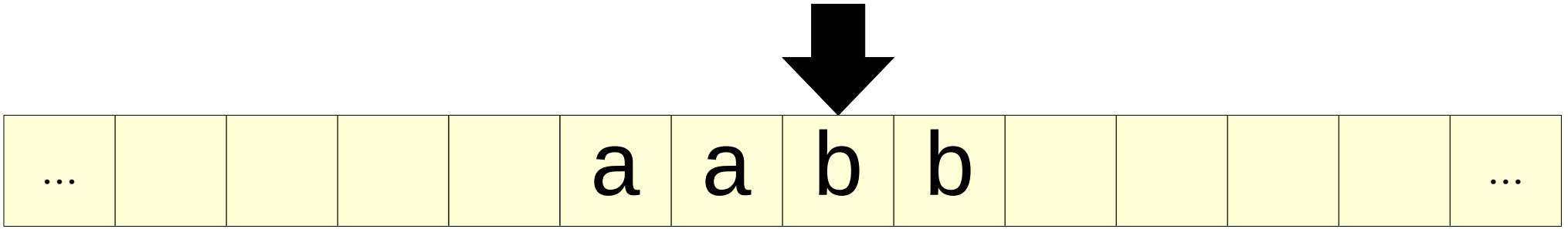
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



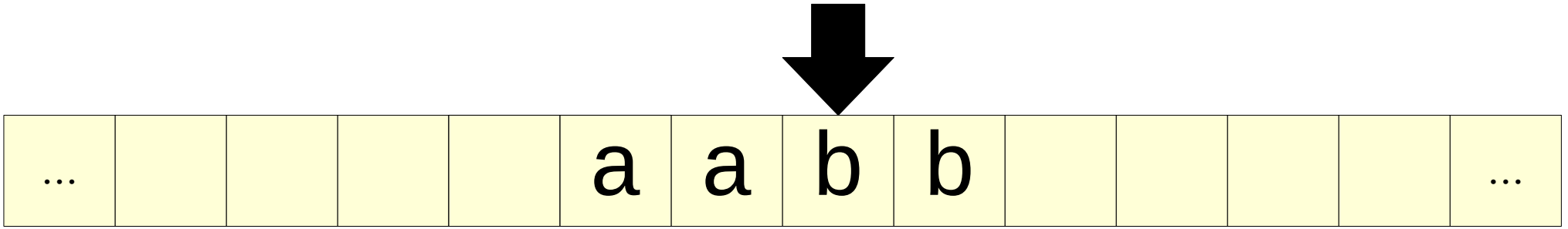
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



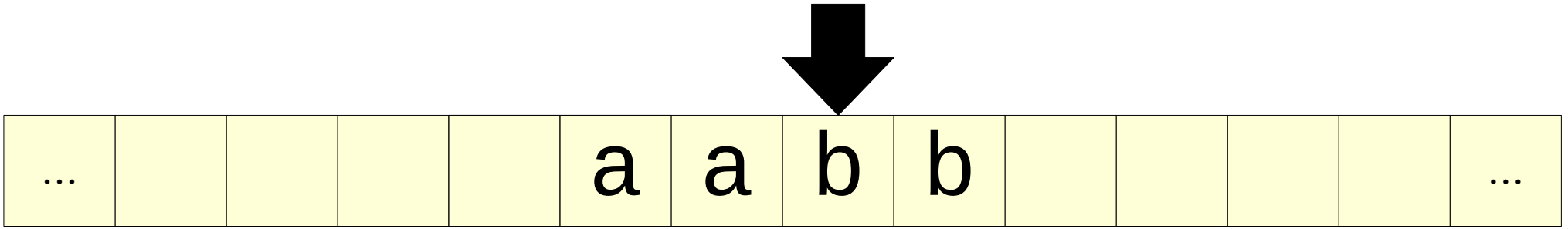
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



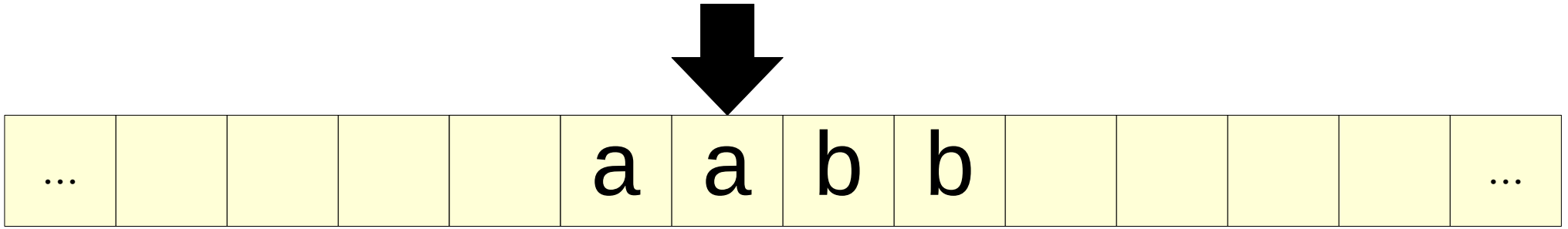
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



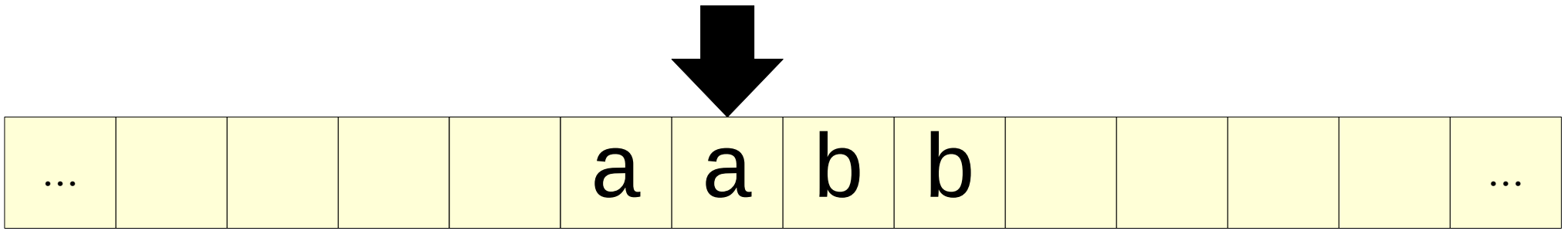
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



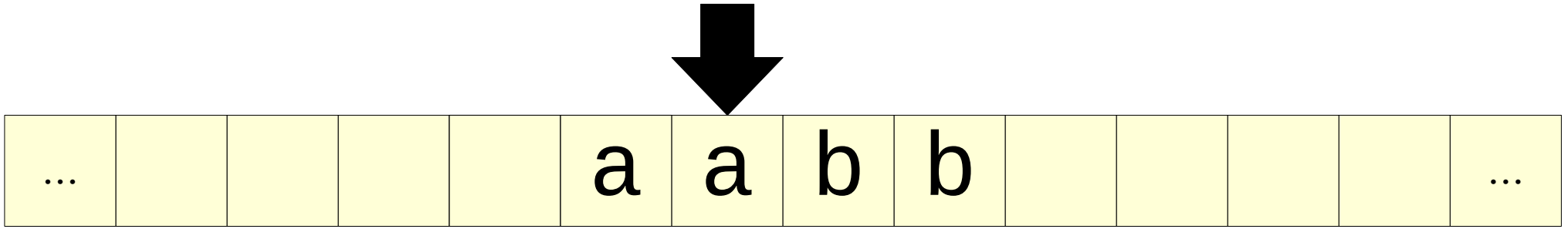
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



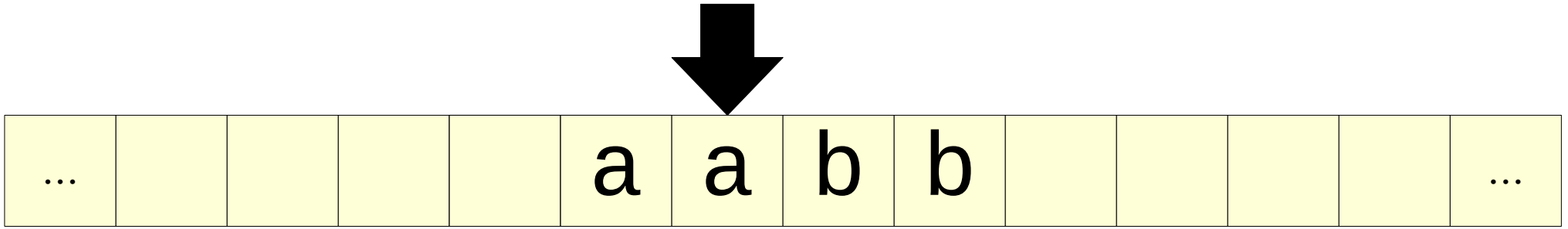
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



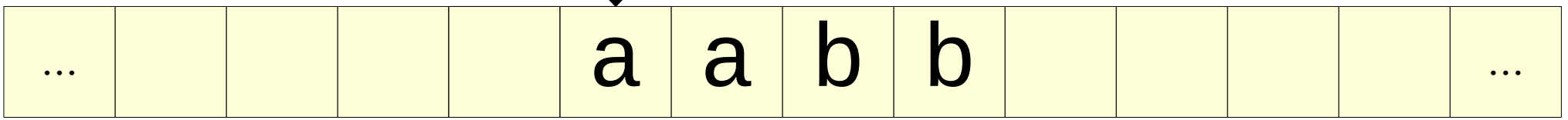
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



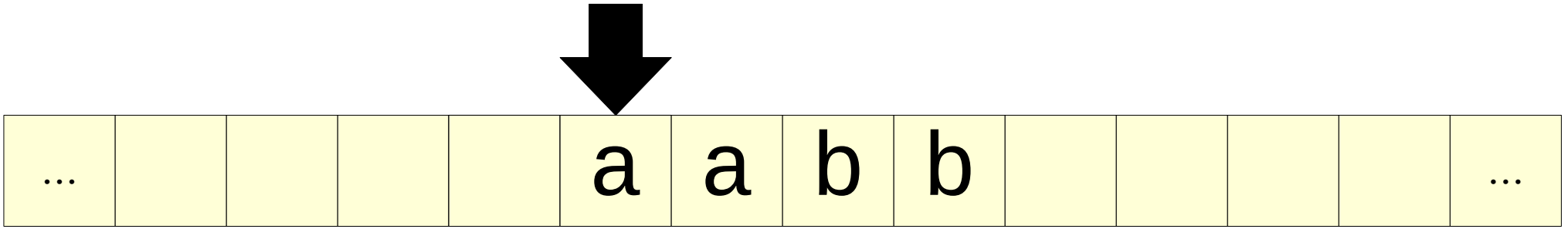
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



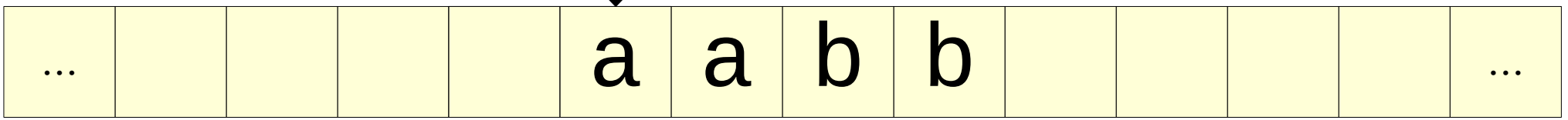
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



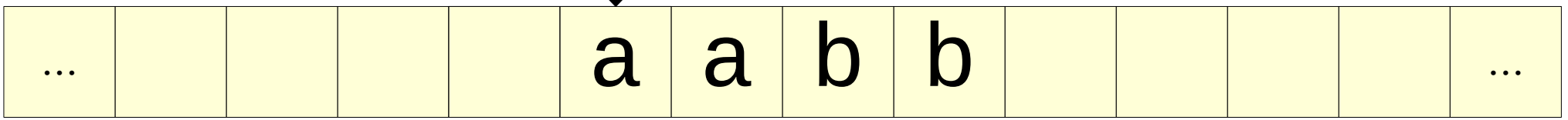
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



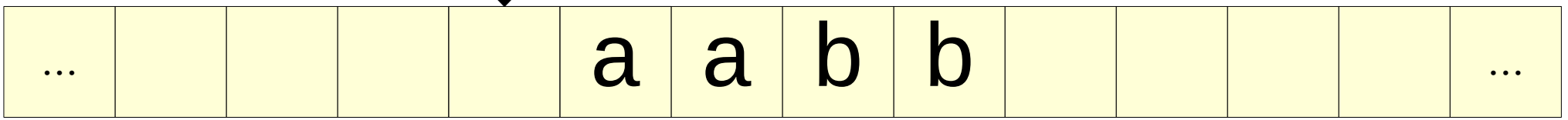
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



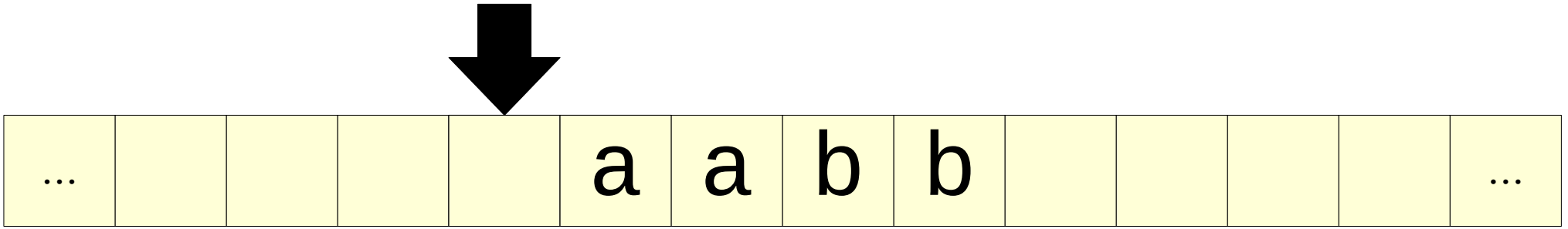
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



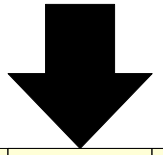
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```



```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```

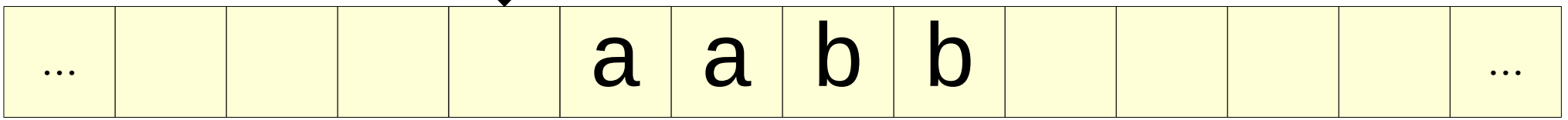


```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
```

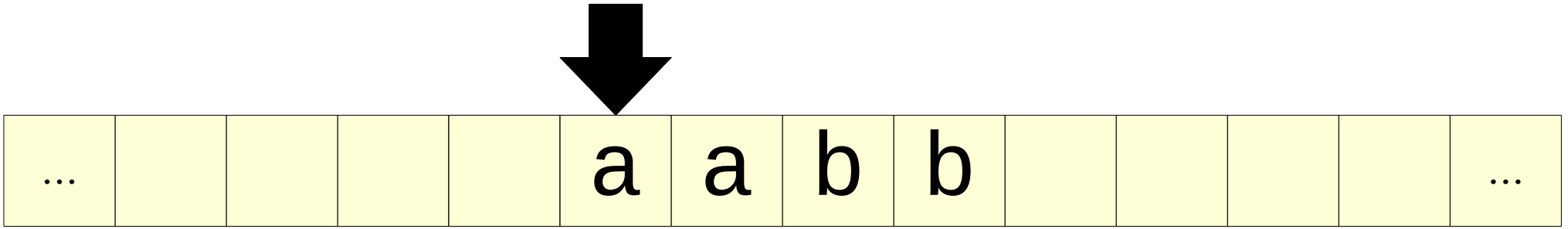


…					a	a	b	b					…

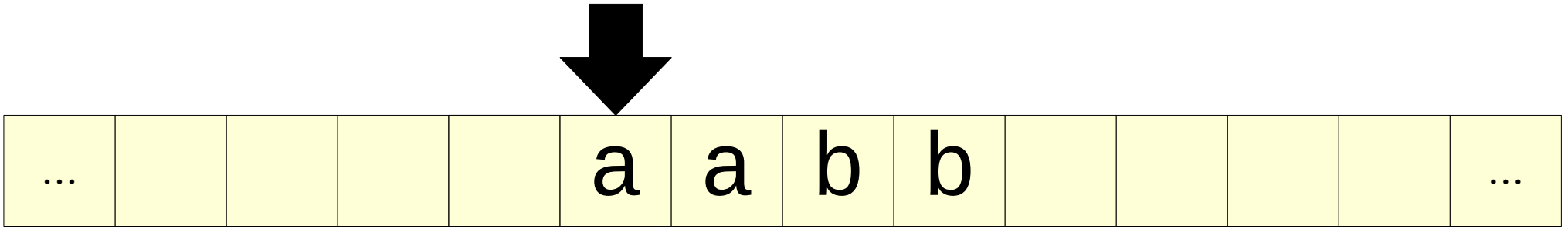
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
```



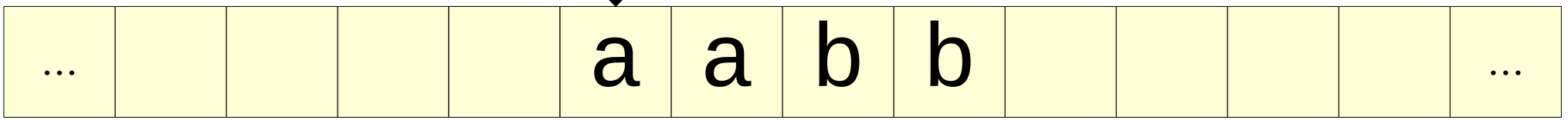
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
```



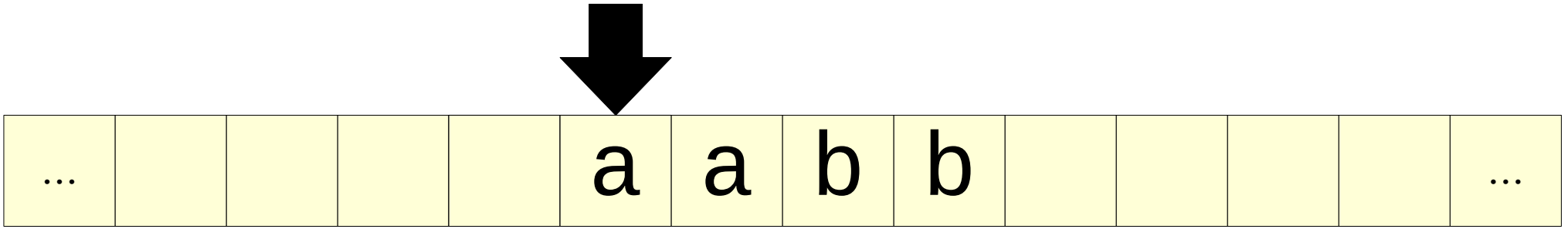
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
```



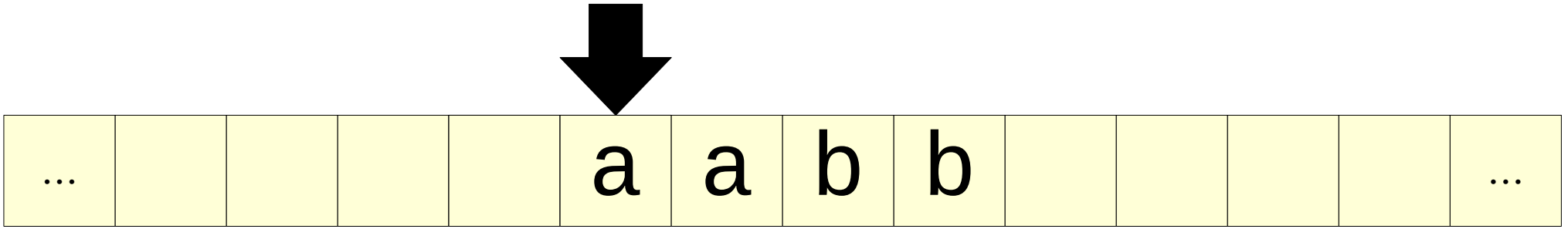
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



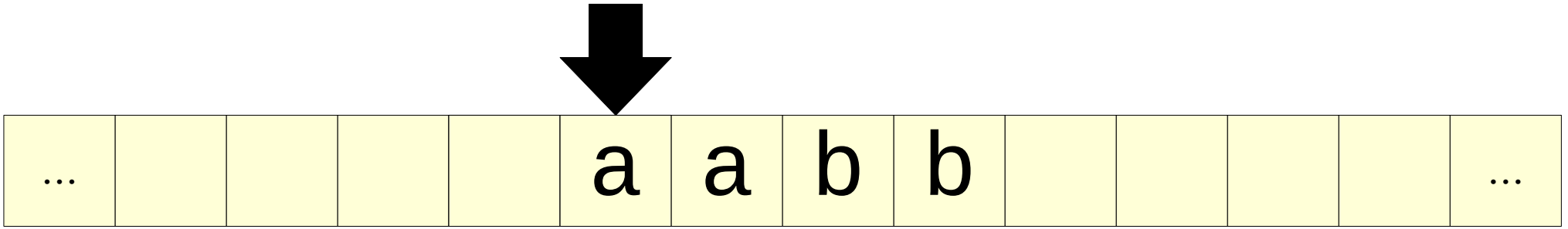
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



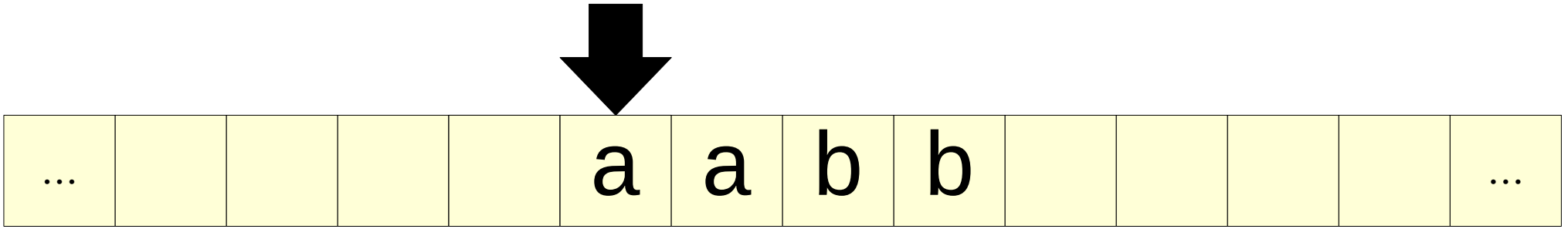
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



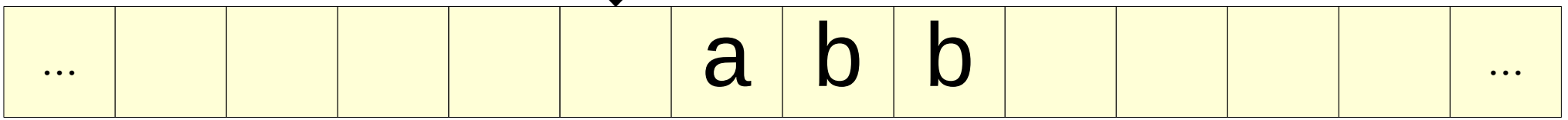
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



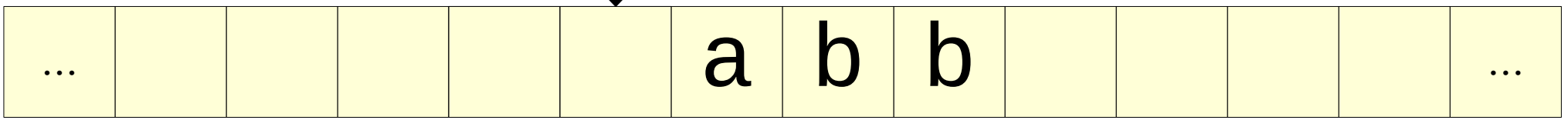
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



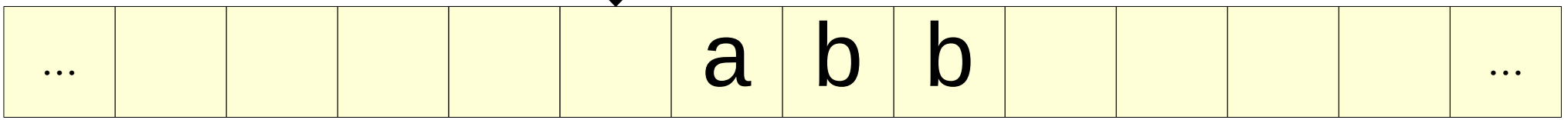
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



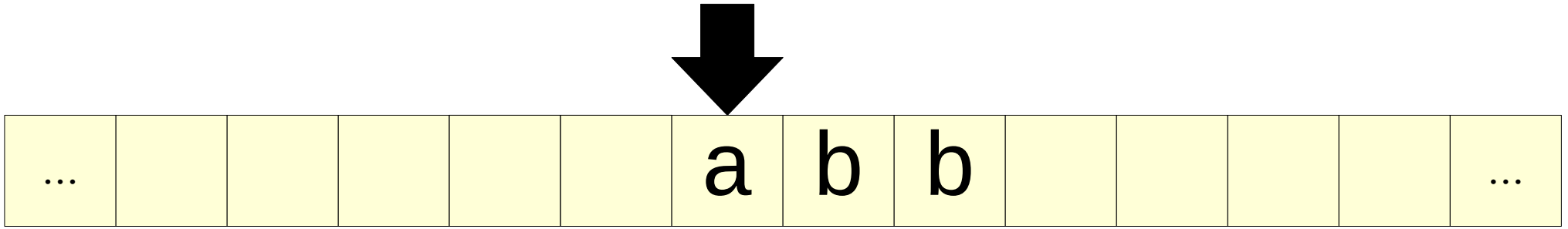
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



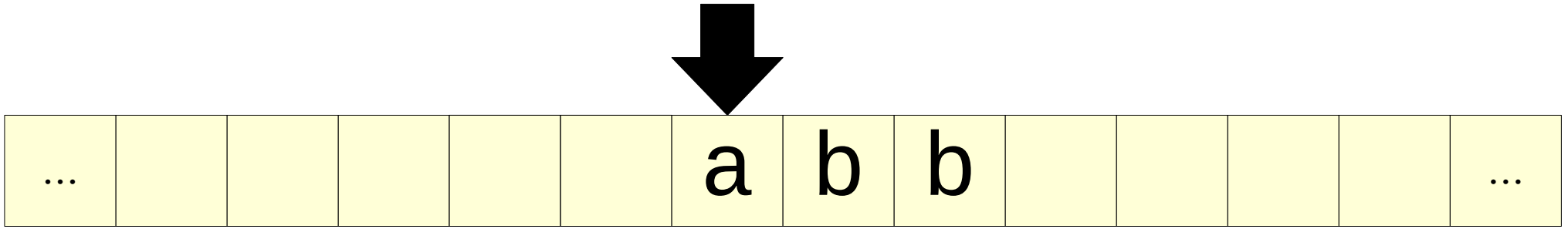
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



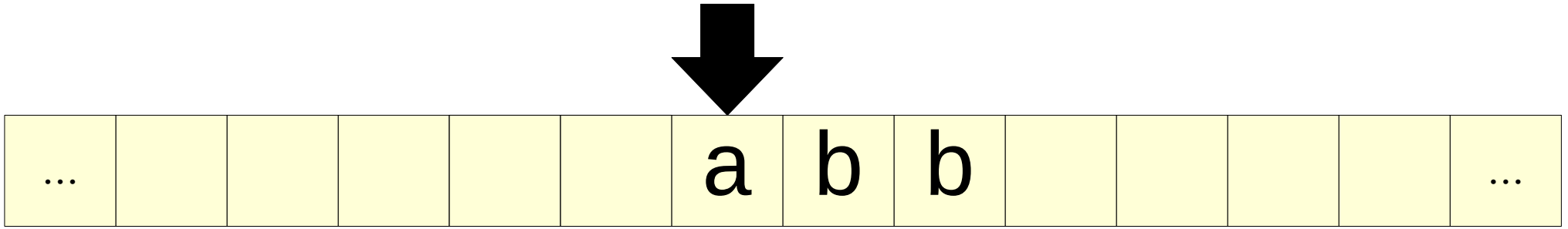
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



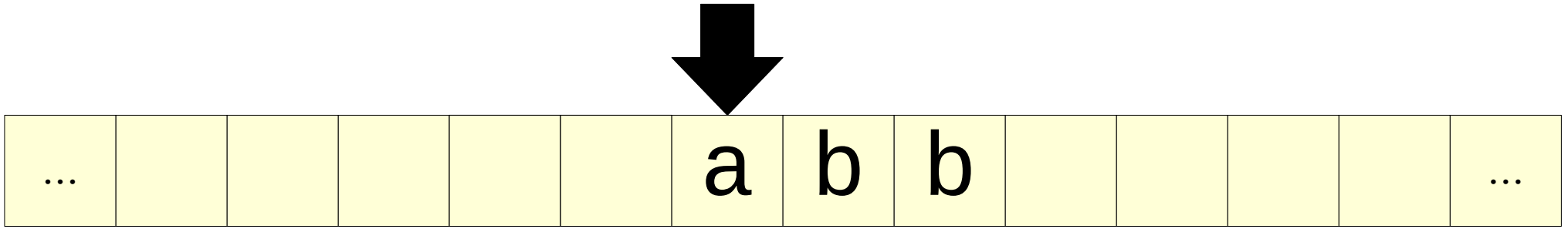
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



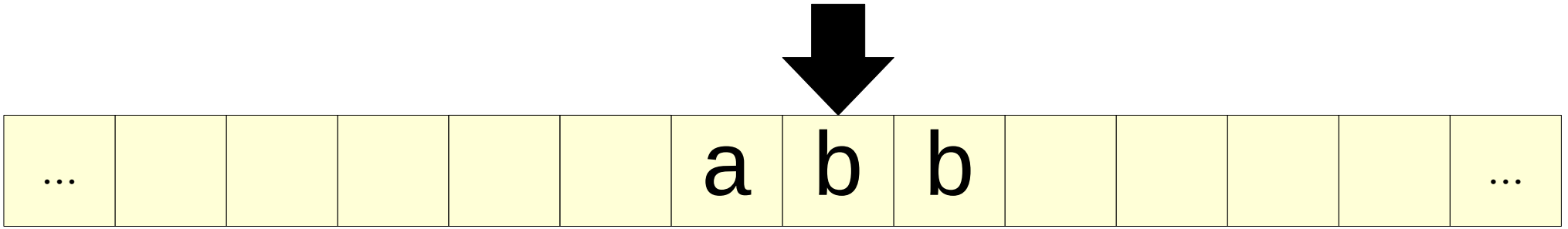
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



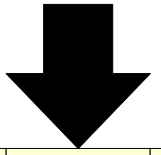
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```

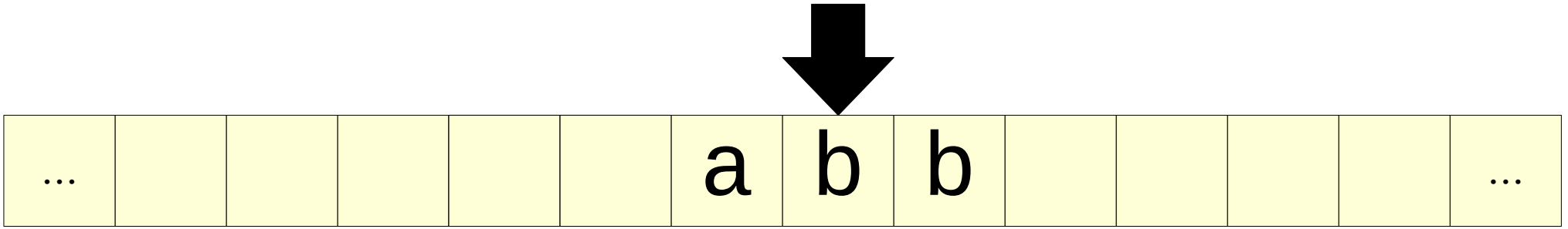


```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```

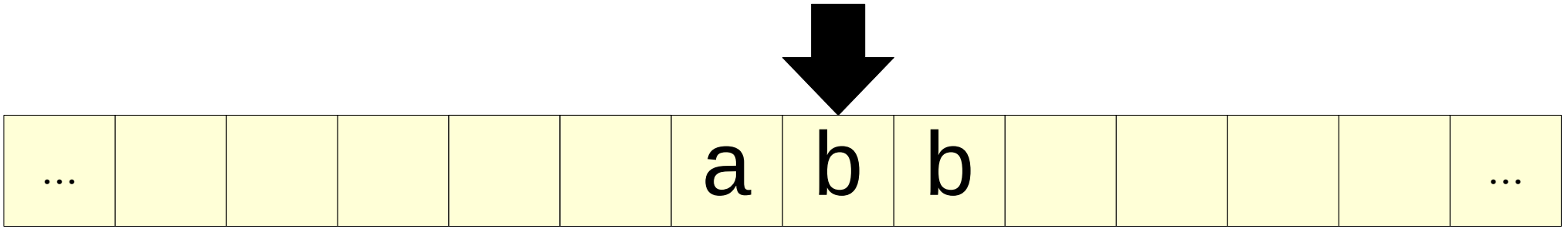


…						a	b	b					…

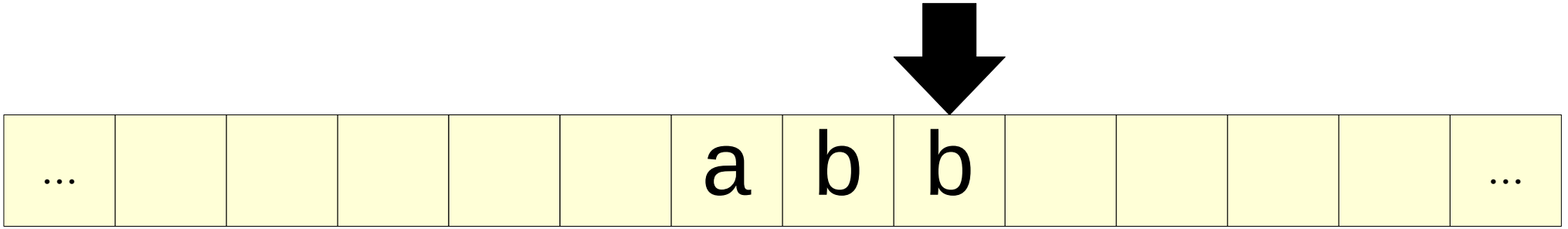
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



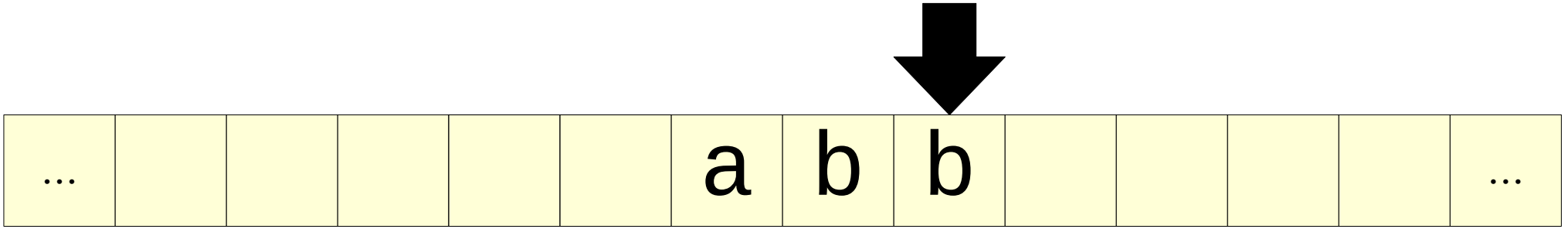
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



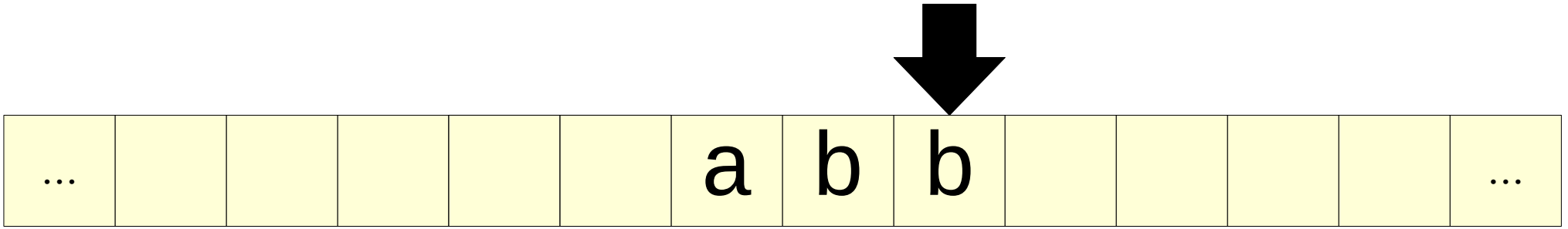
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



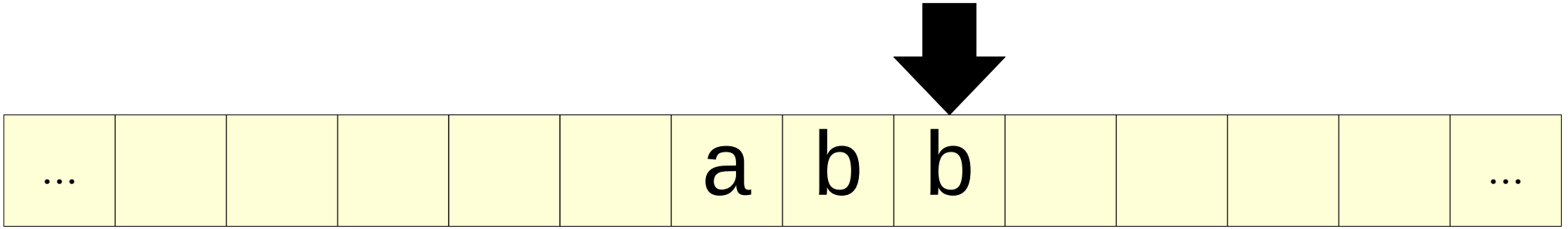
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



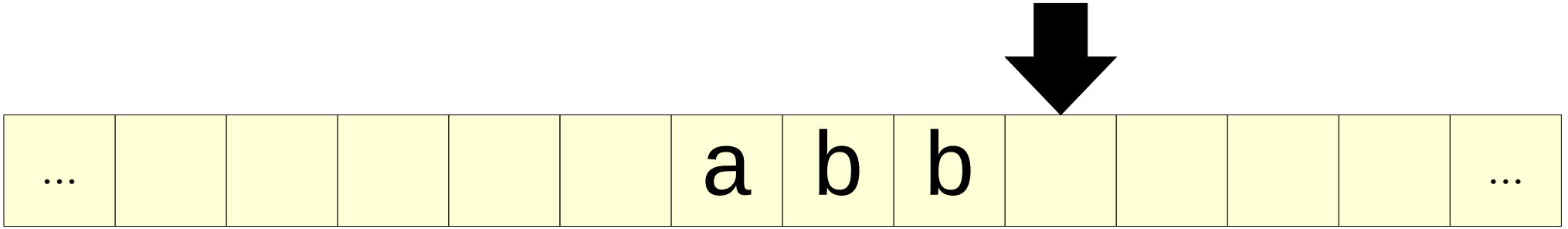
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



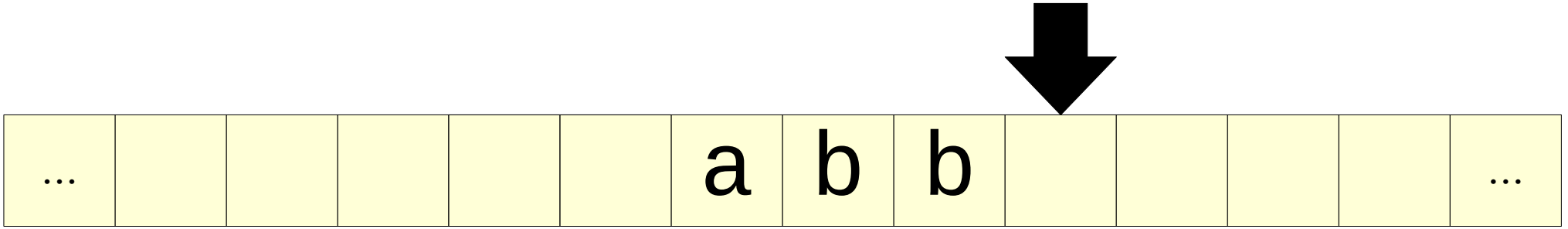
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



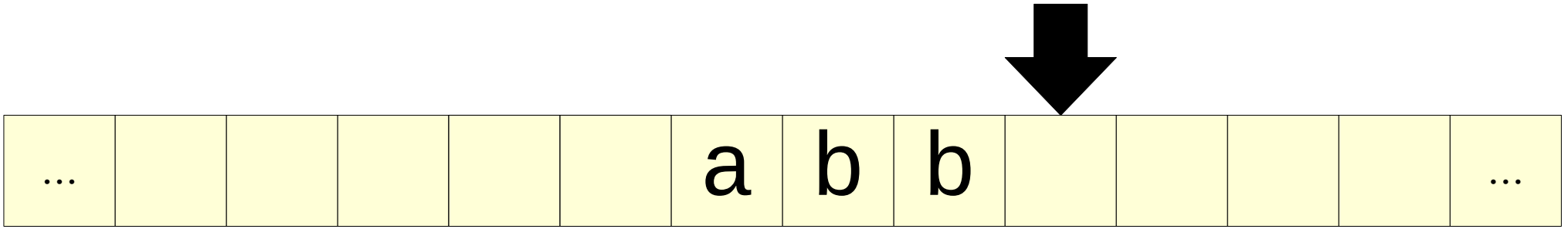
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



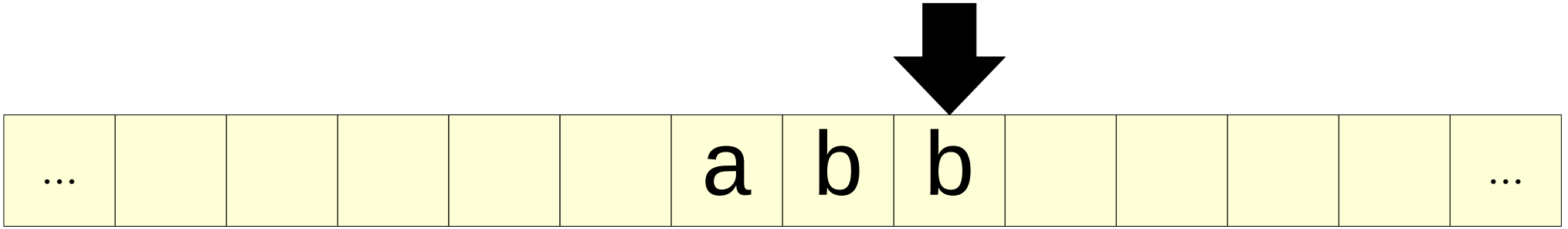
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



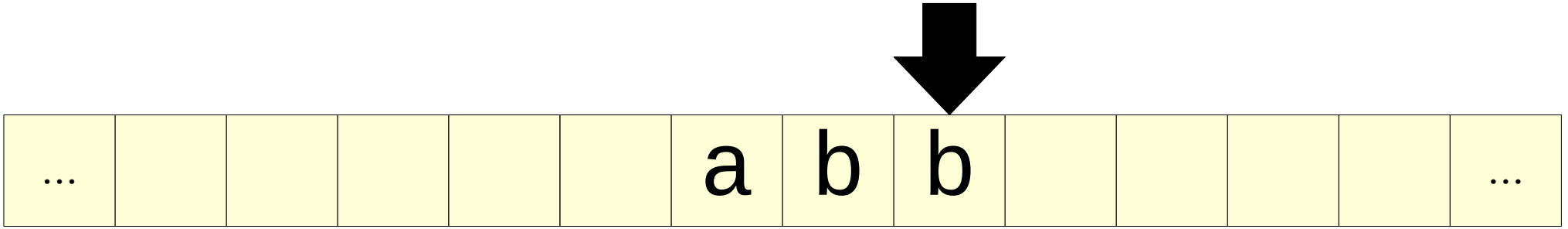
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



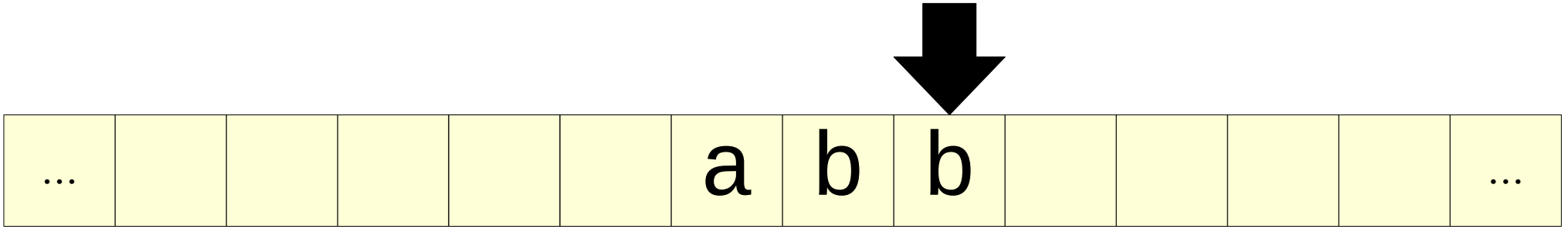
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



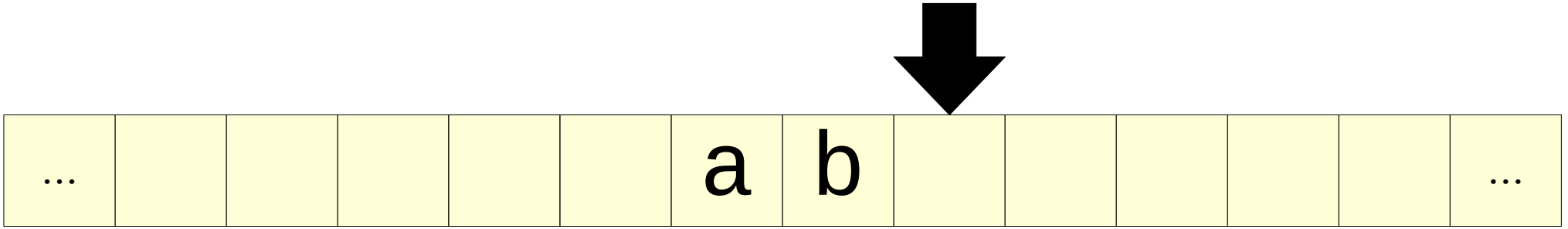
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



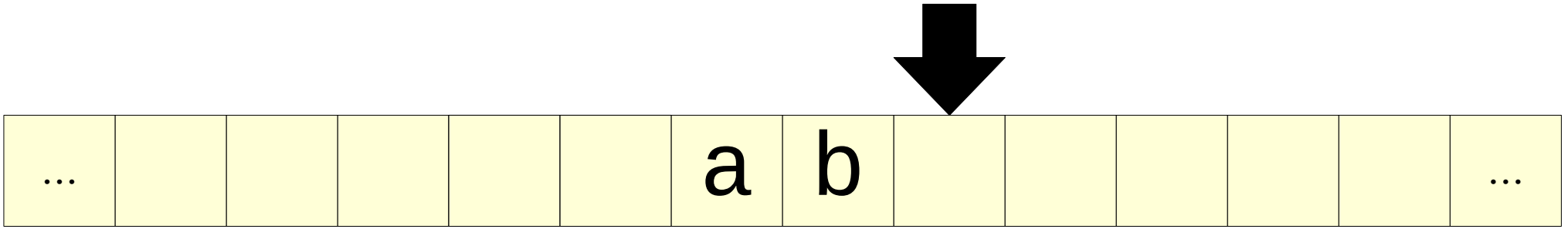
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



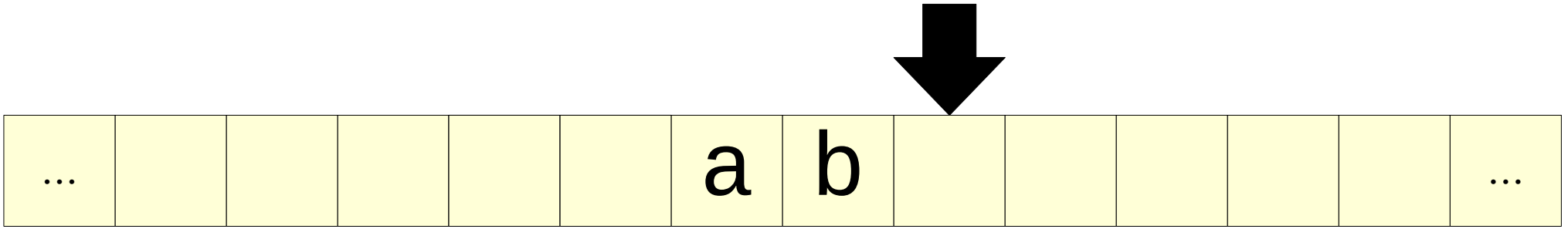
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



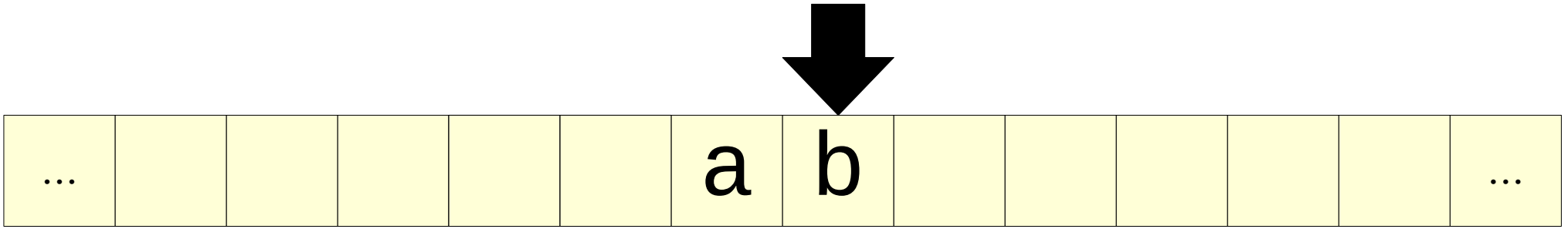
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



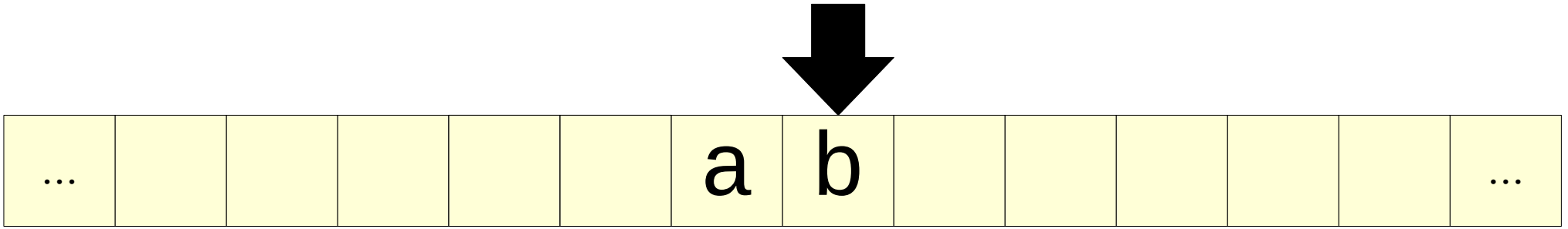
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



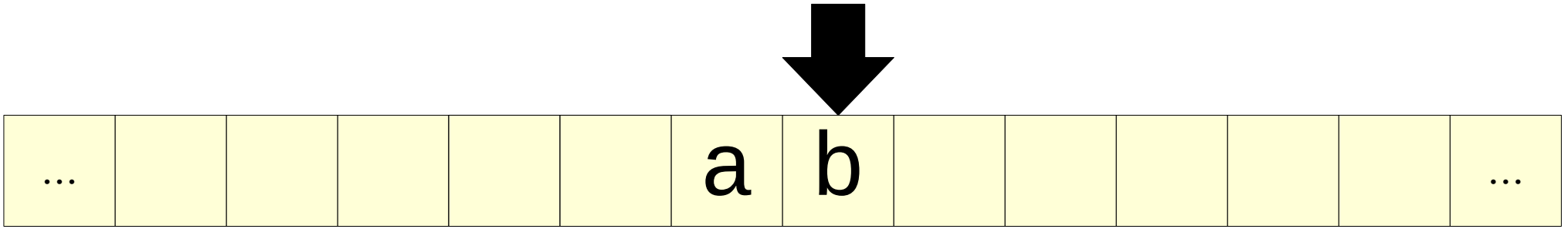
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



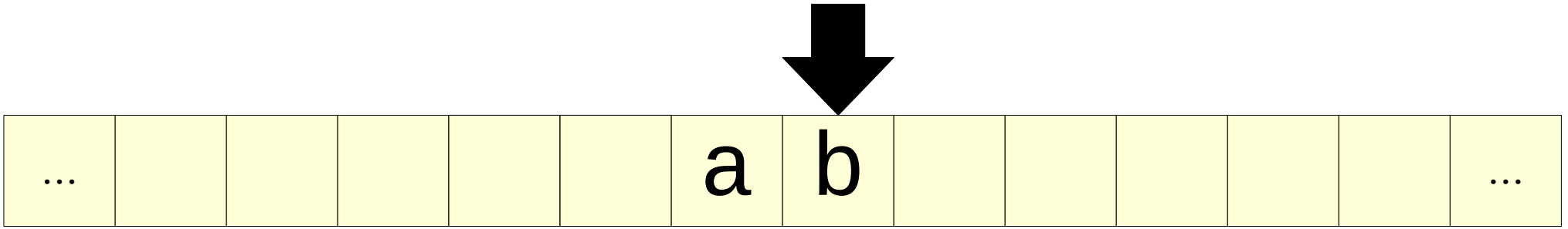
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



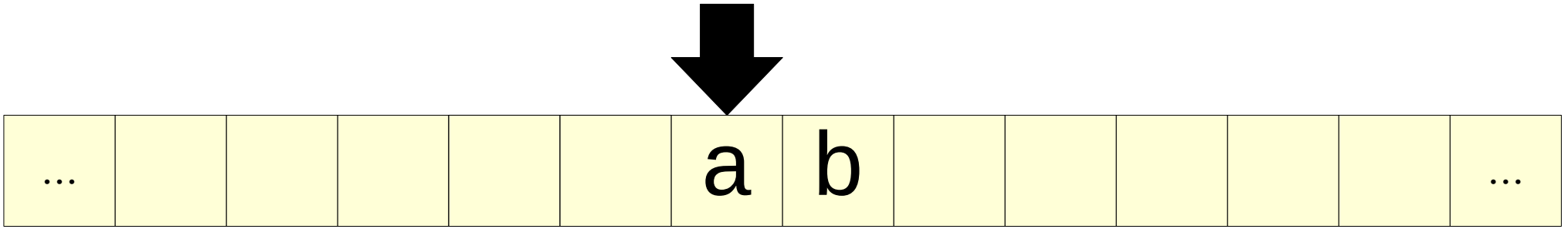
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



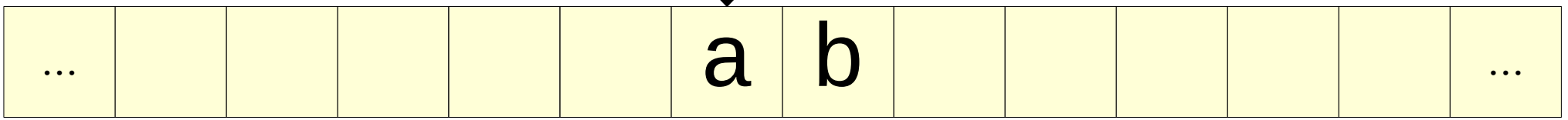
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



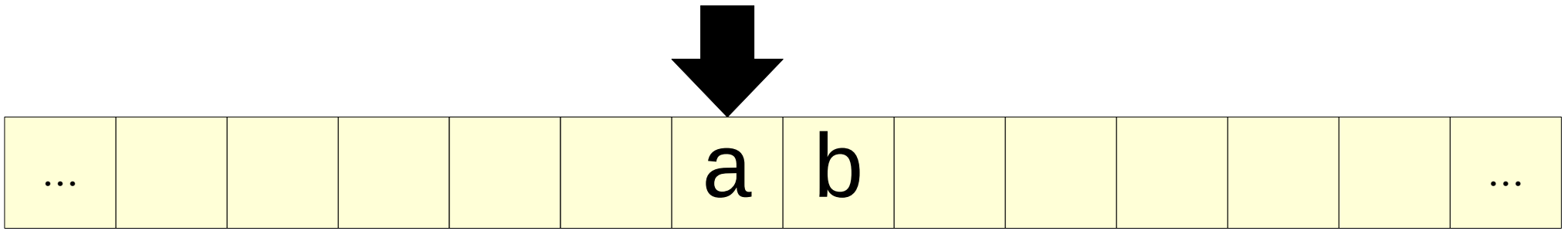
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



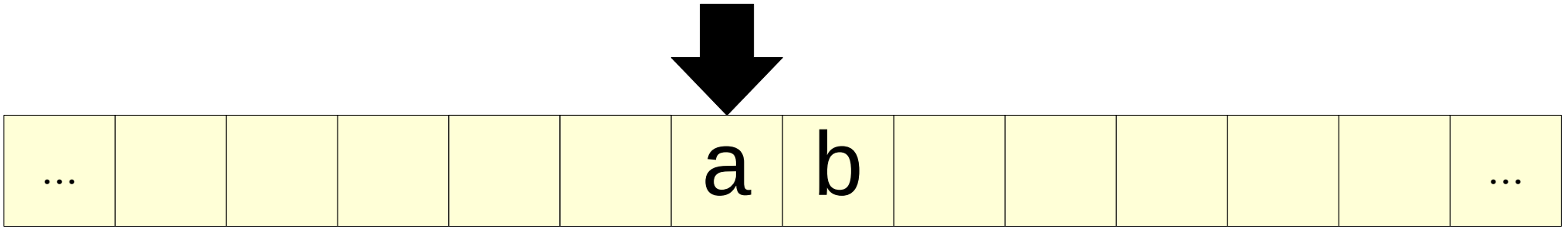
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



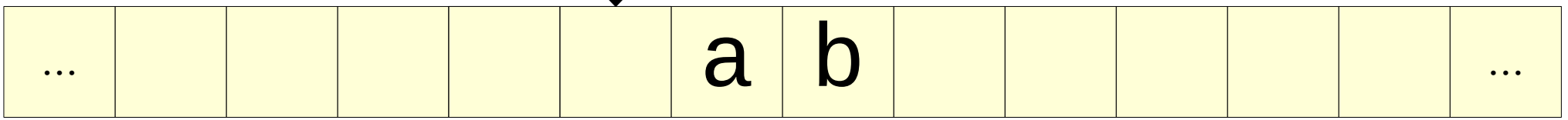
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



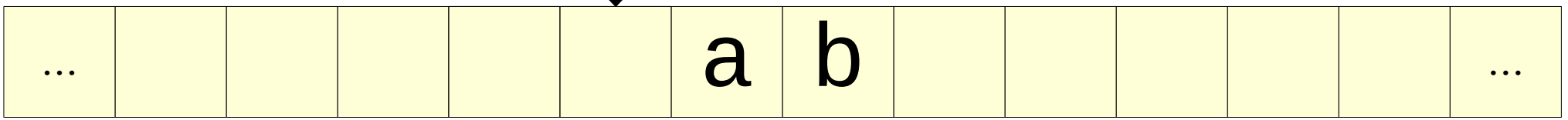
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



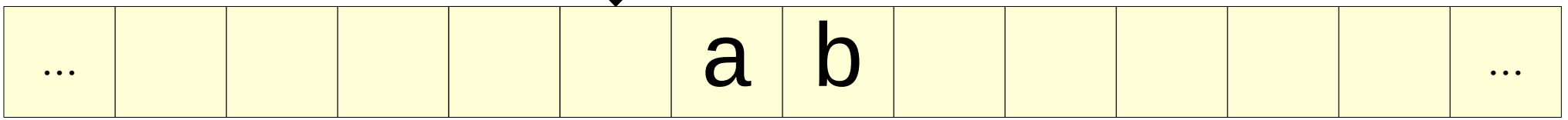
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



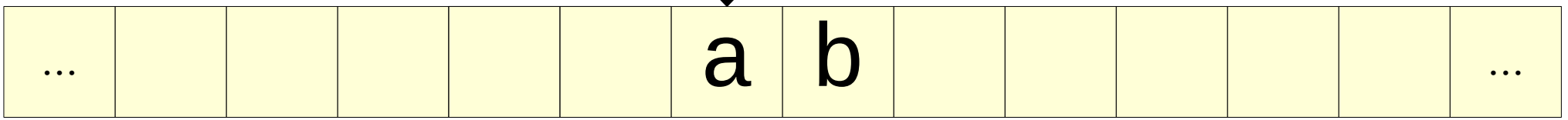
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



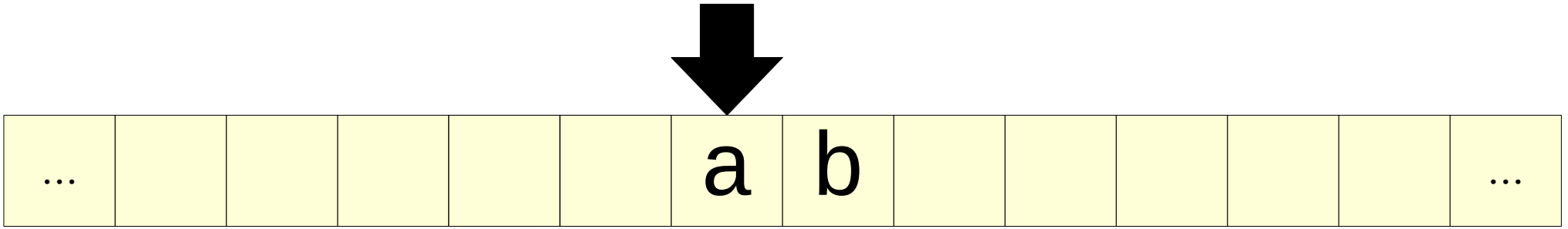
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



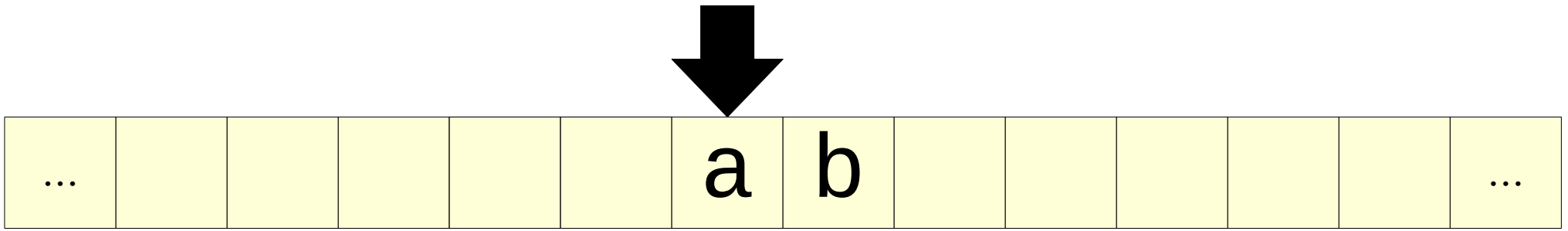
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



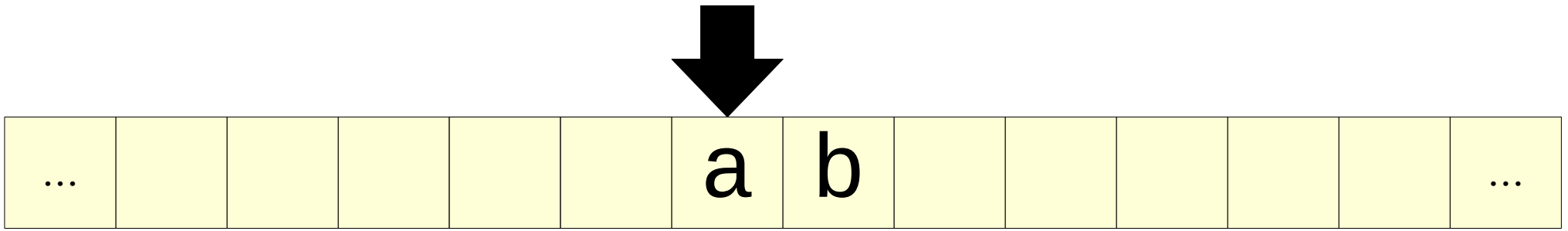
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



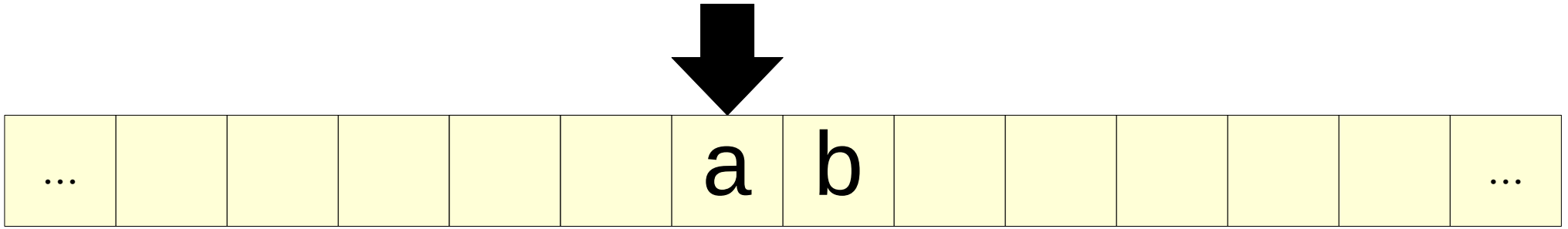
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



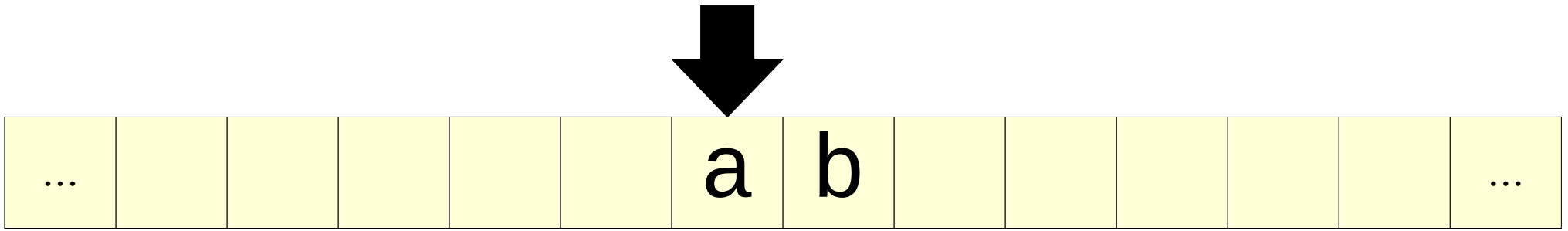
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



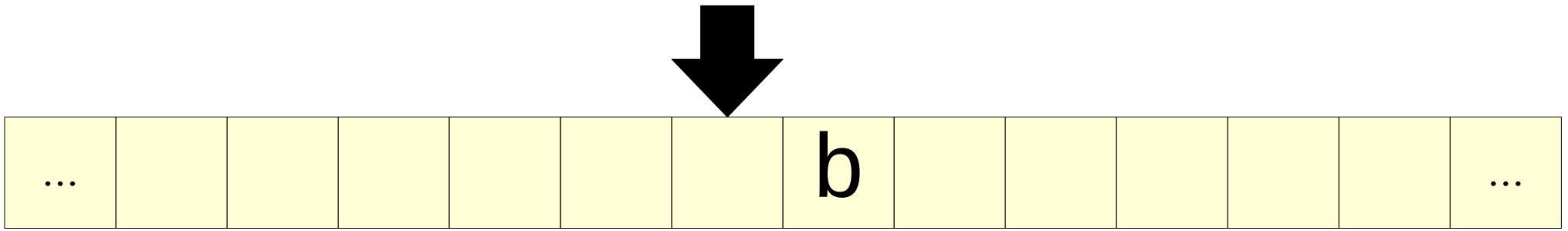
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



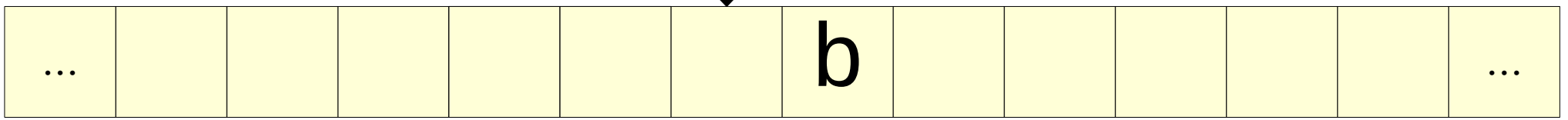
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



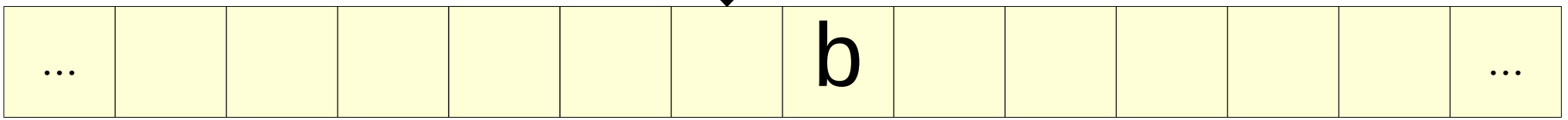
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



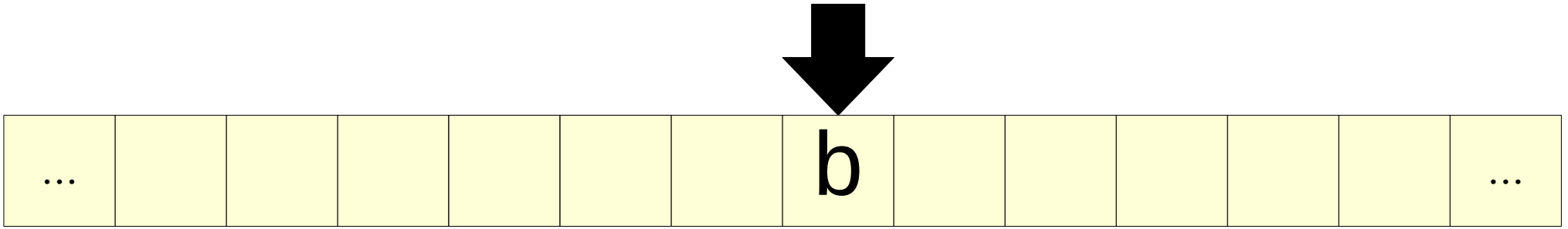
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



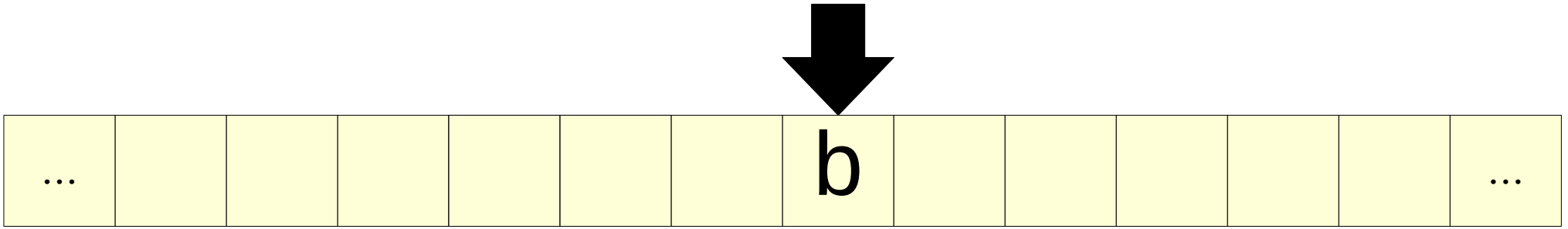
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



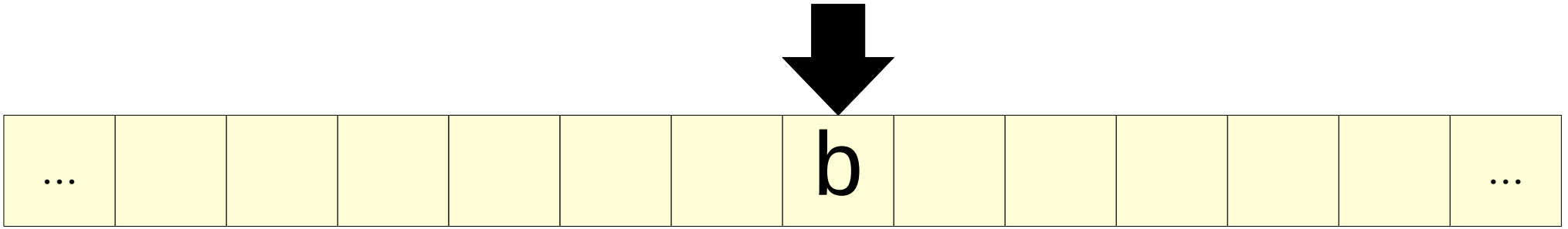
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



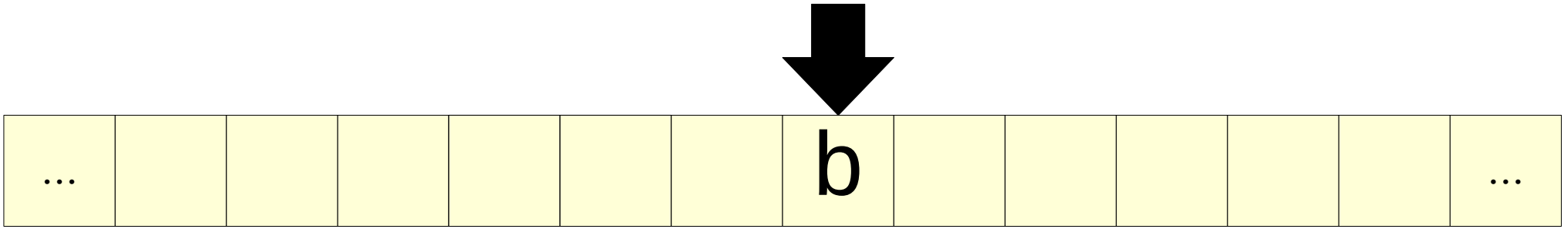
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



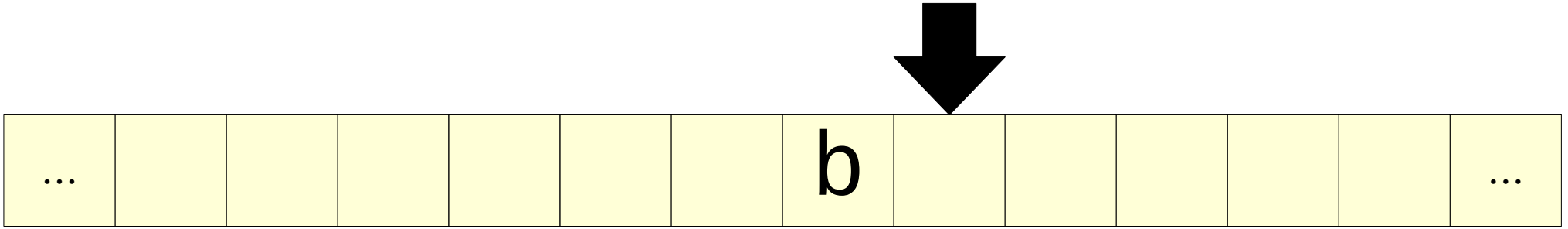
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



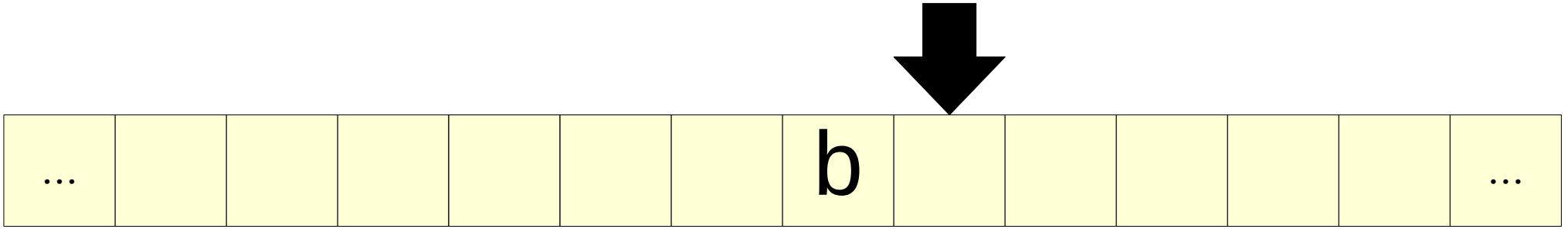
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



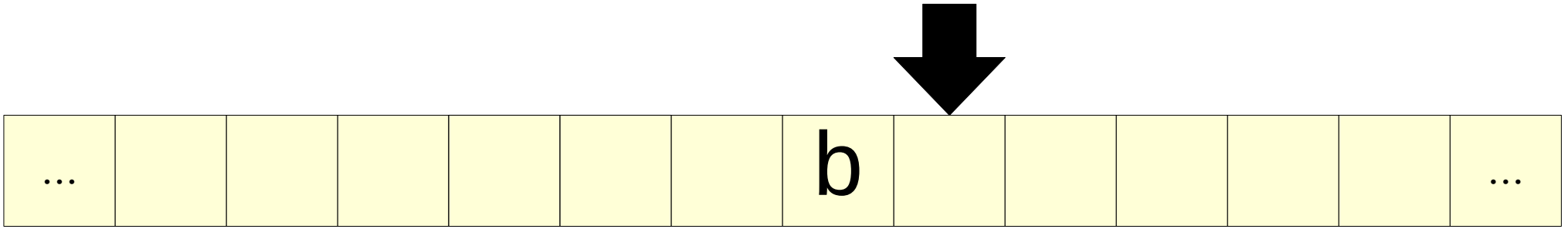
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



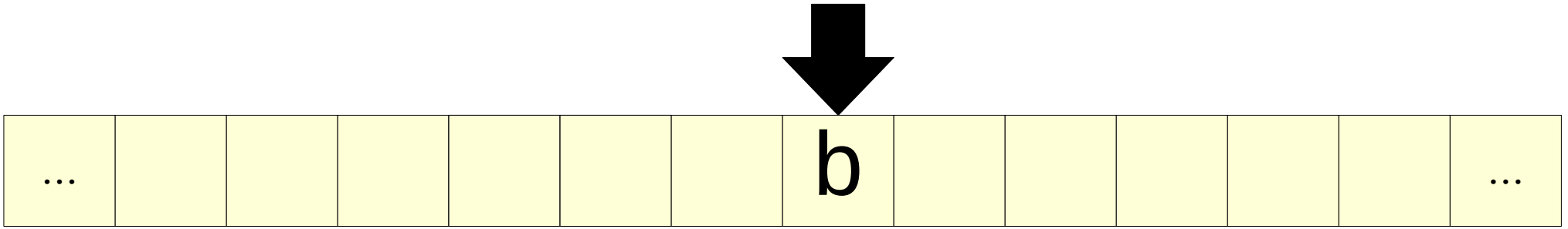
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



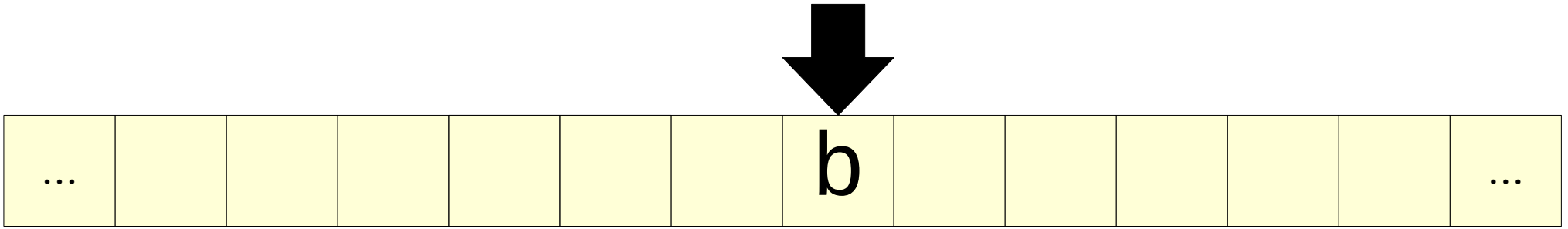
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



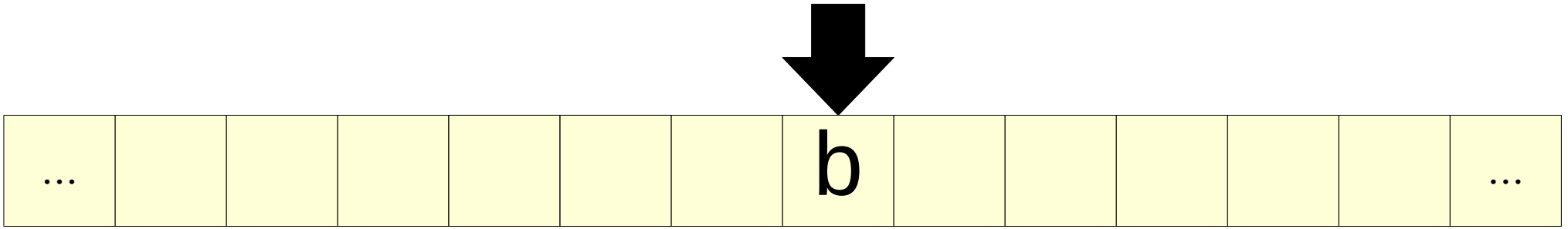
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



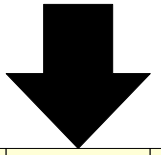
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```

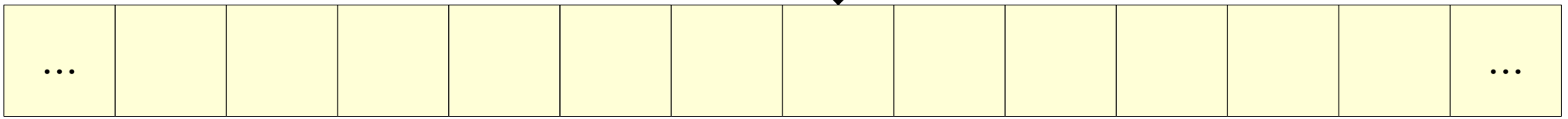


```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```

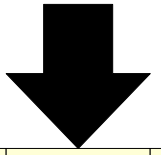


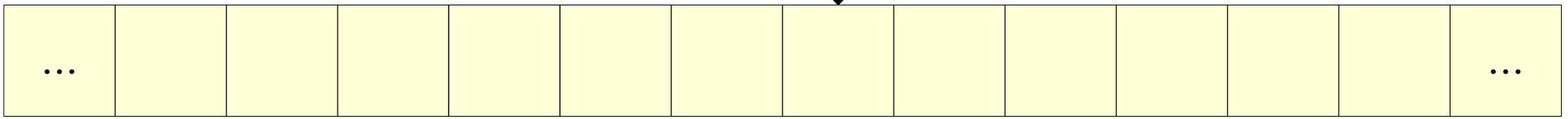
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



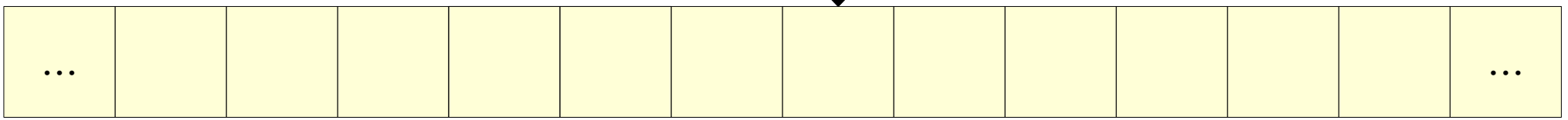


```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```

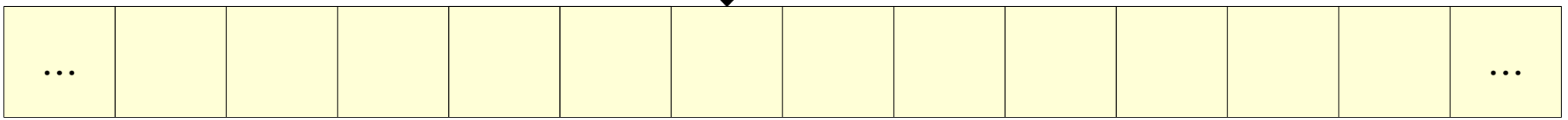




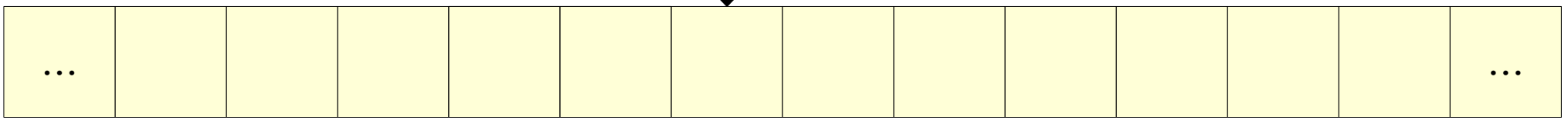
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



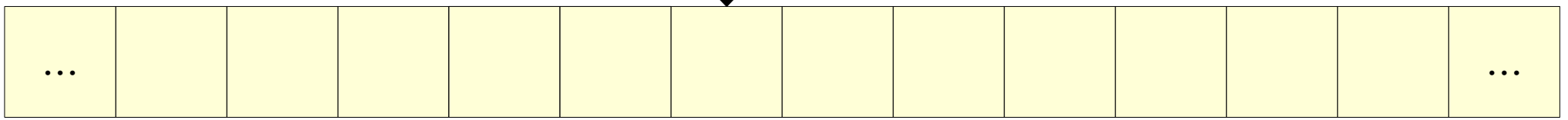
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



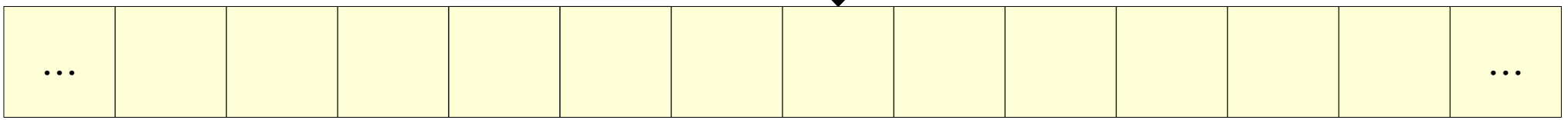
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



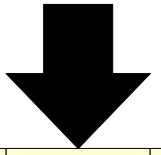
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```

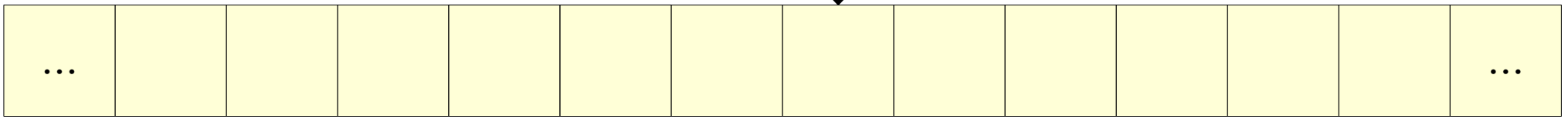


```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```

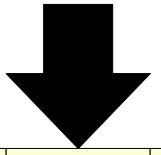


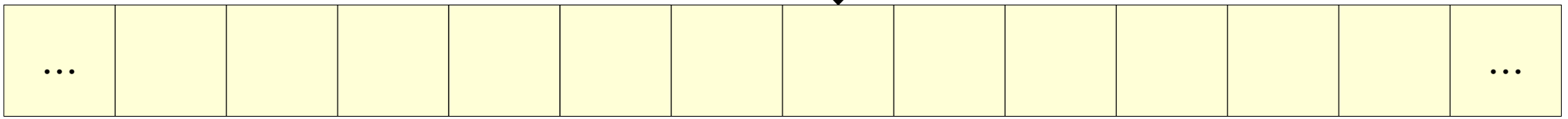
```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



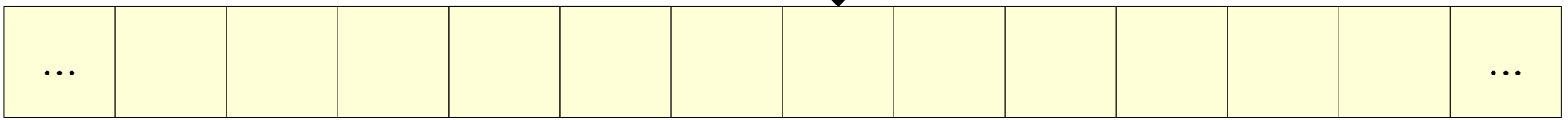


```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```

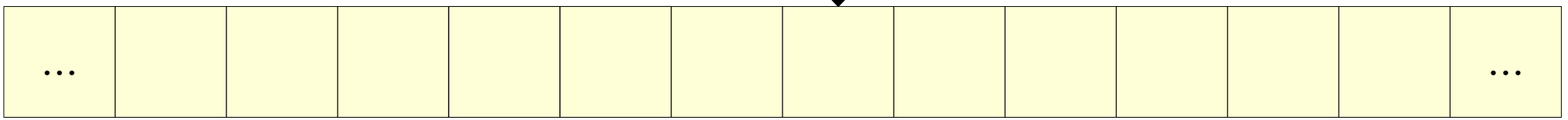




```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



```
Start:
   If Blank Return True
   If 'b' Return False
   Write Blank

ZipRight:
   Move Right
   If Not Blank Goto ZipRight
   Move Left
   If Not 'b' Return False
   Write Blank

ZipLeft:
   Move Left
   If Not Blank Goto ZipLeft
   Move Right
   Goto Start
```



Time-Out for Announcements!

Final Exam Logistics

- The final exam will take place in Hewlett 201 (regular lecture hall) from 7PM – 10PM on Saturday, 8/17.
- PS1-PS7, Lec 00-19.
- Same policy as Midterm.
  - Close AI/other humans, open everything else.
- One of our CAs, Anthony, is working on booking rooms for folks with exam accommodations.

Preparing for the Final Exam

- Highly recommend: go over your past assignments and the midterm and assess for any gaps.
- Use the practice finals and the practice problems to assess your understanding of the topics
- Use OHs and Ed to help you on those gaps.

Back to CS103!

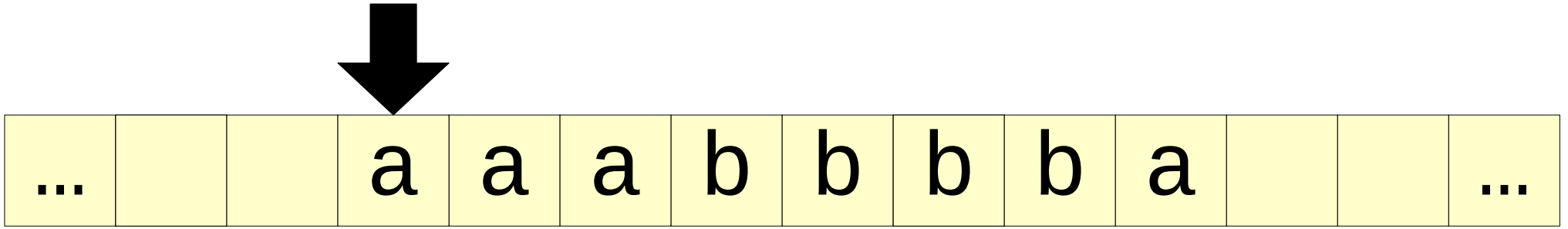
Our Next Challenge

- Let's now take aim at this more general language:

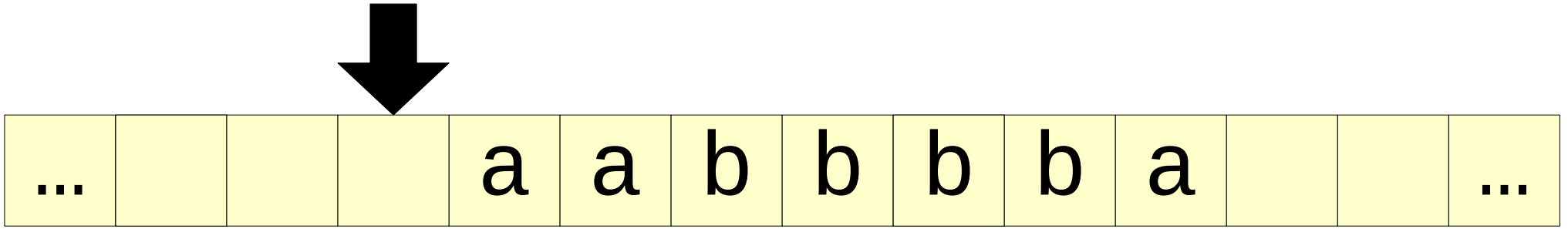
  $\{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w$ has an equal number of **a**'s and **b**'s $\}$

- This language is not regular (do you see why?)
- Let's see how to design a TM for it.

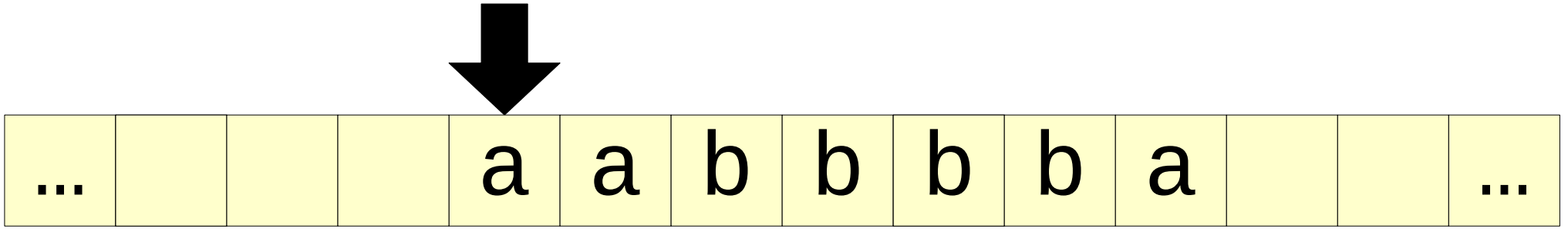
A Caveat



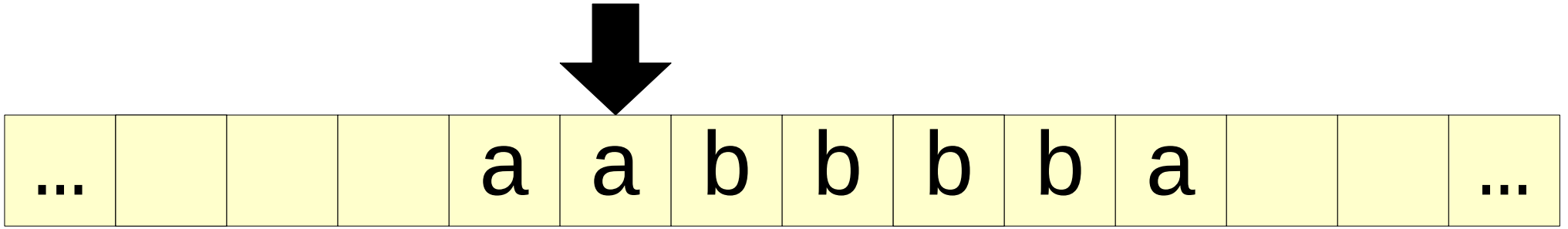
A Caveat



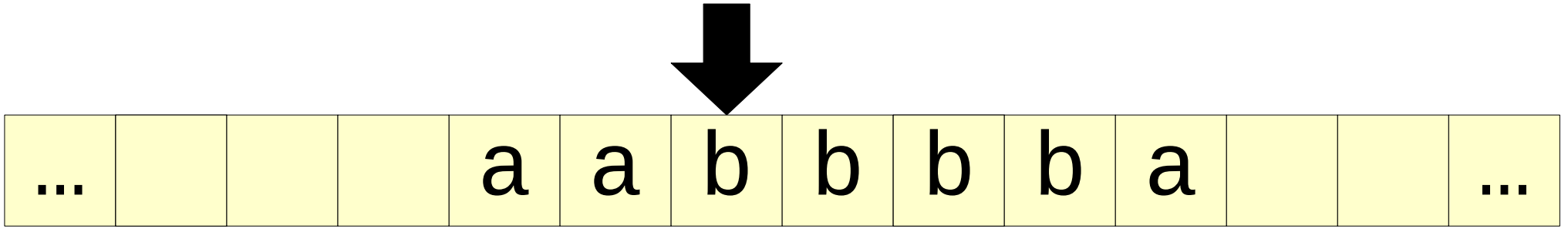
A Caveat



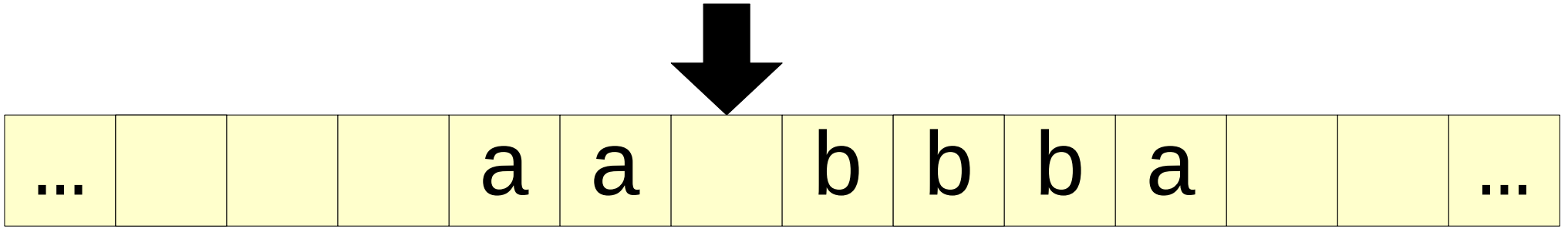
A Caveat



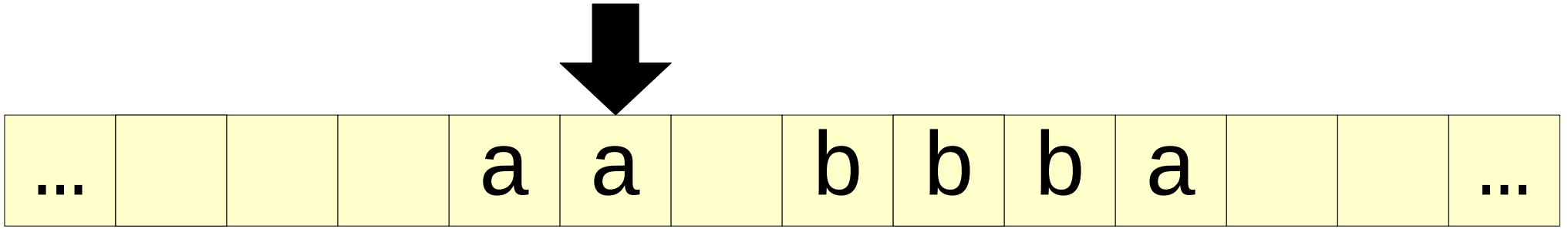
A Caveat



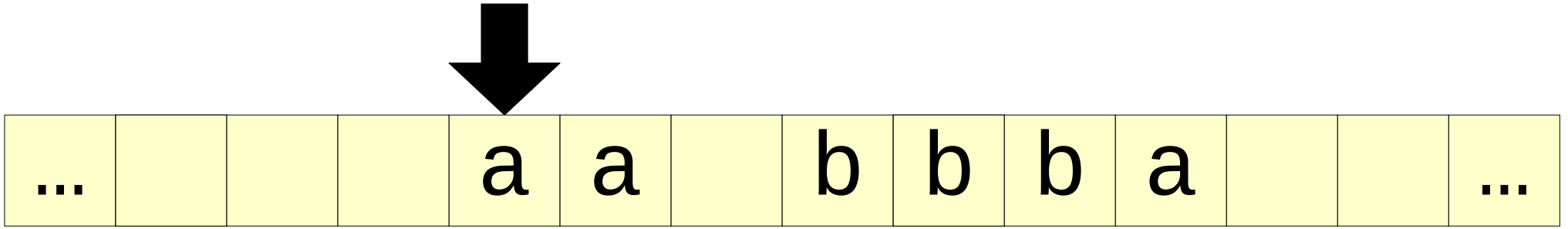
A Caveat



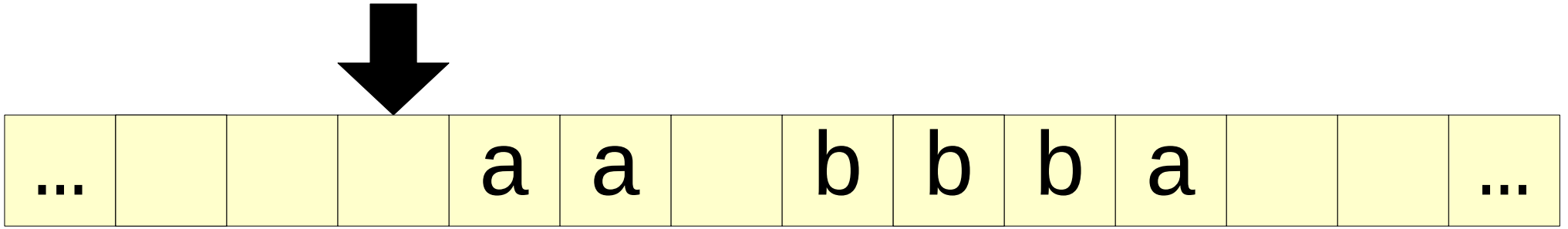
A Caveat



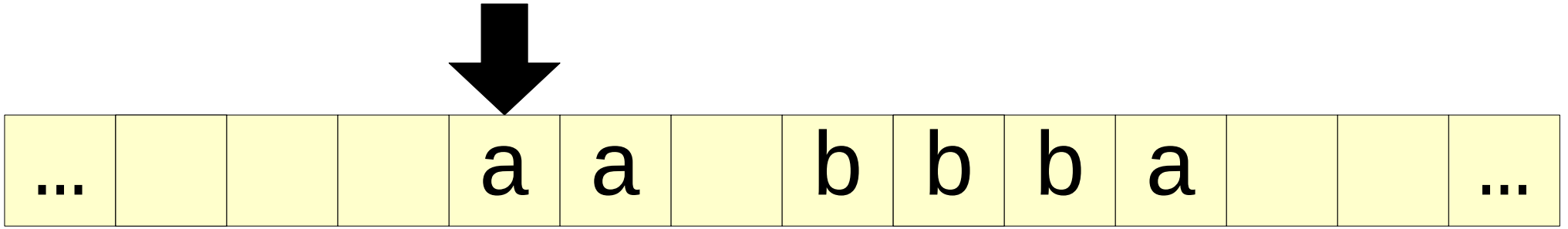
A Caveat



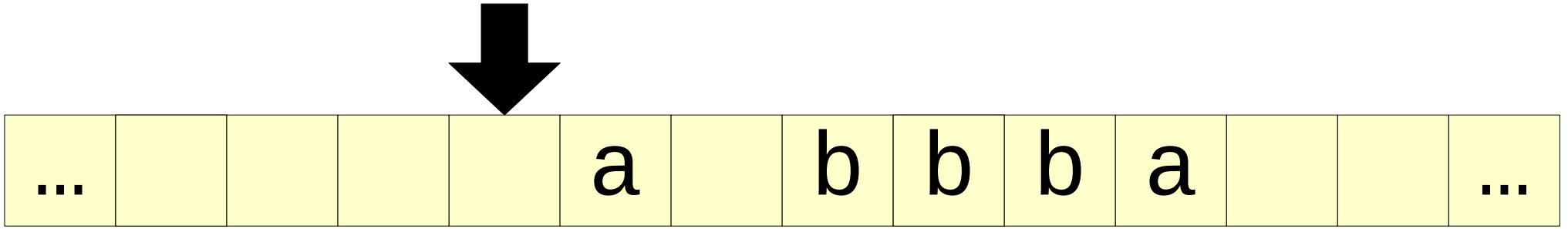
A Caveat



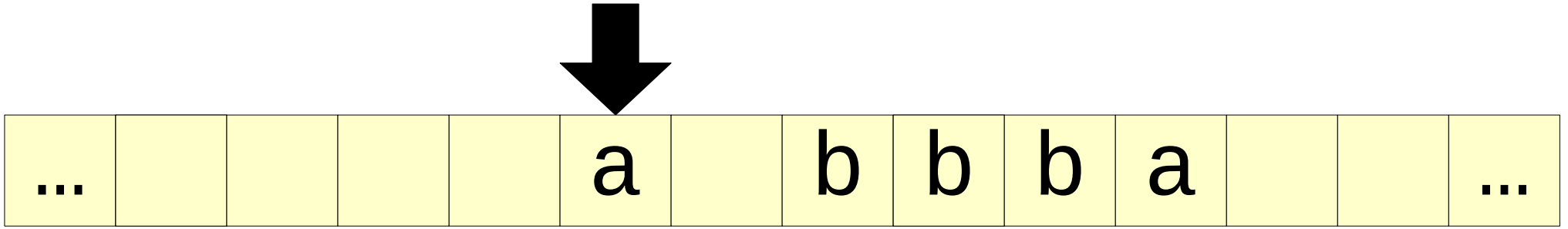
A Caveat



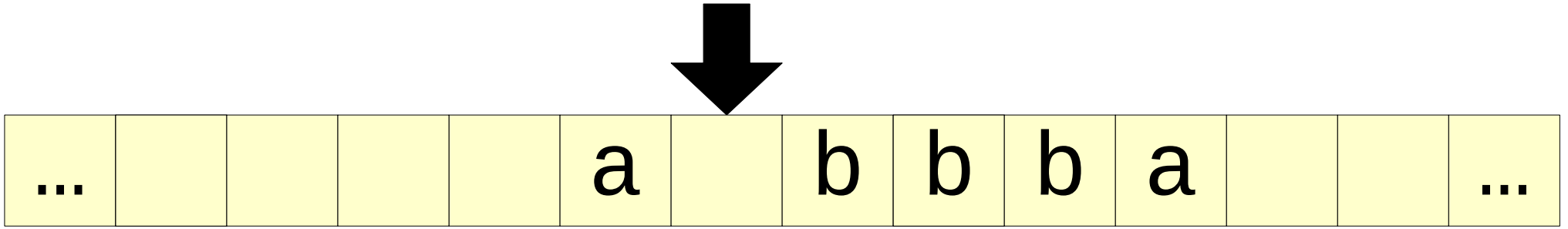
A Caveat



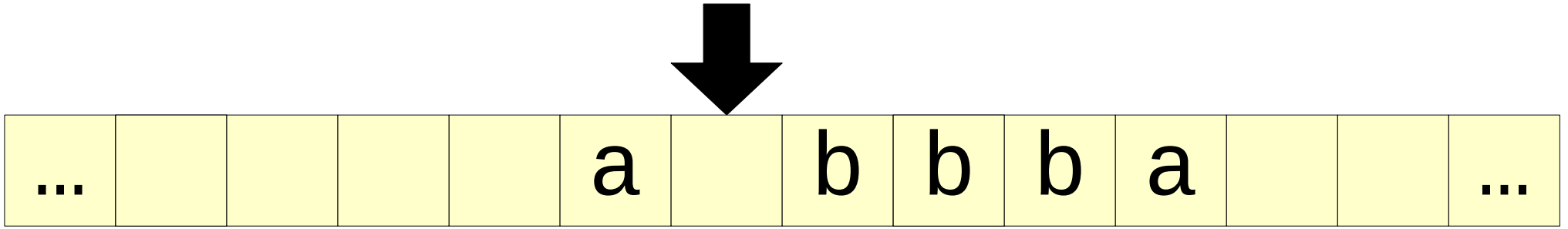
A Caveat



A Caveat

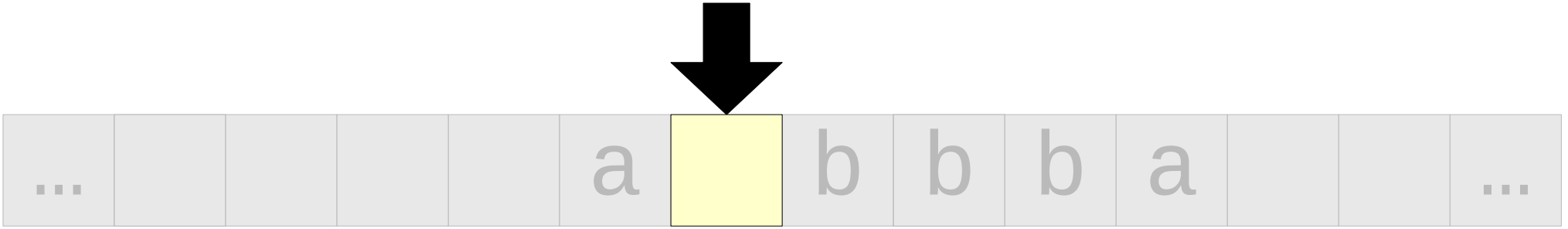


A Caveat



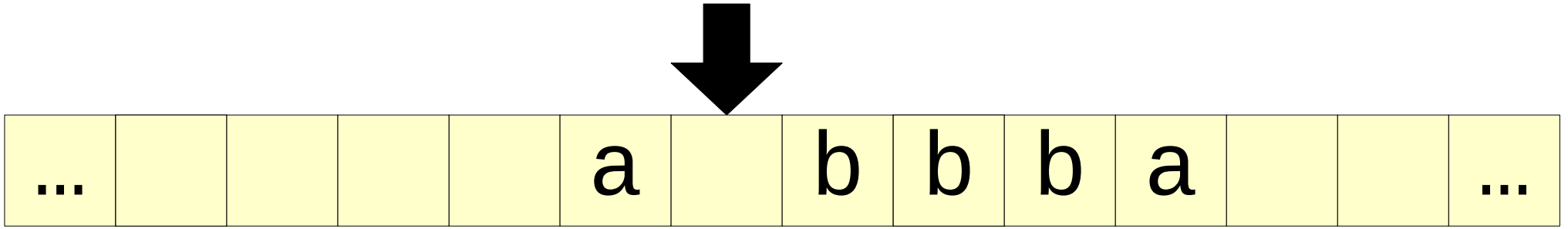
How do we know that this blank isn't one of the infinitely many blanks after our input string?

A Caveat



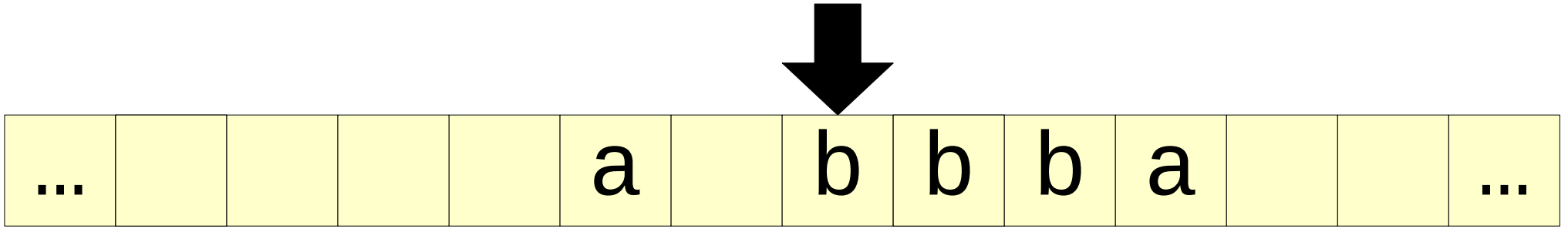
How do we know that this blank isn't one of the infinitely many blanks after our input string?

A Caveat

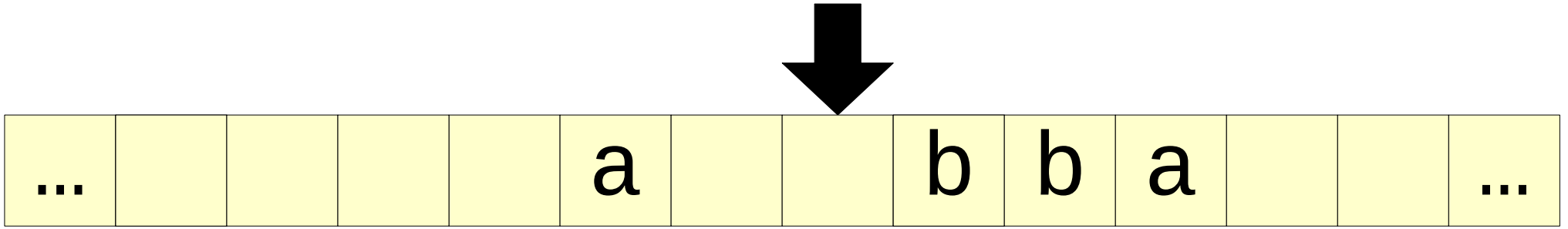


How do we know that this blank isn't one of the infinitely many blanks after our input string?

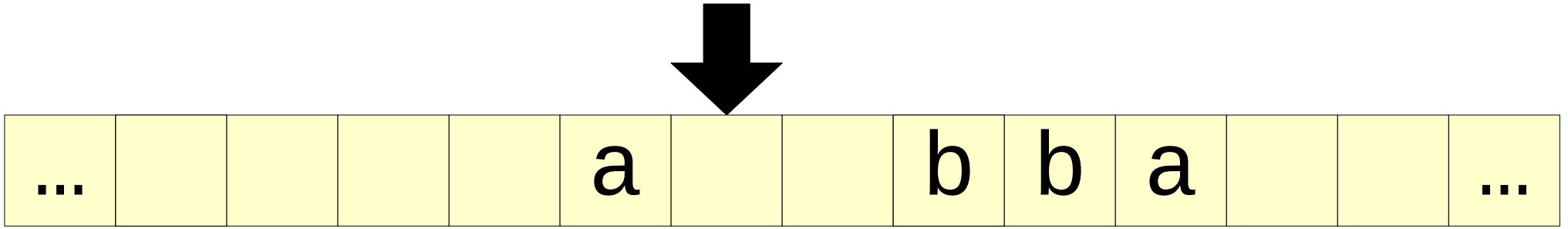
A Caveat



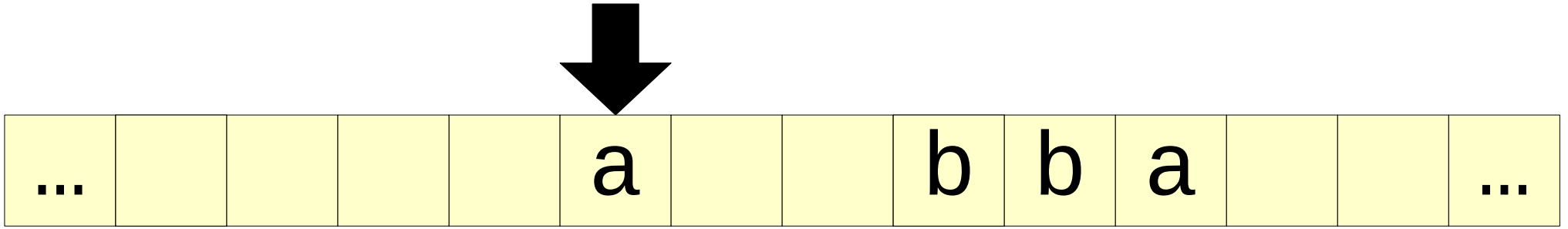
A Caveat



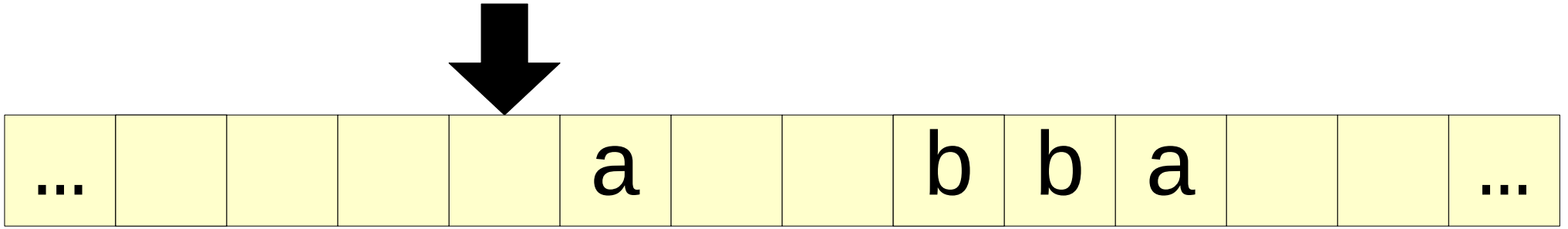
A Caveat



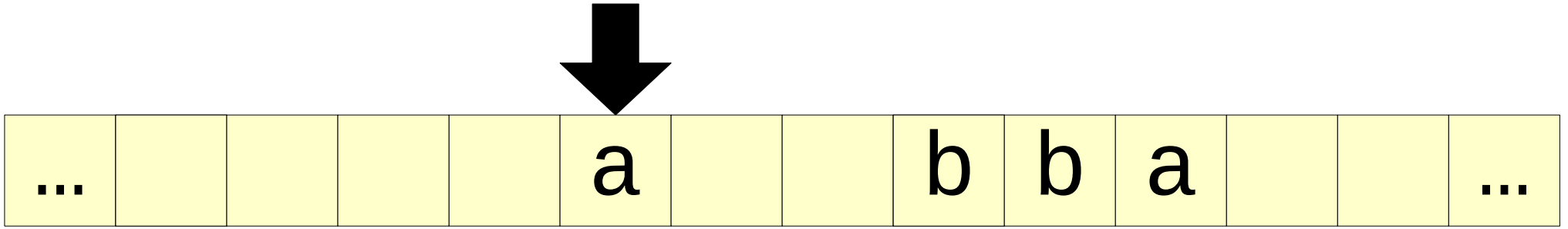
A Caveat



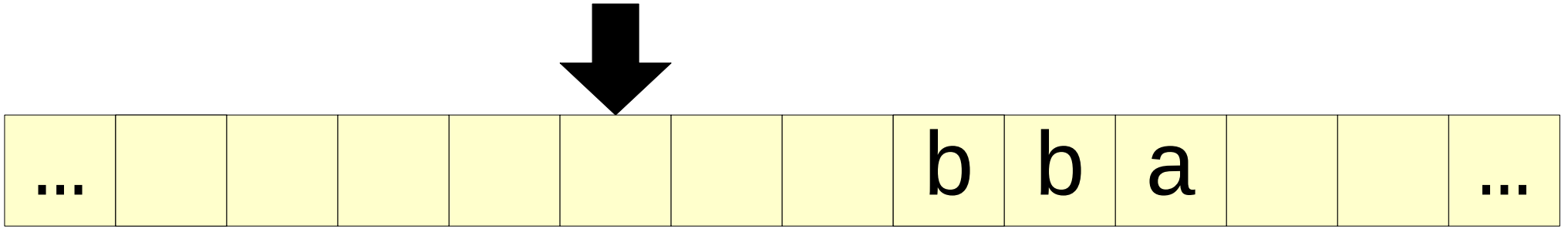
A Caveat



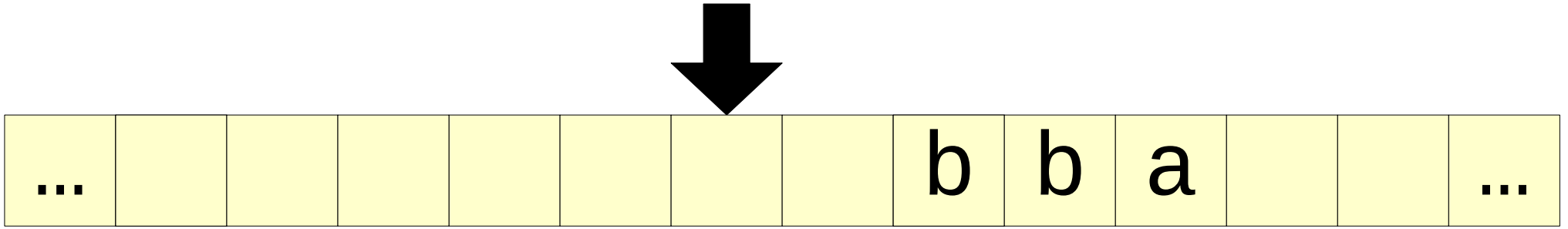
A Caveat



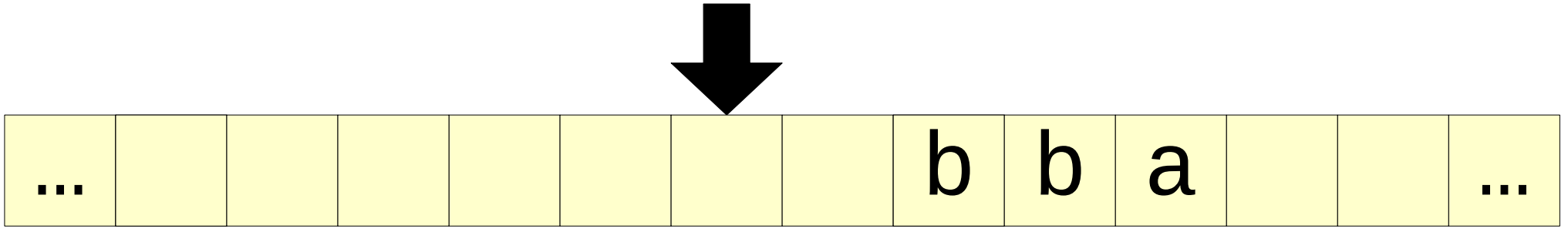
A Caveat



A Caveat

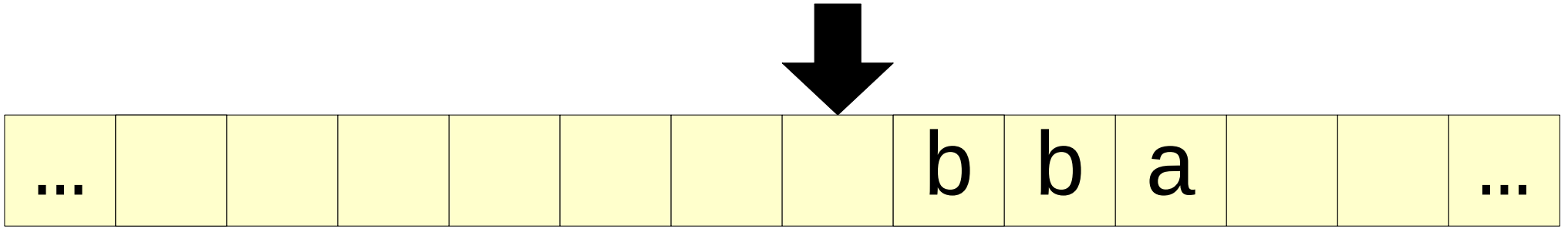


A Caveat

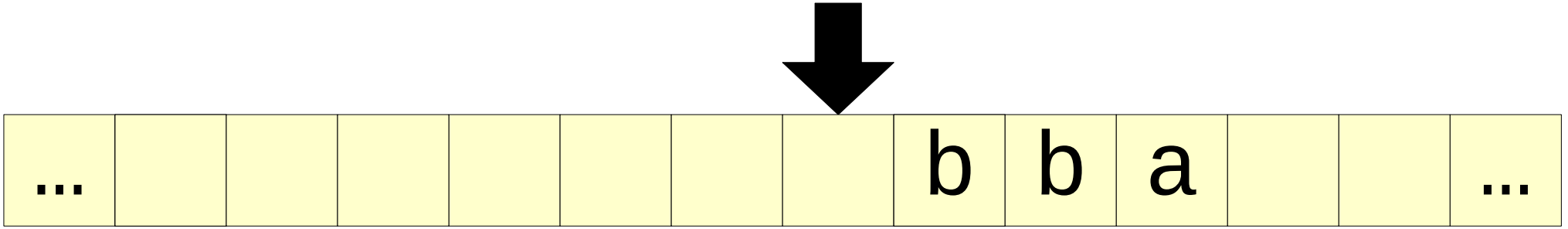


How do we know that this blank isn't one of the infinitely many blanks after our input string?

A Caveat

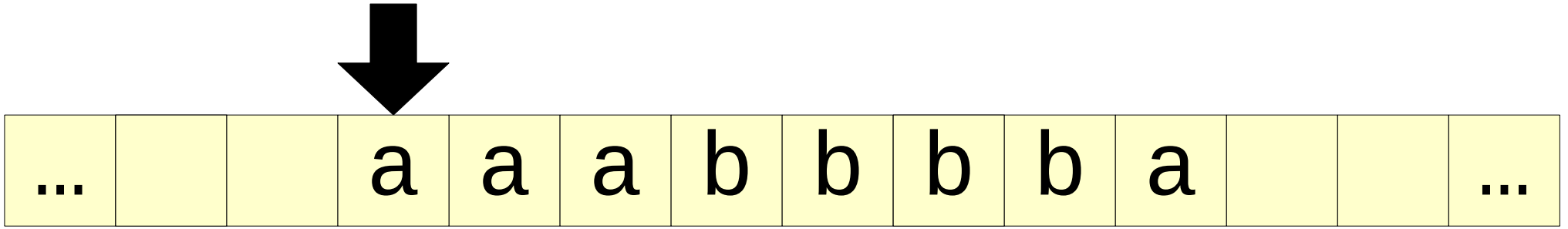


A Caveat

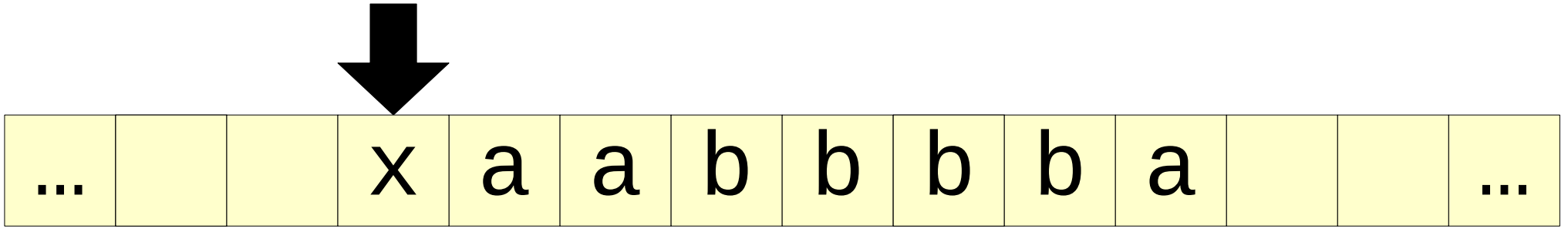


How do we know that this blank isn't one of the infinitely many blanks after our input string?

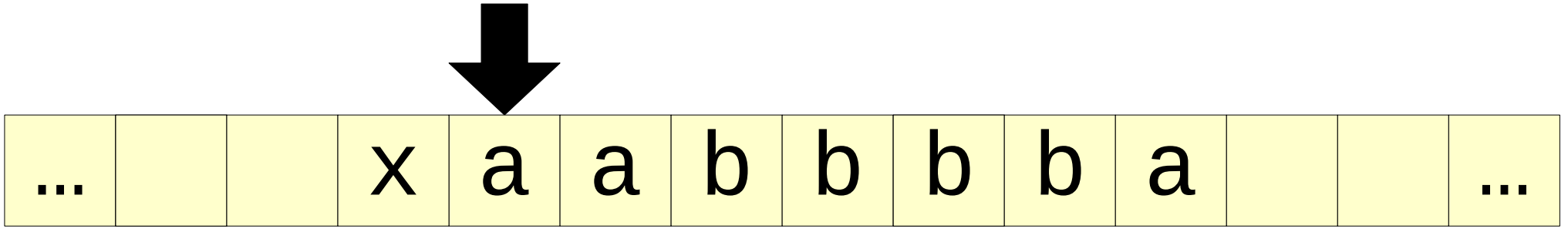
One Solution



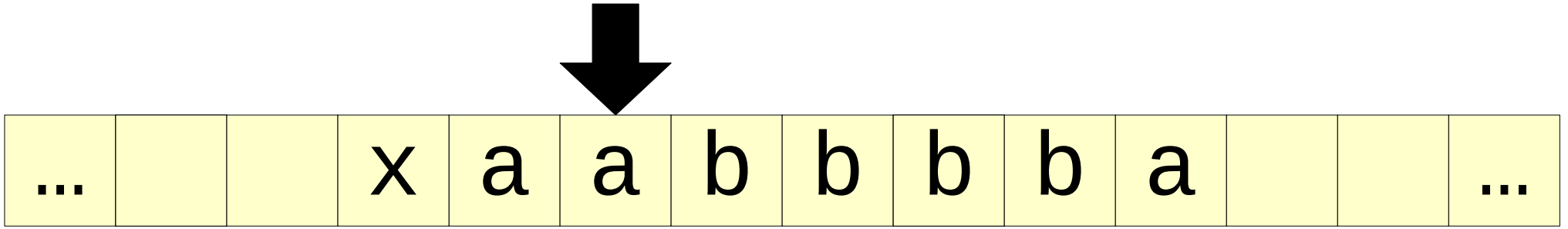
One Solution



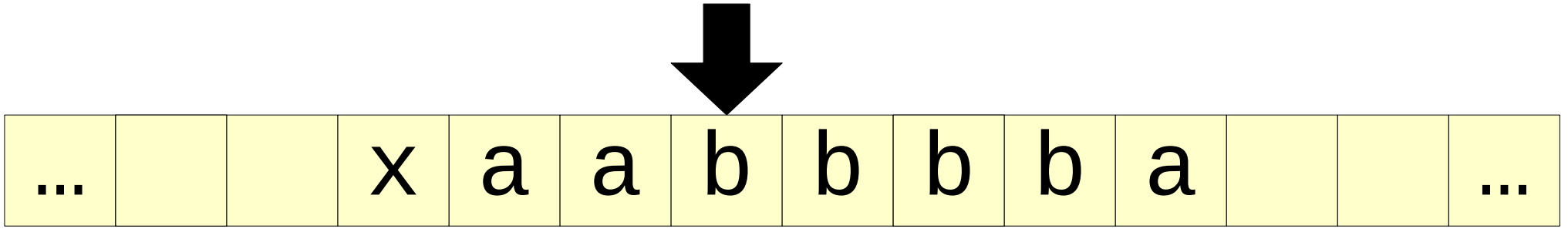
One Solution



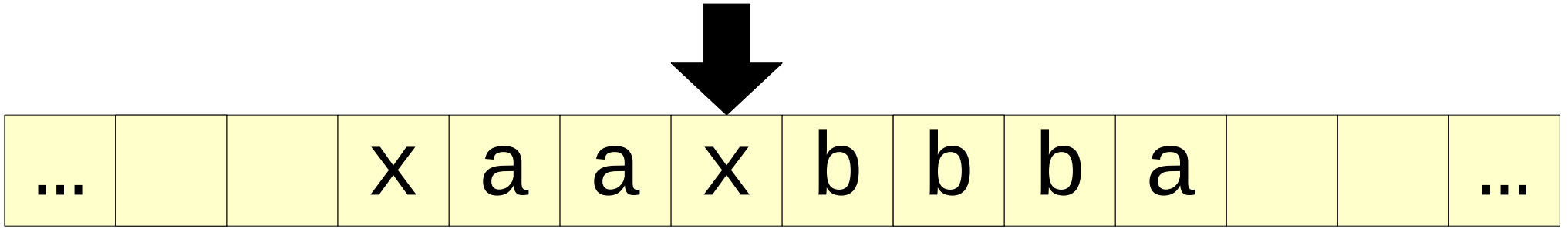
One Solution



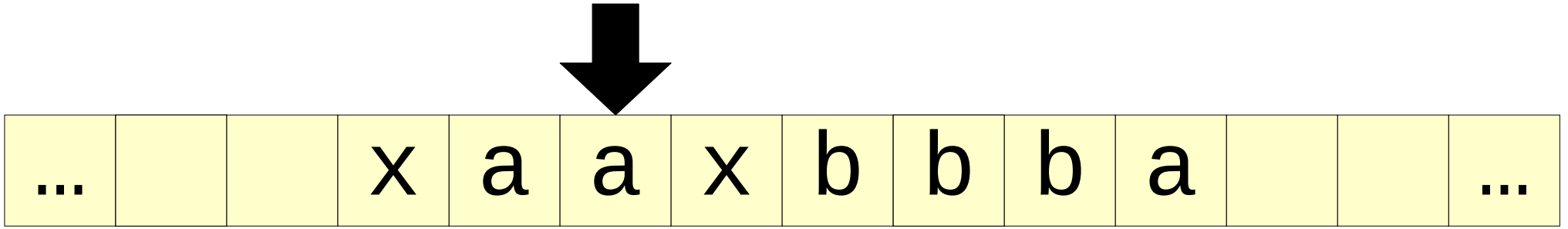
One Solution



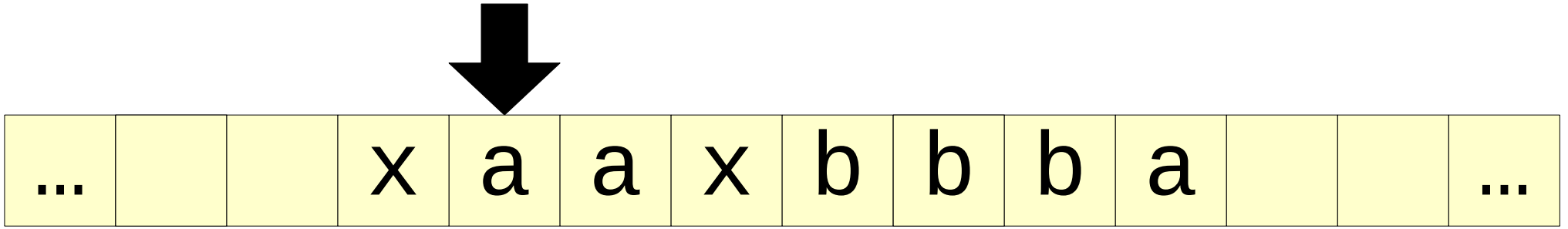
One Solution



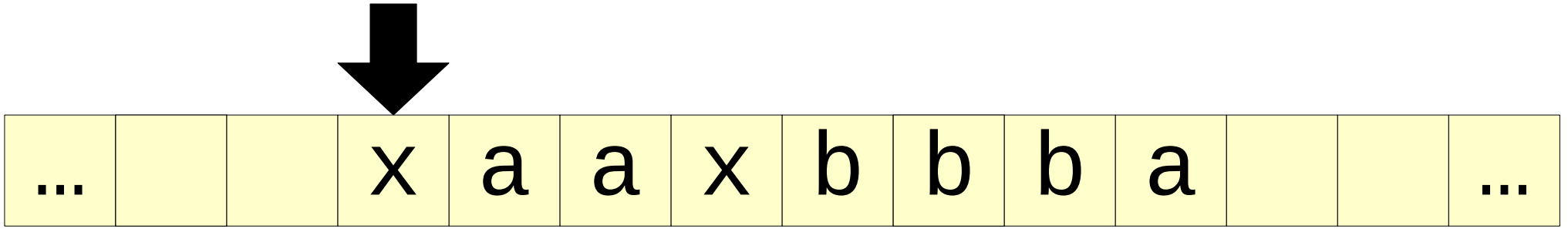
One Solution



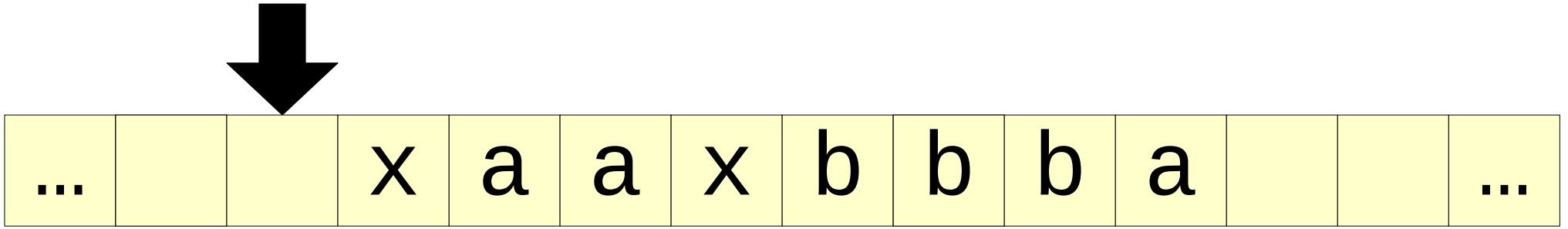
One Solution



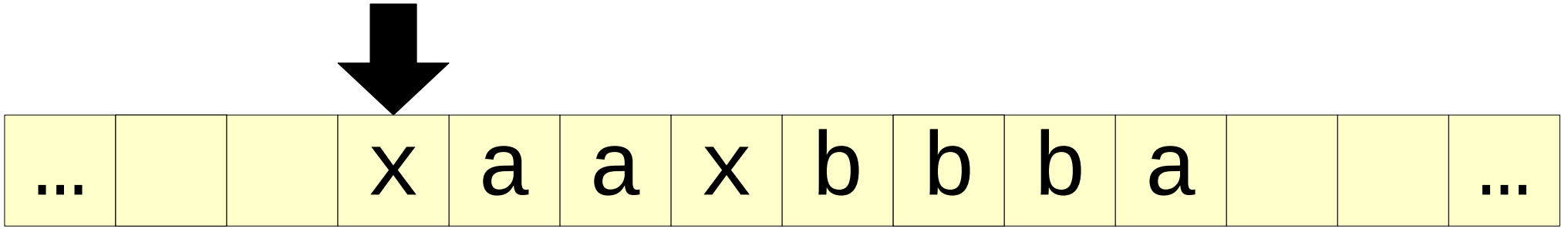
One Solution



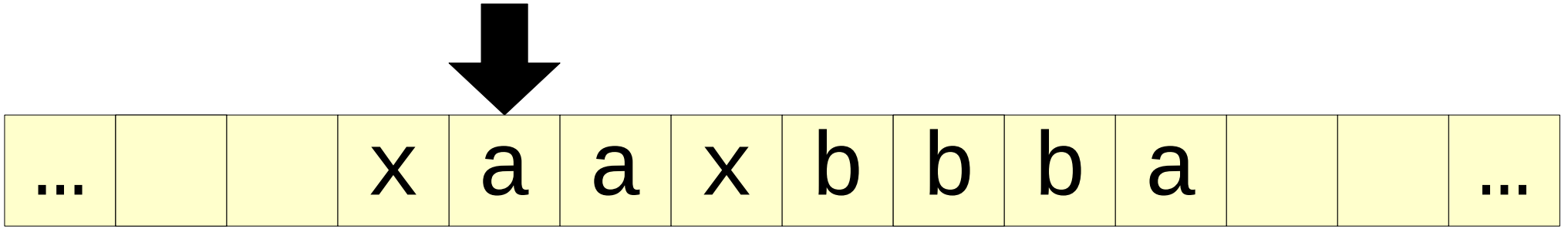
One Solution



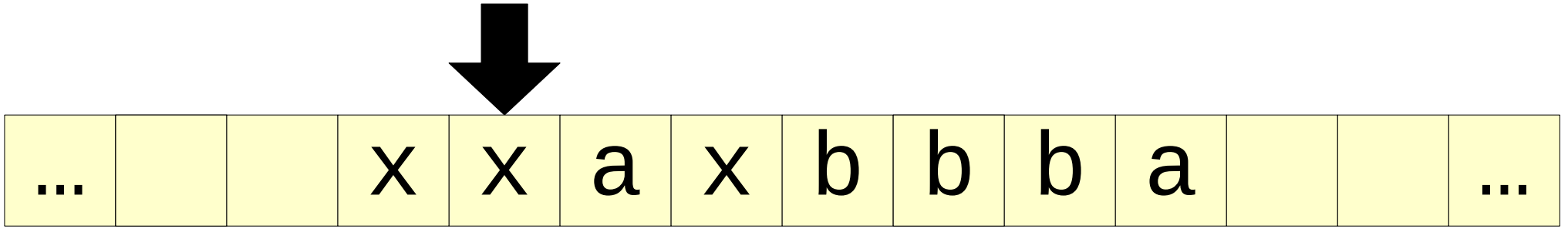
One Solution



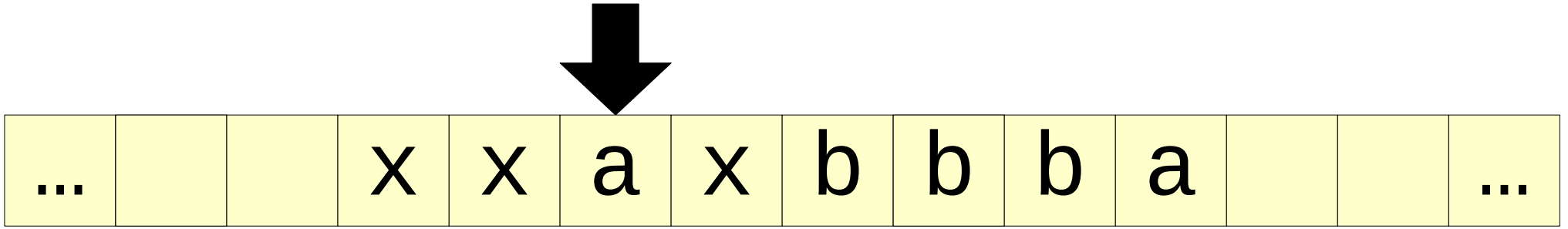
One Solution



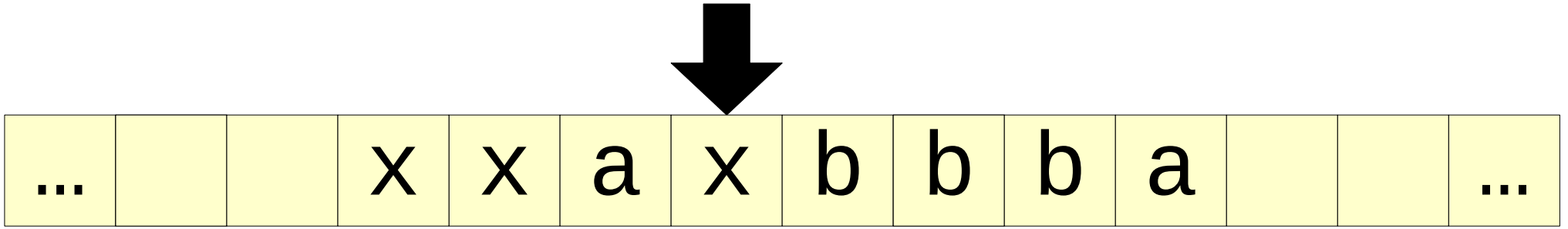
One Solution



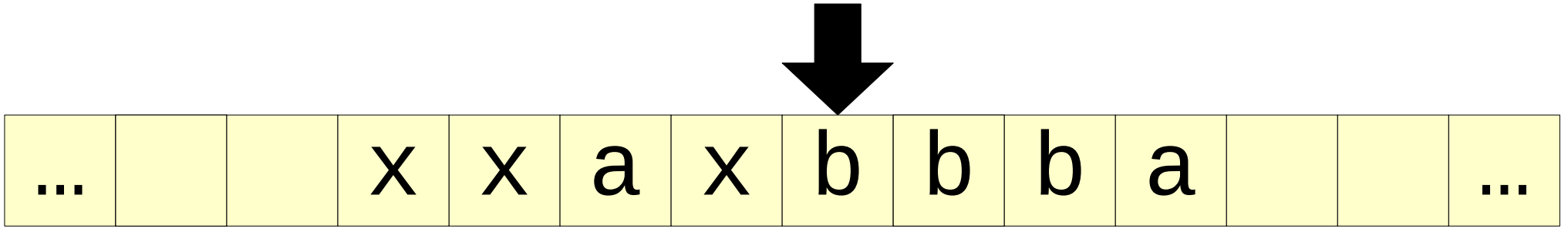
One Solution



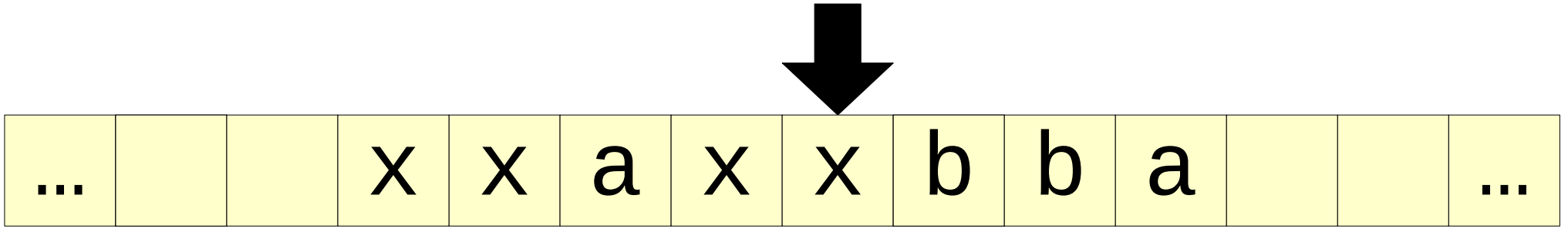
One Solution



One Solution



One Solution



Let’s take a look at this in action!

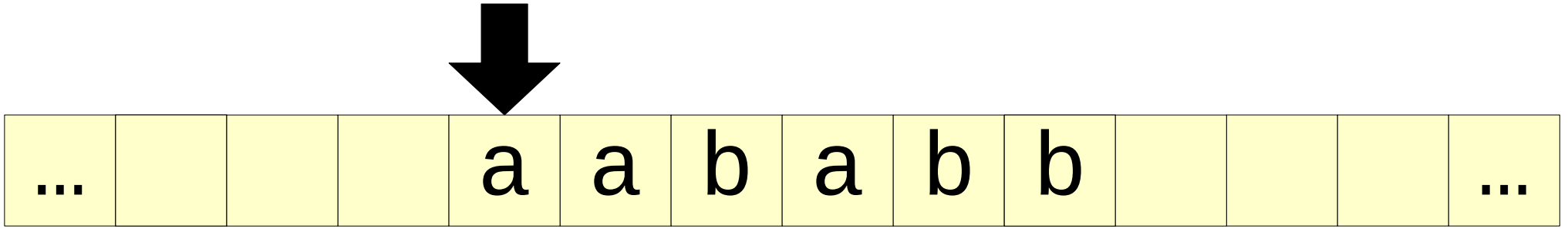
Another Idea

- We just built a TM for the language

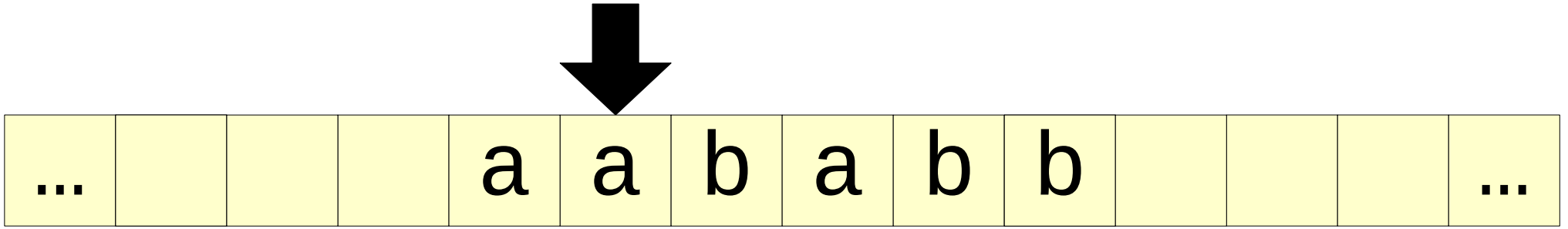
  $\{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w$ has the same number of **a**'s and **b**'s $\}$.

- An observation: this would be a *lot* easier to test for if all the **a**'s came before all the **b**'s.
  - In fact, that would turn this into checking if the string has the form $\mathbf{a}^n\mathbf{b}^n$, which we already know how to do!
- ***Idea:*** Could we sort the characters of our input string?

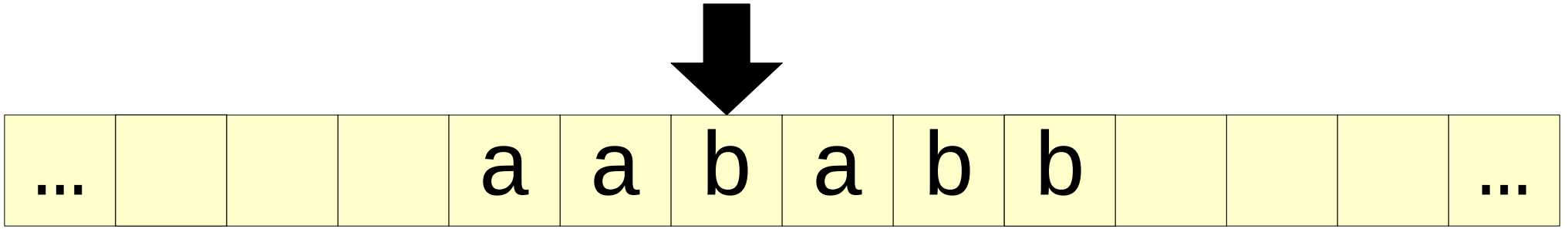
The Idea



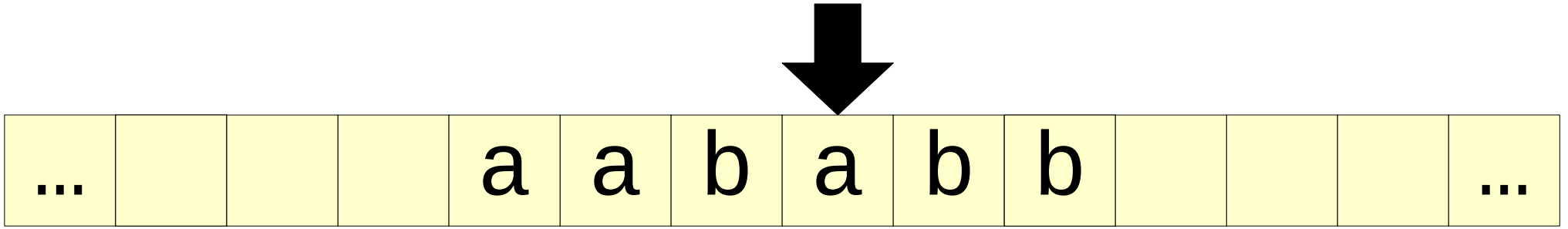
The Idea



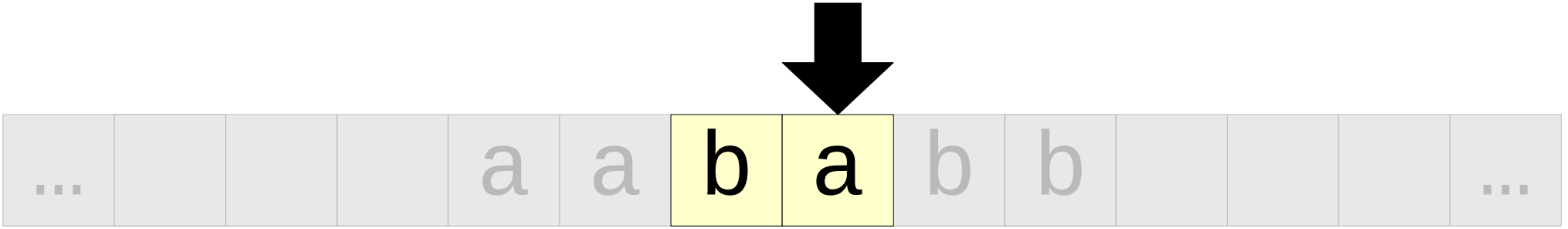
The Idea



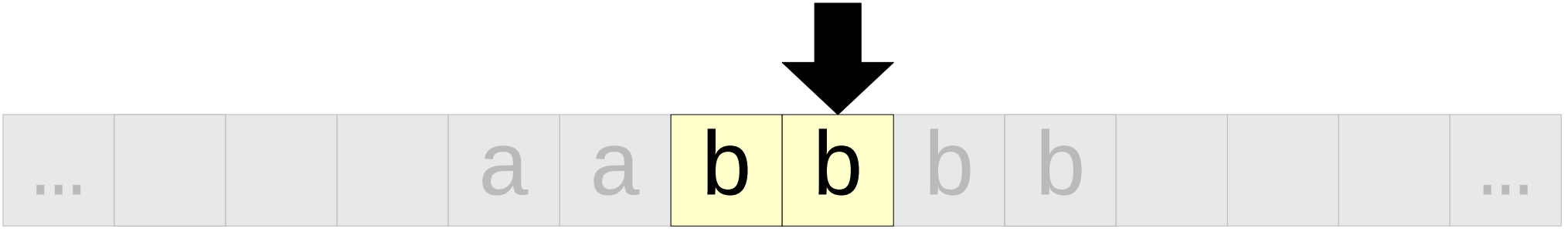
The Idea



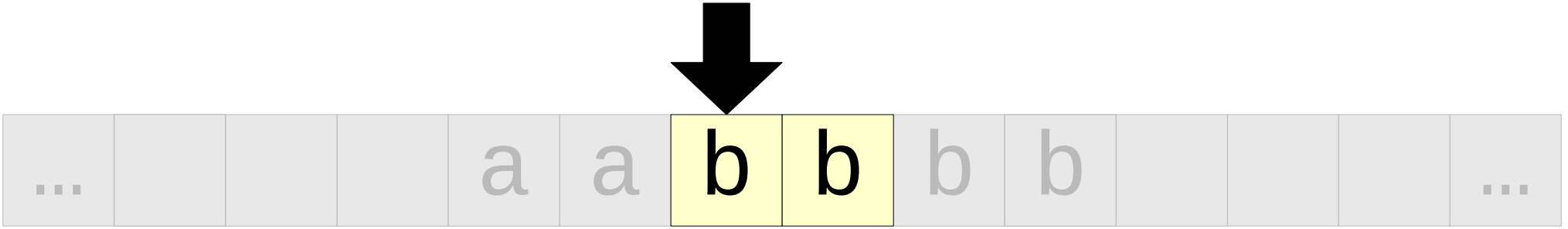
The Idea



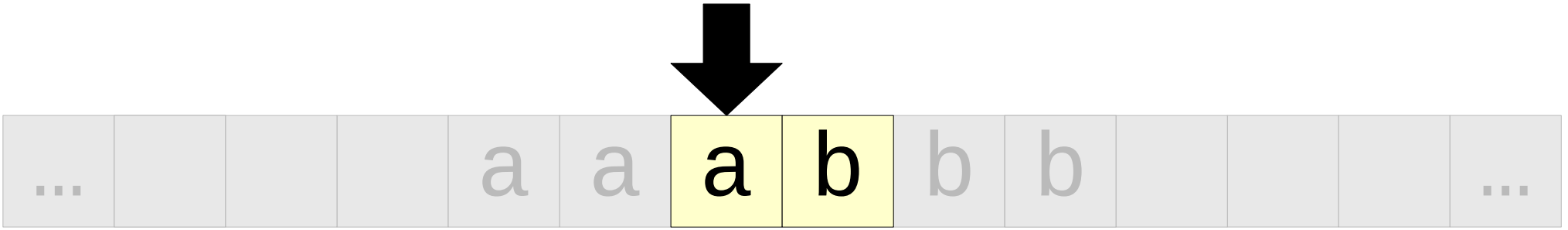
The Idea



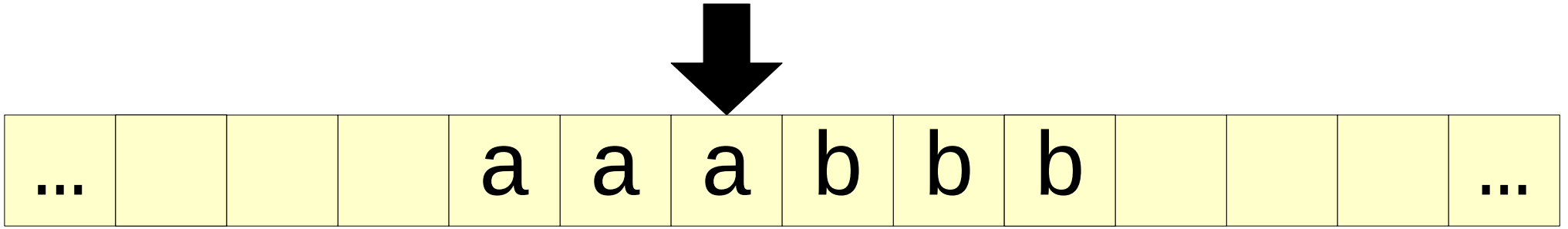
The Idea



The Idea



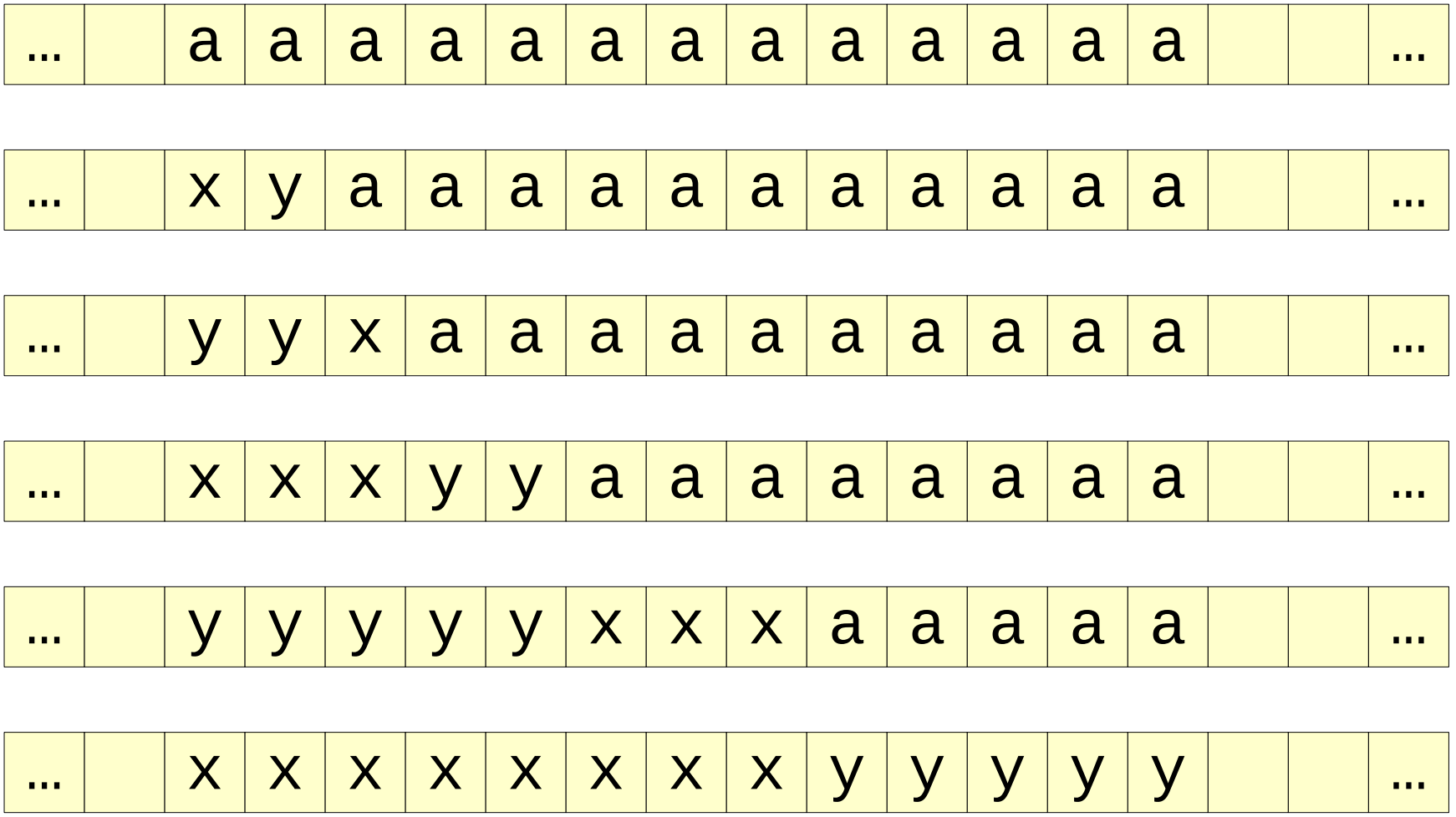
The Idea



Exploring This Idea

Cool TM Tricks 1: ***Fibonacci Numbers***

Fibonacci Numbers

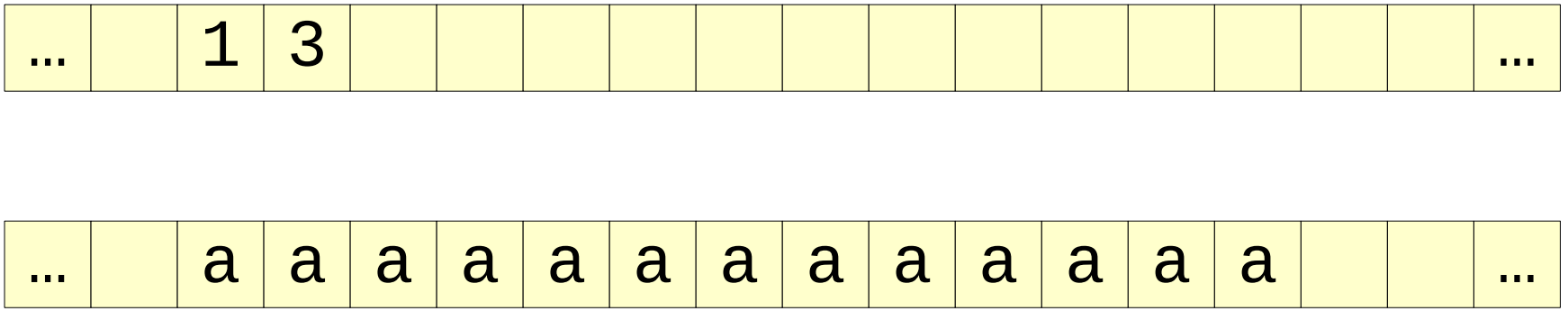


{ $\mathbf{a}^n$ | n is a Fibonacci number }

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Cool TM Tricks 2: ***Decimal Fibonacci***

Decimal Fibonacci



{ $w \in \{$**0**, **1**, **2**, … **9**$\}^*$ | w, interpreted as a base-10 number, is a Fibonacci number. }

Just how powerful are Turing machines?

Real and “Ideal” Computers

- A real computer has memory limitations: you have a finite amount of RAM, a finite amount of disk space, etc.
- However, as computers get more and more powerful, the amount of memory available keeps increasing.
- An ***idealized computer*** is like a regular computer, but with unlimited RAM and disk space. It functions just like a regular computer, but never runs out of memory.

Theorem: Turing machines are equal in power to idealized computers. That is, any computation that can be done on a TM can be done on an idealized computer and vice-versa.

Key Idea: Two models of computation are equally powerful if they can simulate each other.

Simulating a TM

- The individual commands in a TM are simple and perform only basic operations:

```
Move   Write   Goto   Return   If
```

- The memory for a TM can be thought of as a string with some number keeping track of the current index.
- To simulate a TM, we need to
  - see which line of the program we're on,
  - determine what command it is, and
  - simulate that single command.
- ***Claim:*** This is reasonably straightforward to do on an idealized computer.
  - The "core" logic for the TM simulator is under fifty lines of code, including comments.

Simulating a TM

- Because a computer can simulate each individual TM instruction, a computer can do anything a TM can do.
- ***Key Idea:*** Even the most complicated TM is made out of individual instructions, and if we can simulate those instructions, we can simulate an arbitrarily complicated TM.

Simulating a Computer

- Programming languages provide a set of simple constructs.
  - Think things like variables, arrays, loops, functions, classes, etc.
- You, the programmer, then combine these basic constructs together to assemble larger programs.
- ***Key Idea:*** If a TM is powerful enough to simulate each of these individual pieces, it's powerful enough to simulate anything a real computer can do.

What We've Seen

- We've seen TMs use loops to solve problems.
  - Our $\{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$ TM repeatedly pulls off the first and last character from the string.
  - Our sorting TM repeatedly finds **ba** and replaces it with **ab**.
- In some sense, the existence of `Goto` and labels means that TMs have loops.
- Hopefully, it's not too much of a stretch to think that TMs can do `while` loops, `for` loops, etc.

What We've Seen

- We've seen TMs that perform basic arithmetic.
  - We can check if two numbers are equal.
  - We can check if a number is a Fibonacci number.
- Hopefully, it's not too much of a stretch to believe we could also do addition and subtraction, compute powers of numbers, do ceilings and floors, etc.

What We've Seen

- We've seen TMs that maintain variables.
  - You can think of our TM for $\{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$ as storing two variables – one that counts a number of **a**'s, and one that counts a number of **b**'s.
  - Our TM for Fibonacci numbers kinda sorta ish tracks the last two Fibonacci numbers, plus the length of the input string.
- It's a bit larger of a jump to make, but hopefully you're comfortable with the idea that TMs, in principle, can maintain variables.

What We've Seen

- We've seen TMs with helper functions.
  - We saw how to check for equal numbers of **a**'s and **b**'s by first sorting the string, then checking of the string has the form $\mathbf{a}^n\mathbf{b}^n$.
  - We can check if a decimal number is a Fibonacci number by converting it to unary, then running our unary Fibonacci checker.
- Hopefully you're comfortable with the idea that a TM could have multiple "helper functions" that work together to solve some larger problem.

What Else Can TMs Do?

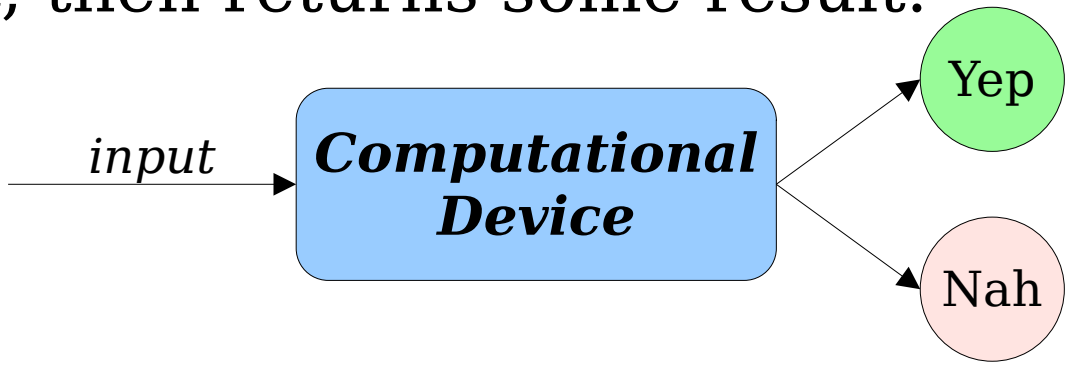
- Maintain strings and arrays.
  - Store their elements separated with some special separator character.
- Support pointers.
  - Maintain an array of what's in memory, where each item is tagged with its "memory address."
- Support function call and return.
  - It's hard, but you can do this if you can do helper functions and variables.

A CS107 Perspective

- Internally, computers execute by using basic operations like
  - simple arithmetic,
  - memory reads and writes,
  - branches and jumps,
  - register operations,
  - etc.
- Each of these are simple enough that they could be simulated by a Turing machine.

A Leap of Faith

- ***Claim:*** A TM is powerful enough to simulate any computer program that gets an input, processes that input, then returns some result.



- The resulting TM might be colossal, or really slow, or both, but it would still faithfully simulate the computer.
- We're going to take this as an article of faith in CS103. If you curious for more details, come talk to me after class.

Can a TM Work With…

“cat pictures?”

Sure! A picture is just a 2D array of colors, and a color can be represented as a series of numbers.

Can a TM Work With…

~~“cat pictures?”~~

“cat videos?”

If you think about it, a video is just a series of pictures!

Can a TM Work With…

“music?”

Sure! Music is encoded as a compressed waveform. That’s just a list of numbers.

“deep learning?”

Sure! That’s just applying a bunch of matrices and nonlinear functions to some input.

Just how powerful ***are*** Turing machines?

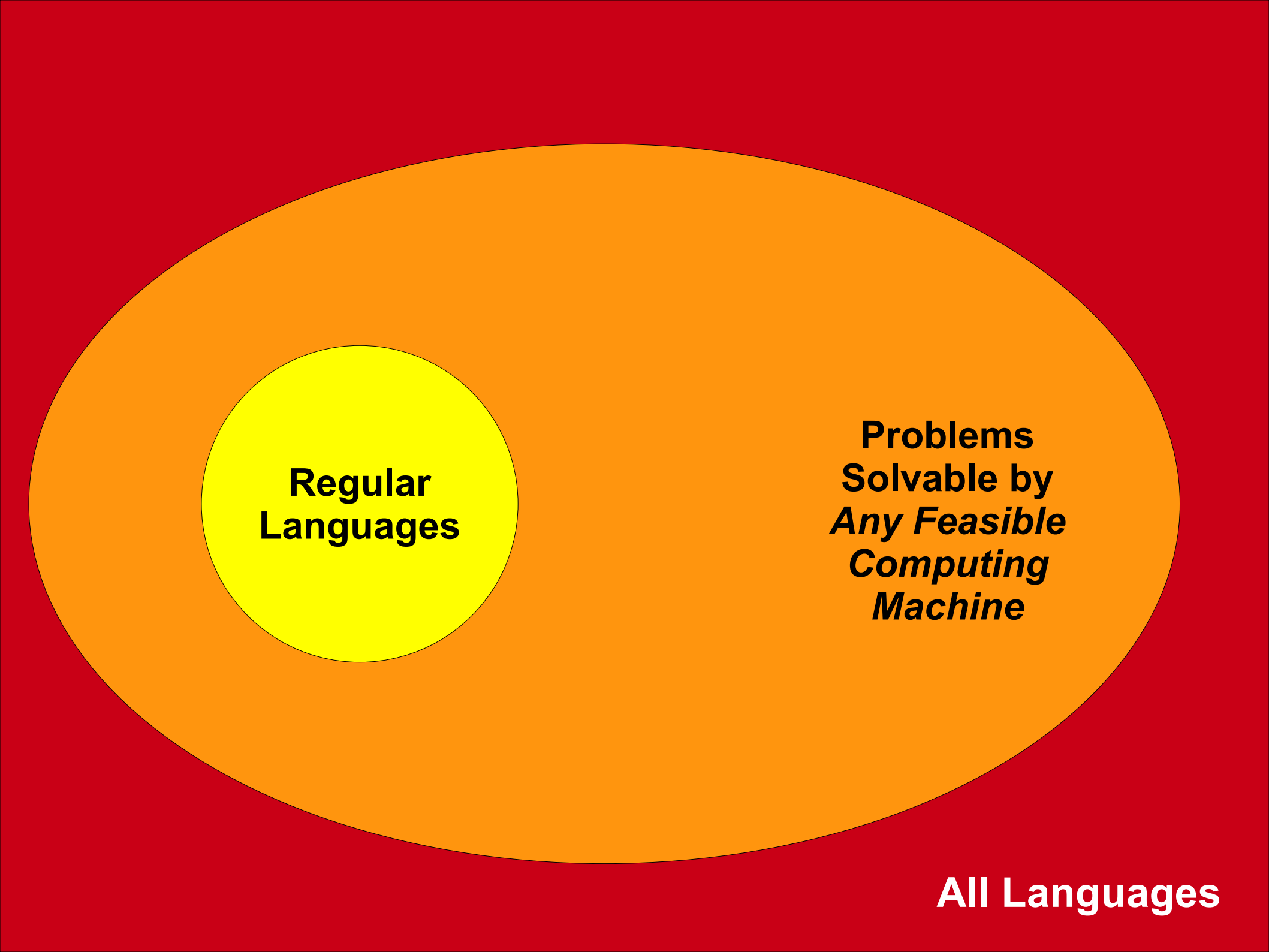
Effective Computation

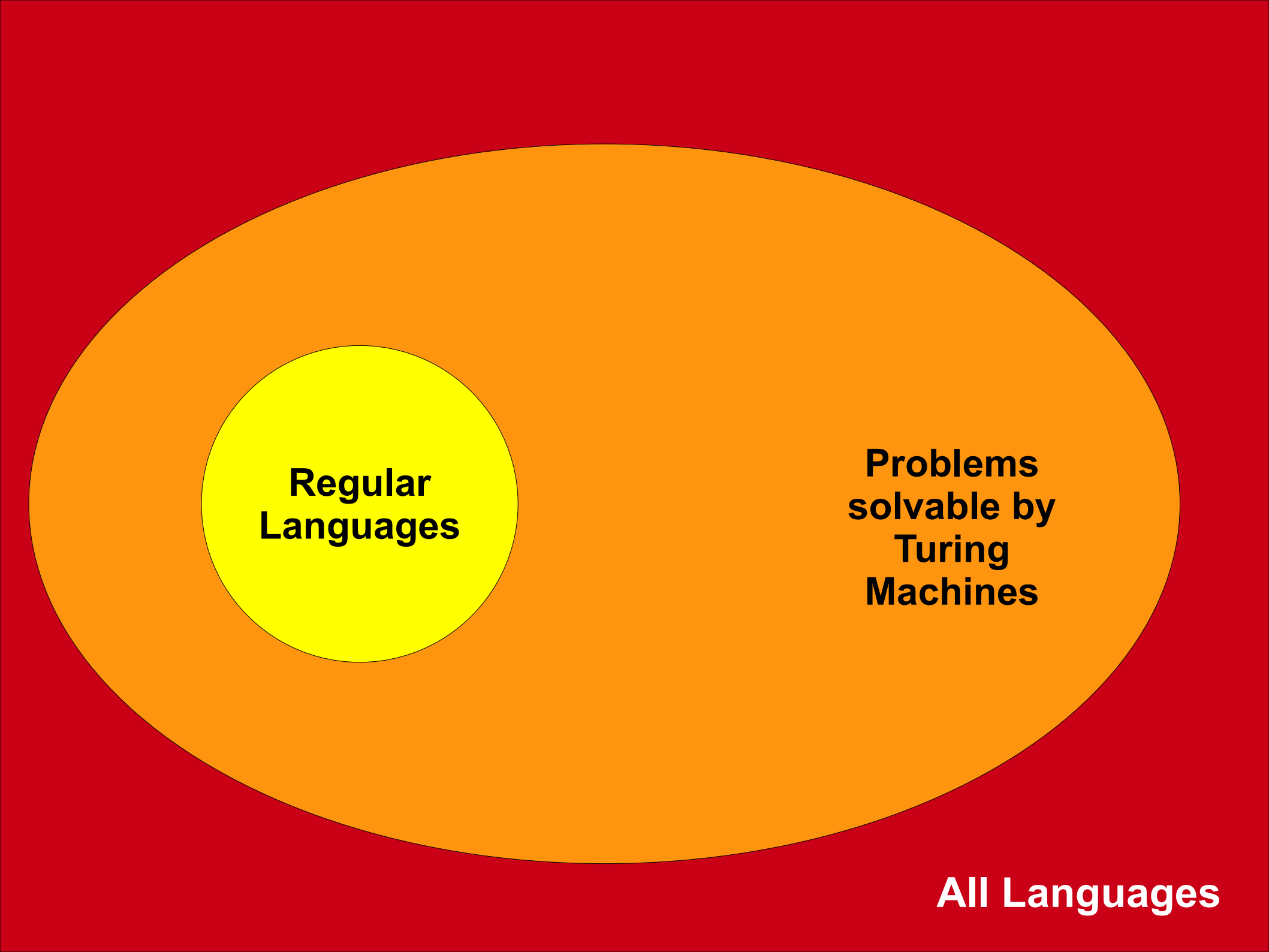
- An ***effective method of computation*** is a form of computation with the following properties:
  - The computation consists of a set of steps.
  - There are fixed rules governing how one step leads to the next.
  - Any computation that yields an answer does so in finitely many steps.
  - Any computation that yields an answer always yields the correct answer.
- This is not a formal definition. Rather, it's a set of properties we expect out of a computational system.

The ***Church-Turing Thesis*** claims that

every effective method of computation is either equivalent to or weaker than a Turing machine.

> “This is not a theorem – it is a falsifiable scientific hypothesis. And it has been thoroughly tested!”
>
> - Ryan Williams


Regular Languages
Problems Solvable by *Any Feasible Computing Machine*
All Languages


Regular Languages
Problems solvable by Turing Machines
All Languages

TMs and Computation

- Because Turing machines have the same computational powers as regular computers, we can (essentially) reason about Turing machines by reasoning about actual computer programs.
- Going forward, we're going to switch back and forth between TMs and computer programs based on whatever is most appropriate.
- In fact, our eventual proofs about the existence of impossible problems will involve a good amount of pseudocode. Stay tuned for details!

Decidability and Recognizability

What problems can we solve with a computer?

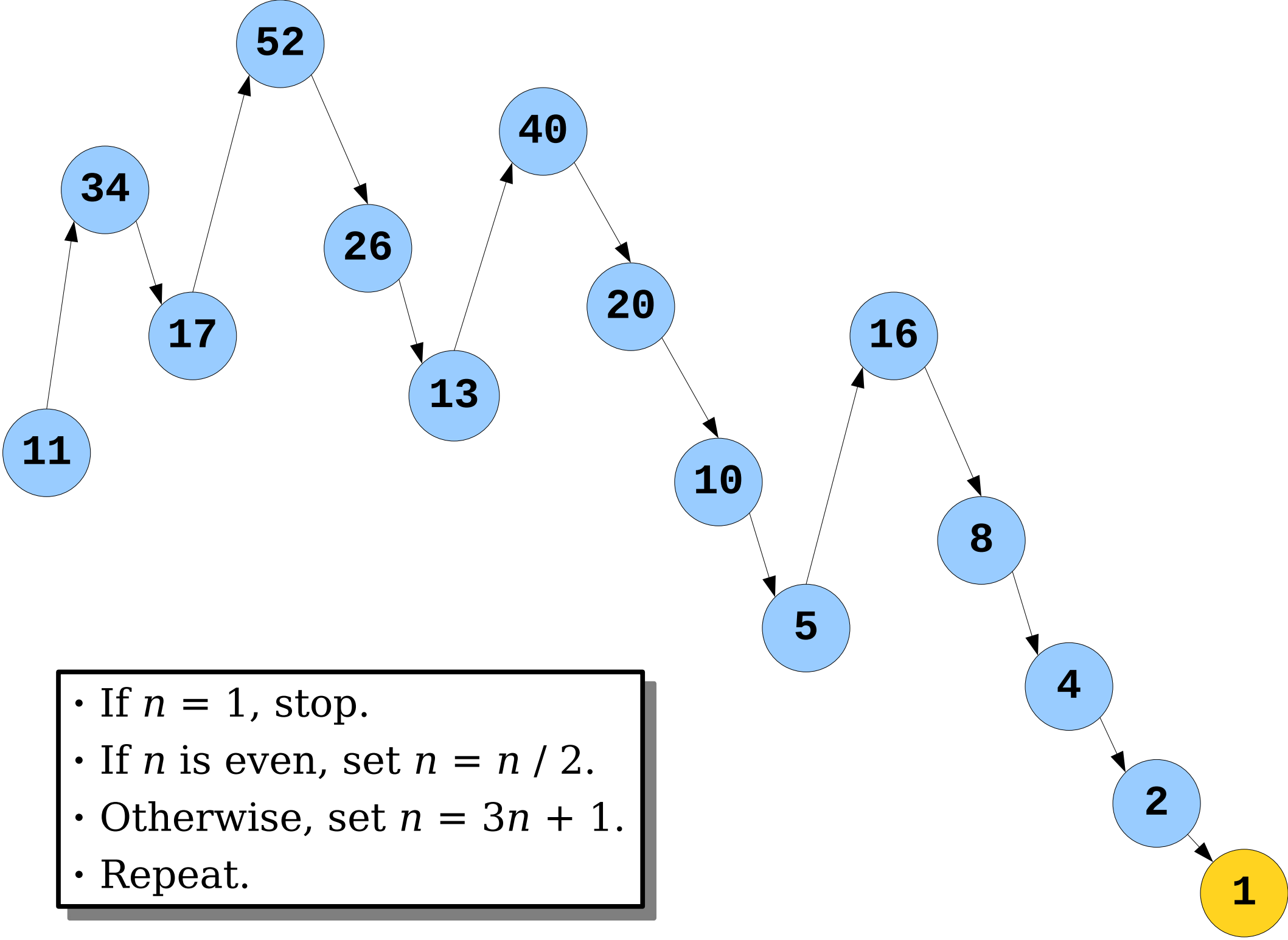
What kind of computer?

What problems can we solve with a computer?

What does it mean to “solve” a problem?

The Hailstone Sequence

- Consider the following procedure, starting with some $n \in \mathbb{N}$, where $n > 0$:
  - If $n = 1$, you are done.
  - If n is even, set $n = n / 2$.
  - Otherwise, set $n = 3n + 1$.
  - Repeat.
- ***Question:*** Given a natural number $n > 0$, does this process terminate?



- If $n = 1$, stop.
- If n is even, set $n = n / 2$.
- Otherwise, set $n = 3n + 1$.
- Repeat.

The Hailstone Sequence

- Consider the following procedure, starting with some $n \in \mathbb{N}$, where $n > 0$:
  - If $n = 1$, you are done.
  - If n is even, set $n = n / 2$.
  - Otherwise, set $n = 3n + 1$.
  - Repeat.
- Does the Hailstone Sequence terminate for…
  - $n = 5$?
  - $n = 20$?
  - $n = 7$?
  - $n = 27$?

The Hailstone Sequence

- Let $\Sigma = \{\mathbf{a}\}$ and consider the language

  $L = \{ \mathbf{a}^n \mid n > 0$ and the hailstone sequence terminates for $n \}$.

- Could we build a TM for L?

The Hailstone Turing Machine

- We can build a TM that works as follows:
  - If the input is ε, reject.
  - While the string is not **a**:
    - If the input has even length, halve the length of the string.
    - If the input has odd length, triple the length of the string and append a **a**.
  - Accept.

Does this Turing machine accept all nonempty strings?

The Collatz Conjecture

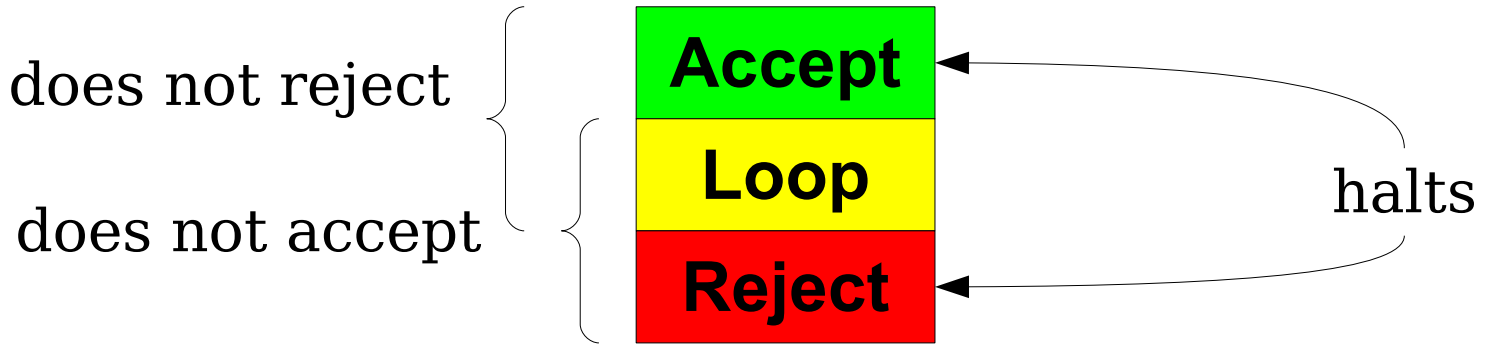
- It is *unknown* whether this process will terminate for all natural numbers.
- In other words, no one knows whether the TM described in the previous slides will always stop running!
- The conjecture (unproven claim) that the hailstone sequence always terminates is called the ***Collatz Conjecture***.
- This problem has eluded a solution for a long time. The influential mathematician Paul Erdős is reported to have said “Mathematics may not be ready for such problems.”

An Important Observation

- Unlike finite automata, which automatically halt after all the input is read, TMs keep running until they explicitly return true or return false.
- As a result, it’s possible for a TM to run forever without accepting or rejecting.
- This leads to several important questions:
  - How do we formally define what it means to build a TM for a language?
  - What implications does this have about problem-solving?

Very Important Terminology

- Let *M* be a Turing machine.
- *M* ***accepts*** a string *w* if it returns true on *w*.
- *M* ***rejects*** a string *w* if it returns false on *w*.
- *M* ***loops infinitely*** (or just ***loops***) on a string *w* if when run on *w* it neither returns true nor returns false.
- *M* ***does not accept w*** if it either rejects *w* or loops on *w*.
- *M* ***does not reject w*** *w* if it either accepts *w* or loops on *w*.
- *M* ***halts on w*** if it accepts *w* or rejects *w*.



Recognizers and Recognizability

- A TM M is called a ***recognizer*** for a language L over Σ if the following statement is true:

$$\forall w \in \Sigma^*. (w \in L \leftrightarrow M \text{ accepts } w)$$

- If you are absolutely certain that $w \in L$, then running a recognizer for L on w will (eventually) confirm this.
  - Eventually, M will accept w.
- If you don't know whether $w \in L$, running M on w may never tell you anything.
  - M might loop on w – but you can't differentiate between "it'll never give an answer" and "just wait a bit more."
- Does that feel like "solving a problem" to you?

Recognizers and Recognizability

- The hailstone TM M we saw earlier is a recognizer for the language

$$L = \{ \mathbf{a}^n \mid n > 0 \text{ and the hailstone sequence terminates for } n \}.$$

- If the sequence does terminate starting at n, then M accepts $\mathbf{a}^n$.
- If the sequence doesn't terminate, then M loops forever on $\mathbf{a}^n$. and never gives an answer.
- If you somehow knew the hailstone sequence terminated for n, this machine would (eventually) confirm this. If you didn't know, this machine might not tell you anything.

```
bool pizkwat(string input) {
  return false;
}
```

```
bool squigglebah(string input) {
  while (true) {
    // do nothing
  }
}
```

```
bool moozle(string input) {
  int oot = 1;
  while (input.size() != oot) {
    oot += oot;
  }
  return true;
}
```

```
bool humblegwah(string input) {
  if (input.size() % 2 != 0) return false;

  for (int i = 0; i < input.size() / 2; i
    if (input[2 * i] != input[2 * i + 1])
      return false;
    }
  }

  return true;
}
```

$$\forall w \in \Sigma^*. (w \in L \quad \leftrightarrow \quad M \text{ accepts } w)$$

Each of these pieces of code is a recognizer for some language.
What language does each recognizer recognize?

Recognizers and Recognizability

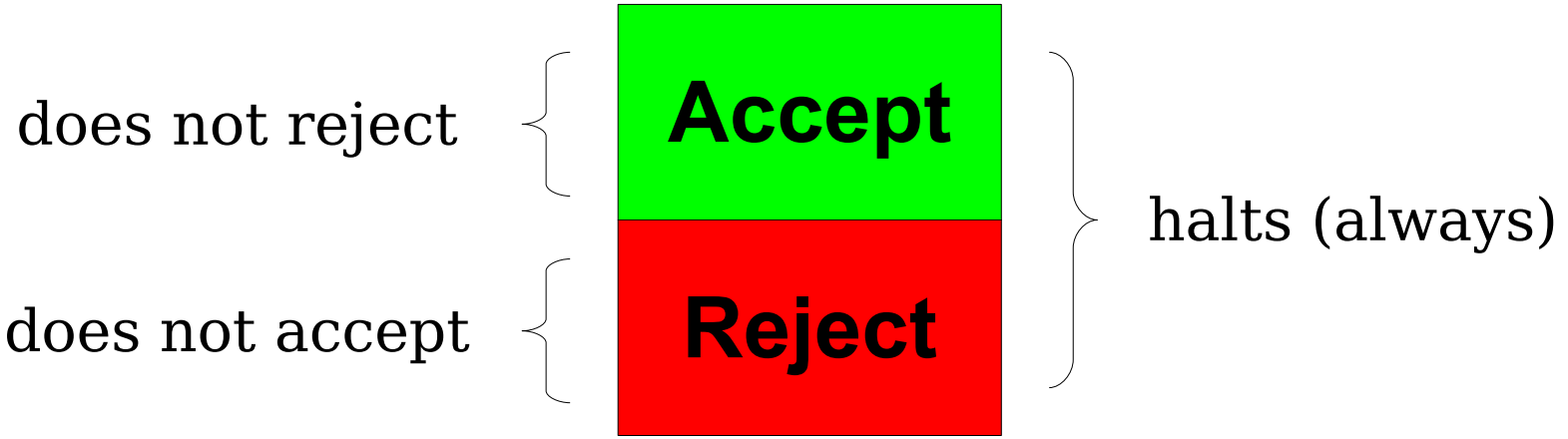
- The class **RE** consists of all recognizable languages.
- Formally speaking:

  $\mathbf{RE} = \{ L \mid L \text{ is a language and there's a recognizer for } L \}$

- You can think of **RE** as "all problems with yes/no answers where "yes" answers can be confirmed by a computer."
  - Given a recognizable language L and a string $w \in L$, running a recognizer for L on w will eventually confirm $w \in L$.
  - The recognizer will never have a "false positive" of saying that a string is in L when it isn't.
- This is a "weak" notion of solving a problem.
- Is there a "stronger" one?

Deciders and Decidability

- Some, but not all, TMs have the following property: the TM halts on all inputs.
- If you are given a TM M that always halts, then for the TM M, the statement "M does not accept w" means "M rejects w."



Deciders and Decidability

- A TM M is called a ***decider*** for a language L over Σ if the following statements are true:

$$\forall w \in \Sigma^*.\ M \text{ halts on } w.$$

$$\forall w \in \Sigma^*.\ (w \in L \ \leftrightarrow \ M \text{ accepts } w)$$

- In other words, M accepts all strings in L and rejects all strings not in L.
- In other words, M is a recognizer for L, and M halts on all inputs.
- If you aren't sure whether $w \in L$, running M on w will (eventually) give you an answer to that question.

Deciders and Decidability

- The hailstone TM M we saw earlier is a ***recognizer*** for the language

$$L = \{ \mathbf{a}^n \mid n > 0 \text{ and the hailstone sequence terminates for } n \}.$$

- If the hailstone sequence terminates for n, then M accepts $\mathbf{a}^n$. If it doesn't, then M does not accept $\mathbf{a}^n$.
- We honestly don't know if M is a decider for this language.
  - If the hailstone sequence always terminates, then M always halts and is a decider for L.
  - If the hailstone sequence doesn't always terminate, then M will loop on some inputs and isn't a decider for L.

```
bool pizkwat(string input) {
  return false;
}
```

```
bool squigglebah(string input) {
  while (true) {
    // do nothing
  }
}
```

```
bool moozle(string input) {
  int oot = 1;
  while (input.size() != oot) {
    oot += oot;
  }
  return true;
}
```

```
bool humblegwah(string input) {
  if (input.size() % 2 != 0) return false;

  for (int i = 0; i < input.size() / 2; i
    if (input[2 * i] != input[2 * i + 1])
      return false;
    }
  }

  return true;
}
```

$$\forall w \in \Sigma^*.\ M \text{ halts on } w$$

$$\forall w \in \Sigma^*.\ (w \in L \quad \leftrightarrow \quad M \text{ accepts } w)$$

Each piece of code is a recognizer for a language.
Which are deciders?

Deciders and Decidability

- The class **R** consists of all decidable languages.
- Formally speaking:

  R = { L | L is a language and there's a decider for L }

- You can think of **R** as "all problems with yes/no answers that can be fully solved by computers."
  - Given a decidable language, run a decider for L and see what happens.
  - Think of this as "knowledge creation" – if you don't know whether a string is in L, running the decider will, given enough time, tell you.
- The class **R** contains all the regular languages, all the context-free languages, most of CS161, etc.
- This is a "strong" notion of solving a problem.

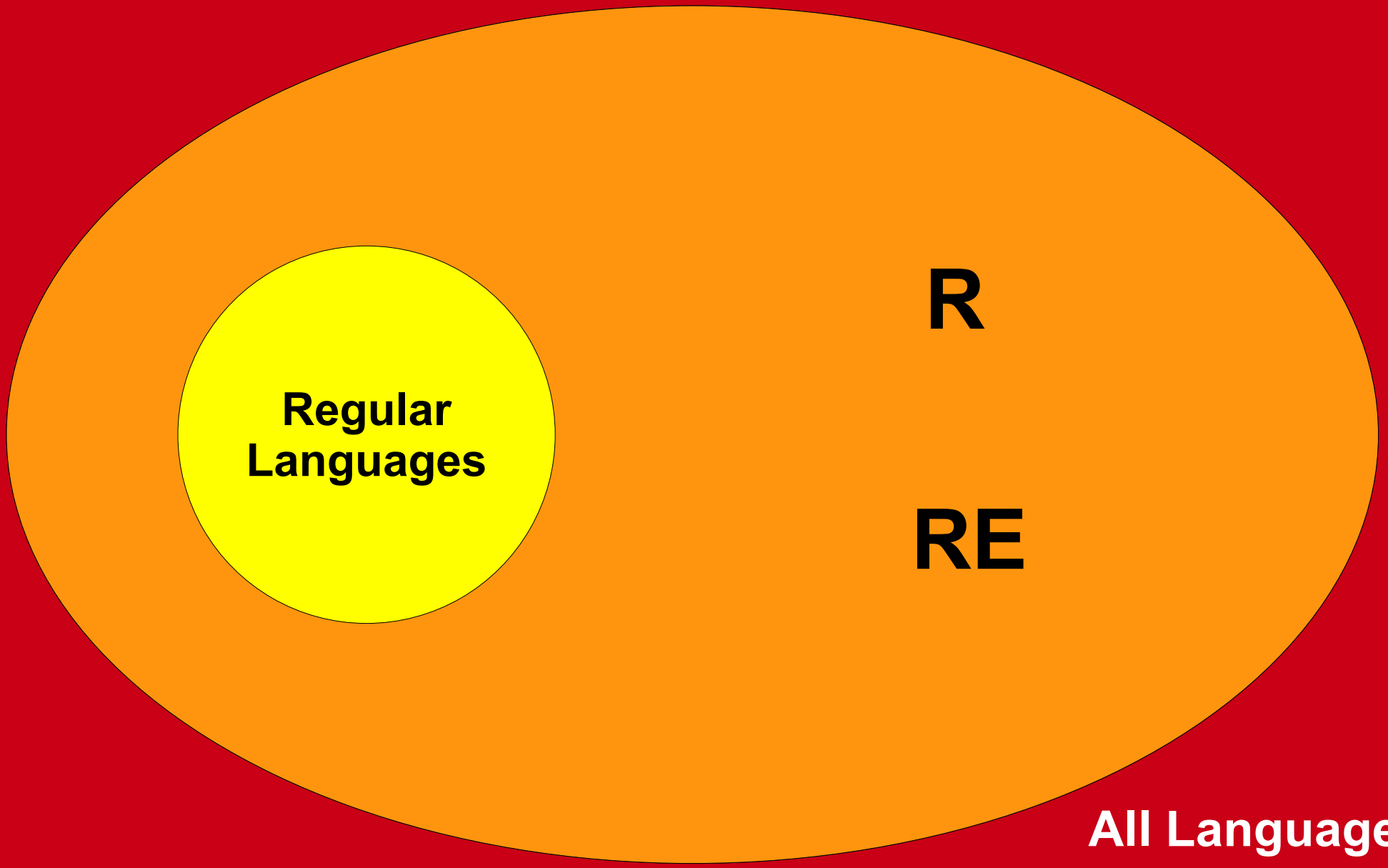
R and **RE** Languages

- Every decider for L is also a recognizer for L.
- This means that $\mathbf{R} \subseteq \mathbf{RE}$.
- Hugely important theoretical question:

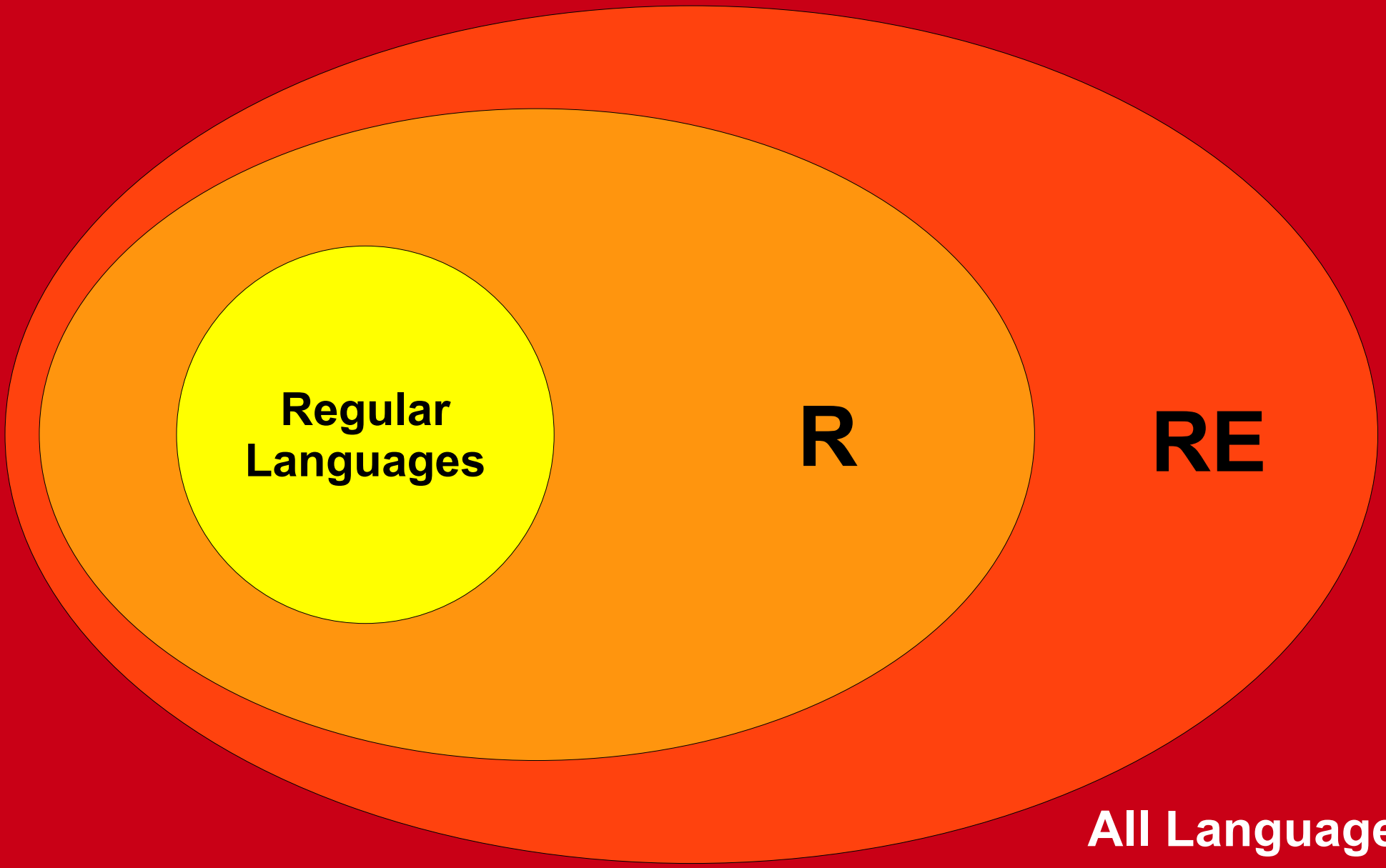
$$\mathbf{R} \stackrel{?}{=} \mathbf{RE}$$

- That is, if you can just confirm "yes" answers to a problem, can you necessarily *solve* that problem?

Which Picture is Correct?



Which Picture is Correct?



Unanswered Questions

- Why exactly is **RE** an interesting class of problems?
- What does the **R** $\stackrel{?}{=}$ **RE** question mean?
- Is **R** = **RE**?
- What lies beyond **R** and **RE**?
- We'll see the answers to each of these in due time.

Next Time

- ***Emergent Properties***
  - Larger phenomena made of smaller parts.
- ***Universal Machines***
  - A single, “most powerful” computer.
- ***Self-Reference***
  - Programs that ask questions about themselves.