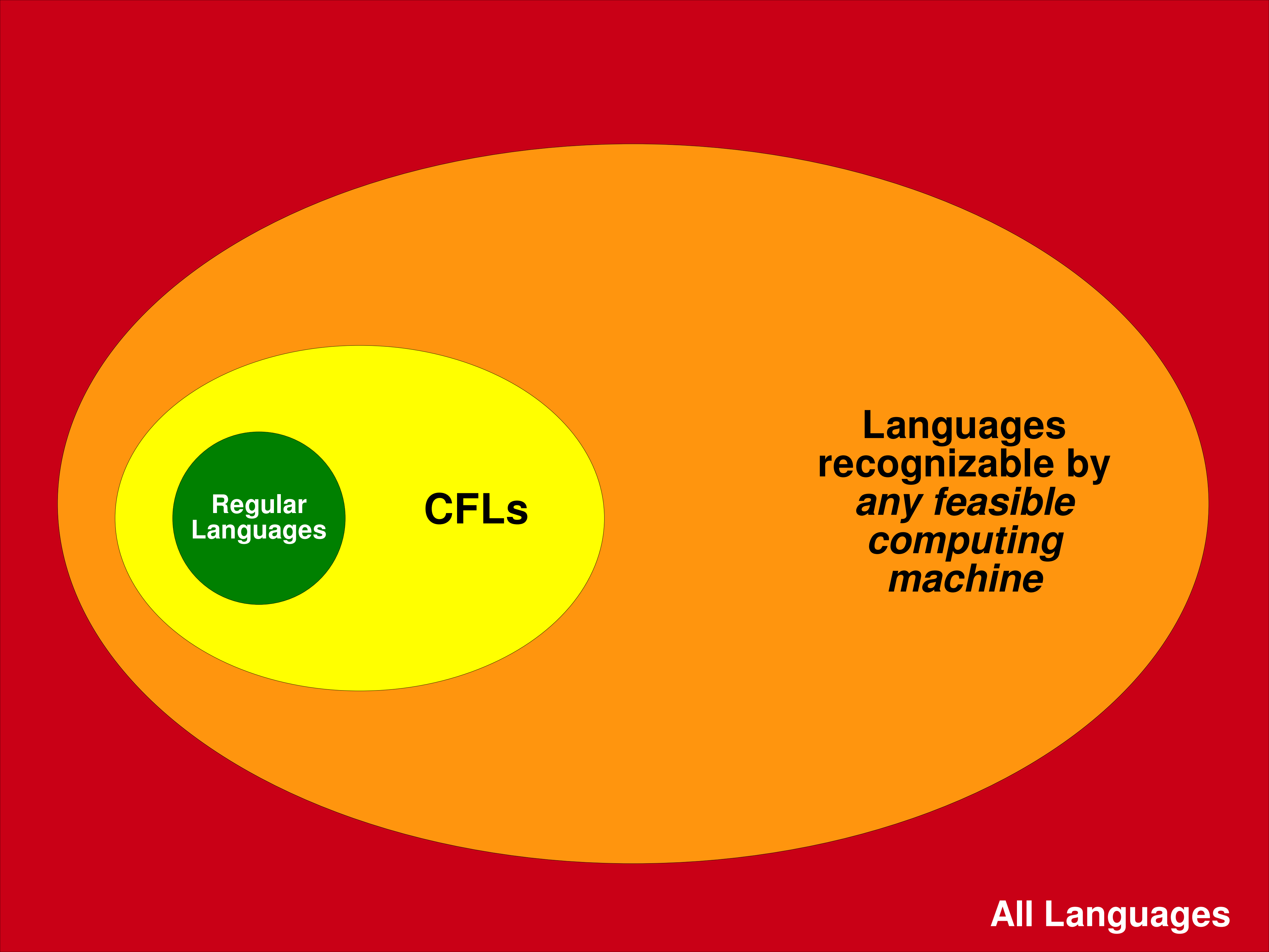
Turing Machines

Part One

What problems can we solve with a computer?


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

That same drawing, to scale.

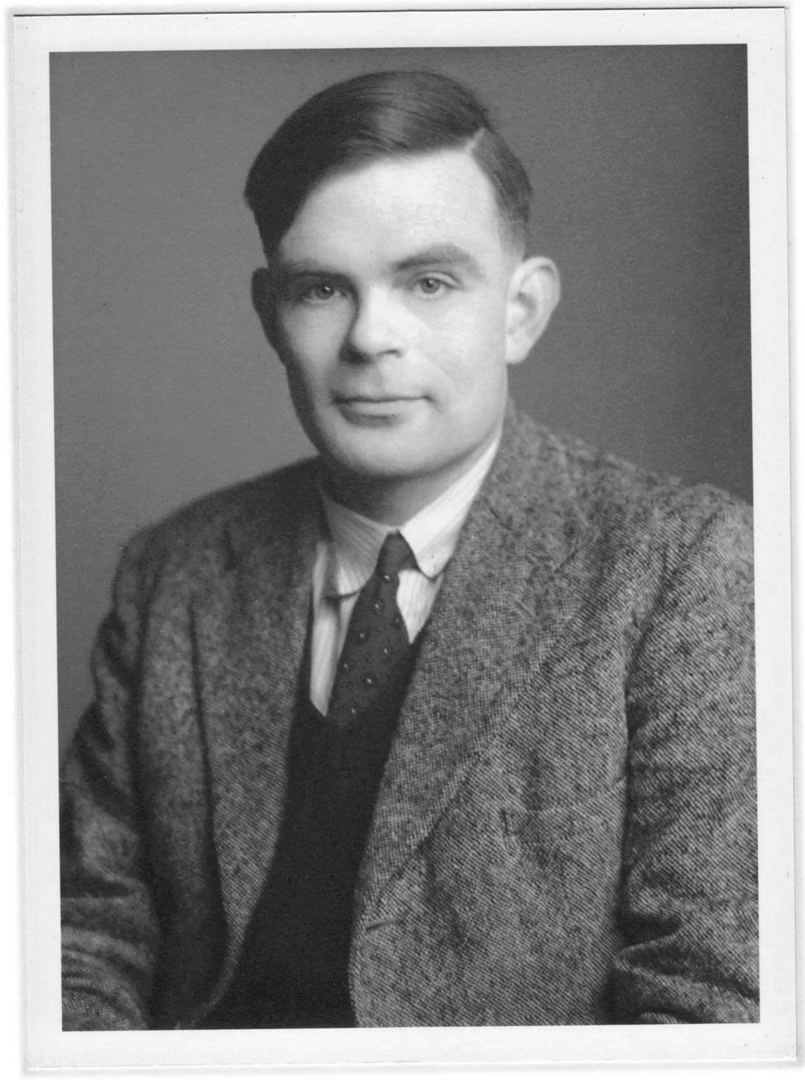
The Problem

- Finite automata accept precisely the regular languages.
- We may need unbounded memory to recognize context-free languages.
  - e.g. $\{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ 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.



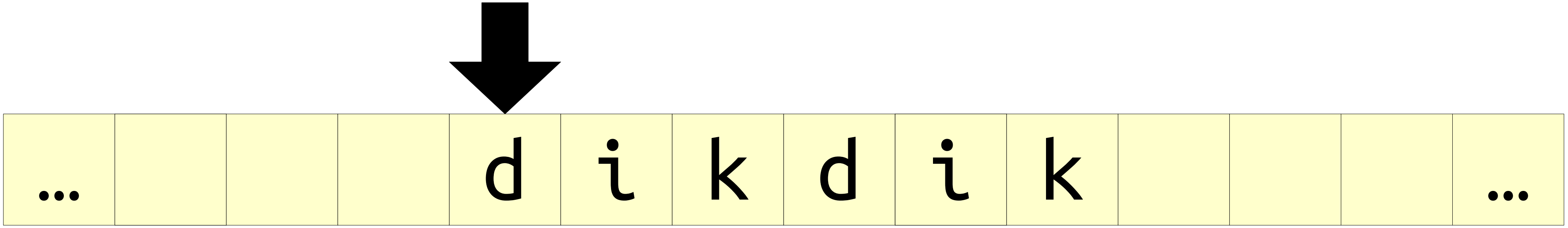
https://static01.nyt.com/images/2019/06/10/obituaries/10overlooked-alanturing-1/f86fc26c86b444c391fb412bb3dcec76-superJumbo.jpg?quality=75&auto=webp

$$
\begin{array}{ccccccccccccccc}
 & & & & & {\scriptstyle 1} & & {\scriptstyle 1} & & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & \\
 & 2 & 7 & 1 & 8 & 2 & 8 & 1 & 8 & 2 & 8 & 4 & 5 & 9 & 0 \\
+ & 3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 & 5 & 3 & 5 & 8 & 9 & 7 \\
\hline
 & 5 & 8 & 5 & 9 & 8 & 7 & 4 & 4 & 8 & 2 & 0 & 4 & 8 & 7
\end{array}
$$

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.



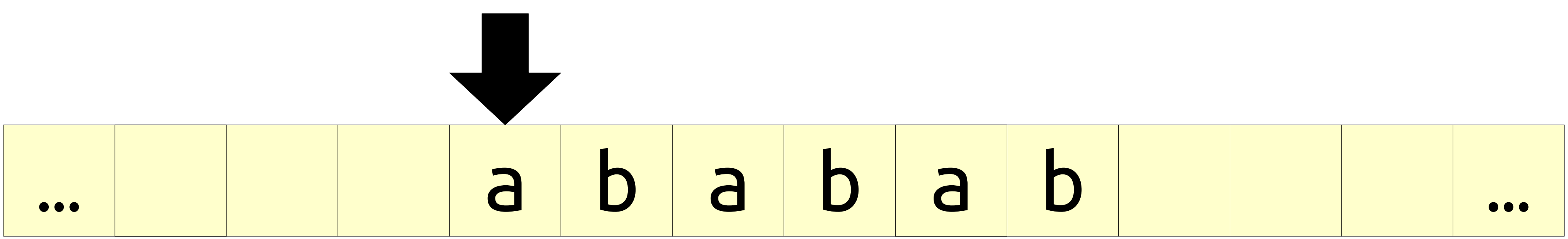
Turing Machines

- 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.)
- If you'd like to learn more about Turing's original version of the Turing machine, come chat with me after class!

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.)

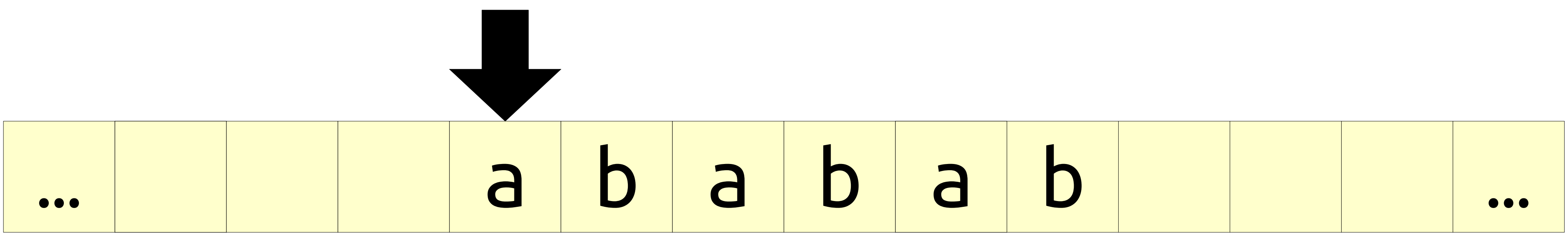
```
Start:
   If Blank Return True
   If 'b' Return False
   Write 'x'
   Move Right
   If Not 'b' Return False
   Write 'x'
   Move Right
   Goto Start
```



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 False
   Write 'x'
   Move Right
   Goto Start
```



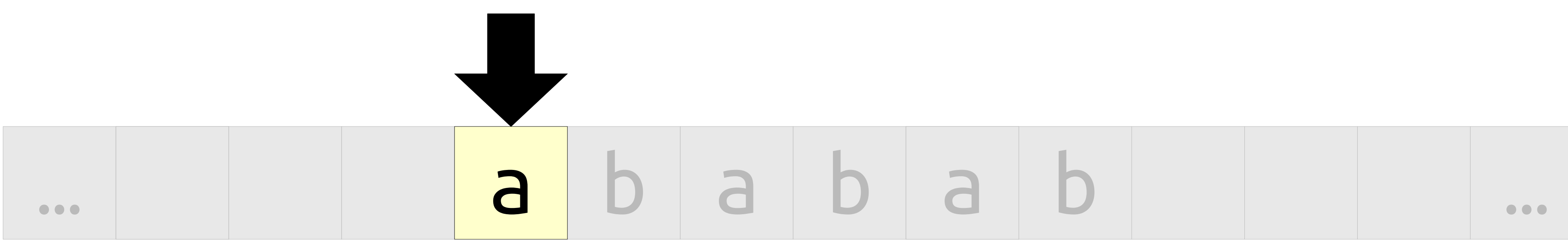
Turing Machines

- 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 False
    Write 'x'
    Move Right
    Goto Start
```



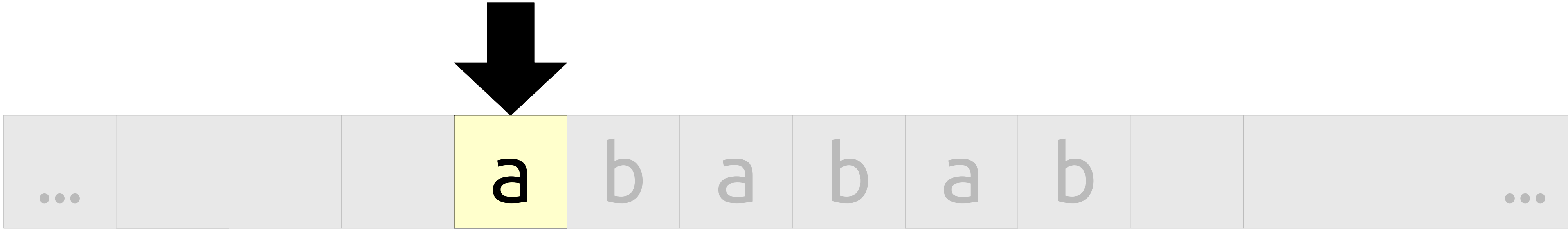
Turing Machines

- 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 False
   Write 'x'
   Move Right
   Goto Start
```



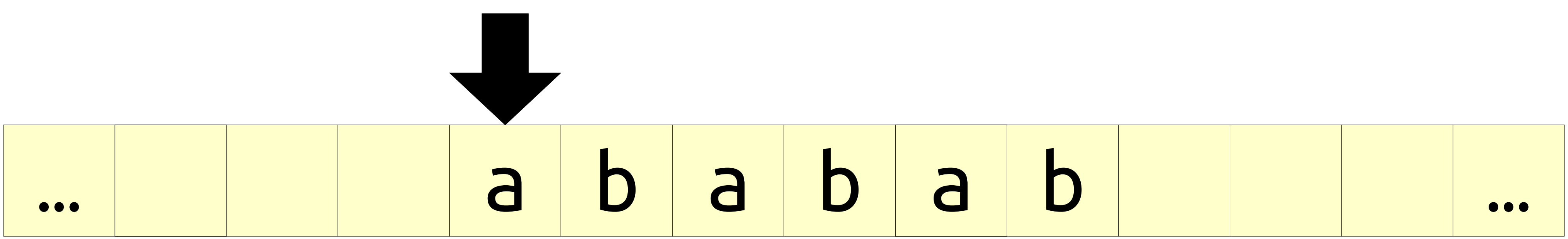
Turing Machines

- 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 False
   Write 'x'
   Move Right
   Goto Start
```



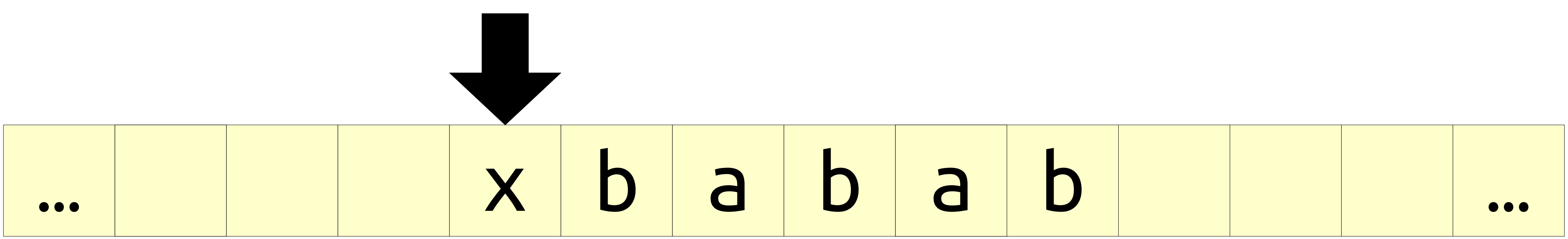
Turing Machines

- 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 False
   Write 'x'
   Move Right
   Goto Start
```



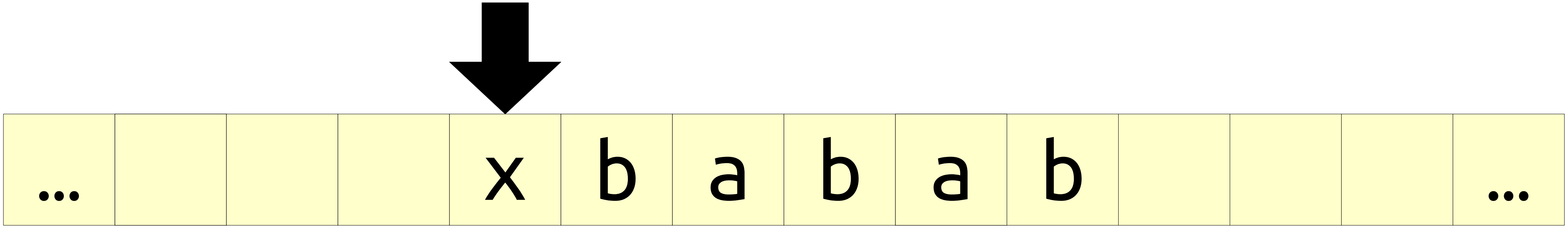
Turing Machines

- 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 False
   Write 'x'
   Move Right
   Goto Start
```



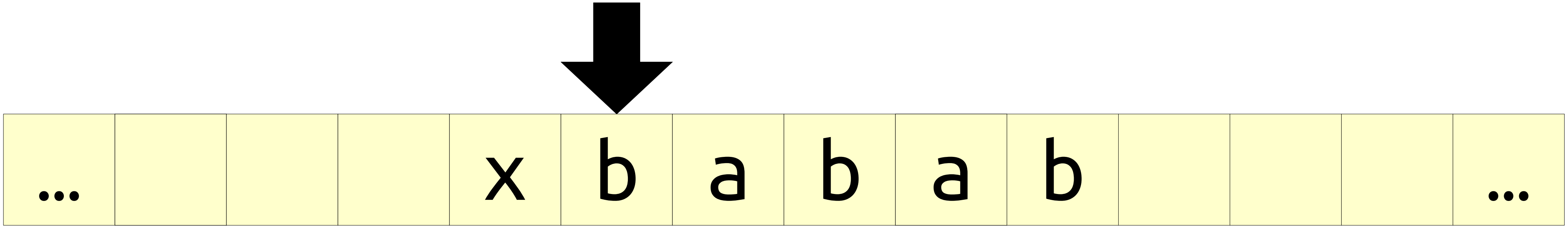
Turing Machines

- 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 False
   Write 'x'
   Move Right
   Goto Start
```



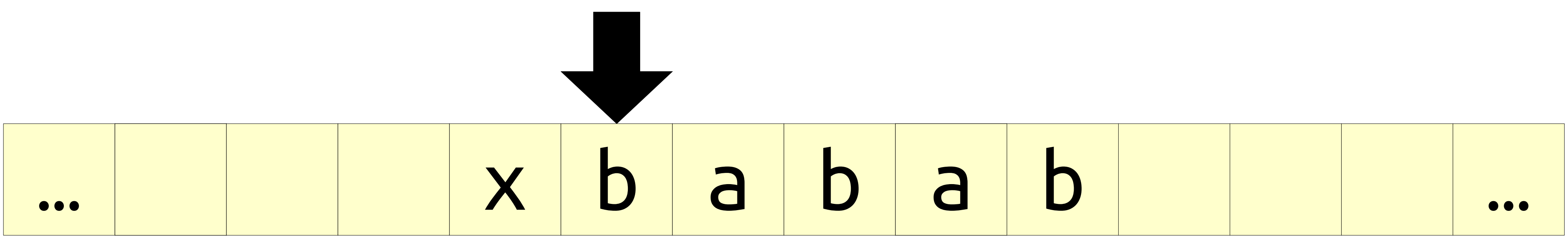
Turing Machines

- 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 False
   Write 'x'
   Move Right
   Goto Start
```



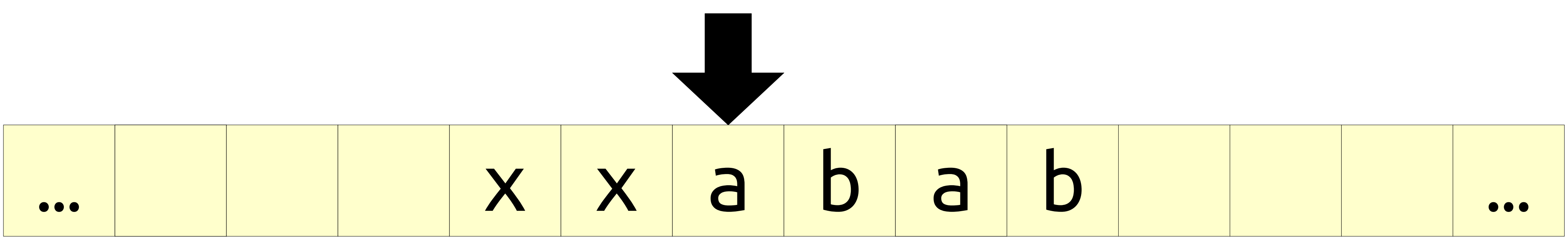
Turing Machines

- 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 False
   Write 'x'
   Move Right
   Goto Start
```



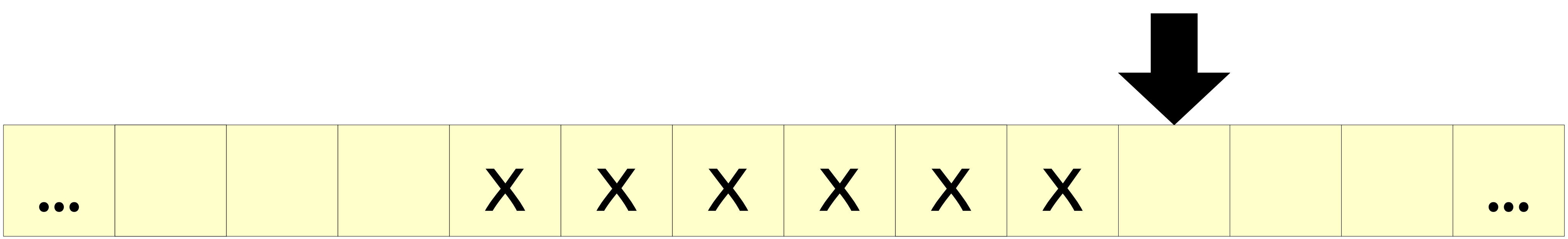
Turing Machines

- 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.)

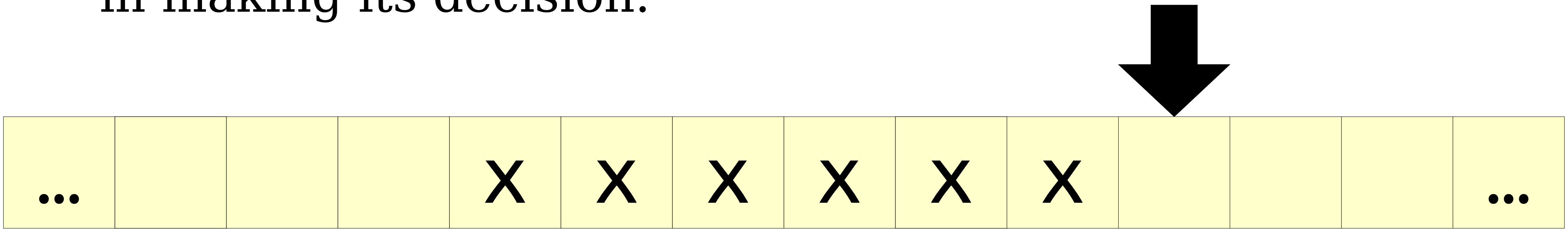
```
Start:
    If Blank Return True
    If 'b' Return False
    Write 'x'
    Move Right
    If Not 'b' Return False
    Write 'x'
    Move Right
    Goto Start
```



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.

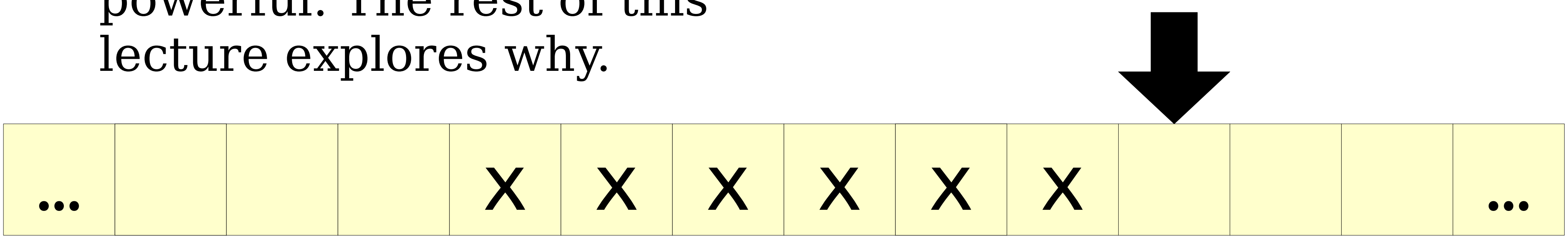
```
Start:
    If Blank Return True
    If 'b' Return False
    Write 'x'
    Move Right
    If Not 'b' Return False
    Write 'x'
    Move Right
    Goto Start
```



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.

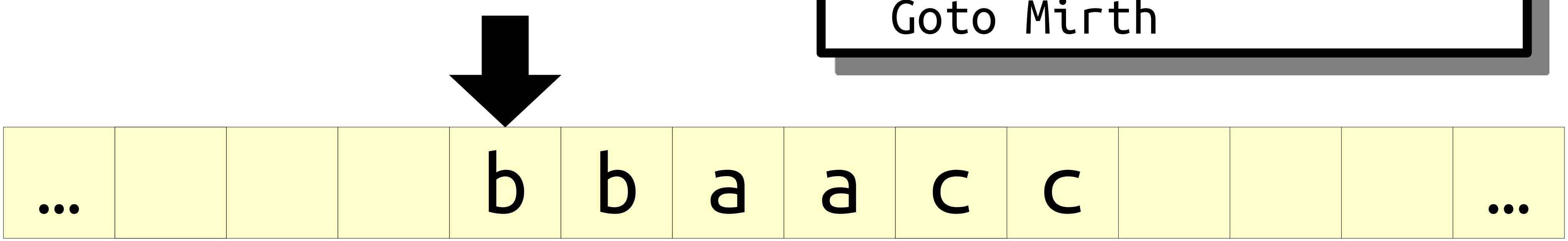
```
Start:
  If Blank Return True
  If 'b' Return False
  Write 'x'
  Move Right
  If Not 'b' Return False
  Write 'x'
  Move Right
  Goto Start
```



Your Turn!

- Draw what the tape and tape head look like when this TM finishes running.
- Is the input `bbaacc` accepted or rejected?
- More generally, what does this TM do?

```
Start:
  If 'a' Goto Mirth
  If Blank Return False
  Move Right
  Goto Start
Mirth:
  If 'b' Return True
  If Blank Return False
  Move Right
  Goto Mirth
```



Programming Turing Machines

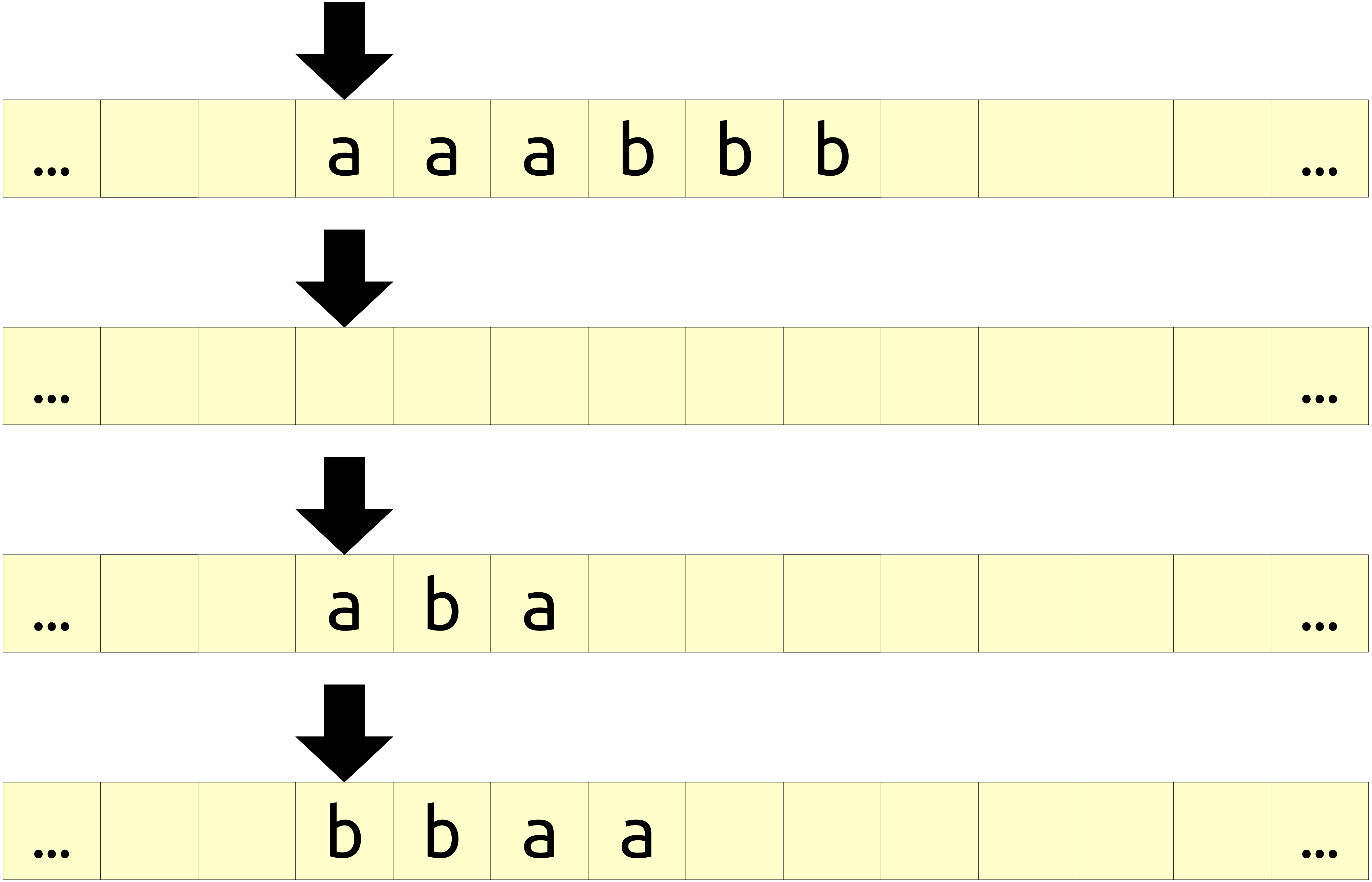
Our First Challenge

- 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.

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



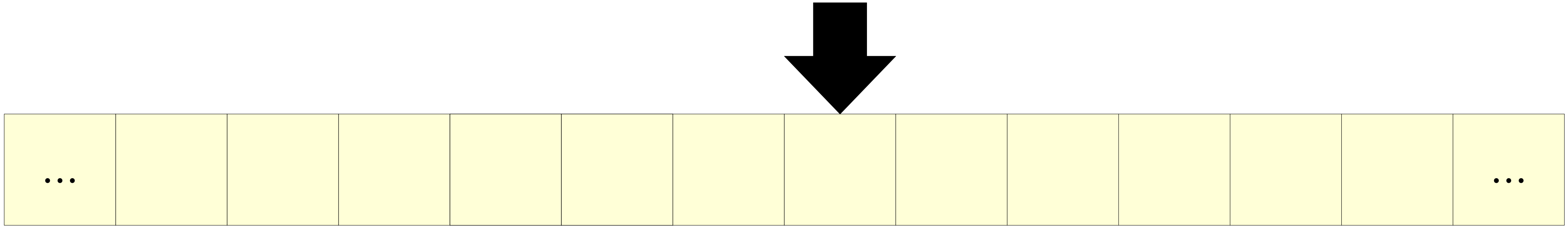
A Recursive Approach

- 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.

```
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!

Second Midterm Complete

- You’re done with the second midterm exam – congratulations!
- We will be grading the exam over the weekend and will get back to you with scores as soon as it’s ready.
- Solutions are up online, along with information about historical grade cutoff markers.
- ***Do not withdraw or change your grading basis*** unless you have run some projections about your raw score! Check what the numbers say first.

Problem Set Six

- The TAs are currently finishing grading PS6. We’ll release scores as soon as they’re ready.
- Solutions are up on the course website. Feel free to read over them in the meantime.

Your Questions

"What is your favorite bike route in the Bay Area?"

There are some beautiful, beautiful trails near Crystal Springs Reservoir – it's really incredible there. The "Bayway" goes from SF to Mountain View and is a great way to tour around the area.

Back to CS103!

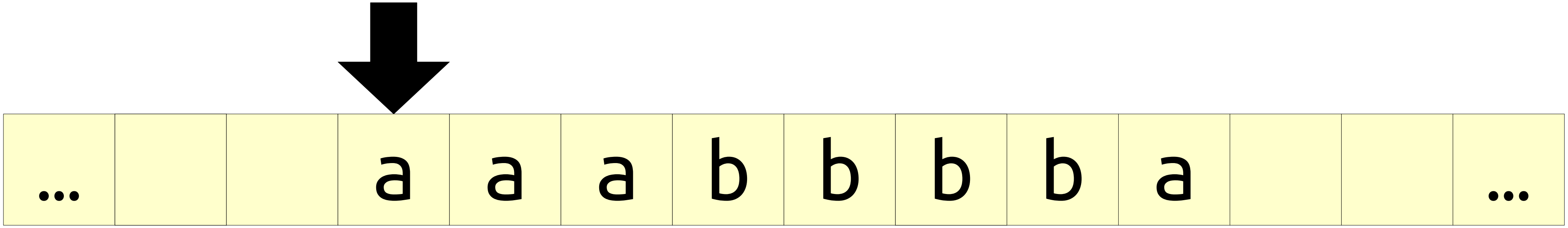
Our Next Challenge

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

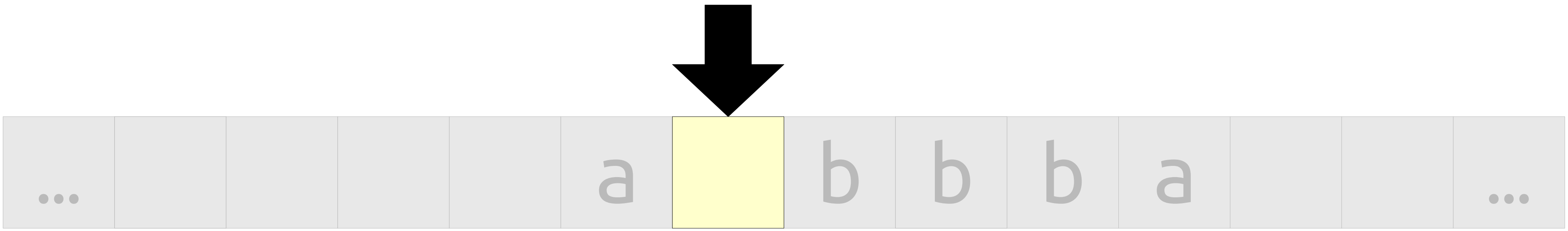
  $\{ w \in \{\texttt{a}, \texttt{b}\}^* \mid w$ has an equal number of `a`'s and `b`'s $\}$

- This language is not regular (do you see why?)
- It is context-free, but it's a bit tricky to write a CFG for it. (See PS8!)
- Let's see how to design a TM for it.

A Caveat

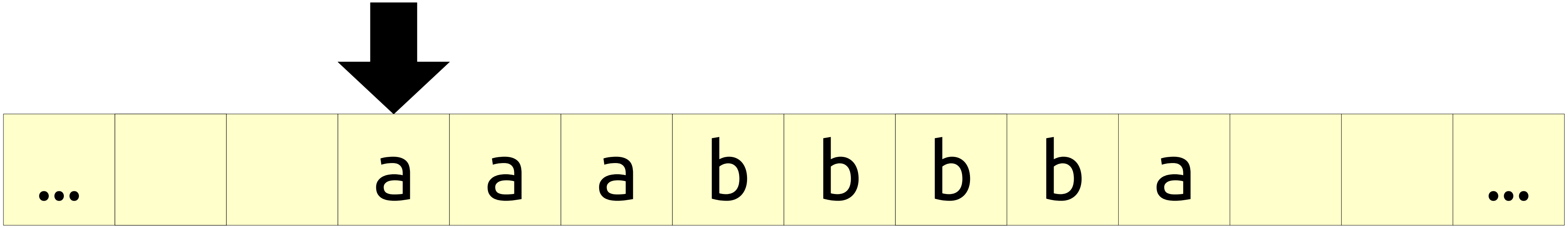


A Caveat

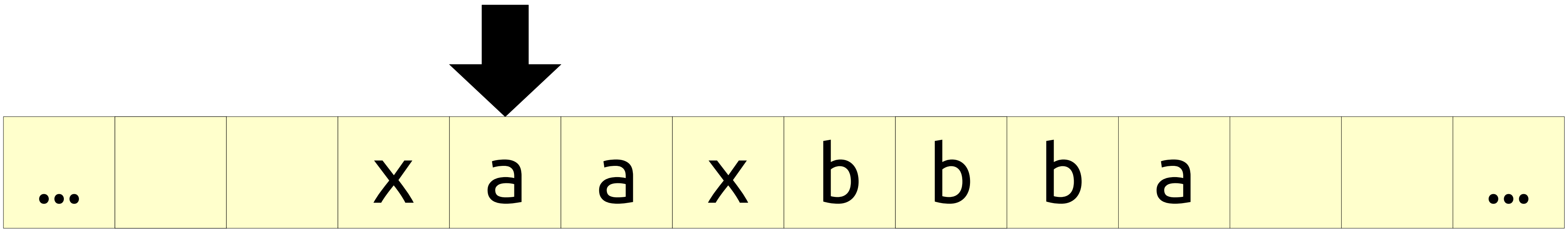


How do we know that this blank isn't one of the infinitely many blanks after our input string?

One Solution



One Solution

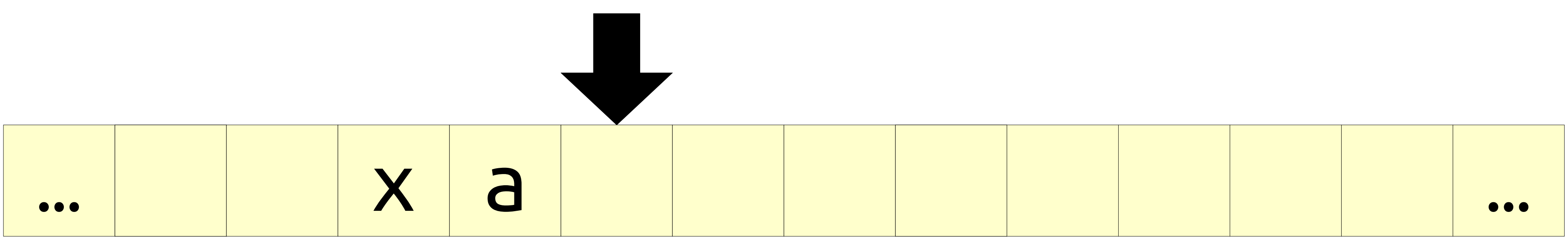


```
Start:
  If 'a' Goto FoundA
  If 'b' Goto FoundB
  If Blank Return True
  Move Right
  Goto Start

FoundA:
  Write 'x'
LoopA:
  Move Right
  If 'a' Goto LoopA
  If 'x' Goto LoopA
  If Blank Return False
  Write 'x'
  Goto GoHome

GoHome:
  Move Left
  If Not Blank Goto GoHome
  Move Right
  Goto Start

FoundB:
  Write 'x'
LoopB:
  Move Right
  If 'b' Goto LoopB
  If 'x' Goto LoopB
  If Blank Return False
  Write 'x'
  Goto GoHome
```



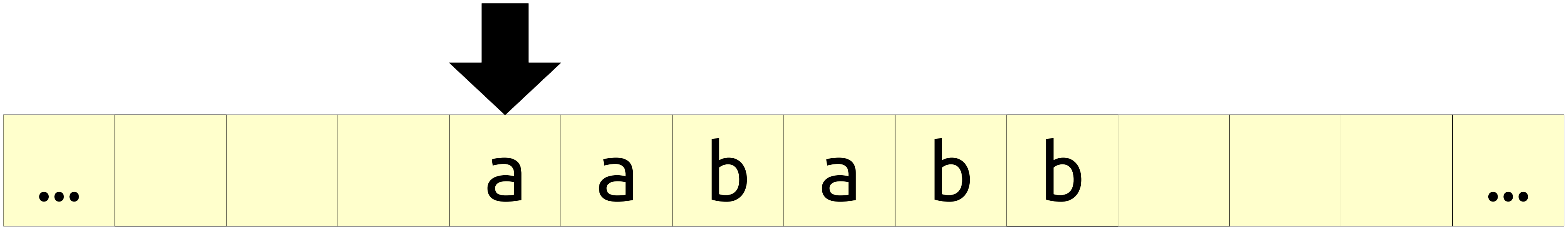
Another Idea

- We just built a TM for the language

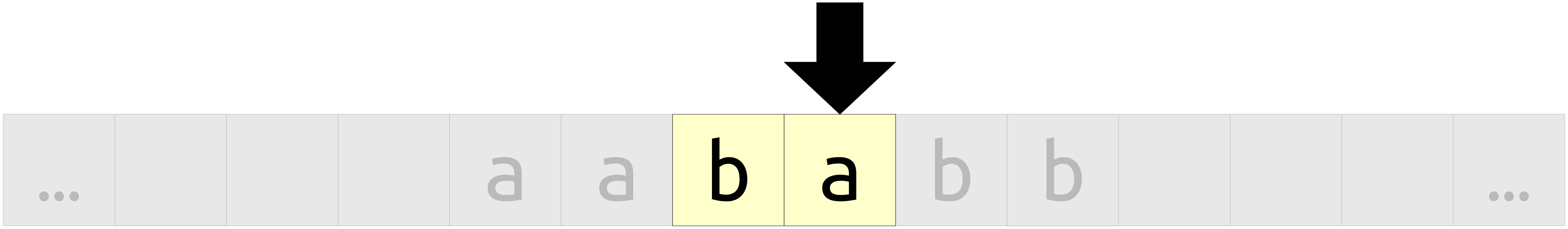
  { w ∈ {**a**, **b**}* | w has the same number of **a**'s and **b**'s }.

- 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?

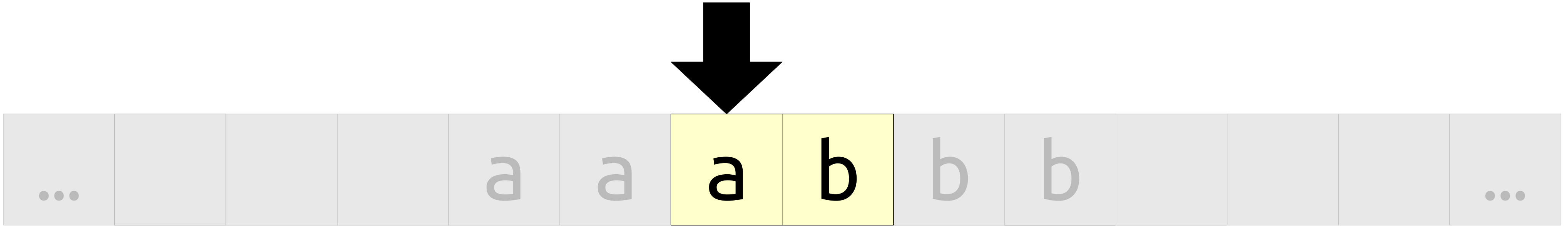
The Idea



The Idea



The Idea



Exploring This Idea

Summary for Today

- Turing machines are abstract computers that issue commands to an infinite tape subdivided into cells.
- Each step of the TM can move the tape head, change what's on the tape, or jump to a different part of the program.
- TMs can be composed together to build larger TMs out of smaller ones.

Next Time

- ***The Church-Turing Thesis***
  - How powerful are Turing machines?
- ***Decidability and Recognizability***
  - Two notions of “solving a problem.”