Collections, Part Two

Outline for Today

- ***Parameters in C++***
  - A third option for parameter passing.
- ***Stacks***
  - Pancakes meets parsing!
- ***Queues***
  - Waiting in line at the Library of Babel.

Parameters in C++

Parameter Passing in C++

- By default, in C++, parameters are passed by value.

```
/* This function gets a copy of the string passed
 * into it, so we only change our local copy. The
 * caller won't see any changes.
 */
void byValue(string text) {
    text += "!";
}
```

- You can place an ampersand after the type name to take the parameter by reference.

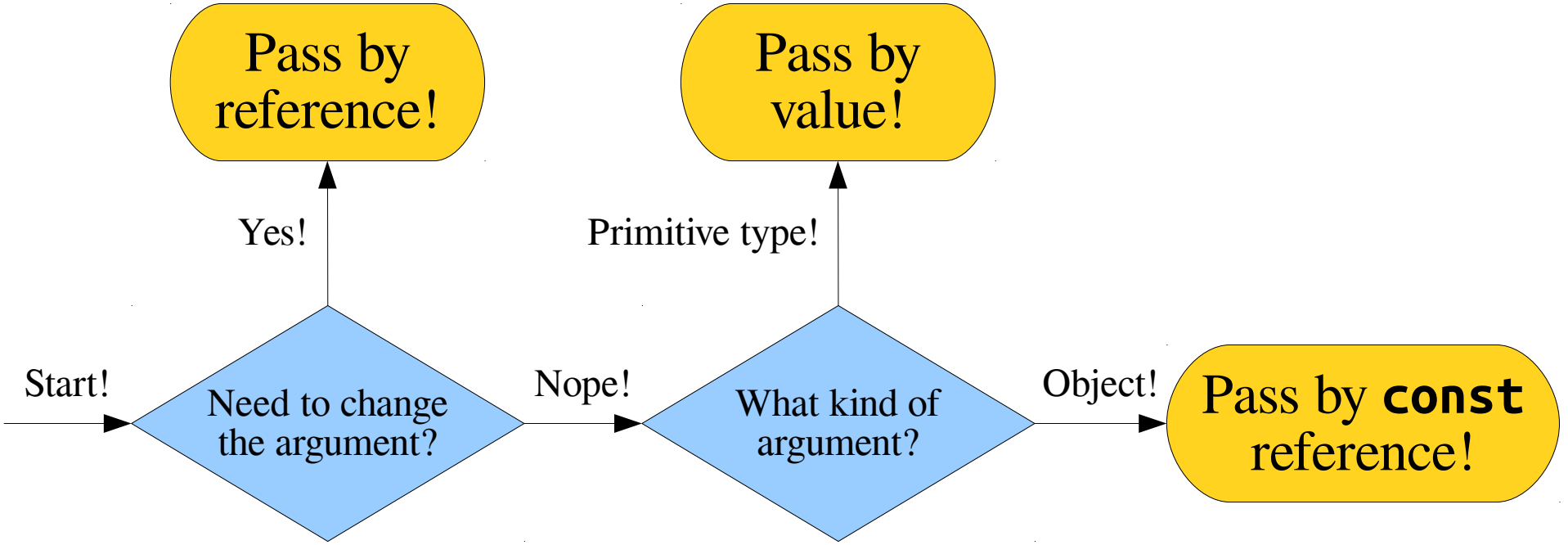
```
/* This function takes its argument by reference, so
 * when the function returns the string passed in will have
 * been permanently changed.
 */
void byReference(string& text) {
    text += "!";
}
```

Pass-by-`const`-Reference

- Passing a large object (e.g. a million-element `Vector`)by value makes a copy, which can take a *lot* of time.
- Taking parameters by reference avoids making a copy, but risks that the object gets tampered with in the process.
- As a result, it's common to have functions that take objects as parameters take their argument by ***`const` reference***:
  - The "by reference" part avoids a copy.
  - The "**`const`**" (constant) part means that the function can't change that argument.
- For example:

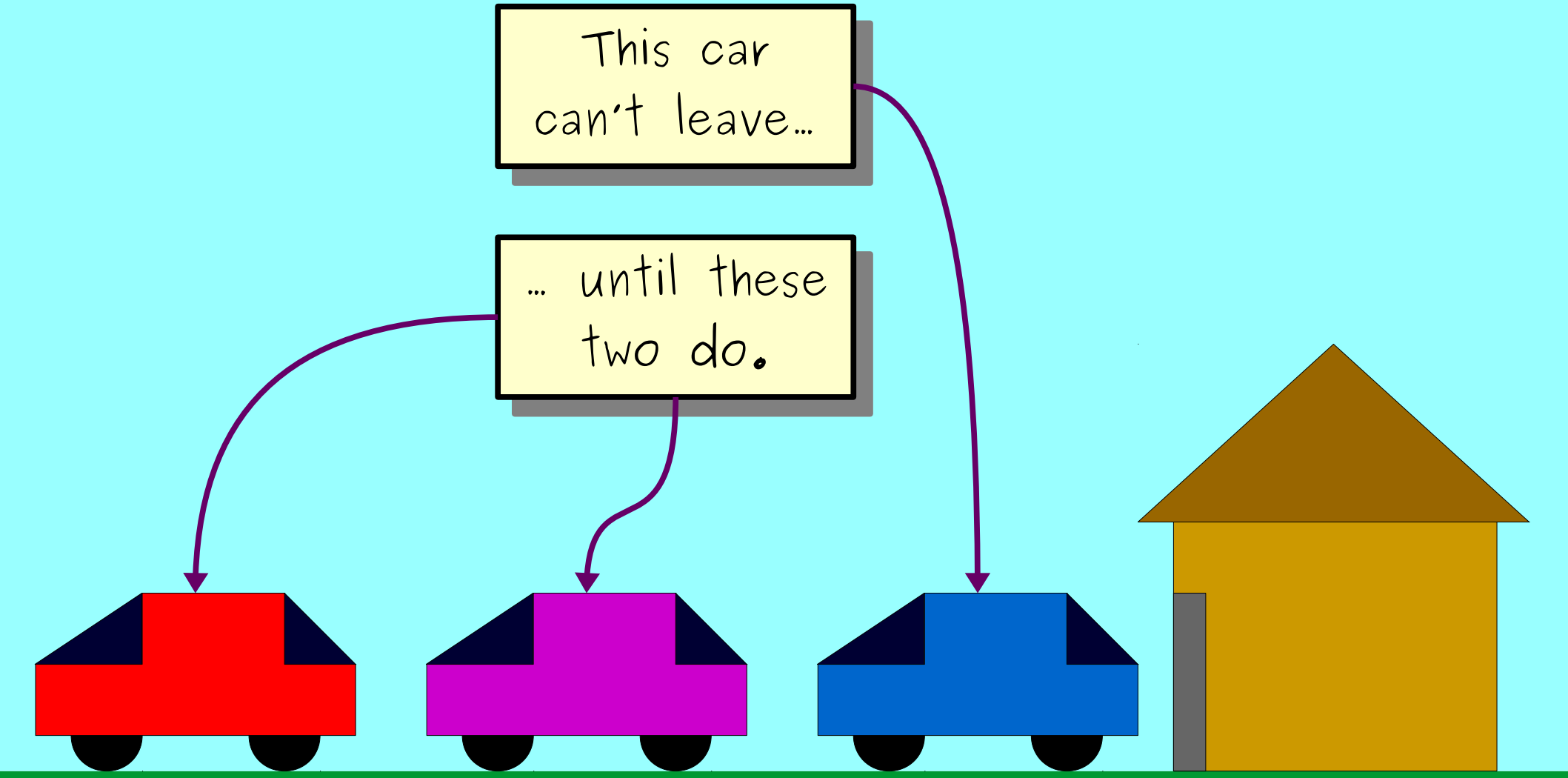
```
void proofreadLongEssay(const string& essay) {
   /* can read, but not change, the essay.  */
}
```

Parameter Flowchart



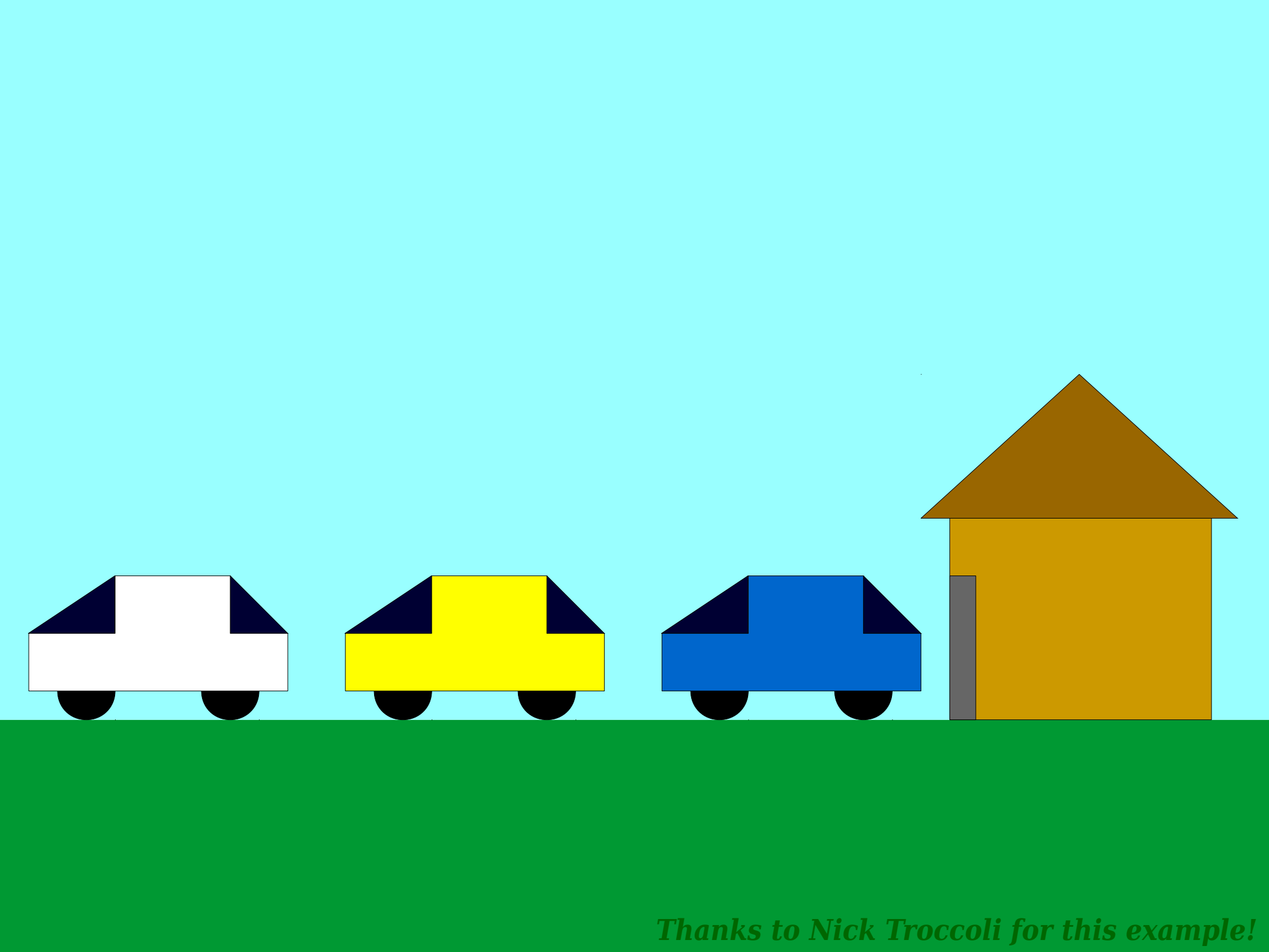
This is the general convention used in C++ programming. Please feel free to ask questions about this over the course of the quarter!

Stack



Thanks to Nick Troccoli for this example!


Any new car precedes all the old cars. Only this car can leave.
Thanks to Nick Troccoli for this example!


Thanks to Nick Troccoli for this example!

Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

Stack

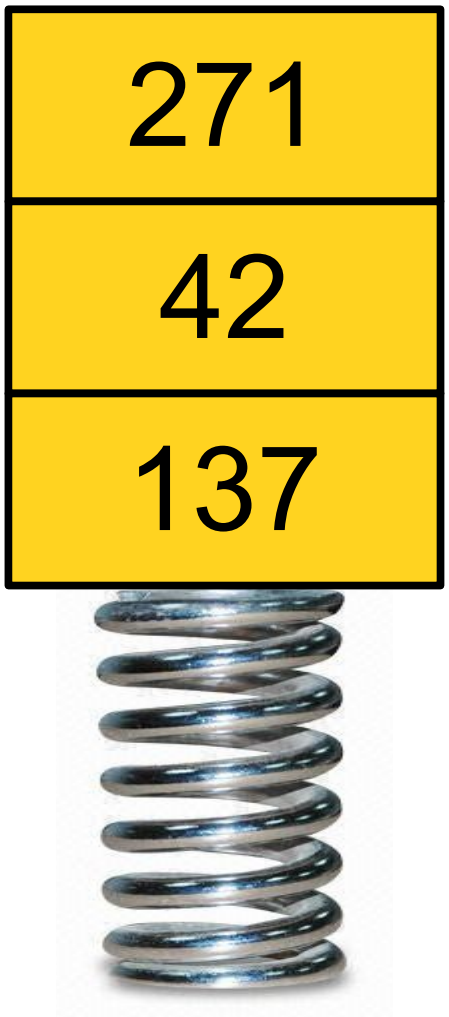
- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

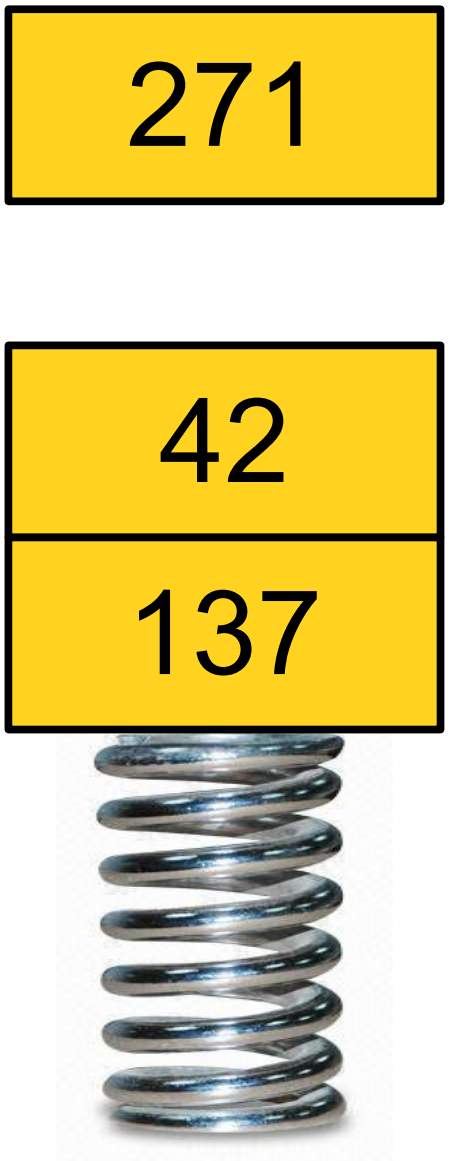
Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.



Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.



Stack

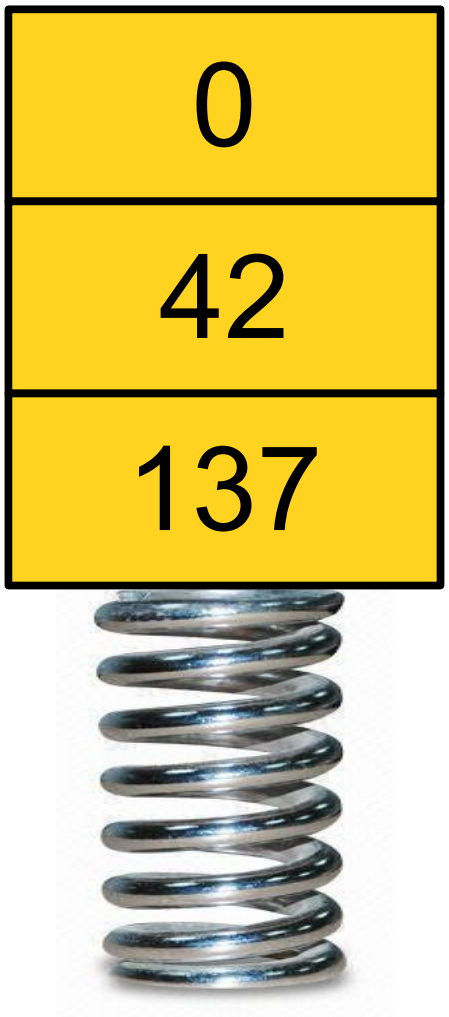
- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

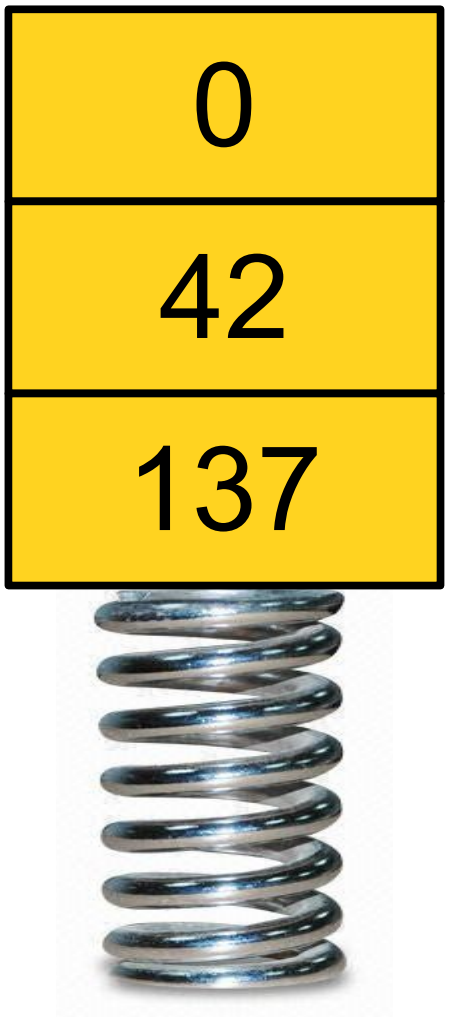
Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.



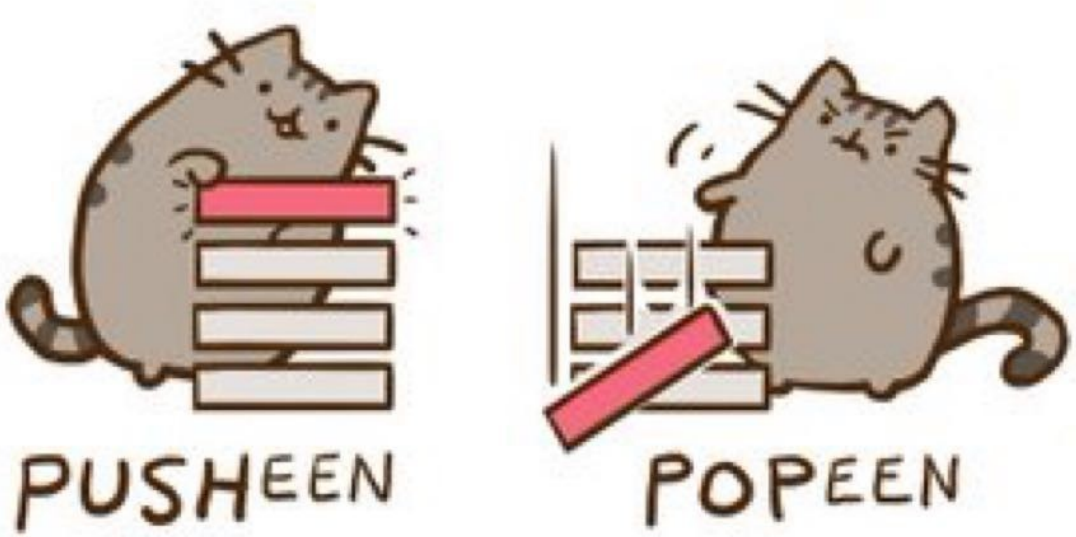
Stack

- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.
- Only the topmost element of a `Stack` can be accessed.
- Do you see why we call it the *call stack* and talk about *stack frames*?



Thanks to Amy Nguyen for this example!

Thanks to Amy Nguyen for this example!



Thanks to Amy Nguyen for this example!

An Application: ***Balanced Parentheses***

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
 ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
  ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
    ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
    ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
      ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
      ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
        ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
        ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
            ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
            ^
```

Balancing Parentheses

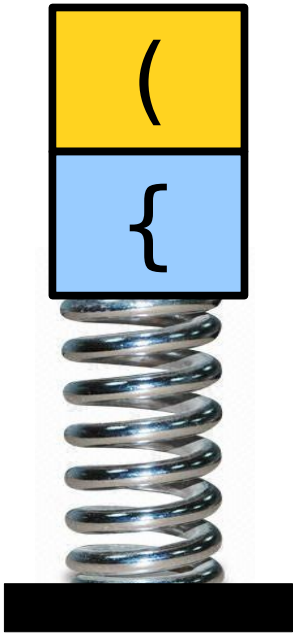
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```

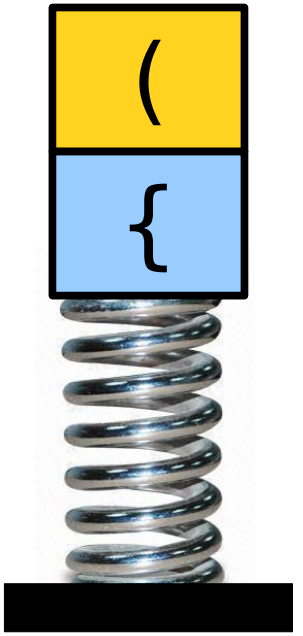
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
          ^
```



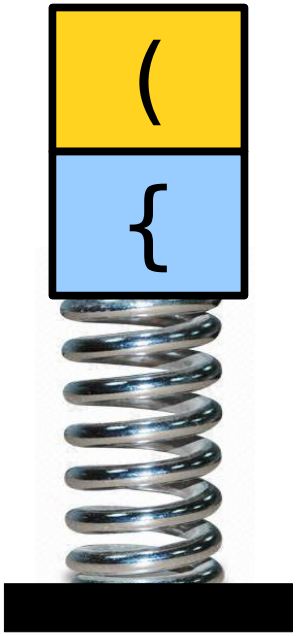
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                ^
```



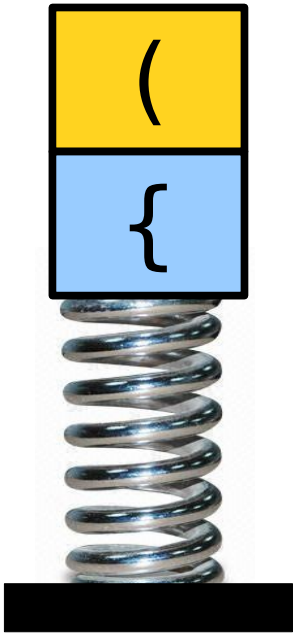
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                 ^
```



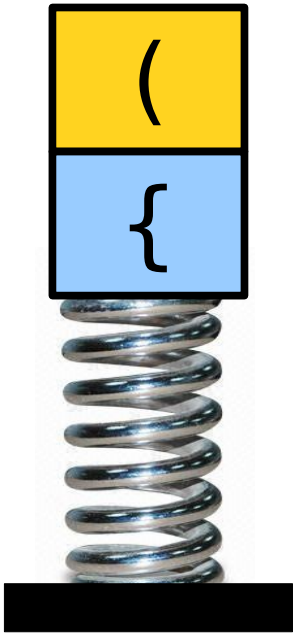
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                  ^
```



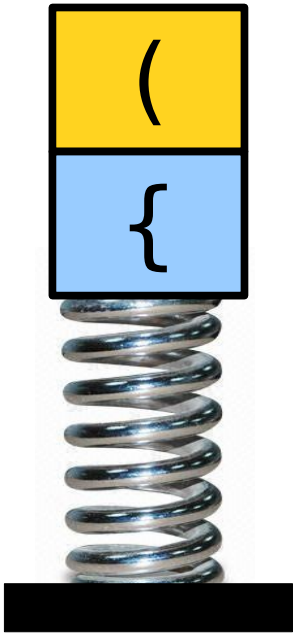
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                  ^
```



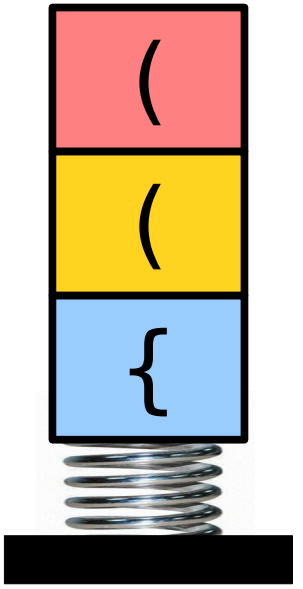
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                    ^
```



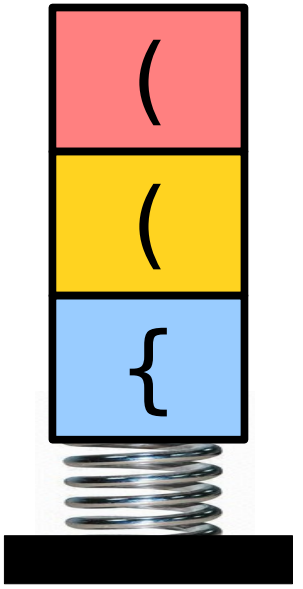
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                    ^
```



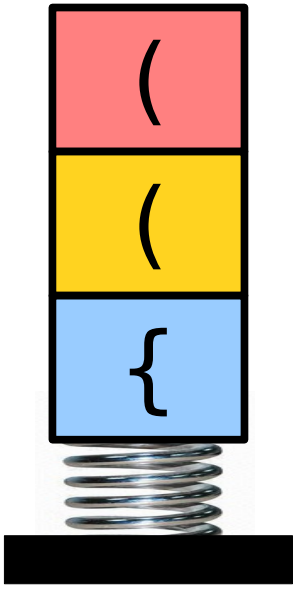
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                    ^
```



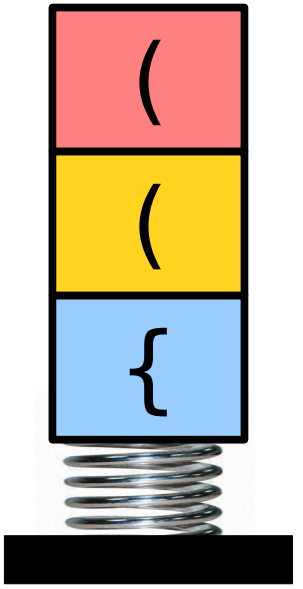
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                    ^
```



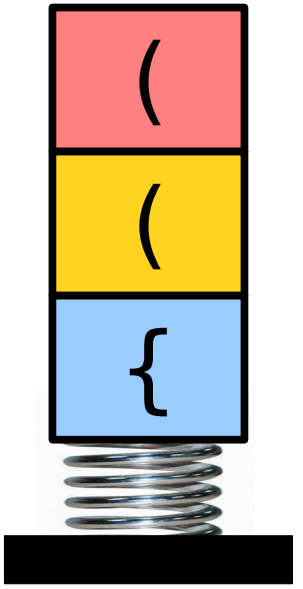
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                        ^
```



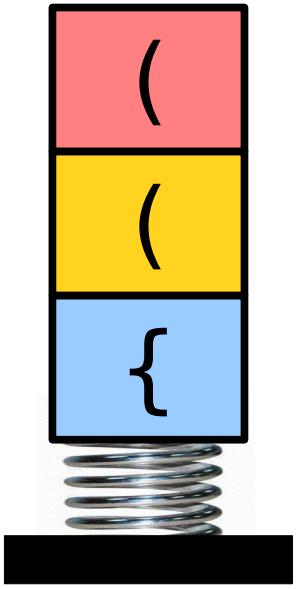
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                          ^
```



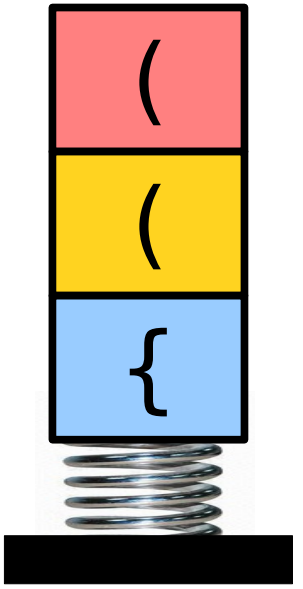
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                          ^
```



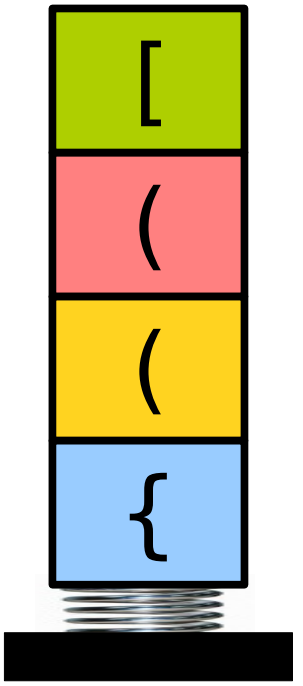
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                          ^
```



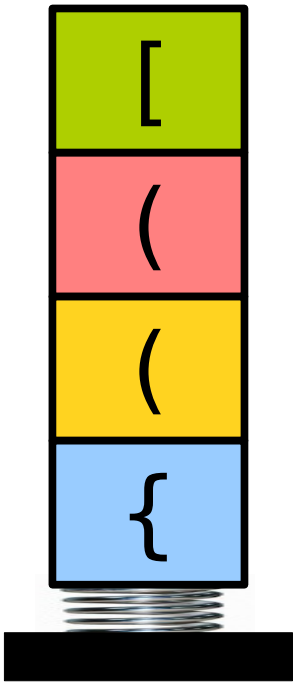
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                          ^
```



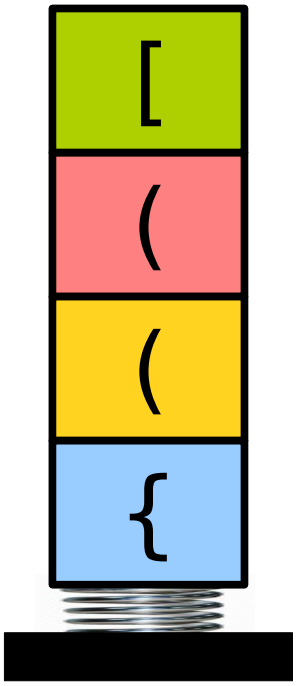
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                          ^
```



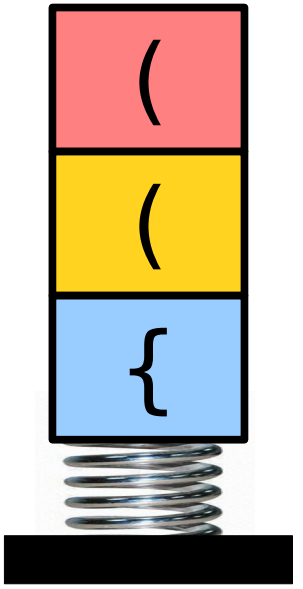
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                              ^
```



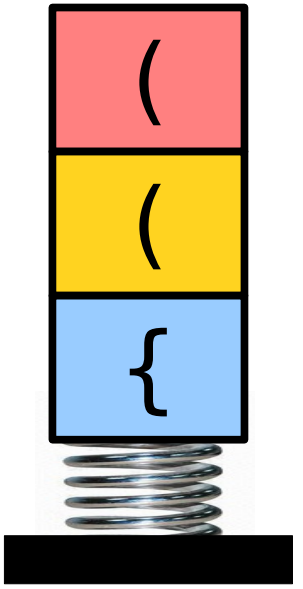
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                              ^
```



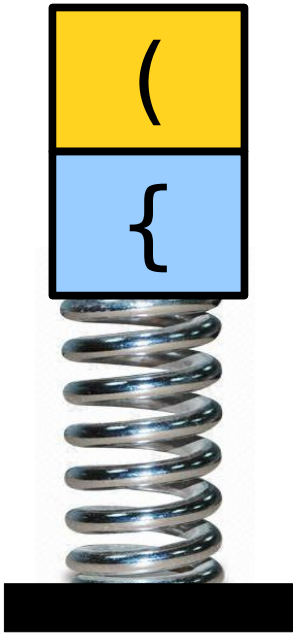
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                ^
```



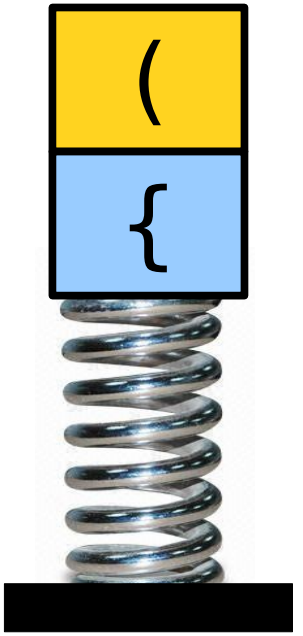
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                ^
```



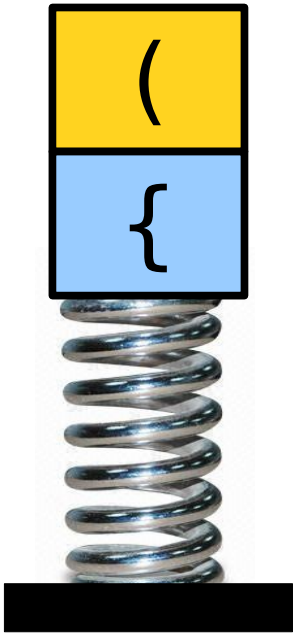
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                ^
```



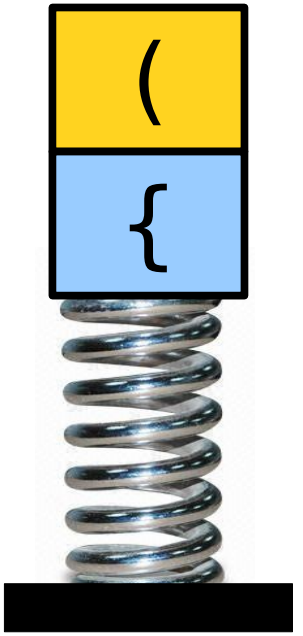
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                ^
```



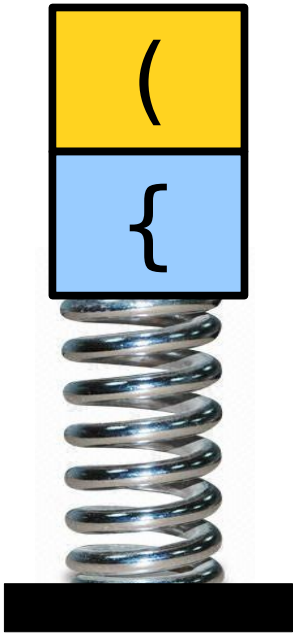
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                 ^
```



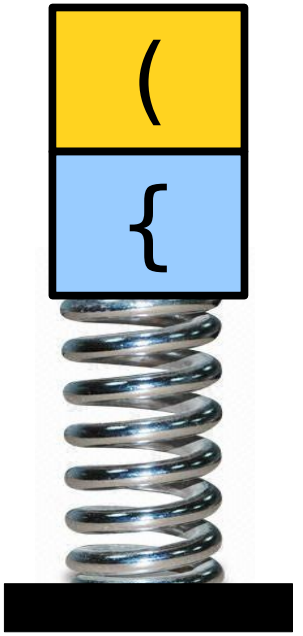
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                  ^
```



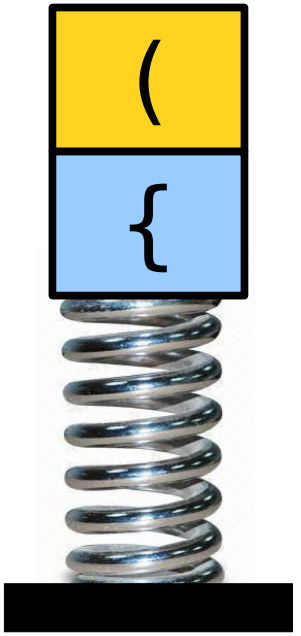
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                  ^
```



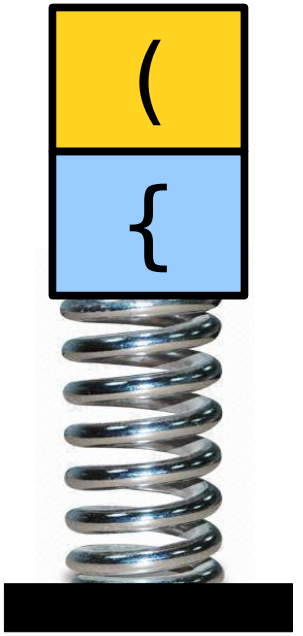
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                  ^
```



Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                    ^
```



Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                  ^
```

Balancing Parentheses

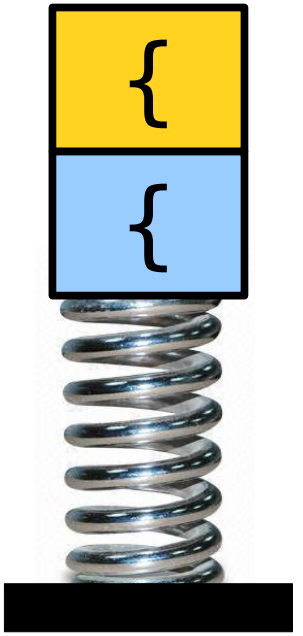
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                      ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                      ^
```

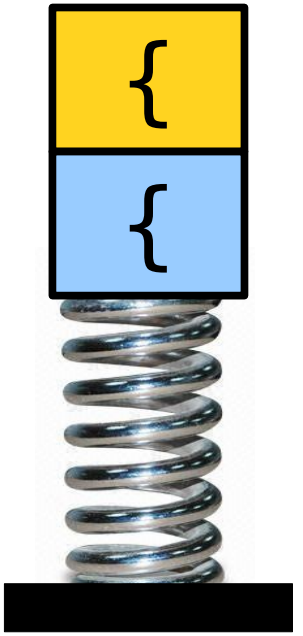
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                      ^
```



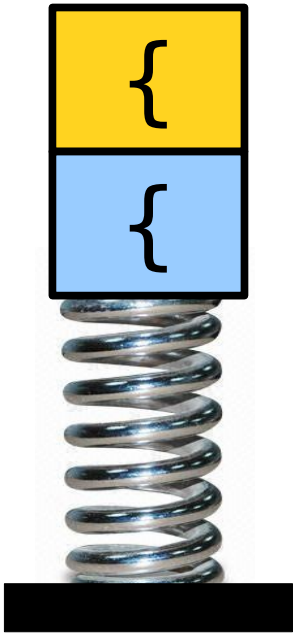
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                      ^
```



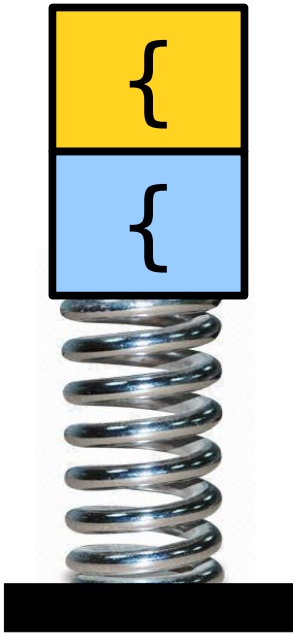
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                        ^
```



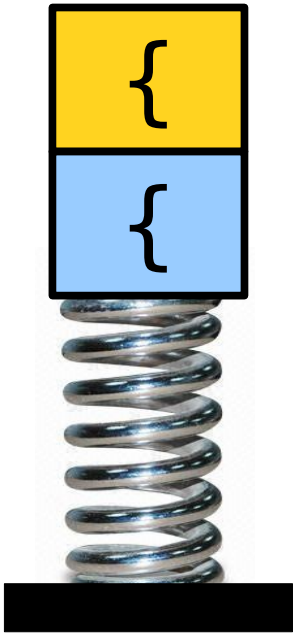
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                        ^
```



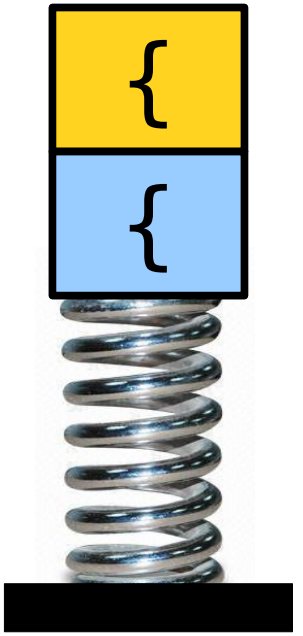
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                        ^
```



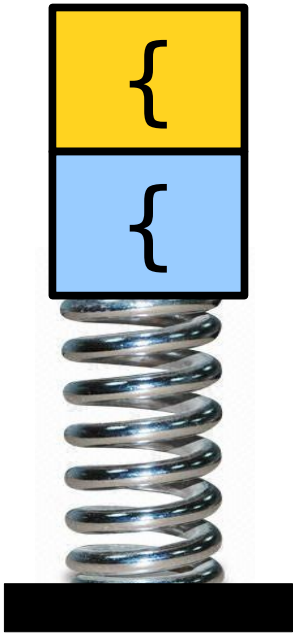
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                          ^
```



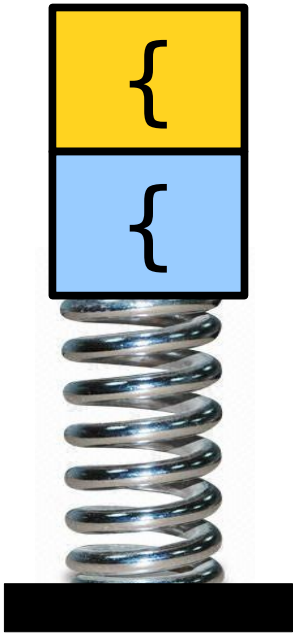
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                          ^
```



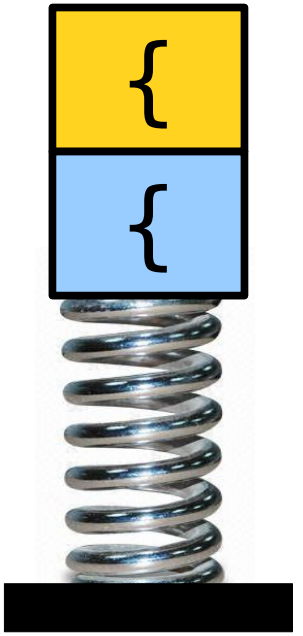
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                          ^
```



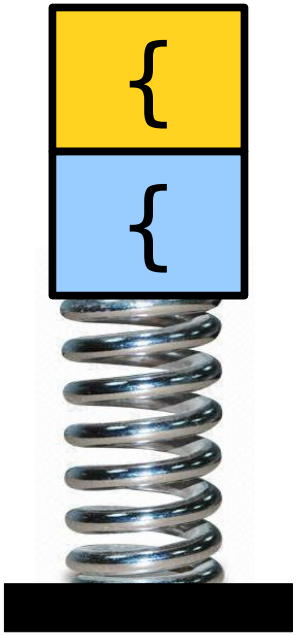
Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                              ^
```



Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                              ^
```



Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                              ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                              ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                                ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                                ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                                ^
```

Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

Balancing Parentheses

([ )]

Balancing Parentheses

([ )]
^

Balancing Parentheses

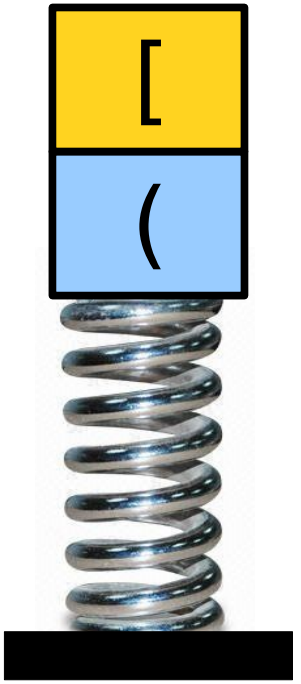
```
( [ ) ]
^
```

Balancing Parentheses

```
( [ ) ]
  ^
```

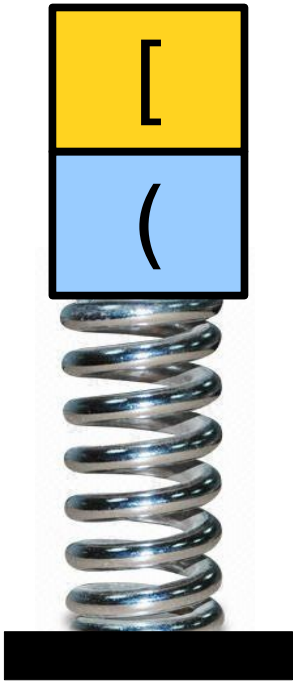
Balancing Parentheses

```
( [ ) ]
  ^
```



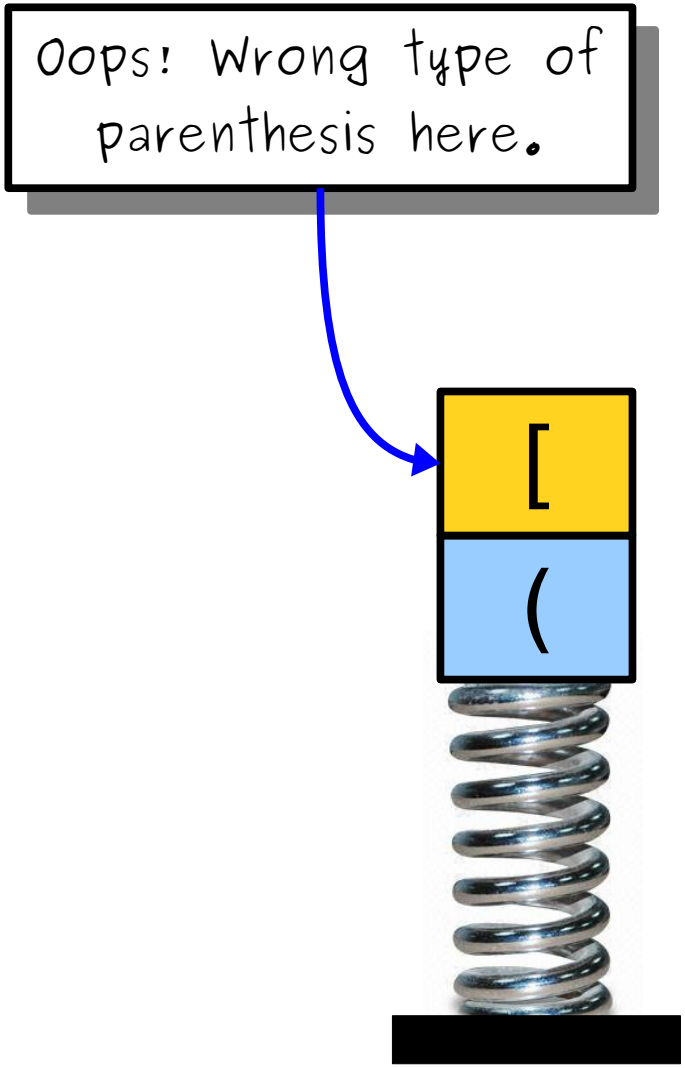
Balancing Parentheses

```
( [ ) ]
    ^
```



Balancing Parentheses

```
( [ ) ]
    ^
```



Balancing Parentheses

((

Balancing Parentheses

((
^

Balancing Parentheses

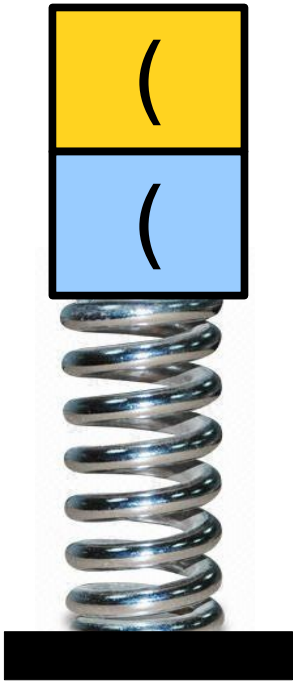
((
^

Balancing Parentheses

((
  ^

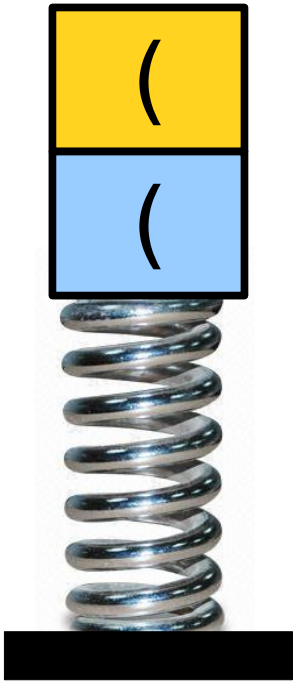
Balancing Parentheses

((
  ^



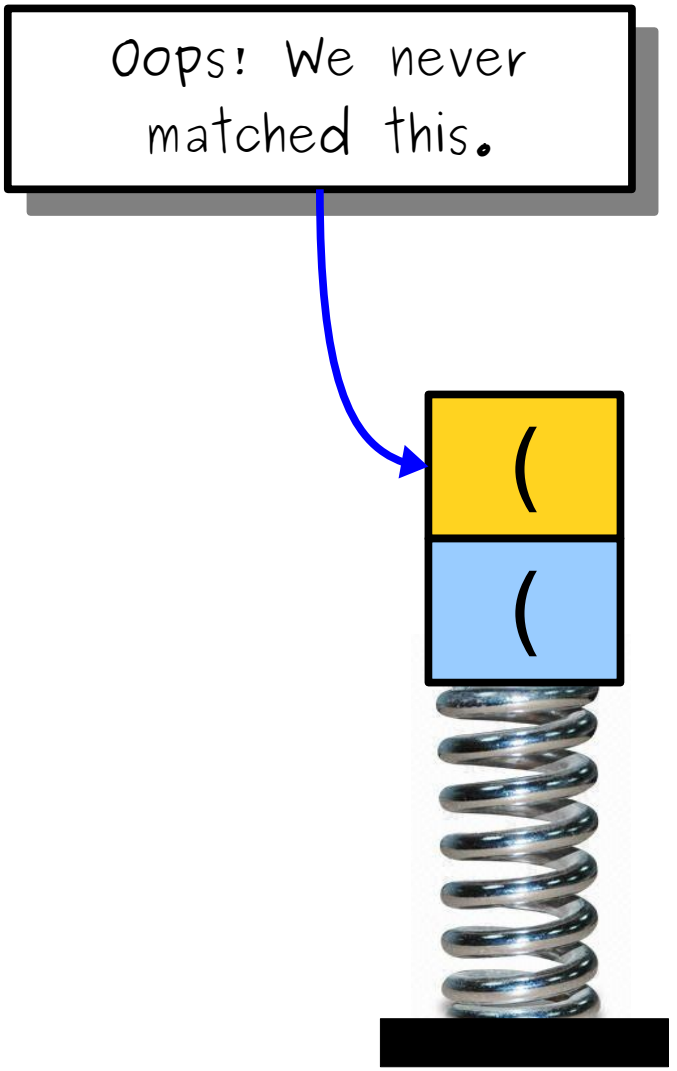
Balancing Parentheses

((



Balancing Parentheses

((



Balancing Parentheses

)

Balancing Parentheses

)
^

Balancing Parentheses

Oops! There's nothing on the stack to match.

```
)
^
```

Our Algorithm

- For each character:
  - If it’s an open parenthesis or brace, push it onto the stack.
  - If it’s a close parenthesis or brace:
    - If the stack is empty, report an error.
    - If the character doesn’t pair with the character on top of the stack, report an error.
- At the end, return whether the stack is empty (nothing was left unmatched.)

Great Exercise: Reimplement this function purely using the *call stack* and *recursion* rather than a `Stack<char>`.

More `Stack` Applications

- Stacks show up all the time in ***parsing***, recovering the structure in a piece of text.
  - Often used in natural language processing; take CS224N for details!
  - Used all the time in compilers – take CS143 for details!
  - There's a deep theorem that says that many structures appearing in natural language are perfectly modeled by operations on stacks; come talk to me after class if you're curious!
- They're also used as building blocks in larger algorithms for doing things like
  - making sure a city's road networks are navigable (finding *strongly connected components;* take CS161 for details!) and
  - searching for the best solution to a problem – stay tuned!

Time-Out for Announcements!

Assignment 1

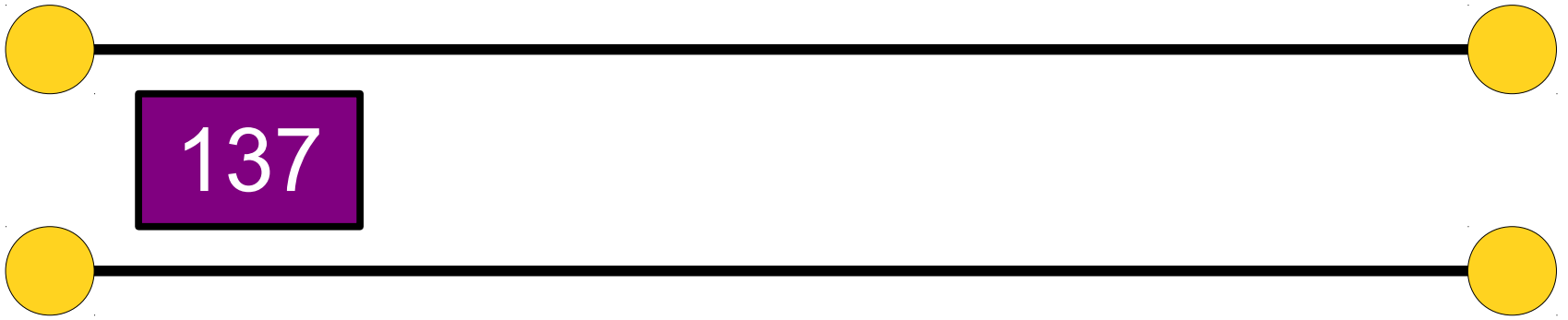
- Assignment 1 is due this Friday at the start of class.
- Have questions?
  - Stop by the LaIR!
  - Ask on Piazza!
  - Email your section leader, once section assignments go out.
- Heads-up for planning purposes: the LaIR will be closed this Sunday, but will be operating as usual on Monday.

```
lecture.pop();
```

Queue

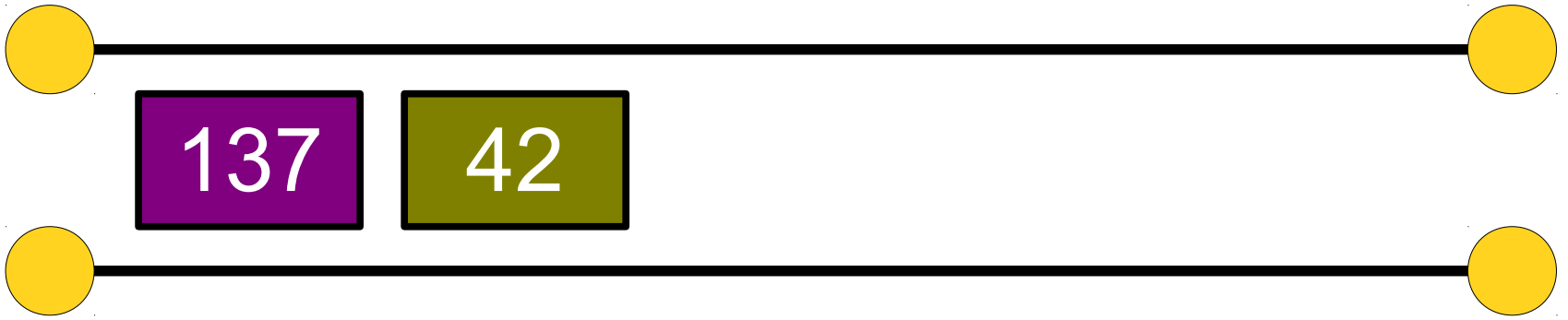
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



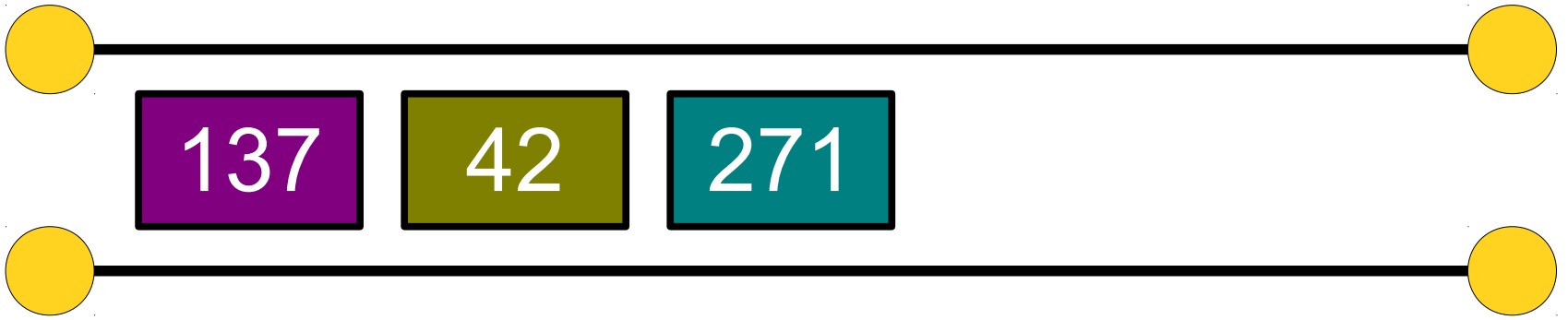
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



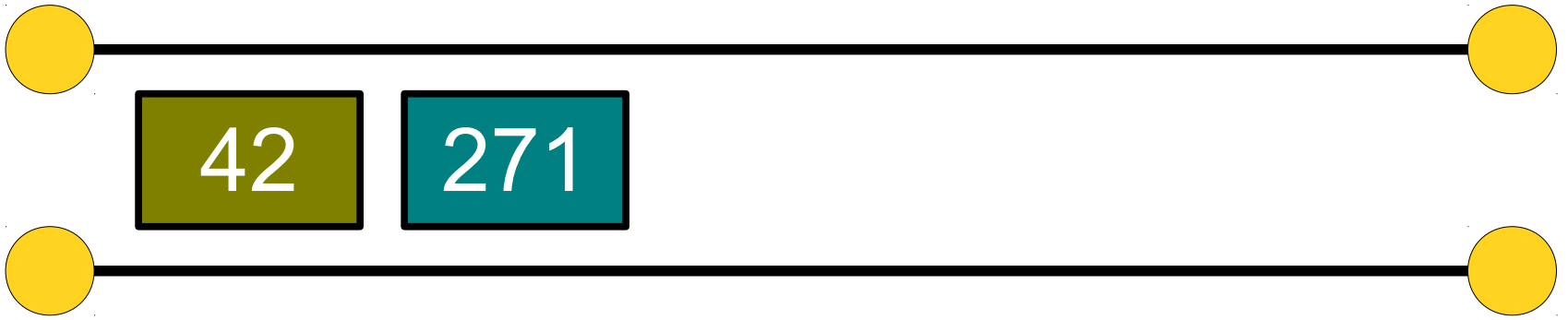
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



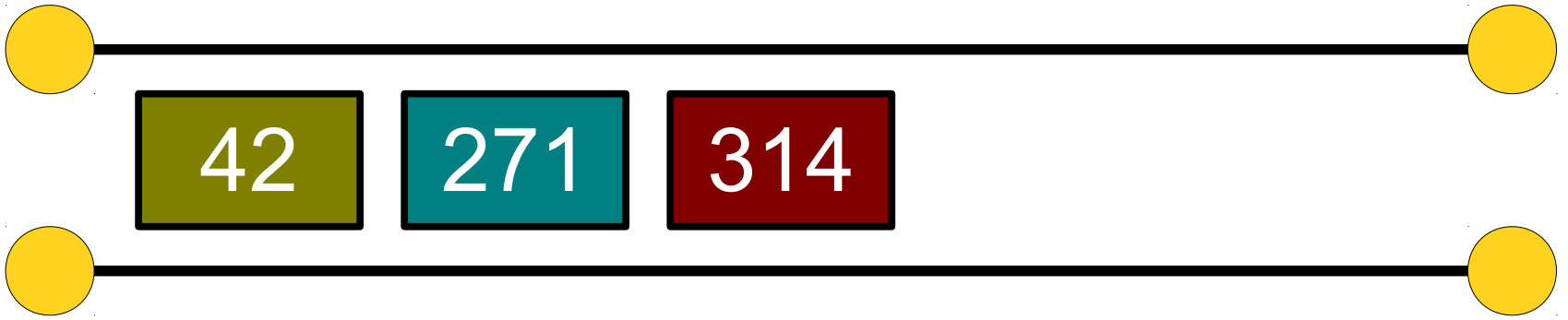
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



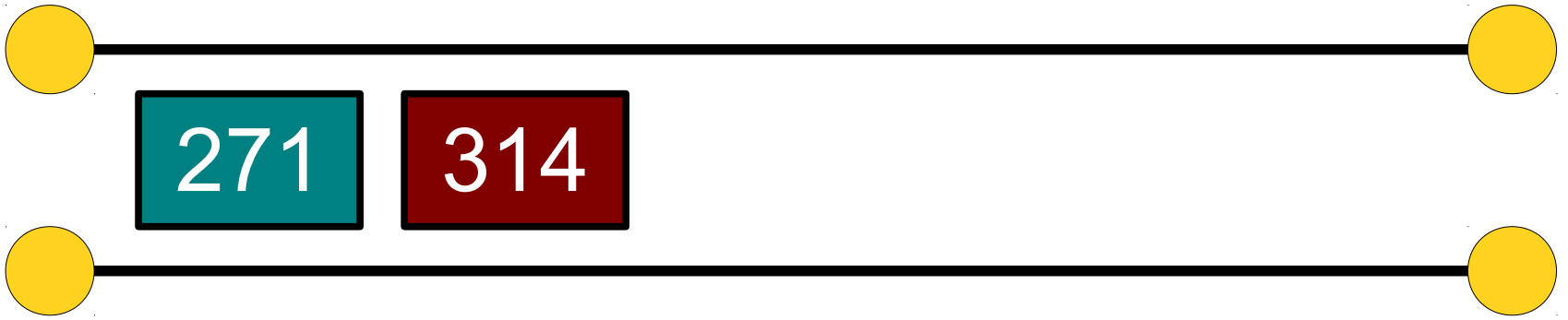
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



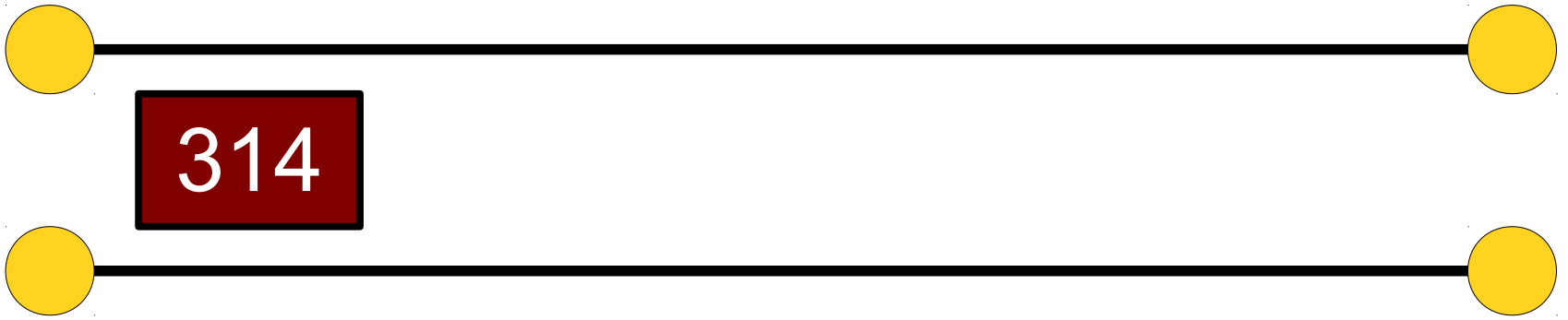
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



An Application: ***Looper***

Loopers

- A ***looper*** is a device that records sound or music, then plays it back over and over again (in a loop).
- These things are way too much fun, *especially* if you're not a very good musician. 😀
- Let's make a simple looper using a `Queue`.

Building our Looper

- Our looper will read data files like the one shown to the left.
- Each line consists of the name of a sound file to play, along with how many milliseconds to play that sound for.
- We'll store each line using the `SoundClip` type, which is defined in our C++ file.

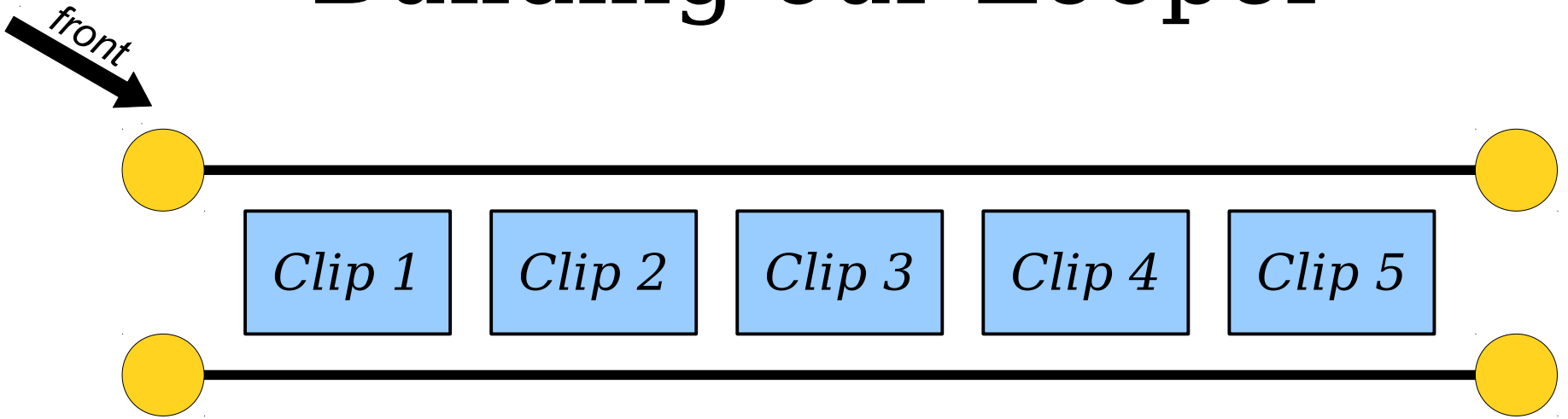
```
B2.wav 500
B3.wav 333.34
Gb3.wav 166.66
B2.wav 500
G2.wav 500
A2.wav 500
B2.wav 333.34
A2.wav 166.66
D3.wav 500
```

Building our Looper

Building our Looper

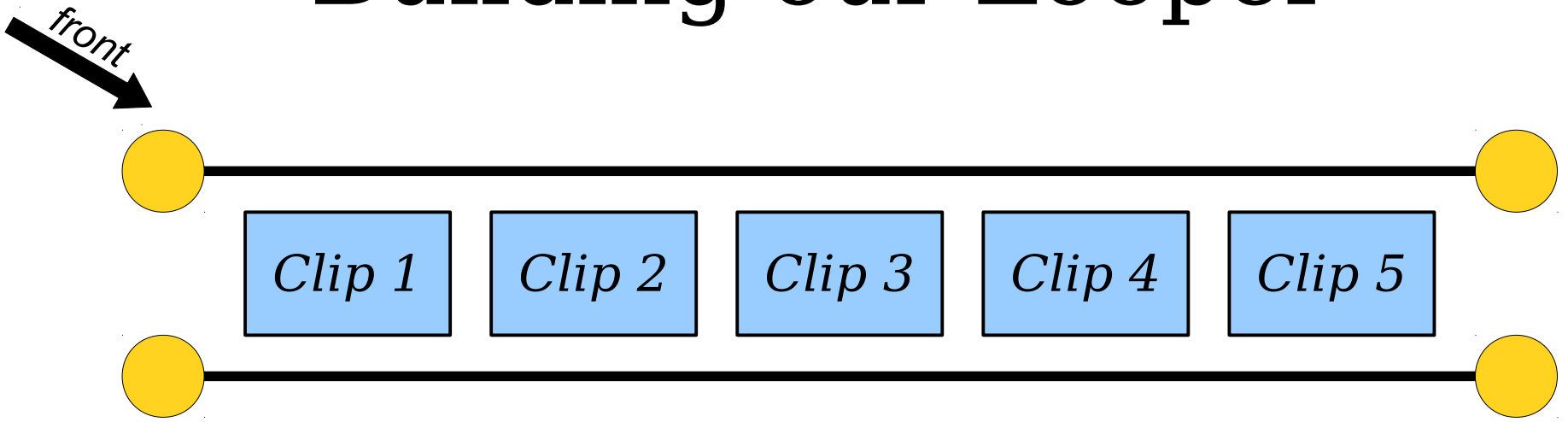
```
Queue<SoundClip> loop = loadLoop(/* … */);
```

Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
```

Building our Looper

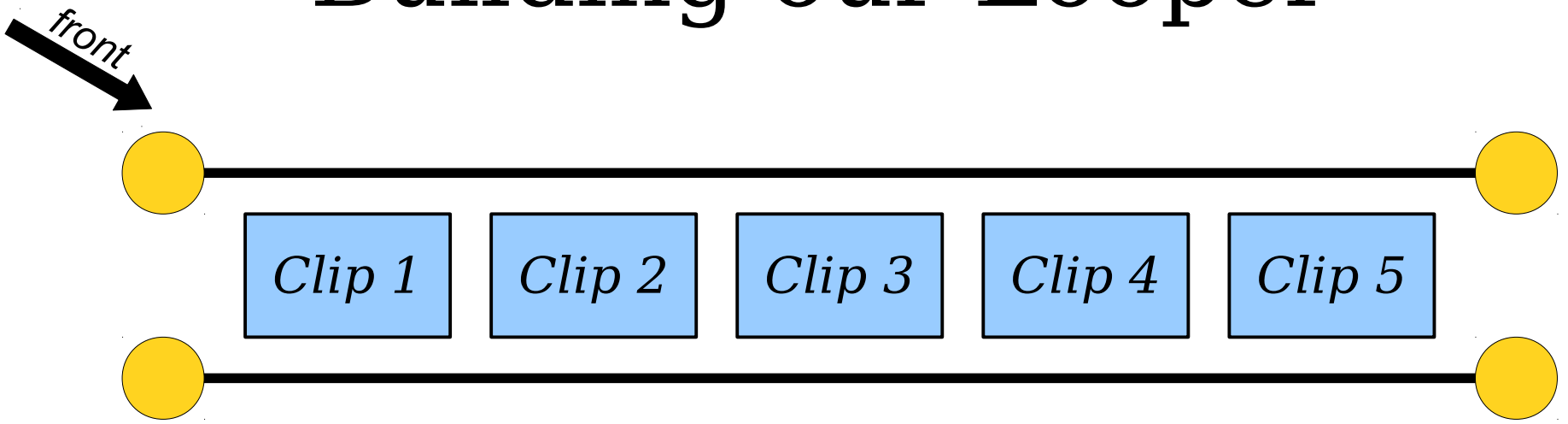



```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {



}
```

Building our Looper

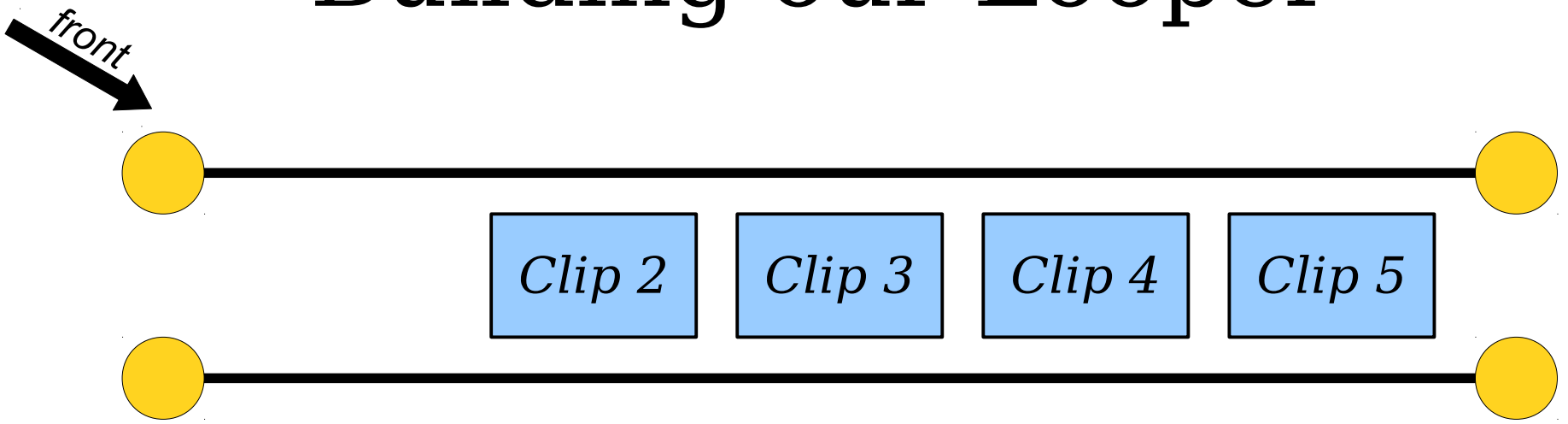



```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();



}
```

Building our Looper



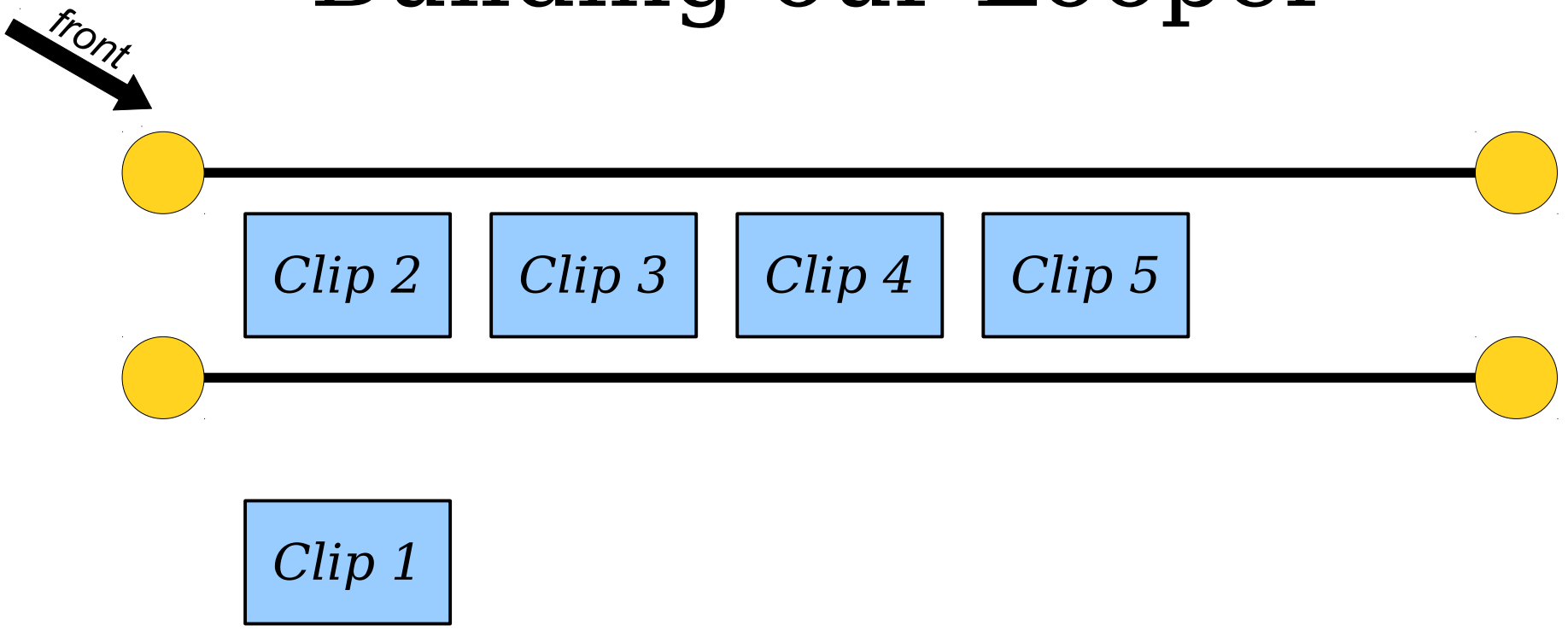
Clip 1

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();



}
```

Building our Looper

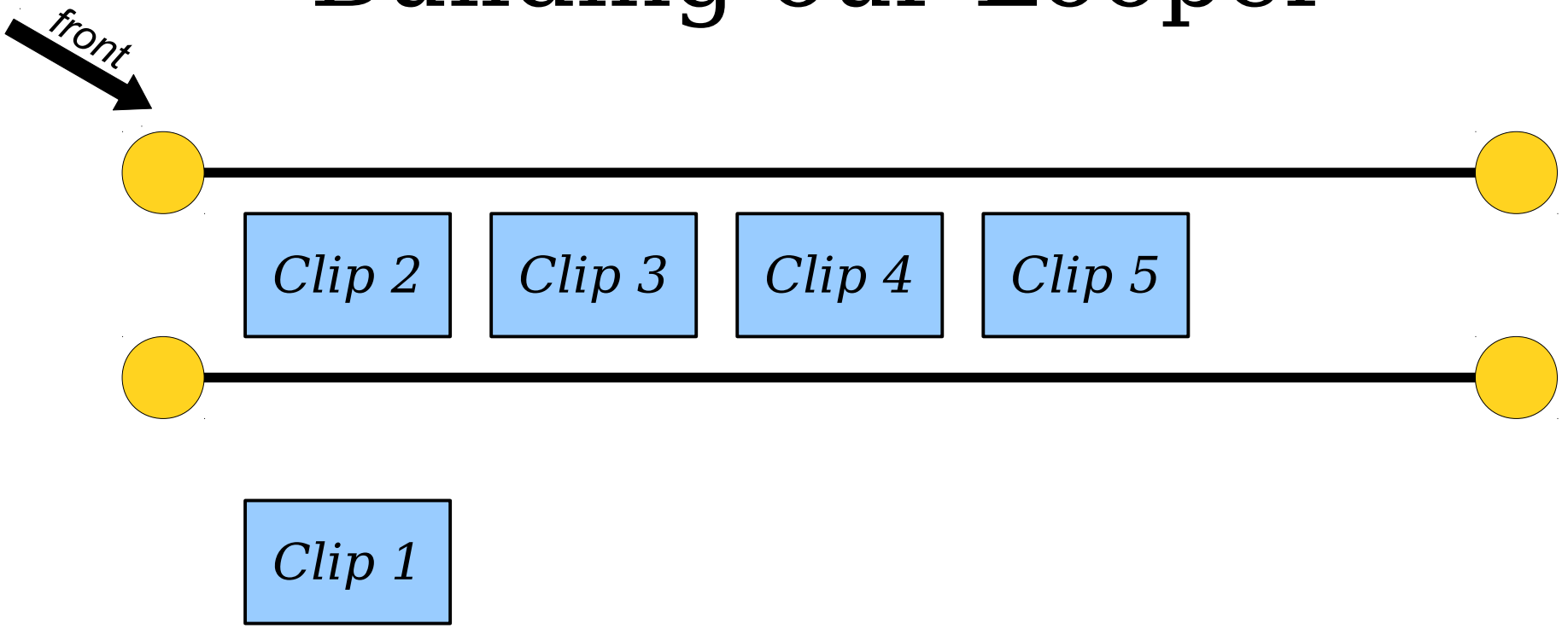



```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();



}
```

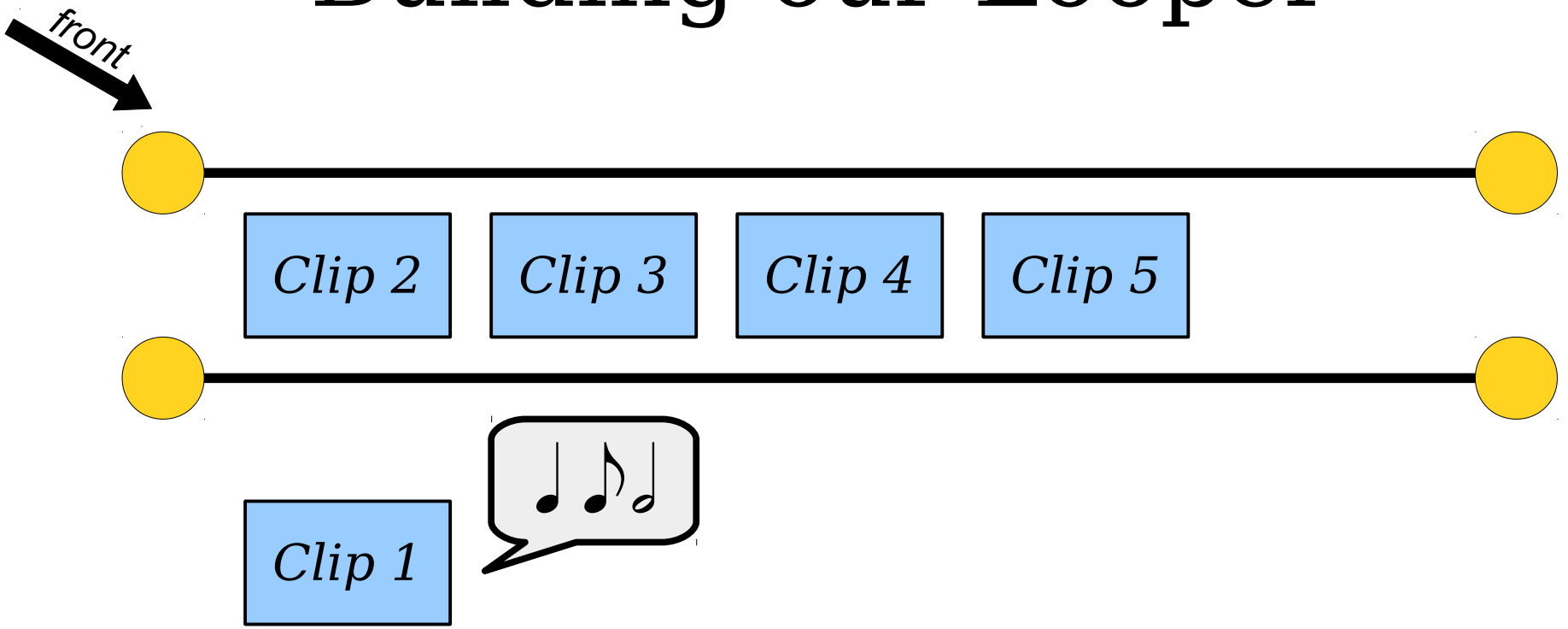
Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);


}
```

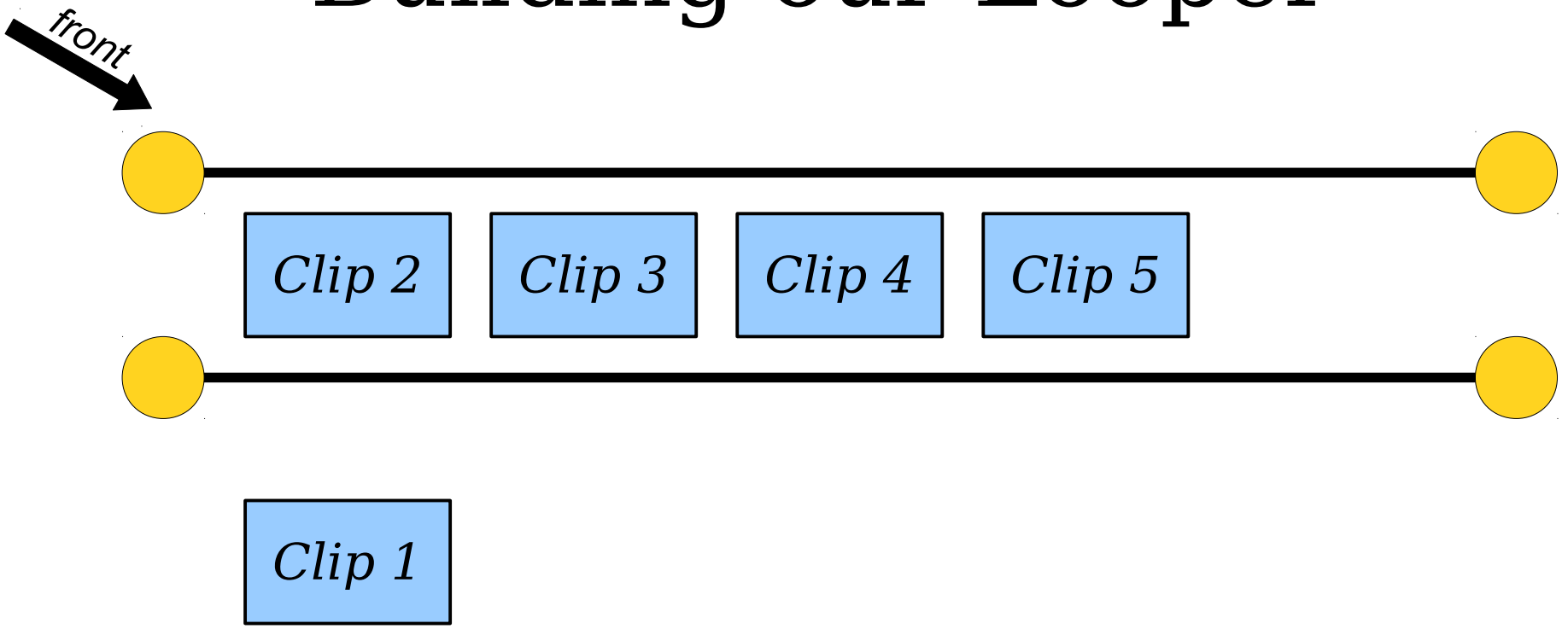
Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);


}
```

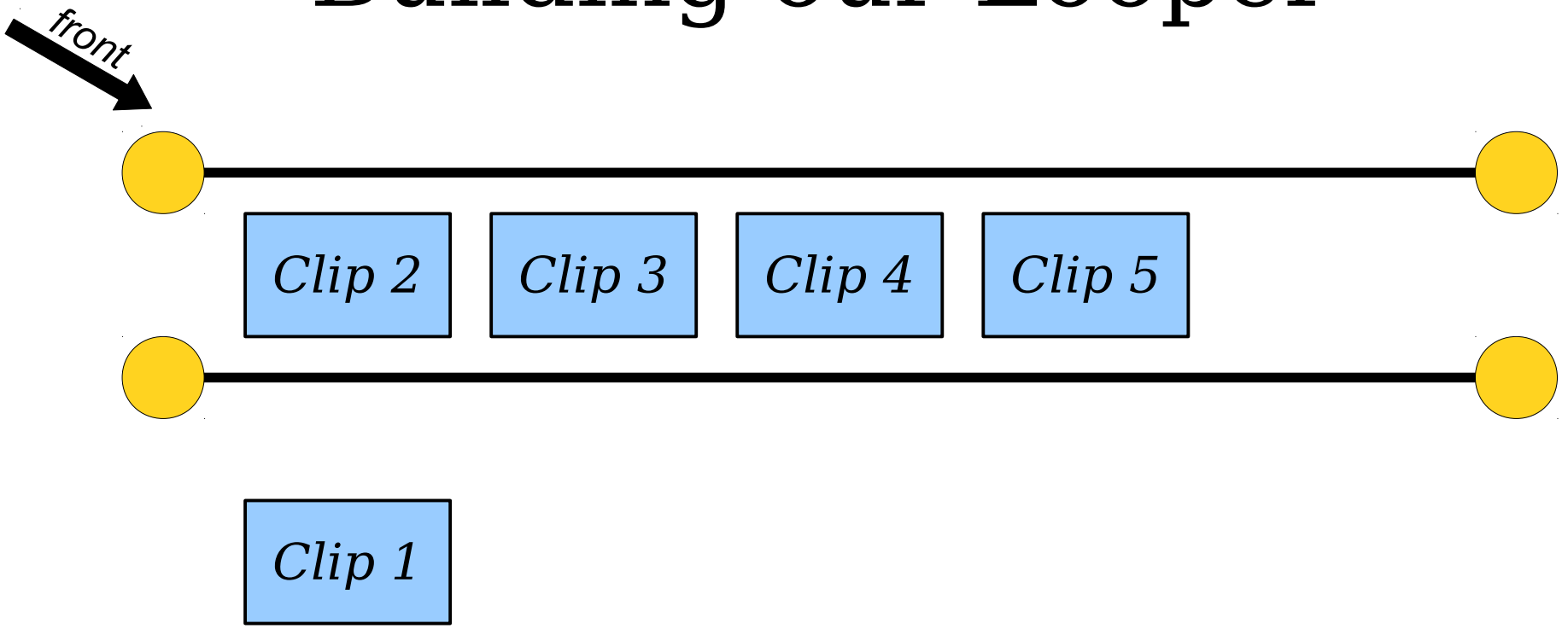
Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);


}
```

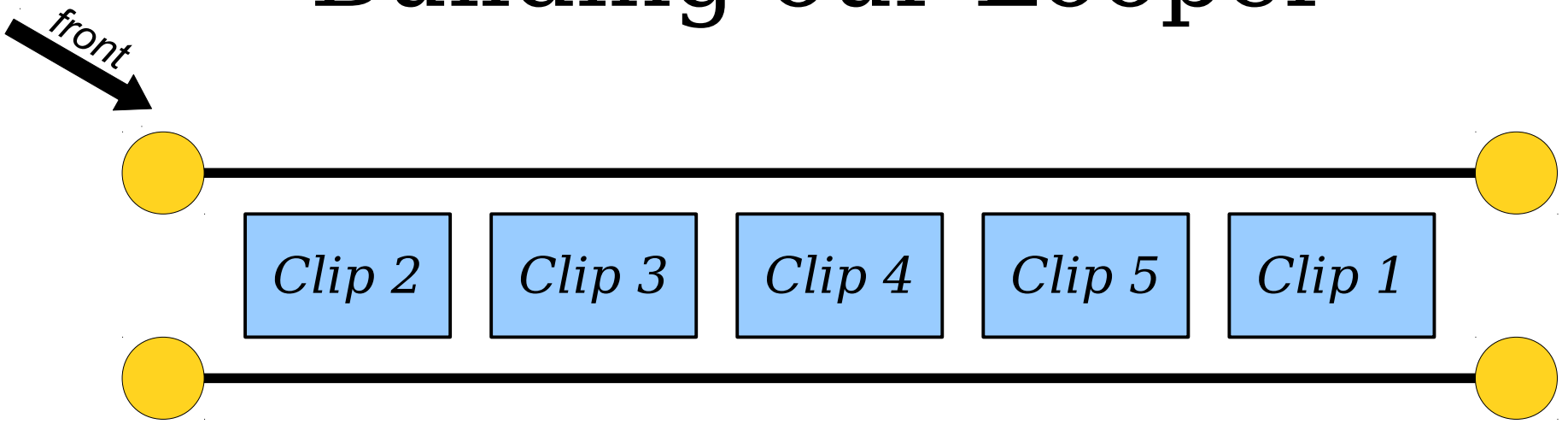
Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

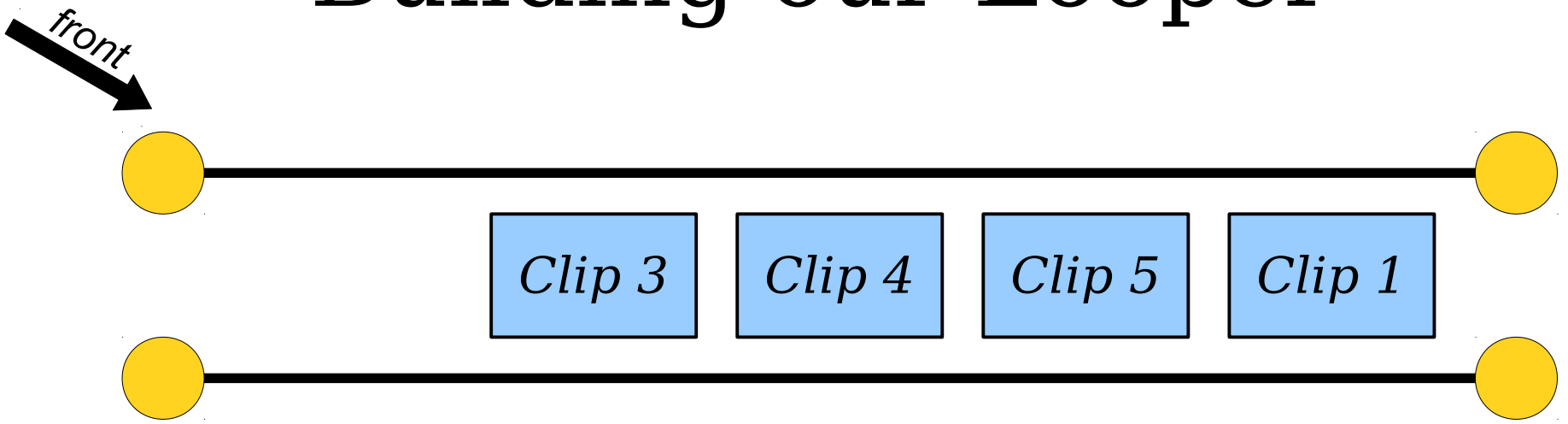
Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper

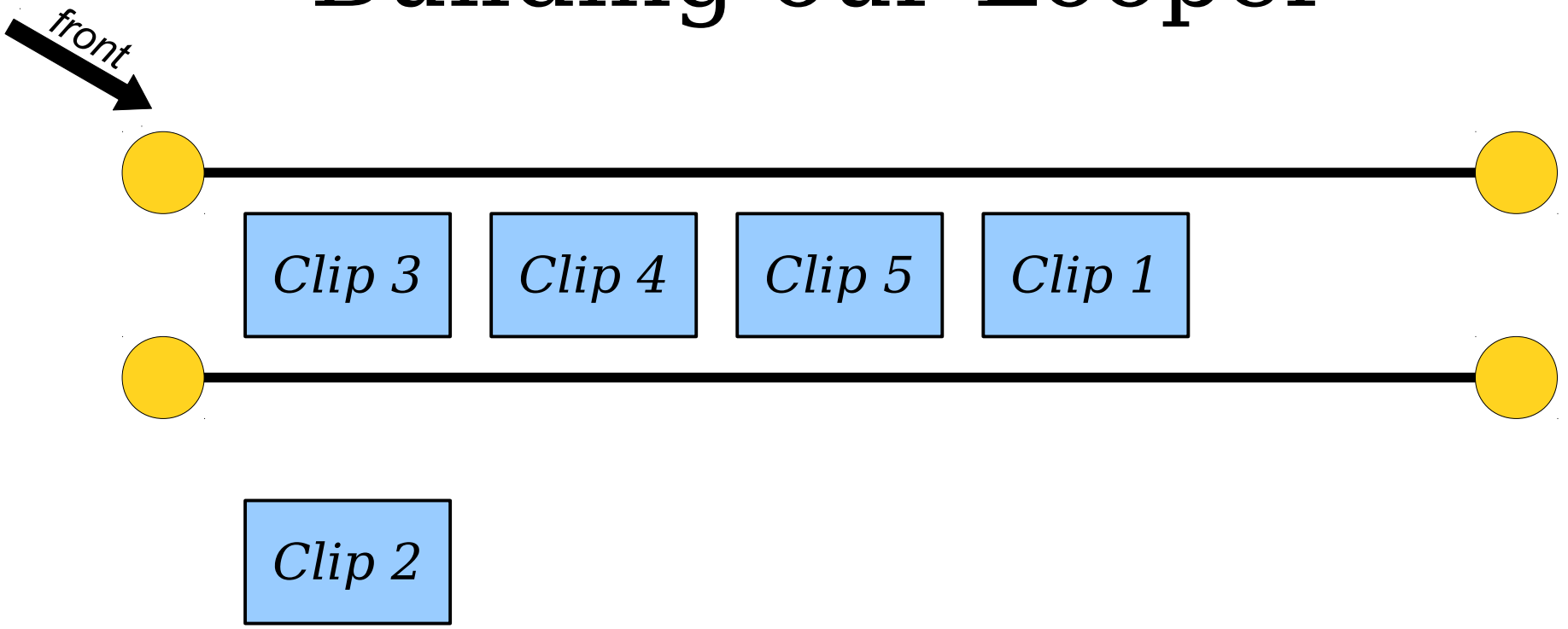


Clip 2

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

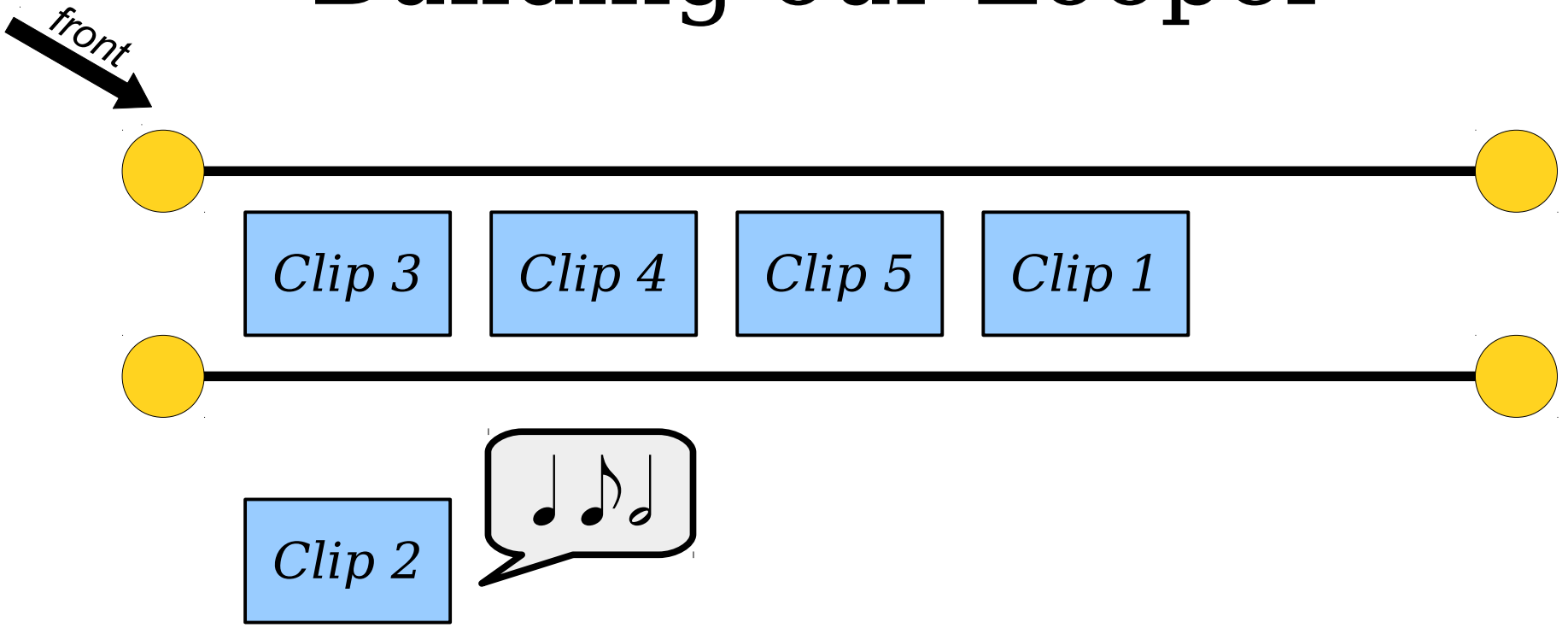
Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

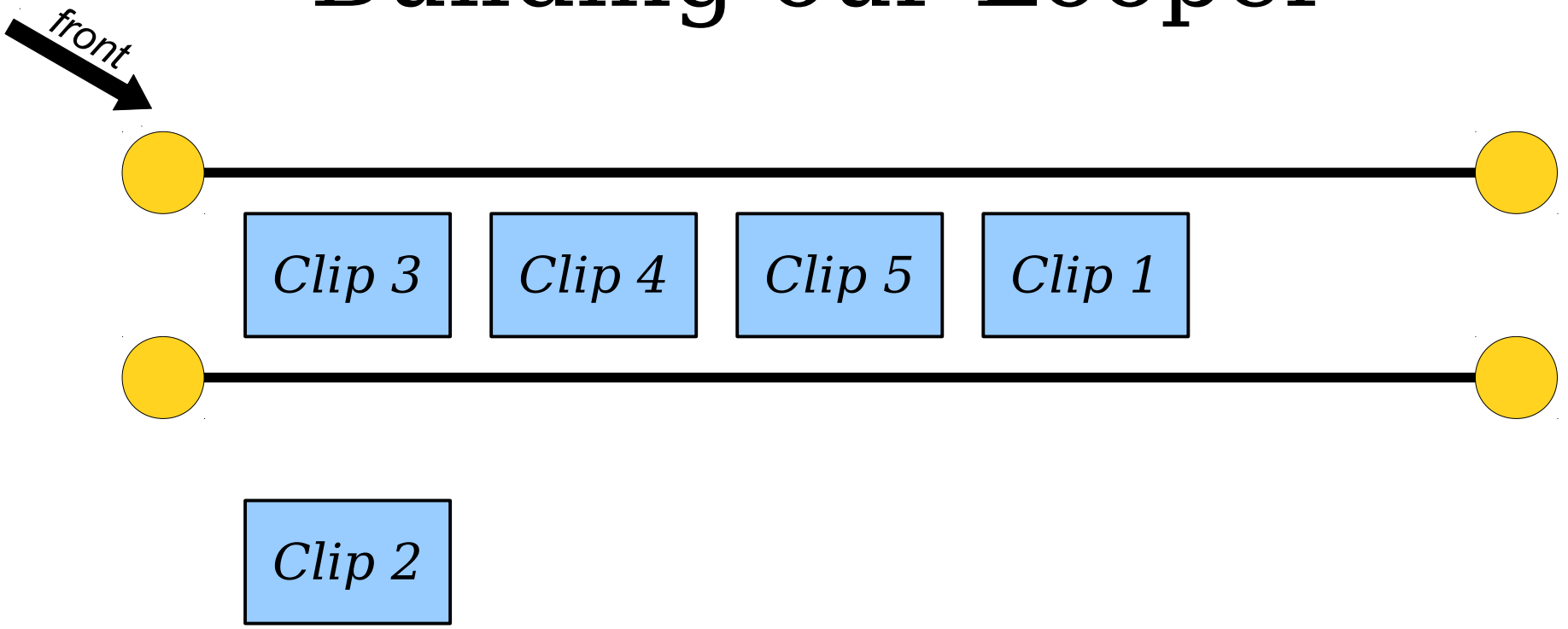
Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

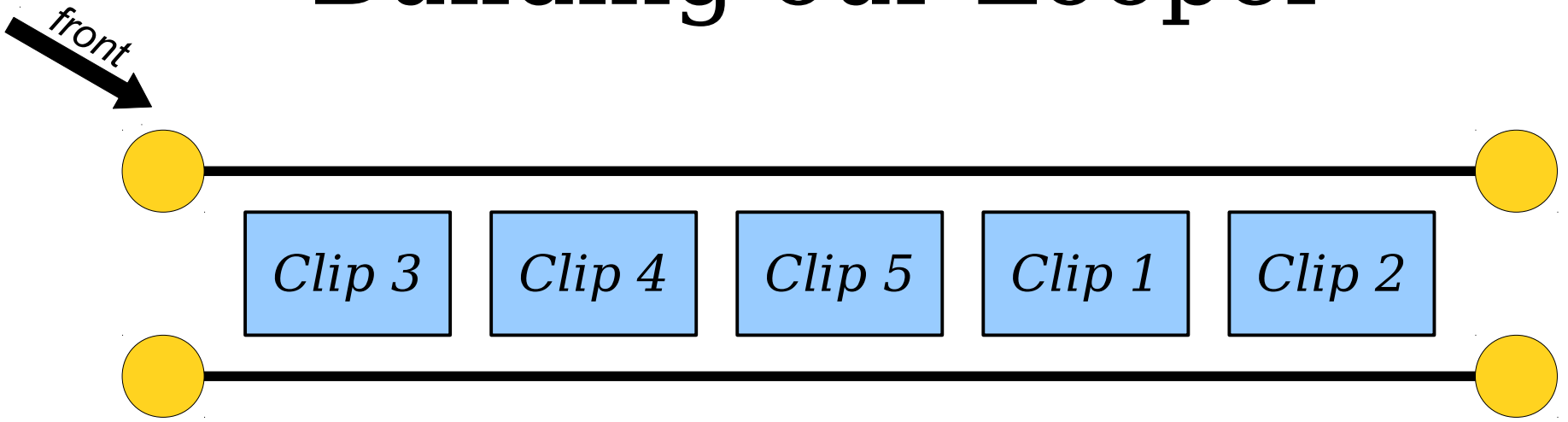
Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Your Action Items

- ***Read Chapter 5.2 and 5.3.***
  - These sections cover more about the `Stack` and `Queue` type, and they're great resources to check out.
- ***Attend your first section!***
  - How exciting!
- ***Finish Assignment 1.***
  - Read the style guide up on the course website for more information about good programming style.
  - Review the Assignment Submission Checklist to make sure your code is ready to submit.

Next Time

- ***Associative Containers***
  - Data sets aren’t always linear!
- ***HashMaps and HashSets***
  - Two ways to organize information.