Stacks and Queues

Collection Classes Part 2

Chris Piech

CS 106B
Lecture 3
Jan 11, 2015

CS + Social Good

CS+SOCIAL GOOD

Wednesday 3:30 – 5:20pm

See website for signup link!

Computer Forum Career Fair

STANFORD COMPUTER FORUM

Computer Forum Career Fair

When:	Wed, Jan 13th
What:	Computer Forum Career Fair
Date:	Wednesday, January 14
Time:	Time: 11am - 4pm
Location:	Lawn between the Gates CS Building and the Electrical Engineering Buildings
Description:	The Computer Forum Career Fairs enable Stanford Engineering students, specifically CS and EE, to get a head start on careers and internships.


Keep a job for me… Im coming!

Collections

Vector

Grid

Map

Stack

Queue

Set

Collections

Vector

Grid

Map

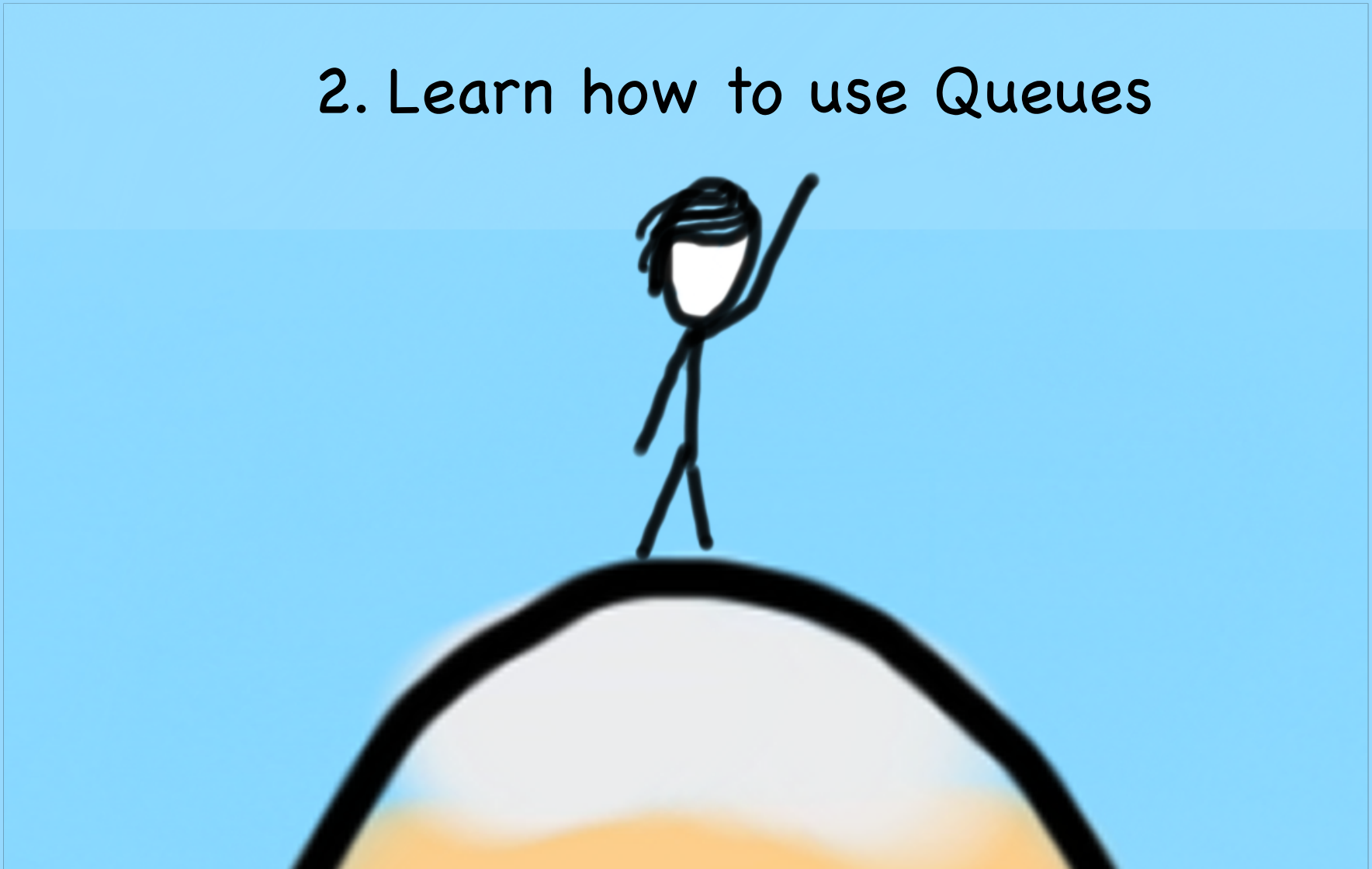
Stack

Queue

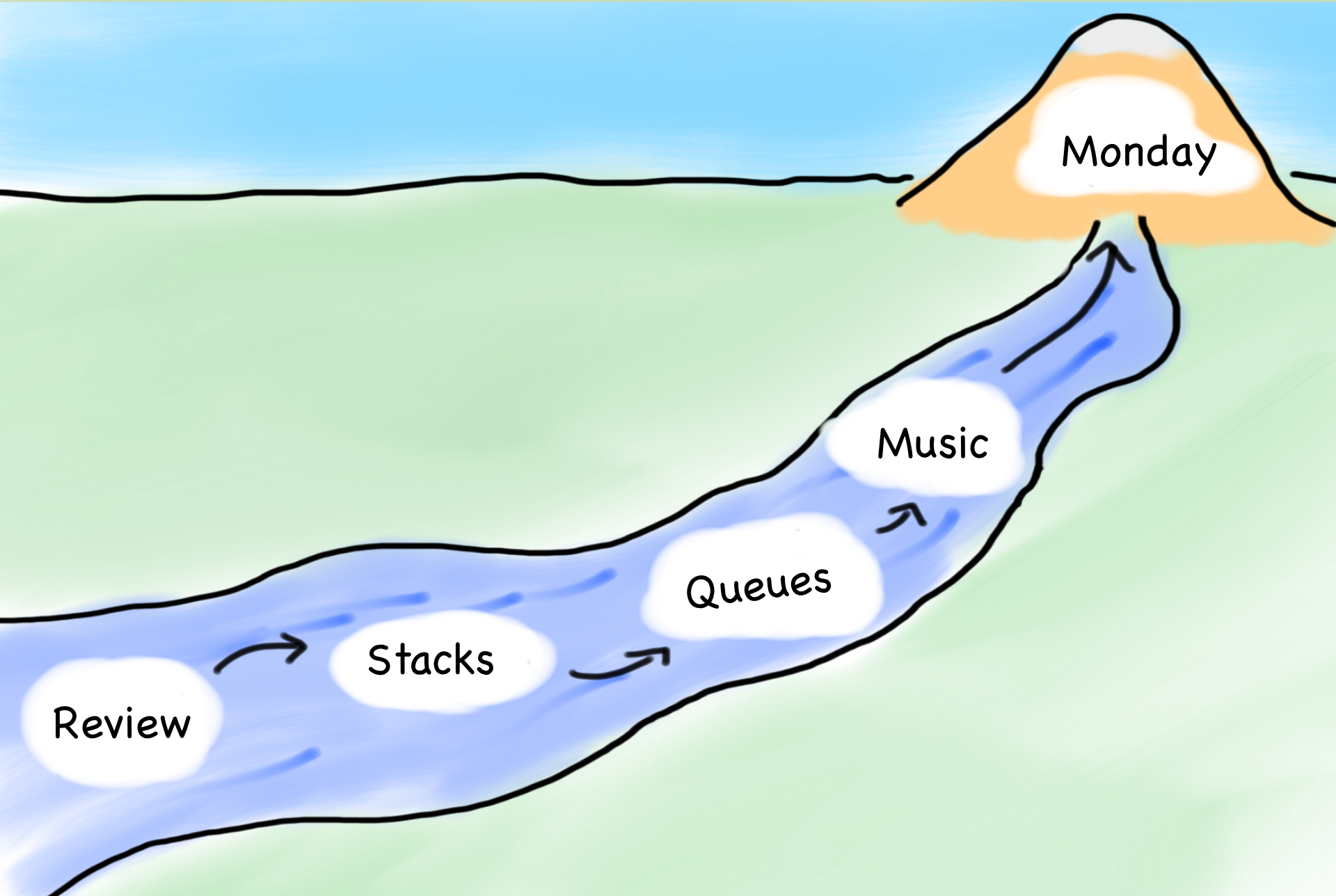
Set

Today's Goals

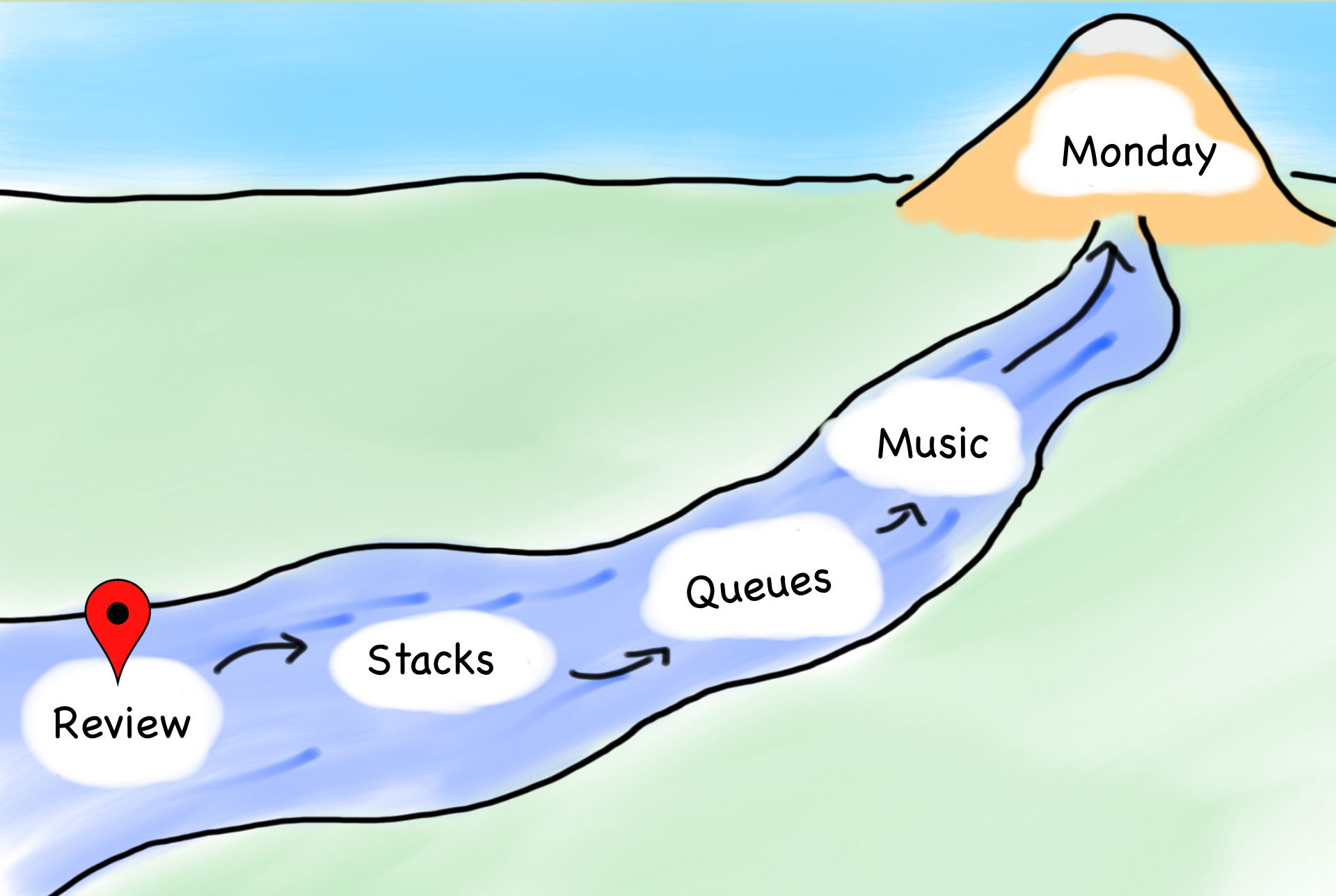
1. Learn how to use Stacks
2. Learn how to use Queues



Today's Goals



Today's Goals



Collection Classes

These classes contain other objects and are called *container* or *collection classes.*

Vector	Grid	Stack	Queue	Map	Set

Here are some general guidelines for using these classes:

- These classes represent *abstract data types* whose details are hidden.
- Each class requires type parameters.
- Any memory for these objects is freed when its declaration scope ends.
- Assigning one value to another *copies* the entire structure.
- To avoid copying, these structures are usually passed by reference.

Template Classes

- The collection classes are implemented as *template classes,* which make it possible for an entire family of classes to share the same code.
- Instead of using the class name alone, the collection classes require a type parameter that specifies the element type. For example, `Vector<int>` represents a vector of integers. Similarly, `Grid<char>` represents a two-dimensional array of characters.
- It is possible to nest classes, so that, for example, you could use the following definition to represent a list of chess positions:

```
Vector<Grid<char> > chessPositions;
```

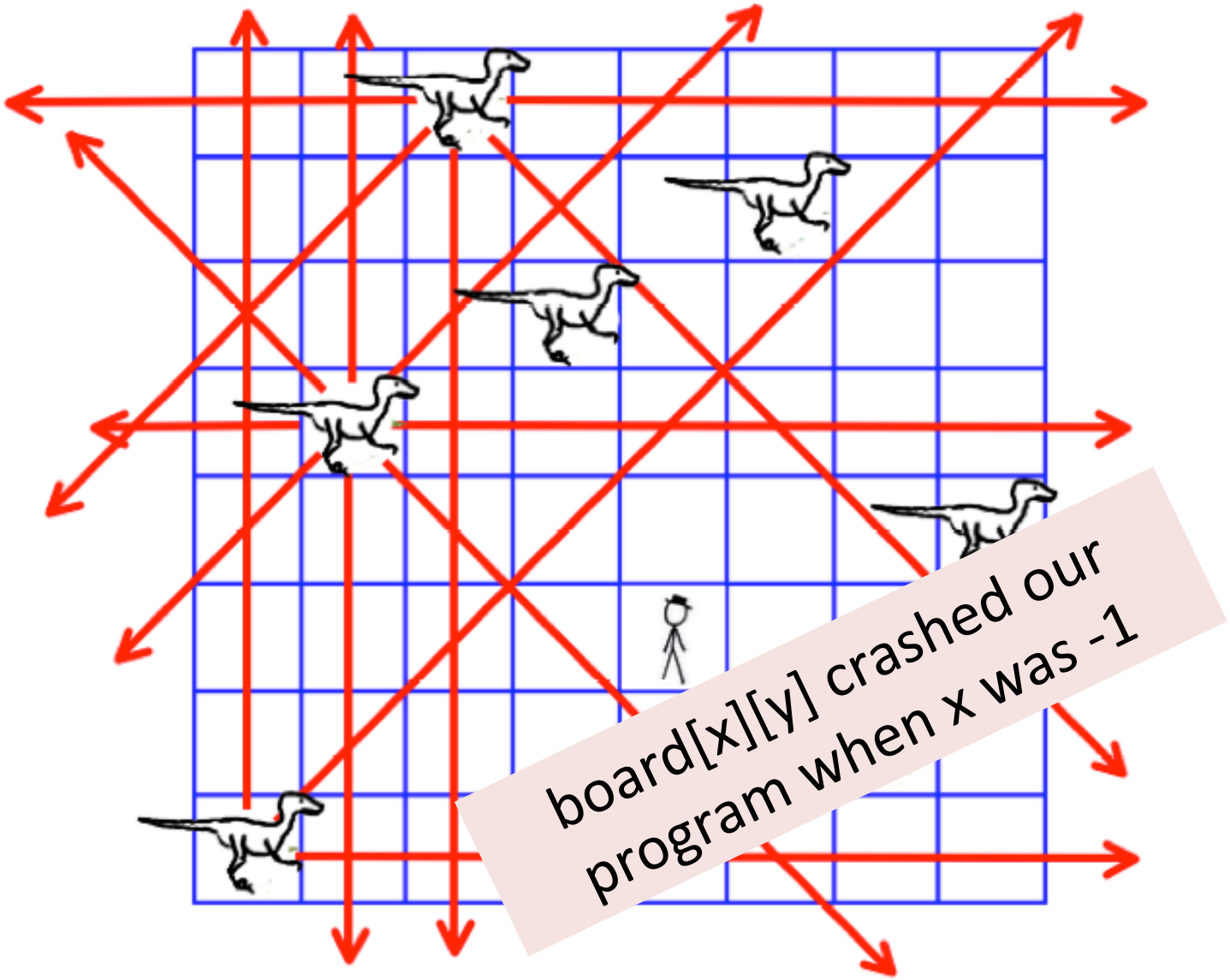
Review: Read File to Vector

```
/*
 * Function: readEntireFile
 * Usage: readEntireFile(is, lines);
 * ---------------------------------
 * Reads the entire contents of the specified input stream
 * into the string vector lines.  The client is responsible
 * for opening and closing the stream
 */

void readEntireFile(istream & is, Vector<string> & lines) {
   lines.clear();
   string line;
   while (getline(is, line)) {
      lines.add(line);
   }
}
```

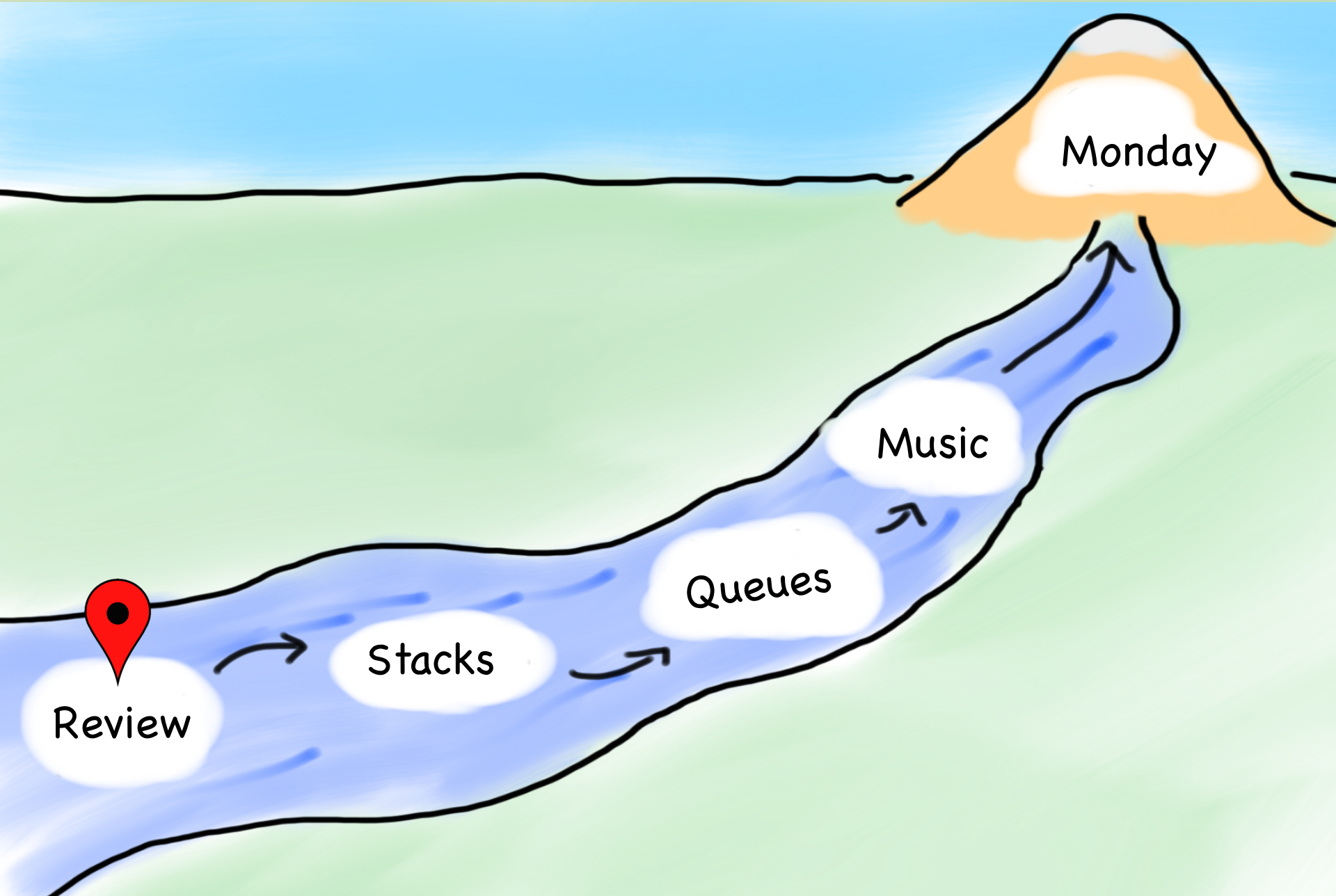
Review: Read File to Vector

```
/*
 * Function: readEntireFile
 * Usage: readEntireFile(is, lines);
 * ---------------------------------
 * Reads the entire contents of the specified input stream
 * into the string vector lines.  The client is responsible
 * for opening and closing the stream
 */

void readEntireFile(istream & is, Vector<string> & lines) {
   lines.clear();
   string line;
   while (getline(is, line)) {
      lines.add(line);
   }
}
```

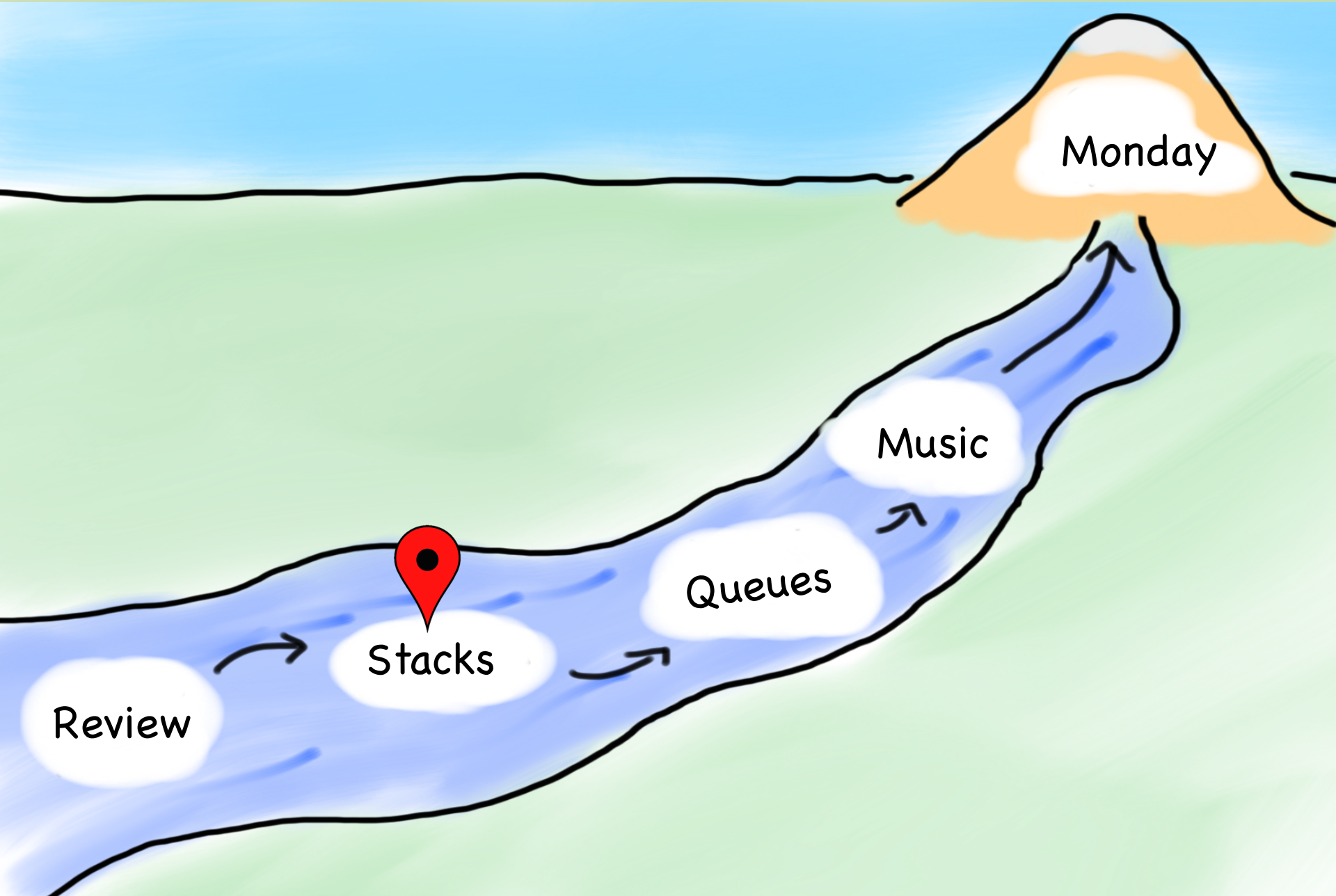
Velociraptor Safety



Today's Goals



Today's Goals

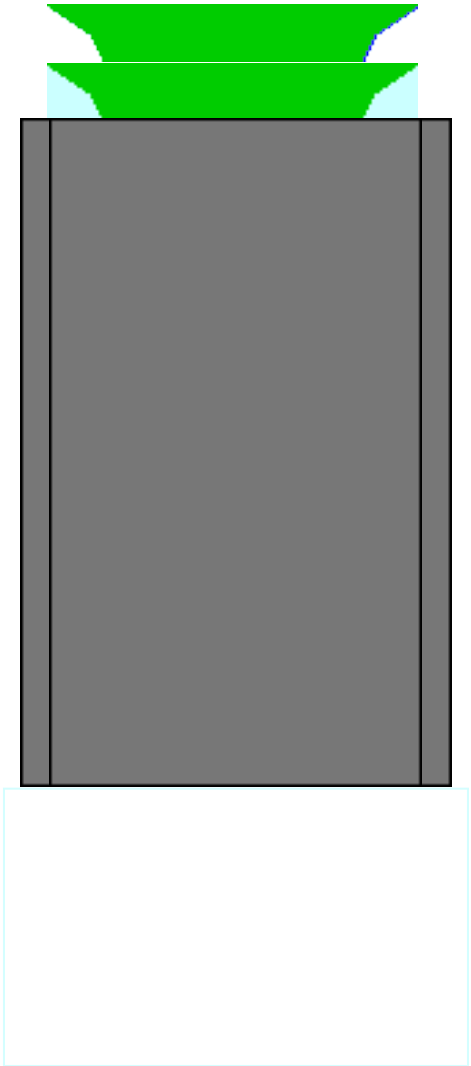


The Stack Metaphor

Last in, first out. LIFO

`push` and `pop`

Dish metaphor



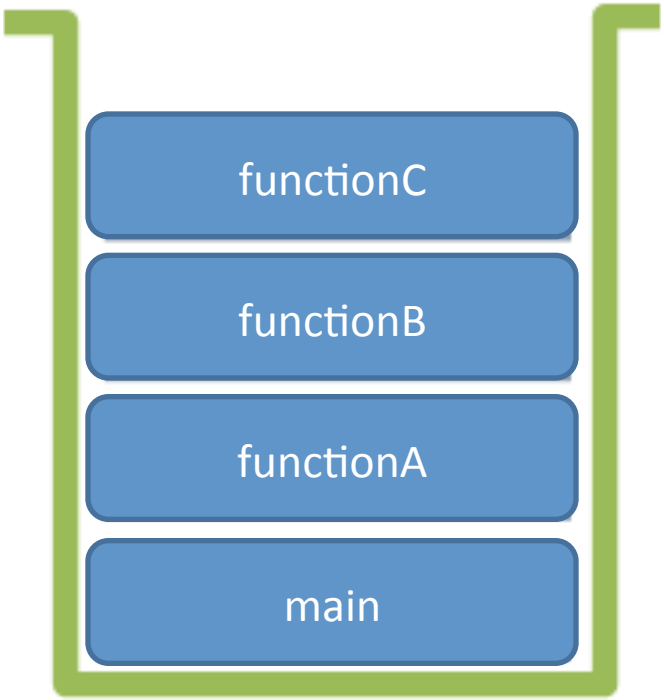
The Stack<*type*>

`stack.size()`	Returns the number of values pushed onto the stack.
`stack.isEmpty()`	Returns `true` if the stack is empty.
`stack.push(value)`	Pushes a new value onto the stack.
`stack.pop()`	Removes and returns the top value from the stack.
`stack.peek()`	Returns the top value from the stack without removing it.
`stack.clear()`	Removes all values from the stack.

The Stack<*type*>

`stack.`**`isEmpty`**`()`	Returns `true` if the stack is empty.
`stack.`**`push`**`(value)`	Pushes a new value onto the stack.
`stack.`**`pop`**`()`	Removes and returns the top value from the stack.

The Call Stack



Function (aka method) call stack:
Last in, First out Queue

Balance Parenthesis

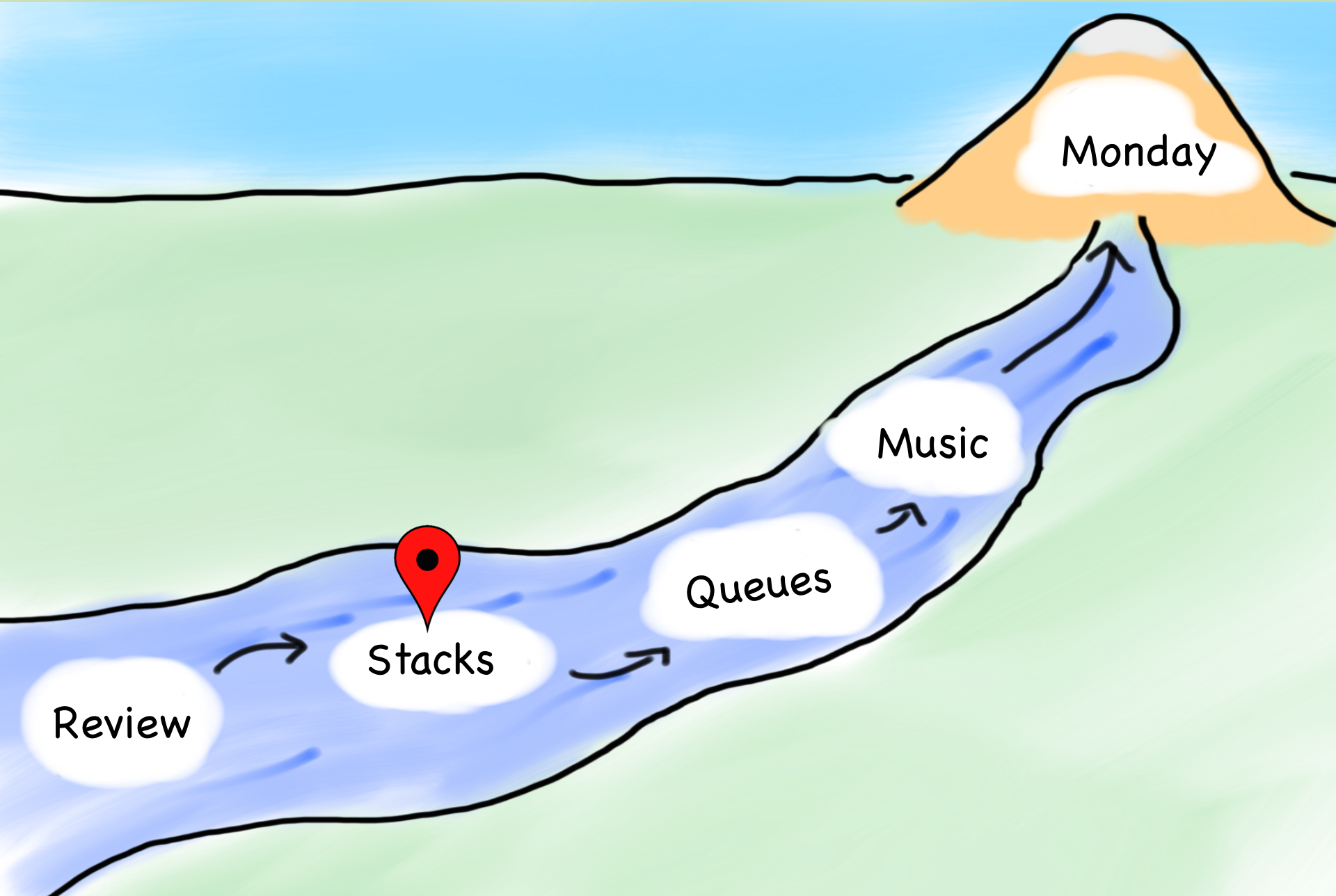
Check if parenthesis are balanced

```
{ s = 2 * (3); x = (1 + (2)); }
```

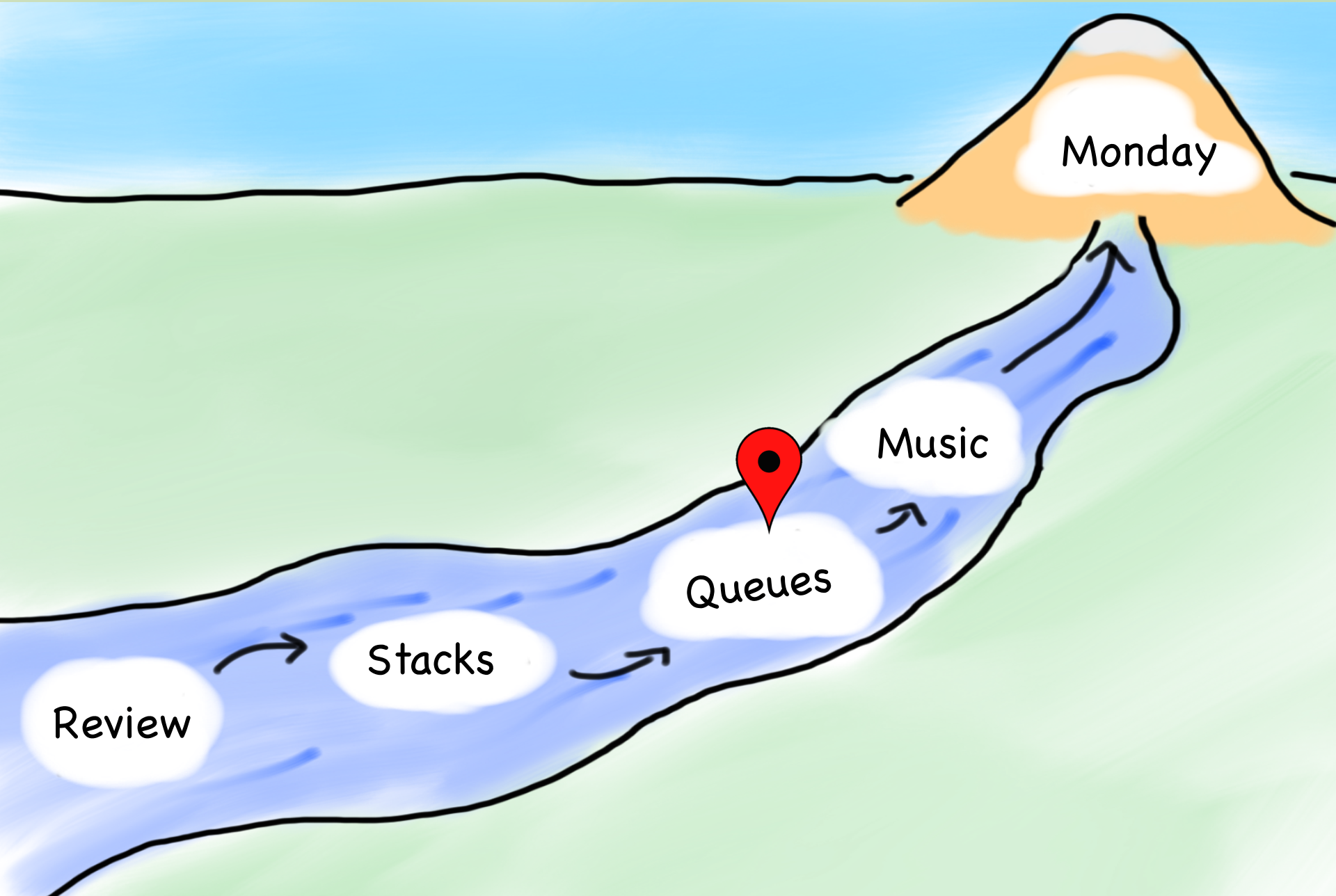
OperatorMatching

```
Enter string: { s = 2 * (3); x = (1 + (2)); }
Brackets are properly nested
Enter string: (a(2) + b(3}
Brackets are incorrect
Enter string:
```

Today's Goals



Today's Goals



The Queue Metaphor

First in, first out. FIFO

`enqueue` and `dequeue`

Line metaphor

The LaIR Queue

Helper Queue

Help requests

Up next:

Schedule

Swaps & Covers

Queue status

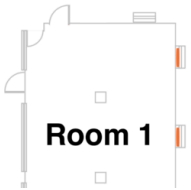
Signups enabled
0 unclaimed requests
Wait time: None!

Active helpers

Heather Kramer Busy
J Evans Busy
Shirin Salehi Busy
- Charissa Plattner
- Isabelle Ziebold
- Conner Smith
- Michael Dworsky

Janet An Busy
- Lucio Dery
- Jeffrey Zoch

SUNet ID Sign in



The LaIR Queue

The LaIR Queue

Helper Queue

Help requests

Up next:

Hermoine Granger

Student Left | Mark Resolved | Reassign to...

gfd

Location: Old Union | Heather Kramer is helping

9:06 PM

Schedule | Swaps & Covers

Queue status

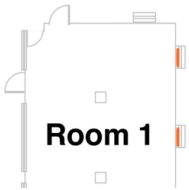
Signups enabled
0 unclaimed requests
Wait time: None!

Active helpers

Heather Kramer Busy
J Evans Busy
Shirin Salehi Busy
- Charissa Plattner
- Isabelle Ziebold
- Conner Smith
- Michael Dworsky

Janet An Busy
- Lucio Dery
- Jeffrey Zoch

SUNet ID | Sign in



The LaIR Queue

The LaIR Queue

Helper Queue

Help requests

Up next:

Hermoine Granger

Student Left | Mark Resolved | Reassign to...

gfd

Location: Old Union | Heather Kramer is helping

9:06 PM

Ron Weasley

Student Left | Mark Resolved | Reassign to...

gfd

Location: Old Union | Heather Kramer is helping

9:06 PM

Schedule | Swaps & Covers

Queue status

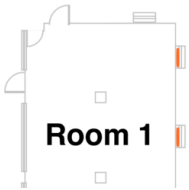
Signups enabled
0 unclaimed requests
Wait time: None!

Active helpers

Heather Kramer Busy
J Evans Busy
Shirin Salehi Busy
- Charissa Plattner
- Isabelle Ziebold
- Conner Smith
- Michael Dworsky

Janet An Busy
- Lucio Dery
- Jeffrey Zoch

SUNet ID | Sign in



The LaIR Queue

The LaIR Queue

Helper Queue

Help requests

Up next:

Ron Weasley

Student Left | Mark Resolved | Reassign to...

gfd

Location: Old Union | Heather Kramer is helping

9:06 PM

Schedule | Swaps & Covers

Queue status

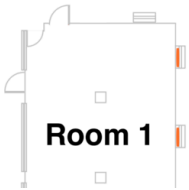
Signups enabled
0 unclaimed requests
Wait time: None!

Active helpers

Heather Kramer Busy
J Evans Busy
Shirin Salehi Busy
- Charissa Plattner
- Isabelle Ziebold
- Conner Smith
- Michael Dworsky

Janet An Busy
- Lucio Dery
- Jeffrey Zoch

SUNet ID | Sign in



Spotify File Edit View Playback Window Help 37% Sun 1

Search

Radio

YOUR MUSIC

Songs

Albums

Artists

Local Files

PLAYLISTS

Winter CS106B 11

Winter CS106B Relax...

Winter CS106B Upbeat

discover by Egemen Eren

Starred

Discover Weekly by S...

Bogazici

Stanford-Bogazici

Mixtape2

Hashing

New Playlist

Finally Moving
Pretty Lights

Play Queue

QUEUE HISTORY

CURRENT TRACK

#	SONG	ARTIST	ALBUM	
	Finally Moving	Pretty Lights	Taking up Your Preciou...	4:38

QUEUED TRACKS

#	SONG	ARTIST	ALBUM	
1	Memories	Petit Biscuit	Memories	3:36
2	Berlin	The Piano Guys	The Piano Guys 2	4:01
3	Swim Until You Can't See Land	Frightened Rabbit	The Winter of Mixed Dri...	4:19
4	Riptide	Vance Joy	Riptide	3:24
5	El Kilo	Orishas	Antidiotico	4:25
6	Old Pine	Ben Howard	Every Kingdom	5:29
7	Islands	The xx	xx	2:41

0:24 4:37 LYRICS

Queue<*type*>

`queue.size()` Returns the number of values in the queue.
`queue.isEmpty()` Returns `true` if the queue is empty.
`queue.enqueue(value)` Adds a new value to the end of the queue (which is often called its *tail*).
`queue.dequeue()` Removes and returns the value at the front of the queue (its *head*).
`queue.peek()` Returns the value at the head of the queue without removing it.
`queue.clear()` Removes all values from the queue.

Queue<*type*>

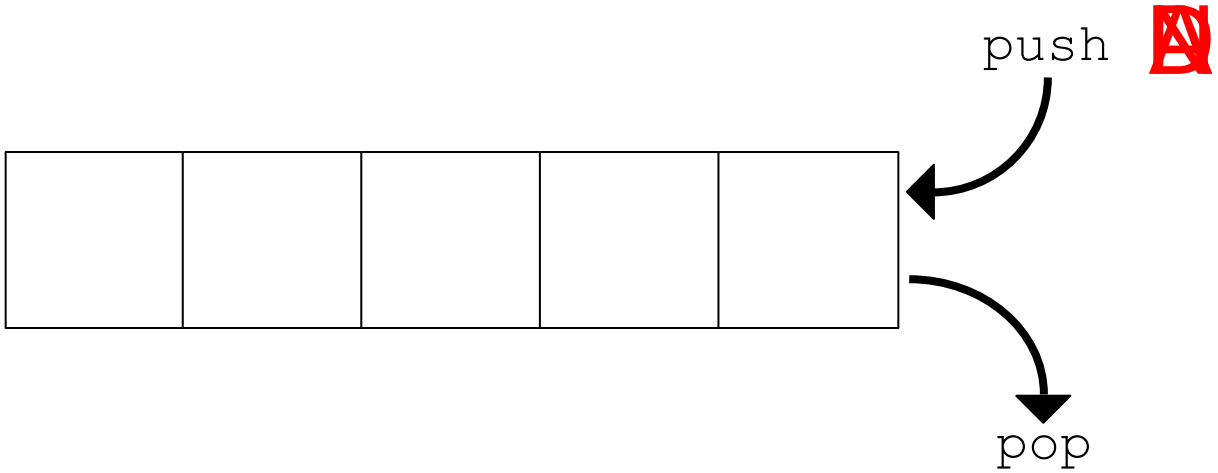
`queue.`**`isEmpty`**`()` Returns `true` if the queue is empty.
`queue.`**`enqueue`**`(value)` Adds a new value to the end of the queue (which is often called its *tail*).
`queue.`**`dequeue`**`()` Removes and returns the value at the front of the queue (its *head*).

Stacks vs Queues

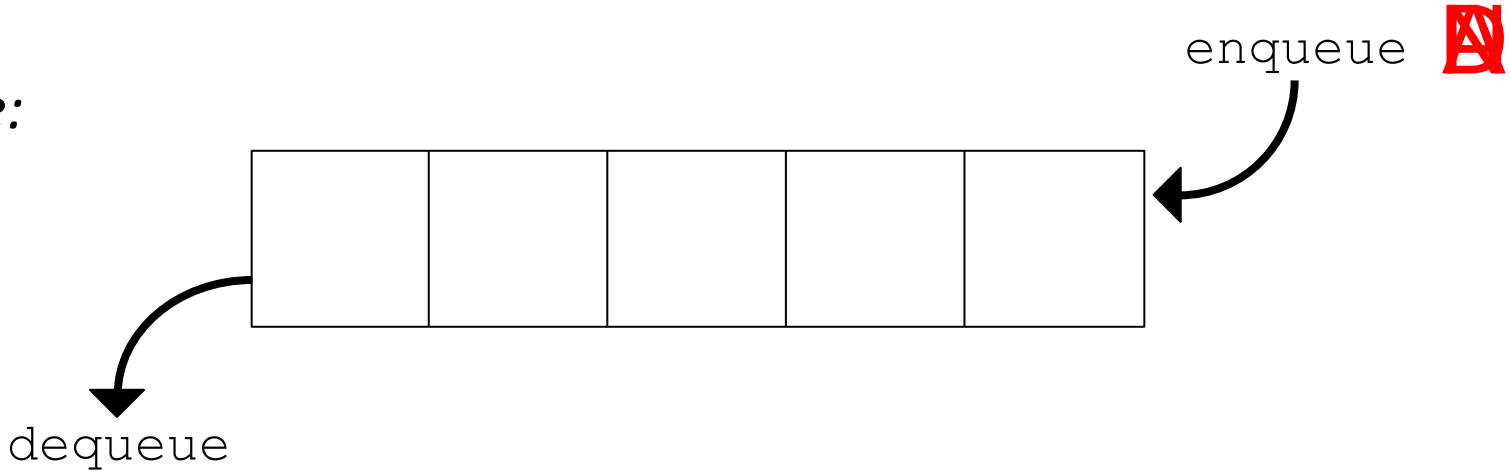


Stacks vs Queues

Stack:

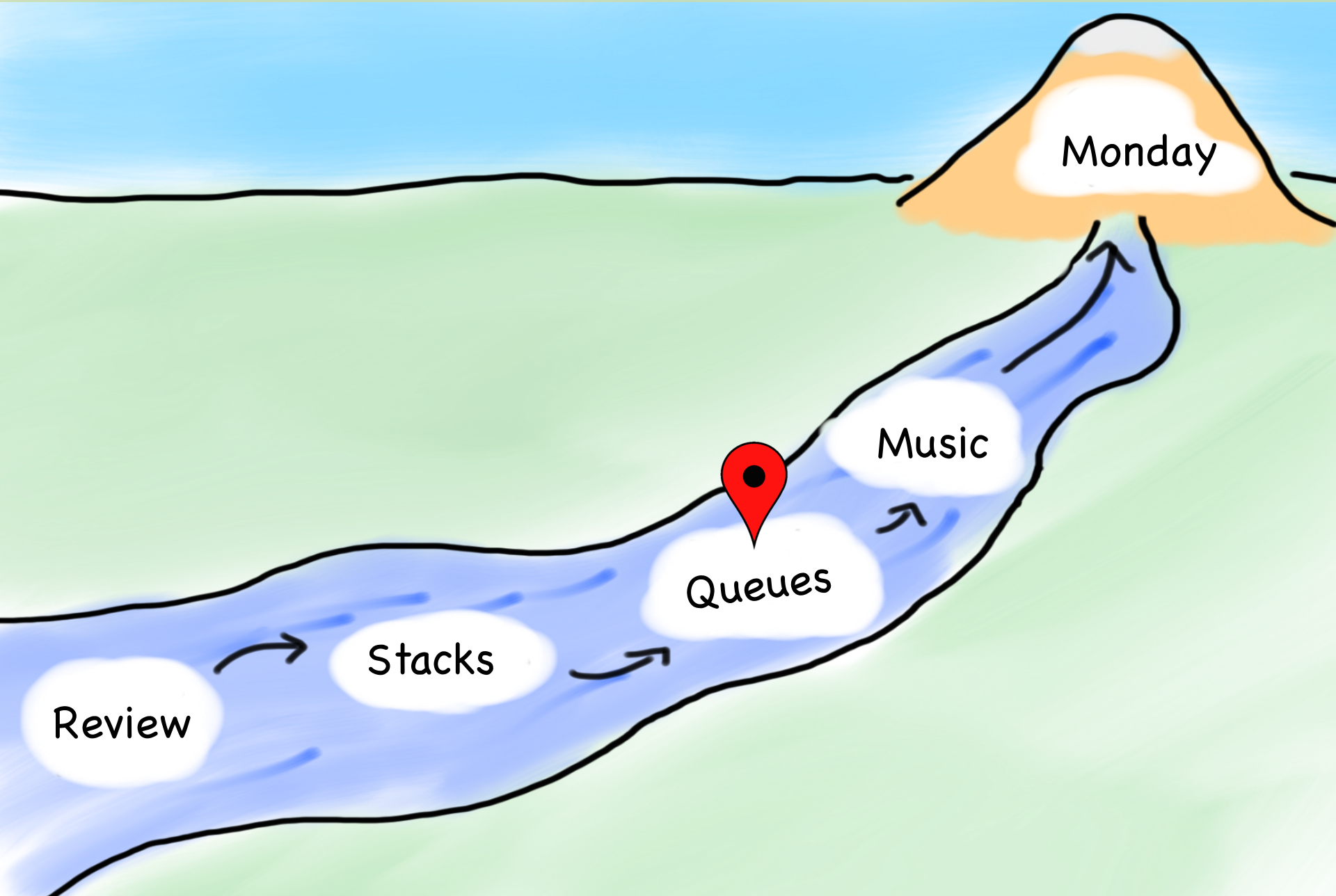


Queue:

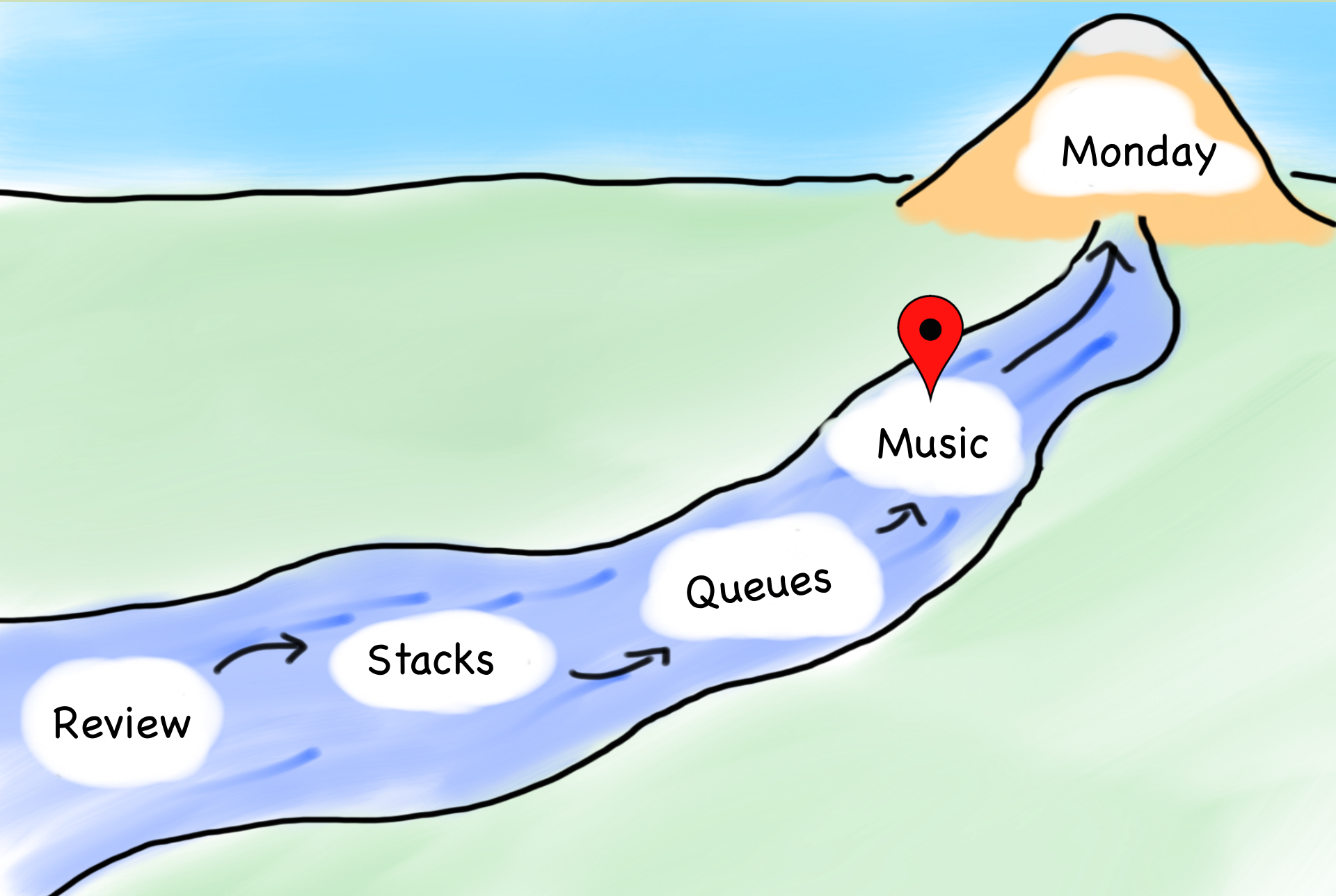


Why not Vector?

Today's Goals



Today's Goals



Digital Music: Queue of Intensities

`song:`

0.01	0.16	0.31	0.45	0.58	0.69

...

Front of the queue

Playing Digital Music

1. Dequeue

2. Play

`song:`

0.01	0.16	0.31	0.45	0.58	0.69

...

Front of the queue

Playing Digital Music

1. Dequeue

2. Play

song:

0.16	0.31	0.45	0.58	0.69	•••

Front of the queue

Playing Digital Music

1. Dequeue

2. Play

song:

0.16	0.31	0.45	0.58	0.69	•••

Front of the queue

Playing Digital Music

1. Dequeue

2. Play

`song:`

0.16	0.31	0.45	0.58	0.69	0.78	•••

Front of the queue

Digital Music

`song:`

0.01	0.16	0.31	0.45	0.58	0.69

The Sun
by
Parov Stelar

0.78	0.85	0.90	0.93	0.95	0.95	0.93

0.02 milliseconds of music

0.91	0.88	0.84	0.81	0.77	0.74	0.71

...

Guitar String

```
void pluck(Queue<double>& sound, double freq) {



}

double sample(Queue<double>& sound) {



}
```

How to pluck a guitar string

Karplus Algorithm: Pluck

- Each string has a frequency.
- Let N be the number of speaker samples in one wave at that frequency.
- **Enqueue N random numbers.**

How to sample a guitar string

Karplus Algorithm: Sample

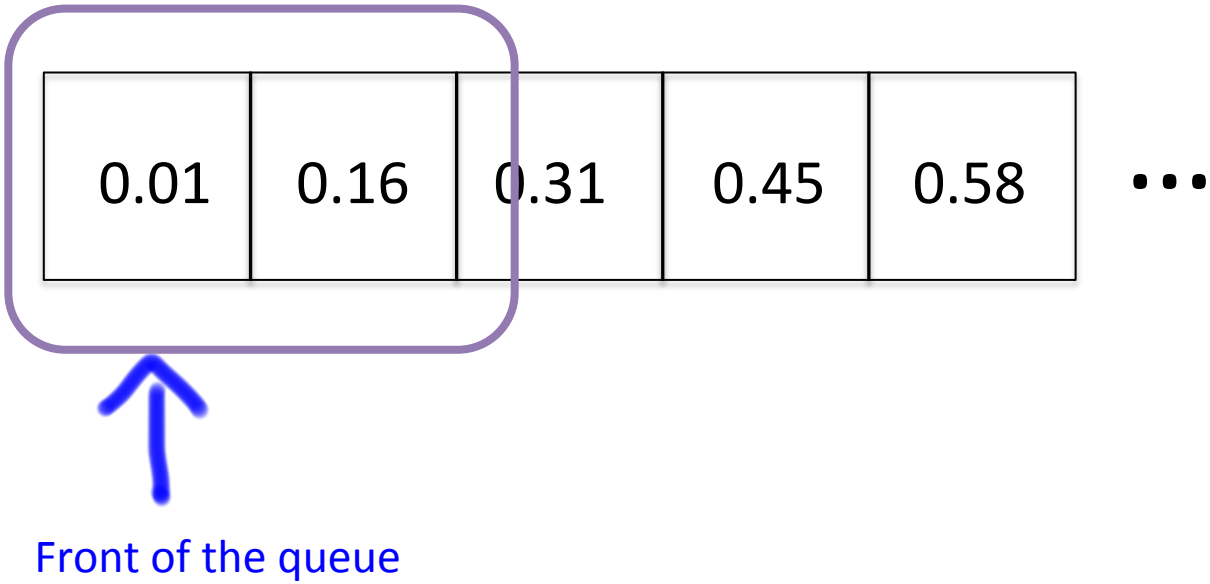
0.01	0.16	0.31	0.45	0.58

…

Front of the queue

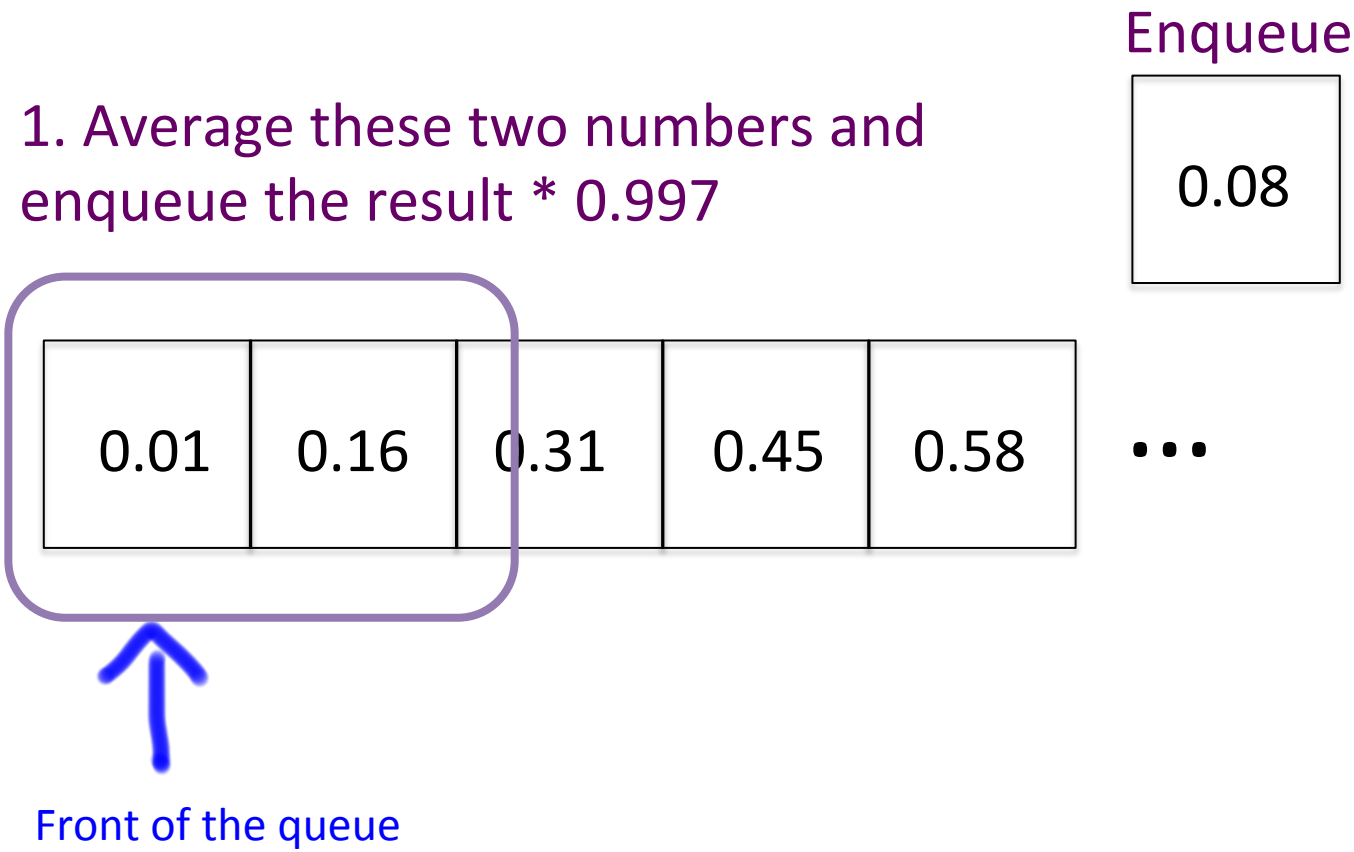
Karplus Algorithm: Sample

1. Average these two numbers and enqueue the result * 0.997



Karplus Algorithm: Sample

1. Average these two numbers and enqueue the result * 0.997



Karplus Algorithm: Sample

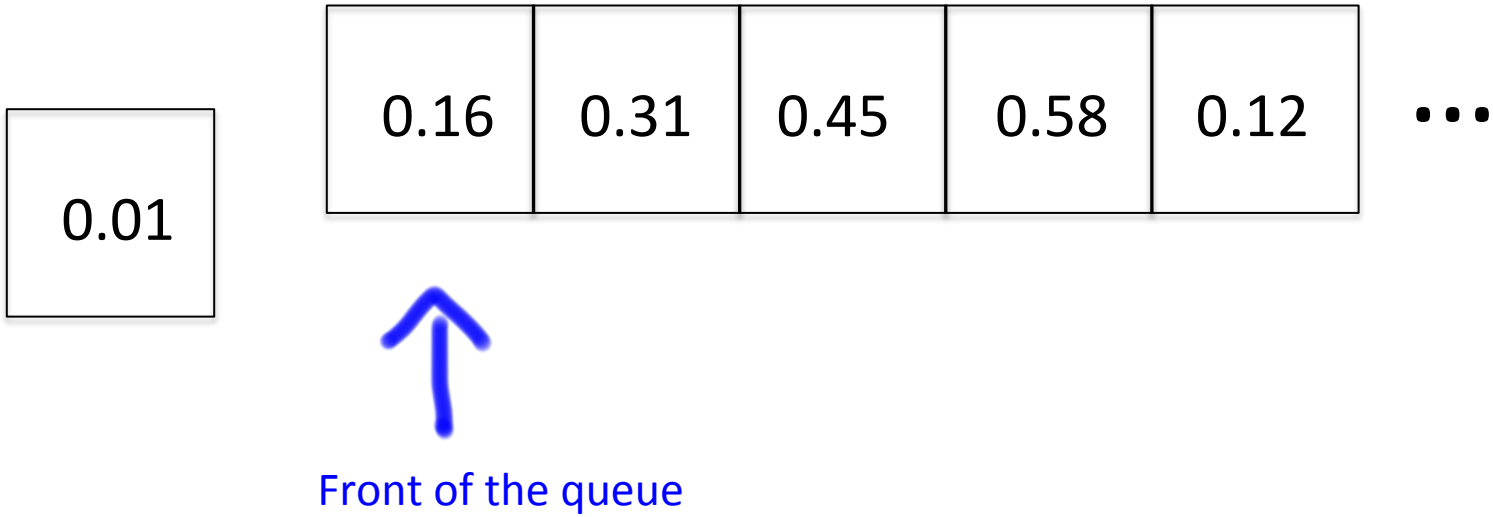
2. Dequeue the top for the speaker to play

0.01	0.16	0.31	0.45	0.58	•••

Front of the queue

Karplus Algorithm: Sample

2. Dequeue the top for the speaker to play

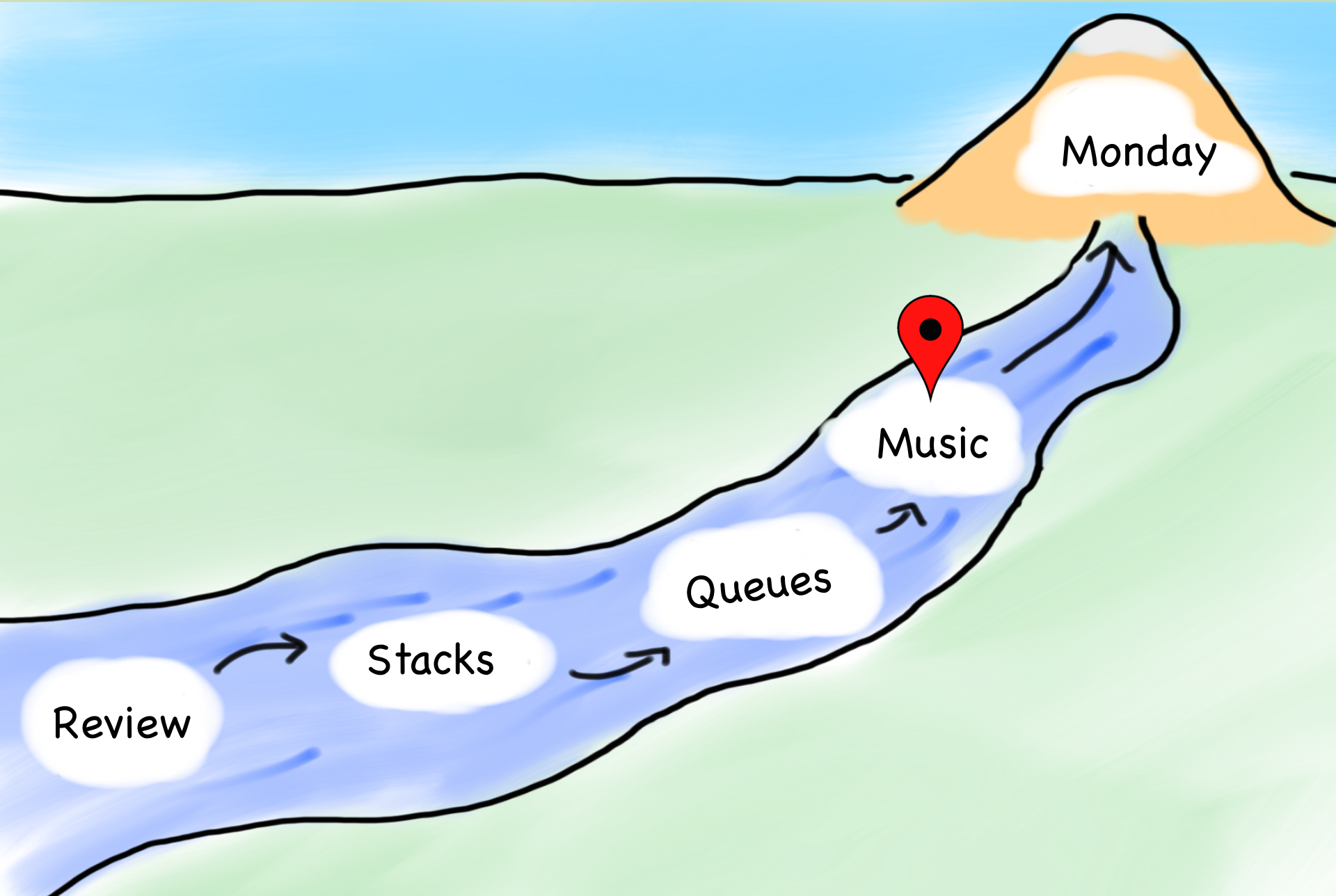


Guitar String

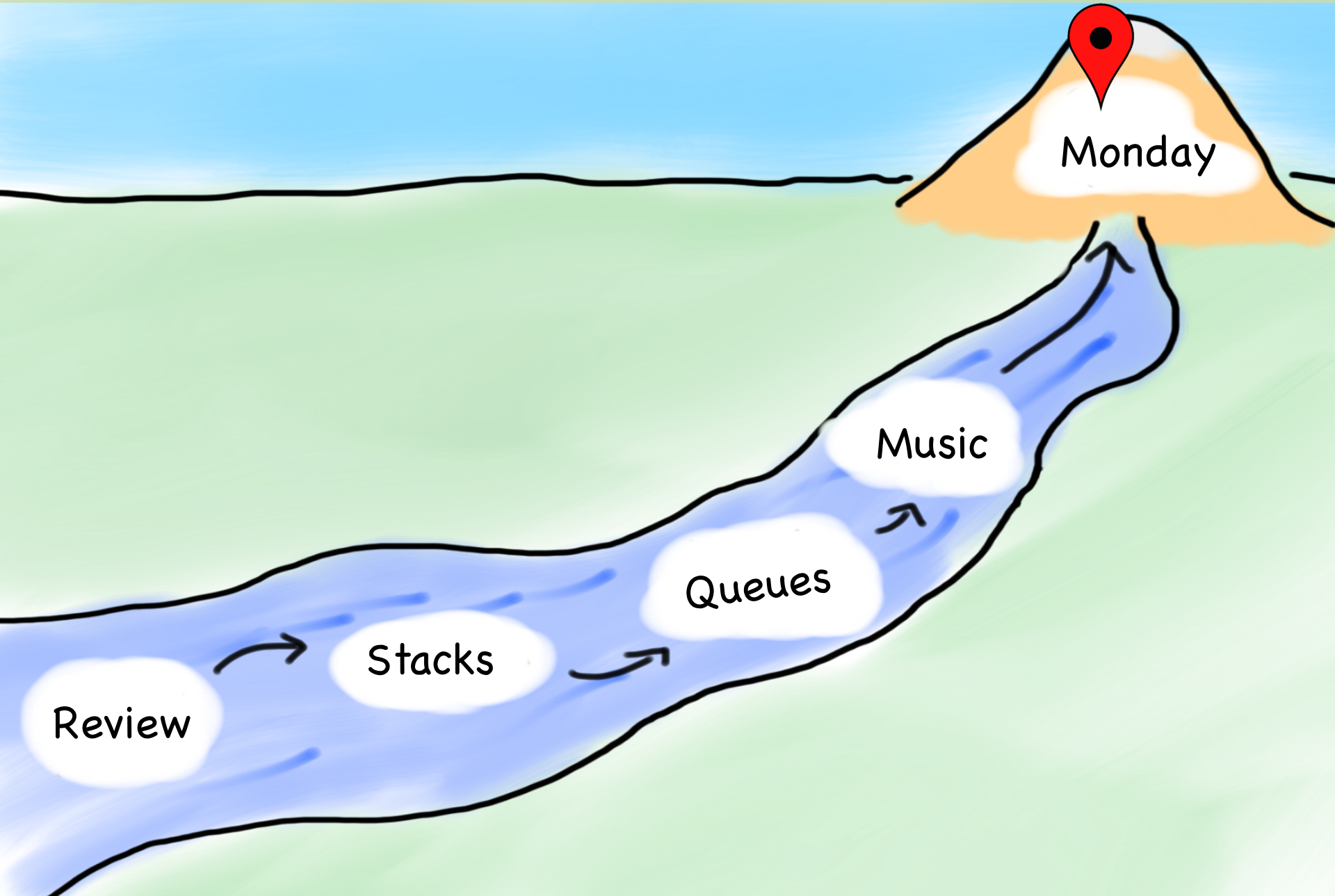
```
void pluck(Queue<double>& sound, double freq) {
   sound.clear();
   int n = RATE / freq;  //how many values we need
   for(int i = 0; i < n; i++) {
      sound.queue(randomReal(-1.0, 1.0));
   }
}


double sample(Queue<double>& sound) {
   double a = sound.dequeue();
   double b = sound.peek();
   double next = 0.997 * (a + b) / 2;
   sound.enqueue(next);
   return a;
}
```


Guitar Strings are an open problem.
Accidents lead to new sounds.

Today's Goals

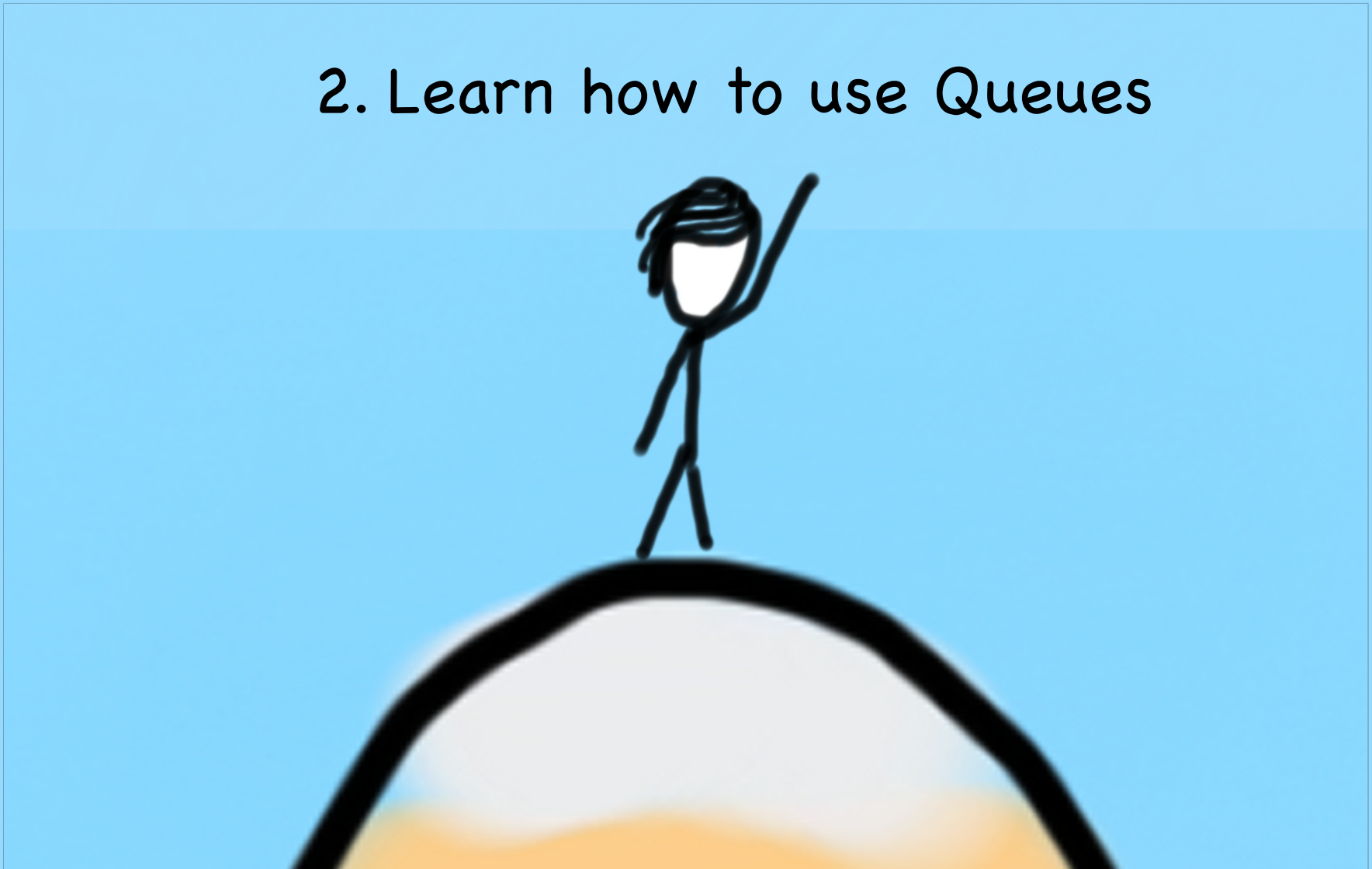


Today's Goals



Today's Goals

1. Learn how to use Stacks
2. Learn how to use Queues



The End