



# BIONODE.IO

Tutorial

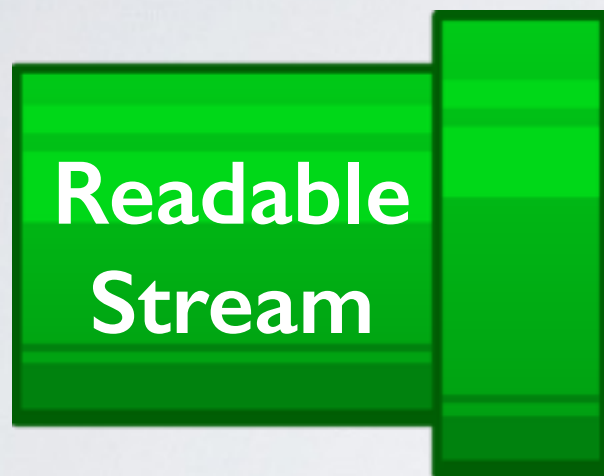
# STREAMS

Streams are a first-class construct in Node.js for handling data.



# PROCESS DATA IN CHUNKS

# PROCESS DATA IN CHUNKS

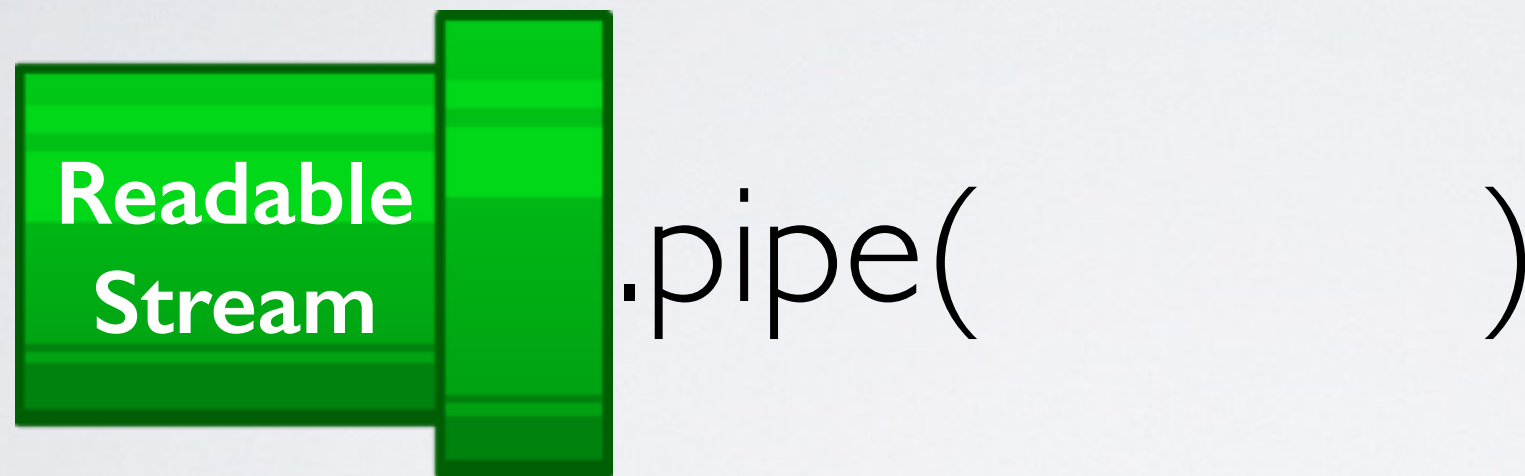


`fs.createReadStream(file)`

`request(url)`

`process.stdin()`

# PROCESS DATA IN CHUNKS



`fs.createReadStream(file)`

`request(url)`

`process.stdin()`



# PROCESS DATA IN CHUNKS



`fs.createReadStream(file)`

`request(url)`

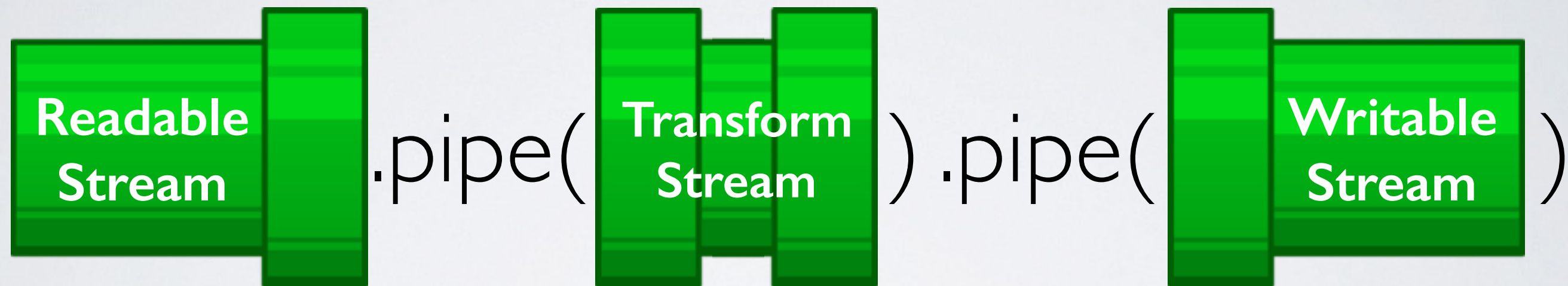
`process.stdin()`

`JSONStream.parse()`

`filterFunction()`

`multithreadAnalysis()`

# PROCESS DATA IN CHUNKS



`fs.createReadStream(file)`

`request(url)`

`process.stdin()`

`JSONStream.parse()`

`filterFunction()`

`multithreadAnalysis()`

`fs.createWriteStream(file)`

`process.stdout()`

Readable  
Stream

The diagram illustrates a stream processing pipeline with four stages, each represented by a vertical green bar with a dark green outline. The bars are connected by horizontal lines at the top and bottom, forming a continuous path. The first bar is labeled 'Readable Stream'. The second bar is labeled 'Transform Stream'. The third bar is labeled 'Readable Stream'. The fourth bar is labeled 'Writable Stream'. The bars are arranged in a slightly staggered fashion, with the second and third bars being slightly offset to the right relative to the first and fourth bars respectively.

Transform  
Stream

Writable  
Stream

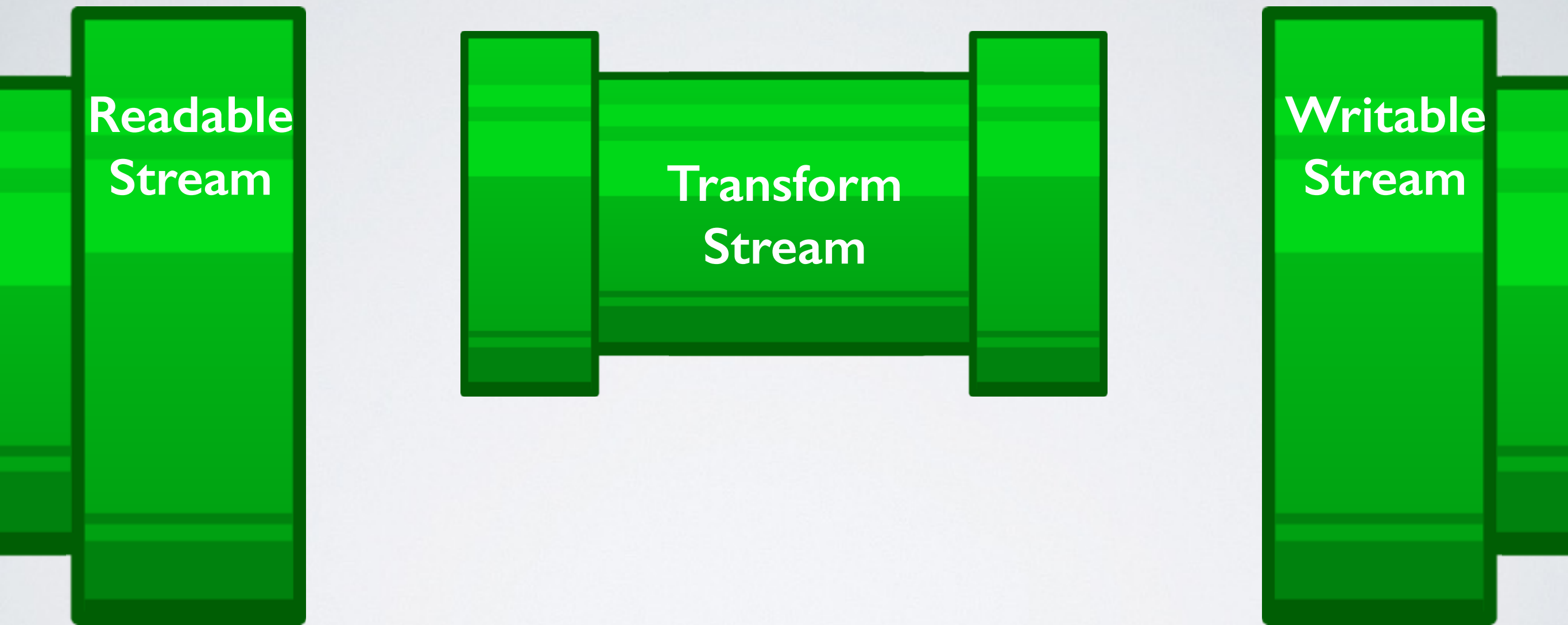


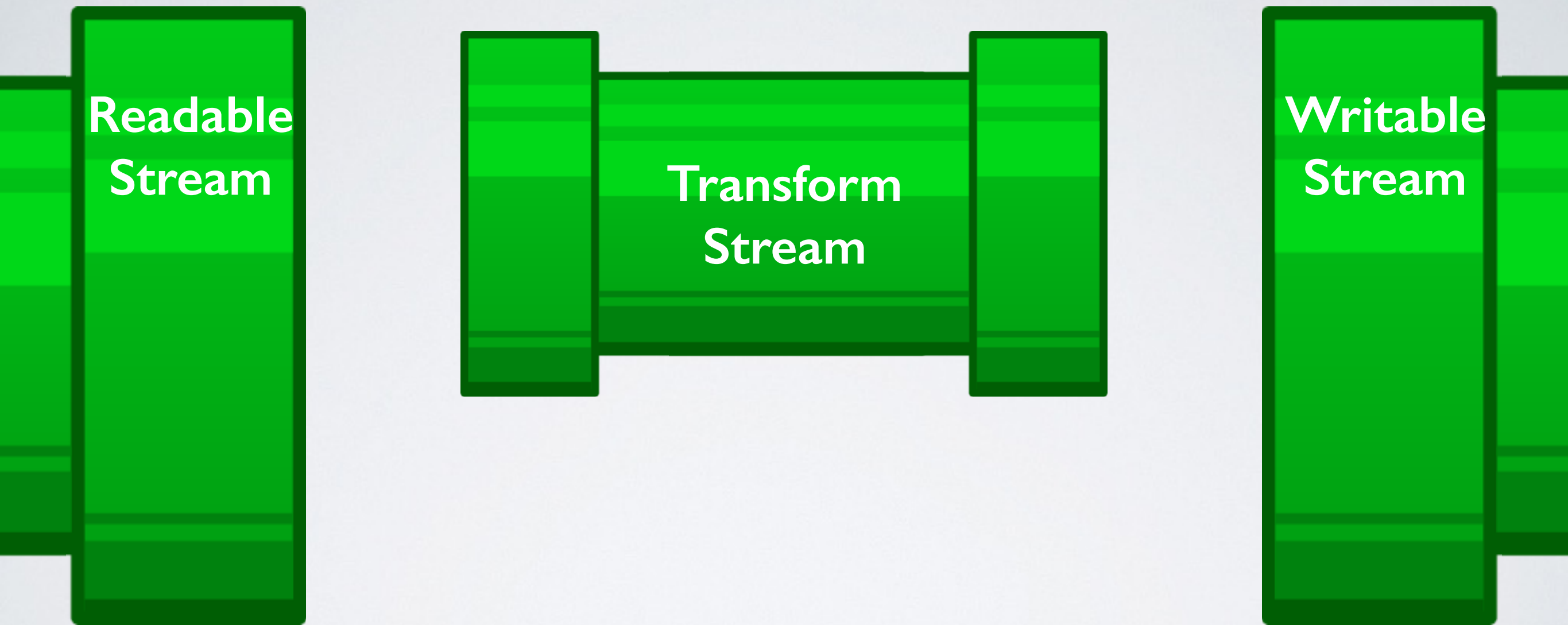
The diagram illustrates the components of a stream. It features three main vertical green bars with dark green outlines. The first bar on the left is labeled 'Readable Stream'. The second bar is labeled 'Transform Stream' and is connected to the first by a horizontal line. The third bar is labeled 'Writable Stream' and is connected to the second by another horizontal line. Each bar has a lighter green top section and a darker green bottom section. The text is white and centered within the top section of each bar.

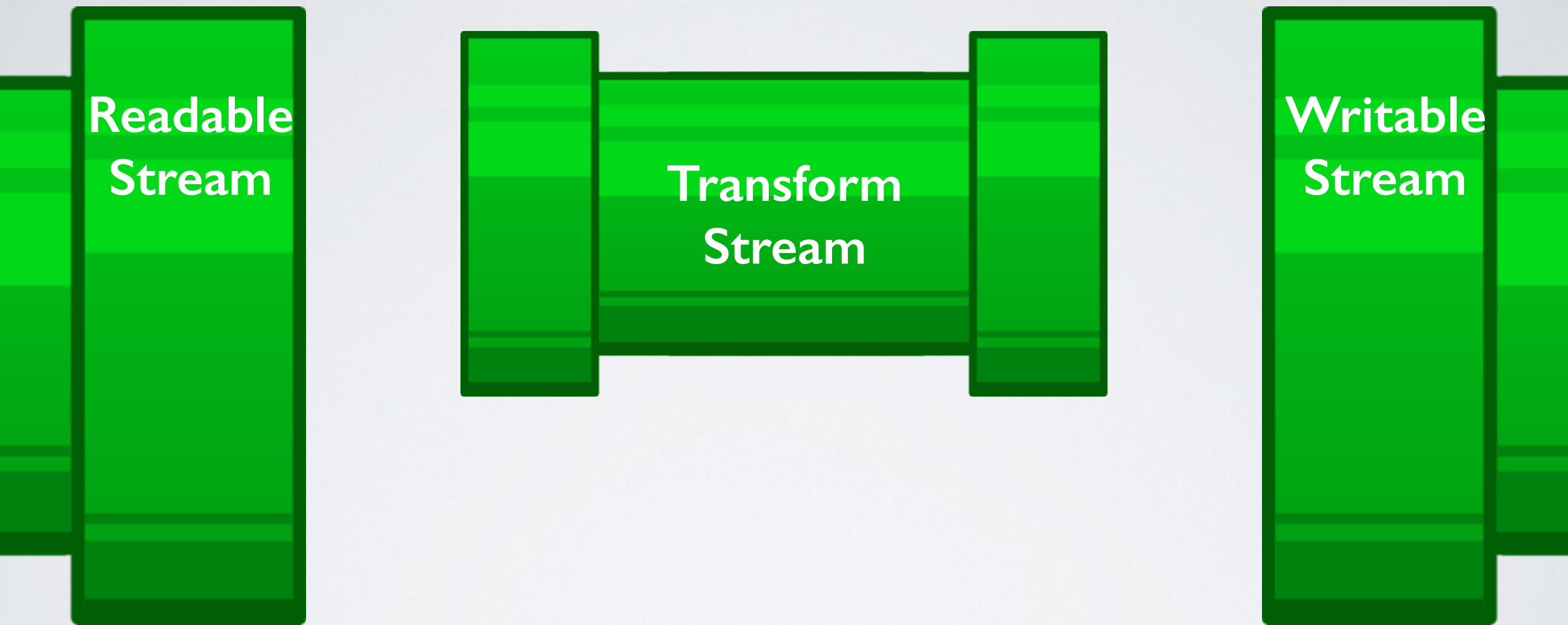
**Readable  
Stream**

**Transform  
Stream**

**Writable  
Stream**





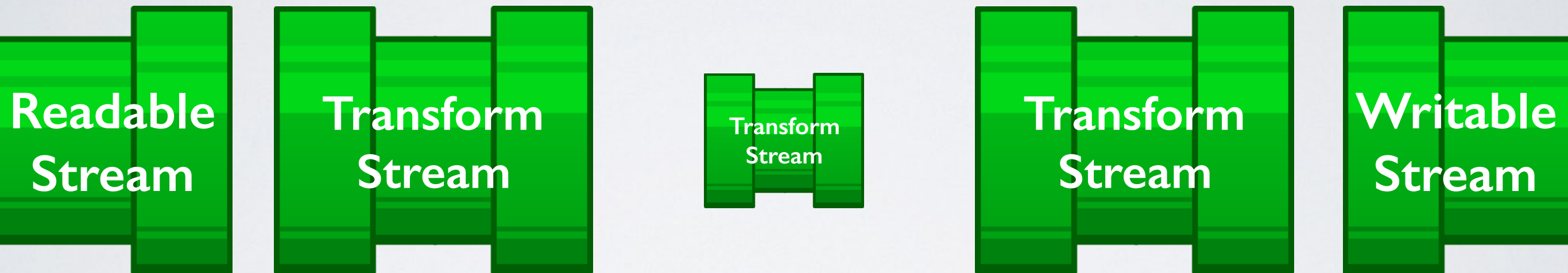


# PROCESS DATA IN CHUNKS





# PROCESS DATA IN CHUNKS



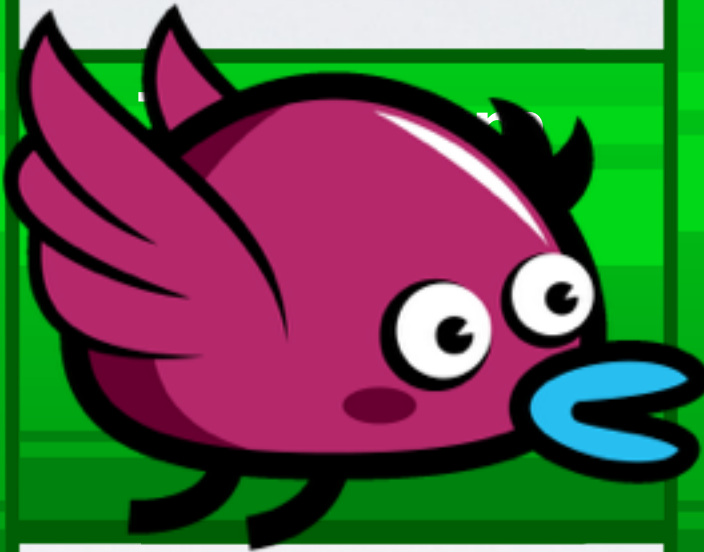
# PROCESS DATA IN CHUNKS



# PROCESS DATA IN CHUNKS



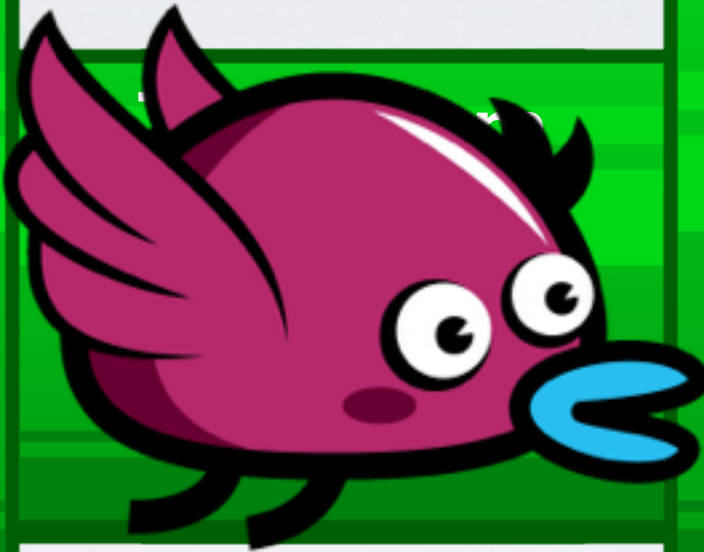
Readable  
Stream



Writable  
Stream



Readable  
Stream

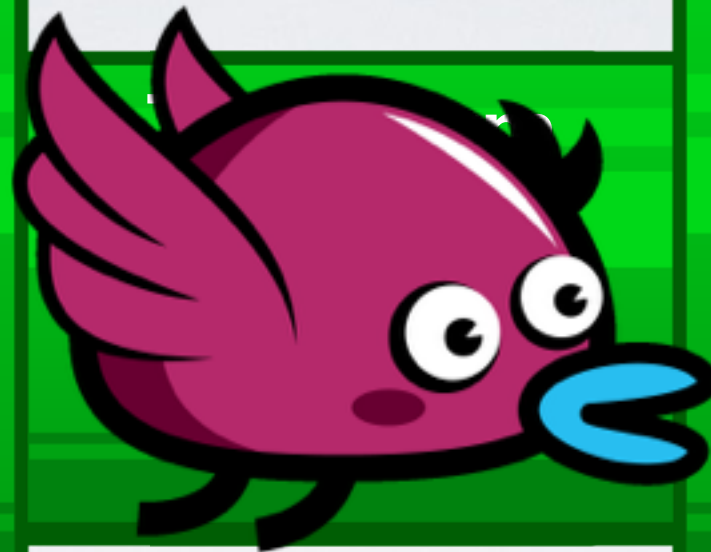


Writable  
Stream





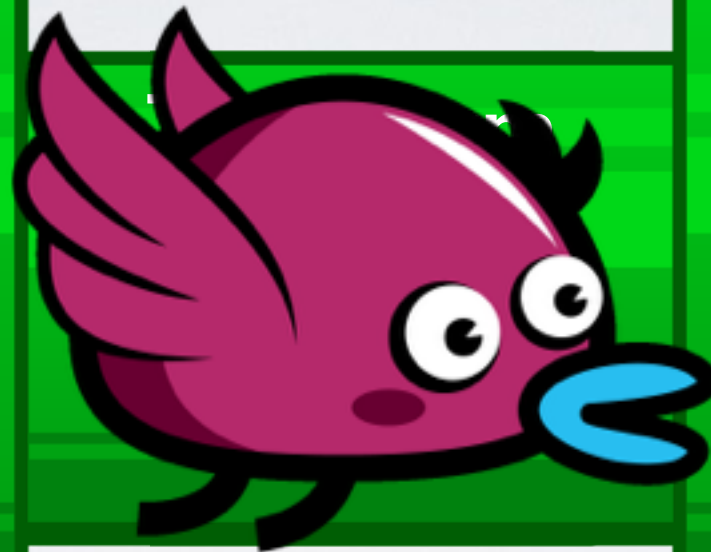
Readable  
Stream



Writable  
Stream

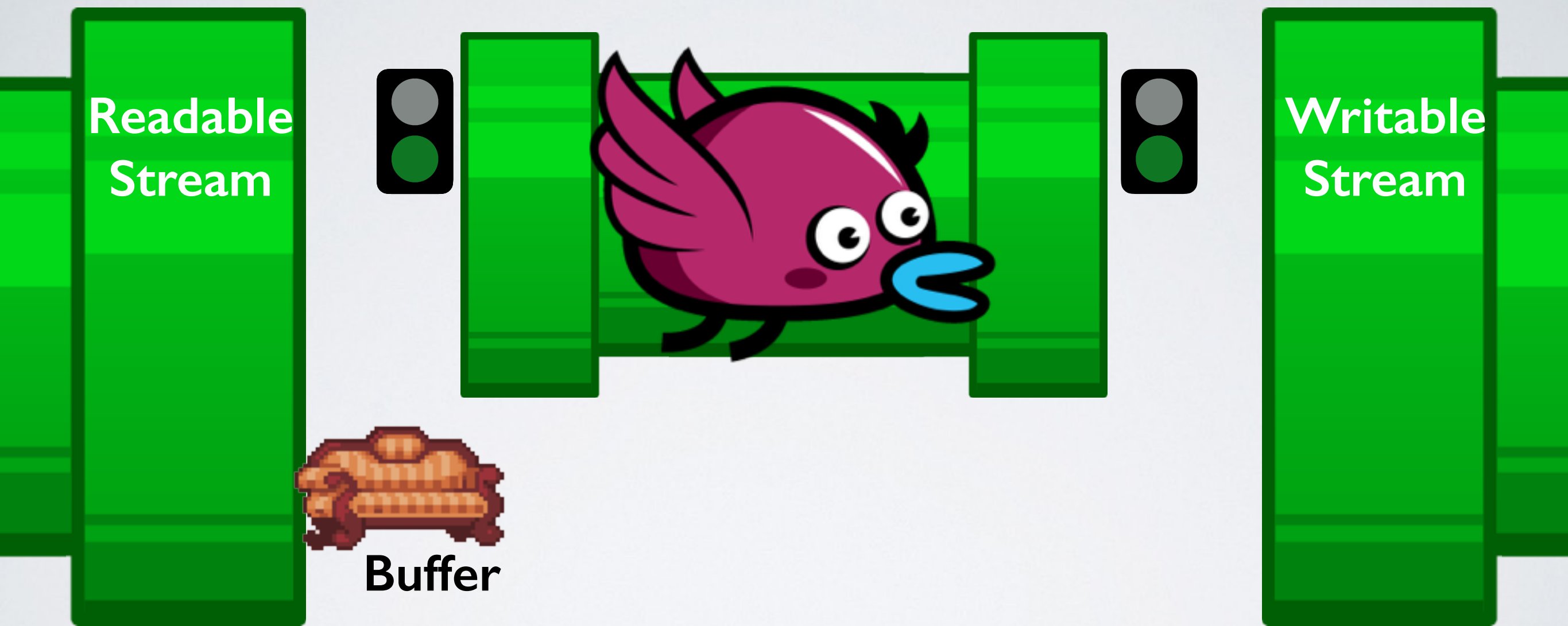


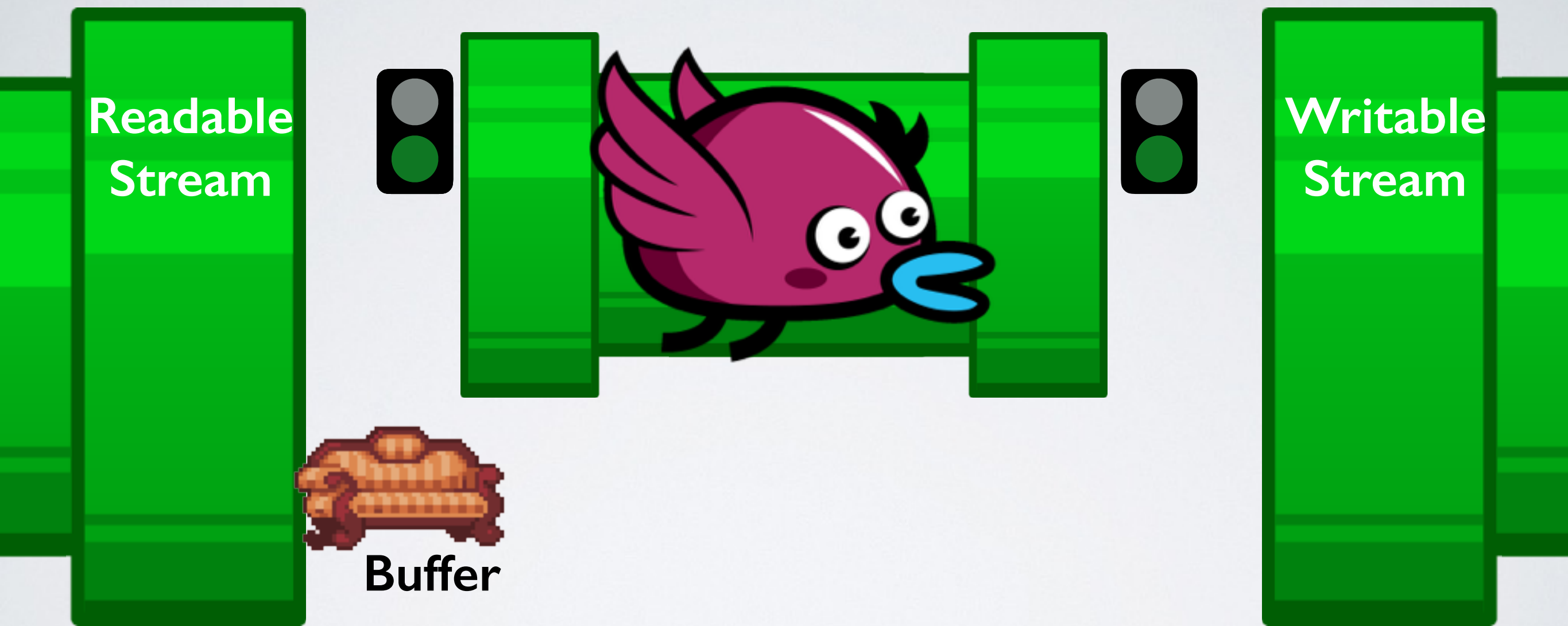
Readable  
Stream



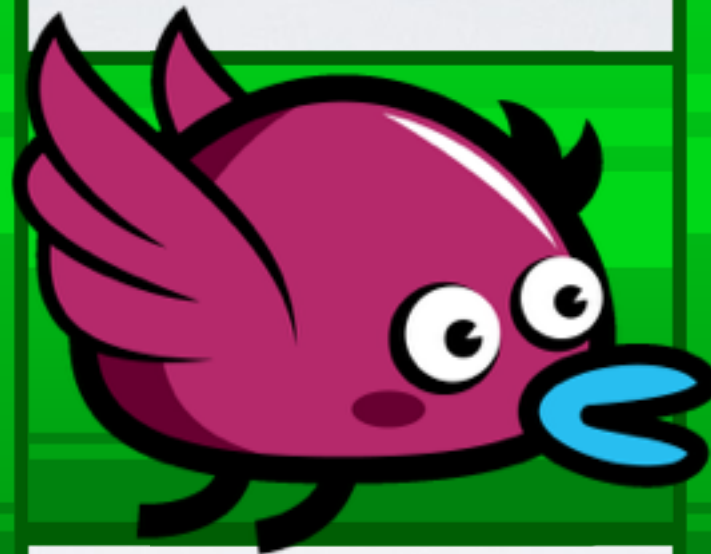
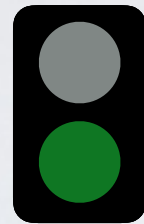
Writable  
Stream







Readable  
Stream



Writable  
Stream

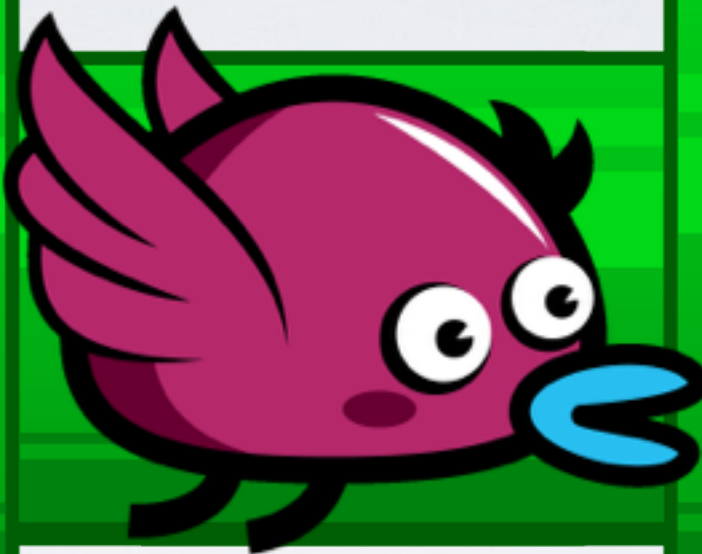
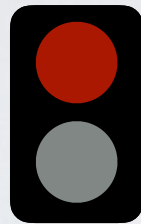


Buffer



**this.push(data)**  
**FALSE**

Readable  
Stream



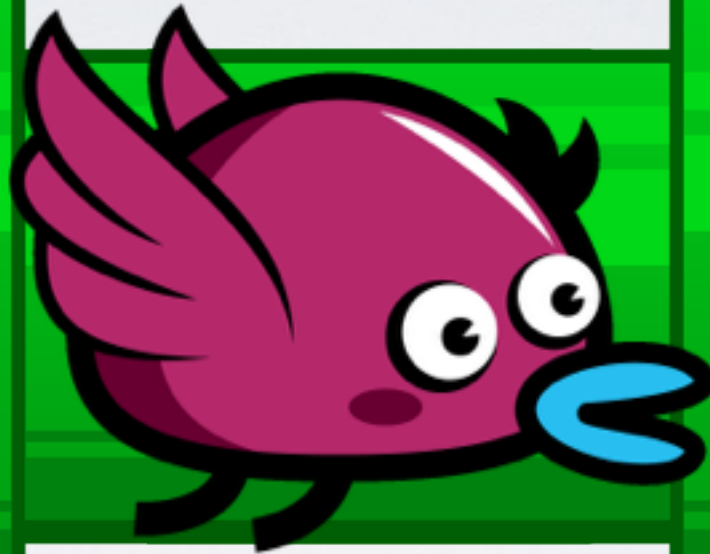
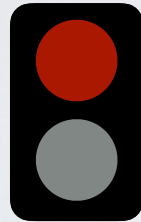
Writable  
Stream



Buffer

**this.push(data)**  
**FALSE**

Readable  
Stream



Writable  
Stream

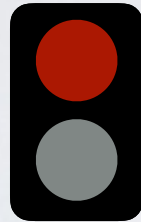


Buffer

**this.push(data)**  
**FALSE**

**this.push(data)**  
**TRUE**

Readable  
Stream



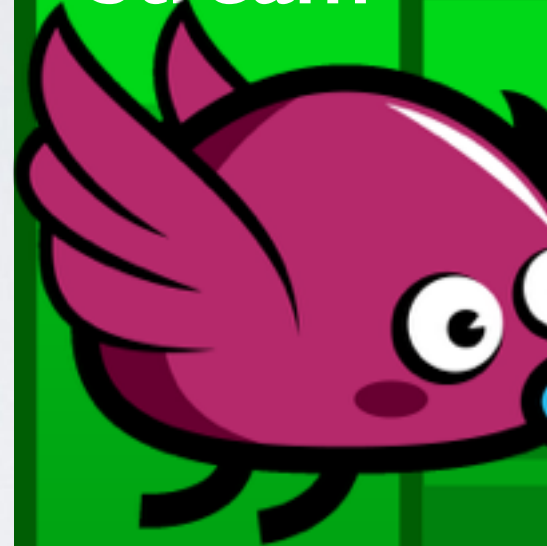
Transform  
Stream

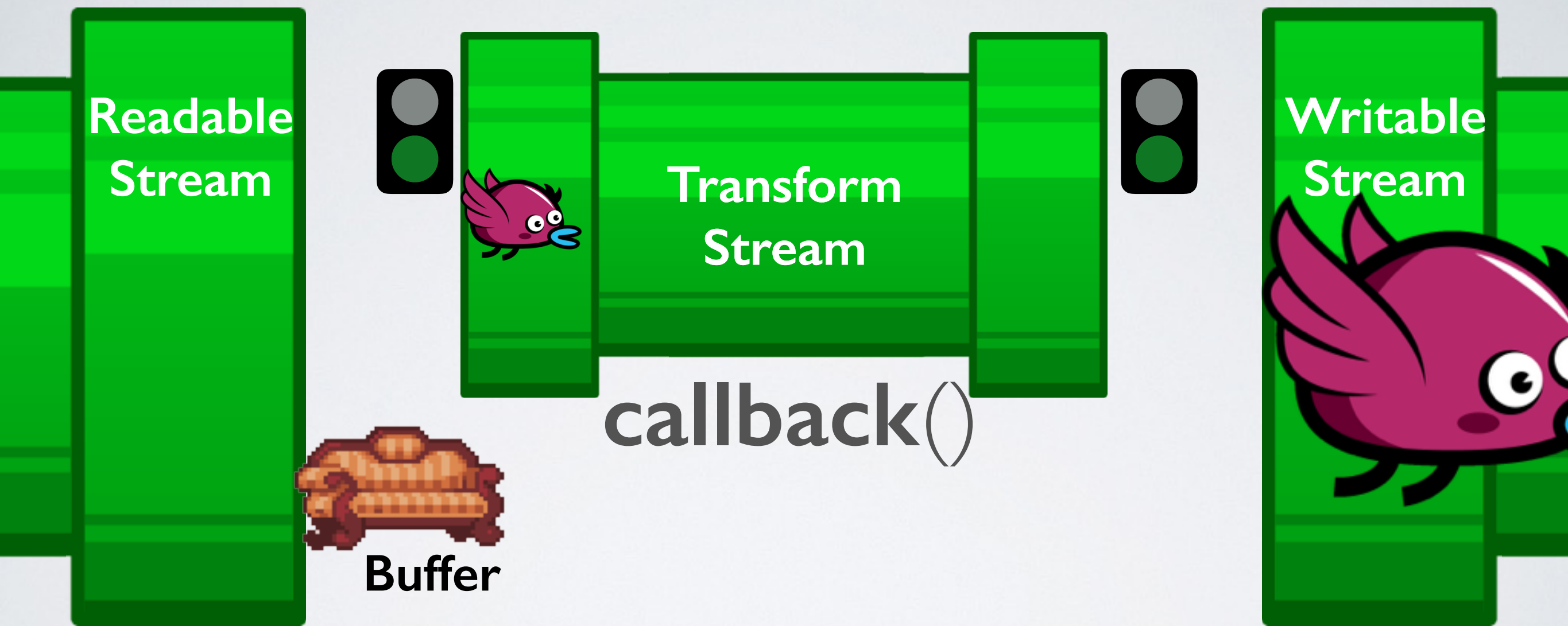


Writable  
Stream

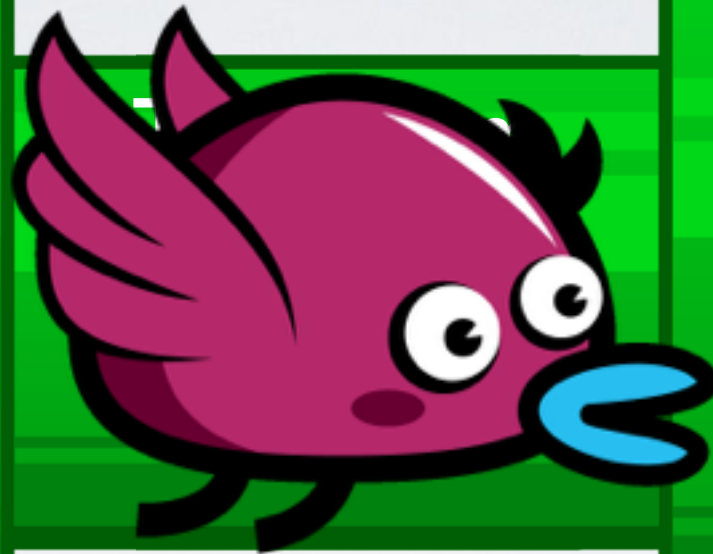
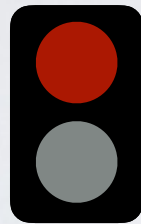


Buffer





Readable  
Stream



Writable  
Stream

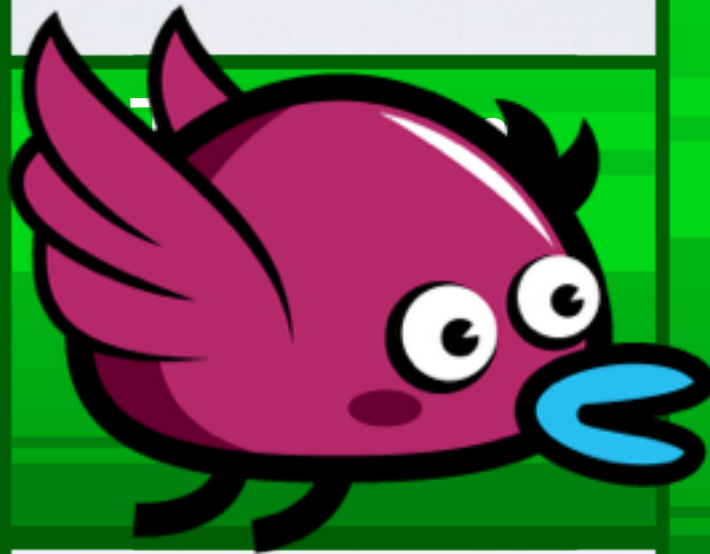
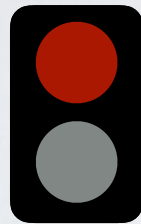


Buffer

**this.push(data)**  
**TRUE**



Readable  
Stream

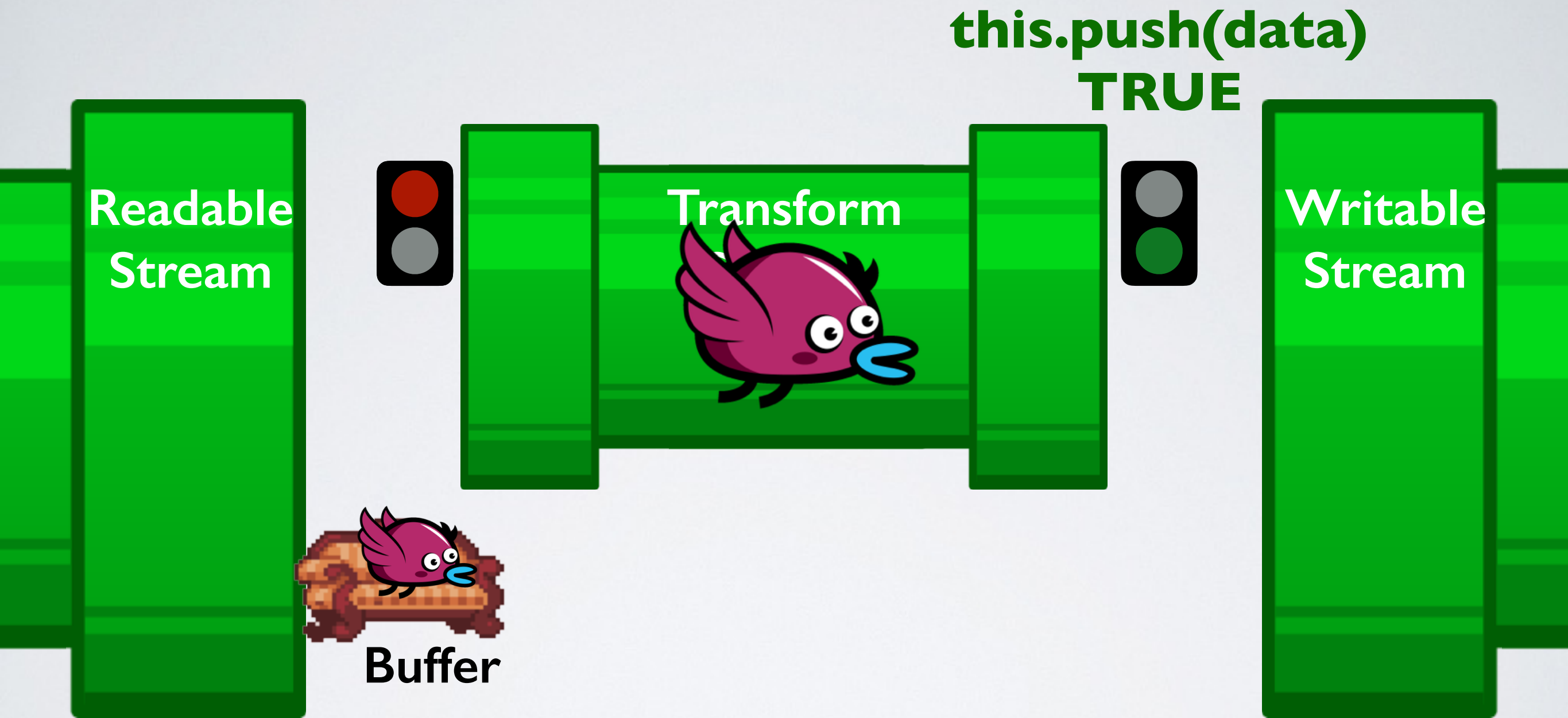


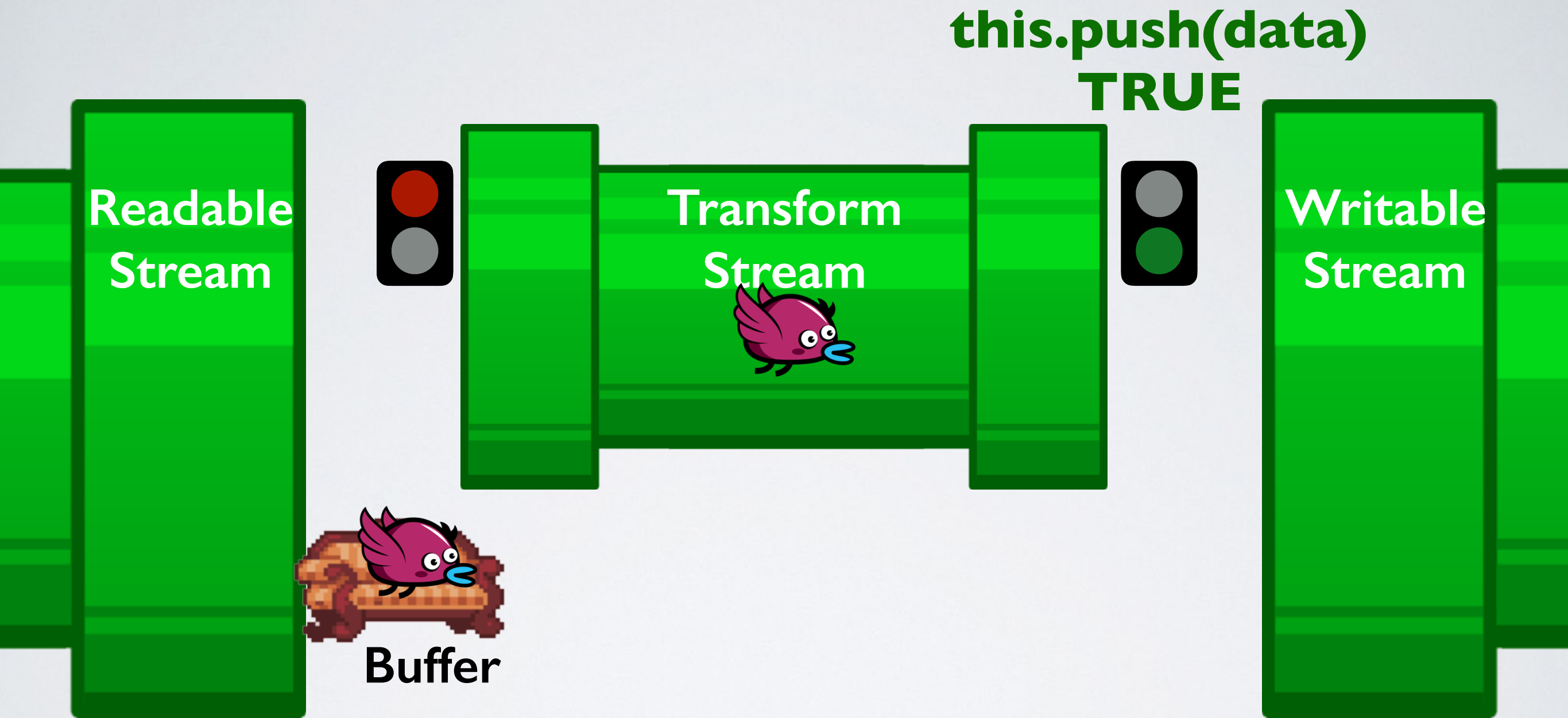
Writable  
Stream

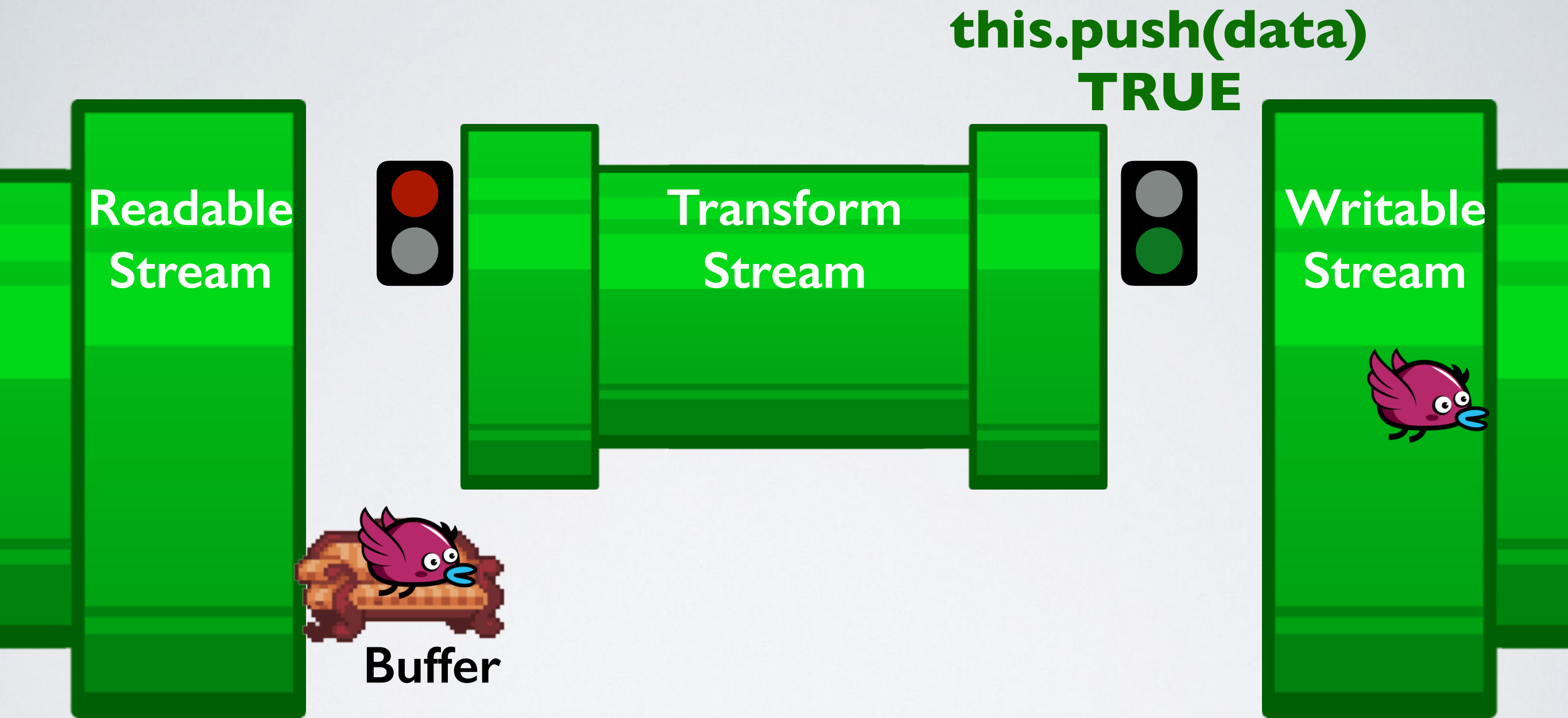


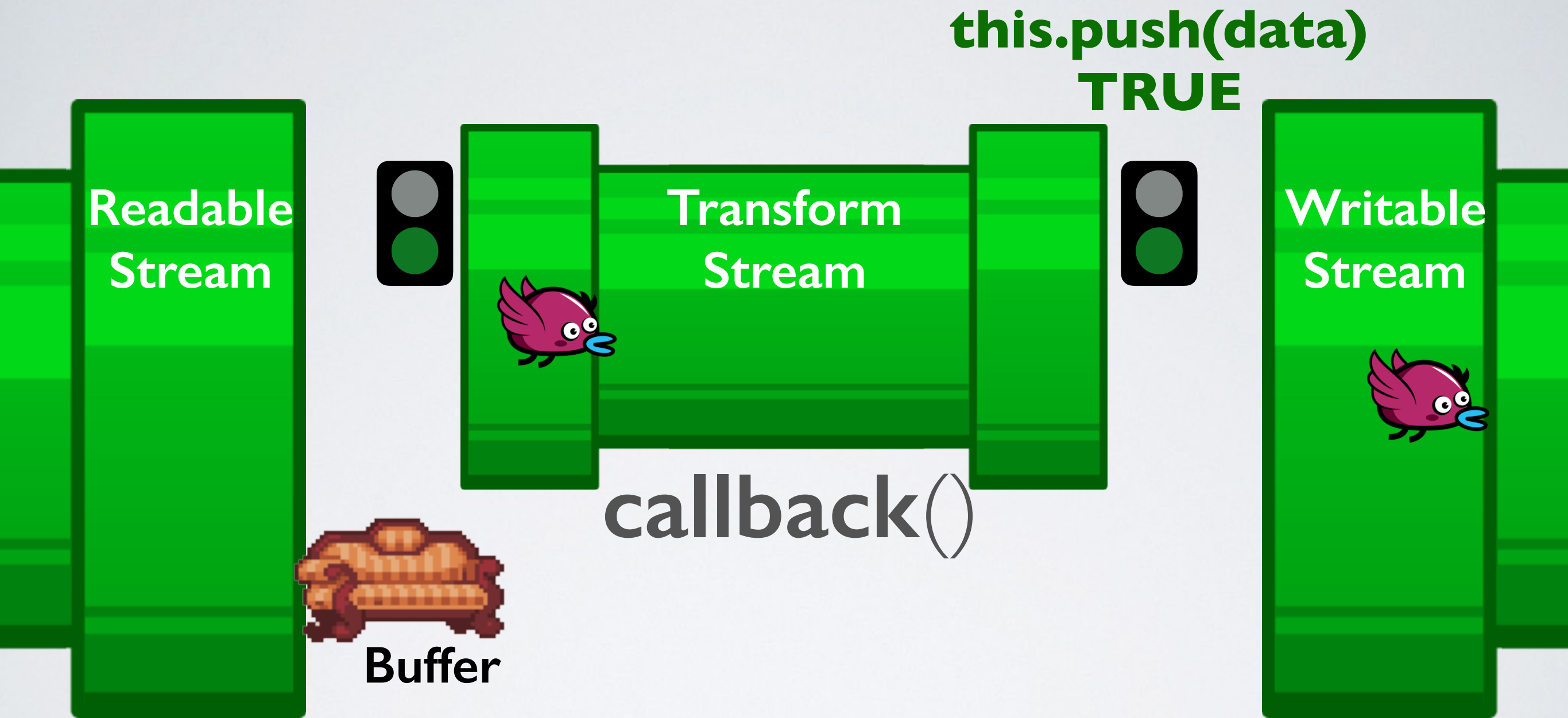
Buffer

**this.push(data)**  
**TRUE**

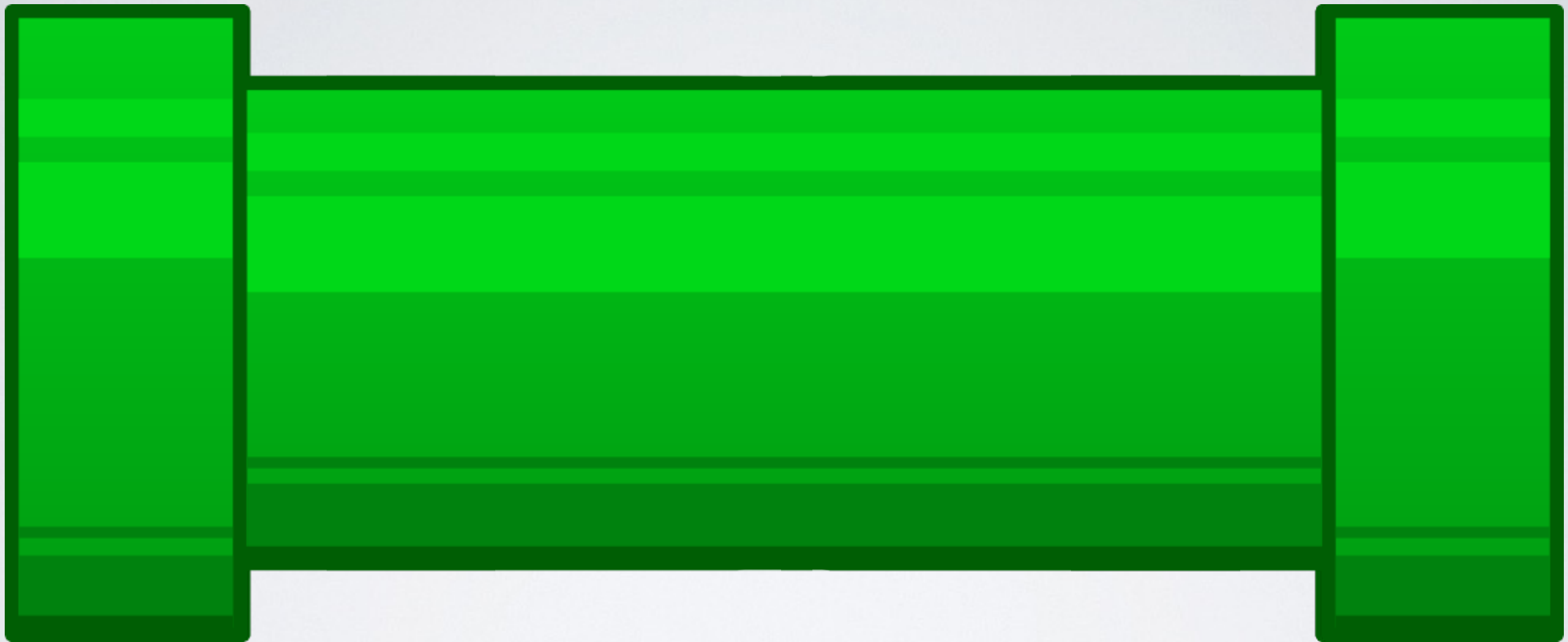








# DUPLEX STREAM

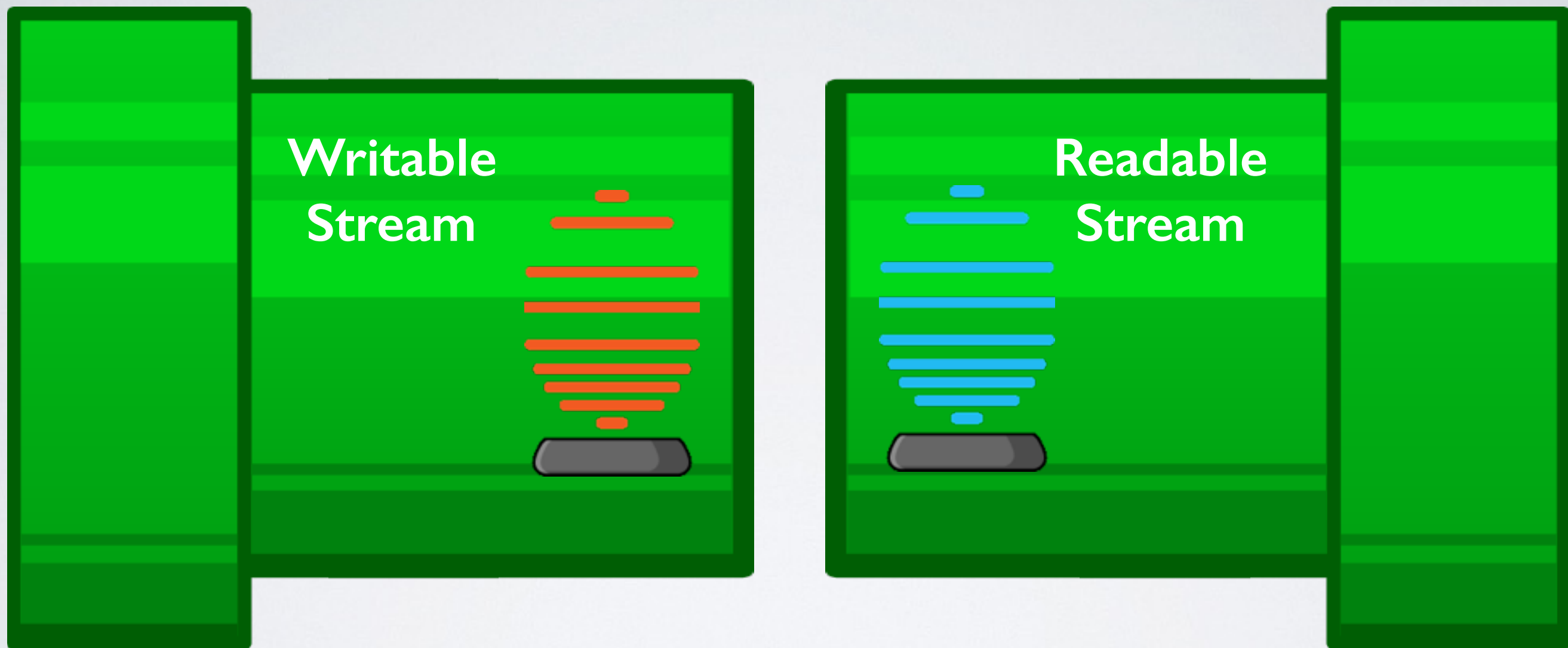




# DUPLEX STREAM



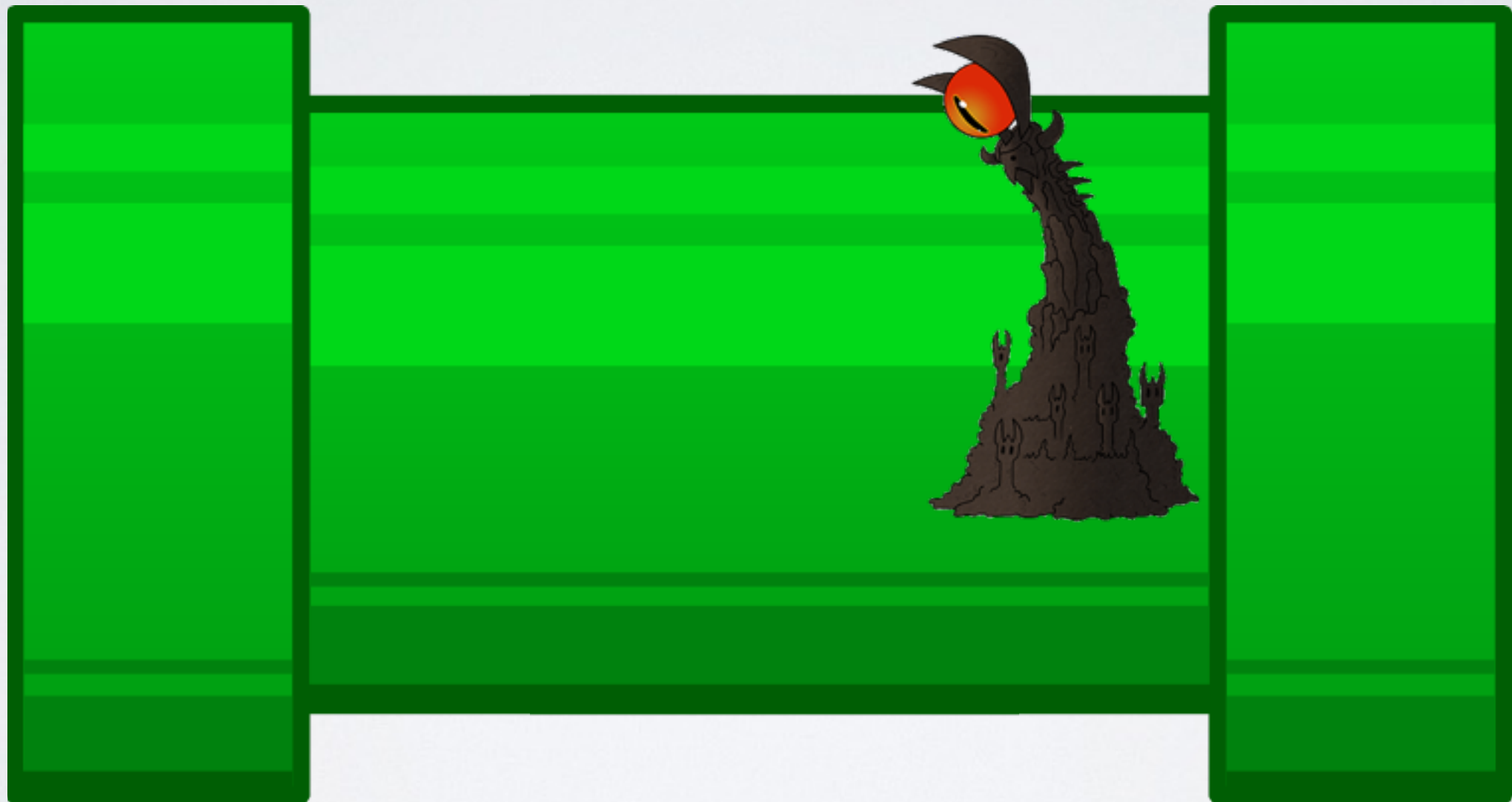
# DUPLEX STREAM



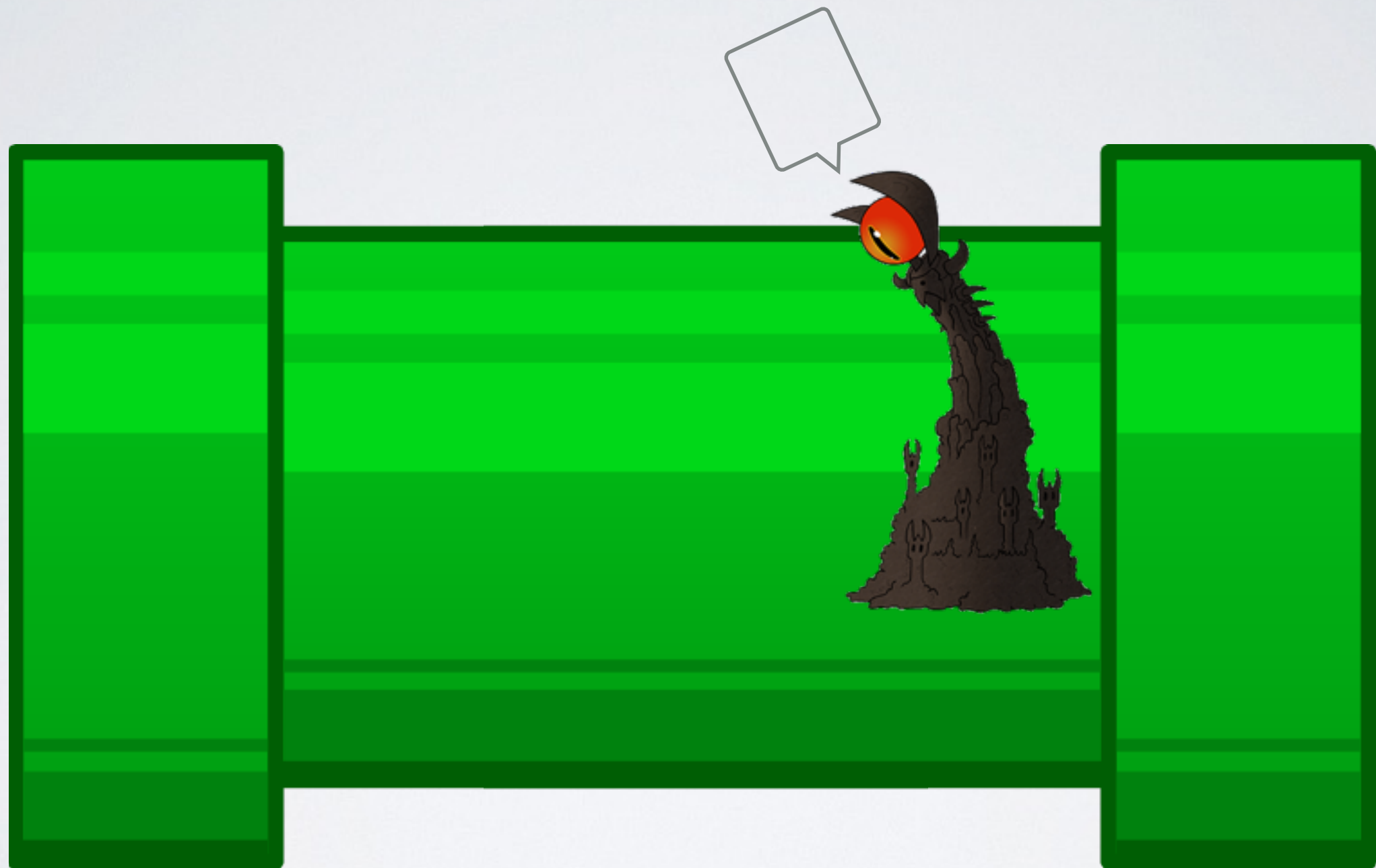
# PASSTHROUGH STREAM



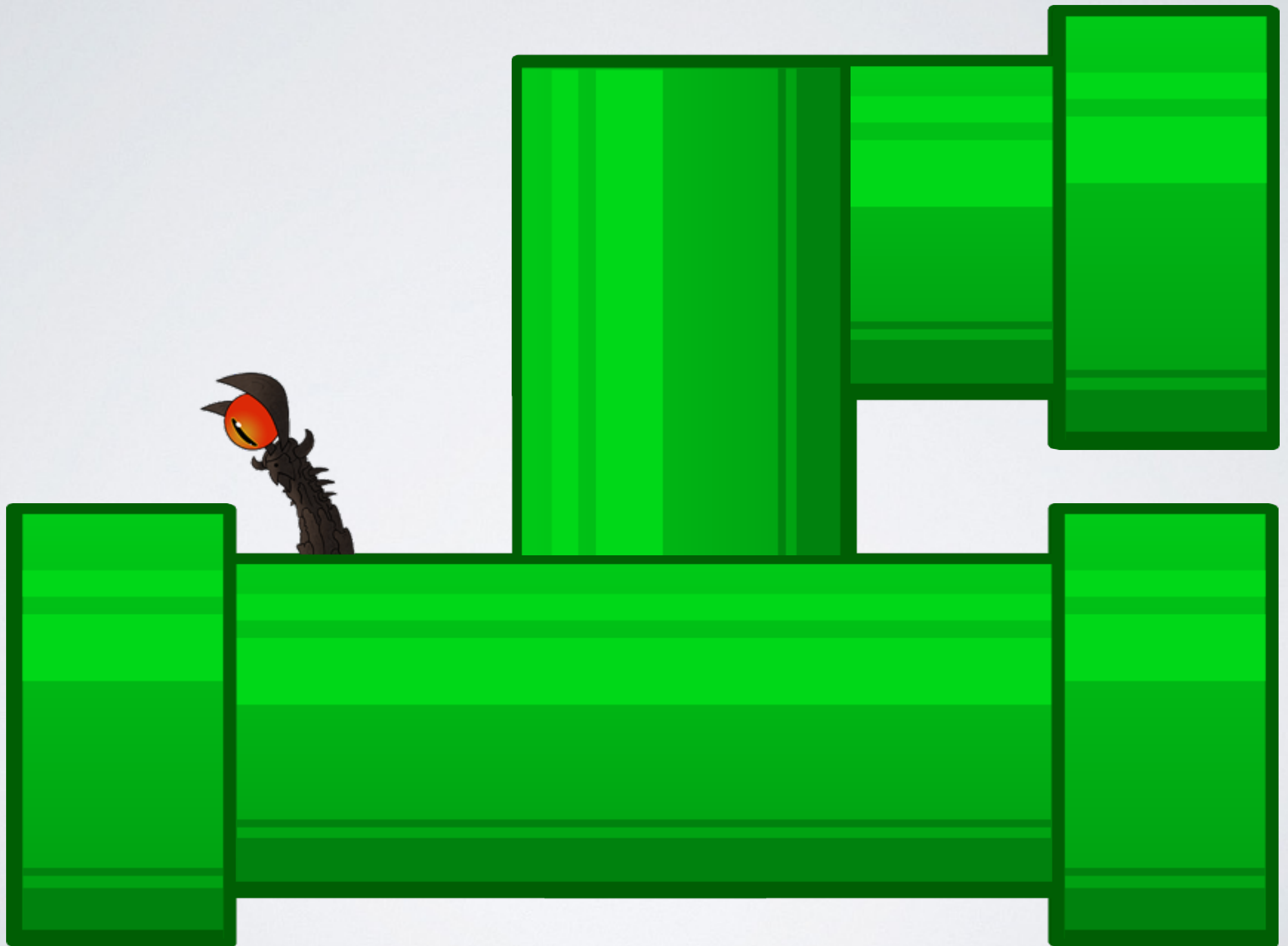
# PASSTHROUGH STREAM



# PASSTHROUGH STREAM

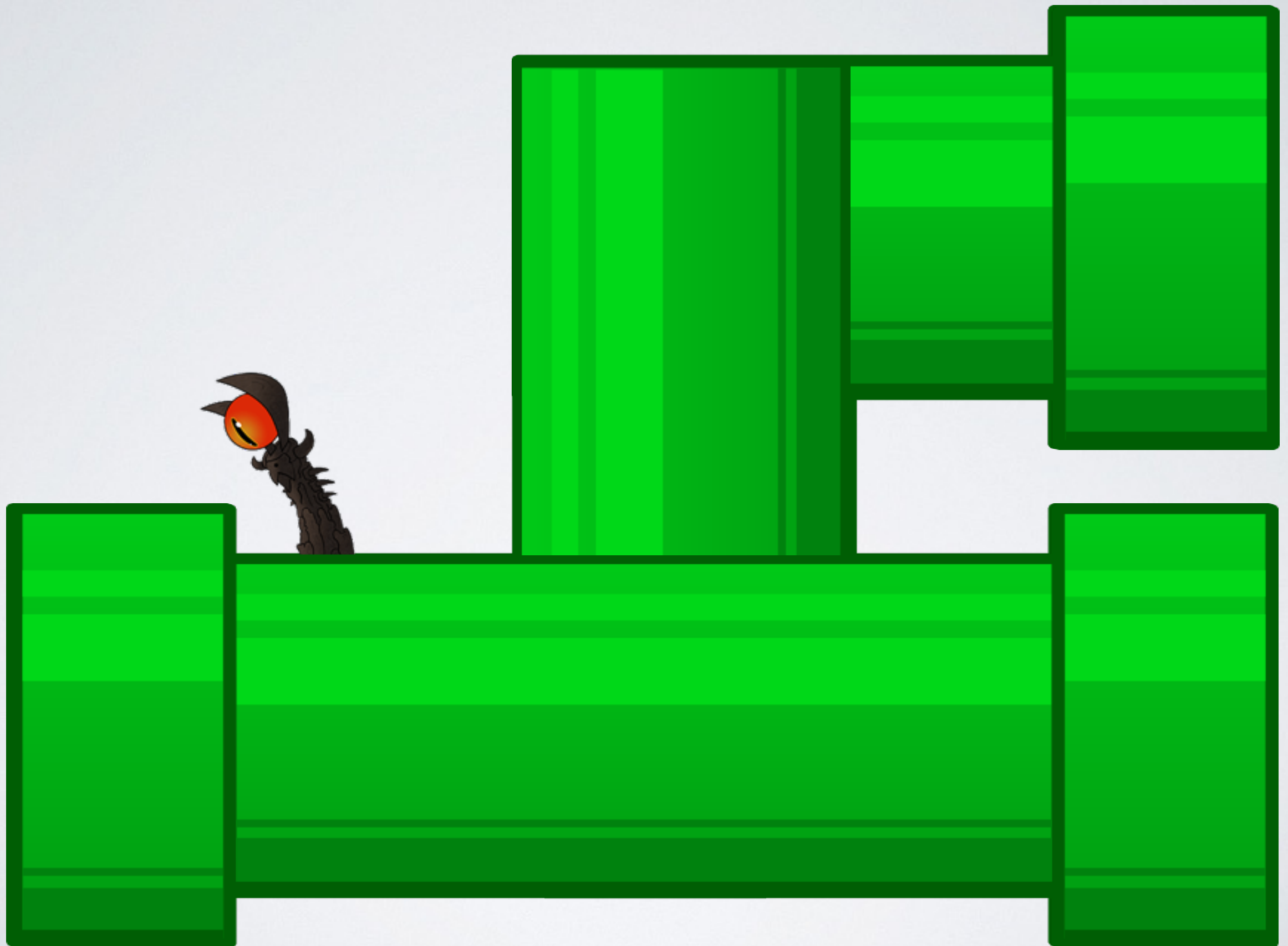


# PASSTHROUGH STREAM

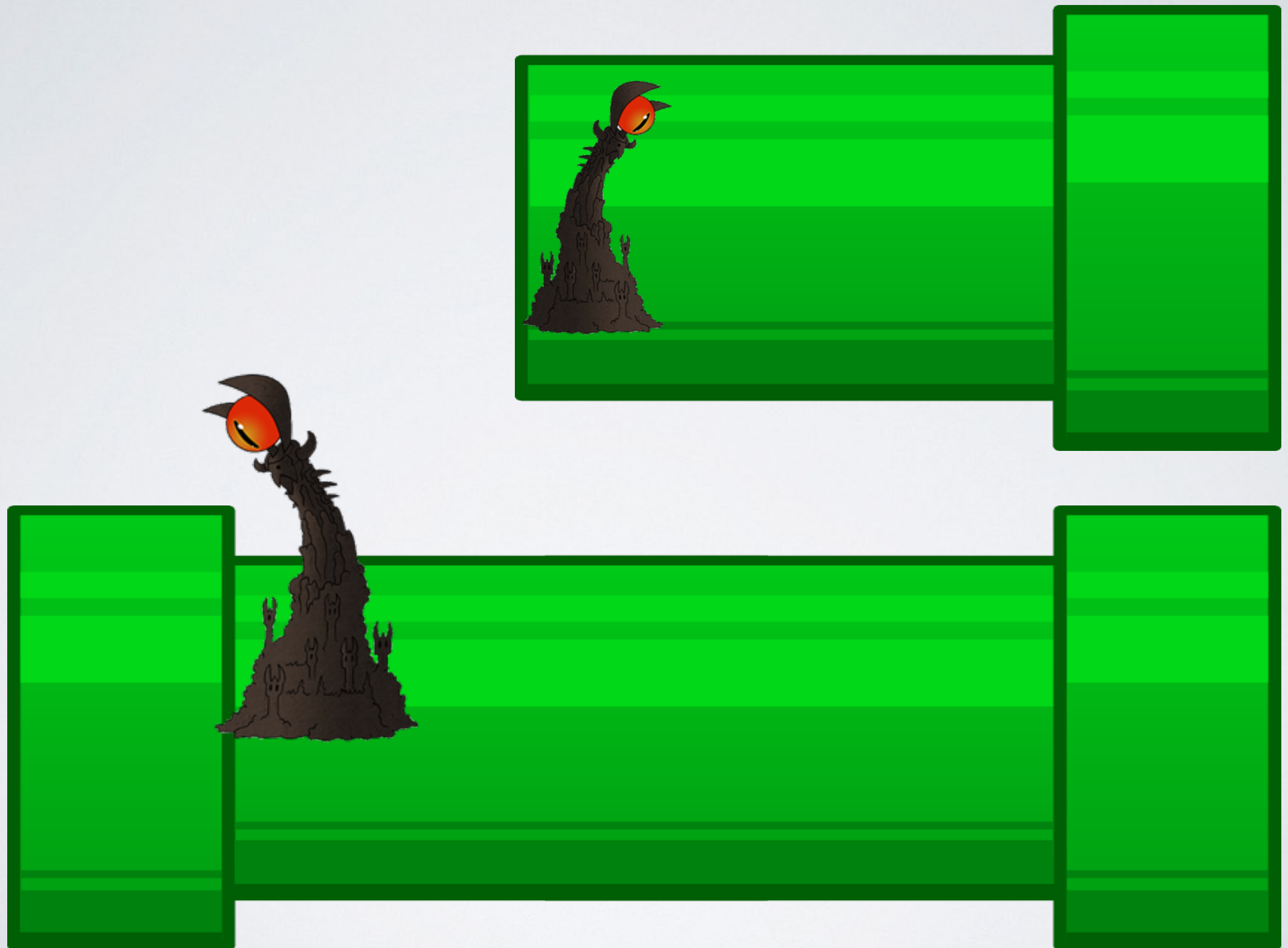




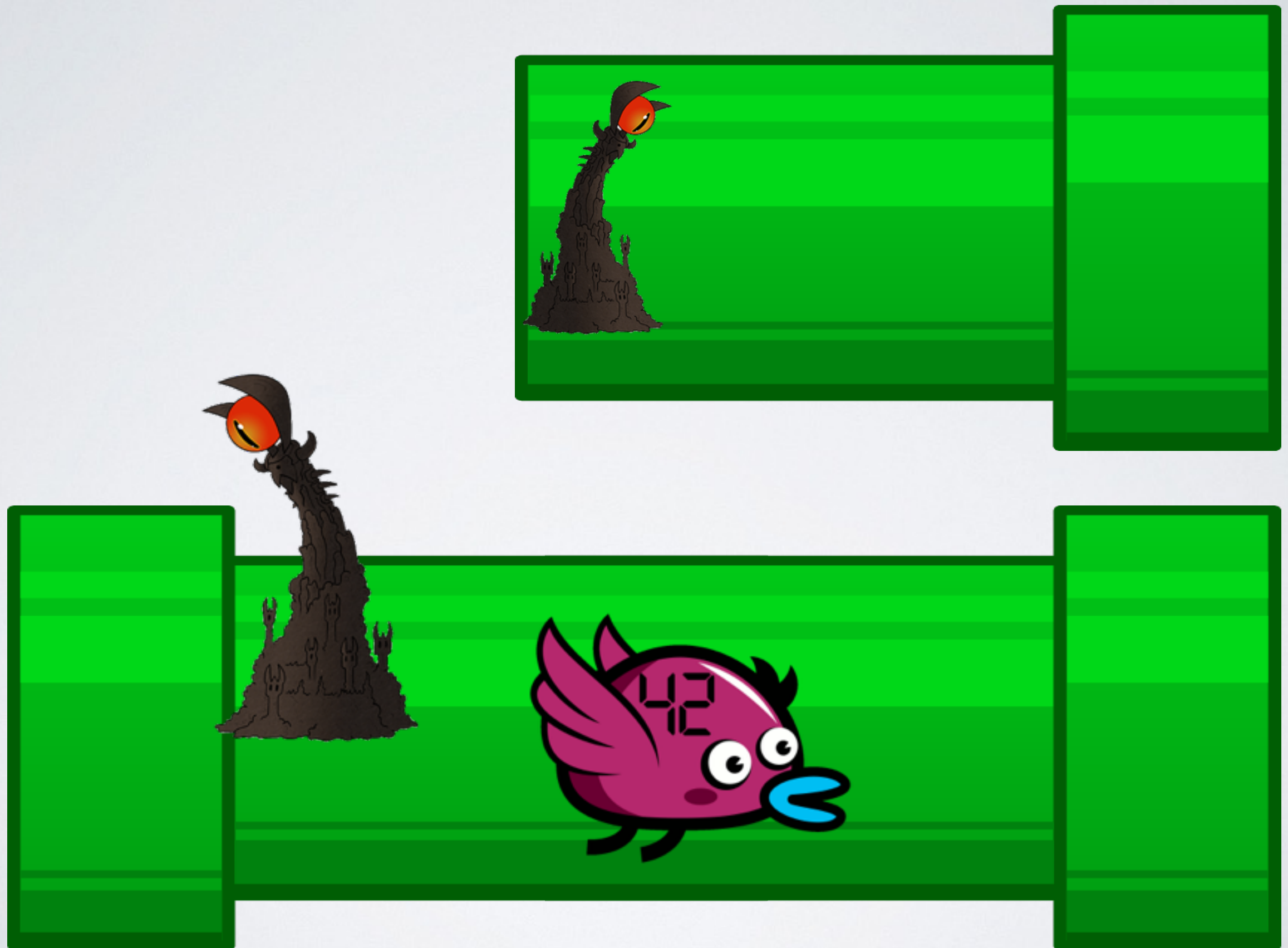
# PASSTHROUGH STREAM



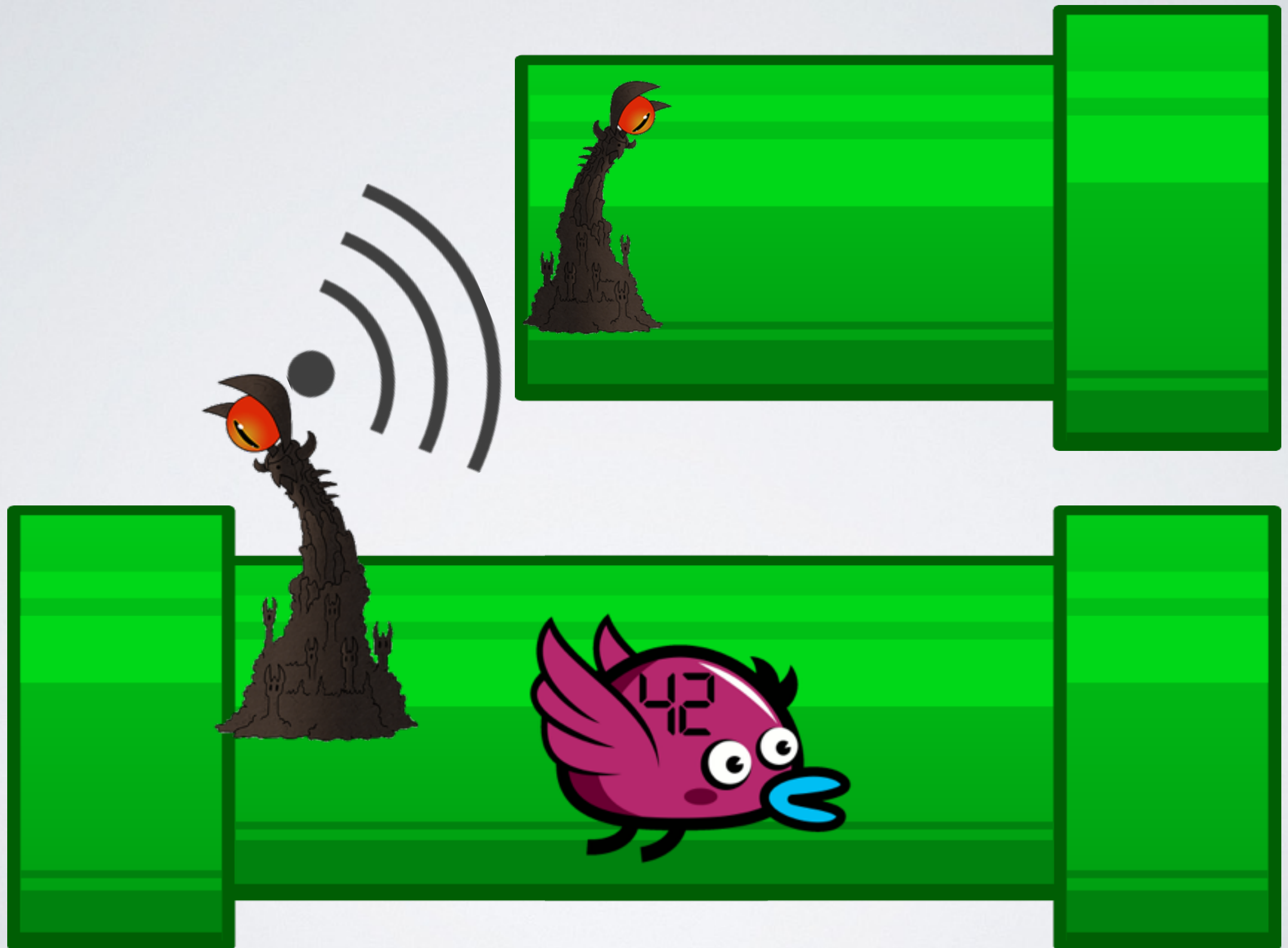
# PASSTHROUGH STREAM



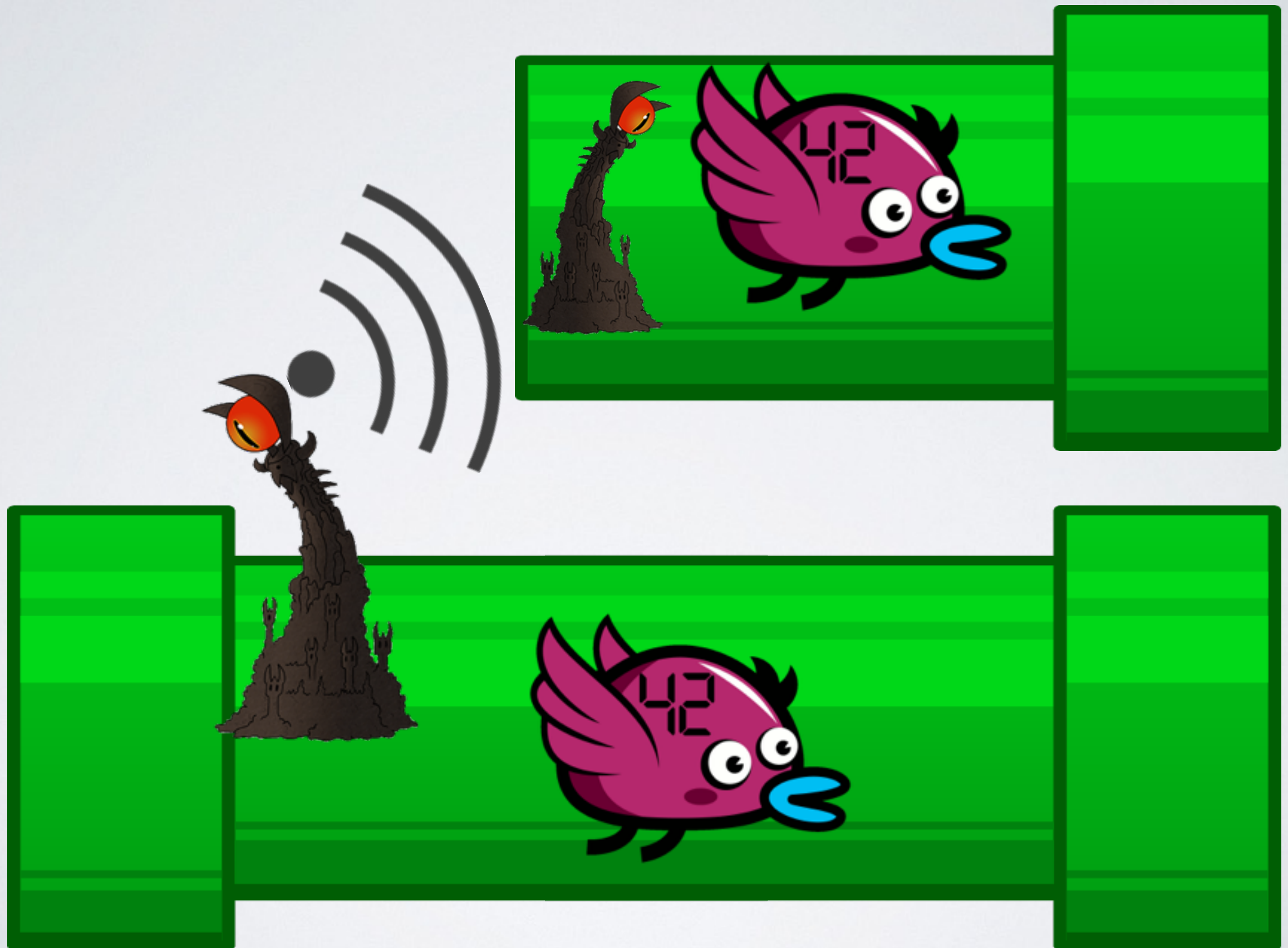
# PASSTHROUGH STREAM



# PASSTHROUGH STREAM



# PASSTHROUGH STREAM





# PASSTHROUGH STREAM





# STREAM BENEFITS



# STREAM BENEFITS

- Lazily produce or consume data in buffered chunks.

# STREAM BENEFITS

- Lazily produce or consume data in buffered chunks.
- Evented and non-blocking

# STREAM BENEFITS

- Lazily produce or consume data in buffered chunks.
- Evented and non-blocking
- Low memory footprint



# STREAM BENEFITS

- Lazily produce or consume data in buffered chunks.
- Evented and non-blocking
- Low memory footprint
- Automatically handle back-pressure

# STREAM BENEFITS

- Lazily produce or consume data in buffered chunks.
- Evented and non-blocking
- Low memory footprint
- Automatically handle back-pressure
- Buffers allow you to work around the V8 heap memory limit



# STREAM BENEFITS

- Lazily produce or consume data in buffered chunks.
- Evented and non-blocking
- Low memory footprint
- Automatically handle back-pressure
- Buffers allow you to work around the V8 heap memory limit
- Most core Node.js content sources/sinks are streams already!

# STREAMS CLASSES



# STREAMS CLASSES

- Readable -- Data Sources

# STREAMS CLASSES

- Readable -- Data Sources
- Writable -- Data Sinks



# STREAMS CLASSES

- Readable -- Data Sources
- Writable -- Data Sinks
- Duplex -- Both a Source and a Sink

# STREAMS CLASSES

- Readable -- Data Sources
- Writable -- Data Sinks
- Duplex -- Both a Source and a Sink
- Transform -- In-flight stream operations



# STREAMS CLASSES

- Readable -- Data Sources
- Writable -- Data Sinks
- Duplex -- Both a Source and a Sink
- Transform -- In-flight stream operations
- Passthrough -- Stream spy

# HOW TO IMPLEMENT



# HOWTO IMPLEMENT

- Use handy abstractions like `mississippi` module (easy way)

# HOWTO IMPLEMENT

- Use handy abstractions like `mississippi` module (easy way)
- Subclass appropriate Stream Class and implement required methods, i.e., `_read()`, `_write()`, etc (hard way)



# github.com/maxogden/mississippi

a collection of useful stream utility modules

```
var miss = require('mississippi')
```

- **from** - Make a custom readable stream
- **to** - Make a custom writable stream
- **through** - Make a custom transform stream.
- **duplex** - Take two separate streams, a writable and a readable, and turn them into a single duplex (readable and writable) stream.
- **pipeline** - Combine streams together

check [github](#) for example code

# LINKS

<https://github.com/bionode-hack/discussions#useful-nodejs-modules>

<https://github.com/bionode-hack/discussions#presentation-slides>

Follow @maxogden, @mafintosh and @substack



# ACKNOWLEDGMENTS

Organisers



Community



Sponsor



Research group



Venue



Friends



# IMAGES SOURCES

<http://opengameart.org/content/bevouliin-green-flappy-bird-sprite-sheets>

<http://neoriceisgood.deviantart.com/art/100-furniture-sprites-405058884>

<http://android272.deviantart.com/art/Teeworlds-Teleport-570298308>

<http://www.how-to-draw-cartoons-online.com/eye-of-sauron.html>