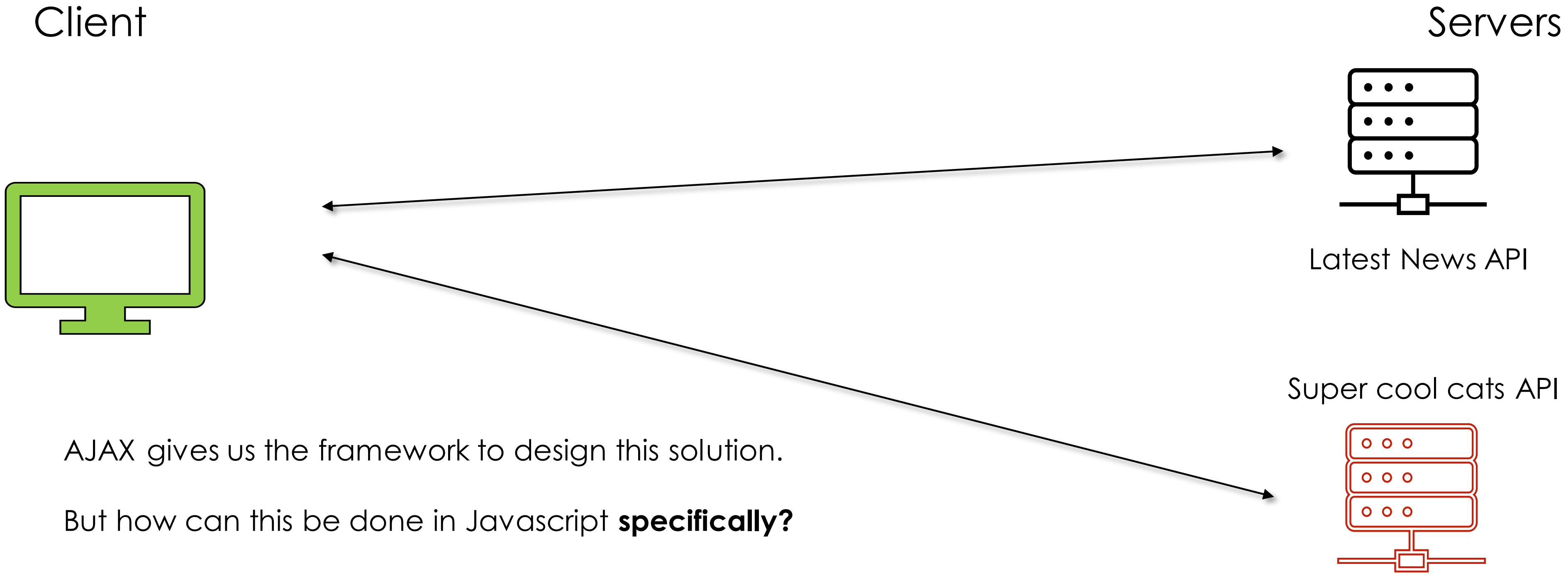
ASYNCHRONOUS NETWORKING

Concurrency & JS

OVERVIEW

- Client-Server Model + AJAX
- Concurrency & JS
- Networking with `XMLHttpRequest()`
- Networking with Promises & `fetch()`
- Networking with `async/await & fetch()`

RECAP



AJAX gives us the framework to design this solution.

But how can this be done in Javascript **specifically?**

Before answering that, need to define what **asynchronous** means

CONCURRENCY 101

Synchronous Programming

- Program executes top-down
- Guaranteed that previous instructions fully complete before executing the next ones
- Very simple to reason about

Asynchronous Programming

- Program has multiple flows of control (called “threads”)
- Threads can interleave or run at the same time
- Much harder to reason about

CONCURRENCY 101 (CONT.)

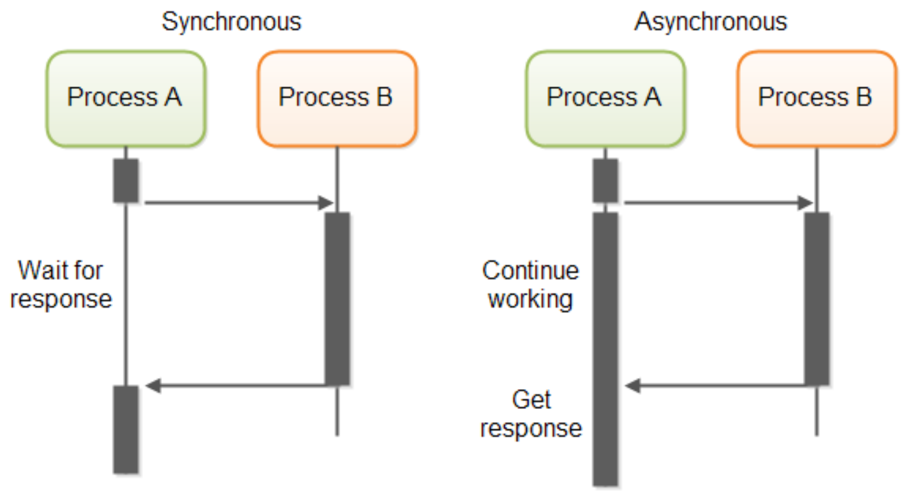


Image credit: Medium

- Synchronous
  - Process A executes first
  - Waits for Process B to finish
  - Then continues

- Asynchronous
  - Process executes first
  - Asks Process B to do its work
  - Whilst Process B is executing, Process A also continues
  - At some point in the future, Process A gets Process B's result

CONCURRENCY MODELS

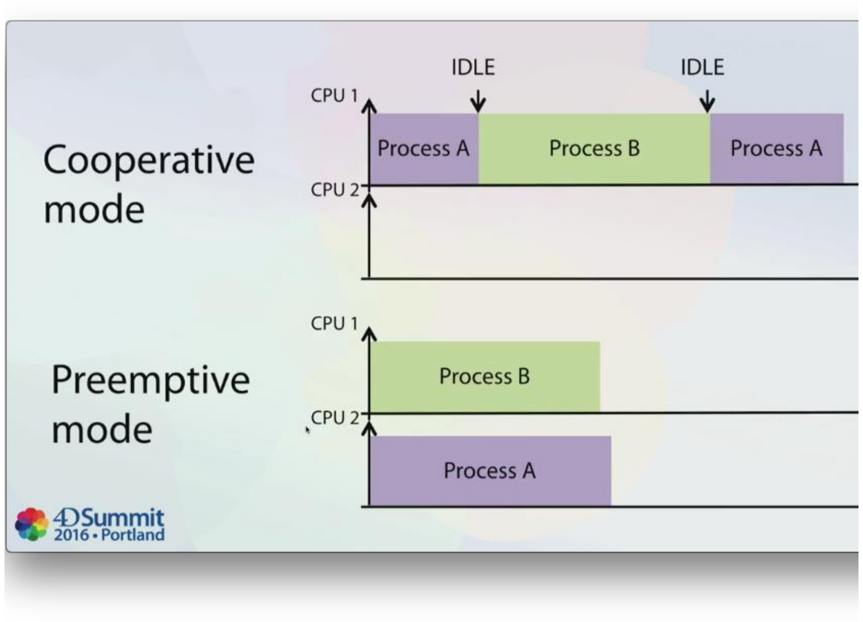
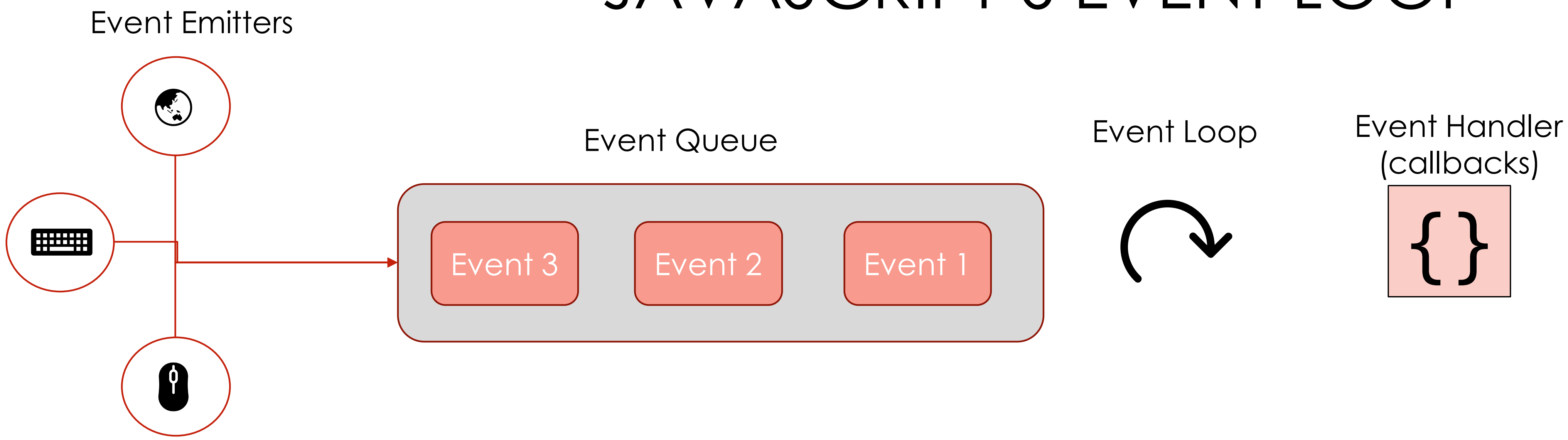


Image Credit: 4D

- Two primary "models" of concurrency:
  - **Pre-emptive multitasking**
  - **Co-operative multitasking**
- Pre-emptive Multitasking:
  - Code runs in parallel on different CPU cores
  - Really good for CPU-intensive workloads
  - Not important to us here
- Co-operative Multitasking:
  - Tasks that need to be done are queued up and executed in a loop
  - Really good for IO-intensive workloads
  - Get the benefits of asynchronous programming with the reasonability of synchronous programming
- Javascript's model is *Co-operative Multitasking*:
  - Tasks that need to be done are "events"
  - Event-handlers are scheduled to execute by the event loop
  - Each event handler runs **synchronously**

JAVASCRIPT'S EVENT LOOP



Each JS Engine generally follows this procedure:

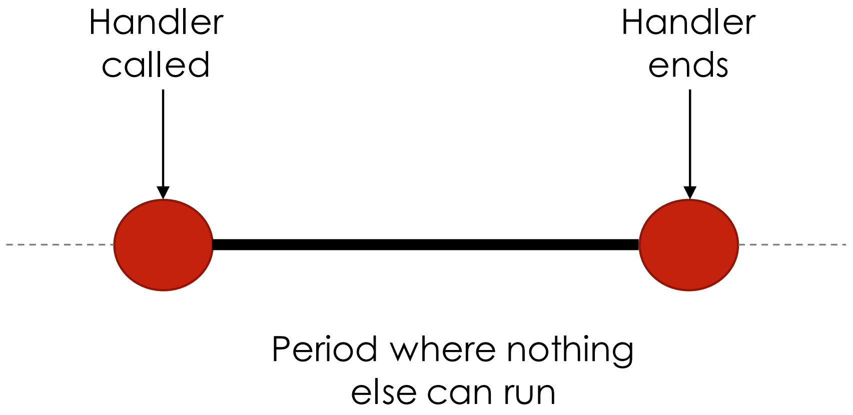
1. Execute your JS script top-down, registering any handlers
2. Wait for events in a loop
3. If an event comes and there is a handler for it, execute the handler ***to completion***
4. Repeat (2) ad infinitum

Important:
Browsers handle:
- Counting down timers
- Networking
- Adding to the event queue

EVENT LOOP DEMO

See examples/event-loop

EVENT LOOP CONSIDERATIONS

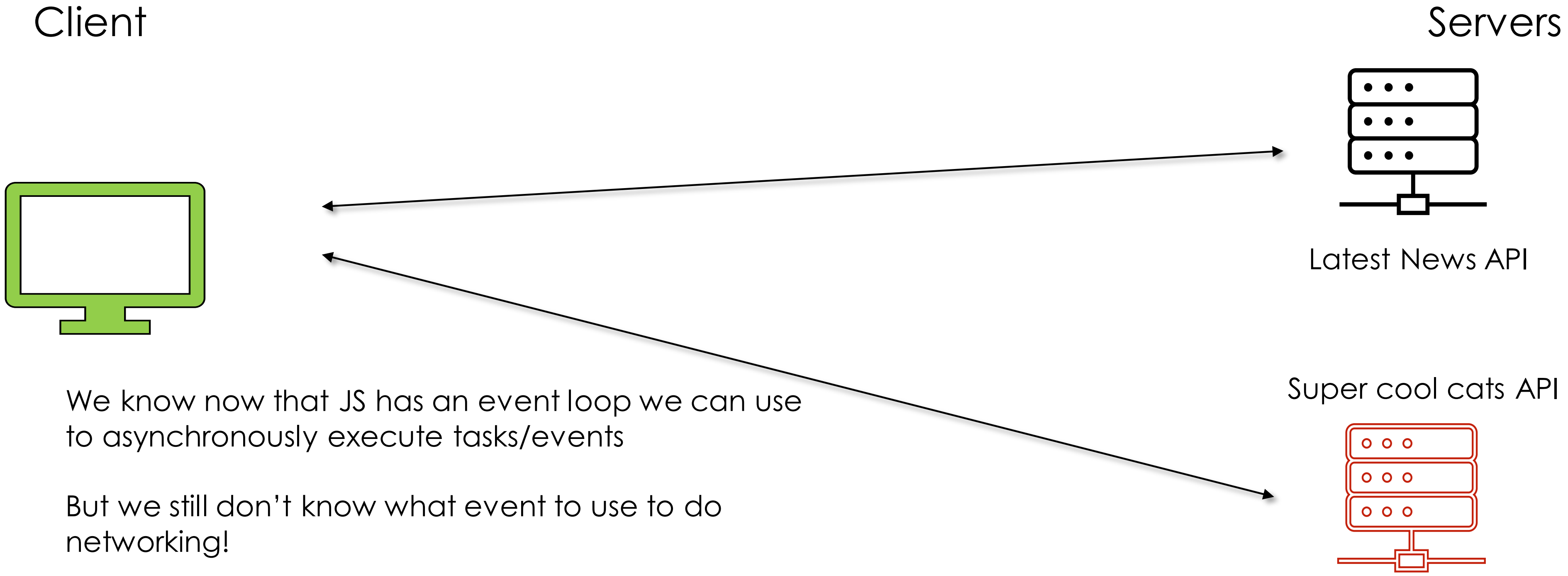


- The event loop runs on a **single** thread
  - Long-running event handlers block everything else
  - Makes the page unresponsive
- Tips on avoiding blocking:
  - Make sure all networking is done asynchronously
  - CPU-intensive calculations should be pushed to a backend server to process

BLOCKING THE LOOP DEMO

See examples/blocking-the-loop

PROGRESSING



We know now that JS has an event loop we can use to asynchronously execute tasks/events

But we still don't know what event to use to do networking!

The first of 3 approaches is the focus of next lecture

SUMMARY

- Today:
  - A brief introduction to concurrency
  - How concurrency is done in Javascript
- Coming Up Next:
  - Networking with `XMLHttpRequest()`