JÖNKÖPING UNIVERSITY

School of Engineering

HANDLING CONCURRENT HTTP REQUESTS

Peter Larsson-Green

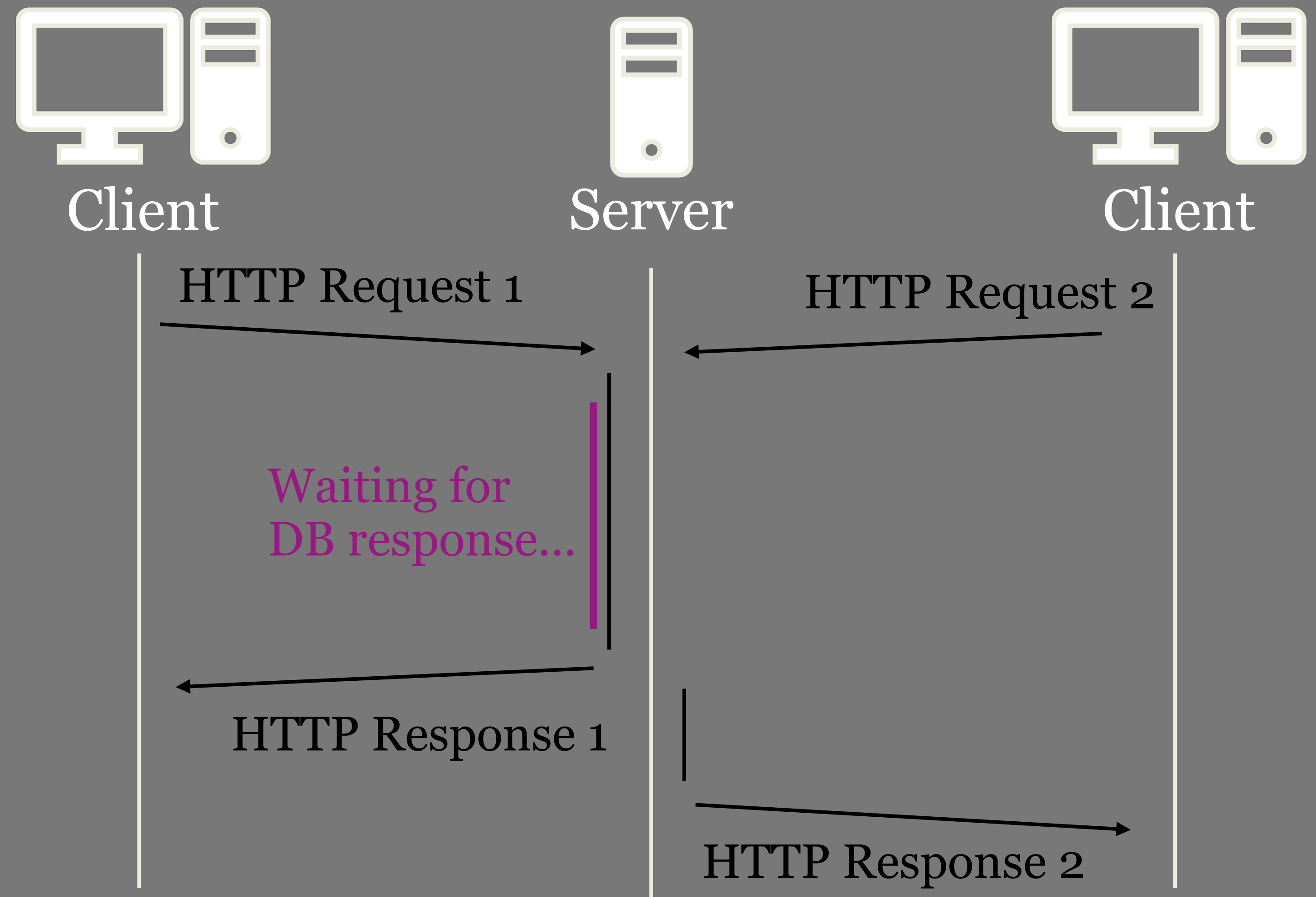
Lecturer at Jönköping University

Spring 2019

HANDLING CONCURRENT REQUESTS

Attempt 1: Process one request at a time, queue the others.

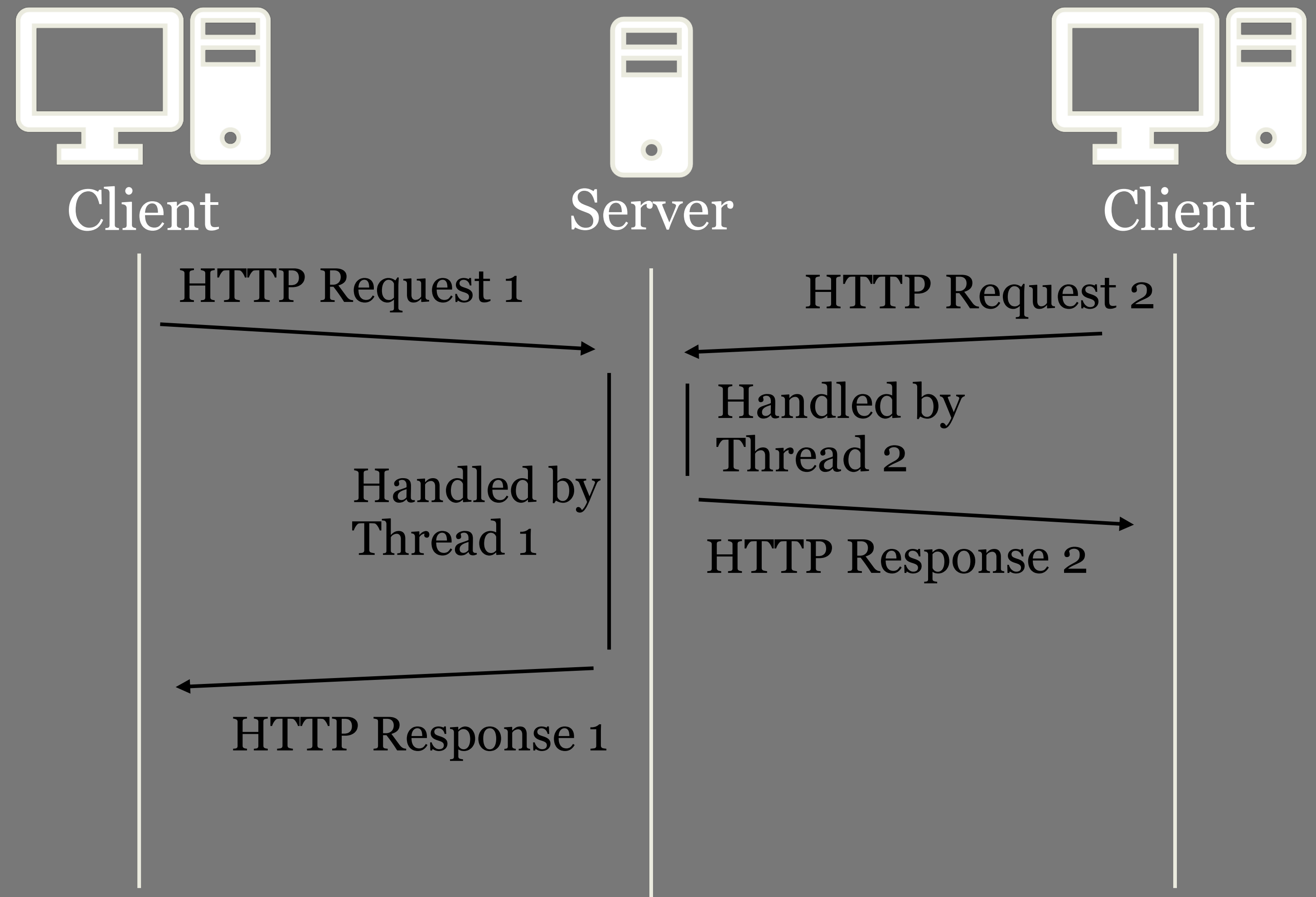
- Bad: Most time wasted on waiting, e.g.:
  - Waiting for DB.
  - Waiting for reading/writing files.
- Very few web applications works this way today.



HANDLING CONCURRENT REQUESTS

Attempt 2: Use threads to process requests simultaneously.

- Requires us to write thread-safe code.
- The way many web applications work still today.
    - Then came Node.js...



HANDLING CONCURRENT REQUESTS

Example

HANDLING CONCURRENT REQUESTS

Attempt 3: Use a single thread with an event loop.

- The event queue contains tasks to be done.
  - Incoming HTTP request are pushed to it.
  - Asynchronous operations are pushed to it.
- The event loop executes tasks from the event queue.

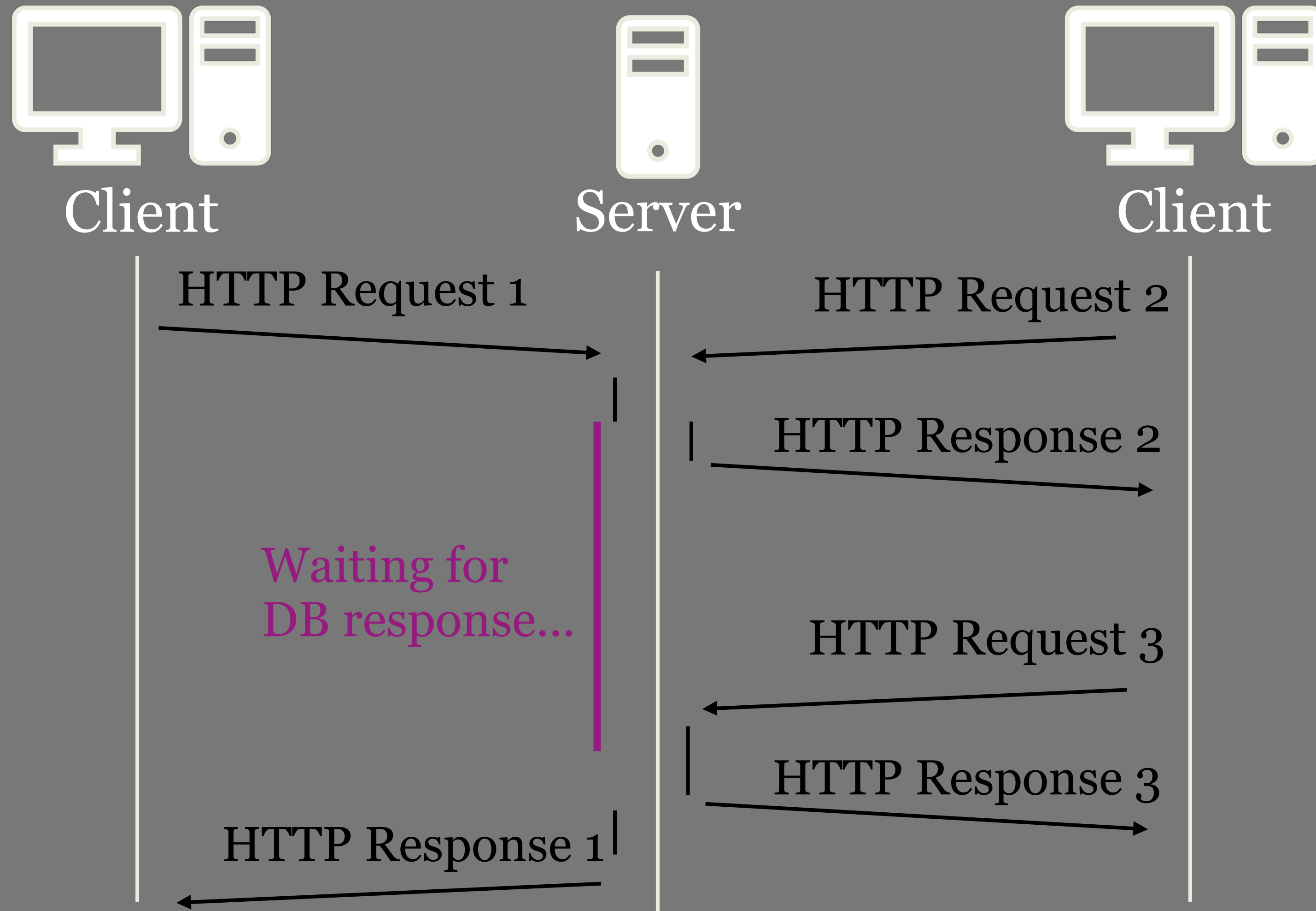
Event Loop/Main code

```
// "pseudocode"
const queue = []
while(true){
  const nextTask =
        queue.unshift()
  nextTask.execute()
}
```

Event Queue

Event Queue
Do this
Do that
Do x
Do y

HANDLING CONCURRENT REQUESTS



Event Queue

Handle Request 1
Handle Request 2
Handle Request 3
Handle DB Response

HANDLING CONCURRENT REQUESTS

Attempt 3: Use a single thread with an event loop.

- Why is this better than multiple threads?
  - Context switches (switching thread) are expensive (takes time).
  - Threads uses a lot of memory.
- Any downside?
  - Asynchronous programming must be used; is a bit harder.

SYNC VS ASYNC

```
app.get("/", function(req, res){
  const accounts = getAllAccounts()
  const posts = getAllPosts()
  response.render("some-view.hbs", {accounts, posts})
})
```

```
app.get("/", function(req, res){
  getAllAccounts(function(accounts){
    getAllPosts(function(posts){
      response.render("some-view.hbs", {accounts, posts})
    })
  })
})
```