Evented I/O based web servers, explained using bunnies

Simon Willison

Part of a talk given at Full Frontal 2009

🐰 Your Web Server
(using a bunny)

🐹

🐰 Your Web Server (using a bunny)

Single threaded (one bunny), so can only handle one request at a time

Happy hamster

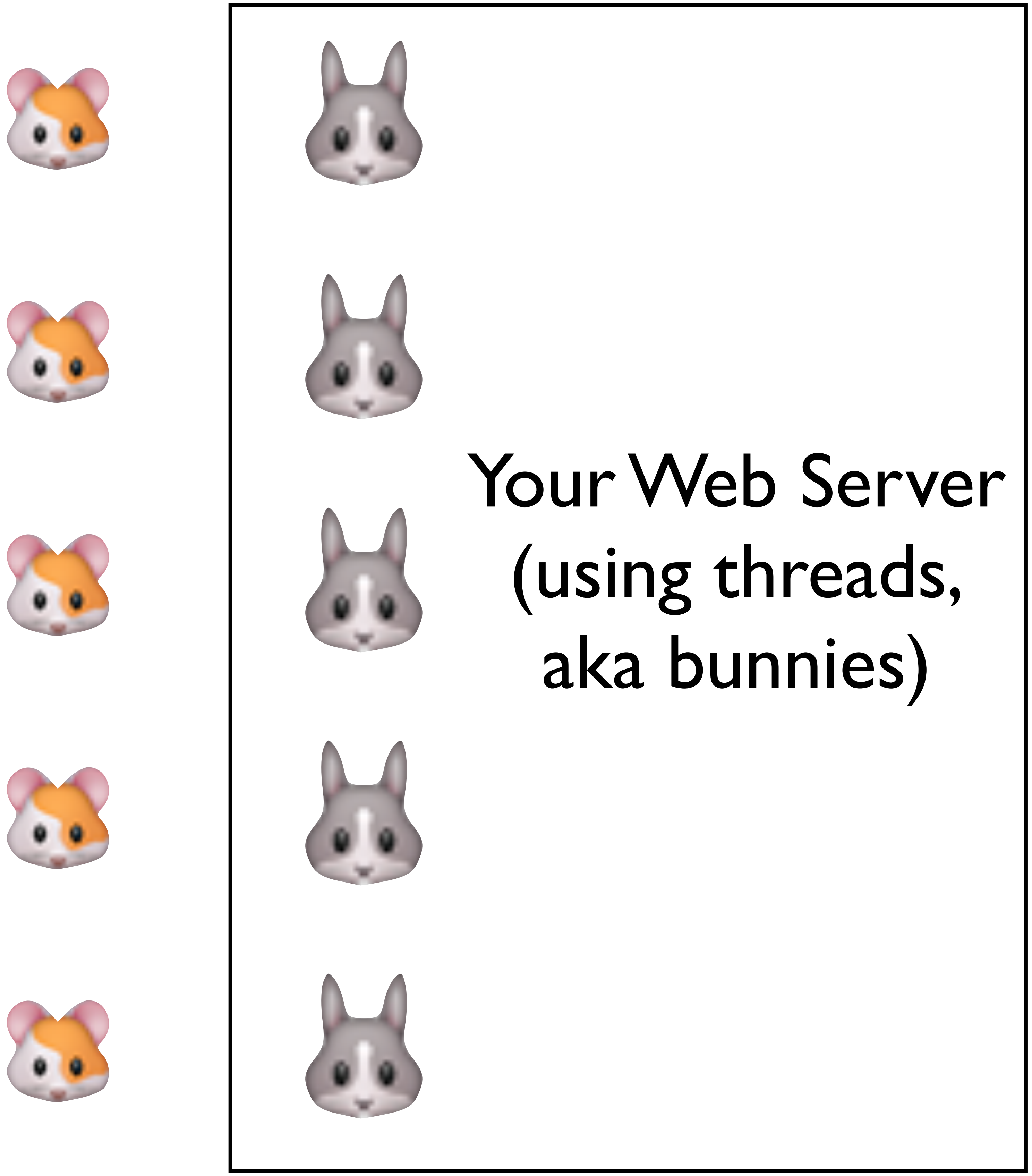
🐹 🐹 🐹

(The hamsters are using web browsers to visit your site)

🐰 Your Web Server (using a bunny)

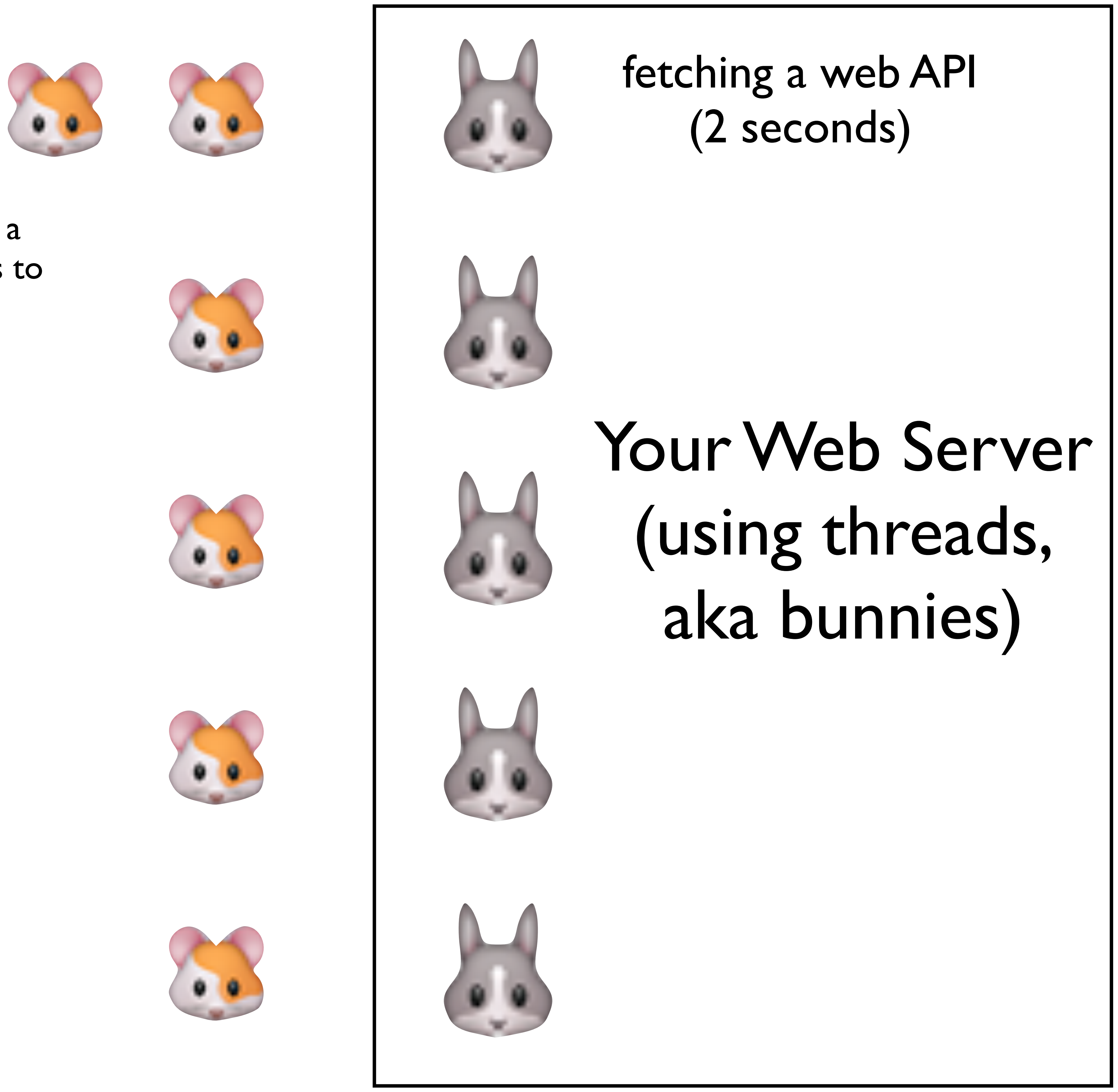
Impatient hamsters

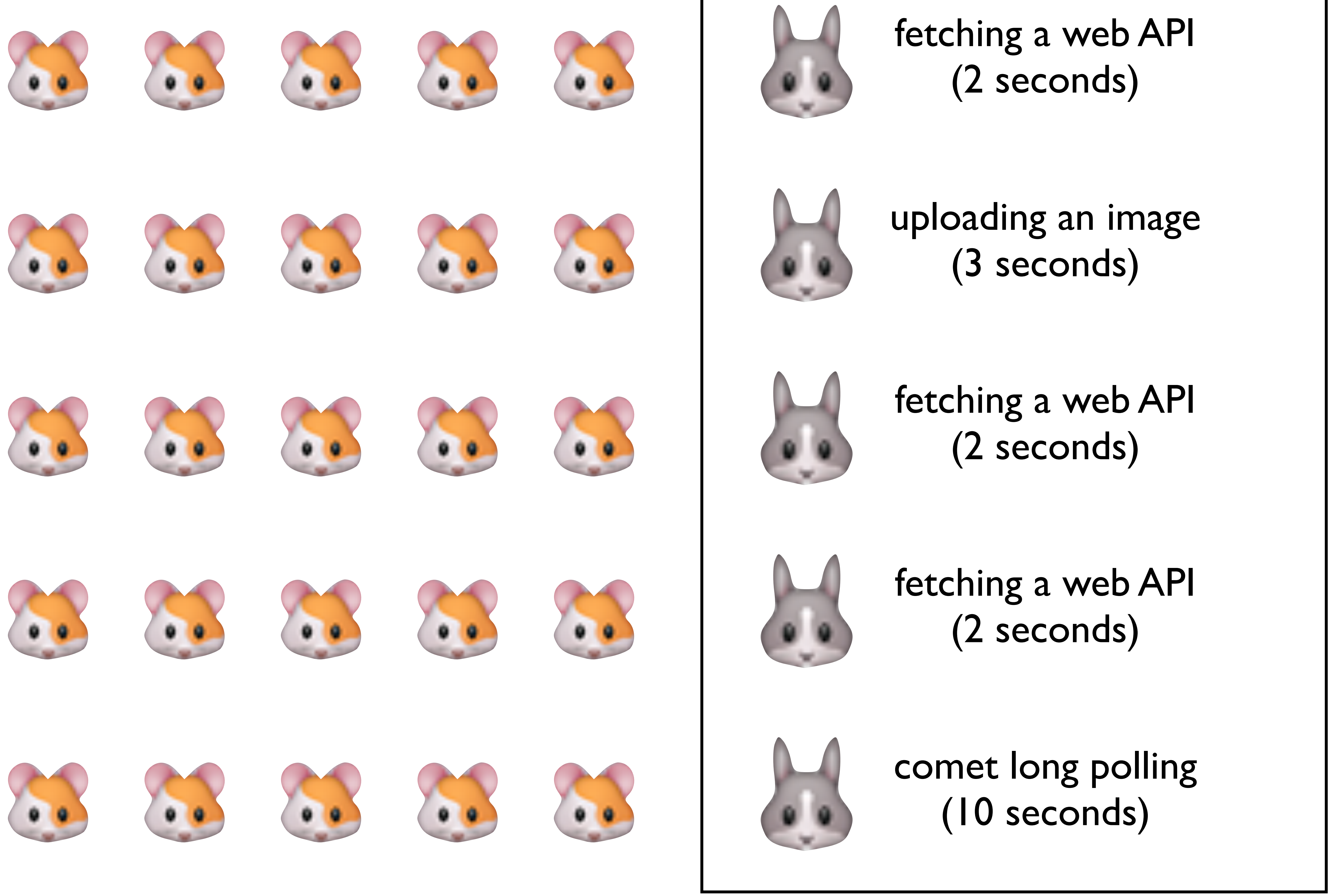
5 bunnies = can handle 5 requests at the same time



Happy hamsters

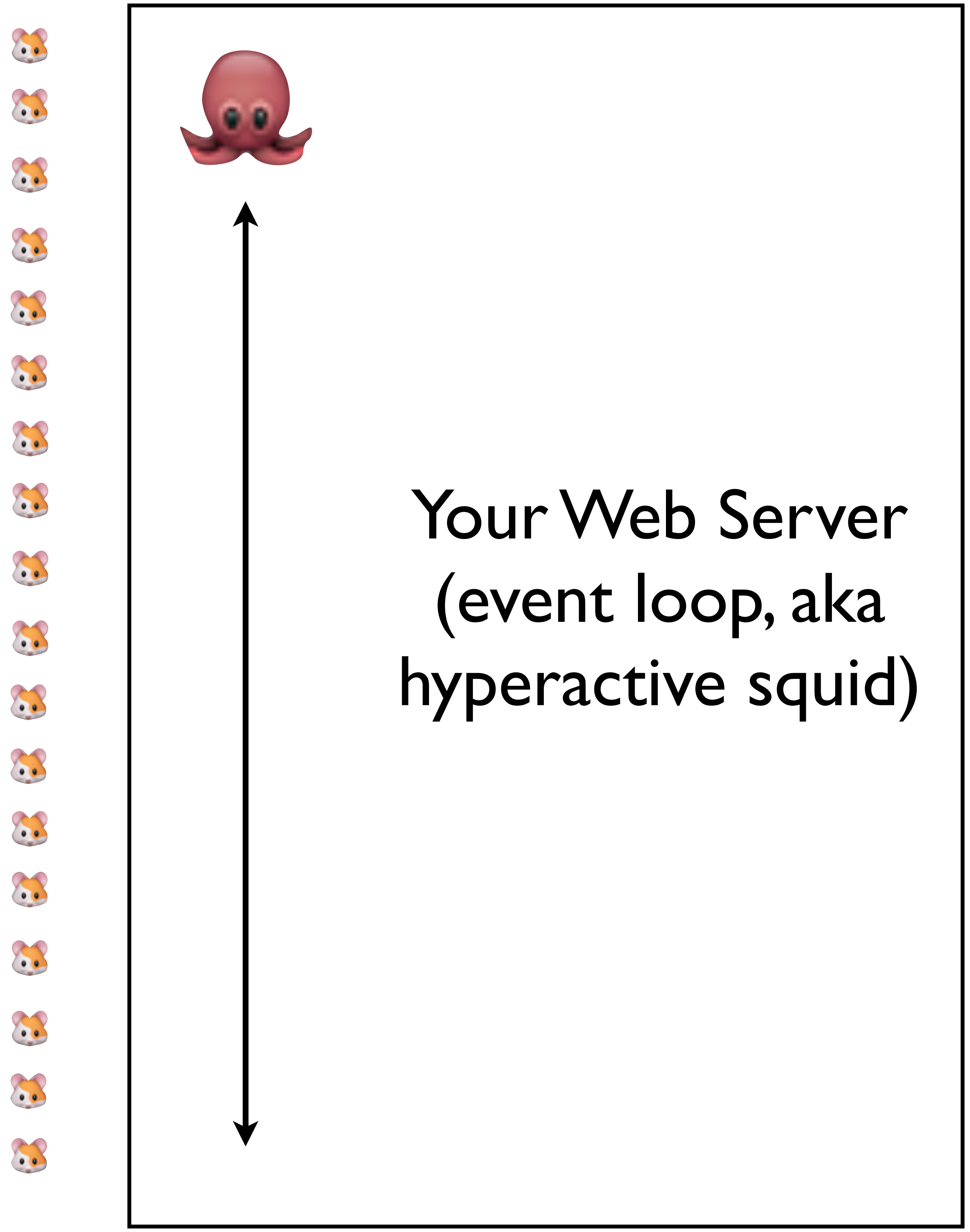
Long running operations cause a thread to block, causing requests to build up in a queue





Impatient hamsters

Replace the bunnies with a single hyperactive squid. The squid runs up and down as fast as it can dealing with each hamster in turn. It fires off any long running I/O operations and then moves on to the next hamster. When the I/O operation reports progress, it does a little more work on behalf of the corresponding hamster.



Bad code

- rows = database.fetch(category = 'news')
- template = read_file('homepage.html')
- json = fetch_url('http://.../')

These functions block and wait for results - blocking the squid and causing the entire event loop (and hence server) to pause until they complete

Good code

- database.fetch(category = 'news', callback)
- read_file('homepage.html', callback)
- fetch_url('http://.../', callback)

These functions specify a callback to be executed as soon as the I/O operation completes