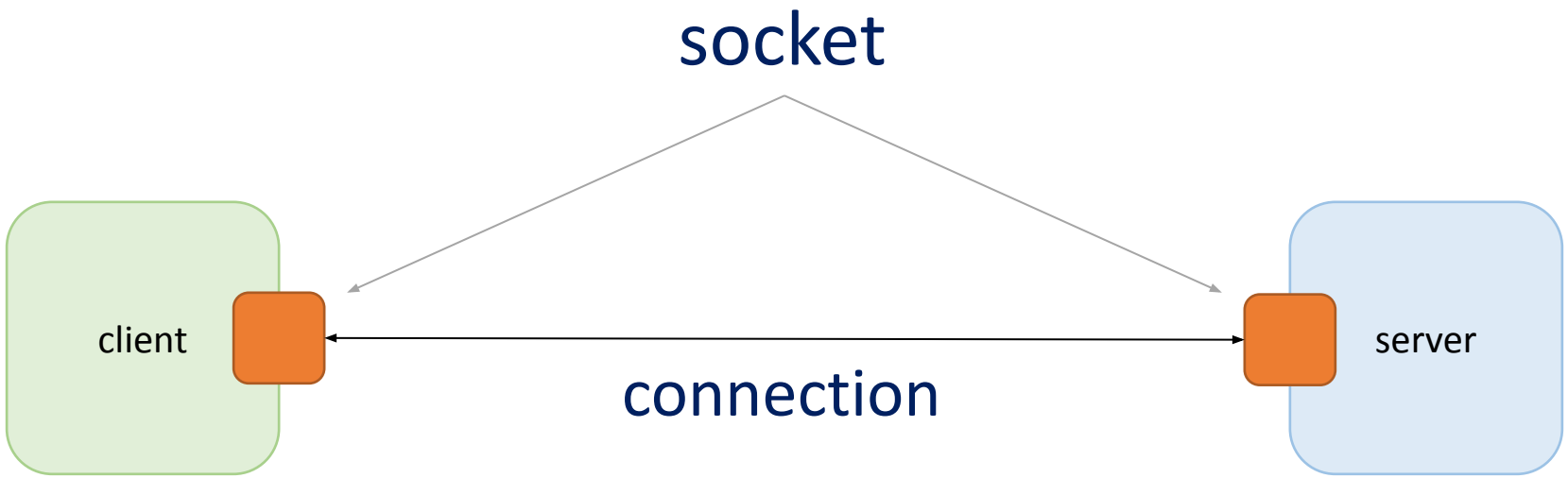
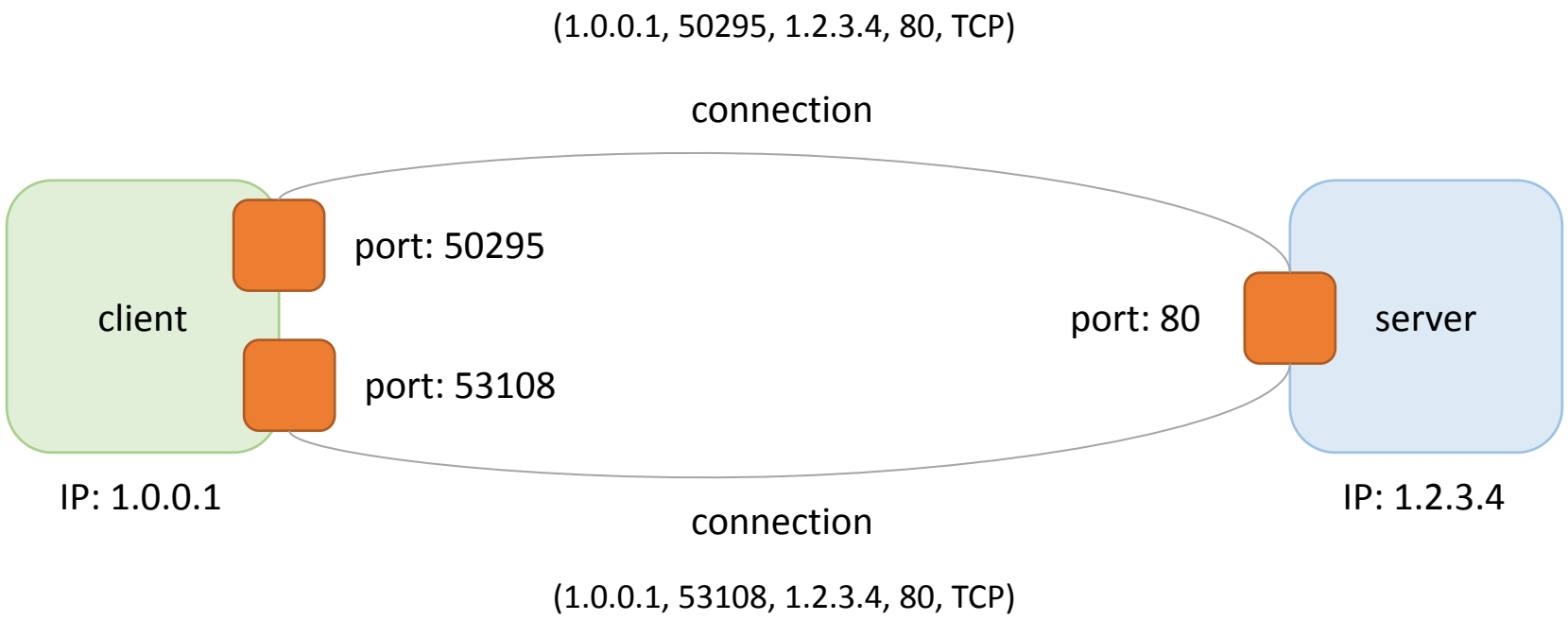
Blocking vs non-blocking I/O

endpoint
(IP address + port number)



tuple
(source IP, source port, destination IP, destination port, protocol)

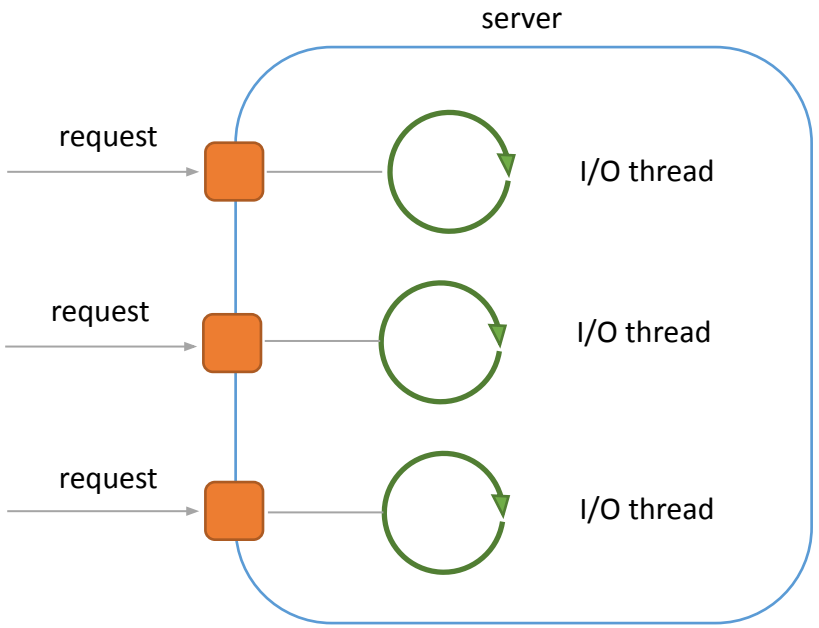
Blocking vs non-blocking I/O



Blocking vs non-blocking I/O

blocking socket

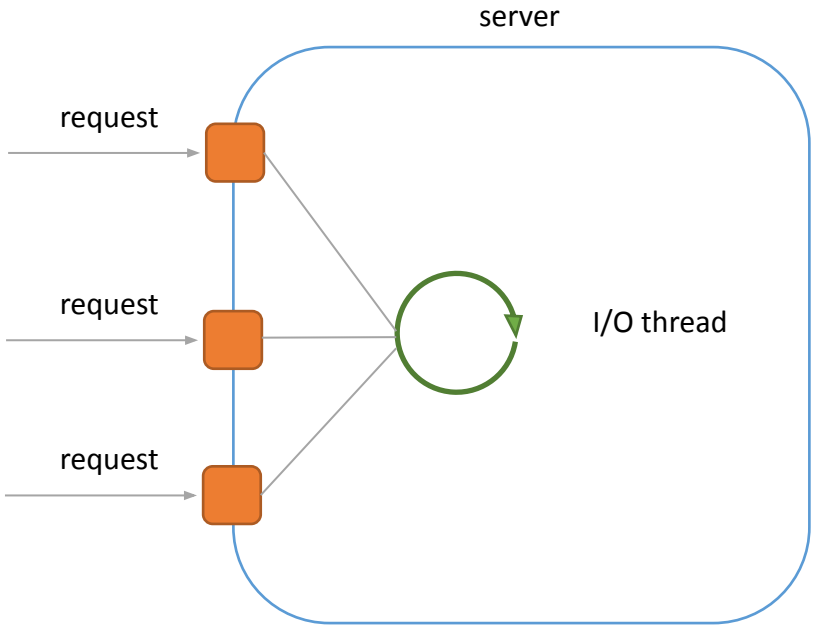
thread is suspended until read/write
from/to the socket completes



blocking I/O

non-blocking socket

thread reads data available in the socket buffer
and does not wait for the remaining data to arrive

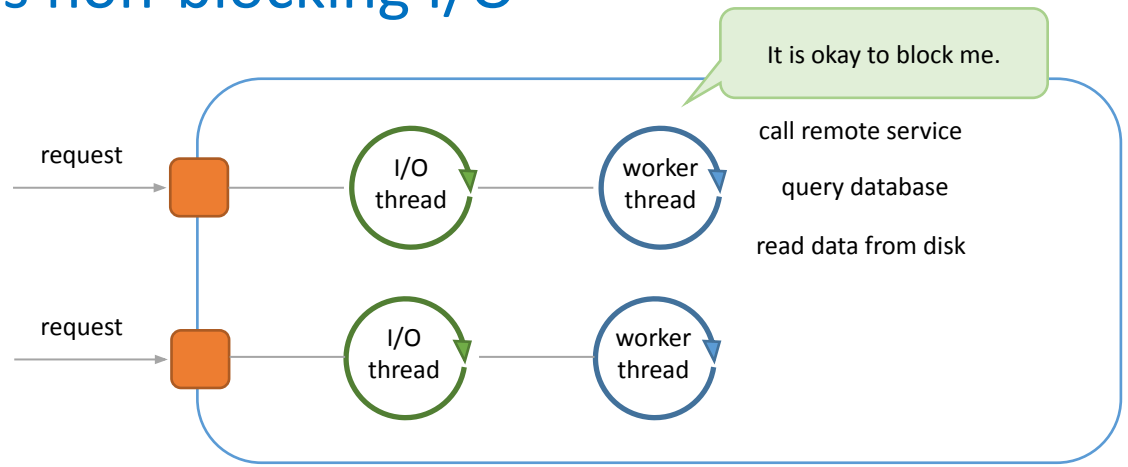


non-blocking I/O

allows the server to handle a lot more connections than blocking I/O,
since connections are cheap and threads are not

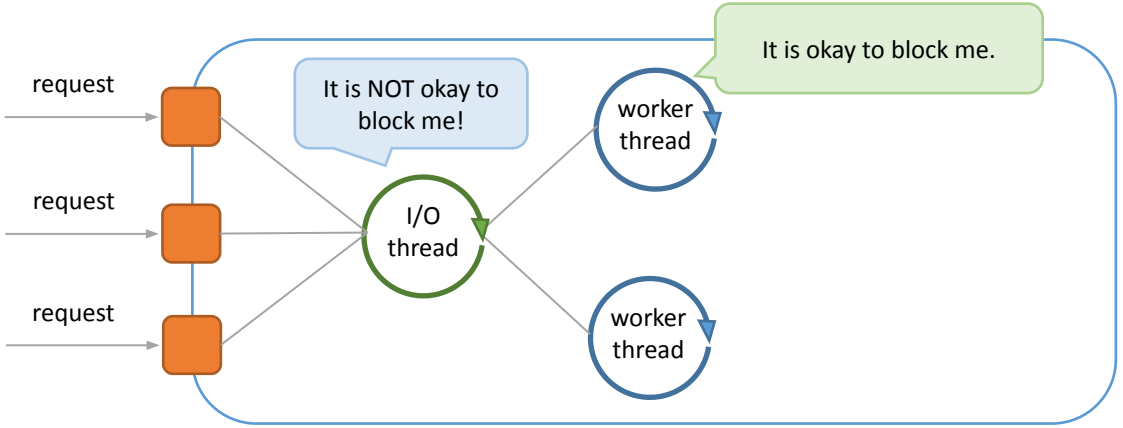
Blocking vs non-blocking I/O

thread per connection

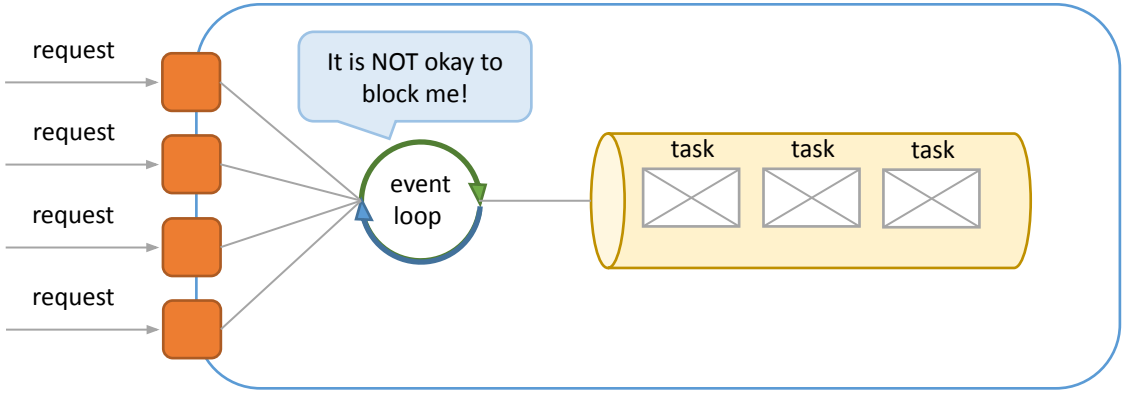


thread per request

with non-blocking I/O

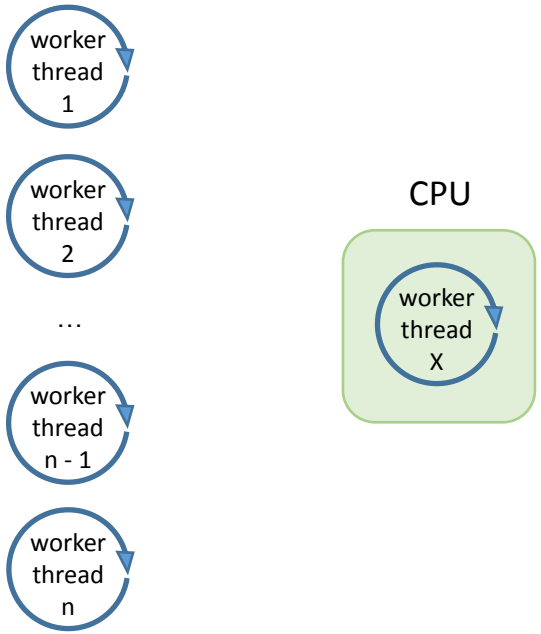


event loop

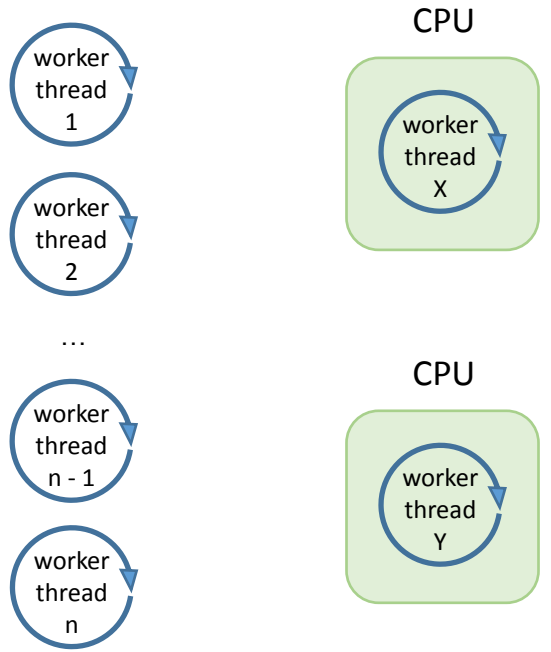


Blocking vs non-blocking I/O

concurrency



parallelism



the optimal number of event loops is
the number of CPUs

Blocking vs non-blocking I/O

outdated
still used in older versions of HTTP servers

thread per connection

Jetty, Nginx, Tomcat

thread per request
with non-blocking I/O

- Easier to implement, test and debug applications.

- Each thread consumes resources (memory, CPU).
  We have to limit the number of concurrent requests
  (load shedding and rate limiting).

good for
CPU-bound (compute-intensive) workloads

Netty, Node.js, Zuul

event loop

- Massive amount of active connections.
- More resilient to sudden traffic spikes.

- Increased development and operational complexity.

good for
I/O-bound workloads (e.g. large files)