

Advanced Networking and Distributed Systems

Module 2: Scalable Servers and Network Performance

GW CSCI 3907/6907

Timothy Wood and Lucas Chaufournier

Outline

Weeks 1-3: Network Programming and Protocols

- Writing simple network programs is easy!
- Providing reliable services over a network is hard!

Weeks 4-5: Scalability and Performance

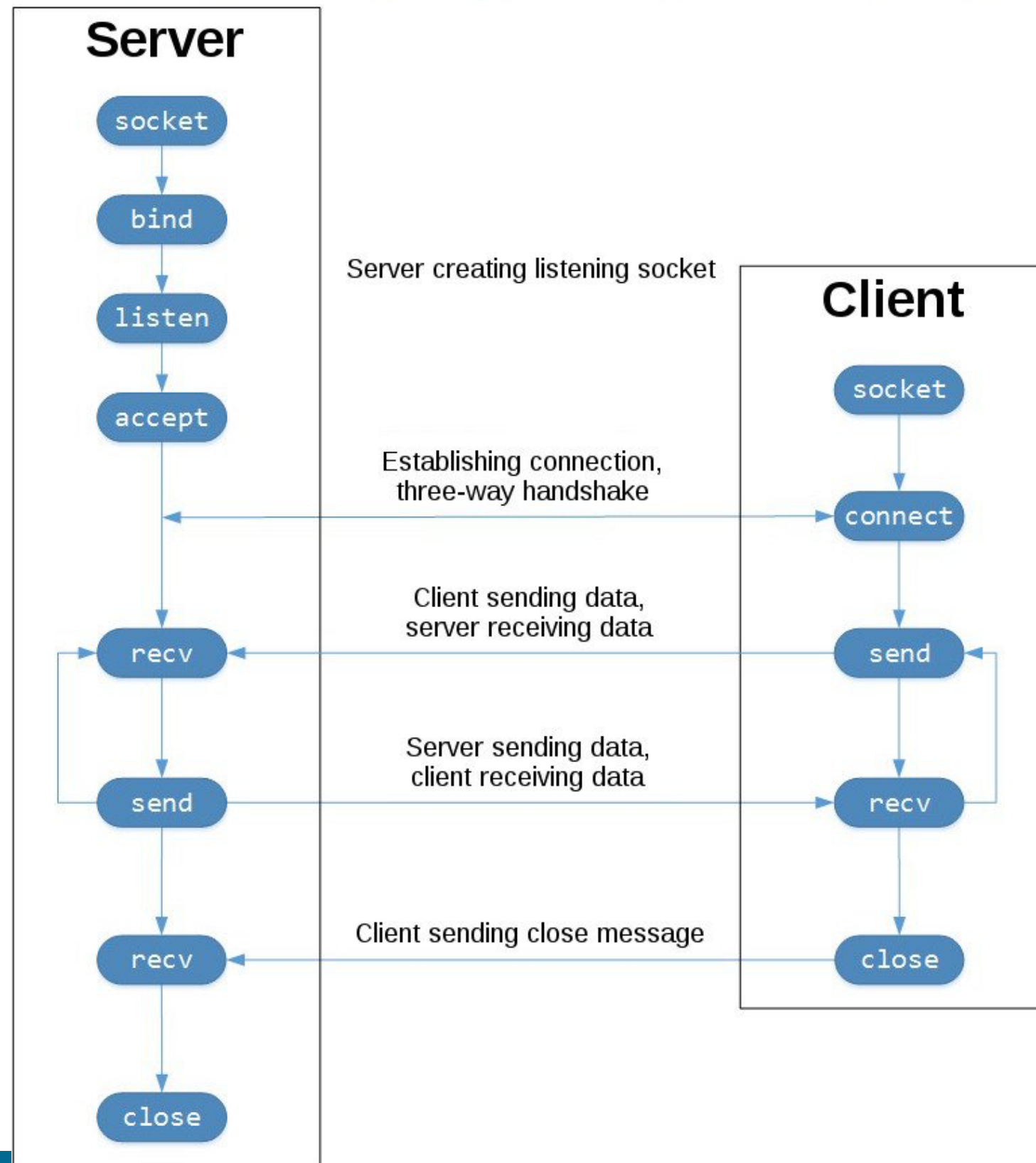
- How can we support many concurrent clients?
- What performance metrics matter for network services?

Weeks 6-7: Network Middleboxes

- How to deploy software *between* clients and servers?
- How to get the speed of HW and flexibility of SW?

Server Architecture

**How many clients
can this server
handle at once?**



Simplest Architecture

Server is a single thread

Network calls are blocking (**recv**, **accept**)

-> server can only handle one client at a time

What happens to other clients who try to connect?

Simplest Architecture

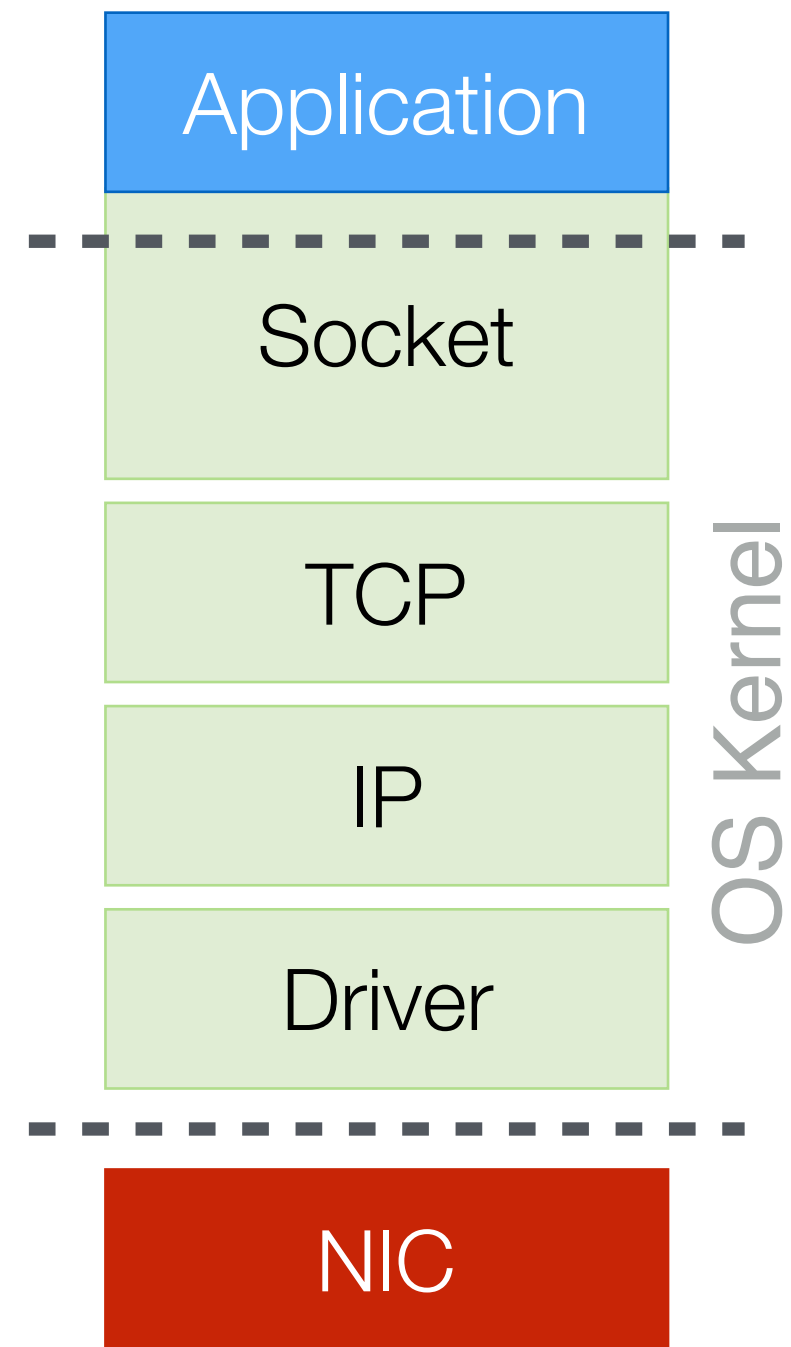
Server is a single thread

Network calls are blocking (**recv**, **accept**)

- Server can only handle one client at a time

What happens to other clients who try to connect?

- Incoming connections are buffered by the OS networking stack
- TCP Backlog parameter controls number of waiting connections
 - How do you think this works?



Threading!

Threading

Allows program to do multiple things at once

- Threads: execution context with its own stack and shared heap
- Processes: execution context with both stack and heap

How many threads or processes can we run?

Threading

Allows program to do multiple things at once

- Threads: execution context with its own stack and shared heap
- Processes: execution context with both stack and heap

How many threads or processes can we run?

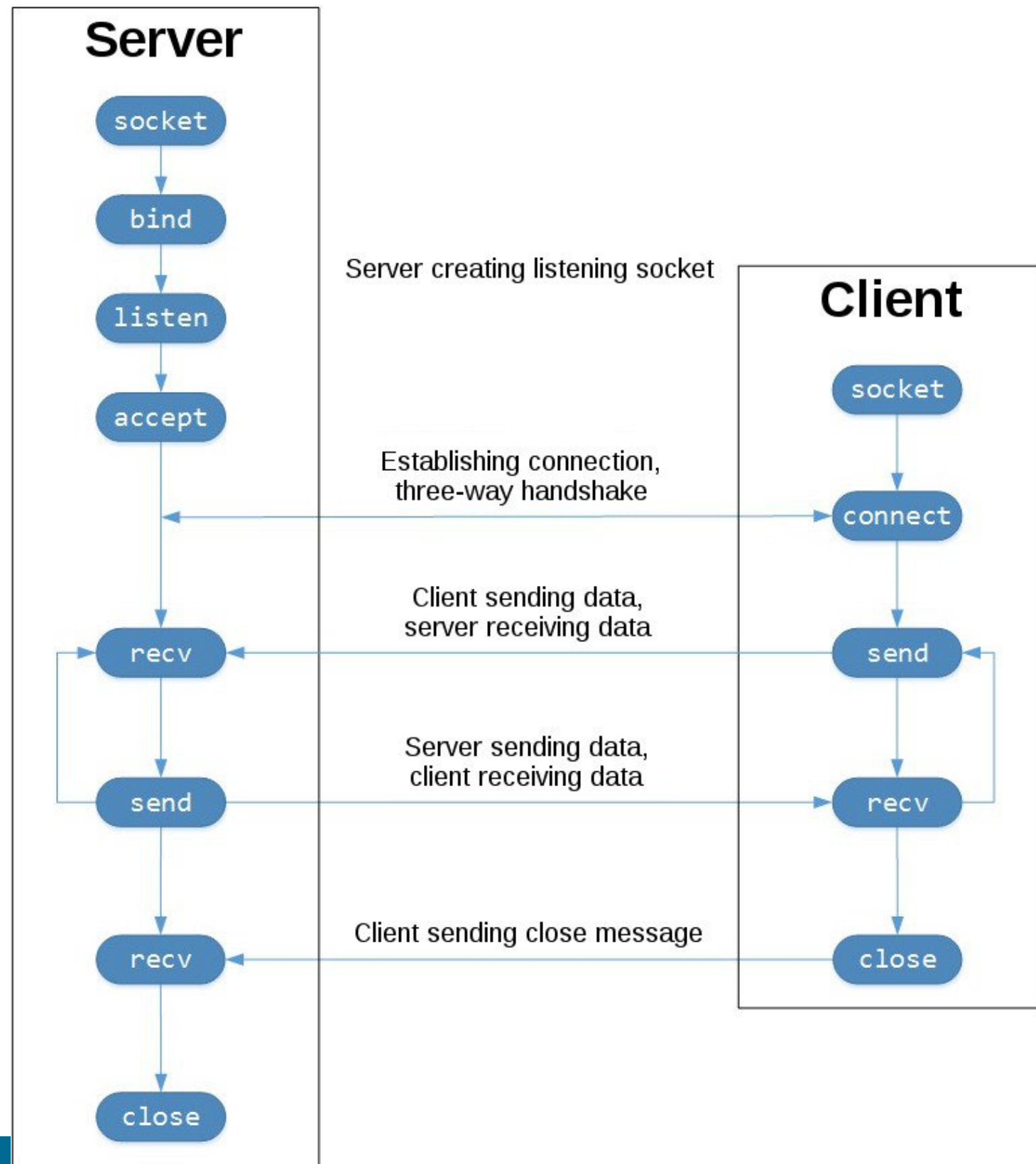
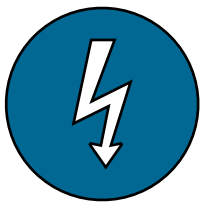
- Depends on available hardware and application type!

Concurrency is limited by...

- Number of CPU cores
- CPU vs IO intensiveness of application
- If CPU bound, then N cores can only run N threads at once
- If I/O bound, then may need \gg N threads to keep N cores busy

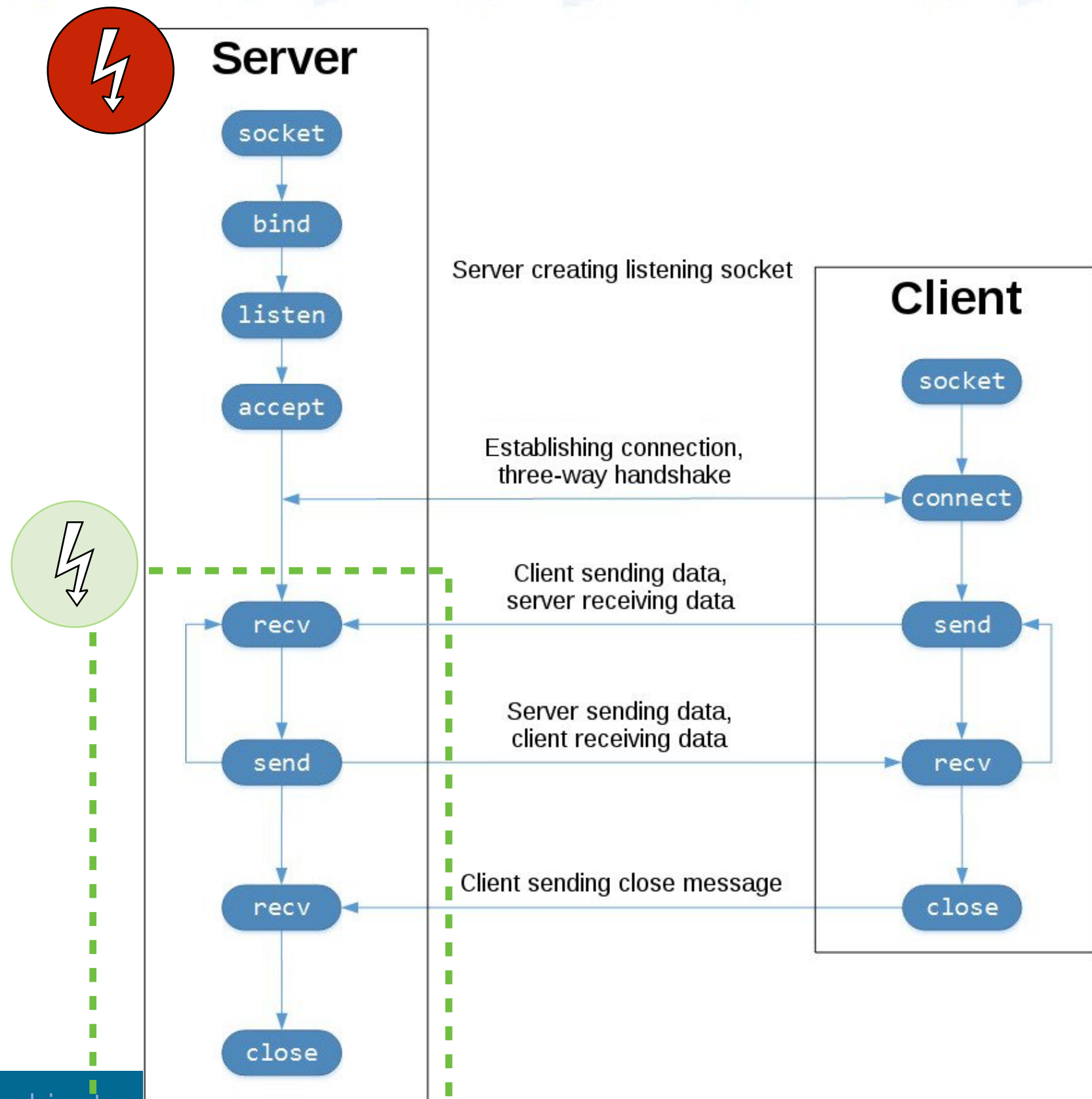
Thread Models

How can we use threads in our Server?



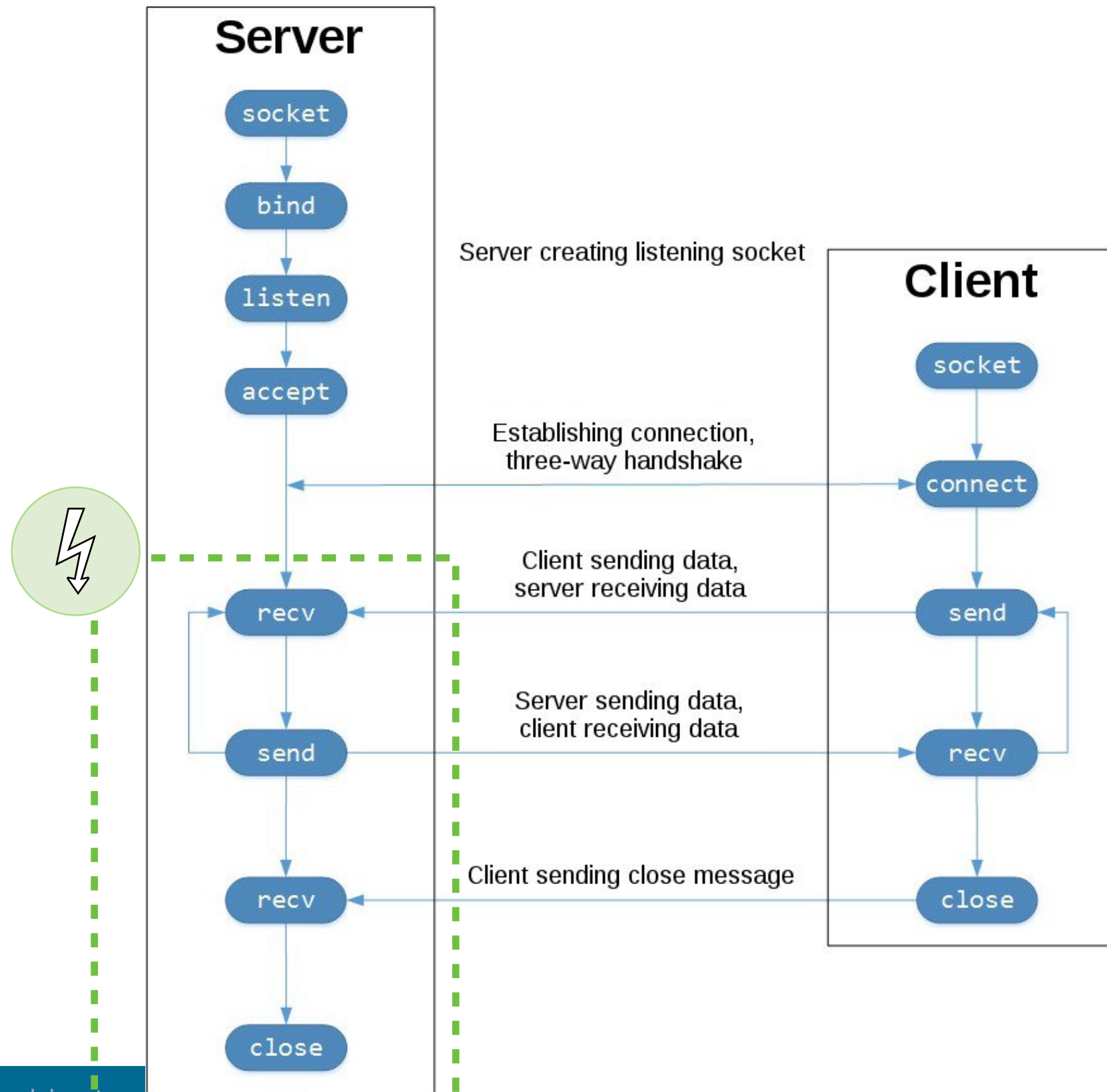
Thread Models

How can we use threads in our Server?



Thread Models

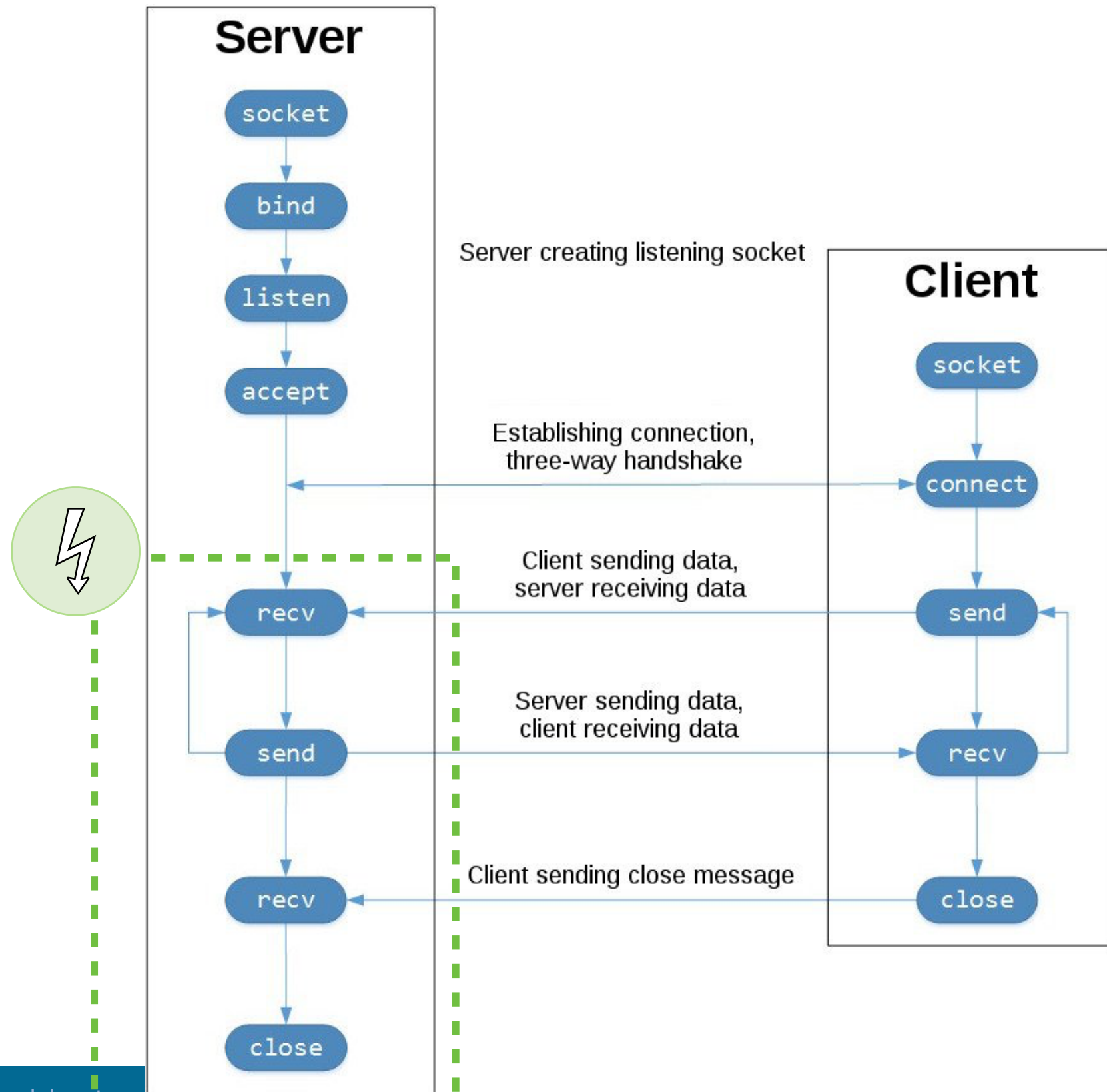
When to start threads?



Thread Models

When to start threads?

1. On every new request create a new thread
2. When program starts create a pool of threads



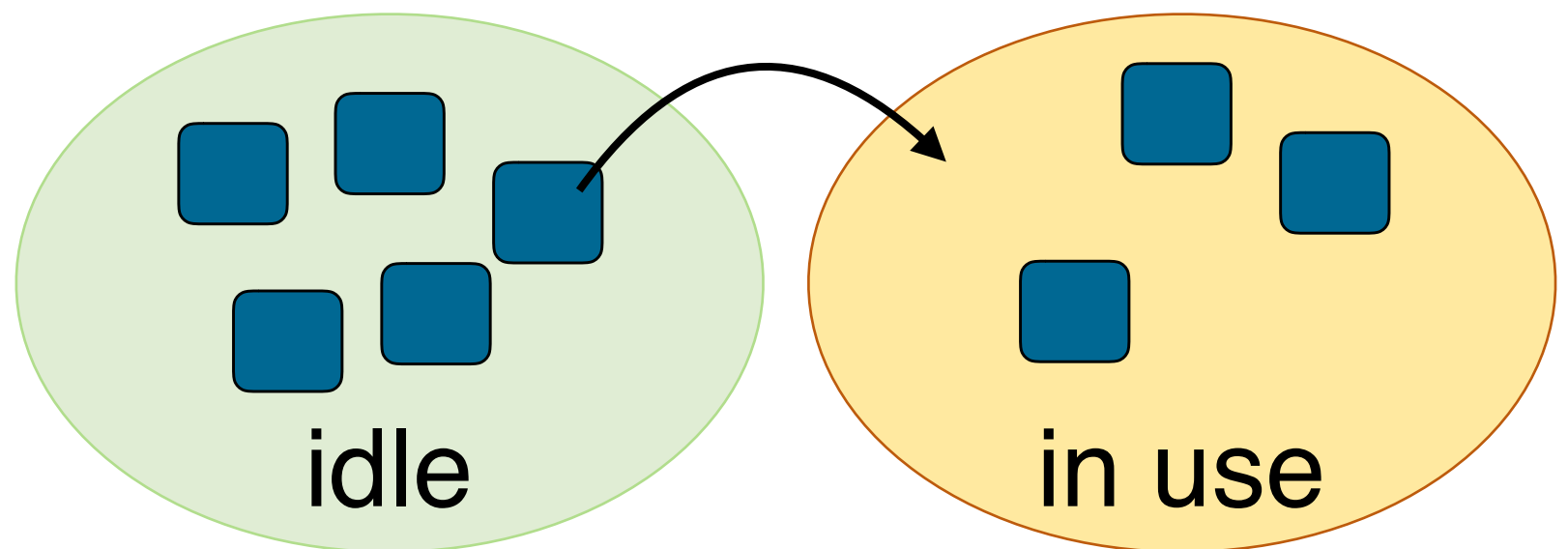
Object Pools

Common design pattern when you need to create and destroy lots of something

Create = malloc

Destroy = free

- Both of these may involve slow system calls
- Even worse if the thing you are creating is a thread!



Object pool just changes an object's state from **idle** to **in use** or back again

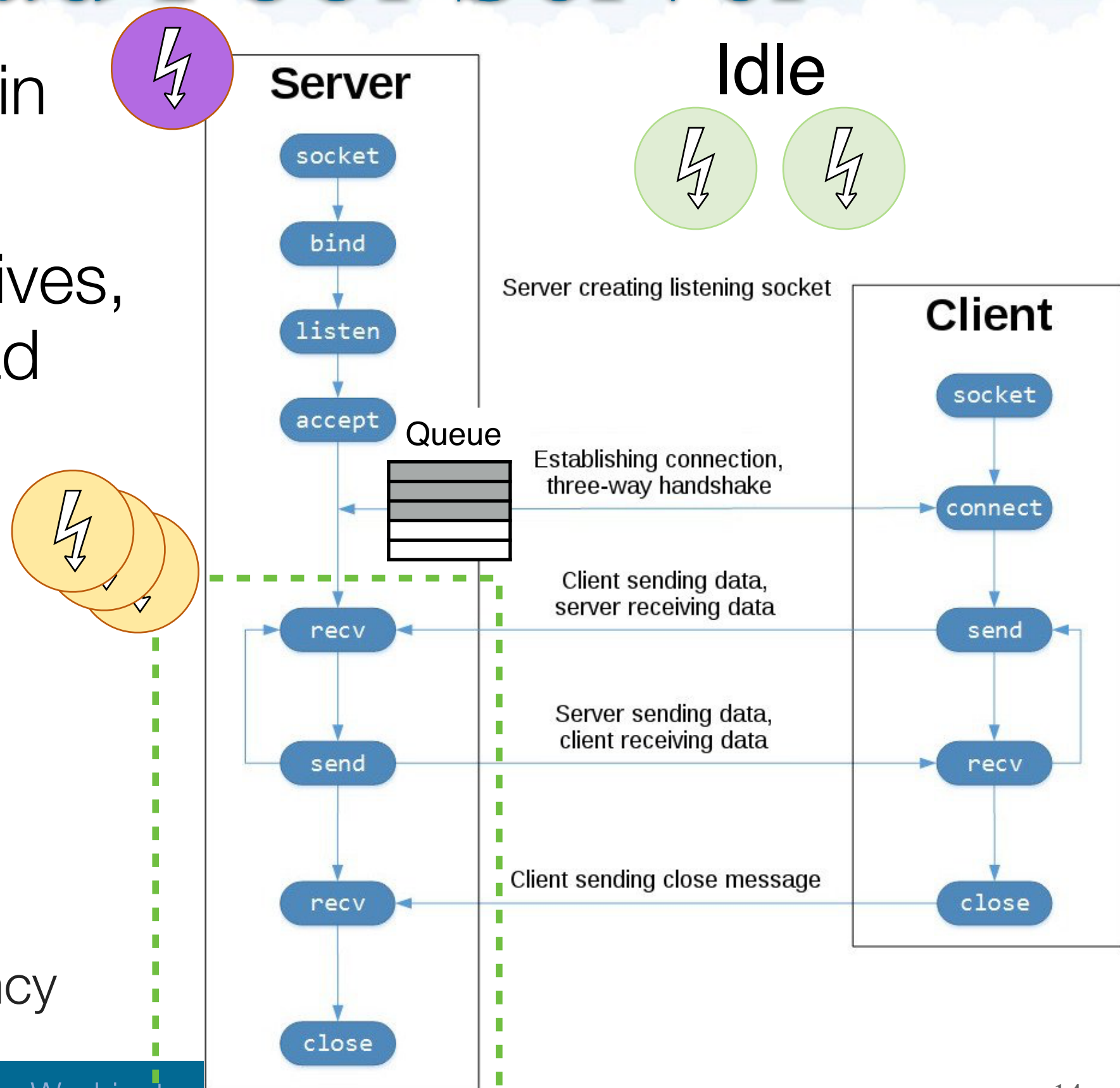
Thread Pool Server

Idle threads wait in pool

When a client arrives, alert an idle thread

How?

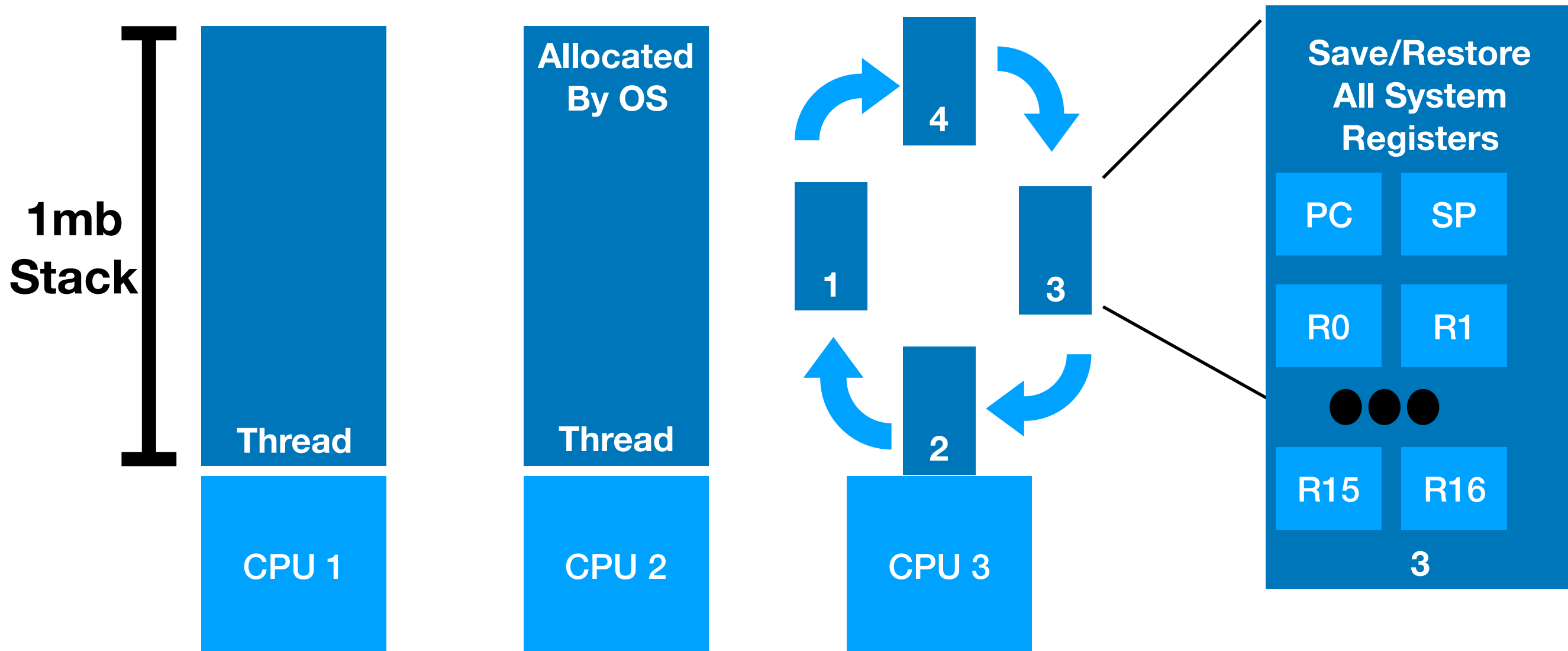
- Put new client into a queue
- Wake idle thread using condition variable
- Remove client from queue using locks for consistency



Lightweight Threads

All about Go routines!

A threads primer

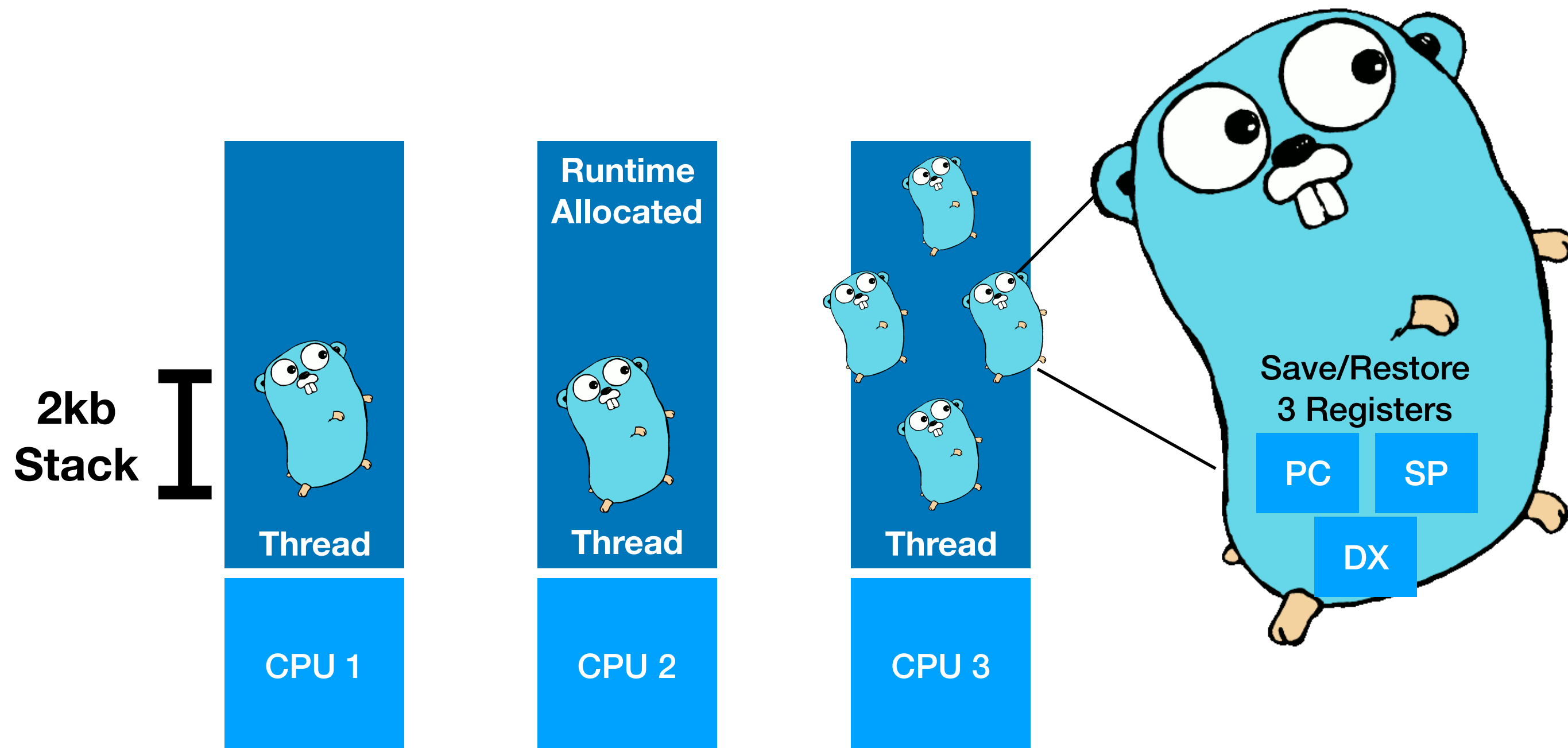


Go Routines

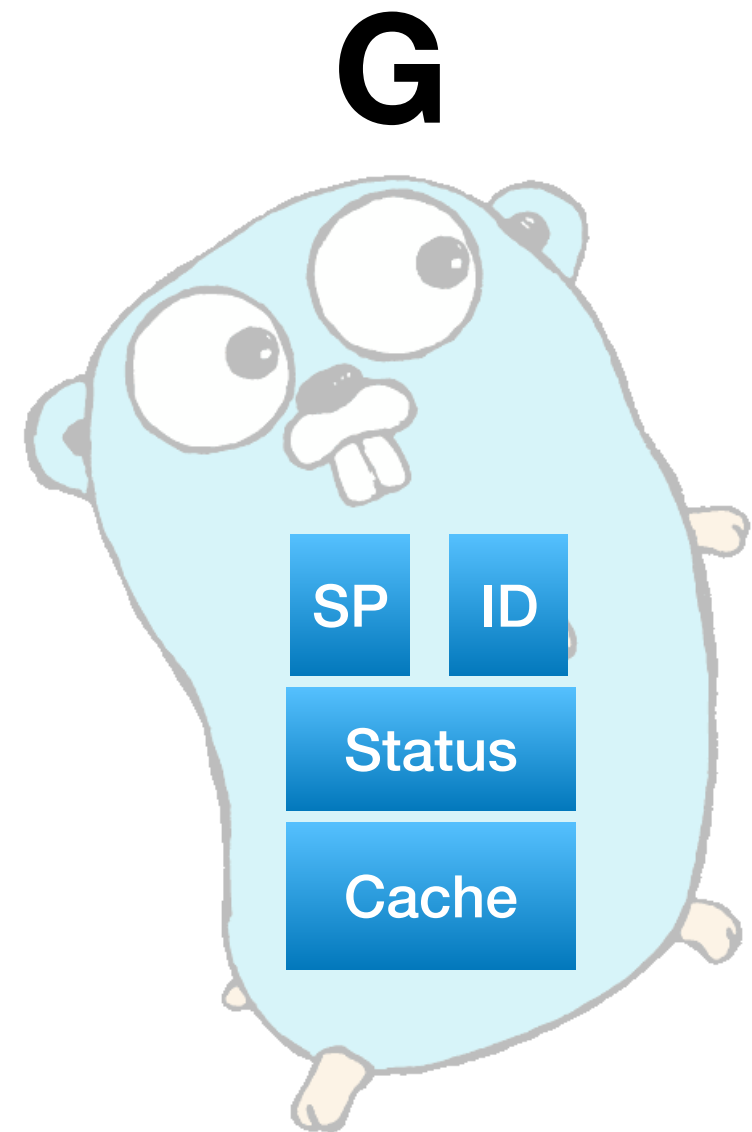
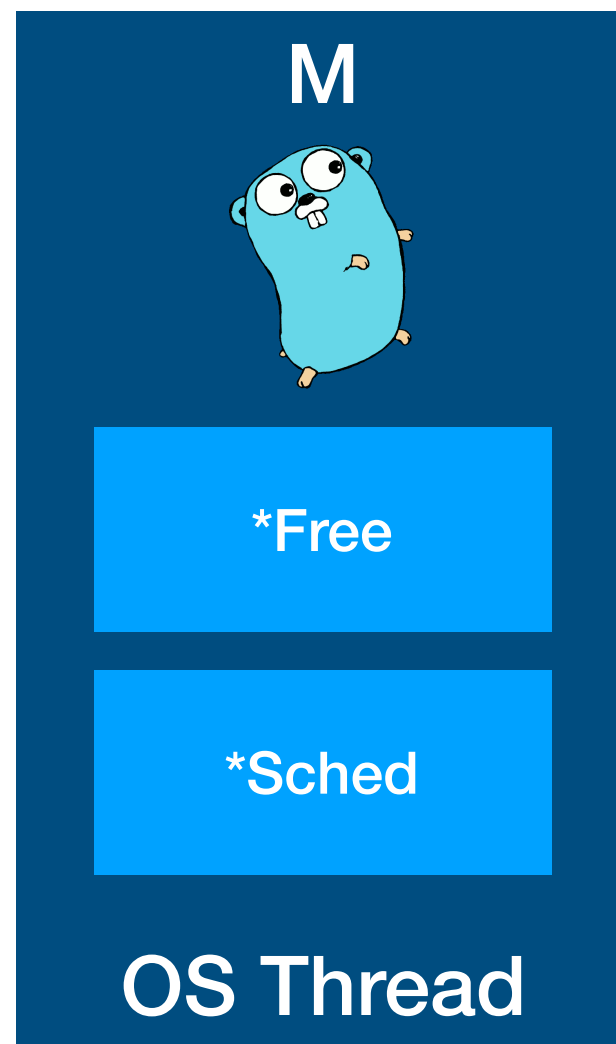
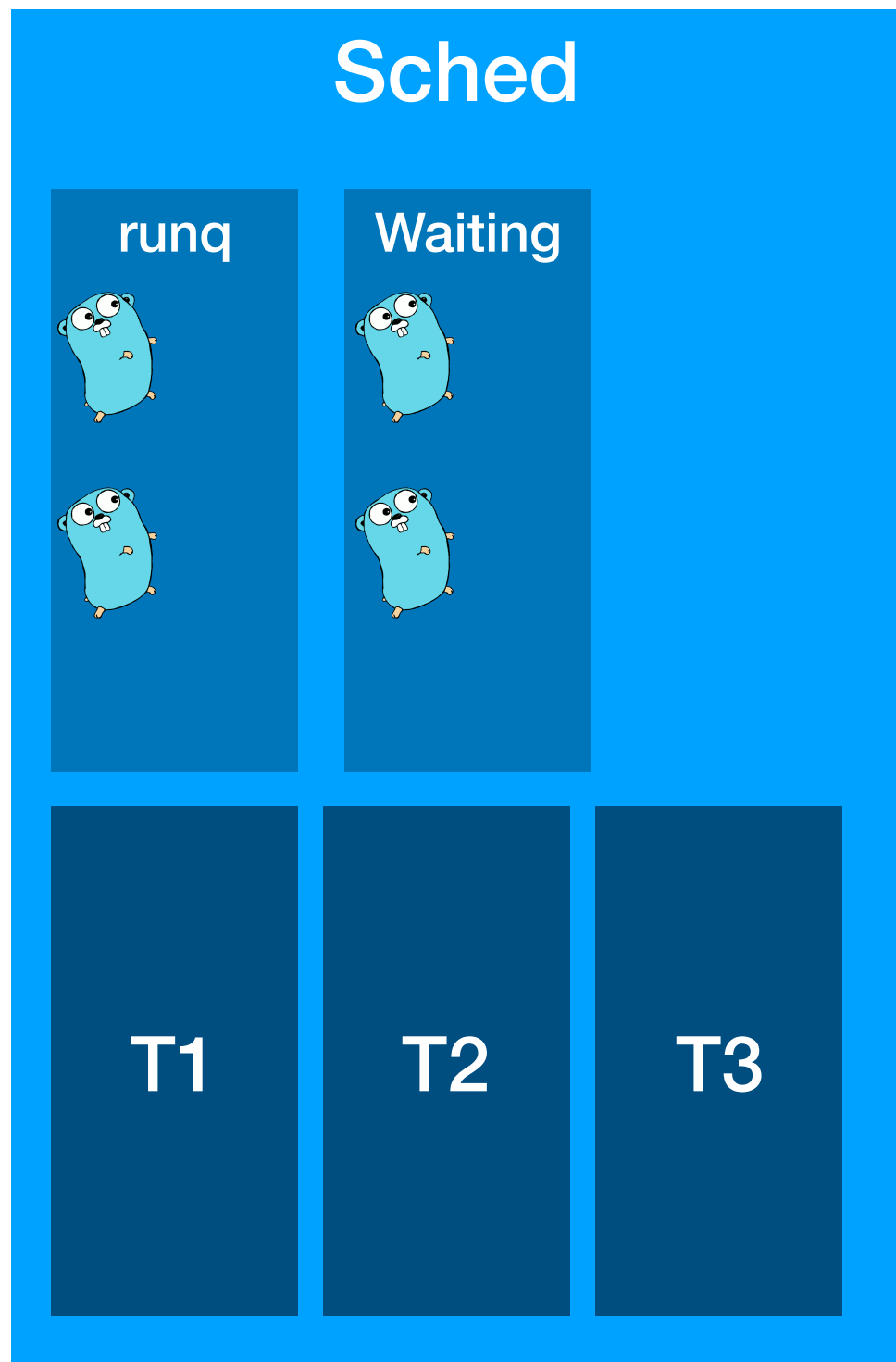
- Golang technique for concurrent programming.
- An abstraction on threading.
- Very lightweight and cheap!
- Allow programs to scale with ease

```
func helloWorld(){  
    fmt.Println("Hello World!")  
}  
  
func main(){  
    go helloWorld()  
  
    go func(txt string){  
        fmt.Println(txt)  
    }("Hello World")  
}
```

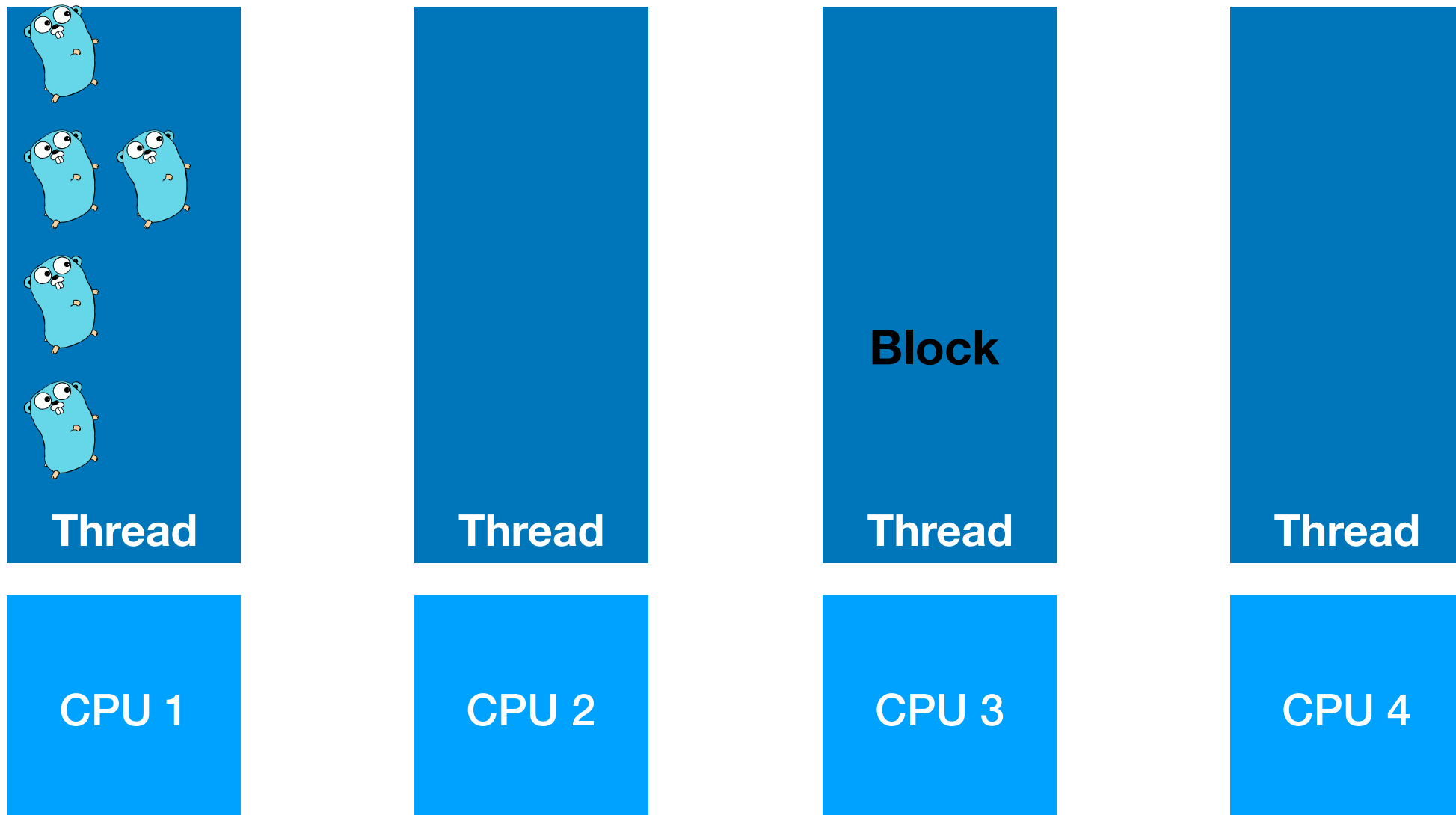
Go Routines



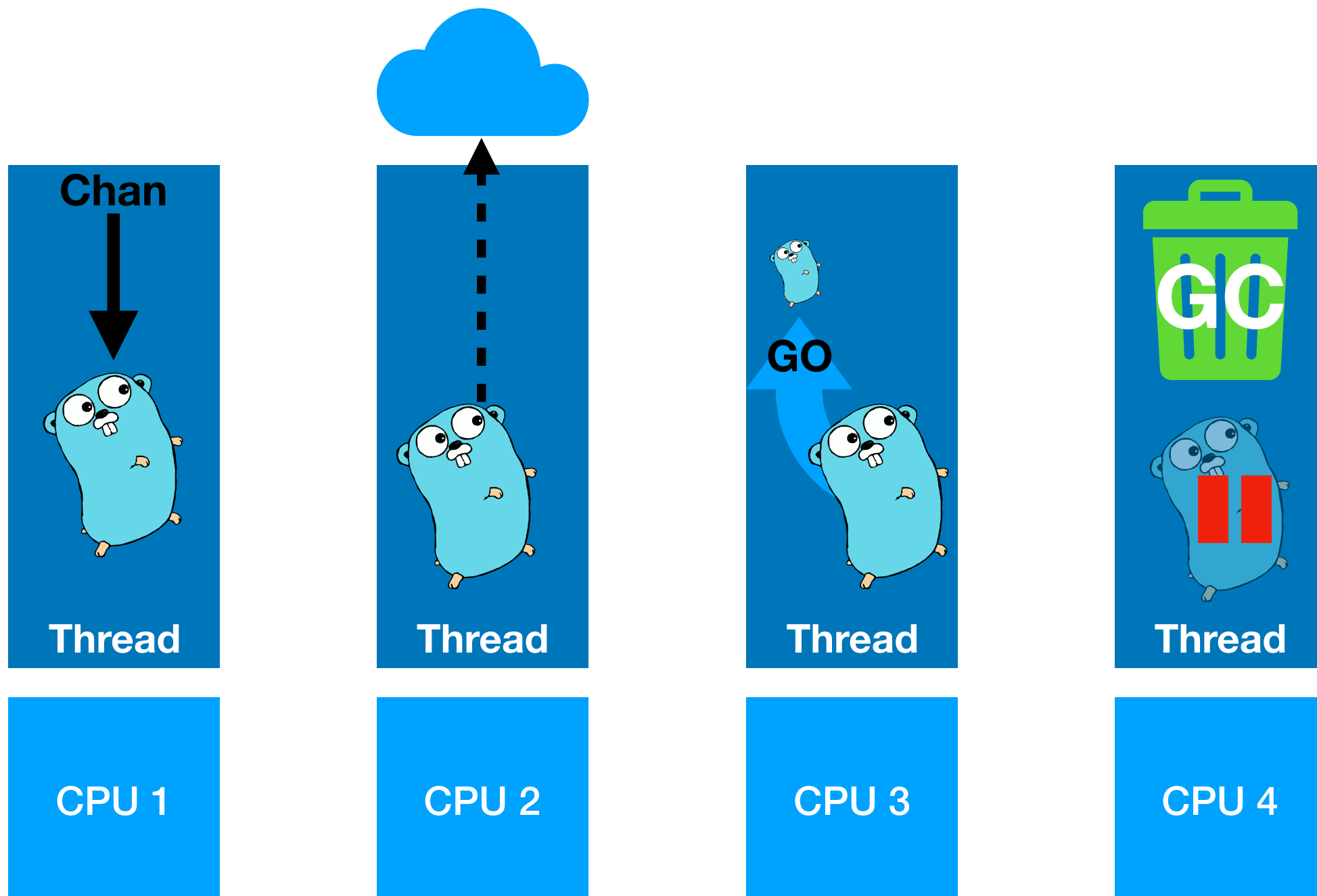
Under the Hood



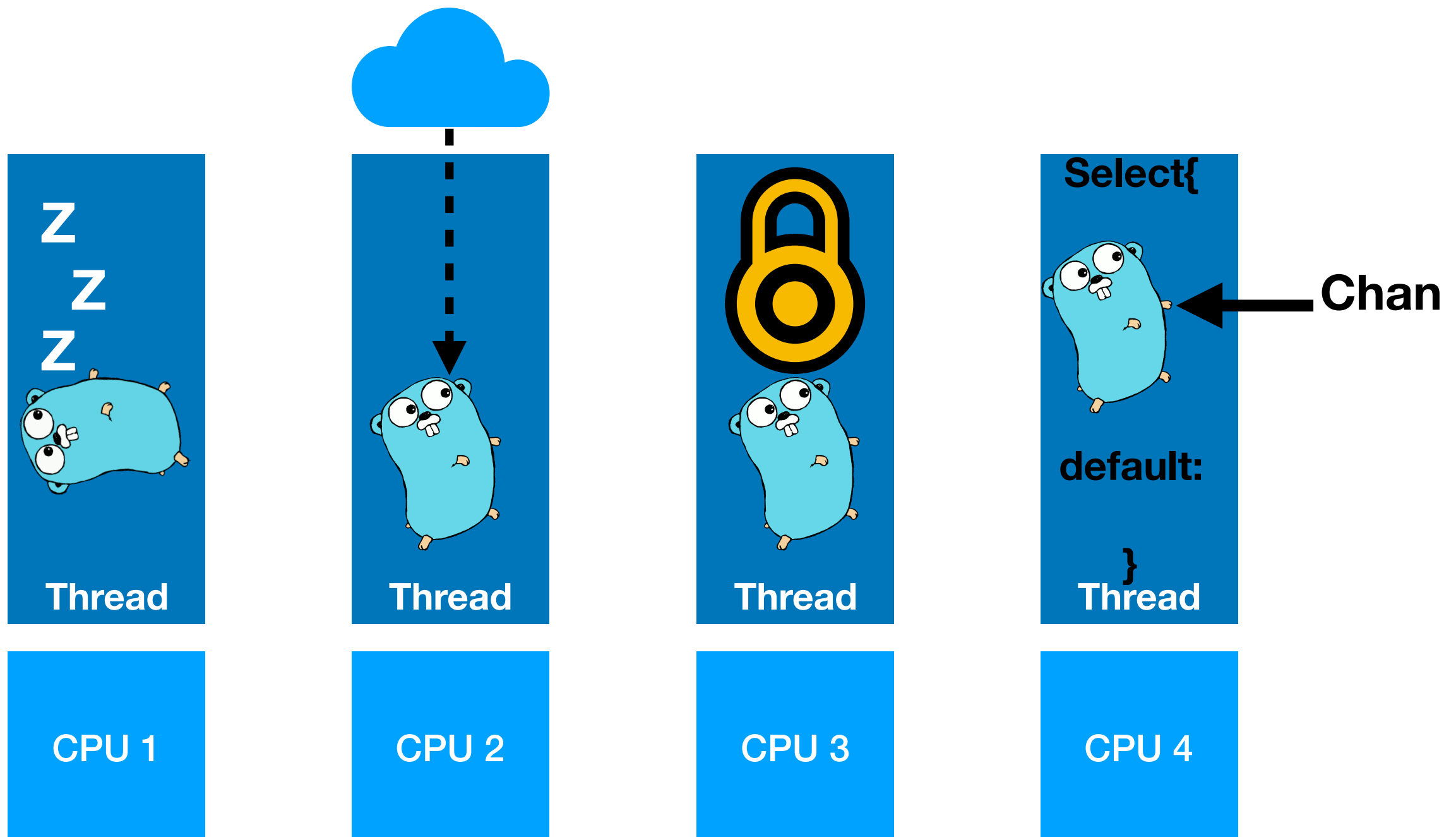
Go Routines



Blocking Go Routines



Non-Blocking Go Routines



More threads?

Is more threads always the answer?

Threads add context switch costs and consume system resources... is there another way?

Non-Blocking IO

Why wait?

Blocking Calls

We needed multiple threads because recv blocks

But is it really necessary to wait on recv?

- You already saw in RUDP project that we don't need to wait forever; we can just wait for a short time and then return

Blocking / Synchronous IO:

- Go to sleep if no data, get woken up when it arrives

Non-Blocking / Asynchronous IO:

- Check if there is data, do something else if no data, check again

Simple Non-Blocking

Sockets can be set to non-blocking mode

```
import socket
# Create a TCP/IP socket in non-blocking mode
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setblocking(0)
```

Then **recv** calls will not wait for data, just return error

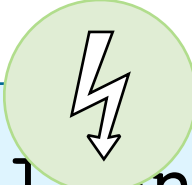
```
while True:
    try:
        data = conn.recv(1024)
    except socket.error:
        print("No data yet")
```

Drawbacks of this approach?

Non-Blocking Server

What happens if we have many clients?

Client 1 
Client 2 
Client 3 
...
Client n 



```
# Accept all clients...  
  
for client in clients:  
    try:  
        data = client.recv(1024)  
        process(data)  
    except socket.error:  
        print("No data yet")
```

Code is messy and inefficient if many clients!

Non-Blocking IO

We need a better way to know what data is ready!

select event polling

- Register a set of IO “file descriptors” you care about
- Sleeps until at least one of them has data -> won't block!

```
int select(int nfd, fd_set *readfds, fd_set *writefds,  
          fd_set *errorfds, struct timeval *timeout);
```

- Assumes a Unix environment where files, sockets, and other types of IO are all mapped to a file interface

Select Example

```
import selectors
import socket
```

```
def accept(sock, mask):
    conn, addr = sock.accept()
    conn.setblocking(False)
    sel.register(conn,
                  selectors.EVENT_READ, read)
```

```
def read(conn, mask):
    data = conn.recv(1000)
    if data:
        conn.send(data)
    else:
        sel.unregister(conn)
        conn.close()
```

```
sel = selectors.DefaultSelector()
sock = socket.socket()
sock.bind(('localhost', 1234))
sock.listen(100)
sock.setblocking(False)
sel.register(sock, selectors.EVENT_READ, accept)

while True:
    events = sel.select()
    for key, mask in events:
        callback = key.data
        callback(key.fileobj, mask)
```

Non-blocking Variants

Languages, runtimes, and OS's typically have several ways to do non-blocking IO

select: system call for checking if things are ready

epoll / kqueue: app/OS interface for checking if things are ready (much more efficient than original select)

But now select can be viewed as an API, and might be implemented with something like epoll.

Event-Based Programming

Registering call backs for events can be a simpler programming model

- Simpler to write... maybe harder to debug!

Adds a layer of abstraction

- Event notification layer checks for events and decides what order to process them in. Why is this helpful/interesting?
- Could use multiple threads to process the events!

node.js

Web framework for javascript-based apps

Probably the most popular event based platform

Single threaded event based server!

- Faster and less resource intensive than many multi-threaded servers!

Other event based frameworks/languages:

- Erlang, Elixir, ...

Assignment 2

Technical Writing

Being able to present ideas is just as important as being able to write code!

[] Write a blog post on a networking topic

- Must be long enough to be interesting
- You must write some code or run experiments
- Present useful information in an understandable way
- Present useful information in a visually appealing way

Ideas

Performance comparison of...

- Node.js vs Apache vs nginx vs ...
- Thread pool vs new thread per request in language X
- http vs https vs http2

Tutorial on...

- how to use wireshark to analyze HTTP traces or solve a puzzle
- how to gather statistics of public wifi traffic (ethically)
- how to use go co-routines and how they work under the hood
- queueing theory 101 with example measurements
- how to use epoll / select / etc in language X
- everything that happens when you open a page in a browser
- python 2 vs python 3 networking code
- how to generate traffic to benchmark a web server

Inspiration

Julia Evans' blog and zines

- <https://jvns.ca/>

Performance

This week with 100% more math!

What does it mean to be fast?

Reminders

Assignment 2: Tech Blog due 2/20

Participation is very important!

- Ask and answer questions!
- I almost never want people to answer with the “right” answer
- I want answers that help us discuss the question
- Wrong answers or partial answers are much more useful!

Metrics

What metrics matter for...

- Amazon's store front
- Netflix video streaming
- Bank of America's savings account site
- High Frequency Traders
- My course website

Metrics

What metrics matter for...

- Amazon's store front
- Netflix video streaming
- Bank of America's savings account site
- High Frequency Traders
- My course website

Key metrics:

- Throughput: requests per second or bits per second
- Latency: time to process a request
- Availability: % of time service is available
- Cost: money matters
- more?

Throughput and Latency

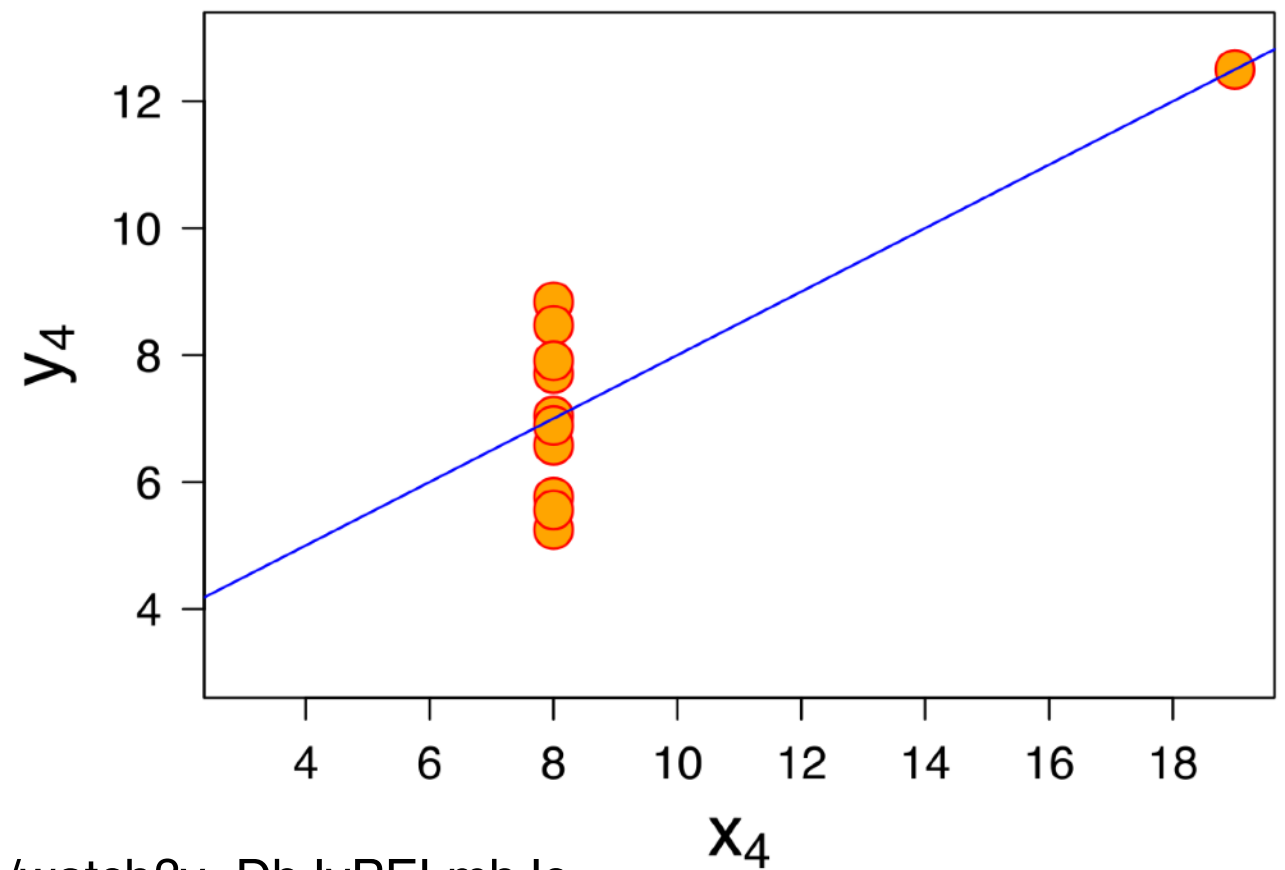
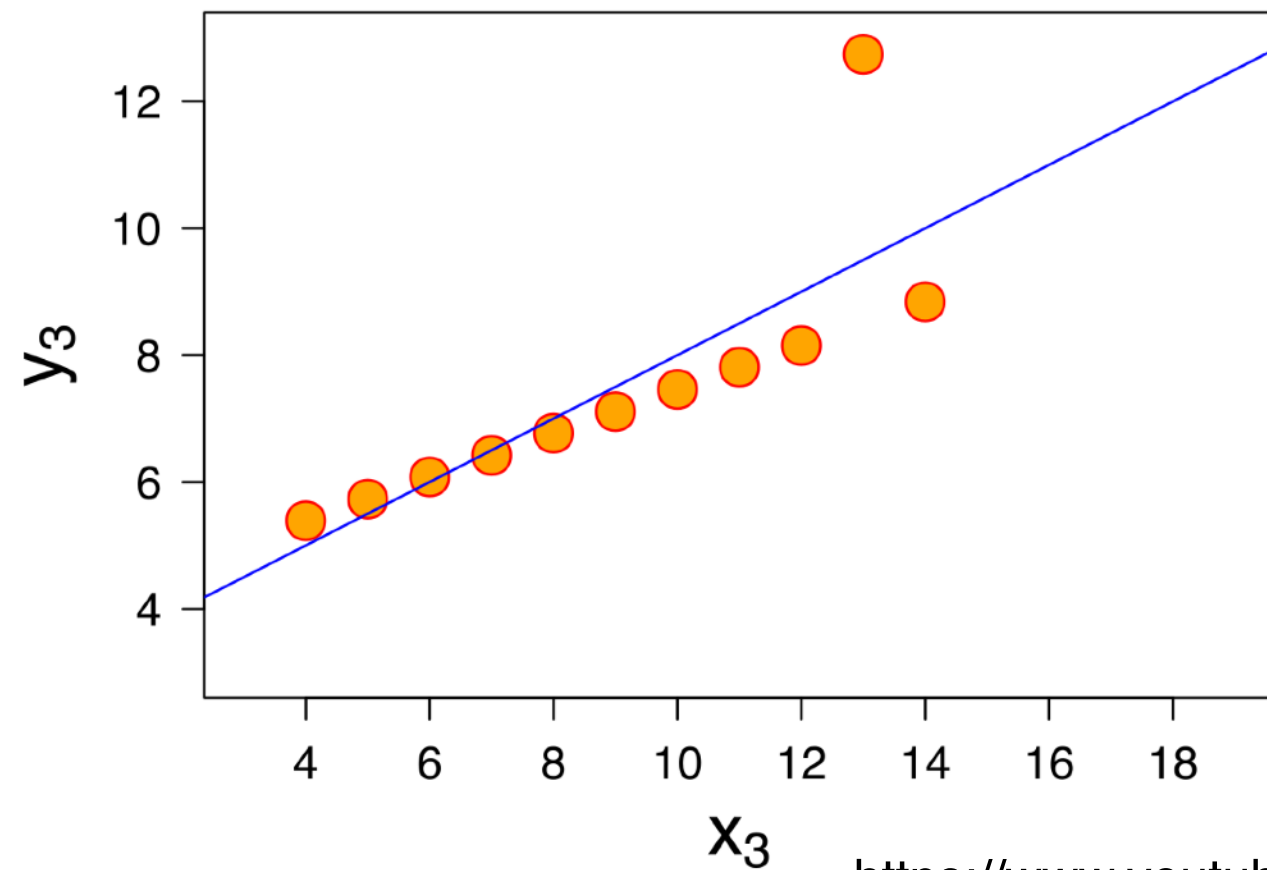
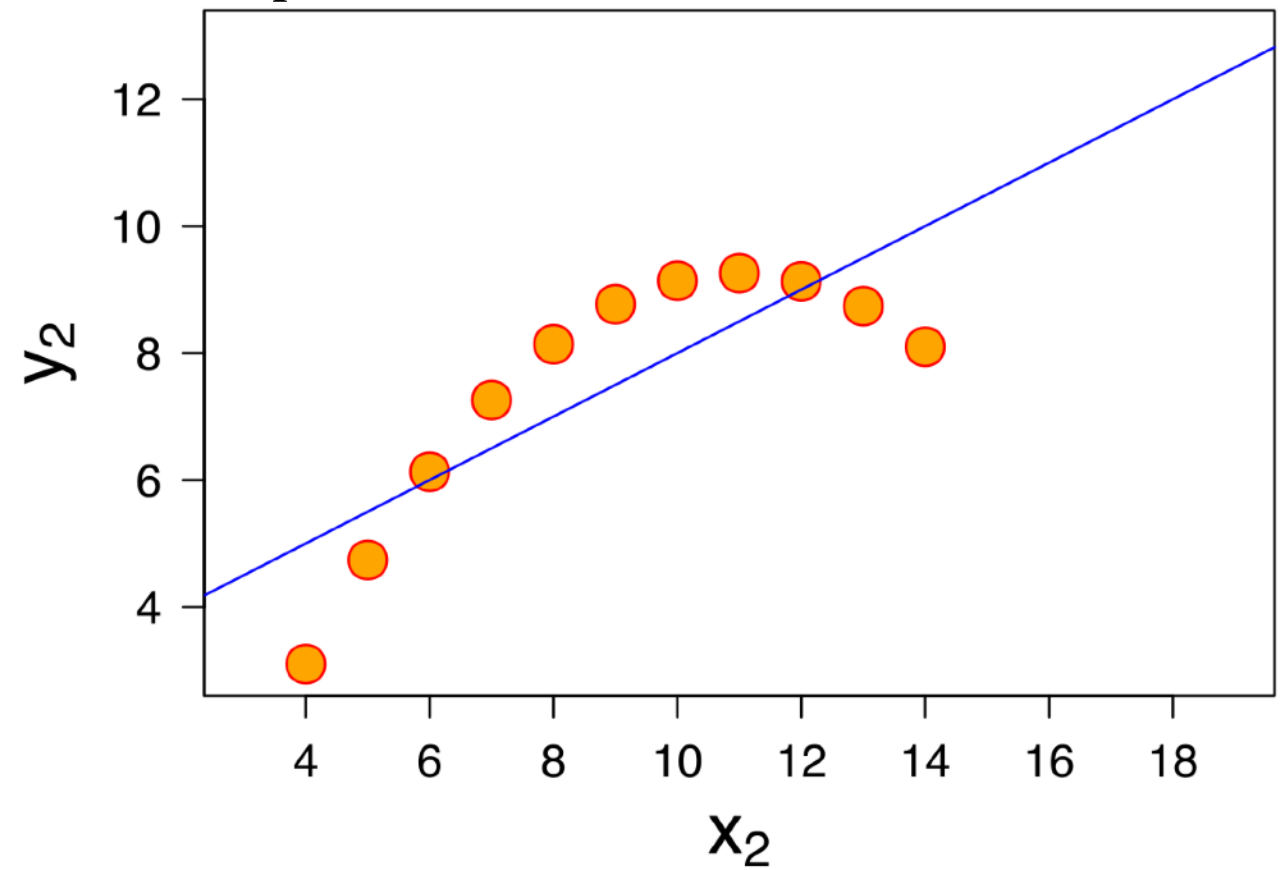
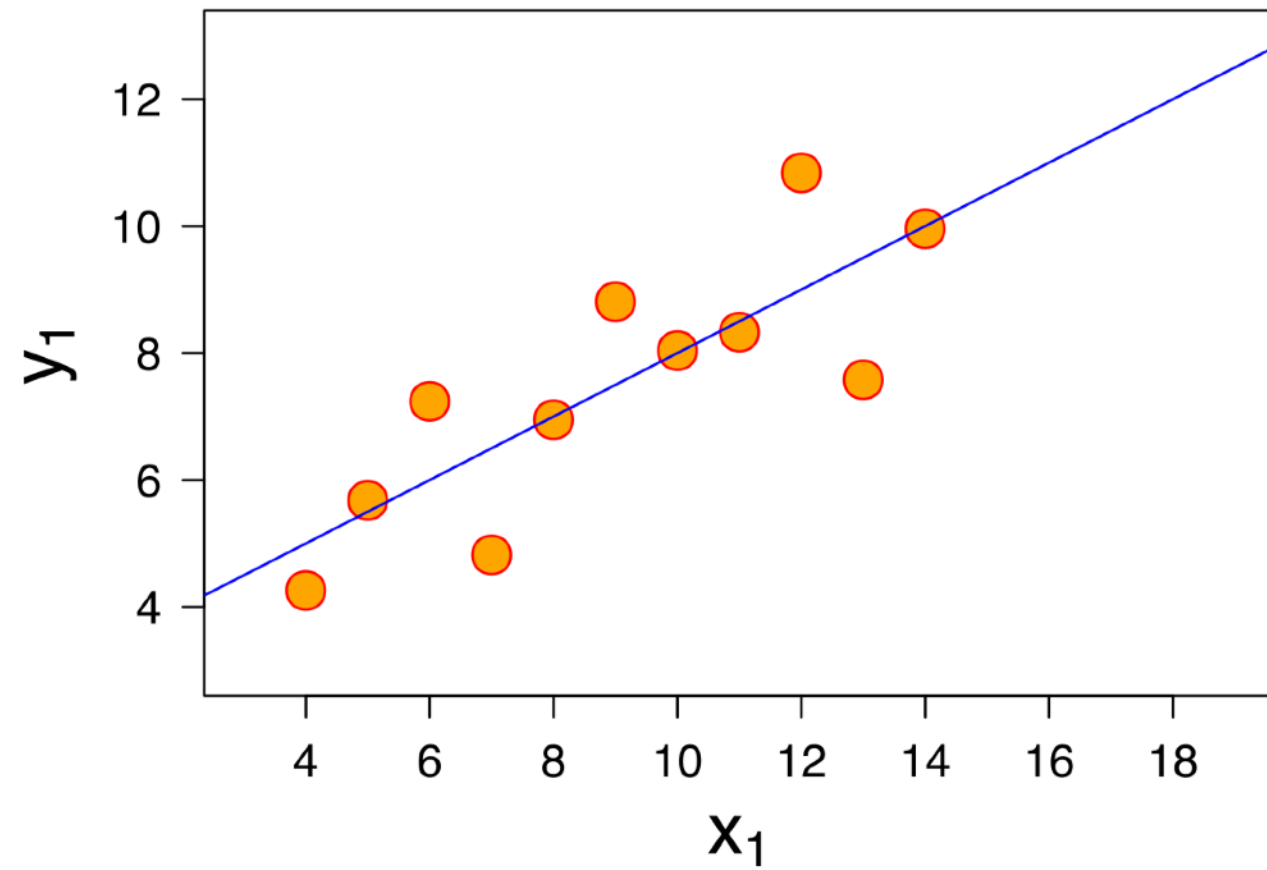
Throughput: units of work completed per time unit

Latency: time from issuing request to getting response

Need more than just the average!

- Min, Avg, Max
- Standard deviation
- More?

Anscombe's quartet from wikipedia



Throughput and Latency

Throughput: units of work completed per time unit

Latency: time from issuing request to getting response

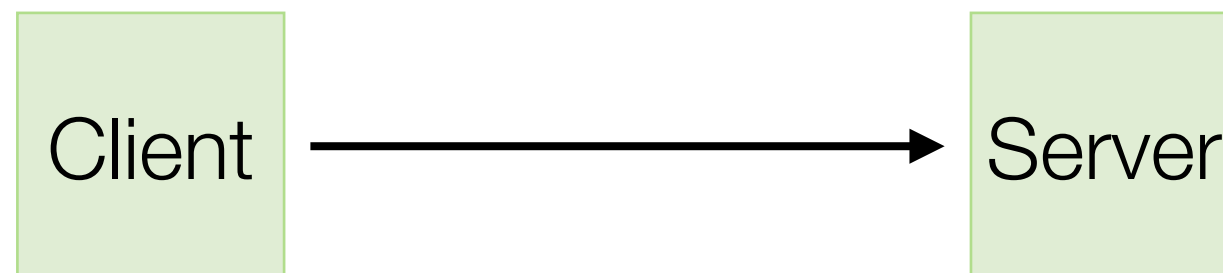
Need more than just the average!

Distributions are important

- Histograms (or PDF)
- Cumulative Distribution Function (CDF)

Throughput and Latency

What affects throughput and latency?



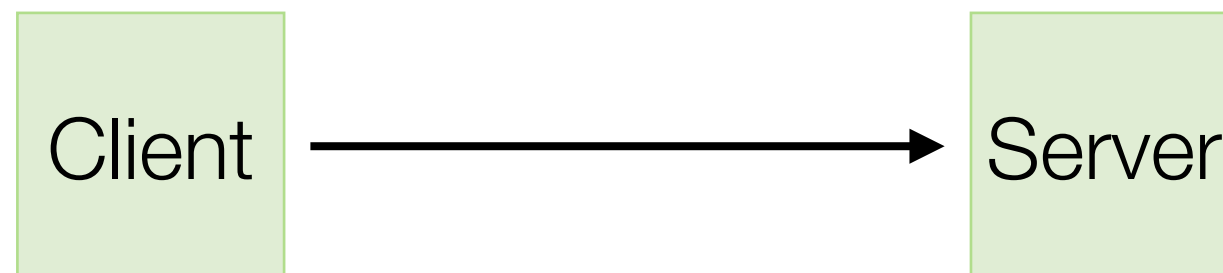
Throughput and Latency

What affects throughput?

- Bandwidth of network
- Processor speed on server
- # of processors on server

What affects latency?

- Network distance
- Processor speed on server
- Load on network/server (queueing delays, retransmissions)



Throughput

Can we predict the max capacity of a web server?

- What info do we need?

Throughput

Can we predict the max capacity of a web server?

- What info do we need?

Service Time = time to process a single request

- with no other load on the system

$$\text{max capacity} = \frac{1}{\text{service time}}$$

(for a single processor system)

How does load affect Latency?



Latency

What contributes to latency?

How can we quantify this?

Latency

What contributes to latency?

How can we quantify this?

$$L = \text{RTT} + \text{queueing delay} + \text{service time}$$

What affects queueing delay?

Latency

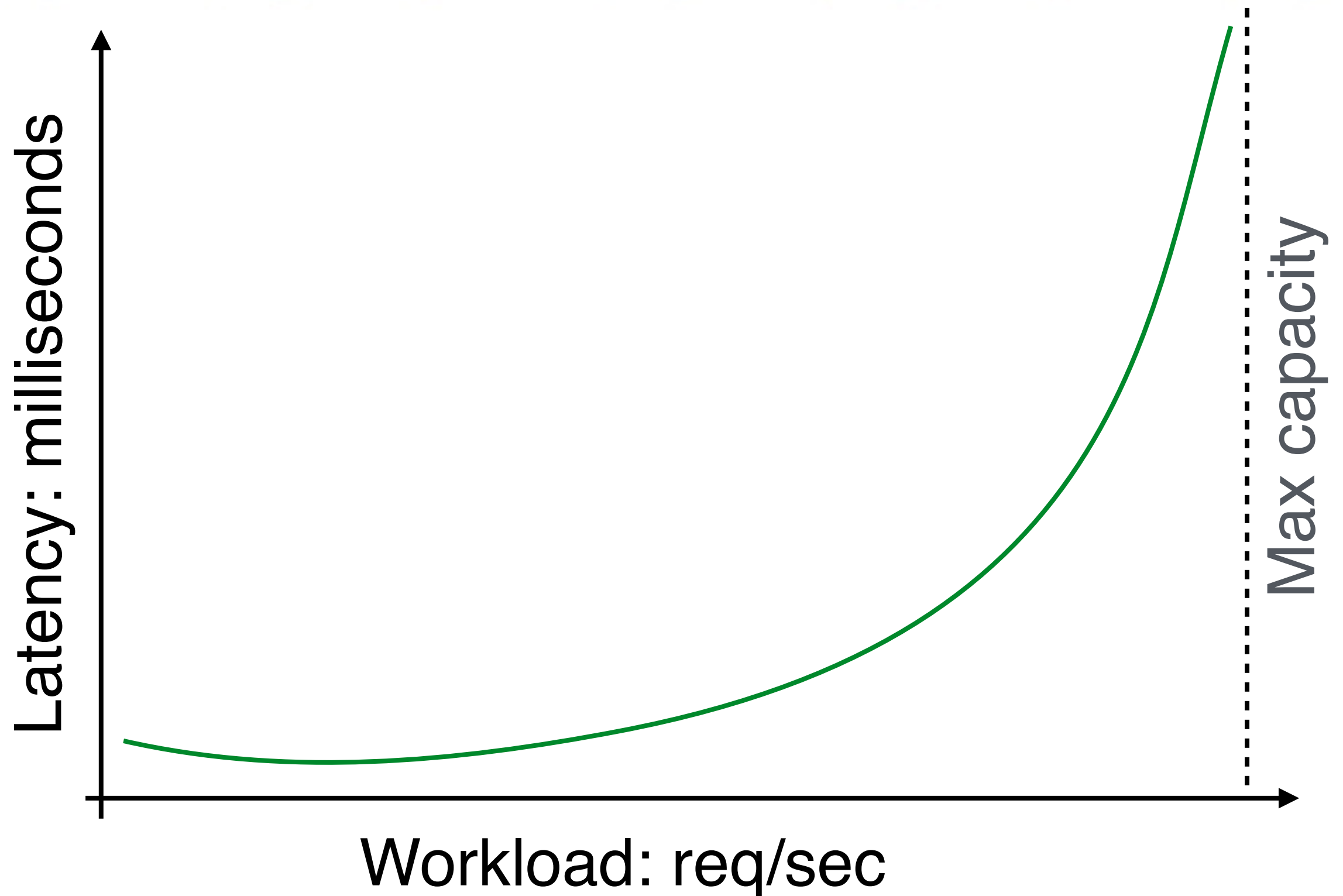
What contributes to latency?

How can we quantify this?

$$L = \text{RTT} + \text{queueing delay} + \text{service time}$$

$$\begin{aligned} \text{queueing delay} &= \frac{1}{(\text{capacity} - \text{load})} \\ \text{max capacity and load in req/sec} \\ \text{capacity must be } > \text{load} \end{aligned}$$
$$\frac{1}{(100 \text{ r/s} - 10 \text{ r/s})} = 0.01 \text{ s}$$
$$\frac{1}{(100 \text{ r/s} - 99 \text{ r/s})} = 1 \text{ s}$$

How does load affect Latency?



Latency Distribution

We've been looking at **average** latency

Why would a business/developer care about other statistics?

Latency Distribution

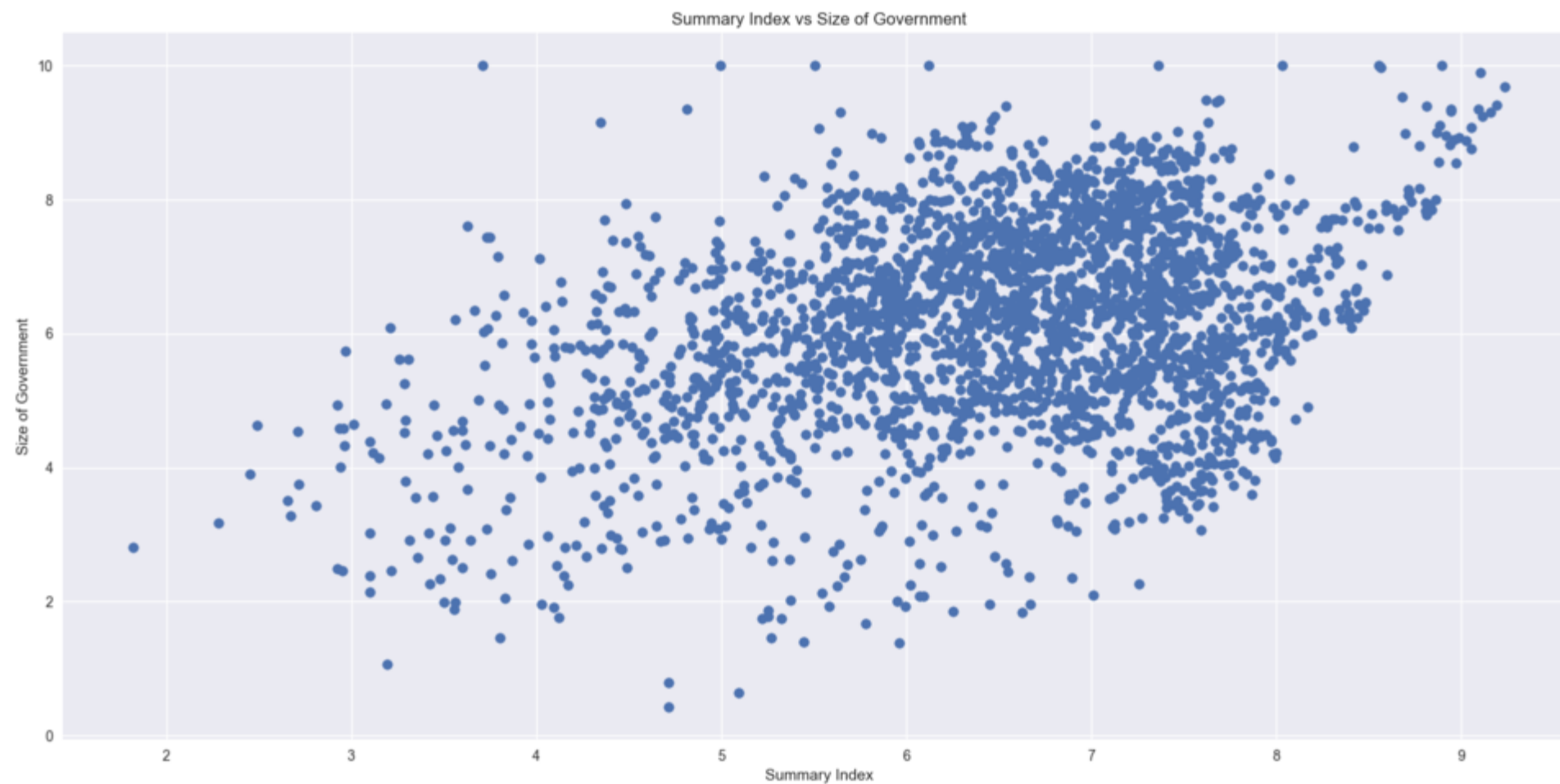
We've been looking at **average** latency

Why would a business/developer care about other statistics?

- Quality of Service (QoS) guarantees might be based on a percentile like “90% of users have latency < 100 msec”
- Worst case response time can be used to guide timeouts

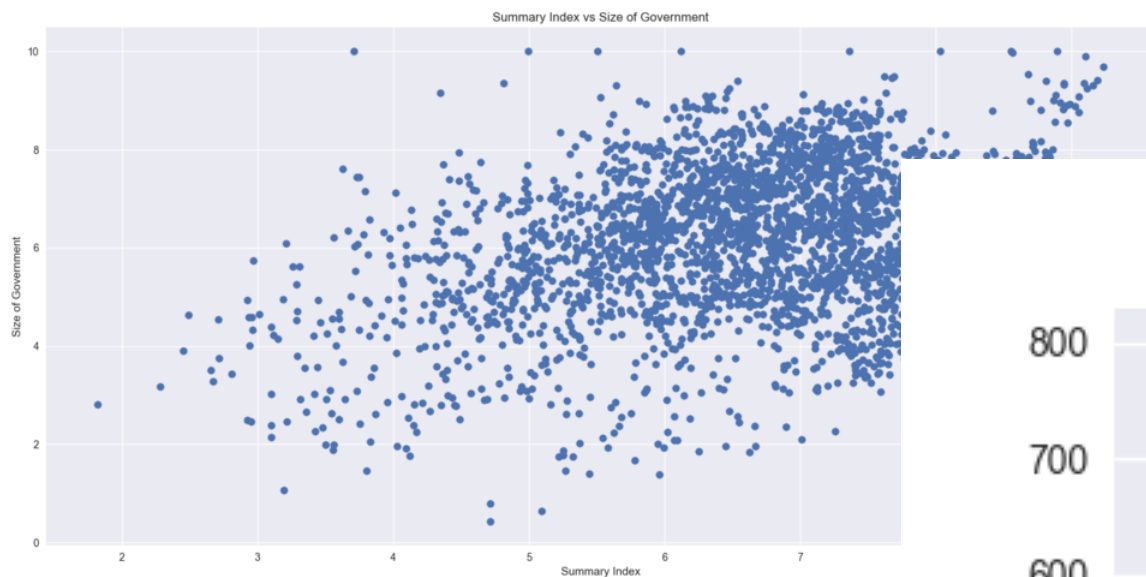
Distributions

Data!

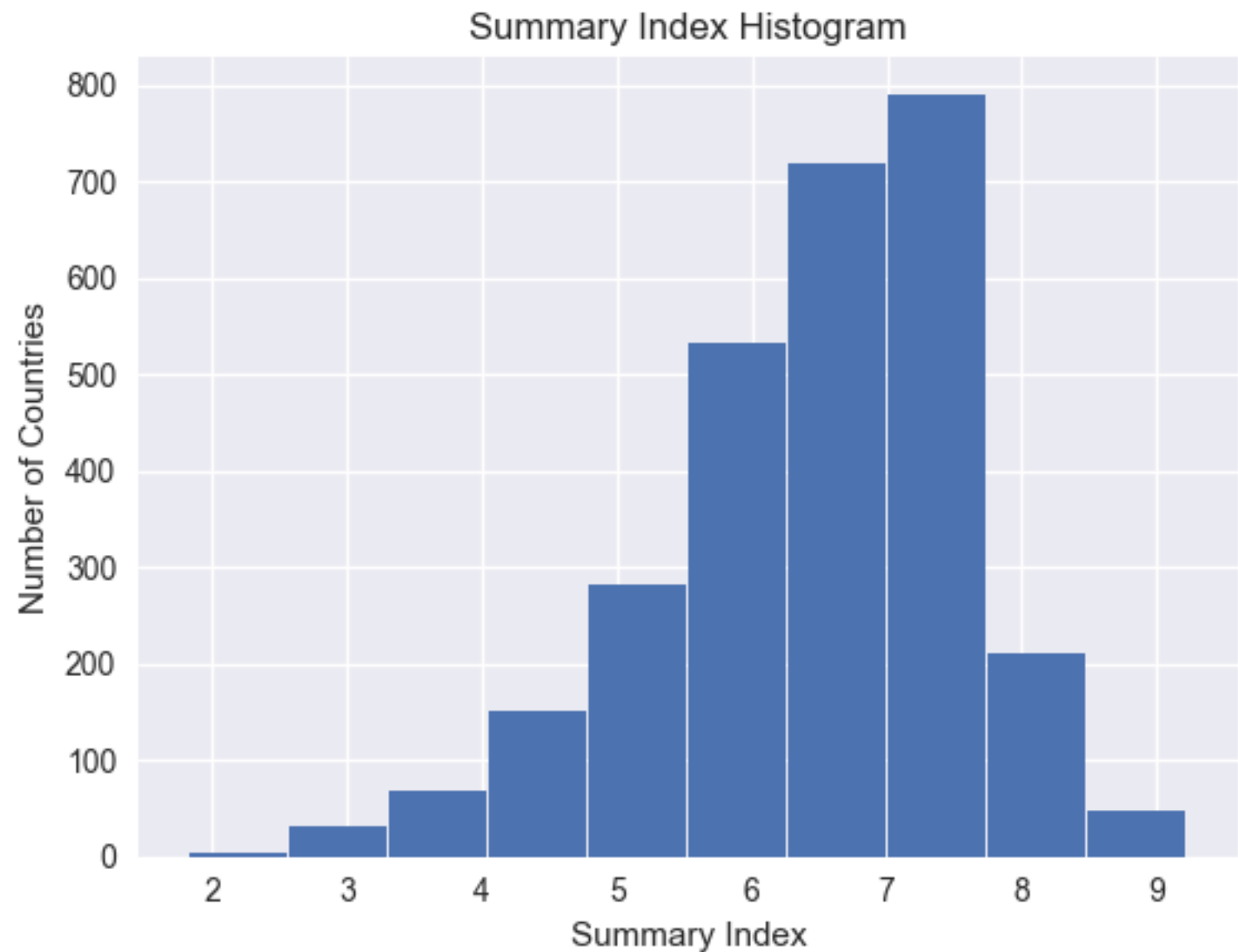


Distributions

Data!

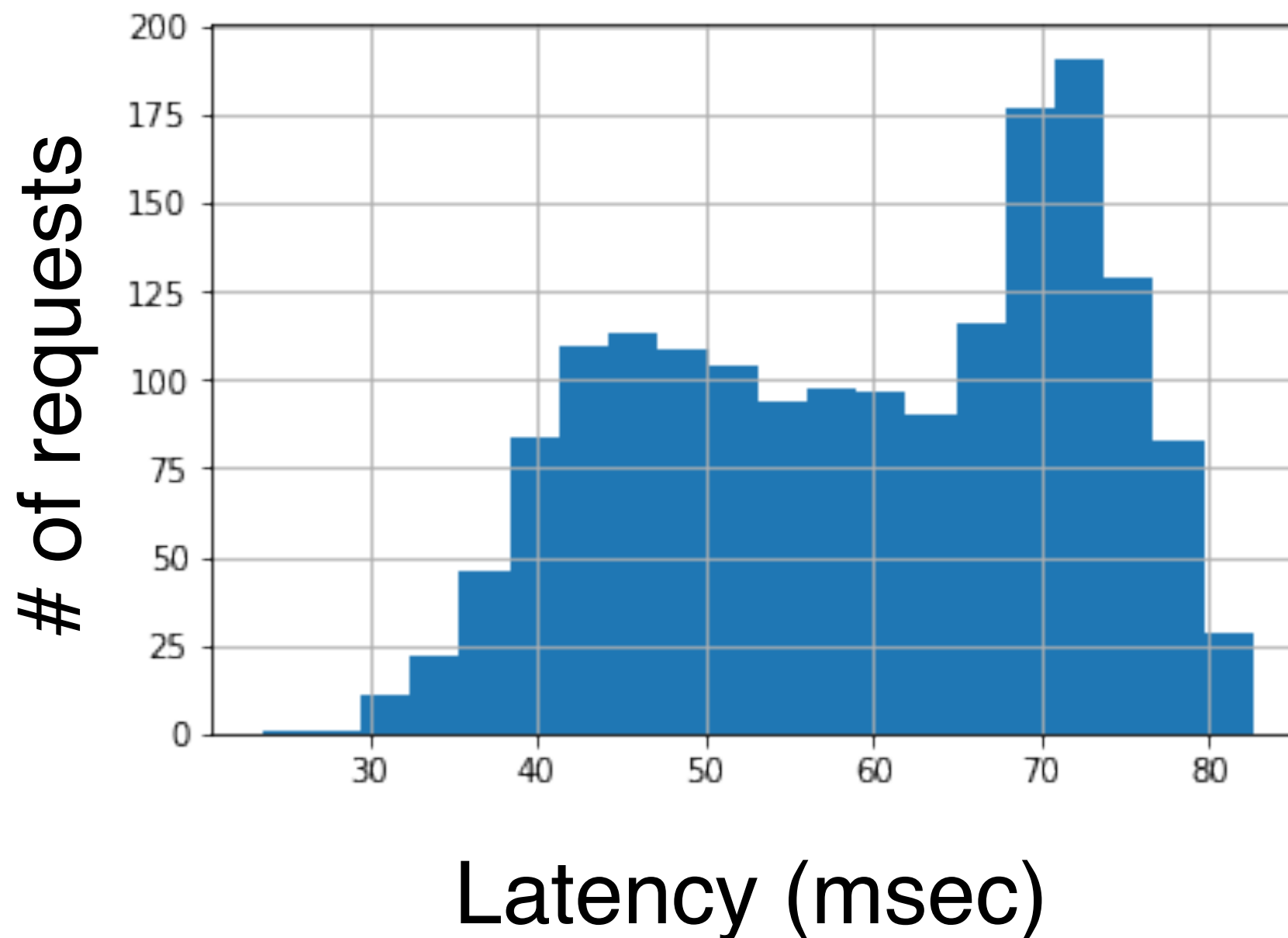


Histogram!



Latency Distribution

Latency Histogram



Jupyter Notebooks

Jupyter Notebooks ~ Jupyter Lab ~ ipython

A web based python execution environment

- "GDB for python in a browser!" — *Rebecca Shanley*

You can do this locally or on Cloud9

```
# Use pip with current python version to install stuff
python -m pip install jupyter numpy scipy pandas matplotlib
seaborn
```

```
# run jupyter on port 8080 (open on cloud9)
ipython3 notebook --ip=0.0.0.0 --port=8080 --no-browser
```

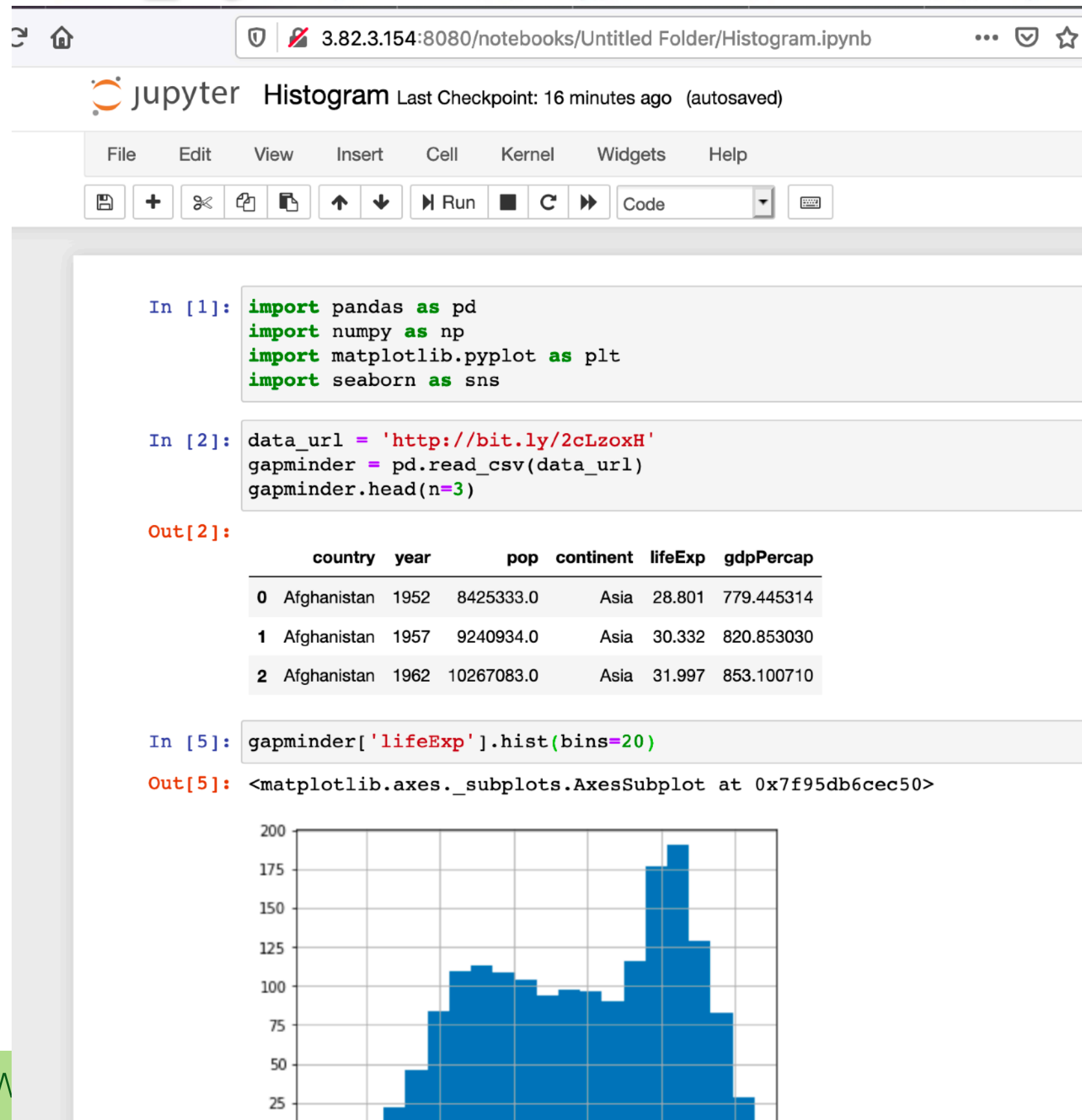
```
# Get IP from C9 Share menu and use browser to go to
http://IP:8080/?token=XXXXX
```

Jupyter

Create New
python 3
notebook

Write code

<Shift-enter>
to run a cell



Measuring Latency

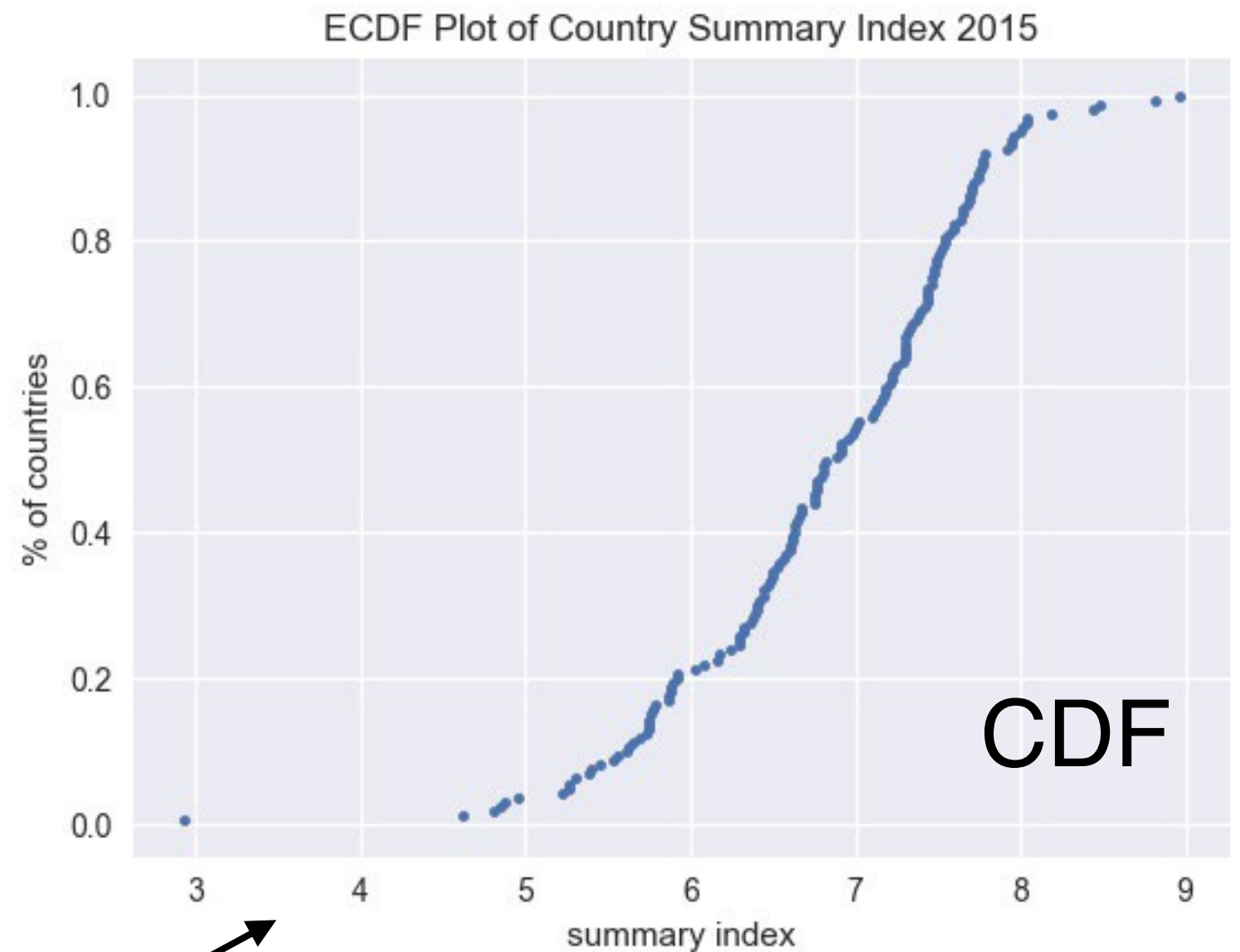
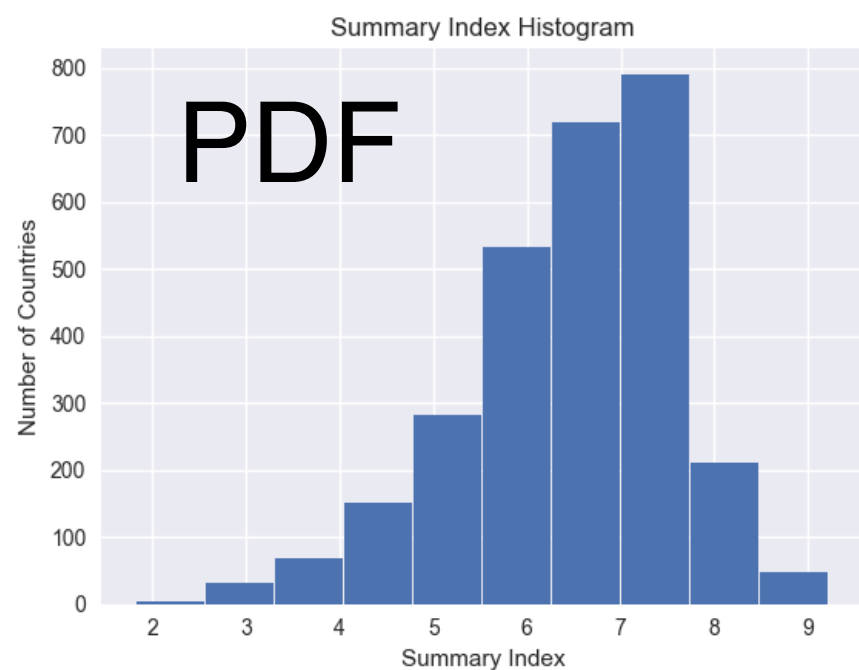
- [] Pick your favorite web site
- [] Make 100 http requests, sleep 1 sec after each
- [] Record response time for each request
- [] Plot a histogram of the response times
- [] (optional) Plot a CDF of the response times

Use a library like **pandas** or **matplotlib** for graphs

- Your favorite search engine can help!

Distributions

Data!



CDF

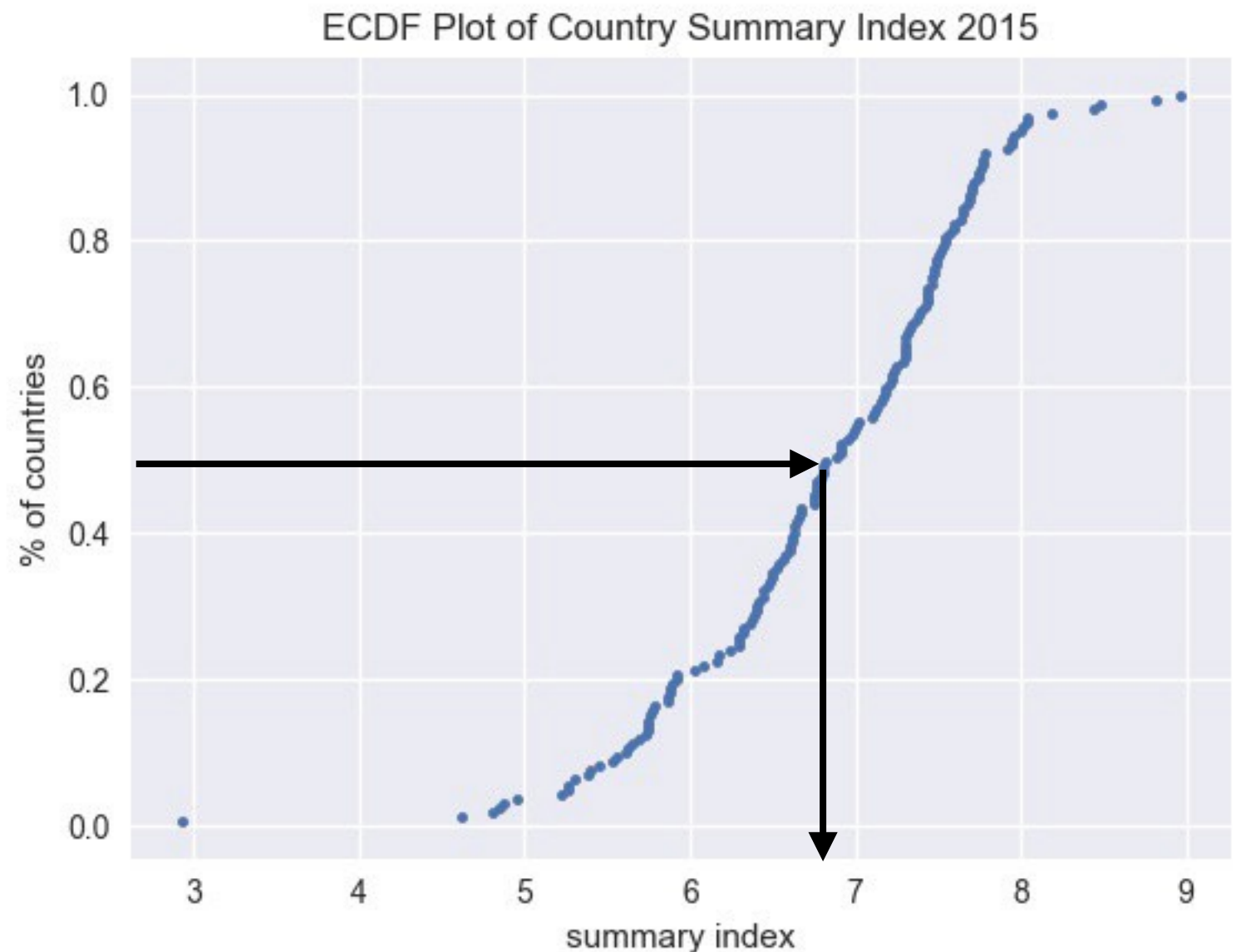
Cumulative Distribution Function

- The integral of a histogram (or PDF)

Tells you what % of measurements are at least **X** good

50% = median

“Half of all countries have an index of at least 6.8”



CDF

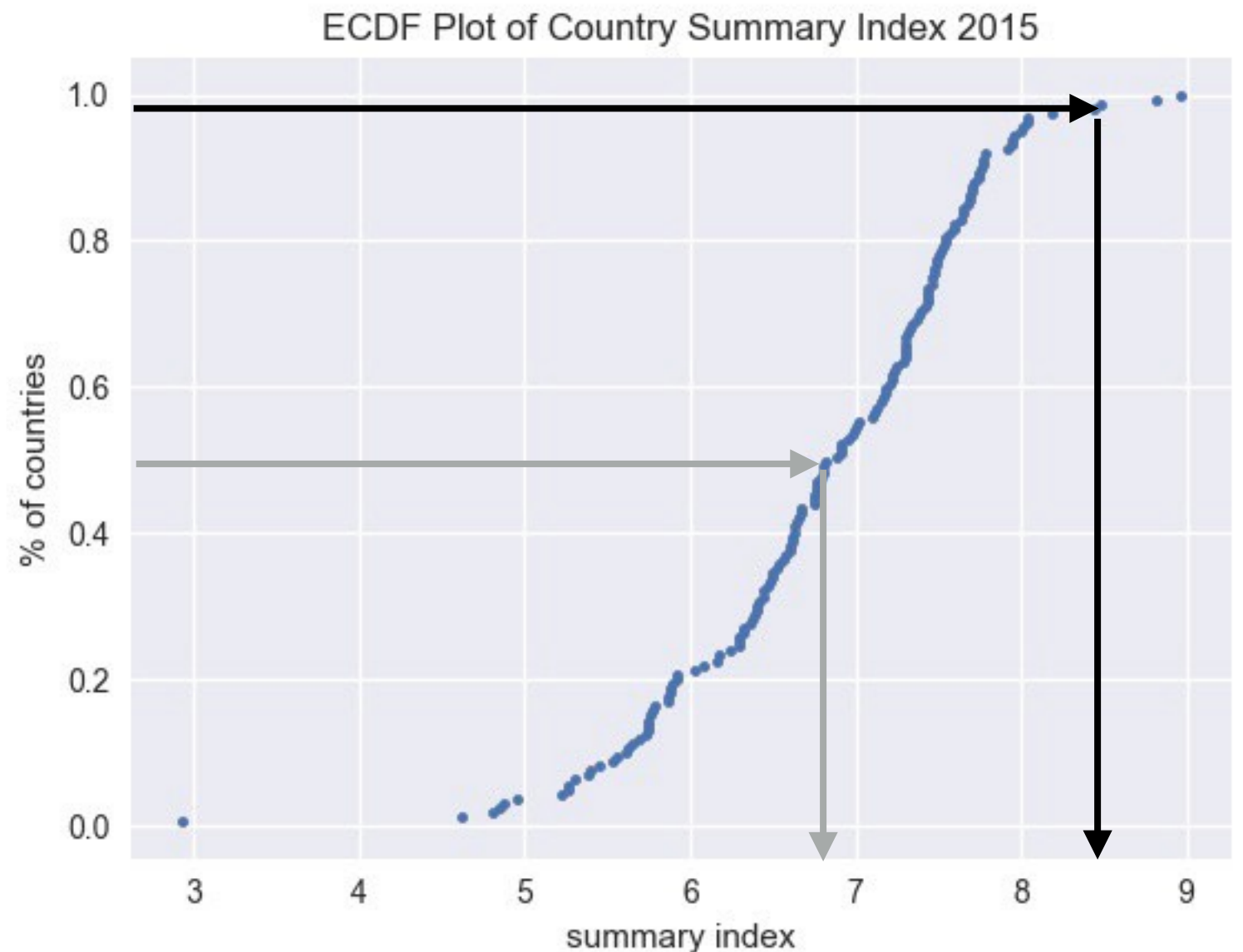
Cumulative Distribution Function

- The integral of a histogram (or PDF)

Tells you what % of measurements are at least **X** good

99th percentile

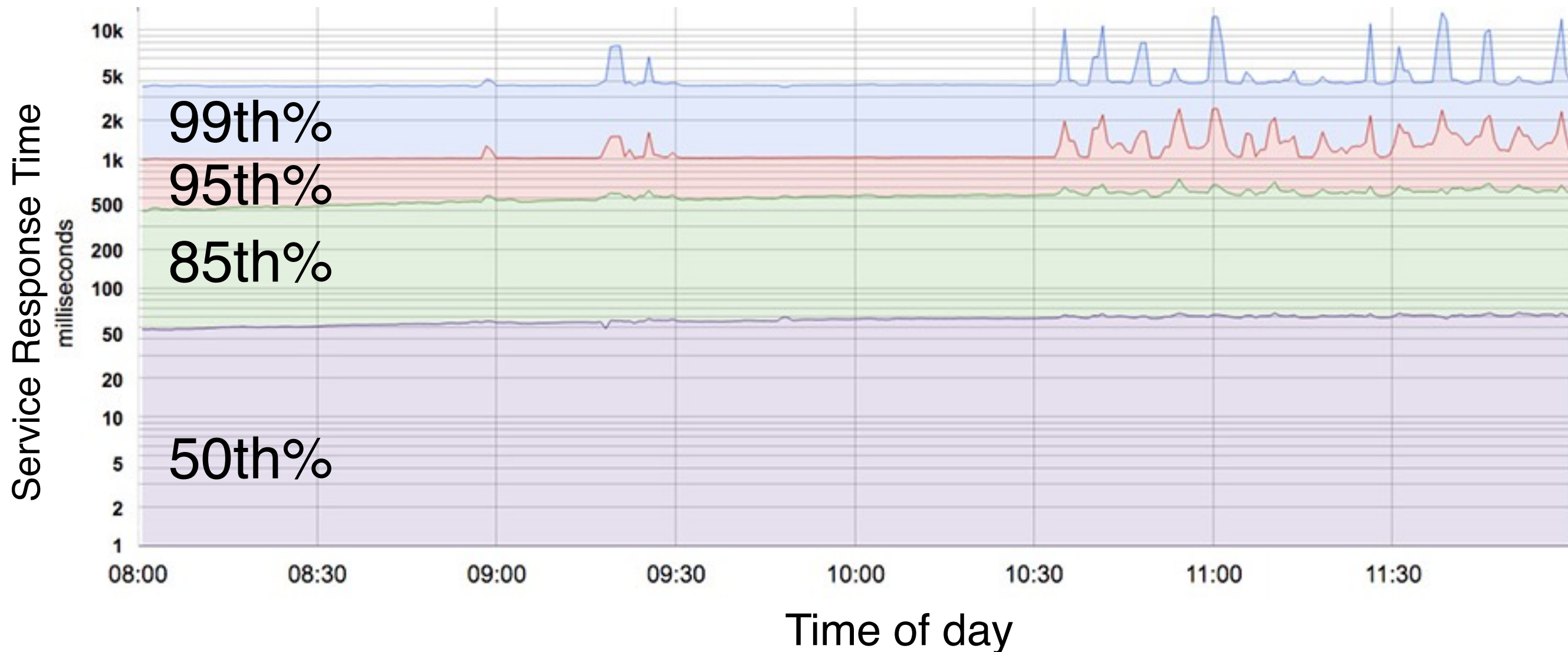
“Almost all countries have an index of at least 8.4”



Latency Distributions

CDF and Percentiles are important for understanding application performance

- More important to help sad/slow customers than fast ones



from <https://landing.google.com/sre/sre-book/chapters/service-level-objectives/>

Service Level Objectives

Performance (or other characteristic) targets

- Also used for things like service up time (e.g., 99% availability)

90% of Get RPC calls will complete in less than 1 ms.

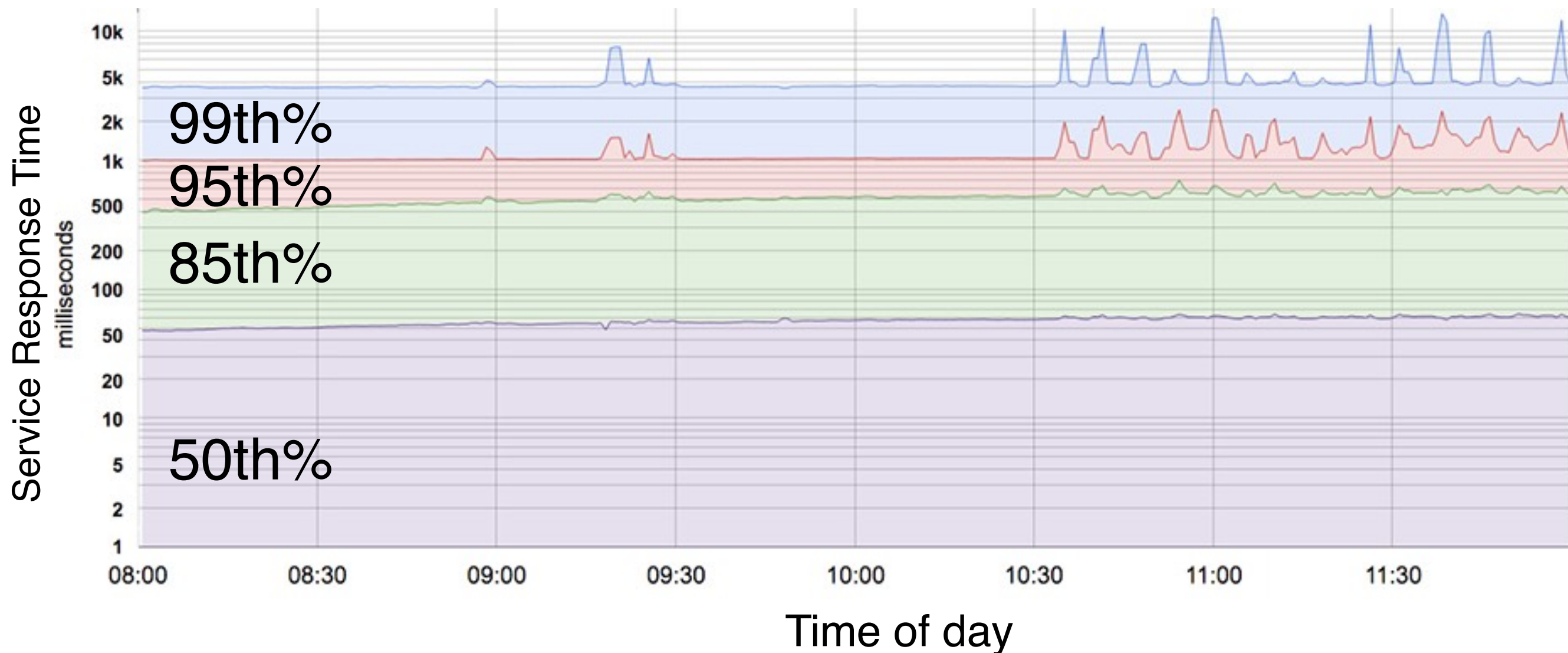
99% of Get RPC calls will complete in less than 10 ms.

99.9% of Get RPC calls will complete in less than 100 ms.

Why not have a rule for 100%?

Latency Distributions

Why do we have variance in latency?

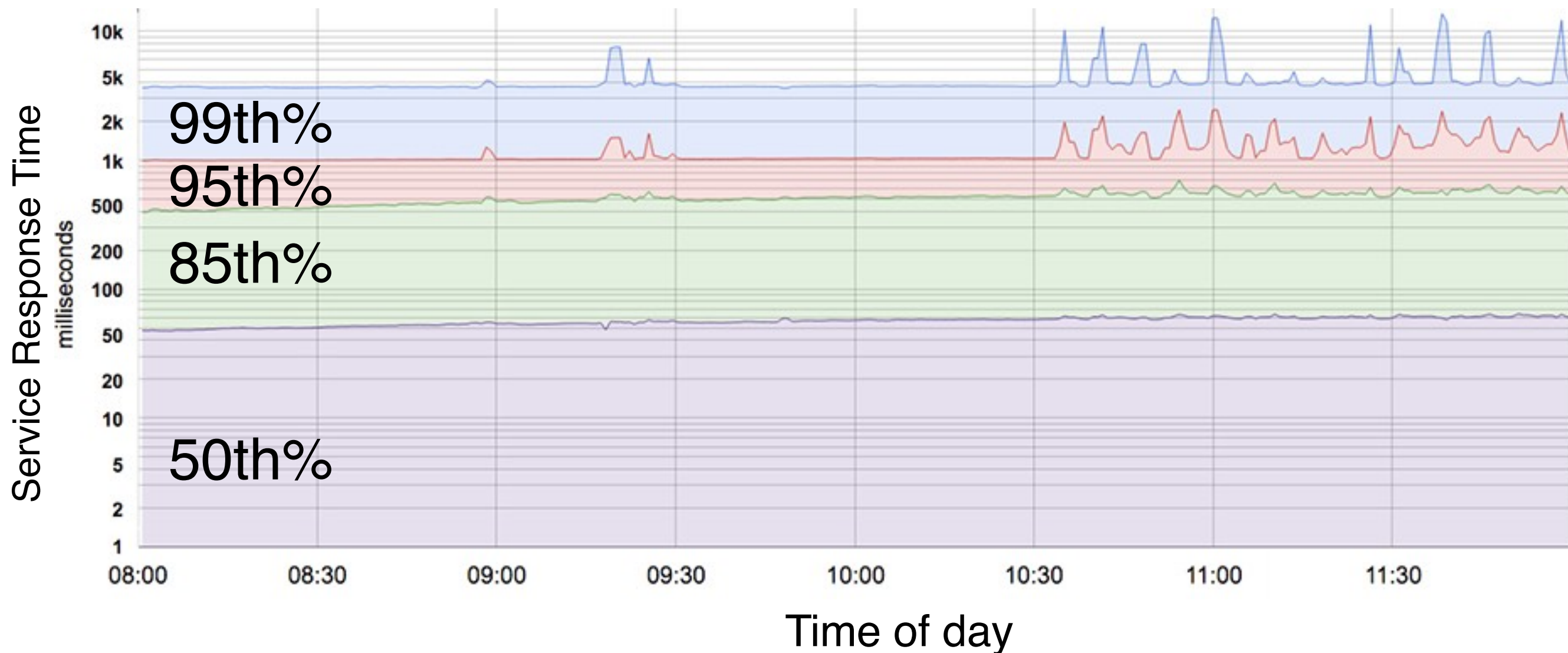


from <https://landing.google.com/sre/sre-book/chapters/service-level-objectives/>

Latency Distributions

Why do we have variance in latency?

- Bursty client workloads, disk seeks, network drops, etc



from <https://landing.google.com/sre/sre-book/chapters/service-level-objectives/>

Workloads

Users send requests to a server...

Closed Loop workload

- Next request delayed until first finishes

Open Loop workload

- Requests arrive at a constant rate

Which is more realistic? Which is harder to handle?

Workloads

Users send requests to a server...

Closed Loop workload

- Next request delayed until first finishes

Open Loop workload

- Requests arrive at a constant rate

Which is more realistic? Which is harder to handle?

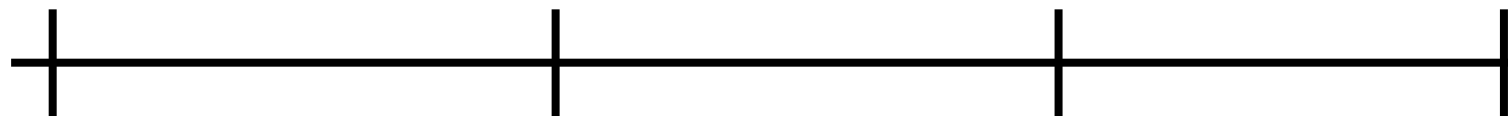
Open loop is more realistic, but closed is harder to handle and simpler to model

Poisson Arrival Process

Consider a grocery store...

Arrival rate = 3 customers per minute

- We know the average rate, but not the specific timings
- Arrival of one customer has no impact on next (independence)
- Two customers can't arrive at exact same time

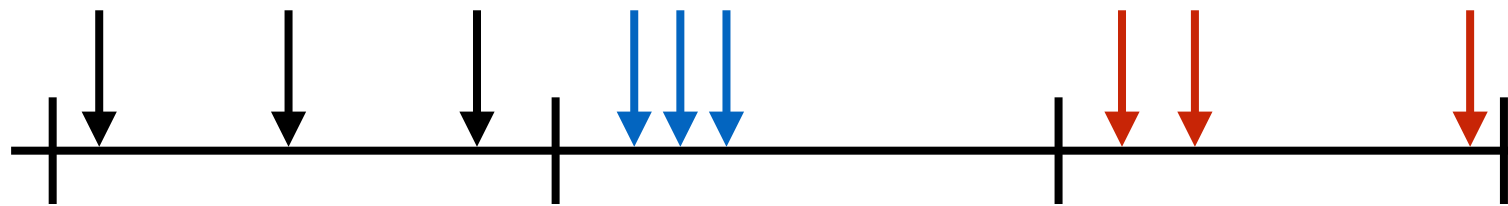


Poisson Arrival Process

Consider a grocery store...

Arrival rate = 3 customers per minute

- We know the average rate, but not the specific timings
- Arrival of one customer has no impact on next (independence)
- Two customers can't arrive at exact same time



Poisson Model

Based on arrival rate we can predict some performance metrics

λ = event rate x observation time

k = number of events

Probability of k events happening:

$$P(k \text{ events in interval}) = e^{-\lambda} \frac{\lambda^k}{k!}$$

Probability of waiting at most t time:

$$P(T \leq t) = 1 - e^{-\frac{\text{events}}{\text{time}} * t}$$

Code Reviews

hello internet

A socket programming resource

[] give your git usernames to prof

[] pick a PR you will review

- You don't need to know the language, but it will help

[] Clone PR code

[] Review code

- Follow template linked on website

Java

C

C++

C#

Go

Lua

Perl

Python

Python/Jupyter

Ruby

Rust

Swift

Scala