Reasoning About Performance in Distributed Systems

COS 316: Principles of Computer System Design
Lecture 14

Nicolaas Kaashoek

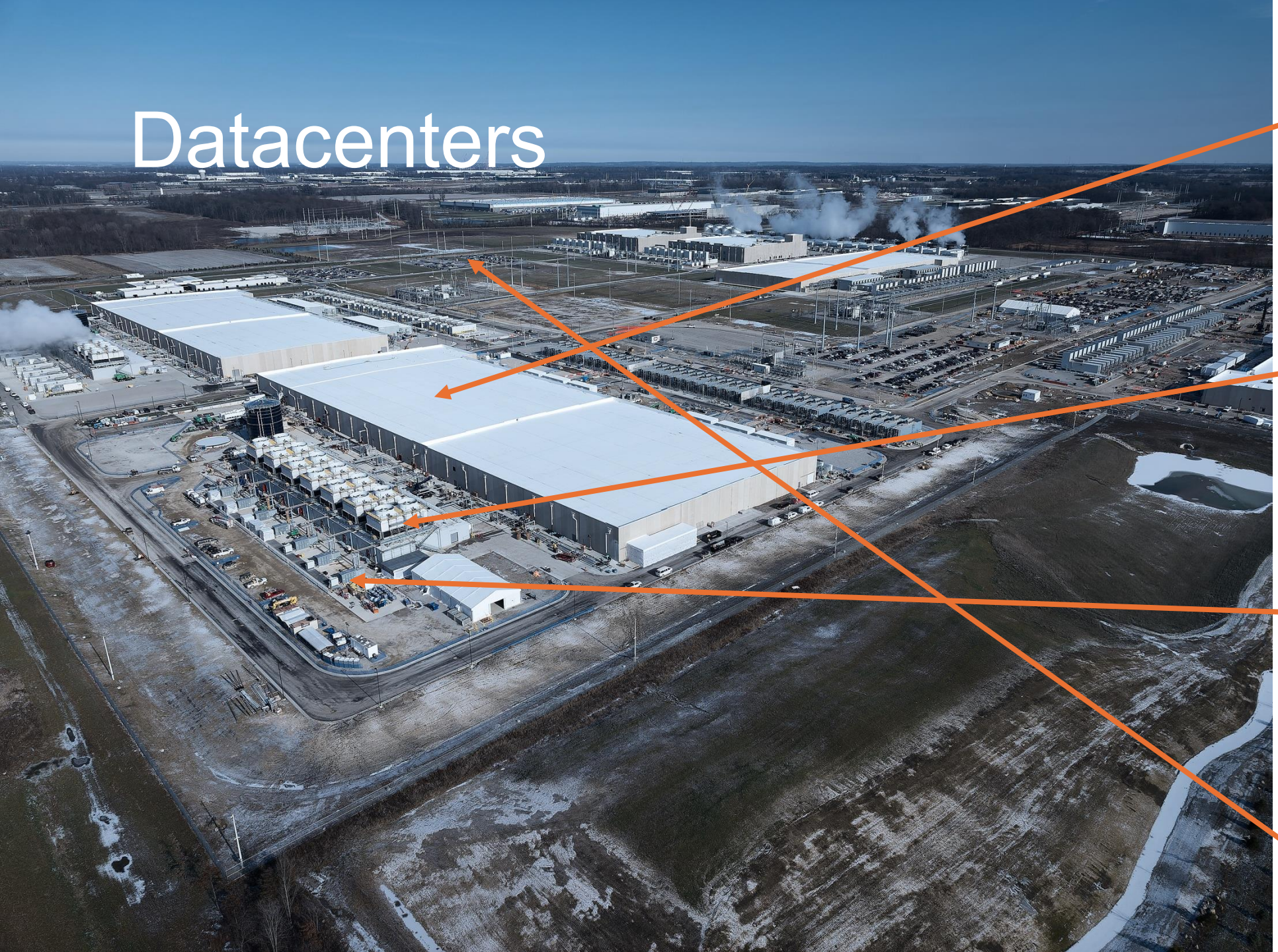
Distributed System Scale

Distributed systems operate at varying scales:

- Assignment 3: 1 User, 1-10 requests in a second
- Canvas: 10s of users, 1-10 requests per second
- New Jersey DMV: Millions of users, 1 request per second*
- Meta: 100s of Millions of users, billions of requests per second

Performance matters!

Datacenters



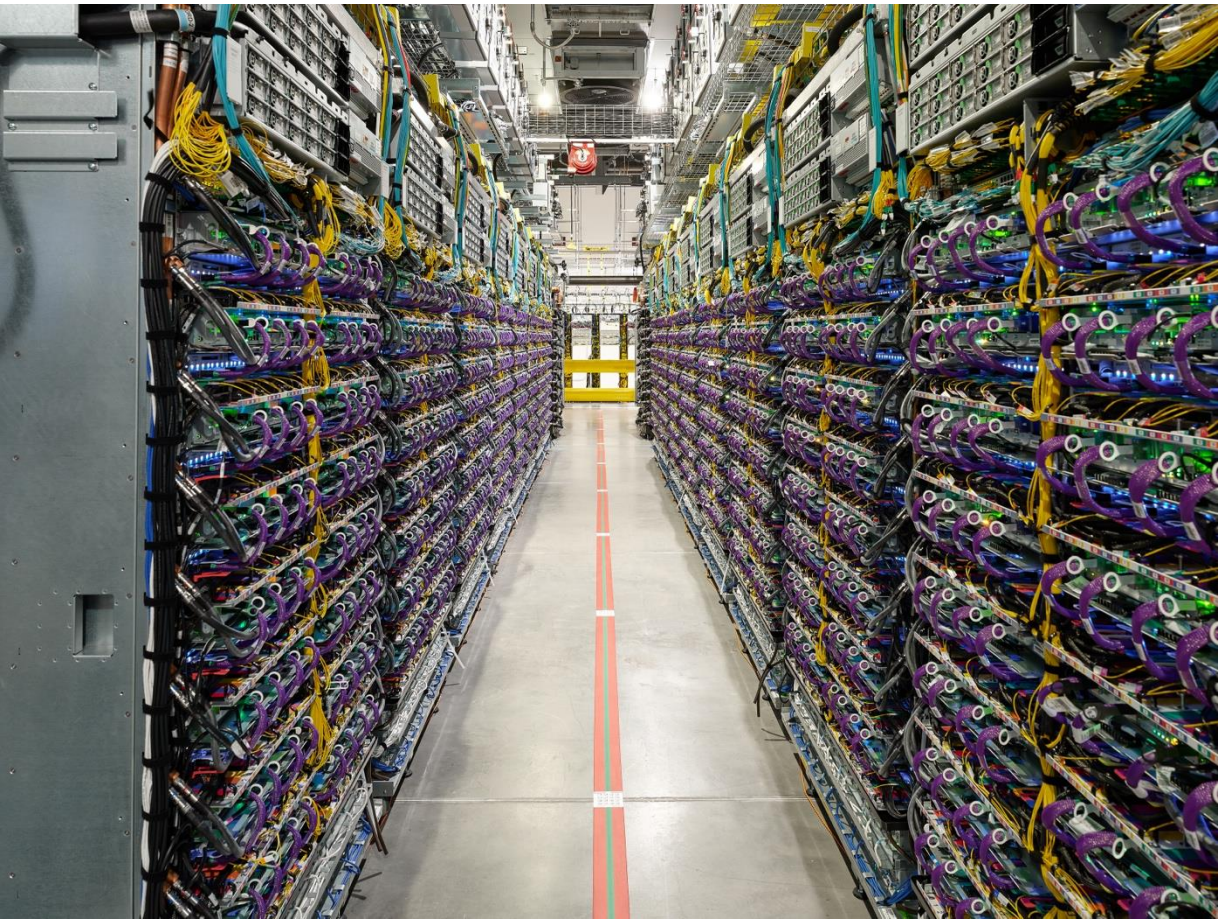
Buildings with Servers

Cooling

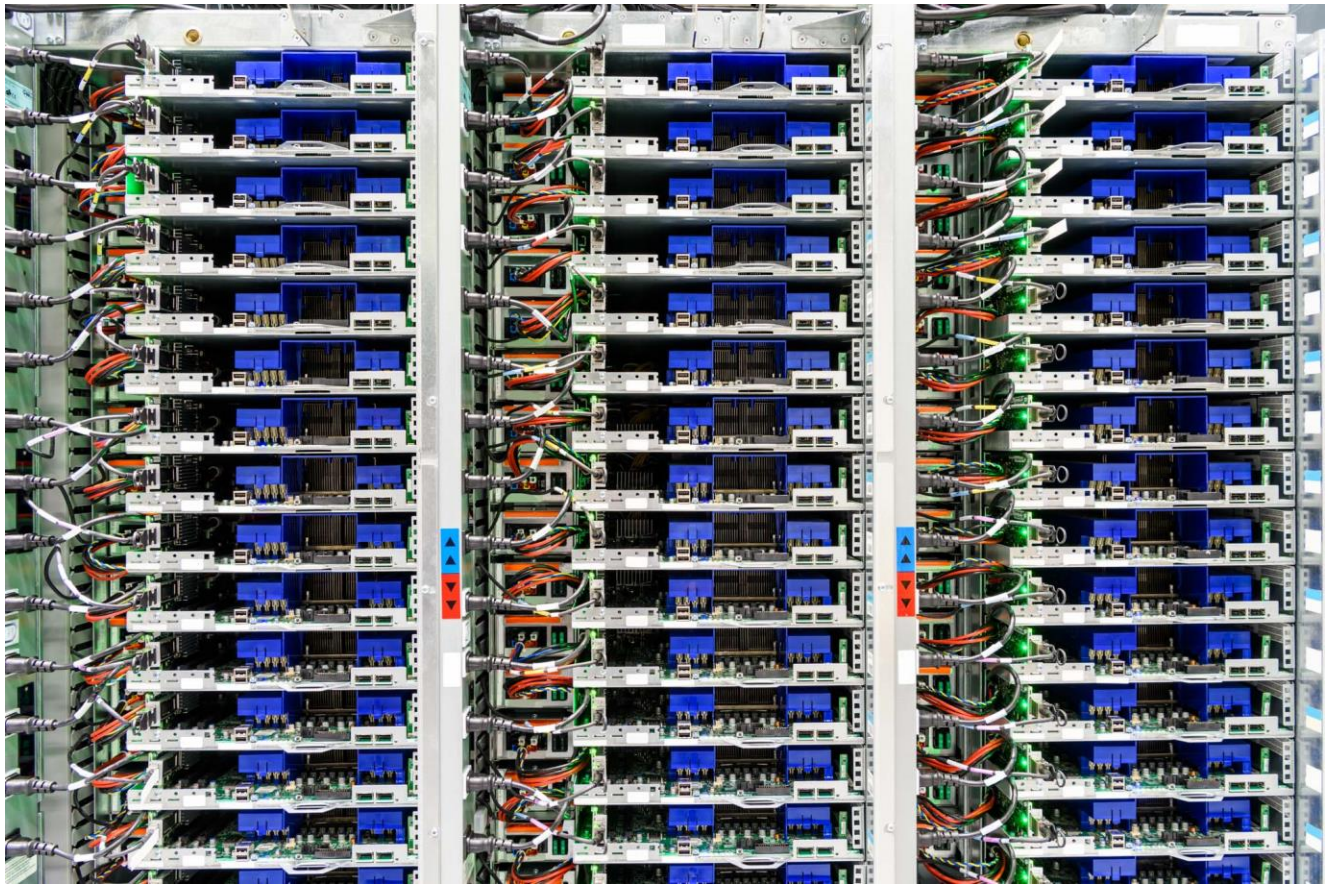
Backup Generators

Electrical Substation

Inside a Datacenter

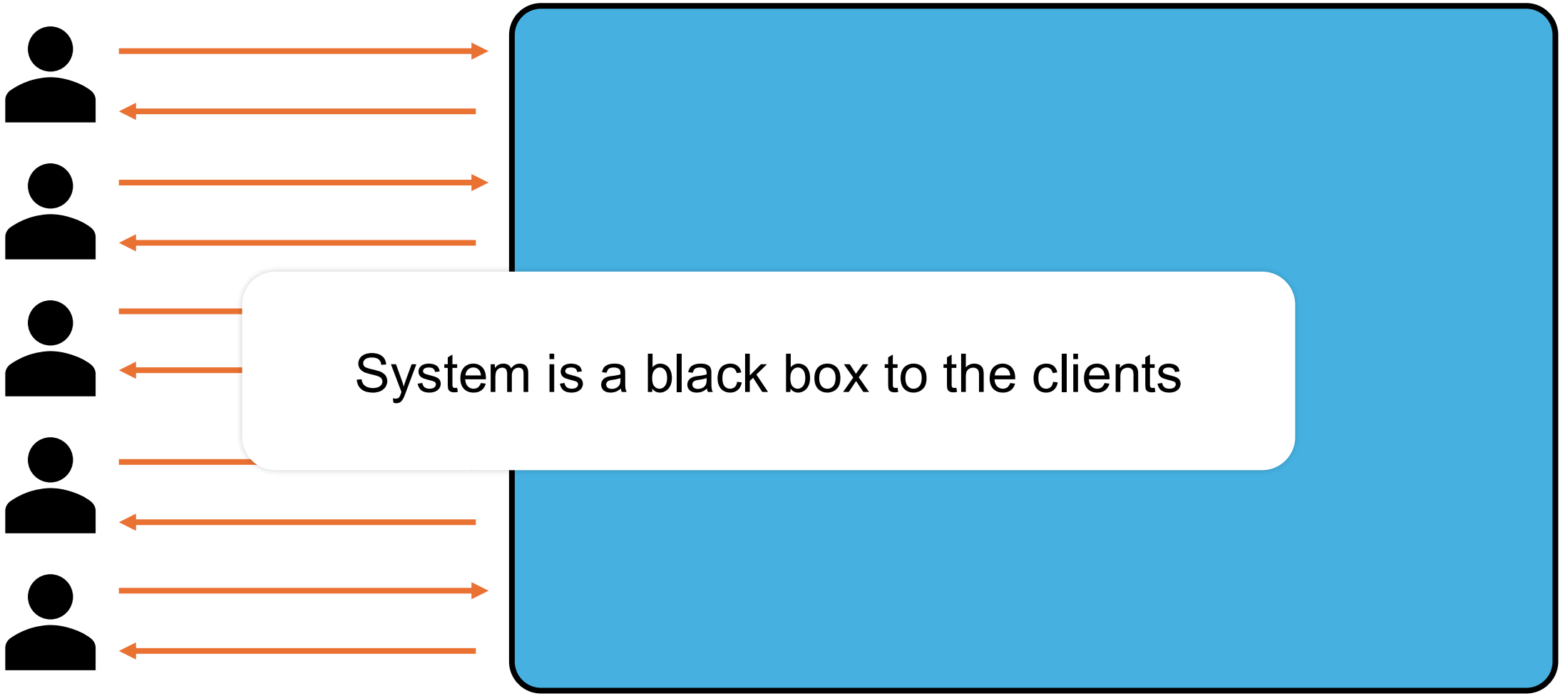


Central Ohio



Eemshaven

Measuring Distributed Systems



Metric 1: Latency

Latency measures the time it takes to do *something*

End-to-end latency measures the time between client request and client response

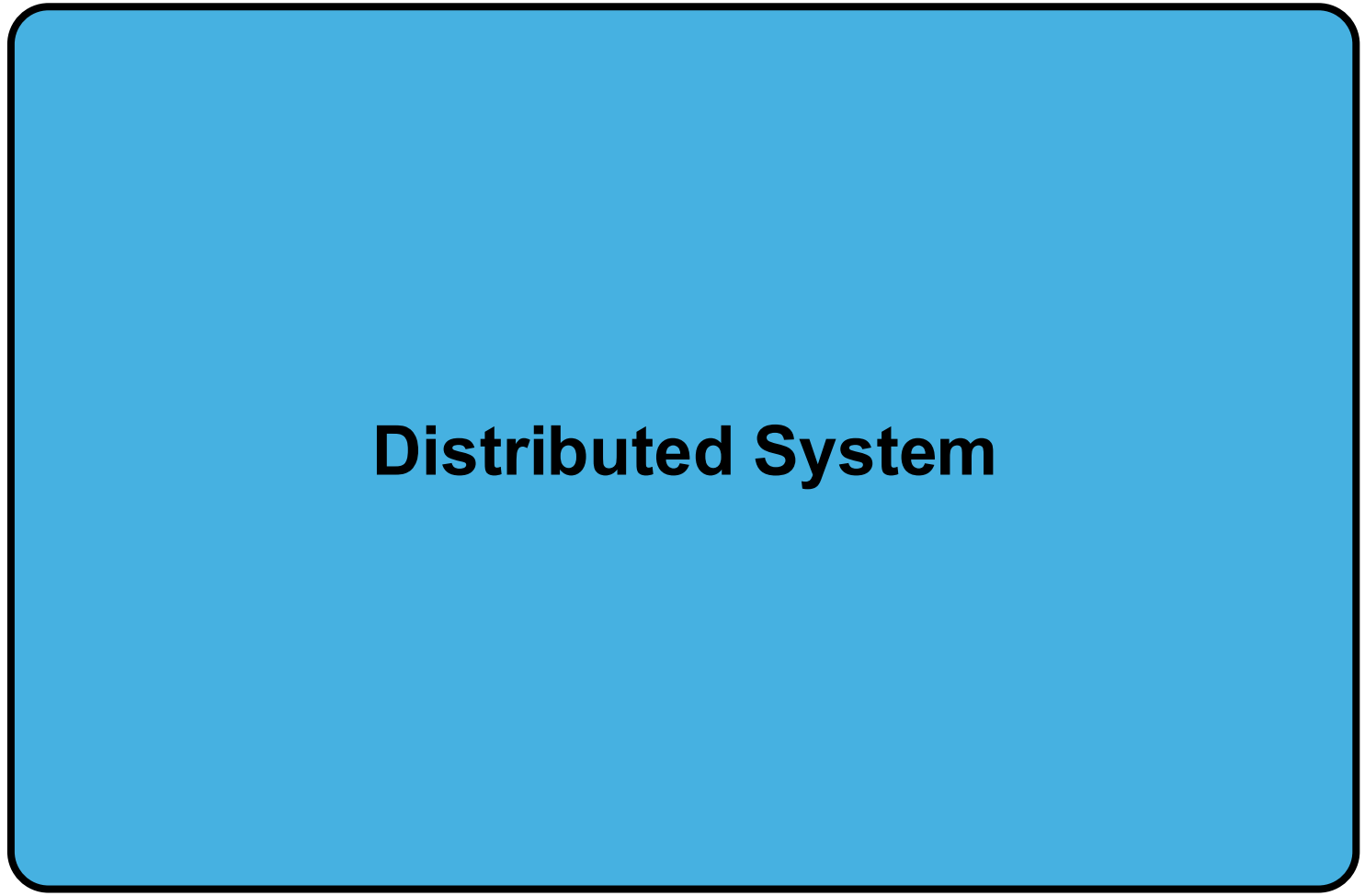
End-to-end latency captures everything going on inside a system, but misses details

Usual unit: ms

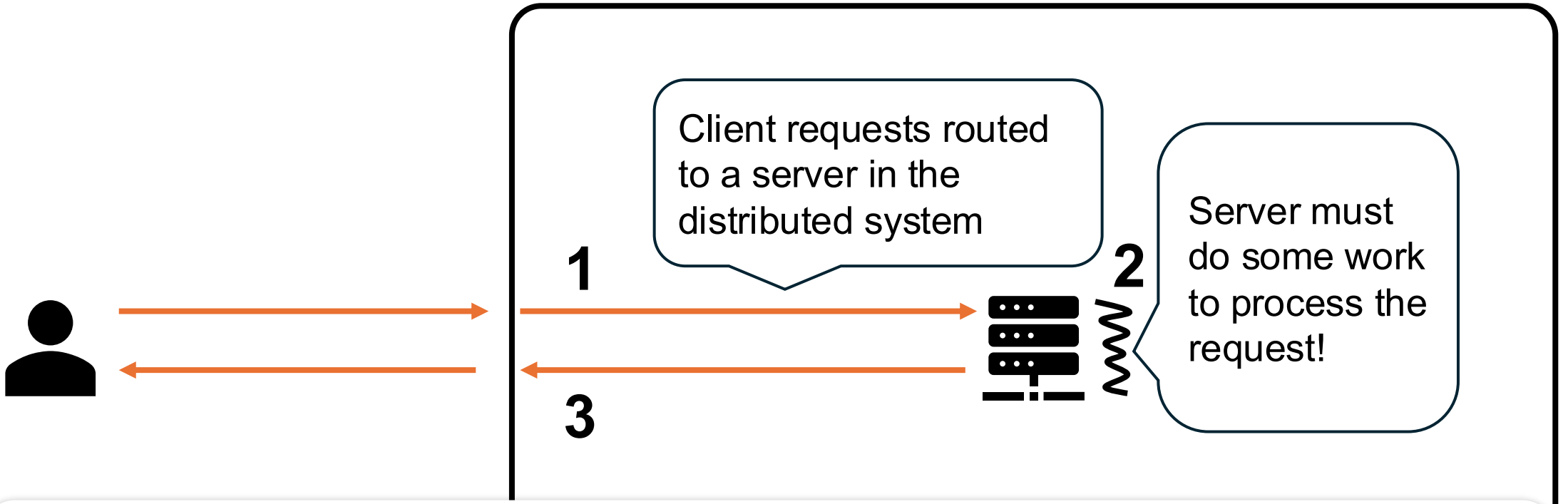
Measuring Latency

Latency measured externally by client

Collect results from many clients!



Breaking Down End-to-End Latency



Total latency = 1 + 2 + 3

Metric 2: Throughput

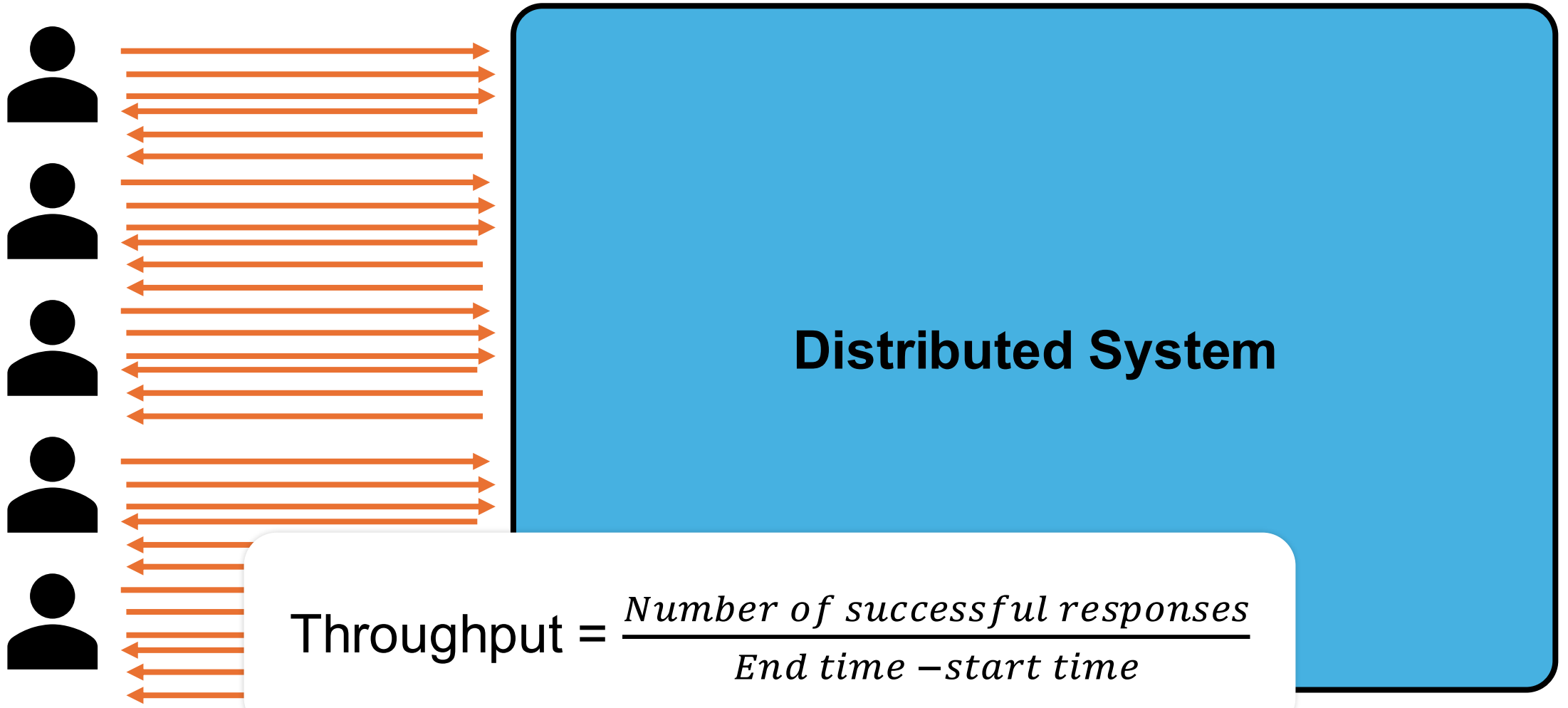
How many requests can our system handle in given period if time?

Analogous to link bandwidth in networking!

Measured externally as rate that responses exit system

Usual unit: reqs/sec

Measuring Throughput



$$\text{Throughput} = \frac{Number\ of\ successful\ responses}{End\ time - start\ time}$$

Throughput/Latency Relationship

The more requests we send to the system, the more it has to process

Eventually the system will overload. Two options:

- Drop all requests until it can start processing them
- Enqueue requests and process as fast as possible

Either way: overload creates higher latencies

Measuring Throughput 2

Blindly hammering a server with requests is not a good idea!

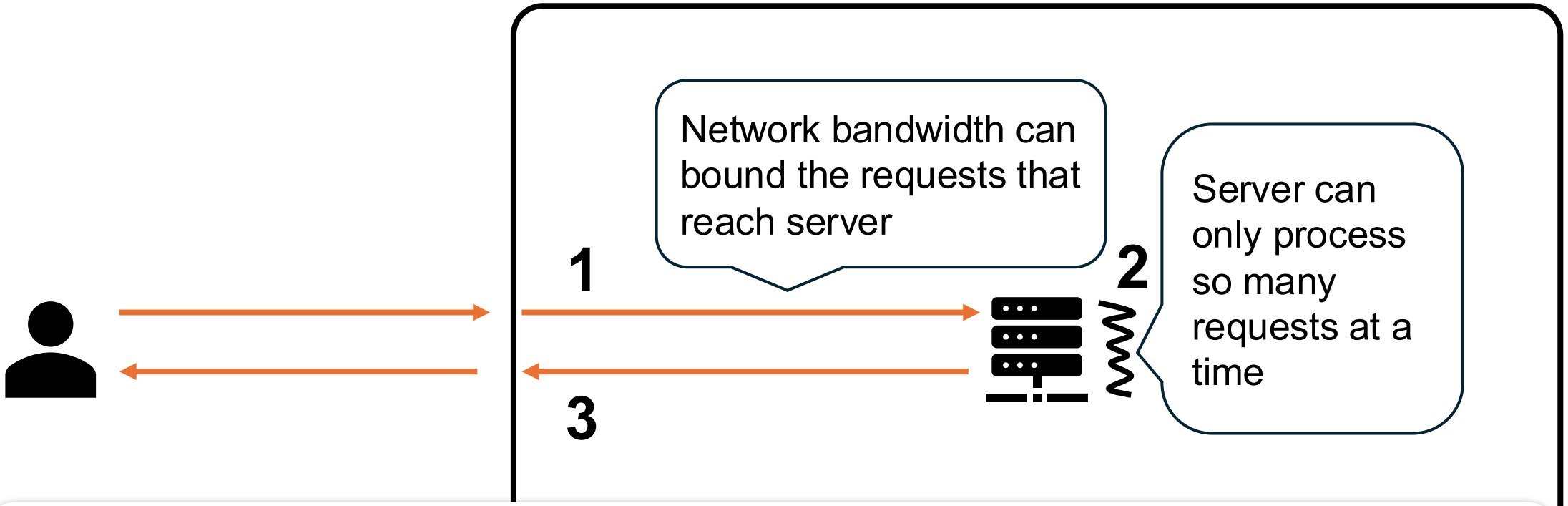
- We learn very little by doing this

But, figuring out where a system overloads is important!

Approach:

- Start with low load
- Increase load
- Repeat until observed throughput levels off

Reasoning About Throughput



Total throughput = min(1, 2, 3)

Throughput Bottlenecks

Bottlenecks to throughput can come from multiple places

Like bandwidth, there's a bottleneck somewhere
- Bottleneck link bandwidth
- Server CPU
- Server network card
- Server's storage device
- Outgoing bottleneck link bandwidth
- Many others!

Load Generation

Closed-loop

- Each “client” sends one request, waits for the response to come back, and then sends another request
- More “clients” => more load

Open-loop

- Load is generated independently of the response rate of the system, typically from a probability distribution
- More directly control the load on the system

Which one is more realistic?

We’ll reason using closed-loop clients

Important Numbers in Distributed Systems

Latency (Review):

Machines in DC: <1ms

Dorm to Cycles: ~1ms

Dorm to NYC: ~10ms

Dorm to VA: ~20ms

Dorm to CA: ~60ms

Dorm to EU: ~100ms

Dorm to JP: ~150ms

Throughput:

Phone: 100s req/s

Laptop: 1-5k req/s

Desktop: 10-20k req/s

Single Server: 50-100k req/s

High-End Server: 500k+ req/s

Hyperscale System: 10^{6-9} req/s

Mental Model of an Experiment

Let’s start with 1 closed loop client, and submit requests to a server

- What’s our expected latency?
- What’s our expected throughput?

Double the number of clients, what happens?

Repeat

Metrics We Need to Know

What do we need to know to do our mental modeling?

- How long it takes to get from the client to the system
- How many requests the server can handle (how long it takes to overload)
- How long it takes the server to process a single request

Throughput-Latency Curves 1

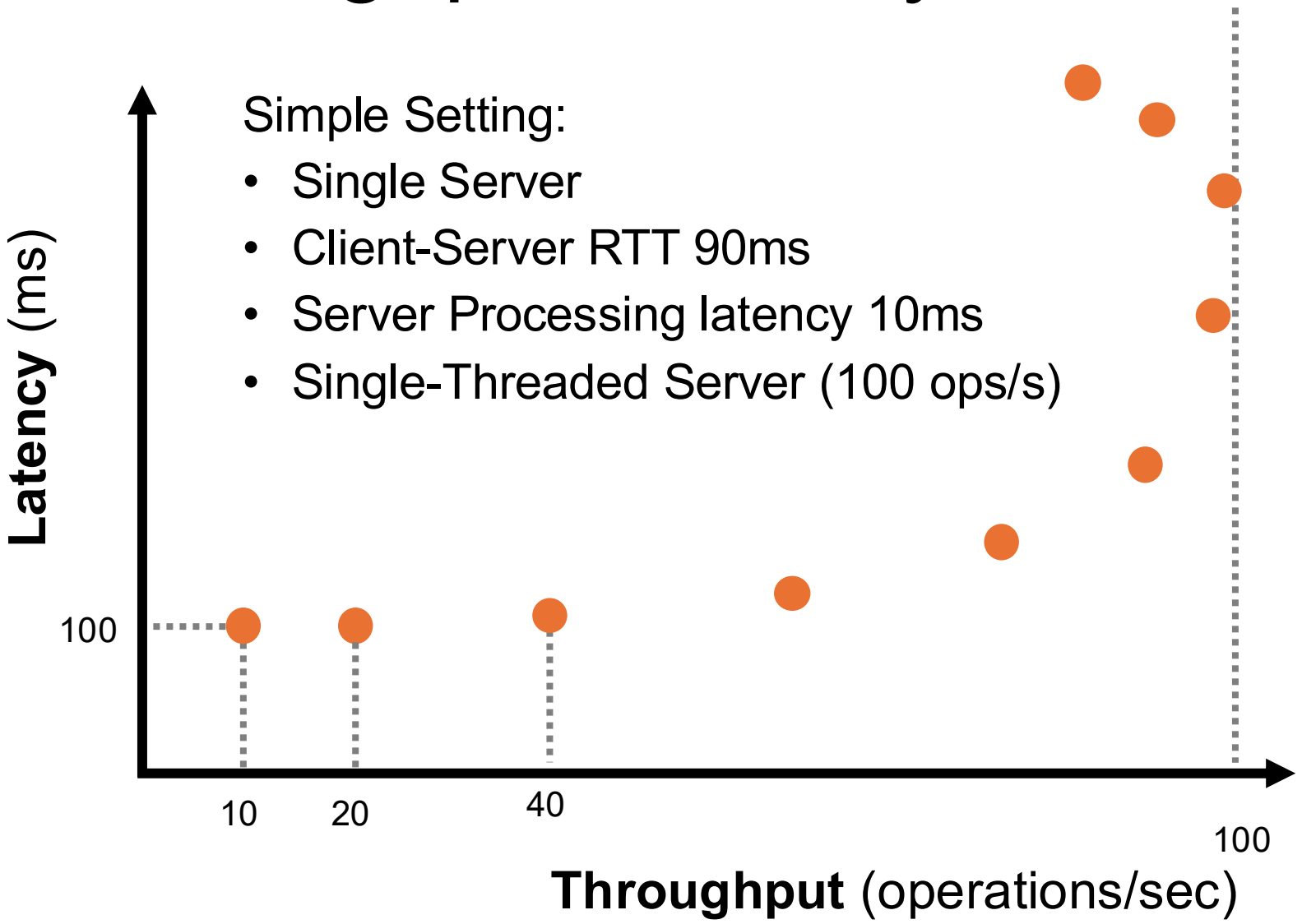
Many distributed systems are benchmarked by a throughput-latency graph

Remember: There's a coupling between throughput and latency!

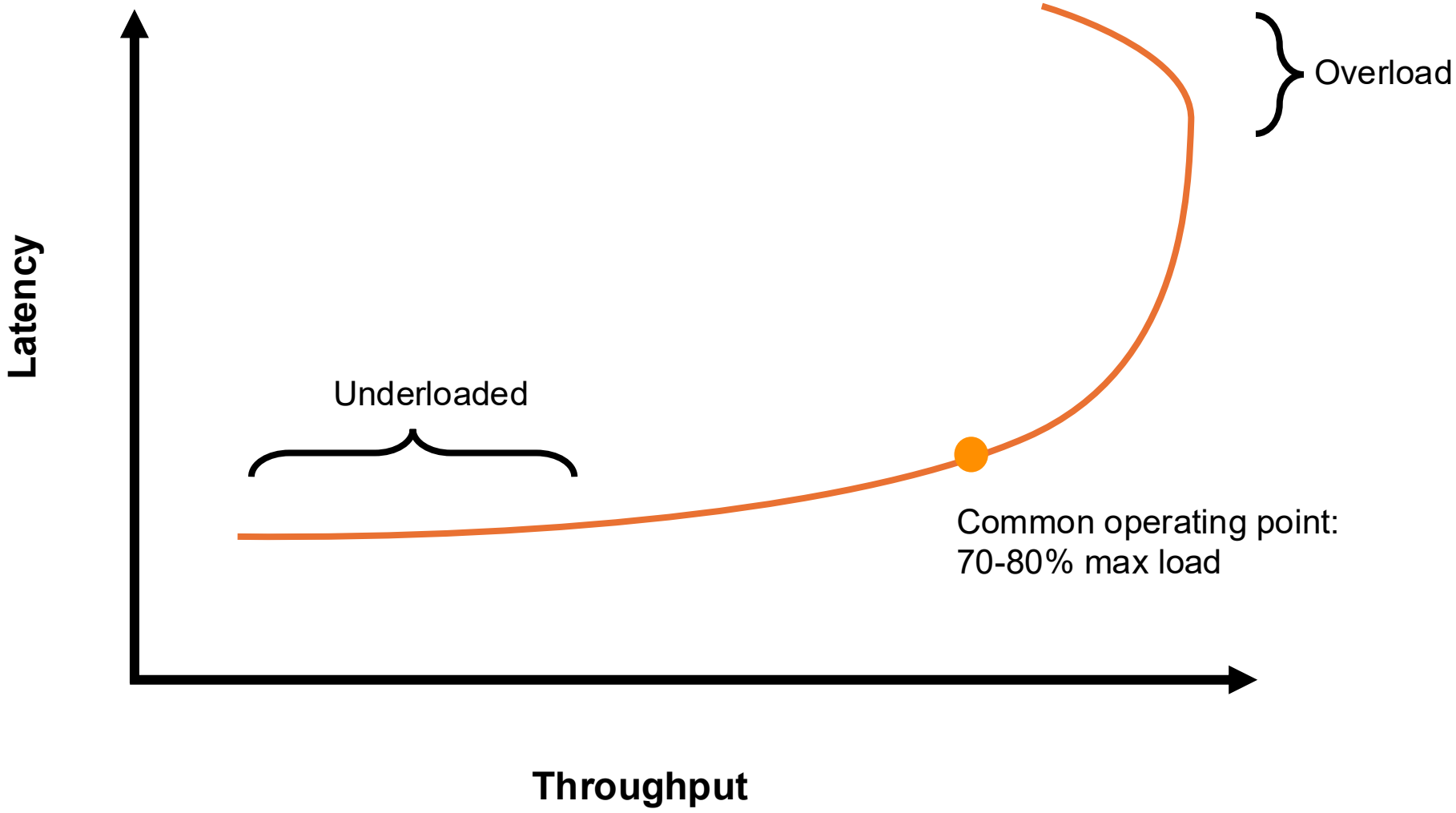
Which one goes on the X axis?

- Throughput!

Throughput-Latency Curves 2



Throughput-Latency Curves 3



Primary-Backup Performance

Clients talk to primary

Primary talks to backups

Ignore coordinator for now -> assume that client already knows who the primary is, and that no failures occur

PB DC Perf: Latency

What is the latency of a single client talking to the primary?

What steps makeup latency, and how long does each take?

1) Client sends request to primary (30ms)
2) Primary adds request to its log (1ms)
3) Primary replicates operation to all replicas (1ms RTT to backups, 1ms for backups to write to log)
4) Primary applies request to internal state machine (7ms)
5) Primary sends response to client (30ms)

Total: 70ms

Note: The numbers are arbitrary, based on realistic values that work out to a nice, round final

PB DC Perf: Throughput

What is our throughput bottleneck going to be?

- Almost certainly the primary!

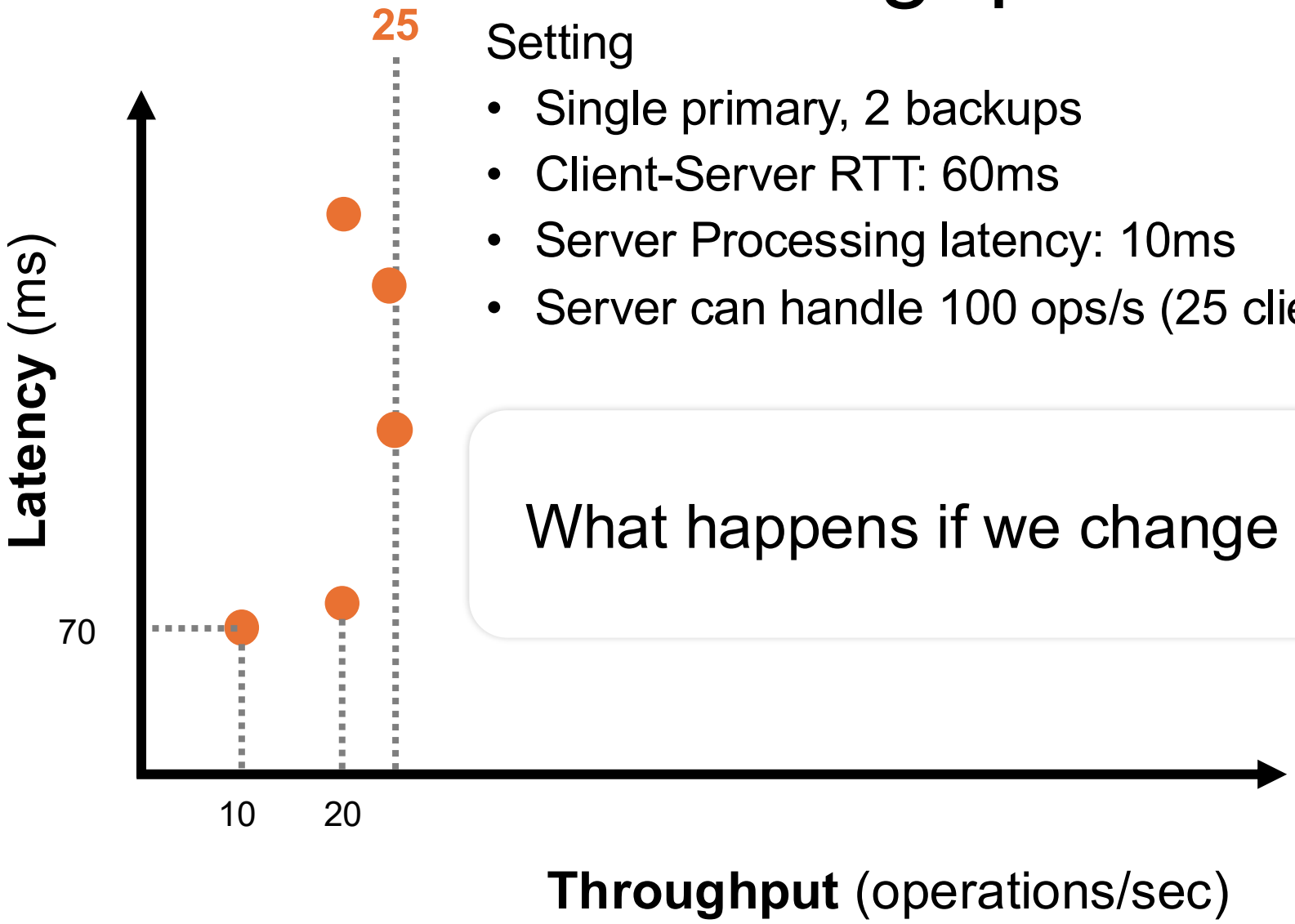
Primary receives messages from client & backups

- Each request from client creates n requests from backups
- Let’s say 3 backups

If primary can handle 100 requests/sec, how many client requests?

- 1 client request requires the primary to reach out to 3 backups, for 4 total requests
- For simplicity, assume each request takes same processing time: 100/4 = 25 client requests/sec.

PB DC Perf: Throughput-Latency



Setting

- Single primary, 2 backups
- Client-Server RTT: 60ms
- Server Processing latency: 10ms
- Server can handle 100 ops/s (25 client req/s)

What happens if we change the workload?

PB DC Perf: Read-Only

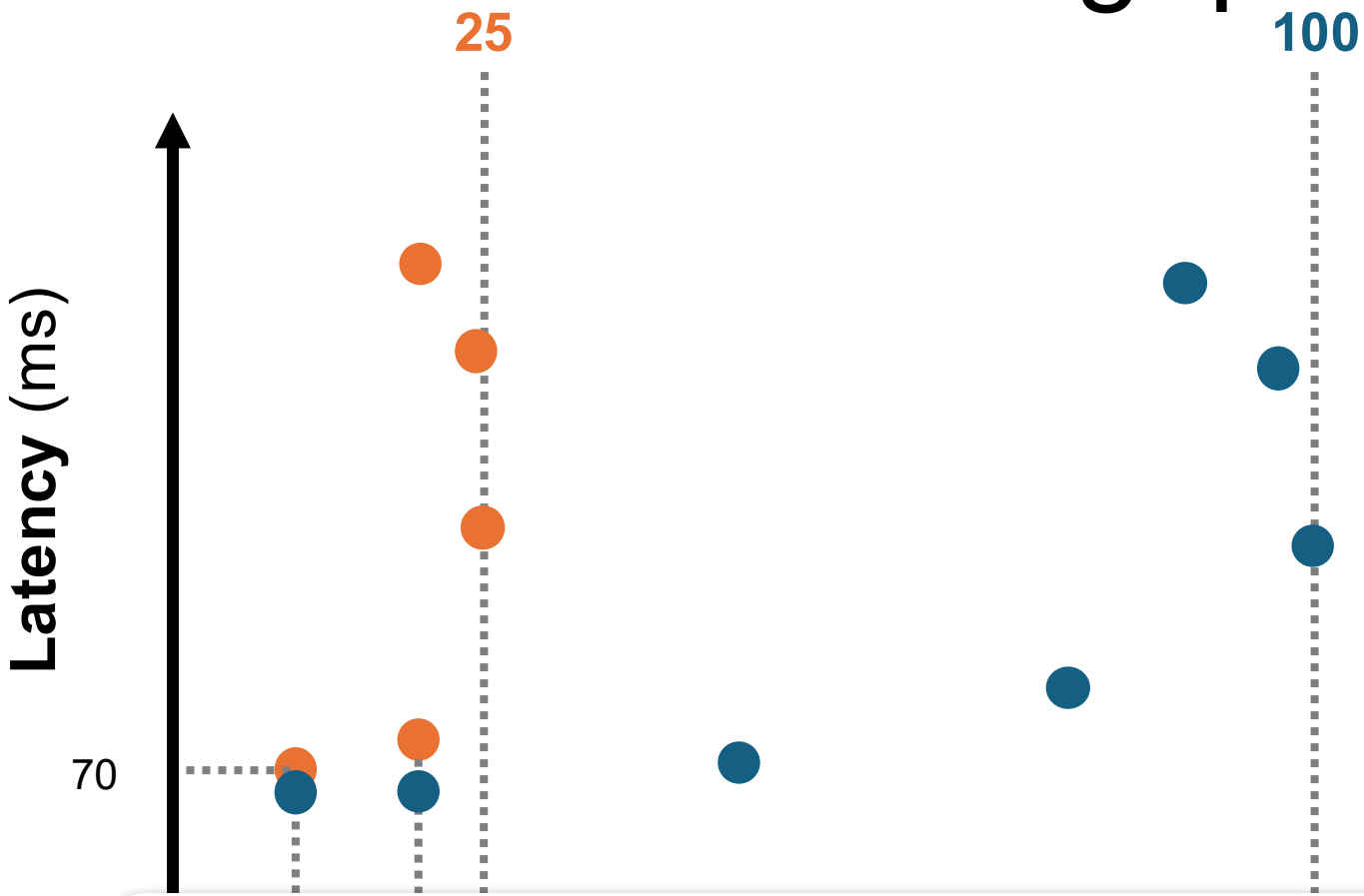
Does latency change?

- Latency decreases slightly because we don’t need to wait for the primary to contact the replicas
- For simplicity, we’ll graph with the same latency to make comparing easier

Does throughput change?

- It should! Previously, a single client request generated 4 requests the primary had to handle. Now it’s a 1:1 relationship.

PB DC Perf: Throughput-Latency 2



Setting

- Single primary, 3 backups
- Client-Server RTT: 60ms
- Server Processing latency: 10ms
- Server can handle 100 ops/s
- **All reads**

Workloads matter!

Primary-Backup WAN Performance

So far, looked at performance when replicas are close by

What happens when replicas are far apart? Why do this?

- Better fault tolerance
- Replicas closer to users?
- Other reasons (power availability, data governance, etc)

PB WAN Perf: Latency

Datacenters: California, Ohio, Virginia

Datacenter Pair	CA <-> OH	CA <-> VA	VA <-> OH
RTT Latency	50	60	20

Client to DC: 60ms RTT

Primary in Ohio, how long for a write for a client in:

- Ohio: 60ms to Ohio, 50ms from Ohio to CA. 110ms.
- California: 60ms to CA, 60ms from CA to VA. 120ms.

How about a read?

- Reads just go to primary, so 110ms for CA and 60 for OH.

PB WAN Perf: Throughput

How does moving to a WAN affect throughput?

- Backups are further apart
- Primary can still process the same number of req/s

Generally: throughput worse. Network delays, etc.

- Never going to be better than in a LAN

Conclusion

Distributed systems can operate at huge scale! This is why both performance and fault tolerance matter

Care about two metrics: throughput and latency

We can reason about throughput and latency with mental models and calculations

Workloads and system deployments matter!