Reasoning about System Performance

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CS 240: Computing Systems and Concurrency
Lecture 21

Marco Canini

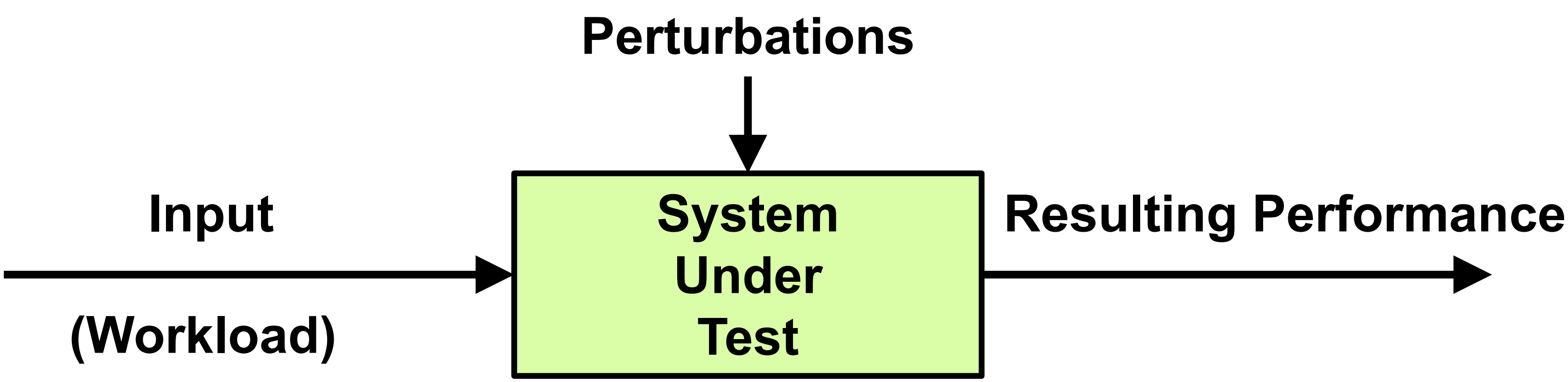
Context and today's outline

- We cared a lot about:
  - Are the results correct?
- But in practice we also need to consider quantitatively:
  - Are the results obtained in a reasonable time?
  - Is a system faster than another one?
- **Today—** How to analyze the performance of a system?

What's systems performance?

- The study of an entire system, including all physical components and the full software stack

- Include anything that can affect performance
  - Anything in the data path, software or hardware
  - For distributed systems, this means multiple servers

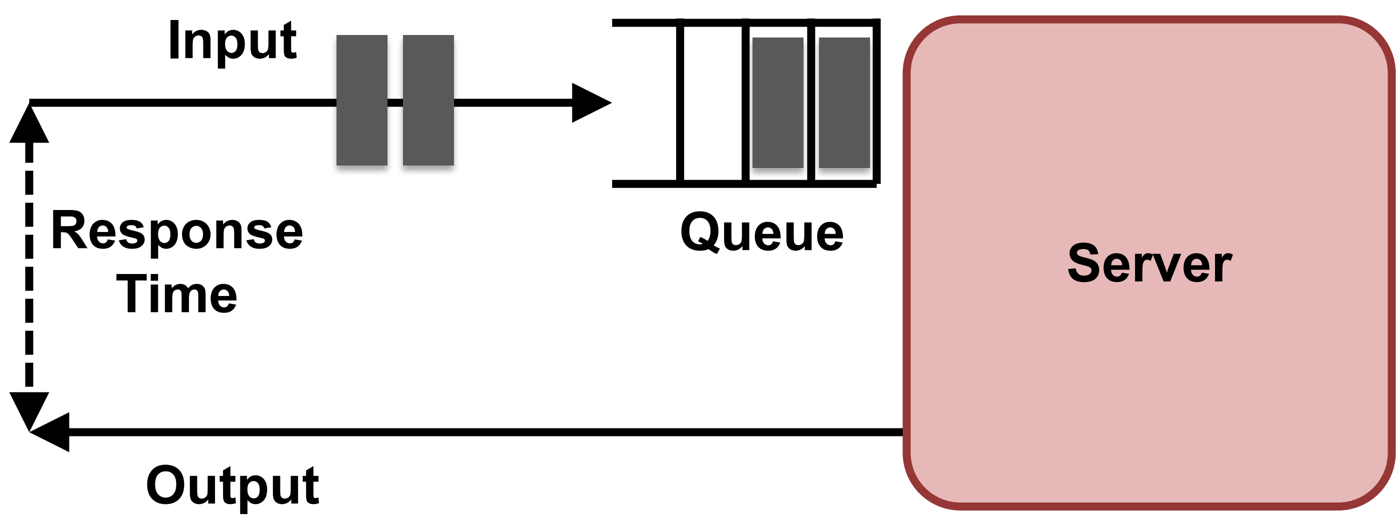


Some terms

- **Workload**
  - The input to the system or load applied
- **Utilization**
  - A measure of how busy a resource is
  - The capacity consumed (for a capacity-based resource)
- **Saturation**
  - The degree to which a resource has queued work it cannot service
- **Bottleneck**
  - A resource that limits the system performance

More terms

- **Response time (also latency at times)**
  - The time for an operation to complete
  - Includes any time spent waiting (**queuing time**) and time spent being serviced (**service time**), and time to transfer the result



Who in interested?

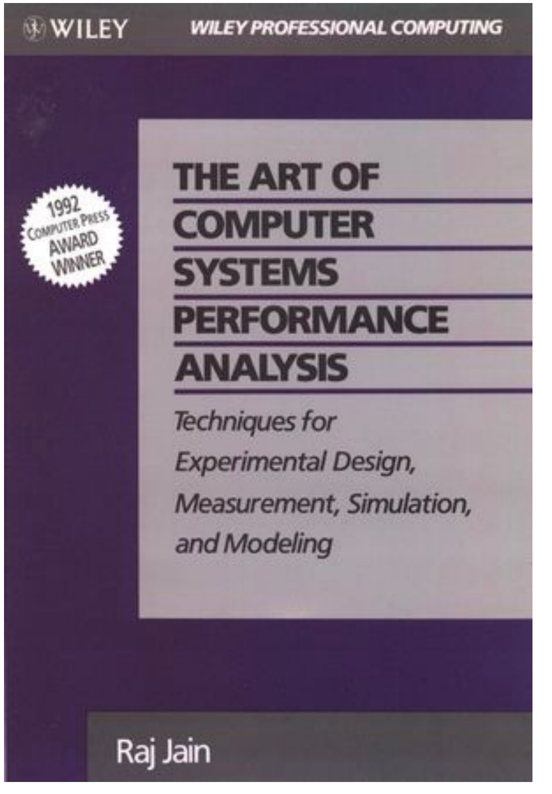
- Many roles:
  - Sys admins / capacity planners
  - Support staff
  - Application developers
  - DB / Web admins
  - Researchers
  - Performance engineers (primary activity)

Performance evaluation is an art

- Like a work of art, a successful evaluation cannot be produced mechanically

- Every evaluation requires an intimate knowledge of the system and a careful selection of methodology, workloads and tools

- **Performance is challenging**



Performance is subjective

- Is there an issue to begin with? If so, when is it considered fixed?

- Consider:
  - The average disk I/O response time is 1 ms
- Is this **good** or **bad**?

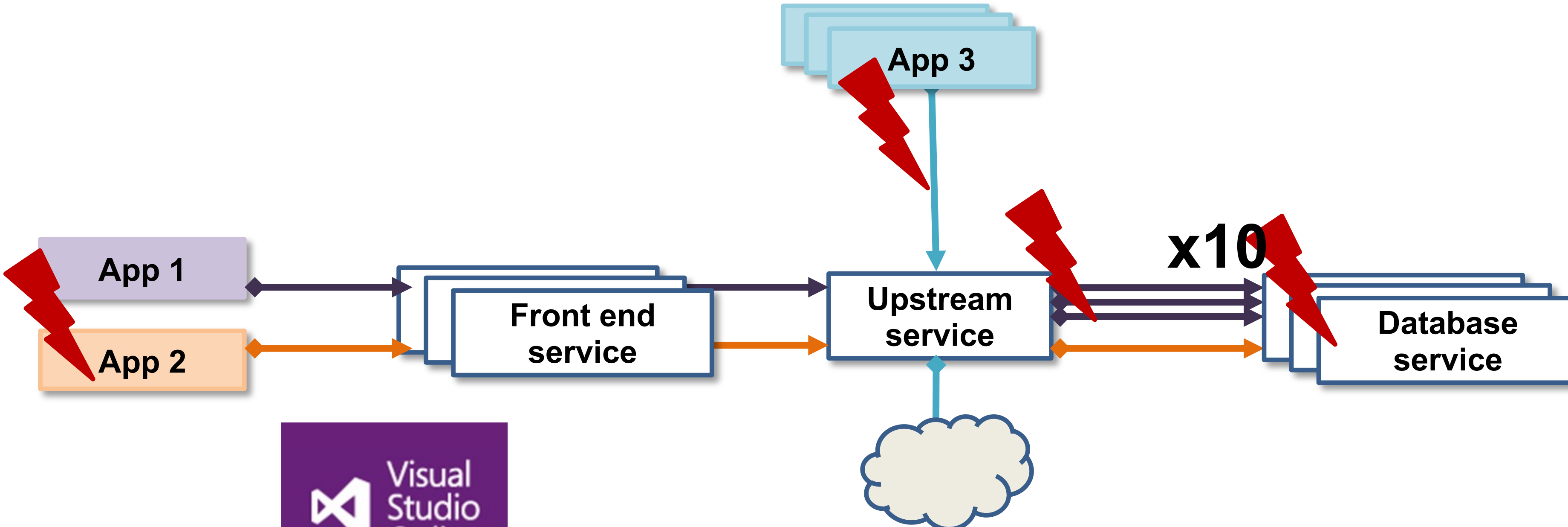
- Response time is one of the best metrics to quantify performance; the difficulty is **interpreting** its information

- Performance objectives and goals need to be clear
  - Orient expectations as well as choice of techniques, tools, metrics and workloads

Systems are complex

- Many components and sources of root causes
- Issues may arise from complex interactions between subsystems that operate well in isolation
  - Cascading failures: when one failed component causes performance issues in others
- Bottlenecks may be complex and related in unexpected ways
  - Fixing one may simply move the bottleneck elsewhere
- Issue may be caused by characteristics of workload that are hard to reproduce in isolation
- Solving complex issues often require a holistic approach
  - The whole system needs to be investigated

Example of cascading failure



Visual Studio Online

August 2014 outage

- One request type was accessing a single slow database and exhausted an upstream service's thread pool
- This starved other unrelated requests… causing widespread application unavailability

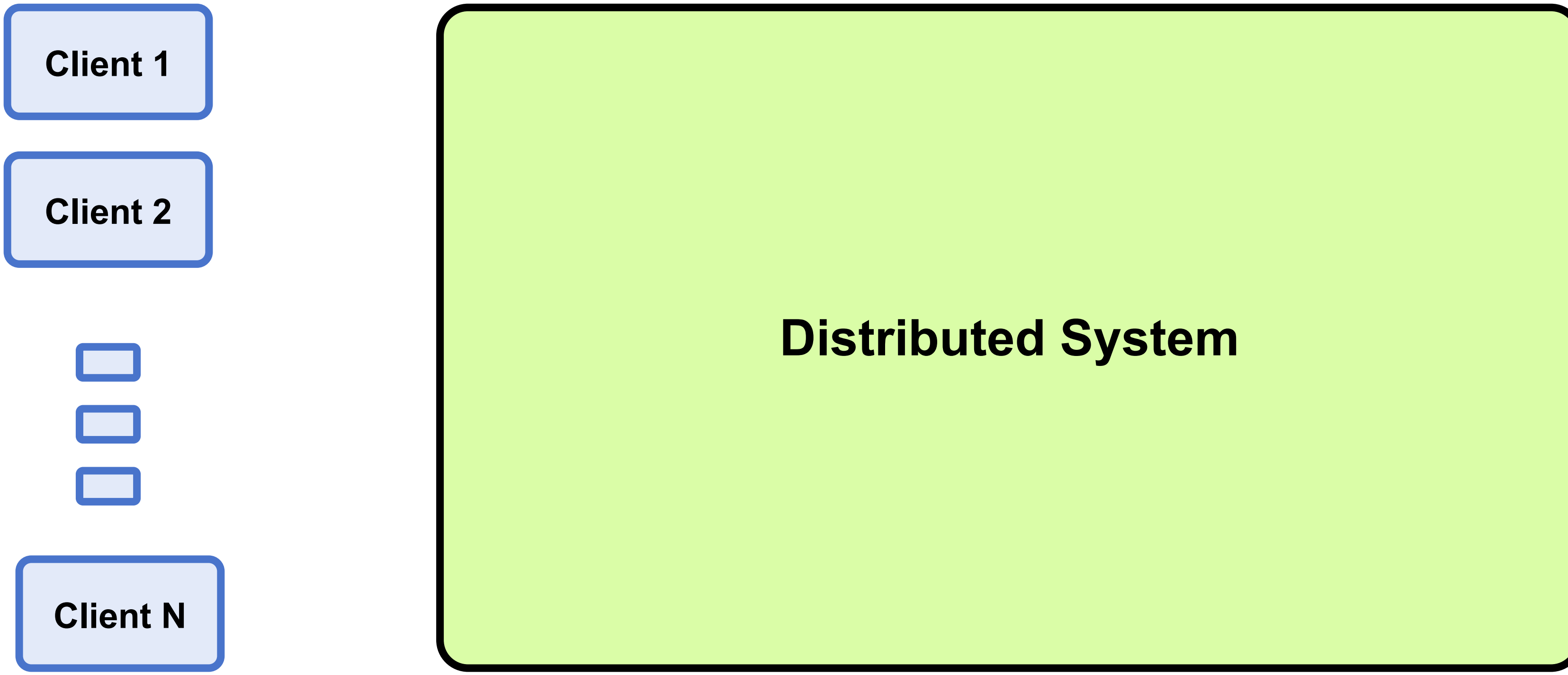
Measurement is crucial

- **You can’t optimize what you don’t know**
- Must quantify the magnitude of issues
- Measuring an existing system helps to see its performance and perhaps the room for possible improvements

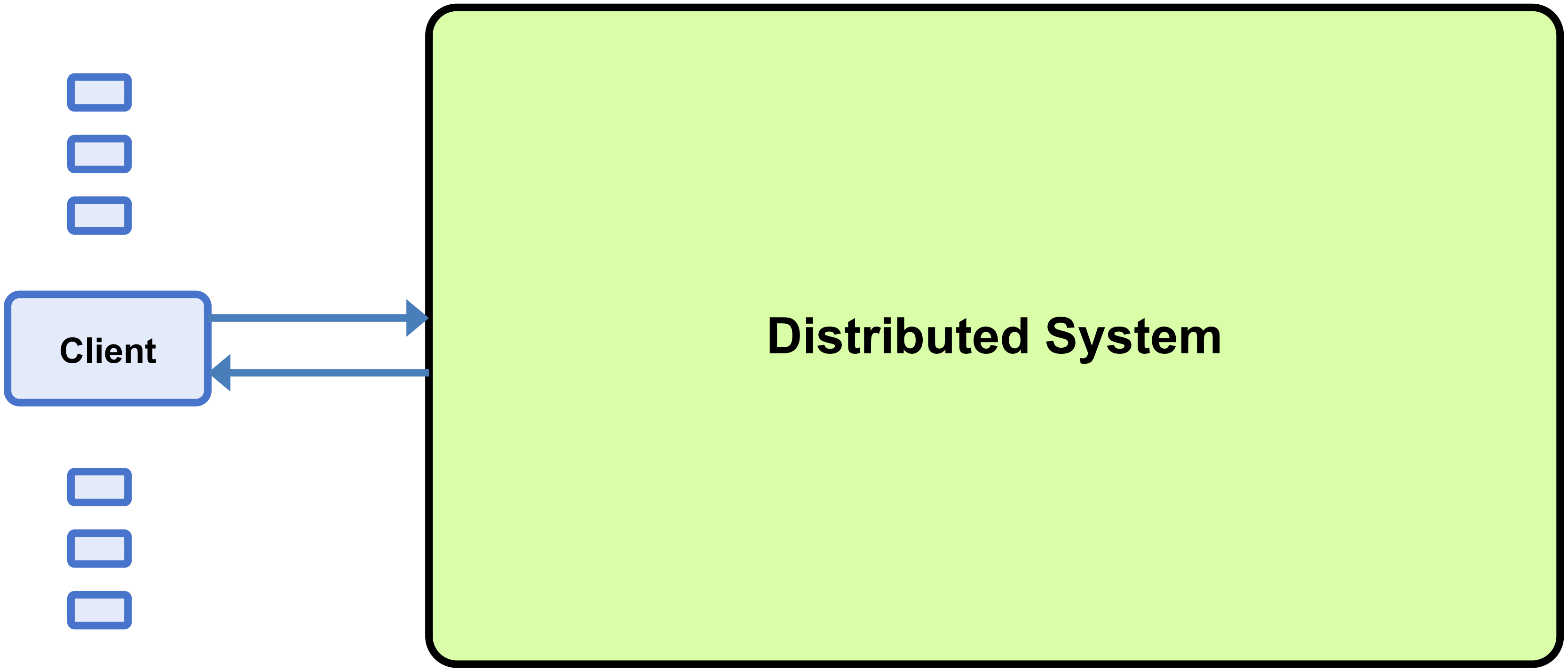
- **Need to define metrics**

- Know your tools!
- Be systematic!
- Don’t reinvent the wheel!

Measuring Distributed Systems



Measuring Distributed Systems



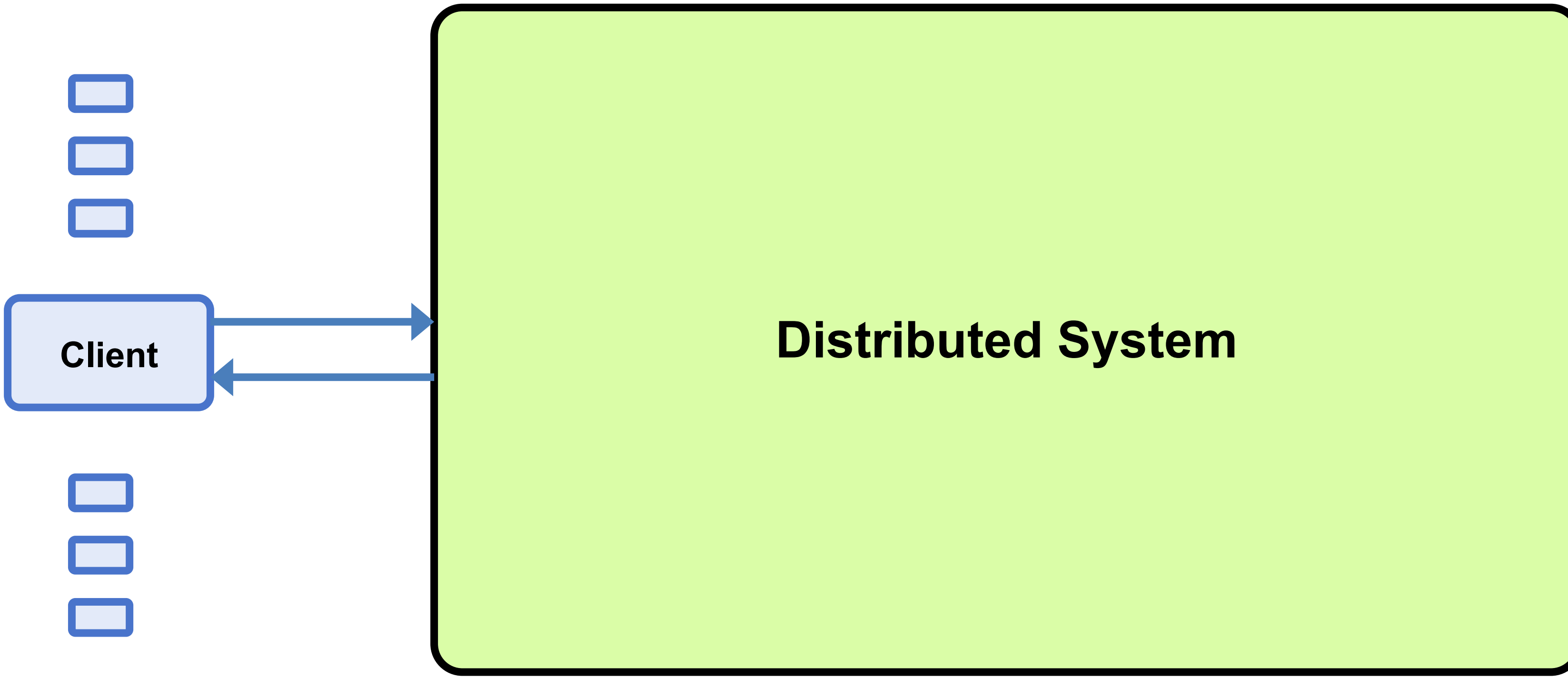
Latency

- The time spent waiting
  - E.g., setup a network connection

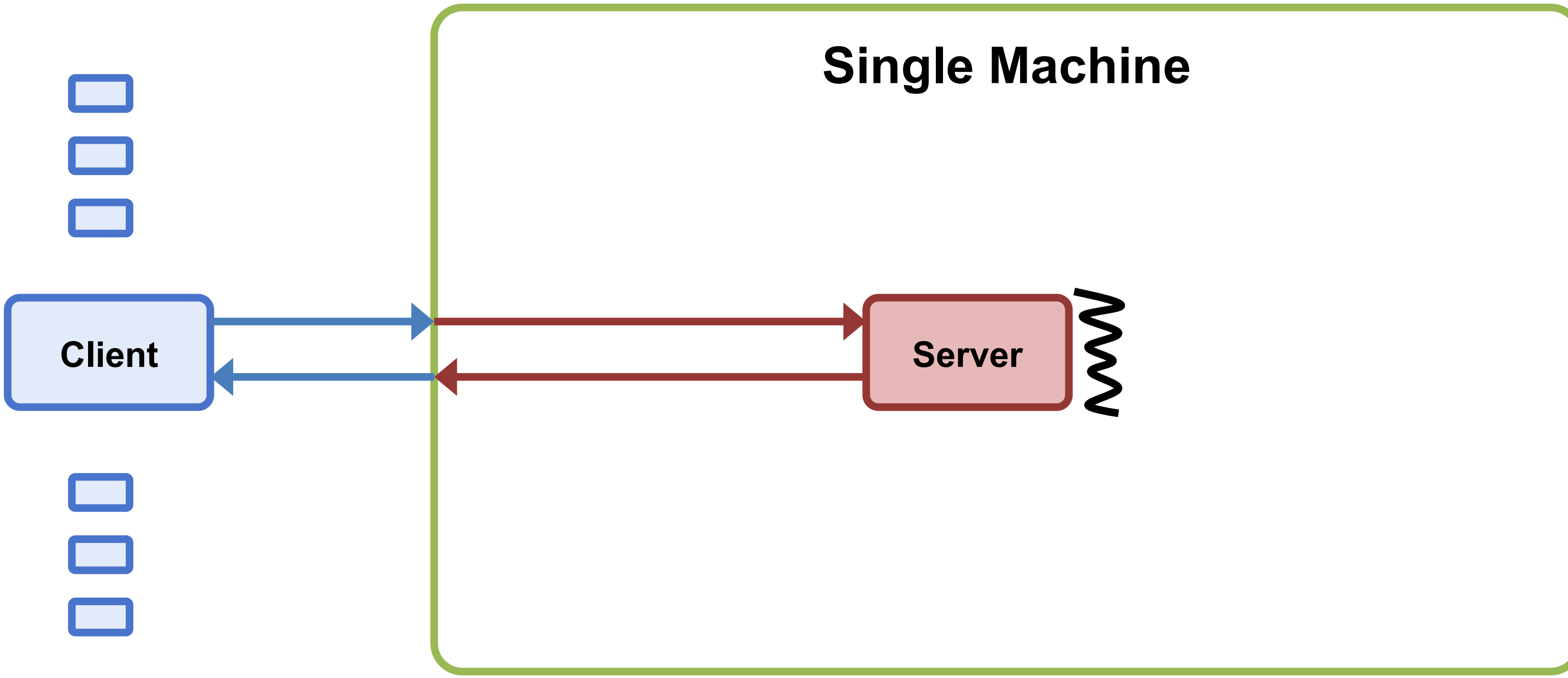
OR (broadly)

- The time for a request/operation to complete
  - E.g., an RPC, data transfer over the network, a DB query, a file system write
- Measured externally from time request is sent until time response is received
- Can allow to estimate maximum speedup
  - E.g., assume the network had infinite capacity and transfer were instantaneous, how fast would the system go?

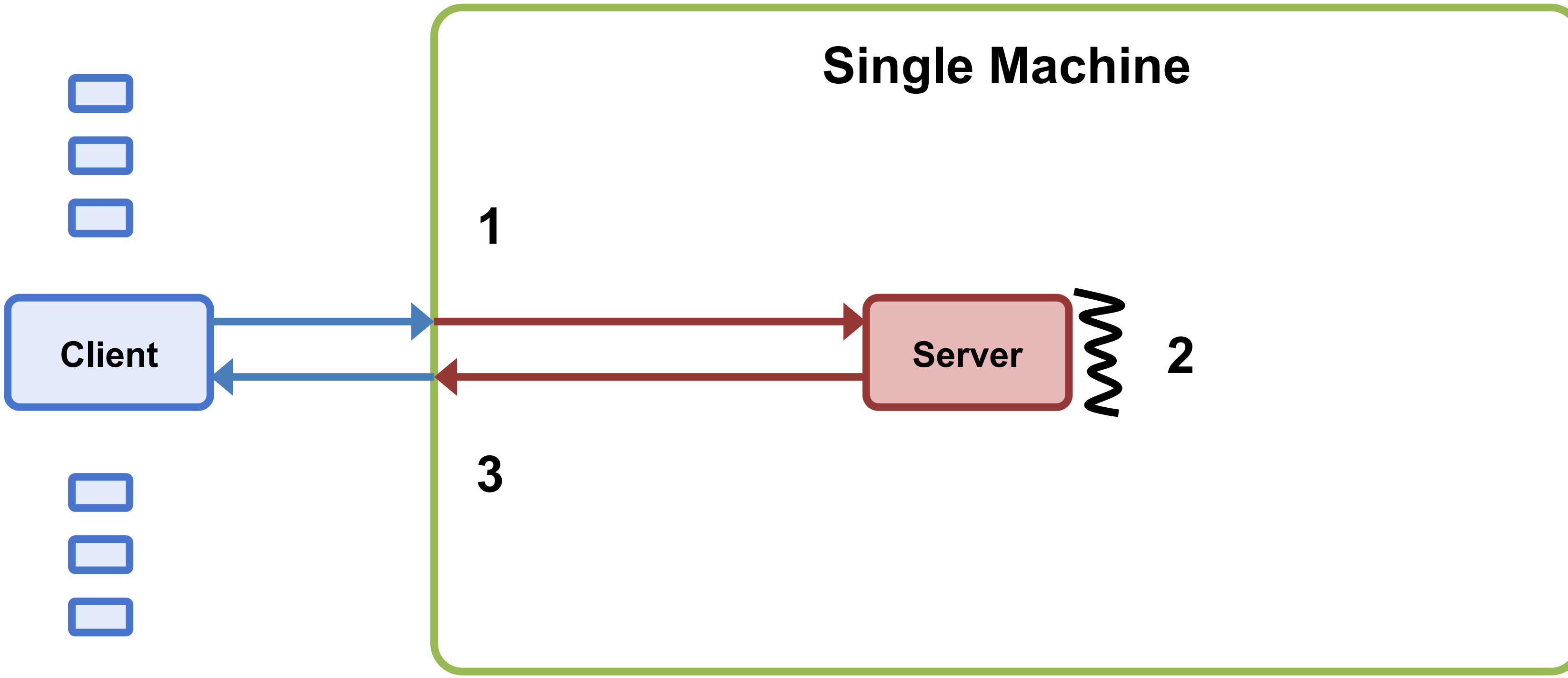
Latency, Measure Externally



Latency, Reason Internally



Latency, Reason Internally

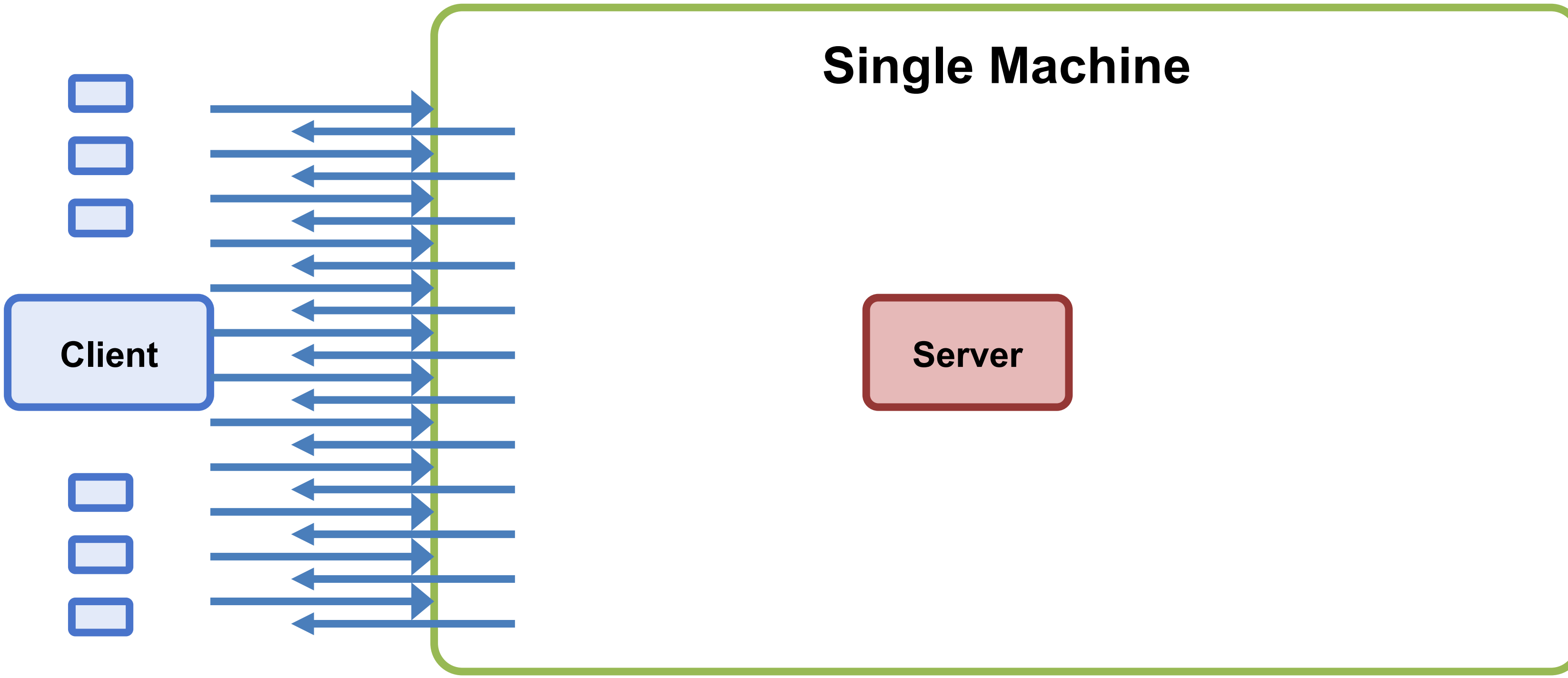


Latency = 1 + 2 + 3

Throughput

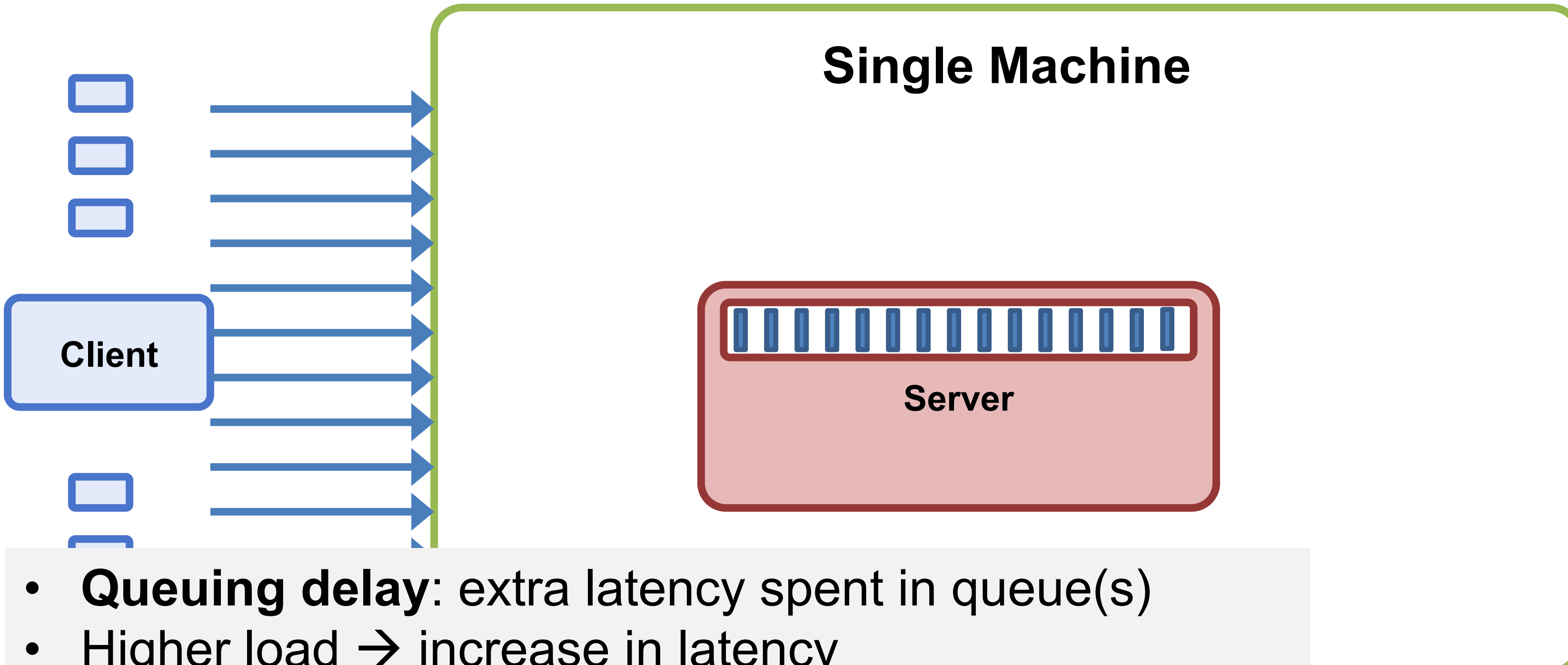
- The rate of work performed: how many operations per unit time (ops/s) a system can handle
  - In communication:
    - Data rate: bytes per second, bits per second
    - (Goodput useful throughput: rate for the payload only)
  - Systems:
    - Operation rate: ops per second, txns, per second
  - IOPS
    - Input/output operations per second
      - E.g., reads and writes to disk per second
- Measured externally as the rate that responses come out of the system

Max Throughput Example (Not Ideal)



Throughput = $\dfrac{\text{Number of (valid) responses received by all clients}}{\text{End time – start time}}$

Queuing Delay & Overload



- **Queuing delay**: extra latency spent in queue(s)
- Higher load → increase in latency

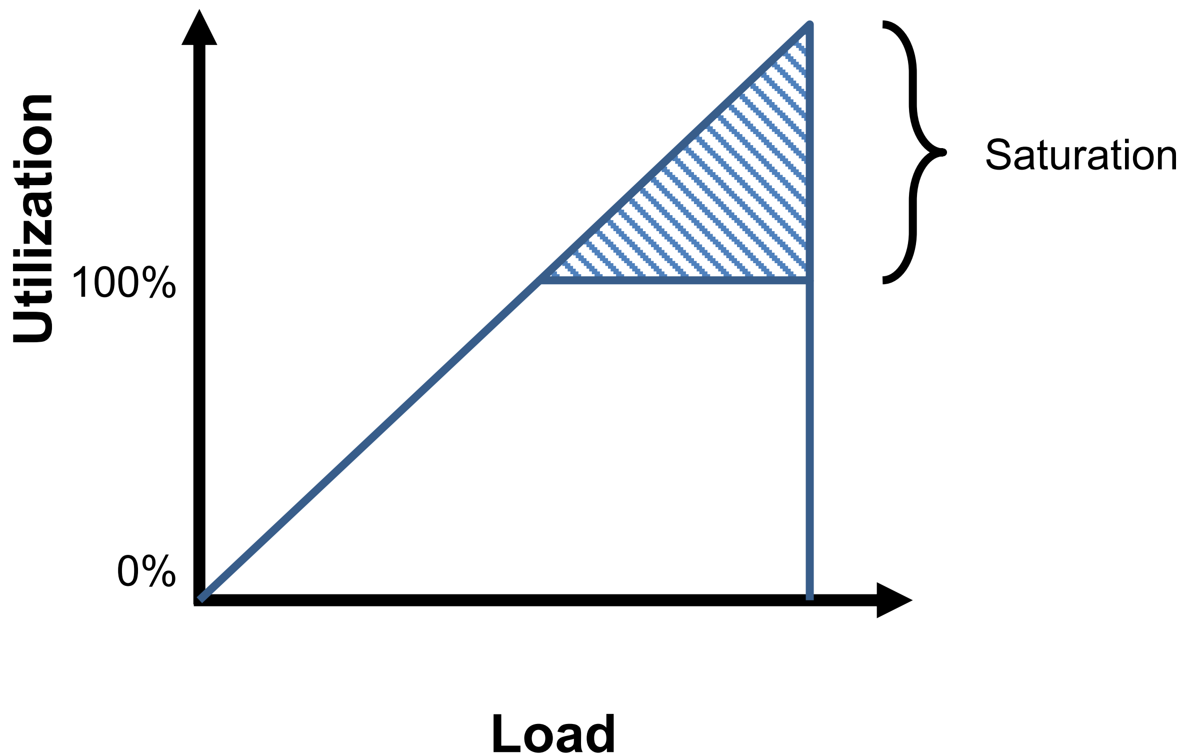
- **Overload**: offered load > max system throughput
    - Queues get really long
    - Other weird/bad things happen
    - →Observed throughput < max system throughput

Utilization, Saturation

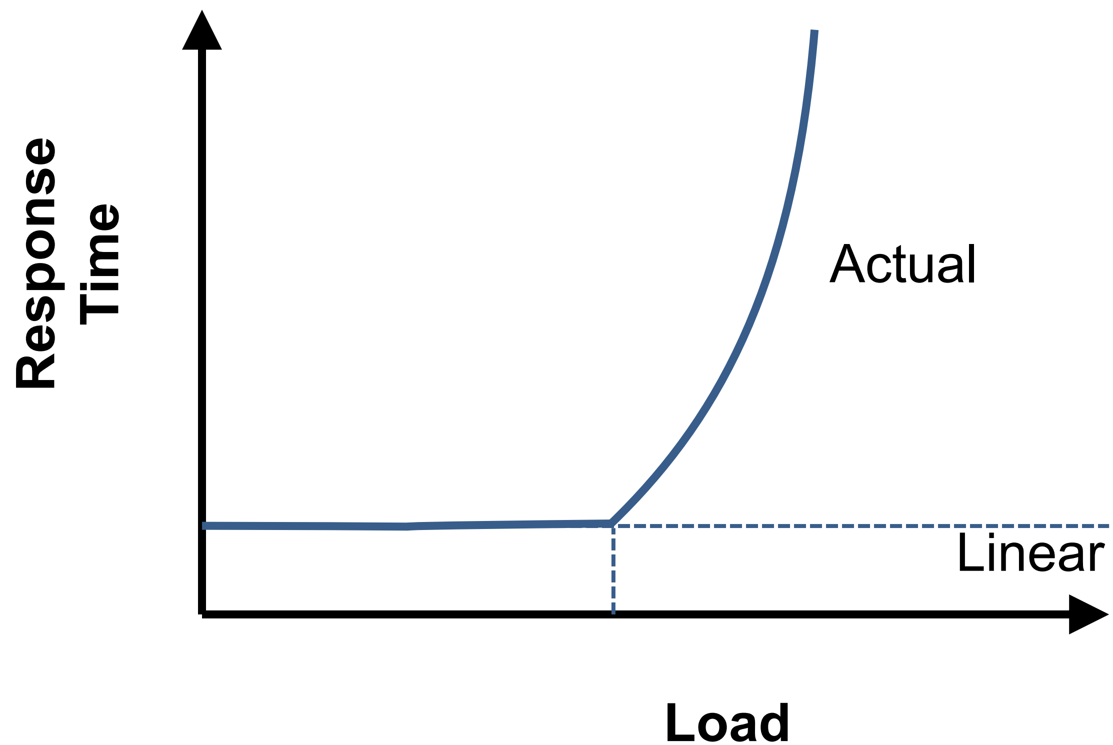
Utilization (time-based) = B/T

B is amount of time the resource was busy during T

Intuitively, how busy a component is



Performance degradation



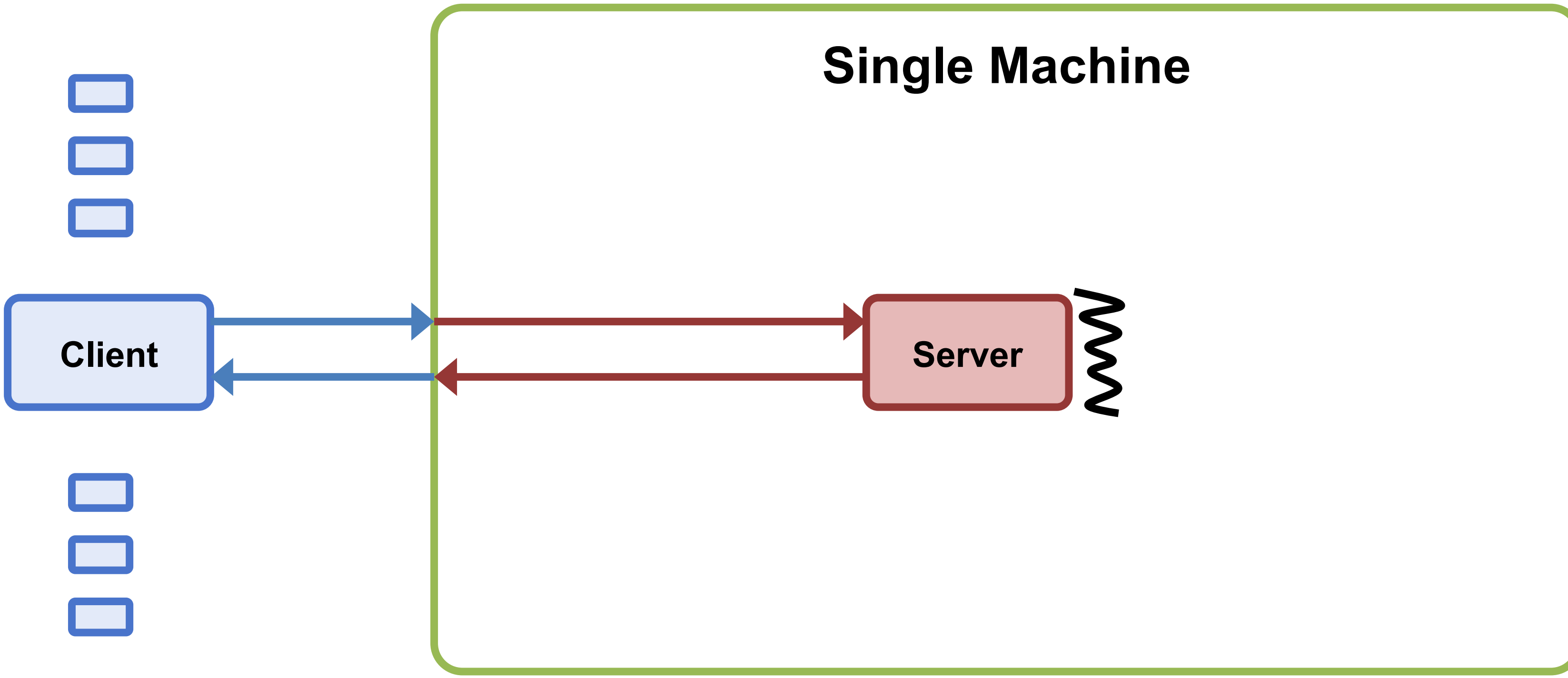
Measuring Throughput Method

1. Starting with low load

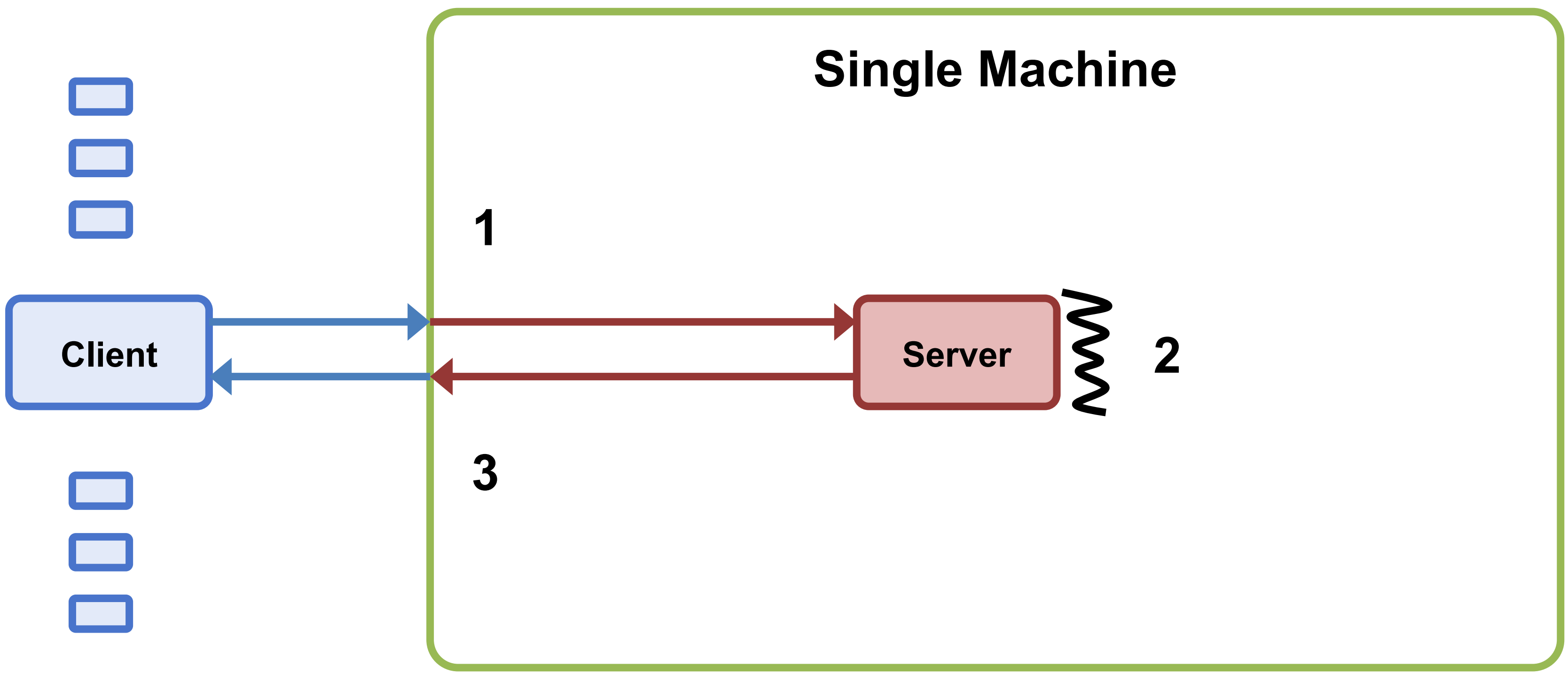
2. Increase load

3. Repeat until measured throughput stops increasing

Throughput, Reason Internally

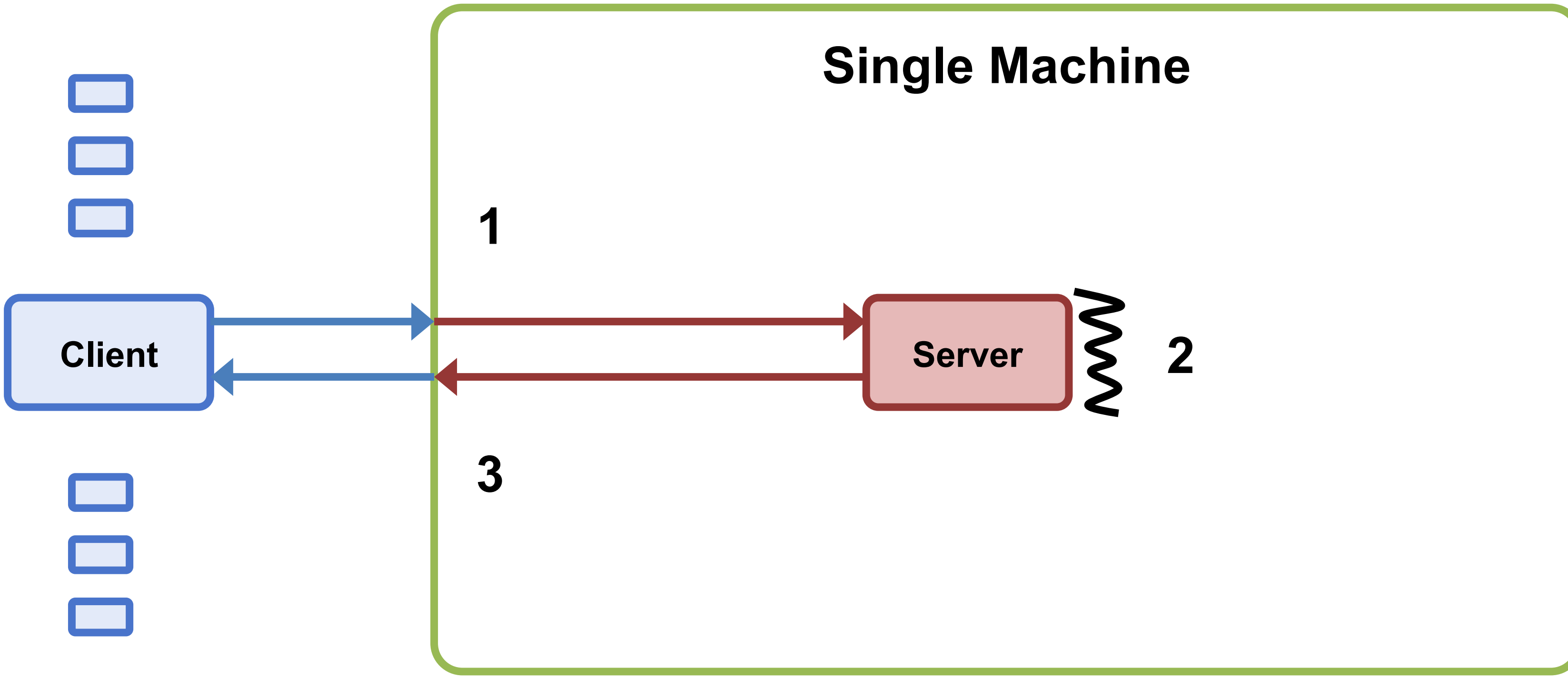


Throughput, Reason Internally



Throughput = min(1, 2, 3)

Throughput Bottlenecks (simplified)



Max throughput limited by some bottleneck resource:

1) Incoming bandwidth
2) Server CPU
3) Outgoing bandwidth

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

Mental Experimental Setup

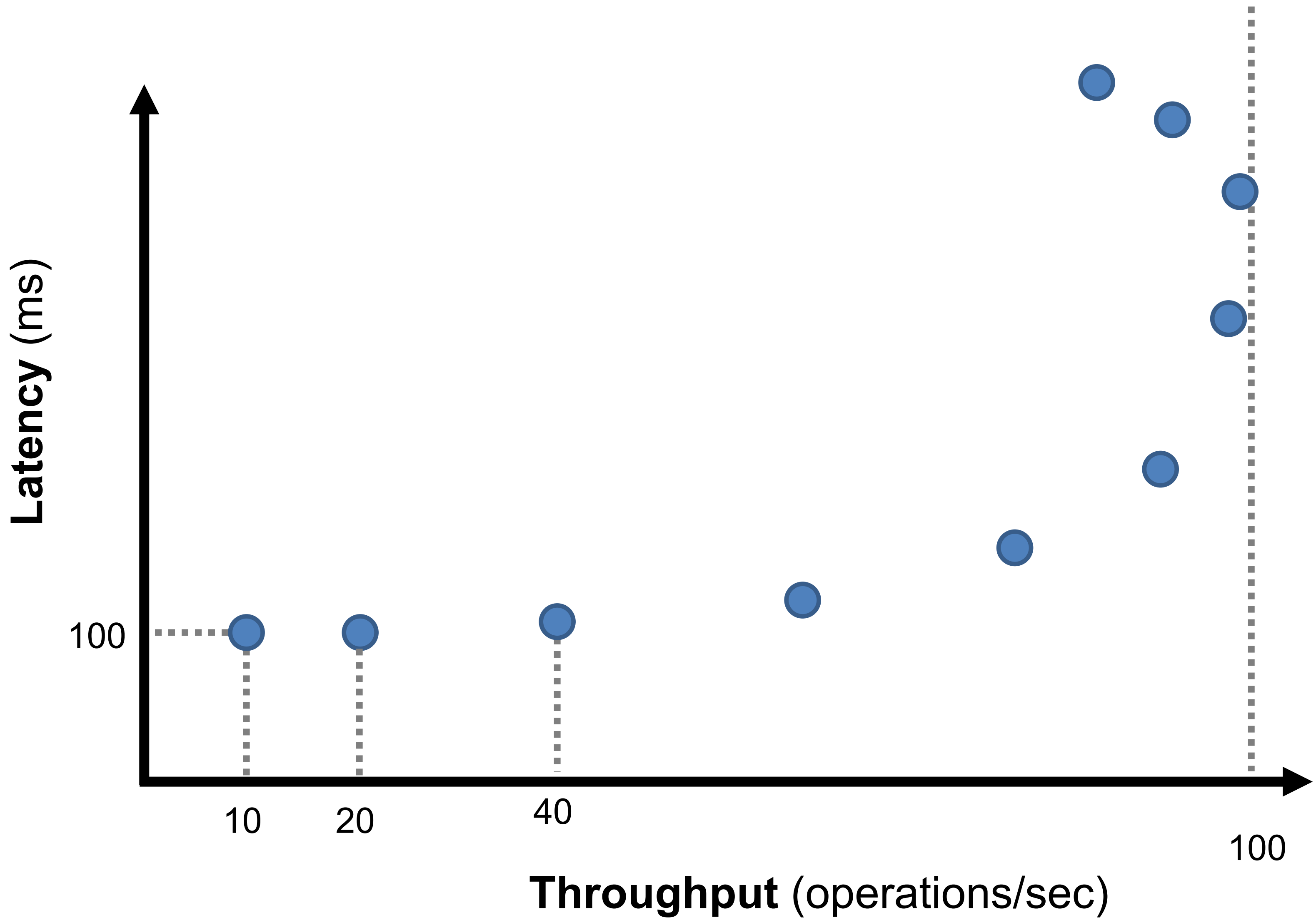
- Start with 1 closed-loop client
  - Expected latency?
  - Expected throughput?

- Double number of closed-loop clients
  - Expected increase in latency?
  - Expected increase in throughput?

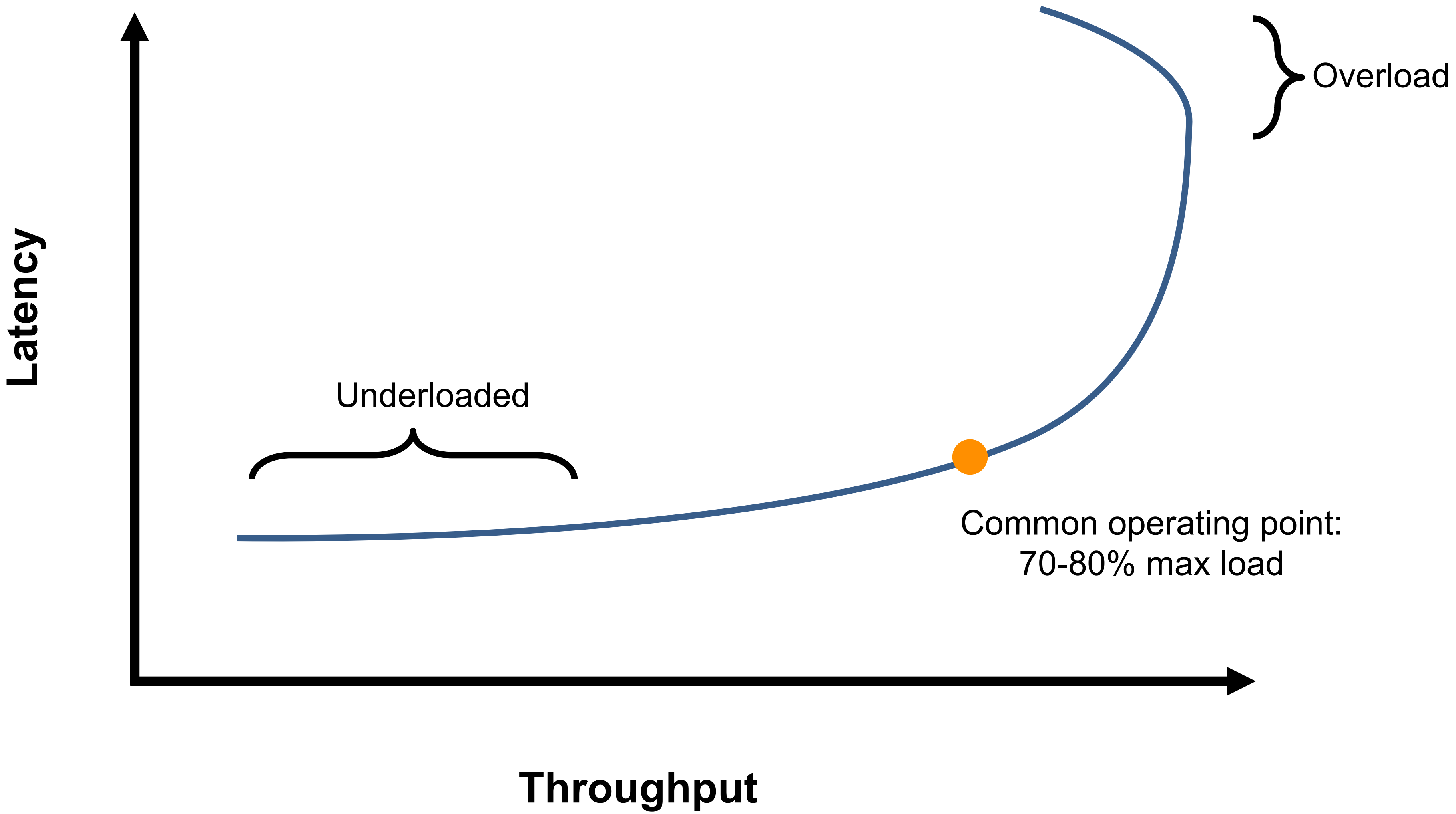
- Repeat

Throughput-Latency Graph

Simple Setting: Single Server; Client-Server RTT 90ms;
Server Processing latency 10ms; Single-Threaded Server (100 ops/s)



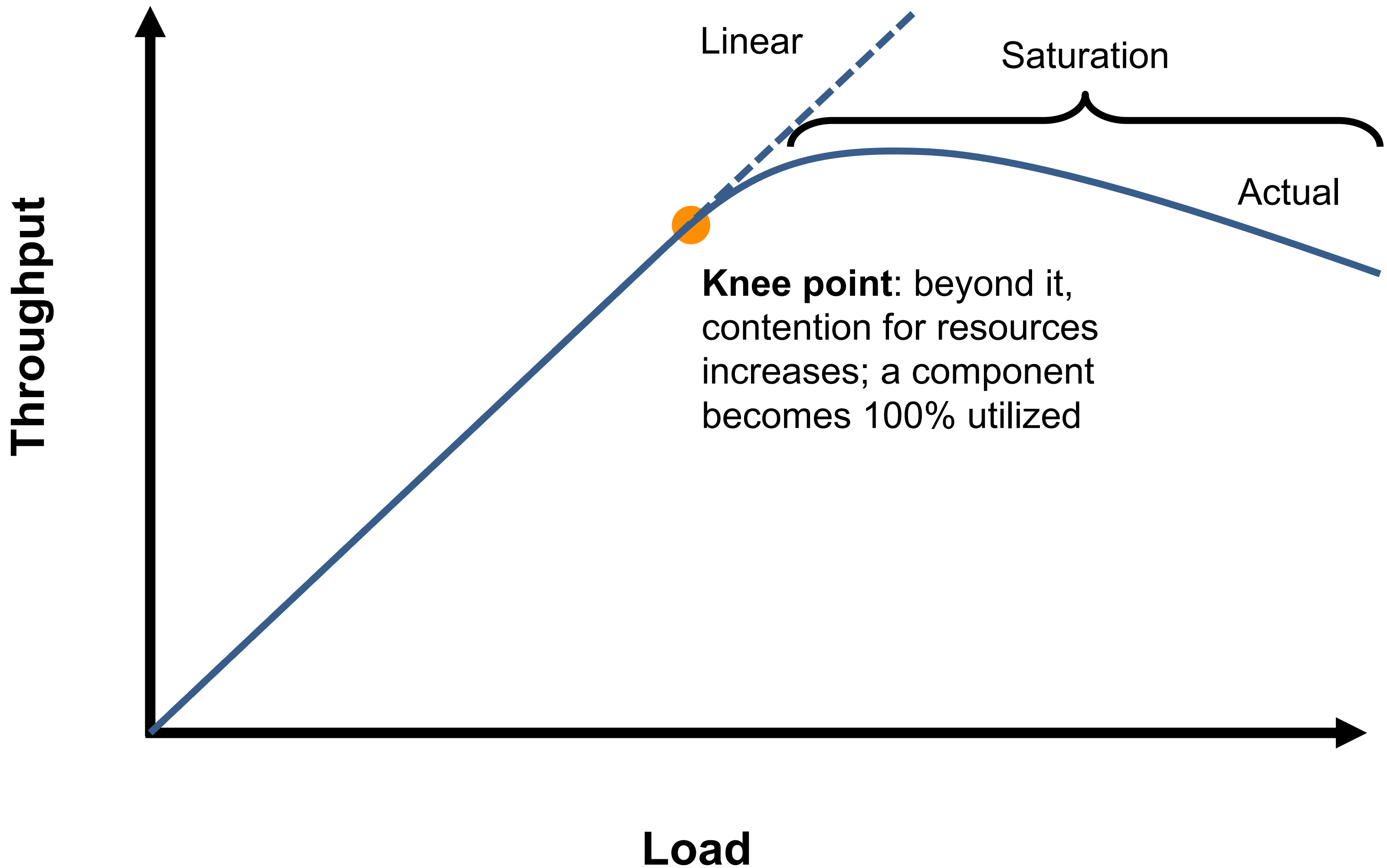
Throughput-Latency Graph



Throughput / Latency Relationship

- Proportional at low load … but not high load
- Because measured throughput is a function of latency
  - i.e., throughput bottleneck is offered load
- Related, but you should reason about both
- For system A vs system B, all are possible:
  - A has lower latency and higher throughput than B
  - A has lower latency and lower throughput than B
  - A has higher latency and lower throughput than B
  - A has higher latency and higher throughput than B

Scalability



Evaluation in Minutes not Months

- Reasoning using your mental model is much much faster than really doing it

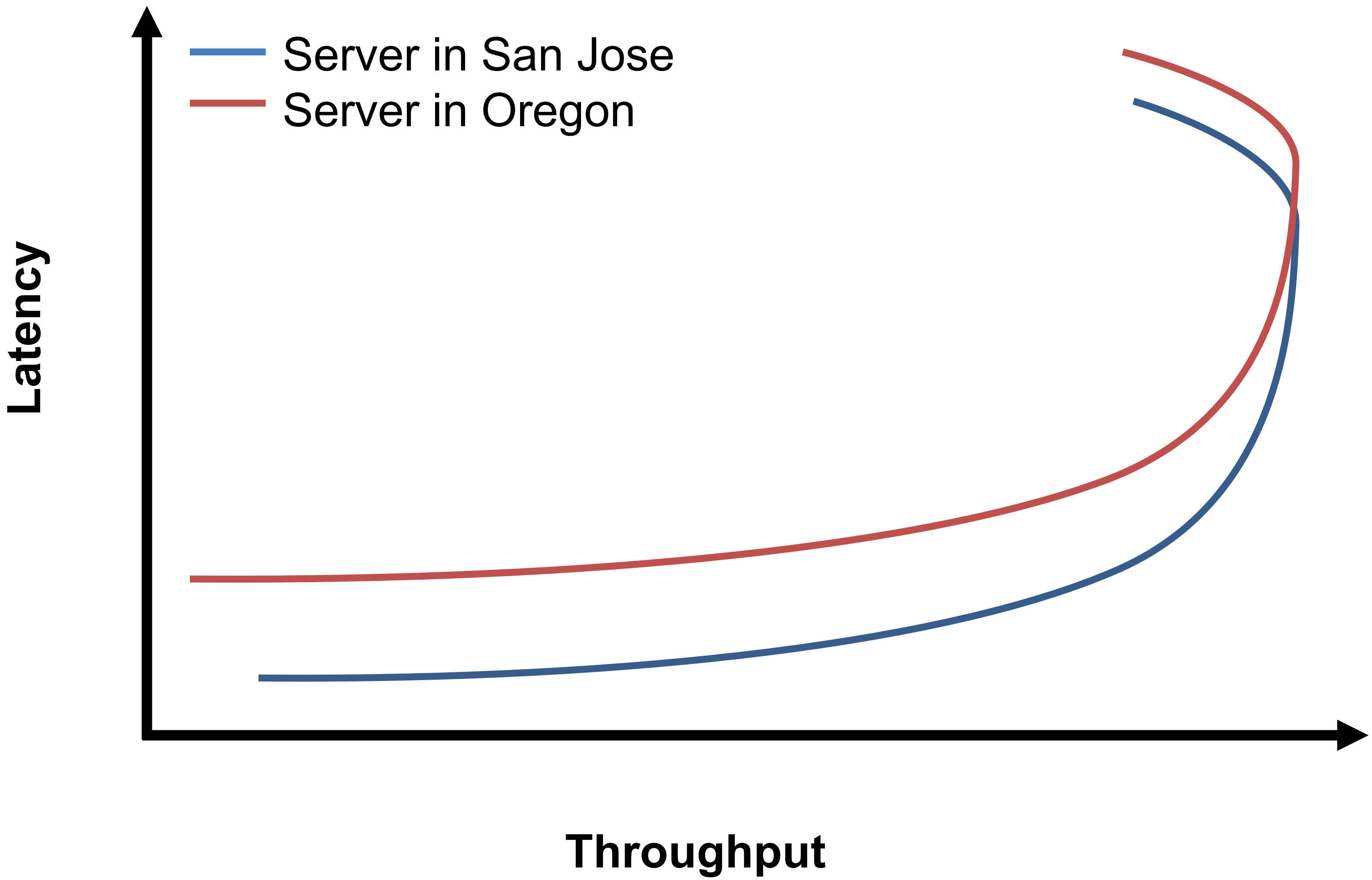
- What would happen if?
  - I moved my servers from the San Jose datacenter to Oregon?
  - I switch from c5.xlarges to c5.24xlarges for my servers?
  - I doubled the number of servers?
  - I switch from system design X to system design Y?
    - replace single server with Paxos-replicated system?
    - replace Paxos with eventually consistent design?
    - add batching?
    - replace Paxos with new variant?

Let’s use these tools!

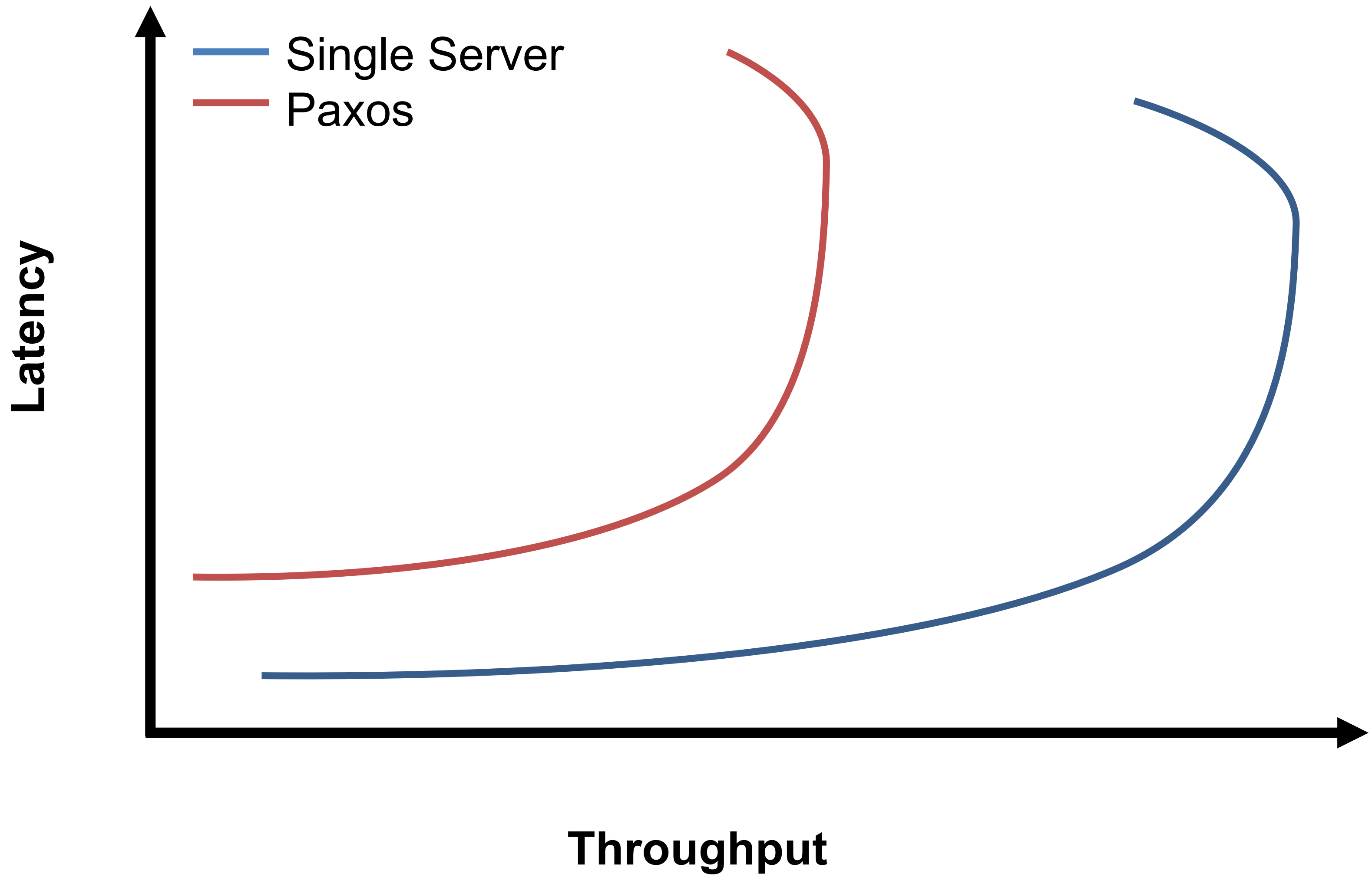
Mental Experimental Setup

- System A versus System B
- From 1 to N closed-loop clients loading each
- Compare throughput and latency

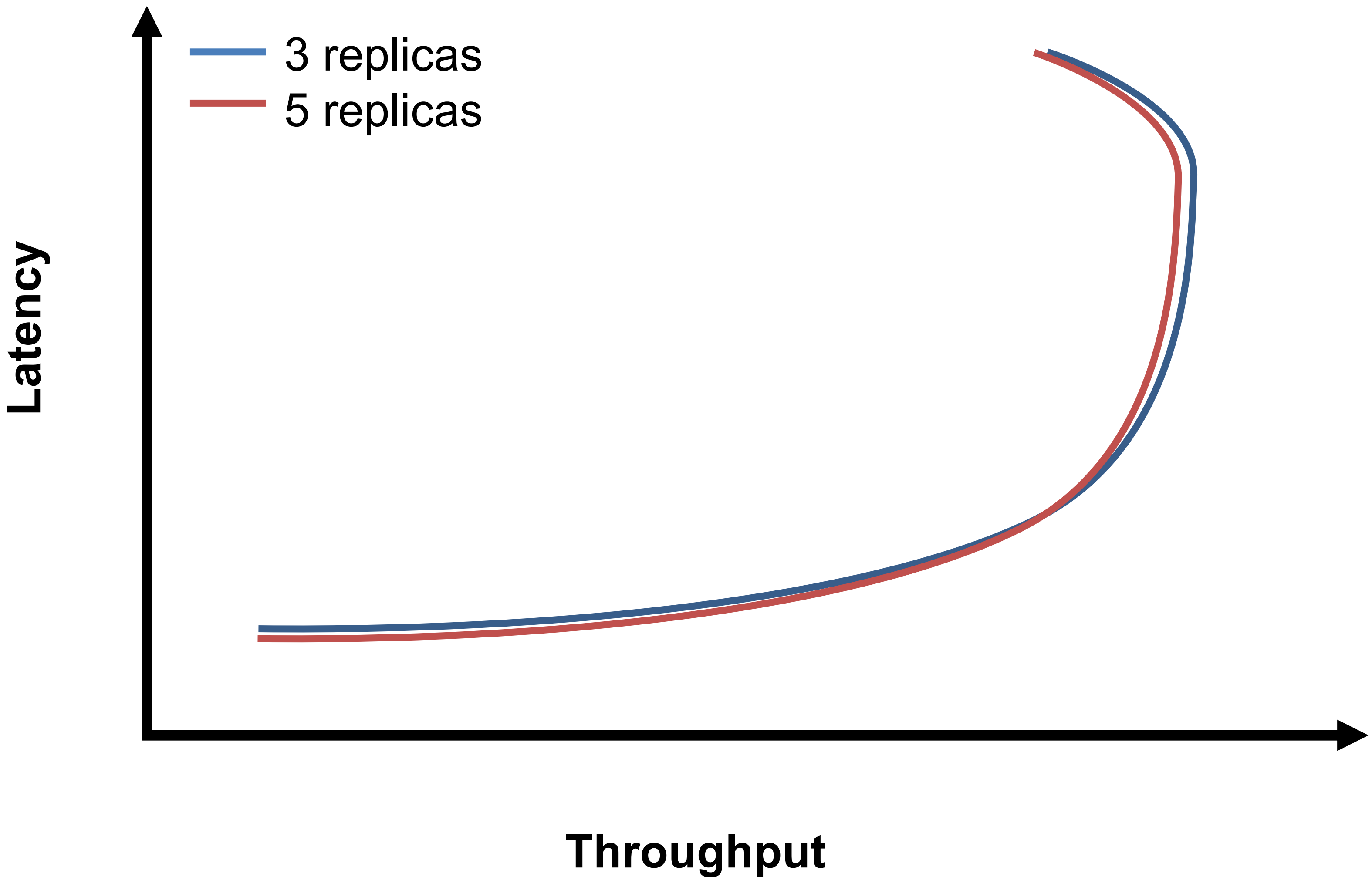
Move Single Server from San Jose to Oregon
(Clients in San Jose)



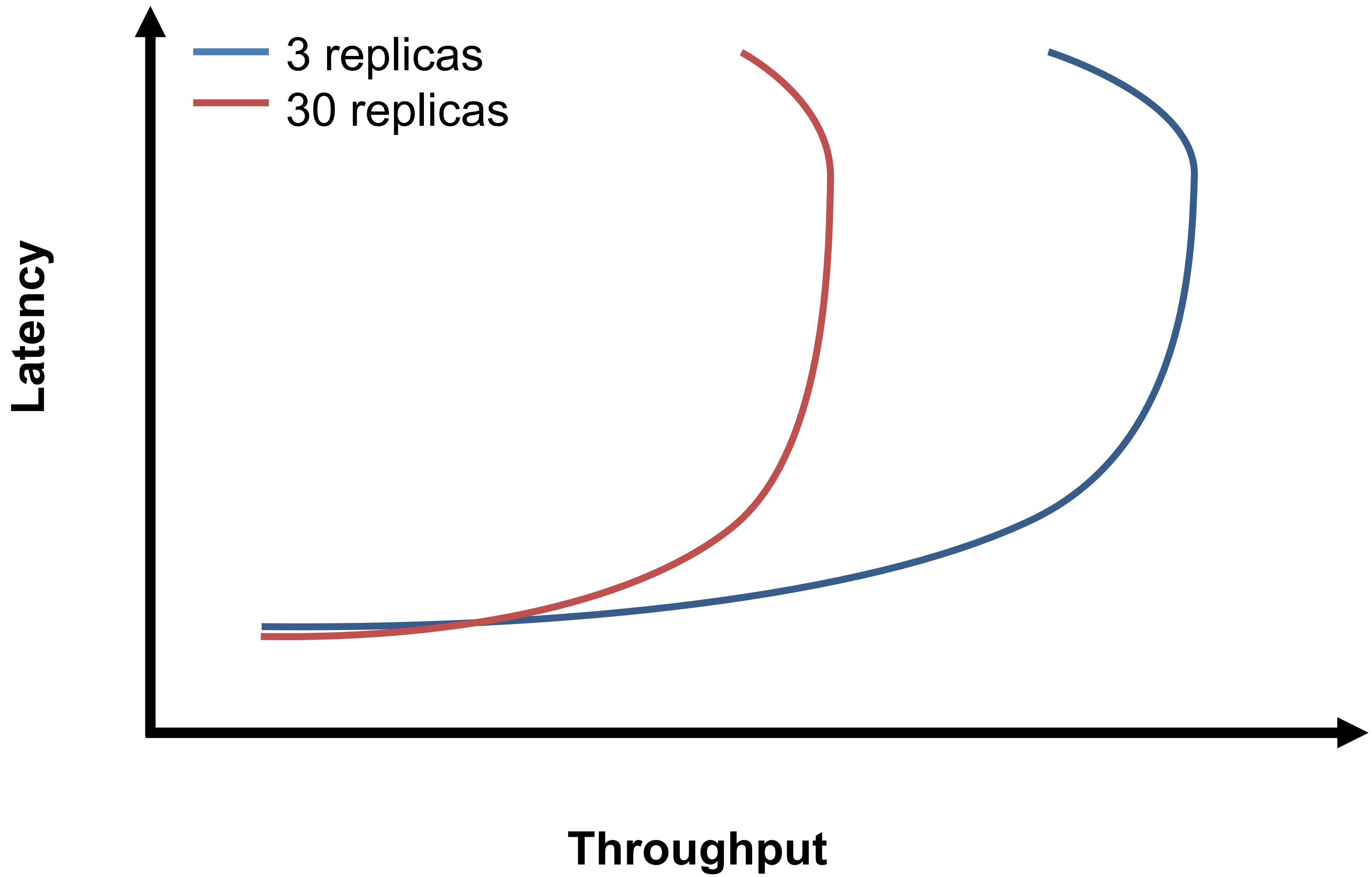
Replace Single Server with Paxos
(Clients and servers in same datacenter, 3 replicas)



Paxos: 3 replicas to 5 replicas
(Clients and servers in same datacenter)



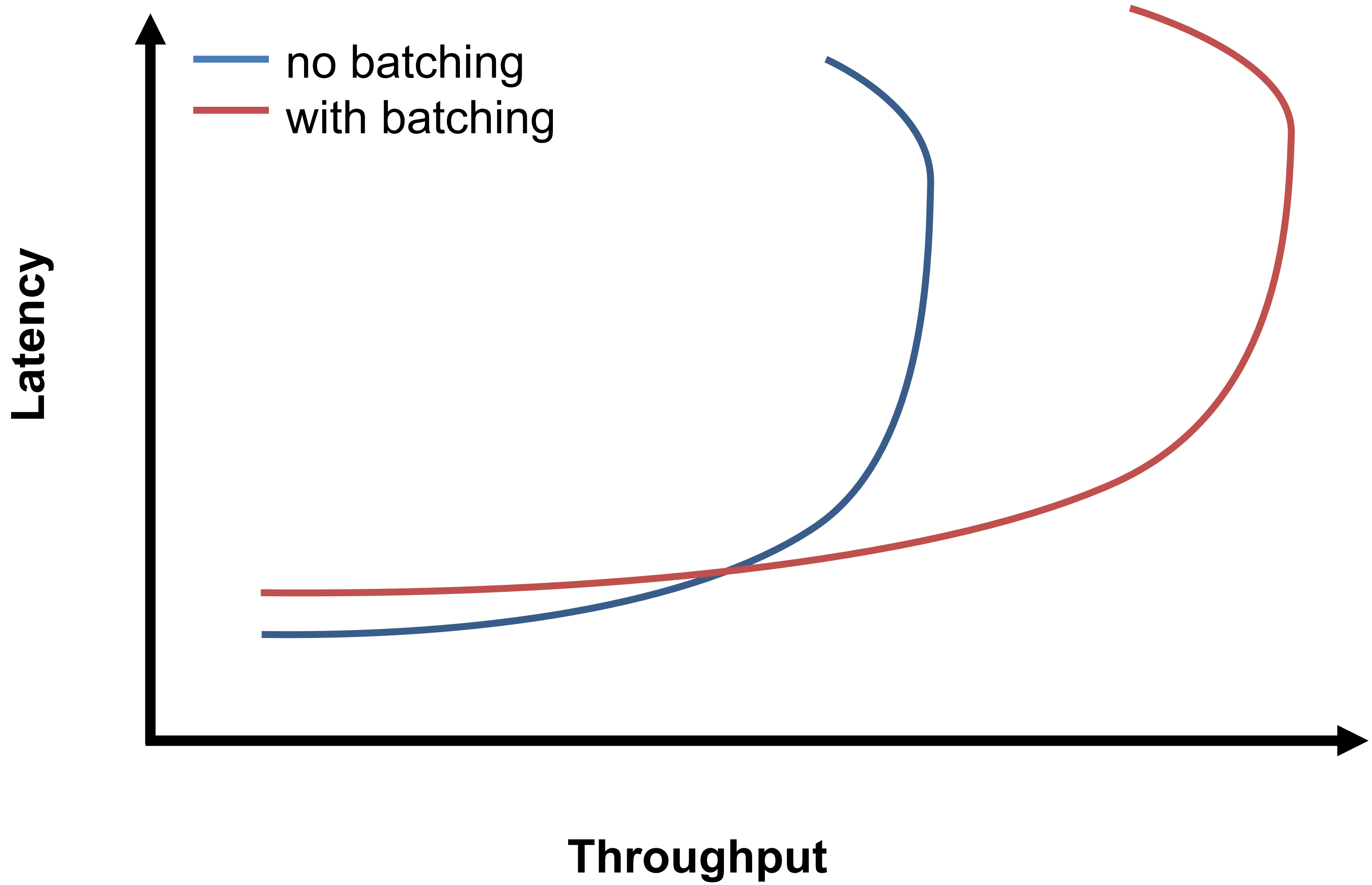
Paxos: 3 replicas to 30 replicas
(Clients and servers in same datacenter)



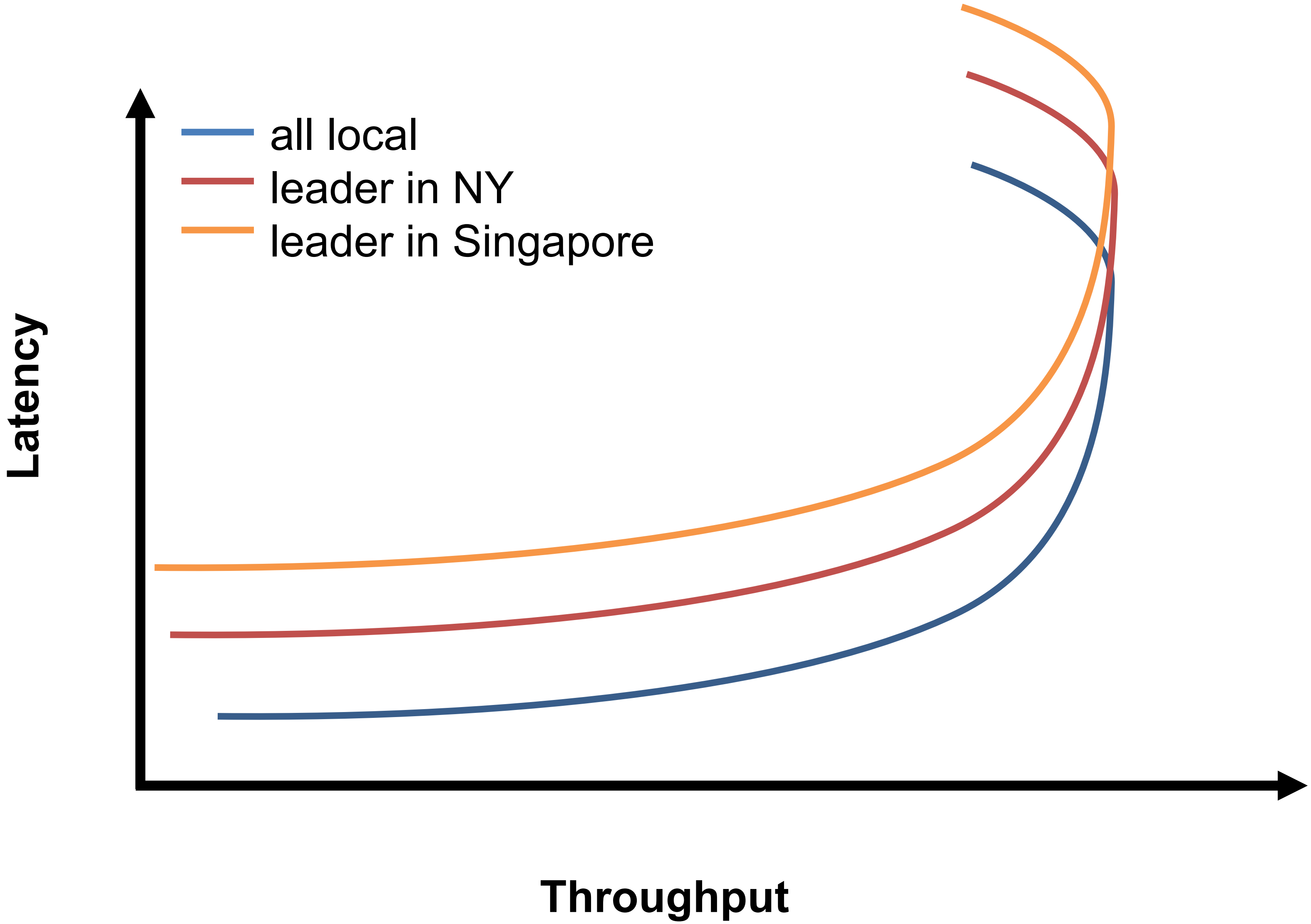
Batching

- Group together multiple operations
- Improves throughput, e.g.,
  - Marshall data together
  - Send to network layer together
  - Unmarshall data together
  - Handle group of operations together
- Delay processing/sending operations to increase batch size
  - Common way to trade an increase in latency for increase in throughput

Paxos with batching
(Clients and servers in same datacenter, 3 replicas)



Paxos: 3 local replicas to geo-replicated
(Clients in NY; replicas in NY, Oregon, Singapore



Summary

- Measure distributed systems externally

- Latency: how long operations take
- Throughput: how many operations/sec

- Reason about latency and throughput using internal knowledge of system design
  – (and back-of-the-envelope calculations)

- Reason about effects on latency and throughput from changes to system choice, deployment, design
  – Critical tool in system design

Five ways not to fool yourself or: designing experiments for understanding performance

Tim Harris

https://timharris.uk/misc/five-ways.pdf

Measure as you go

- Develop good test harness for running experiments **early**
- Have scripts for plotting results
- Automate as much as possible
  - Ideally it is a single click process!
- Divide experimental data from plot data

Gain confidence (and understanding)

- Plot what you measure
- Be careful about trade-offs
- Beware of averages
- Check experiments are reproducible

- (Also statistics! Deal with outliers, repetitions)

Include lightweight sanity checks

- It’s easy for things to go wrong… and without noticing…
- Make sure you catch problems
- Have sufficiently cheap checks to leave on in all runs
- Have sanity checks at the end of a run
- And don’t output results if any problem occurs

Understand simple cases first

- Start with simple settings and check the system behaves as expected
- Be in control of sources of uncertainty to the largest extent possible
  - And use checks to detect if that assumption does not hold
- Simplify workloads and make sure experiments are long enough
- Use these as a performance regression test for the future

Look beyond timing

- End to end improvements are great but are they happening because of your optimization?
- Try to link differences in workloads with performance
- Look further into differences in resource utilization and statistics from performance counters

Toward production setting

- Do observations made in simple controlled settings hold in more complex environments?
- If that is not true, try to decouple a number of aspects of this problem
- Change one factor at a time
- Try to understand the differences

Document results

- You will forget!
  - What did that experiment produce?
  - Where did I see that result?
- Pick a good convention to save data
- Use non destructive approaches
- Write summary of observations and possible explanations
  - Recall: our objective is better understanding
- Pick a good tool for experimenting, documenting and sharing
  - Try Jupyter

That's all Folks!

Final exam: Wed 12/14, 8:00-10:50AM , 9-4223

Please fill in course feedback