CSE 124 AND CSE 224:

PERFORMANCE, PERFORMANCE METRICS, AND TAIL LATENCY

George Porter
April 17, 2025

UC San Diego

ATTRIBUTION

- These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license
- These slides incorporate material from:
  - Computer Networks: A Systems Approach, 5e, by Peterson and Davie
  - Michael Freedman and Kyle Jamieson, Princeton University (also under a CC BY-NC-SA 3.0 Creative Commons license)

Today's agenda

- Performance metrics
- The problem of tail latency
- Case study: Memcached

PERFORMANCE METRICS

- Bandwidth: number of bits transmitted per unit of time in a channel
- Latency = Propagation + Transmit + Queue
  - Propagation = Distance/SpeedOfLight(*)
  - Transmit = 1 bit/Bandwidth
  - Queue = Time waiting in switches/routers behind other traffic (traffic jam)
- Overhead
  - # secs for CPU to put message on wire
- Error rate
  - Probability P that message will not arrive intact

* In that particular medium

TOTAL LATENCY GIVEN PROPAGATION AND BW

1 Byte Object

	Propagation: 1 ms	Propagation: 100 ms
Bandwidth: 1 Mbps	1,008 µs	100,008 µs
Bandwidth: 100 Mbps	1,000 µs	100,000 µs

10 MB Object

	Propagation: 1 ms	Propagation: 100 ms
Bandwidth: 1 Mbps	80.001 s	80.1 s
Bandwidth: 100 Mbps	.801 s	.9 s

NETWORK PERFORMANCE MEASUREMENT UNITS

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

TERMINOLOGY STYLE

- Mega versus Mega, Kilo versus Kilo
  - Computer architecture: Mega □ 2^20, Kilo □ 2^10
  - Computer networks: Mega □ 10^6, Kilo □ 10^3
- Mbps versus MBps
  - Networks: typically megabits per second
  - Architecture: typically megabytes per second
- Bandwidth versus throughput
  - Bandwidth: available over link
  - Throughput: available to application
    - E.g. subtract protocol headers, etc.

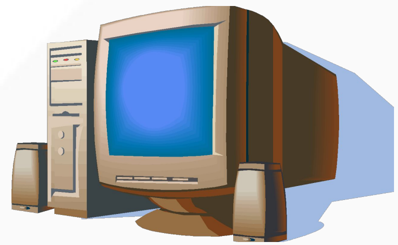
PERFORMANCE TOOLS

- Ping
  - Test if other side is “alive”
  - Measures round-trip latency
- iperf3
  - Times how long it takes to send N bytes to the other endpoint
  - Used to calculate bandwidth

Today’s agenda

- Performance metrics
- The problem of tail latency
- Case study: Memcached

CLIENT-SERVER OVERVIEW



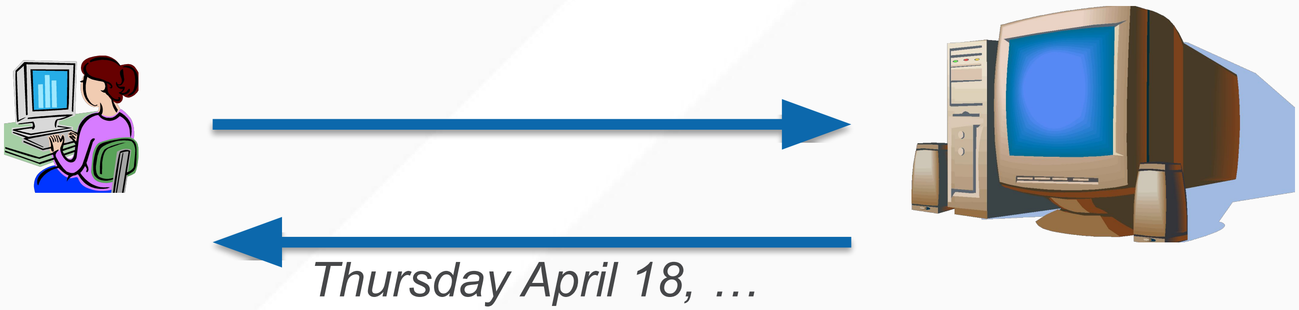
`net.Listen()`

`net.Accept()`

`net.Dial()` → `var conn net.Conn = net.Accept()`

`conn.Write()/conn.Read()` ↔ `conn.Write()/conn.Read()`

SIMPLE EXAMPLE: DATETIME SERVER



- No real “state” to worry about on the server
- A single machine can fully implement this protocol
- But what about a data-intensive application?

MORE COMPLEX EXAMPLE: WEB SEARCH

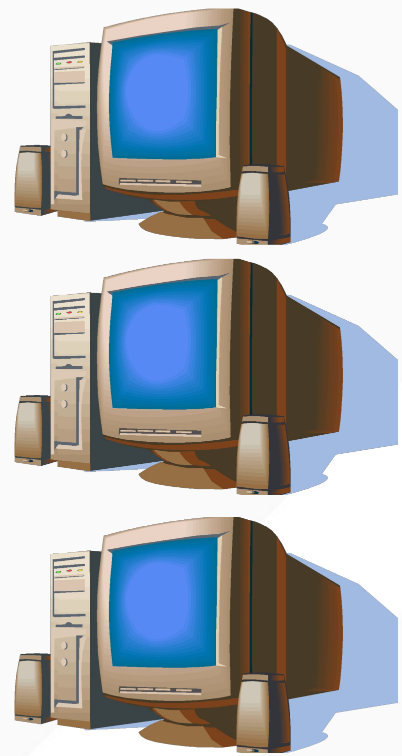
Search index servers

Search terms held by server

Turing?

Turing?

Turing?



CSE, Triton, Price Center

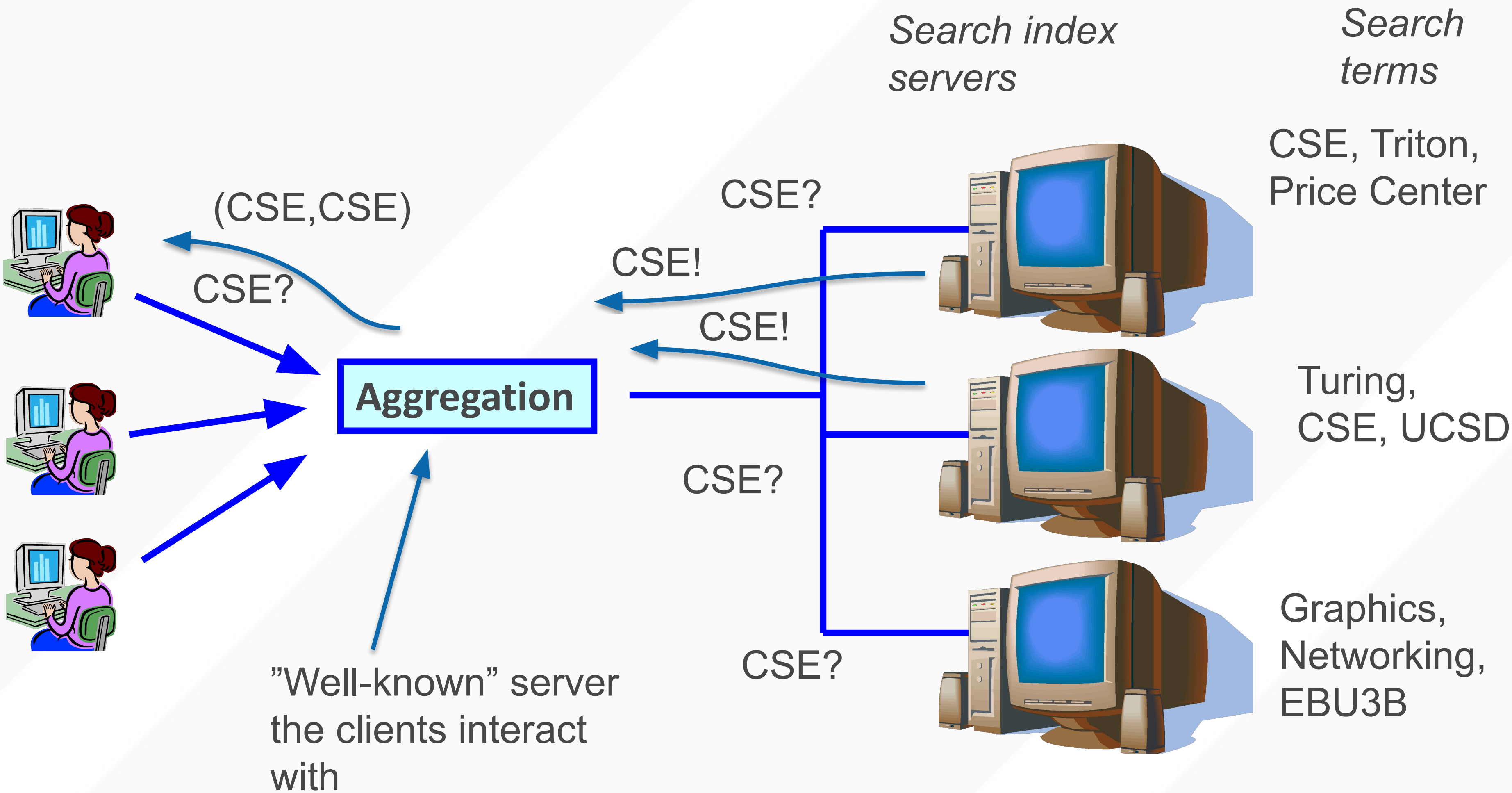
Turing, CSE, UCSD

Graphics, Networking, EBU3B

Problems:

- Client must query each server
- Client must know about each server
- Client does a lot of work
- Hard to update/maintain the servers without exposing internal organization of the service to the client

AGGREGATION SERVER / FRONT-END SERVER



FOCUS OF TODAY'S DISCUSSION: TAIL LATENCY

- "If my application runs on N servers, and I add M more servers, will the performance as seen by the user be better? The same? Worse?"
- Better?
- The Same?
- Worse?
- Well… it's kinda complicated…

"THE TAIL AT SCALE"

Head of Google ai; (Co-)designed Google's Ad engine, Web crawler, indexer, and query serving system. Created Spanner, BigTable, MapReduce, LevelDB, TensorFlow (AI/ML system), …

Google Fellow, VP of Engineering, Technical lead of Google's infrastructure and datacenters

Jeffrey Dean and Luiz André Barroso. The tail at scale. Communication of the ACM 56, 2 (February 2013), 74-80. DOI: https://doi.org/10.1145/2408776.2408794

PERCENTILES

- Imagine a set of runners R run a foot race, and each finishes in time Ri
- We can sort these times, and compute various *percentiles*
- 50^{th} percentile is the value where 50 percent are below that value
- 90^{th} percentile is the value where 90 percent are below that value
- Used in testing, performance analysis, etc…

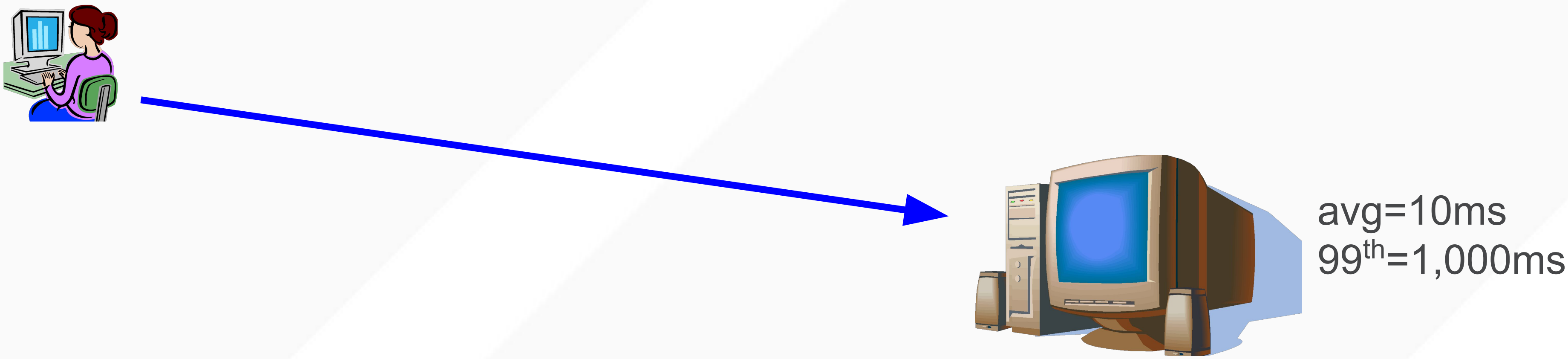
EFFECT OF LATENCY VARIATION

service to feel responsive.

Variability in the latency distribution of individual components is magnified at the service level; for example, consider a system where each server typically responds in 10ms but with a 99th-percentile latency of one second. If a user request is handled on just one such server, one user request in 100 will be slow (one second). The figure here outlines how service-level latency in this hypothetical scenario is affected by very

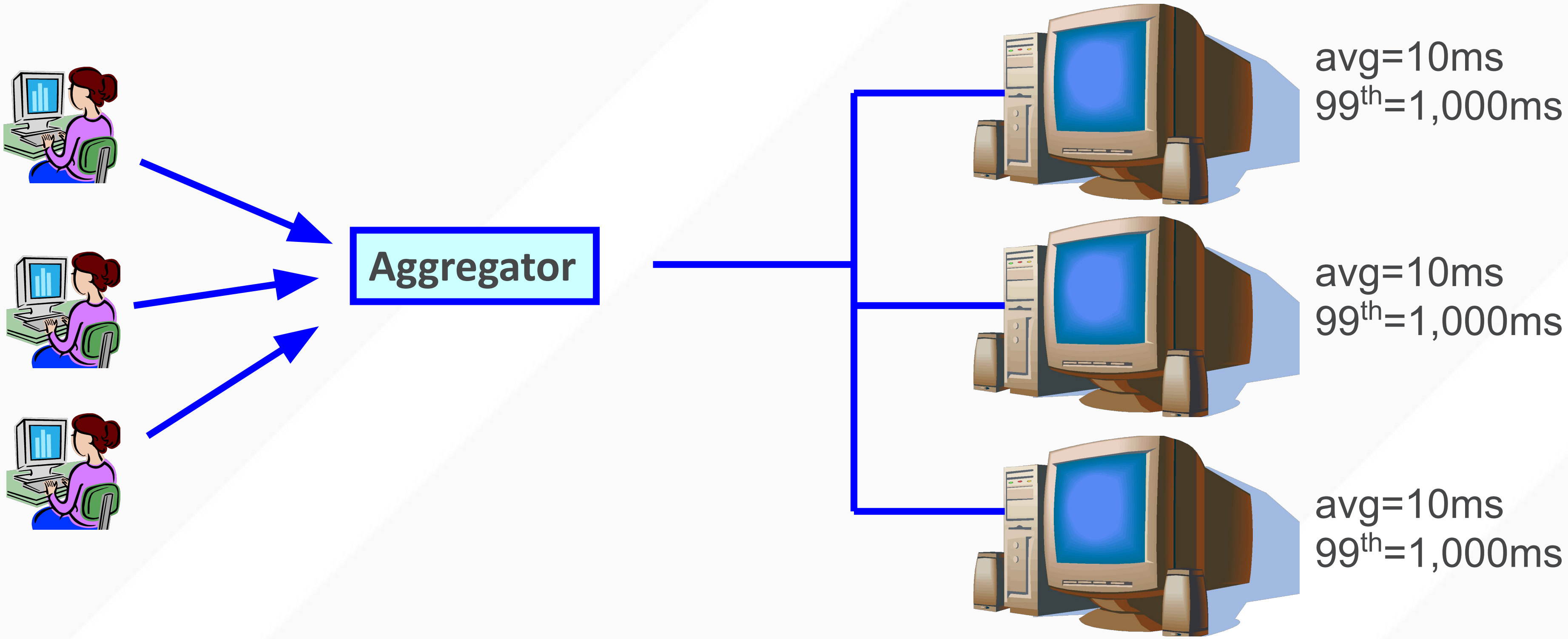
higher-level queuing. Di
vice classes can be used
uling requests for whic
ing over non-interactiv
low-level queues short
policies take effect mor
ample, the storage ser
cluster-level file-system
few operations outstar
erating system's disk
maintaining their own
of pending disk reques

PERFORMANCE OF A SINGLE SERVER



- What is the expected time to service one request to one server?
  - 10ms? more? less?

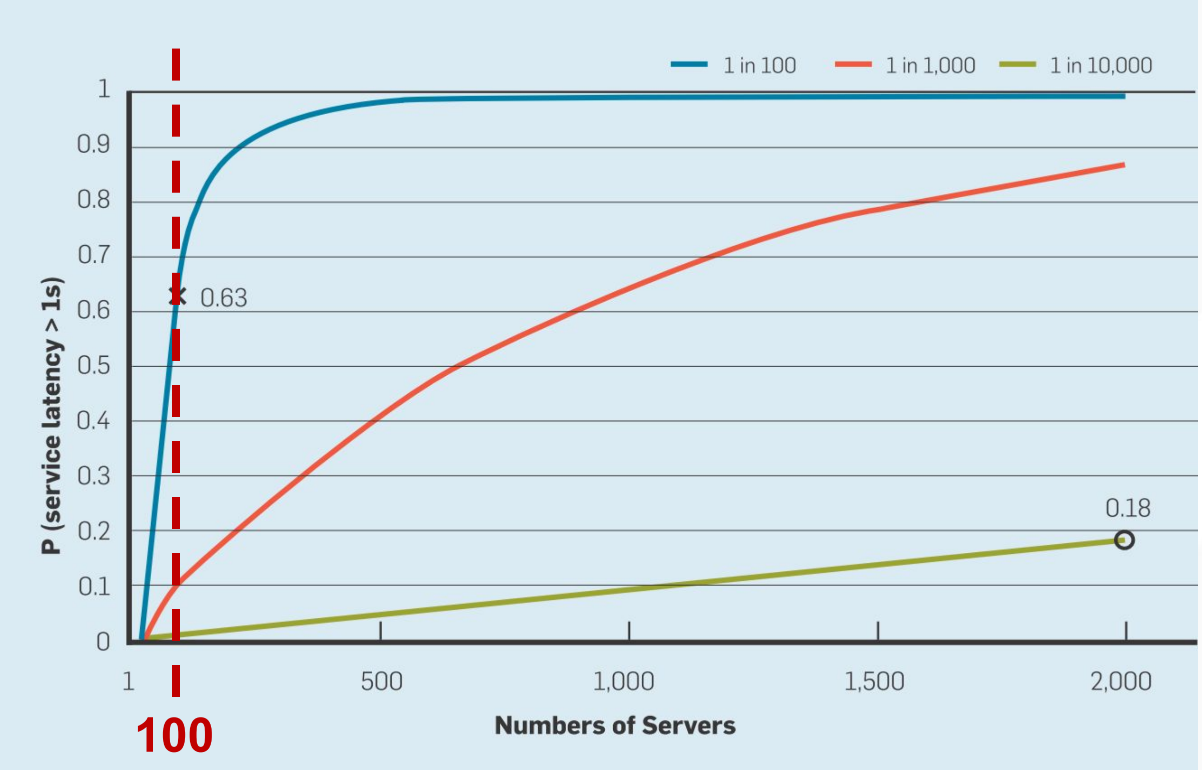
PERFORMANCE AT SCALE



- What is the expected time to service three correlated requests to three servers?
  - Must wait until all complete before the load balancer can return a result to the user
  - 10ms? more? less?

COMPONENT VARIABILITY AMPLIFIED BY SCALE

- Latency variability is magnified at the service level.



REQUEST LATENCY MEASUREMENT

	50%ile latency	95%ile latency	99%ile latency
One random leaf finishes	1ms	5ms	10ms
95% of all leaf requests finish	12ms	32ms	$\boxed{70ms}$
100% of all leaf requests finish	40ms	87ms	$\boxed{140ms}$

50%+

- Key Observation:
  - 5% servers contribute nearly 50% latency.
  - *Why not just rid of those “slow” 5% of the servers?*

FACTORS OF VARIABLE RESPONSE TIME

- Shared Resources (Local)
  - CPU cores
  - Processors caches
  - Memory bandwidth
- Global Resource Sharing
  - Network switches
  - Shared file systems
- Daemons
  - Scheduled Procedures

FACTORS OF VARIABLE RESPONSE TIME

- Maintenance Activities
  - Data reconstruction in distributed file systems
  - Periodic log compactions in storage systems
  - Periodic garbage collection in garbage-collected languages
- Queueing
  - Queueing in intermediate servers and network switches

FACTORS OF VARIABLE RESPONSE TIME

- Power Limits
  - Throttling due to thermal effects on CPUs
- Garbage Collection
  - Random access in solid-state storage devices
  - Twitter’s interesting take on GC…
- Energy Management
  - Power saving modes
  - Switching from inactive to active modes

HEDGING REQUESTS

- Tied Requests - send requests to two servers
  - Hedged requests with cancellation mechanism.

	Mostly idle cluster			With concurrent terasort		
	No hedge	Tied request after 1ms		No hedge	Tied request after 1ms	
50%ile	19ms	16ms	(–16%)	24ms	19ms	(–21%)
90%ile	38ms	29ms	(–24%)	56ms	38ms	(–32%)
99%ile	67ms	42ms	(–37%)	108ms	67ms	(–38%)
99.9%ile	98ms	61ms	(–38%)	159ms	108ms	(–32%)

REDUCING COMPONENT VARIABILITY

- Differentiating Service Classes
  - Differentiate non-interactive requests
- High Level Queuing
  - Keep low level queues short
- Reduce Head-of-line Blocking
  - Break long-running requests into a sequence of smaller requests.
- Synchronize Disruption
  - Do background activities altogether.

LARGE INFORMATION RETRIEVAL SYSTEMS

- Google search engine
  - No certain answers
- ChatGPT?
- “Good Enough”
  - Google’s IR systems are tuned to occasionally respond with good-enough results when an acceptable fraction of the overall corpus has been searched.

LARGE INFORMATION RETRIEVAL SYSTEMS

- Canary Requests
    - Some requests exercising an untested code path may cause crashes or long delays.
    - Send requests to one or two leaf servers for testing.
    - The remaining servers are only queried if the root gets a successful response from the canary in a reasonable period of time.

HARDWARE TRENDS AND THEIR EFFECTS

- Hardware will only be more and more diverse
  - So tolerating variability through software techniques are even more important over time.
- Higher bandwidth reduces per-message overheads.
  - It further reduces the cost of tied requests (making it more likely that cancellation messages are received in time).

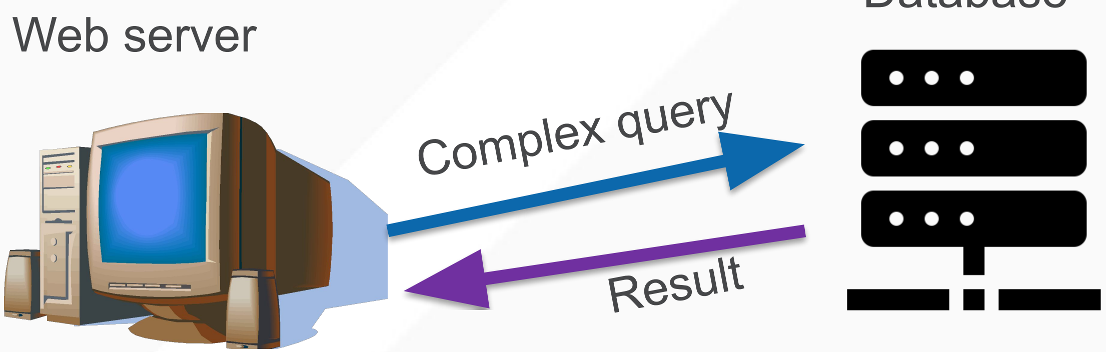
Today’s agenda

- Performance metrics
- The problem of tail latency
- Case study: Memcached

CASE STUDY: MEMCACHED

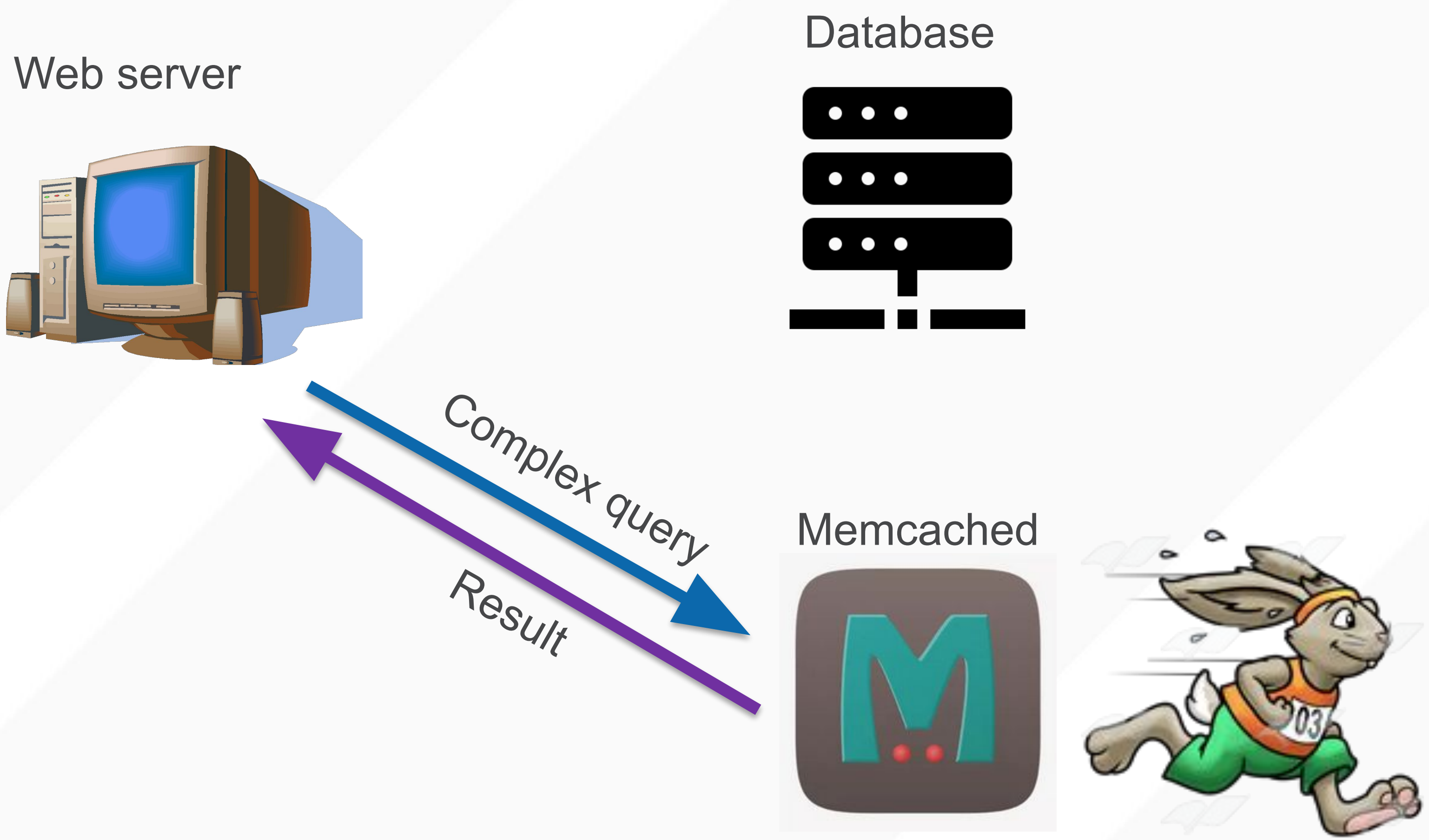
- Popular in-memory cache
- Simple get() and put() interface
- Useful for caching popular or expensive requests
- LRU replacement policy
- Data stored in RAM so access is O(1) and fast

BASELINE: DATABASE-DRIVEN WEB QUERY

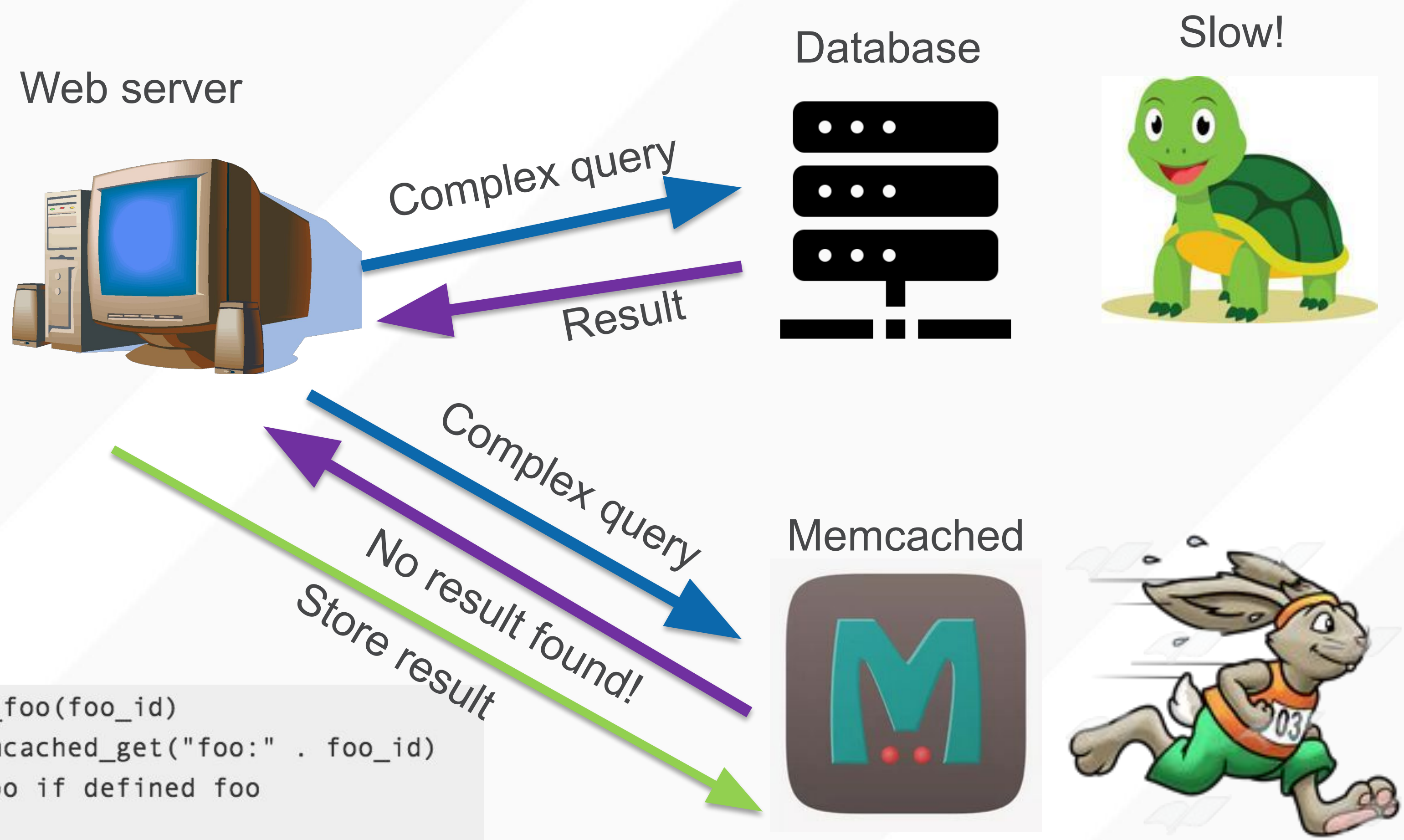


Slow!

MEMCACHED EXAMPLE: CACHE HIT



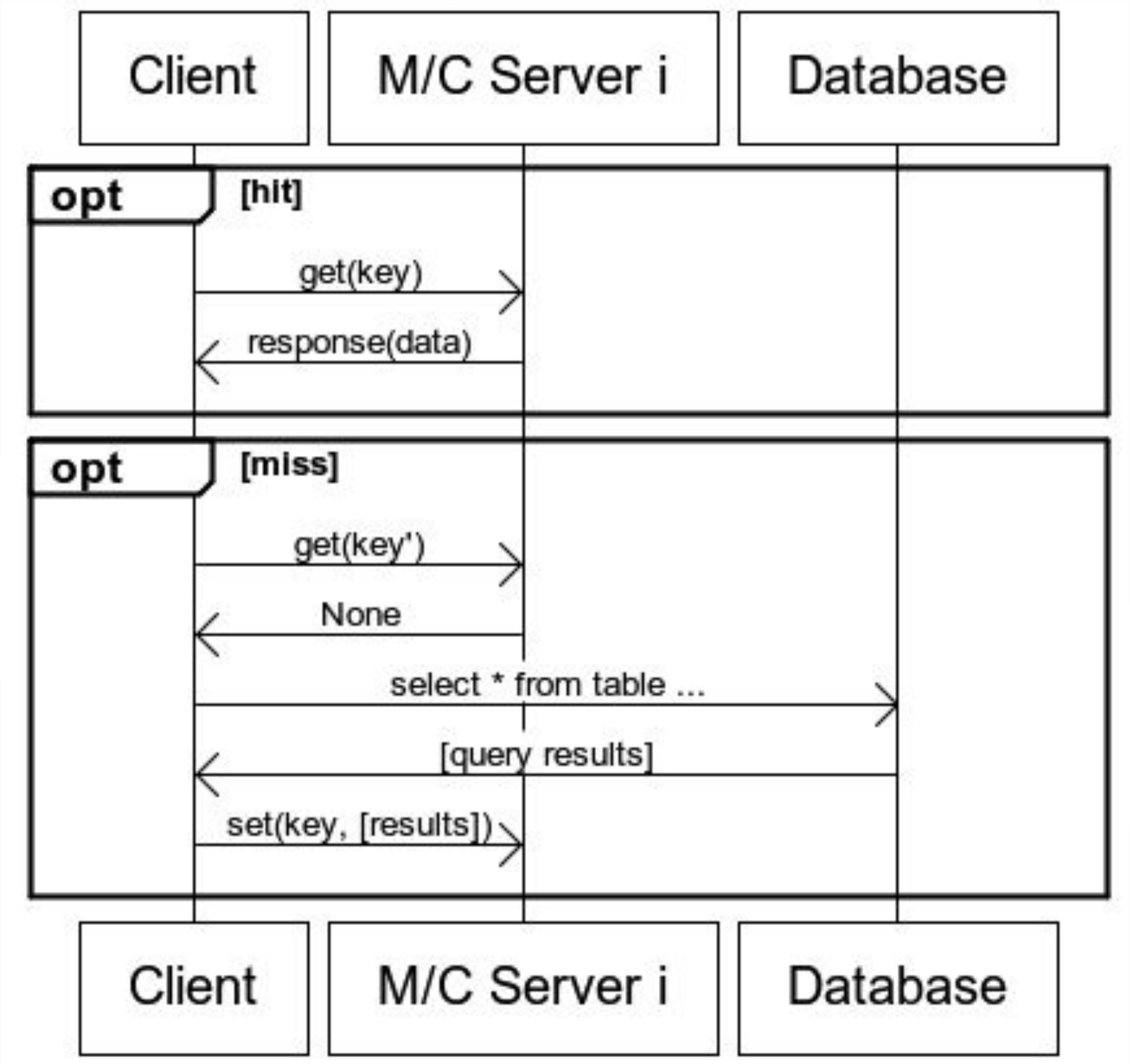
MEMCACHED EXAMPLE: CACHE MISS




```
function get_foo(foo_id)
    foo = memcached_get("foo:" . foo_id)
    return foo if defined foo

    foo = fetch_foo_from_database(foo_id)
    memcached_set("foo:" . foo_id, foo)
    return foo
end
```

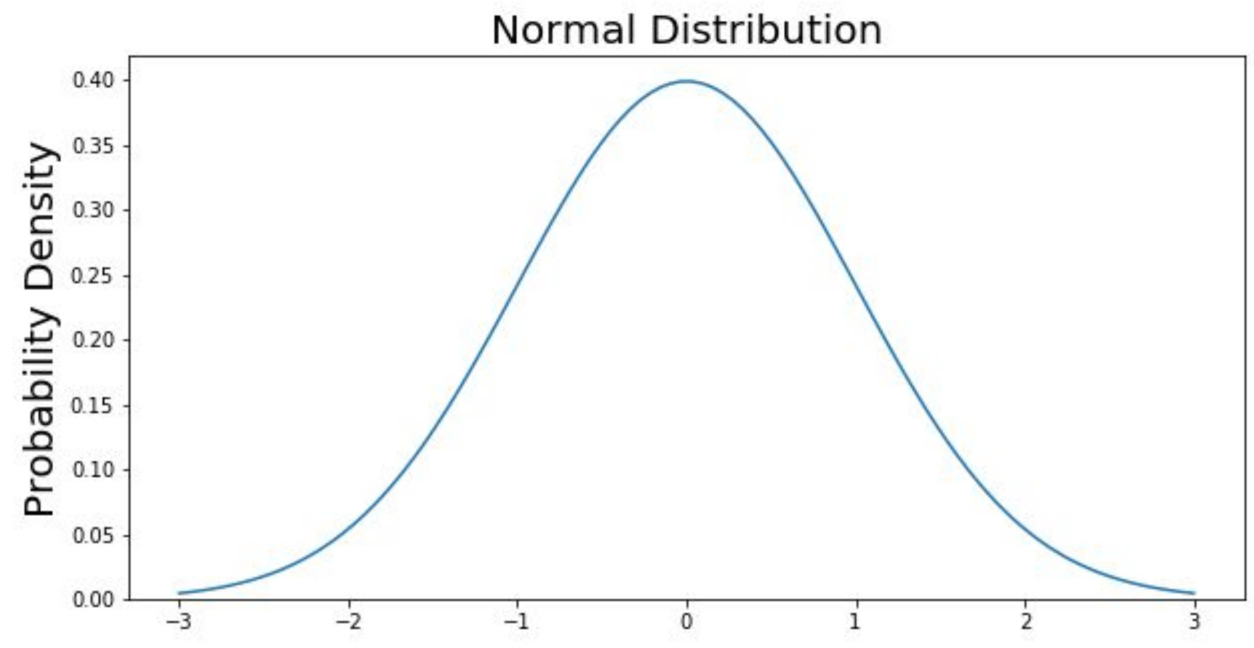
MEMCACHED DATA FLOW



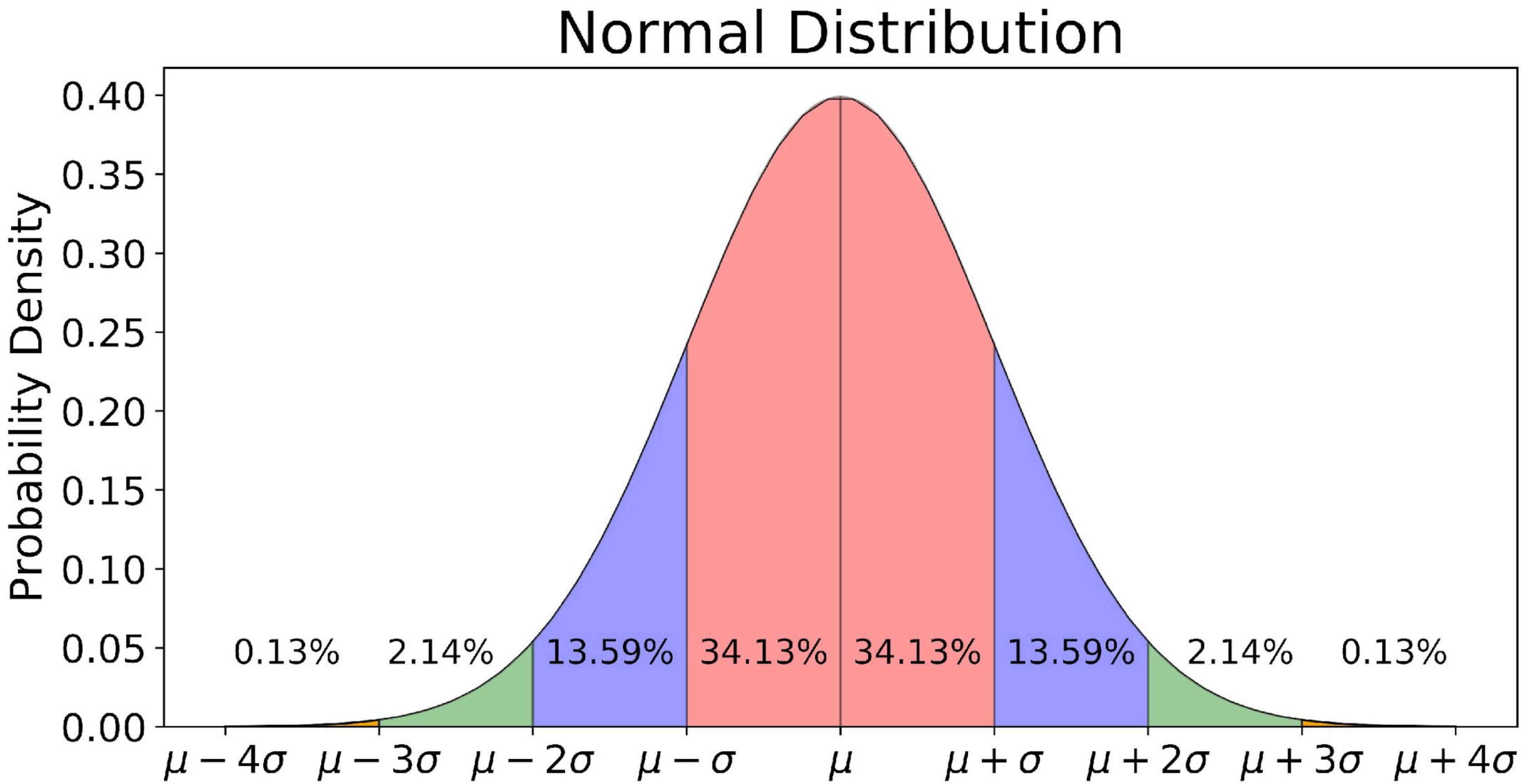
EXPERIMENT: GET/SET WITH MEMCACHED

```
[demo code]
```

RANDOM VARIABLES: NORM(0,1)

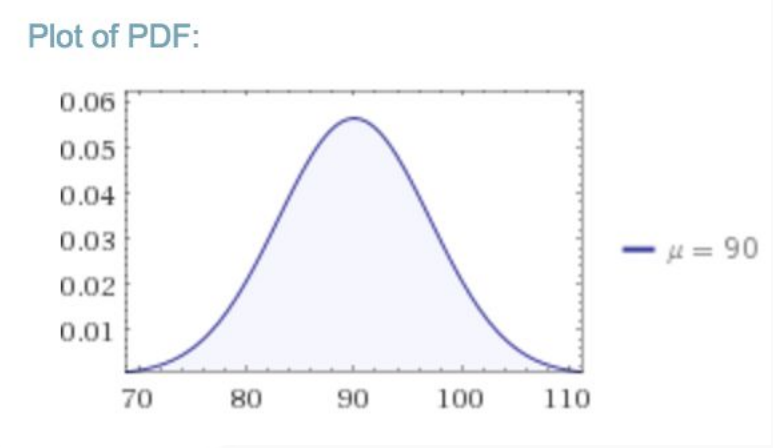


RANDOM VARIABLES: NORM(μ, σ)



TAIL TOLERANCE: PARTITION/AGGREGATE

- Consider distributed memcached cluster
  - Single client issues request to S memcached servers
    - Waits until all S are returned
  - Service time of a memcached server is normal w/ μ = 90us, σ = 7us
    - Roughly based on measurements from a former student of mine, Rishi Kapoor



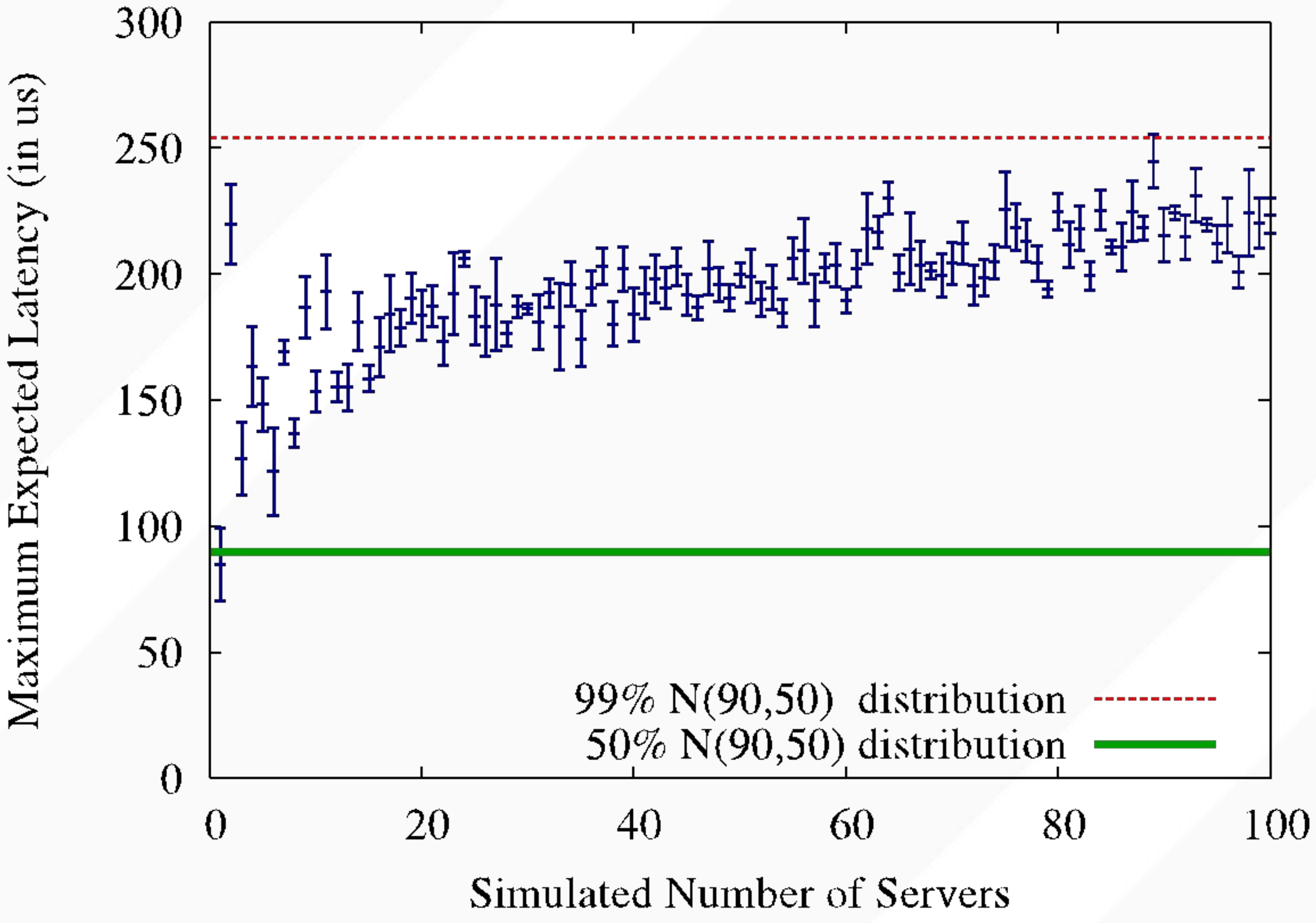
EXPLORING NORMAL RANDOM VARIABLES WITH GOOGLE SHEETS

- You too can generate observations of a normal random variable by adding this to a google sheets (or excel, numbers, etc) document:
  - `=NORMINV(rand(),0,1)`

EXPLORING NORMAL RANDOM VARIABLES WITH GOOGLE SHEETS

- You too can generate observations of a normal random variable by adding this to a google sheets (or excel, numbers, etc) document:
  - Based on Memcached:
  - `=NORMINV(rand(),90,7)`

MATLAB SIMULATION



UC San Diego