Review

COMPSCI 210 Recitation

7th Dec 2012

Vamsi Thummala

Latency Comparison

L1 cache reference	0.5	ns	
Branch mispredict	5	ns	
L2 cache reference	7	ns	14x L1 cache
Mutex lock/unlock	25	ns	
Main memory reference	100	ns	20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	3,000	ns	
Send 1K bytes over 1 Gbps network	10,000	ns	0.01 ms
Read 4K randomly from SSD	150,000	ns	0.15 ms
Read 1 MB sequentially from memory	250,000	ns	0.25 ms
Round trip within same datacenter	500,000	ns	0.5 ms
Read 1 MB sequentially from SSD	1,000,000	ns	1 ms 4X memory
Disk seek	10,000,000	ns	10 ms 20x data center roundtrip
Read 1 MB sequentially from disk	20,000,000	ns	20 ms 80x memory, 20X SSD
Send packet CA->Netherlands->CA	150,000,000	ns	150 ms

Wait, what is a nanosecond?

Abstractions: Beauty and Chaos

- Context
- Component
- Connector
- Channel
- Event
- Entity
- Identity
- Attribute
- Label
- Principal
- Reference Monitor
- Subject
- Object
- Guard

Still ...

http://www.clipartoday.com/clipart/cartoons/cartoon/cartoon_264115.html

Context: Unix

- Pipeline example:

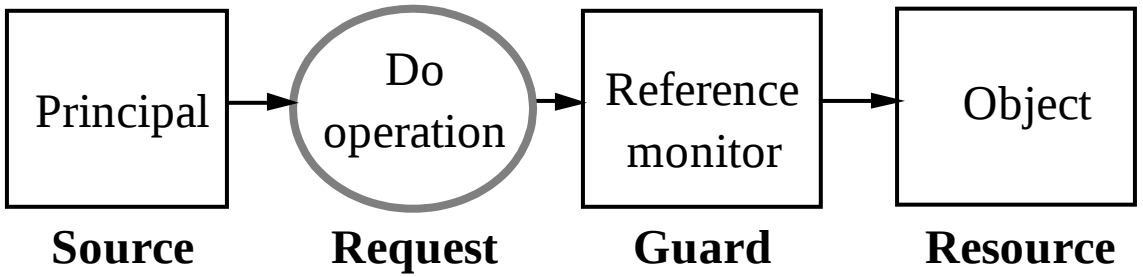
```
cat compsci210.txt | wc | mail -s "word count" chase@cs.duke.edu
```

- Component
    - Executable program
- Context
    - Components in context
    - Process
- Connector
    - Pipes
- In general, an OS:
    - Sets up the contexts
    - Enforces isolation
    - Mediates interaction

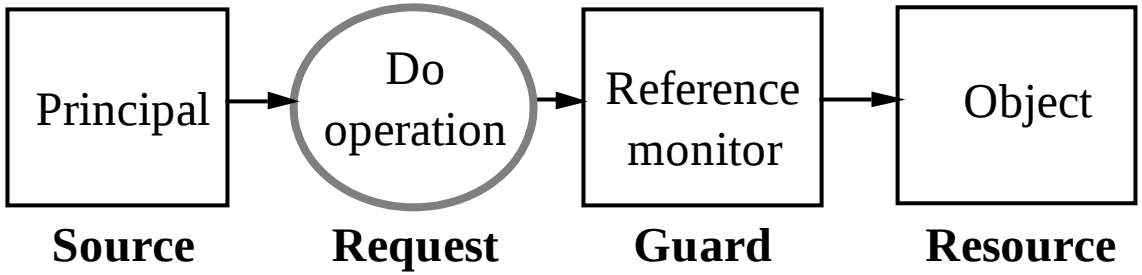
Context: Protection

Excerpt from Notes on Security:

The Unix example exposes some principles that generalize to other systems. In general, all of the OS platforms we consider execute programs (or components, or modules) in processes (or some other protected context, or sandbox, or protection domain) on nodes linked by communication networks. A platform's protection system labels each running program context with attributes representing “who it is”, and uses these labels to govern its interactions with the outside world.



More on Protection



Principal may do	*Operation*	on	*Object*
Chase	Read		dFile
Alice	Pay invoice 4325		Account Q34
Bob	Fire three rounds		Bow gun

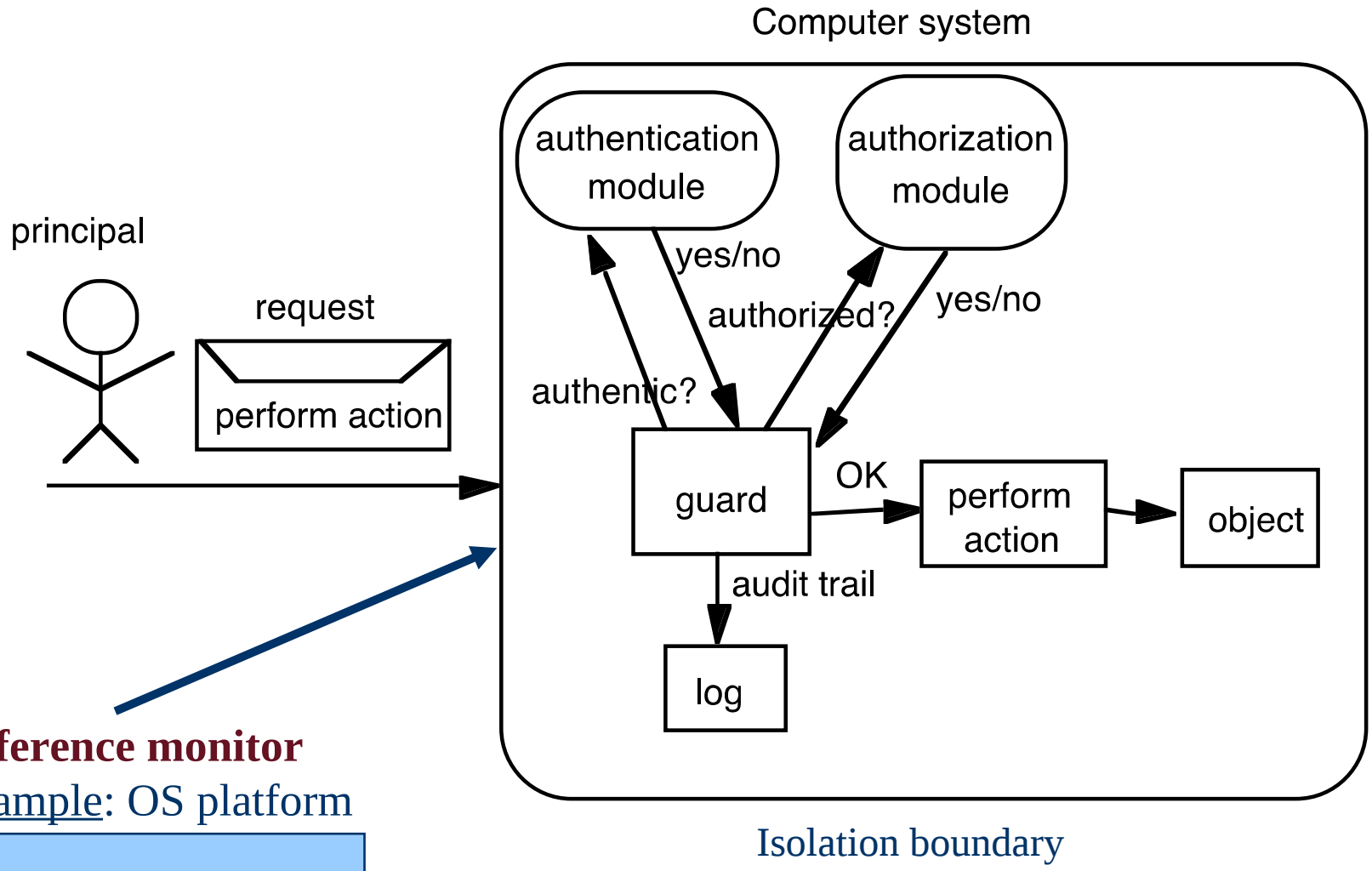
Authentication: Who sent a message?
Authorization: Who is trusted?

Principal: Abstraction of “who”

- People: Chase, Alice
- Services: DeFiler

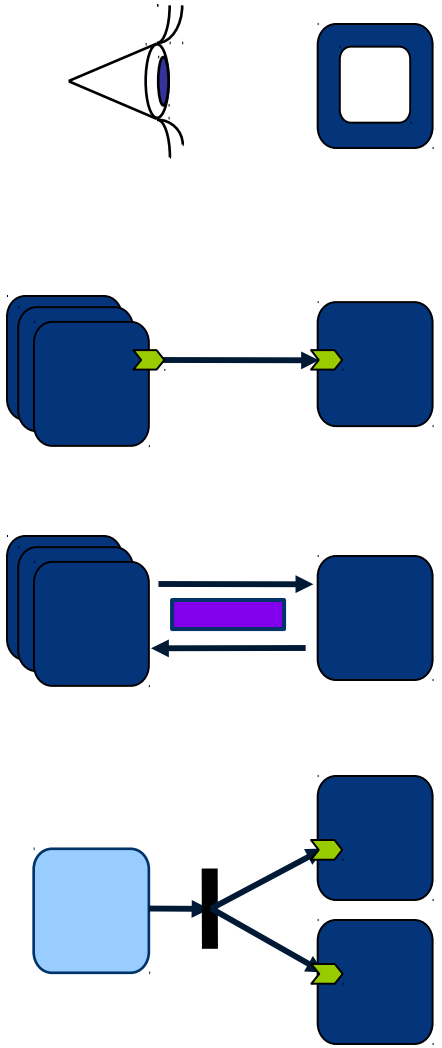
Principles for Computer System Design, Turing Award Lecture, 1983

Protection Systems 101



Context: Android

- The four component types
  - Activity. Display a screen.
    - Push on a “back stack”.
    - May be launched by other apps.
  - Service. Serve an API.
    - Establish an external binder interface.
    - Public methods are externally visible.
  - Provider. Get/put content objects.
    - Serve a URI space with MIME types.
    - Backed by SQLite database tables.
  - Receiver. Respond to events.
    - E.g., low battery.



Synchronization

- Practice problem

  Larry, Moe, and Curly are planting seeds. Larry digs the holes. Moe then places a seed in each hole. Curly then fills the hole up.

  There are several synchronization constraints:

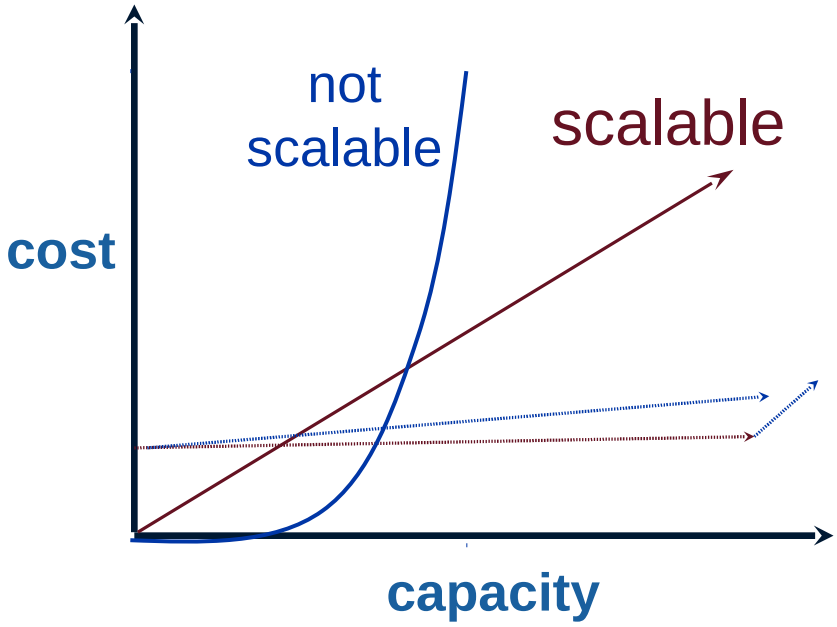
  - Moe cannot plant a seed unless at least one empty hole exists, but Moe does not care how far Larry gets ahead of Moe.
  - Curly cannot fill a hole unless at least one hole exists in which Moe has planted a seed, but the hole has not yet been filled. Curly does not care how far Moe gets ahead of Curly.
  - Curly does care that Larry does not get more than MAX holes ahead of Curly. Thus, if there are MAX unfilled holes, Larry has to wait.
  - There is only one shovel with which both Larry and Curly need to dig and fill the holes, respectively.

  Sketch out the pseudocode for the 3 processes which represent Larry, Curly, and Moe using semaphores as the synchronization mechanism.

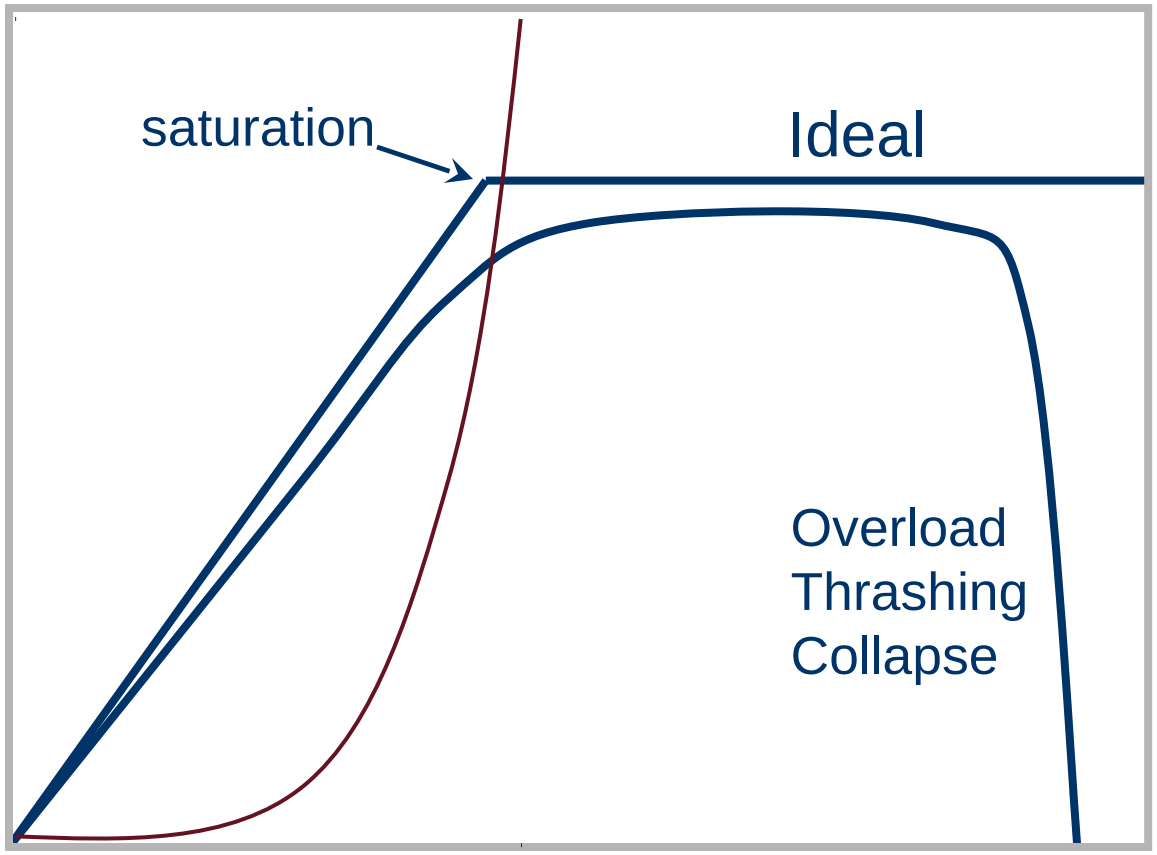
Performance

- Single node OS
  - Latency/Response time
  - Throughput

- Internet Scale systems
  - Consistency
  - Availability
  - Partition Tolerance
  - Incremental scalability



Servers Under Stress



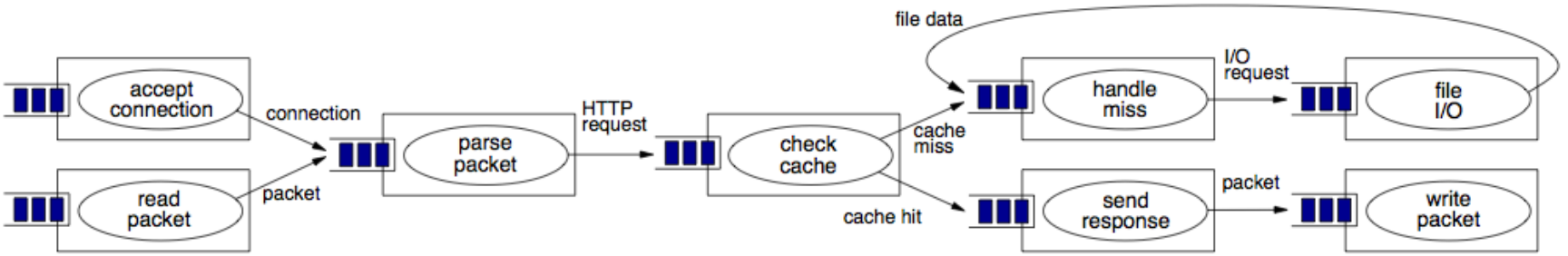
Response rate (throughput)

Response time

Request arrival rate (offered load)

[Von Behren]

Staged Event-Driven Architecture (SEDA)



Decompose service into *stages* separated by *queues*

- Each stage performs a subset of request processing
- Stages internally event-driven, typically nonblocking
- Queues introduce execution boundary for isolation and conditioning

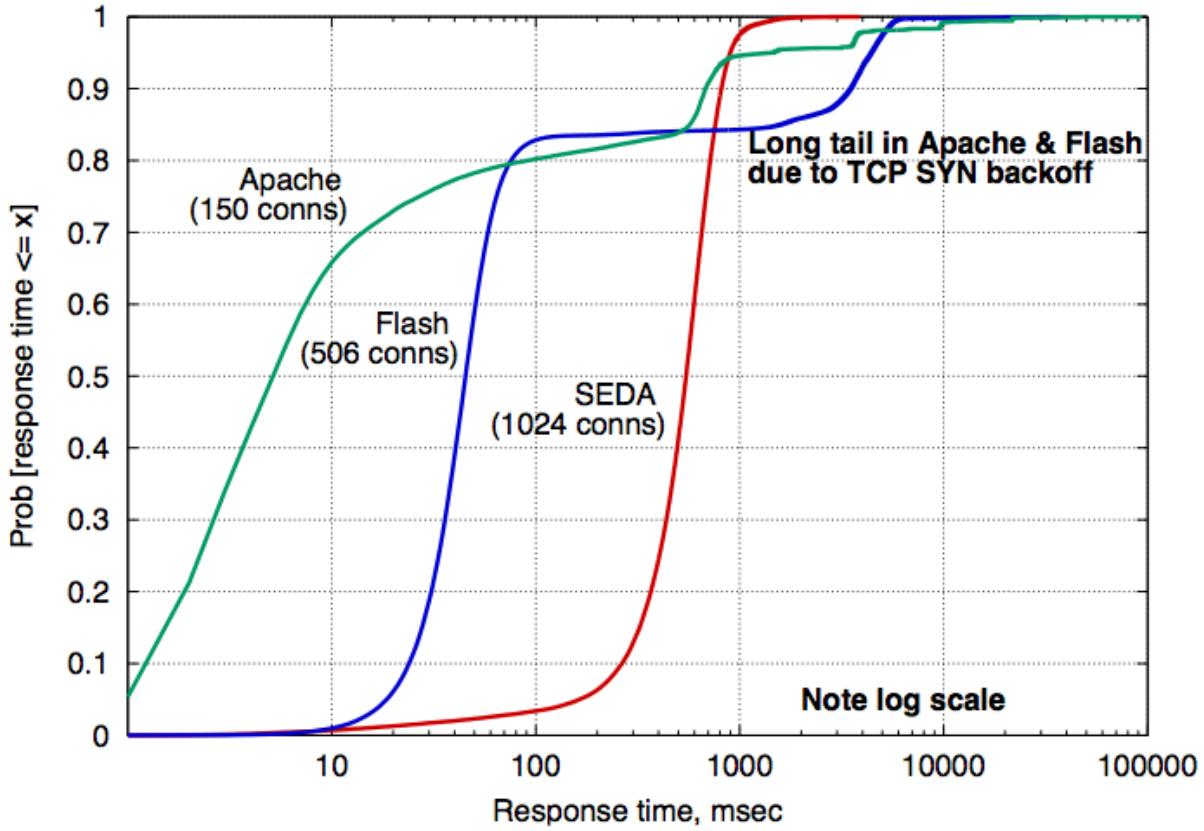
Each stage contains a *thread pool* to drive stage execution

- However, threads are not exposed to applications
- Dynamic control grows/shrinks thread pools with demand
  - ▷ *Stages may block if necessary*

Best of both threads and events:

- Programmability of threads with explicit flow of events

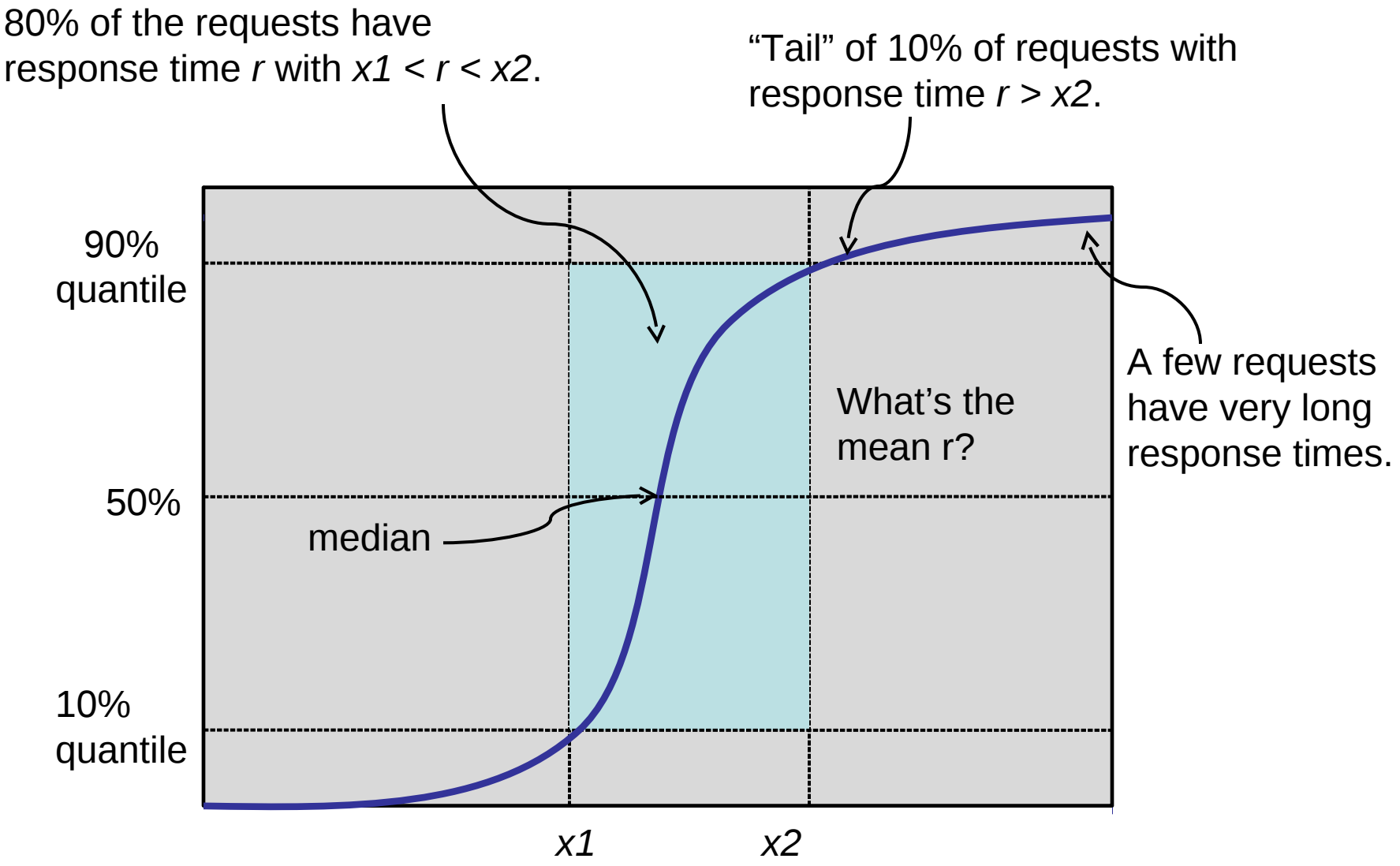
Response Time Distribution - 1024 Clients



	SEDA	Flash	Apache
Mean RT	547 ms	665 ms	475 ms
Max RT	3.8 sec	37 sec	1.7 minutes

- SEDA yields **predictable performance** - Apache and Flash are very **unfair**
  - *''Unlucky'' clients see long TCP retransmit backoff times*
  - *Everyone is ''unlucky'': multiple HTTP requests to load one page!*

Cumulative Distribution Function (CDF)



Understand how the mean (average) response time can be misleading.

SEDA Lessons

- Means/averages are almost never useful: you have to look at the distribution.
- Pay attention to quantile response time.
- All servers must manage overload.
- Long response time tails can occur under overload, and that is bad.
- A staged structure with multiple components separated by queues can help manage performance.
- The staged structure can also help to manage concurrency and and simplify locking.