A (quick) retrospect

COMPSCI210 Recitation

22th Apr 2013

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	

Abstractions: Beauty and Chaos

- ✓ Context
- ✓ Component
- ✓ Connector
- ✓ Channel
- ✓ Event
- ✓ Entity
- ✓ Identity
- ✓ App
- ✓ Signature
- ✓ Attribute
- ✓ Label
- ✓ Principal
- ✓ Reference Monitor
- ✓ Subject
- ✓ Object
- ✓ Guard
- ✓ Service
- ✓ Module

Case Study: Unix

- Example program:

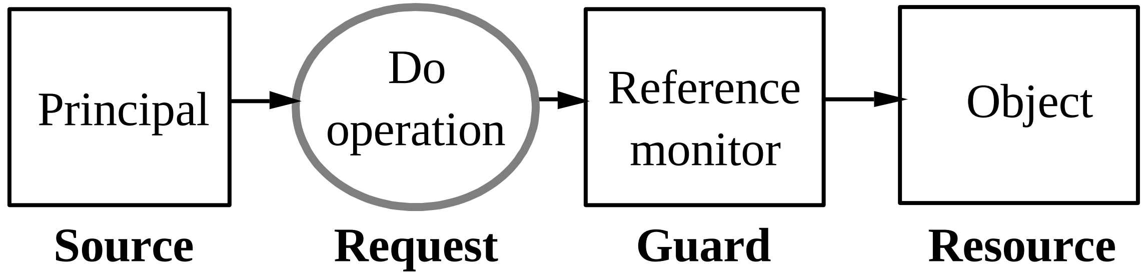
```
cat compsci210.txt | wc | mail -s "word count" chase@cs.duke.edu
```

- Component: Executable program
- Context: Process that executes the component
- Connector: Pipes
- In general, an OS:
  - Sets up the context
  - Enforces isolation
  - Mediates interaction

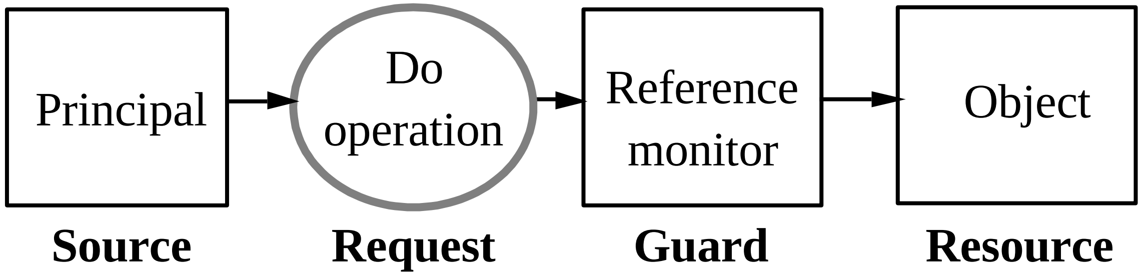
Case Study: Unix protection

- 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

Case Study: Android

- What is a component?
  - Types of components?
- What is an App?
- What is a Binder service?
- What is a Zygote?
  - Why does Andorid context needs just a fork() but not exec()?
- How does Android protection differs from Unix?

Prof. Chase slides

Concurrency

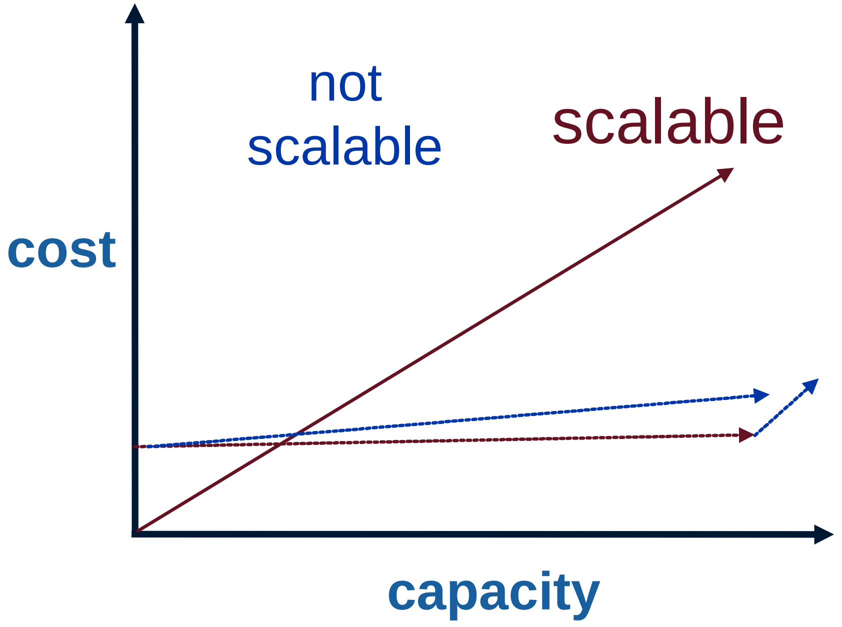
- Mutual exclusion
  - Lock/mutex; too much milk
- Monitor
  - CV + mutex; scheduling threads; ping-pong
- Semaphore
  - Numeric resources; producer-consumer soda example
- EventBarrier
  - Scheduling in phases/batches; Elevator

- Implement one primitive in terms of the other
  - E.g., Implement a Semaphore using only a monitor

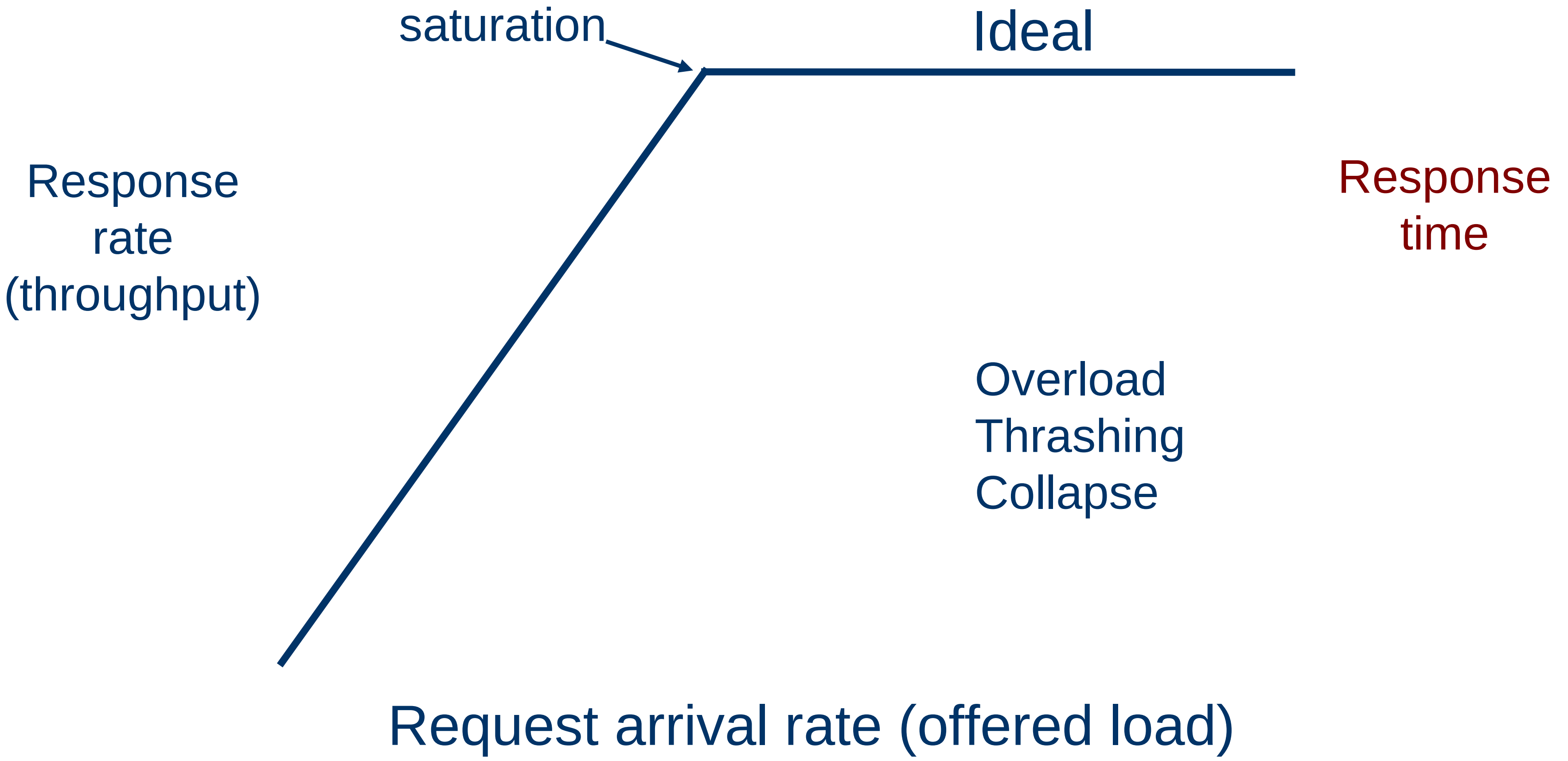
Performance

- Single node OS
  - Latency/Response time
  - Throughput

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

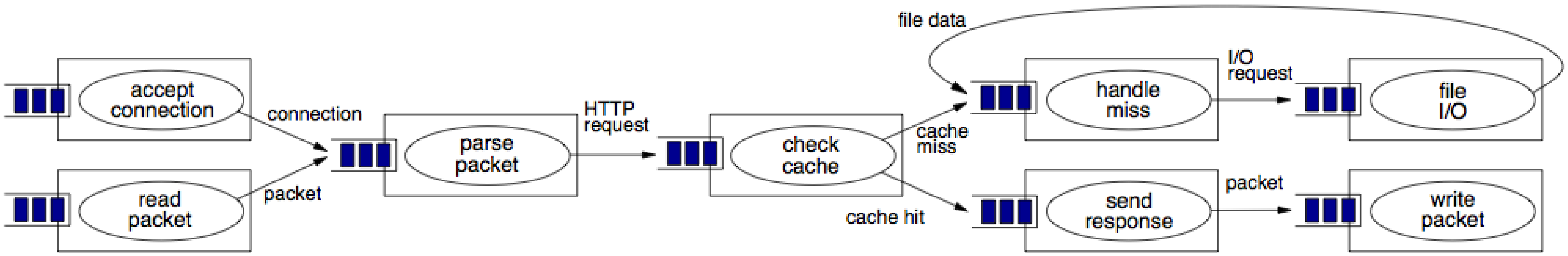


Servers Under Stress



[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*

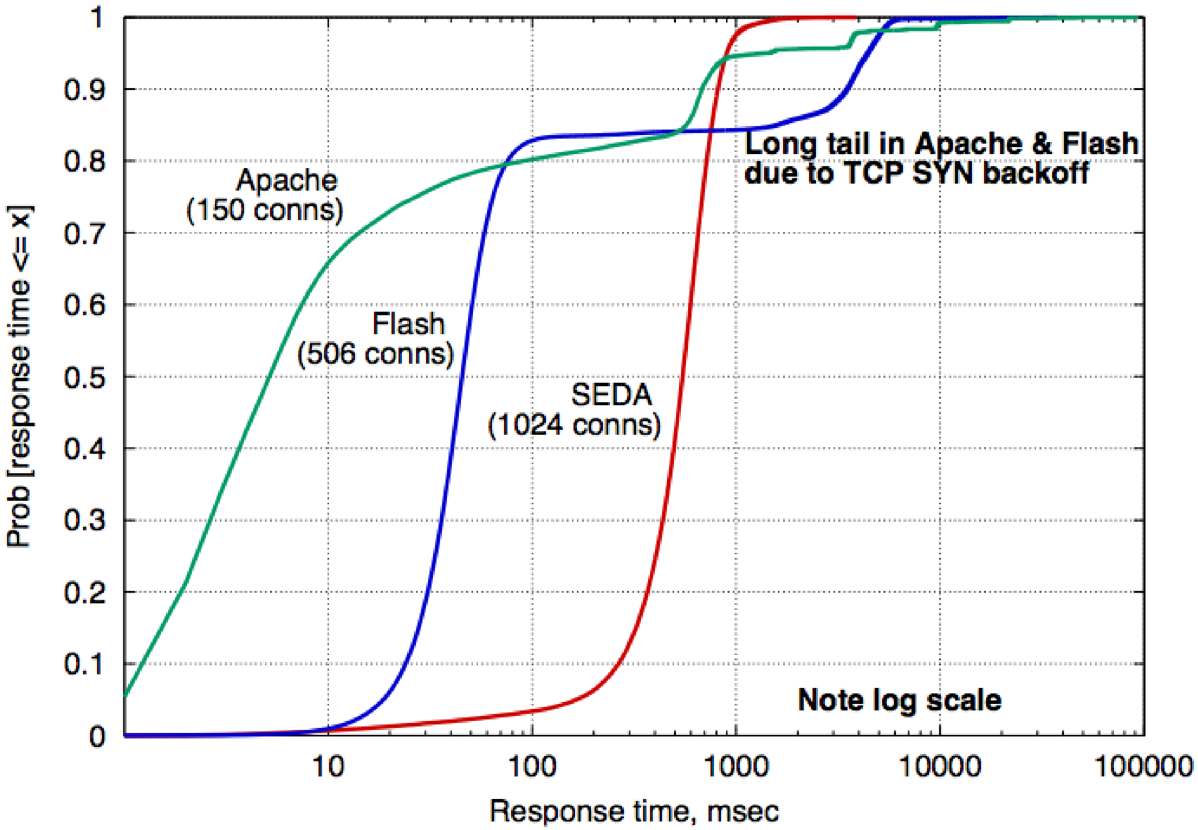
Best of both threads and events:

- Programmability of threads with explicit flow of events

Crypto: Concept checkers

- What is the basic assumption that cryptography relies on?
- What is a hash/finger print/digest?
- What is a digital signature?
- Symmetric vs Asymmetric crypto
- What is a nonce?
- What is a security/treat model?
- Type of attacks and defenses

Response Time Distribution - 1024 Clients



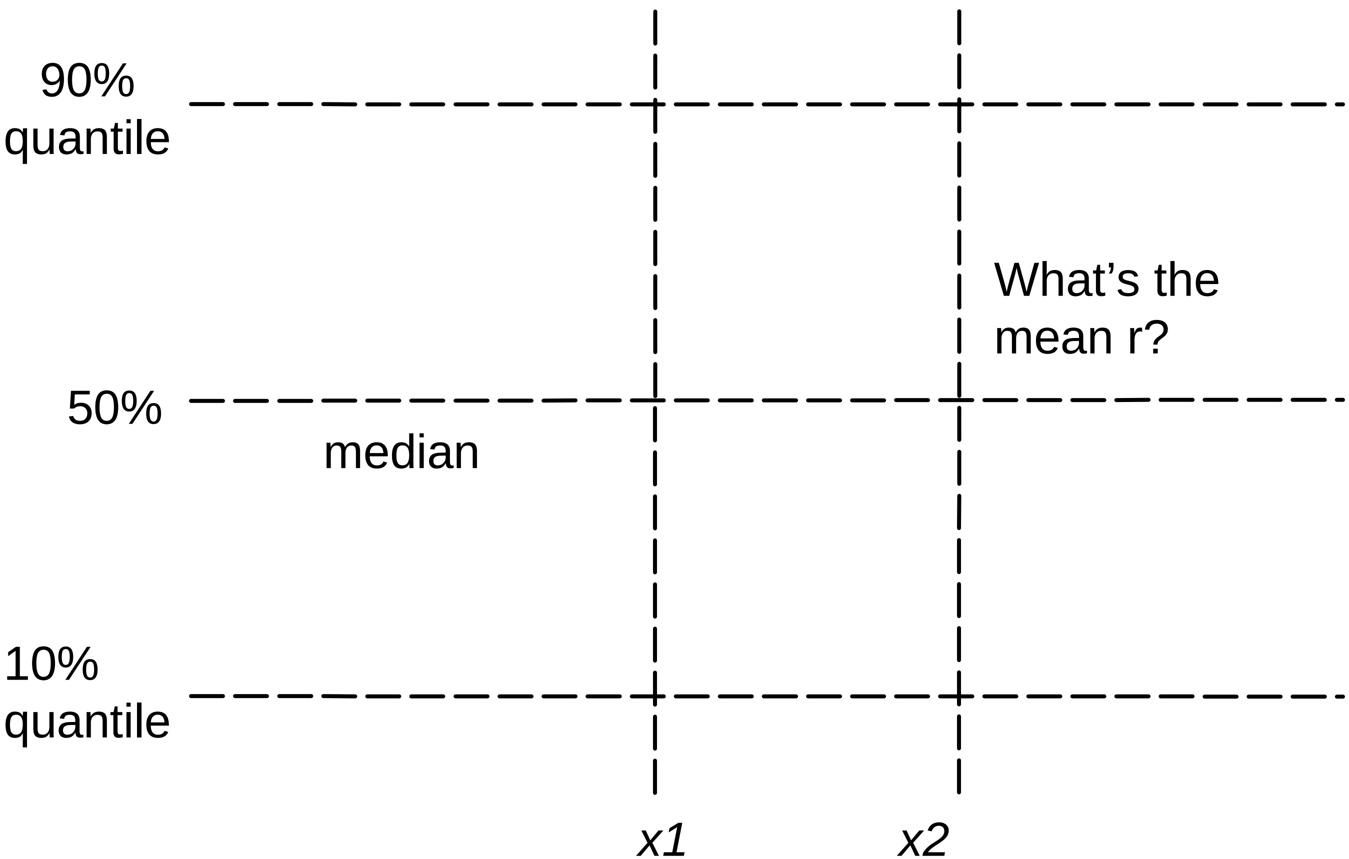
	SEDA	Flash	Apache
Mean RT	547 ms	665 ms	475 ms
Max RT	3.8 sec	37 sec	1.7 minutes

- 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)

80% of the requests have response time *r* with *x1* < *r* < *x2*.

“Tail” of 10% of requests with response time *r* > *x2*.



A few requests have very long response times.

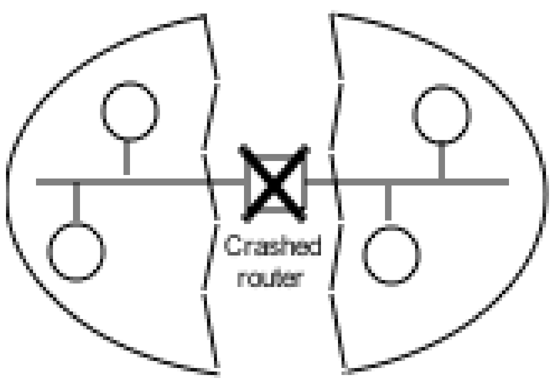
Understand how the mean (average) response time can be misleading.

SEDA Lessons

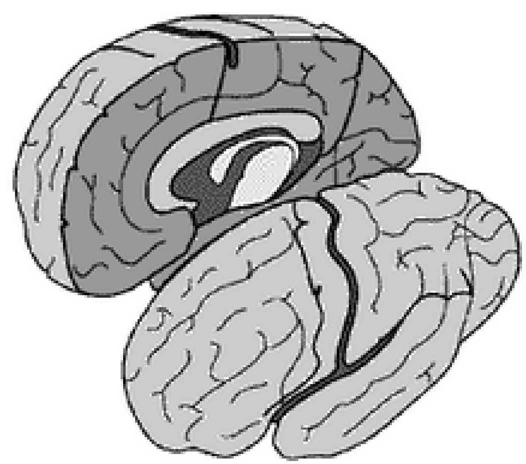
- Means/averages are almost never useful: you have to look at the distribution.
- Pay attention to quantile response time.
- All servers must manage overload.
- Long response time tails can occur under overload, and that is bad.
- A staged structure with multiple components separated by queues can help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

Fischer-Lynch-Patterson (1985)

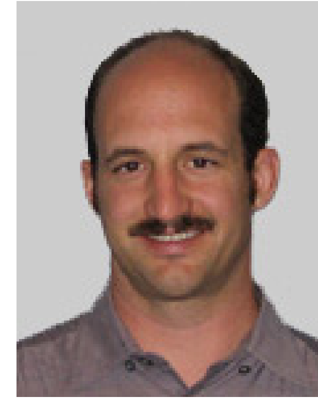
- **No consensus can be guaranteed in an asynchronous system in the presence of failures.**
- **<u>Intuition</u>: a "failed" process may just be slow, and can rise from the dead at exactly the wrong time.**
- **Consensus may occur recognizably, rarely or often.**



Network partition

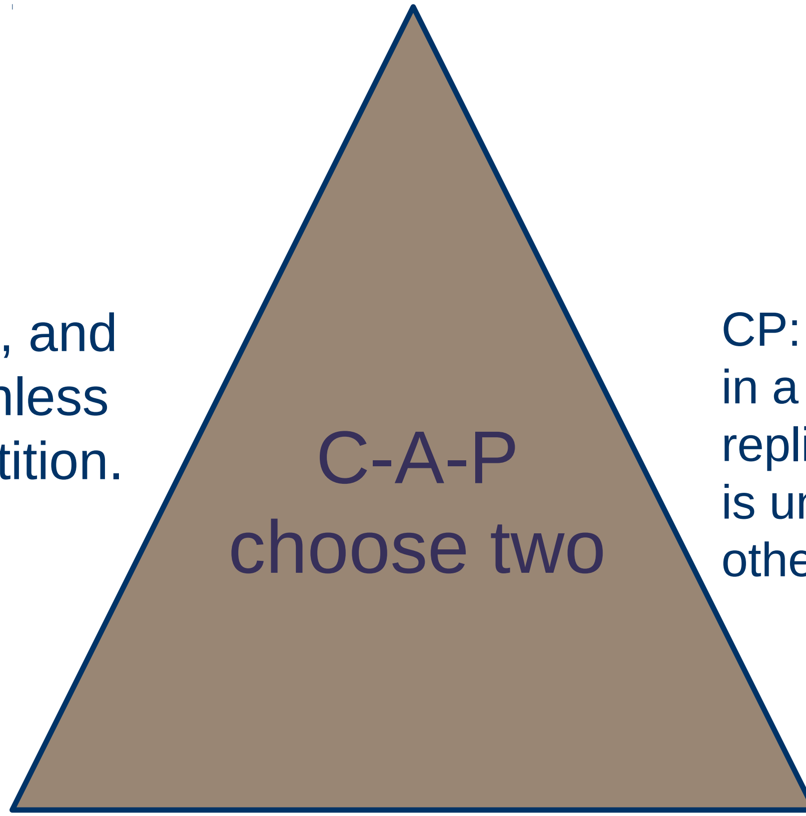


Split brain



consistency

C



CA: available, and consistent, unless there is a partition.

CP: always consistent, even in a partition, but a reachable replica may deny service if it is unable to agree with the others (e.g., quorum).

A

Availabilit
y

AP: a reachable replica provides service even in a partition, but may be inconsistent.

P

Partition-resilience

Coordination in Distributed Systems

- **Master coordinates, dictates consensus**
  - **e.g., lock service**
  - **Also called “primary”**
- **Remaining consensus problem: who is the master?**
  - **Master itself might fail or be isolated by a network partition.**
  - **Requires a high-powered distributed consensus algorithm (Paxos).**