# Latency Comparison

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

<table>
<thead>
<tr>
<th>Principal may do</th>
<th>Operation</th>
<th>on</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chase</td>
<td>Read</td>
<td></td>
<td>dFile</td>
</tr>
<tr>
<td>Alice</td>
<td>Pay invoice 4325</td>
<td>Account Q34</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>Fire three rounds</td>
<td>Bow gun</td>
<td></td>
</tr>
</tbody>
</table>

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

Reference monitor
Example: OS platform

Isolation boundary

Computersystem

principal

request

perform action

authentication module

authorization module

authorized?

yes/no

authentic?

yes/no

guard

OK

perform action

object

audit trail

log

Principles of Computer System Design © Saltzer & Kaashoek 2009
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.
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

Load (concurrent requests, or arrival rate)

Response arrival rate (offered load)

Response rate (throughput)

Response time

saturation

Ideal

Overload

Thrashing

Collapse

Request arrival rate (offered load)

[Von Behren]
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 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!
80% of the requests have response time $r$ with $x_1 < r < x_2$.

“Tail” of 10% of requests with response time $r > x_2$.

A few requests have very long response times.

What’s the mean $r$?

What’s the median?

10% quantile

50% cumulative distribution function (CDF)

90% quantile

$x_1$ $x_2$

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 simplify locking.