Intro to distributed systems

What is a distributed system?

A distributed system consists of hardware and software components located in a network of computers that communicate and coordinate their actions only by passing messages. [Coulouris]

A distributed system is a collection of independent computers that appears to its users as a single coherent system. [Tanenbaum & van Steen]

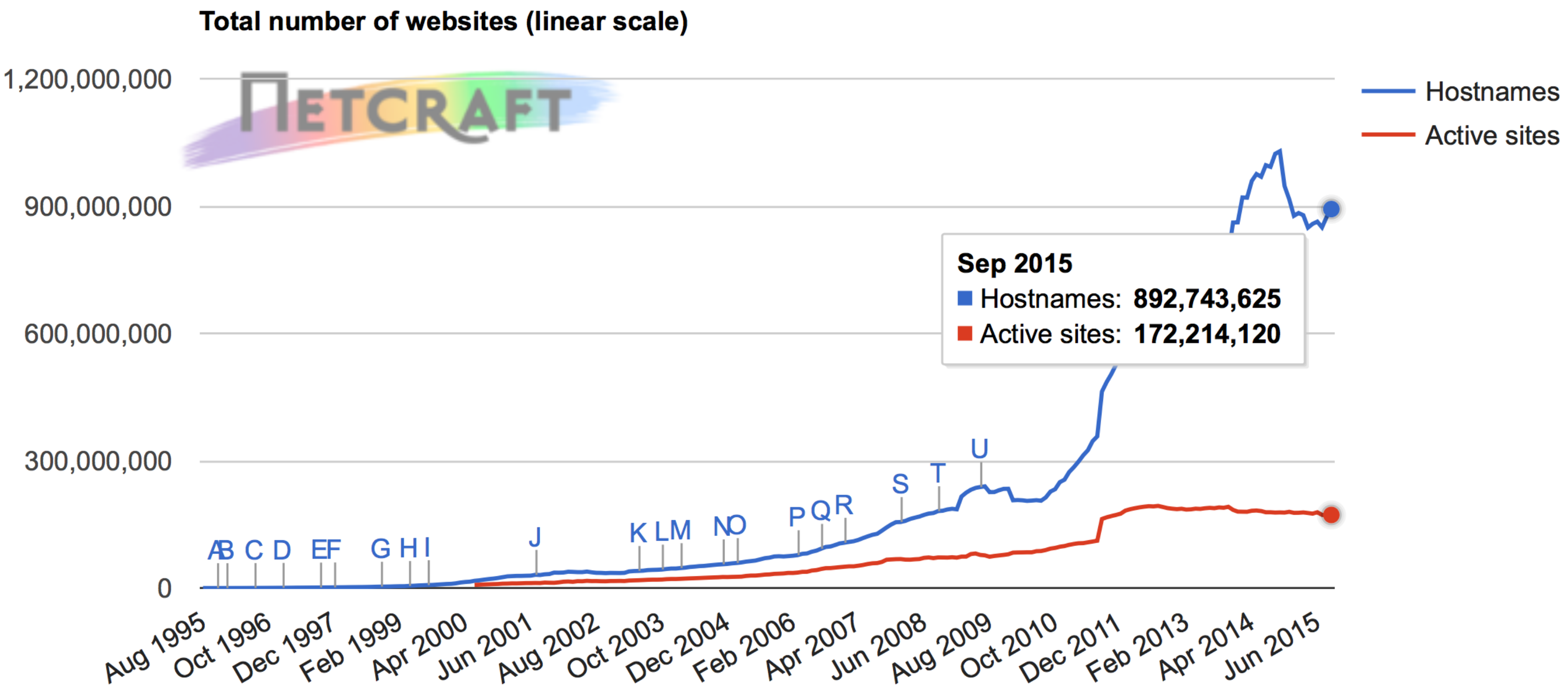
A distributed system is a system that prevents you from doing any work when a computer you have never heard about, fails. [Lamport]

- The above definitions take different perspectives
  - Operational perspective
  - User perspective
  - DS characteristics perspective

Examples of distributed systems

- Intra-net, inter-net, WWW
- DNS: Hierarchical distributed database
- Network of workstations (NOW), Cluster Computers
- Email
- Electronic banking
- Airline reservation system
- Peer-to-peer networks
- Sensor networks
- Mobile and Pervasive Computing
- Cellular phone systems
- IP Telephony
- Flight management system in an aircraft
- Automotive control systems (50+ embedded processors in a Mercedes S-class)
- Distributed file systems (NFS, Samba)
- P2P file sharing
- Etc., etc., etc.

The Internet



Total number of websites (linear scale)

Sep 2015
🟦 Hostnames: **892,743,625**
🟥 Active sites: **172,214,120**

Google

“Google is technologically a large supercomputer. It's a distributed supercomputer among many data centers doing all sorts of interesting things over fiber optic network that eventually are services available to end-users.” Eric Schmidt, Google CEO 2007

“Google runs on hundreds of thousands of servers—by one estimate, in excess of 450,000—racked up in thousands of clusters in dozens of data centers around the world. It has data centers in Dublin, Ireland; in Virginia; and in California, where it just acquired the million-square-foot headquarters it had been leasing. It recently opened a new center in Atlanta, and is currently building two football-field-sized centers in The Dalles, Ore.” 2006

An estimate today says that Google runs more than 2,000,000 server in 36 data centers around the globe

Google

"Our view is it's better to have twice as much hardware that's not as reliable than half as much that's more reliabl. You have to provide reliability on a software level. If you're running 10,000 machines, something is going to die every day.” Jeff Dean, Google fellow, 2008

“A typical search will require actions from between 700 to 1,000 machines today.” Maryssa Mayer, vice president of Google’s search products and user experience, 2008

Google processes over 40,000 queries every second on average, which translates to over 3.5 billion searches per day (internetlivestats)

“Our current generation — Jupiter fabrics — can deliver more than 1 Petabit/sec of total bisection bandwidth. To put this in perspective, such capacity would be enough for 100,000 servers to exchange information at 10Gb/s each, enough to read the entire scanned contents of the Library of Congress in less than 1/10th of a second.” Amin Vahdat, Google Fellow, 2015

Google data centers

E.g., Dallas site:
Three 68,000 square foot
data center buildings

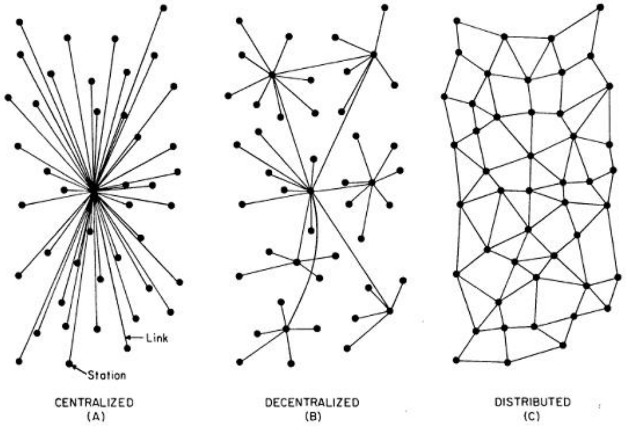
https://www.google.com/about/datacenters/

Distributed systems vs centralized systems

- Concurrency
  - In centralized systems, concurrency is a design choice.
  - In distributed systems, computers run concurrently.

- Independent and partial failures
  - Centralized systems usually fail completely.
  - Distributed systems usually fail partially, often because of communication. When a component fails, the others are still running. Detecting failures may be hard. Recovery is also hard because the state of an application is distributed.

- Absence of global clock
  - In centralized systems, the physical clock of the computer can be used for synchronization.
  - In distributed systems clocks may not be in sync
  - Example: Bank account, starting balance = $100
    - Client at bank machine A makes a deposit of $100
    - Client at bank machine B makes a withdrawal of $150
    - Which event happened first?
    - Should the bank charge the overdraft fee?



The eight fallacies of distributed systems

”Essentially everyone, when they first build a distributed application makes the following eight simplifying assumptions. All false in the long run (Peter Deutsch)”

1. The network is reliable.
2. The network is secure.
3. The network is homogeneous.
4. The topology does not change.
5. Latency is zero.
6. Bandwidth is infinite.
7. Transport cost is zero.
8. There is one administrator.

Building distributed systems is hard!

These assumptions ultimately prove false, resulting either in the failure of the system, a substantial reduction in system scope, or in large, unplanned expenses required to redesign the system to meet its original goals

Why Distributed Systems

- Functional distribution
  - Computers have different functional capabilities (e.g., File server, printer) yet may need to share resources
    - Client / server
    - Data gathering / data processing
- Incremental growth
  - Easier to evolve the system
  - Modular expandability
- Inherent distribution in application domain
  - Banks, reservation services, distributed games, mobile apps
  - physically or across administrative domains
  - cash register and inventory systems for supermarket chains
  - computer supported collaborative work

Why Distributed Systems

- Economics
  - collections of microprocessors offer a better price/ performance ratio than large mainframes.
  - Low price/performance ratio: cost effective way to increase computing power.
- Better performance
  - Load balancing
  - Replication of processing power
  - A distributed system may have more total computing power than a mainframe. Ex. 10,000 CPU chips, each running at 50 MIPS. Not possible to build 500,000 MIPS single processor since it would require 0.002 nsec instruction cycle. Enhanced performance through load distributing.
- Increased Reliability
  - Exploit independent failures property
  - If one machine crashes, the system as a whole can still survive.
- Another driving force: the existence of large number of personal computers, the need for people to collaborate and share information.

Goals and challenges of distributed systems

- Transparency
  - How to achieve the single-system image
- Performance
  - The system provides high (computing, storage, ..) performance
- Scalability
  - The ability to serve more users, provide acceptable response times with increased amount of data
- Openness
  - An open distributed system can be extended and improved incrementally
  - Requires publication of component interfaces and standards protocols for accessing interfaces
- Reliability / fault tolerance
  - Maintain availability even when individual components fail
- Heterogeneity
  - Network, hardware, operating system, programming languages, different developers
- Security
  - Confidentiality, integrity and availability

Transparency

- How to achieve the single-system image, i.e., how to make a collection of computers appear as a single computer.

- Hiding the distribution at two levels:
  - Hide the distribution **from users**
  - At a lower level, make the system look transparent **to programs**.

    -> Require uniform interfaces such as access to files, communication.

- Different forms of transparency in a DS (ISO, 1995).
- Trade-off between transparency and performance of a system

Transparency in distributed systems

Access transparency: enables local and remote resources to be accessed using identical operations.

- Dropbox
- SQL queries

Location transparency: enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).

- Users cannot tell where hardware and software resources such as CPUs, printers, files, data bases are located.
- Navigation in the web
- Tables in distributed database

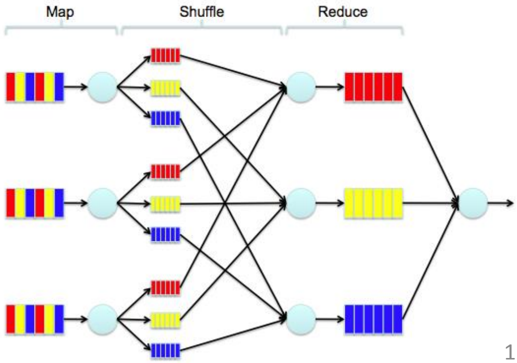
Migration Transparency: resources must be free to move from one location to another without their names changed.
E.g., /usr/lee, /central/usr/lee

Transparency in distributed systems

Concurrency transparency: enables several processes to operate concurrently using shared resources without interference.

- The users are not aware of the existence of other users.
- Need to allow multiple users to concurrently access the same resource. Lock and unlock for mutual exclusion.
- Distributed file system, distributed database

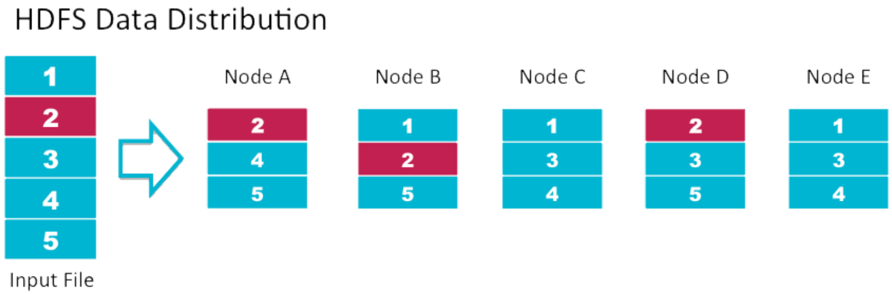
Parallelism Transparency: Automatic use of parallelism without having to program explicitly. The holy grail for distributed and parallel system designers.



Transparency in distributed systems

Replication transparency: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

- OS can make additional copies of files and resources without users noticing.



Failure transparency: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.

- DBMS, Big Data processing systems

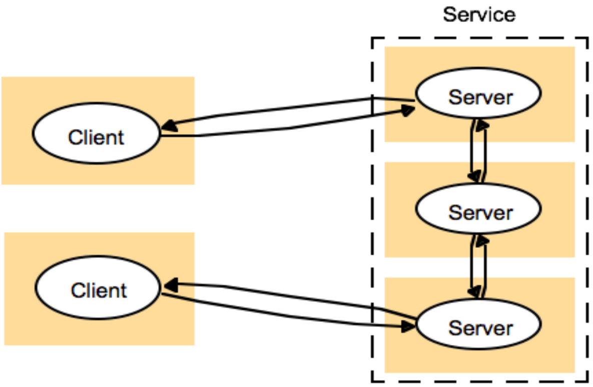
Transparency in distributed systems

Mobility transparency: allows the movement of resources and clients within a system without affecting the operation of users or programs.

- Roaming, moving between two access points.

Performance transparency: allows the system to be reconfigured to improve performance as loads vary.

Scaling transparency: allows the system and applications to expand in scale without change to the system structure or the application algorithms.



Transparency in distributed systems

In certain cases transparency is impracticable or not convenient

- Some things cannot be made transparent
  - Timezones
  - Communication delays

- Hiding too much may have a negative performance impact
  - Accessing multiple times a remote object without knowing

- Sometimes transparency is just undesirable
  - Users do not always want complete transparency:
    a fancy printer 1000 miles away

Reliability

- Hardware, software and network fail
- DS must maintain availability even in cases where hardware/software/network have low reliability
- Failures in distributed systems are partial
  - Makes error handling particularly difficult

- **Detection** of failures – may be impossible
  - In some cases it is easy, e.g., checksum in communication
  - Has a component crashed? Or is it just slow?
  - Is the network down? Or is it just slow?
  - If it’s slow – how long should we wait?

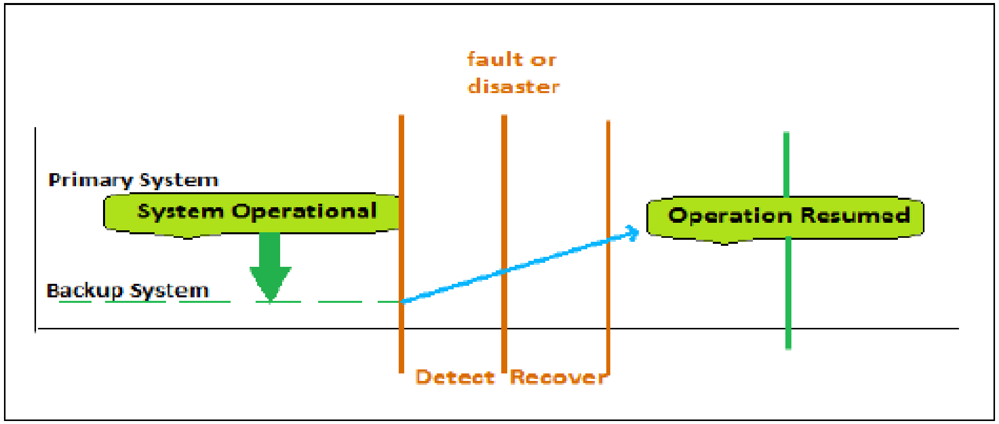
- Many techniques for **handling** failures
  - Masking failures (retransmission in protocols)
  - Tolerating failures, degrading the offered service (as in web-browsers)
  - Recovery from failures (periodically save state of a component, roll back partially completed task)
  - Redundancy (replicate servers in failure-independent ways, duplicate network routes)

Reliability

- Distributed system should be more reliable than single system.
- Example:
  - Single machine: 0.95 probability of being up.
  - System with 3 machines (all machines need to break): 1-.05**3 probability of being up.

Availability: fraction of time the system is usable.

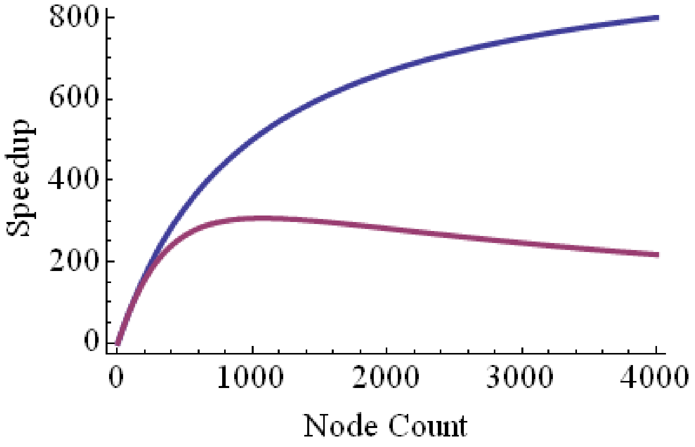
- Redundancy improves it
- Recovery between failures
  - Need to maintain consistency
  - Need to mask failures



Performance

- Without gain on this, why bother with distributed systems.
- Performance loss due to communication delays:
  - – fine-grain parallelism: high degree of interaction
  - – coarse-grain parallelism
- Performance loss due to making the system fault tolerant.

Scalability



- System remains effective as it grows?
- As you add more components:
  - More synchronization
  - More communication –> the system runs slowly.

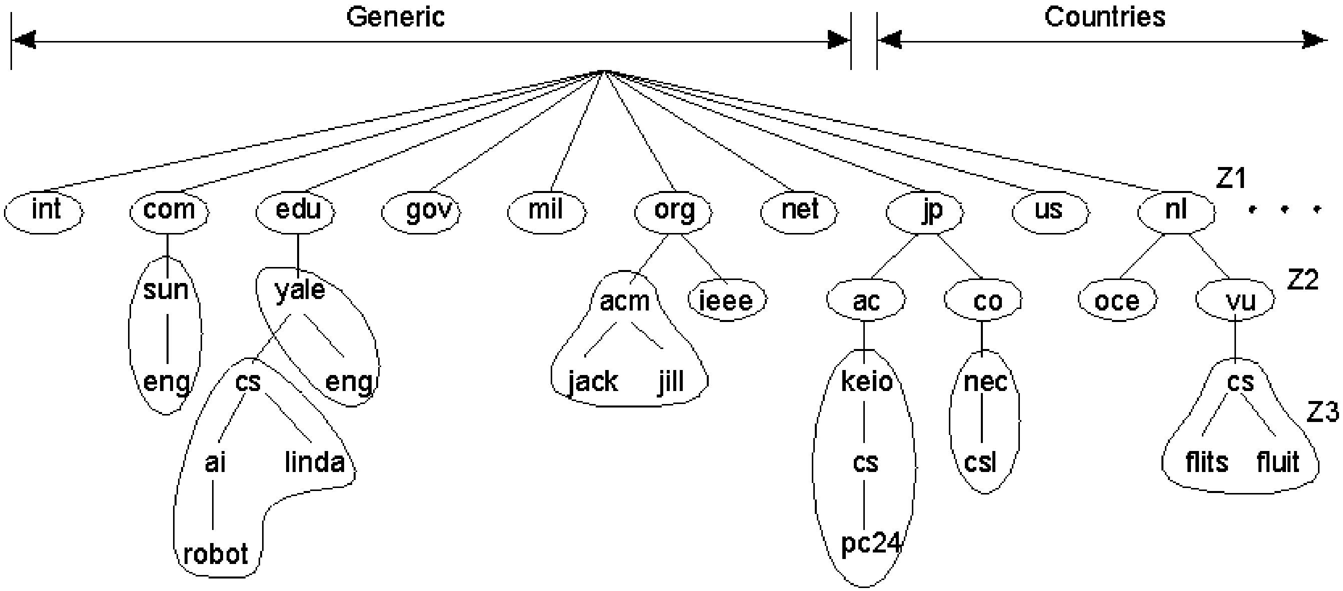
- A system is scalable if it remains effective when there is a significant increase in the amount of resources (data) and number of users
  - Internet: no of users and services has grown enormously
- Scalability denotes the ability of a system to handle an increasing future load

Scalability

- Requirements of scalability often leads to a distributed system architecture (several computers)
- Systems grow with time or become obsolete.
  - Techniques that require resources linearly in terms of the size of the system are not scalable.
  - E.g., broadcast based query won't work for large distributed systems.

- Examples of bottlenecks: Everyone is waiting for a single shared resource
  - Centralized services: a single mail server
  - Centralized data: a single URL address book
  - Centralized algorithms: routing based on complete information

Scaling techniques



Distribution

- Splitting a resource (such as data) into smaller parts, and spreading the parts across the system (cf DNS)

Scaling techniques: DNS

Initially, all host-addess mappings were in a file hosts.txt (in /etc/hosts)

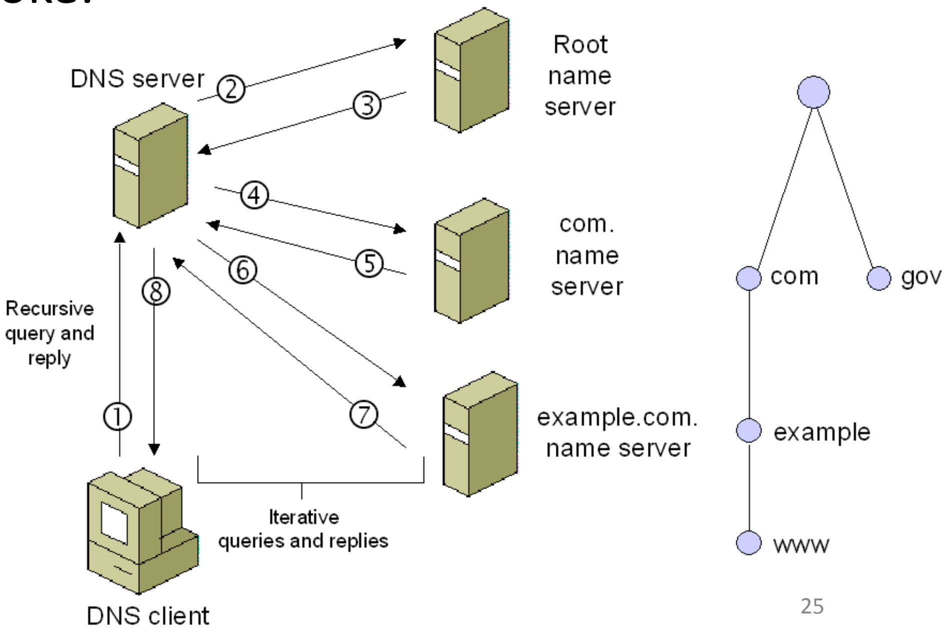
- Changes were submitted to SRI (Stanford Research Institute) by email
- New versions of hosts.txt ftp'd periodically from SRI
- An administrator could pick names at their discretion
- Any name is allowed: eugenesdesktopatrice (flat namespace)

As the internet grew this system broke:

- SRI couldn't handled the load
- Hard to enforce name uniqueness
- Many hosts: inaccurate hosts.txt

Domain Name System (DNS) was born in '83

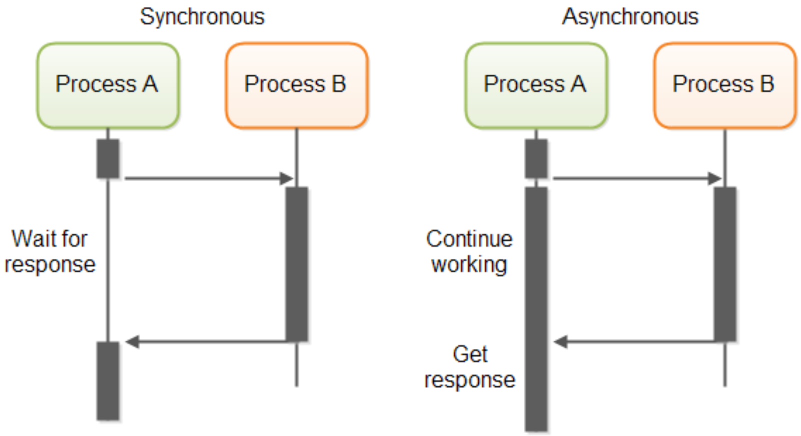
Recursive mode also possible.
What is the issue?



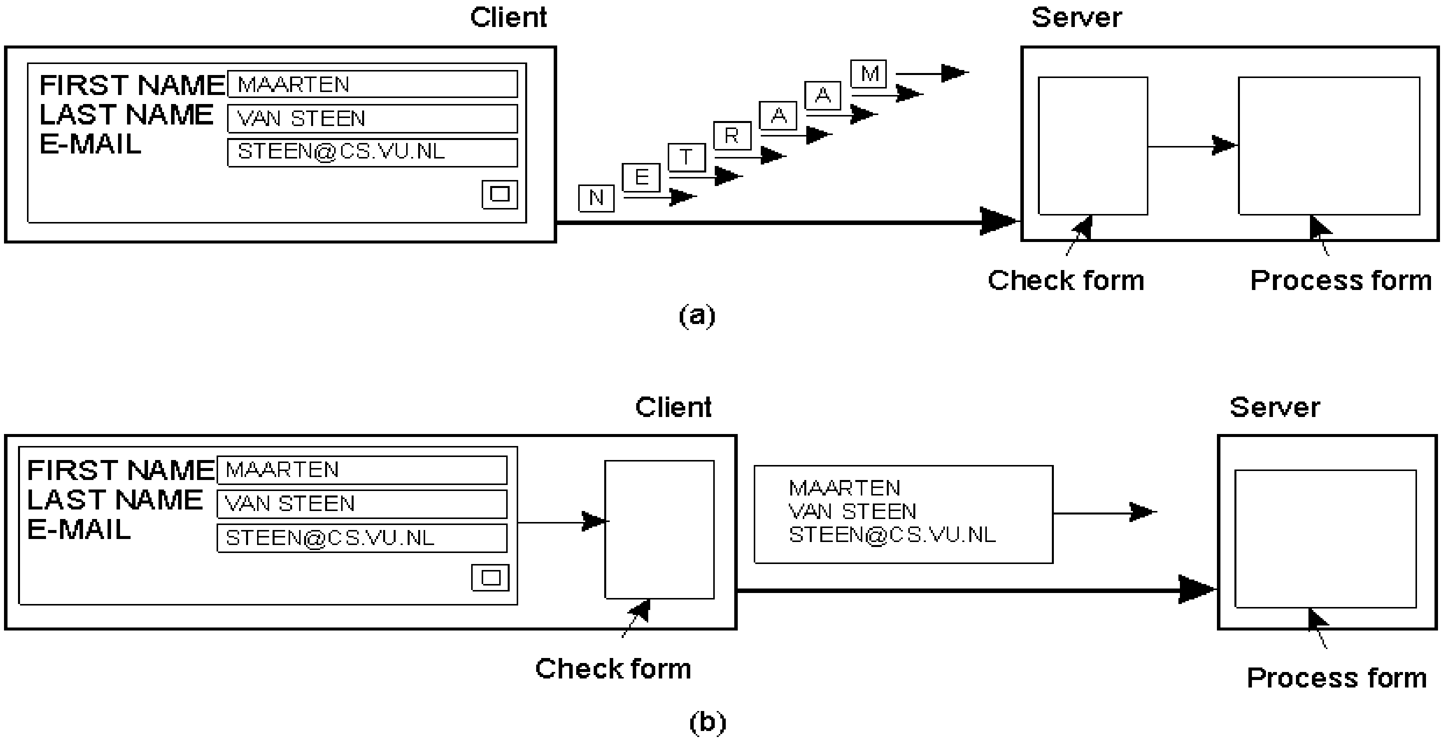
Scaling techniques

- Replication
  - Replicate resources (services, data) across the system, can access them in multiple places
  - Caching to avoid recomputation
  - Increased availability reduces the probability that a bigger system breaks

- Hiding communication latencies
  - Avoid waiting for responses to remote service requests
  - Use asynchronous communication



Scaling techniques



FIRST NAME MAARTEN
LAST NAME VAN STEEN
E-MAIL STEEN@CS.VU.NL

- Reducing amount of remote requests: The difference between letting (a) a server or (b) a client check forms as they are being filled.

Openness

- Can the systems be extended and reimplemented in various ways?

- To be achieved
  - Publish all key aspects of the system
    - Protocols
    - Interfaces to services
  - Adopting standards as much as possible
  - Take design decisions that favor interoperability and portabiliy

Example: The Internet. RFCs and an open standardization body (IETF)

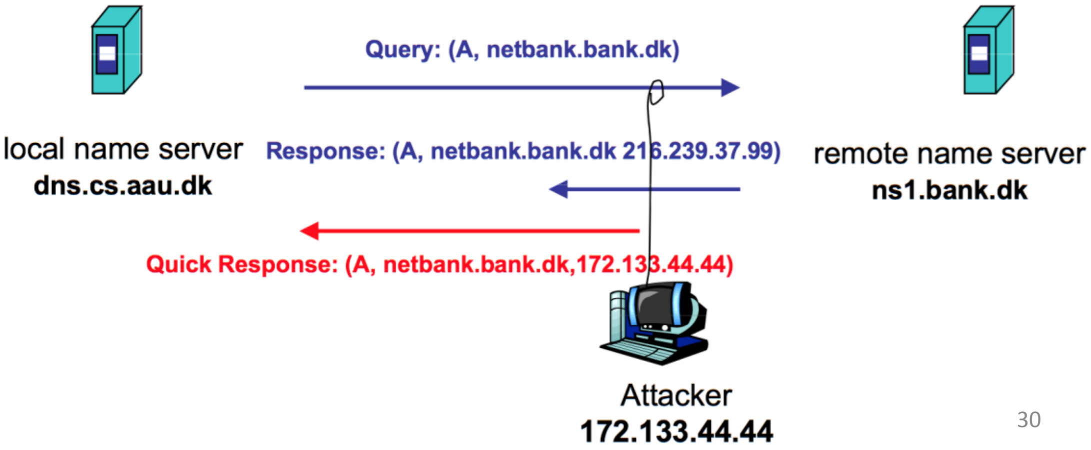
Heterogeneity

- Hardware and software (e.g., operating systems, processors)
  - How can an Intel/Windows system understand messages sent by an Macintosh OS X system?
  - Different performance. E.g. mobile devices have low computing power
- Different network infrastructures (Ethernet, 802.11 – wireless)
- Programming languages
  - How can a Java program and a C program communicate?

Security

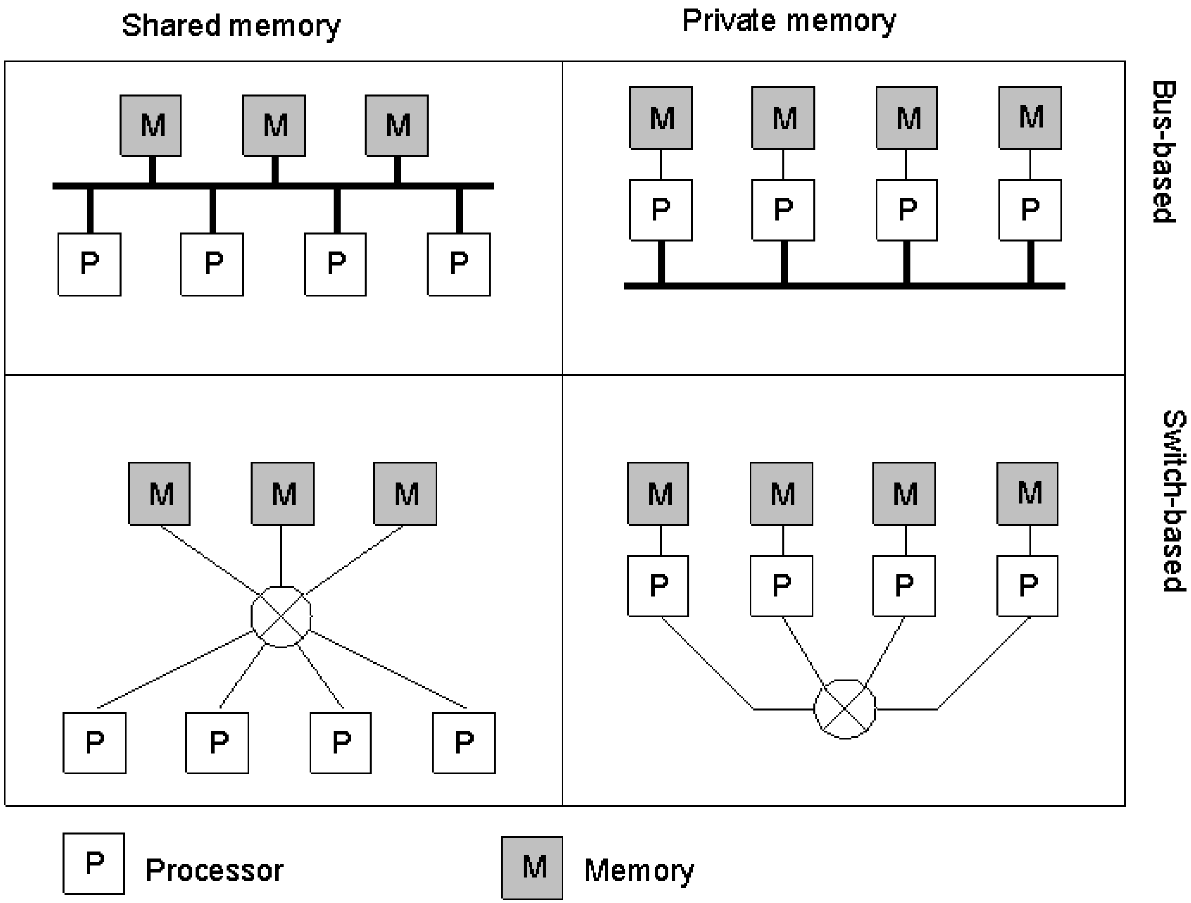
- Security for the information resources made available and maintained in the distributed system has three components
  - **Confidentiality**: Protection against disclosure to unauthorized individuals
  - **Integrity**: Protection against alteration or corruption
  - **Availability**: Protection against interference with the means to access the resource (e.g., DOS attack)
- Encryption is a powerful mechanism but several issues are still open
  - DOS attacks
  - Mobile code
  - …

Example:
DNS Spoofing

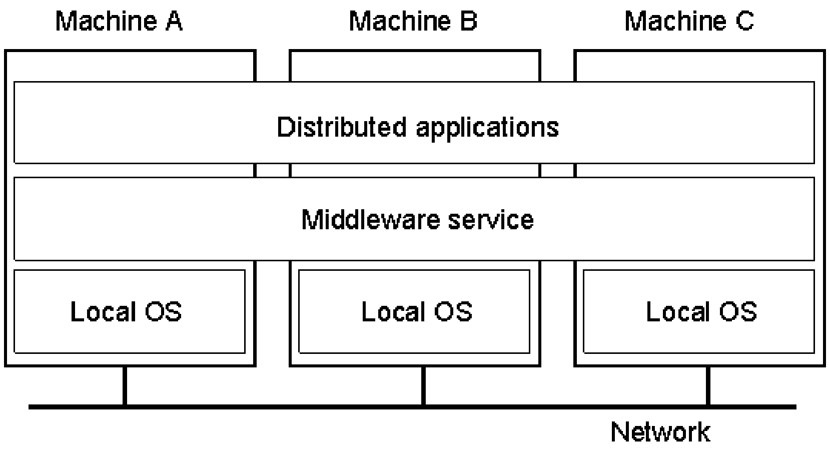

Query: (A, netbank.bank.dk)
local name server
dns.cs.aau.dk
Response: (A, netbank.bank.dk 216.239.37.99)
remote name server
ns1.bank.dk
Quick Response: (A, netbank.bank.dk,172.133.44.44)
Attacker
172.133.44.44

(More) Basic concepts

Parallel vs. distributed computing

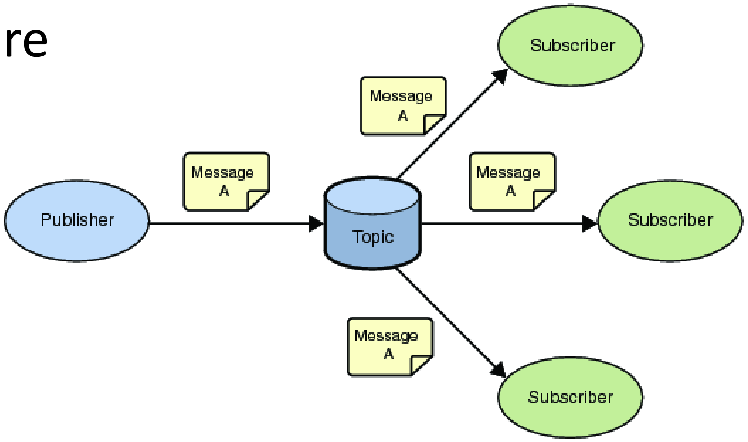


Middleware

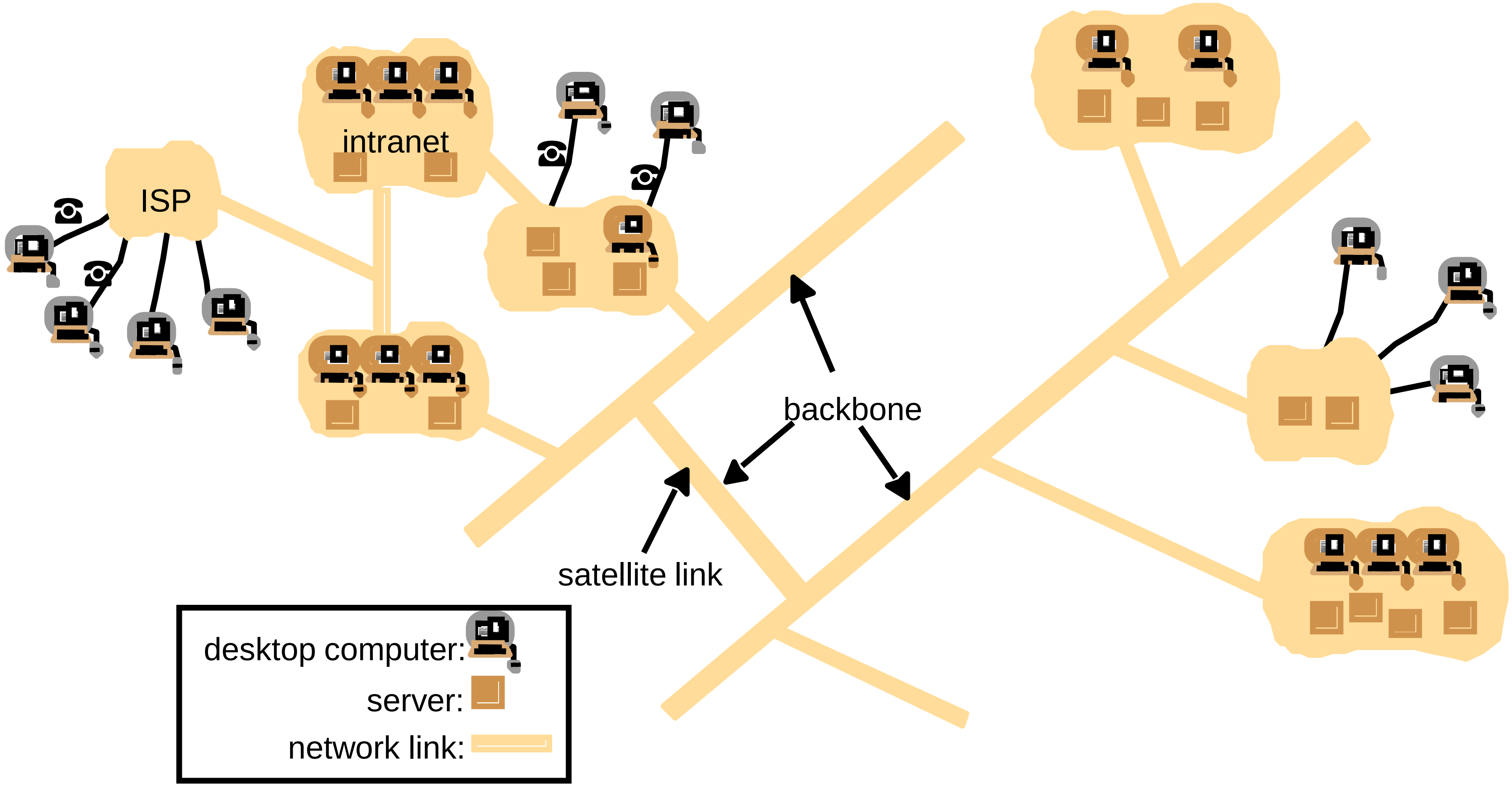


- Middleware provides horizontal services to help building distributed applications
  - It masks platforms differences

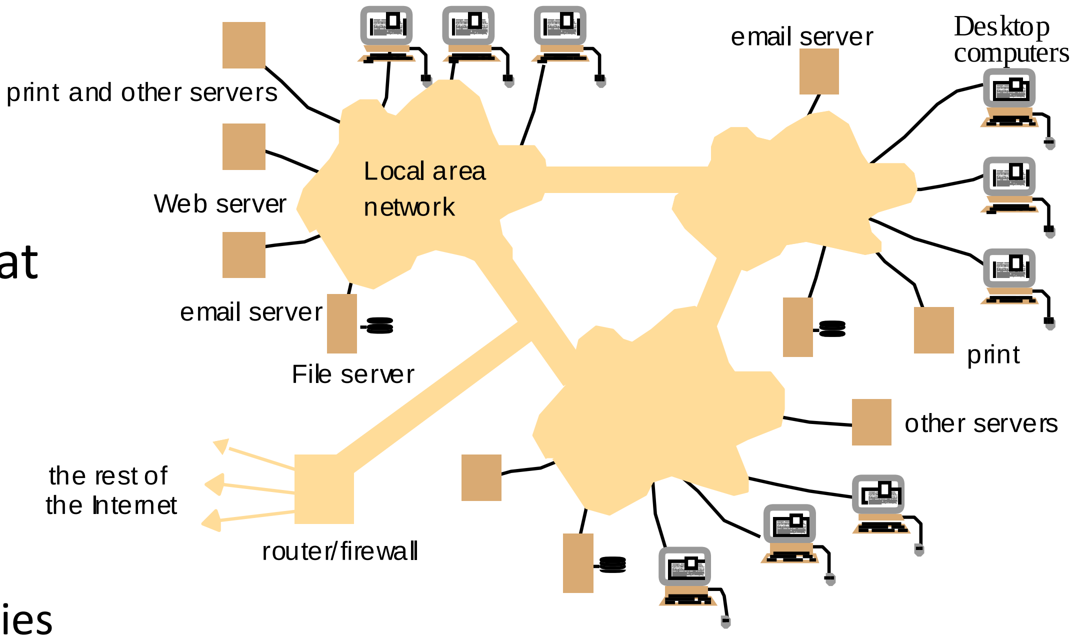
- Example: message oriented middleware
  - Store (buffer), route, or transform messages converting them from senders to receivers



Intranet: A portion of the Internet



Intranet



A portion of the Internet that

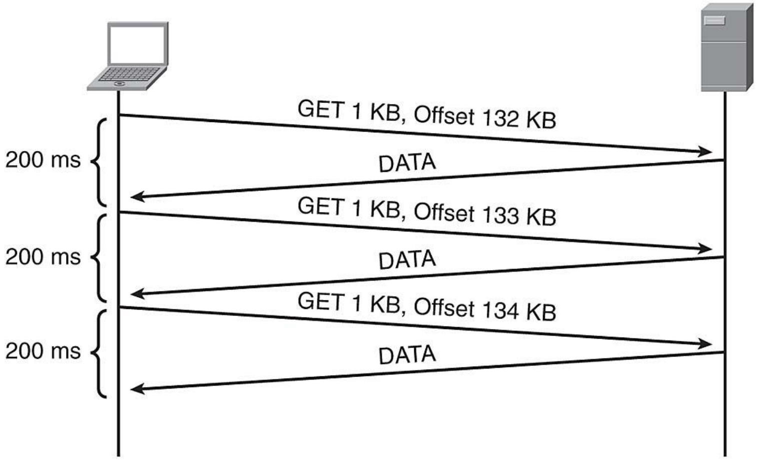
- is separately administered
- usually proprietary
- provides internal and external services
- can be configured to enforce local security policies
  - may use a firewall to prevent unauthorized messages leaving or entering
- may be connected to the internet via a router

Services:

- File, print services, backup, program-sharing, user-, system-administration, internet access

Throughput / Latency

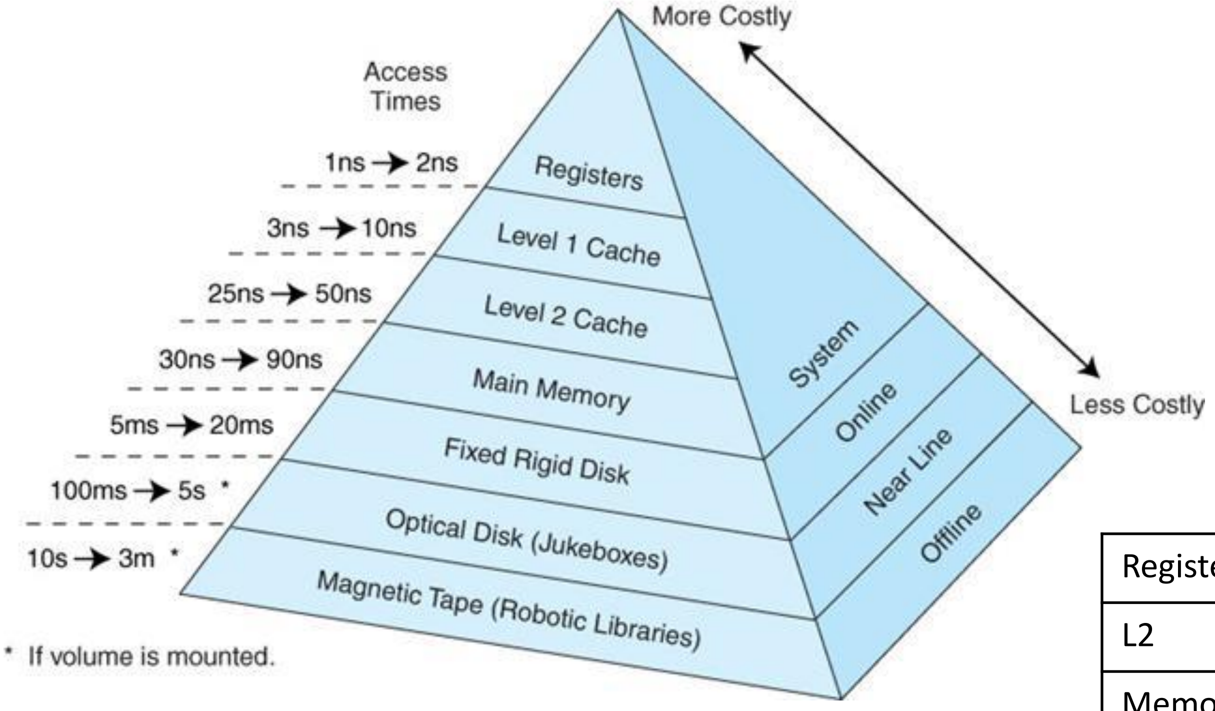
- Latency – “wire delay”
  - Time to send and recv one byte of data
  - Depends on “distance”



- Throughput
  - Bytes per second
  - Depends on the size of the vehicle

- Latency is often the bottleneck
  - Improves slower than bandwidth
  - Speed of light
  - Routes in the middle (traffic stops)
  - Request-respond cycle often dominates the application

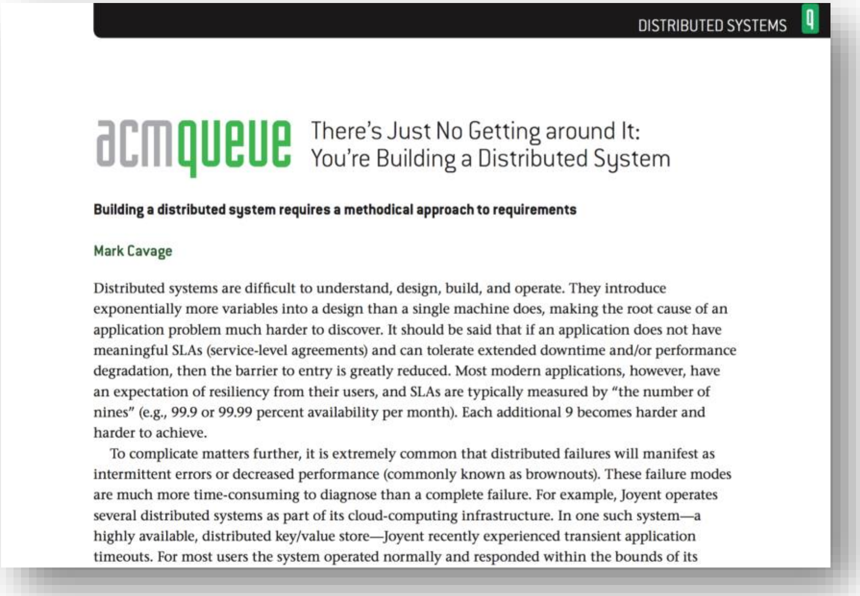
Performance scales



Register	1
L2	10
Memory	200
LAN	100,000
Disk	2,000,000
WAN	20,000,000

Exercise

Exercise



DISTRIBUTED SYSTEMS

acmqueue There's Just No Getting around It: You're Building a Distributed System

Building a distributed system requires a methodical approach to requirements

Mark Cavage

Distributed systems are difficult to understand, design, build, and operate. They introduce exponentially more variables into a design than a single machine does, making the root cause of an application problem much harder to discover. It should be said that if an application does not have meaningful SLAs (service-level agreements) and can tolerate extended downtime and/or performance degradation, then the barrier to entry is greatly reduced. Most modern applications, however, have an expectation of resiliency from their users, and SLAs are typically measured by "the number of nines" (e.g., 99.9 or 99.99 percent availability per month). Each additional 9 becomes harder and harder to achieve.

To complicate matters further, it is extremely common that distributed failures will manifest as intermittent errors or decreased performance (commonly known as brownouts). These failure modes are much more time-consuming to diagnose than a complete failure. For example, Joyent operates several distributed systems as part of its cloud-computing infrastructure. In one such system—a highly available, distributed key/value store—Joyent recently experienced transient application timeouts. For most users the system operated normally and responded within the bounds of its

Reading

- There's Just No Getting around It: You're Building a Distributed System [Mark Cavage]
- Answer questions in the exercise sheet available on the website
  - Open questions for discussion

Questions?

Sources & References

http://www.uio.no/studier/emner/matnat/ifi/INF5040/h07/undervisningsmateriale/INF5040_WK35_IntroDS.pdf

http://www.distributed-systems.net/index.php?id=distributed-systems-principles-and-paradigms

http://corsi.dei.polimi.it/distsys/materials.html