Distributed Computing Systems

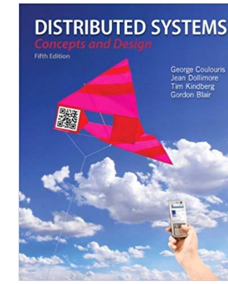
Prof. Nalini Venkatasubramanian
Dept. of Computer Science
Donald Bren School of Information and Computer Sciences
University of California, Irvine

Distributed Computing Systems
Winter 2021

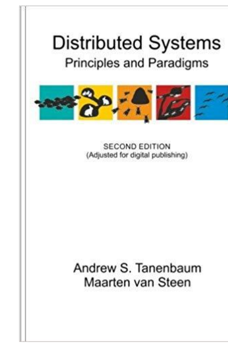
Lecture 1 - Introduction to Distributed Computing
CS 230: Mon/Wed 2 - 3:20pm (VRTL)
CS 230P: Mon/Wed 6:30 - 7:50 pm (VRTL)

Prof. Nalini Venkatasubramanian
nalini@uci.edu

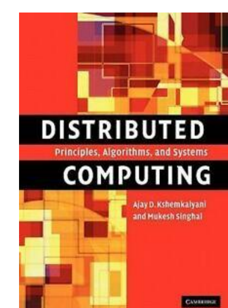
Course logistics and details



- Course Web page -
  - http://www.ics.uci.edu/~cs230
- Lectures
  - Mon/Wed 2:00 – 3:20 p.m, Virtual synchronous lecture
  - See webpage/canvas for zoom link



- Must Read: Course Reading List
  - **Collection of Technical papers and reports by topic**
- Reference Books (recommended)
  - **Distributed Systems: Concepts & Design**, 5th ed. by Coulouris et al.(preferred)
  - **Distributed Systems: Principles and Paradigms**, 2nd ed. by Tanenbaum & van Steen.
  - **Distributed Computing: Principles, Algorithms, and Systems**, 1st ed. by Kshemkalyani & Singhal.
- TA for Course
  - Praveen Venkateswaran(praveenv@uci.edu)



Course logistics and details

- Homeworks
  - Written homeworks
    - Problem sets
    - Includes paper summaries (1-2 papers on the specific topic from the reading list)
- Course Examination (tentatively Week 9)
- Course Project
  - In groups of 3
  - Will require use of open source distributed computing platforms
  - Suggested projects will be available on webpage

Prerequisite Knowledge

- Necessary – Operating Systems Concepts and Principles, basic computer system architecture
- Highly Desirable – Understanding of Computer Networks, Network Protocols
- Necessary – Basic programming skills in Java, Python, C++,…

CompSci 230 Grading Policy

- Homeworks - 40% of final grade
  - **4 homeworks - one for each segment of the course**
    - Problem sets, paper summaries (2 in each set)
  - **A homework due approximately every 2 weeks**
  - **Make sure to follow instructions while writing and creating summary sets.**
- Course Exam – 30% of final grade
- Class Project - 30% of final grade
  - Part 1: Due Week 6
  - Part 2: Due Finals Week
- Final assignment of grades will be based on a curve.

Syllabus and Lecture schedule

- Part 0 - Introduction to Distributed Systems
- Part 1: Time and State in Distributed Systems
  - Physical Clocks, Logical Clocks, Clock Synchronization
  - Global Snapshots and State Capture
- Part 2: From Operating Systems to Distributed Systems
  - Architectural Possibilities, Communication Primitives (Distributed Shared Memory, Remote Procedure Calls)
  - Distributed Coordination (mutual exclusion, leader election, deadlocks)
  - Scheduling and Load Balancing in distributed systems
  - Distributed Storage and FileSystems
- Part 3: Messaging and Communication in Distributed Systems
  - ALM. Mesh/Tree Protocols, Group Communication, Distributed Publish/Subscribe
- Part 4: Reliability and Fault Tolerance in Distributed Systems
  - Fault Tolerance, Consensus, Failure Detection, Replication, Handling Byzantine Failures

Wk	Dates	Lecture Topic	Deadlines for activities
1	**Jan 4, 6**	Introduction to distributed systems and models	**Project group formation (set up AWS accounts)**
2	**Jan 11, 13**	Time in Distributed Systems (Physical/Logical Clocks, Clock Synchronization)	**Project proposal: Jan 15** **(Lab: Hadoop intro and setup tutorial)** **HW 1 released**
3	**Jan 18 (holiday), 20**	Global State in Distributed Systems	**Homework 1 due Jan 23**
4	**Jan 25, 27**	Global State (cont), Distributed Coordination - RPC, DSM, Distributed Mutual Exclusion, Deadlocks	**Hands-on Project Step 1: due Jan 29** **HW 2 released**
5	**Feb 1, 3**	Distributed Resource management Scheduling,Migration, Load Balancing,	**Homework 2 due Feb 6**
6	**Feb 8,10**	Distributed FileSystems Group Communication,	**Hands-on Project Step 2: due Feb 16**
7	**Feb 15(holiday), 17**	ALM, Publish/Subscribe, Fault Tolerance	**Project Step 3 meetings** **HW 3 released**
8	**Feb 22, 24** **Feb 26 (CS Seminar)**	**Fault Tolerance, Failure Detection**	**Homework 3 due Feb 27** **Project Step 3 meetings**
9	**Mar 1, 3**	Course Exam, Consensus	**Project update** **HW 4 released**
10	**Mar 8, 10**	Replication, Replicated State Management	**Homework 4 due Mar 12**
11	**Mar 15-19**	**Project demos, reports, slides due**	

Lecture Schedule

- **Week 1 (Part 0): Distributed Systems Introduction**
  - Needs/Paradigms
  - Basic Concepts and Terminology, Concurrency
- **Weeks 2,3 (Part 1): Time and State in Distributed Systems**
  - Physical and Logical Clocks
  - Distributed Snapshots and State Capture
- **Week 4,5,6: Distributed Coordination and Resource Management**
  - Interprocess Communication
    - Remote Procedure Calls, Distributed Shared Memory
  - Distributed Process Coordination/Synchronization
    - Distributed Mutual Exclusion/Deadlocks, Leader Election
  - Distributed Process and Resource Management
    - Task Migration, Load Balancing
  - Distributed I/O and Storage Subsystems
    - Distributed FileSystems

Lecture Schedule

- Weeks 7,8: Messaging and Communication in Distributed Systems
  - **Messaging in Distributed Systems, ALM**
  - **Group Communication and Synchrony**
  - **Publish/Subscribe Based Communication**
- Weeks 9,10: Fault Tolerance in Distributed Systems
  - **Failure Models**
  - **Fault Detection**
  - **Consensus**
  - **Replication, Replicated State Machines**

What is not covered

- Security in Distributed Systems (Prof. Tsudik)
- Distributed Database Management and Transaction Processing (CS 223, Prof. Mehrotra)
- Distributed Objects and Middleware Platforms (CS237 - Spring Quarter 2020, Prof. Nalini)

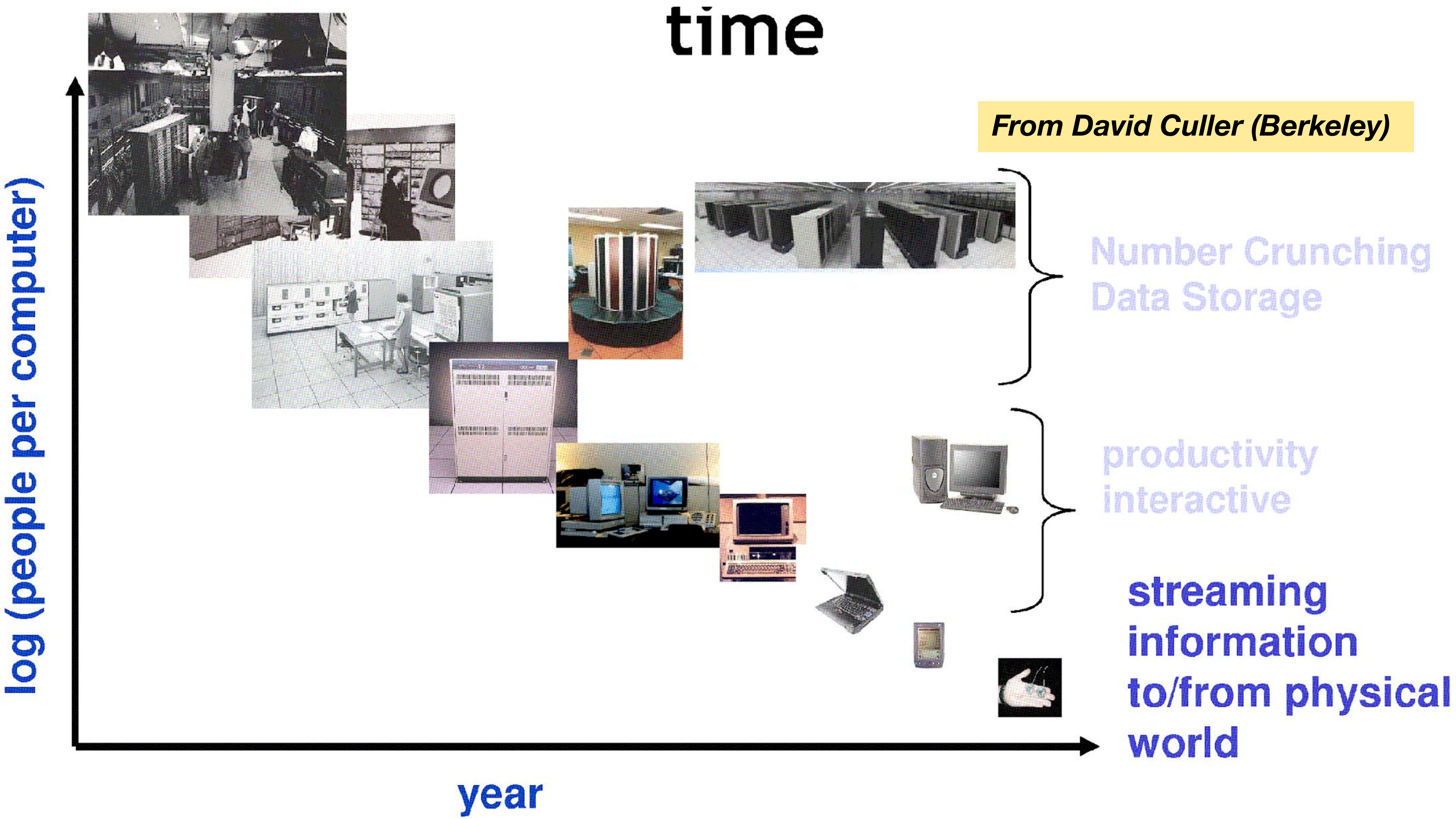
Distributed Systems

- **Lamport's Definition**
  - " You know you have one when the crash of a computer you have never heard of stops you from getting any work done."
  - "A number of interconnected autonomous computers that provide services to meet the information processing needs of modern enterprises."
- **Andrew Tanenbaum**

  A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
- "An interconnected collection of autonomous processes" - Wak Fokknik (an algorithmic view)
- **FOLDOC (Free on-line Dictionary) -??**

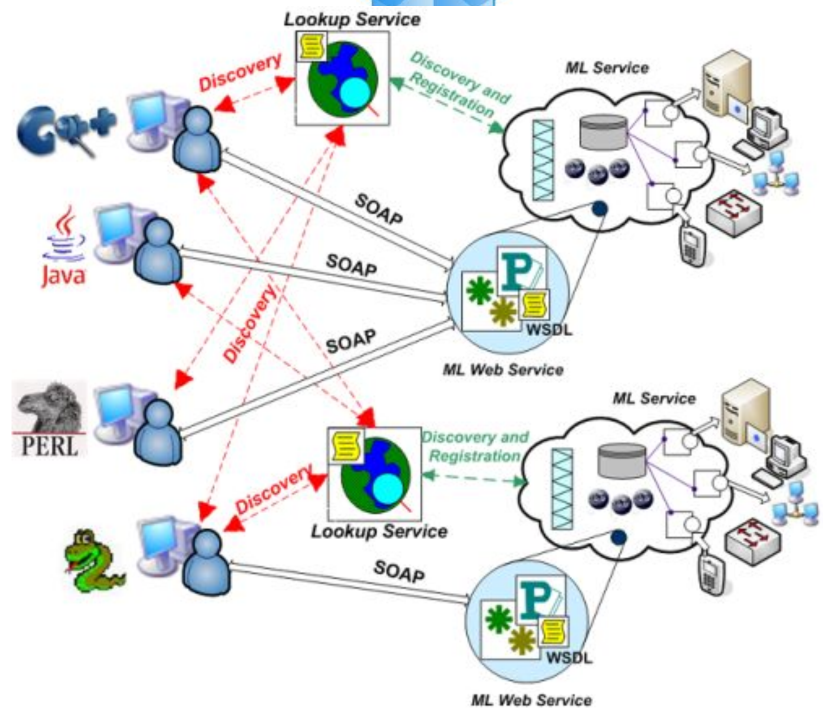
  A collection of (probably heterogeneous) automata whose distribution is transparent to the user so that the system appears as one local machine. This is in contrast to a network, where the user is aware that there are several machines, and their location, storage replication, load balancing and functionality is not transparent. Distributed systems usually use some kind of "client-server organization"

People-to-Computer Ratio Over Time

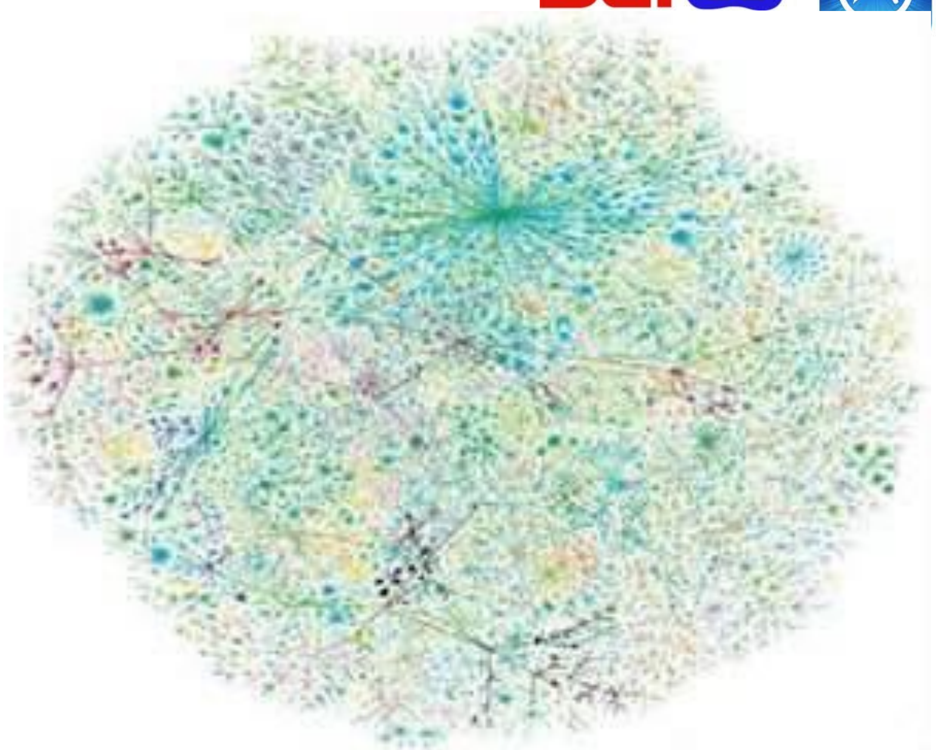


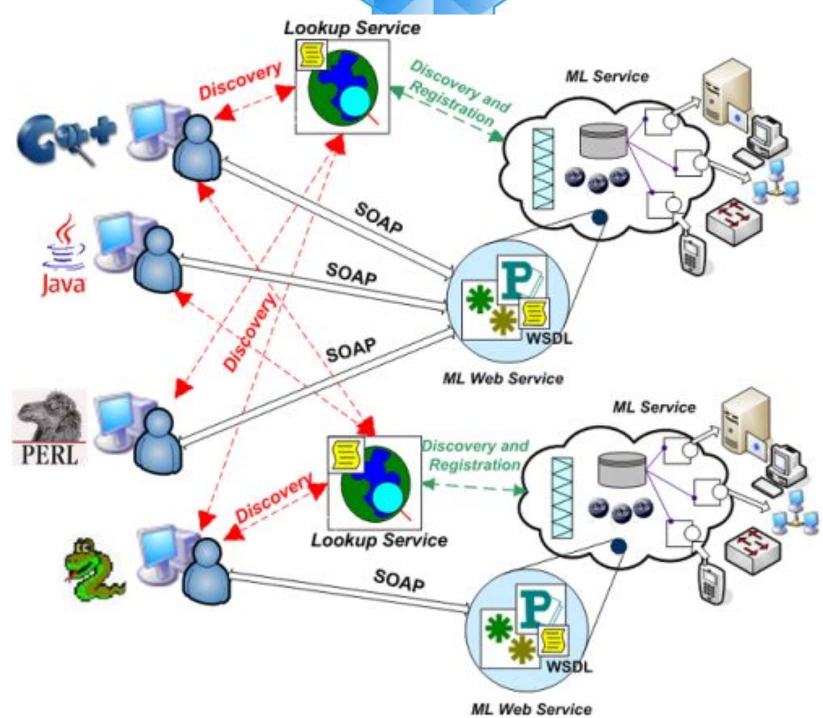
What is a Distributed System?

What is a Distributed System?



What is a Distributed System?

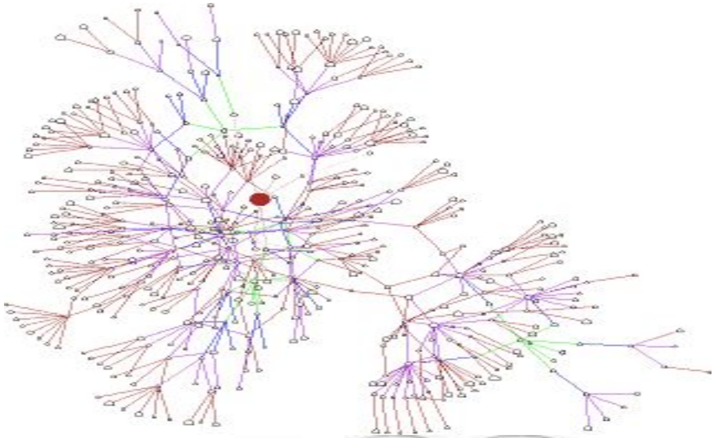


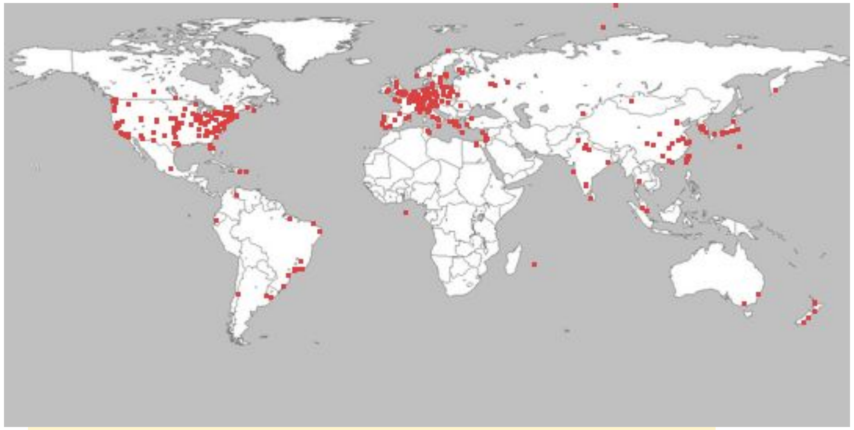


Distributed Computing Systems

Globus Grid Computing Toolkit

Gnutella P2P Network





PlanetLab

Cloud Computing Offerings

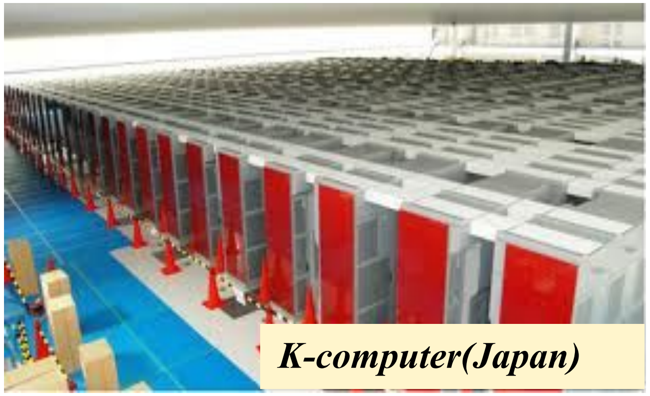
Parallel Systems

- Multiprocessor systems with more than one CPU in close communication.
- Improved Throughput, increased speedup, increased reliability.
- Kinds:
  - Vector and pipelined
  - Symmetric and asymmetric multiprocessing
  - Distributed memory vs. shared memory
- Programming models:
  - Tightly coupled vs. loosely coupled ,message-based vs. shared variable

Parallel Computing Systems

ILLIAC 2 (UIllinois)

Climate modeling, earthquake simulations, genome analysis, protein folding, nuclear fusion research, …..



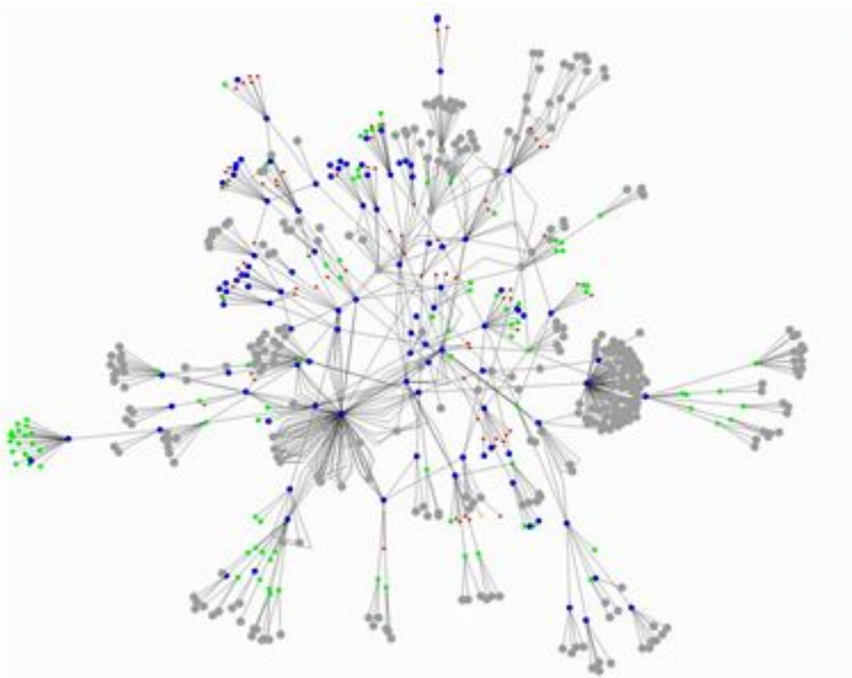
K-computer(Japan)

Connection Machine (MIT)

Tianhe-1(China)

IBM Blue Gene

Peer to Peer Systems



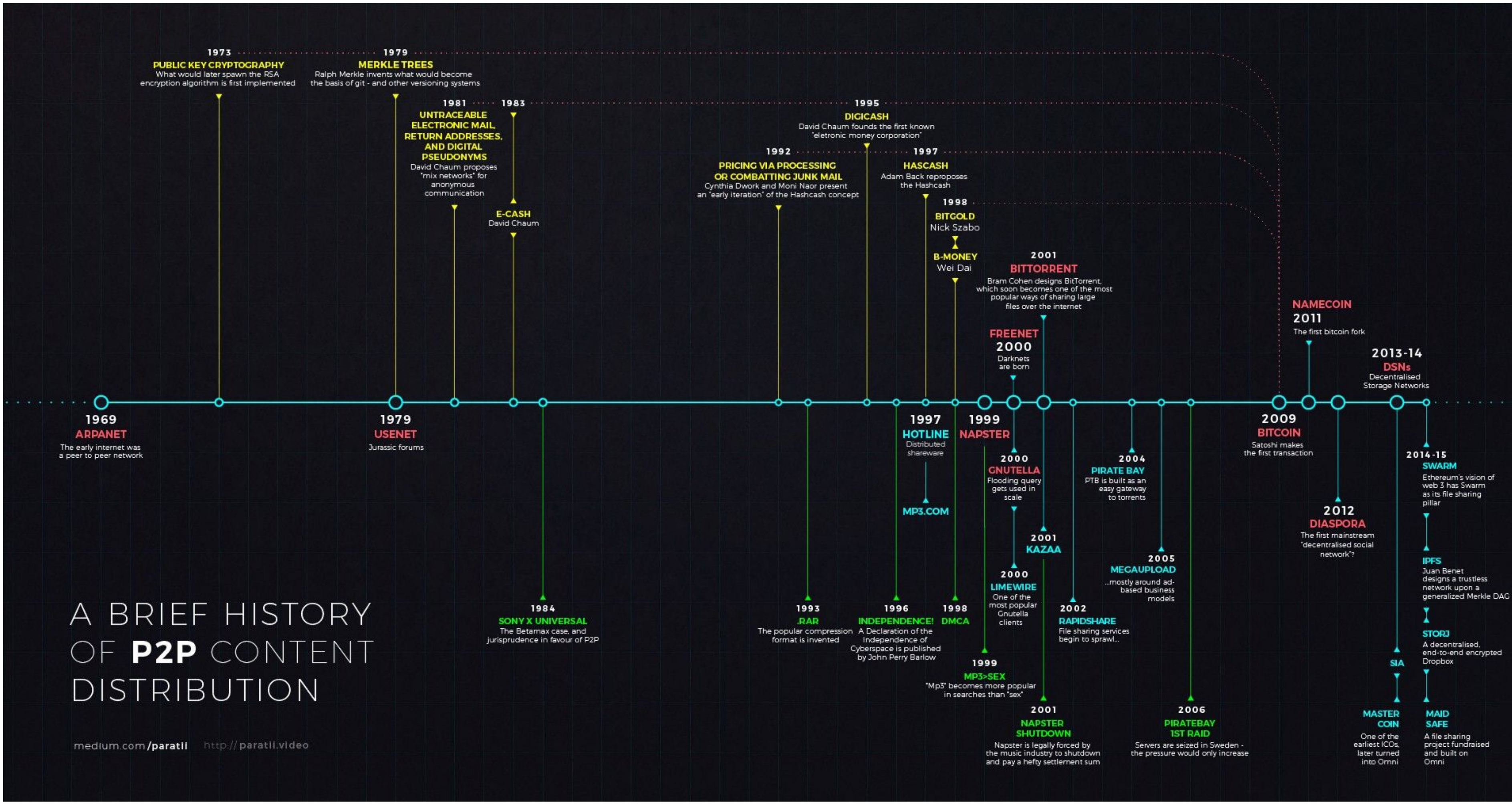
P2P File Sharing
- Napster, Gnutella, Kazaa, eDonkey, BitTorrent
- Chord, CAN, Pastry/Tapestry, Kademlia

P2P Communications
- MSN, Skype, Social Networking Apps

P2P Distributed Computing
- Seti@home

Use the vast resources of machines at the edge of the Internet to build a network that allows resource sharing without any central authority.

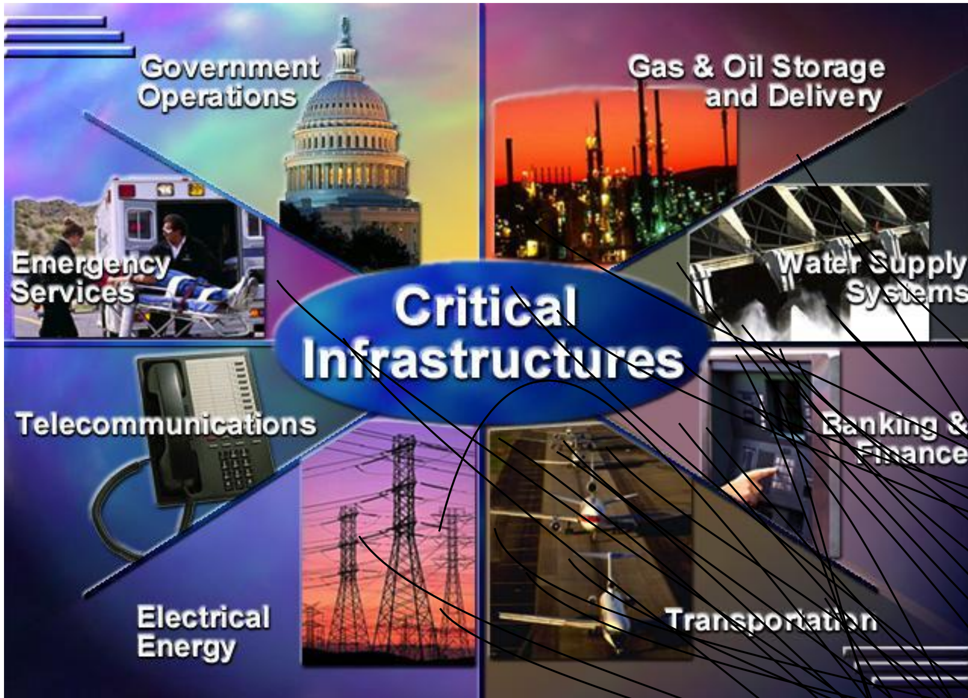
P2P: Napster to BitCoin



Real-time distributed systems

- Correct system function depends on timeliness
- Feedback/control loops
- Sensors and actuators
- Hard real-time systems -
  - Failure if response time too long.
  - Secondary storage is limited
- Soft real-time systems -
  - Less accurate if response time is too long.
  - Useful in applications such as multimedia, virtual reality.

New application domains

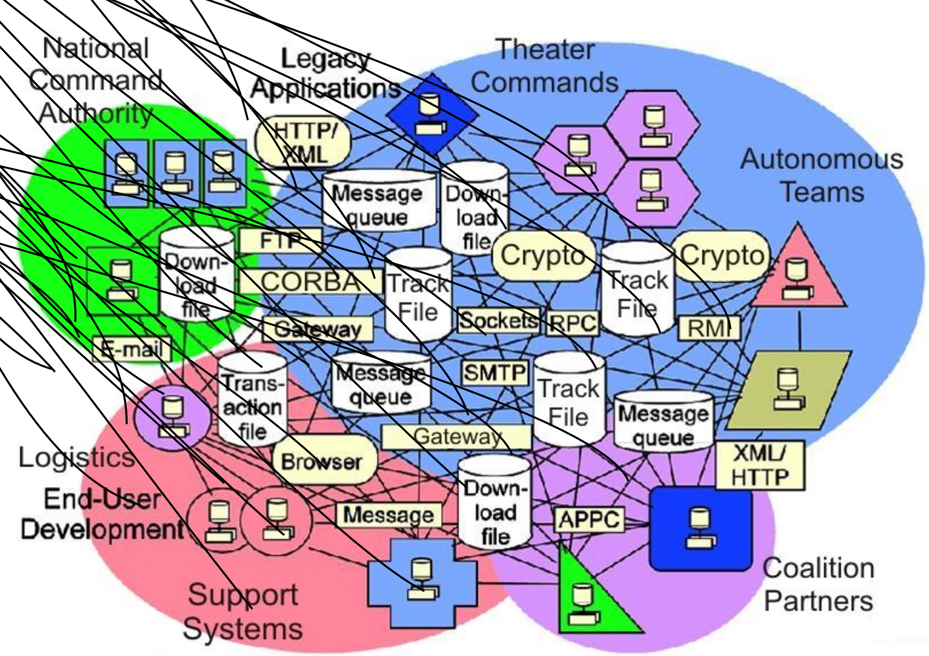


Key problem space challenges

- Highly dynamic behavior
- Transient overloads
- Time-critical tasks
- Context-specific requirements
- Resource conflicts
- Interdependence of (sub)systems
- Integration with legacy (sub)systems

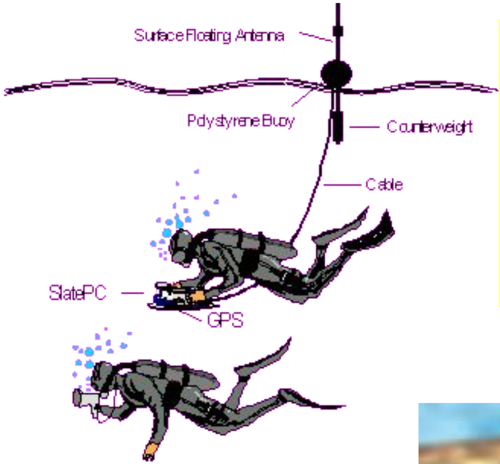
Key solution space challenges

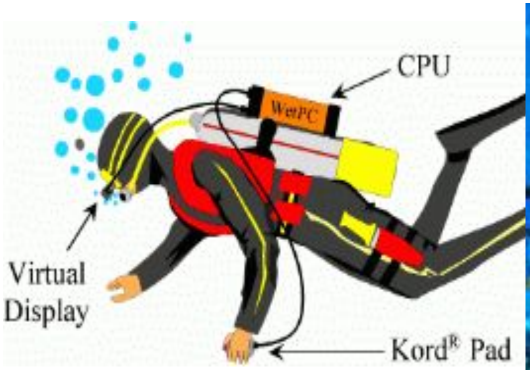
- Enormous accidental & inherent complexities
- Continuous evolution & change
- Highly heterogeneous platform, language, & tool environments

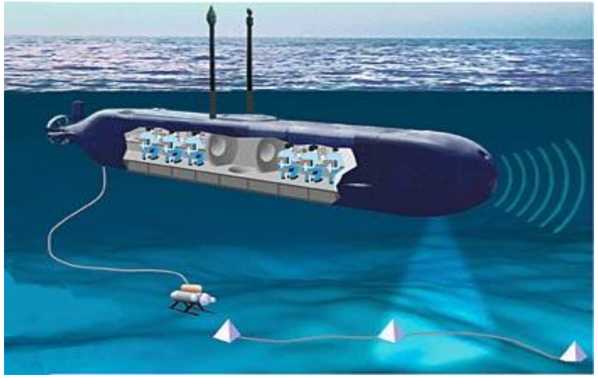


Mapping problem space requirements to solution space artifacts is very hard!

Mobile & ubiquitous distributed systems

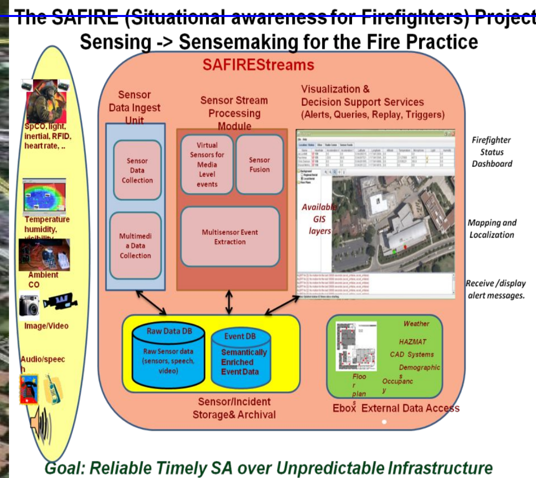




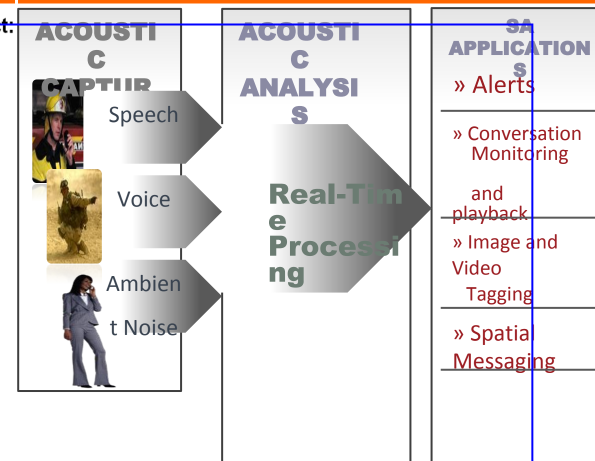


Responsphere - A Campus-wide infrastructure to instrument, monitor, disaster drills & technology validation

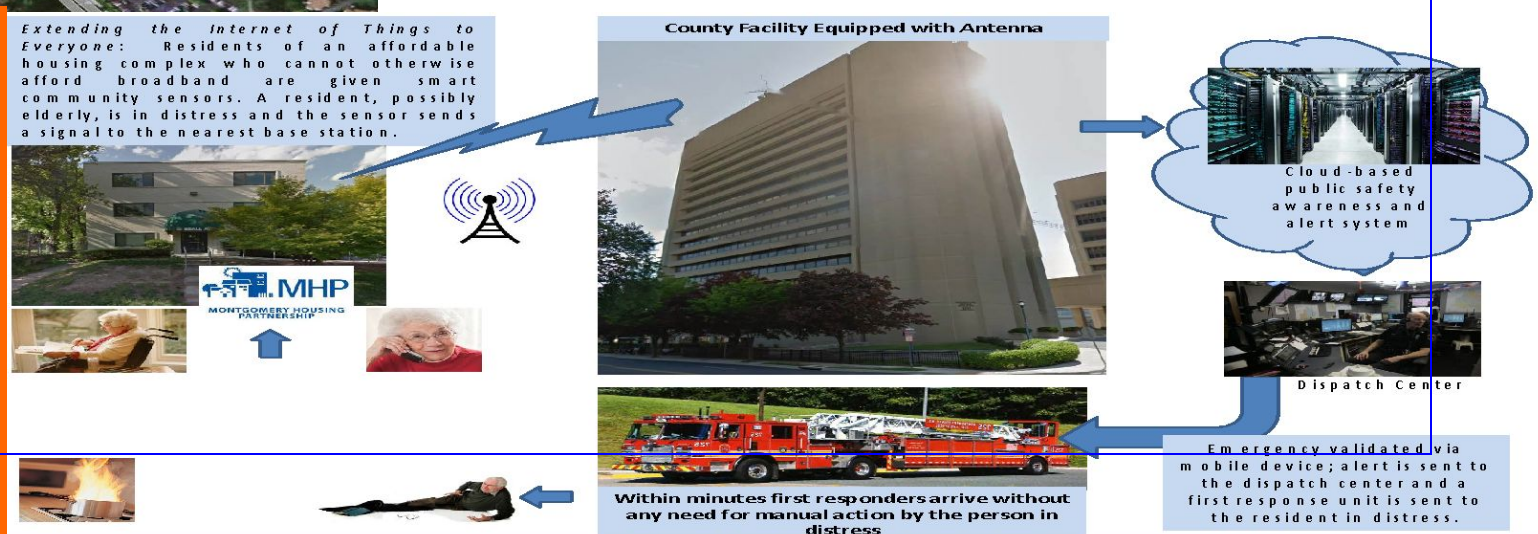
SAFIRE – Situational awareness for fire incident command



OpsTalk– Speech based awareness & alerting system for soldiers on the field



SCALE – A smart community awareness and alerting testbed @ Montgomery County, MD. A *NIST/Whitehouse SmartAmerica* Project extended to *Global Cities* Challenge.



Today's Platforms Landscape - examples

System	Goal
BitTorrent	swarm-style (unstructured peer-oriented) downloads - used in Twitter datacenter
Memcached	A massive key-value store
Hadoop (+ HDFS)	Reliable, scalable, high-performance distirbuted computing platform for data reduction
MapReduce	Programming massively parallel/distributed applications
Spark	Programming massively parallel/distributed real-time applications
Zookeeper	Support for coordination in distributed clusters
Spanner	Globally distributed database solution/storage service
Storm	**Dealing with Stream Data processing**
Dynamo	Amazon's massively replicated key-value store
Spread	Group communication and replicated data

Distributed Systems

Hardware – ***very cheap*** ; **Human – *very expensive***

Characterizing Distributed Systems

- **Multiple Autonomous Computers**
  - each consisting of CPU's, local memory, stable storage, I/O paths connecting to the environment
  - Multiple architectural possibilities
    - client/server, peer-oriented, cloud computing, edge-cloud continuum
  - Distribute computation among many processors.
  - Geographically Distributed
- **Interconnections**
  - some I/O paths interconnect computers that talk to each other
  - Various communication possibilities
- **Shared State**
  - No shared physical memory - loosely coupled
  - Systems cooperate to maintain shared state
  - Maintaining global invariants requires correct and coordinated operation of multiple computers.

Why Distributed Computing?

- Inherent distribution
  - Bridge customers, suppliers, and companies at different sites.
  - remote data access - e.g. web
- Support for interaction - email/messaging/social media
- Computation Speedup - improved performance
- Fault tolerance and Reliability
- Resource Sharing
  - Exploitation of special hardware
- Scalability
- Flexibility

Why are Distributed Systems Hard?

- Scale
  - numeric, geographic, administrative
- Loss of control over parts of the system
- Unreliability of message passing
  - unreliable communication, insecure communication, costly communication
- Failure
  - Parts of the system are down or inaccessible
  - Independent failure is desirable

The Eight Fallacies of Distributed Computing

Peter Deutsch

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause *big* trouble and *painful* learning experiences.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

For more details, read the article by Arnon Rotem-Gal-Oz

An entertaining talk: https://www.youtube.com/watch?v=JG2ESDGwHHY

Design goals of a distributed system

- Sharing
  - HW, SW, services, applications
- Openness(extensibility)
  - use of standard interfaces, advertise services, microkernels
- Concurrency
  - compete vs. cooperate
- Scalability
  - avoids centralization
- Fault tolerance/availability
- Transparency
  - location, migration, replication, failure, concurrency

Modeling Distributed Systems

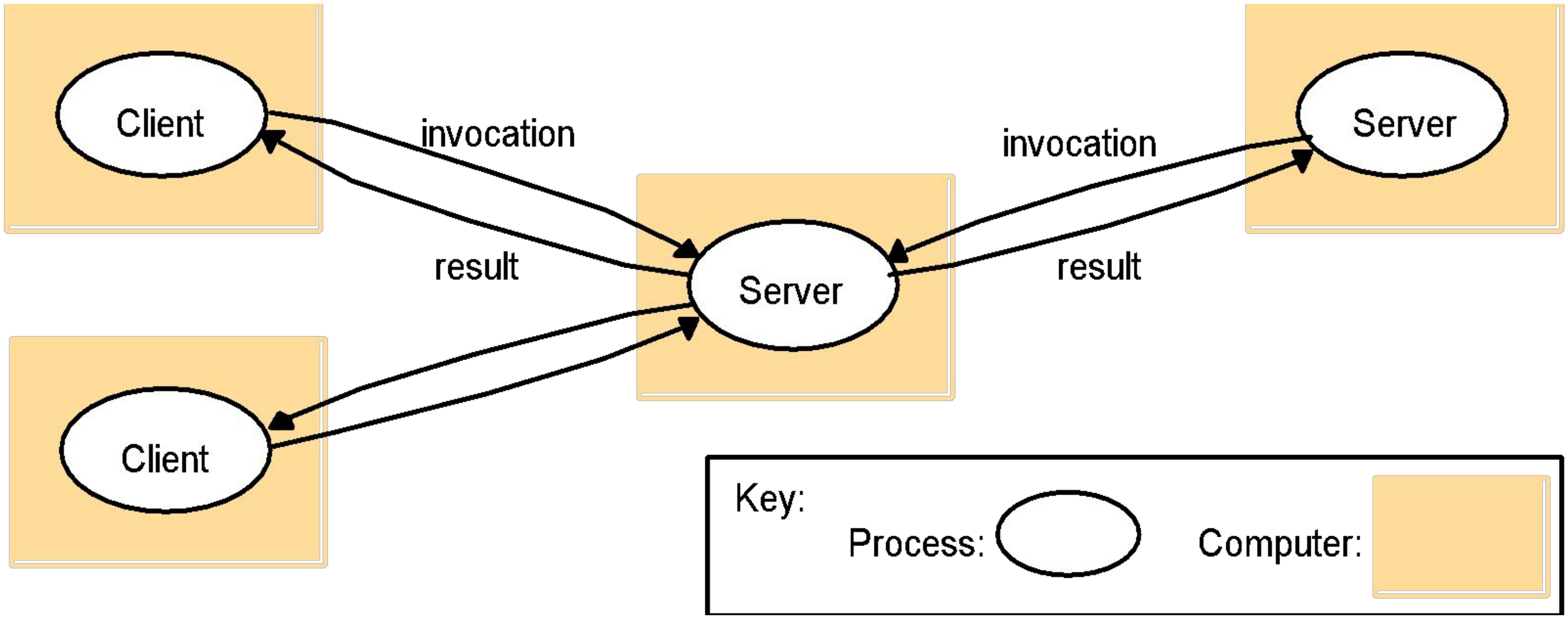
Key Questions

- What are the main entities in the system?
- How do they interact?
- How does the system operate?
- What are the characteristics that affect their individual and collective behavior?

Classifying Distributed Systems

- Based on Architectural Models
  - Client-Server, Peer-to-peer, Proxy based,…
- Based on computation/communication - degree of synchrony
  - Synchronous, Asynchronous
- Based on communication style
  - Message Passing, Shared Memory
- Based on Fault model
  - Crash failures, Omission failures, Byzantine failures
  - how to handle failure of processes/channels

Architectural Models: Client-server

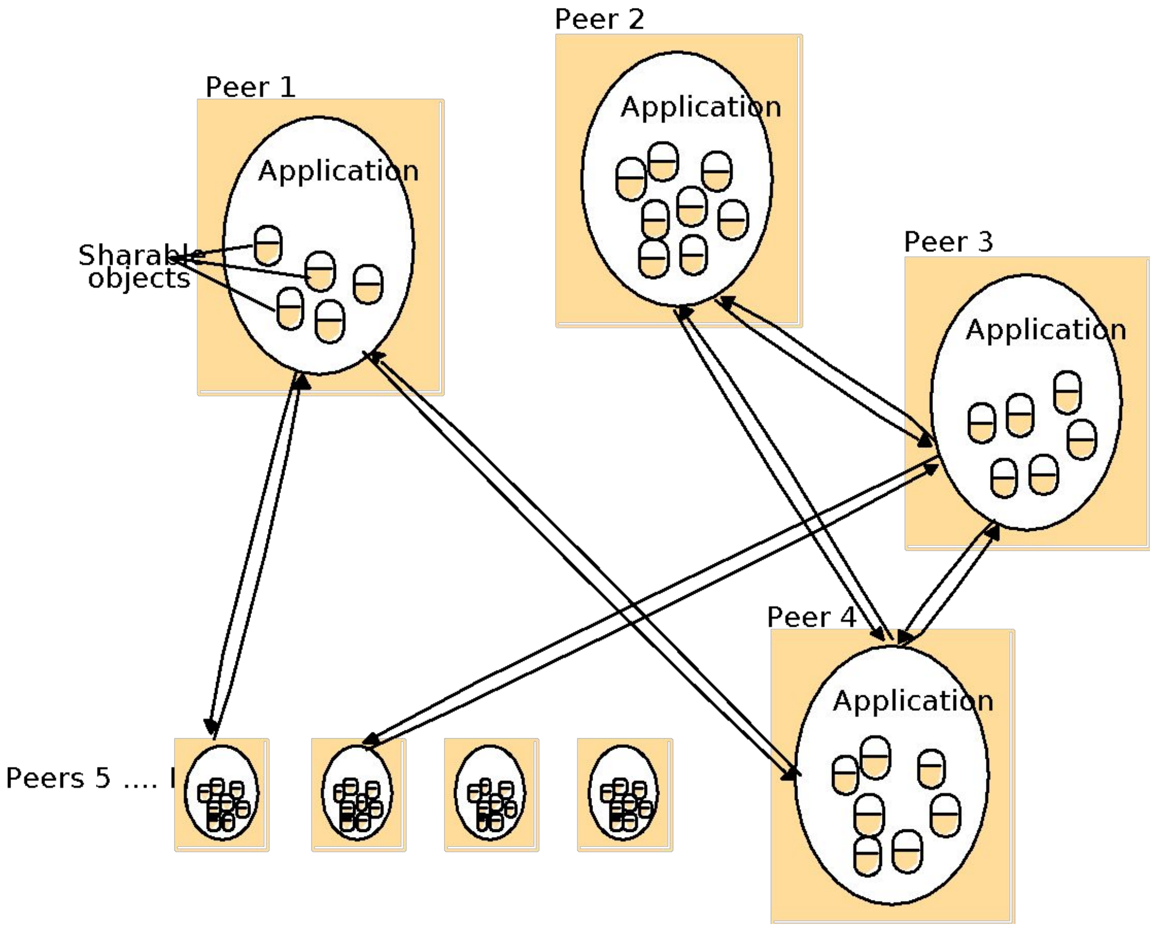


- Client/server computing allocates application processing between the client and server processes.
- Request-response paradigm
- A typical application has three basic components:
  - Presentation logic, Application logic, Data management logic

Client/Server Models

- There are at least three different models for distributing these functions:
  - Presentation logic module running on the client system and the other two modules running on one or more servers.
  - Presentation logic and application logic modules running on the client system and the data management logic module running on one or more servers.
  - Presentation logic and a part of application logic module running on the client system and the other part(s) of the application logic module and data management module running on one or more servers

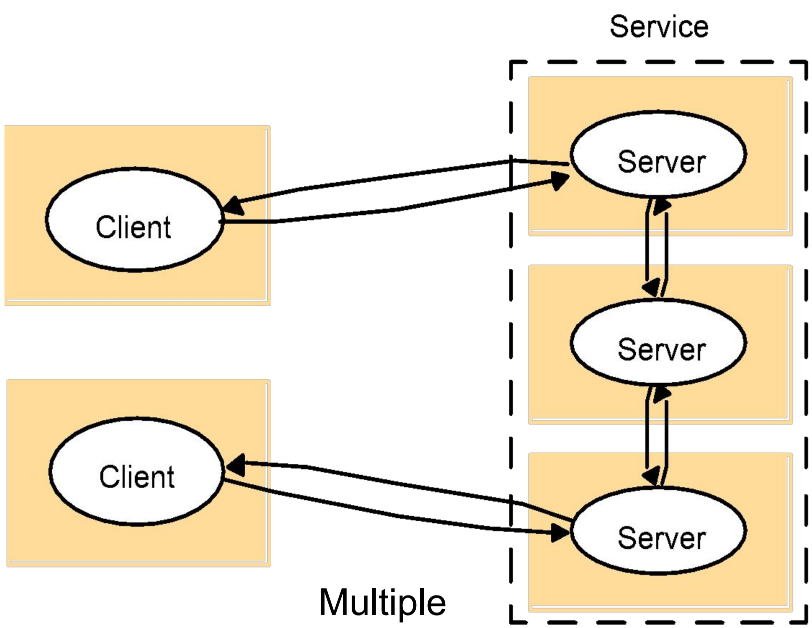
Architectural Models: Peer-to-peer



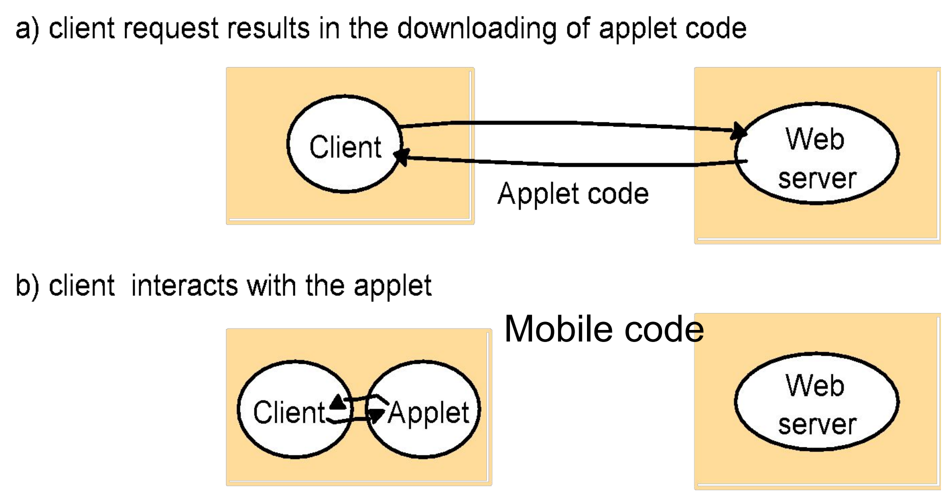
- No single node server as a server
- All nodes act as client (and server) at a time

More Architectural Models

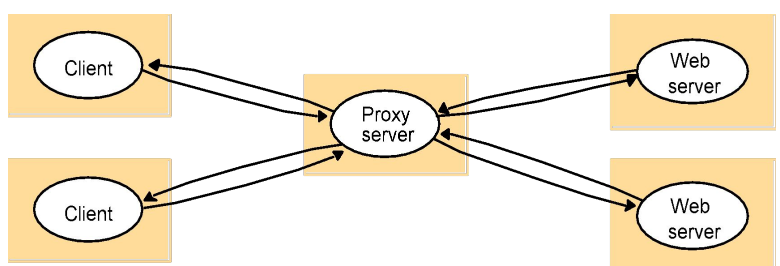
Multiple servers, proxy servers and caches, mobile code, …



Multiple servers



Mobile code



Proxy

Computation in distributed systems

Two variants based on <u>bound on timing of events</u>

- Asynchronous system
  - no assumptions about process execution speeds and message delivery delays
- Synchronous system
  - make assumptions about relative speeds of processes and delays associated with communication channels
  - constrains implementation of processes and communication
- Concurrent Programming Models
  - Communicating processes, Functions, Logical clauses, Passive Objects, Active objects, Agents

Concurrency issues

- Concurrency and correctness - general properties
  - Safety
  - Liveness
- Consider the requirements of transaction based systems
  - Atomicity - either all effects take place or none
  - Consistency - correctness of data
  - Isolated - as if there were one serial database
  - Durable - effects are not lost

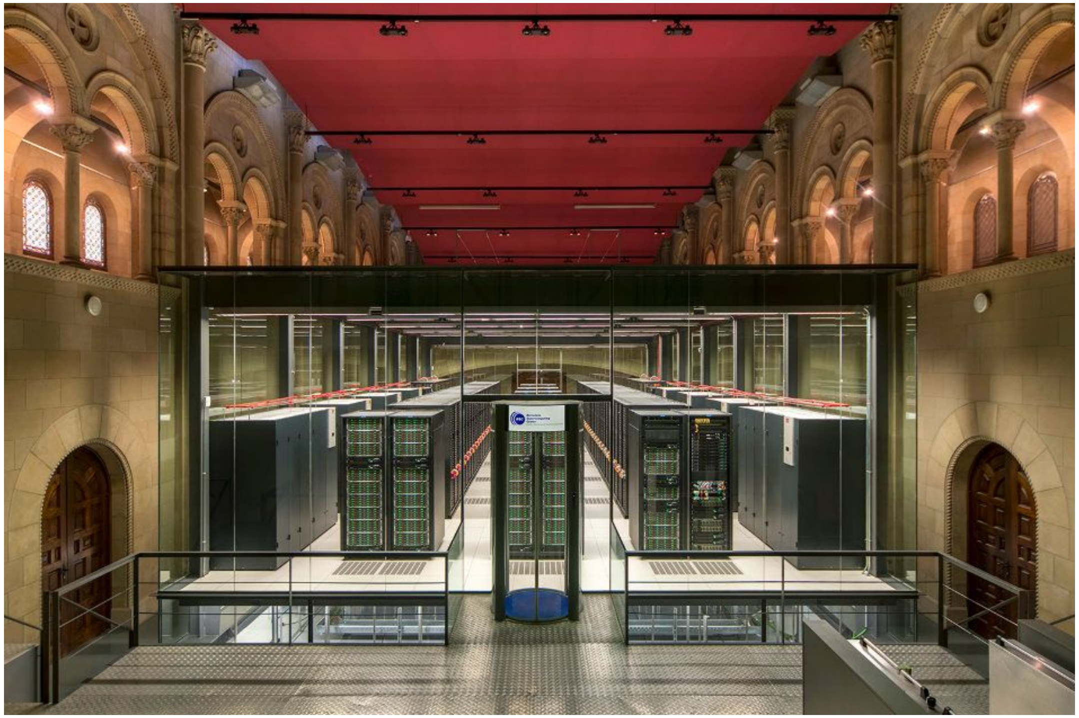
Parallel Computing Systems

- Special case of a distributed system
  - often to run a special application(s)
  - Designed to run a single program faster
- Supercomputer - high-end parallel machine

Barcelona - BSC MareNostrum 4

(165,888 cores, 24 cores/processor)

The world's most elegant supercomputer



Intel -Cray Theta @Argonne
281,888 core, 64 cores per processors
11.69 Peta-flops

Aurora: USA's First ExaSCALE computer

Imagine …

- A computer so powerful that it can predict future climate patterns, saving millions of people from drought, flood, and devastation.

- A computer so powerful that it can simulate every activity of a cancer cell, at the sub-atomic level, with such accuracy that we can effectively cure it, or create a personalized treatment, just for you.

cf: Argonne National Labs

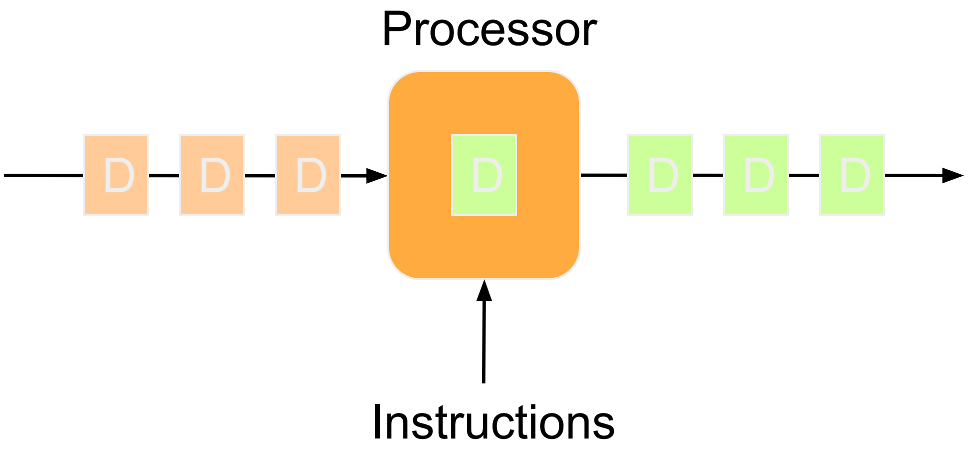
https://youtu.be/dYUEFvqQso8

Flynn's Taxonomy for Parallel Computing

Data \ Instructions	Single (SI)	Multiple (MI)
Single (SD)	**SISD** Single-threaded process	**MISD** Pipeline architecture
Multiple (MD)	**SIMD** Vector Processing	**MIMD** Multi-threaded Programming

Parallelism – A Practical Realization of Concurrency

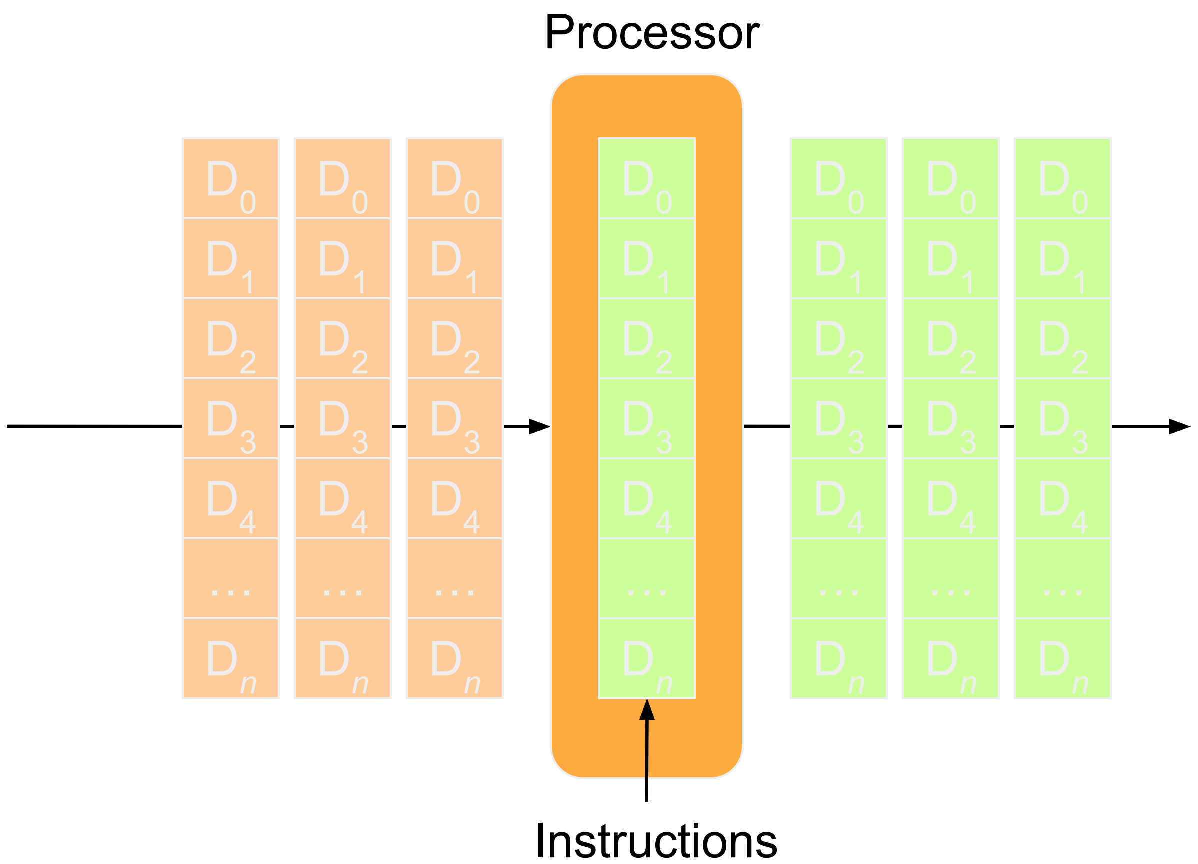
SISD (Single Instruction Single Data)



A sequential computer which exploits no parallelism in either the instruction or data streams.

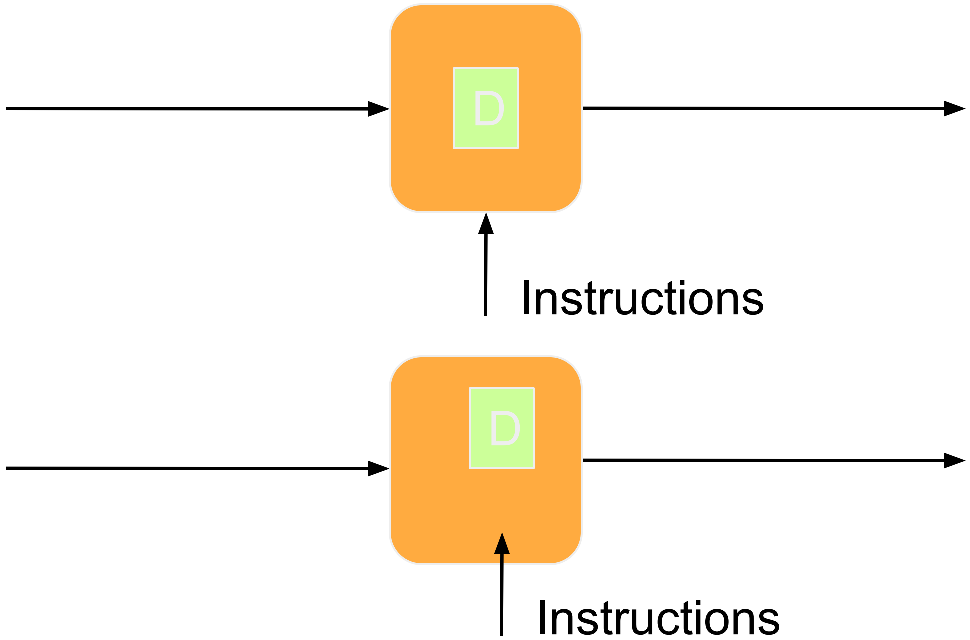
Examples of SISD architecture are the traditional uniprocessor machines (currently manufactured PCs have multiple processors) or old mainframes.

SIMD (Single Instruction Multiple Data)



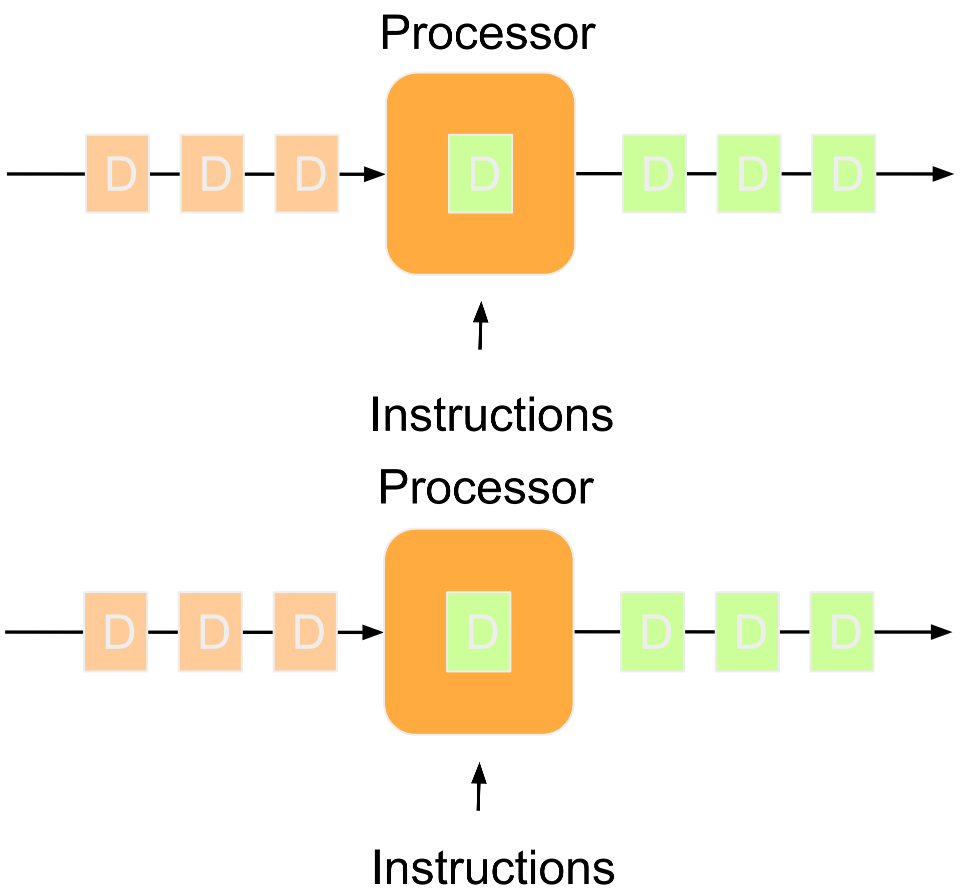
A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized.
For example, an array processorFor example, an array processor or GPU.

MISD (Multiple Instruction Single Data)



Multiple instructions operate on a single data stream.
Uncommon architecture which is generally used for fault tolerance.
Heterogeneous systems operate on the same data stream and aim to agree on the result.
Examples include the Space Shuttle flight control computer.

MIMD(Multiple Instruction Multiple Data)



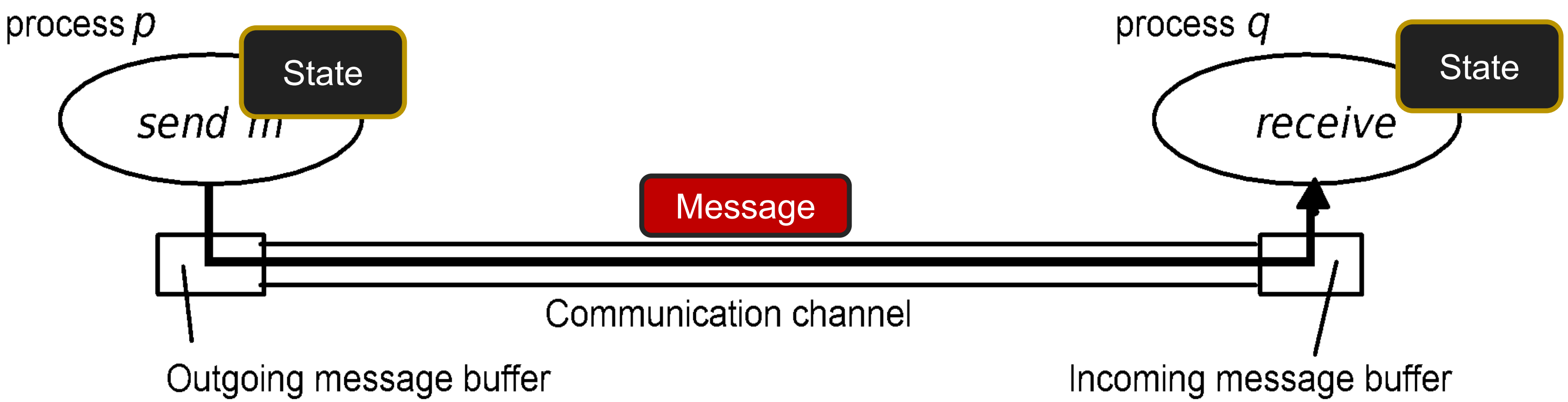
Multiple autonomous processors simultaneously executing different instructions on different data.

Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

Communication in Distributed Systems

- Provide support for entities to communicate among themselves
  - Centralized (traditional) OS's - local communication support
  - Distributed systems - communication across machine boundaries (WAN, LAN).
- 2 paradigms
  - Message Passing
    - **Processes communicate by sharing messages**
  - Distributed Shared Memory (DSM)
    - **Communication through a virtual shared memory.**

Message Passing



- Basic primitives
  - Send message, Receive message

Properties of communication channel
Latency, bandwidth and jitter

Messaging issues

Synchronous

- atomic action requiring the participation of the sender and receiver.
- Blocking send: blocks until message is transmitted out of the system send queue
- Blocking receive: blocks until message arrives in receive queue

Asynchronous

- Non-blocking send:sending process continues after message is sent
- Blocking or non-blocking receive: Blocking receive implemented by timeout or threads. Non-blocking receive proceeds while waiting for message. Message is queued(BUFFERED) upon arrival.

- Unreliable communication
  - Best effort, No ACK's or retransmissions
  - Application programmer designs own reliability mechanism
- Reliable communication
  - Different degrees of reliability
  - Processes have some guarantee that messages will be delivered.
  - Reliability mechanisms - ACKs, NACKs.

Synchronous vs. Asynchronous

Communication	**Type (sync/async)**
Personal greetings	Sync
Email	Async
Voice call	Sync
Online messenger/chat	Sync ?
Letter correspondence	Async
Skype call	Sync
Voice mail/voice SMS	Async
Text messages	Async

Remote Procedure Call

- Builds on message passing
  - extend traditional procedure call to perform transfer of control and data across network
  - Easy to use - fits well with the client/server model.
  - Helps programmer focus on the application instead of the communication protocol.
  - Server is a collection of exported procedures on some shared resource
  - Variety of RPC semantics
    - **“maybe call”**
    - **“at least once call”**
    - **“at most once call”**

Distributed Shared Memory

- Abstraction used for processes on machines that do not share memory
  - Motivated by shared memory multiprocessors that do share memory
- Processes read and write from virtual shared memory.
  - Primitives - read and write
  - OS ensures that all processes see all updates
- Caching on local node for efficiency
  - Issue - cache consistency

Fault Models in Distributed Systems

- Crash failures
  - A processor experiences a crash failure when it ceases to operate at some point without any warning. Failure may not be detectable by other processors.
    - **Failstop - processor fails by halting; detectable by other processors.**
- Byzantine failures
  - completely unconstrained failures
  - conservative, worst-case assumption for behavior of hardware and software
  - covers the possibility of intelligent (human) intrusion.

Other Fault Models in Distributed Systems

- Dealing with message loss
  - Crash + Link
    - **Processor fails by halting. Link fails by losing messages but does not delay, duplicate or corrupt messages.**
  - Receive Omission
    - **processor receives only a subset of messages sent to it.**
  - Send Omission
    - **processor fails by transmitting only a subset of the messages it actually attempts to send.**
  - General Omission
    - **Receive and/or send omission**

Failure Models

Omission and arbitrary failures

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Channel	A process completes a *send,* but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Failure Models

Timing failures

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Other distributed system issues

- Concurrency and Synchronization
- Distributed Deadlocks
- Time in distributed systems
- Naming
- Replication
  - improve availability and performance
- Migration
  - of processes and data
- Security
  - eavesdropping, masquerading, message tampering, replaying

EXTRA MATERIAL

Middleware for distributed systems

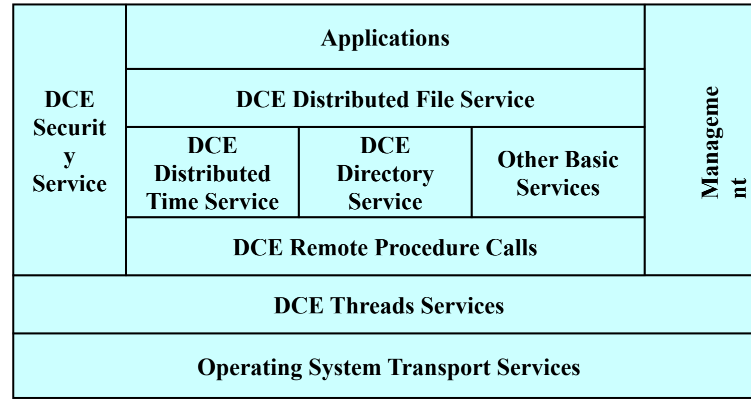
- Middleware is the software between the application programs and the Operating System/base networking.
  - An Integration Fabric that knits together applications, devices, systems software, data
- Distributed Middleware
  - Provides a comprehensive set of higher-level distributed computing capabilities and a set of interfaces to access the capabilities of the system.
  - Provides Higher-level programming abstraction for developing distributed applications
    - Higher than "lower" level abstractions, such as sockets, monitors provided by the OS operating system
  - Includes software technologies to help manage complexity and heterogeneity inherent to the development of distributed systems/applications/information systems. Enables modular interconnection of distributed "services".

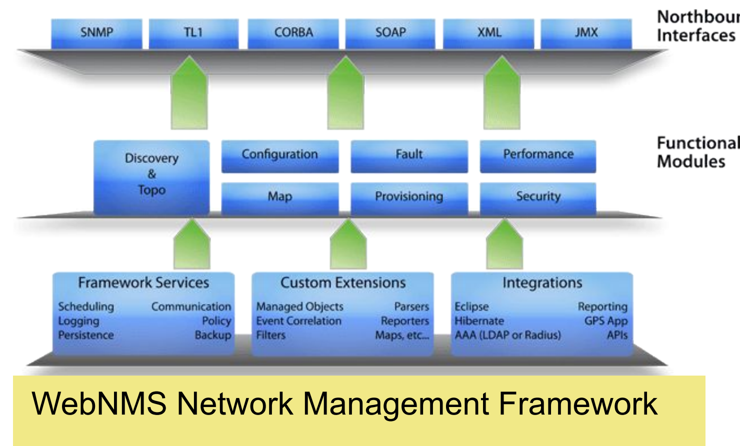
Useful Management Services: Naming and Directory Service, State Capture Service. Event Service, Transaction Service, Fault Detection Service, Discovery/trading Service, Replication Service, Migration Services

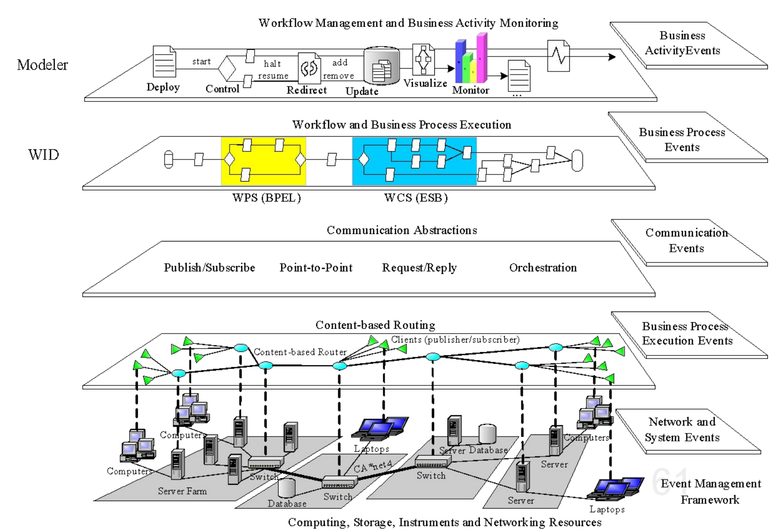
cf: Arno Jacobsen lectures, Univ. of Toronto

Types of Middleware

- Integrated Sets of Services
  - DCE from OSF - provides key distributed technologies, including RPC, a distributed naming service, time synchronization service, a distributed file system, a network security service, and a threads package.
- Domain Specific Integration frameworks
  - Transaction processing, workflows, network management
- Distributed Object Frameworks
- Component services and frameworks
- Web-Service Based Frameworks
- Enterprise Service Buses
- Cloud Based Frameworks

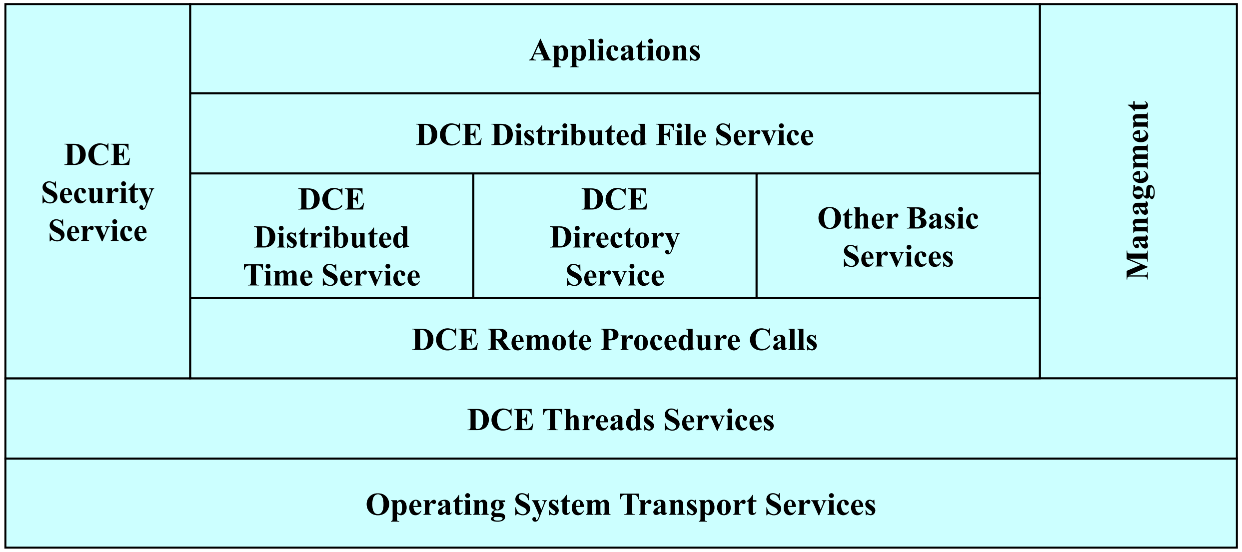






Distributed Computing Environment (DCE)

- DCE - from the Open Software Foundation (OSF), offers an environment that spans multiple architectures, protocols, and operating systems (supported by major software vendors)
  - It provides key distributed technologies, including RPC, a distributed naming service, time synchronization service, a distributed file system, a network security service, and a threads package.



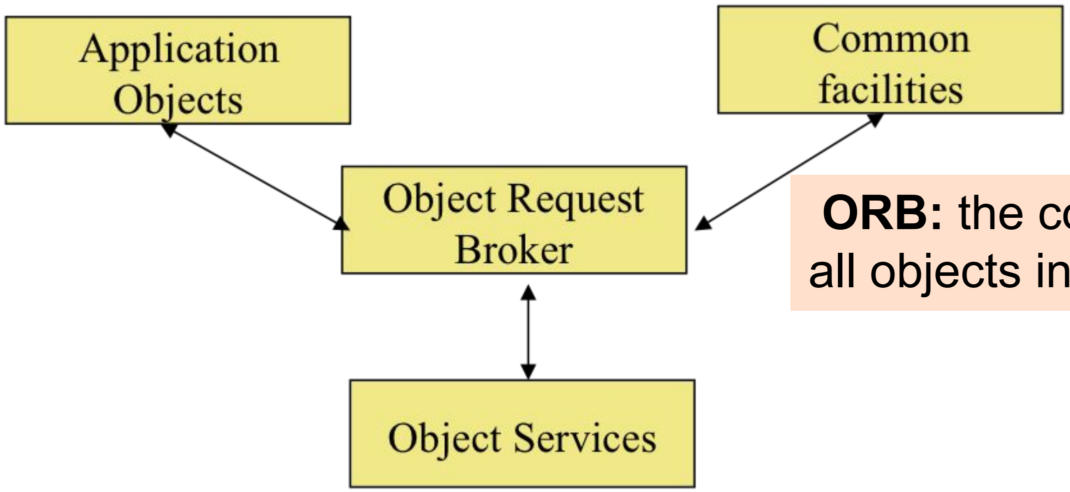
Distributed Object Models

- Goal: Merge distributed computing/parallelism with an object model
  - **Object Oriented Programming**
    - Encapsulation, modularity, abstraction
    - Separation of concerns
  - **Concurrency/Parallelism**
    - Increased efficiency of algorithms
    - Use objects as the basis (lends itself well to natural design of algorithms)
  - **Distribution**
    - Build network-enabled applications
    - Objects on different machines/platforms communicate
      - The use of a broker like entity or bus that keeps track of processes, provides messaging between processes and other higher level services
  - CORBA, COM, DCOM, JINI, EJB, J2EE, Agent and actor-based models

The Object Management Architecture (OMA)

Application objects: document handling objects.

Common facilities: accessing databases, printing files, etc.



ORB: the communication hub for all objects in the system

Object Services: object events, persistent objects, etc.

Objects and Threads

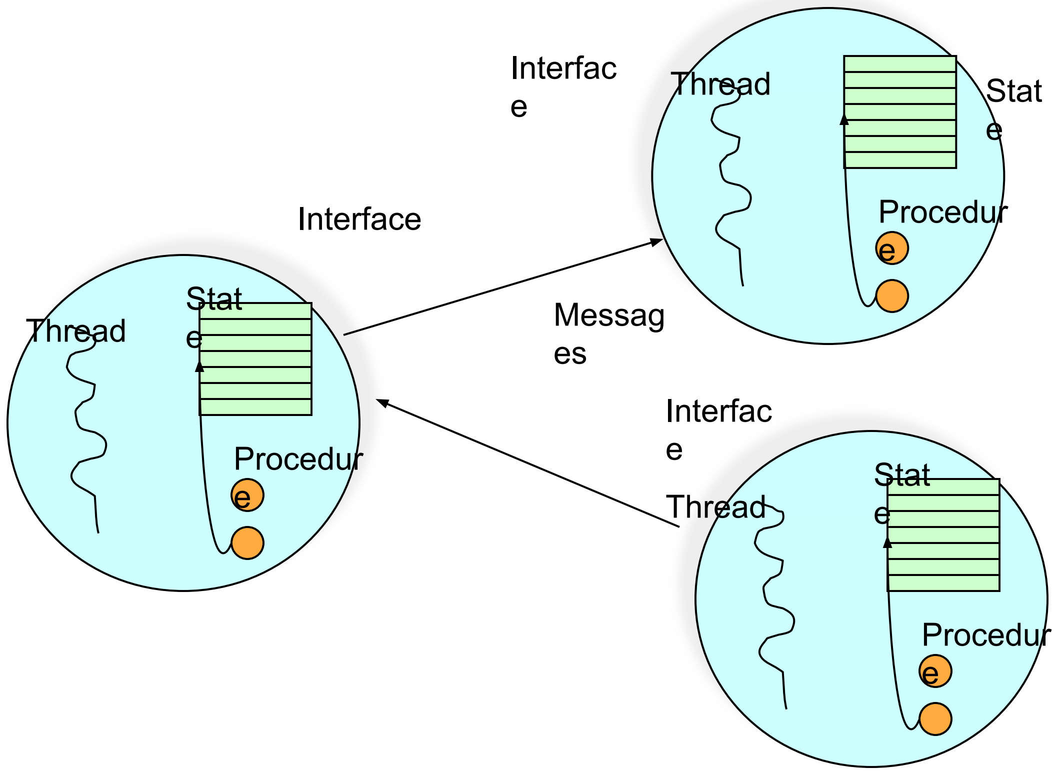
- C++ Model
  - Objects and threads are tangentially related
  - Non-threaded program has one main thread of control
    - **Pthreads (POSIX threads)**
      - **Invoke by giving a function pointer to any function in the system**
      - **Threads mostly lack awareness of OOP ideas and environment**
      - **Partially due to the hybrid nature of C++?**

- Java Model
  - Objects and threads are separate entities
    - **Threads are objects in themselves**
    - **Can be joined together (complex object implements java.lang.Runnable)**
      - **BUT: Properties of connection between object and thread are not well-defined or understood**

Java and Concurrency

- Java has a passive object model
  - Objects, threads separate entities
    - **Primitive control over interactions**
  - Synchronization capabilities also primitive
    - **"Synchronized keyword" guarantees safety but not liveness**
    - **Deadlock is easy to create**
    - **Fair scheduling is not an option**

Actors:
A Model of Distributed Objects



Actor system - collection of independent agents interacting via message passing

Features

- Acquaintances
  - initial, created, acquired
- History Sensitive
- Asynchronous communication

An actor can do one of three things:

1. Create a new actor and initialize its behavior
2. Send a message to an existing actor
3. Change its local state or behavior

Distributed Objects

- Techniques
  - Message Passing
    - Object knows about network;
    - Network data is minimum
  - Argument/Return Passing
    - Like RPC.
    - Network data = args + return result + names
  - Serializing and Sending Object
    - Actual object code is sent. Might require synchronization.
    - Network data = object code + object state + sync info
  - Shared Memory
    - based on DSM implementation
    - Network Data = Data touched + synchronization info

- Issues with Distributed Objects
  - Abstraction
  - Performance
  - Latency
  - Partial failure
  - Synchronization
  - Complexity
  - .....

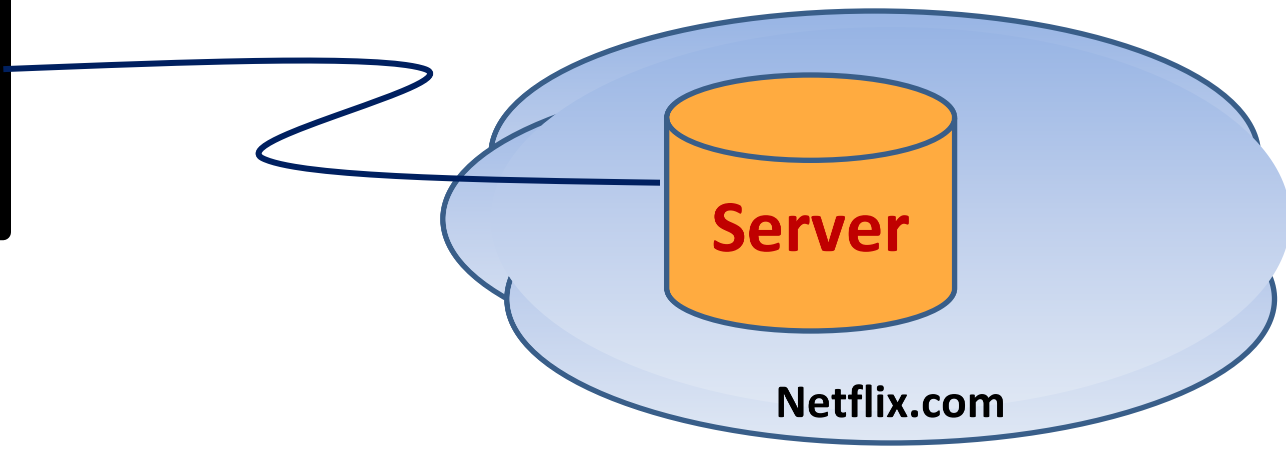
Cloud Computing

- Cloud - Large multi-tenant data centers hosting storage, computing, analytics, applications as services.
  - Amazon, Salesforce, Google, Microsoft

- An example: Netflix
  - Offers Online streaming video service (17,000+ titles in 2010)
  - Netflix website with support for video search
  - Recommendation engines
  - Instant playback on 100s of devices including xbox, game consoles, roku, mobile devices, etc.
  - Transcoding service
  - …

Netflix App: version 0 (how it started)

- Plays movies on demand on a mobile device



Simple Design

- Web Services standards
- Netflix owns the data center
- Uses a fairly standard server

Challenges with Version 0

- Incredible growth in customers and devices led to
  - Need for horizontal scaling of every layer of software stack.
  - Needed to support high availability, low latency, synchronization, fault-tolerance, …
- Had a decision to make:
  - Build their own data centers to do all the above OR
  - Write a check to someone else to do all that instead

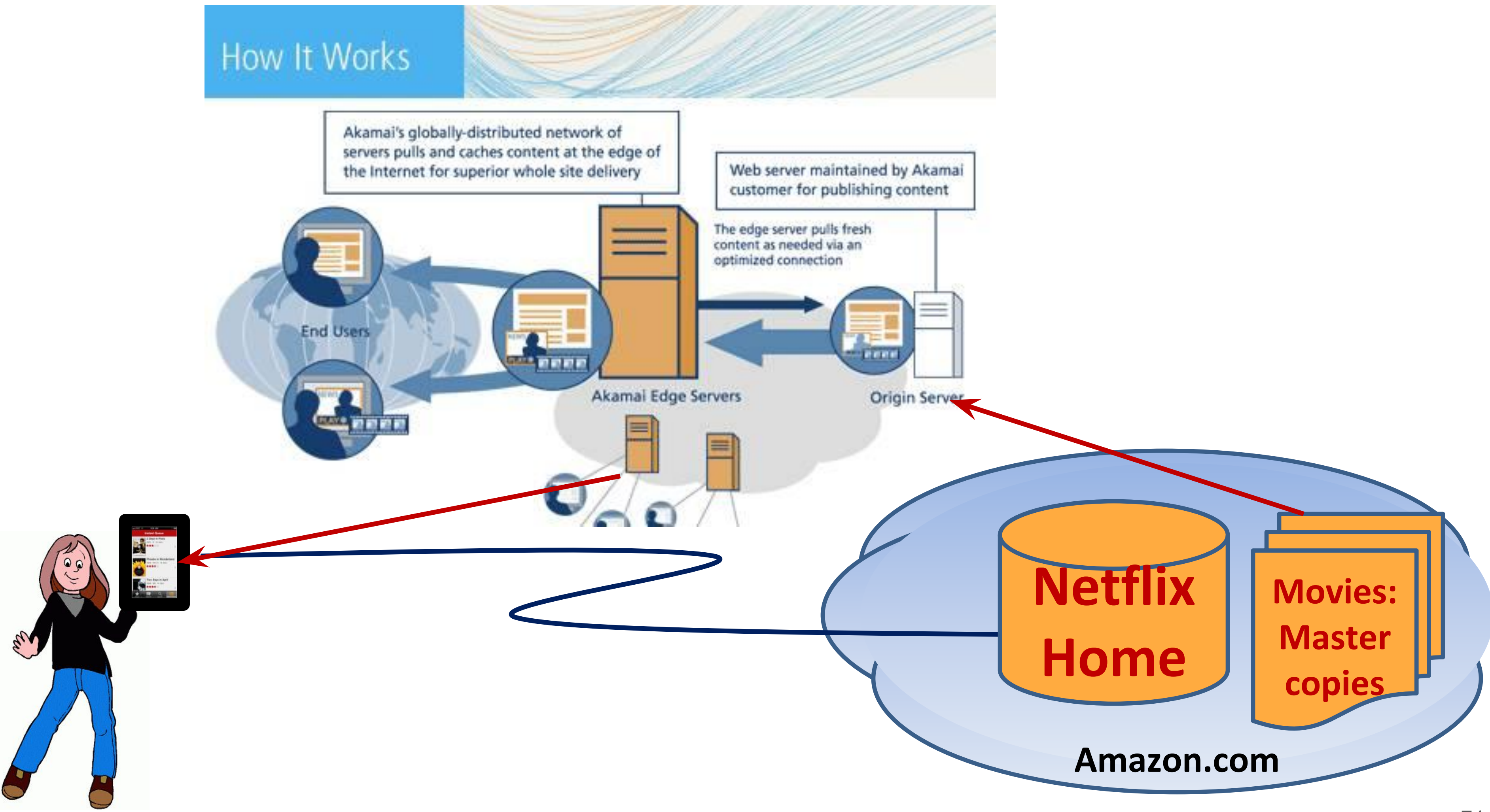
Netflix migrated to Amazon AWS

- John Ciancutti, VP engg. Netflix 2010 [Technical Blog]
  - Letting Amazon focus on data center infrastructure allows our engineers to focus on building and improving our business.
    - **Amazon calls their web services "undifferentiated heavy lifting" and that's what it is. The problems they are trying to solve are incredibly difficult ones, but they aren't specific to our business. Every successful company has to figure out great storage, hardware failover, network infrastructure, etc.**
  - We're not very good at predicting customer growth or device engagement.
    - **Netflix has revised our public guidance for the number of customers we will end 2010 with three times over the course of the year. We are operating in a fast-changing and emerging market. How may subscribers would you guess used our Wii application the week it was launched? How many would you guess will use it next month? We have to ask ourselves these questions for each device we launch because our software systems need to scale to the size of the business, every time.**
    - **Cloud environments are ideal for horizontally scaling architecture. We don't have to guess months ahead what our hardware, storage, and networking needs are going to be. We can programmatically access more off these resources from shared pools within AWS almost instantly.**

Netflix “outsourcing” components

- Think of Netflix in terms of main components
  - The API you see that runs on your client system
  - The routing policy used to connect you to a data center
  - The Netflix “home page” service in that data center
  - The movie you end up downloading
- Netflix cloud-based design
  - breaks the solution into parts
  - Builds each of these aspects itself
  - But then pays a hosting company to run each part, and not necessarily just one company!

Netflix Version 1



Features of new version

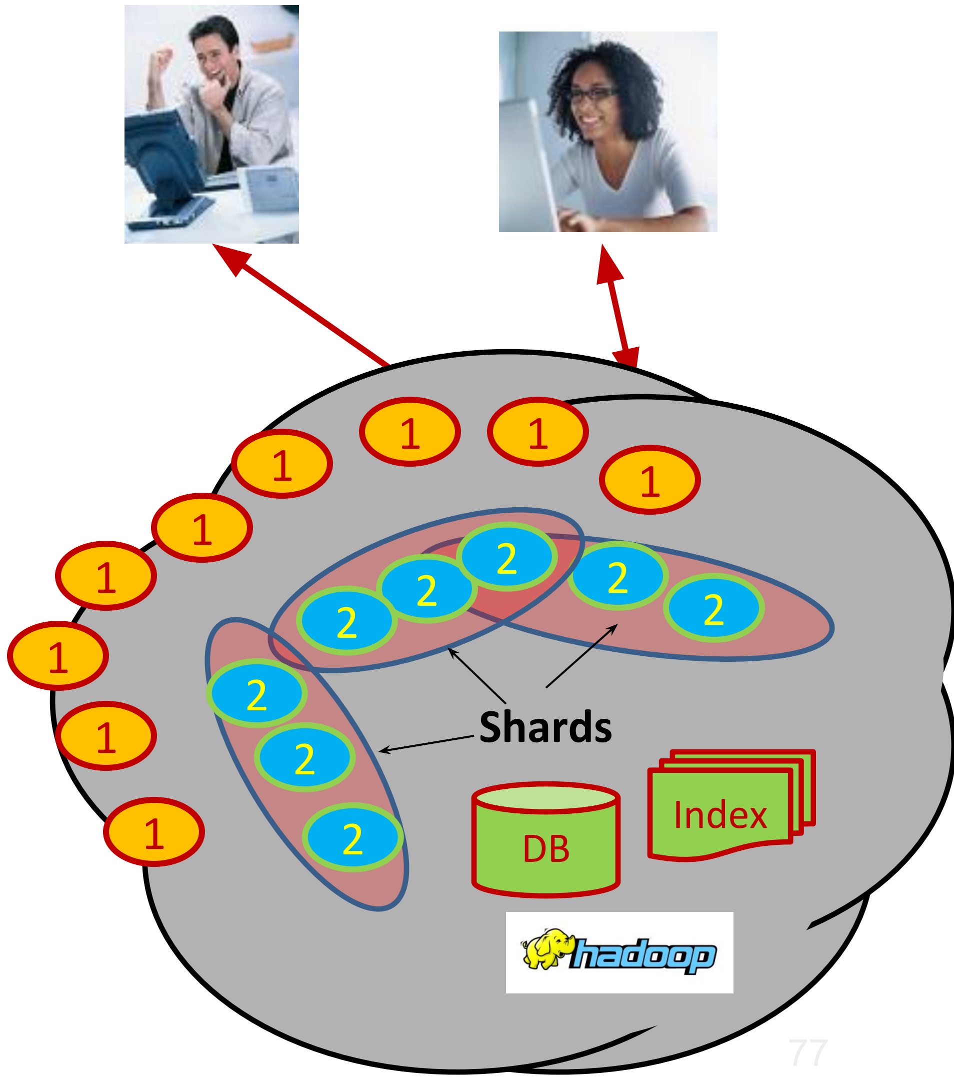
- Netflix.com is actually a "pseudonym" for Amazon.com
  - An IP address domain within Amazon.com
  - Amazon's control over the DNS allows it to vector your request to a nearby Amazon.com data center, then on arrival, Amazon gateway routes request to a Netflix cloud service component
  - The number of these varies elastically based on load Netflix is experiencing
- Amazon AWS used to host the master copies of Netflix movies

Akamai

- Akamai is an example of a “content distribution service”
  - A company that plays an intermediary role
  - Content is delivered to the service by Netflix.com (from its Amazon.com platform)
  - Akamai makes copies “as needed” and distributes them to end users who present Akamai with appropriate URLs
- Netflix.com (within Amazon.com) returns a web page with “redirection” URLs to tell your browser app what to fetch from Akamai

Multi-tier View of Cloud Computing

- Good to view cloud applications running in a data center in a tiered way
- Outer tier near the edge of the cloud hosts applications & web-sites
  - Clients typically use web browsers or web services interface to talk to the outer tier
  - focus is on vast numbers of clients & rapid response.
- Inside the cloud (next tier) we find high volume services that operate in a pipelined manner, asynchronously
  - Caching to support nimble outer tier services
- Deep inside the cloud is a world of virtual computer clusters that are scheduled to share resources and on which applications like MapReduce (Hadoop) are very popular



In the outer tiers replication is key

- We need to replicate
  - Processing: each client has what seems to be a private, dedicated server (for a little while)
  - Data: as much as possible, that server has copies of the data it needs to respond to client requests without any delay at all
  - Control information: the entire structure is managed in an agreed-upon way by a decentralized cloud management infrastructure

But, In a more general setting - with updates and faults, consistency becomes hard to maintain across the replicas (more later)

Tradeoffs in Distributed Systems

Some interesting experiences

HOPELESSNESS
AND CONFIDENCE
IN DISTRIBUTED
SYSTEMS DESIGN

https://youtu.be/TIU1opuCXB0

Tradeoffs: The CAP Conjecture (Eric Brewer: PODC 2000 Keynote)

It is impossible for a networked shared-data system to provide following *three guarantees at the same time*:

- **Consistency**
- **Availability**
- **Partition-tolerance**

Proved in 2002 by Gilbert and Lynch (CAP Theorem)

Will revisit later…