Introduction to Networks and Distributed Computing

CECS 327

Hailu Xu

1. Introduction

Textbook: Distributed Systems, concepts and design, Fifth Edition

Thanks the slides from Andrew Tanenbaum and Marten van Steen, *Distributed Systems – Principles and Paradigms,* Ying Lu, UNL, *CSCE990 Advanced Distributed Systems Seminar*
http://cse.unl.edu/~ylu/csce990/notes/Introduction.ppt

The Rise of Distributed Systems

- Computer hardware prices falling, power increasing
- Network connectivity increasing
  - Everyone is connected with networks, even when moving
- It is *easy* to connect hardware together
  - Layered abstractions have worked very well

- Definition: a *distributed system* is

  "A collection of <u>independent computers</u> that appears to its users as a <u>single coherent system</u>"

<u>Enslow's Definition</u>

Distributed System = Distributed hardware + Distributed control + Distributed data

Why Distributed Computing?

A. Big data continues to grow.

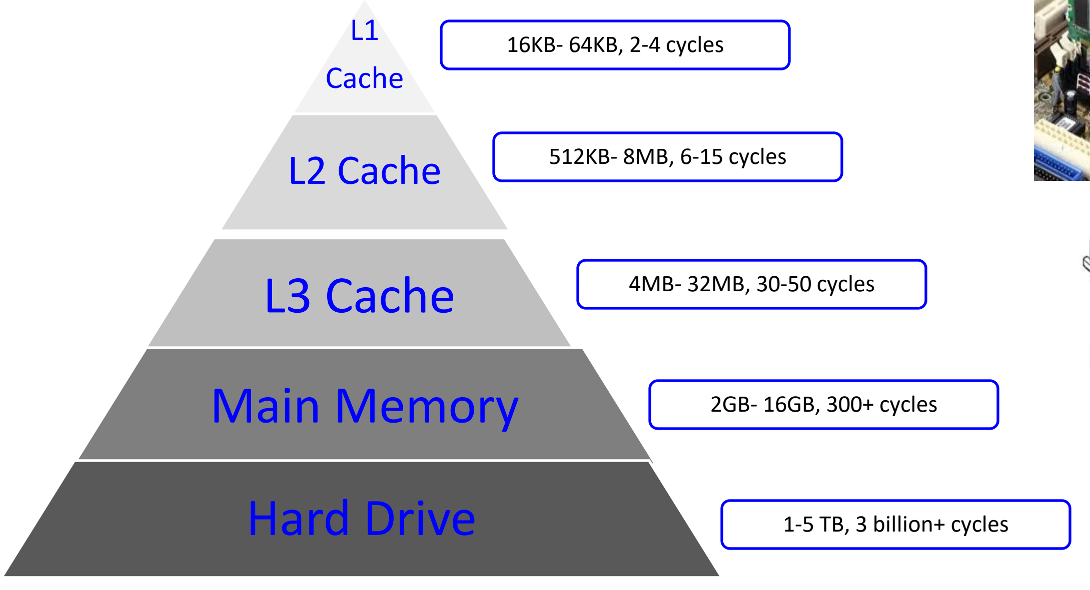
B. Applications are becoming *data-intensive*.

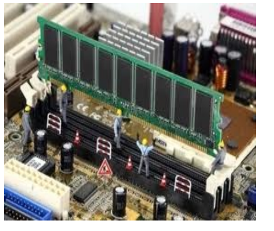
- Big data - large pools of data captured, communicated, aggregated, stored, and analyzed
- Google processes 20 petabytes of data per day
- E.g., data-intensive app: astronomical data parsing

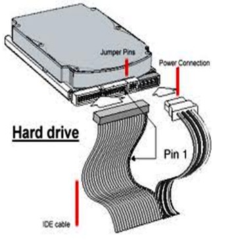
Why Distributed Systems?

C. Individual computers have limited resources compared to scale of current problems & application domains:

1. Caches and Memory:

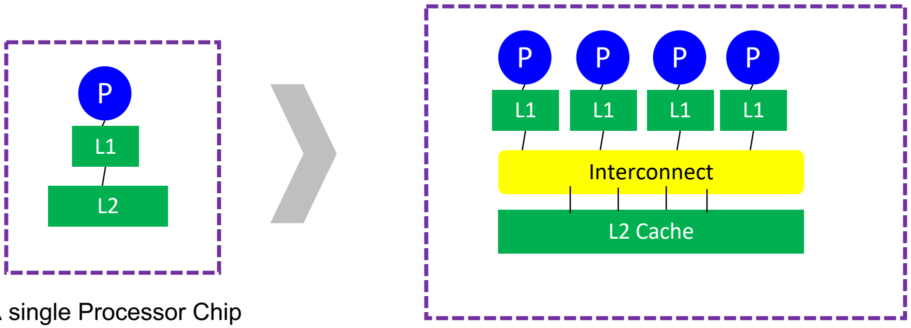






Why Distributed Systems?

2. Processor:
   - Number of transistors integrated on single die has continued to grow at Moore's pace
   - Chip Multiprocessors (*CMPs*) are now available

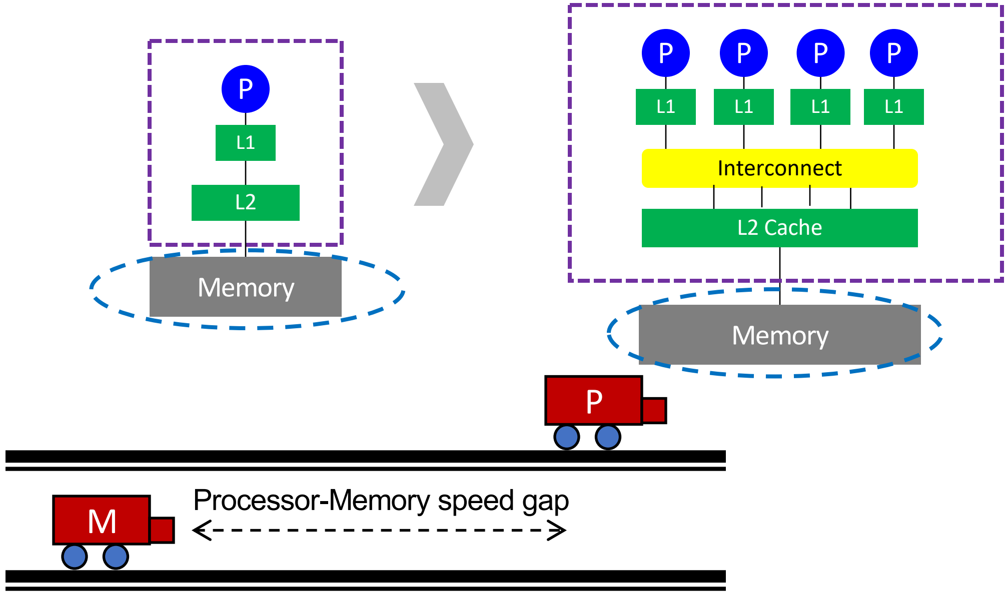


A single Processor Chip

A CMP

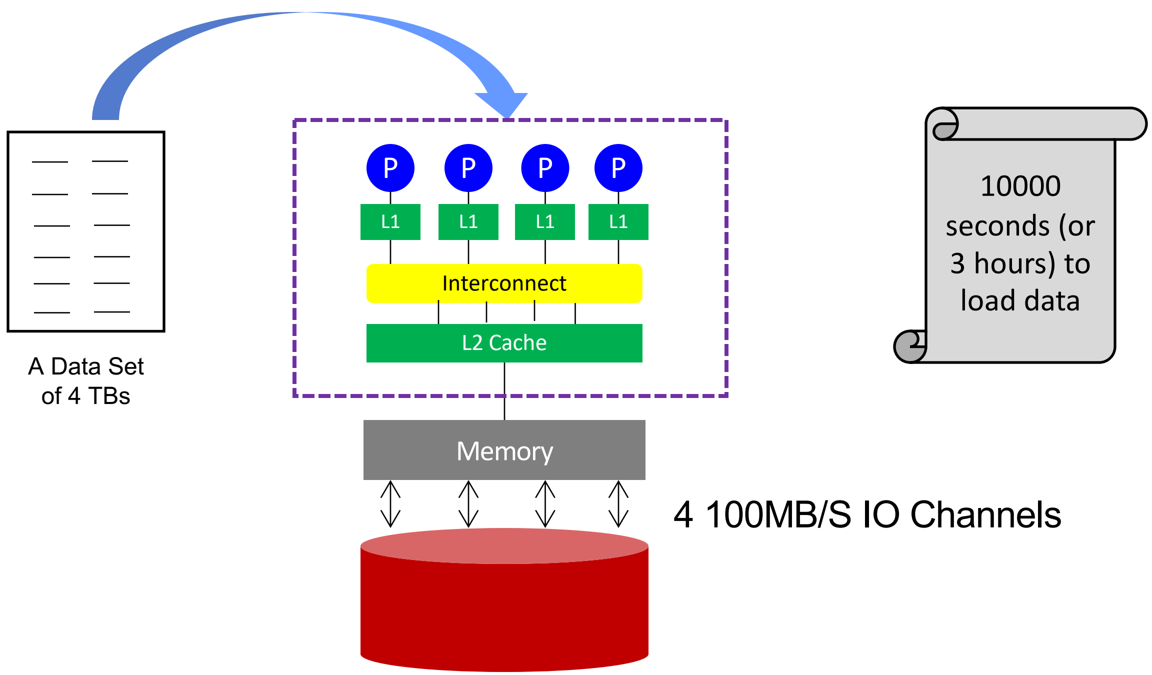
Why Distributed Systems?

3. Processor (continued):
   - CPU speed grows at rate of 55% annually, but mem speed grew only 7%

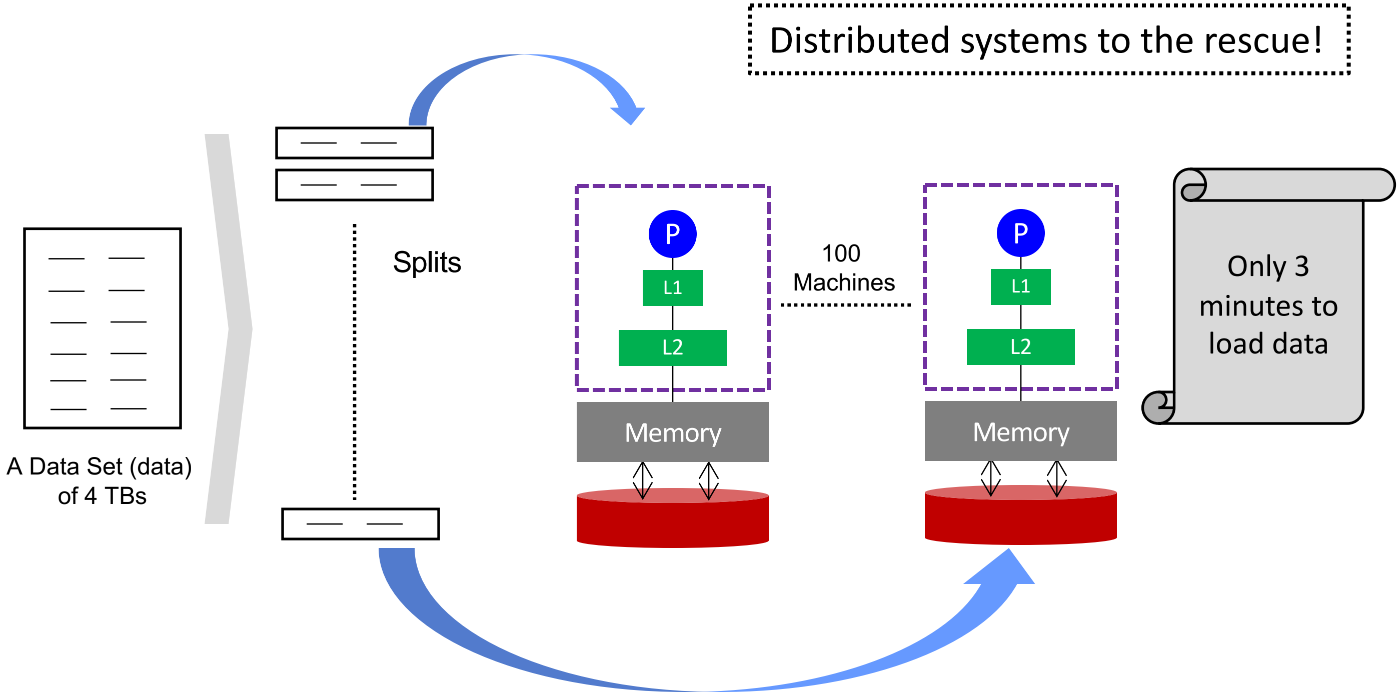


Why Distributed Systems?

- Even if 100s or 1000s of cores are placed on CMP, challenge to deliver stored data to cores fast enough for processing



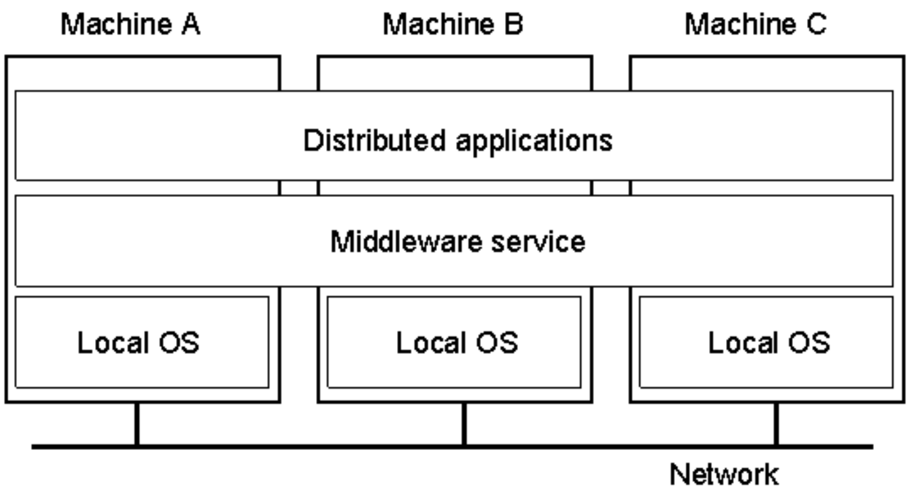
Why Distributed Systems?



But this brings new requirements

- A way to express problem as parallel processes and execute them on different machines (Programming Models and Concurrency).
- A way for processes on different machines to exchange information (Communication).
- A way for processes to cooperate with one another and agree on shared values (Synchronization).
- A way to enhance reliability and improve performance (Consistency and Replication).
- A way to recover from partial failures (Fault Tolerance).
- A way to protect communication and ensure that process gets only those access rights it is entitled to (Security).
- A way to extend interfaces so as to mimic behavior of another system, reduce diversity of platforms, and provide high degree of portability and flexibility (Virtualization)

Depiction of a Distributed System



Examples:
- The Web
- Processor pool
- Shared memory pool
- Airline reservation
- Network game
- The Cloud

- Distributed system organized as middleware. Note middleware layer extends over multiple machines.
- Users can interact with system in consistent way, regardless of where interaction takes place (e.g., RPC, `memcached`, …
- Note: Middleware may be "part" of application in practice

Introduction

- Overview (done)
- Goals (next)
- Software
- Architecture
- Examples

Goal - Transparency

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be copied
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Goal - Scalability

- As systems grow, centralized solutions are limited
  - Consider LAN name resolution (ARP) vs. WAN

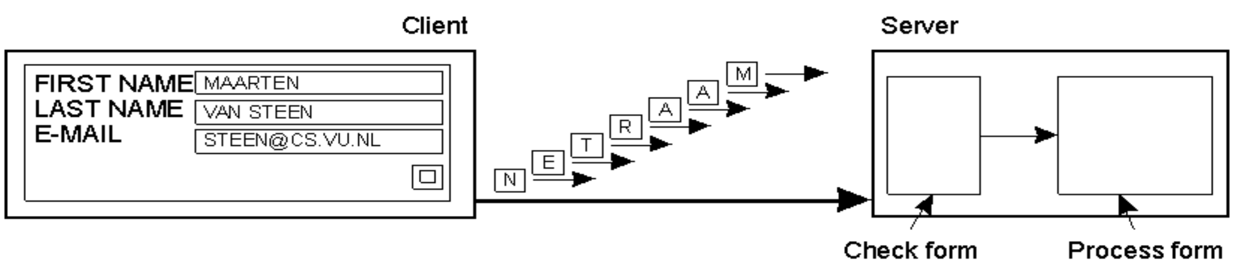
Concept	**Example**
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

- Ideally, collect information in distributed fashion and distribute in distributed fashion
- But sometimes, hard to avoid (e.g., consider money in bank)
- Challenges: geography, ownership domains, time synchronization
- Scaling techniques? → Hiding latency, distribution, replication (next)

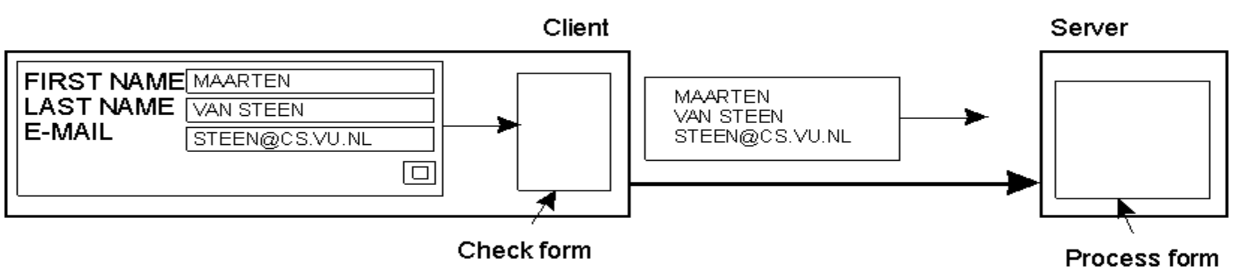
Scaling Technique: Hiding Communication Latency

- Especially important for interactive applications
- If possible, do *asynchronous communication* – continue working so user does not notice delay

  - Not always possible when client has nothing to do
- Instead, can hide latencies



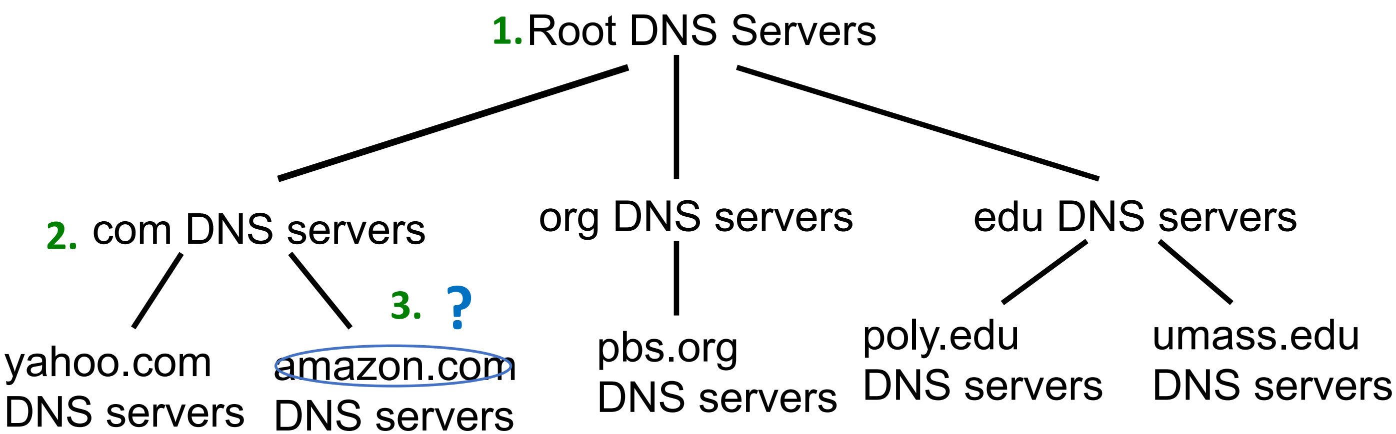
(a)



(b)

Scaling Technique: Distribution

- Spread information/processing to more than one location



Client wants IP for www.amazon.com (*approximation*):

1. Client queries root server to find `.com` DNS server
2. Client queries `.com` DNS server to get `amazon.com` DNS server
3. Client queries `amazon.com` DNS server to get IP address for `www.amazon.com`

Scaling Technique: Replication

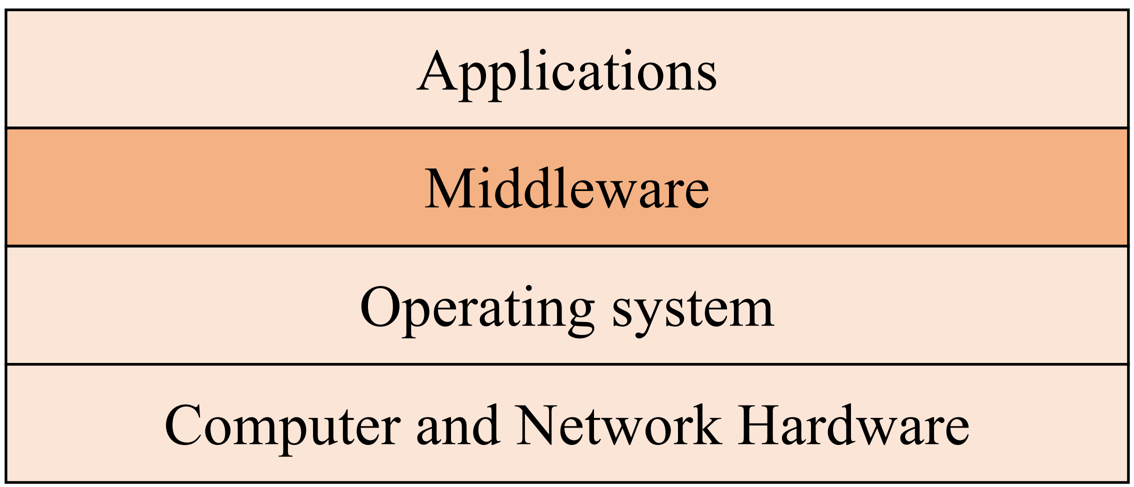
- Copy of information to increase availability and decrease centralized load
  - Example: File caching is replication decision made by client
  - Example: CDNs (e.g., *Akamai*) for Web
  - Example: P2P networks (e.g., *BitTorrent*) distribute copies uniformly or in proportion to use
- Issue: Consistency of replicated information
  - Example: Web browser cache– how to tell it is out of date?

Introduction

- Overview (done)
- Goals (done)
- Software (next)
- Architecture
- Examples

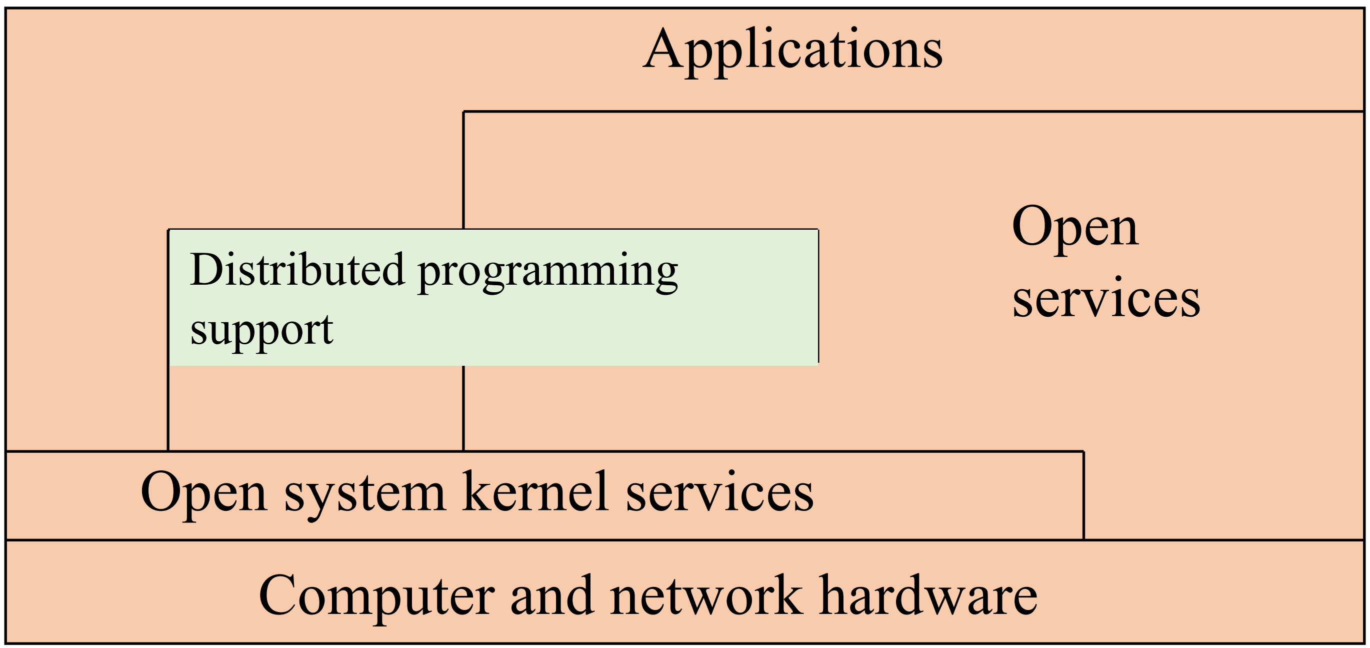
Software Structure

- Layers in centralized computer systems:



Software Structure

- Layers and dependencies in distributed systems:



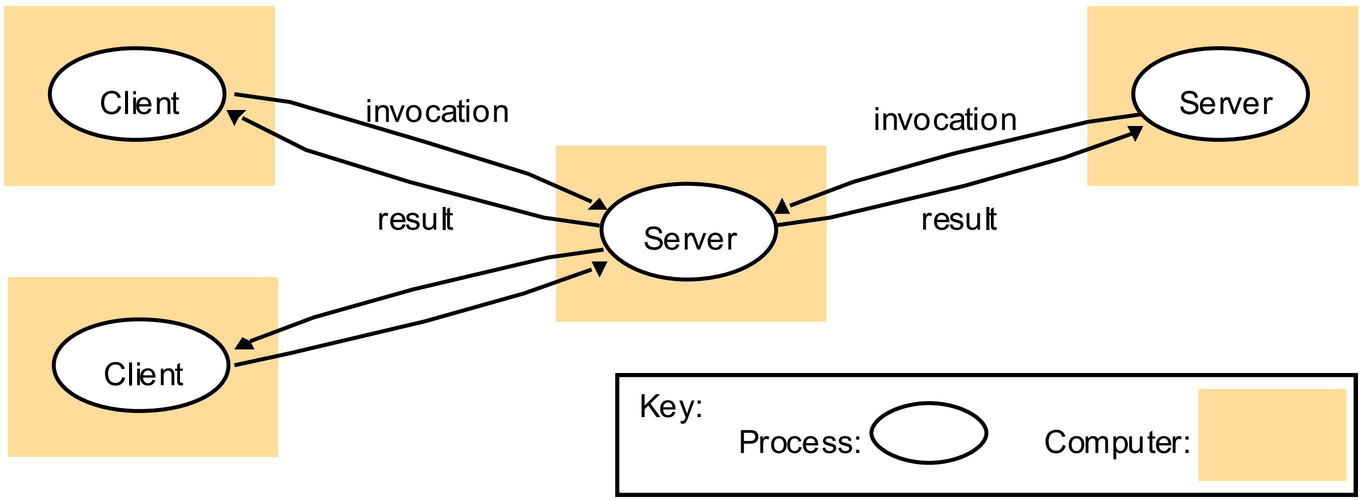
Introduction

- Overview (done)
- Goals (done)
- Software (done)
- Architecture (next)
- Examples

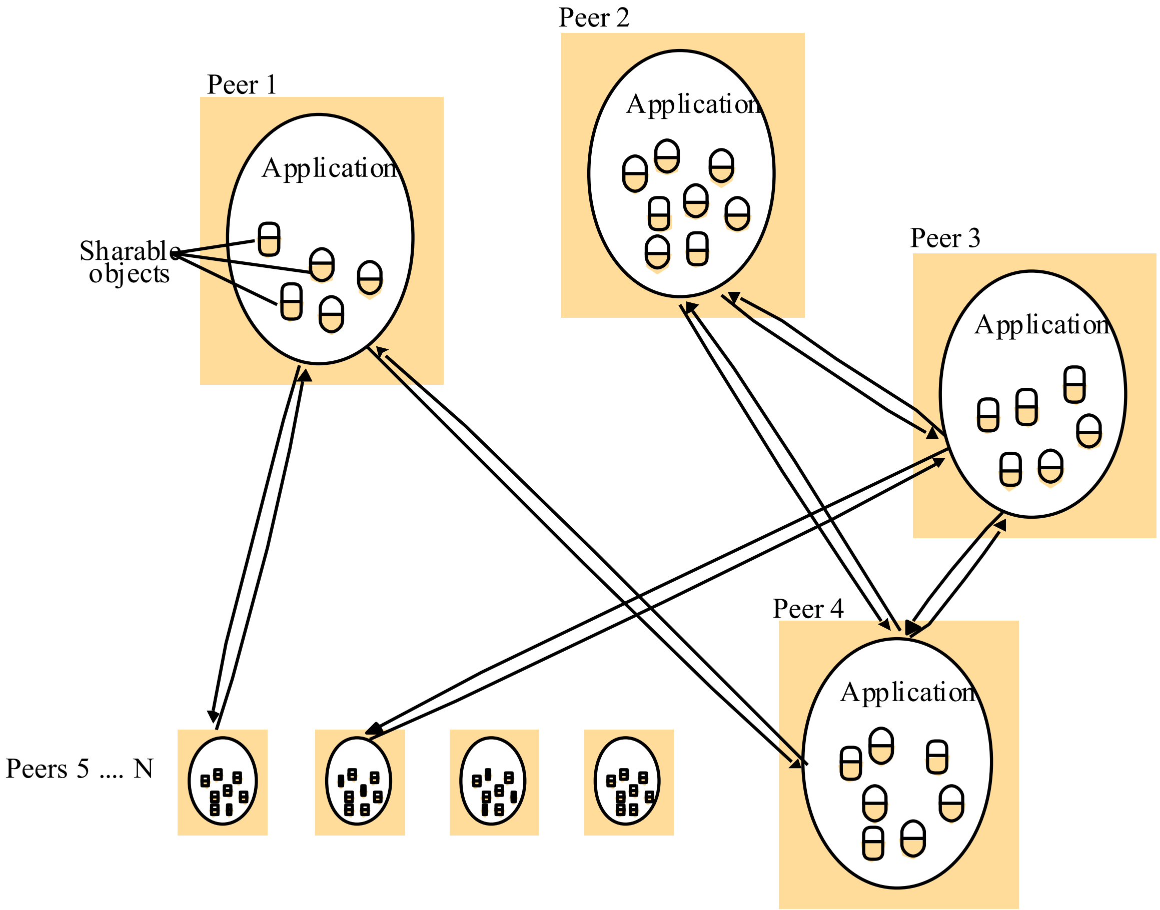
System Architectures

- Client-Server
- Peer-to-Peer
- Services provided by multiple servers
- Proxy servers and caches
- Mobile code and mobile agents
- Network computers
- Thin clients and mobile devices

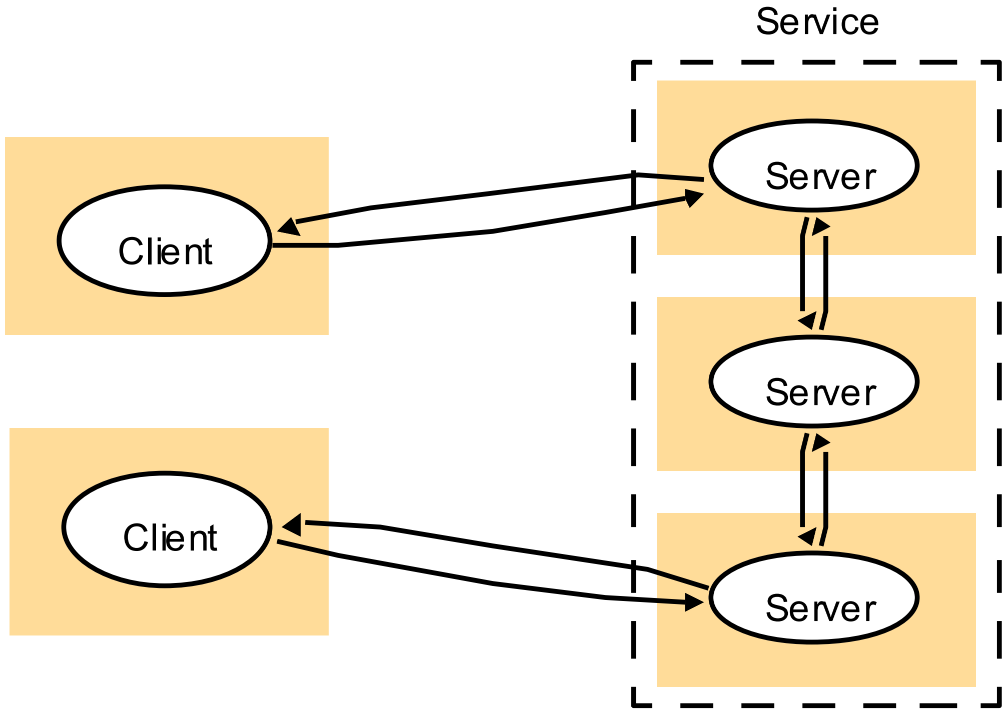
1. Clients Invoke Individual Servers



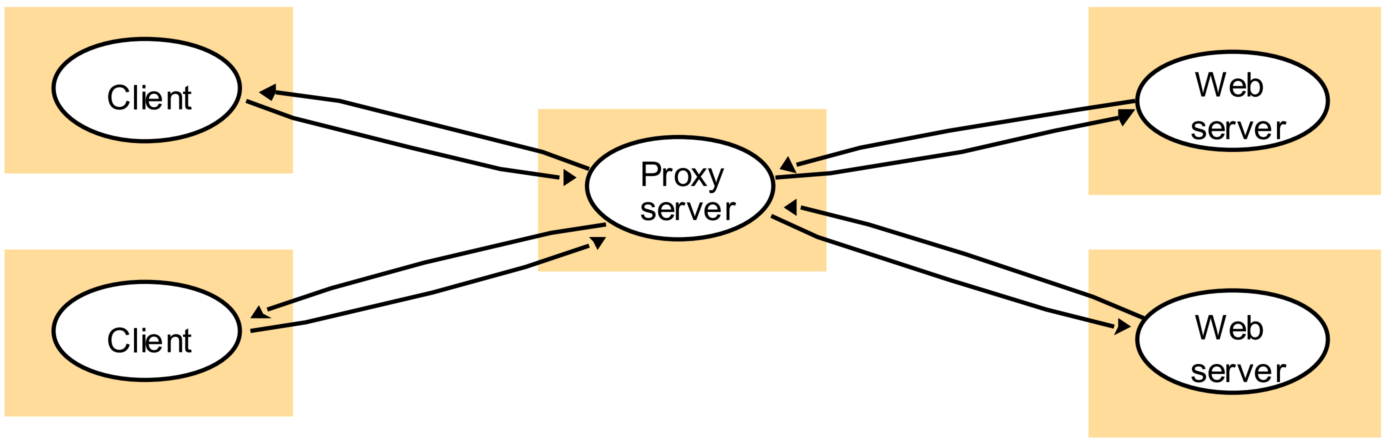
2. Peer-to-peer Systems



3. A Service by Multiple Servers

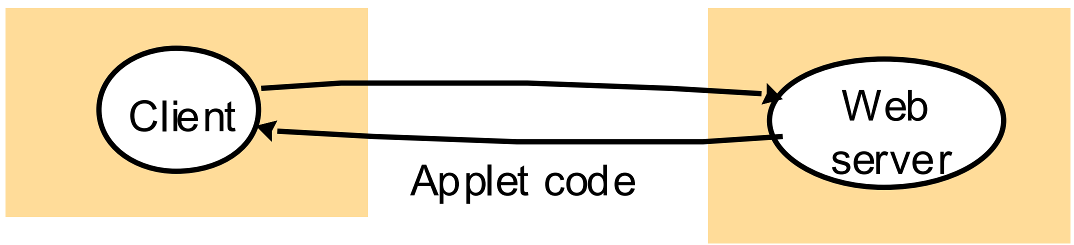


4. Web Proxy Server



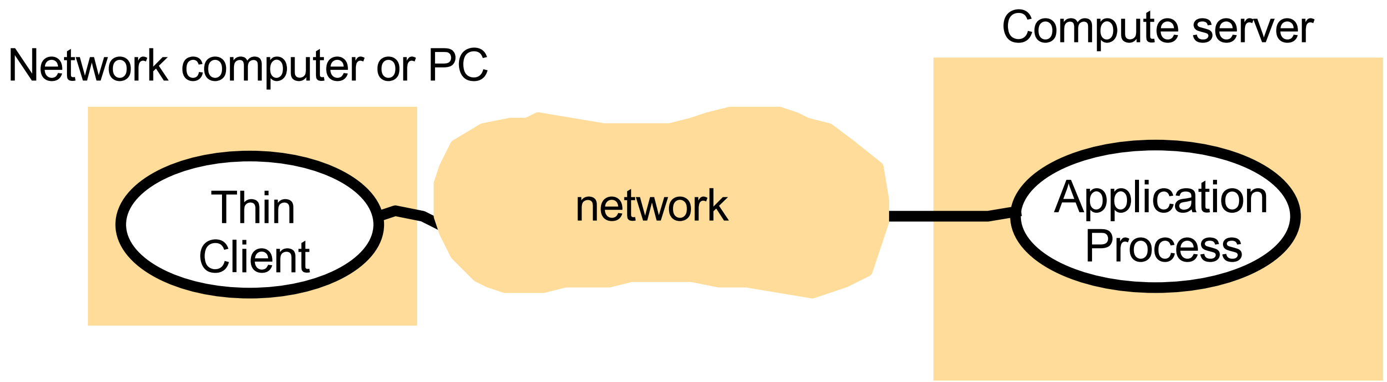
5. Web Applets

a) client request results in the downloading of applet code



b) client interacts with the applet

6. Thin Clients and Compute Servers



Introduction

- Overview (done)
- Goals (done)
- Software (done)
- Architecture (done)
- Examples (next)

Examples of Distributed Systems

- Cloud Computing and Xaas

Distributed Computing (old time)

- The Problem
  - Want to run compute/data intensive task
  - But don't have enough resources to run job locally
    - At least, to get results within sensible timeframe
  - Would like to use another, more capable resource
- Solution → Distributed Computing

Images: nasaimages, Extra Ketchup, Google Maps, Dave Page

Distributed Computing (Now)

- Compute *and* data – if you need more, you go somewhere else to get it
- Olden times - Small number of “fast” computers
  - Very expensive
  - Centralized
  - Used nearly all time
  - Time allocations for users

Cray-1 1976 - $8.8 mill, 160 MFLOPS, 8MB RAM
- PS4 ~1 TFLOP
- Smartphones ~200 MFLOPS

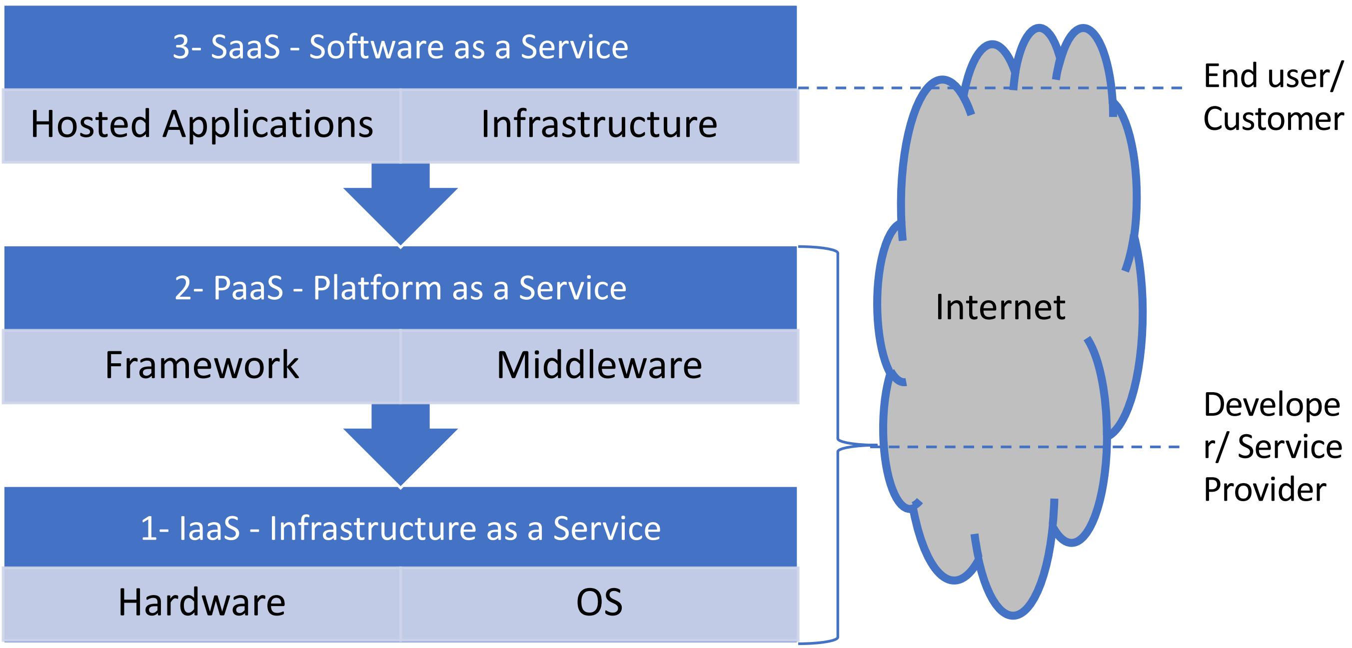
- Modern times
  - Cloud and Grid (next)

What is Cloud Computing?

- Many ways to define it (maybe one for every supplier of "cloud")
- Key characteristics:
  - On demand, dynamic allocation of resources – "elasticity"
  - Abstraction of resource
  - Self-managed
  - Billed for *what you use,* e.g., CPU, time, storage space
  - Standardized interfaces

[FZRL08] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Proceedings of Grid Computing Environments Workshop (GCE),* Austin, TX, USA, Nov. 2008, pp. 1–10

Cloud Architecture



- Cloud computing can deliver at any of these levels
- These levels are often blurred and routinely disputed!
- Resources provided on demand

IaaS – Infrastructure as a Service

- User gets access to (usually) virtualised hardware
  - Servers, storage, networking
  - Operating system
- User responsible for managing OS, middleware, runtime, data, application (development)
- e.g., Amazon EC2
  - Get complete virtualized PC (e.g., Linux instance)

PaaS – Platform as a Service

- Integrated development environment
  - e.g., application design, testing, deployment, hosting, frameworks for database integration, storage, app versioning, etc.
- Develop applications on top
- Responsible for managing data, application (development)
- Example - Google App Engine

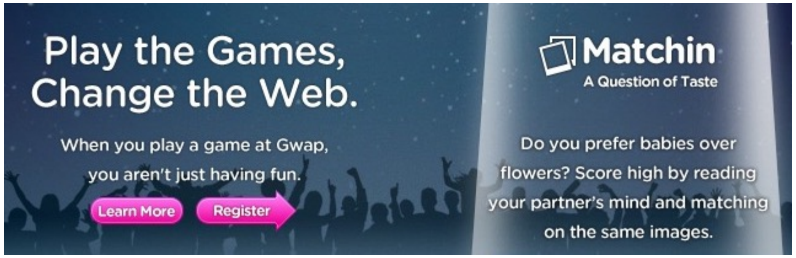
SaaS – Software as a Service

- Top layer consumed directly by end user – the ‘business’ functionality
- Application software provided, you configure it (more or less)
- Various levels of maturity:
    - **Level 1:** each customer has own customised version of application in own instance
    - **Level 2:** all instances use same application code, but configured individually
    - **Level 3:** single instance of application across all customers
    - **Level 4:** multiple customers served on load-balanced ‘farm’ of identical instances
    - Levels 3 & 4: separate customer data! (Somewhat similar to PaaS)
- e.g. Gmail, Google Sites, Google Docs, Facebook

Also HuaaS – Human as a Service

- Extraction of information from crowds of people
- Arbitrary (e.g., notable YouTube videos)
- On-demand task

Games with a Purpose



Amazon Mechanical Turk

Where to Apply Distributed Systems?

Application Domain	Associated Networked Application
Finance and commerce	E-commerce (e.g., Amazon and eBay, PayPal), online banking and trading
The information society	Web information and search engines, e-books, Wikipedia; social networking: Facebook and Instagram, Twitter.
Creative industries and entertainment	Online gaming, music and film in the home, user-generated content, e.g. YouTube, Flickr
Healthcare	Health informatics, on online patient records, monitoring patients
Education	E-learning, virtual learning environments; distance learning
Transport and logistics	GPS in route finding systems, map services: Google Maps, Google Earth
Science	The Grid as an enabling technology for collaboration between scientists
Environmental management	Sensor technology to monitor earthquakes, floods or tsunamis