
Distributed Systems (COMP9243)

Session 1, 2018



Ihor Kuz

cs9243@cse.unsw.edu.au

DISTRIBUTED SYSTEMS (COMP9243)

Lecture 1: Introduction

- ① Distributed Systems - what and why
- ② Hardware and Software
- ③ Goals
- ④ Overview - principles and paradigms
- ⑤ Course details
- ⑥ Erlang

DISTRIBUTED SYSTEMS

What is a *distributed system*?

→ Andrew Tannenbaum defines it as follows:

A distributed system is a collection of independent computers that appear to its users as a single coherent system.

→ Is there any such system? ~~Hardly!~~ Kind of

→ We will learn about the challenges in building “true” distributed systems

For the time being, we go by a weaker definition of distributed system:

A distributed system is a collection of independent computers that are used jointly to perform a single task or to provide a single service.

Examples of distributed systems

- Cray XK7 & CLE (massive multiprocessor)
- Distributed file system on a LAN
- Domain Name Service (DNS)
- Collection of Web servers: distributed database of hypertext and multimedia documents

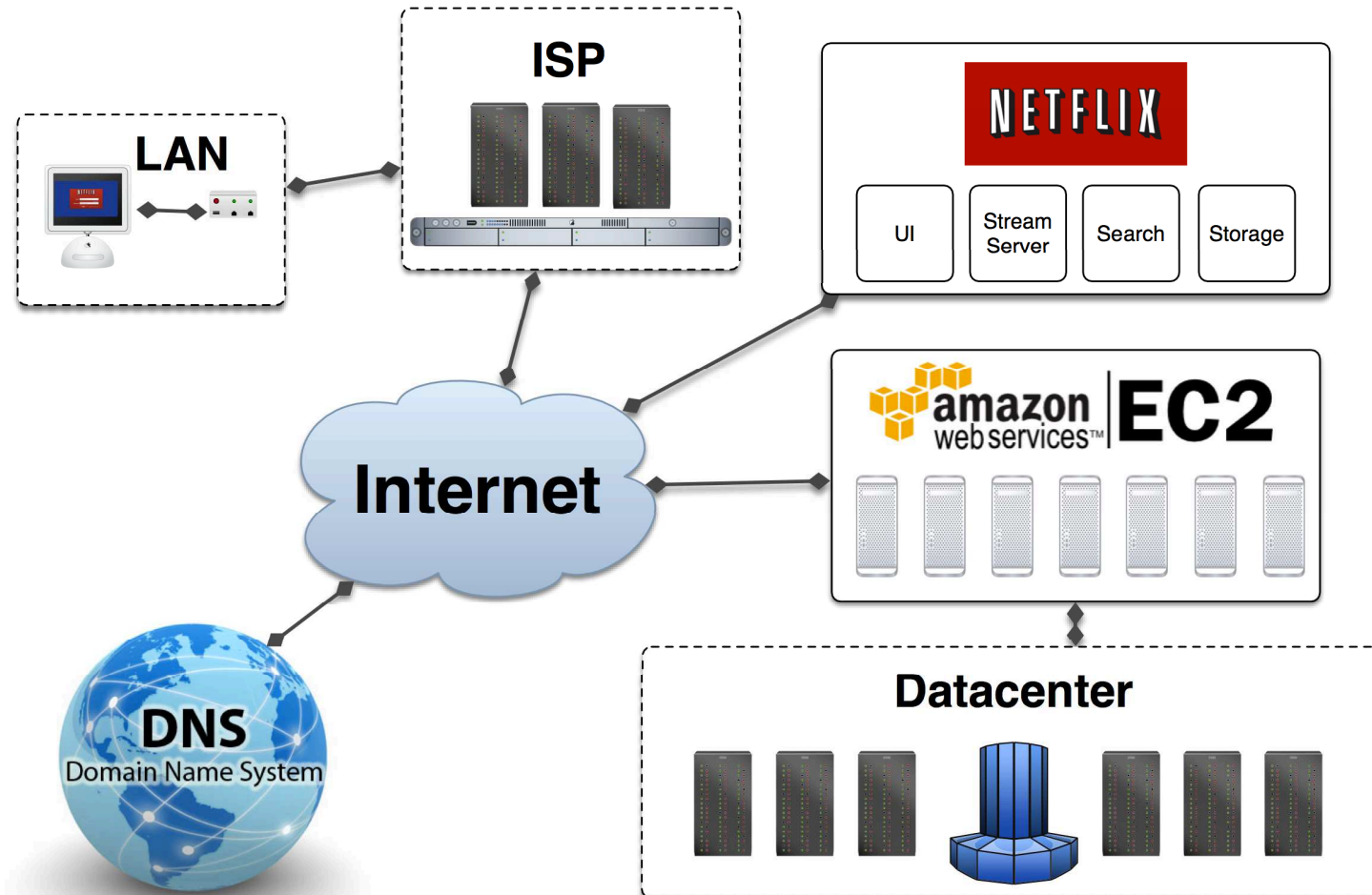
Find more examples of distributed systems:

Remember

A distributed system is a collection of independent computers that are used jointly to perform a single task or to provide a single service.

What's the difference between a distributed application and distributed system?

INTERDEPENDENCE OF DISTRIBUTED SYSTEMS



THE ADVANTAGES OF DISTRIBUTED SYSTEMS

What are economic and technical reasons for having distributed systems?

Cost. Better **price/performance** as long as commodity hardware is used for the component computers

Performance. By using the **combined processing and storage capacity** of many nodes, performance levels can be reached that are out of the scope of centralised machines

Scalability. Resources such as processing and storage capacity can be **increased incrementally**

Reliability. By having **redundant** components, the impact of hardware and software faults on users can be reduced

Inherent distribution. Some applications like the Web are **naturally distributed**

THE DISADVANTAGES OF DISTRIBUTED SYSTEMS

What problems are there in the use and development of distributed systems?

New component: network. Networks are needed to connect independent nodes, are subject to **performance limits**

Software complexity. Distributed software is more complex and harder to develop than conventional software; hence, it is more expensive and **harder to get right**

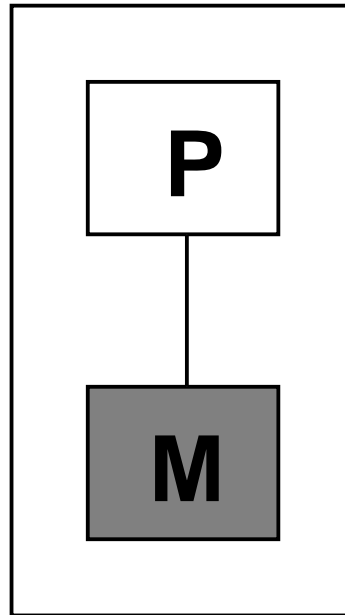
Failure. **More elements that can fail**, and the failure must be dealt with

Security. **Easier to compromise** distributed systems

Distributed systems are hard to build and understand
➔ this course is going to be very challenging!

HARDWARE ARCHITECTURE

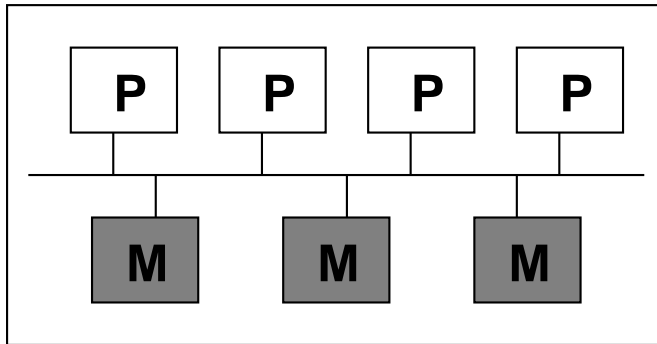
Uniprocessor:



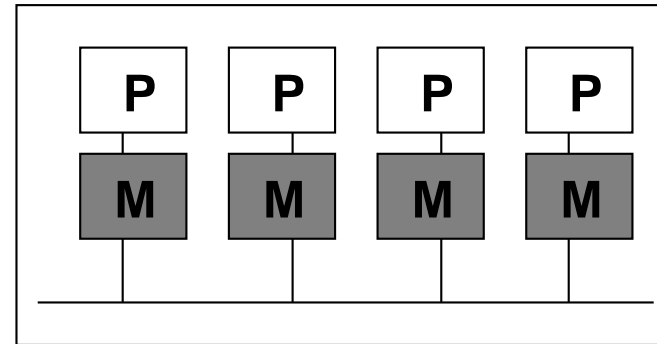
Properties:

- Single processor
- Direct memory access

Multiprocessor:



Uniform

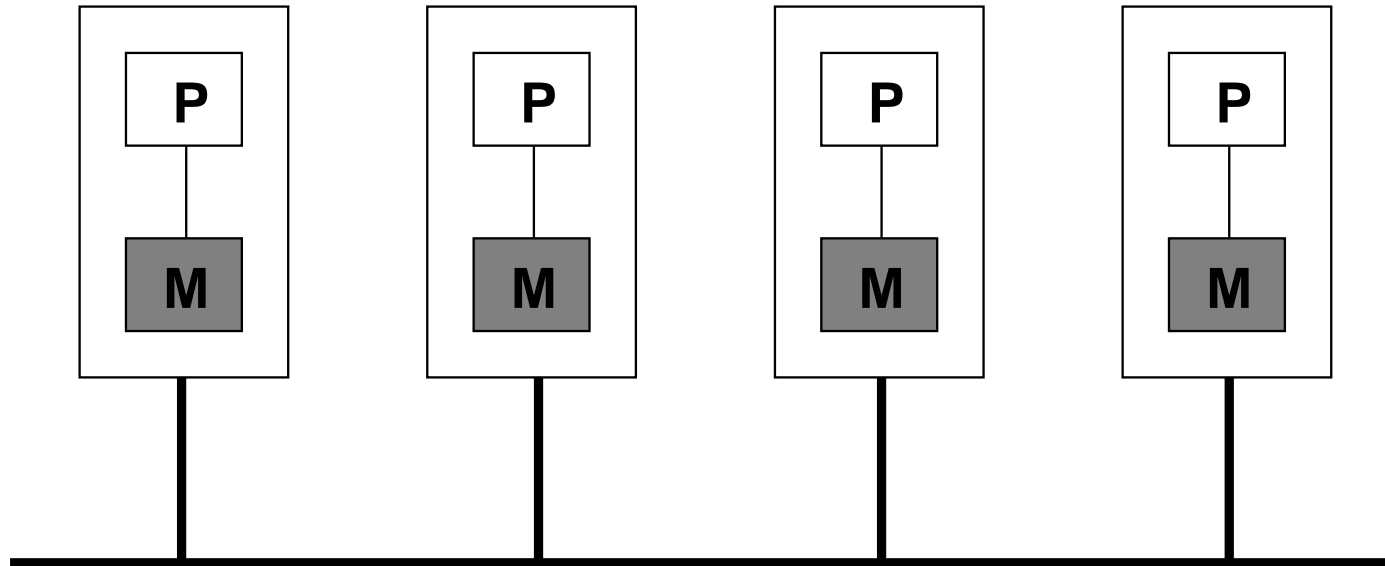


Nonuniform

Properties:

- Multiple processors
- Direct memory access
 - Uniform memory access (e.g., SMP, multicore)
 - Nonuniform memory access (e.g., NUMA)

Multicomputer:

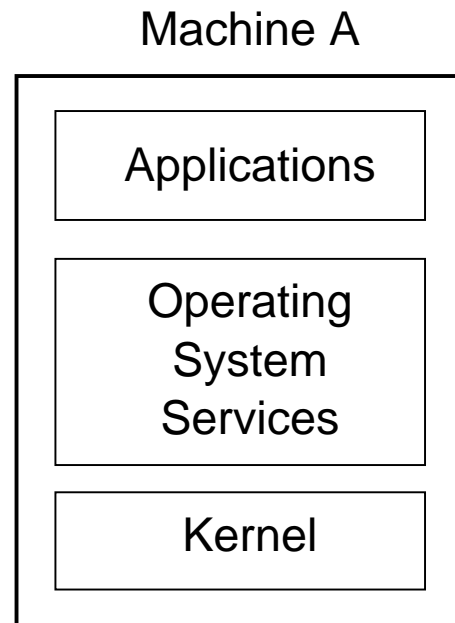


Properties:

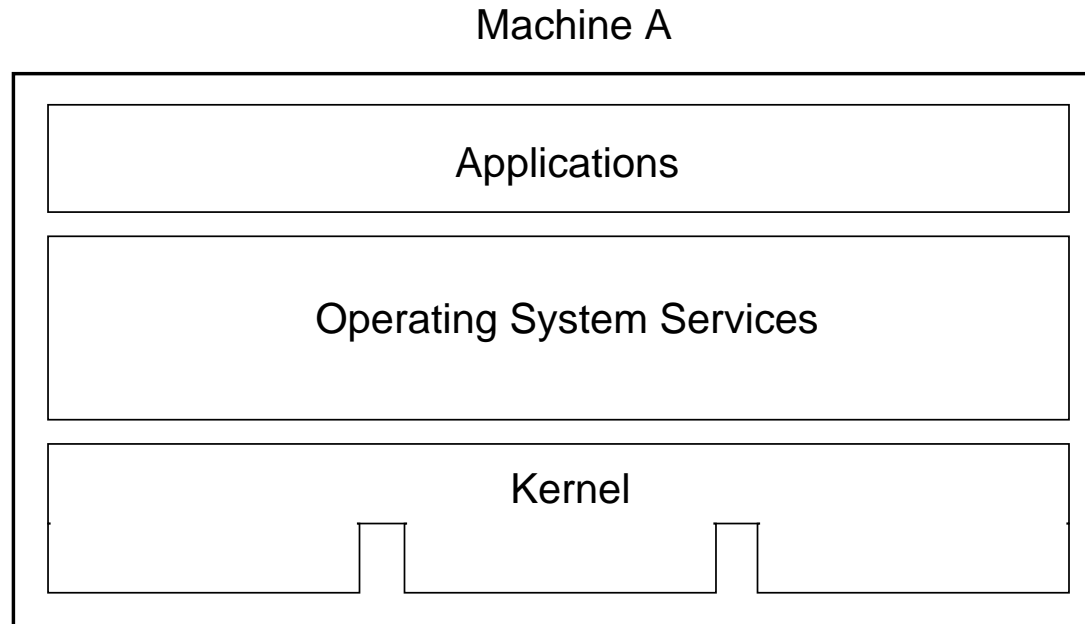
- Multiple computers
- No direct memory access
- Network
- Homogeneous vs. Heterogeneous

SOFTWARE ARCHITECTURE

Uniprocessor OS:



Multiprocessor OS:

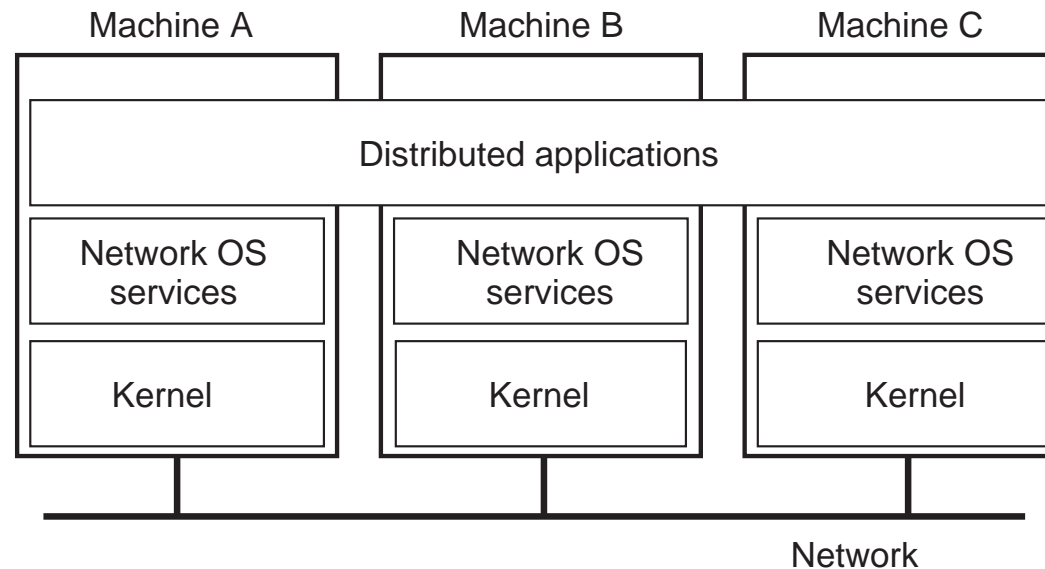


Similar to a uniprocessor OS but:

- Kernel designed to handle multiple CPUs
- Number of CPUs is transparent
- Communication uses same primitives as uniprocessor OS
- Single system image

What's the limitation here?

Network OS:

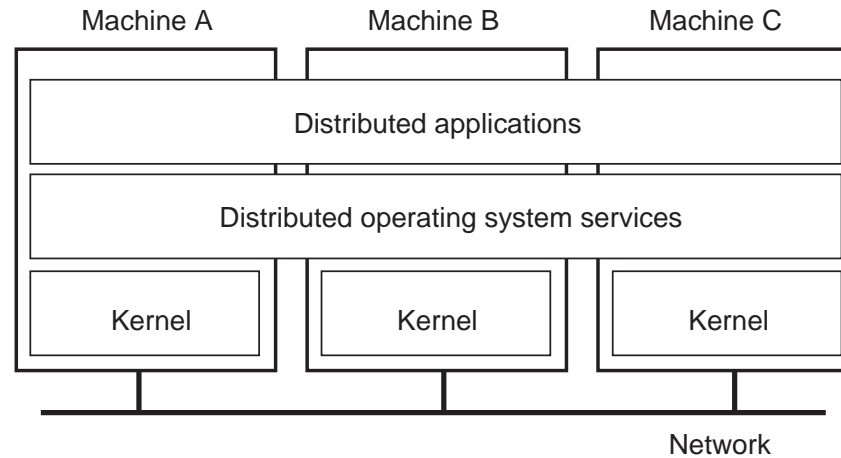


Properties:

- No single system image. Individual nodes are highly autonomous
- All distribution of tasks is explicit to the user
- Examples: Linux, Windows

What's the challenge with this approach?

Distributed OS:

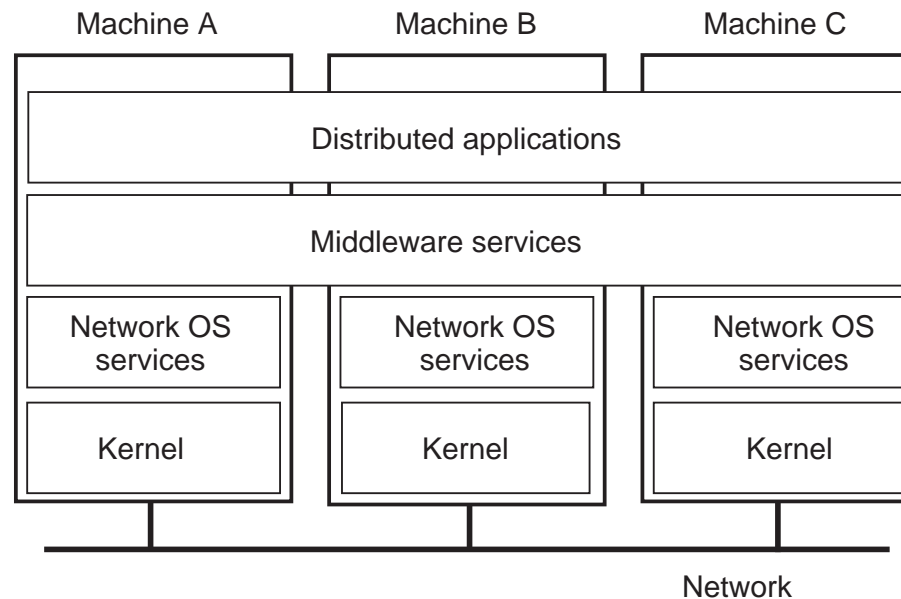


Properties:

- High degree of transparency
- Single system image (FS, process, devices, etc.)
- Homogeneous hardware
- Examples: Amoeba, Plan 9, Chorus, Mungi

Are there any problems with this approach?

Middleware:



Properties:

- System independent interface for distributed programming
- Improves transparency (e.g., hides heterogeneity)
- Provides services (e.g., naming service, transactions, etc.)
- Provides programming model (e.g., distributed objects)

Why is Middleware 'Winning'?:

- Builds on commonly available abstractions of network OSES (processes and message passing)
- Examples: RPC, NFS, CORBA, MQSeries, SOAP, REST, MapReduce
- Also languages (or language modifications) specially designed for distributed computing
- Examples: Erlang, Ada, Limbo, Go, etc.
- ✓ Usually runs in user space
- ✓ Raises level of abstraction for programming ➡ less error-prone
- ✓ Independence from OS, network protocol, programming language, etc. ➡ Flexibility
- ✗ Feature dump and bloated interfaces

DISTRIBUTED SYSTEMS AND PARALLEL COMPUTING

- Parallel computing: improve performance by using multiple processors per application
- There are two flavours:
 1. **Shared-memory systems:**
 - Multiprocessor (multiple processors share a single bus and memory unit)
 - SMP support in OS
 - Much simpler than distributed systems
 - Limited scalability
 2. **Distributed memory systems:**
 - Multicomputer (multiple nodes connected via a network)
 - These are a form of distributed systems
 - Share many of the challenges discussed here
 - Better scalability & cheaper

DISTRIBUTED SYSTEMS IN CONTEXT

Networking:

- Network protocols, routing protocols, etc.
- Distributed Systems: **make use of networks**

Operating Systems:

- Resource management for single systems
- Distributed Systems: **management of distributed resources**

This Course:

- Generalised solutions to distributed systems problems and challenges
- Infrastructure software to help build distributed applications

BASIC GOALS OF DISTRIBUTED SYSTEMS

We want distributed systems to have the following properties:

- Transparency
- Dependability
- Scalability
- Performance
- Flexibility

This course will examine approaches and techniques for designing and building distributed systems that achieve these goals.

TRANSPARENCY

Concealment of the separation of the components of a distributed system (single image view).

There are a number of forms of transparency

Access: Local and remote resources accessed in same way

Location: Users unaware of location of resources

Migration: Resources can migrate without name change

Replication: Users unaware of existence of multiple copies

Failure: Users unaware of the failure of individual components

Concurrency: Users unaware of sharing resources with others

Is transparency always desirable? Is it always possible?

DEPENDABILITY

- Dependability of distributed systems is a double-edged sword:
 - Distributed systems promise higher **availability**:
 - Replication
 - But **availability may degrade**:
 - More components ➡ more potential points of failure
- Dependability requires consistency, security, and fault tolerance

SCALABILITY

A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity

(B. Clifford Neuman)

Scale has three dimensions:

Size: number of users and resources (problem: overloading)

Geography: distance between users and resources (problem: communication)

Administration: number of organisations that exert administrative control over parts of the system (problem: administrative mess)

Note:

- Scalability often conflicts with (small system) performance
- Claim of scalability is often abused

Scaling Up or Out?

Vertical Scaling: Scaling UP Increasing the resources of a single machine

Horizontal Scaling: Scaling OUT Adding more machines

Techniques for scaling:

- Hiding communication latencies (asynchronous communication, reduce communication)
- Distribution (spreading data and control around)
- Replication (making copies of data and processes)
- Decentralisation

Decentralisation

Avoid centralising:

- Services (e.g., single server)
- Data (e.g., central directories)
- Algorithms (e.g., based on complete information).

With regards to algorithms:

- Do not require machine to hold complete system state *Why?*
- Allow nodes to make decisions based on local info *Why?*
- Algorithms must survive failure of nodes *Why?*
- No assumption of a global clock *Why?*

Decentralisation is *hard*

PERFORMANCE

- Any system should strive for maximum performance
- In distributed systems, performance directly conflicts with some other desirable properties
 - Transparency
 - Security
 - Dependability
 - Scalability

How?

NUMBERS EVERY PROGRAMMER SHOULD KNOW

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns = 3 us
Send 2K bytes over 1 Gbps network	20,000 ns = 20 us
Read 1 MB sequentially from memory ..	250,000 ns = 250 us
Round trip within same datacenter ...	500,000 ns = 0.5 ms
Disk seek	10,000,000 ns = 10 ms
Read 1 MB sequentially from disk .	20,000,000 ns = 20 ms
Send packet CA->Netherlands->CA .	150,000,000 ns = 150 ms

(from Peter Norvig, Jeff Dean, see also http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html)

FLEXIBILITY

- Build a system out of (only) required components
- Extensibility: Components/services can be changed or added
- Openness of interfaces and specification
 - allows reimplementations and extensions
- Interoperability
- Separation of policy and mechanism
 - standardised internal interfaces

COMMON MISTAKES

False assumptions commonly made:

- ① Reliable network
- ② Zero latency
- ③ Infinite bandwidth
- ④ Secure network
- ⑤ Topology does not change
- ⑥ One administrator
- ⑦ Zero transport cost
- ⑧ Everything is homogeneous

PRINCIPLES

Several key principles underlying the functioning of all distributed systems

- System Architecture
- Communication
- Partitioning, Replication and Consistency
- Synchronisation & Coordination
- Naming
- Fault Tolerance
- Security

Discussion of these principles will form the core content of the course

PARADIGMS

Most distributed systems are built based on a particular paradigm (or model)

- Shared memory
- Distributed objects
- Distributed file system
- Distributed coordination
- Service Oriented Architecture and Web Services
- Distributed Database
- Shared documents
- Agents

This course will cover the first five in detail.

MISCELLANEOUS 'RULES OF THUMB'

Trade-offs Many of the challenges provide conflicting requirements. For example better scalability can cause worse overall performance. Have to make trade-offs - what is more important?

Separation of Concerns Split a problem into individual concerns and address each separately

End-to-End Argument Some communication functions can only be reliably implemented at the application level

Policy vs. Mechanism A system should build mechanisms that allow flexible application of policies. Avoid built-in policies.

Keep It Simple, Stupid make things as simple as possible, *but no simpler.*

READING LIST

End-to-end Arguments in System Design A classic paper arguing the end-to-end argument with excellent examples.

A Note on Distributed Computing Another classic paper showing the dangers of too much transparency in RPC-based distributed systems.

Fallacies of Distributed Computing Explained A good explanation of the 8 common mistakes made by architects and designers of distributed systems.

Scale in Distributed Systems A really good paper to read if you are interested in understanding more about scalability in distributed systems.

OVERVIEW OF COURSE

- ① Introduction and Erlang
- ② System Architecture and Communication
- ③ Replication and Consistency, Distributed Shared Memory
- ④ Synchronisation and Coordination
- ⑤ Dependability and Fault Tolerance
- ⑥ Naming
- ⑦ Distributed File Systems
- ⑧ Middleware, Distributed Objects, Publish/Subscribe, SOA, Web Services
- ⑨ Cloud Computing
- ⑩ Security

Extras:

- ① Distributed Systems in Practice
- ② Parallel Programming and Clusters
- ③ Research and Other Topics

PRACTICAL COURSE DETAILS

→ Course Outline Page

<http://www.cse.unsw.edu.au/~cs9243/outline.html>

→ Papers: classic and research: some mandatory, some optional

→ Homework/Exercises: Familiarisation, DS programming

→ Assignments: 3 assignments. 100 marks total.

→ Exam: Open book exam, 100 marks

→ Final Mark:

- weighted average: exam mark (60%) and total assignment mark (40%).
- Exam mark must be at least 50% of maximum possible exam mark.

Note:

Difficult course. Lots of work. Be prepared.

And start the assignments on time!

HOMework

Examples of Distributed Systems:

- Choose an existing distributed system and
 - ① Research its structure (i.e. what is its internal architecture?)
 - ② Evaluate how it satisfies each of the goals discussed

Hacker's edition:

- For your chosen system:
 - ① Are there any obvious mistakes in the architecture and design?
 - ② Are there any strange design decisions? Why might they have been made?