Distributed Systems

DAT520 - Spring 2026

Introduction

Prof. Hein Meling

University
of Stavanger

Distributed Systems
A Whirlwind Tour

* Computing

DISTRIBUTED COMPUTING

DECENTRALIZED COMPUTING

CENTRALIZED COMPUTING

AUTONOMOUS COMPUTING

PARALLEL COMPUTING

VOLUNTEER COMPUTING

CLUSTER COMPUTING

TRUSTWORTHY COMPUTING

GRID COMPUTING

SERVICE-ORIENTED COMPUTING

UTILITY COMPUTING

CLOUD COMPUTING

PERVASIVE COMPUTING

EDGE COMPUTING

FOG COMPUTING

UBIQUITOUS COMPUTING

SERVERLESS COMPUTING

What is this?

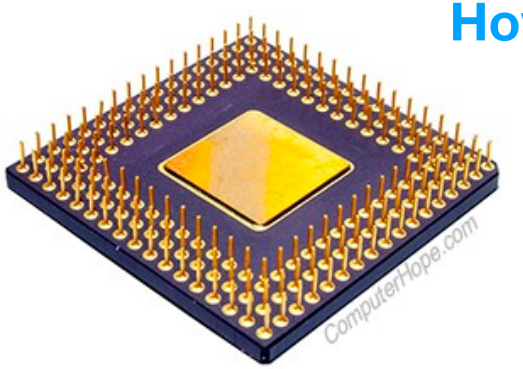
What is this?

Cray-1 supercomputer (1975)

160 MFLOPS

Cori @ Berkeley Lab

Paradigms in Computing



How do we increase speed?

4 GHz processor performs
4,000,000,000 clock cycles per second

High Performance Computing (HPC)

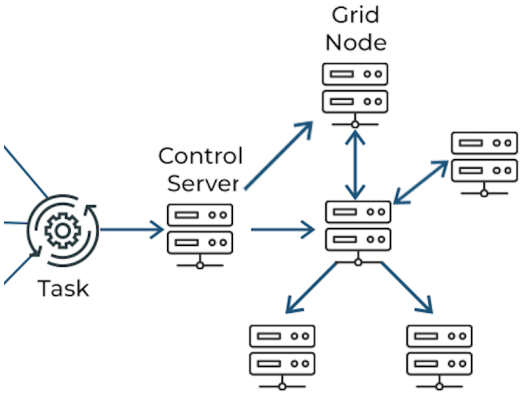
Parallel Computing

1970s

Massively parallel processors are made by chaining together hundreds or thousands of inexpensive commercial microprocessors.

Electric Power Grid ~ Grid / Utility Computing

Paradigms in Computing



How do we reduce cost of computing, increase reliability, and increase flexibility?

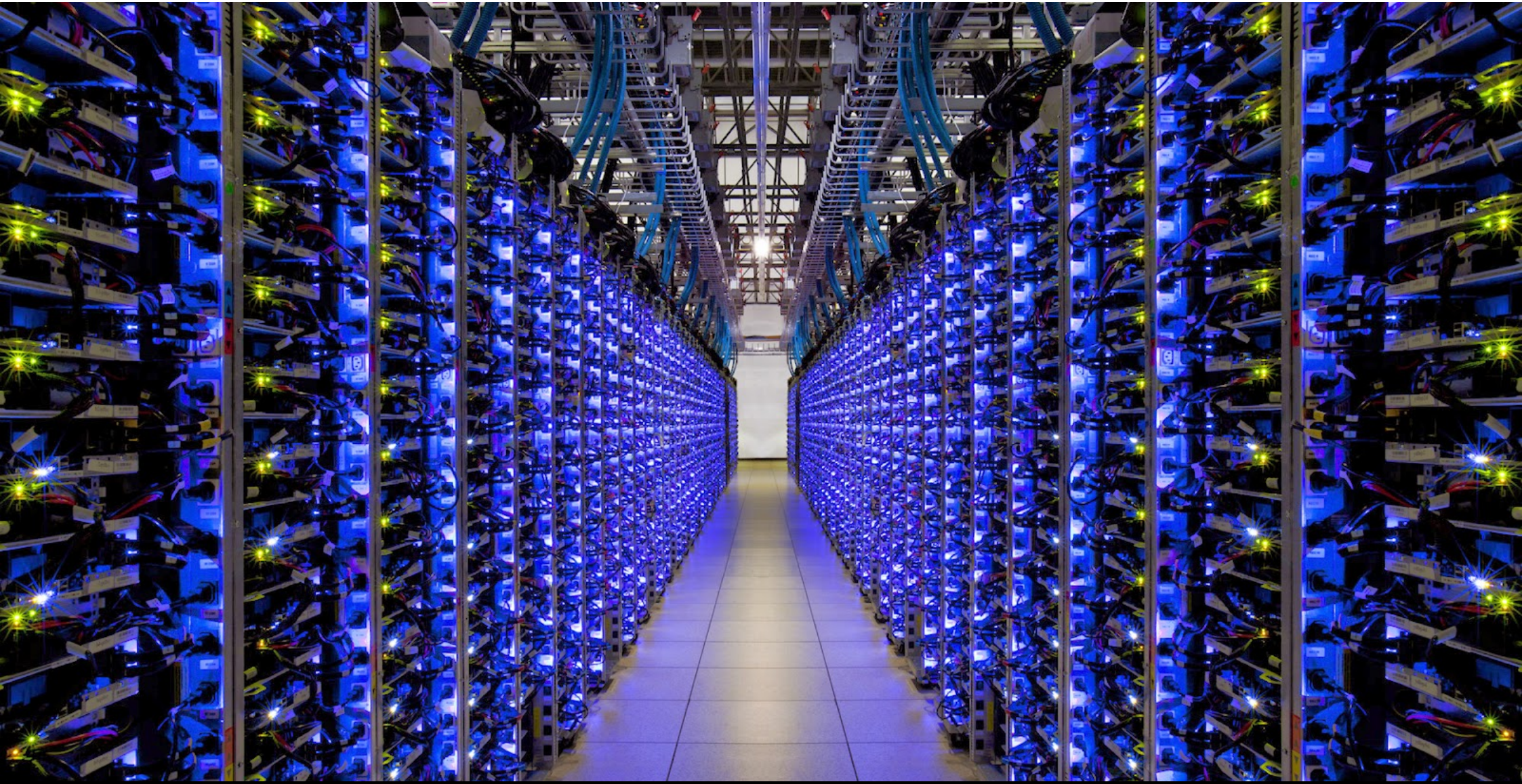
- Open Science Grid (US Funded)
- European Grid Infrastructure

High Performance Computing (HPC)

Grid Computing

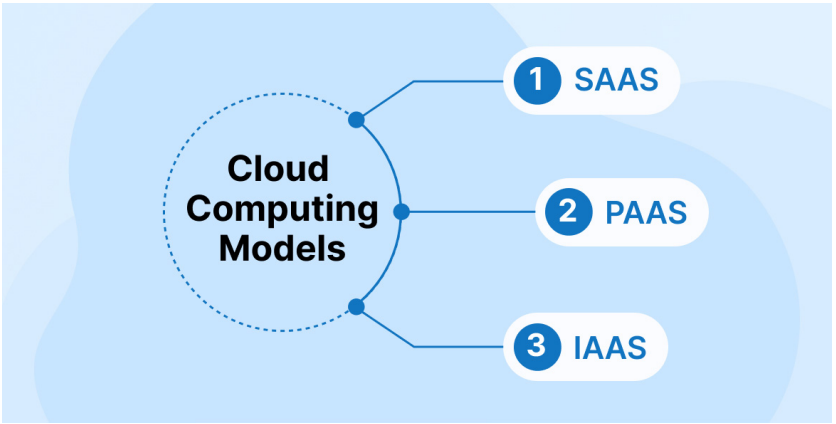
1990s

The grid combines various computing resources, databases, and measuring devices that comprise a pool of resources for coordinated, integrated and flexible shared use.



Cloud Computing

Paradigms in Computing



How do we scale?

Infrastructure as a Service (IaaS)	Amazon EC2 / S3
Platform as a Service (PaaS)	Google App Engine
Software as a Service (SaaS)	Salesforce

*** as a Service**

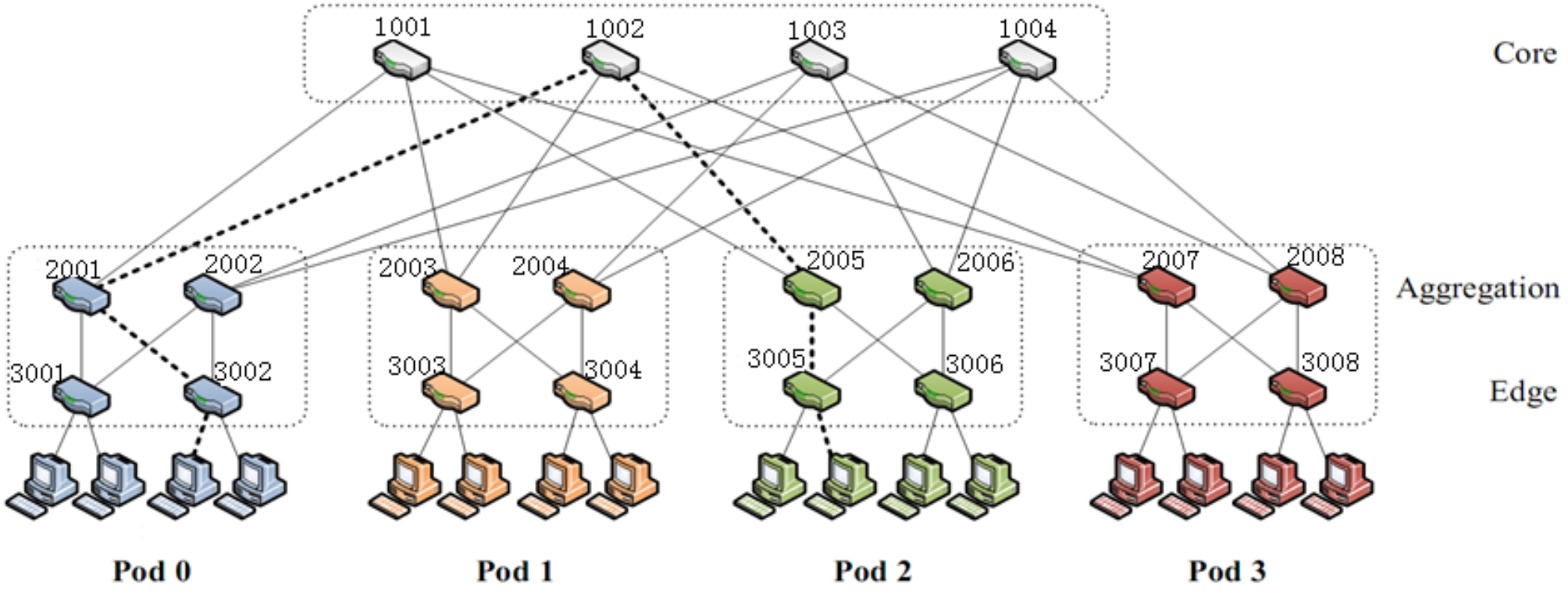
Cloud Computing

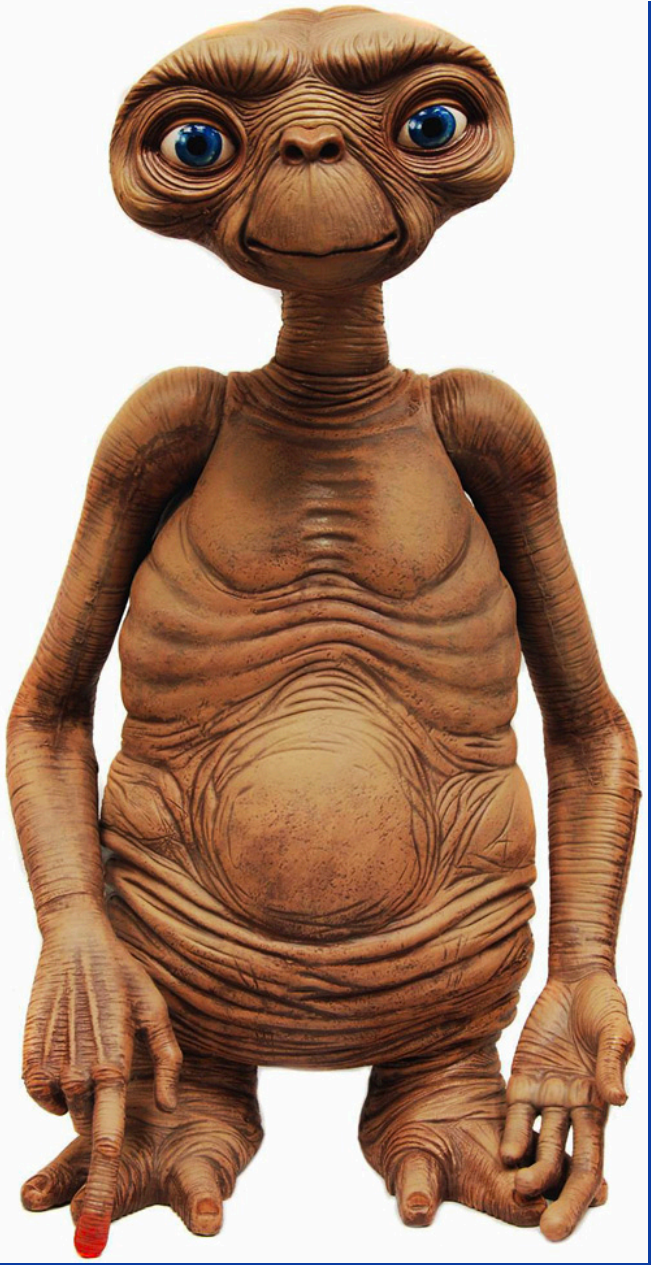
(today)

Economies of scale: a pool of abstracted, virtualized, dynamically-scalable, computing power, storage, platforms, and services are delivered on demand over the Internet.

How can we connect tens of thousands of machines in a data center together?

Small Fat Tree Example

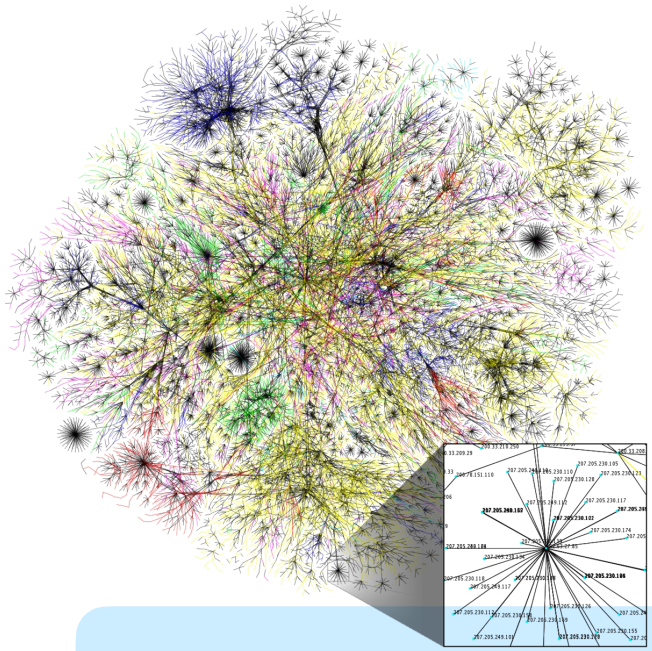




SETI HOME

SETI@home was a scientific experiment, based at UC Berkeley, that used Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). You could participate by running software that downloads and analyzes radio telescope data.

Paradigms in Computing



How do we scale?

Millions of *unreliable* and *heterogeneous* machines

High Performance Computing (HPC)

Volunteer Computing

Internet-connected computers, volunteered by their owners, as a source of computing power and storage.

Distributed Systems

Computer Networks: the autonomous computers are explicitly visible and must be specifically addressed

Distributed systems: the multiple autonomous computers and components are transparent to the user

Parallel computers: single system view without physical separation

Many problems in common:

- Scheduling
- Load balancing
- Resource and data sharing/distribution

Networks are in some sense also distributed systems (e.g. name services) and every distributed system relies on services provided by a network

Distributed systems may serve the same purpose as parallel computers: high performance

Distributed Systems Examples

Distributed Systems - Examples

Web Search

Search engines need to index the entire content of the web.

Google's index contains **hundreds of billions of webpages**.

- physical infrastructure of networked computers across data centers
- distributed file system that supports very large files
- distributed storage system with very large datasets
- distributed locking and agreement
- a programming model for very large parallel and distributed computation

Distributed Systems - Examples

Generative Model Learning and Inference

AI models also need to encode the entire content of the web.

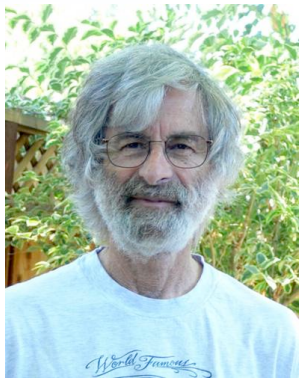
- massive parallel processing capabilities: CPUs, GPUs, Tensor PUs, Neural Engine
- efficient data pipelines: for feeding data into training and inference processes
- scalable network infrastructure: to handle high volume of data transfer between nodes during training and inference
- scalable model serving infrastructure: to handle thousands of simultaneous inference requests

Distributed Systems Definitions

"A system in which hardware or software components located at networked computers **communicate and coordinate their actions** only by message passing." [Coulouris]

"A distributed system is a collection of independent computers that **appear to the users of the system as a single computer**." [van Steen/Tanenbaum]

"A distributed system is one in which the **failure of a machine you have never heard of can cause your own machine to become unusable.**" [Lamport]

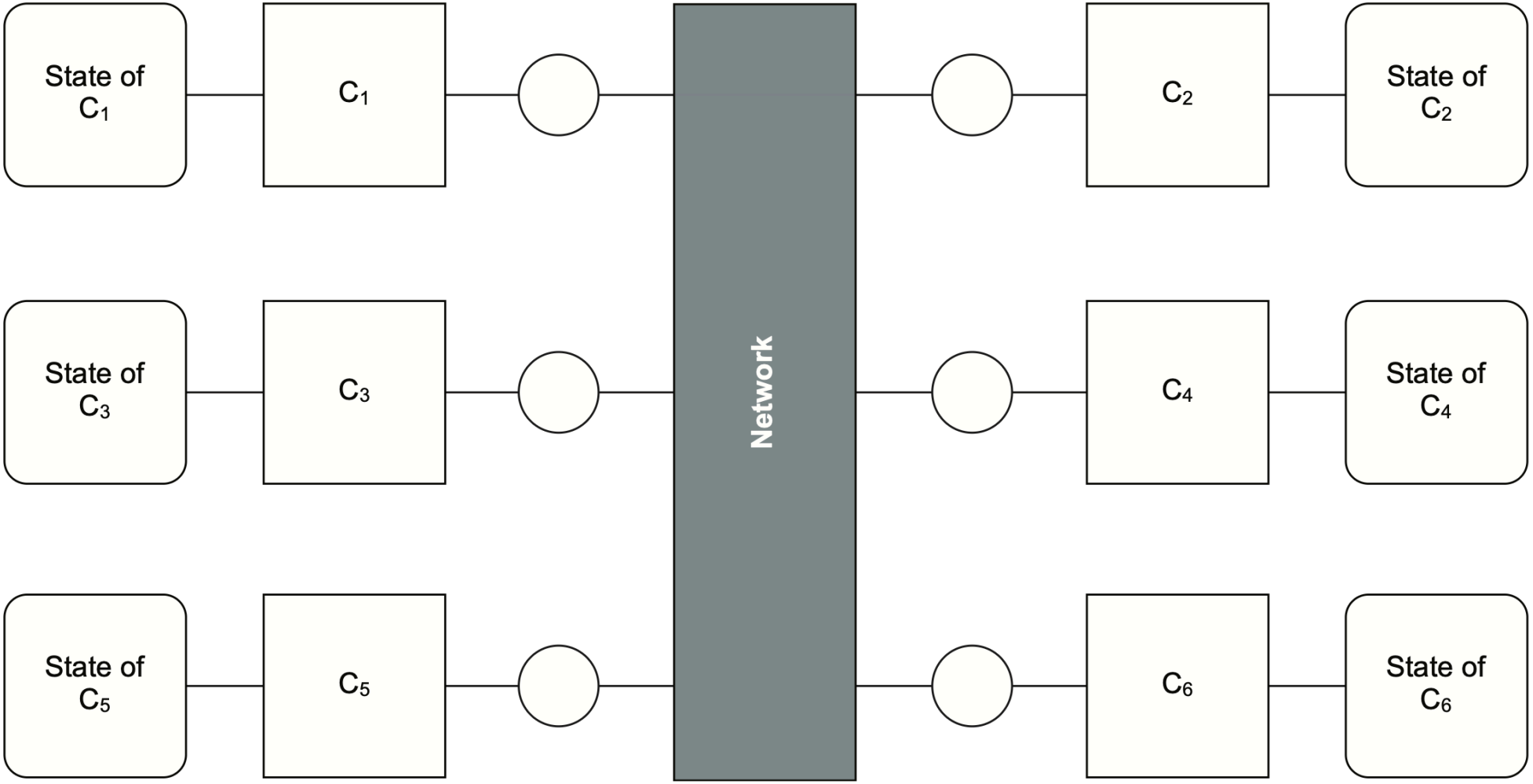


Distributed Systems - Definitions

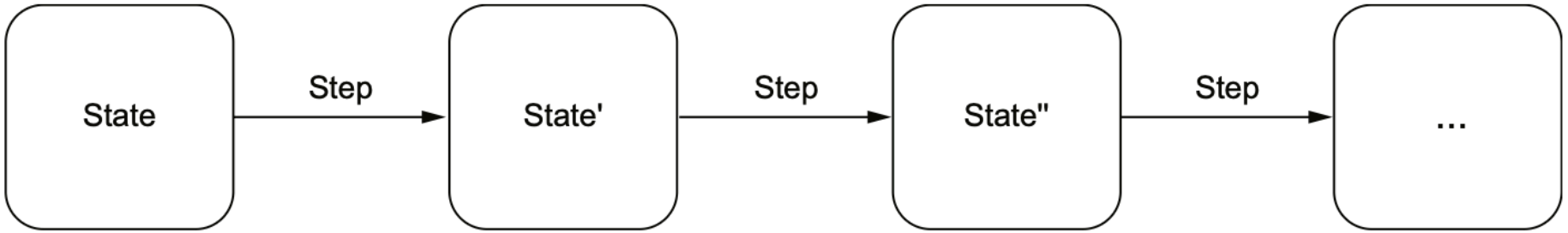
What they have in common?

- Distributed hardware (computers)
- Distributed data
- Distributed control
- Connected through some network

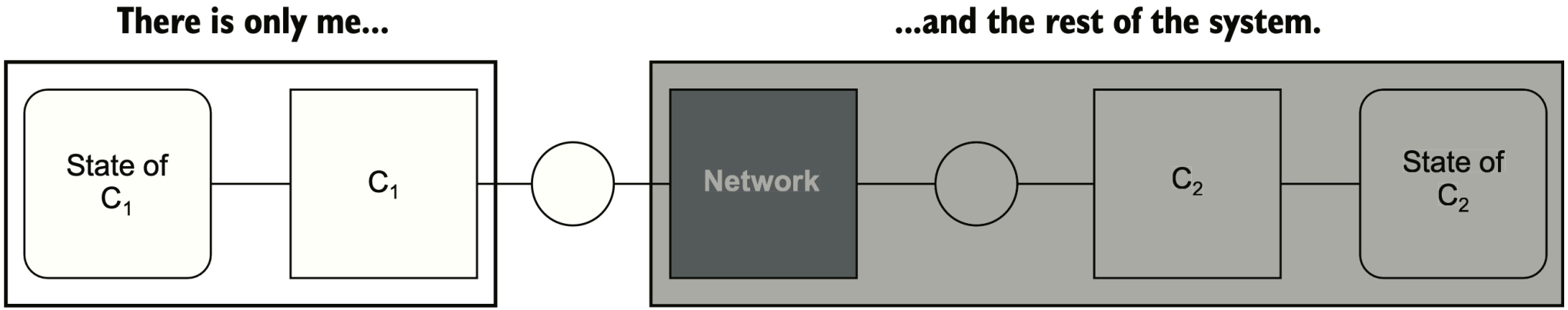
Distributed Systems - Definitions



Distributed Systems - Definitions



Distributed Systems - Definitions



Joke time!

In theory, distributed systems and local systems are the same.

In theory, distributed systems and local systems are the same.

In practice, they are not.

Why Distributed Systems?

Inherent Distribution

Distribution as an Artifact

Distributed Systems: Why?

A distributed system should

- make resources easily accessible;
- hide the fact that resources are distributed across a network;
- be open;
- be scalable.

Distributed Systems: Why?

Make resources easily accessible

Example resources: compute and storage facilities, data, files, services, and networks.

There are many reasons for wanting to share resources.

One obvious reason is economics.

Distributed Systems: Why?

Hide that resources are distributed across a network

Transparency	Description
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located **(Naming)**
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Distributed Systems: Why?

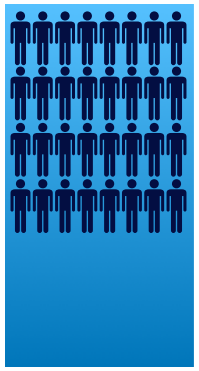
Be open

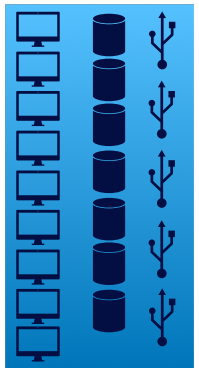
Interoperability, composability, and extensibility

An **open distributed system** is a system that offers components that can **easily be used** by, or **integrated into** other systems.

Often **itself composed** of components that **originate from elsewhere**.

Distributed Systems: Why?



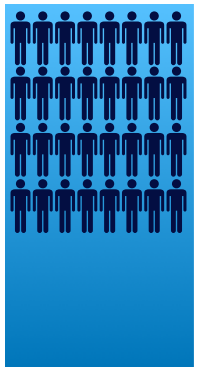


Be scalable

- Size scalability
- Geographical scalability
- Administrative scalability

Three scalability dimensions

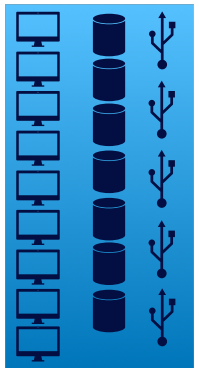
Distributed Systems: Why?



Be scalable

Size scalability

- more users
- more resources

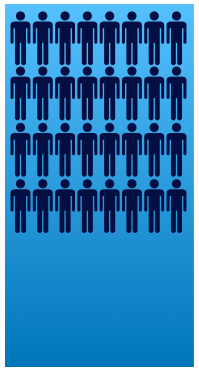


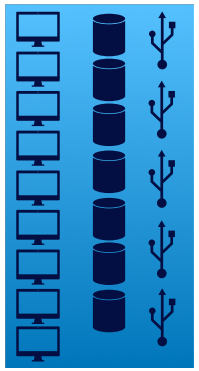
Potential bottlenecks:

computation, storage, and network capacity

Three scalability dimensions

Distributed Systems: Why?





Be scalable

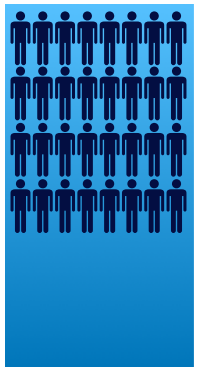
Three scalability dimensions

Geographical scalability

Users and resources may lie far apart, but the fact that communication delays may be significant is hardly noticed.

Portability to wide area networks: synchronous communications, reliability, bandwidth, multipoint communication.

Distributed Systems: Why?

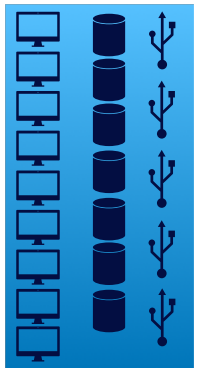


Be scalable

Three scalability dimensions

Administrative scalability

Can still be easily managed even if it spans many independent administrative organizations.



Scalability (continued)

Scalability Problems

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

Scaling Techniques

Hiding communication latencies

- geographical scalability
- asynchronous communication

Move stuff closer to consumer (e.g., use a CDN)

Don’t wait for responses; do something else while waiting

Partition and distribution

Partition data and computation across many computers

Replication

- availability
- load balance
- latency

Make copies of data (and computation) available at many machines

Scalability Problems

Inconsistencies & Global synchronization

- **Observation 1**: Applying scaling techniques is easy, except:
  - Having multiple copies (cached or replicated) leads to **inconsistencies**; modifying one copy makes it different from other copies
  - Always keeping copies consistent requires **global synchronization** of every modification
  - Global synchronization does not scale

Scalability Problems

Inconsistencies & Global synchronization

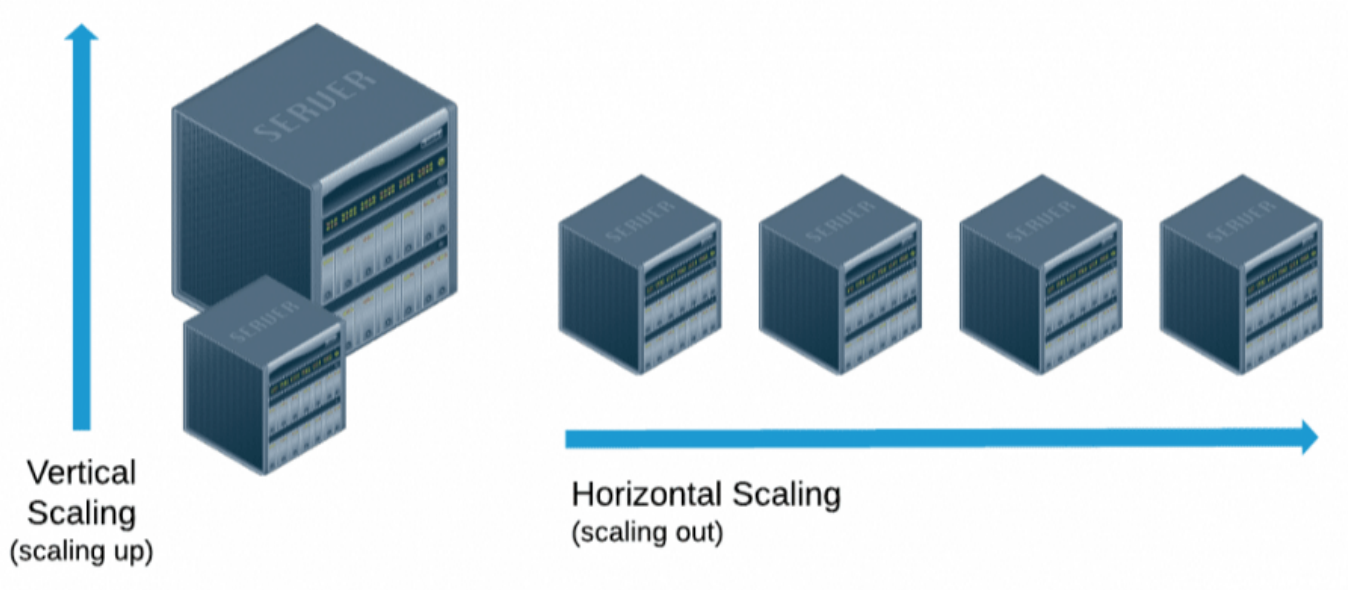
- **Observation 2**: Applying scaling techniques is easy, except:
  - We may sometimes **tolerate inconsistencies**; hence reduce the need for global synchronization
  - Tolerating inconsistencies is **application dependent**

Scaling Techniques

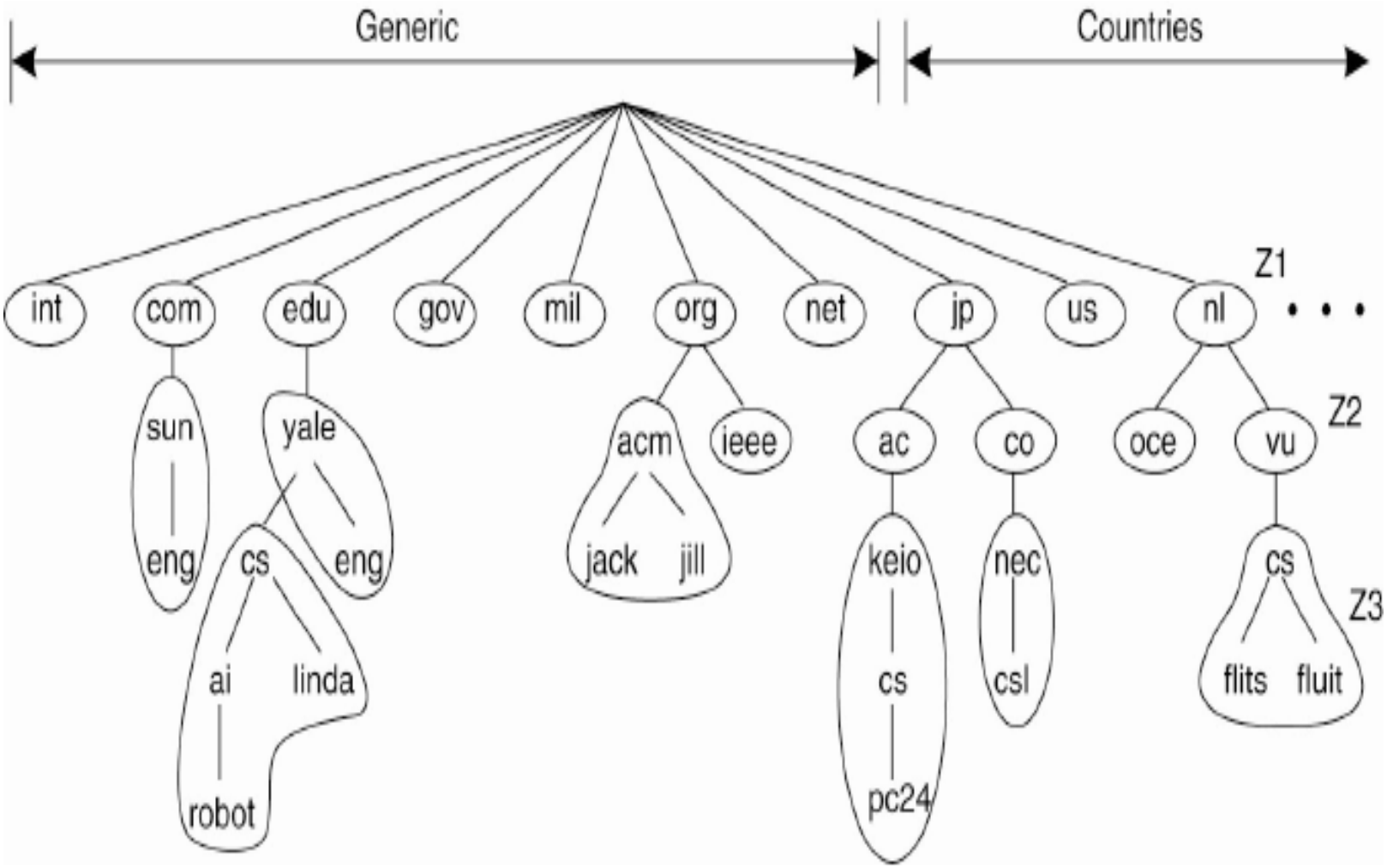
Performance problems due to limited (server & network) capacity

Scaling up (vertical)
improve capacity

Scaling out (horizontal)
add more machines



Scaling Techniques: Example



Failure Handling & Concurrency

Failure Handling

Challenge: Consistency, Consistency…

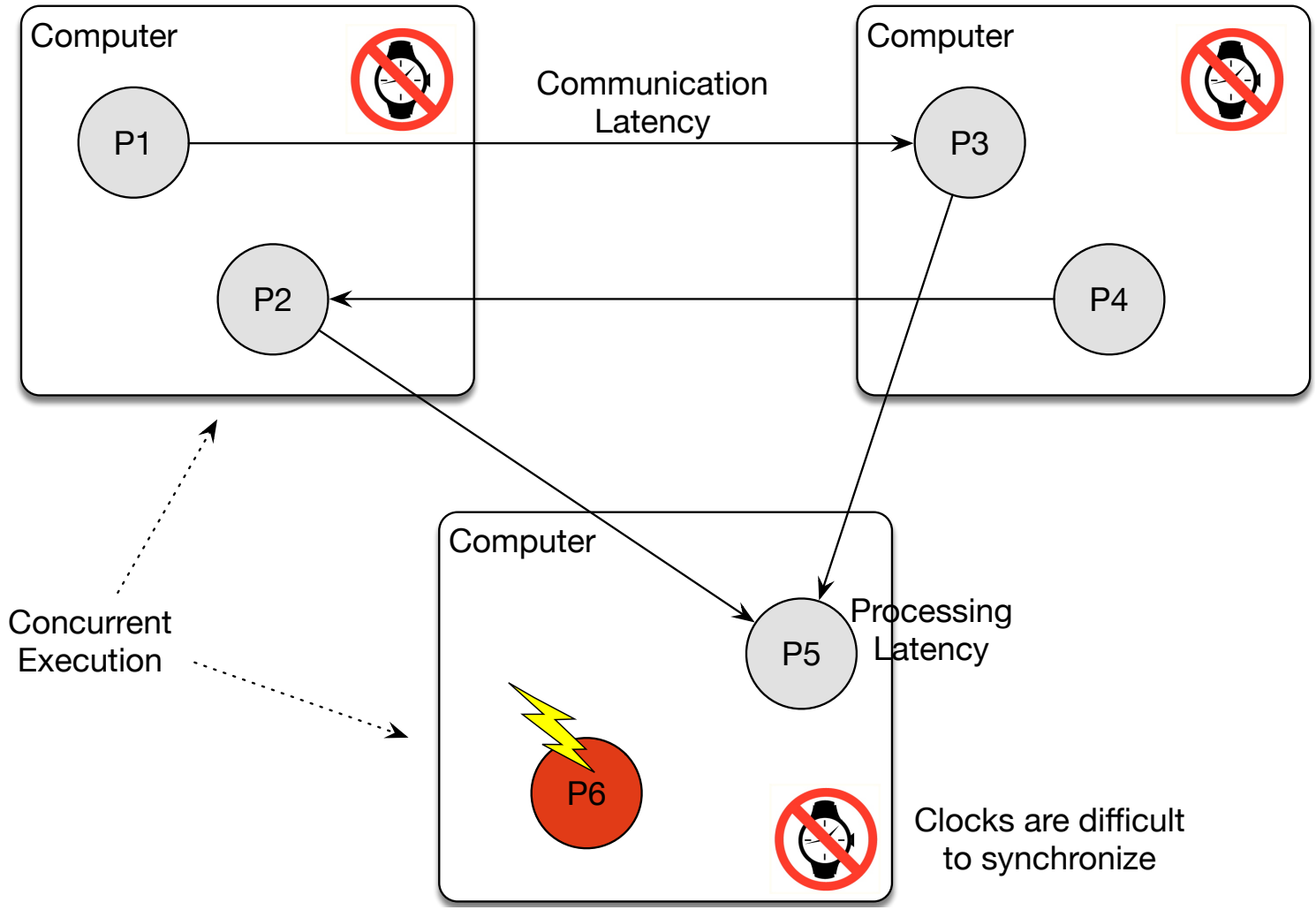
- Failure detection
  - Checksum, timeout
- Failure masking (also called Fault tolerance)
  - Message retransmission
  - Failover to another server (requires replicated servers)
- Failure recovery
  - Rollback a transaction

Challenge: Consistency, Consistency…

- Execute programs in parallel on different machines
  - Increase efficiency
- Need to synchronize access to shared resources
  - Across the network
  - To avoid conflicting updates (consistency!)

Characteristics of a Distributed System

Characteristics of a Distributed System



Characteristics of a Distributed System

Summary of characteristic challenges

- Concurrent execution of programs
- Communication latency
- Processing latency
- Lack of a global clock
- Components can fail
  - Can lead to partial system failures

Characteristics of a Distributed System

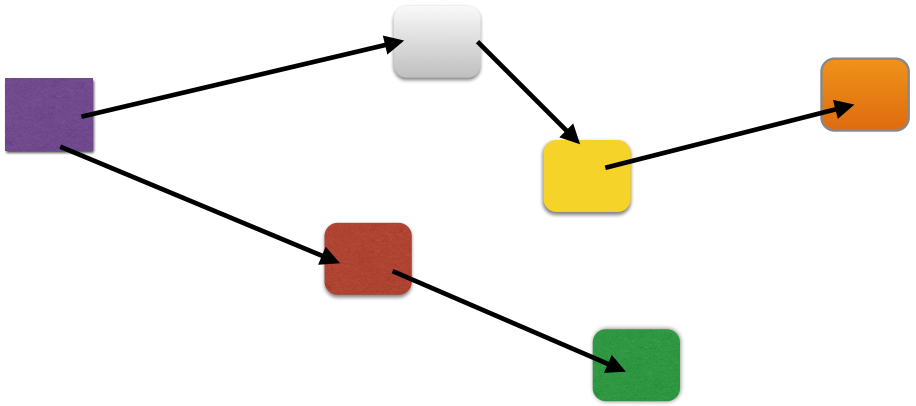
No globally synchronized clock

- Cannot use timestamps
  - To **accurately** synchronize and coordinate actions of different components

Characteristics of a Distributed System

Dependencies cause failures

- Dependencies between distributed components may exacerbate the impact of failures
- Distributed systems **complicates** the life of **developers**

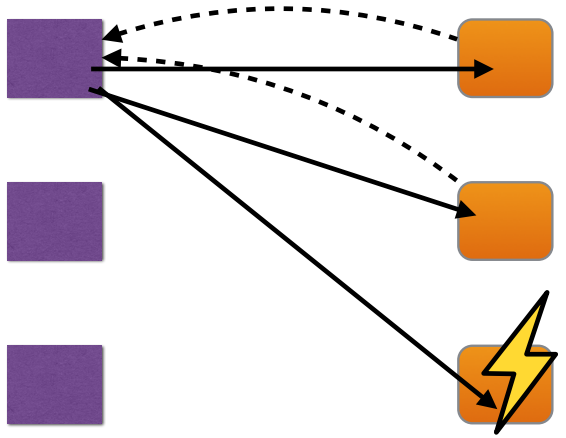


Dependency graph

Characteristics of a Distributed System

Independencies of failures

- Replication: multiple independent processes **enables** fault tolerance
- Distributed systems **should simplify** the life of **users**

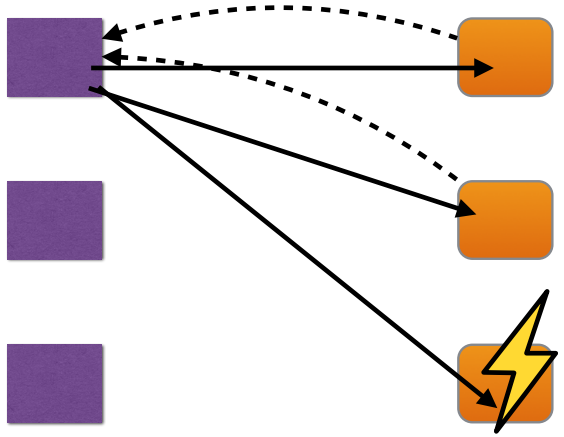


Replication can hide failures!

Characteristics of a Distributed System

Partial failures

- A subset of processes (components) can
  - fail or
  - become disconnected.
- How should remaining processes behave?
  - ★ nothing bad happens
  - ★ something useful (eventually) happens

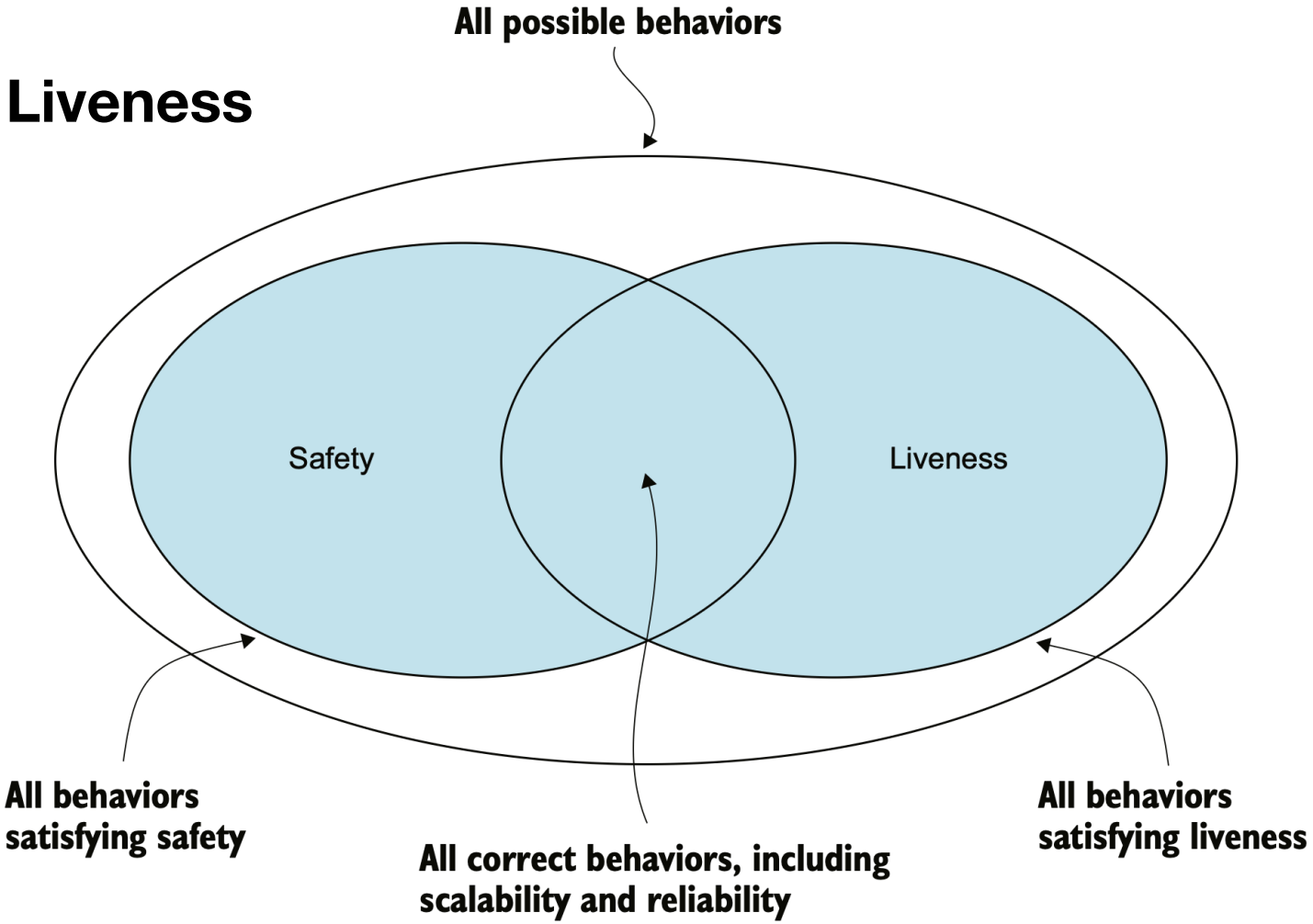


Replication can hide failures!

Correctness

- **Safety** requires that nothing bad happens during execution
- **Liveness** requires that something good eventually happens during execution
  - ✦ the program's ability to make progress
  - ✦ termination is not necessarily a requirement for liveness

Safety vs Liveness



Safety Property

Property	The bad thing
Mutual exclusion	two processes executing in a critical section simultaneously
Deadlock freedom	cycle in the wait-for-graph
Livelock freedom	something is happening but the program does not make progress
Partial correctness	terminating in a state that does not satisfy the postcondition after having been started in a state that satisfy the precondition
First come first serve	servicing a request that was made after one not yet serviced

Liveness Property

Property	The good thing
Starvation freedom	a process make progress infinitely often
Termination freedom	a program does not run forever
Guaranteed service	every request for service is satisfied eventually

Degree of Transparency

Observation: Full distribution transparency may be too much

- **Completely hiding failures** is impossible
  - Cannot distinguish a slow computer from a failed one
  - Can never be sure that a server actually performed an operation before a crash
- Full transparency will **cost performance**, exposing distribution of the system
  - Keeping caches exactly up-to-date with master copy
  - Immediately writing to disk for fault tolerance

Distributed System Architectures

Distributed Communication Network

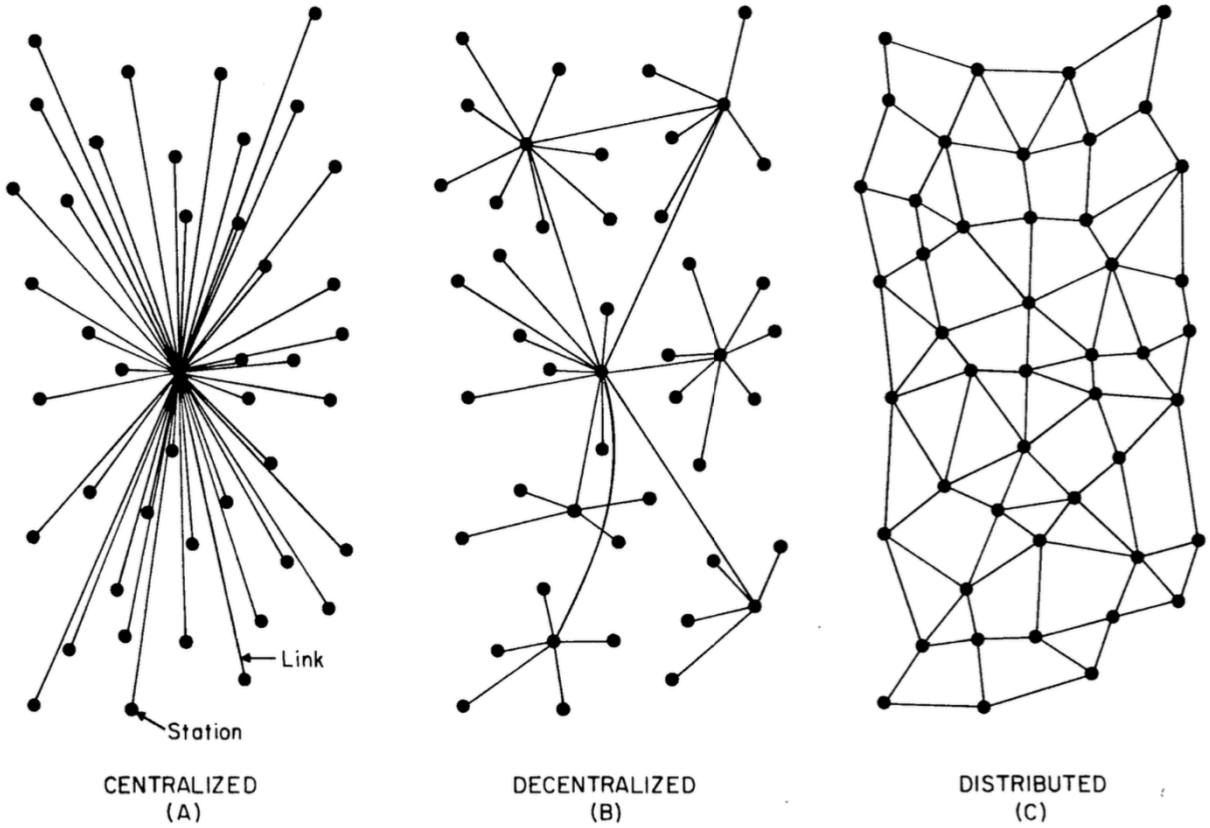


FIG. 1 — Centralized, Decentralized and Distributed Networks

Distributed System Architectures

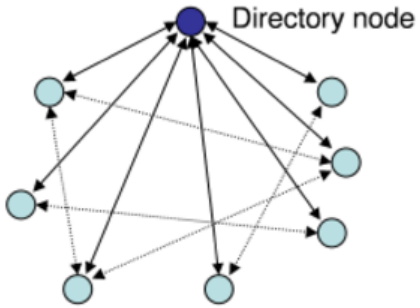
Centralized architecture

- client-server

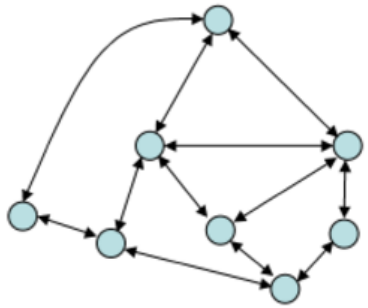
Decentralized architecture

- peer-to-peer (p2p)

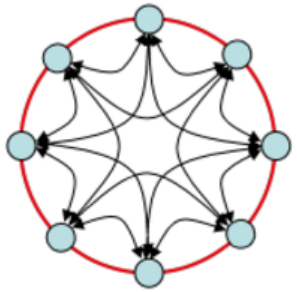
Overlay Networks



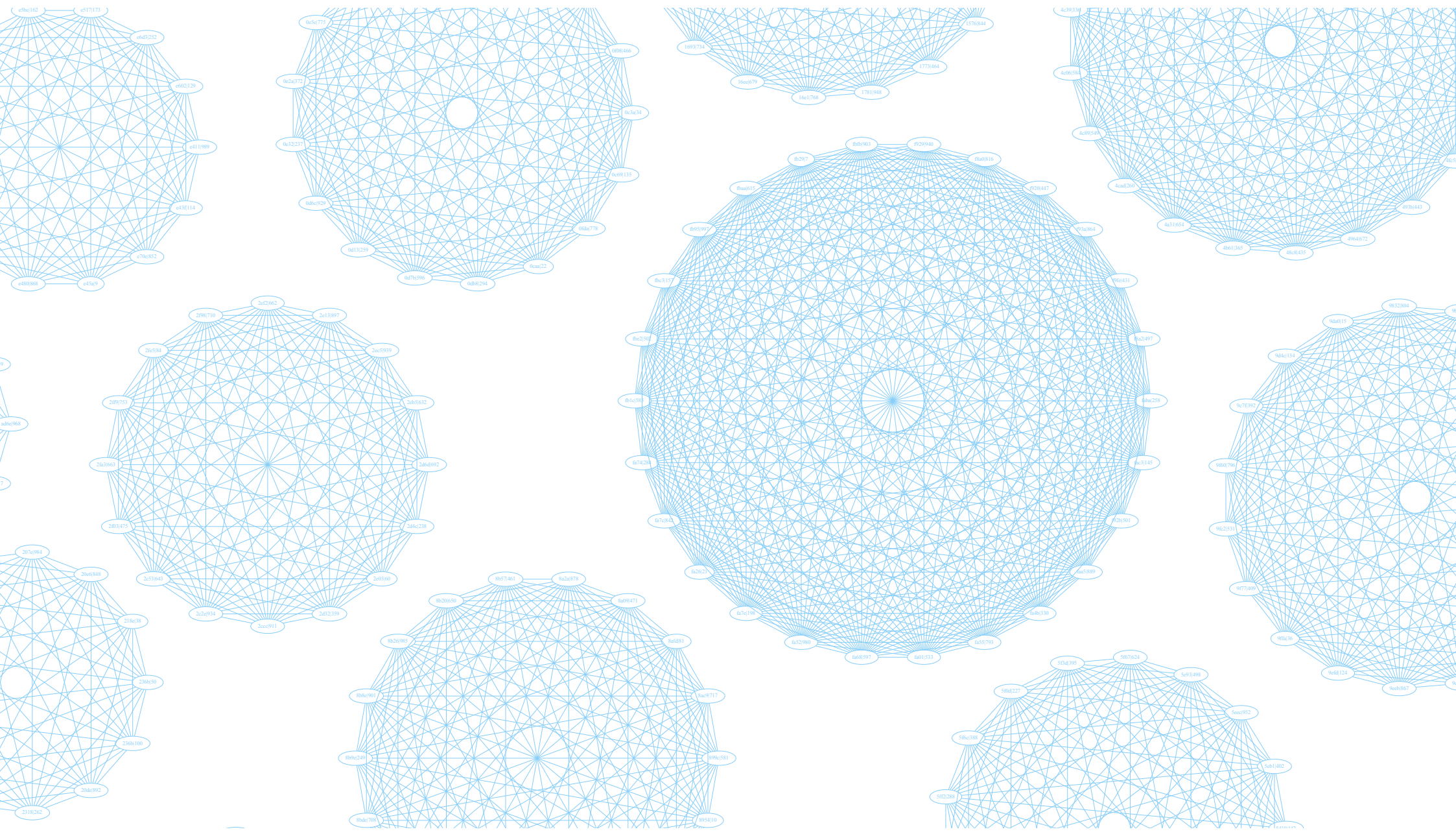
Hybrid (Napster)



Unstructured overlay



Structured overlay


e58e1162
e517f173
e6d3f252
e602f129
e41f1989
e43f1114
e70e1852
e52a59
e4803808
0e5ef775
0f0f8466
0c2aj372
0c3ef34
0c32f237
0x69f135
0d6ac778
0cata122
0dba8294
0d7b1596
0d13f259
1576f844
1693f734
1773f464
16ee1679
16e1f768
178119448
4c396334
4c06f584
4c89f548
4cadf266
4a31f654
4b613365
48c8f435
4993b443
4964b672
46c5
f93af864
f996f431
fba21497
fdbac258
fac3f145
f62fc501
fada5889
fa48e330
fa55f793
fa01f833
fa68f597
fa52f980
fa7ef198
fa7c9f42
fa74f284
fbc3f157
fbe2f502
fb1cf583
fbf9f903
f929f940
fb297
f8a0f816
f920f447
fb65f997
9832f804
9dfa9f15
9d4ef154
9c7ff392
96bdf796
96c25331
9677f489
9e6f124
9eeb8667
98fac36
2ef2f662
2f98f710
2e13f897
2fe5f84
2ec5f939
2d95f753
2db5f632
2fa3f663
2d6df692
2f03f475
2d4ef238
2c53f643
2e03f60
2c2ef934
2d32f359
2ecef911
207ef984
20ef6849
218ef38
230fb50
236fb100
20def892
2318f262
8b57f461
8a2af878
8b20f650
8af09f471
8b26f985
8afdf81
8b8ef901
8ac9f717
8b9f2349
899f5f81
8bdef708
8954f410
5f3df395
59f67f24
5ef93f498
5f0df227
5eecf952
59fcf388
5efb14402
5ff2f288

Decentralized Architecture

Characteristics of Decentralized Algorithms

- No machine has complete information about the system state
- Machines make decisions based only on local information
- Failure of one machine does not ruin the algorithm
- There is no implicit assumption that a global clock exists

Decentralized Architecture

Characteristics of Decentralized Algorithms

Pitfalls of Distributed System

Pitfalls when Developing Distributed Systems

Observation: Needless complexity due to false assumptions made by developers

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator

Questions?