THE UNIVERSITY OF CHICAGO

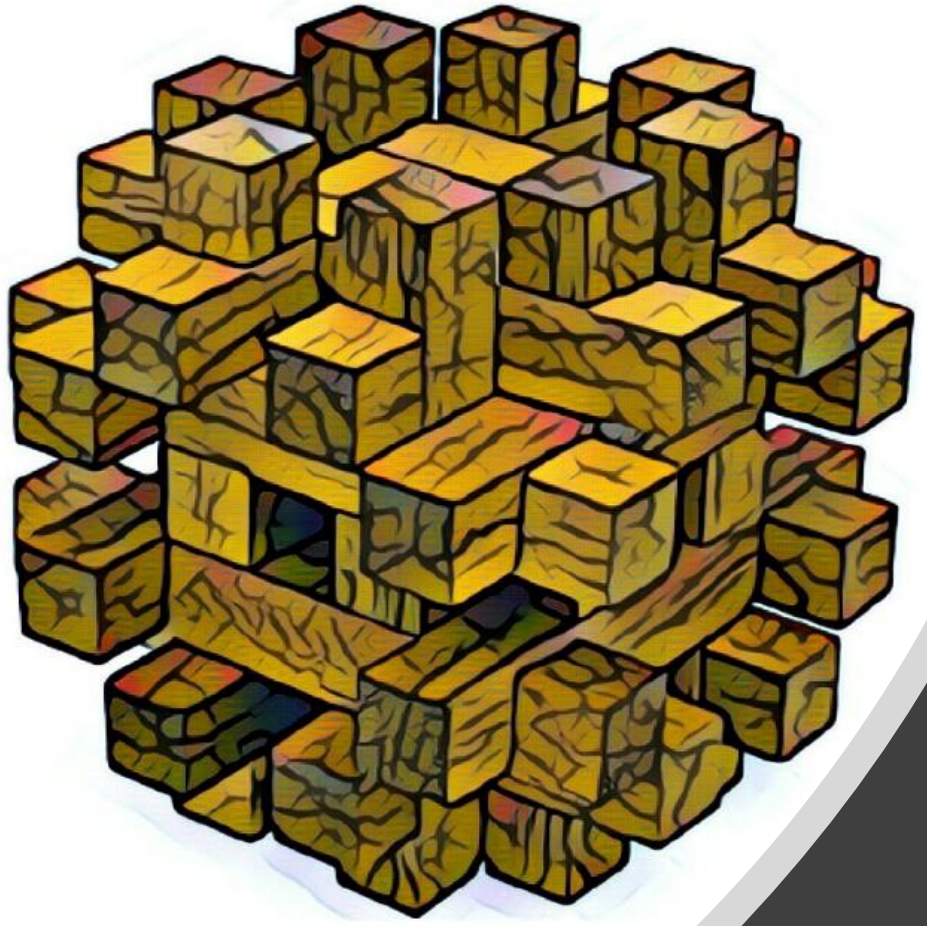
MPCS 52040
1. Introduction to Distributed Systems

Kyle Chard

chard@uchicago.edu

Outline

- Part 1: What is a distributed system?
  - How is it different than a non-distributed system?
- Part 2: What are some of the challenges with distributed systems?
- Part 3: What are some examples of distributed systems?



Textbook Chapter 1

What is a distributed system?

Tanenbaum:

- … a collection of autonomous computing elements that appears to its users as a single coherent system

Colouris:

- … is one in which components located at networked computers communicate and coordinate their actions only by passing messages

Schroeder:

- … is several computers doing something together. Thus, a distributed system has three primary characteristics: multiple computers, interconnections, and shared state.

Wikipedia:

- … is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another

1) More than one computing element

- May contain few to many computing elements (called a *node)*
  - A node may be a personal computer, laptop, HPC node, or small device (e.g., embedded computers, sensors, etc.)

1) More than one computing element

- May contain few to many computing elements (called a *node)*
  - A node may be a personal computer, laptop, HPC node, or small device (e.g., embedded computers, sensors, etc.)

- Implication: nodes must communicate with one another
  - Typically via messages where node reacts to incoming message, processes, and then communicates with other nodes via message

2) Independent computing elements

- Each node is independent from the others (autonomous)
  - Each node may therefore operate **concurrently** with respect to the others

2) Independent computing elements

- Each node is independent from the others (autonomous)
  - Each node may therefore operate **concurrently** with respect to the others
- Implication: how do we manage independent elements? How do we even track which nodes are part of the distributed system
  - How do you know that you're communicating with an authorized member?

3) No global clock

- Each node has its own notion of time
  - May be different, run at different rates, unpredictable communication latency

3) No global clock

- Each node has its own notion of time
  - May be different, run at different rates, unpredictable communication latency
- Implication: no common reference for time
  - How do we agree on *anything* if we can't agree on time?
  - How do we know what happened first?

=> One of the most fundamental problems in distributed systems

4) Independent failures

- Independent nodes == independent failures
- Implication: If one node fails what happens to the rest of the system?
  - Ideally it should not result in system failure
  - Challenge: how do we tell if a node failed? Or if it was just slow to reply?
- Hiding failures (and recovery) is challenging
  - Lamport: a distributed system is “.. one in which the failure of a computer you didn’t even know existed can render your own computer unusable”

5) Single coherent system

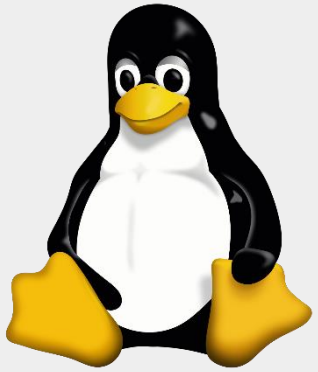
- Processes, data, and control may be distributed across a network
- Ideally we would like to make it look like a single system… but this is hard

- Implication: We have to cut corners…
  - *Appearing* to be coherent. That is, the system behaves according to the expectations of its users
  - Strive for distributed transparency:
    - User shouldn't be able to tell where a process executes or where data is stored
  - Partial failures: at any time part of the system may fail
    - Hiding partial failures/recovery is challenging

How do we manage a single computer?

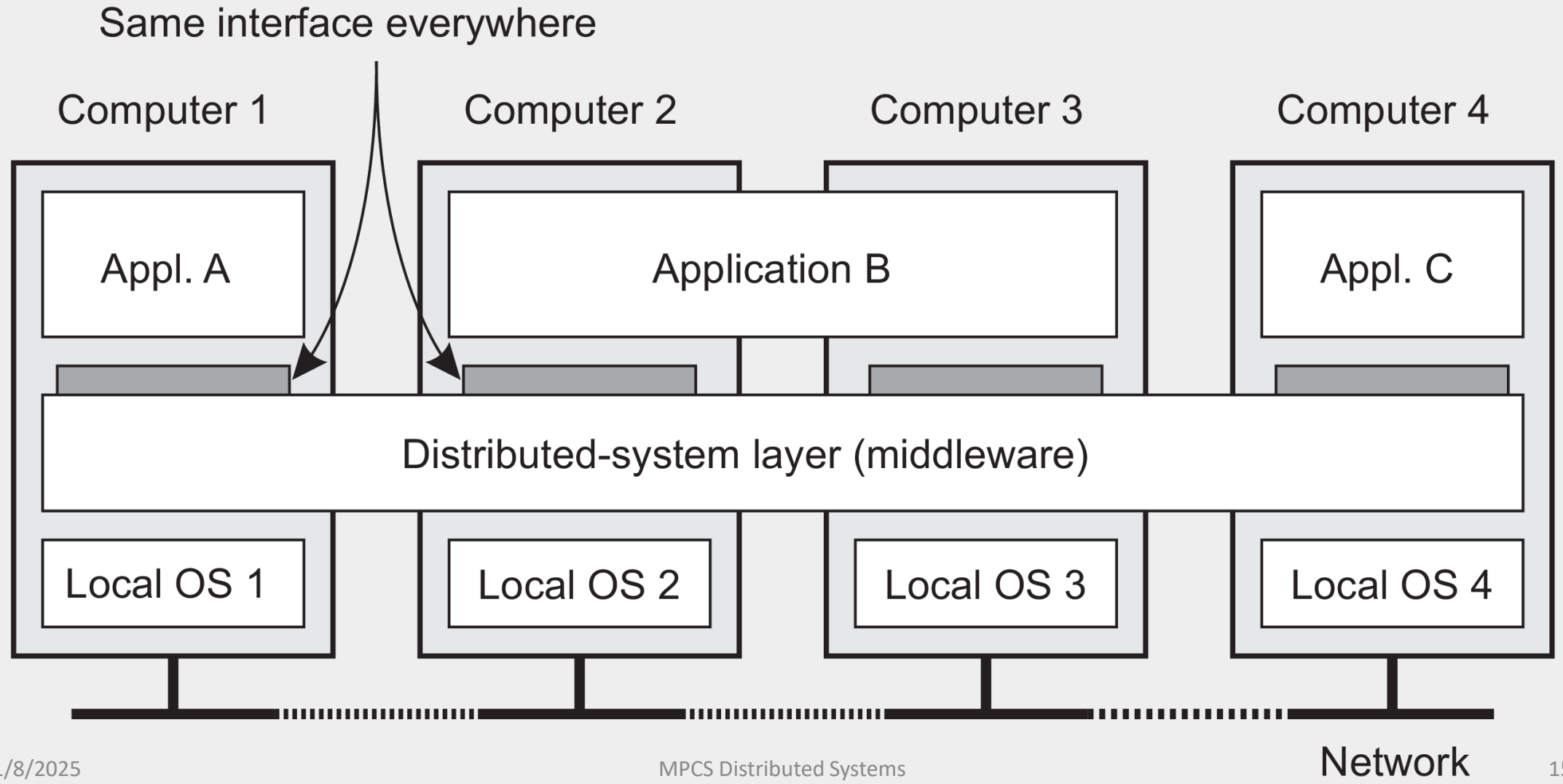
Wikipedia:

- Operating system is a system software that manages computer hardware and software resources and provides common services for computer programs



- What is the operating system for a distributed system?

Middleware: a distributed operating system



Summary

- Characteristics of a distributed system
  1. More than one node
  2. Independent nodes
  3. No global clock
  4. Independent failures
  5. A single coherent system
- We manage a distributed system using middleware
  - Middleware provides services that are necessary for the applications that are hosted on a distributed system

Part 2: Design goals for building a distributed system

Design goals for building a distributed system

How do we decide if it is worthwhile to build a distributed system?

A distributed system should:

1. Make (remote) resources accessible (sharing)
2. Hide that resources are distributed (transparency)
3. Be open such that components can be easily used by others (openness)
4. Scale to increasing number of users, resources, locations, … (scalable)

1) Resource sharing

Make it easy for an application to access and share remote resources

Resource: peripherals, storage facilities, data, files, services, and networks

- Why share?
  - Economics: cheaper to share than own resources
  - Collaboration: centered around a common view of resources
  - Necessary to achieve a goal: sending files and resources to others, accessing an instrument

- Examples:
  - Cloud storage
  - P2P networks
  - Outsourced mail services

2) Distribution transparency

Hide the fact that processes and resources are physically distributed across multiple computers (possibly by large distances)

Transparency	Description
Access	Hide differences (e.g., architecture, OS) in data representation and how an object (e.g., process, resource, data) is accessed
Location	Hide where an object is located
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move to another location (e.g., phone moves)
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by sever independent users
Failure	Hide the failure and recovery of an object

Degree of distribution transparency

Aiming for full distribution transparency may be too much:

- There are communication latencies that cannot be hidden
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - You cannot distinguish a slow computer from a failing one
  - You can never be sure that a server performed an operation before a crash
- Full transparency will cost performance (and expose distribution of the system)
  - Keeping replicas exactly up-to-date with the master takes time
  - Making reliable requires immediately flushing write operations to disk

3) Openness (interoperability)

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

- Systems should conform to **well-defined interfaces**
- Systems should **interoperate**
- Systems should support **portability** of applications
- Systems should be easily **extensible**

4) Scalability

What do we mean by scalability of a system?

- Size scalability: Add more users and resources without loss of performance
- Geographic scalability: Users and resources may be far apart, but latency delays are not noticed
- Administrative scalability: Easily managed even if it spans many administrative organizations

→ When a system needs to scale there are different types of problems that need to be solved

Size scalability problems

- Scalability problems with centralized servers:
  - The computational capacity, limited by CPUs
  - The storage capacity, including the I/O transfer rate
  - The network between the user and the centralized service

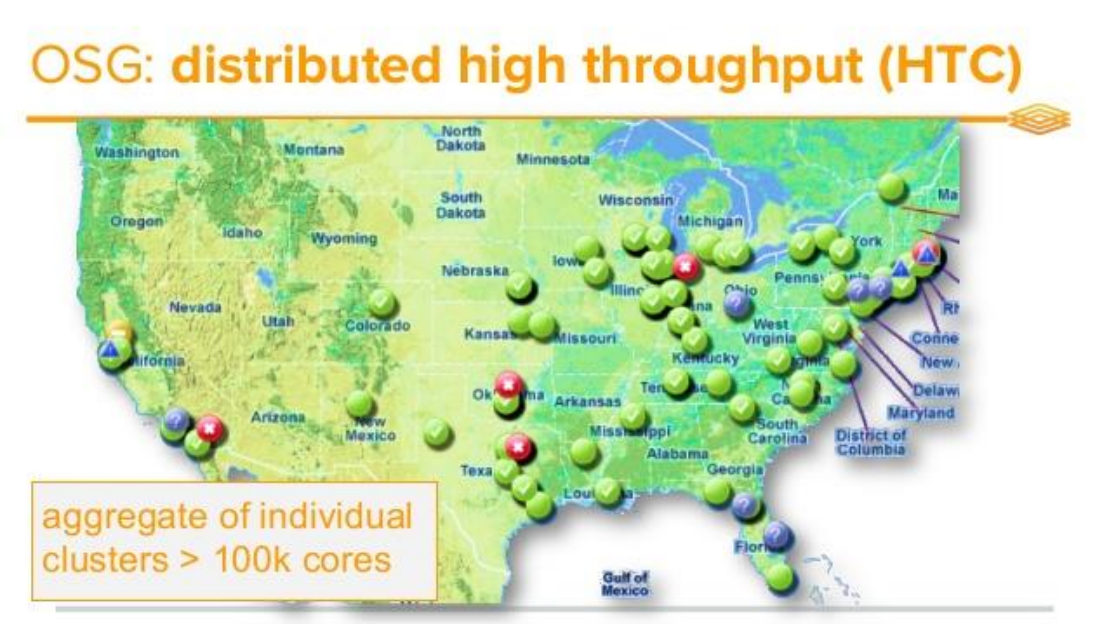
Geographical scalability problems

- Cannot simply go from LAN to WAN
  - Many LAN-based systems assume synchronous interactions: client sends request and waits for an answer
- WAN links are inherently unreliable
  - Moving streaming video from LAN to WAN is bound to fail
- Lack of multipoint communication, so that a simple search broadcast cannot be deployed
  - A solution is to develop separate naming and directory services, but they have their own scalability problems

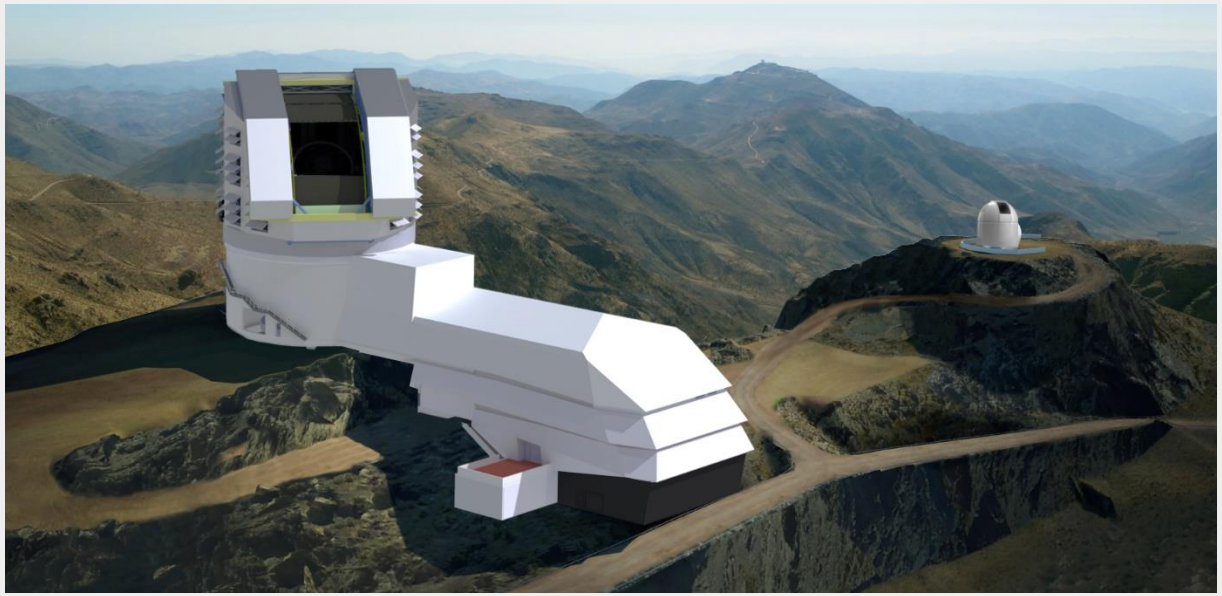
Administrative scalability problems

The most difficult, and often open, question

- A major problem: conflicting policies with respect to resource usage (and payment), management, and security.



Computational grids: share expensive resources between different domains.



Shared equipment: how to control, manage, and use a shared telescope constructed as large-scale shared sensor network?

How can we address these scaling challenges?

Scaling solutions: Scale up/out

Scalability problems in distributed systems most often appear as performance problems caused by limited capacity of servers/network

Solution:

- Scale up: expand their capacity (e.g., increase CPUs, mem)
  - +ve: Cost-effective, maximizes existing hardware, easy to implement
  - -ve: limited to size of hardware, migration to new hardware
- Scale out: expand the distributed system by deploying more nodes
  - +ve: Break free from individual capacity/performance constraints, no limits (assuming good scaling performance), can continue to scale in the future
  - -ve: hide communication latency, distributed work, state replication

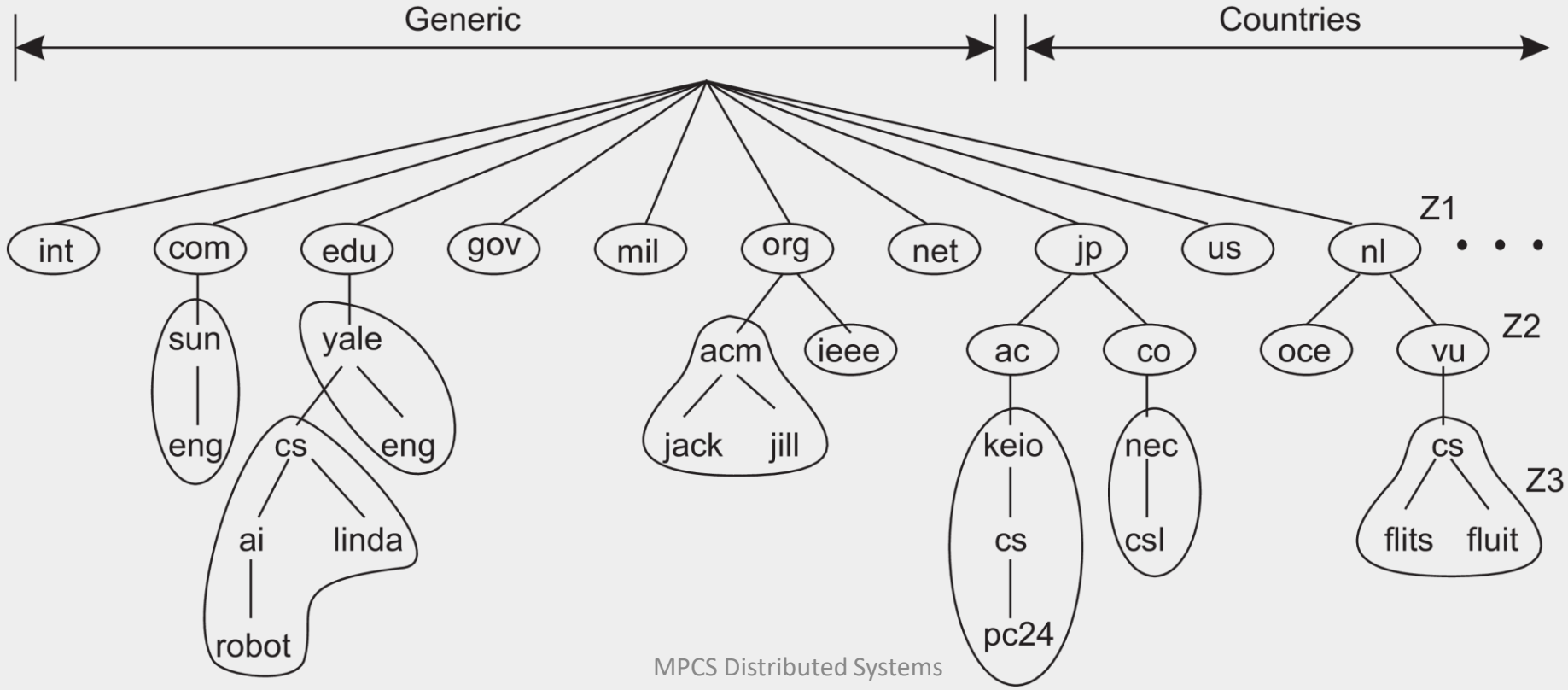
Scaling solutions: Hide communication latency

Try to avoid waiting for responses to remote-service requests as much as possible

- Make use of asynchronous communication
- Have separate handler for incoming response
- Problem: not every application fits this model (e.g., interactive applications)
  - Must reduce communication (e.g., caching, moving compute to client, etc.)

Scaling solutions: Distribute work

Take a component, split it into smaller parts, and subsequently spread those parts across the system



Scaling solutions: Replication

- Replication and caching: make copies of data available at different machines
  - Replicated file servers and databases
  - Mirrored Web sites
  - Web caches (in browsers and proxies)
  - File caching (at server and client)
- Applying replication is easy, except for one thing ..
  - Having multiple copies (cached or replicated), leads to inconsistencies:
    - Modifying one copy makes it different from the rest
    - Keeping copies consistent (in a general way) requires global synchronization
    - Global synchronization precludes large-scale solutions

The eight fallacies of distributed computing

Many distributed systems are needlessly complex caused by mistakes that required patching later. Many false assumptions are often made...

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

Peter Deutsch

Ghostscript, Smalltalk, etc.

Summary

Goals for a distributed system:

1. Sharing: make resources easily sharable and discoverable
2. Transparency: hide the fact that resources are distributed across a network
3. Openness: components that can be easily used by others
4. Scalability: Increasing number of users, resources, locations, …

Achieving these goals is challenging and there are many pitfalls

Part 3: Types of distributed systems

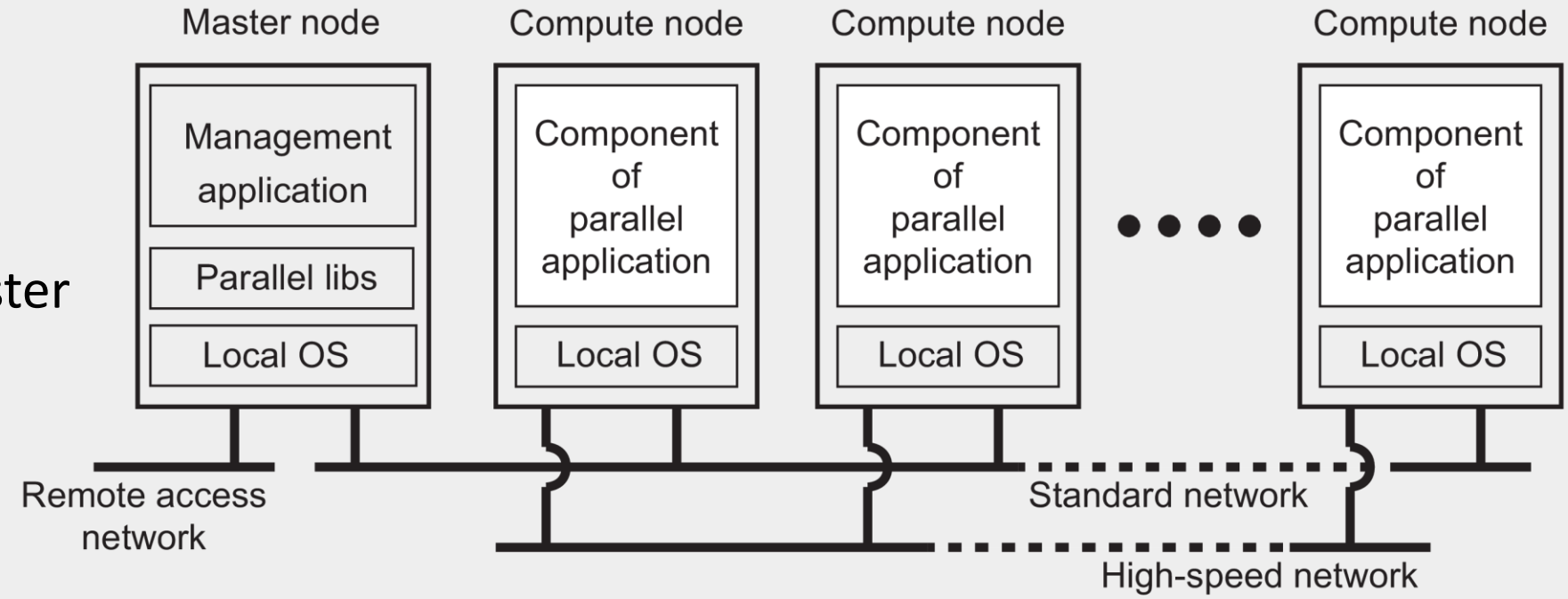
Types of distributed systems

1. **High performance distributed computing (HPC) systems**
   - **Clusters, Grids, Clouds**
2. Distributed information systems
3. Distributed systems for pervasive computing

Cluster computing

- Essentially a group of high-end systems connected through a LAN
  - Homogeneous: same OS, near-identical hardware
  - Single managing node

Linux-based Beowulf cluster



Supercomputers

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	
4	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	
5	**HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	

Grid computing

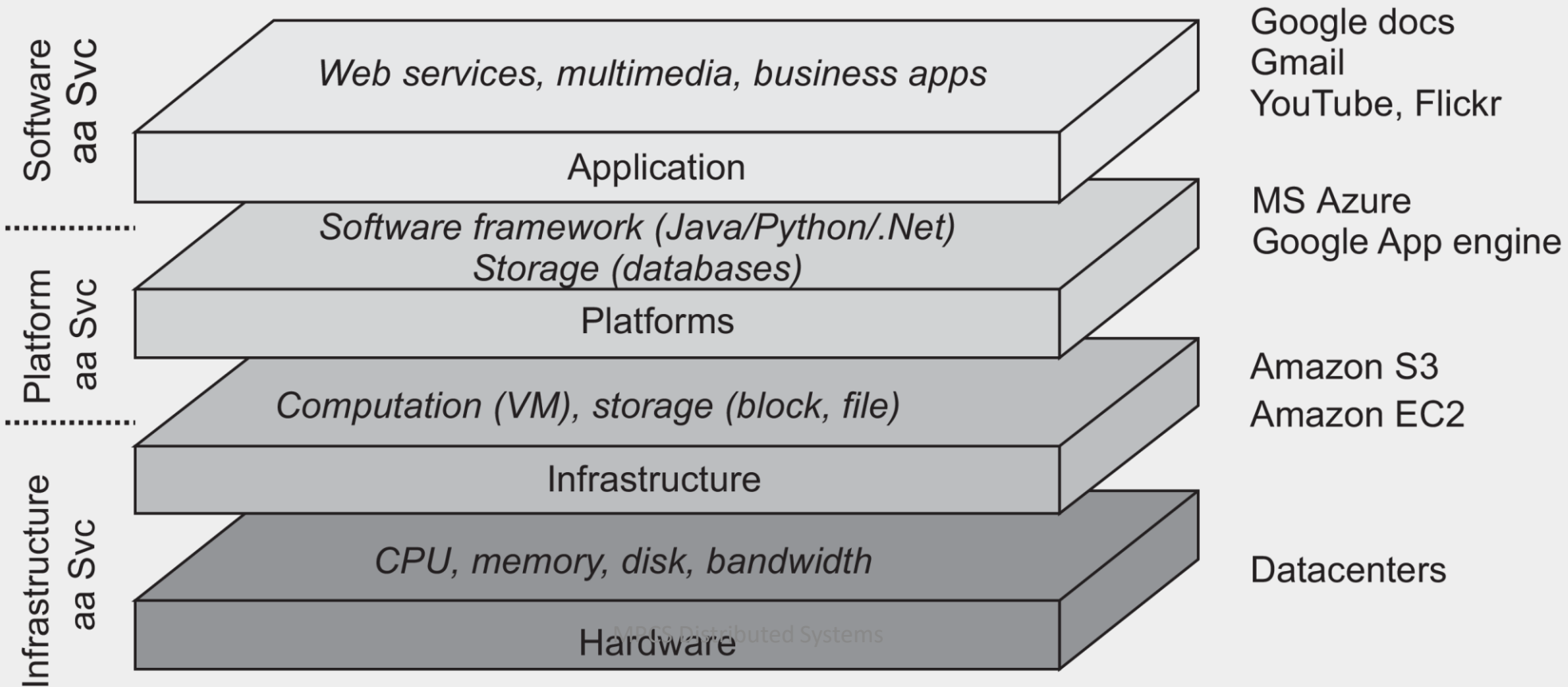
The next step: lots of nodes from everywhere

- Heterogeneous
- Dispersed across several organizations
- Span a wide-area network

- Grids generally use virtual organizations. This is a grouping of users (or better: their IDs) that will allow for authorization on resource allocation.

Cloud computing

The *next* next step is to simply outsource the entire infrastructure that is needed for applications



Types of distributed systems

1. High performance distributed computing systems
   - Clusters, Grids, Clouds
2. **Distributed information systems**
   - **Distributed transactions, enterprise application integration**
3. Distributed systems for pervasive computing

Distributed information systems

One of the main examples of distributed systems is in the management of information:

- Distributed databases with distributed transactions
- Enterprise application communication (e.g., microservices)

Distributed systems need to be able to exchange information between nodes (e.g., state)

Databases

- Transactions
  - All or nothing semantics (everything succeeds, or transaction fails)
- ACID properties
  - **Atomic**: To the outside world, the transaction happens indivisibly
  - **Consistent**: The transaction does not violate system invariants
  - **Isolated**: Concurrent transactions do not interfere with each other
  - **Durable**: Once a transaction commits, the changes are permanent

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Application communication

- Methods for allowing application components to communicate directly with one another (without going through a shared database)
- Message passing interface (MPI) enables different messaging models in a networked environment
- Remote procedure call (RPC)
  - One component can send a request to another by performing a local procedure call
    - Request is packaged and message sent to the other component
    - Result is returned the same way
- Remote method invocation (RMI)
  - RPC but operates on objects instead of functions
- Asynchronous messaging
  - RPC/RMI require both components to be running at the same time and they need to know how to talk to each other (e.g., via stubs)
  - Message oriented middleware allows components to send messages to logical contact points (e.g., a subject) following different patterns (e.g., pub/sub)

Types of distributed systems

1. High performance distributed computing systems
   - Clusters, Grids, Clouds
2. Distributed information systems
   - Distributed transactions, enterprise application integration
3. **Distributed systems for pervasive computing**
   - **Mobile computing, sensor networks**

Pervasive computing

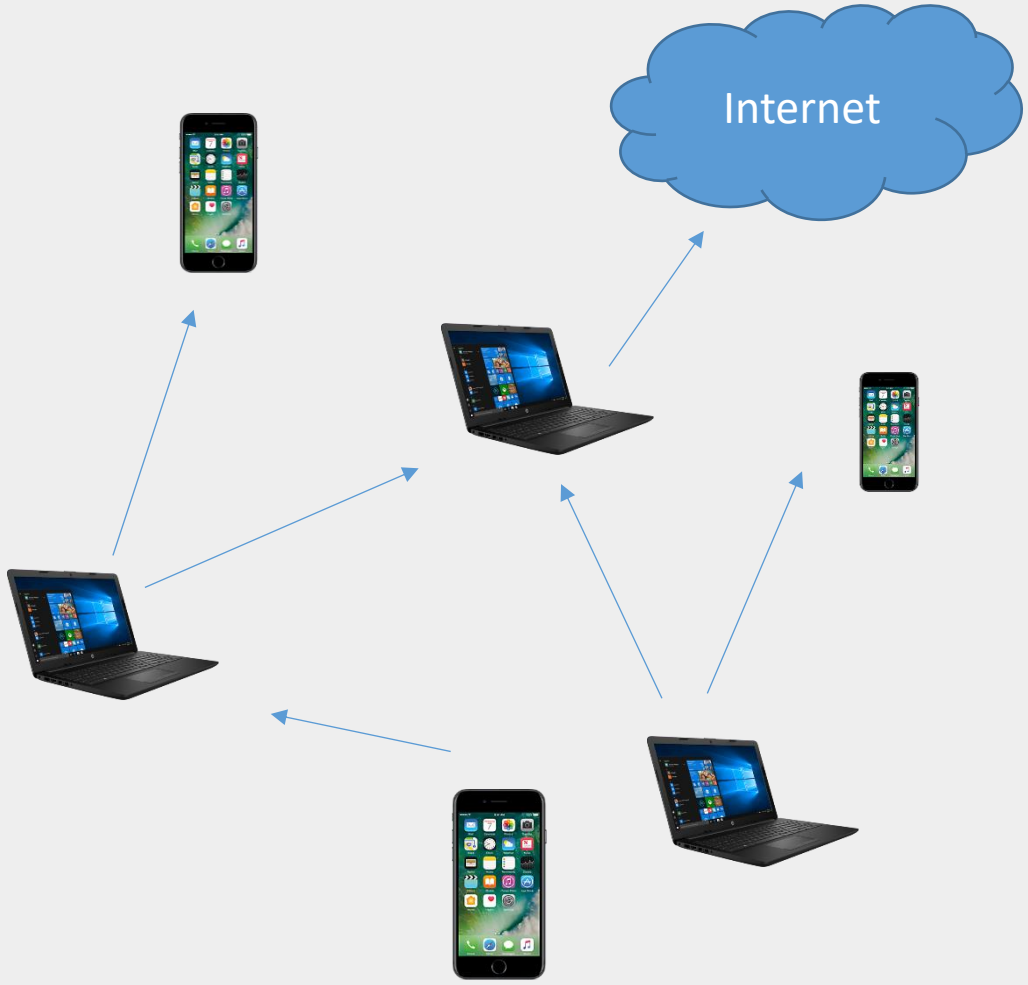
- Pervasive systems naturally blend into the environment
  - E.g., as a result of mobile and embedded systems
- Nodes are no longer permanent, they may move, fail, leave, ..
- Different challenges than the more permanent systems discussed previously:
  - Separation between user/system is blurred
  - Lack of single dedicated interface
  - Often has many sensors (pick up aspects of user behavior)
  - Actuators to provide information/feedback
- Ubiquitous computing, mobile computing, sensor networks

Example: Mobile computing

- Examples: phones, tablets, watches, cars, GPS devices, ..
- The location of a device is assumed to **change** over time
  - What does this mean?
  - How do we work out how to communicate with a device?
  - How do we discover services? How do we announce our presence to others? How do we locate other devices?

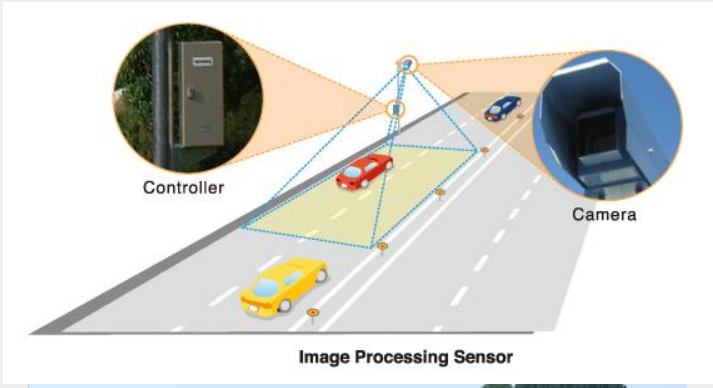
Wireless mobile ad hoc network (MANET)

- Form an ad hoc network amongst nodes
- How to route messages?
  - Pass messages along a path of nodes.. Inevitably will break as intermediate nodes will move out of neighbors' range
- Rely on two ideas
  - Flooding messages that propagate throughout the network (significant overhead)
  - Let intermediate nodes store received messages until they can be passed on (deals with disruption)



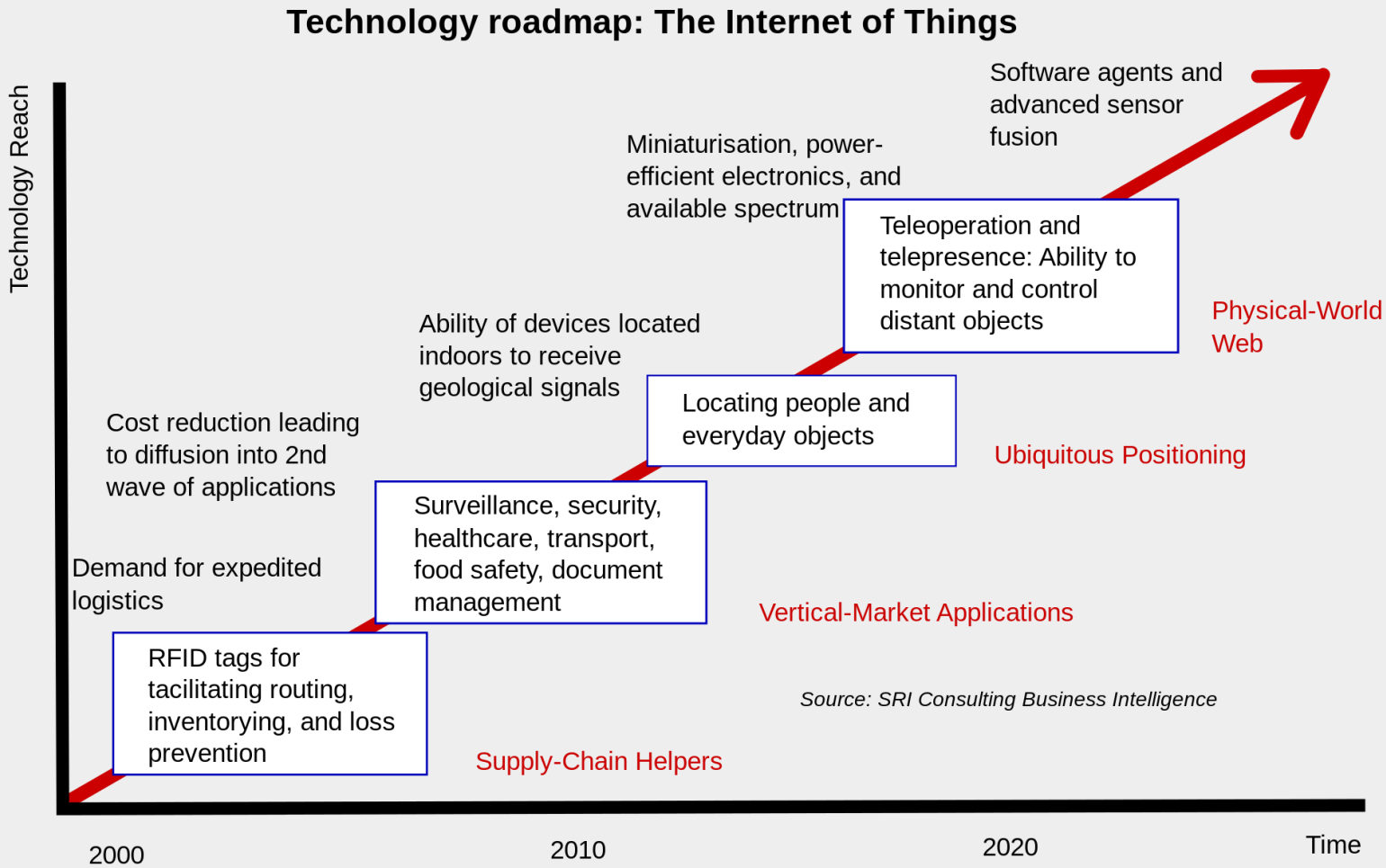
Example: Sensor networks

- Thousands of small nodes each with sensing devices
- Often wireless communication and battery powered
- Challenges include limited resources, communication, constrained power
- Scope-based communication
  - E.g., neighbors, systemwide, individual node



Example: Internet of Things (IoT)

- The Internet of things (IoT) is the network of devices such as vehicles, and home appliances that contain electronics, software, sensors, actuators, and connectivity which allows these things to connect, interact and exchange data
- E.g., Smart home, transportation systems, agriculture, urban management, healthcare



Edge computing

- Aim to bring computing closer to where it is needed
- Most often used to mean that computation is primarily conducted on distributed devices
  - Some consider anything not in a data center to be the edge
- Common in IoT and sensor networks
- Increasingly relied upon in distributed computing as data sizes increase, more powerful devices, and for other reasons such as stranded power, efficiency, etc.

Summary

- There are many classes of distributed systems including:
  - Designed for computation: e.g., HPC clusters
  - Designed for information sharing: e.g., distributed databases
  - Designed for pervasiveness: e.g., mobile computing and sensor networks

- And others we will touch on throughout the course

THE UNIVERSITY OF CHICAGO

MPCS: 52040 Distributed Systems Homework 1

Kyle Chard

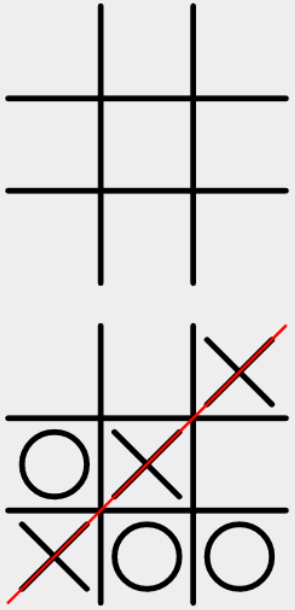
chard@uchicago.edu

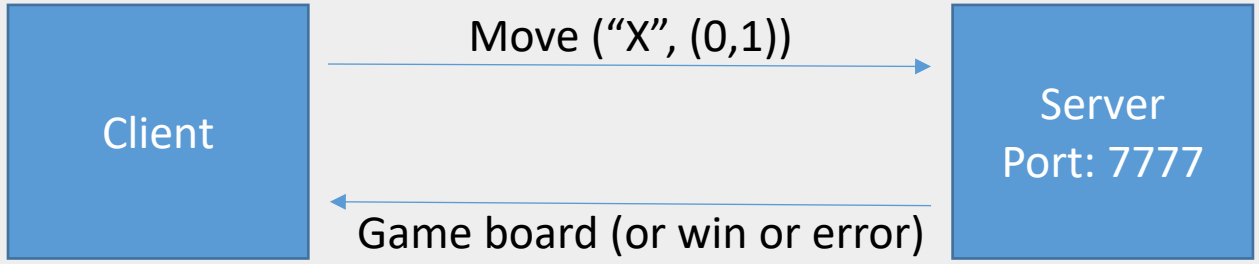
Homework 1

Aim: create a client-server Tic-Tac-Toe game

- Server maintains the game board
- Client sends moves with a player (X or O) and a position (x,y)
- Server should check the move, check if a player wins, and return an error, win message, or game board

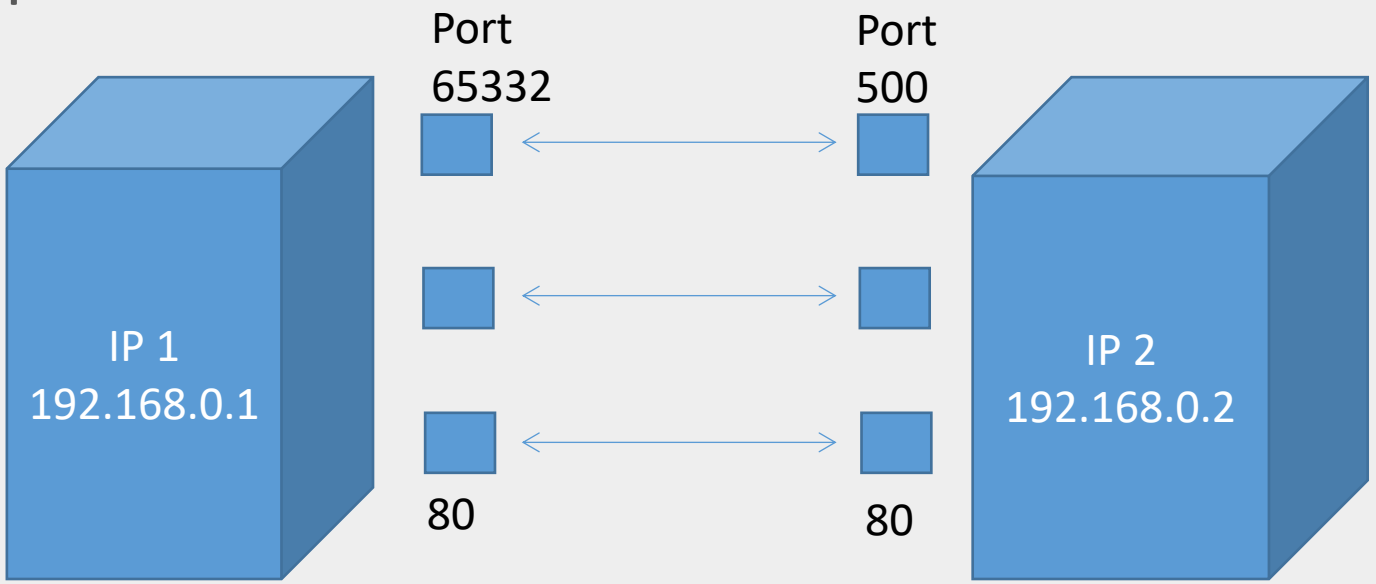
- Implementation
  - Server: single thread, single loop, single port
  - Client: calls the server over the port to make a move
  - You will need to develop the message protocol





Sockets

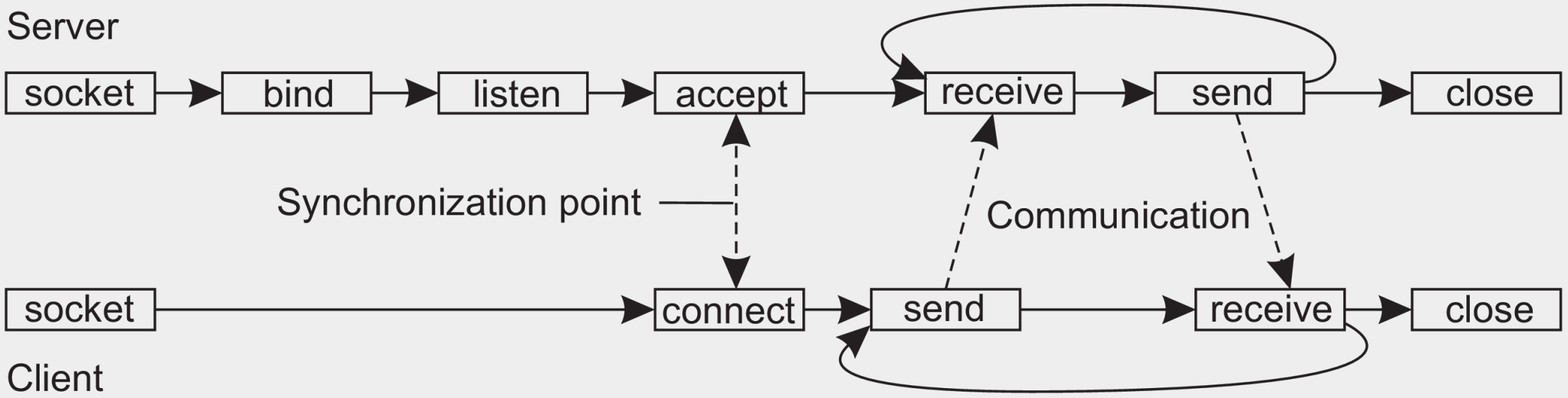
- Communication endpoint that an application can write/read data that will be sent/received from the network
- Simple abstraction over a network port for a specific transport protocol



Socket operations

- Socket: Create a new communication endpoint
- Bind: Attach a local address to a socket
- Listen: Tell OS max number of pending connection requests
- Accept: Block caller until a connection request arrives
- Connect: Actively attempt to establish a connection
- Send: Send some data over the connection
- Receive: Receive some data over the connection
- Close: Release the connection

Sockets in action



Understanding messages?

- Hosts may operate on different machines => different hardware, operating system, software, etc.
  - Little vs big endian
  - Bit representation of primitive types?
  - Encoding of complex types?
  - Nested objects?

Python Pickle

- Simple protocol for serializing/de-serializing Python object structure
  - Creates a byte stream that can then be interpreted on the other end
- Uses a compact binary representation

```
import pickle

data = {1:"A",2:"B",3:"C",4:"D",5:"E"}

picked_string = pickle.dumps(data)
```

```
import pickle

data = pickle.loads(data)
```

Submission instructions

- GitHub classroom (link on assignment)
- Clone your repo locally and add your code to that repo
- Commit during the next week
  - We will take the final commit (on the master branch) before the deadline

- Has everyone used GitHub?