Distributed Systems

CSCI-B 534/434/ ENGR E-510

Spring 2023

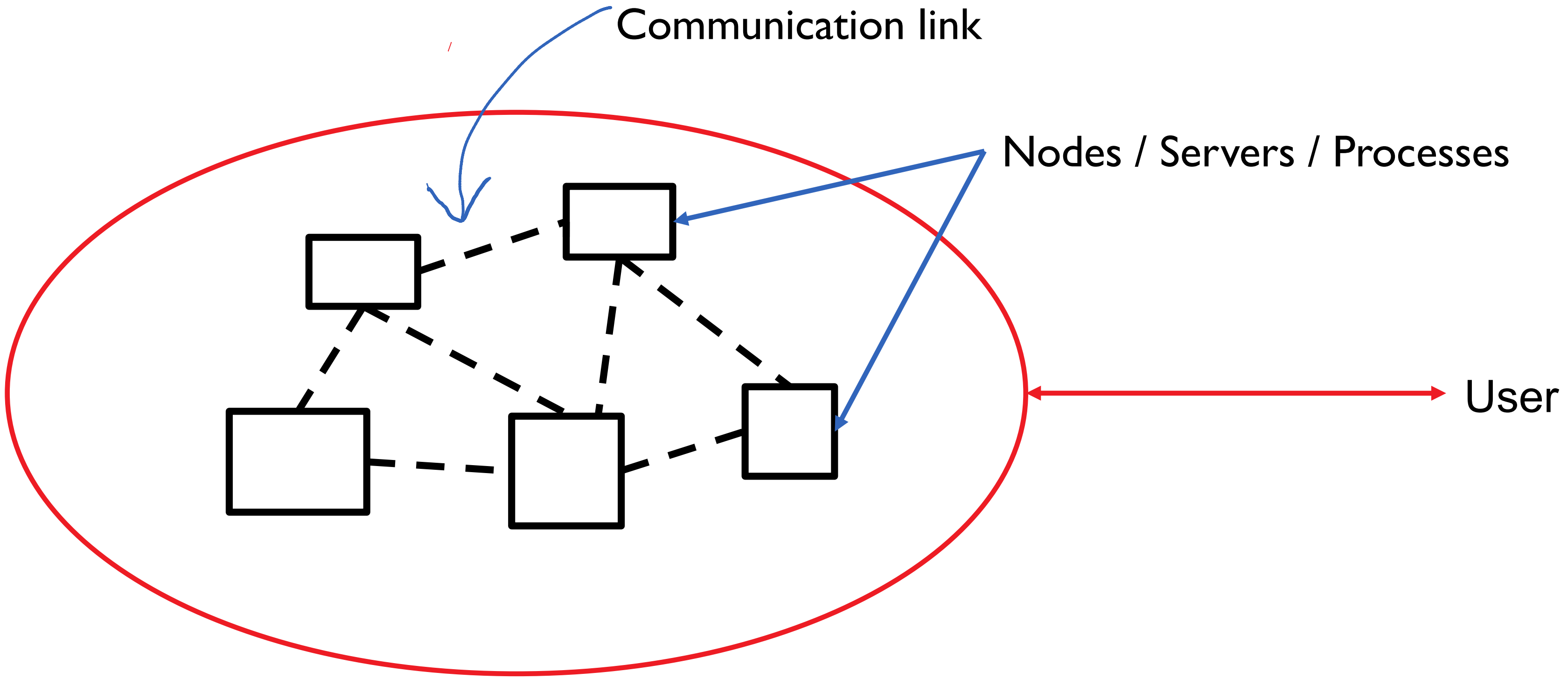
Instructor: Prateek Sharma

Welcome!

- What is a distributed system?
- Where are distributed systems found?
- Why you should take this course?
- Small taste of challenges in distributed systems
- Course contents, outline, structure, etc.

What Is A Distributed System?

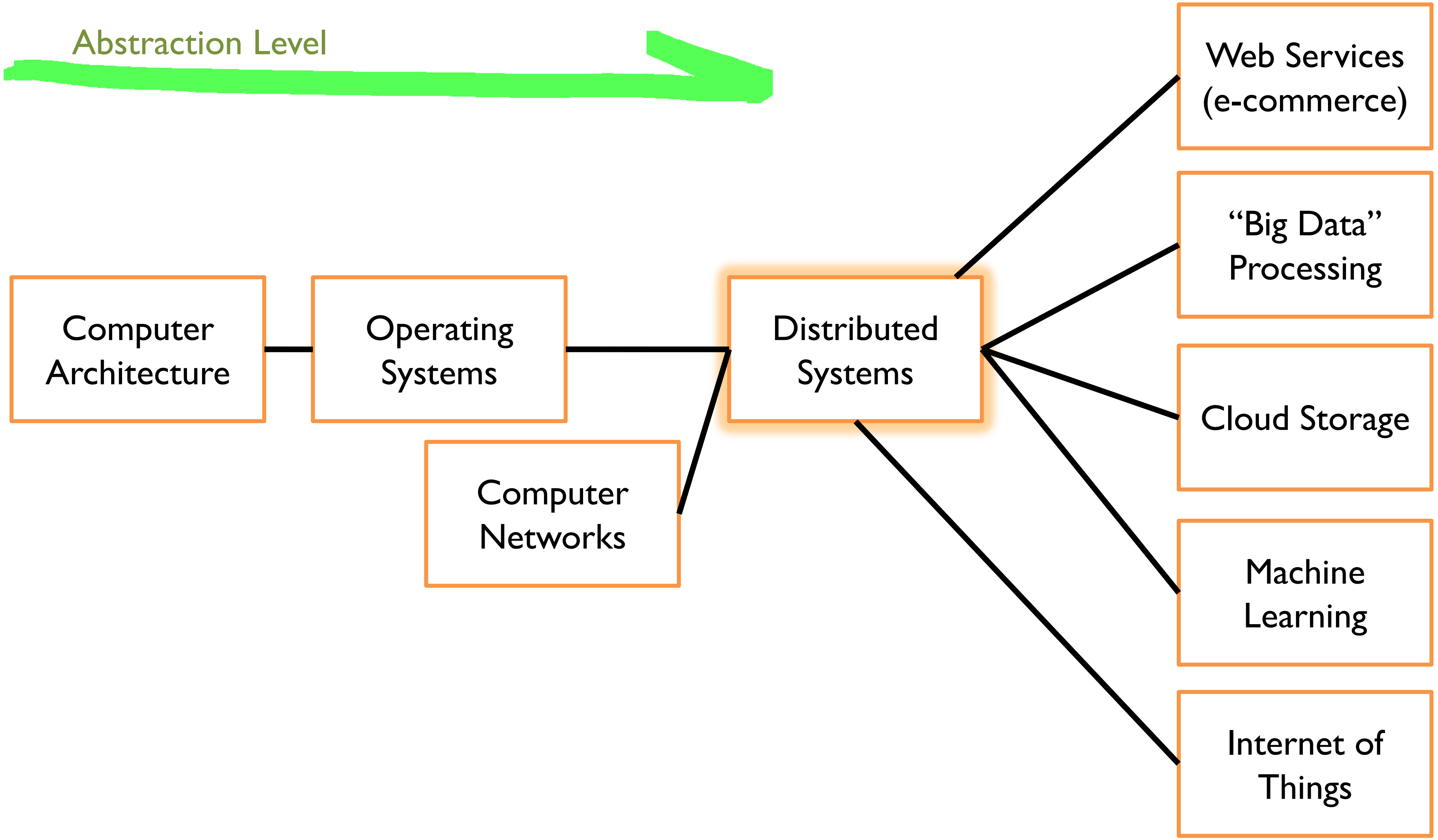
- Collection of **autonomous** computing elements that appears to its users as a **single coherent system**
- Computing elements: hardware devices or software processes
- Single coherent system: Users and applications perceive a single system



Distributed Systems Are Everywhere

- Large-scale Internet Services
  - Web clusters for high-traffic websites
- Cloud storage
  - Dropbox, Google Drive,...
- Large-scale data processing
  - Map-reduce to process TB's of data
- Graph processing
  - Social network analysis
- Large-scale machine learning
  - Model training and inference
- Sensor Networks
  - Internet of Things
- Modern multi-core architectures

Where Distributed Systems Fit In



Large-Scale Distributed Systems

- Conventionally: application deployed on a single server
- Warehouse-scale computing: meet increasing computing needs of applications
- How to handle computing, storage, and networking needs of **millions** of users?

Sports Analogy

- Soccer team comprising of multiple "autonomous" and independent players
  - There's no single "puppet master" controlling the game
- Complex coordination problem
  - Players must act based on other players' positions & intent
  - Information via visual and audio cues
  - Perfect information is not available
    - Limited field of vision, noise
- And yet we see this! (Tiki Taka)
  - https://www.youtube.com/watch?v=m1MZJeevZ6E

Course Prerequisites

- Almost all software systems are distributed systems
- This course will teach you the fundamental concepts
- Through programming exercises: build complex systems from scratch

This is a challenging course!

- *"You can have a second computer if you can show you know how to use the first one"*
  - ---Paul Barham

Course Combines Theory + Practice

- Distributed algorithms for fundamental problems
  - Understand the problem, come up with an algorithm, and prove or provide some justification about its correctness and other properties
  - You should be comfortable with showing the correctness of algorithms using induction and other techniques
    - Eg: Prove QuickSort correct using induction
- Designing, building, and testing non-trivial distributed systems
  - Proficient in "simple" user-space programming, systems programming (operating systems and networks)
- **Even "easy" problems in conventional non-distributed computing are hard or even impossible in distributed settings**

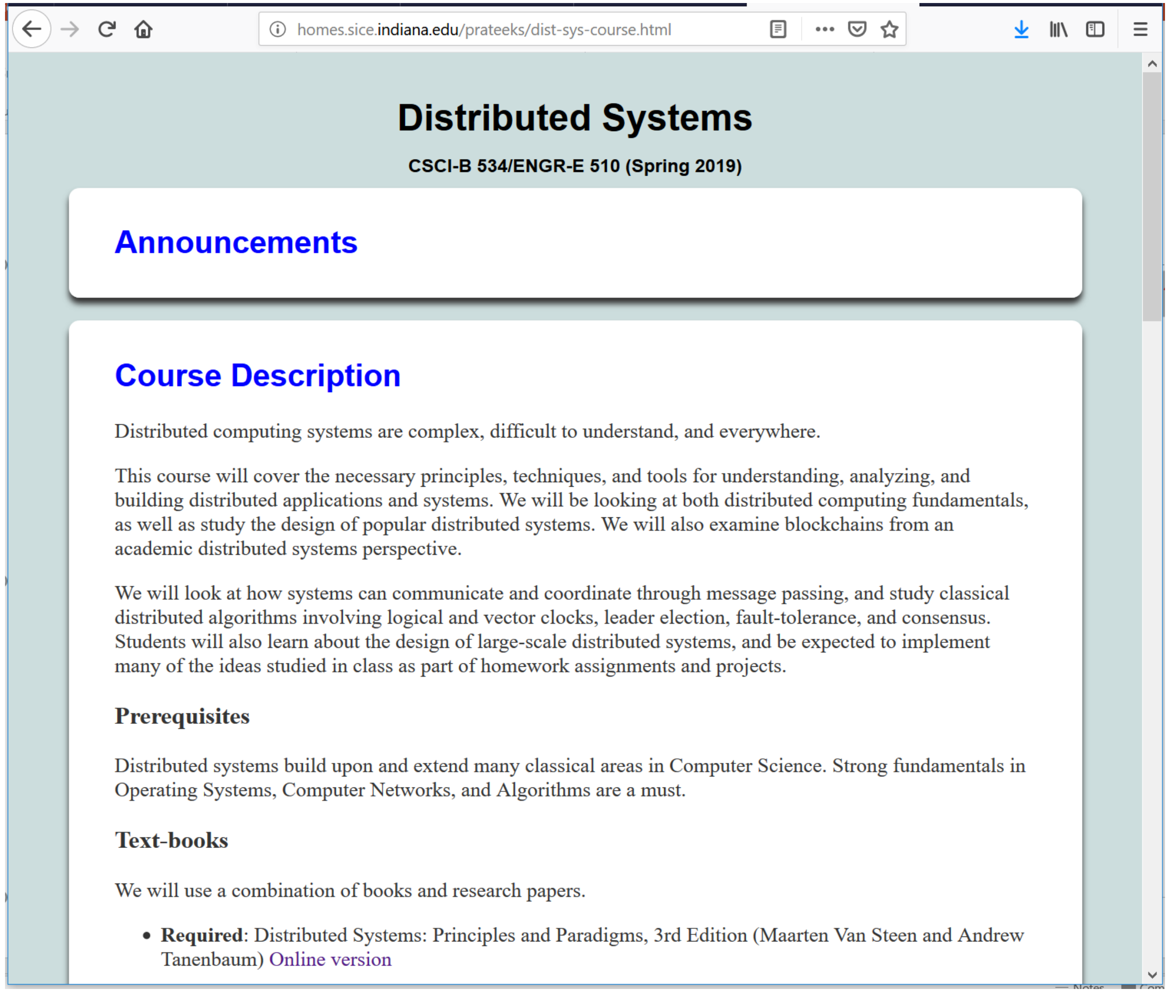
Learning Objectives

- A fundamental shift in how you think about computing: from serial programs to loosely coupled asynchronous distributed systems.
- Design and implement moderately complex distributed systems of your own
- Understand classic distributed algorithms for synchronization, consistency, fault-tolerance, etc.
- Reason about correctness of distributed algorithms, and derive your your own algorithms for special cases
- Understand how modern distributed systems are designed and engineered.

How To Succeed In This Course

Visit Course Web-page Regularly

http://homes.sice.indiana.edu/prateeks/dist-sys-course.html



homes.sice.indiana.edu/prateeks/dist-sys-course.html

Distributed Systems

CSCI-B 534/ENGR-E 510 (Spring 2019)

Announcements

Course Description

Distributed computing systems are complex, difficult to understand, and everywhere.

This course will cover the necessary principles, techniques, and tools for understanding, analyzing, and building distributed applications and systems. We will be looking at both distributed computing fundamentals, as well as study the design of popular distributed systems. We will also examine blockchains from an academic distributed systems perspective.

We will look at how systems can communicate and coordinate through message passing, and study classical distributed algorithms involving logical and vector clocks, leader election, fault-tolerance, and consensus. Students will also learn about the design of large-scale distributed systems, and be expected to implement many of the ideas studied in class as part of homework assignments and projects.

Prerequisites

Distributed systems build upon and extend many classical areas in Computer Science. Strong fundamentals in Operating Systems, Computer Networks, and Algorithms are a must.

Text-books

We will use a combination of books and research papers.

- **Required**: Distributed Systems: Principles and Paradigms, 3rd Edition (Maarten Van Steen and Andrew Tanenbaum) Online version

Attend Classes and Labs

- **Lectures**: In-person (prior recordings also on YouTube)
- Class discussions, and/or Canvas quizzes and Q&A
- **Lab-sessions:** For all assignment help.
- Check registrar website for exact details regarding your section
  - Labs will be run by TA's who will assist in:
    - Programming assignments: how to get started, some initial debugging.
    - Also serve as "office hours" and for grading all programming assignments

Read The Text Book

- Readings assigned for each lecture
  - Read **before** coming to class!
- Text-book: "Distributed Systems". Maarten van Steen and Andrew Tanenbaum
  - Soft-copy available on the web
- Many lectures will also discuss research papers
- Reference book for distributed algorithms:
  - "Elements of Distributed Computing". Vijay Garg

Some Challenges In Distributed Systems

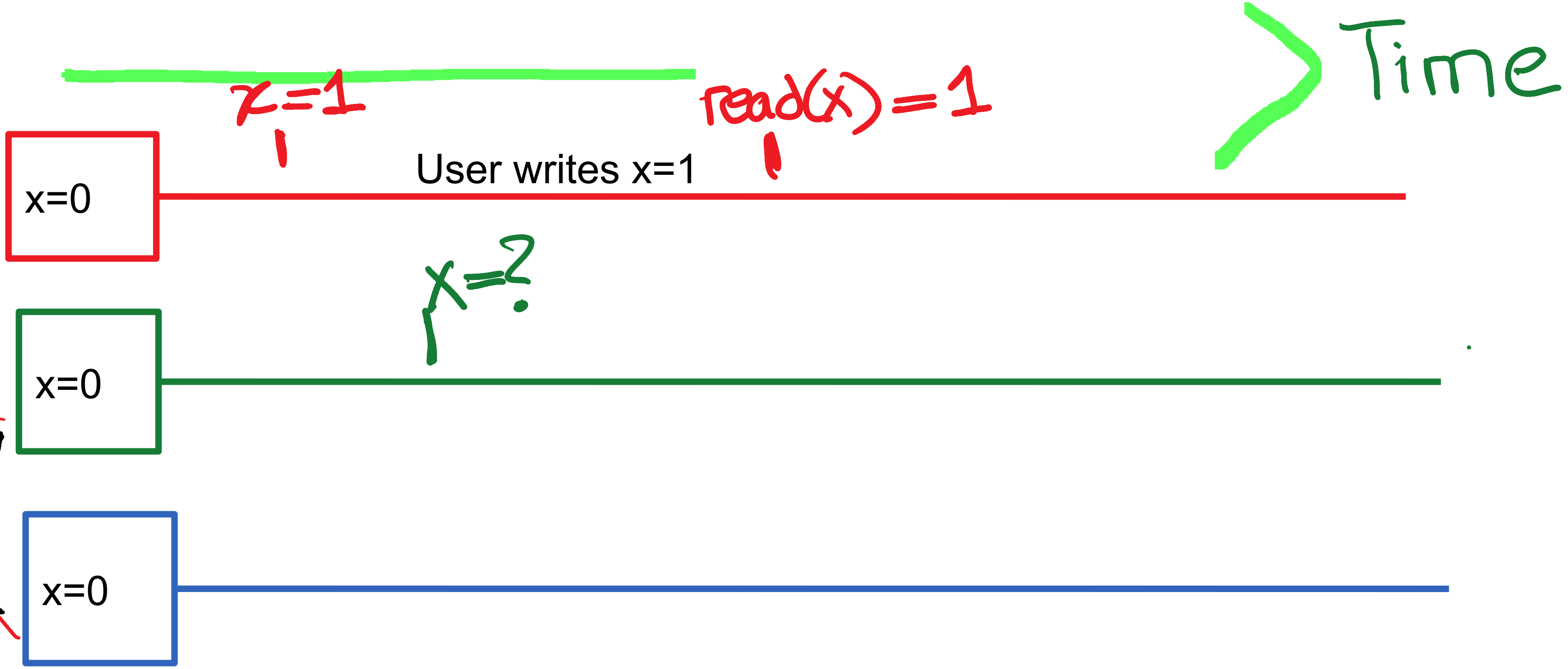
1. Distributed reads and writes
2. How to build distributed systems --- Middleware
3. Two Generals Problem

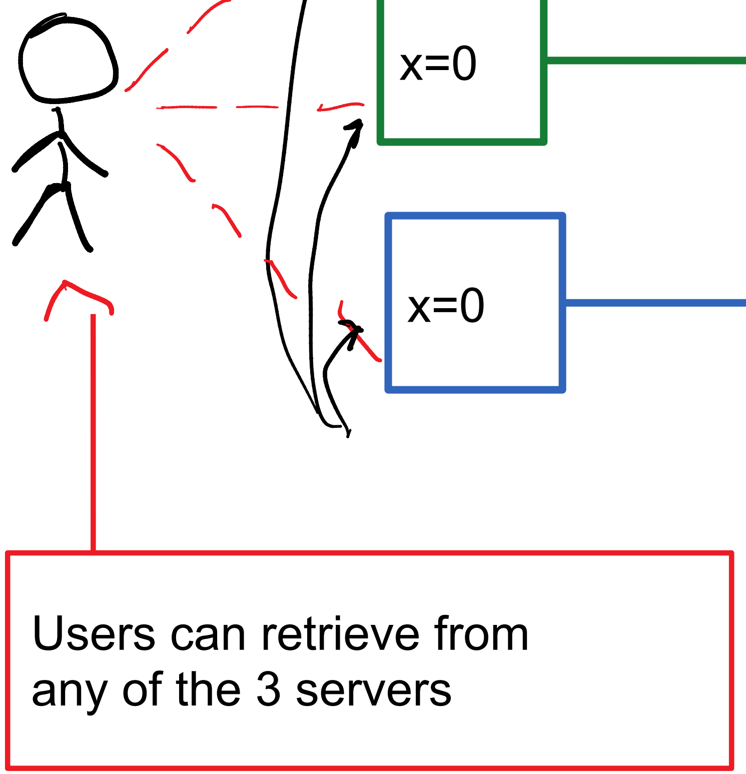
Conventional Program Semantics

```
def foo():
  X=0 ;
  X=1 ;
  print(X);
```

- Writes take effect "in order" of their issue
  - Aka "Strong consistency"

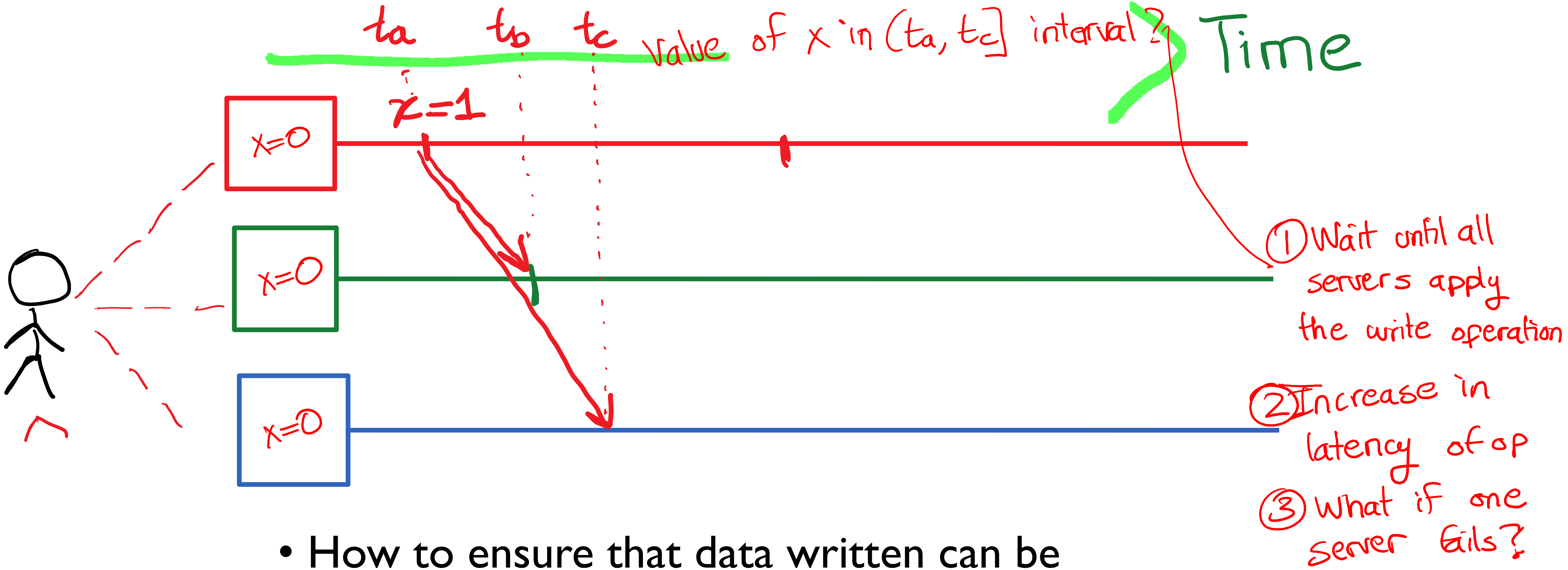
Distributed Reads and Writes





- How to ensure that data written can be retrieved from any server?

Distributed Reads and Writes



- How to ensure that data written can be retrieved from any server?
  - Replication!
  - Broadcast values after a write

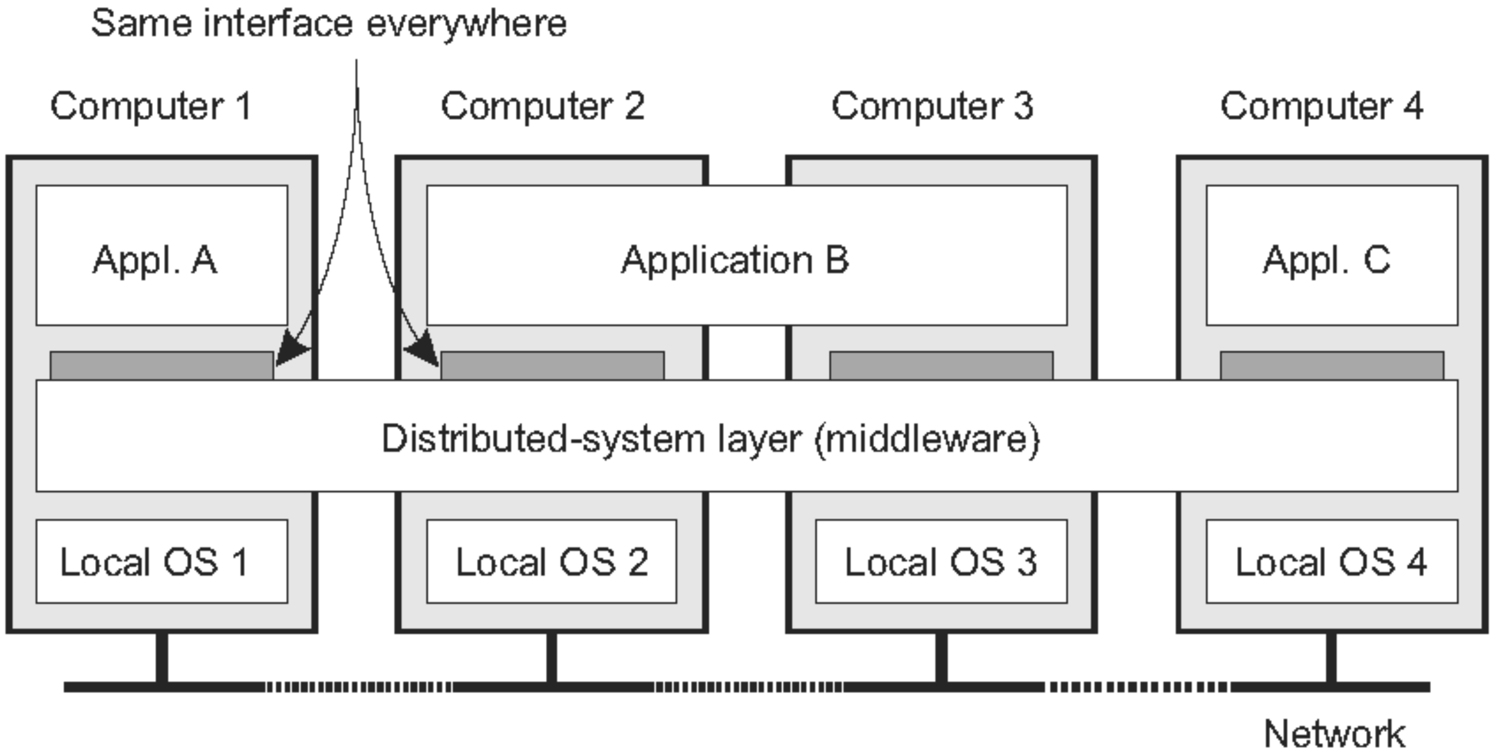
Tradeoffs In Distributed Systems

- Previous consistency example:
- Sacrifice latency for "strong consistency"
- Or, have a "looser" consistency for lower latency
- Another classic tradeoff is Cost vs. Performance:

Low Cost, High Performance (Doesn't exist)	High Cost, High Performance
Low Cost, Low Performance	High Cost, Low Performance (Undesirable)

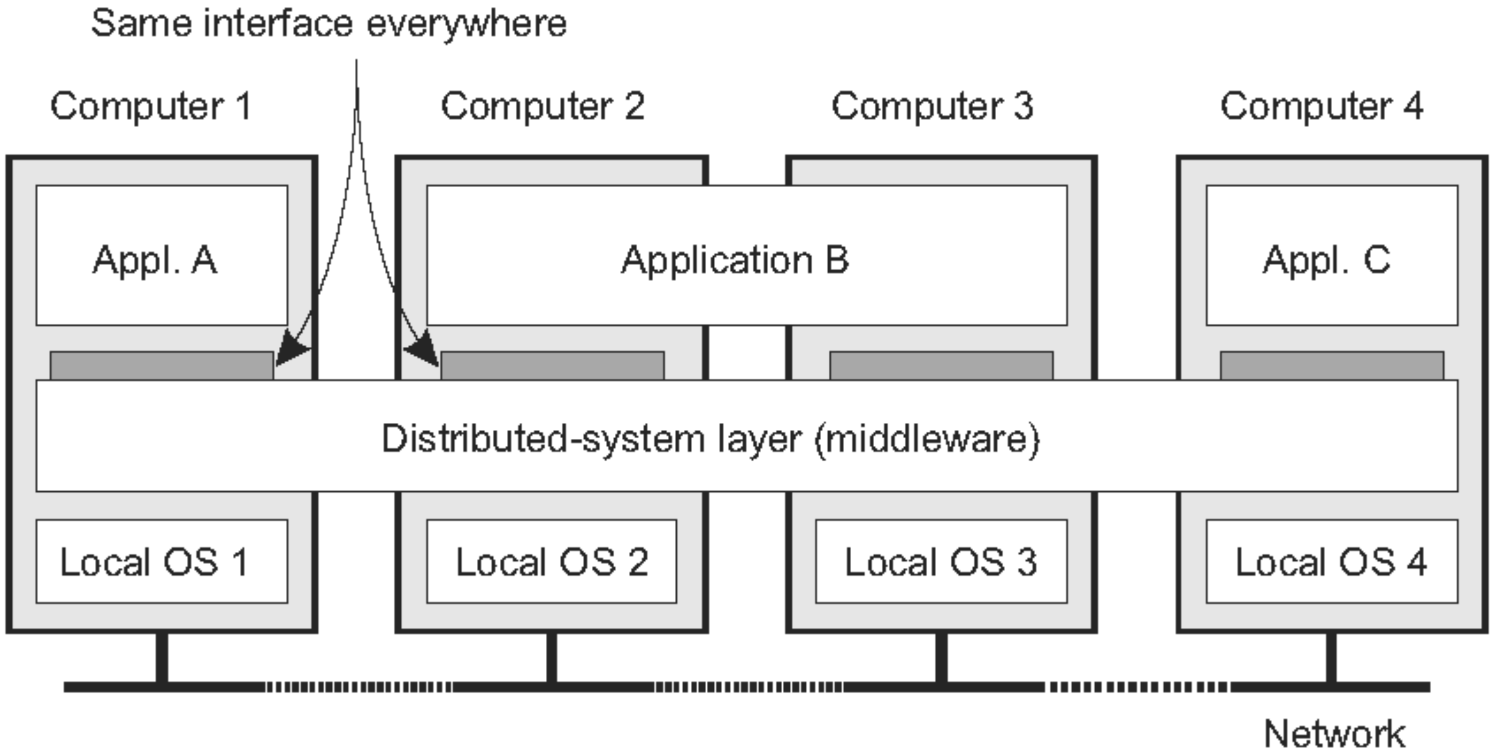
Middleware: The OS of Distributed Systems

- Commonly used components and functions for distributed applications



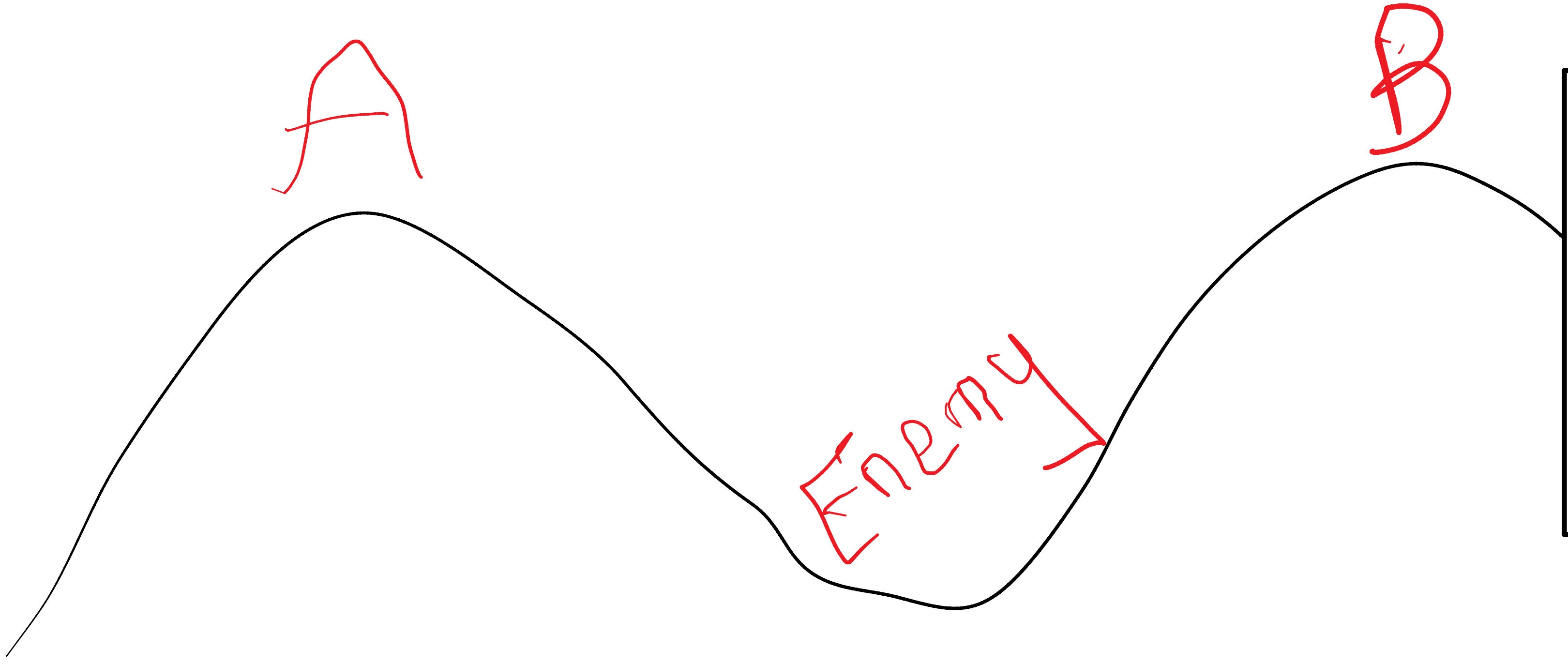
Middleware Goals

- Resource Sharing
- Distribution Transparency
- Openness (Other nodes can join)
- Scalability



Two Generals Problem

- Two Roman Generals want to co-ordinate an attack on the enemy
  - Both must attack simultaneously. Otherwise, both will lose
- Only way to communicate is via a messenger
  - But messengers can get captured/lost.
  - Perfectly-reliable communication system not available



Task: Design a protocol that ensures the two generals always attack simultaneously

Impossibility Proof

- Claim: There is no non-trivial protocol that guarantees that the two generals will always attack simultaneously
- Proof by induction on the number of messages
- Let d messages be delivered at the time of attack
- Base case: d=0. Claim holds (Impossible without any delivered messages)
- Suppose impossibility claim holds for d=n. Then, we'll show for d=n+1
- Consider message n+1
  - Sender attacks without knowing if message is delivered or not
  - Receiver must then attack too, even if msg not received
  - So the last message (n+1) was irrelevant, and n messages suffice
  - But that's a contradiction: since n+1 was supposed to be the smallest number of messages

Common Knowledge

- Solving the Two Generals Problem requires common knowledge
- Common knowledge cannot be achieved with unreliable communication channels

What if all entities shared a single global clock?

- Common knowledge possible with global clock
- Message m sent at t, delivered within delay d
- Send event becomes common knol at t+d

Next Time

- Distributed systems building blocks
    - Refresher on Operating System Processes and Threads