

# Introduction + Course Overview



جامعة الملك عبد الله  
للعلوم والتقنية  
King Abdullah University of  
Science and Technology

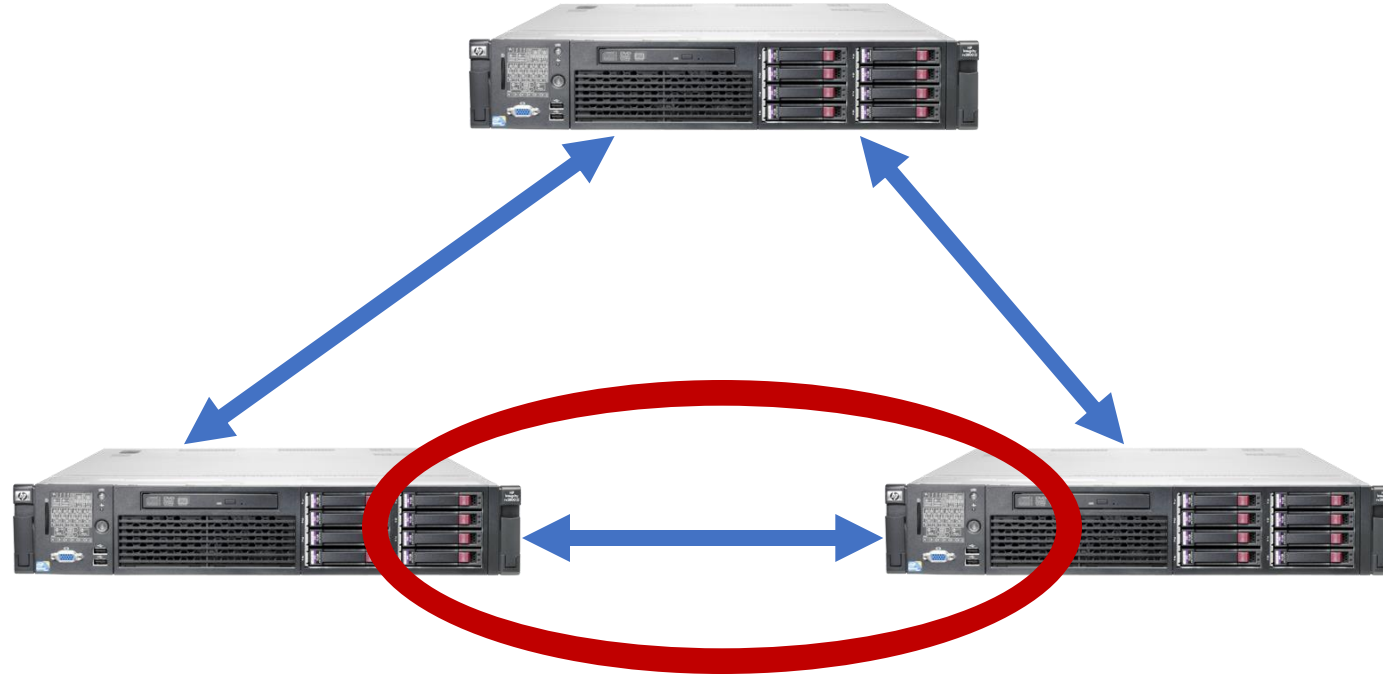
---

## CS 240: *Computing Systems and Concurrency* Lecture 1

Marco Canini

# Distributed Systems, What?

---



- 1) Multiple computers
- 2) Connected by a network
- 3) Doing something together

# Distributed Systems, Why?

---

- Or, why not 1 computer to rule them all?
- Failure
- Limited computation/storage/...
- Physical location



**Backrub (Google) 1997**



# Google 2012



**“The Cloud” is not amorphous**





# Microsoft



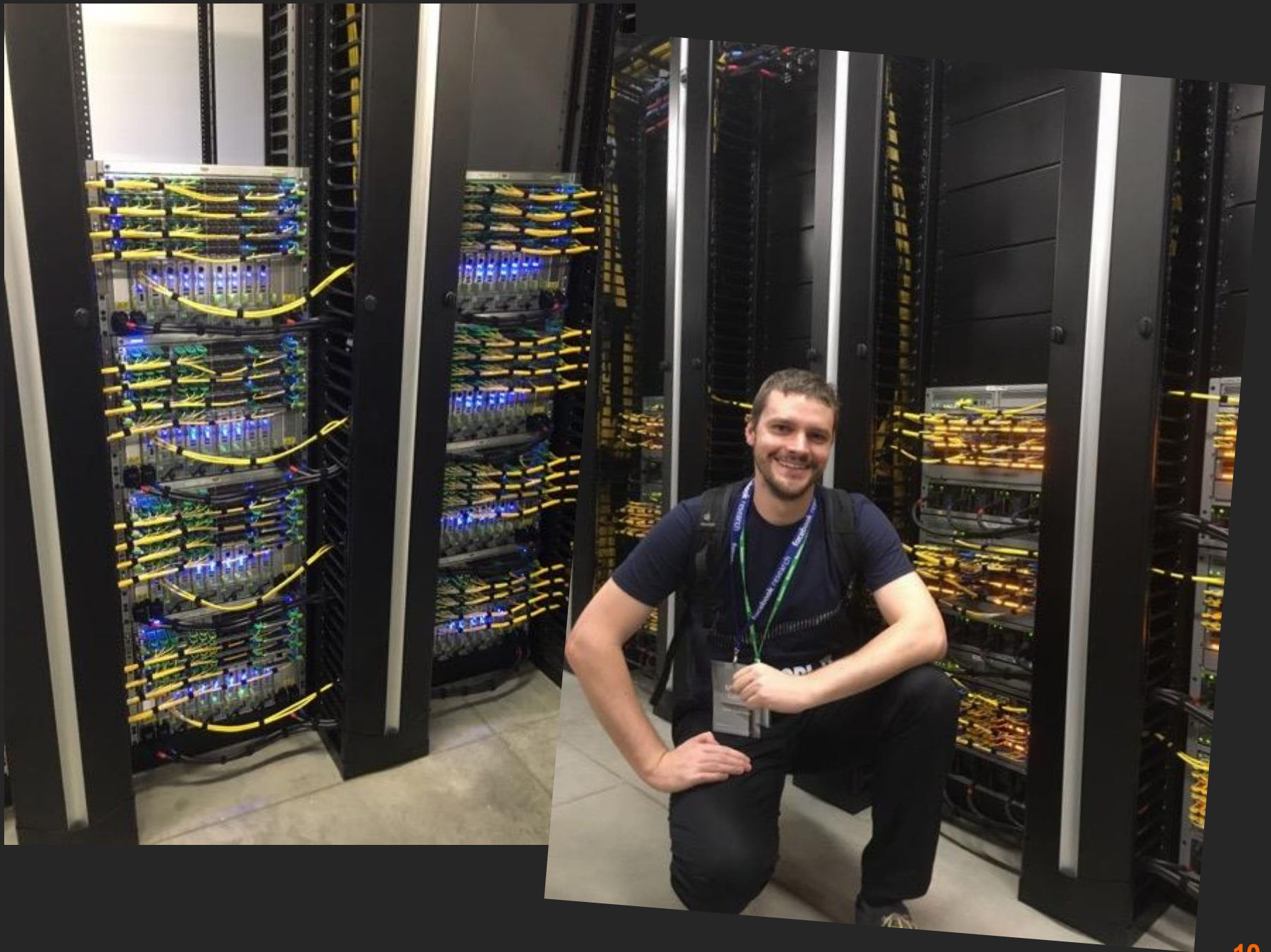
Google





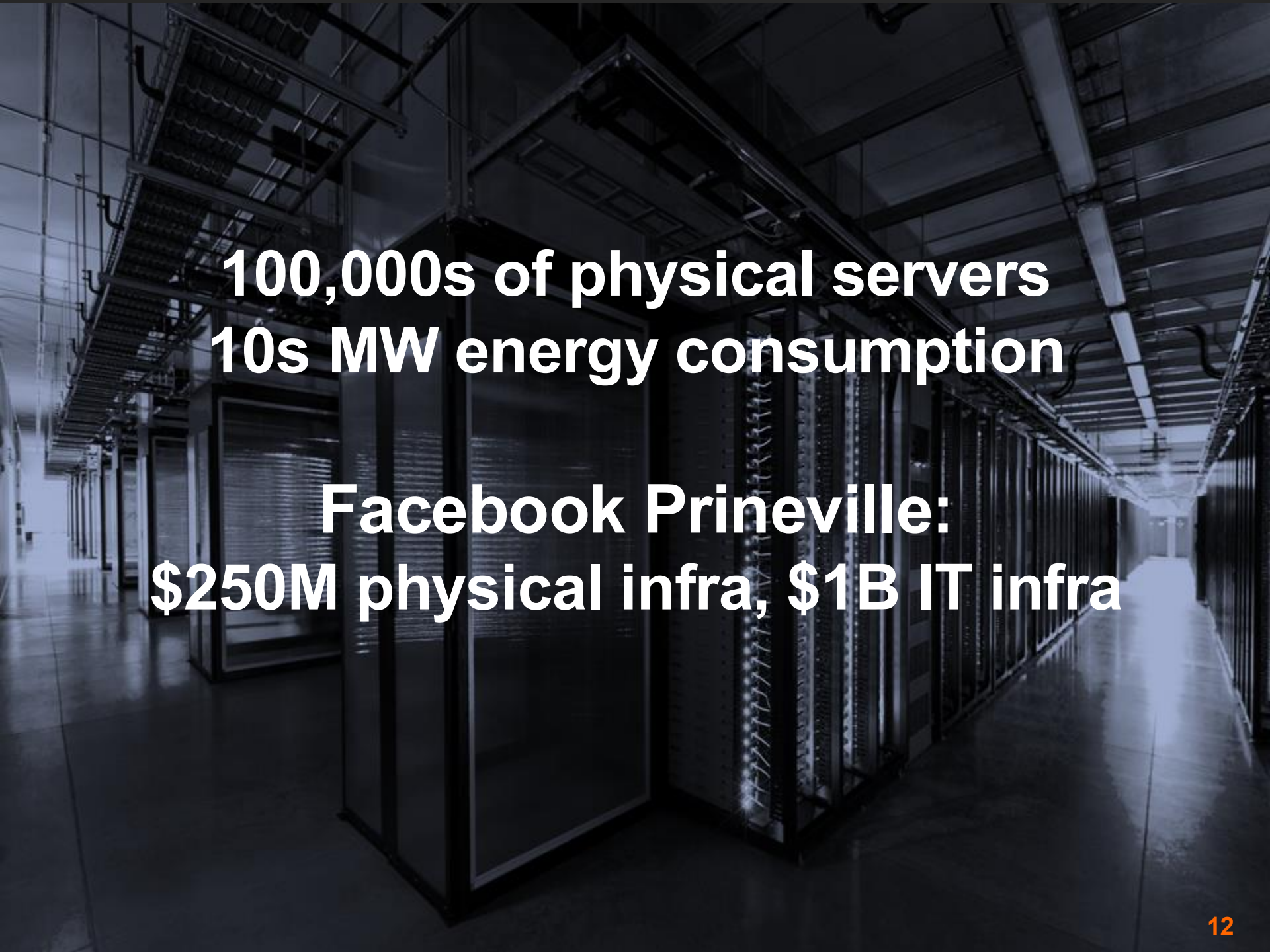


Facebook









**100,000s of physical servers  
10s MW energy consumption**

**Facebook Prineville:  
\$250M physical infra, \$1B IT infra**





**GDH DC @ KAUST**  
**~10,000 servers**  
**14.4 MW IT load**  
**8,000 m<sup>2</sup> of DC space**



# AI Tour



2025 DC in the AI Era



# The goal of “distributed systems”

---

- Service with higher-level abstractions/interface
  - e.g., file system, database, key-value store, programming model, RESTful web service, ...
- Hide complexity
  - Scalable (scale-out)
  - Reliable (fault-tolerant)
  - Well-defined semantics (consistent)
- Do “heavy lifting” so app developer doesn’t need to

# What is a distributed system?

---

- “A collection of independent computers that appears to its users as a single coherent system”
- Features:
  - No shared memory
  - Message-based communication
  - Each runs its own local OS
  - Heterogeneity
- Ideal: to present a single-system image:
  - The distributed system “looks like” a single computer rather than a collection of separate computers



# Distributed system characteristics

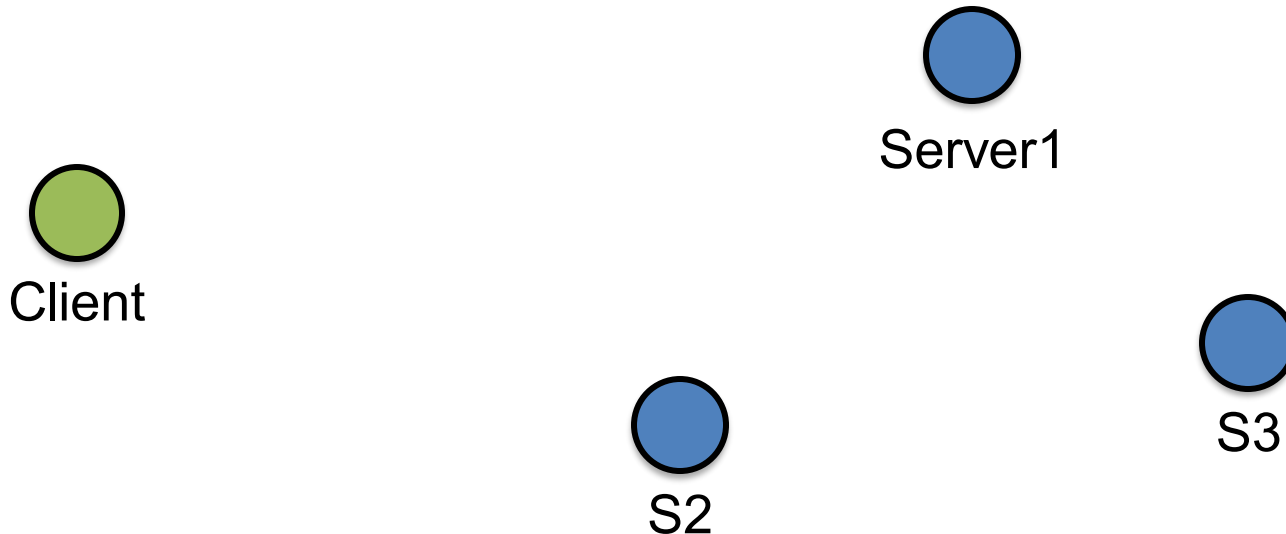
---

- To present a single-system image:
  - Hide internal organization, communication details
  - Provide uniform interface
- Easily expandable
  - Adding new computers is hidden from users
- Continuous availability
  - Failures in one component can be covered by other components

# Example

---

- Assume a distributed storage
  - Clients can read and write files





# System model

---

- $N$  **processes**  $p_1, \dots, p_N$  in the system (no process failures)
  - Every process executes an algorithm
    - An automation with set of states, set of inputs, set of outputs and a state transition function  $S \times I \rightarrow S \times O$
- There are two first-in, first-out, unidirectional **channels** between every process pair  $p_i$  and  $p_j$ 
  - Call them **channel**( $p_i, p_j$ ) and **channel**( $p_j, p_i$ )
  - All messages sent on channels arrive intact and in order
  - Channel cannot duplicate, create or modify messages

# System model

---

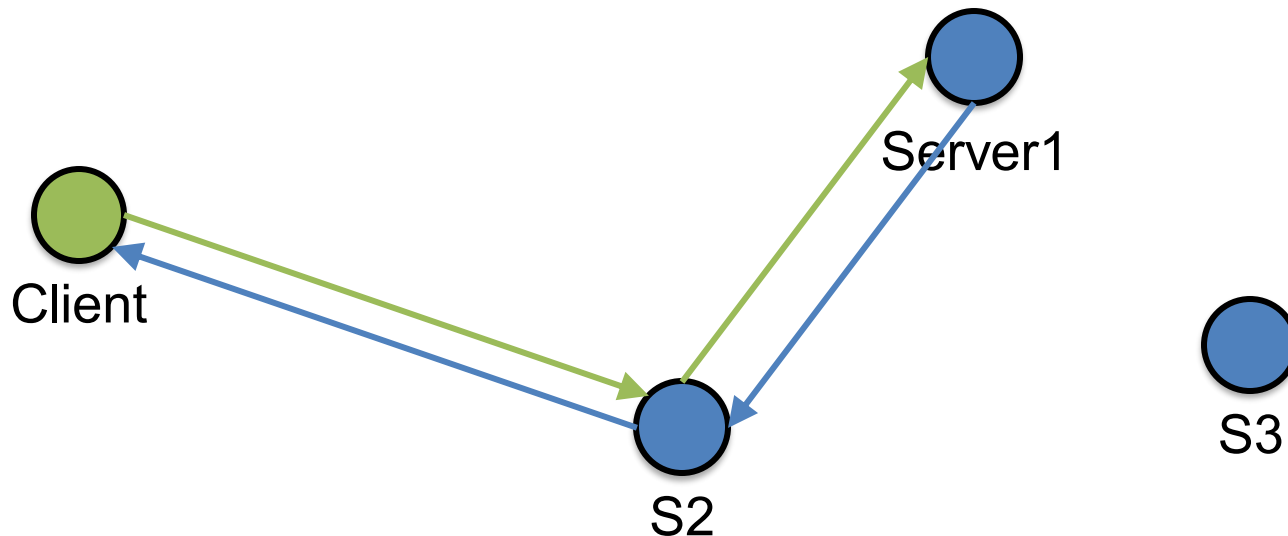
- Message passing
- No failures (for now)
- Two possible timing assumptions
  1. Synchronous System
  2. Asynchronous System
    - No upper bound on **message delivery**
    - No bound on relative **process speeds**



# Example execution

---

- Assume a distributed storage
  - Clients can read and write files



# Execution of the system

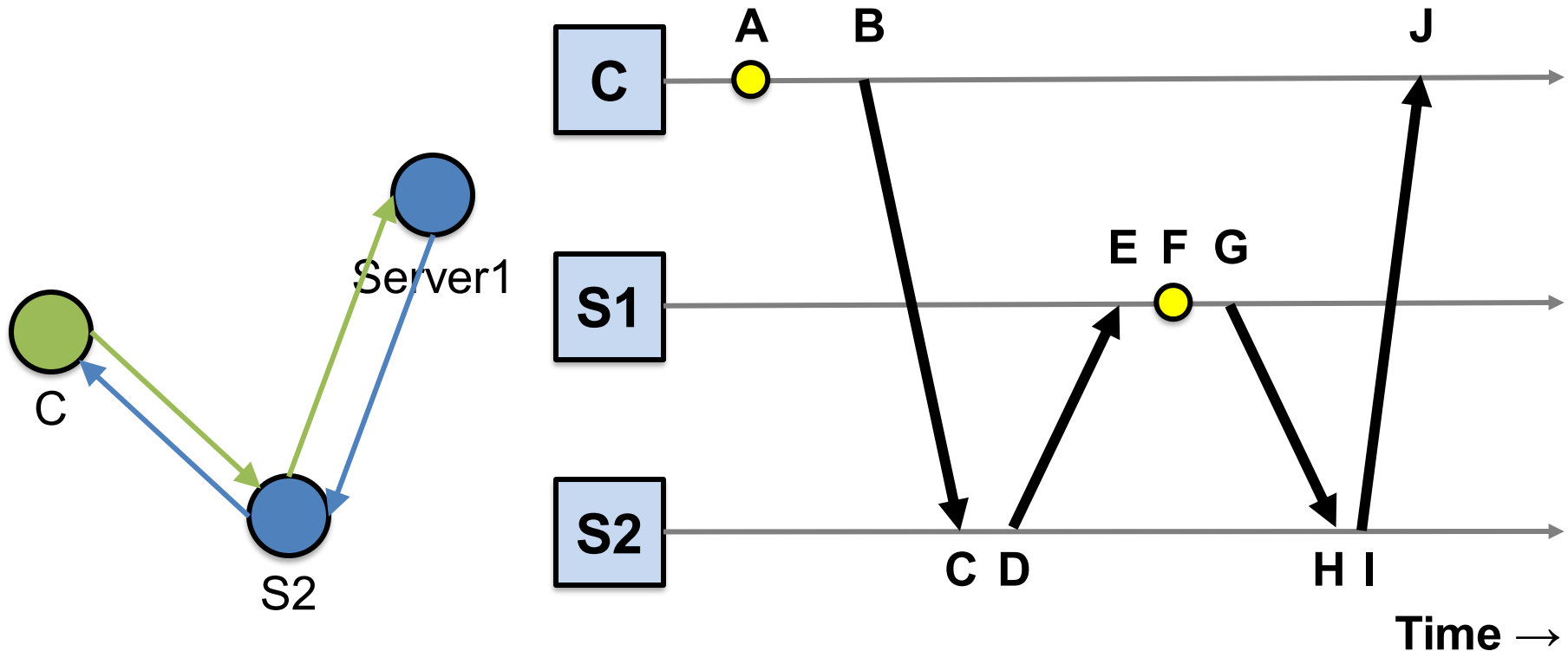
---

- Processes execute sequences of events
  - events can be of 3 types: local, send or receive
- An execution (or run) is a sequence of events that respect the system-wide distributed algorithm
  - each process is consistent with the local sequences
  - a message is sent by a process only if its (local) algorithm prescribes it to do it given the preceding sequence of its inputs
  - every received message was previously sent, and no message is received twice



# Space-Time diagrams

- A graphic representation of distributed execution



# Common failure assumption

---

- Generally, a failure occurs when a process deviates from the algorithm assigned to it
- A process is *correct* if it never fails
- *crash* failure: the faulty process prematurely stops taking steps of its algorithm
- A typical assumption is that, in every possible execution out of  $N$  processes, at most  $f < N$  can be faulty
- We call such a system *f-resilient*

# Scalable systems in this class

---

- Scale computation across many machines
  - MapReduce
- Scale storage across many machines
  - Chord, Dynamo, COPS, Spanner



# Fault tolerant systems in this class

---

- Retry on another machine
  - MapReduce
- Maintain replicas on multiple machines
  - Primary-backup replication
  - Paxos
  - RAFT
  - Bayou
  - Dynamo, COPS, Spanner

# Range of abstractions and guarantees

---

- Eventual Consistency
  - Dynamo
- Causal Consistency
  - Bayou, COPS
- Linearizability
  - Paxos, RAFT, Primary-backup replication
- Strict Serializability
  - 2PL, Spanner

# Summary

---

- Distributed Systems
  - Multiple machines doing something together
  - Pretty much everywhere and everything computing now
- “Systems”
  - Hide complexity and do the heavy lifting (i.e., **interesting!**)
  - Scalability, fault tolerance, guarantees



# Course Overview

# Philosophy and Recurring Themes

---

- Keep it real! This is the real world:
  - Things break. Components fail.
  - Latency matters. Can't beat speed of light.
  - Certain things are impossible. Need work arounds.
- How do we build systems that work at **very large scale** and **tolerate failures**?
- Given systems span many nodes, how do we enable different nodes to **agree** on “things” (e.g., time, order of operations, state of the system)?

# Learning Objectives

---

- Reasoning about concurrency
- Reasoning about failure
- Reasoning about performance
  
- Building systems that correctly handle concurrency and failure
  
- Knowing specific system designs and design components



# Course Goals

---

- Gain an understanding of the principles and techniques behind the design of modern, reliable, and high-performance systems
- In particular learn about distributed systems
  - Learn general systems principles (modularity, layering, ...)
  - Practice implementing real, larger systems that must run in nasty environment

# Course Organization

<http://sands.kaust.edu.sa/classes/CS240/F25/>

# Learning the material: People

---

- Lecture
  - Professor Marco Canini
  - Slides available on course website
  - Office hours: by appointment
- TAs
  - Jihao Xin
  - Achref Rebai
- Main Q&A forum: [www.campuswire.com](http://www.campuswire.com)
  - No anonymous (to instructors) posts or questions
  - Can send private messages to instructors





# Learning the material: Books

---

- Lecture notes!
- No required textbooks
- Check the website for recommended references
- Main references available in the Library
  - (linked on the website)

# Grading

---

- Active participation (10% total)
  - Includes attendance, discussion and peer-review of project proposals
- Three programming assignments (15% total)
  - 3% 1st, 3% 2nd, 9% 3rd
- Open-note final exam (25% total)
- Term project (50% total)
  - 35% report and deliverables, 15% presentation and Q&A

# Final Exam

---

- Test learning objectives mostly using designs covered in lectures
- And test knowledge of specific design patterns and designs
- Open note/book (but if you don't study it will create time pressure)
- Recipe for success:
  - Attend lecture and actively think through problems
  - Ask questions during lecture and afterwards in my office hours
  - Actively work through problems
  - Complete programming assignments
  - Study lecture materials for specific design patterns and designs
  - Run the system designs in your mind and see what happens

# About Assignments

---

- Systems programming somewhat different from what you might have done before
  - Low-level (C / Go)
  - Often designed to run indefinitely (error handling must be rock solid)
  - Must be secure - horrible environment
  - Concurrency
  - Interfaces specified by documented protocols
- TAs' Office Hours
- Read: Dave Andersen's "[Software Engineering for System Hackers](#)"
  - Practical techniques designed to save you time & pain



# Why use Go?

---

- Easy concurrency w/ goroutines (lightweight threads)
- Garbage collection and memory safety
- Libraries provide easy RPC
- Channels for communication between goroutines

# Where is Go used?

---

- Google, of course!
- Docker (container management)
- CloudFlare (Content delivery Network)
- Digital Ocean (Virtual Machine hosting)
- Dropbox (Cloud storage/file sharing)
- Netdata (monitoring platform)
- ... and many more!

# About Assignments

---

- Reinforce / demonstrate learning objectives!
- 1: Sequential Map/Reduce (due September 17)
- 2: Distributed Map/Reduce (due September 24)
- 3: RAFT Consensus (due October 19)

# Programming Assignments

---

- Recipe for disaster
  - Start day assignment is due
  - Write code first, think later
  - Test doesn't pass => randomly flip some bits
  - Assume you know what program is doing



# Programming Assignments

---

- Recipe for success
  - Start early (weeks early)
  - Think through a complete design
  - Progressively build out your design (using tests to help)
  - Checkpoint progress in git (and to gitlab) frequently
  - Debug, debug, debug
    - Verify program state is what you expect (print it out!)
    - Write your own smaller test cases
    - Reconsider your complete design
  - Attend office hours

# About Term Project

---

- Open ended project within course scope
  - Must hit at least one trait of distributed systems: reliability, scalability, correctness/guarantees
  - Topics and suggestions will be available
  - Research reproductions possible
- Solo or groups of two students
- Structure
  - Lightweight proposal (due October 29)
  - Final report (based on template, due December 7) and in-class presentation with Q&A

# Reproduction: How to Approach It?

---

1. Pick an interesting paper
2. Ask what kind of “reproduction” is appropriate?
3. Does the primary result of the paper hold up?
4. What happens if you vary a parameter the original experimenter didn't consider?
5. Having reproduced the primary result, can you now extend or improve the work?
6. What was difficult to reproduce?

# Policies: Collaboration

---

- Working together important
  - Discuss course material
  - Work on problem debugging
  - Work on term project
- Parts **must** be your own work
  - Exam, programming assignments
- What we hate to say: we run cheat checkers...  
**they work surprisingly well**
- Please *\*do not\** put code on *\*public\** repositories

# Policies: Write Your Own Code

---

Programming is an individual creative process. At first, discussions with friends is fine. When writing code, however, the program must be your own work.

Do not copy another person's programs, comments, README description, or any part of submitted assignment. This includes character-by-character transliteration but also derivative works. Cannot use another's code, etc. even while "citing" them.

Writing code for use by another or using another's code is academic fraud in context of coursework.

Do not publish your code e.g., on Github, during/after course!



# Policies: AI Tools

---

- AI tools cannot be used for the three programming assignments
- AI tools can be used as aid in the term project
  - Including for coding
- Note: with regard to the term project, it is expected that you fully understand your system design, design decisions and what has been implemented

# Policies: Late Work

---

- 72 late hours on programming assignments
- After that, each additional day late will incur a 10% lateness penalty
  - (1 min late counts as 1 day late)
- Submissions late by 3 days or more will no longer be accepted
  - (Fri and Sat count as days)
- In case of illness or extraordinary circumstance (e.g., emergency), talk to us early!

# Conclusion

---

- Attend lectures, think actively!
- Start programming assignments early, use the right strategy!
- Super cool distributed systems stuff starts Thu!

