EECS 482
Introduction to Operating Systems

Winter 2018

Harsha V. Madhyastha

Recap: RPC

- Make network communication transparent to developer
- Given definitions of server functions, automate
  - Generation of client-side and server-side stubs
  - Communication between stubs

- Example: Project 4
  - Client library includes client-side stubs

Making a distributed system look like a local system

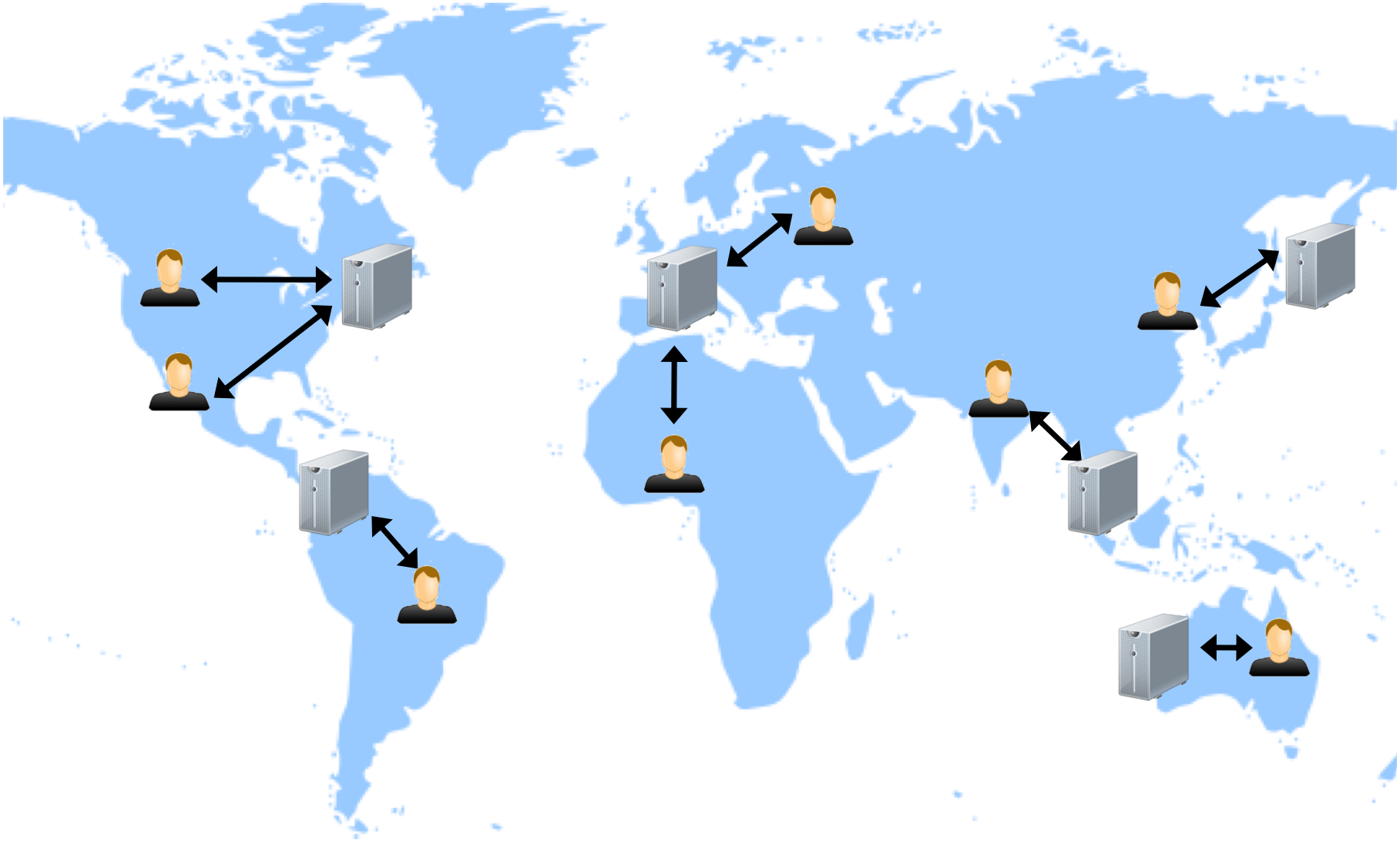
- RPC: make request/response look like function call/return
- DSM: make multiple memories look like a single memory
- DFS: make disks on multiple computers look like a single file system
- Parallelizing compilers: make multiple CPUs look like one CPU
- Process migration (and RPC): allow users to easily use remote processors

Example Scenario

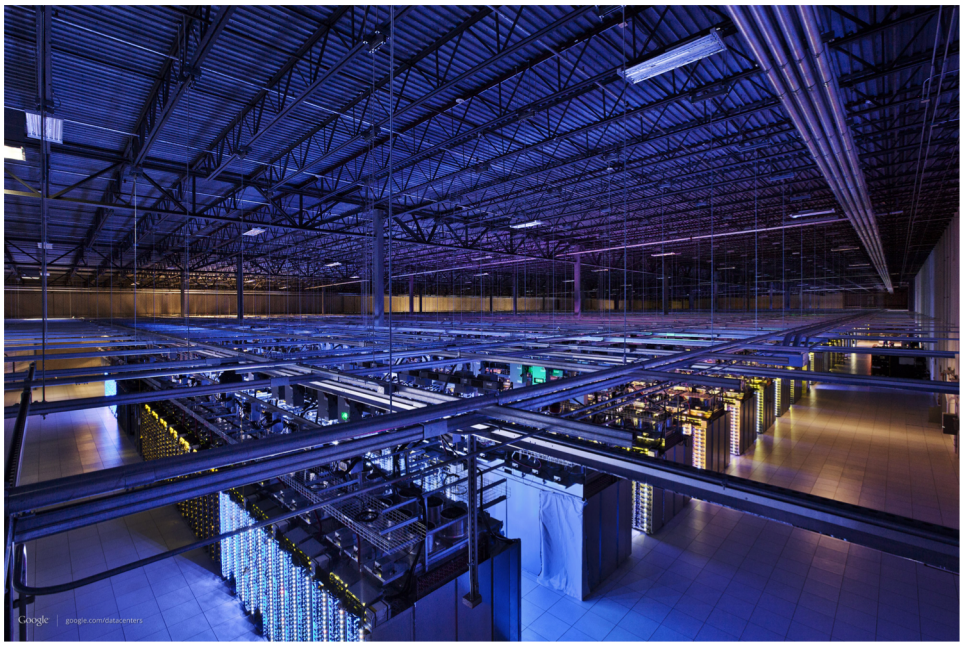
- Consider user issuing search query to Google
- Google's objectives in serving query?
  - Resilience to failures
  - Low latency
  - Most relevant results
- What distributed systems necessary to achieve these goals?

Google in 1997

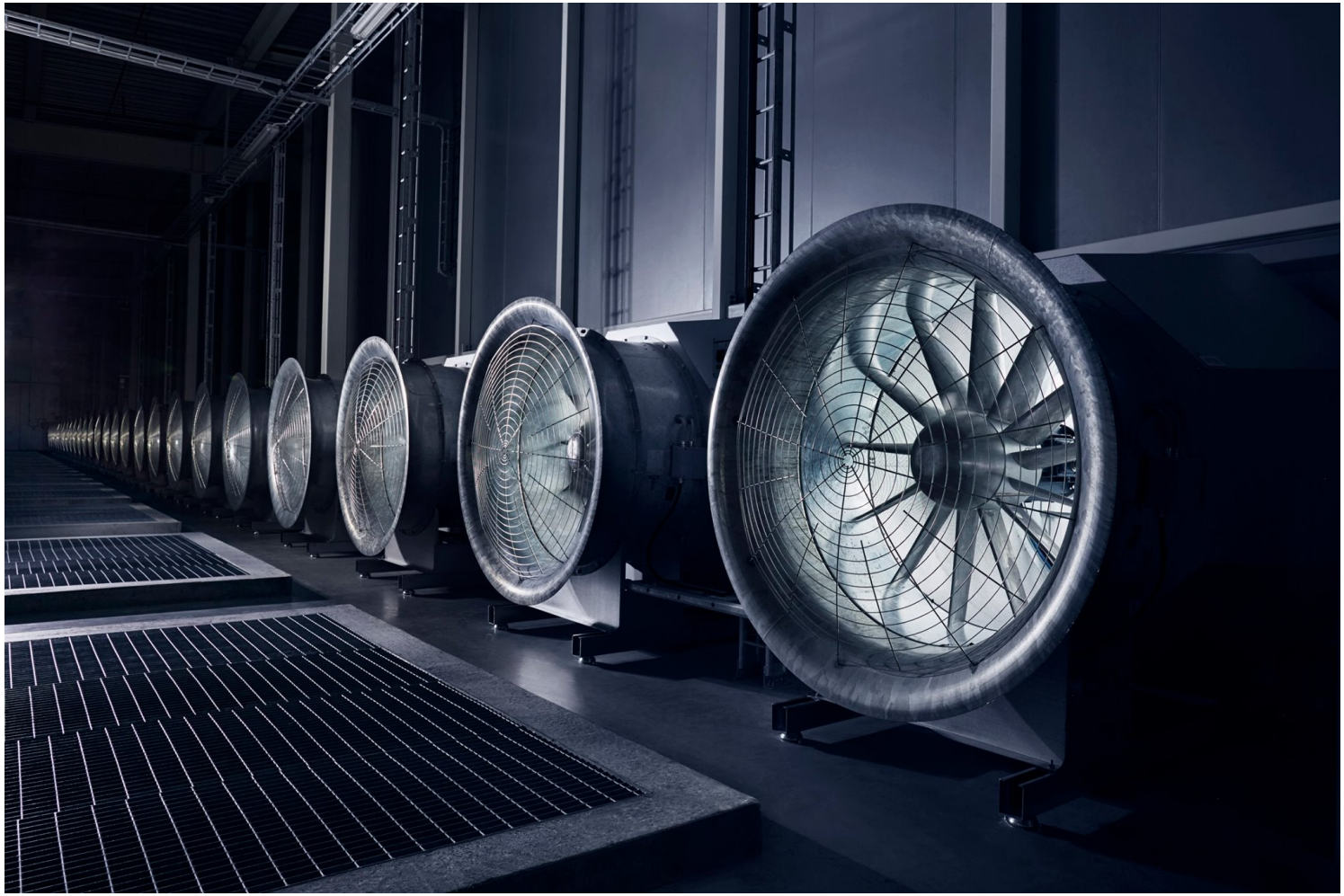
Geo-Distributed Services



Data Centers



- Spread services and data storage/processing across 100s of thousands of machines



Other Distributed Systems

- Internet
- Data center network

- Domain Name System

- OS for multi-core processors

Why Distributed Systems?

- Conquer geographic separation
  - Facebook and Google customers span the planet
- Build reliable systems with unreliable components
- Aggregate systems for higher capacity
  - CPU cycles, memory, disks, network bandwidth
  - Cost grows non-linearly
- Customize computers for specific tasks
  - Example: cache server, speech-to-text conversion server

Jeff Dean "Facts"

Jeff Dean writes directly in binary. He then writes the source code as a documentation for other developers.

Compilers don't warn Jeff Dean. Jeff Dean warns compilers.

Jeff Dean builds his code before committing it, but only to check for compiler and linker bugs.

Challenge 1: Partial failures

"A distributed system is one where you can't get your work done because some machine you've never heard of is broken." – Leslie Lamport

Facebook's Prineville Data Center

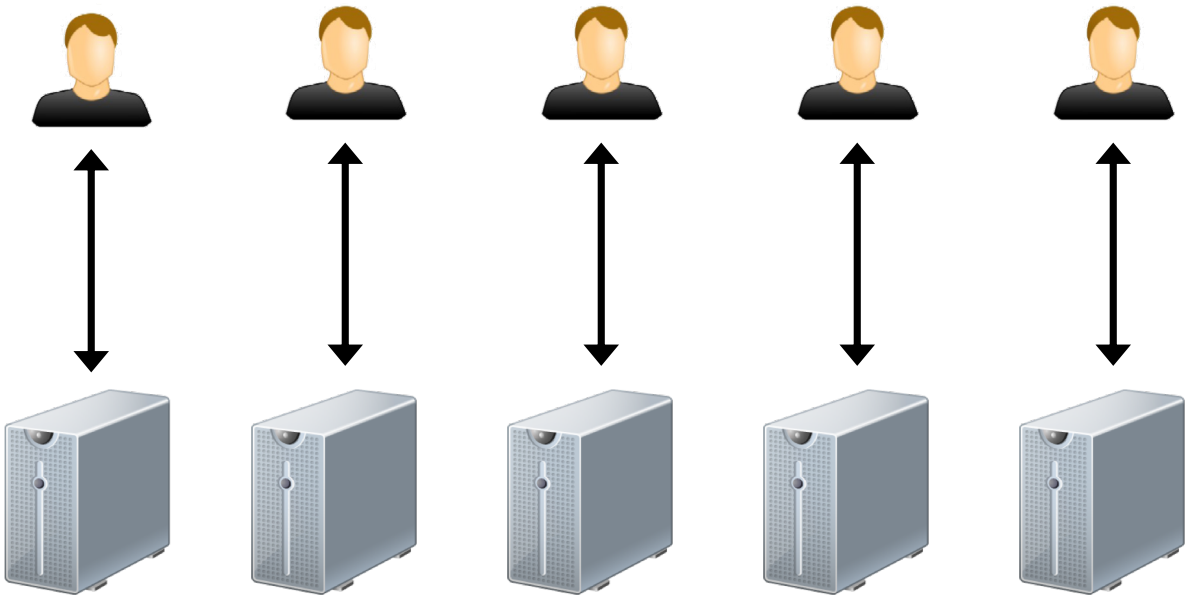
- Contents (approx.):
  - 200K+ servers
  - 500K+ disks
  - 10K network switches
  - 300K+ network cables
- At any instant, likelihood that all components correctly functioning?
- Personal anecdote: Tritonsort

Challenge 2: Ambiguous failures

- If a server doesn't reply, how to tell if
  - The server has failed
  - The network is down
  - Neither; they are both just slow
- Makes failure detection hard

Challenge 3: Concurrency

Why not partition users across machines?

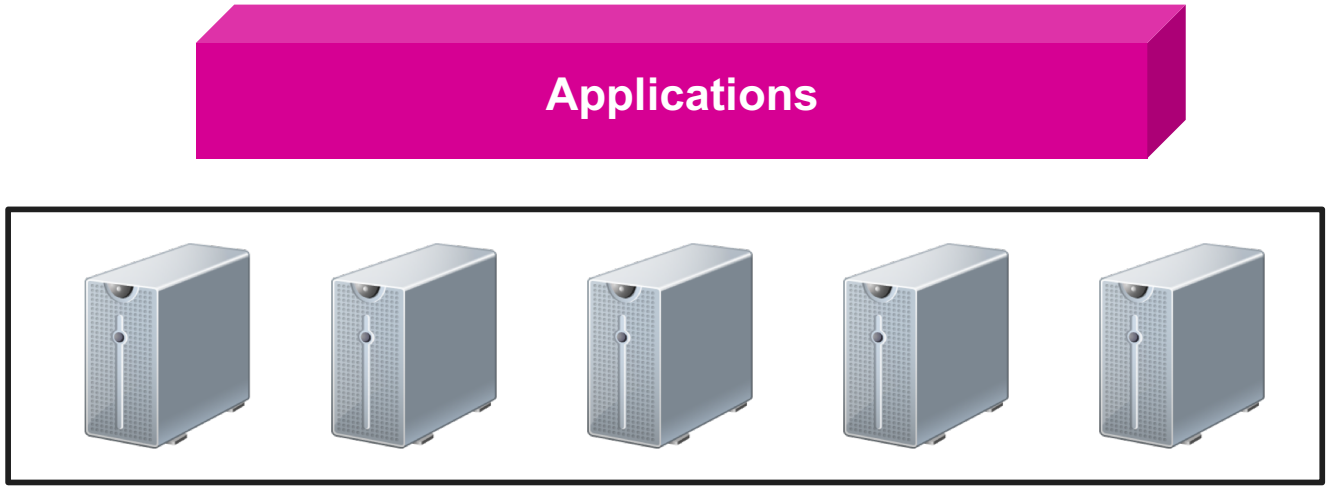


Challenge 3: Concurrency

- How to ensure consistency of distributed state in the face of concurrent operations?
- Use mutex, cv, semaphore, etc.?
- Need to synchronize based on unreliable messages

Distributed Abstractions

- Unify several machines from app perspective
- Enable concurrent use, hide failures

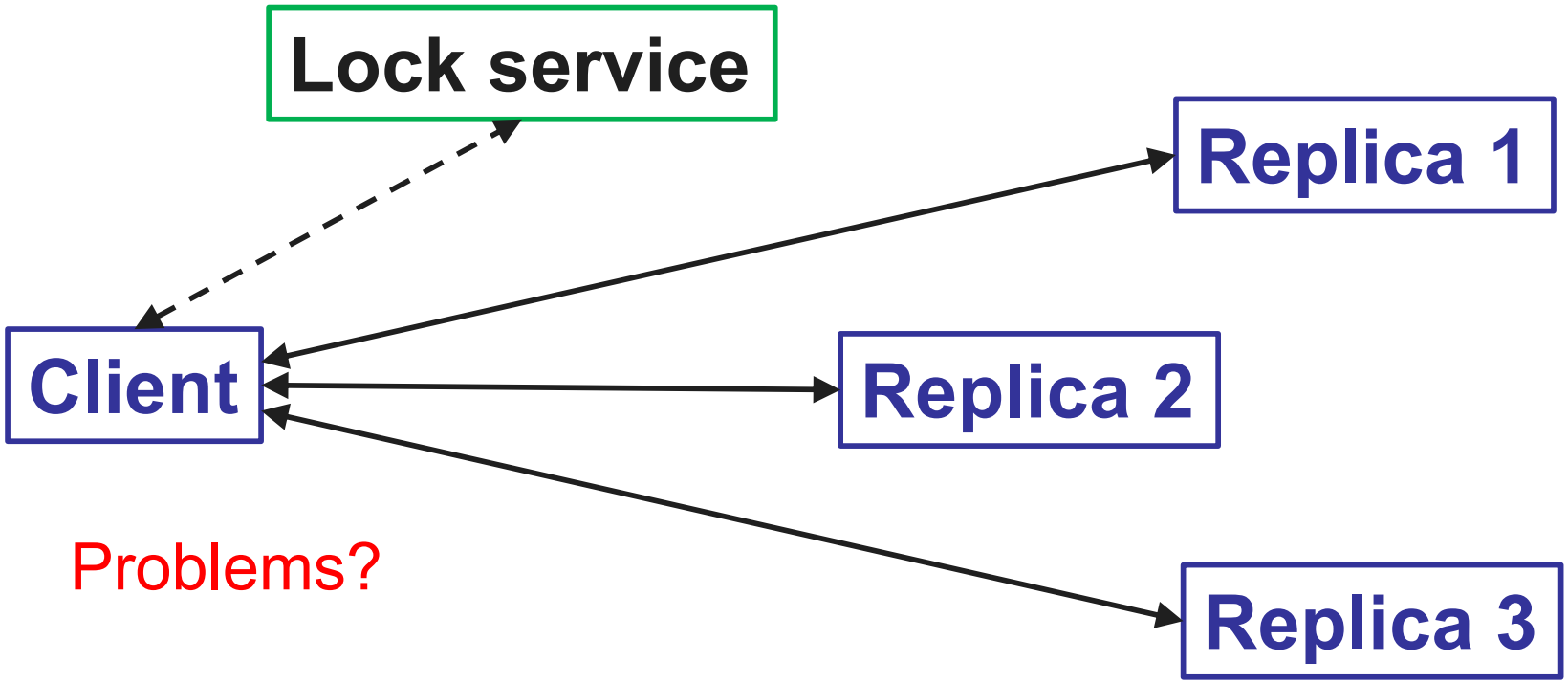


- Do “heavy lifting” so developers don’t need to

Project 4

- Use assertions to catch errors early
  - No. of free disk blocks matches file system contents?
  - Are you unlocking a lock that you hold?
  - Verify initial file system is not malformed
- Use showfs to verify that contents of file system match your expectations
- Write test cases to get file server to crash

Distributed Mutual Exclusion



Problems?

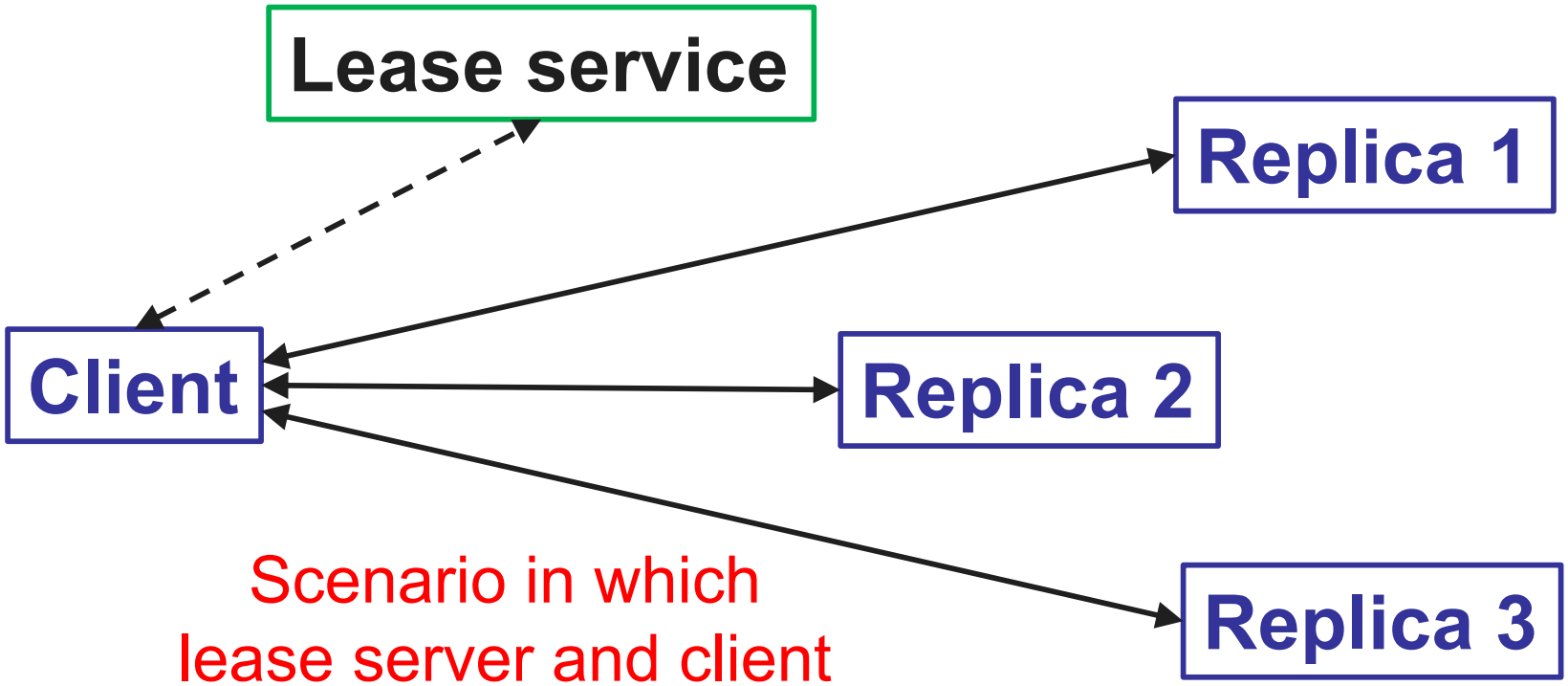
Client failures!

Lease

- Lock with timeout
- If lease holder fails, not a problem because lease will expire

- How to pick lease timeout value?
  - Short timeout → Client needs to renew lease
  - Long timeout → Unnecessarily block operations

Discrepancy in Lease Validity

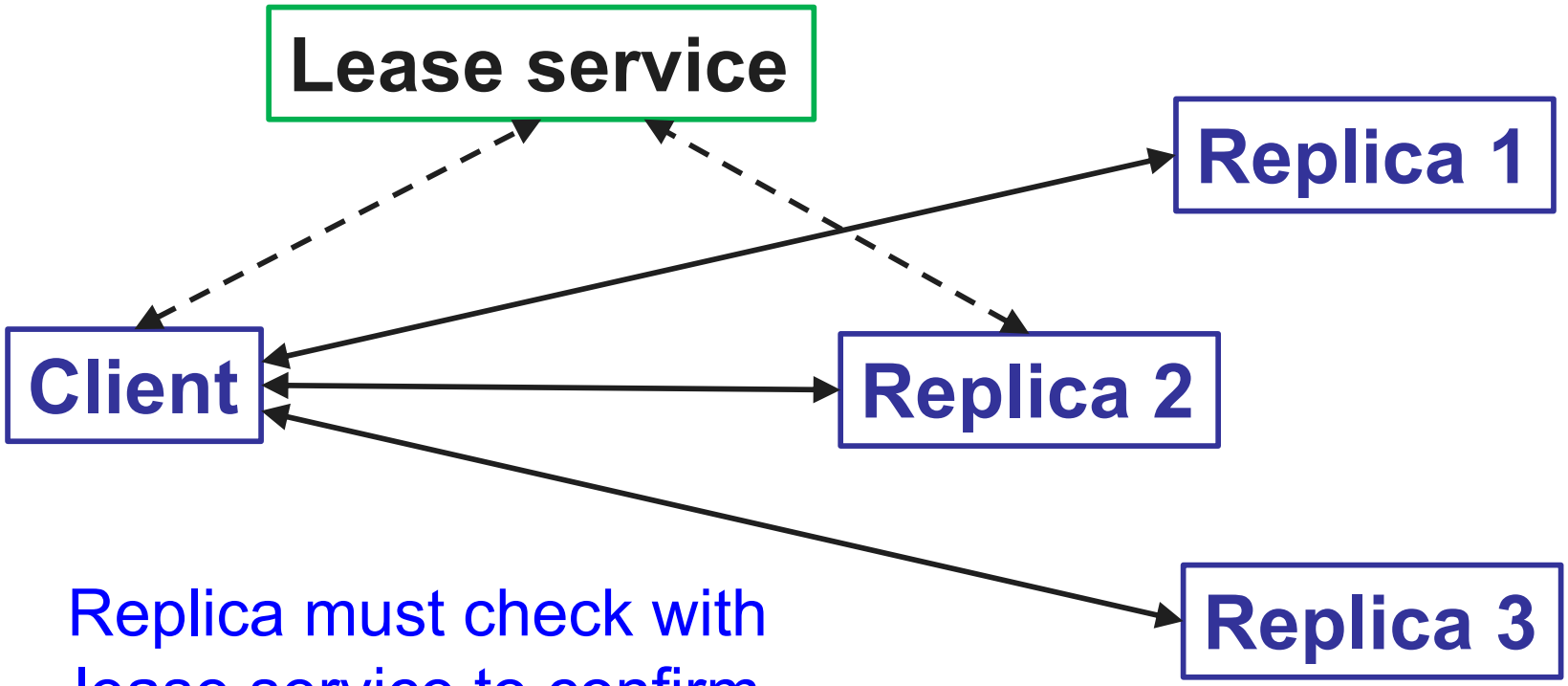


Scenario in which
lease server and client
differ about lease validity?

Discrepancy in Lease Validity

- Message that grants lease may have high delay
- Clock at lease holder and lease service may not be in sync
- How to account for potential discrepancy?

Discrepancy in Lease Validity

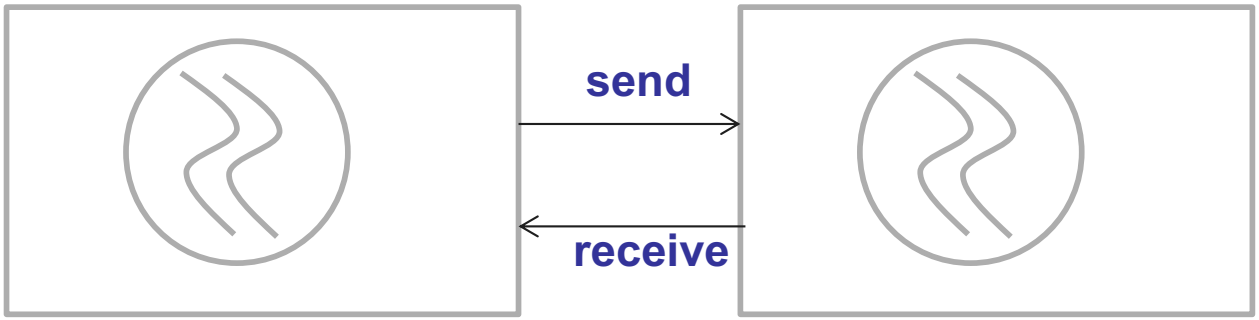


Replica must check with lease service to confirm lease validity

Structuring a concurrent system

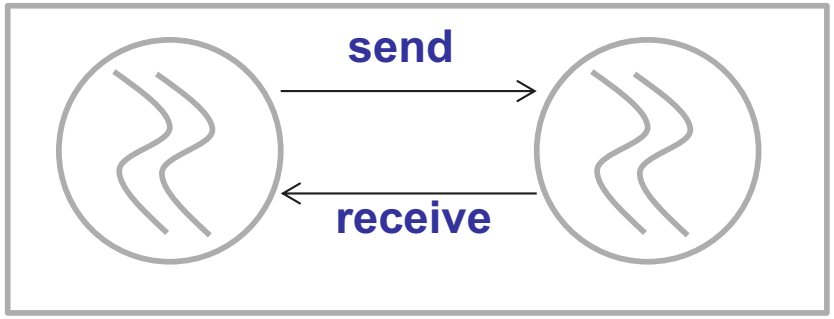
- One multi-threaded process on one computer

- Several multi-threaded processes on several computers



Structuring a concurrent system

- Several multi-threaded process on each of several computers



- Why separate threads on one computer into separate address spaces, then use send/receive to communicate and synchronize?
  - Protects modules from each other
- Microkernels
  - OS structure that separates OS functionality into several server processes, each in its own address space

Next time ...

- Distributed file system