

Based on slides by Harsha V. Madhyastha

# EECS 482 Introduction to Operating Systems

Spring/Summer 2020

Lecture 23: Distributed systems

Nicole Hamilton

[https://web.eecs.umich.edu/~nham/  
nham@umich.edu](https://web.eecs.umich.edu/~nham/nham@umich.edu)

# Agenda

1. Project 3.
2. Project 4.
3. Distributed systems.

# Agenda

1. Project 3.
2. Project 4.
3. Distributed systems.

# Project 3 scores

I forgot we were going to do style grading and mistakenly posted the raw AG scores to Canvas.

We will again do style grading on Gradescope.

I will update your scores once the style grading is done.

# Agenda

1. Project 3.
2. **Project 4.**
3. Distributed systems.

# Project 4

Use **assertions** to catch errors early.

Number of free disk blocks should be consistent with the file system contents.

Ensure you properly unlock locks 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.

If a request is invalid, it doesn't matter why, you just reject it.

# Project 4 Testing

## State space coverage

Test every request type with every possible state

## For example: FS\_CREATE

File vs. directory

In root directory vs. elsewhere

Already exists vs. does not exist

Adding direntry in first data block vs. later

Test close to resource limits

Disk size, max path name, max file name, ...

# Agenda

1. Project 3.
2. Project 4.
3. **Distributed systems.**



# Distributed Systems

What's hard about making a system distributed?

What's required to enable distributed systems abstractions?

Intro to these topics today and next Monday.

If this piques your interest, take either:

EECS 491 W21 Intro to Distributed Systems

EECS 591 F20 Distributed Systems

# Recap: RPC

Make distributed system look like local system

Given definitions of server functions, automate:

1. Generation of client-side and server-side stubs
2. Communication between stubs

**Example:** Client library for project 4

**Assumption in Project 4: Single server**

**Why would we want to have multiple servers?**

Think “how to use P4 code to support Dropbox?”

# Making a distributed system look like a local system

**RPC** Remote Procedure Call. Make request/response look like function call/return.

**DSM** Distributed Shared Memory. Make multiple memories look like a single memory.

**DFS** Distributed File System. 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.

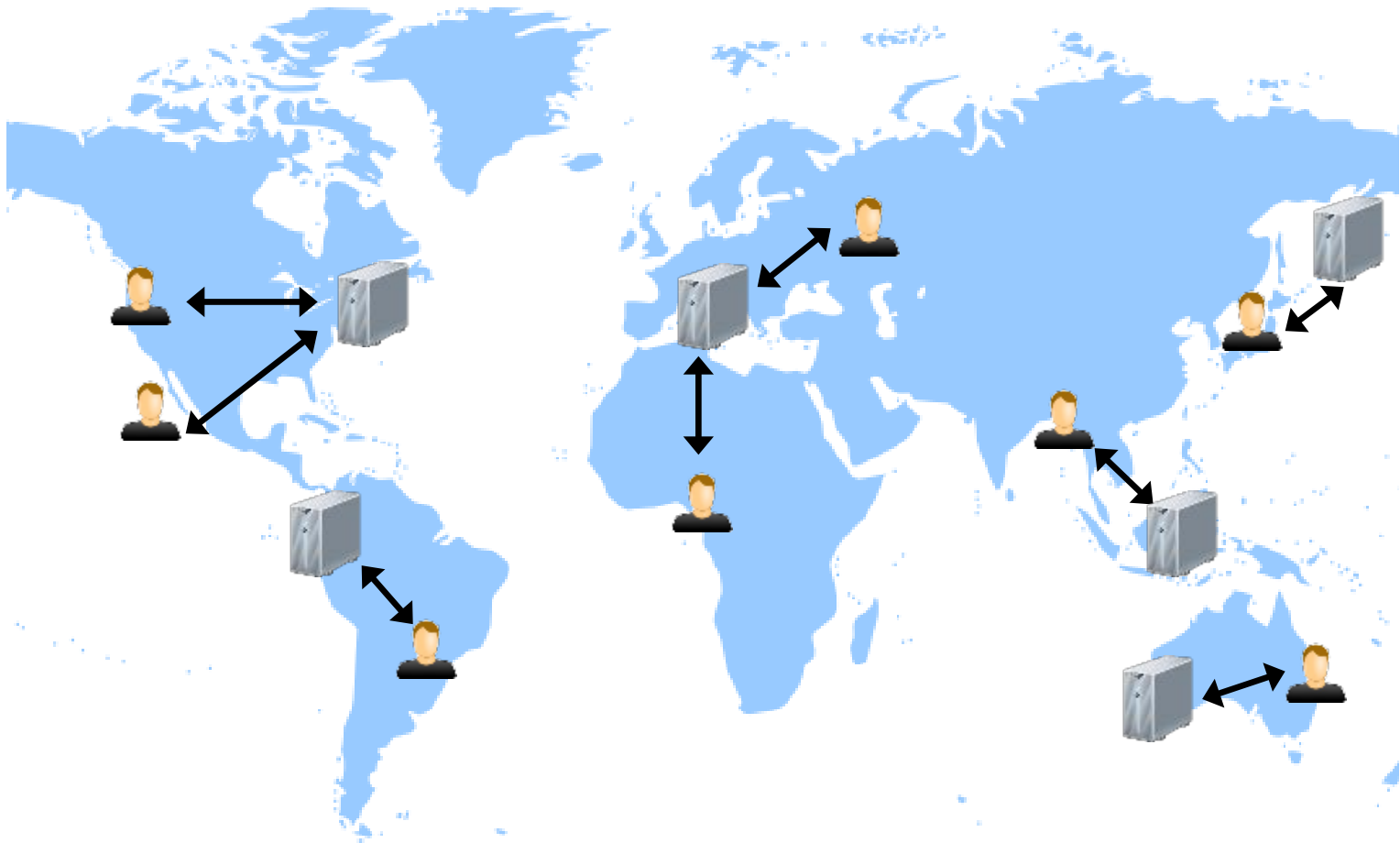
What is necessary for their distributed system to achieve these goals?

# Google in 1997



# Why Distributed Systems?

Conquer geographic separation. Customers may span the planet.



# Why Distributed Systems?

Customize computers for specific tasks.

Example: cache server, speech-to-text conversion server.

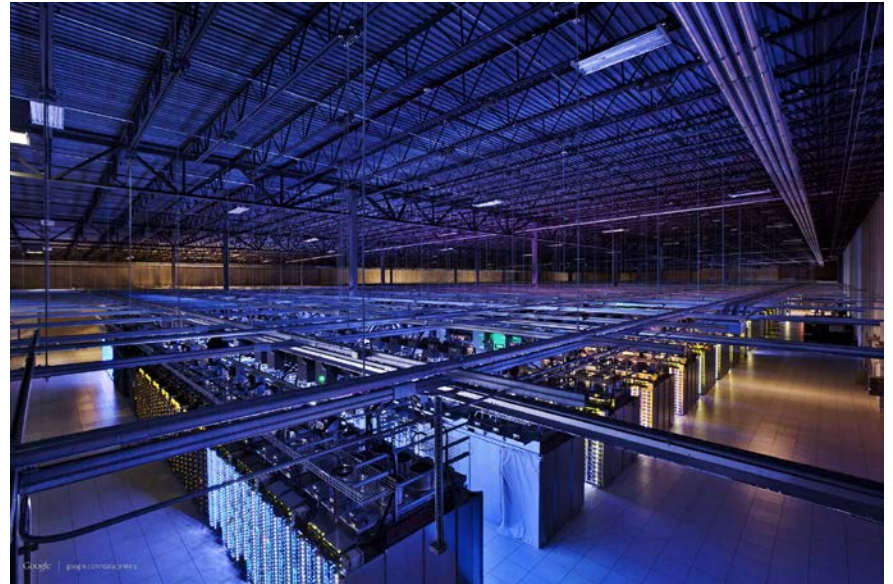


# Data Centers

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

Build reliable systems with unreliable components.

Aggregate systems for higher capacity.





# Data Centers

They require enormous amounts of power and cooling, so that's why they're often placed near hydroelectric or geothermal plants in cool climates.




(4) Inside Facebook's Oregon data center

youtube.com/watch?v=4A\_A-CmrqQ

Seattle PI WSJ WSJ NY Times top WaPost Canvas Course Info Center Crappy Scheduler BofA Wikipedia Other bookmarks

YouTube Search



0:00 / 1:48

Inside Facebook's Oregon data center (CNET News)

120,760 views • Jul 14, 2016

894 66 SHARE SAVE ...

[https://www.youtube.com/watch?v=4A\\_A-CmrqQ](https://www.youtube.com/watch?v=4A_A-CmrqQ)

# Why Distributed Systems?

Conquer geographic separation.

Facebook and Google customers span the planet.

Build reliable systems with unreliable components and aggregate systems for higher capacity.

Objective is more CPU cycles, memory, disks, network bandwidth.

Cost grows non-linearly with increased performance of an individual system.

Customize computers for specific tasks.

Example: cache server, speech-to-text conversion server.

# Jeff Dean at Google

Head of AI at Google.

Known for many of Google's key distributed systems technologies, including MapReduce, Bigtable, Spanner and TensorFlow.



Image source: [https://media.wired.com/photos/5df3cd70aa0b880008b6fc35/1:1/w\\_714,h\\_714,c\\_limit/Biz-JeffDean-h\\_15006736.jpg](https://media.wired.com/photos/5df3cd70aa0b880008b6fc35/1:1/w_714,h_714,c_limit/Biz-JeffDean-h_15006736.jpg)

# Jeff Dean at Google

Little-known facts about Jeff Dean

**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.**



Image source: [https://media.wired.com/photos/5df3cd70aa0b880008b6fc35/1:1/w\\_714,h\\_714,c\\_limit/Biz-JeffDean-h\\_15006736.jpg](https://media.wired.com/photos/5df3cd70aa0b880008b6fc35/1:1/w_714,h_714,c_limit/Biz-JeffDean-h_15006736.jpg)

# 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, 2013 Turing Award winner



# Facebook's Prineville Data Center

Contents (approx.):

- 200K+ servers

- 500K+ disks

- 10K network switches

- 300K+ network cables

Typical failure rate for disks is 2% to 4% per year.

**At any instant, unrealistic to expect everything will be working.**

# Challenge 2: Ambiguous failures

If a server doesn't reply, how do you to tell if:

1. The server has failed.
2. The network is down.
3. Neither; they are both just slow.

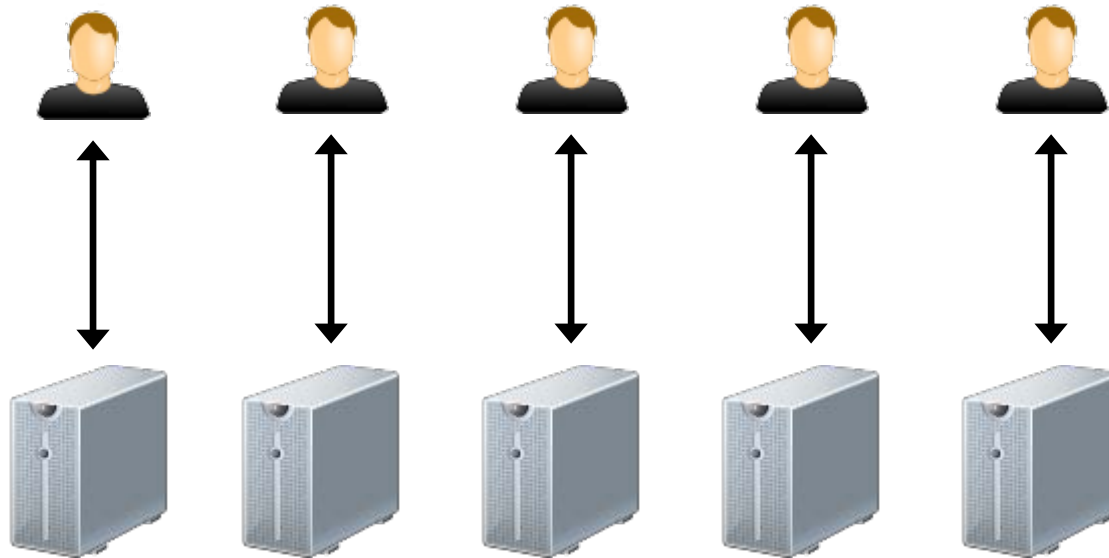
Solution: Might see if you can reach other sites and if they can reach that server, e.g., with a proxy server.

Detecting failures can be hard.



# Challenge 3: Concurrency

Why not partition users across machines?



**Shared State**

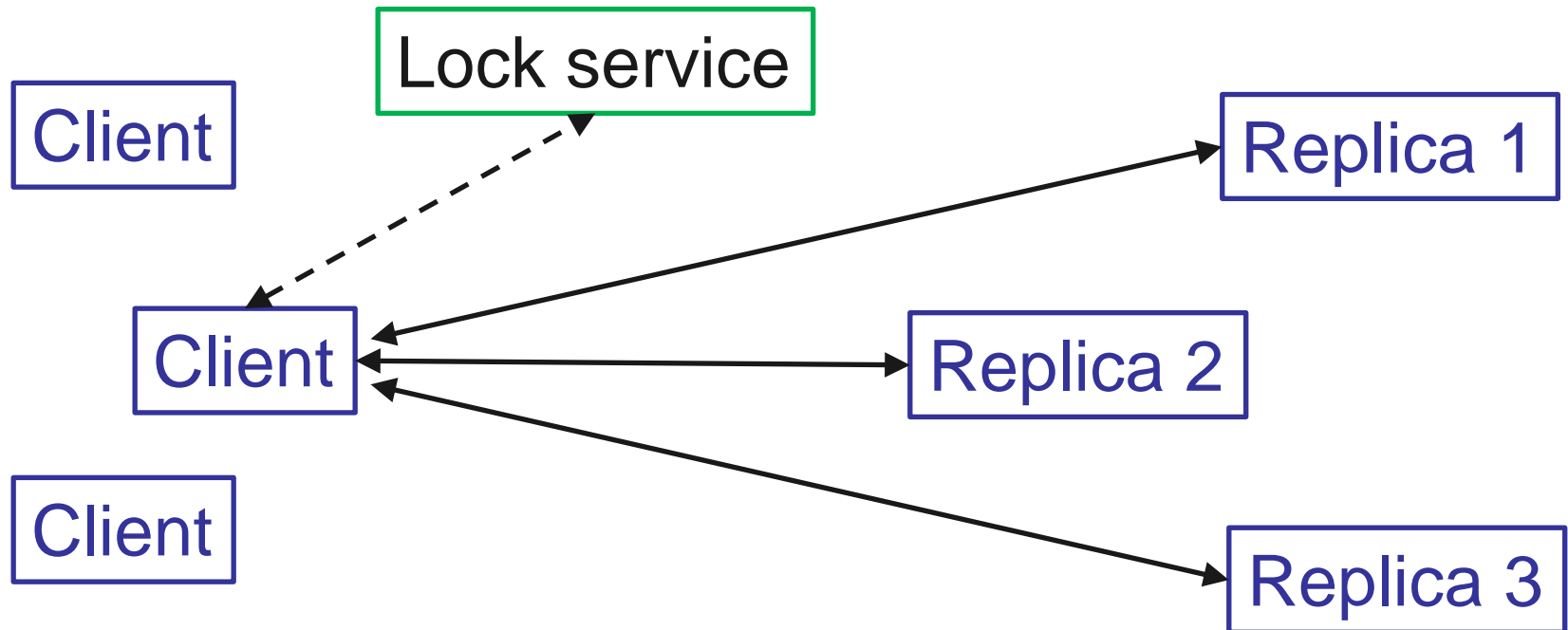
# Challenge 3: Concurrency

How to ensure consistency of distributed state in the face of concurrent operations?

Use mutex, cv, semaphore, etc.?

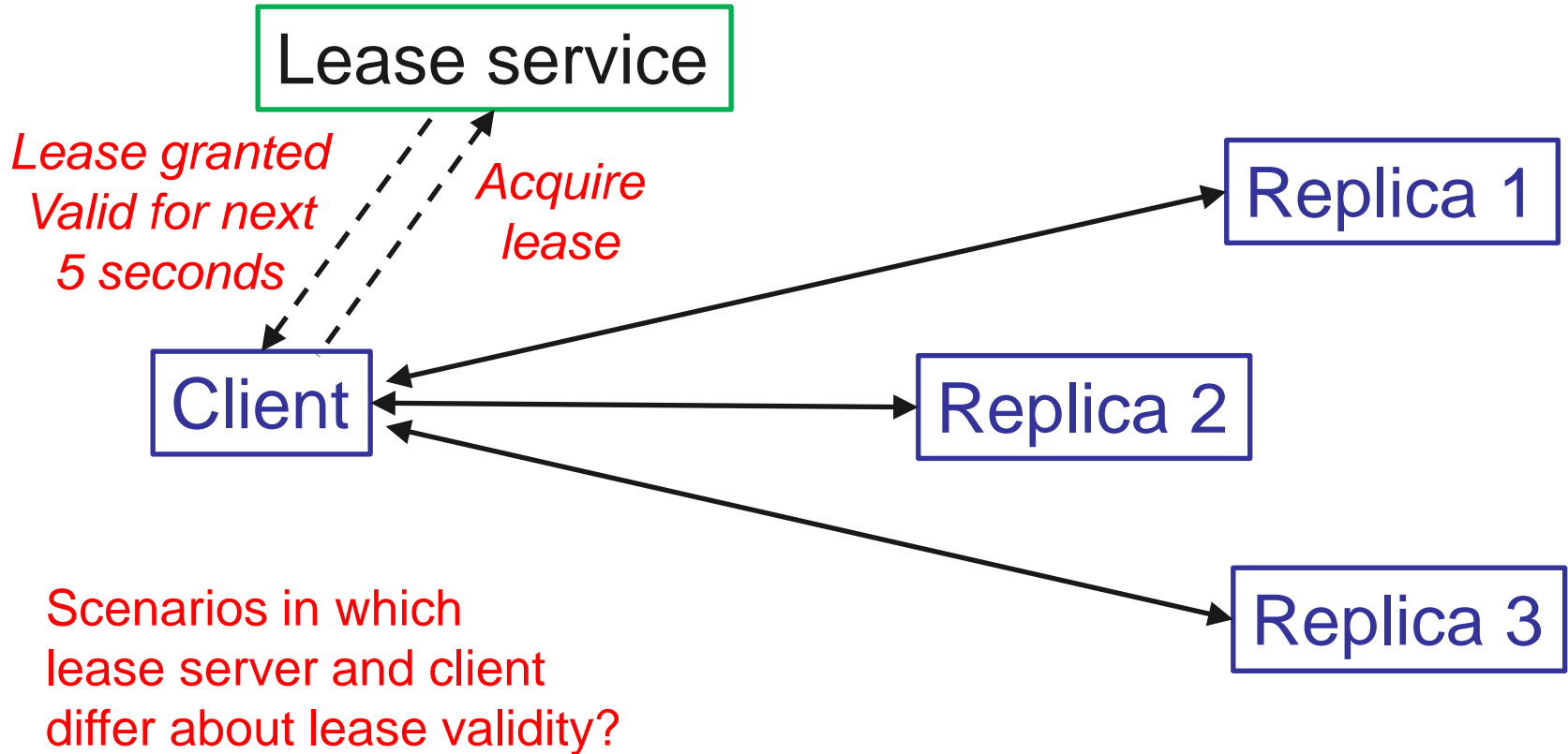
Need to synchronize based on unreliable messages.

# Distributed Mutual Exclusion

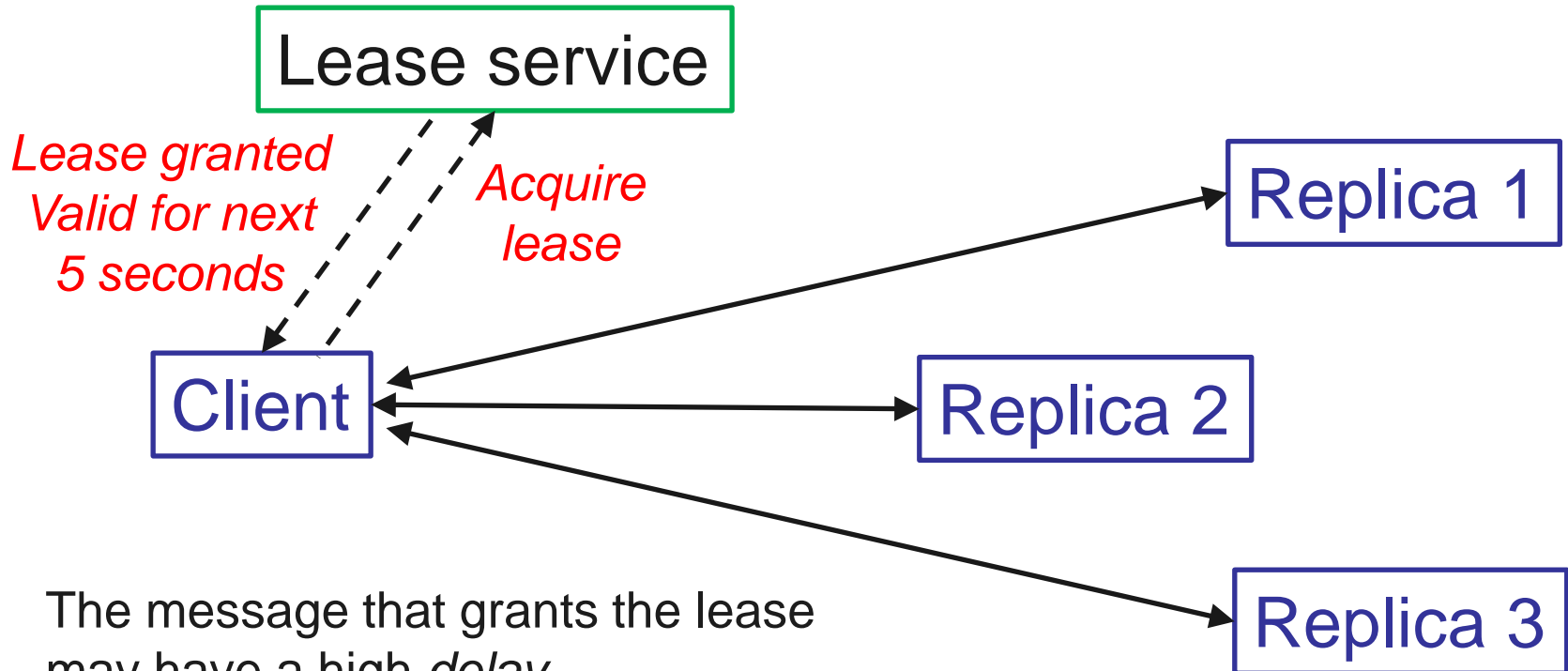


Problems?  
Client failures!

# Lease: Lock with timeout



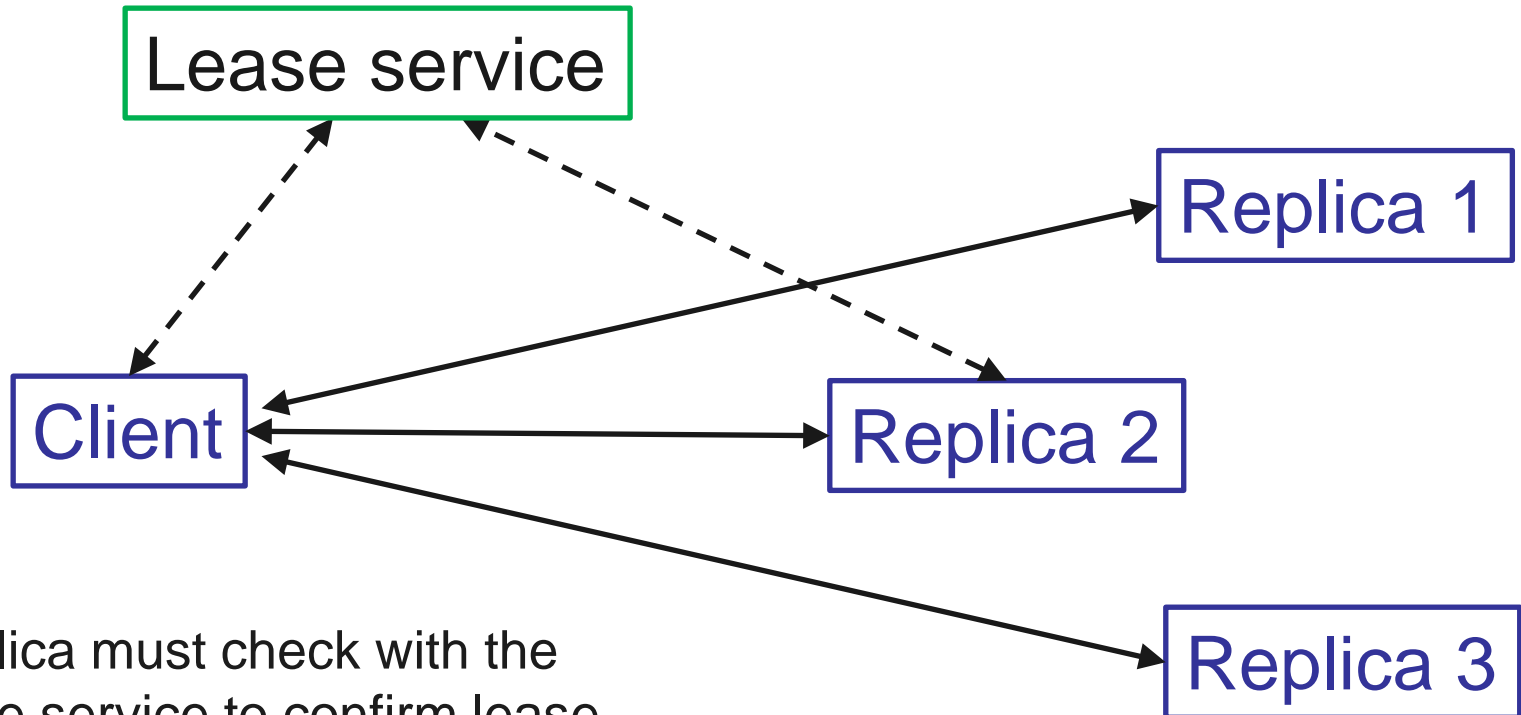
# Discrepancy in Lease Validity



The message that grants the lease may have a high *delay*.

There may be *skew* between clocks at lease holder and lease service.

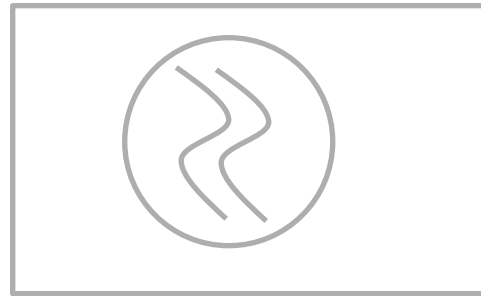
# Discrepancy in Lease Validity



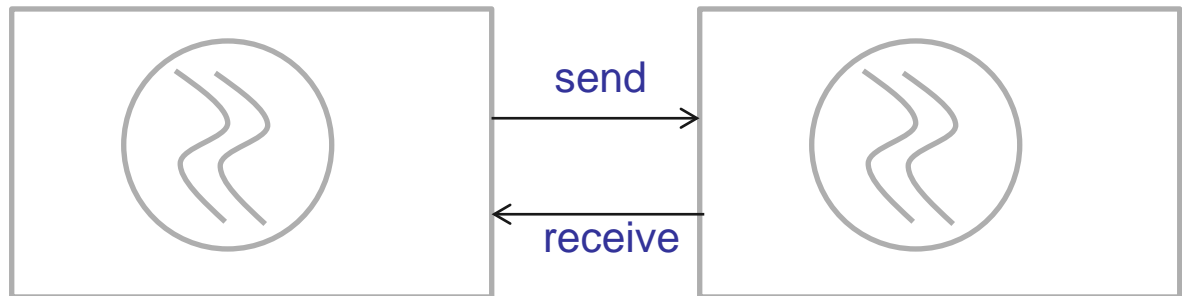
Replica must check with the lease service to confirm lease validity.

# Structuring a concurrent system

One multi-threaded process on one computer.

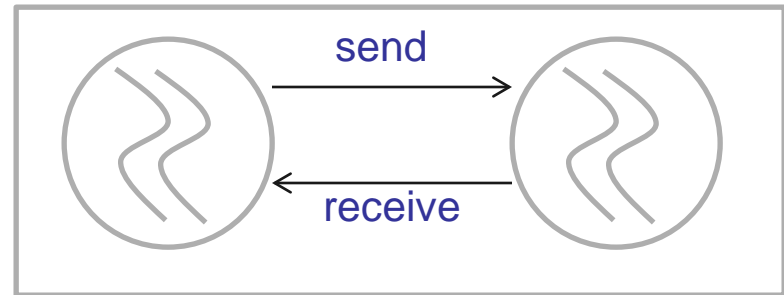


Several multi-threaded processes on several computers.



# Structuring a concurrent system

Might also structure a concurrent system running on a single machine as multiple processes that communicate with messages.



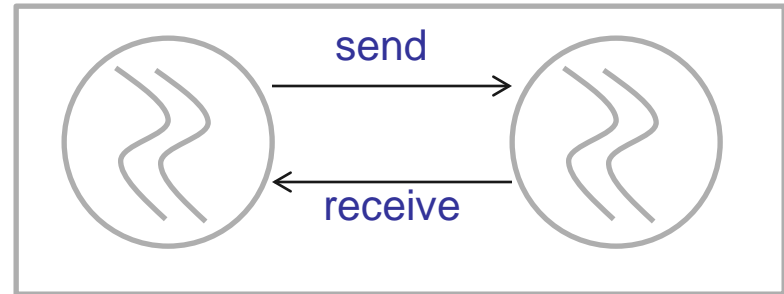
**Why would you do this?**

Better security and reliability.  
Protect modules from each other.



# Structuring a concurrent system

**Microkernels** break the OS structure into multiple server processes, each in its own address space.



# Next time ...

Distributed file system