

Introduction to Distributed Systems

SWE 622, Spring 2017

Distributed Software Engineering

Today

- Logistics + introductions
- Distributed systems: high level overview and key concepts
- Homework description and introduction
 - Time permitting, we'll step into some code
- Relevant links:
 - HW 1: <http://www.jonbell.net/swe-622-spring-2017/homework-1/>

Course Topics

- This course will teach you **how** and **why** to build distributed systems
- This course will give you theoretical knowledge of the tradeoffs that you'll face when building distributed systems
- This course will give you significant **hands-on** experience working with real distributed systems, working with well-used systems like Redis and Zookeeper

Prerequisites

- “SWE Foundation or equivalent”, AKA:
 - INFS 501 Discrete and Logical Structures for Information Systems
 - INFS 515 Computer Organization
 - INFS 519 Program Design and Data Structures
 - SWE 510 Object Oriented Programming in Java
- You need to know how to program Java
 - Awareness of threads and related synchronization issues is a big plus, but not mandatory

Logistics

- Syllabus: <http://www.jonbell.net/swe-622-spring-2017/>
 - 50% homework (we'll come back to this)
 - 20% midterm
 - 20% final
 - 10% participation
- Piazza for Q+A
- Reminders
 - Honor code
 - Late policy (10% deducted if < 24 hrs late, no credit after 24 hrs late)
 - NO extra credit

Introductions

- Prof Jonathan Bell (me)
 - Office hour: ENGR 4422 Weds 3:30-4:30 pm or by appointment; can do Google Hangouts too.
 - Areas of research: Software Engineering, Program Analysis, Software Systems

Two hobbies: cycling, ice cream

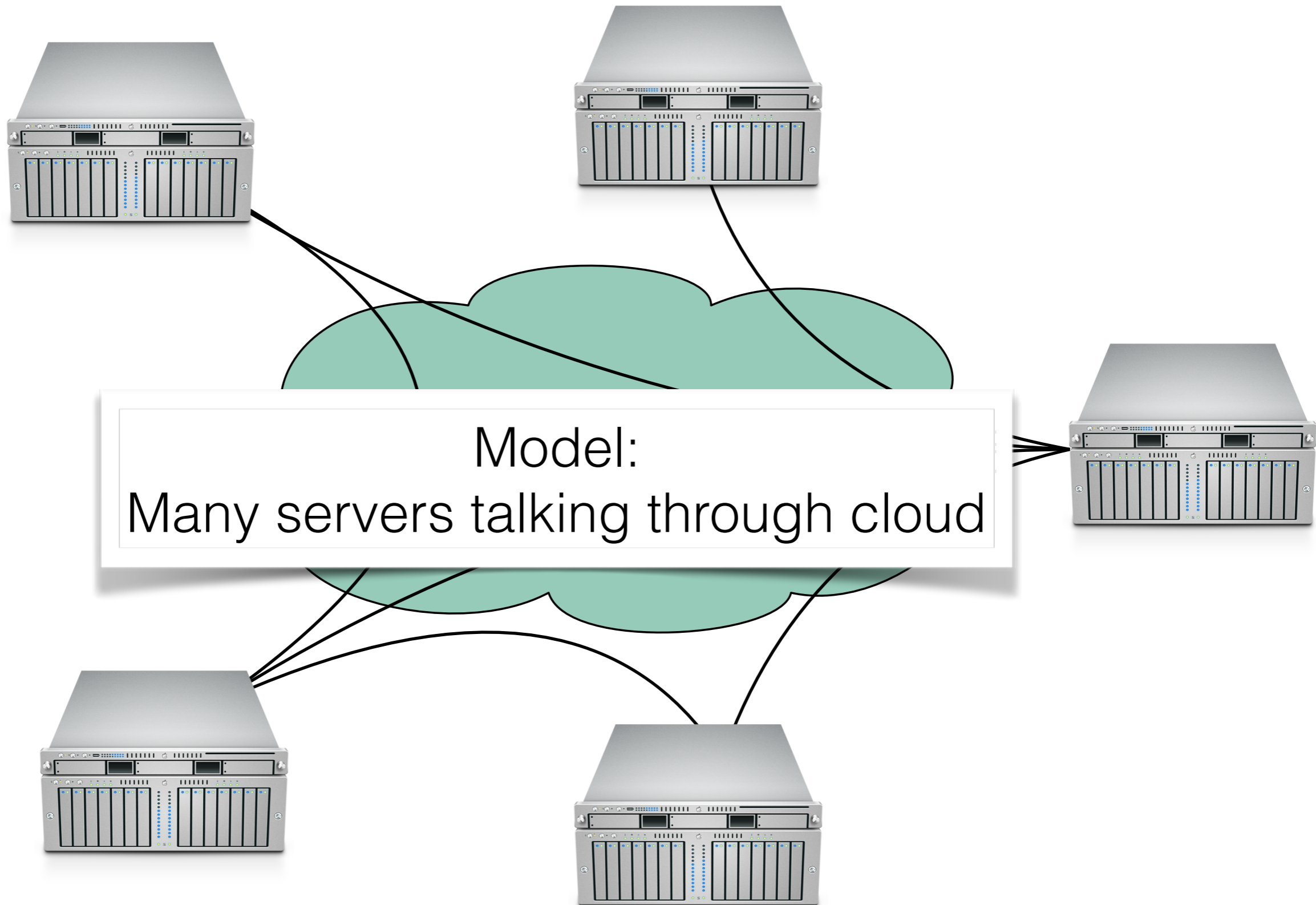


Introductions (from you!)

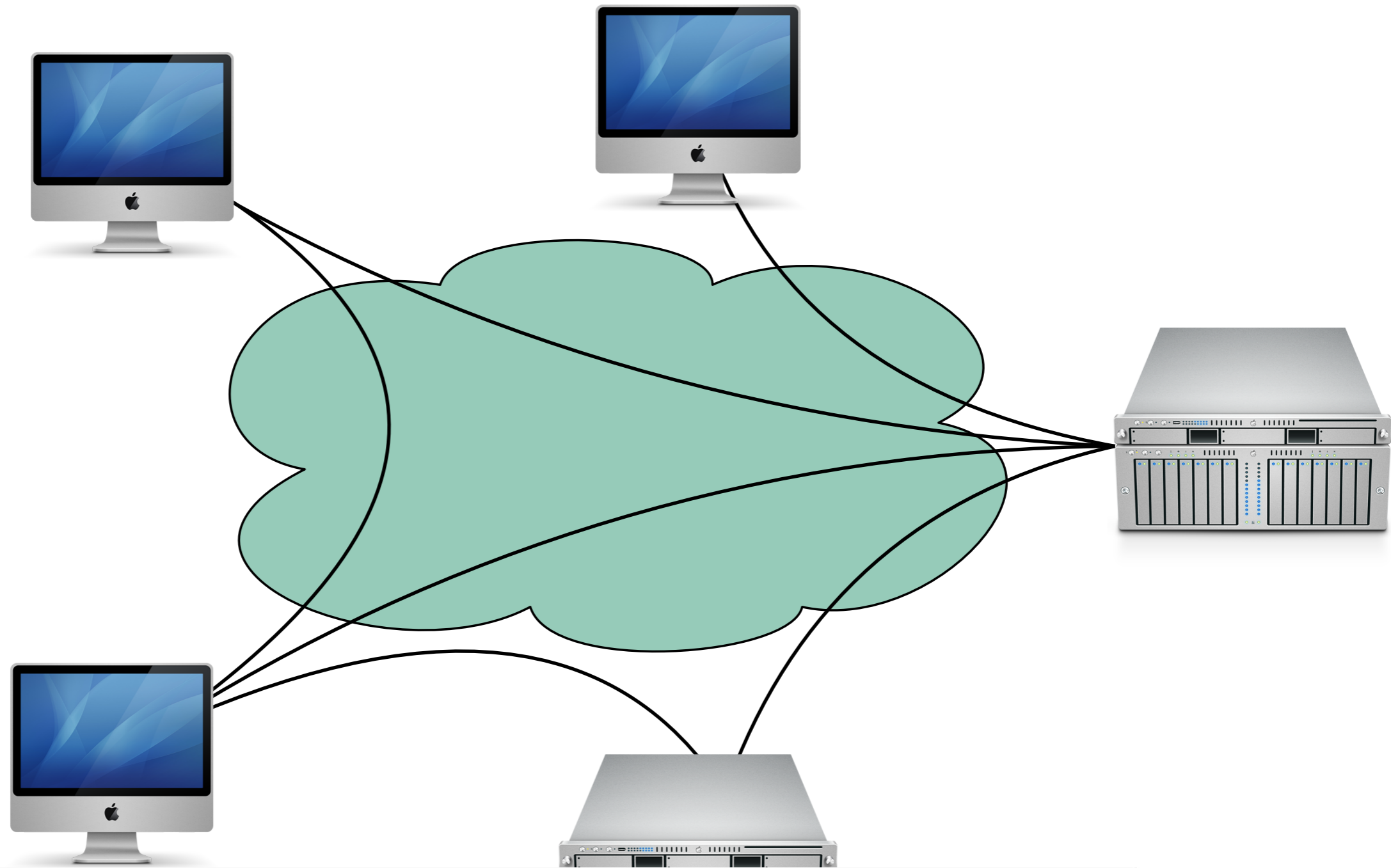
Distributed Systems

- Tannenbaum:
 - Distributed System is “a collection of independent computers that appears to its users as a single coherent system”
- Takada:
 - “Given infinite money and infinite R&D time, we wouldn't need distributed systems. All computation and storage could be done on a magic box - a single, incredibly fast and incredibly reliable system that you pay someone else to design for you.”

Distributed Systems

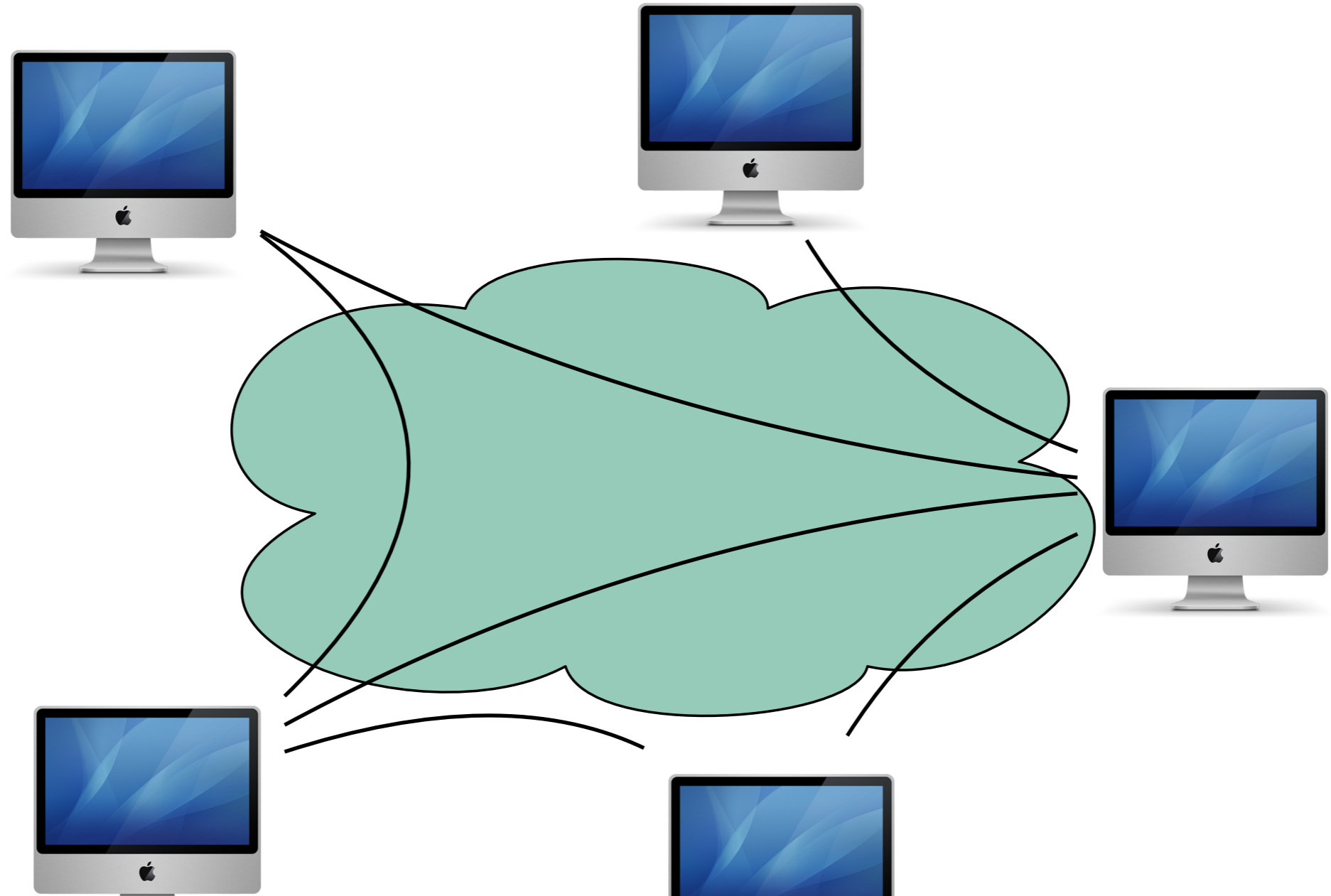


Distributed Systems



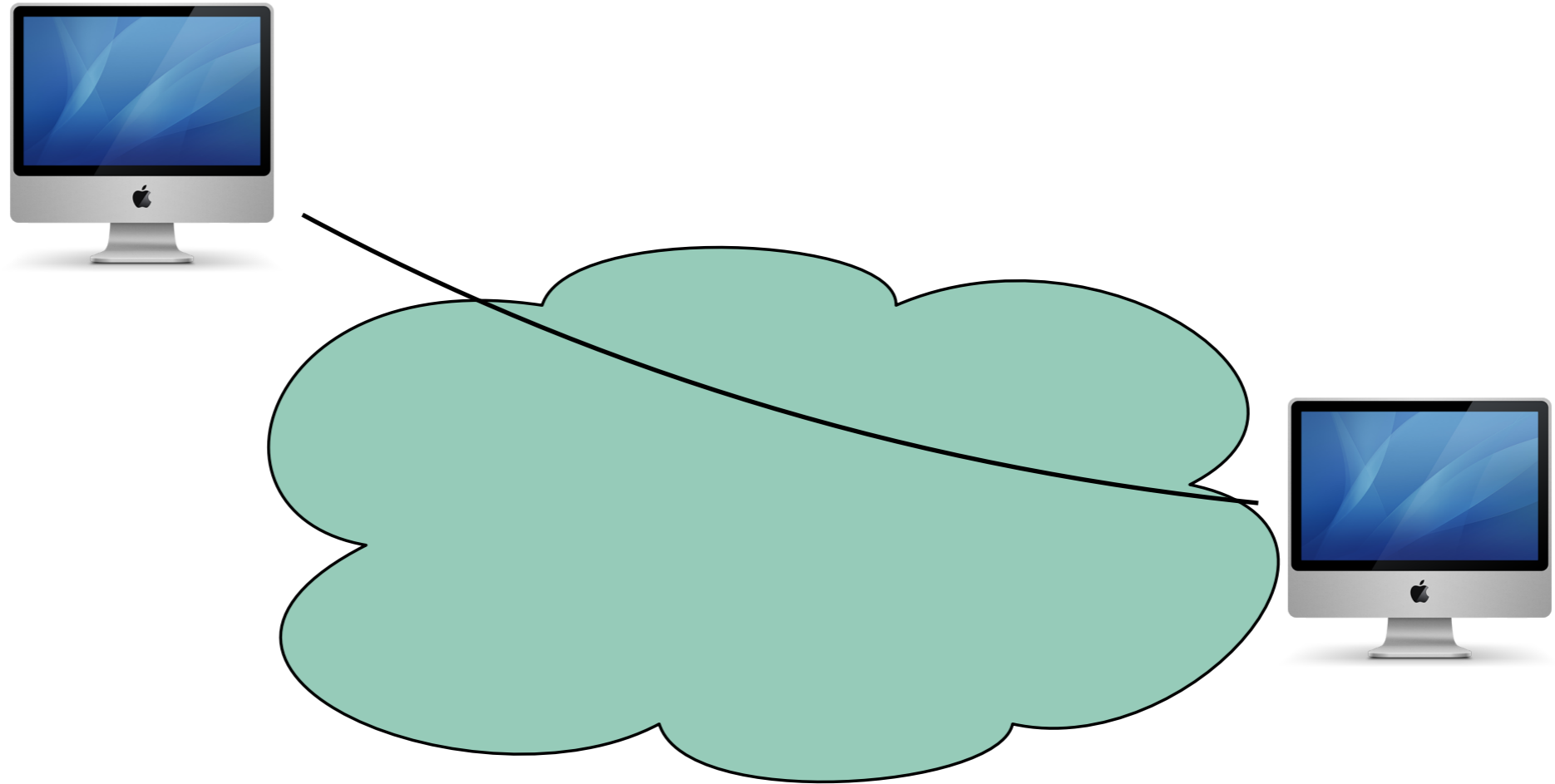
Model:
Servers and Clients talking through cloud

Distributed Systems



Model:
Many clients talking through cloud

Distributed Systems



Model:
Two clients talking through cloud

What do we want from Distributed Systems?

- Scalability
- Performance
- Latency
- Availability
- Fault Tolerance

“Distributed Systems for Fun and Profit”, Takada

Distributed Systems Goals

- **Scalability**
- Performance
- Latency
- Availability
- Fault Tolerance

“the ability of a system, network, or process, to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth.”

“Distributed Systems for Fun and Profit”, Takada

Distributed Systems Goals

- Scalability
- **Performance**
- Latency
- Availability
- Fault Tolerance

“is characterized by the amount of useful work accomplished by a computer system compared to the time and resources used.”

Distributed Systems Goals

- Scalability
- Performance
- **Latency**
- Availability
- Fault Tolerance

“The state of being latent; delay, a period between the initiation of something and the it becoming visible.”

Distributed Systems Goals

- Scalability
- Performance
- Latency
- **Availability**
- Fault Tolerance

“the proportion of time a system is in a functioning condition. If a user cannot access the system, it is said to be unavailable.”

Availability = uptime / (uptime + downtime).

Often measured in “nines”

Availability %	Downtime/year
90%	>1 month
99%	< 4 days
99.9%	< 9 hours
99.99%	<1 hour
99.999%	5 minutes
99.9999%	31 seconds

Distributed Systems Goals

- Scalability
- Performance
- Latency
- Availability
- **Fault Tolerance**

“ability of a system to behave in a well-defined manner once faults occur”

What kind of faults?

Disks fail

Power supplies fail

Networking fails

Security breached

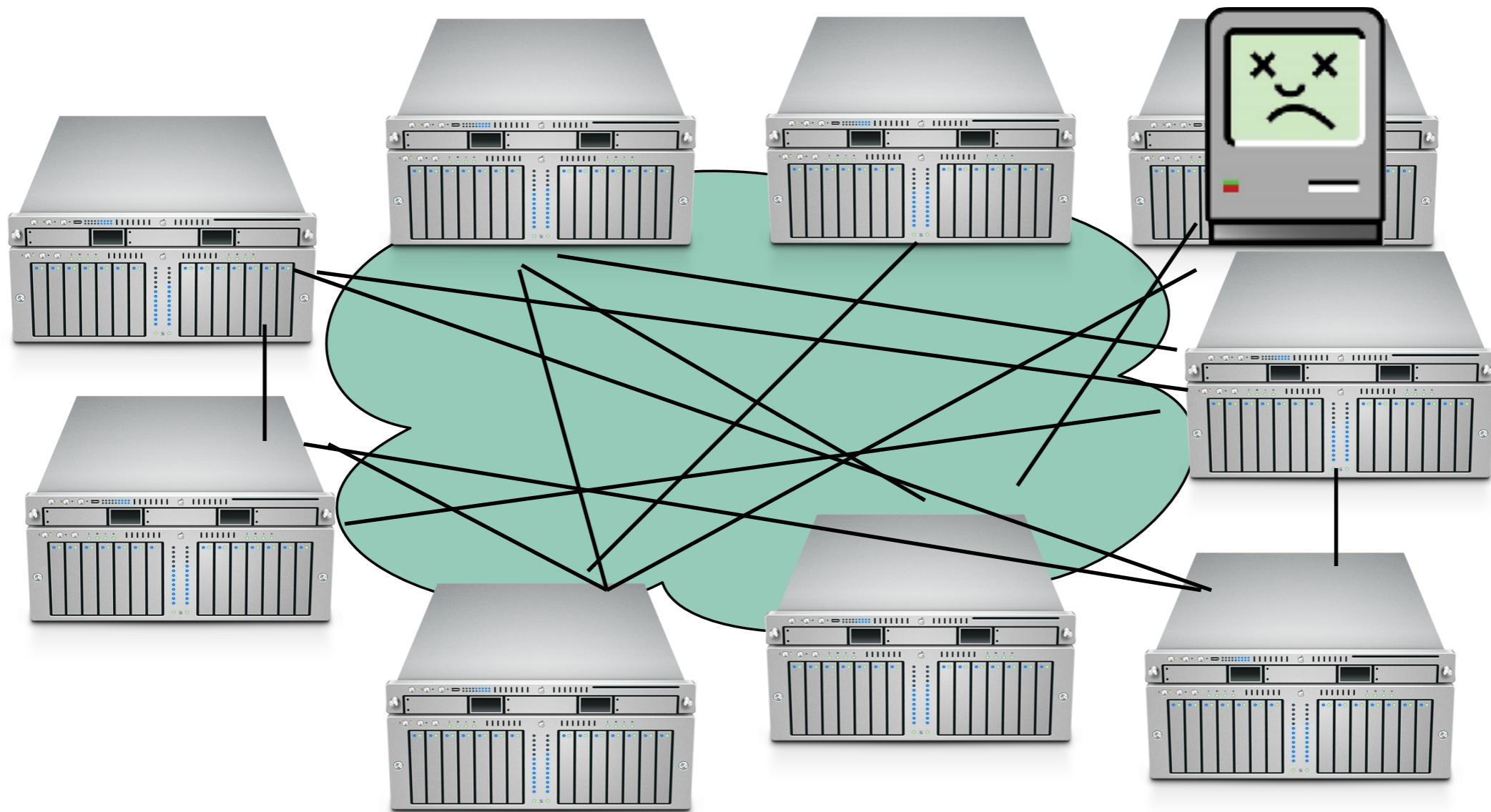
Power goes out Datacenter goes offline

More machines, more problems

- Say there's a 1% chance of having some hardware failure occur to a machine (power supply burns out, hard disk crashes, etc)
- Now I have 10 machines
 - Probability(at least one fails) = 1 - Probability(no machine fails) = $1 - (1 - .01)^{10} = 10\%$
- 100 machines -> 63%
- 200 machines -> 87%
- So obviously just adding more machines doesn't solve fault tolerance

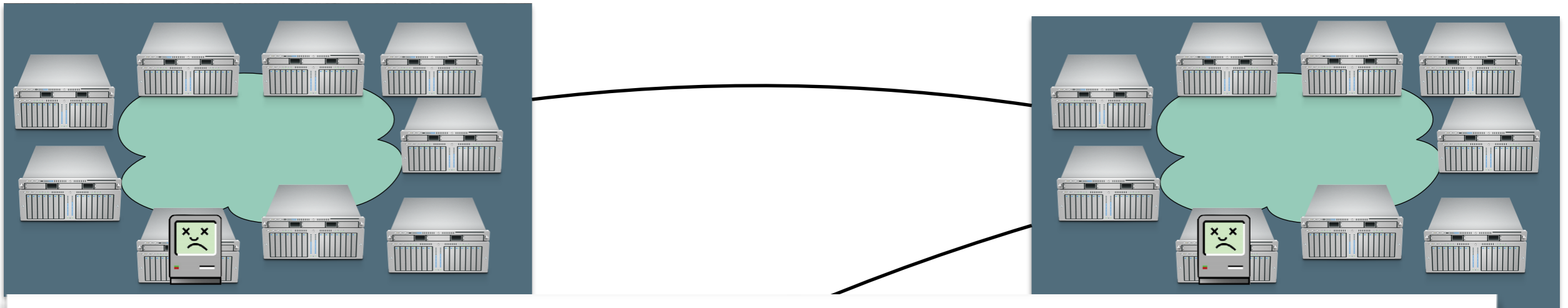
Constraints

- Number of nodes
- Distance between nodes

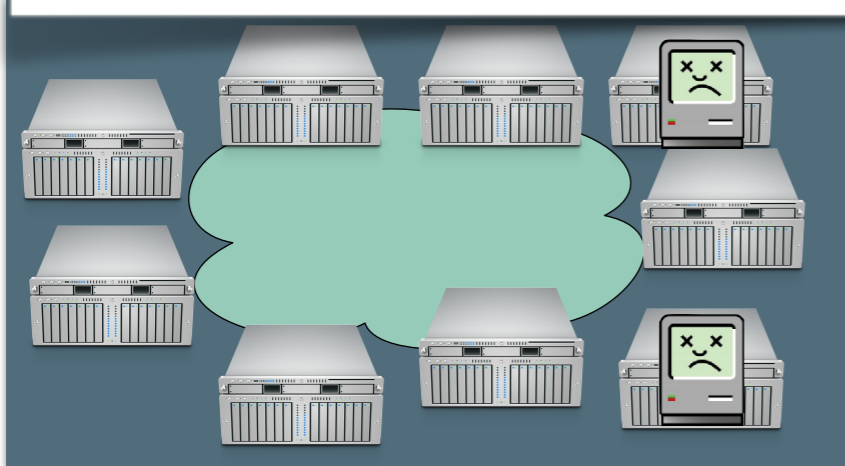


Constraints

- Number of nodes
- Distance between nodes



Even if cross-city links are fast and cheap (are they?)
Still that pesky speed of light...

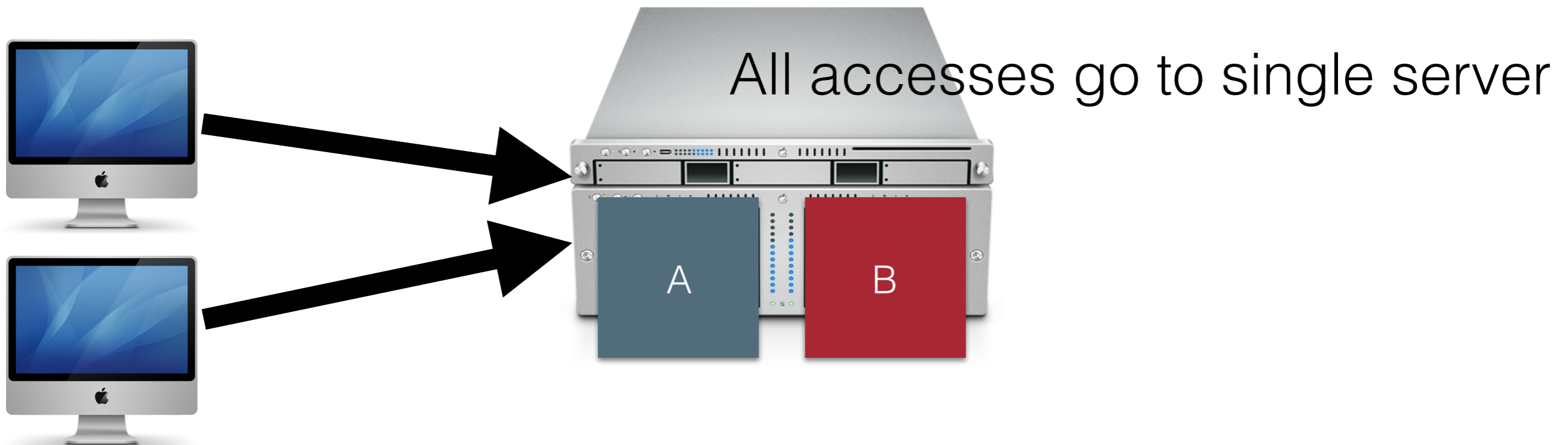


DC



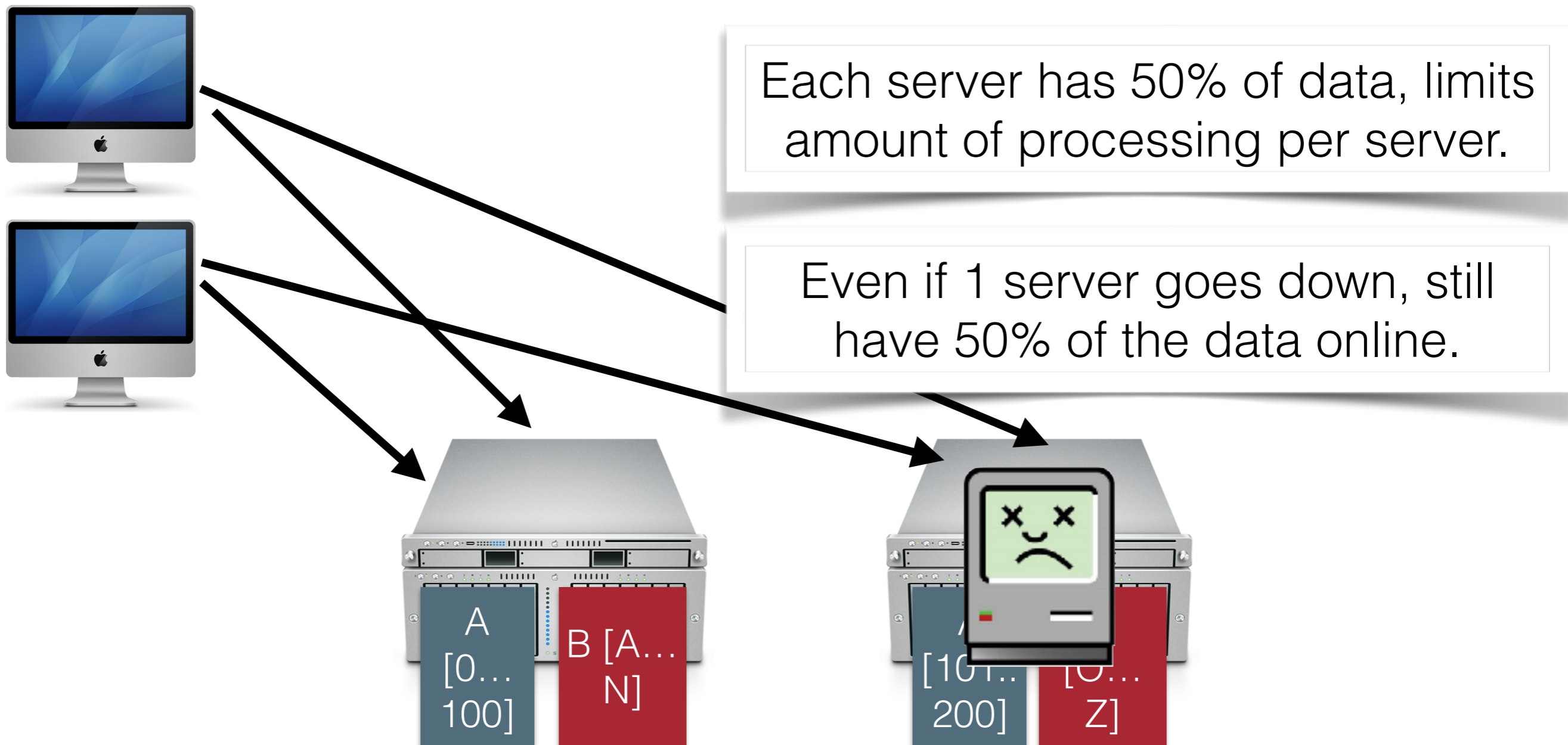
LONDON

Recurring Solution #1: Partitioning

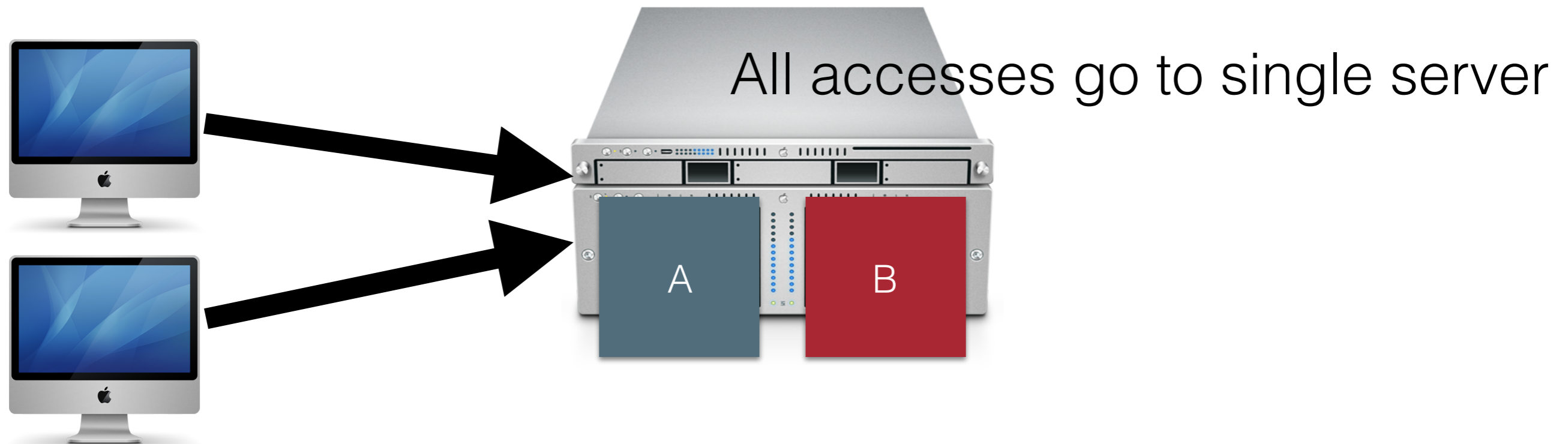


Recurring Solution #1: Partitioning

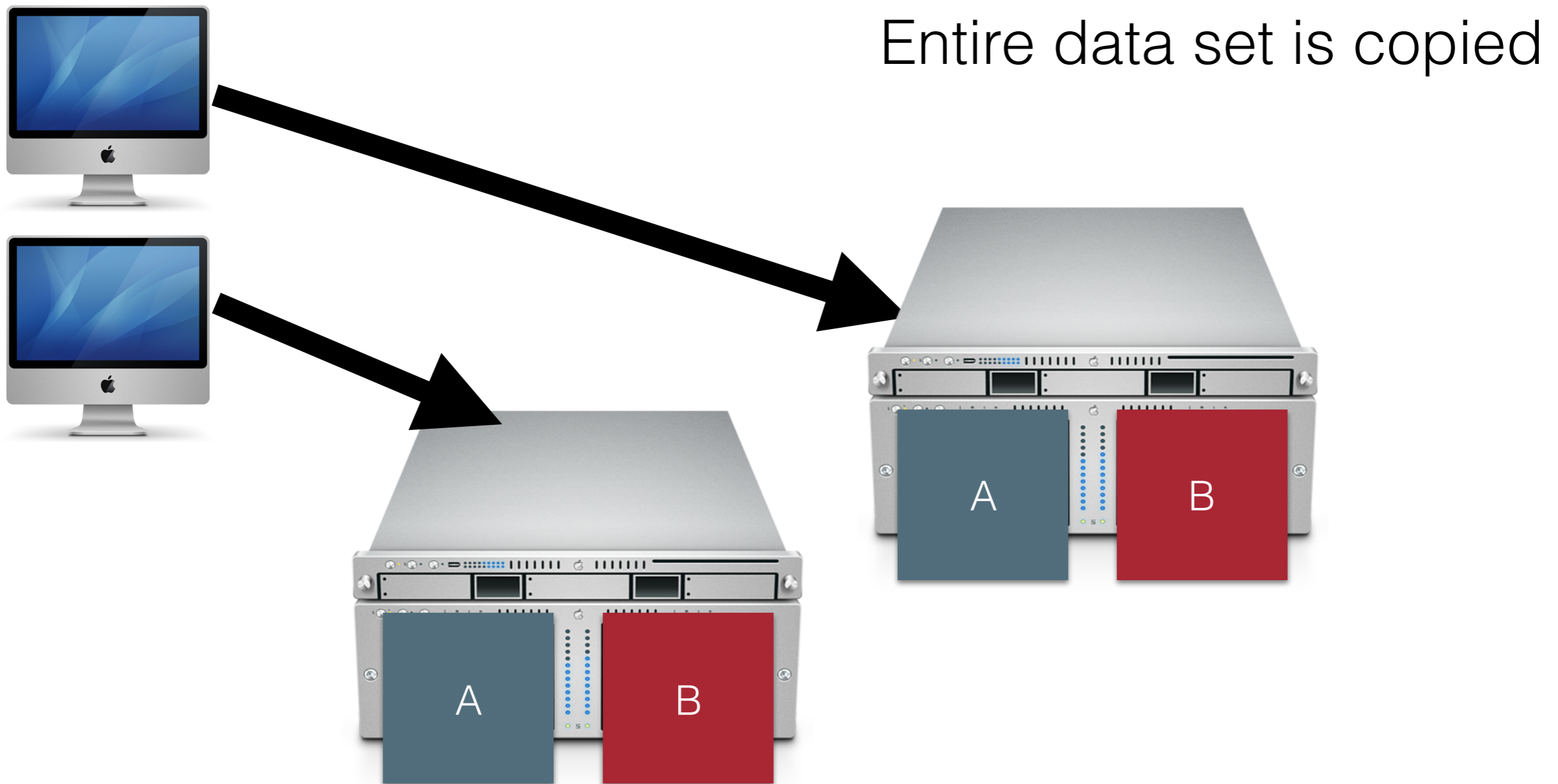
- Divide data up in some (hopefully logical) way
- Makes it easier to process data concurrently (cheaper reads)



Recurring Solution #2: Replication



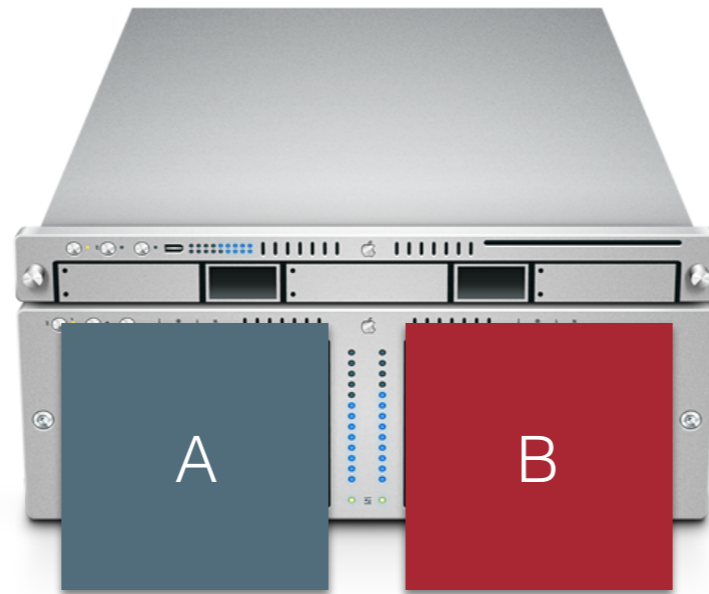
Recurring Solution #2: Replication



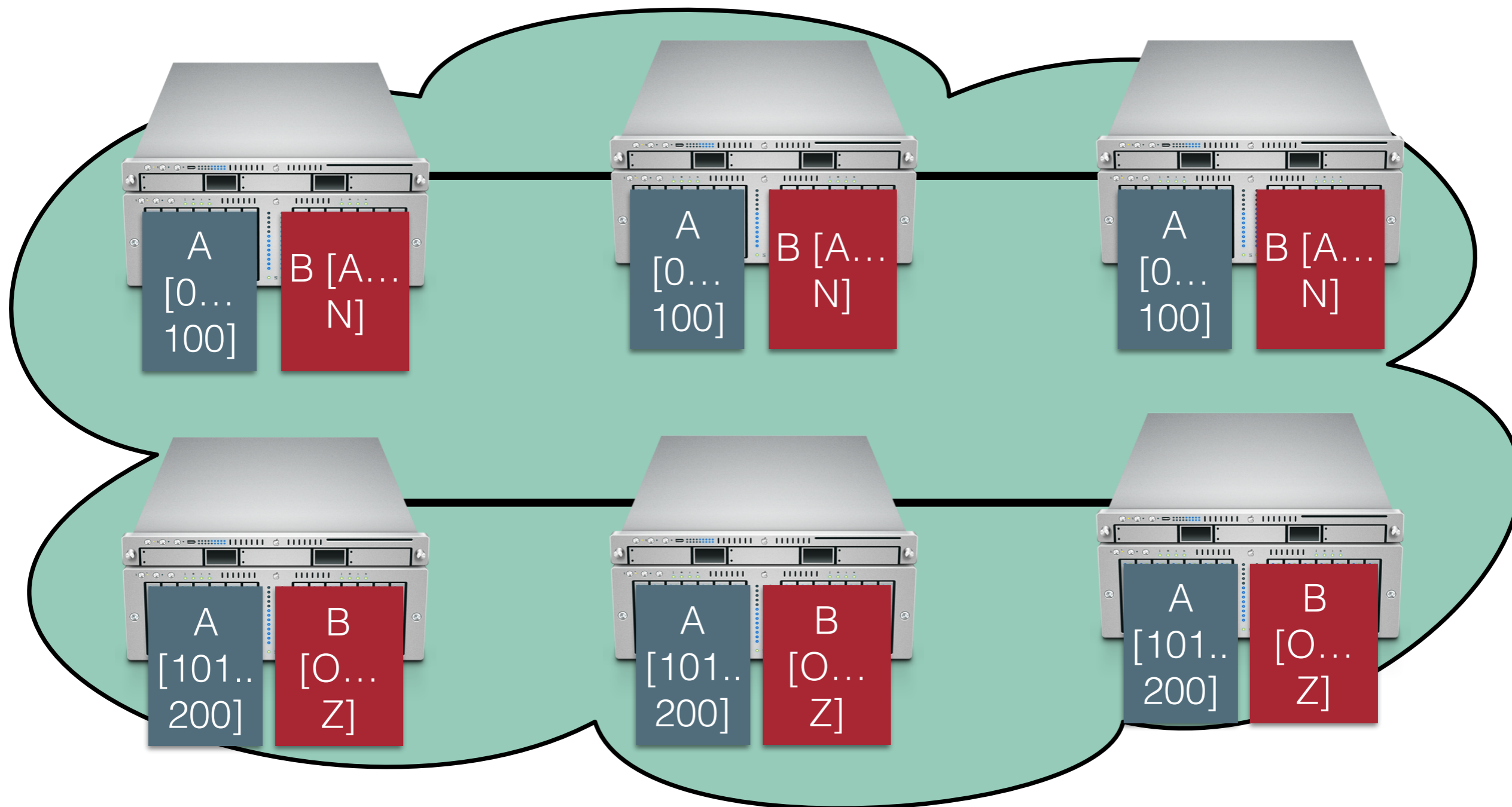
Recurring Solution #2: Replication

- Improves performance:
 - Client load can be evenly shared between servers
 - Reduces latency: can place copies of data nearer to clients
- Improves availability:
 - One replica fails, still can serve all requests from other replicas

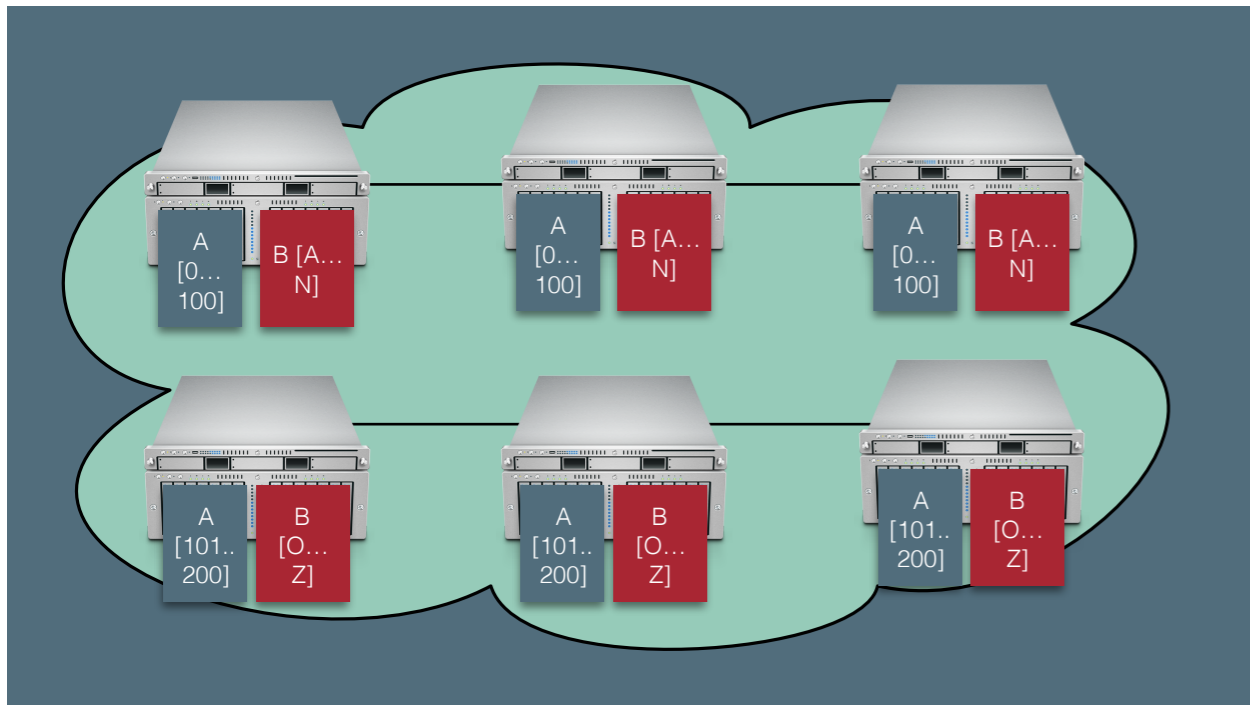
Partitioning + Replication



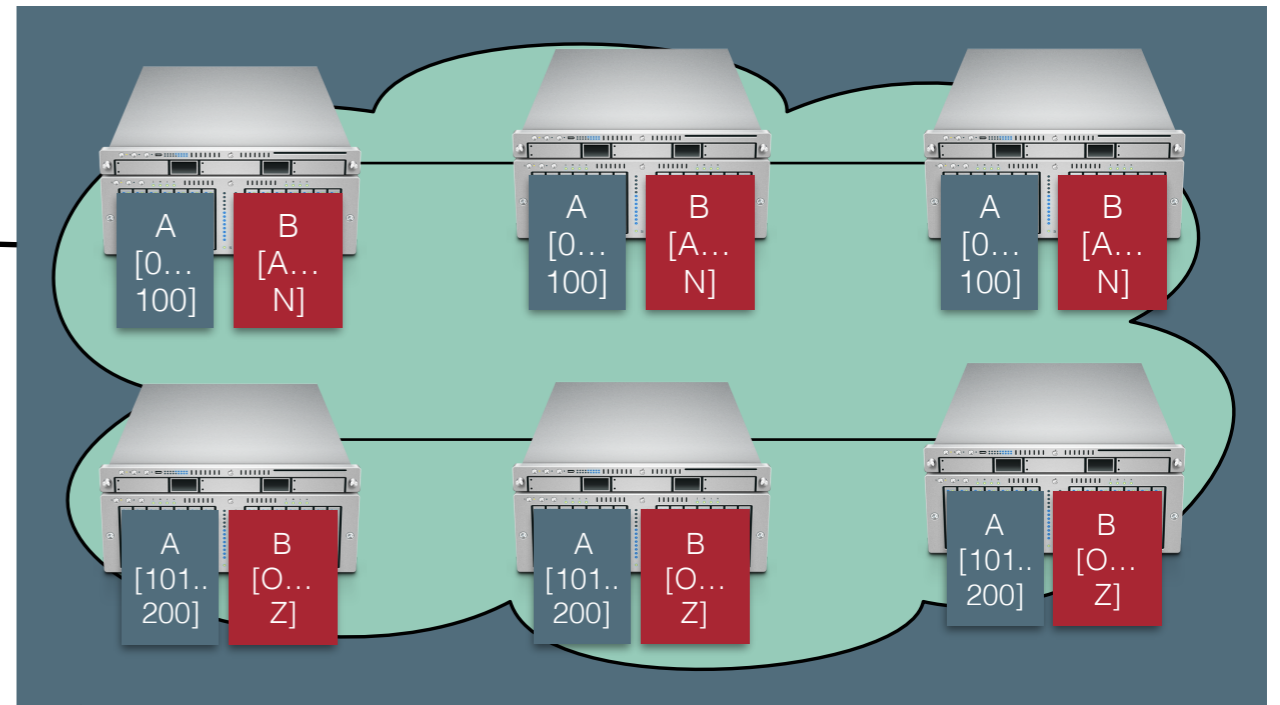
Partitioning + Replication



Partitioning + Replication



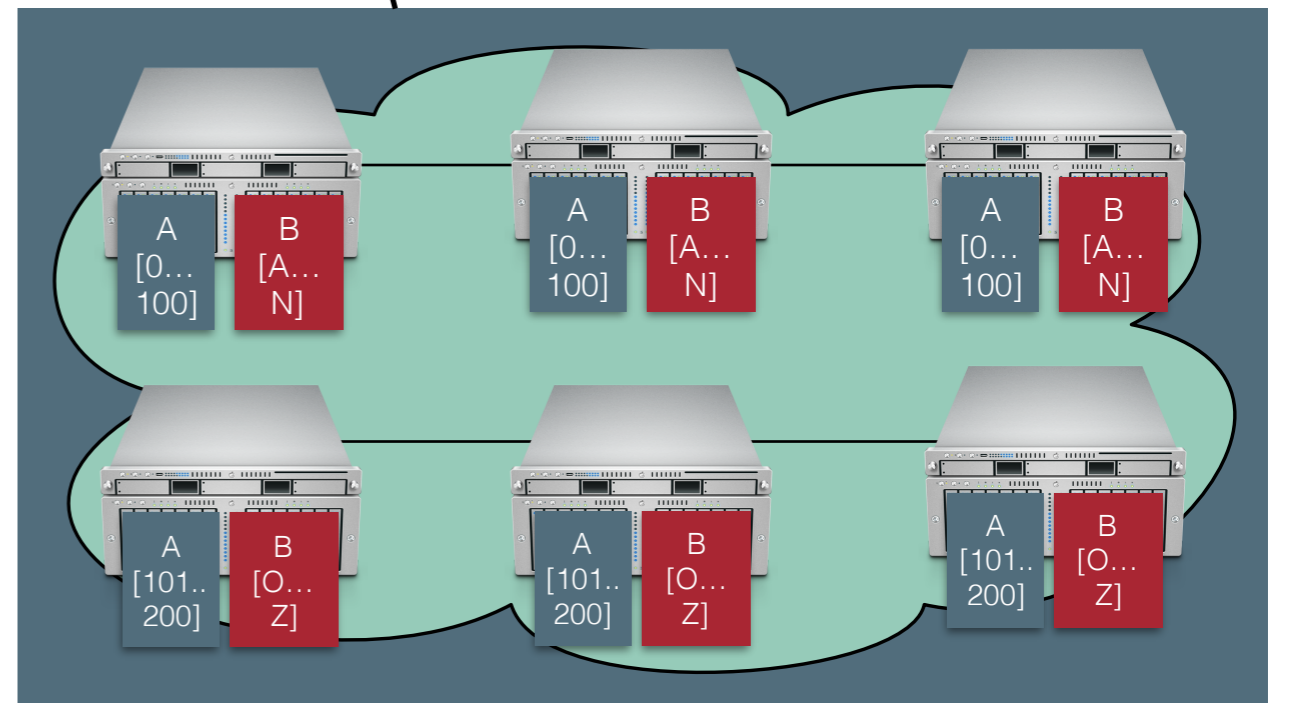
DC



NYC



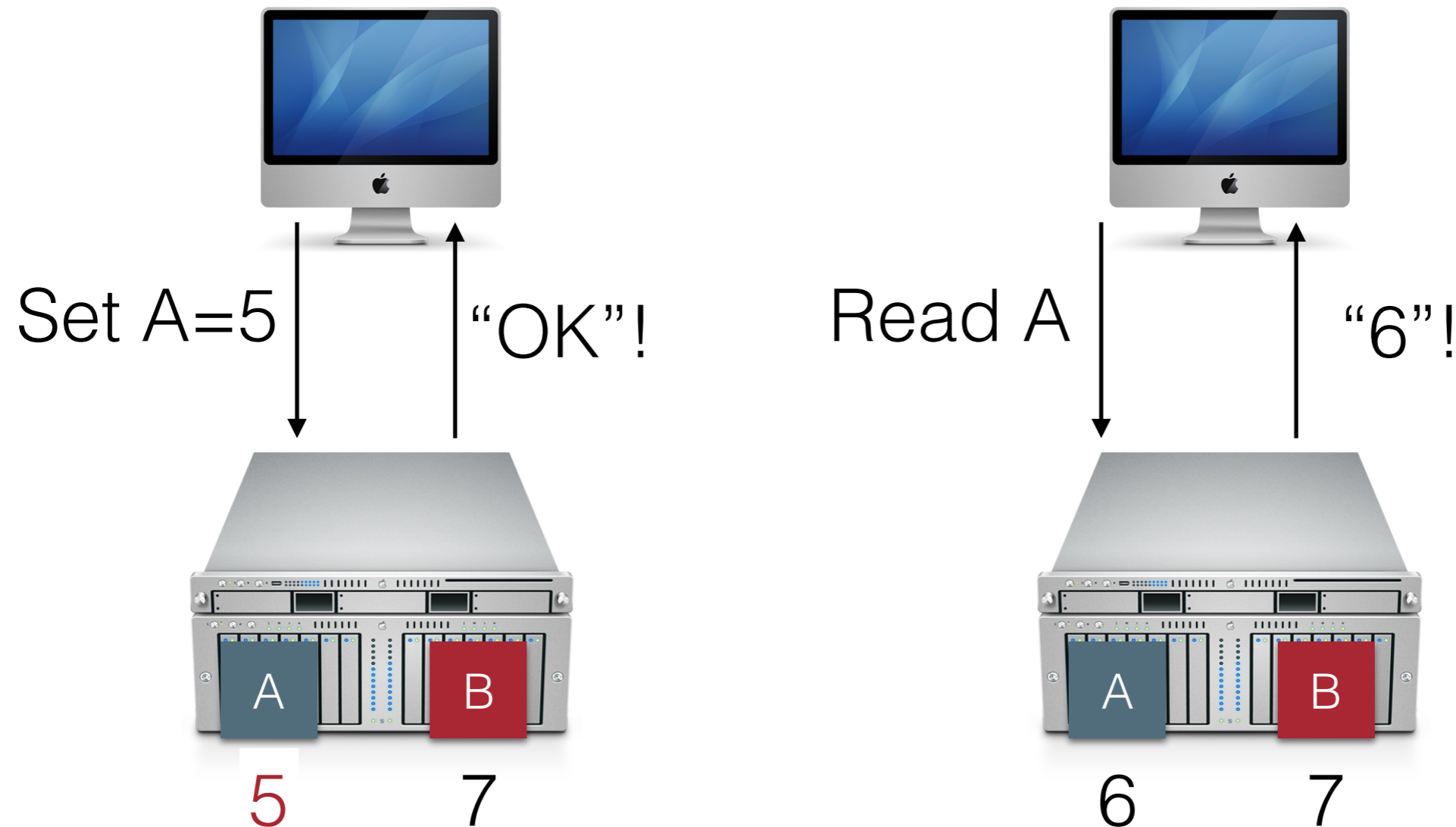
SF



London

Recurring Problem: Replication

- Replication solves some problems, but creates a huge new one: consistency

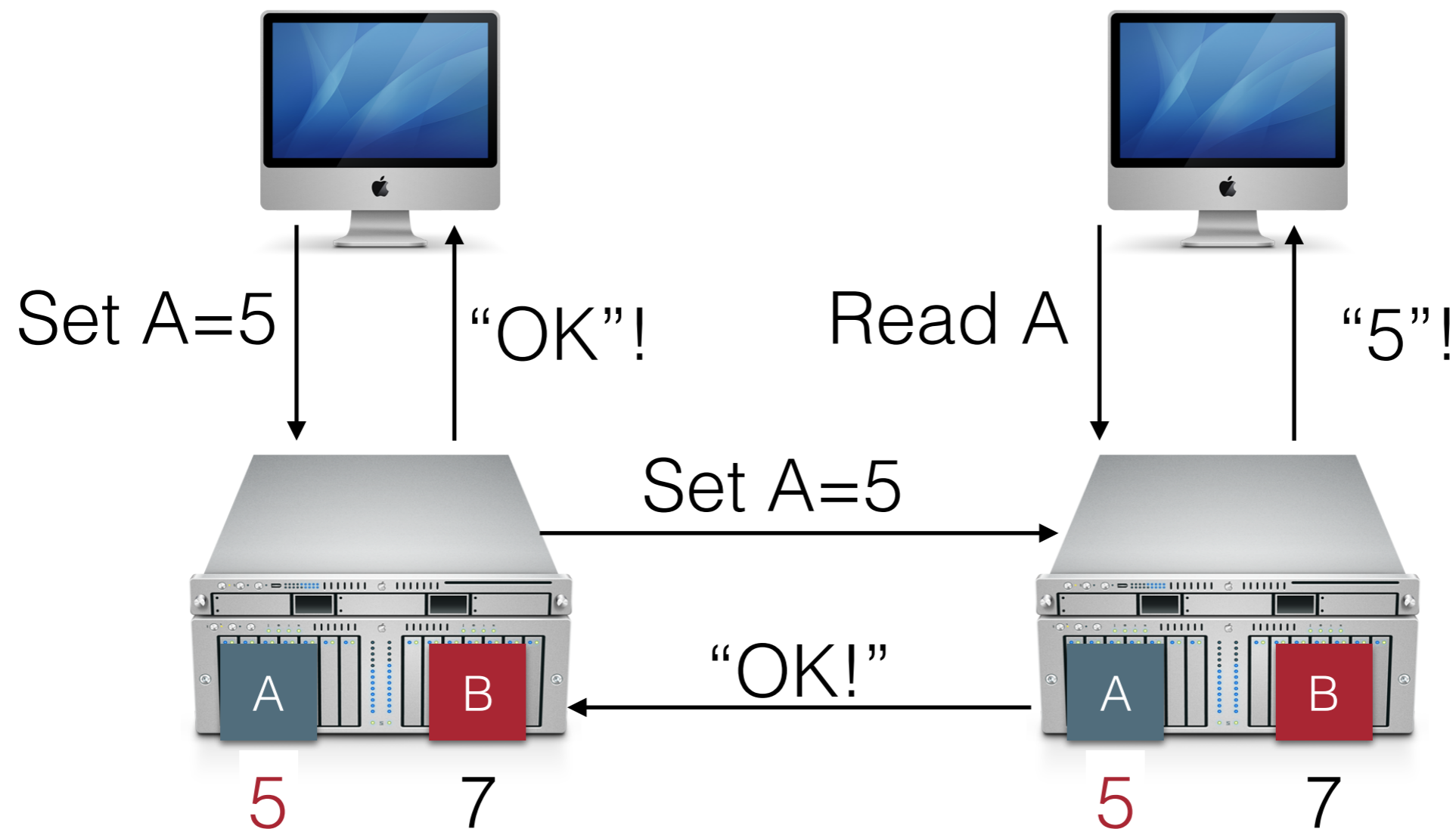


OK, we obviously need to actually do something here to replicate the data... but what?

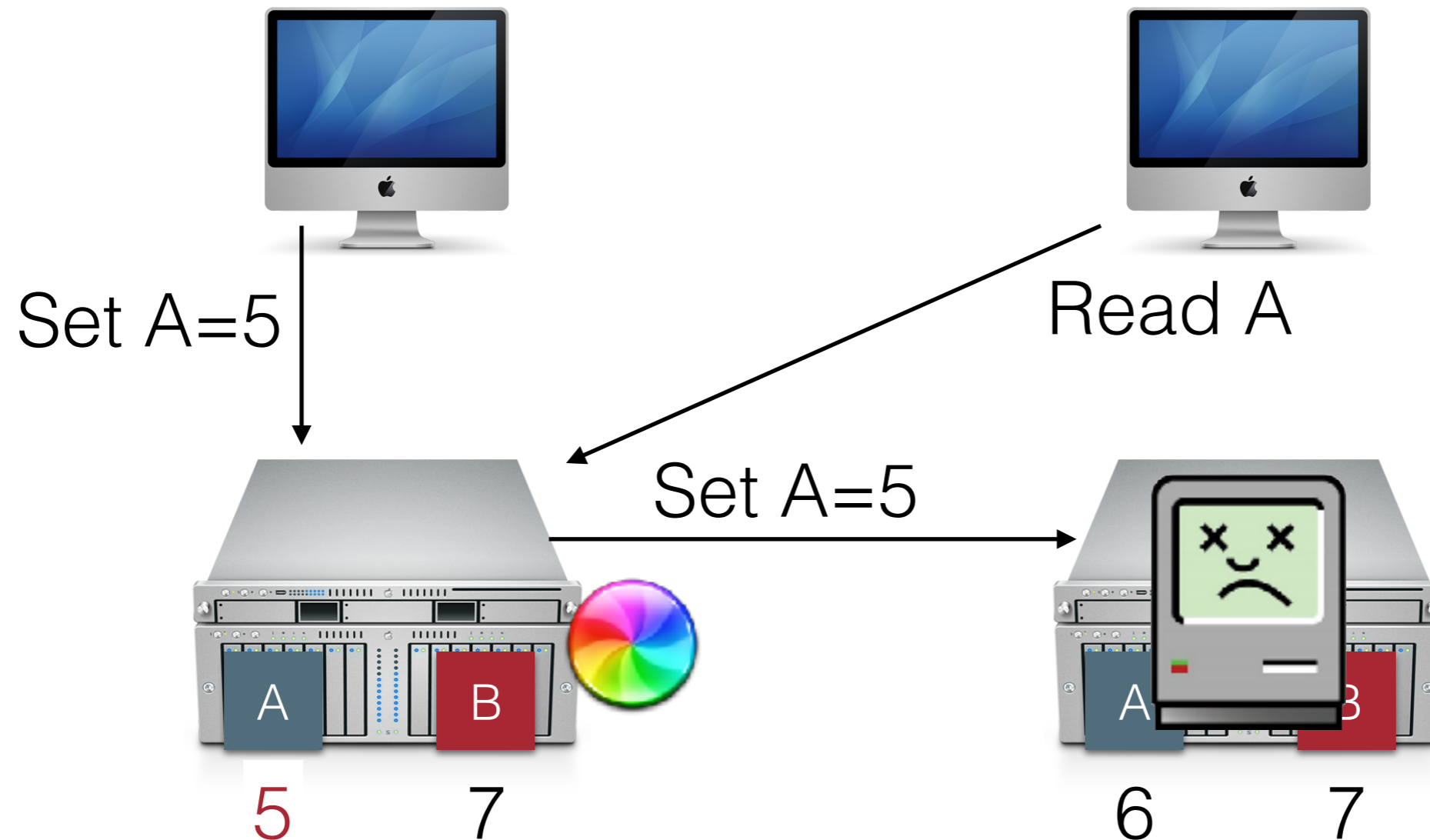
Replication

- Was it OK for the replicas to be out of sync?
- When they diverge, we say that they are not *consistent*
- What is consistent?
- For now, let's talk about *sequential* consistency, which will *guarantee* that the data updates exactly as if there were no replication

Sequential Consistency

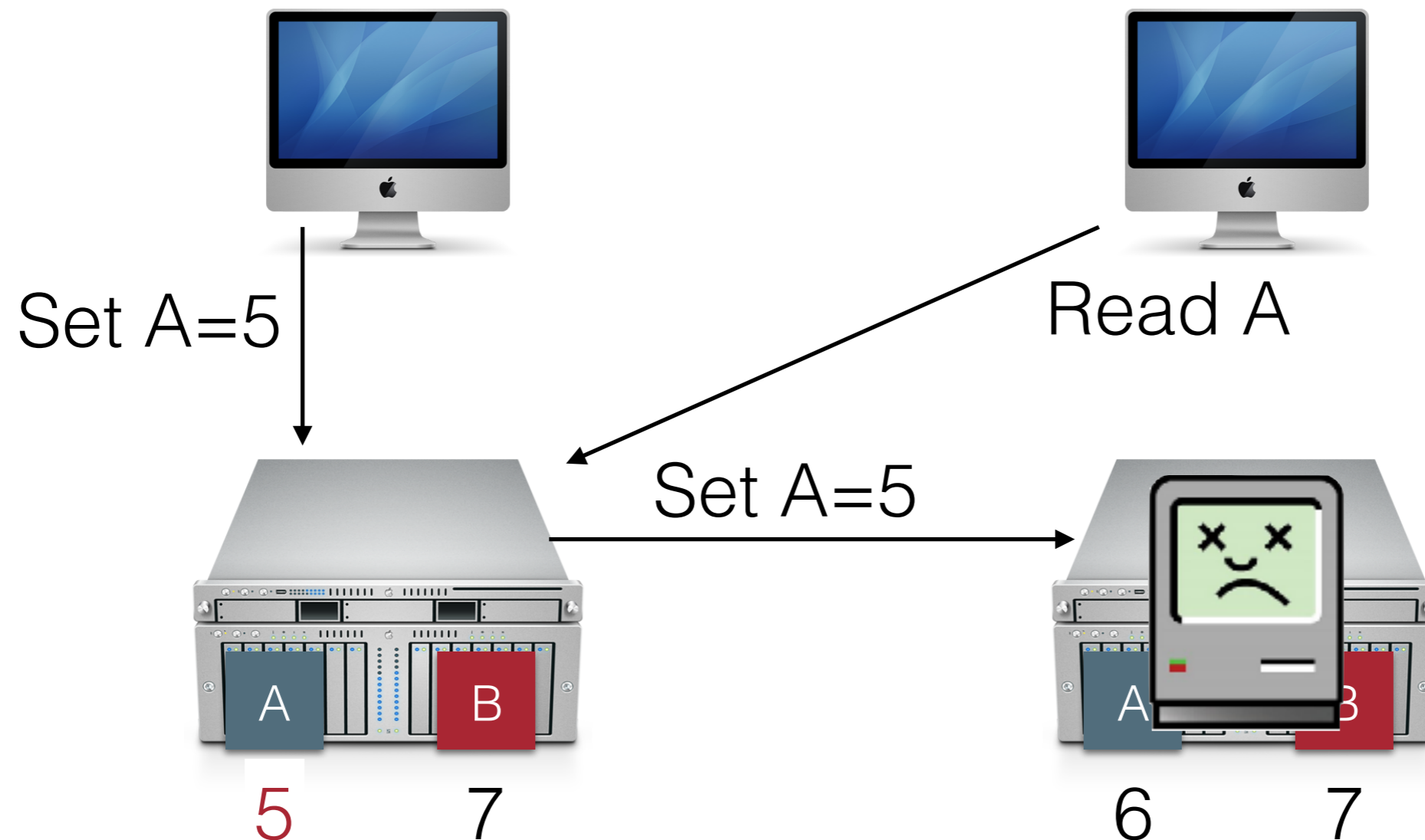


Broken Sequential Consistency

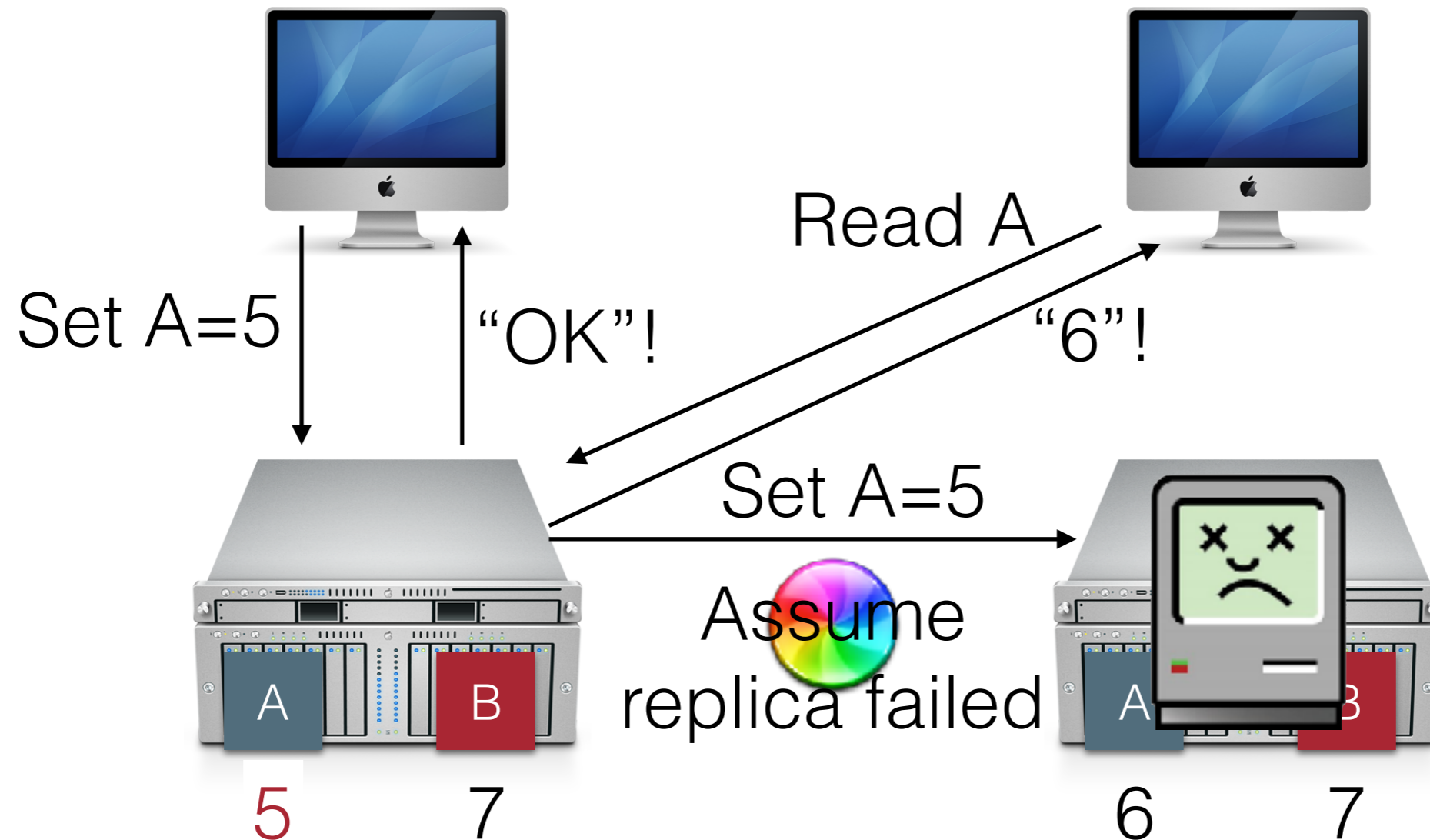


Availability

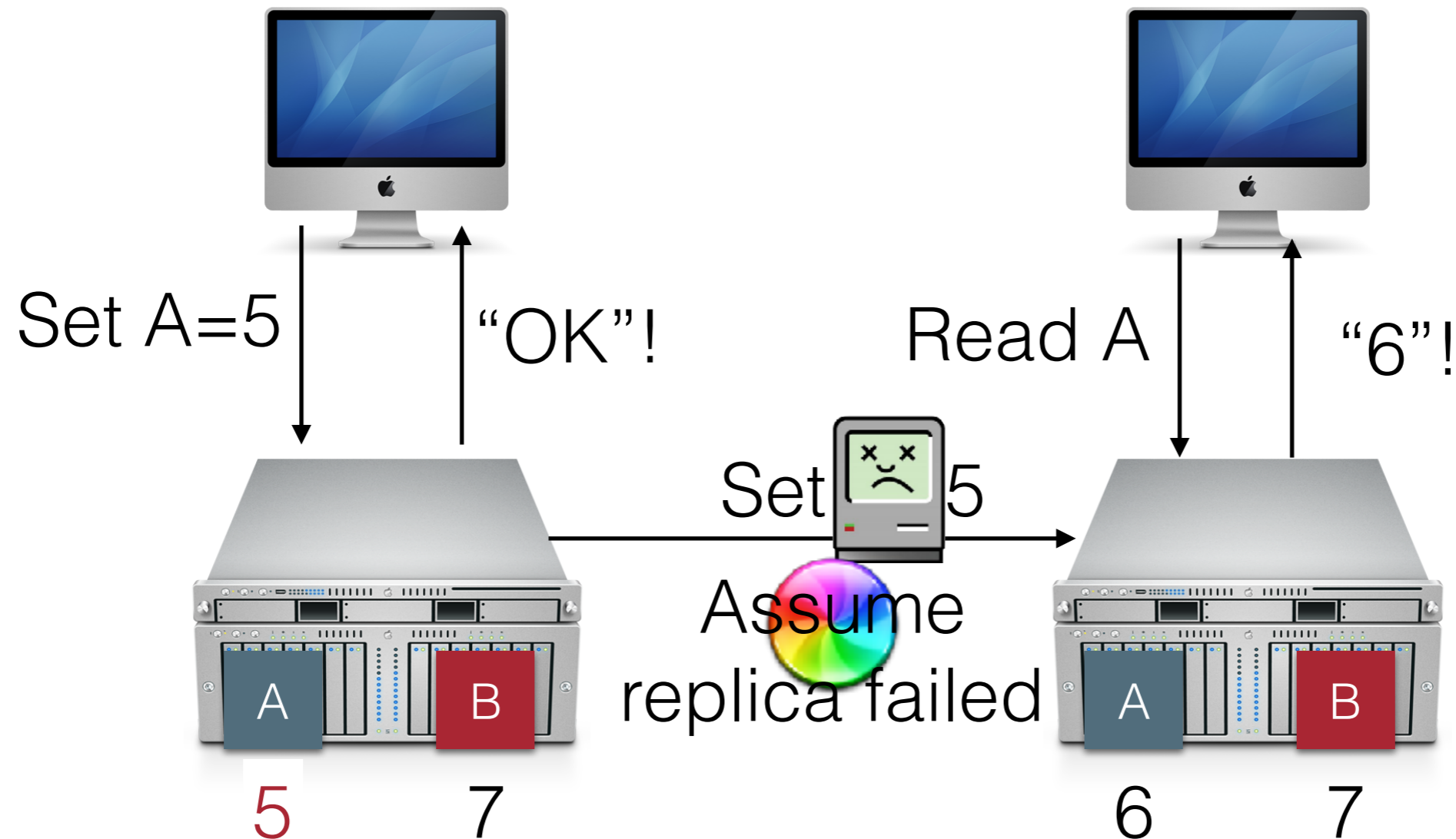
- Our protocol for sequential consistency does NOT guarantee that the system will be available!



Consistent + Available

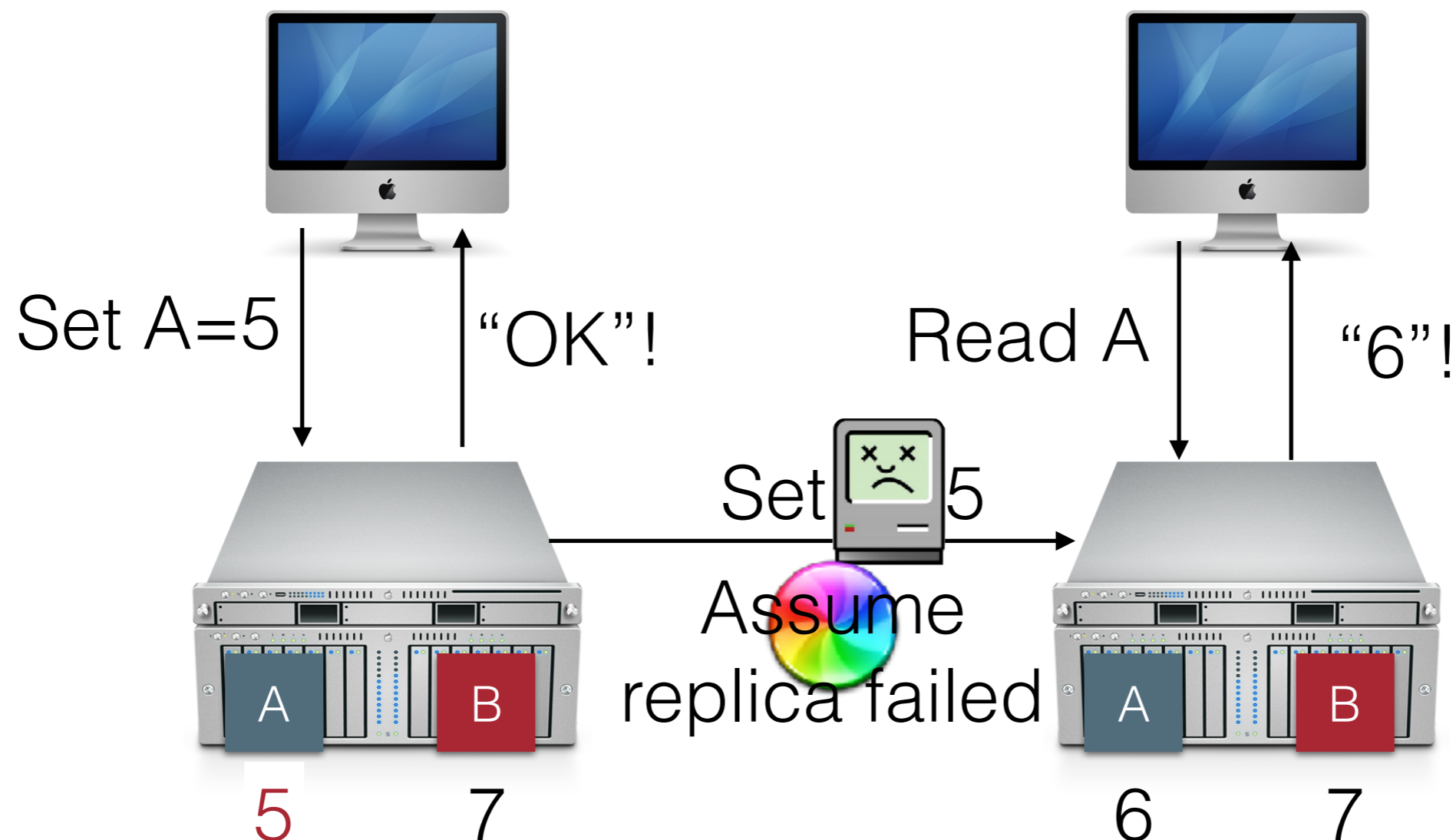


Still broken...



Network Partitions

- The communication links between nodes may fail arbitrarily
- But other nodes might still be able to reach that node



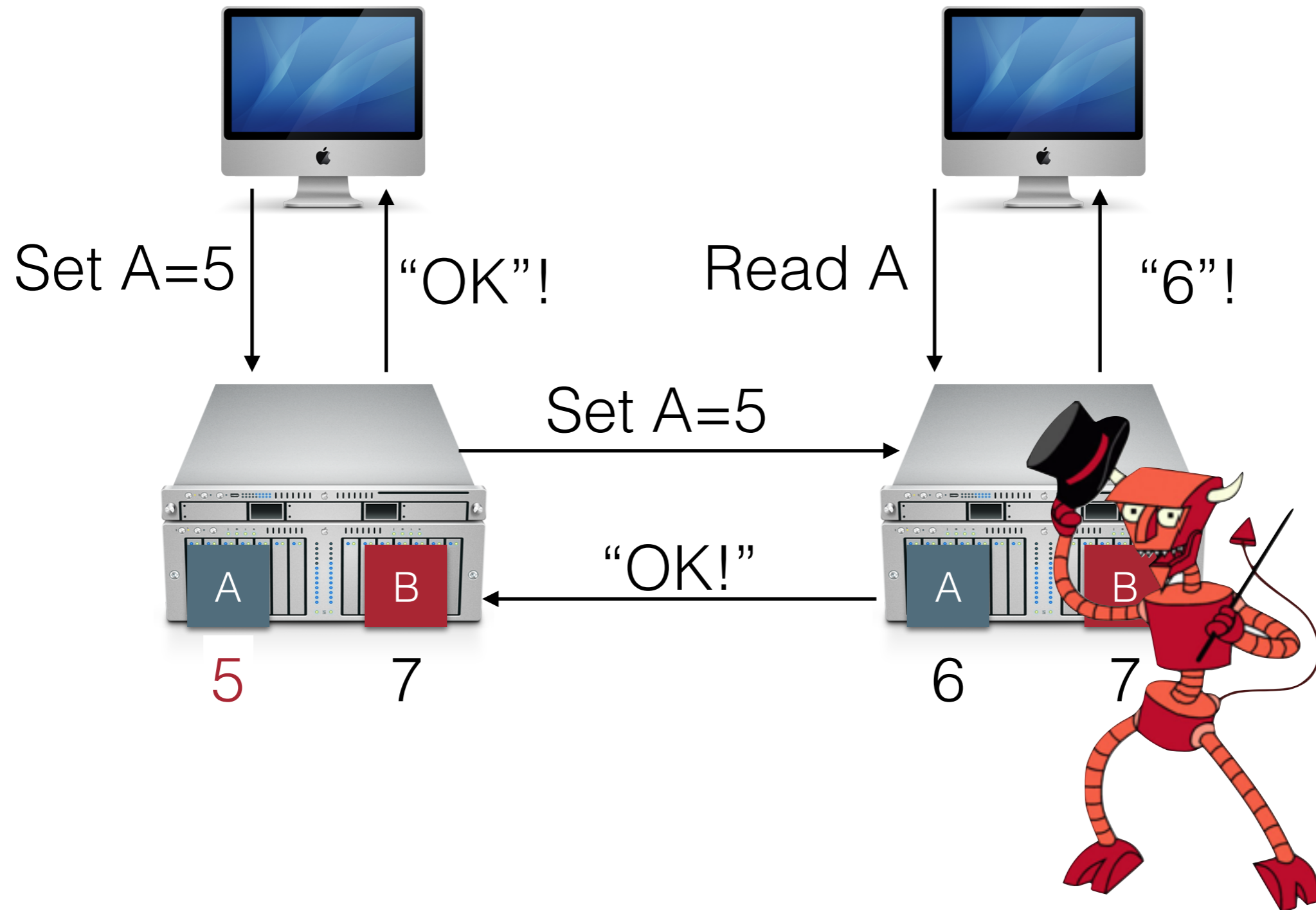
CAP Theorem

- Pick two of three:
 - Consistency: All nodes see the same data at the same time (strong consistency)
 - Availability: Individual node failures do not prevent survivors from continuing to operate
 - Partition tolerance: The system continues to operate despite message loss (from network and/or node failure)
- **You can not have all three, ever***
 - If you relax your consistency guarantee (we'll talk about in a few weeks), you might be able to guarantee THAT...

CAP Theorem

- C+A: Provide strong consistency and availability, assuming there are no network partitions
- C+P: Provide strong consistency in the presence of network partitions; minority partition is unavailable
- A+P: Provide availability even in presence of partitions; no strong consistency guarantee

Still broken...



Byzantine Failures

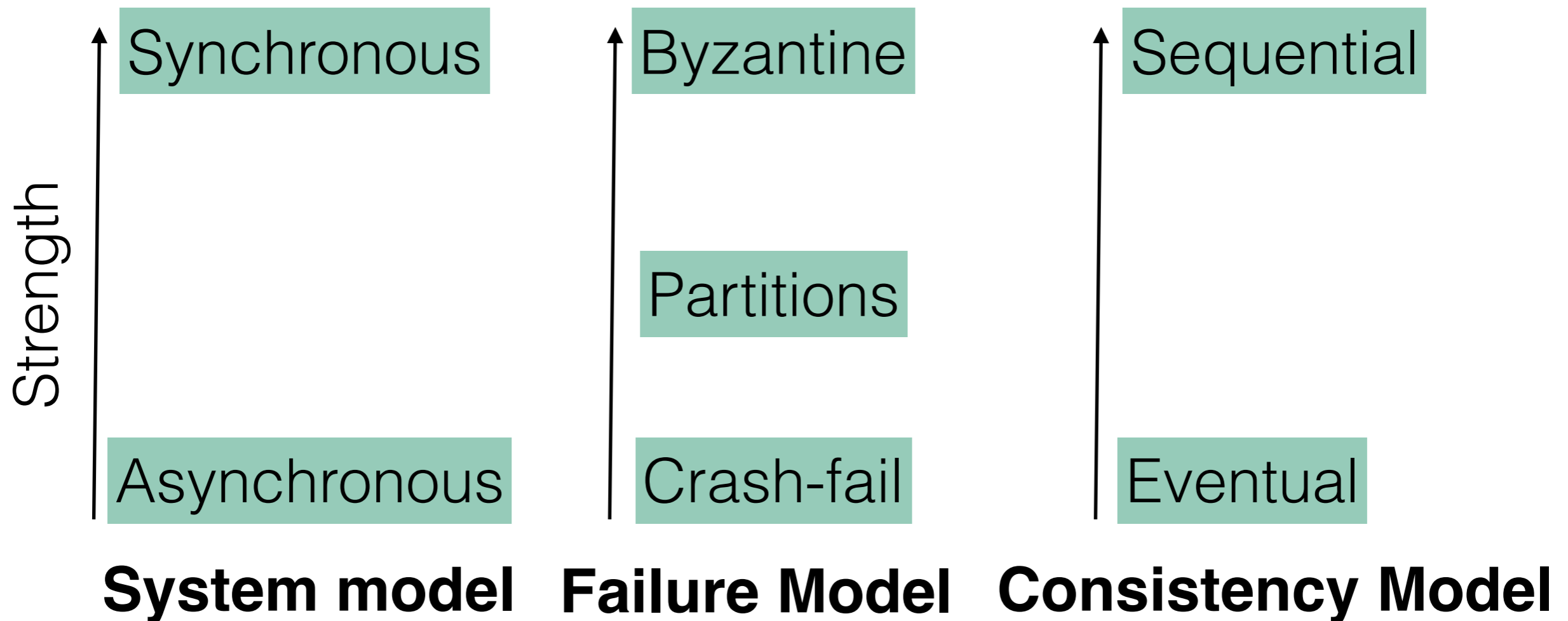
- We typically assume that we can control how our actors behave
- But perhaps, some begin to behave arbitrarily
- Generally, this is very very hard (computationally) to control for, and is often ignored in real systems

Designing and Building Distributed Systems

To help design our algorithms and systems, we tend to leverage abstractions and models to make assumptions

Generally: Stronger assumptions -> worse performance

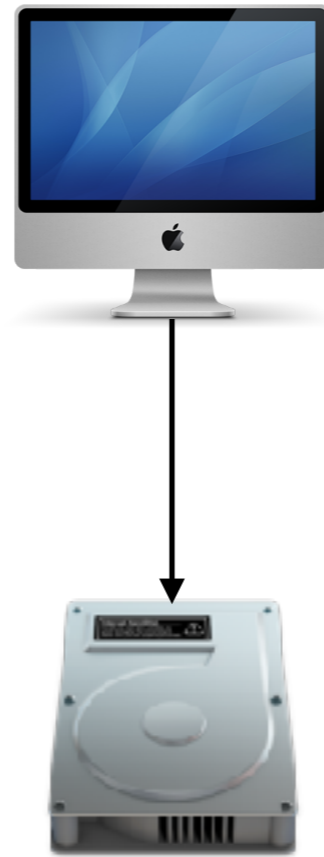
Weaker assumptions -> more complicated



Review

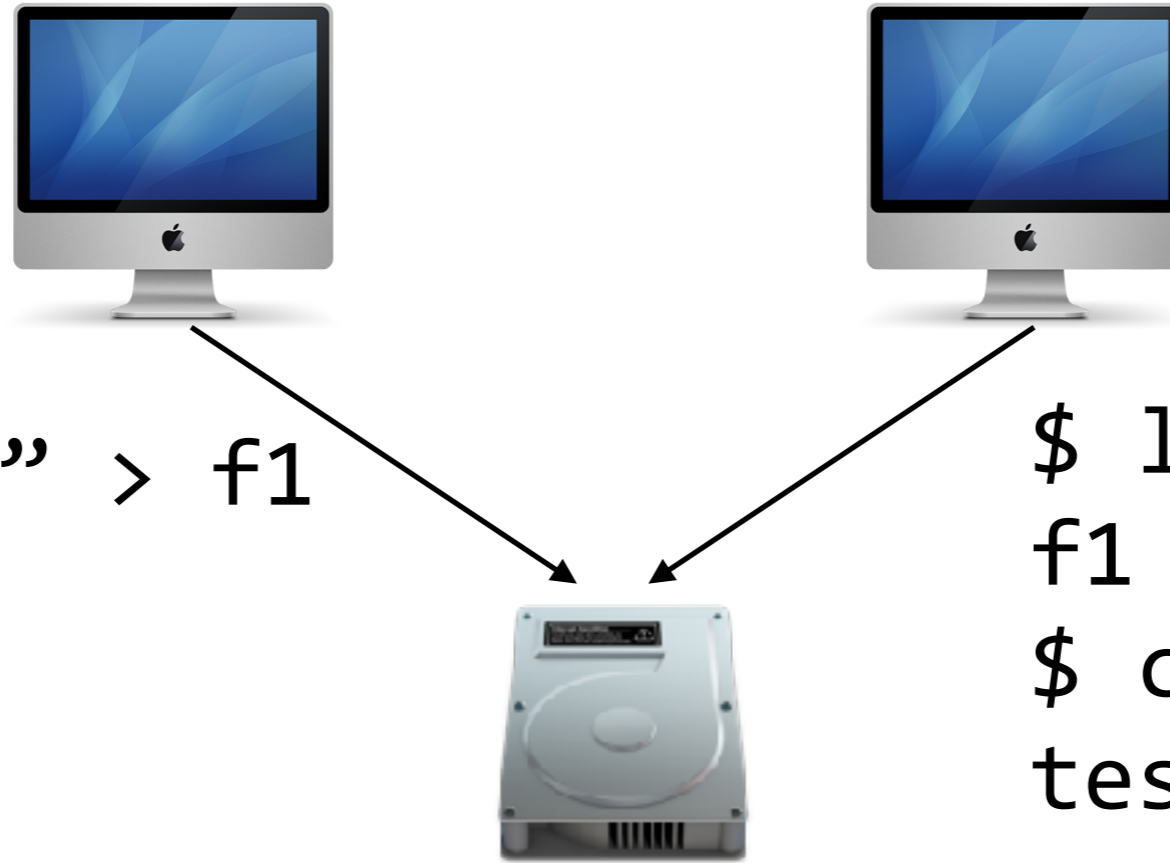
- Distributed systems can help us with:
 - Scalability
 - Performance
 - Latency
 - Availability
 - Fault Tolerance
- We usually partition + replicate our data to achieve these goals
- Replication is not trivial

A Distributed Filesystem



```
$ echo "test" > f1  
$ ls  
f1  
$ cat f1  
test
```

A Distributed Filesystem



```
$ echo "test" > f1
```

```
$ ls  
f1  
$ cat f1  
test
```

A Distributed Filesystem

- This semester, you will create a distributed filesystem
- Motivation:
 - Storing files in memory is faster than on disk (much lower latency)
 - But...
 - Practical limits of MB/machine
 - Very ephemeral: machine crashes/reboots, it's gone
- Solution:
 - Store data in memory of *many* computers, can partition (to store more than can fit on 1), replicate (fault tolerance)
 - Use a permanent backing store to keep a canonical store of files

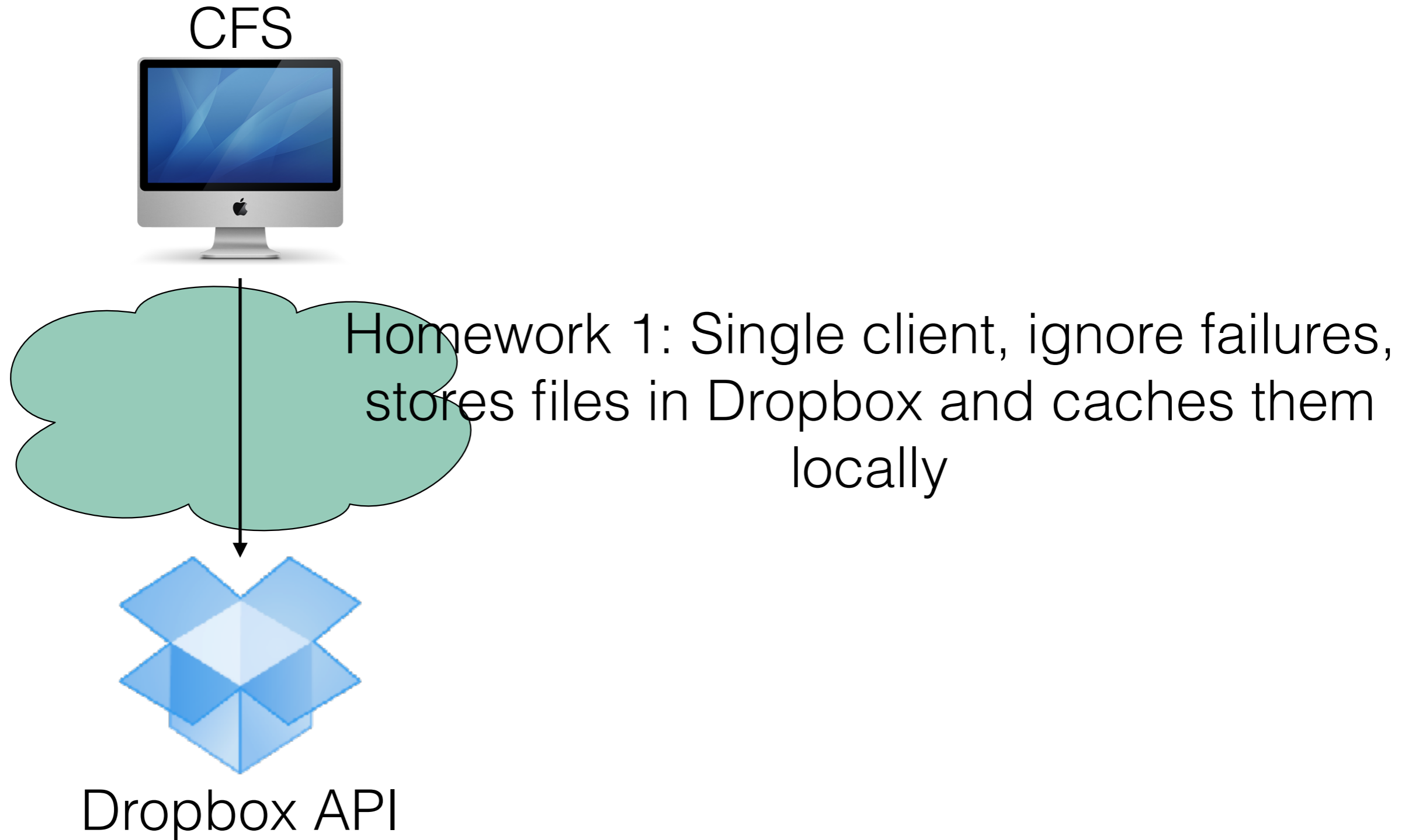
High level design questions

- How do we expose the system?
 - Files (e.g. NFS)?
 - Blocks (e.g. SANs)?
 - Key/value (e.g. S3)?
 - Database?
- No right answer for a generic solution - always tradeoffs
- For practicality, we'll use a **file** interface, but make it look a lot like a key/value interface for simplicity

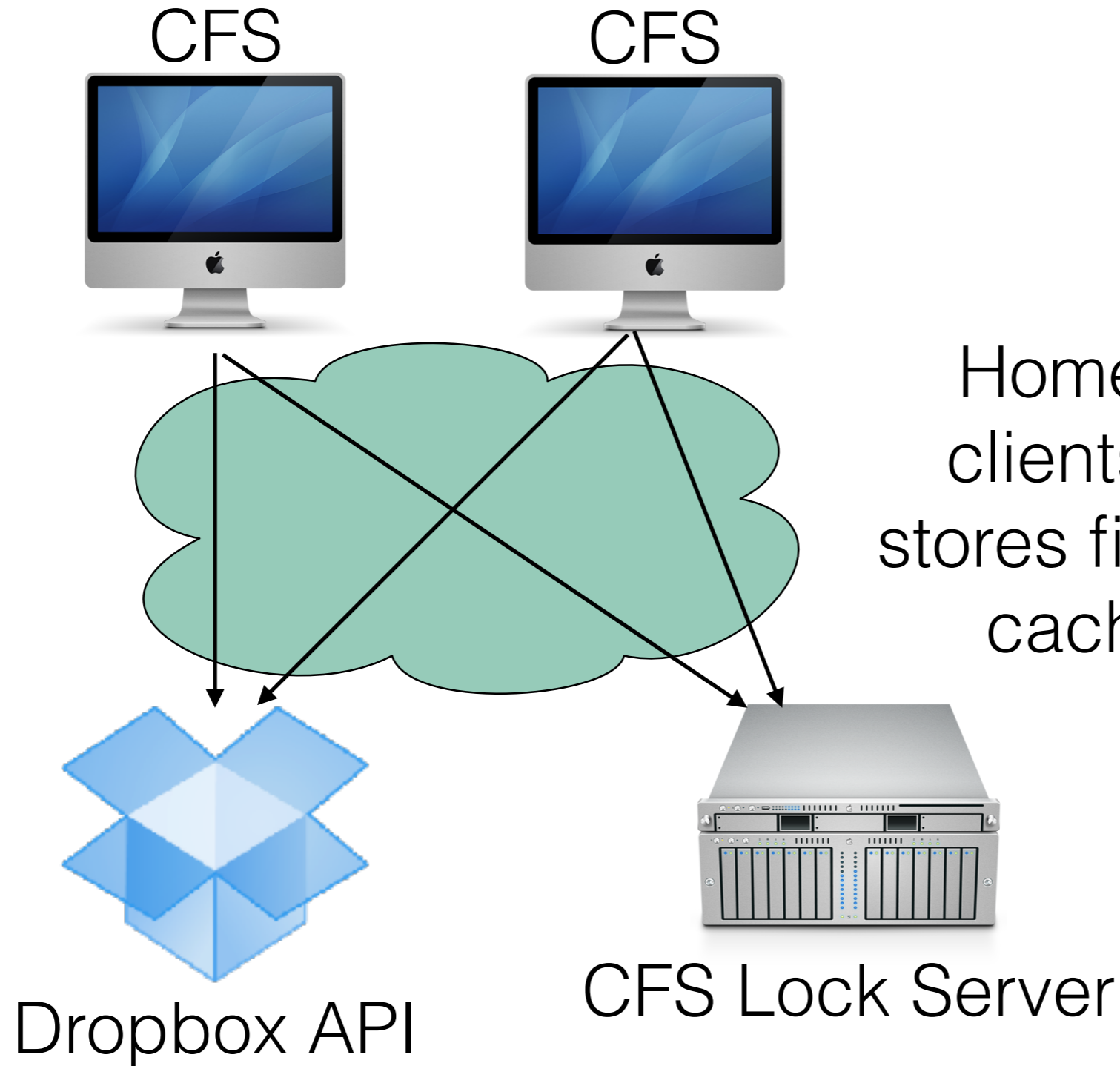
High level design questions

- How do we maintain consistency?
 - What if two clients try to move the same folder at the same time? Or create a new file at the same time?
- How do we scale? How do we partition?
- How do we maintain consistency despite node failures?
- How do clients/servers communicate?
 - Are we writing our own network protocol stack?
- How do we handle concurrent clients?

CloudFS

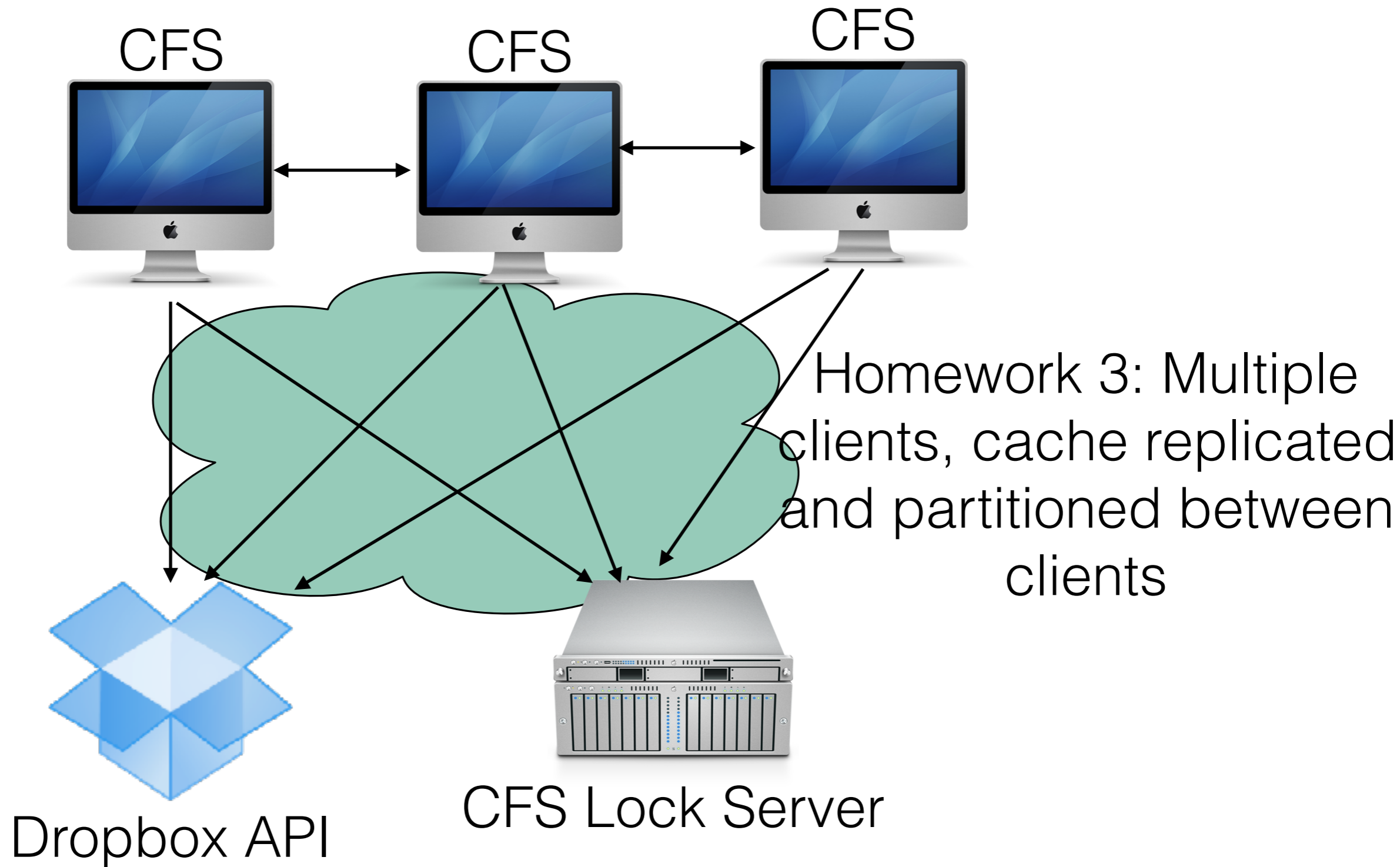


CloudFS

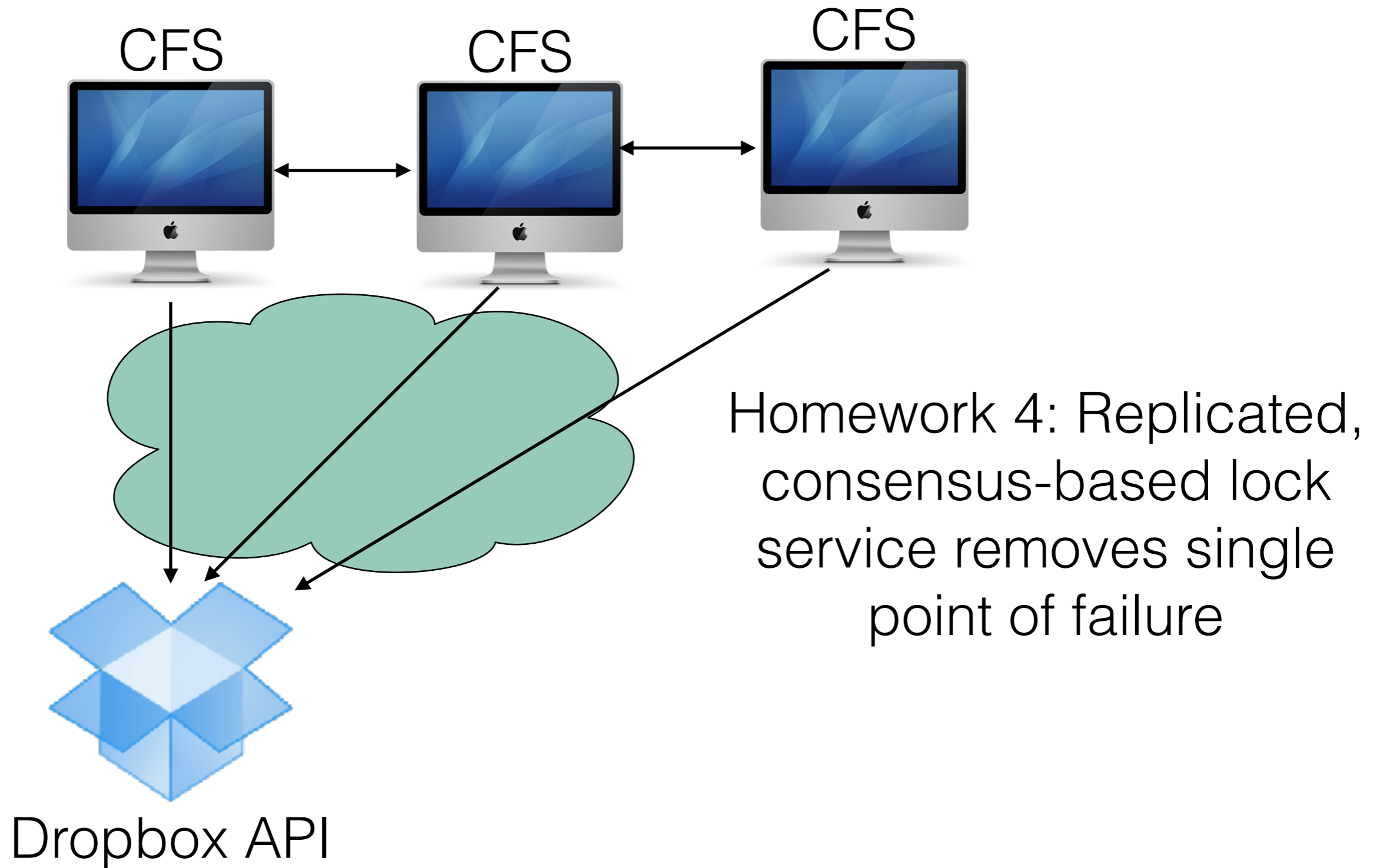


Homework 2: Multiple clients, ignore failures, stores files in Dropbox and caches them locally

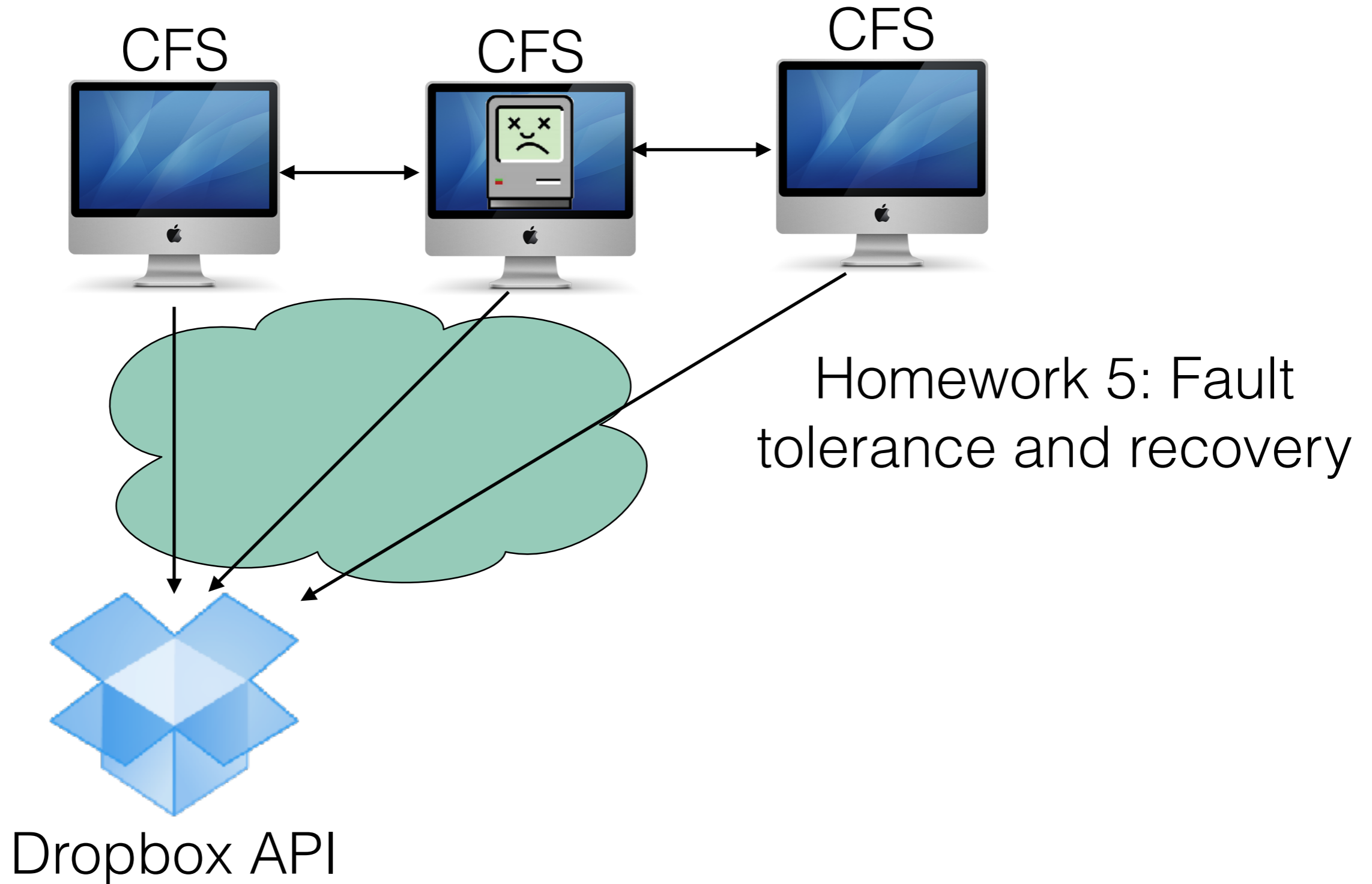
CloudFS



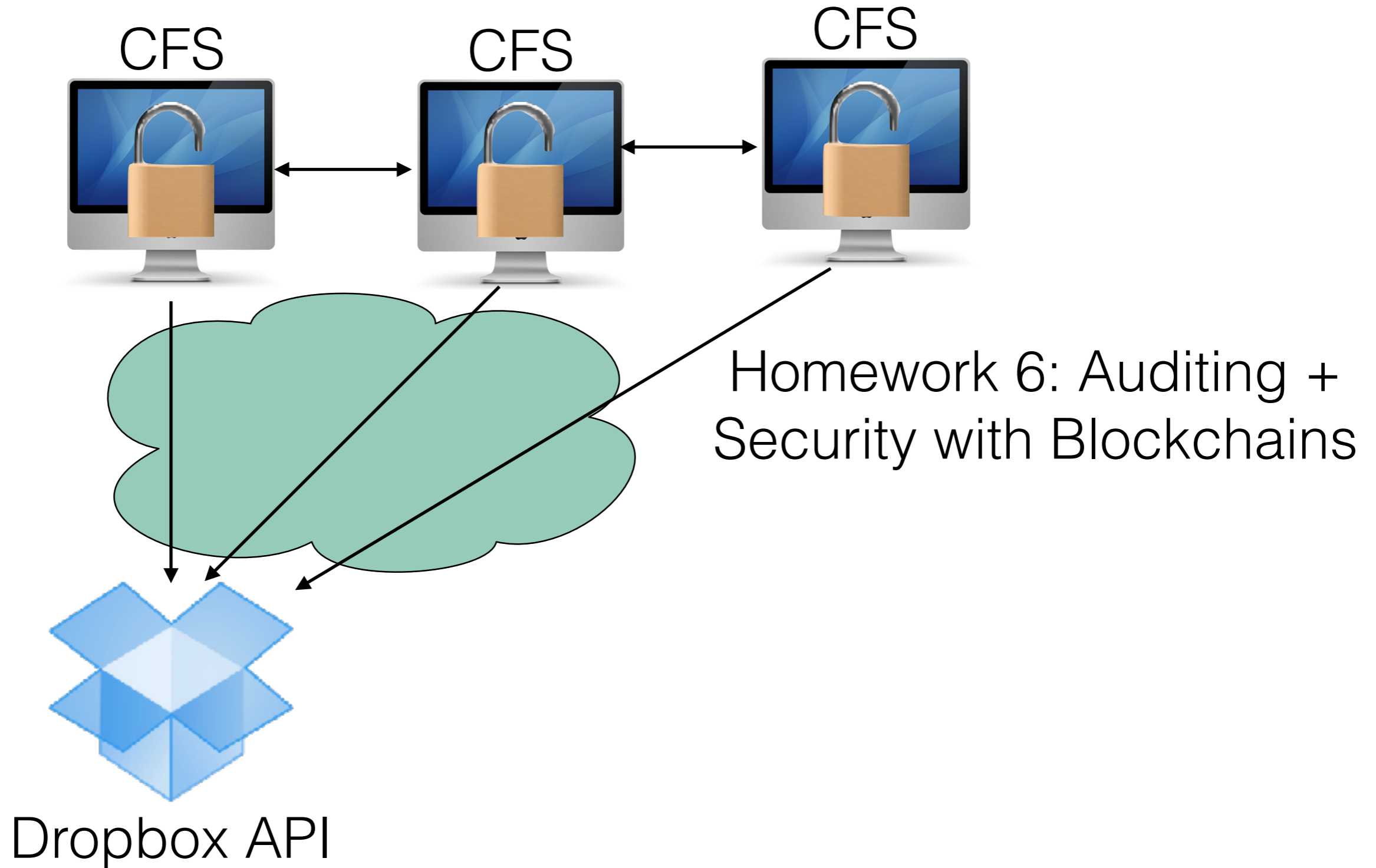
CloudFS



CloudFS



CloudFS



Homework 1

- Due 2/8, 4:00pm
- Base code is provided to connect to Dropbox, create and mount a filesystem
- Your job: add a cache
- Warning: it's tricky! Even with just 1 client, you can still have concurrency (multiple apps on same machine accessing same files)
- Next week: java concurrency refresher
- Very important HW logistics reminders:
 - Late policy (>24 hrs is not accepted)
 - Submission is via GitHub Classroom (use invite link to get your repository, do NOT fork mine directly), you must make a release to submit
 - If it doesn't run in the provided VM, you won't do very well.

Online activity

Go to: <https://b.socrative.com/> and click on Login (student), then “SWE622” for room name