CS 4530: Fundamentals of Software Engineering

Module 9.2: Distributing Data

Adeel Bhutta, Mitch Wand

Khoury College of Computer Sciences

Learning Goals for this Lesson

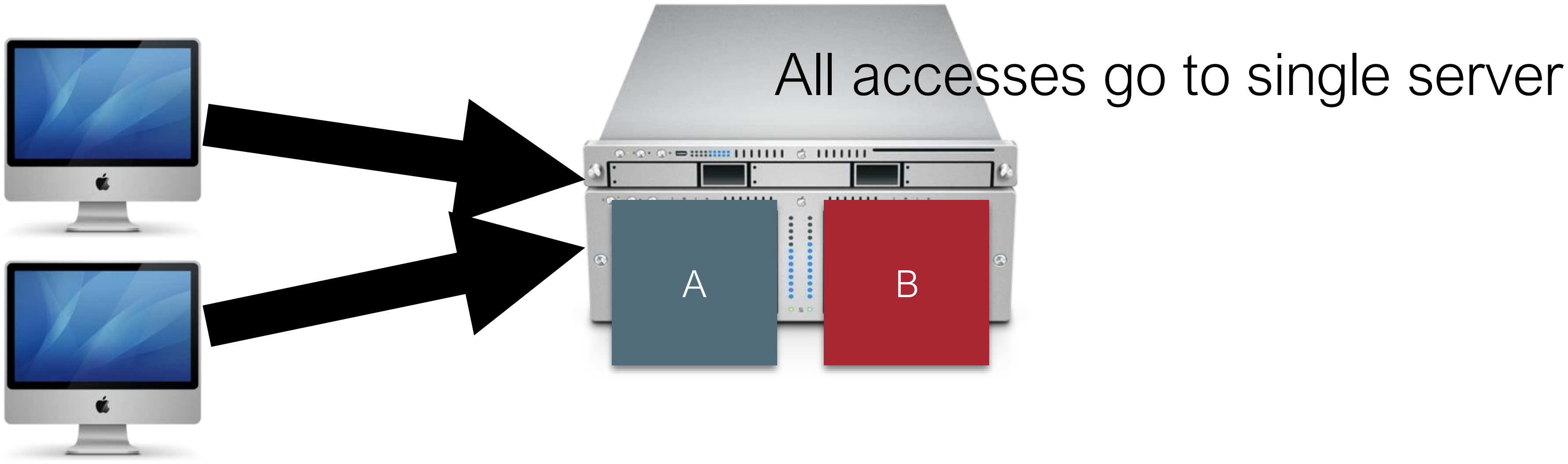
- At the end of this lesson, you should be able to
  - explain the concepts of data partition and replication
  - List and explain the major benefits and pitfalls of each of these
  - Explain the CAP theorem

Dealing with shared data is a challenge

- Most distributed systems have some shared data
- How important is it to:
  - Retrieve data quickly?
  - Update data quickly?
  - Make sure all users see the same data?
  - Make sure all users can always see (some values of) the data
- This all depends on the goals of the system.

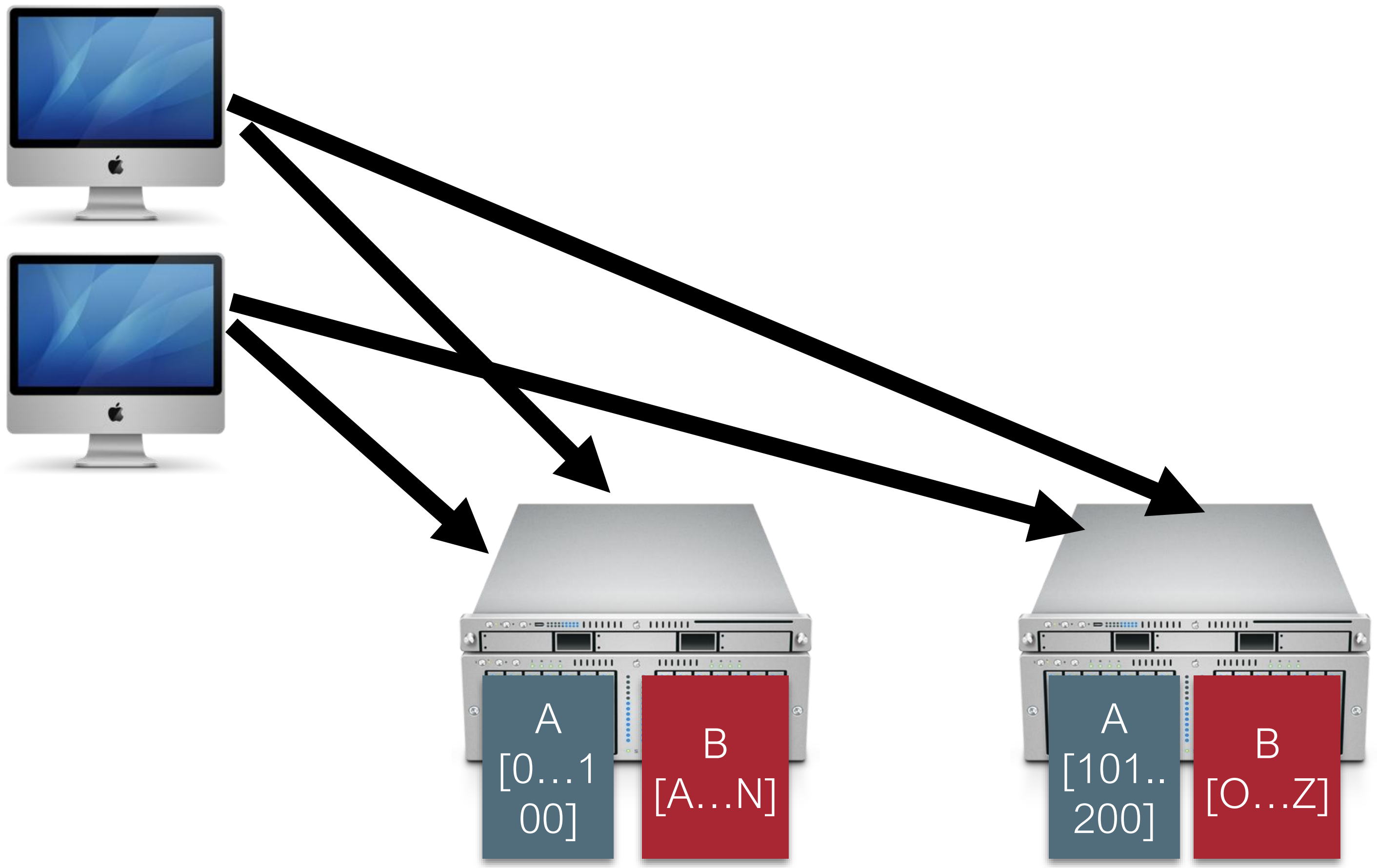
Recurring Problem #1: Too Much Data

- In a non-distributed system, all accesses go to a single server.



Recurring Solution #1: Partitioning

- Divide up the data in some (hopefully logical) way.
- Each server is responsible for only some of the data



Partitioning has some advantages

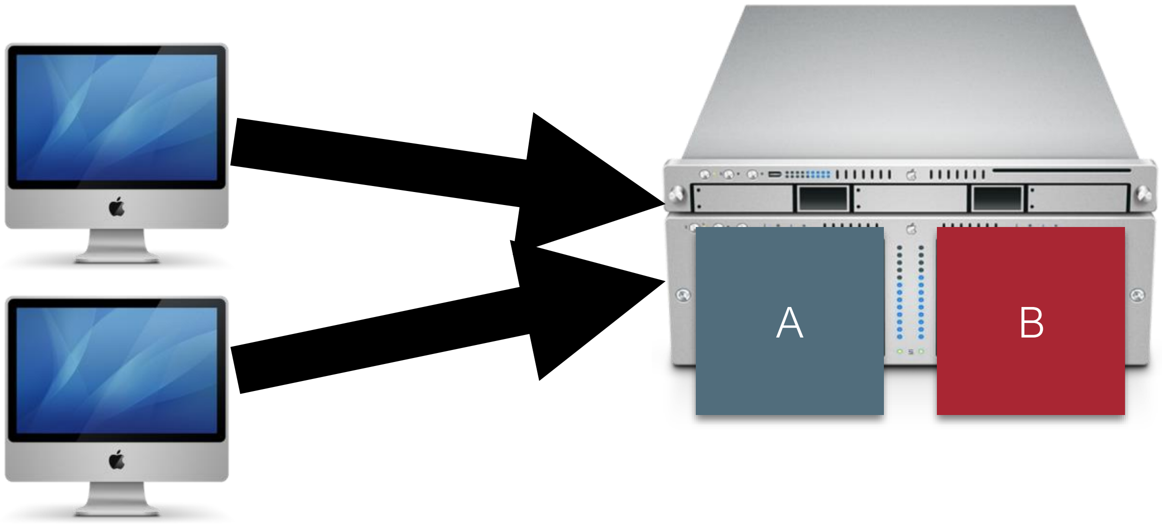
- Each server has 50% of the data
- Requires less processing power per server
- Allows concurrency in reads/writes
- Even if one server goes down, still have access to 50% of the data

Partitioning also has a big challenge

- What’s a good way to divide the data?
  - Depends on the nature of the application
  - We’ll see this in our case studies

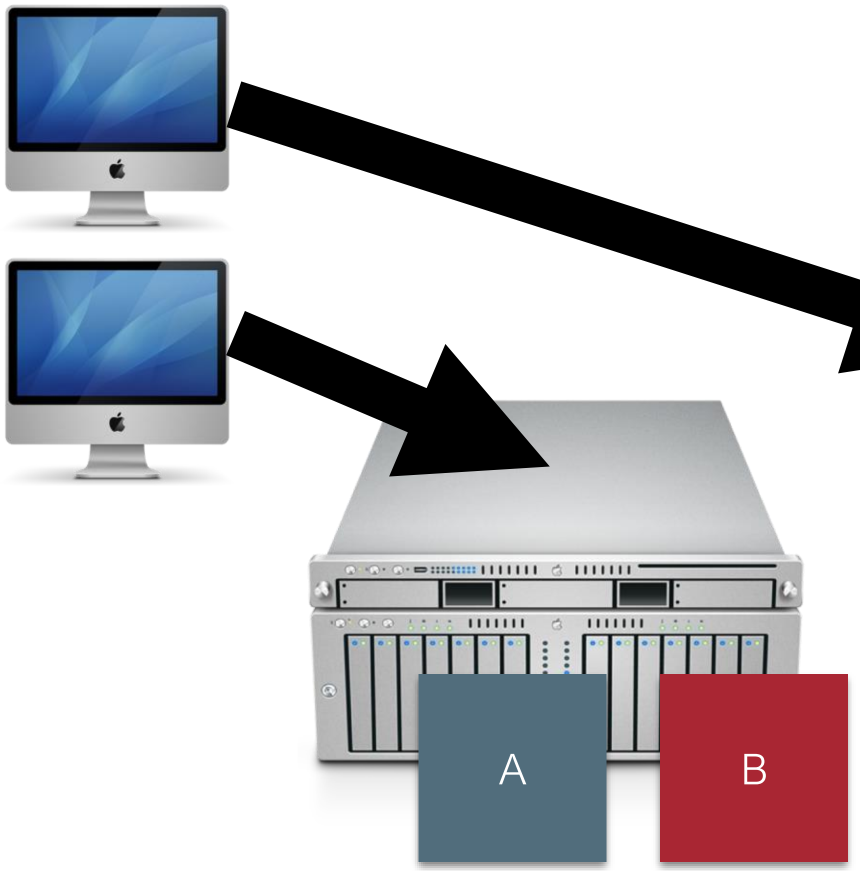
Recurring Problem #2: Too Many Requests

- In a non-distributed system, all requests go to a single server.



Recurring Solution #2: Replication

- Entire data set is copied
- Either server can handle any request

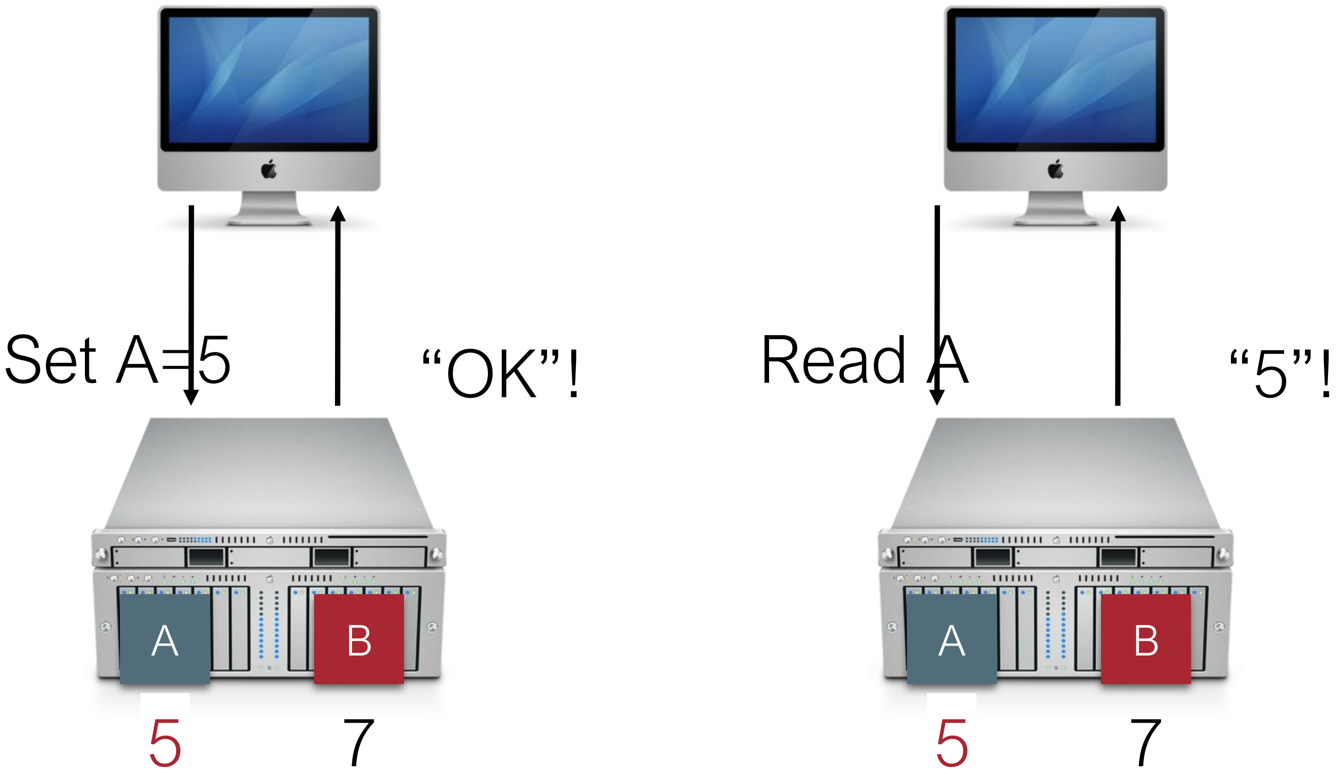


Replication has advantages

- 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

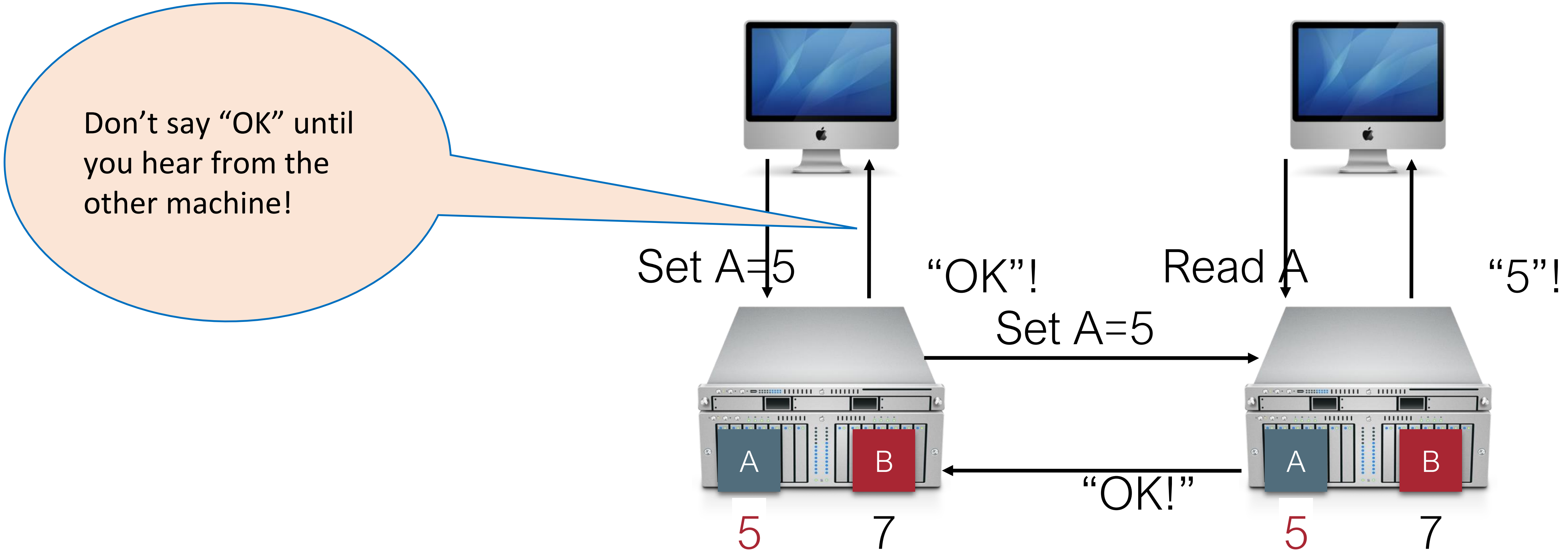
But replication has a big problem: Consistency

- We probably want our system to work like this:
- If we tell the machine on the left to set A to 5, then we expect the machine on the right to return 5 if we ask it for the value of A.



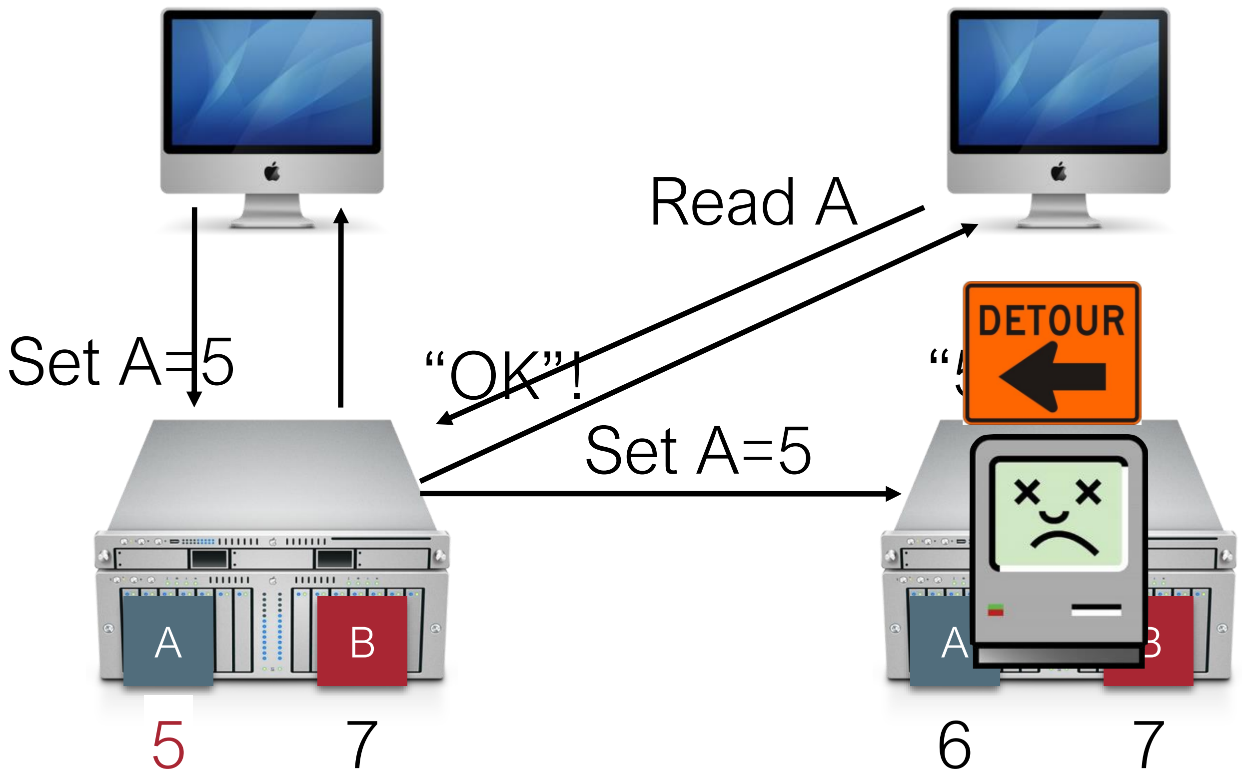
Sequential Consistency is the Ideal

- AKA: Behaves like a single machine would
- Possible algorithm: two-phase commit



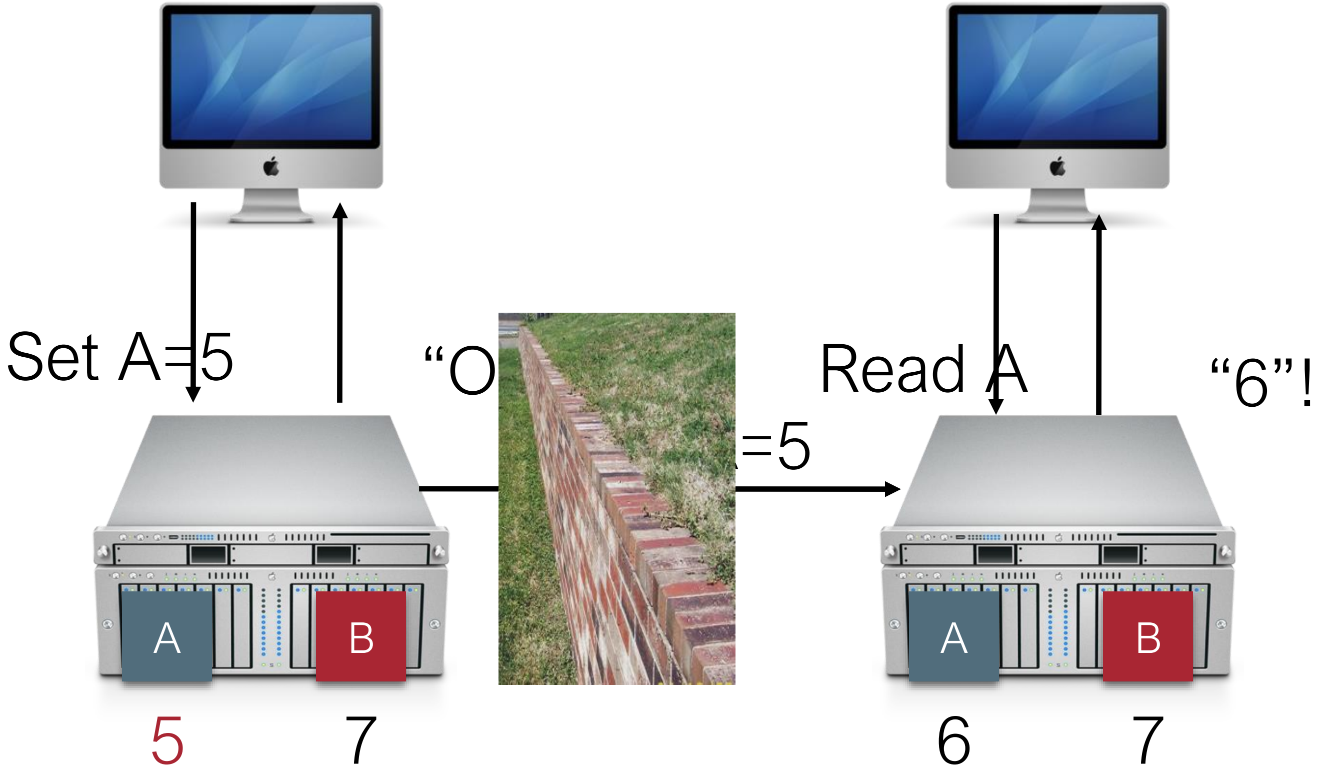
One of the replicas might crash

- On timeout, assume node is crashed
- Reroute requests to live nodes



But if the *network* fails?

- No way to tell whether it’s the network or the remote machine.



CAP Theorem: Consistency or Availability

- 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)
    - Can’t drop this for a DS - networks can always fail

Luckily, there are possible compromises

- Sacrifice some availability for consistency (eg in a chat system: you want the chats to appear in order)
- Sacrifice some consistency for availability (eg you may not care in what order the cats appear)
- Or you may want different policies for reads vs. writes.
- Doing this is beyond the limits of this course (whew!)

Most distributed systems combine both partitioning and replication

Learning Goals for this Lesson

- You should now be able to
  - explain the concepts of data partition and replication
  - List and explain the major benefits and pitfalls of each of these
  - Explain the CAP theorem