Midterm Review & Agreement/Consensus

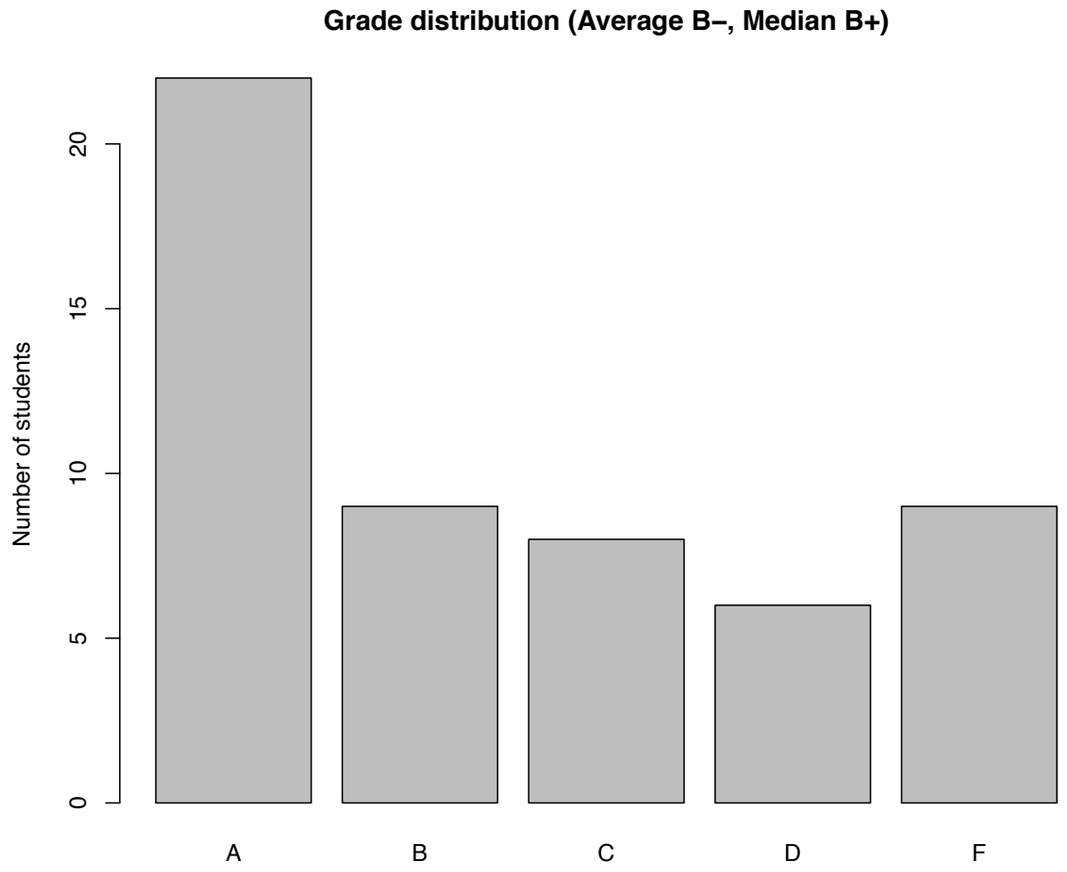
CS 475, Spring 2018
Concurrent & Distributed Systems

Announcements

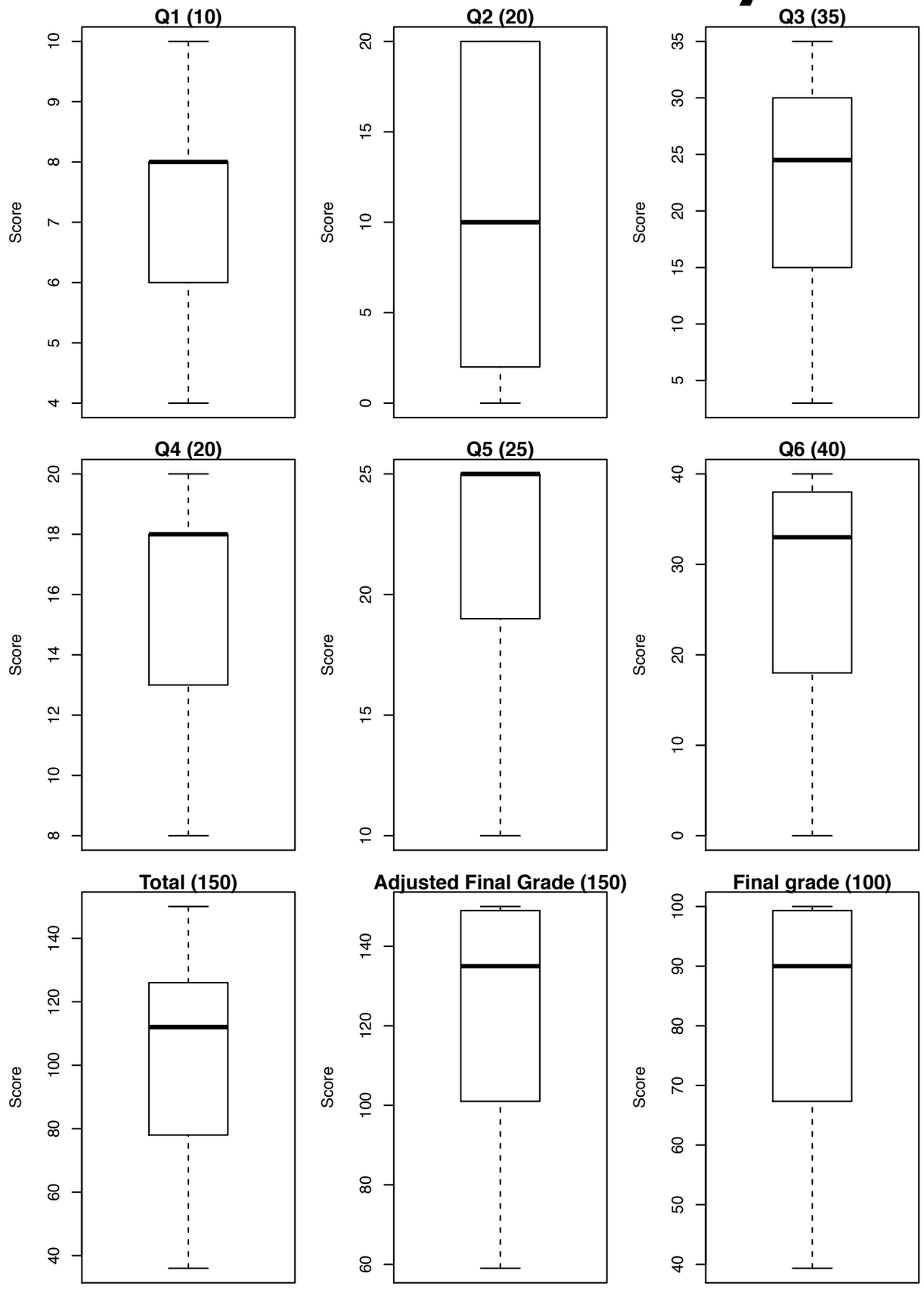
- HW4 is out!
  - http://www.jonbell.net/gmu-cs-475-spring-2018/homework-4/
- Today:
  - Midterm review
  - Agreement in distributed systems
- Weds:
  - Video lecture (Prof Bell at secret meeting)

Midterm Summary

- Yes, it was long
- Grade statistics
  - Pre-curve: 50% of class > 73%
  - Post-curve: 50% of class > 89% (roughly eq to saying “drop the lowest scoring problem on each exam” but more favorable)
  - Wide distribution: 25% had A+, 25% had F



Midterm Summary Grades



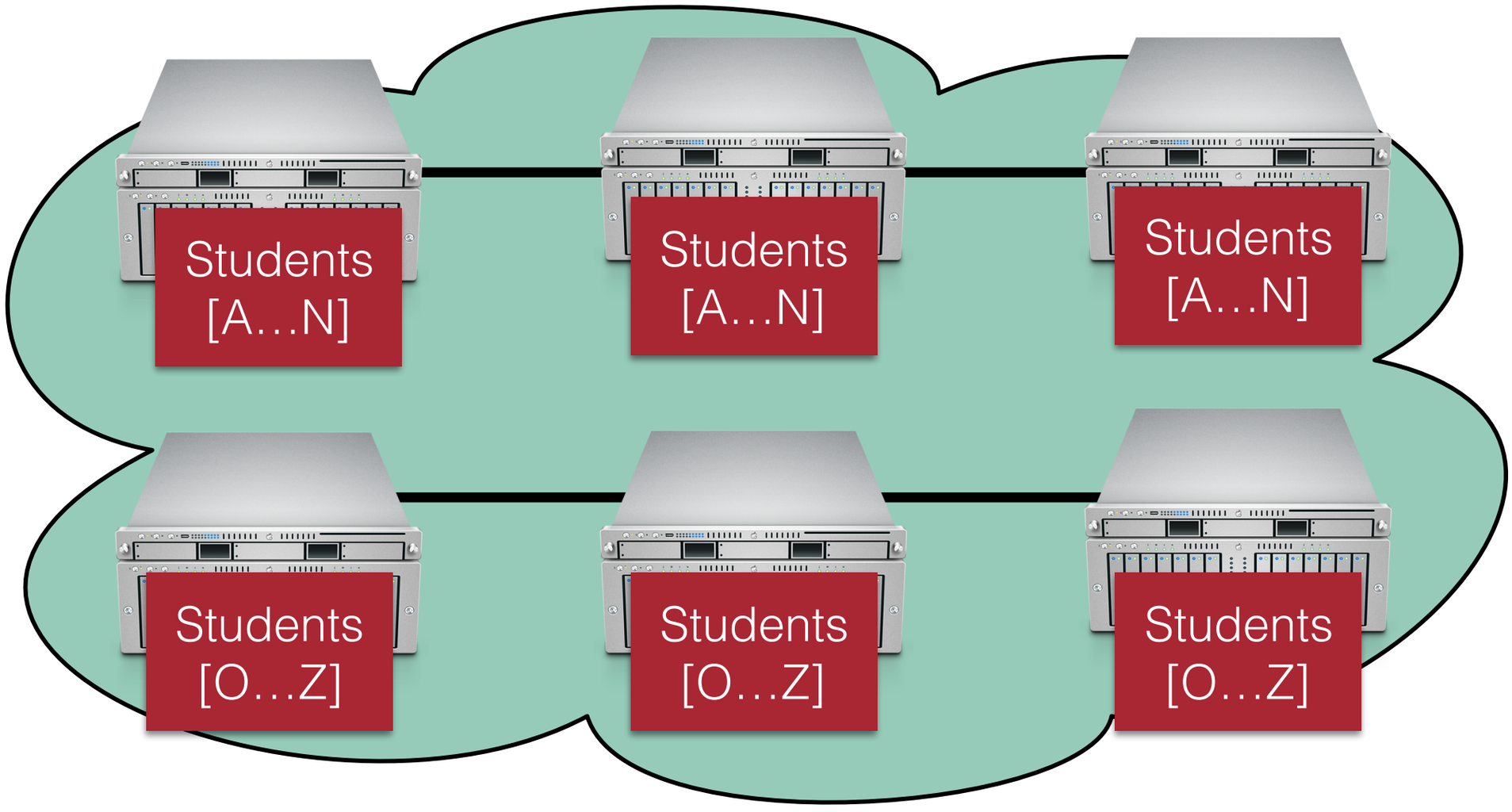
(Review Exam Q by Q)

Review: Strawman Sharding Scheme

In this class:

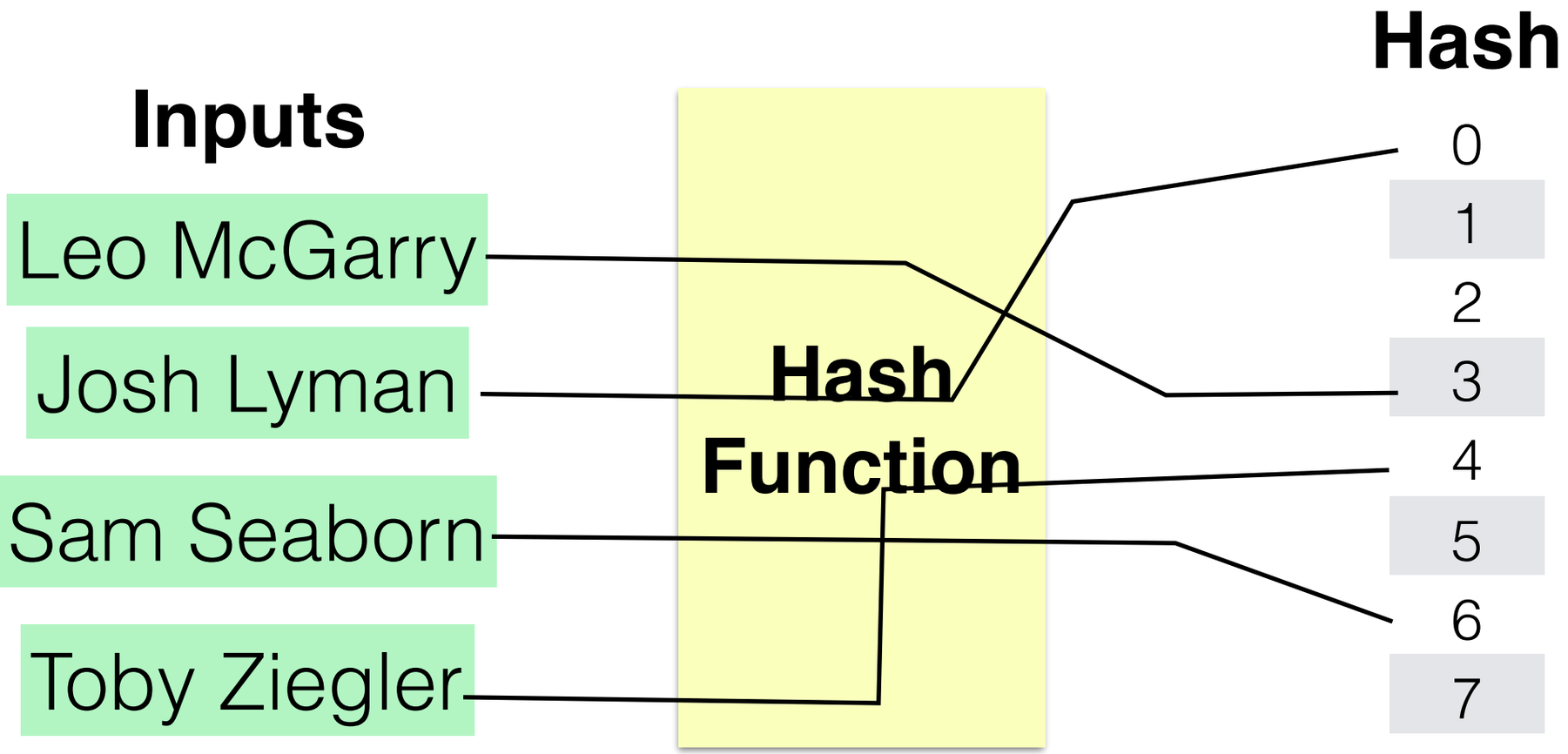
40 students

16 students



Review: Hashing

- Compresses data: maps a variable-length input to a fixed-length output
- Relatively easy to compute
- Example:



Review: Consistent Hashing

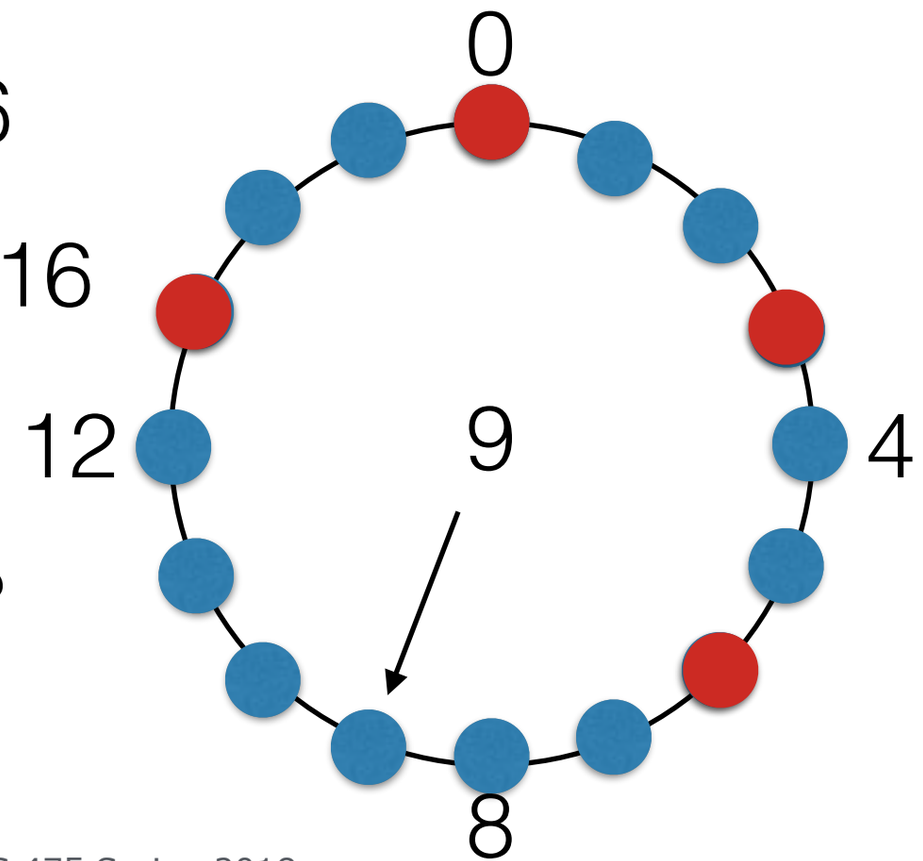
- Construction:
  - Assign each of C hash buckets to random points on mod 2^n circle, where hash key size = n
  - Map object to pseudo-random position on circle
  - Hash of object is the closest clockwise bucket

Example: hash key size is 16

Each ● is a value of hash % 16

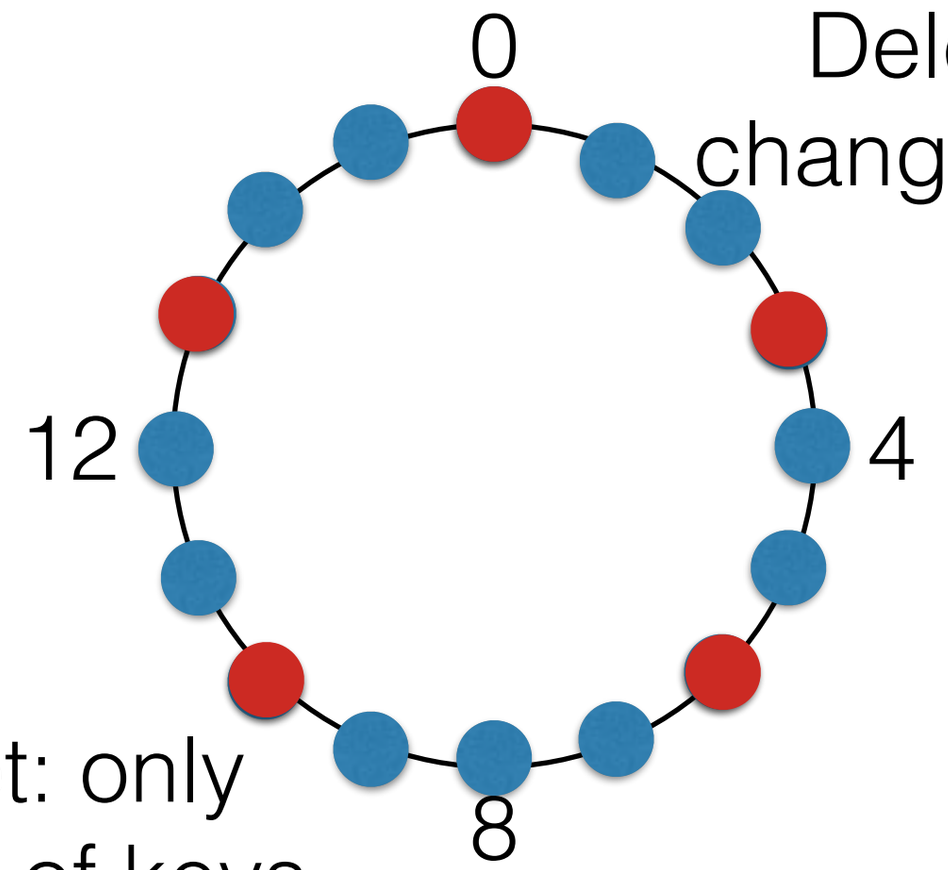
Each ● is a bucket

Example: bucket with key 9?



Review: Consistent Hashing

It is relatively smooth: adding a new bucket doesn't change that much



Delete bucket: only changes location of keys 1,2,3

Add new bucket: only changes location of keys 7,8,9,10

Today: Transactions

```
boolean transferMoney(Person from, Person to, float amount){
    if(from.balance >= amount)
    {
        from.balance = from.balance - amount;
        to.balance = to.balance + amount;
        return true;
    }
    return false;
}
```

What can go wrong here?

Transactions: Classic Example

```
boolean transferMoney(Person from, Person to, float amount){
    if(from.balance >= amount)
    {
        from.balance = from.balance - amount;
        to.balance = to.balance + amount;
        return true;
    }
    return false;
}
```

transferMoney(P1, P2, 100)	**transferMoney(P1, P2, 200)**
P1.balance (200) >= 100	P2.balance (200) > 200
P1.balance = 200 - 100 = 0	
P2.balance = 200 + 100 = 300	
return true;	P1.balance = 100 - 200 = -100
	P2.balance = 300 + 200 = 500
	return true;

What's wrong here?
Need isolation (prevent overdrawing)

Transactions: Classic Example

```
boolean transferMoney(Person from, Person to, float amount){
    synchronized(from){
        if(from.balance >= amount)
        {
            from.balance = from.balance - amount;
            to.balance = to.balance + amount;
            return true;
        }
        return false;
    }
}
```

transferMoney(P1, P2, 100)	transferMoney(P1, P2, 200)
P1.balance (200) >= 100	
P1.balance = 200 - 100 = 0	
P2.balance = 200 + 100 = 300	
return true;	
	P1.balance <= 200
	return false;

Adding a lock: prevents accounts from being overdrawn

But: shouldn't we lock on `to` also?

Transactions: Classic Example

```
boolean transferMoney(Person from, Person to, float amount){
    synchronized(from, to){
        if(from.balance >= amount)
        {
            from.balance = from.balance - amount;
            to.balance = to.balance + amount;
            return true;
        }
        return false;
    }
}
```

transferMoney(P1, P2, 100)	**transferMoney(P1, P2, 200)**
P1.balance (200) >= 100	
P1.balance = 200 - 100 = 0	
P2.balance = 200 + 100 = 300	
return true;	
	P1.balance <= 200
	return false;

Locking on both from, to at same time

Transactions: Classic Example

```
boolean transferMoney(Person from, Person to, float amount){
    synchronized(from, to){
        if(from.balance >= amount)
        {
            from.balance = from.balance - amount;
            to.balance = to.balance + amount;
            return true;
        }
        return false;
    }
}
```

transferMoney(P1, P2, 100)	**transferMoney(P1, P2, 200)**
P1.balance (200) >= 100 P1.balance = 200 - 100 = 0 	P1.balance <= 200 return false;

Problem: `P1.balance` was deducted `P2.balance` not incremented! ("Atomicity violation")

Transactions

- How can we provide some consistency guarantees **across operations**
- Transaction: unit of work (grouping) of operations
  - Begin transaction
  - Do stuff
  - Commit OR abort

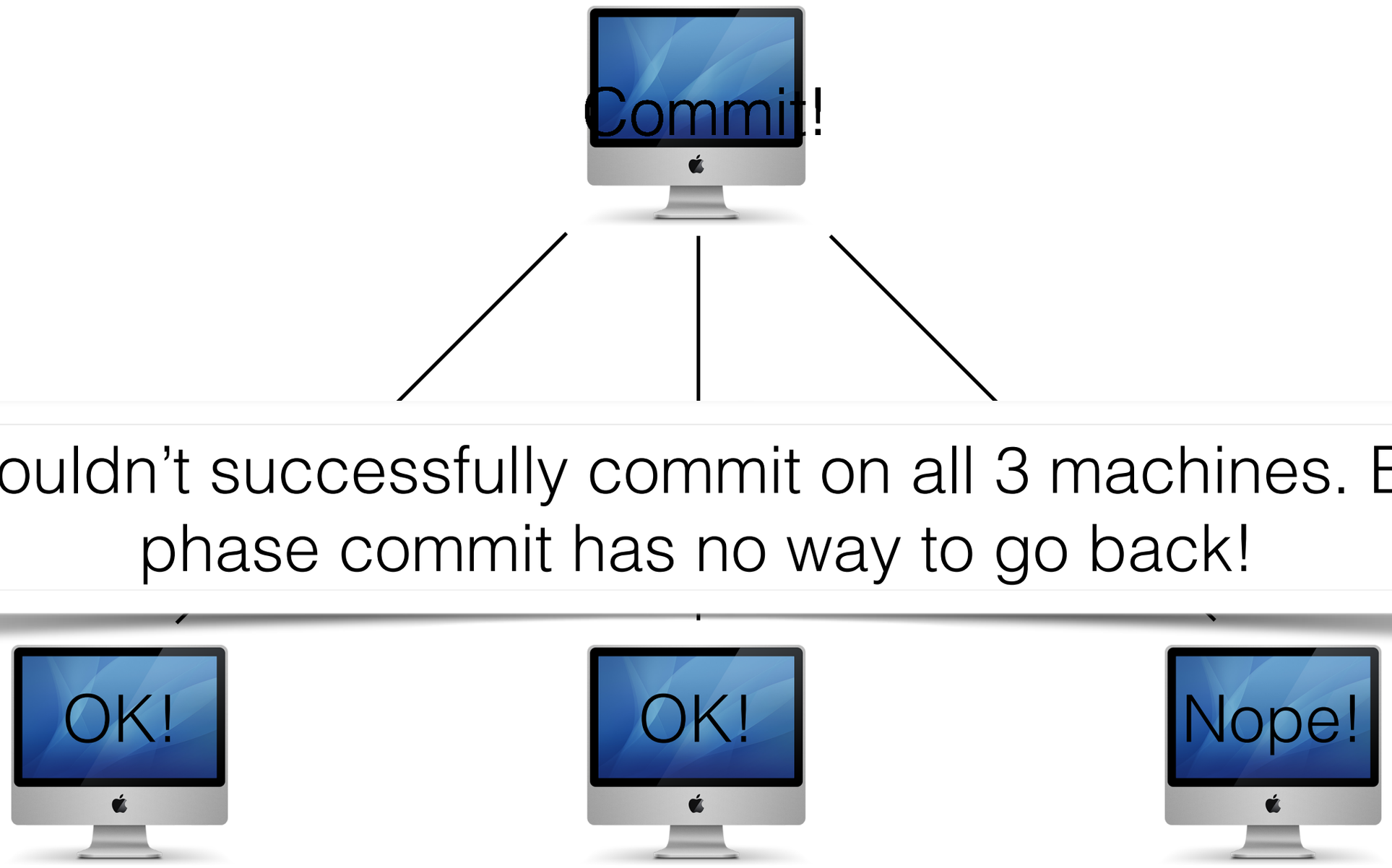
Properties of Transactions

- Traditional properties: ACID
- **Atomicity**: transactions are “all or nothing”
- **Consistency**: Guarantee some basic properties of data; each transaction leaves the database in a valid state
- **Isolation**: Each transaction runs as if it is the only one; there is some valid serial ordering that represents what happens when transactions run concurrently
- **Durability**: Once committed, updates cannot be lost despite failures

1-Phase Commit

- Naive protocol: coordinator broadcasts out "commit!" continuously until participants all say "OK!"
- Problem: what happens when a participants fails during commit? How do the other participants know that they shouldn't have really committed and they need to abort?

1-Phase Commit



We couldn't successfully commit on all 3 machines. But 1-phase commit has no way to go back!

Distributing Transactions

- System model: data stored in multiple locations, multiple servers participating in a single transaction. One server pre-designated "coordinator"
- Failure model: messages can be delayed or lost, servers might crash, but have persistent storage to recover from

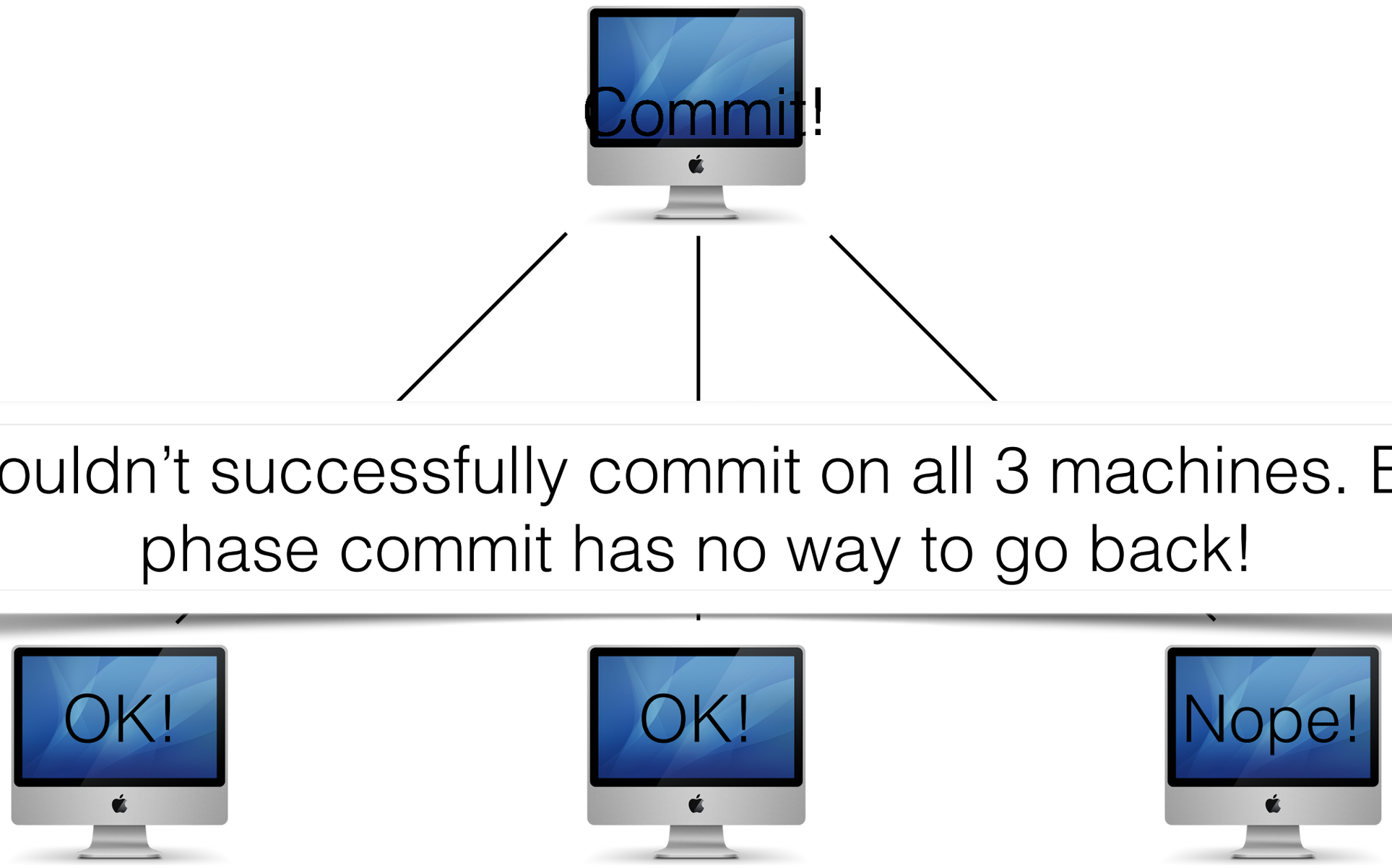
Distributed Transactions

- Coordinator: Begins a transaction
  - Assigns a unique transaction ID
  - Responsible for commit + abort
  - In principle, any client can be the coordinator, but all participants need to agree on who is the coordinator
- Participants: everyone else who has the data used in the transaction

1-Phase Transaction Commit

- Naive protocol: coordinator broadcasts out "commit!" continuously until participants all say "OK!"
- Problem: what happens when a participants fails during commit? How do the other participants know that they shouldn't have really committed and they need to abort?

1-Phase Commit



We couldn't successfully commit on all 3 machines. But 1-phase commit has no way to go back!