Systems Programming & Safety
Computer Systems Programming, Spring 2025

Instructor: Travis McGaha

Teaching Assistants:

Andrew Lukashchuk	Ashwin Alaparthi	Lobi Zhao
Angie Cao	Austin Lin	Pearl Liu
Aniket Ghorpade	Hassan Rizwan	Perrie Quek

Poll Everywhere

pollev.com/tqm

- Any Questions for me?

Administrivia

- Final Project Details Posted
  - SOME of it is auto graded. There is a lot of functionality that is not autograded that you will need to implement
- HW09 Manual checks posted
- Next check-in to be posted soon
- End of semester survey to be posted soon(?
- Exam logistics & Practice exam to be posted soon

Lecture Outline

- **Intro to Distributed Systems**
  - Sequential Consistency
  - Logical Clocks & Ordering
  - Fault Tolerance
  - Performance

What are distributed systems?

- A group of computers communicating over the network by sending messages, which interact to accomplish some common task
  - There is no shared hardware (e.g. memory) other than the network
  - Individual computers (nodes) can fail
  - The network itself can fail (Drop messages, corrupt messages, delay messages, etc.)

Why do we care?

- They are really interesting problem to work with

- Most applications we interact with are distributed systems of some sort:

Why do we care?

- They are really interesting problem to work with
- Distributed systems typically allow a system to scale well. Need more work to be done? Just add a new computer to the system
- Distributed systems can also allow for some amount of "fault tolerance". If one computer crashes, the rest of the computers will probably keep running.

Distributed Systems Concerns

- How do we make it so that the computers work together:
  - Correctly
  - Consistent
  - Efficiently
  - At (huge) scale
  - High availability
- Despite issues with the network
- Despite some computers crashing
- Despite some computers being compromised

Distributed Systems: Pessimistic View

- Considered a very hard topic
  - Involves many of the topics covered in this course and more
  - CIS 5050 spends ~8 lectures covering things already introduced here. (out of 25 lectures)

- “The most thought per line of code out of any course”
  - Hal Perkins Circa 2019

- “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, circa 1990

Lecture Outline

- **Intro to Distributed Systems**
  - **Sequential Consistency (with threads first)**
  - Logical Clocks & Ordering
  - Fault Tolerance
  - Performance

Poll Everywhere

pollev.com/tqm

- Consider the following code.
  Each function is called by different threads.
  Is it possible for the code to crash?

```
bool set = false;
int value = 0;

void* thrd1(void* arg) {
  value = 3034;
  set = true;
}

void* thrd2(void* arg) {
  if (set) {
    assert(value == 3034);  // crashes if expression is false.
  }
}
```

Aside: Instruction & Memory Ordering

- Do we know that **`t`** is set before **`g`** is set?

```
bool g = false;
int t = 0

void some_func(int arg) {
  t = arg;
  g = true;
}
```

Aside: Instruction & Memory Ordering

- The compiler may generate instructions with different ordering if it does not appear that it will affect the semantics of the function
  - Since `g = true;` is not affected by `t = arg;` then either one could execute first.

- The Processor may also execute these in a different order than what the compiler says

- Why? Optimizations on program performance
  - If you want to know more, look into "Out-of-Order Execution" and "Memory Order"

Aside: Memory Barriers

- How do we fix this?

- We can emit special instructions to the CPU and/or compiler to create a "memory barrier"
  - "all memory accesses before the barrier are guaranteed to happen before the memory accesses that come after the barrier"

  - A way to enforce an order in which memory accesses are ordered by the compiler and the CPU

  - This is done for us when we mark a variable as atomic or use a lock.

Sequential Consistency

- The property that
  - "The result of an execution is the same **AS IF** the operations of all the processors were executed in some sequential order…"
  - "…and the operations of each individual processor appear in this sequence in the order specified by the program."

- Short version:
  - The execution appears to occur in a sequential order
  - And it is the same order specified by the program

Lecture Outline

- **Intro to Distributed Systems**
  - Sequential Consistency
  - **Logical Clocks & Ordering**
  - Fault Tolerance
  - Performance

Shared Nothing Architecture

- Consistency and sharing data is hard in a threaded program
- What about distributed systems?
  - Distributed systems are typically “Shared nothing” meaning that it is a collection of computers communicating over the network
  - There is no shared memory
  - There is no shared disk/storage
- How can we get a cluster (group of machines) to agree on some state?
  - How do the computers in the system reason about each other?

Muddy Foreheads

- Assume the following situation
  - There are n children, k get mud on their foreheads
  - Children sit in circle.
  - Teacher announces, "Someone has mud on their forehead
  - Teacher repeatedly asks "Raise your hand if you know you have mud on your forehead."
  - What happens?

Muddy Foreheads

- Assume the following situation
  - There are n children, k get mud on their foreheads
  - Children sit in circle.
  - Teacher announces, "Someone has mud on their forehead
  - Teacher repeatedly asks "Raise your hand if you know you have mud on your forehead."
  - What happens?
    - The answer is not "no one raises their hand"

Common Knowledge

- There's a difference between what you know and what you know others know
- And what others know you know
- And what others know you know about what you know
- And what you know others know you know about what they know

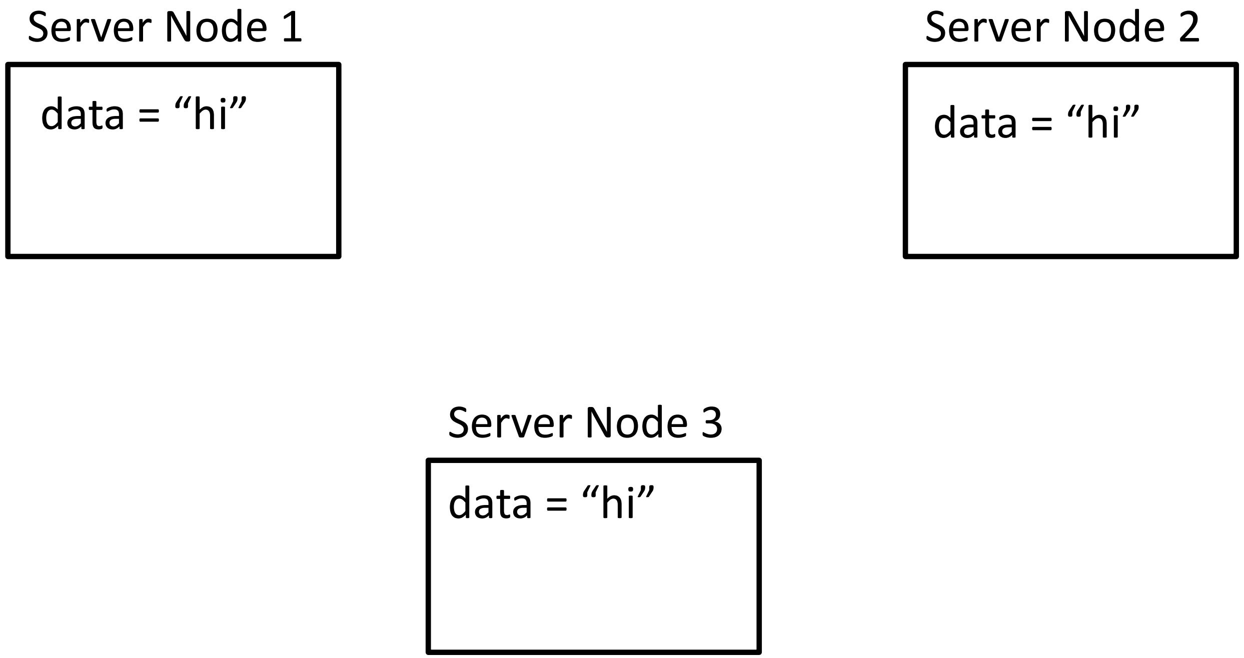
Muddy Forehead Alteration

- What if the teacher pulled each student aside individually and told them “at least one student has mud on their forehead”?

  - Would our solution still work?

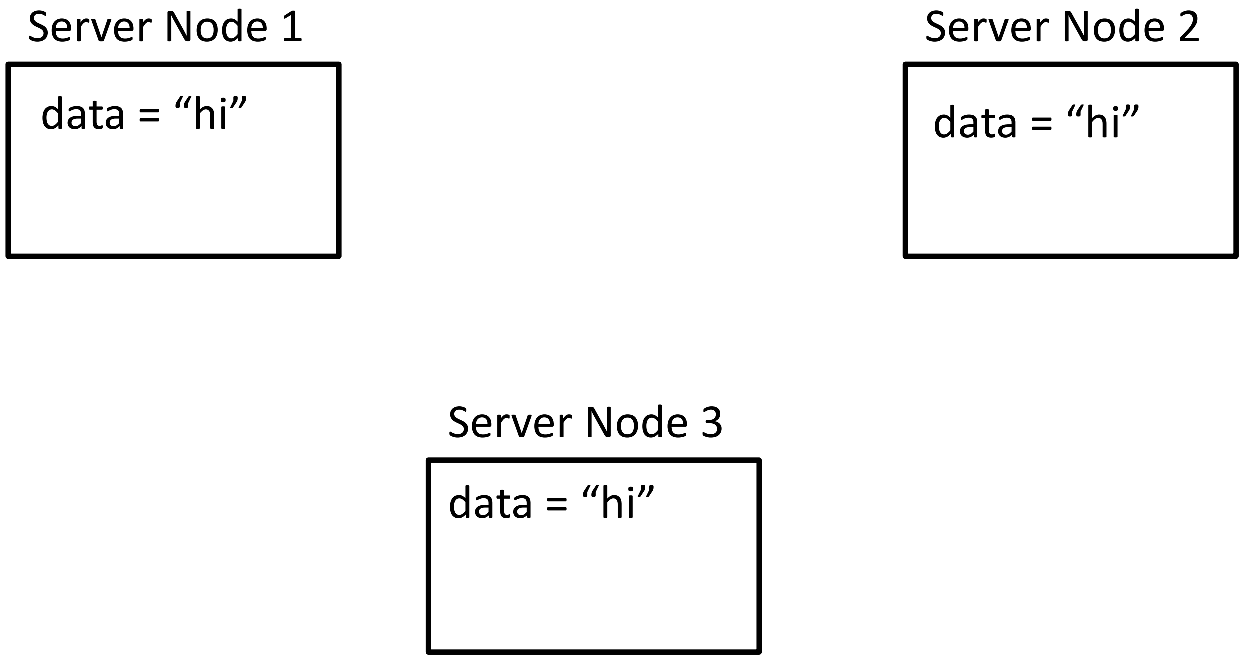
Back to Consistency in Distributed Systems

- Let's say we have a collection of computers that together share the state of a single string.



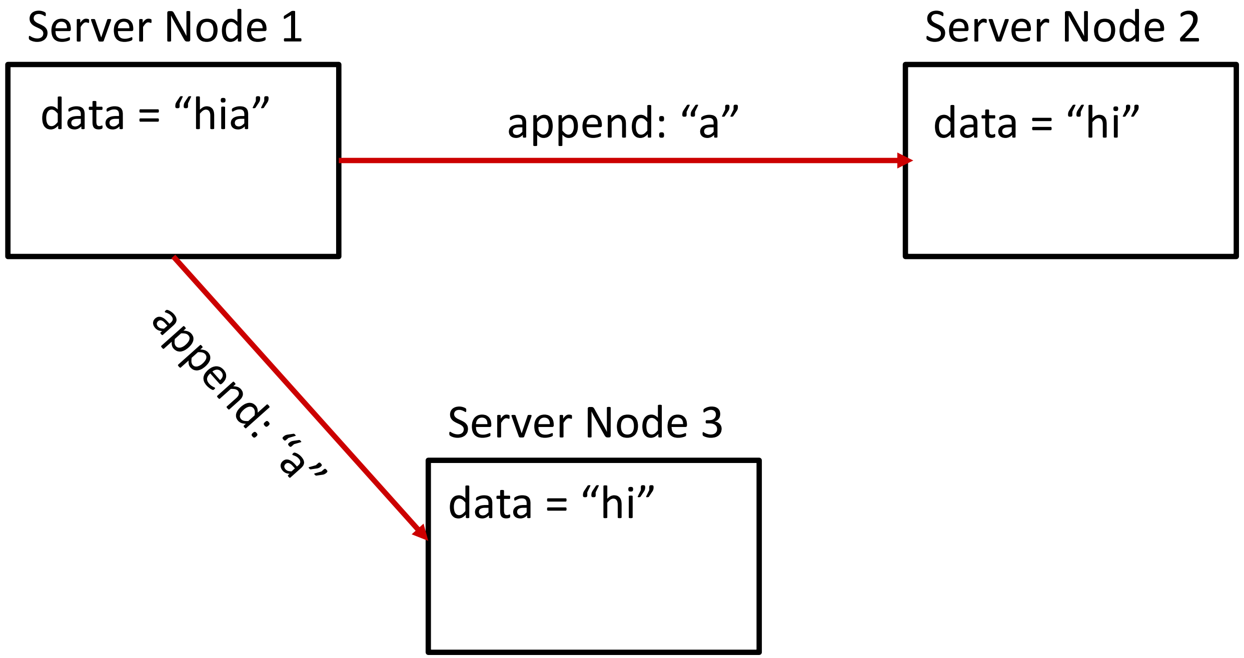
Back to Consistency in Distributed Systems

- Let’s say we have a collection of computers that together share the state of a single string.
- Lets say server 1 gets a request to append “a” to the end of the string.
  How do we maintain a consistent state?



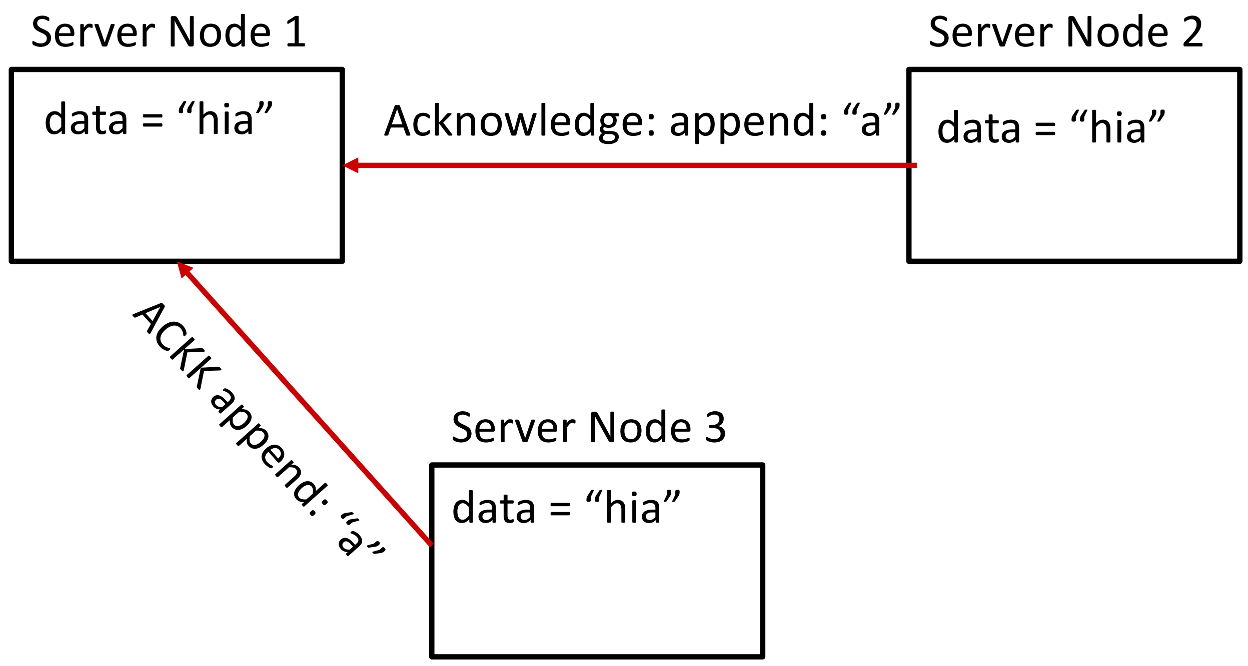
Back to Consistency in Distributed Systems

- Lets say server 1 gets a request to append "a" to the end of the string. How do we maintain a consistent state?
- Simple solution: send the message to other nodes



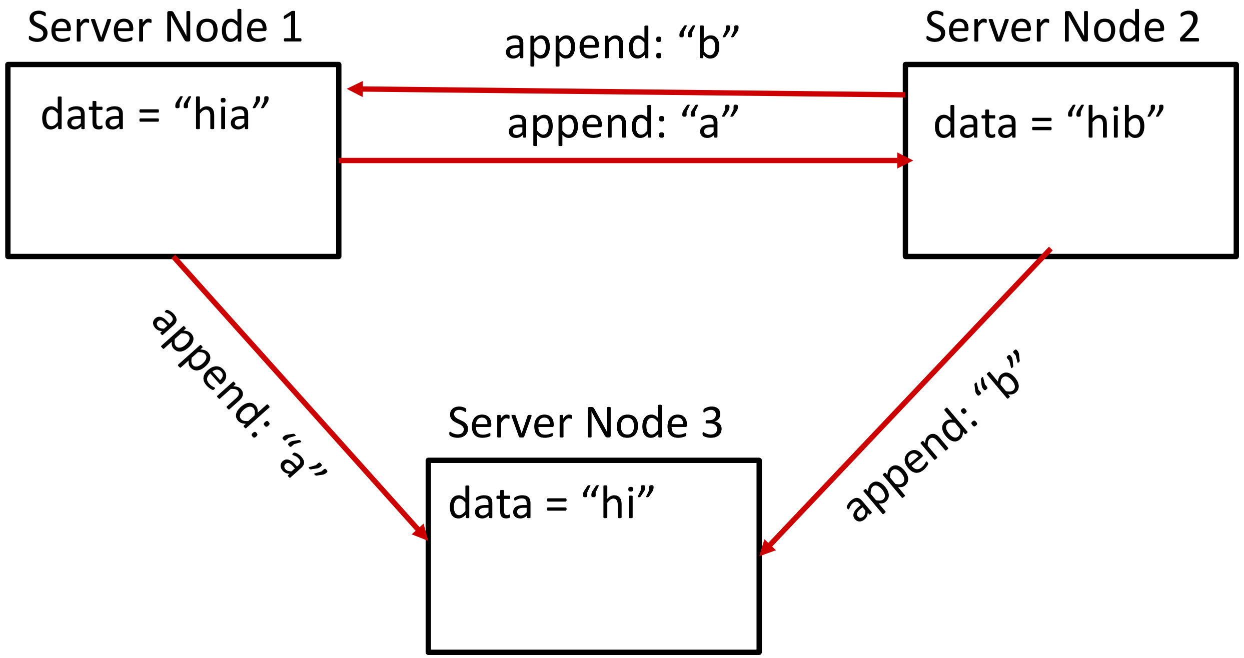
Back to Consistency in Distributed Systems

- Lets say server 1 gets a request to append “a” to the end of the string. How do we maintain a consistent state?
- Simple solution: send the message to other nodes and they acknowledge it



Back to Consistency in Distributed Systems

- Lets say that node 1 wants to append a but at the same time node 2 wants to append b
- Which happens first? How do we maintain a consistent state?



What happens first?

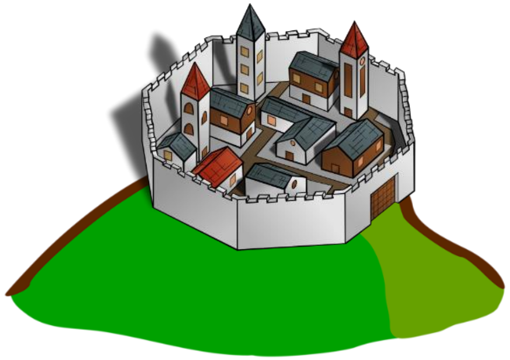
- Messages can get delayed when sent over the network
- Can we use a timestamp?
  - What if the computers clock is slightly off?
- Does TCP fix it?

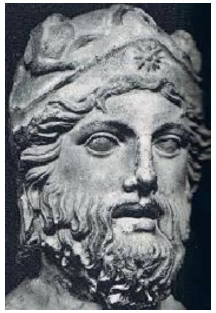
Lecture Outline

- **Intro to Distributed Systems**
  - Sequential Consistency
  - Logical Clocks & Ordering
  - **Fault Tolerance**
  - Performance

Generals Problem

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will both be defeated.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?





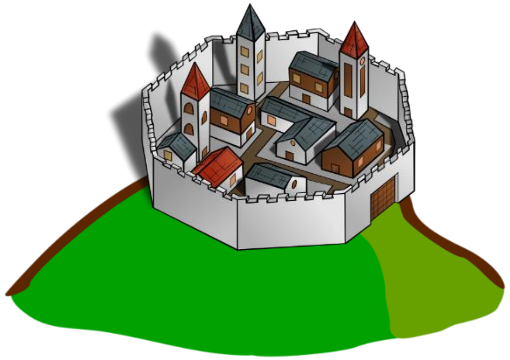
Generals Problem

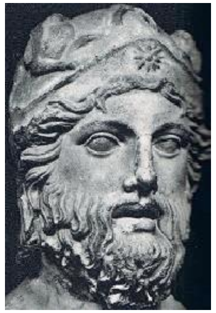
- To coordinate an attack, the problem requires common knowledge

- With the messengers, common knowledge is never reached.

- What happens when we add more generals?

- What happens when some of the generals are malicious?





Example: RPC

- Remote Procedure Call: When a program is able to invoke a function on another computers address space, and then get the results.

- Usually done as a form of “Message Passing”
  - Client calls a function that sends a “message” over the network
  - A server receives the message, executes the function, and sends the response back

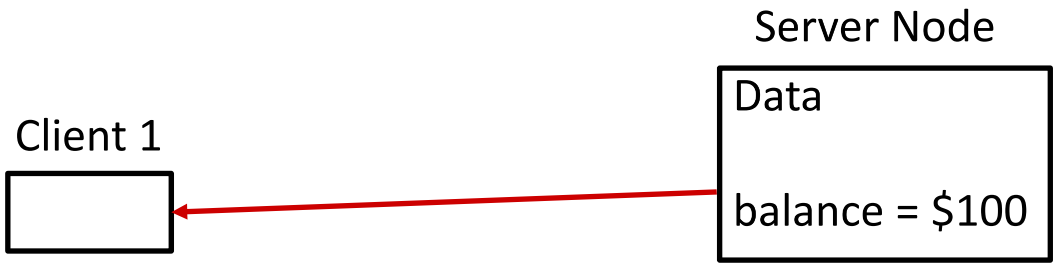
- Even in this simple, example, issues can arise

Example: RPC

- Consider: Client wants to read their current Bank Account Balance
  - Client may call a function like get_balance()

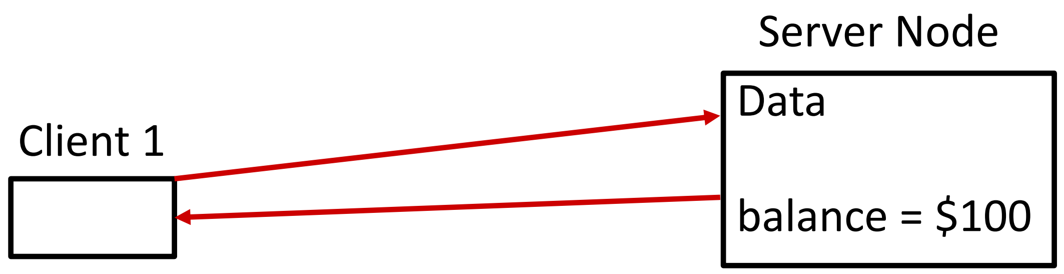
Example: RPC

- Consider: Client wants to read their current Bank Account Balance
  - Client may call a function like get_balance()
  - get_balance() will reach out to the server across the network



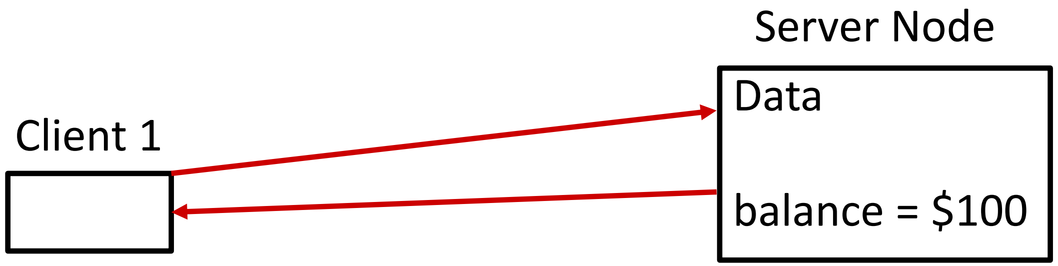
Example: RPC

- Consider: Client wants to read their current Bank Account Balance
  - Client may call a function like get_balance()
  - get_balance() will reach out to the server across the network
  - Server processes the request, and sends it back



Example: RPC

- Consider: Client wants to read their current Bank Account Balance
  - Client may call a function like get_balance()
  - get_balance() will reach out to the server across the network
  - Server processes the request, and sends it back
  - Client returns from the function "get_balance()"



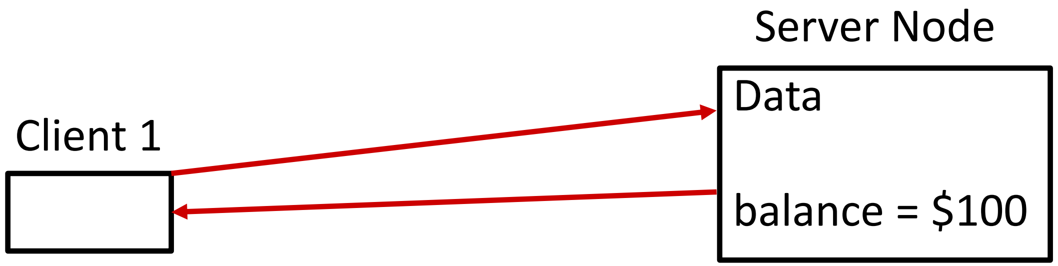
Client was blocked while waiting for the server to respond.

Program that called **`get_balance()`** probably doesn't need to know much about the network messaging

Blank Slide

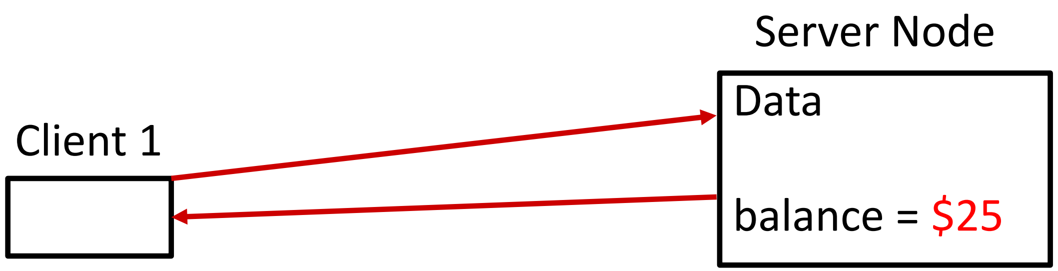
Example: RPC Transaction

- Consider: Client wants to withdraw $75 from their bank account
  - Client may call a function like withdraw(75)
  - withdraw() will reach out to the server across the network



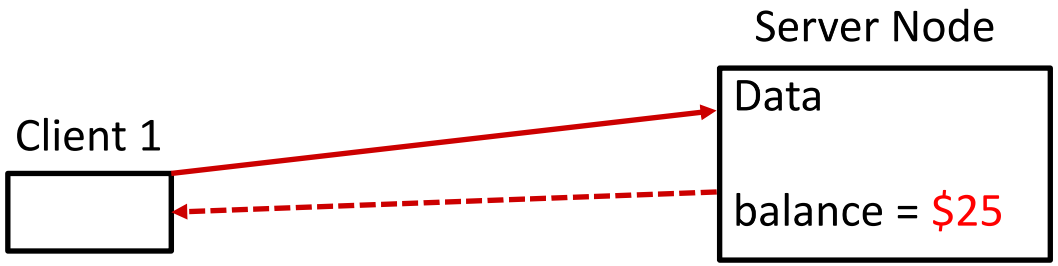
Example: RPC Transaction

- Consider: Client wants to withdraw $75 from their bank account
  - Client may call a function like withdraw(75)
  - withdraw() will reach out to the server across the network
  - Server processes the request, and sends it back



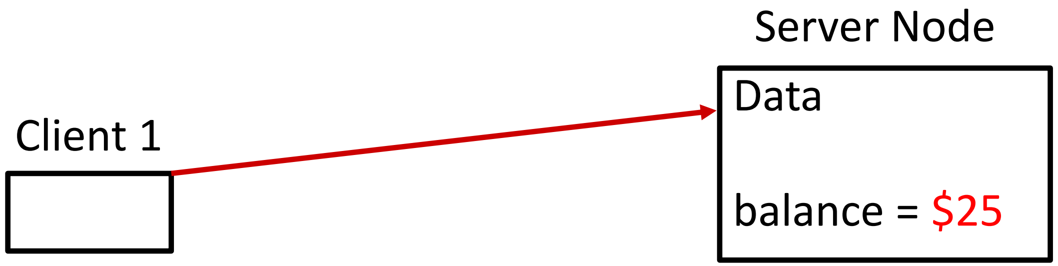
Example: RPC Transaction

- Consider: Client wants to withdraw $75 from their bank account
  - Client may call a function like withdraw(75)
  - withdraw() will reach out to the server across the network
  - Server processes the request, and sends it back
    - … But what if the connection is dropped before client receives response!



Example: RPC Transaction

- Server processes the withdraw request, and sends it back
  - … But what if the connection is dropped before client receives response!
- Let's say connection is re-established and client resends "withdraw(75)"…



Question: Does TCP Solve This?

- If we were using TCP, is this situation even possible?
  - TCP: provides an abstraction of a reliable stream of bytes.
  - TCP: each packet is acknowledged between user and receiver and automatically resent.

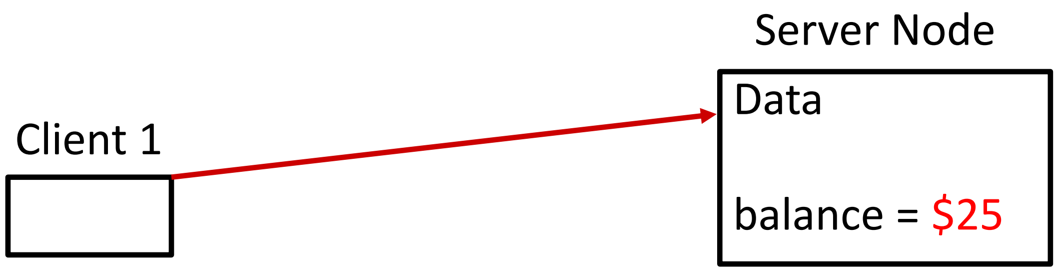
Question: Does TCP Solve This?

- If we were using TCP, is this situation even possible?
  - TCP: provides an abstraction of a reliable stream of bytes.
  - TCP: each packet is acknowledged between user and receiver and automatically resent.

- Yes: this can still happen.
  - TCP Ensures that packets are sent in a specific order and are acknowledged before it is "successfully written".
  - Does not ensure that the network (or server itself) goes down
  - Does not ensure that the function we want to execute on the server worked or whether it actually happened.

Example: RPC Transaction

- Server processes the withdraw request, and sends it back
  - … But what if the connection is dropped before client receives response!
- Let's say connection is re-established and client resends "withdraw(75)"…
  - How does the server know if this is the same request as last time, or another request to withdraw $75
  - How does the server know what the client is "intending"



Terminology

- Exactly Once:
  - Hardest to guarantee
  - That something happens and it only happens exactly one time.
  - Requires that the clients have an ID and each request has an ID number.
  - Servers must also keep a history of previously processed requests and their ID number so that the server can respond to duplicate/old requests.

Terminology

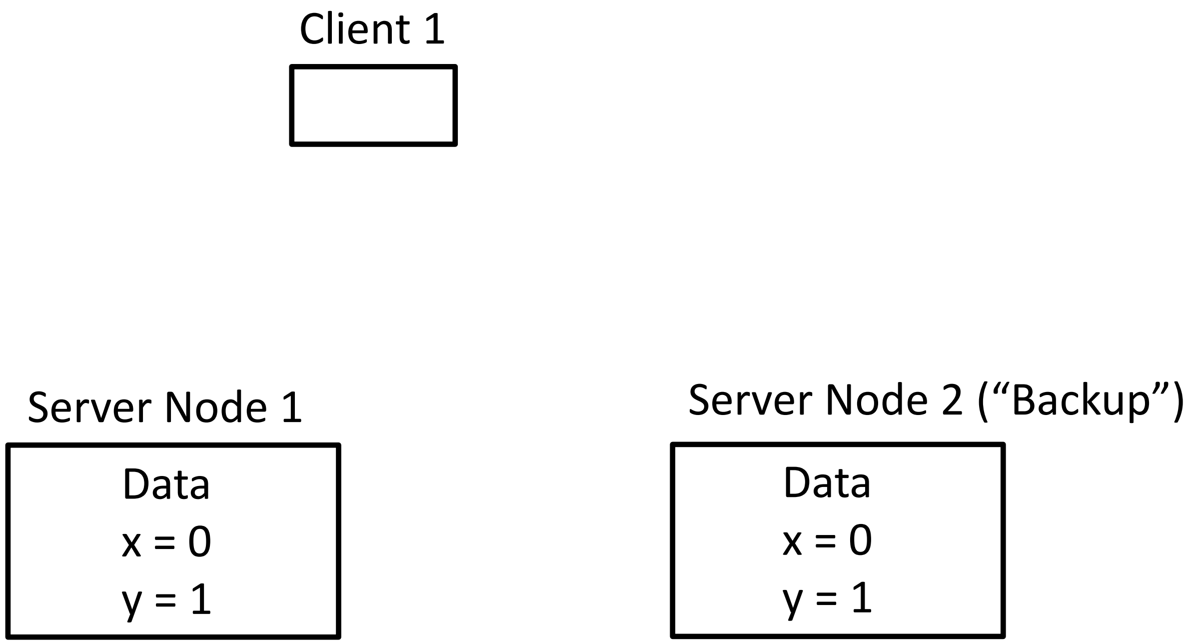
- At Most Once:
  - That a request is executed at most once (e.g. 0 times or 1 time)
  - Usually means the client sends the request once and only once.
  - Usable in some cases, but sometimes we need to guarantee that something happened.
- At Least Once:
  - That the thing is executed at least one time.
  - This is fine for things like “Reading a value” or “setting” a value
    Other operations may get different results if done multiple times
    (Like our transaction)

Fix?

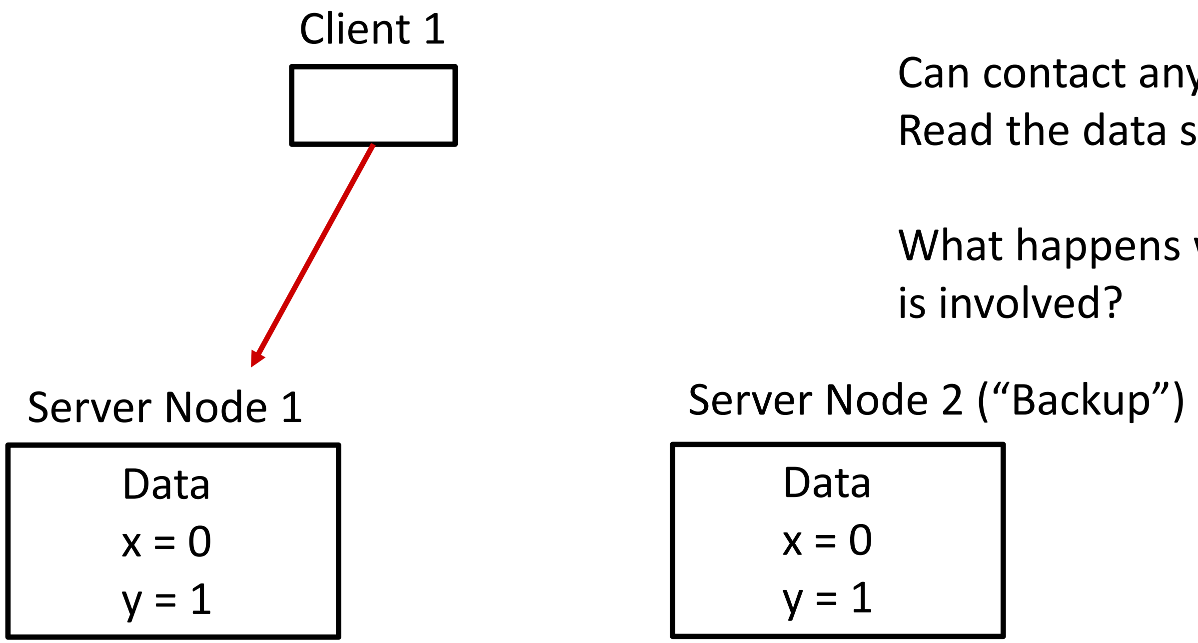
- How do we ensure that each transaction is done exactly once?
  - Thoughts?

Blank Slide

Example: Consistent State



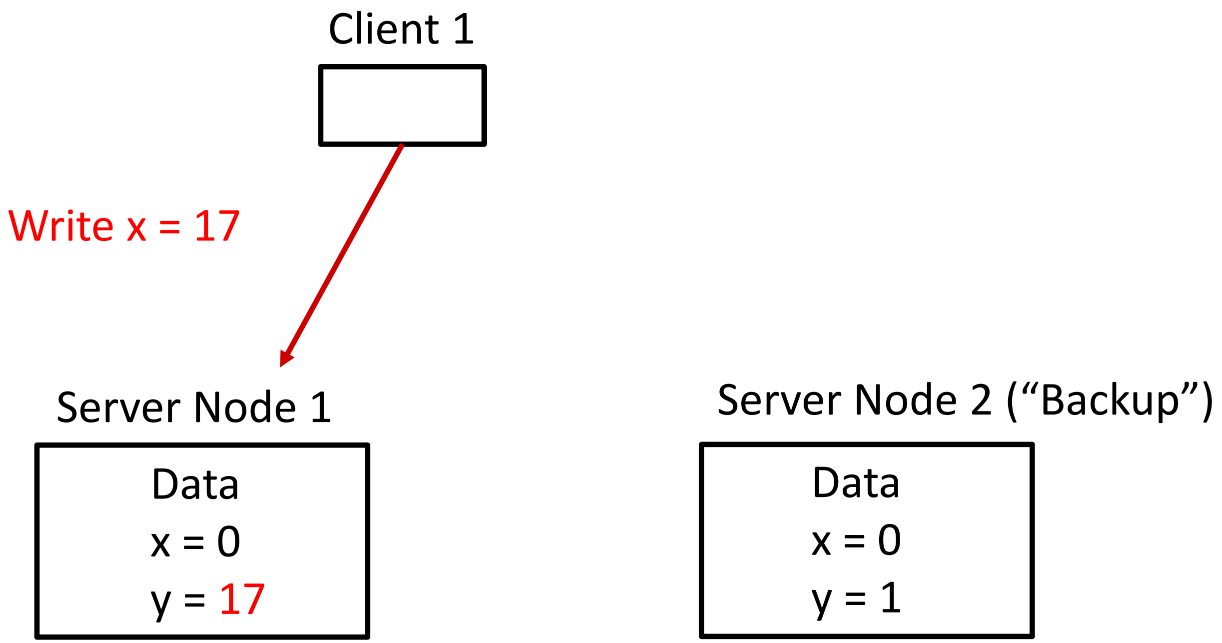
Example: Consistent State



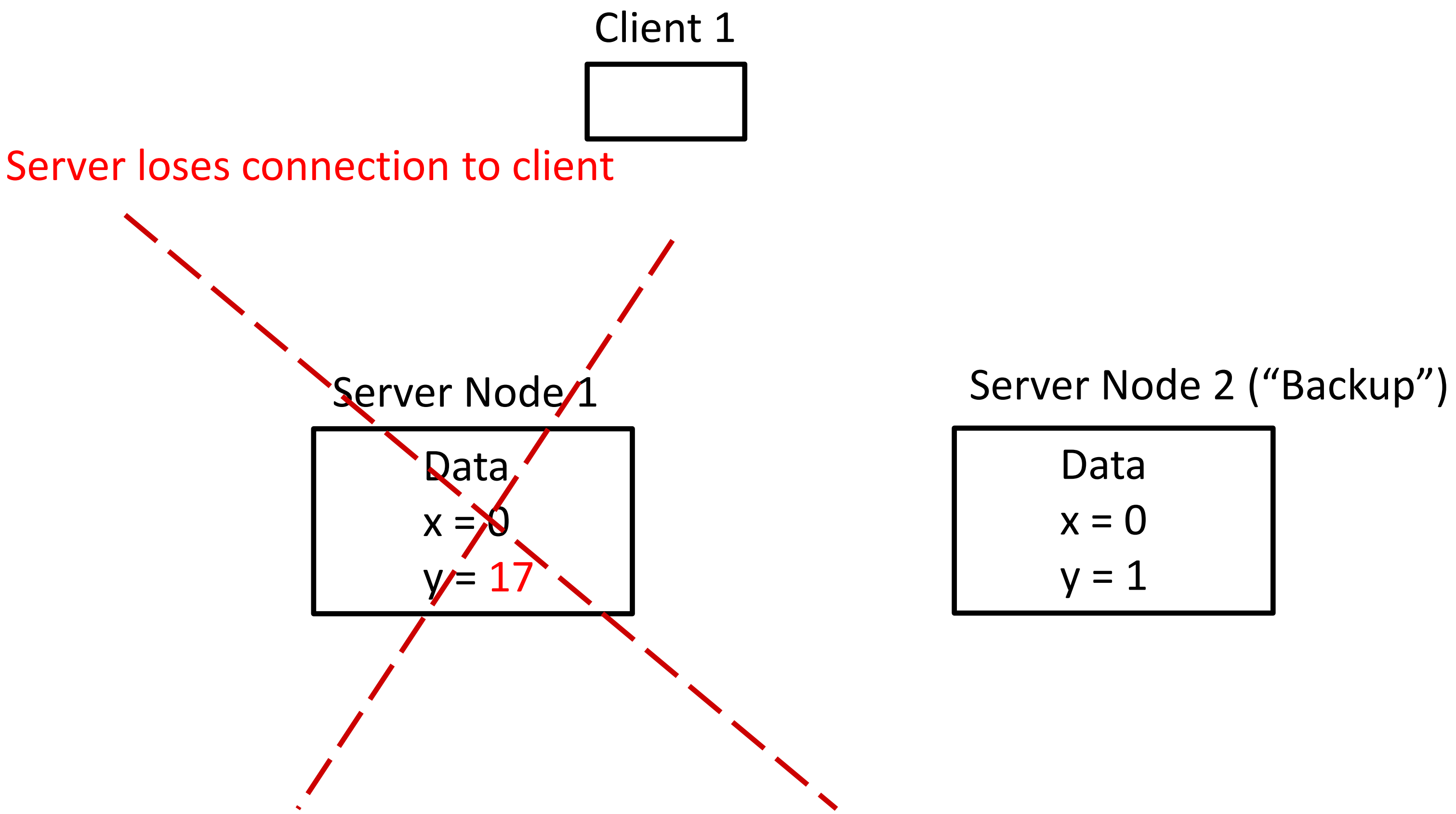
Can contact any node to
Read the data stored

What happens when writing
is involved?

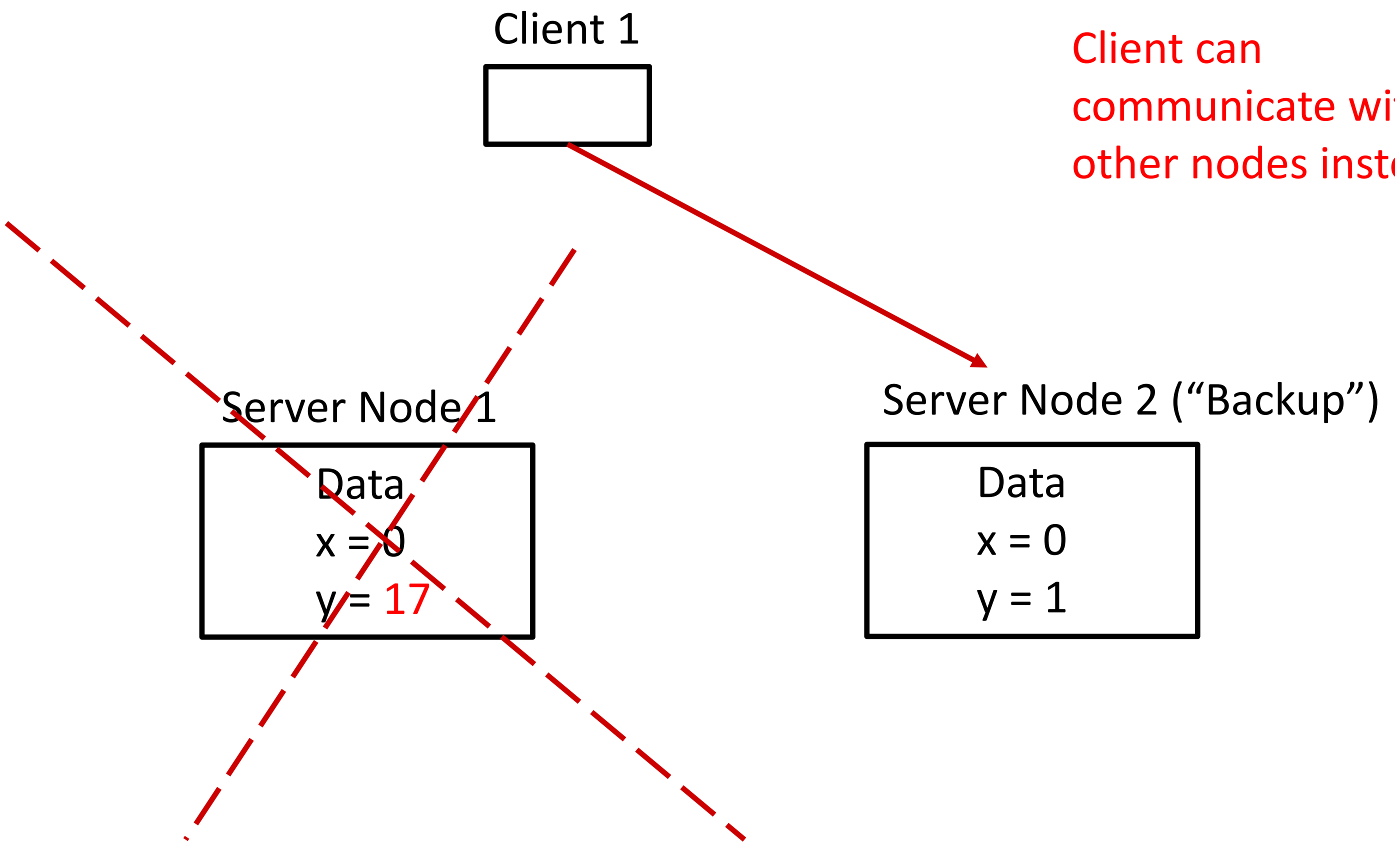
Example: Consistent State



Example: Consistent State

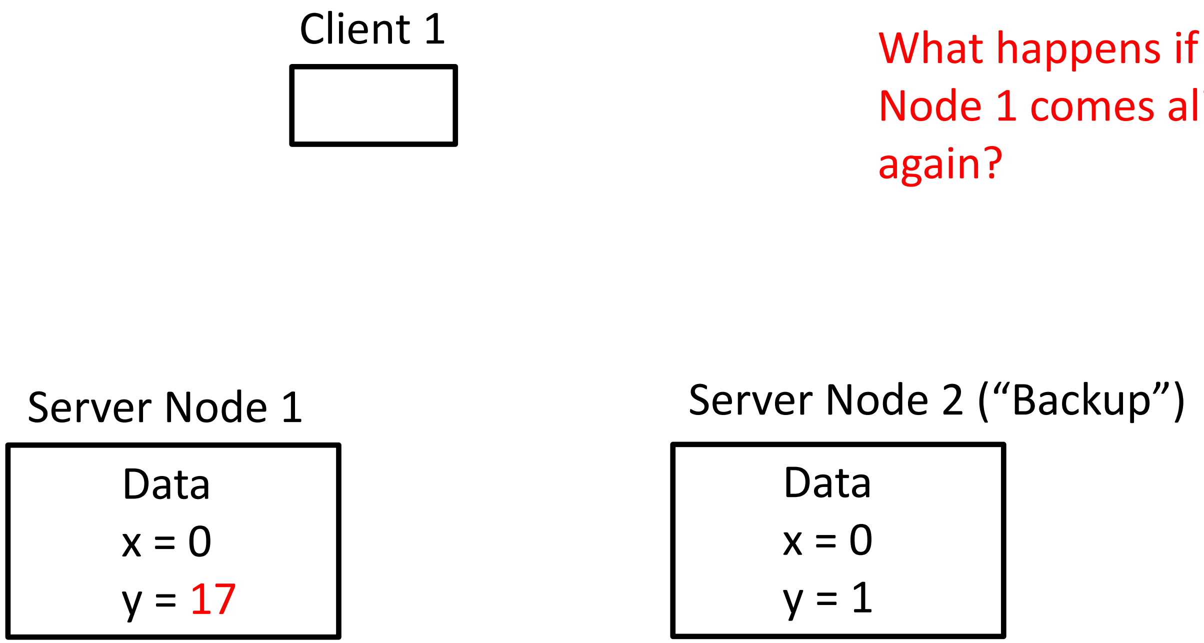


Example: Consistent State



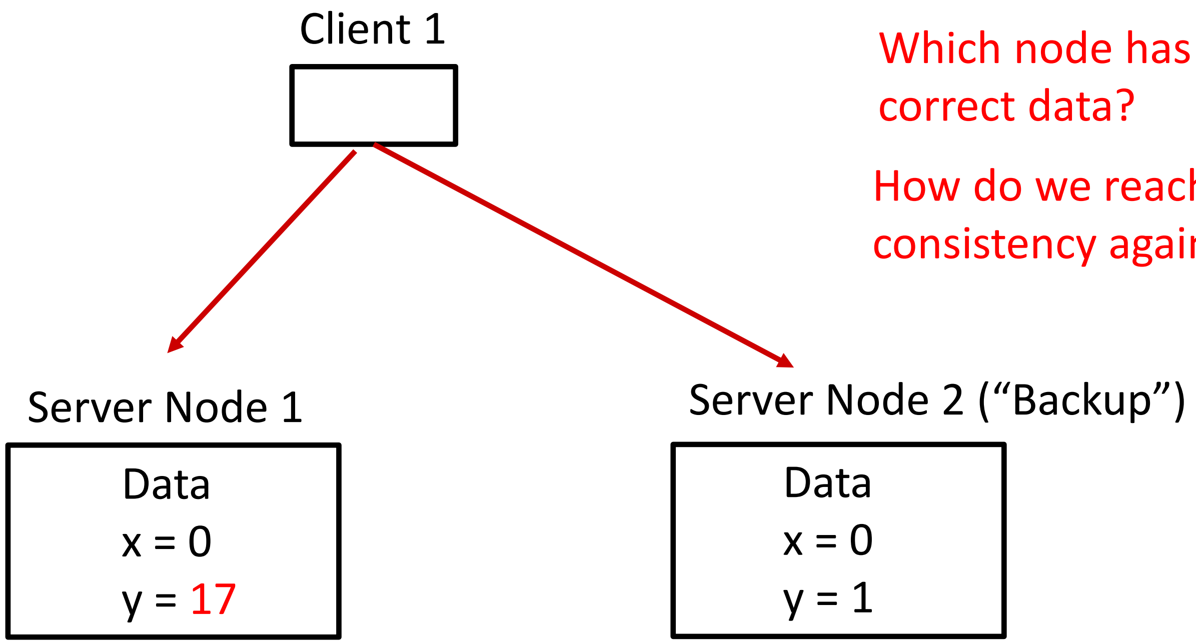
Client can communicate with other nodes instead

Example: Consistent State



What happens if Node 1 comes alive again?

Example: Consistent State



Which node has the correct data?

How do we reach consistency again?

PAXOS

- No deterministic fault-tolerant consensus protocol can guarantee progress in an asynchronous network.
- PAXOS is a protocol for solving consensus while being **resistant** to **unreliable** or **failable** processors in the system
  - Unreliable and failable could mean just that
    - the system crashes
    - packet (messages) are being sent and received inconsistently
    - Becomes malicious and behaves incorrectly “on purpose”
    - And in paxos, could possibly recover from any of these
- Paxos guarantees consistency, and the conditions that could prevent it from making progress are difficult to provoke.

Lecture Outline

- **Intro to Distributed Systems**
  - Sequential Consistency
  - Logical Clocks & Ordering
  - Fault Tolerance
  - **Performance**

Performance

- Taking a step back from fault tolerance
- Another concern with doing actions across a distributed system is trying to make efficient utilization of the nodes in the system

- If we have a large task, how do we split up the work roughly evenly across nodes in the network so that it is completed faster?
  - Avoid having one “coordinator” node if possible
    - Then nodes may have to wait for the coordinator to tell them what to do and there is less coordinators)
  - Try to treat the nodes equally like rational actors so that they can all do work at the same time.

Microsoft Interview Question:

- 100 Nodes in a cluster of computers
- Each Node is numbered 0 through 99
- Each node has 1,000,000 integers
  - Each node can only hold a little more than a 1,000,000 integers
- We want to sort all the numbers so that node 0 contains the first 1% of the integers in sorted order (the lowest million integers). Node 1 contains the next million lowest integers, etc.

- How do we do this efficiently?

This was just an “intro” to the field ☺

- Lots of details left out, but these concepts apply to distributed systems.
- If a bank or database runs on a collection of nodes. How do we agree on whether a transaction occurred?
  - How do we ensure that the transaction went through and won’t get “lost” due to faults?
- What if data was split across different nodes and multiple clients needed data from multiple nodes at the same time?