Distributed Systems Intro

Computer Systems Programming, Spring 2023

Instructor: Travis McGaha

TAs:

Kevin Bernat	Jialin Cai
Mati Davis	Donglun He
Chandravaran Kunjeti	Heyi Liu
Shufan Liu	Eddy Yang

Poll Everywhere

pollev.com/tqm

- Any questions from previous lectures?

Logistics

- HW4 Posted — Due Thursday 4/20 @ 11:59
  Extended to 4/26 @11:59 pm

- Project Released! — Due Wednesday 4/26 @ 11:59

- HW2 grades & Midterm grades posted
  - Can fix HW2 submissions
  - Midterm has regrades & the clobber policy

Logistics

- Final Exam Scheduling:
  - 96 hours (4 days)
  - Opens Tuesday May 2nd @ Noon
  - Closes Saturday May 6th @ noon

- Office hours next week
  - MTW will happen as normal, I will discuss with Tas about the others

Lecture Outline

- **Intro to Distributed Systems**

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 state (e.g. memory)
  - Individual computers (nodes) can fail
  - The network itself can fail (Drop messages, corrupt messages, delay messages, etc.)

Why do we care?

- They are a really interesting problem to work with

- Most applications we interact with are distributed systems

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

Distributed Systems Topics

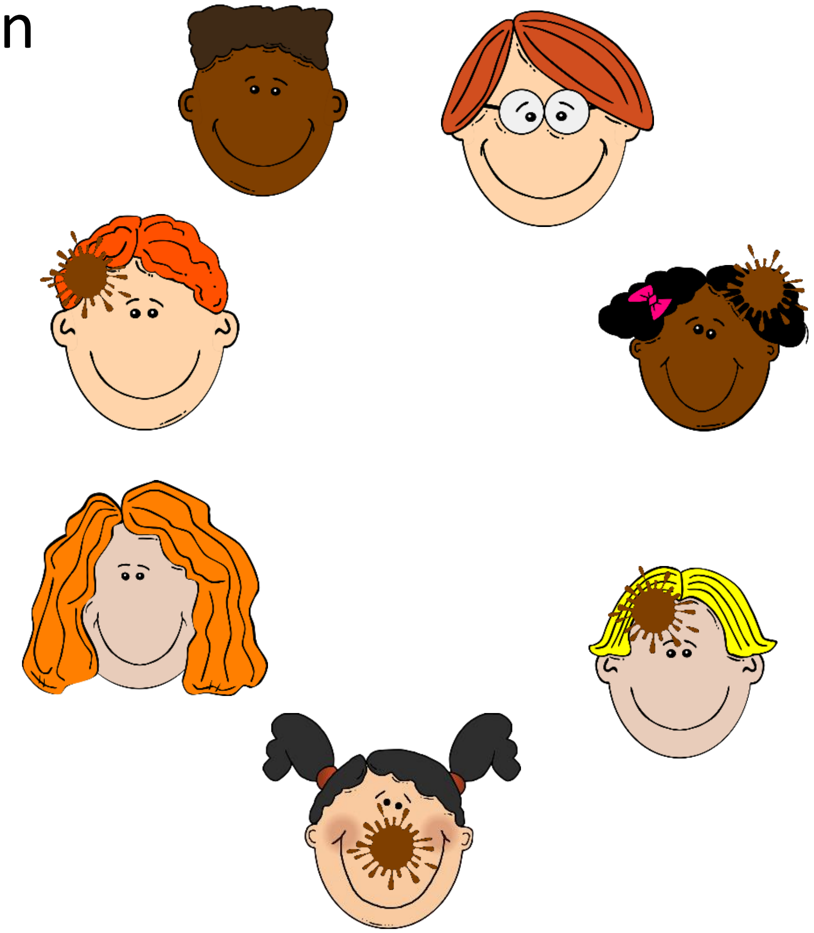
- Concurrency on a single node
  - Threads, processes, pipes, locks, etc.
- Networking
  - HTTP, DNS, TCP, Sockets, etc.
- Synchronization across network nodes
  - Common Knowledge, Clocks, coordination, leader elections, etc.
- Fault Tolerance & Robustness
  - Byzantine fault tolerance, ACID, etc.

Distributed Systems Topics

- Concurrency on a single node
  - Threads, processes, pipes, locks, etc.
- Networking
  - HTTP, DNS, TCP, Sockets, etc.
- **Synchronization across network nodes**
  - **Common Knowledge, Clocks, coordination, leader elections, etc.**
- **Fault Tolerance & Robustness**
  - **Byzantine fault tolerance, ACID, etc.**

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"

Muddy Foreheads

- **Answer**: On the kth round, all of the muddy children know they have mud on their forehead, raise their hands

- **"Proof" by induction on k**. When k=1, the muddy child knows no other child is muddy, must be muddy themself. When k=2, on the first round, both muddy children see each other, cannot conclude they themselves are muddy. But after neither raises their hand, they realize there must be two muddy children, raise their hand.

- In general, when k>1, after round k-1, if there were k-1 muddy foreheads, all of those children would have raised their hands (by induction). Therefore, each muddy child knows they're muddy and raises their hand on the kth round

The Muddy Forehead "Paradox"

- If k > 1, the teacher didn’t say anything anyone didn’t already know!

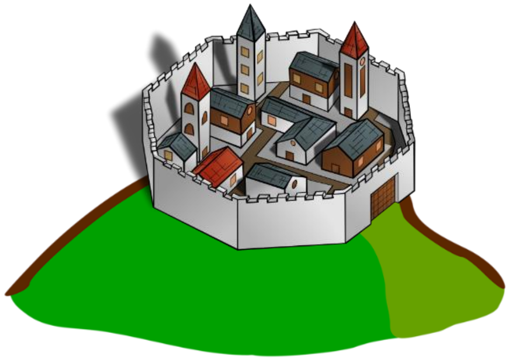
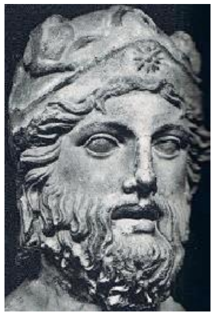
- Yet the information is crucial to let the children solve the problem

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

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?

Cooridnated Attack

- **Answer**: There does not exist a protocol to decide when and whether to attack.
- **Proof by contradiction**. Assume a protocol exists. Let the minimum number of messages received in any terminating execution be n. Consider the last message received in one such execution.
- The sender's decision to attack does not depend on whether or not the message is received; sender must attack. Since the sender attacks, the receiver must also attack when the message is not received.
- Therefore, the last message is irrelevant, and there exists an execution with n-1 message deliveries. n was the minimum! Contradiction.

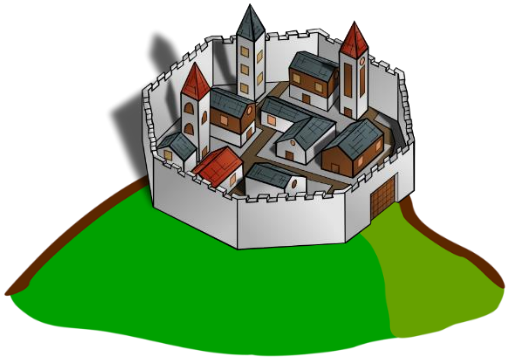
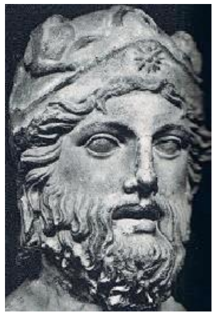
Generals Problem

- To coordinate an attack, the problem requires common knowledge

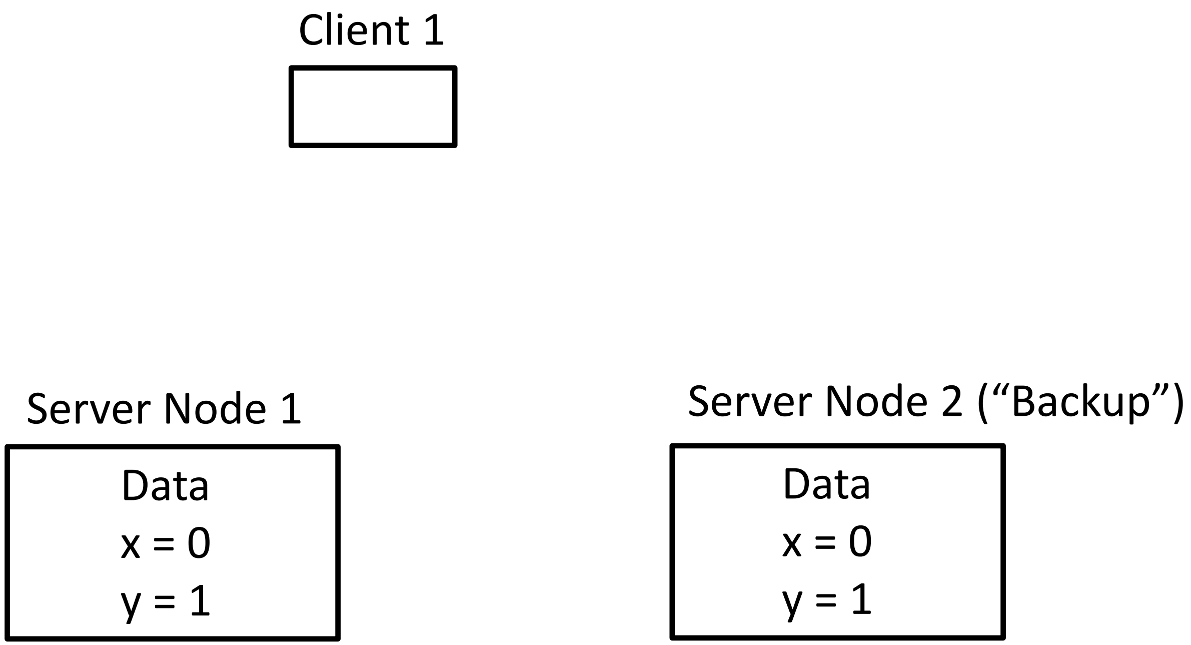
- With the messengers, common knowledge is never reached.

- What happens when we add more generals?

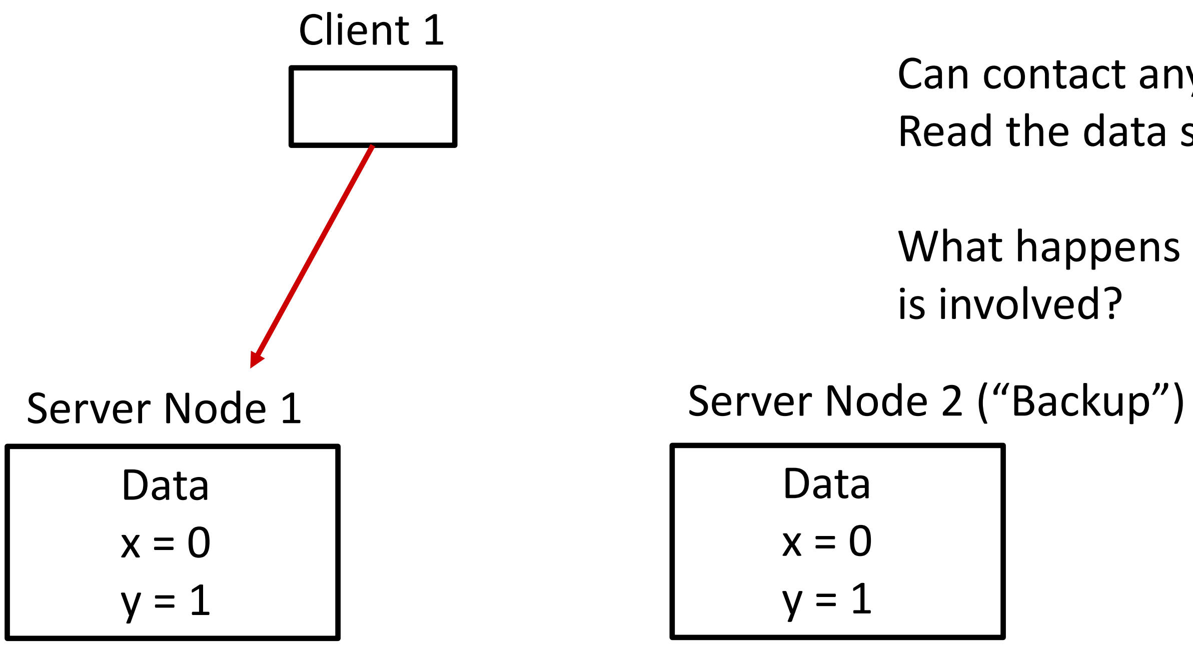
- What happens when some of the generals are malicious?

Examples


Client 1
Server Node 1
Data
x = 0
y = 1
Server Node 2 ("Backup")
Data
x = 0
y = 1

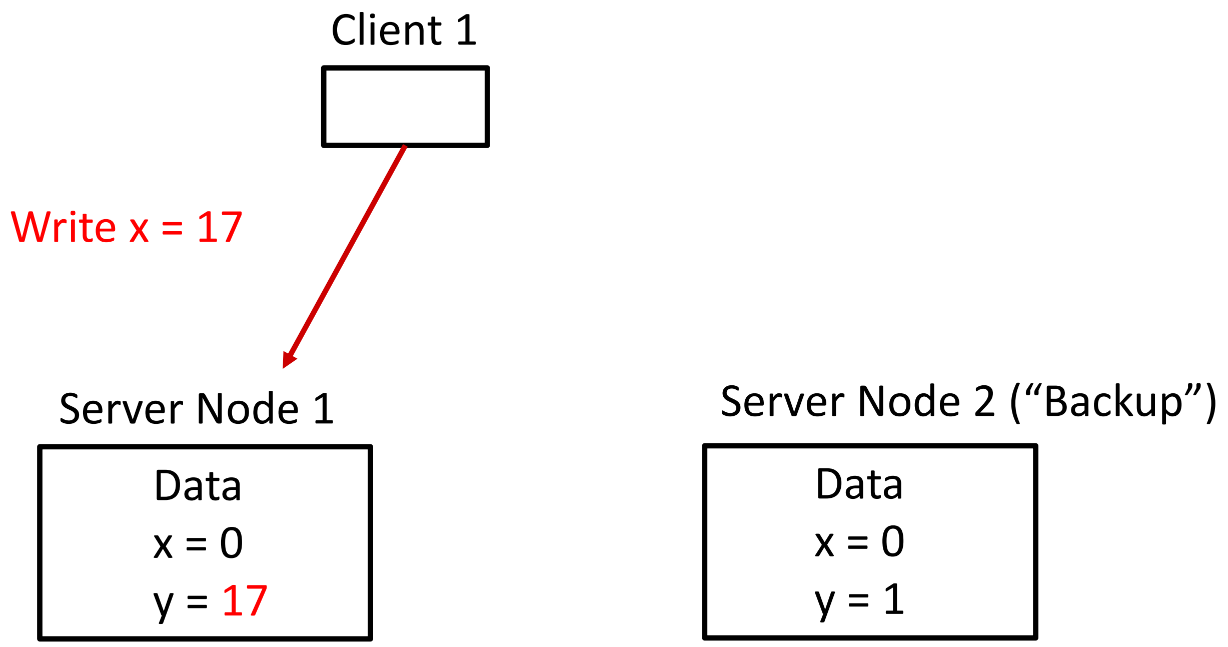
Examples



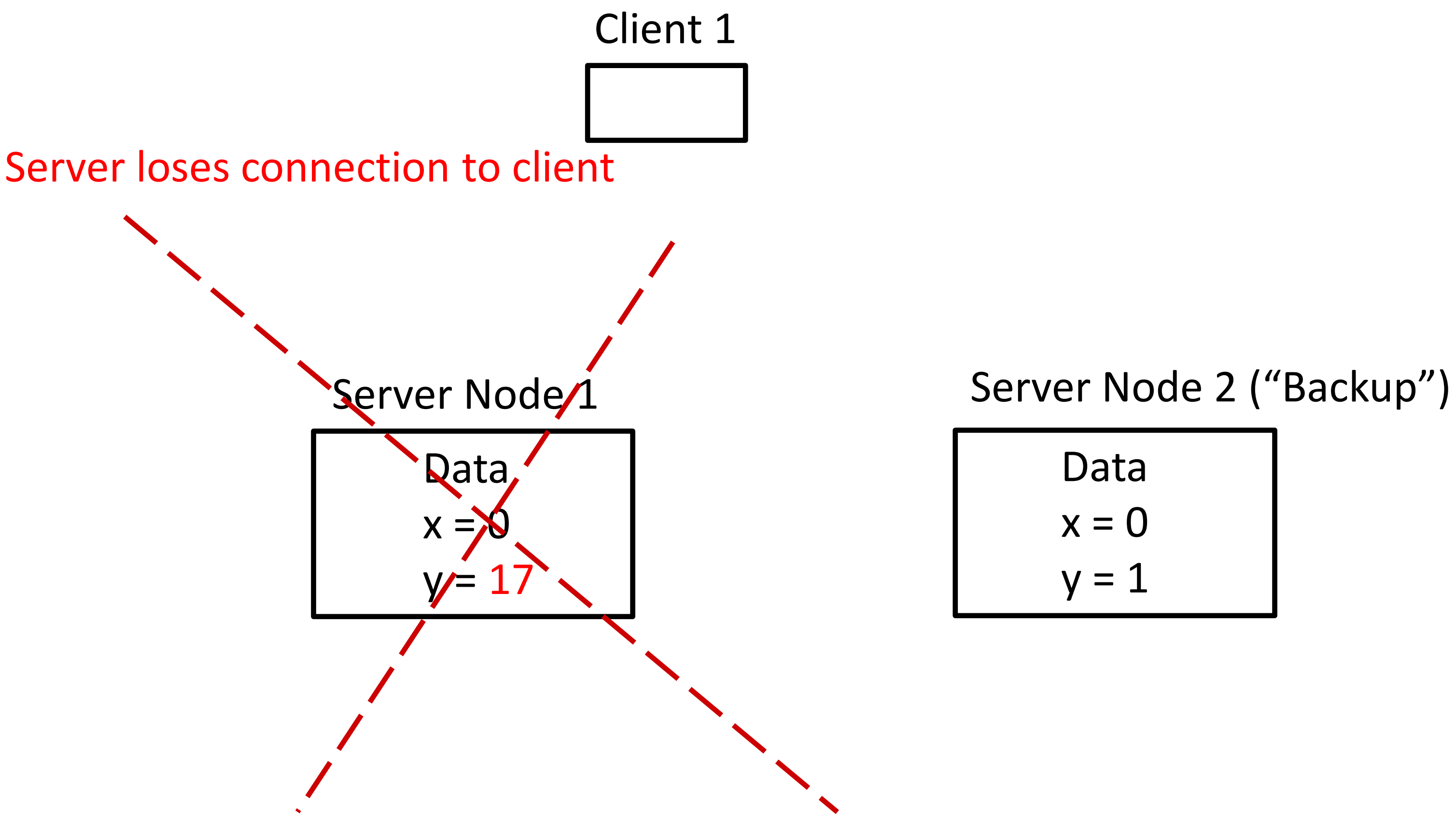
Can contact any node to Read the data stored

What happens when writing is involved?

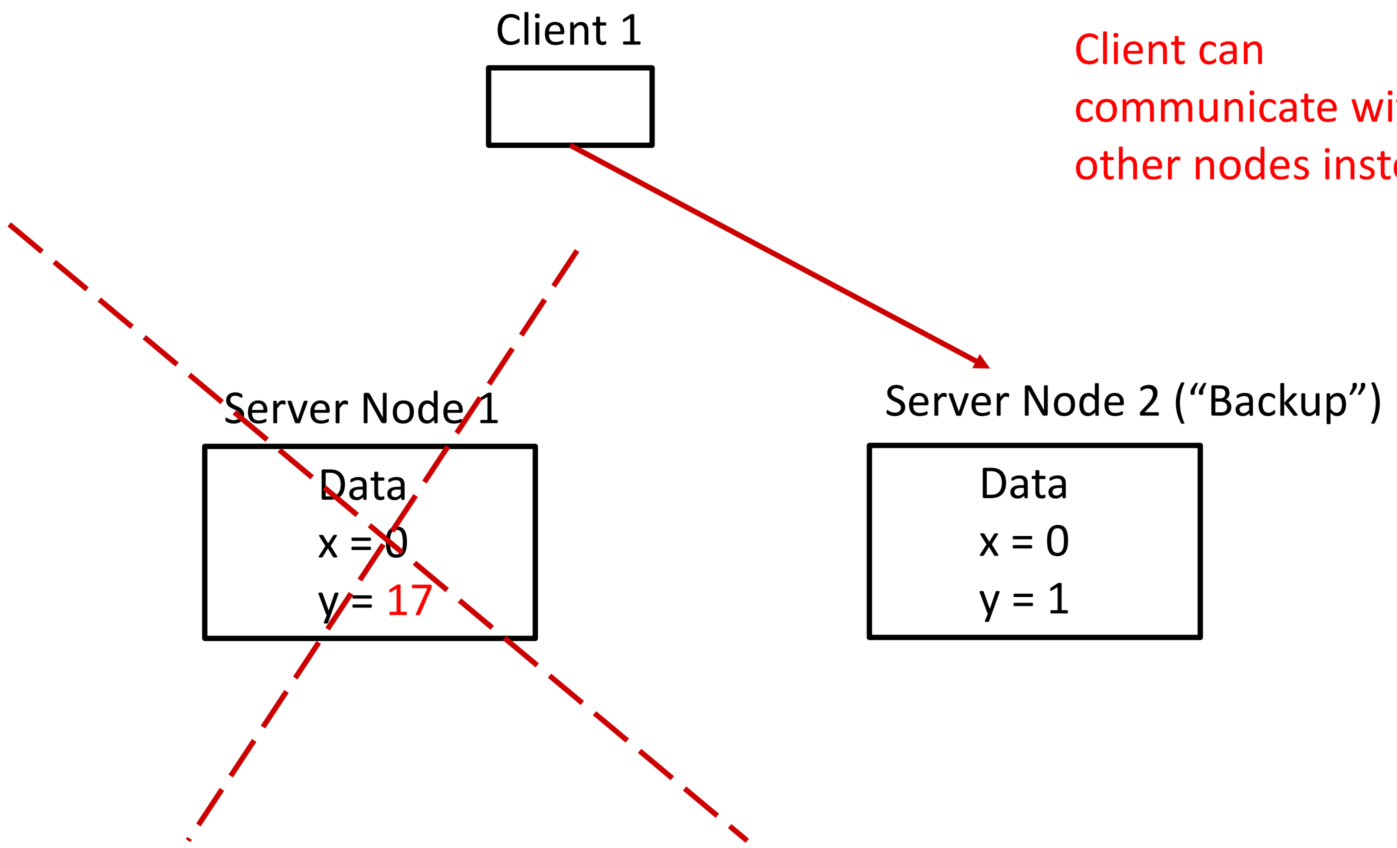
Examples


Client 1
Write x = 17
Server Node 1
Data
x = 0
y = 17
Server Node 2 ("Backup")
Data
x = 0
y = 1

Examples

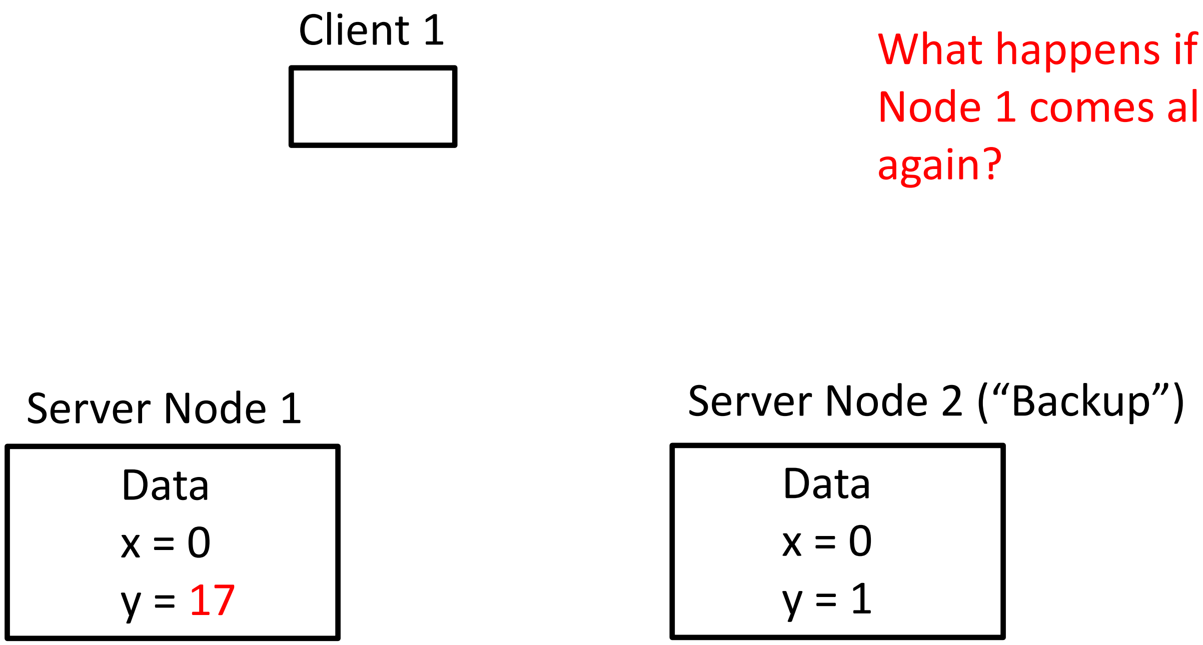

Client 1
Server loses connection to client
Server Node 1
Data
x = 0
y = 17
Server Node 2 (“Backup”)
Data
x = 0
y = 1

Examples



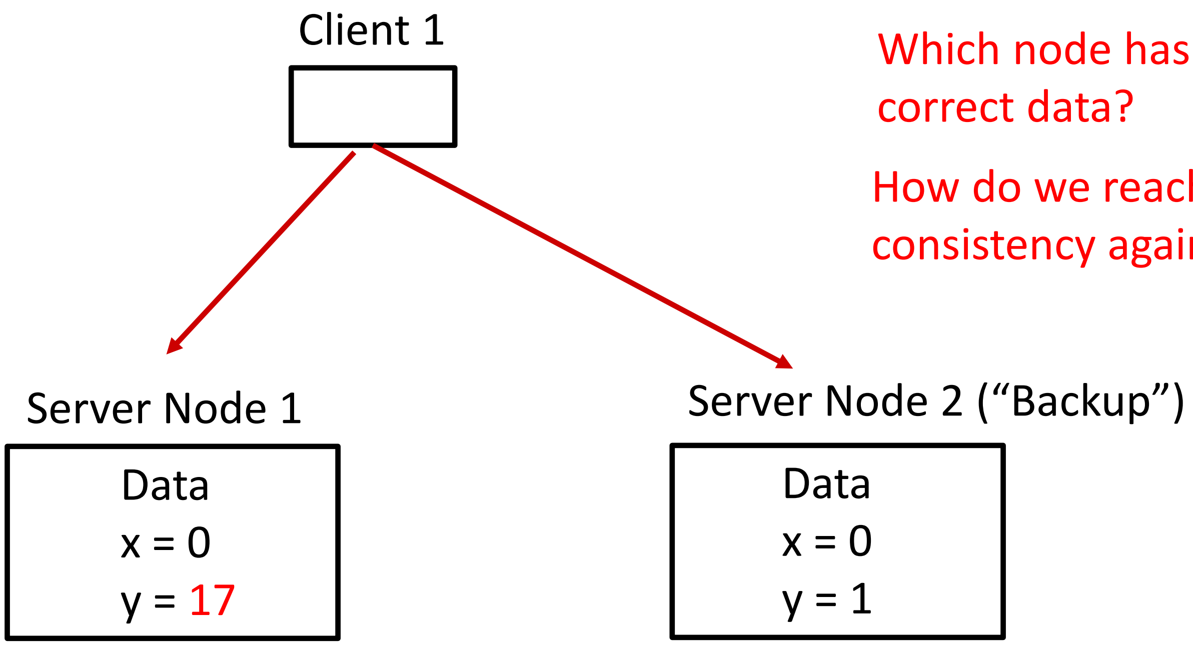
Client can communicate with other nodes instead

Examples



What happens if Node 1 comes alive again?

Examples



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.

Real Life Equivalents

- While what we went over aren’t “real” examples, 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?