CC5212-1
Procesamiento Masivo de Datos
Otoño 2016

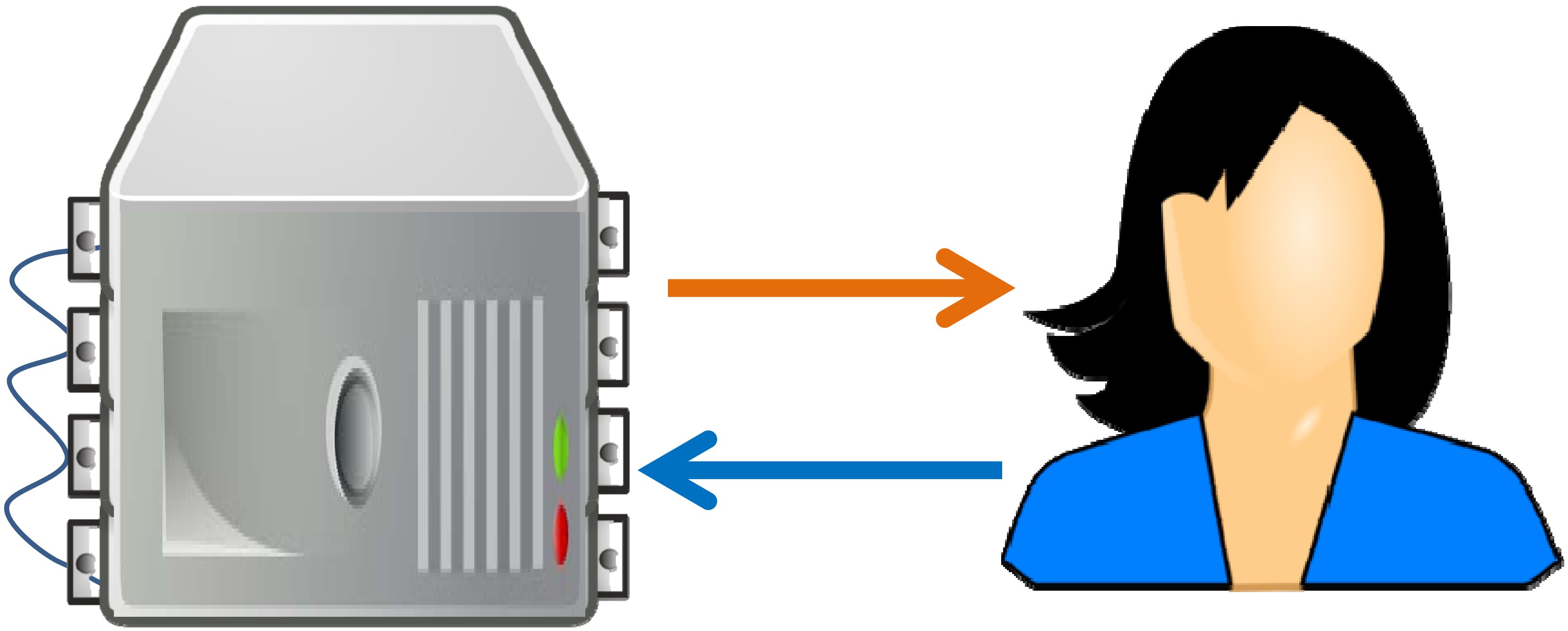
Lecture 3: Distributed Systems II

Aidan Hogan
aidhog@gmail.com

TYPES OF
DISTRIBUTED SYSTEMS …

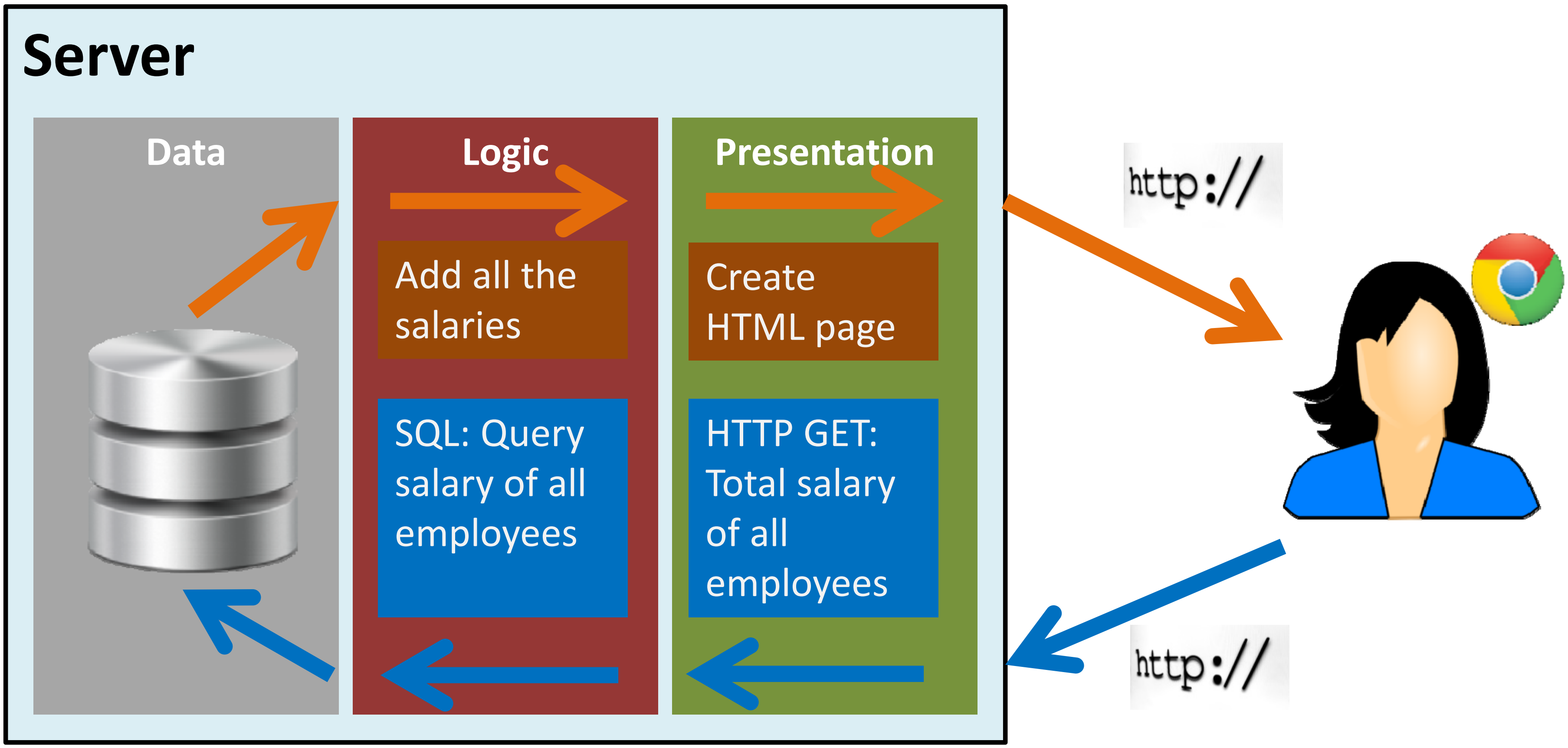
Client–Server Model

- Client makes request to server
- Server acts and responds

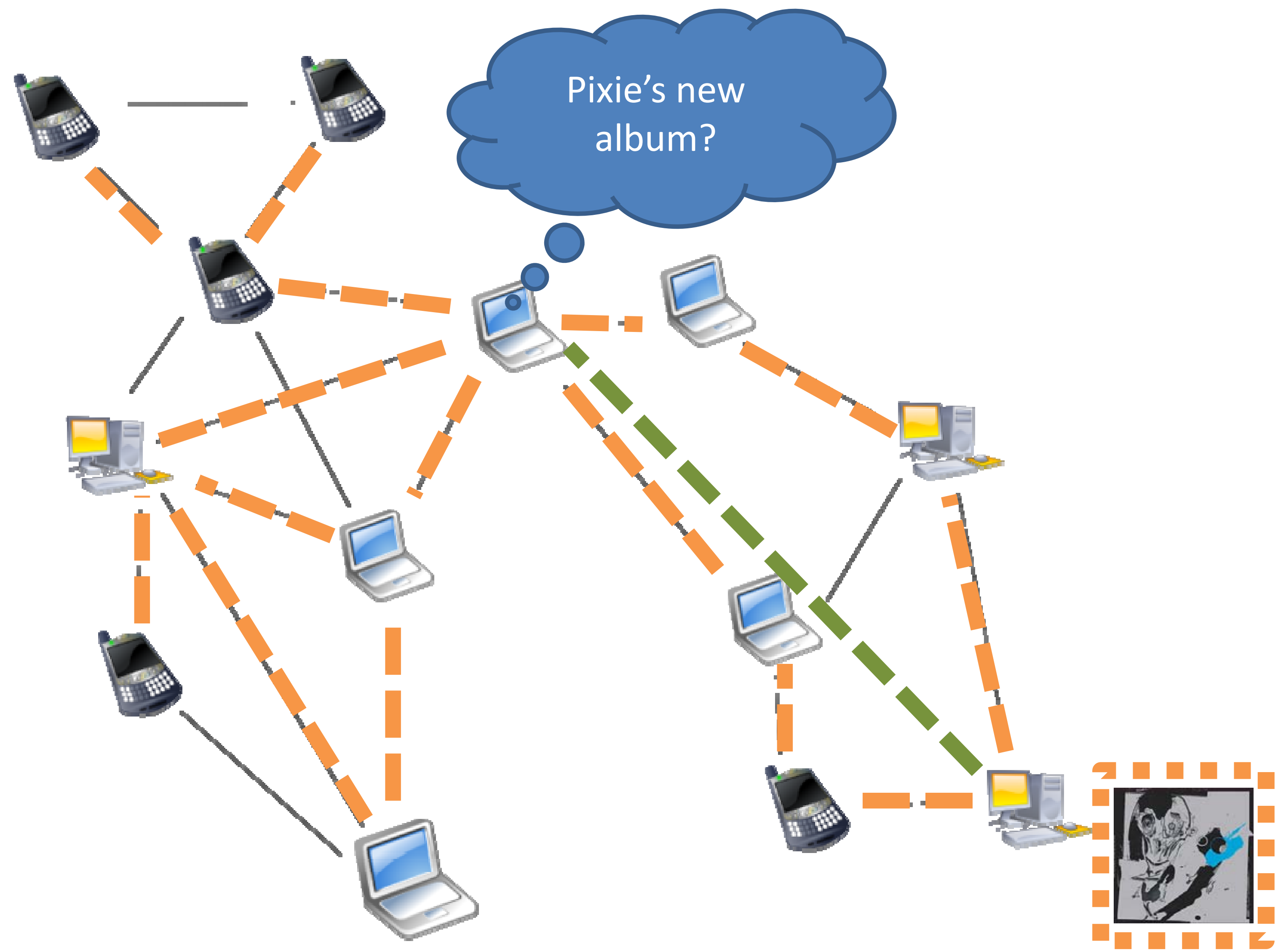


(<u>For example</u>: *Email, WWW, Printing, etc.*)

Client–Server: *Three-Tier Server*



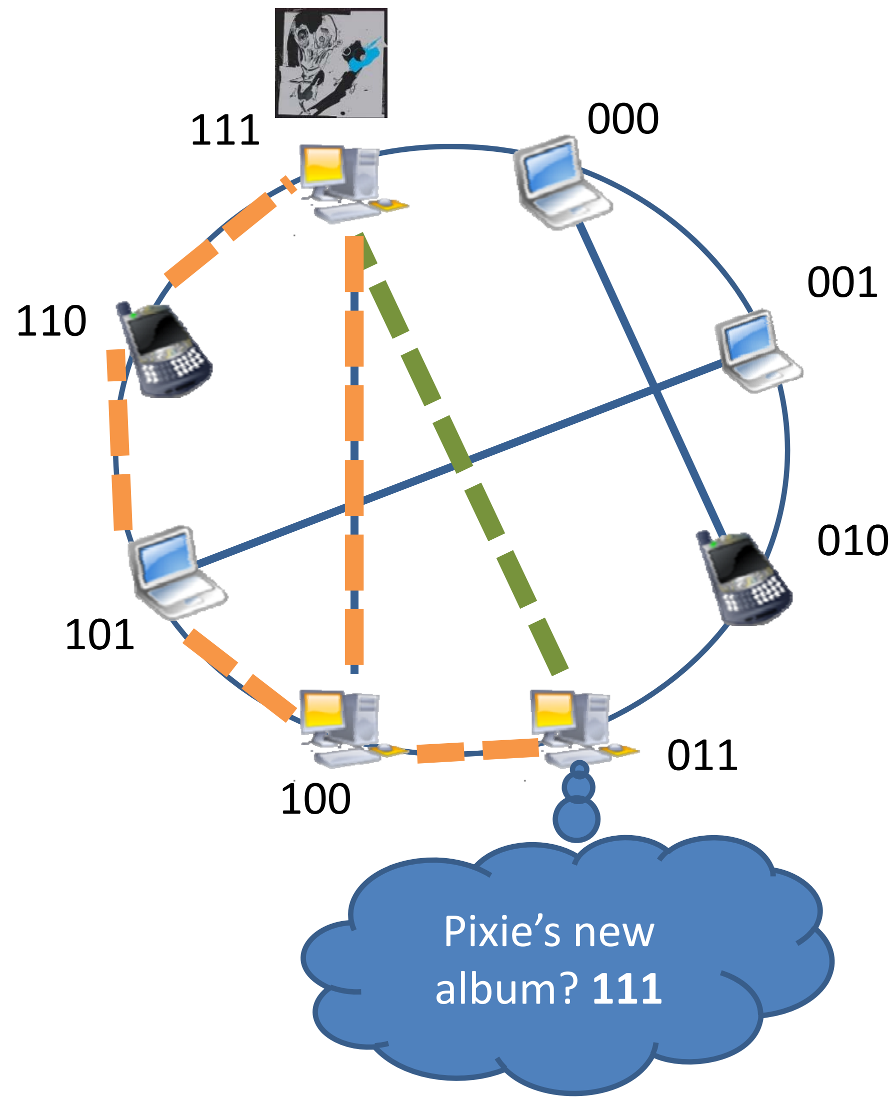
Peer-to-Peer: *Unstructured*



(<u>For example</u>: *Kazaa, Gnutella*)

Peer-to-Peer: *Structured (DHT)*

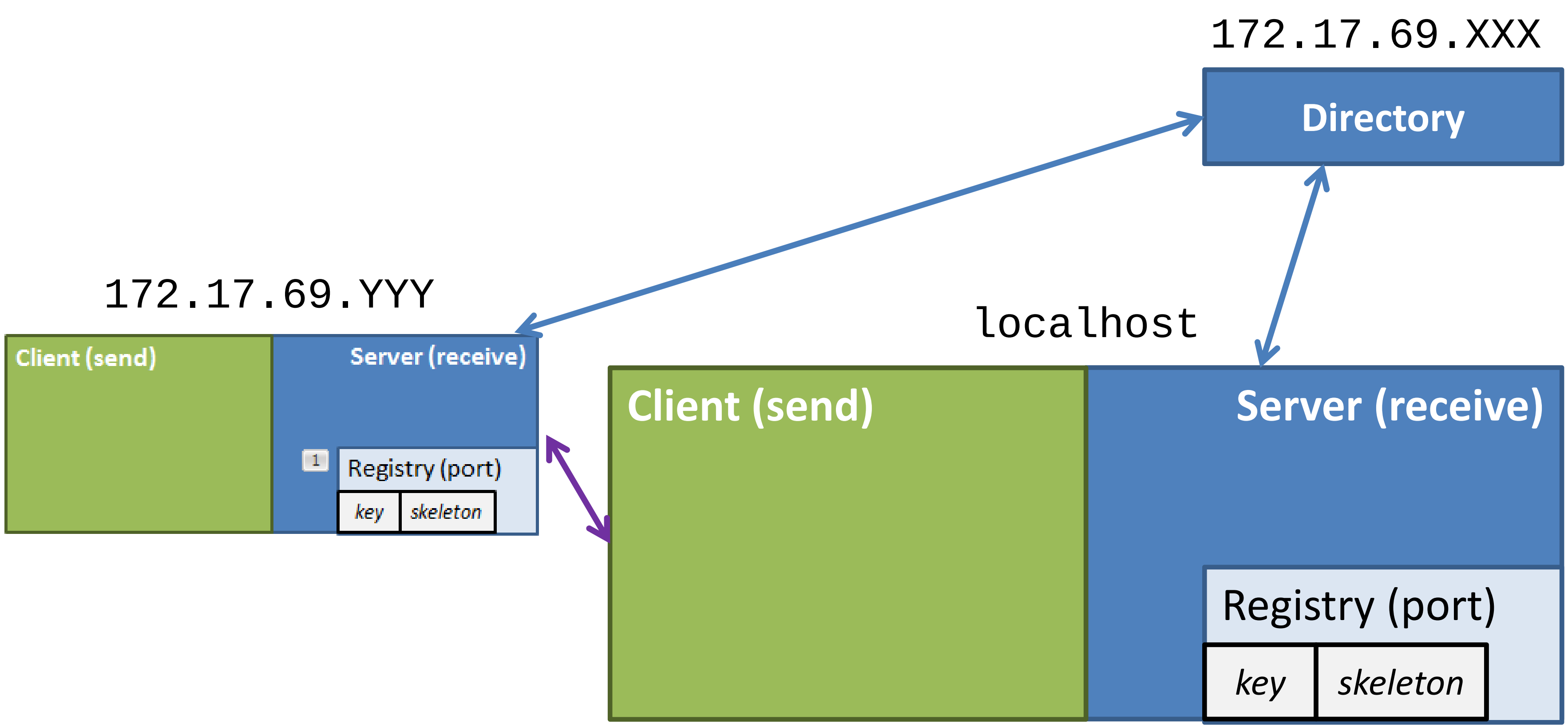
- Circular DHT:
  - Only aware of neighbours
  - O(*n*) *lookups*
- Implement shortcuts
  - Skips ahead
  - Enables binary-search-like behaviour
  - O(log(*n*)) *lookups*



Desirable Criteria for Distributed Systems

- Transparency:
  - Appears as one machine
- Flexibility:
  - Supports more machines, more applications
- Reliability:
  - System doesn’t fail when a machine does
- Performance:
  - Quick runtimes, quick processing
- Scalability:
  - Handles more machines/data efficiently

Java RMI in the lab …



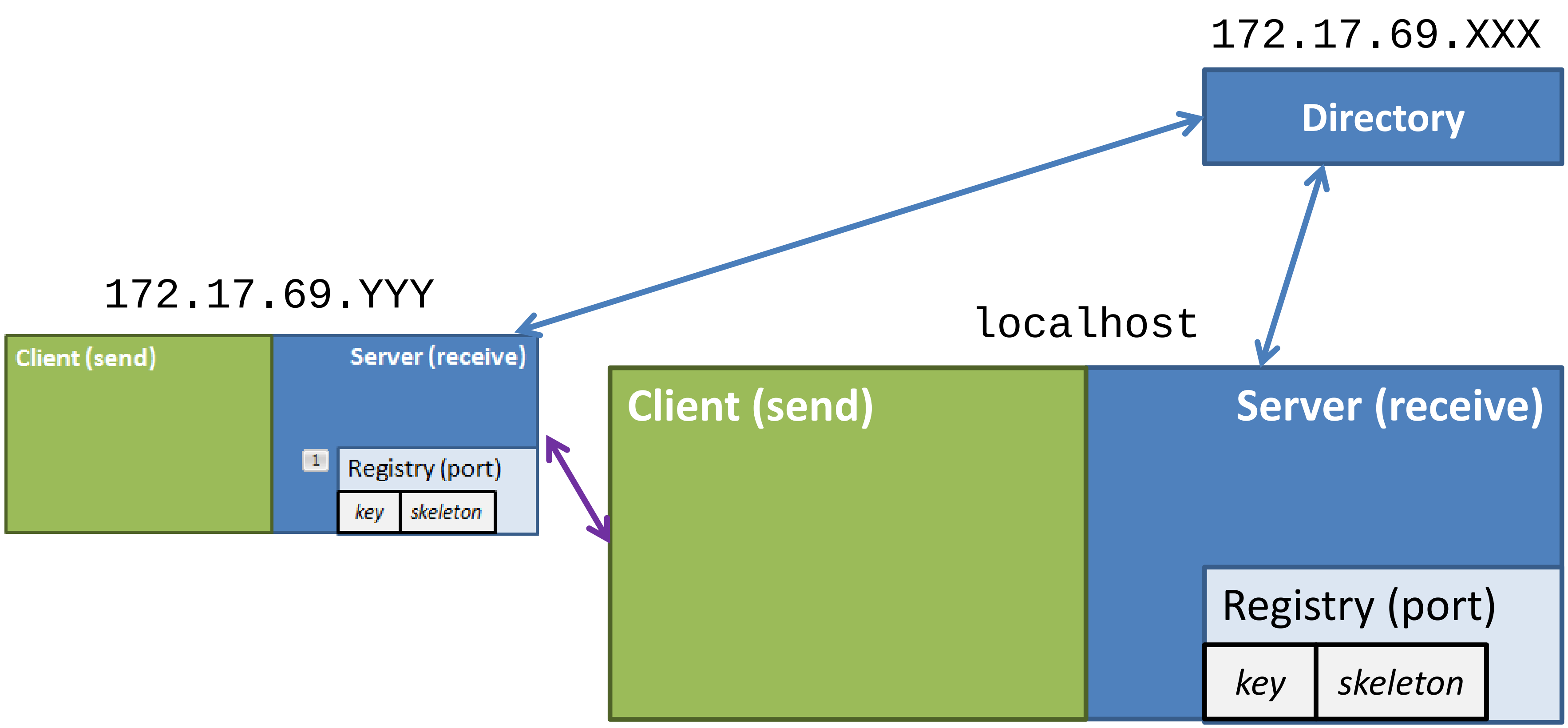
Eight Fallacies (to avoid)

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn’t change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

What about the system we built in the lab?

LET'S THINK ABOUT LAB 3

Using Java RMI to count trigrams ...



LIMITATIONS OF DISTRIBUTED COMPUTING: CAP THEOREM

But first … ACID

Have you heard of ACID guarantees in a database class?

For traditional (non-distributed) databases …

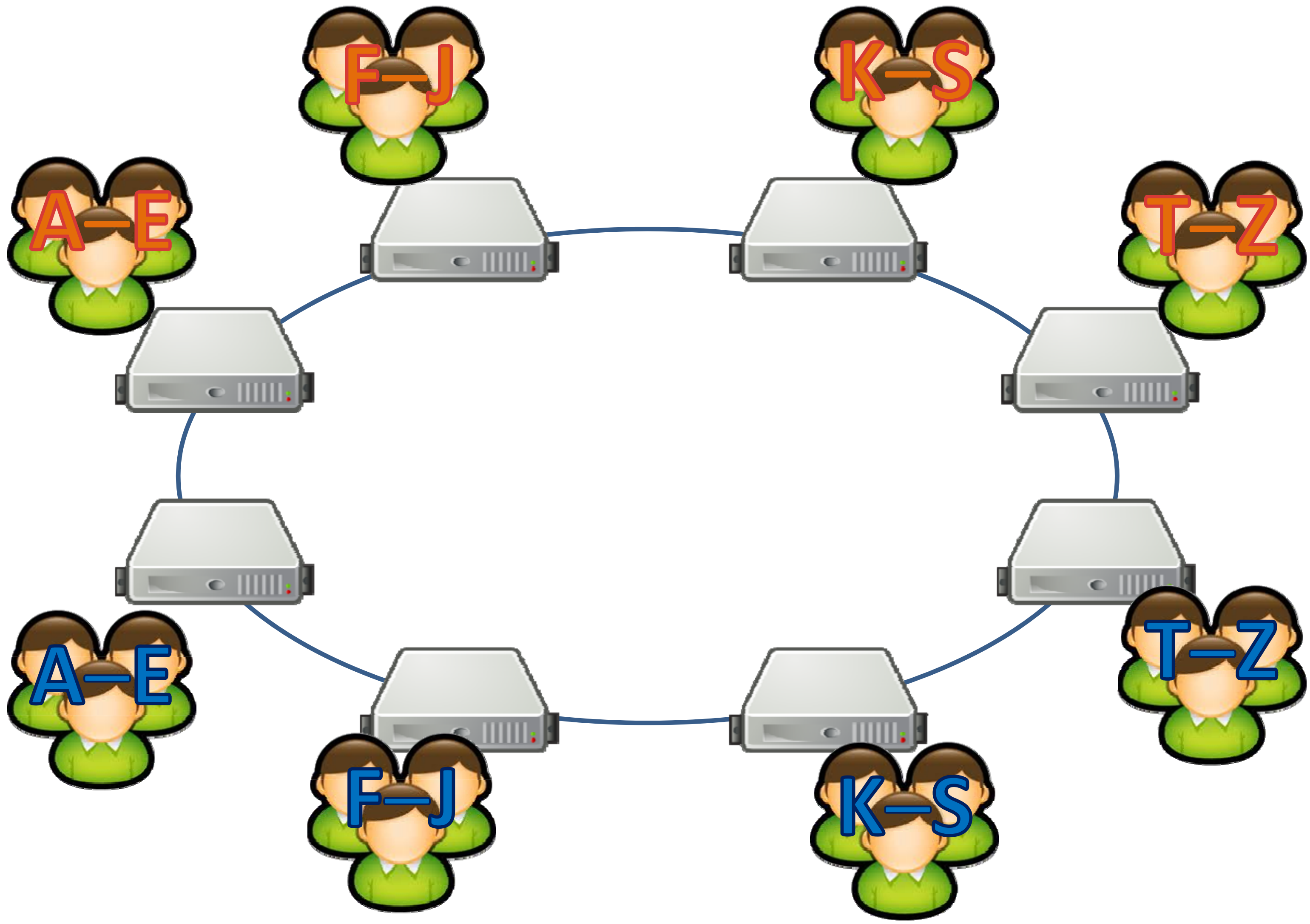
1. **A**tomicity:
   – Transactions all or nothing: fail cleanly
2. **C**onsistency:
   – Doesn't break constraints/rules
3. **I**solation:
   – Parallel transactions act as if sequential
4. **D**urability
   – System remembers changes

What is CAP?

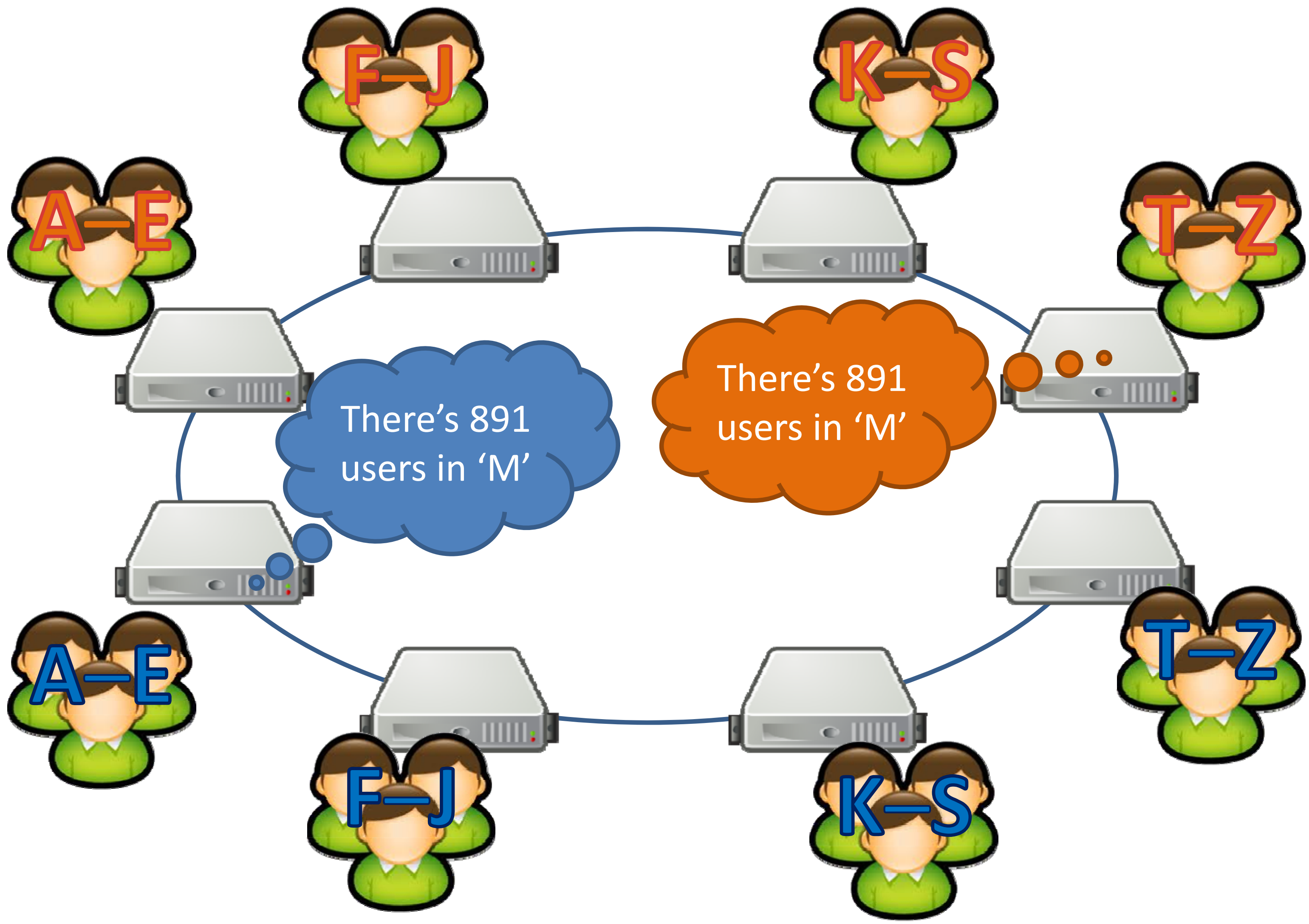
Three *guarantees* a distributed sys. could make

1. **C**onsistency:
   - All nodes have a consistent view of the system
2. **A**vailability:
   - Every read/write is acted upon
3. **P**artition-tolerance:
   - The system works even if messages are lost

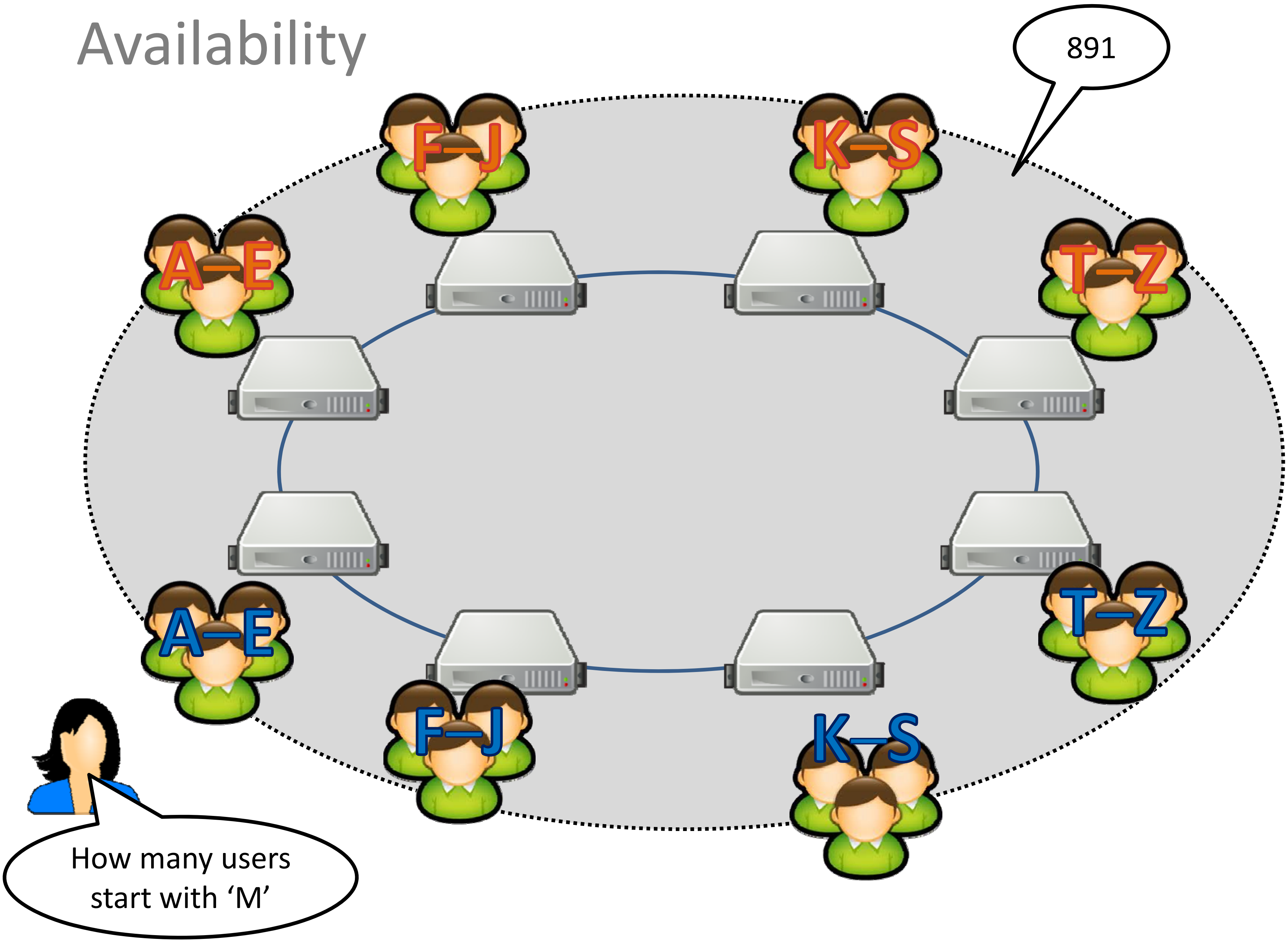
A Distributed System (Replication)



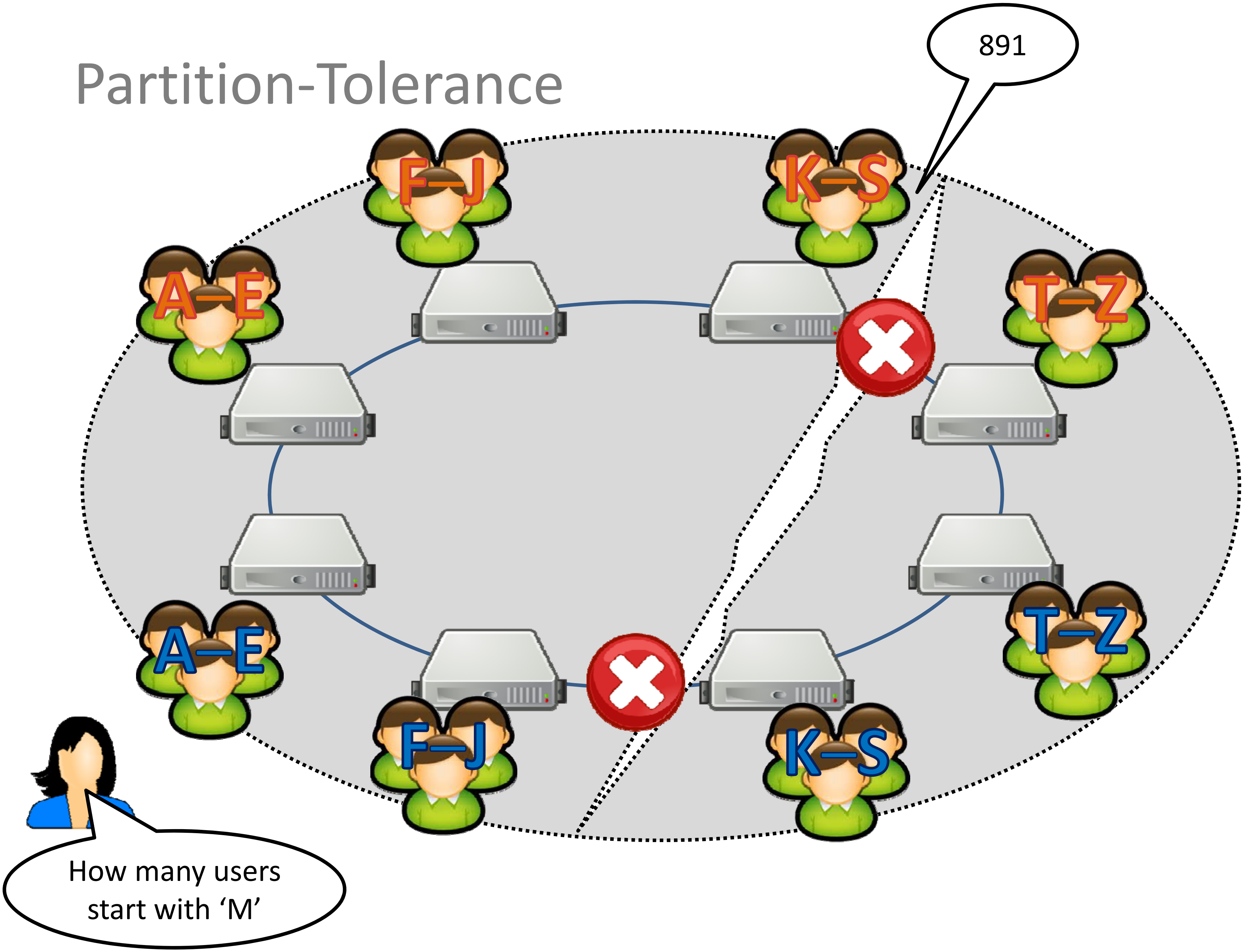
Consistency



Availability



Partition-Tolerance



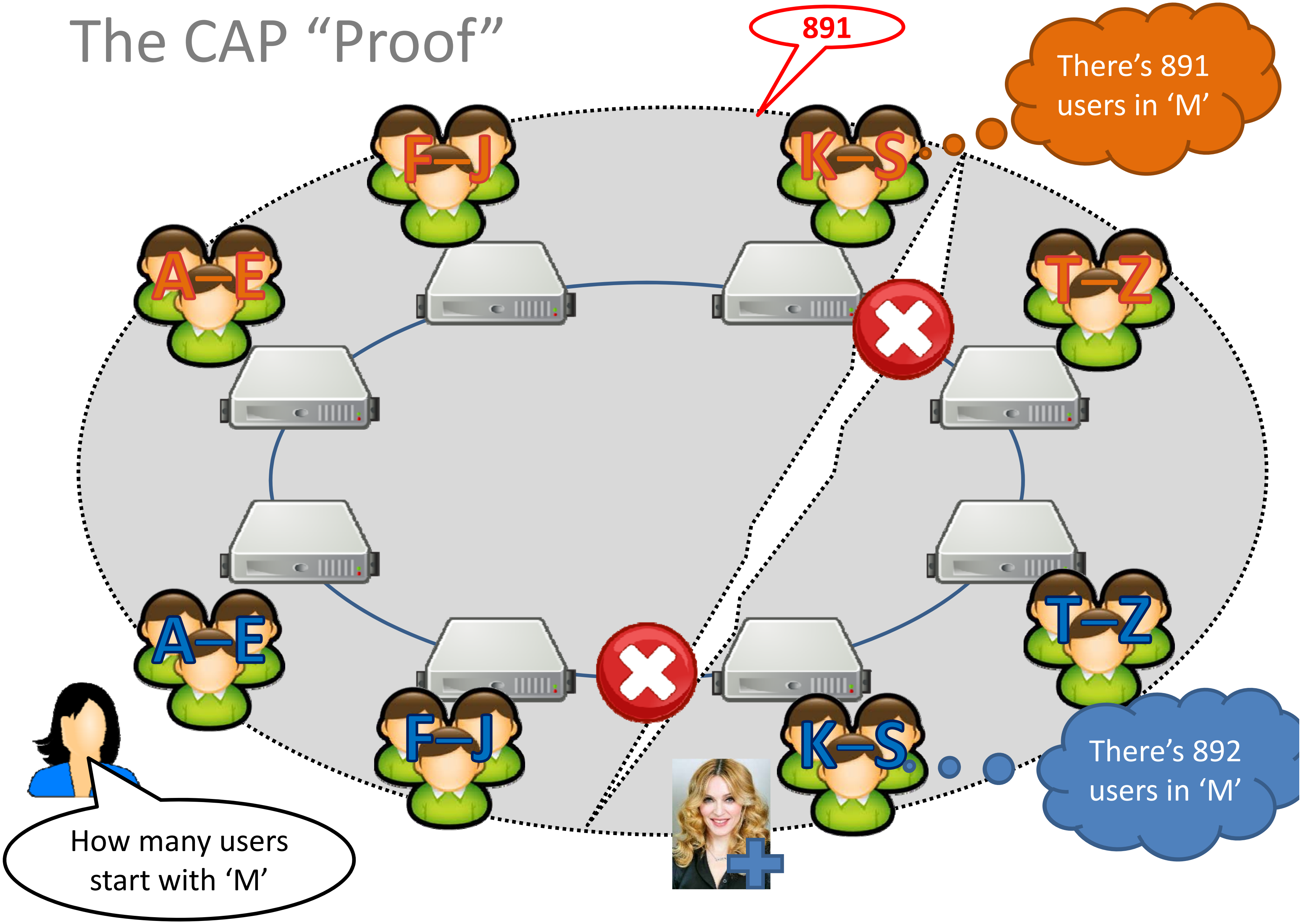
The CAP Question

Can a distributed system guarantee consistency (all nodes have the same up-to-date view), availability (every read/write is acted upon) and partition-tolerance (the system works even if messages are lost) at the same time?

What do you think?

The CAP Answer

The CAP "Proof"



The CAP "Proof" (in boring words)

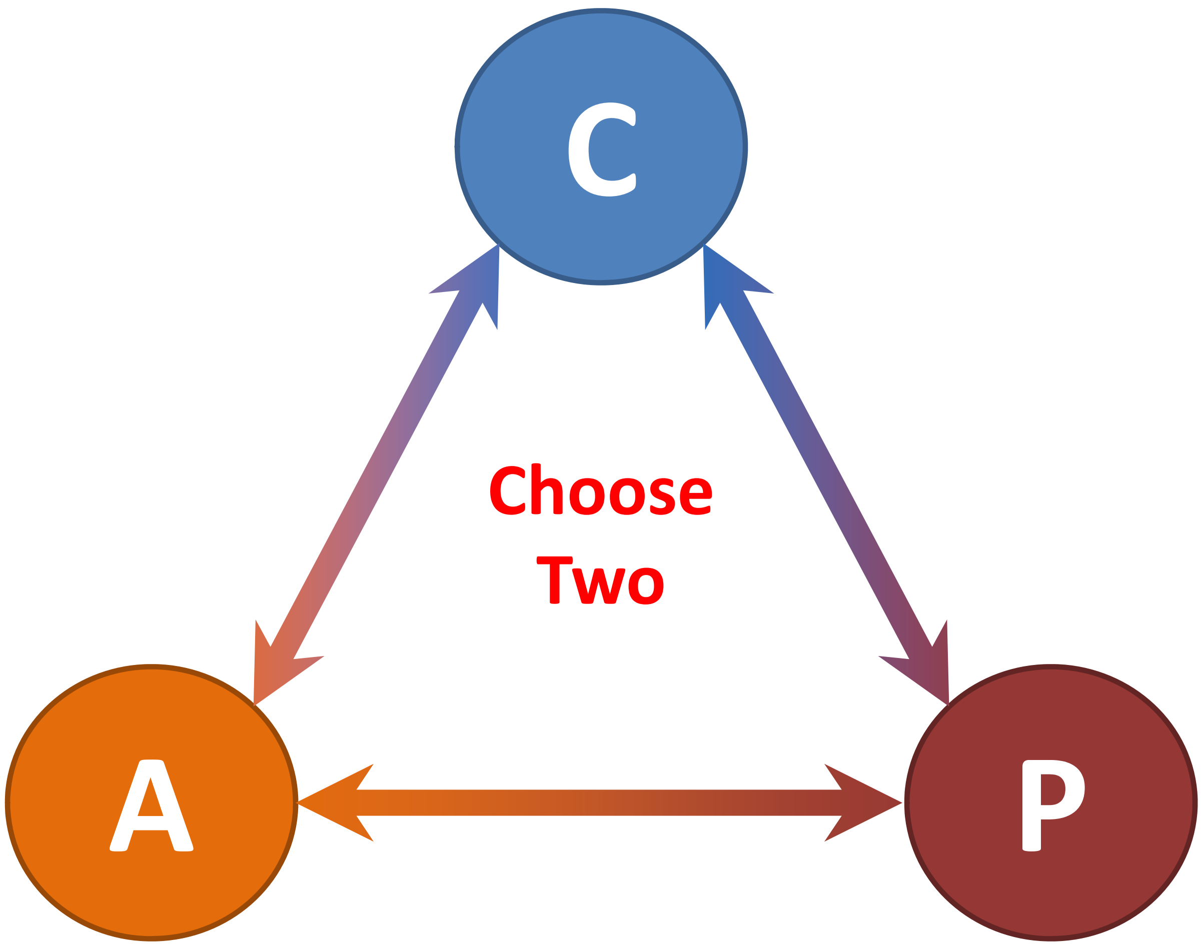
- Consider machines m_1 and m_2 on either side of a partition:
  - If an update is allowed on m_2 (**A**vailability), then m_1 cannot see the change: (loses **C**onsistency)
  - To make sure that m_1 and m_2 have the same, up-to-date view (**C**onsistency), neither m_1 nor m_2 can accept any requests/updates (lose **A**vailability)
  - Thus, only when m_1 and m_2 can communicate (lose **P**artition tolerance) can **A**vailability and **C**onsistency be guaranteed

The CAP Theorem

A distributed system cannot guarantee consistency (all nodes have the same up-to-date view), availability (every read/write is acted upon) and partition-tolerance (the system works even if messages are lost) at the same time.

(“Proof” as shown on previous slide ☺)

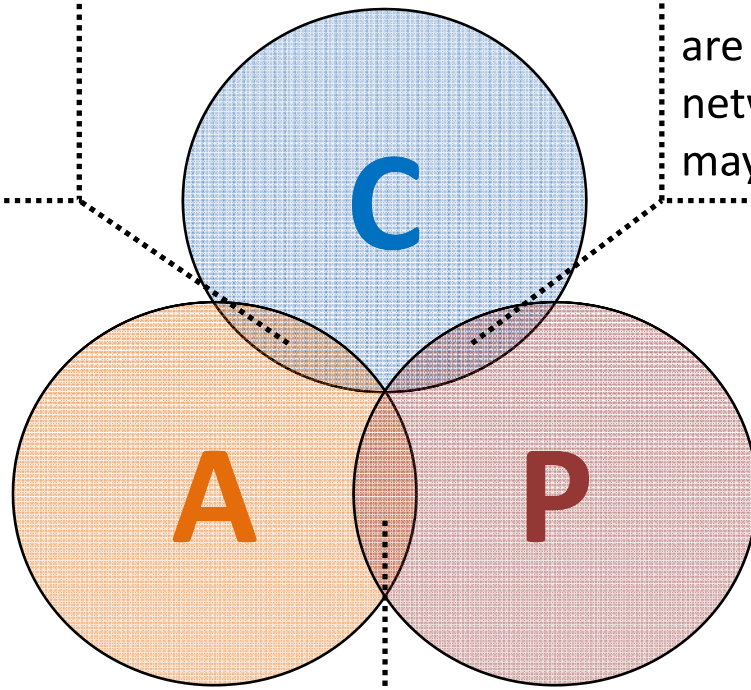
The CAP Triangle



CAP Systems

CA: Guarantees to give a correct response but only while network works fine (*Centralised / Traditional*)

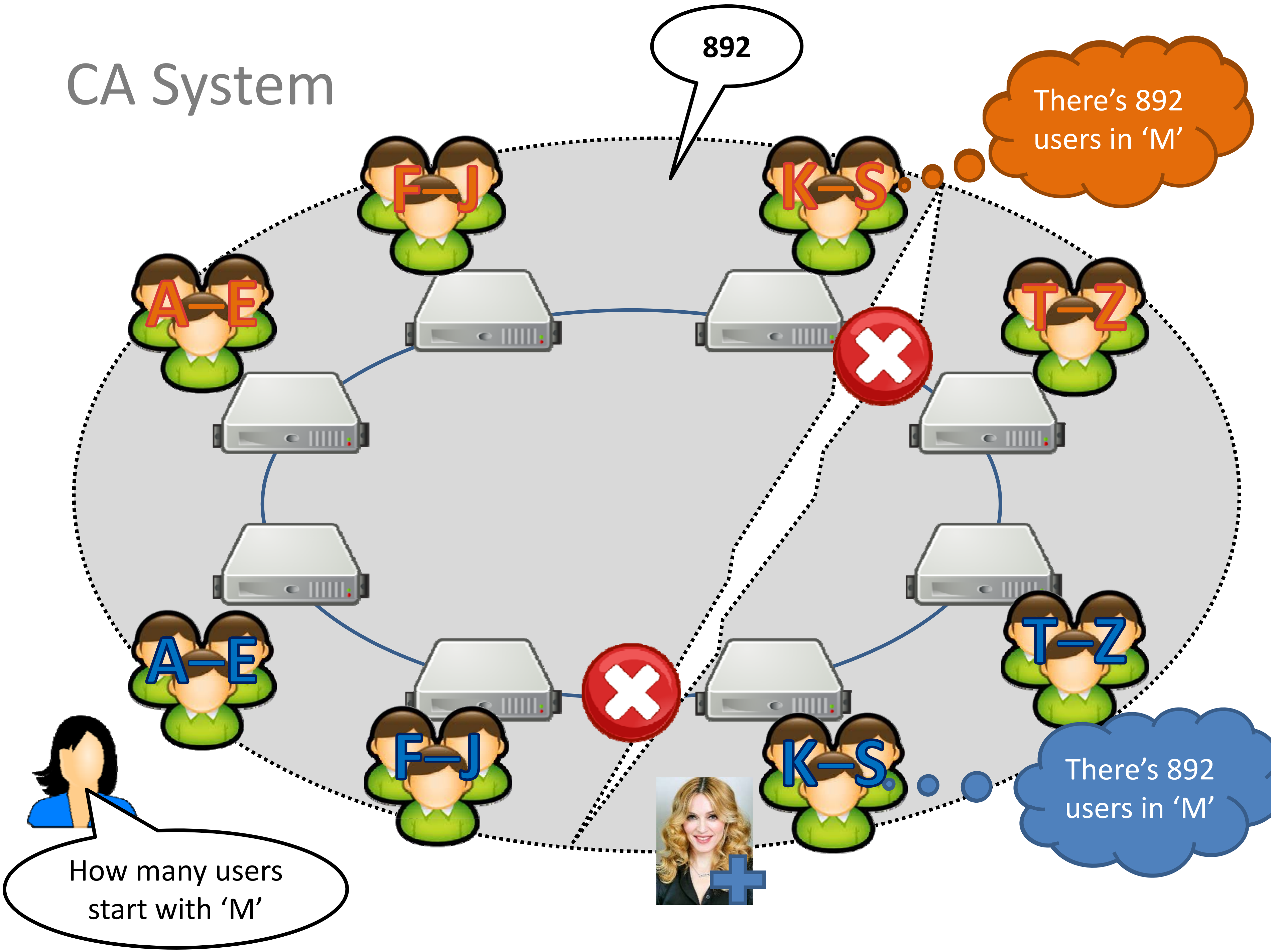
CP: Guarantees responses are correct even if there are network failures, but response may fail (*Weak availability*)



(No intersection)

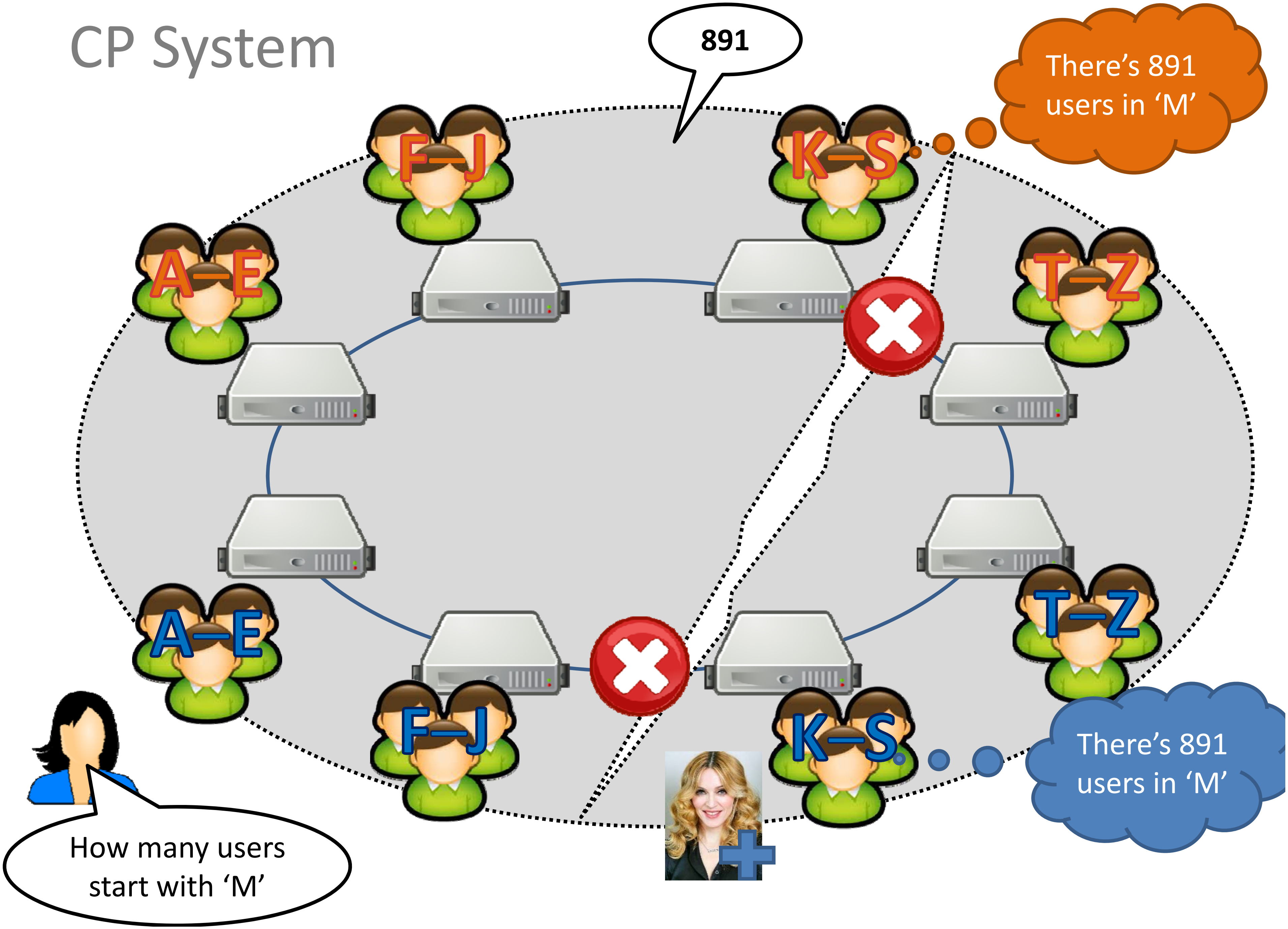
AP: Always provides a "best-effort" response even in presence of network failures (*Eventual consistency*)

CA System



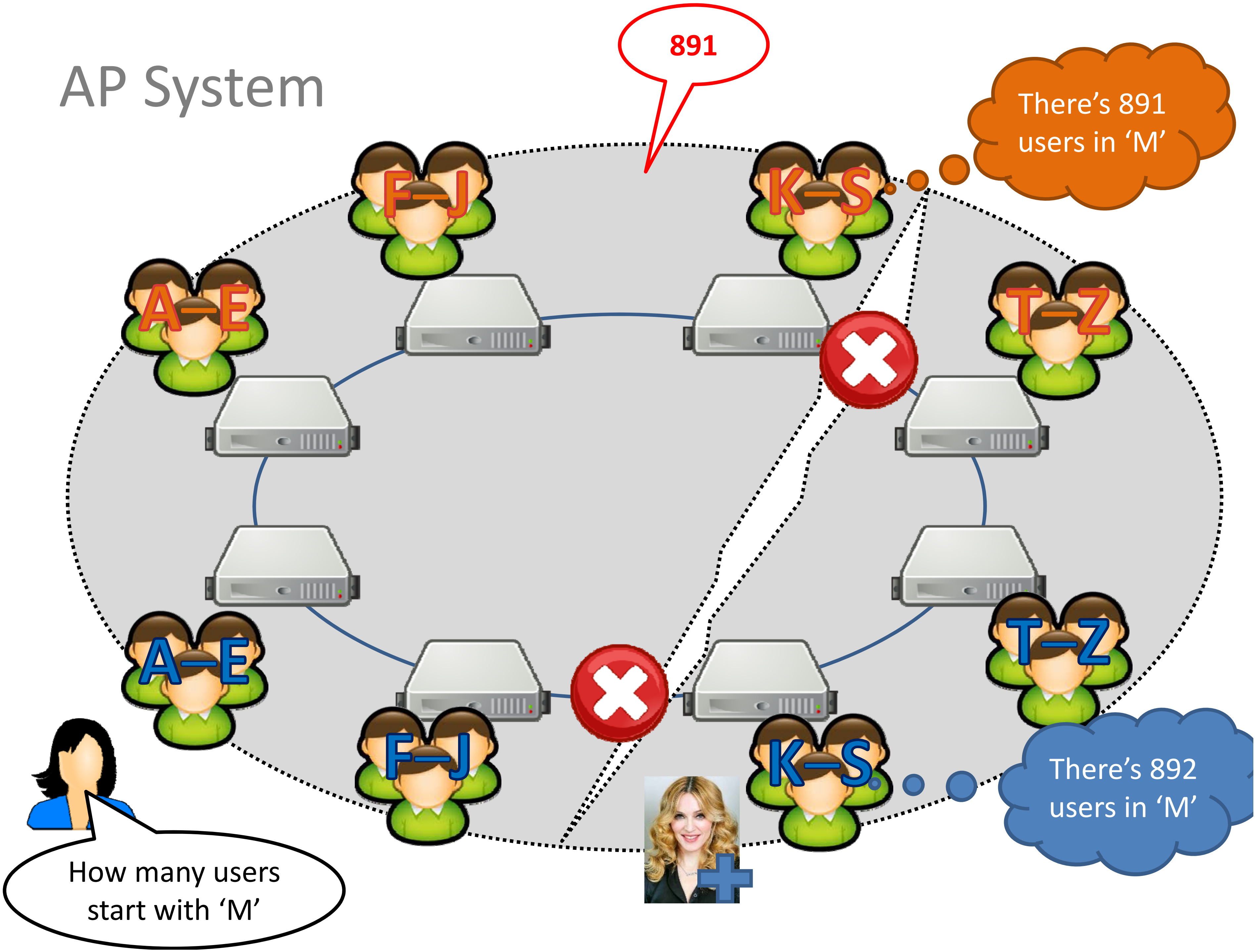
892
There's 892 users in 'M'
A–E
F–J
K–S
T–Z
A–E
F–J
K–S
T–Z
There's 892 users in 'M'
How many users start with 'M'

CP System



891
There's 891 users in 'M'
F–J
K–S
A–E
T–Z
A–E
F–J
T–Z
K–S
There's 891 users in 'M'
How many users start with 'M'

AP System


891
There's 891 users in 'M'
F–J
K–S
A–E
T–Z
A–E
F–J
K–S
T–Z
There's 892 users in 'M'
How many users start with 'M'

BASE (AP)

In what way was Twitter operating under BASE-like conditions?

- **B**asically **A**vailable
  - Pretty much always "up"
- **S**oft State
  - Replicated, cached data
- **E**ventual Consistency
  - Stale data tolerated, for a while

The CAP Theorem

- C,A in CAP ≠ C,A in ACID

- Simplified model
  - Partitions are rare
  - Systems may be a mix of CA/CP/AP
  - C/A/P often continuous in reality!

- But concept useful/frequently discussed:
  - How to handle Partitions?
    - Availability? or
    - Consistency?

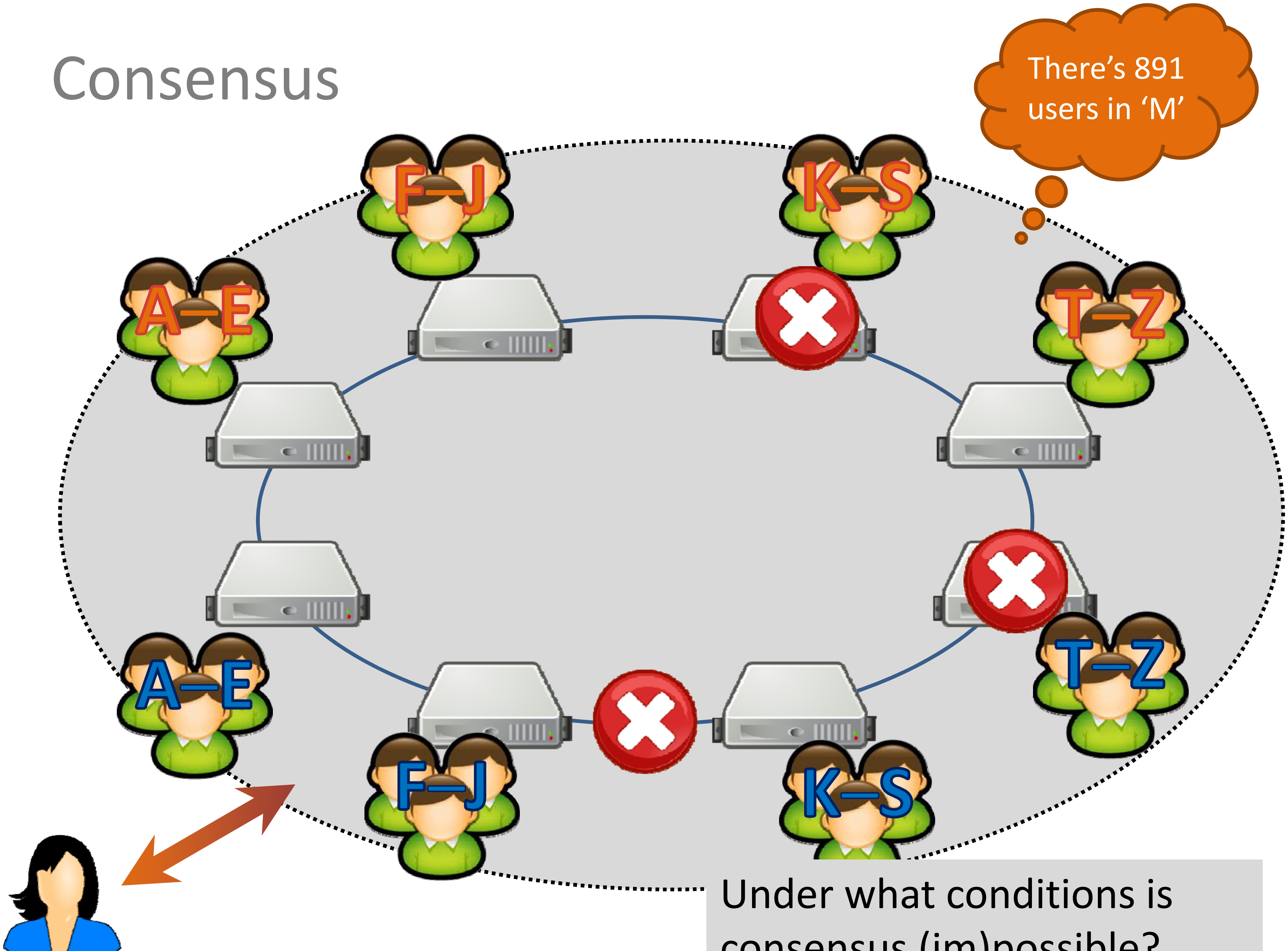
CONSENSUS

Consensus

- Goal: Build a reliable distributed system from unreliable components
  - "stable replica" semantics: distributed system as a whole acts as if it were a single functioning machine

- Core feature: the system, as a whole, is able to *agree* on values (consensus)
  - Value may be:
    - Client inputs
      - What to store, what to process, what to return
    - Order of execution
    - Internal organisation (e.g., who is leader)
    - …

Consensus



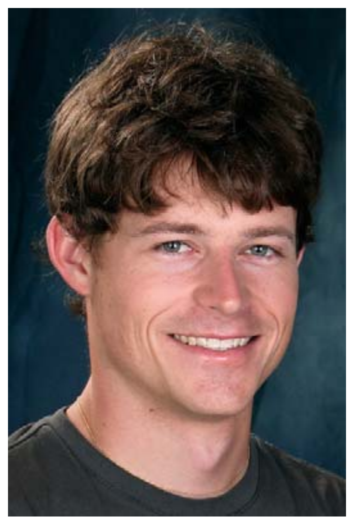
Under what conditions is consensus (im)possible?

Lunch Problem

Bob

10:30AM. Alice, Bob and Chris work in the same city. All three have agreed to go downtown for lunch today but have yet to decide on a place and a time.

Alice

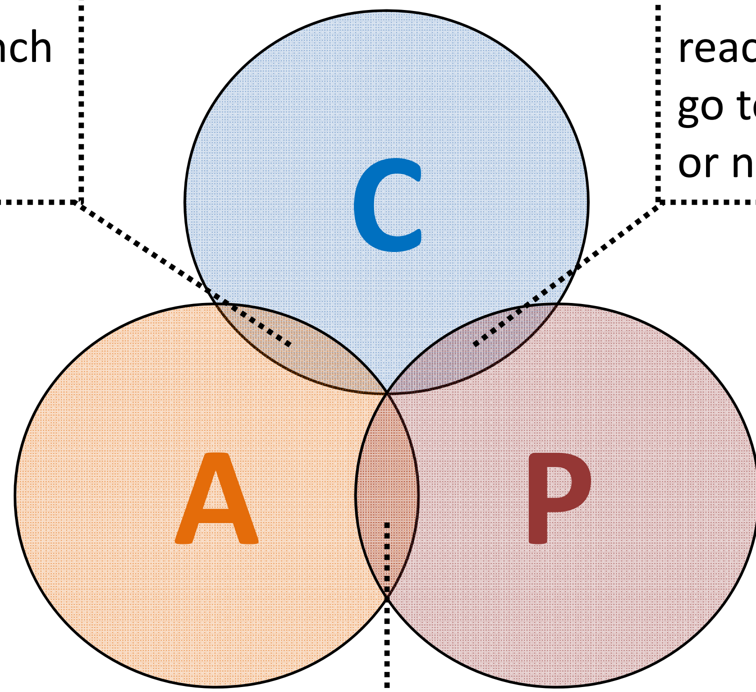


Chris

CAP Systems (for example …)

CA: They are guaranteed to go to the same place for lunch as long as each of them can be reached in time.

CP: If someone cannot be reached in time, they either all go to the same place for lunch or nobody goes for lunch.



(No intersection)

AP: If someone cannot be reached in time, they all go for lunch downtown but might not end up at the same place.

But how easily they can reach consensus depends on how they communicate!

SYNCHRONOUS VS. ASYNCHRONOUS

Synchronous vs. Asynchronous

- Synchronous distributed system:
  - Messages expected by a given time
    - E.g., a clock tick
  - Missing message has meaning
- Asynchronous distributed system:
  - Messages can arrive at any time
  - Missing message could arrive any time!

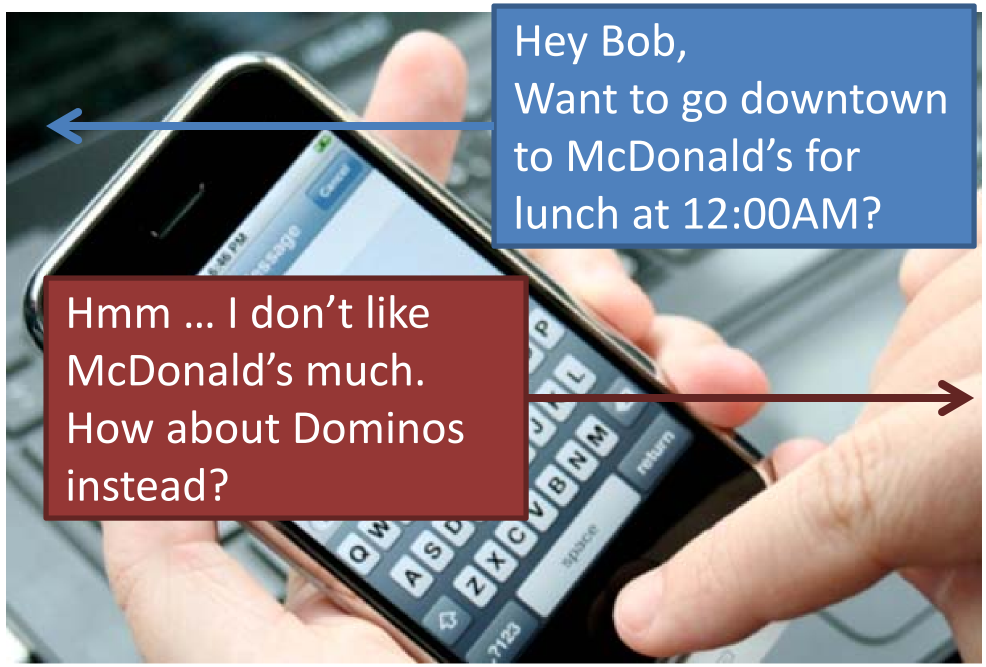
Asynchronous Consensus: Texting

10:45 AM. *Alice tries to invite Bob for lunch …*

11:35 AM. *No response. Should Alice head downtown?*

Asynchronous Consensus: Texting

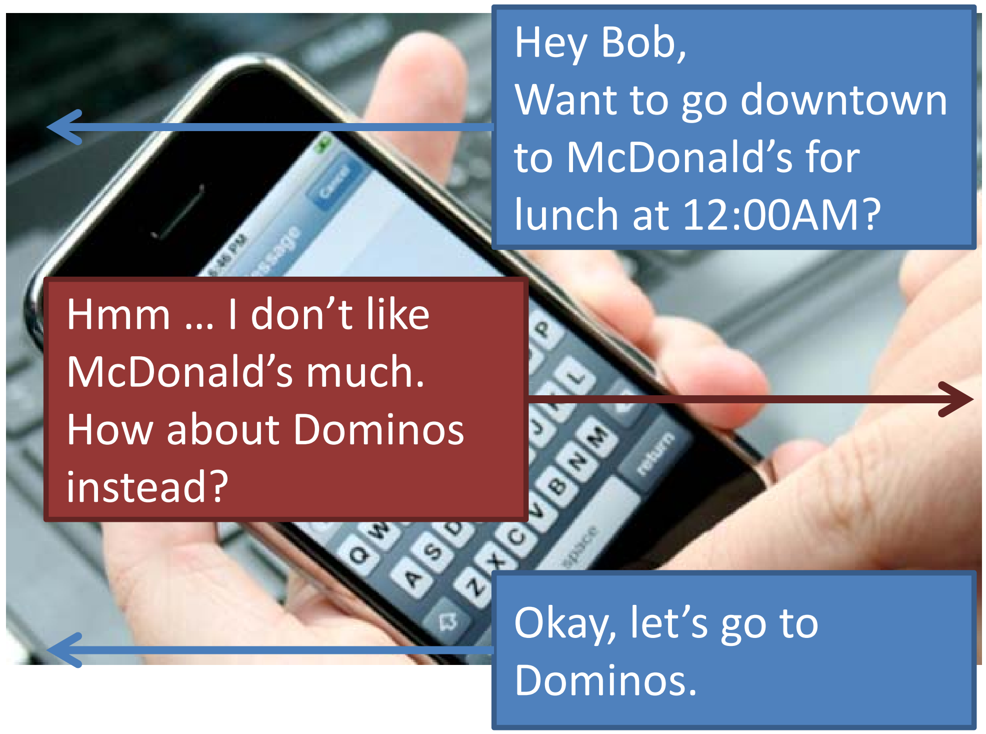
10:45 AM. *Alice tries to invite Bob for lunch …*



11:42 AM. *No response. Where should Bob go?*

Asynchronous Consensus: Texting

10:45 AM. *Alice tries to invite Bob for lunch …*



11:38 AM. *No response. Did Bob see the acknowledgement?*

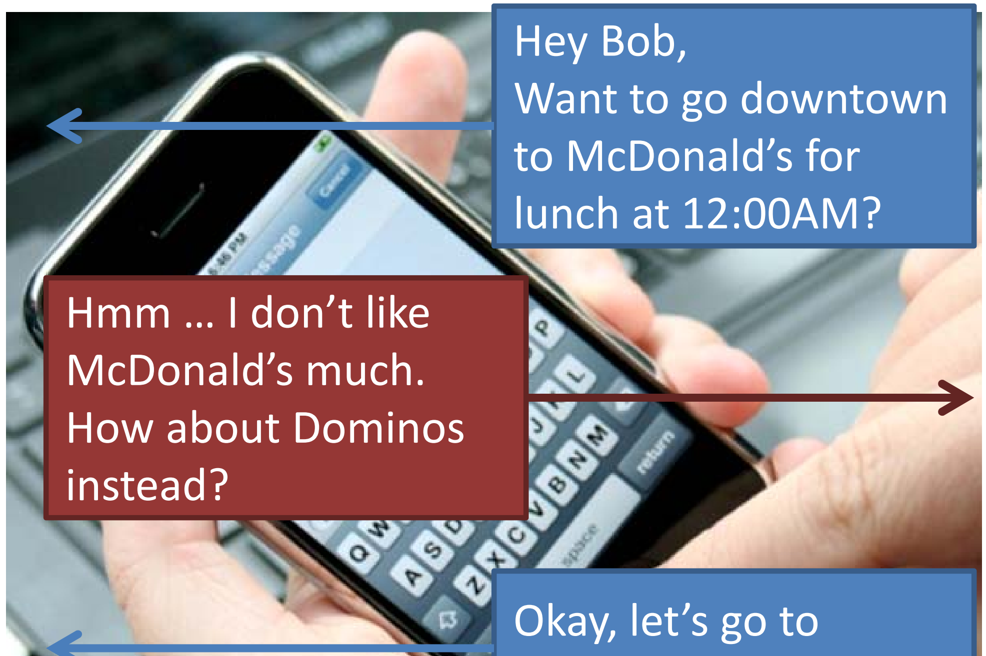
Asynchronous Consensus

- Impossible to guarantee!
  - A message delay can happen at any time and a node can wake up at the wrong time!
  - Fischer-Lynch-Patterson (1985): No consensus can be <u>guaranteed</u> amongst working nodes if there is even a single failure

- But asynchronous consensus can happen
  - As you should realise if you've ever successfully organised a meeting by email or text ;)

Asynchronous Consensus: Texting

10:45 AM. *Alice tries to invite Bob for lunch …*



Hey Bob,
Want to go downtown to McDonald's for lunch at 12:00AM?

Hmm … I don't like McDonald's much. How about Dominos instead?

Okay, let's go to Dominos.

11:38 AM. *No response. Bob's battery died. Alice misses the train downtown waiting for message, heads to the cafeteria at work instead. Bob charges his phone …*

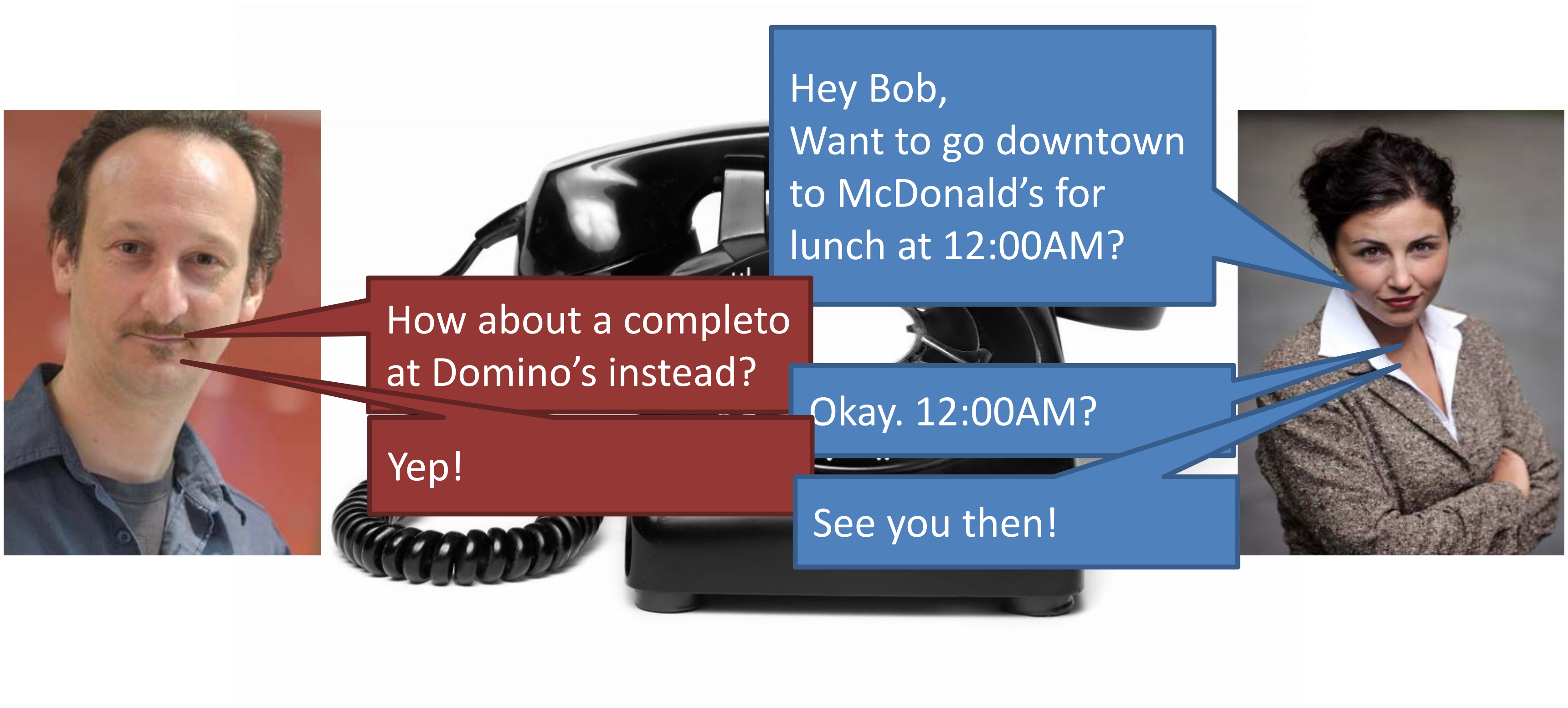
Heading to Dominos now. See you there!

Asynchronous Consensus: Texting

How could Alice and Bob find consensus on a time and place to meet for lunch?

Synchronous Consensus: Telephone

10:45 AM. *Alice tries to invite Bob for lunch …*



10:46 AM. **Clear consensus!**

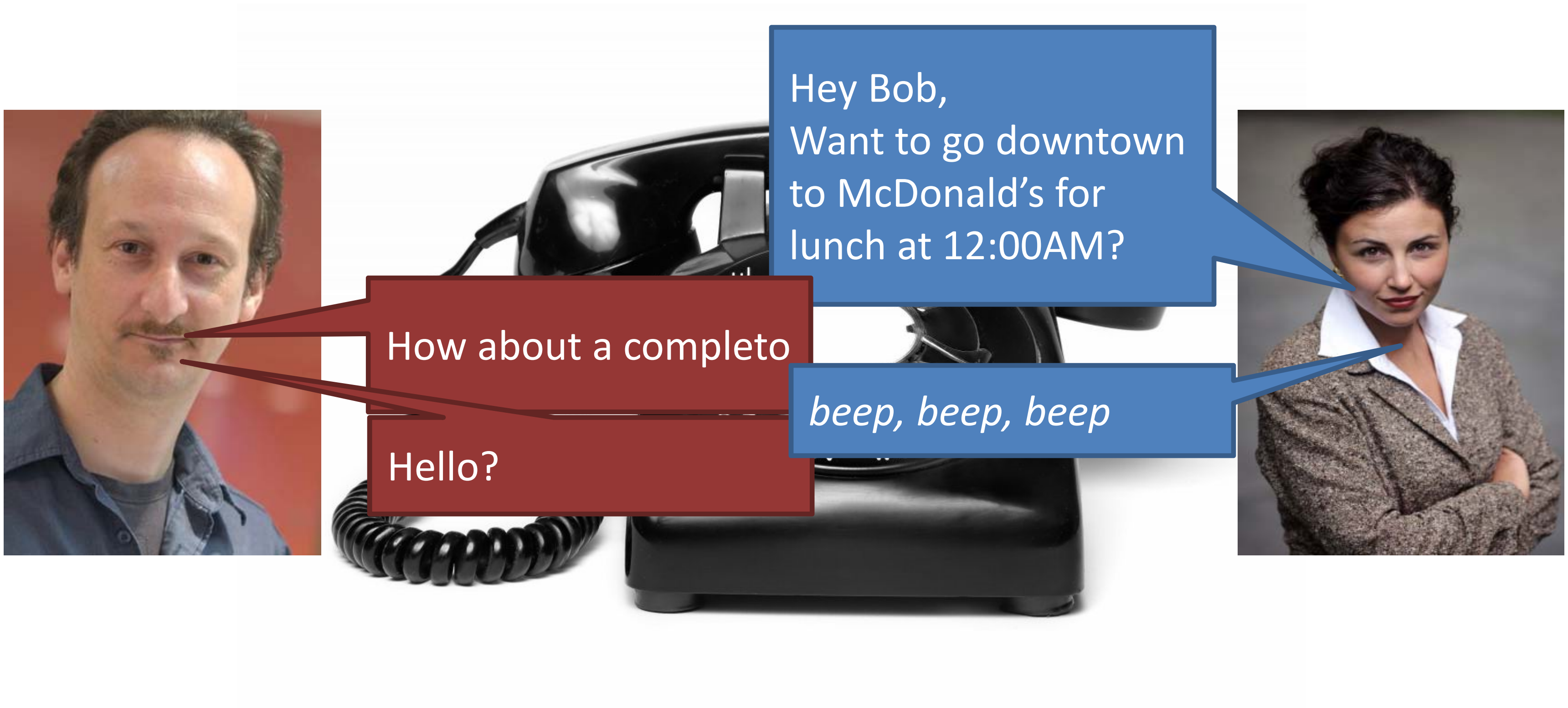
Synchronous Consensus

- Can be guaranteed!
  - But only under certain conditions …

What is the core difference between reaching consensus in synchronous (texting or email) vs. asynchronous (phone call) scenarios?

Synchronous Consensus: Telephone

10:45 AM. *Alice tries to invite Bob for lunch …*



10:46 AM. **What's the protocol?**

From asynchronous to synchronous

How could we (in some cases) turn an asynchronous system into a synchronous system?

- Agree on a timeout Δ
  - Any message not received within Δ = failure
  - If a message arrives after Δ, system returns to being asynchonous
    - If Δ is unbounded, the system is asynchronous
    - May need a large value for Δ in practice

Eventually synchronous

- Eventually synchronous: Assumes *most* messages will return within time Δ
  - More precisely, the number of messages that do not return in Δ is *bounded*
    - We don't need to set Δ so high
    - True in many practical systems

Why might consensus be easier in an eventually synchronous system?

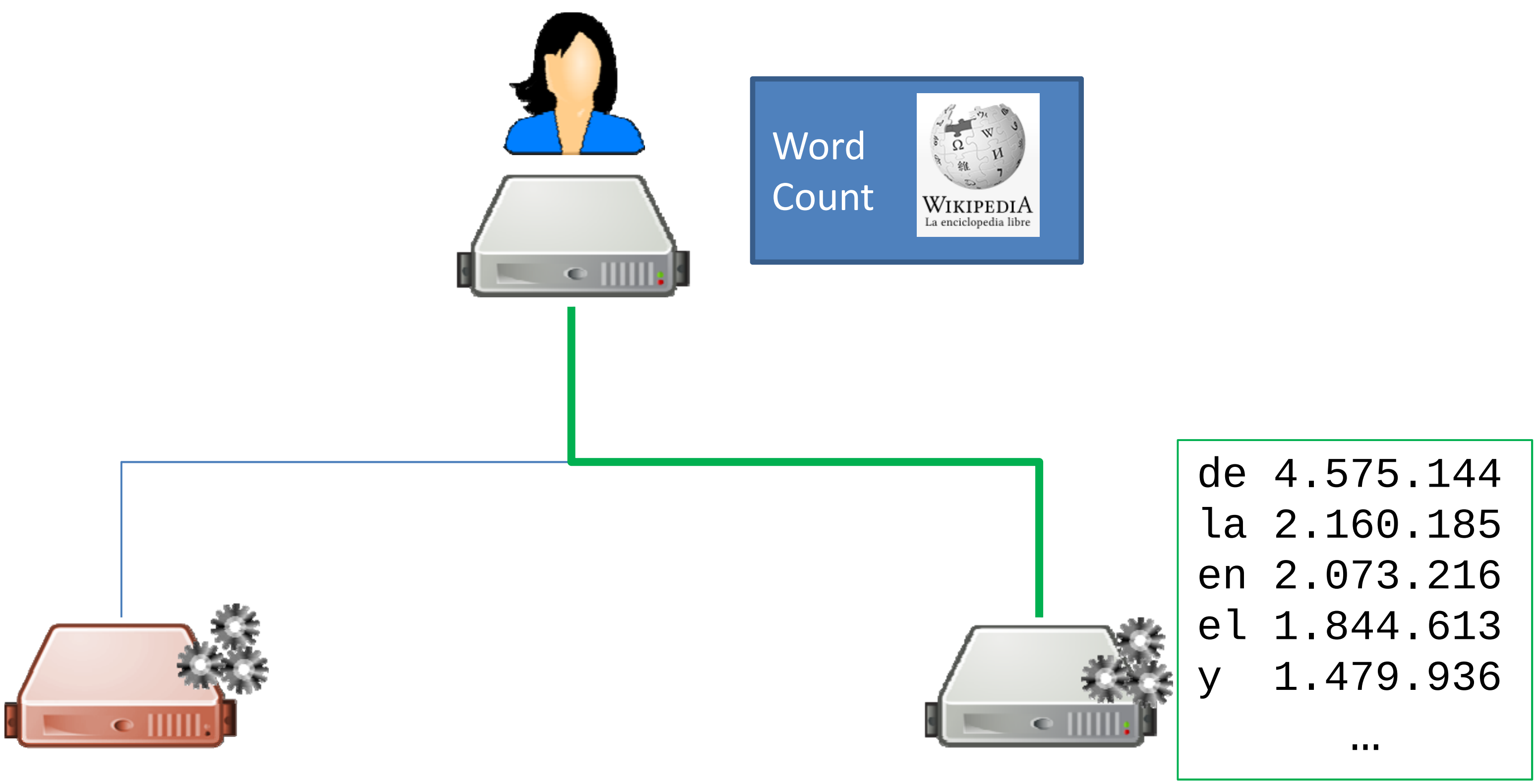
  - If a message does not return in time Δ, if we keep retrying, eventually it will return in time Δ

FAULT TOLERANCE:
FAIL–STOP VS. BYZANTINE

Faults

Fail–Stop Fault

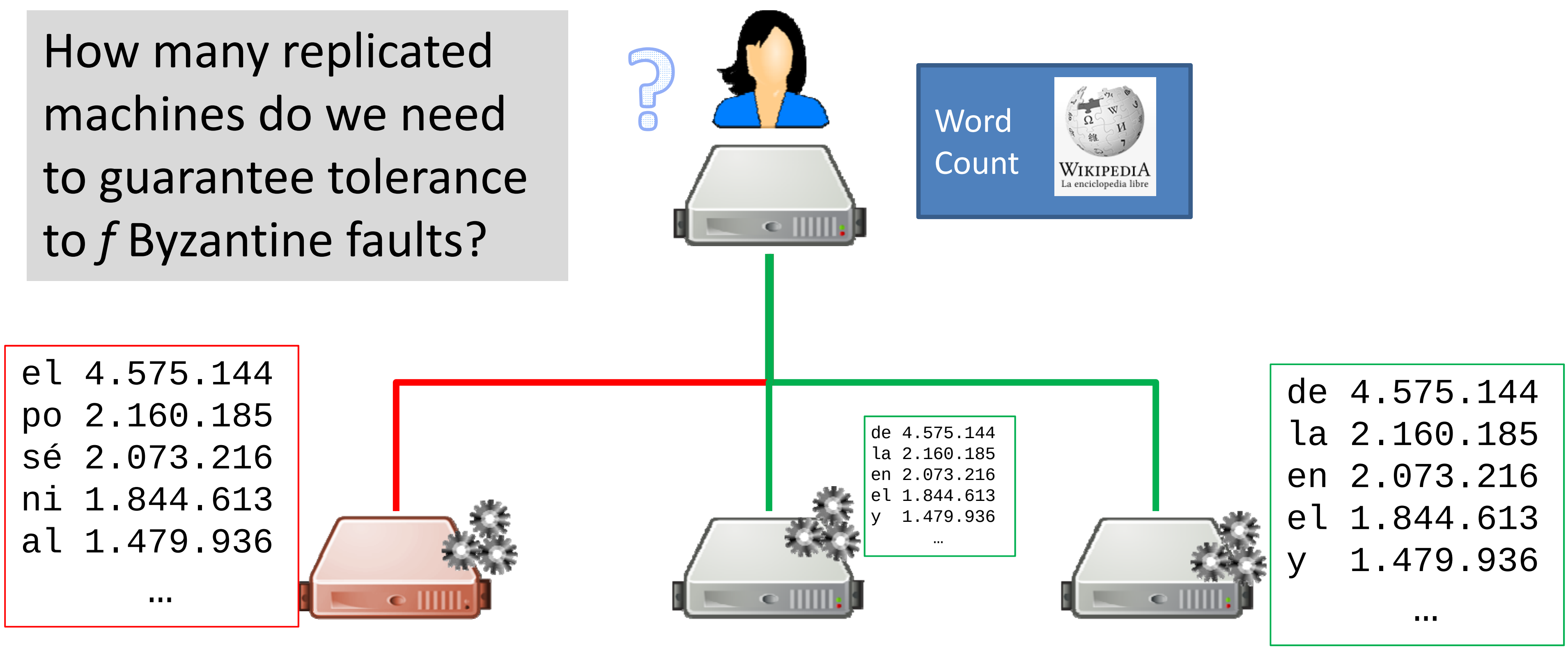
- A machine fails to respond or times-out (often hardware or load)
- Need *at least f*+1 replicated machines? (beware asynch.!)
  - *f* = number of clean failures



Byzantine Fault

- A machine responds incorrectly/maliciously (often software)
- Need *at least* **2*f***+1 replicated machines?
  - *f* = number of (possibly Byzantine) failures

How many replicated machines do we need to guarantee tolerance to *f* Byzantine faults?



Fail–Stop/Byzantine

- Naively:
  - Need *f*+1 replicated machines for fail–stop
  - Need 2*f*+1 replicated machines for Byzantine

- Not *so* simple if nodes must agree beforehand!

- **Replicas must have consensus to be useful!**

CONSENSUS GUARANTEES

Consensus Guarantees

- Under certain assumptions; for example
  - synchronous, eventually synchoronous, asynchronous
  - fail-stop, byzantine
  - no failures, one node fails, less than half fail

… there are methods to provide consensus with certain guarantees

A Consensus Protocol

- Agreement/Consistency [Safety]: All working nodes agree on the same value. Anything agreed is final!
- Validity/Integrity [Safety]: Every working node decides at most one value. That value has been proposed by a working node.
- Termination [Liveness]: All working nodes eventually decide (after finite steps).

- Safety: Nothing bad ever happens
- Liveness: Something good eventually happens

A Consensus Protocol for Lunch

- Agreement/Consistency [Safety]: Everyone agrees on the same place downtown for lunch, or agrees not to go downtown.
- Validity/Integrity [Safety]: Agreement involves a place someone actually wants to go.
- Termination [Liveness]: A decision will eventually be reached (hopefully before lunch).

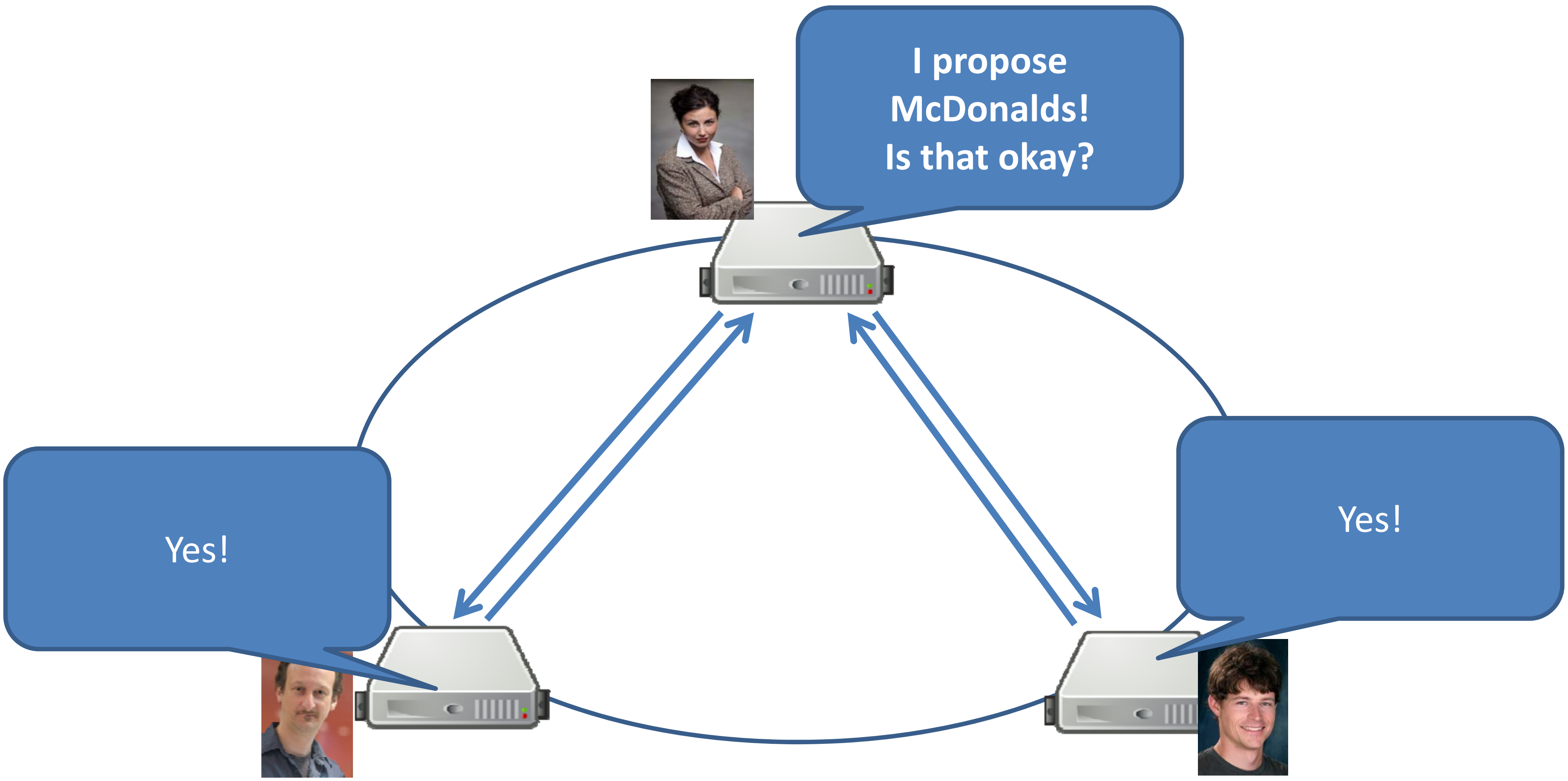
CONSENSUS PROTOCOL: TWO-PHASE COMMIT

Two-Phase Commit (2PC)

- Coordinator & cohort members
- **Goal:** Either all cohorts commit to the same value or no cohort commits to anything
- Assumes synchronous, fail-stop behaviour
  - Crashes are known!

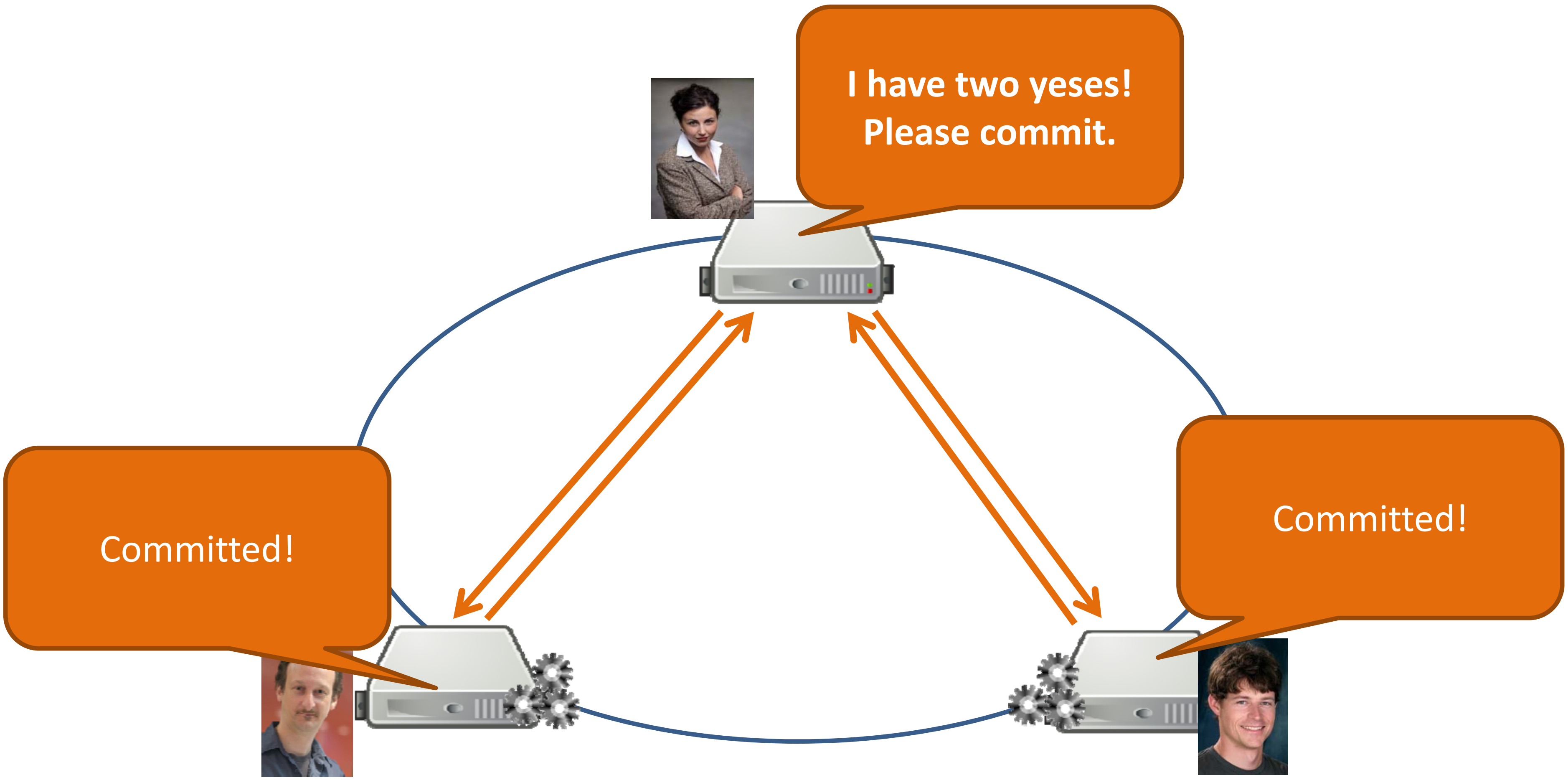
Two-Phase Commit (2PC)

1. Voting:



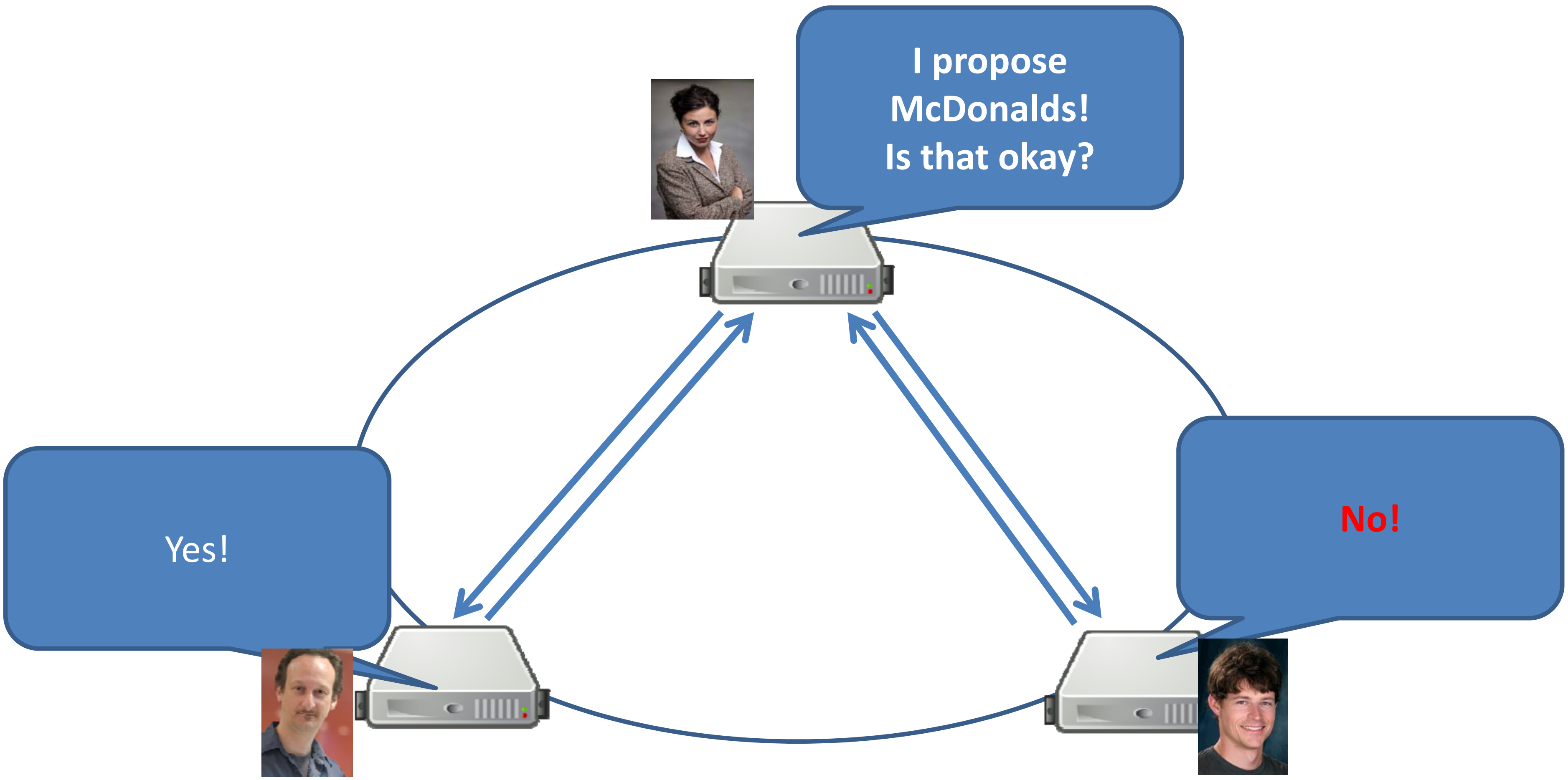
Two-Phase Commit (2PC)

2. Commit:



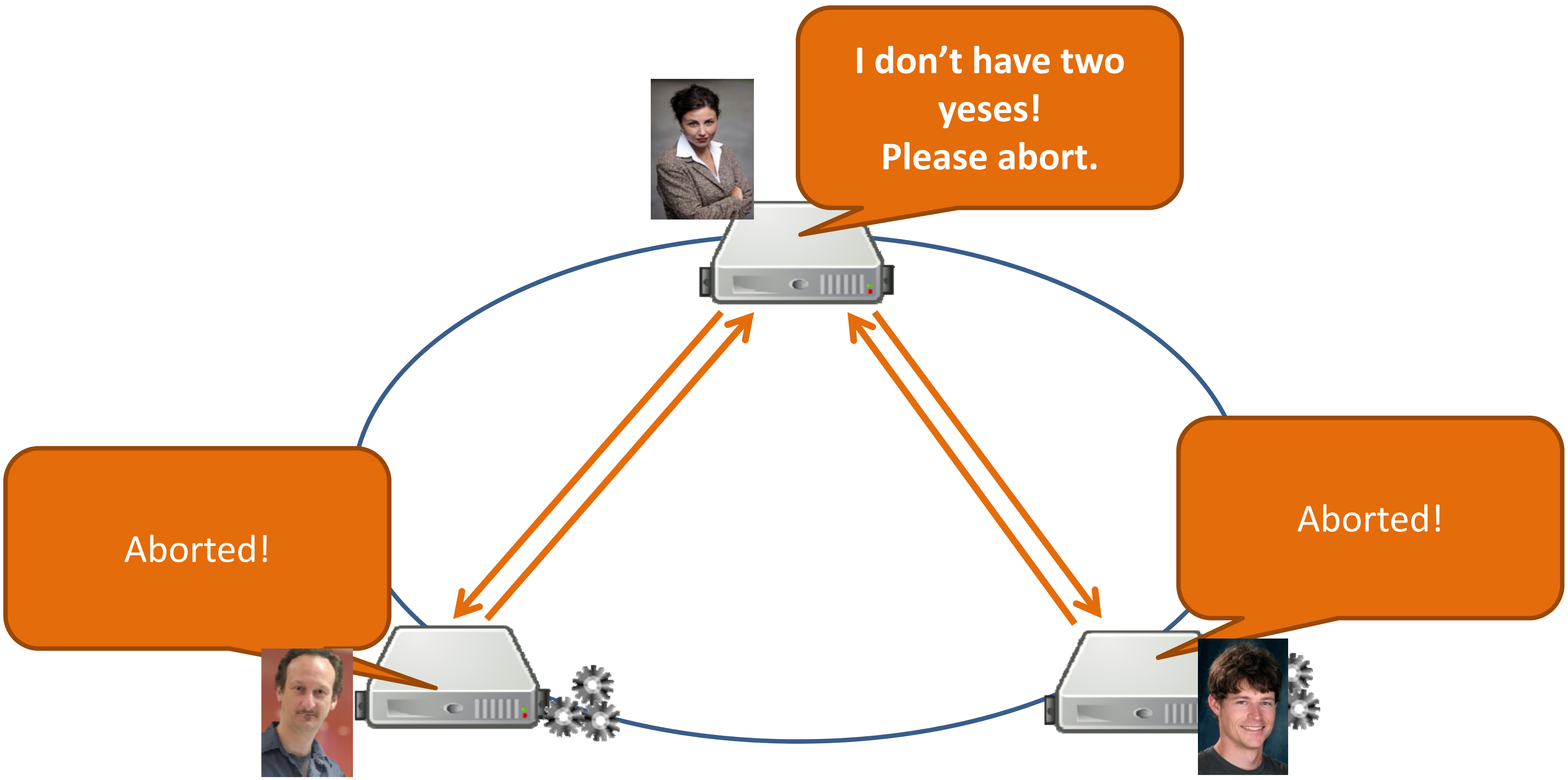
Two-Phase Commit (2PC) [Abort]

1. Voting:



Two-Phase Commit (2PC) [Abort]

2. Commit:



Two-Phase Commit (2PC)

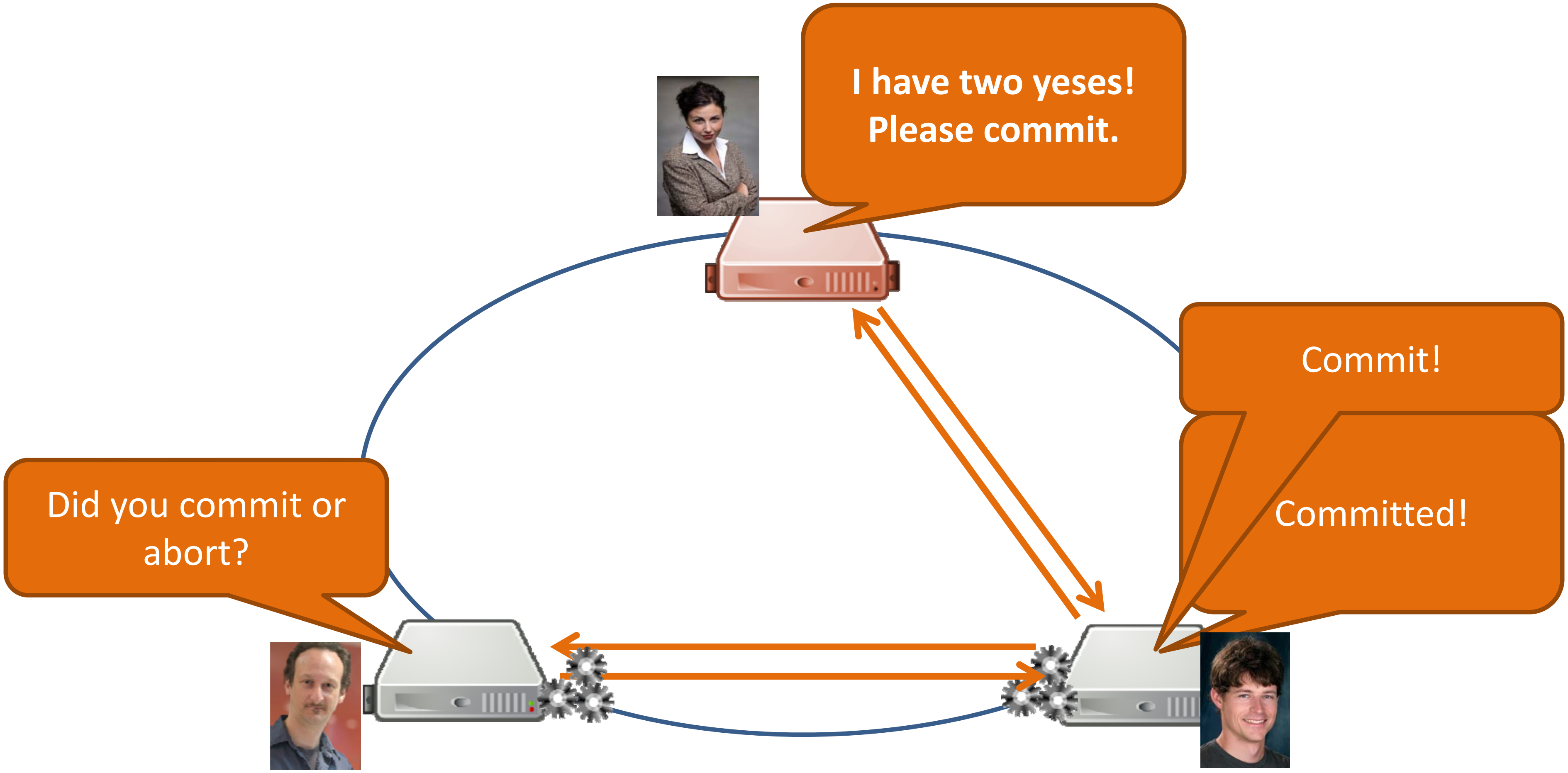
1. Voting: A coordinator proposes a commit value. The other nodes vote “yes” or “no” (they cannot propose a new value!).
2. Commit: The coordinator counts the votes. If all are “yes”, the coordinator tells the nodes to accept (commit) the answer. If one is “no”, the coordinator aborts the commit.

- For *n* nodes, in the order of 4*n* messages.
  - 2*n* messages to propose value and receive votes
  - 2*n* messages to request commit and receive acks

Two-Phase Commit (2PC)

What happens if the coordinator fails?

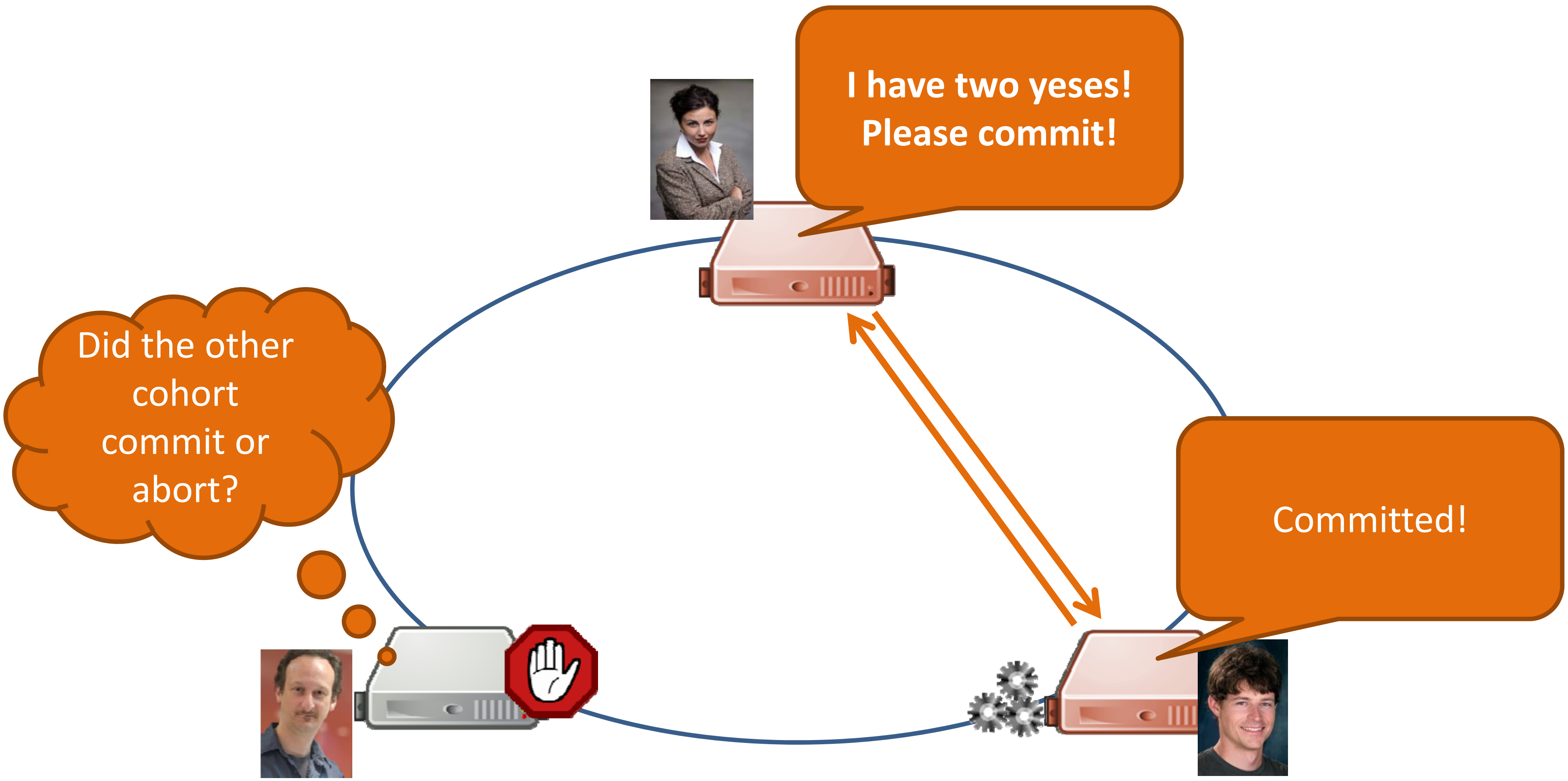
- Cohort members know coordinator has failed!



Two-Phase Commit (2PC)

What happens if a coordinator and a cohort fail?

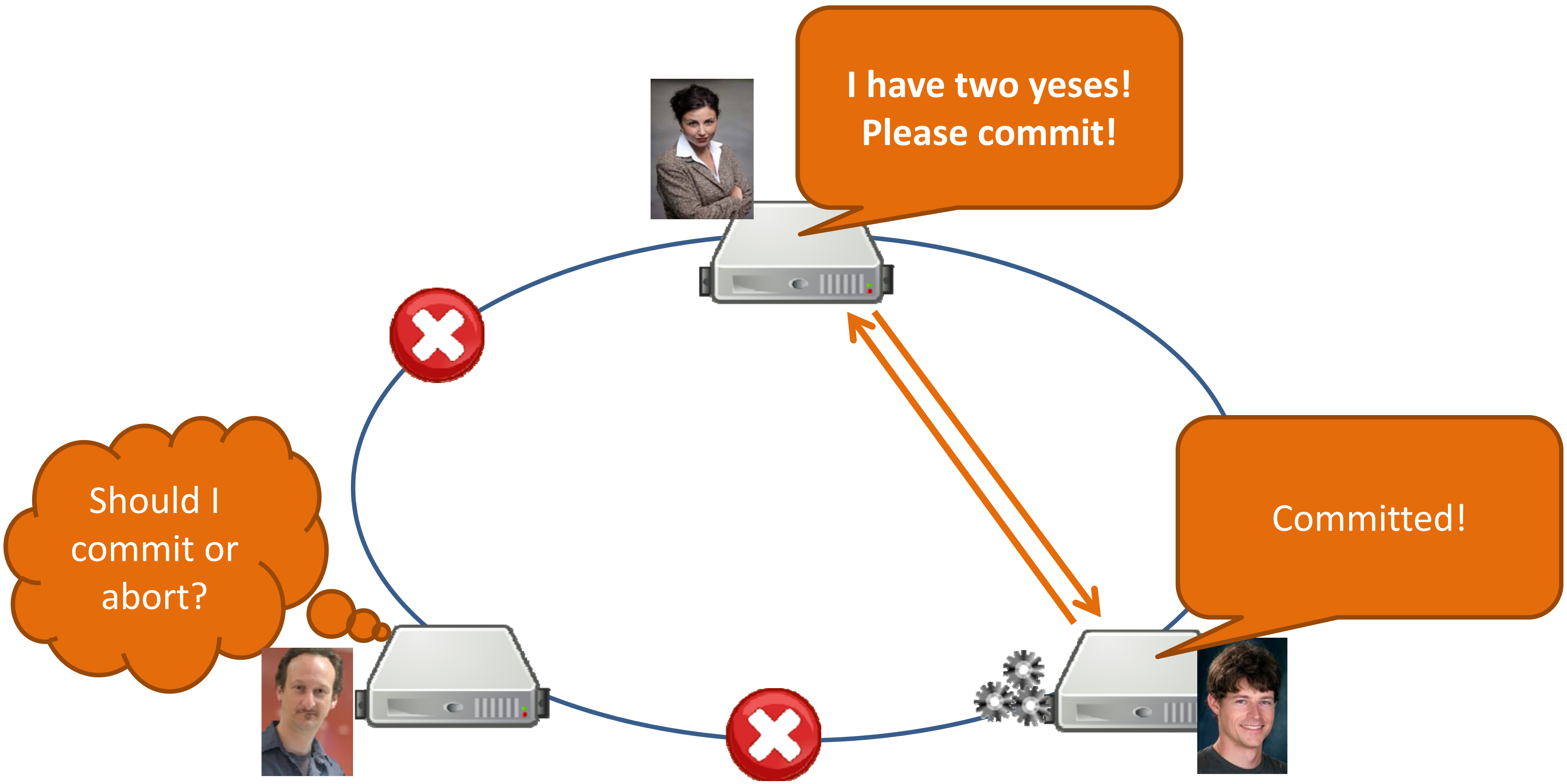
Not fault-tolerant!



Two-Phase Commit (2PC)

What happens if there's a partition?

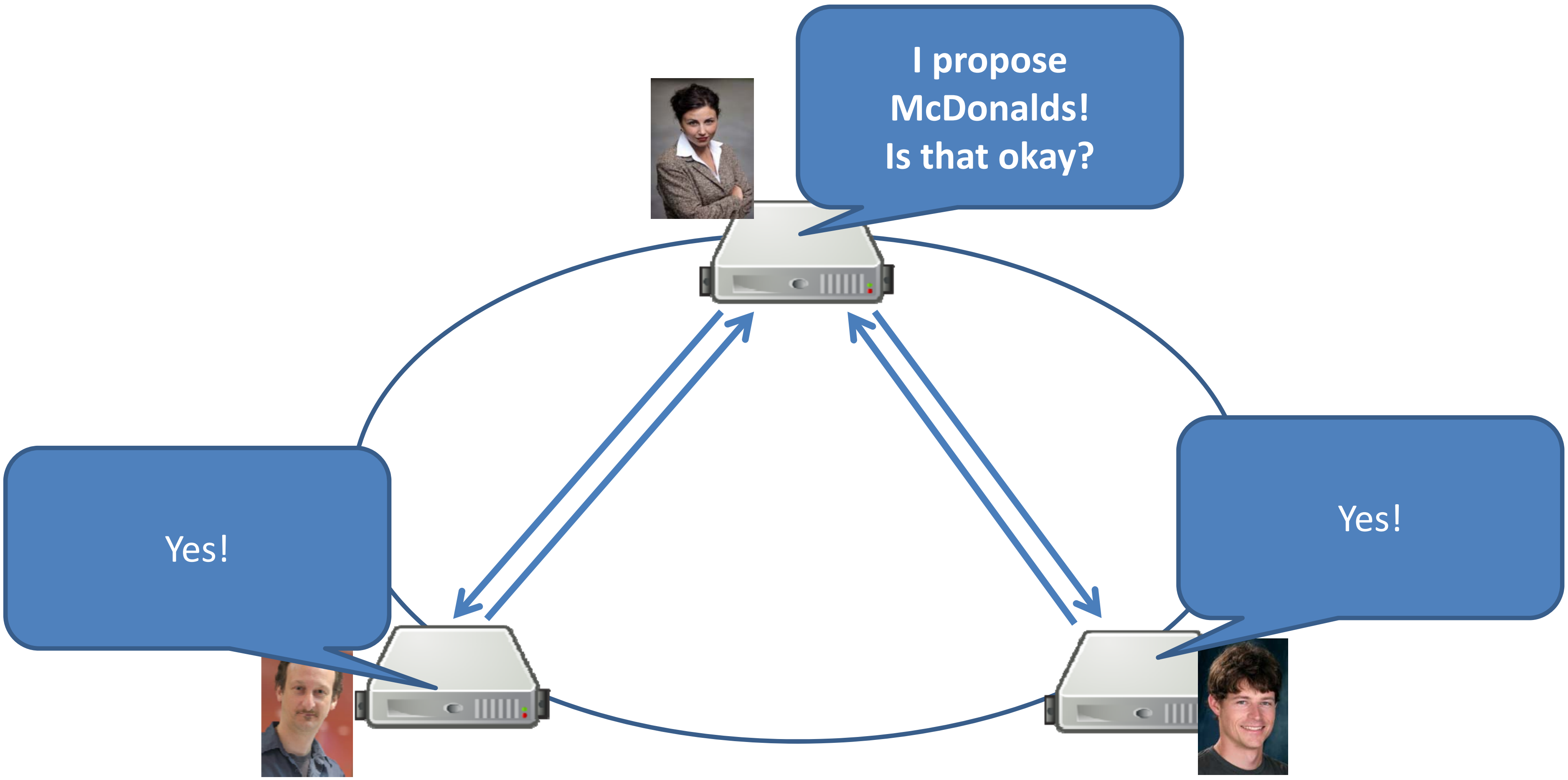
Not fault-tolerant!



CONSENSUS PROTOCOL: THREE-PHASE COMMIT

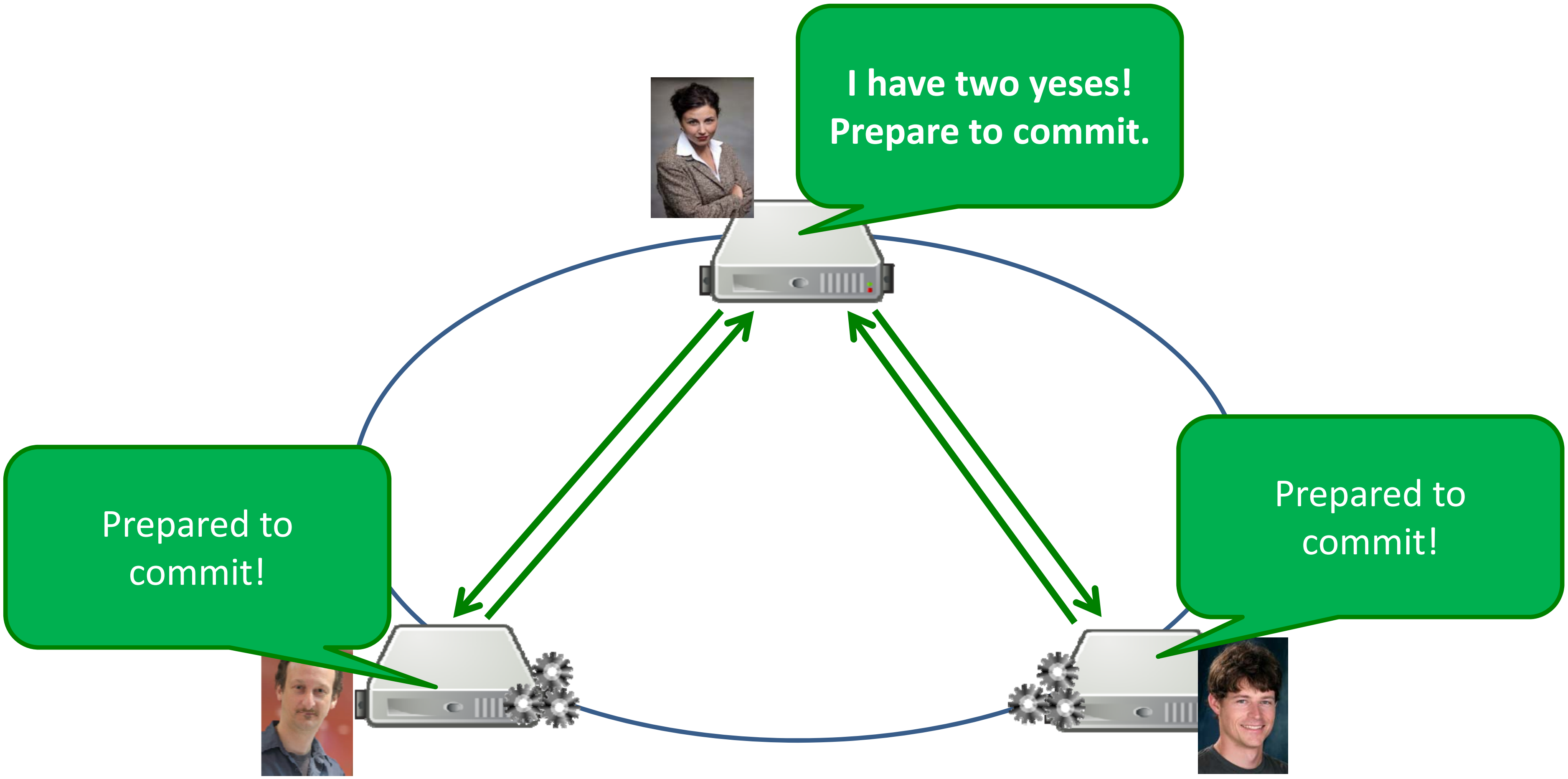
Three-Phase Commit (3PC)

1. Voting:



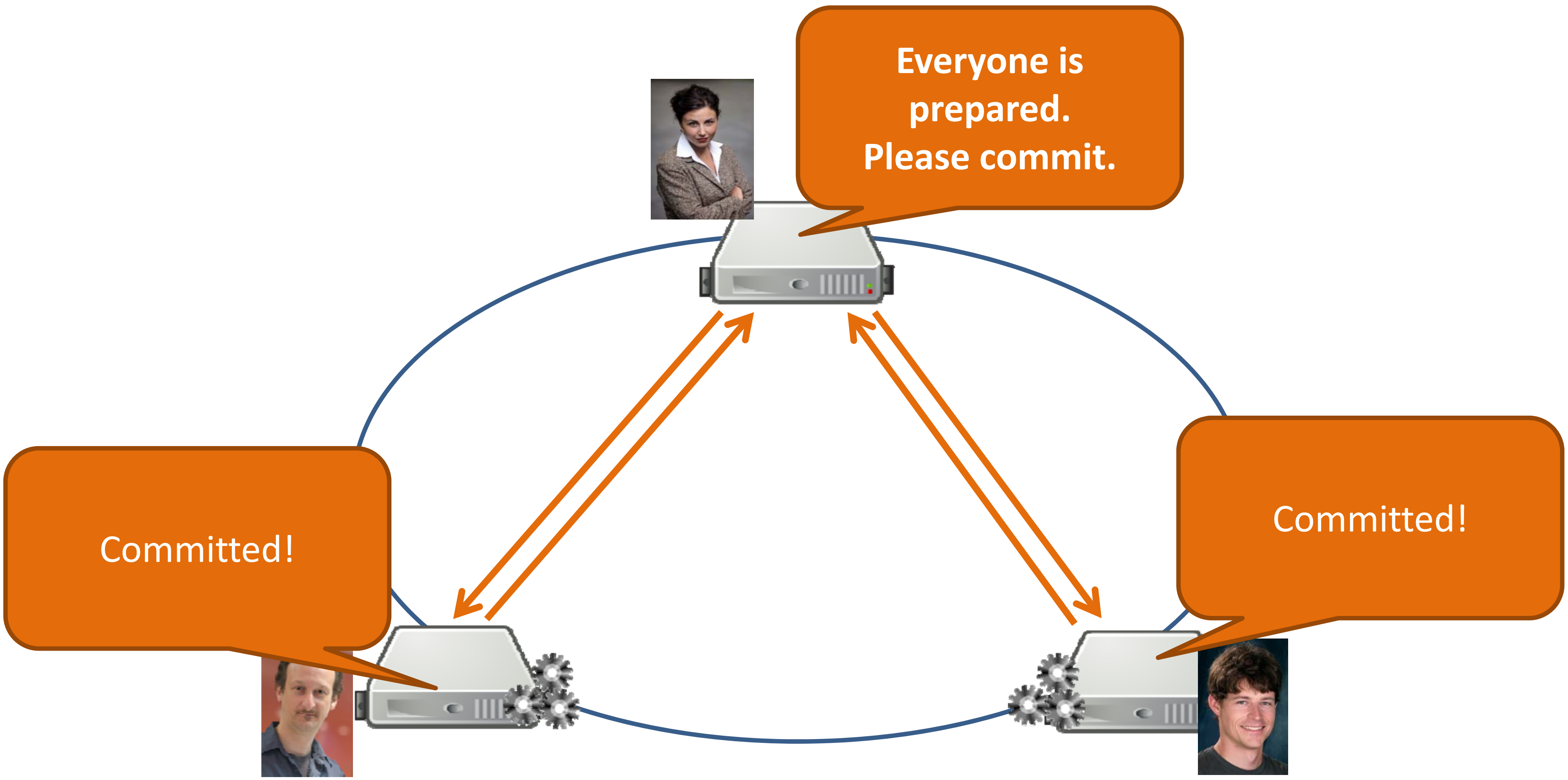
Three-Phase Commit (3PC)

2. Prepare:



Three-Phase Commit (3PC)

3. Commit:



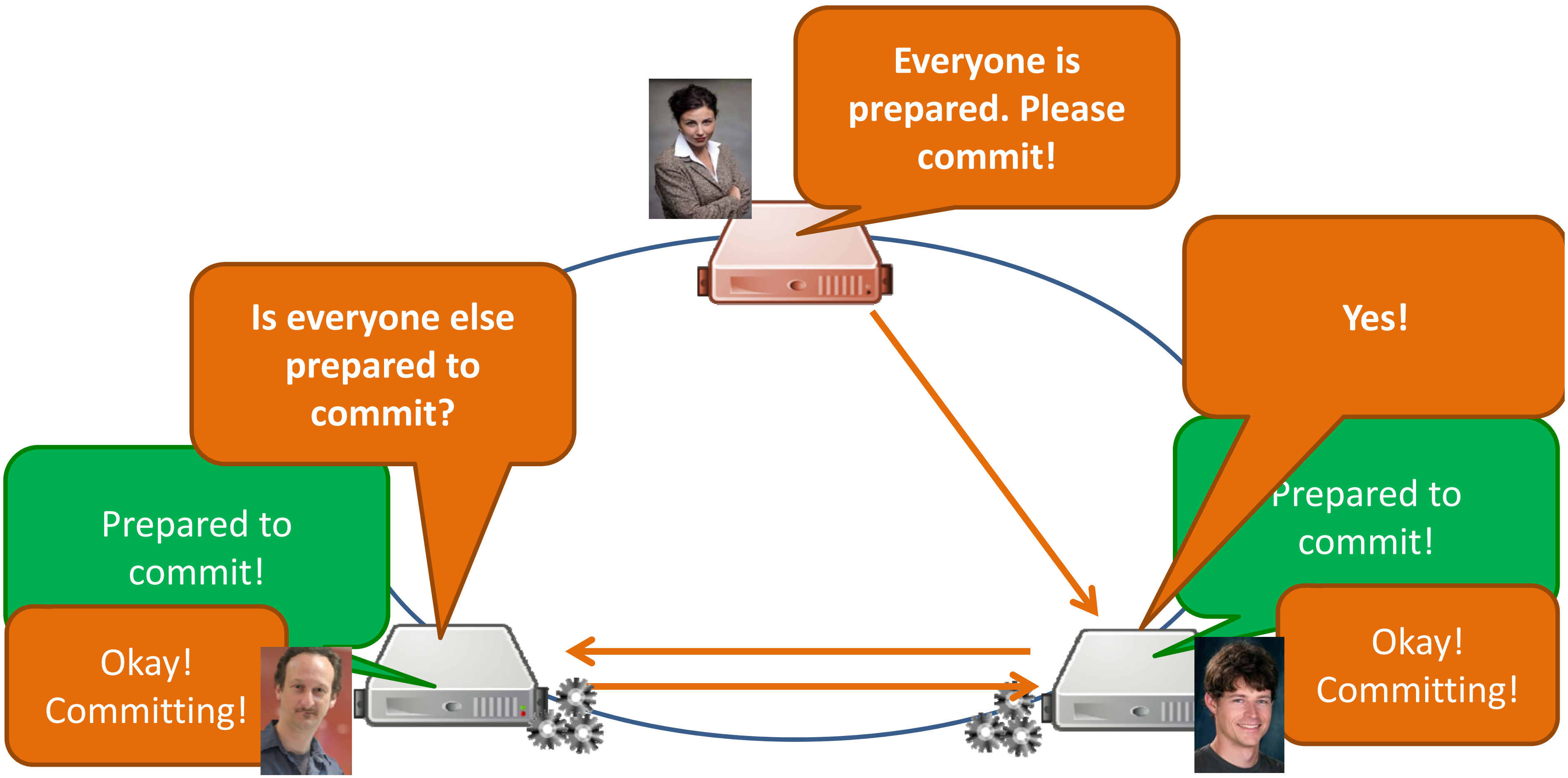
Three-Phase Commit (3PC)

1. Voting: (As before for 2PC)
2. Prepare: If all votes agree, coordinator sends and receives acknowledgements for a "prepare to commit" message
3. Commit: If all acknowledgements are received, coordinator sends "commit" message

- For *n* nodes, in the order of 6*n* messages.
  - *4n* messages as for 2PC
  - +2*n* messages for "prepare to commit"+ "ack."

Three-Phase Commit (3PC)

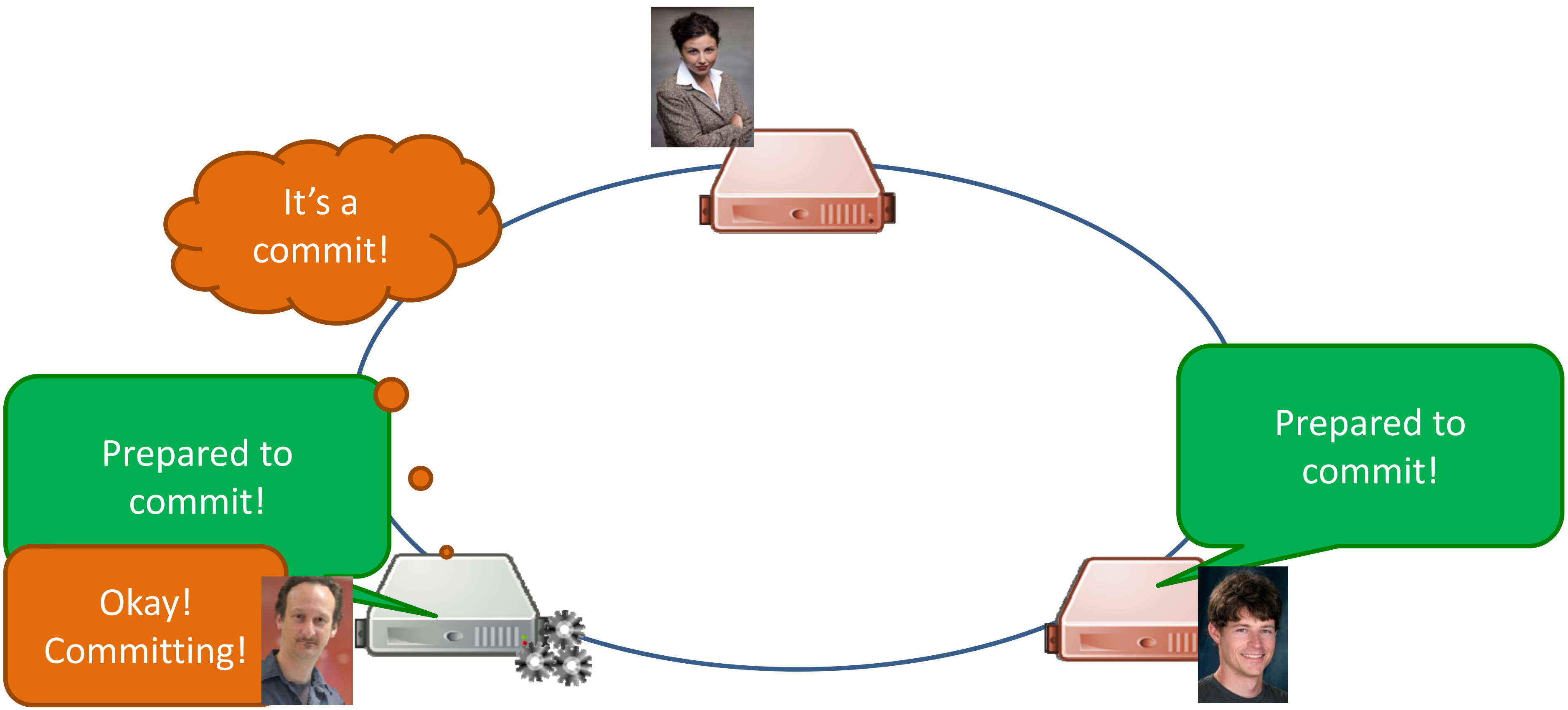
What happens if the coordinator fails?



Three-Phase Commit (3PC)

What happens if coordinator and a cohort member fail?

- Rest of cohort know if abort/commit!

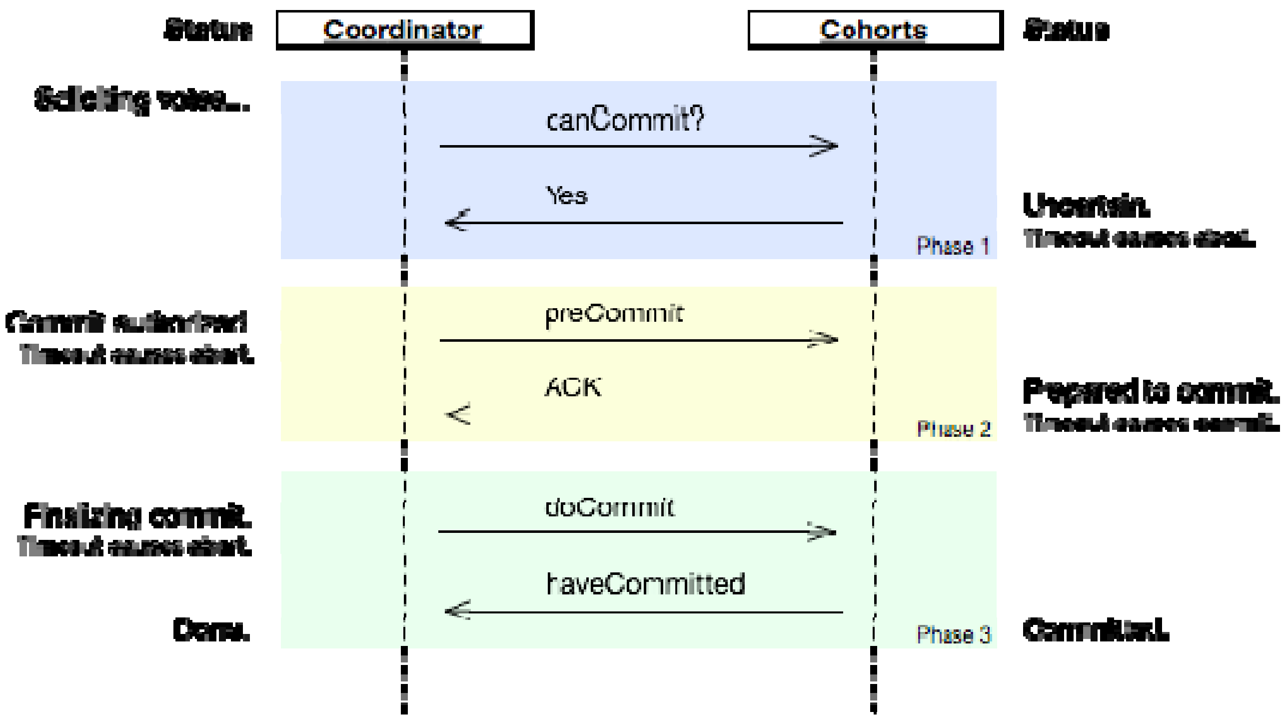


Two-Phase vs. Three Phase

Did you spot the difference?

- In 2PC, in case of failure, one cohort may already have committed/aborted while another cohort doesn’t even know if the decision is commit or abort!
- In 3PC, this is not the case!

3PC useful to avoid locking



Two/Three Phase Commits

- Assumes synchronous behaviour!
- Assumes knowledge of failures!
  - Cannot be guaranteed if there's a network partition!
- Assumes fail–stop errors

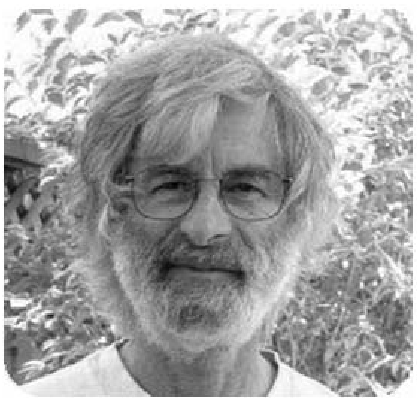
How to decide the leader?

We need a leader for consensus … so what if we need consensus for a leader?

CONSENSUS PROTOCOL: PAXOS

Turing Award: Leslie Lamport

- One of his contributions: PAXOS



LESLIE LAMPORT

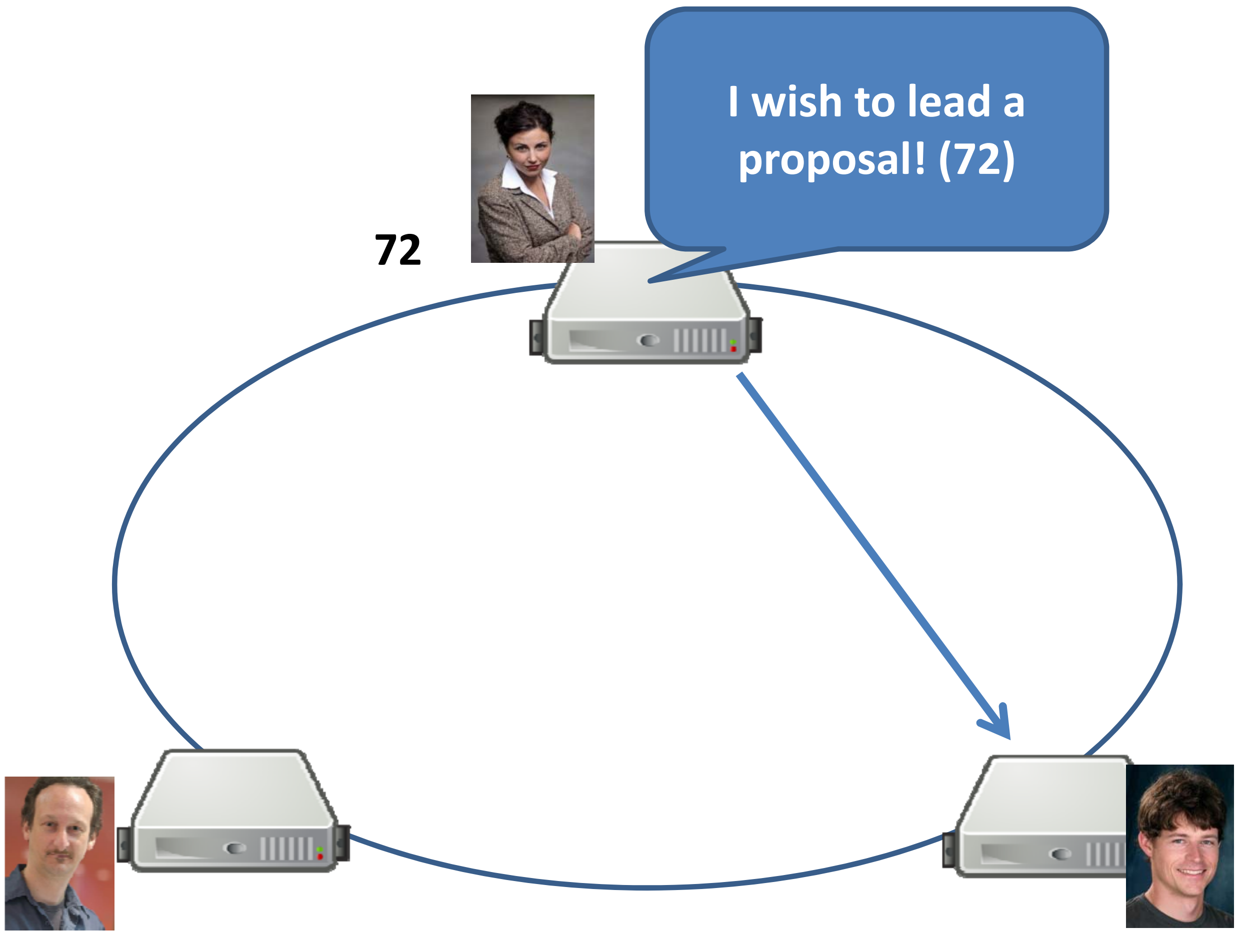
United States – 2013

CITATION

For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.

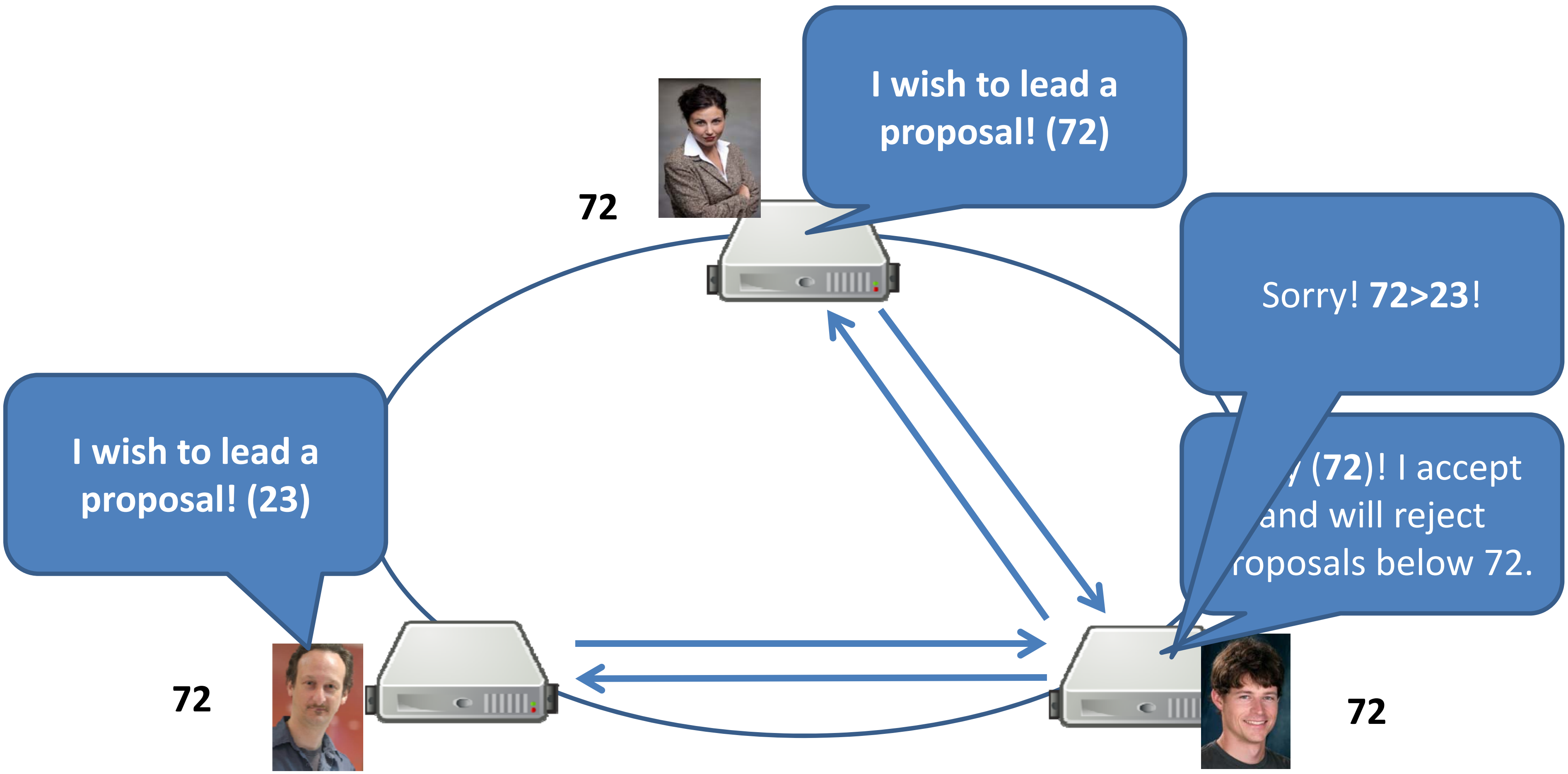
PAXOS Phase 1a: Prepare

- A coordinator proposes with a number *n*



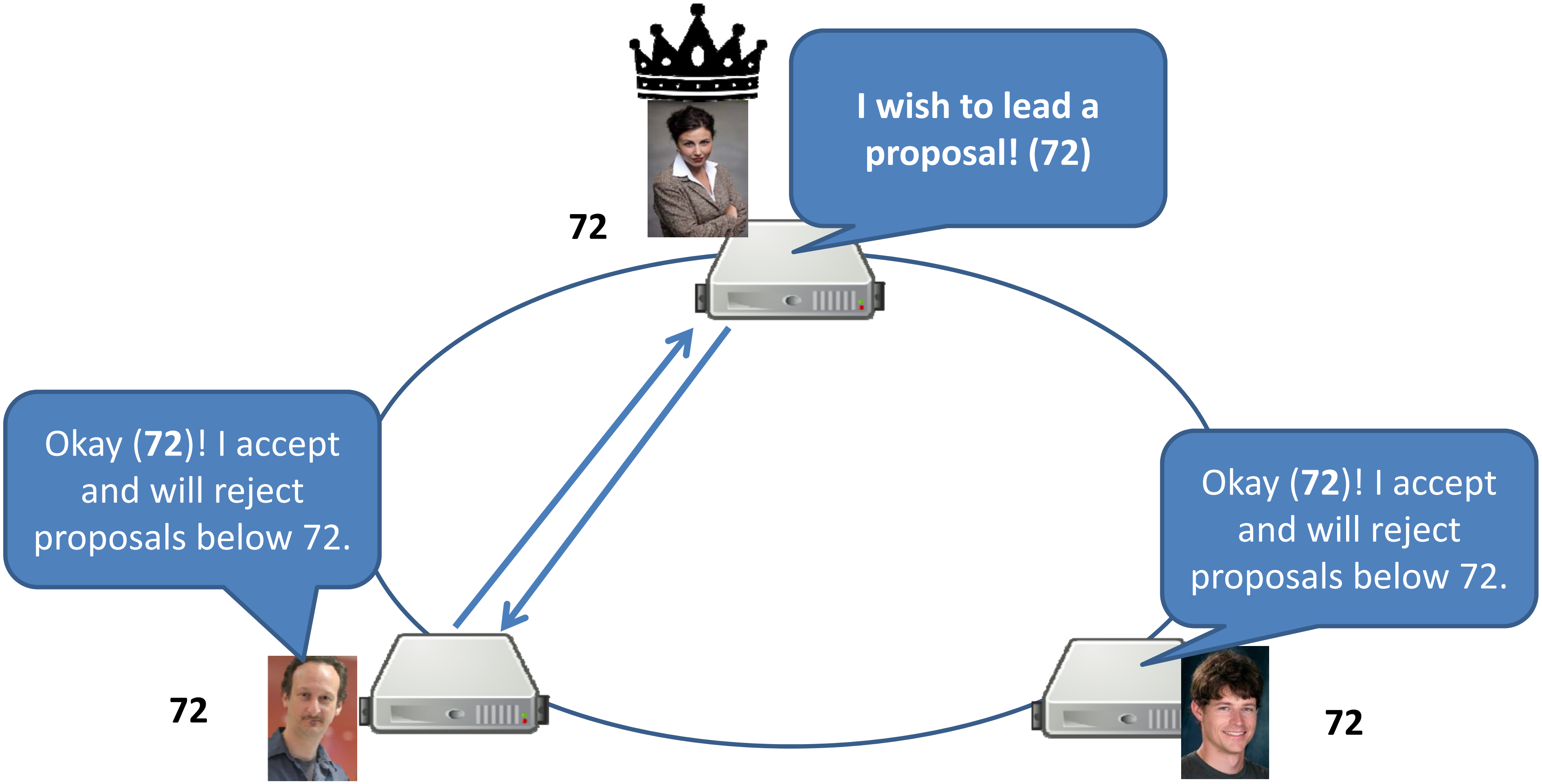
PAXOS Phase 1b: Promise

- By saying "okay", a cohort agrees to reject lower numbers



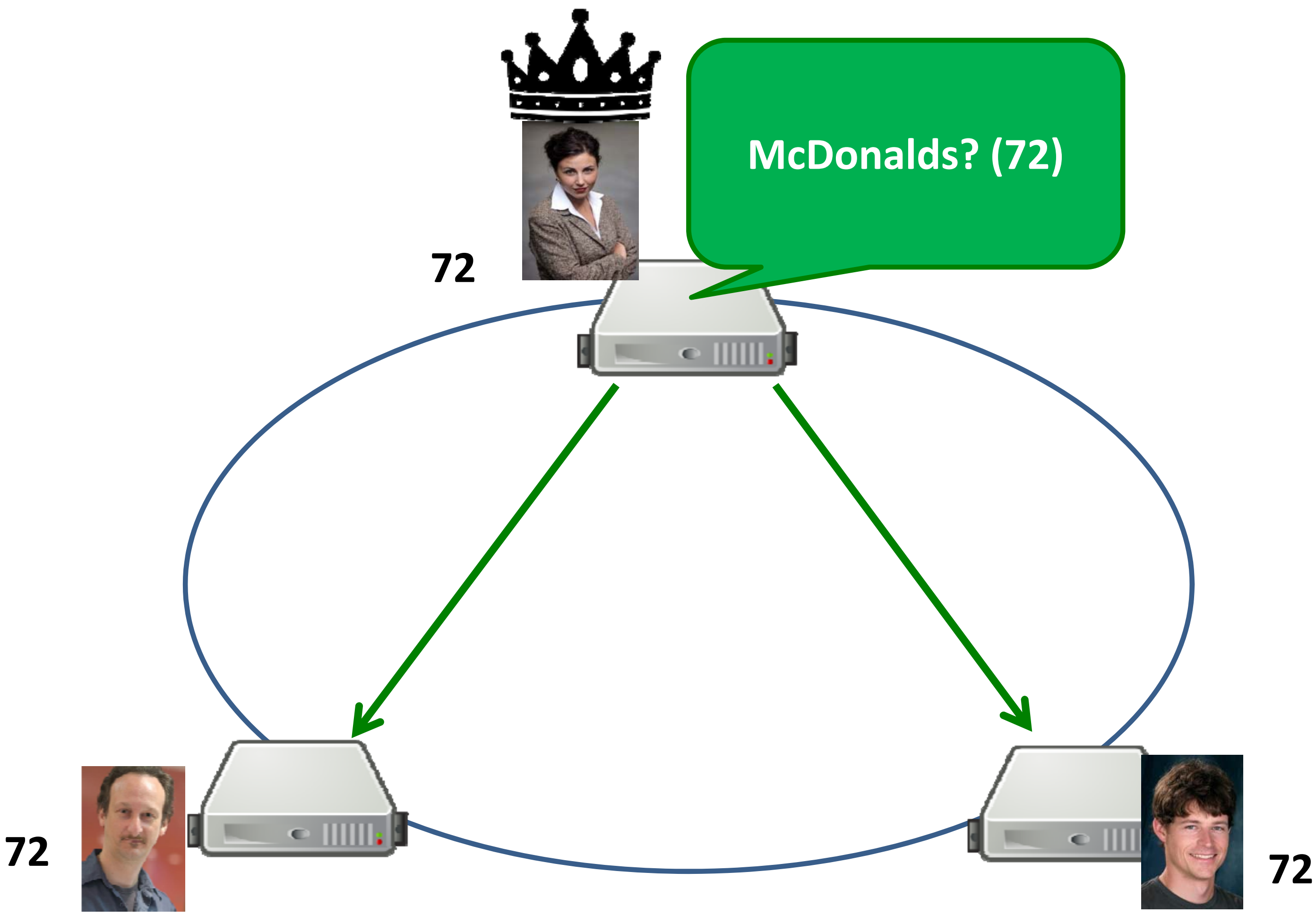
PAXOS Phase 1a/b: Prepare/Promise

- This continues until a majority agree and a leader for the round is chosen …



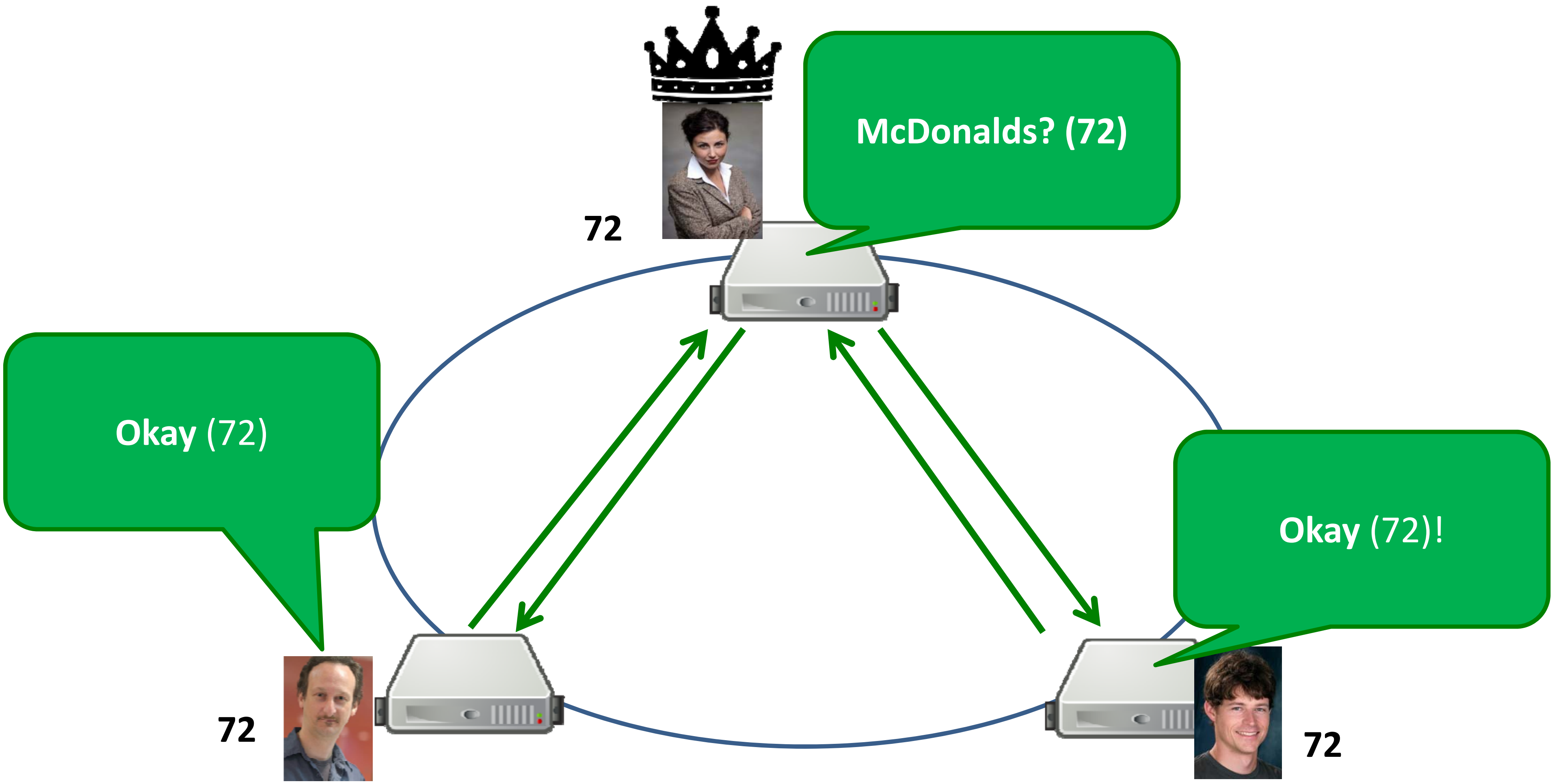
PAXOS Phase 2a: Accept Request

- The leader must now propose the value to be voted on this round …



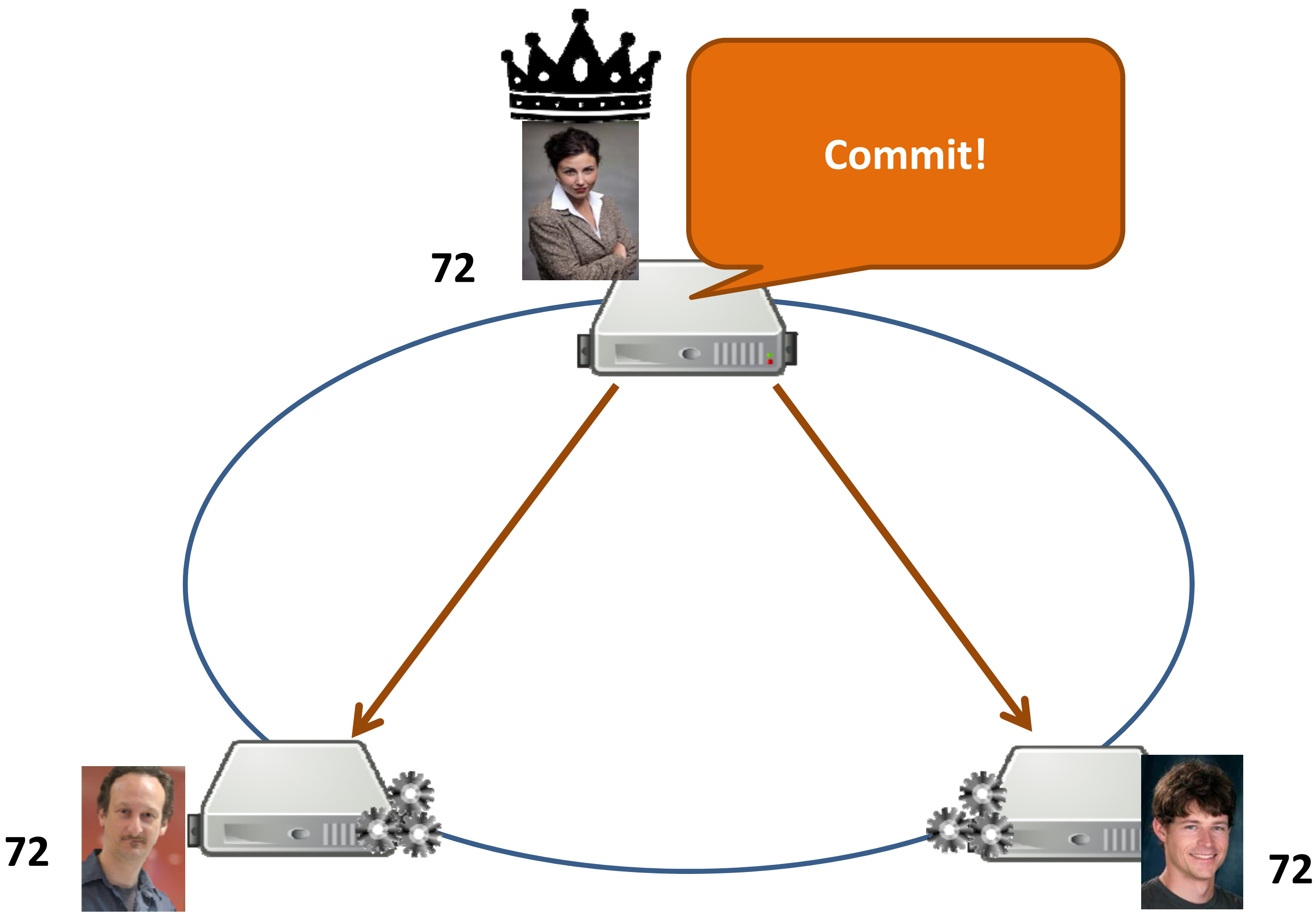
PAXOS Phase 2b: Accepted

- Nodes will accept if they haven't seen a higher request and acknowledge …

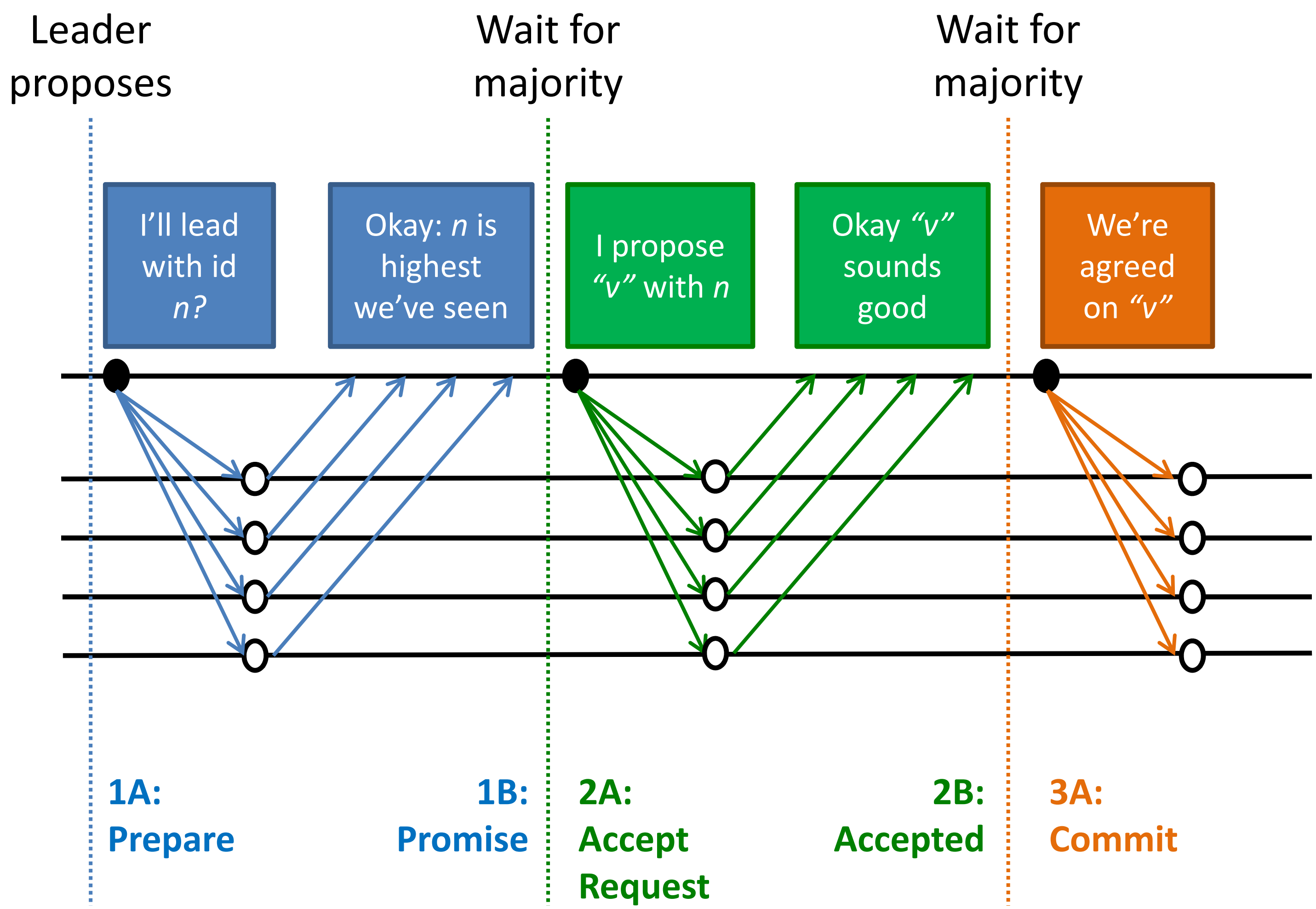


PAXOS Phase 3: Commit

- If a majority pass the proposal, the leader tells the cohort members to commit …



PAXOS Round



Leader proposes
Wait for majority
Wait for majority
I'll lead with id *n?*
Okay: *n* is highest we've seen
I propose "*v*" with *n*
Okay "*v*" sounds good
We're agreed on "*v*"
1A: Prepare
1B: Promise
2A: Accept Request
2B: Accepted
3A: Commit

PAXOS: No Agreement?

- If a majority cannot be reached, a new proposal is made with a higher number (by another member)

PAXOS: Failure Handling

- Leader is fluid: based on highest ID the members have stored
  - If Leader were fixed, PAXOS would be like 2PC
- Leader fails?
  - Another leader proposes with higher ID
- **Leader fails and recovers** (asynchronous)?
  - Old leader superseded by new higher ID
- Partition?
  - Requires majority / when partition is lifted, members must agree on higher ID

PAXOS: Guarantees

- Validity/Integrity:

  (assumes fail-stop errors)

  – Value proposed by a leader

- Agreement/Consistency

  (assumes fewer than half encounter errors and that all errors are fail-stop)

  – A value needs a majority to pass

  – Each member can only choose one value

  – Therefore only one agreed value can have a majority in the system!

PAXOS Guarantees:

- Termination/Liveness:

  (only if at least eventually synchronous)

  - Apply PAXOS in rounds based on the timeout Δ
  - If messages exceed Δ, retry in later round

PAXOS variations

- Some steps in classical PAXOS not always needed; variants have been proposed:
  - Cheap PAXOS / Fast PAXOS / Byzantine PAXOS …

PAXOS In-Use

Google

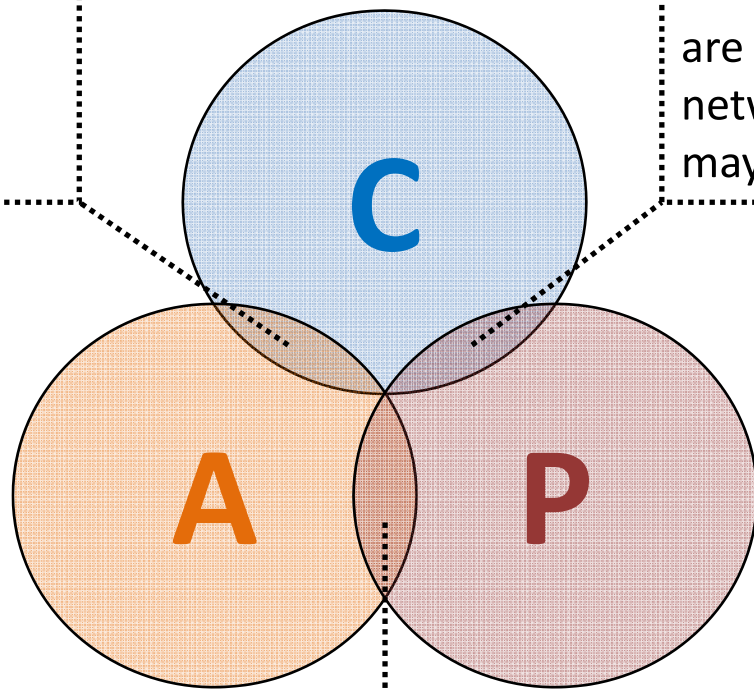
Chubby: “Paxos Made Simple”

RECAP

CAP Systems

CA: Guarantees to give a correct response but only while network works fine (*Centralised / Traditional*)

CP: Guarantees responses are correct even if there are network failures, but response may fail (*Weak availability*)



(No intersection)

AP: Always provides a “best-effort” response even in presence of network failures (*Eventual consistency*)

Consensus for CP-systems

- Synchronous vs. Asynchronous
  - Synchronous less difficult than asynchronous
- Fail–stop vs. Byzantine
  - Byzantine typically software (arbitrary response)
  - Fail–stop gives no response

Consensus for CP-systems

- Two-Phase Commit (2PC)
  - Voting
  - Commit
- Three-Phase Commit (3PC)
  - Voting
  - Prepare
  - Commit

Consensus for CP-systems

- PAXOS:
  - 1a. Prepare
  - 1b. Promise
  - 2a. Accept Request
  - 2b. Accepted
  - 3. Commit

Questions?