NETWORK DISTRIBUTED SYSTEMS

FAILURES & DISTRIBUTED CONSENSUS

Dr. Christina Thorpe

THE PLAYERS

- Choose from a large set of interchangeable terms:
  - Processes, threads, tasks,…
  - Processors, nodes, servers, clients,…
  - Actors, agents, participants, partners, cohorts...
- The term "node" or "actor" is also popular
  - Short and sweet
  - A logical/virtual entity: may be multiple logical nodes per physical machine.
  - General with regard to role and internal structure
  - Tend to use "actor" if self-interest is an issue

PROPERTIES OF NODES/ ACTORS

Essential properties typically assumed by model:

- Private state
  - Distributed memory: model sharing as messages
- Executes a sequence of state transitions
  - Some transitions are reactions to messages
  - May have internal concurrency, but hide that
- Deterministic vs. nondeterministic
- Unique identity vs. anonymous nodes
- Local clocks with arbitrary drift vs. global time strobe (e.g., GPS satellites)

NODE FAULTS AND FAILURES

- **Fail-stop:** Nodes/actors may fail by stopping.
- **Byzantine:** Nodes/actors may fail without stopping.
  - Arbitrary, erratic, unexpected behaviour
  - May be malicious and disruptive
- **Unfaithful behaviour:**
  - Actors may behave unfaithfully from self-interest.
  - If it is rational, then it is expected.
  - If it is expected, then we can control it.
  - Design in incentives for faithful behaviour, or disincentives for unfaithful behaviour.

NODE RECOVERY

- Fail-stopped nodes may revive/restart
  - Retain identity
  - Lose messages sent to them while failed
  - Arbitrary time to restart...or maybe never
- Restarted node may recover state at time of failure.
  - Lose state in volatile (primary) memory.
  - Restore state in non-volatile (secondary) memory.
  - Writes to non-volatile memory are expensive.
  - Design problem: recover complete states reliably, with minimal write cost.

MESSAGES

- Processes communicate by sending messages.
- Unicast typically assumed
  - Build multicast/broadcast on top
- Use unique process identity (pid) as destination.
- Optional: cryptography
  - (optional) Sender is authenticated.
  - (optional) Message integrity is assured.
  - E.g., using digital signatures or Message Authentication Codes.

DISTRIBUTED SYSTEM MODELS

- **Synchronous model**
  - Message delay is bounded and the bound is known.
  - E.g., delivery before next tick of a global clock.
  - Simplifies distributed algorithms
    - "learn just by watching the clock"
    - absence of a message conveys information.
- **Asynchronous model**
  - Message delays are finite, but unbounded/unknown
  - More realistic/general than synchronous model.
    - "Beware of any model with stronger assumptions."
  - Strictly harder/weaker than synchronous model.
    - Consensus is not always possible

MESSAGING PROPERTIES

- Other possible properties of the messaging model:
  - Messages may be lost.
  - Messages may be delivered out of order.
  - Messages may be duplicated.
- Do we need to consider these in our distributed system model?
- Or, can we solve them within the asynchronous model, without affecting its foundational properties?
  - E.g., reliable transport protocol such as TCP

THE NETWORK

- Picture a cloud with open unicast and unbounded capacity/ bandwidth.
  - Squint and call it the Internet.
- Alternatively, the network could be a graph:
  - Graph models a particular interconnect structure.
  - Examples: star, ring, hypercube, etc.
  - Nodes must forward/route messages.
  - Issues: cut-through, buffer scheduling, etc.
  - Bounded links, blocking send: may deadlock.

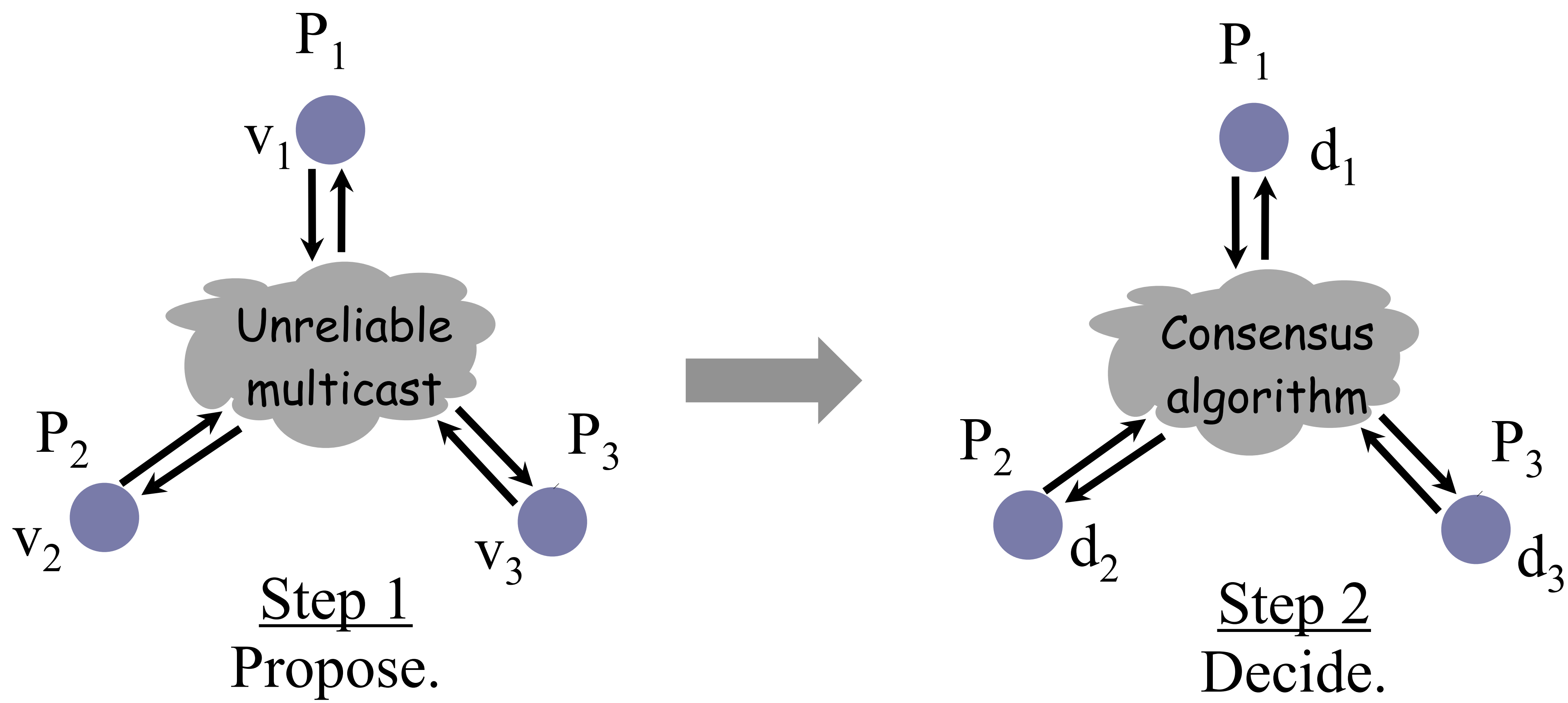
STANDARD ASSUMPTIONS

- For this module, we make reasonable assumptions for general Internet systems:
  - Nodes with local state and (mostly) local clocks
  - Asynchronous model: unbounded delay but no loss
  - Fail-stop or Byzantine
  - Node identity with (optional) authentication
    - Allows message integrity
  - No communication-induced deadlock.
    - Can deadlock occur? How to avoid it?
  - Temporary network interruptions are possible.
    - Including partitions

COORDINATION

- If the solution to availability and scalability is to decentralise and replicate functions and data, how do we coordinate the nodes?
  - data consistency
  - update propagation
  - mutual exclusion
  - consistent global states
  - group membership
  - group communication
  - event ordering
  - distributed consensus
  - quorum consensus

CONSENSUS



Generalizes to N nodes/processes.

PROPERTIES FOR CORRECT CONSENSUS

Termination

Every correct process decides some value.

Validity

If all processes propose the same value v, then all correct processes decide v.

Integrity

Every correct process decides at most one value, and if it decides some value v, then v must have been proposed by some process.

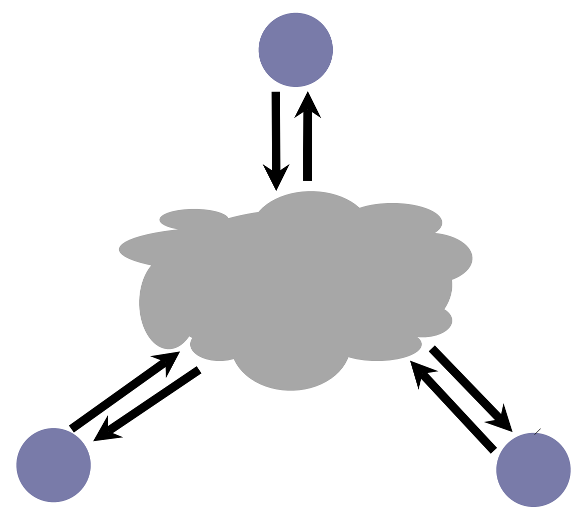
Agreement

Every correct process must agree on the same value.

PROPERTIES OF DISTRIBUTED ALGORITHMS

- Agreement is a safety property.
  - Every possible state of the system has this property in all possible executions.
  - I.e., either they have not agreed yet, or they all agreed on the same value.
- Termination is a liveness property.
  - Some state of the system has this property in all possible executions.
  - The property is stable: once some state of an execution has the property, all subsequent states also have it.

VARIANT I: CONSENSUS (C)



$d_i = v_k$

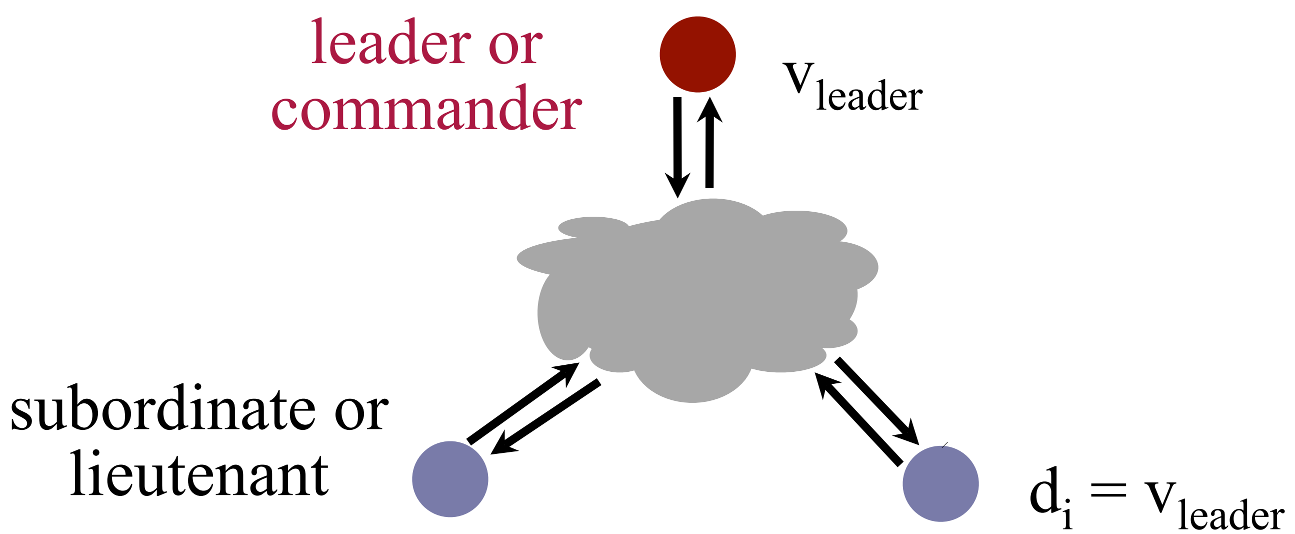
P_i selects d_i from $\{v_0, \ldots, v_{N-1}\}$.

All P_i select d_i as the same v_k .

If all P_i propose the same v, then $d_i = v$, else d_i is arbitrary.

Coulouris and Dollimore

VARIANT II: COMMAND CONSENSUS (BG)



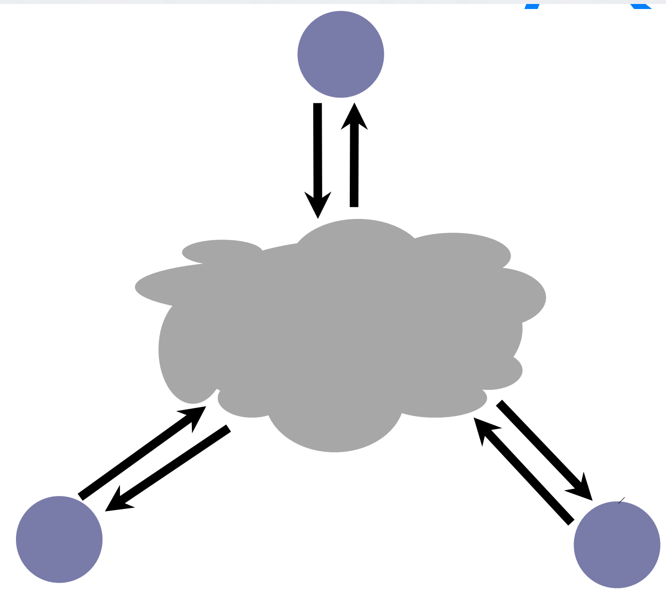
P_i selects $d_i = v_{leader}$ proposed by designated leader node P_{leader} if the leader is correct, else the selected value is arbitrary.

As used in the *Byzantine generals* problem.

Also called *attacking armies*.

Coulouris and Dollimore

VARIANT III: INTERACTIVE CONSISTENCY (IC)



P_i selects $d_i = [v_0 , \ldots, v_{N-1}]$ vector reflecting the values proposed by all correct participants.

Coulouris and Dollimore

FISCHER-LYNCH-PATTERSON (1985)

- No consensus can be guaranteed in an asynchronous communication system in the presence of any failures.
- Intuition: a “failed” process may just be slow, and can rise from the dead at exactly the wrong time.
- Consensus may occur recognisably, rarely or often.
    - e.g., if no inconveniently delayed messages
- FLP implies that no agreement can be guaranteed in an asynchronous system with Byzantine failures either. (More on that later.)

CONSENSUS IN PRACTICE I

- What do these results mean in an asynchronous world?
  - Unfortunately, the Internet is asynchronous, even if we believe that all faults are eventually repaired.
  - Synchronized clocks and predictable execution times don't change this essential fact.
- Even a single faulty process can prevent consensus.
- The FLP impossibility result extends to:
  - Reliable ordered multicast communication in groups
  - Transaction commit for coordinated atomic updates
  - Consistent replication
- These are practical necessities, so what are we to do?

CONSENSUS IN PRACTICE II

- We can use some tricks to apply synchronous algorithms:
  - **Fault masking:** assume that failed processes always recover, and reintegrate them into the group.
    - If you haven't heard from a process, wait longer...
    - A round terminates when every expected message is received.
  - **Failure detectors:** construct a failure detector that can determine if a process has failed.
    - A round terminates when every expected message is received, or the failure detector reports that its sender has failed.
  - **But:** protocols may block in pathological scenarios, and they may misbehave if a failure detector is wrong.

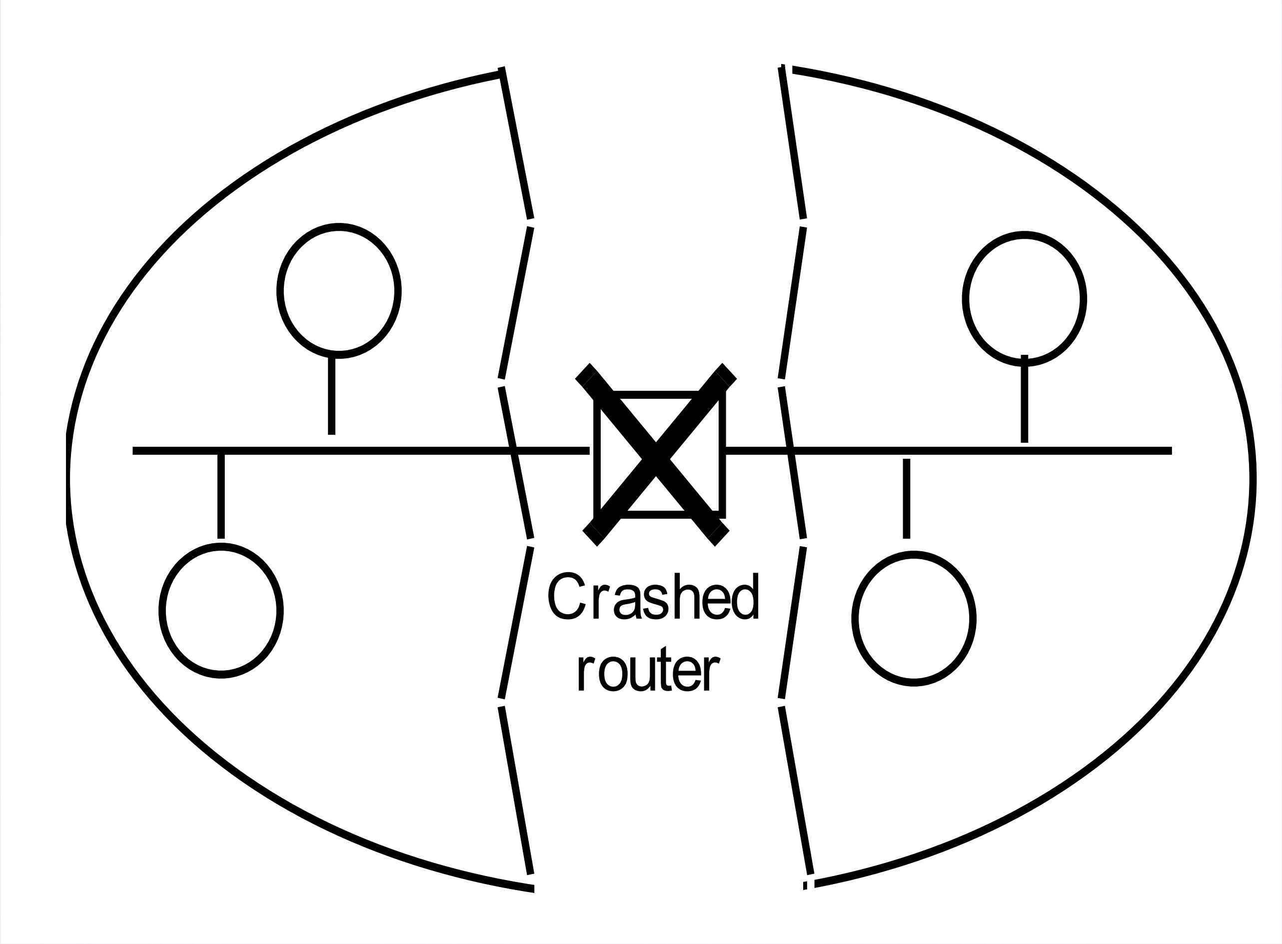
FAILURE DETECTORS

- How to detect that a member has failed?
  - pings, timeouts, beacons, heartbeats
  - recovery notifications
    - "I was gone for awhile, but now I'm back."
- Is the failure detector accurate?
- Is the failure detector live (complete)?
- In an asynchronous system, it is possible for a failure detector to be accurate or live, but not both.
  - FLP tells us that it is impossible for an asynchronous system to agree on anything with accuracy and liveness!

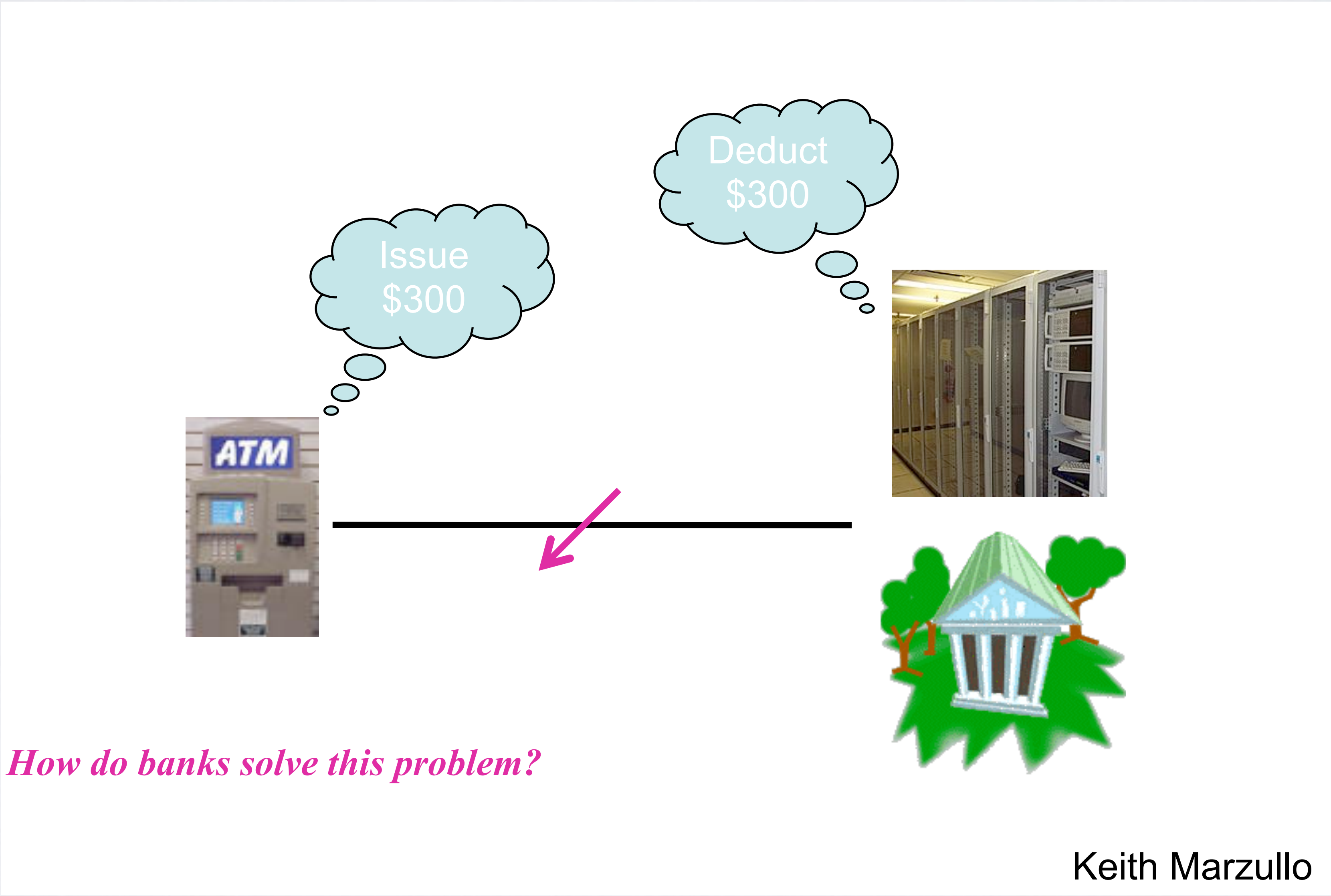
FAILURE DETECTORS IN REAL SYSTEMS

- Use a detector that is accurate but not live.
  - "I'm back....hey, did anyone hear me?"
  - Can't wait forever...
- Use a detector that is live but not accurate.
  - Assume bounded processing delays and delivery times.
  - Timeout with multiple retries detects failure accurately with high probability. Tune it to observed latencies.
  - If a "failed" site turns out to be alive, then restore it or kill it (fencing, fail-silent).
  - Example: leases and leased locks
- What do we assume about communication failures? How much pinging is enough? What about network partitions?

A NETWORK PARTITION



TWO GENERALS IN PRACTICE



COMMITTING DISTRIBUTED TRANSACTIONS

- Transactions may touch data at more than one site.
- Problem: any site may fail or disconnect while a commit for transaction T is in progress.
  - Atomicity says that T does not "partly commit", i.e., commit at some site and abort at another.
  - Individual sites cannot unilaterally choose to abort T without the agreement of the other sites.
  - If T holds locks at a site S, then S cannot release them until it knows if T committed or aborted.
  - If T has pending updates to data at a site S, then S cannot expose the data until T commits/aborts.

COMMIT IS A CONSENSUS PROBLEM

- If there is more than one site, then the sites must agree to commit or abort.
- Sites (Resource Managers or RMs) manage their own data, but coordinate commit/abort with other sites.
  - "Log locally, commit globally."
- We need a protocol for distributed commit.
  - It must be safe, even if FLP tells us it might not terminate.
- Each transaction commit is led by a coordinator (Transaction Manager or TM).