CS 686: Special Topics in Big Data

Byzantine Fault Tolerance

Lecture 13

Looking Ahead

- This week
  - Wrapping up consistency
  - Paper 3
- Next week
  - Big data programming models
  - MapReduce
  - Analysis Methods
- Coming up
  - Streaming analysis
  - Project deadline (10/13)

Today's Agenda

- Spanner
- Chubby
- Two Generals Problem
- Byzantine Generals Problem

Today's Agenda

- **Spanner**
- Chubby
- Two Generals Problem
- Byzantine Generals Problem

Spanner

- Our paper this week is:
  - *Spanner: Google's Globally-Distributed Database*
- Moves to a more relational-style data model rather than key-value, wide column, or documents
- Also provides stronger consistency guarantees
  - You'll be hearing about Paxos, 2PC

While You Read

- What are the trade-offs being made?
- What are the use cases?
  - Would some applications be better or worse with Spanner?
- What parts were difficult / confusing?

Today's Agenda

- Spanner
- **Chubby**
- Two Generals Problem
- Byzantine Generals Problem

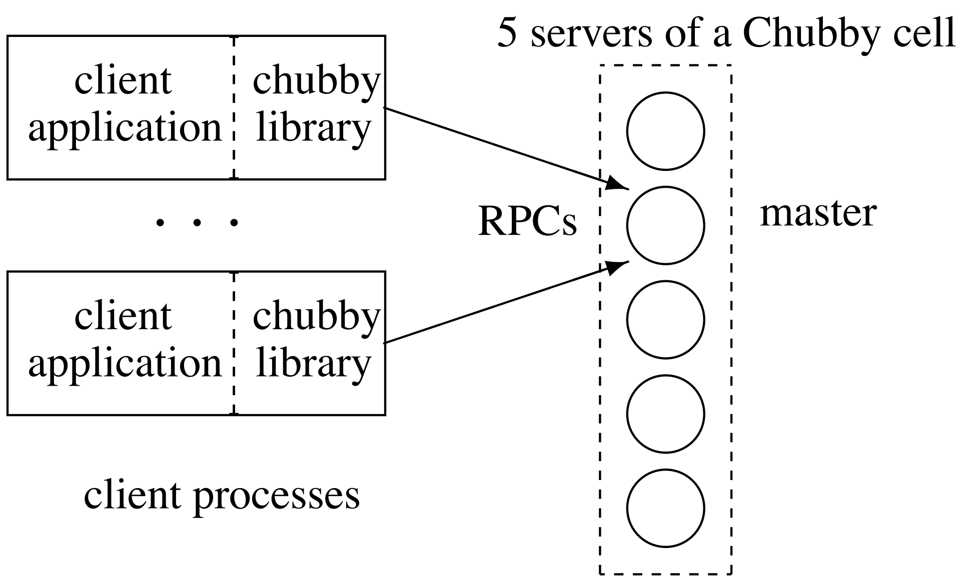
"Chubby is intended to operate within a single company, and so malicious denial-of-service attacks against it are rare. However, mistakes, misunderstandings, and the differing expectations of our developers lead to effects that are similar to attacks."

-- Mike Burrows,
Google, Inc.,
The Chubby lock service for loosely-coupled distributed systems

Chubby

- Chubby is used to coordinate between components at Google
  - Locking, name services, config store
- Partially inspired by the VMS operating system
  - General purpose, global lock service
- Provides coarse-grained locking capabilities and simple storage facilities
  - Based on a file system model

Overview


5 servers of a Chubby cell
client application
chubby library
. . .
client application
chubby library
RPCs
master
client processes

File System Interface

- /ls/foo/wombat/pouch
- ls – 'lock service'
- foo – the chubby **cell**, or instance of the system
  - Found via DNS lookup
- wombat/pouch – directory and file name
  - Files are just arrays of bytes

Abusive Clients

- As mentioned, incorrectly using Chubby is similar to an attack
- Initially, the system had no storage quotas
  - Not intended for a data store
  - Used for one anyway… 1.5 MB file rewritten for **every** client action
- Publish/subscribe
  - Can be used to publish changes, but **not** the intended use case

Lessons Learned

- Developers rarely consider availability
  - Chubby outages have caused cascading effects!
- Be careful with API design expectations
  - The system provides an event notification when a master failover occurs
    - Should help developers know that they need to verify the most recent actions
    - Instead, most applications decided to just crash
- Developers want to use their own favorite language

Today's Agenda

- Spanner
- Chubby
- **Two Generals Problem**
- Byzantine Generals Problem

Failures

- We've spent some time discussing failure scenarios in distributed systems
  - Sometimes it's difficult to know what even counts as a failure
- What are the weaknesses of our DFS's heartbeat scheme?
- We have another type of failures to consider, though:
  - **Byzantine failures**

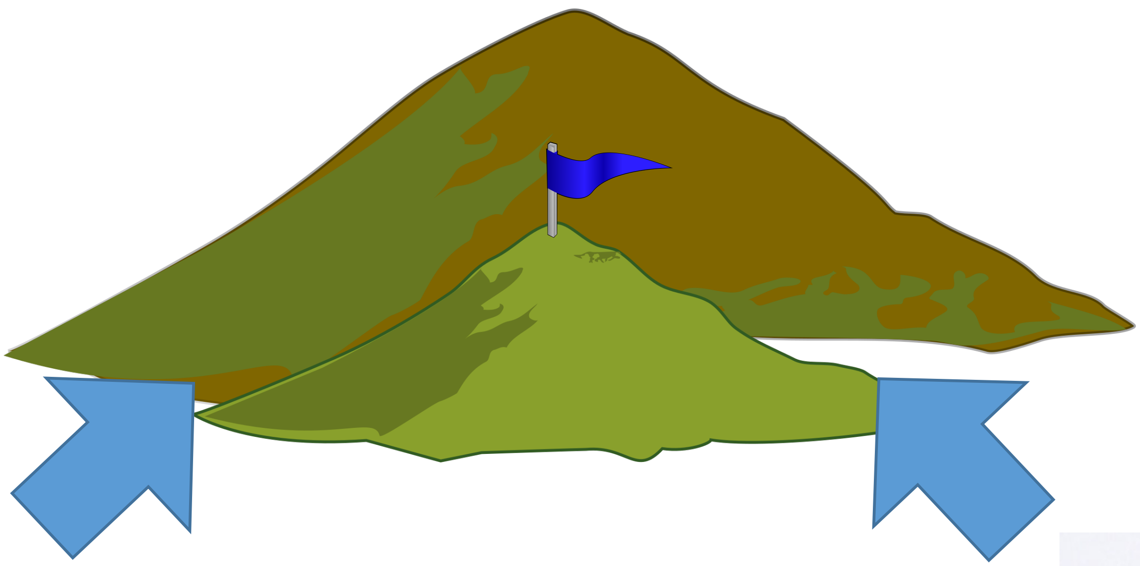
Byzantine Failures

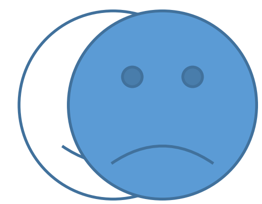
- Any fault presenting different symptoms to different observers
- A machine with failing RAM may happily produce corrupted files/messages
  - Cosmic radiation or faulty hardware can cause bit flips
- Multiple nodes might think they are the coordinator
- Digital vs. analog
  - Bits stuck at ½ instead of 0 or 1

Two Generals Problem

- Suppose two armies are preparing to attack a heavily-fortified enemy base
- If both armies march, then the attack will be a success
  - If only one marches, they will be defeated
- The armies are geographically separated and have an unreliable communication medium (messengers)
- How do we solve this problem?

Two Generals Problem





Note: no army gets to have dragons

Acknowledgment

- One approach is to **acknowledge** the order to attack
  - "We shall attack at dawn on September 25!"
  - "Confirmed: attack at dawn on September 25"
- Of course, then we'd need to acknowledge the acknowledgment
  - Etc.
- Proven to be unsolvable

Reducing Uncertainty

- Another approach would be to continue to send acknowledgments
  - Each increases your confidence in the attack time
  - This wastes resources (dead messengers)
- We could monitor message throughput
- Sequence numbers let us judge the reliability of the communications channel
  - How many messages get lost on average?

Further Complications

- We haven't considered the issue of traitorous messengers
- What happens if a messenger is captured by the enemy and they extract the details of the attack?
  - How would we know this has happened?
- A predetermined protocol could help…
  - But stronger consistency guarantees reduce the likelihood of attacking

Applicability

- We generally don't concern ourselves with military strategy when it comes to big data
- We also have mobile phones, the internet, etc…
- But: one general may be an ATM, and the other your bank
  - Hopefully the general that dispenses cash goes ahead while the one that deducts from your account retreats

Two-Phase Commit

- The two-phase commit protocol we discussed previously is one approach
  - (note: **not** a solution)
- During a transaction, locks are acquired across all replicas
- Replicas attempt to apply the transaction to their log
  - Allows roll-back in the case of disagreement
- If all replicas agree, the transaction is **finalized**

Why 2PC Works

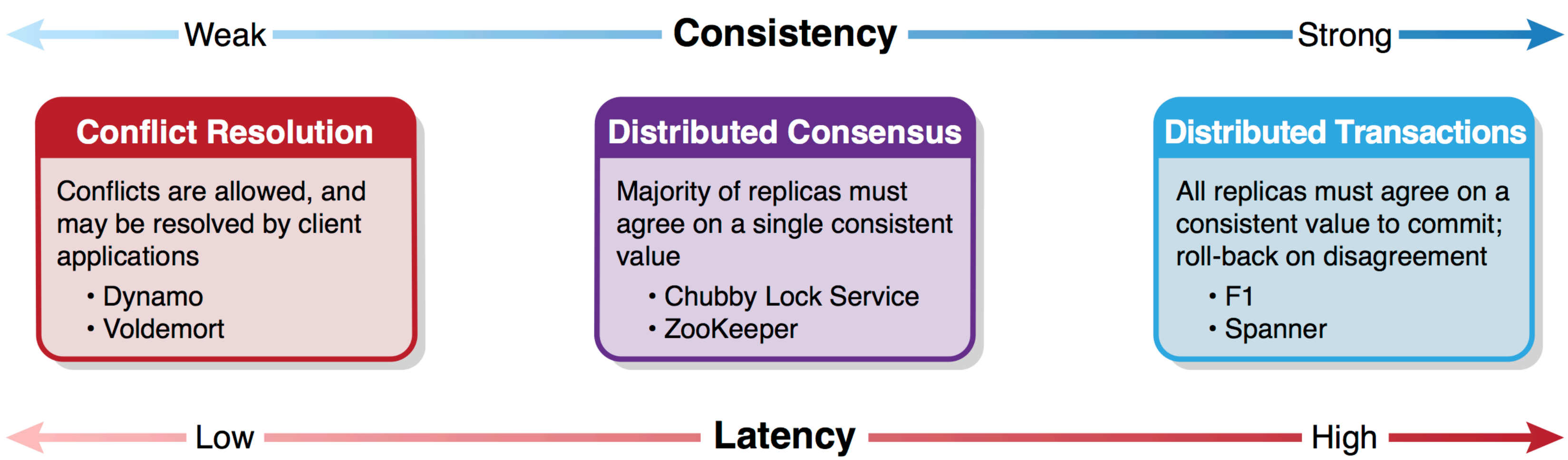
- We essentially **centralize** decision making by introducing a coordinator node
  - Not technically a solution

- **Only** when everyone is in agreement, the decision is broadcast to all participants and the protocol ends
  - We may be waiting a while to attack

2PC Downsides

- Does **not** guarantee liveness
  - The protocol may run indefinitely
- The approach used by Google in Spanner reduces the chance we'll get stuck forever, **but**:
  - A push toward liveness will reduce our confidence in the consistency of the algorithm
- If we can't trust the messengers, we still have a problem

Recall: Consistency-Latency Tradeoff



Today's Agenda

- Spanner
- Chubby
- Two Generals Problem
- **Byzantine Generals Problem**

Byzantine Generals Problem

- Several armies encircle a city
- The generals have to decide: **attack** or **retreat**?
  - Once again, all the generals must **all** agree or they will face their demise
  - This time we'll assume messages are not lost
- The complication: there could be traitorous generals
- Described by Lamport, Shostak, and Pease in *The Byzantine Generals Problem*

Manipulating the Vote

- A general with ill intent could send a vote of ‘yea’ to a certain set of generals and ‘nay’ to others
- The paper proves that to be resilient to such an attack we need:
  - **3m + 1** generals to deal with **m** traitors
  - Each general must be connected to the others by at least **2m + 1** communication paths
  - **m + 1** rounds of messages exchanged

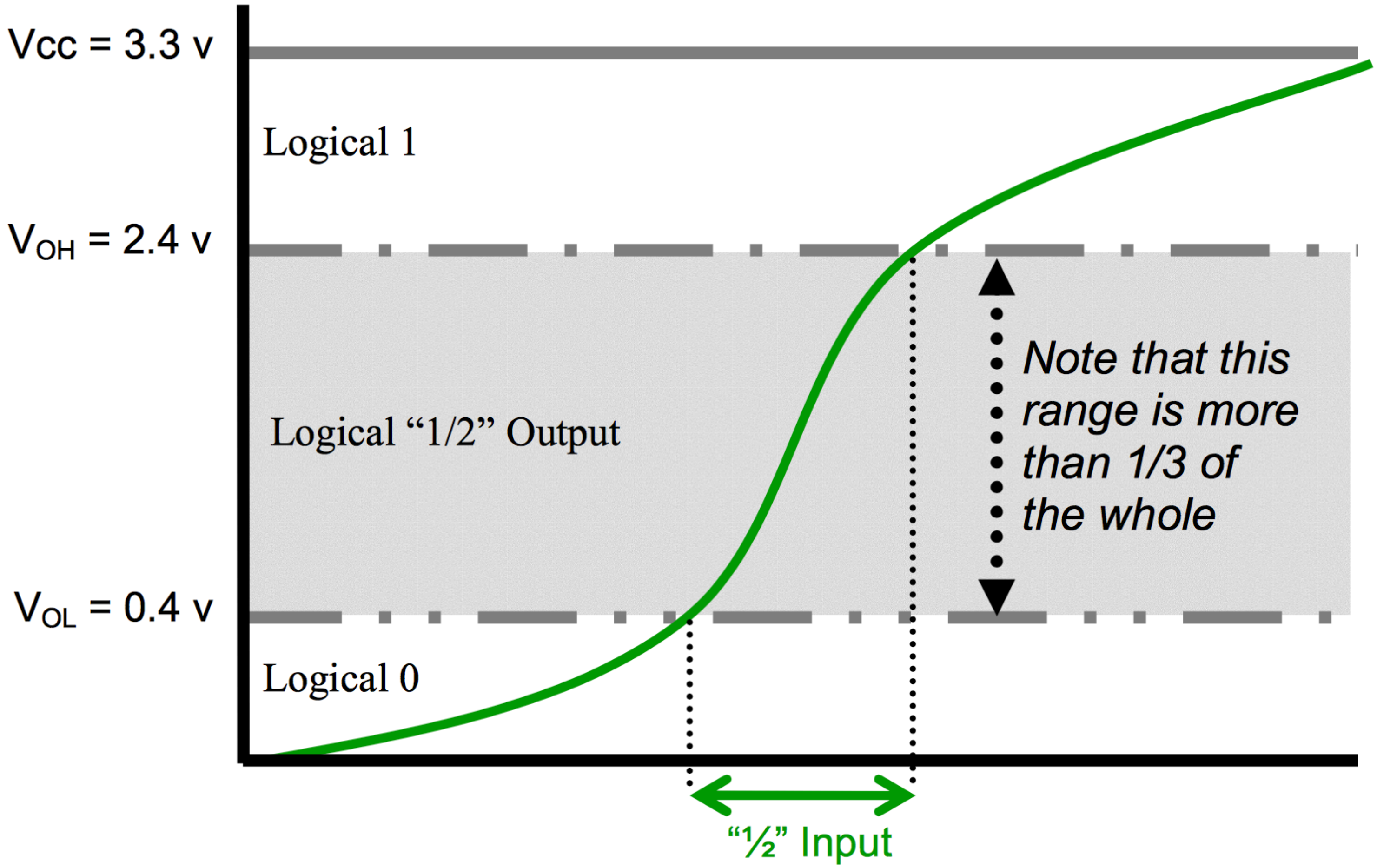
Detecting Traitors

- This approach is fairly intuitive: we basically need to be able to confirm with a majority of generals
- Each round of messages helps us build confidence in the decision
- The problem is, communication is expensive and certainly not guaranteed to work
- At the end of the day, we still can only retreat once we know that a traitor exists

Great, but who cares?

- Byzantine fault tolerance is often overlooked
  - Our computers certainly aren't people, and they aren't traitors!
- Think about all the events Google/Amazon/Facebook process each day
  - A one in a billion event doesn't seem so rare anymore
- NASA, Boeing, Airbus, SpaceX all have to think about Byzantine failures **a lot**

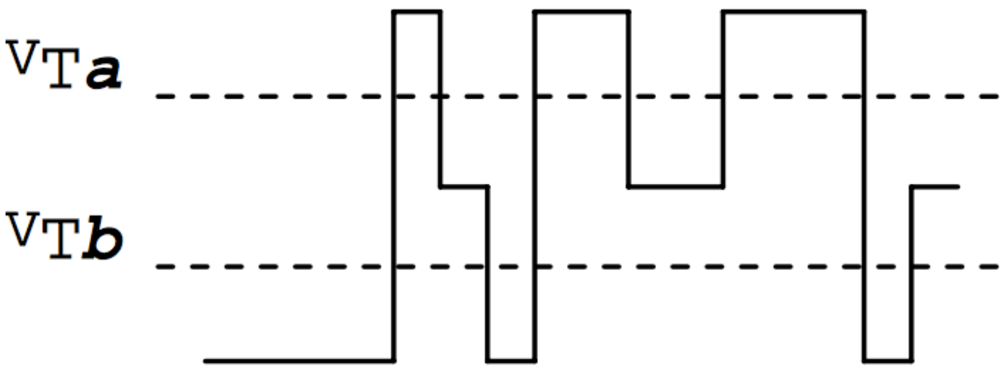
Stuck at ½



Driscoll et al., *Byzantine Fault Tolerance, from Theory to Reality*

Schrödinger's CRC (CCITT-8 CRC)

TX 00001½011½½1110½

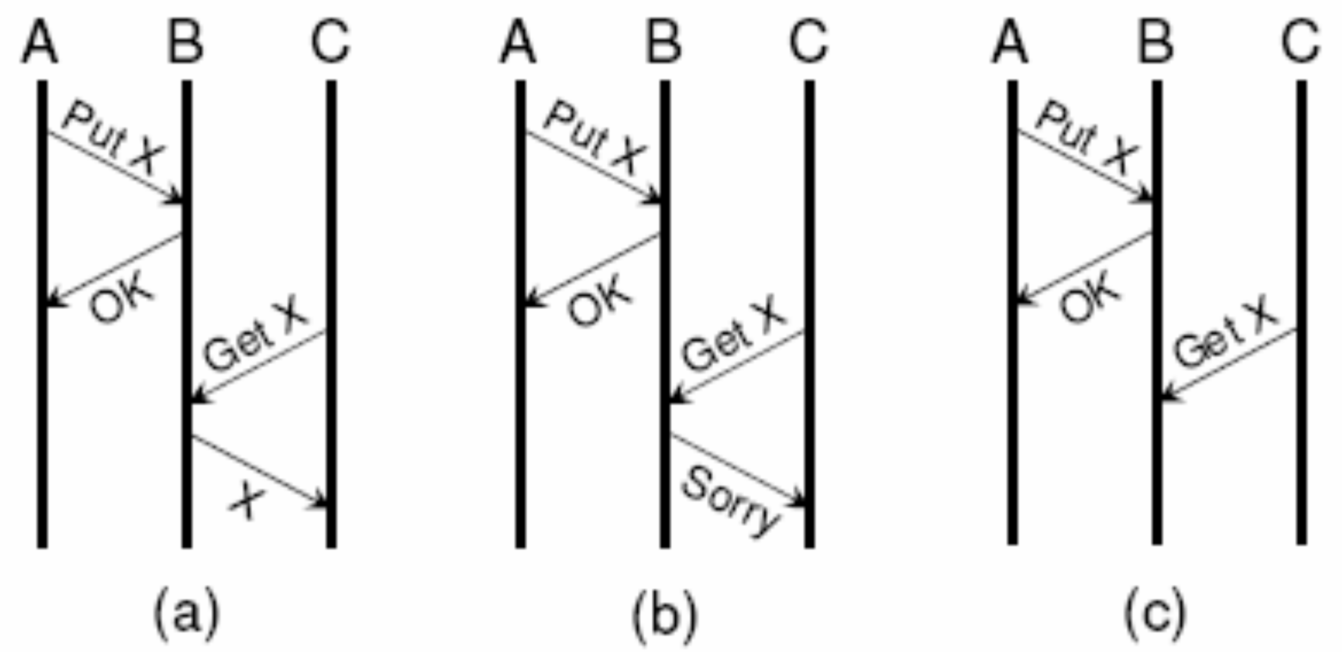


Rxa 0000100110011100

RXb 0000110111111101

Driscoll et al., *Byzantine Fault Tolerance, from Theory to Reality*

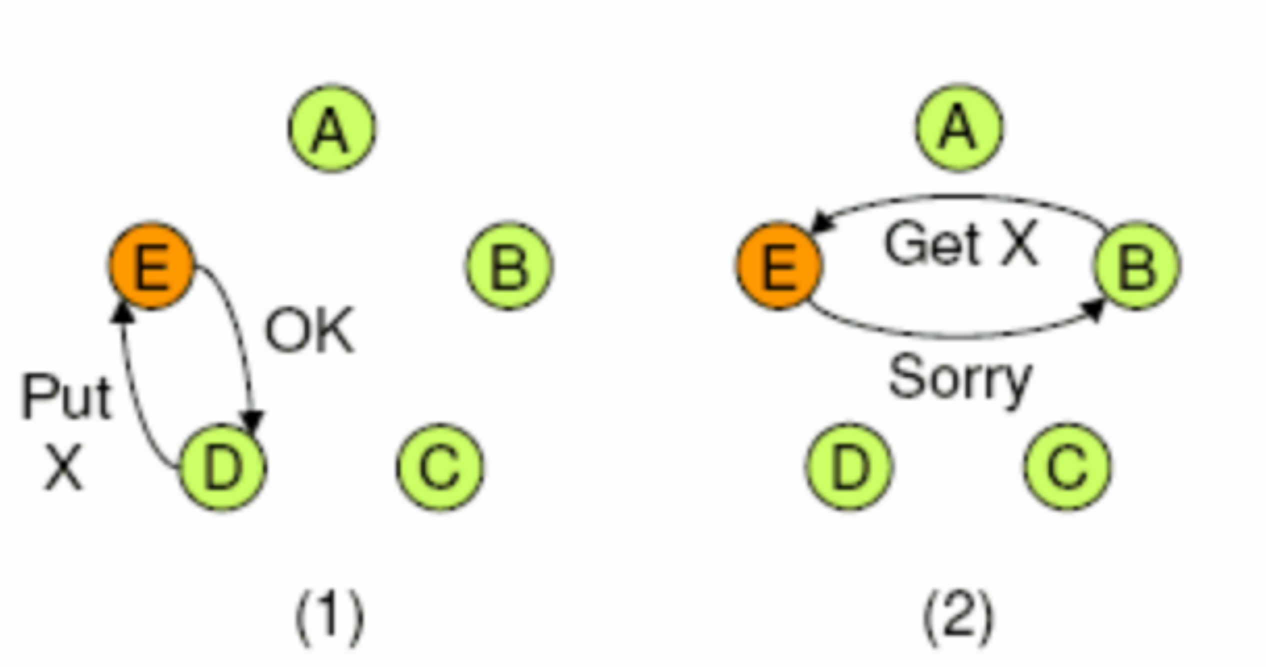
Detecting Faults (1/3)



Node B is: (a) correct, (b) detectably faulty, and (c) detectably ignorant

Haeberlen et al., *The Case for Byzantine Fault Detection*

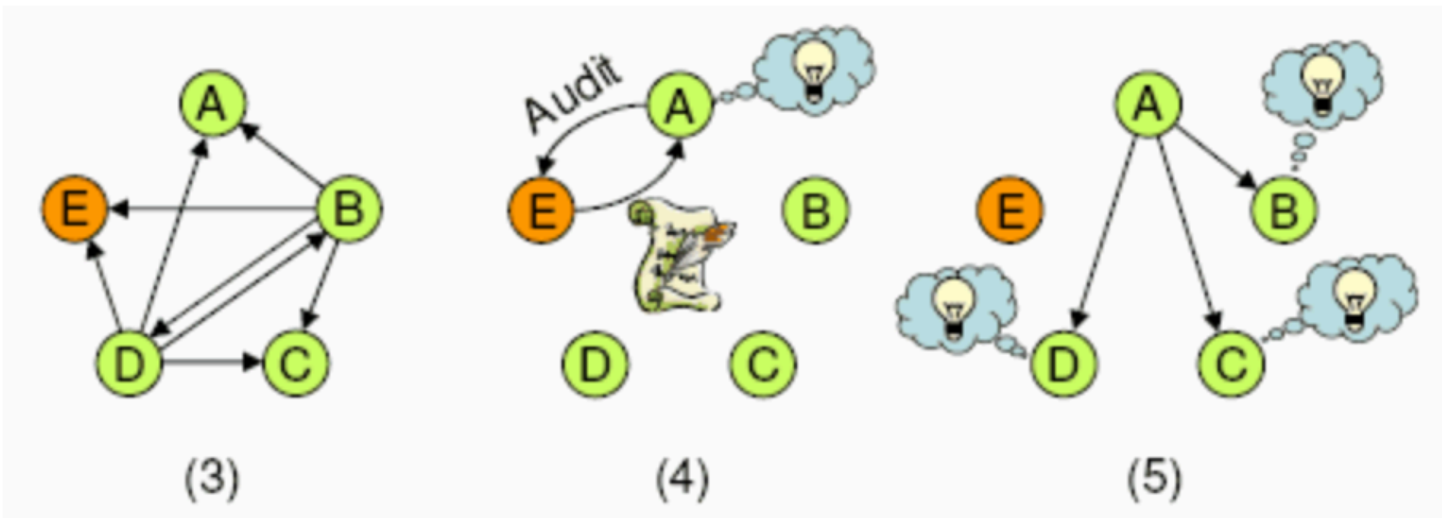
Detecting Faults (2/3)



Node E stores an object for client D (1) and then tries to hide it from client B (2)

Haeberlen et al., *The Case for Byzantine Fault Detection*

Detecting Faults (3/3)



The two clients broadcast authenticators they have obtained from E (3). Later, A audits and exposes E (4). Finally, node A broadcasts its evidence against E, so the other nodes can expose E as well (5).

Haeberlen et al., *The Case for Byzantine Fault Detection*

Detecting Faults: Hardware

- One of the most common methods for dealing with Byzantine failures in hardware is redundancy
- Submit the inputs to two identical components
- Make sure the outputs are the same
- Planes, space shuttles, etc.
  - The downside: this is expensive!

Preventing Faults

- Cryptocurrency systems such as Bitcoin have to deal with attacks from both sides:
  - Byzantine failures can occur on the wide variety of hardware/software participating in the network
  - There is money at stake, so subverting the system has obvious benefits
- **Proof-of-work** schemes help verify goodwill
  - I'll expend some computational resources to prove I'm legitimate