State Machine Replication

Robbert van Renesse

CS6410

The “Cloud”

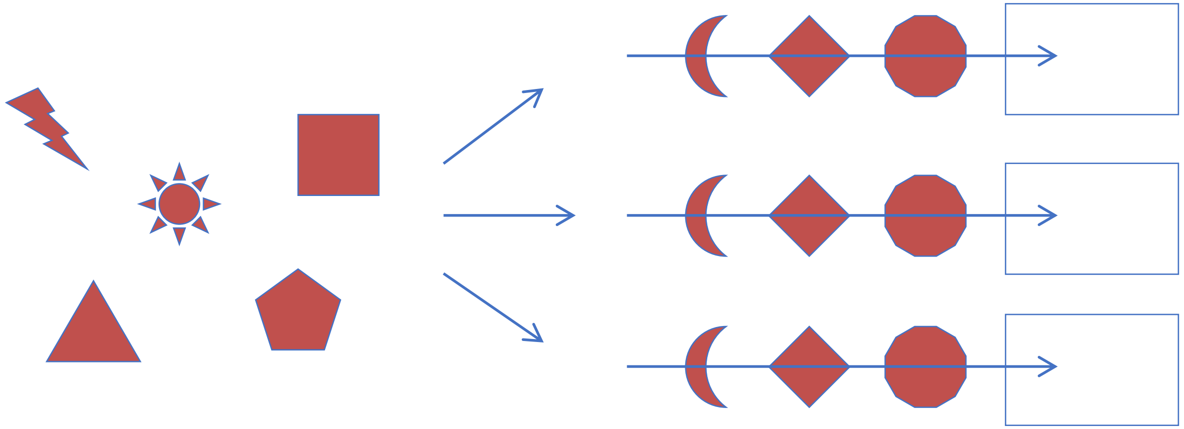
- Cloud-based services run on many computers
- Computers fail sometimes
- But cloud-based services never seem to fail???

How?

- Redundancy *and* Independence
    - run multiple replicas that fail independently
    - but how to keep the replicas in sync?

State Machine Replication (Lamport'78)

- A generic way to tolerate failures
- Run multiple copies of a deterministic state machine and keep them in sync by agreeing on the transitions (operations) and the order in which to apply them



Keeping Replicas Synchronized

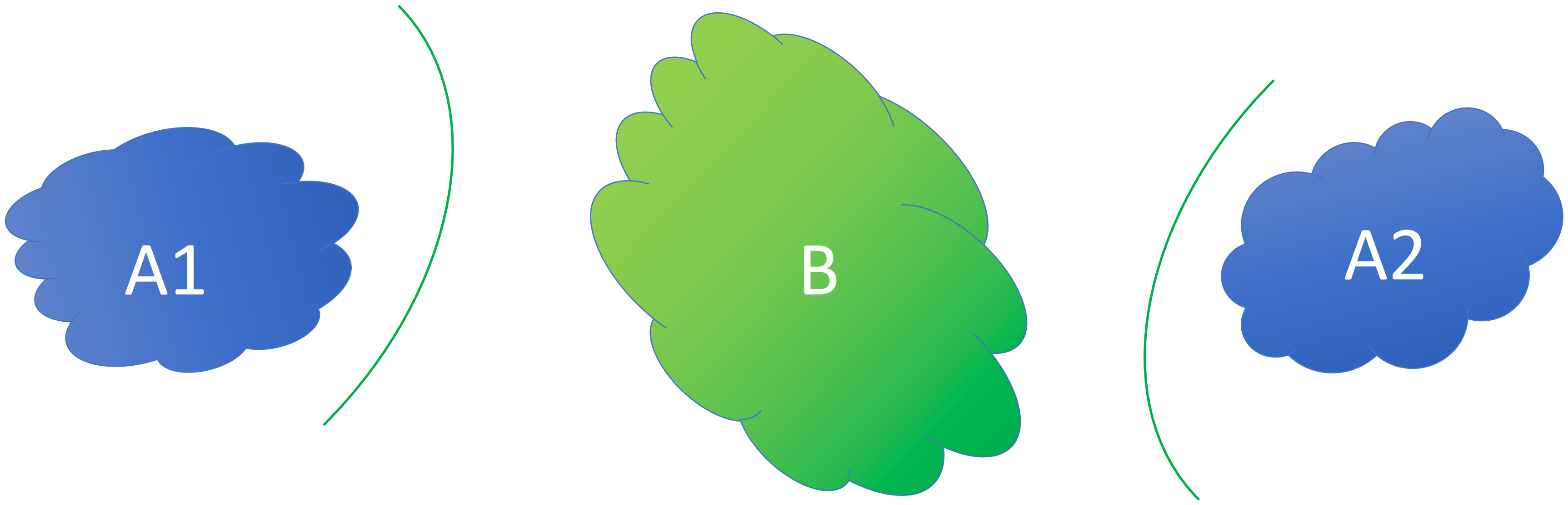
- The replicas agree on the transitions (operations) and the order in which to apply them
- The problem of a set of processes agreeing on something is called "consensus"
- Think of the sequence of transitions as a list of "slots"
- For each slot, State Machine Replication (SMR) has to solve consensus on a set of candidate transitions ("proposals")

How to agree?

- And what is agreement in any case?

Two Generals' Problem

a thought experiment



- "A" can only win if A1 and A2 both attack. If one attacks, it will be decimated
- Generals of armies A1 and A2 can only communicate through messengers
- Messengers can get intercepted and killed when trying to pass through army B

This is an “agreement” problem

- Suppose there exists a deterministic protocol
- Let n be the minimal number of messages required
- Since messages may or may not arrive, omitting the last message should also work
- Therefore, n = 0
- So, only possible if the generals had decided ahead of time (“Global Knowledge”)

2 Generals in practice

When is it safe to garbage collect TCP endpoints?

They have to agree on the fact that the connection has terminated

A1 → A2: let's terminate

A2 → A1: ok, let's (unfortunately, gets lost)

- A2 cannot decide to garbage collect because it may leave A1 hanging

A1 → A2: let's terminate (retransmission)

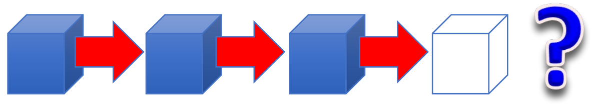
A2 → A1: ok, let's

- A2 still cannot terminate for same reason as before
- A1 receives the message, but needs to inform A2 so
- …

- In practice, time-outs are used

What is Consensus?

- A way for multiple participants to agree on
  - the next update to perform in a replicated service
  - a leader
  - whether to abort or commit a transaction
  - a recovery action after a failure
  - *the next block in a block chain*



- Surprisingly hard with participant and network failures
- Even harder in the face of *asynchrony*
  - complete lack of bounds on latency

Consensus Formalized

- Agreement:
  - if two replicas decide, they must decide the same proposed operation
- Validity:
  - a replica can only decide an operation that was proposed by some replica
    - without this, replicas could just decide “no-op” each time
- Termination:
  - a correct (non-crashing) replica must eventually decide (assuming at least one operation was proposed)

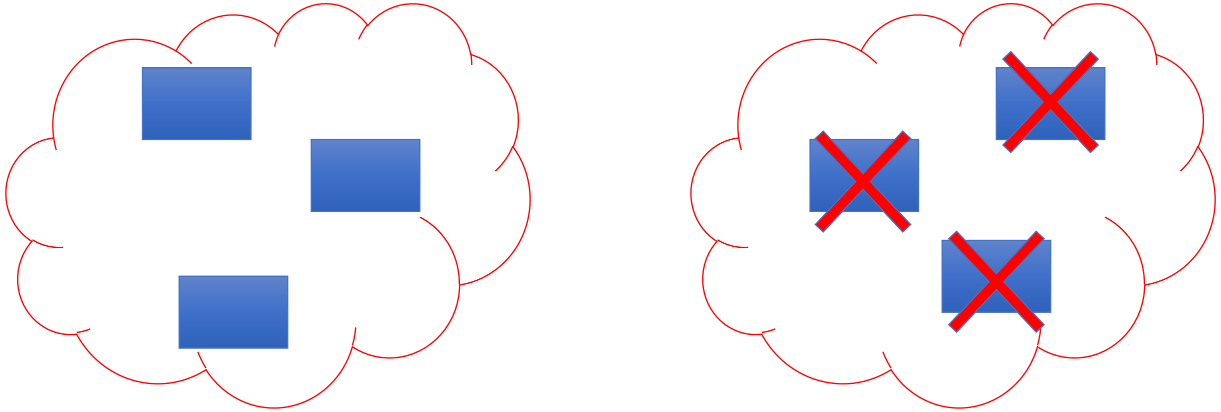
Lower Bound on number of participants

In an asynchronous system with crash failures, you need at least $2f + 1$ replicas to tolerate f crash failures

- $2f$ is not enough: consider the difference between two groups of f processes being separated by a network partition and one group of those processes crashing: can the other group see the difference?

indistinguishability argument

($f = 3$)



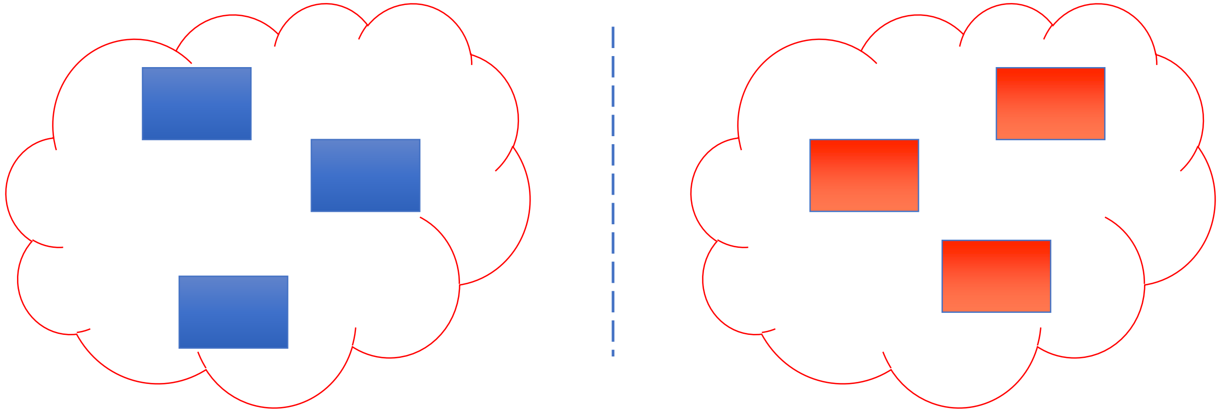
Lower Bound on number of participants

In an asynchronous system with crash failures, you need at least $2f + 1$ replicas to tolerate f crash failures

- $2f$ is not enough: consider the difference between two groups of f processes being separated by a network partition and one group of those processes crashing: can the other group see the difference?

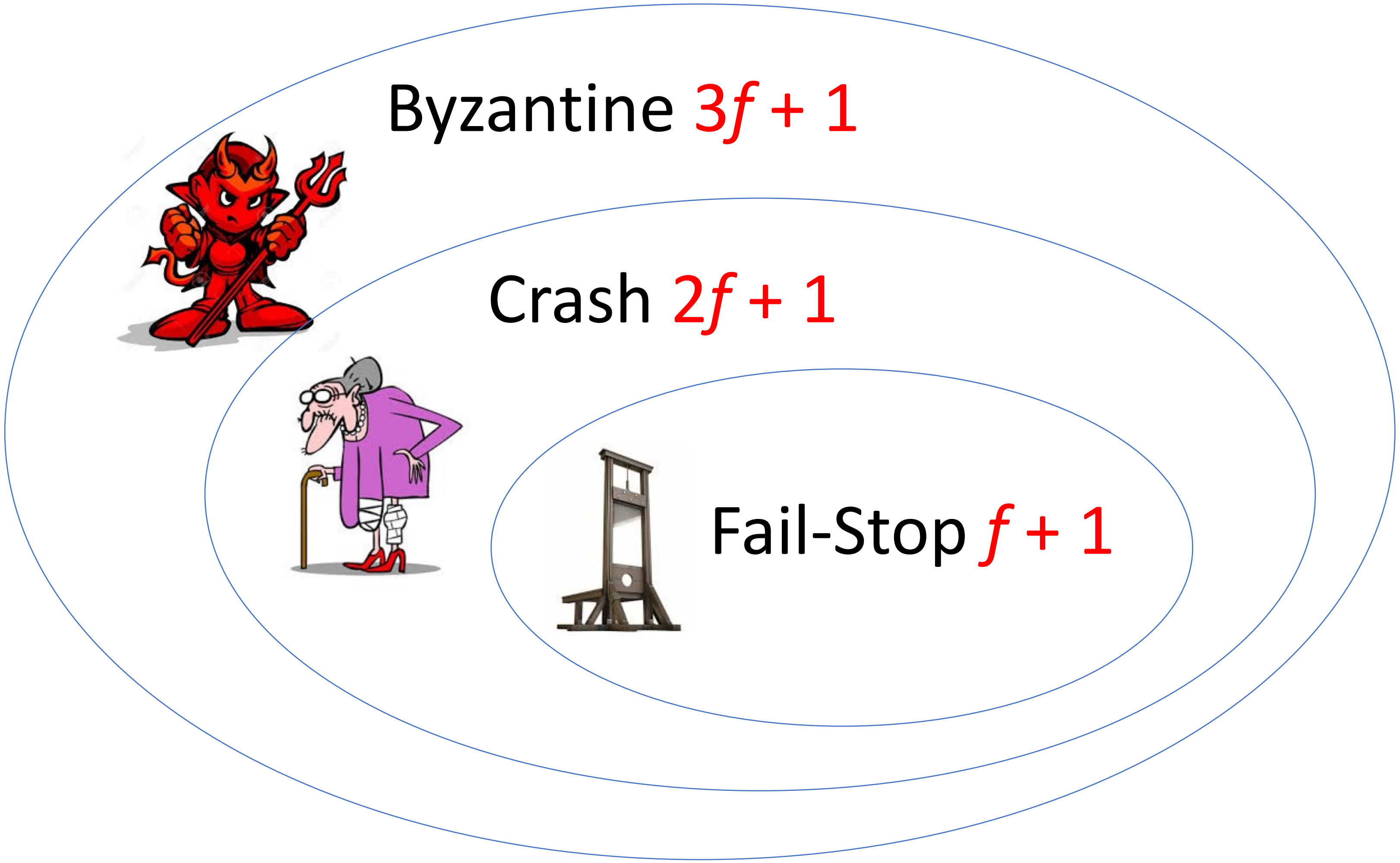
indistinguishability argument

($f = 3$)



if $2f$ were enough, each group could decide independently of the other

Other Lower Bounds



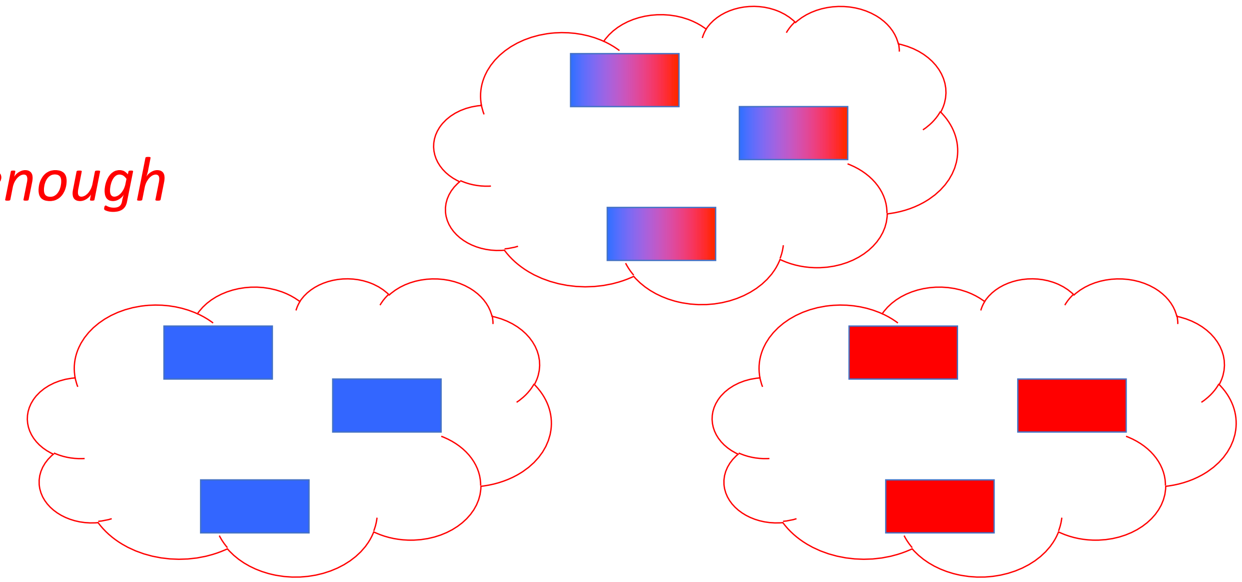
Lower Bound with Byzantine Failures

In an asynchronous environment, you need at least $3f + 1$ participants to tolerate f Byzantine failures

indistinguishability

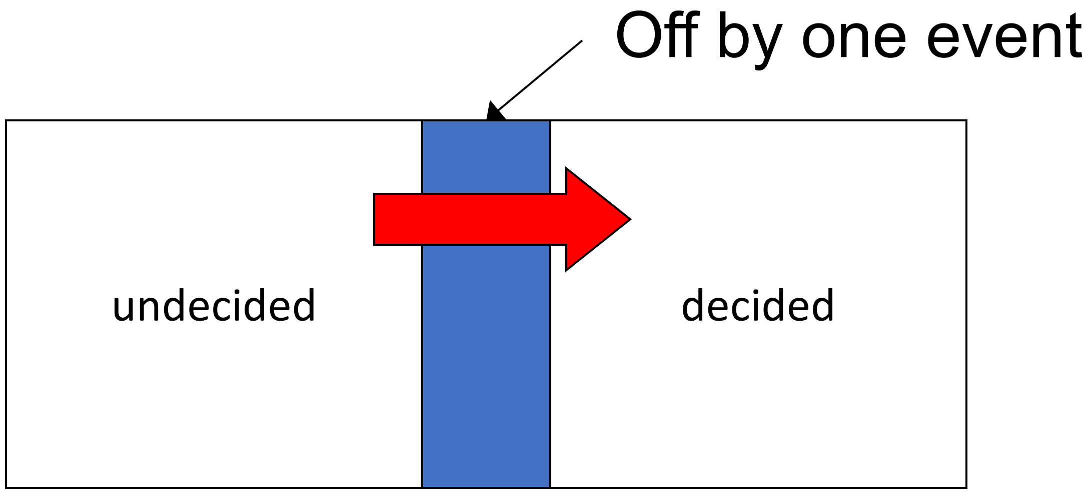
argument: 3f is not enough

(f = 3)

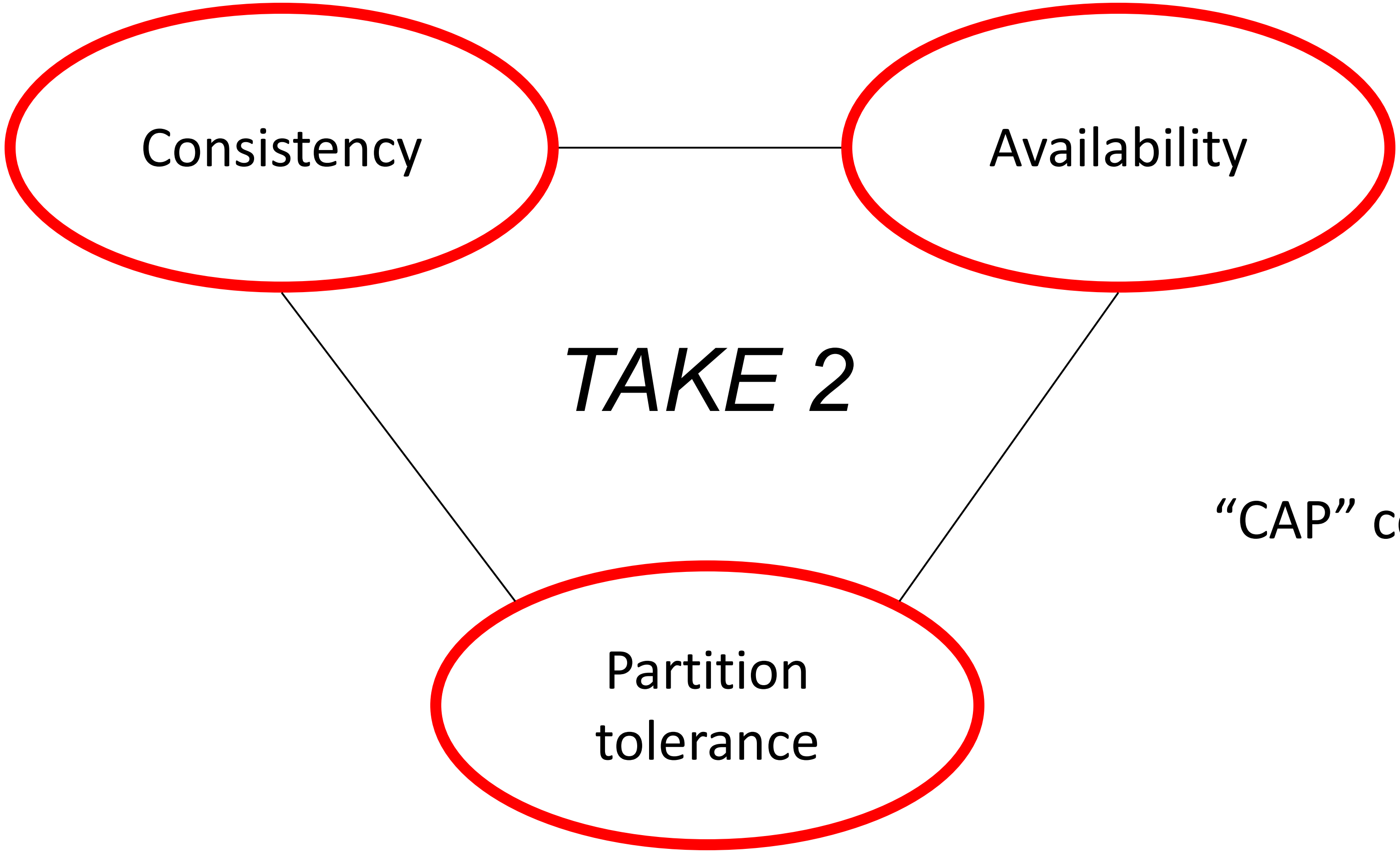


Solving consensus is hard…

Crash failures + no assumptions about timing ⇒ *solving consensus is impossible* (FLP’83, FLP’85)



Add Network Failures…



“CAP” conjecture

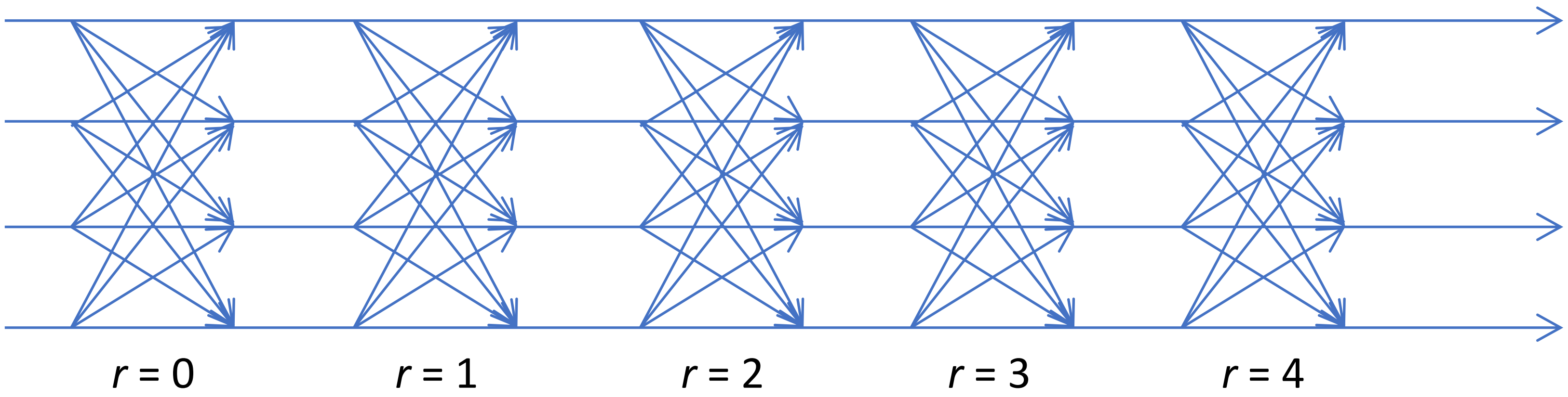
Example consensus protocol with $3f + 1$ processes: setup

- Asynchronous environment
- $3f + 1$ processes, at most f crash failures
  - note: $3f + 1$ is more than the lower bound $2f + 1$
    - this protocol will not be optimal in the number of processes
- The processes run *rounds* of communication
- Each process maintains a round number r and an estimate e
- Initially $r = 0$ and e is the proposal of the process

Protocol with $3f + 1$ processes

1. Broadcast $< r, e >$ "vote" (including to self)
2. Wait for $2f + 1$ votes (out of $3f + 1$)
   - *Note*: because as many as f may fail, this is the maximum a process can safely wait for
3. If a majority of the $2f + 1$ votes contains the same proposal, change e to that proposal
   - *Note*: because $2f + 1$ is odd, there cannot be a tie
4. If not, set e to a proposal in any of the votes received
5. If all votes contain the same proposal (unanimity), decide that proposal
6. $r := r + 1$
7. Repeat (go to Step 1, starting next round)

Approximate Message Behavior



Example Run with $f = 1$

	Process 1	Process 2	Process 3	Process 4
Vote 0	RED	RED	BLUE	BLUE
Receive	RRB	BRB	RRB	RBB
Vote 1	RED	BLUE	RED	BLUE
Receive	BRB	BBR	RRB	RBR
Vote 2	BLUE	BLUE	RED	RED
Receive	BRB	RBB	RRB	BBR
Vote 3	BLUE	BLUE	RED	BLUE
Receive	BBR	**BBB**	RBB	**BBB**
Vote 4	BLUE	BLUE	BLUE	BLUE
Receive	**BBB**	BBB	**BBB**	BBB

Example Run with $f = 1$

	Process 1	Process 2	Process 3	Process 4
Vote 0	RED	RED	BLUE	BLUE
Receive	RRB	BRB	RRB	RBB
Vote 1	RED	BLUE	RED	BLUE
Receive	BRB	BBR	RRB	RBR
Vote 2	BLUE	BLUE	RED	RED
Receive	BRB	RBB	RRB	BBR
Vote 3	BLUE	BLUE	RED	BLUE
Receive	BBR	**BBB**	RBB	**BBB**
Vote 4	BLUE	BLUE	BLUE	BLUE
Receive	**BBB**	BBB	**BBB**	BBB

univalence

Validity?

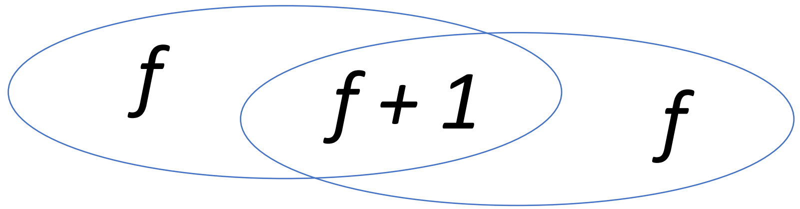
Obvious:

- no proposals invented by the protocol
- processes always vote for one of the original proposals

Agreement?

By contradiction:

- two processes deciding *e* and *e'* in the same round?
  - can't happen because they each need $2f + 1$ votes for their proposal, and there are only $3f + 1$ processes
- two proc's deciding *e* in round *r* and *e'* in round *r'*?
  - can't happen: if a process decides *e* in round *r*, then $2f+1$ processes must have voted for *e*. Thus, any correct process must have received at least $f + 1$ votes for *e* in the same round and change its estimate to *e*. Hence, starting in round $r + 1$, all votes will be for *e* and no other value can be decided.



Termination?

This protocol doesn't guarantee it

- Suppose f = 1, and thus there are four processes
- In round 0, two processes propose RED and two processes propose BLUE.
- In round 1
  - two processes receive two RED and one BLUE vote and set their estimate to RED
  - the other two processes receive one RED and two BLUE votes and set their estimate to BLUE
- *Status quo maintained*…
  - this scenario can be repeated indefinitely

Meeting the 2*f*+1 lower bound

- The trick is to create a protocol that guarantees that *if* two processes vote in the same round, they vote for the same proposal
- One instantiation of this trick is to assign to each round a “leader”
  - for example, the leader role could rotate among the processes from round to round
- Processes are allowed to abstain from voting, for example if they don’t hear from the leader within a reasonable amount of time

$2f + 1$ consensus protocols

- Two *phases* to a round:
  1. Determine a single proposal to vote on
     - For example, by leader or majority
     - This may fail
  2. Vote on the proposal if there is one
     - Protocol decides if majority votes (for the proposal)
     - Processes may abstain, so no guarantee that a decision is made

What is Paxos?

- Paxos is a state machine replication protocol for asynchronous environments with crash failures [Leslie Lamport, 1989].
- It uses a consensus protocol called “Synod” that meets the $2f$+1 lower bound
- “ballots” similar to rounds
  - but uses “leader” to select a single value in phase 1 rather than majority vote---*reduces contention*
  - requires a timer to time-out on slow or faulty leaders
  - *same ballot can be re-used to make multiple decisions*

Protocol with $3f + 1$ processes

1. Broadcast $< r, e >$ "vote" (including to self)
2. Wait for $2f + 1$ votes (out of $3f + 1$)
   - *Note*: because as many as f may fail, this is the maximum a process can safely wait for
3. If a majority of the $2f + 1$ votes contains the same proposal, change e to that proposal
   - *Note*: because $2f + 1$ is odd, there cannot be a tie
4. If not, set e to a proposal in any of the votes received
5. If all votes contain the same proposal (unanimity), decide that proposal
6. $r := r + 1$
7. Repeat (go to Step 1, starting next round)

Byzantine Protocol with $5f + 1$ processes

1. Broadcast < *r*, *e* > "vote" (including to self)
2. Wait for $4f + 1$ votes (out of $5f + 1$)
   - *Note*: because as many as f may fail, this is the maximum a process can safely wait for
3. If a majority of the $4f + 1$ votes contains the same proposal, change e to that proposal
   - *Note*: because $4f + 1$ is odd, there cannot be a tie
4. If not, set e to a proposal in any of the votes received
5. If all votes contain the same proposal (unanimity), decide that proposal
6. $r := r + 1$
7. Repeat (go to Step 1, starting next round)

Example Run with $f = 1$

	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
Vote 0	RED	RED	BLUE	BLUE	BLUE	RED/BLUE
Receive	RRRBB	BRBBB	RRRBB	RRBBB	RRRBB	
Vote 1	RED	BLUE	RED	BLUE	RED	RED/BLUE
Receive	BRRBB	BBRRB	RRRRB	RBRRR	RRRBB	
Vote 2	BLUE	BLUE	RED	RED	RED	RED/BLUE
Receive	BBRRB	RRBBB	BRRBB	BBBRR	RRRBB	
Vote 3	BLUE	BLUE	BLUE	BLUE	BLUE	RED/BLUE
Receive	BBBBR	$\boxed{\textbf{BBBBB}}$	RBBBB	$\boxed{\textbf{BBBBB}}$	BBRBB	
Vote 4	BLUE	BLUE	BLUE	BLUE	BLUE	RED/BLUE
Receive	$\boxed{\textbf{BBBBB}}$	BBBRB	$\boxed{\textbf{BBBBB}}$	BRBBB	BBRBB	