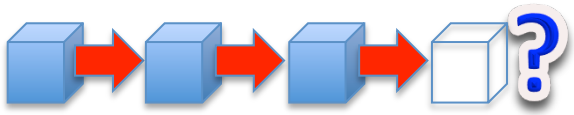
Federated Consensus

Robbert van Renesse
joint work with Isaac Sheff and Andrew Myers
Cornell University

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*
    - *Dfinity/Pebble, HyperLedger, Ripple, Stellar*
    - *Alternative to Proof of Work / Proof of **

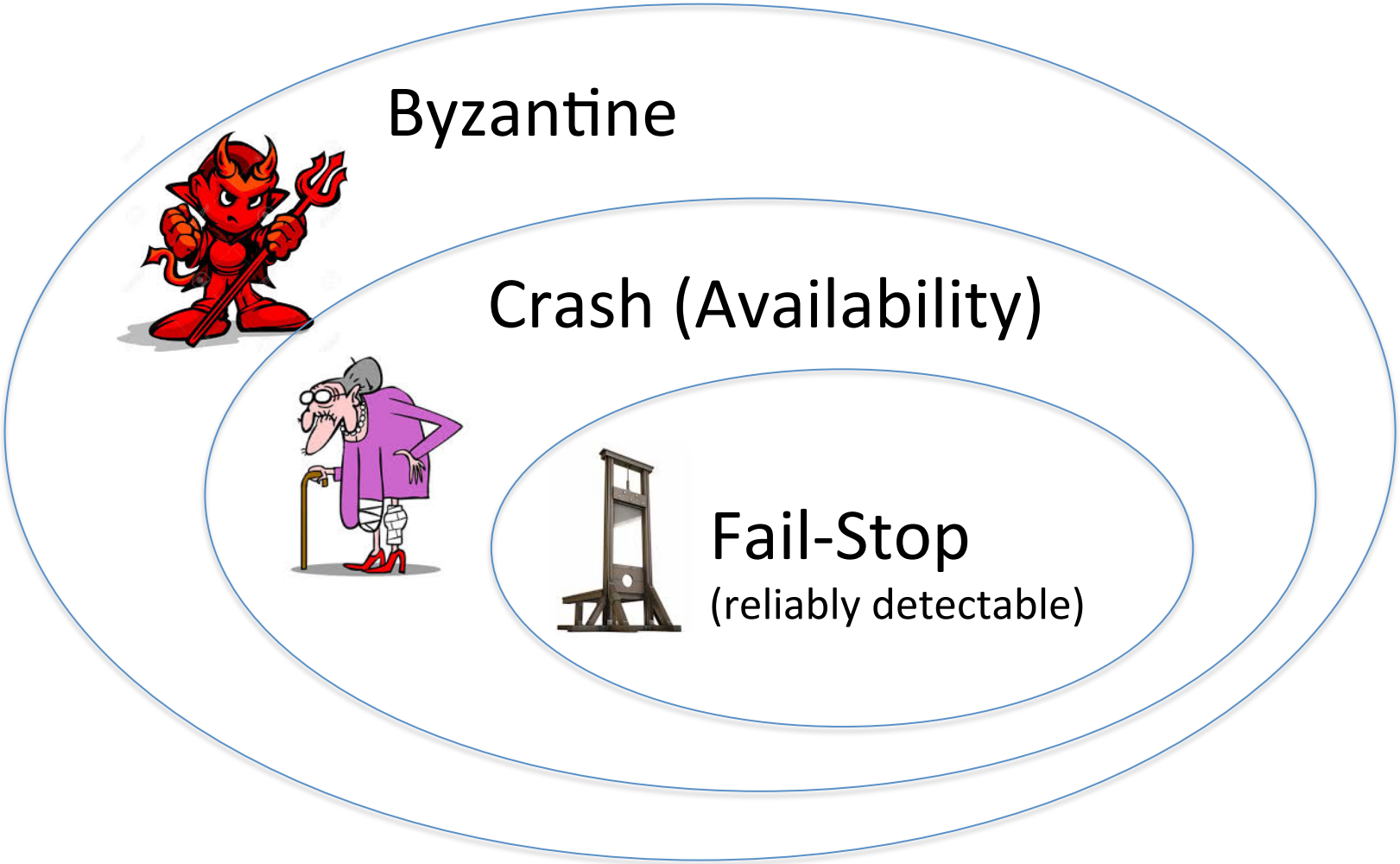


- Surprisingly hard with participant and network failures
  - whether accidental or malicious
- Even harder in the face of *asynchrony*
  - complete lack of bounds on latency

Consensus Formalized

- Agreement:
  - if multiple correct participants decide, they must decide the same value
- Validity:
  - if all correct participants propose the same value, then that value must be decided
    - other non-triviality conditions are possible
- Termination:
  - a correct participant must eventually decide

Failure Type Hierarchy



Integrity Failure = Byzantine – Availability Failure

Generic Asynchronous Consensus Protocol (Simplified)

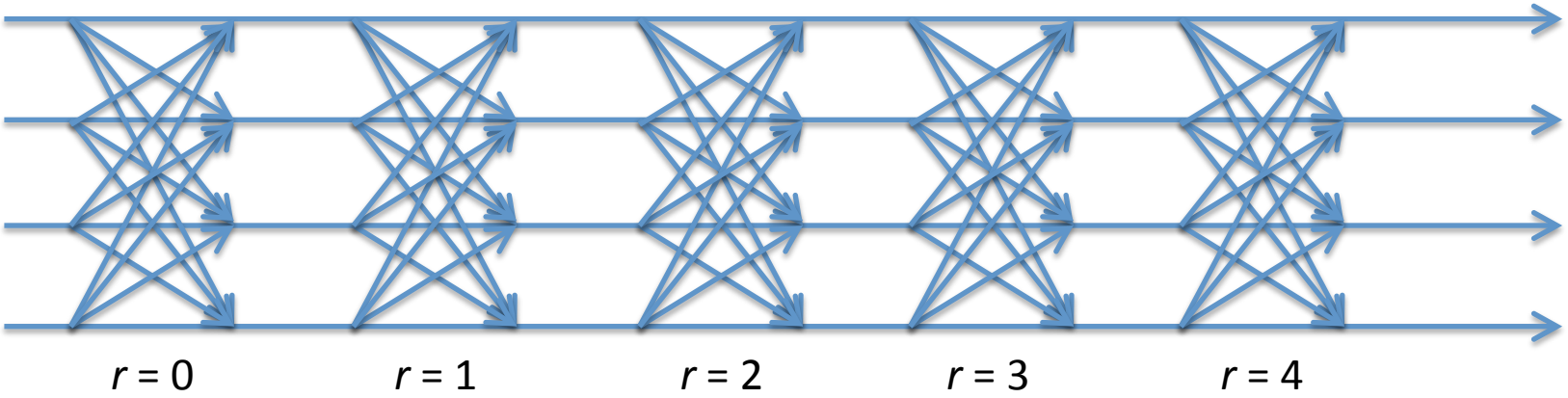
- The participants run *rounds* of communication
  - aka *ballots*
- Each participant maintains
  - a round number *r,* initially 0
  - an estimate *e,* initially the proposal of the participant
- Communication channels between participants are point-to-point and *private*
  - no public key crypto required

Based on Yee Jiun Song and Robbert van Renesse. Bosco: One-Step Byzantine Asynchronous Consensus. 22nd International Symposium on Distributed Computing (DISC 08). September 2008

Generic Asynchronous Consensus

(for crash or Byzantine failures, or combination)

1. Broadcast < *r*, *e* > "vote" to N participants (including to self)
2. Wait for T1 ≤ N votes
3. If T2 ≤ T1 votes are unanimous, decide that proposal
4. If T3 ≤ T2 votes are unanimous, change *e* to that proposal
5. *r* := *r* + 1
6. Repeat (go to Step 1, starting next round)



Thresholds

	N (total)	T1 (wait)	T2 (decide)	T3 (change)
Crash	$3f+1$	$2f+1$	$2f+1$	$f+1$
Byzantine	$5f+1$	$4f+1$	$4f+1$	$2f+1$

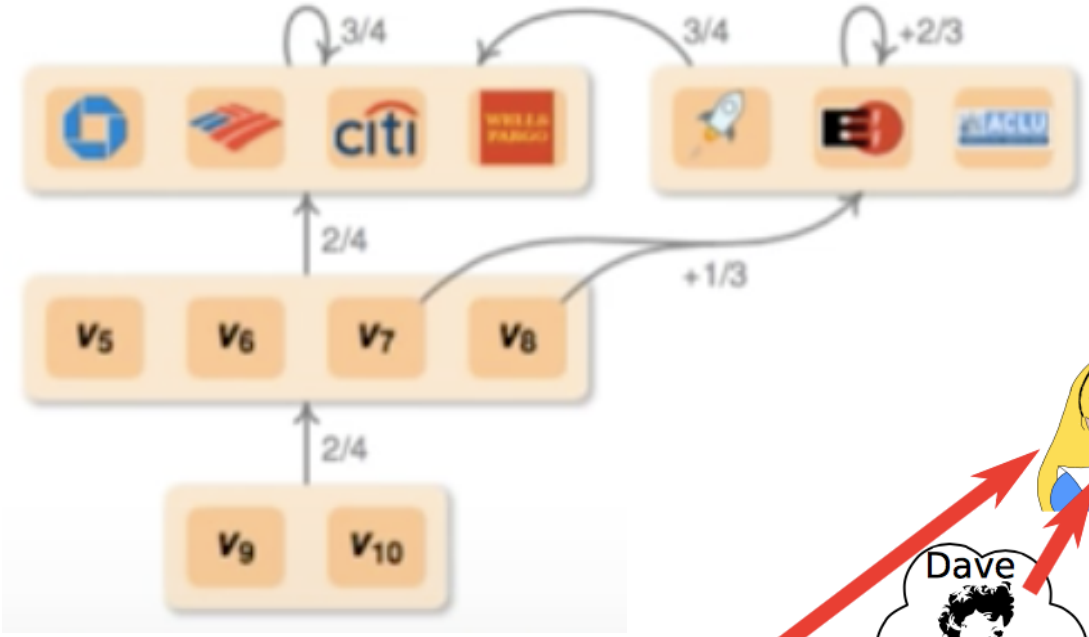
Notes:

- N does not meet lower bounds
  - meeting lower bounds requires two voting phases per round
- T1 = N – f
- T2 + T2 > N (*quorums intersect*)
- T3 + T3 > T1 (T3 is a majority of received votes)
- Everybody agrees *a priori* on N, T1, T2, and T3

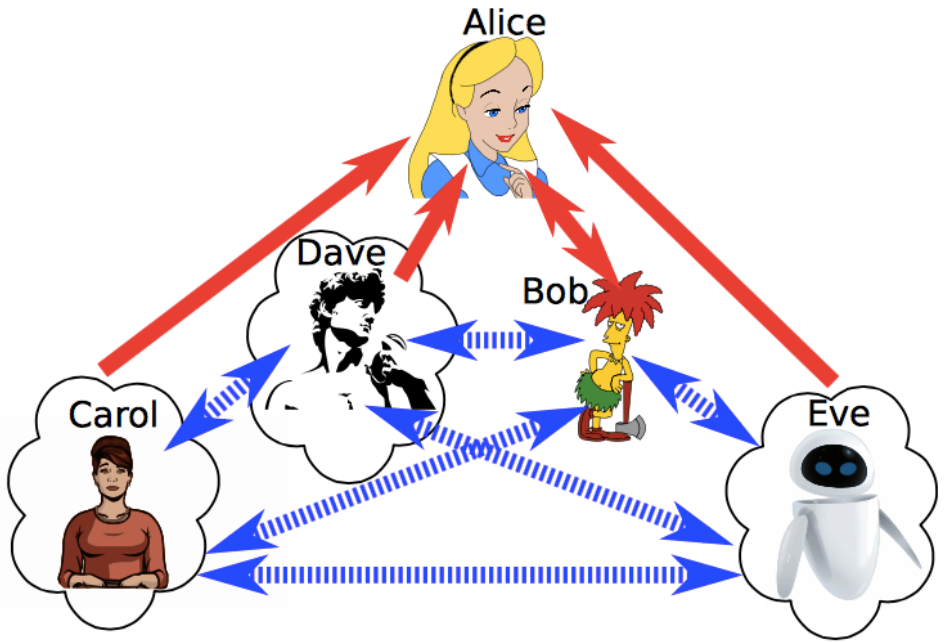
Trust in Practice?

- Every principal (participant) has potentially *different trust* in other principals
- Can be further refined with respect to
  - Trust in Integrity
  - Trust in Availability
  - (Trust in Confidentiality)
- Not *homogeneous*, but *heterogeneous*

Heterogeneous Trust



from Stellar YouTube presentation
(David Mazières)



from "*Distributed Protocols and Heterogeneous Trust*" by Sheff et al. 2014
distinguishes integrity and availability trust assumptions

Quorum Slices

- Each principal trusts one or more groups of principals. Stellar calls each such group a *slice*.
- E.g., if some principal trust 3 out of { a, b, c, d }, then there are four slices
- We would express this as follows

$$(a \wedge b \wedge c) \vee (a \wedge b \wedge d) \vee (a \wedge c \wedge d) \vee (b \wedge c \wedge d)$$

Revisit Consensus Thresholds

- *Each threshold (T1, T2, T3) is a set of slices*
- Need to determine for each participant:
  - T1: when to stop waiting for votes
    - i.e., it has votes from all participants in a T1 slice
  - T2: when to decide
    - i.e, it has unanimous votes from all participants in a T2 slice
  - T3: when to change estimate
    - i.e., it has unanimous votes from all participants in a T3 slice

Not all honest participants are the same

An honest participant may accidentally choose

- T1 slices too large: get stuck waiting for votes
  - similar to crash failure
- T2 or T3 slices too large: never decide or never change estimates
- T2 slices too small: decide too quickly and possibly inconsistently with other honest participants
  - similar to Byzantine failure
- T3 slices too small: change estimates too quickly and possibly confuse other honest participants

Revisit Consensus Properties

- Classes of honest participants:
  - Chumps: trust assumptions violated
  - Gurus: trust assumptions always hold
- Stellar concepts and terminology:
  - intact: non-crashing guru
  - befouled: not intact
- Heterogeneous Consensus Properties:
  - if two *gurus* decide, they decide the same value
  - gurus never get stuck
    - can always potentially decide and terminate
  - hard to provide properties to chumps
    - what you trust can hurt you!

How to select thresholds/slices?

Very large space of potential solutions to explore

- Our approach (2014):
  - use an SMT solver to determine T1, T2, and T3
  - requires knowing participants and their trust assumptions
  - does not scale well
- Stellar's approach (2015):
  - given slices, construct *quorums*:
    - a quorum is a set of principals such that each of its members has a slice that is contained within that quorum
  - then make it the *responsibility* of each participant to choose slices such that any two quorums intersect *after removing Byzantine participants from the quorums*

How to select slices in Stellar

systems thanks to the duplicity of the ill-behaved nodes. In short, FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$ can survive Byzantine failure by a set of nodes $B \subseteq \mathbf{V}$ iff $\langle \mathbf{V}, \mathbf{Q} \rangle$ enjoys quorum intersection after deleting the nodes in B from $\mathbf{V}$ and from all slices in $\mathbf{Q}$. More formally:

Definition (*delete*). If $\langle \mathbf{V}, \mathbf{Q} \rangle$ is an FBAS and $B \subseteq \mathbf{V}$ is a set of nodes, then to *delete* B from $\langle \mathbf{V}, \mathbf{Q} \rangle$, written $\langle \mathbf{V}, \mathbf{Q} \rangle^B$, means to compute the modified FBAS $\langle \mathbf{V} \setminus B, \mathbf{Q}^B \rangle$ where $\mathbf{Q}^B(v) = \{ q \setminus B \mid q \in \mathbf{Q}(v) \}$.

It is the responsibility of each node v to ensure $\mathbf{Q}(v)$ does not violate quorum intersection. One way to do so is to pick conservative slices that lead to large quorums. Of course, a malicious v may intentionally pick $\mathbf{Q}(v)$ to violate quorum intersection. But

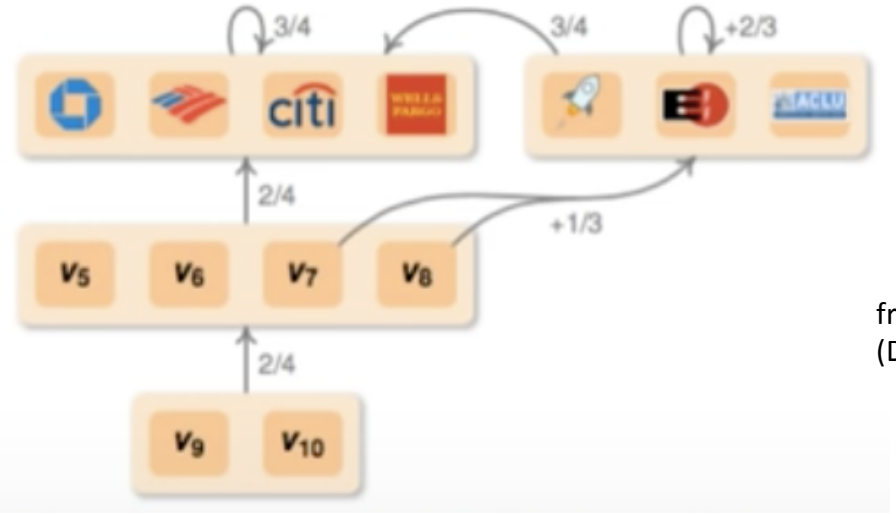
— **Decentralized control.** Anyone is able to participate and no central authority dictates whose approval is required for consensus.

from: David Mazières. *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus (April 2015)*

Potential Problems with Stellar's Slice Selection

- If you know the Byzantine participants or a superset thereof, there is a trivial solution
- To ensure quorums intersect, you need to know the slices of the members in the quorums
  - *Not so open…*
- To know if a quorum is "*large,*" you need to know how many participants there are in total
- Large quorums are good for safety but bad for liveness
  - It had better be that most participants in slices are active, or there will be no quorum consisting of intact participants and hence no decision

Tiered Transitive Trust



from Stellar YouTube presentation
(David Mazières)

- Improves chances of any two quorums intersecting without having to know the entire membership
- Helps with so-called *Sybil attacks* (a single attacker pretending to be multiple principals)
- Vulnerable to failures in Tier 1 participants
  - *Guess*: 3/4 in Tier 1 (sink component) not chosen arbitrarily (3*f*+1)
  - why not just delegate/centralize? *(add auditing)*

Conclusion

- It is not yet clear (to me) that successful *open* Byzantine consensus protocols can be built on trust between principals or federations of principals
  - but see "*Byzantine Consensus with Unknown Participants.*" Edward A. Alchieri et al. In 12th International Conference on Principles of Distributed Systems. 2008.
- Did not discuss another design option for federation
  - replicate the participants of a consensus protocol
  - leads to a hierarchy of replicated state machines
    - see "*STEWARD: Scaling Byzantine Fault-Tolerant Replication to Wide Area Networks.*" Yair Amir et al. IEEE Transactions on Dependable and Secure Computing 7(1). 2010.

backup slides

What is a Crash Failure?

- aka Availability Failure
- A participant that stops *indefinitely*
  - But it follows specification until then
- Crash failures cannot be reliably detected in an asynchronous environment
  - If you ping a participant and don't receive a response, you don't know if the participant is faulty or the system is simply slow

What is Asynchrony?

- No bounds on timing
  - no bounds on message latency
    - but between correct processes messages are eventually delivered
  - no bounds on how fast clocks run
    - but they do run monotonically increasing
  - no bounds on how skewed the clocks are
    - clocks on different machines show arbitrarily different times
  - no bounds on processing time
- Not to be confused with “non-blocking”
  - “asynchronous RPC” and “asynchronous system calls” are misnomers

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

($f = 3$)