Byzantine Agreement with Less Communication

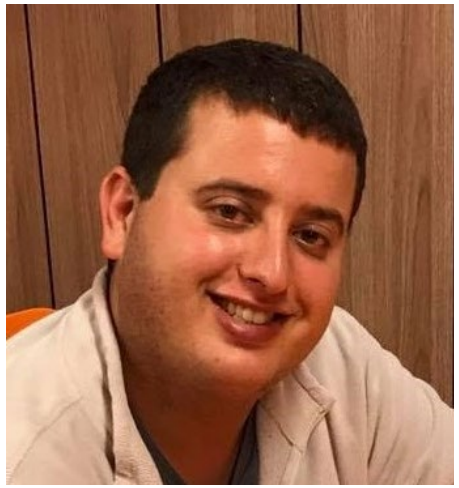
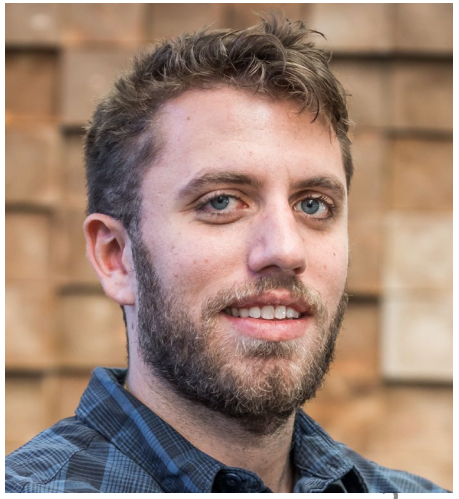
Idit Keidar, Technion

Shout Out

Not a COINcidence: Sub-Quadratic Asynchronous Byzantine Agreement WHP.
Shir Cohen, Idit Keidar, and Alexander Spiegelman

All You Need is DAG.
Idit Keidar, Oded Naor, Lefteris Kokoris-Kogias, and Alexander Spiegelman



Byzantine Agreement (BA)

- Consensus among *n* processes
- Up to *f* can be controlled by an adversary and act arbitrarily

- Byzantine Atomic Broadcast (BAB)
  - Agree on a sequence of messages
  - Reliable Broadcast + Total Order

- A building block for State Machine Replication (SMR)

New Frontiers for BA & BAB

- Permissioned blockchains – shared ledger
- Other FinTech infrastructures

Quorum

diem

HYPERLEDGER

MultiChain

ethereum

r3.corda

ORBS

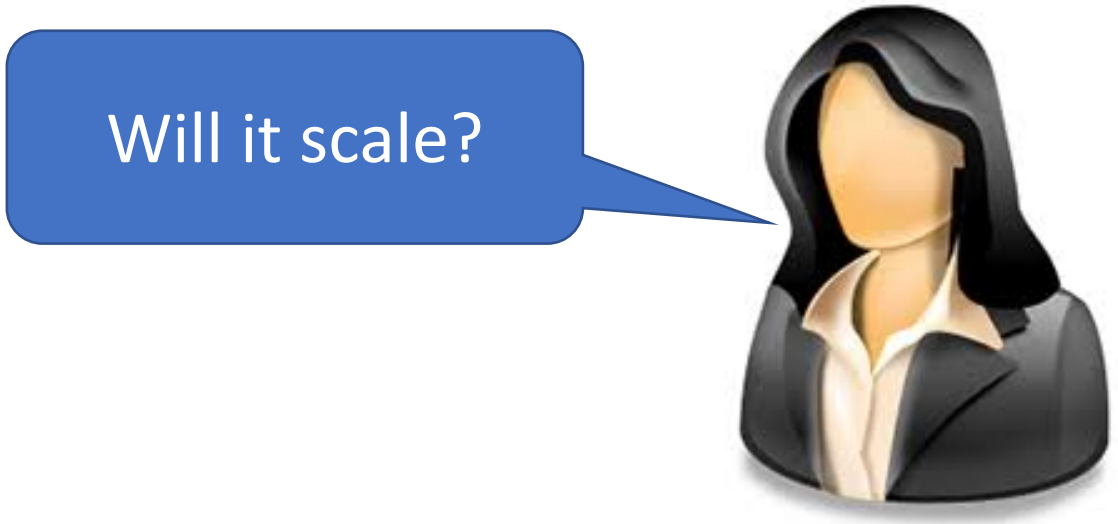
parity

Algorand™

BA Has Been Around for Four Decades

[Pease, Shostak, Lamport 1980], [Lamport, Pease, Shostak 1980]

- 2500+, 7000+ citations, resp.
- Traditional use-cases – a handful of processes



Traditional BFT According to James Mickens

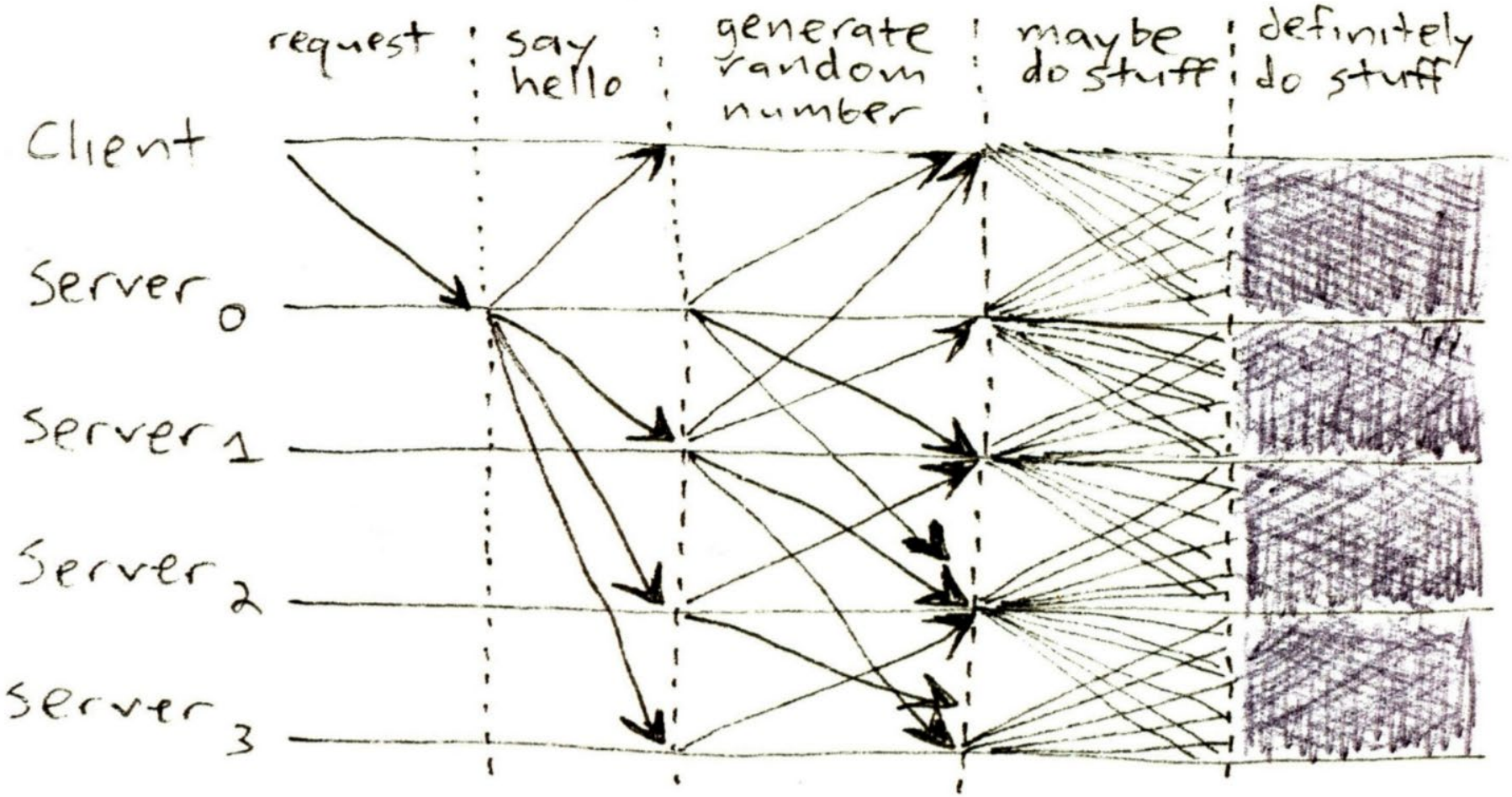


Figure 1: Typical Figure 2 from Byzantine fault paper: Our network protocol

Scalability Challenges

- Communication (word) complexity (of all processes together)
  - $\Omega(n^2)$ lower bound
    In the worst-case, in deterministic algorithms, regardless of synchrony
    [Dolev and Reischuk 1985]
  - Randomization can help

- Synchrony vs. asynchrony
  - Synchrony is not robust, latency bounds defined in minutes

  $\Rightarrow$ Consider asynchrony
  - Randomization required [Fisher, Lynch, Paterson 1985]

Making It Scale

VRFs

- Solve BA
  with high probability (WHP)
  (probability of being correct tends to 1 as $n \to \infty$)
- Complexity: $\widetilde{O}(n)$

Perfect coin

- Solve BAB
  with deterministic safety,
  probabilistic liveness
- *Amortized* complexity:
  O(n) or *O(n log n)* per agreement

Not a COINcidence: Sub-Quadratic Asynchronous Byzantine Agreement WHP

Shir Cohen, Idit Keidar, Alexander Spiegelman

Technion *Novi Research*

DISC 2020

Contribution

The first sub-quadratic asynchronous BA WHP algorithm

- $\tilde{O}(n)$ word complexity and $O(1)$ expected time
- Safety and Liveness properties are gurenteed WHP
- Binary BA

- Previous sub-quadratic works made synchrony assumptions

[King and Saia 2011], Algorand [Gilad et al. 2017]

Model

- Asynchronous
- n processes (permissioned)
- Up to f Byzantine processess for $n \approx 4.5f$
- Trusted PKI
  - Inherent for sub-quadratic algorithms
    [Abraham et al. 2019] [Blum et al. 2020] [Rambaud 2020]
- Delayed adaptive adversary:
  - Can use the contents of a message m sent by a correct process for scheduling a message m' only if $m \rightarrow m'$

Verifiable Random Function (VRF)

- A pseudorandom function that provides a proof of its correct computation
- For a secret key sk with a matching public key pk
  - $VRF_{sk}(x)$ is a random value
  - Verifiable using pk

random number + proof

Use VRFs for

1. Flipping an (imperfect) shared coin
   - First step: $O(n^2)$ word complexity
2. Committee sampling
   - Cryptographic sortition
   - Reduces word complexity to $O(n \log n)$

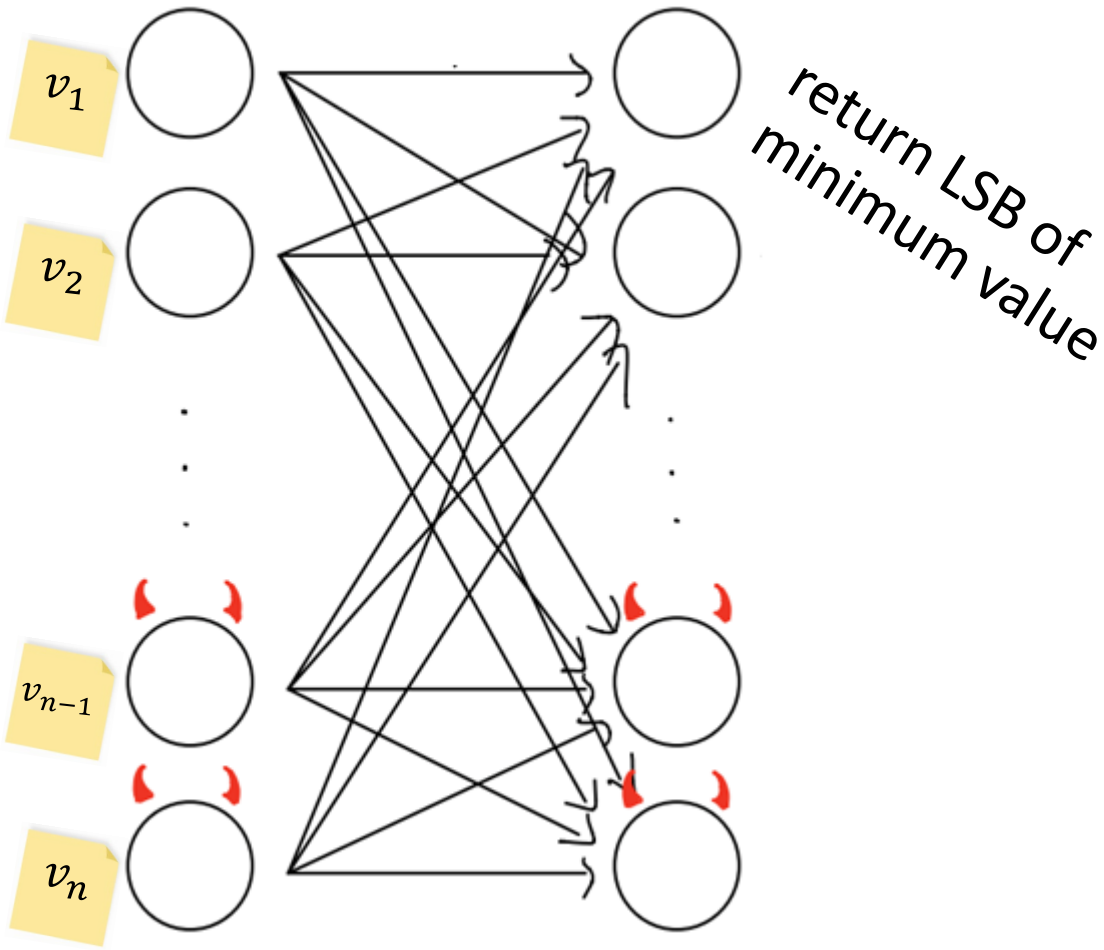
Following Algorand [Gilad et al. 2017]

Shared Coin with Success Rate ρ

All correct processes output b with probability at least ρ, for any value $b \in \{0,1\}$

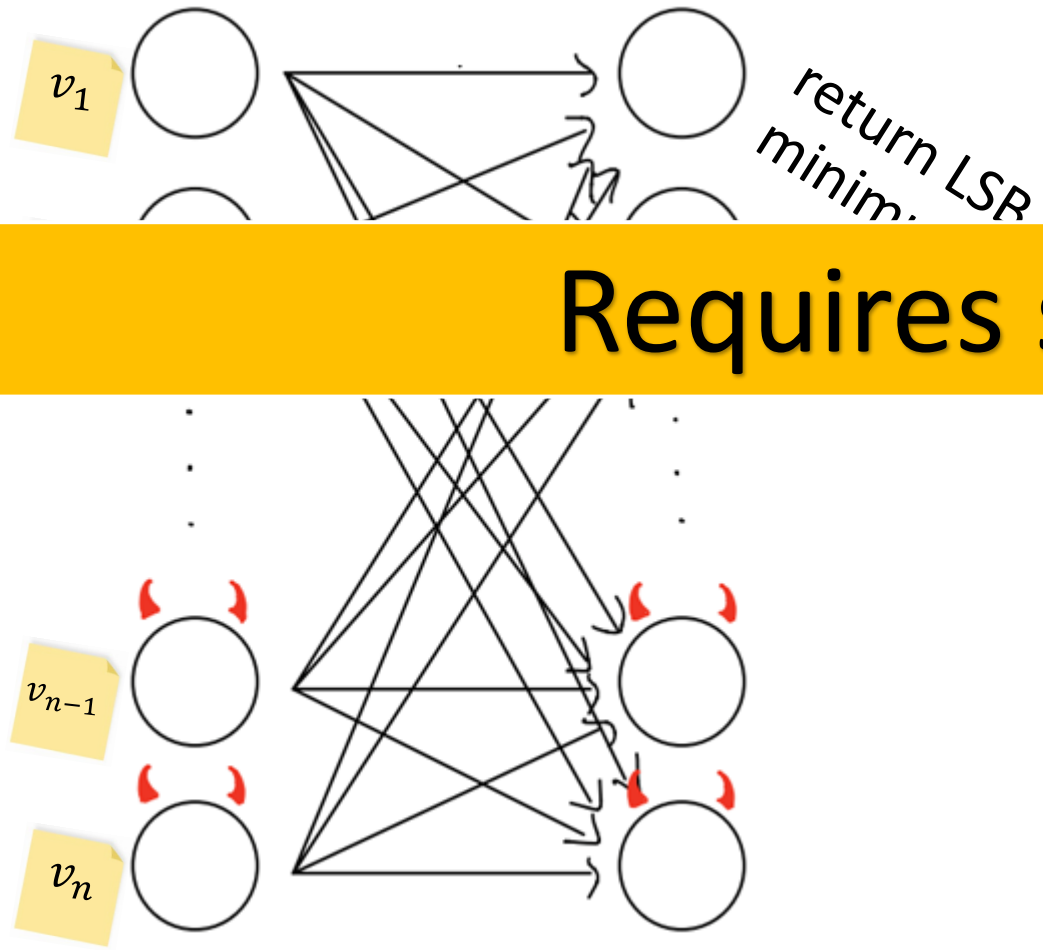
Shared Randomness

Background: A Simple VRF-Based Shared Coin



- Synchronous [Micali 2017]
- If the minimum VRF is of a correct process, all agree
  - With probability $\geq \frac{2}{3}$

Background: A Simple VRF-Based Shared Coin

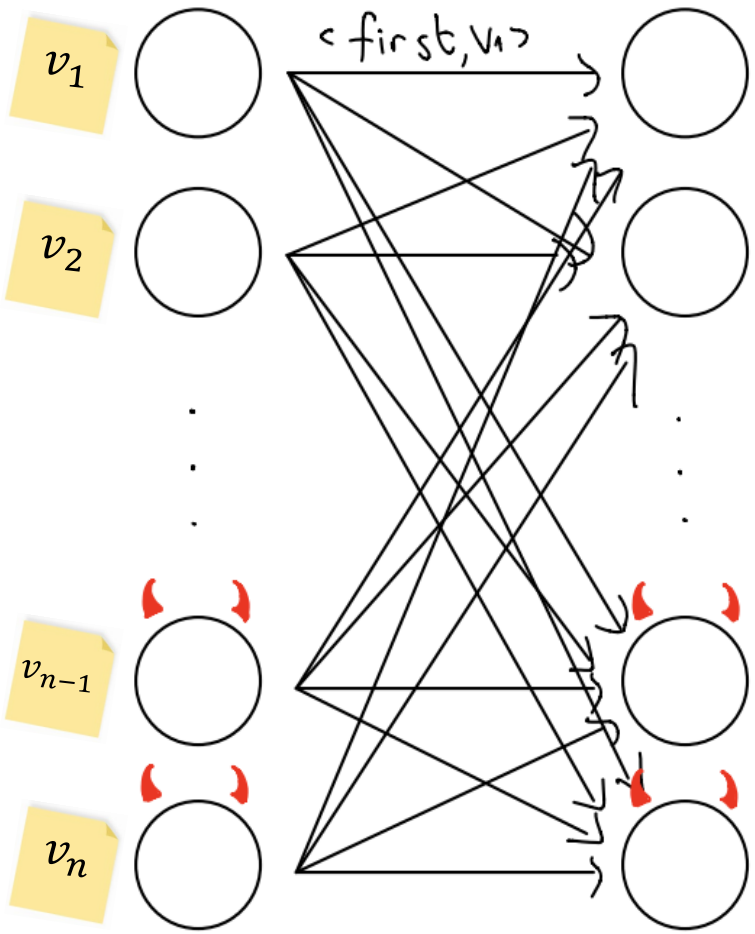


- Synchronous [Micali 2017]
- If the minimum VRF is of a
- With probability $\geq \frac{1}{3}$

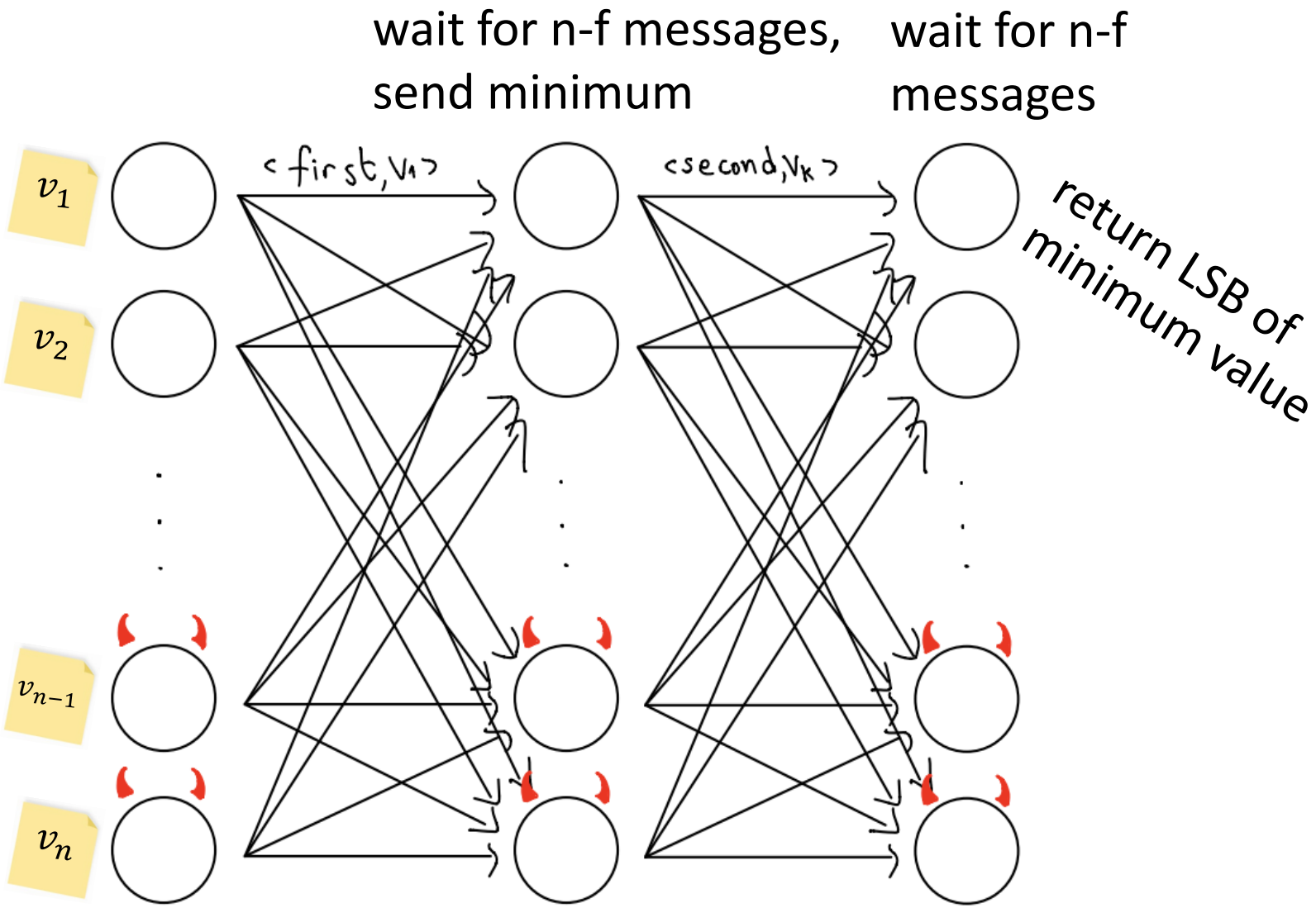
Requires synchrony

Asynchronous Shared Coin – Take 1

wait for n-f messages



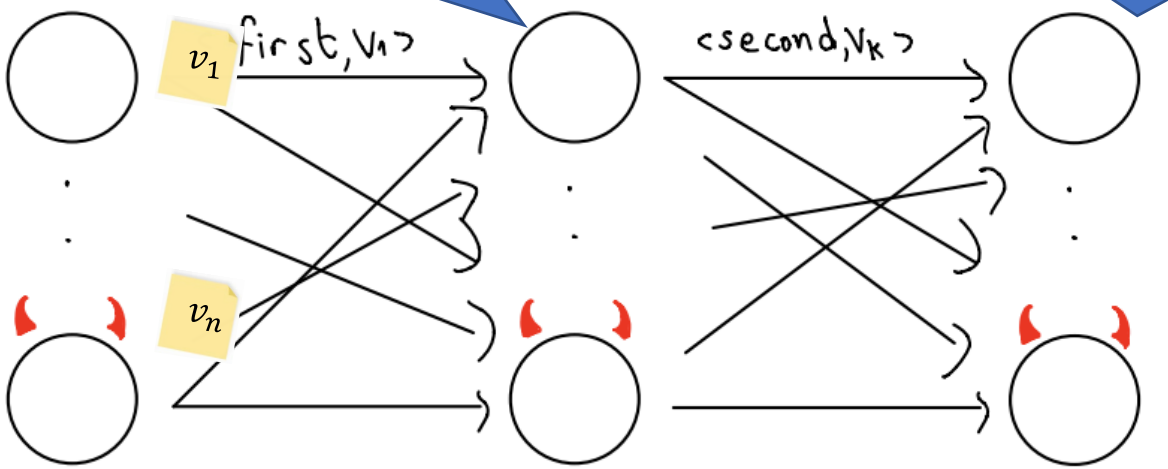
Asynchronous Shared Coin – Take 1



Asynchronous Shared Coin - Analysis

a value that reaches $f+1$ correct processes is *common*

common values reach all correct processes



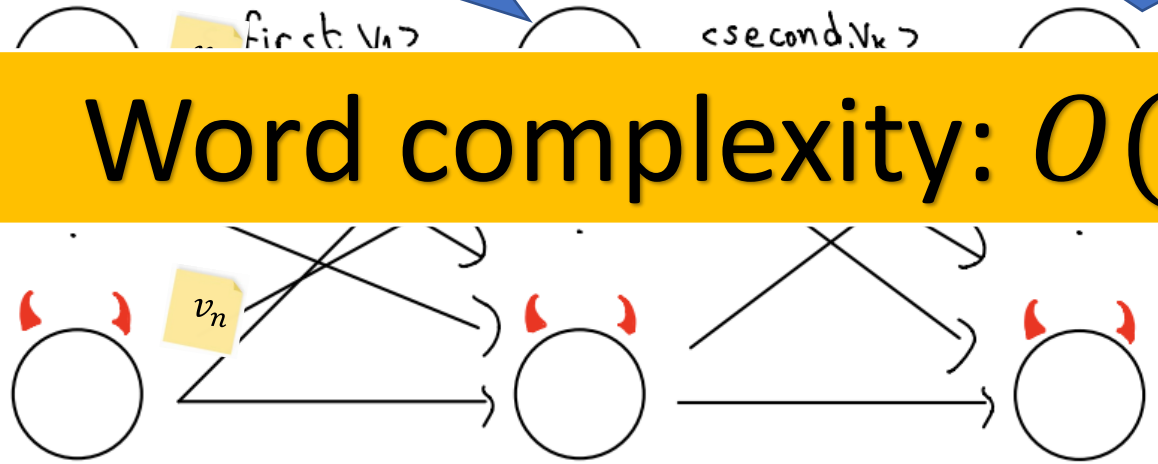
- We prove:
  - $\Omega(\epsilon)$ bound the number of common values
  - our adversary "commits" to them in advance

$\Rightarrow$With a constant probability, the global minimum is common

Asynchronous Shared Coin - Analysis

a value that reaches $f+1$ correct processes is *common*

common values reach all correct processes



Word complexity: $O(n^2)$

- We prove:
  - $\Omega(\epsilon)$ bound the number of common values
  - our adversary "commits" to them in advance

$\Rightarrow$With a constant probability, the global minimum is common

Use VRFs for

1. Flipping a shared coin
   - First step: $O(n^2)$ word complexity
2. Committee sampling
   - Cryptographic sortition
   - Reduces word complexity to $O(n \log n)$

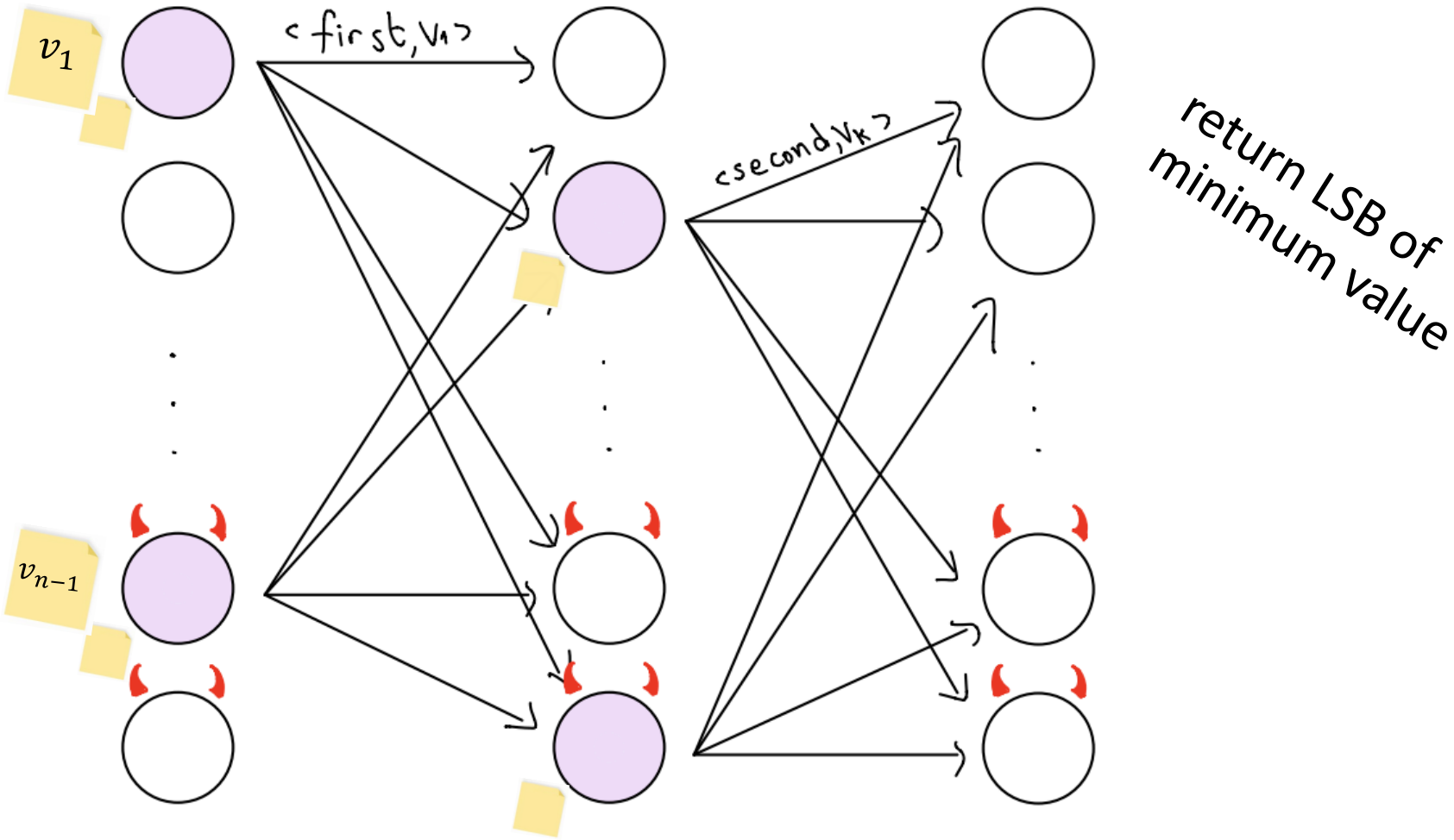
Following Algorand [Gilad et al. 2017]

Committee Sampling

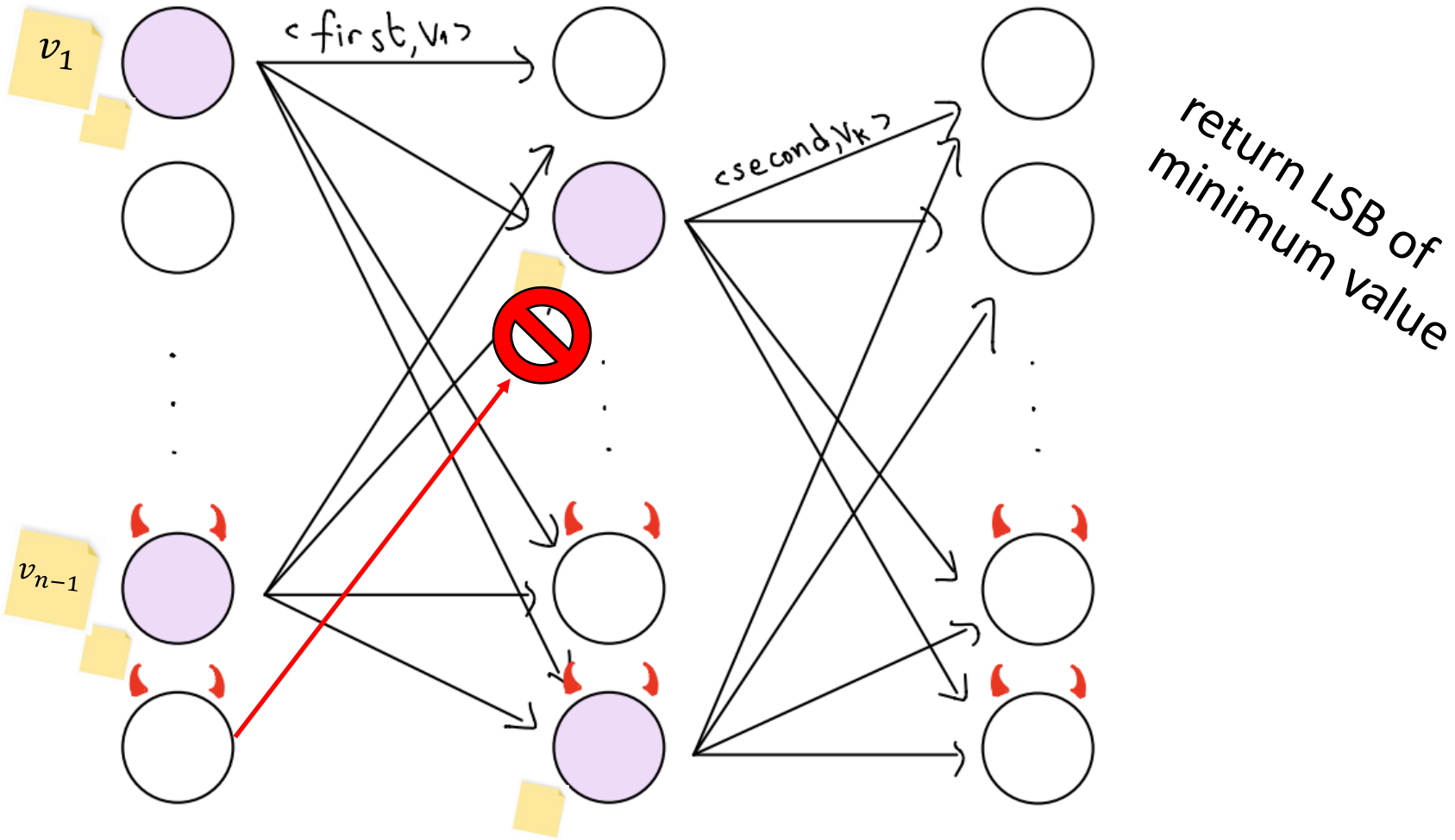
- Use the VRF to sample O(log n) processes to a committee in each round
- Replace all-to-all rounds with committee-to-all rounds

- Evading the adversary:
  - Use a new committee in each round
  - Send to all since committees are unpredictable
  - By Chernoff bounds, “not too many” faulty processes in each committee

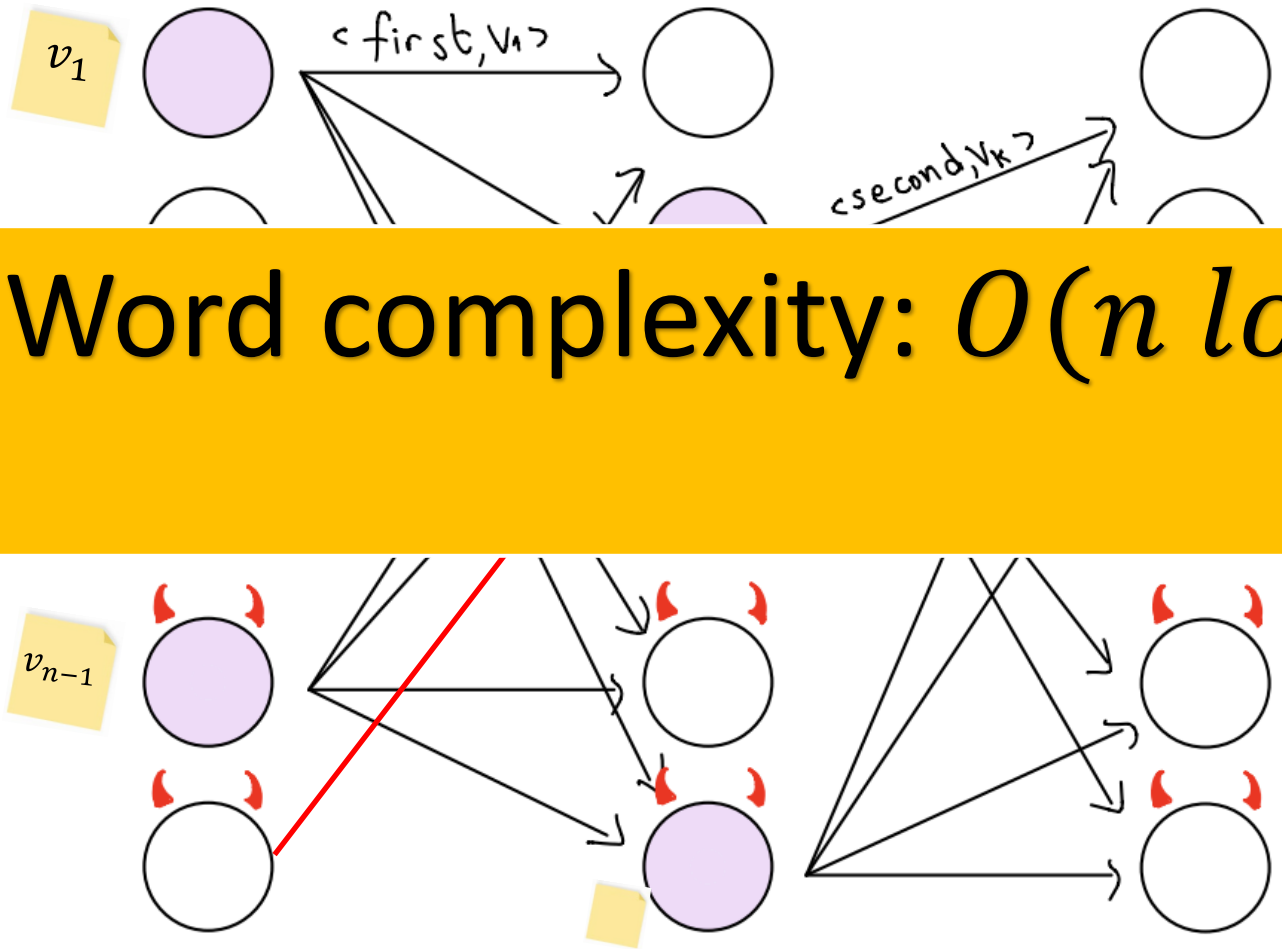
Shared Coin – Take 2



Shared Coin – Take 2

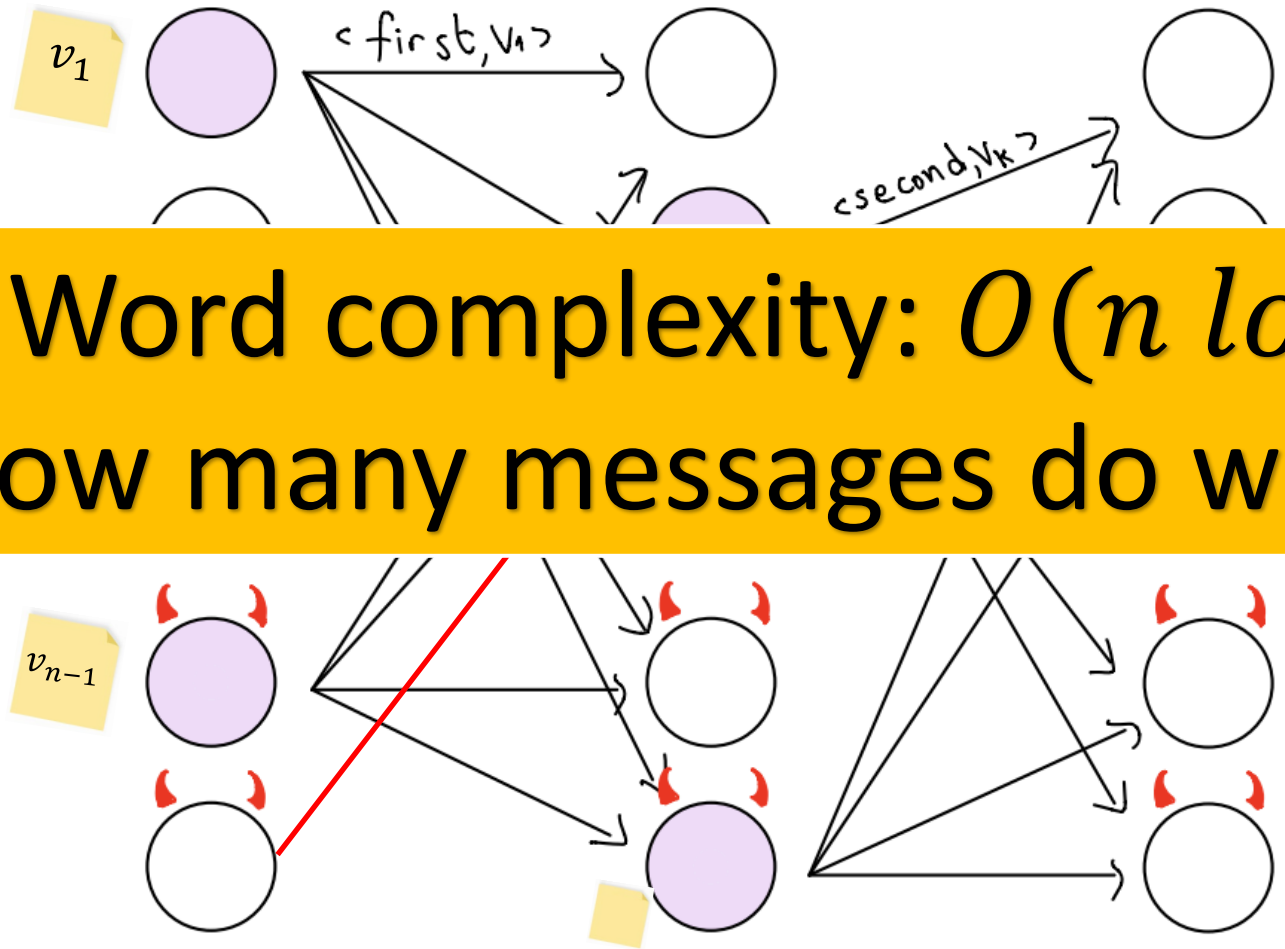


Shared Coin – Take 2



Word complexity: $O(n \log n)$

Shared Coin – Take 2



Word complexity: $O(n \log n)$

But how many messages do we wait for?

Committee Sampling in Asynchronous Model

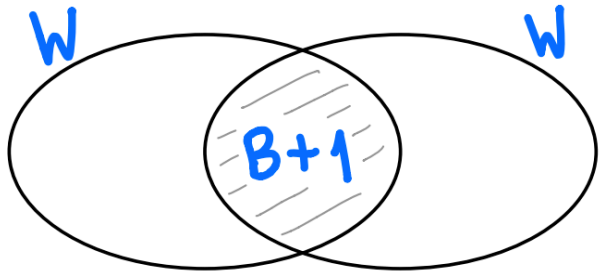
- Committee based protocols cannot wait for $n - f$ processes. Instead, they wait for W processes.

- We choose W, B so that using Chernoff bounds, WHP:

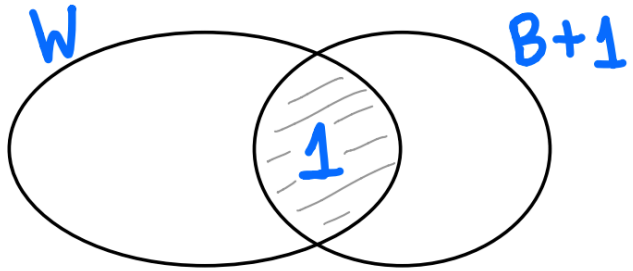
1. At least W processes in each committee are correct
2. At most B processes in in each committee are Byzantine

Committee Sampling in Asynchronous Model

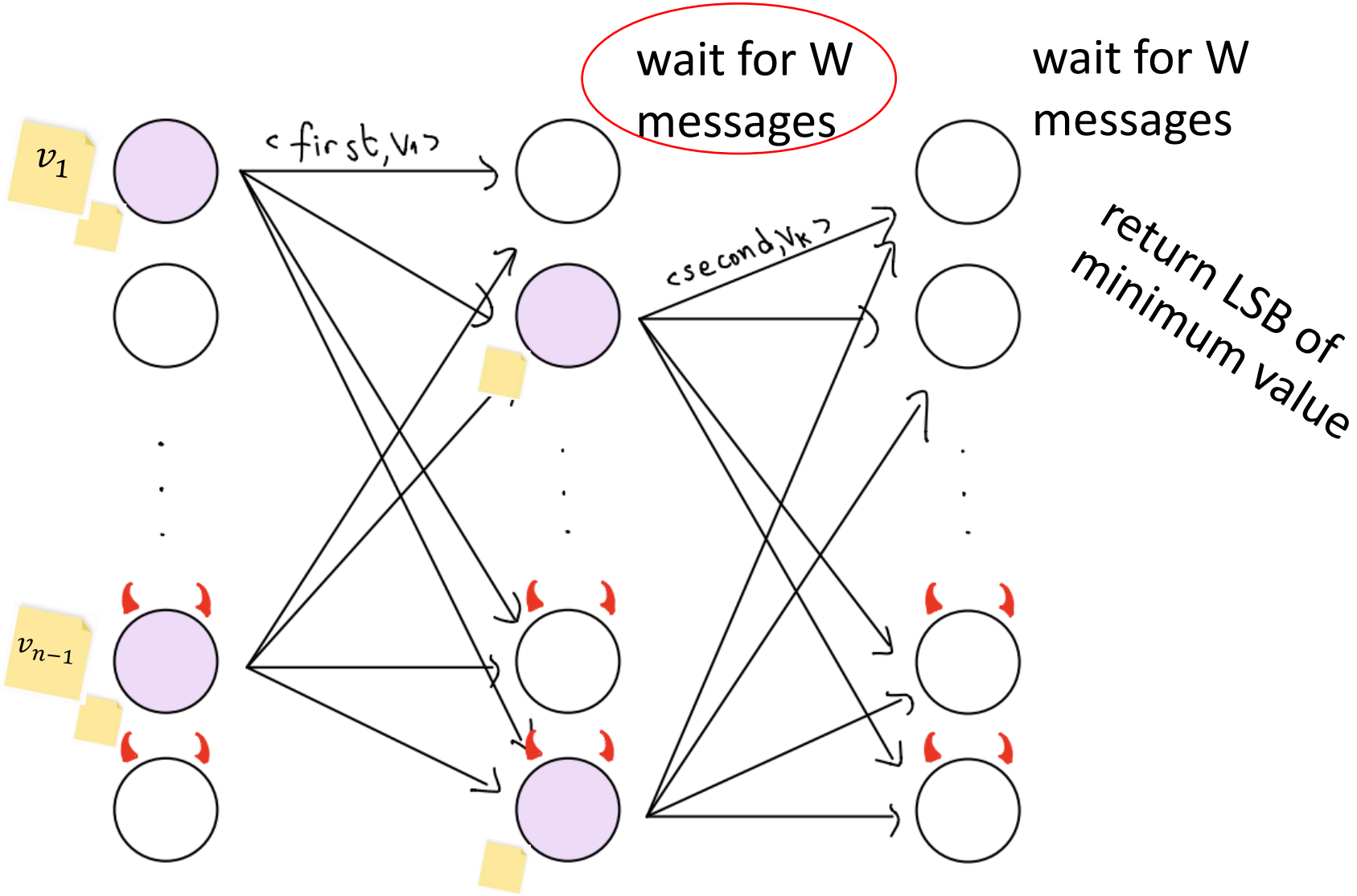
3. Every two subsets in a committee of size W intersect by at least $B + 1$ processes



4. Every two subsets in a committee of size W and $B + 1$ intersect by at least 1 process



Shir Cohen's Shared Coin



From Coin Flipping to (Binary) BA WHP

Approver

+

BA WHP

- Approver based on [Bracha 1987] – reliable broadcast
  - But with committee sampling
- BA based on [Mostefaoui et al. 2015]

Approver

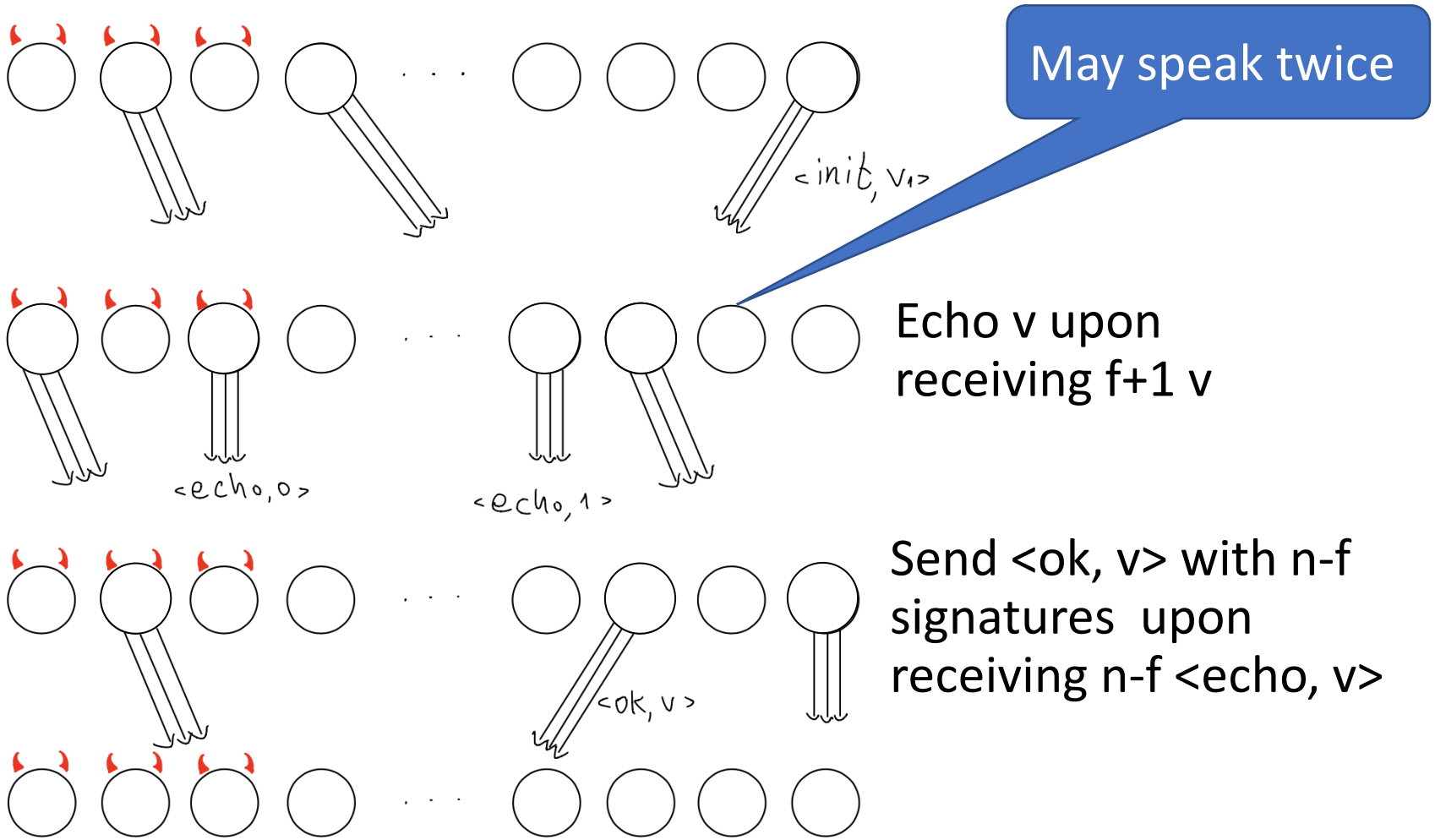
API: *approve*$_i$*(v*$_i$*)* returns a <u>set</u> of values

We assume *approve* is called with at most two different values

WHP the following hold:

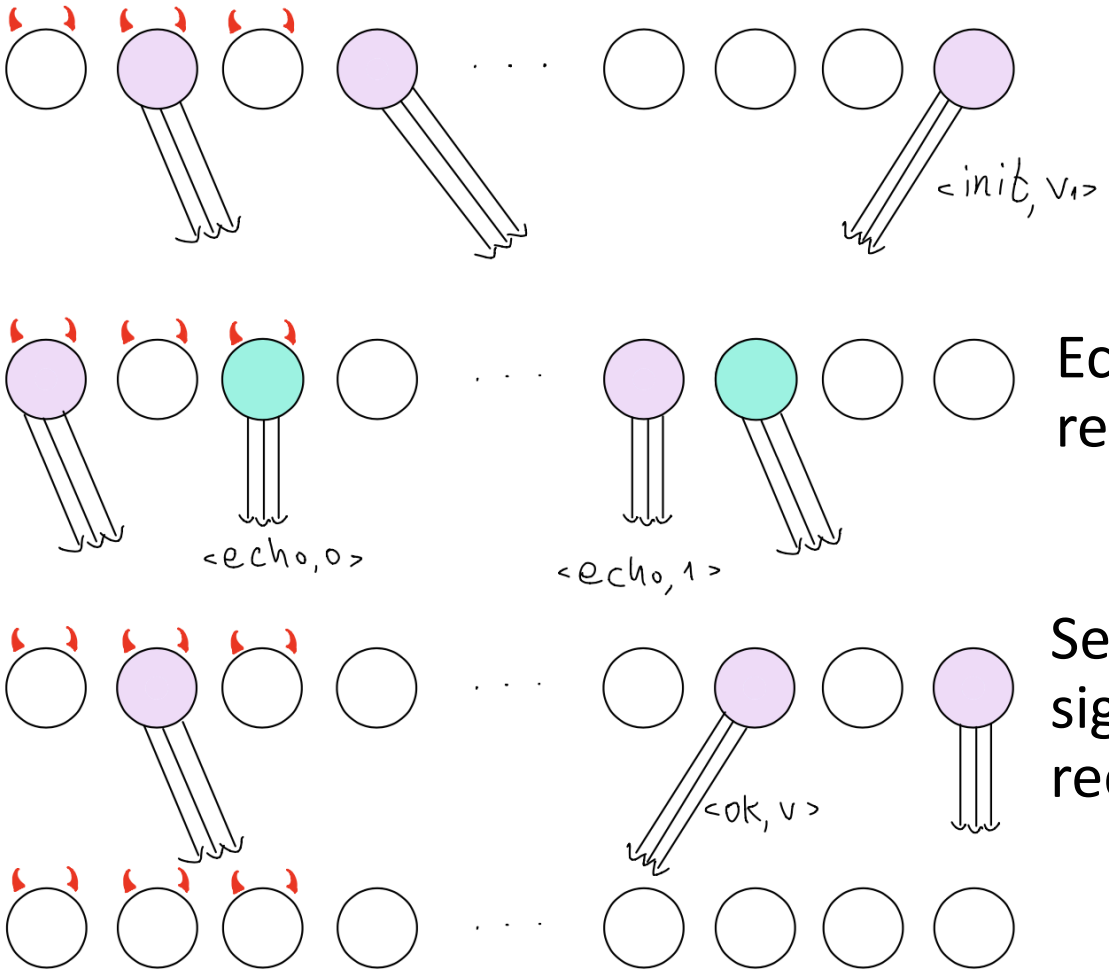
- Validity: If all correct processes invoke $approve(v)$ then the only possible return value of correct processes is $\{v\}$
- Graded agreement: If correct processes return both $\{v\}$ and $\{w\}$ then $v = w$
- Termination: If all correct processes invoke *approve* then it returns with a non-empty set at all of them

Approver 👍 Without Sampling



Return the set of values in the first n-f ok messages

Approver 👍 With Sampling

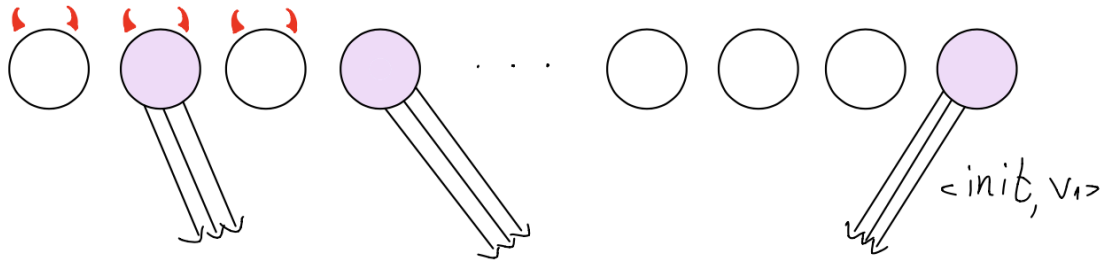


Echo v upon receiving B+1 v

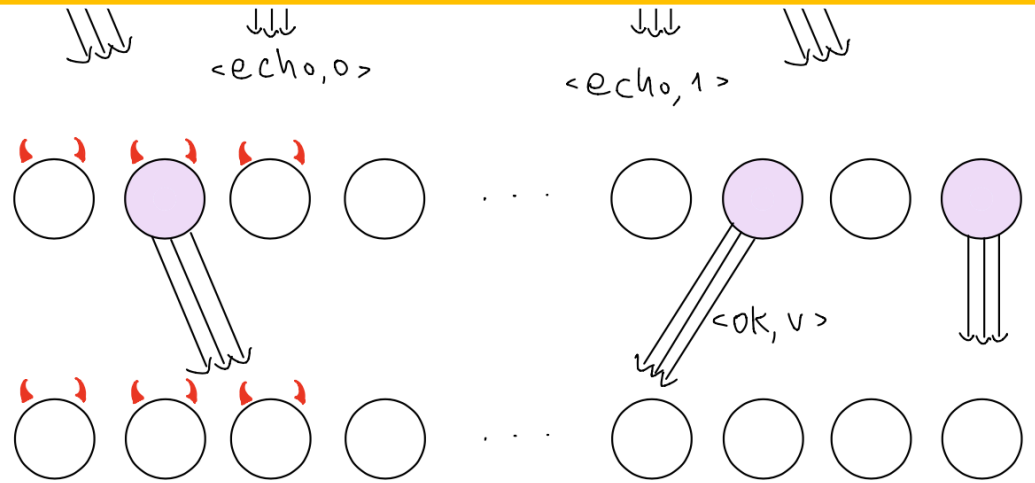
Send <ok, v> with W signatures upon receiving W <echo, v>

Return the set of values in the first W ok messages

Approver 👍 With Sampling



Word complexity: $O(n \log^2 n)$



Send <ok, v> with W signatures upon receiving W <echo, v>

Return the set of values in the first W ok messages

From Coin Flipping to (Binary) BA WHP

Approver

+

→

BA WHP

- Approver based on [Bracha 1987] – reliable broadcast
  - But with committee sampling
- BA based on [Mostefaoui et al. 2015]

BA WHP

```
est_i ← v_i
decision_i ← ⊥

for r = 0, 1, ... do
    vals ← approve(est_i)
    if vals = {v} for some v then
        propose_i ← v
    otherwise propose_i ← ⊥

    c ← whp_coin(r)
    props ← approve(propose_i)
    if props = {v} for some v ≠ ⊥ then
        est_i ← v
        if decision_i = ⊥ then
            decision_i ← v
    else
        if props = {⊥} then
            est_i ← c
        else          %props = {v, ⊥}
            est_i ← v
```

BA WHP

$est_i \leftarrow v_i$
$decision_i \leftarrow \perp$

for $r = 0, 1, \ldots$ **do**
  $vals \leftarrow$ approve(est_i)
  if $vals = \{v\}$ for some v **then**
    $propose_i \leftarrow v$
  otherwise $propose_i \leftarrow \perp$

  $c \leftarrow$ whp_coin(r)

  $props \leftarrow$ approve($propose_i$)
  if $props = \{v\}$ for some $v \neq \perp$ **then**
    $est_i \leftarrow v$
    if $decision_i = \perp$ **then**
      $decision_i \leftarrow v$
  else
    if $props = \{\perp\}$ **then**
      $est_i \leftarrow c$
    else $\quad$ %$props = \{v, \perp\}$
      $est_i \leftarrow v$

BA WHP

$est_i \leftarrow v_i$
$decision_i \leftarrow \perp$

if $vals = \{v\}$ for some v **then**
  $propose_i \leftarrow v$
otherwise $propose_i \leftarrow \perp$

$c \leftarrow$ whp_coin(r)

$props \leftarrow$ approve$(propose_i)$
if $props = \{v\}$ for some $v \neq \perp$ **then**
  $est_i \leftarrow v$

else
  if $props = \{\perp\}$ **then**
    $est_i \leftarrow c$
  else $\%props = \{v, \perp\}$
    $est_i \leftarrow v$

Word complexity: $O(n \log^2 n)$

Not a COINcidence Summary

- First formalization of randomly sampled committees using cryptography in asynchronous settings
- First sub-quadratic asynchronous shared coin and BA WHP algorithms
- Expected $\tilde{O}(n)$ word complexity and $O(1)$ expected time

Limitations:

- Binary consensus only
- Safety and liveness only WHP
- One-shot algorithm (not BAB/SMR)
- Non-optimal resilience – improved by [Blum et al. 2020]

Making It Scale

VRFs

- Solve BA
  with high probability (WHP)
  (probability of being correct tends to 1 as $n \to \infty$)
- Complexity: $\widetilde{O}(n)$

Perfect coin

- Solve BAB
  with deterministic safety,
  probabilistic liveness
- *Amortized* complexity:
  O(n) or *O(n log n)* per agreement

All You Need is DAG

Idit Keidar
Technion

Oded Naor
Technion

Eleftherios Kokoris-Kogias
Novi Research

Alexander Spiegelman
Novi Research

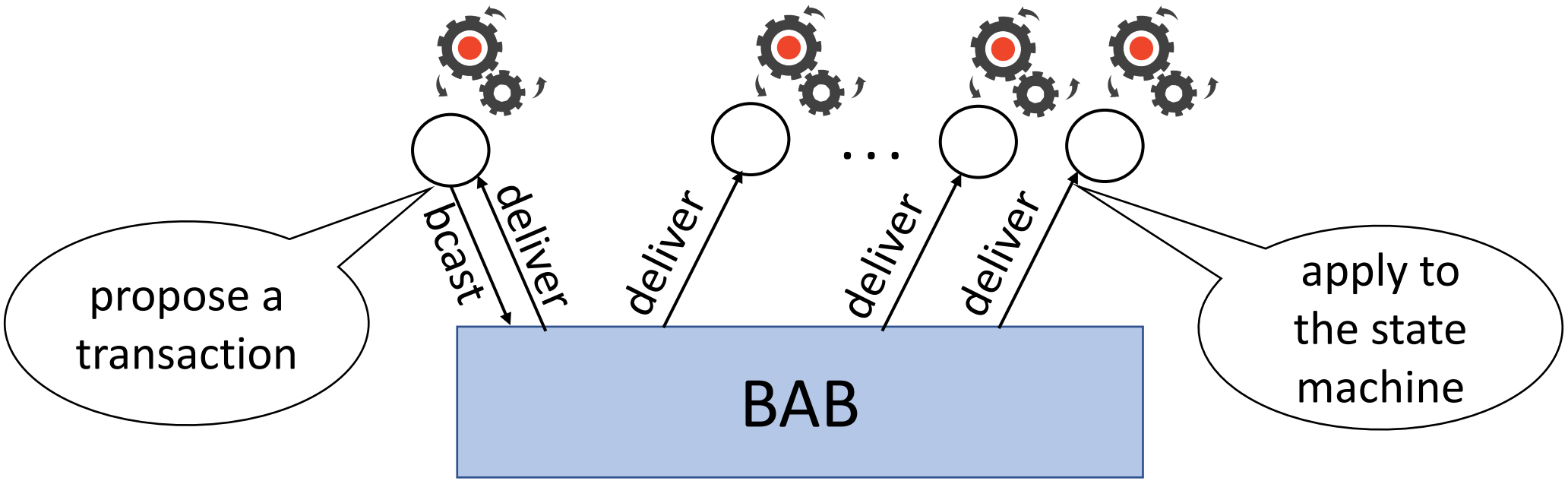
PODC 2021

Model

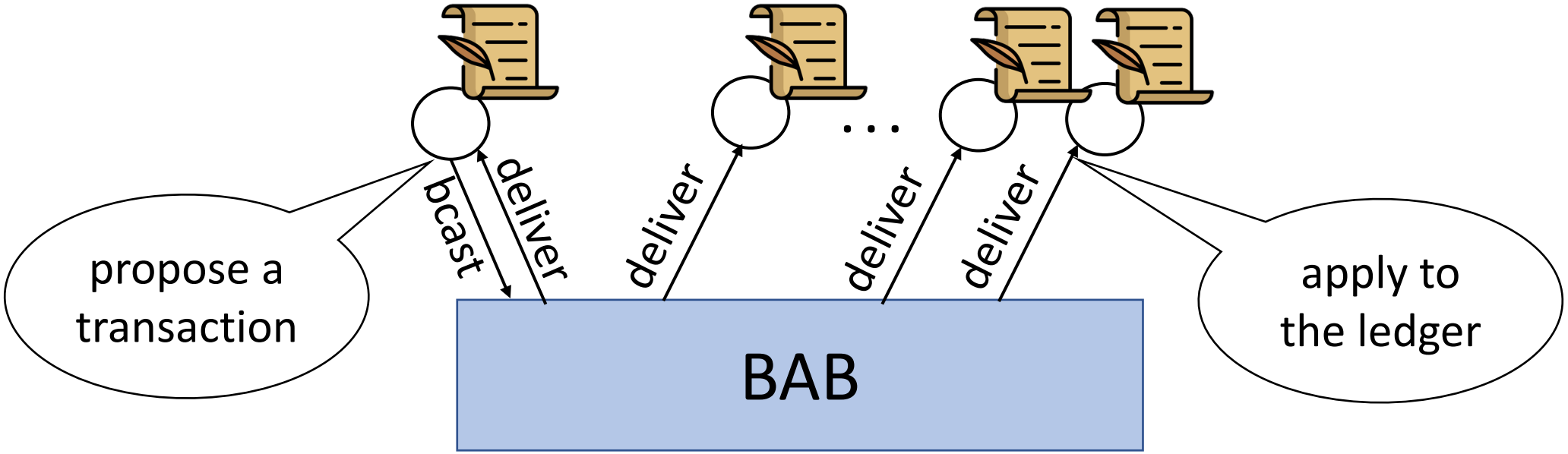
- Asynchronous
- Byzantine faults, optimal resilience: $f < n/3$

- Crypto: PKI, threshold signatures
  - Can implement global perfect coin using secret sharing
  - Always safe (information theoretically)
  - Unpredictable to a computationally bounded adversary

BAB: Byzantine Atomic Broadcast



BAB: Byzantine Atomic Broadcast



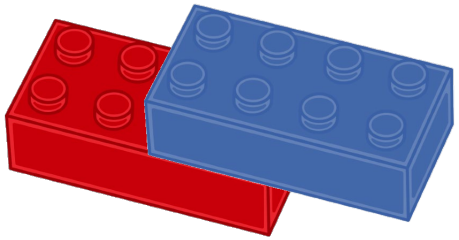
BAB: Byzantine Atomic Broadcast

Reliable Broadcast + Total Order

- All messages sent by correct processes are eventually delivered (fairness)
- Correct processes deliver the same messages
  - possibly in different orders
- Liveness with probability 1

- Correct processes deliver messages in the same order

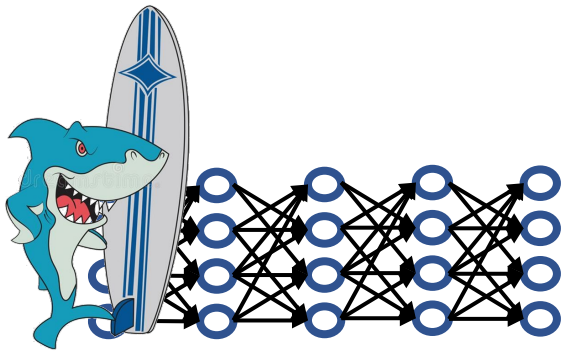
Building Blocks



- Reliable broadcast
  - Bracha broadcast
  - Guerraoui et al. (gossip-based, success probability 1-ε)
  - Cachin & Tessaro verifiable dispersal

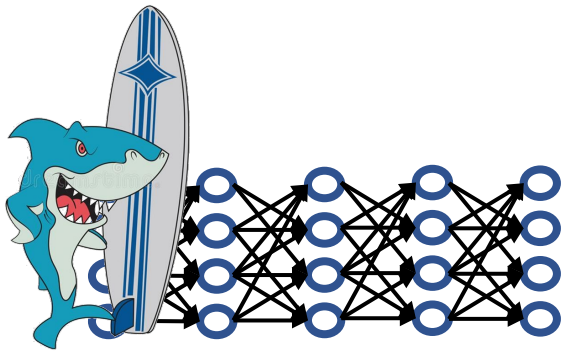
- Unpredictable leader election
  - Based on the perfect coin
  - Same sequence of leaders at all nodes – information theoretically secure
  - Unpredictable by a bounded adversary – *needed only for liveness*

DAG-Rider



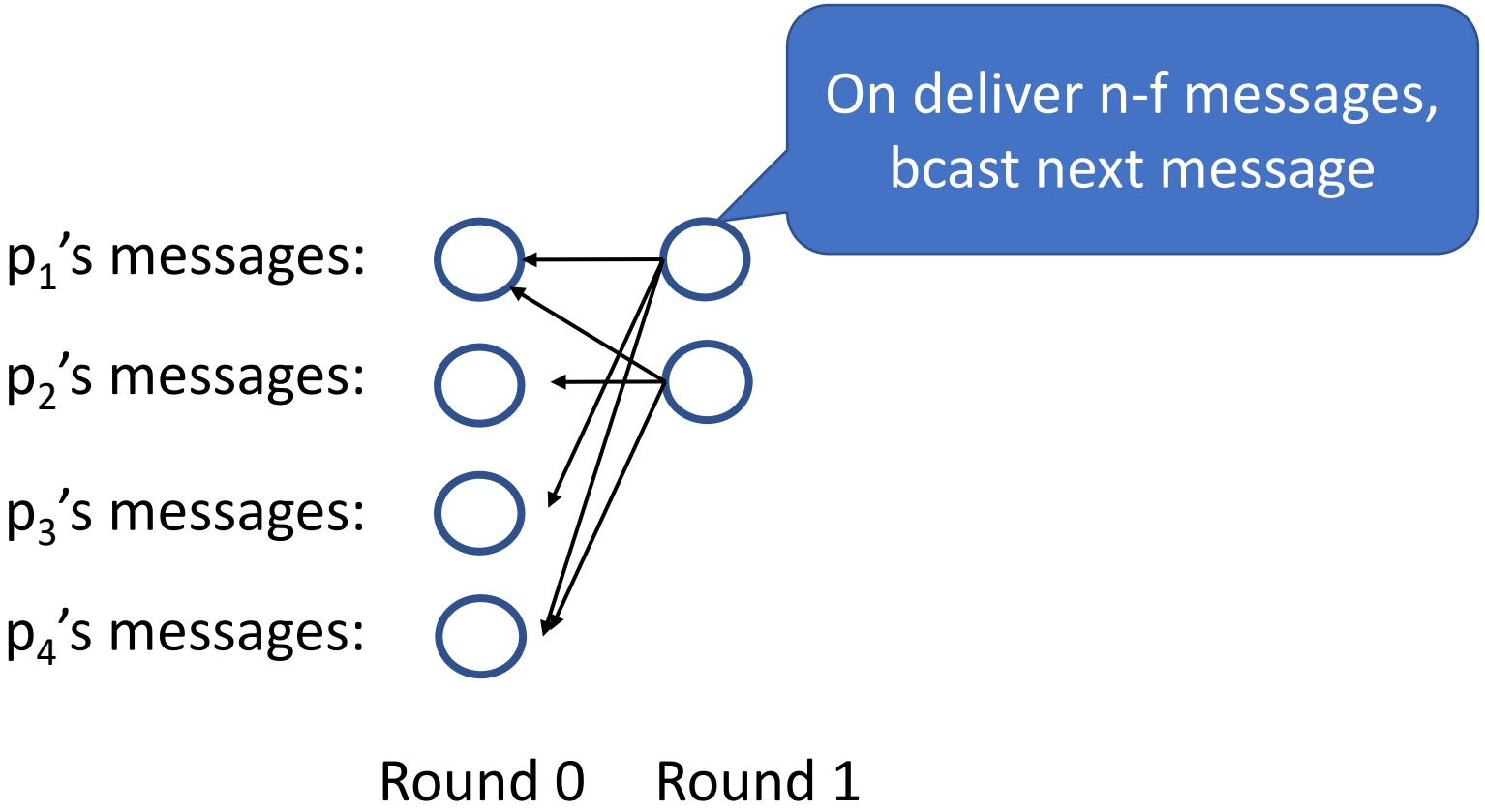
1. Build a DAG
   - Representing causal dependencies among messages
   - Reliably broadcast the DAG
   - ~30 lines of pseudo-code
2. Elect random leaders (vertices in the DAG)
   - Use the DAG to generate shared randomness
3. Order Messages
   - Sequence the leaders' entire causal history
   - Locally, zero overhead
   - ~30 lines of pseudo-code

DAG-Rider

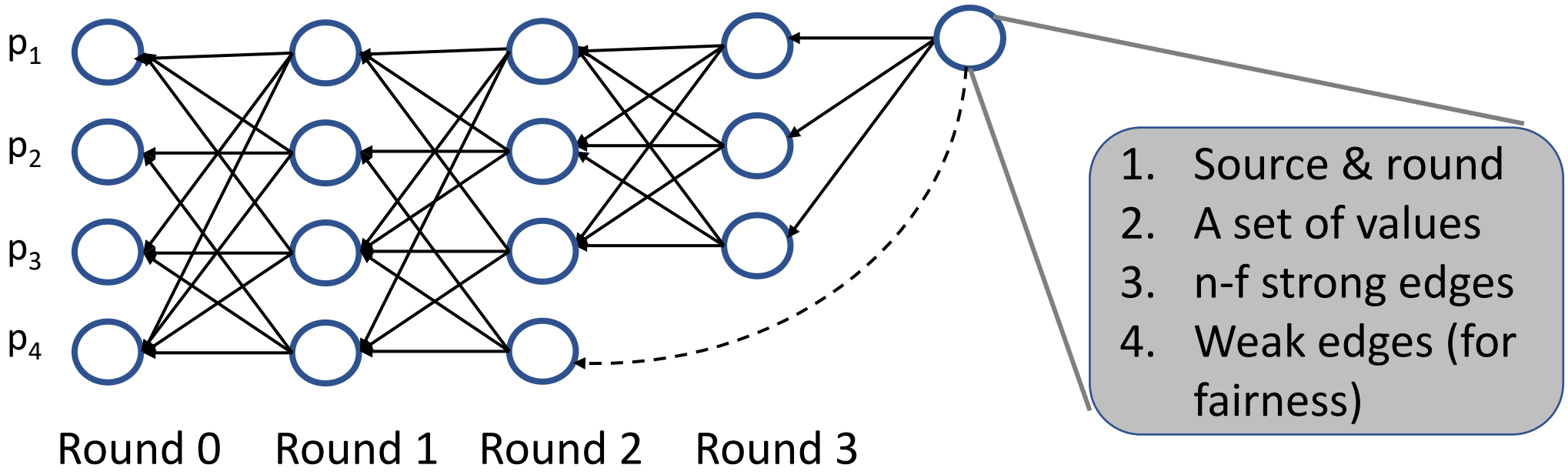


1. Build a DAG
   - Representing causal dependencies among messages
   - Reliably broadcast the DAG
   - ~30 lines of pseudo-code
2. Elect random leaders (vertices in the DAG)
   - Use the DAG to generate shared randomness
3. Order Messages
   - Sequence the leaders' entire causal history
   - Locally, zero overhead
   - ~30 lines of pseudo-code

1. Build a DAG


On deliver n-f messages, bcast next message
p_1's messages:
p_2's messages:
p_3's messages:
p_4's messages:
Round 0
Round 1

The DAG

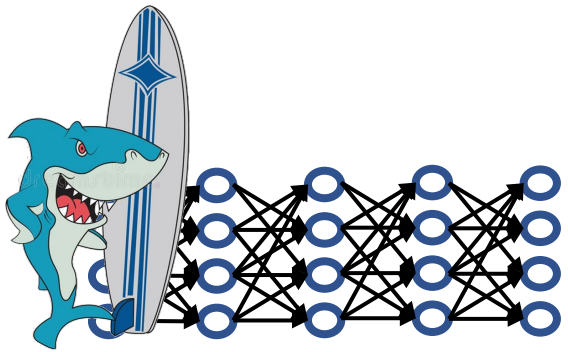

p_1
p_2
p_3
p_4
Round 0
Round 1
Round 2
Round 3
1. Source & round
2. A set of values
3. n-f strong edges
4. Weak edges (for fairness)

Reliably Broadcast the DAG

- No equivocation: all processes see the same DAG, eventually
- But their partial views may temporarily differ

- Complexity:
  - Each vertex includes a linear number of edges
  - Propose $\Omega(n)$ messages in each vertex to amortize costs
  - Reliable broadcast cost depends on underlying protocol

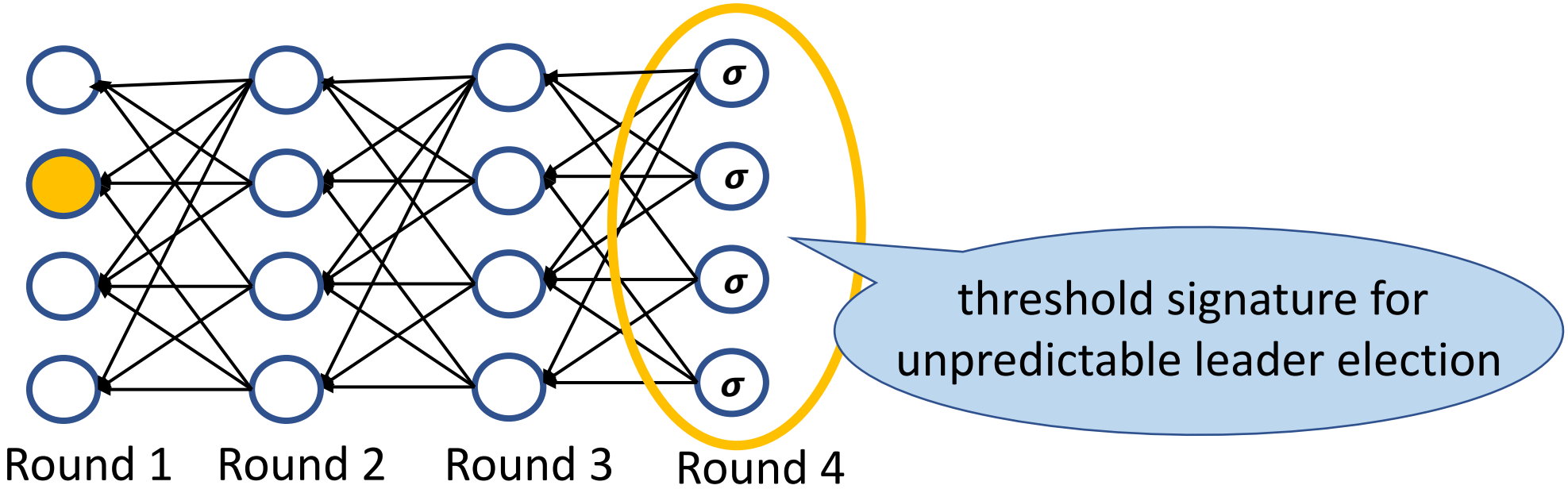
DAG-Rider



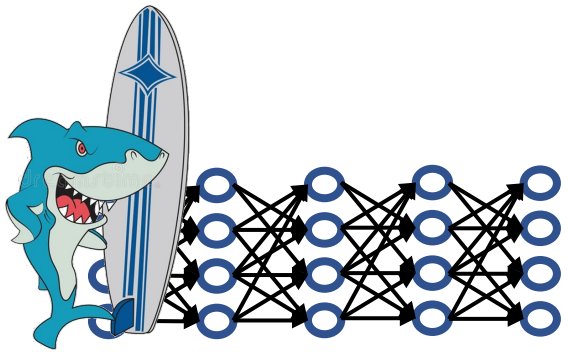
1. Build a DAG
   - Representing causal dependencies among messages
   - Reliably broadcast the DAG
   - ~30 lines of pseudo-code
2. Elect random leaders (vertices in the DAG)
   - Use the DAG to generate shared randomness
3. Order Messages
   - Sequence the leaders' entire causal history
   - Locally, zero overhead
   - ~30 lines of pseudo-code

2. Elect Leaders

- Divide DAG into waves of four rounds
- Use round i+3 messages to elect a leader in round i



DAG-Rider

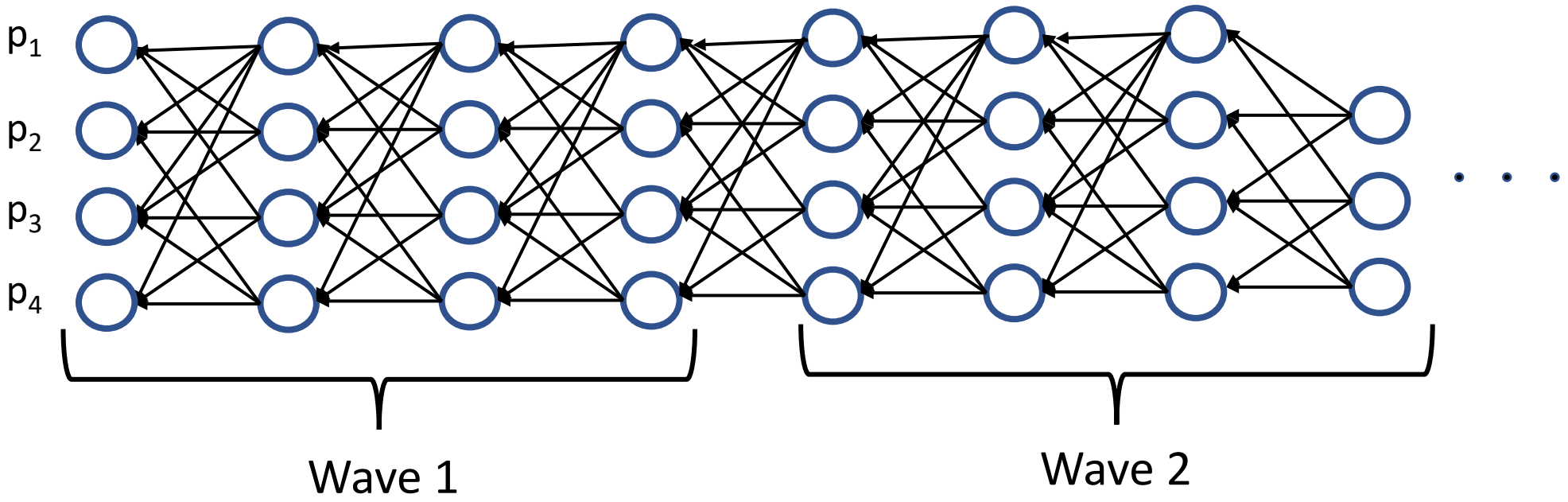


1. Build a DAG
   - Representing causal dependencies among messages
   - Reliably broadcast the DAG
   - ~30 lines of pseudo-code
2. Elect random leaders (vertices in the DAG)
   - Use the DAG to generate shared randomness
3. Order Messages
   - Sequence the leaders' entire causal history
   - Locally, zero overhead
   - ~30 lines of pseudo-code

3. Order Messages

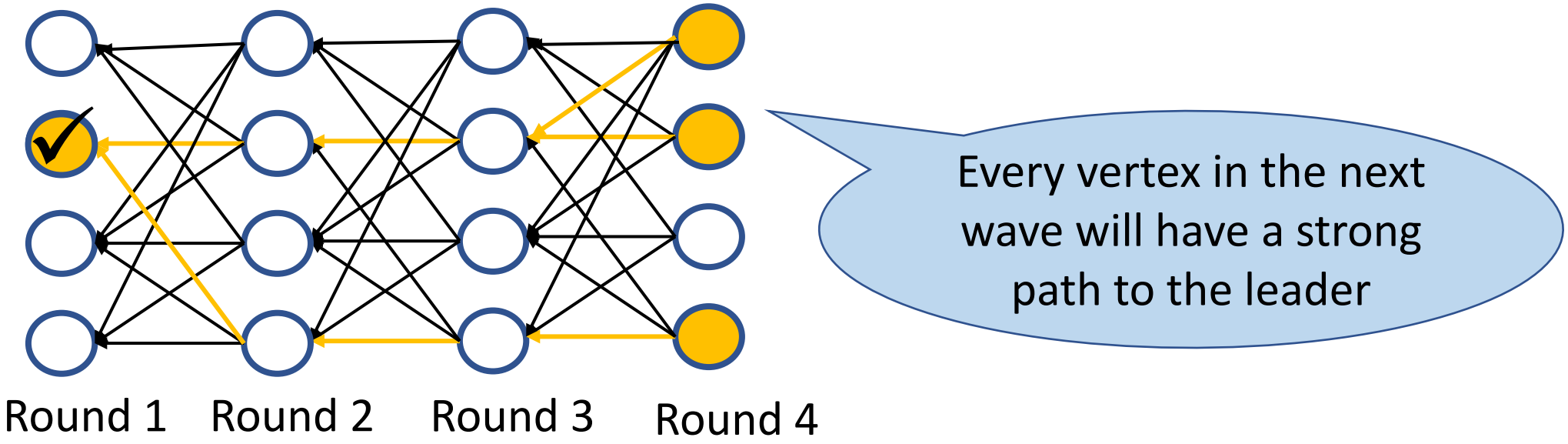
- Based on the local view of the DAG + elected leaders
- Because processes may have different views, *commit* only leaders that have sufficient support
  - Appear in sufficiently many DAGs

Divide the DAG into Waves



Wave Leader Commit Rule

- Commit the wave's elected leader if there are 2f+1 vertices in the 4th round with strong paths to it
  - Weak edges do not count



Leveraging the Common Core Principle

"After three rounds of all-to-all sending and collecting accumulated sets of values from *n-f* processes, all correct processes have at least $2f+1$ common values"

∴ At least 2f+1 round 1 vertices satisfy the commit rule

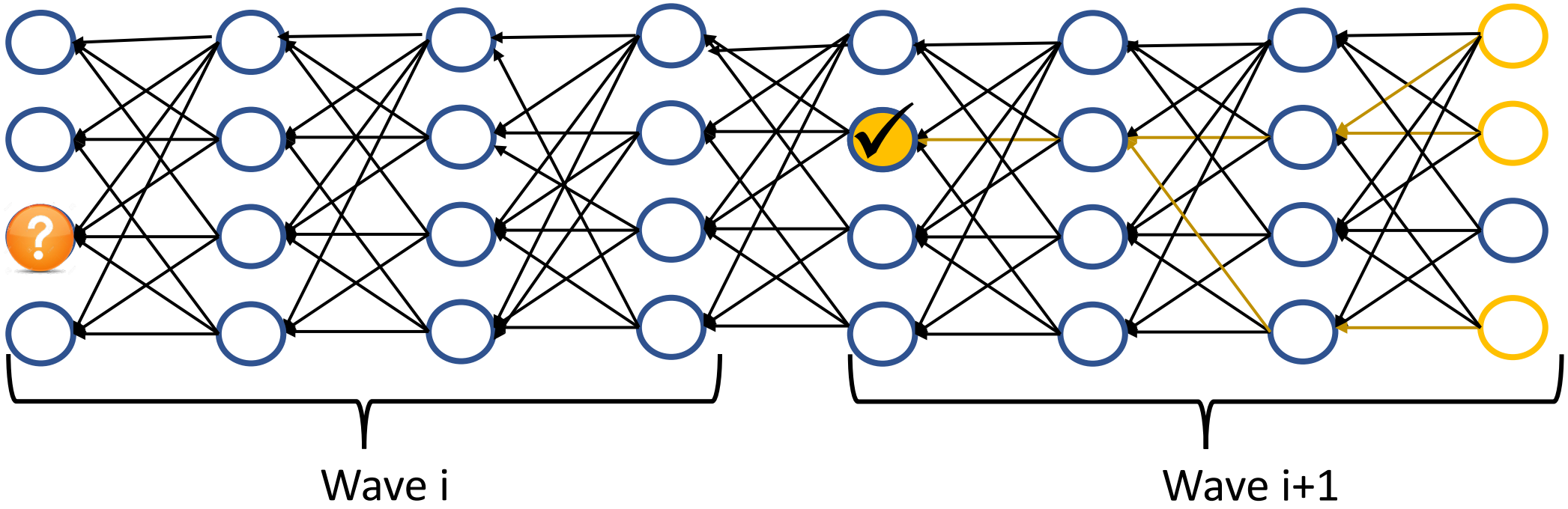
∴ The elected vertex satisfies it with probability at least 2/3

- It is unpredictable until the end of round 4

∴ O(1) expected latency until a leader is committed

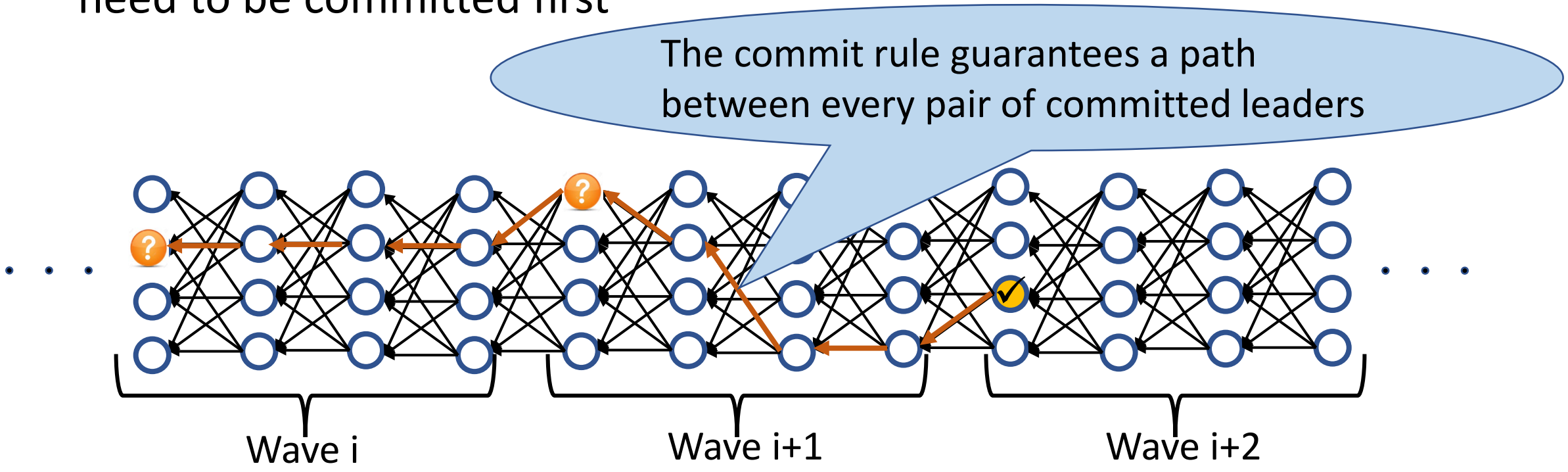
What About Uncommitted Leaders?

- Another process may have committed wave i



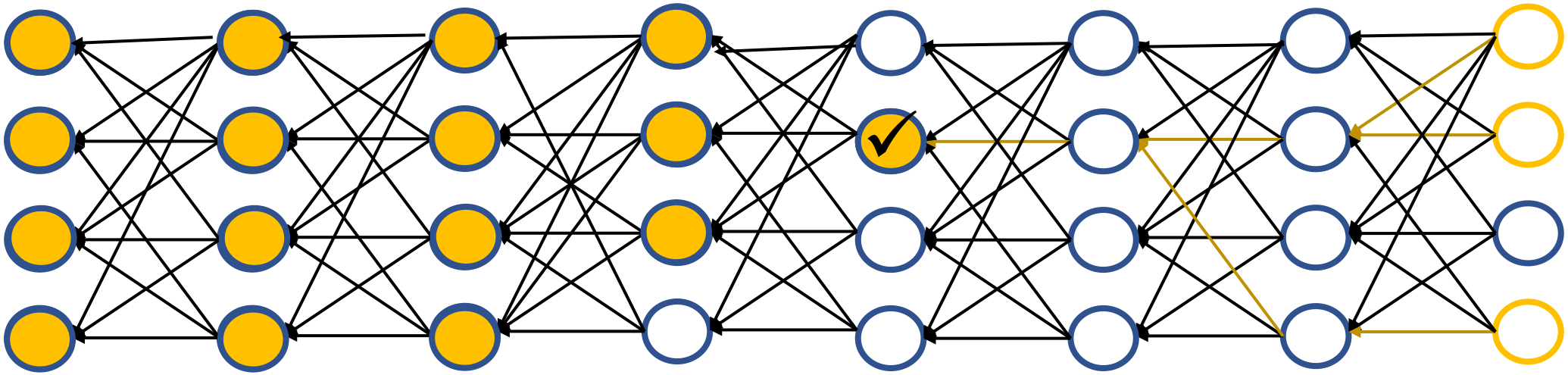
Committing the Same Leaders

- When committing a leader in wave i, check if any previous leaders need to be committed first



Sequencing Messages

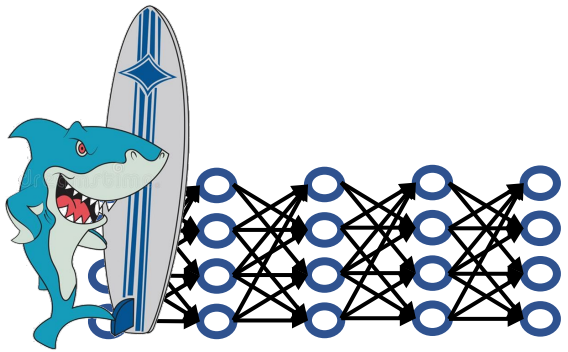
- For each leader, sequence its entire causal history
- Weak edges count



DAG-Rider Variants vs. Previous Work

	Communication	Latency	Fairness	Post-quantum
VABA SMR	$O(n^2)$	$O(\log n)$	no	Unsafe
Dumbo SMR	amortized $O(n)$	$O(\log n)$	no	Unsafe
DAG-Rider + Bracha	amortized $O(n^2)$	$O(1)$	with probability 1	Safe
DAG-Rider + Gossip	amortized $O(n \log n)$	$O(\frac{\log n}{\log \log n})$	with probability $1-\varepsilon$	Safe
DAG-Rider + verifiable dispersal	amortized $O(n)$	$O(1)$	with probability 1	Safe

DAG-Rider Summary



- O(1) expected latency
- O(n) amortized message complexity per agreement
  - With appropriate reliable broadcast and batching
- Post-quantum safe
  - Use crypto only for leader unpredictability
- Fair with probability 1
  - Every message sent by a correct process is eventually delivered