PACE: Fully Parallelizable BFT from Reproposable Byzantine Agreement

Mansheel Agarwal, Yinuo Tang, Shu Zhang,
Yufeng Chen, Xiran Li

Background

Consensus

Definition:

- A group of nodes agree on a single shared state
- Ensure reliability and consistency for important systems (blockchains, database)

Key Feature

- Agreement: all non-faulty nodes must agree on same value → consistency
- Validity: agreed value must valid
- Fault Tolerance: must tolerate certain number of faults
- Termination: comes to an end, with all non-faulty nodes deciding on a value

Asychronous BFT

Definition

- Consensus protocols that operate in asynchronous systems
- No guarantees on message delivery times
- Real-world Scenarios: Global internet networks, where delays can vary and unpredictable

Key Properties, how asynchronous bft work

- No timing assumptions
- Byzantine Fault Tolerance: up to $\lfloor (n-1)/3 \rfloor$ Byzantine node with n nodes
- Probabilistic guarantees: asynchronous BFT protocols often use randomness to reach agreement with high probability
  - allows the system to function effectively in environments with high uncertainty.

HoneyBadger

Asynchronous BFT for distributed systems

Key Components

- Reliable Broadcast (RBC): all non-faulty nodes have the same messages
  - a node broadcast a value to all other nodes
  - nodes validate the message and rebroadcast it to ensure consistency
  - sufficient nodes receives and validate the message → considered delivered
- Asynchronous Binary Agreement (ABA): all nodes agree on same binary value
  - Nodes propose a binary value (e.g., 0 or 1).
  - Multiple rounds of communication are used to exchange votes.
  - A quorum-based approach ensures agreement even with Byzantine nodes.

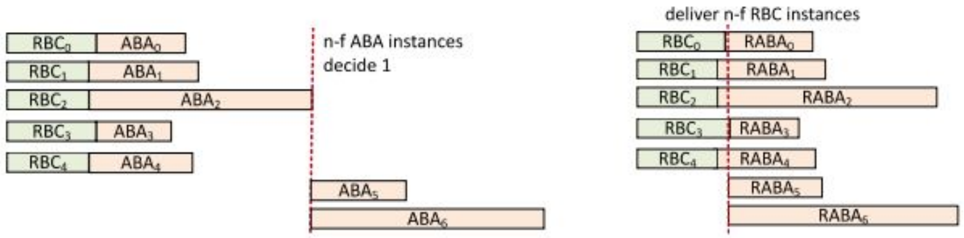
HoneyBadger

Workflow of HoneyBadger

- Proposal Phase:
  - Each propose a batch of transaction using RBC
- Agreement Phase:
  - Node run ABA to decide which proposal to include
- Output Phase:
  - Agreed-upon batches are combined and added to blockchain

PACE

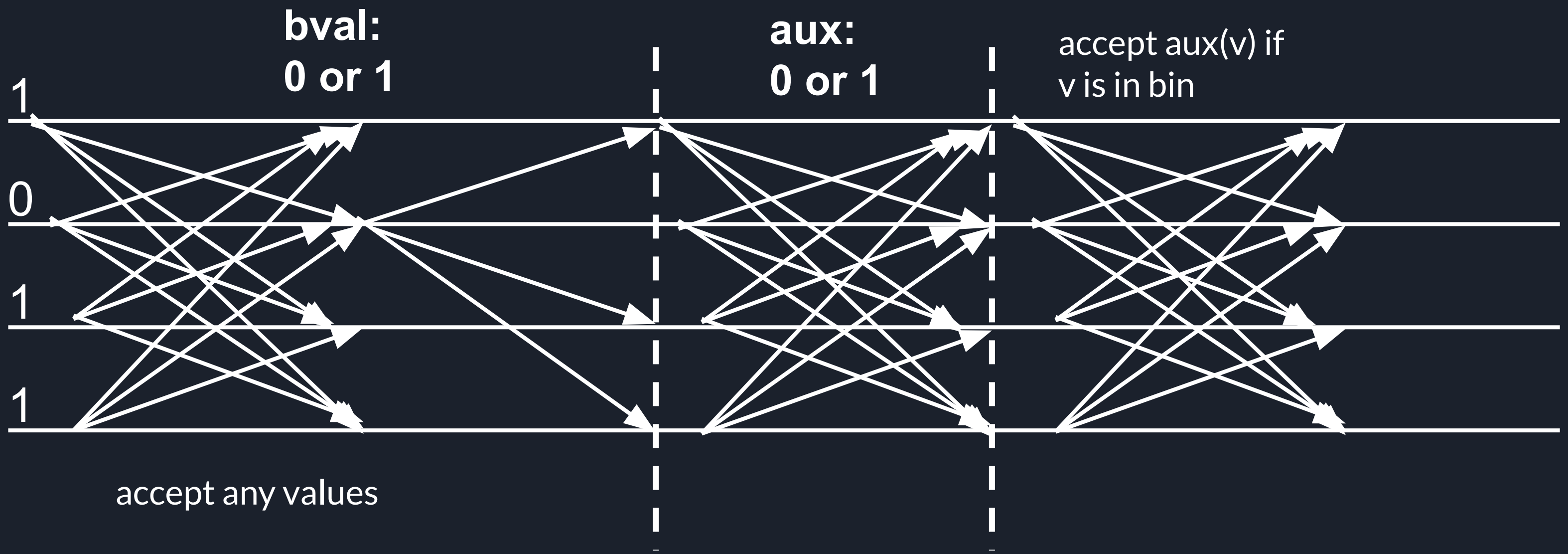
- WHAT is PACE
  - a framework that enables fully parallelizable ABA instances
  - key feature: Reproposable ABA (RABA) — allows replicas to change votes and make decisions faster.
- WHY:
  - Remove BKR bottleneck



(a) BKR (e.g., HoneyBadgerBFT, BEAT). **(b) PACE.**

Existing ABA Protocols

MMR ABA protocol



- every replica broadcasts bval(b)
- upon receiving f + 1 bval(v), send bval(v)
- upon receiving 2f + 1 bval(v), add v to binv
- upon receiving 2f + 1 bval(v), send aux(v) (i.e., the first bin value)

dispersal phase

- upon receiving 2f + 1 auxf(v) and v = coin, decide (v), otherwise set est_ r+1 to coin and enter the next round

agreement phase

MMA ABA con't ...

Liveness issues

- a malicious network schedular can force correct replica to always enter next round with inconsistent value)

Crain (L/H protocol (improved MMR protocol)

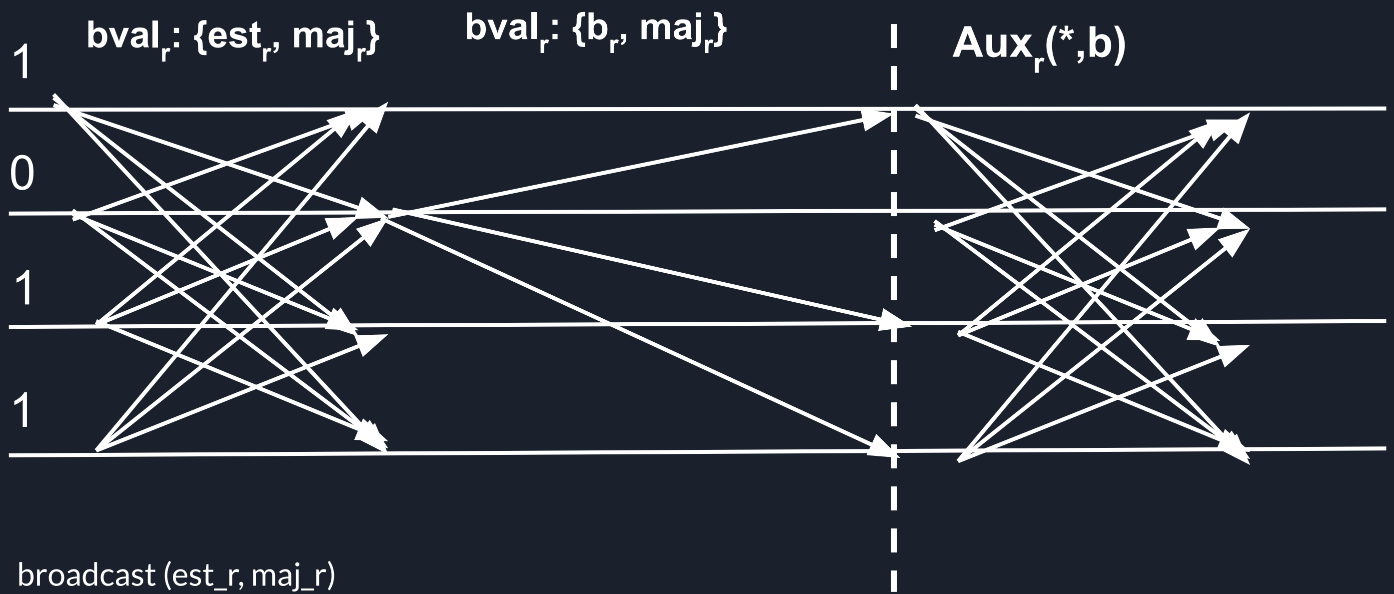
Crain L:

- Similar to MMR - using weak common coin
- It combines proposals and auxiliary message into a single step
- This optimization reduces the number of messages exchanged per round

Crain H:

- High threshold coin
- Stronger randomness / more cryptographically secure coin → faster convergence
- more expensive

Pillar



- broadcast (est_r, maj_r)
- upon receiving f+1 bval_r(b, ∗) messages, broadcast bvalr(b,maj_r), where the b depends on the mar_r
- upon receiving 2f + 1 bval_r(b, ∗) messages add b to the bin

- upon receiving 2f + 1 bval_r(b, ∗) messages, broadcast aux_r(b,b), depending on the value of the validity flag (aux(b,b) or aux(⊥, b))
- upon receiving 2f + 1 auxiliary messages, make a decision based on the common coin
- fallback to set est_r+1 = s_r in r = 0 or est_r+1 = majority(vals_r) for f> 0

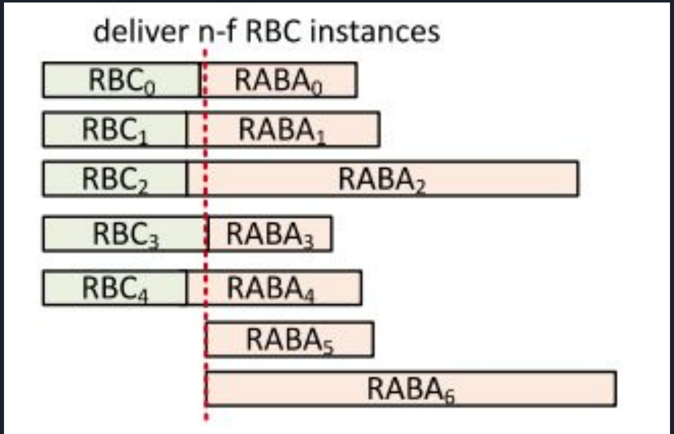
Pillar’s solution to guarantee the liveness

Dual-Value Messaging:

- By requiring auxiliary values to support the main value, it guarantees that correct nodes only vote for the same value in the **agreement phase**, even in the presence of adversarial nodes.

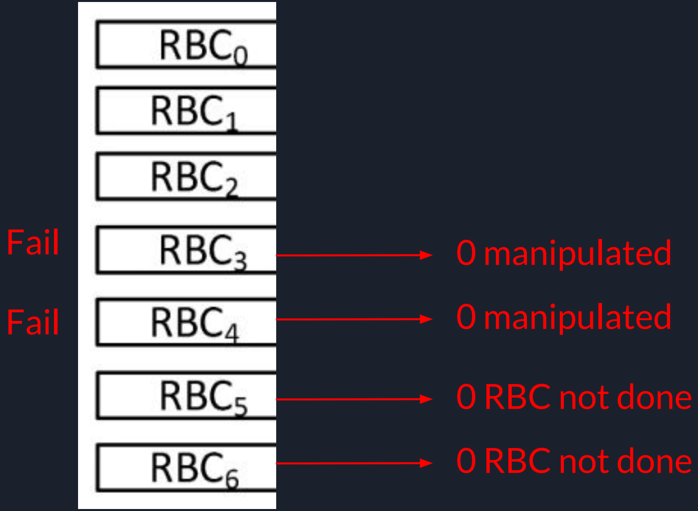
e.g. 4 replica R1-R4, R4 (faulty), in MMA, dispersal phase, say R1-R3 broadcast 1 to all the other nodes and R4 broadcast 1 to R2 and R3 but 0 to R1 , this will result R1 receiving conflicting message {0,1}. in pillar, due to the presence of maj_r, R1 would receive {0,1,1} in its maj_r set and adopt 1 as it is the majority value

RABA (reproposable ABA)



- A modification of ABA
- Add repropose feature that make RABA able to invoke once after n-f RBCs finished.

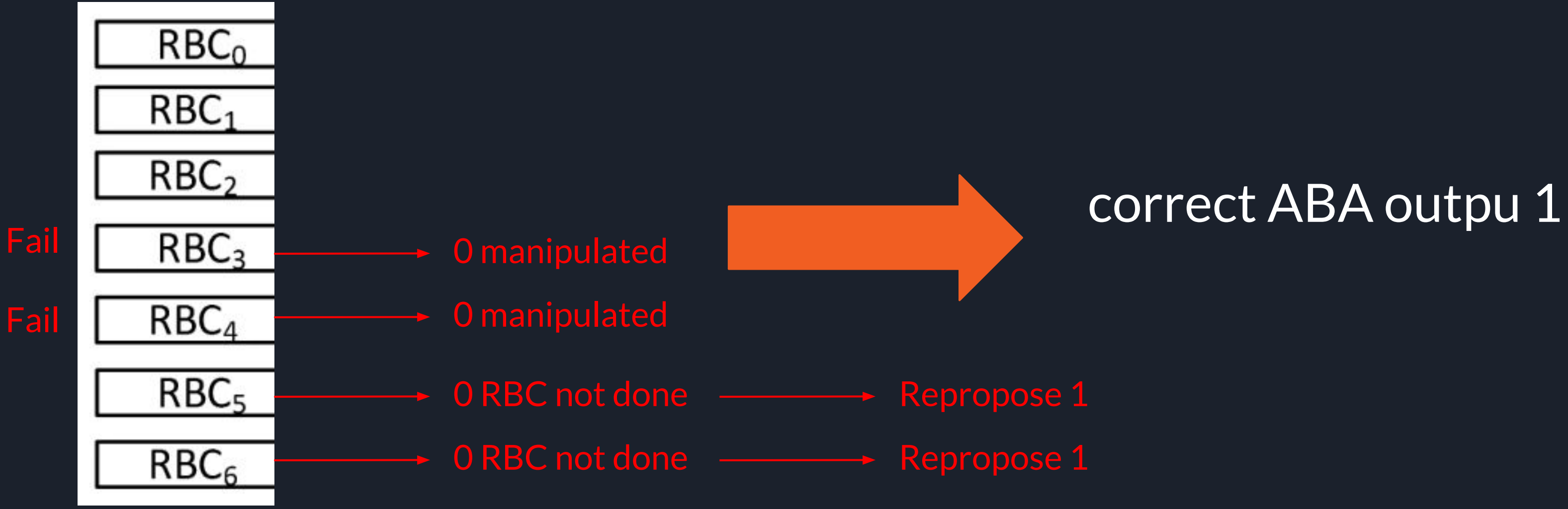
Why directly run after n-f not good



All ABA outpu 0
or extra round

Upon completing n-f RBC instances: for those received ones, vote 1, otherwise, vote 0.

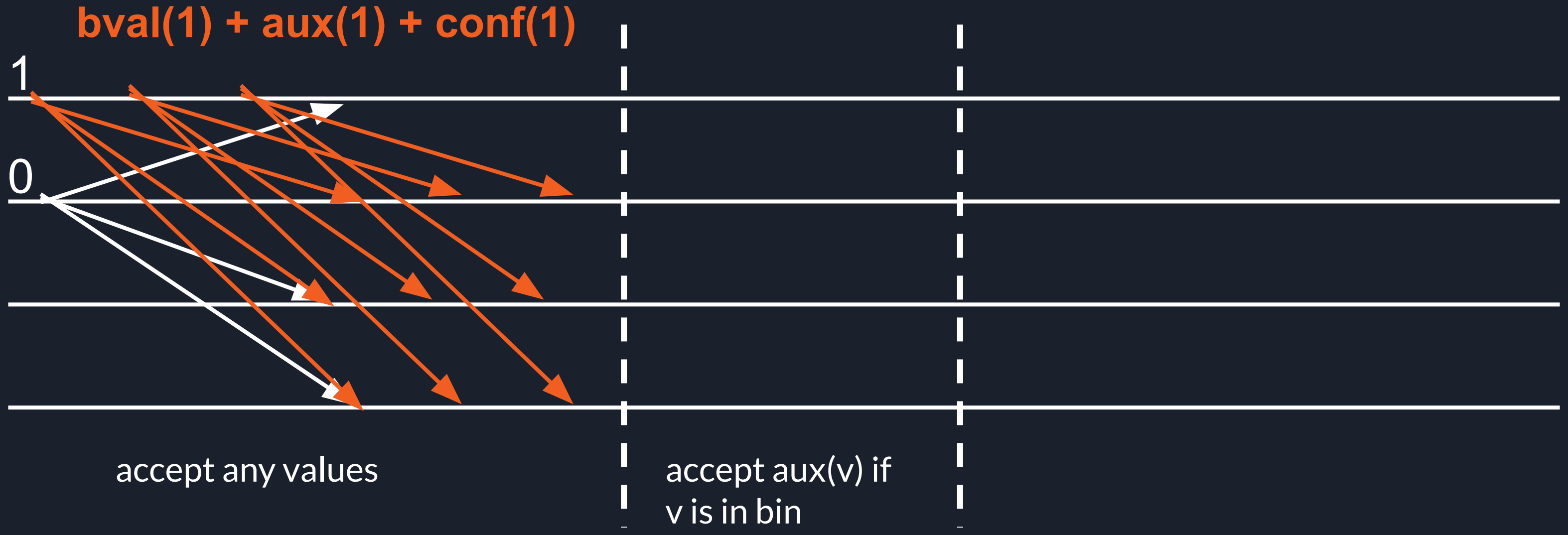
Repropose definition



A replica who previously voted 0 (RBC not done at that time) can have chance to vote 1 (now the RBC done).

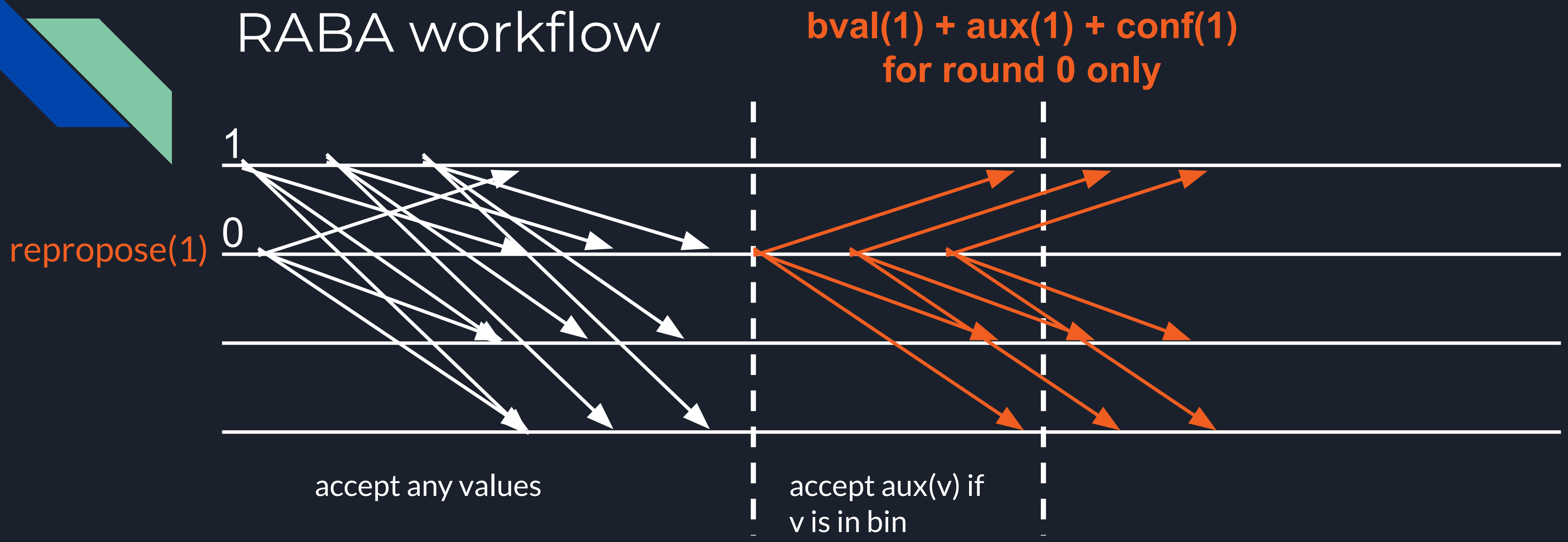
Only 0 to 1, not 1 to 0.

RABA for Cobalt ABA workflow



propose(v): if v=1, send bval(1), aux(1), conf(1) simultaneously

RABA workflow

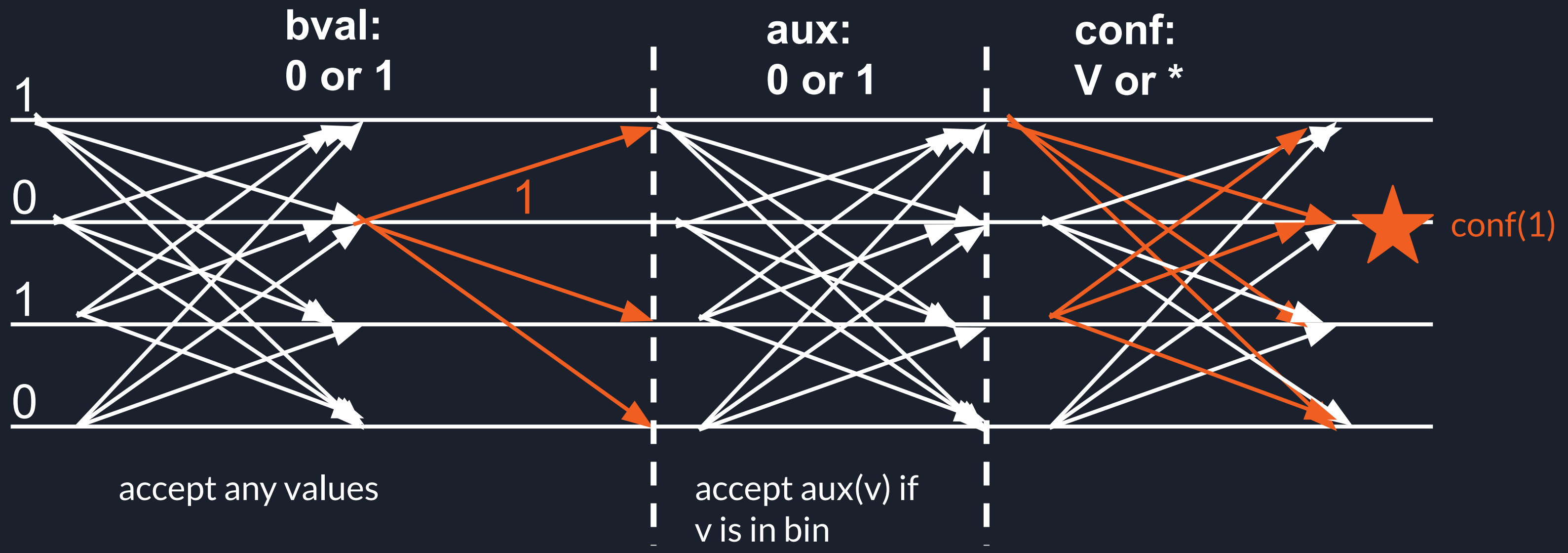


propose(v): if v=1, send bval(1), aux(1), conf(1) simultaneously
repropose(v): if v=1, send bval(1), aux(1), conf(1) for round 0 only (regardless of which round the replica is in)
coin value for round 0 is set to 1

RABA Properties

- Validity: If all correct replicas propose v and never repropose v', then any correct replica that terminates decides v.
- Unanimous termination: If all correct replicas propose v and never repropose v', then all correct replicas eventually terminate.
- Agreement: If a correct replica decides v, then any correct replica that terminates decides v.
- Biased validity: If f+1 correct replicas propose 1, then any correct replica that terminates decides 1.
- Biased termination: If the total number of replicas that propose 1 or repropose 1 >= 2f+1, the protocol terminates.
- Integrity: No correct replica decides twice.

RABA workflow

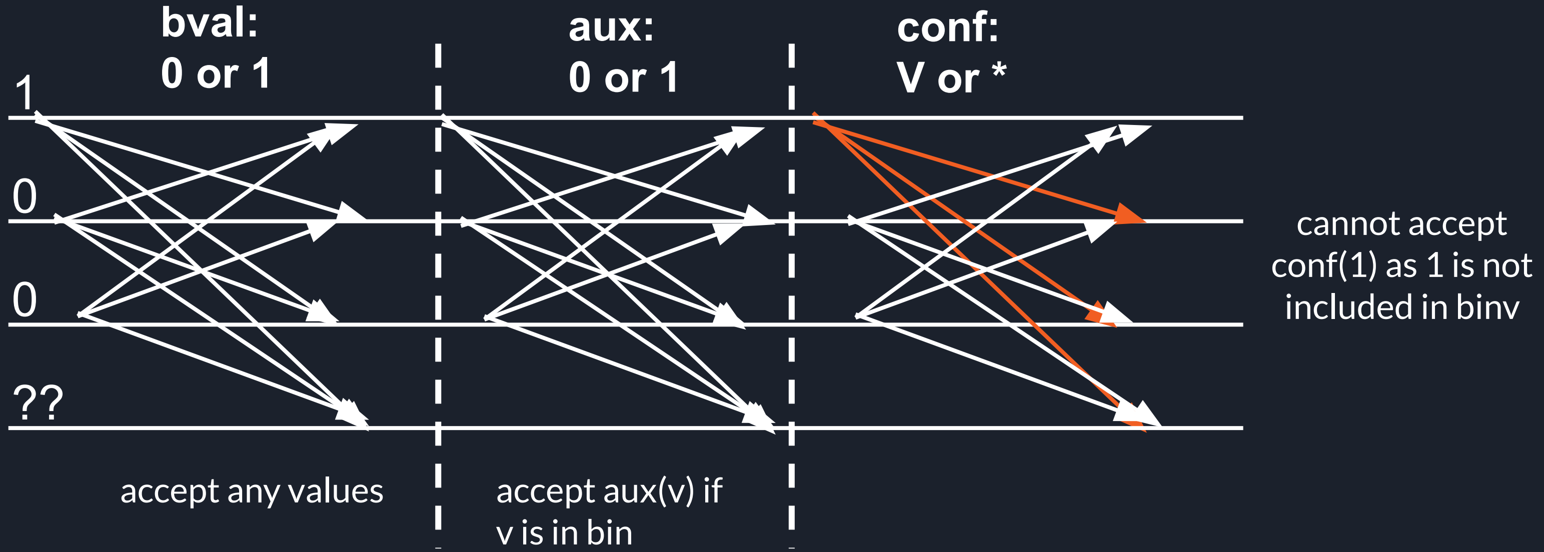


Biased validity: If f+1 correct replicas propose 1, then any correct replica that terminates decides 1.

Why biased validity?

- If f+1 replicas vote for 1, no replica will receive n-f conf(0)

RABA workflow

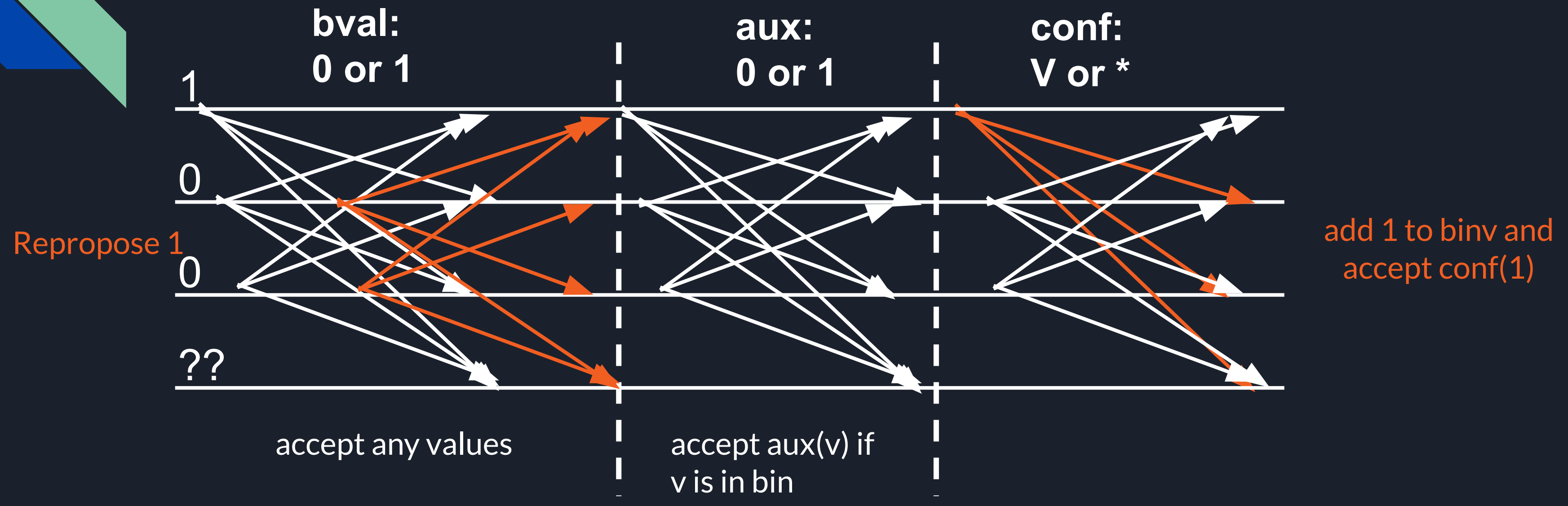


The Termination Issus

- If fewer than f+1 correct replicas propos 1

Biased termination: If the total number of replicas that propose 1 or repropose 1 >= 2f+1, the protocol terminates

RABA workflow



The Termination Issus

- If fewer than f+1 correct replicas propos 1

Biased termination: If the total number of replicas that propose 1 or repropose 1 >= 2f+1, the protocol terminates

Pisa

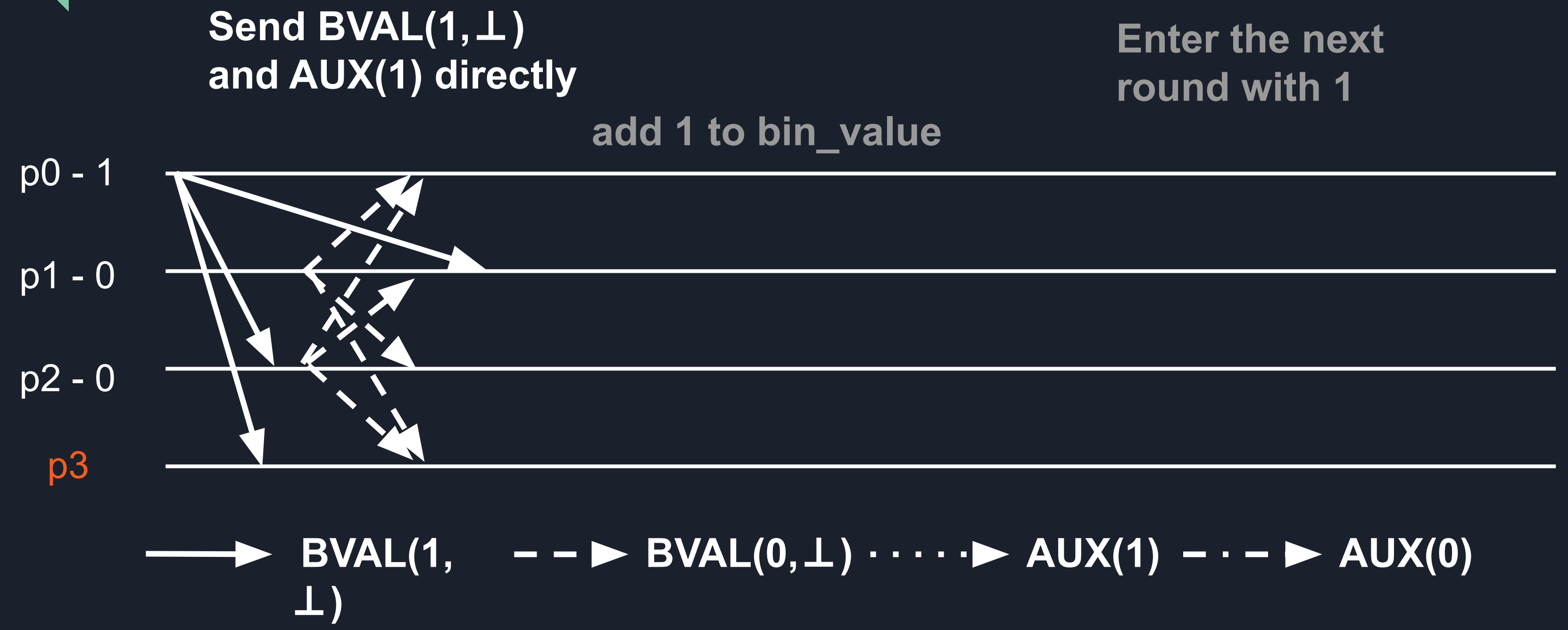
Pisa is a **RABA** protocol built on top of **Pillar**

Key innovation: allows replica to repropose

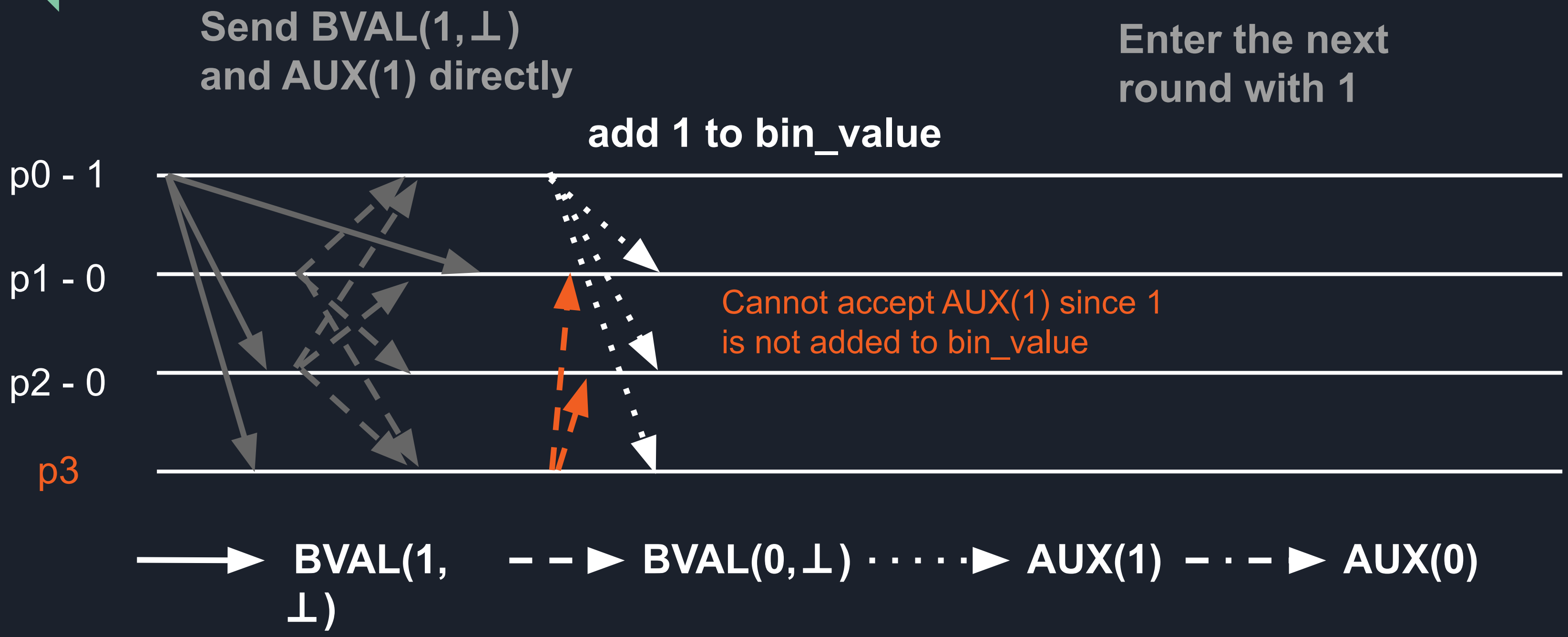
Modifications: only in the initial round

1. if the repropose() event is triggered, replica $\mathbf{p}_i$ broadcasts BVAL(1, ⊥)
2. When a correct replica $\mathbf{p}_i$ proposes 1, it immediately adds 1 to bin_values and broadcasts aux(1, 1)
3. If a replica $\mathbf{p}_i$ propose 0 and receives f+1 BVAL(1, ⊥) messages, it broadcasts BVAL(1, ⊥) and add 1 to bin_values . It also sends aux(1, 1) if it hasn't send aux message
4. The common coin is set to 1, ensuring that if a replica receives n - f aux(1, 1) messages, it can directly terminate the protocol

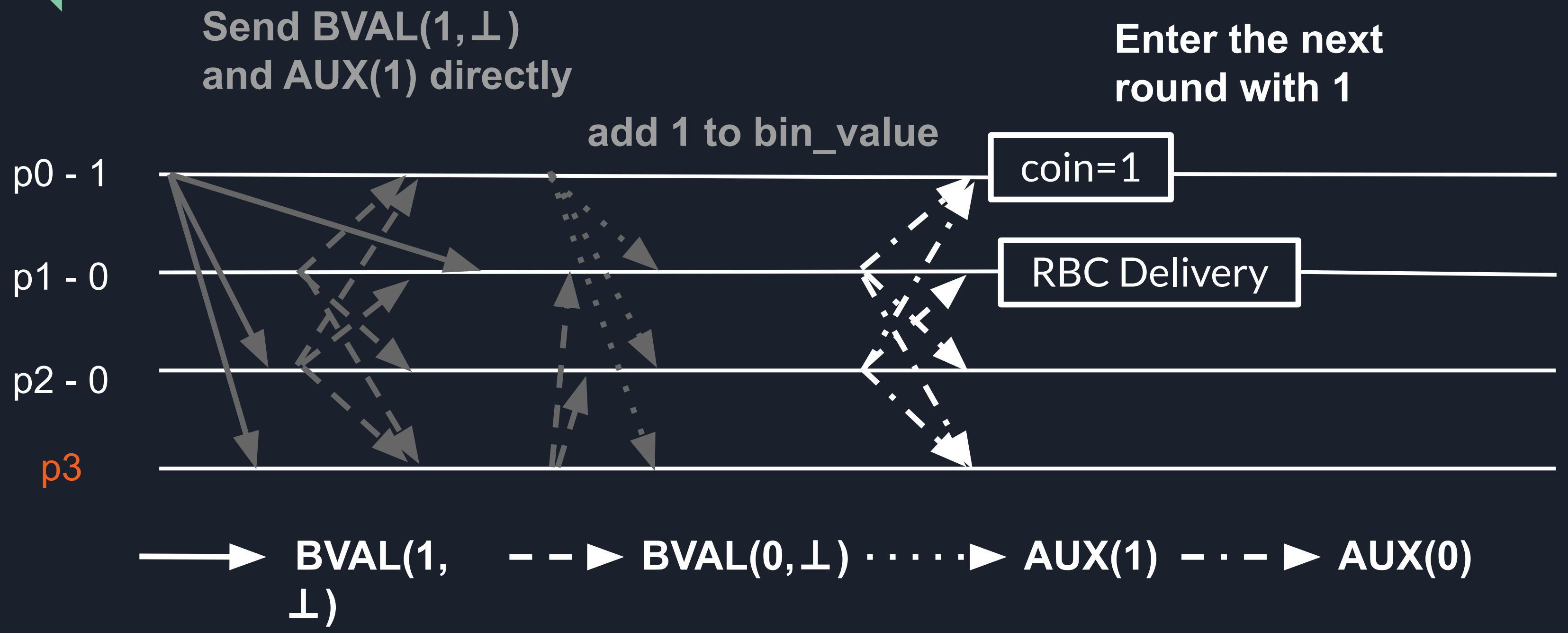
Pisa’s biased termination



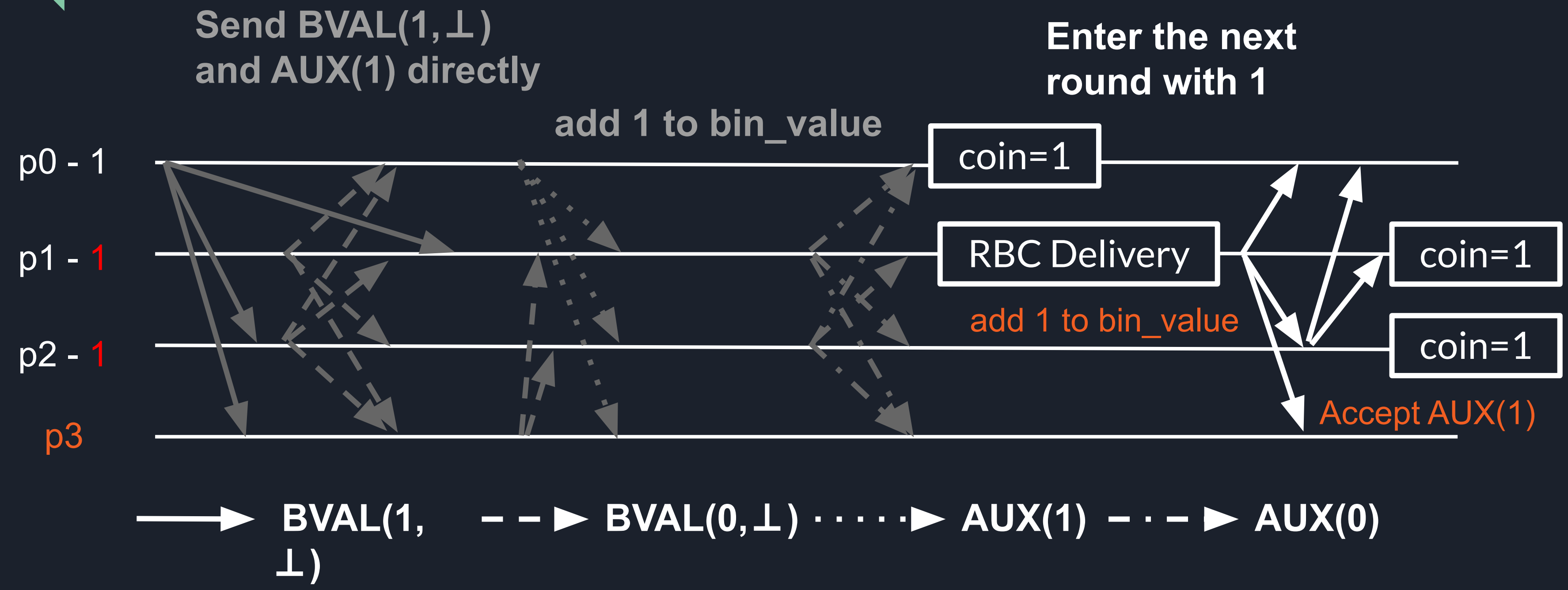
Pisa’s biased termination



Pisa's biased termination



Pisa's biased termination



PACE FRAMEWORK

init
$e \leftarrow 0$ {epoch number}
upon selecting m_i from the buffer of p_i
r-broadcast($[e, i], m_i$) for RBC$_i$
upon r-deliver($[e, j], m_j$) for RBC$_j$
if RABA$_i$ has not been started
propose ($[e, j]$, 1) for RABA$_j$
else
repropose ($[e, j]$, 1) for RABA$_j$
upon delivery of $n - f$ RBC instances
for RABA instances that have not been started
propose ($[e, j]$, 0)
upon decide ($[e, j]$, v) for any value v for all RABA instances
let S be set of indexes for RABA instances that decide 1
wait until r-deliver($[e, j], m_j$) for all RABA$_j$ such that $j \in S$
a-deliver($\cup_{j \in S} \{m_j\}$) in some deterministic order
$e \leftarrow e + 1$

PACE FRAMEWORK

init

$e \leftarrow 0$ {epoch number}

upon selecting m_i from the buffer of p_i

r-broadcast($[e, i]$,m_i) for RBC$_i$

upon r-deliver($[e, j]$,m_j) for RBC$_j$

if RABA$_i$ has not been started

propose ($[e, j]$, 1) for RABA$_j$

else

repropose ($[e, j]$, 1) for RABA$_j$

upon delivery of $n - f$ RBC instances

for RABA instances that have not been started

propose ($[e, j]$, 0)

upon decide ($[e, j]$, v) for any value v for all RABA instances

let S be set of indexes for RABA instances that decide 1

wait until r-deliver($[e, j]$,m_j) for all RABA$_j$ such that $j \in S$

a-deliver($\cup_{j \in S} \{m_j\}$) in some deterministic order

$e \leftarrow e + 1$

Assume nodes p1, p2, p3 ,......, pn with f faulty replicas

For each epoch :

- Replica p1 broadcasts a message m1 to others using the r-broadcast primitive of RBC in the format [e,1],m1
- The nodes that receive (or r-deliver) this message :
  - Propose 1 for the corresponding RABA1 if it has not started yet
  - Repropose 1 for RABA1 to ensure agreement if it has already started
- Once n-f RBC proposals are r-delivered, the nodes propose 0 for the RABA instances that have not started yet.
- Every replica has a set S that it updates with the RABA instances that decide 1
- Replicas wait until all of the RABA instances from set S have been delivered
- Finally, the replicas a-delivers the received messages corresponding to set S in some deterministic order.
- Protocol moves to the next epoch <- e+1

WHEN WILL IT STOP ?

RABA doesn’t itself attain termination, rather RBC is used to carefully control the API of RABA and force it to meet the unanimous termination condition or the biased termination condition as demonstrated in the following cases :

- **Case 1: All correct replicas propose 1 for some RABA**: According to unanimous termination, the RABA instance eventually terminates with output 1.
- **Case 2: All correct replicas propose 0:**
  - **Case a: They never repropose 1:** The RABA instance eventually terminates due to unanimous termination.
  - **Case b: Some repropose 1:** The protocol will terminate according to biased termination.
- **Case 3: Some correct replicas propose 0 and some propose 1:** The RABA instance will terminate (similar to Case 2(b)).

	#0	#1	#2	#3	#4	#5	#6
p_0							
p_1							
p_2							
p_3							
p_4							
p_5							

	#0	#1	#2	#3	#4	#5	#6
p_0							
p_1							
p_2	1						
p_3							
p_4							
p_5							

	#0	#1	#2	#3	#4	#5	#6
p_0		1					
p_1		1					
p_2	1						
p_3							
p_4							
p_5							

	#0	#1	#2	#3	#4	#5	#6
p_0		1	1				
p_1		1	1				
p_2	1		1				
p_3			1				
p_4			1				
p_5							
			> f+1				

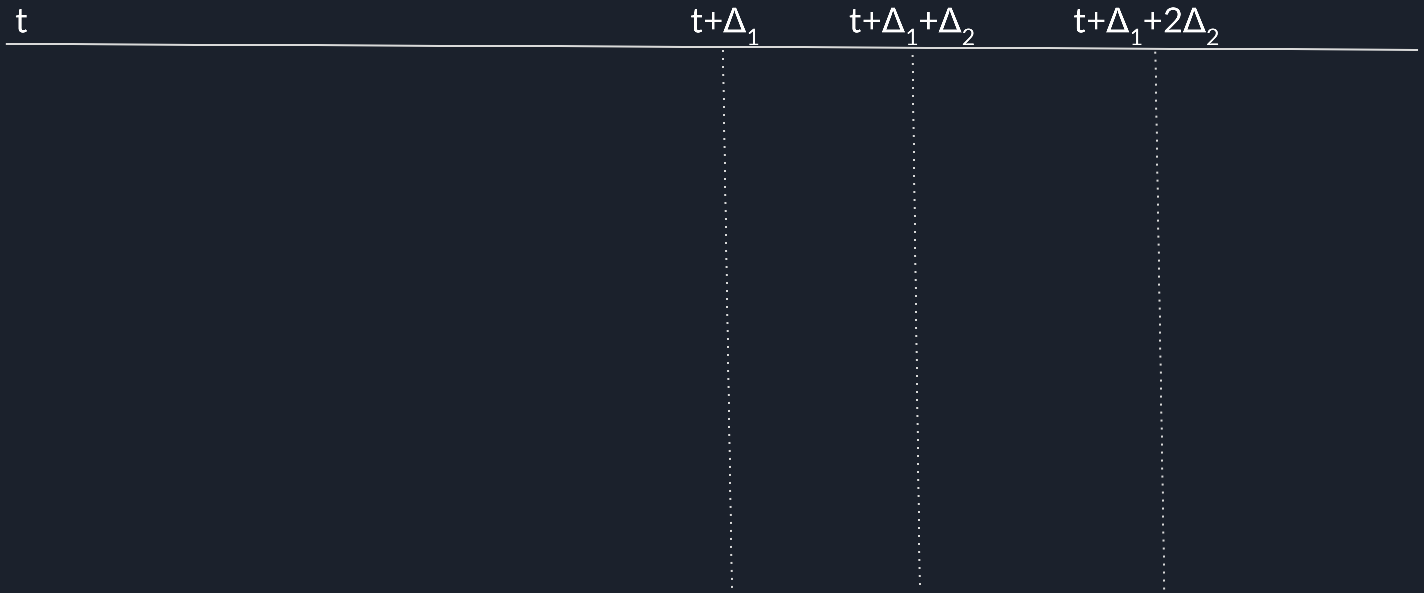
	#0	#1	#2	#3	#4	#5	#6
p_0		1	1	1	1	1	
p_1		1	1	1	1	1	
p_2	1		1	1	1	1	
p_3			1	1	1	1	
p_4			1	1	1	1	
p_5							
			> f+1	> f+1	> f+1	> f+1	

	#0	#1	#2	#3	#4	#5	#6
p_0		1	1	1	1	1	
p_1		1	1	1	1	1	
p_2	1		1	1	1	1	
p_3			1	1	1	1	1
p_4			1	1	1	1	1
p_5							
			$> f+1$	$> f+1$	$> f+1$	$> f+1$	

Better Throughput than BKR

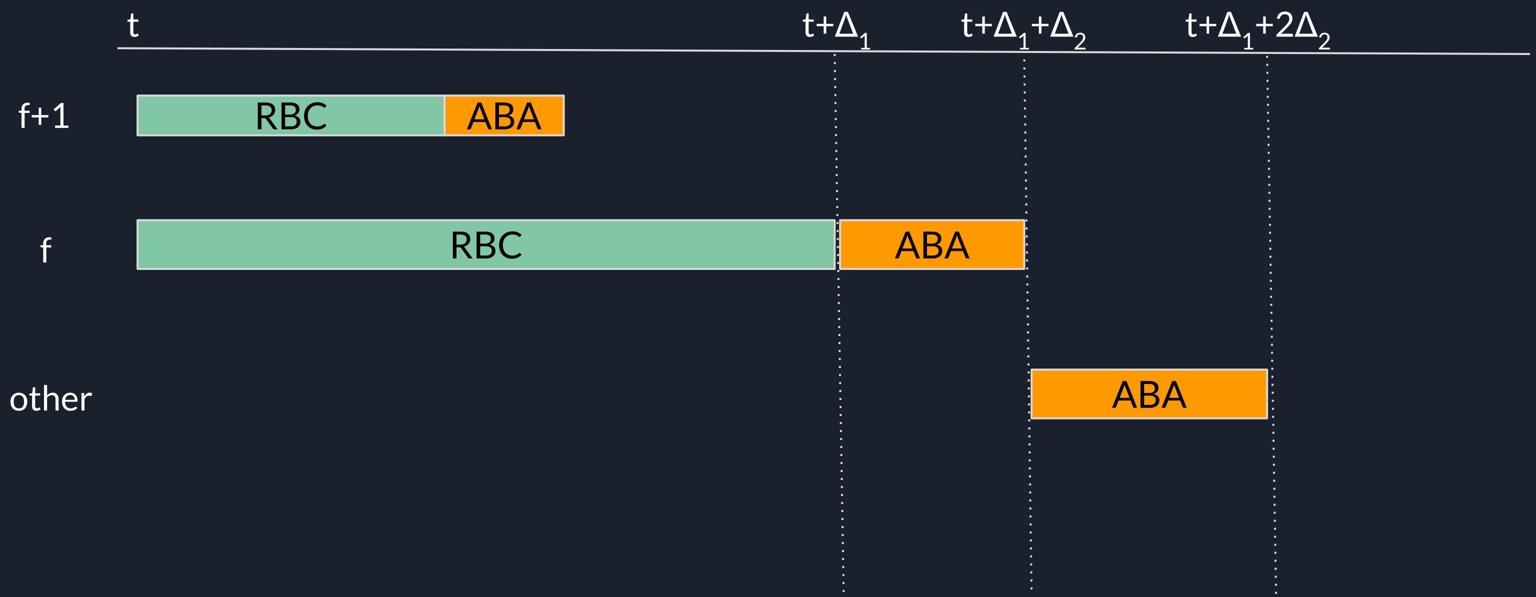
PACE vs BKR: WORST CASE SCENARIO

BKR



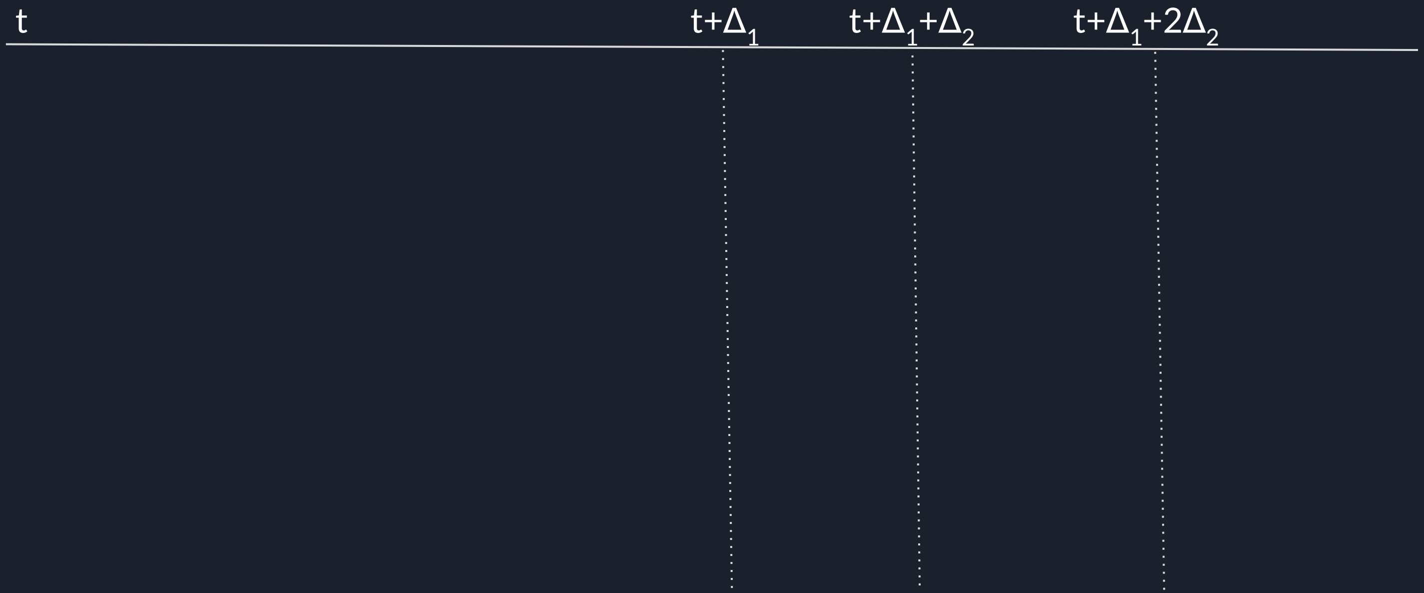
PACE vs BKR: WORST CASE SCENARIO

BKR



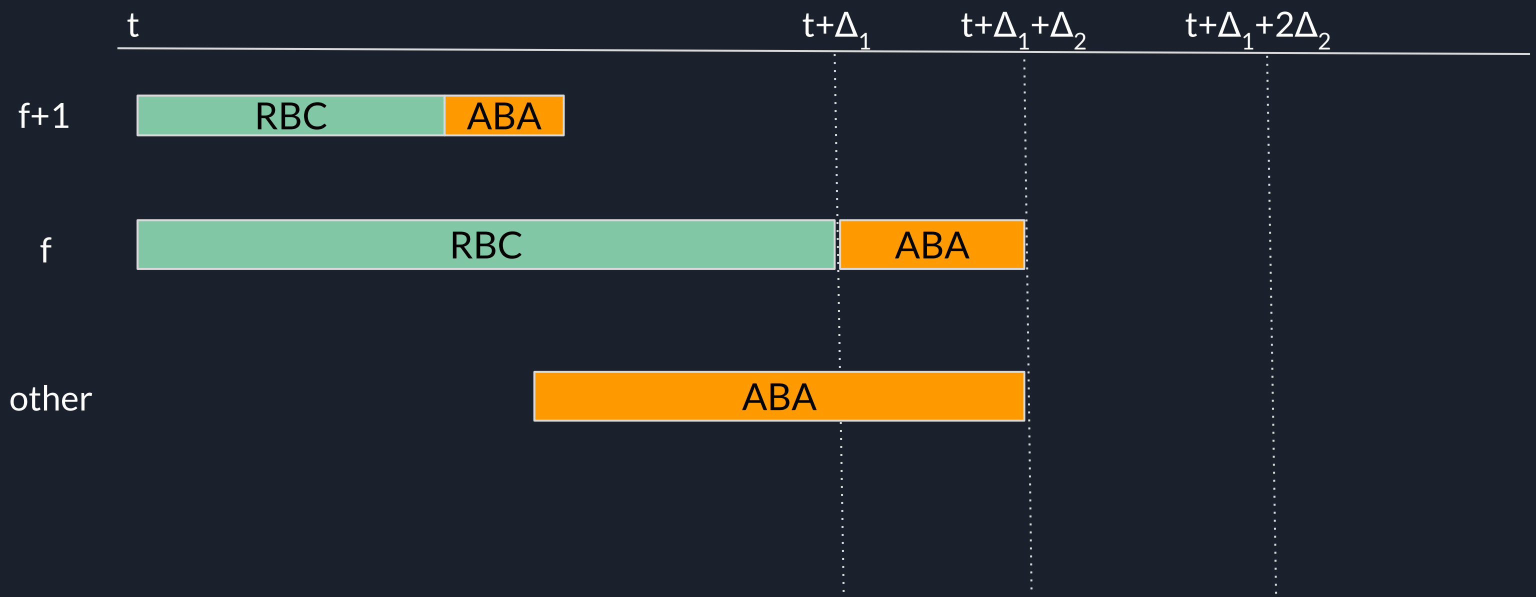
PACE vs BKR: WORST CASE SCENARIO

PACE

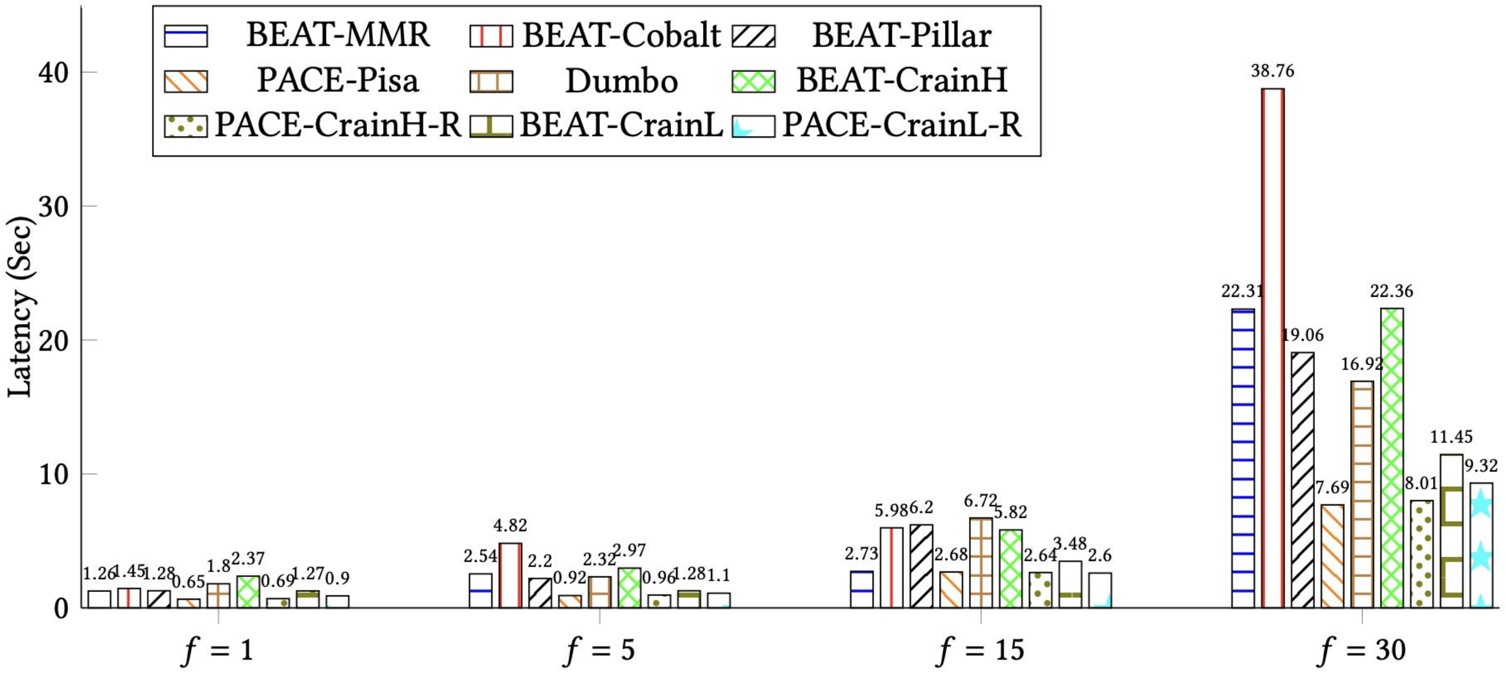


PACE vs BKR: WORST CASE SCENARIO

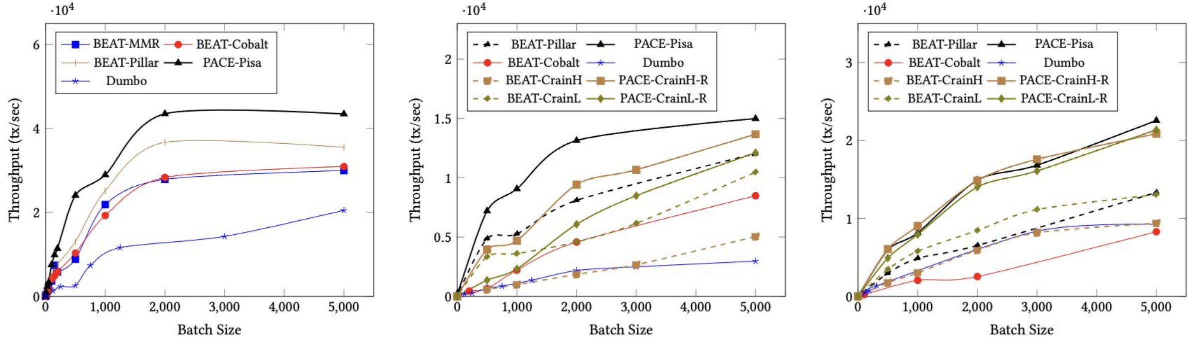
PACE



EVALUATION RESULTS



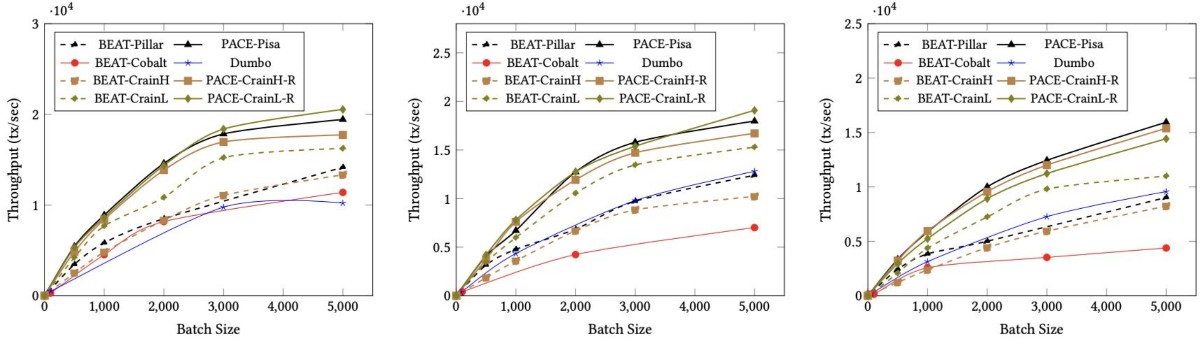
EVALUATION RESULTS



(a) Throughput for $f = 1$ where the replicas are located in the same DC.

(b) Throughput for $f = 1$ where the replicas are from 4 different DCs.

(c) Throughput for $f = 5$ where the replicas are from different DCs.



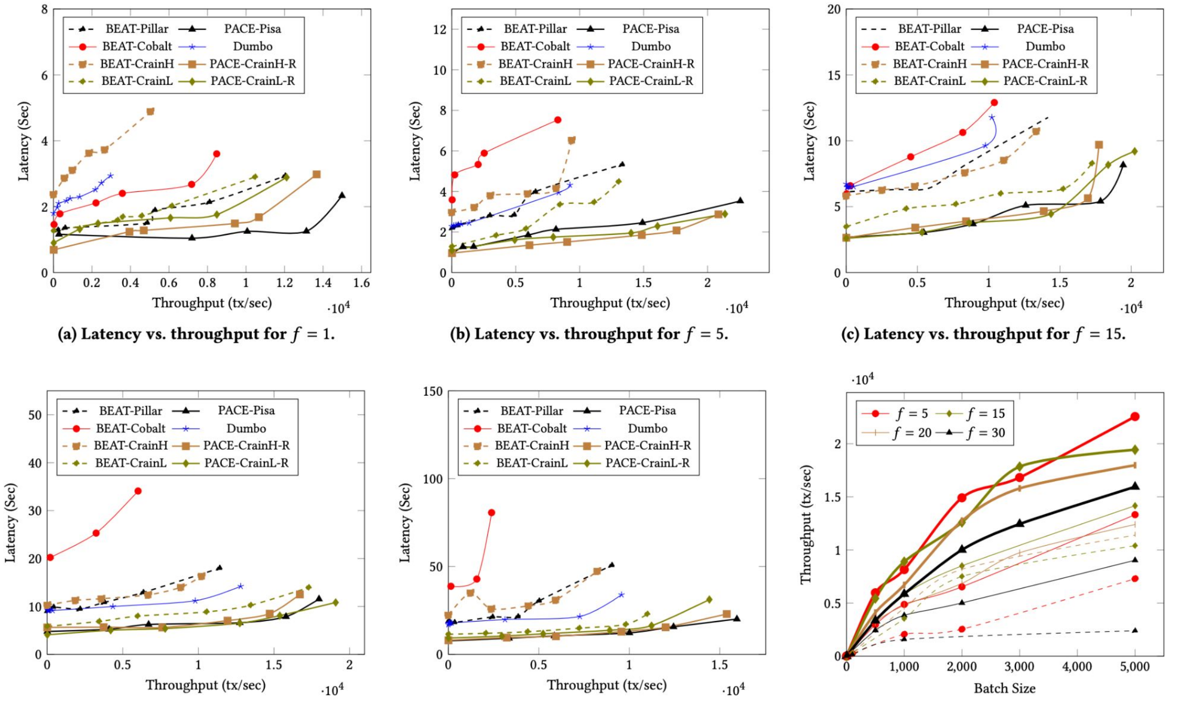
(d) Throughput for $f = 15$ where the replicas are from different DCs

(e) Throughput for $f = 20$ where the replicas are from different DCs.

(f) Throughput for $f = 30$ where the replicas are from different DCs.

Figure 9: Throughput of the protocols as f increases.

EVALUATION RESULTS



(a) Latency vs. throughput for $f = 1$.

(b) Latency vs. throughput for $f = 5$.

(c) Latency vs. throughput for $f = 15$.

(d) Latency vs. throughput for $f = 20$.

(e) Latency vs. throughput for $f = 30$.

(f) Scalability where f varies from 2 to 30. Thick lines denote the throughput of PACE-Pisa, thin lines denote that of BEAT-Pillar, and dashed lines denote that of BEAT-Cobalt.

Figure 10: Scalability results where replicas are from different DCs.

THANK YOU