Robust BFT Protocols

Sonia Ben Mokhtar, LIRIS, CNRS, Lyon

Joint work with

Pierre Louis Aublin**,** Grenoble university

Vivien Quéma, Grenoble INP

18/10/2013

Who am I?

- CNRS reseacher, LIRIS lab, DRIM research group
- Fault-tolerant distributed systems
  - Byzantine fault tolerance
    - State machine replication (BFT)(e.g., robust BFT[ICDCS'13])
  - Byzantine fault detection
    - Accountability (e.g., accountable mobile systems, performance issues in accountable systems[ongoing])
  - Robustness against selfish behavior
    - Game theory (e.g., RR spam filtering[SRDS'10], RR anonymous communication[ICDCS'13], RR live streaming[ongoing])

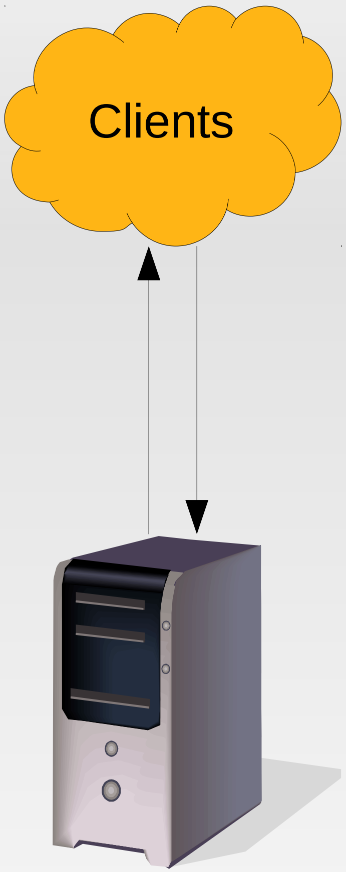
Who am I?

- CNRS reseacher, LIRIS lab, DRIM research group.
- Fault-tolerant distributed systems
  - Byzantine fault tolerance
    - State machine replication (BFT)(e.g., robust BFT[ICDCS'13])
  - Byzantine fault detection
    - Accountability (e.g., accountable mobile systems, performance issues in accountable systems[ongoing])
  - Robustness against selfish behavior
    - Game theory (e.g., RR spam filtering[SRDS'10], RR anonymous communication[ICDCS'13], RR live streaming[ongoing])
- → Privacy (mobile systems, reputation/recommender systems, systems enforcing accountability)

Outline

- **What is BFT?**
- BFT under attack: the *robustness* problem
- Existing robust BFT protocols
- Can we do better?

State machine replication



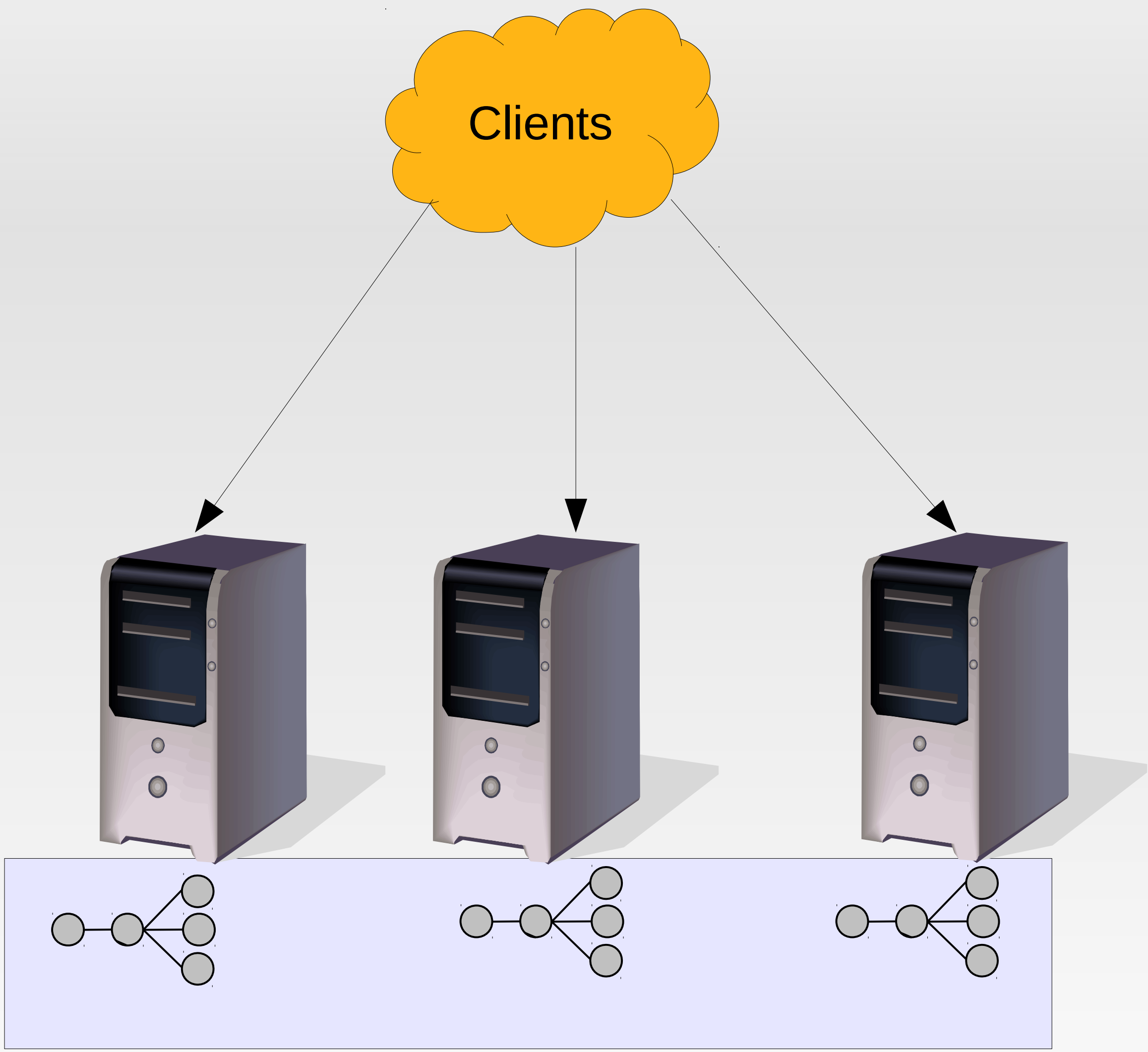
State machine replication

State machine replication

State machine replication

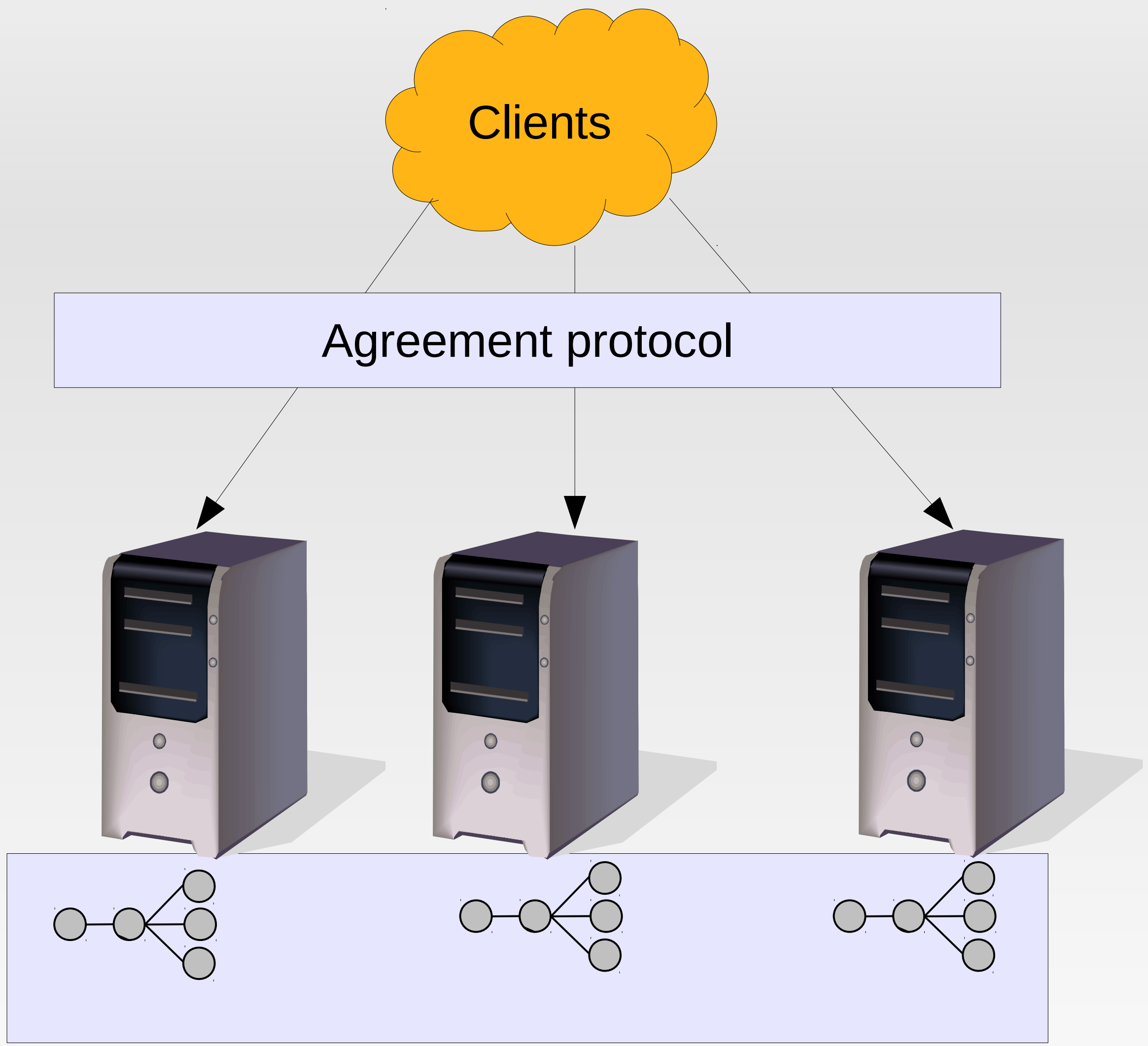
(1) Place **copies** of a **deterministic state machine** on multiple, independent servers.

State machine replication



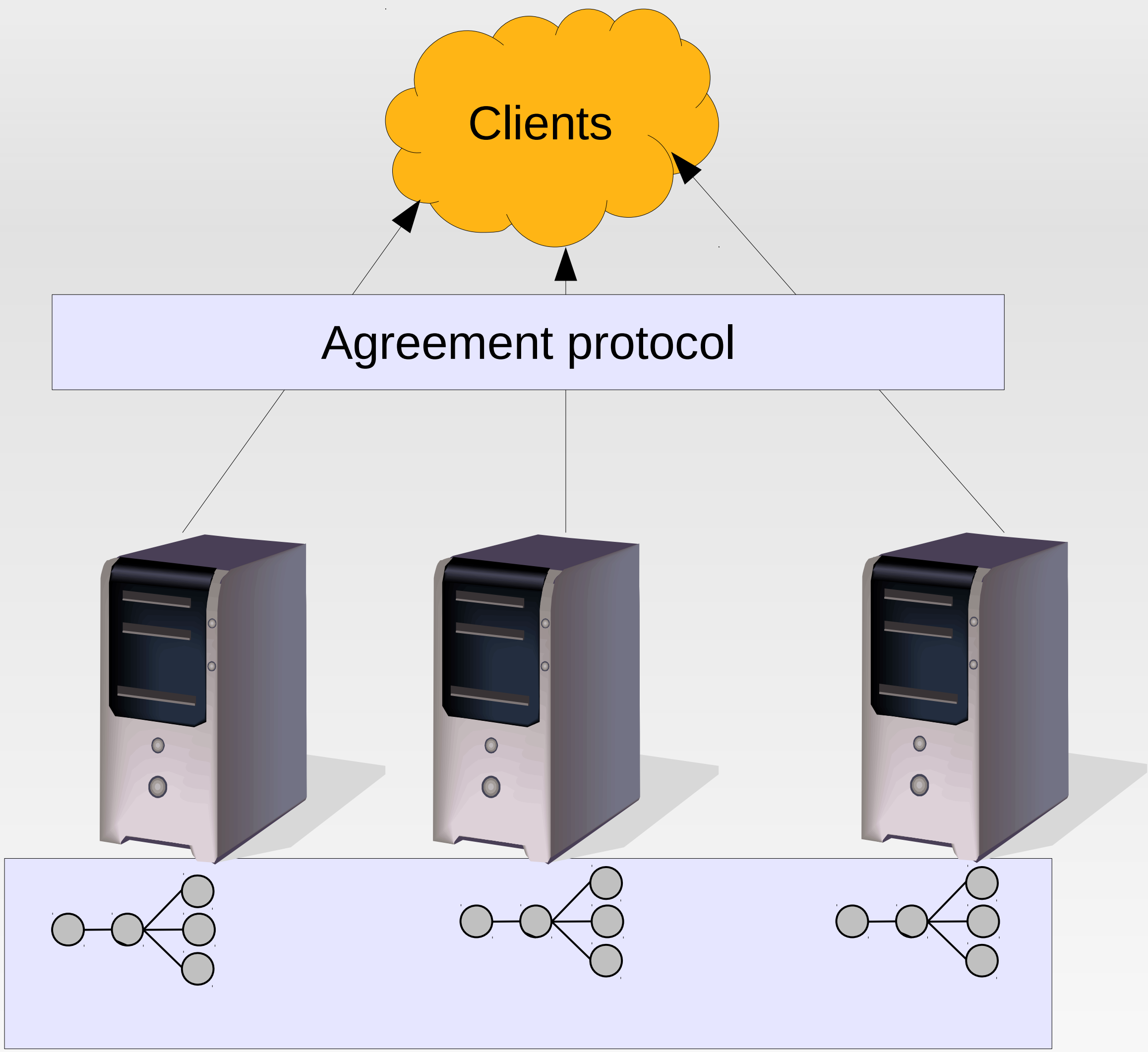
(2) Receive **client requests** (inputs to the state machine).

State machine replication



(3) Define an **ordering** for the inputs and **execute** them in the chosen order on each server.

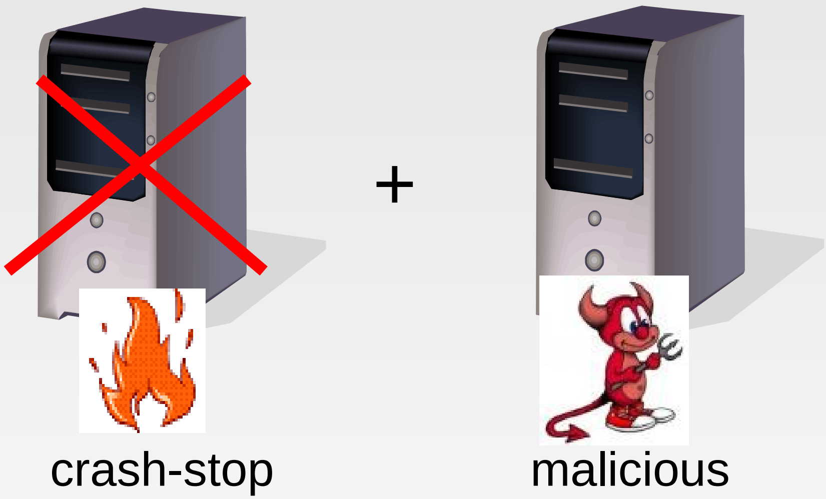
State machine replication



(4) Respond to clients with the output from the state machine.

BFT state machine replication

- BFT = Byzantine Fault Tolerance
- The term Byzantine dates back to the seminal paper by Lamport, Shostak, Pease: The Byzantine Generals Problem, ACM TPLS, 1982.
- Byzantine failure = arbitrary failure



- BFT state machine replication = state machine replication that tolerates Byzantine failures

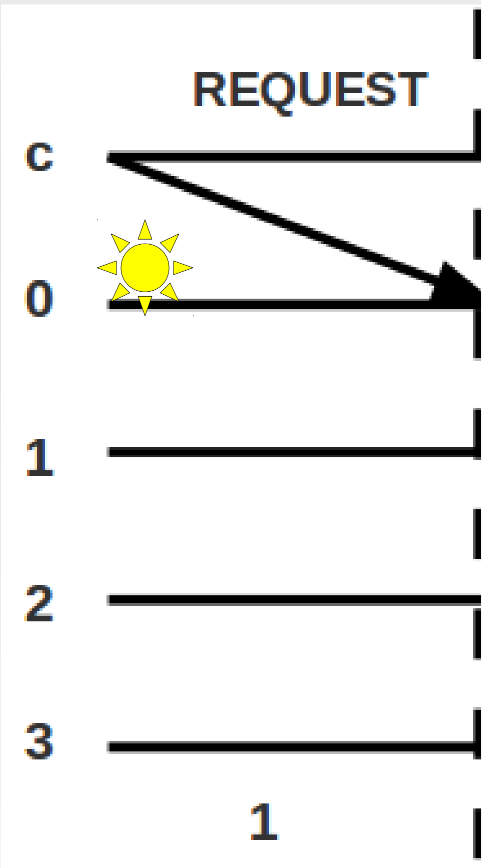
BFT evolution

- Lamport, Shostak, Pease: The Byzantine generals problem, 1982
- **Castro, Liskov: Practical BFT [OSDI'99]**
- BFT in 2011 (a decade+ later)
  - Efficient BFT: Q/U [SOSP'05], HQ [OSDI'06], Zyzzyva [SOSP'07], Chain and Quorum [EuroSys'10]
  - Cheap BFT: zz [Umass Eurosys'11]
  - **Robust BFT**: **Aardvark [NSDI'09], Spinning [SRDS'09], Prime [DSN'08], RBFT[ICDCS'13]**

BFT with an example: PBFT

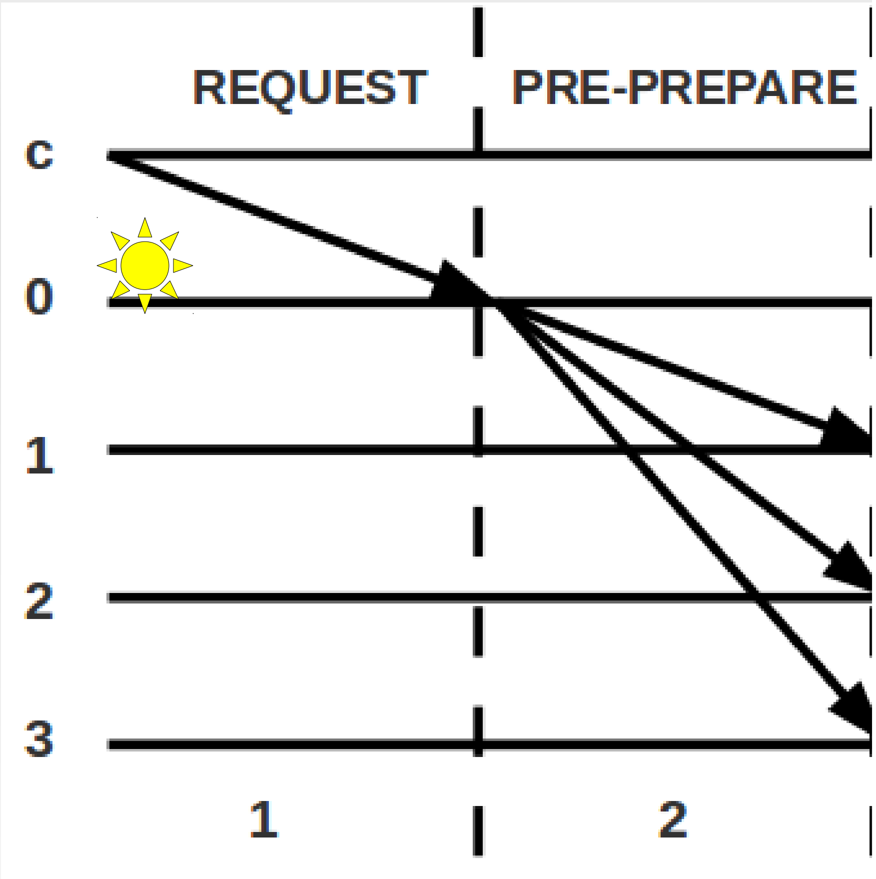
- Message-passing with unreliable communication links
- Byzantine faults
  - Any number of clients
  - Less than 1/3 of replicas are faulty (optimal)
- Cryptographic techniques cannot be violated
- Eventual synchrony

PBFT: protocol steps



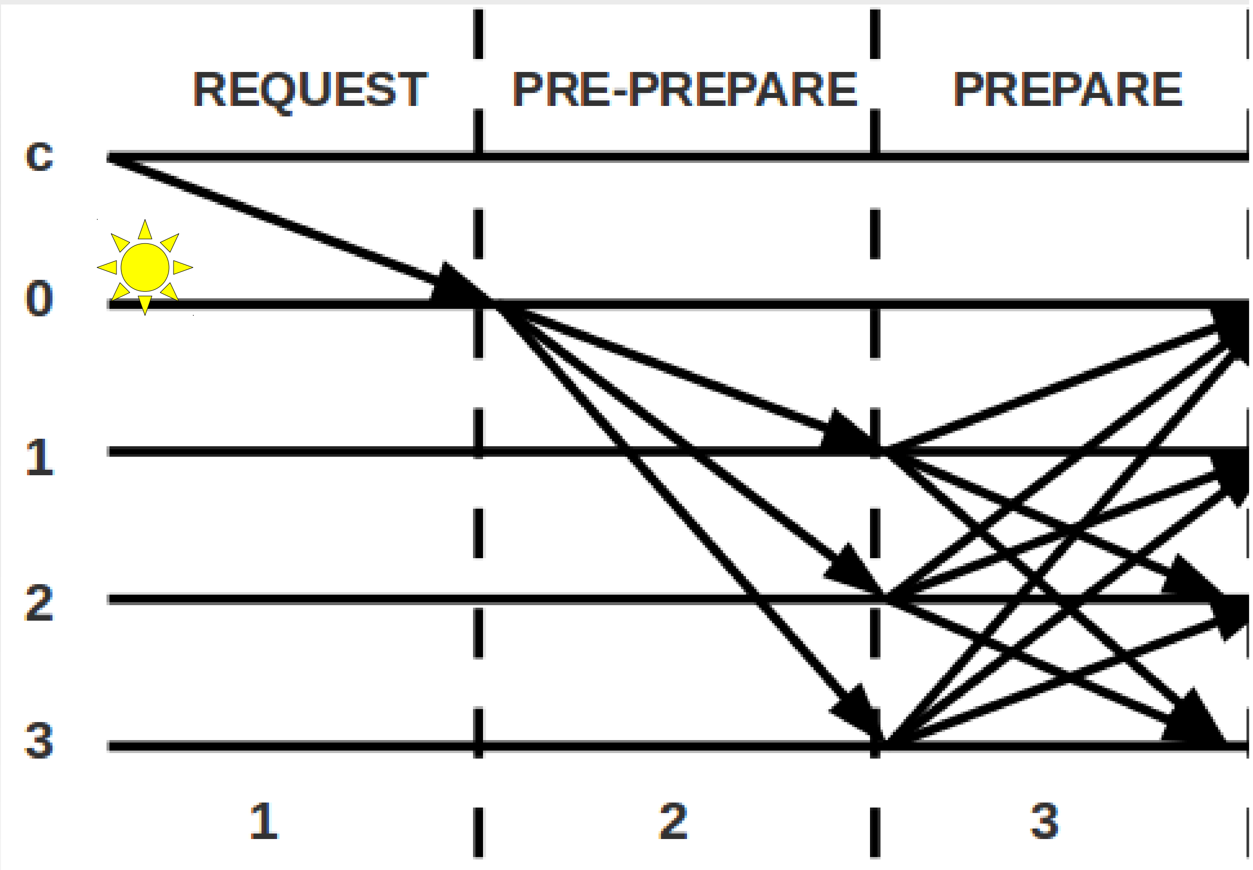
Client sends a request to the primary

PBFT: protocol steps



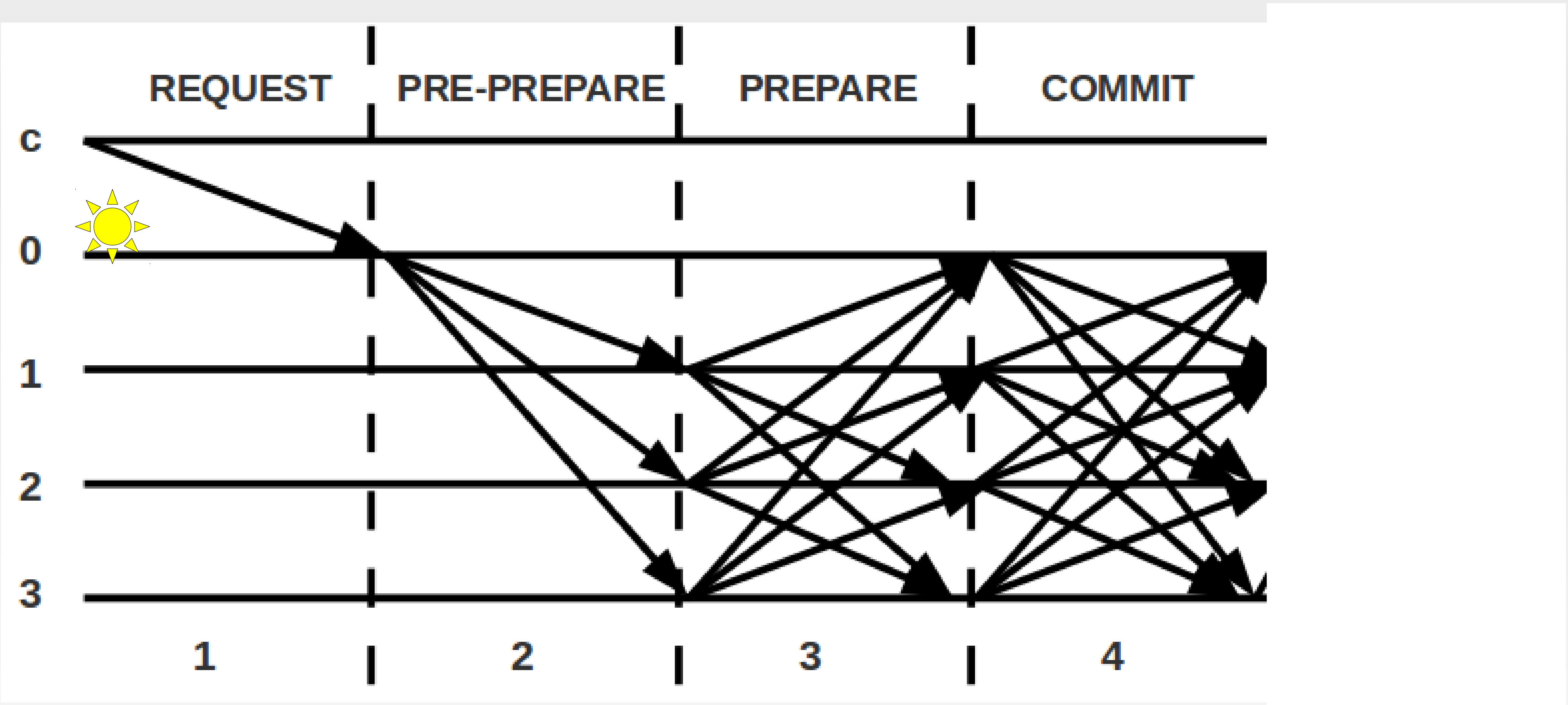
The primary assigns a seqno to the request

PBFT: protocol steps



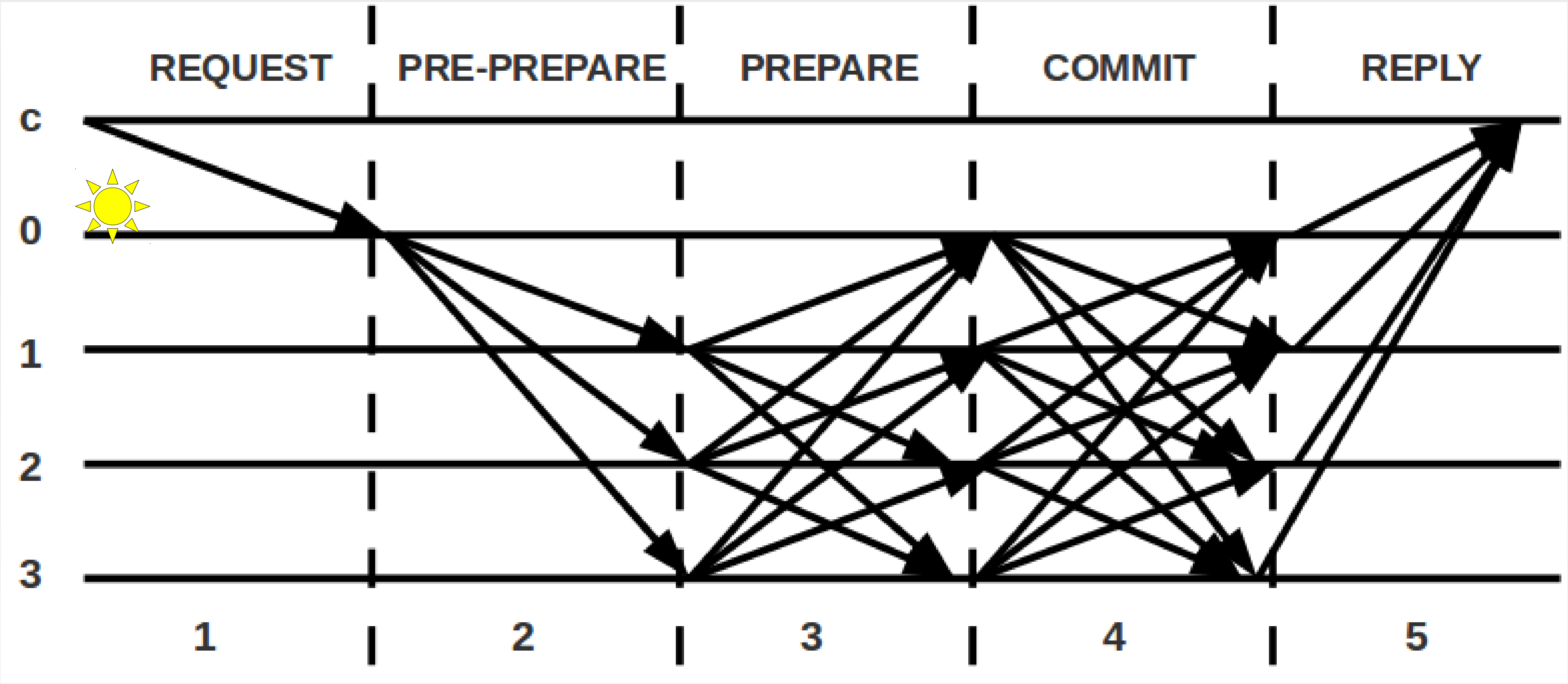
Replicas agree on the assigned seqno

PBFT: protocol steps



Replicas know 2f+1 replicas that agreed on the proposed seqno

PBFT: protocol steps



Replicas execute the request and reply to the client

Outline

- What is BFT?
- **BFT under attack: the *robustness* problem**
- Existing robust BFT protocols
- Can we do better?

BFT under attack: the robustness problem

”*BFT protocols do not tolerate Byzantine faults* ***very well***” [NSDI'09]

System	Peak throughput (req/s)	Throughput under attack (req/s)
PBFT	61710	0
Q/U	23850	0
HQ	7629	N/A
Zyzzyva	65999	0

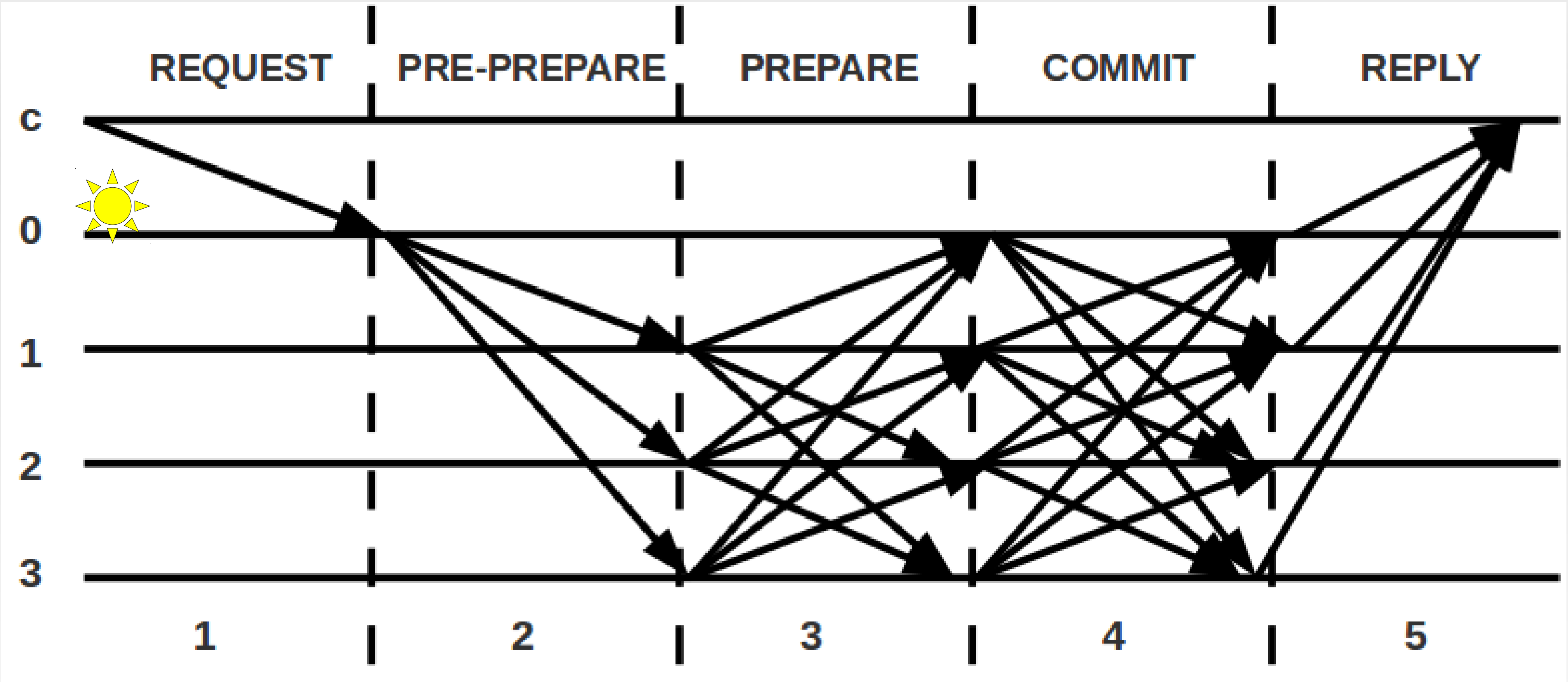
Outline

- What is BFT?
- BFT under attack: the *robustness* problem
- **Existing robust BFT protocols**
- Can we do better?

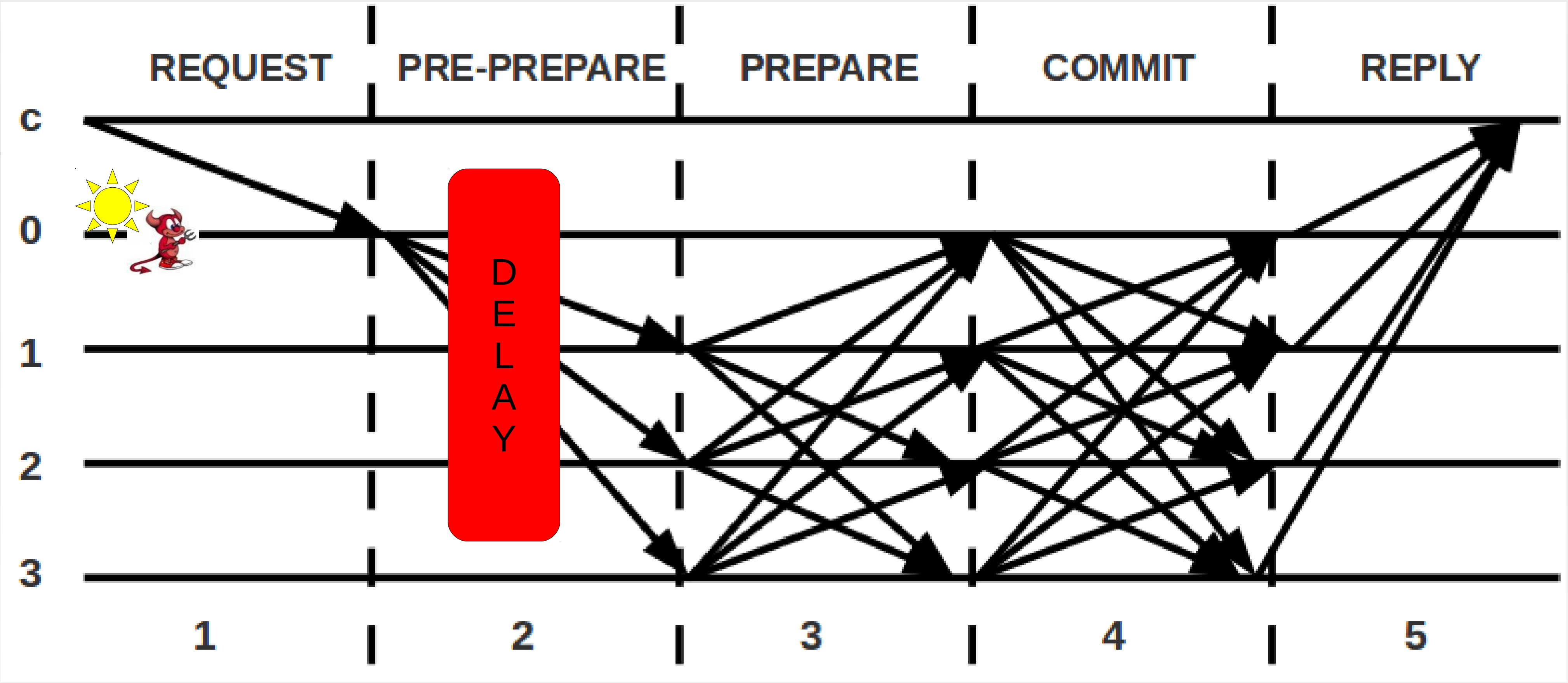
Robust BFT state machine replication

- Guarantees a lower bound on performance during uncivil executions
  - Uncivil executions:
    - Synchronous network
    - Up to *f* servers and any number of clients are Byzantine
  - Lower bound:
    - k% of the theoretical maximum (with the same workload)
    - k should be as high as possible

Malicious primary

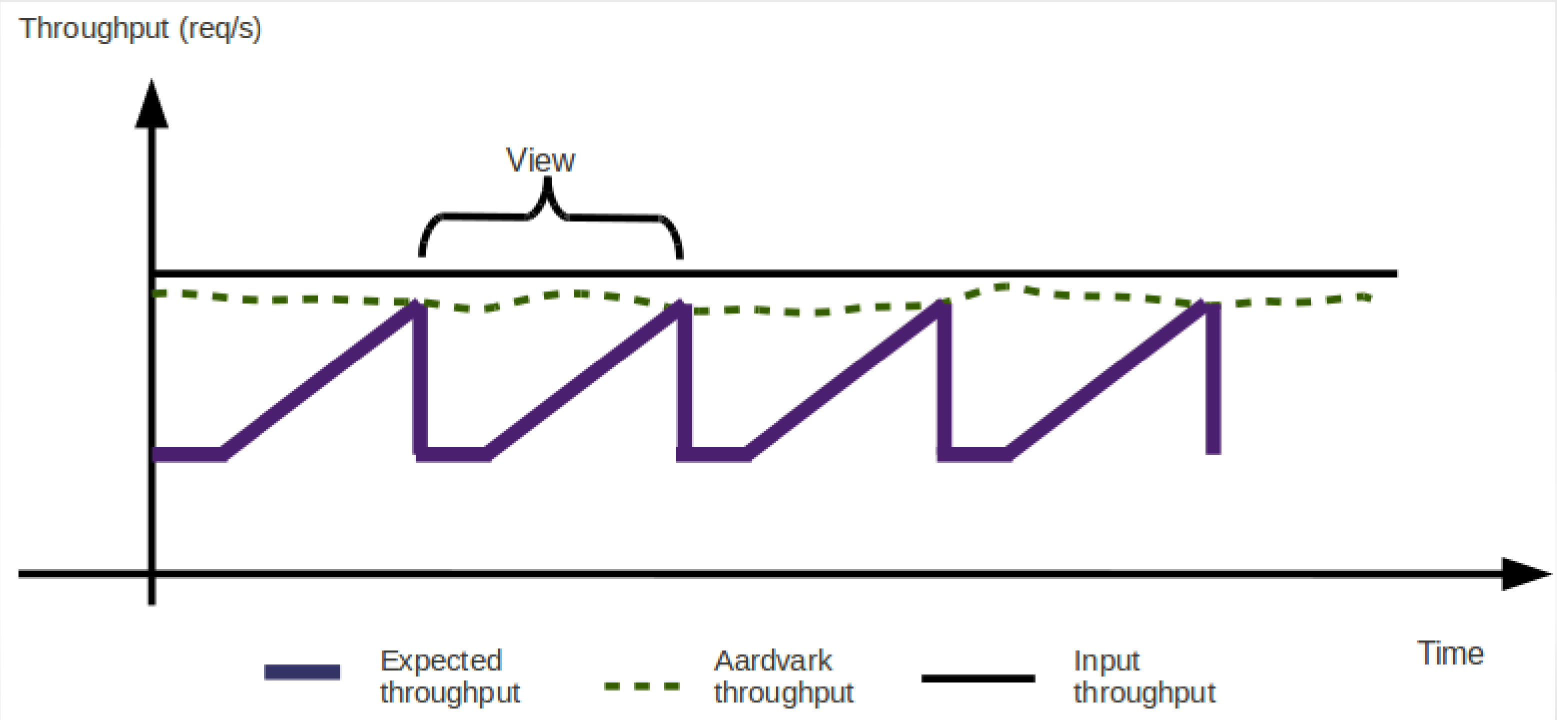


Malicious primary



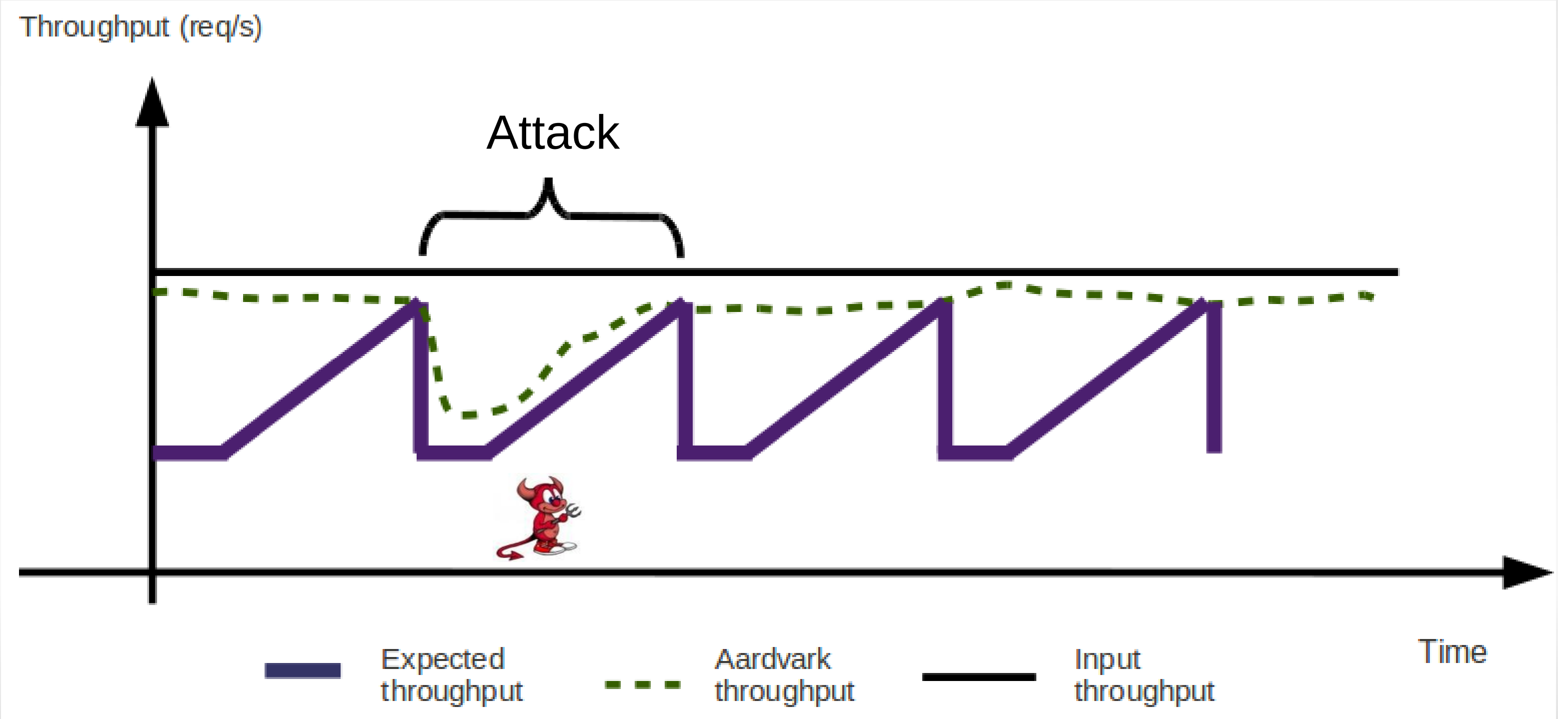
Aardvark [NSDI'09]

- Principle: Regular primary changes
  - Increasing throughput expectations
  - Monitoring of the current throughput
  - Change the primary when the current throughput is below the expected thourhgput

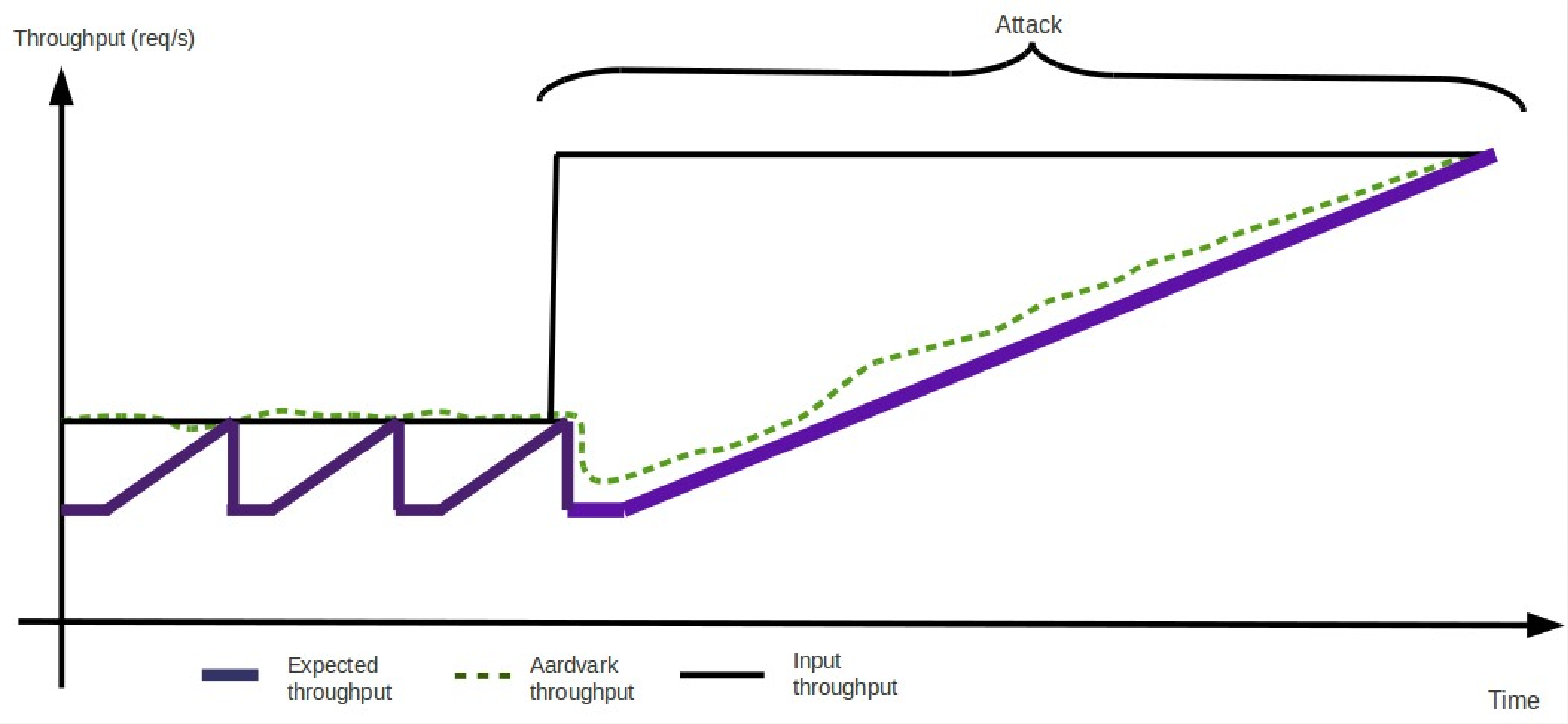


Aardvark

- A malicious primary is bounded in:
  - The delay it can add to requests
  - The amount of time it acts as a primary
  - Only works under constant load

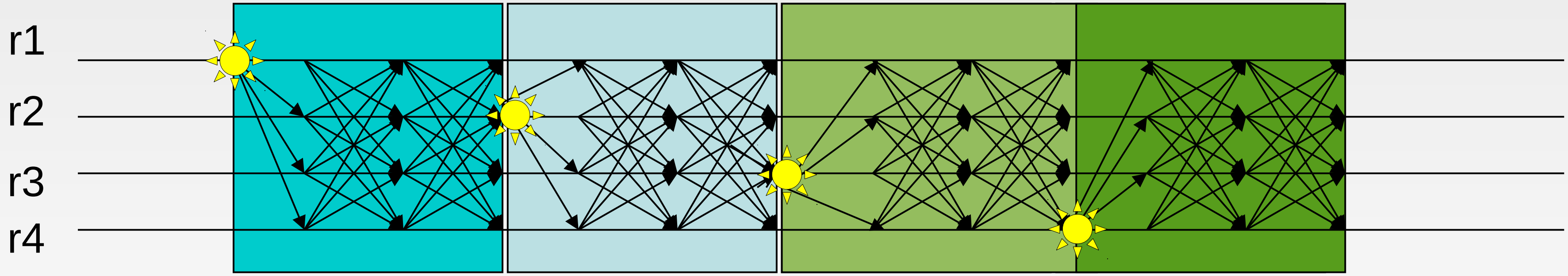


Aardvark under fluctuating load



Spinning [SRDS'09]

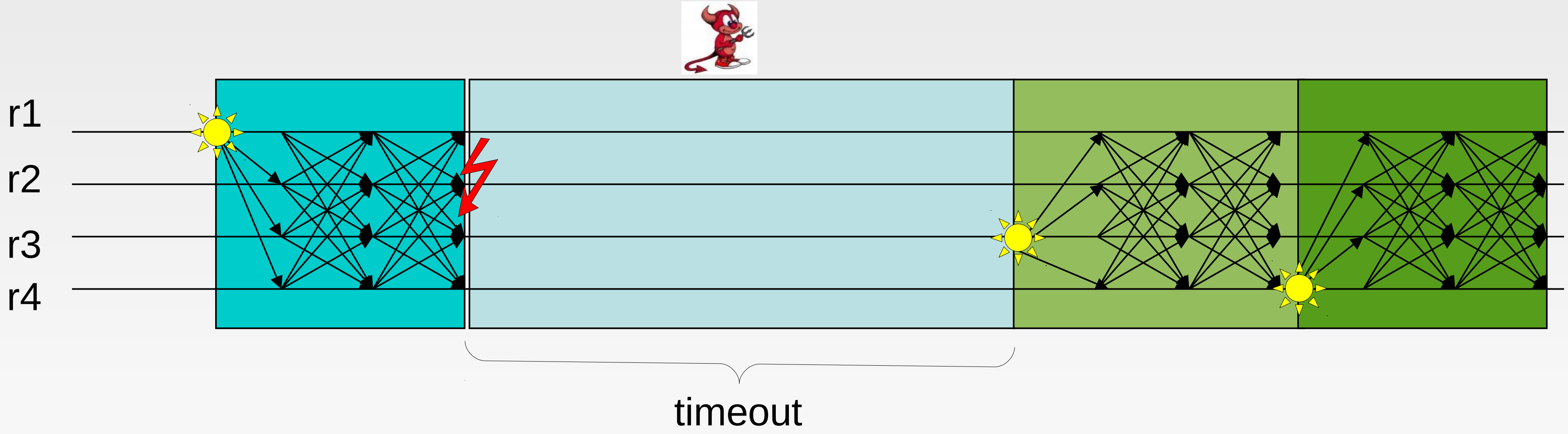
- Principle:
  - Each primary orders a fixed number of requests
  - The primary is changed if no request is ordered before a timeout



Spinning

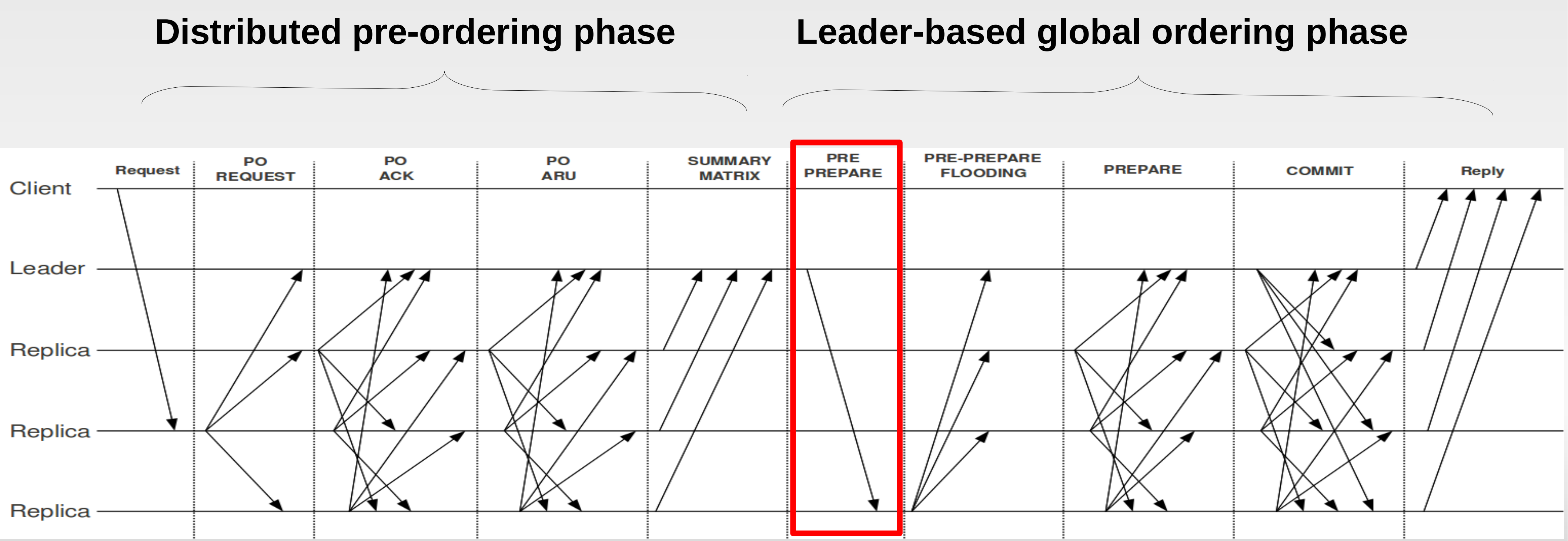
- Spinning throughput with a malicious primary that delays client requests by up to timeout:

$$1/(1+F*timeout)*t_{peak}$$



Prime [DSN'08]

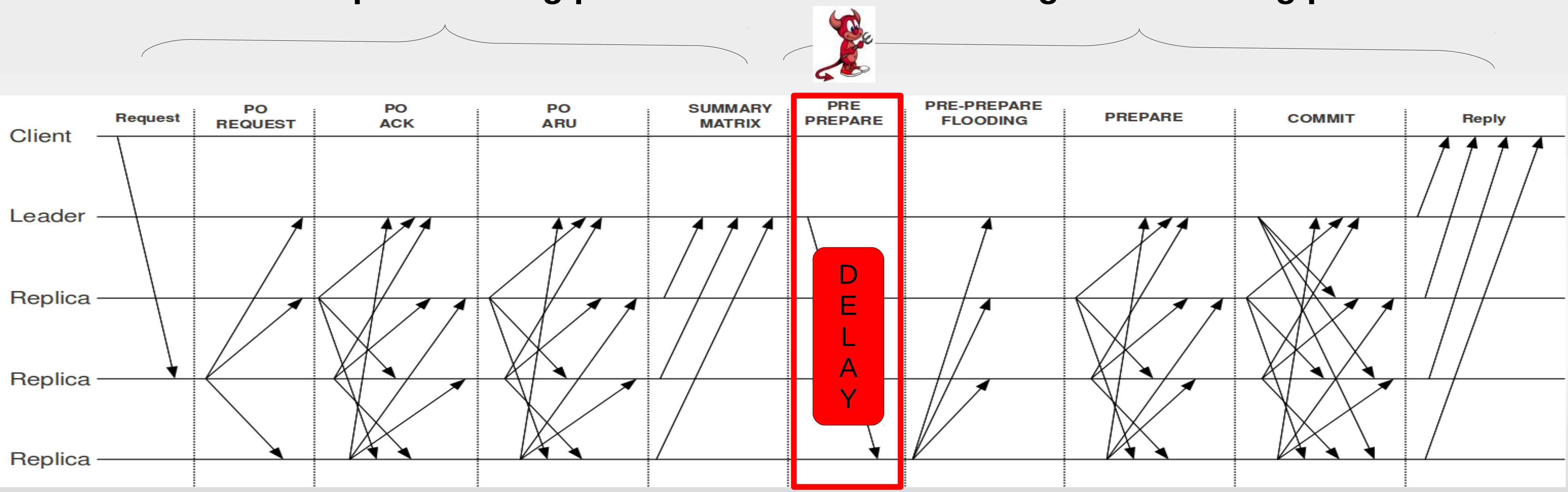
- Principle:
  - The primary periodically sends messages of the same size in the network (fixed workload)
  - Replicas monitor the primary



Prime

- The latency of any update initiated by a correct client is bounded
  - Only if the network guarantees bounded variance

Distributed pre-ordering phase **Leader-based global ordering phase**



Outline

- What is BFT?
- BFT under attack: the *robustness* problem
- Existing robust BFT protocols
- **Can we do better?**

What is wrong with existing protocols?

- The primary is a single point of failure
  - Aardvark and Prime: monitor the primary
  - Spinning: bound the time spent with a faulty primary

- Robustness conditions are strong:
  - Aardvark: constant load
  - Prime: bounded variance

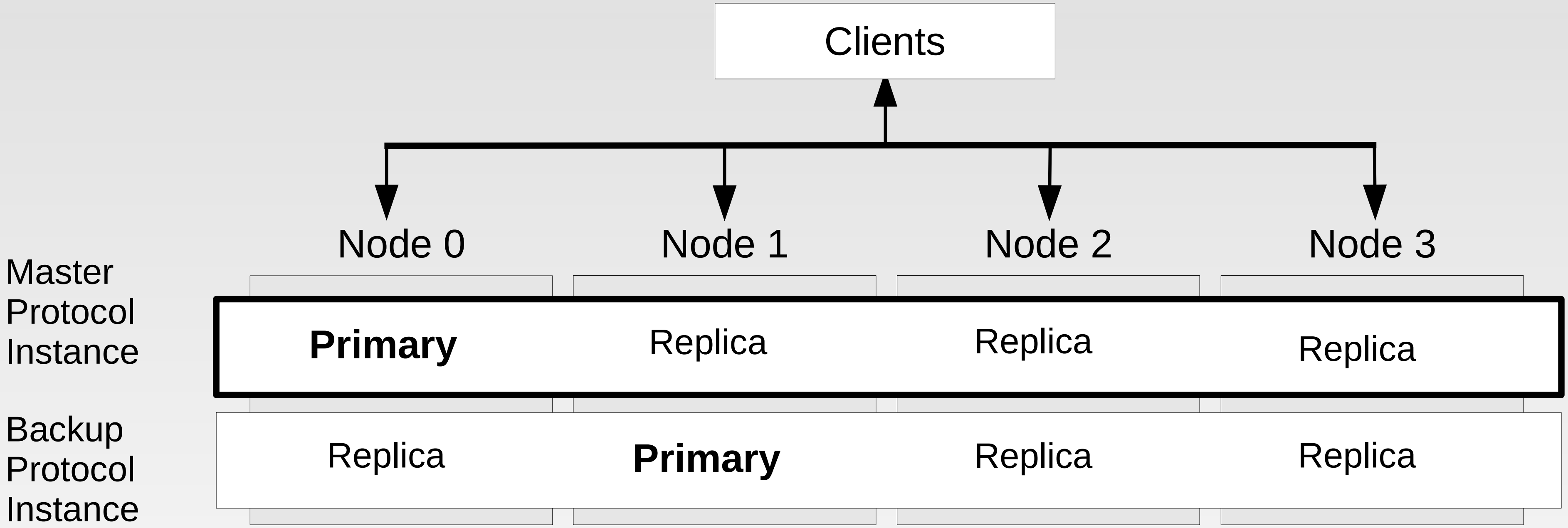
What is wrong with existing protocols?

- The primary is a single point of failure
  - Aardvark and Prime: monitor the primary
  - Spinning: bound the time spent with a faulty primary

- Robustness conditions are strong:
  - Aardvark: constant load
  - Prime: bounded variance

Question: Can we run multiple instances of a protocol simultaneously?

The RBFT protocol

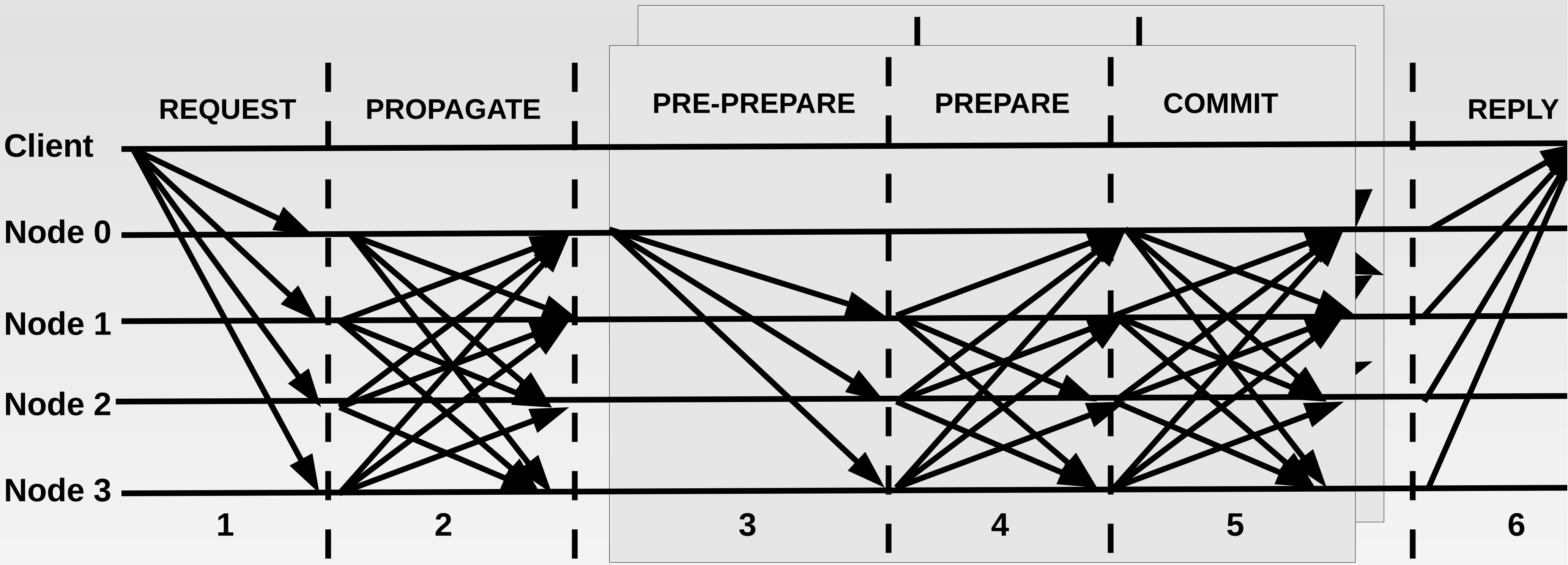


The RBFT protocol

	Node 0	Node 1	Node 2	Node 3
Master Protocol Instance	**~~Primary~~** ➡	**Primary**		
Backup Protocol Instance		**~~Primary~~** ➡	**Primary**	

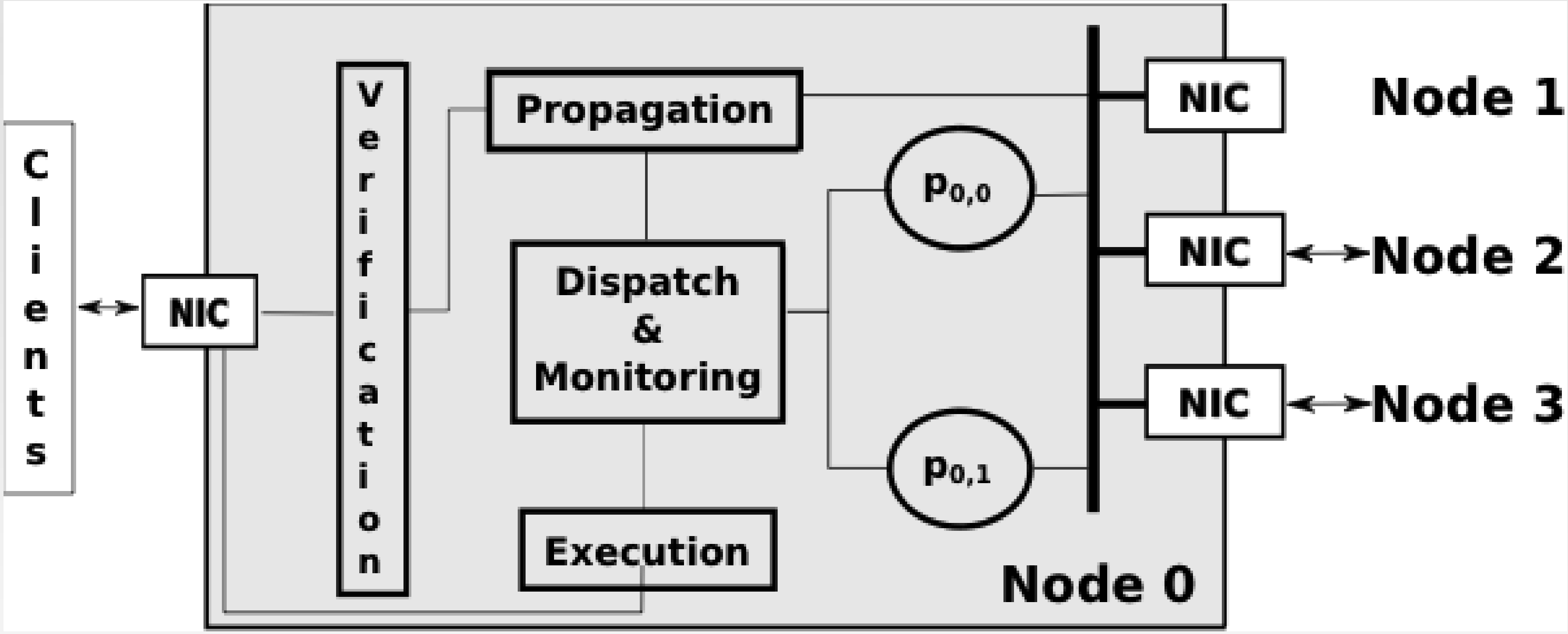
➡ *Primary change*

RBFT Redundant Agreement

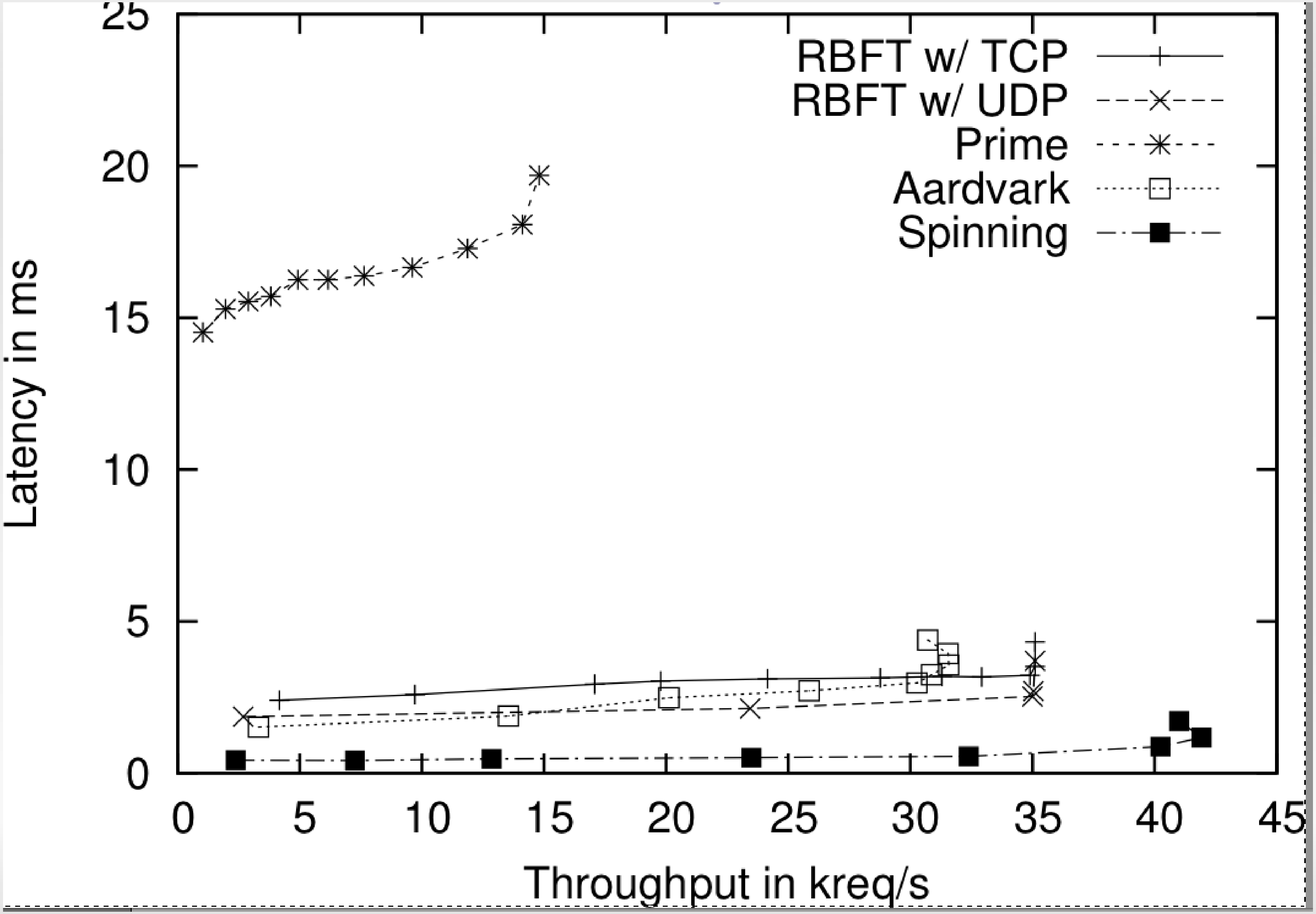


Redundant agreement performed by the replicas

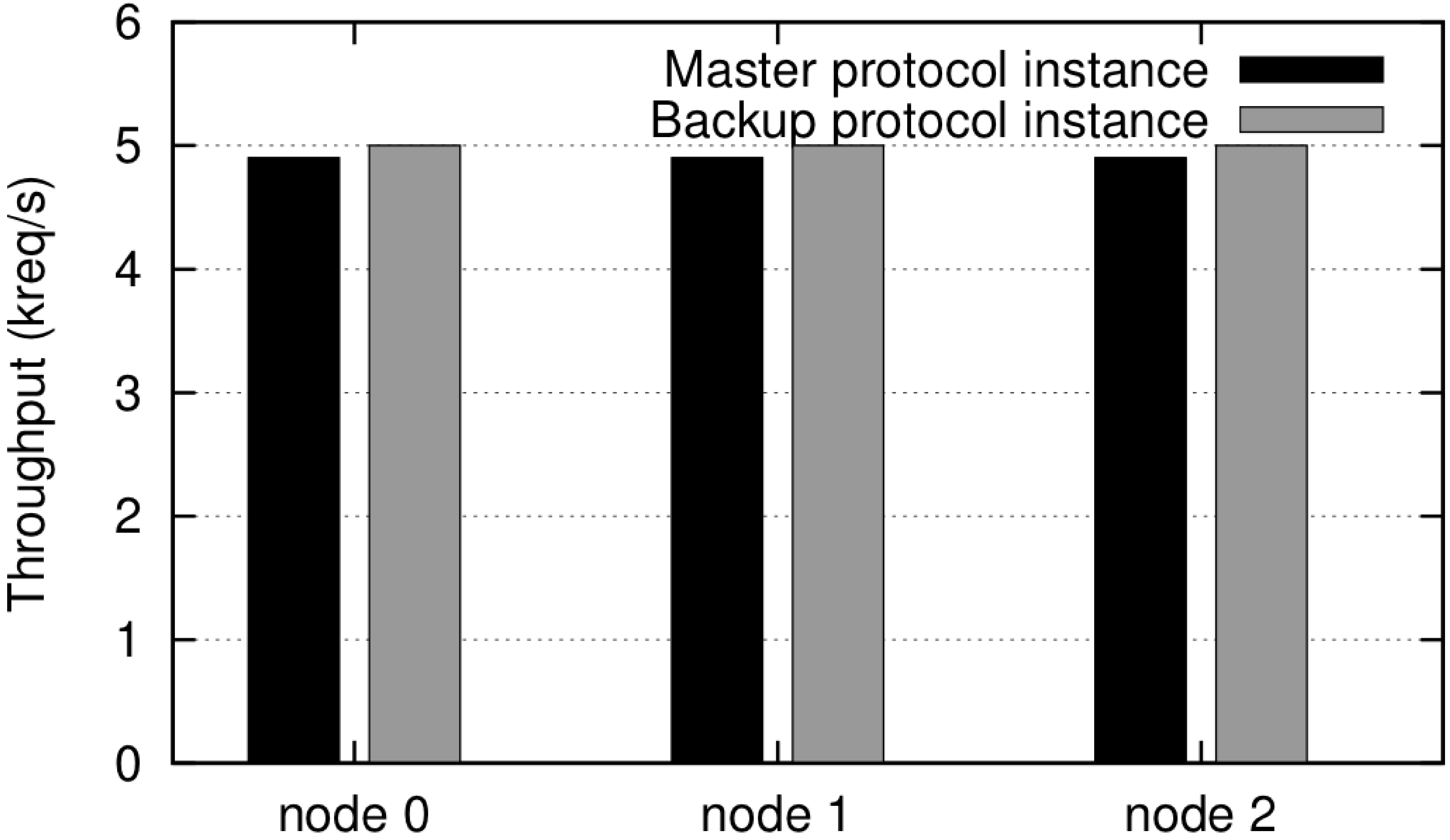
RBFT Node Design



RBFT Performance



RBFT under attack



Conclusion

- We need BFT protocols (to tolerate arbitrary faults)
- Current BFT protocols are either:
  - Robust (e.g., RBFT) or
  - Efficient (e.g., Chain, Quorum)
- Future work
  - Dynamic switching: can we design a BFT protocol that smartly combines robustness and efficiency?

Thank you!