The Bedrock of Byzantine Fault Tolerance

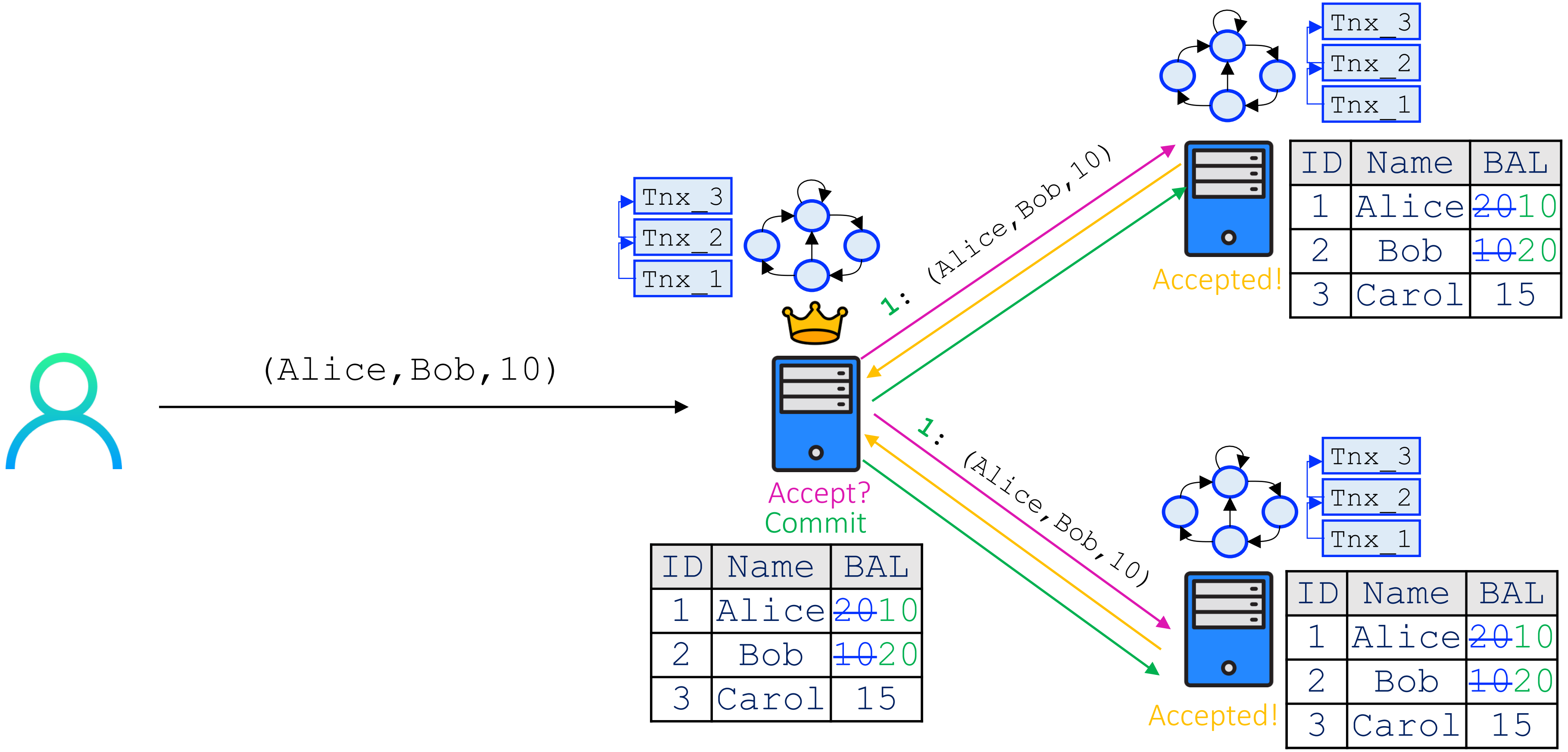
A Unified Platform for BFT Protocols Analysis, Implementation, and Experimentation

Mohammad Javad Amiri[1], Chenyuan Wu[2], Divyakant Agrawal[3], Amr El Abbadi[3], Boon Thau Loo[2], Mohammad Sadoghi[4]

[1]Stony Brook University, [2]UPenn, [3]UC Santa Barbara, [4]UC Davis

nsdi '24

Distributed transaction processing

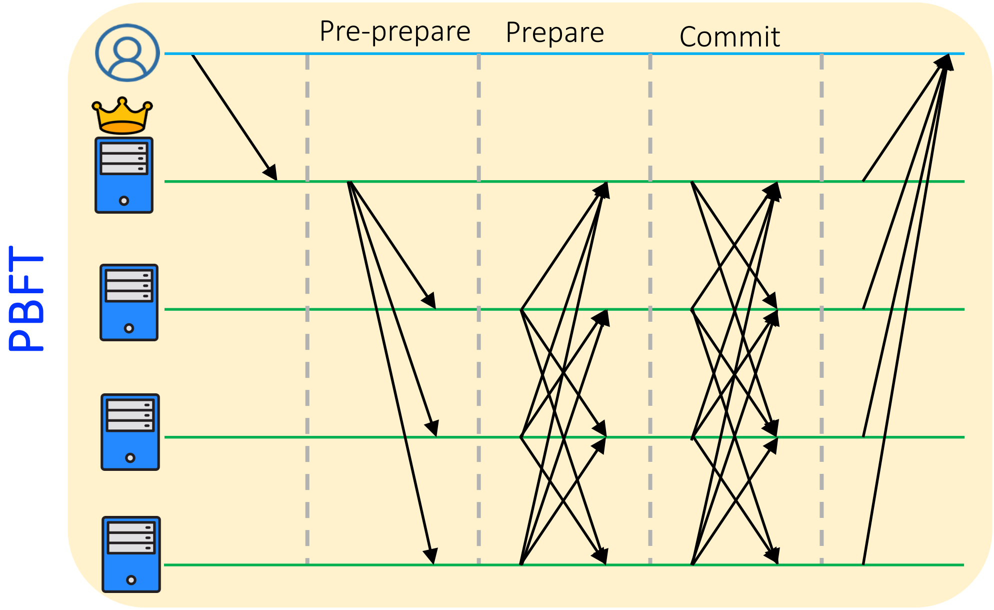


State Machine Replication: a replicated service whose state is mirrored across different deterministic replicas

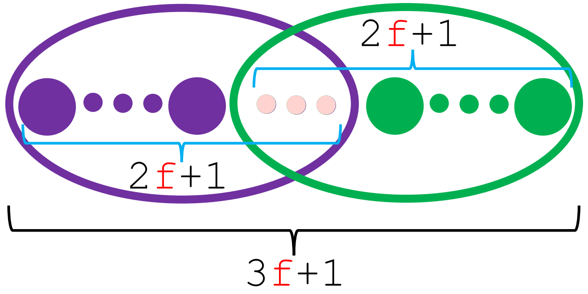
- Assign order to each client request in the global service history and execute it in that order

Byzantine fault-tolerant protocol: PBFT

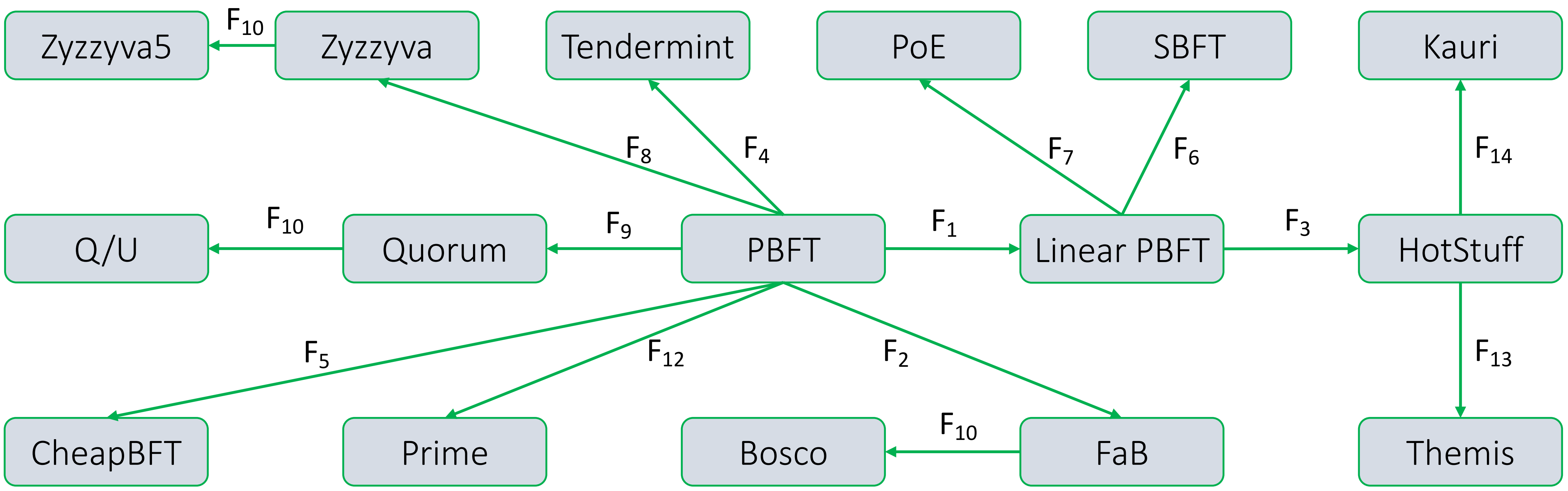
Nodes can fail arbitrarily, including deviating from the protocol



At Most f
Byzantine Failures



BFT protocols landscape



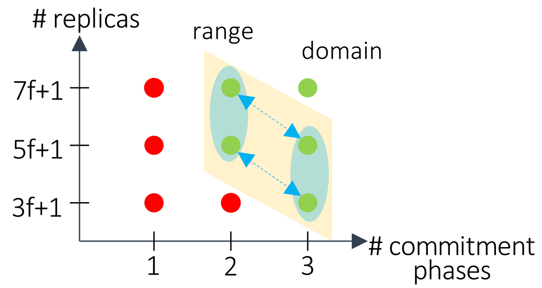
What protocol best fits our needs?

Analysis **Implementation** **Experimentation**

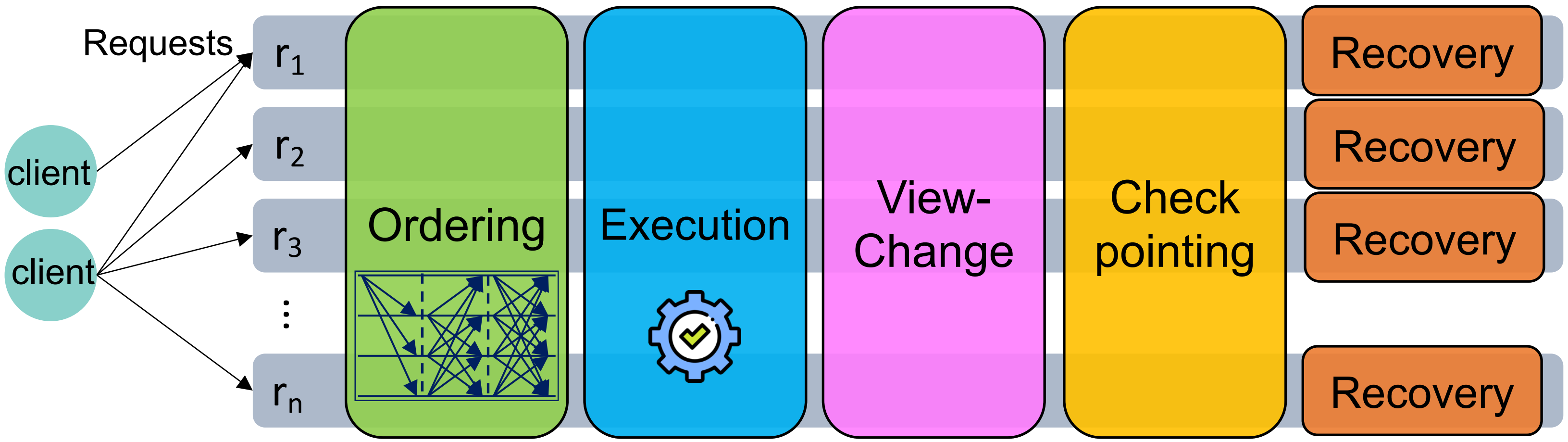
nsdi '24

BFT protocols design space and design dimensions

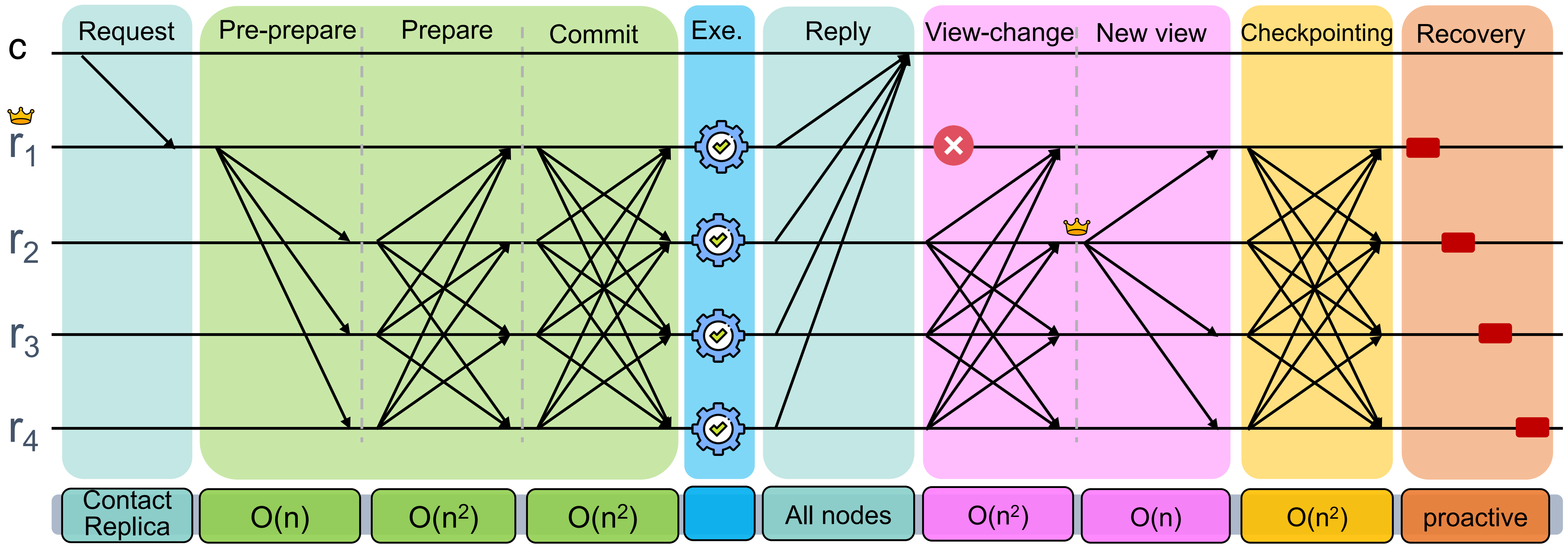
- Design space
    - A set of dimensions to analyze BFT protocols
- Design choices
    - Trade-offs between dimensions
    - A set of one-to-one functions, each maps protocols in its domain to protocols in its range
- Focus on partially synchronous BFT protocols



Different stages of replicas in a BFT protocol



PBFT



Request
Pre-prepare
Prepare
Commit
Exe.
Reply
View-change
New view
Checkpointing
Recovery
c
r1
r2
r3
r4
Contact Replica
O(n)
O(n²)
O(n²)
All nodes
O(n²)
O(n)
O(n²)
proactive

Design space of BFT protocols

Protocol structure

P1. Commitment strategy
P2. Number of commitment phases
P3. View-change
P4. Checkpointing
P5. Recovery
P6. Types of clients

Environmental Settings

E1. Number of replicas
E2. Communication topology
E3. Authentication
E4. Responsiveness, synchronization, and timers

Quality of Service

Q1. Order-fairness
Q2. Load balancing

Performance Optimization

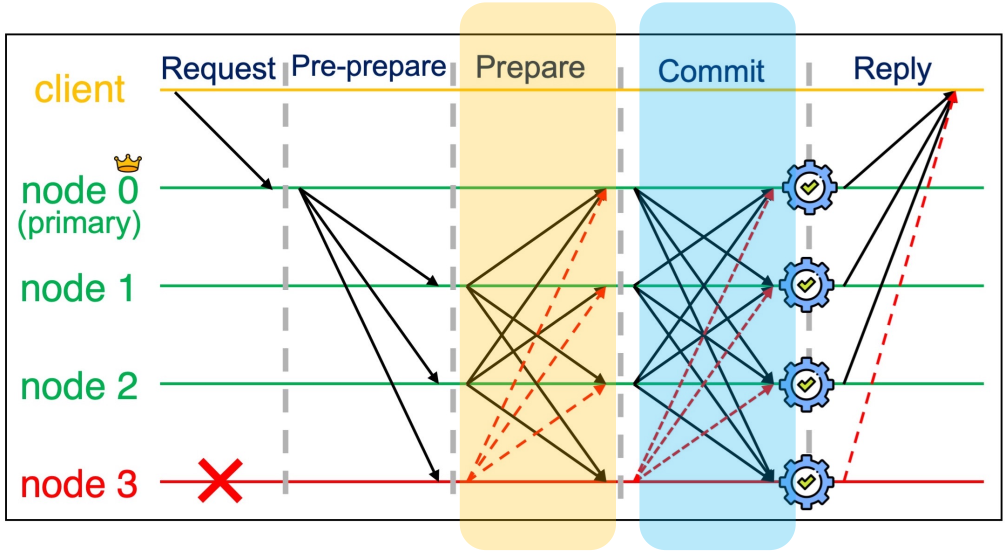
O1. Out-of-order processing
O2. Request pipelining
O3. Parallel ordering
O4. Parallel execution
O5. Read-only requests processing
O6. Separating ordering and execution
O7. Trusted hardware
O8. Request/reply dissemination

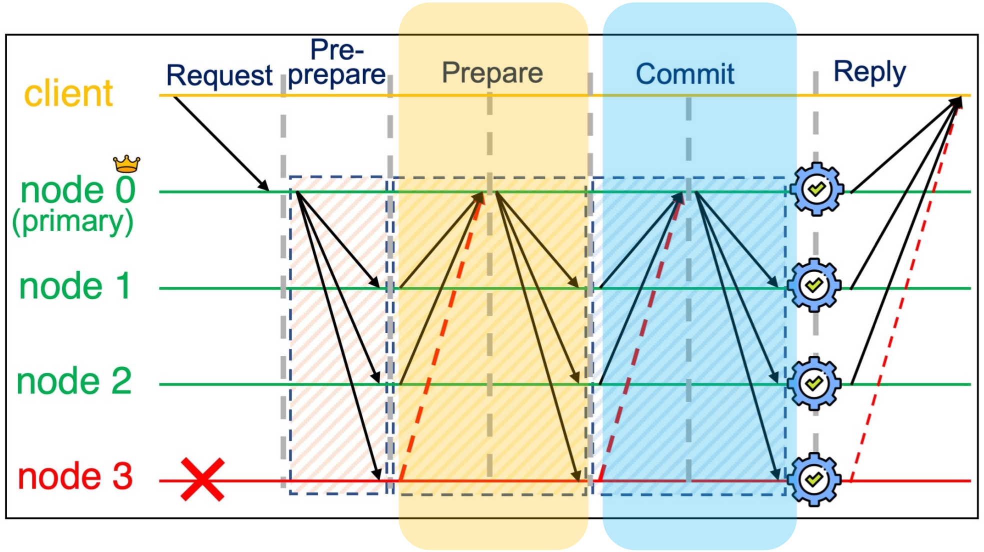
Design choices

1. Linearization
2. Phase reduction through redundancy
3. Leader rotation
4. Non-responsive leader rotation
5. Optimistic replica reduction
6. Optimistic phase reduction
7. Speculative phase reduction
8. Speculative execution
9. Optimistic conflict-free
10. Resilience
11. Authentication
12. Robust
13. Fair
14. Tree-based LoadBalancer

Design choice 1: Linearization

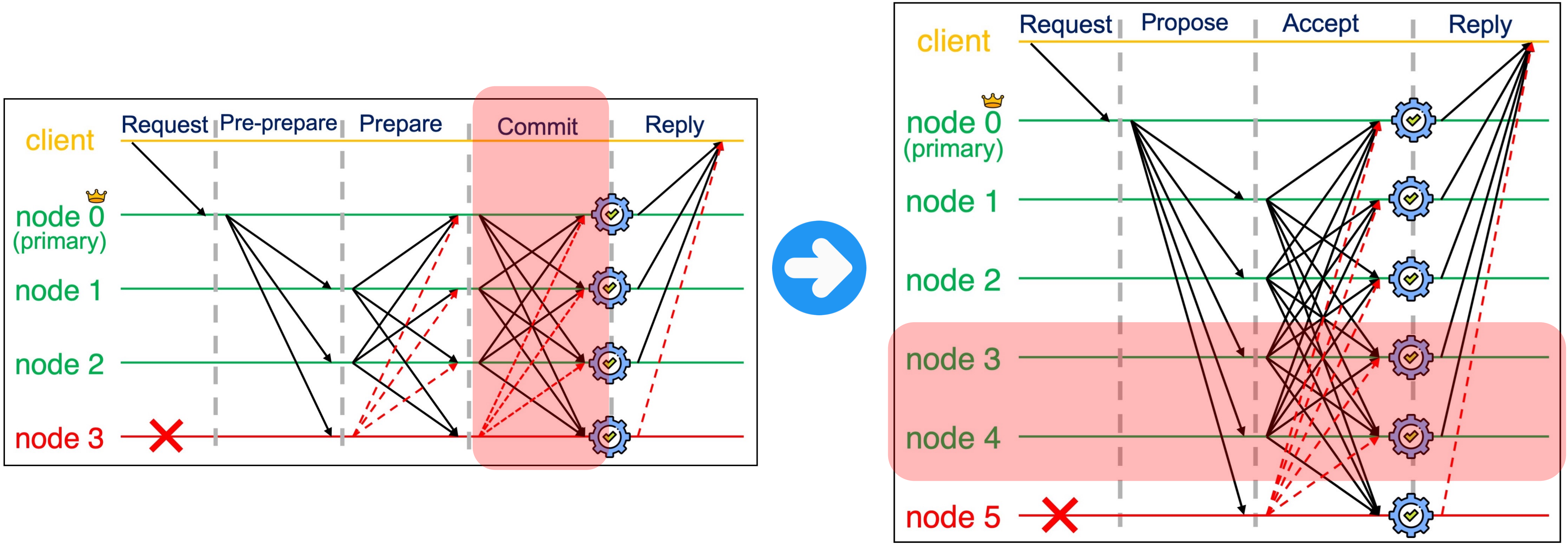
- Trade-off between communication topology and communication phases.
  - Linear PBFT
    - The collector needs to send a certificate of having received the required signatures.





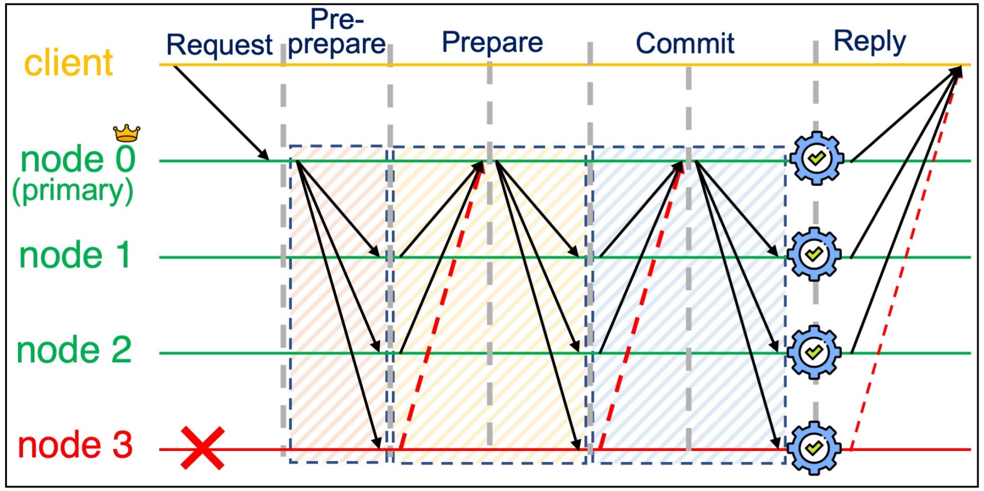
Design choice 2: Phase reduction through redundancy

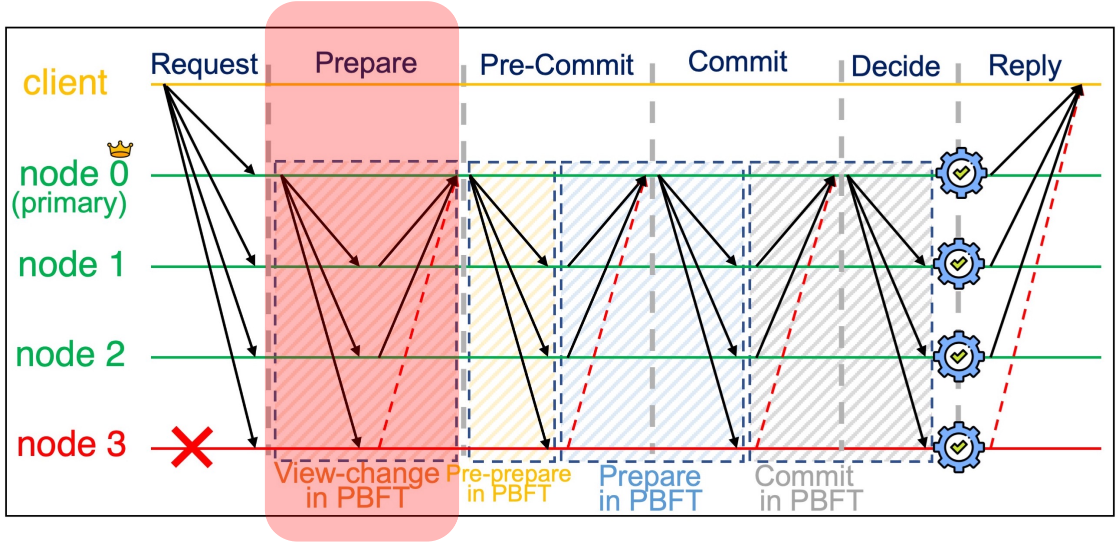
- Trade-off between the number of ordering phases and the number of replicas
  - FaB



Design choice 3: Leader rotation

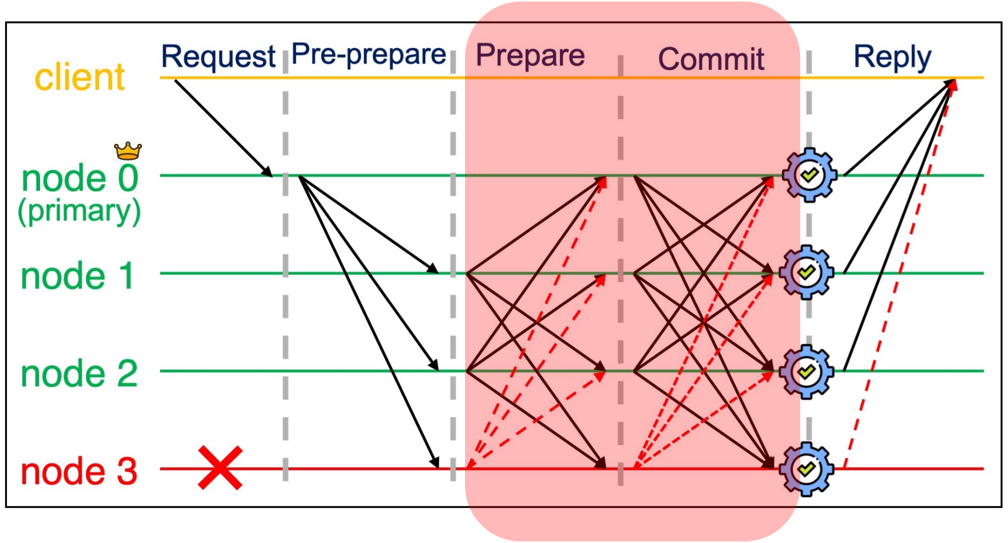
- Replace the stable leader with the rotating leader mechanism by adding one phase
  - HotStuff

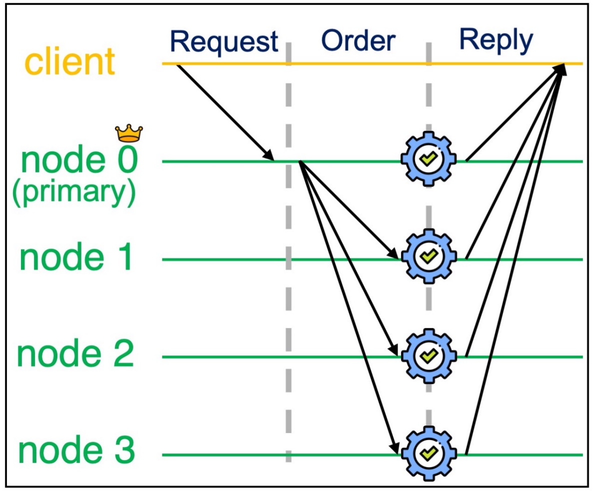




Design choice 8: Speculative execution

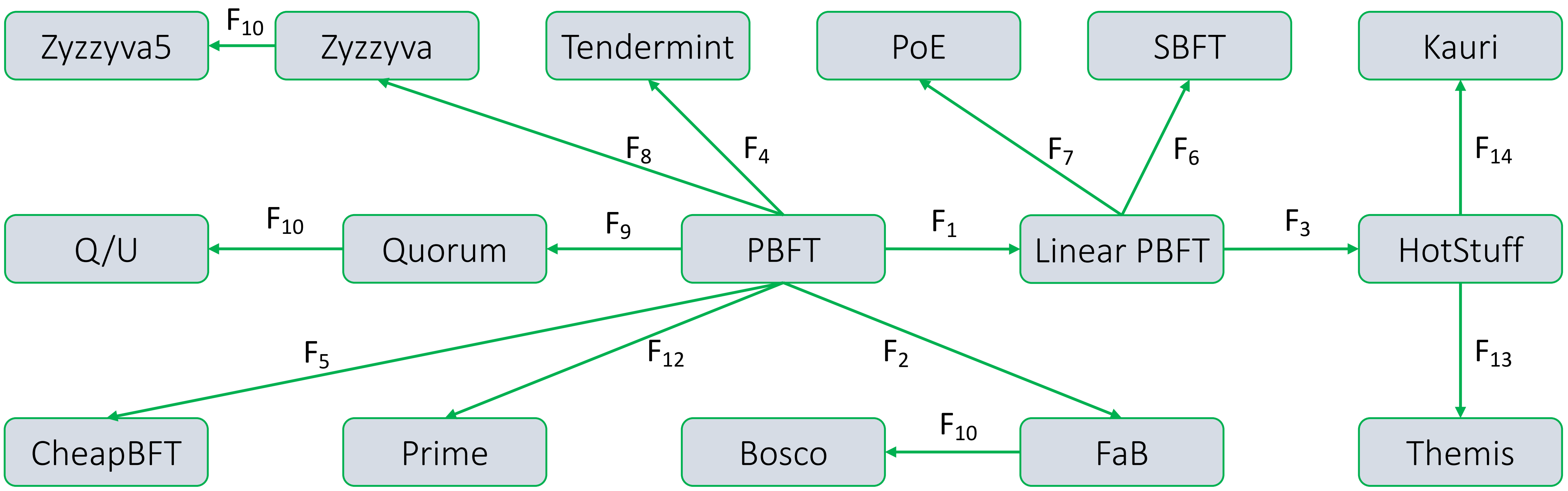
- Eliminate the prepare and commit phases while optimistically assuming that all replicas are non-faulty
  - Zyzzyva





Derivation of protocols from PBFT using design choices

1. Linearization
2. Phase reduction through redundancy
3. Leader rotation
4. Non-responsive leader rotation
5. Optimistic replica reduction
6. Optimistic phase reduction
7. Speculative phase reduction
8. Speculative execution
9. Optimistic conflict-free
10. Resilience
11. Authentication
12. Robust
13. Fair
14. Tree-based LoadBalancer



Implementation

- The core unit
  - Defines entities, e.g., clients and nodes, and maintains the application logic and data
  - Defines workloads and benchmarks

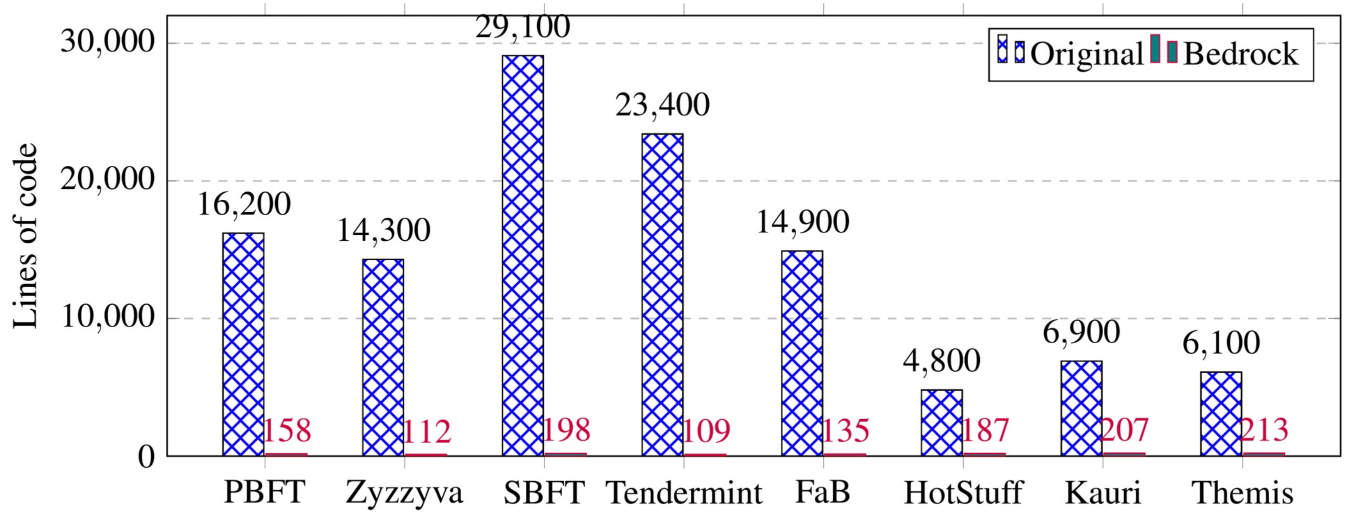
The state manager
- Enables the core unit to track the states and transitions of each entity according to the protocol
- Defines a domain-specific language (DSL) to rapidly prototype BFT protocols

The plugin manager
- Implements protocol-specific behaviors that cannot be handled by the protocol config
- Enables users to define their own dimensions/values or to update existing dimensions without requiring changes to the platform code or rebuilding the platform binaries

The run-time unit
- Manages the run-time execution
- E.g., manages benchmarks, setups all entities, enables plugins to run, reports results

DSL code

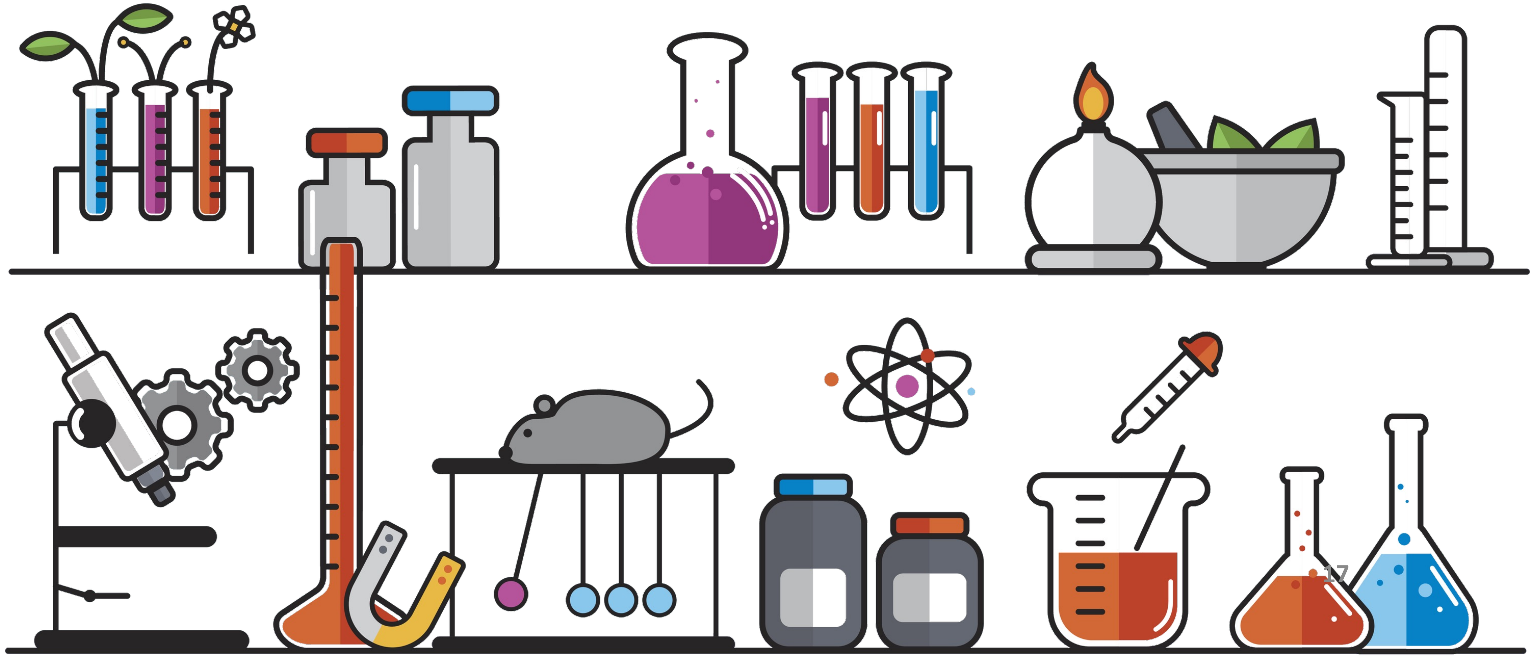
- Written in the protocol config
- Defines different dimensions and their chosen values, the list of roles, phases, states, messages, quorum conditions, and plugins
- Reduces the effort needed to write a BFT protocol



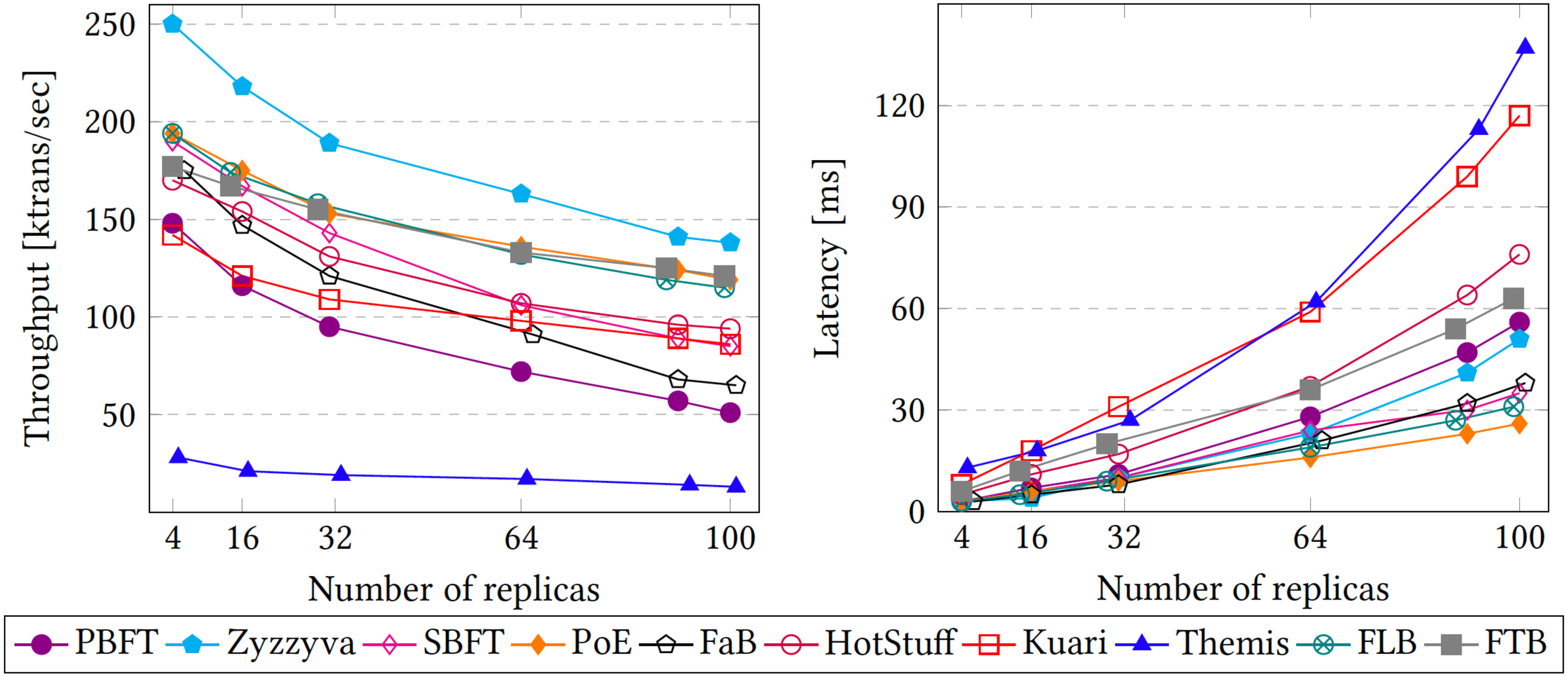

```yaml
protocol:
  general:
    leader: stable
    requestTarget: primary

  roles:
    - primary
    - nodes
    - client

  phases:
    - name: normal
      states:
        - idle
        - wait_prepare
        - wait_commit
        - executed
      messages:
        - name: request
          requestBlock: true
        - name: reply
          requestBlock: true
        - name: preprepare
          requestBlock: true
        - prepare
        - commit
    - name: view_change
      states:
        - wait_view_change
        - wait_new_view
      messages:
        - view_change
        - new_view
    - name: checkpoint
      messages:
        - checkpoint

  transitions:
    from:
      - role: client
        state: idle
        to:
          - state: executed
            update: sequence
            condition:
              type: msg
              message: reply
              quorum: 2f + 1
```

Experimental settings

- Platform: Amazon EC2
- PBFT, Zyzzyva, SBFT, FaB, PoE, (Chained-)HotStuff, Kauri, Themis, FLB and FTB.
- Evaluate the impact of the impact of design choices 1, 2, 3, 6, 7, 8, 10, 11, 13, and 14
- Workload with client request/reply payload sizes of 128/128 byte.
- Measuring performance
  - Throughput
  - Latency

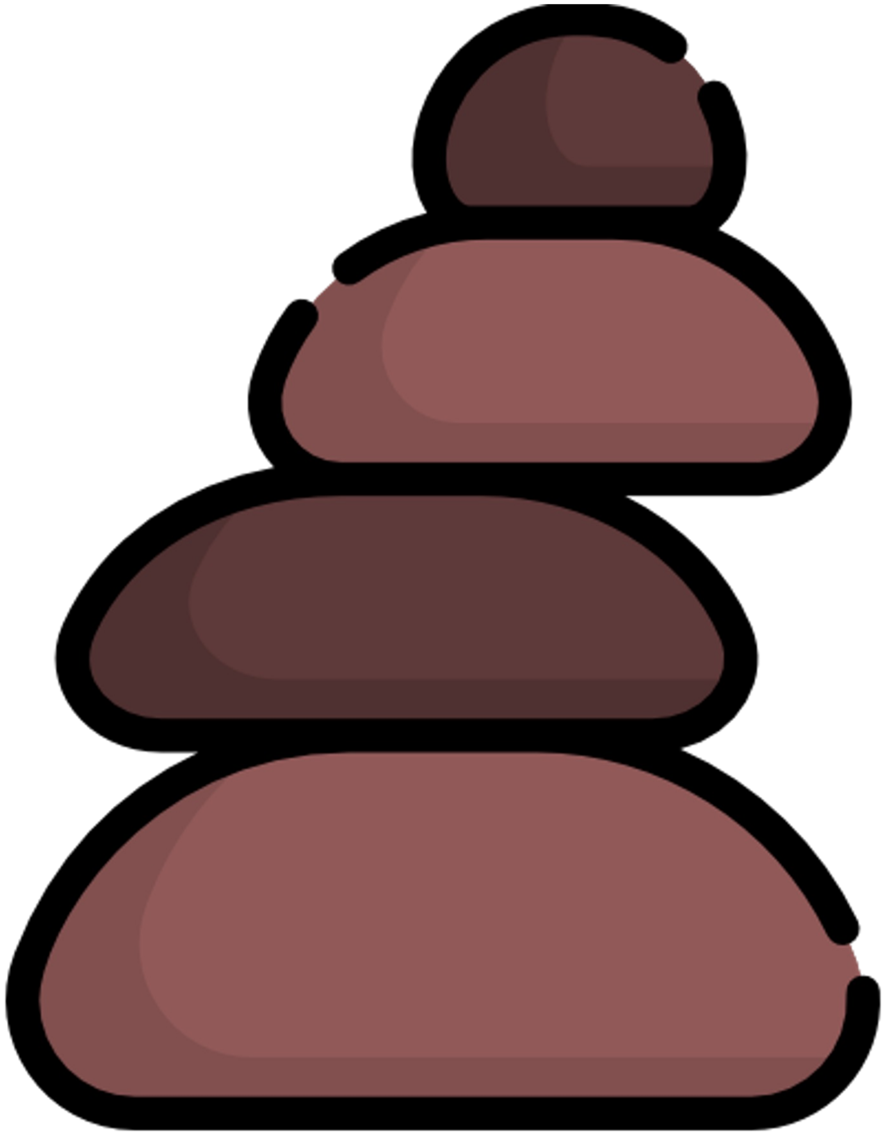


Performance with different number of replicas



Future work

- Enabling users to check the correctness of their written protocols
  - Transforming the DSL code written in Bedrock to the language used by verification tools
- Diversifying replica implementation using n-version programming
  - To ensure the independent failure of replicas
- Extending the supported protocols
  - E.g., adding synchronous and fully asynchronous protocols
- Enabling scalable transaction processing
  - Running different instances of consensus protocols in parallel
- Incorporating automatic selection strategies in Bedrock
  - Using machine learning to select the appropriate BFT protocol, or switch protocols at runtime



Questions?