Stony Brook University

Adaptive Transaction Management in Permissioned Blockchains

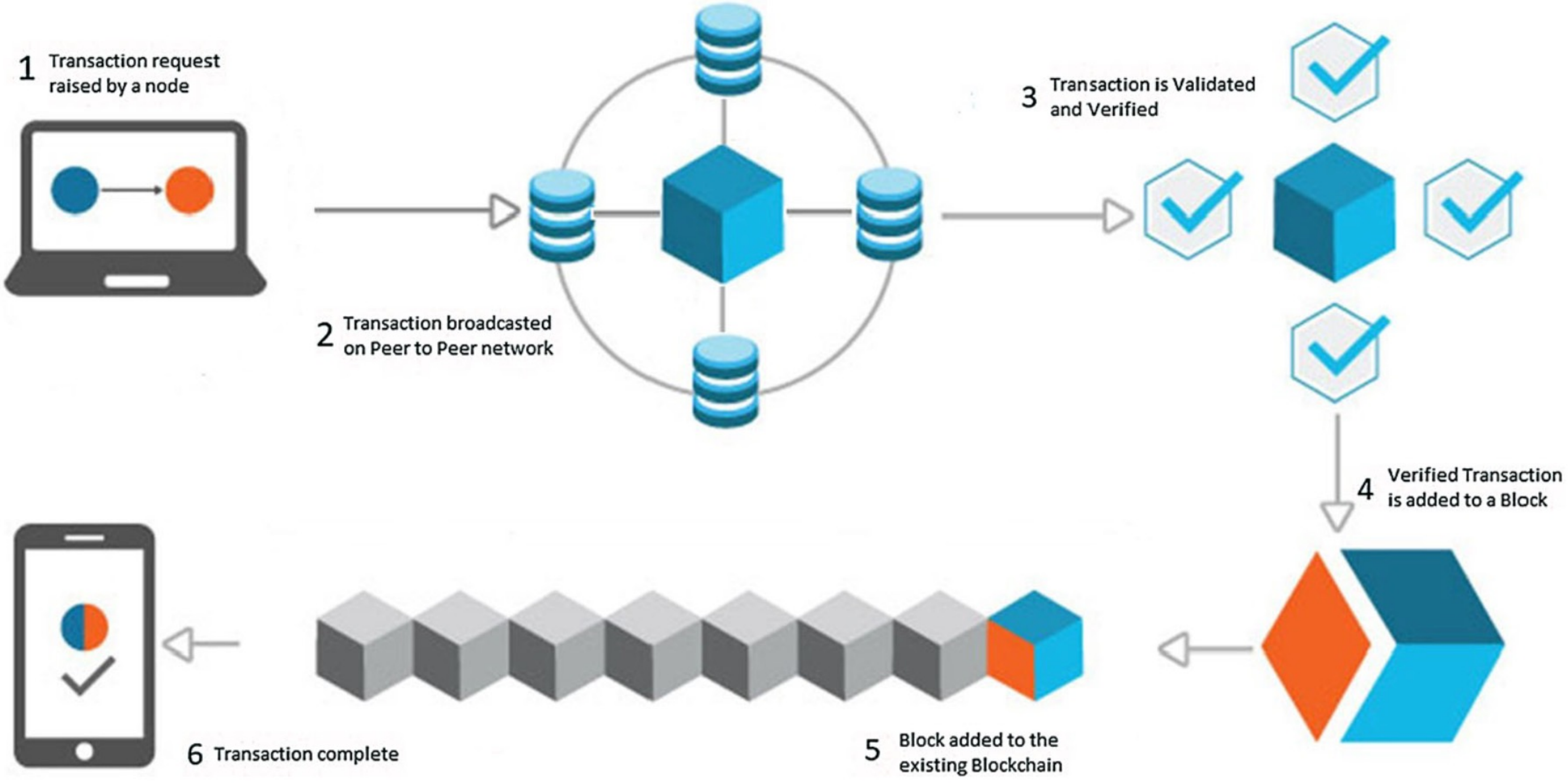
Sai Praneeth Reddy, Siddhartha Malladi, Madhu Lakkoju

FAR BEYOND

CSE 590: Distributed and Decentralized Data Management

Contents

Blockchain


1 Transaction request raised by a node
2 Transaction broadcasted on Peer to Peer network
3 Transaction is Validated and Verified
4 Verified Transaction is added to a Block
5 Block added to the existing Blockchain
6 Transaction complete

Problem

- **Blockchain as a Service offering has a vast variety of use cases that lead to different workloads.**
- Workload Characteristics:
  - varying read/write ratios
  - skewness of popular key
  - compute intensity
- Not adaptable to Different workloads
- Lead to poor performance and inefficient resource utilization.
- If we use some heuristics, if hardware configurations change the heuristics might fail.

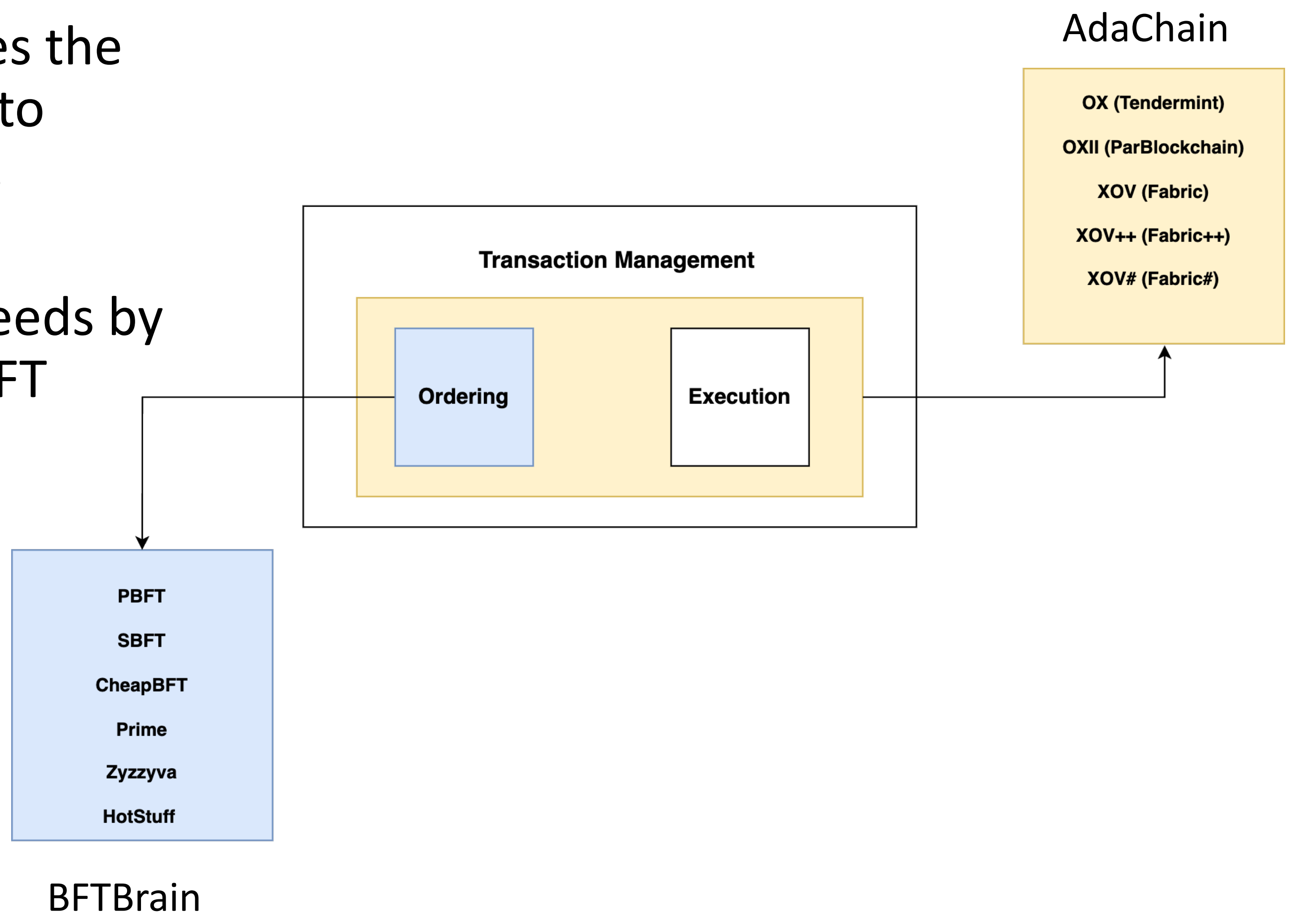
Workloads:

Workload	f	Write Ratio	Contention Level	Load	Compute Intensity	Req. size	Proposal slowness	Absent count
A	1	Low	High	High	High	1 KB	0ms	0
B	1	Mid	High	Low	Mid	1 KB	100 ms	0
C	4	Mid	Mid	High	Low	4 KB	20 ms	4
D	4	High	Low	High	High	100 KB	0 ms	0

Different Workloads demand different requirements like compute, parallel execution, and block size
So, no one could identify the optimal configuration.

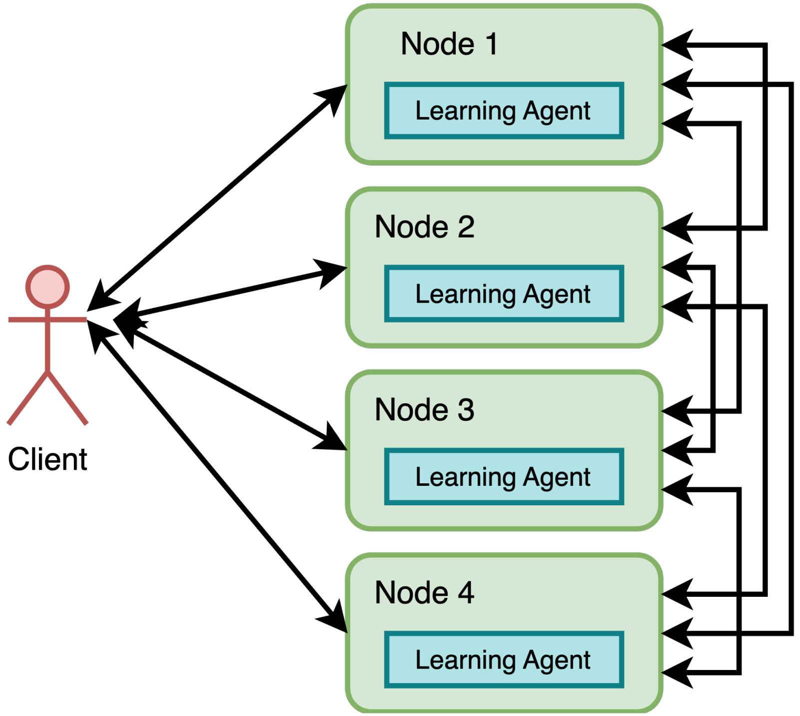
Existing Solutions

- **AdaChain:** Adaptively chooses the best blockchain architecture to produce optimal throughput.
- **BFT Brain:** Adapts to system conditions and application needs by switching between a set of BFT protocols in real time.



Proposed Solution: Fully Adaptive Transaction Management System

- This automatically **switches** between configurations (Architecture and BFT Protocol) to optimize performance online, compensating for changes in **workload** and **network conditions** to **maximize throughput.**
- Liveness, Safety.
- A fully decentralized machine-learning approach.
- Hardware change resistant
- Give best performance

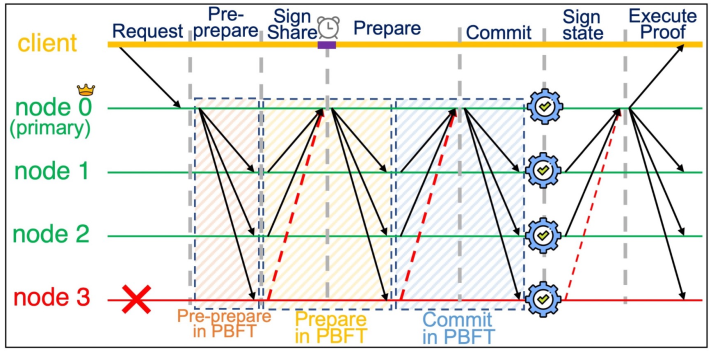


Configuration Landscape

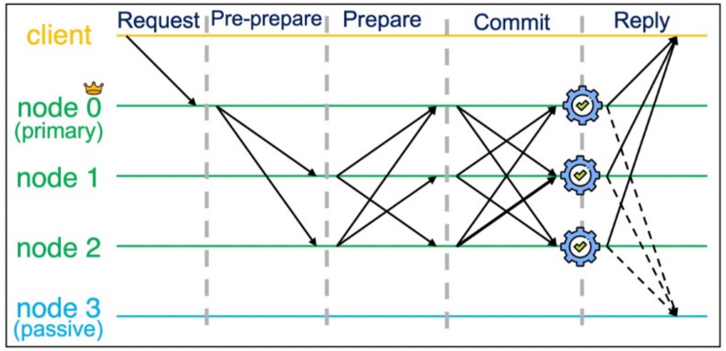
- **Architecture Pool:**
  - OX (Tendermint)
  - OXII (ParBlockchain)
  - XOV (Fabric)
  - XOV++ (Fabric++)
  - XOV# (Fabric#)

  Performance Params: Block Size, Early Execution, Dependency Graph, Early Abort, Parallel Execution

- **Protocol Pool:**
  - PBFT
  - SBFT
  - CheapBFT
  - Prime
  - Zyzzyva
  - HotStuff-2

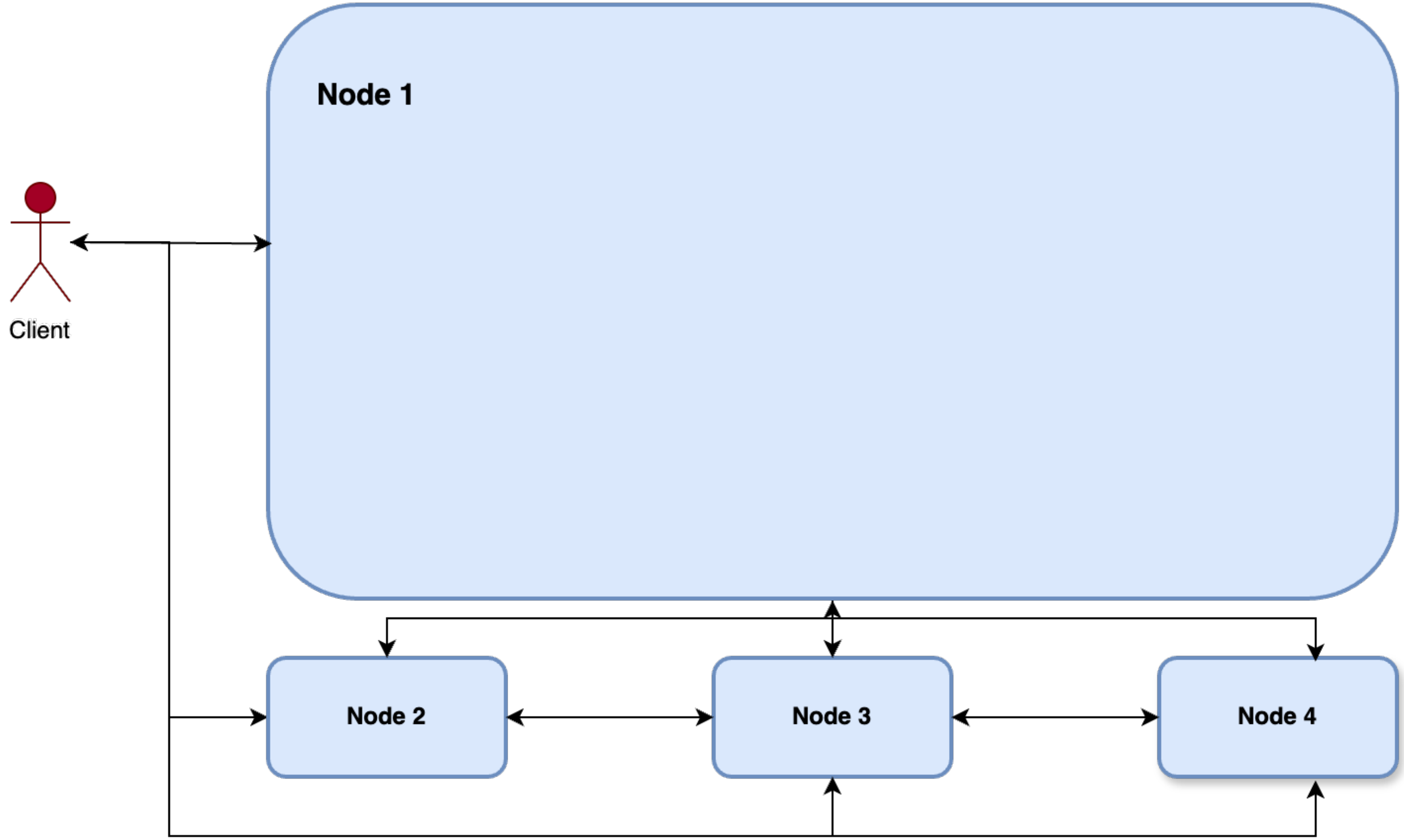


SBFT

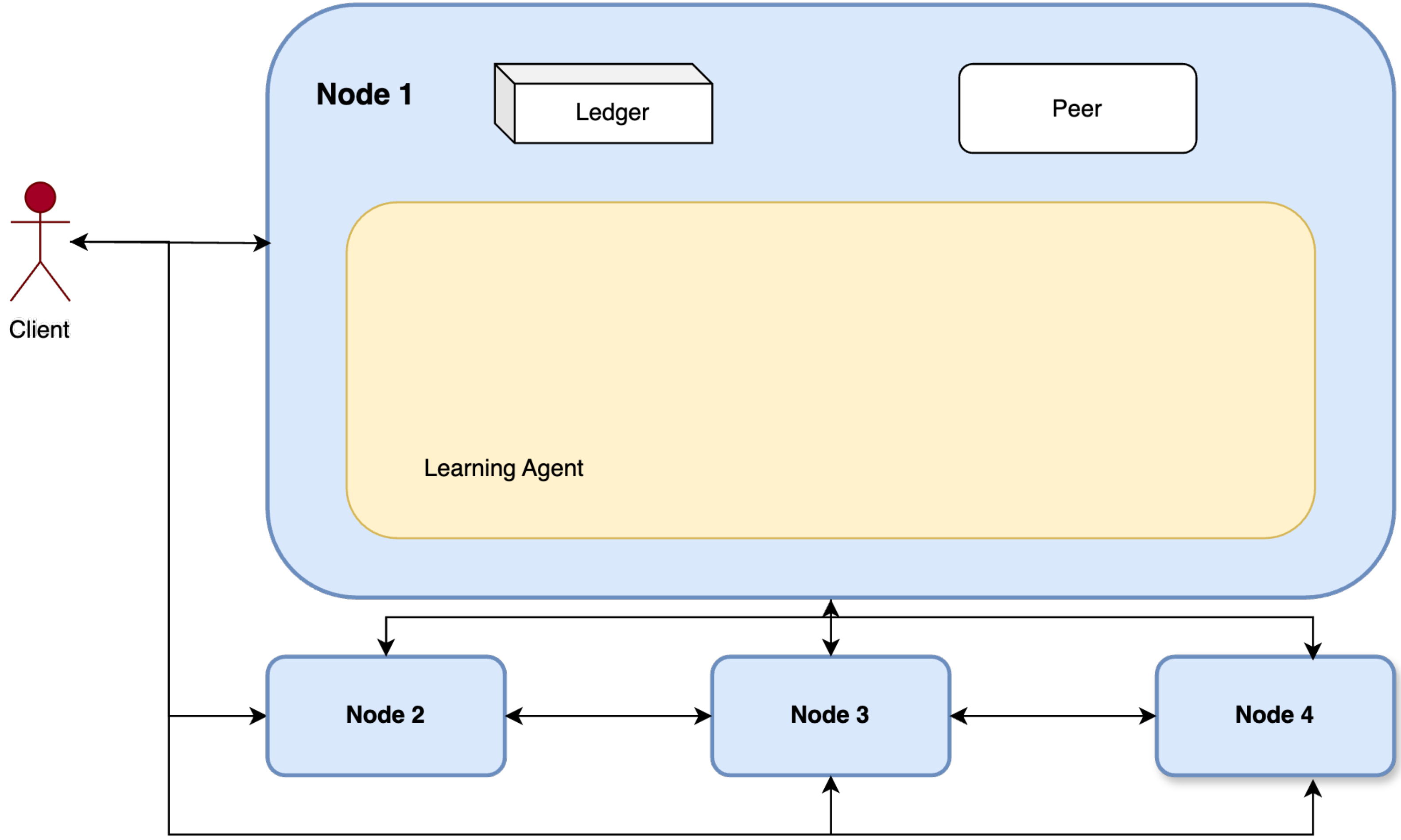


CheapBFT

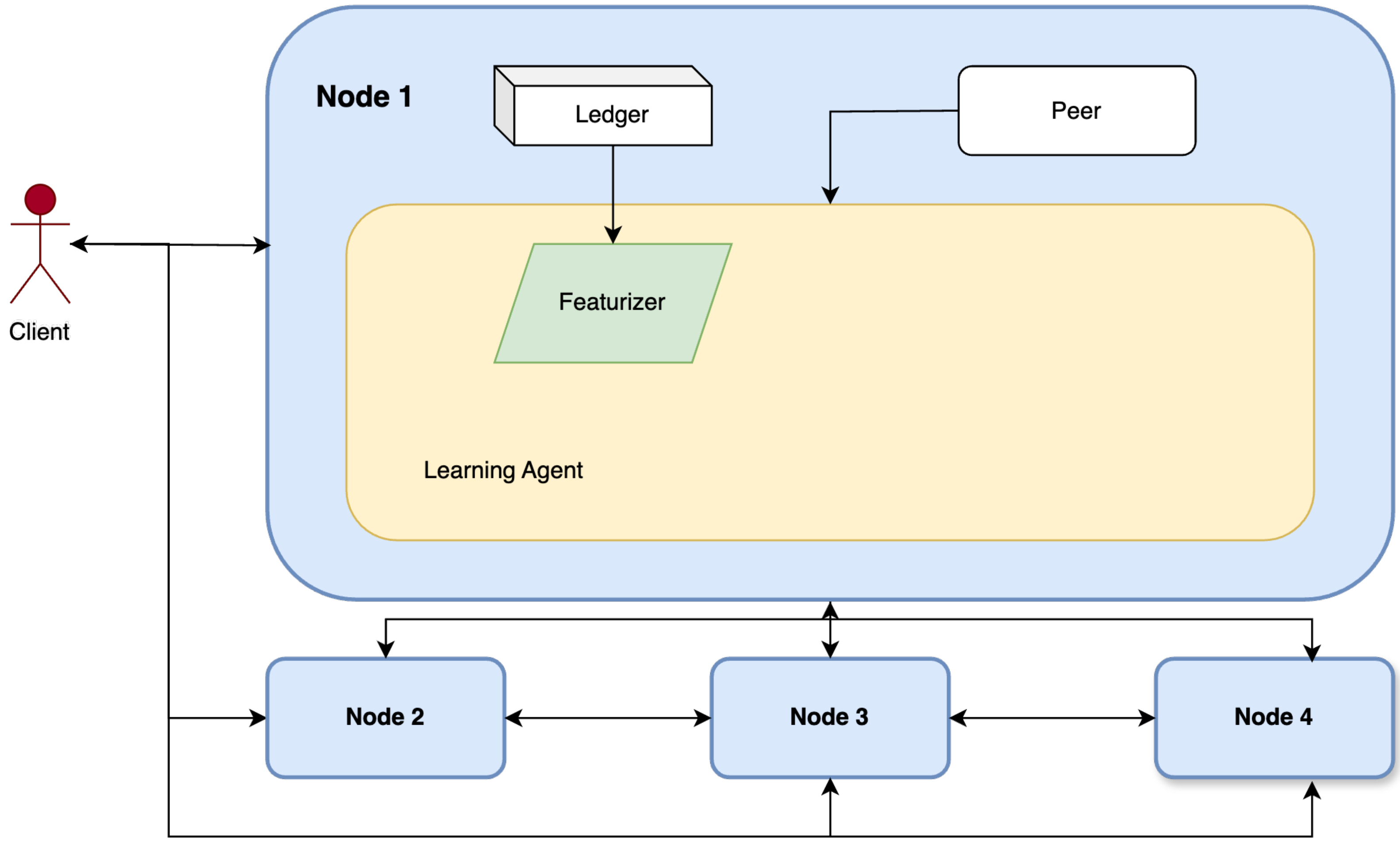
System Design



System Design

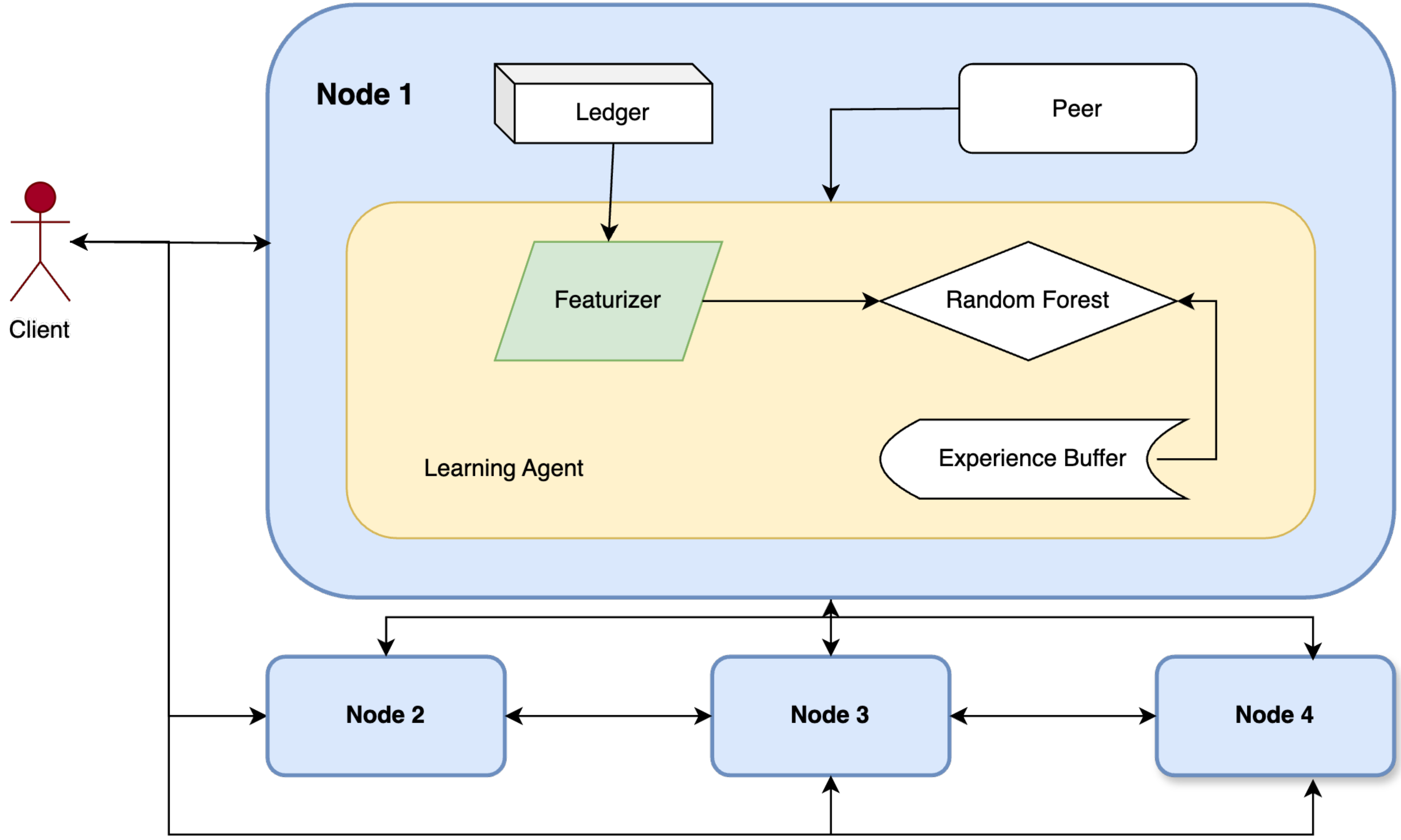


System Design



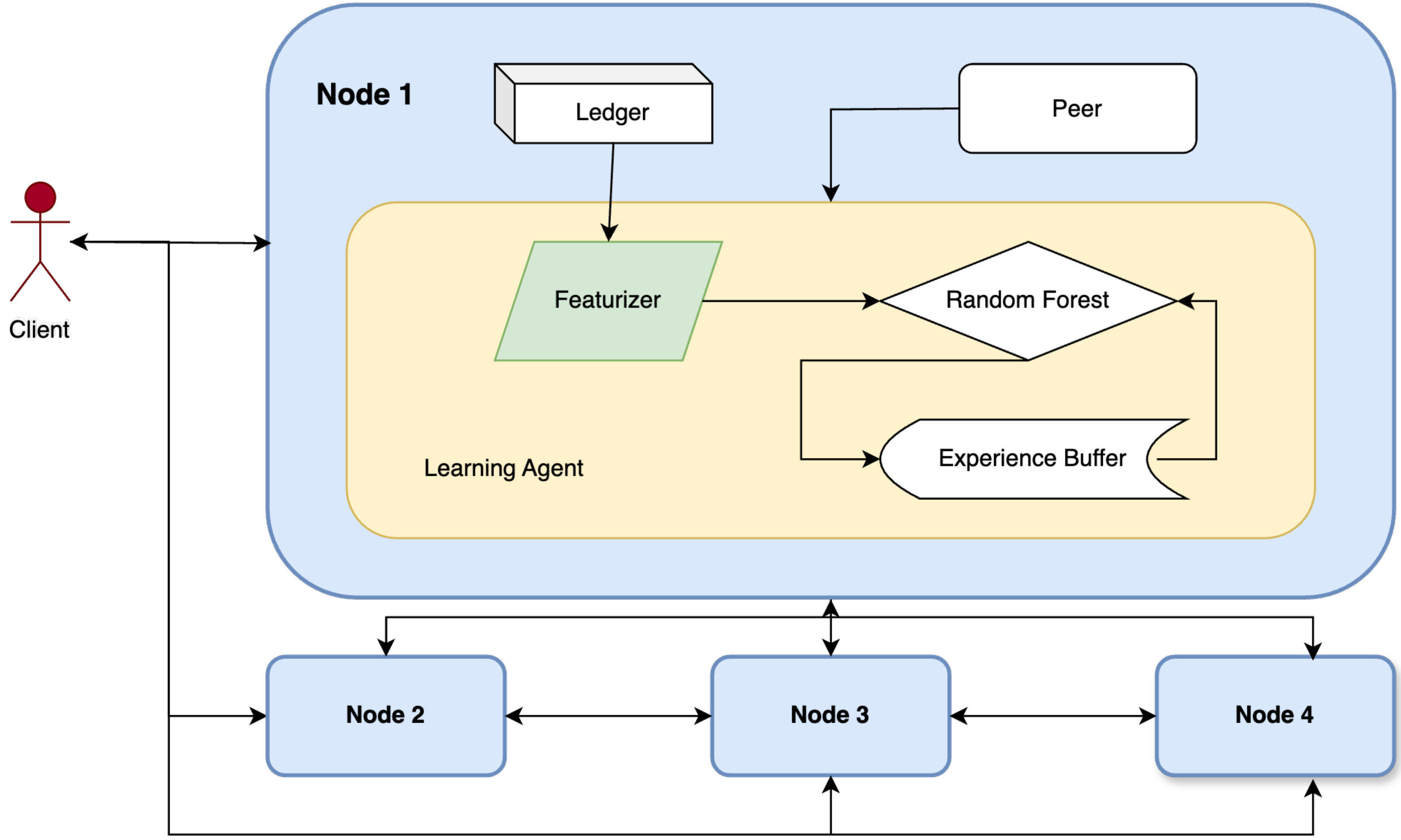
1. Notifying the learning agent

System Design



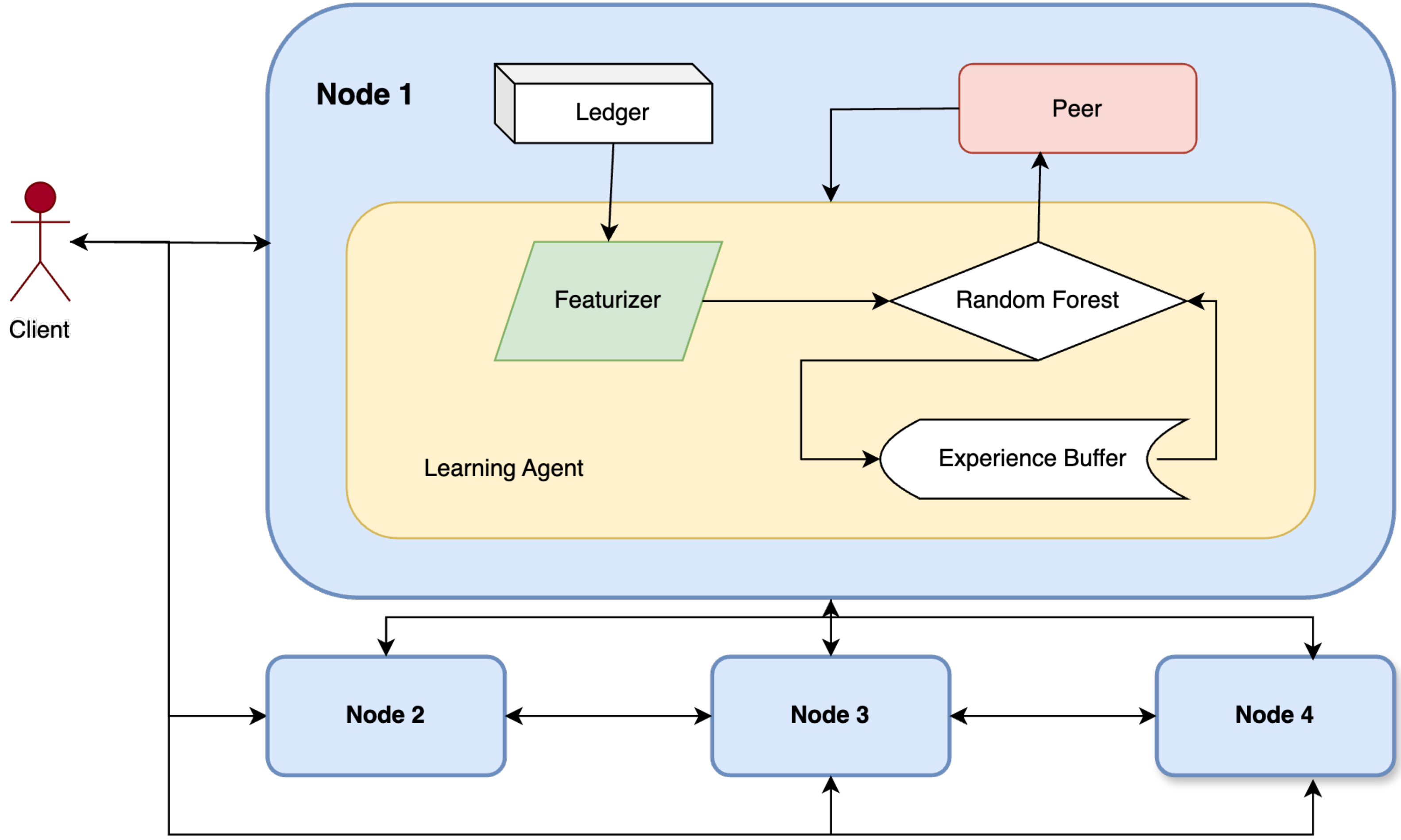
1. Notifying the learning agent
2. Featurization

System Design



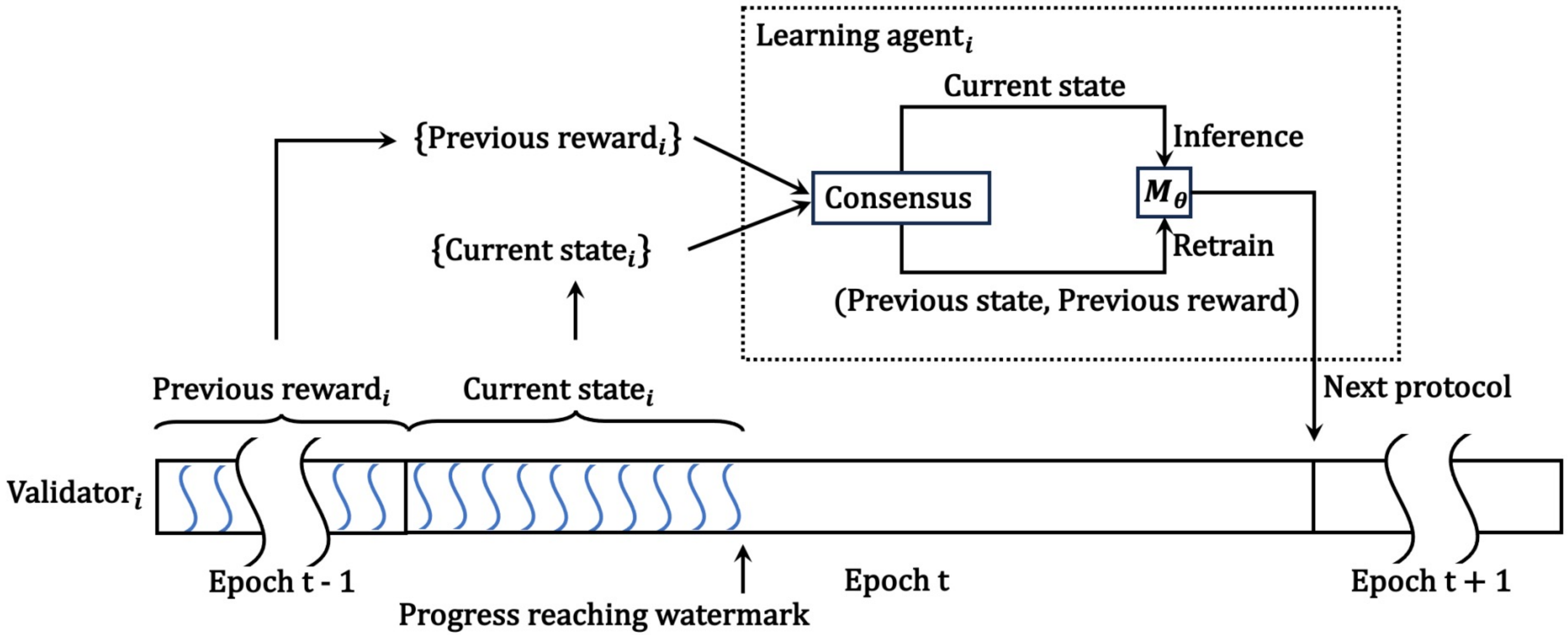
1. Notifying the learning agent
2. Featurization
3. Exchanging performance metrics
4. Estimating the performance of each architecture

System Design



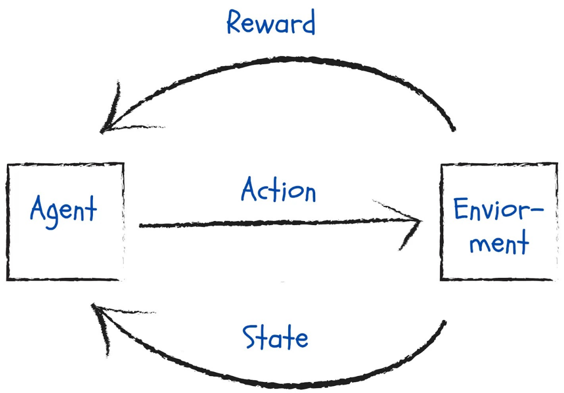
1. Notifying the learning agent
2. Featurization
3. Exchanging performance metrics
4. Estimating the performance of each architecture
5. Building experience buffer
6. Retraining

Episodes: Watermarks and Actions



Reinforcement Learning Approach:

- Learns from its **mistakes** and self-corrects through trials.
- To develop a good heuristic, an expert needs to **exhaustively experiment**, **not needing upfront data.**
- **Why not a supervised learning model?** - Supervised learning that assumes **training data is complete** and requires a separate data collection process before **deployment**
- If hardware changes full re-training is required



Learning Agent and Algorithms

- We leverage reinforcement learning.
- Contextual multi-armed bandit problem – Minimize regret
  - **Context:** Workload
  - **Arms:** Configurations
  - **Reward:** Effective throughput
- **Problem Formulation:** $r_n = (optimal\ configuration - selected\ configuration)$
- **Thompson Sampling** - Balances Exploitation & Exploration
- **Predictive model** - Random Forest (with Thompson Sampling)
  - **INPUT:** Workload, configurations
  - **OUTPUT:** performance
- **Learning Coordination:** Learning Agent choose the best config, consensus between the learning agent is done.

State Space and Action Space

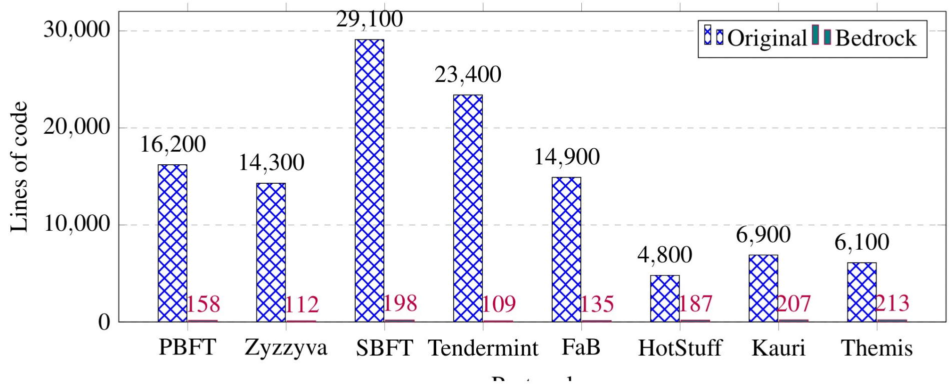
- All aborted or invalidated transactions are still written to the ledger with a **validity flag**
- **State Space:** Helps in featurizing the current state, measurement in between 1 block
  - Write a ratio: Ratio of write transactions
  - Hot-key ratio
  - Transaction Arrival Rate
  - Execution Delay
- **Action space:**
  - Consists of a set of blockchain architectures and BFT Consensus protocols to choose from.

Switching Configurations – Safety and Liveness

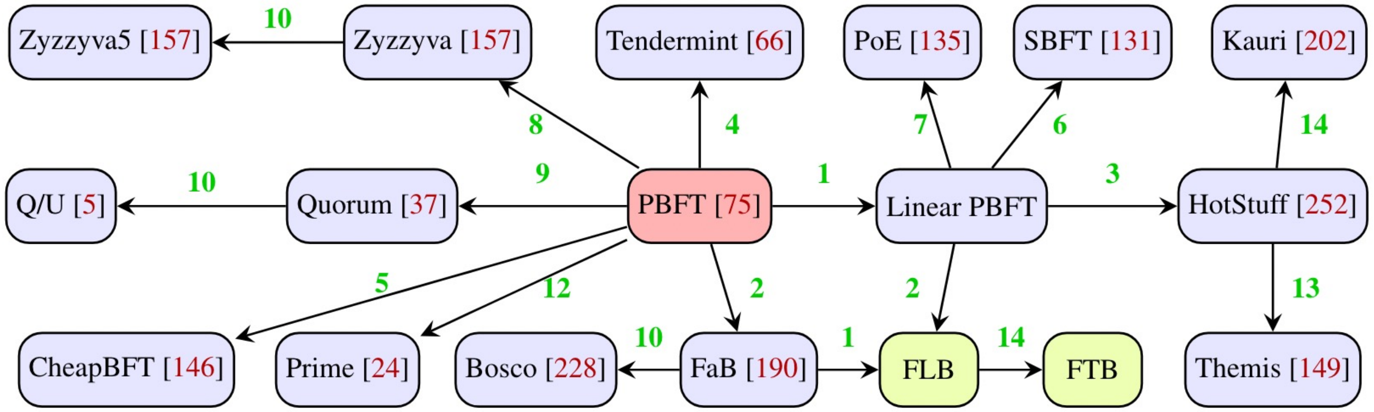
- **Normal Path:** Leader calculates median throughput
- **Slow Path:**
  - Bad configs, stalled commits, Timeouts happen
  - Stop block formation and create a consensus
  - Helpful in breaking bad configs, Consistency

- **Liveness:** Switch configs in a distributed fashion, without stalling the system.

Bedrock



- Outstanding Paper NSDI 2024
- Plug and Play system
- A platform for :
  - BFT protocol **analysis**
  - BFT Protocols **implementation**
  - BFT Protocols **experimentation**.
- Capture the trade-offs between different design space dimensions.



Stony Brook University

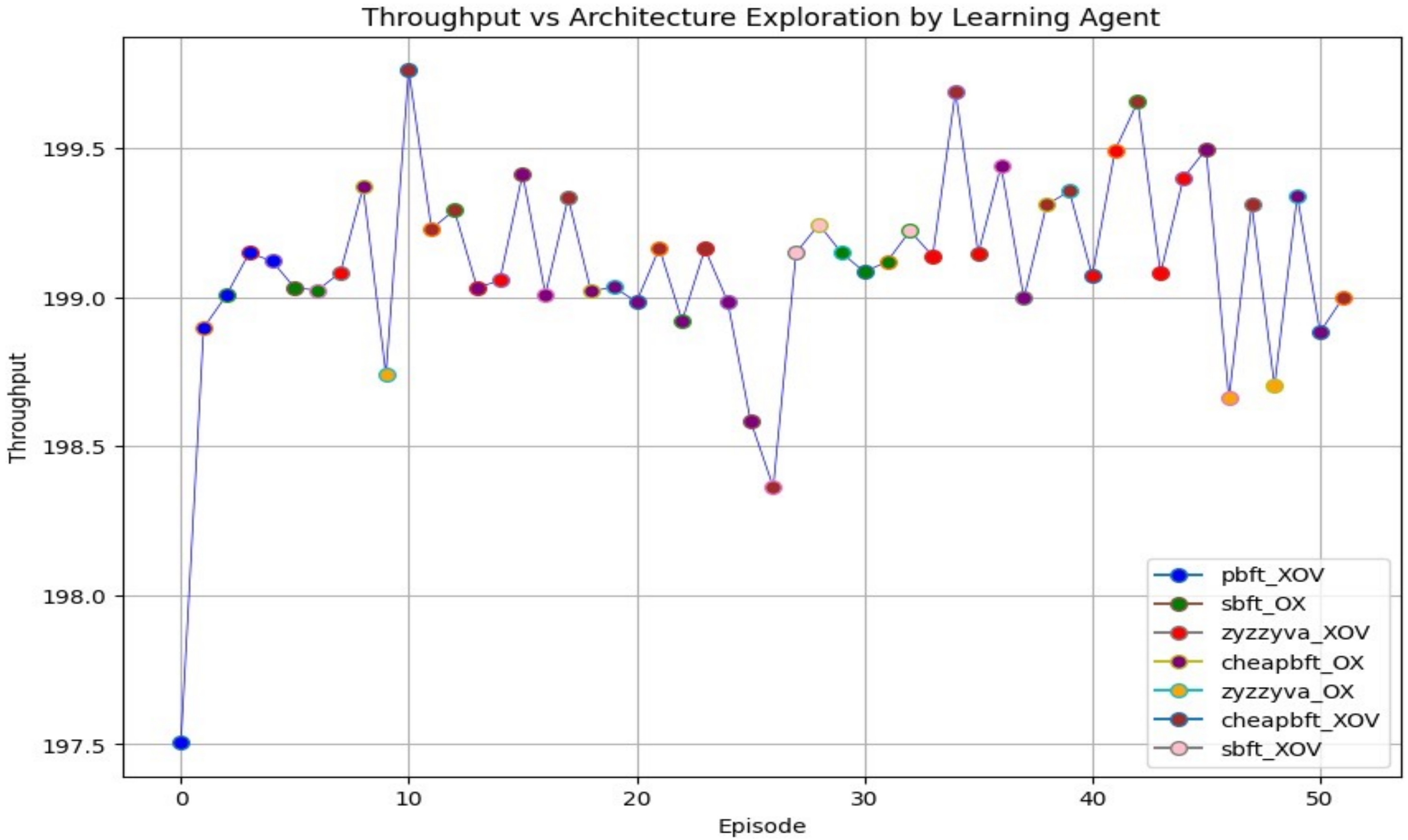
Implementation

- Our implementation is in a layer on top of the **Bedrock** framework.
- Communication is done over RPC (Client, Learning Agent, Peers)
- **Learning agent:** Python, SkLearn

Experimental Setup

- Currently running on the local server with 4 nodes.
- As mentioned, each server has its learning agent.
- Each node has a dedicated connection to other nodes

Evaluation Results – Throughput

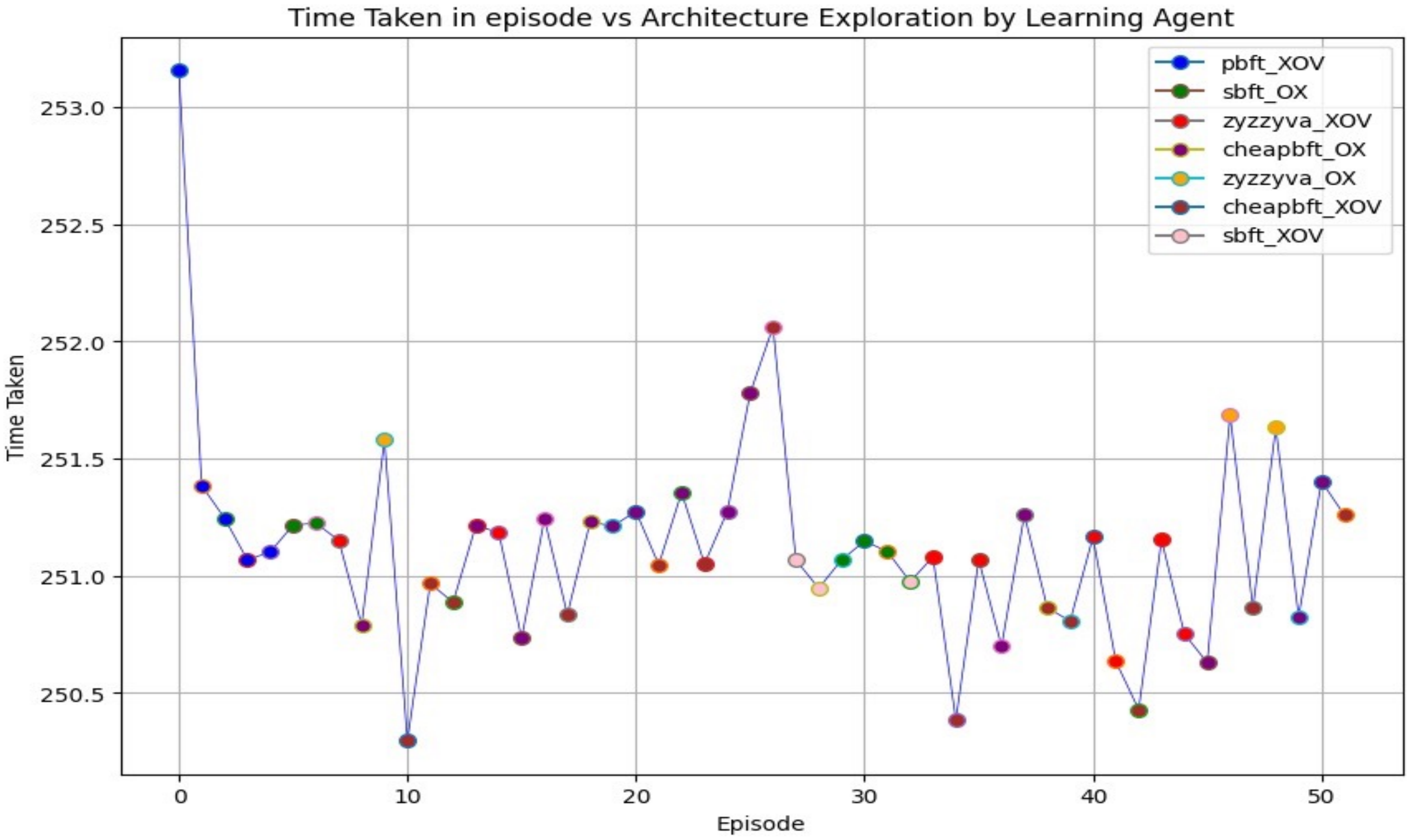


Throughput vs Episode

$$Throughput = \frac{Successful\ Transactions\ Count}{Time}$$

- The Learning Agent explores by selecting architecture combinations and updating its reward.
- Over 50 episodes, the system demonstrates its ability to adapt by choosing different architectures, and protocols, and retraining itself.

Evaluation Results - Average execution time



- The Learning Agent explores by selecting architecture combinations and updating its reward.
- Over 50 episodes, the system demonstrates its ability to adapt by choosing different architectures, and protocols, and retraining itself.

Conclusion

- Fully adaptive system. Adapts to workloads.
- Dynamically selects the top-performing configuration
- Higher throughput compared to fixed configurations

References

- Adachain - https://arxiv.org/abs/2211.01580
- BedRock - https://www3.cs.stonybrook.edu/~amiri/papers/bedrock.pdf
- Code : https://github.com/madhulakkoju/BFTBrain

Thank you