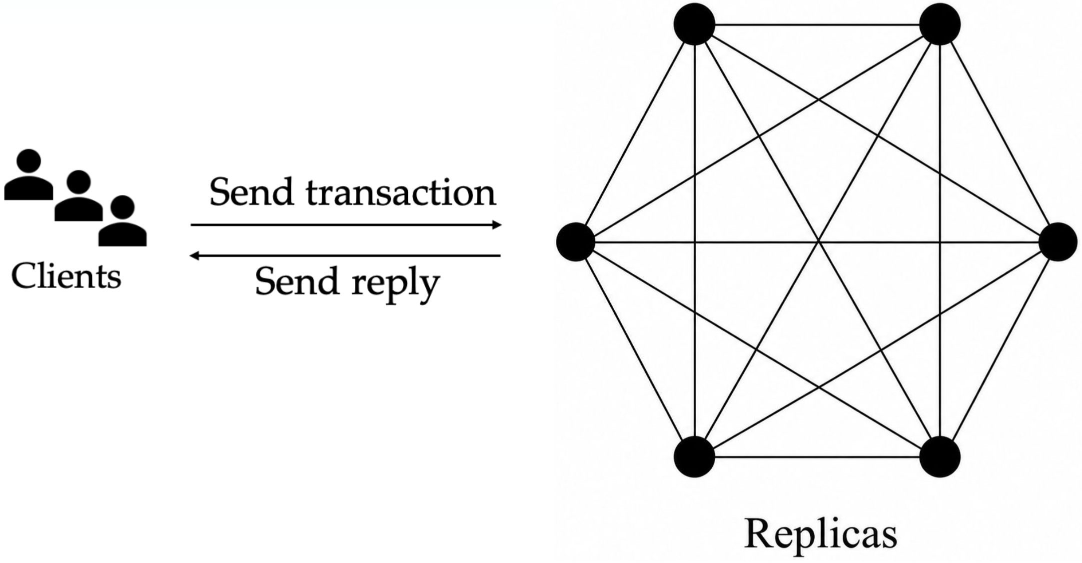
HYDRA: Breaking the Global Ordering Barrier in Multi-BFT Consensus

Hanzheng Lyu*, Shaokang Xie†, Jianyu Niu‡, Mohammad Sadoghi†,
Yinqian Zhang§, Cong Wang‡, Ivan Beschastnikh*, Chen Feng*

* University of British Columbia
† University of California, Davis
‡ City University of Hong Kong
§ Southern University of Science and Technology

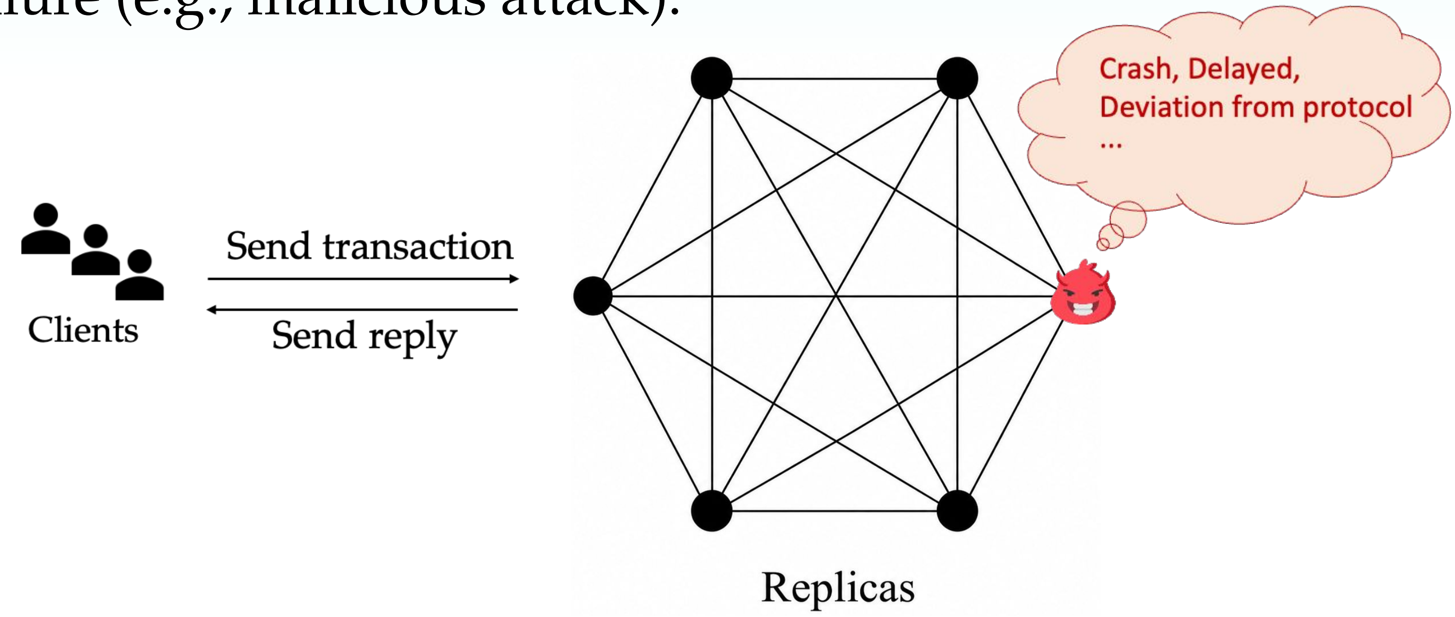
BFT Consensus

- **Byzantine Fault Tolerant (BFT) Consensus** allows a distributed system to reach agreement among replicas despite Byzantine failure (e.g., malicious attack).

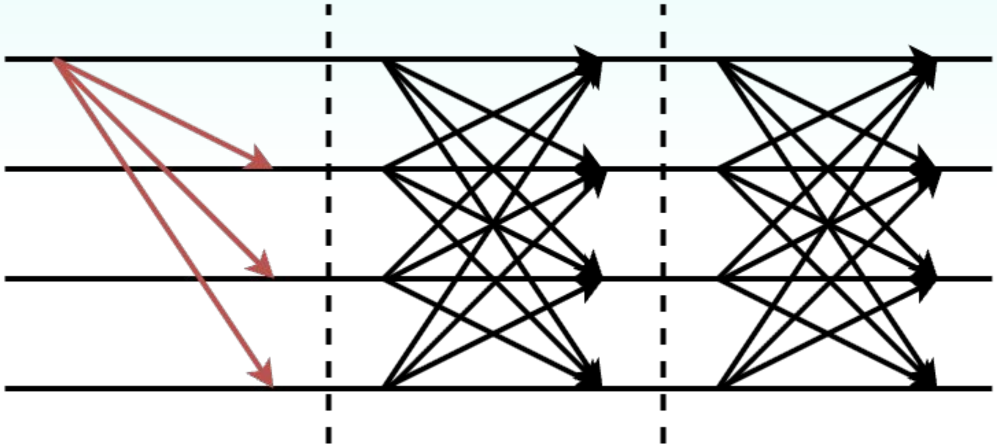


BFT Consensus

- **Byzantine Fault Tolerant (BFT) Consensus** allows a distributed system to reach agreement among replicas despite Byzantine failure (e.g., malicious attack).



Leader-based BFT[1]

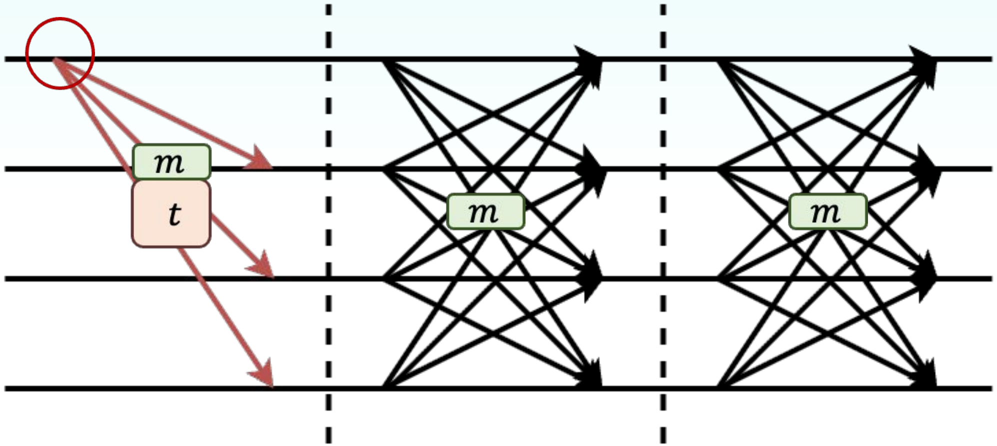


Practical Byzantine Fault Tolerance (PBFT) protocol

[1] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In OSDI, 1999.

Leader-based BFT[1]

m: protocols's message (~100B)
t: the client's payload (~500KB)

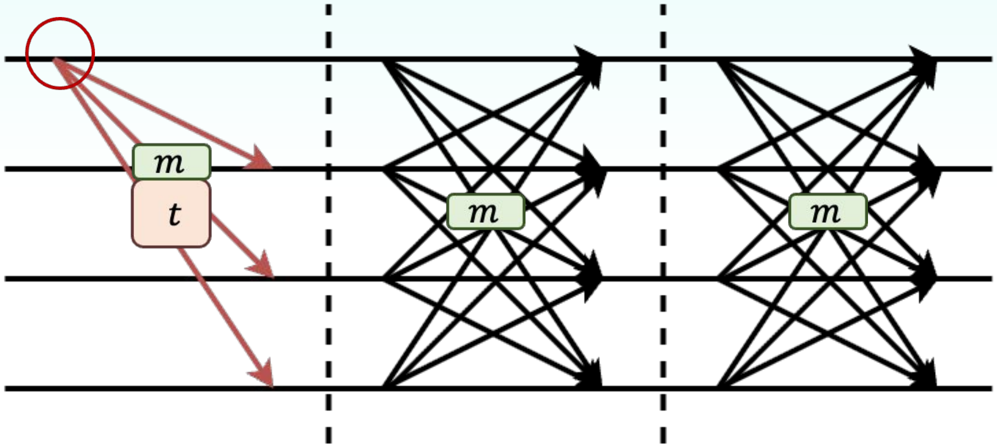


Practical Byzantine Fault Tolerance (PBFT) protocol

[1] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In OSDI, 1999.

Leader-based BFT[1]

m: protocols's message (~100B)
t: the client's payload (~500KB)



Practical Byzantine Fault Tolerance (PBFT) protocol

size(t) >> size(m)

Load Unbalanced!

$$T_{max} \approx \frac{B}{(n-1)size(t)}$$

Maximum throughput

Leader's bandwidth

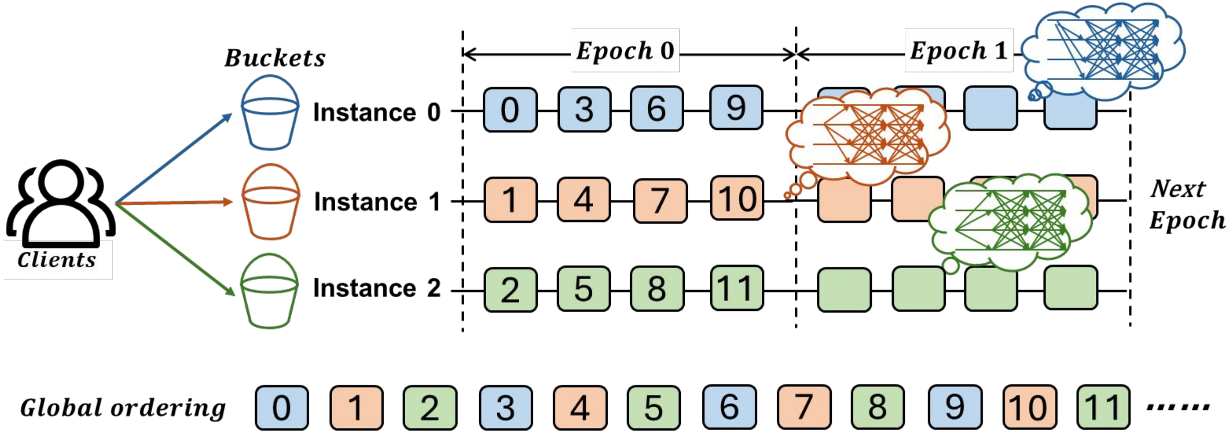
number of replicas

This cause the **Leader Bottleneck**: leader is too busy while others are idling.

[1] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In OSDI, 1999.

Multi-BFT Consensus[1-3]

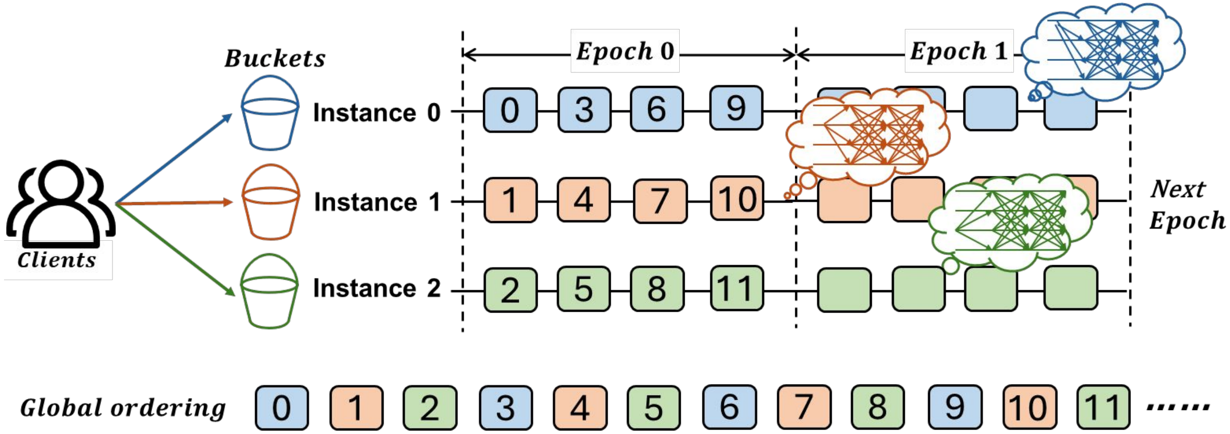
1. Client's transactions are split into distinct **Buckets**.



[1] Stathakopoulou et. al. State Machine Replication Scalability Made Simple, in ACM EuroSys'22.
[2] Stathakopoulou et. al. Mir BFT: Scalable and Robust BFT for Decentralized Networks, in Jsys'22.
[3] Gupta et. al. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing, in IEEE ICDE'21.

Multi-BFT Consensus[1-3]

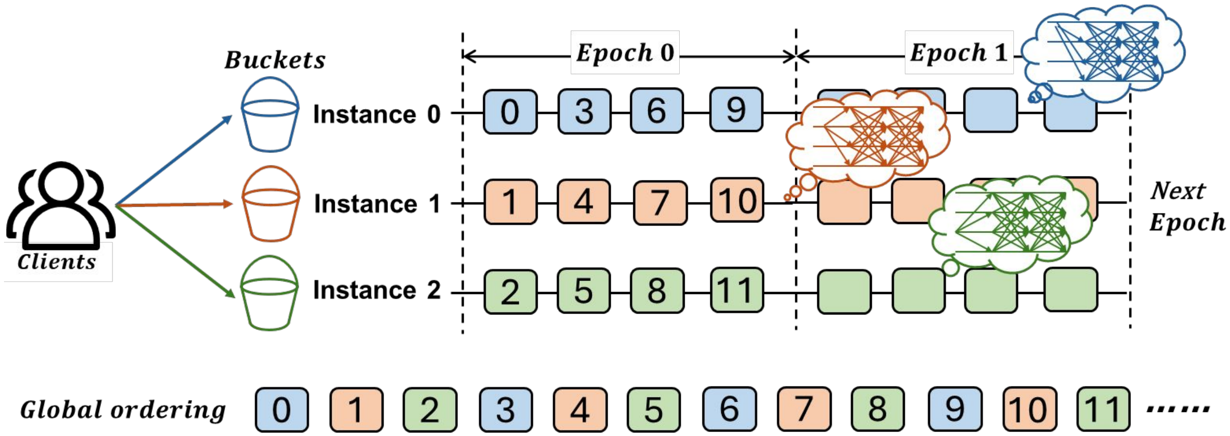
1. Client's transactions are split into distinct **Buckets**.
2. Each replica acts as the **leader of one instance** and backups of others.



[1] Stathakopoulou et. al. State Machine Replication Scalability Made Simple, in ACM EuroSys'22.
[2] Stathakopoulou et. al. Mir BFT: Scalable and Robust BFT for Decentralized Networks, in Jsys'22.
[3] Gupta et. al. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing, in IEEE ICDE'21.

Multi-BFT Consensus[1-3]

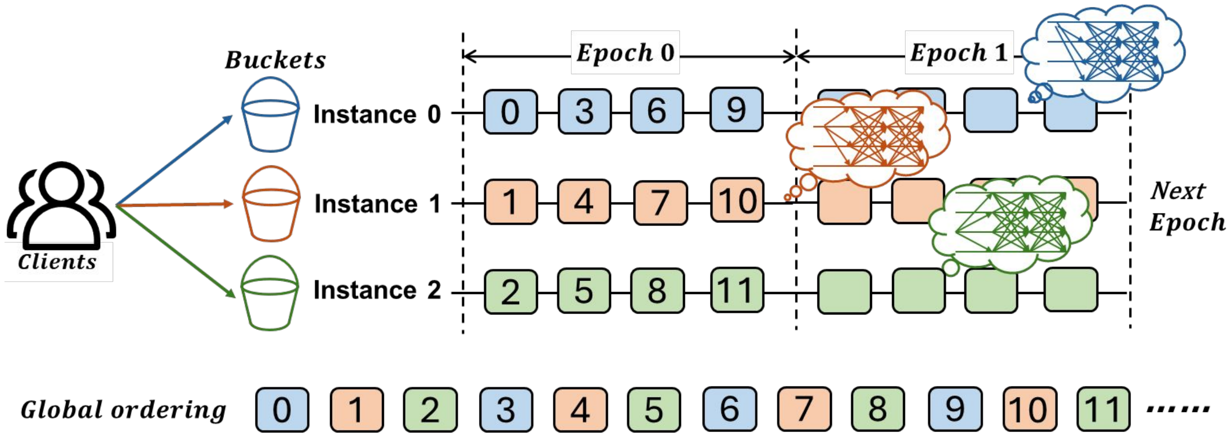
1. Client's transactions are split into distinct **Buckets**.
2. Each replica acts as the **leader of one instance** and backups of others.
3. Each instance independently outputs a sequence of **partially committed blocks**.



[1] Stathakopoulou et. al. State Machine Replication Scalability Made Simple, in ACM EuroSys'22.
[2] Stathakopoulou et. al. Mir BFT: Scalable and Robust BFT for Decentralized Networks, in Jsys'22.
[3] Gupta et. al. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing, in IEEE ICDE'21.

Multi-BFT Consensus[1-3]

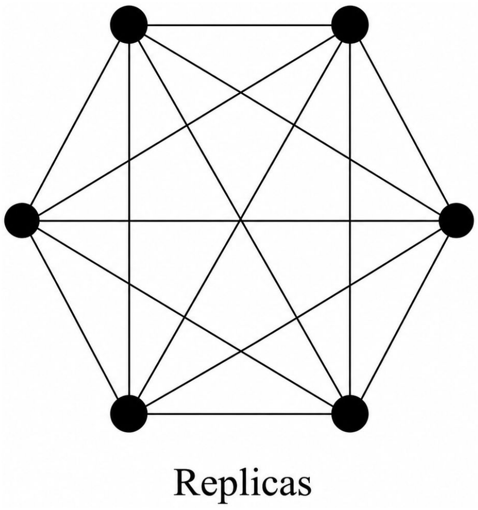
1. Client's transactions are split into distinct **Buckets**.
2. Each replica acts as the **leader of one instance** and backups of others.
3. Each instance independently outputs a sequence of **partially committed blocks**.
4. All replicas need **Coordination** to globally order all blocks.
   - **A slow instance will easily stall the progress.**



[1] Stathakopoulou et. al. State Machine Replication Scalability Made Simple, in ACM EuroSys'22.
[2] Stathakopoulou et. al. Mir BFT: Scalable and Robust BFT for Decentralized Networks, in Jsys'22.
[3] Gupta et. al. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing, in IEEE ICDE'21.

Straggler in the Network

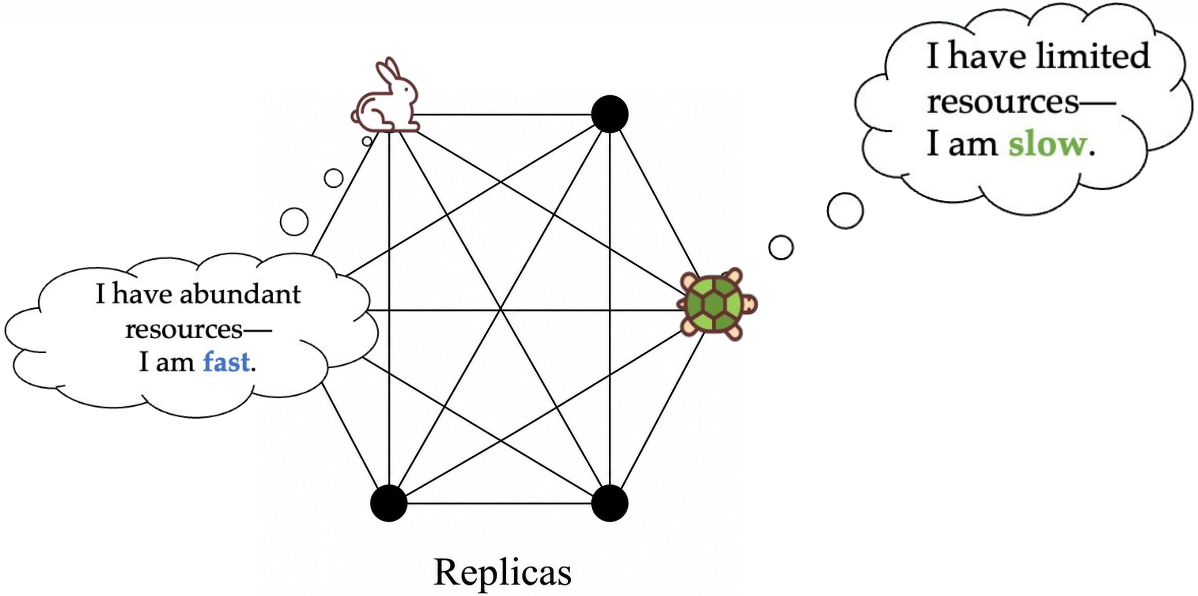
Stragglers are slow replicas which may be significantly slower than others.



Straggler in the Network

Stragglers are slow replicas which may be significantly slower than others.

- Limited computational resources and network bandwidth

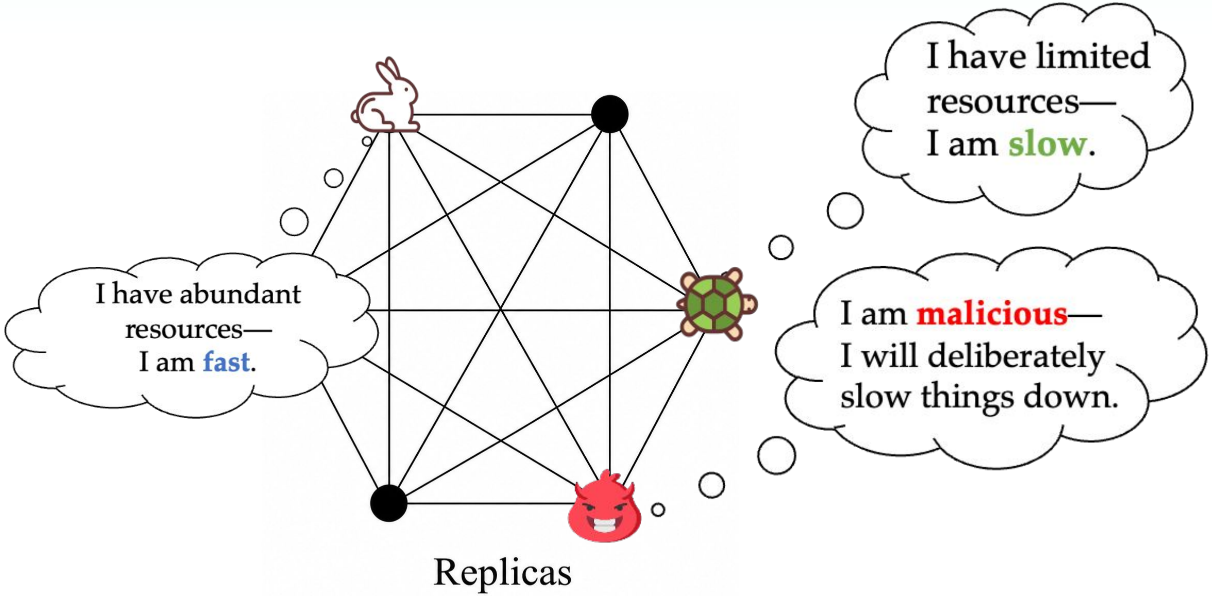


UC DAVIS · THE UNIVERSITY OF BRITISH COLUMBIA · SUSTech Southern University of Science and Technology · CityU · Apache ResilientDB Incubating

Straggler in the Network

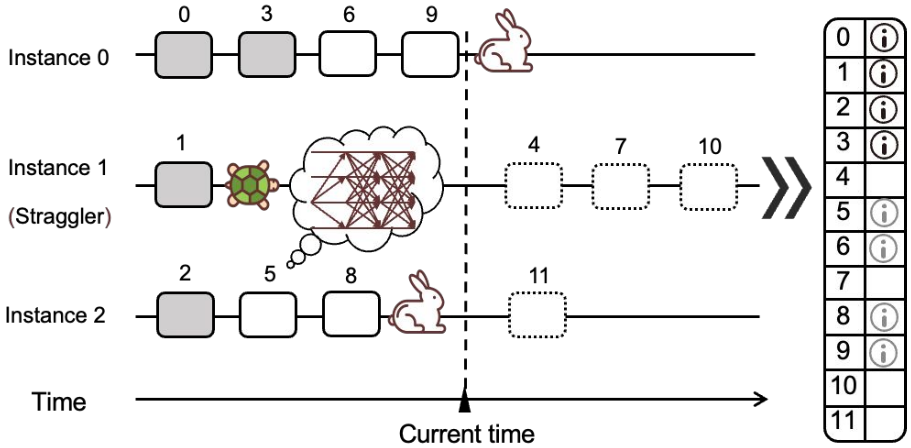
Stragglers are slow replicas which may be significantly slower than others.

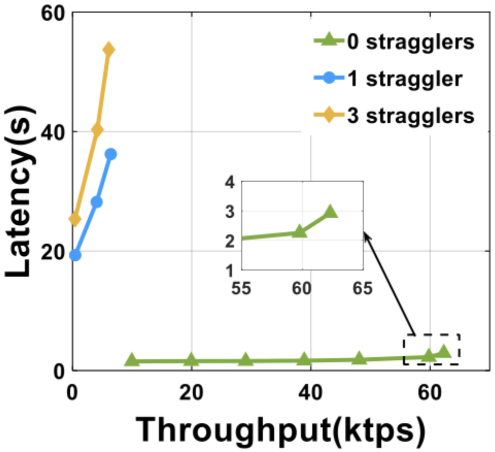
- Limited computational resources and network bandwidth
- Malicious behavior, where Byzantine replicas intentionally delay their behaviors



The impact of Stragglers

- Instance 1 is led by a **straggler replica**.
- Global Ordering needs **coordination**.
- **Performance degradation** from **waiting** for the straggler Instance 1.

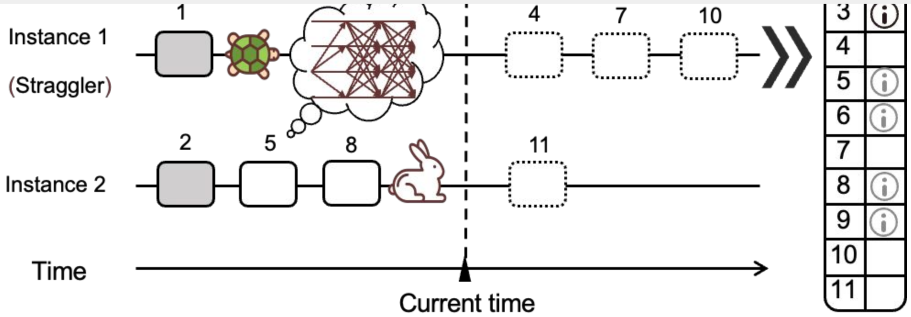


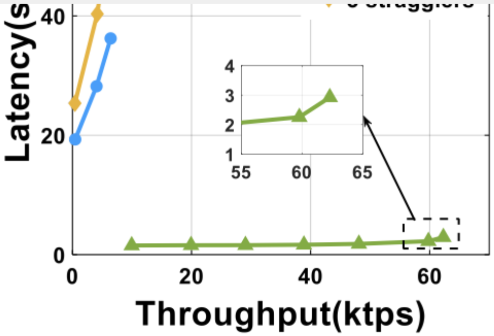


The impact of Stragglers

- Instance 1 is led by a **straggler replica**.
- Global Ordering needs **coordination**.
- Performance degradation from **waiting** for the straggler Instance 1

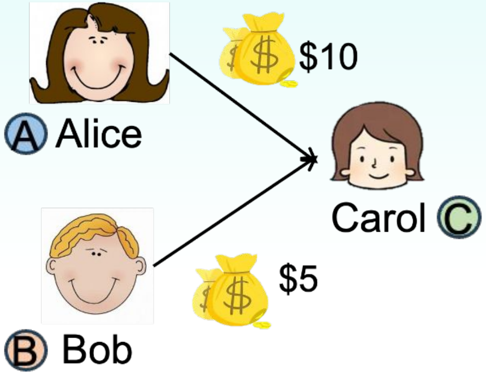
Can we eliminate the Global Ordering?





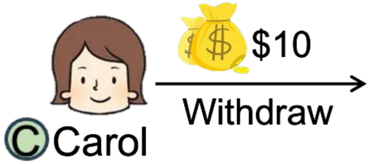
Key Observation

tx_0:



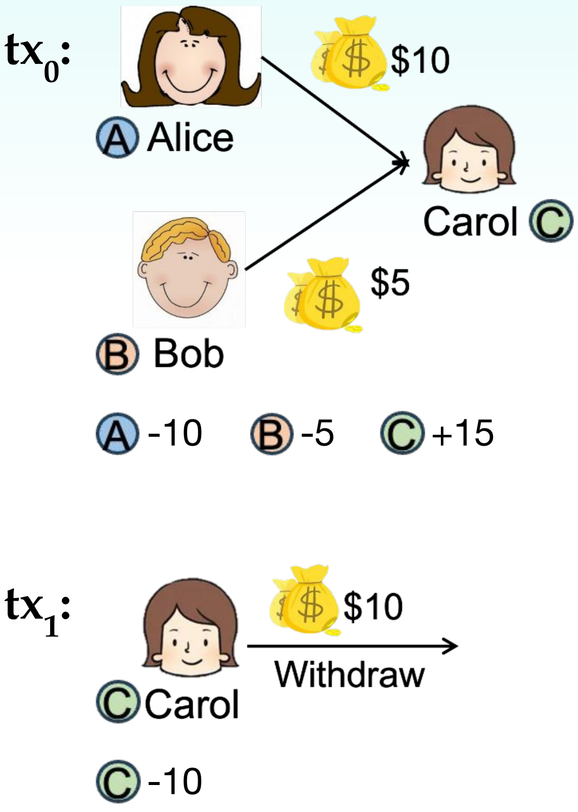
A -10 B -5 C +15

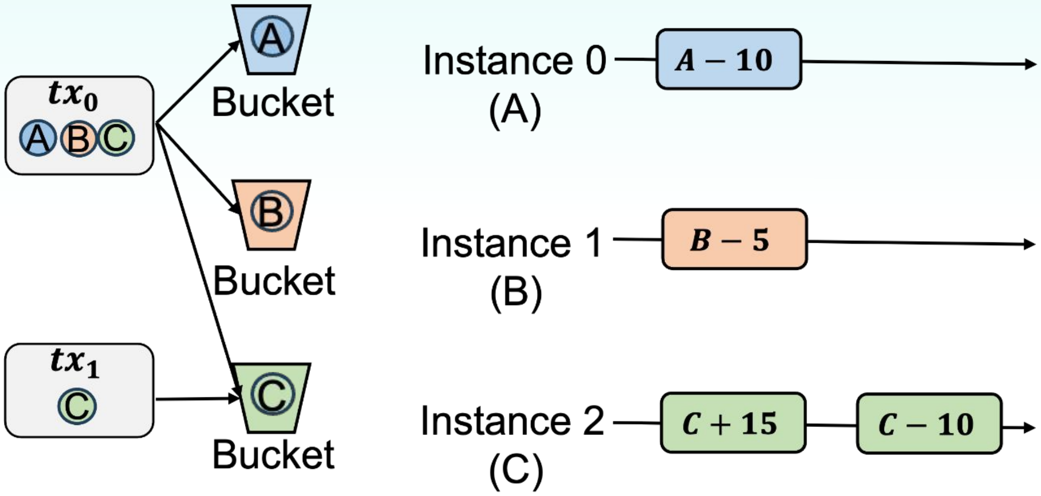
tx_1:



C -10

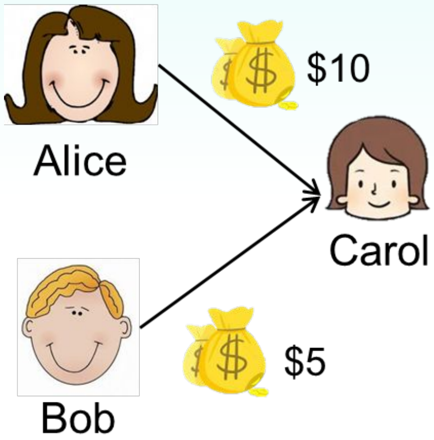
Key Observation

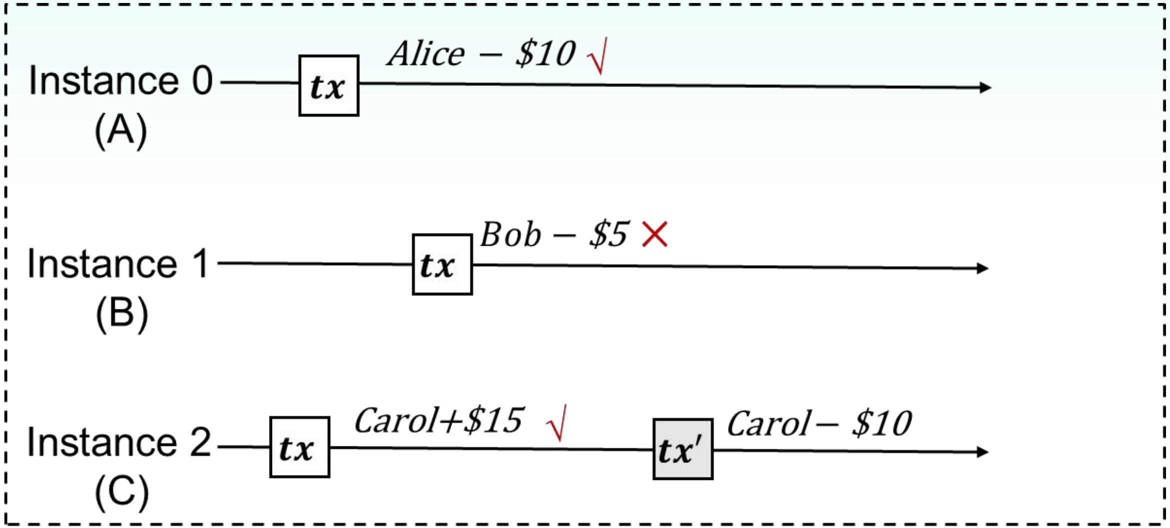




We can shard transactions at the object level, let each instance agree independently.

Challenge 1: Ensuring **atomicity** of transactions across instances.

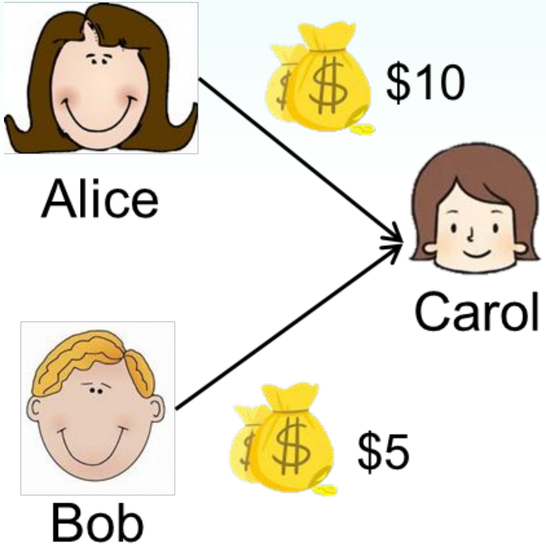


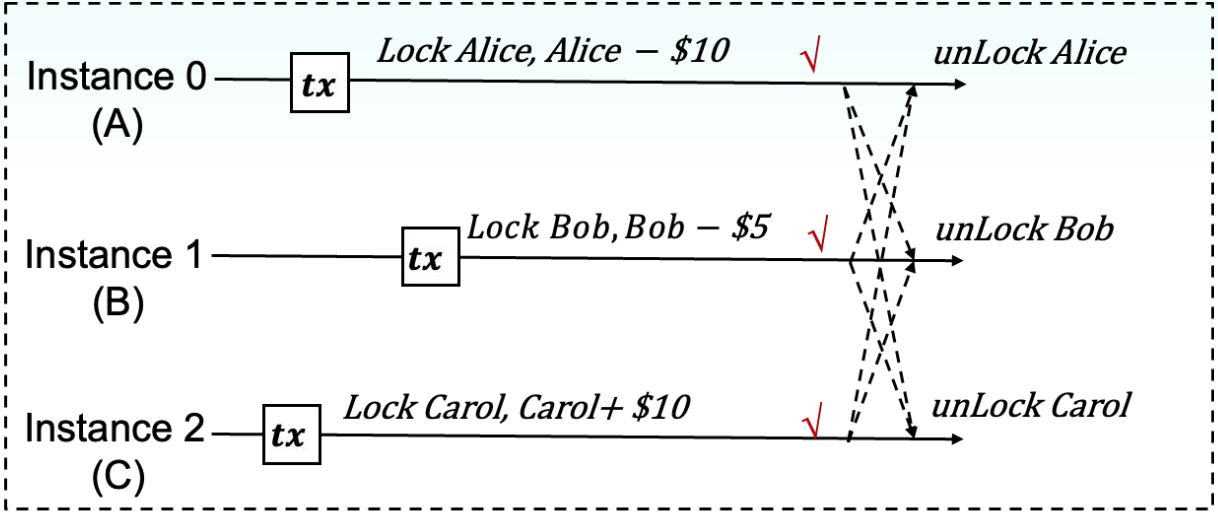


tx:

Alice − $10
Bob − $5
Carol + $15

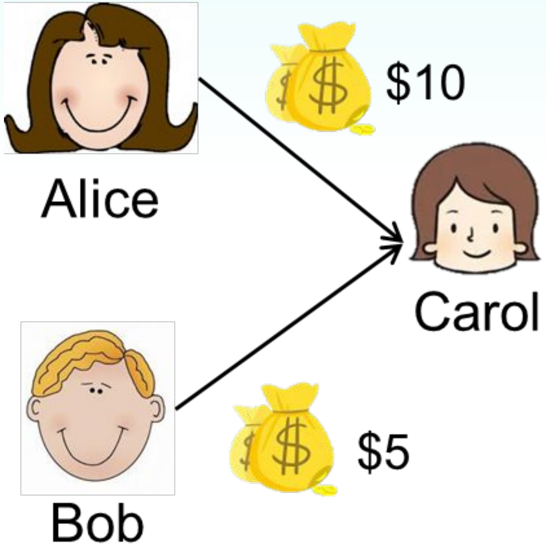
Solution: Before executing a transaction, **lock** all involved objects.

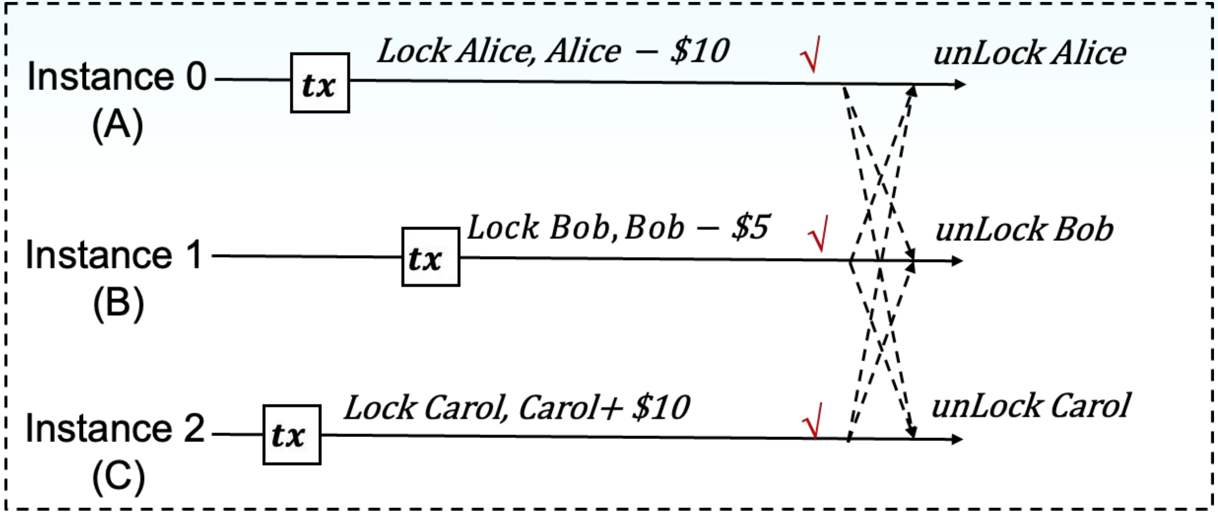




If execution on all objects succeeds, unlock them.

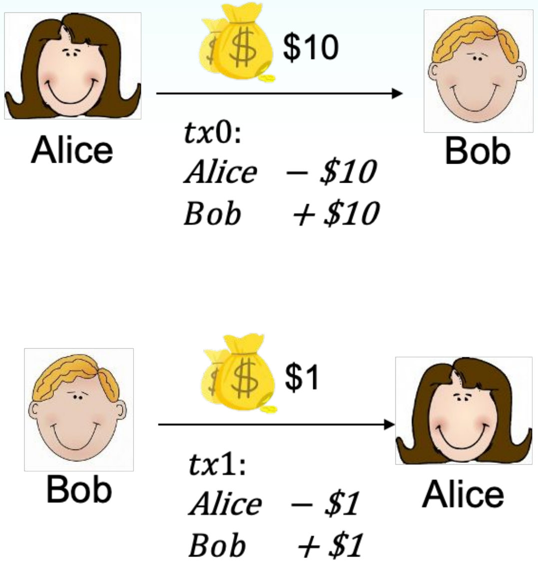
Solution: Before executing a transaction, **lock** all involved objects.

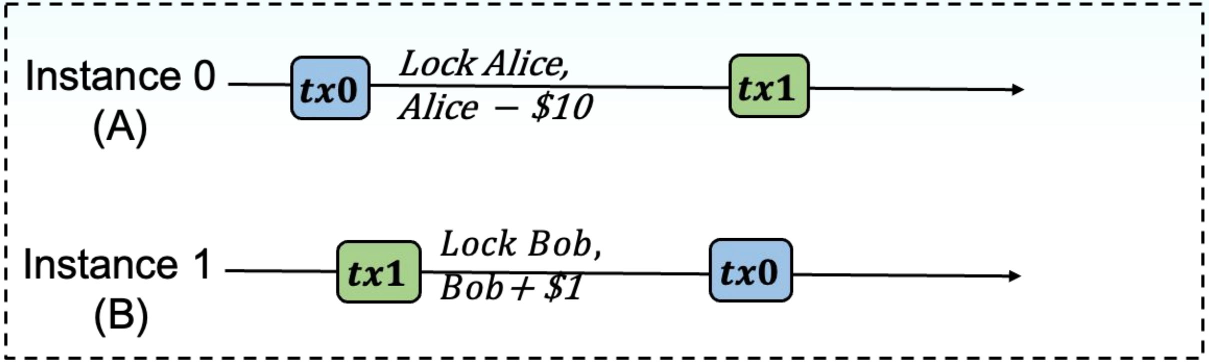




If execution on any object fails, roll back the successful ones and then unlock all objects.

Challenge 2: Locking may lead to deadlock.

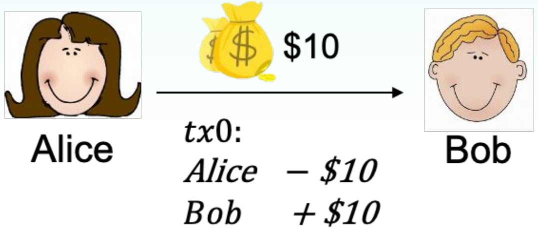


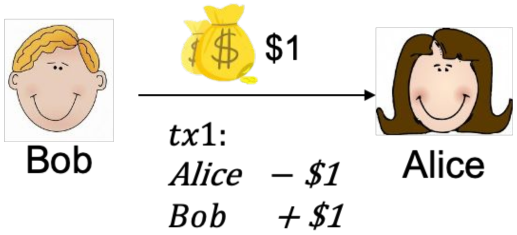


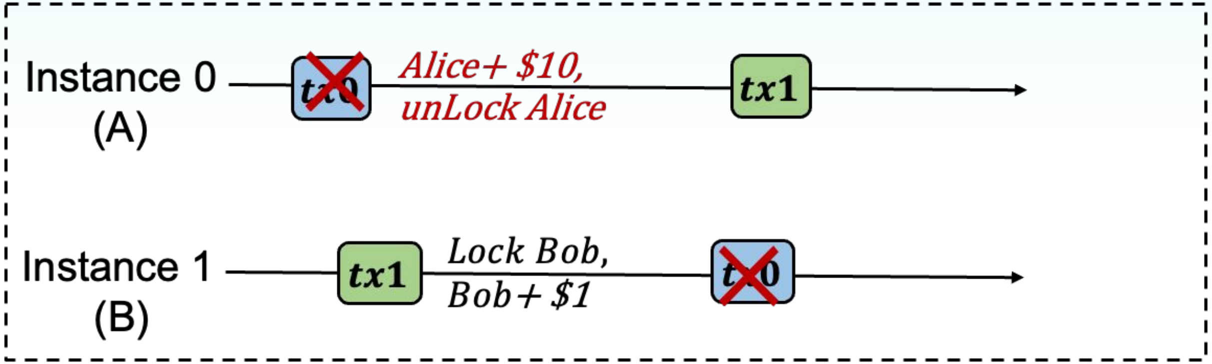
tx0: lock Alice, waiting for lock Bob

tx1: lock Bob, waiting for lock Alice

Solution: Detect **deadlock cycles** and deterministically abort transactions.

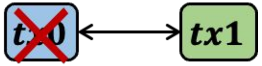






~~**tx0:** lock Alice, waiting for lock Bob~~

tx1: lock Bob, waiting for lock Alice



Evaluation Setting

Testbed:

- AWS EC2 c5a.2xlarge machines
- 8vCPUs and 16GB RAM Ubuntu Linux 22.04
- Deployed both in LAN and WAN

Baseline protocols:

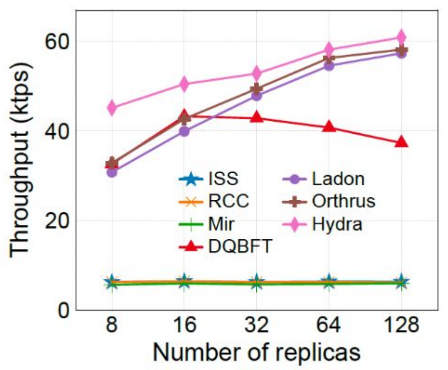
- ISS[1]
- Mir-BFT[2]
- RCC[3]
- DQBFT[4]
- Ladon[5]
- Orthrus[6]

Selected Research Questions:

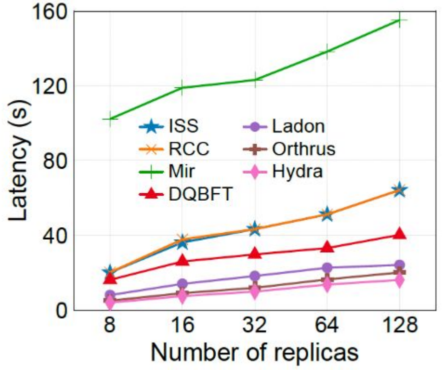
- How does Hydra perform as compared to the Baseline protocols?
- How does Hydra perform in varying proportions of cross-instance transactions?

[1] Stathakopoulou et. al. State Machine Replication Scalability Made Simple, in ACM EuroSys'22.
[2] Stathakopoulou et. al. Mir BFT: Scalable and Robust BFT for Decentralized Networks, in Jsys'22.
[3] Gupta et. al. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing, in IEEE ICDE'21.
[4] Balaji Arun and Binoy Ravindran. Scalable Byzantine Fault Tolerance via Partial Decentralization, in PVLDB'22.
[5] Lyu et. al. Ladon: High-Performance Multi-BFT Consensus via Dynamic Global Ordering, in ACM EuroSys'25.
[6] Lyu et. al. Orthrus: Accelerating Multi-BFT Consensus through Concurrent Partial Ordering of Transactions, in IEEE ICDE'25.

Scalability of Hydra and other Multi-BFT protocols



(a) #Straggler = 1, WAN



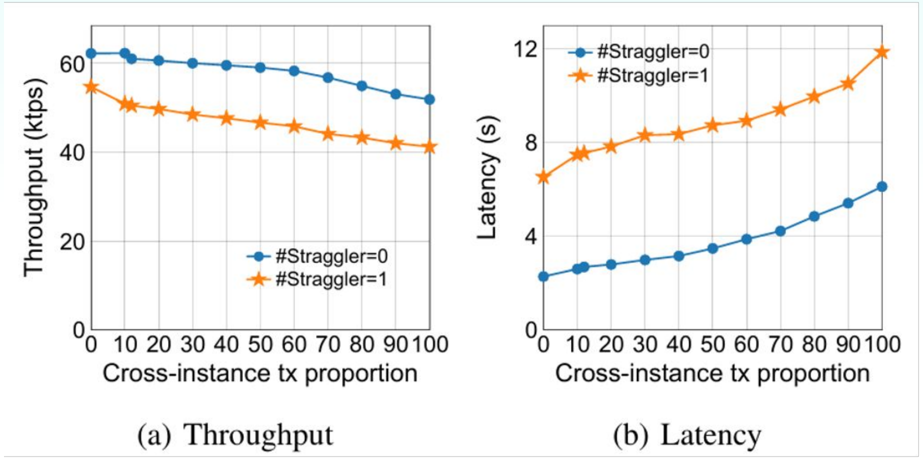
(b) #Straggler = 1, WAN

- 12%[1] cross-instance transactions
- 1 Stragglers
- WAN

- **With one straggler**
  - higher throughput **(9X)**
  - Lower latency

[1] Zhang et. al. TxAllo: Dynamic Transaction Allocation in Sharded Blockchain Systems, in IEEE ICDE'2023.

Performance of Hydra with different proportions of cross-instance transactions



- 0-100% cross-instance transactions
- 0 / 1 Stragglers
- WAN

- **With one straggler**
  - throughput decreases by 24.6%
- **Without stragglers**
  - throughput decreases by 16%

Contributions

Hydra: Concurrent Consensus Without **Global Ordering Barrier**

- Adopts an **Object-Centric** atomic execution model.
- **Eliminates the Global Ordering bottleneck** in Multi-BFT.