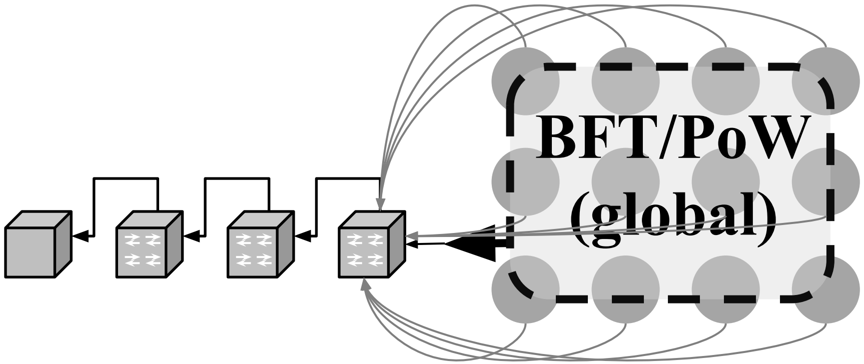
Kronos: A Secure and Generic Sharding Blockchain Consensus with Optimized Overhead

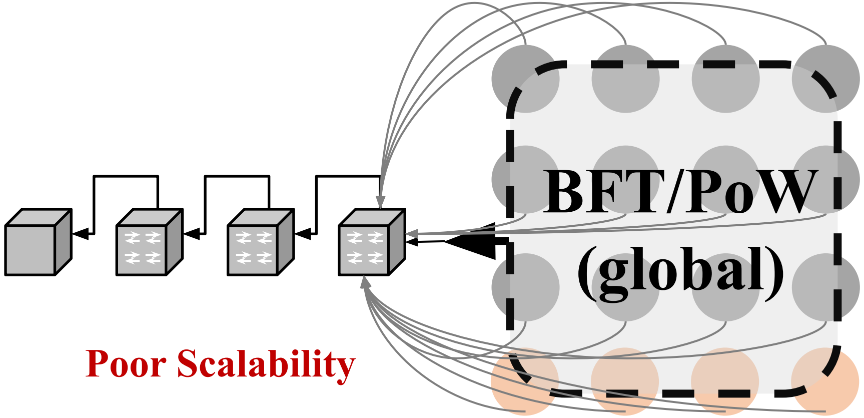
Yizhong Liu[1], Andi Liu[1], Yuan Lu[2], Zhuocheng Pan[1], Yinuo Li[3], Jianwei Liu[1],
Song Bian[1], **Mauro Conti[4]**

[1]Beihang University, [2]Institute of Software, Chinese Academy of Sciences
[3]Xi'an Jiaotong University, [4]University of Padua

1 Sharding Blockchain

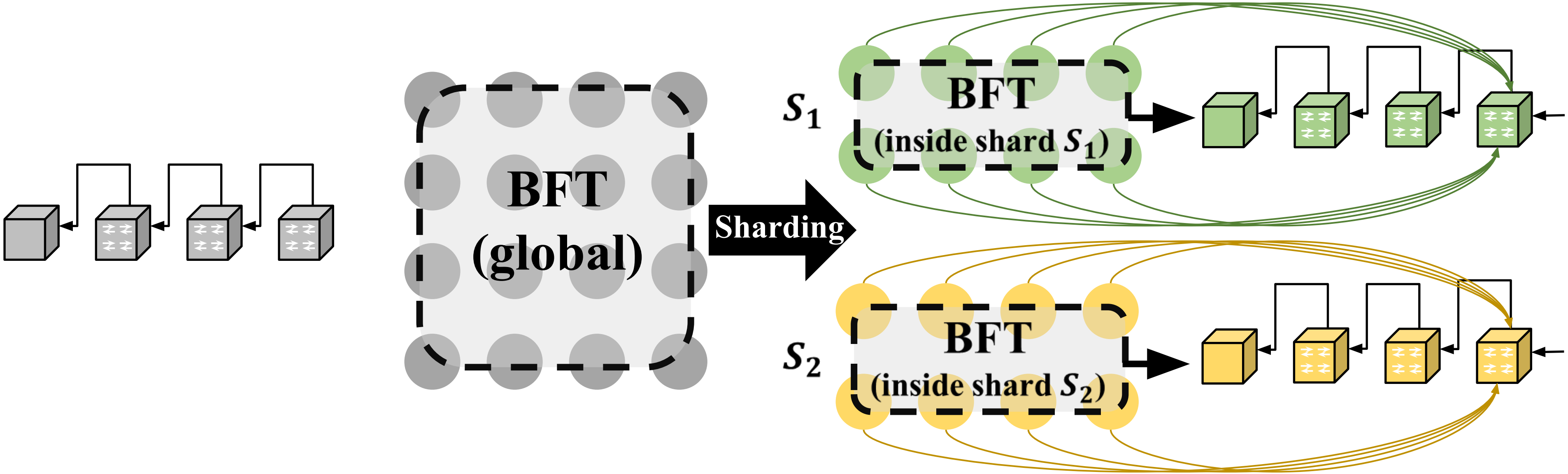


1 Sharding Blockchain



Poor Scalability

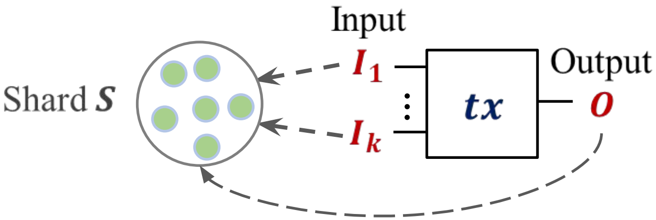
1 Sharding Blockchain



Improved Scalability

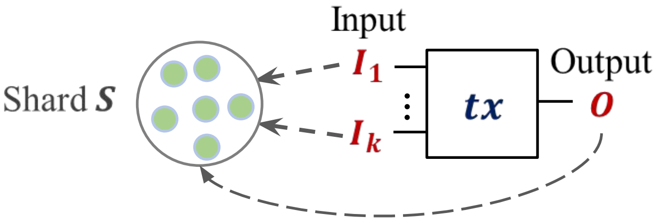
2 Cross-Shard Transaction

- **Intra-Shard Transaction:**

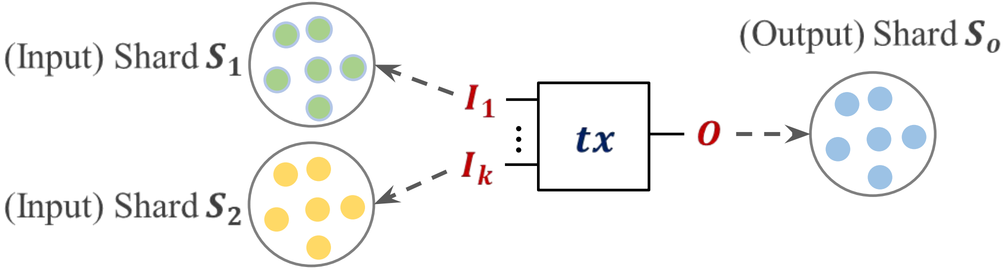


2 Cross-Shard Transaction

- **Intra-Shard Transaction:**

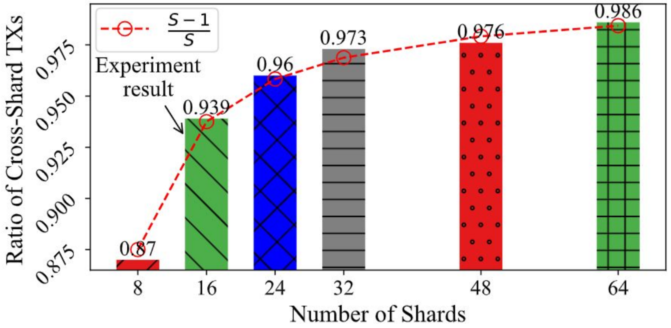


- **Cross-Shard Transaction:**



2 Cross-Shard Transaction

Why is Cross-Shard Transaction Critical?



Huang H, Peng X, Zhan J, et al. Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding[C]//IEEE INFOCOM, 2022: 1968-1977.

In Ethereum, the value of **multi-input transactions** (including high-value crowd-funding ones and consolidated payments) in 2024 has reached **1 billion USD**[1].

[1] https://coincodex.com/ico-calendar/ethereum/

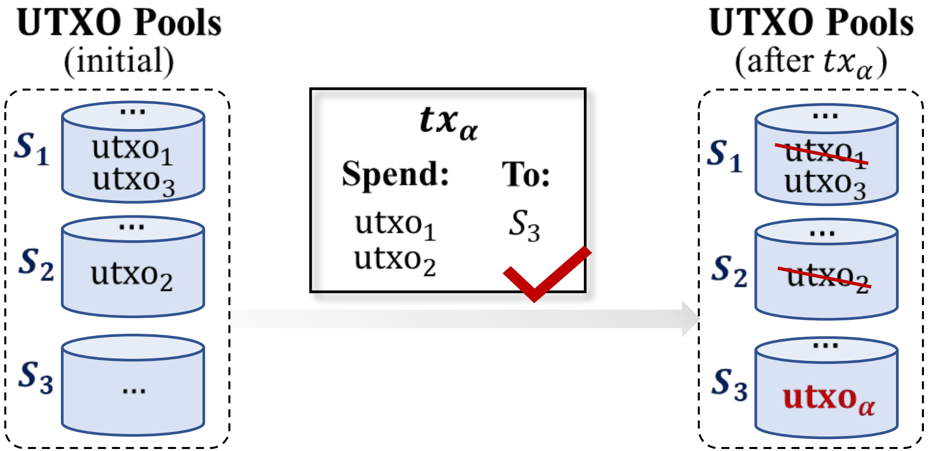
2 Cross-Shard Transaction

Cross-shard transaction processing dominating system **security**.

2 Cross-Shard Transaction

Cross-shard transaction processing dominating system **security**.

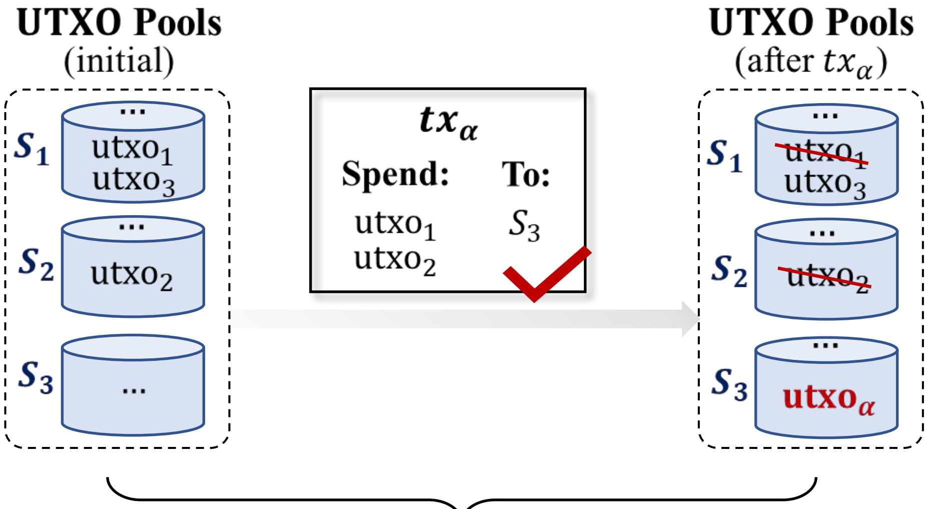
Critical security property: Atomicity



2 Cross-Shard Transaction

Cross-shard transaction processing dominating system **security**.

Critical security property: Atomicity

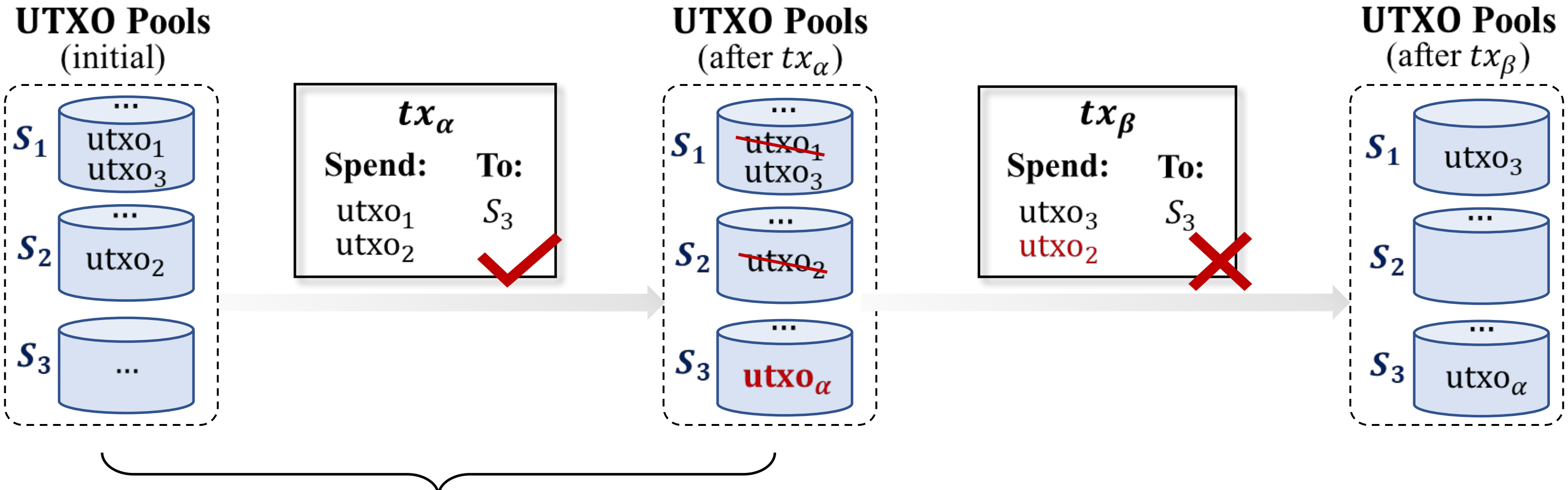


All involved shards commit valid requests.

2 Cross-Shard Transaction

Cross-shard transaction processing dominating system **security**.

Critical security property: Atomicity

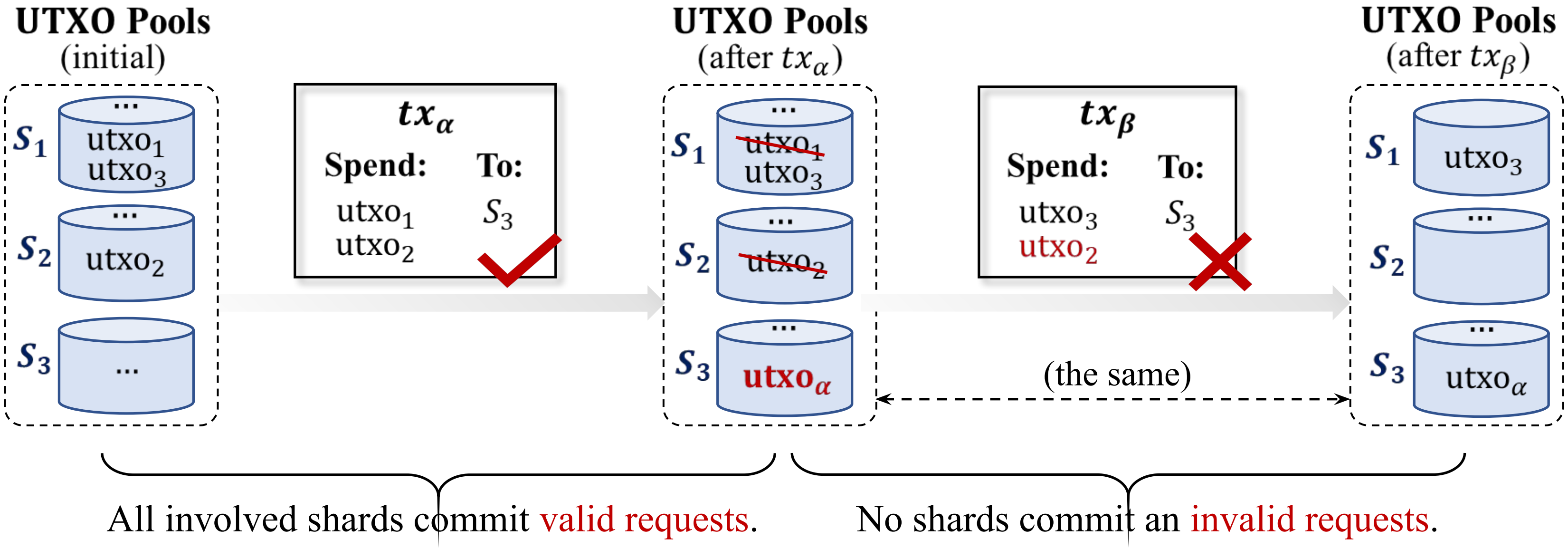


All involved shards commit valid requests.

2 Cross-Shard Transaction

Cross-shard transaction processing dominating system **security**.

Critical security property: Atomicity

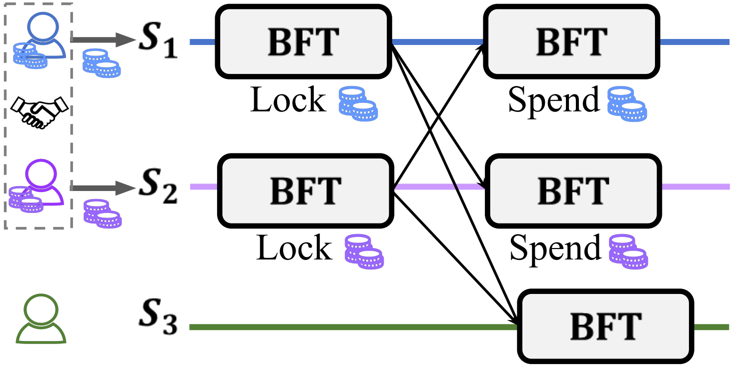


All involved shards commit valid requests.

No shards commit an invalid requests.

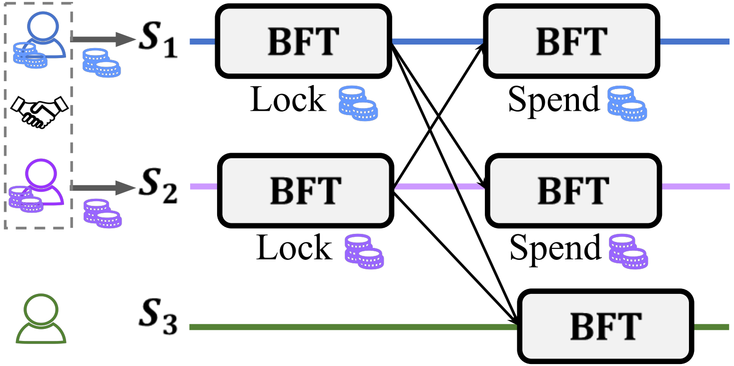
3 Remaining Issues of Prior Solutions

Existing solution: Two-Phase Commit (2PC)



3 Remaining Issues of Prior Solutions

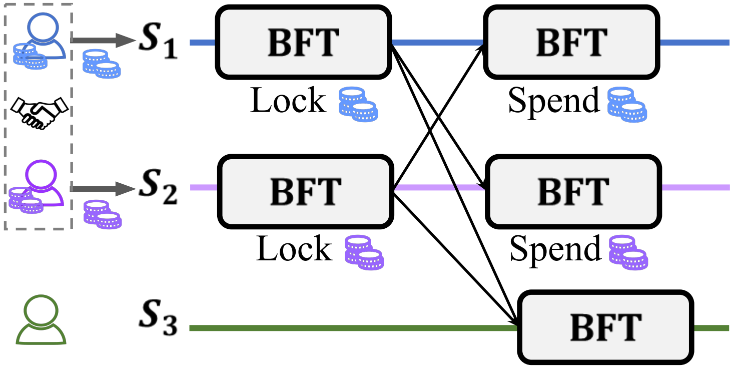
Existing solution: Two-Phase Commit (2PC)



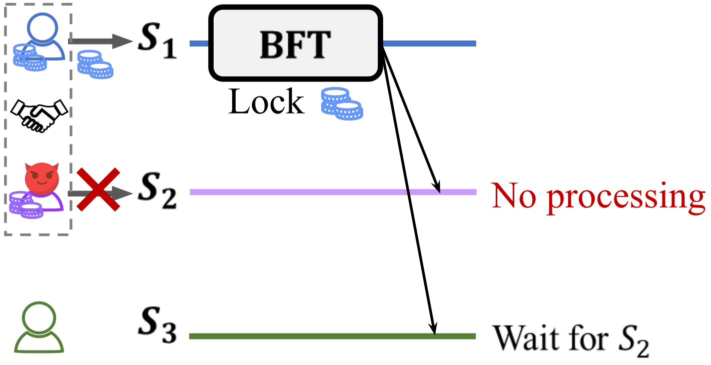
- 2PC ensures atomicity by locking mechanism, where each input shard must execute BFT 2 times.
- Directly spending available input through 1 BFT is low-cost, but easily compromises atomicity.

3 Remaining Issues of Prior Solutions

Existing solution: Two-Phase Commit (2PC)



- 2PC ensures atomicity by locking mechanism, where each input shard must execute BFT 2 times.
- Directly spending available input through 1 BFT is low-cost, but easily compromises atomicity.

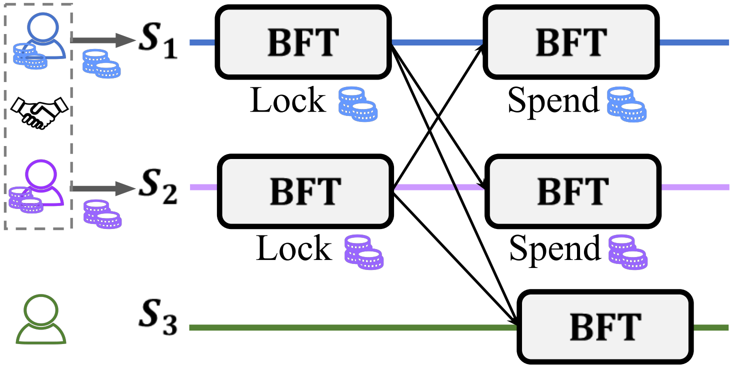


- 2PC **cannot withstand silence attack**, where malicious clients selectively send requests to some shards while neglecting others.
- Timeouts is inapplicable to asynchronous ones.

3 Remaining Issues of Prior Solutions

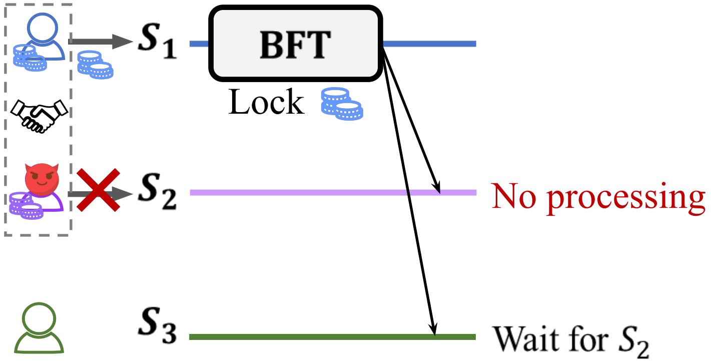
Existing solution: Two-Phase Commit (2PC)

Extra BFT execution → High overhead



- 2PC ensures atomicity by locking mechanism, where each input shard must execute BFT 2 times.
- Directly spending available input through 1 BFT is low-cost, but easily compromises atomicity.

Forever lock → Weak atomicity

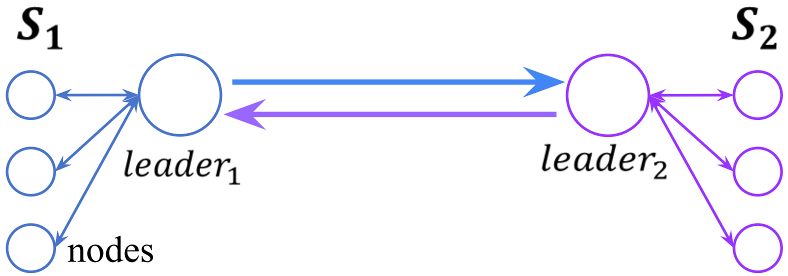


- 2PC **cannot withstand silence attack**, where malicious clients selectively send requests to some shards while neglecting others.
- Timeouts is inapplicable to asynchronous ones.

3 Remaining Issues of Prior Solutions

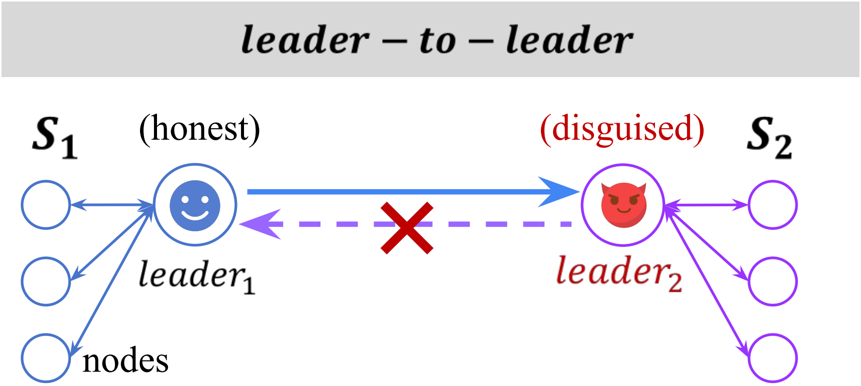
Existing cross-shard certification:

$\boldsymbol{leader - to - leader}$



3 Remaining Issues of Prior Solutions

Existing cross-shard certification:

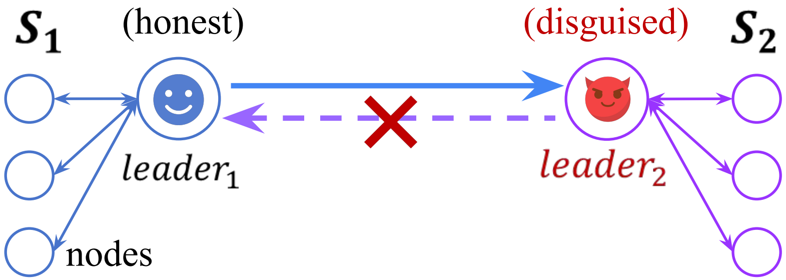


- **Disguise Attack**: Malicious shard leader does not send crucial messages (proof) to relevant shards, or forward messages from other shards to nodes within its shard.

3 Remaining Issues of Prior Solutions

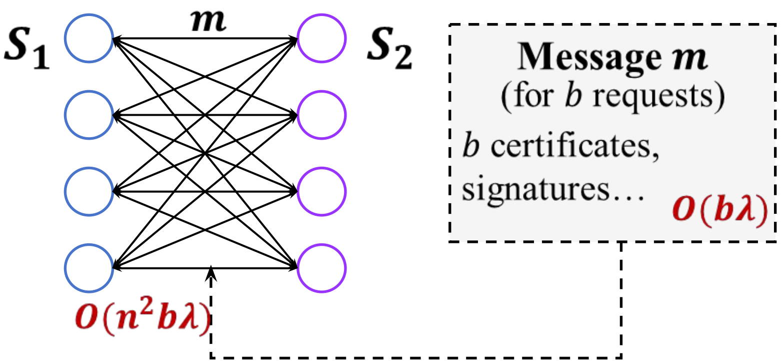
Existing cross-shard certification:

$\boldsymbol{leader - to - leader}$



- **Disguise Attack**: Malicious shard leader does not send crucial messages (proof) to relevant shards, or forward messages from other shards to nodes within its shard.

$\boldsymbol{O(n) - to - O(n)}$



n: shard size, b: transaction number, λ: security parameter

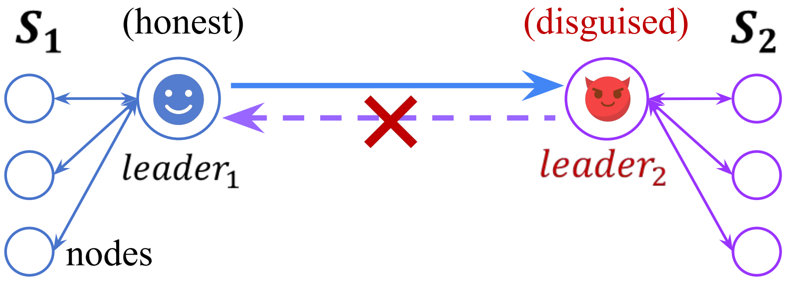
$O(n)$-to-$O(n)$? **Expensive communication!**

- Cross-shard communication overhead for processing b transactions: $CS\text{-}\omega = O(n^2b\lambda)$

3 Remaining Issues of Prior Solutions

Existing cross-shard certification:

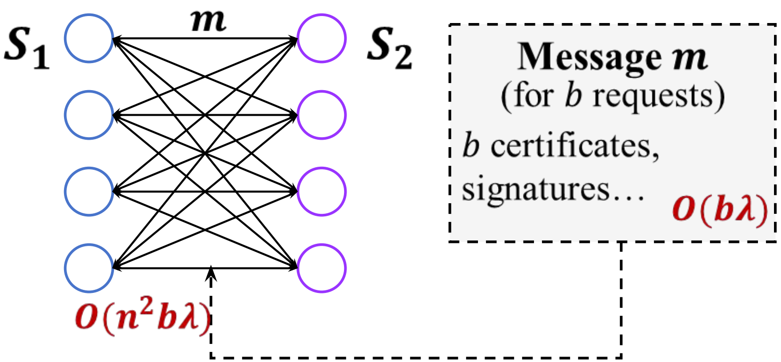
leader – to – leader



- **Disguise Attack**: Malicious shard leader does not send crucial messages (proof) to relevant shards, or forward messages from other shards to nodes within its shard.

Forever wait → Unreliable communication

$O(n) – to – O(n)$



$O(n)$-to-$O(n)$? **Expensive communication!**

- Cross-shard communication overhead for processing b transactions: $CS\text{-}\omega = O(n^2b\lambda)$

Heavy message → Inefficient communication

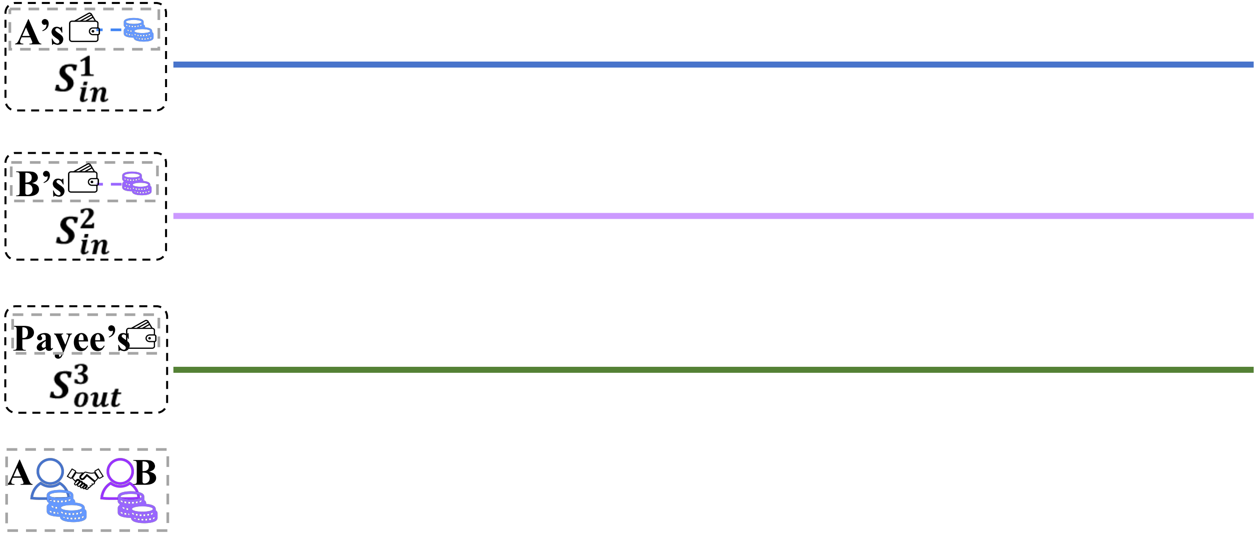
3 Remaining Issues of Prior Solutions

Can we design a generic sharding blockchain consensus achieving security and efficiency with optimized overhead?

4.1 Transaction Processing Pattern of Kronos

Valid transaction processing pattern

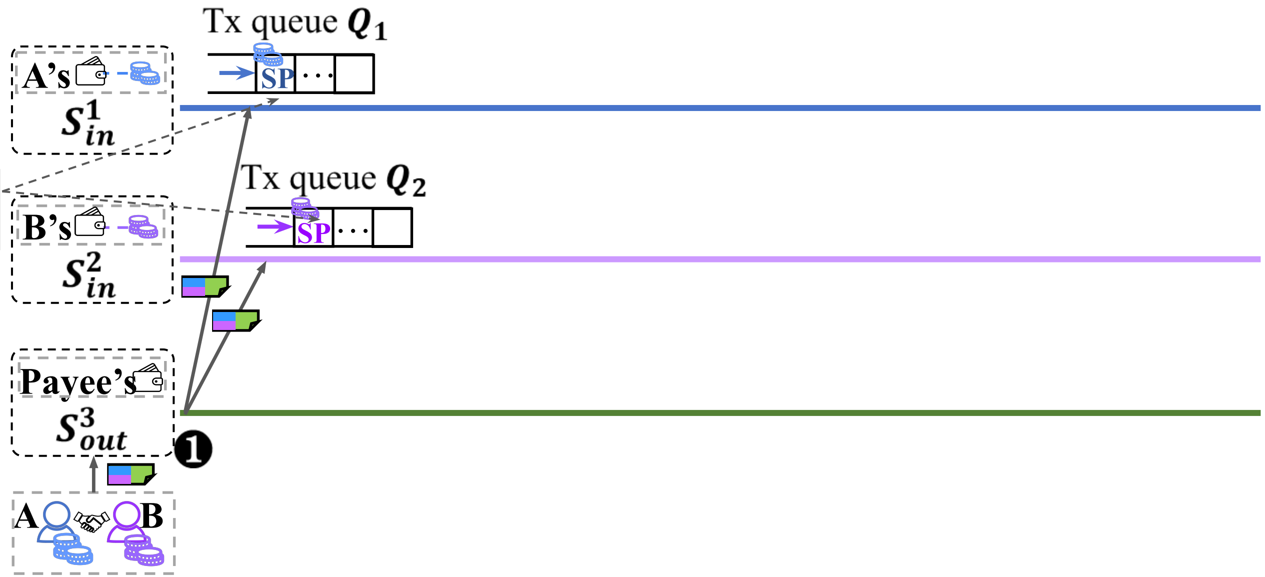
*Each shard is managed with a majority (e.g., 2/3) of honest nodes.



4.1 Transaction Processing Pattern of Kronos

Valid transaction processing pattern

Step ❶
Request delivery



SP: spend-transaction to spend available coins

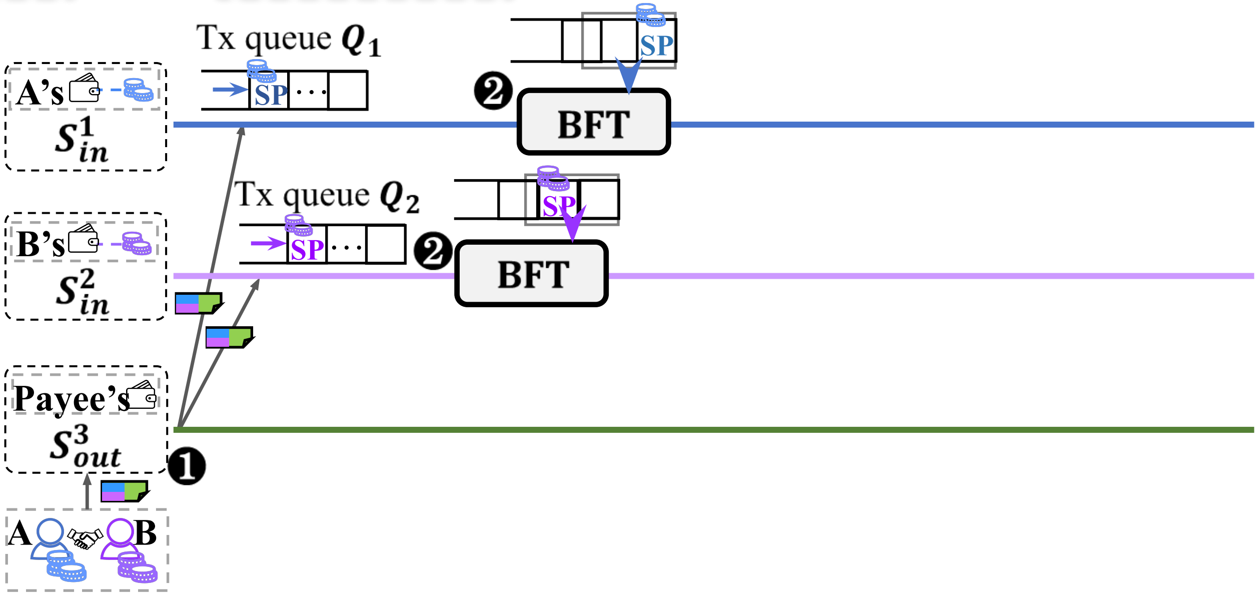
4.1 Transaction Processing Pattern of Kronos

Valid transaction processing pattern

Step ❶
Request delivery

⇨

Step ❷
Available input
spending

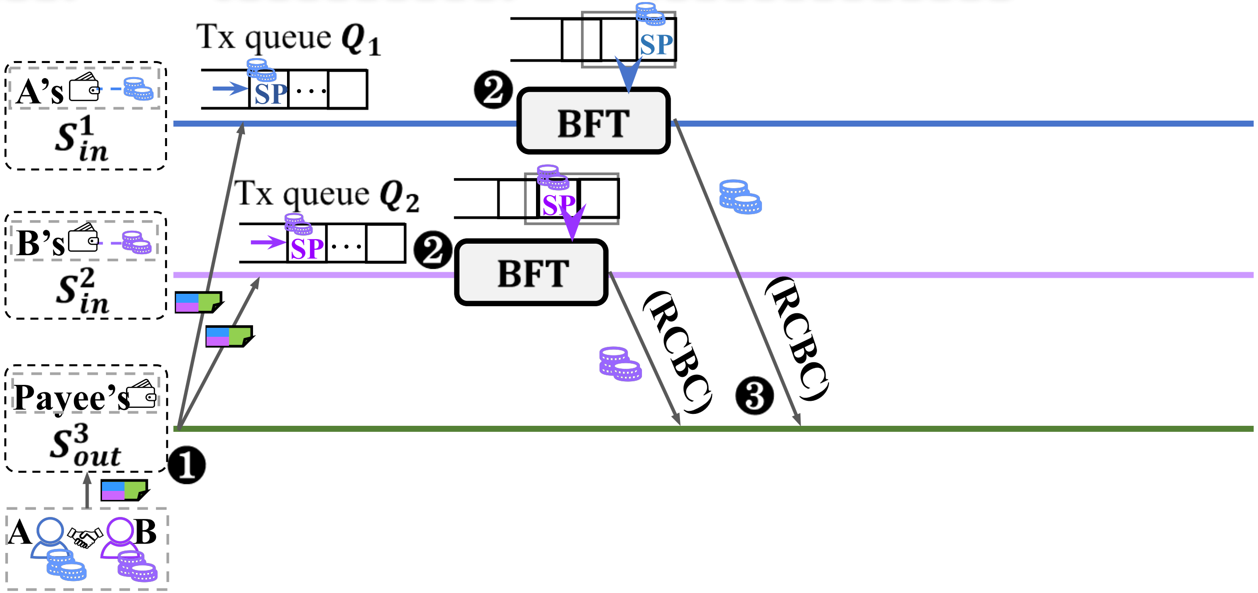


4.1 Transaction Processing Pattern of Kronos

Valid transaction processing pattern

*RCBC ensures that cross-shard messages arrive at the output shard.

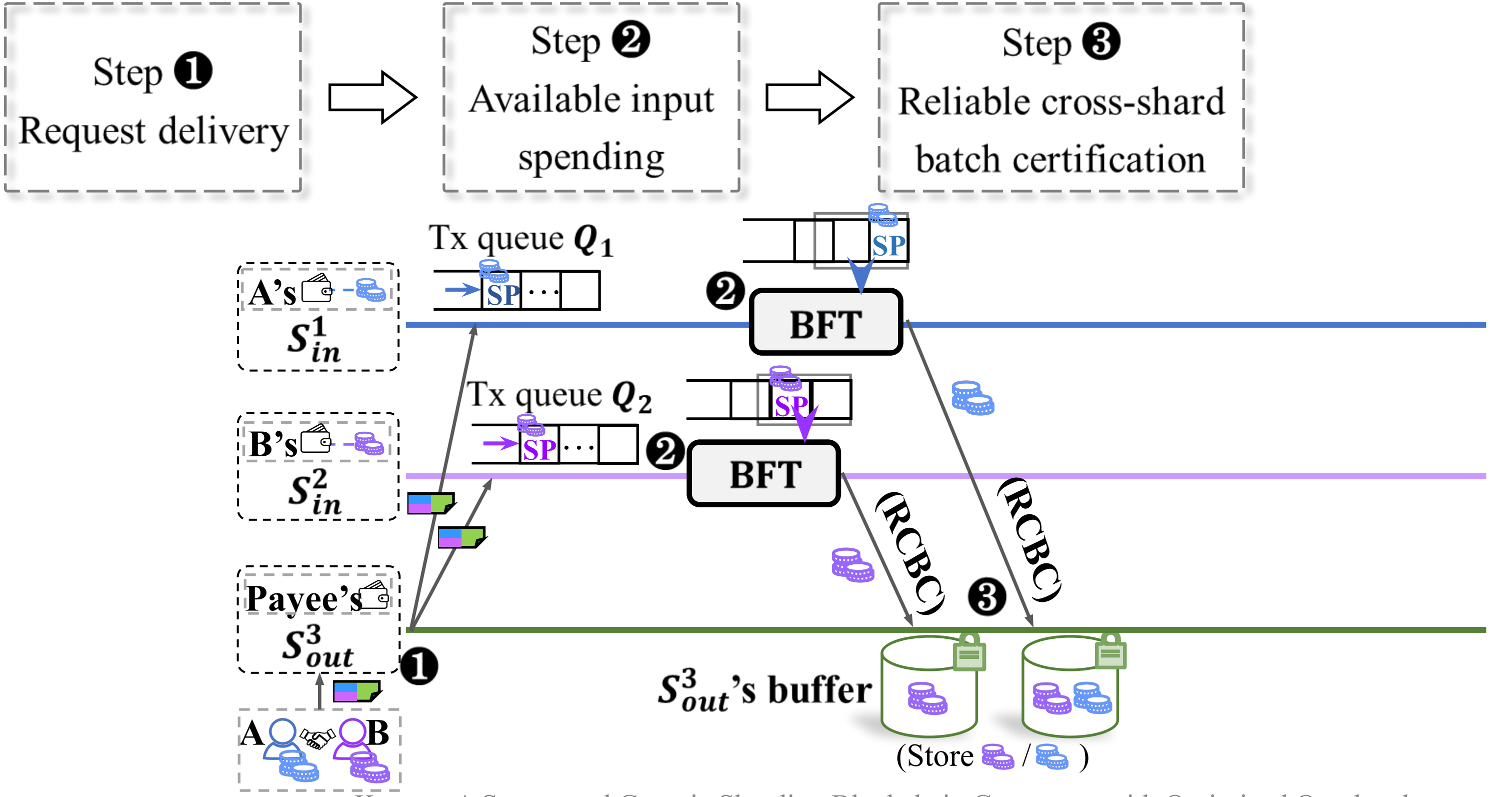
Step ❶ Request delivery → Step ❷ Available input spending → Step ❸ Reliable cross-shard batch certification



4.1 Transaction Processing Pattern of Kronos

Valid transaction processing pattern

*RCBC ensures that cross-shard messages arrive at the output shard.



Step ❶ Request delivery

Step ❷ Available input spending

Step ❸ Reliable cross-shard batch certification

4.1 Transaction Processing Pattern of Kronos

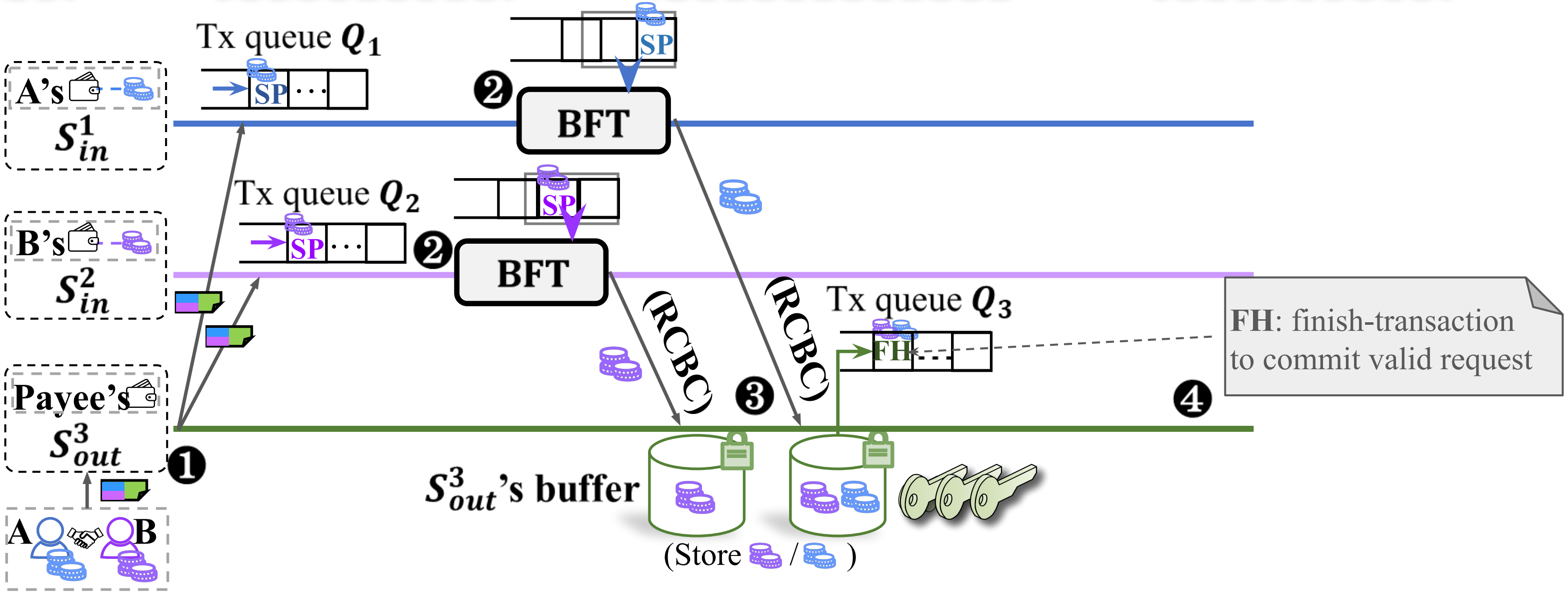
Valid transaction processing pattern

Step ❶
Request delivery

Step ❷
Available input spending

Step ❸
Reliable cross-shard batch certification

Step ❹
Valid request finalization

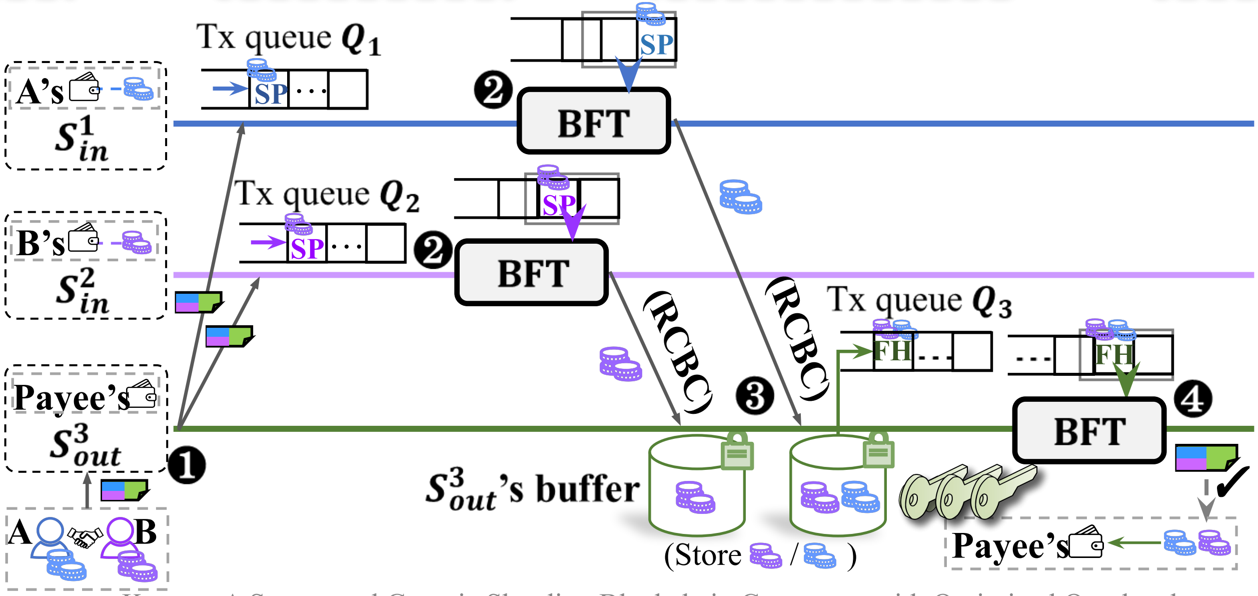


FH: finish-transaction to commit valid request

4.1 Transaction Processing Pattern of Kronos

Valid transaction processing pattern

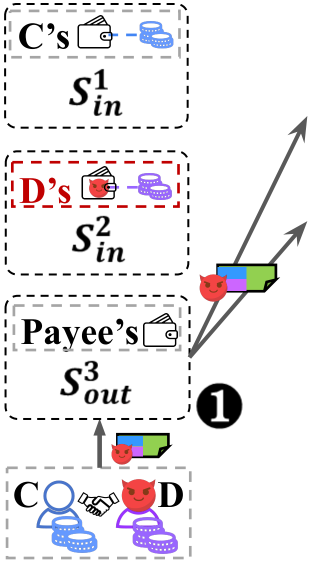
Step ❶ Request delivery → Step ❷ Available input spending → Step ❸ Reliable cross-shard batch certification → Step ❹ Valid request finalization



4.1 Transaction Processing Pattern of Kronos

□ Invalid transaction processing pattern

Step ❶
Request delivery



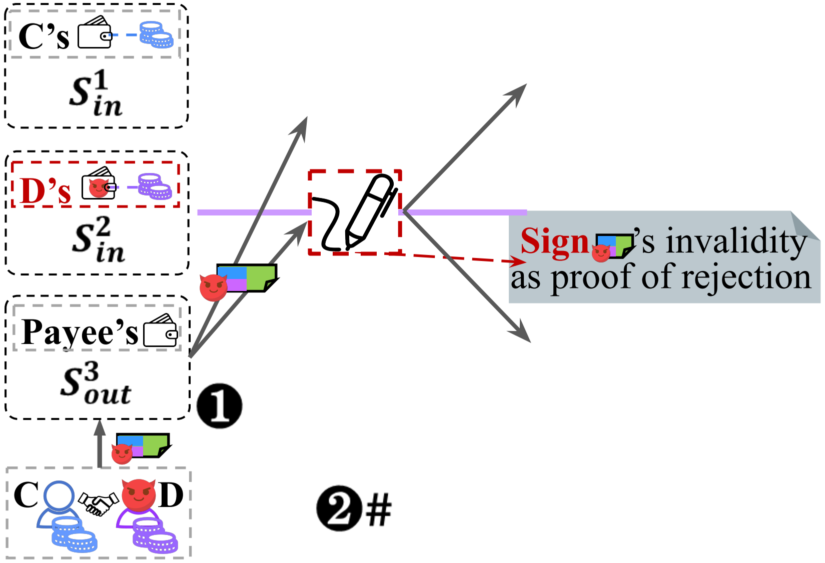
4.1 Transaction Processing Pattern of Kronos

- **Invalid transaction processing pattern**

Step ❶
Request delivery

⇨

Step ❷#
Unavailable input proving



Sign 's invalidity
as proof of rejection

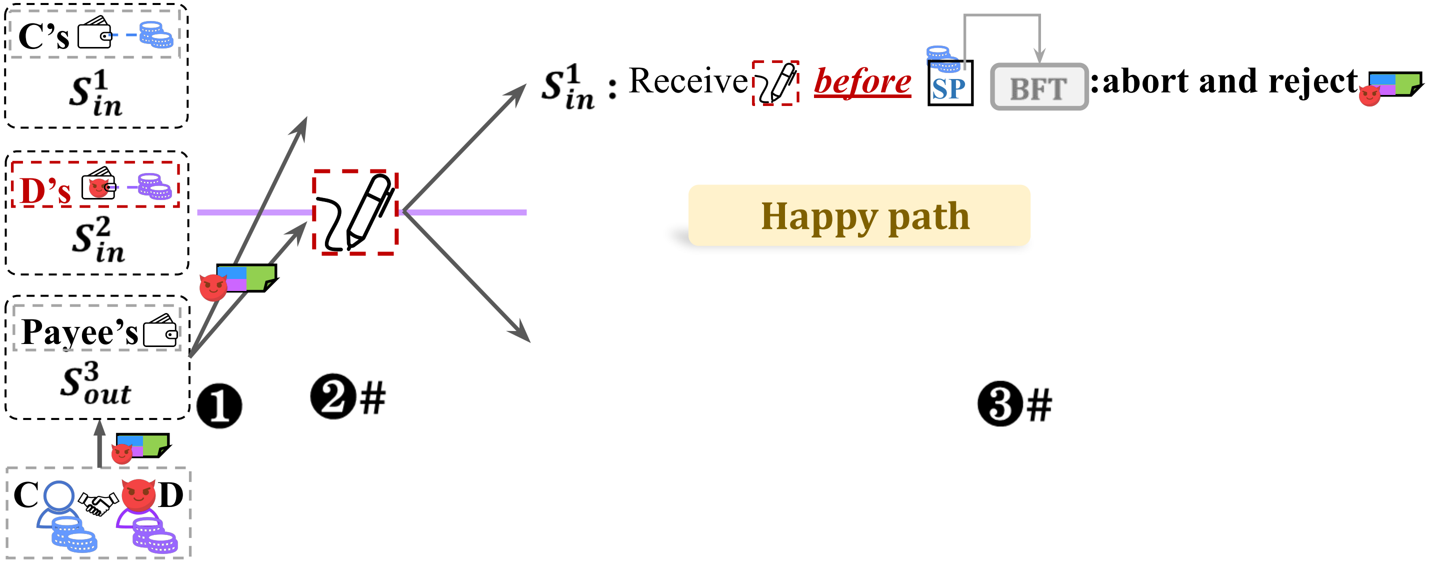
4.1 Transaction Processing Pattern of Kronos

- **Invalid transaction processing pattern**

Step ❶
Request delivery

Step ❷#
Unavailable input proving

Step ❸#
Invalid request rejection



S_{in}^{1} : Receive ***before*** SP BFT **:abort and reject**

4.1 Transaction Processing Pattern of Kronos

Invalid transaction processing pattern

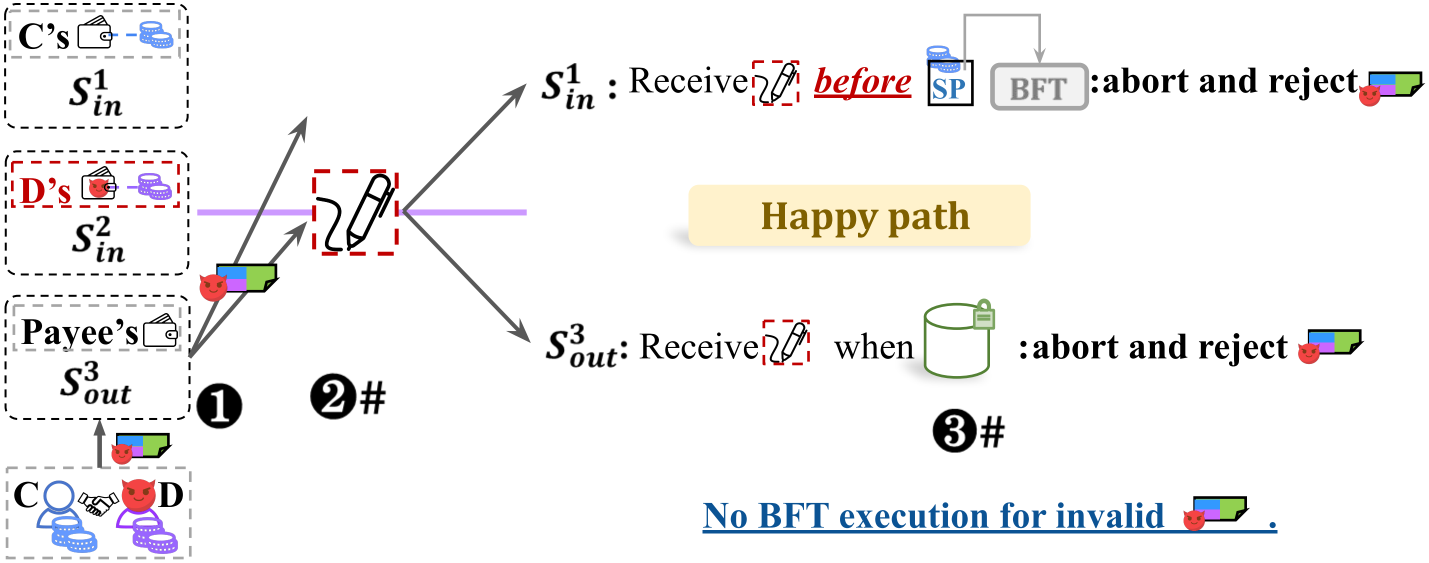
Step ❶
Request delivery

⇨

Step ❷#
Unavailable input proving

⇨

Step ❸#
Invalid request rejection



S^1_{in} : Receive ✍ ***before*** SP BFT :**abort and reject**

Happy path

S^3_{out}: Receive ✍ when :**abort and reject**

No BFT execution for invalid .

4.1 Transaction Processing Pattern of Kronos

▯ Invalid transaction processing pattern

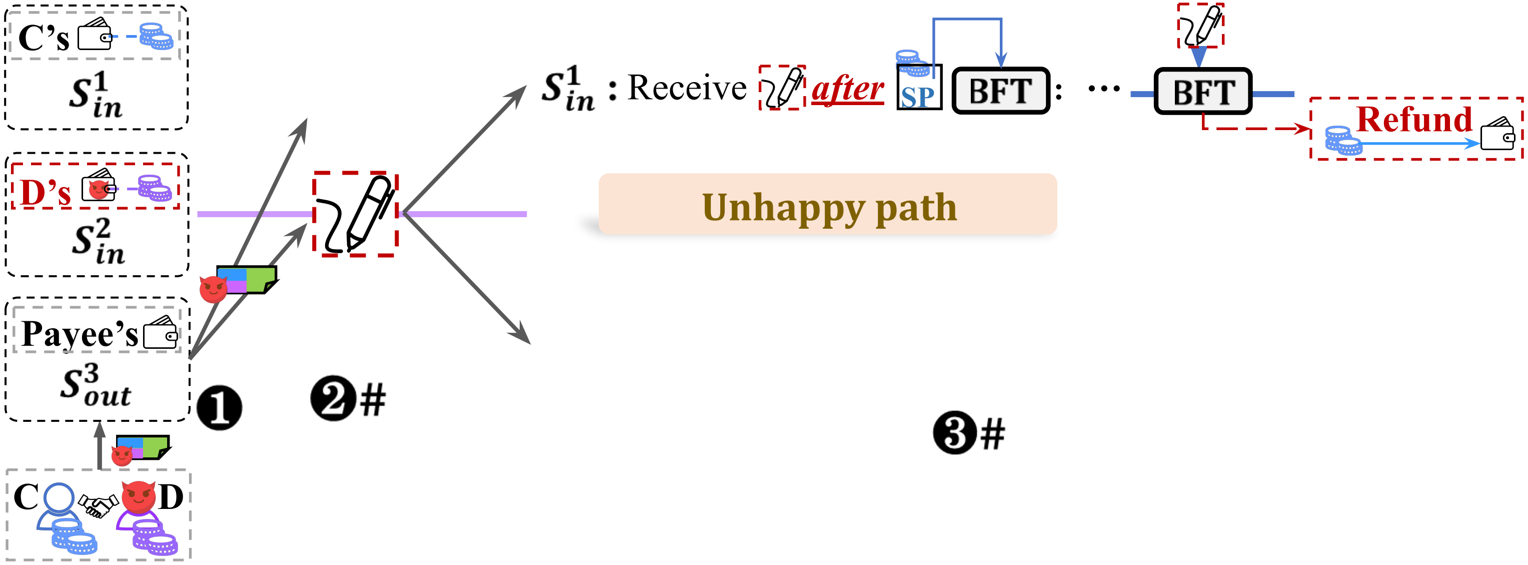
Step ❶
Request delivery

⇨

Step ❷#
Unavailable input proving

⇨

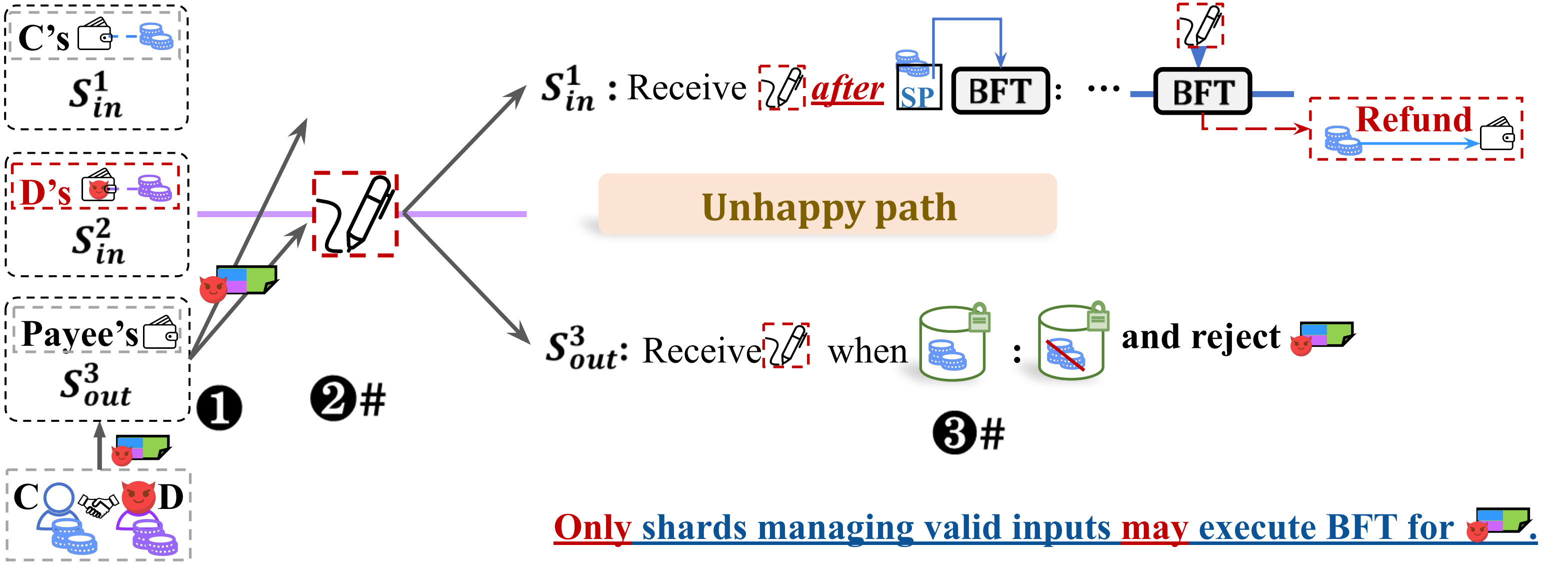
Step ❸#
Invalid request rejection



4.1 Transaction Processing Pattern of Kronos

Invalid transaction processing pattern

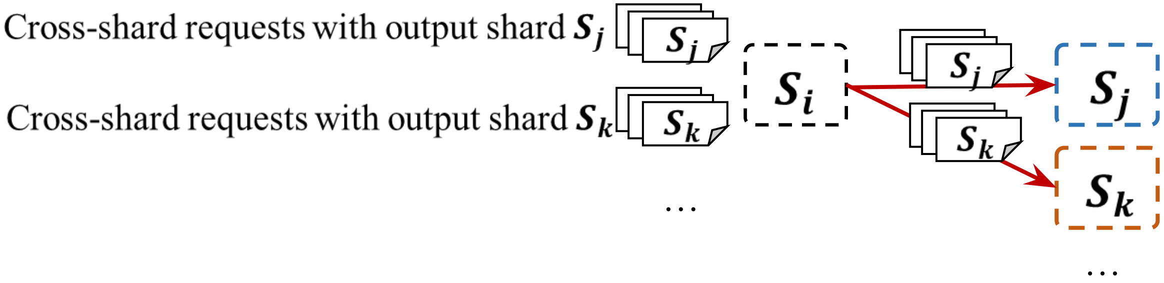
Step ❶ Request delivery ⇨ Step ❷# Unavailable input proving ⇨ Step ❸# Invalid request rejection



C's
S_{in}^1
D's
S_{in}^2
Payee's
S_{out}^3
C
D
❶
❷#
S_{in}^1 : Receive after SP BFT : ··· BFT
Refund
Unhappy path
S_{out}^3: Receive when : and reject
❸#

Only shards managing valid inputs may execute BFT for .

4.2 Reliable Cross-Shard Batch Certification (RCBC)



Cross-shard requests with output shard S_j

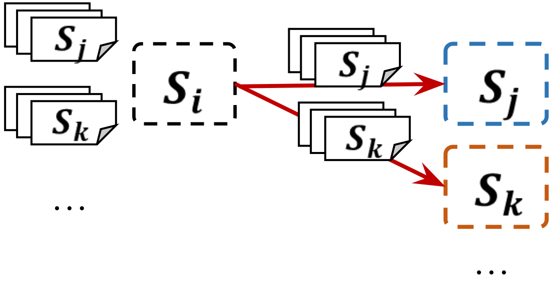
Cross-shard requests with output shard S_k

…

4.2 Reliable Cross-Shard Batch Certification (RCBC)

Method: Hybrid-tree-based RCBC (HT-RCBC) :

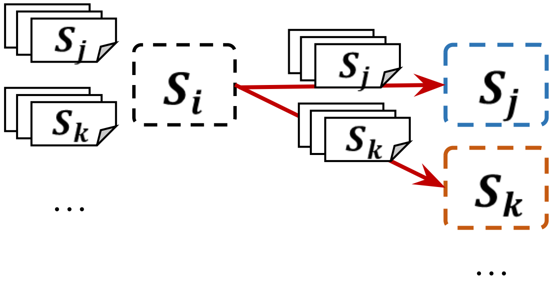
- Merkle tree + Erasure coding



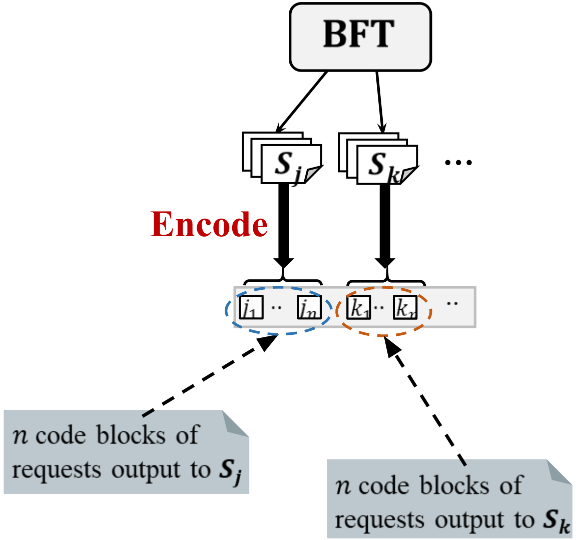
4.2 Reliable Cross-Shard Batch Certification (RCBC)

Method: Hybrid-tree-based RCBC (HT-RCBC) :

- Merkle tree + Erasure coding



Input shard S_i: Encode + Commit



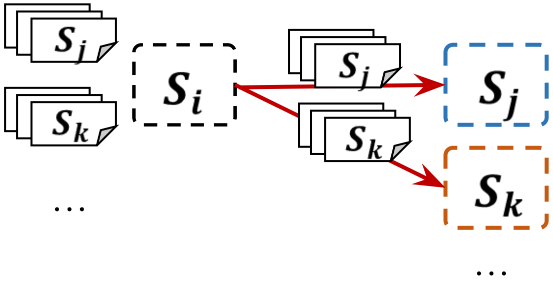
n code blocks of requests output to S_j

n code blocks of requests output to S_k

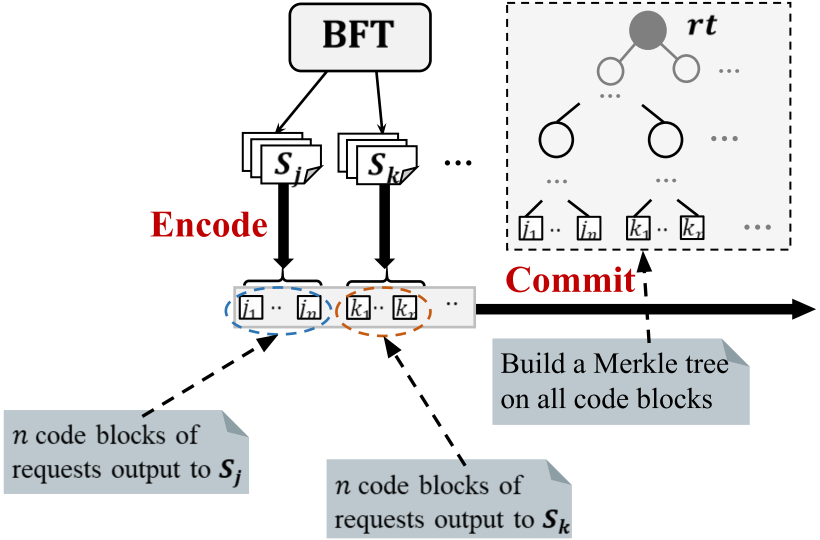
4.2 Reliable Cross-Shard Batch Certification (RCBC)

Method: Hybrid-tree-based RCBC (HT-RCBC) :

- Merkle tree + Erasure coding



Input shard S_i: Encode + Commit



Build a Merkle tree on all code blocks

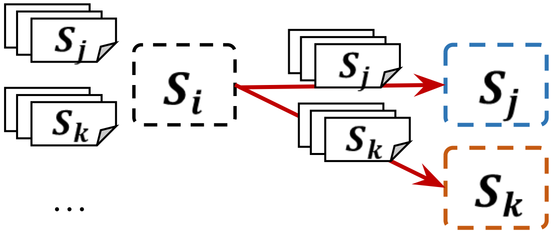
n code blocks of requests output to S_j

n code blocks of requests output to S_k

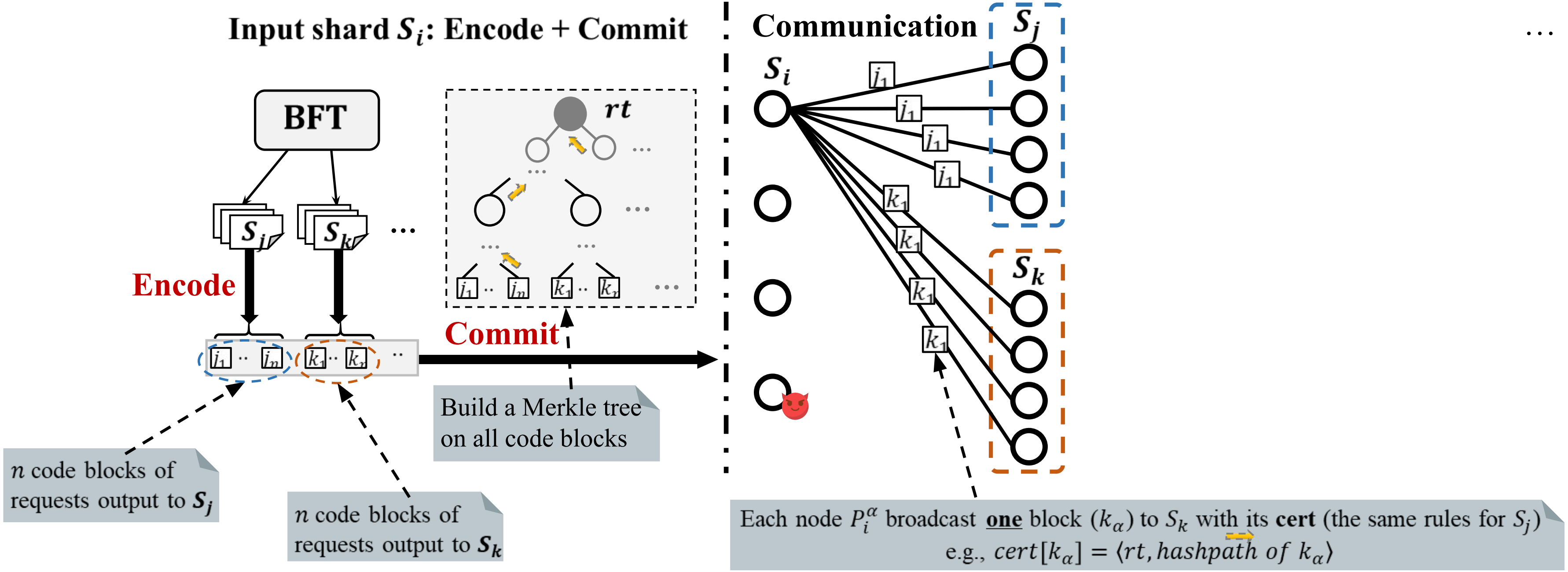
4.2 Reliable Cross-Shard Batch Certification (RCBC)

Method: Hybrid-tree-based RCBC (HT-RCBC) :

- Merkle tree + Erasure coding



Cross-Shard Communication



n code blocks of requests output to S_j

n code blocks of requests output to S_k

Build a Merkle tree on all code blocks

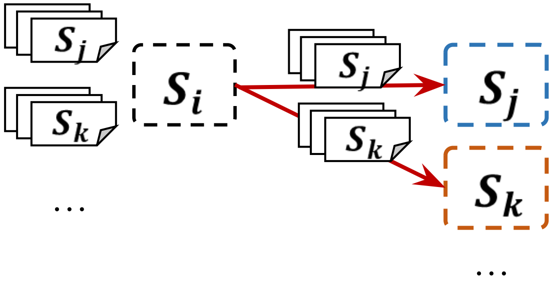
Each node P_i^α broadcast **one** block (k_α) to S_k with its **cert** (the same rules for S_j)

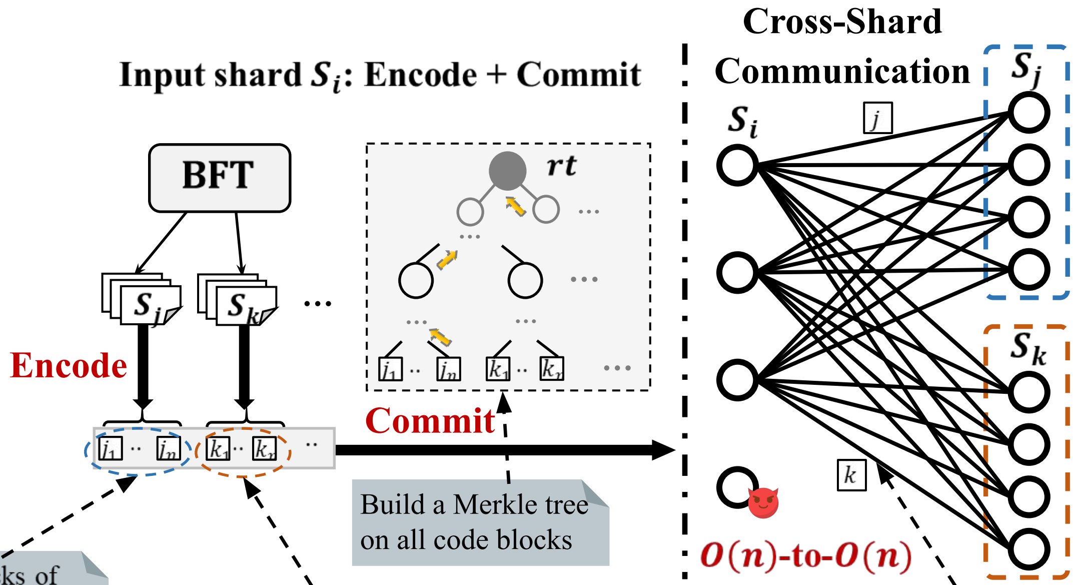
e.g., $cert[k_\alpha] = \langle rt, hashpath\ of\ k_\alpha \rangle$

4.2 Reliable Cross-Shard Batch Certification (RCBC)

Method: Hybrid-tree-based RCBC (HT-RCBC) :

- Merkle tree + Erasure coding





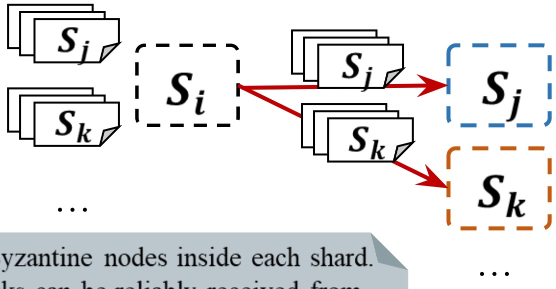
n code blocks of requests output to S_j

n code blocks of requests output to S_k

Build a Merkle tree on all code blocks

Each node P_i^{α} broadcast **one** block (k_α) to S_k with its **cert** (the same rules for S_j)
e.g., $cert[k_\alpha] = \langle rt, hashpath\ of\ k_\alpha \rangle$

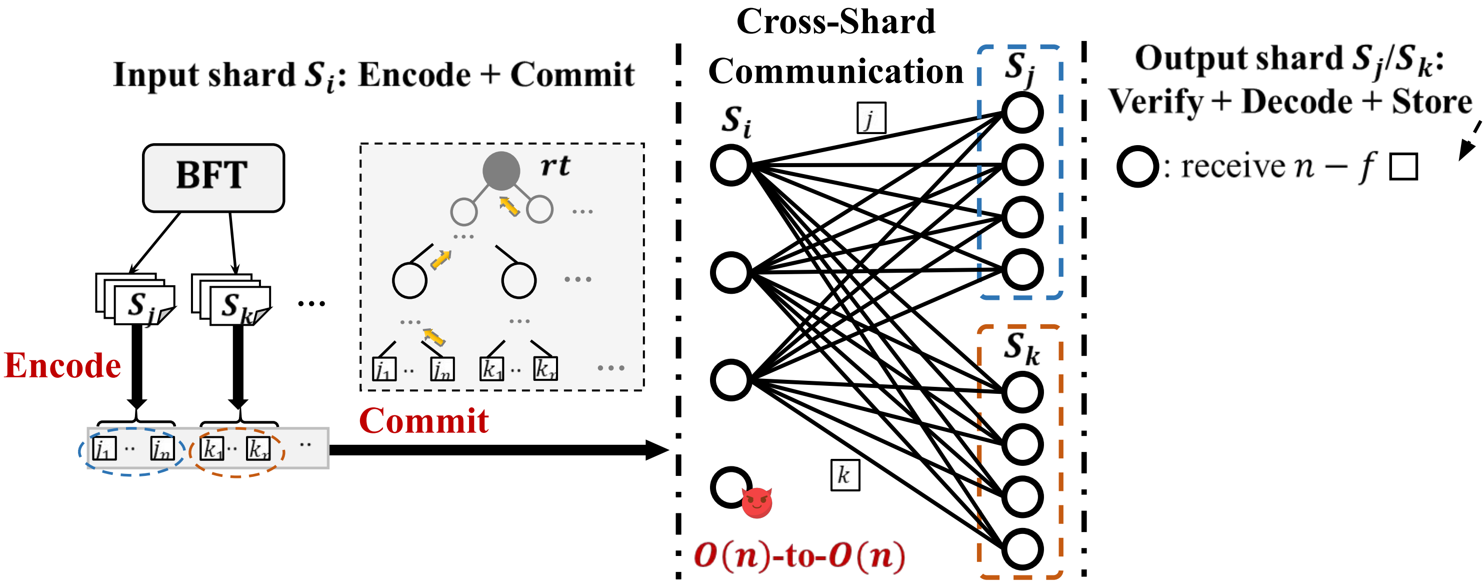
4.2 Reliable Cross-Shard Batch Certification (RCBC)



Method: Hybrid-tree-based RCBC (HT-RCBC) :

- Merkle tree + Erasure coding

f: the number of Byzantine nodes inside each shard. At least $n - f$ blocks can be reliably received from **the majority of honest nodes** in S_i

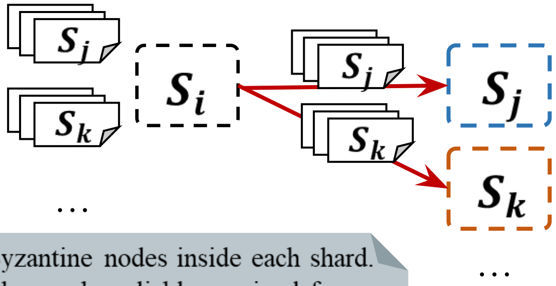

Input shard S_i: Encode + Commit

Cross-Shard Communication

Output shard S_j/S_k: Verify + Decode + Store

○: receive $n - f$ □

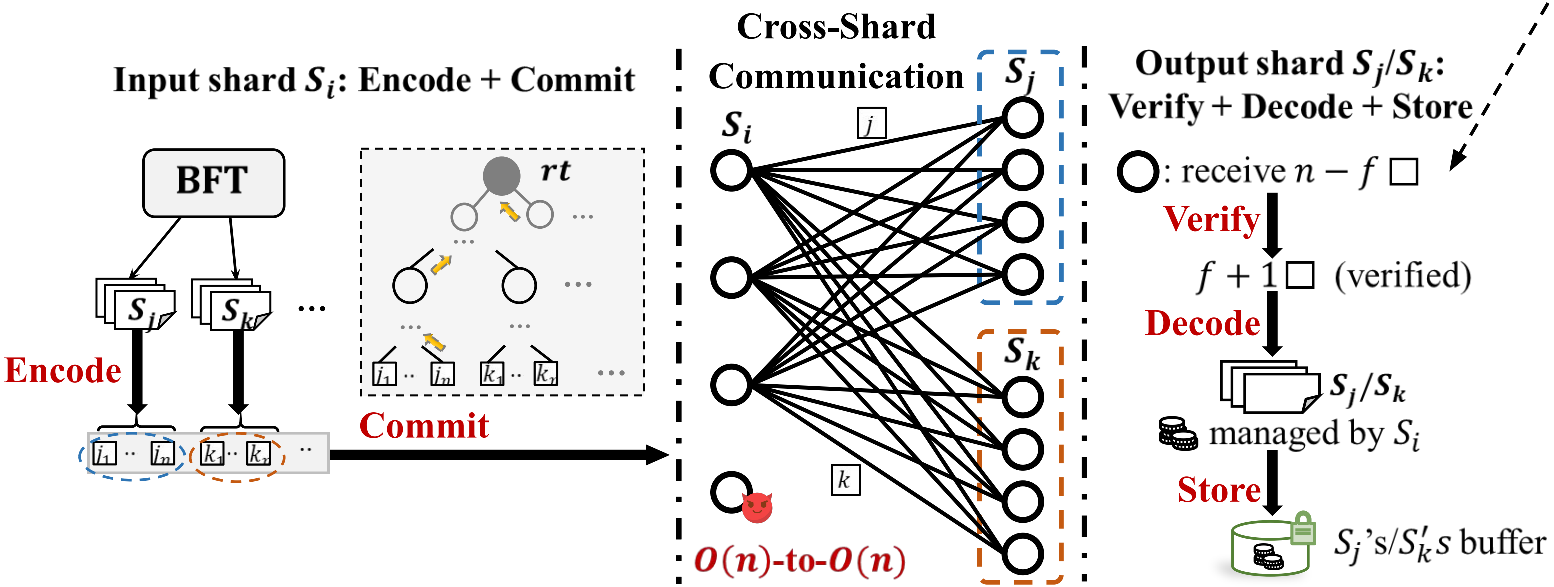
4.2 Reliable Cross-Shard Batch Certification (RCBC)



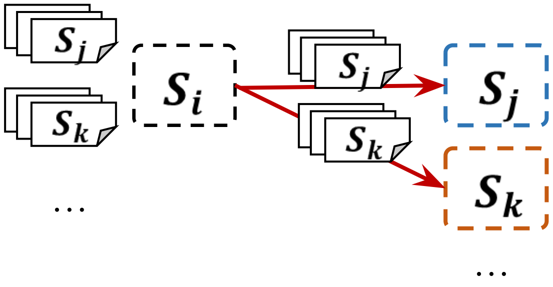
Method: Hybrid-tree-based RCBC (HT-RCBC) :

- Merkle tree + Erasure coding

f: the number of Byzantine nodes inside each shard. At least $n-f$ blocks can be reliably received from **the majority of honest nodes** in S_i

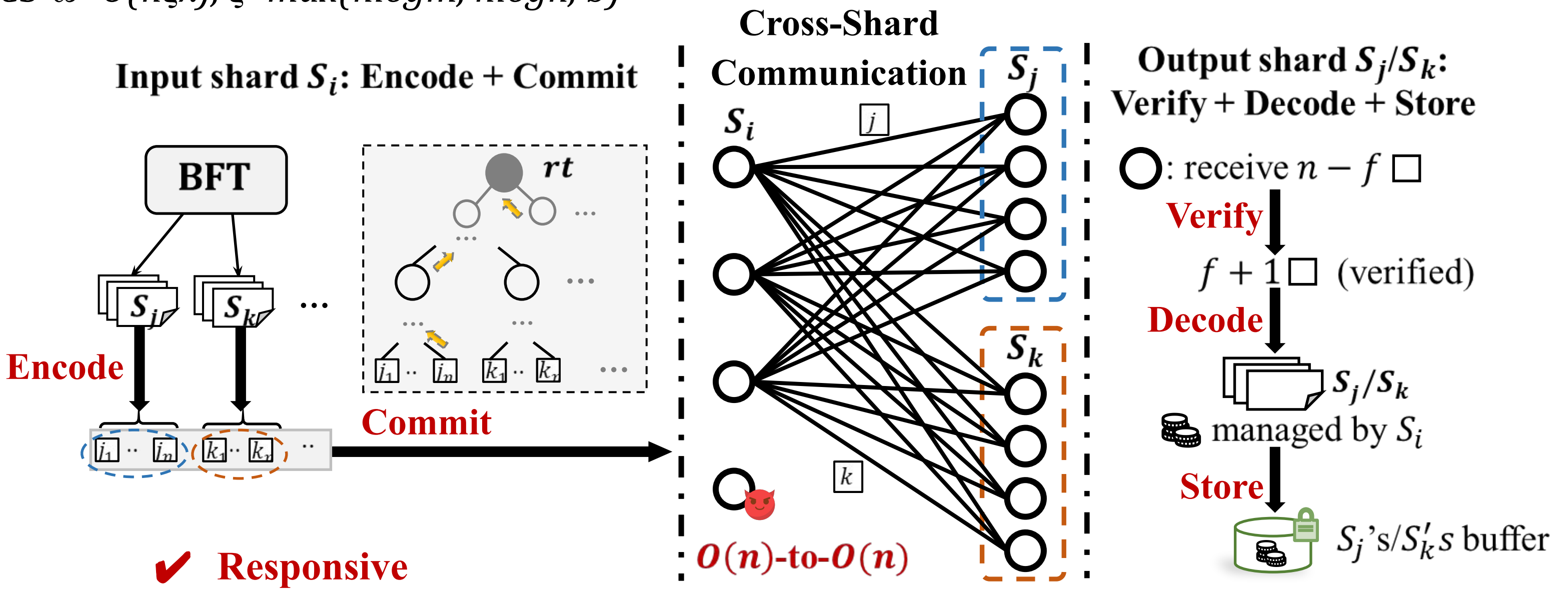


4.2 Reliable Cross-Shard Batch Certification (RCBC)



Method: Hybrid-tree-based RCBC (HT-RCBC) :

- Merkle tree + Erasure coding
- *CS-ω=O(nξλ), ξ=max(nlogm, nlogn, b)*



Input shard S_i: Encode + Commit

Cross-Shard Communication

Output shard S_j/S_k: Verify + Decode + Store

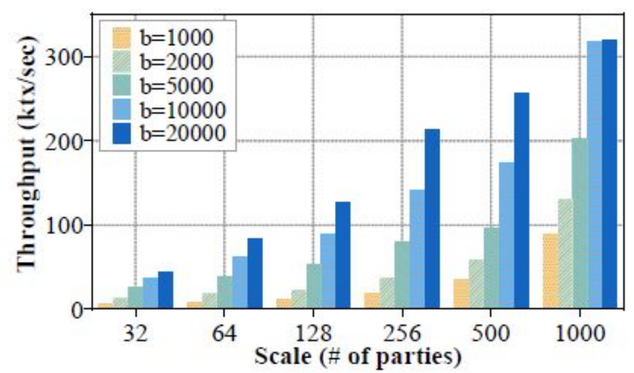
✔ **Responsive**

✔ **In batch:**

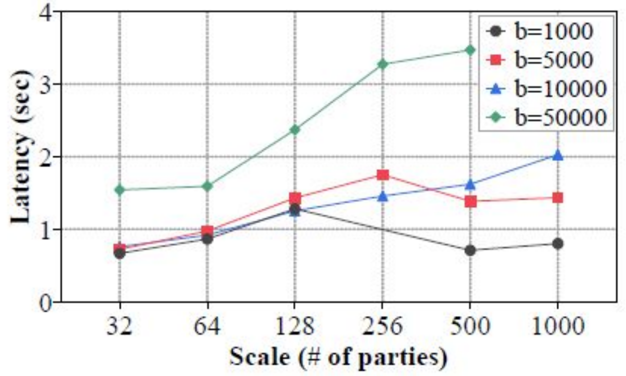
✔ **Reliable**

5 Evaluation

- **Implementation: using Speeding Dumbo[1] (asynchronous BFT) or Hotstuff[2] (partially synchronous BFT) for intra-shard consensus**
- **Setting: 32 to 1000 nodes running in AWS EC2 instances**
- **Results: Averaged over 5 experimental runs**



(a) Peak throughput



(b) Latency

Scalability: ✓ (Throughput increases as network size N scales to 1000).

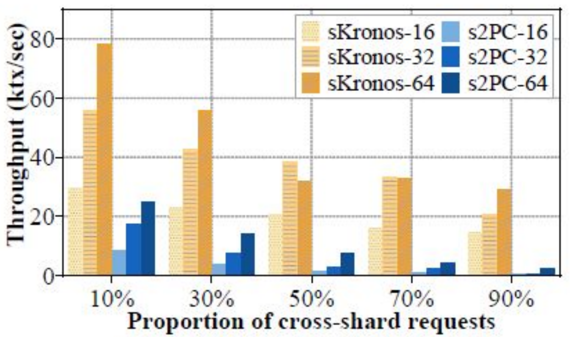
[1] B. Guo, Y. Lu, Z. Lu, et al., , "Speeding dumbo: Pushing asynchronous BFT closer to practice," in NDSS'22. ISOC, 2022.

[2] M. Yin, D. Malkhi, M. K. Reiter et al., "Hotstuff: Bft consensus with linearity and responsiveness," in PODC'19. ACM, 2019, pp. 347–356.

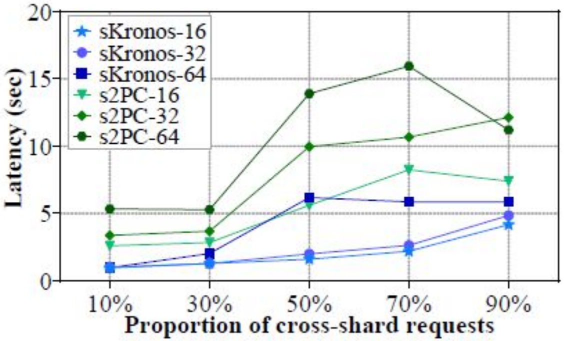
5 Evaluation

Comparison with 2PC:

Number of shards: 16, 32, and 64, with 4 nodes per shard (optimal shard size in tests).



(b) sKronos v.s. s2PC (TPS)



(c) sKronos v.s. s2PC (latency)

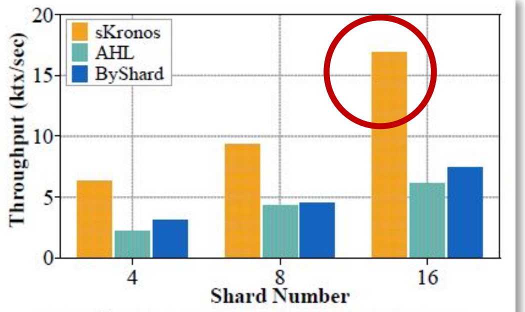
- **sKronos outperforms s2PC**

(TPS: up to 12×, Latency: nearly 1/2).

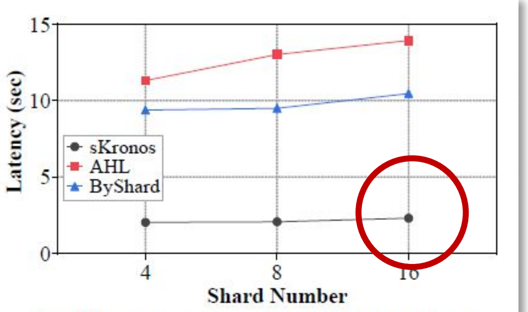
- **sKronos:** Kronos using Speeding Dumbo for intra-shard consensus.
- **s2PC:** 2PC using Speeding Dumbo for intra-shard consensus.

5 Evaluation

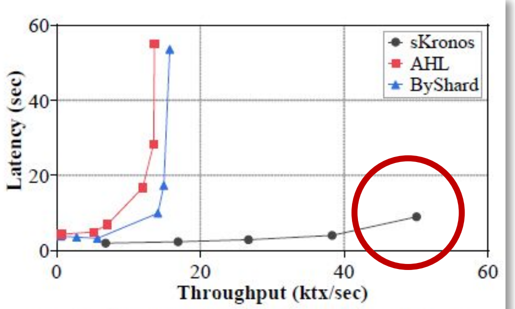
Comparison with other sharding blockchains



(a) sKronos v.s. AHL and ByShard (TPS)



(b) sKronos v.s. AHL and ByShard (latency)



(c) sKronos v.s. AHL and ByShard (trade-off)

Kronos outperforms AHL[1] and ByShard[2] in all cases (TPS: 2.3× (ByShard), 2.7× (AHL), Latency: below 1/3).

[1] H. Dang, T. T. A. Dinh, D. Loghin et al., “Towards scaling blockchain systems via sharding,” in SIGMOD’19. ACM, 2019, pp. 123–140

[2] J. Hellings and M. Sadoghi, “Byshard: sharding in a byzantine environment,” VLDB J., vol. 32, no. 6, pp. 1343–1367, 2023.

6 Conclusion

System	Malicious Leader Tolerance	Malicious Client Tolerance	Atomicity	IS-Overhead	CS-Overhead	Genericity
Kronos-HT **Kronos-VC***	✓	✓	✓	kB	$O(n\xi\lambda)$ $O(nb\lambda)$	(Partially) Sync./**Async.**

$\xi = max(nlogm, nlogn, b)$

Kronos-HT: Kronos with HT-RCBC.

* Kronos-VC: A variant of Kronos-HT that uses *vector commitments* instead of Merkle trees to commit code blocks.

6 Conclusion

System	Malicious Leader Tolerance	Malicious Client Tolerance	Atomicity	IS-Overhead	CS-Overhead	Genericity
Kronos-HT **Kronos-VC**	✓	✓	✓	kB	$O(n\xi\lambda)$ $O(nb\lambda)$	(Partially) Sync./**Async.**

Atomicity under malicious leader and client and optimal intra-shard overhead

6 Conclusion

System	Malicious Leader Tolerance	Malicious Client Tolerance	Atomicity	IS-Overhead	CS-Overhead	Genericity
Kronos-HT Kronos-VC	✓	✓	✓	kB	$O(n\xi\lambda)$ $O(nb\lambda)$	(Partially) Sync./**Async.**

Atomicity under malicious leader and client and optimal intra-shard overhead

Reliable cross-shard transfer with low communication overhead

6 Conclusion

System	Malicious Leader Tolerance	Malicious Client Tolerance	Atomicity	IS-Overhead	CS-Overhead	Genericity
Kronos-HT **Kronos-VC**	✓	✓	✓	kB	$O(n\xi\lambda)$ $O(nb\lambda)$	(Partially) Sync./**Async.**

Atomicity under malicious leader and client and optimal intra-shard overhead

Reliable cross-shard transfer with low communication overhead

Genericity under asynchronous network and scalability for exiting BFT protocols

Thank you!

Questions?

Comparison

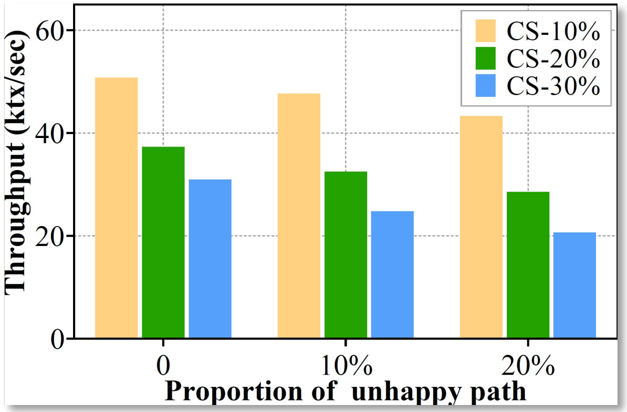
System	Malicious Leader Tolerance	Malicious Client Tolerance	Atomicity	IS-Overhead	CS-Overhead	Genericity
Omniledger[1]	×	×		$2kB$	$O(b(logb+\lambda))$	Partially Sync.
Chainspace[2]	✓	×		$2kB$	$O(n^2b\lambda)$	Partially Sync.
ByShard[3]	✓	×		$2kB$	$O(n^2b\lambda)$	Sync.
RapidChain[4]	×	✓	⍻	kB	$O(n^2b\lambda)$	Sync.
Sharper[5]	×	✓		-	$O(n^2b\lambda)$	Partially Sync.
AHL[6]	✓	×		$(2k+3)B$	$O(n^2b\lambda)$	Partially Sync.
Pyramid[7]	✓	×		$(k+1)B$	$O(n^2b\lambda)$	Partially Sync.
Monoxide[8]	×	×	⍻	$kB, k=2$	$O(nb\lambda)$	Partially Sync.
Kronos-HT **Kronos-VC**	✓	✓	✓	kB	$O(n\xi\lambda)$ $O(nb\lambda)$	(Partially) Sync./**Async.**

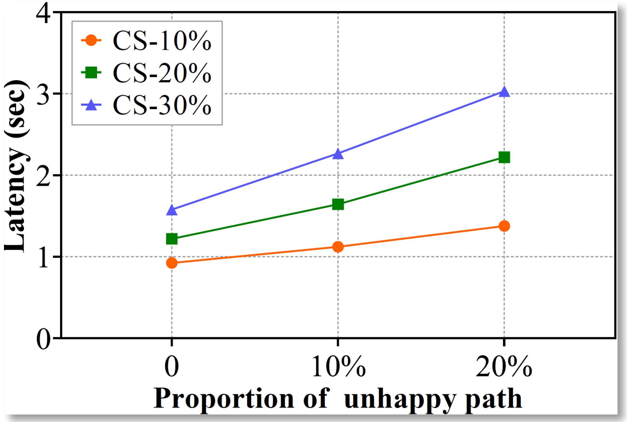
[1] E. Kokoris-Kogias, P. Jovanovic, L. Gasser et al., “Omniledger: A secure, scale-out, decentralized ledger via sharding,” in SP’18. IEEE, 2018.
[2] M. Al-Bassam, A. Sonnino, S. Bano et al., “Chainspace: A sharded smart contracts platform,” in NDSS’18. ISOC, 2018.
[3] J. Hellings and M. Sadoghi, “Byshard: sharding in a byzantine environment,” VLDB J., vol. 32, no. 6.
[4] M. Zamani, M. Movahedi, and M. Raykova, “Rapidchain: Scaling blockchain via full sharding,” in CCS’18. ACM, 2018.
[5] M. J. Amiri, D. Agrawal, and A. El Abbadi, “Sharper: Sharding permissioned blockchains over network clusters,” in SIGMOD’21. ACM,2021.
[6] H. Dang, T. T. A. Dinh, D. Loghin et al., “Towards scaling blockchain systems via sharding,” in SIGMOD’19. ACM, 2019.
[7] Z. Hong, S. Guo, P. Li, and W. Chen, “Pyramid: A layered sharding blockchain system,” in INFOCOM’21. IEEE, 2021.
[8] J. Wang and H. Wang, “Monoxide: Scale out blockchains with asynchronous consensus zones,” in NSDI’19, vol. 2019, 2019.

Evaluation

Invalid Transaction Processing:

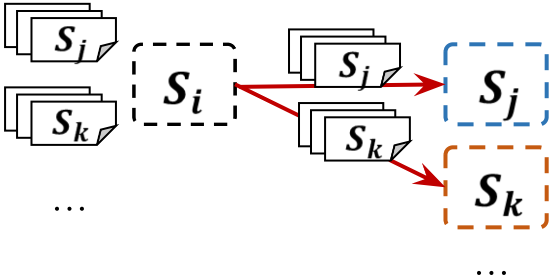
- Cross-shard request proportions: 10%, 20%, and 30%.
- Invalid transaction proportion: 20%





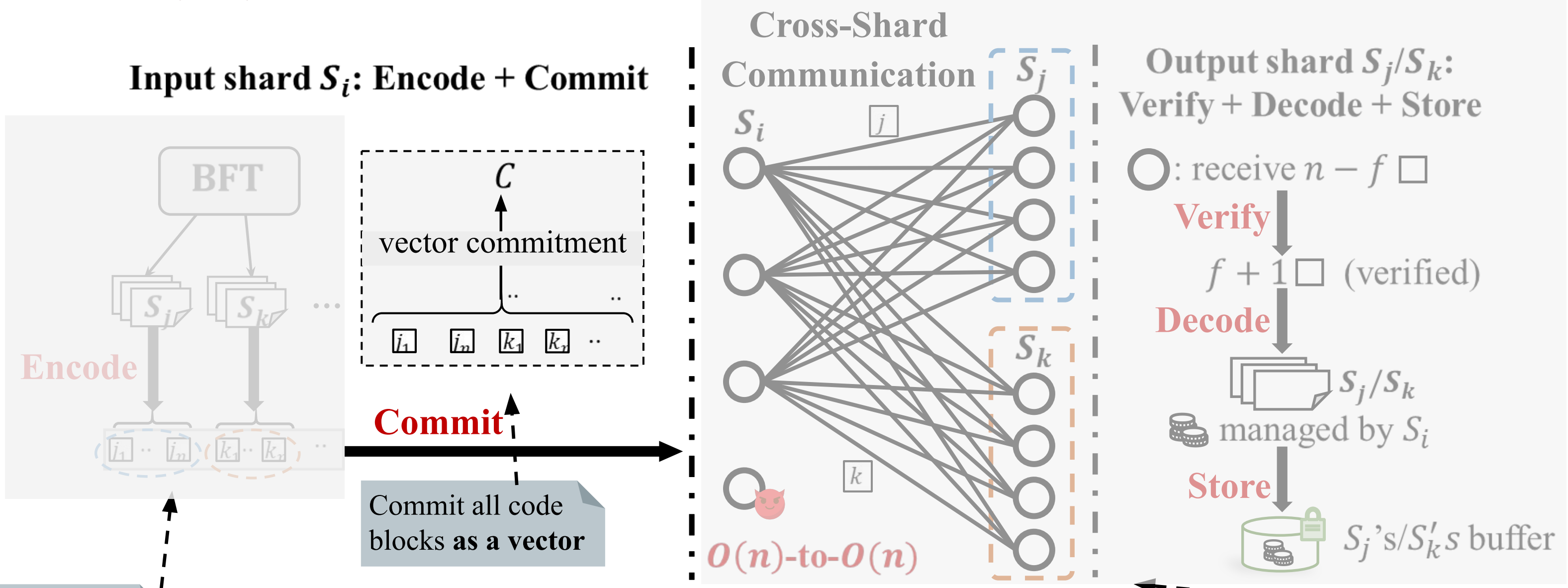
- **Kronos rejects invalid requests with low impact on performance.**

Reliable Cross-Shard Batch Certification (RCBC)



Method 2: Vector-commitment-based RCBC (VC-RCBC) :

- Vector commitment + Erasure coding
- $CS\text{-}\omega = \boldsymbol{O(nb\lambda)}$



Commit all code blocks as a **vector**

the same as HT-RCBC

e.g., $cert[k_\alpha] = \langle C, \Lambda_\alpha^k \rangle$

C: succinct-length commitment, Λ_i^3: succinct-length proof of k_α

the same as HT-RCBC