Scaling Blockchains

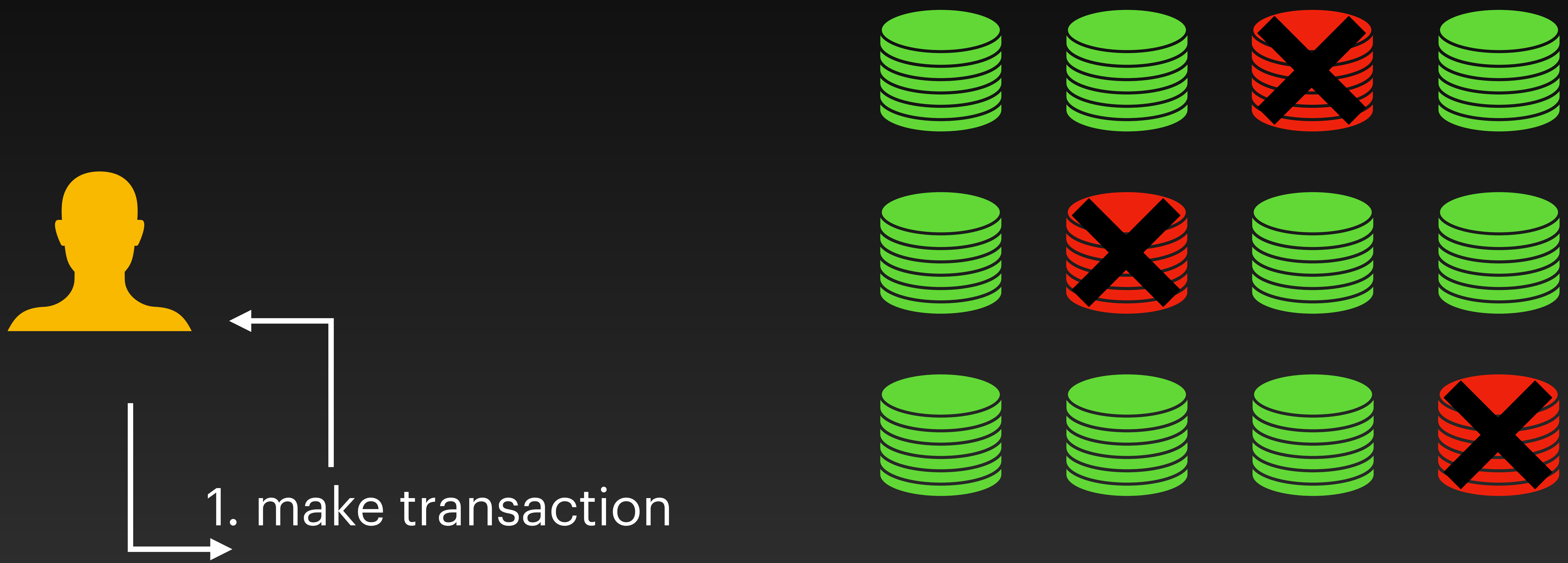
Achieving Arbitrary Throughput and Sub-Second Latency

Introduction

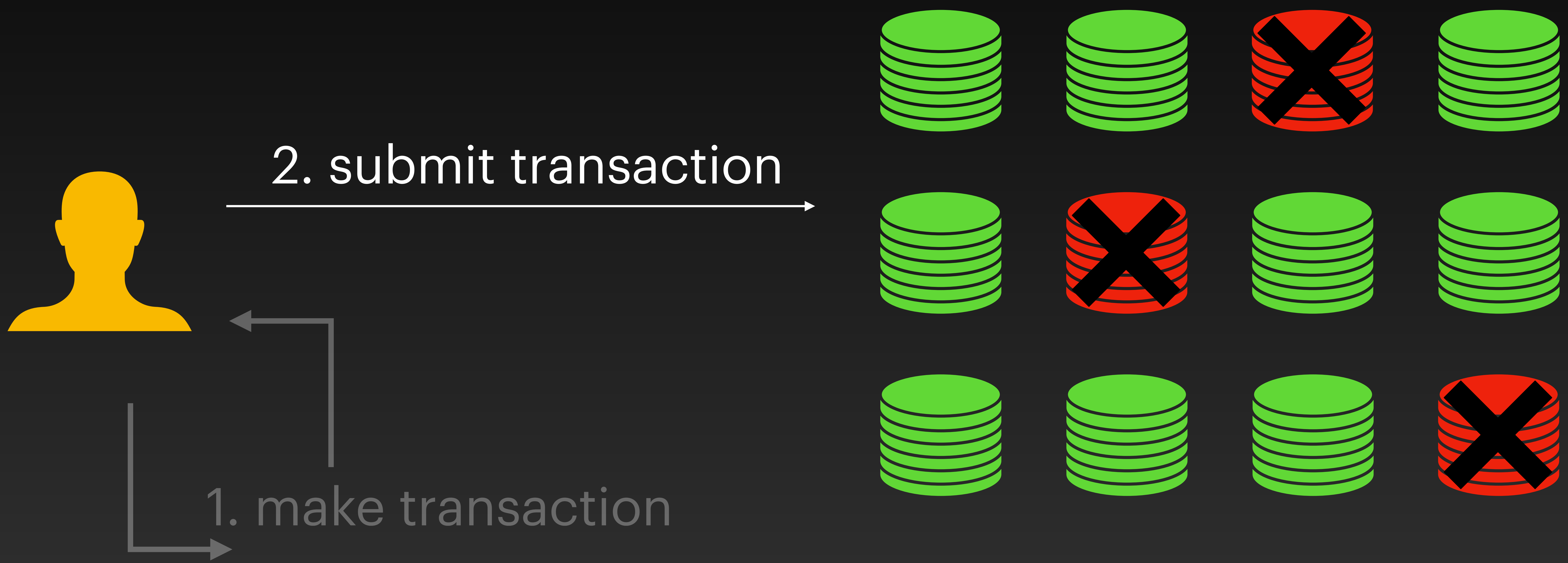
A set of nodes

Byzantine Fault Tolerance

Blockchains

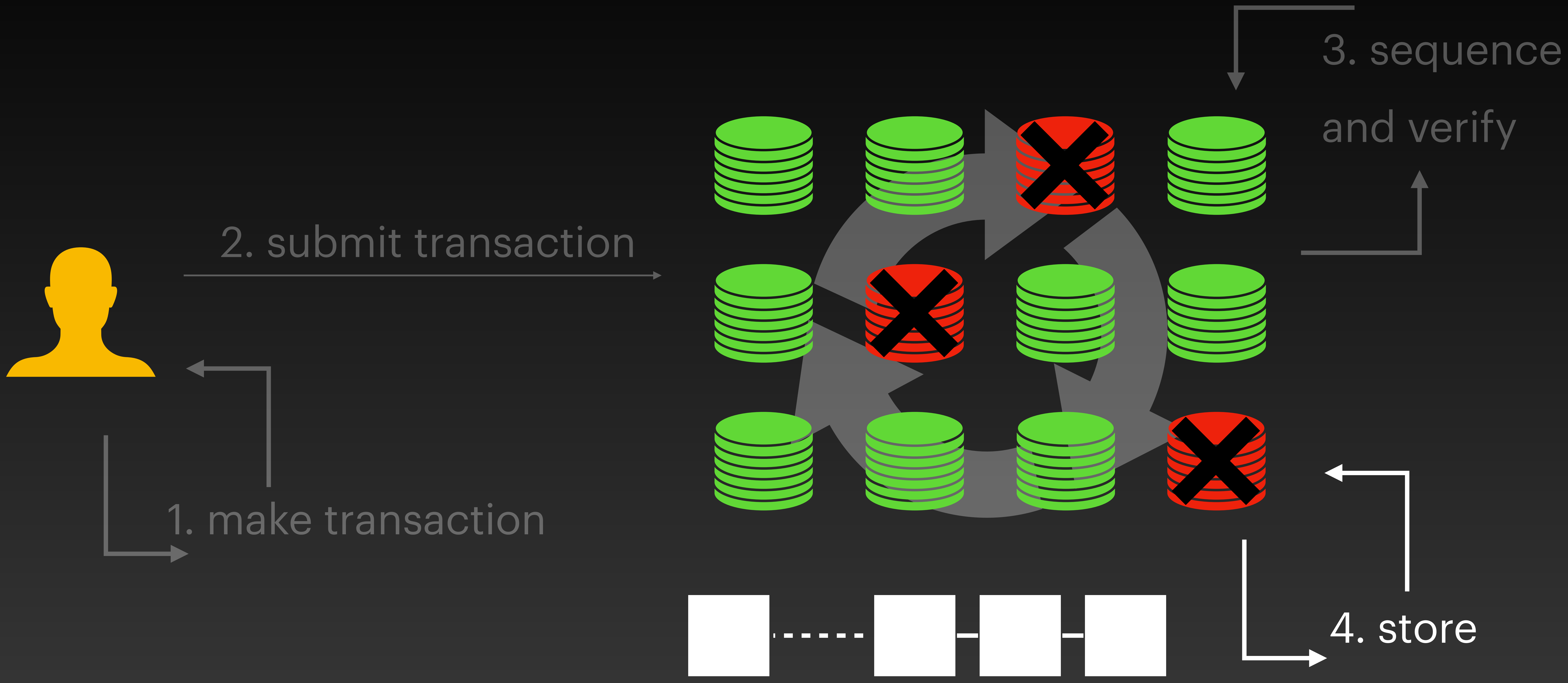


Blockchains

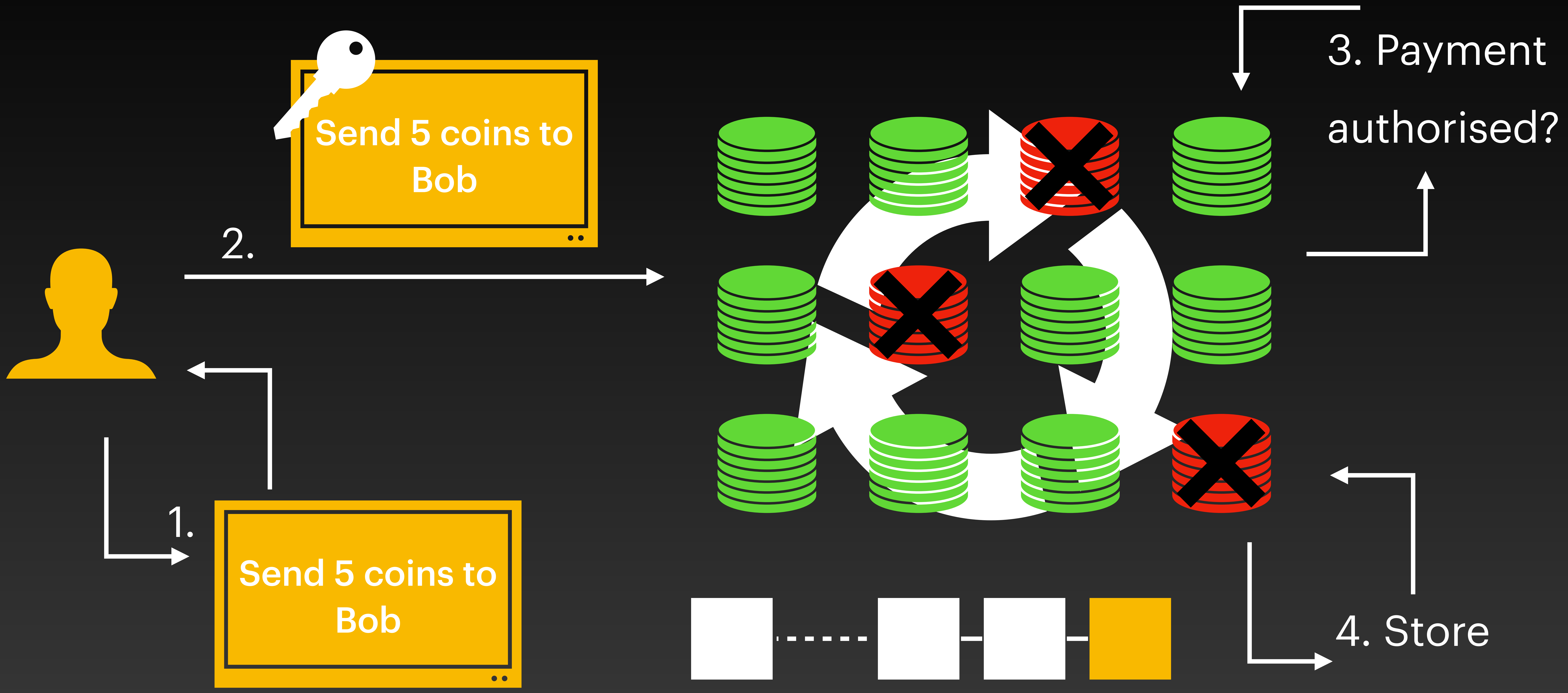


Blockchains

Blockchains



The best example

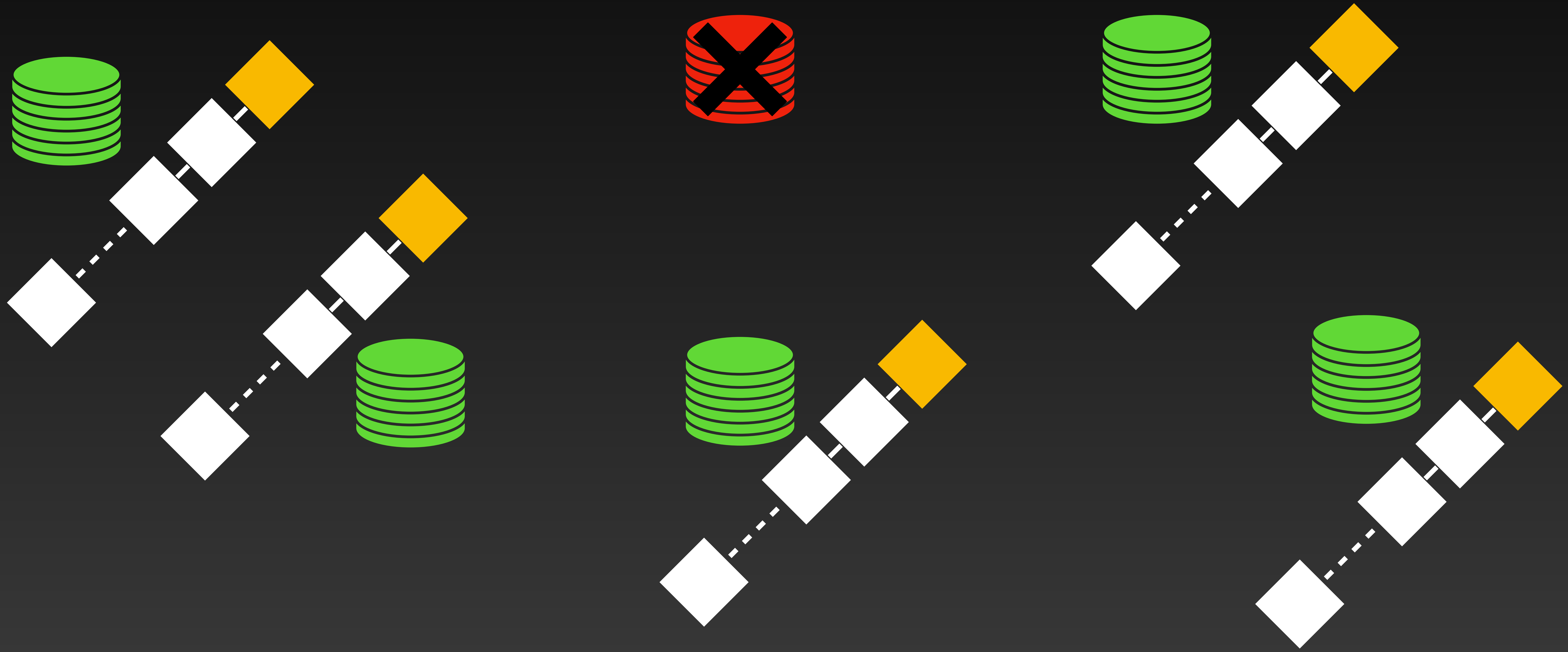


High latency

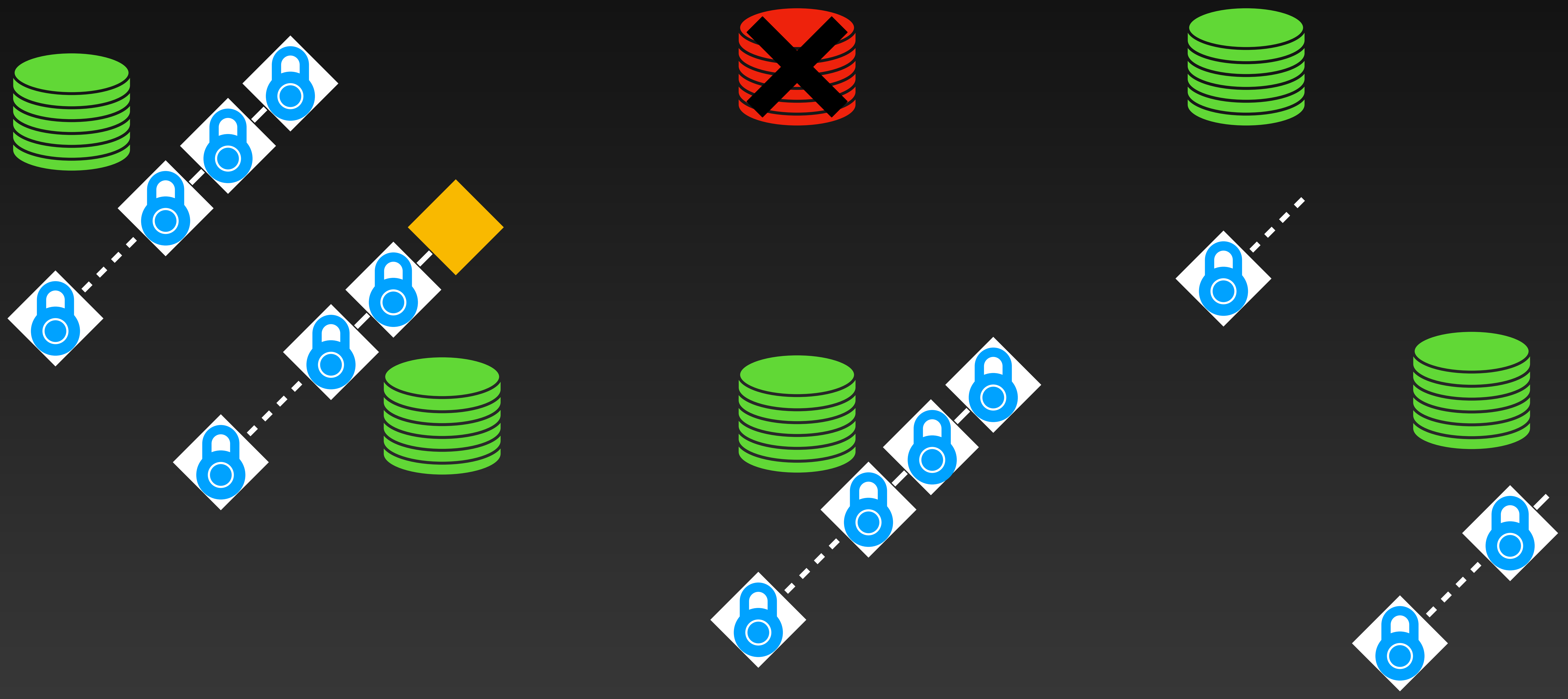
Slow finality

Low throughput

Hight Latency



Slow Finality



Low Throughput

Outline

Part I: Increase throughput

through sharding

Part II: Reduce latency

with side infrastructures

Outline

Part I: Increase throughput

Key concepts of state sharding

Cross-shard consensus

Replay attacks against common systems

Secure Cross-shard consensus

Evaluation

Part II: Reduce latency

Outline

Part I: Increase throughput

Part II: Reduce latency

Blockchains for retails payments

FastPay as side infrastructure

Interfacing FastPay with a primary system

Implementation

Evaluation

Main Takeaways

Part I: Increase throughput

Key concepts of sharded distributed ledgers

Main challenges in building secure sharded systems in practice

Part II: Reduce latency

Side-infrastructures to bring blockchain-based payment systems to physical points of sales

How to integrate those infrastructures into a primary distributed ledger

What is not covered

Privacy on blockchain

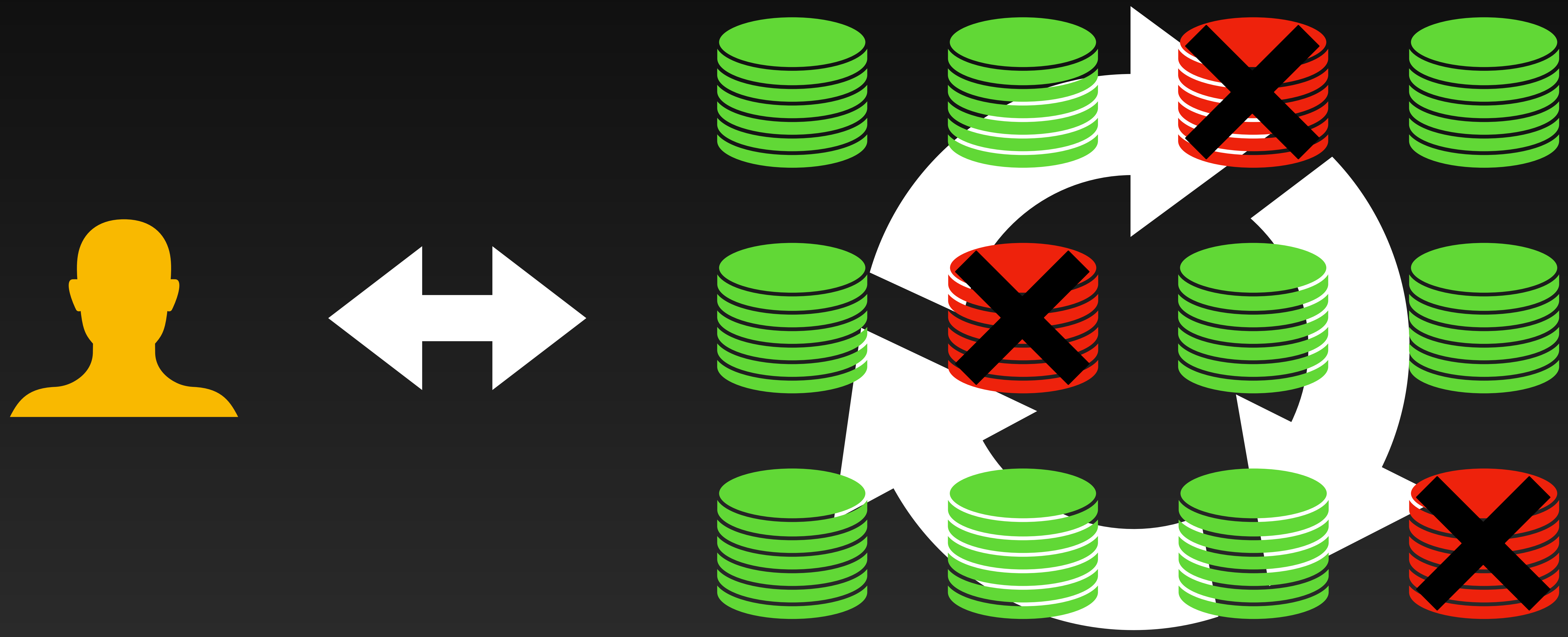
Sybil resistance mechanisms

Incentives of nodes operators

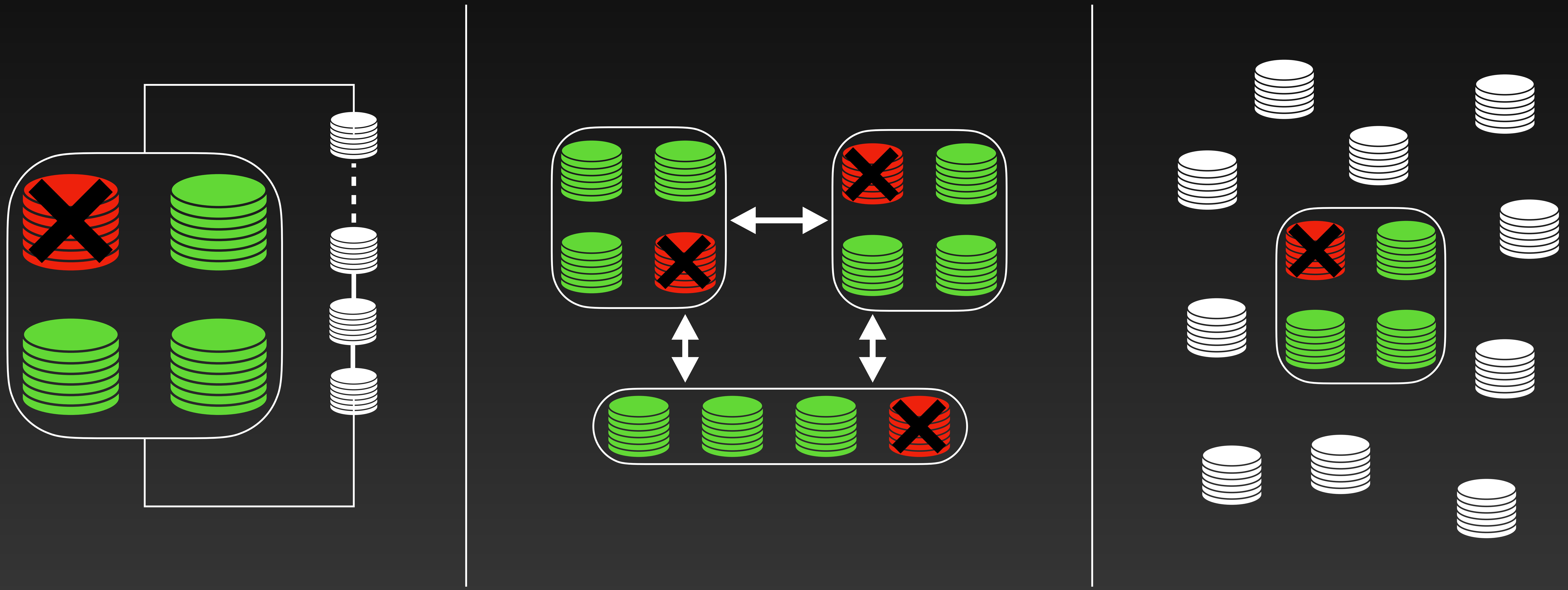
Increasing Throughput

with Sharded Blockchains

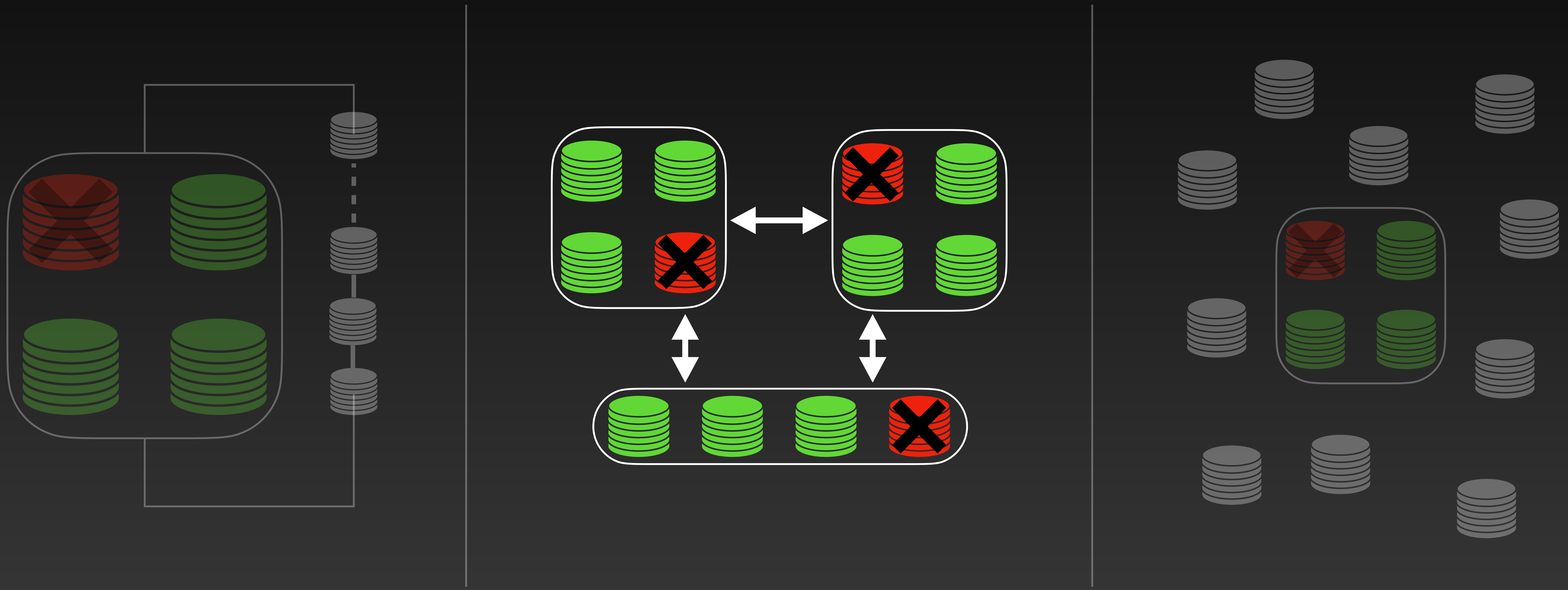
Part I



Scaling blockchains



Scaling blockchains



Hight throughput

BFT resilience

Fast finality

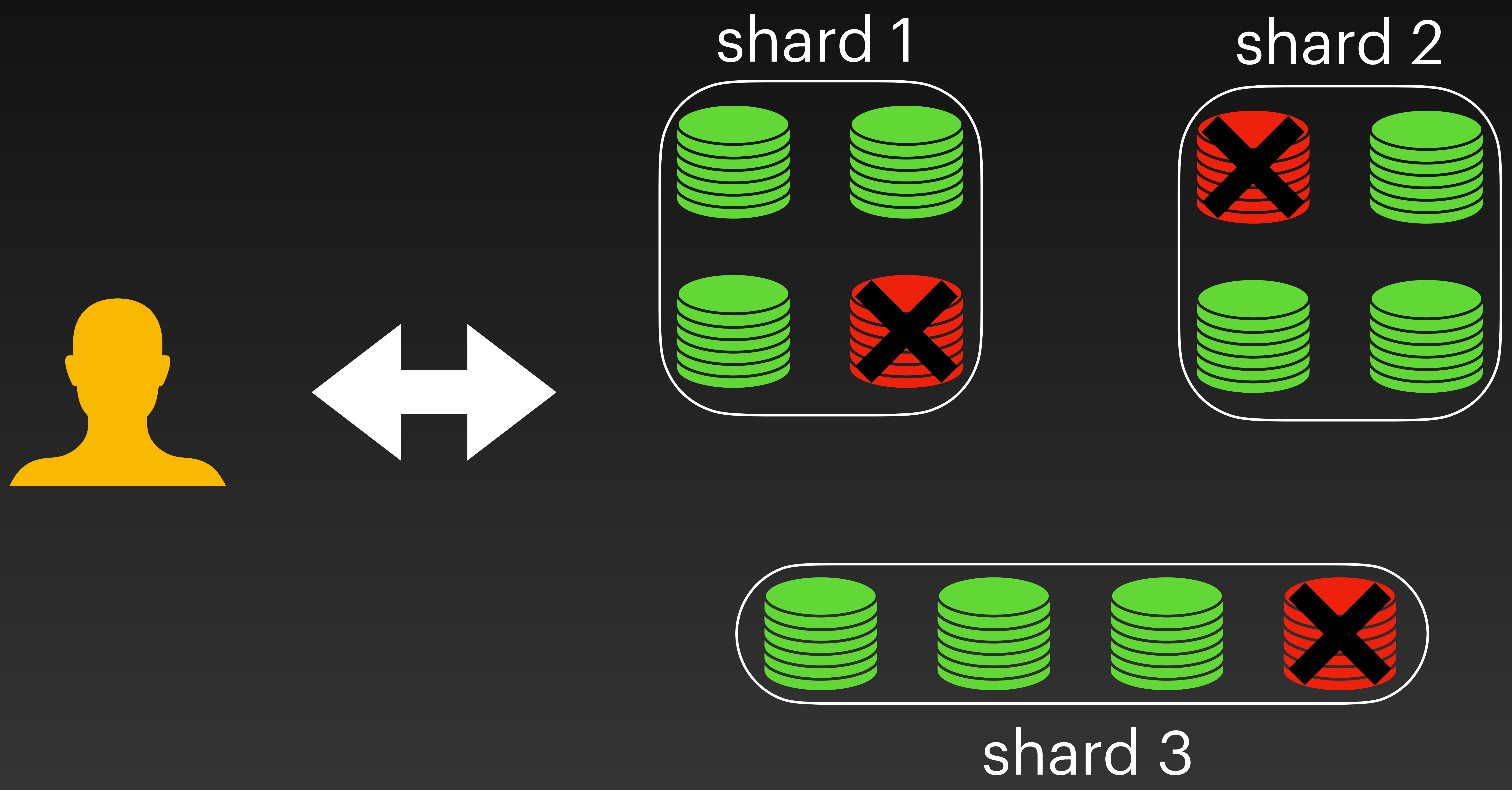
Linear scalability

The more machines you have, the bigger your throughput

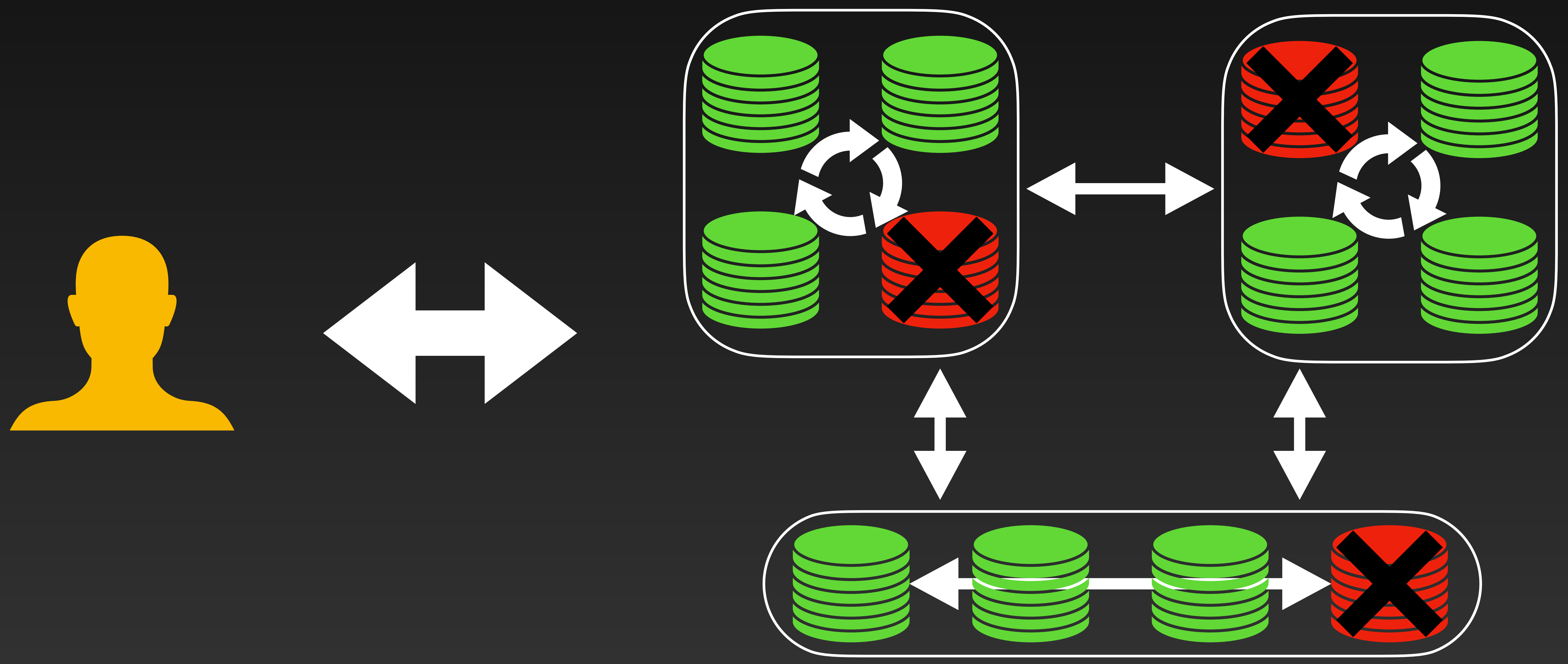
Scalability

State Sharding

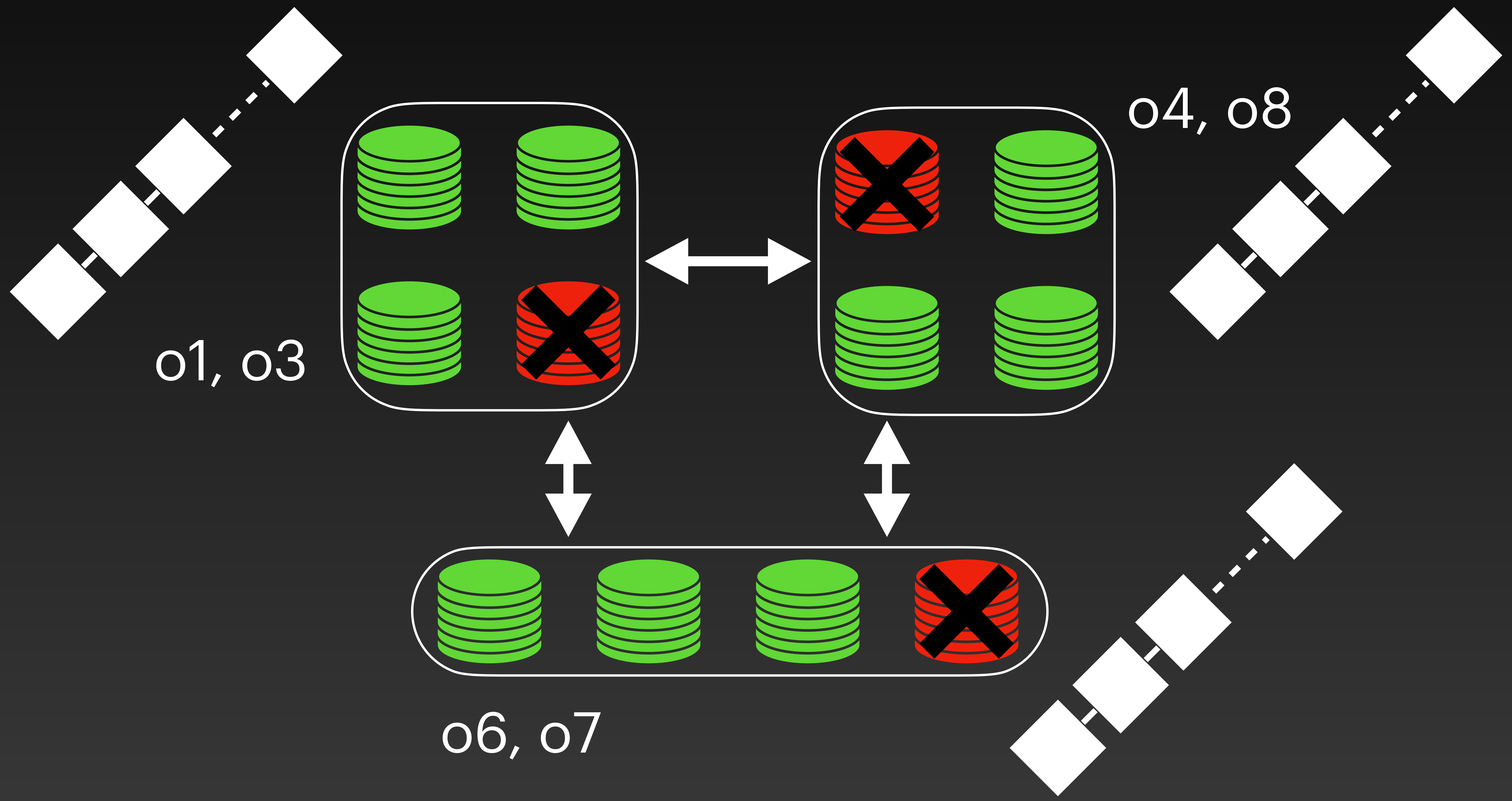
State Sharding



State Sharding

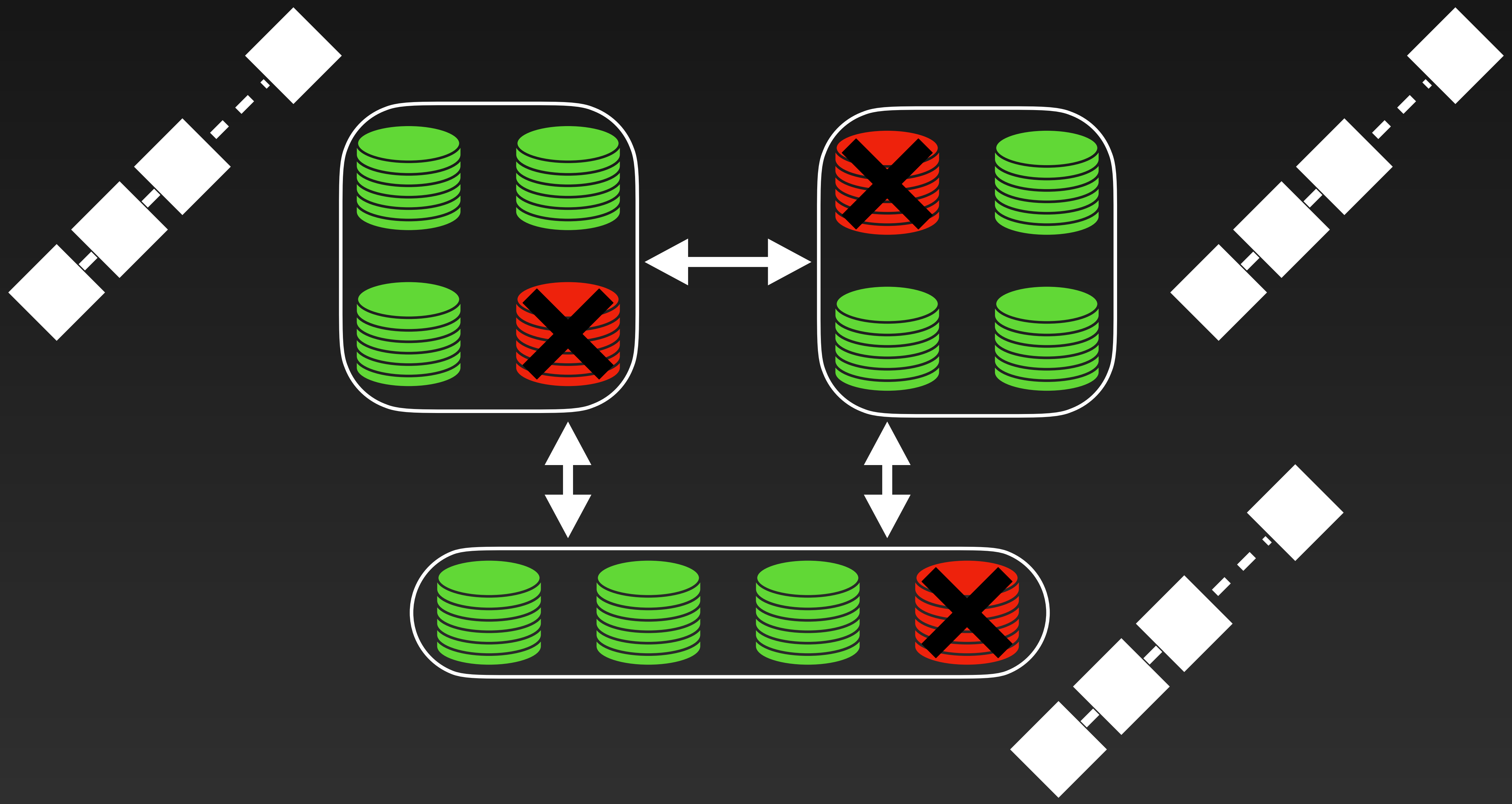


State Sharding



Traditional

Sharding



State Sharding

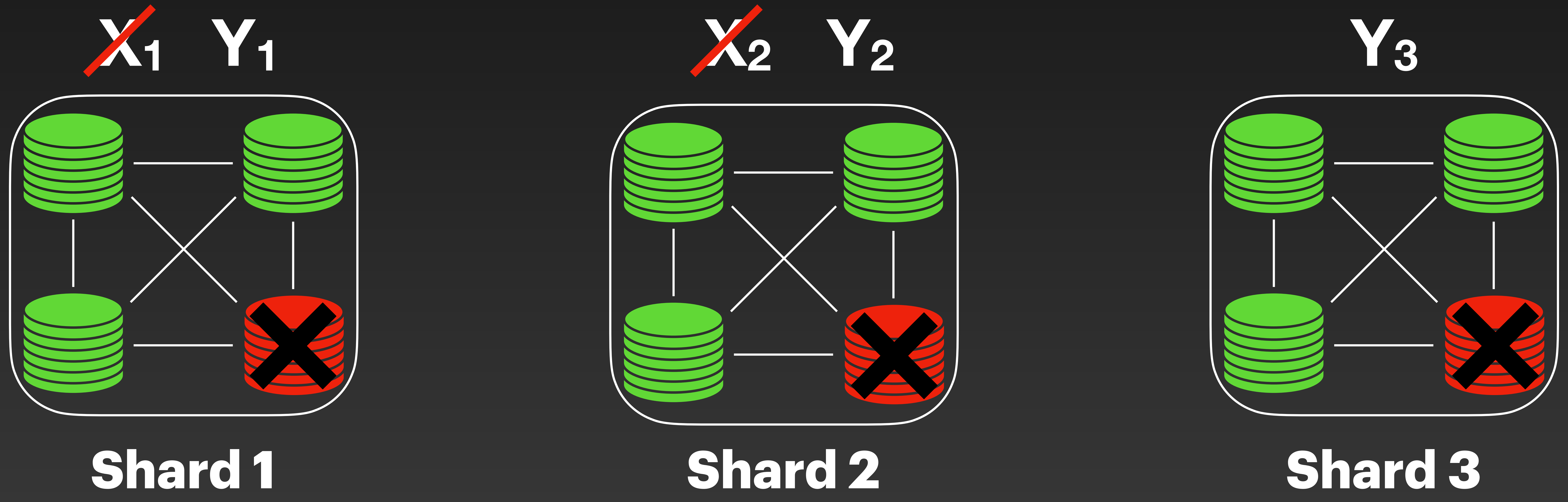
An example transaction

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$

State Sharding

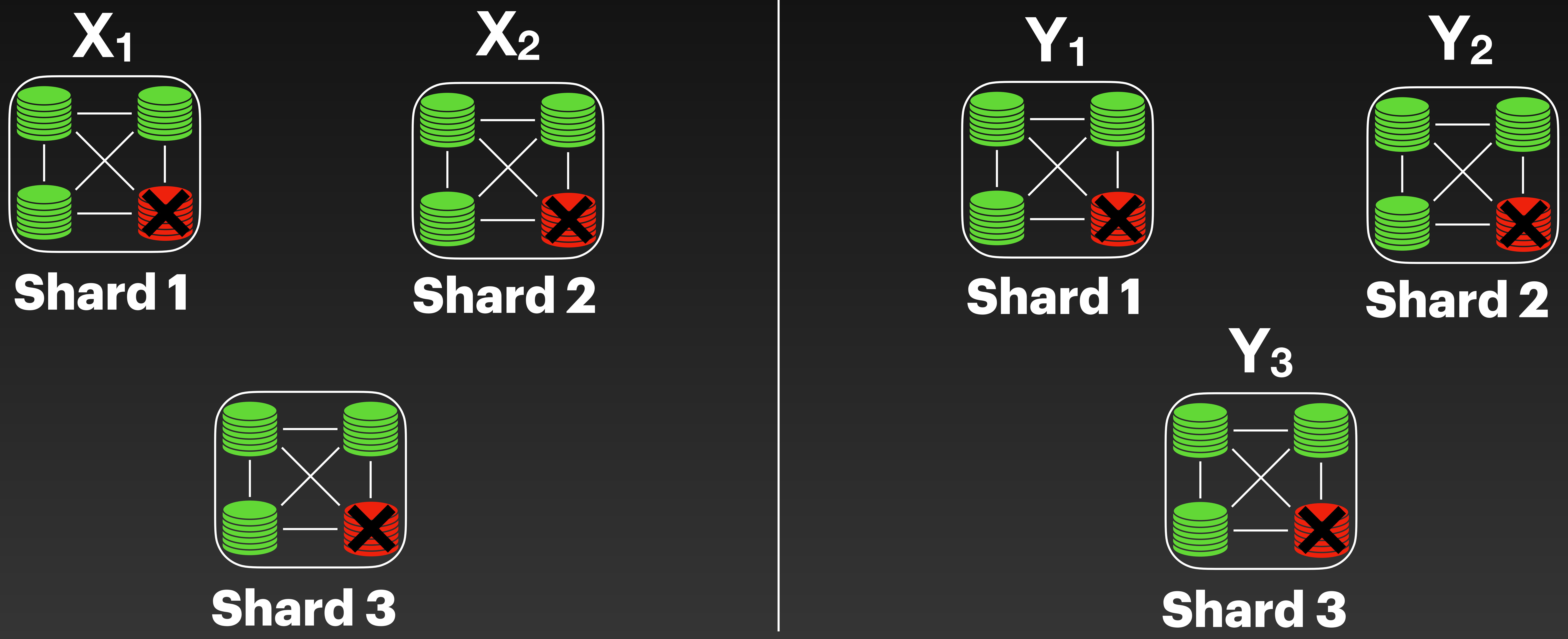
An example transaction

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



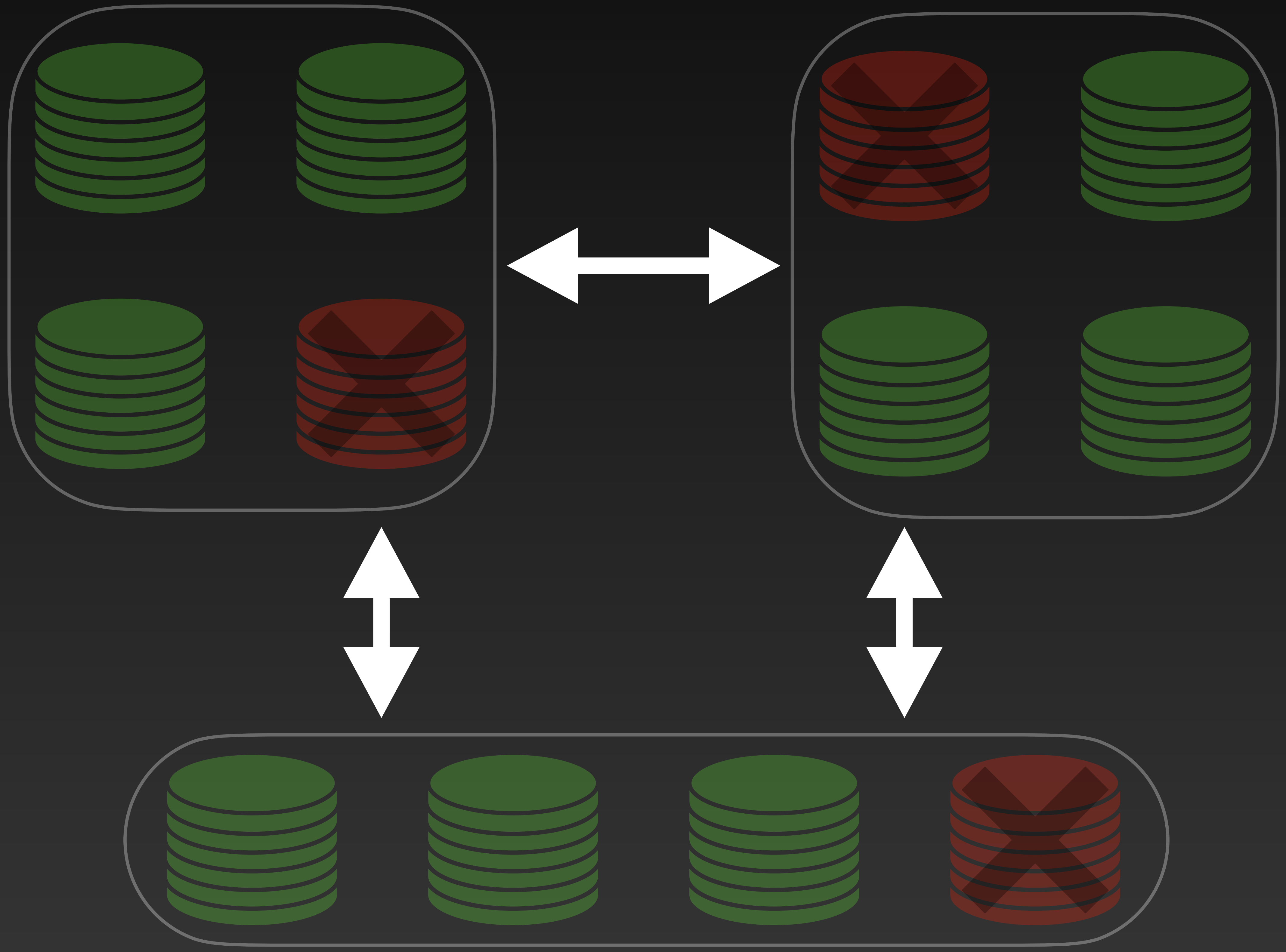
State Sharding

Only two acceptable final states



Cross-Shard Consensus

How do shards communicate with each other?



S-BAC

Chainspace: A Sharded Smart Contracts Platform

Mustafa Al-Bassam*, Alberto Sonnino*, Shehar Bano*, Dave Hrycyszyn† and George Danezis*
* University College London, United Kingdom
† constructiveproof.com

NDSS'18

Atomix

OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding

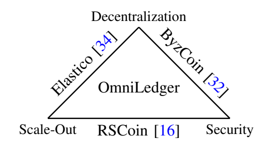


Fig. 1: Trade-offs in current DL systems.

S&P'18

Cross-Shard Consensus

Byzantine Agreement

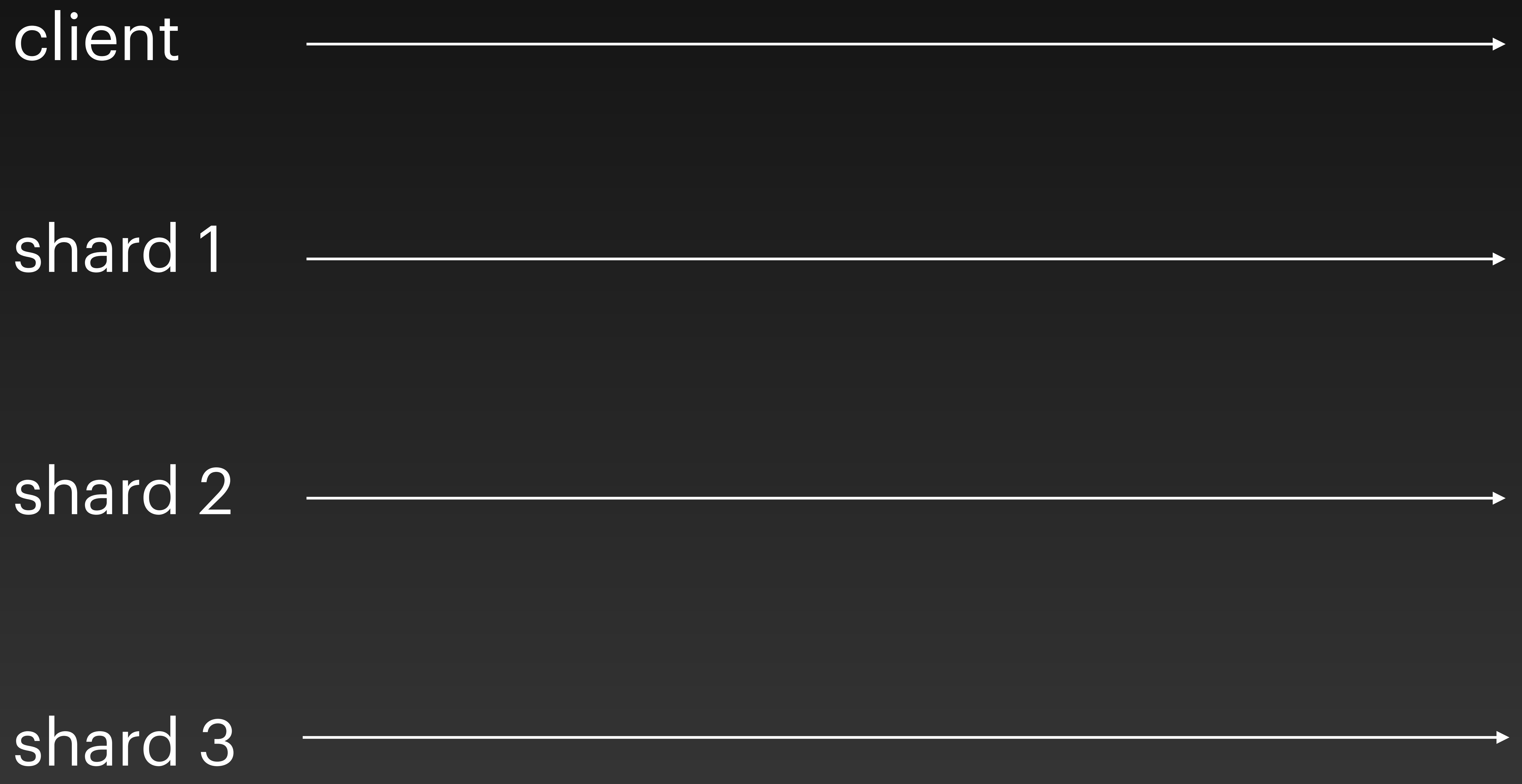
+

2-Phases Atomic Commit

Spoiler alert: Insecure under parallel composition

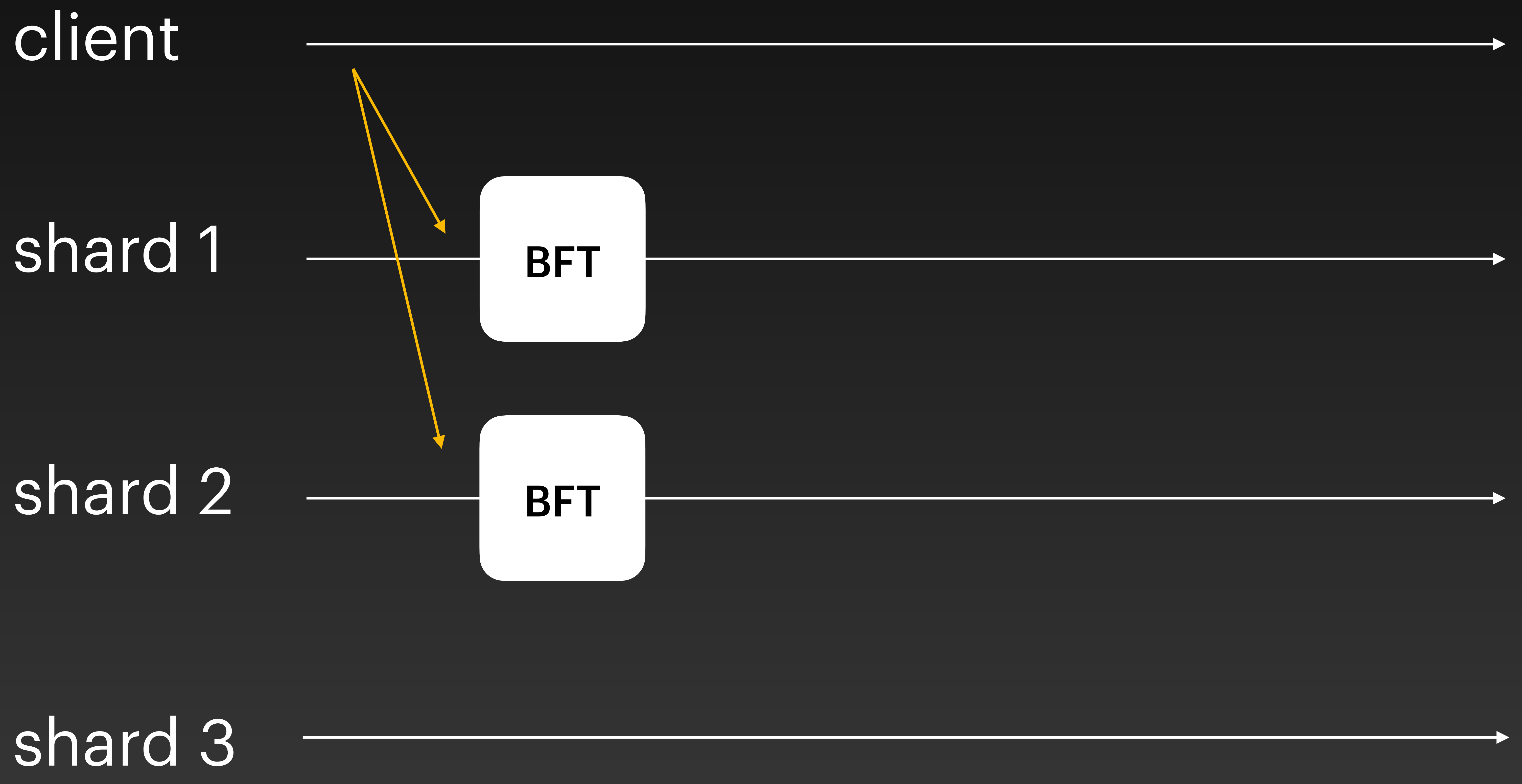
S-BAC

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



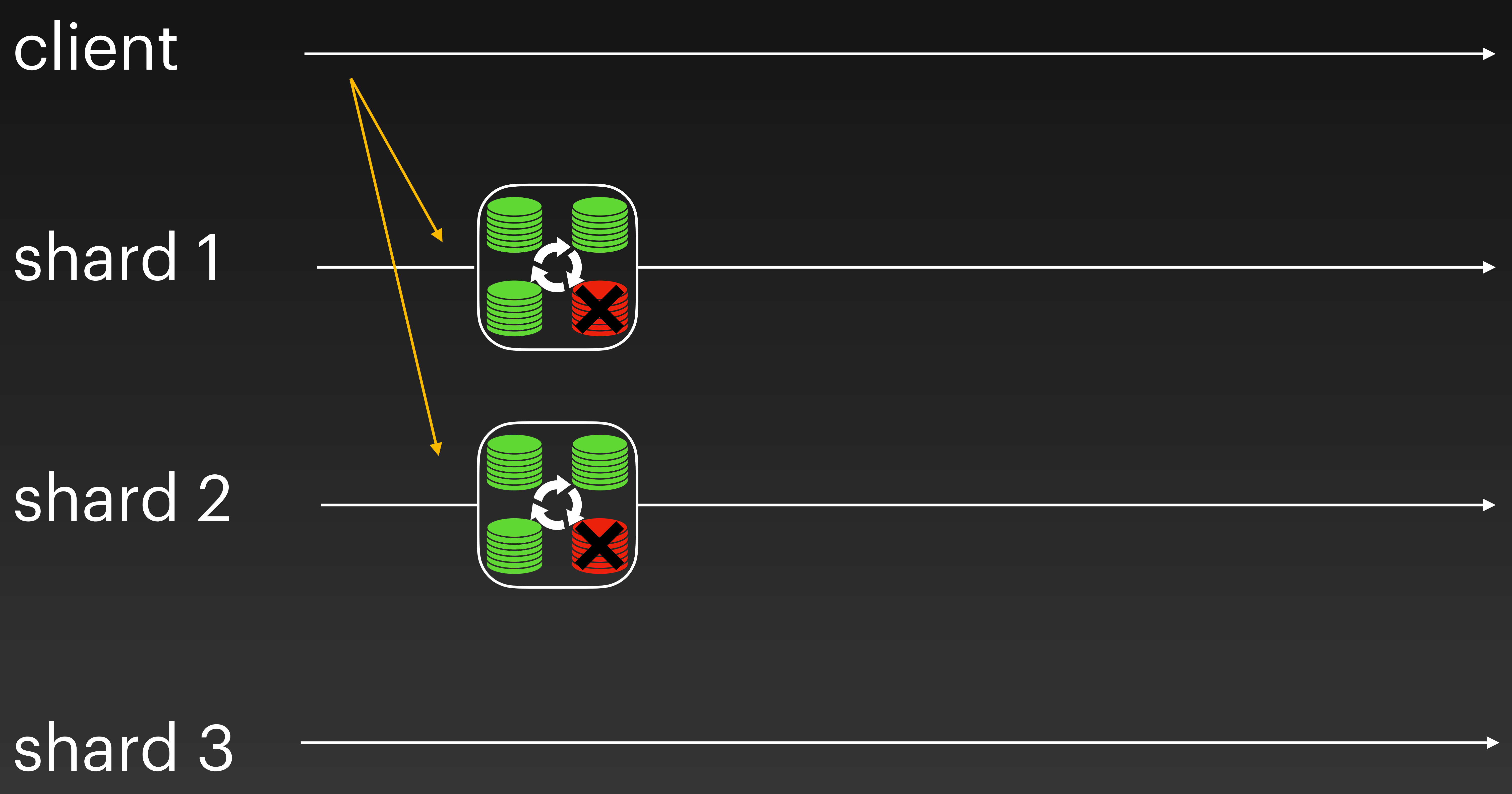
S-BAC

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



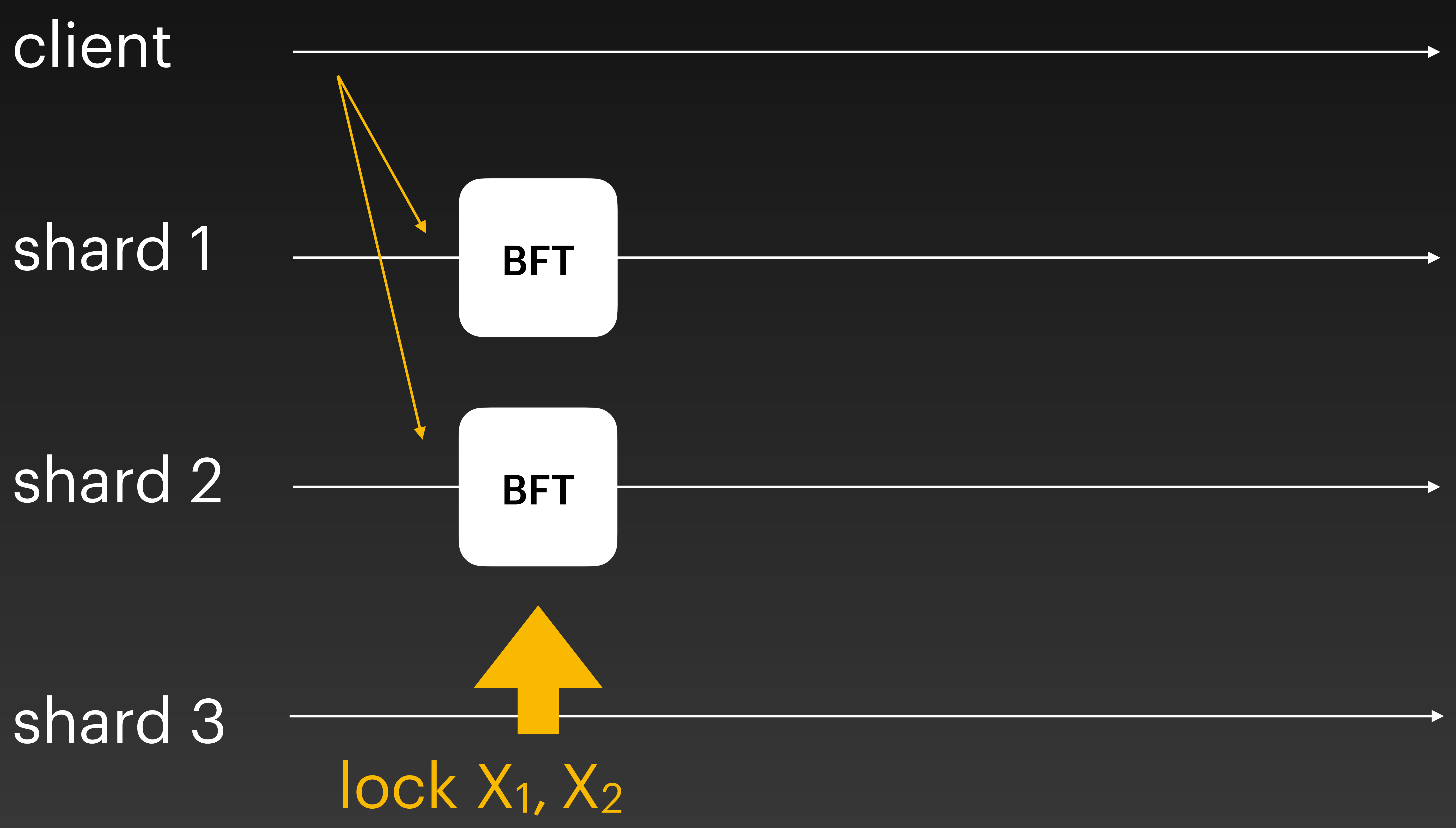
S-BAC

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



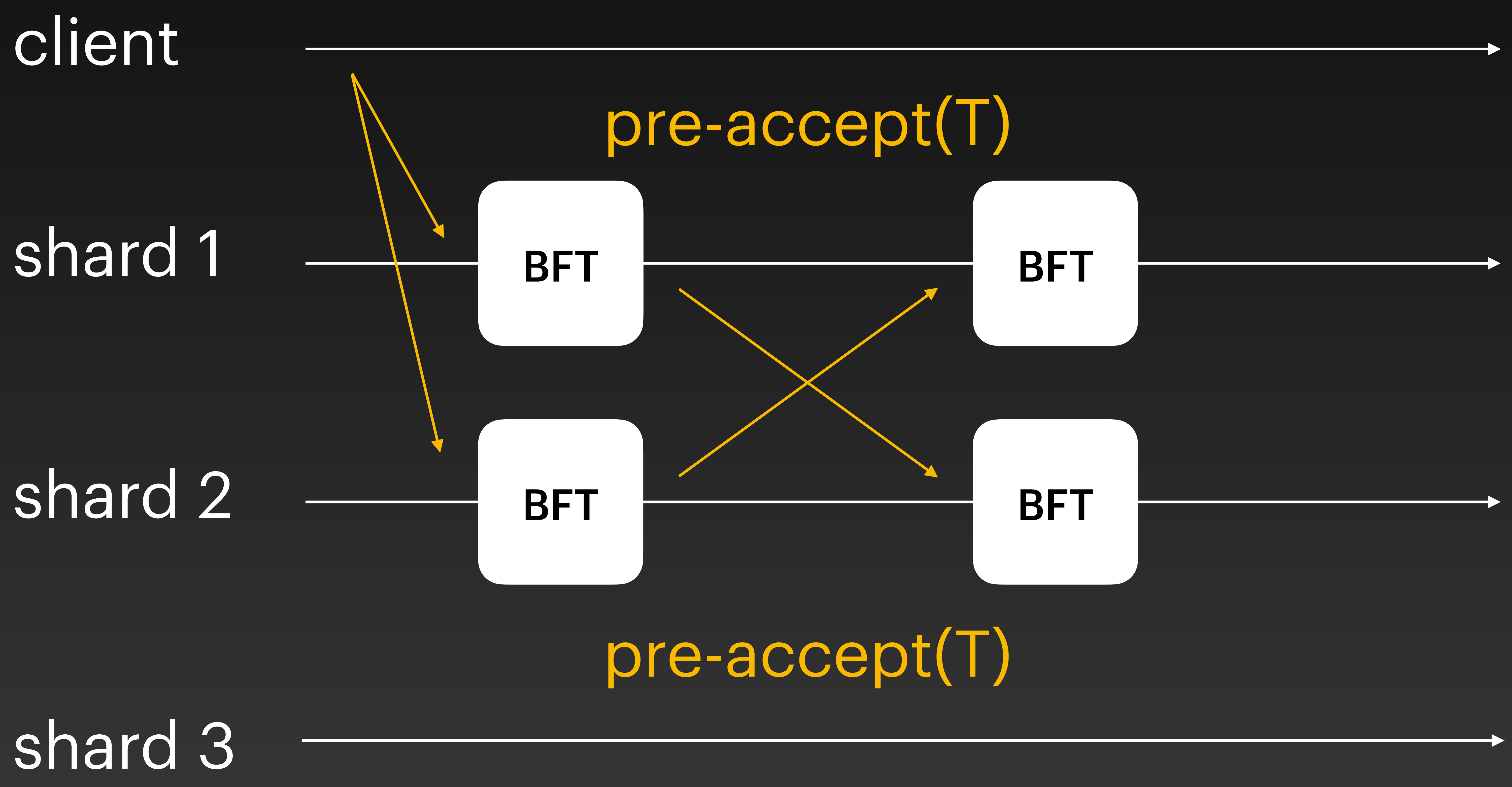
S-BAC

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



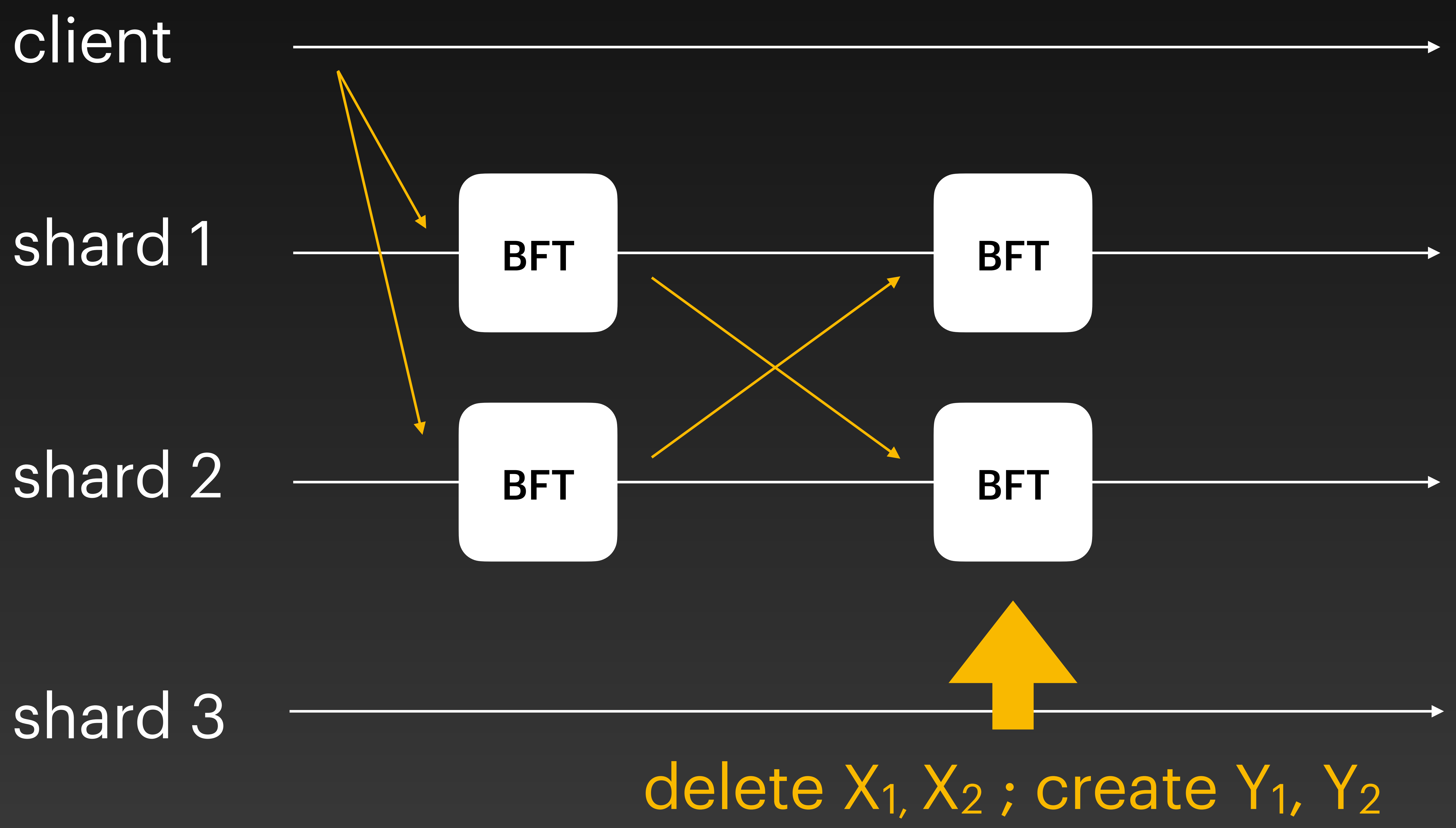
S-BAC

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



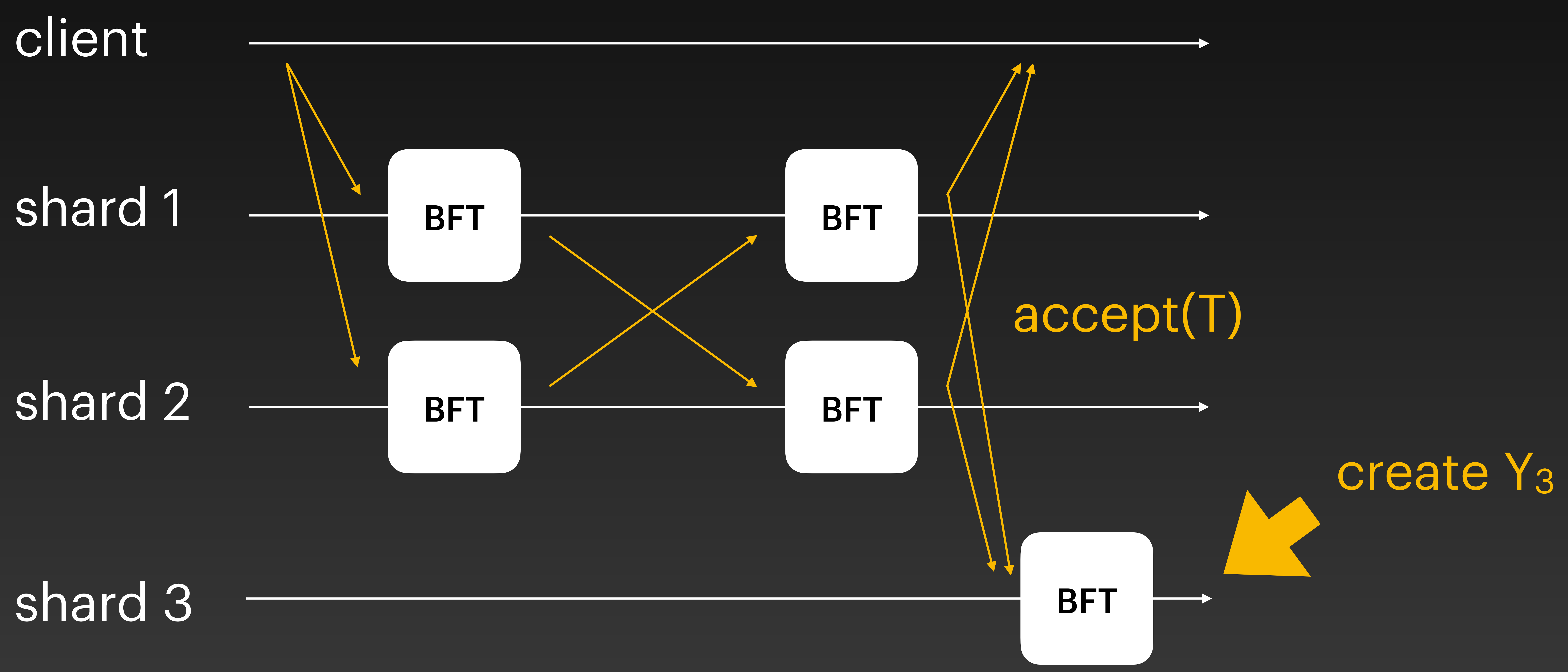
S-BAC

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



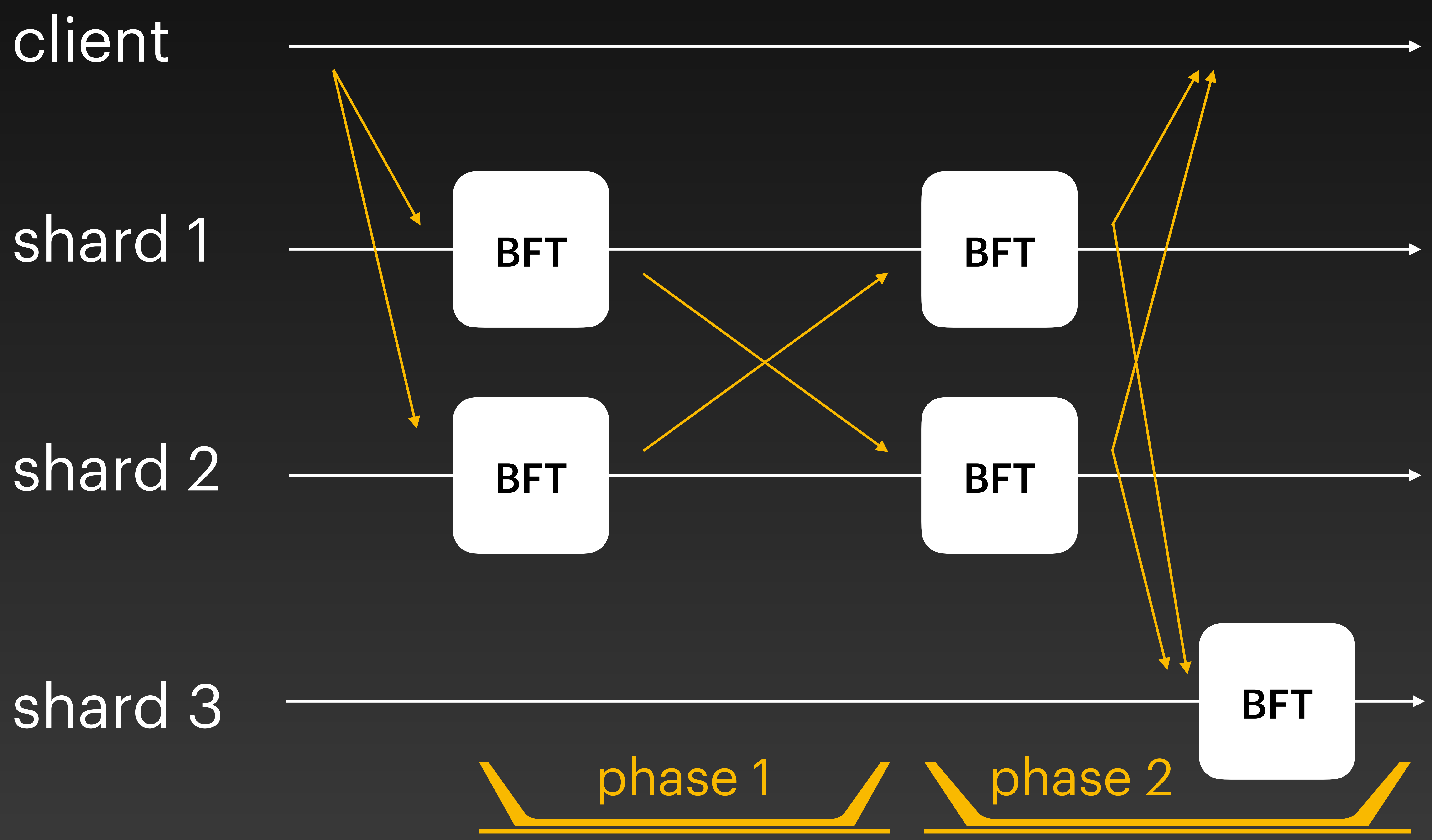
S-BAC

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



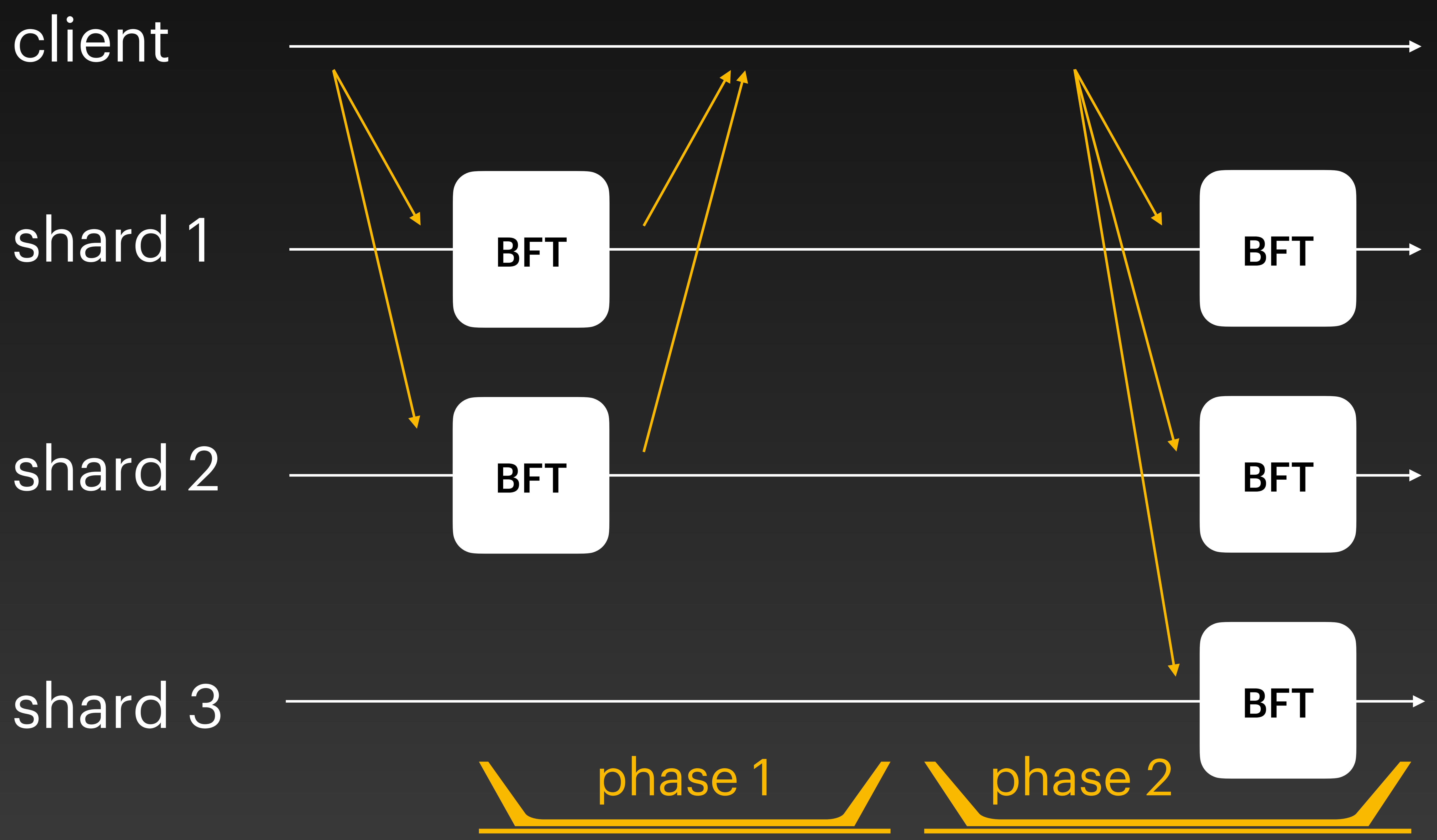
S-BAC

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



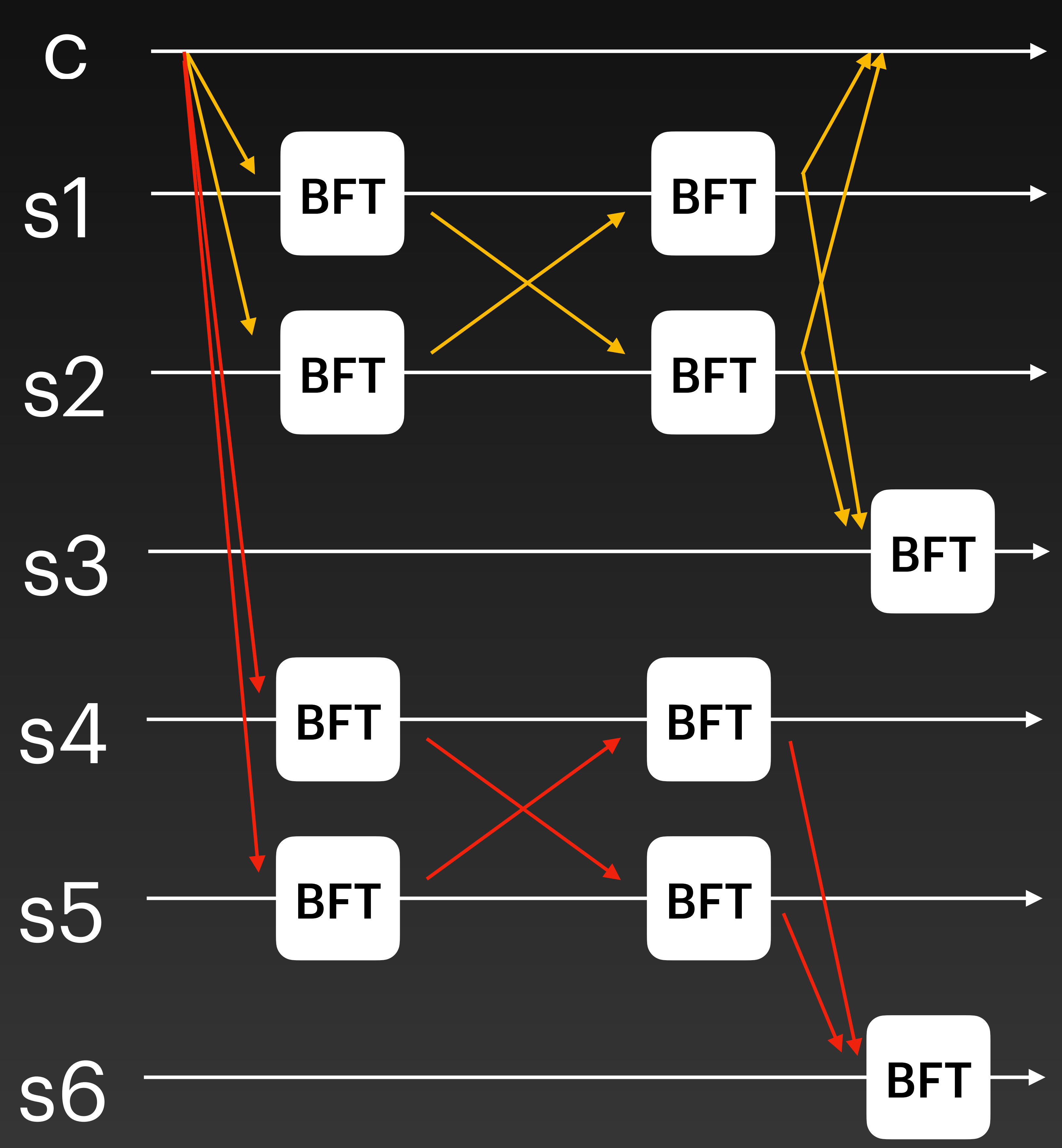
Atomix

$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



Cross-Shard Consensus

How does it achieve linear scalability?



Insecure under parallel composition

Attacks

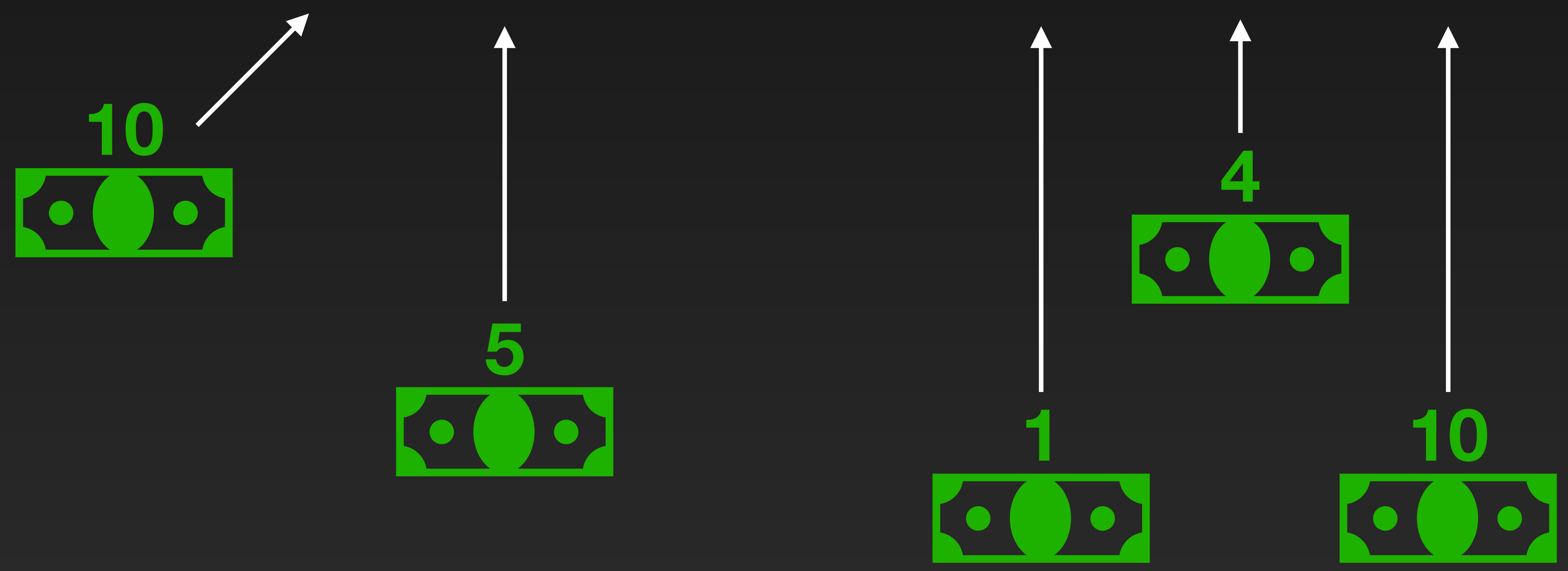
Double spend any object

- Does not need to collude with any node
- Acts as client or passive observer
- Re-orders network messages (not always needed)

Attack against S-BAC

Double-spend X_1

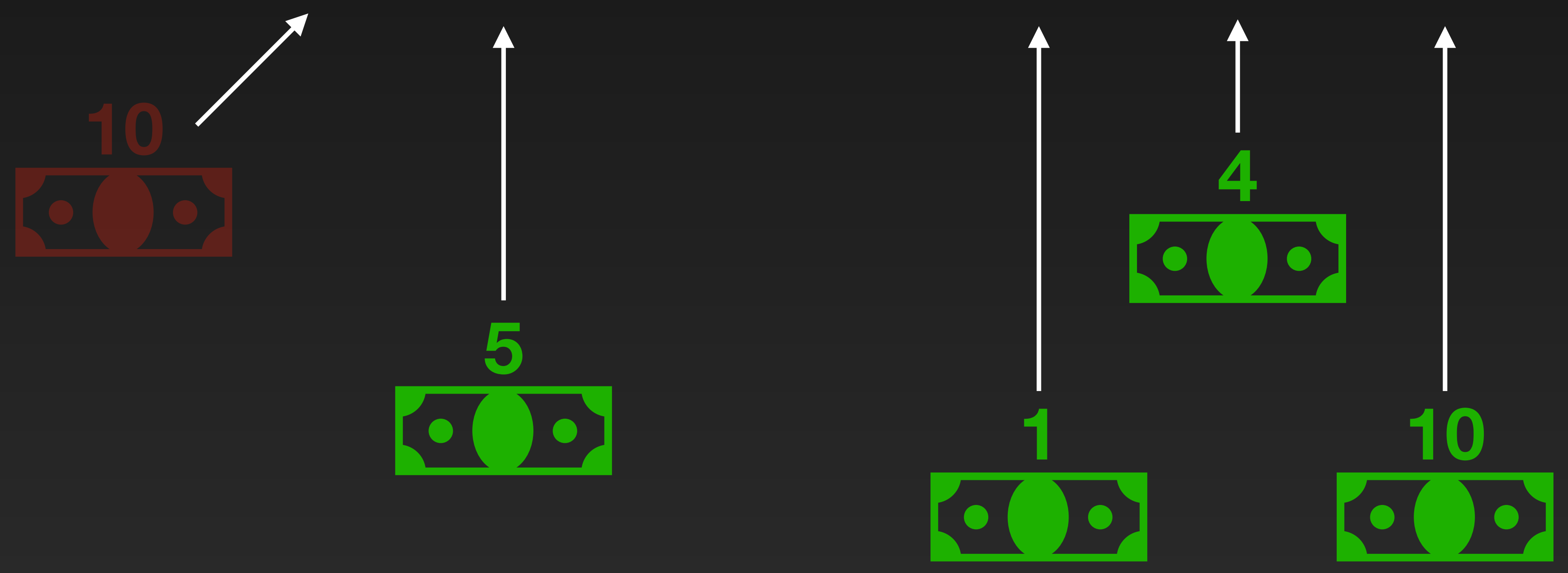
$$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$$



Attack against S-BAC

Double-spend X_1

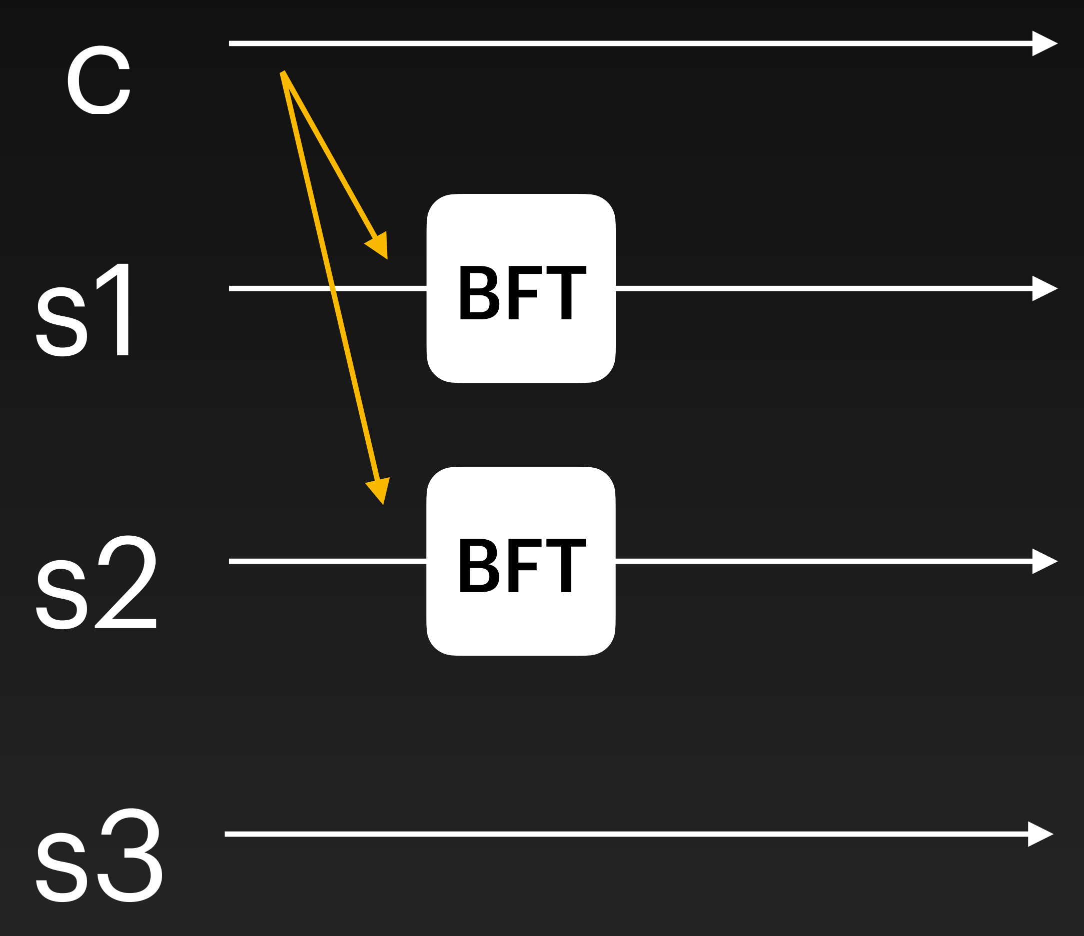
$$T'(\widetilde{x_1}, x_2) \rightarrow (y_1, y_2, y_3)$$



Attack against S-BAC

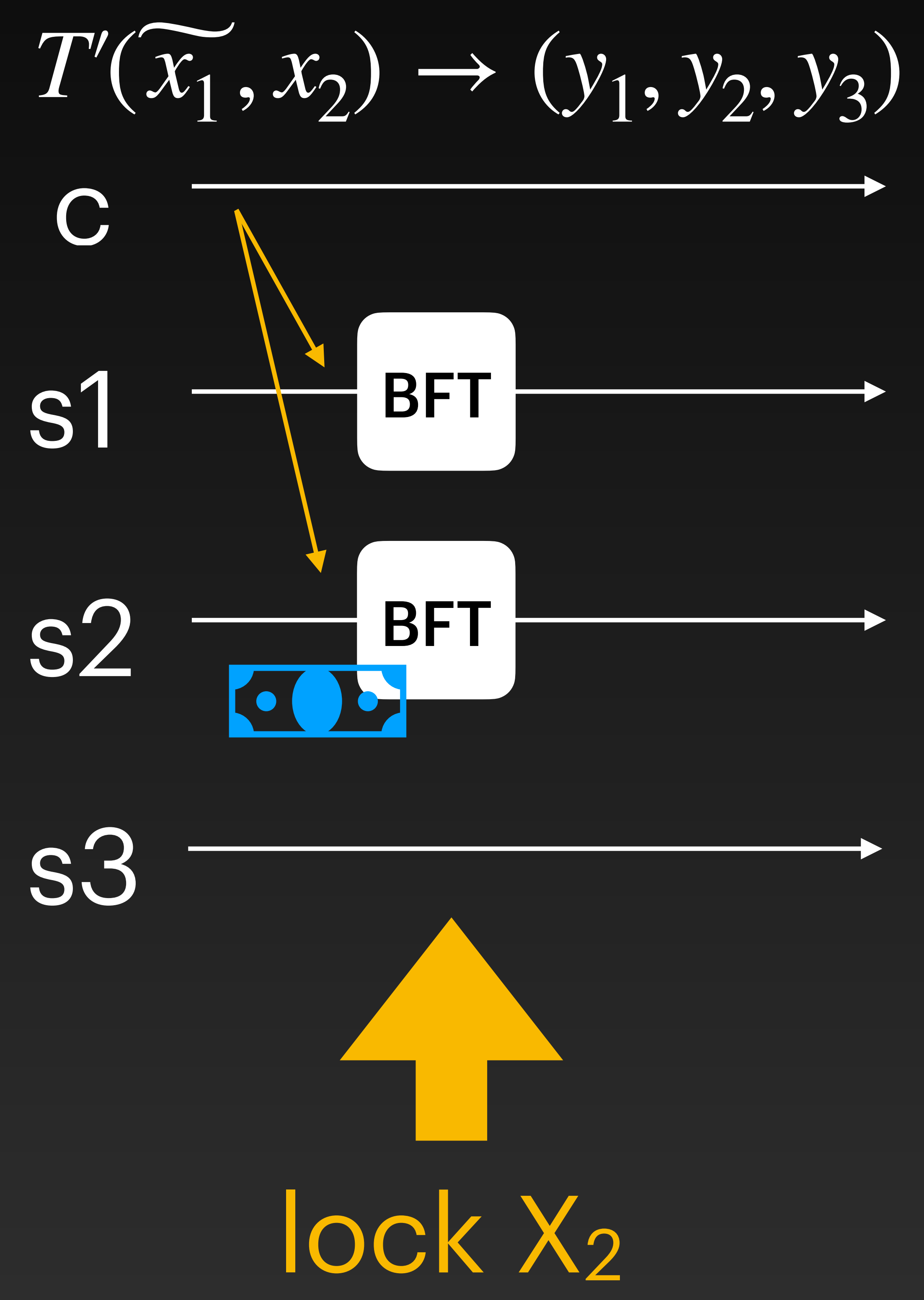
Double-spend X_1

$T'(\widetilde{x_1}, x_2) \rightarrow (y_1, y_2, y_3)$



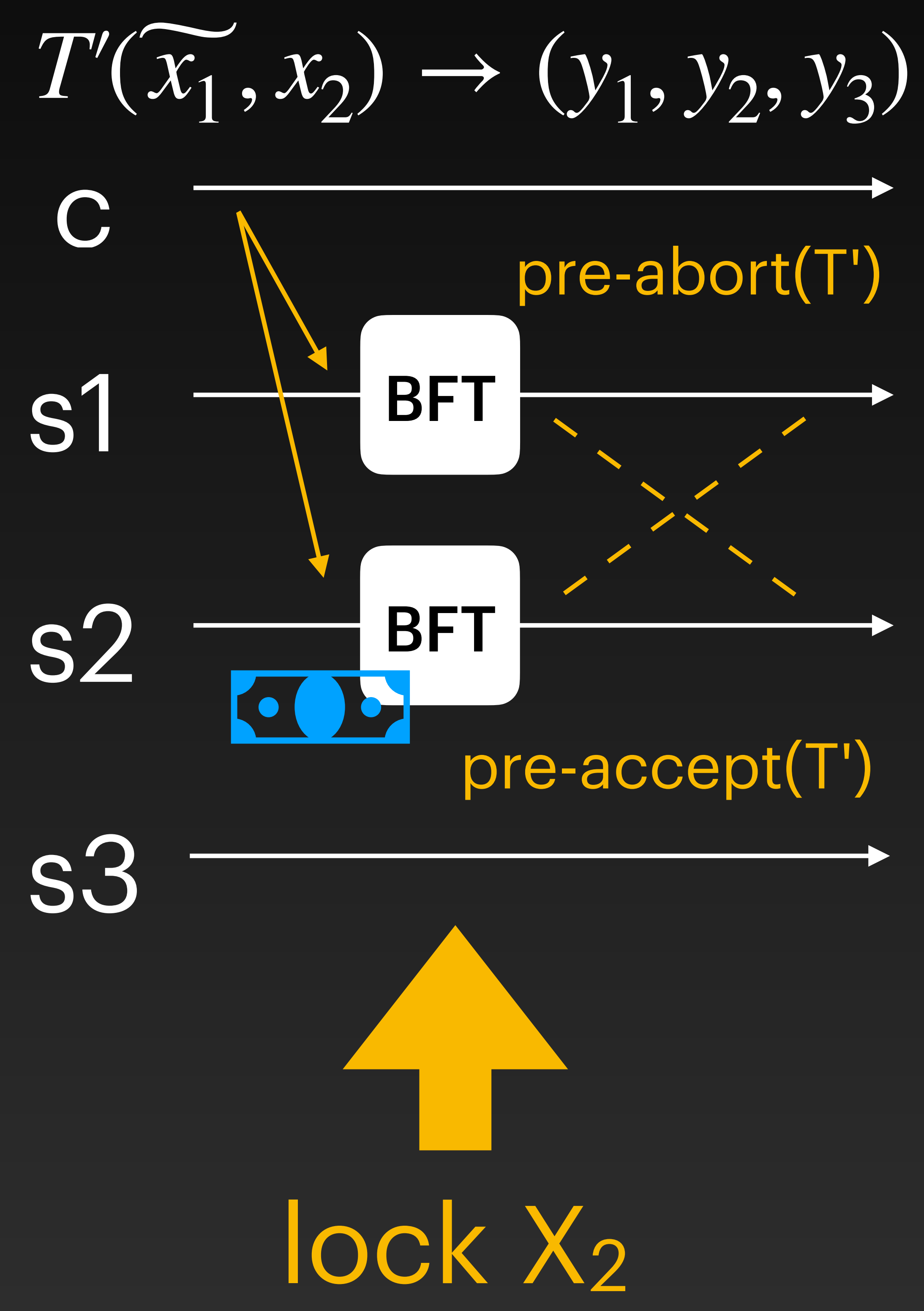
Attack against S-BAC

Double-spend X_1



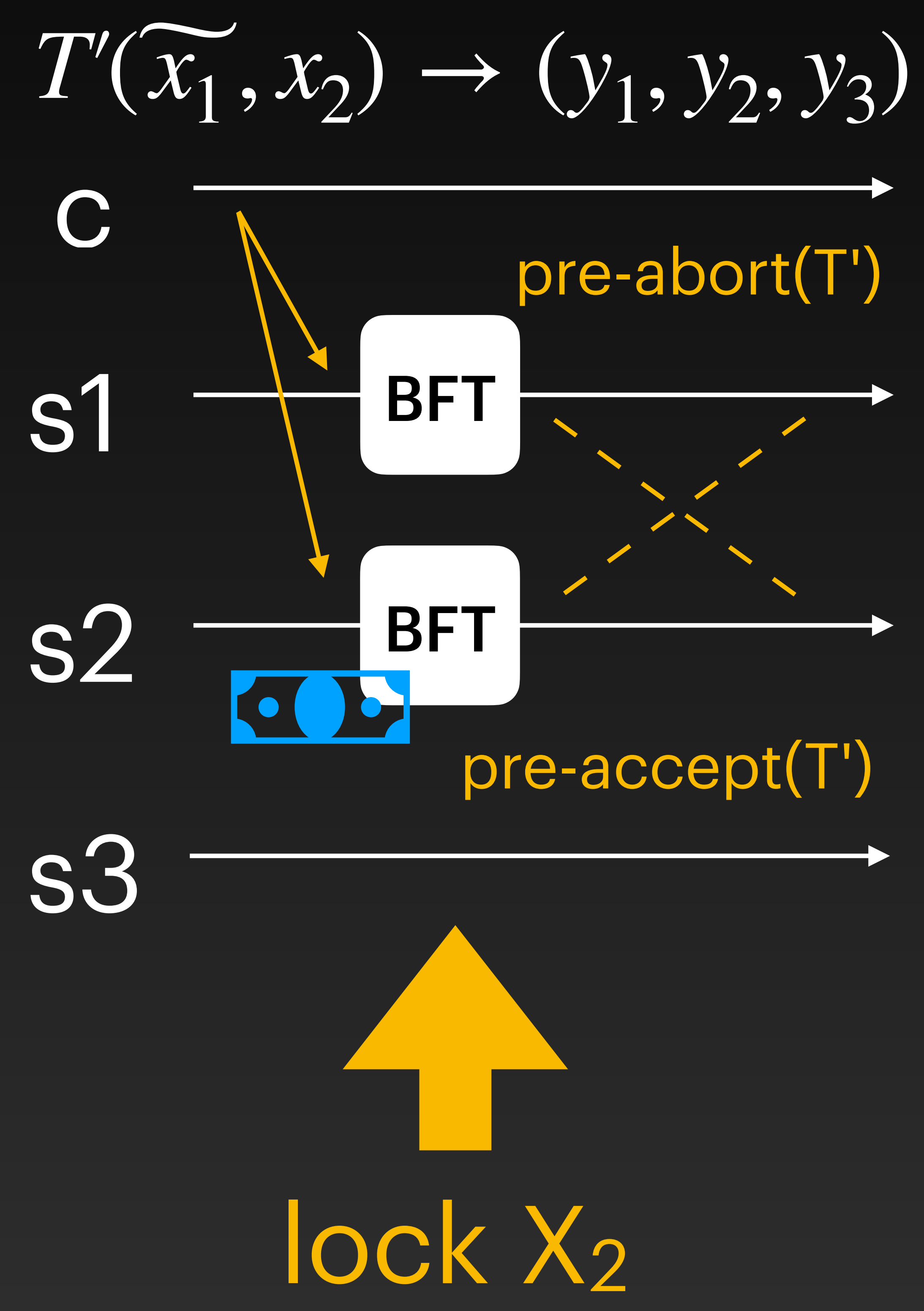
Attack against S-BAC

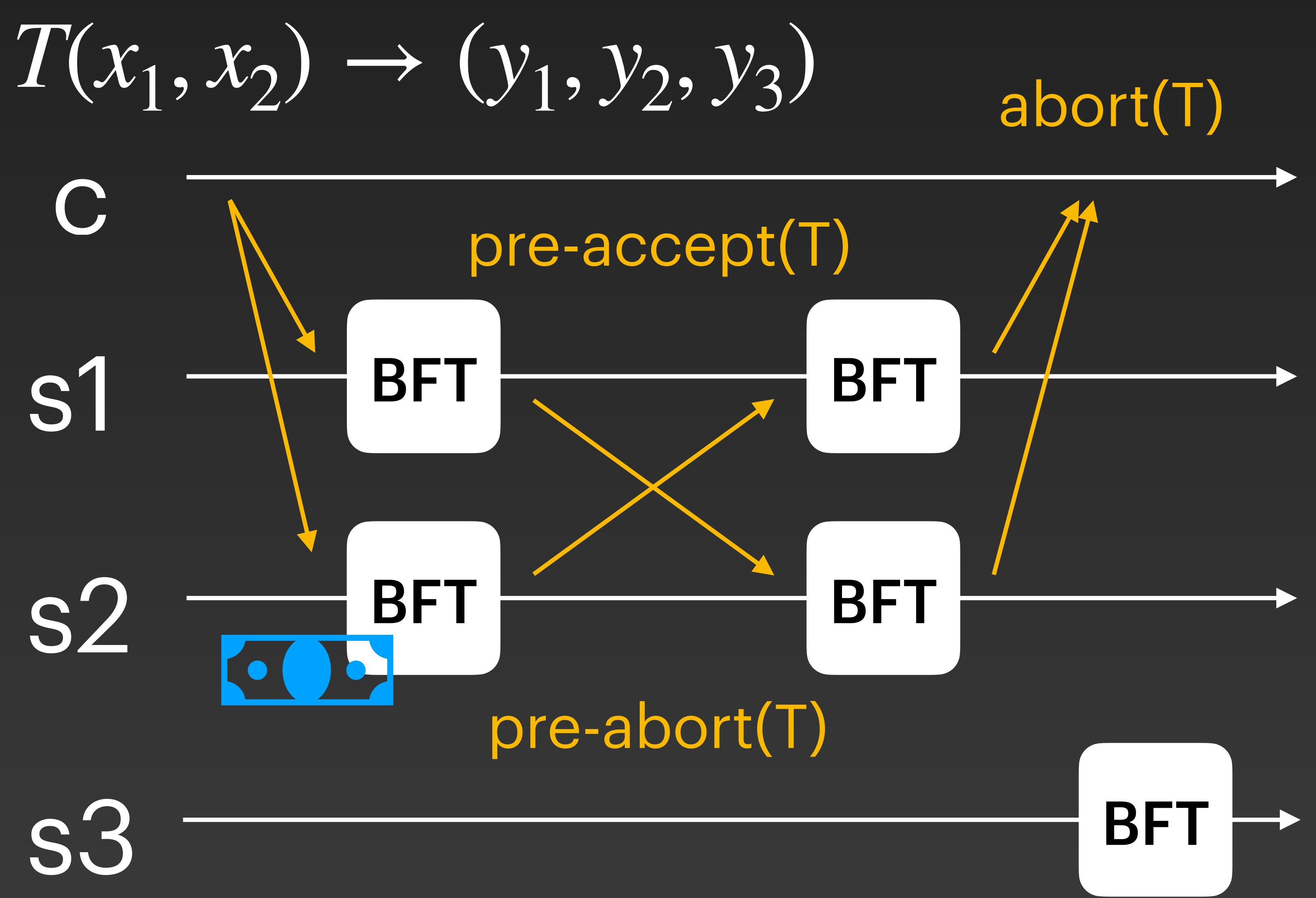
Double-spend X_1



Attack against S-BAC

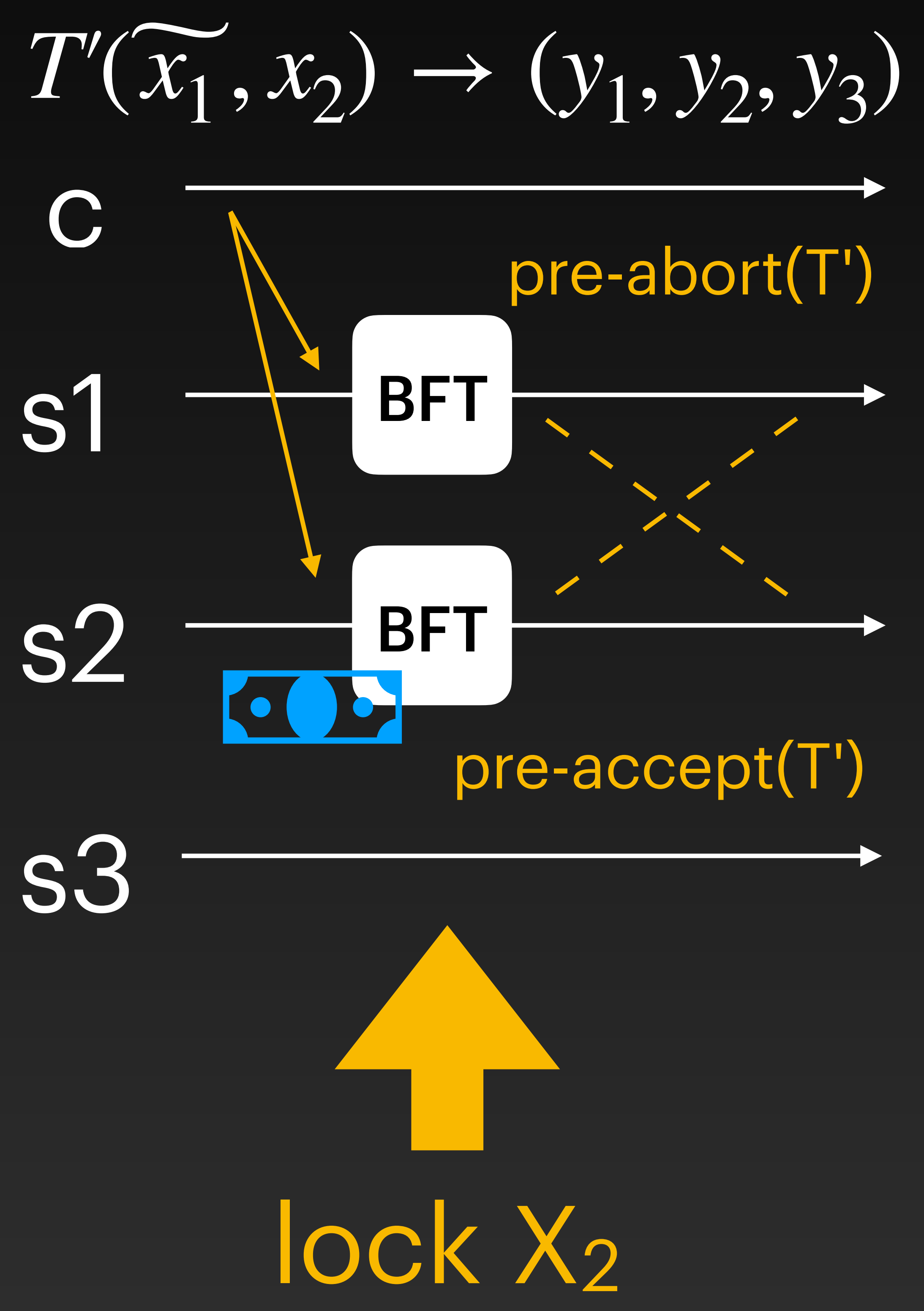
Double-spend X_1

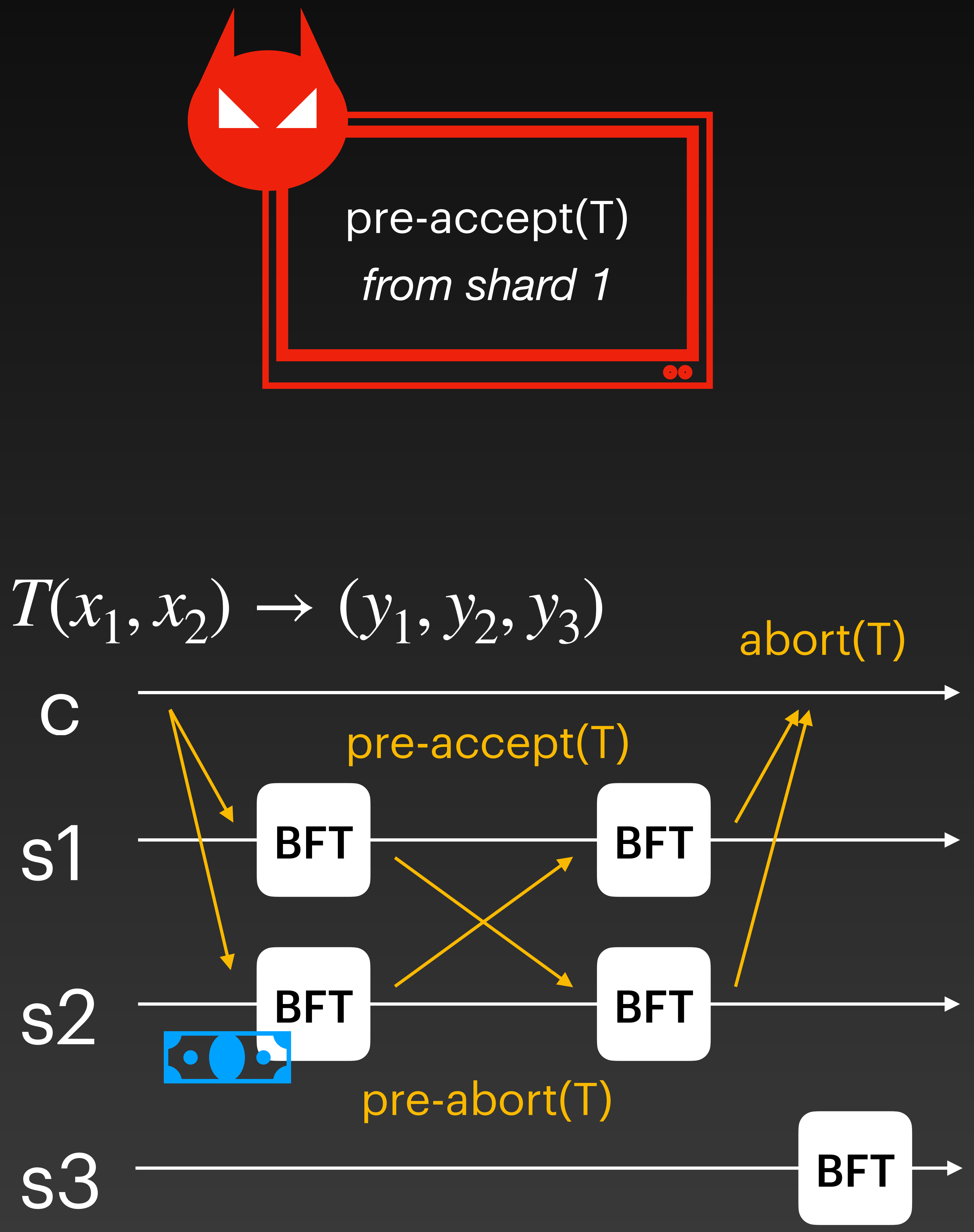




Attack against S-BAC

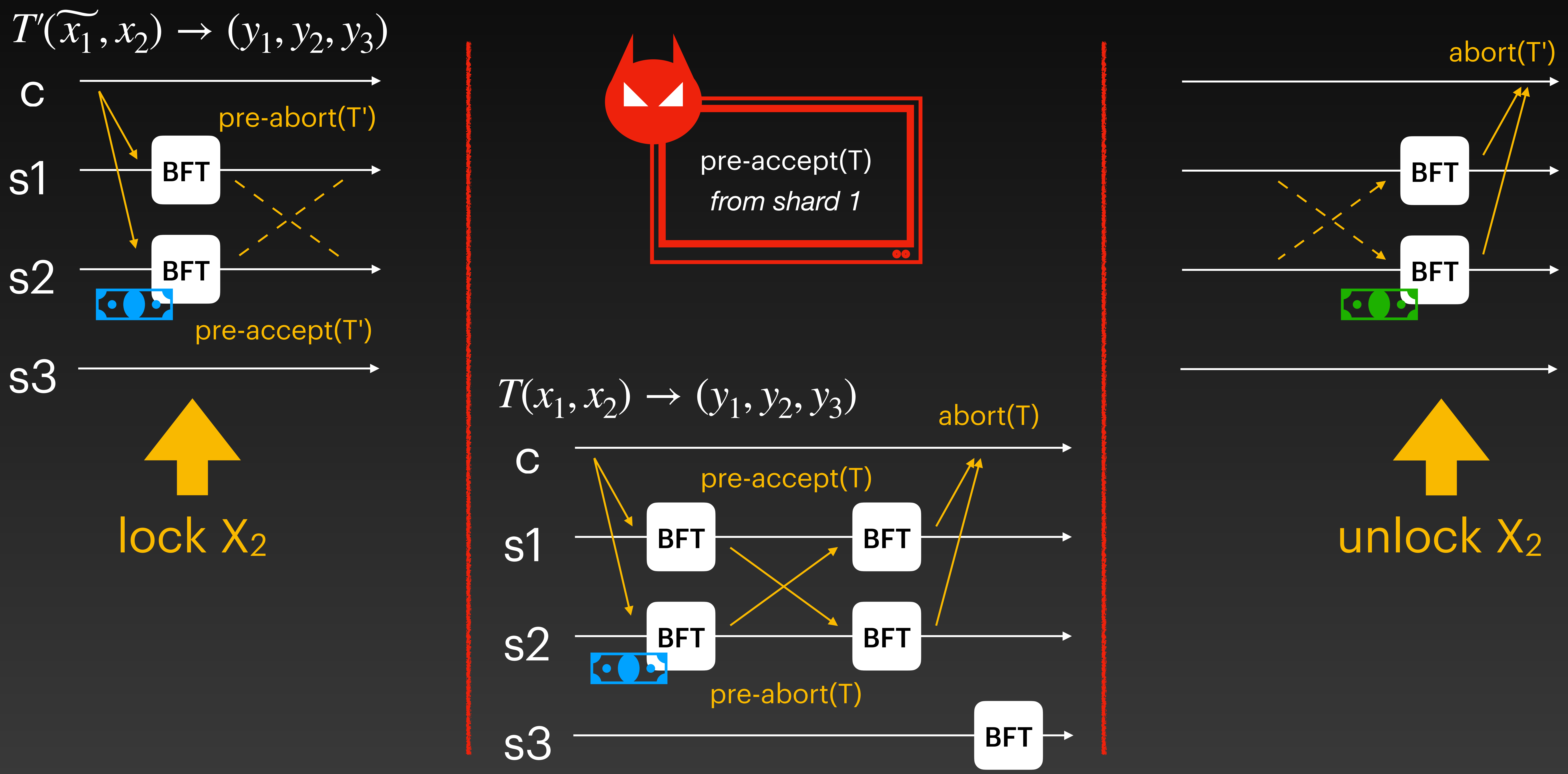
Double-spend X_1





Attack against S-BAC

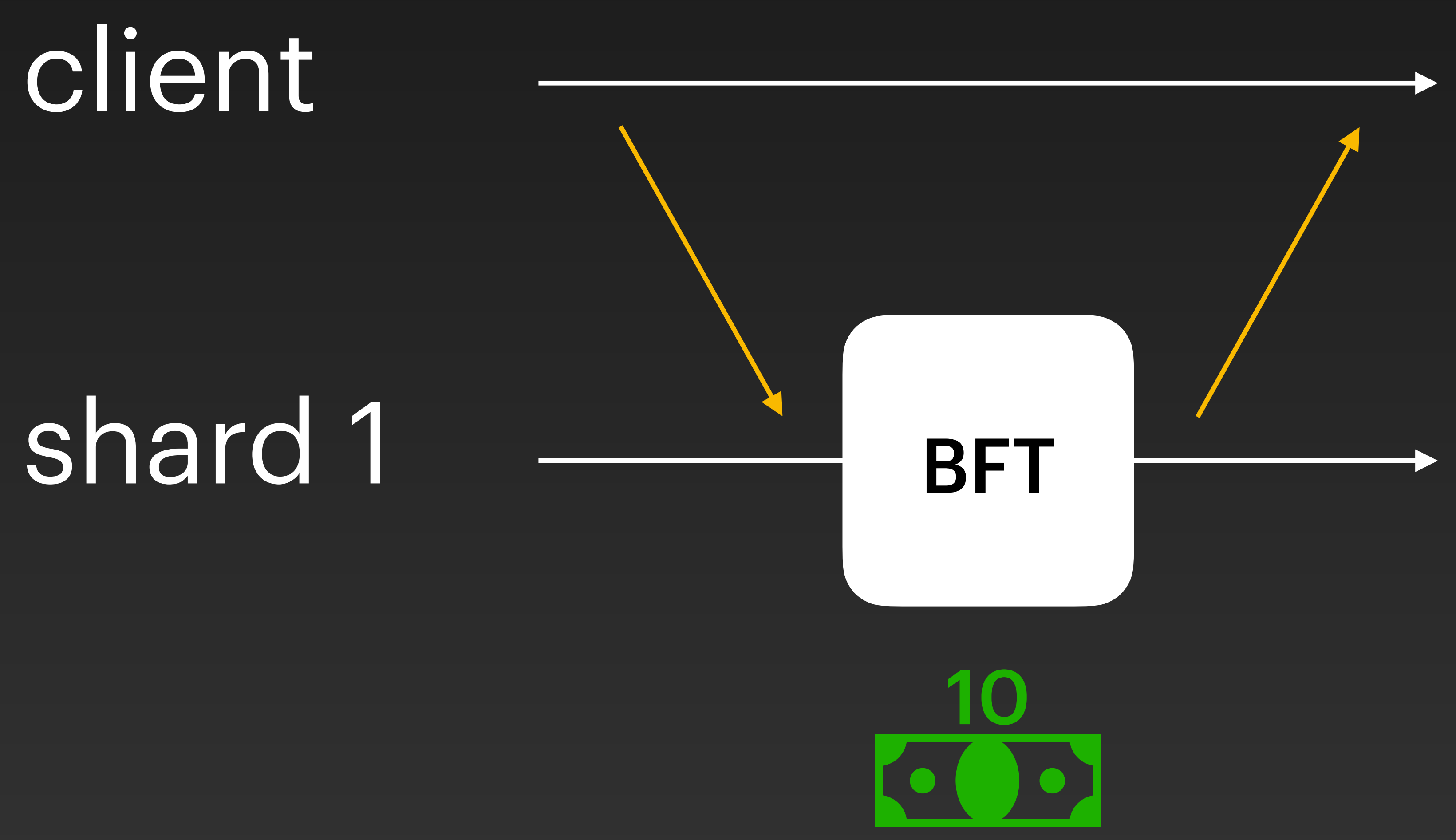
Double-spend X_1



Attack against S-BAC

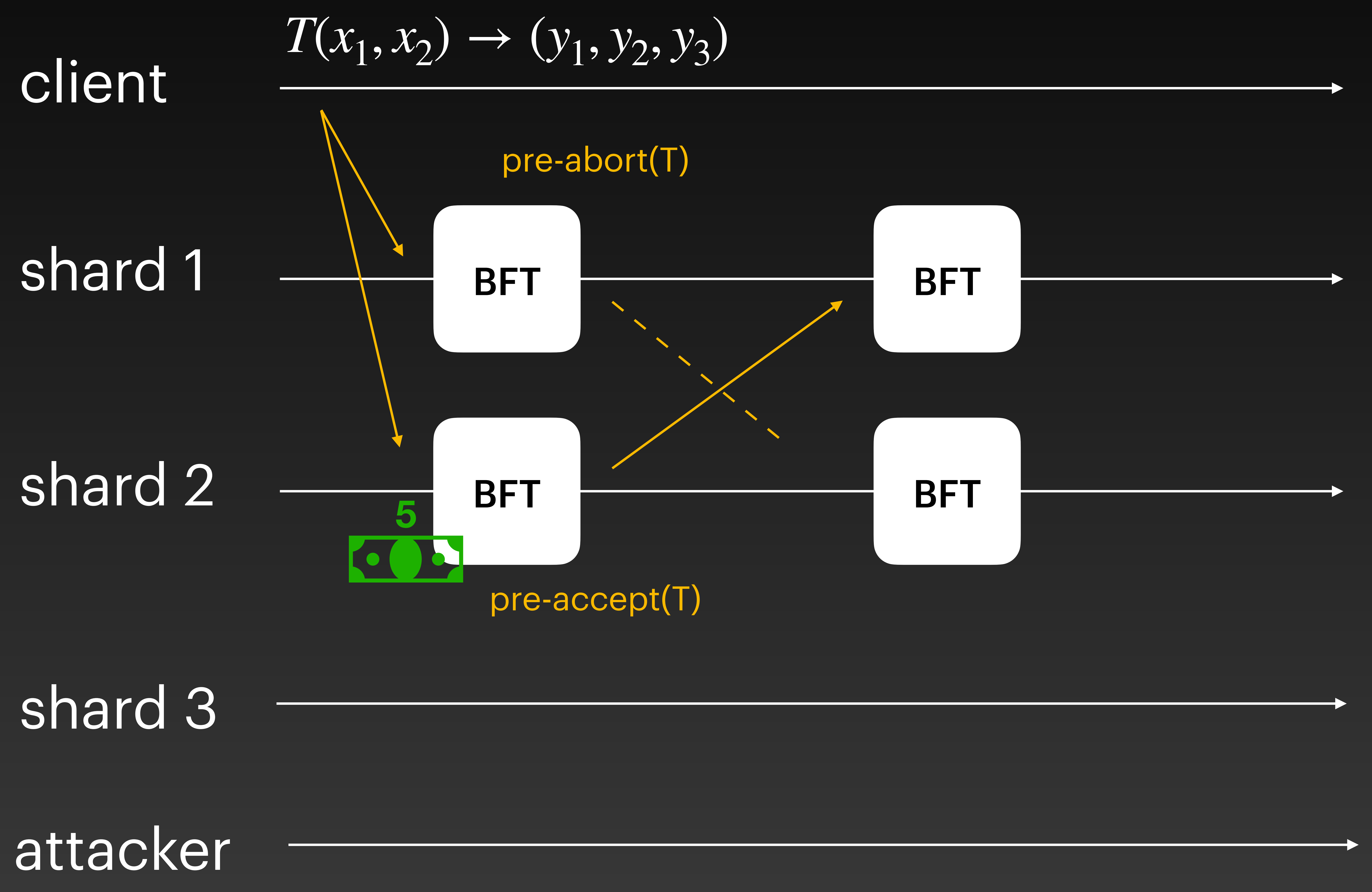
Double-spend X_1

$$T^*(x_1) \rightarrow (y_*)$$



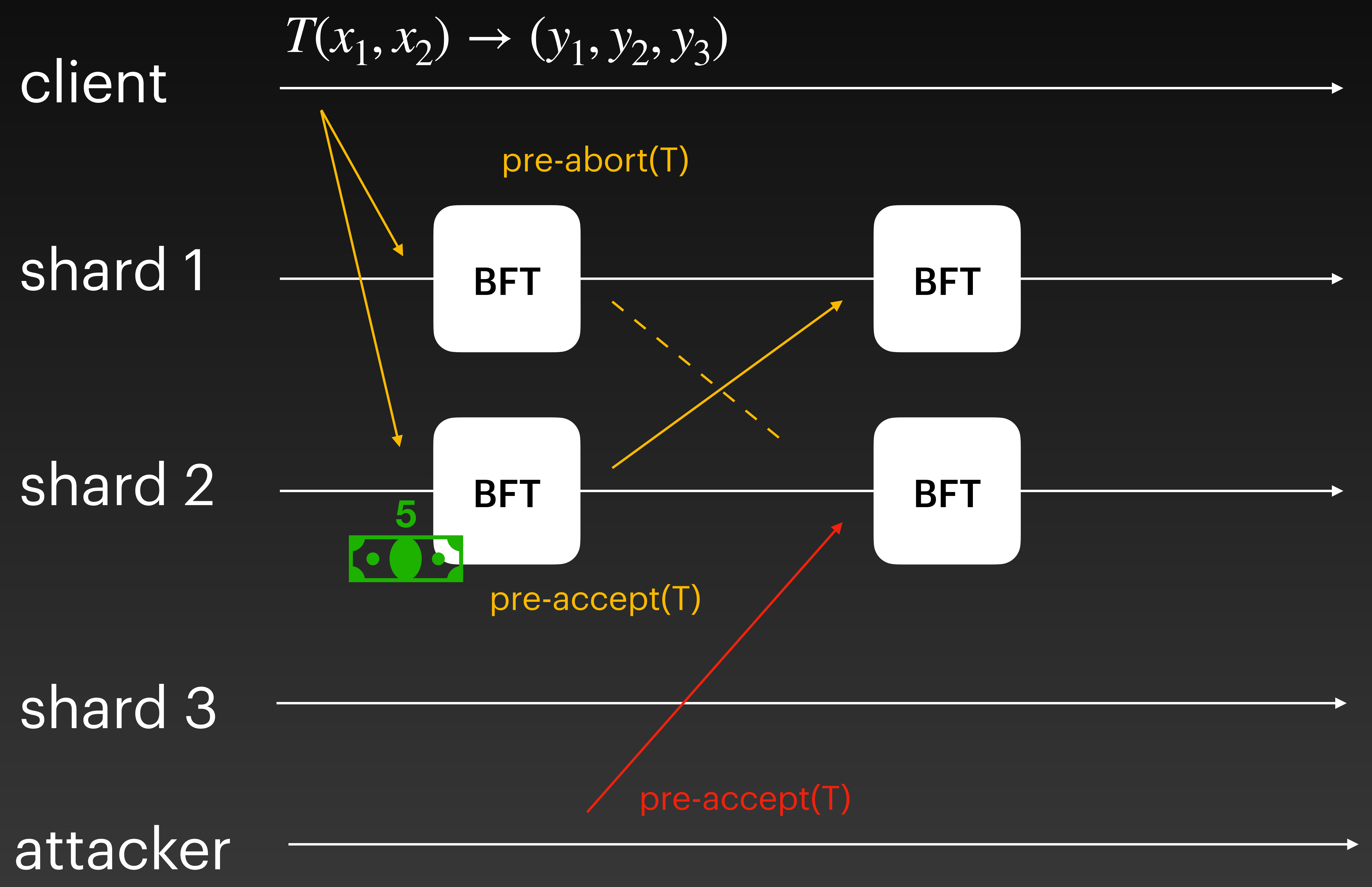
Attack against S-BAC

Double-spend X_1



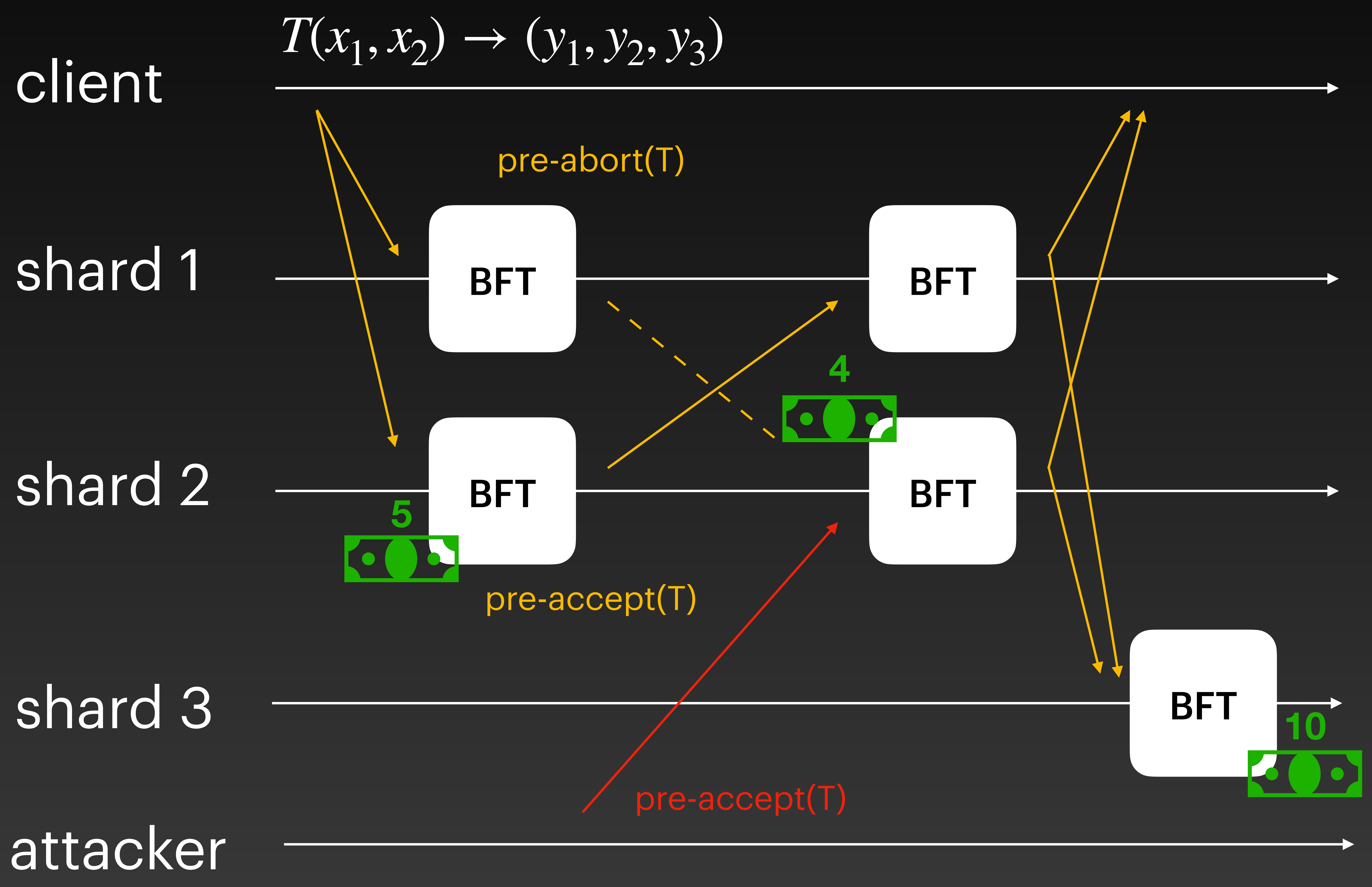
Attack against S-BAC

Double-spend X_1



Attack against S-BAC

Double-spend X_1



Attack against S-BAC

Double-spend X_1

Before attack

X_1 10

X_2 5

After attack

Y_* 10

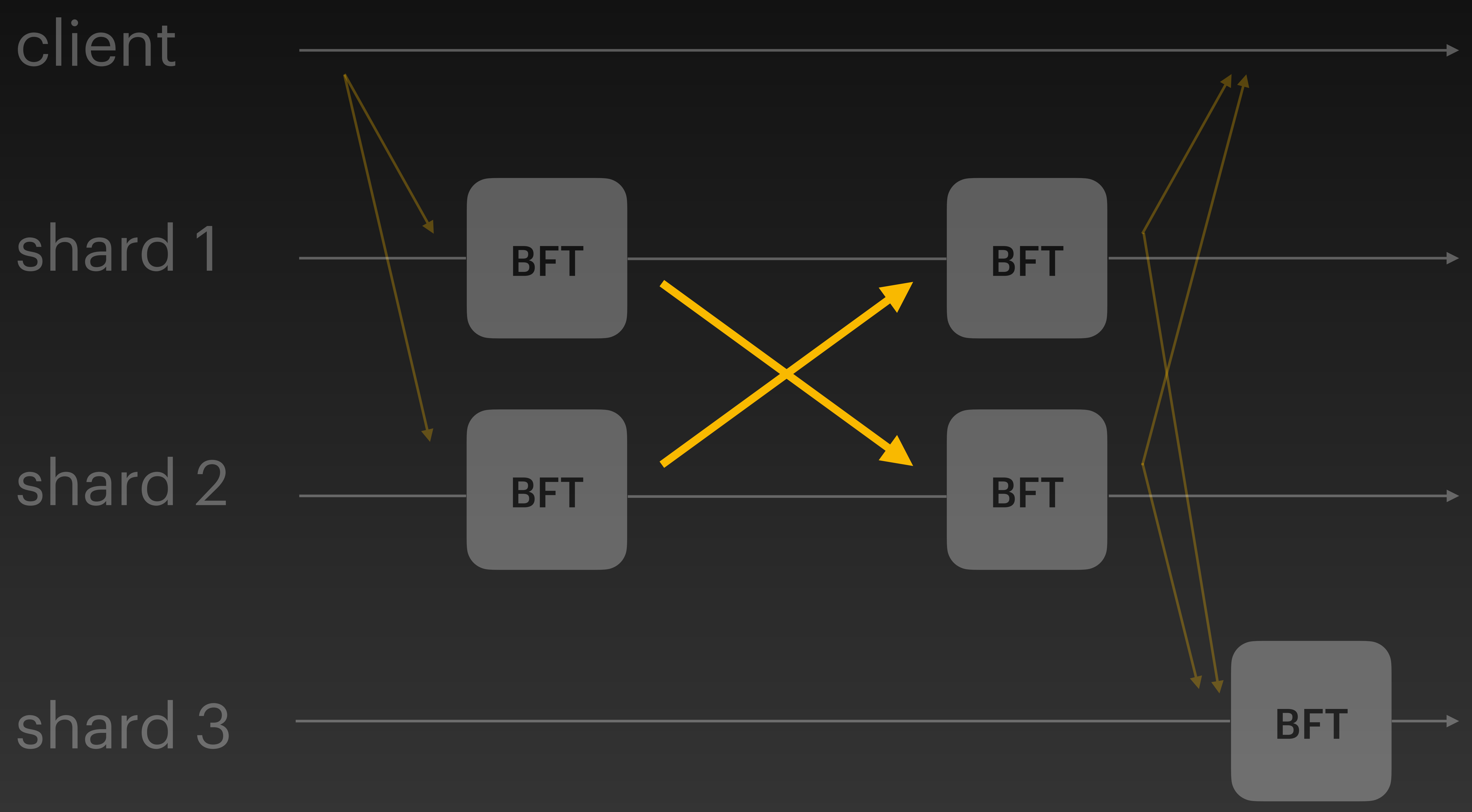
Y_2 4

Y_3 10

If it is not implemented, it does not work

Attacks against S-BAC

First phase



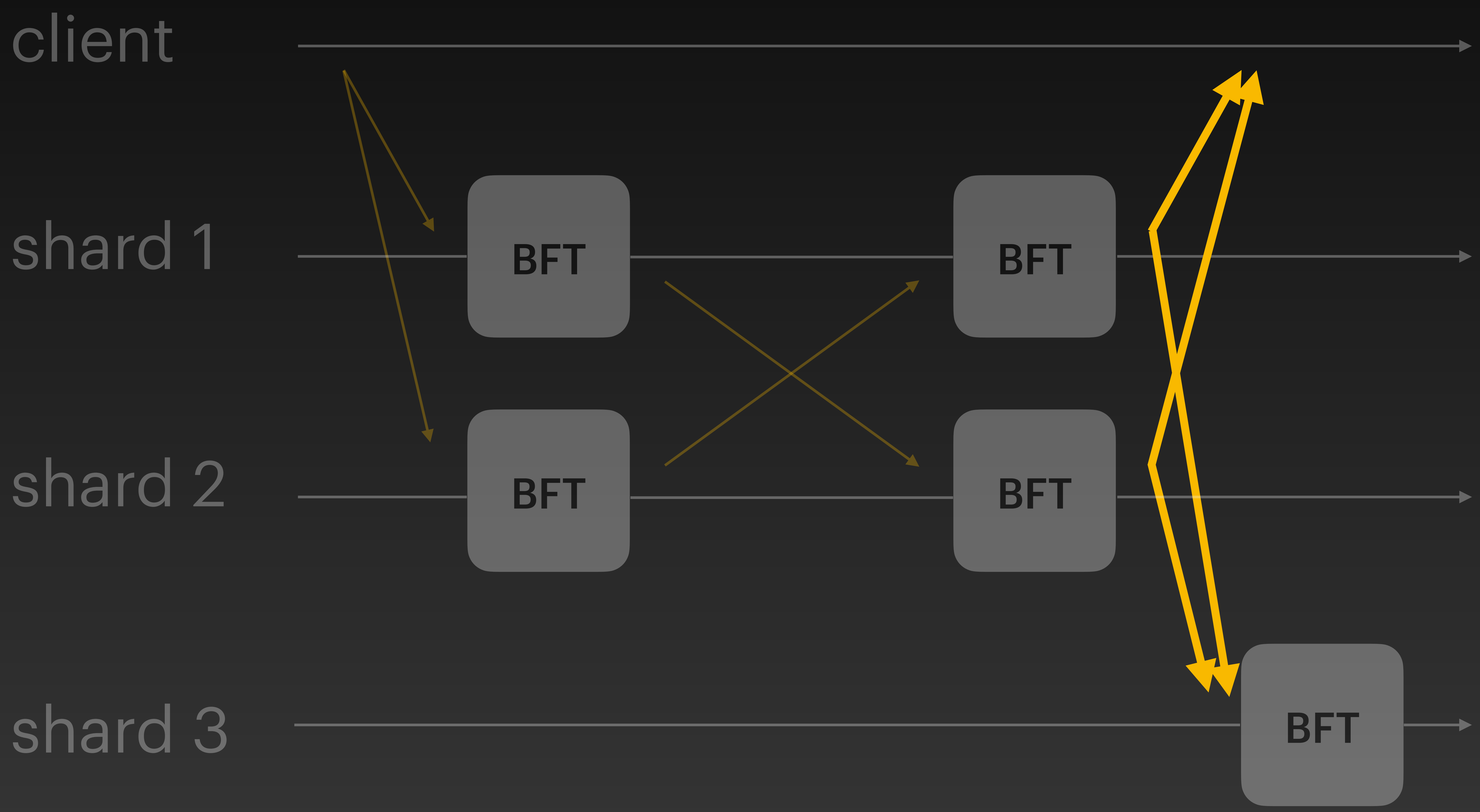
Attacks against S-BAC

First phase

	Phase 1 of S-BAC		Phase 2 of S-BAC		
	Shard 1 (potential victim)	**Shard 2** (potential victim)	**Shard 1** (potential victim)	**Shard 2** (potential victim)	**Shard 3** (potential victim)
1	pre-accept(T) lock x_1	pre-accept(T) lock x_2	accept(T) create y_1; inactivate x_1	accept(T) create y_2; inactivate x_2	- create y_3
2	▷pre-abort(T)		accept(T) create y_1; inactivate x_1	abort(T) unlock x_2	- create y_3
3		▷pre-abort(T)	abort(T) unlock x_1	accept(T) create y_2; inactivate x_2	- create y_3
4	▷pre-abort(T)	▷pre-abort(T)	abort(T) unlock x_1	abort(T) unlock x_2	-
5	pre-abort(T) -	pre-accept(T) lock x_2	abort(T) -	abort(T) unlock x_2	-
6	▷pre-accept(T)		abort(T) -	accept(T) create y_2; inactivate x_2	- create y_3
7	pre-accept(T) lock x_1	pre-abort(T) -	abort(T) unlock x_1	abort(T) -	-
8		▷ pre-accept(T)	accept(T) create y_1; inactivate x_1	abort(T) -	- create y_3
9	pre-abort(T) -	pre-abort(T) -	abort(T) -	abort(T) -	-

Attacks against S-BAC

Second phase



Attacks against S-BAC

Second phase

Phase 2 of S-BAC

	Shard 1	Shard 2	Shard 3 (potential victim)
1	accept(T) create y_1; inactivate x_1	accept(T) create y_2; inactivate x_2	- create y_3
2	▷accept(T)		create y_3
3		▷accept(T)	create y_3
4	▷accept(T)	▷accept(T)	create y_3
5	abort(T) (unlock x_1)	abort(T) (unlock x_2)	- -
6	▷accept(T)		create y_3
7		▷accept(T)	create y_3
8	▷accept(T)	▷accept(T)	create y_3

What causes these issues?

Issue 1. Input shards cannot associate protocol messages to a specific protocol execution.

Issue 2. Output shards (that are not also input shards) do not experience the first phase of the protocol

Easy Fix?

Global sequence numbers?

~~Linear Scalability~~

Wait for messages to arrive?

~~Hight Throughput~~

Byzcuit

S-BAC + Atomix

Byzcuit

Acknowledgments

Mustafa Al-Bassam

Alberto Sonnino

Bano Shehar

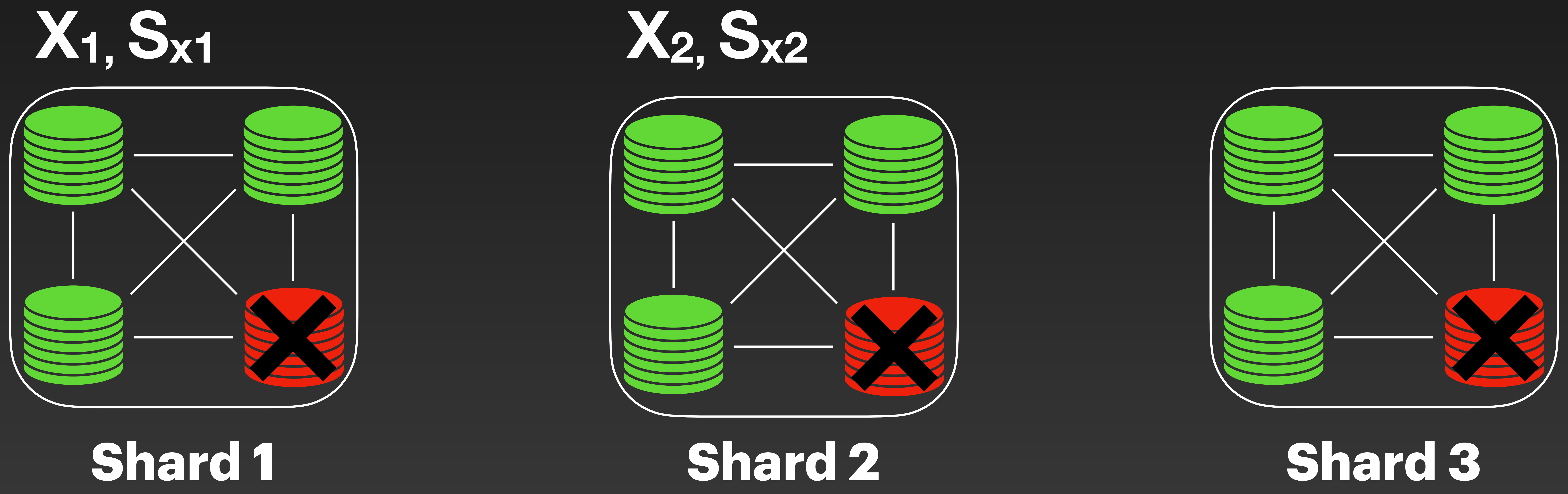
George Danezis

University College London

Byzcuit

Fix issue 1

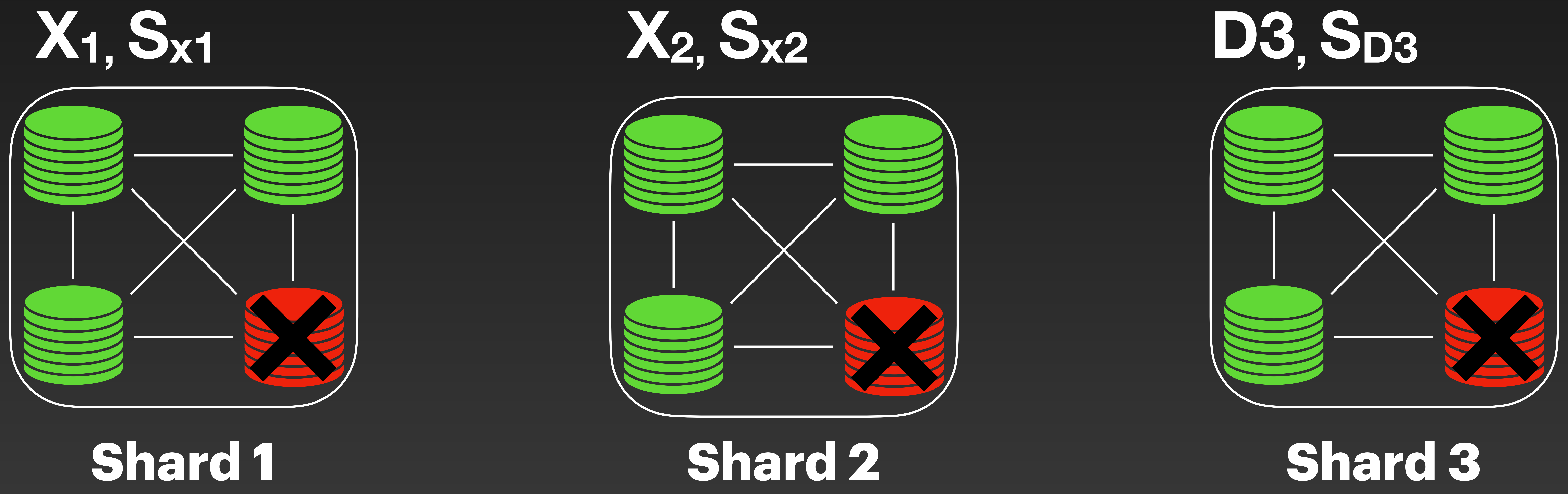
Add sequence numbers per object



Byzcuit

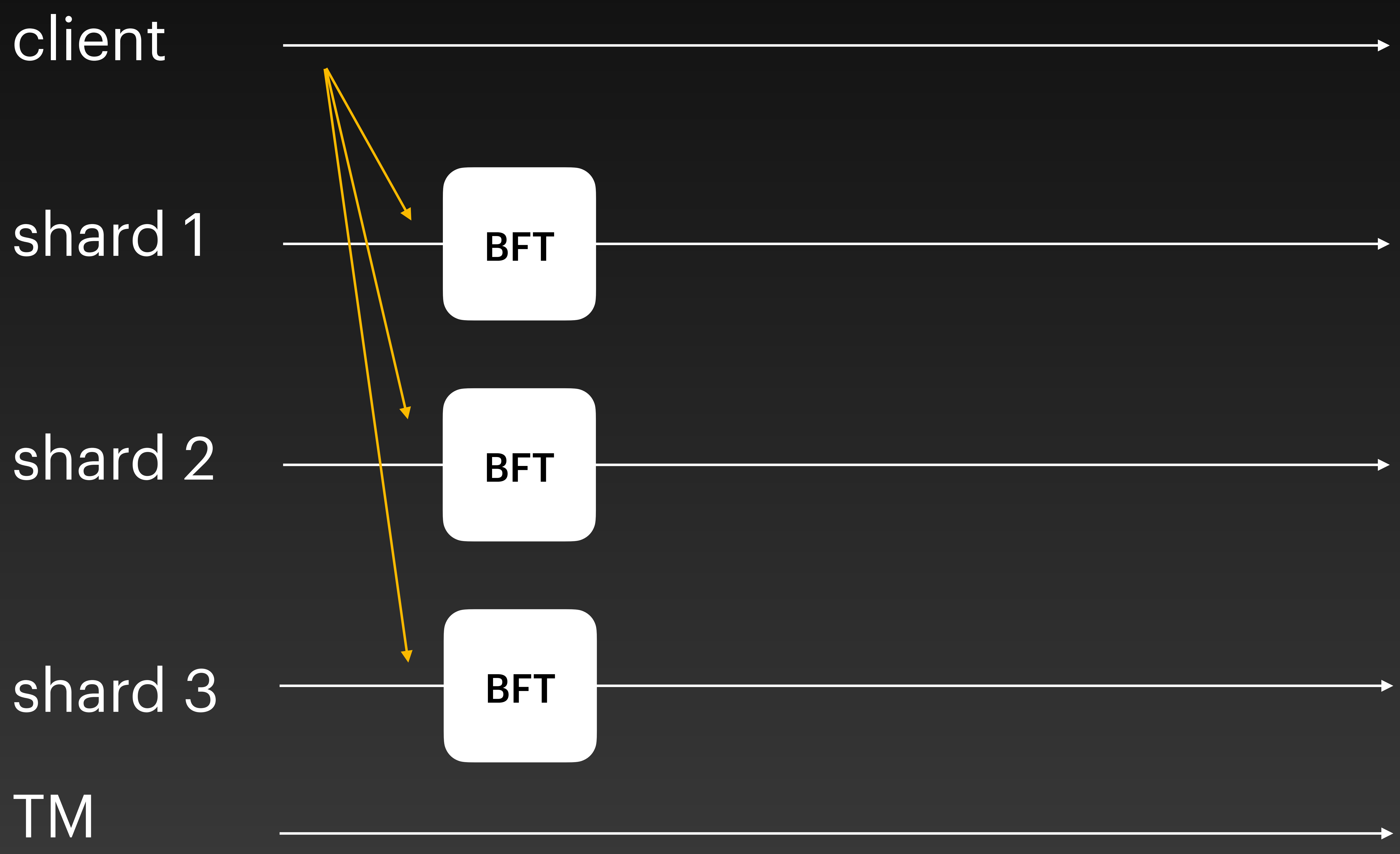
Fix issue 2

Dummy objects for output shards



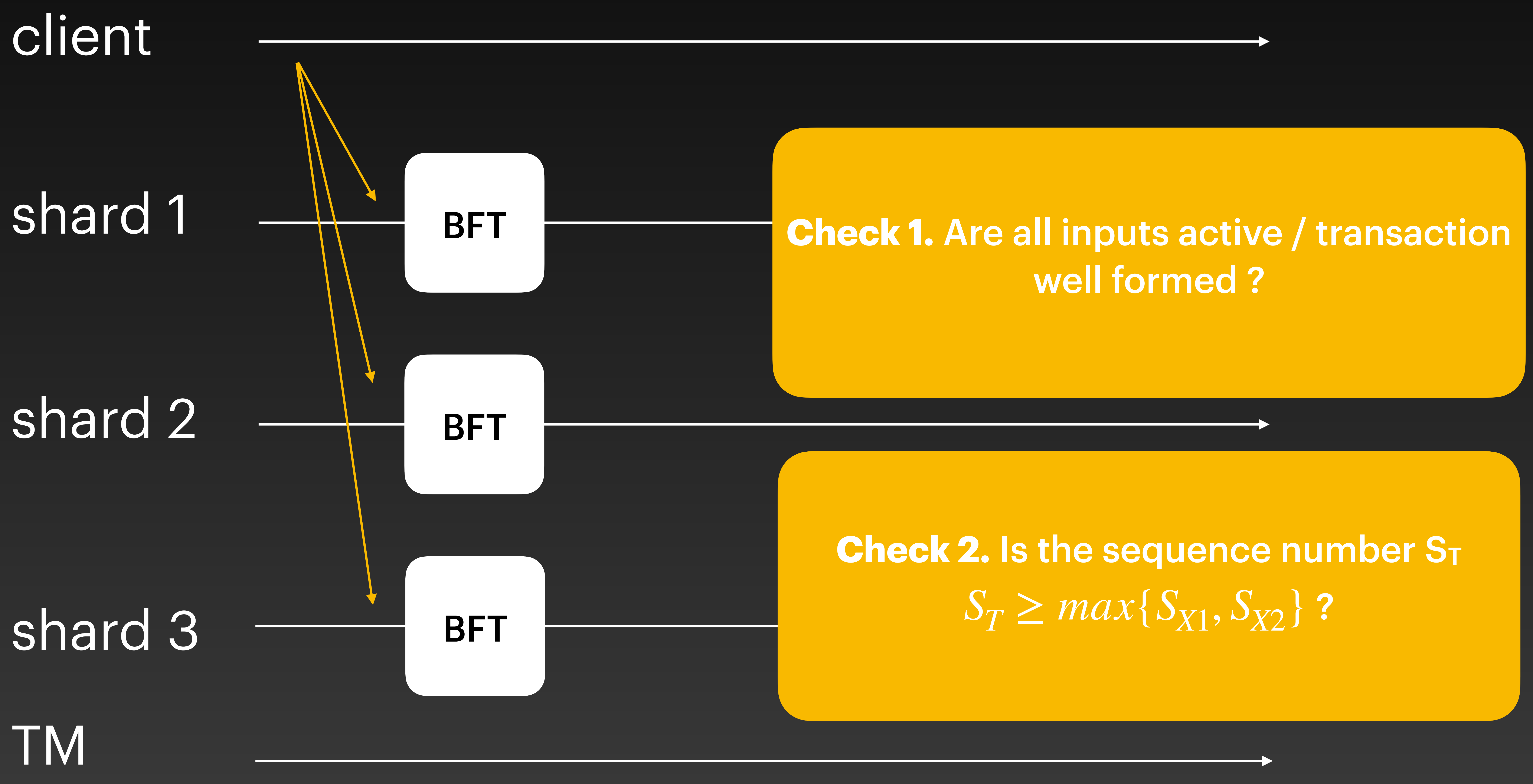
Byzcuit

$$\{S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3)\}$$



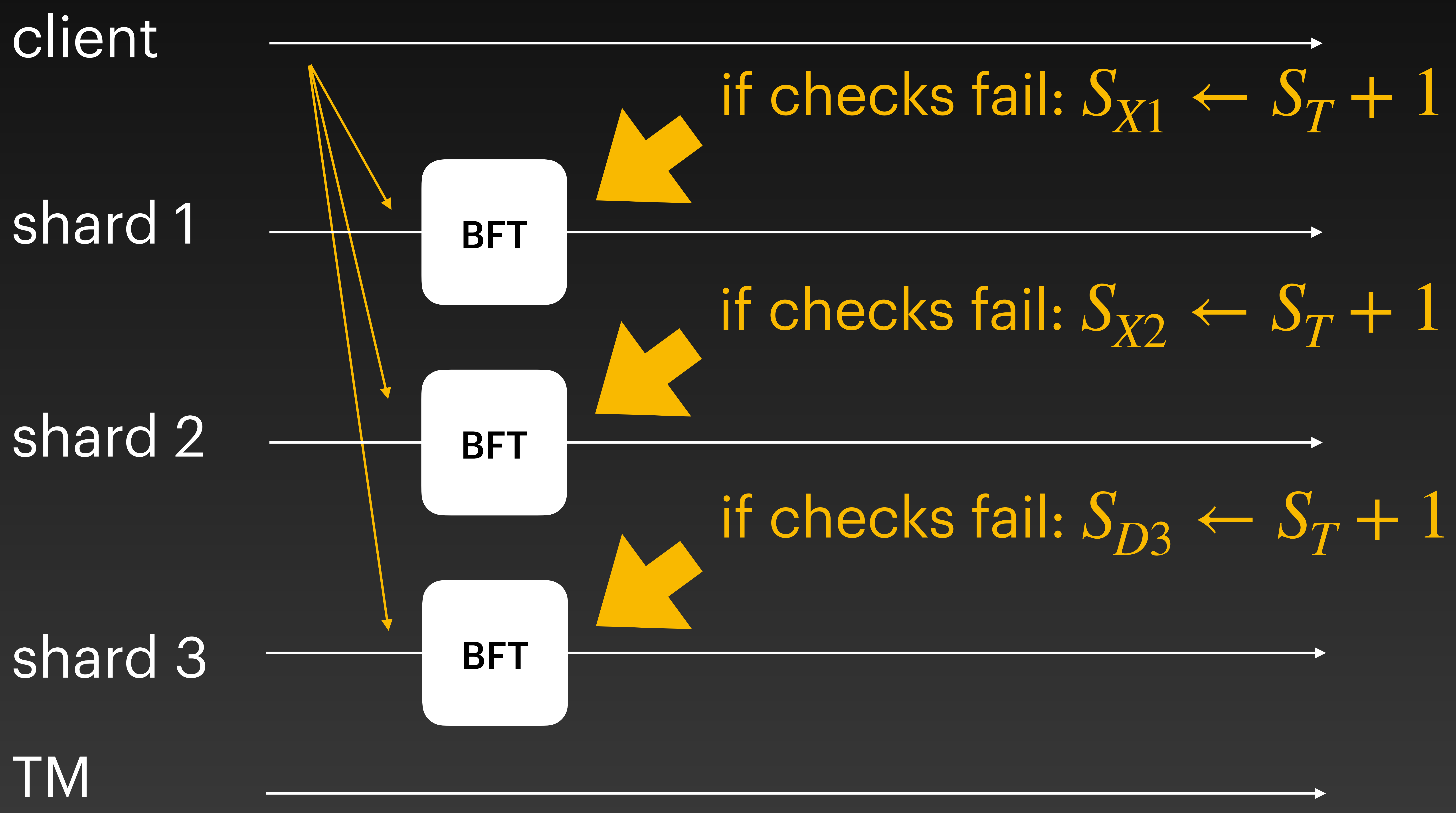
Byzcuit

$$\{S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3)\}$$



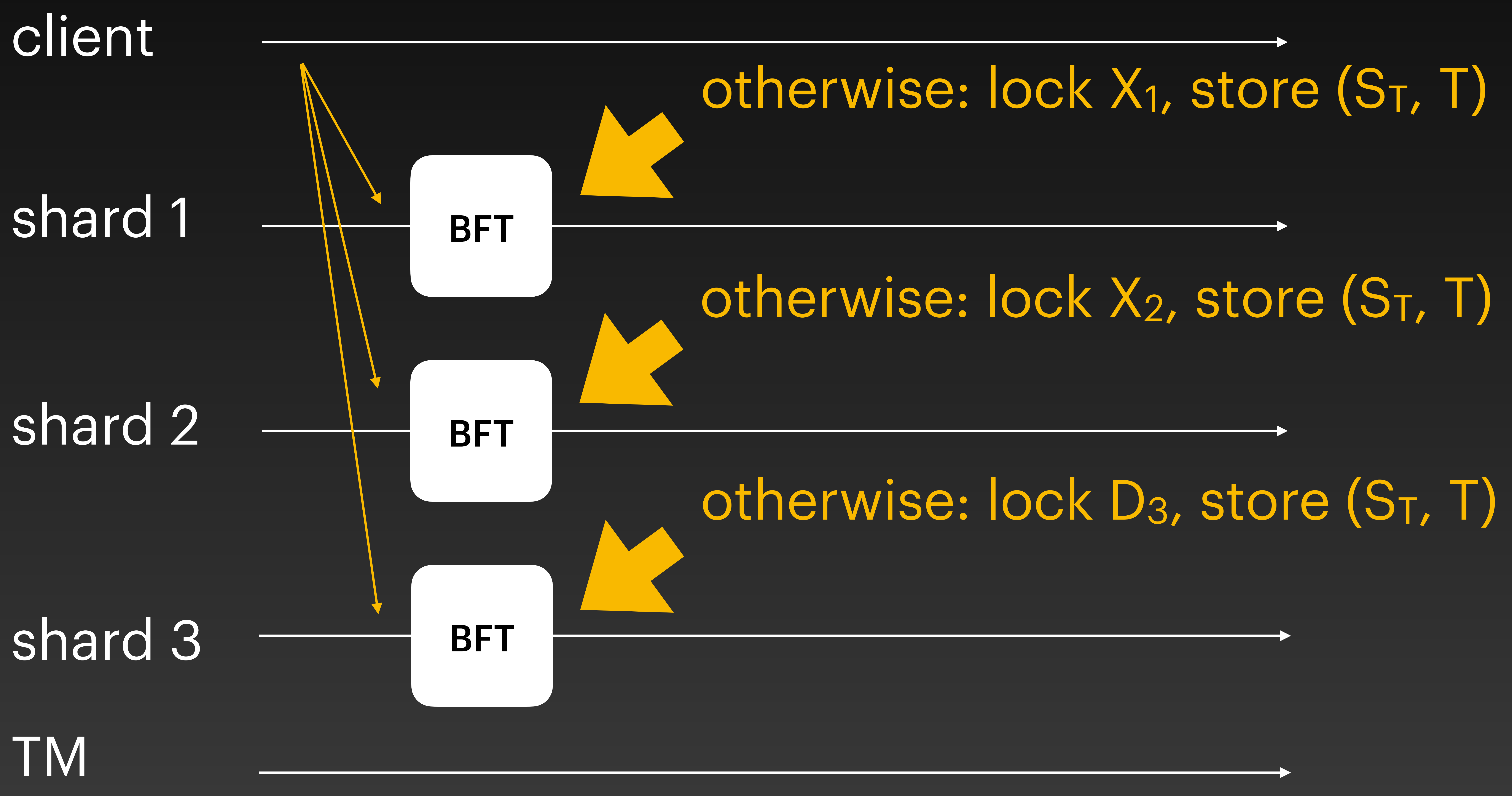
Byzcuit

$$\{S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3)\}$$



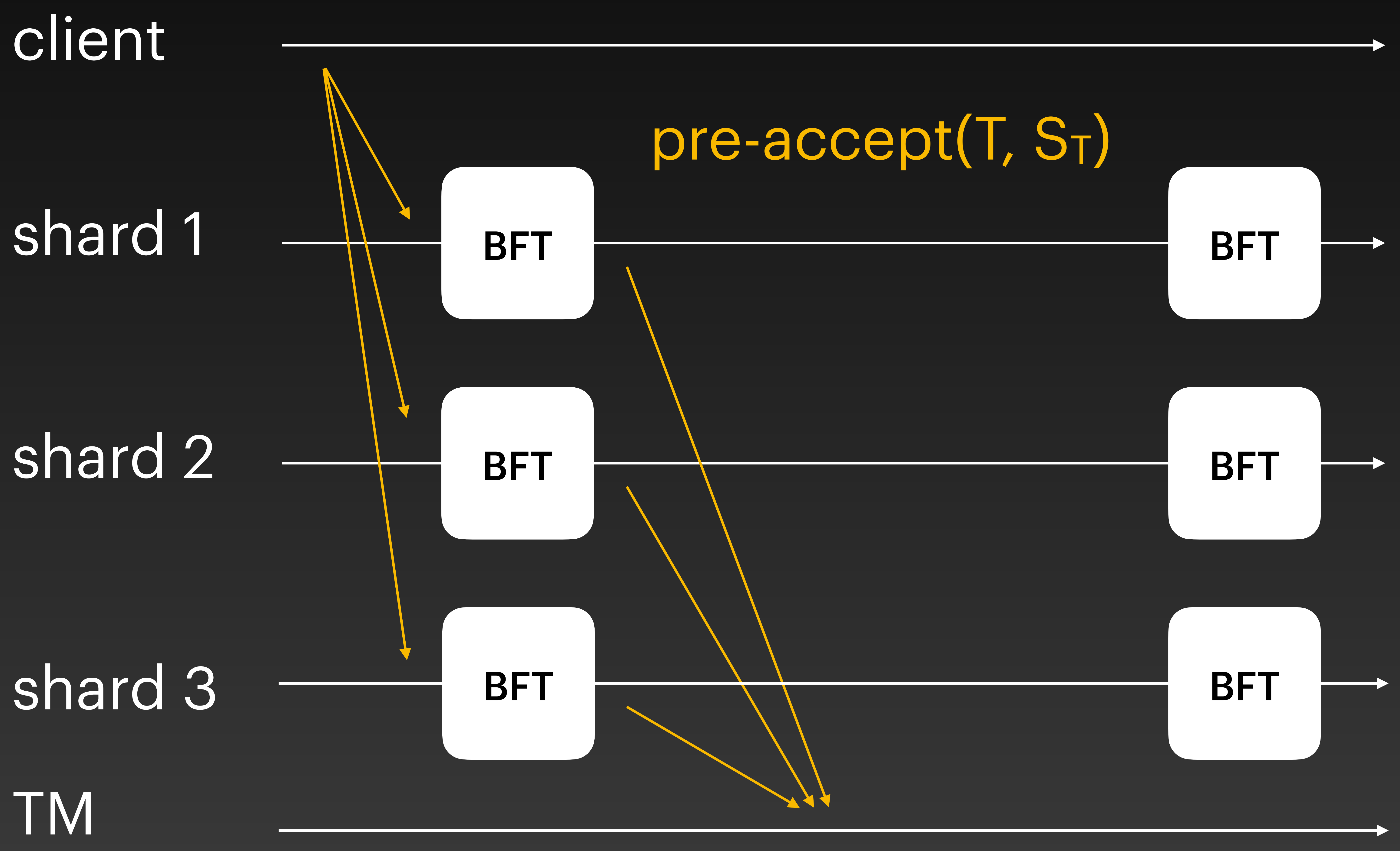
Byzcuit

$$\{S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3)\}$$



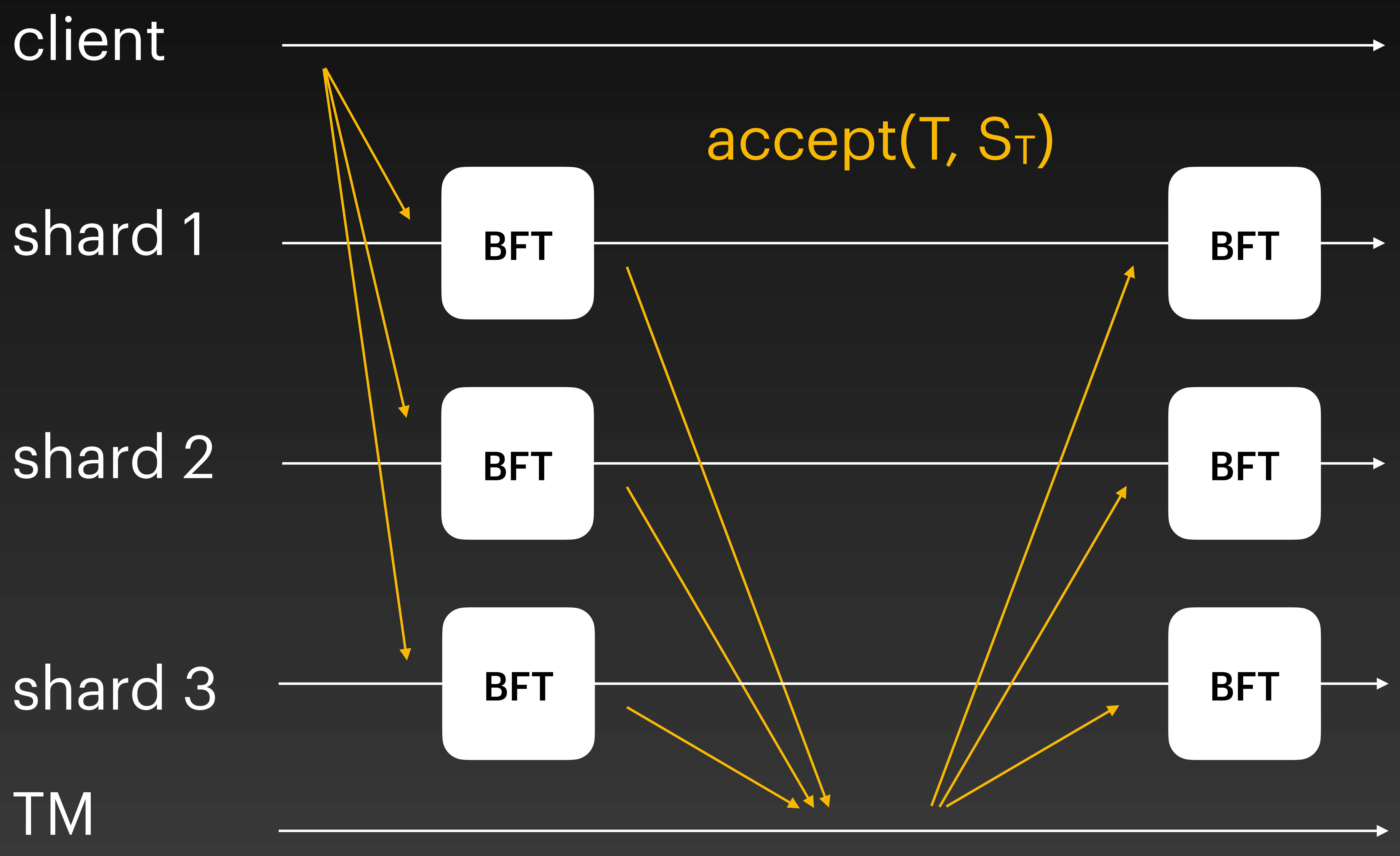
Byzcuit

$$\{S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3)\}$$



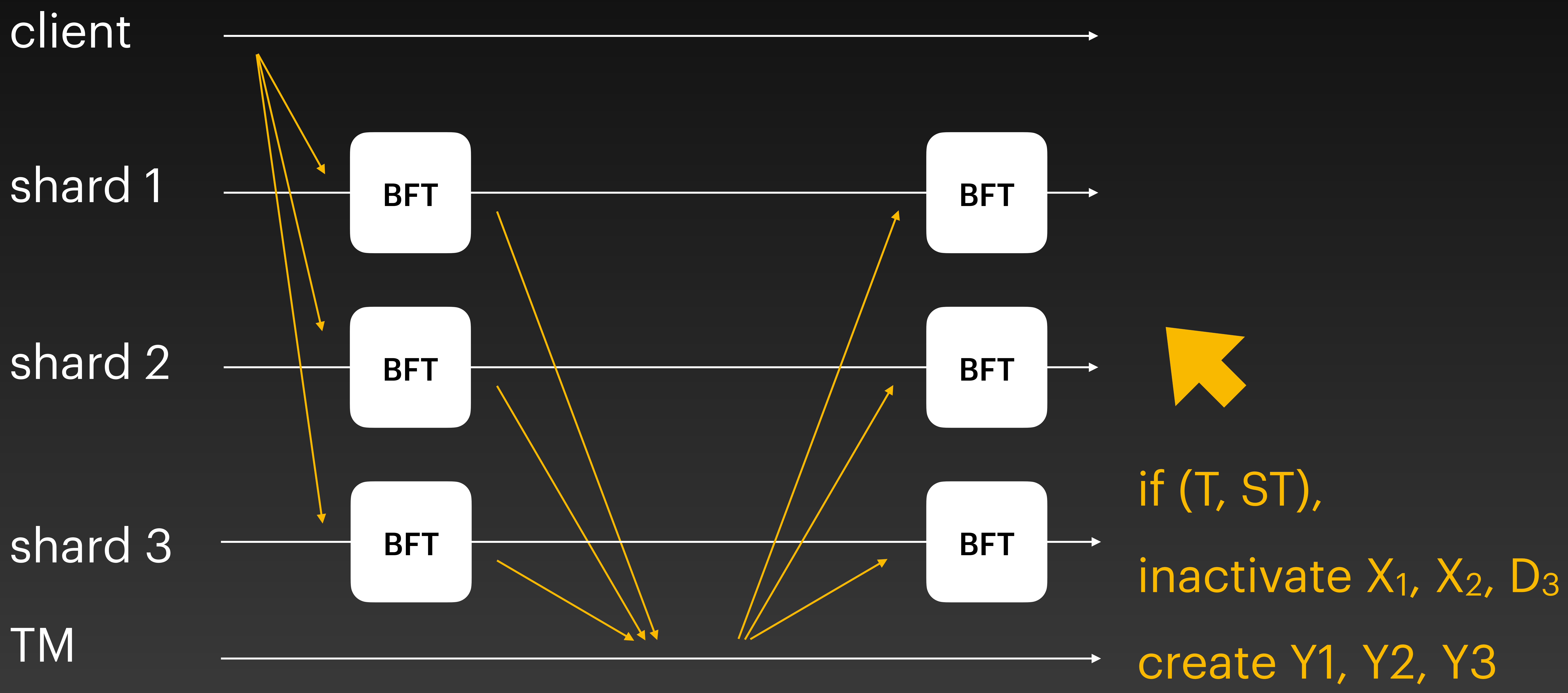
Byzcuit

$$\{S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3)\}$$



Byzcuit

$$\{S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3)\}$$



Why is Byzcuit secure?

Issue 1. Input shards cannot associate protocol messages to a specific protocol execution.

Sequence numbers:

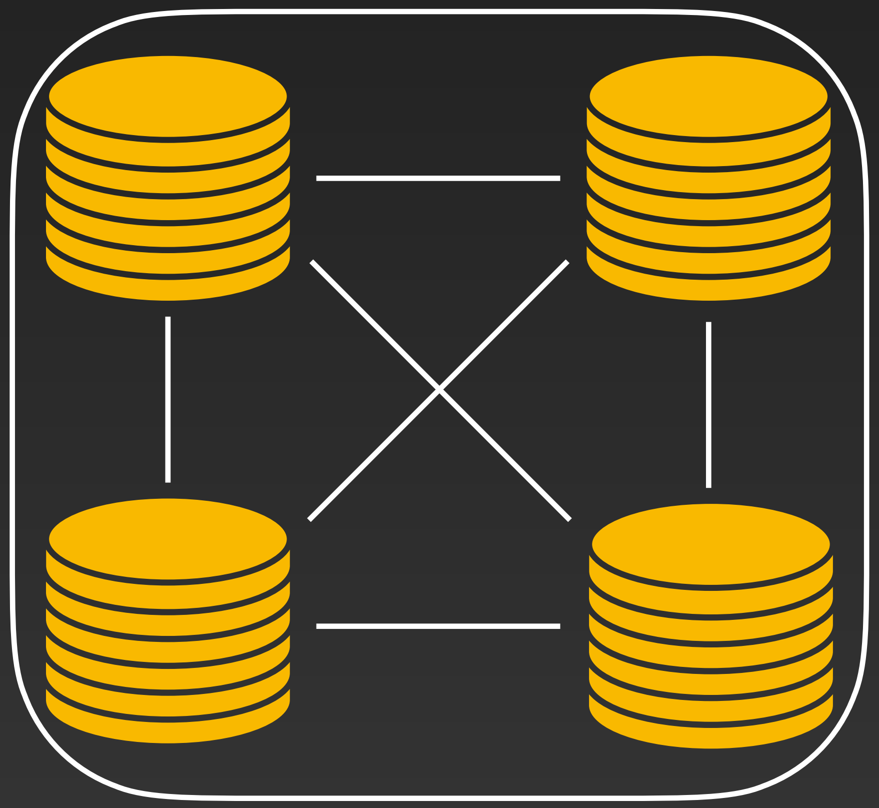
act as session ID

Issue 2. Output shards (that are not also input shards) do not experience the first phase of the protocol

Dummy objects:

all shards experience the first phase of the protocol

Anyone can be a TM



Byzcuit

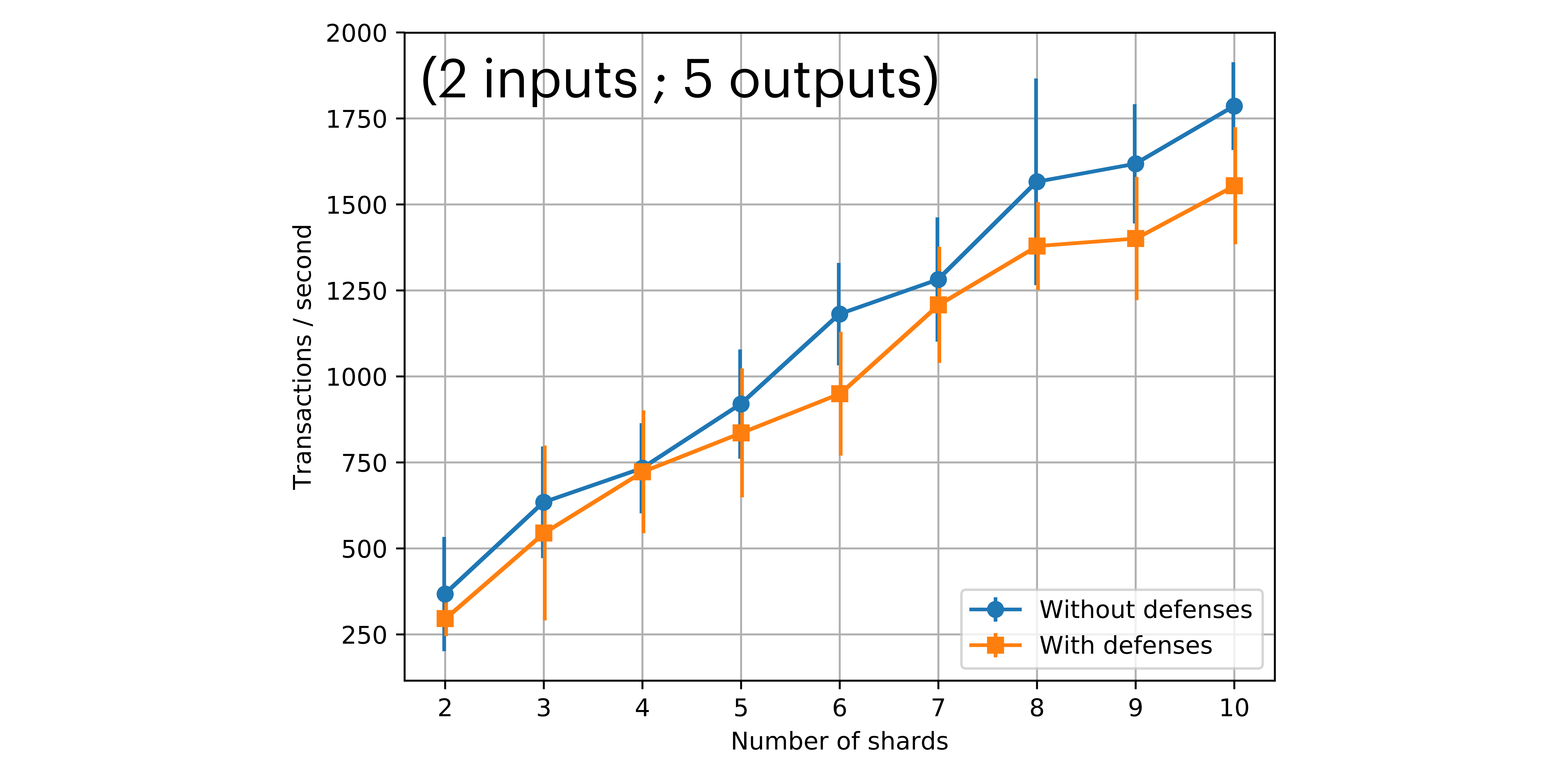
Implementation

- Fork of Java Chainspace
- Based on BFT-SMART
- Only a prototype to demonstrate its properties

https://github.com/sheharbano/byzcuit

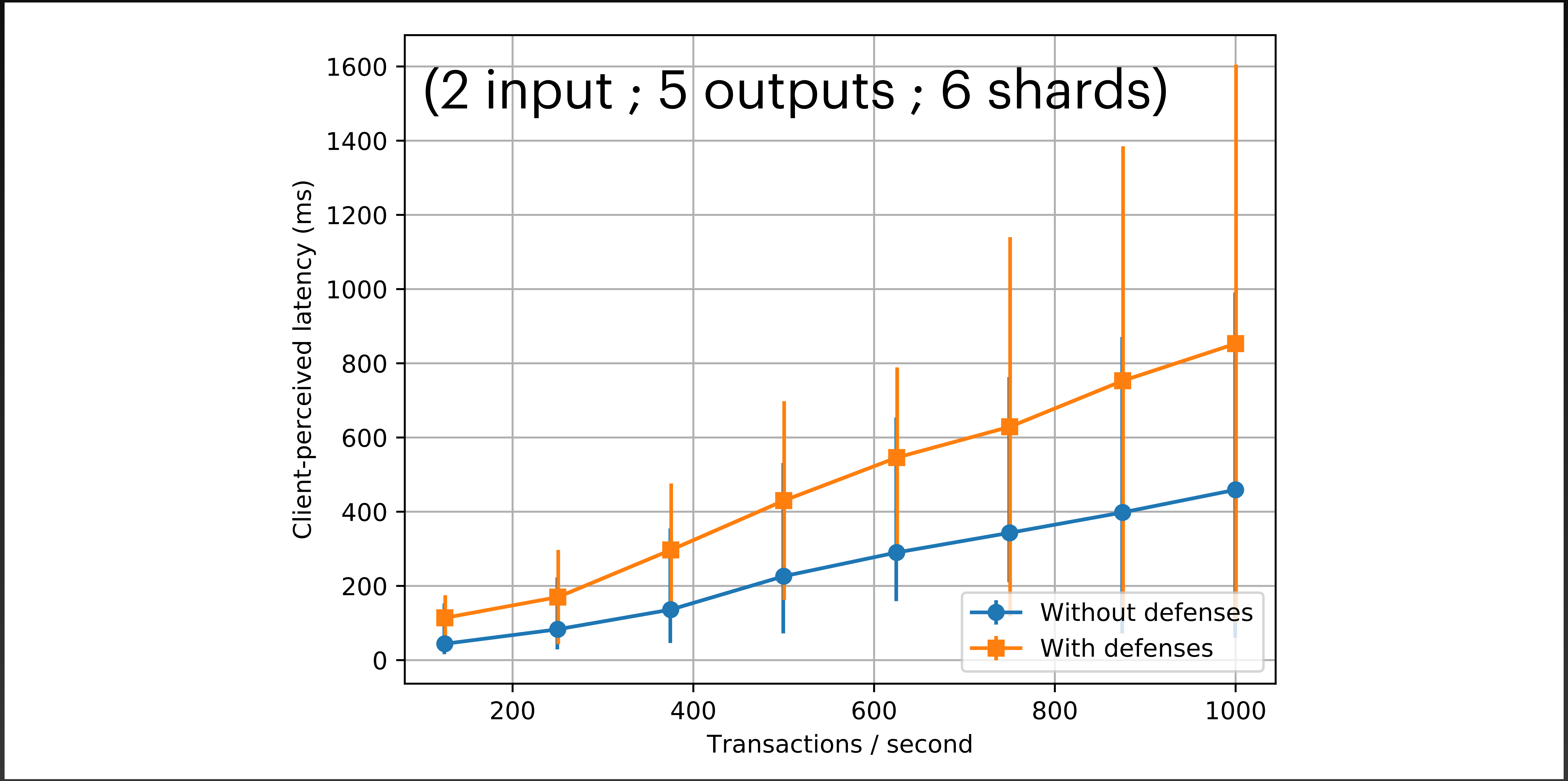
Byzcuit

Linear scalability



Byzcuit

Finality



Conclusion
Part I - Increasing Throughput

Byzcuit

- S-BAC + Atomix
- High throughput, linear scalability, BFT resilience, Fast finality

- **Paper:** https://arxiv.org/abs/1901.11218
- **Code:** https://github.com/sheharbano/byzcuit

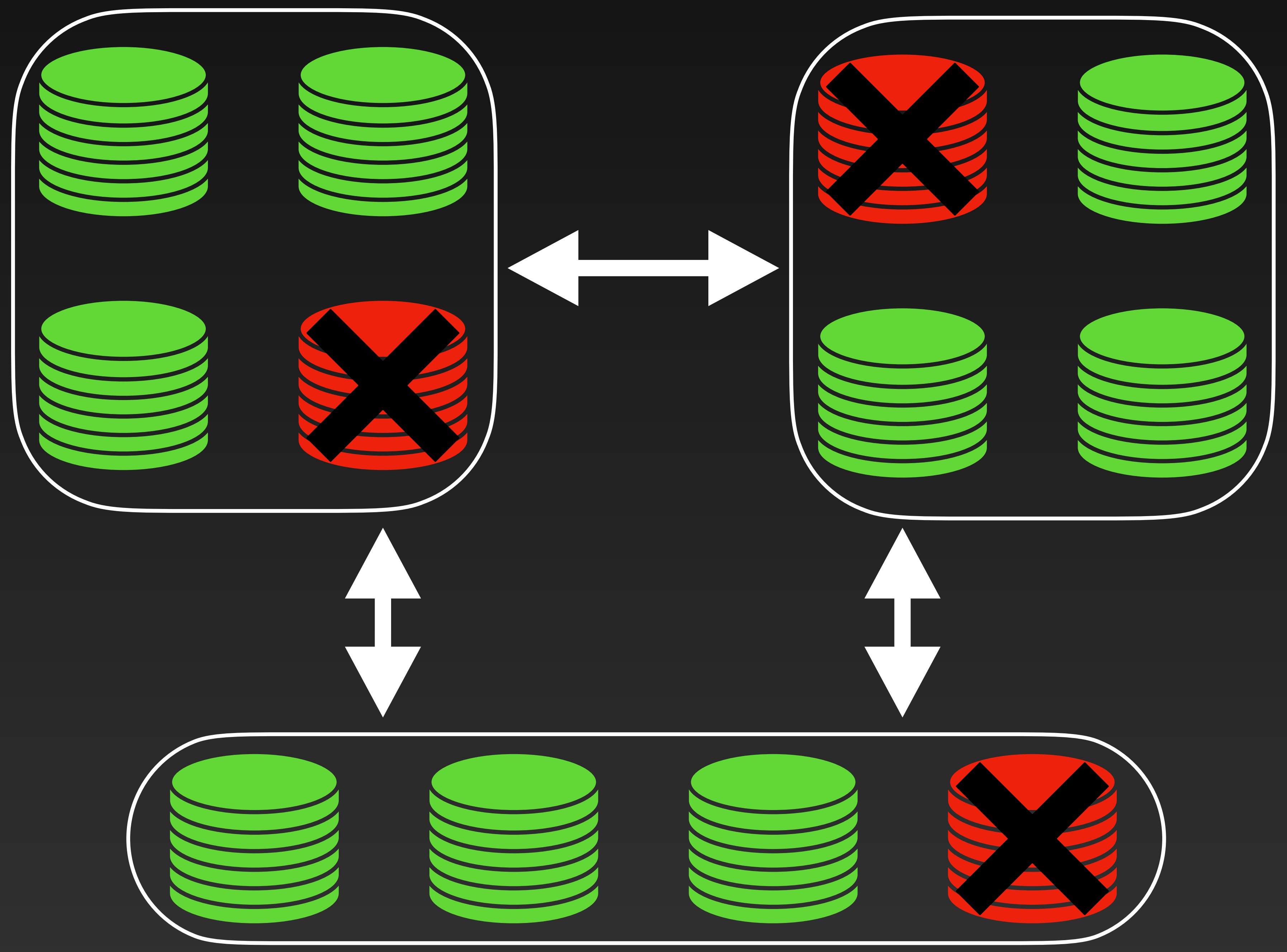
Reducing Latency

Using Side Infrastructures

Part II

What we have so far

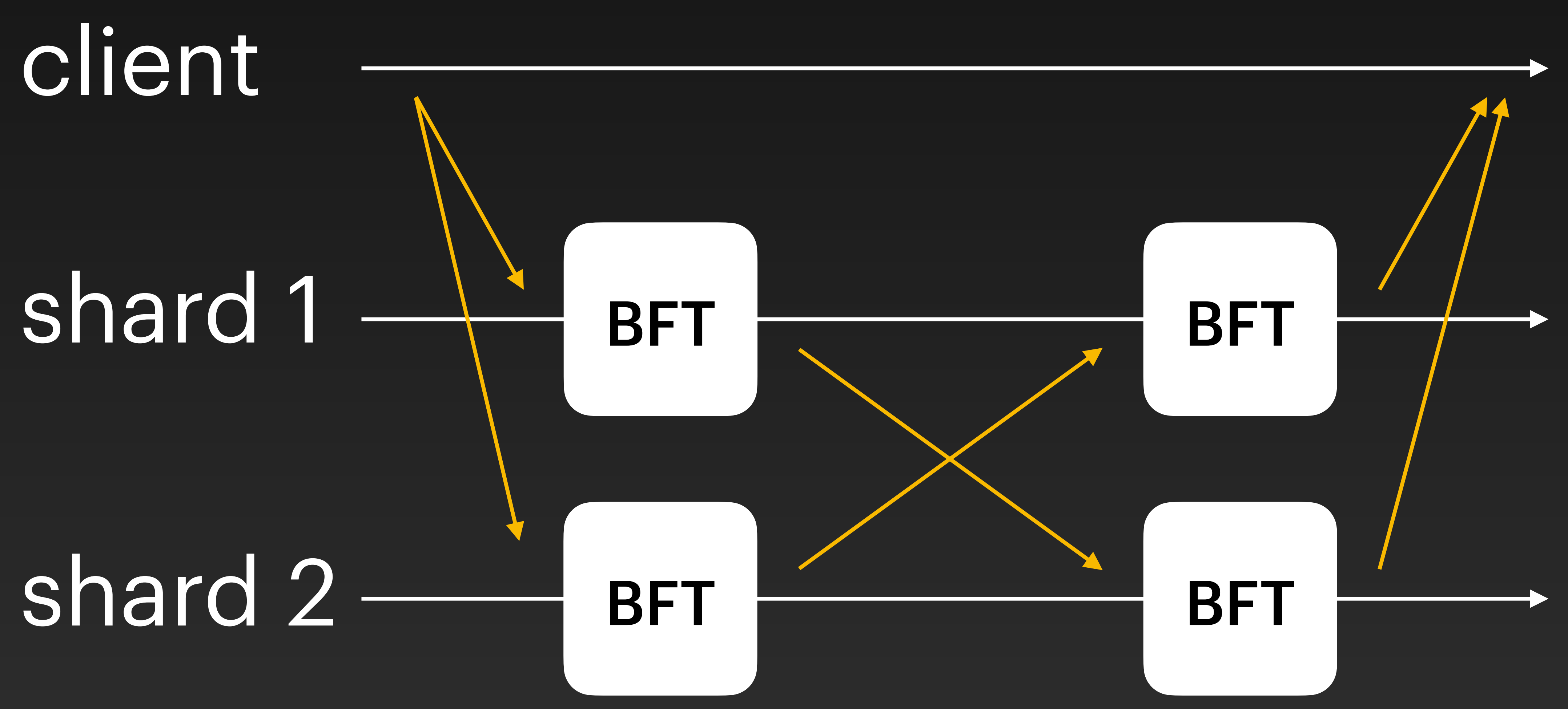
from Part I



- Hight throughput
- Linear scalability
- BFT resilience
- Fast finality

What we have so far

and what is missing



Total Latency:

slowest shard during phase 1

+

slowest shard during phase 2

+

all communications

Make it practical for retail payment at physical points of sale

That is the ambition

What do we need?

Properties

What we want

- Low latency
- BFT reliance
- Fast finality
- Hight capacity

Current industry

?

Centralized systems

Slow Finality

Little capacity

In summary

What we want

- Low latency
- BFT reliance
- Fast finality
- Hight capacity

Current industry

- Low latency
- Centralized
- Slow finality
- Little capacity

FastPay

High-Performance Byzantine Fault Tolerant Settlement

FastPay
Acknowledgments

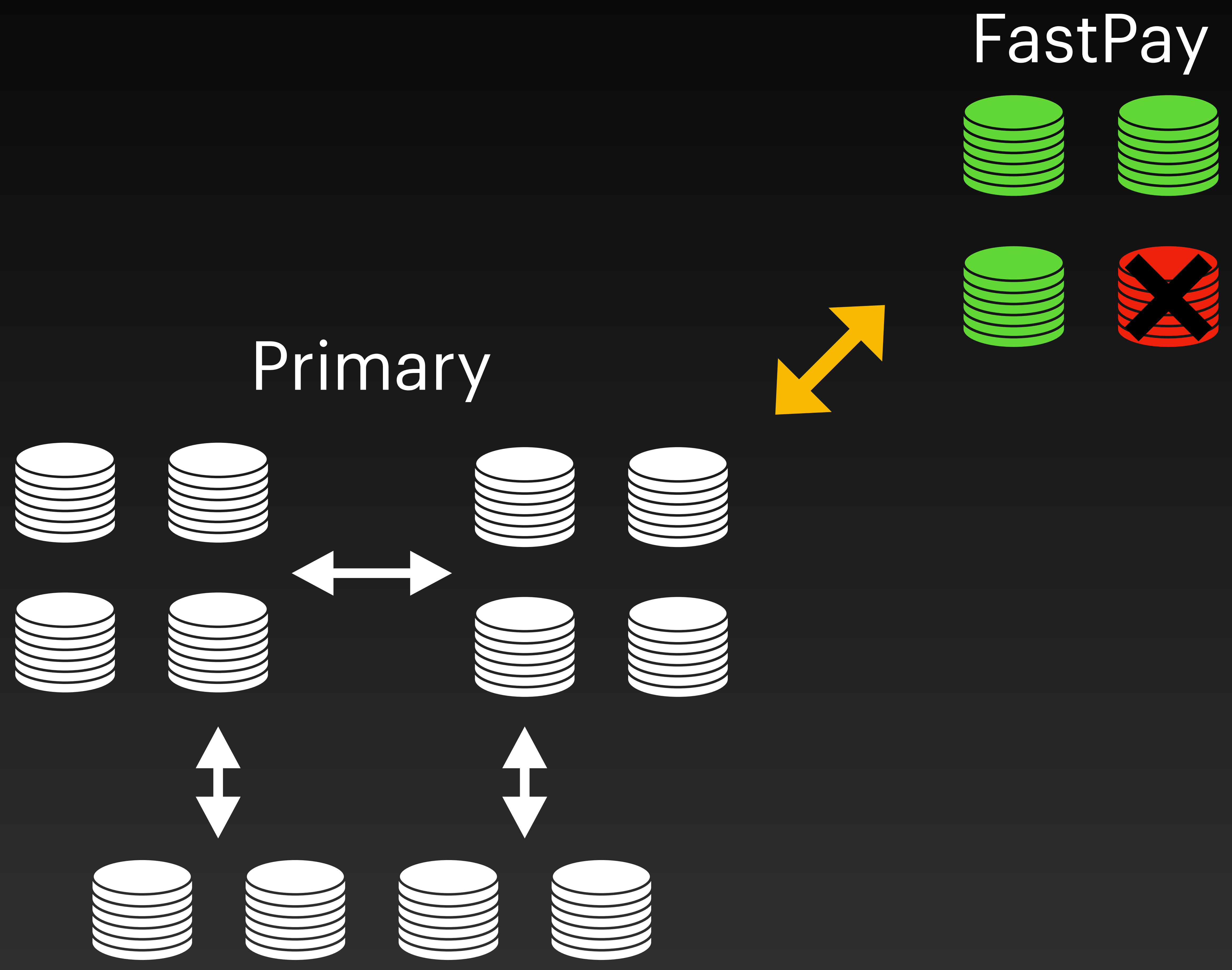
Mathieu Baudet

George Danezis

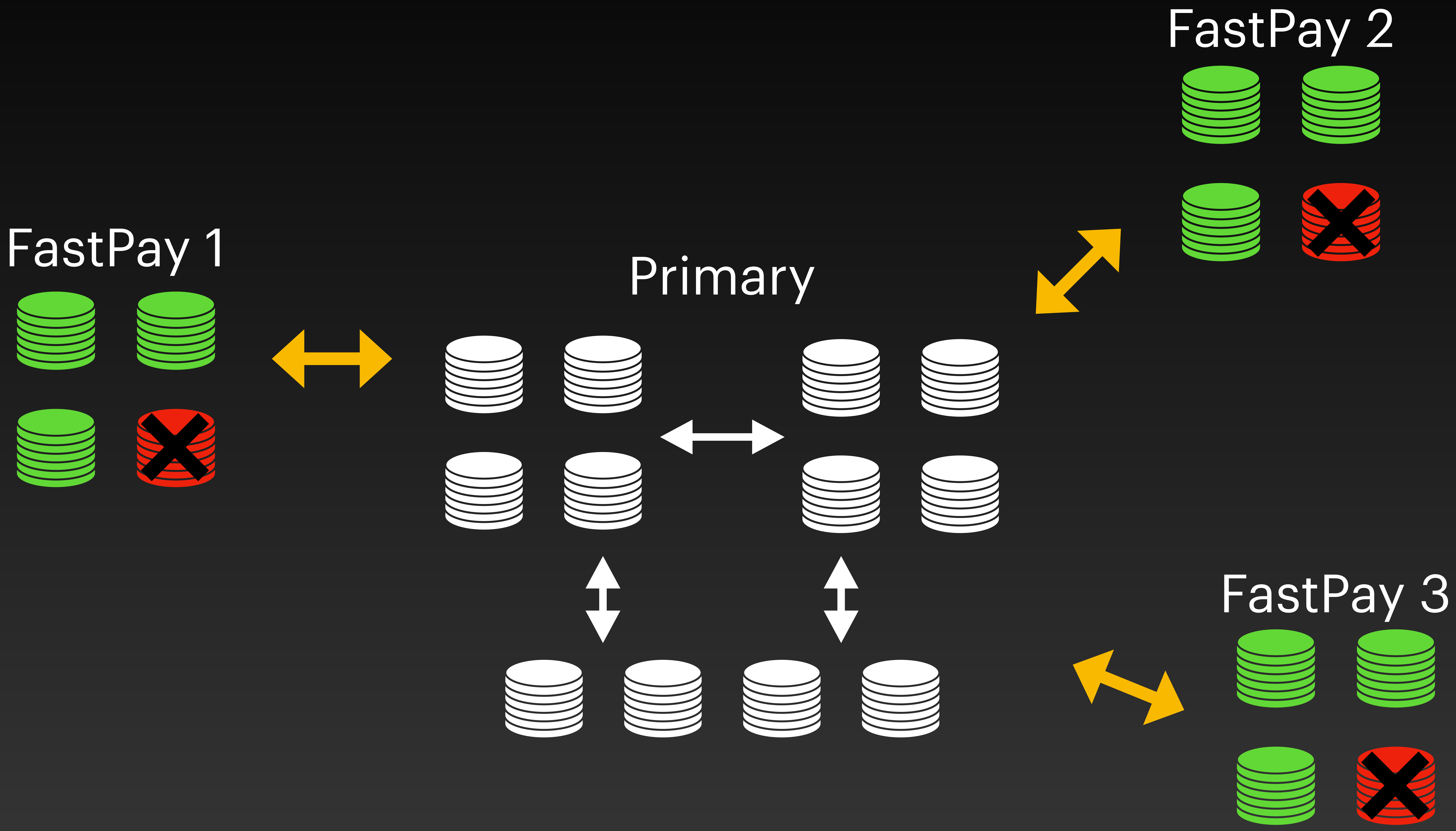
Alberto Sonnino

Facebook Novi

Overview

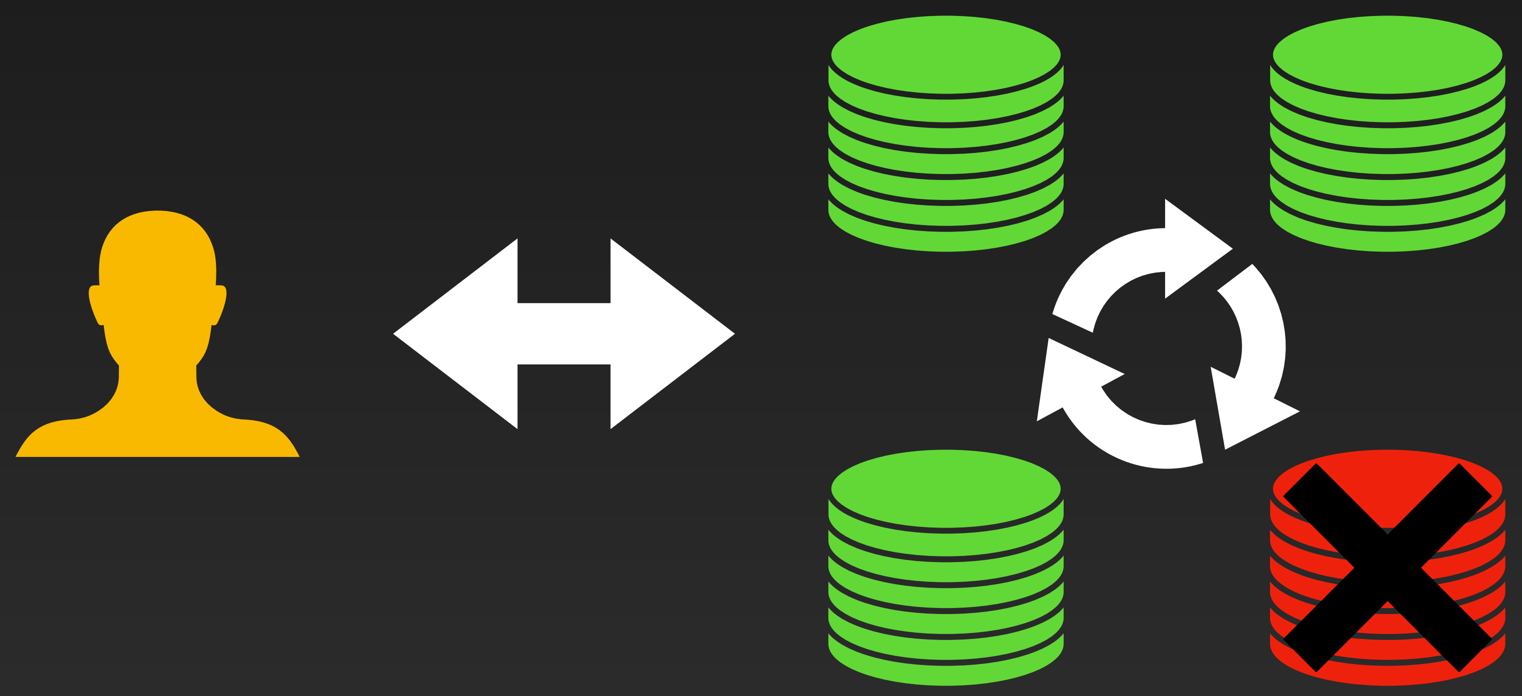


Overview



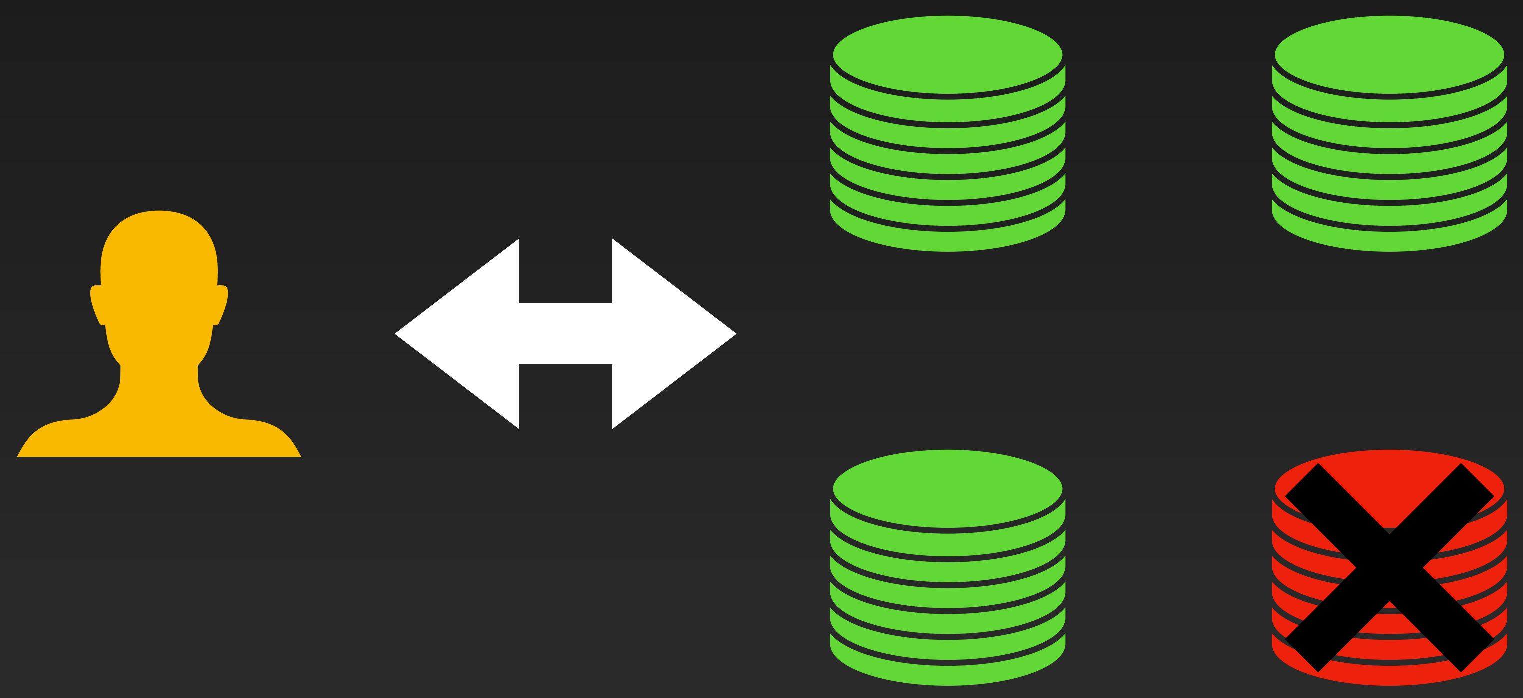
Difference with blockchains

Blockchains



Byzantine Consensus

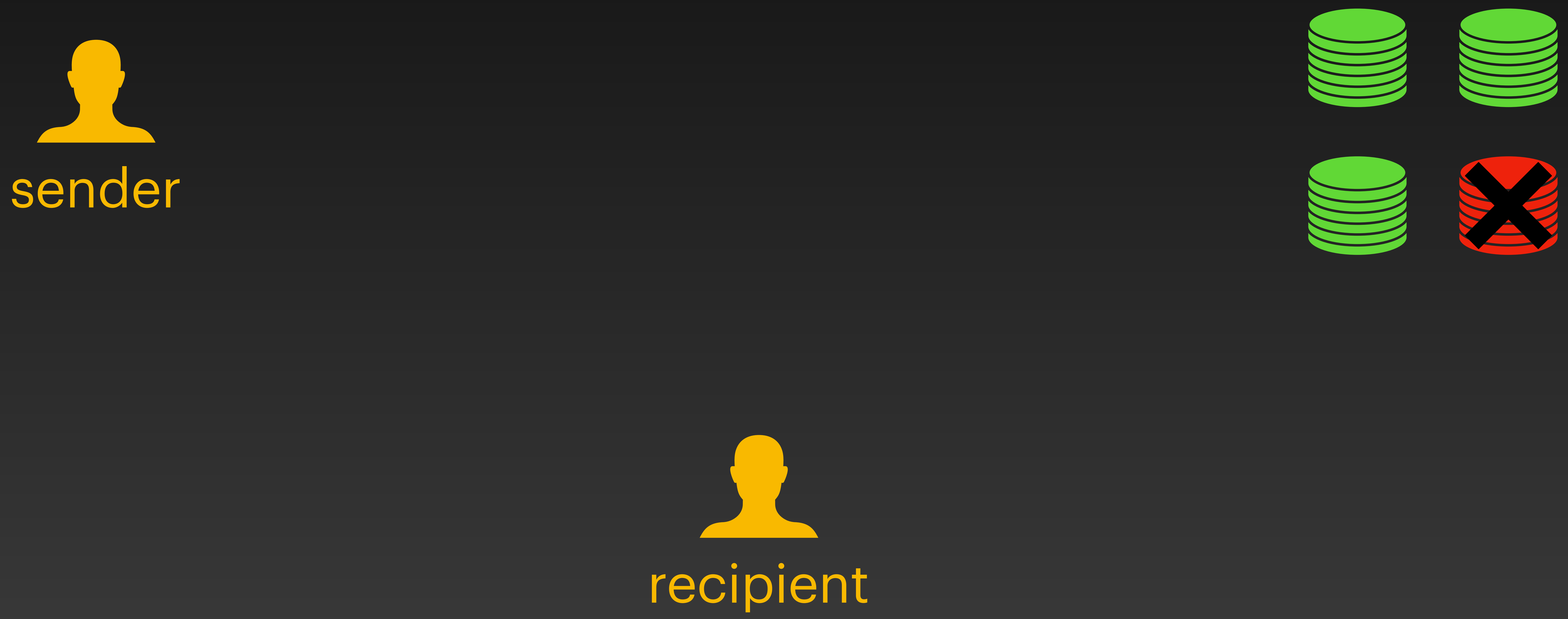
FastPay



Byzantine Consistent Broadcast

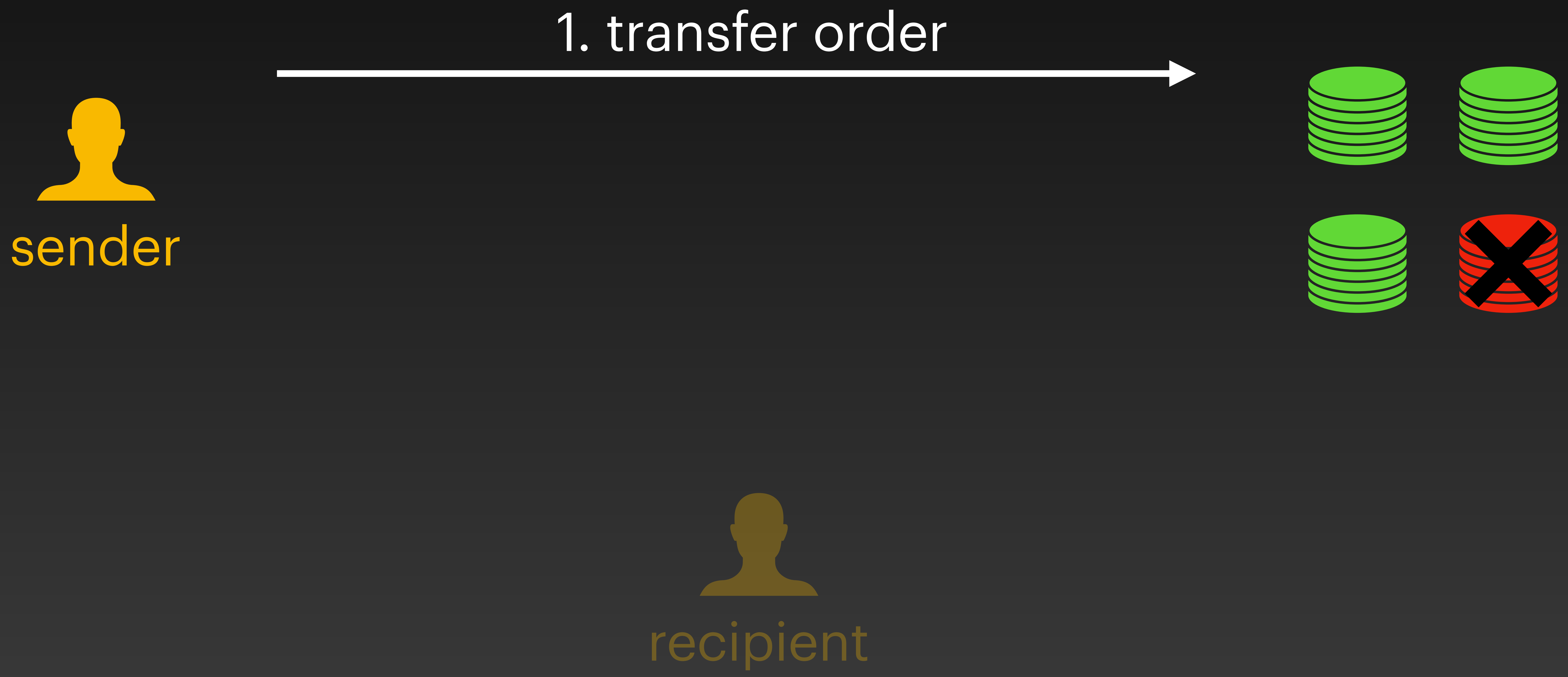
FastPay

How does it work?



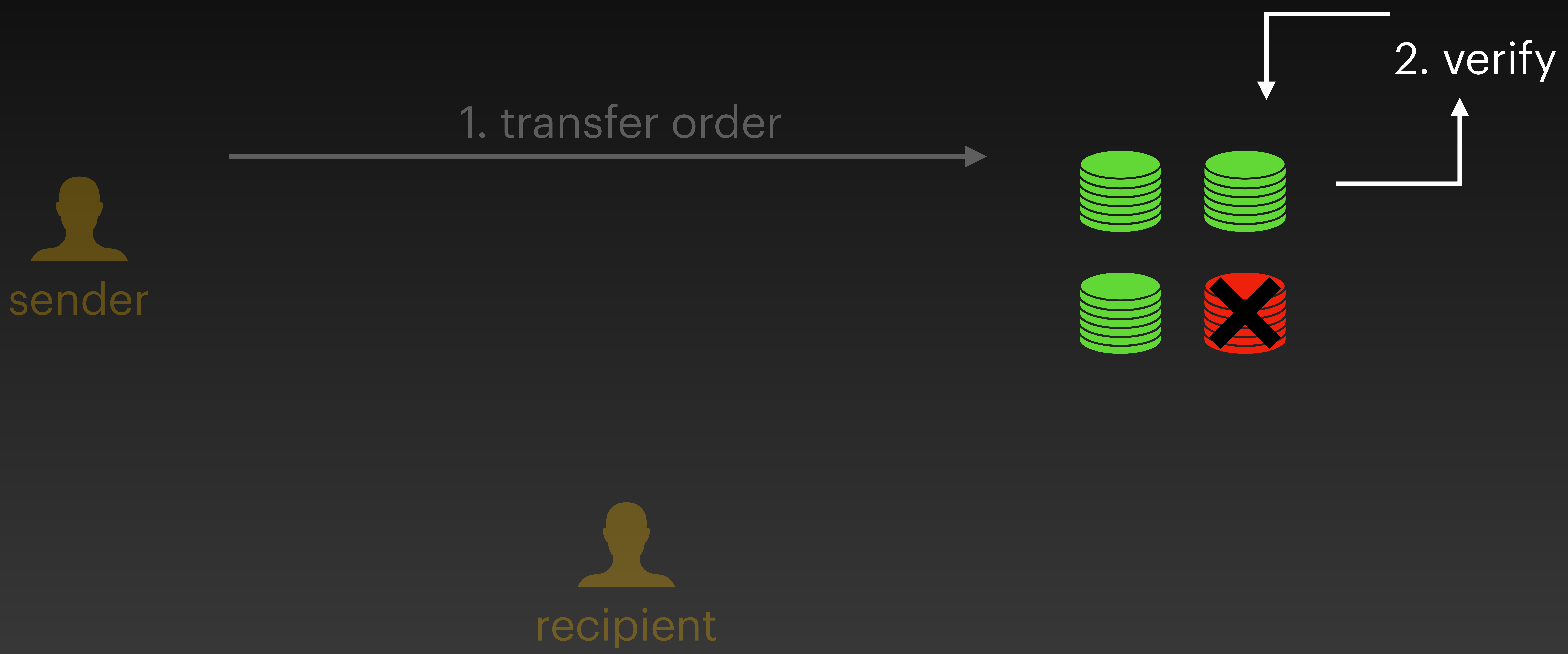
FastPay

How does it work?

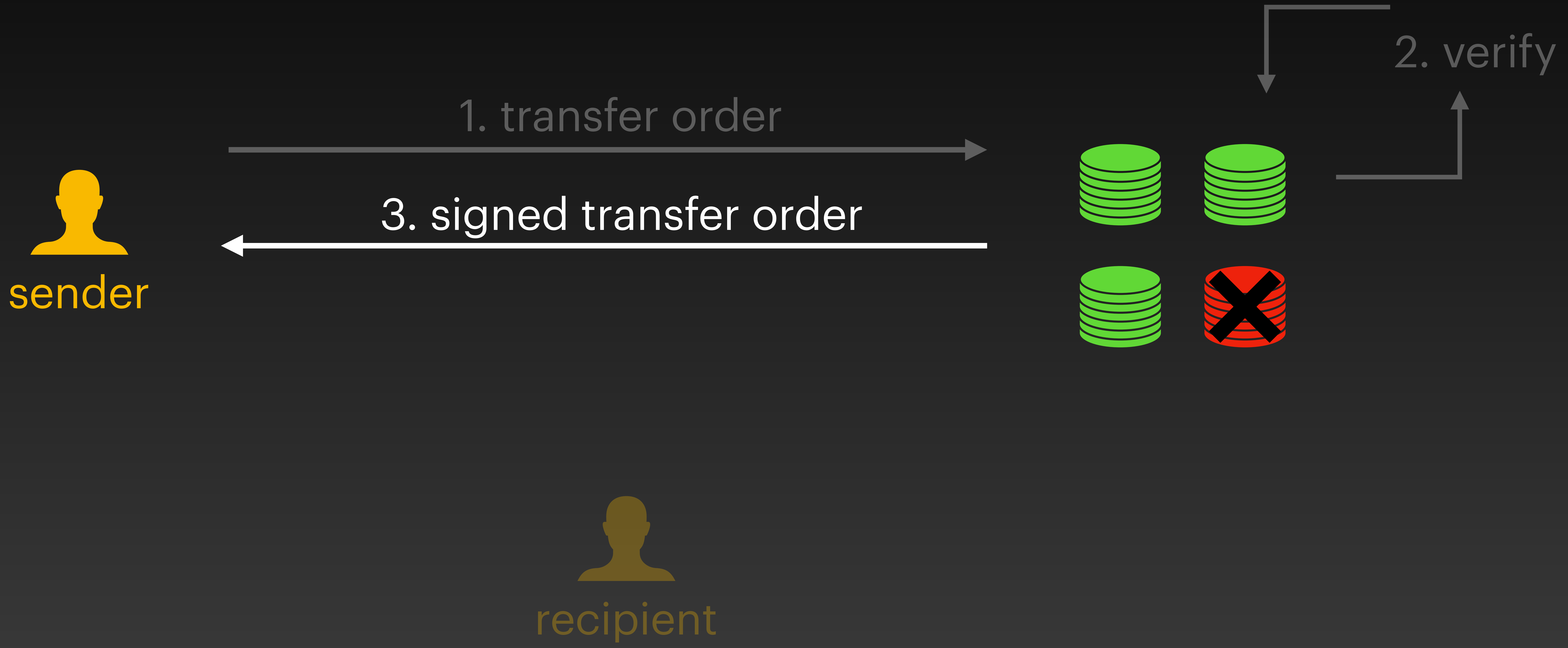


FastPay

How does it work?

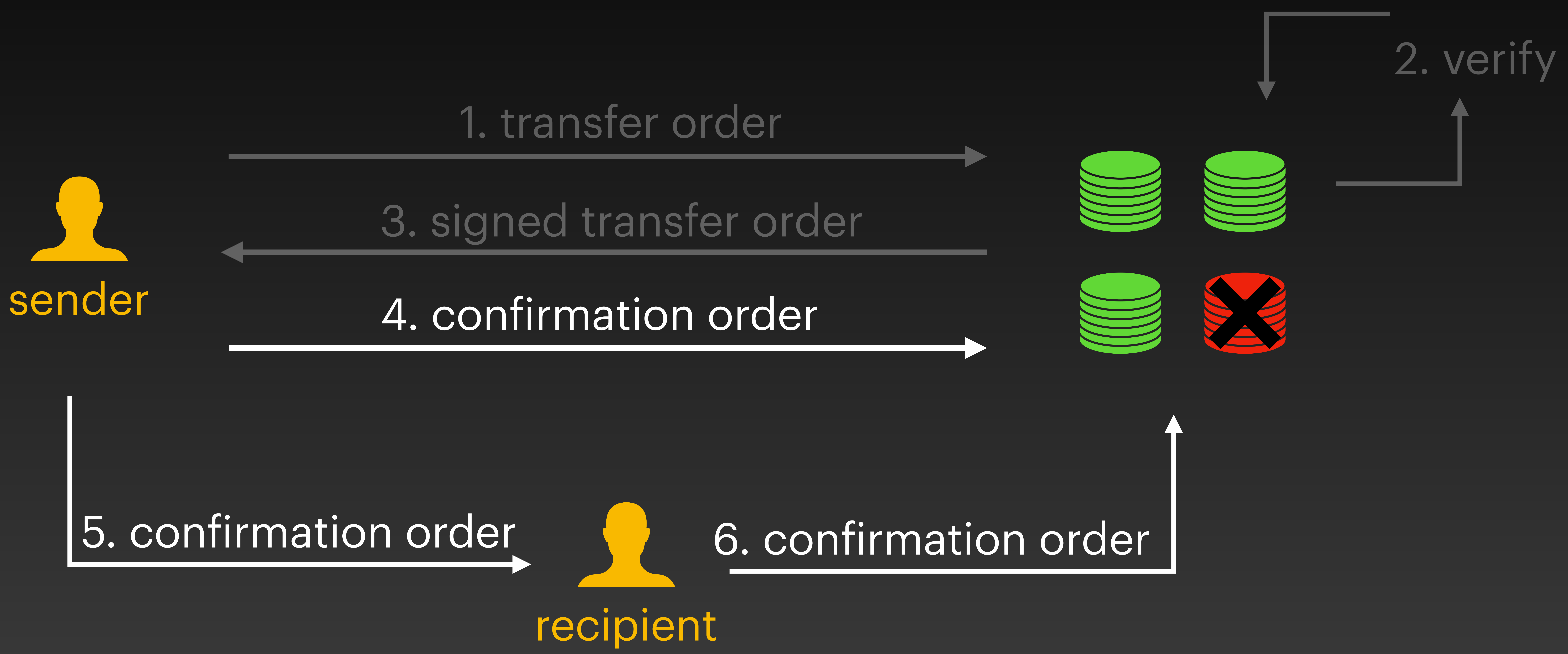


FastPay
How does it work?



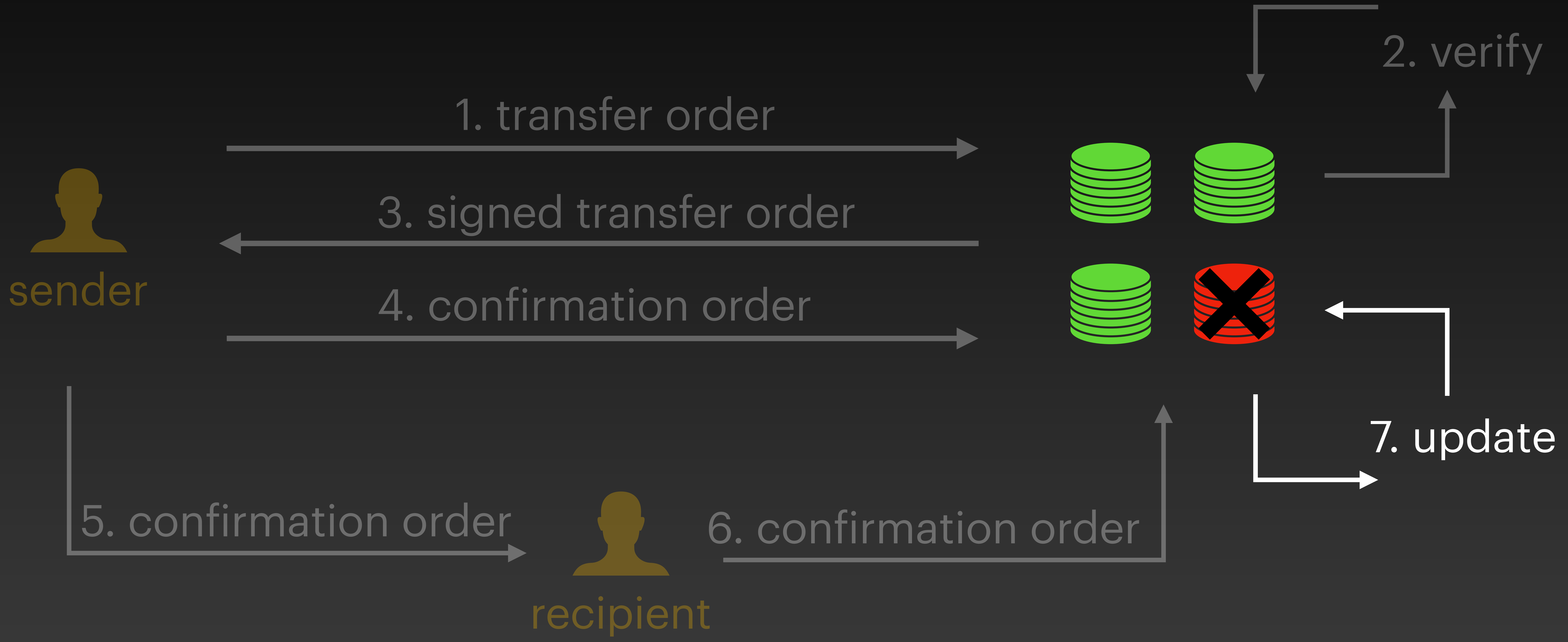
FastPay

How does it work?



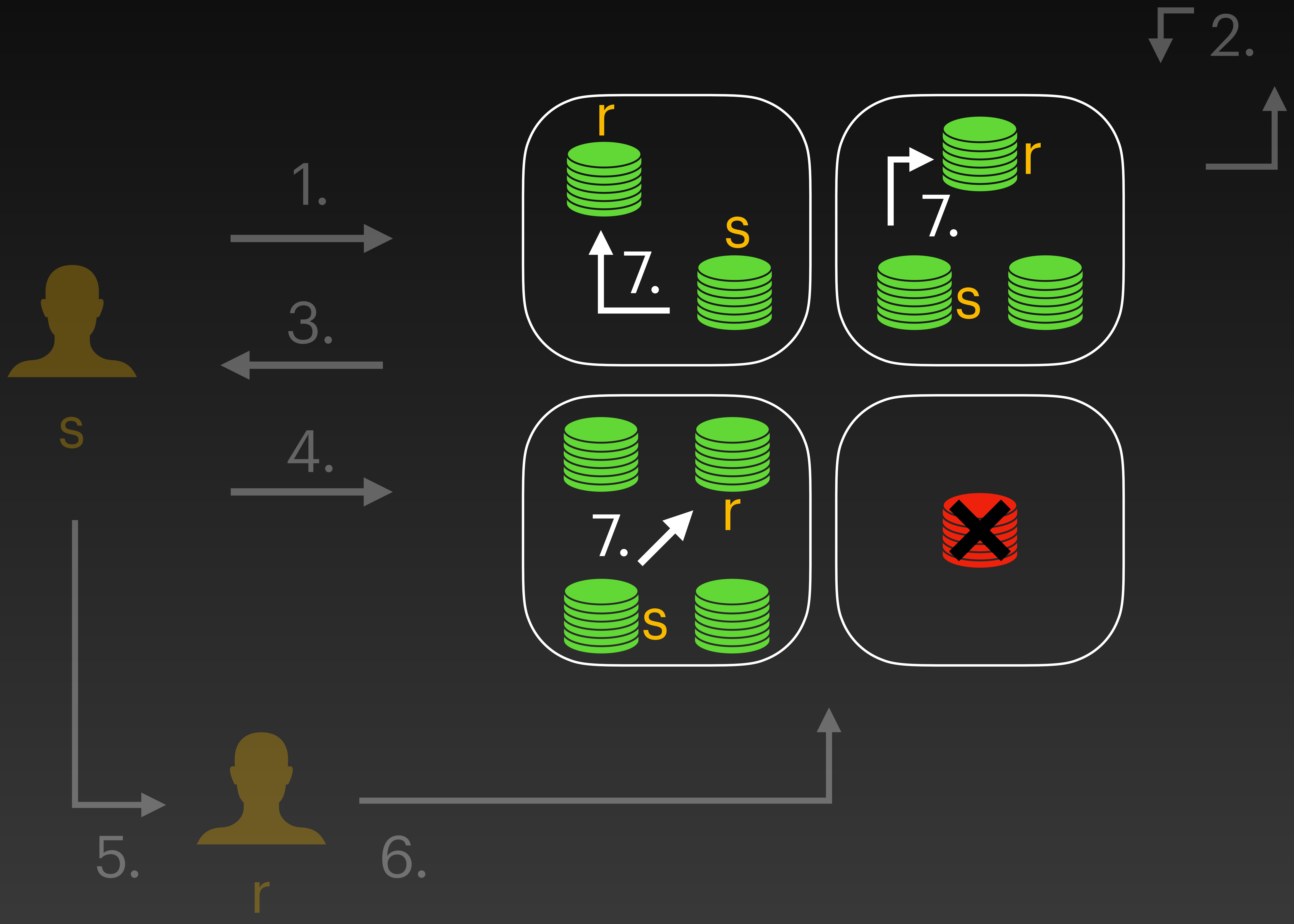
FastPay

How does it work?



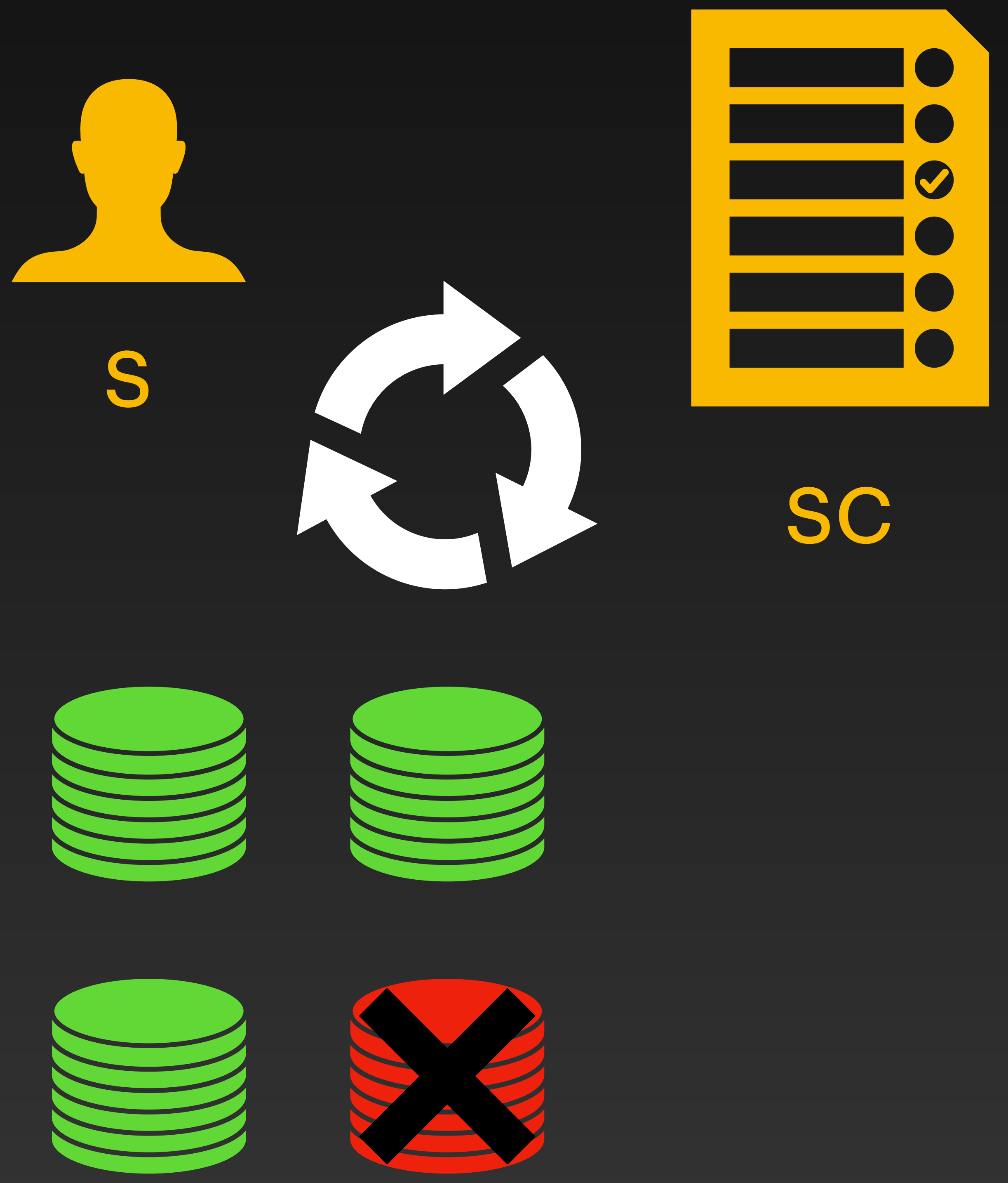
FastPay

Increasing capacity



FastPay

Interface it with a primary infrastructure

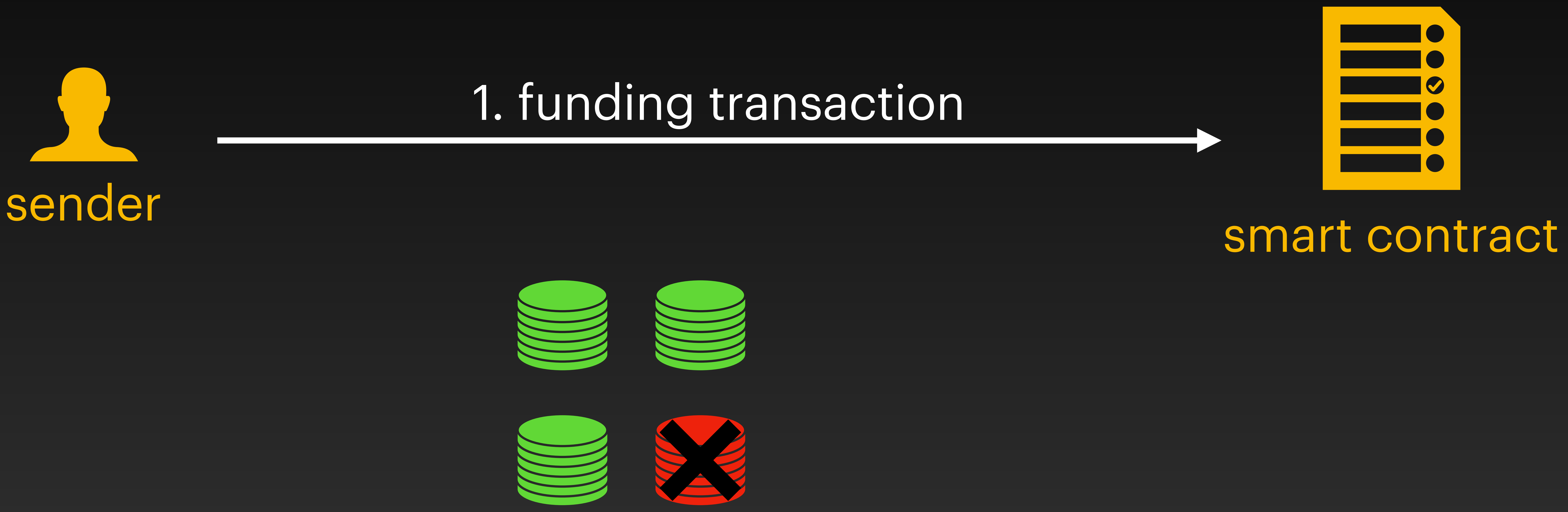


Smart Contract's state

- The committee information
- Total funds in the contract
- Last primary tx index
- "Redeem log"

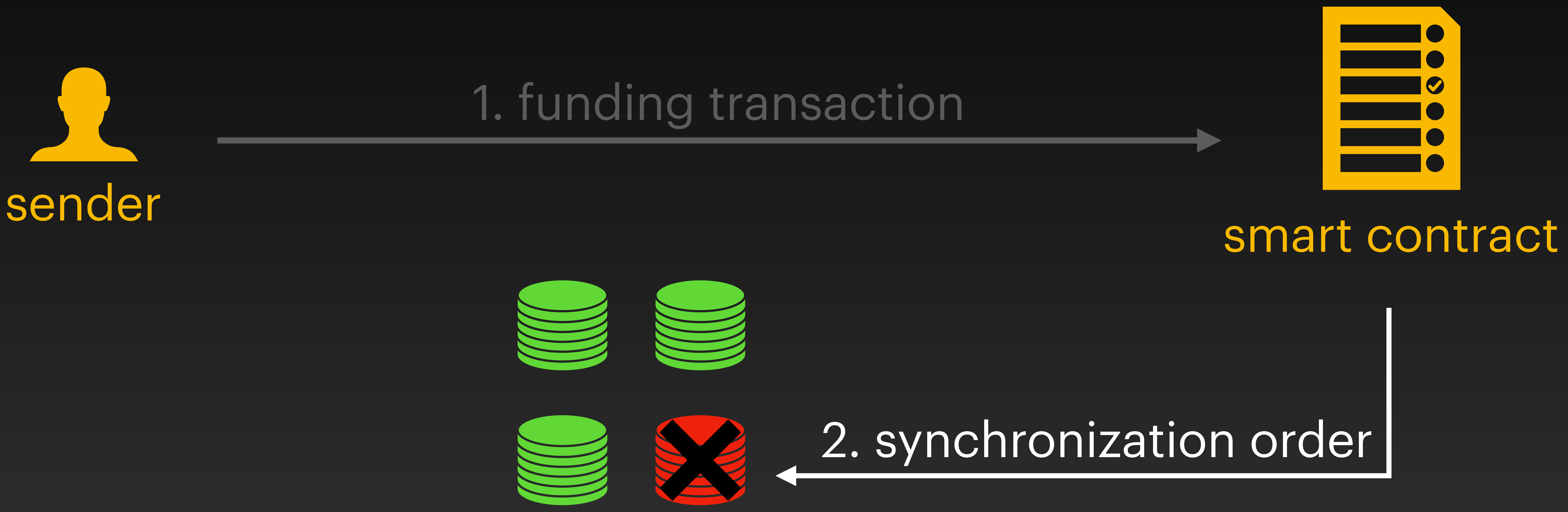
FastPay

From primary infrastructure to FastPay



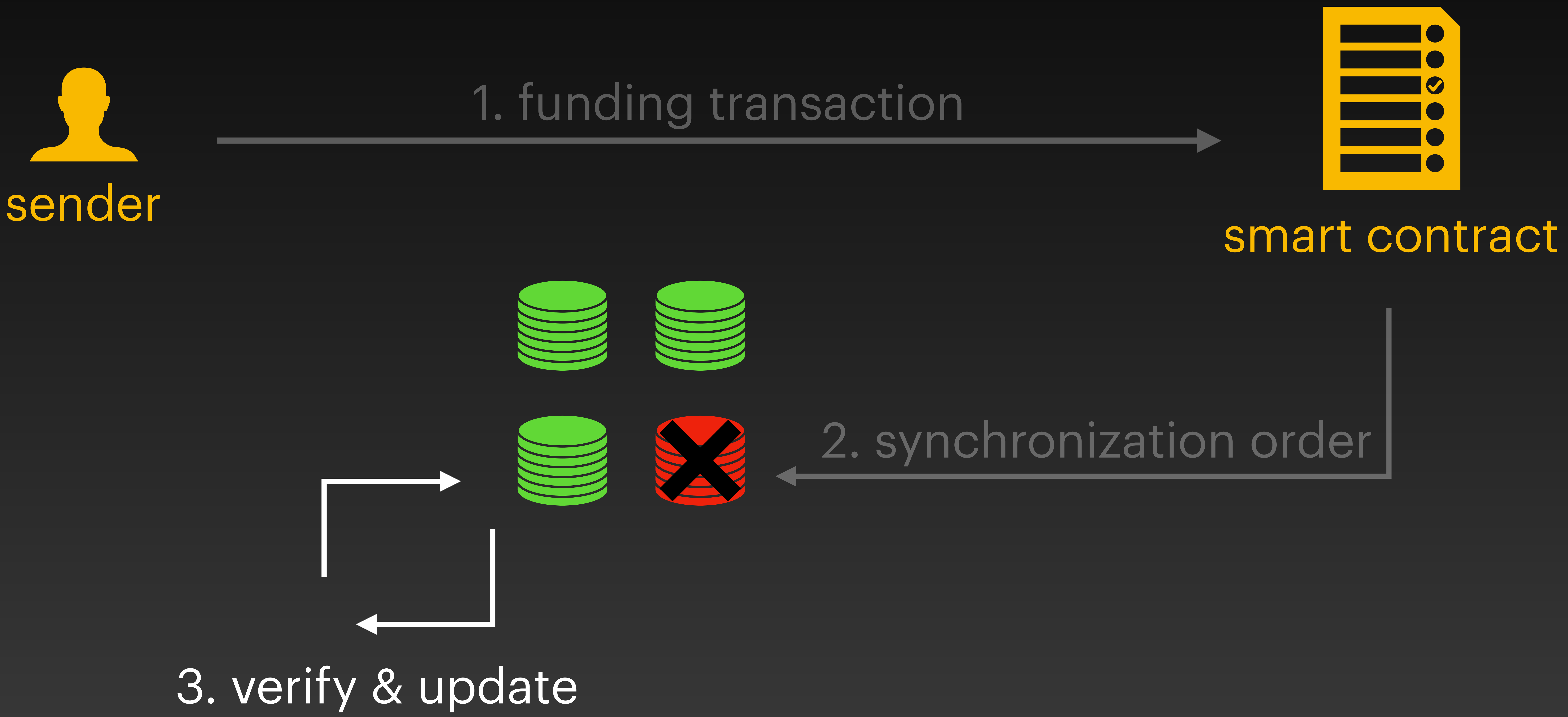
FastPay

From primary infrastructure to FastPay



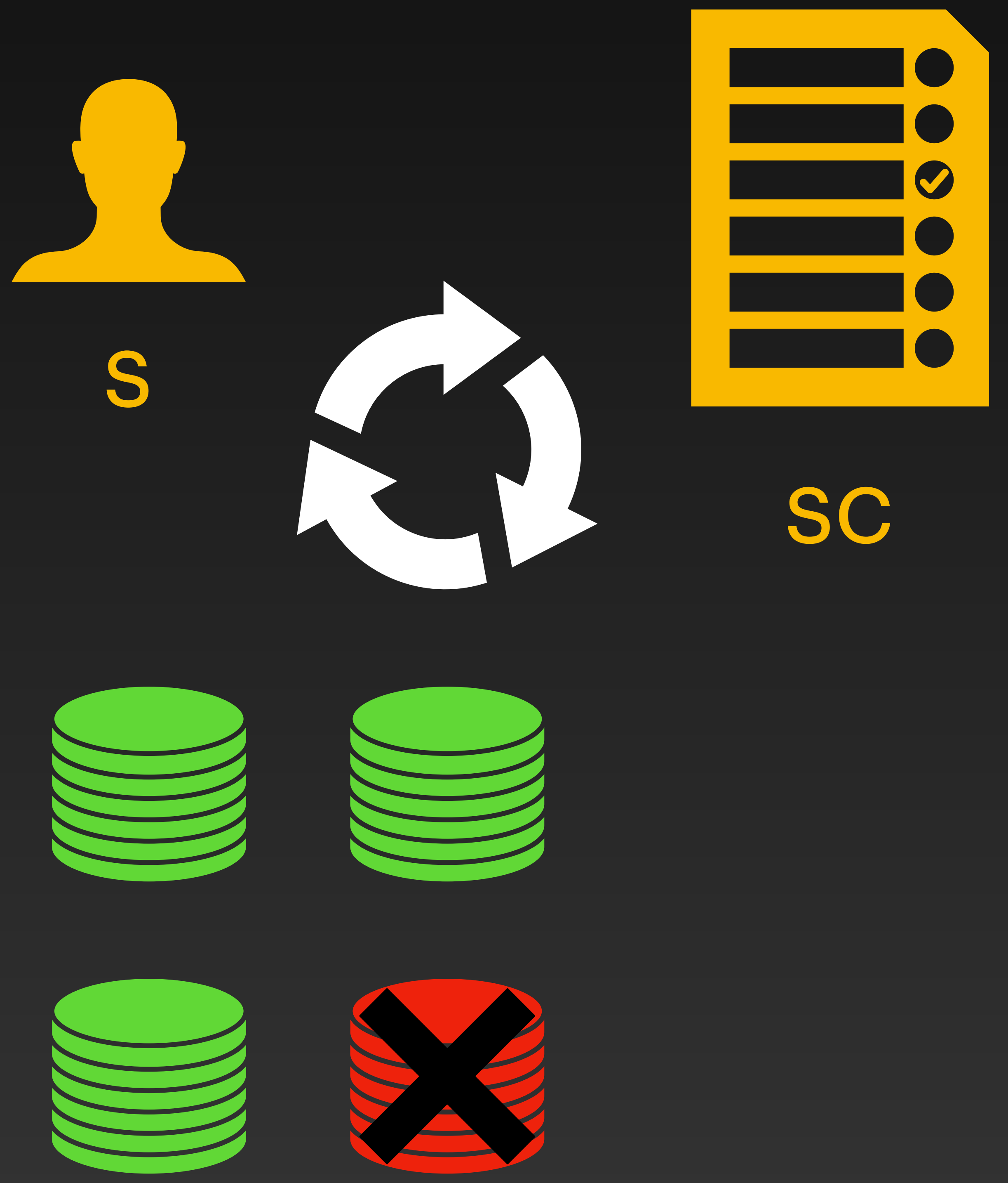
FastPay

From primary infrastructure to FastPay



FastPay

Interface it with a primary infrastructure

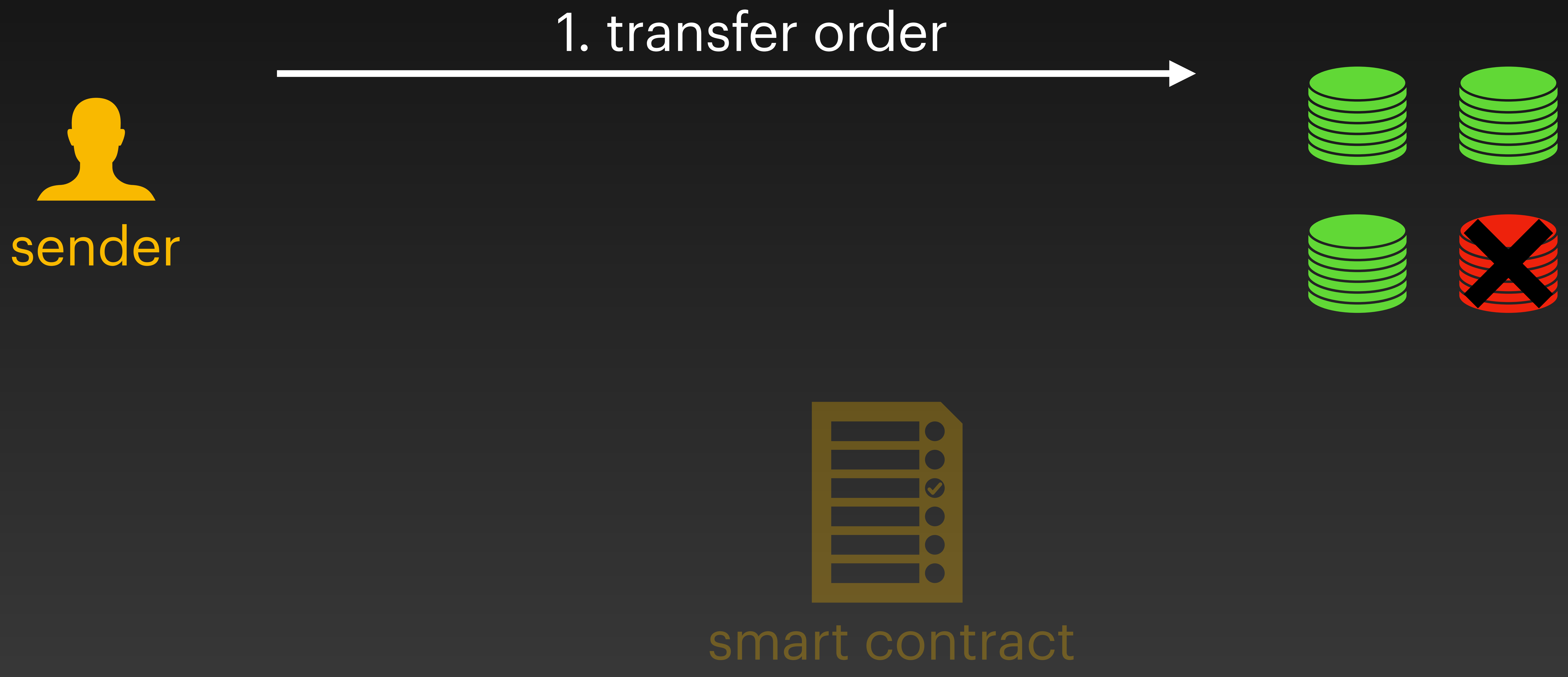


Smart Contract's state

- The committee information
- Total funds in the contract
- Last primary tx index
- "Redeem log"

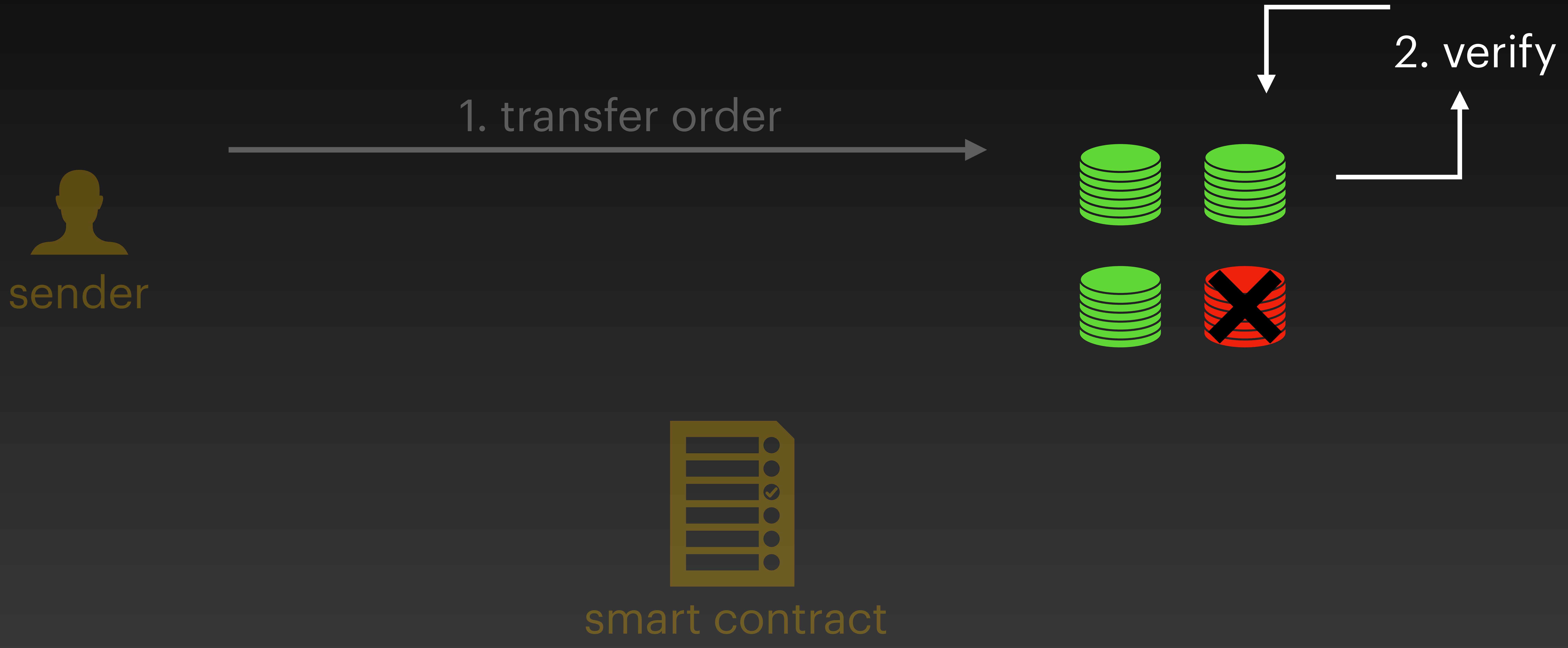
FastPay

From the primary infrastructure to FastPay



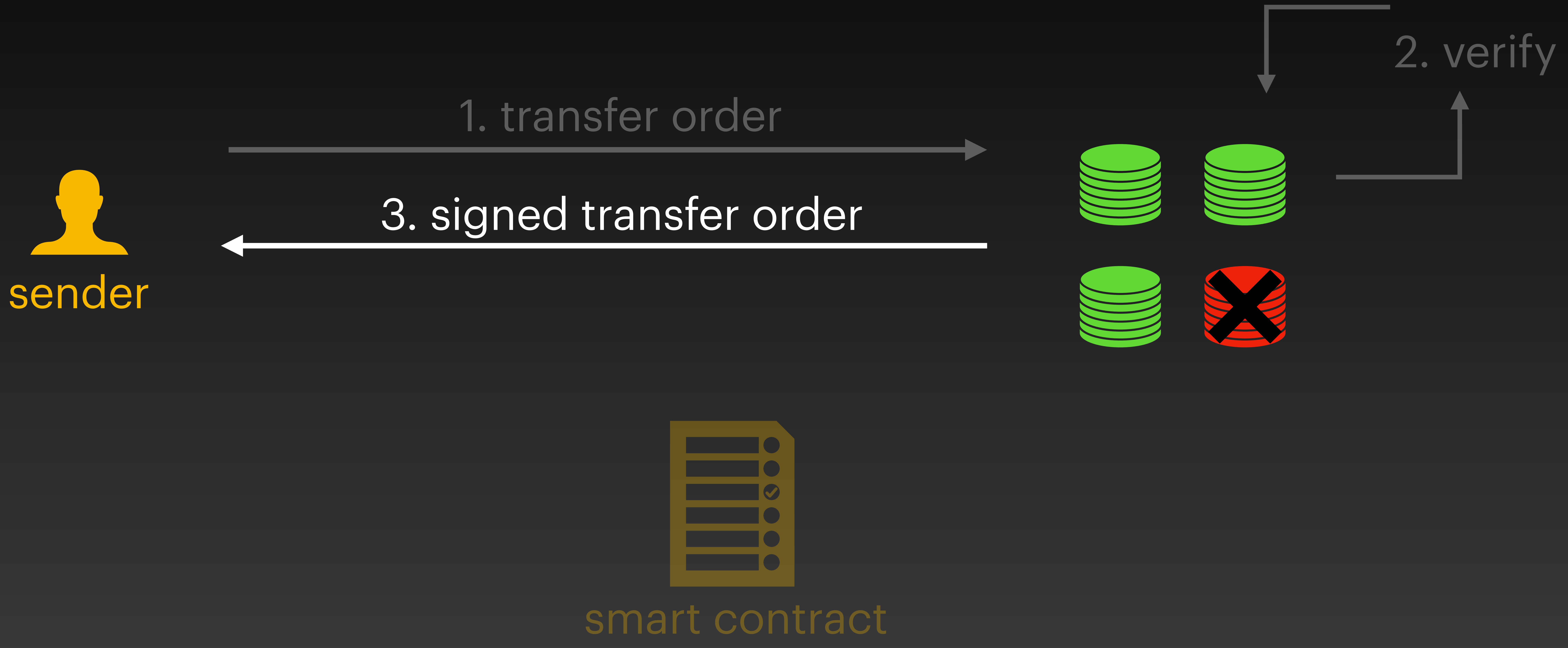
FastPay

From the primary infrastructure to FastPay



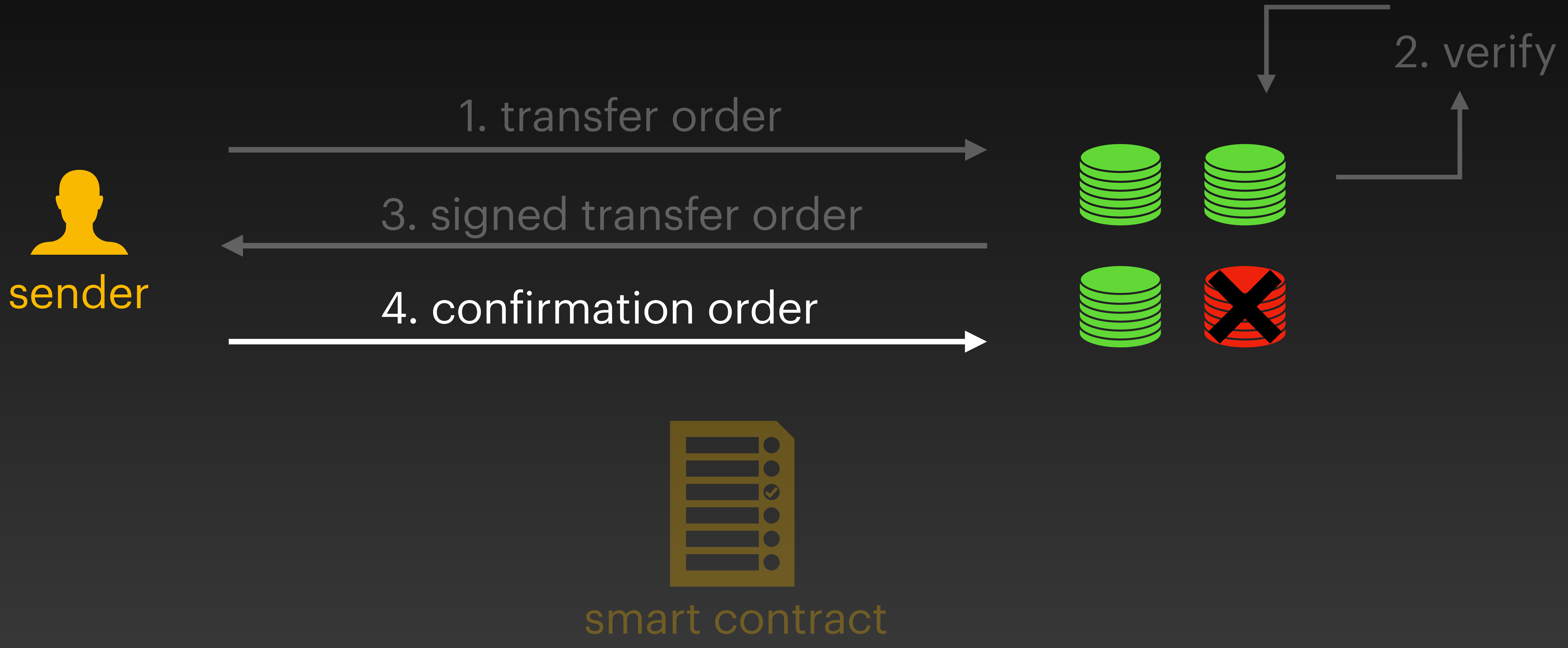
FastPay

From the primary infrastructure to FastPay



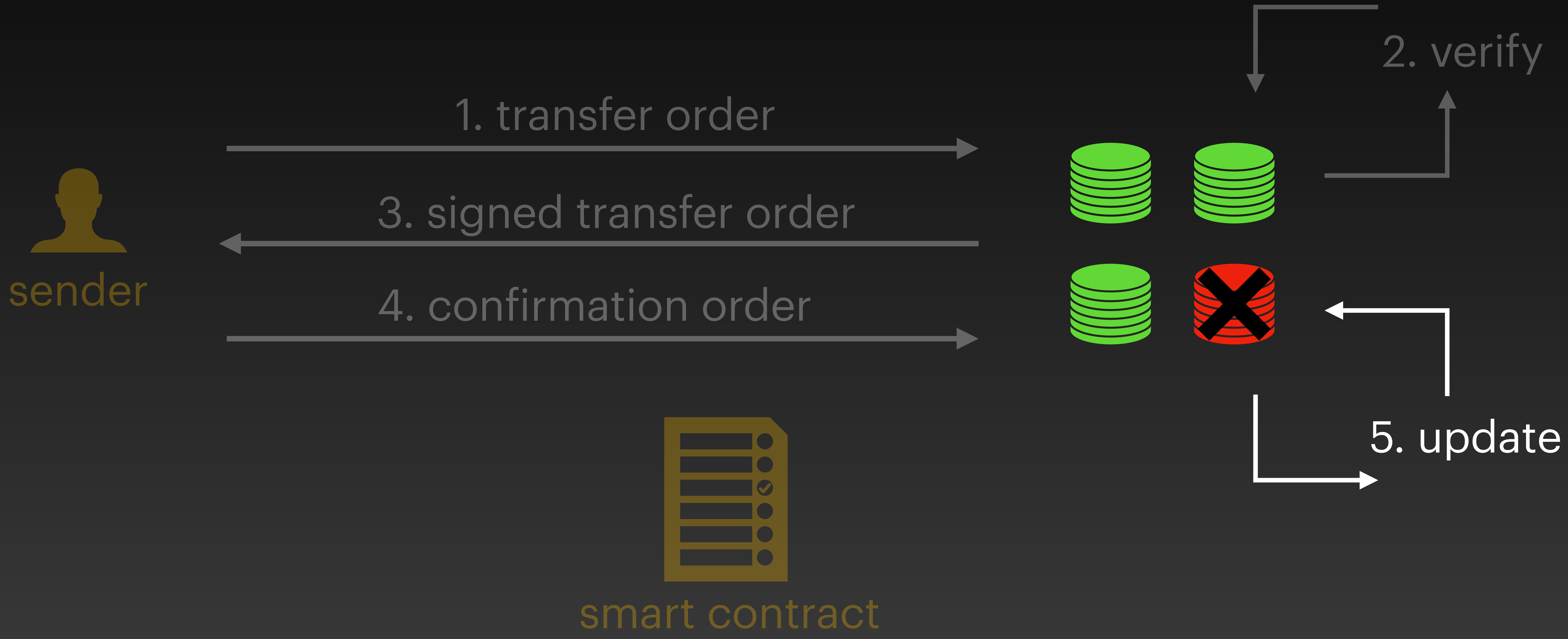
FastPay

From the primary infrastructure to FastPay



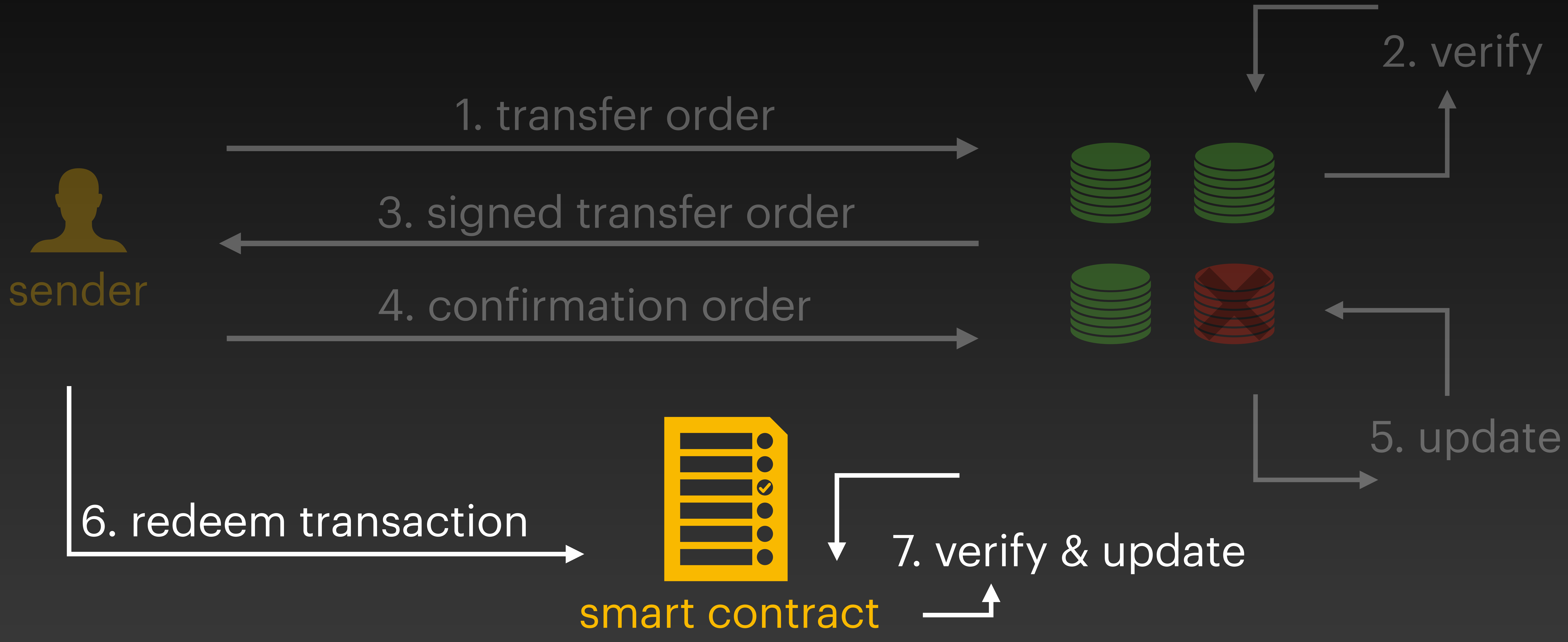
FastPay

From the primary infrastructure to FastPay



FastPay

From the primary infrastructure to FastPay



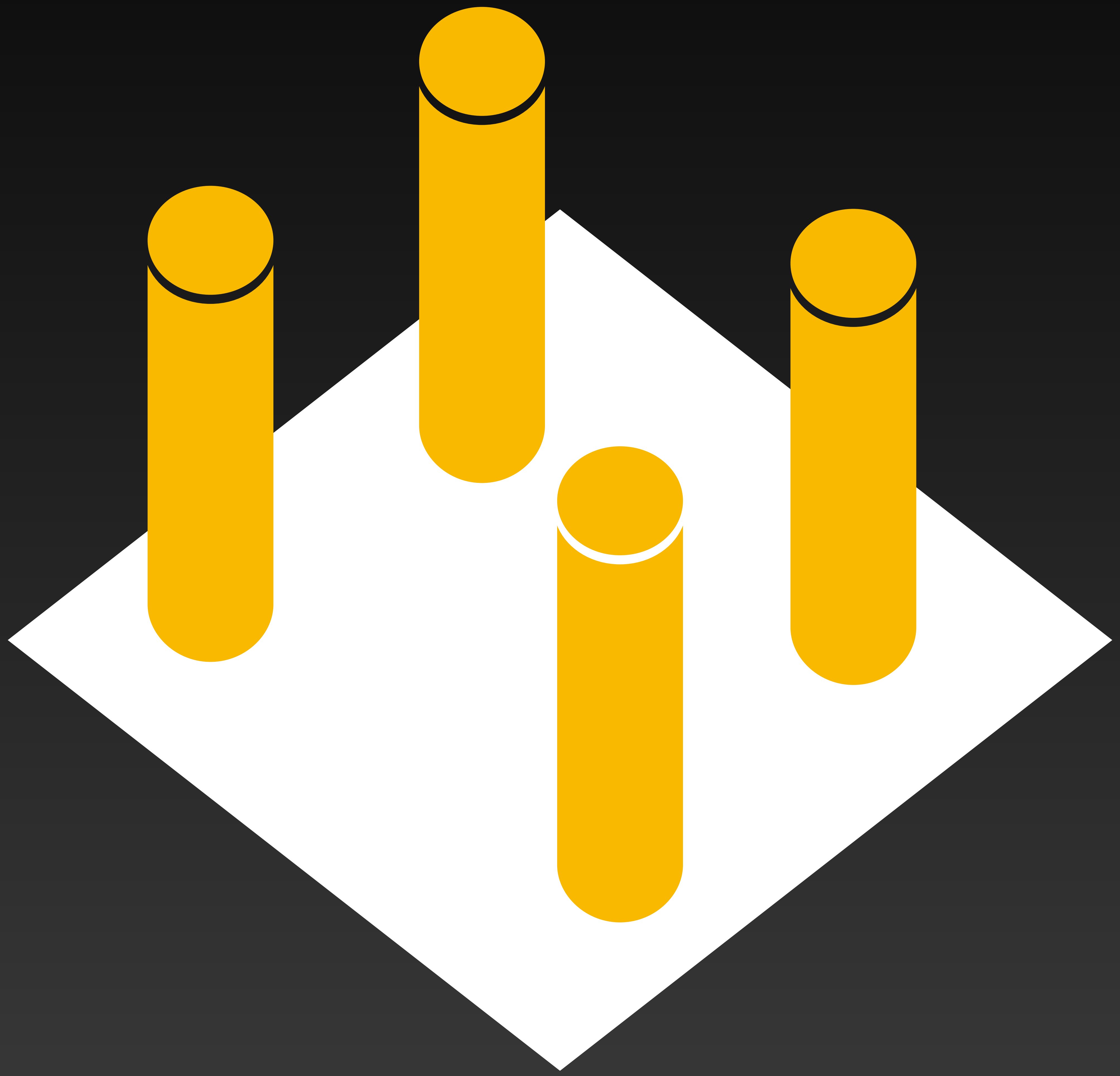
FastPay

Implementation

- Written in Rust
- Networking: Tokio & UDP
- Cryptography: ed25519-dalek

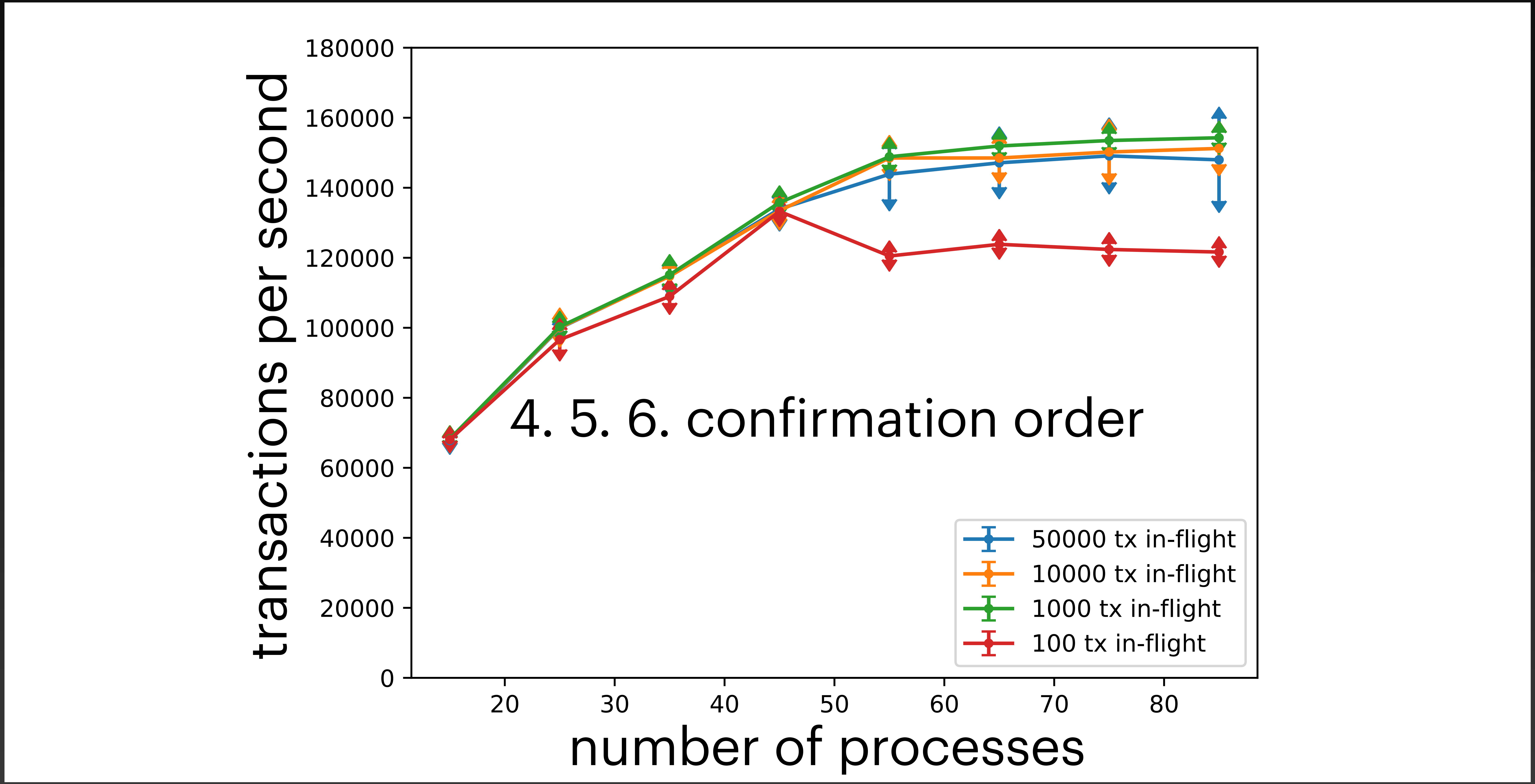
https://github.com/calibra/fastpay

FastPay
Throughput Evaluation



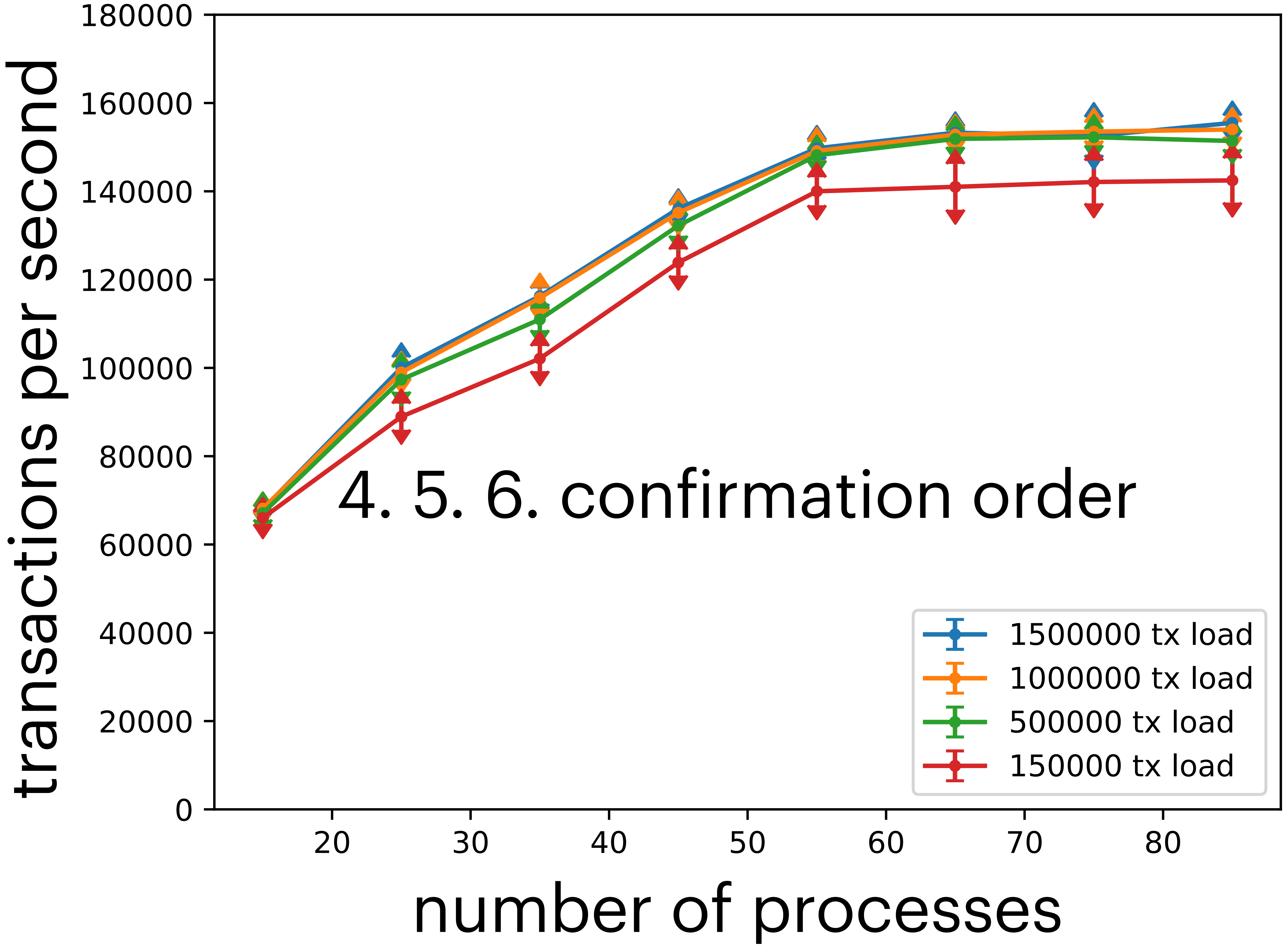
FastPay

High concurrency



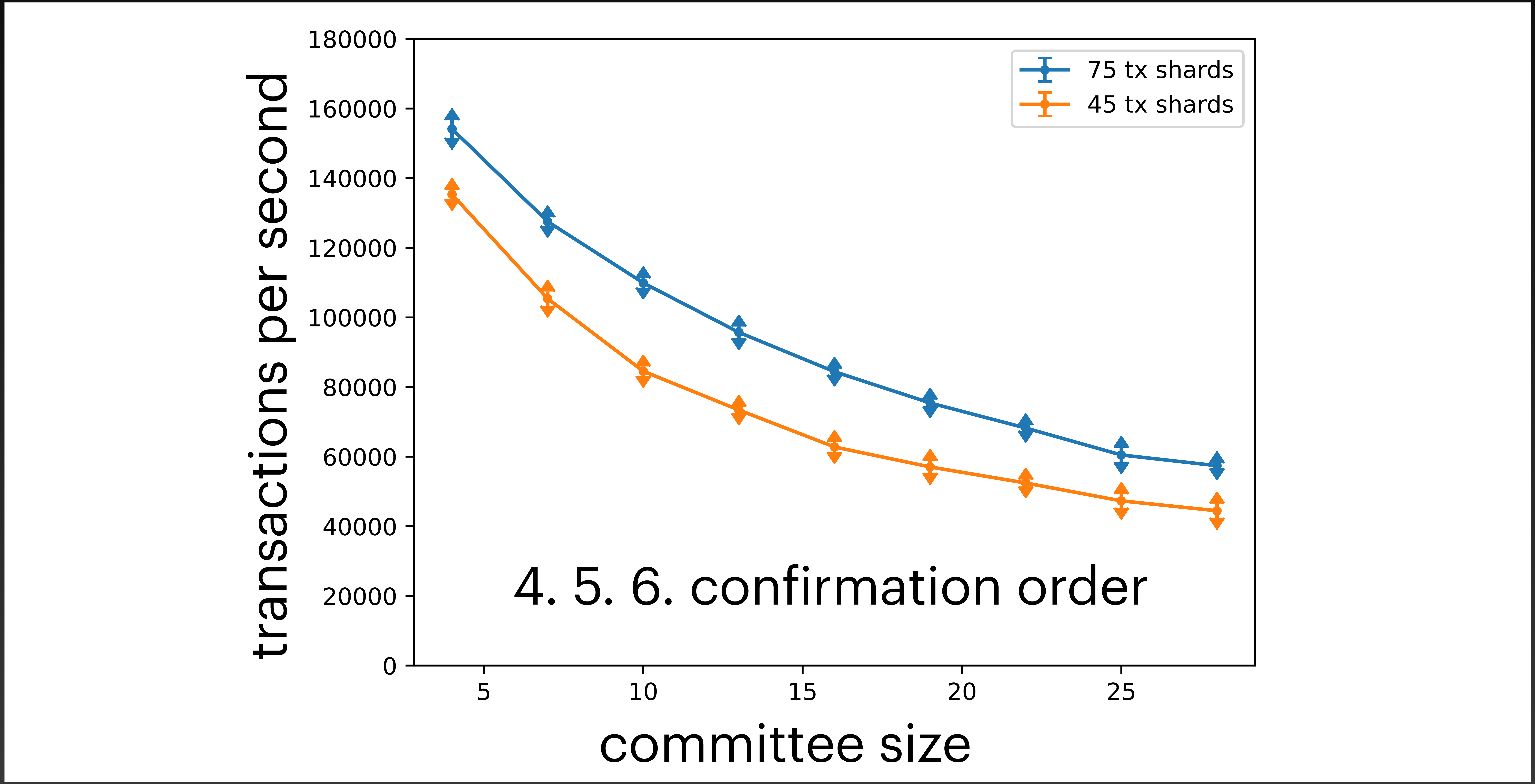
FastPay

Robustness

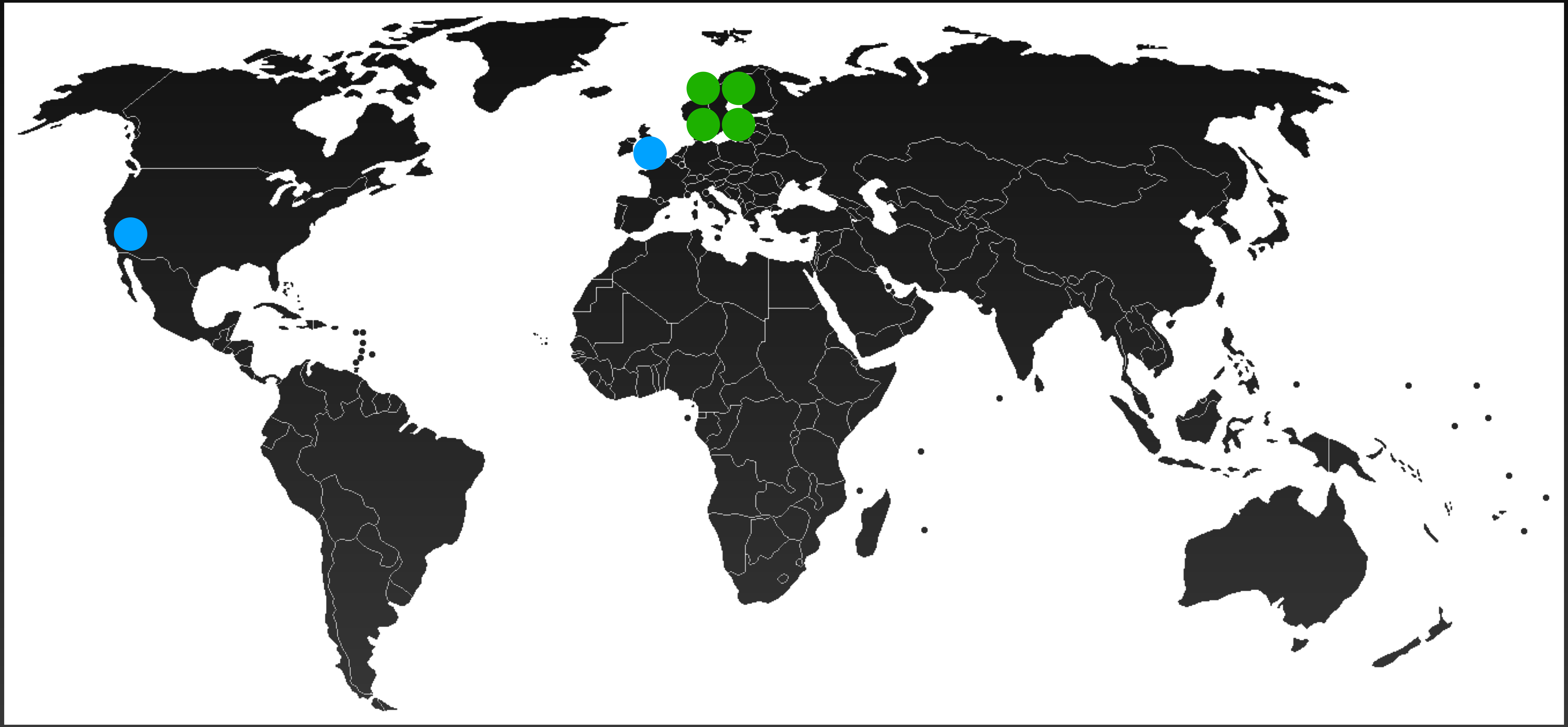


FastPay

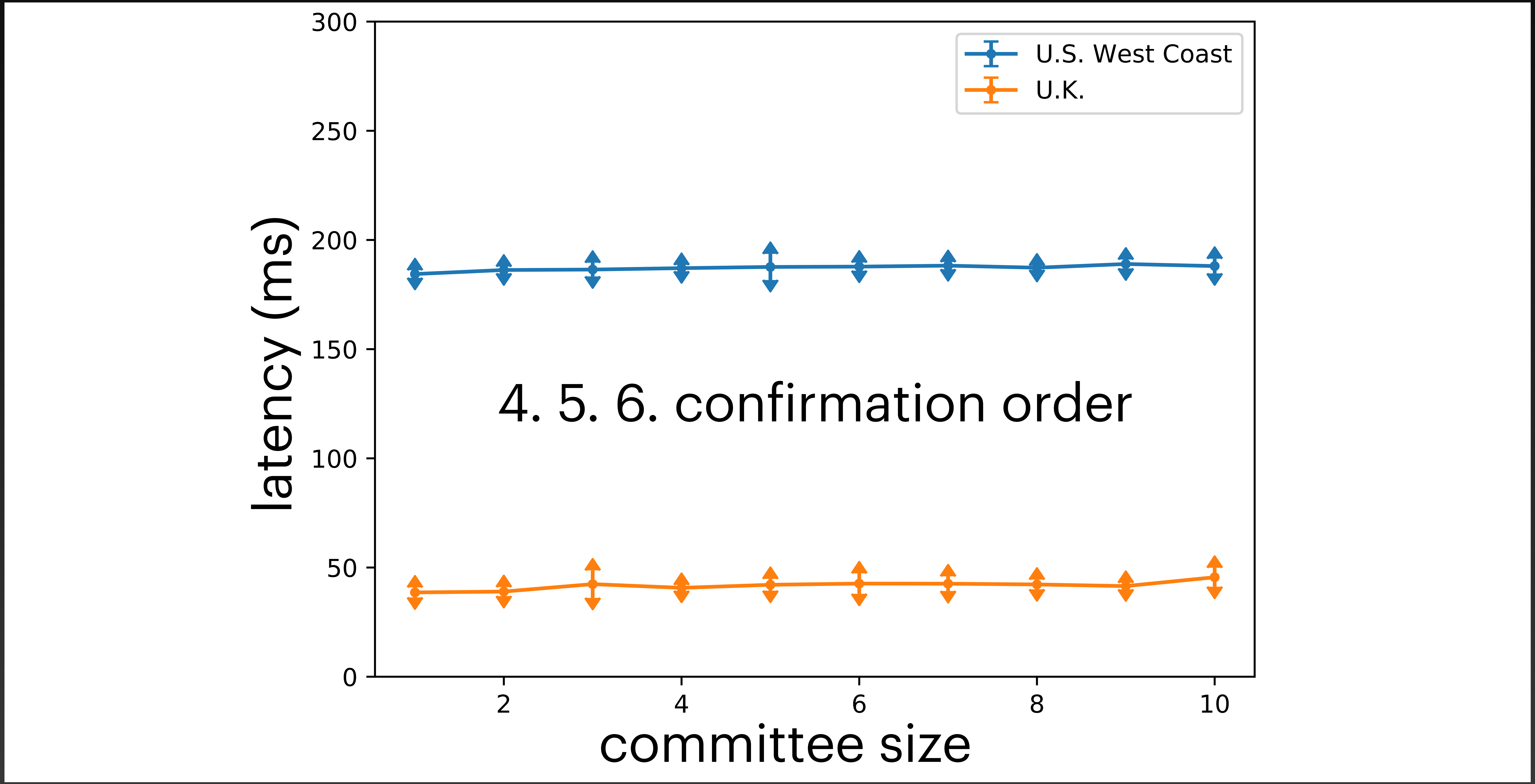
Influence of the number of authorities



FastPay
Latency setup

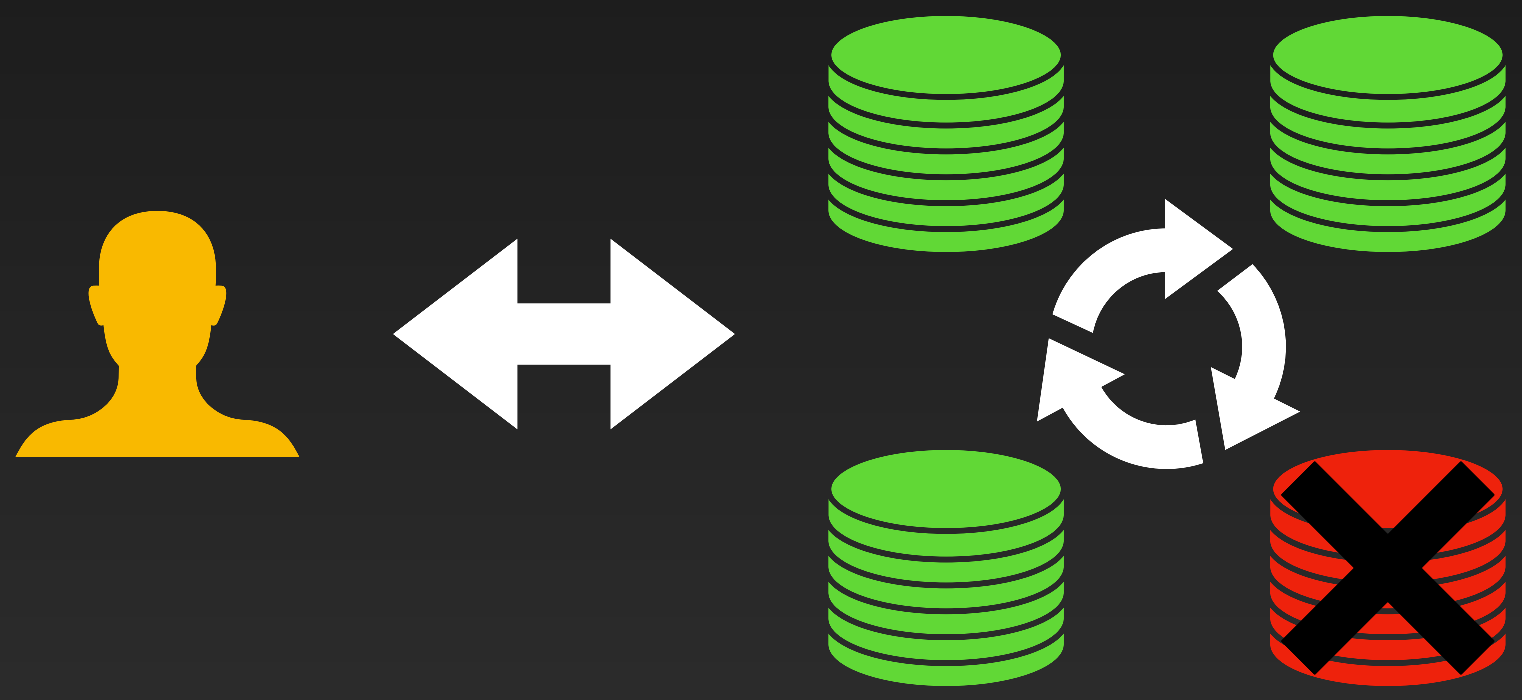


FastPay
Latency



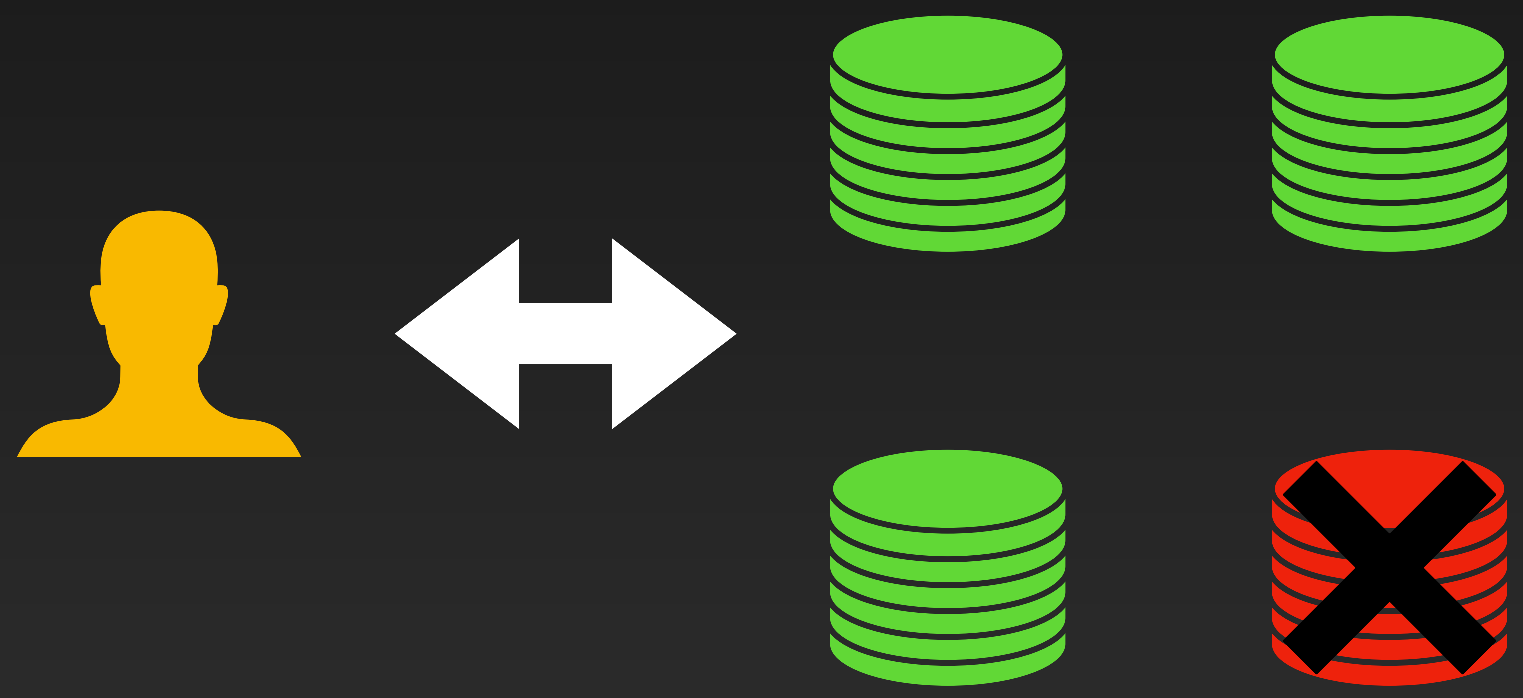
Worst-case efficiency

Blockchains



Bad leader can slow down the protocol

FastPay



No leader, nothing changes

Simplicity favors robustness and performance

Lesson I

Making it simple is hard

Lesson II

FastPay

The cost of simplicity

- Less than 4,000 LOC
- Over 1,500 Git commits
- Took 2.5 months to 3 engineers

FastPay
Deployment costs

- AWS m5d.8xlarge instance
- ~ 5 USD / hour

Conclusion

Part II - Reducing Latency

FastPay

- Based on Byzantine Consistent Broadcast
- Simple design, low latency, high capacity, very robust

- **Paper:** https://arxiv.org/abs/2003.11506
- **Code:** https://github.com/calibra/fastpay

Scaling Blockchains

Achieving Arbitrary Throughput and Sub-Second Latency

Conclusion

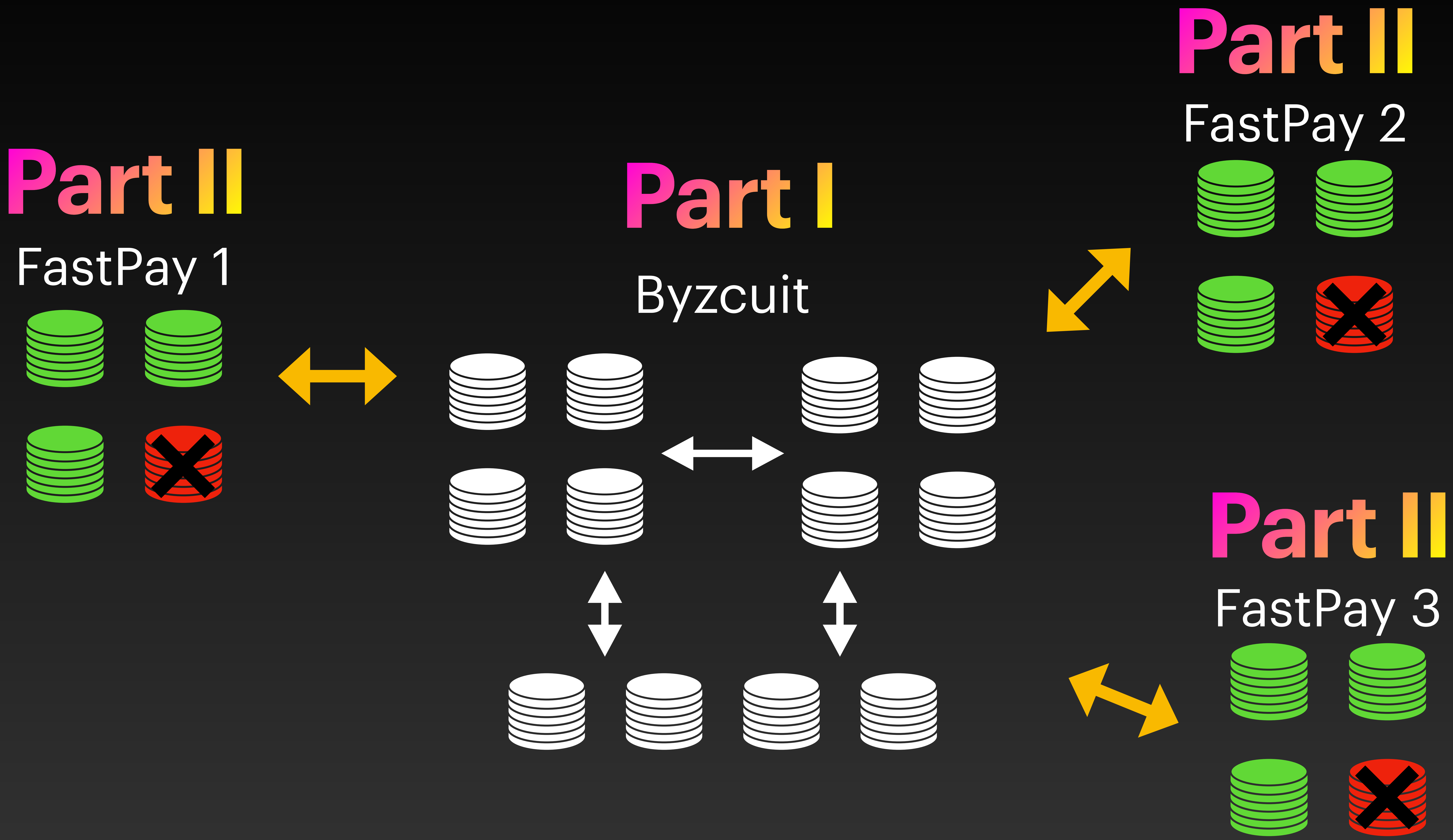
Outline

Part I:
Increase throughput

Through sharding

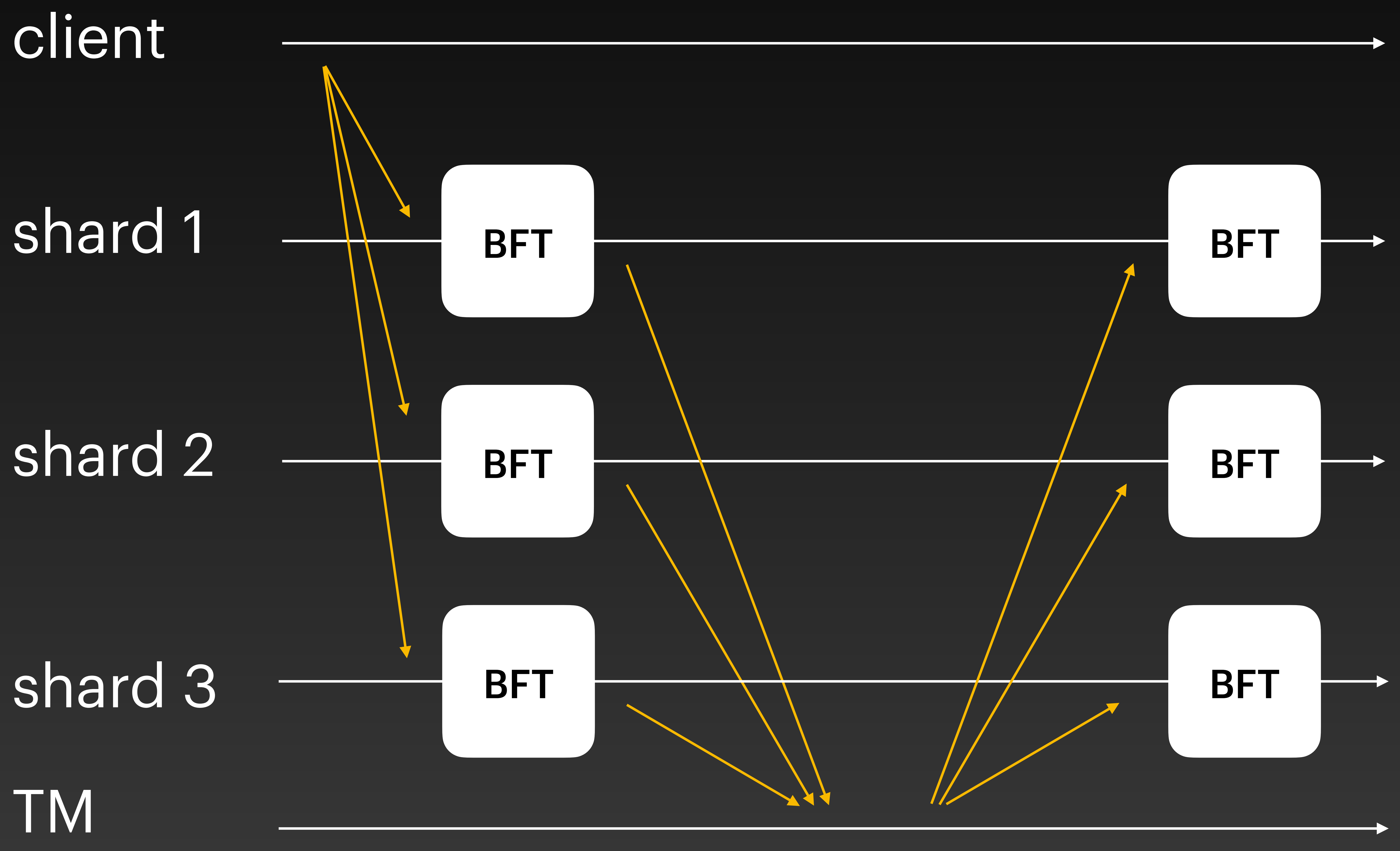
Part II:
Reduce latency

with side infrastructures


Part II
FastPay 2
Part II
FastPay 1
Part I
Byzcuit
Part II
FastPay 3

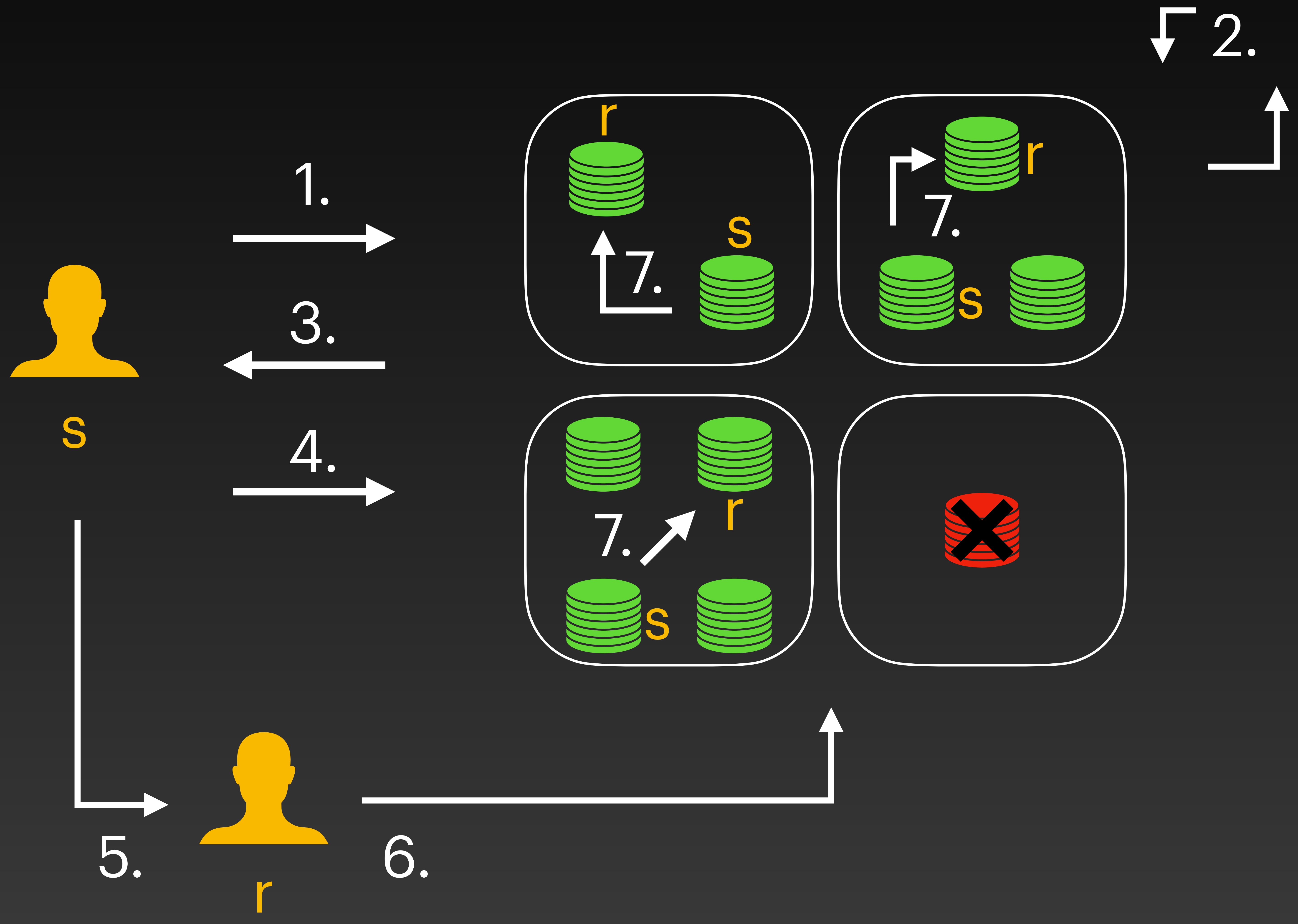
Cross-Shard Consensus

Tricky to implement right



BFT Side-Infrastructures

Byzantine Consistent Broadcast



Main Takeaways

Part I: Increase throughput

Key concepts of sharded distributed ledgers

Main challenges in building secure sharded systems in practice

Part II: Reduce latency

Side-infrastructures to bring blockchain-based payment systems to physical points of sales

How to integrate those infrastructures into a primary distributed ledger

asonnino@fb.com

Facebook Novi