Cerberus

Cerberus: Minimalistic Multi-shard Byzantine -resilient Transaction Processing. Arxiv'20

By: Jelle Hellings, Daniel P. Hughes, Joshua Primero, and Mohammad Sadoghi

Explained By **Prem Rao** and **Greesan Gurumurthy**

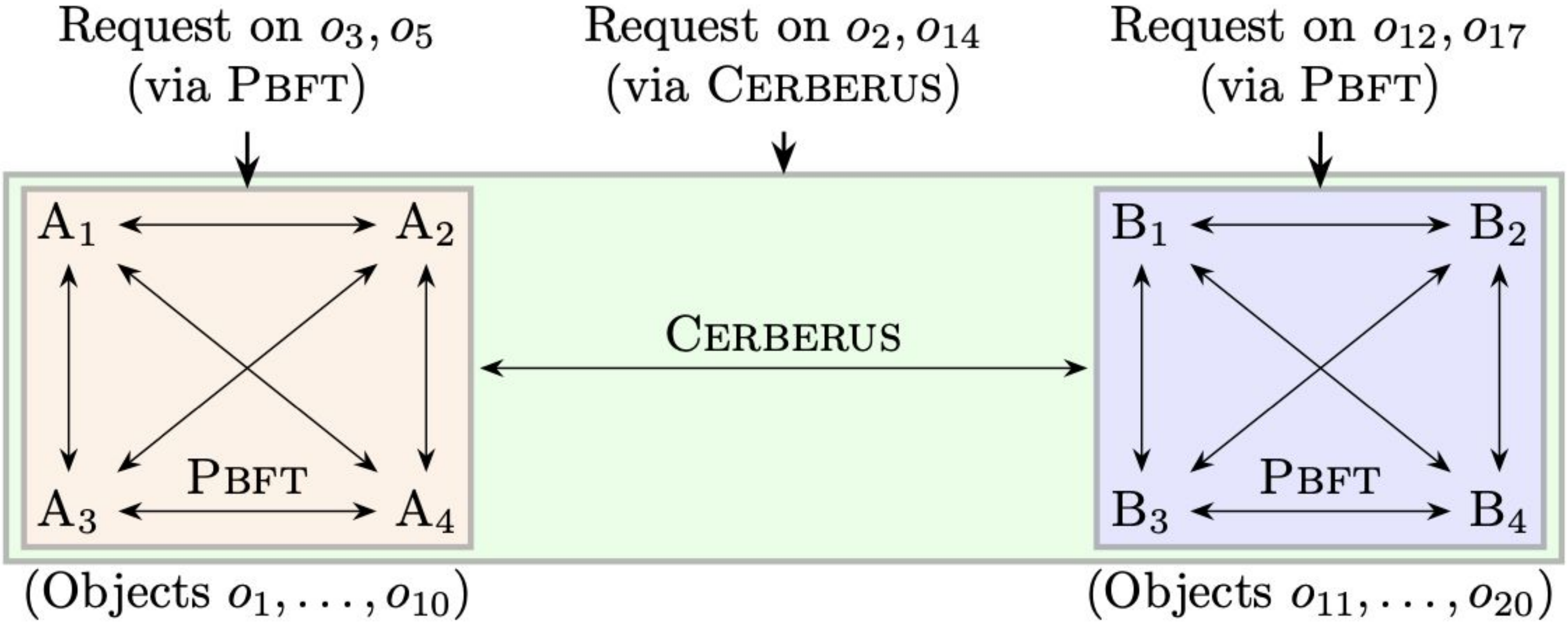
Table of Contents

What is Cerberus?

- Cerberus, a set of minimalistic primitives for processing single-shard and multi-shard UTXO-like transactions. Cerberus aims at maximizing parallel processing at shards while minimizing coordination within and between shards.
- **Easy Version**: A sharding protocol bred out of need for fast transaction speed and easy scalability

Why Cerberus?

1. Scalability
2. Transaction Speed
3. Comparison to Bitcoin/Mainstream Cryptocurrency
4. Cerberus can reach million transactions per second(STILL THEORETICAL)
5. Permissioned Blockchain vs Public Blockchain


Request on o_3, o_5 (via PBFT)
Request on o_2, o_{14} (via CERBERUS)
Request on o_{12}, o_{17} (via PBFT)
A_1
A_2
A_3
A_4
PBFT
CERBERUS
B_1
B_2
B_3
B_4
PBFT
(Objects $o_1, \ldots, o_{10}$)
(Objects $o_{11}, \ldots, o_{20}$)

Types of Cerberus

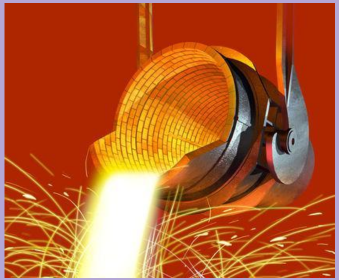
1. **Core Cerberus**:uses strict environmental requirements to enable simple yet powerful multi-shard transaction processing
2. **Optimistic Cerberus**:a protocol that does not require any additional coordination phases in the well-behaved optimistic case, while requiring intricate coordination when recovering from attack
3. **Pessimistic Cerberus**:a protocol that adds sufficient coordination to the well-behaved case of Core-Cerberus, allowing it to operate in a general-purpose fault-tolerant environment without significant costs to recover from attack

Shards explained

- Basically a cluster of nodes that hold a unique set of data
- Intra-shard transaction(txn): data impacted or used by txn is located in a single shard, therefore the shard can process the txn and report back
- Cross-shard transaction: data impacted or used by txn located in multiple shards
    - Harder to understand: need to cooperate among shards

Unspent transaction (UTXO) model

Transactions are done by destroying inputs to create specific outputs defined by the transaction message

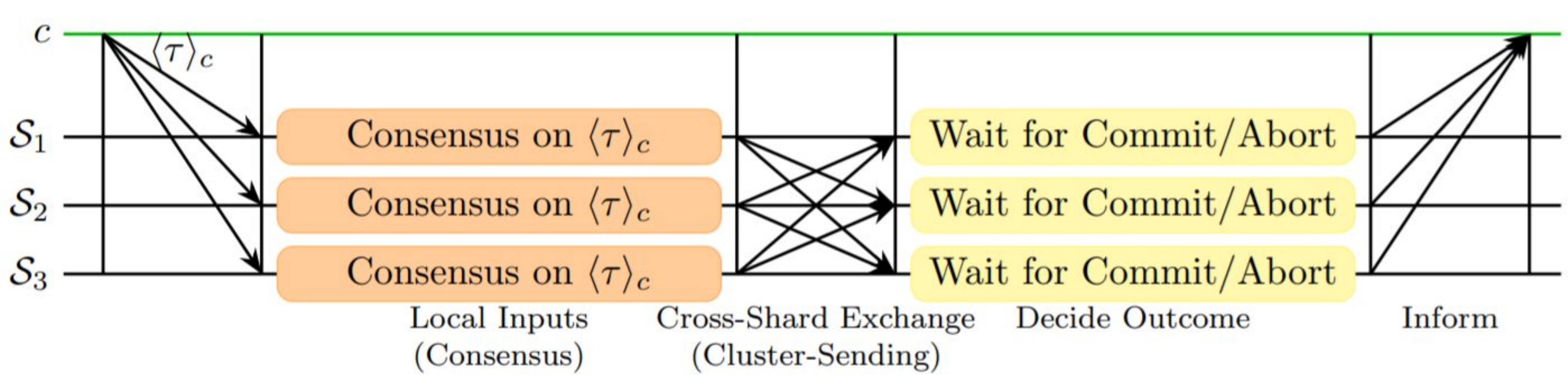


Core Cerberus

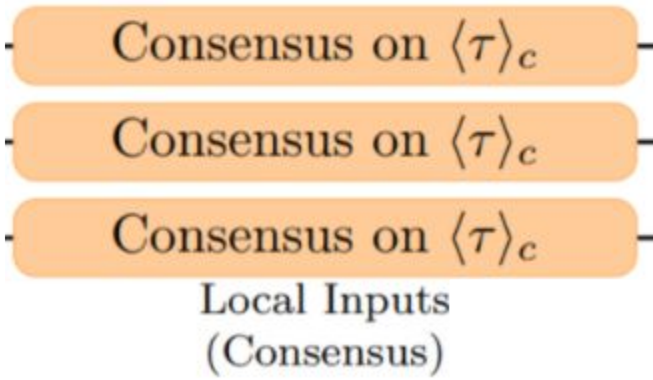
Mainly focusing on cross-shard txns

3 main steps:

- Local Inputs: locally decide if shard can contribute inputs to txn
- Cross-Shard Exchange: affected shards exchange/pledge inputs to all other affected shards
- Decide Outcome: if all inputs required by txn have been pledged, shards execute the txn
    - Destroy the inputs to produce the outputs



Message flow of CCerberus



Notation + Local Inputs:

Txn: $\langle\tau\rangle_c$

Participating Shard: $\mathcal{S} \in \texttt{shards}(\tau)$

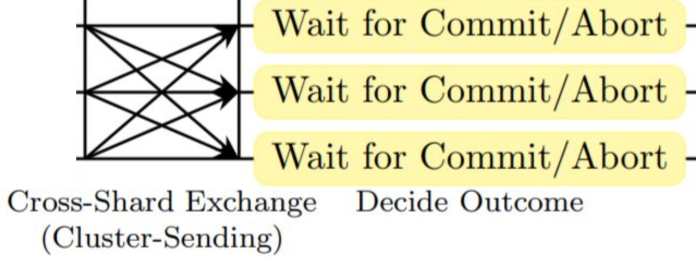
Inputs of S to txn: $I(\mathcal{S}, \tau) = \{o \in \texttt{Inputs}(\tau) \mid \mathcal{S} = \texttt{shard}(o)\}$

Currently available inputs of S to txn: $D(\mathcal{S}, \tau)$

If $I(\mathcal{S}, \tau) = D(\mathcal{S}, \tau)$ shard's primary $\mathcal{P}(\mathcal{S})$ pledges inputs to txn, sending

Message: $m(\mathcal{S}, \tau)_\rho = (\langle\tau\rangle_c, I(\mathcal{S}, \tau), D(\mathcal{S}, \tau))$ where ρ refers to the consensus round

X-shard + Deciding Outcome:



Cross-shard exchange:Shards broadcast message $m(\mathcal{S}, \tau)_\rho$ and wait to receive similar messages from all shards involved in txn

Decide Outcome:

If ($I(\mathcal{S}, \tau) = D(\mathcal{S}, \tau)\ \forall \mathcal{S} \in \mathtt{shards}(\tau)$) → commit

Else → abort

Each replica informs clients of execution outcome, if client receives identical results from f+1 replicas from each $\mathcal{S} \in \mathtt{shards}(\tau)$,client can verify that txn has been executed or aborted

Issue: deadlocks

Shards need confirmation from all other involved shards at Xshard exch

Shards might never reach cross-shard exchange step:

Ex: 2 txns: $\mathit{Inputs}(\tau_1) = \mathit{Inputs}(\tau_2) = \{o_1, o_2\}$ $\mathit{shard}(o_1) = \mathcal{S}_1$ $\mathit{shard}(o_2) = \mathcal{S}_2$

$\mathcal{S}_1$ Processes τ_1 first, $\mathcal{S}_2$ processes τ_2 first

$$\mathcal{S}_1 \underset{m(\mathcal{S},\tau_2)_{\rho_2} = ((\langle\tau_2\rangle_{c_2}, \{o_2\}, \{o_2\})}{\overset{m(\mathcal{S},\tau_1)_{\rho_1} = ((\langle\tau_1\rangle_{c_1}, \{o_1\}, \{o_1\})}{\rightleftarrows}} \mathcal{S}_2$$

Fix:

Internal propagation: shards broadcast msg(txn) to all other involved shards, therefore preventing net issues between client and clusters from stopping communication of the txn

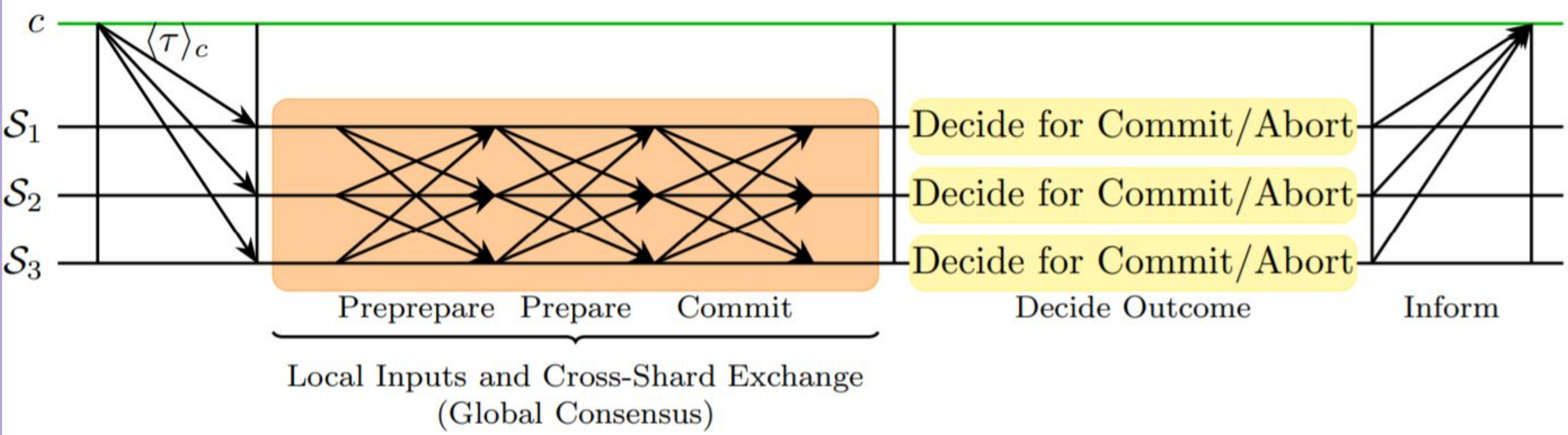
Concurrent resolution: Replicas implement first-pledge and ordered-commit: stops concurrent execution leading to inconsistent state updates

- First Pledge: shard pledges objects made in round p if txn is first transaction proposed after round p requiring those objects
- Ordered-Commit: only commit txn accepted in round p after previous rounds have finished execution

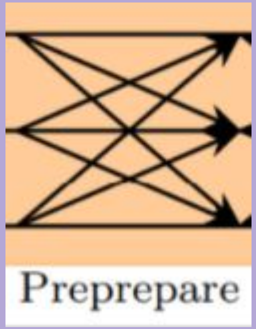
Optimistic Cerberus

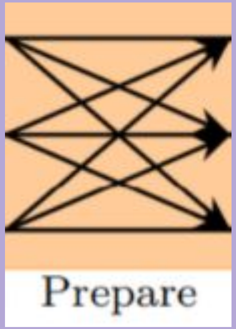
Drawbacks with CCerberus that OCerberus attempts to fix:

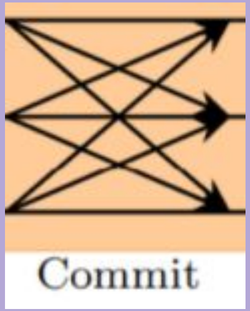
- Concurrent txns assumed to be result of malicious clients
  - Locks out objects
- Requires consensus than intershard exchange → txn speed decreased
- OCerberus optimized for when no concurrent txns occur
  - Provides recovery for concurrent without locking out objects
  - If malicious entities detected any individual replica can start recovery process
  - Lessens txn processing latency
  - Malicious entity detection and cross-shard coordination spun into one consensus



message flow of OCerberus







Global Preprep:
All primaries of involved shards send msg(txn) to all replicas of all other shards

global preprep certificate:
{msg(txn)}, ready for next step

Global Prep:
All replicas of involved shards send prep_msg to all other replicas in other involved shards

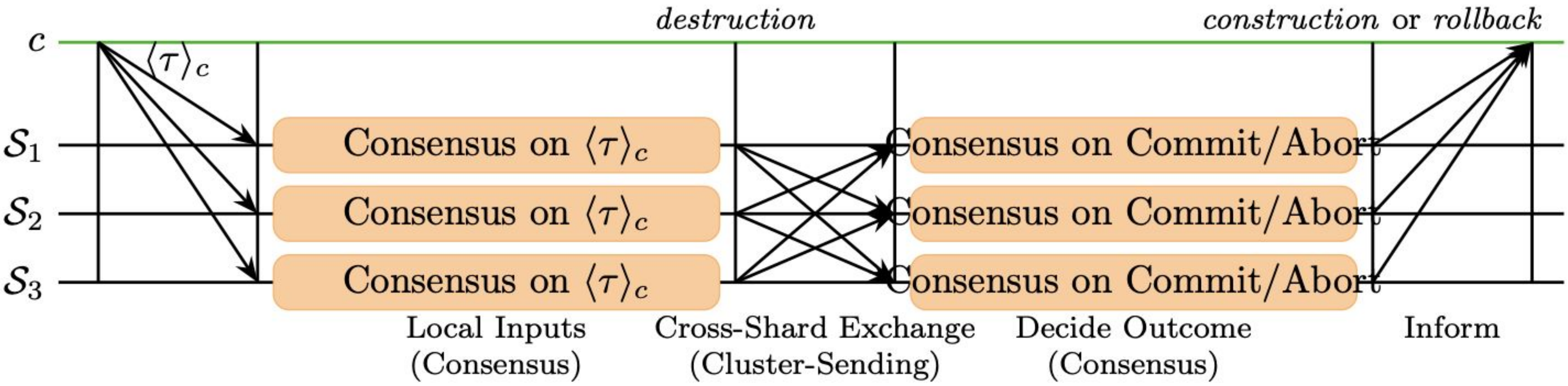
local prep certificate:
{prep_msg}
if length >= 2f+1, ready for next step

Global Commit:
All replicas of involved shards send commit_msg to all other replicas in other involved shards

global commit certificate:
{commit_msg}:
if length >= 2f+1
Ready to decide outcome

Pessimistic Cerberus

- Pessimistic Cerberus is essentially transaction processing under attack.
- We apply a pessimistic approach to CCerberus that is processed to recover from concurrent transactions and made for minimizing the influence of malicious behaviour.
- To better explain PCerberus we can use the following illustration to understand the basic functioning



The message flow of PCerberus for a 3-shard client request that is committed.

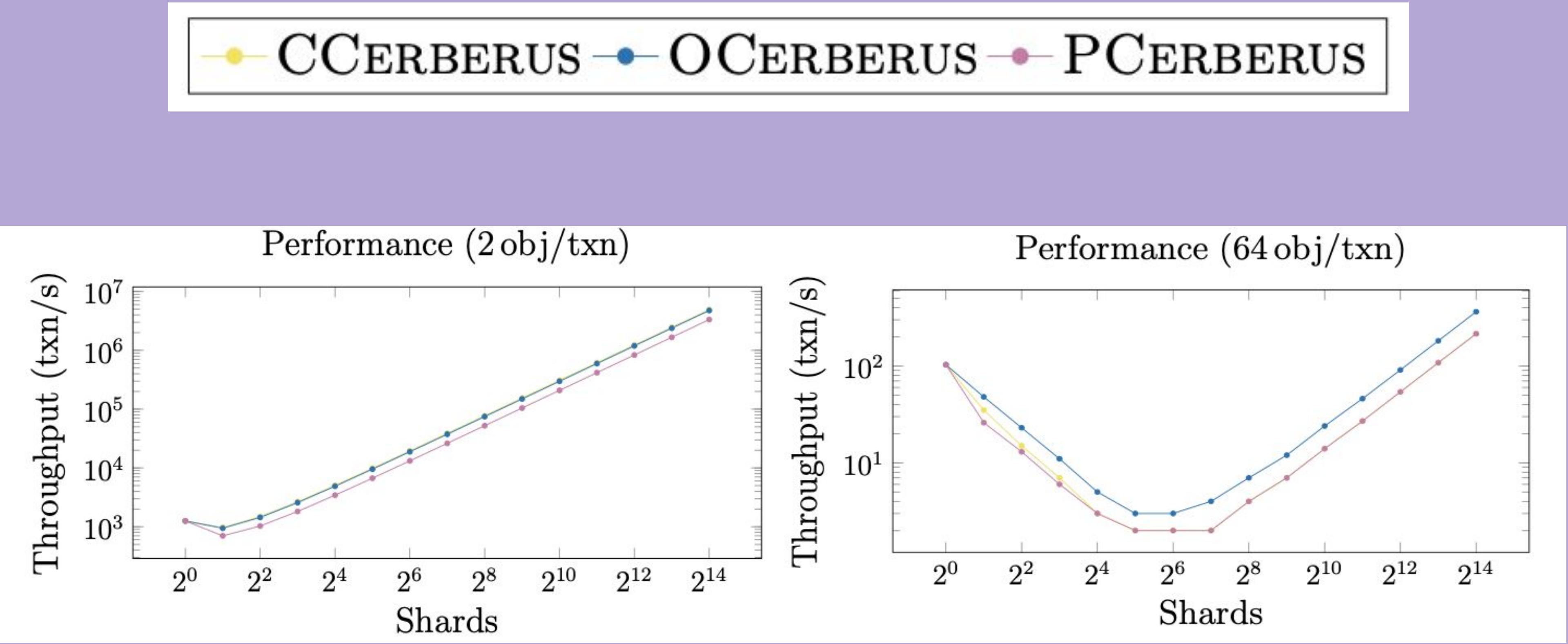
The design of PCerberus adds on to the framework of Core Cerberus by adding coordination to the cross-shard exchange and decides outcome steps.

The basic **process** is as follows:

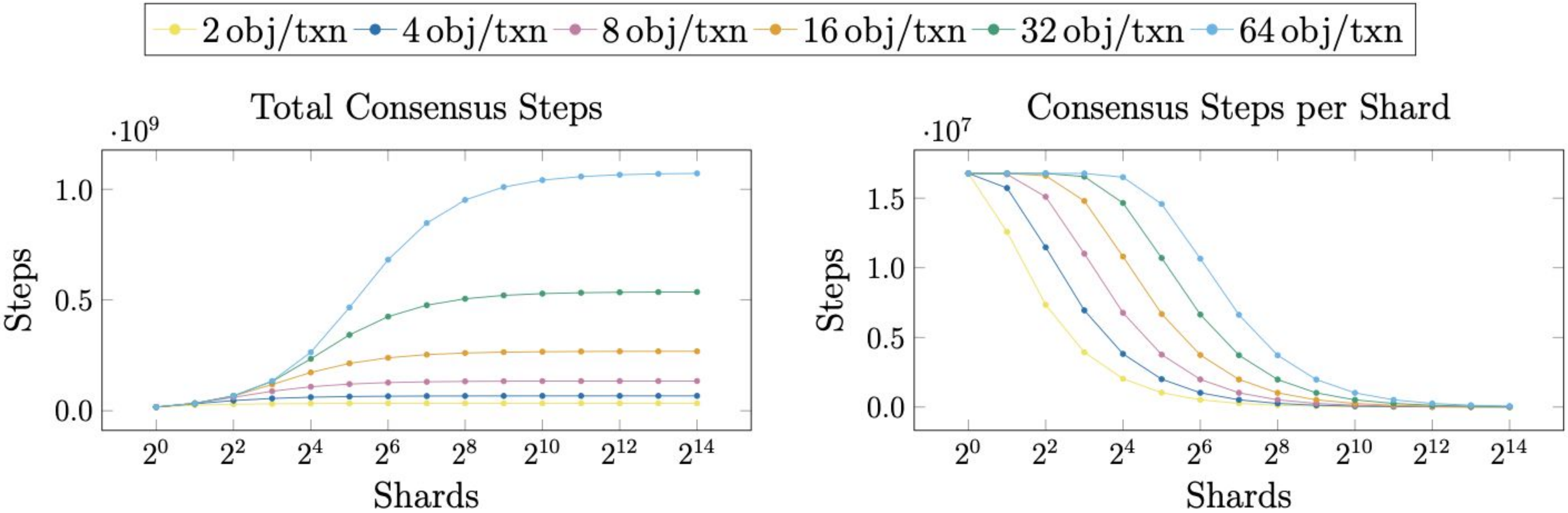
- Acceptance of message by all the replicas -> local inputs step
- Before cross shard exchange, the replicas in the Shard destruct the objects in the readily available inputs(D), thereby fully pledging these objects to τ until the commit or abort decision
- Then, the shard performs cross-shard exchange by broadcasting messages to all the other shards , while the replicas in the shard wait until they receive messages from all other shards
- After cross-shard exchange comes the final decide outcome step.
- We notice that processing a multi-shard transaction via PCerberus requires **two consensus steps** per shard.

Strengths of Cerberus

- The main advantages of this protocol through its three mediums are multifaceted:
- Provides Serializable execution
- Maximises per-shard throughput by harnessing out-of-order processing
- Very scalable; one of the reasons for its inception
- Fast transaction speeds, high attainable performance
- Reasonable cost basis; in theory



For every 2^n increase in objects per transaction, the throughput decreases by a factor of 10



Amount of work, in terms of consensus steps, for the shards involved in processing the transactions

Thank you!

Any Questions?