

WormSpace: A Modular Foundation for Simple, Verifiable Distributed Systems

ACM Symposium on Cloud Computing

Nov 22, 2019

Ji-Yong Shin¹ Jieung Kim¹ Wolf Honore¹ Hernan Vanzetto¹
Srihari Radhakrishnan² Mahesh Balakrishnan³ Zhong Shao¹

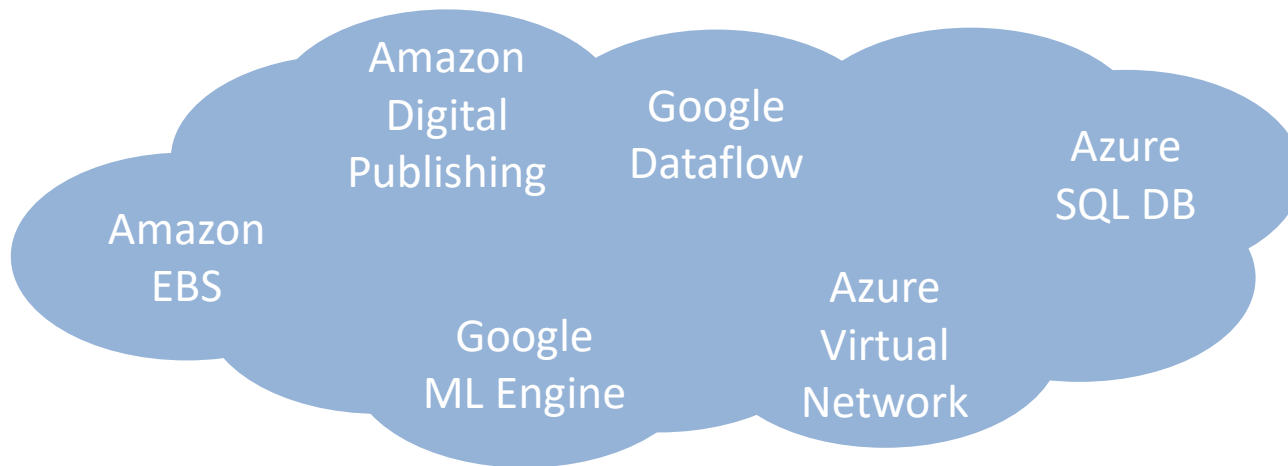
¹Yale University

²Duke University

³Facebook

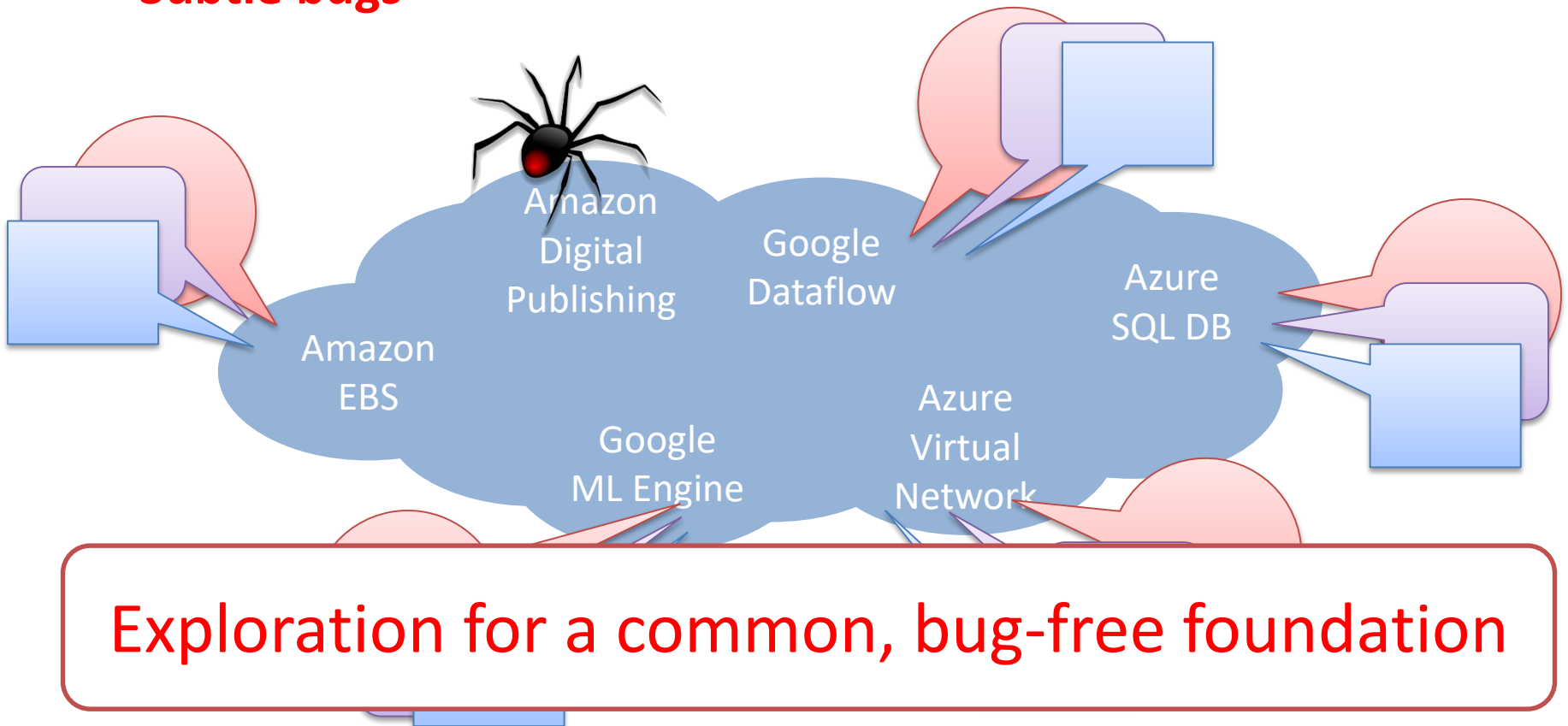
Cloud and Distributed Application Environment

- Numerous distributed services are readily available
- New applications are built by combining existing building blocks
- New services are continuously developed and deployed



Cloud and Distributed Application Environment

- Distributed services use and re-implement similar features
Redundant efforts
- Distributed systems are complex and difficult to build correctly
Subtle bugs



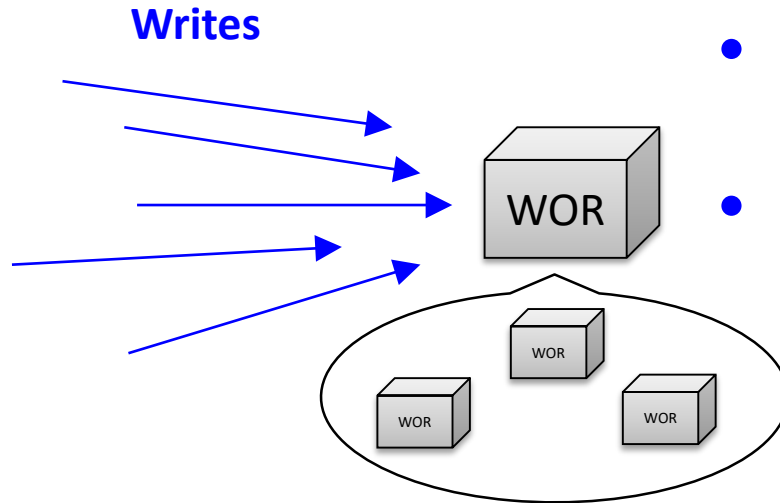
Design Goals

1. Supports common needs for most systems
2. Simple and easy-to-understand APIs
3. Flexible support for optimizations
4. Guaranteed correctness with extensibility

System design

Formal Verification

Write once register (WOR)



- Logically equivalent to consensus (Paxos, Chain-replication, PBFT, etc.)
- Lowest common denominator

- Distributed register
 - Replicated by construction (**fault tolerance, availability, durability**)
- Write-once-read-many abstraction
 - Atomically writes data (**consistency**)
 - Only one of concurrent writes succeeds (**concurrency control, immutability**)

WORs in Existing Systems

- State machine replication (SMR) and multi-Paxos
 - Append / sequential read to WORs
- Shared log: Corfu, Tango
 - Append / random read to WORs
- Transaction coordinator: 2 phase commit
 - Random write / random read to WORs
- Coordination service: chubby, zookeeper
 - File APIs over SMR on WORs
- Group communication: pub/sub
 - Append / sequential read to WORs

WOR APIs

- Capture

- Preemptible lock concept
- Coordination before write
- Returns a capture token

Paxos: phase 1 prepare
PBFT: pre-prepare + prepare
Chain-replication: no-op

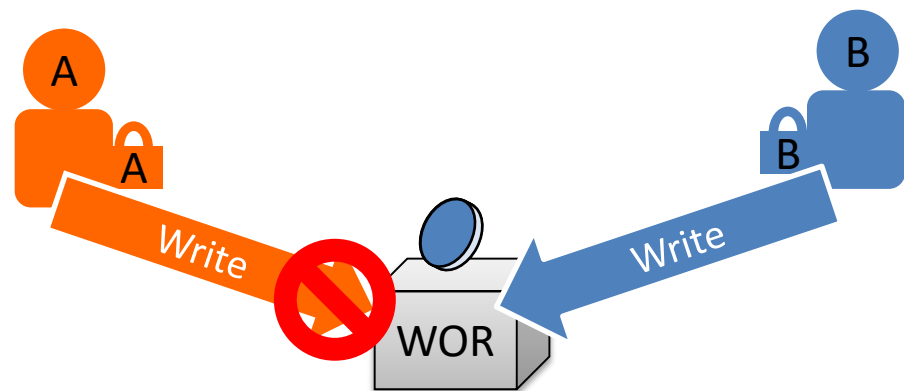
- Write

- Writes to the WOR
- Capture must be valid

Paxos: phase 2 accept
PBFT: commit
Chain-replication: write to the chain

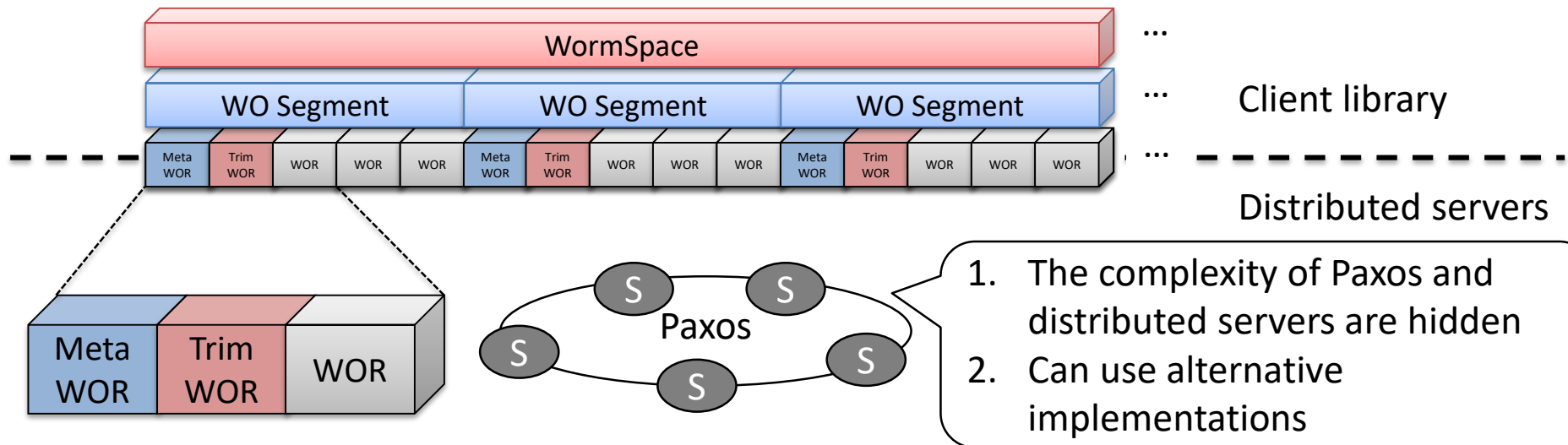
- Read

- Reads the register
- Returns data or “empty”



WormSpace (Write-Once-Read-Many Address Space)

- An address space of WORs
- Write-once-segment (WOS) for management
 - Unit of allocation (**alloc**) and garbage collection (**trim**)
 - Consists of special WORs and data WORs
 - Support for **batch-capture** and **batch-write** to all WORs



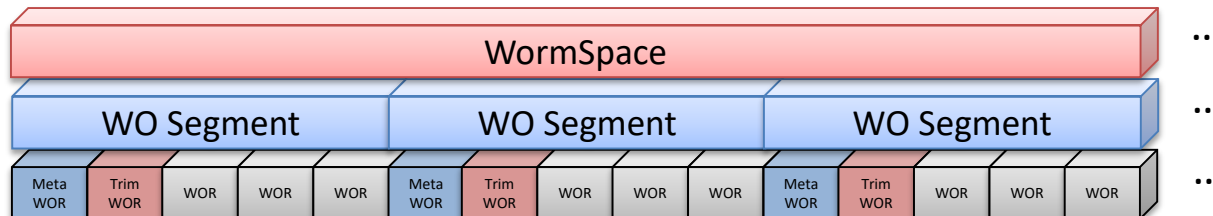
WormSpace Applications

- WormPaxos
 - Multi-Paxos / state machine replications

- WormLog
 - Corfu / shared-log

Please refer to the paper for interesting latency optimizations

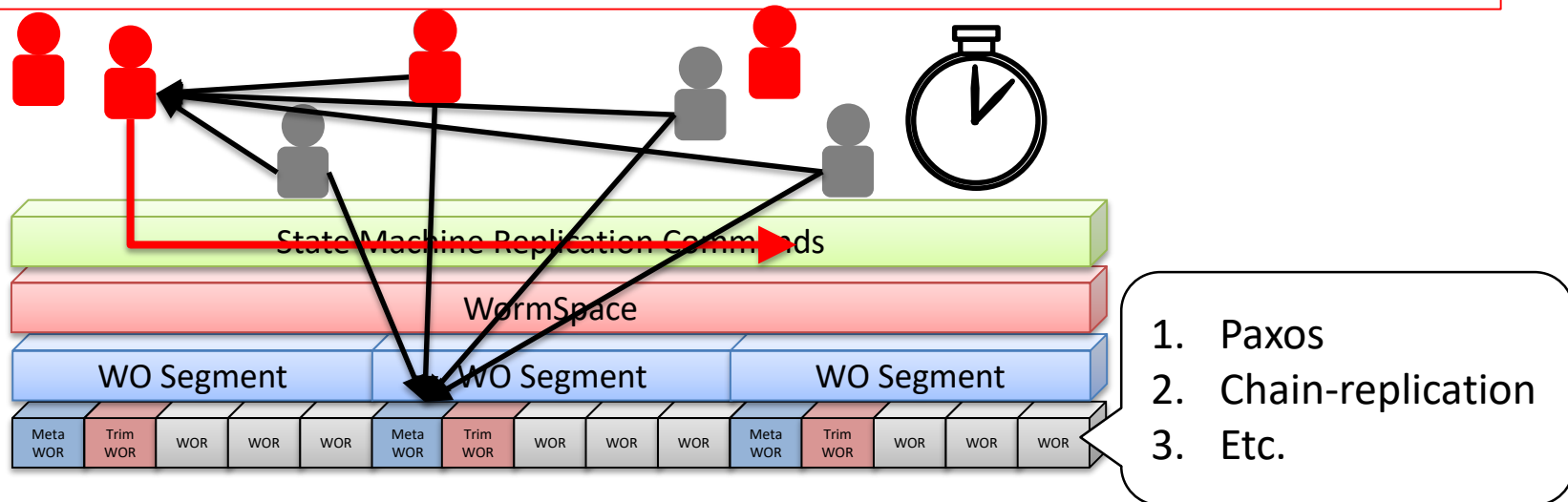
- WormTX
 - 2PC variant / non-blocking atomic commit



WormPaxos: Flexible Design Choices

- Multi-Paxos variant for state machine replication
- Design decisions can be easily configured
 - Various single-degree consensus protocols
 - Mencius-like rotating leaders are easy to implement
 - Raft-like leader election can be implemented orthogonally with a timer
 - Leader election: who allocates a WOS and batch captures it?
 - When to call trim call determines durability

WormSpace APIs are enough and no need to understand Paxos



Formal Verification

- WOR is primitive, but encapsulates key distributed properties
 - Consistency, durability and availability

Can we verify WOR once and reuse it multiple times?

- Concurrent Certified Abstraction Layer (CCAL) [Gu, et al. PLDI 18]
 - Divides software into layers
 - Verifies each layer
 - Verifies layers interact correctly
 - Lower layer properties hold in higher layers

Certified Concurrent Abstraction Layer (CCAL)

$$\forall (t': context), L_{List} (Impl_{Queue} \oplus Impl_{Sched} \oplus t') \sqsubseteq L_{Sched}(t')$$

L_{Sched} (Uses L_{Queue})

Contextual refinement proof

L_{Queue} (Uses L_{List})

Contextual refinement proof

Specification

L_{List} Refinement proof

C Implementation



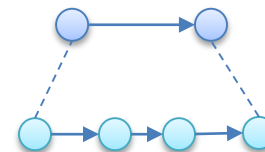
Sched does not need to know about List at all!

$$\forall (t': context), L_{Queue} (Impl_{Sched} \oplus t') \sqsubseteq L_{Sched}(t')$$

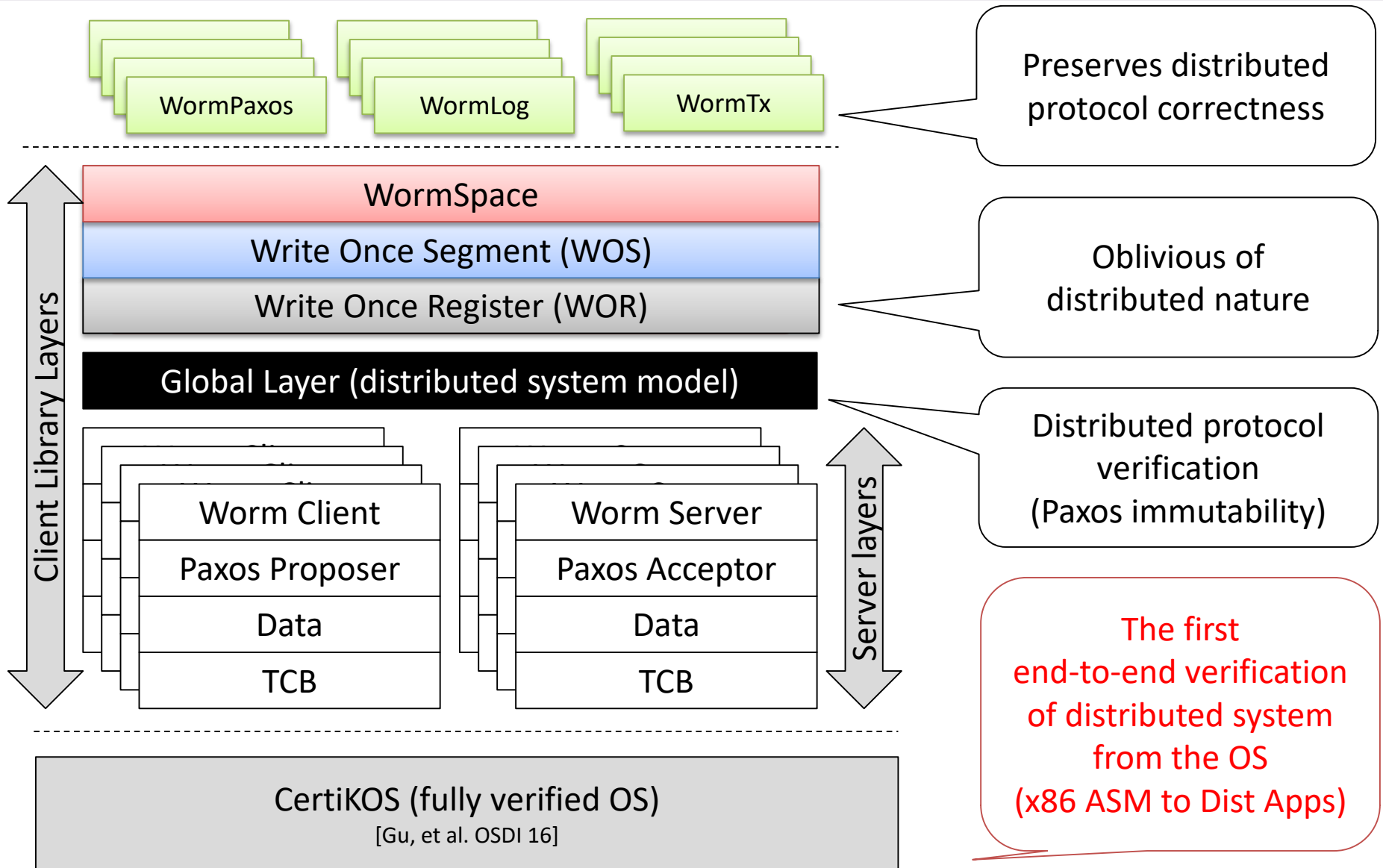
$$\forall (t: context), L_{List} (Impl_{Queue} \oplus t) \sqsubseteq L_{Queue}(t)$$

Informally,

when we run **ANY Program (context)** on the **Queue**,
the state reached by the Queue has a matching state with
the **List** which runs the **Queue's impl** and the **Program**.



Verification Details



Experience

108K Lines of Coq Proof

359
CLoC

WormPaxos

362
CLoC

WormLog

547
CLoC

WormTx

< 1 month

WormSpace

Write Once Segment (WOS)

Write Once Register (WOR)

4.5K CLoC

6 months

Global Layer (distributed system model)

1.5 months

Worm Client

Paxos Proposer

Data

TCB

Worm Server

Paxos Acceptor

Data

TCB

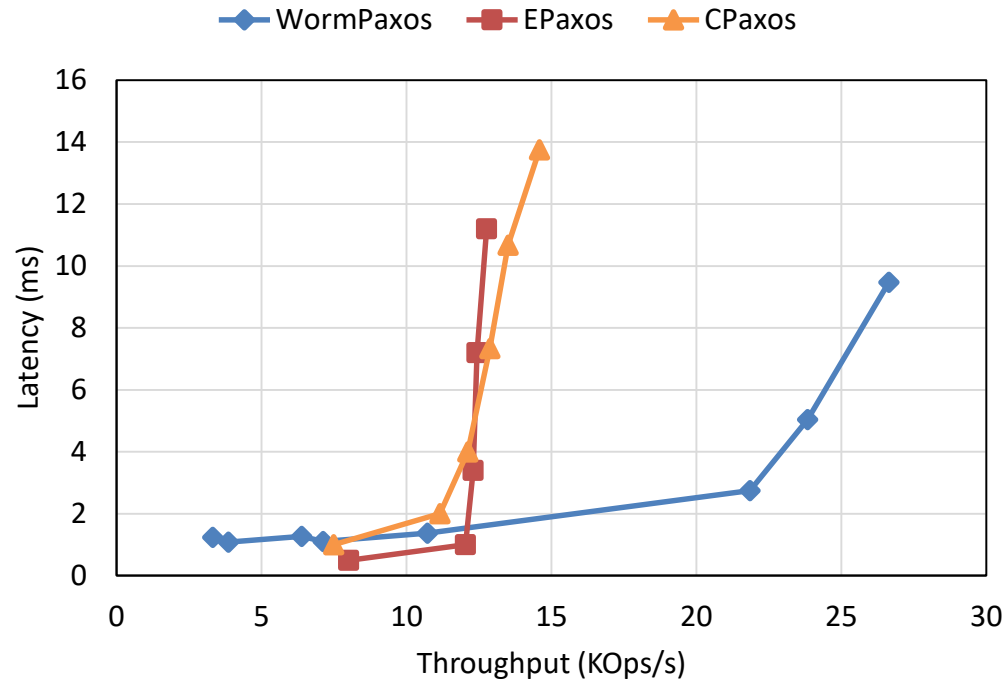
Client Library Layers

Server layers

- Simple API and no need to understand distributed protocols
- Distributed verification is hidden, but verified properties hold

Evaluation

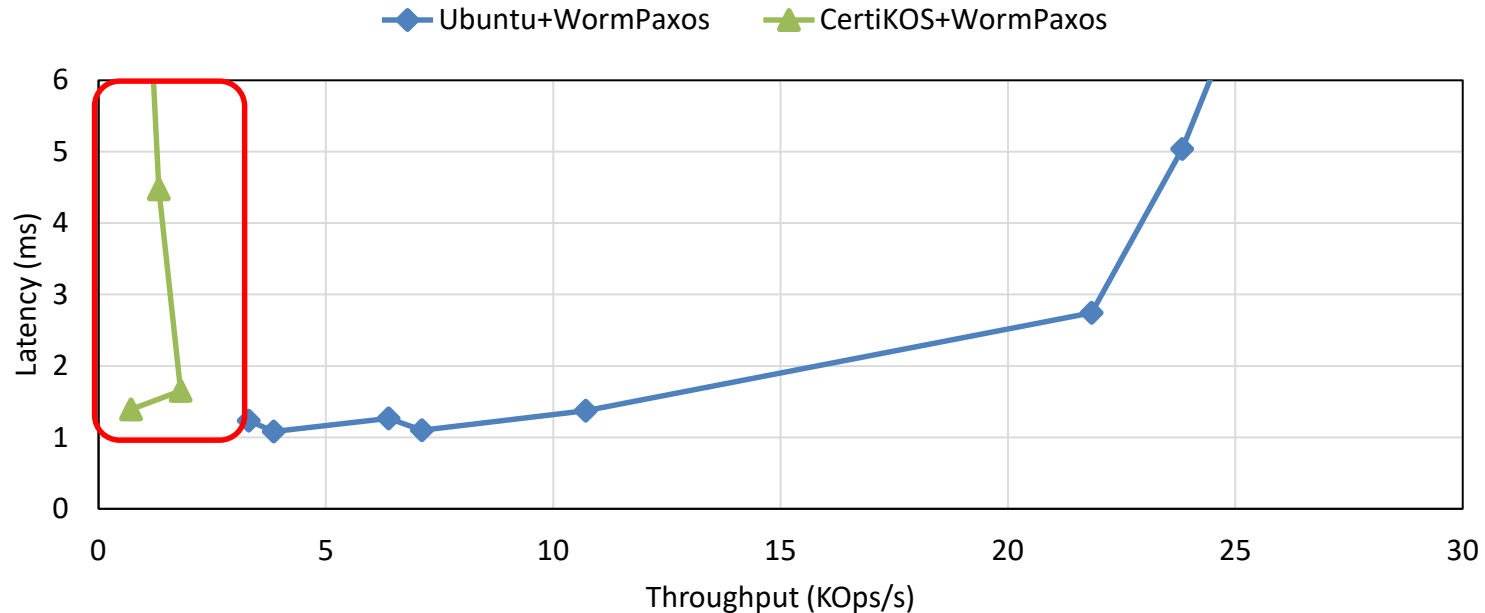
- WormPaxos vs Egalitarian Paxos and its classical multi-Paxos impl.
 - Amazon EC2: 3 servers and 16 client nodes
 - Write-only benchmark
 - C vs. Go and different internals



Verified systems are not slow!

Evaluation

- WormSpace over CertiKOS
 - Local cloud with same configuration as Amazon EC2



- Over 10X lower throughput and over 1.5X higher latency
- Mainly due to inefficiencies in LwIP of CertiKOS

Conclusion

- Write once registers for programming
 - Lowest common denominator for most systems
 - Source of consistency, availability, and durability
- Write once register for verification
 - Primitive module that encapsulates key distributed system properties
 - Can be verified once and reused to simplify application verification
- WormSpace for simple, verifiable distributed systems
 - Address space of WOR and with extra APIs
 - Allows for simple and flexible distributed application designs
 - Facilitates verification of distributed applications

Thank you

Questions?