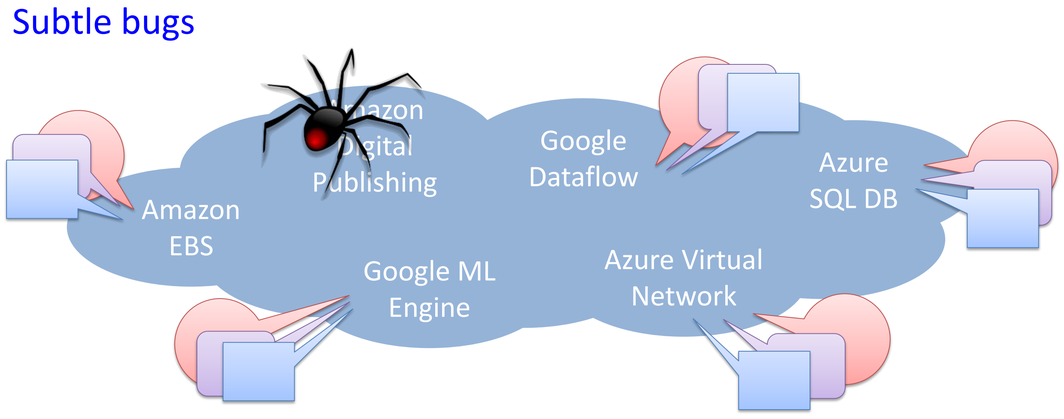
WormSpace: A Modular Foundation for Simple, Verifiable Distributed Systems

Ji-Yong Shin[1] Jieung Kim[1] Wolf Honore[1] Hernan Vanzetto[1] Srihari Radhakrishnan[2] Mahesh Balakrishnan[3] Zhong Shao[1]

[1]Yale University [2]Duke University [3]Facebook

Cloud and Distributed Application Environment

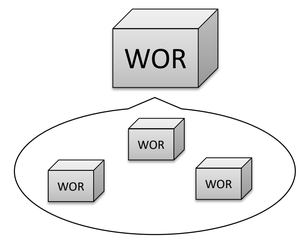
- Distributed services use and re-implement similar features (e.g., replication, logging, transaction, etc.)
  Redundant efforts
- Distributed systems are complex and difficult to build correctly
  Subtle bugs



A common, bug-free foundation is necessary

Write once register (WOR) Abstraction

- Distributed register
  - Replicated by construction (fault tolerance, availability, durability)
  - Stores a data unit
- Write-once-read-many semantics
  - Atomically writes data (consistency)
  - Only one of concurrent writes succeeds (concurrency control, immutability)
- Logically equivalent to consensus but not tied to a specific protocol (e.g., Paxos, chain-replication, PBFT, etc.)



APIs

Capture	**Write**	**Read**
Preemptible lock Coordination for write (e.g., Paxos p1-prepare)	Writes to WOR Capture must be valid (e.g., Paxos p2-accept)	Reads a WOR Returns data or 'empty'

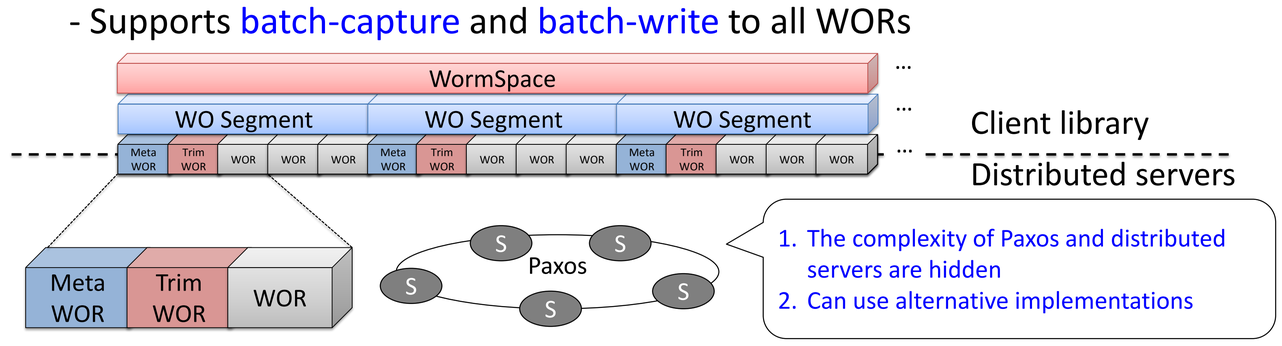
WORs in Existing Systems

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

WORs are hidden under high-level APIs

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

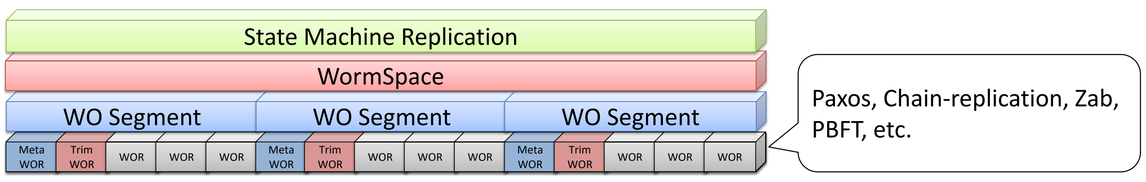
- A common foundation for distributed systems
- An address space of WORs
- Write-once-segment (WOS) for management
  - Unit of allocation (alloc) and garbage collection (trim)
  - Consists of special WORs and data WORs
  - Supports batch-capture and batch-write to all WORs



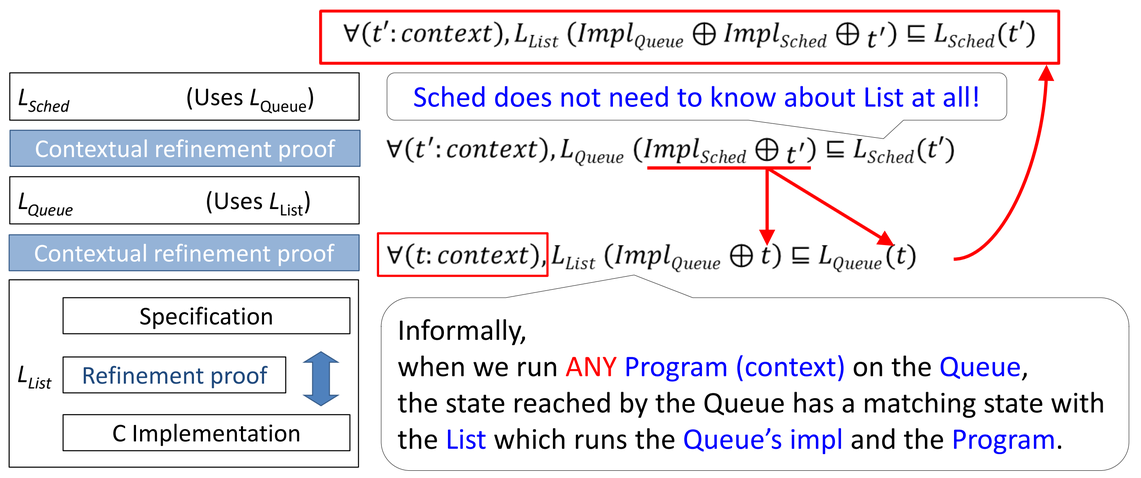
Flexible Application Design Choices (Multi-Paxos/SMR Example)

- Flexible consensus
  - Different choice of consensus protocols (not tied to Paxos)
- Flexible leader election
  - Basic election: who allocates a WOS and batch captures it?
  - Mencius-like rotating leaders: assign a leader per segment
  - Raft-like leader election: can be implemented orthogonally
- Various durability policies
  - When to call trim call determines durability

WormSpace APIs are sufficient (don't need to know Paxos)



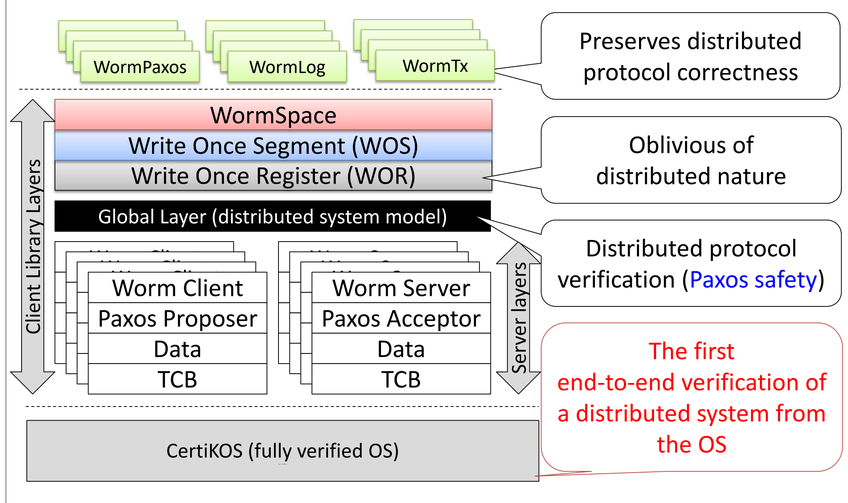
Verification Approach (Concurrent Certified Abstraction Layer)



1. Verify each layer independently
2. Combine adjacent layers with refinement proofs

* Correctness properties transitively hold from bottom to top layers

WormSpace Verification



Distributed protocol proof is encapsulated but verified properties hold in higher layers

➡ WormApp verification becomes trivial

Experience

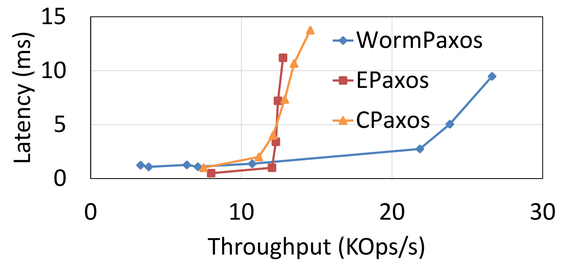
- WormSpace Apps are easy to build (CLoC)

WormSpace	WormPaxos	WormLog	WormTX
4,551	359	362	547

- WormSpace Apps are easy to verify (Person month)

WormSpace	WormPaxos	WormLog	WormTX
6 months	1 week	1 week	1 week

Performance



Verified systems are not necessarily slow!

Conclusion

- WormSpace from systems viewpoint
  - Use of WOR as a programming abstraction
  - Lowest common layer for dist. sys.
  - Facilitates easy ands flexible system building
- WormSpace from verification viewpoint
  - Primitive and modular WOR-based
  - Encapsulates core dist. sys. properties
  - Easy reuse of proofs with layered verification