TLA+ Quinceañera

FLoC TLA+ Workshop

David Langworthy

Microsoft Engineer

2003: WS-Transaction

Formal Specification of a Web Services Protocol

James E. Johnson, David E. Langworthy, Leslie Lamport

Microsoft

Friedrich H. Vogt

University of Technology Hamburg-Harburg

April 2015: How AWS Uses Formal Methods

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

December 2015: Christmas TLA+

- December 26th: Email from Satya to Me & 20 or so VPs
  - Not Common
- TLA+ is Great
- We should do this
- Go!

April 2016: TLA+ School

- 2 Days Lecture & Planned Exercises
- 1 Day Spec'ing
- Goal: Leave with the start of a spec for a real system
- 80 seats
  - Significant waitlist
- 50 finished
- 13 Specs Started
- Now run 3 times

April 2018

- TLA+ Workshop
- Application of TLA+ in production engineering systems
  - Mostly Azure
- 3 Execs
- 6 Engineers
- Real specs of real systems finding real bugs

Quinceañera

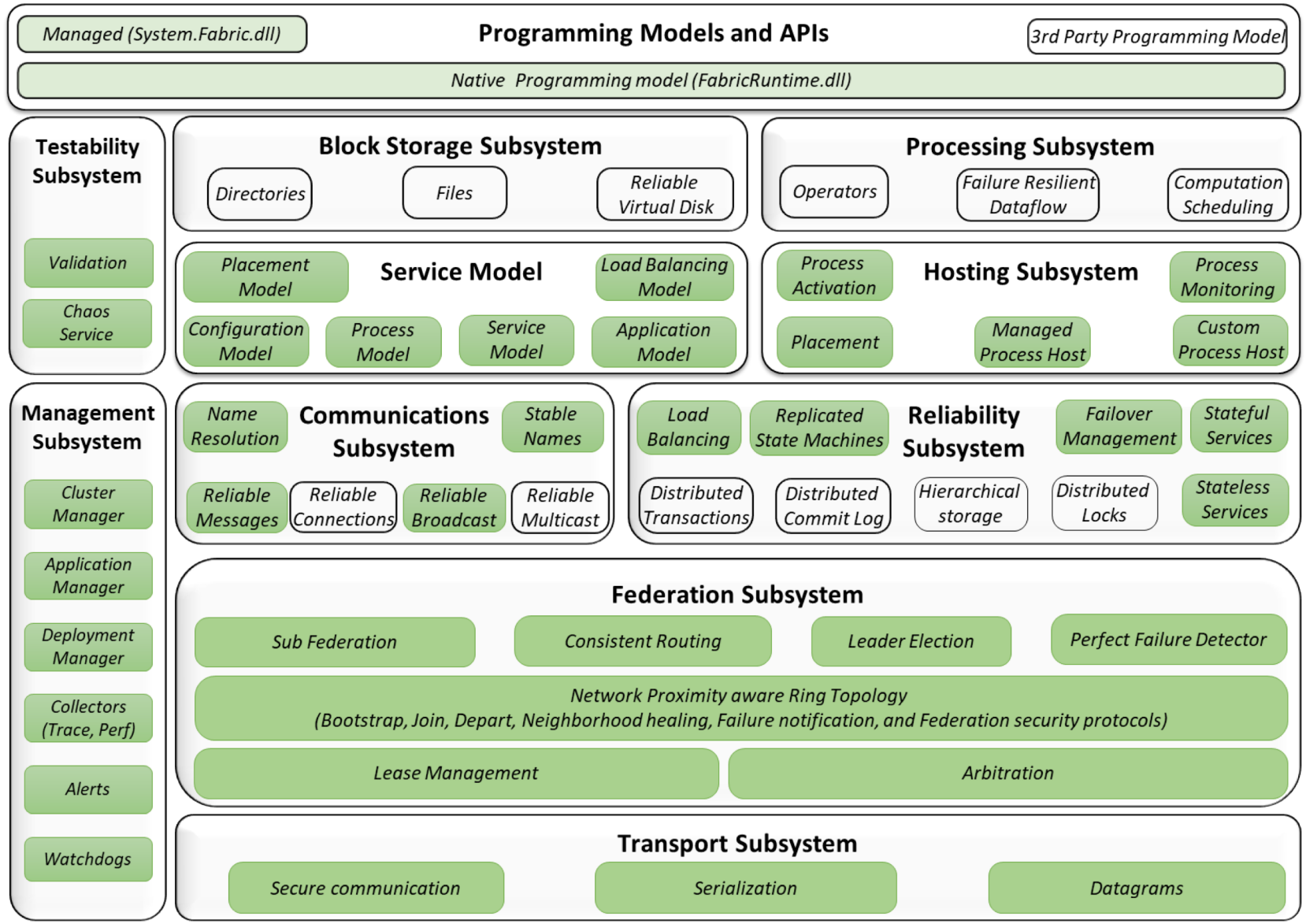
- Latin American tradition
- Woman's 15th birthday
- Introduction to society as an adult
  - Party
  - Fancy dress

Systems

- Service Fabric
- Azure Batch
- Azure Storage
- Azure Networking
- Azure IoT Hub

Product: Service Fabric

- People
  - Gopal Kakivaya
  - Tom Rodeheffer
- System: Federation Subsystem


Managed (System.Fabric.dll)
Programming Models and APIs
3rd Party Programming Model
Native Programming model (FabricRuntime.dll)
Testability Subsystem
Validation
Chaos Service
Block Storage Subsystem
Directories
Files
Reliable Virtual Disk
Processing Subsystem
Operators
Failure Resilient Dataflow
Computation Scheduling
Placement Model
Service Model
Load Balancing Model
Configuration Model
Process Model
Service Model
Application Model
Process Activation
Hosting Subsystem
Process Monitoring
Placement
Managed Process Host
Custom Process Host
Management Subsystem
Cluster Manager
Application Manager
Deployment Manager
Collectors (Trace, Perf)
Alerts
Watchdogs
Name Resolution
Communications Subsystem
Stable Names
Reliable Messages
Reliable Connections
Reliable Broadcast
Reliable Multicast
Load Balancing
Replicated State Machines
Reliability Subsystem
Failover Management
Stateful Services
Distributed Transactions
Distributed Commit Log
Hierarchical storage
Distributed Locks
Stateless Services
Federation Subsystem
Sub Federation
Consistent Routing
Leader Election
Perfect Failure Detector
Network Proximity aware Ring Topology
(Bootstrap, Join, Depart, Neighborhood healing, Failure notification, and Federation security protocols)
Lease Management
Arbitration
Transport Subsystem
Secure communication
Serialization
Datagrams

Product: Service Fabric

- People
  - Gopal Kakivaya
  - Tom Rodeheffer
- System: Federation Subsystem
- Invariant violations found by TLC : None noted
- Insights:
  - Clear definition of system
  - Verification with TLC

Product: Azure Batch

- People: Nar Ganapathy
- System: Pool Server
  - PoolServer manages the creation/resize/delete of pools
  - Has to enforce and maintain batch account quota
  - Need to track persistent data across many operations

PoolServer

- A role in Azure Batch Service with multiple instances
- Responsible for Pool Entity in the REST API
- Underneath a pool is a collection of VMSS deployments (e.g., 1000 VMs could be 200 deployments of 50 VMs each)
- A pool can be really large (can hold 10K-100K VMs)
- PoolServer manages the creation/resize/delete of pools
- Has to enforce and maintain batch account quota
- Maintain subscription quotas
- Has to build a deployment breakdown of the pool across many subscriptions
- Create deployments by talking with RDFE/CRP
- Pool creation is a long process and failovers can happen any time

What did I get out of my experience

- A compact, precise model of core pool server functionality
  - Real code is several 10s of thousands of lines
  - Eliminate environmental complications that are not germane to core algorithms
    - E.g., skipped modeling VMSS mechanisms, updating table storage
  - Relatively easy to explain to someone new
- Precisely understood the safety and liveness properties
- Developing the invariants where very valuable and these carried over into the code
  - The TLA+ rigor makes my ability to write asserts more effective
- I later decided to adopt an MSR state machine runtime called Psharp which has many of the properties of TLA+ but at a much lower level
- TLA+ model helped me write the safety and liveness properties in Psharp

Product: Azure Batch

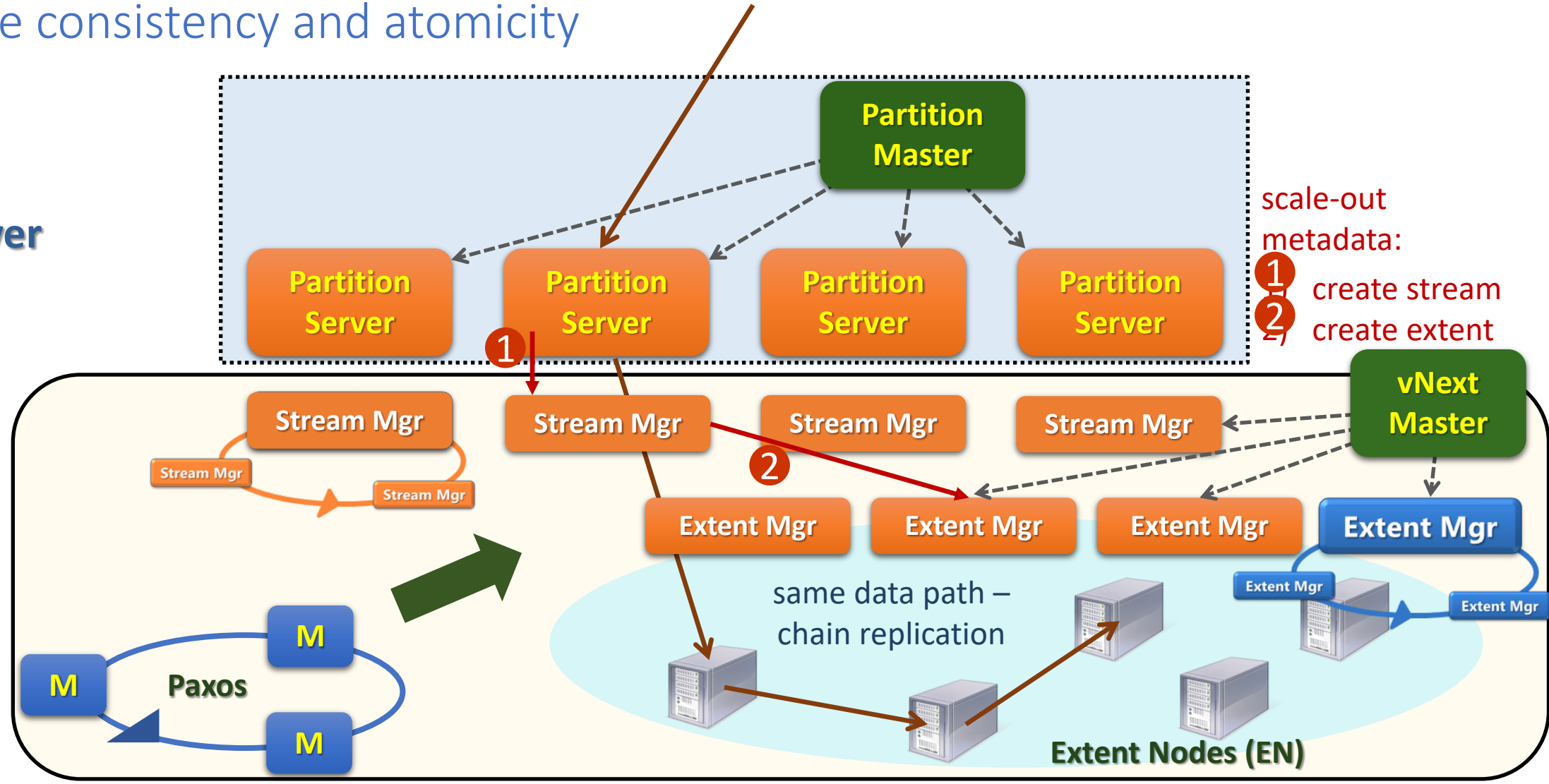
- People: Nar Ganapathy
- System: Pool Server
  - PoolServer manages the creation/resize/delete of pools
  - Has to enforce and maintain batch account quota
  - Need to track persistent data across many operations
- Invariant violations found by TLC : None Noted
- Insights
  - A compact, precise model of core pool server functionality
  - Precisely understood the safety and liveness properties
  - Developing the invariants was very valuable

Product: Azure Storage

- People: Cheng Huang
- System: Paxos Ring Management

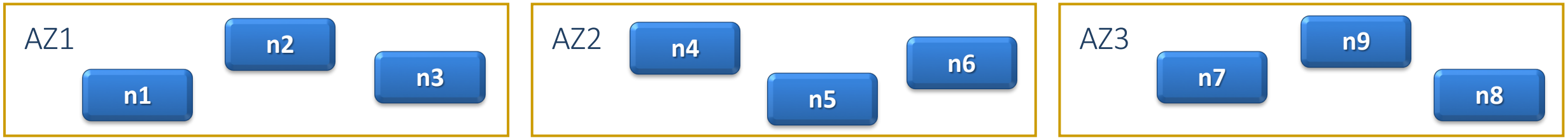
Azure Storage vNext Architecture

- scale out metadata management
- keep same consistency and atomicity



Managing Many Paxos Rings

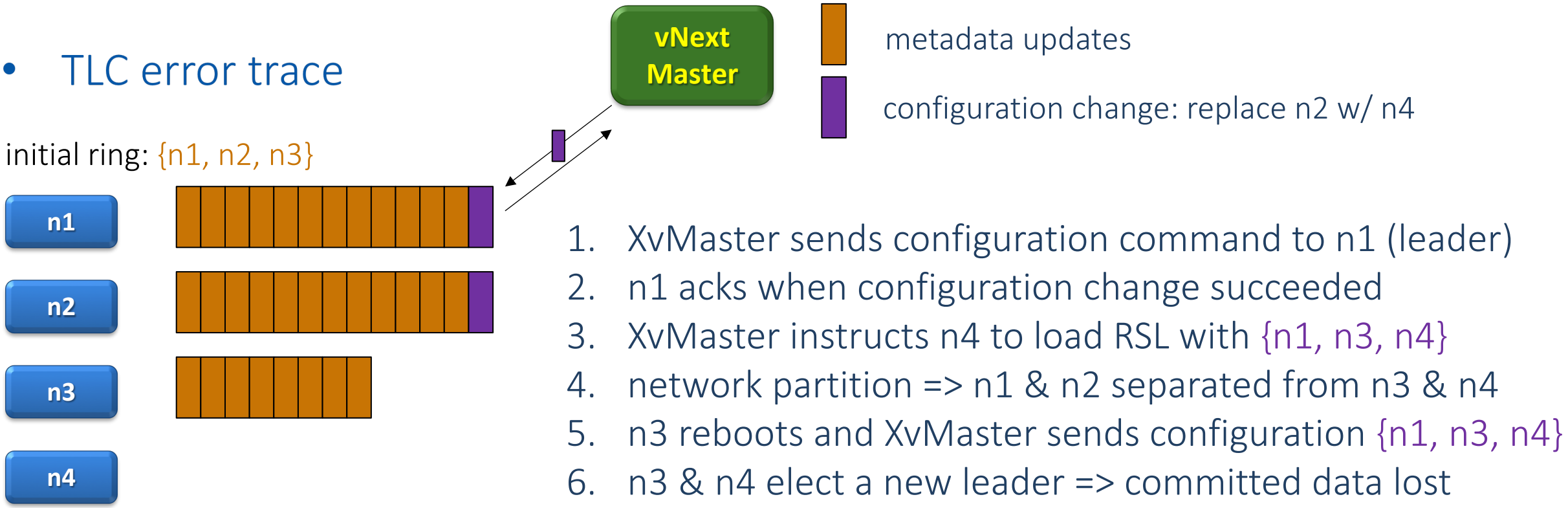
- Each shard of StreamManager and ExtentManager is a Paxos ring
- XvMaster manages all Paxos rings
  - not on critical path
  - monitors all nodes and updates the Paxos rings dynamically
- Dynamic Paxos ring management – two cases
  - Case I: node replacement based on health, clock, etc.
    - when a node is offline for long, replacing it with a new node
    - when node's clock is skewed, replacing it with a new node
  - Case II: ring resizing in multiple availability zones (AZ)
    - AZ3 failure reduces ring from 9 to 6
    - AZ3 recovery increases ring from 6 to 9



Safety Violation Discovered by TLC

- Straightforward node replacement is unsafe
  - To change Ring from {n1, n2, n3} to {n1, n3, n4}
    - XvMaster sends a configuration change command to the ring
    - XvMaster blocks until the configuration change command is confirmed by the ring
    - XvMaster then sends command and instructs n4 to load RSL engine with the new configuration

- TLC error trace



1. XvMaster sends configuration command to n1 (leader)
2. n1 acks when configuration change succeeded
3. XvMaster instructs n4 to load RSL with {n1, n3, n4}
4. network partition => n1 & n2 separated from n3 & n4
5. n3 reboots and XvMaster sends configuration {n1, n3, n4}
6. n3 & n4 elect a new leader => committed data lost

Product: Azure Storage

- People: Cheng Huang
- System: Paxos Ring Management
- Invariant violations found by TLC:
  - Quorum split on server swap
- Insights
  - Trust the log not the manager

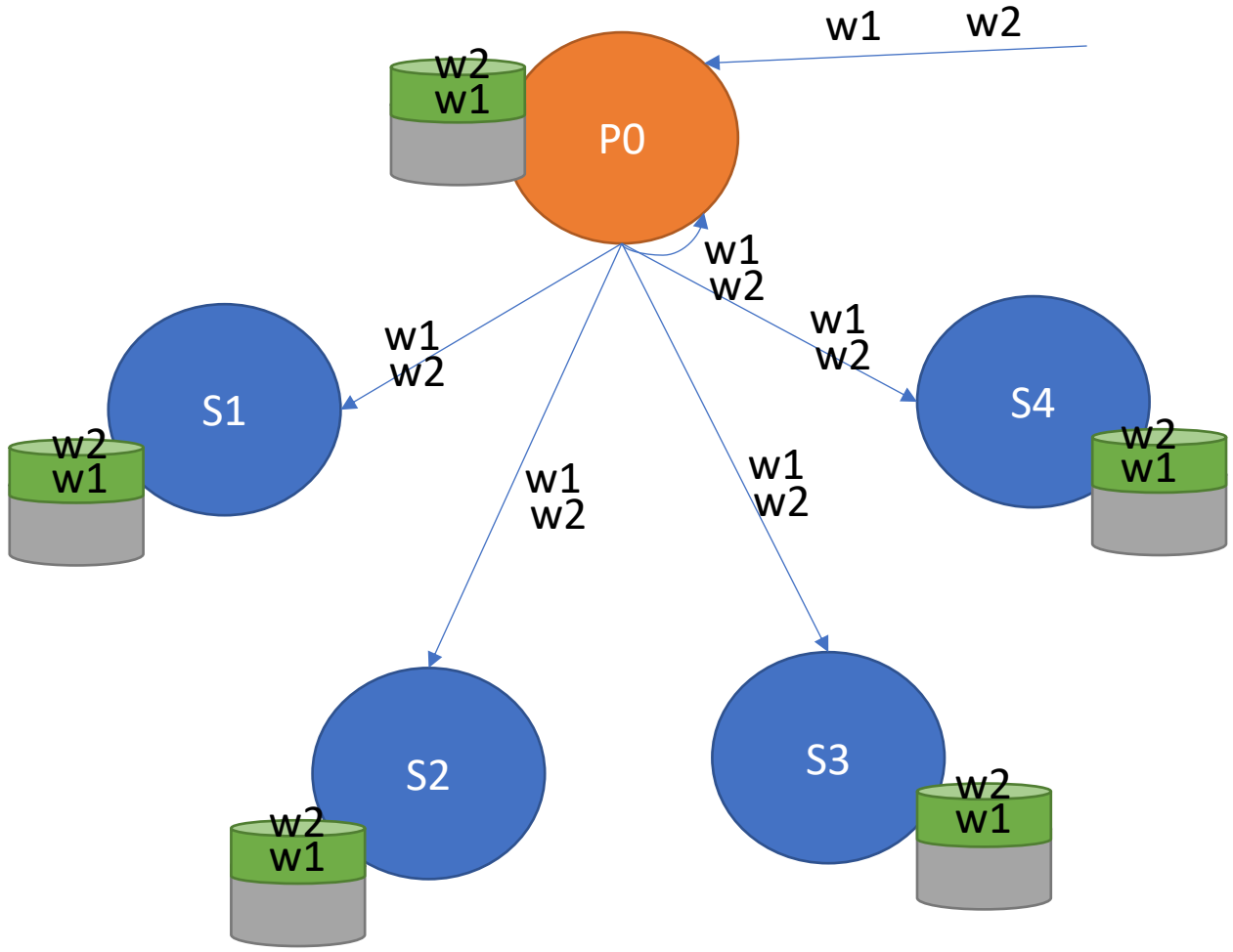
Product: Azure Networking

- People
  - Albert Greenburg
  - Luis Irun-Briz
  - Andrew Helwer
- System
  - RingMaster
    - Global replication
    - Checkpoint coordination
  - Cloud DNS
  - Record propagation
  - Distributed Load Shedding
  - MacSec encryption key rollover orchestration

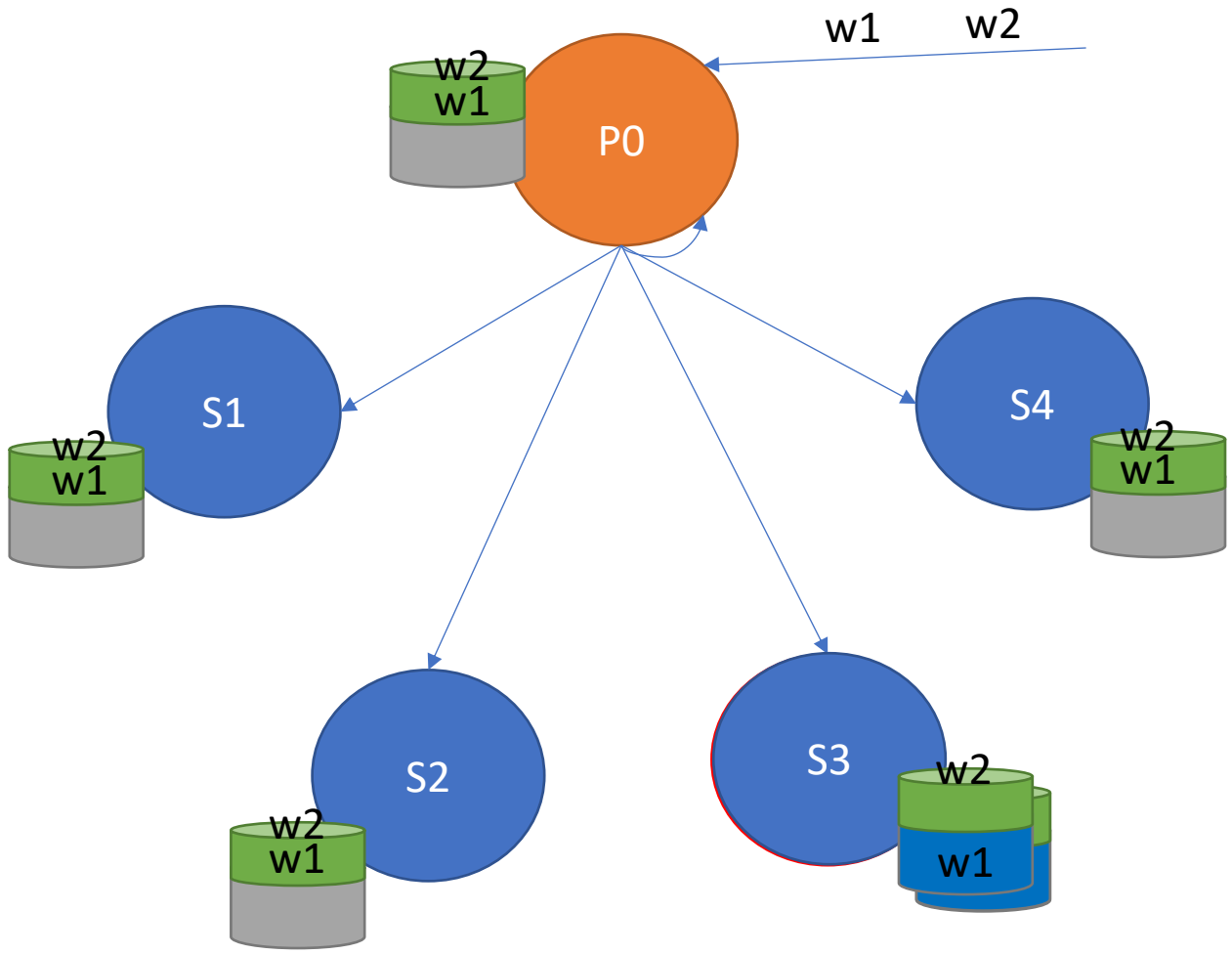
Problem Statement

- Goal: Balance checkpoints
- Guarantee a healthy checkpoint frequency,
  - Allowing for frequent checkpointing
  - To reduce restart time after failure
- Guarantee a minimum rqps across the cluster,
  - Limiting the simultaneous checkpointing
  - … which freezes updates on that replica for the duration of the checkpoint
- Avoid common pitfalls:
  - No global time
  - No locks “acquire…release”

Problem Statement

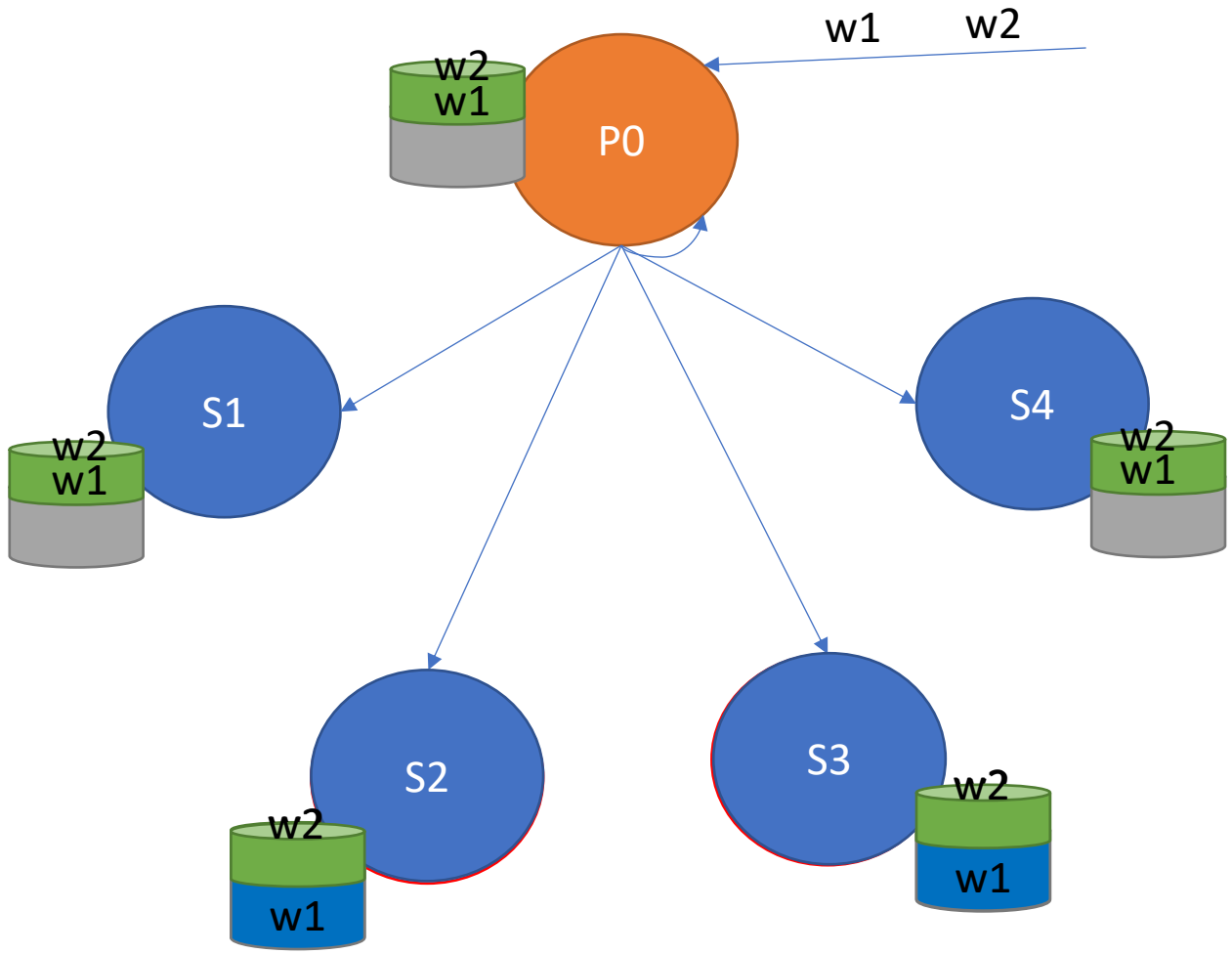


Problem Statement



100% rqps!

Problem Statement



60% rqps!

Invariants and Failure Model

- Safety Invariants:
  - the primary never takes a checkpoint
  - multiple secondary replicas never take a checkpoint concurrently
- Temporal Invariants:
  - all secondary replicas eventually take a checkpoint
  - a checkpoint always eventually completes or is aborted
- Failure Model:
  - Replicas crash, then later recover
  - Network links fail between any two replicas in the cluster, then recover
  - The rate of passage of time differing between replicas

Discarded Implementations

- Simple time slice approach
  - each replica is allocated a slice of time in an uncoordinated round-robin fashion.
  - Checkpoint time is not known. → Requires coordination to vary time slice size
  - Local time passage rate can vary between replicas → drift out of coordination
  - Inefficient: if a replica is down (or for primary) that time-slot is wasted

- Pseudo-time slice based on decree numbers instead of real time
  - The rate of new decrees can vary widely → inconsistent slices to checkpoint.
  - Still inefficient if replica is down
  - Replicas can see decrees at different times → incorrect!

Selected Implementation

- Primary Lacks Knowledge of Lease:
  - A brand new cluster with no checkpoint lease issued in its history
  - The following execution trace:
    1. Checkpoint lease given to secondary A
    2. Secondary A finishes checkpoint before timeout
    3. Secondary B dies
    4. Secondary B recovers
    5. Secondary B is rehydrated from checkpoint created by secondary A
    6. Leader dies
    7. Secondary B becomes leader
- Is it safe for the primary to issue a new lease immediately?
  - Yes, as long a replica can only become primary after executing all previous decrees.

Selected Implementation

- New Primary discards lease to itself:
  - If a replica promotes to primary and sees a checkpoint lease extended to itself.
  - Is it safe for the primary to issue a new lease immediately?

Selected Implementation

- New Primary discards lease to itself:
  - If a replica promotes to primary and sees a checkpoint lease extended to itself.
  - Is it safe for the primary to issue a new lease immediately?
- No,
    - n1 becomes leader
    - n1 send “n2 gets lease”(m1) → n1 gets it
        - n2 gets “n2 gets lease”
    - n1 dies
        - n2 becomes leader (and ignores the current lease)
        - n2 send “n3 gets lease” (m2) → n2 gets it
            - n3 gets “n3 gets lease”
        - n2 dies
        - n2 recovers and starts from scratch
        - n2 gets “n2 gets lease” (m1)
          --> n2 can take checkpoint
            - n3 gets “n2 gets lease” (m1)
            - n3 gets “n3 gets lease” (m2)
              --> n3 can take checkpoint

!!

Product: Azure Networking

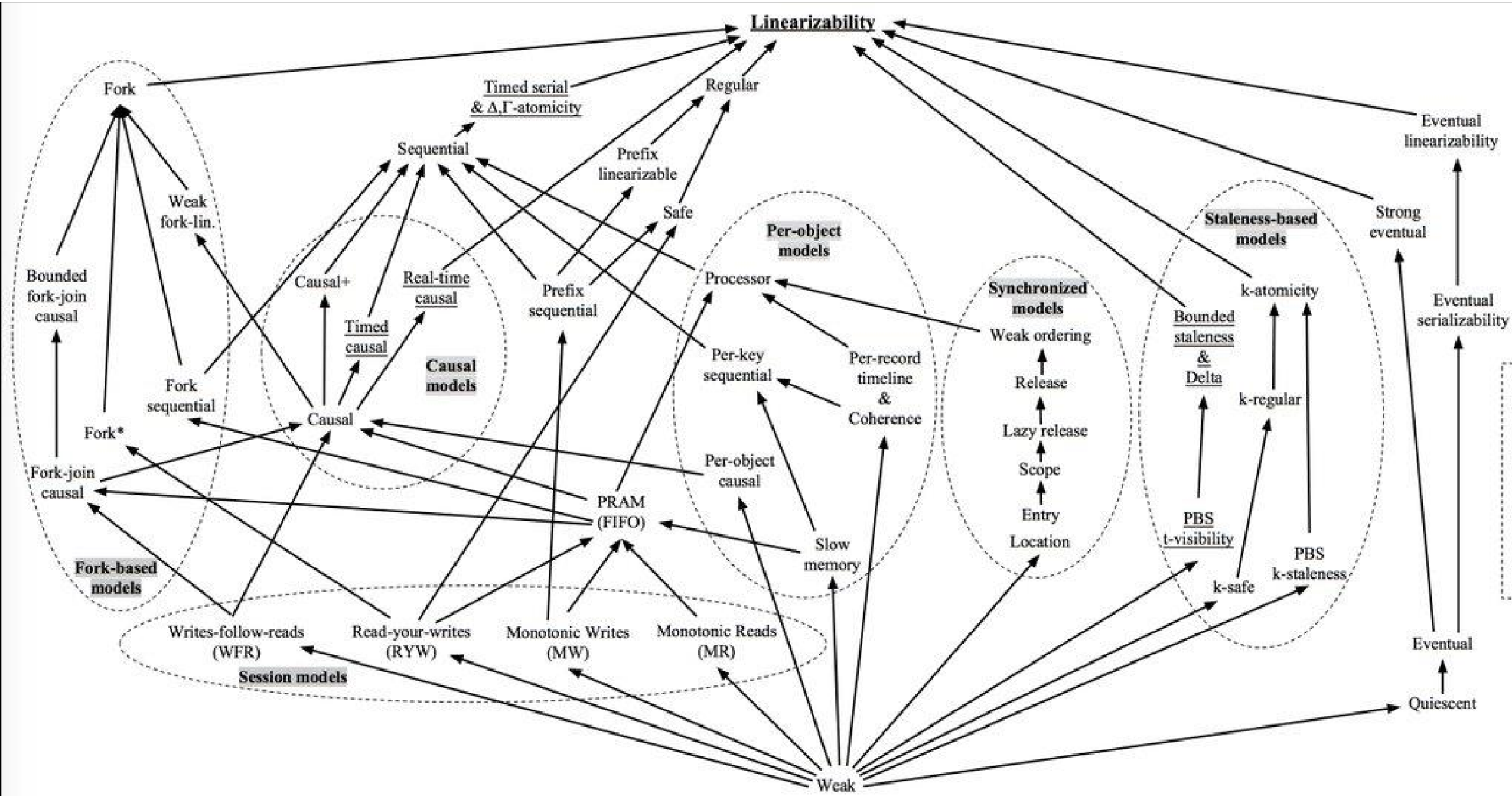
- People
  - Albert Greenburg
  - Luis Irun-Briz
  - Andrew Helwer
- System
  - RingMaster
    - Global replication
    - Checkpoint coordination
  - Cloud DNS
  - Record propagation
  - Distributed Load Shedding
  - MacSec encryption key rollover orchestration
- Invariant violations found by TLC:
  - Checkpoint coordination: A sequence of 12 events would cause two replicas to take a checkpoint at the same time
  - RingMaster Global Replication: accidentally apply a transaction twice.
- Insights

Product: Azure IoT

- People: Kapil Agarwal
- System: TBA
- Invariant violations found by TLC
  - 4
- Insights
  - TBA

Product: Cosmos DB

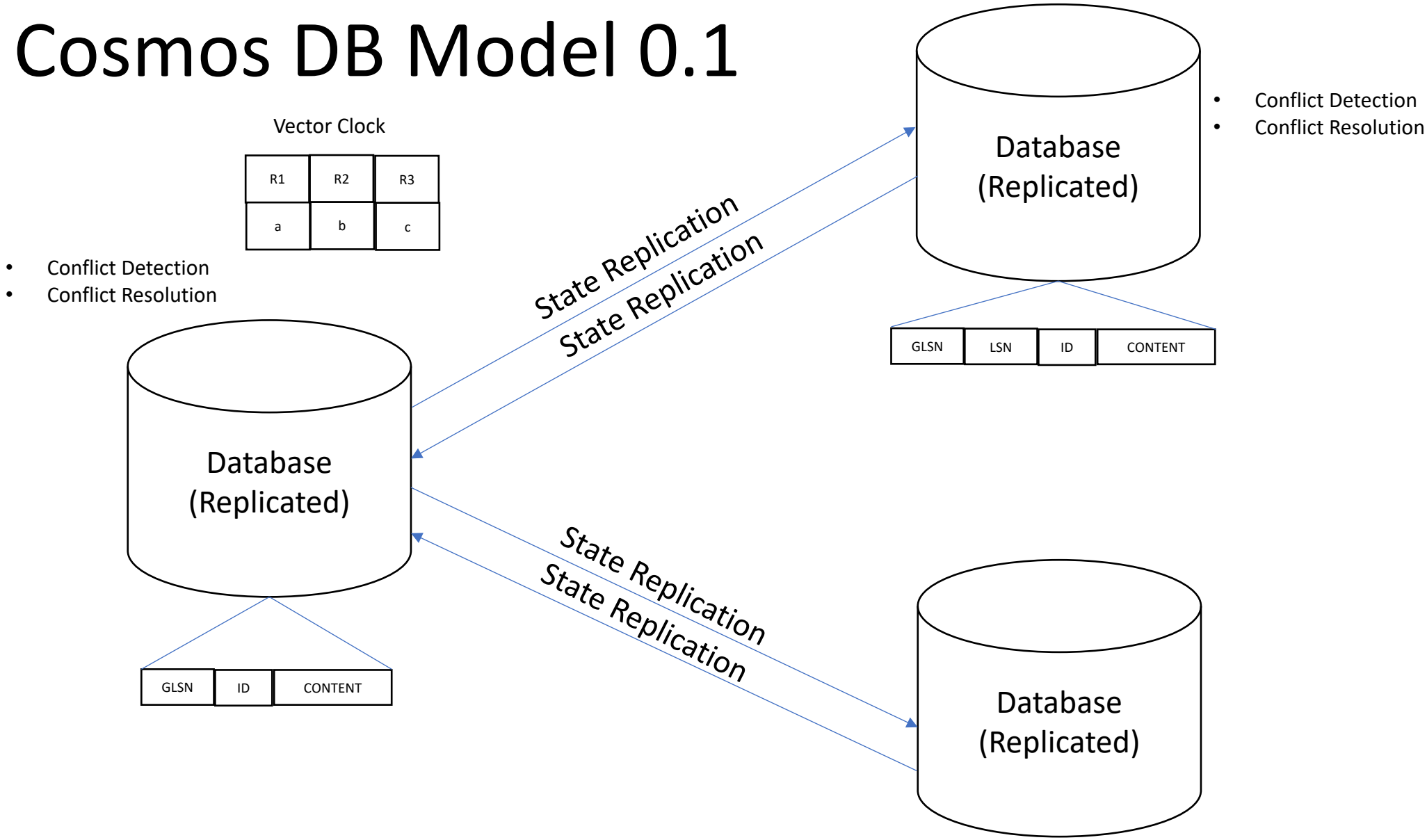
- People
  - Dharma Shukla
  - Karthik Raman
- System: Strict Convergence on Multi Master System


Linearizability
Fork
Timed serial
& Δ,Γ-atomicity
Regular
Sequential
Prefix
linearizable
Safe
Eventual
linearizability
Weak
fork-lin.
Strong
eventual
Per-object
models
Staleness-based
models
Bounded
fork-join
causal
Causal+
Real-time
causal
Prefix
sequential
Processor
k-atomicity
Synchronized
models
Eventual
serializability
Timed
causal
Causal
models
Weak ordering
Bounded
staleness
&
Delta
Per-key
sequential
Per-record
timeline
&
Coherence
Release
Fork
sequential
k-regular
Fork*
Causal
Lazy release
Scope
Entry
Location
Per-object
causal
Fork-join
causal
PRAM
(FIFO)
PBS
t-visibility
PBS
k-staleness
Slow
memory
k-safe
Fork-based
models
Writes-follow-reads
(WFR)
Read-your-writes
(RYW)
Monotonic Writes
(MW)
Monotonic Reads
(MR)
Session models
Eventual
Quiescent
Weak

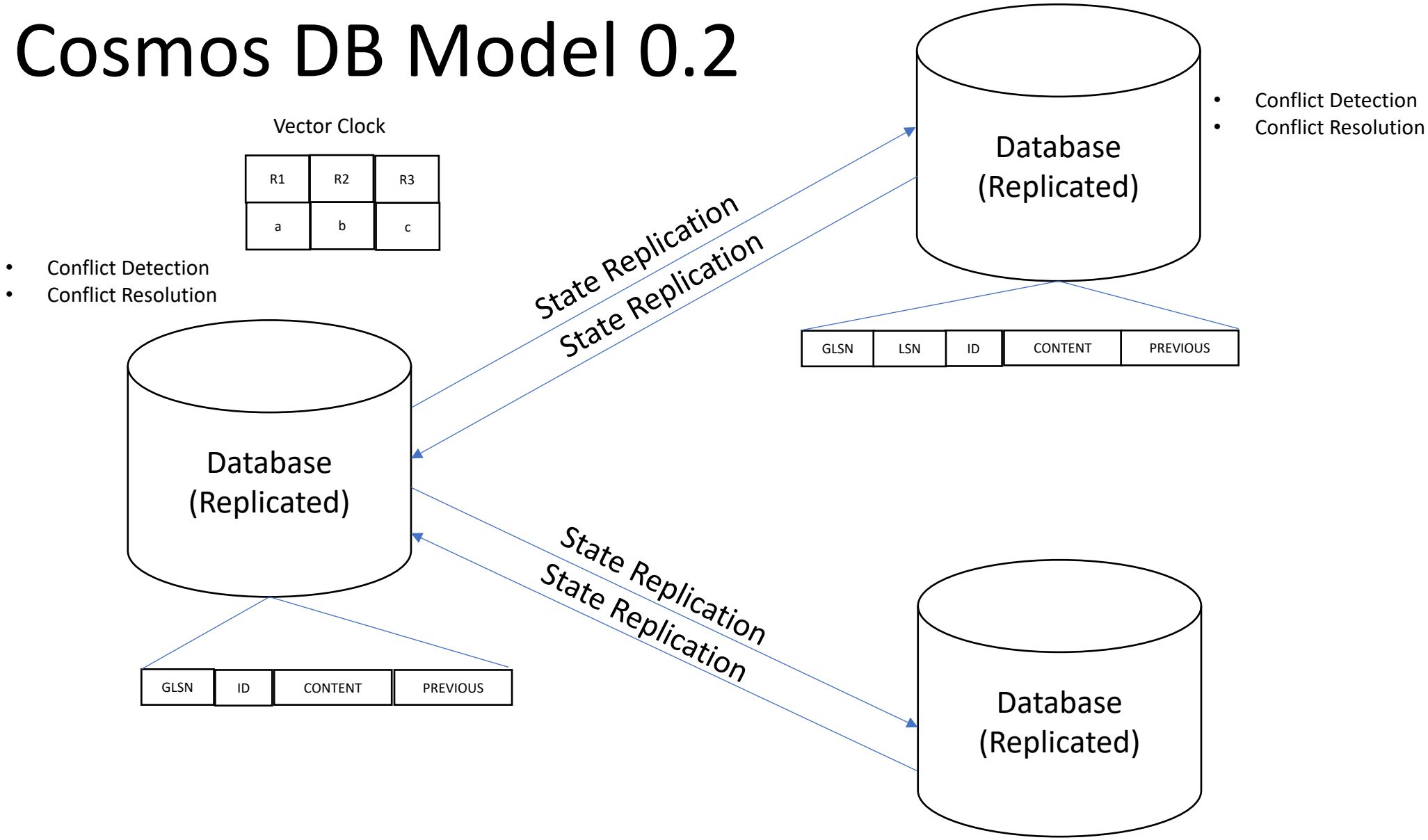
5 Consistency Levels

- Strong
- Bounded Staleness
- Eventual Consistency
- Session
- Consistent Prefix

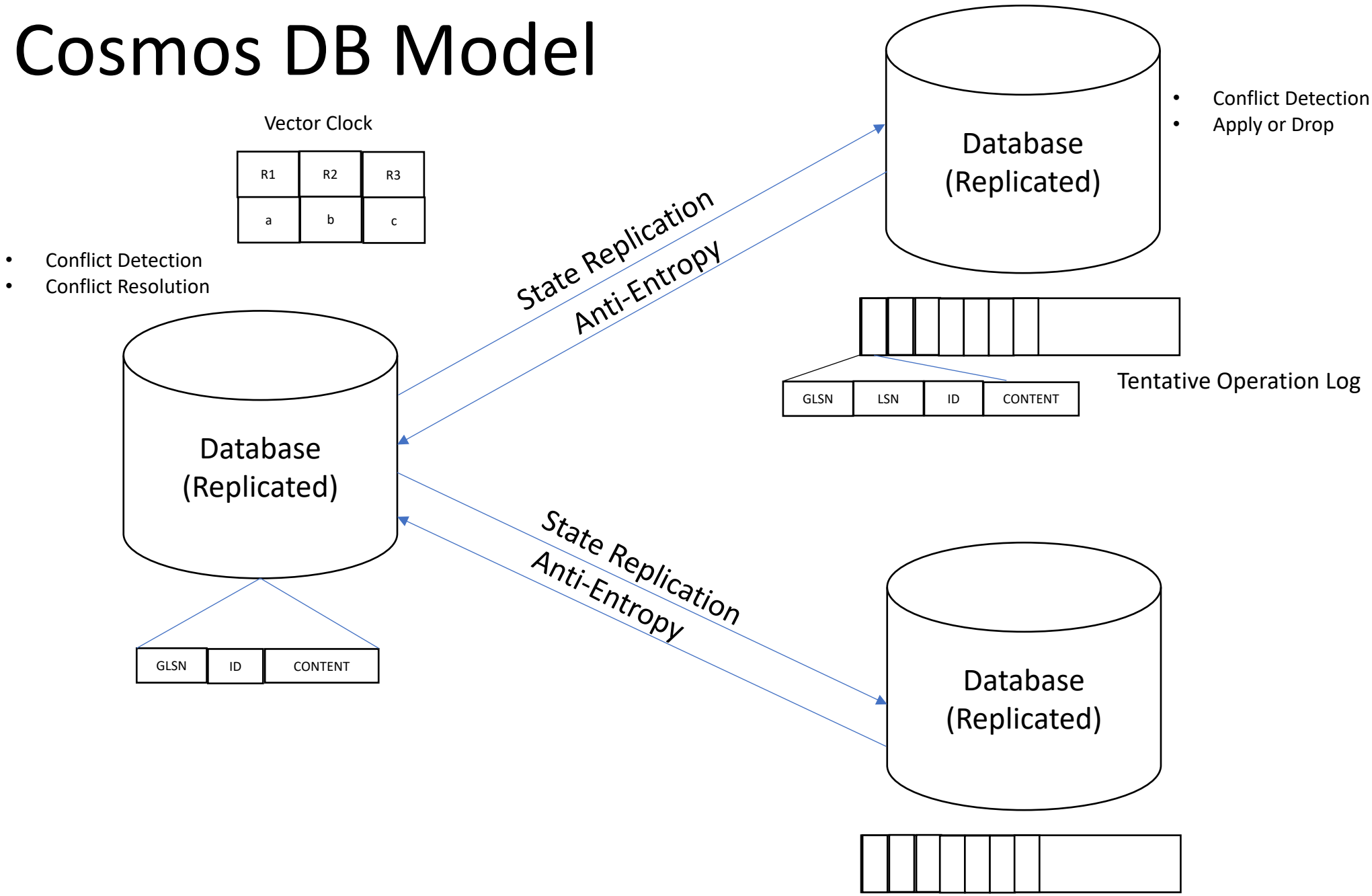
Cosmos DB Model 0.1



Cosmos DB Model 0.2



Cosmos DB Model



Product: Cosmos DB

- People
  - Dharma Shukla
  - Karthik Raman
- System: Strict Convergence on Multi Master System
- Invariant violations found by TLC
  - 0.1
    - Replicas diverge due to lack of previous image for performing correct undo.
    - Due to ID conflicts, multi-record correctness issue could happen.
  - 0.2
    - State replication was not enough.
      - Missing intermediate version results in divergence.
    - Multi-record conflicts could result in complex list of previous image map.
- Insights

Observations

- Syntax is not the problem
- Starting in isolation is a difficult
- TLC: Anti Runtime
- Benefits the Author

The End