



SalesforceDB

Salesforce's Cloud-Native OLTP RDBMS
Powering the Salesforce Platform

Jamie Martin
Distinguished Engineer
jameison.martin@salesforce.com

<https://sfdc.co/transformed>



SalesforceDB and Lessons Learned



SalesforceDB is battle-tested and widely in production today

Today we will discuss lessons learned

- Scaling out
- Preventing, mitigating, and remediating corruptions
- In production over 5 years
- Over 5 million tenants
- 1,000s of database instances
- Dozens of regions
- 50+ Petabytes of data
- 6+ trillion SQL statements per day

Read our papers to learn more

Salesforce Database: A Cloud-Native Multi-Tenant OLTP Database

Atish Agrawal, Vaibhav Arora, Gary Baker,
Milena Bergant, Benjamin Busjaeger, Subho Chatterjee, Terry Chong, Roy Chowdhuri, Doug Doole,
Haiyan Du, Thomas Fanghaenel, Pat Helland, JV Jujjuri, Jim Mace, Ashish Mahajan, Jamie Martin,
Nicolas Michael, Ayman Mobarak, Steven Raspudic, Randy Spalten, Simon Wong, Nat Wyatt
Salesforce

ABSTRACT

Salesforce Database (SalesforceDB) is a multi-tenant relational database built by Salesforce to power the transactional workload of its Customer Relationship Management applications and platform. It is the system of record and an integral part of the Salesforce stack. Using its own database allows Salesforce to innovate from the data-

1 INTRODUCTION

Transactional workloads are a fundamental part of Salesforce's applications. The transition to a modern public cloud infrastructure [9] with increased growth and scale across multiple regions and mission critical applications requires increased resiliency and scalability throughout the stack. In particular, the database layer needs to pro-

[VLDB 2026](#)

A Multi-tenant Relational OLTP Database at Salesforce

Vaibhav Arora Salesforce vaibhav.arora@salesforce.com	Subho Chatterjee Salesforce schatterjee@salesforce.com	Terry Chong Salesforce tchong@salesforce.com
Thomas Fanghaenel Salesforce tfanghaenel@salesforce.com	Pat Helland Salesforce phelland@salesforce.com	Jamie Martin Salesforce jameison.martin@salesforce.com
Kaushal Mittal Salesforce kaushal.mittal@salesforce.com	Nat Wyatt Salesforce nwyatt@salesforce.com	

Abstract

Salesforce Database (SalesforceDB) is a relational OLTP database that directly supports the Salesforce multi-tenant application model

More details on this journey will be published elsewhere; this paper illustrates some techniques that were used to get good OLTP read performance on an LSM-based database design in a commer-

[CIDR 2026](#)

Architecture Overview



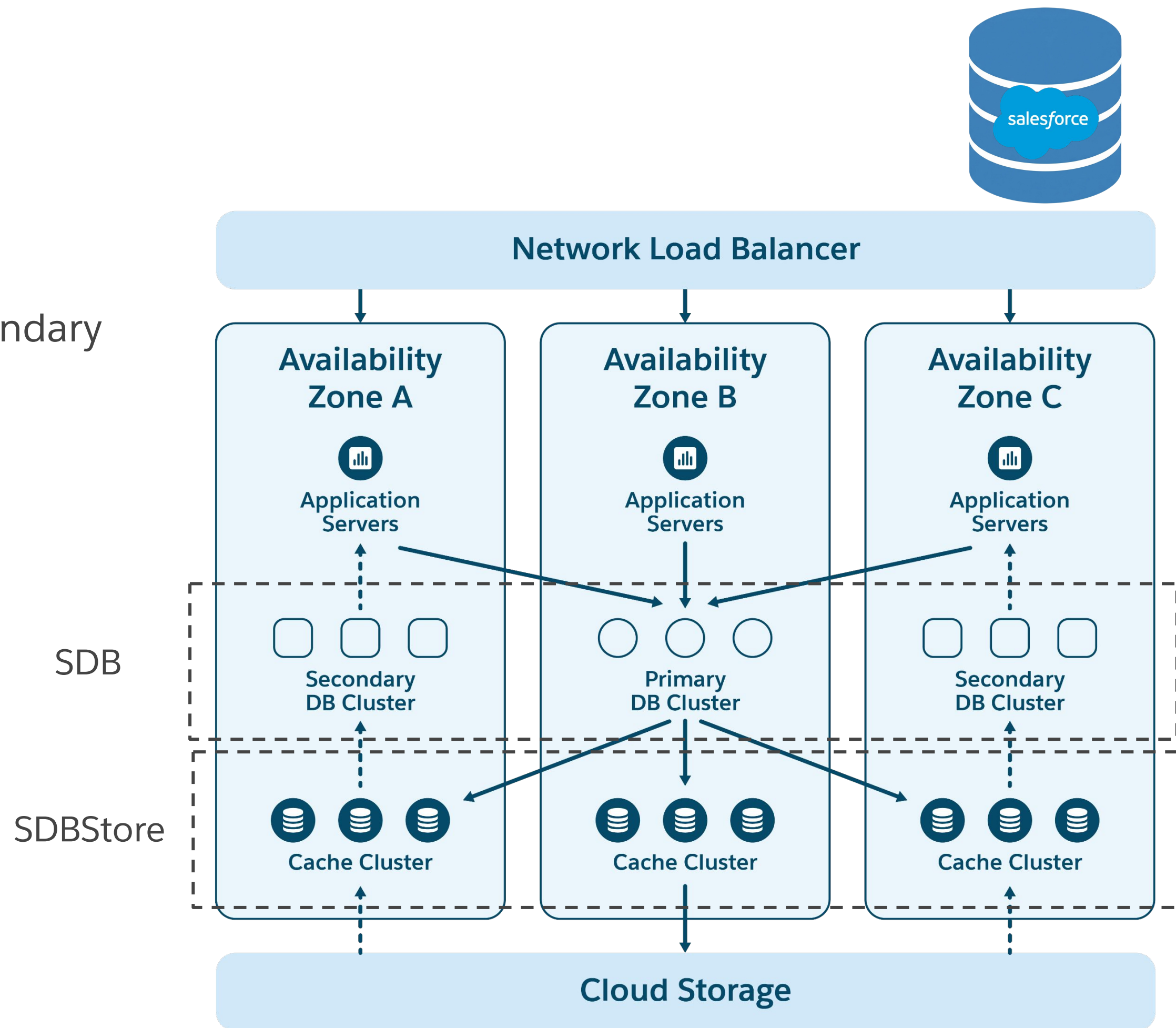
System Topology

Primary service tiers

- Application servers
- SQL compute clusters, primary/secondary
- Cache clusters over cloud storage

Service properties

- Multi-tenanted
- Horizontally scalable
- Fault tolerance of AZ+1
- Cross-region async replication





LSM: Writes, Reads and Storage

Writes are always KV inserts to LSM

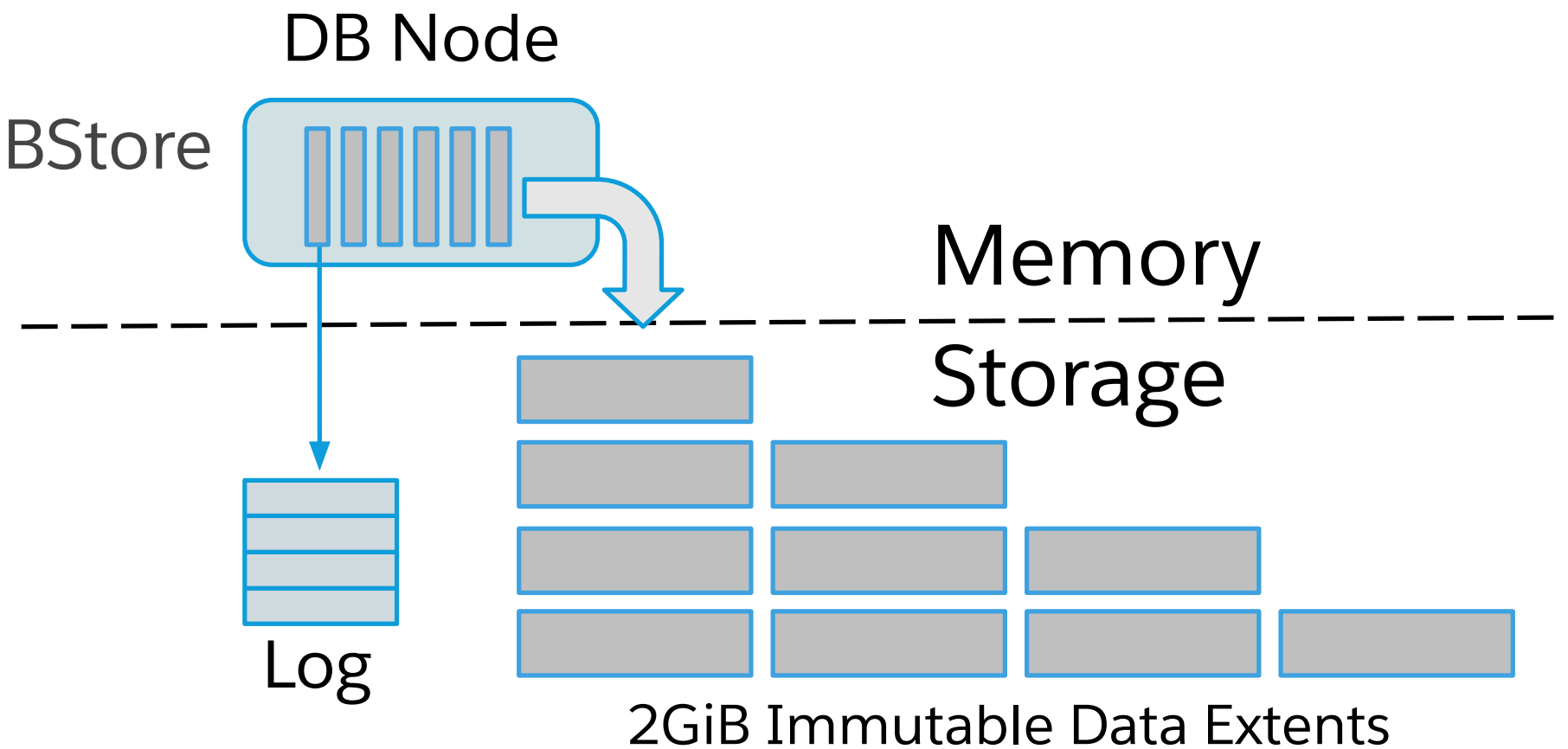
- Buffered in memory in Memstore
- Logged to append-only transaction log in SDBStore
 - quorum write, quorum read on recovery fence
 - writeback cache to cloud storage

Async data writes to 2GiB Data Extents

- Periodic flushes of Memstore to storage
- Data is reorganized by compaction

Reads span Memstore and storage

Data is always immutable



Tenants in Immutable KV LSM



Tenant organized records

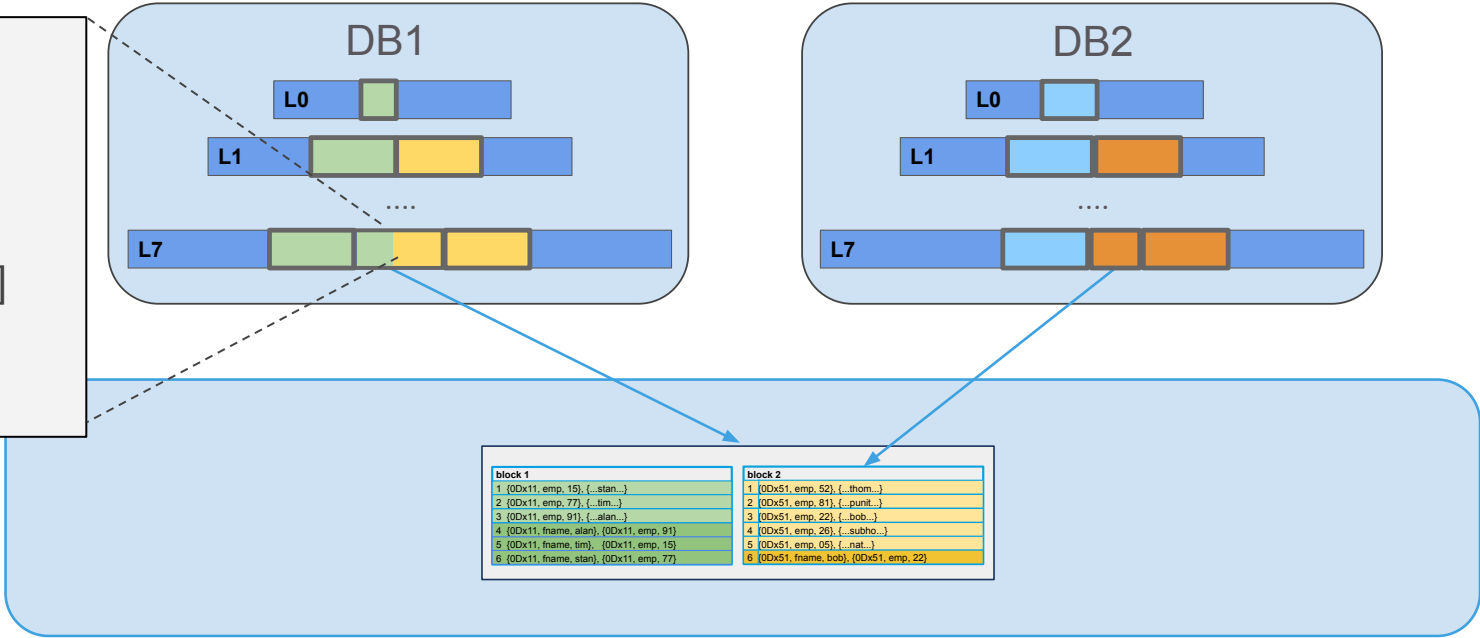
- Ordered by {TenantId, RelationId, Key}
- Forms unique key in "galactic" keyspace
- 2GiB multi-tenanted data Extents
- Efficient storage, scans, encryption

SalesforceDB Data Extent		
block 1		
1	{0Dx11, emp, 15}, {...stan...}	
2	{0Dx11, emp, 77}, {...tim...}	
3	{0Dx11, emp, 91}, {...alan...}	
4	{0Dx11, fname, alan}, {0Dx11, emp, 91}	
5	{0Dx11, fname, tim}, {0Dx11, emp, 15}	
6	{0Dx11, fname, stan}, {0Dx11, emp, 77}	
block 2		
1	{0Dx51, emp, 52}, {...thom...}	
2	{0Dx51, emp, 81}, {...punit...}	
3	{0Dx51, emp, 22}, {...bob...}	
4	{0Dx51, emp, 26}, {...subho...}	
5	{0Dx51, emp, 05}, {...nat...}	
6	{0Dx51, fname, bob}, {0Dx51, emp, 22}	

Extents accessed through virtualization layer

- Extent References scope a DB's view of an Extent
- Efficient export/import

Extent Reference
ExtentId: {123e4567}
Key Range
[{0Dx11 emp, 15},
{0Dx51 fname, bob}]
Time Range
[020, 050]



Multi-Tenant Compute

Multi-Tenant Shared Storage



Scaling Out

Lessons Learned

Scaling Requirements

Goal: Scale DB CPU for big customers

Scaling reads alone is not enough

- relatively high proportion of traffic is writes
- reads and writes are intermingled

Long running transactions

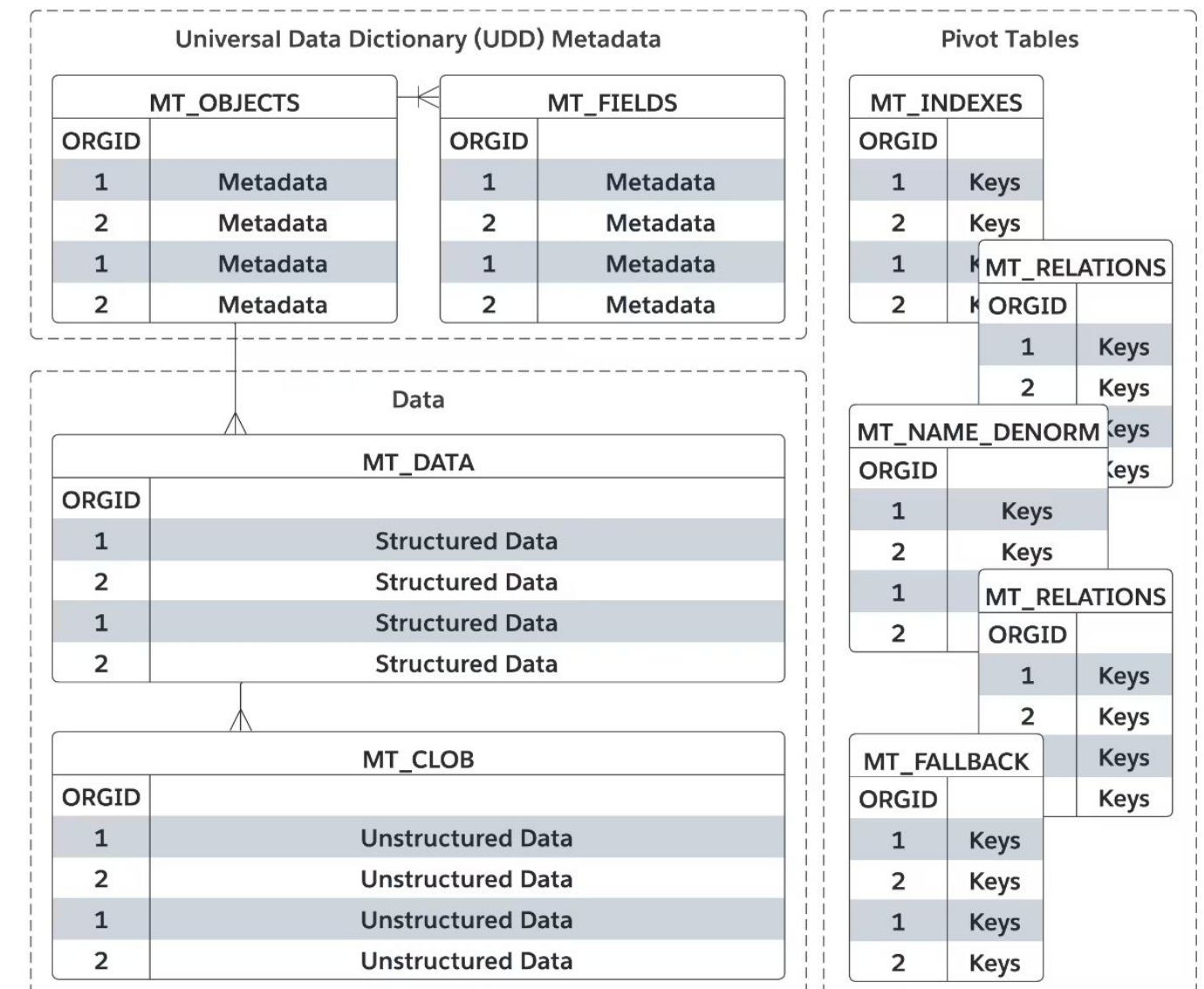
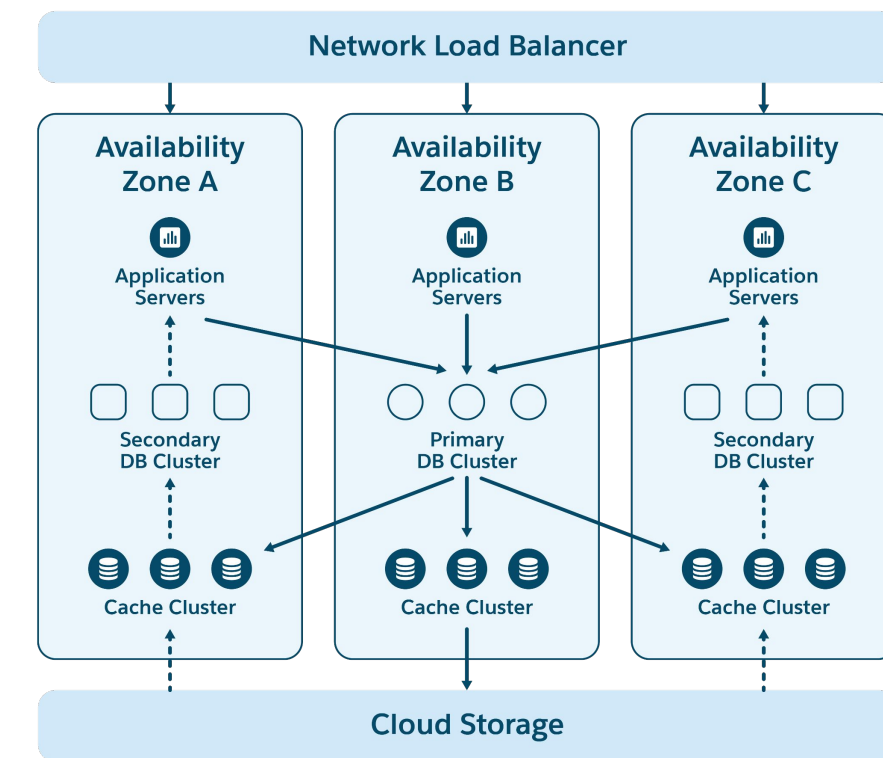
Hot keyspaces

Poorly partitioned intra-tenant schema

Full SQL contract including lots of server side SQL

External consistency

On commodity hardware



Complex schema in Salesforce platform

Cloud-Native OLTP Scale-Out Design Space



SalesforceDB scales out writes on a shared-storage LSM

	Shared-Nothing	Shared-Storage
Page-Based	 Vitess 	 Aurora  
Log-Structured (LSM)	 Cloud Spanner 	

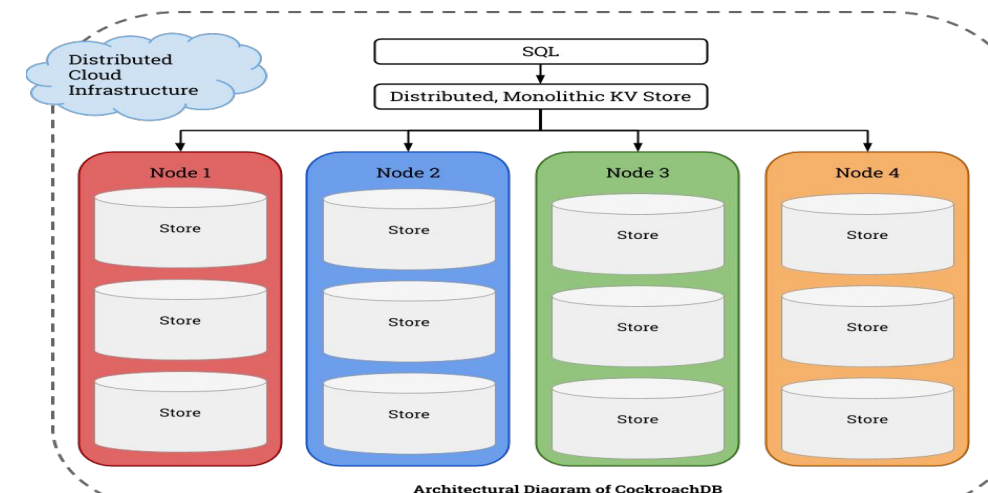
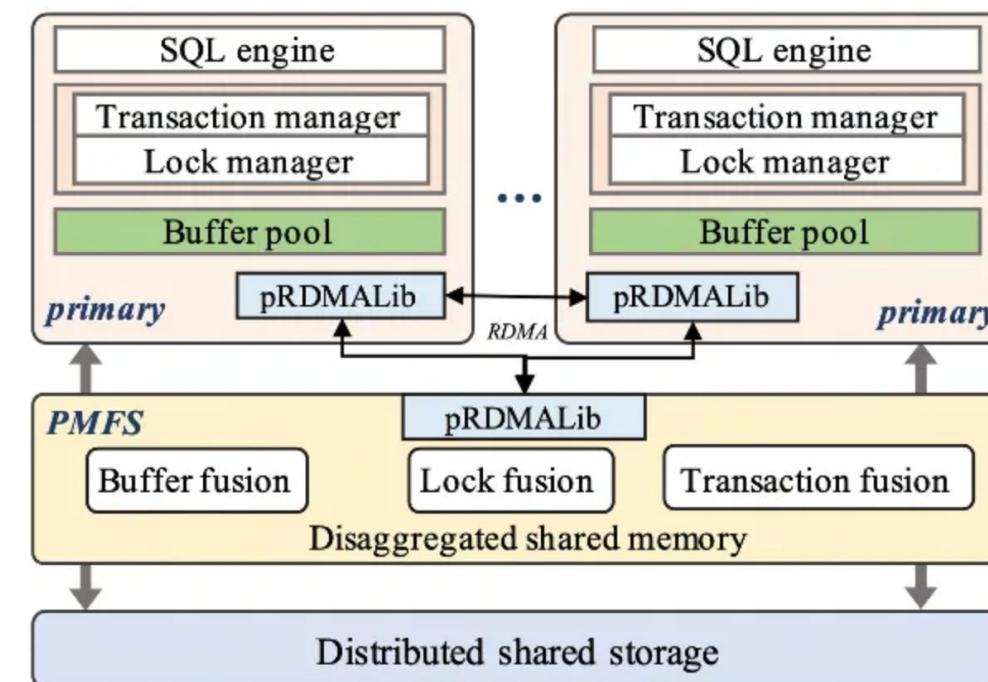
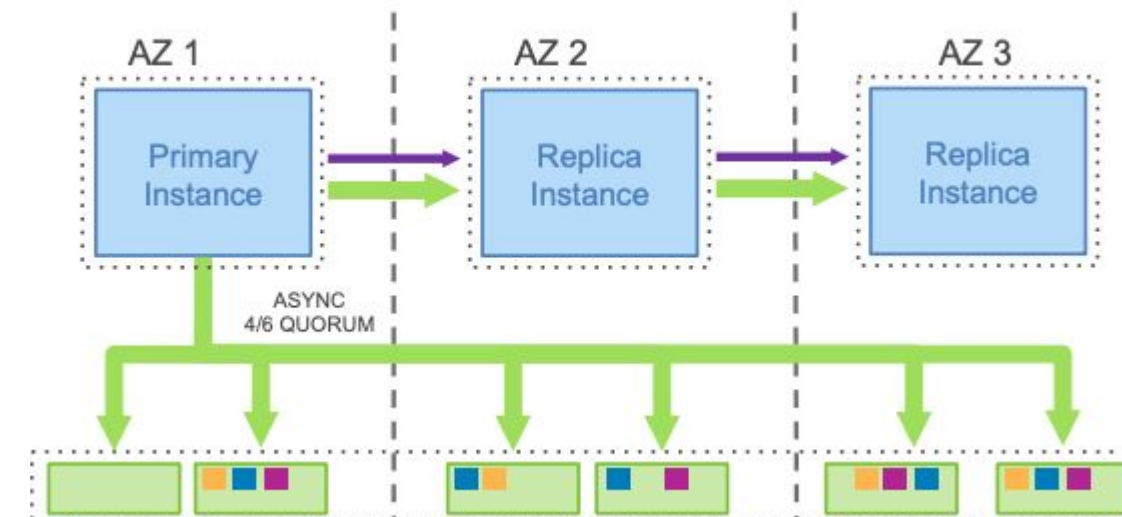
Canonical OLTP Scale-Out

Page-Based Shared-Storage

- Single-Writer Multi-Reader
 - lagging reads is straightforward
 - strongly consistent reads typically requires redo log replication to keep buffer cache consistent and current pageLSN
- Multi-Writer
 - requires page-level coordination
 - typically relies on specialized hardware

Shared-Nothing

- Works best with partition locality
- Difficult to support all SQL



[AWS Aurora](#)

SQL Server

AlloyDB

Taurus

PolarDB

Neon

[PolarDB-MP](#)

Taurus-MM

Oracle

IBM pureScale

[CockroachDB](#)

Spanner

Shared-Storage Multi-Writer LSM

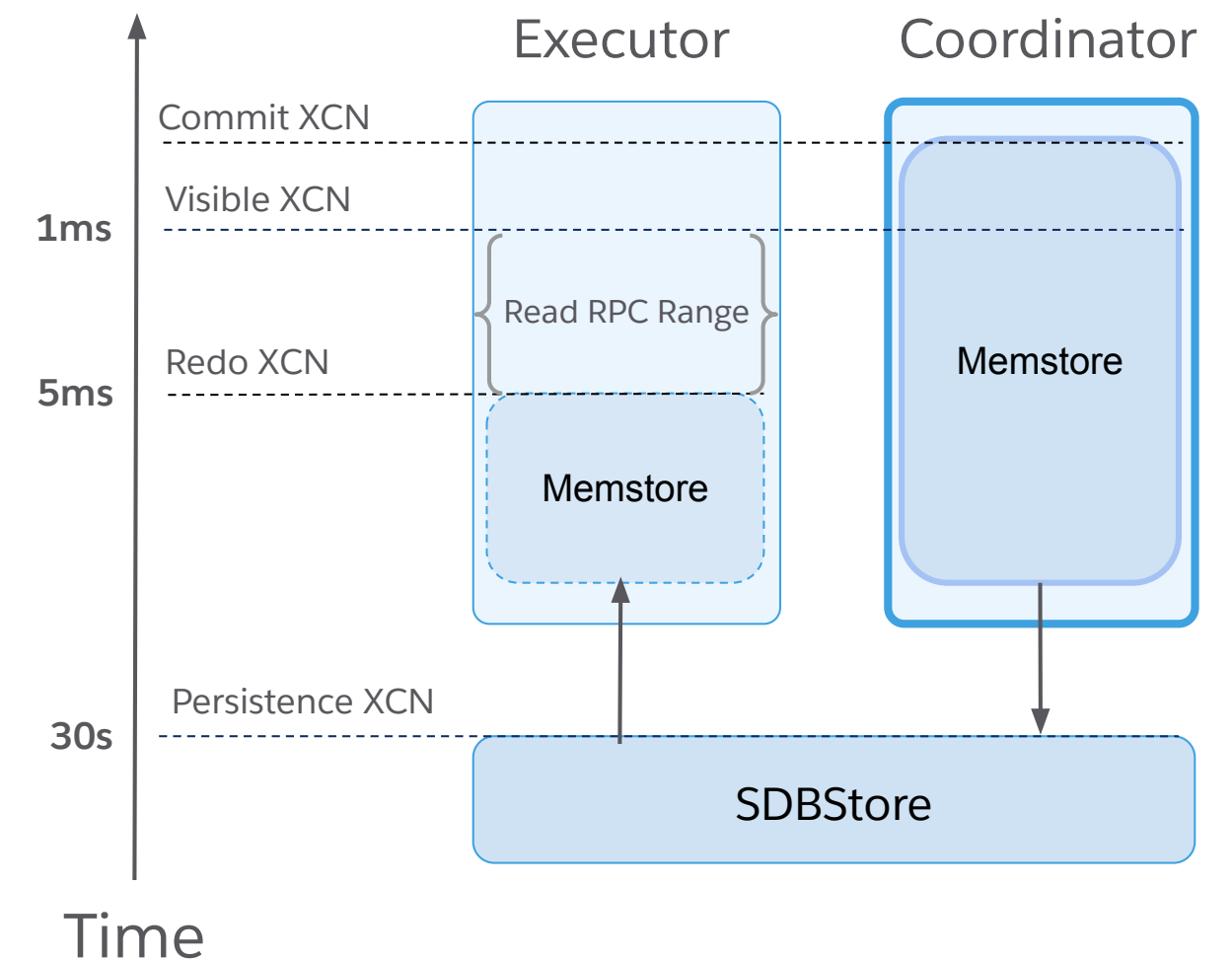
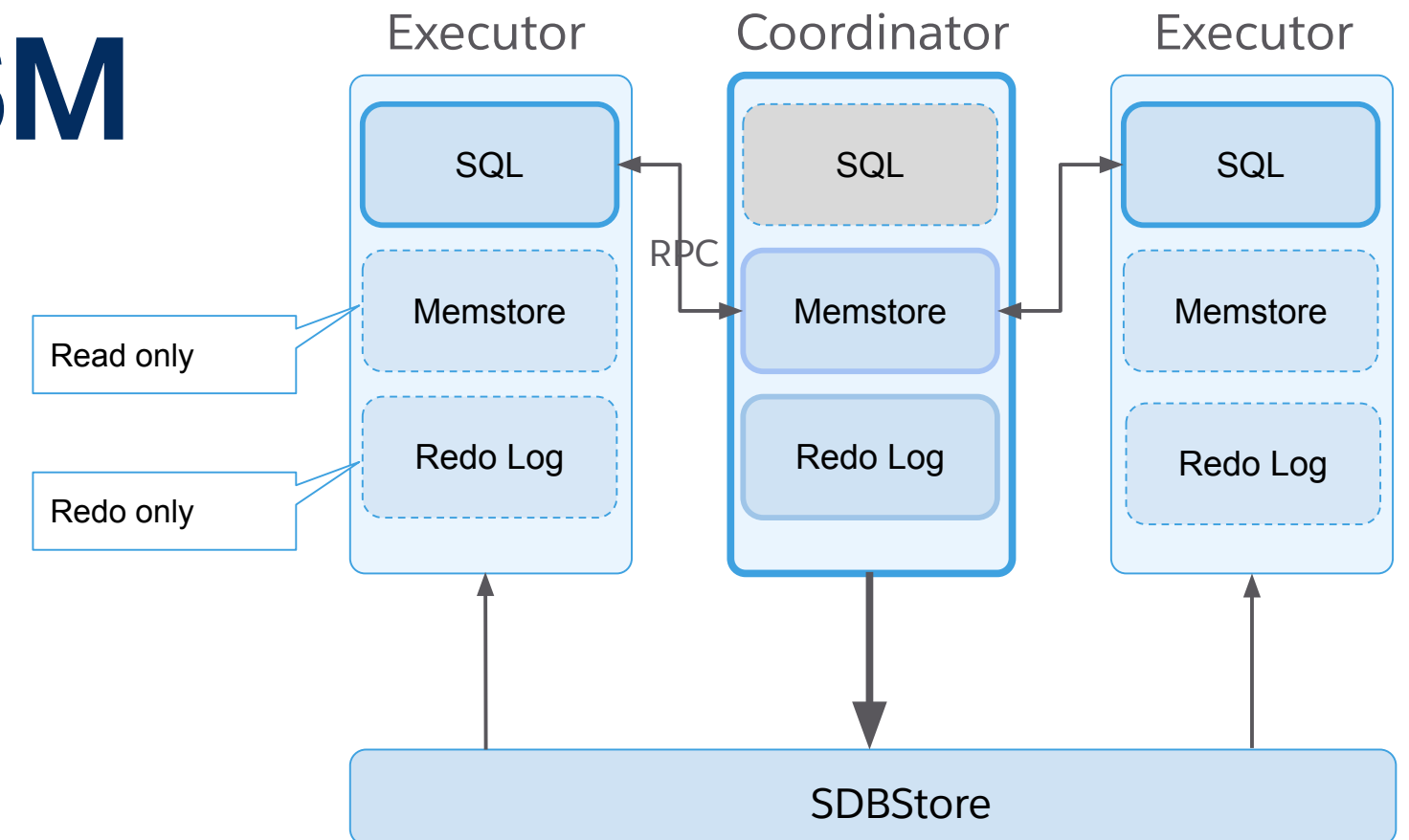
Primary challenge is scaling out Memstore

Writes

- SQL processed on Executor
- Record writes RPCed to Coordinator

Snapshot reads

- Acquire timestamp locally
- Most reads through Block Cache over immutable storage
- Avoid submitting reads of recent data to Coordinator
 - filters
 - leverage local Memstore populated by log tailing



Scales DB CPU Quite Well

Has scaled quite well, up to 20 nodes

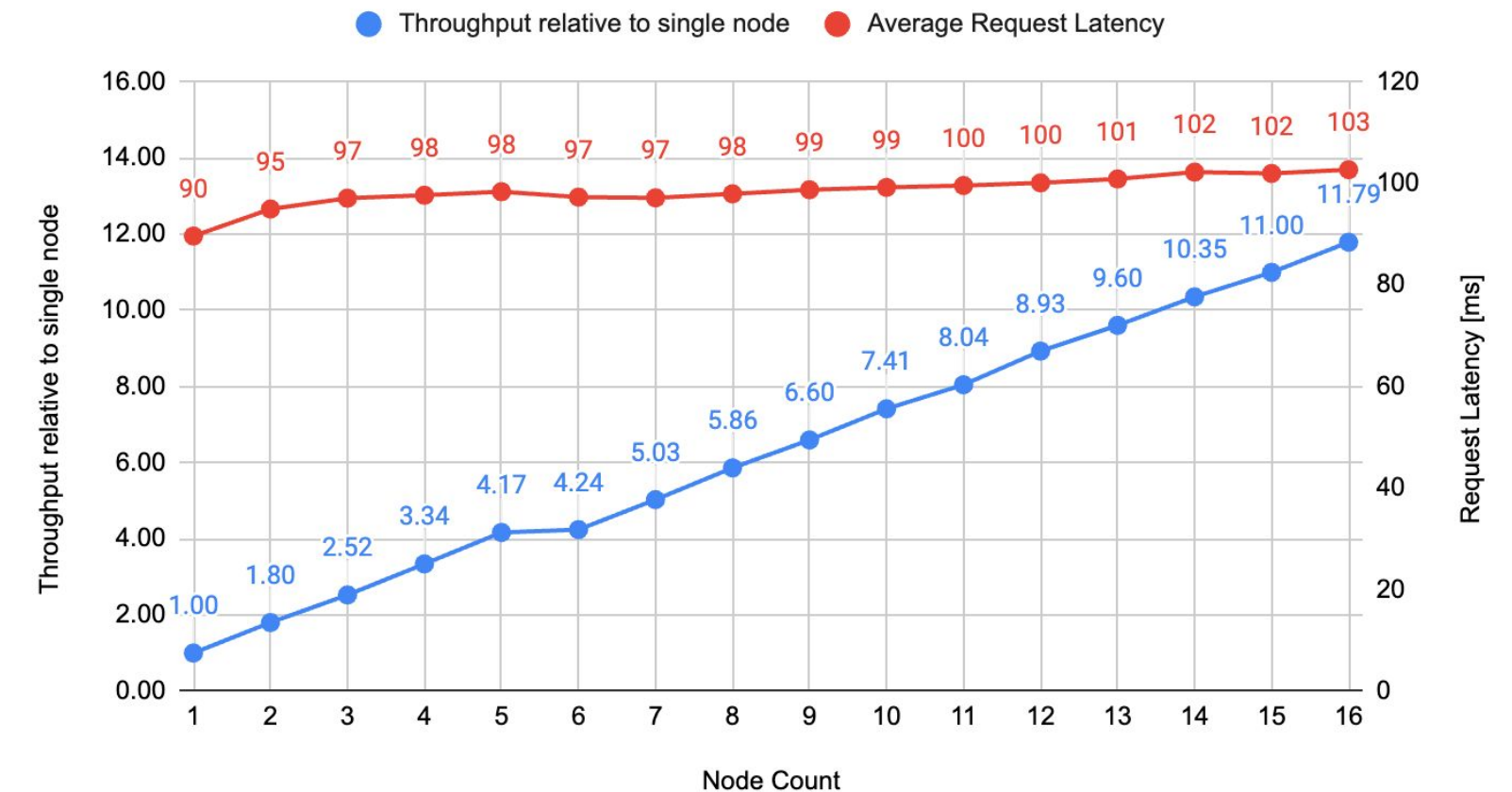
Most DB work requires little coordination

- Most work is processed locally at the Executor
- Even a SQL INSERT runs mostly locally
- Only a small sliver of work is pushed to Coordinator

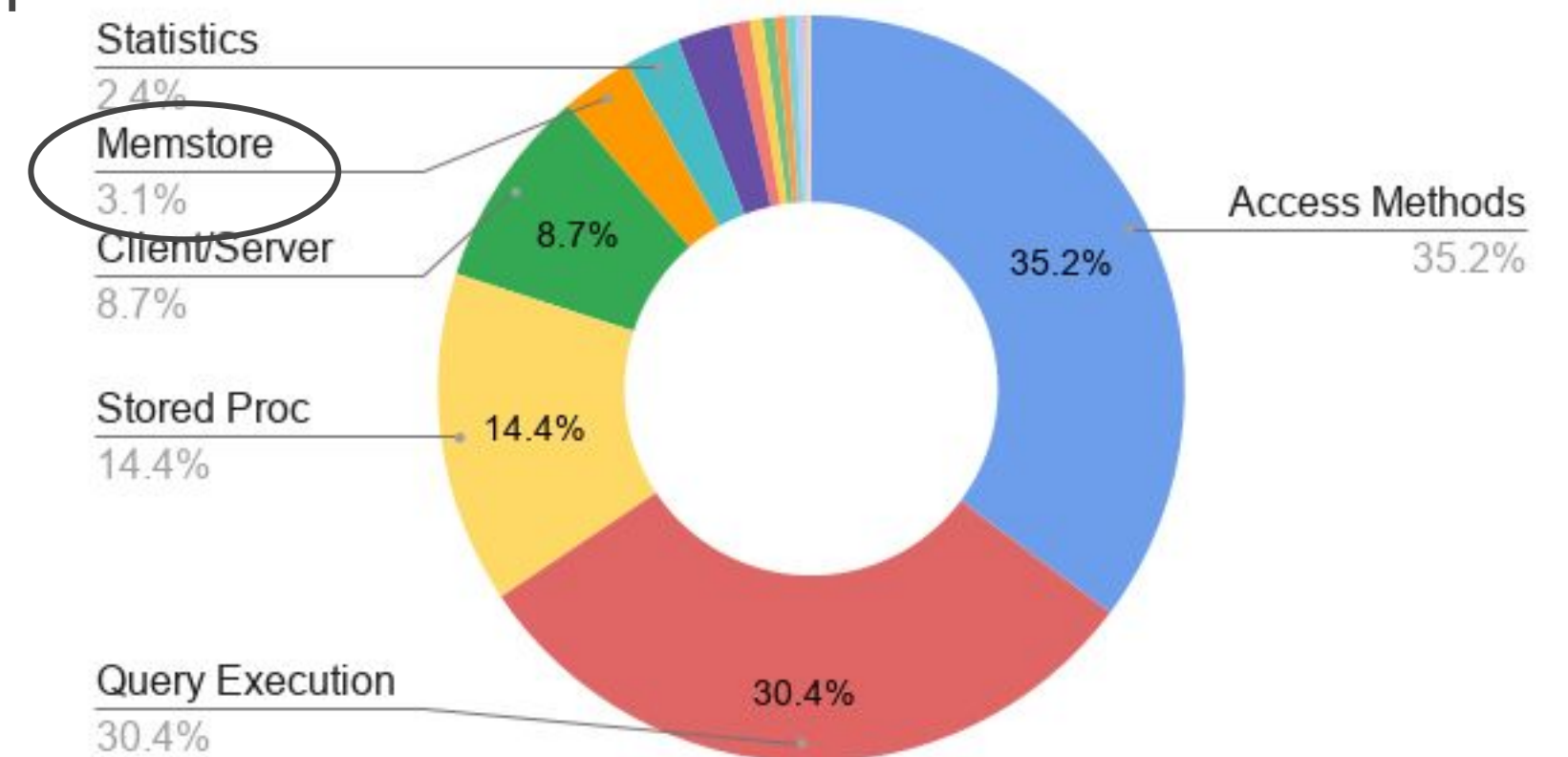
Customer impact has been limited to outlier latency due to high proportion of RPCs - multi-row DML

- Batching related RPCs
- Asynchronous RPCs
- Reading uncommitted writes locally

SDB Compute Scaling



DB CPU Time by Component



Successfully Rolling Out Multi-Node Clusters



Multi-node awareness is isolated in few subsystems

- Query execution is unchanged
 - all scale-out execution is under access methods
- No impact on Query Optimizer

In-place config change: compatible storage and ops

- Automated flip between single-node or multi-node
- Equivalent to adjusting the cluster size

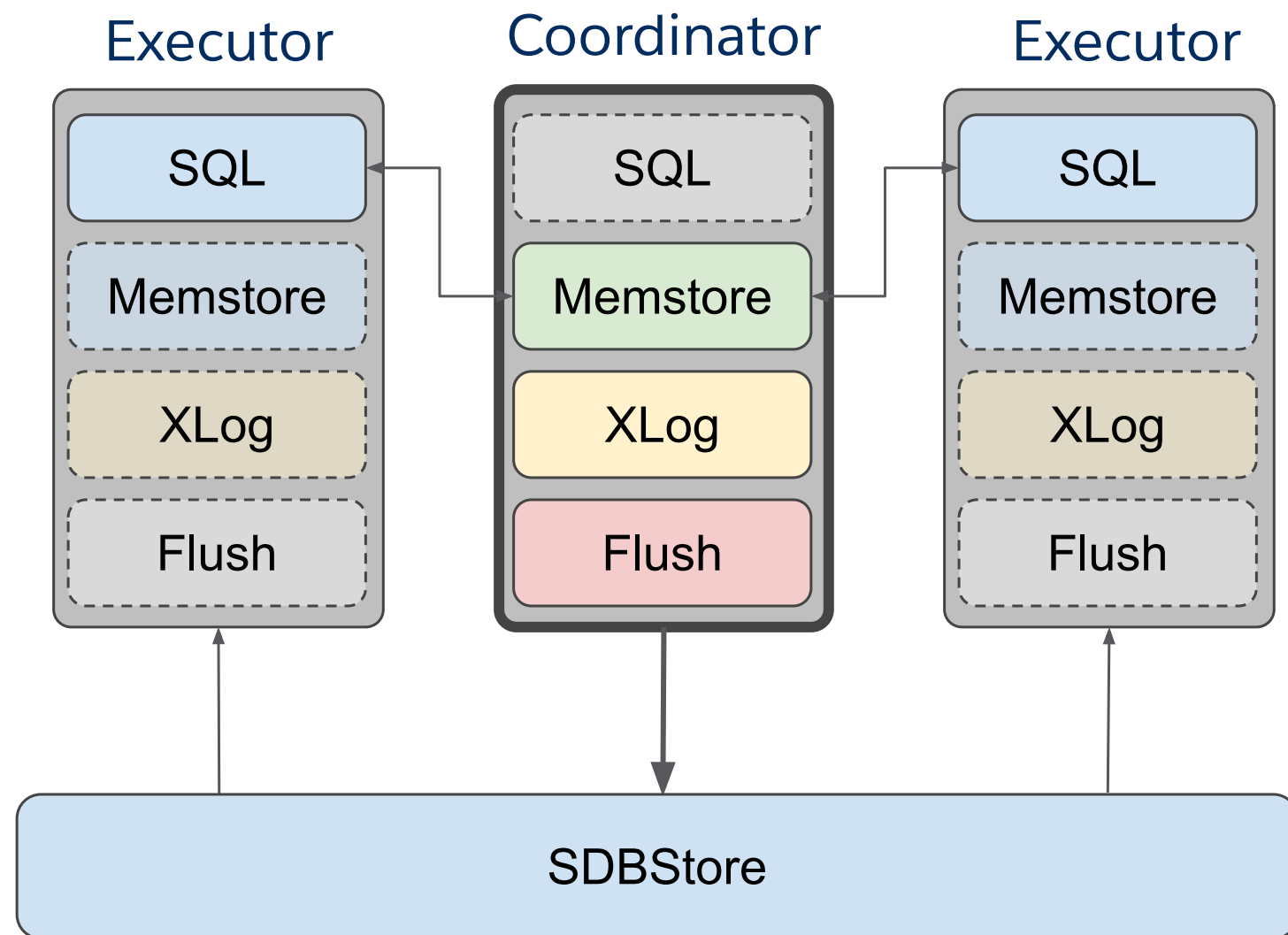


The Next Evolution

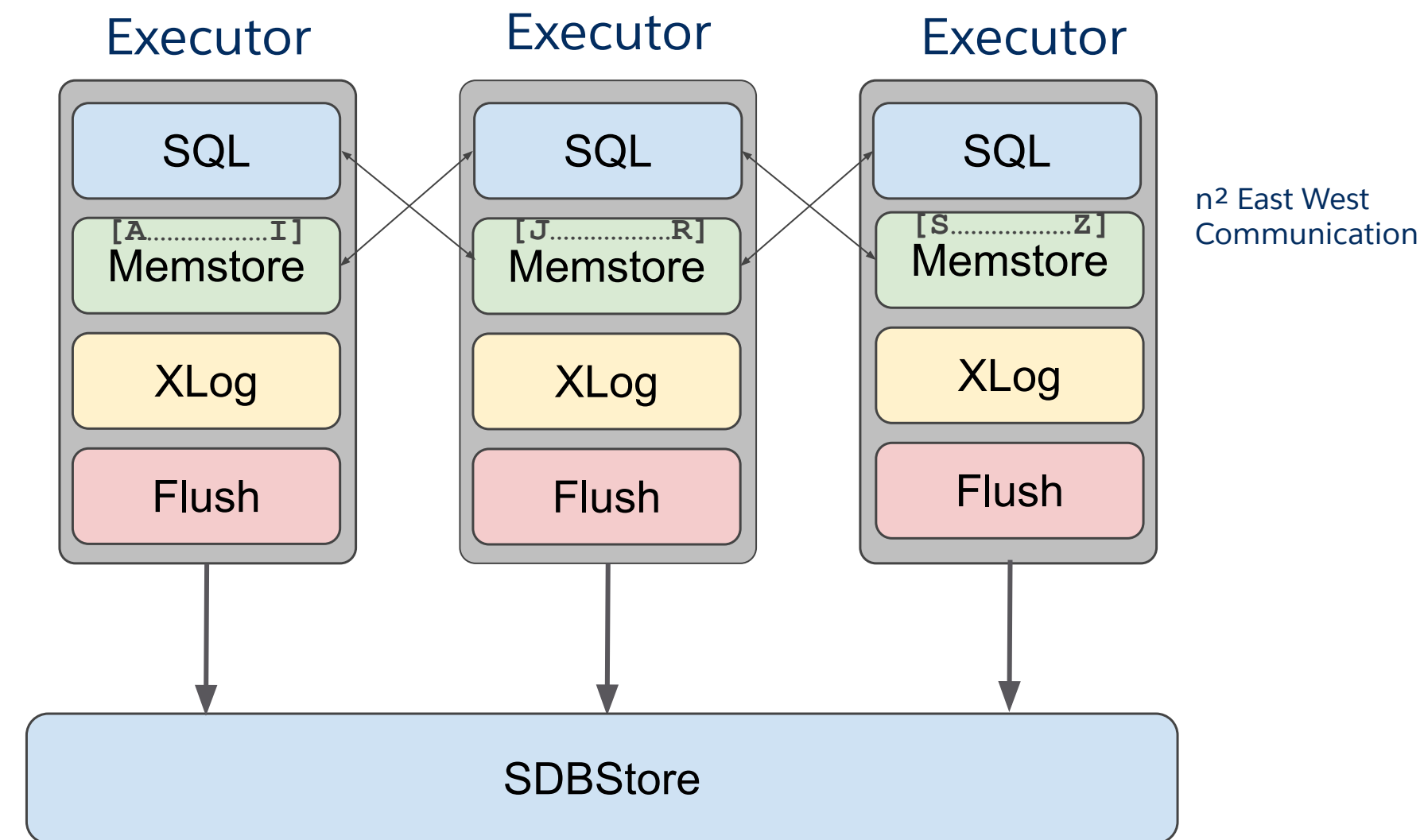


Diminishing returns scaling a single Coordinator to serve entire cluster

Explored symmetric Executors with Memstore partitioning...



Asymmetric Architecture



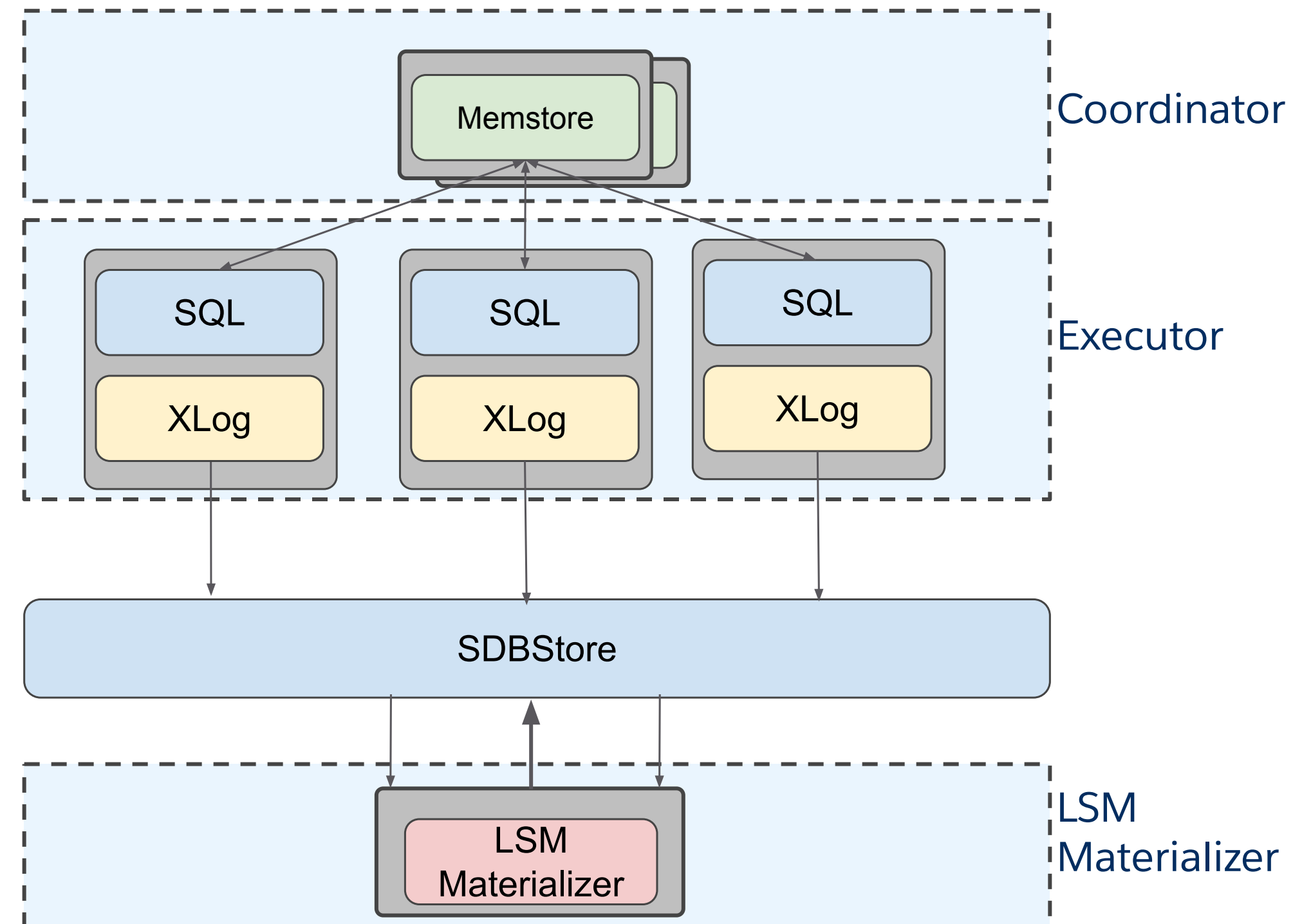
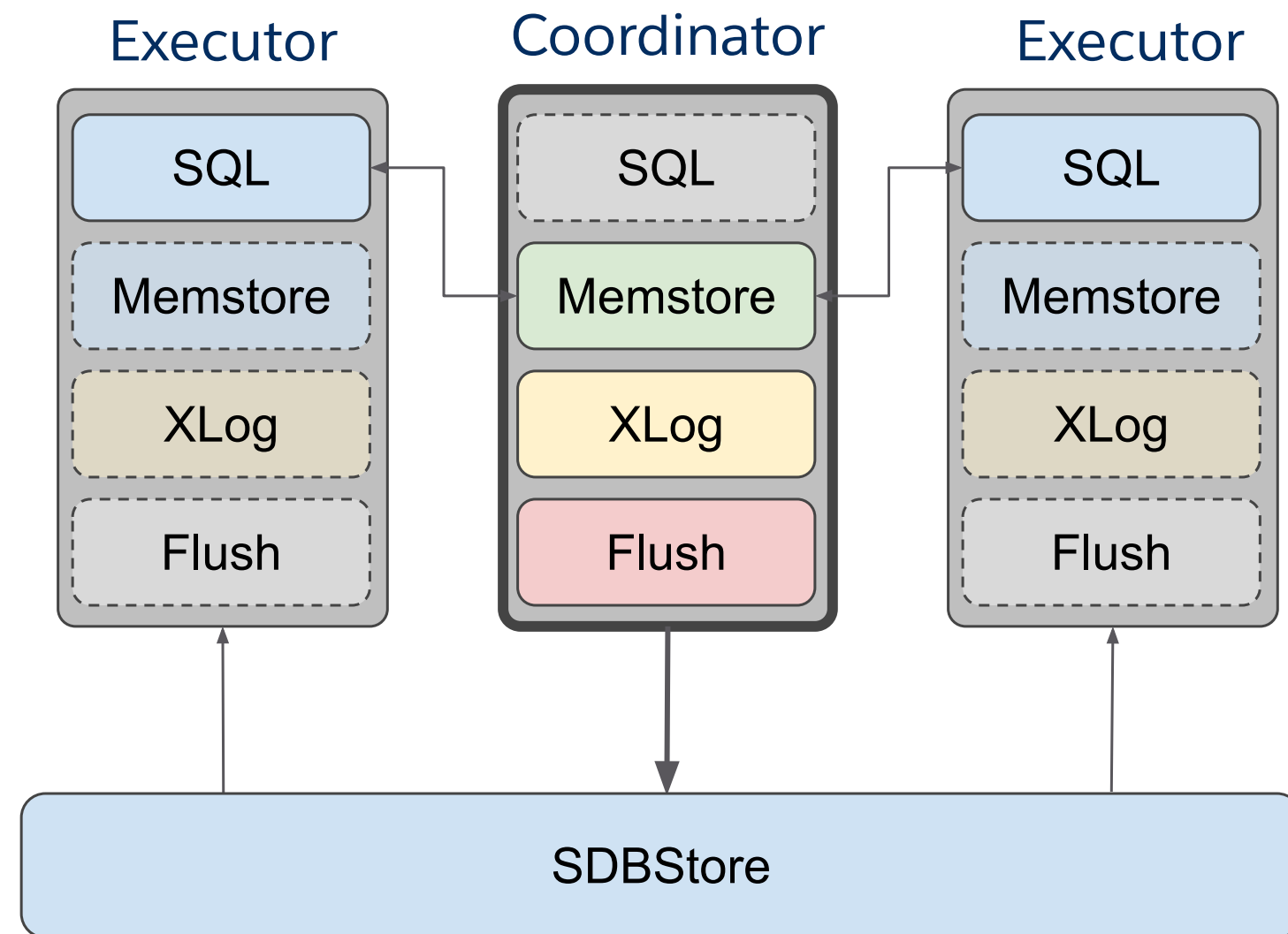
Partitioned Memstore

Disaggregating the Coordinator



Decided to break up Coordinator into separate tiers

Rolling out in-situ



Lessons Learned



Scaling out an LSM is Different than Scaling a Page-Based Database

How you log has profound implications on scale-out

- Coordinate Memstore not entire DB
- No buffer cache coordination
- LSM Materializer is *analogous* to PageServer in page-based shared storage databases but data is read directly from storage

Immutability is the Gift that Keeps Giving

Addition benefits beyond coordination:

- Eliminates a class of corruptions (next talk)
- Simplified storage contract
- Simplifies archival and replication
- Tenant mobility
- Enables expensive one-time write operations like tenant encryption

Software Must be Designed to be Launched Incrementally

In-Situ deployment is a huge topic

- Design to be compatible on multiple dimensions: storage formats, internal module boundaries, client APIs, etc
- Dark launch and other incremental enablement strategies
- Observability
- Incremental rollout



Thank you

We're hiring! Join our team and tackle fun, challenging database problems – from multi-tenancy to scaling distributed storage systems to boosting query performance

