Project Pravega

Storage Reimagined for a Streaming World

Market Drivers

Massive
Data Growth

Emergence of
Real-Time Apps

Monetize Data
with Analytics

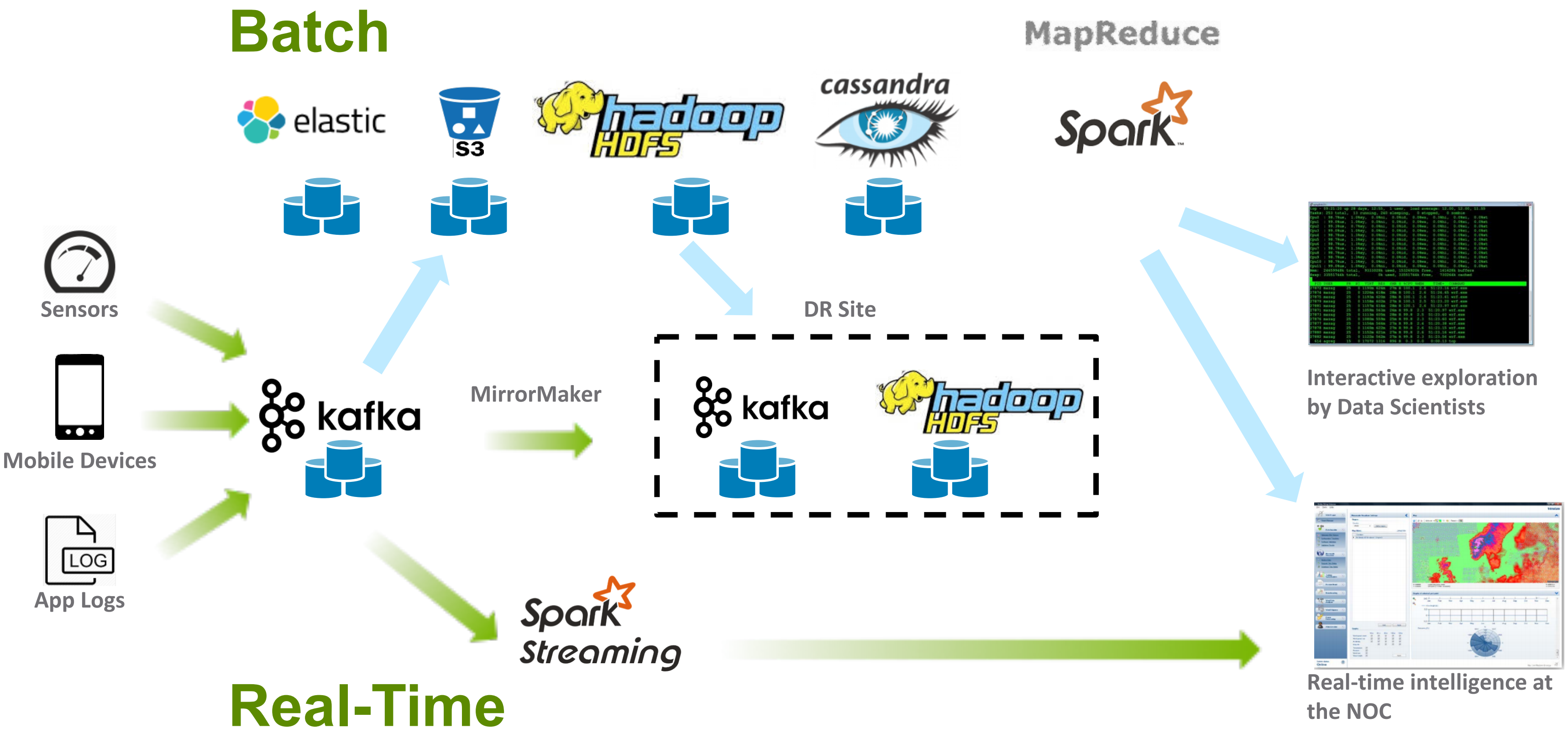
Rapid Dissemination
of Data to Apps

Data Velocity
and Variety

Infrastructure
Commoditization
and Scale-Out

Open Source
Community

Today's "Accidental Architecture"



Batch

MapReduce

DR Site

MirrorMaker

Sensors

Mobile Devices

App Logs

Interactive exploration by Data Scientists

Real-time intelligence at the NOC

Real-Time

A New Architecture Emerges: Streaming

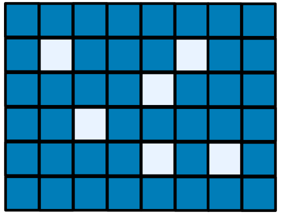
- A new class of streaming systems is emerging to address the accidental architecture's problems and enable new applications not possible before
- Some of the unique characteristics of streaming applications
  - Treat data as continuous and infinite
  - Compute correct results in real-time with stateful, exactly-once processing
- These systems are applicable for real-time applications, batch applications, and interactive applications
- Web-scale companies (Google, Twitter) are beginning to demonstrate the disruptive value of streaming systems
- What are the implications for storage in a streaming world?

Let’s Rewind A Bit: The Importance of Log Storage

Traditional Apps/Middleware ⟷ Streaming Apps/Middleware

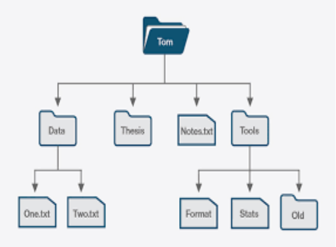
BLOCKS

- Structured Data
- Relational DBs



FILES

- Unstructured Data
- Pub/Sub
- NoSQL DBs

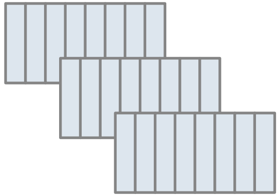


OBJECTS

- Unstructured Data
- Internet Friendly (REST)
- Scale over Semantics
- Geo

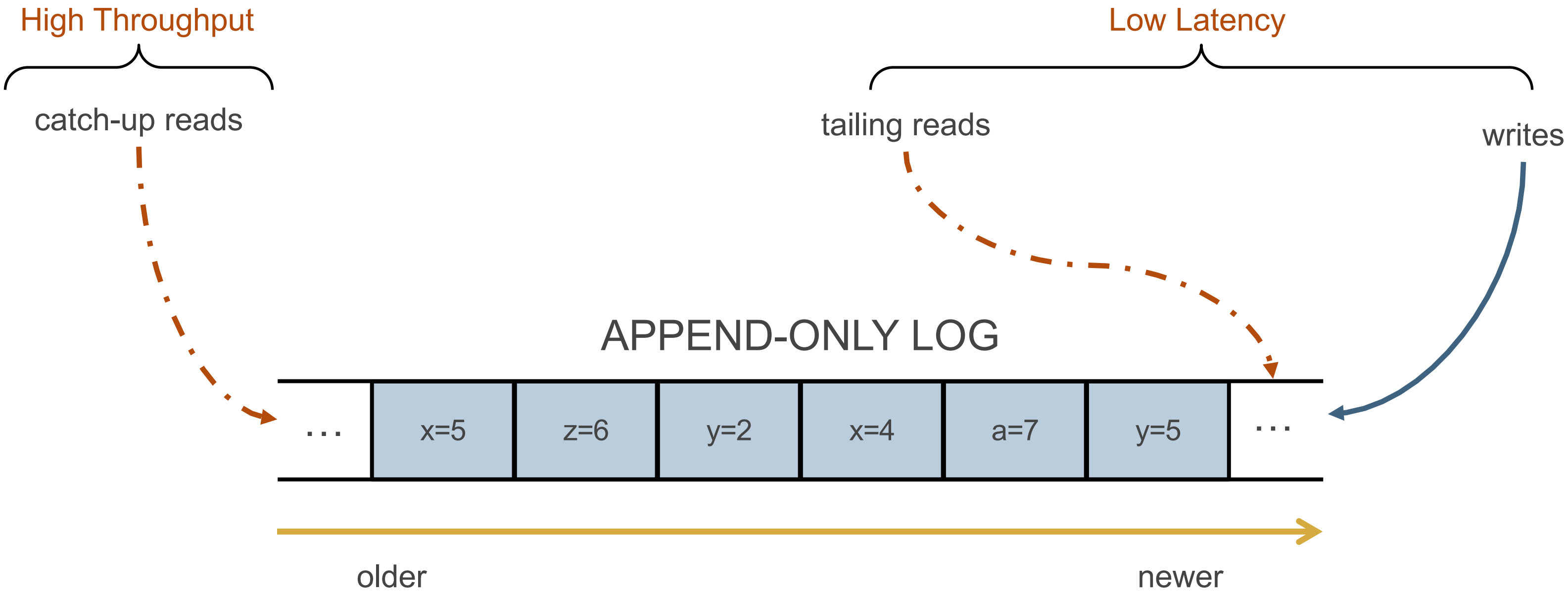
LOGS

- Append-only
- Low-latency
- Tail Read/Write



The Importance of Log Storage

The Fundamental Data Structure for Scale-out Distributed Systems



High Throughput

catch-up reads

Low Latency

tailing reads

writes

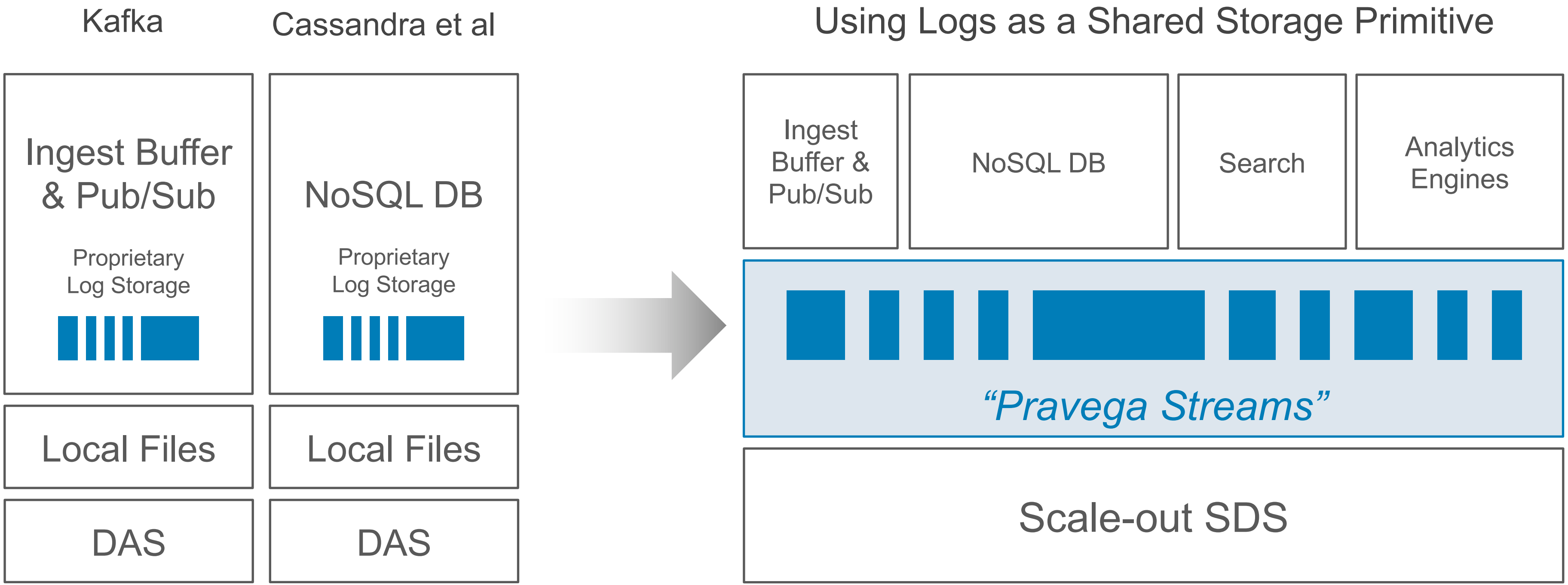
APPEND-ONLY LOG

… x=5 z=6 y=2 x=4 a=7 y=5 …

older

newer

Our Goal: Refactor the “Accidental Storage Stack”



Introducing Pravega Streams

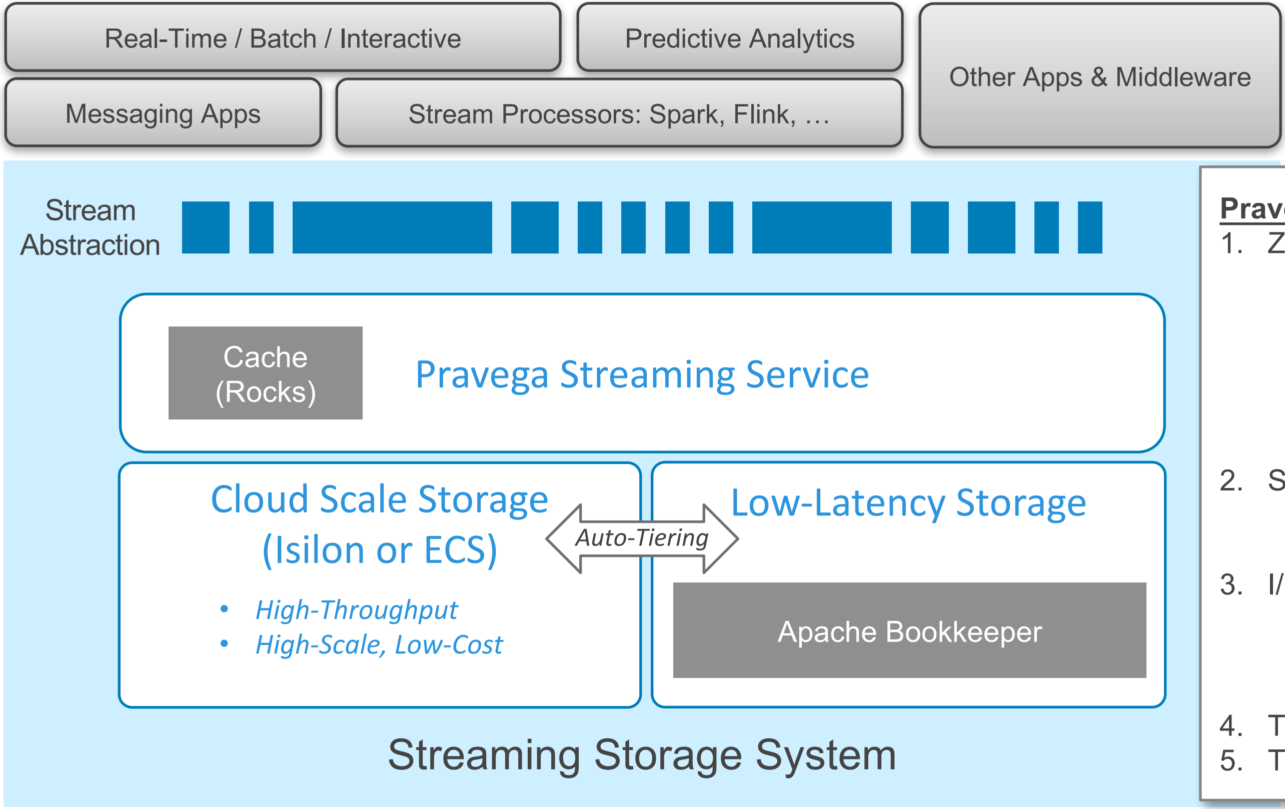
A New Log Primitive Designed Specifically For Streaming Architectures

- Pravega is an open source distributed storage service offering a new storage abstraction called a stream
- A stream is the foundation for building reliable streaming systems: a high-performance, durable, elastic, and infinite append-only log with strict ordering and consistency
- A stream is as lightweight as a file – you can create millions of them in a single cluster
- Streams greatly simplify the development and operation of a variety of distributed systems: messaging, databases, analytic engines, search engines, and so on

Pravega Architecture Goals

- All data is durable
  - Data is replicated and persisted to disk before being acknowledged
- Strict ordering guarantees and exactly once semantics
  - Across both tail and catch-up reads
  - Client tracks read offset, Producers use transactions
- Lightweight, elastic, infinite, high performance
  - Support tens of millions of streams
  - Dynamic partitioning of streams based on load and throughput SLO
  - Size is not bounded by the capacity of a single node
  - Low (<10ms) latency writes; throughput bounded by network bandwidth
  - Read pattern (e.g. many catch-up reads) doesn’t affect write performance

Architecture



Real-Time / Batch / Interactive

Predictive Analytics

Other Apps & Middleware

Messaging Apps

Stream Processors: Spark, Flink, …

Stream Abstraction

Cache (Rocks)

Pravega Streaming Service

Cloud Scale Storage (Isilon or ECS)

- *High-Throughput*
- *High-Scale, Low-Cost*

Auto-Tiering

Low-Latency Storage

Apache Bookkeeper

Streaming Storage System

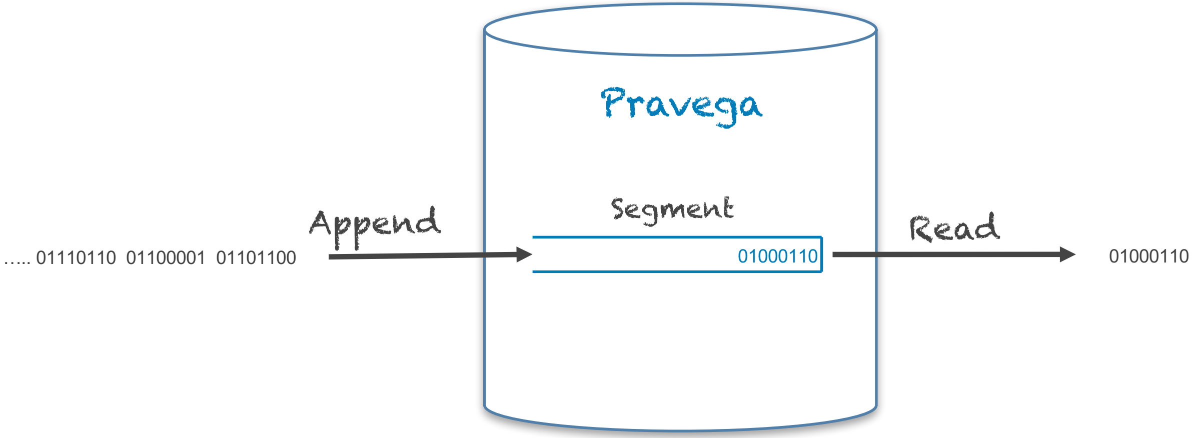
<u>Pravega Design Innovations</u>

1. Zero-Touch Dynamic Scaling
   - Automatically scale read/write parallelism based on load and SLO
   - No service interruptions
   - No manual reconfiguration of clients
   - No manual reconfiguration of service resources
2. Smart Workload Distribution
   - No need to over-provision servers for peak load
3. I/O Path Isolation
   - For tail writes
   - For tail reads
   - For catch-up reads
4. Tiering for “Infinite Streams”
5. Transactions For “Exactly Once”

Pravega Fundamentals

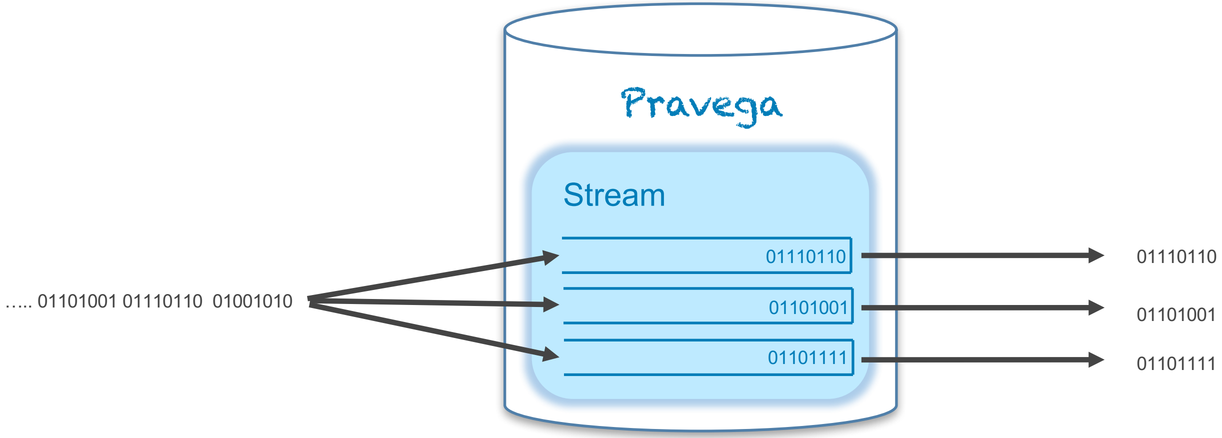
Segments

- Base storage primitive is a segment
- A segment is an append-only sequence of bytes
- Writes durably persisted before acknowledgement



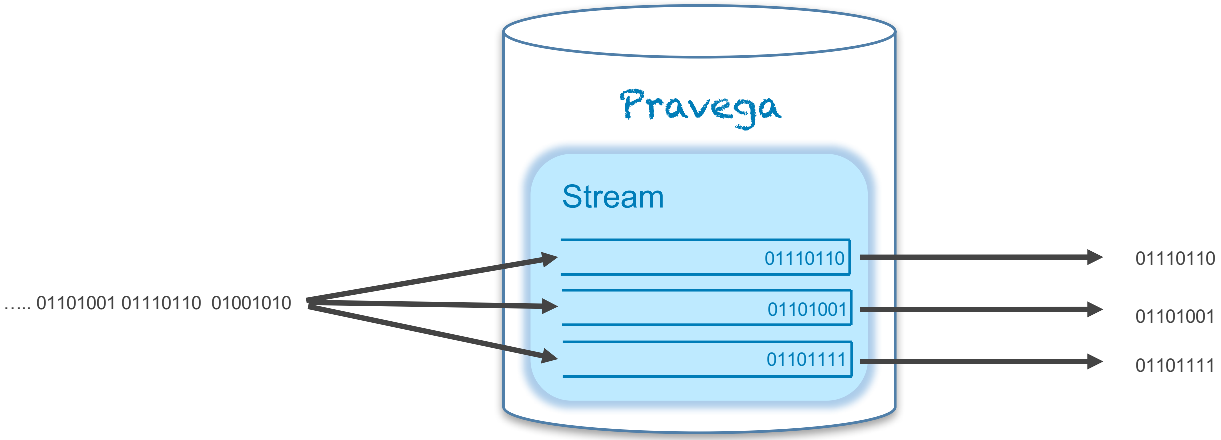
Streams

- A stream is composed of one or more segments
- Routing key determines the target segment for a stream write
- Write order preserved by routing key; consistent tail and catch-up reads



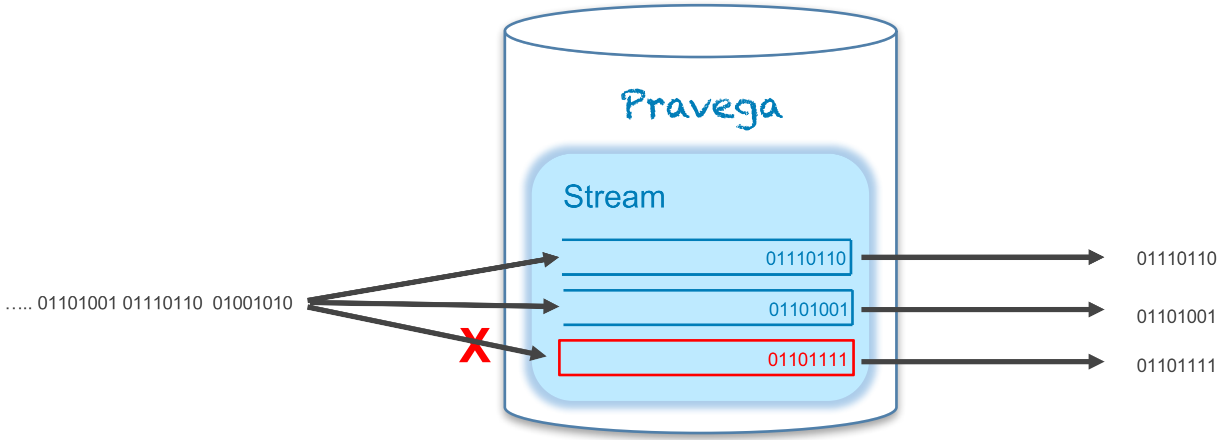
Streams

- There are no architectural limits on the number of streams or segments
- Each segment can live in a different server
- System is not limited in any way by the capacity of a single server

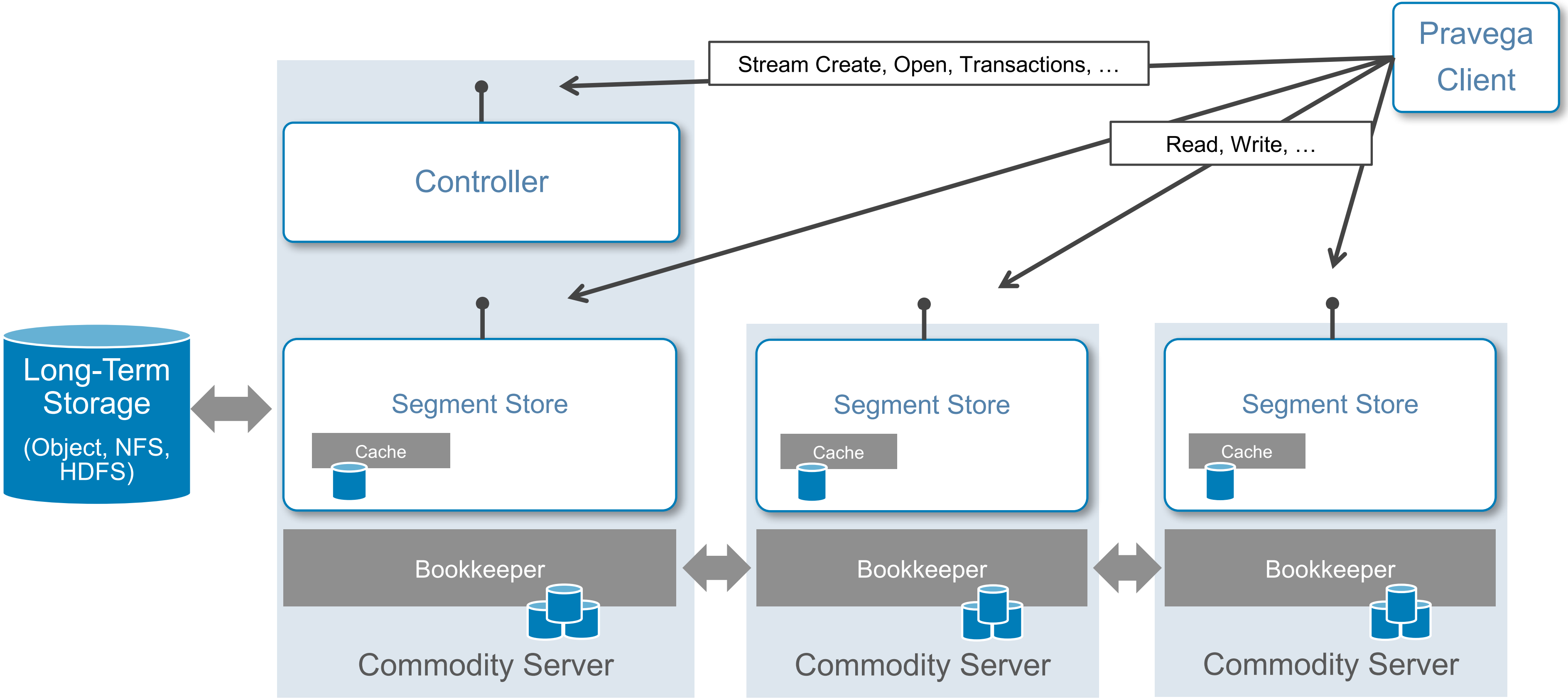


Segment Sealing

- A segment may be sealed
- A sealed segment cannot be appended to any more
- Basis for advanced features such as stream elasticity and transactions



Pravega System Architecture

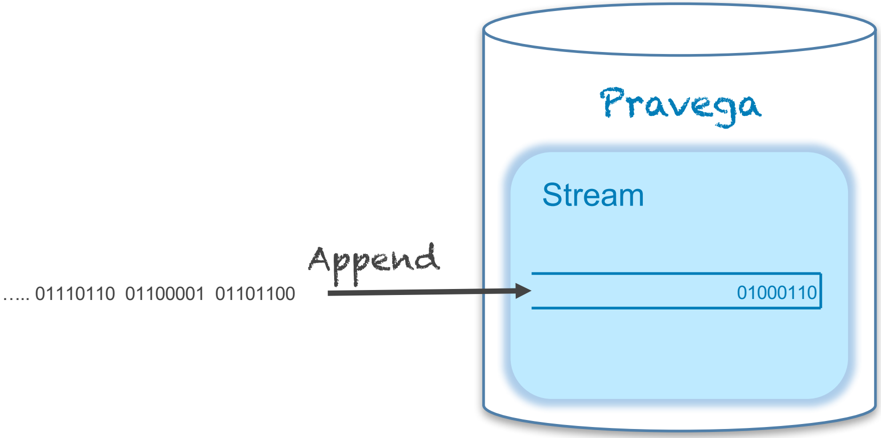


Beyond the Fundamentals

Stream Elasticity, Unbounded Streams, Transactions, Exactly Once

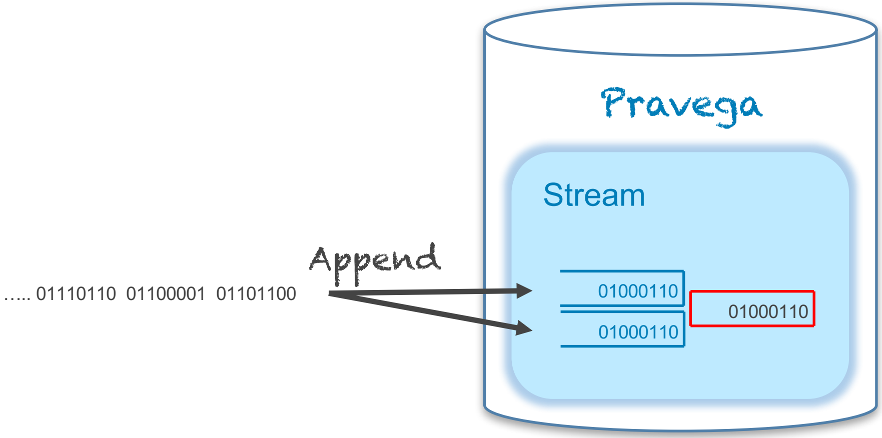
Stream Elasticity

- Data arrival volume increases – more parallelism needed!

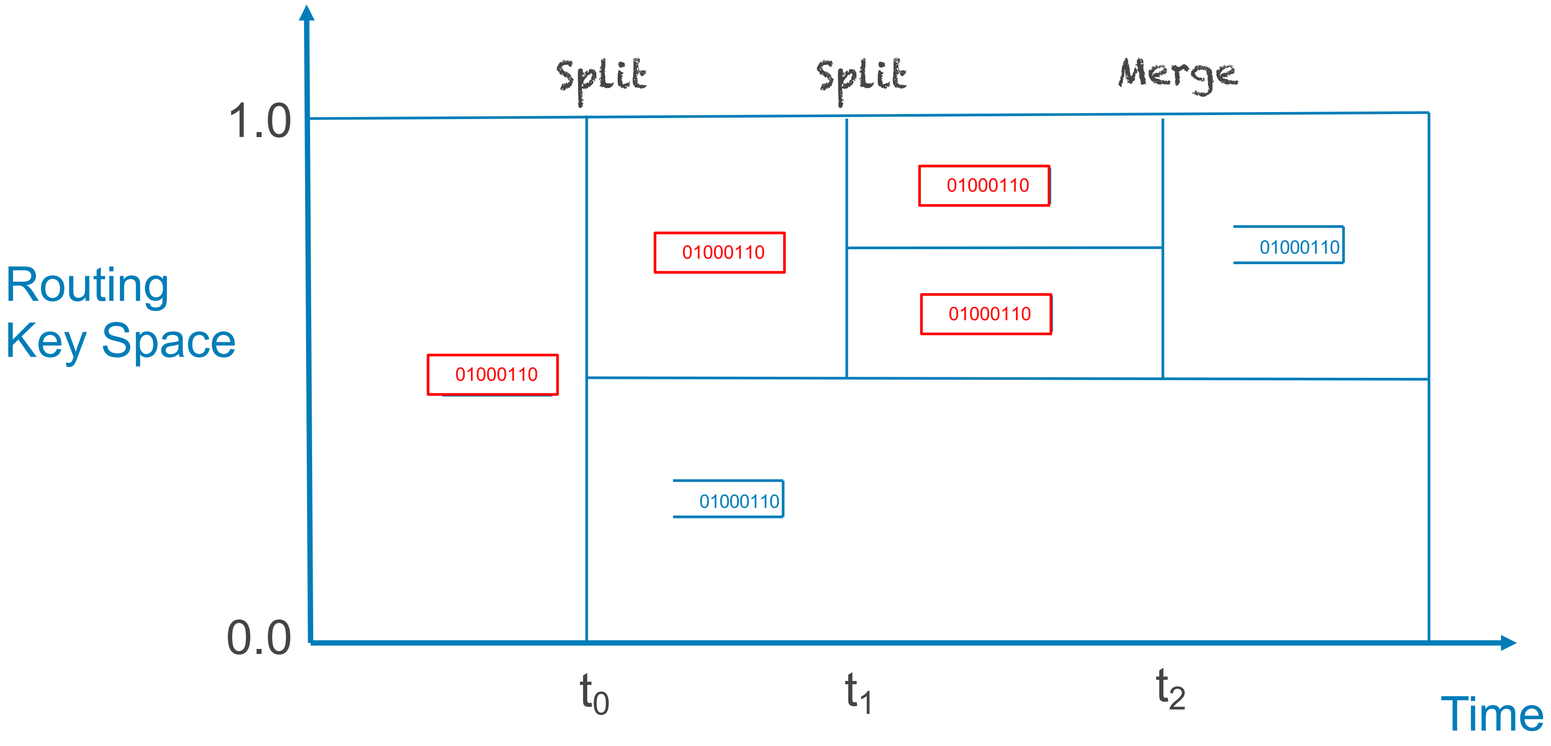


Stream Elasticity

- Seal original segment
- Replace with two new ones!
- New segments may be distributed throughout the cluster balancing load

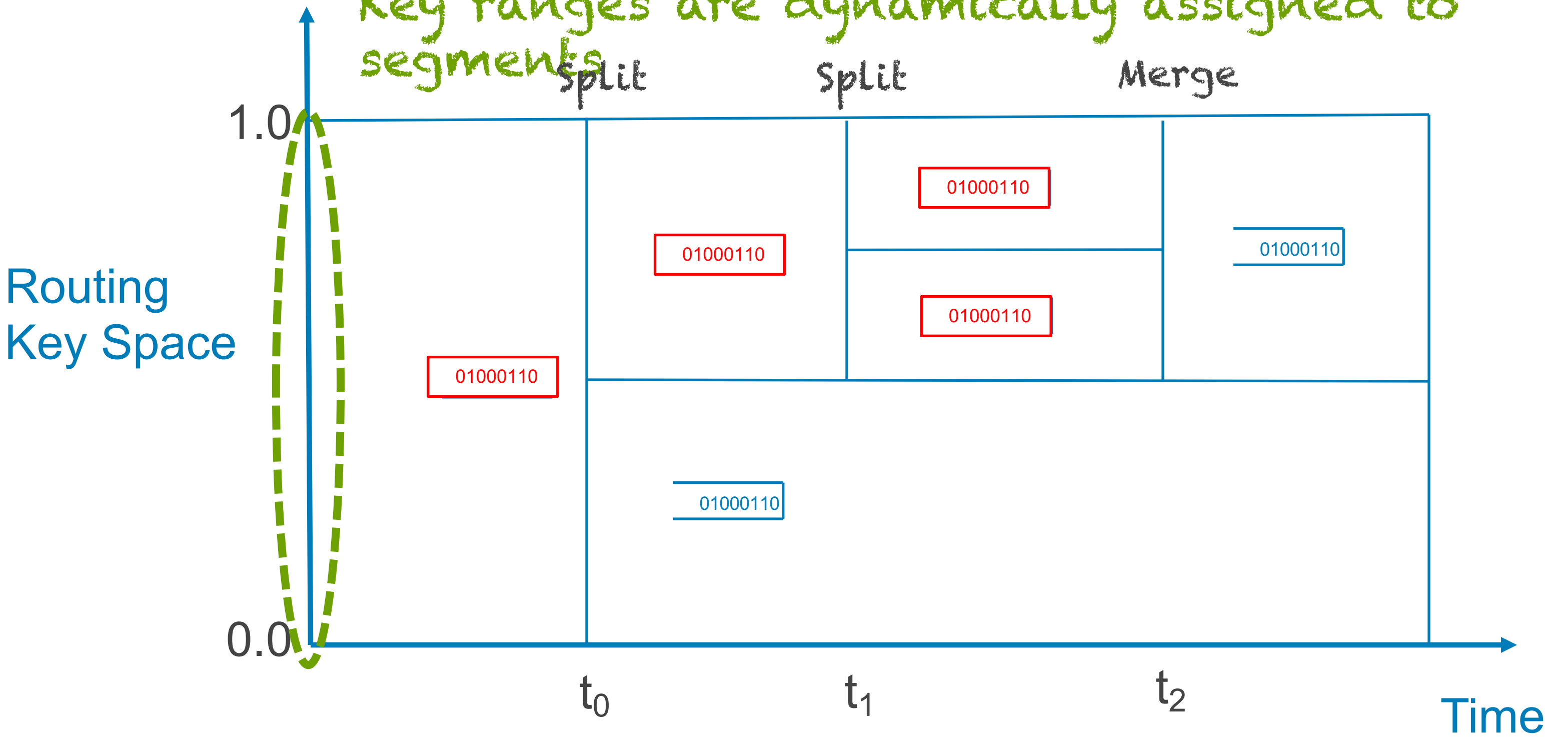


Stream Elasticity

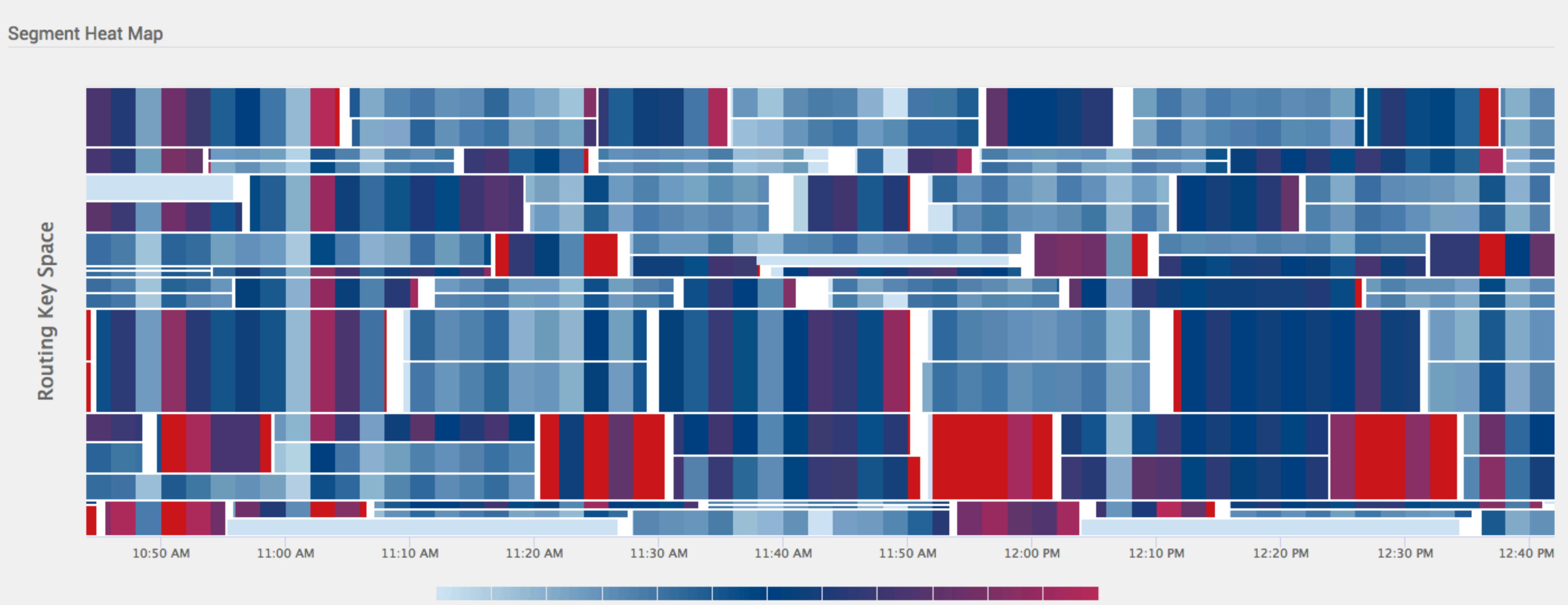


Stream Elasticity

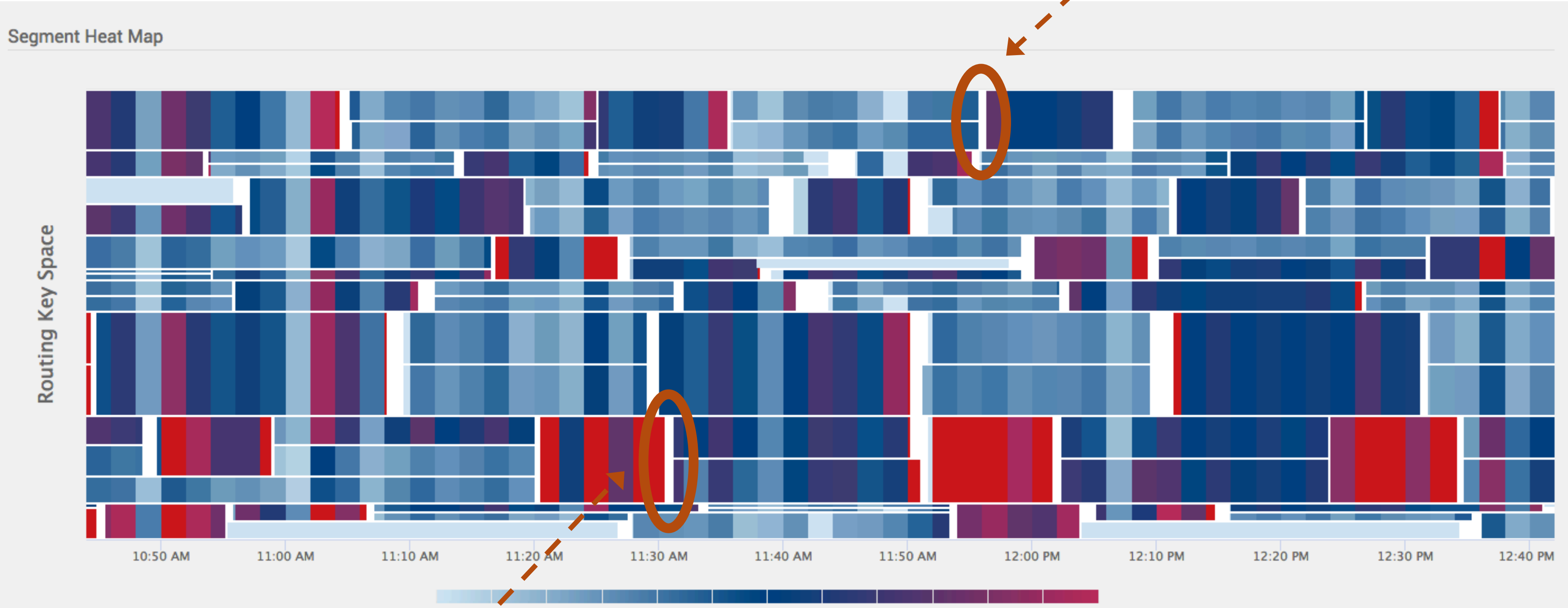
Key ranges are dynamically assigned to segments



Stream Elasticity



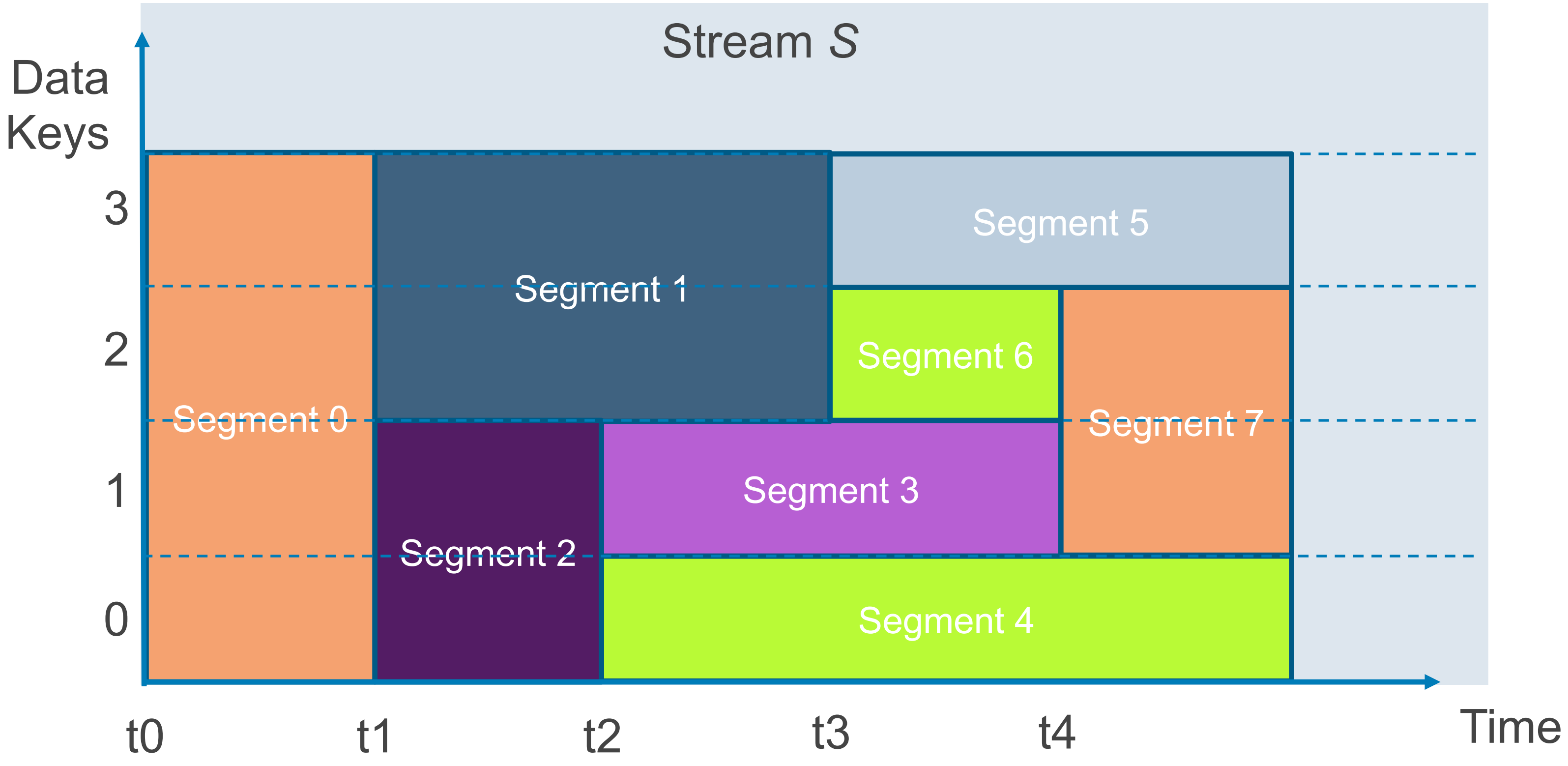
Stream Elasticity



Scale down

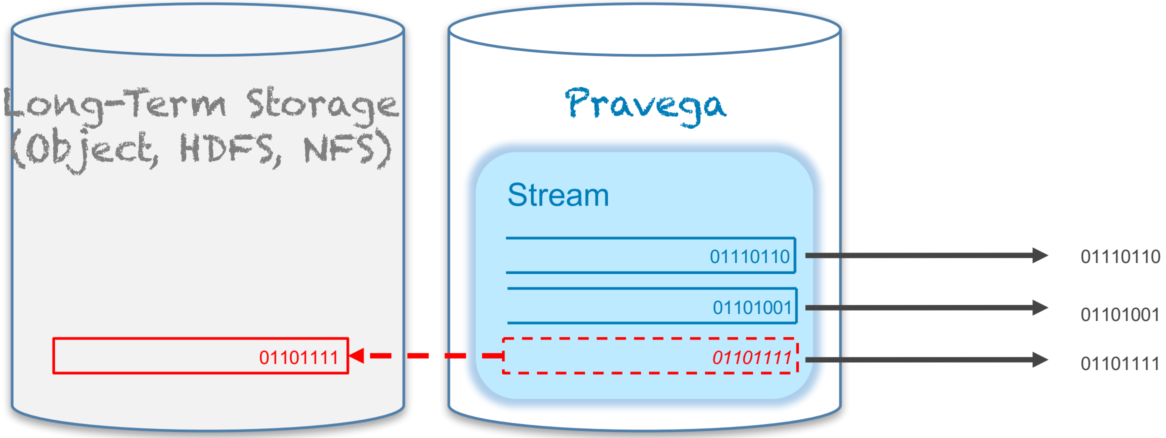
Scale up

Zero-Touch Scaling: Segment Splitting & Merging

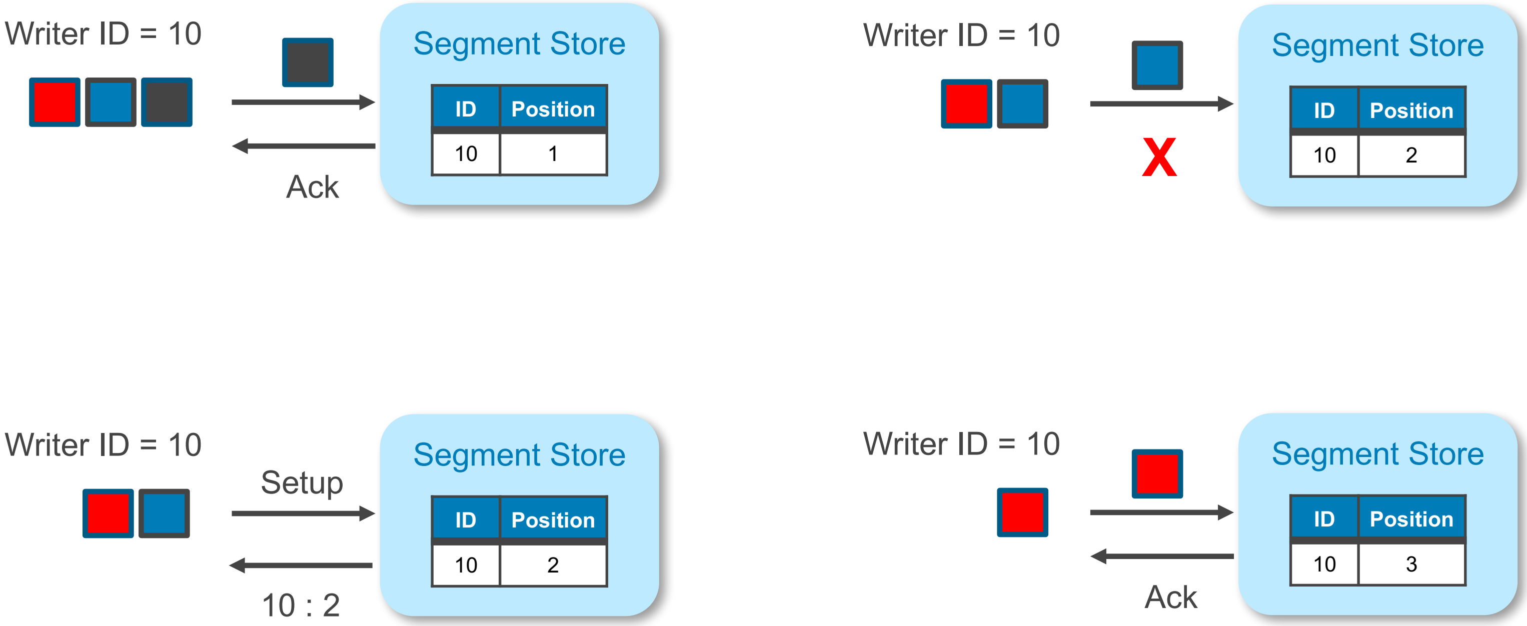


Unbounded Streams

- Segments are automatically tiered to long-term storage
- Data in tiered segments is transparently accessible for catch-up reads
- Preserves stream abstraction while lowering storage costs for older data



Exactly Once



Writer ID = 10

Ack

Segment Store

ID	Position
10	1

Writer ID = 10

X

Segment Store

ID	Position
10	2

Writer ID = 10

Setup

10 : 2

Segment Store

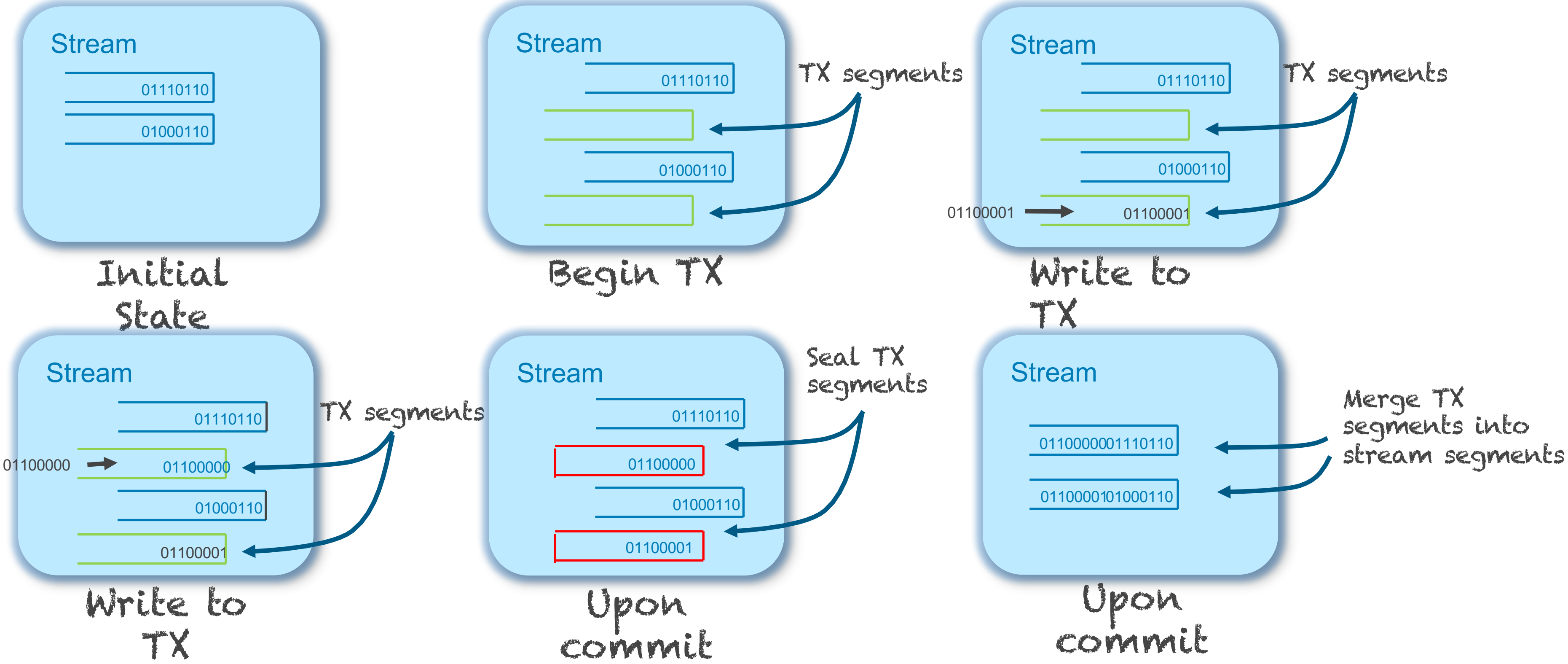
ID	Position
10	2

Writer ID = 10

Ack

Segment Store

ID	Position
10	3



Stream

01110110

01000110

Initial State

Stream

01110110

01000110

TX segments

Begin TX

Stream

01110110

01000110

01100001

01100001

TX segments

Write to TX

Stream

01110110

01100000

01100000

01000110

01100001

TX segments

Write to TX

Stream

01110110

01100000

01000110

01100001

Seal TX segments

Upon commit

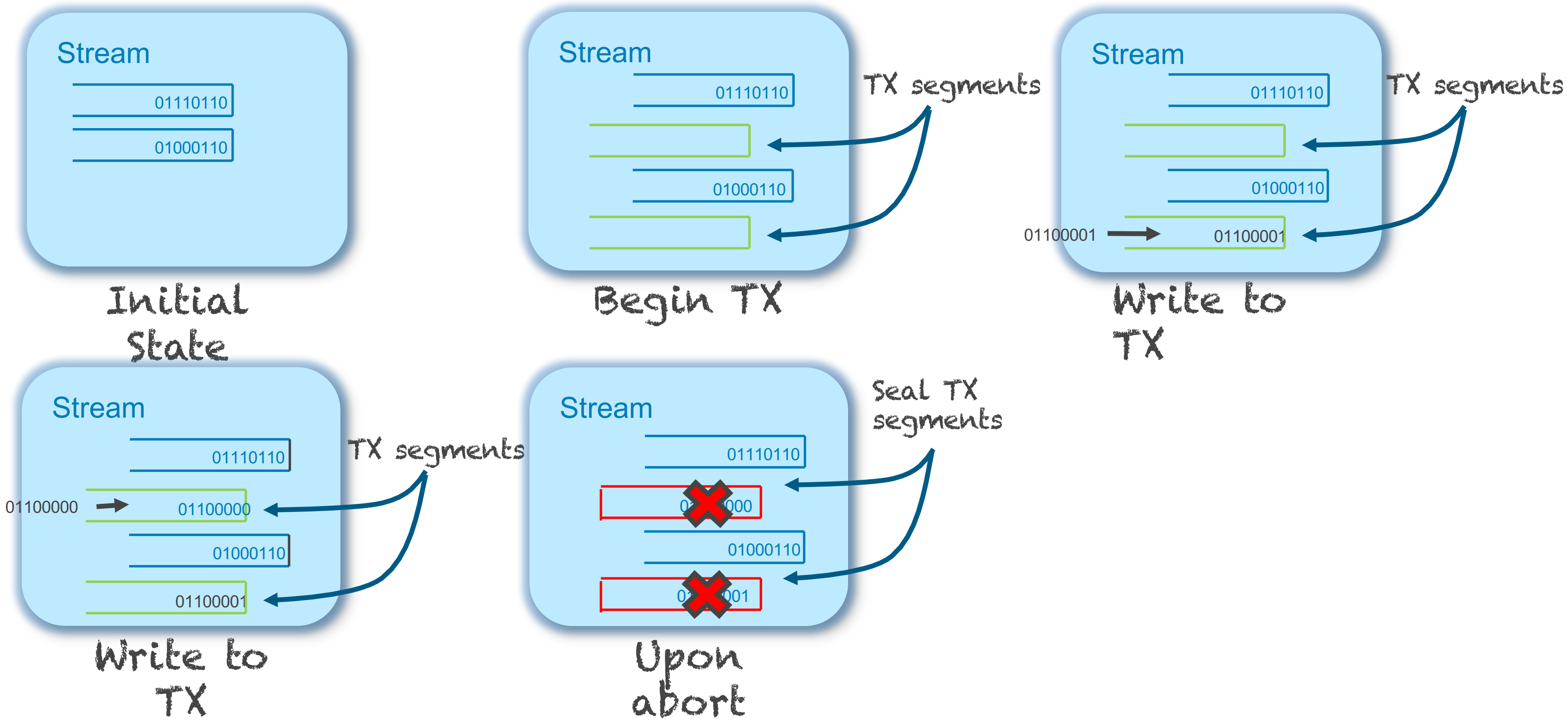
Stream

0110000001110110

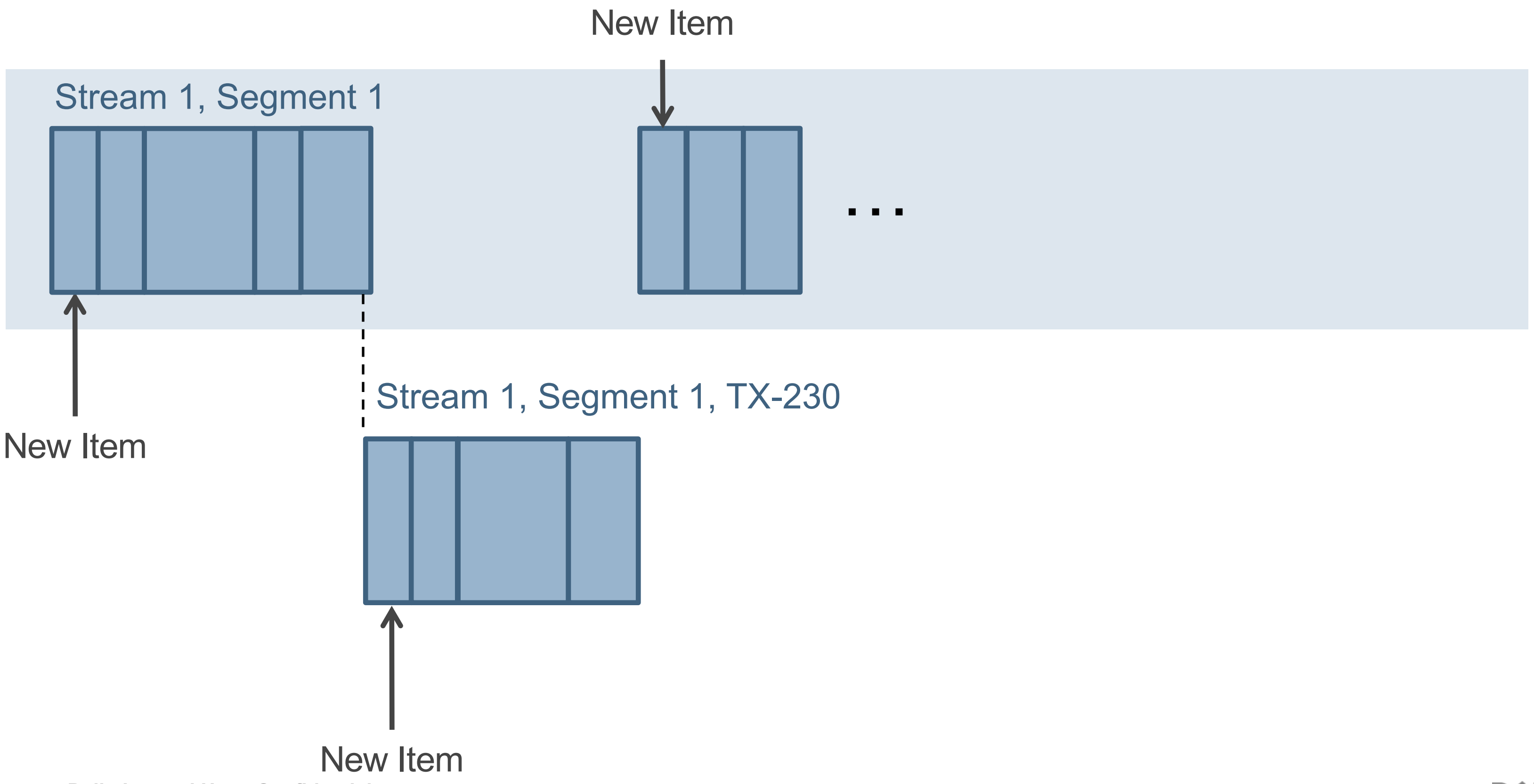
0110000101000110

Merge TX segments into stream segments

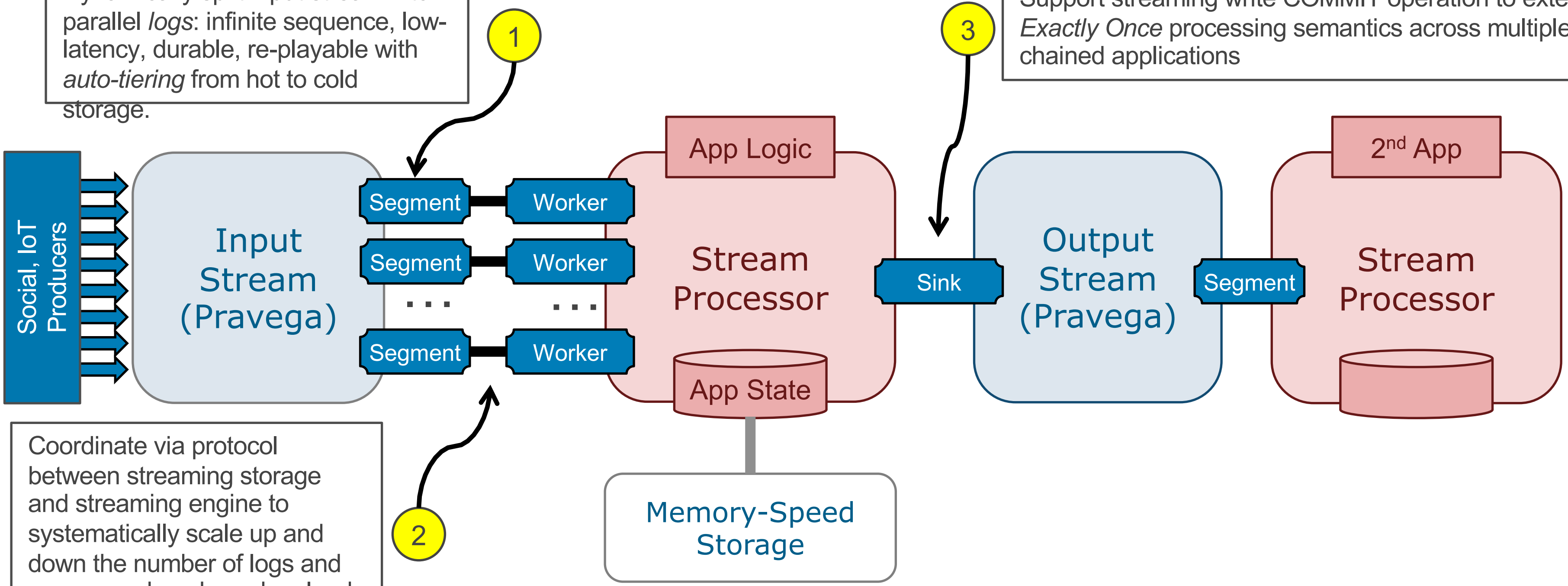
Upon commit


Stream
01110110
01000110
Initial State
Stream
01110110
01000110
TX segments
Begin TX
Stream
01110110
01000110
01100001
01100001
TX segments
Write to TX
Stream
01110110
01100000
01100000
01000110
01100001
TX segments
Write to TX
Stream
01110110
0 000
01000110
0 001
Seal TX segments
Upon abort

Transactional Semantics For “Exactly Once”



Pravega Optimizations for Stream Processors

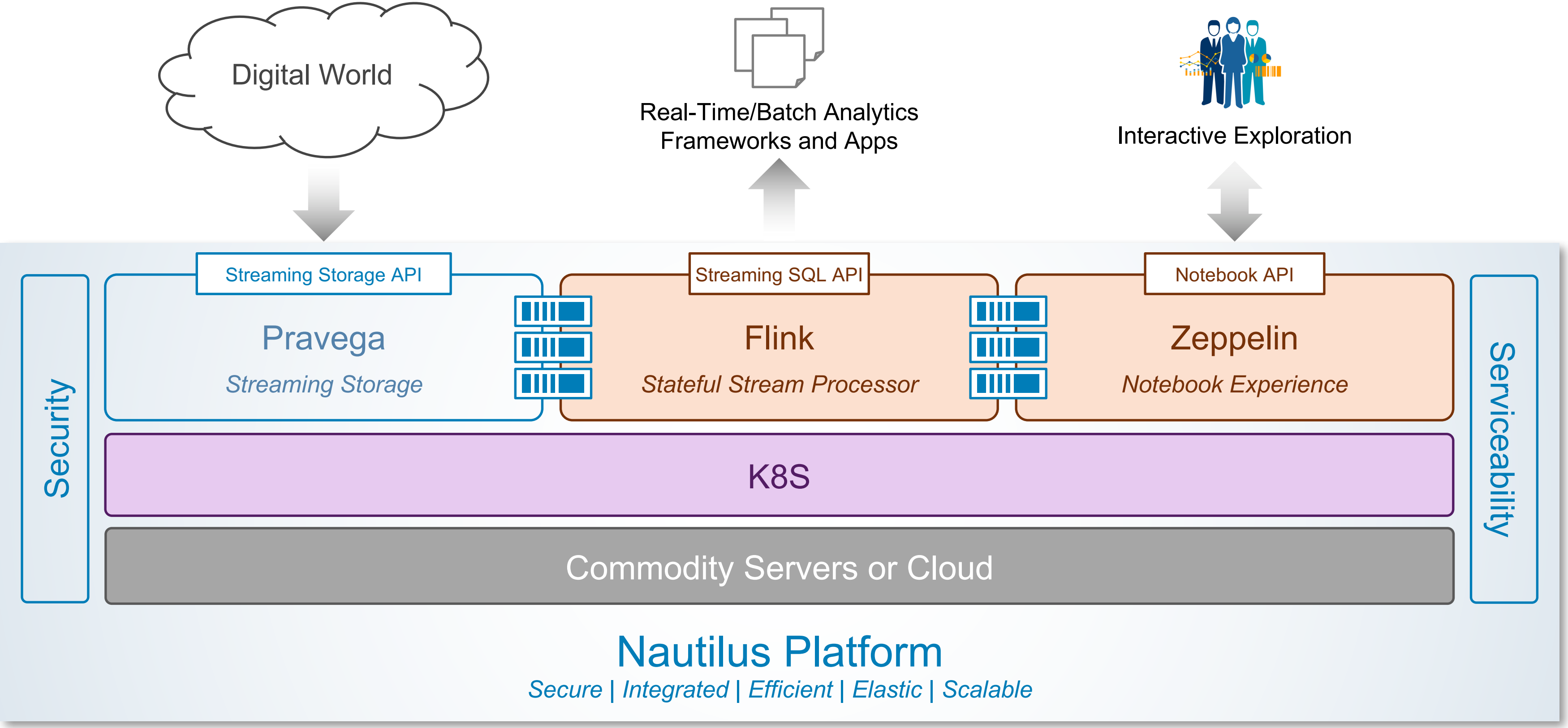


1. Dynamically split input stream into parallel *logs*: infinite sequence, low-latency, durable, re-playable with *auto-tiering* from hot to cold storage.

2. Coordinate via protocol between streaming storage and streaming engine to systematically scale up and down the number of logs and source workers based on load variance over time

3. Support streaming write COMMIT operation to extend *Exactly Once* processing semantics across multiple, chained applications

A Turn-Key Streaming Data Platform



Summary

1. “Streaming Architecture” replaces “Accidental Architecture”
   - Data: infinite/continuous vs. static/finite
   - Correctness in real-time: Exactly once processing + consistent storage
2. Pravega Streaming Storage Enables Storage Refactoring
   - Infinite, durable, scalable, re-playable, elastic append-only log
   - Open source project
3. Unified Storage + Unified Data Pipeline
   - The New Data Stack!

DELL EMC

Comparing Pravega and Kafka Design Points

Unlike Kafka, Pravega is designed to be a durable and permanent storage system

Quality	Pravega Goal	Kafka Design Point
Data Durability	Replicated and persisted to disk before ACK	Replicated but not persisted to disk before ACK ✗
Strict Ordering	Consistent ordering on tail and catch-up reads	Messages may get reordered ✗
Exactly Once	Producers can use transactions for atomicity	Messages may get duplicated ✗
Scale	Tens of millions of streams per cluster	Thousands of topics per cluster ✗
Elastic	Dynamic partitioning of streams based on load and SLO	Statically configured partitions ✗
Size	Log size is not bounded by the capacity of any single node	Partition size is bounded by capacity of filesystem on its hosting node ✗
	Transparently migrate/retrieve data from Tier 2 storage for older parts of the log	External ETL required to move data to Tier 2 storage; no access to data via Kafka once moved ✗
Performance	Low (<10ms) latency durable writes; throughput bounded by network bandwidth	Low-latency achieved only by reducing replication/reliability parameters ✗
	Read pattern (e.g. many catch-up readers) does not affect write performance	Read patterns adversely affects write performance due to reliance on OS filesystem cache ✗