Storage Infrastructure behind LinkedIn's Recommendations

Monday April 17th, 2017

By Siddharth Singh
Engineering Manager, Storage Infrastructure

Agenda

- What is LinkedIn ?
  - High Level Web Architecture
- What is Primary and Derived Data ?
  - Recommendation Data Lifecycle
- Derived Data Serving
  - Voldemort Read Only (RO): Architecture and Key Details
  - Lamda Architecture at LinkedIn
  - Beyond Lambda
  - Venice: Architecture and Key Details
- Challenges & how we solved them
- Early Wins and Future Prospects
- Q&A

LinkedIn - World's Largest Professional Network

484M

Members

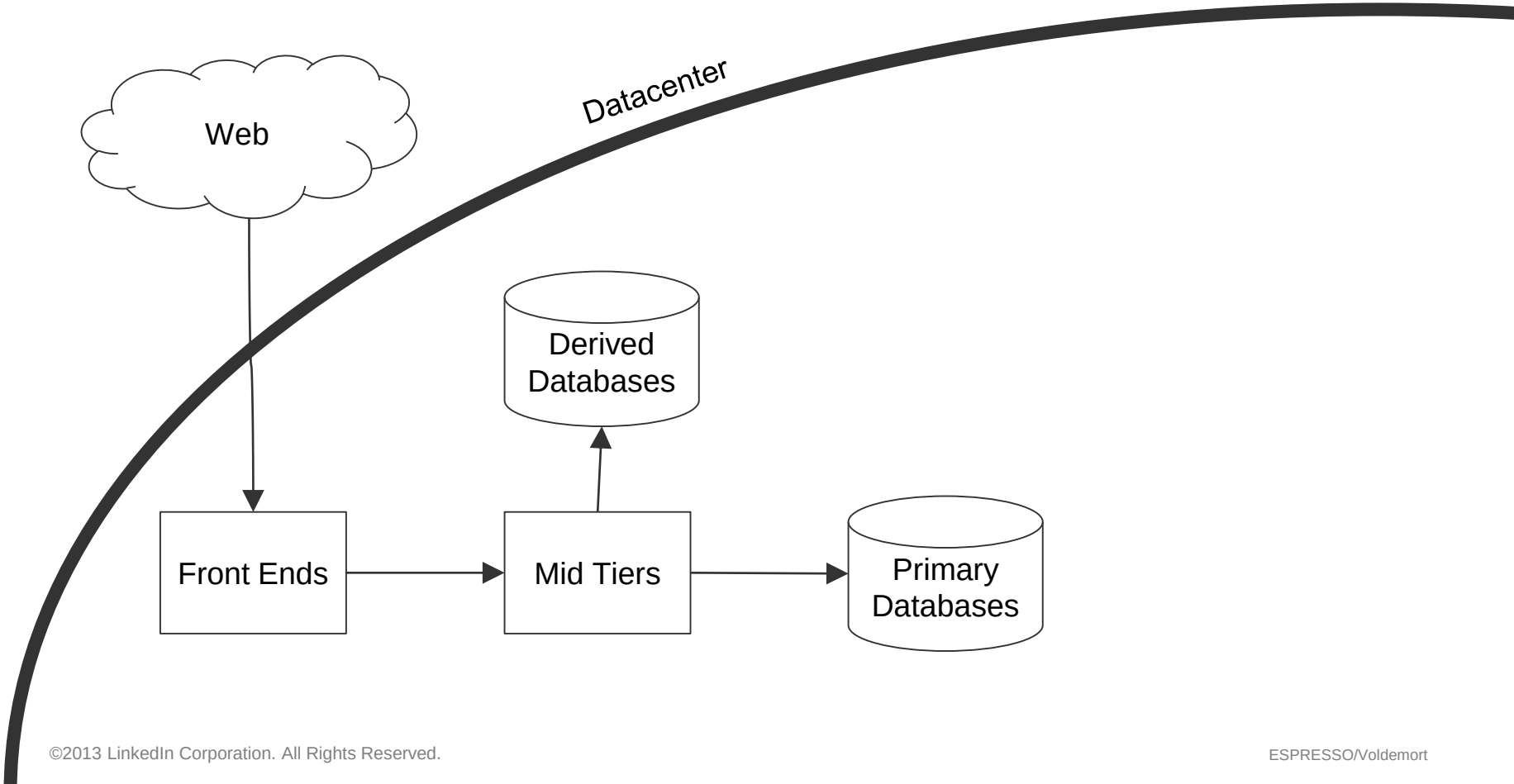
>2

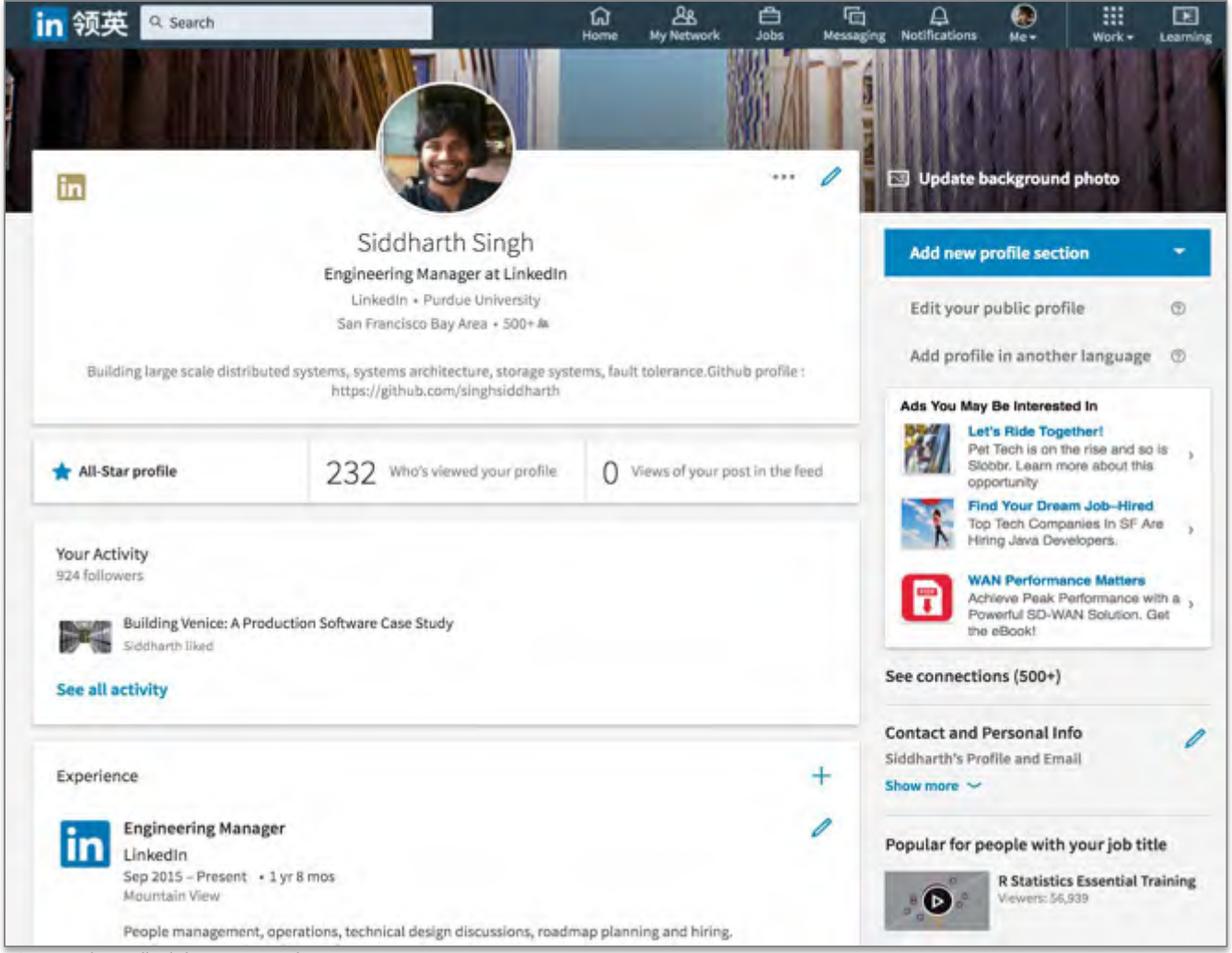
New Members
Per Second

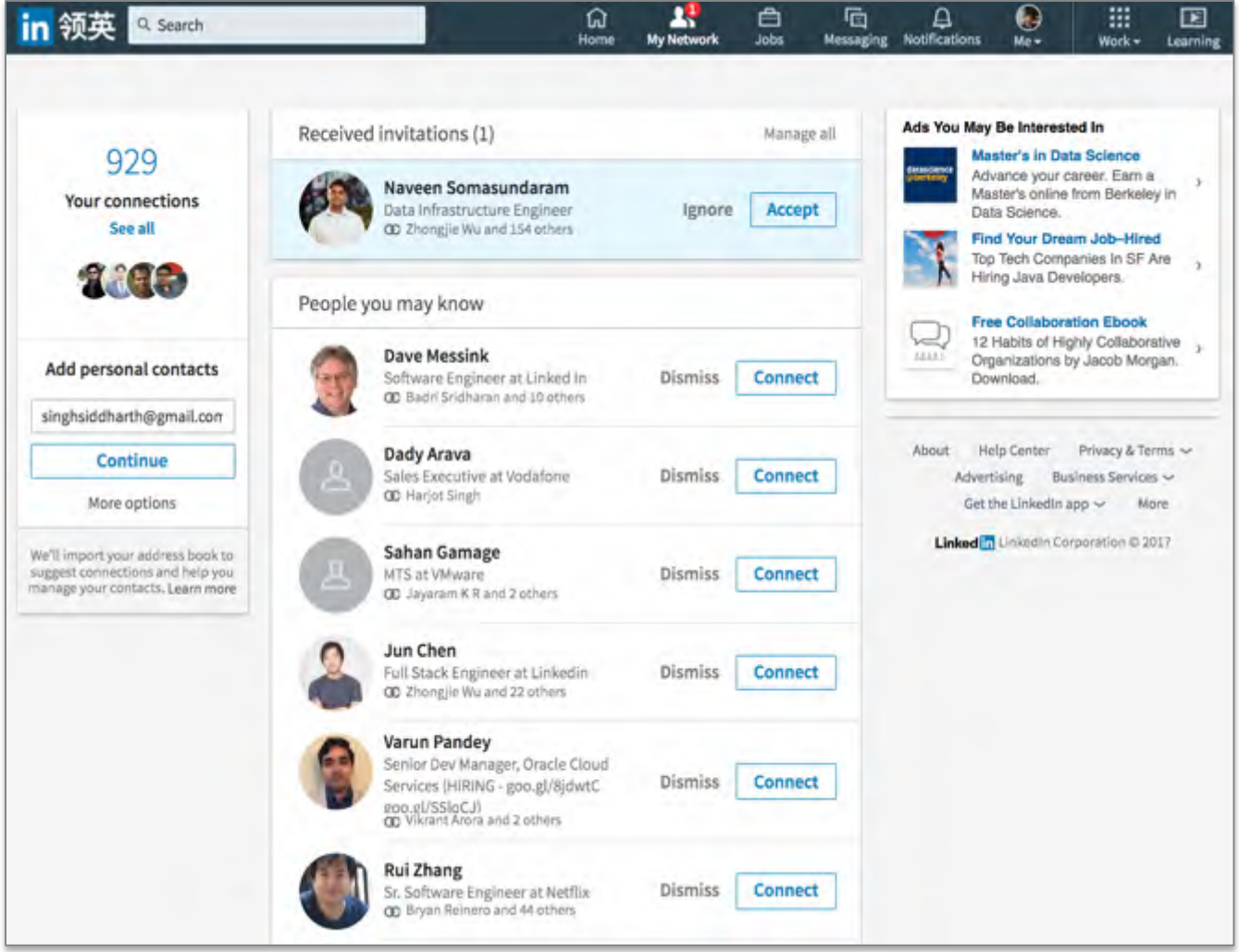
107M

Monthly Unique
Visitors

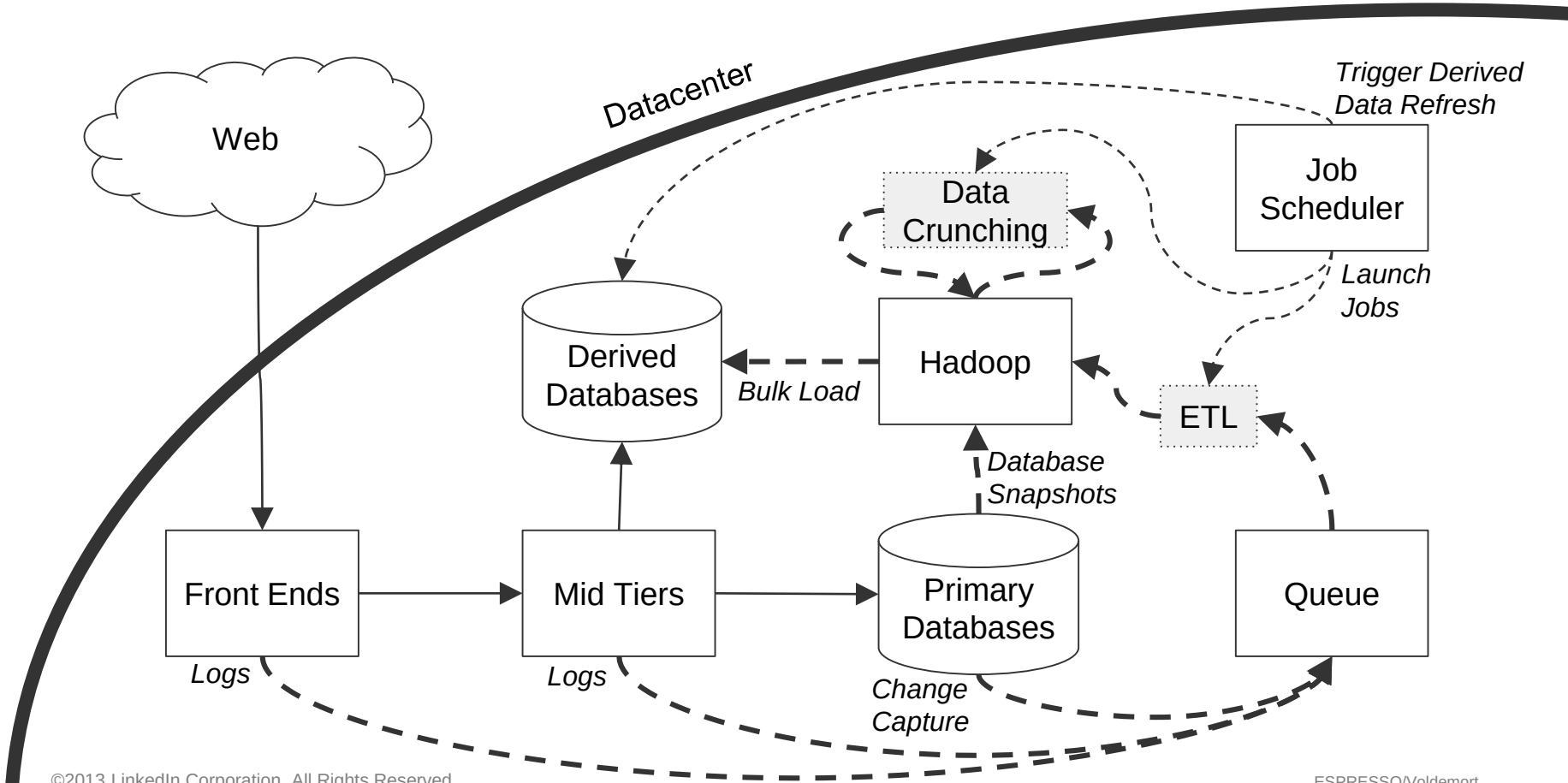
High Level Web Architecture







Recommendation Data Lifecycle



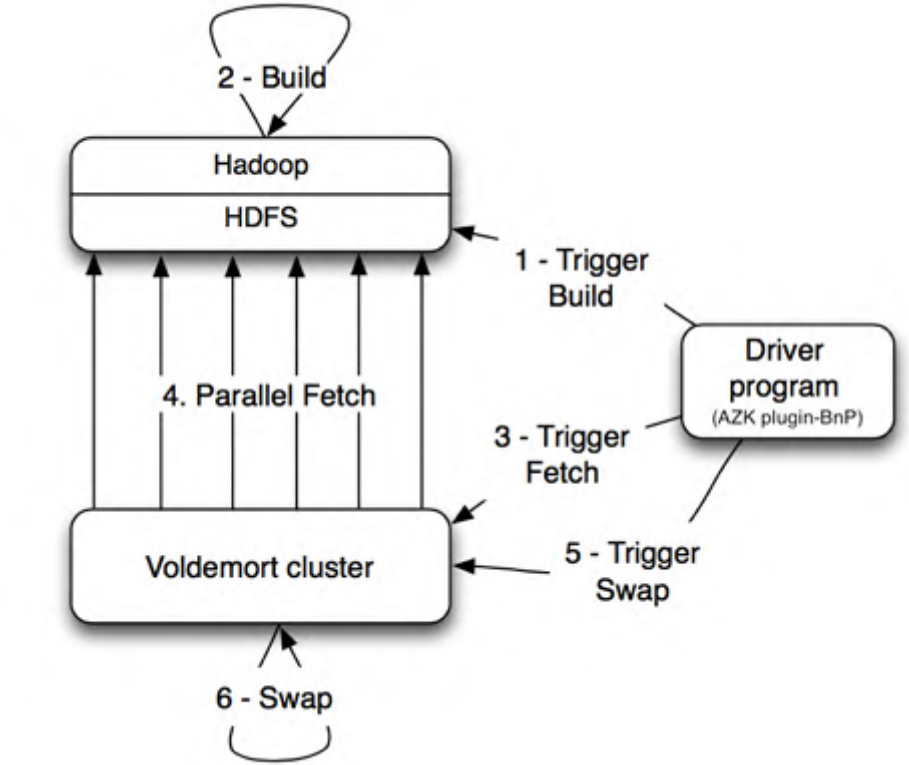
Derived Data Serving

Voldemort

- Distributed Key Value Store
  - Consistent hashing
  - Partitions
- Shared Nothing
- Pluggable architecture
  - Storage Engine : Read-Only or Read-Write
  - Serialization (Avro, JSON etc.)
  - Local or Global

Voldemort Read Only (RO) – Build and Push

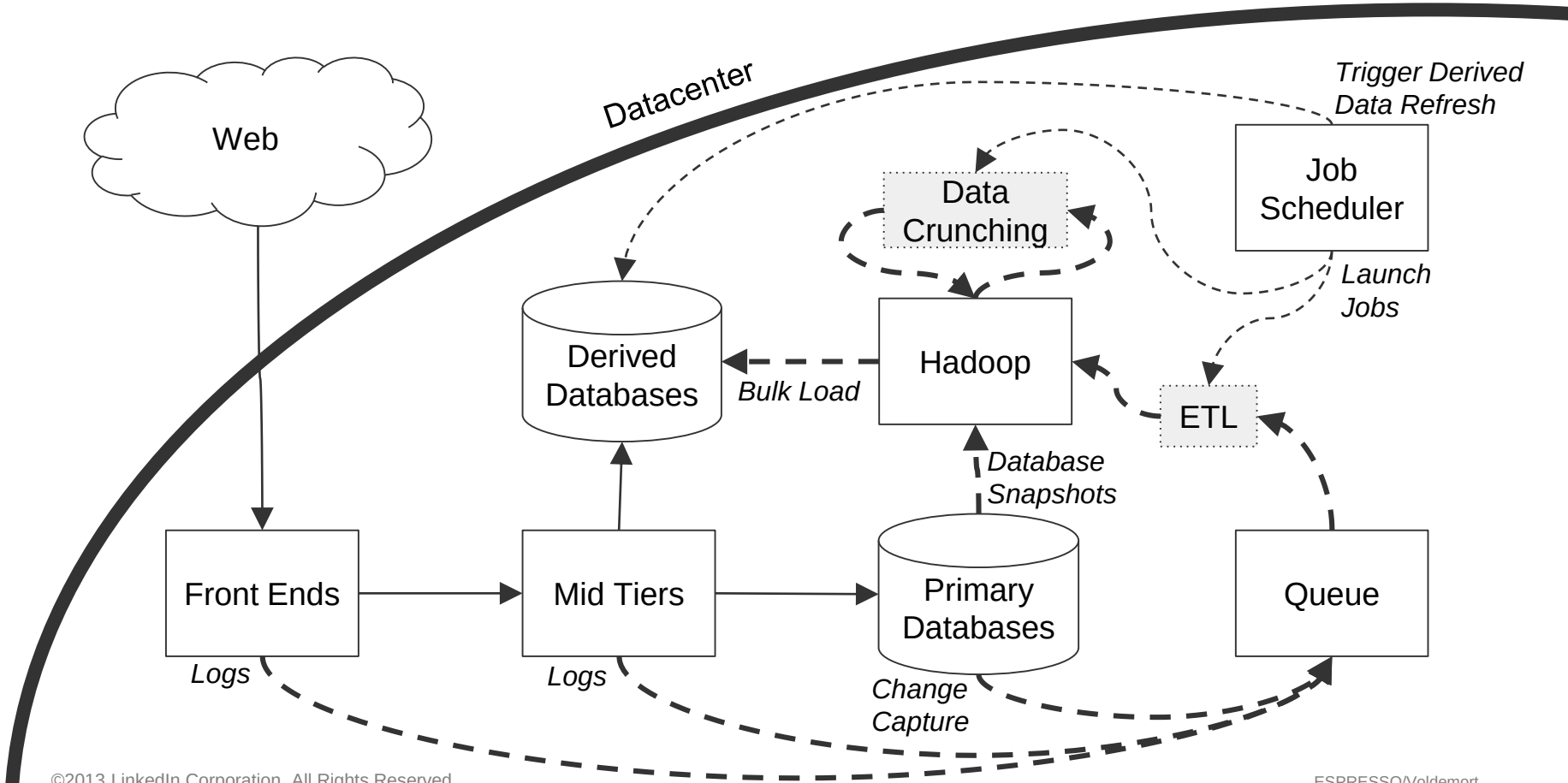
- Scalable offline index construction and data partitioning using MapReduce on Hadoop (Build Phase)
- Complete immutable data set fetched, bulk loaded and swapped for online serving from Voldemort RO. (Push Phase)
- Data set is versioned. Keeps one older version for quick rollback.



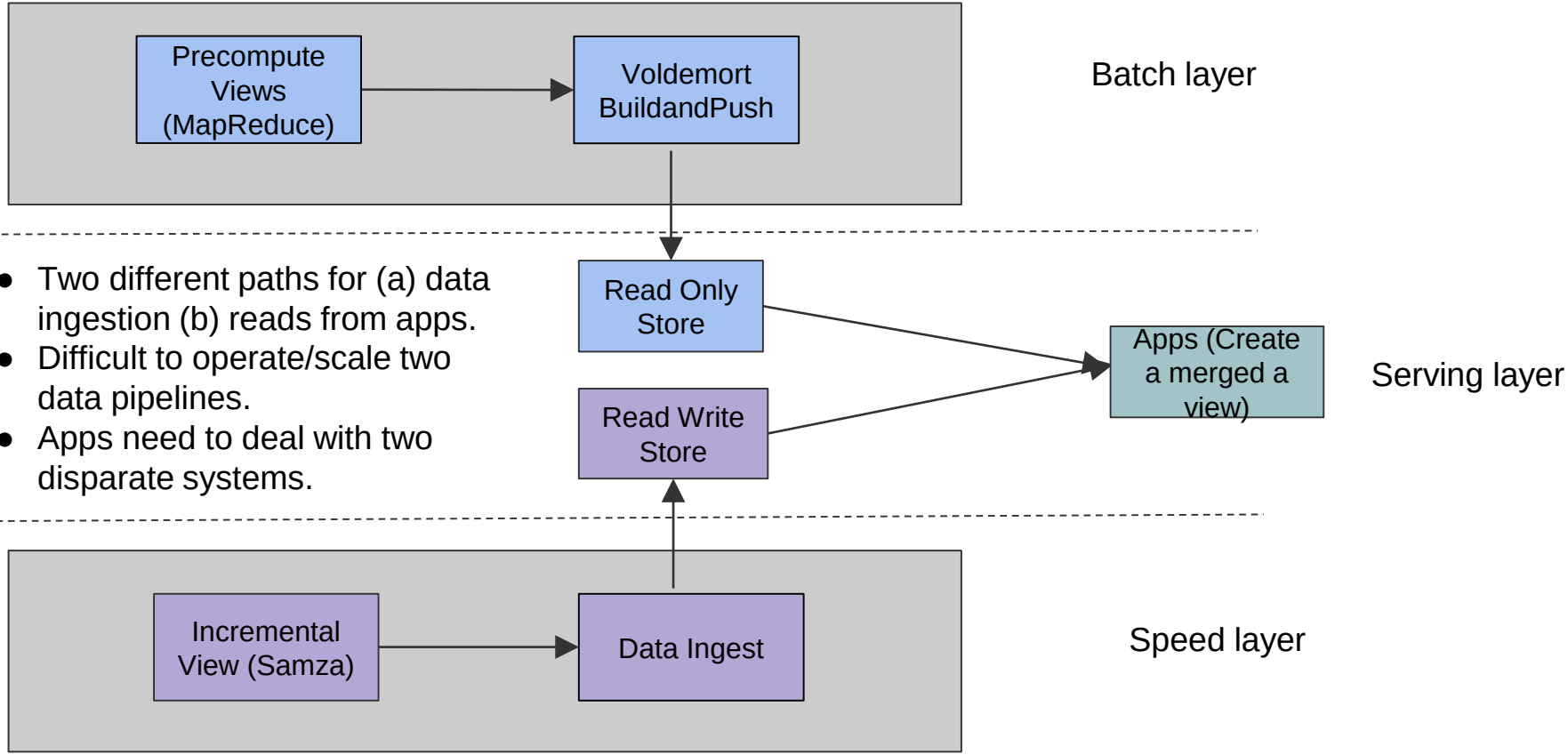
Voldemort RO – Key Details

- General Methodology
  - Index lookup in memory
  - Data lookup as a single SSD seek
  - RO client side latency: p99 < 1.5ms
- Read-Only custom storage engine
  - Pairs of index/data files
  - Index mmaped and mlocked
  - Checksum of checksums for data integrity
  - Index files fetched after data files to take advantage of OS page cache.
- 650 stores, 100TB+ of data moved between Hadoop and Voldemort daily
- Architectural Limitations:
  - Tightly coupled with Hadoop
  - No support for incremental pushes

Recommendation Data Lifecycle



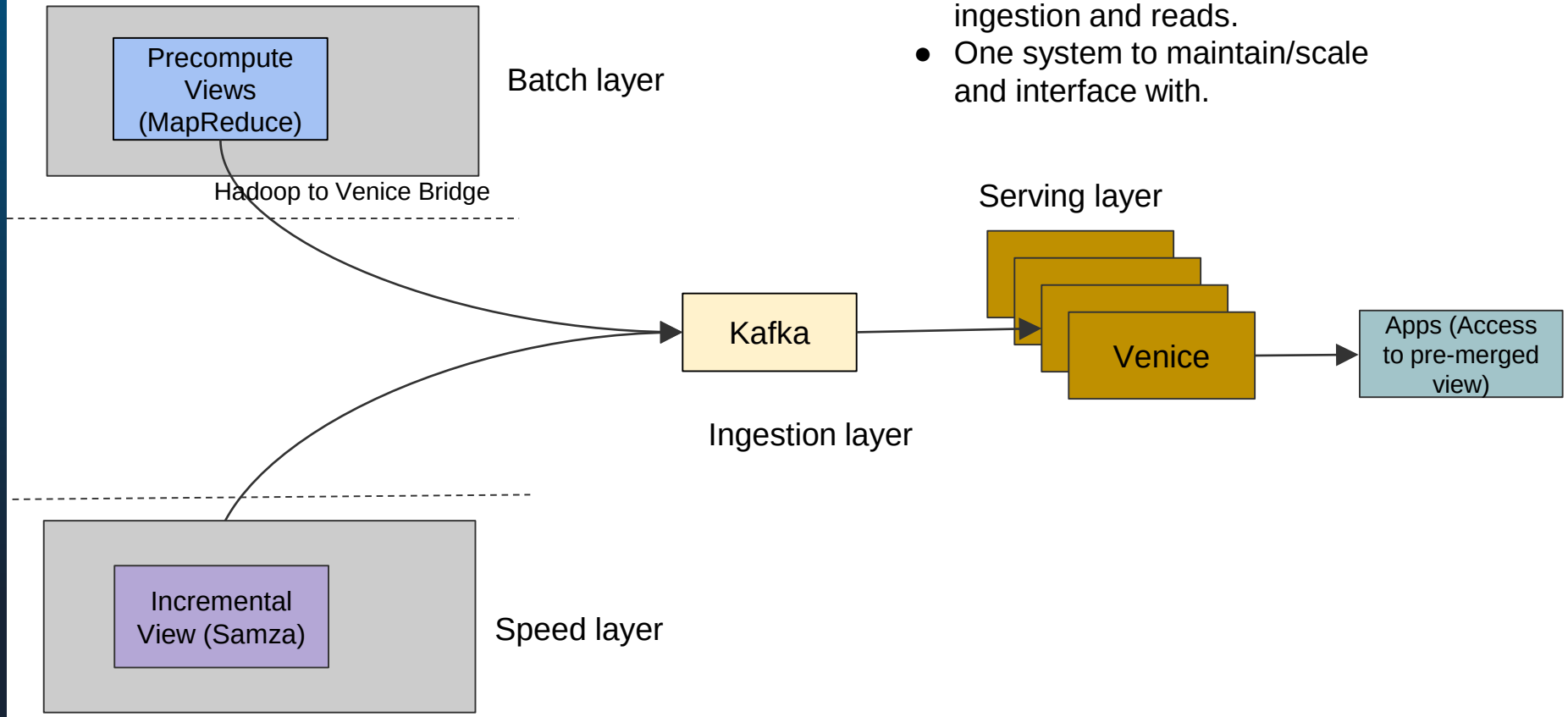
Lambda Architecture @ LinkedIn



- Two different paths for (a) data ingestion (b) reads from apps.
- Difficult to operate/scale two data pipelines.
- Apps need to deal with two disparate systems.

Beyond Lambda

- One path: both for data ingestion and reads.
- One system to maintain/scale and interface with.

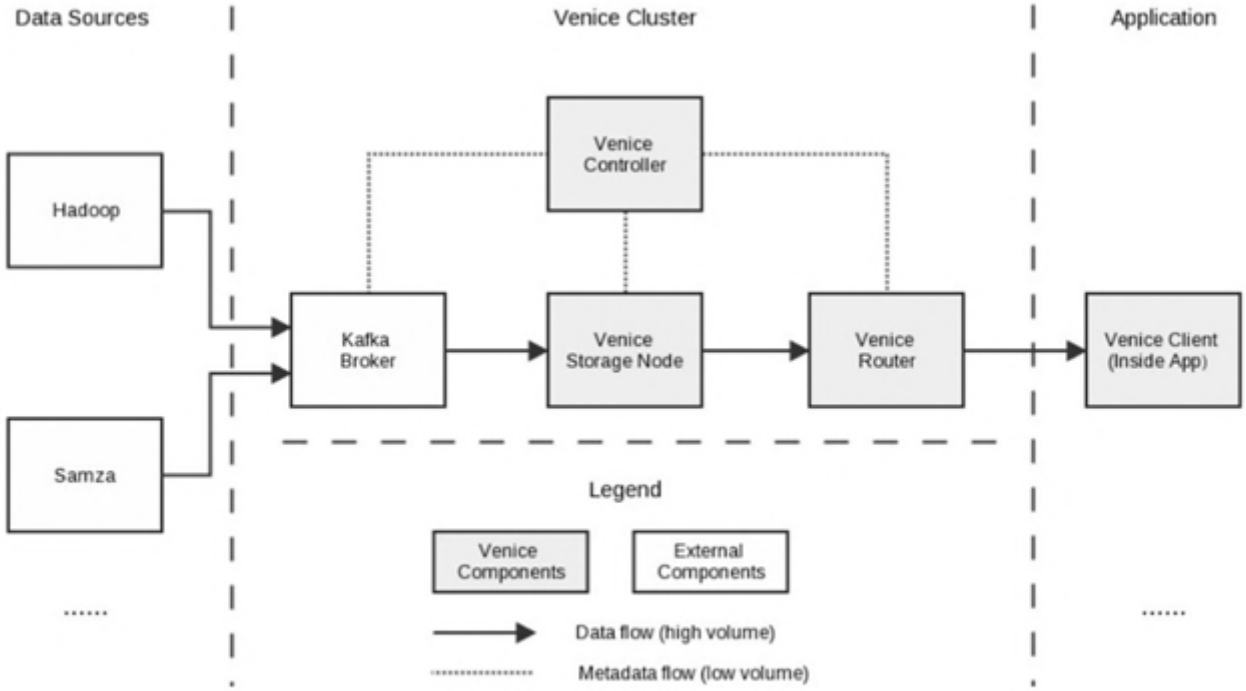


Venice

- Asynchronous derived data serving platform which provides :
  - **High throughput ingestion** from processing platforms like Hadoop, Samza etc.
  - **Low latency** key/value lookups

- Unified solution for serving of derived data
  - Handle batch and stream processing cases
  - Easy to operate
  - Support both Lambda and Kappa Architecture.

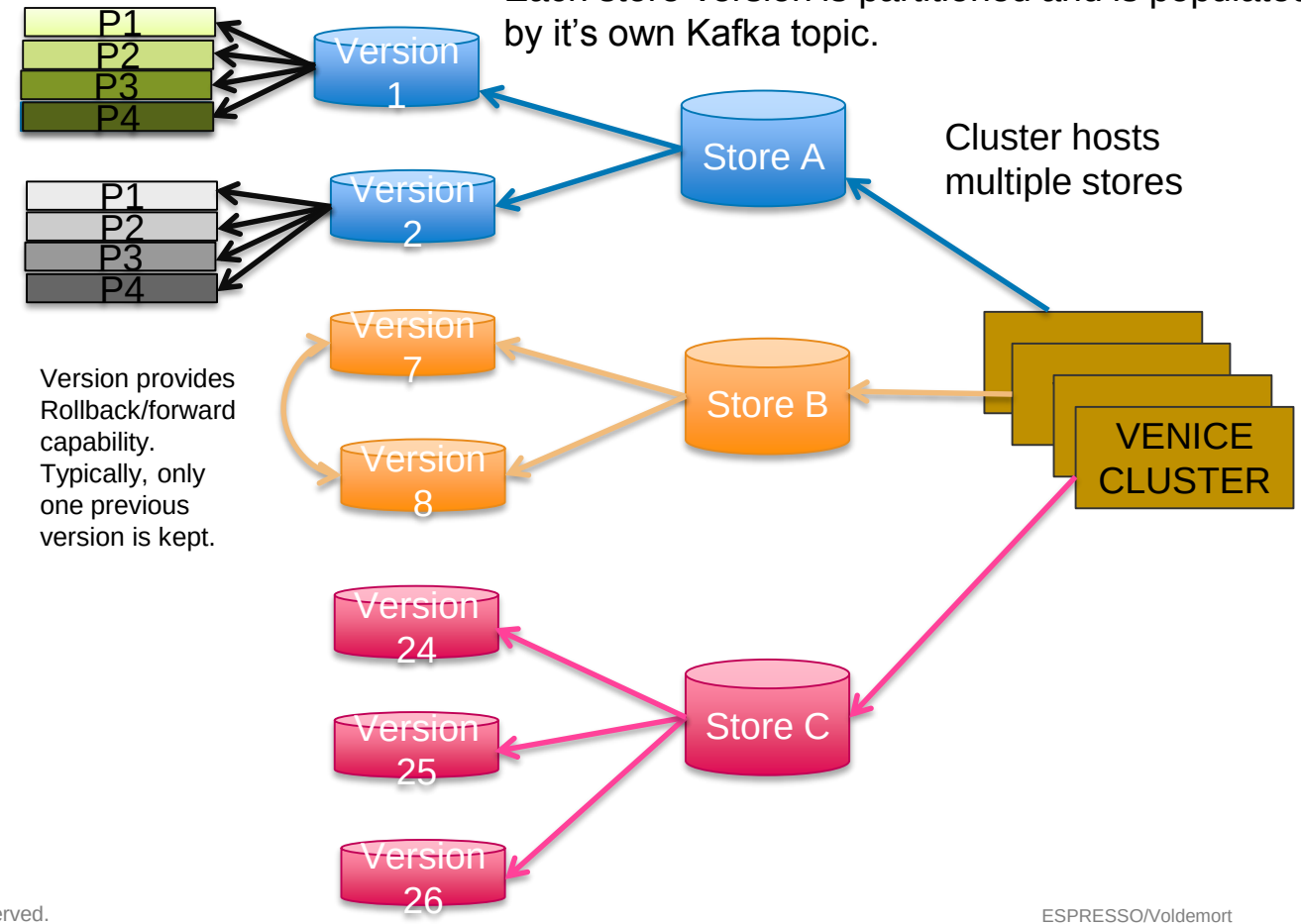
Venice – High Level Architecture



Venice architecture

Store-version

A store can have many versions.
Each store-version is partitioned and is populated by it's own Kafka topic.

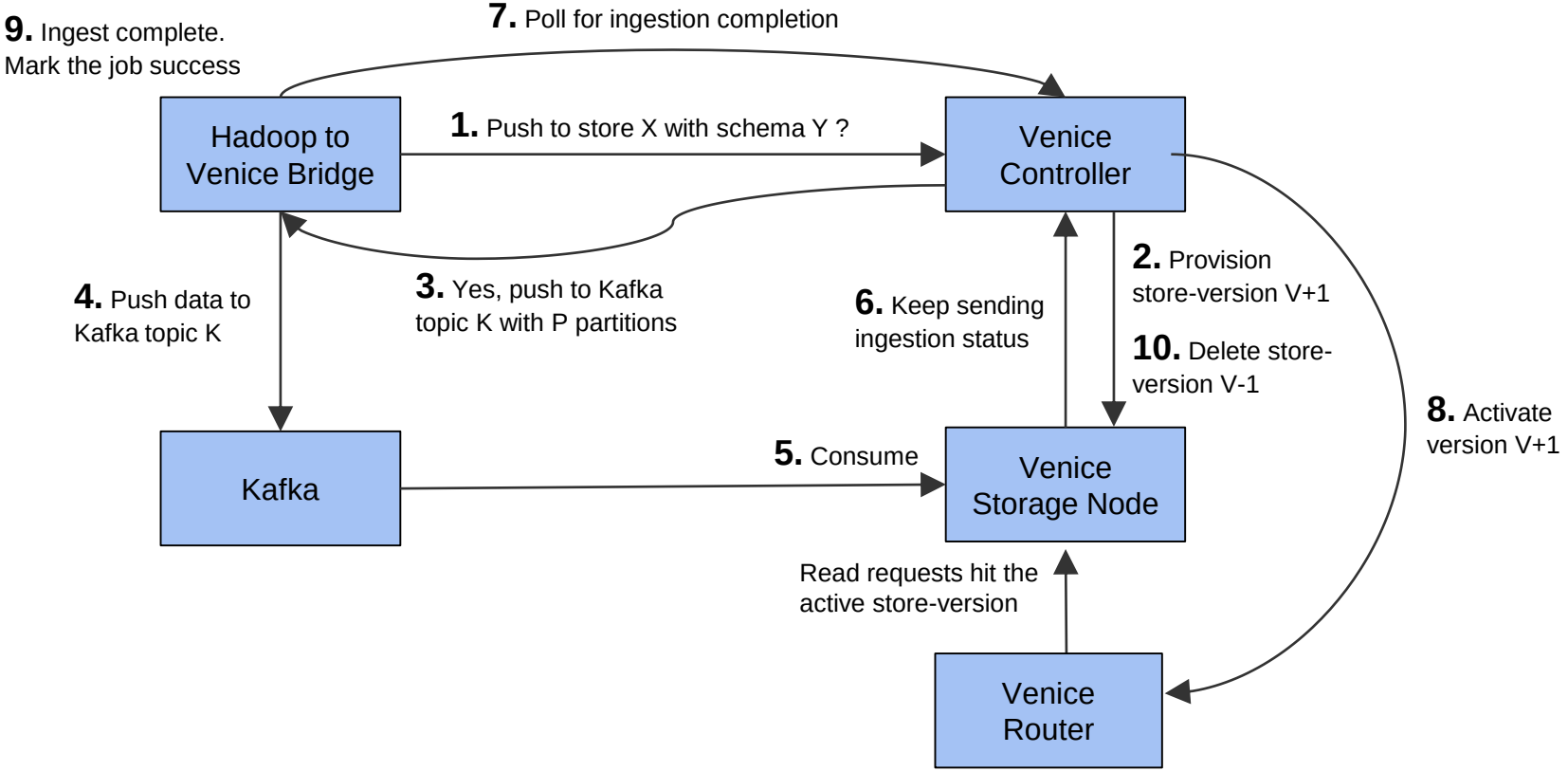


Version provides Rollback/forward capability. Typically, only one previous version is kept.

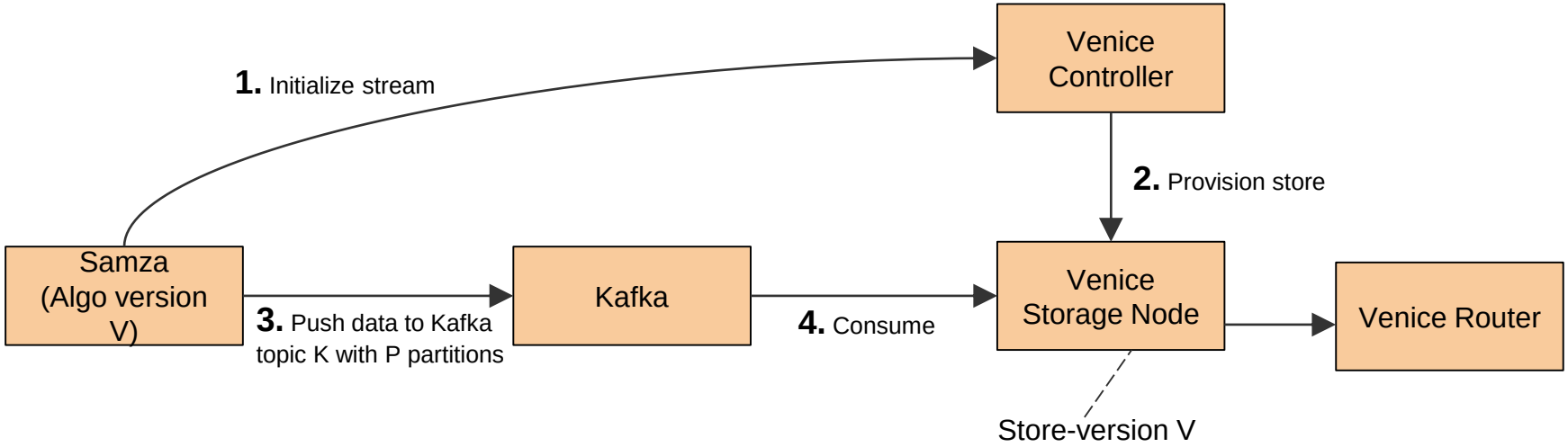
Partitions have replicas. Partitions and replicas get distributed across nodes on a cluster.

Venice – Three trick pony

Batch support



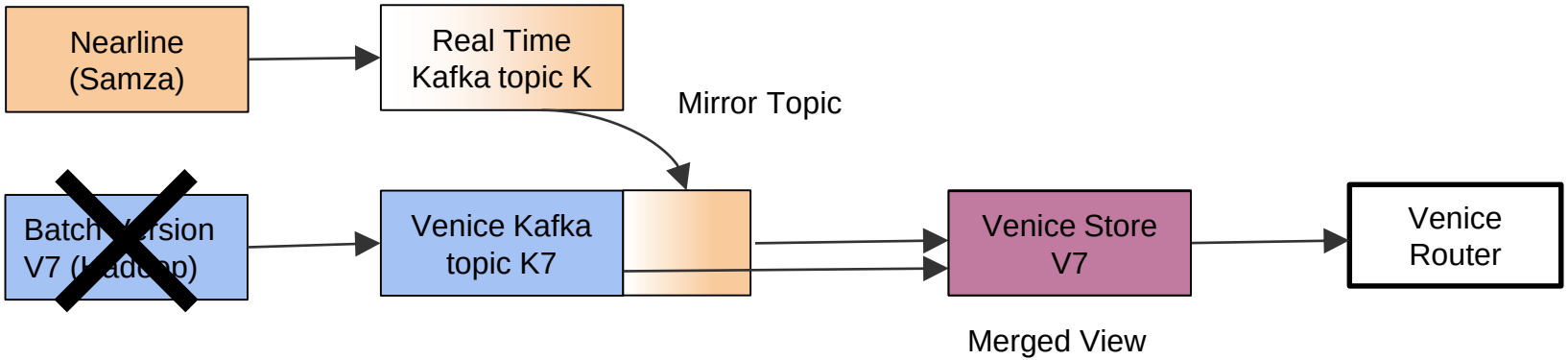
Nearline Support



- Workflow is similar to that of the batch support.
- Venice views both batch and streaming systems as the same.
- Samza stream will be consumed by Venice storage nodes and written to a versioned store. Quotas/Throttling to not affect live queries.
- New algorithm to be processed and stored in a new store.

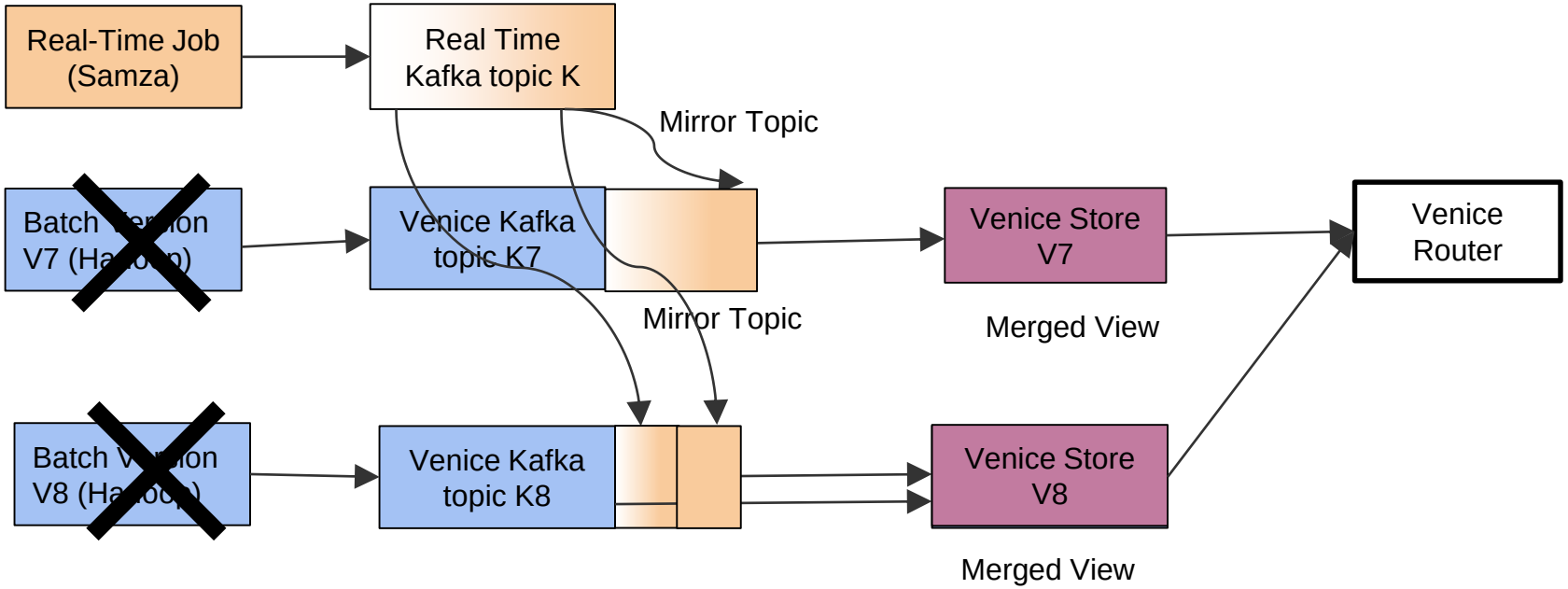
Hybrid support (Batch+Nearline)

- Steady state – In between bulk loads



Hybrid support (Batch+Nearline)

1. Offline bulkload into a new store-version
2. Offline bulkload finished, start buffer replay
3. Replay caught up, router switches to new store-version

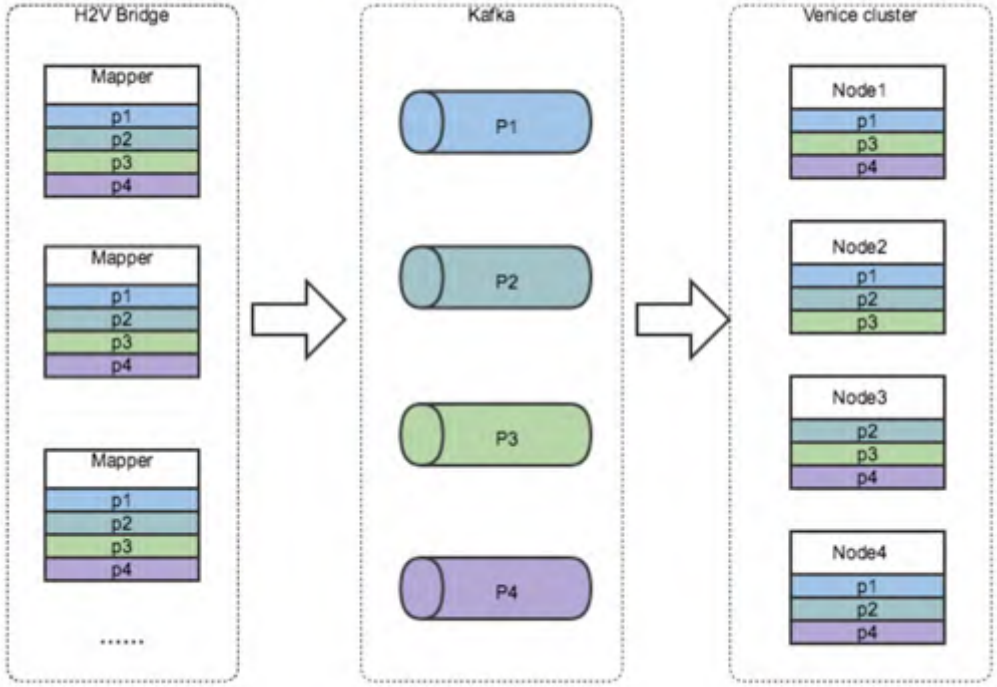


Challenges

- Challenges in building Venice:
  - High throughput Ingest Consumption
  - Data Guarantees
  - Dynamic Topic Lifecycle Management (creation/deletion)
  - Low read latency

Dataset Partitioning & Ingestion

- Each store-version is partitioned
- **1-to-1 mapping** between Venice and Kafka logical partitions
- Each dataset version has its own Kafka topic
- Controller decides partition assignment and tells storage nodes
- On storage node, two separate thread pools (a) to pull data out of Kafka and (b) process and write to the storage engine
- Cleanup of old store dataset and corresponding topic after the new version is swapped and is being served



Scenario with four machines, a dataset with four partitions, and a replication factor of 3.

Dataset Validation

Handling missing and duplicate data

1. Before producing to any given partition, a producer sends a control message in order to uniquely identify itself before producing regular messages.
2. Then, on each produced messages, the producer includes some sequence number in the message's metadata. There is a distinct sequence number for each partition, and it is incremented by one for each new message.
3. The consumer keeps track of the last sequence number seen for each unique producer/partition combination.
    4. Gaps in sequence signal **missing data**.
    5. **Duplicates** can be safely ignored.
6. **Checksum** computation to signal **corrupt data**.

Dataset Validation

7. For hybrid case, use configurable log compaction point to ensure most recent data is never compacted. Storage node lenient when ingesting records for more than a certain threshold.

Early Wins and Future Prospects

- Early Wins !
  - Venice data ingest pipeline ~25% faster than Voldemort (further speedup expected through the year)
  - Read latency comparable to that of Voldemort (p99 ~4-5 ms).
  - Ease of operability - cluster maintenance, expansion etc. are much easier.

- Some thoughts around what might be next:
  - Priority topic ingestion
  - Self-throttling mechanism
  - Auto-rewind capabilities based on offset lag
  - Limited server side transforms (may be ??)

Questions?

(We’re hiring!)