KAPPA+

Moving from Lambda and Kappa Architectures to Kappa+ at UBER

ROSHAN NAIK

FLINK FORWARD 2019 – SAN FRANCISCO

PROBLEM

Realtime jobs often need an offline counterpart:

- **Backfill** – Retroactively fix, or recompute values once all data has arrived.
- **Offline Experimentation & Testing:** Before taking the job online.
- **Online and offline feature generation:** for ML.
- **Bootstrapping State:** for realtime jobs.

CURRENT SOLUTIONS

1. **Lambda Architecture [2011]**
   - Nathan Marz (Creator of Apache Storm)
     - "How to beat the CAP theorem"
   - Evidence of prior art [1983]:
     - Butler Lampson (Turing Award Laureate)
     - "Hints for Computer System Design" – Xerox PARC
   - Core Idea: Streaming job for realtime processing. Batch job for offline processing.
2. **Kappa Architecture [2014]**
   - Jay Krepps (Creator of Kafka, CoFounder/CEO Confluent)
     - "Questioning the Lambda Architecture"
   - Core Idea: Long data retention in Kafka. Replay using realtime code from an older point.

LIMITATIONS : LAMBDA ARCHITECTURE

- Maintain dual code for Batch and Streaming
- Batch APIs often lack required constructs (e.g. sliding windows)
- **Variation:** Unified API : SQL / Beam. – Offline job run in batch mode
- **Limitations of Batch mode (e.g. Spark):**
  - Divide large jobs into smaller ones to limit resource consumption
  - Manual/automated sub job co-ordination
  - Windows that span batch boundaries are problematic

LIMITATIONS : KAPPA ARCHITECTURE

- **Longer retention in Kafka: Expensive, Infeasible**
  - Kafka not really a data warehouse. More expensive than HDFS.
  - Retention beyond a few days not feasible. Single node storage limits partition size.
- **Workaround 1:** Tiered Storage (Pulsar)
  - Data duplication: Usually need a separate queryable copy in Hive/warehouse.
  - Low utilization: old data accessed only by Backfill jobs.
- **Workaround 2:** Mini batches
  - Load small batches into Kafka and process one batch at a time.
  - Sort before loading in Kafka. Try to recreate original arrival order.
  - Expensive: Copying to Kafka and sorting are both costly.
- **Issues when using multiple sources**
  - Low volume topic drains faster → messes up windowing → dropped data or OOM

DESIRED CHARACTERISTICS

- Reuse code for Online and Offline Processing.
- Windowing should work well in offline mode as well.
- No splitting jobs. Single job should processes any amount of data.
- Hardware requirements should not balloon with size of input data.
- Not have to rewrite jobs using new APIs.
- Efficient.

KAPPA+

Introducing the Architecture

KEY CHANGE IN PERSPECTIVE

- **Decoupling Concepts:**
  - Bounded vs Unbounded (Nature of Data)
  - Batch vs Streaming (Nature of Compute)
  - Offline vs Realtime (Mode of Use)
- **Instead of thinking:** How to enable any job in Streaming and Batch mode.
  - Lambda / SQL / Beam / Unified APIs
- **Think:** Limits to job types that can run in Realtime (and Offline) mode.
  - Kappa+

Impact:

- No need to support every type of batch job (departure from Unified API approach).
- Identify the types of jobs to support: Kappa+ job classification system.

ARCHITECTURE

Central Idea - counter intuitive

- Use Streaming compute to process data directly from warehouse. (i.e. not tied to Kafka)

Architectural Components:

1. **Job classification system**
   - 4 categories
2. **Processing model**
   - Same processing basic model with tweaks based on job category

Assumes: Data in warehouse (Hive/Hdfs/etc) is partitioned by time (hourly/daily/etc).

JOB CLASSIFICATION SYSTEM

- **Category 1 : Stateless Jobs**
  - No windowing. Memory not a concern.
  - Data order usually not concern.
- **Category 2 : Windowing with aggregation (Low - Medium Memory)**
  - **Eg:** Aggregated Windows: sum / avg / count / min / max / reduce
  - Retains only aggregate value in each window.
  - Order of data is important. But solvable without need for strict ordering.
- **Category 3 : Windowing with retention (High Memory)**
  - Holds on to all records till window expiration.
  - **Eg.** Joins, pattern analysis within window.
  - Memory requirements much higher than cat 2.
- **Cat 4 : Global Windows with retention**
  - **E.g.** Sorting entire input data. Joins without windowing.
  - Not found in realtime jobs.

PROCESSING MODEL

1. Partially ordered reads

- **Strict Ordering <u>across</u> partitions:** Only one partition at a time, older partitions first.
  - Constrains memory/container requirements to what is needed to process 1 partition.
  - Single job can process any number of partitions with finite resources.
  - Helps windowing correctness.
- **Un-Ordered reads <u>within</u> partition**
  - Read records/files within a partition in any order. <u>Opens up concurrency</u> and high throughputs.
  - Order could be exploited if necessary.

2. Emit watermark when switching to next partition

- Allows Out-Of-Order reads within partition and windowing correctness.

3. Lockstep progression, in case of multiple sources

- All sources move to next partition at the same time.
- Prevents low volume sources from racing ahead.

HANDLING EACH CATEGORY

- Cat 1 : Stateless
  - Nothing special. Set parallelisms based on desired throughput.
- Cat 2 : Windowing with aggregation (Low – Med mem)
  - Employ **memory state backend**.
  - **Windowing parallelism** based on amount of data hosted in memory for one partition. Other parallelisms, based on throughput.
- Cat 3 : Windowing with retention (High Mem)
  - **Either:** Use RocksDB state backend.
  - **Or:** reduce partition size, and use Mem state backend.
  - **Or:** Look into exploiting order within partition.

BENEFITS OVER BATCH

BATCH (SQL/ BEAM/ UNIFIED API)

1. Resource requirements grows with **total** data volume.
   - Tricky to estimate and allocate
2. Split into smaller jobs and coordinate them.
   - Windows that cross batch boundaries are problematic.
3. Results visible after all data is processed.

KAPPA+

1. Resources bounded by amount needed to process **1 partition**.
   - Easier to estimate and allocate
2. Single job can process any number of partitions.
3. Results visible after each partition.

Note: Kappa+ processing model could be adopted in Unified APIs to address these limitations.

IMPLEMENTATION

Architecture is not tied any Streaming Engine.

ADOPTING KAPPA+ ON STREAMING ENGINES

- **No new APIs.**

- **Hdfs/Hive/etc. Sources need behavioral change:**
  1. One partition at a time, older partitions first.
  2. Concurrent reads within partition.
  3. Lock step progression in case of multiple sources.
     - Kafka source needs to only support #3 since data is already in order.
- **Watermarking:**
  - Emit watermarks at the end of partition to flush windows**.**

A JOB SUPPORTING REALTIME & OFFLINE

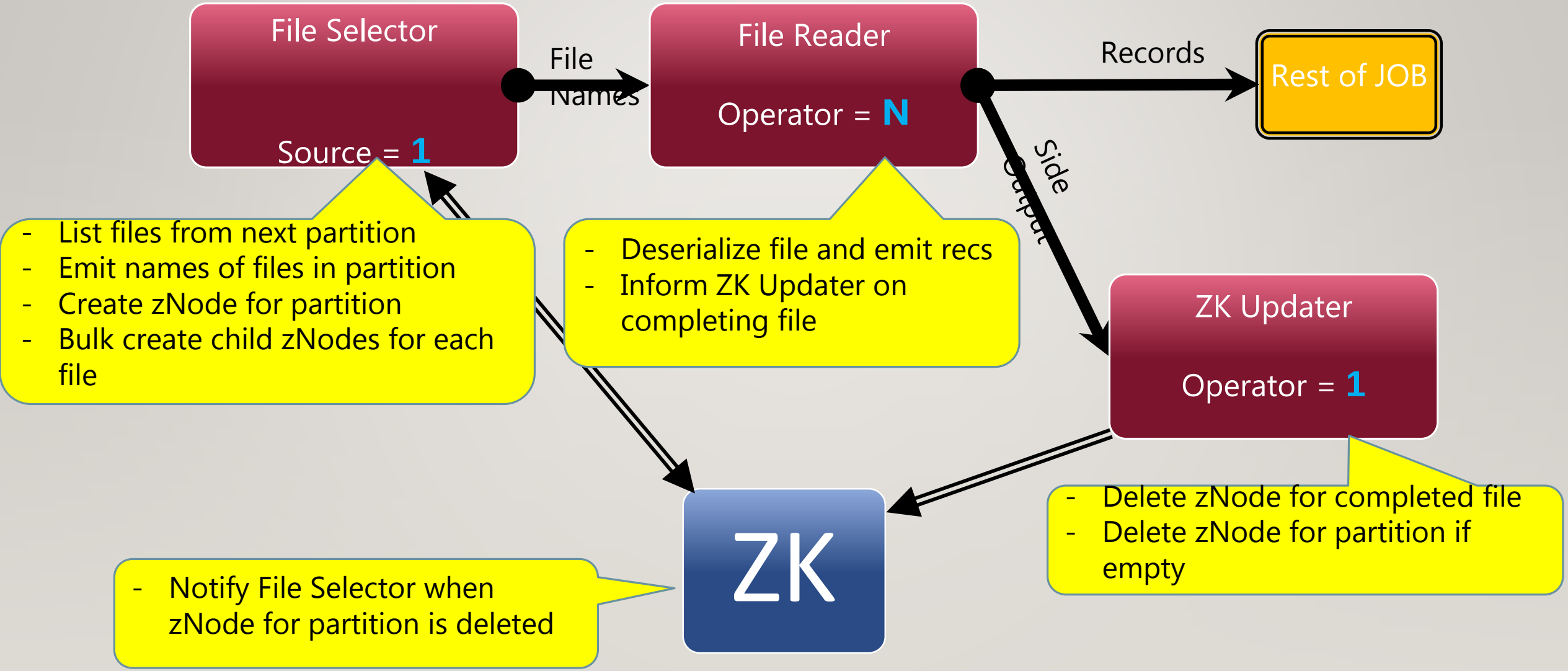
```
dataSource = offlineMode ? hiveSource : kafkaSource;
watermaker = offlineMode ? new OfflineWM() : new RealtimeWM();
dataSource.assignWatermarkGenerator(watermarker);

// Same logic. Adjust parallelisms for offline & online modes.
job = dataSource.transform(..)
                .filter(..)
                .keyBy(..)
                .window(..)
                ...
```

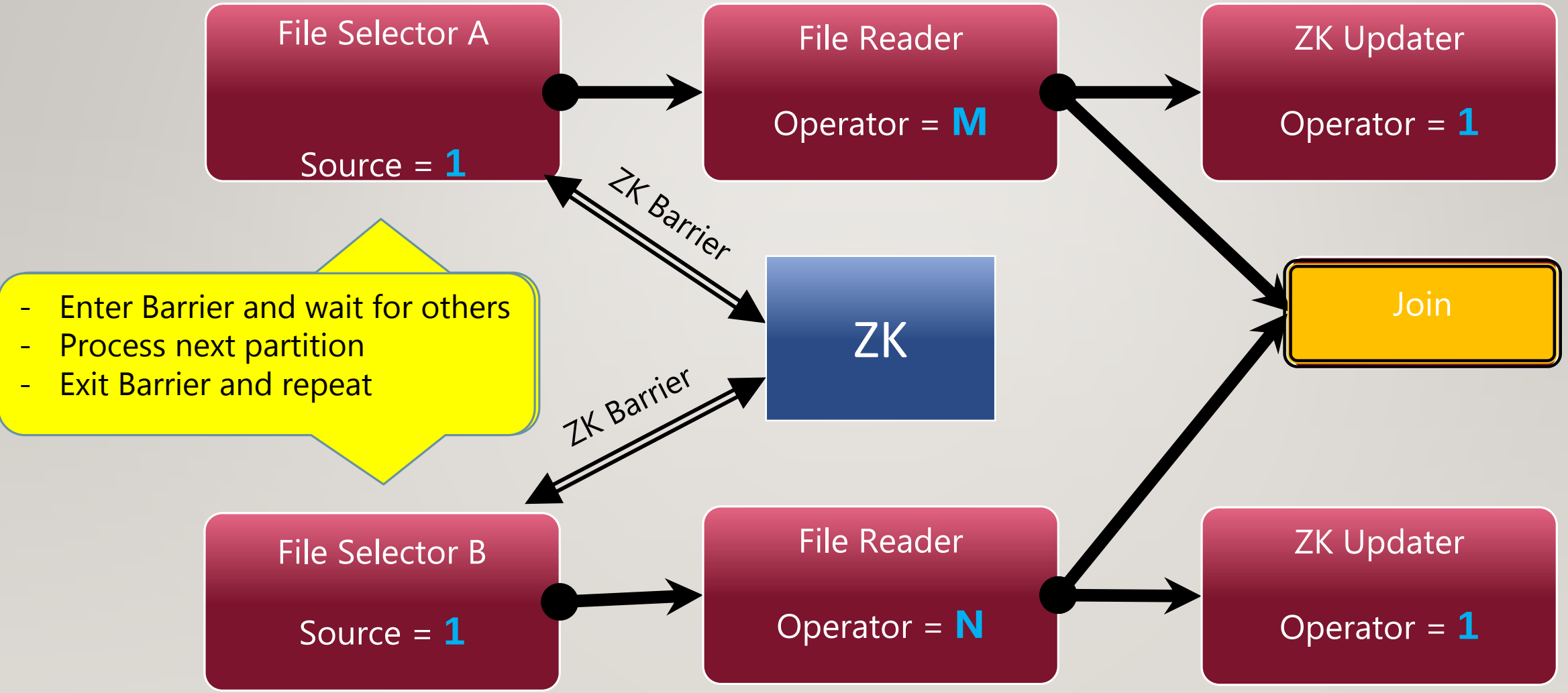
KAPPA+ ON FLINK

UBER internal Hive (/HDFS) source with Kappa+ support.

ONE PARTITION AT A TIME & CONCURRENT READS



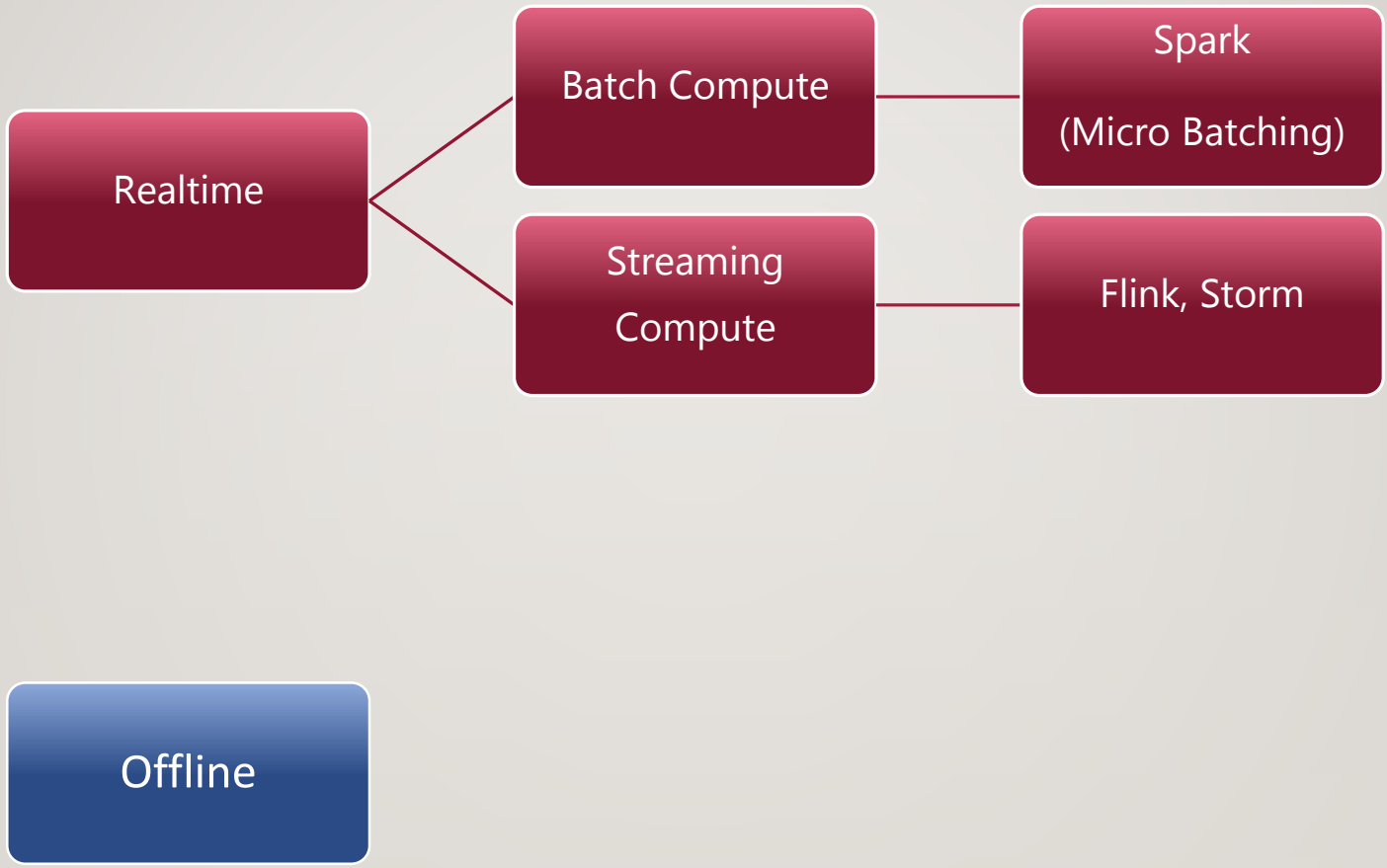
MULTI SOURCE LOCK STEP PROGRESSION



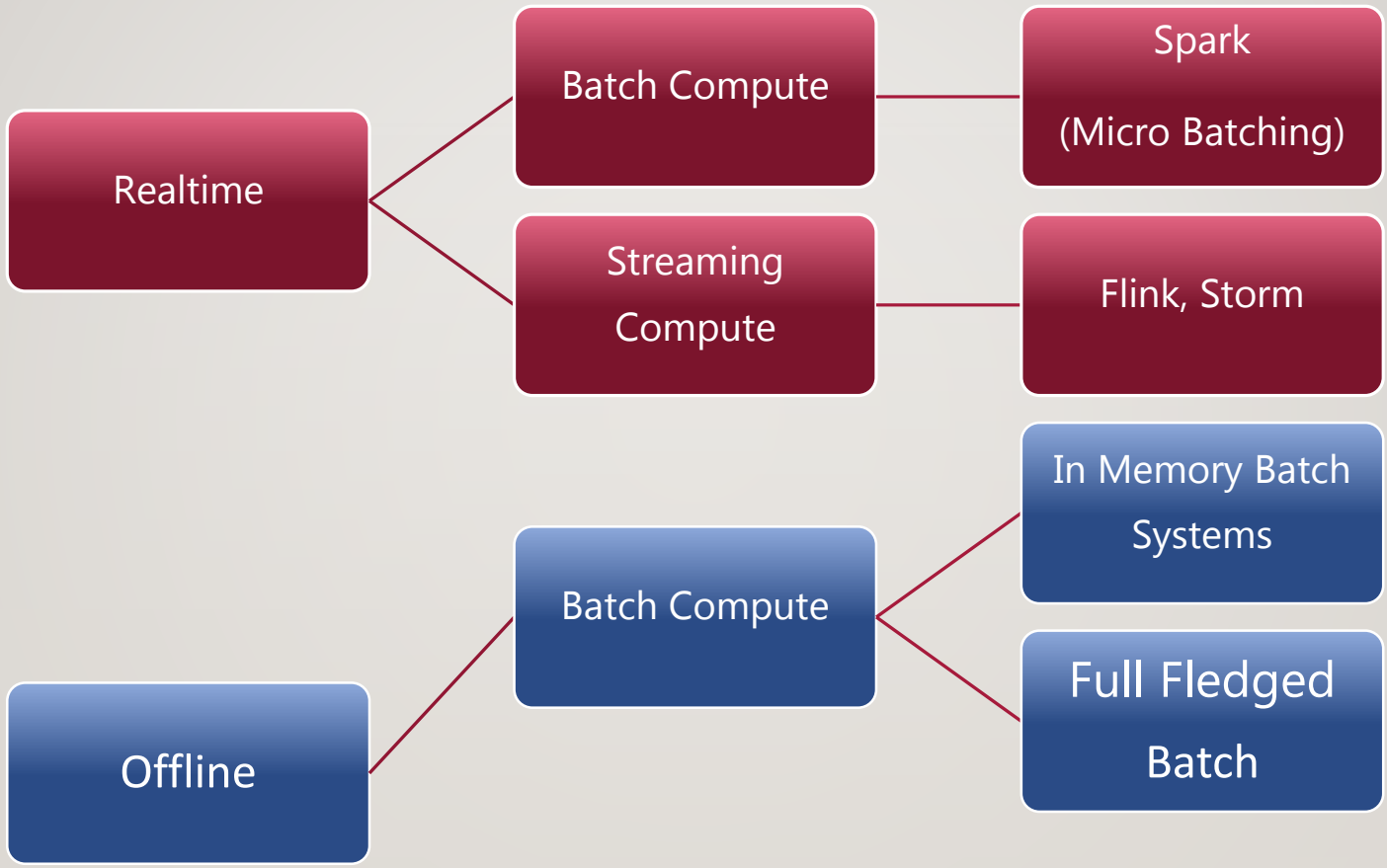
DETAILS

1. **Time Skew:** If arrival time is used for partitioning data in the warehouse, instead of event creation time (used by job). There can be two types of data skews:
   - **Forward skew:** Some events move into a future partition. For example due to late arrival.
     - Could lead to appearance of missing data.
     - Consider processing an additional partition after the last one, if this is an issue.
   - **Backward skew:** Some events moving into an older partition.
     - Can lead to appearance of data loss. As the events are not in the partition that you processed.
     - Improper watermarking can close Windows prematurely and cause data loss.
2. **Differing partition strategies:** Job has two sources. First source reads Hive table with daily partitions, second source reads table with hourly partition.
   - **Solution:** Watermark progression dictated by daily (i.e. larger) partition
3. May need to **throttle** throughput of offline job if writing to production critical destination.

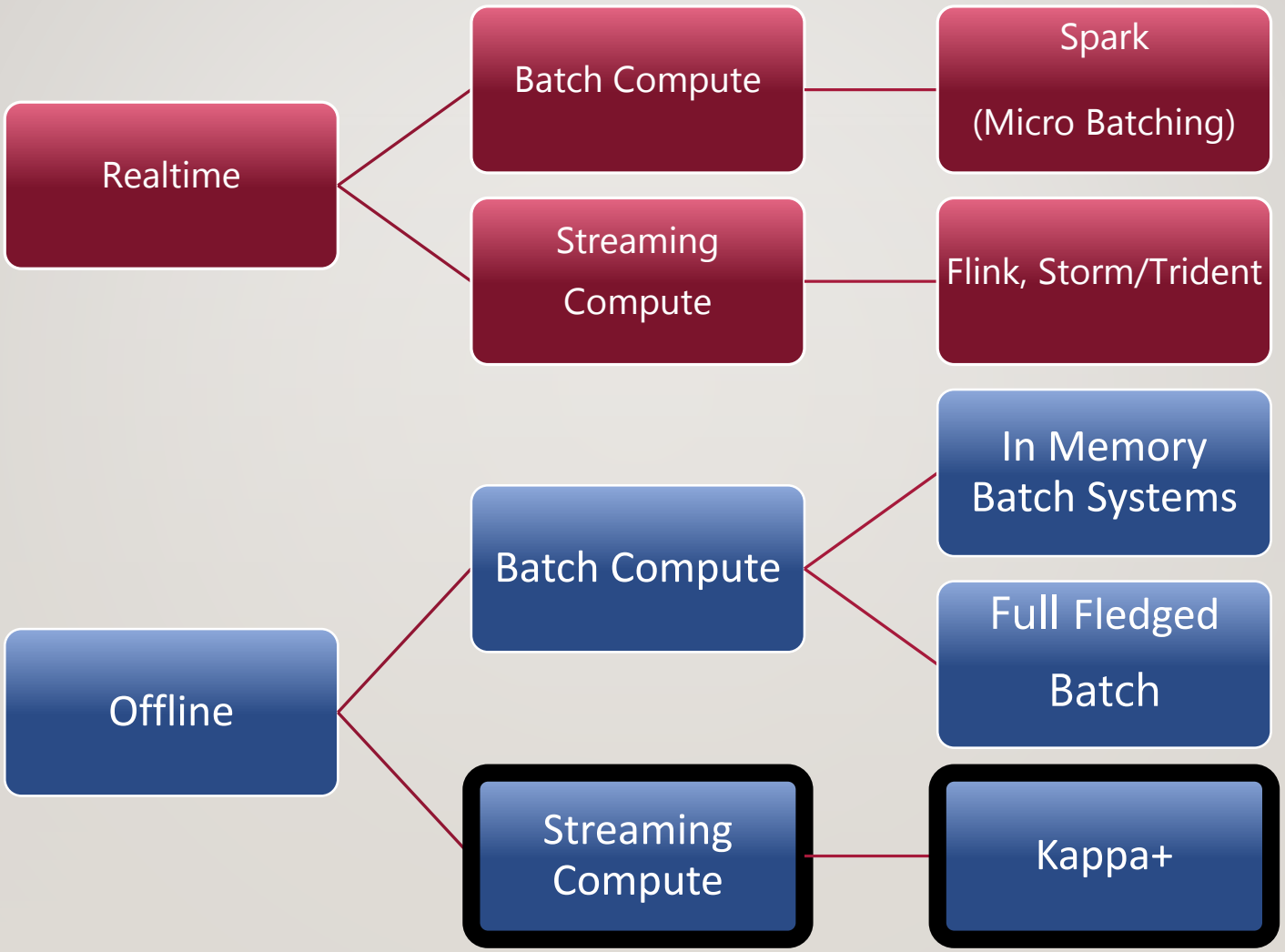
DISTRIBUTED COMPUTING



DISTRIBUTED COMPUTING



DISTRIBUTED COMPUTING



QUESTIONS

Email: roshan@uber.com

Twitter: @naikrosh ,

@UberEng

UBER Engineering Blog: eng.uber.com

UBER is hiring!! Realtime Platform needs your expertise!