kafka summit

tinder

Sept 2nd, 2019

Patterns and Pitfalls When Marrying a Microservices Architecture with an Event Driven Architecture

Who Am I: Kyle Bendickson*

Data Software Engineer, Machine Learning @ Tinder

(all views are my own...)

Preferred Pronouns:
He/Him

On the Pride@Tinder ERG

Extensive Work in Large Scale Real Time ETLs

ML Infrastructure / Modeling / Personalization in Applications

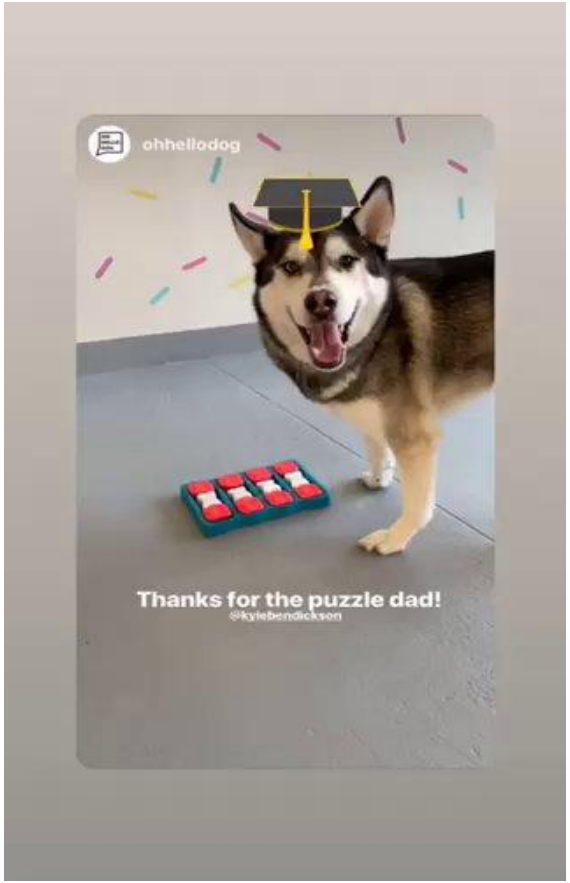
Ad-Hoc Developer Advocate / Evangelist for All Things Data, Both Big and Fast, @Tinder

Also, A Dog Dad

Who I think you wonderful people are:

- Respectful* of others
- Have worked in a microservices architecture
- Aware of the basics of Apache Kafka
- Can read basic and speak SQL
- Want to use tools from the Apache Ecosystem to scale out and decouple your application components and infrastructure via an Event-Driven Application Architecture. Particularly with Kafka as a Shared Source of Truth.

So what will we cover?



Covering:

- Quick review of Tinder + some of our scale
- Discuss Quickfire, Tinder's current unified data event platform for client and server side data, as well as its continued evolution
- Discuss some patterns which have helped us move to a more event-driven architecture via Kafka, as well as pitfalls along the way
- Why you should embrace SQL

A Quick Review of the Classic Tinder Swiping Experience

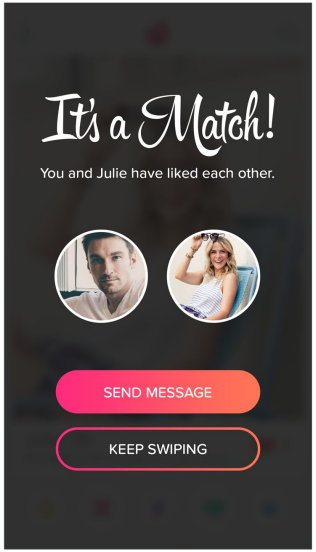
tinder

The Original Swiping Experience

Strictly One Card at a Time

Swipe Left to Nope

Swipe Right to Like



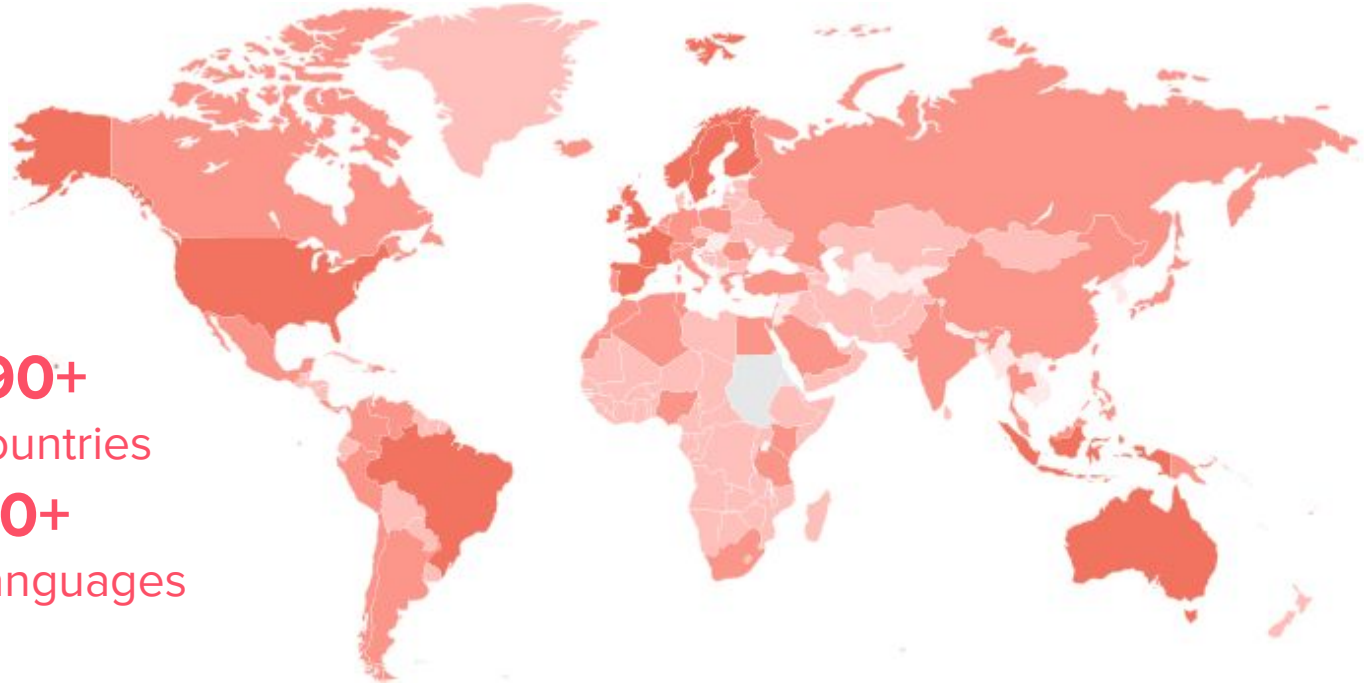
If both swipers like, It's a Match!

A Quick Primer on The Quickfire Pipeline - Our Primary Unified Pipeline For Client and Server-Side Events

Global Scale

2.2B+
swipes daily

30B
matches



190+
countries

40+
languages

More Tinder Scale

~86B
Events/Day

~1M
Events/Sec

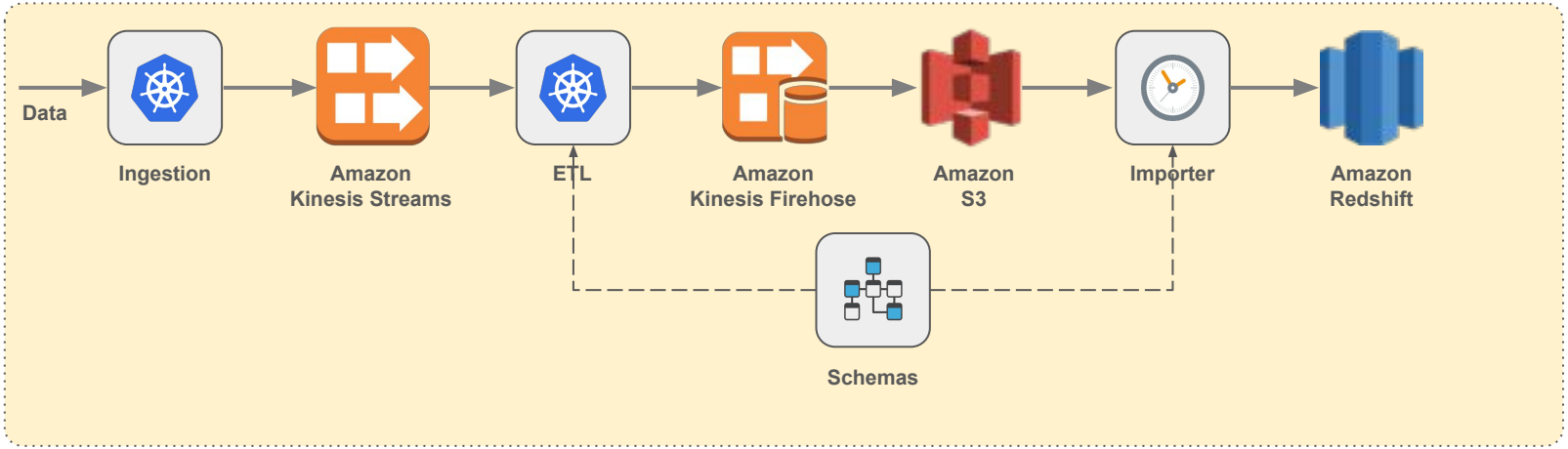
>40TB
Data/Day

Kafka delivers the performance and throughput needed to sustain this scale of data processing

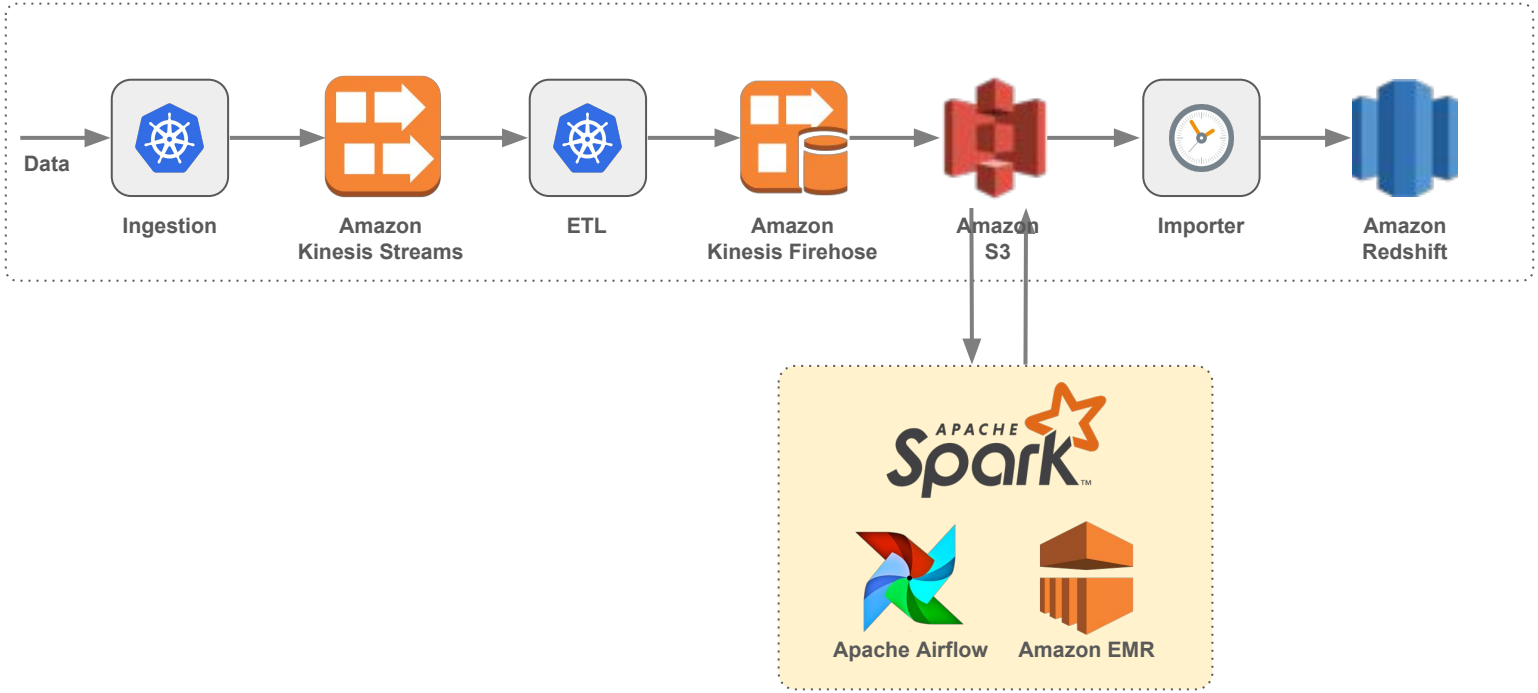
~90%
Cost Savings

Using Kafka over SQS / Kinesis / SNS / Lambda / Kinesis Firehoses, etc saves us approximately 90% on costs.*

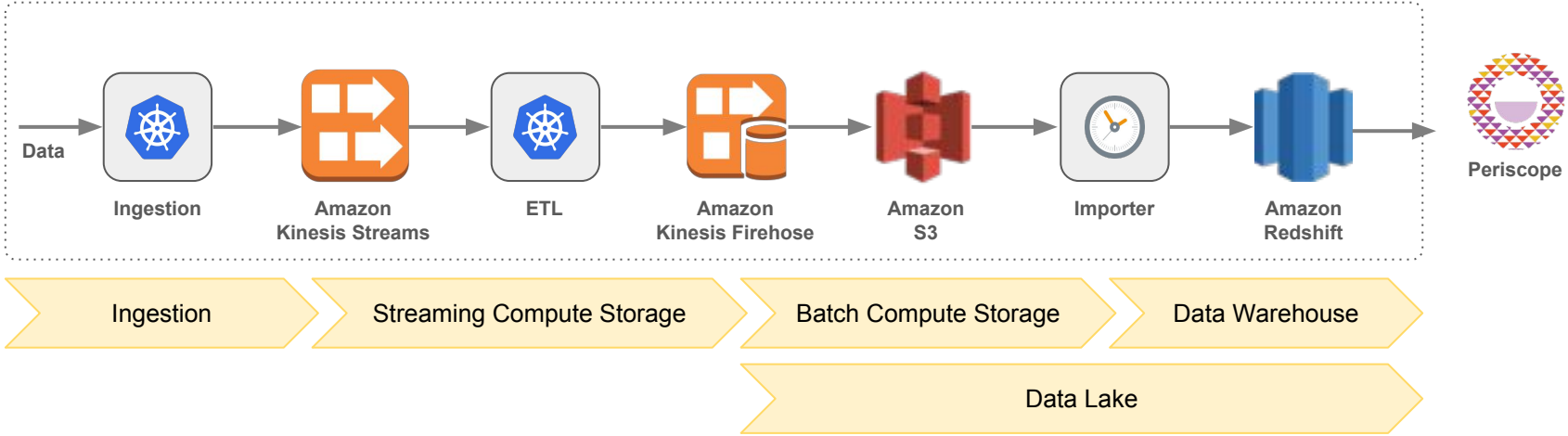
Look Back / Data Backbone



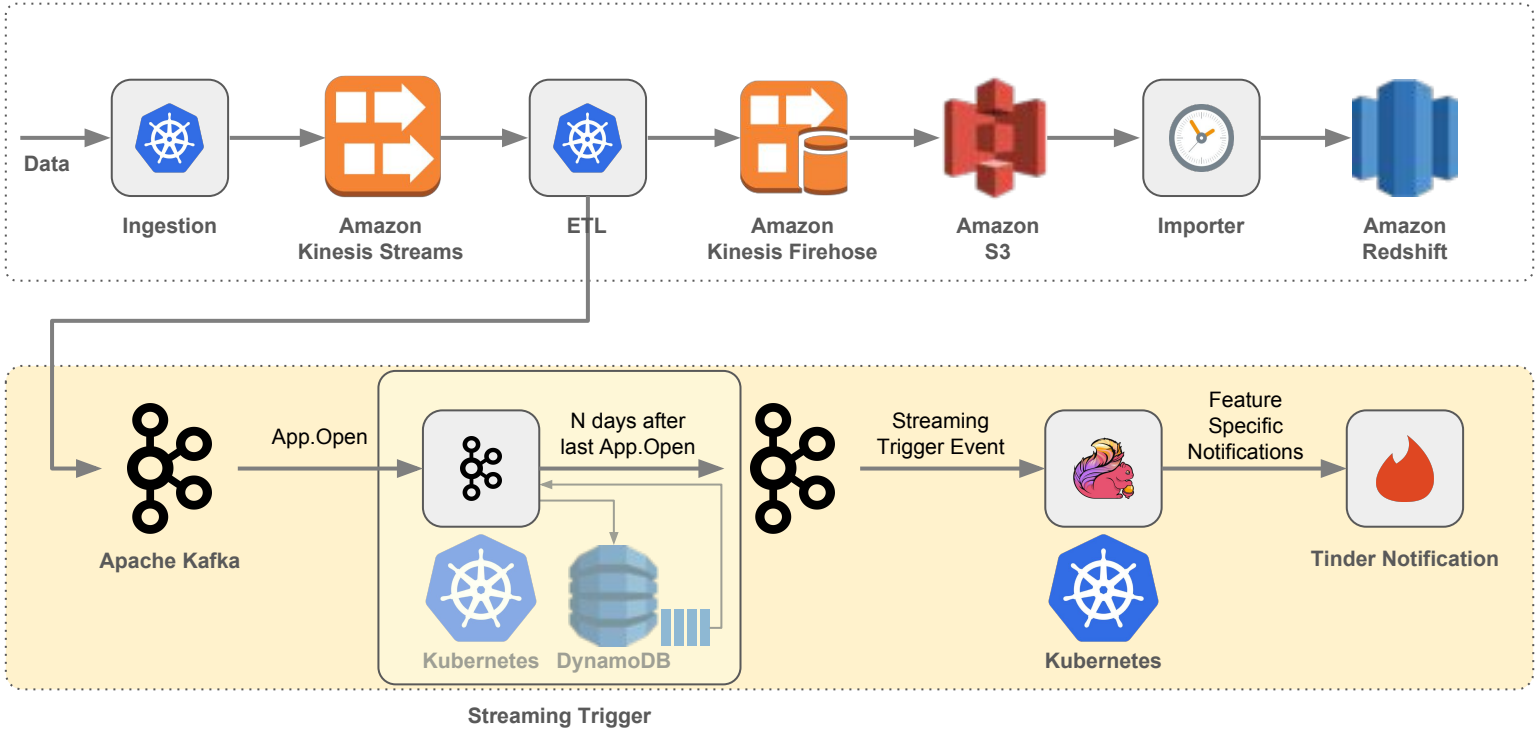
Look Back / Batch Compute



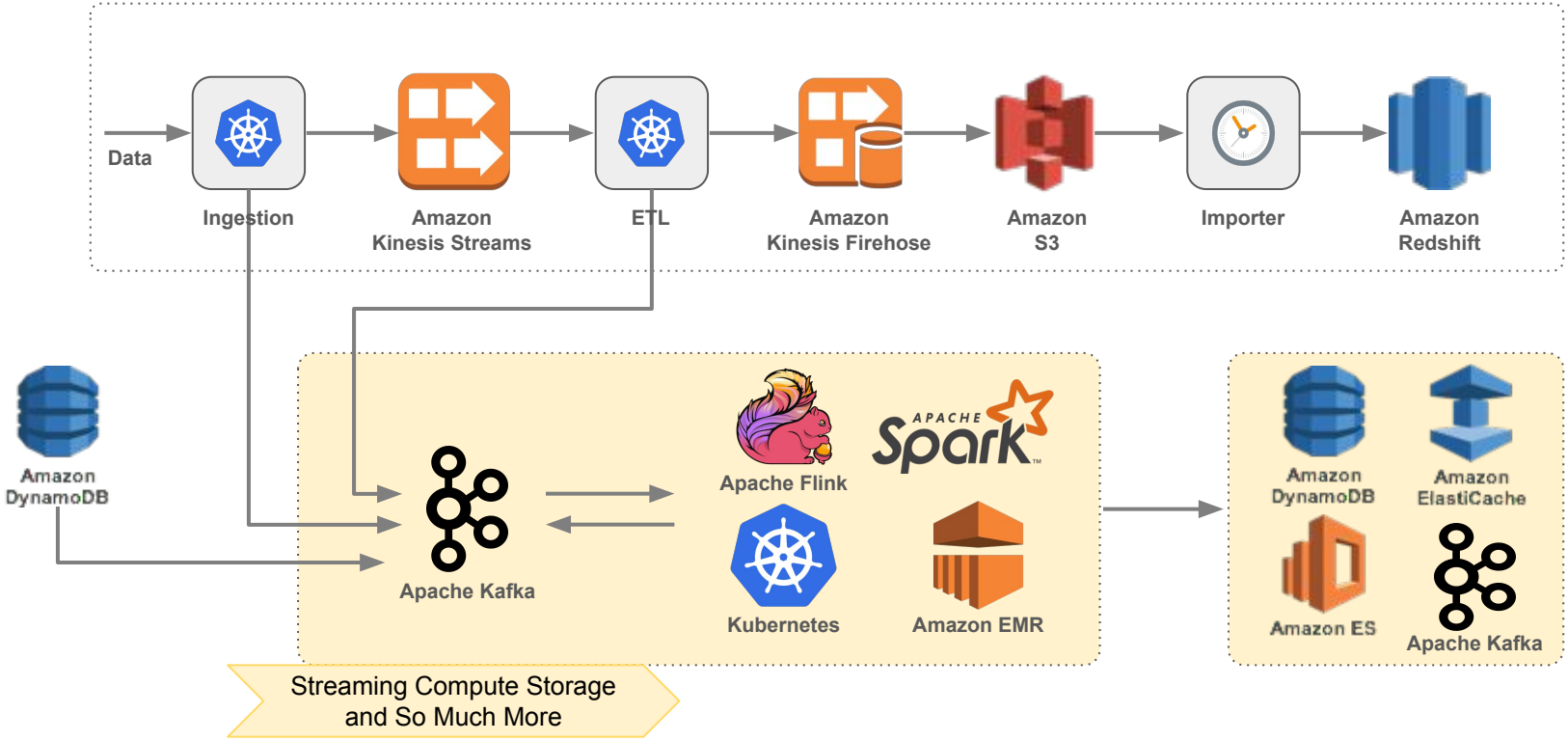
Look Back at the Quickfire Pipeline - Ingestion to Data Lake and Warehouse.



Event-Driven Apps - Engagement & Notification



Look Now / Streaming + Batch Compute



Motivation to Move to an Event-Driven Data Processing Model

Why Event-Driven?

Treat the Events as First Class Citizens in all systems throughout Tinder.

- These events are the culmination of all user experiences + happenings in the app.

Why Event-Driven?

Allow backend developers access to rich amounts of intelligence data that has traditionally been used for offline analysis

- Getting this data online would otherwise add additional RPC calls and further microservices fanout for each HTTP client-side request

Why Event-Driven?

Turning the Database Inside Out!

Allow state that is typically managed by individual services to be shared, immutably, in the transaction log to allow decoupled access by other services... including FUTURE services that have yet to be conceived.

Why Event-Driven?

The Ability to Capture and Join Data from Many Very Different Sources, in Real-Time.

- ***Query*** for user info originally from one location (e.g. Postgres), ***hydrate with information from another table*** (e.g. DynamoDB), and then ***continue the event processing pipeline, all from within Kafka*** via Kafka Streams and KSQL.

Why Event-Driven?

Replace “Fire and Forget” calls from a microservices architecture into a more robust, fault-tolerant stream of data for interested applications to consume the relevant data as needed.

Why Event-Driven: The Unbounded Nature of Data Streams

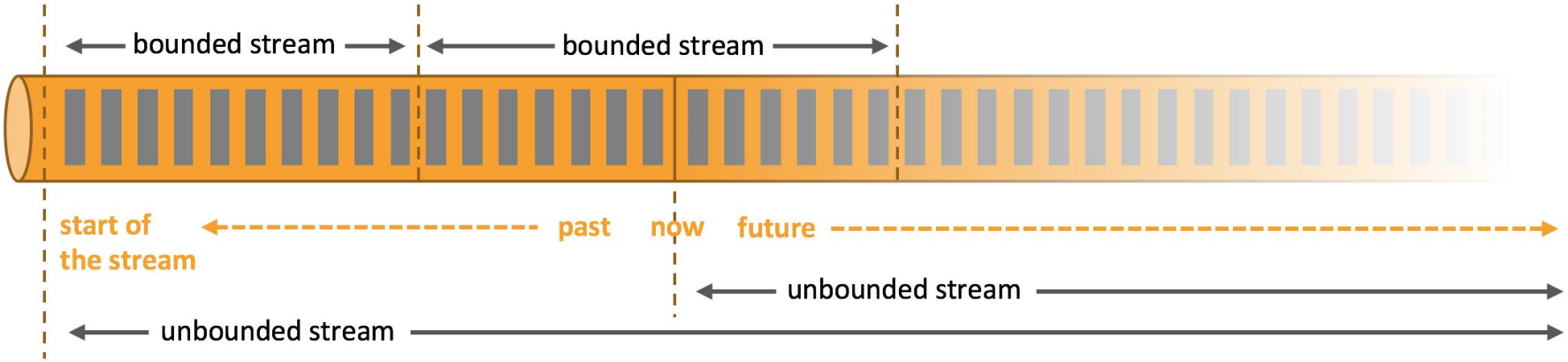


Image Credit: Tyler Akidau,
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101

Solution to Very Unbounded Dataset - Split It Into Chunks!

Window the data somehow to make it smaller!

Windowing is simply the notion of taking a data source (either unbounded or bounded), and chopping it up along temporal boundaries (or by some other aspect of the data) into finite chunks for processing.

Windows can be *time driven* (example: every 30 seconds) or *data driven* (example: every 100 elements). One typically distinguishes different types of windows, such as *tumbling windows* (no overlap), *sliding windows* (with overlap), and *session windows* (punctuated by a gap of inactivity). For further info on joins and types and windowing, see the concurrent talk “Zen and the Art of Streaming Joins” by Nick Dearden

Various Types of Windows

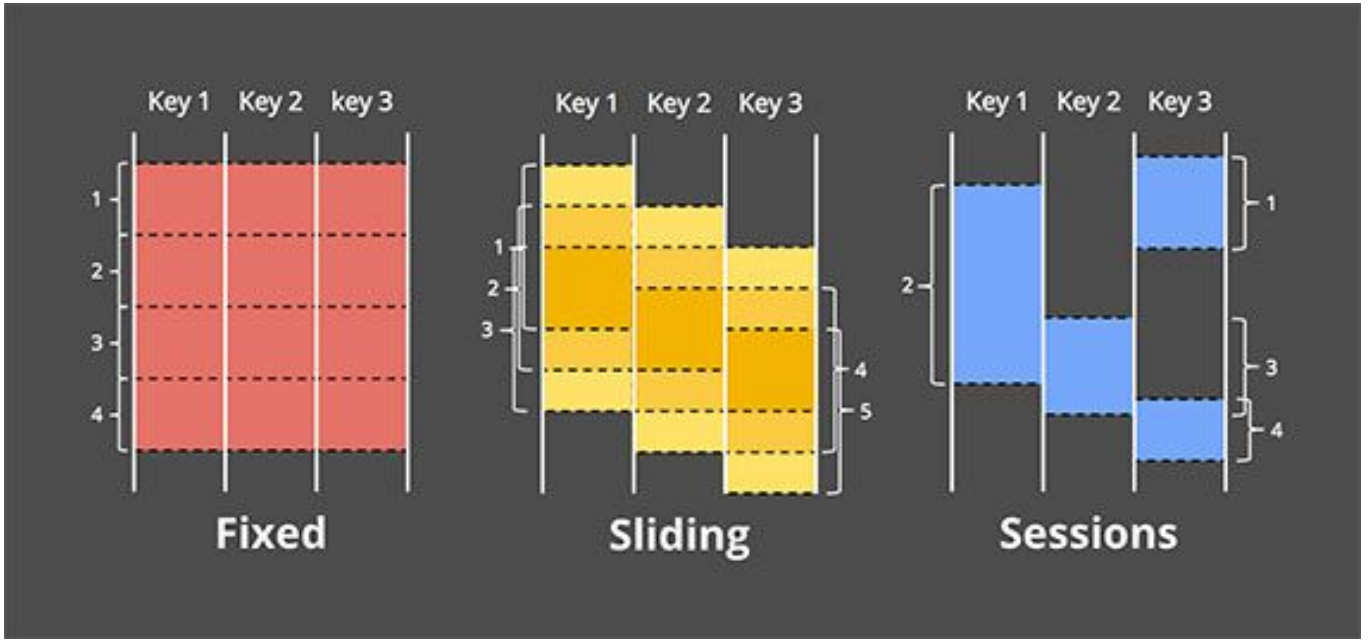


Image Credit: Tyler Akidau,
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101

Traditional Batch Processing For Counting By Hour

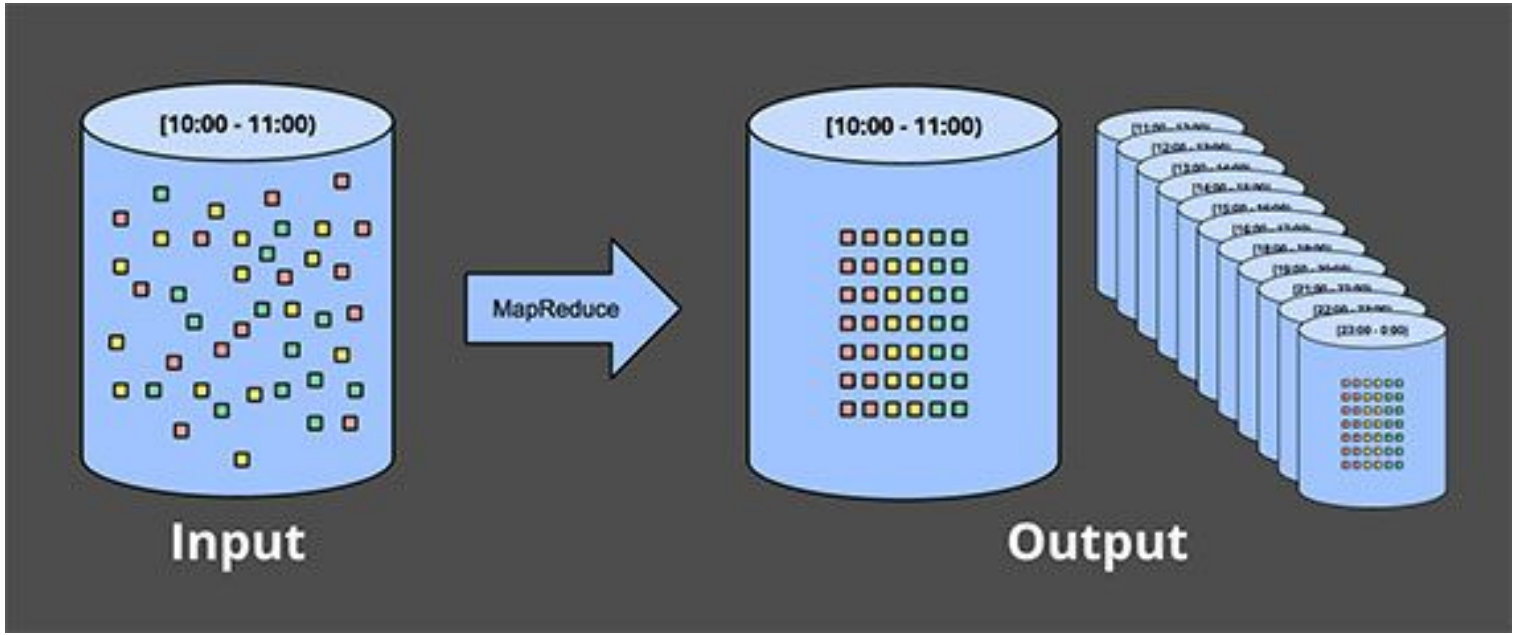


Image Credit: Tyler Akidau,
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101

What's One Problem in the Batch Based Counting Problem?

We have to wait at least an hour for the data to arrive!

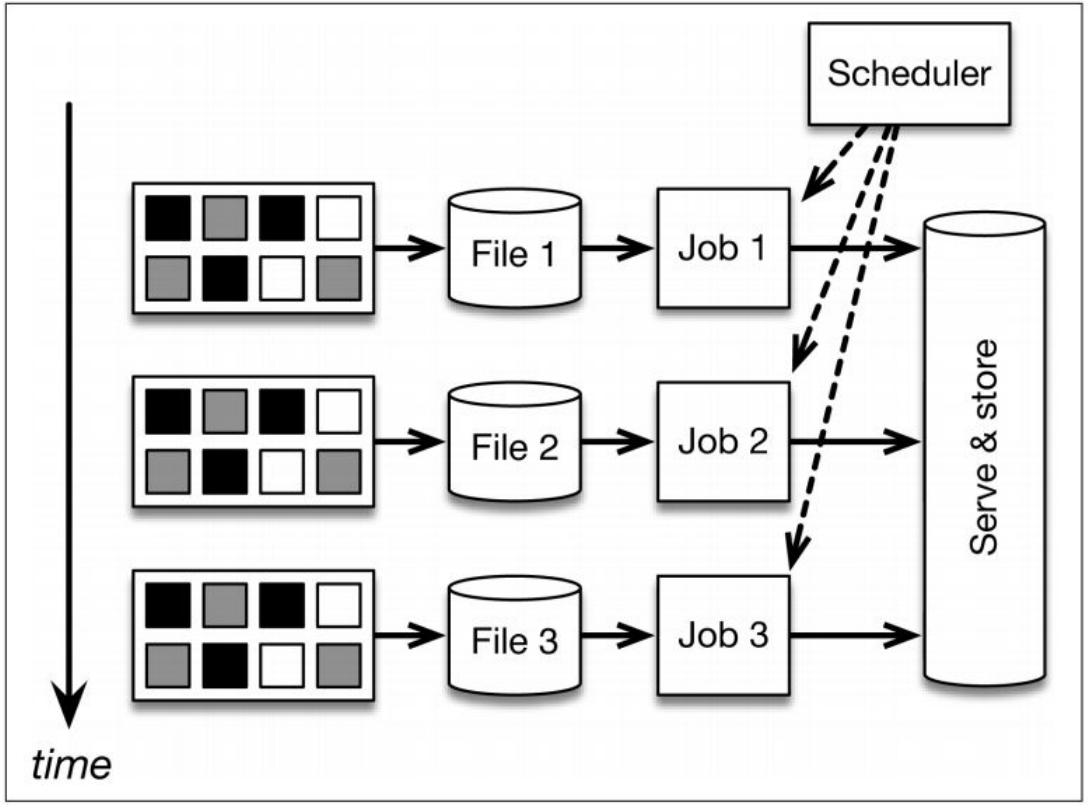
- **And it won't all be there by then!**

This can lead to results that take hours to days to calculate! 

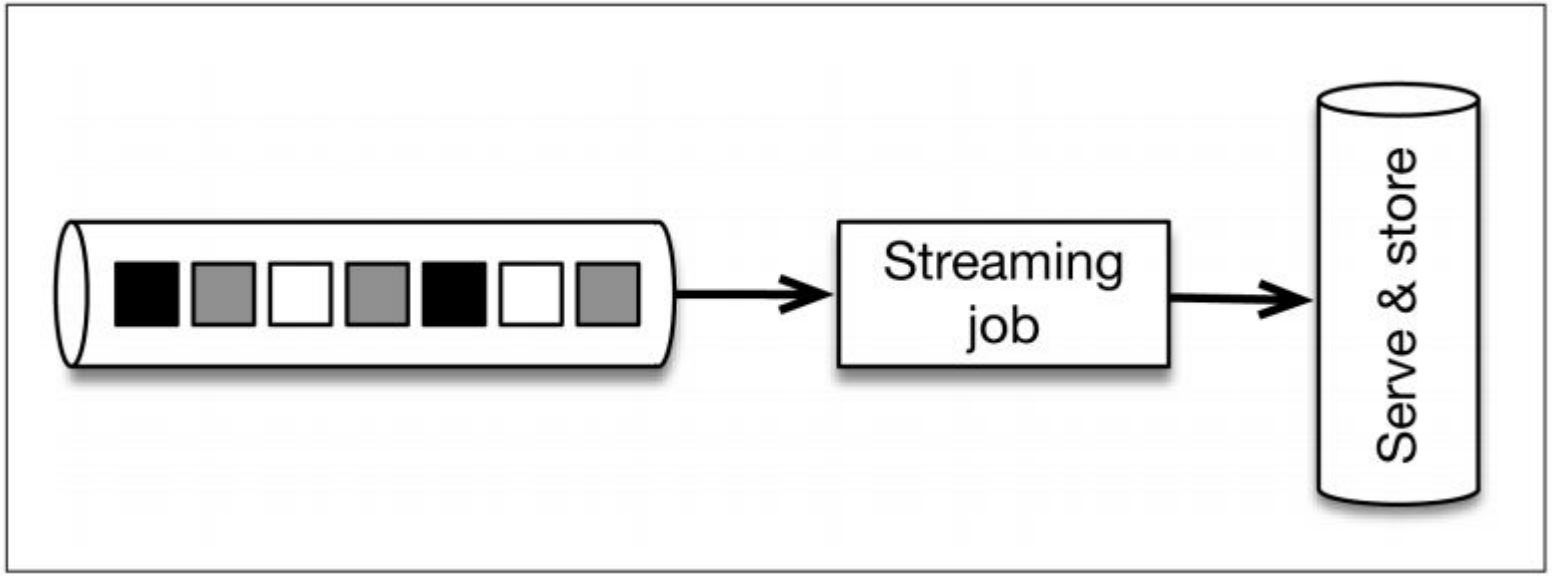
What's One Problem in the Batch Based Counting Problem?

This could be achieved in an order of magnitude less time in an online setting. Data in motion should stay in motion as much as possible.

Traditional (Hourly) Batch Processing Over Unbounded Data



Modeling This Process Streaming First



Why Event-Driven?

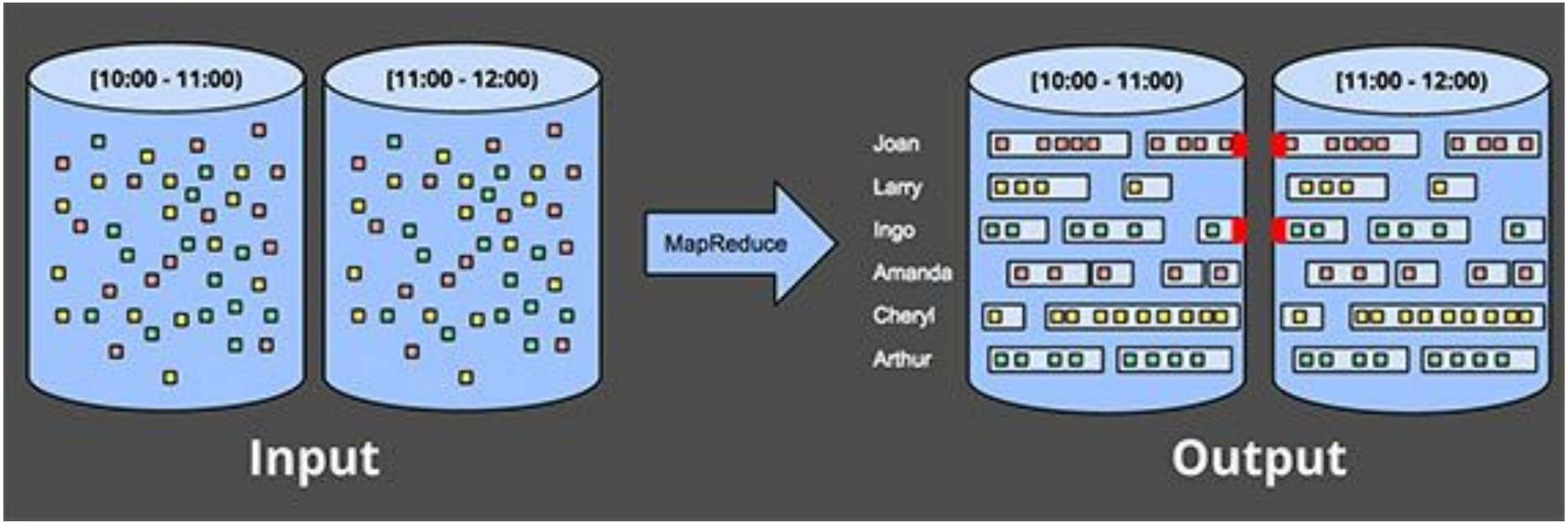
- Requirements for lower latency and/or speculative results (e.g. partial results before all events for a given window have arrived).
  - Know about and react to system changes as they occur, not well after the fact.

Exploring The Overall Problem Space Further

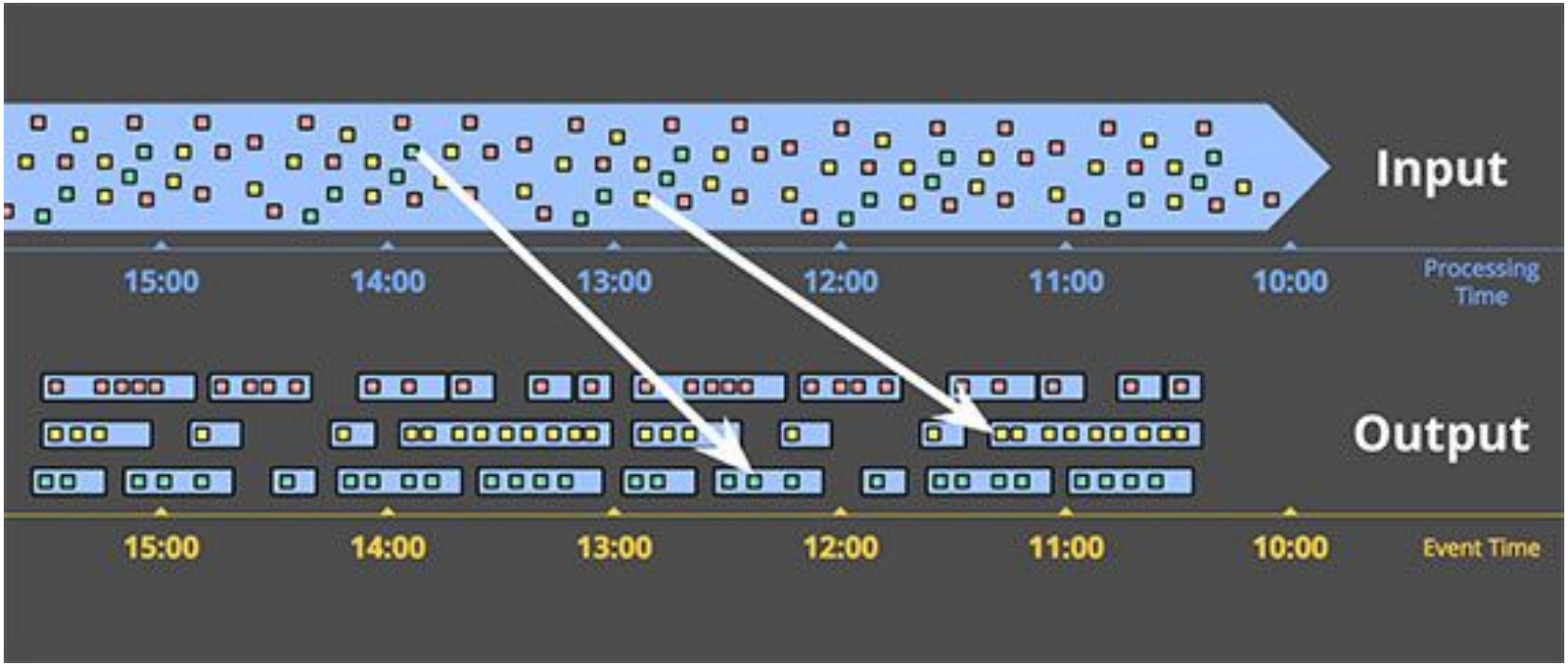
Problem: Extracting Useful Information From An Unbounded Stream of Data, which is almost always out of order, late, and can often be pretty messy.

Problem: Storing and Processing Large Historical Amounts of Data Without Paying Infinite Amounts of $$$ for Excessive and Often Times Unnecessary Intermediate Caches and Other Data Stores etc.

Traditional Batch Processing For Generating User Sessions - Shows How Hard and Messy Window Edges are for Batch Systems.

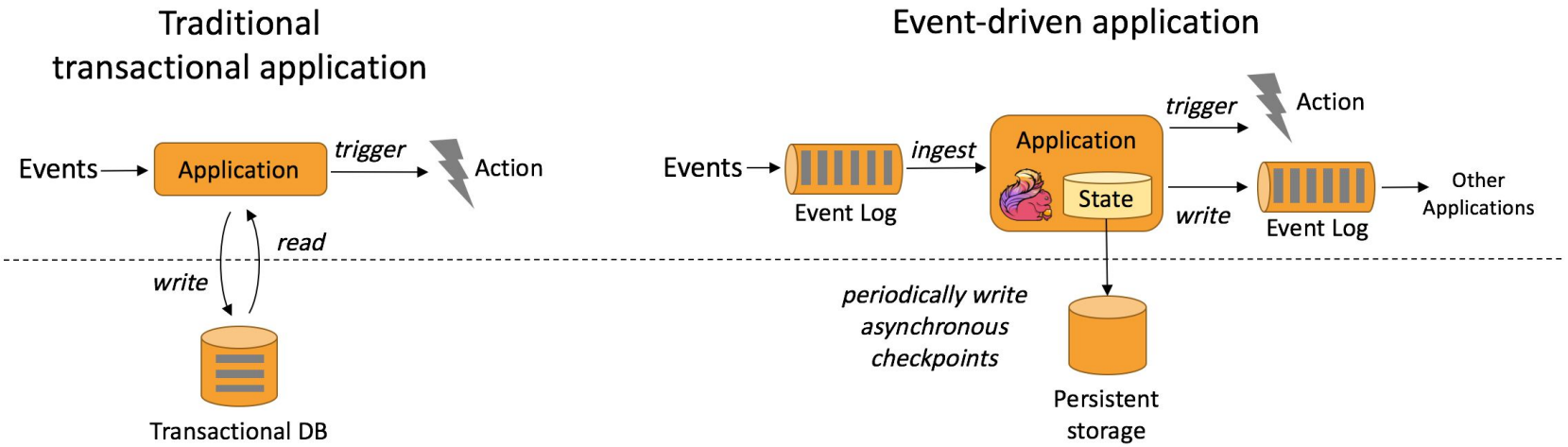


Reordering and Building Online User Sessions



What Does Any of This Have to do with Event Driven Applications?

Traditional Applications vs Event Driven Applications



What Happens in the Microservices Application when one HTTP request results into several HTTP requests to other various microservices, in the worst case?

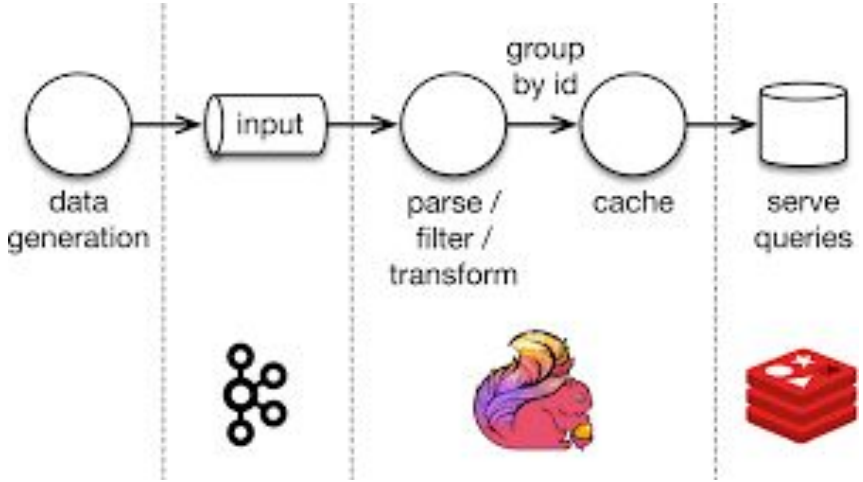
- Some services may succeed and others won't.
  - Even timeouts can cause false successes.
- Although the client might see the expected server side error on their end, the results may get partially updated to various state stores.
  - Data inconsistency can result
- Adding more and more HTTP requests, even "fire and forget" still adds latency to every request.

Why Event-Driven?

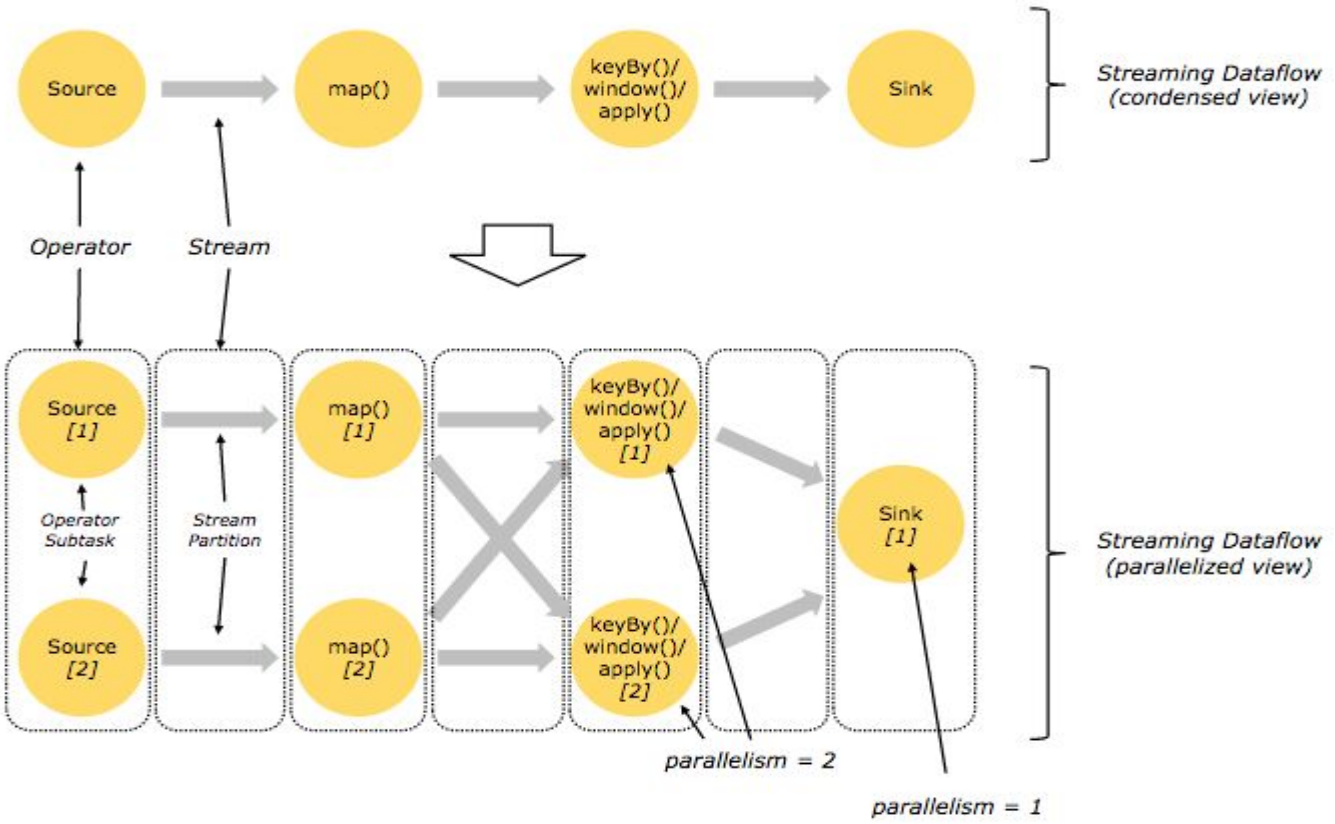
Ensure All Stakeholders Know What Is Covered By The System

- You may or may not be very surprised at how often you don’t know what you don’t know.

Streaming Event-Driven Approach: Simplified View for Microservice Consumption

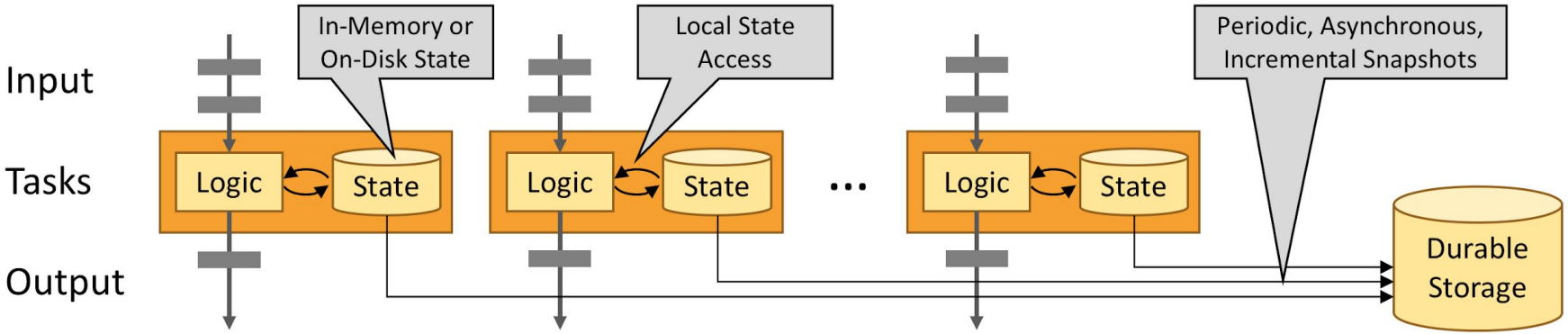


Distributed Architecture - Parallel View

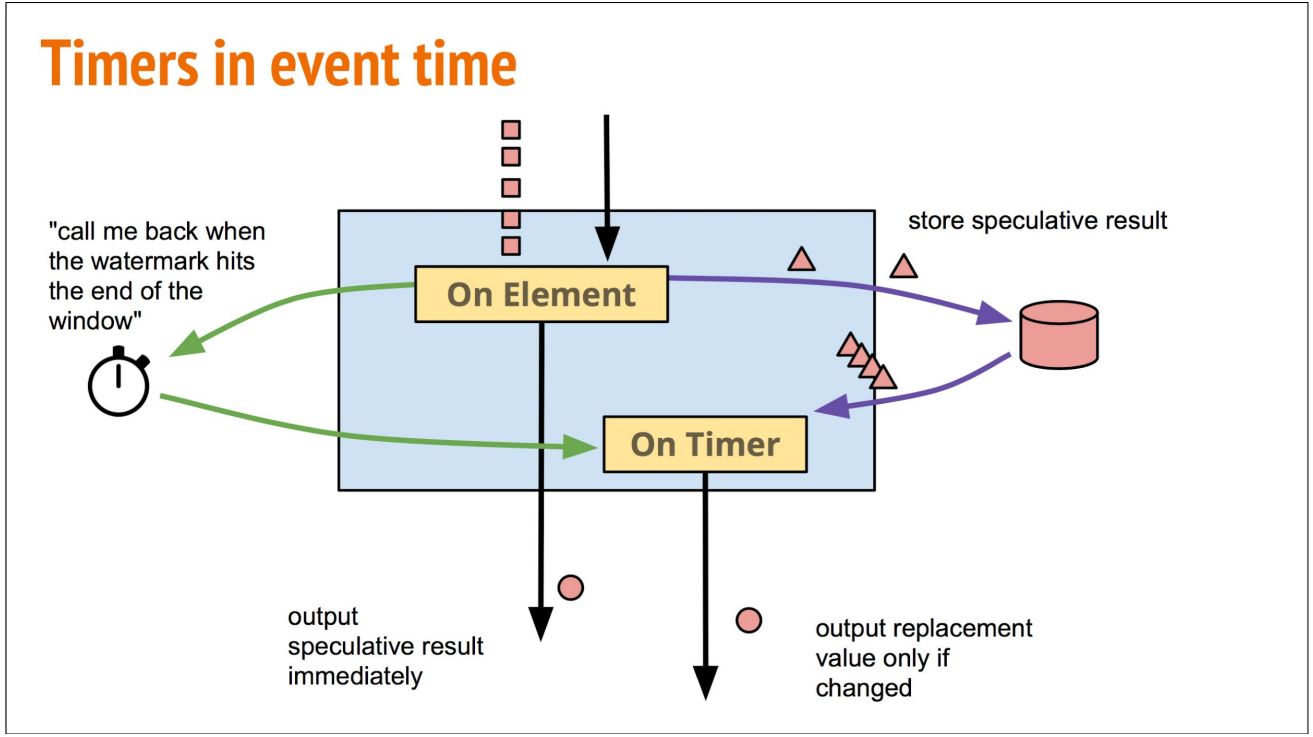


Source: https://flink.apache.org/introduction.html

Data Locality For Fast Local Processing of Distributed Datasets



Efficient per Key State wrt Stream Time in Flink

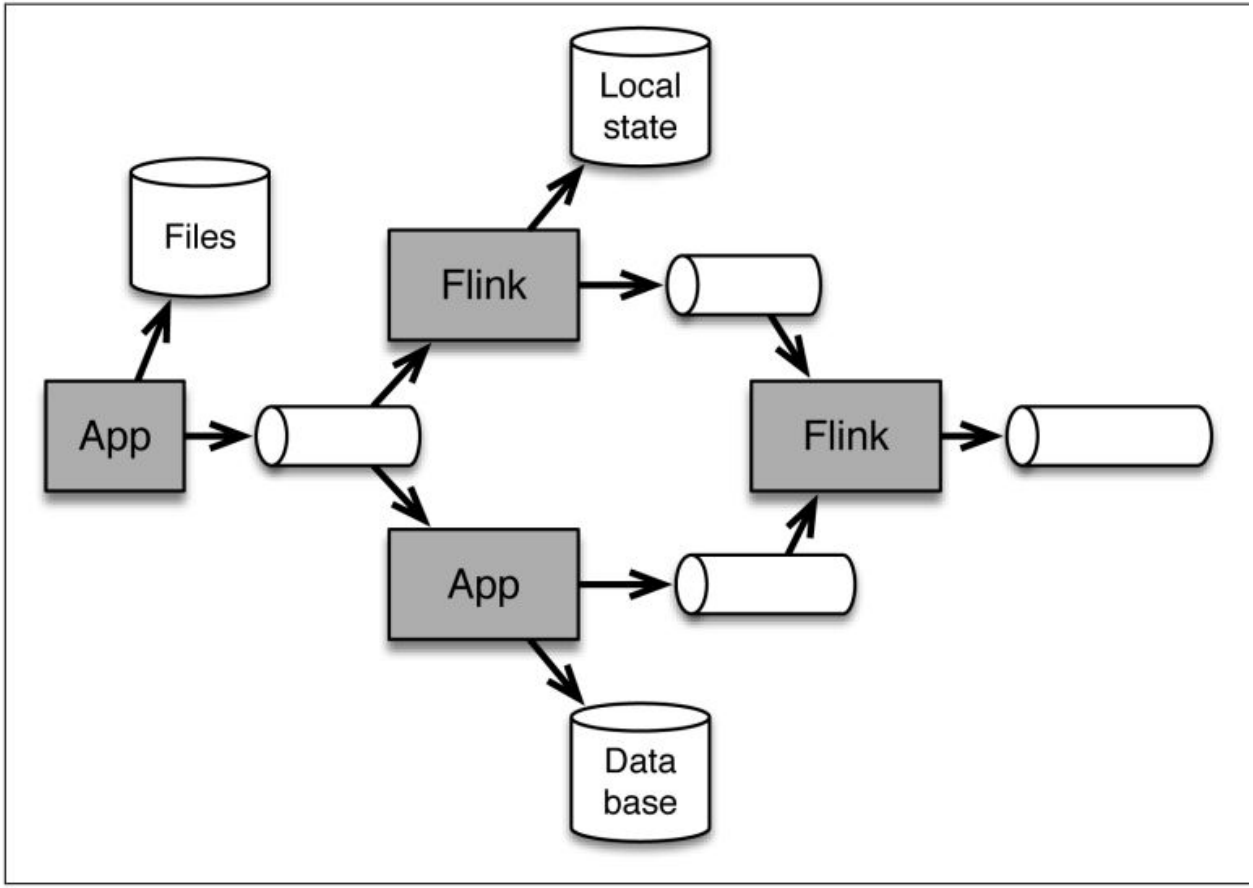


Can use KStream#Process or Punctuate APIs as well as KTable#Supress methods in Kafka Streams DSL to achieve some similar semantics for most use cases.

Ideal Streaming First Architecture - Full Data Flow

Let the message bus, Kafka, be the shared source of truth!

Localized views can serve the needs of different microservices



Now for a higher abstraction

An event stream can be considered as the changelog of a database table.

Streams and Tables Are Duals

Deriving each user's most recent location from a geolocation event stream

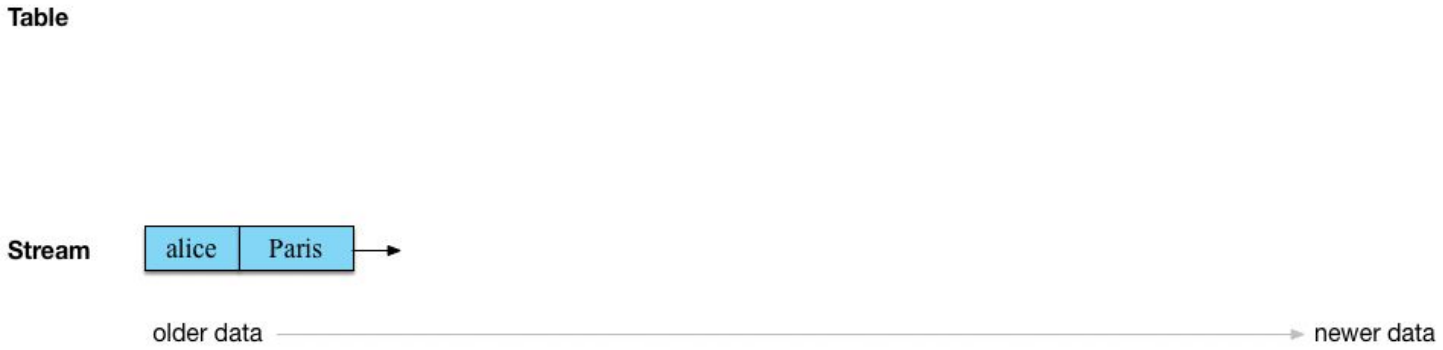


Image Credit: Michael Noll, Confluent:
https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/

Streams and Tables Are Duals

Deriving per user check in counts from the same stream.

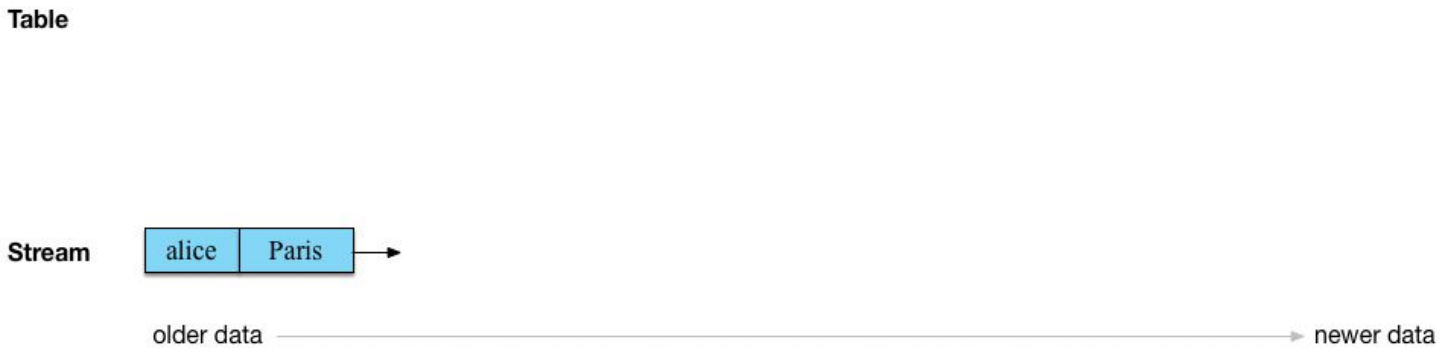
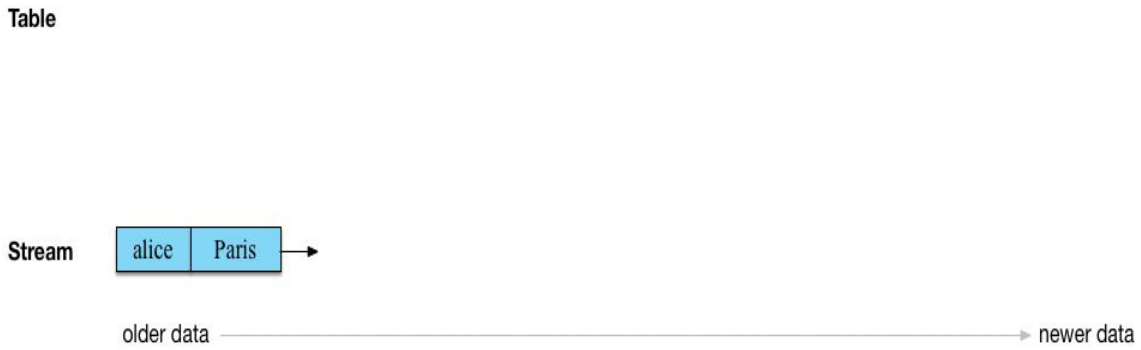


Diagram Credit: Michael Noll, Confluent
https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/

Side Note - This is all just map only SQL! (KSQL here to be exact)

```
CREATE TABLE user_location_counts_table AS
SELECT user, count(*)
FROM user_location_stream
WINDOW TUMBLING (Size 1 hour)
GROUP BY user
```



Section II: Event Driven Design Patterns and Pitfalls... Finally

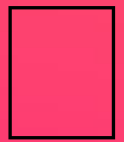
👍 Design Patterns 👍

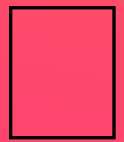
🚨 Pitfalls - Bad Idea 🚨

👍 Most Important Event Driven Pattern for Organizational Decoupling (In My Opinion) 👍

Event Storming

Wait... Event Storming?
What Did He Say?!





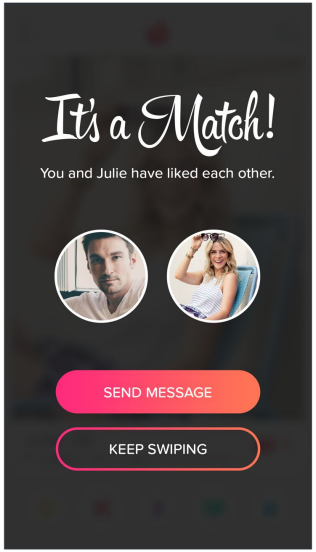
Event Storming or Domain Driven Modeling

TLDR: Flesh out all events and interactions in the domain of the project and the application, as a way to model the entire overall architecture.

Event Storming - Why?

- Treat the Events as First Class Citizens in the System
- Ensure All Stakeholders Know What Is Covered By The System
  - You'd be very surprised at how often you don't know what you don't know.

Event Storming Example - Swiping



Strictly One Card at a Time

Swipe Left to Nope

Swipe Right to Like

If both swipers like, It's a Match!

Let's Model A Few of The Basic Event(s) Involved in the Classic Swiping Experience!

```
case class Swipe(userId: String /* Swiper */,
                 otherId: String /* Swipee */ ,
                 like: Boolean,
                 ts: Long /* Event Time */
                 st: Long, /* IngestionTime*/
                  ...
)
```

What Other Events / Facts Took Place Here?

```
case class Match(user1: String, user2: String)
```

What Other Events / Facts Took Place Here?

/ What is this missing from the overall match event? */*

```
case class Match(u1: String, u2: String)
```

What could we have captured when we had the chance?

```
/* Swiper / Swipee Info ! */
case class Match(u1: String, u2: String, ...)

/* Provides more information as */
case class Match(swiper: String ,
                 swipee: String)
```

Recall the three basic event types from the CQRS pattern:

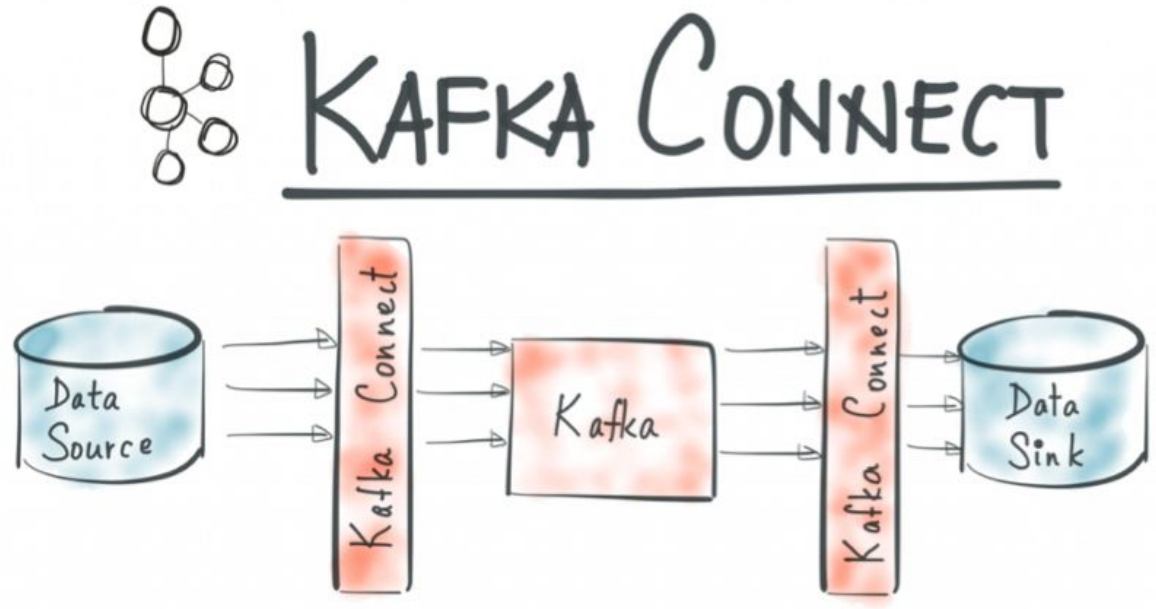
- Plain Old Events - Simple statements of facts that took point in time, such as a swipe event or a new match event. These are the foundation of an event-driven platform.
- Commands - Calls to action
- Queries - Asking a question, e.g. of an aggregate or over a specific view.

Event Driven Pattern 👍

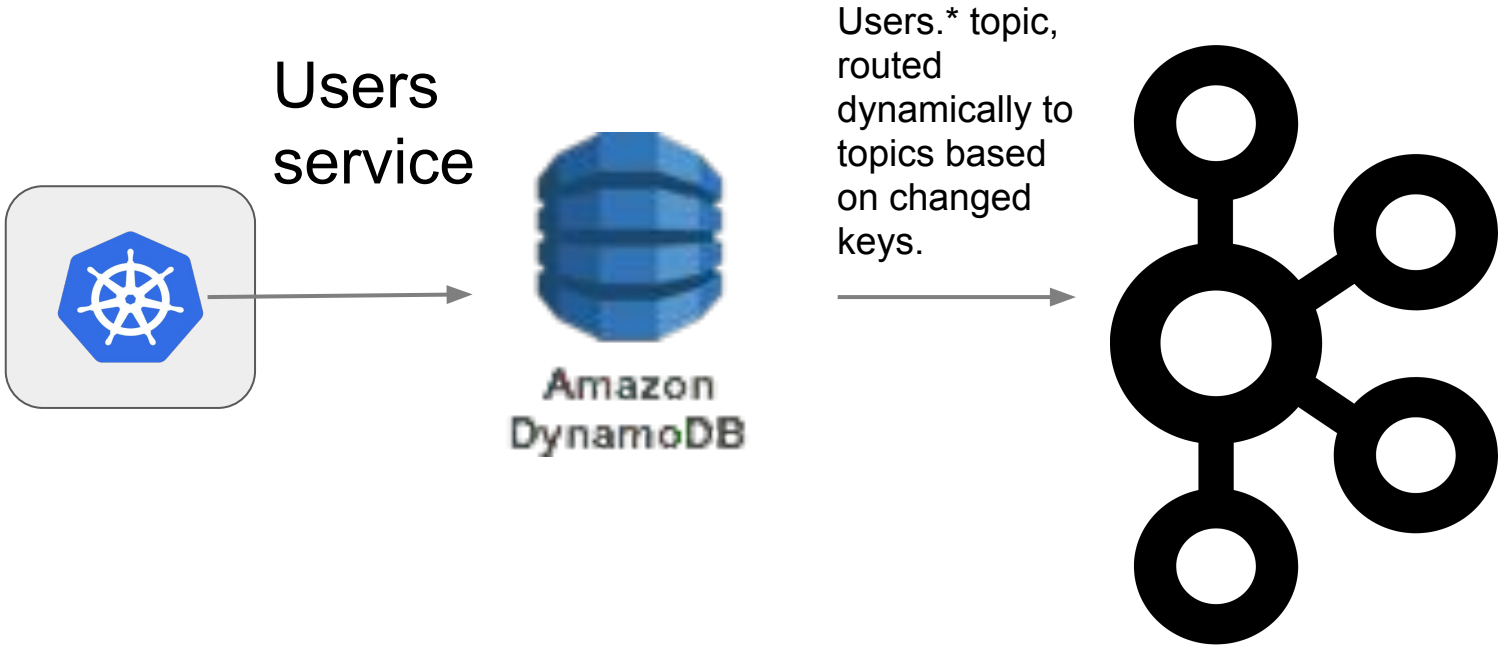
Leveraging ***Change Data Capture*** whenever Available.

Especially Useful for Migrating to Kafka as a Shared Source of Truth

Design Pattern: CDC in Action with Kafka Connect



Design Pattern: CDC in Action via Existing Sources of Truth, namely DynamoDB streams



Benefits of DynamoDB Streams or other CDC Systems:

- Can view new and old versions of the record
- Easy means for notifications of delete events, to propagate deletions throughout the system for data hygiene and compliance
- Can parse updates on keys only
- With true change data capture, backfills of all time data isn’t as pertinent needed.

Event-Driven Pitfall 🚨🚨🚨

Not considering how long older data is retained locally before its expired

Migrational Pitfall: Not Considering Data Retention 🚨

NOTE: It’s totally possible to store data forever in Kafka.

Migrational Pitfall: Not Considering Data Retention 🚨

With partitioned data by key, the application doesn’t always know when that key and its corresponding state will be expired.

- Keyed session identifiers
- Any aggregation (technically the window is part of the grouping)

Migrational Pitfall: Not Considering Data Retention

How to handle this:

- Manually in the code or via state retention.
- Configured in the state store builder for Kafka backed state stores in the Kafka Streams DSL.
- In Flink:

  StreamingQueryConfig#withIdleStateRetentionTime

Event-Driven Migrational Pitfall

🚨🚨🚨

Not considering the truthiness of data that’s onboarded to Kafka in your early phases

Migrational Pitfall: Truthiness of your CDC source

Assume this is a **Derived View** and **not a Source of Truth**.... Can + WIII Likely diverge from whichever siloed data is considered the “Ground Truth”

Migrational Pitfall: Truthiness of your CDC source

Proper advice in the last scenario:

- Don’t onboard this data into kafka if you’re using it as a shared source of truth unless it REALLY is your source of truth 

Migrational Pitfall: Truthiness of your CDC source

- 👎 Practical advice in the last scenario:
- Onboard this if you need to, depending on your own application tolerance for possibly incorrect data. 👎
- Label sketchy data as such:
  - E.g. topic prefix namespacing: user-unreliable-* topics vs user-* topics

Pitfall: Not considering the source of truth of your events!

- Client side events are inherently less reliable than server side events.
  - Bugs introduced to app versions are harder to fix than server side bugs.
- Client side events are less reliable in general, as they may come much later than server side events.

Pitfall: Not considering the source of truth of your events!

- What can we do about this?
  - How much truthiness does your application realistically require?
  - Can we add a corresponding server side event that would be more reliable and easier to evolve over time (e.g. new releases for services which emit events happen every day)
  - Extracting events from some datasource that is closer to the source of truth than other possible options.

Event Driven Pattern 👍

Using ***Broadcasted State*** to share data sets across all consuming members of streaming event-driven services.

Event Driven Pattern 👍

Performing previously fanned out microservice aggregational ***Joins*** on data once it’s in kafka.

Design Pattern: Broadcasted State

- Submits a single event stream to all consuming members in a topology.
  - Aka a *GlobalKTable* in the Kafka Streams DSL.
  - *Single writer* and *many consumers* of a changelog topic to build a globally replicated view.

Design Pattern: Broadcasted State

- **Usefulness:**
  - ***Small-to-Medium Sized Tables Only***, as the resulting broadcasted table from the changelog stream is **replicated to all tasks.**
  - ***Dynamic filter configurations***, to allow program managers etc to update topology rules.

Design Pattern: Broadcasted State

- Drawbacks:
  - Is not really distributed (that's the point)
  - In Kafka Streams DSL, no easy way to ensure that only a single process writes to the changelog for the broadcasted table.

Event Driven Pattern 👍

Shared Logging Framework

Shared Logging Framework.

Adding large amounts of event logging is sometimes initially viewed *incorrectly* as an extra burden to microservice developers.

However, no microservice developer is realistically opposed to having to call a `log.info()` function.

Shared Logging Framework.

Allow developers to focus on the business logic of their existing microservices without thinking of the event stream they're generating and any associated challenges of "Pipelines" and "Data flows" being added elsewhere.

- Can be as simple as `.log` to kafka function for a JSON or Avro Payload.

Event Driven Pattern 👍

Side Outputting Data

- ***Late data***
- ***Data deemed “invalid”***

Side Outputting Data

Even If You Think You Don’t Need It!

- Late data may need to be replayed if there’s a long enough outage or large unforeseen watermark skew.

Side Outputting Data

Late data may need to be replayed if there's a long enough outage or large unforeseen watermark skew.

Side Outputting Data

How To

- Can be as simple as DLQ topic,
- Using a side-output in Apache Flink,
- Expressing the side output as a portion of your topology (e.g. for data deemed “invalid”) so that it benefits from the same semantics as the entire topology.

Detour - Allowing Non-Programmers to Write and Deploy Event-Driven Streaming Analytics for Intelligence Gathering

Generic Streaming Aggregation Abstraction

- Generic streaming services/abstractions for real-time analytics
- Expressed by relational query
- “Correct by default” when interacting with sources/sinks/external system
- Monitoring + cost billing attribution per team / pod in the organization
- Aim to support custom logic while fitting into the incremental model
- Can be chained together to create an ad-hoc data flow

Motivation for the Generic Streaming Aggregation Abstraction

- Two Types of Stream Processing (from Gartner)
  - Stream Data Integration
    - Primarily cover streaming ETL
    - Integration of data source and data sinks
    - Filter and transform data (Enrich data)
    - Route data
  - Stream Analytics
    - Calculating aggregates
    - Detecting patterns to generate higher-level, more relevant summary information

Motivating Case study: Phoenix AB Test Experiment Assignment

Want to know, in near real-time, how many users have been assigned to various A/B testing experiments, including the relevant subgroups etc.

Motivating Case study: Phoenix AB Test Experiment Assignment

Spoiler Alert: Get Ready for A Variant of Word Count!

Case study: AB Test Experiment Assignment - Possible Events

```
case class ExperimentAssignment(
                        uid: String,
                        ts: Long,
                        experimentName: String,
                        experimentGroup: String,
                        experimentBucket: String)

case class ExperimentCount(
                       experimentGroup: String,
                       experimentName: String,
                       experimentBucket: String,
                       countUsers: Long = 0L)
```

Case study: Phoenix AB Test Experiment Assignment - Basic (Flink/Apache Calcite) SQL to Solve this Problem!

```
val sql: String =
 """
   | SELECT experimentGroup,
   |       , experimentName
   |       , experimentBucket
   |       , COUNT(uid) as countUsers
   | FROM
   |     experiment_assignment
   | GROUP BY
   |     1, 2, 3, TUMBLE(ts, interval '1' HOUR)
 """
```

Case study: Phoenix AB Test Experiment Assignment

```
pipeline:
  - operation: parse
    parse:
      inputCol: value
      parameters:
        - jsonPath: st
          outputType: milliToTimestamp
          outputCol: st
        - jsonPath: experimentGroup
          outputType: string
          outputCol: experimentGroup
        - jsonPath: experimentName
          outputType: string
          outputCol: experimentName
        - jsonPath: experimentBucket
          outputType: string
          outputCol: experimentBucket
  - operation: window_aggregate
    window:
      timestampColumn: st
      duration: 5.minutes
      slideDuration: 5.minutes
      watermark: 1.minute
    groupByKeys: [experimentGroup,experimentName,experimentBucket]
    aggregations:
      - reducer: count
        outputColumn: count
```

Kafka source:

```
source:
  name: kafka-data-prod-spark
  parameters:
    topic: quickfire-Experiment.Assignment
    failOnDataLoss: false
```

Case study: Phoenix AB Test Experiment Assignment Partial Configuration File Generated From the UI or Typed Out by Lazier Developers (like me)

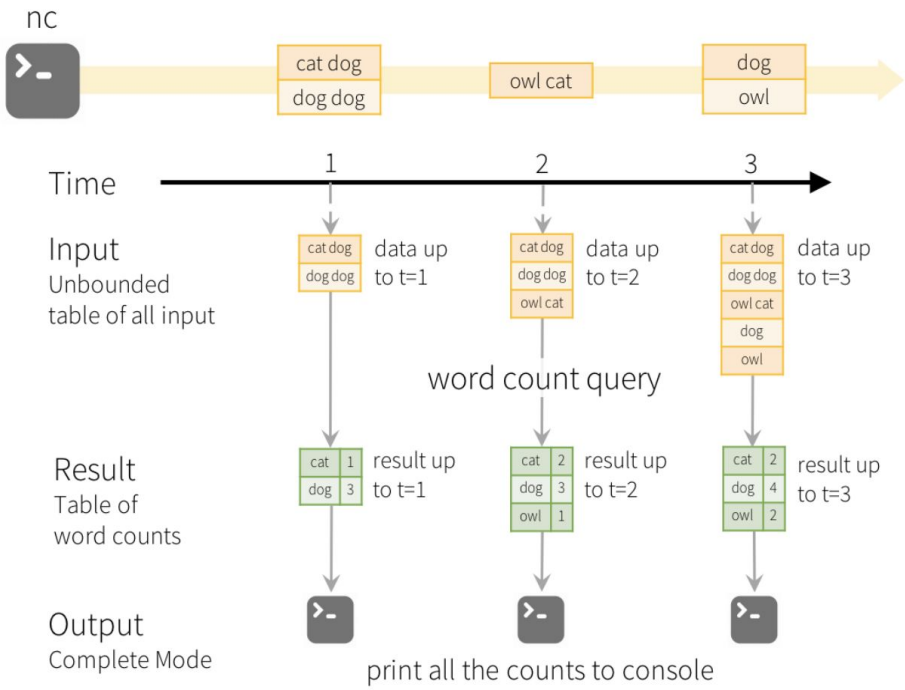
```
pipeline:
  - operation: parse
    parse:
      inputCol: value
      parameters:
        - jsonPath: st
          outputType: milliToTimestamp
          outputCol: st
        - jsonPath: experimentGroup
          outputType: string
          outputCol: experimentGroup
        - jsonPath: experimentName
          outputType: string
          outputCol: experimentName
        - jsonPath: experimentBucket
          outputType: string
          outputCol: experimentBucket
  - operation: window_aggregate
    window:
      timestampColumn: st
      duration: 5.minutes
      slideDuration: 5.minutes
      watermark: 1.minute
    groupByKeys: [experimentGroup,experimentName,experimentBucket]
    aggregations:
      - reducer: count
        outputColumn: count
```

Kafka source:

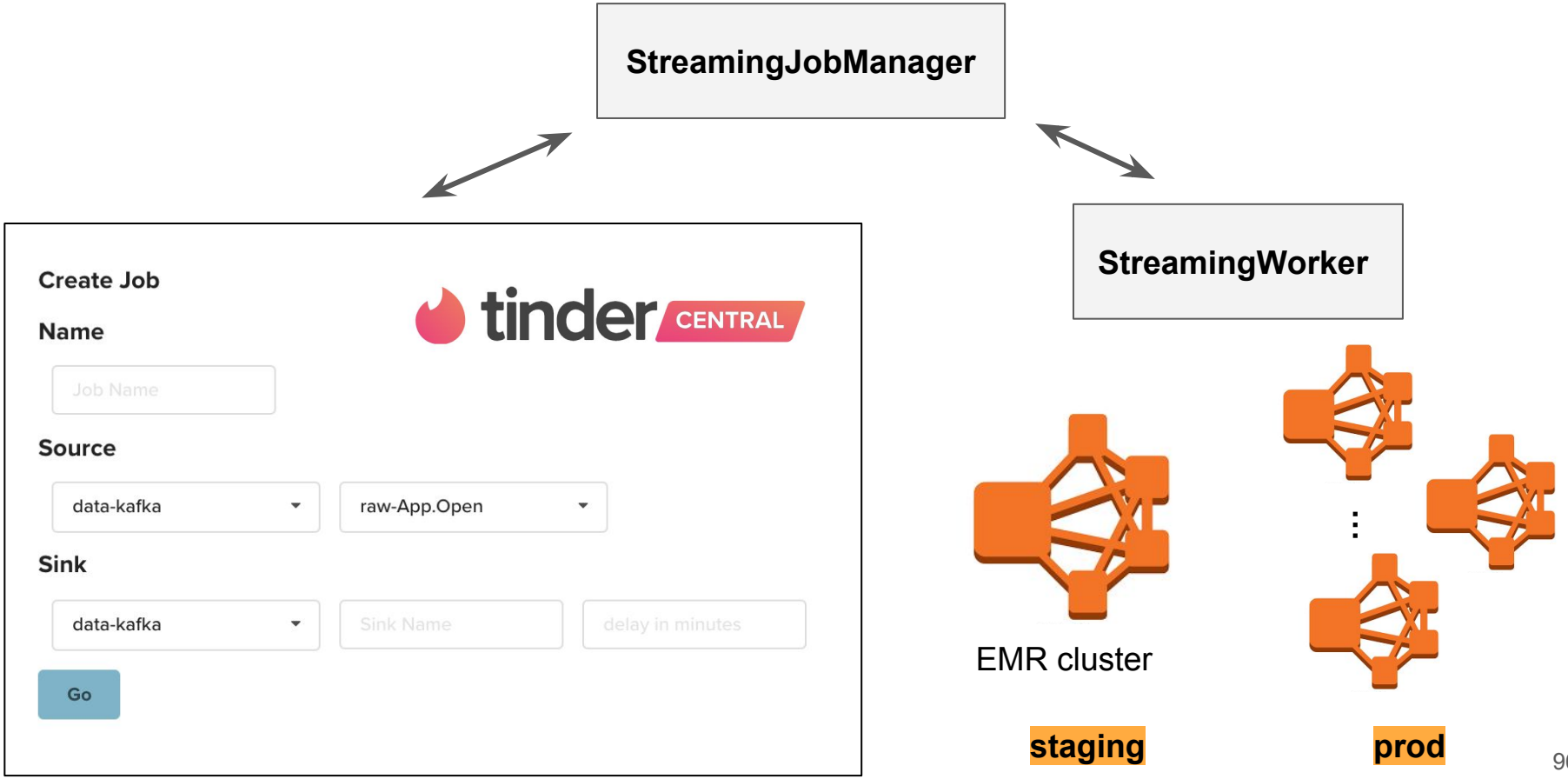
```
source:
  name: kafka-data-prod-spark
  parameters:
    topic: quickfire-Experiment.Assignment
    failOnDataLoss: false
```

Streaming Feature Generation with Apache Spark Structured Streaming

- Purely declarative API
- Unbounded table + Incremental query
- Spark SQL execution engine
- High throughput, low latency and fault-tolerant
- Unified Batch/Streaming API
- Streaming ML

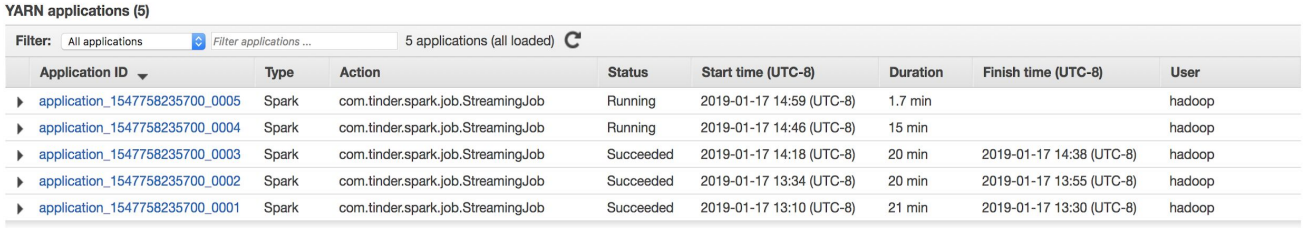


Streaming Job Manager



Streaming Job Manager

- Trigger a new job
- Terminate a job
- Get job status



Coming soon…

- Testing environment for job queries
- Sample job output
- Exception message if job fails

Streaming Job Manager

```
TableName: streaming-jobs-config
AttributeDefinitions:
  - AttributeName: jobId
    AttributeType: S
  - AttributeName: teamName
    AttributeType: S
  - AttributeName: jobName
    AttributeType: S
  - AttributeName: s3Key
    AttributeType: S
  - AttributeName: clusterId
    AttributeType: S
  - AttributeName: status
    AttributeType: S
  - AttributeName: exceptions
    AttributeType: S
KeySchema:
  - AttributeName: partitionKey (teamName_jobName)
    KeyType: HASH
  - AttributeName: version (timestamp when job is created)
    KeyType: RANGE
```

- Query job with

  `teamName + jobName`
- Get job info with

  `teamName + jobName + version`
- Terminate job with

  `teamName + jobName + version`
- Job `status`
  - Active
  - Terminated
  - FallingBehind

Motivating Case study: Streaming - Extra Neat Features

- Support for custom User Defined Aggregate Functions
- Many of these are derived from a UI from a set of drop down aggregations, etc.
- Devs find novels ways to chain these back into the application, typically via Kafka and then DynamoDB or ES

Many Thanks!

- Tao Tao, Kate Yortsos, Shan Jing, Eric Lane, Winston Lee, Kristin Collins Jackson, Mary Dittmar, Cooper Erlendsson, Yating Zhu, Chris Dower, Matthew Orlove, Aijia Liu, Anne Ying-An Lai, Wei Xu, Jinqiang Han, Michael Allman, Jessica Hickey, Jingwei Wu, Yibing Gu, and so many others!

Further References

- Streaming Systems Book - Tyler Akidau, http://streamingsystems.net/ O'Reilly and other DRM-free platforms
- The Data Dichotomy: Rethinking the Way We Treat Data and Services - Ben Stopford, https://www.confluent.io/blog/data-dichotomy-rethinking-the-way-we-treat-data-and-services/
- High Performance Spark - Holden Karau + Rachel Warren via O'Reilly and other DRM-free platforms
    - Not necessarily streaming focused, but a great read for thinking about data processing with spark at scale
    - https://www.oreilly.com/library/view/high-performance-spark/9781491943199/