Foundations of streaming SQL
or: how I learned to love stream & table theory

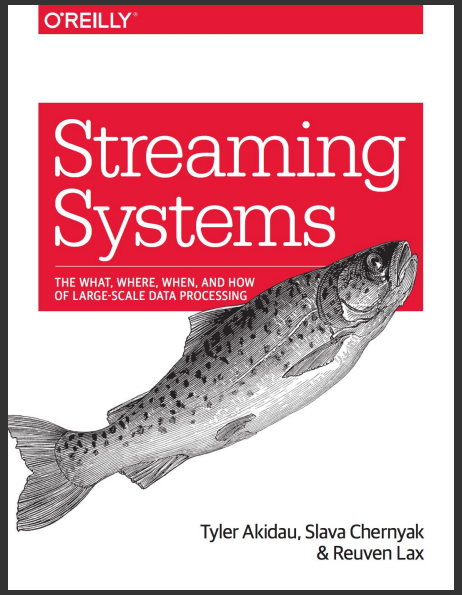
Slides: https://s.apache.org/streaming-sql-qcon-london

Tyler Akidau
Apache Beam PMC
Software Engineer at Google
@takidau

Covering ideas from across the Apache Beam, Apache Calcite, Apache Kafka, and Apache Flink communities, with thoughts and contributions from Julian Hyde, Fabian Hueske, Shaoxuan Wang, Kenn Knowles, Ben Chambers, Reuven Lax, Mingmin Xu, James Xu, Martin Kleppmann, Jay Kreps and many more, not to mention that whole database community thing...

QCon London 2018

Table of Contents



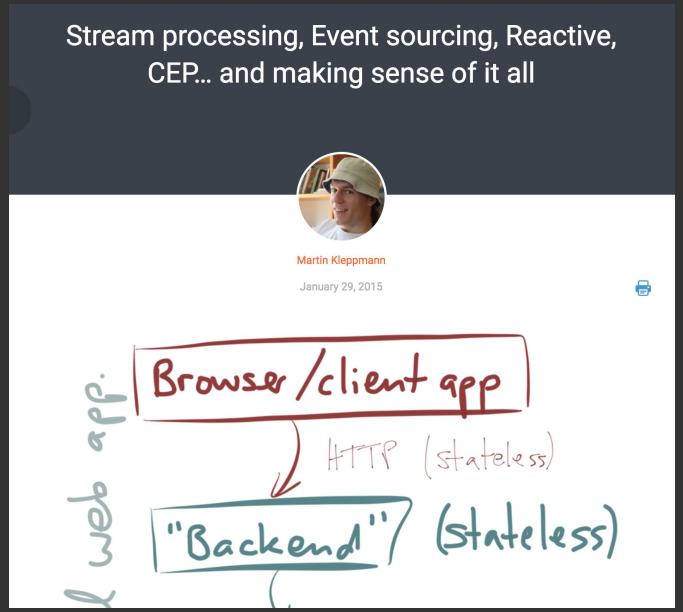
01 Stream & Table Theory

TFW you realize everything you do was invented by the database community decades ago...

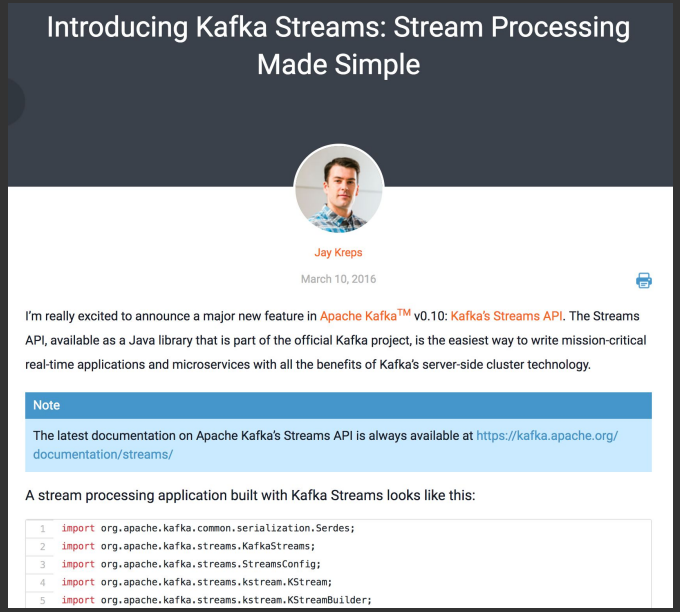
A Basics

B The Beam Model

Stream & table basics



https://www.confluent.io/blog/making-sense-of-stream-processing/



https://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple/

Special theory of stream & table relativity

streams → tables: The aggregation of a stream of updates over time yields a table.

tables → streams: The observation of changes to a table over time yields a stream.

Non-relativistic stream & table definitions

Tables are data ***at rest***.

Streams are data in ***motion***.

01 Stream & Table Theory

TFW you realize everything you do was invented by the database community decades ago...

A Basics

B The Beam Model

The Beam Model

What results are calculated?

Where in event time are results calculated?

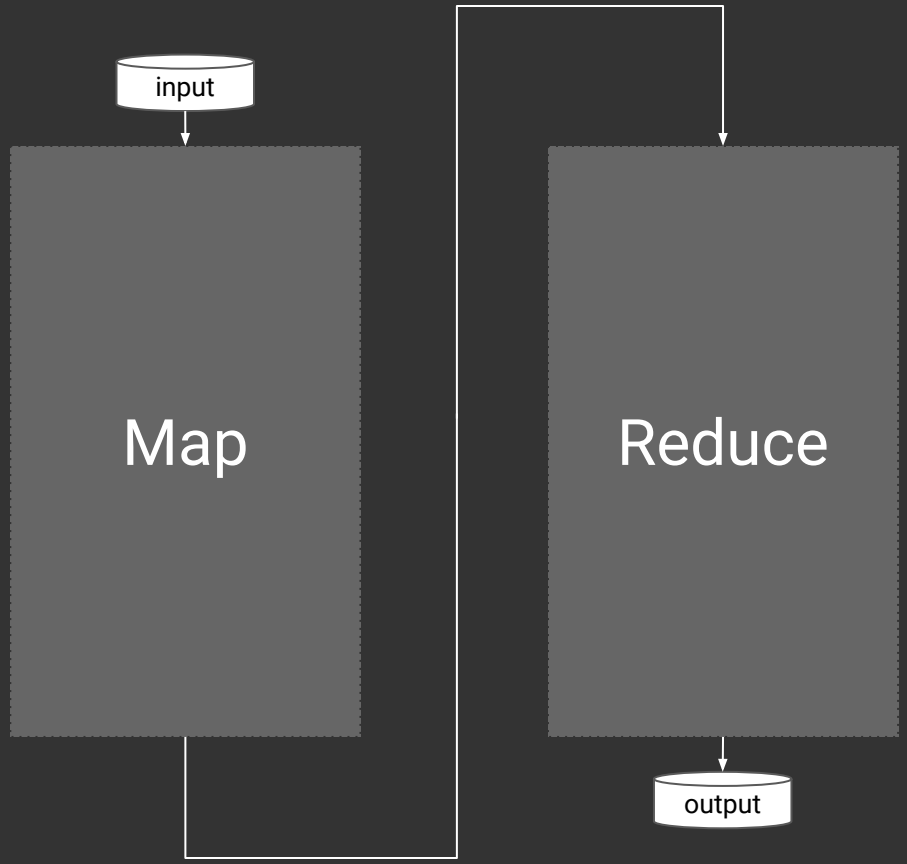
When in processing time are results materialized?

How do refinements of results relate?

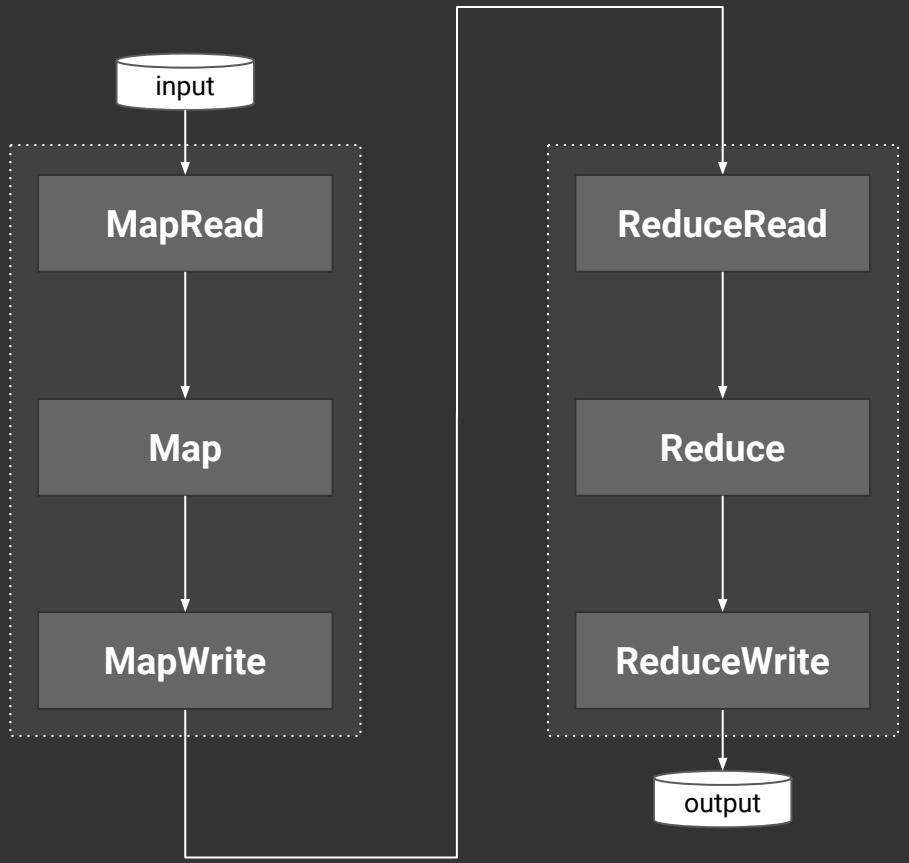
Reconciling streams & tables w/ the Beam Model

- How does batch processing fit into all of this?
- What is the relationship of streams to bounded and unbounded datasets?
- How do the four ***what***, ***where***, ***when***, ***how*** questions map onto a streams/tables world?

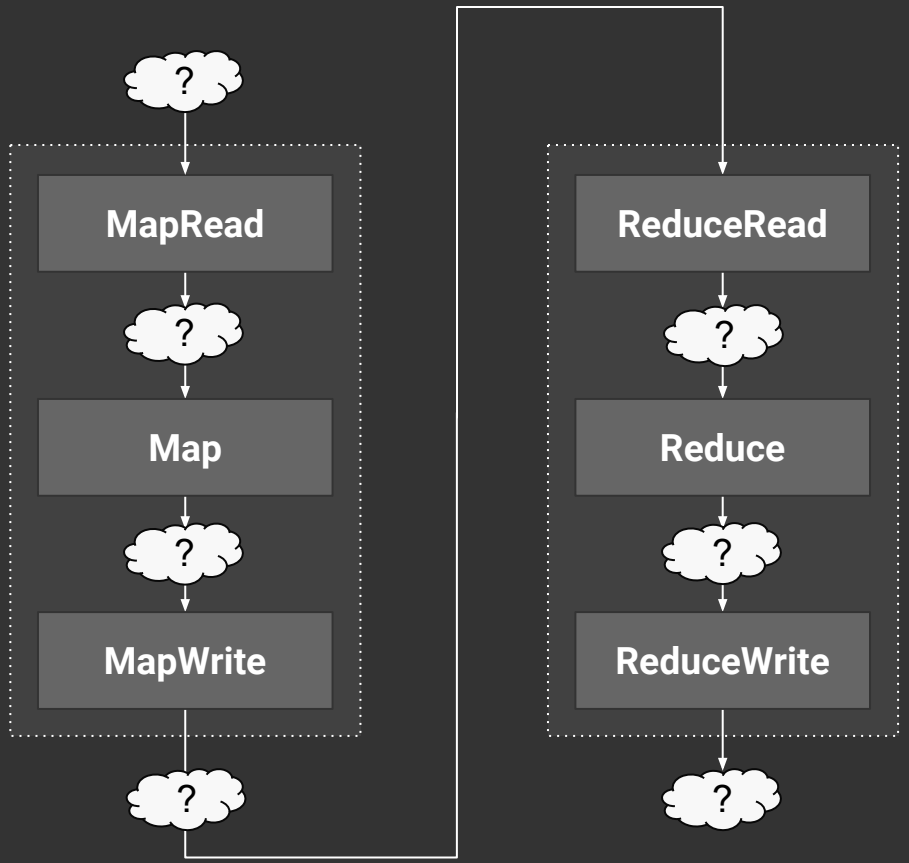
MapReduce



MapReduce


input
MapRead
Map
MapWrite
ReduceRead
Reduce
ReduceWrite
output

MapReduce



MapReduce

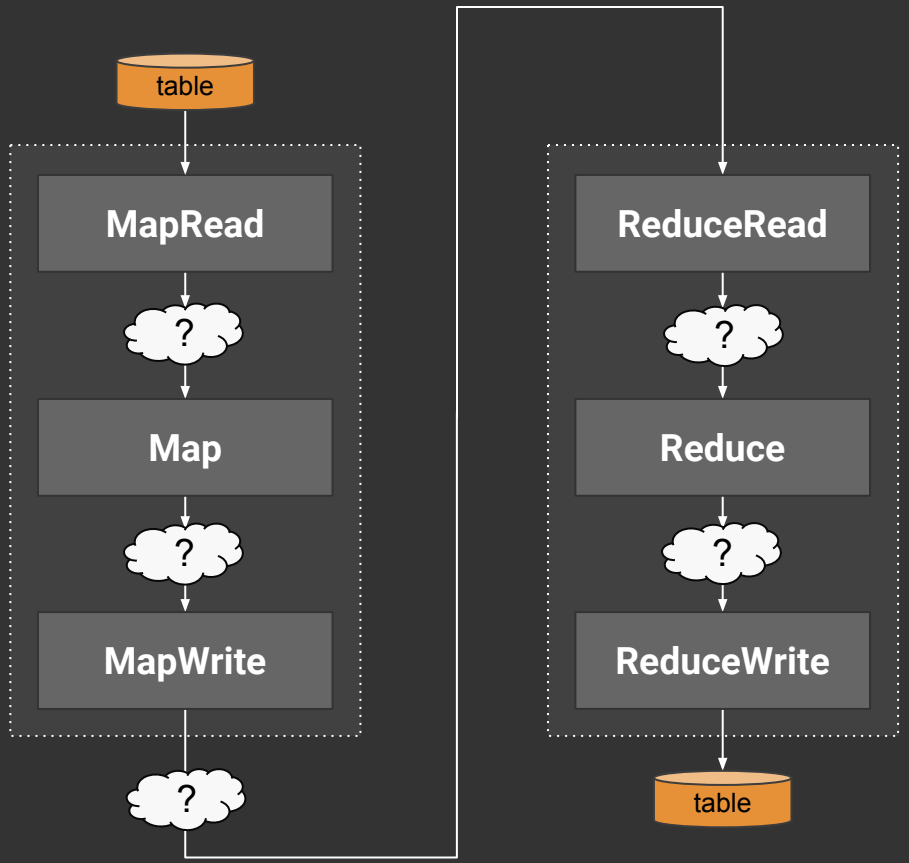
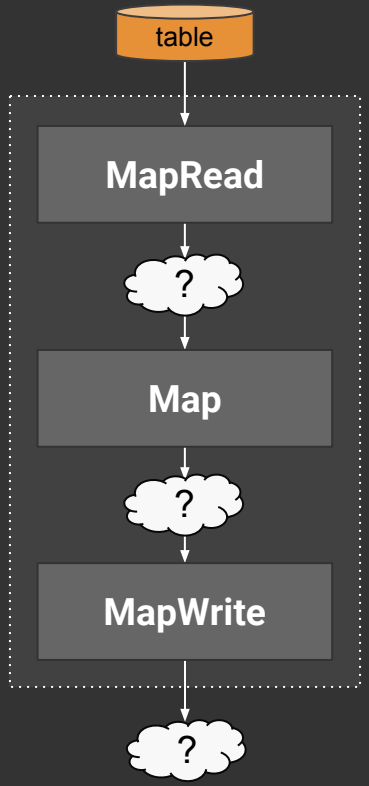


table
MapRead
?
Map
?
MapWrite
?
ReduceRead
?
Reduce
?
ReduceWrite
table

Map phase



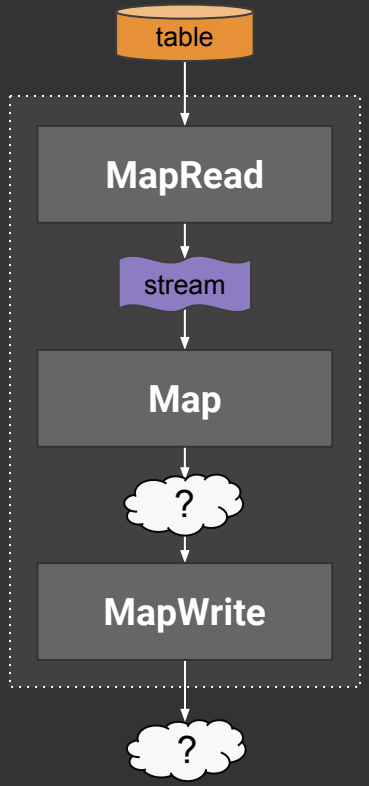
Map phase API

```
void map(K1 key, V1 value, Emit<K2, V2>);
```

Map phase API

```
void map(K1 key, V1 value, Emit<K2, V2>);
```

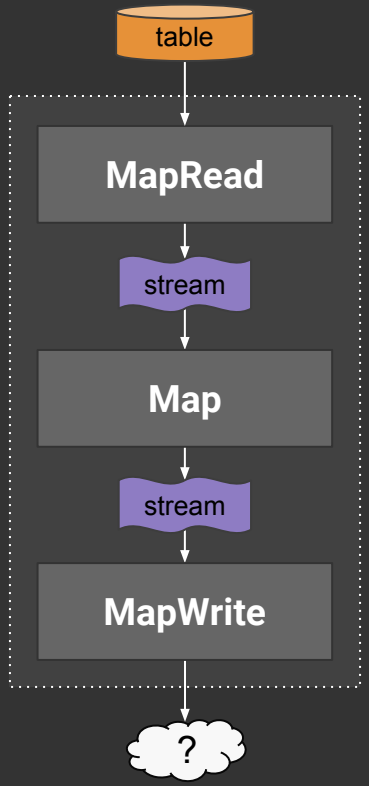
Map phase



Map phase API

```
void map(K1 key, V1 value, Emit<K2, V2>);
```

Map phase

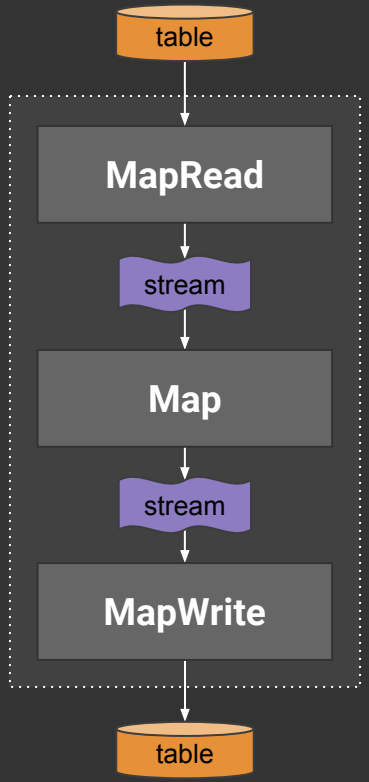


Map phase API

```
void map(K1 key, V1 value, Emit<K2, V2>);

void reduce(K2 key, Iterable<V2> value, Emit<V3>);
```

Map phase



MapReduce

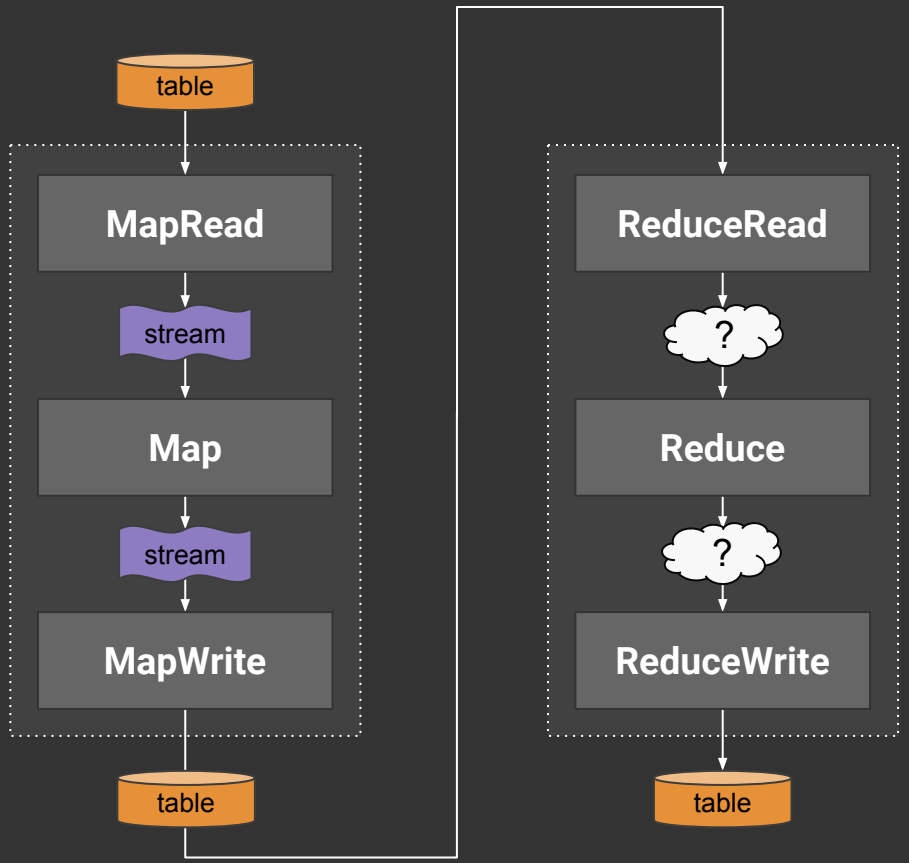


table
MapRead
stream
Map
stream
MapWrite
table
ReduceRead
?
Reduce
?
ReduceWrite
table

Map phase API

```
void map(K1 key, V1 value, Emit<K2, V2>);

void reduce(K2 key, Iterable<V2> value, Emit<V3>);
```

Map phase API

```
void map(K1 key, V1 value, Emit<K2, V2>);

void reduce(K2 key, Iterable<V2> value, Emit<V3>);
```

MapReduce

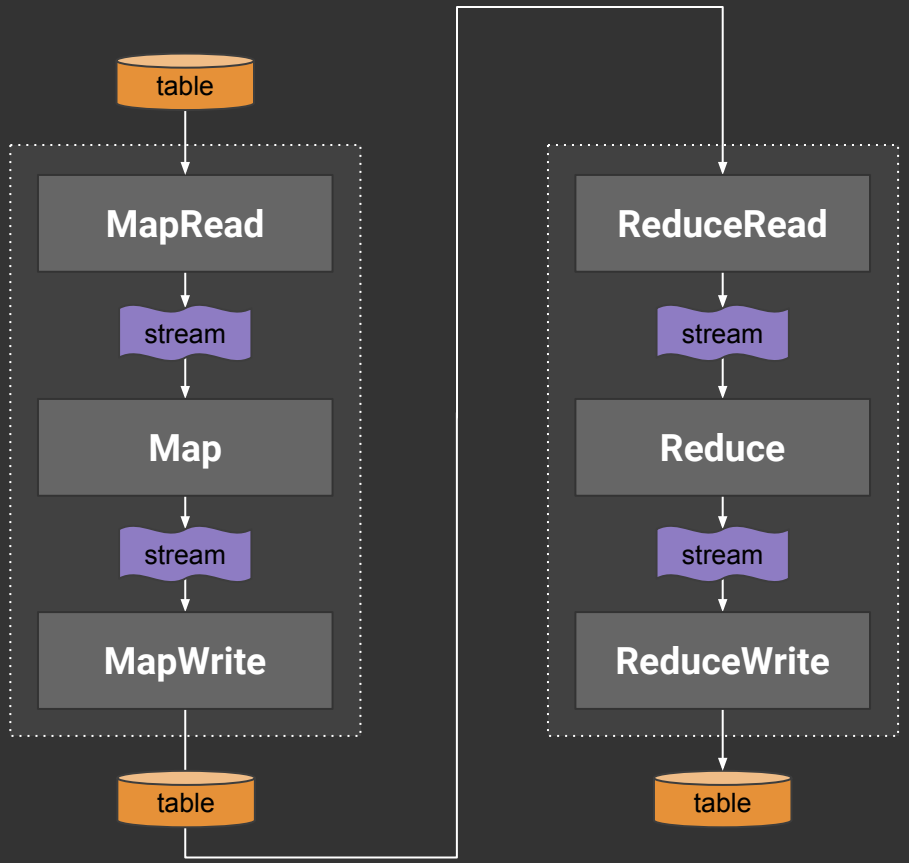


table
MapRead
stream
Map
stream
MapWrite
table
ReduceRead
stream
Reduce
stream
ReduceWrite
table

Reconciling streams & tables w/ the Beam Model

- How does batch processing fit into all of this?

1. **Tables** are read into **streams**.
2. **Streams** are processed into new **streams** until a grouping operation is hit.
3. Grouping turns the **stream** into a **table**.
4. Repeat steps 1-3 until you run out of operations.

Reconciling streams & tables w/ the Beam Model

- How does batch processing fit into all of this?
- What is the relationship of streams to bounded and unbounded datasets?

Streams are the ***in-motion*** form of data

both bounded ***and*** unbounded.

Reconciling streams & tables w/ the Beam Model

- How does batch processing fit into all of this?
- What is the relationship of streams to bounded and unbounded datasets?
- How do the four ***what***, ***where***, ***when***, ***how*** questions map onto a streams/tables world?

The Beam Model

What results are calculated?

Where in event time are results calculated?

When in processing time are results materialized?

How do refinements of results relate?

The Beam Model

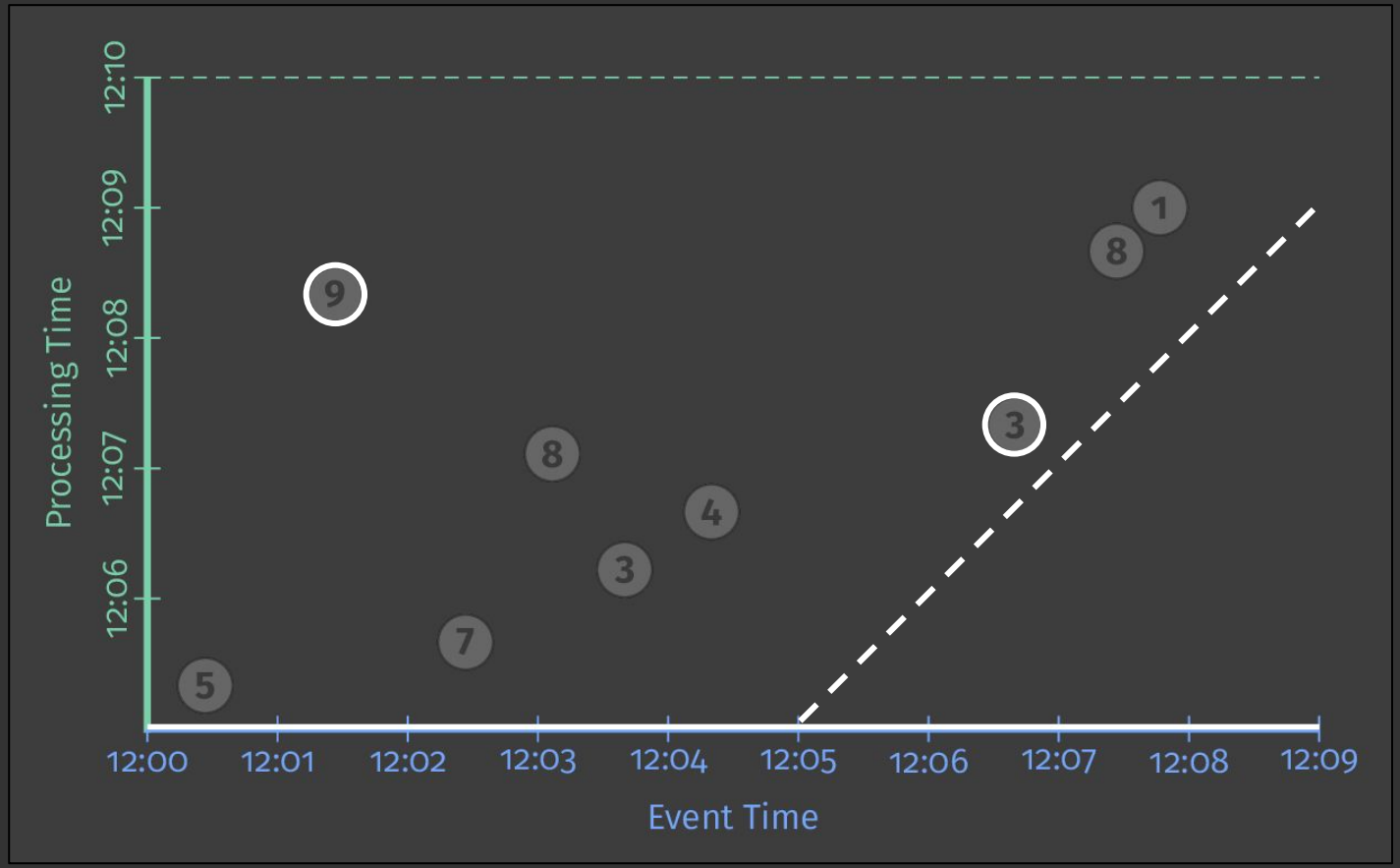
What results are calculated?

Where in event time are results calculated?

When in processing time are results materialized?

How do refinements of results relate?

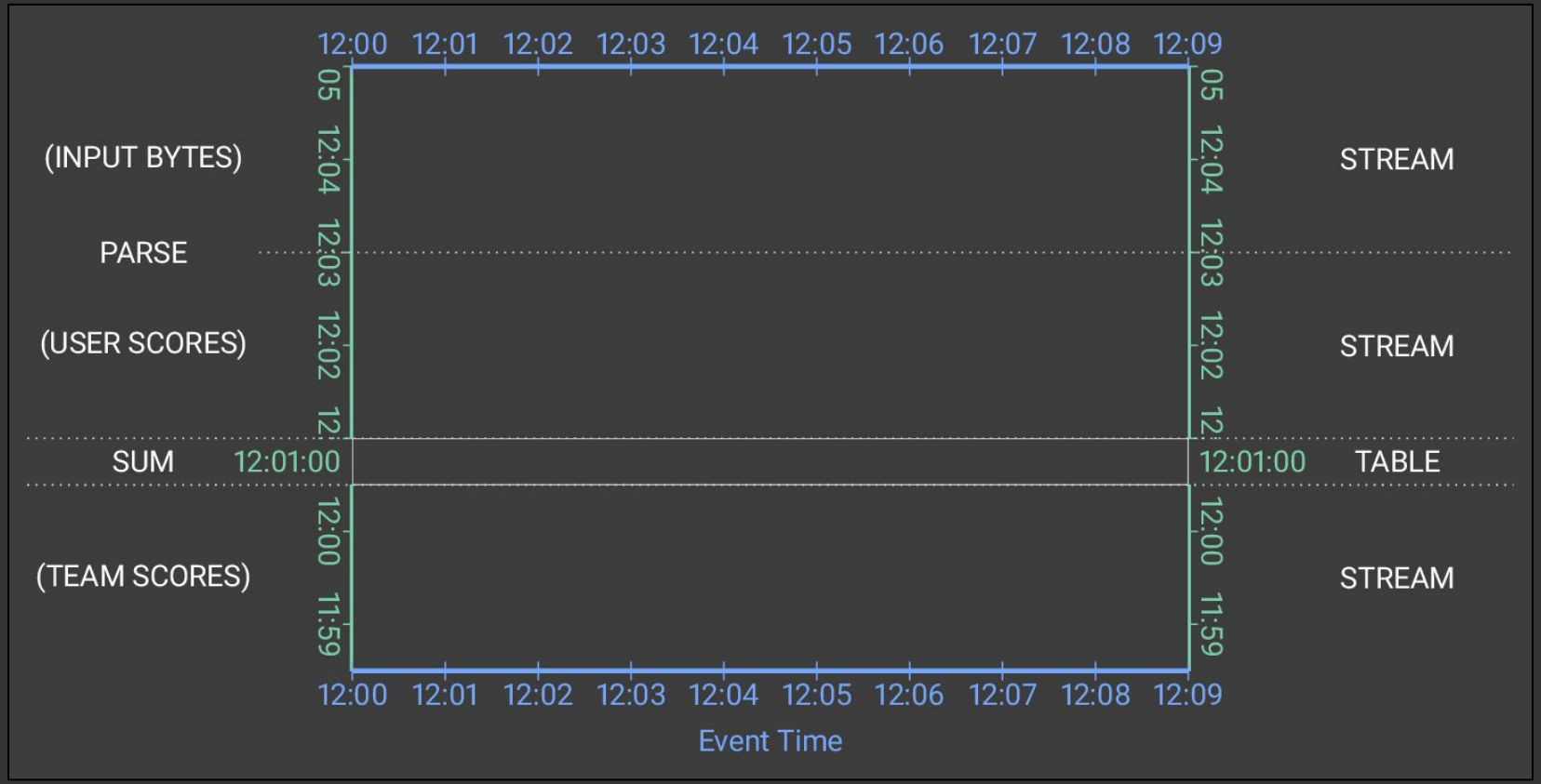
Example data: individual user scores



What is calculated?

```
PCollection<KV<Team, Score>> input = IO.read(...)
    .apply(ParDo.of(new ParseFn());
    .apply(Sum.integersPerKey());
```

What is calculated?



The Beam Model

What results are calculated?

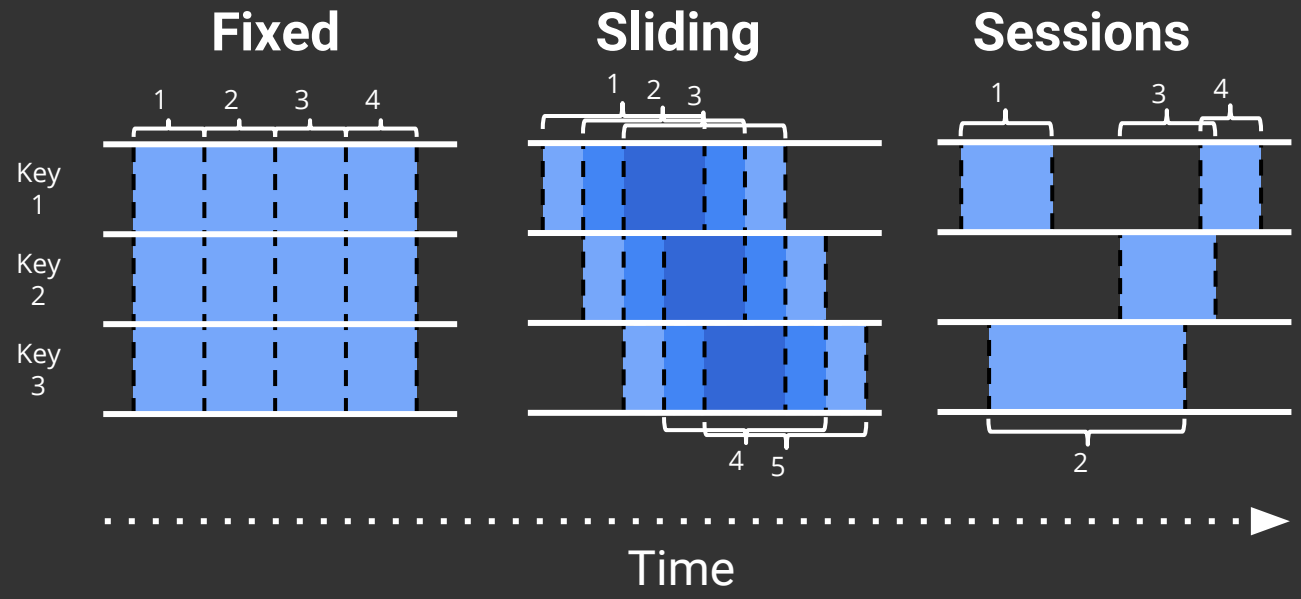
Where in event time are results calculated?

When in processing time are results materialized?

How do refinements of results relate?

Where in event time?

Windowing divides data into event-time-based finite chunks.

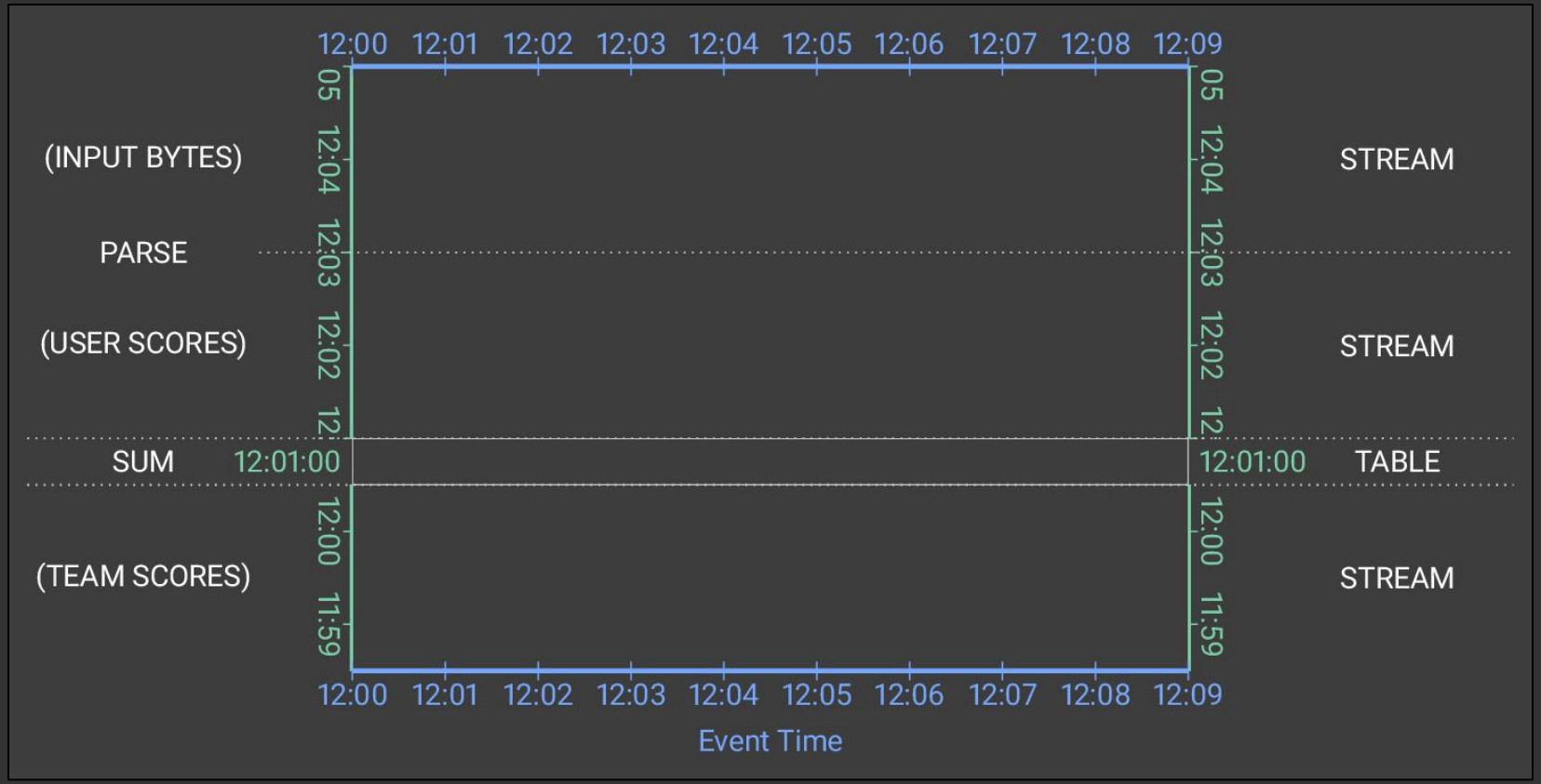


Often required when doing aggregations over unbounded data.

Where in event time?

```
PCollection<KV<User, Score>> input = IO.read(...)
    .apply(ParDo.of(new ParseFn());
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))))
    .apply(Sum.integersPerKey());
```

Where in event time?



The Beam Model

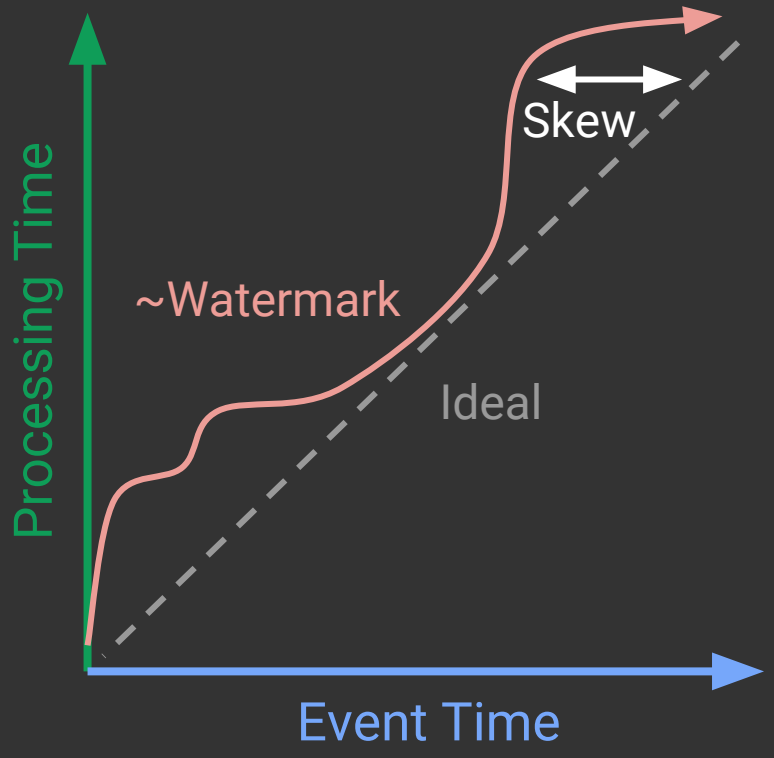
What results are calculated?

Where in event time are results calculated?

When in processing time are results materialized?

How do refinements of results relate?

When in processing time?

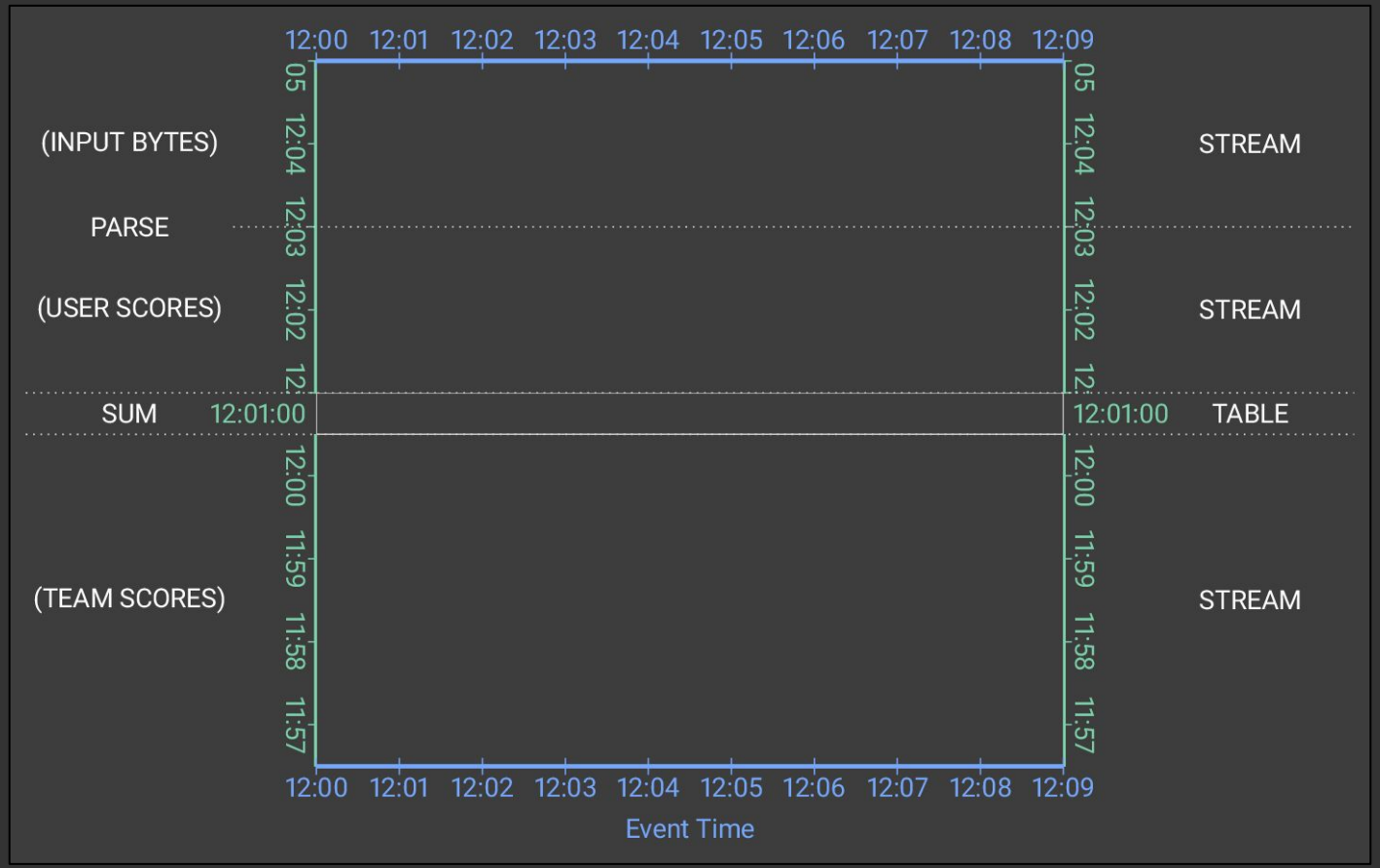


- Triggers control when results are emitted.
- Triggers are often relative to the watermark.

When in processing time?

```
PCollection<KV<User, Score>> input = IO.read(...)
    .apply(ParDo.of(new ParseFn());
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
                  .triggering(AtWatermark())
    .apply(Sum.integersPerKey());
```

When in processing time?



The Beam Model: asking the right questions

What results are calculated?

Where in event time are results calculated?

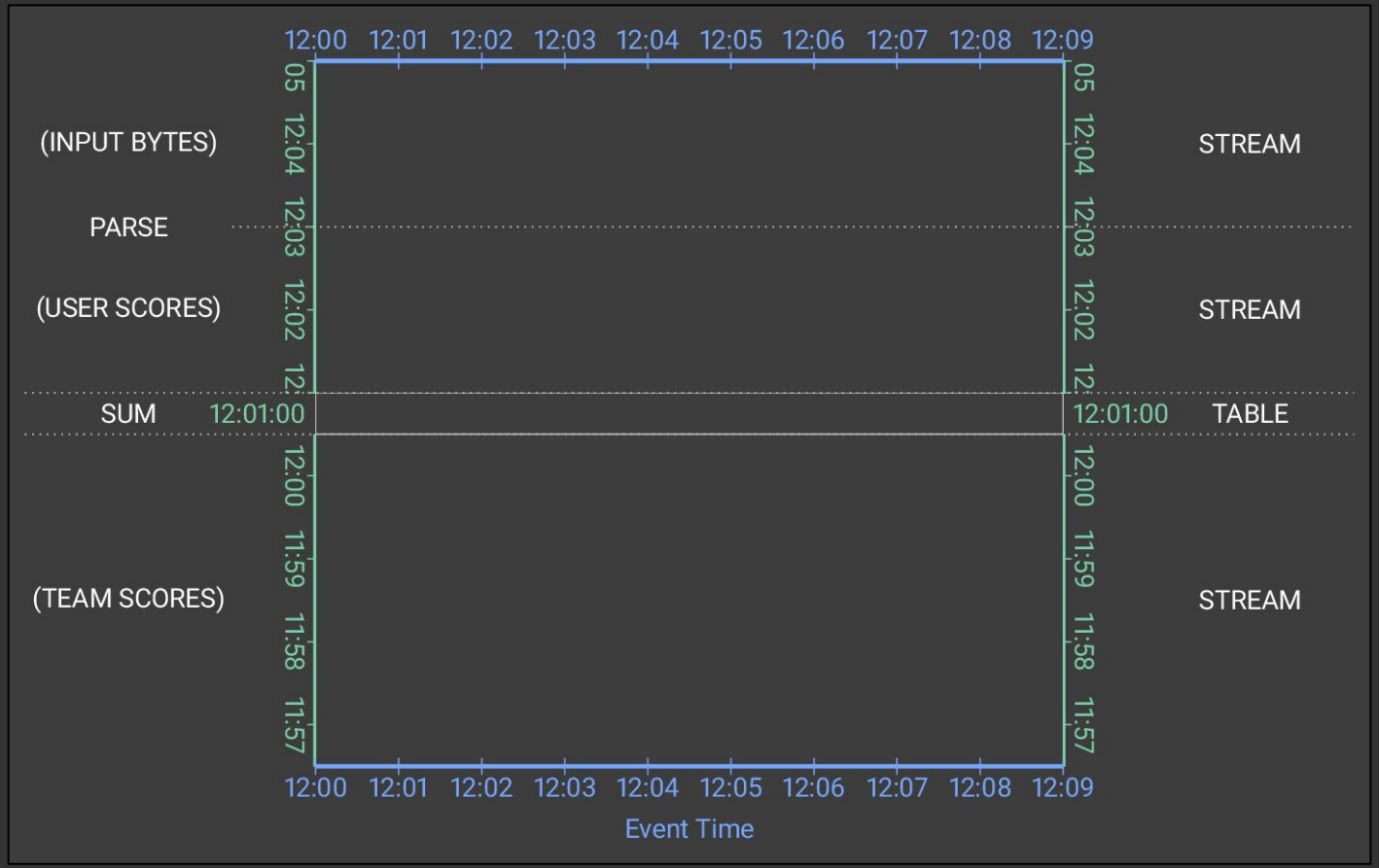
When in processing time are results materialized?

How do refinements of results relate?

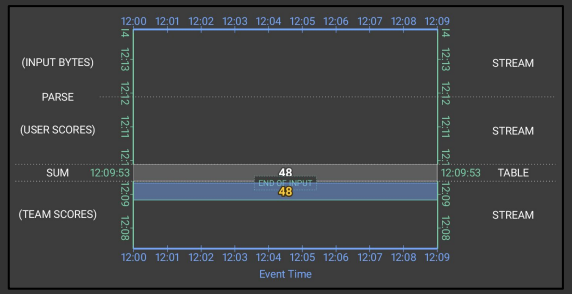
How do refinements relate?

```
PCollection<KV<User, Score>> input = IO.read(...)
    .apply(ParDo.of(new ParseFn());
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
                  .triggering(AtWatermark().withLateFirings(AtCount(1)))
                  .accumulatingFiredPanes())
    .apply(Sum.integersPerKey());
```

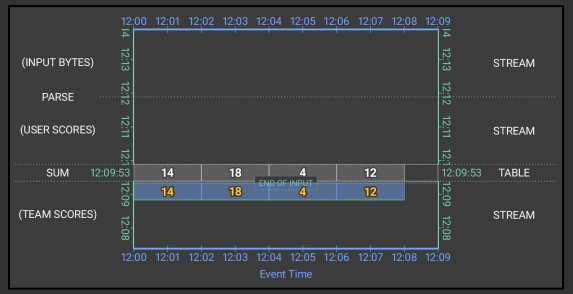
How do refinements relate?



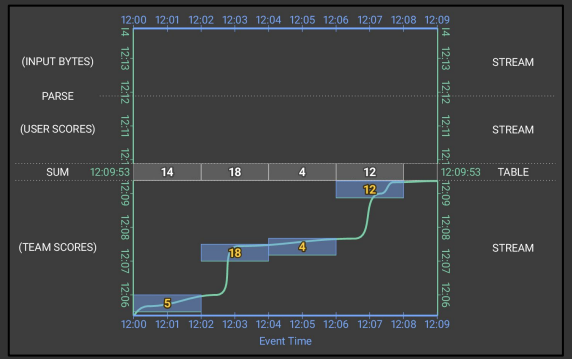
What/Where/When/How Summary



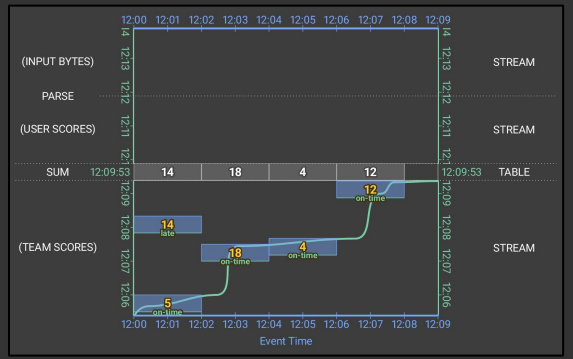
1. Classic Batch



2. Windowed Batch



3. Streaming



4. Streaming + Late Data Handling

Reconciling streams & tables w/ the Beam Model

- How does batch processing fit into all of this?
- What is the relationship of streams to bounded and unbounded datasets?
- How do the four ***what***, ***where***, ***when***, ***how*** questions map onto a streams/tables world?

General theory of stream & table relativity

Pipelines : **tables** + **streams** + **operations**
Tables : data at rest
Streams : data in motion
Operations : (**stream** | **table**) → (**stream** | **table**) transformations

- **stream** → **stream**: Non-grouping (element-wise) operations
  Leaves stream data in motion, yielding another stream.

- **stream** → **table**: Grouping operations
  Brings stream data to rest, yielding a table.
  Windowing adds the dimension of time to grouping.

- **table** → **stream**: Ungrouping (triggering) operations
  Puts table data into motion, yielding a stream.
  Accumulation dictates the nature of the stream (deltas, values, retractions).

- **table** → **table**: (none)
  Impossible to go from rest and back to rest without being put into motion.

02 Streaming SQL

Contorting relational algebra for fun and profit

A Time-varying relations

B SQL language extensions

Relational algebra

Relation

UserScores

User	Score	Time
Julie	7	12:01
Frank	3	12:03
Julie	1	12:03
Julie	4	12:07

Relational algebra

$\pi_{\text{Score,Time}}(\text{UserScores})$

Score	Time
7	12:01
3	12:03
1	12:03
4	12:07

SQL

```
SELECT Score, Time
FROM UserScores;
----------------
| Score | Time  |
----------------
|     7 | 12:01 |
|     3 | 12:03 |
|     1 | 12:03 |
|     4 | 12:07 |
----------------
```

Relations evolve over time

```
12:00> SELECT * FROM UserScores;
------------------------
| Name  | Score | Time  |
------------------------
------------------------
```

```
12:01> SELECT * FROM UserScores;
------------------------
| Name  | Score | Time  |
------------------------
| Julie |     7 | 12:01 |
------------------------
```

```
12:03> SELECT * FROM UserScores;
------------------------
| Name  | Score | Time  |
------------------------
| Julie |     7 | 12:01 |
| Frank |     3 | 12:03 |
| Julie |     1 | 12:03 |
------------------------
```

```
12:07> SELECT * FROM UserScores;
------------------------
| Name  | Score | Time  |
------------------------
| Julie |     7 | 12:01 |
| Frank |     3 | 12:03 |
| Julie |     1 | 12:03 |
| Julie |     4 | 12:07 |
------------------------
```

Classic SQL vs Streaming SQL

Classic SQL :: classic relations :: single point in time

Streaming SQL :: time-varying relations :: every point in time

Classic SQL vs Streaming SQL

Classic SQL :: classic relations :: single point in time

Streaming SQL :: time-varying relations :: every point in time

Classic relations

```
12:00> SELECT * FROM UserScores;
------------------------
| Name  | Score | Time  |
------------------------
------------------------
```

```
12:01> SELECT * FROM UserScores;
------------------------
| Name  | Score | Time  |
------------------------
| Julie |     7 | 12:01 |
------------------------
```

```
12:03> SELECT * FROM UserScores;
------------------------
| Name  | Score | Time  |
------------------------
| Julie |     7 | 12:01 |
| Frank |     3 | 12:03 |
| Julie |     1 | 12:03 |
------------------------
```

```
12:07> SELECT * FROM UserScores;
------------------------
| Name  | Score | Time  |
------------------------
| Julie |     7 | 12:01 |
| Frank |     3 | 12:03 |
| Julie |     1 | 12:03 |
| Julie |     4 | 12:07 |
------------------------
```

Time-varying relation

```
12:07> SELECT * FROM UserScores;
```

```
[-inf, 12:01)
------------------------
| Name  | Score | Time  |
------------------------
|       |       |       |
|       |       |       |
|       |       |       |
|       |       |       |
------------------------
```

```
[12:01, 12:03)
------------------------
| Name  | Score | Time  |
------------------------
| Julie |     7 | 12:01 |
|       |       |       |
|       |       |       |
|       |       |       |
------------------------
```

```
[12:03, 12:07)
------------------------
| Name  | Score | Time  |
------------------------
| Julie |     7 | 12:01 |
| Frank |     3 | 12:03 |
| Julie |     1 | 12:03 |
|       |       |       |
------------------------
```

```
[12:07, now)
------------------------
| Name  | Score | Time  |
------------------------
| Julie |     7 | 12:01 |
| Frank |     3 | 12:03 |
| Julie |     1 | 12:03 |
| Julie |     4 | 12:07 |
------------------------
```

Closure property of relational algebra
remains intact with time-varying relations.

Time-varying relations: variations

```
12:07> SELECT * FROM UserScores;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	7	12:01	Julie	7	12:01
						Frank	3	12:03	Frank	3	12:03
						Julie	1	12:03	Julie	1	12:03
									Julie	4	12:07

Time-varying relations: filtering

```
12:07> SELECT * FROM UserScores;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	7	12:01	Julie	7	12:01
						Frank	3	12:03	Frank	3	12:03
						Julie	1	12:03	Julie	1	12:03
									Julie	4	12:07

```
12:07> SELECT * FROM UserScores WHERE Name = "Julie";
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	7	12:01	Julie	7	12:01
						Julie	1	12:03	Julie	1	12:03
									Julie	4	12:07

Time-varying relations: grouping

```
12:07> SELECT * FROM UserScores;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	7	12:01	Julie	7	12:01
						Frank	3	12:03	Frank	3	12:03
						Julie	1	12:03	Julie	1	12:03
									Julie	4	12:07

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

How does this relate to streams & tables?

Time-varying relations: tables

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

12:00> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;

Name	Score	Time

12:01> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;

Name	Score	Time
Julie	7	12:01

12:03> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;

Name	Score	Time
Julie	8	12:03
Frank	3	12:03

12:07> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;

Name	Score	Time
Julie	12	12:07
Frank	3	12:03

Time-varying relations: tables

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

```
12:01> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01

```
12:07> SELECT TABLE Name, SUM(Score), MAX(Time) AS OF SYSTEM TIME '12:01' FROM UserScores GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01

Time-varying relations: tables

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

```
12:00> SELECT TABLE Name,
SUM(Score), MAX(Time) FROM
UserScores GROUP BY Name;
```

Name	Score	Time

```
12:00> SELECT STREAM Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

Name	Score	Time

...

12:00

Time-varying relations: streams

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

```
12:00> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;
```

Name	Score	Time

```
12:00> SELECT STREAM Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01

...

12:01

Time-varying relations: streams

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

```
12:01> SELECT TABLE Name,
SUM(Score), MAX(Time) FROM
UserScores GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01

```
12:00> SELECT STREAM Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01
...		

12:01

Time-varying relations: streams

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

```
12:01> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01

```
12:00> SELECT STREAM Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01
Frank	3	12:03
Julie	8	12:03

...

12:03

Time-varying relations: streams

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

```
12:03> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;
```

Name	Score	Time
Julie	8	12:03
Frank	3	12:03

```
12:00> SELECT STREAM Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01
Frank	3	12:03
Julie	8	12:03

...

12:03

Time-varying relations: streams

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

```
12:03> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;
```

Name	Score	Time
Julie	8	12:03
Frank	3	12:03

```
12:00> SELECT STREAM Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01
Frank	3	12:03
Julie	8	12:03
Julie	12	12:07
...		

12:07

Time-varying relations: streams

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

[-inf, 12:01)			[12:01, 12:03)			[12:03, 12:07)			[12:07, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	7	12:01	Julie	8	12:03	Julie	12	12:07
						Frank	3	12:03	Frank	3	12:03

```
12:07> SELECT TABLE Name, SUM(Score), MAX(Time) FROM UserScores GROUP BY Name;
```

Name	Score	Time
Julie	12	12:07
Frank	3	12:03

```
12:00> SELECT STREAM Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

Name	Score	Time
Julie	7	12:01
Frank	3	12:03
Julie	8	12:03
Julie	12	12:07
...		

12:07

How does this relate to streams & tables?

Tables capture a point-in-time snapshot of a time-varying relation.

Streams capture the evolution of a time-varying relation over time.

02 Streaming SQL

Contorting relational algebra for fun and profit

A Time-varying relations

B SQL language extensions

When do you need SQL extensions for streaming?

good defaults = often not needed

How is output consumed?

As a table: SQL extensions rarely needed.

As a stream: SQL extensions sometimes needed.

When do you need SQL extensions for streaming?*

Explicit table / stream selection

- SELECT TABLE * from X;
- SELECT STREAM * from X;

Timestamps and windowing

- Event-time columns
- Windowing. E.g.,
  `SELECT * FROM X GROUP BY SESSION(<COLUMN> INTERVAL '5' MINUTE);`
  - Grouping by timestamp
  - Complex multi-row transactions inexpressible in declarative SQL (e.g., session windows)

Sane default table / stream selection

- If all inputs are tables, output is a table
- If any inputs are streams, output is a stream

Simple triggers

- Implicitly defined by characteristics of the sink
- Optionally be configured outside of query.
- Per-query, e.g.: `SELECT * from X EMIT <WHEN>;`
- Focused set of use cases:
  - Repeated updates
    `... EMIT AFTER <TIMEDELTA>`
  - Completeness
    `... EMIT WHEN WATERMARK PAST <COLUMN>`
  - Repeated updates + completeness (e.g., early/on-time/late pattern)
    `... EMIT AFTER <TIMEDELTA> AND WHEN WATERMARK PAST <COLUMN>`

* Most of these extensions are theoretical at this point; very few have concrete implementations.

Summary

streams ⇄ tables

streams & tables : Beam Model

time-varying relations

SQL language extensions

Thank you!

These slides: http://s.apache.org/streaming-sql-big-data-spain

Streaming SQL spec (WIP: Apex, Beam, Calcite, Flink): http://s.apache.org/streaming-sql-spec
Streaming in Calcite (Julian Hyde): https://calcite.apache.org/docs/stream.html
Streams, joins & temporal tables (Julian Hyde): http://s.apache.org/streams-joins-and-temporal-tables

Streaming 101: http://oreilly.com/ideas/the-world-beyond-batch-streaming-101
Streaming 102: http://oreilly.com/ideas/the-world-beyond-batch-streaming-102

Apache Beam: http://beam.apache.org
Apache Calcite: http://calcite.apache.org
Apache Flink: http://flink.apache.org

Twitter: @takidau

In early release now
streamingsystems.net