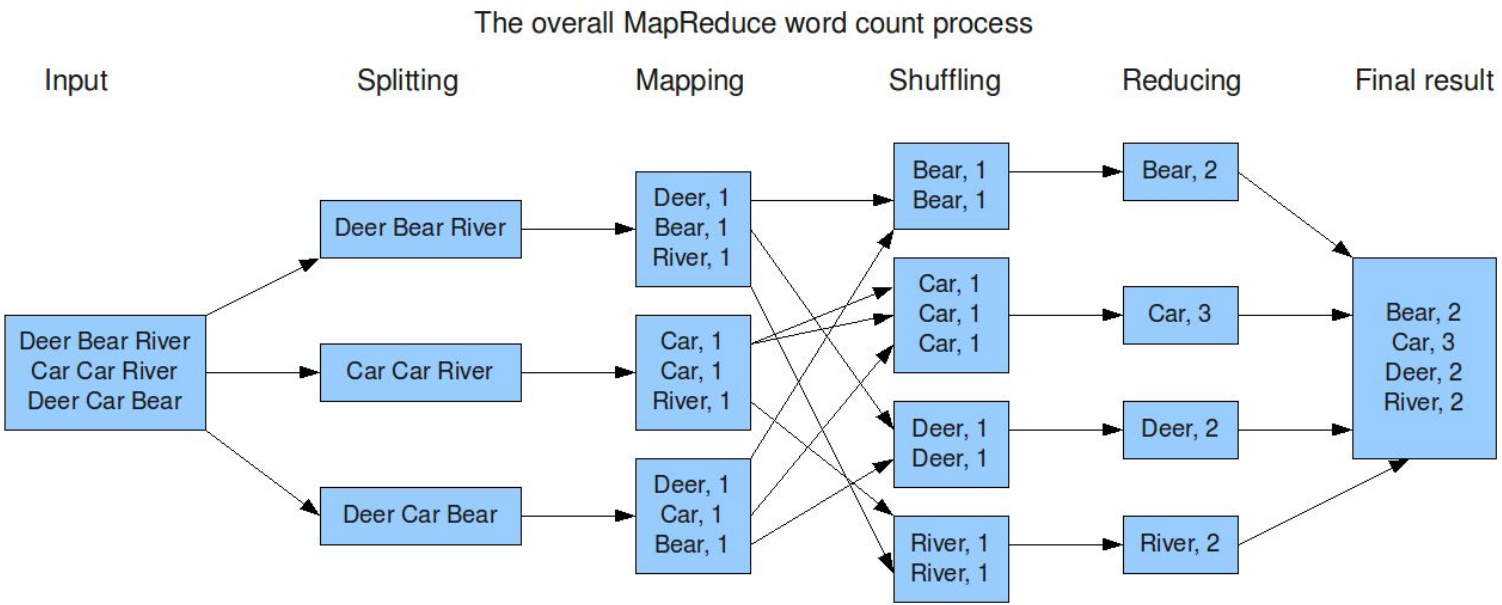
RICKER LYMAN
ROBOTIC

Parallel, Functional & Streaming Programming with Scala

Introduction to STREAM COMPUTING

Map reduce


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

Map reduce

Good

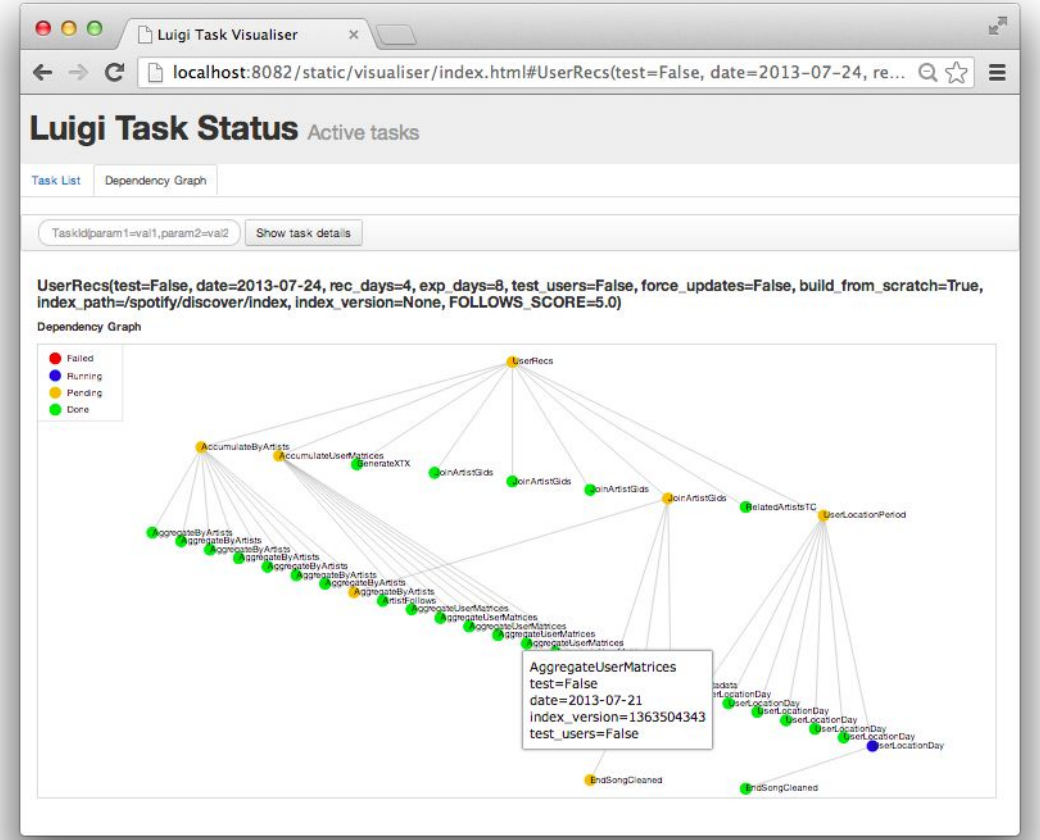
- Hadoop distributed file system (HDFS)
- Distributed & massively parallel
- Move the compute to the data
- **Accomplish what was impossible before**

Not so good

- **Finite data set**
- **Batch process**
- Begin to chain jobs together
- Failure?
- Recovery?
- Idempotent?

Workflows

Oozie
Luigi
Azkaban
Airflow
Pinball
Cascading
Taskflow



Streaming frameworks

Streams are all around us

Stock market trading

Social media

Sensors

Weather

Transportation traffic

Video

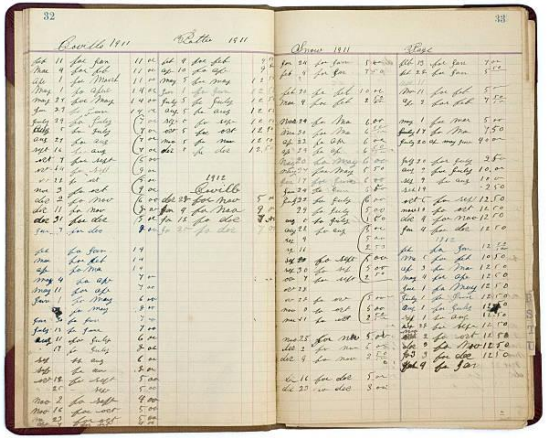
The opposite of stream is BATCH processing

Has a beginning and an end

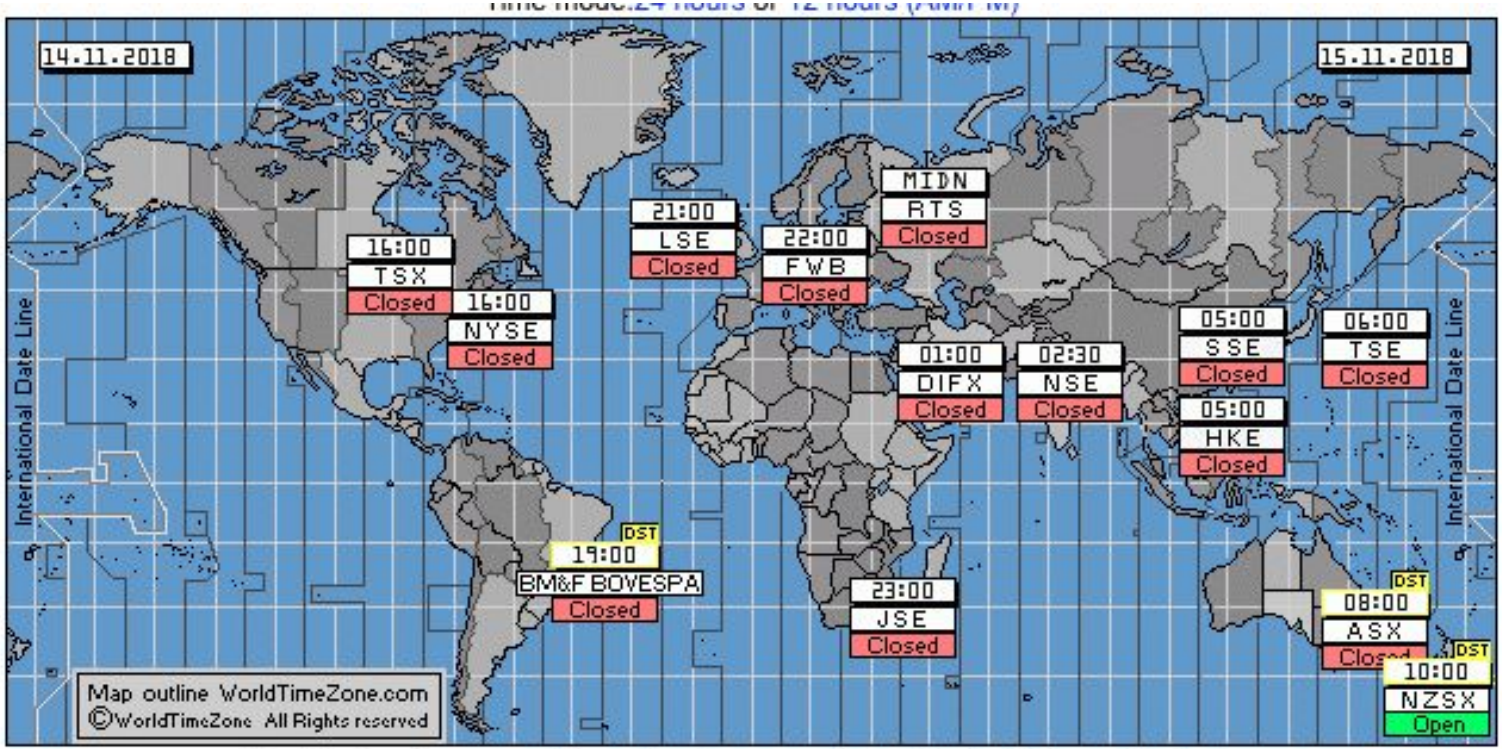
The data is time bounded and finite

Example:

- Sales from a single store in a month
- One day trading on a single stock exchange



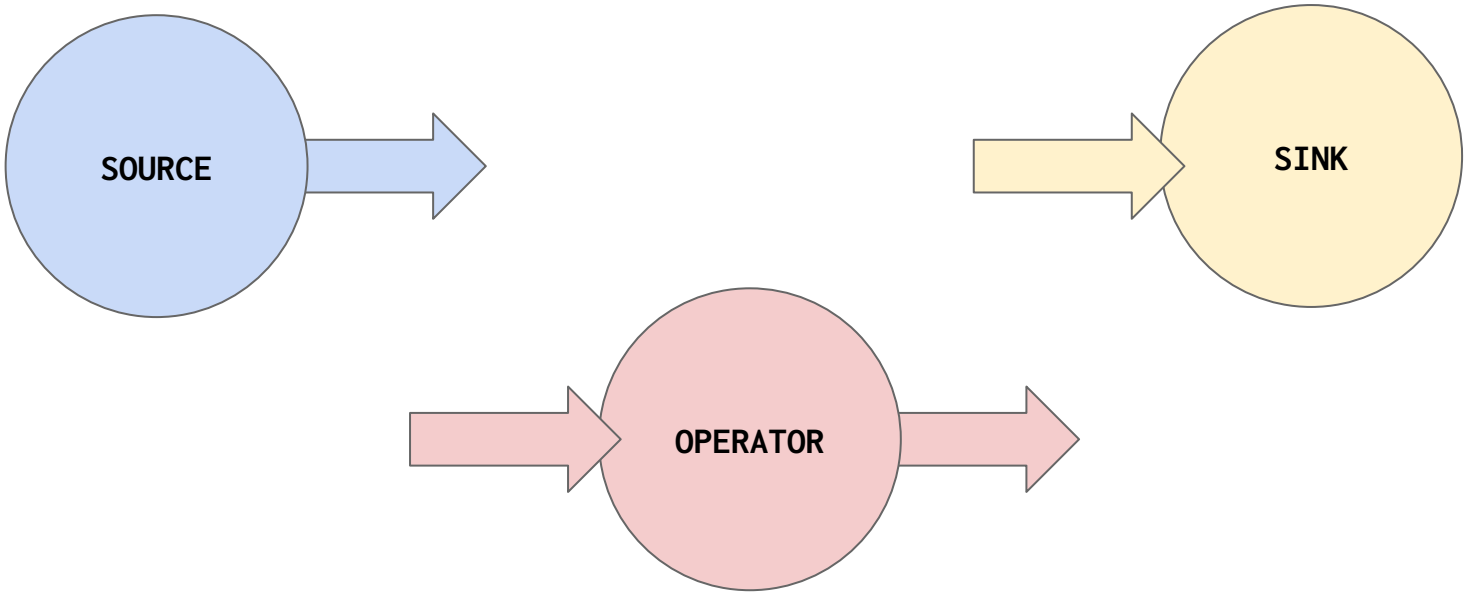
Stocks are ticking somewhere



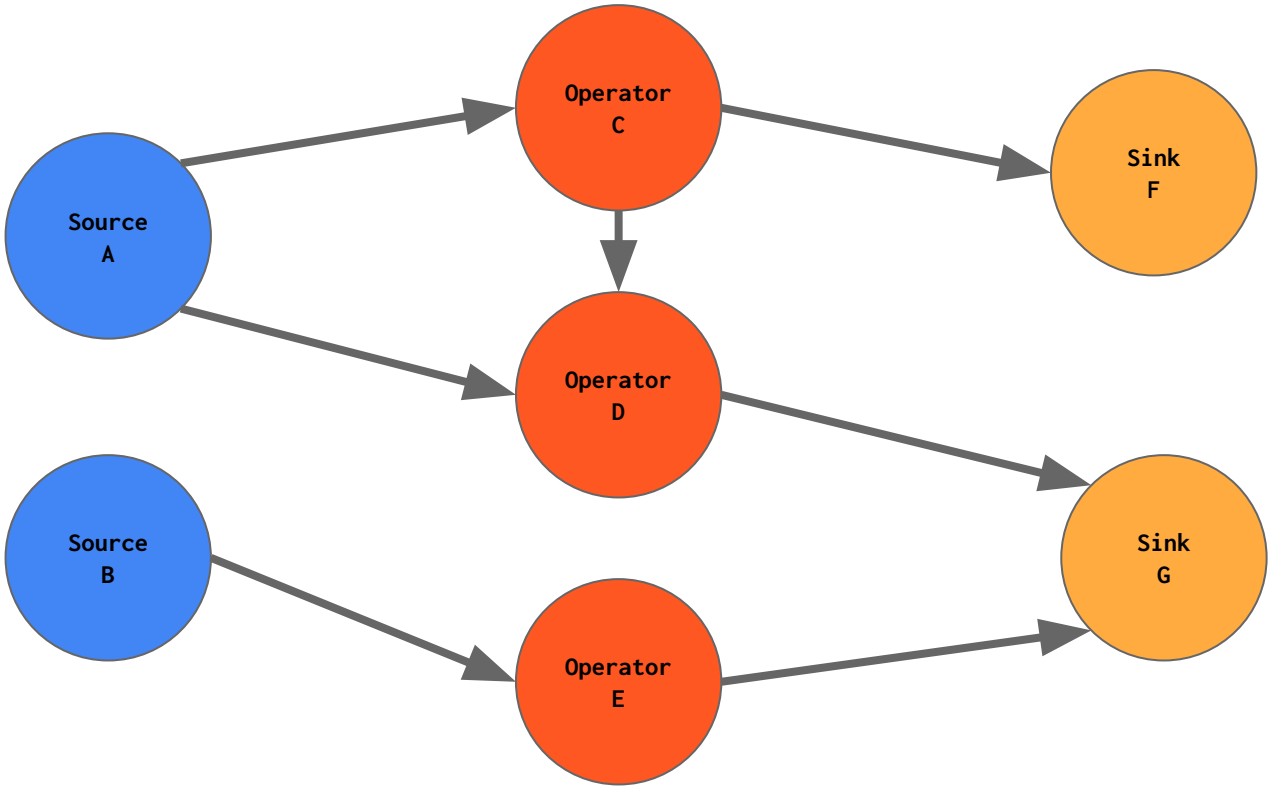
The real world never shuts up, never pauses, never stops.

So how do we program it?

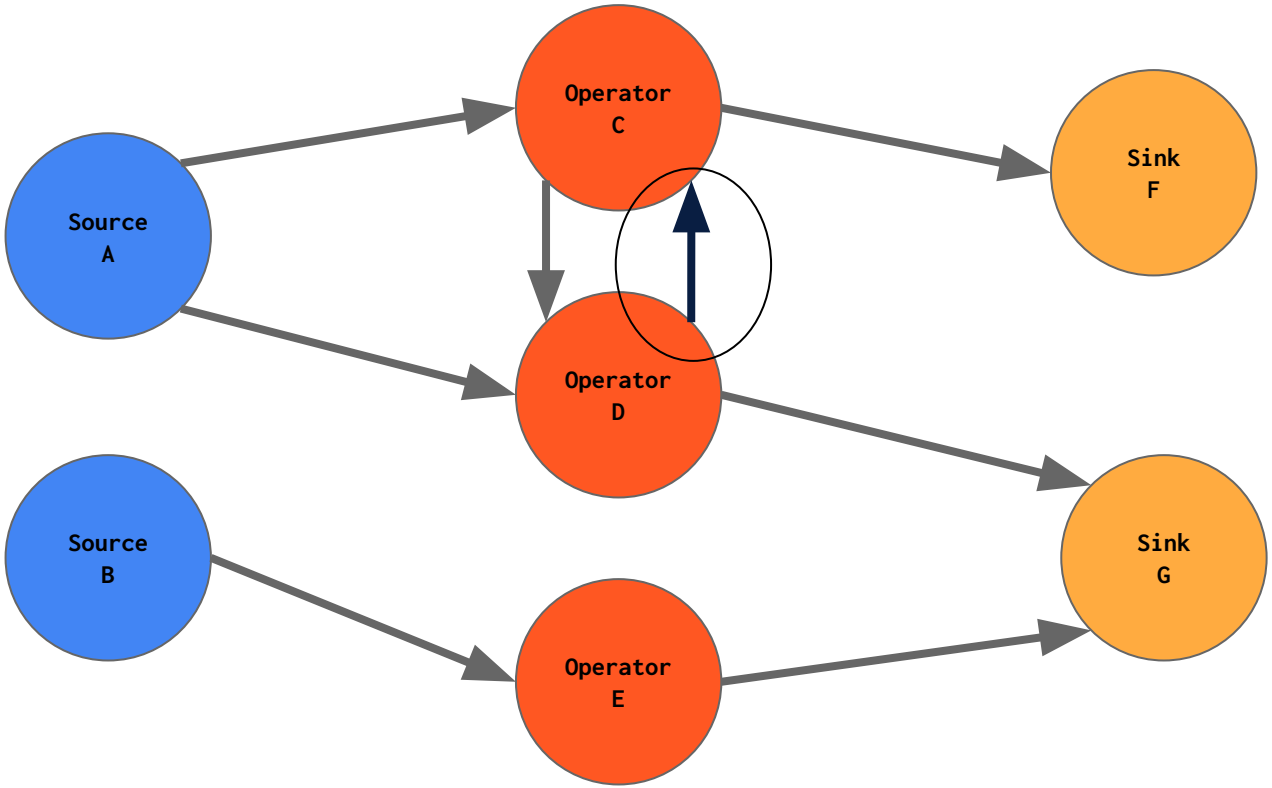
Source, sink, operator



Directed acyclic graph (DAG)



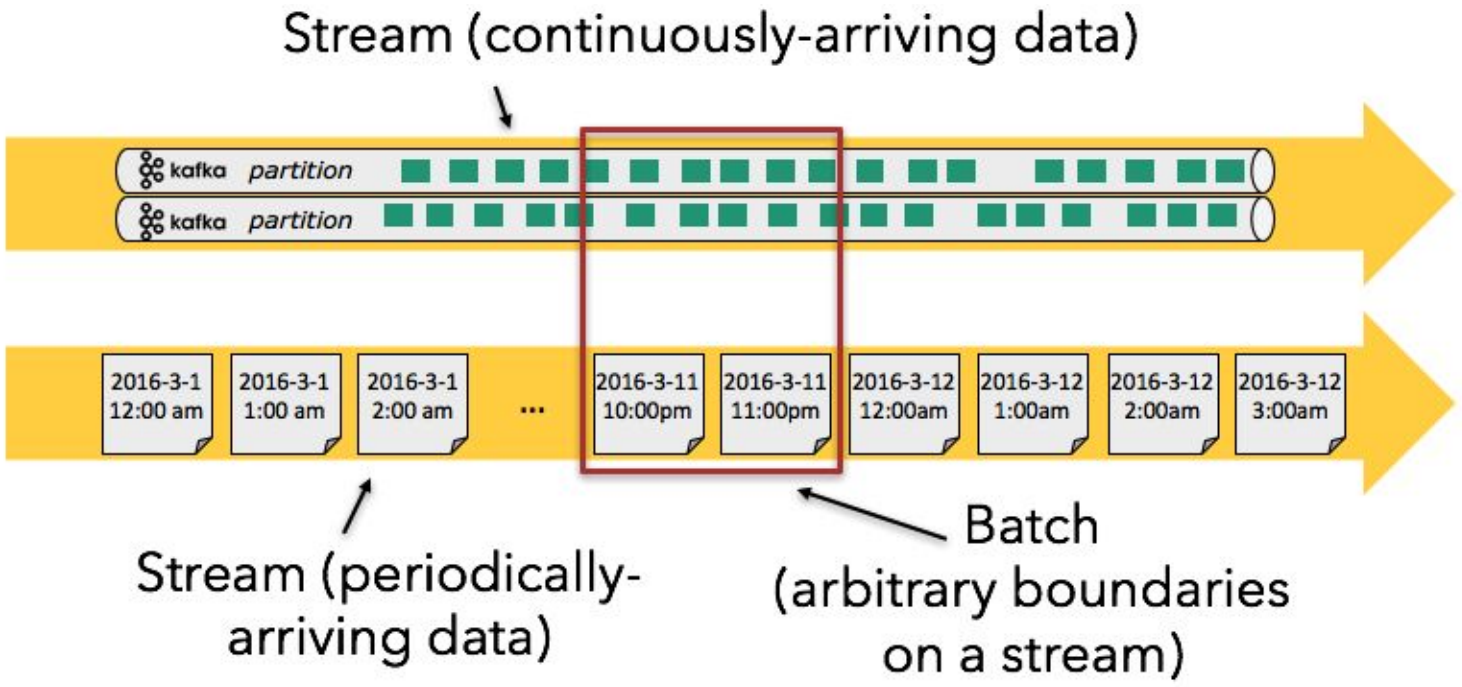
Cyclic Bad!


Operator
C
Sink
F
Source
A
Operator
D
Source
B
Sink
G
Operator
E

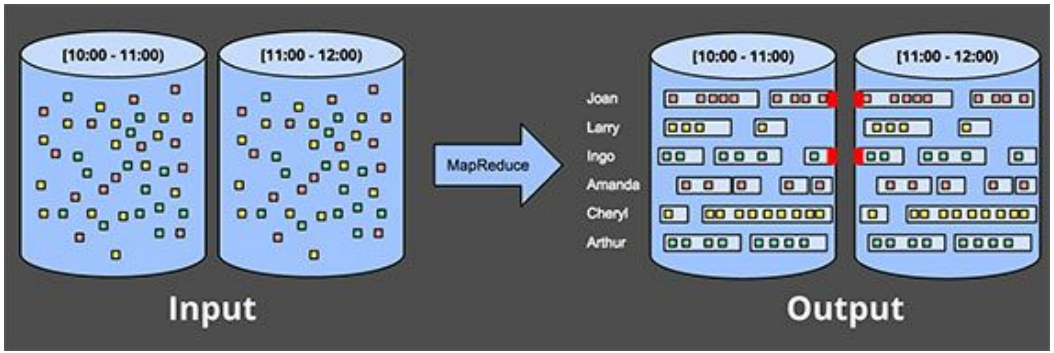
Streams are different

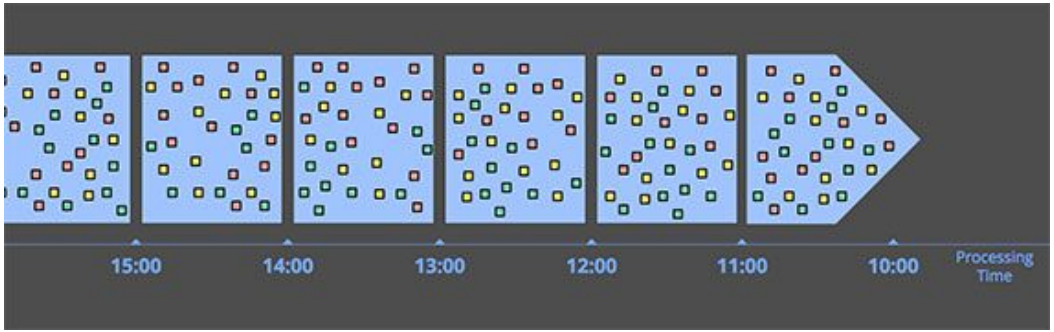
1. Data is unbounded or infinite
2. Continuous processing
3. There is no now
4. Eventual consistency vs false sense of consistency
5. Closer to reality

#1 Unbounded


Stream (continuously-arriving data)
kafka partition
kafka partition
2016-3-1 12:00 am
2016-3-1 1:00 am
2016-3-1 2:00 am
...
2016-3-11 10:00pm
2016-3-11 11:00pm
2016-3-12 12:00am
2016-3-12 1:00am
2016-3-12 2:00am
2016-3-12 3:00am
Stream (periodically-arriving data)
Batch (arbitrary boundaries on a stream)

#2 Batch vs stream





Credit:
Tyler Akidau.

Calculate closing (end of day) balance

Batch

Start with yesterday’s closing balance

Add up all the events for the day

Publish a new closing balance

Stream

Recalculate with each new event

Emit new balance with each event

Tag a balance event as closing balance

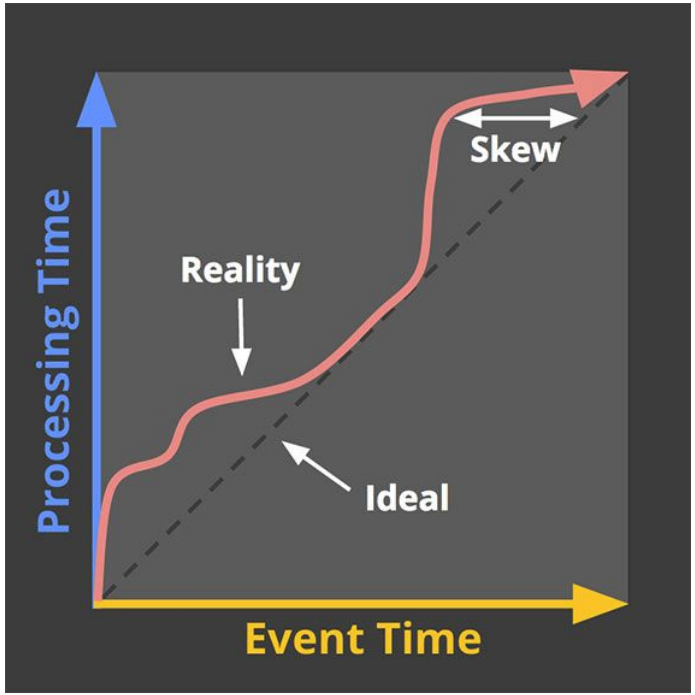
#3 Time

Event time:
the time at which events actually occurred

Ingest time:
the time the event is written to the system

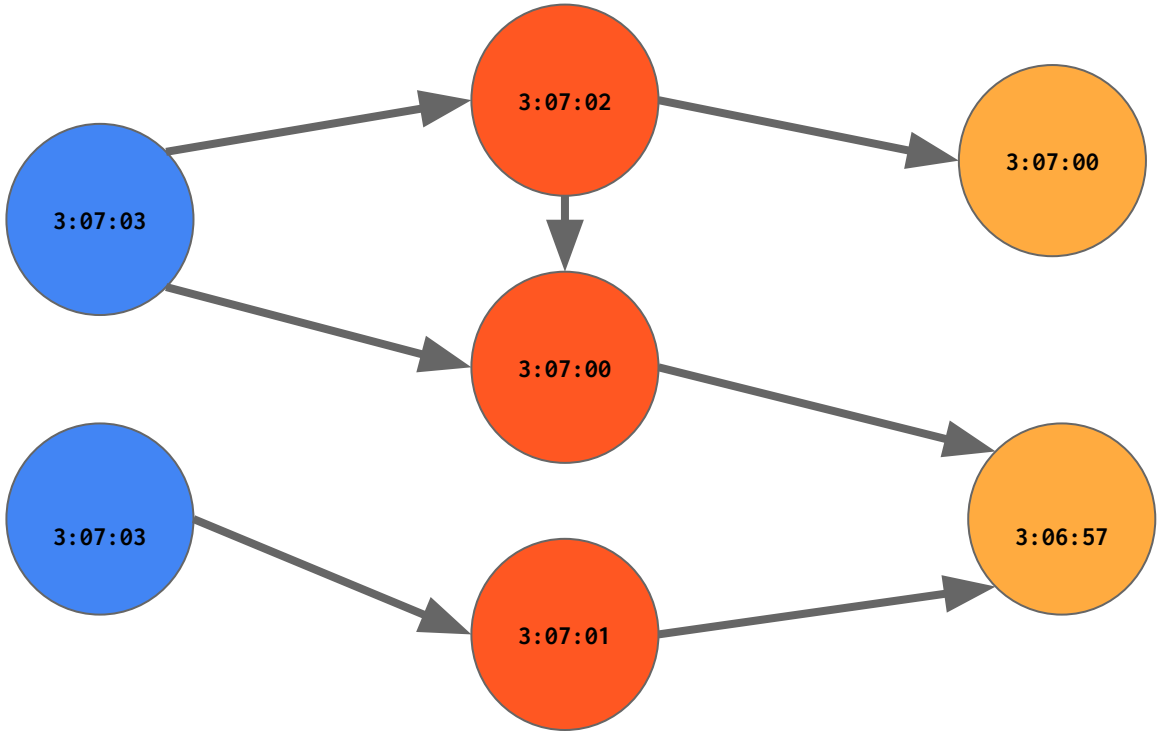
Process time:
the time at which events are observed in the system

Time skew



Credit:
Tyler Akidau.

When is now?



#4 Consistency

Trade data arrives at end of day (EOD)

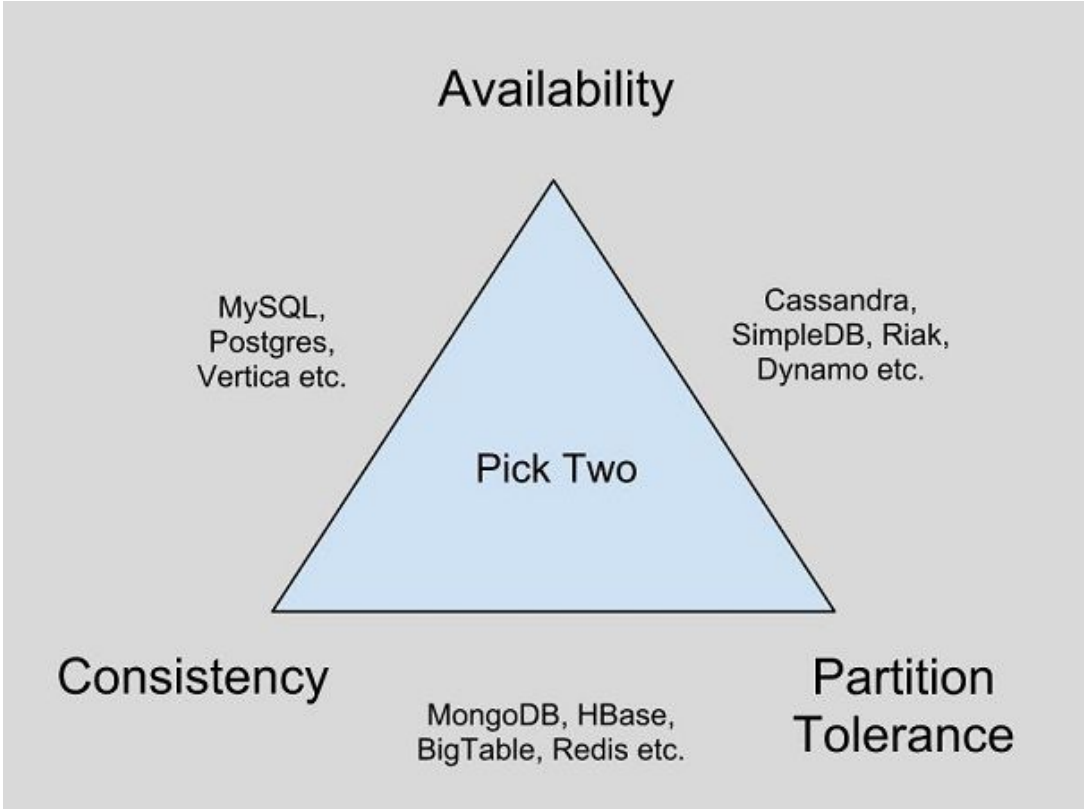
Processing runs to create EOD status of trades

Corrections exist for previous days

Previous EOD is also changed

Batched processes give us a false sense of consistency

Eventual consistency



Eventual consistency

Applies to organizations too

Eventually all the downstream processes will have consistent values

Batch:
Around 6 pm all the systems have the same value,
but it is probably wrong

Stream:
At 6:00:00.0 pm all systems have a value,
but perhaps not the exact same value

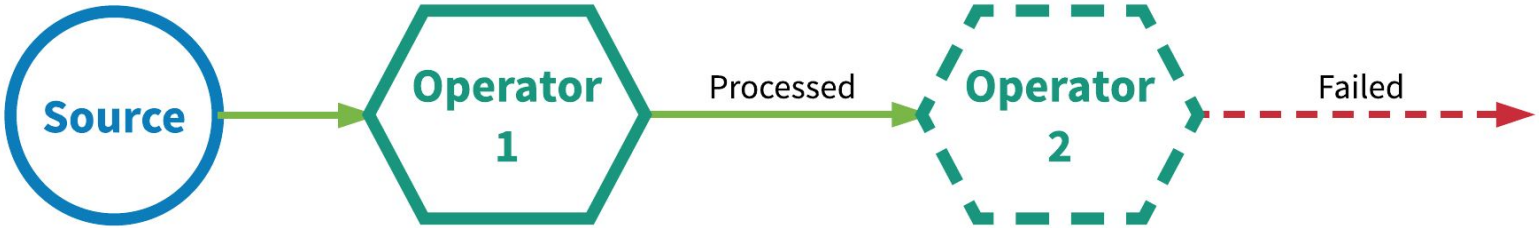
Considerations

1. Message delivery
    a. At most once
    b. At least once
    c. Exactly once
2. Windowing
    a. Fixed
    b. Sliding
    c. Session
3. Joins
    a. Inner
    b. Left
    c. Outer
4. Scaling

At most once delivery

“Best effort” approach

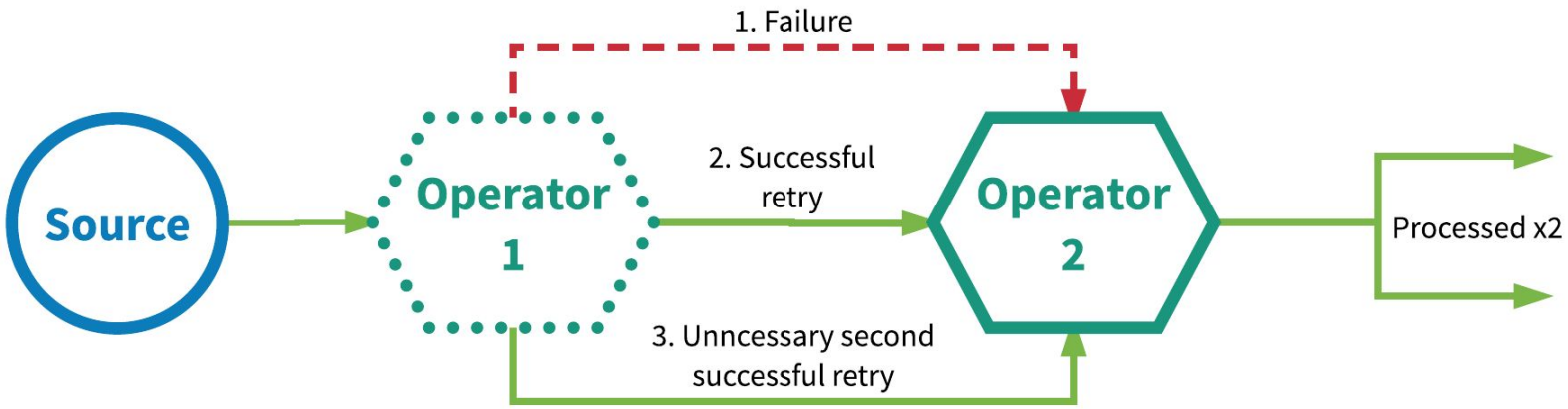
Messages may be lost



At least once delivery

“Guaranteed delivery”

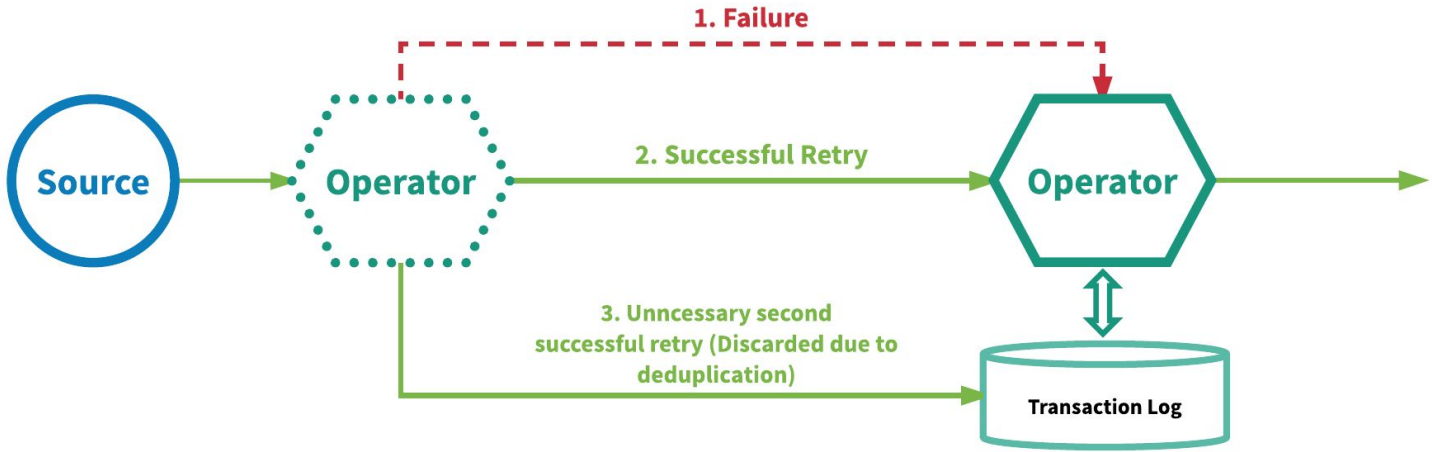
Duplicates will occur



Exactly once delivery

Expensive!

- Distributed snapshot/state checkpointing
- At-least-once event delivery plus message deduplication



Hurst's Law

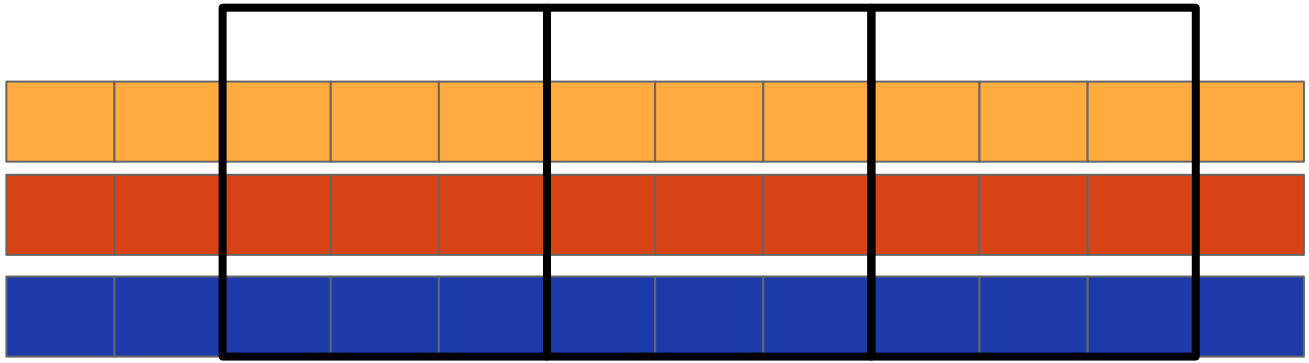
Complexity can neither be created nor destroyed; it can only be displaced.

Pay attention to where it went!

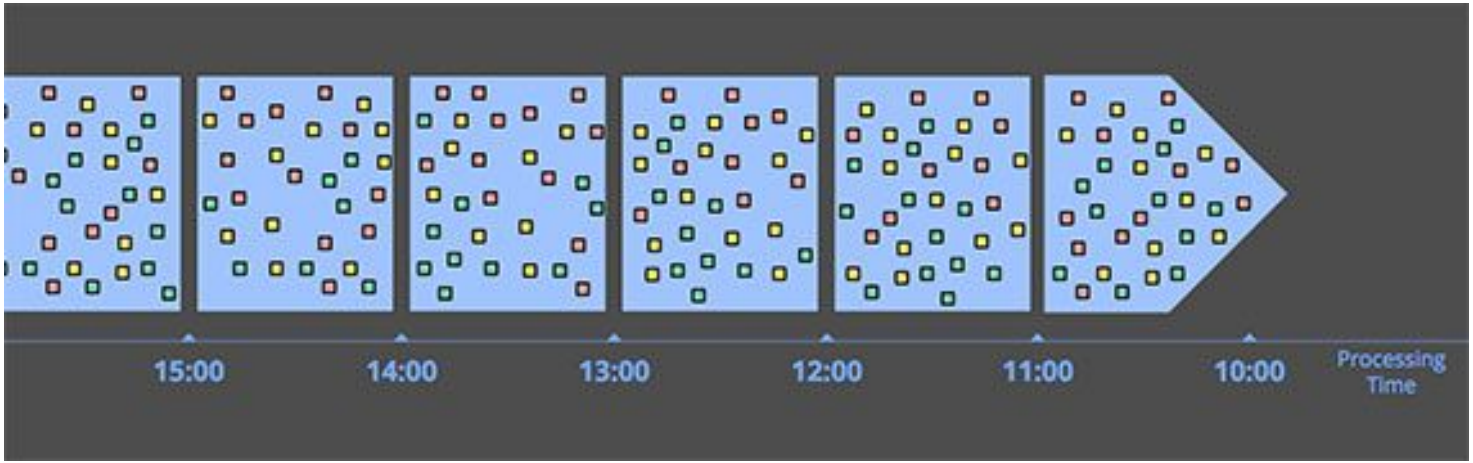
Fixed window

Time is partitioned into same-length, non-overlapping chunks.

Each event belongs to exactly one window

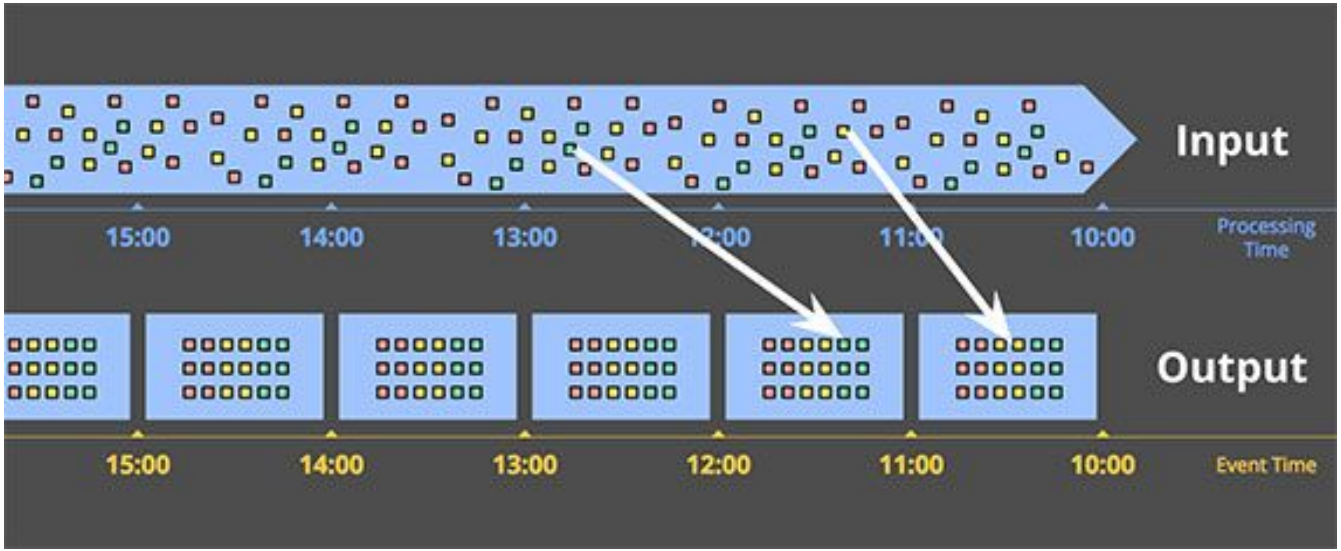


Window by process time



Credit:
Tyler Akidau.

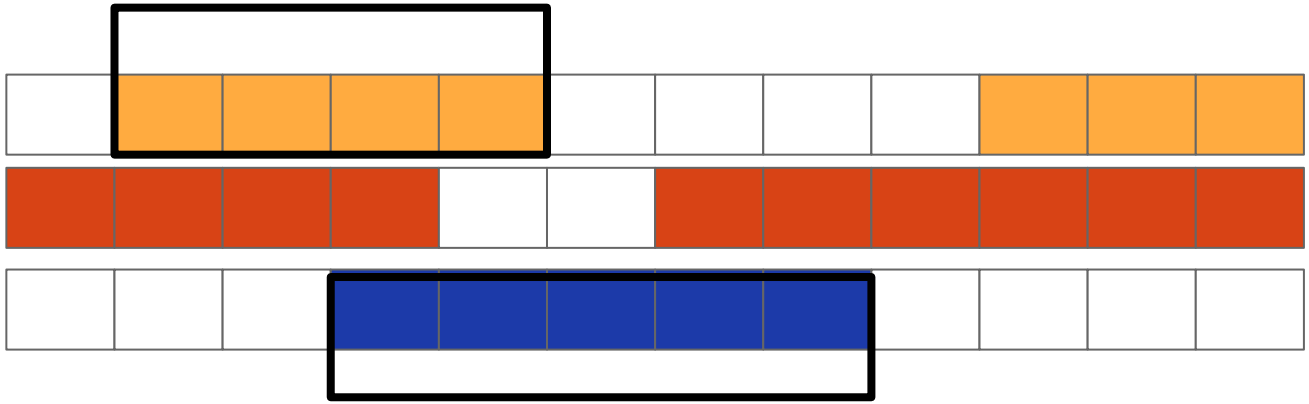
Window by event time



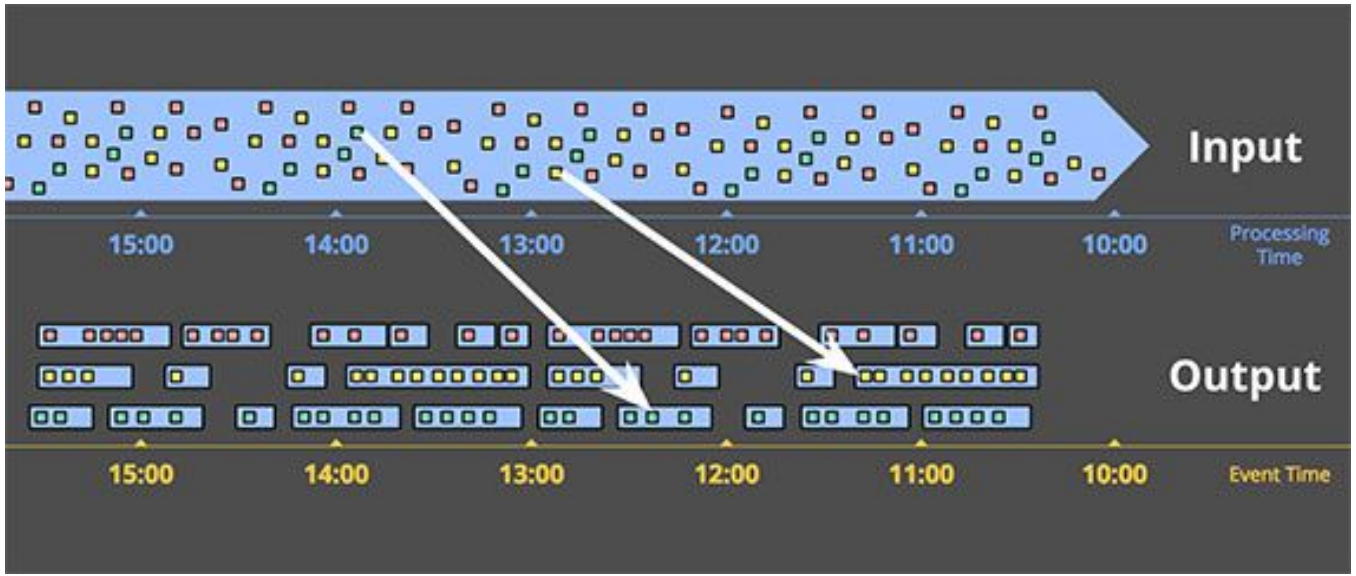
Credit:
Tyler Akidau.

Session window

Sequences of events terminated by a gap of inactivity greater than some timeout



Window into session by event time

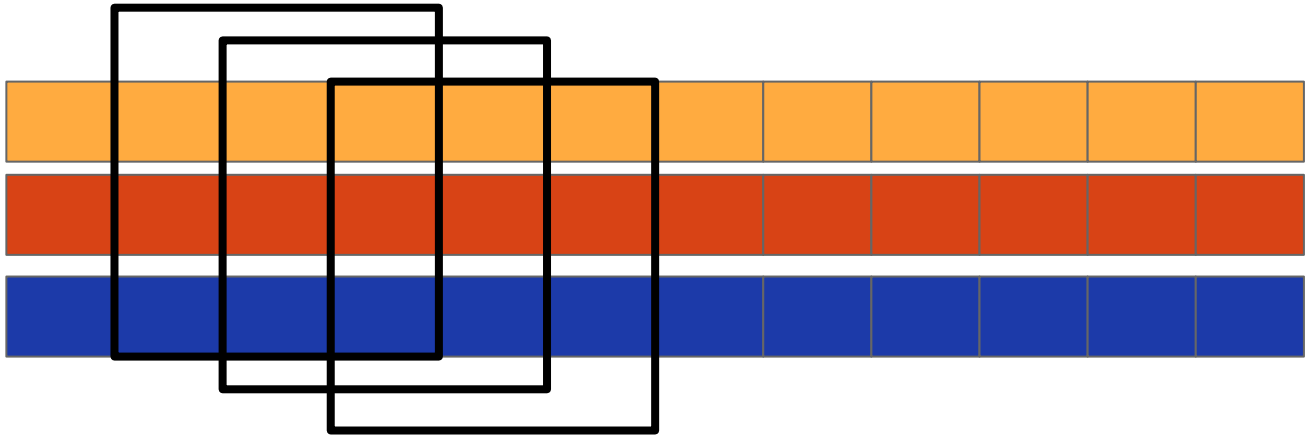


Credit:
Tyler Akidau.

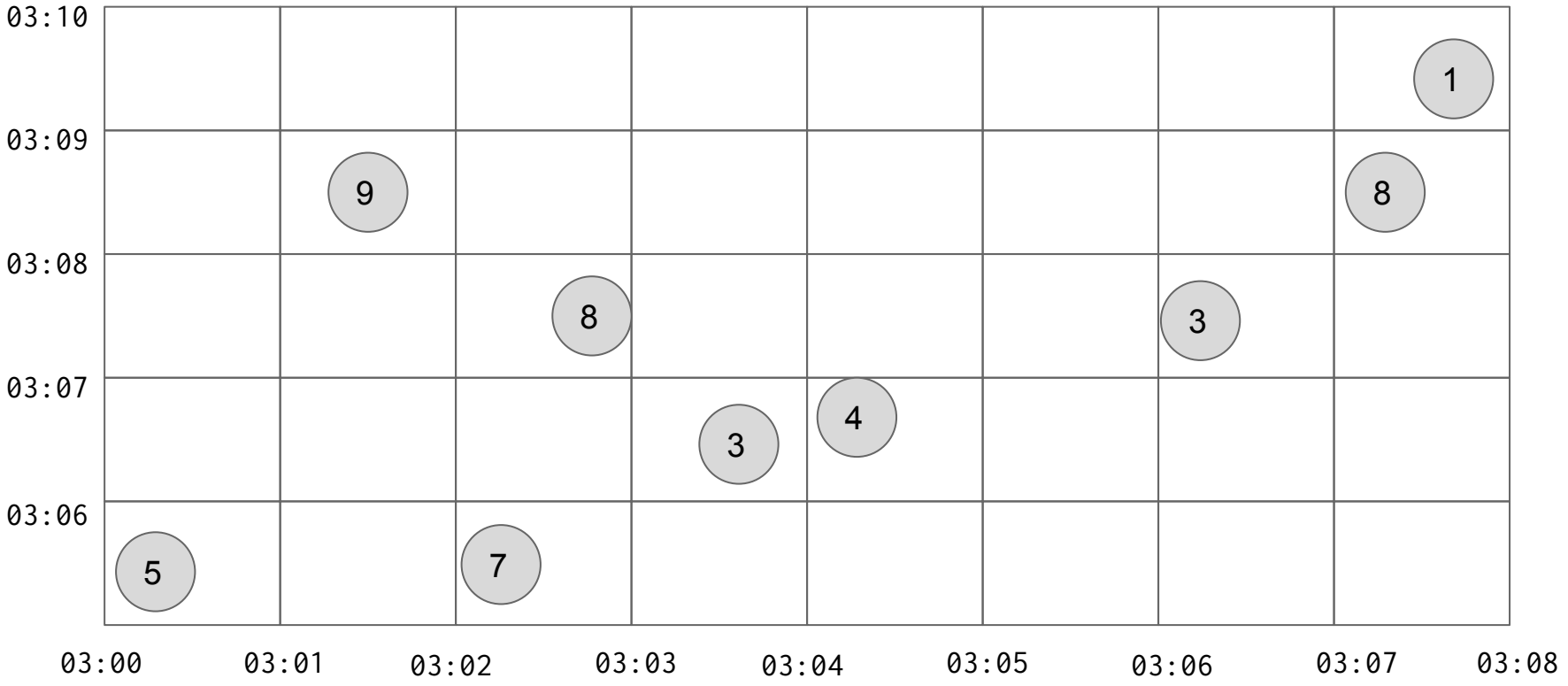
Sliding window

Fixed length and fixed period

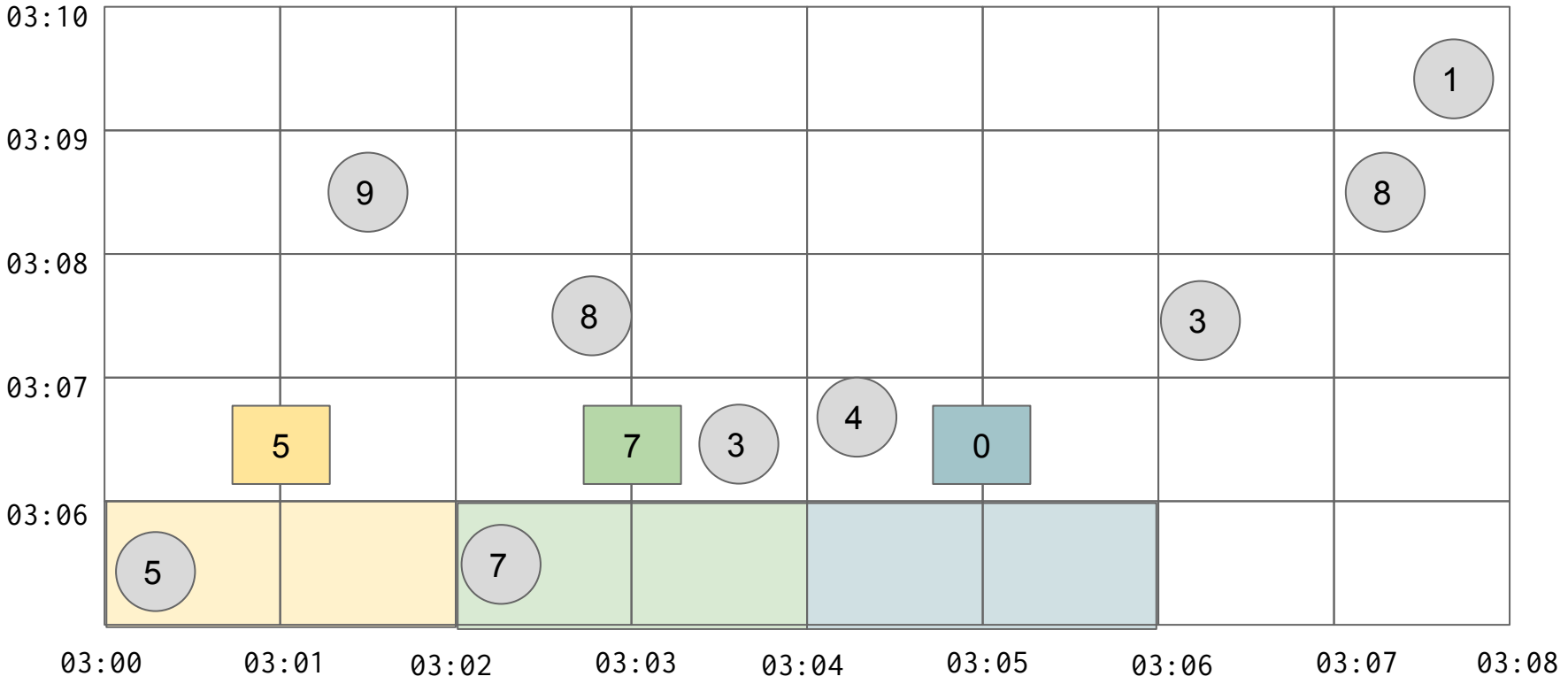
Eg, the last 10 seconds reviewed every 2 seconds



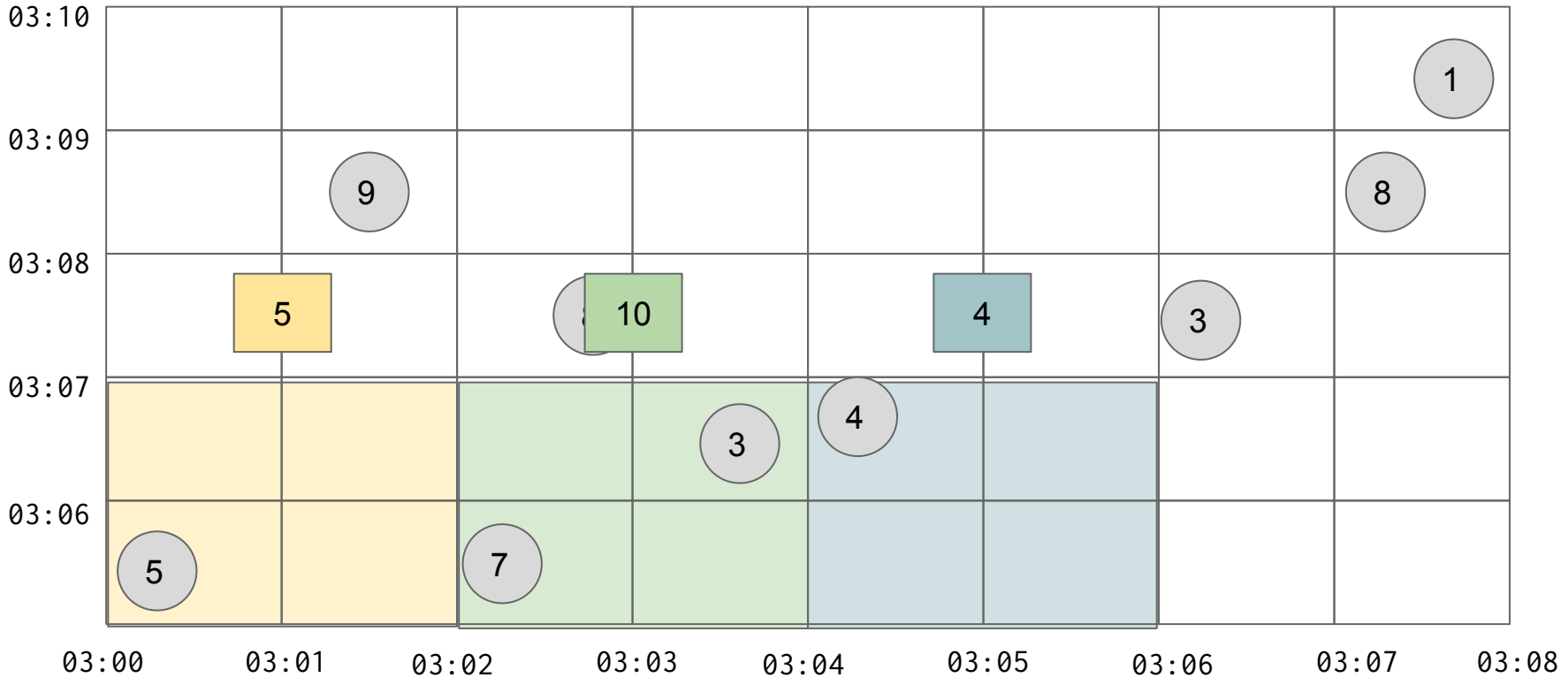
Process vs event time



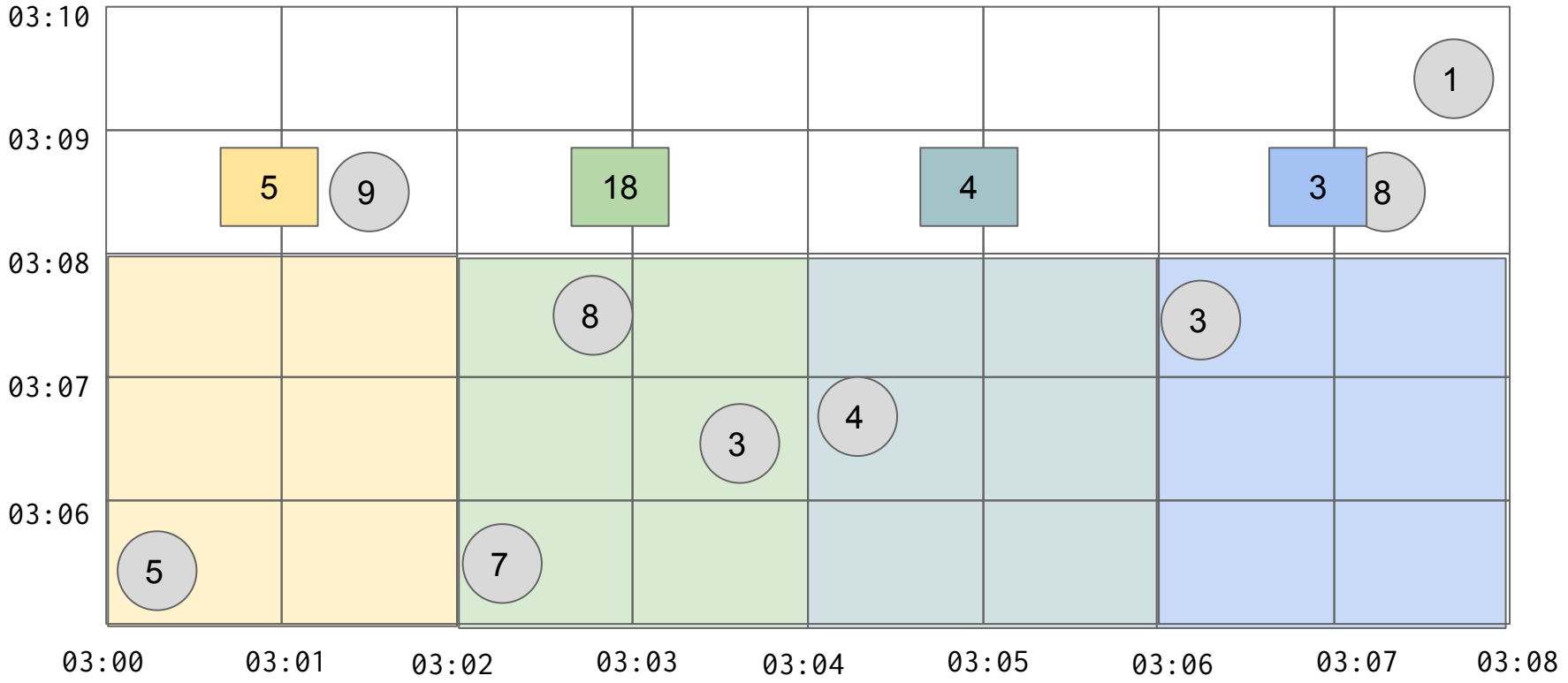
03:06 process time



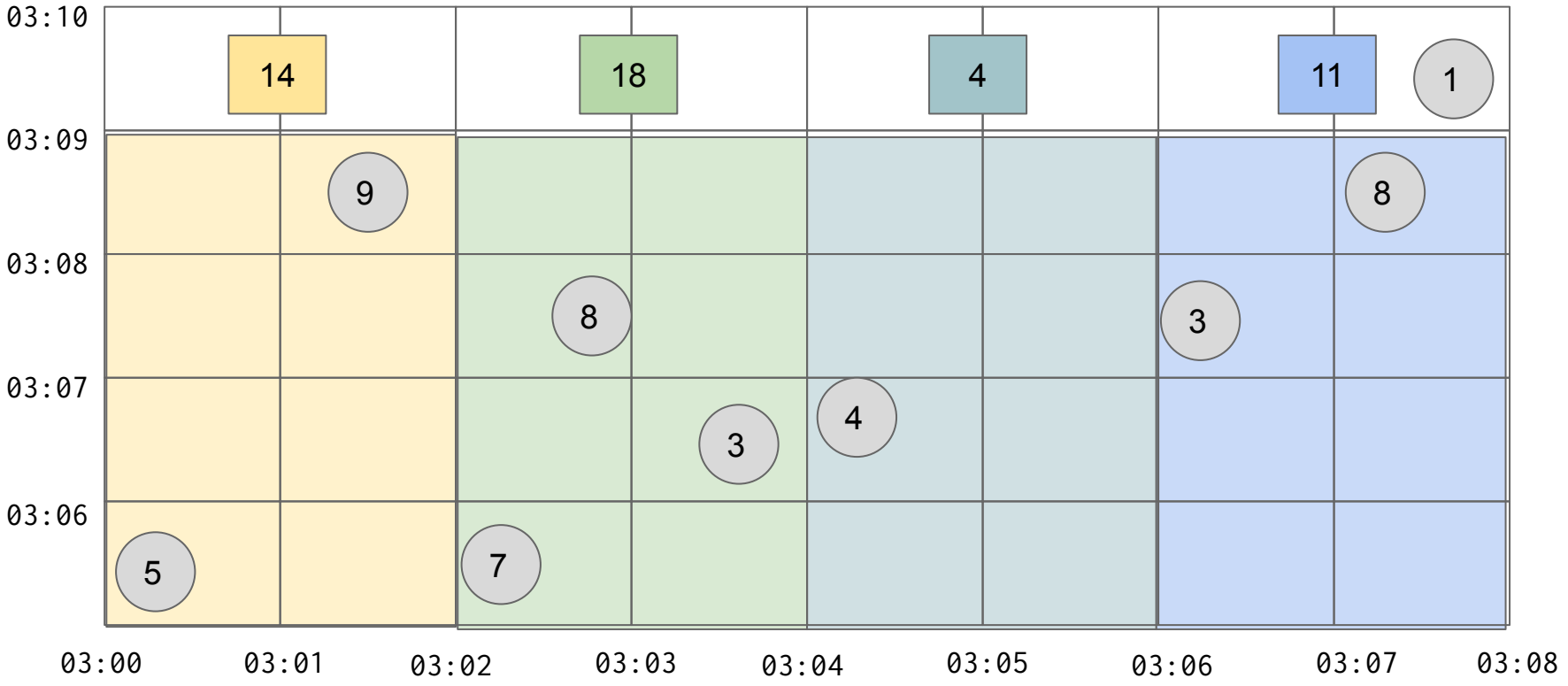
03:07


03:10
03:09
03:08
03:07
03:06
03:00
03:01
03:02
03:03
03:04
03:05
03:06
03:07
03:08
1
9
8
5
10
4
3
4
3
5
7

03:08


03:10
03:09
03:08
03:07
03:06
03:00
03:01
03:02
03:03
03:04
03:05
03:06
03:07
03:08
5
9
18
4
3
8
1
8
3
4
3
5
7

03:09


03:10
03:09
03:08
03:07
03:06
14
18
4
11
1
9
8
8
3
4
3
5
7
03:00
03:01
03:02
03:03
03:04
03:05
03:06
03:07
03:08

Watermarks

“All input data with event times less than X have been observed”

Heuristic

Best guess, never perfect

~~Crossing~~ Joining the streams

Joins

Customer data with web traffic data

Securities data with trading data

Weather data with transportation traffic data

Two or more sensors

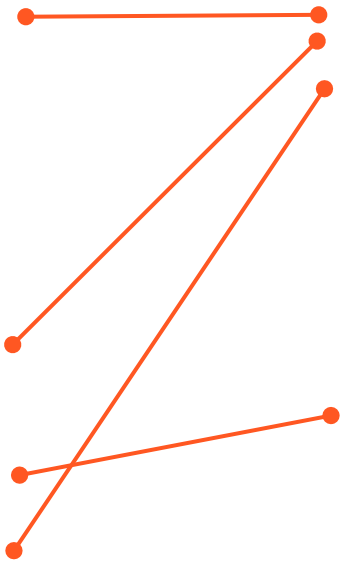
Join data

- 1000001: 12501 : 45.0
- 1000009: 12508 : 45.1
- 1000012: 12406 : 45.3
- 1000013: 12518 : 45.2
- 1000021: 12508 : 45.4
- **1000007: 12501 : 45.2**
- 1000026: 12409 : 45.5
- 1000029: 12402 : 46.0
- 1000035: 12502 : 46.4
- ...

- 1000004: 12501 : S.L.RW
- 1000005: 12502 : ..FW
- 1000005: 12503 : S.L.RW
- 1000018: 12504 : S.L.RW
- 1000021: 12505 : S.L.RW
- 1000021: 12401 : ..FW
- 1000022: 12402 : M..G
- 1000025: 12403 : M..G
- 1000027: 12404 : S.L.G
- ...

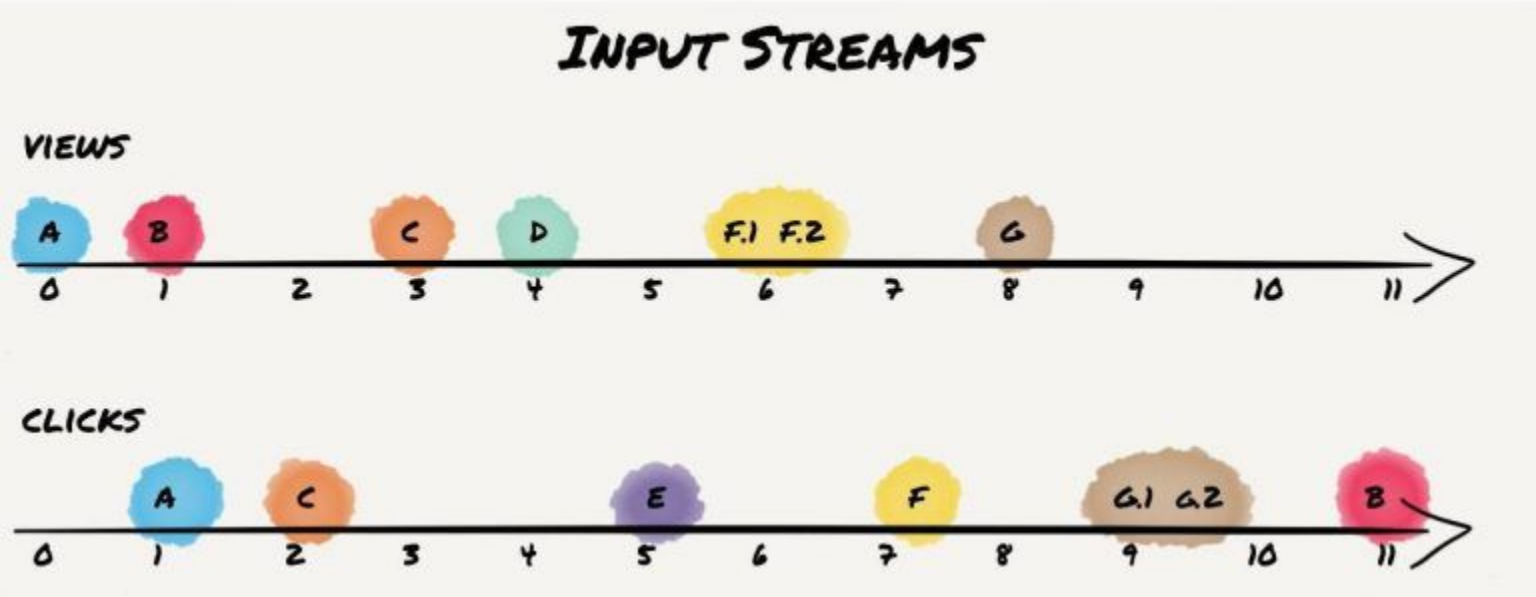
Join data

- 1000001: 12501 : 45.0
- 1000009: 12508 : 45.1
- 1000012: 12406 : 45.3
- 1000013: 12518 : 45.2
- 1000021: 12508 : 45.4
- **1000007: 12501 : 45.2**
- 1000026: 12409 : 45.5
- 1000029: 12402 : 46.0
- 1000035: 12502 : 46.4
- ...



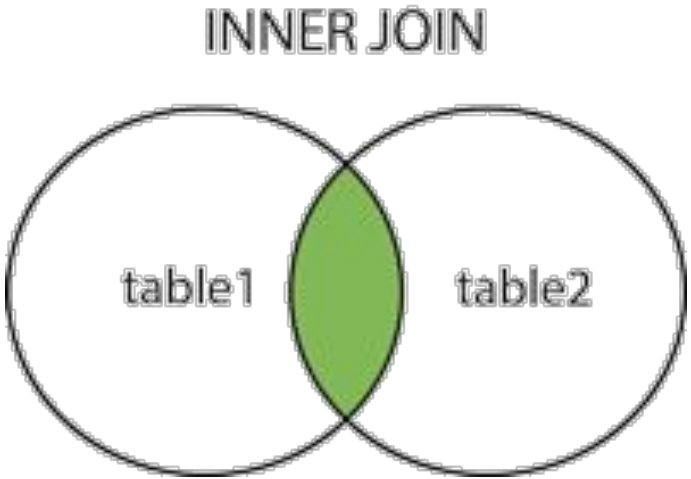
- 1000004: 12501 : S.L.RW
- 1000005: 12502 : ..FW
- 1000005: 12503 : S.L.RW
- 1000018: 12504 : S.L.RW
- 1000021: 12505 : S.L.RW
- 1000021: 12401 : ..FW
- 1000022: 12402 : M..G
- 1000025: 12403 : M..G
- 1000027: 12404 : S.L.G
- ...

Input stream example

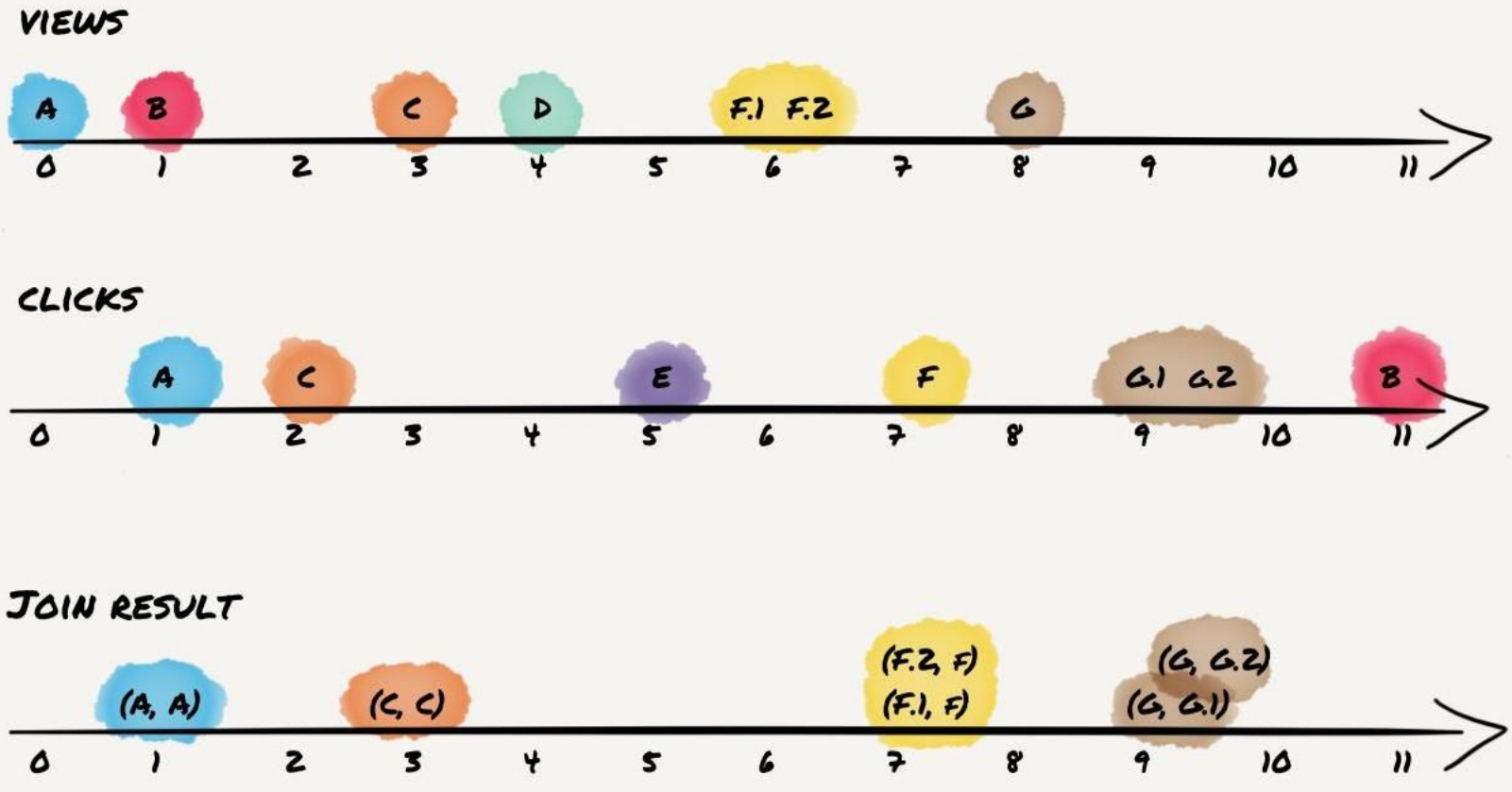


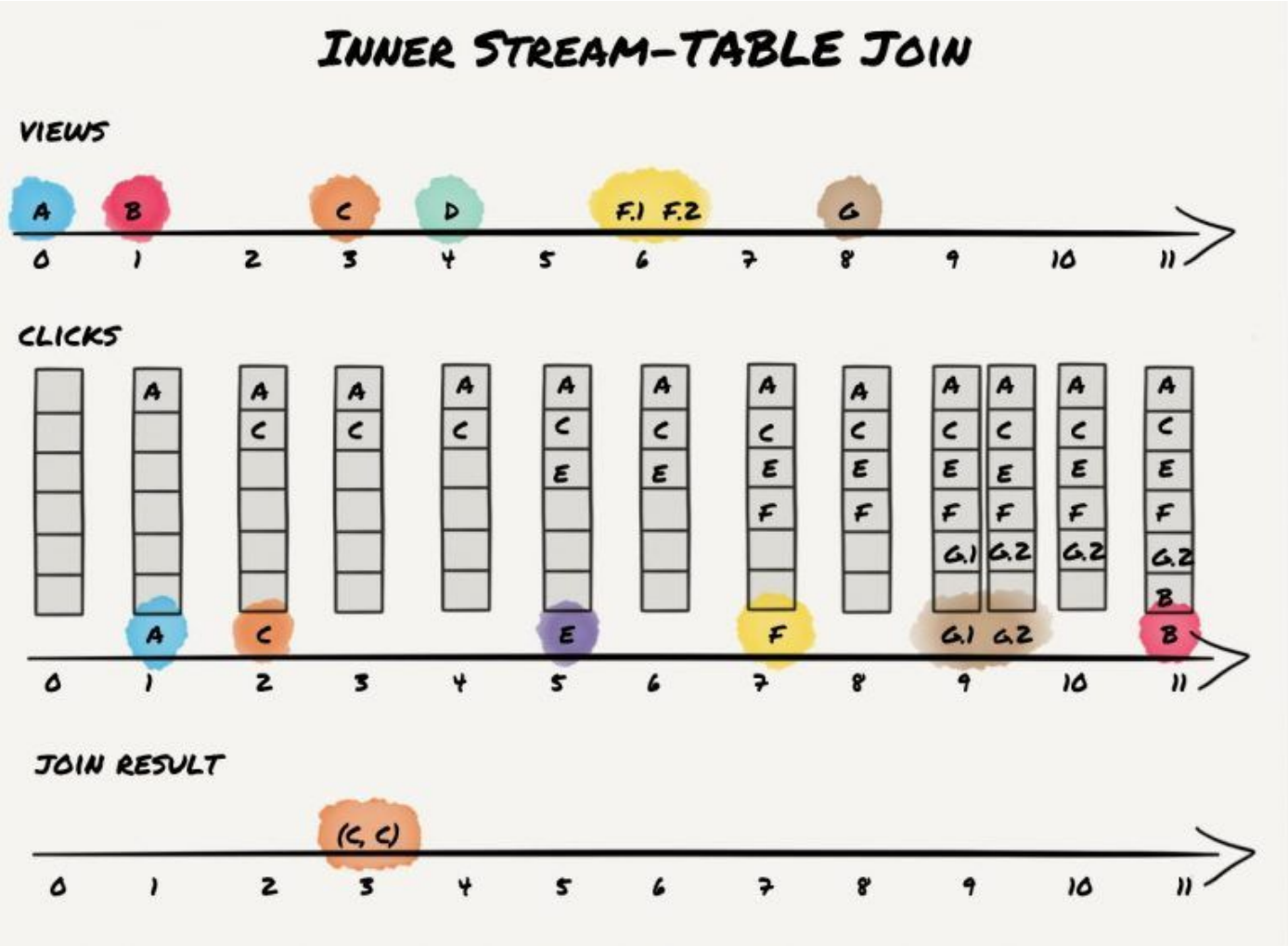
Inner join

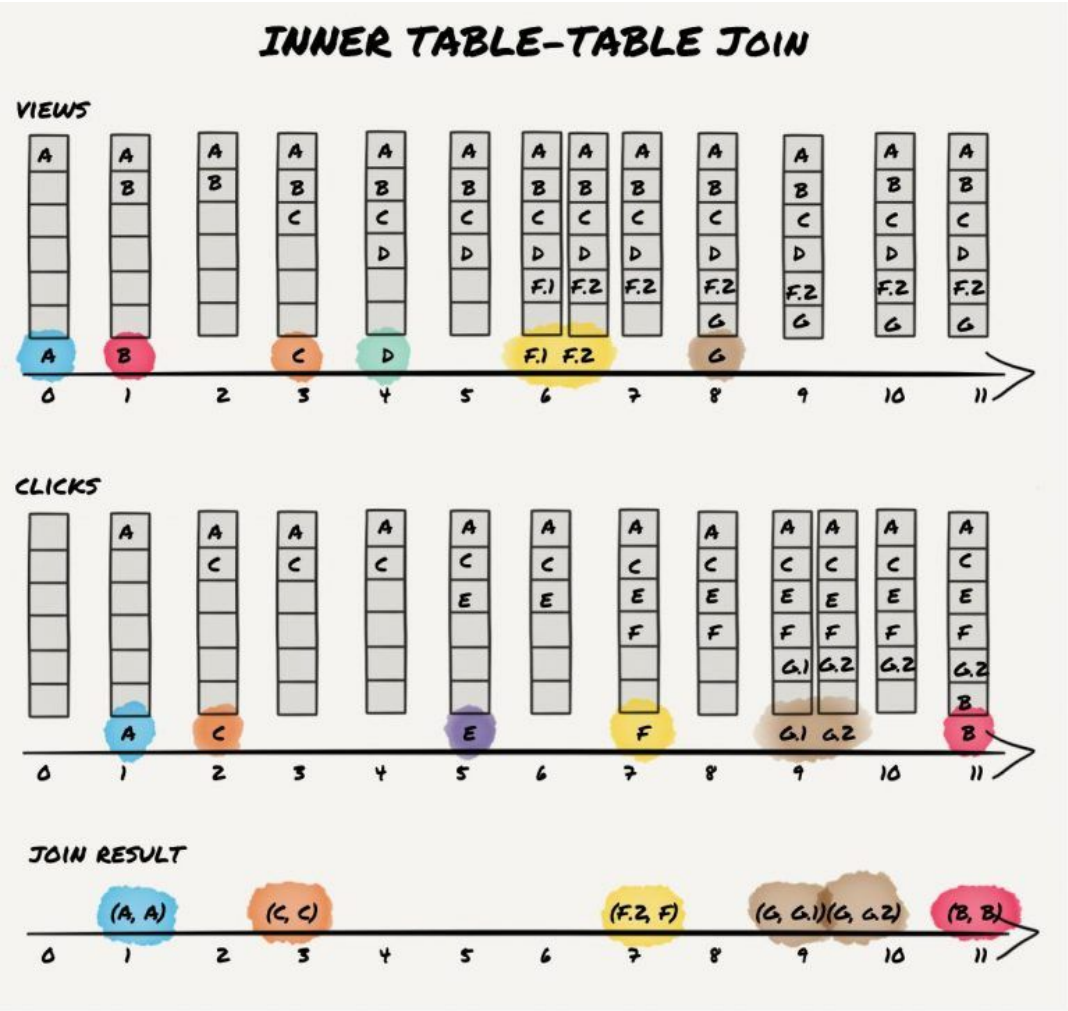
Emits an output when both input sources have records with the same key.



Inner Stream-Stream Join

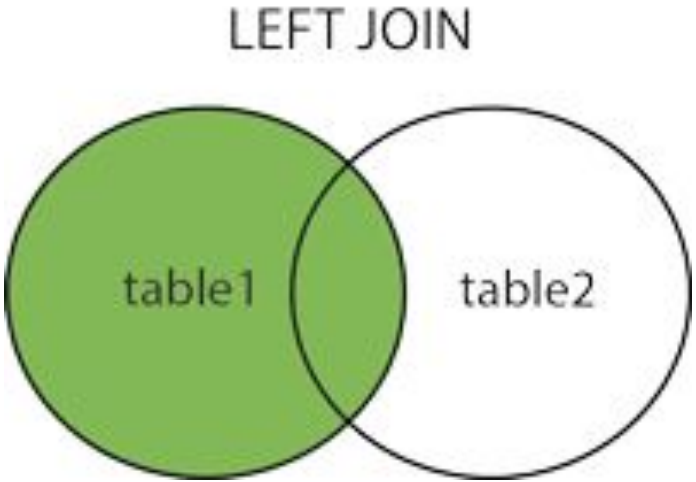


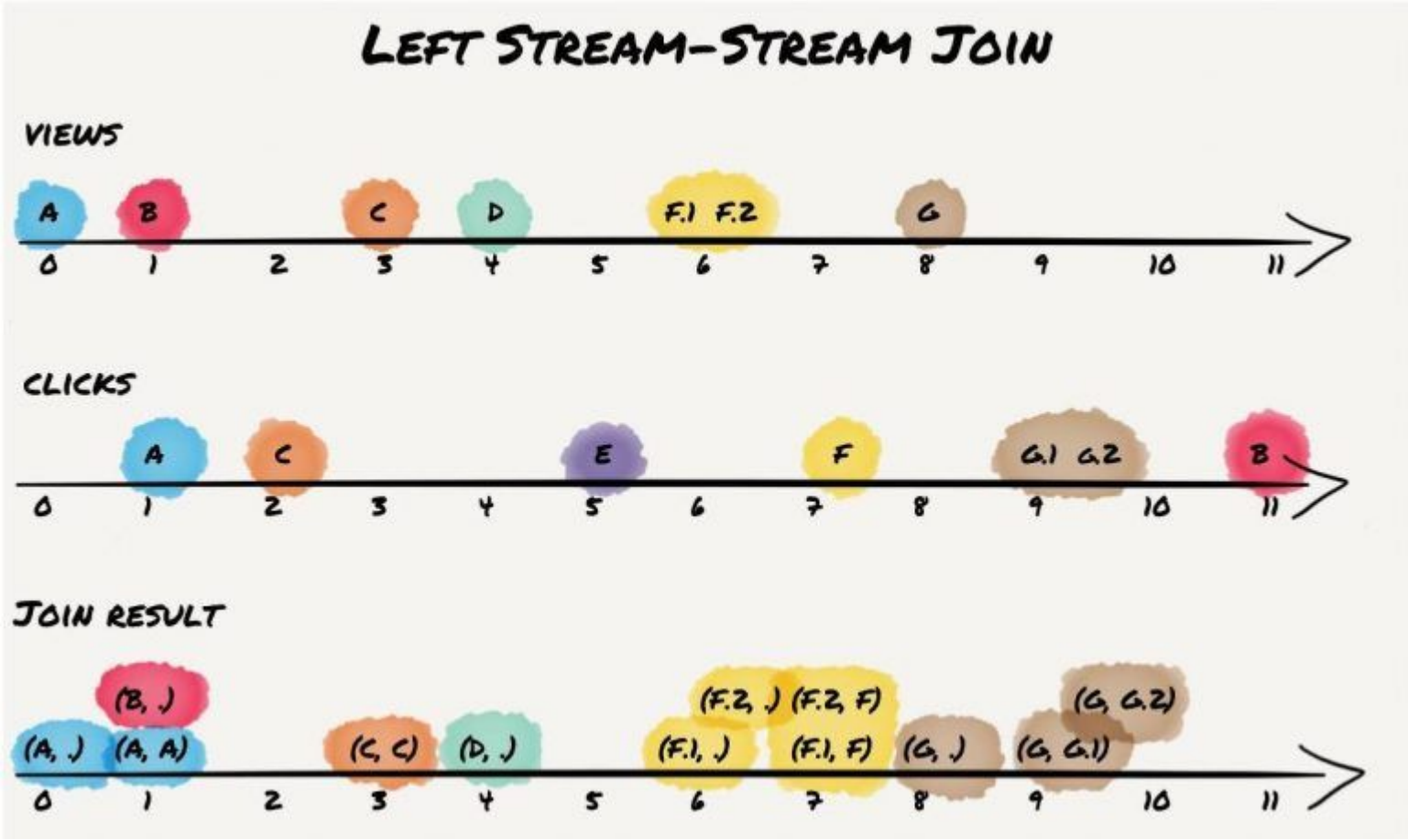

INNER STREAM-TABLE JOIN
VIEWS
A
B
C
D
F.1 F.2
G
0 1 2 3 4 5 6 7 8 9 10 11
CLICKS
A
A C
A C E
A C E F
A C E F G.1
A C E F G.2
A C E F G.2 B
A C
E
F
G.1 G.2
B
0 1 2 3 4 5 6 7 8 9 10 11
JOIN RESULT
(C, C)
0 1 2 3 4 5 6 7 8 9 10 11


INNER TABLE-TABLE JOIN
VIEWS
CLICKS
JOIN RESULT
(A, A)
(C, C)
(F.2, F)
(G, G.1)(G, G.2)
(B, B)
0 1 2 3 4 5 6 7 8 9 10 11

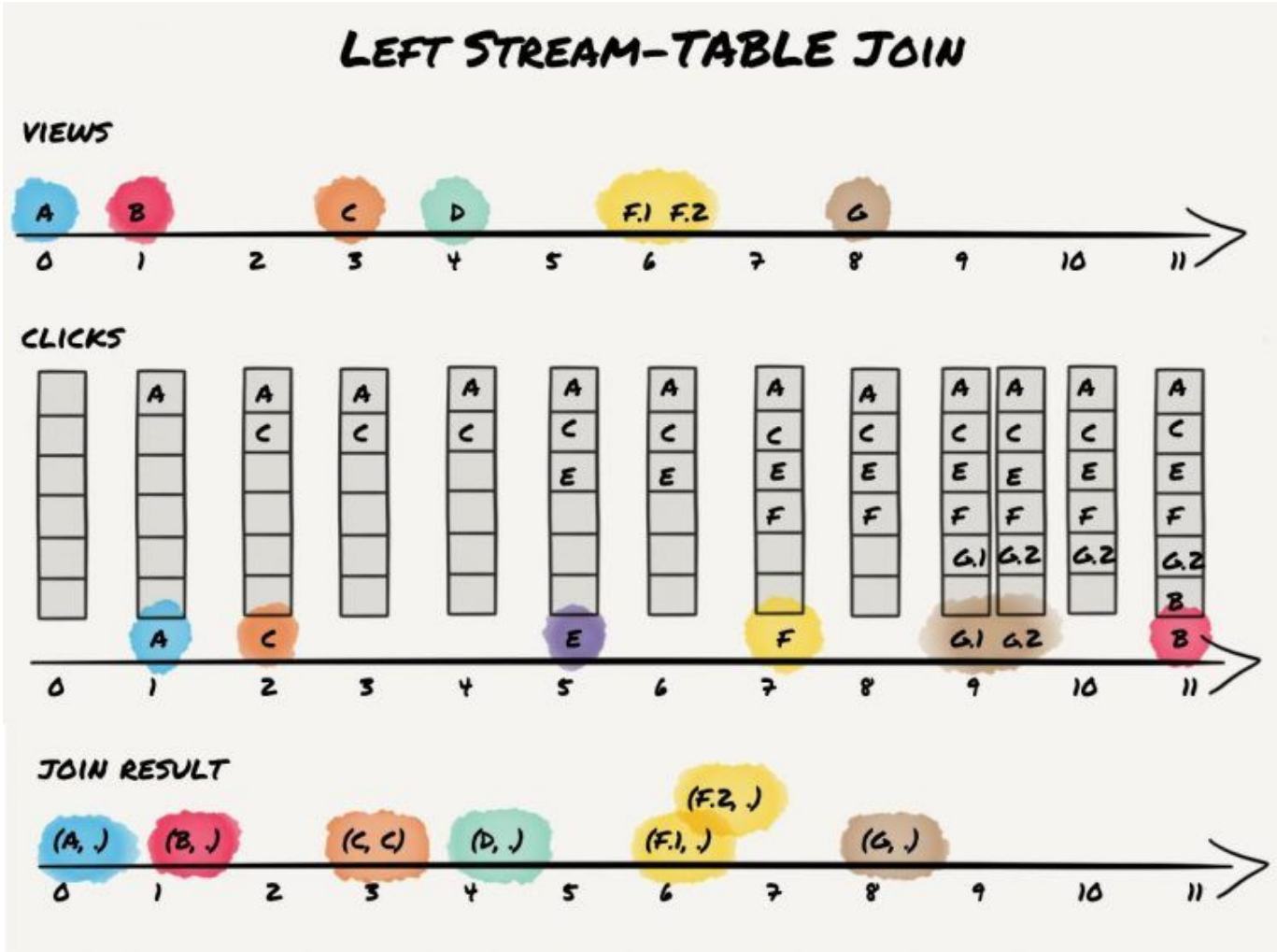
Left join

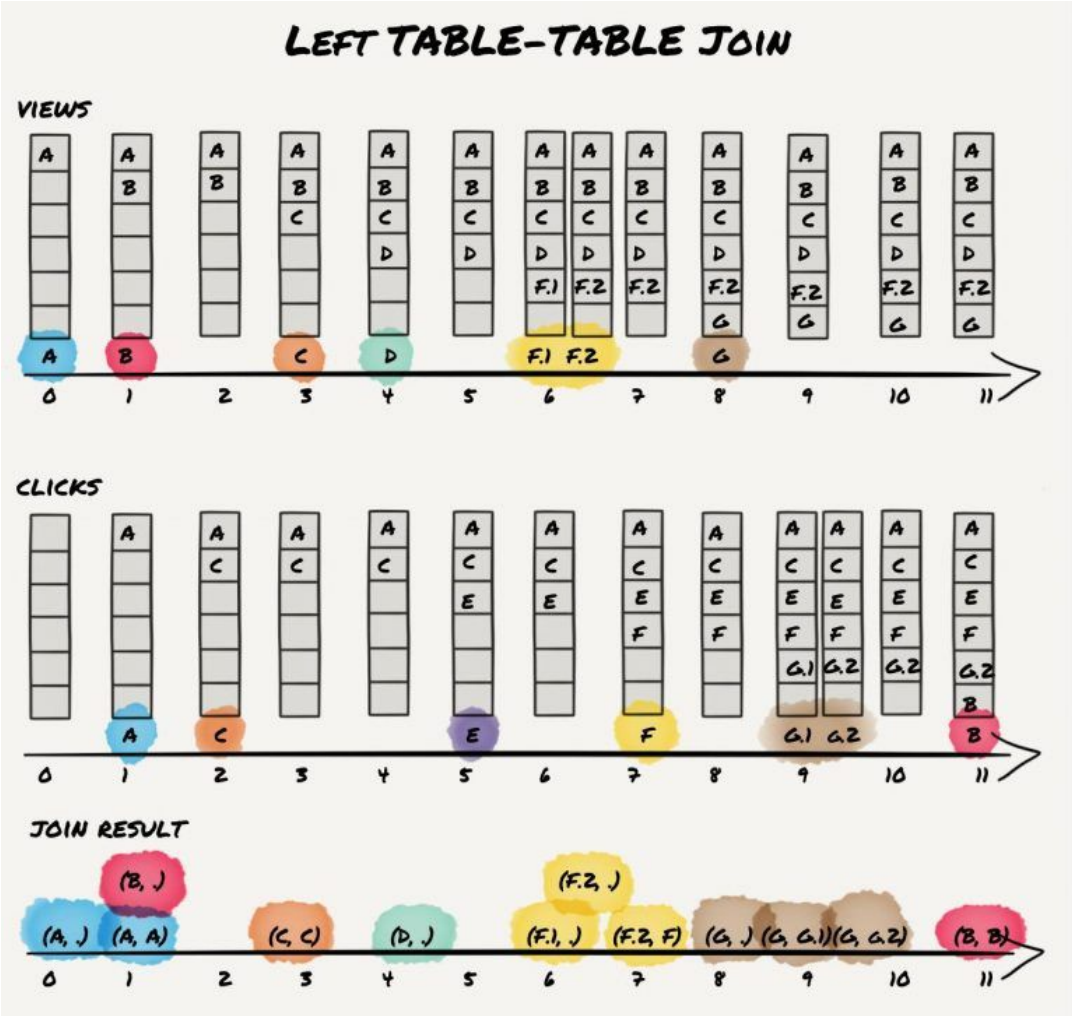
Emits an output for each record in the left or primary input source. If the other source does not have a value for a given key, it is set to null.




LEFT STREAM-STREAM JOIN
VIEWS
A
B
C
D
F.1 F.2
G
0 1 2 3 4 5 6 7 8 9 10 11
CLICKS
A
C
E
F
G.1 G.2
B
0 1 2 3 4 5 6 7 8 9 10 11
JOIN RESULT
(B, .)
(A, .)
(A, A)
(C, C)
(D, .)
(F.2, .)
(F.2, F)
(F.1, .)
(F.1, F)
(G, .)
(G, G.2)
(G, G.1)
0 1 2 3 4 5 6 7 8 9 10 11

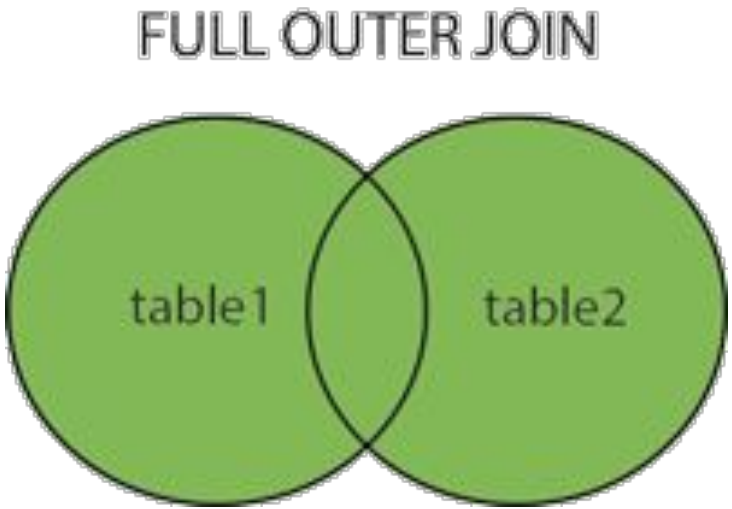
Left Stream-Table Join

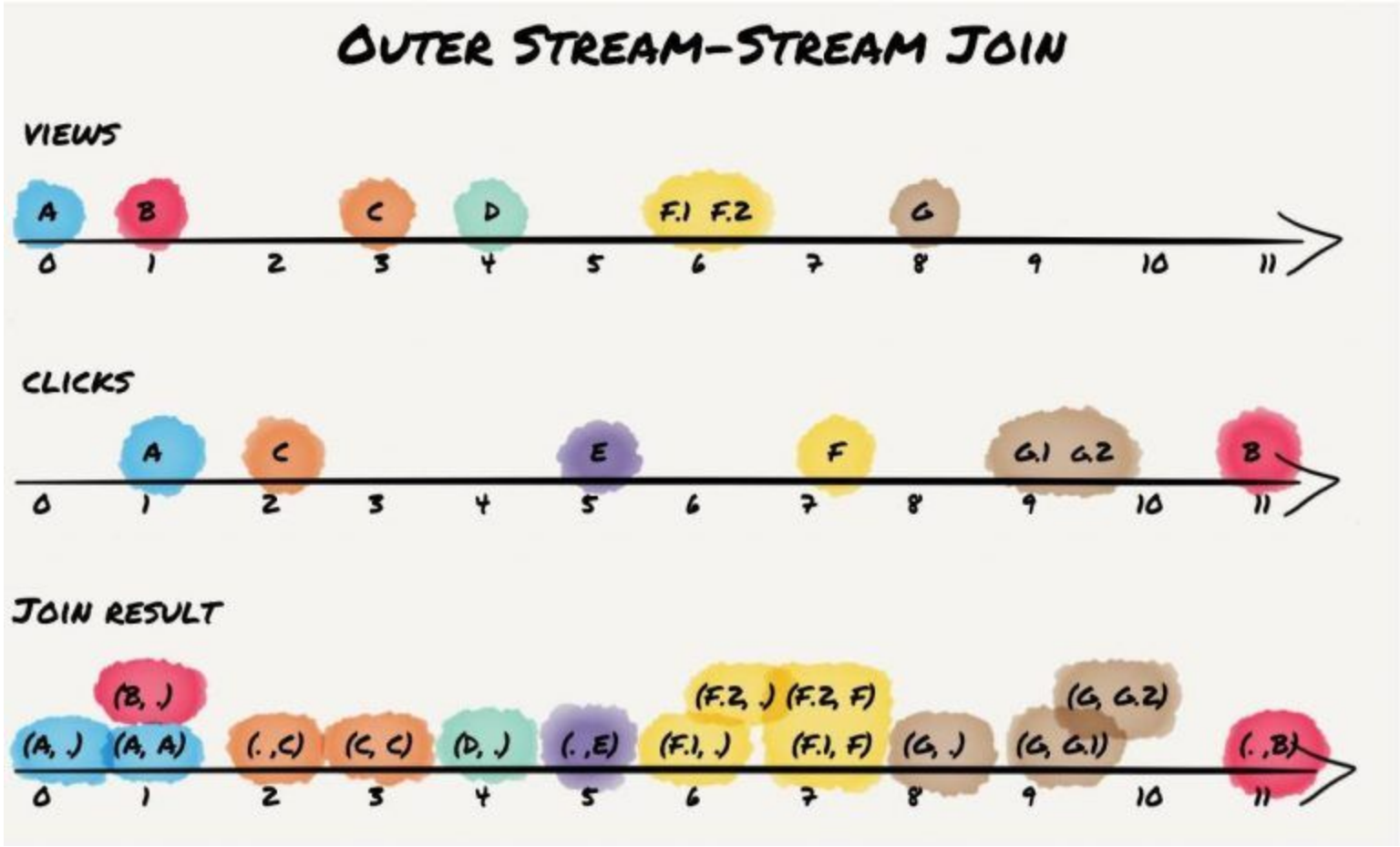



LEFT TABLE-TABLE JOIN
VIEWS
CLICKS
JOIN RESULT
(A, ,)
(A, A)
(B, ,)
(C, C)
(D, ,)
(F.1, ,)
(F.2, ,)
(F.2, F)
(G, ,)
(G, G.1)
(G, G.2)
(B, B)
0
1
2
3
4
5
6
7
8
9
10
11

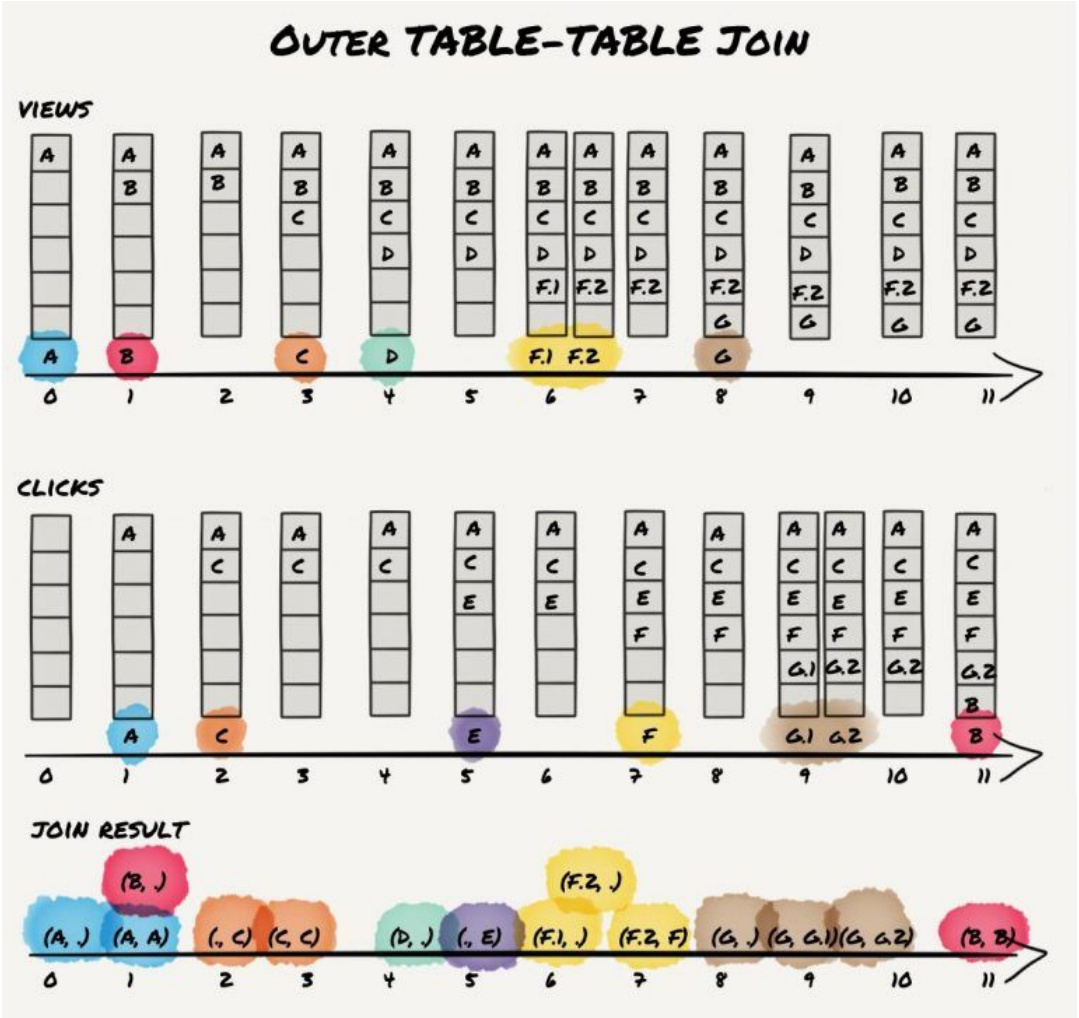
Outer join

Emits an output for each record in either input source. If only one source contains a key, the other is null.



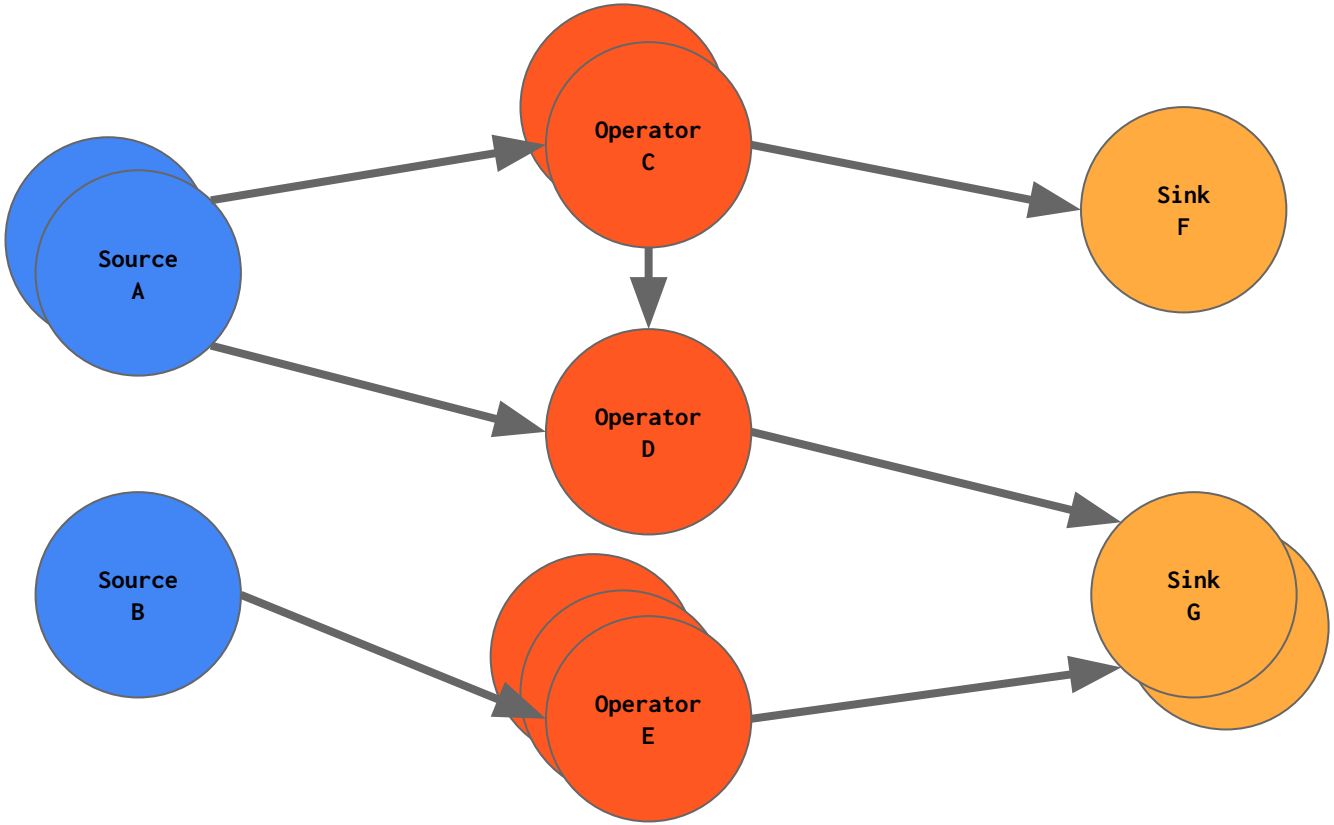

OUTER STREAM-STREAM JOIN
VIEWS
A
B
C
D
F.1 F.2
G
0 1 2 3 4 5 6 7 8 9 10 11
CLICKS
A
C
E
F
G.1 G.2
B
0 1 2 3 4 5 6 7 8 9 10 11
JOIN RESULT
(B, .)
(A, .)
(A, A)
(., C)
(C, C)
(D, .)
(., E)
(F.2, .)
(F.2, F)
(F.1, .)
(F.1, F)
(G, .)
(G, G.2)
(G, G.1)
(., B)
0 1 2 3 4 5 6 7 8 9 10 11

OUTER TABLE-TABLE JOIN




Scaling

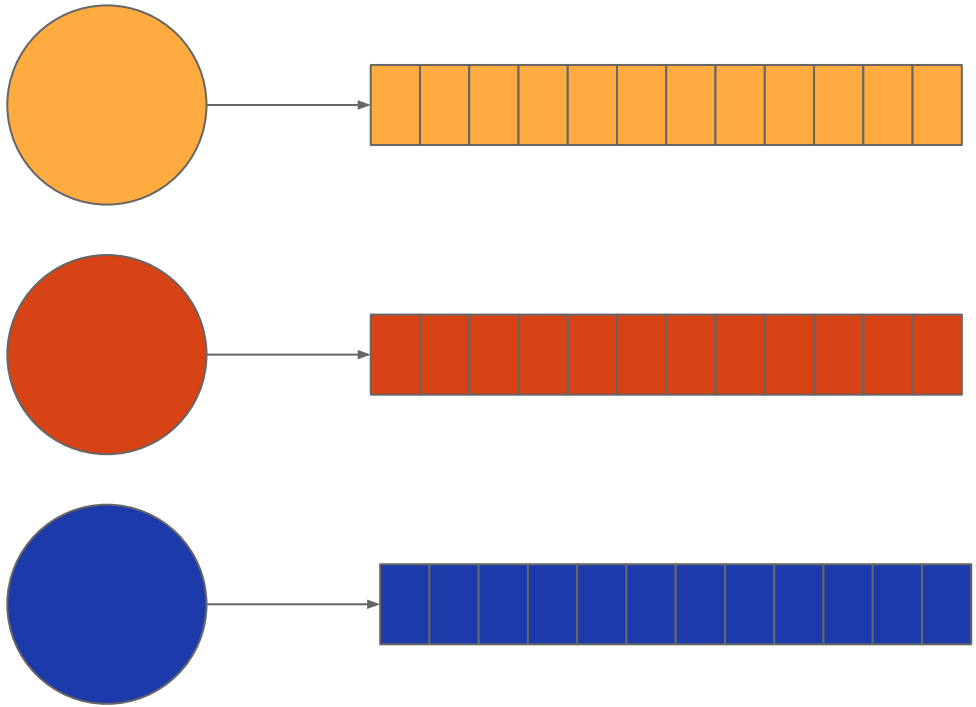
Parallelism



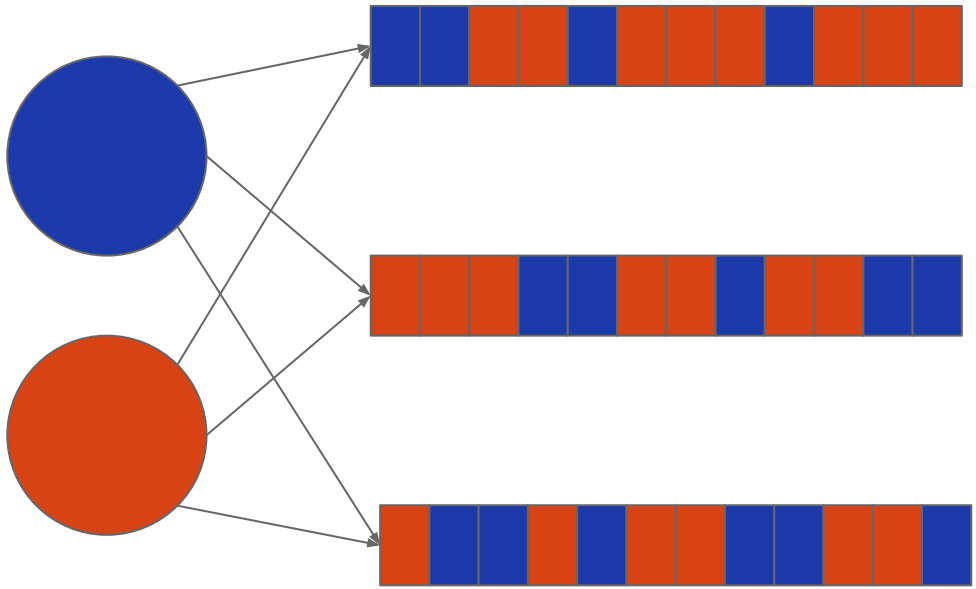
Fundamental challenge of parallel processing?

Two approaches to parallel processing?

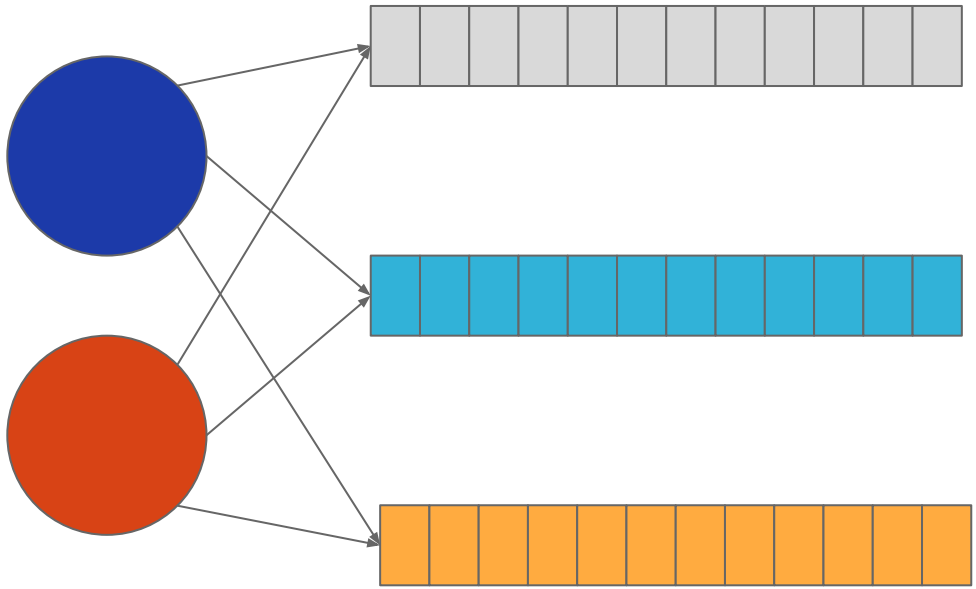
Partition by source



Partition round robin



Partition by key



Keys

Types of keys

1. **Natural key** such as a place name
2. **Generated key** (aka surrogate key)
3. **Composite key**

One man’s generated key is another man’s natural key

Never create keys with metadata!

UUID

Universally unique identifier

Sequence identifiers will NOT work in a distributed environment

128-bit

4fedefdb-4c7c-42b5-ae00-b6d286034b2c

Collision: generate 1 billion UUIDs per second for about 85 years

https://www.uuidgenerator.net/

Partition by hash-mod

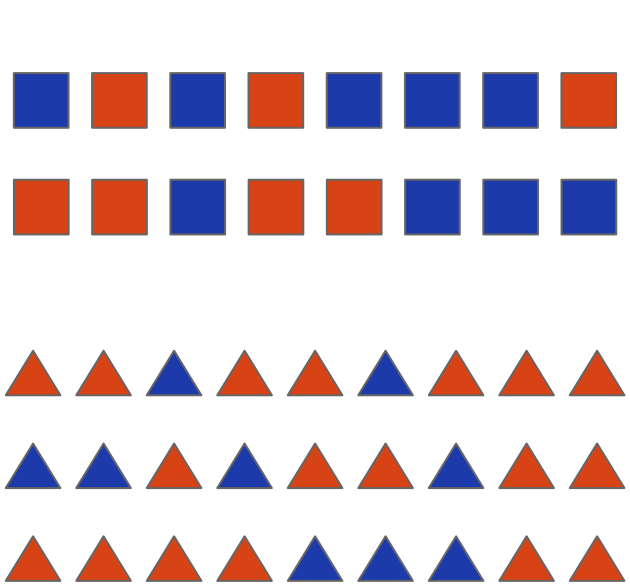
```
hash(key) % partitions
```

```
hash(“Amsterdam”) % 10

8763125 % 10

5
```

Joins on partitions

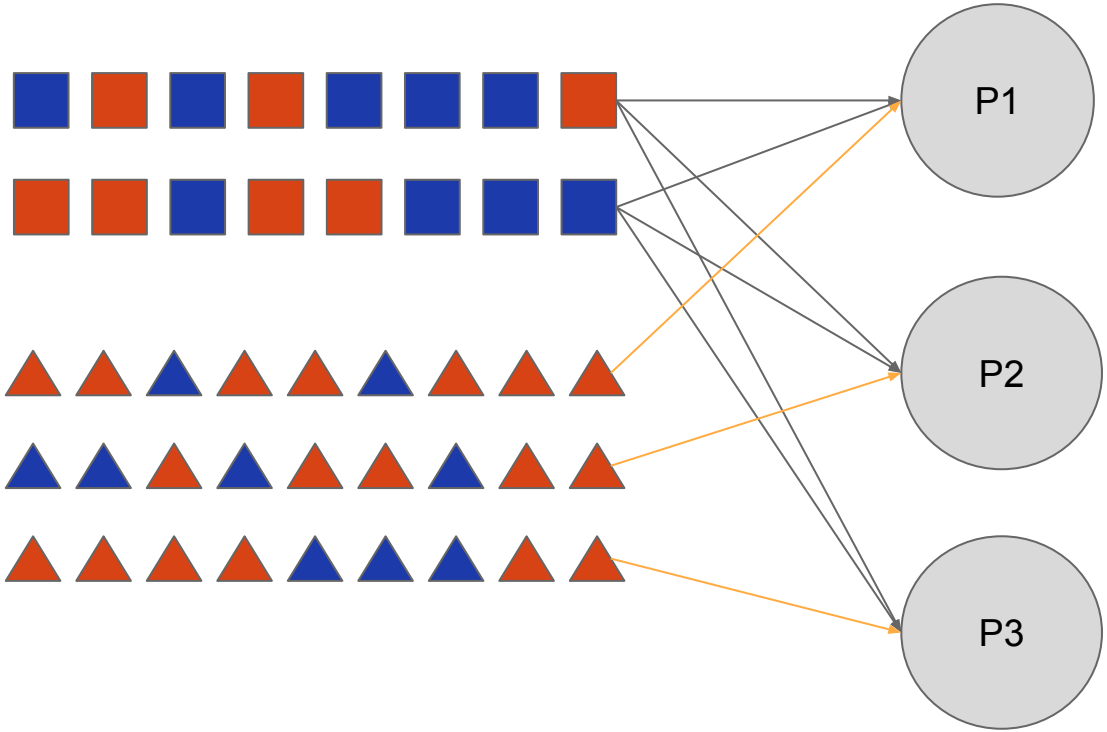


P1

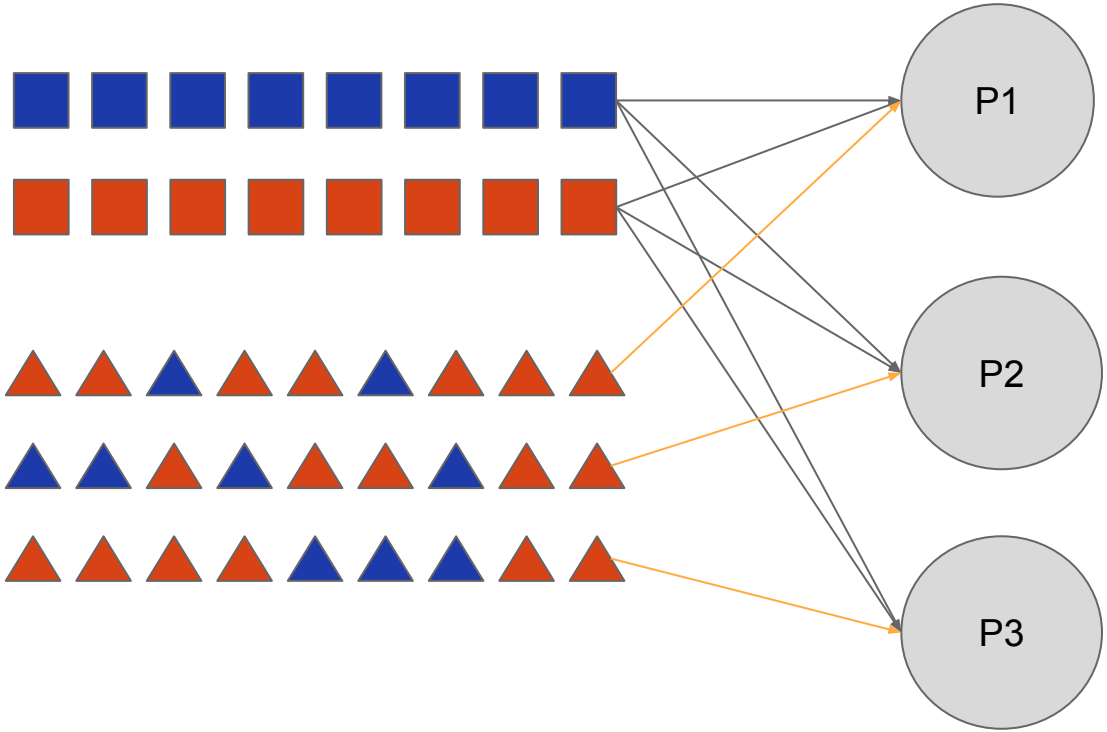
P2

P3

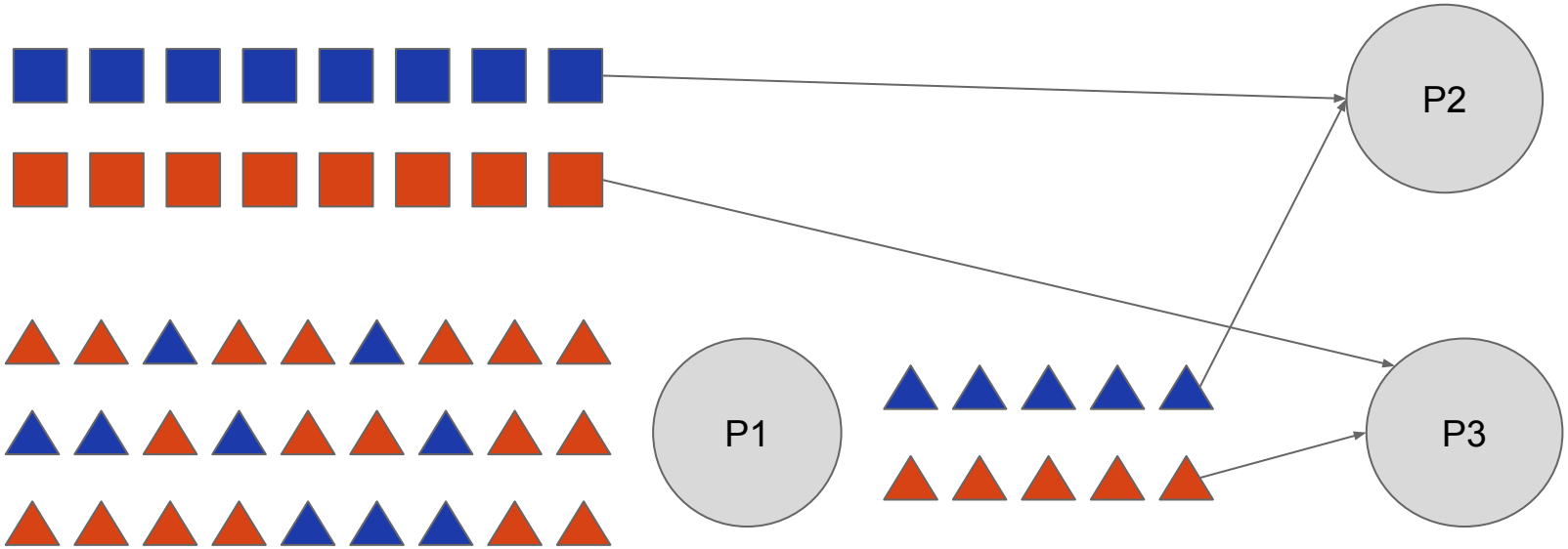
Joins on partitions



Joins on partitions



Joins on partitions



Partitions of partitions

Initial

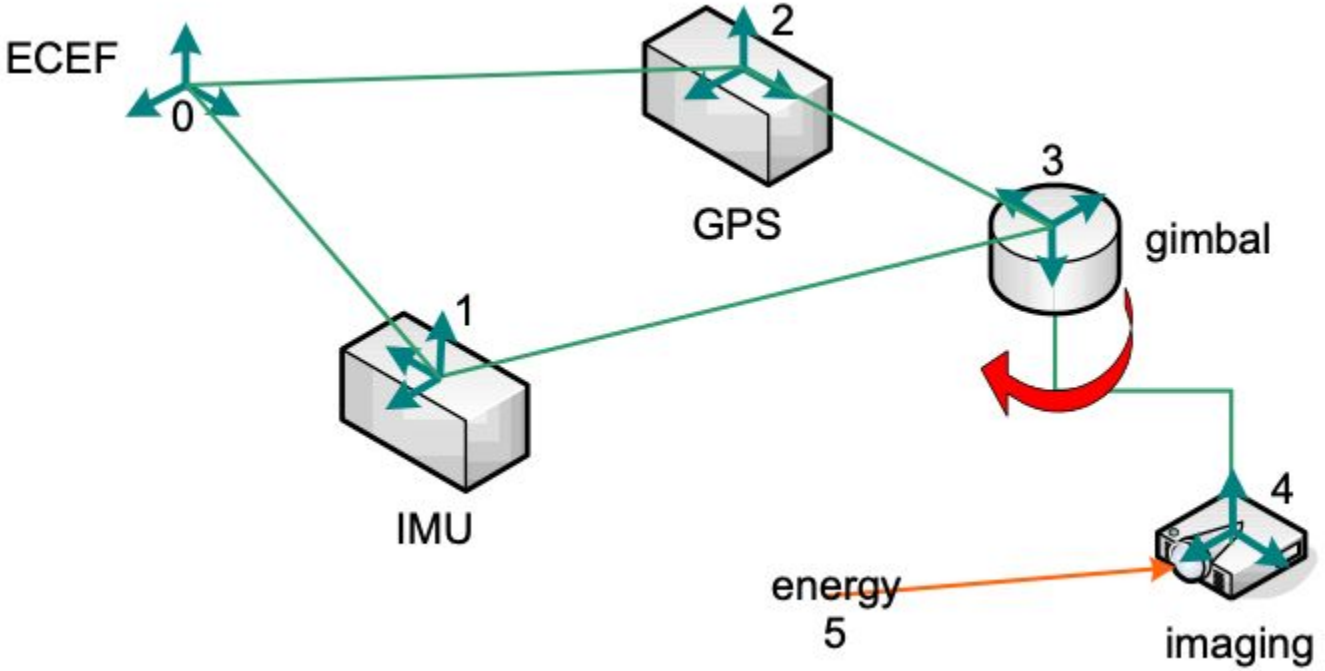
1. 0..9
2. 10..19
3. 20..29
4. 30..39

Expanded

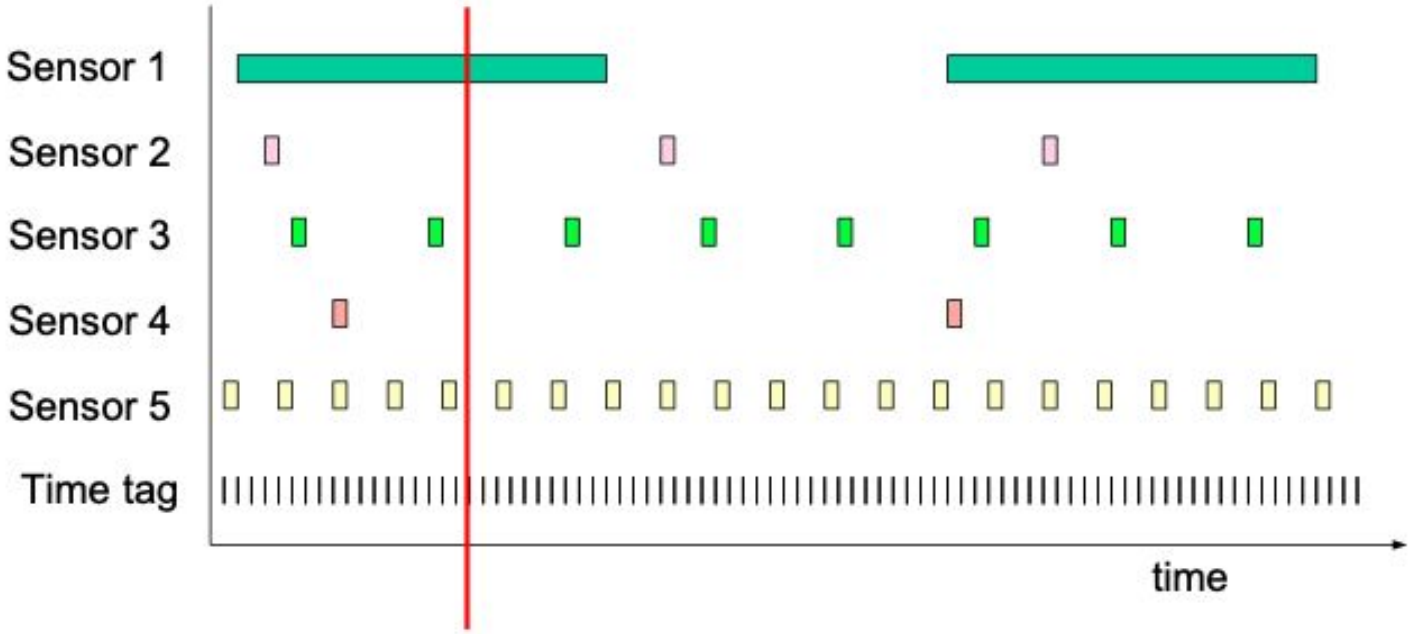
1. 0..6
2. 10..16
3. 20..26
4. 30..36
5. 7..9,17..19
6. 27..29,37..39

Real world example

System example

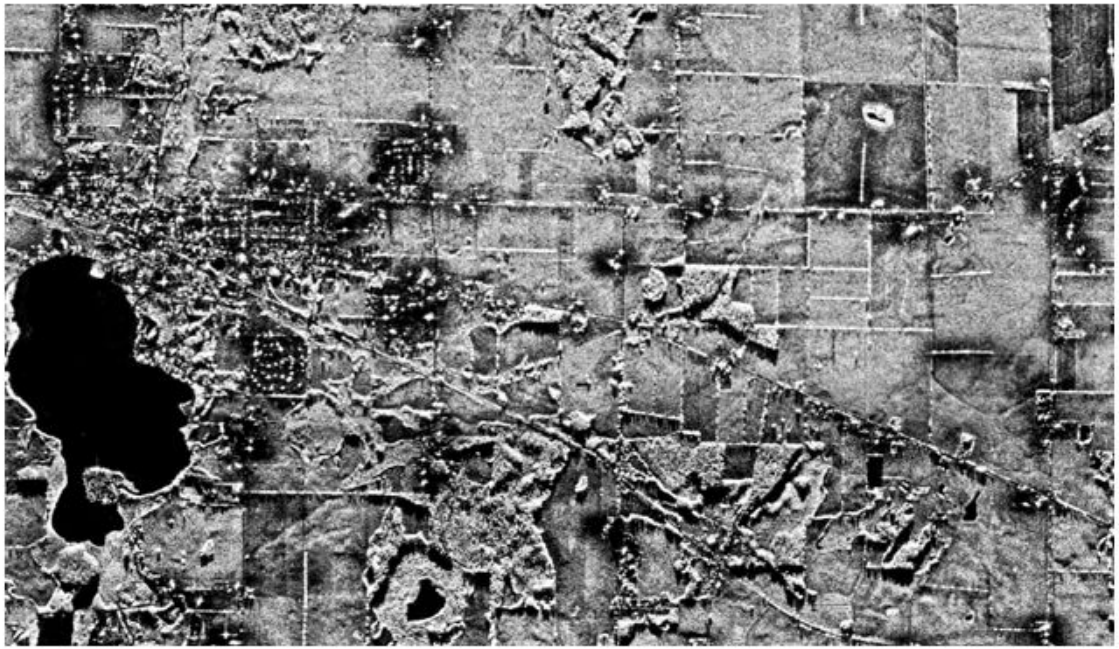


Sample rates

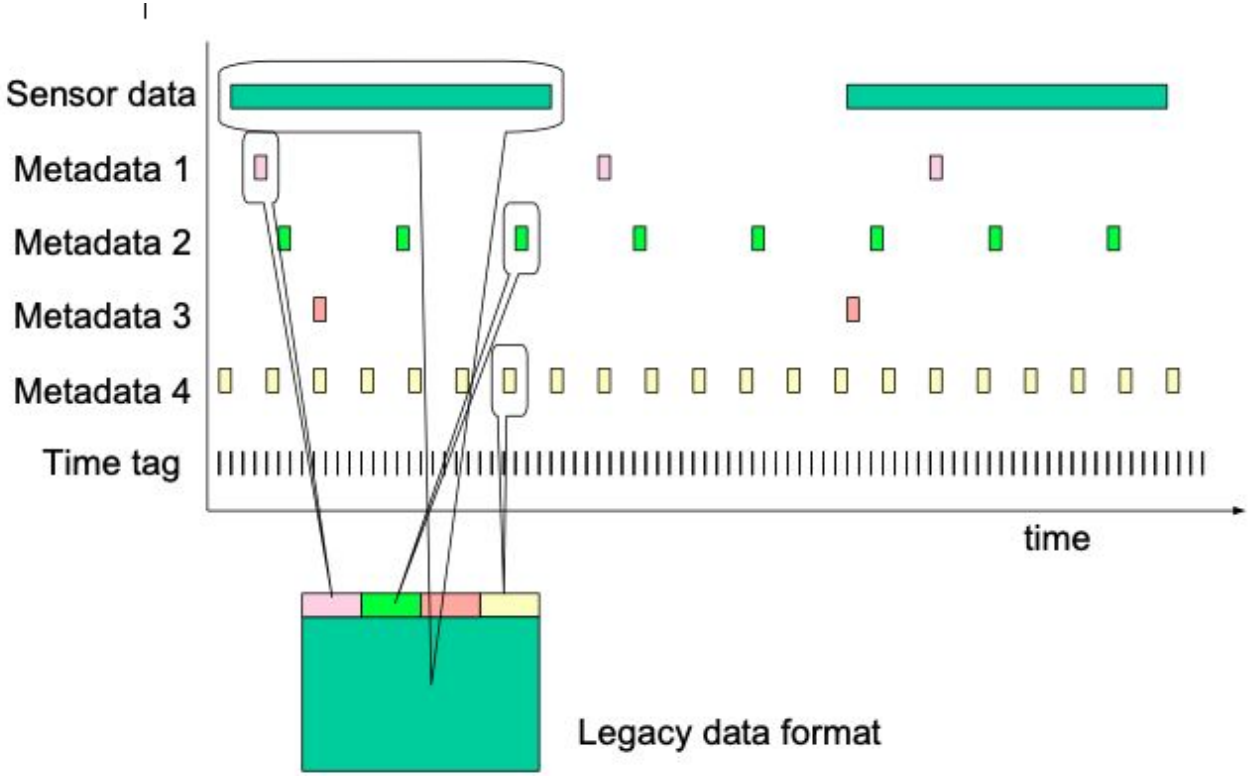


Legacy processing

Longitude	Latitude	Altitude	Pitch	Roll	Heading



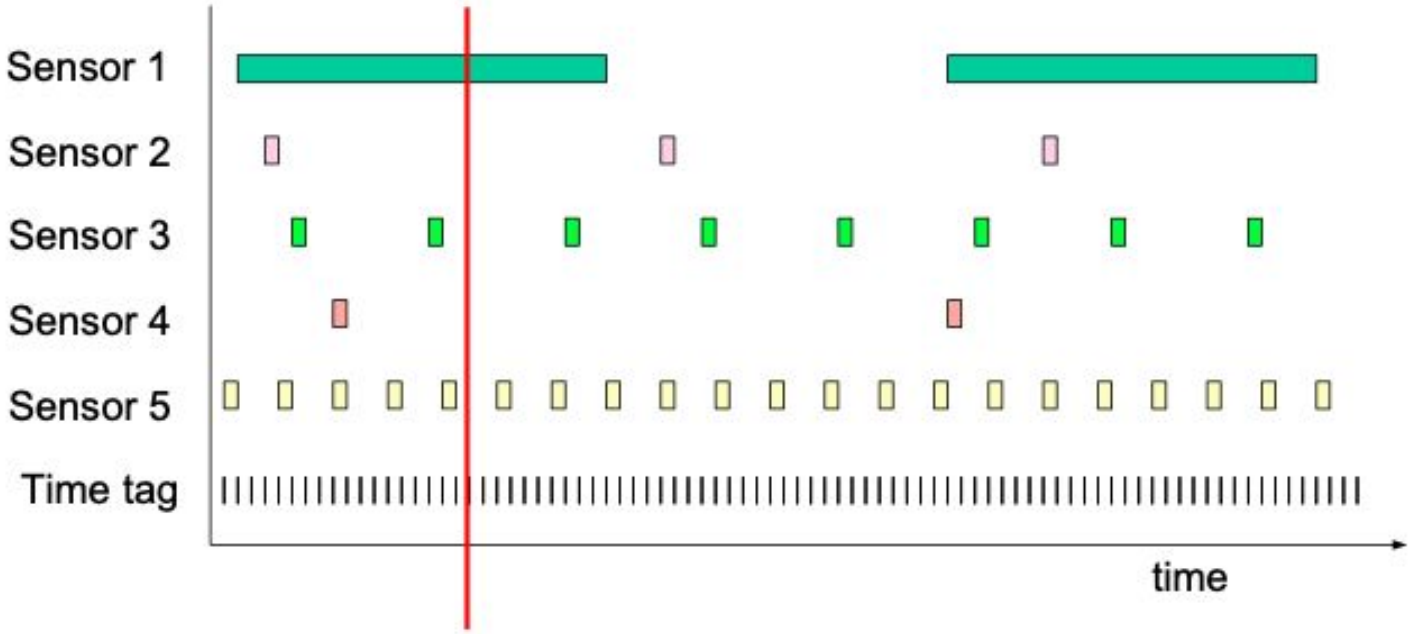
Legacy batch processing



Stream processing

Time	Sensor reading
2456	Latitude, longitude, altitude
2635	
2762	Pitch, roll, heading
2789	
2812	Latitude, longitude, altitude
2893	
2910	Pitch, roll, heading
2998	
3015	Latitude, longitude, altitude
3075	
3147	Pitch, roll, heading
3222	

Sample rates



Stream processing

$$G(2789) = G_{2762} + ((2789-2762)/(2910-2762))G_{2910}$$

Time	Sensor reading
2456	Latitude, longitude, altitude
2635	
2762	Pitch, roll, heading
2789	
2812	Latitude, longitude, altitude
2893	
2910	Pitch, roll, heading
2998	
3015	Latitude, longitude, altitude
3075	
3147	Pitch, roll, heading
3222	

THANK YOU!

Calculate the average

- ...
- 100001: 45.0
- 100009: 45.1
- 100012: 45.3
- 100013: 45.2
- 100021: 45.4
- **100007: 45.2**
- 100026: 45.5
- 100029: 46.0
- 100035: 46.4
- ...

Process vs event time