Optimizing Big Scio Pipelines
(in Scala!)

Naheon Kim
6th March 2019

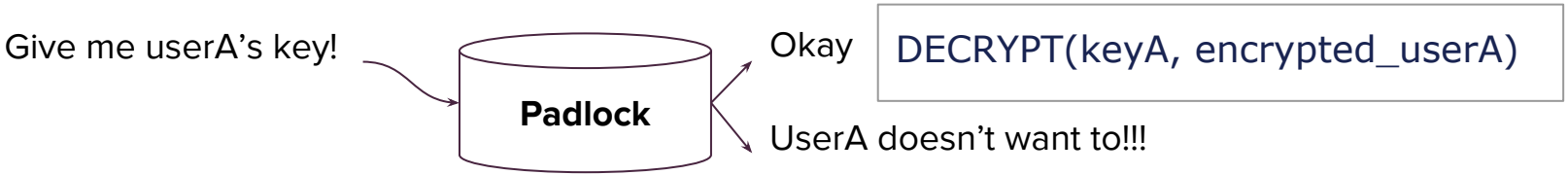
Who I am

- Data engineer at Spotify
- Was backend engineer in Korea

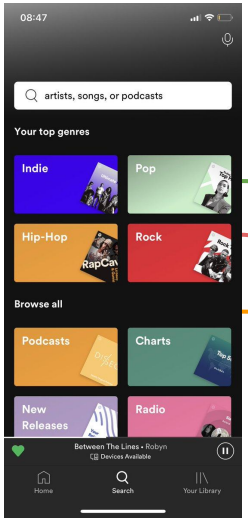
We lock privacy

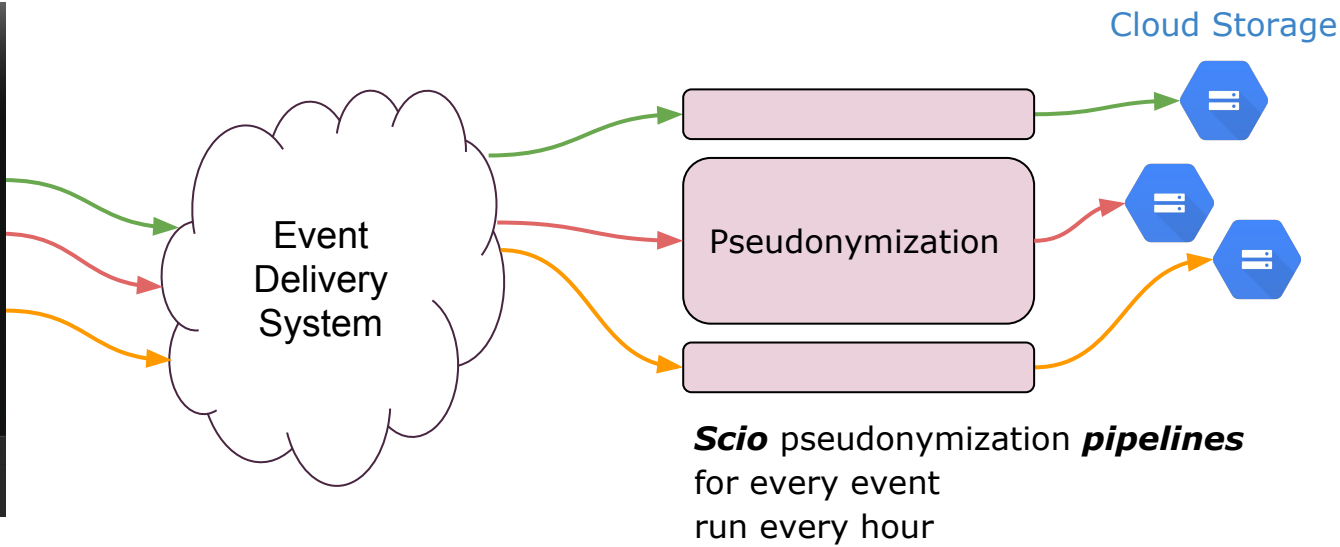
- **Padlock - Key management system**
- **Right to be forgotten**
  - Make private data forgotten by wiping keys out

ENCRYPT(keyA, userA)



Event processing at Spotify

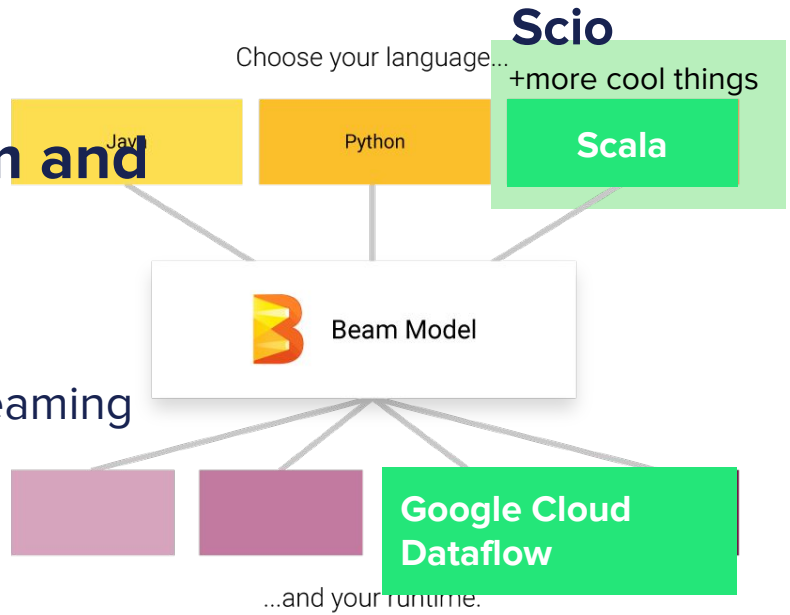




Scio pseudonymization ***pipelines***
for every event
run every hour

We use Scio

- **Scala API for Apache Beam and Google Cloud Dataflow**
- **Apache Beam**
  - Unified model for batch and streaming processing
  - Dataflow : cloud-driven



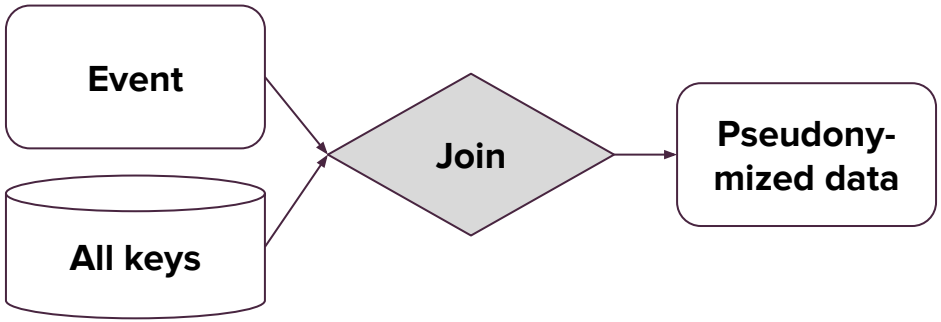
Scio gives...

- Elegant syntax
- Stronger type safety
- In-house optimization

From Beam to Scio

Pseudonymize events

- Generalized Scio pipeline to pseudonymize any data set
- Join hourly event with hourly key dump



Technical challenges

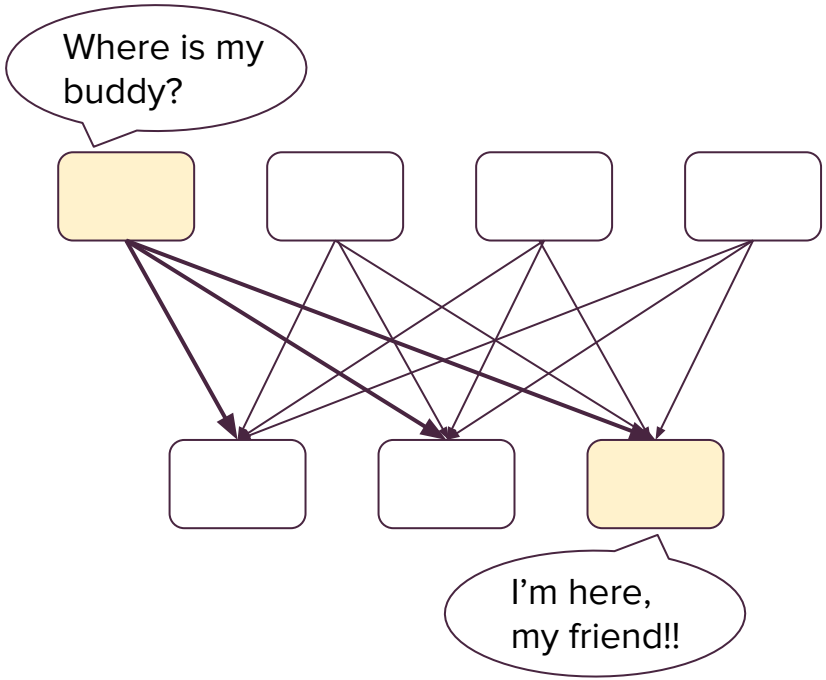
- > 400 event types
- Various data size
- Users are not active all the time

small event optimization

Optimize small event

Join is still slow

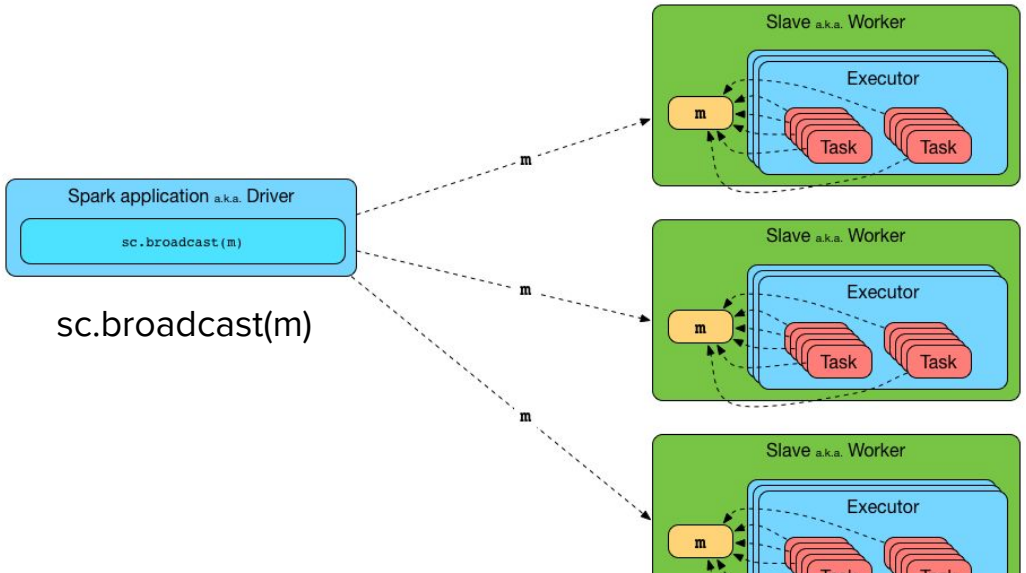
- **Key dump is still big**
  - 207M MAU (2018)
- **Shuffling**
  - Journey to find the same key
  - Network IO is expensive



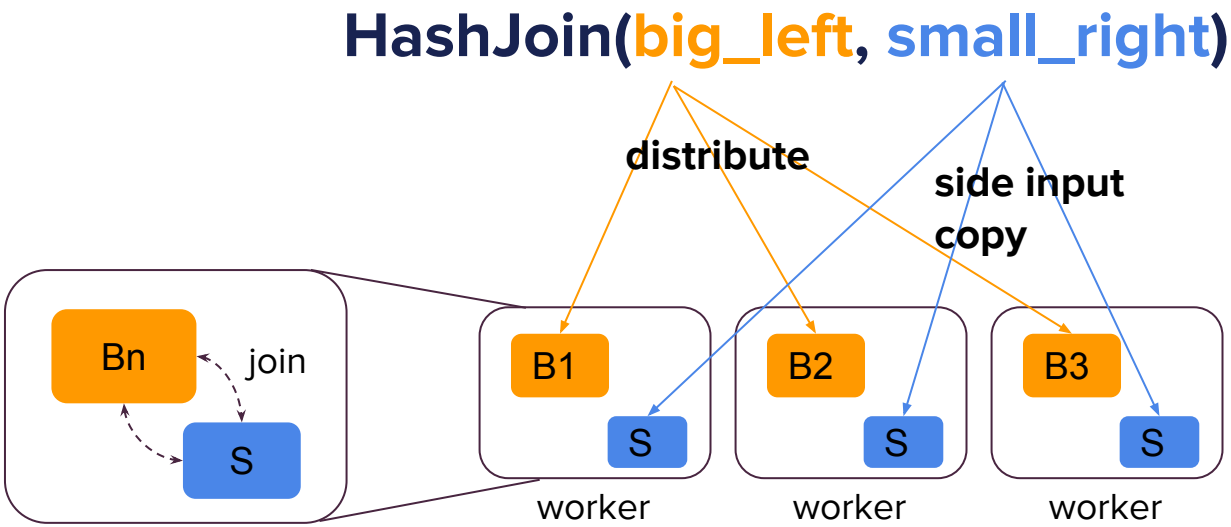
Side input

~~An additional input that your DoFn can access each time it processes an element in the input PCollection~~

Beam copies side input to every worker



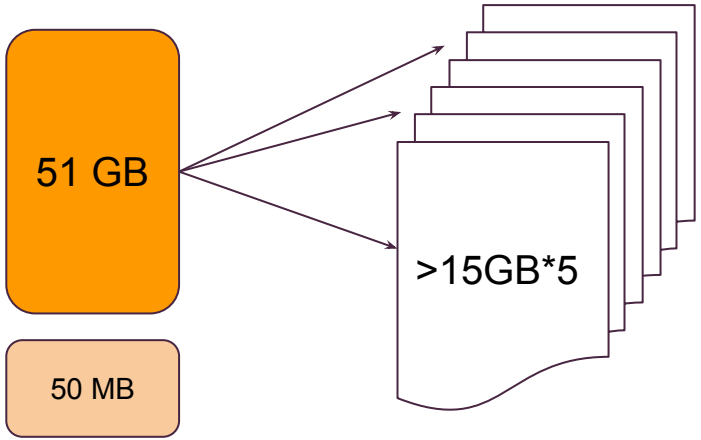
Hash join: idea combined



Verify hash join

Experimentation

- Giant and tiny data set
- Join them either by naive join or by hash join
- Monitor and compare resources



Experimentation pipelines

Expected result

- Hash join is faster
- Can see different resource patterns

Expected result

- Hash join is faster **: 30min > 10min**
- Can see different resource patterns **: YES!**

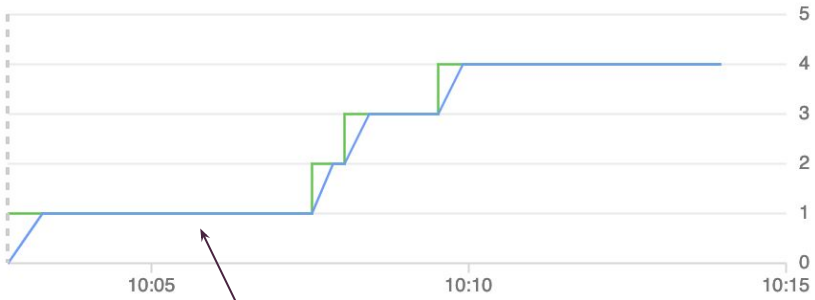
Task scheduling

Hash join

Naive join

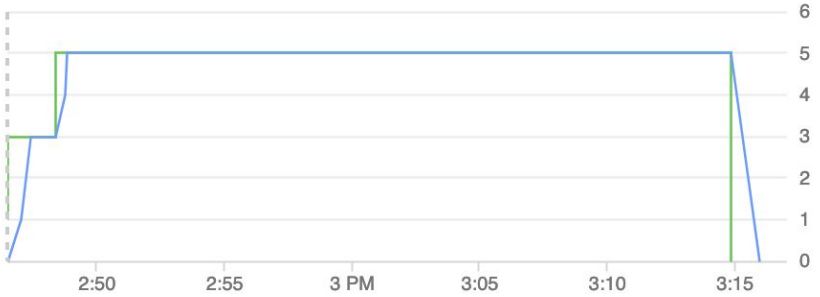
Autoscale workers

Hash join



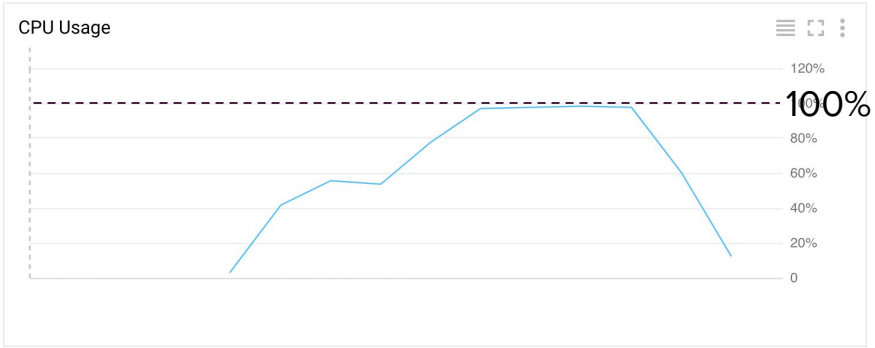
IO for small input

Naive join

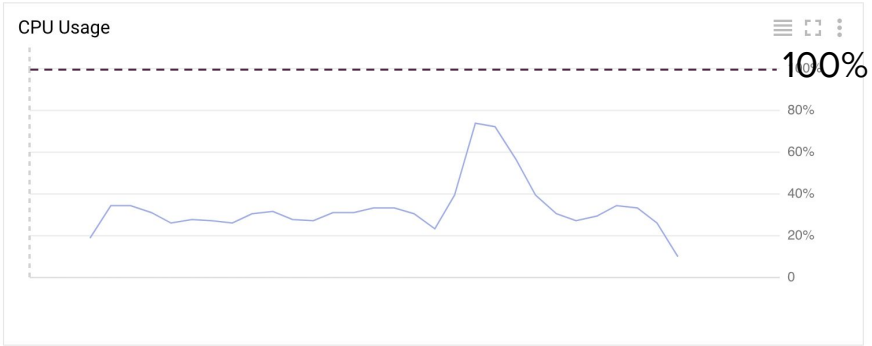


CPU

Hash join



Naive join



Network traffic

Hash join

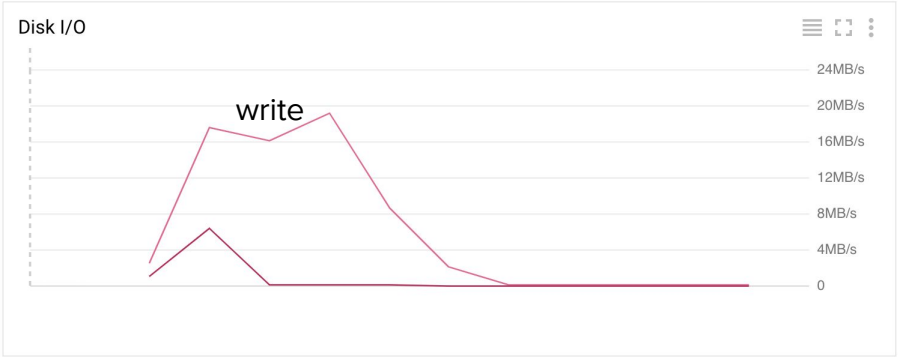
Received bytes at once

Naive join

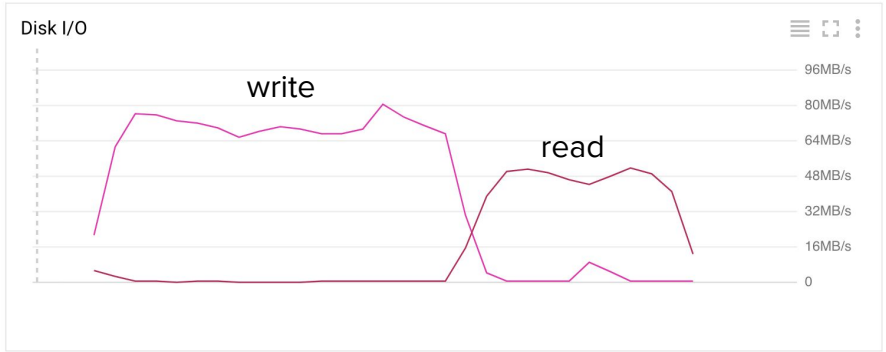
In and out - do shuffle

Disk

Hash join



Naive join



hashJoin implementation

big_left.hashJoin(small_right)

```
def hashJoin[W: Coder](that: SCollection[(K, W)])(implicit koder: Coder[K],
                                                   voder: Coder[V]): SCollection[(K, (V, W))] =
1   self.transform { in =>
2     val side = combineAsMapSideInput(that)
3     implicit val vwCoder = Coder[(V, W)]
4     in.withSideInputs(side)
5       .flatMap[(K, (V, W))] { (kv, s) =>    // (every key-value from big_left, SideInputContext)
6         s(side)
7           .getOrElse(kv._1, ArrayBuffer.empty[W])
8           .iterator
9           .map(w => (kv._1, (kv._2, w)))
10      }
11      .toSCollection
    }
```

W00t! Small events are happy (:

Let’s optimize big event

Optimize big event

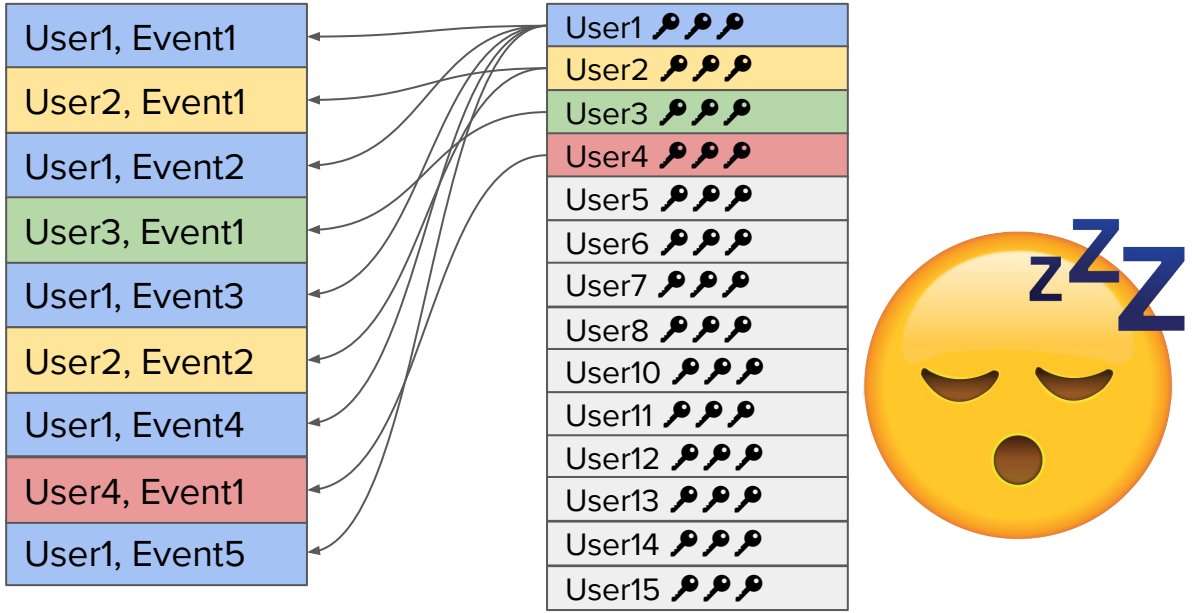
- **They never fit in memory**
- **Turn the other side out : is it possible not to shuffle key dump?**

JOIN(**key_dump**, small_event)

JOIN(big_event, **key_dump**)

Key dump shortcut

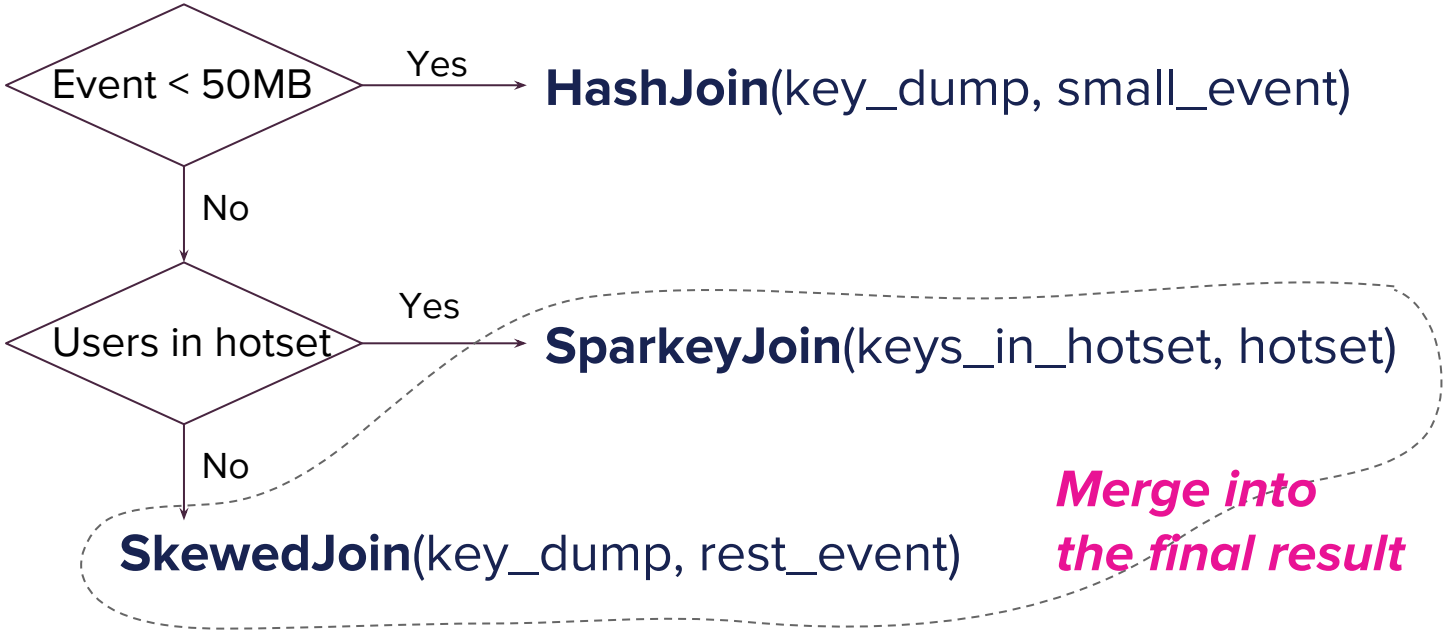
Users are not active all the time



Sparkey join

- **Summarize active users from the most active event**
- **Pre-generate hotset called ‘Sparkey’**
  - You can think of it as disk cache
- **Put Sparkey as side input and join**

Let’s merge optimizations



Let’s merge optimizations

They are different join types.

Our codes are not messy...


Google is expensive.
We save a lot of money!

Lesson learned

- Spotify scale and technical challenges I’ve never seen before
- Don’t guess or imagine. Prove by making hands dirty and have fun

Thank you! :)