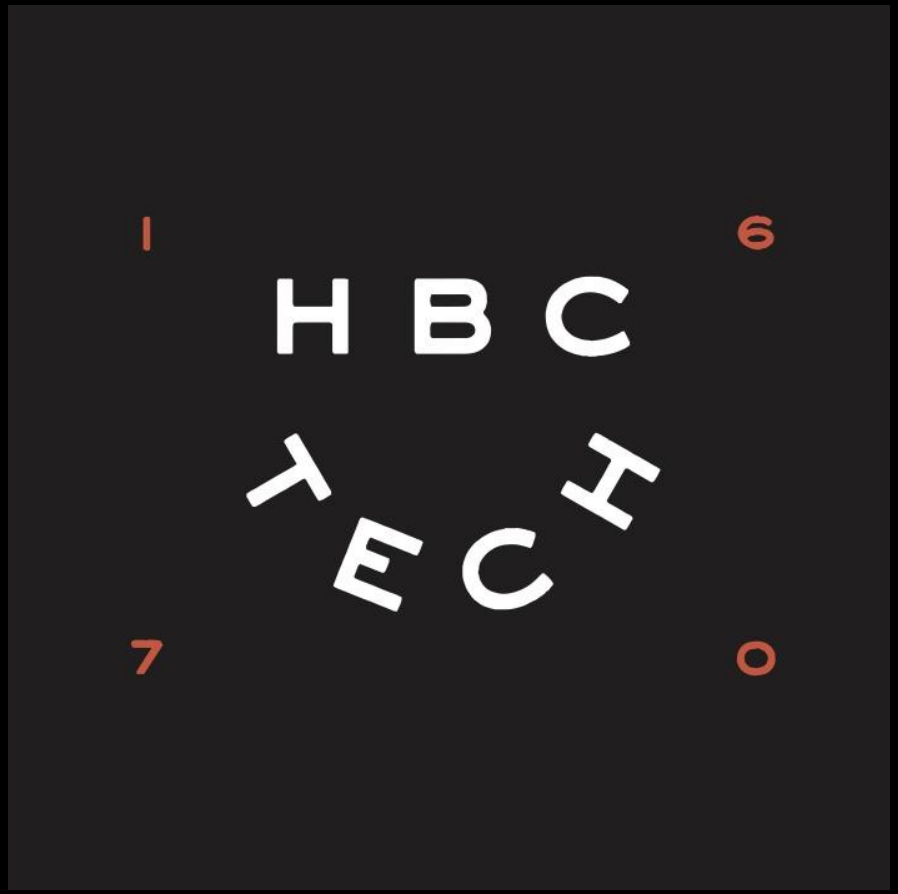


A snapshot, a stream, and a bunch of deltas

Applying Lambda Architectures in a post-Microservice World

Q-Con London March 6th 2018

Ade Trenaman, SVP Engineering, Raconteur, HBC Tech
t: @adrian_trenaman

http://tech.hbc.com

t: @hbcdigital fa: @hbcdigital in: hbc_digital


Hudson's Bay Company.
INCORPORATED 2ND MAY 1670
HISTORIC TRADING POSTS and TERRITORIES
of the Governor & Company of Adventurers of England Trading into Hudson's Bay
ALASKA
GREENLAND
ARCTIC OCEAN
QUEEN ELIZABETH ISLANDS
BAFFIN ISLAND
DAVIS STRAIT
HUDSON STRAIT
HUDSON BAY
PACIFIC OCEAN
ATLANTIC OCEAN
THE BARREN LANDS
MACKENZIE
SASKATCHEWAN
ASSINIBOIA
KEEWATIN
ALBERTA
ONTARIO
QUEBEC
NEWFOUNDLAND
Prince Rupert
Governor


NETFLIX
FRONTIER


HUDSON'S BAY

Saks Fifth Avenue

Saks Fifth Avenue
OFF 5TH

Lord & Taylor
THE DRESS ADDRESS

GALERIA
KAUFHOF

GILT

~$3.5Bn

annual e-commerce revenue


00's of
Stores

What this talk is about

Solving the problem of microservice dependencies with lambda architectures:

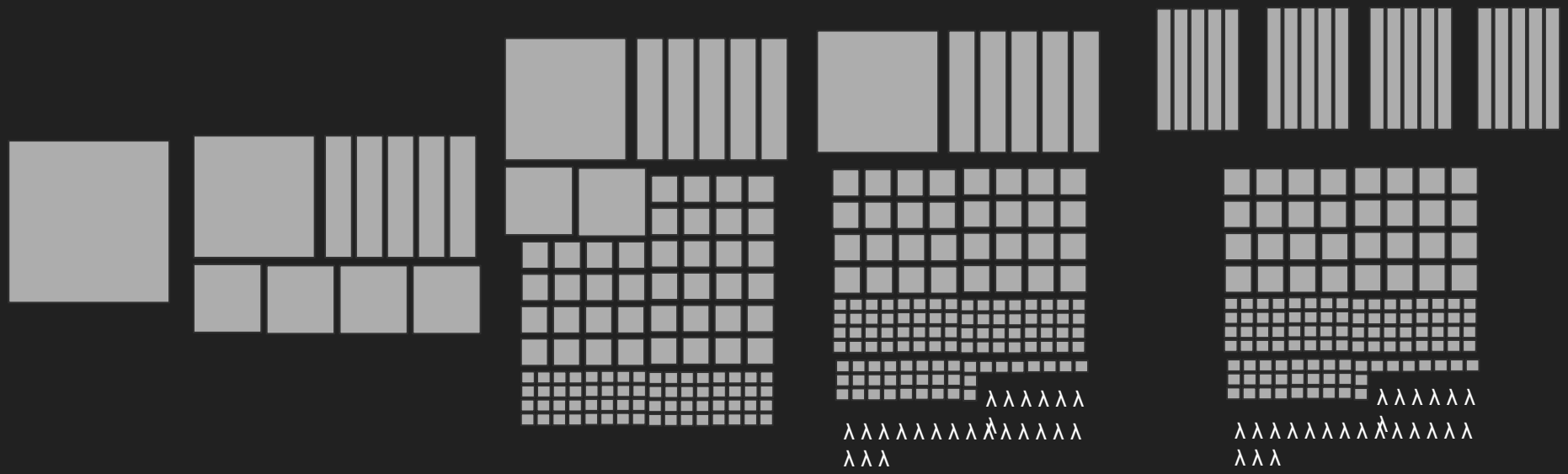
> performance, scalability, reliability

Lambda architecture examples:

> product catalog, search, real-time inventory, third-party integration

Lessons learnt:

> It’s not all rainbows and unicorns
> Kinesis vs. Kafka



2007
Monolith

2010
Service
Oriented

2012
μ-Services

2016
Rise of
Serverless λ

2018+
Multi-banner
Multi-tenant
Multi-region

λ architectures
Streams
GraphQL In the
seams

Some context: a minimalist abstraction of our architectural evolution

6. io explosion

Each service call begets more service calls; some of which are redundant...

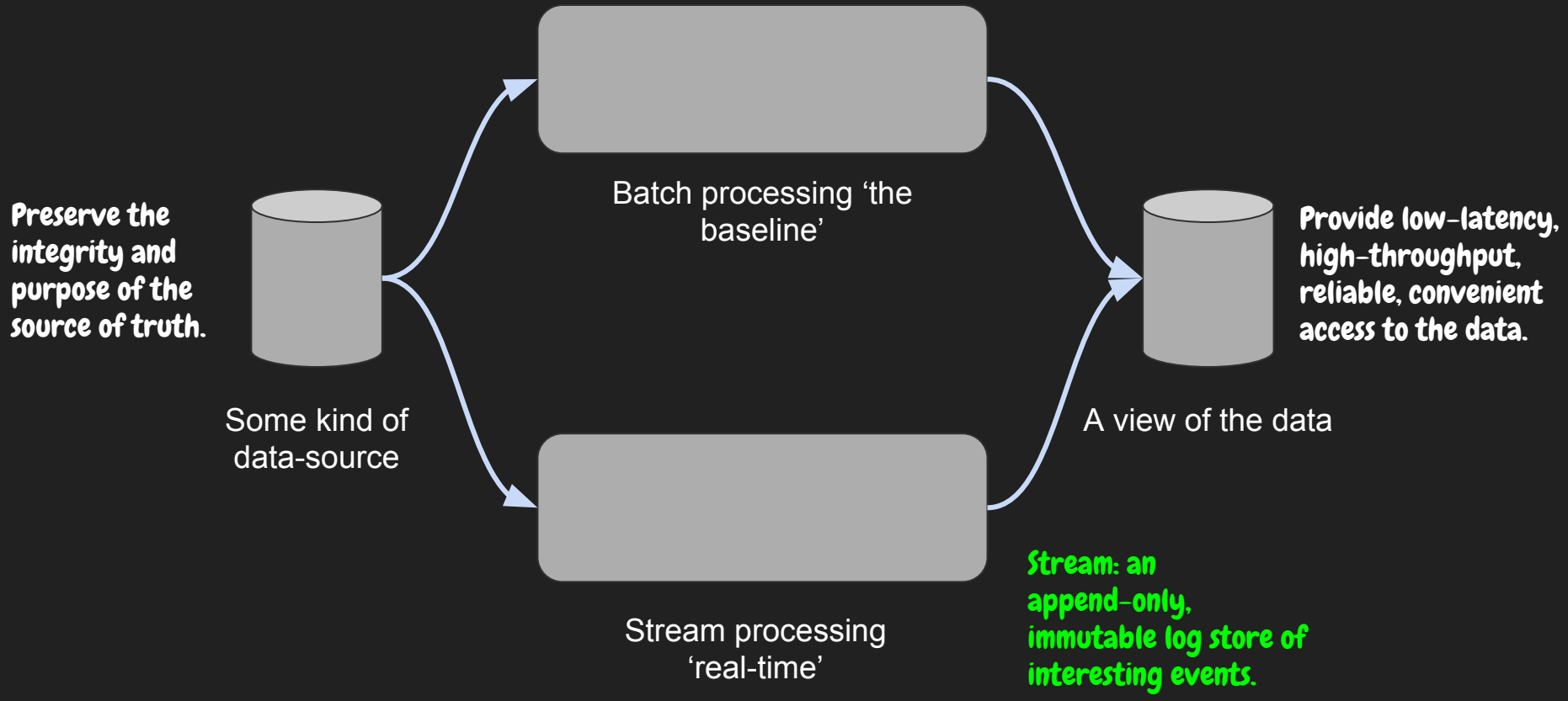
=> unintended complexity and performance

Looking to lambda architecture for critical-path APIs: precompute, real-time updates, O(1) lookup

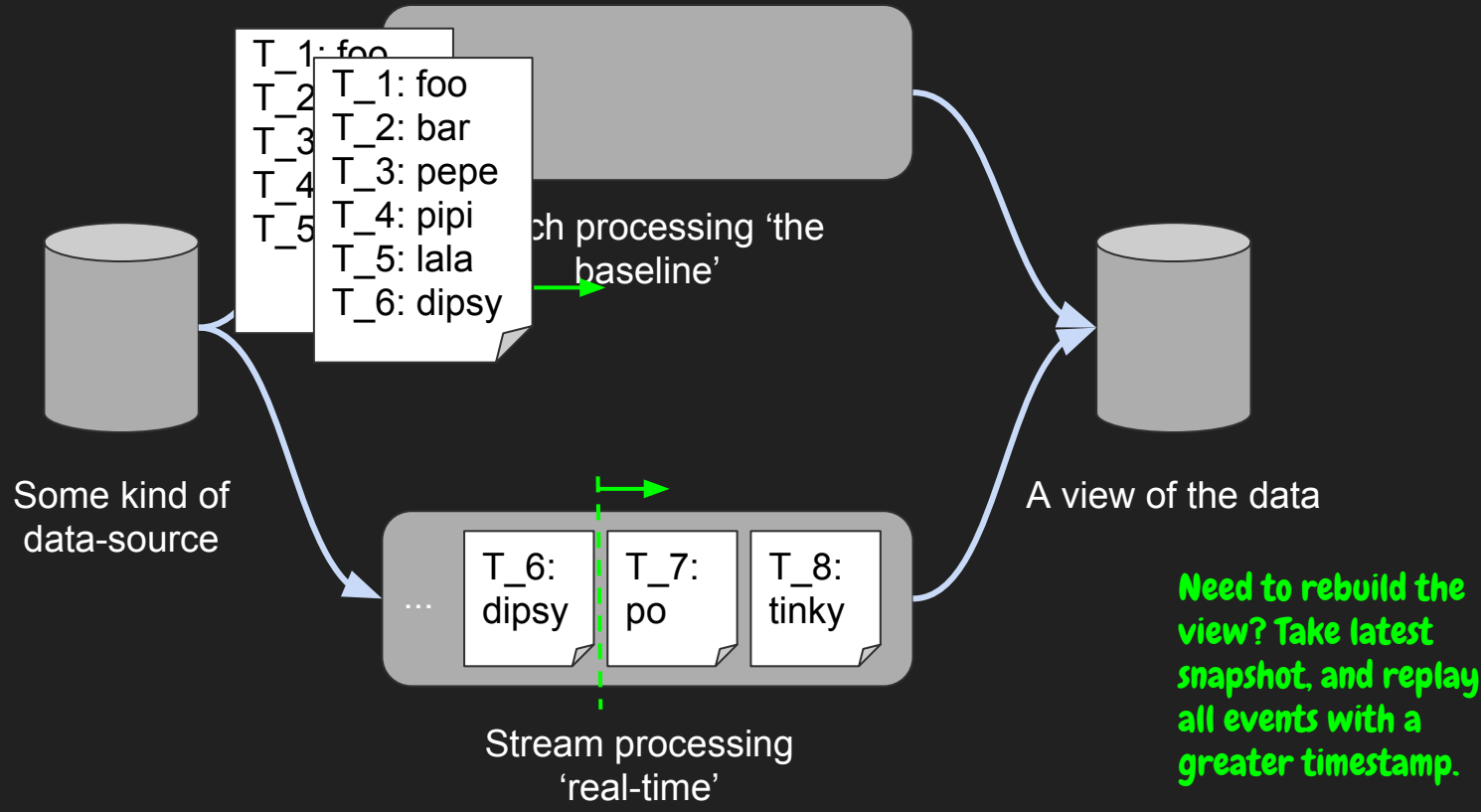
Slam on the breaks! Dublin Microservices Meetup, Feb 2015

Part 0

In which we briefly describe lambda architecture, and the Hollywood Principle



Lambda architecture: making batch processing sexy again.



Lambda architecture: making batch processing sexy again.


“Don’t call us,
we’ll call you.”
HOLLYWOOD

Preserve the integrity and purpose of the source of truth.

Provide low-latency, high-throughput, reliable, convenient access to the data.

Some kind of data-source

A view of the data

Inversion of control: previously, we ask for data when we need it.

Preserve the integrity and purpose of the source of truth.

Provide low-latency, high-throughput, reliable, convenient access to the data.

Some kind of data-source

A view of the data

Inversion of control: now, when the data changes, we are informed.

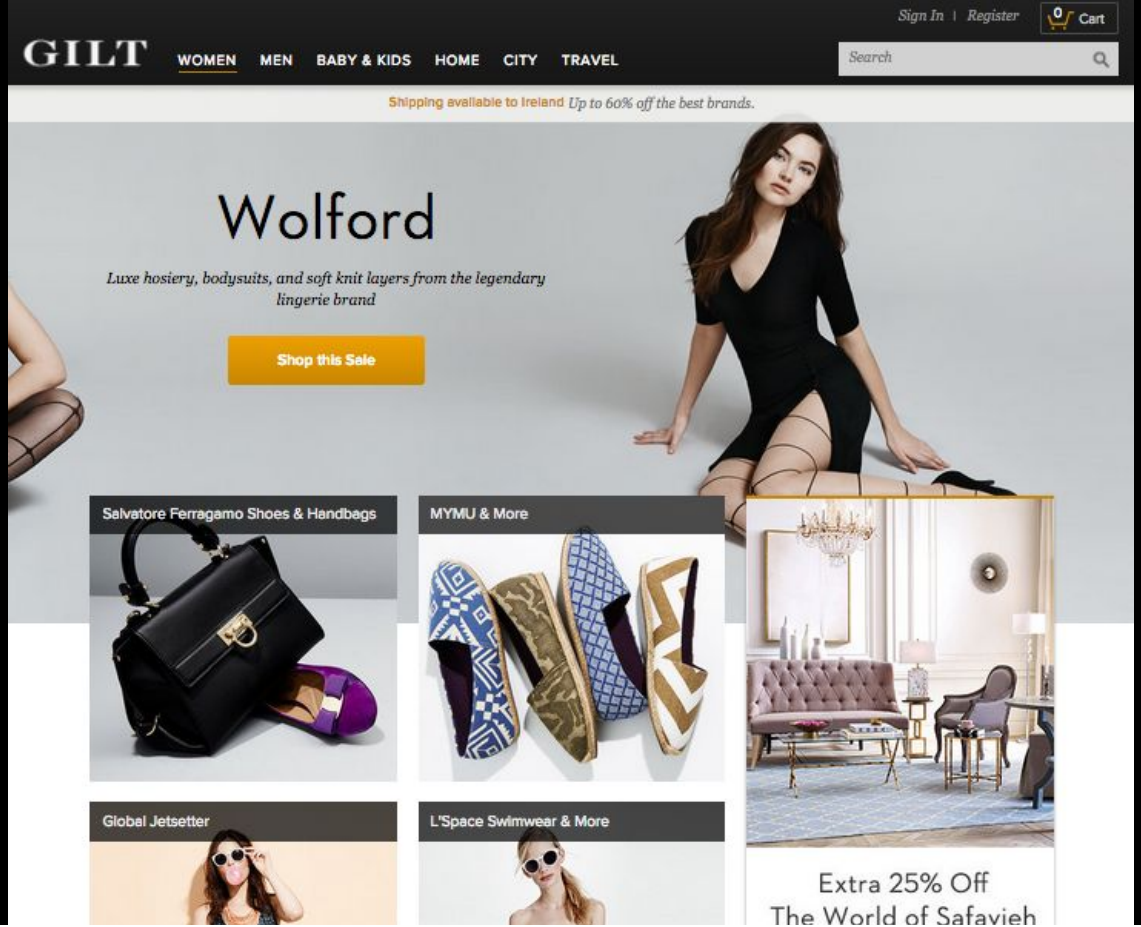
Part I

In which we learn the perils of caching in a microservices architecture, and how lambda architecture helped us out.

Gilt: we source luxury brands...

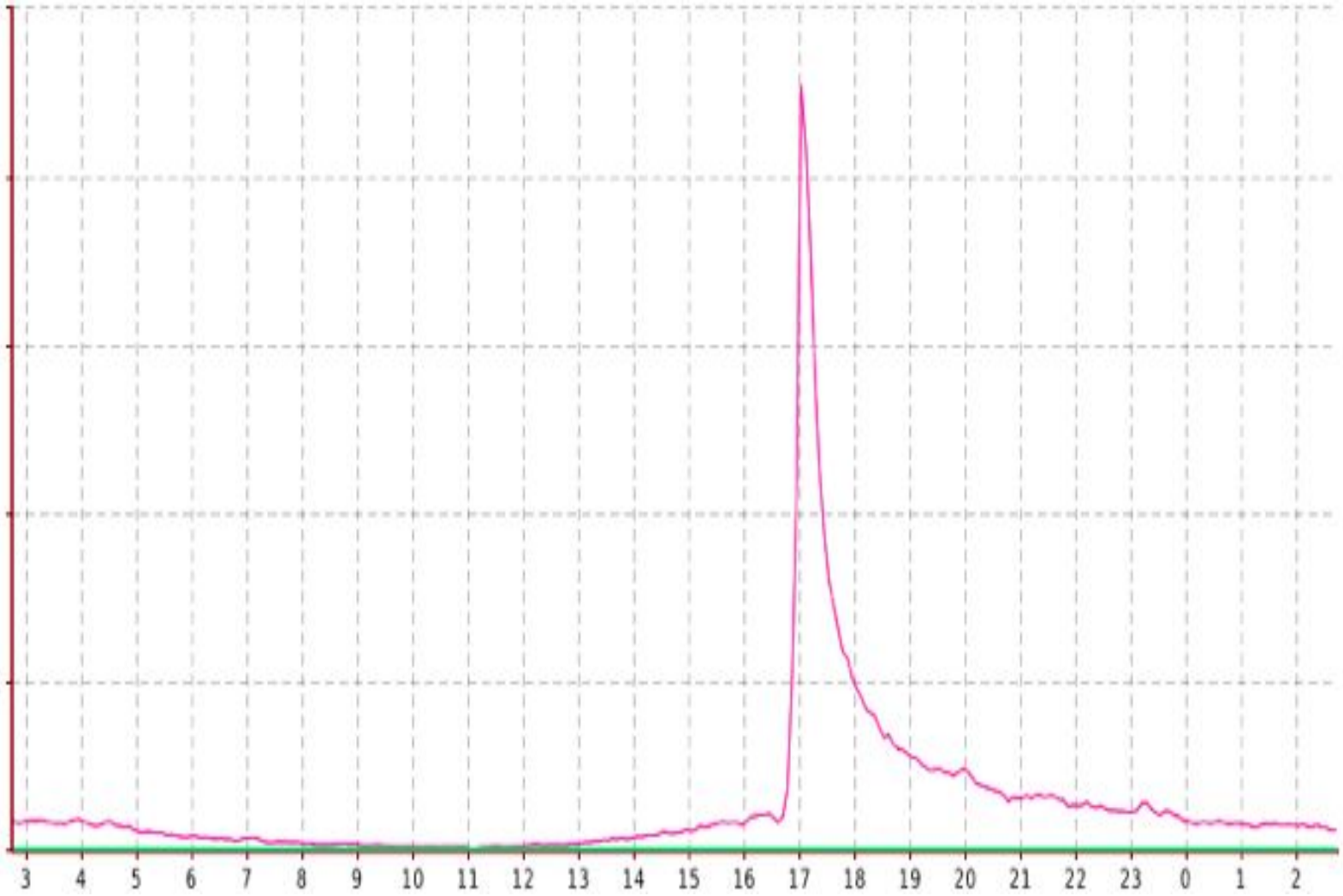
… we shoot the product in our studios

… we receive



… we sell every day at noon

… stampede!


3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
0
1
2

The Gilt Problem

Massive pulse of traffic, *every* day.

=> serve fast

Low inventory quantities of high value merchandise, changing rapidly

=> can’t cache

Individually personalised landing experiences

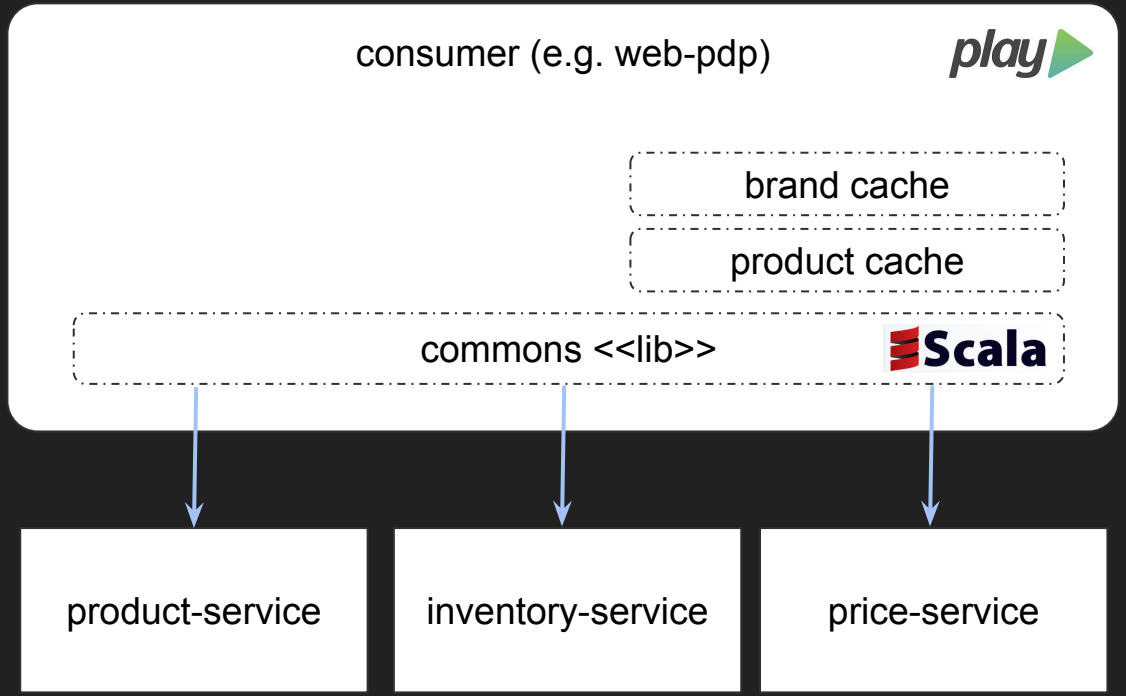
=> can’t cache

Caching

"Just say no."

"Until you *have* to say yes."

"Then, just say *maybe*."

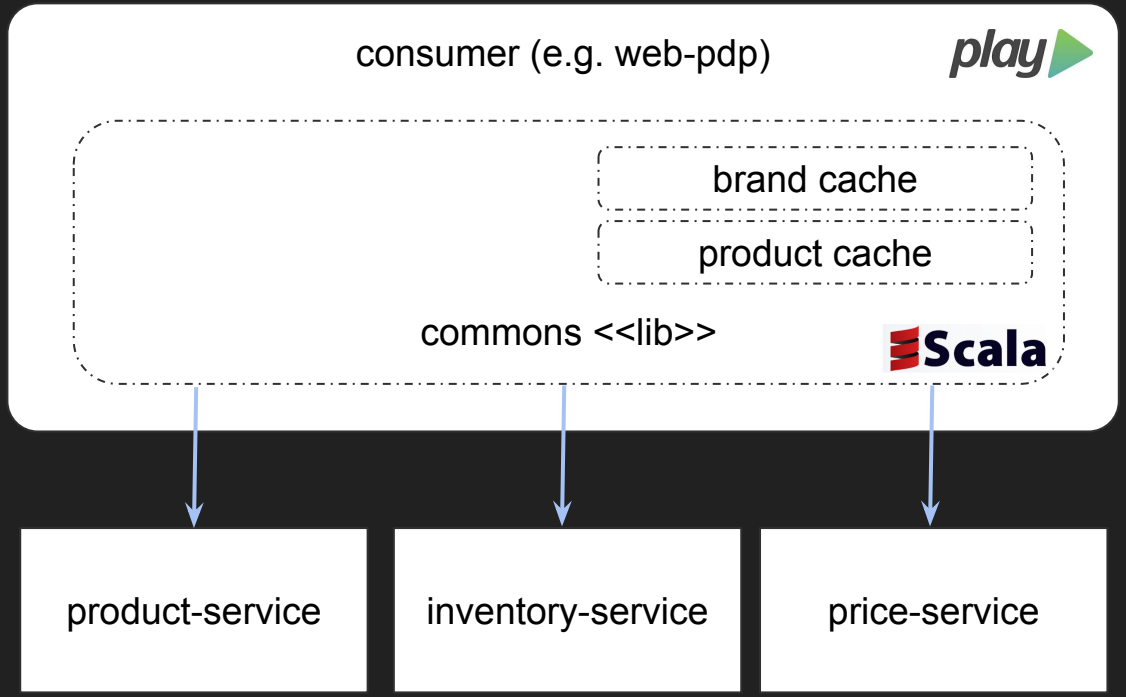


Hmm, engineer adds a local brand cache to reduce network calls..

... and then later, another cache for product information.

Leads to (1) arbitrary caching policies, & (2) duplicated cache information.

A stateless, cache-free library, *busted*.



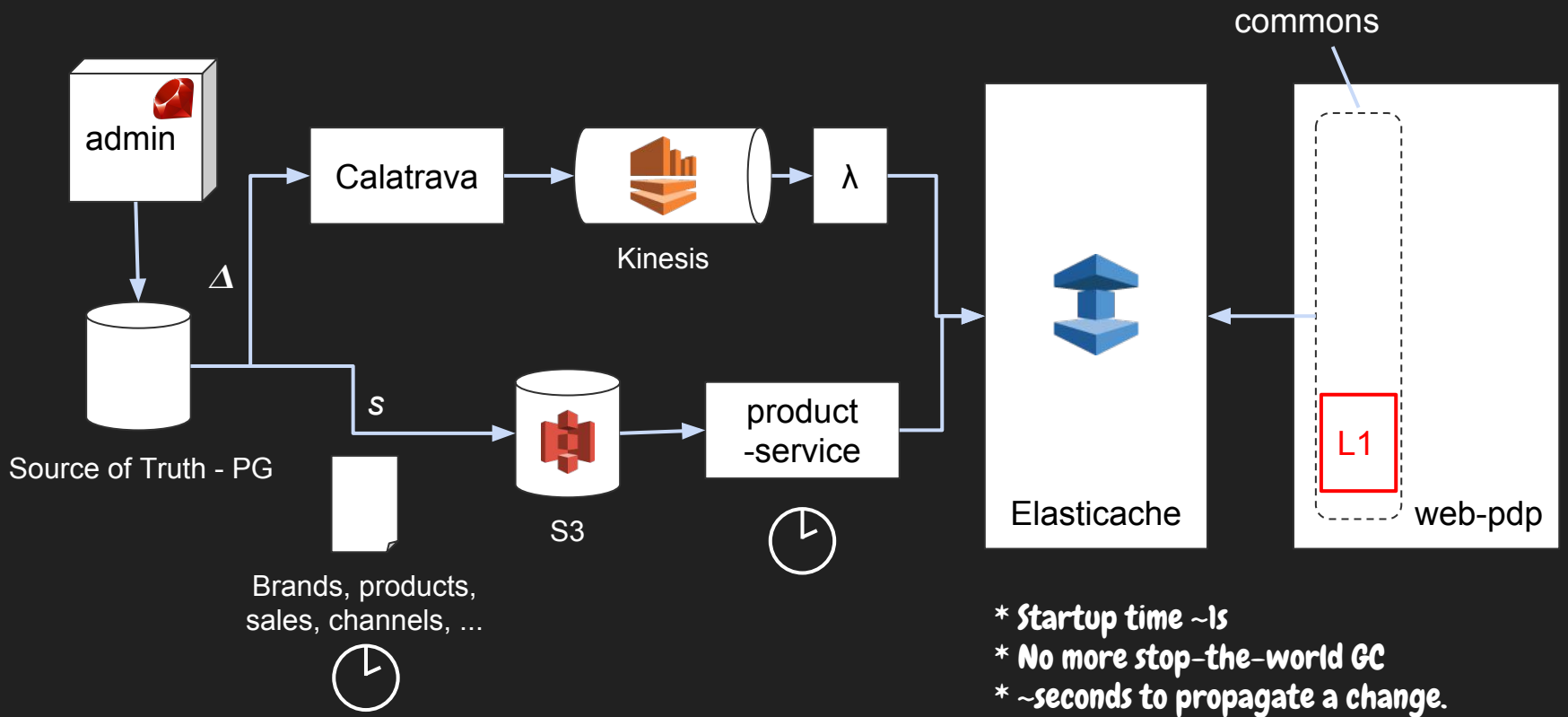
We changed the commons library to cache products with a consistent, timed refresh (20m).

Worked well, until the business changed its mind about one small thing: let's make *everything* in the warehouse sellable.

Orders of magnitude more SKUs:

* JSON from product service > 1Gb
* Startup time > 10m
* JVM garbage collection every 20m on cache clear
* ~1hr to propagate a change.
* m4.xlarge, w/ 14Gb JVM Heap

A caching library. Worked well initially, but...



Near real-time caching at scale

* Startup time ~1s
* No more stop-the-world GC
* ~seconds to propagate a change.
* c4.xlarge (CPU!!!), w/ 6Gb JVM Heap

Next: replace JSON marshalling with binary OTW format (e.g. AVRO)

README.md

calatrava build unknown

Calatrava is a service that can monitor changes in database tables and turn them into events in a Kinesis stream.

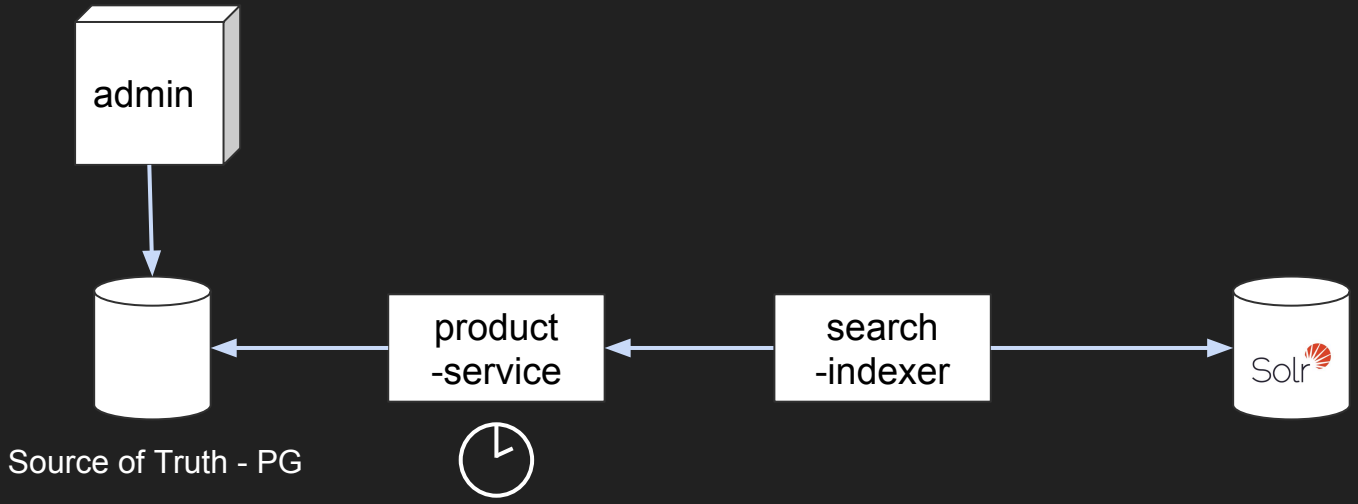
From the moment you activate the **bridge** (a connection from a database table to a stream), every INSERT, UPDATE or DELETE happening in that table will appear as an event that carries a `before` and an `after` field in JSON form.

See documentation.

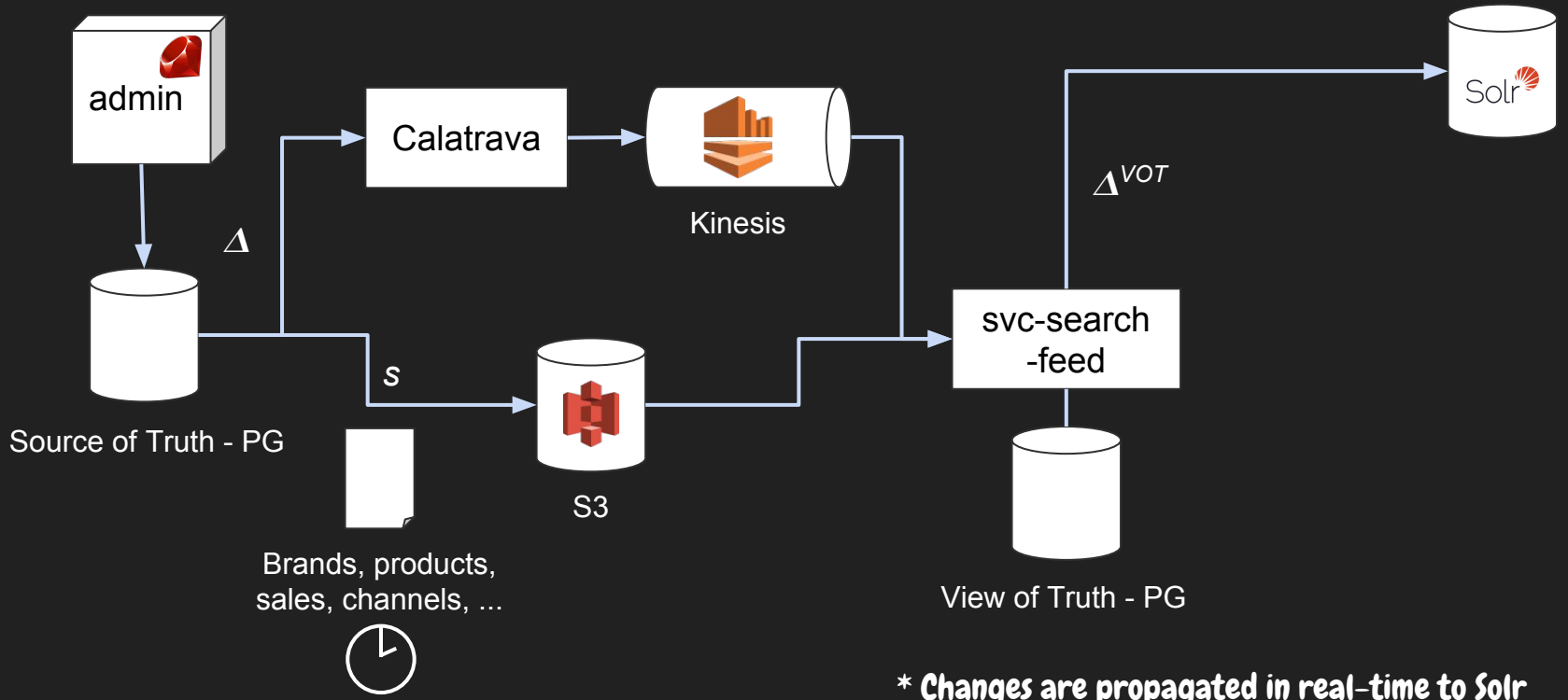
https://github.com/gilt/calatrava - soon to be public

Part 2

In which we learn how we've used Lambda architecture to implement a near real-time search index, but needed an additional *relational* 'view of truth'.



Problem: polling a polling service means changes to product data are *not* reflected in realtime.

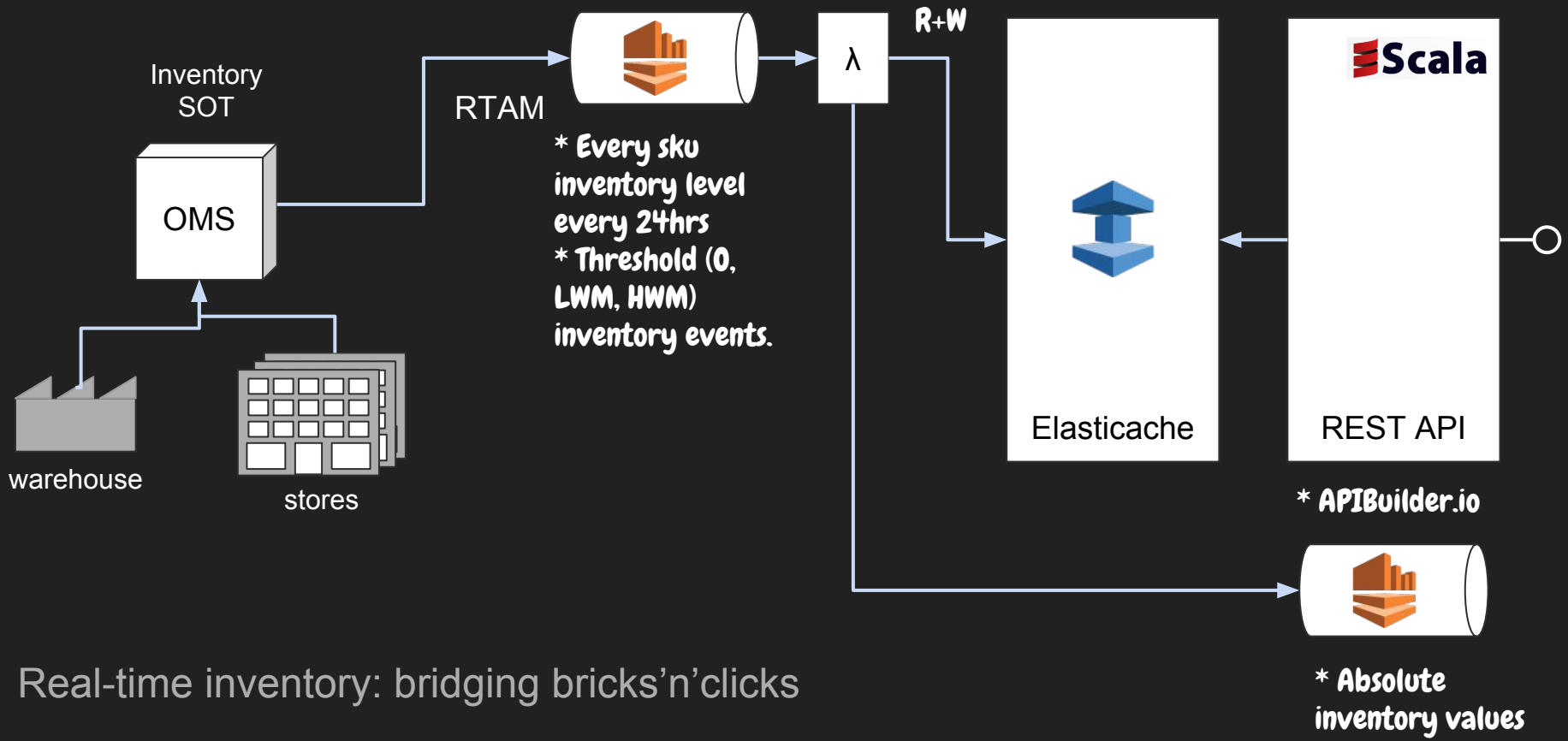


* Changes are propagated in real-time to Solr
* Rebuild of index (s + Δ*) with zero down time
* Same logic for batch & stream (thank you akka-streams)
* V.O.T.: "We needed a relational DB to solve a relational problem"

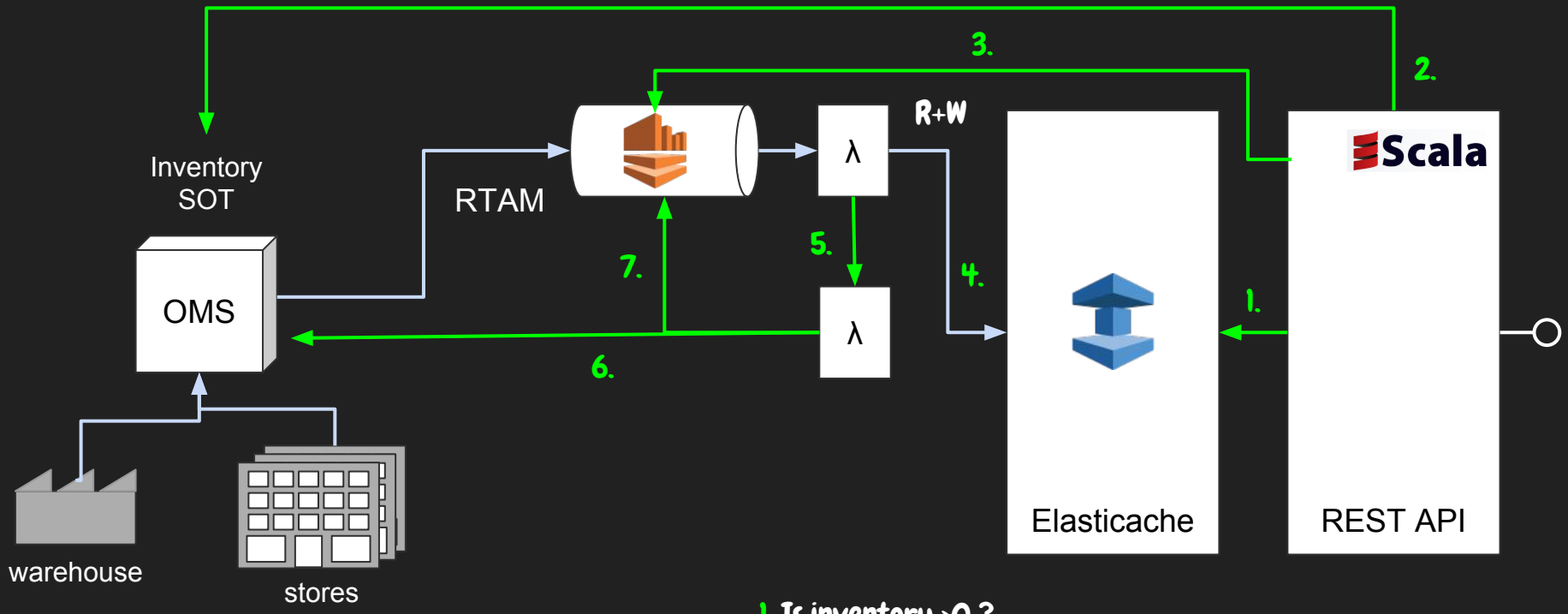
Part 3

In which we use a lambda architecture to facade an unscalable unreliable system as a reliable R+W API… and benefit from *always using the same flow*.

Real-time inventory: bridging bricks’n’clicks



Real-time inventory: bridging bricks’n’clicks



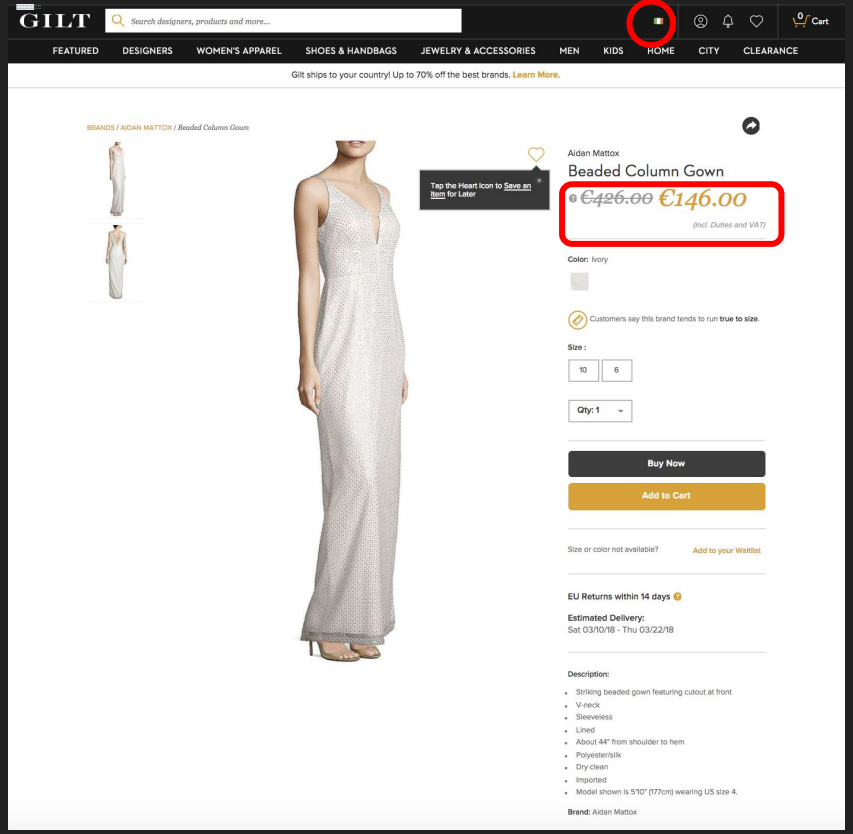
1. Is inventory >0 ?
2. Attempt a reservation with OMS. IF it fails, generate a random reservation ID.
3. Put the change on the RTAM stream
4. Update the cache (and stream, not shown)
5, 6, 7. Trigger a best effort to true-up inventory with ATP (available to purchase)

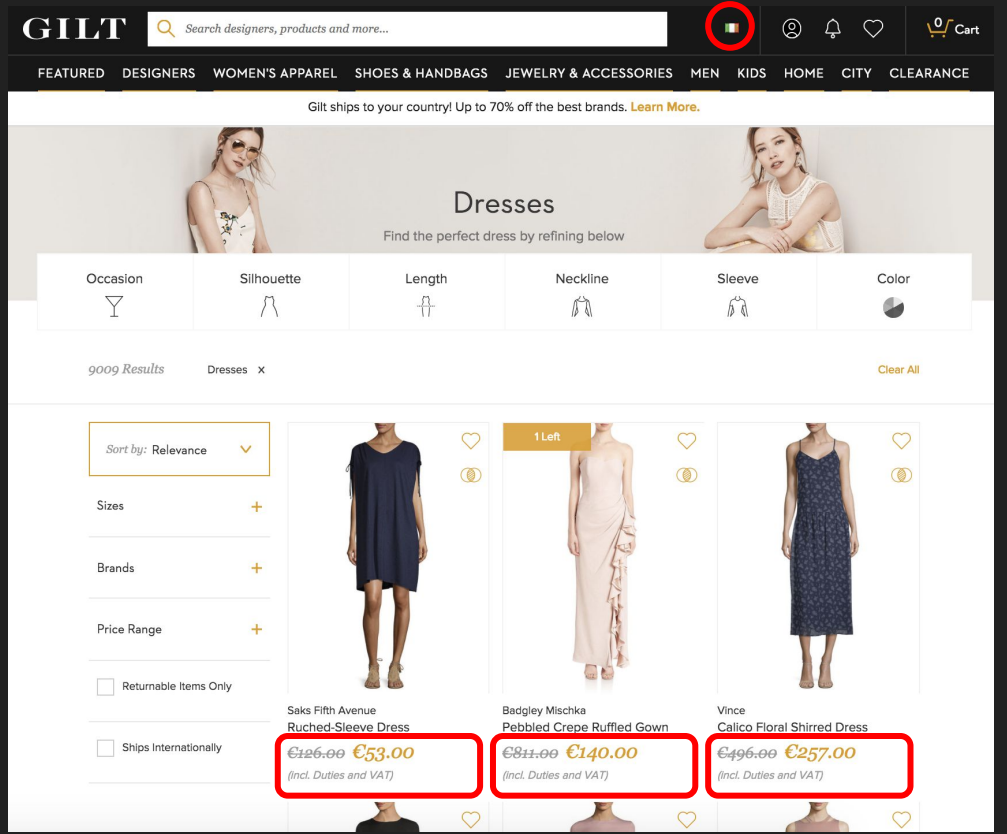
Making a web reservation


THERE IS ONLY ONE PATH
X

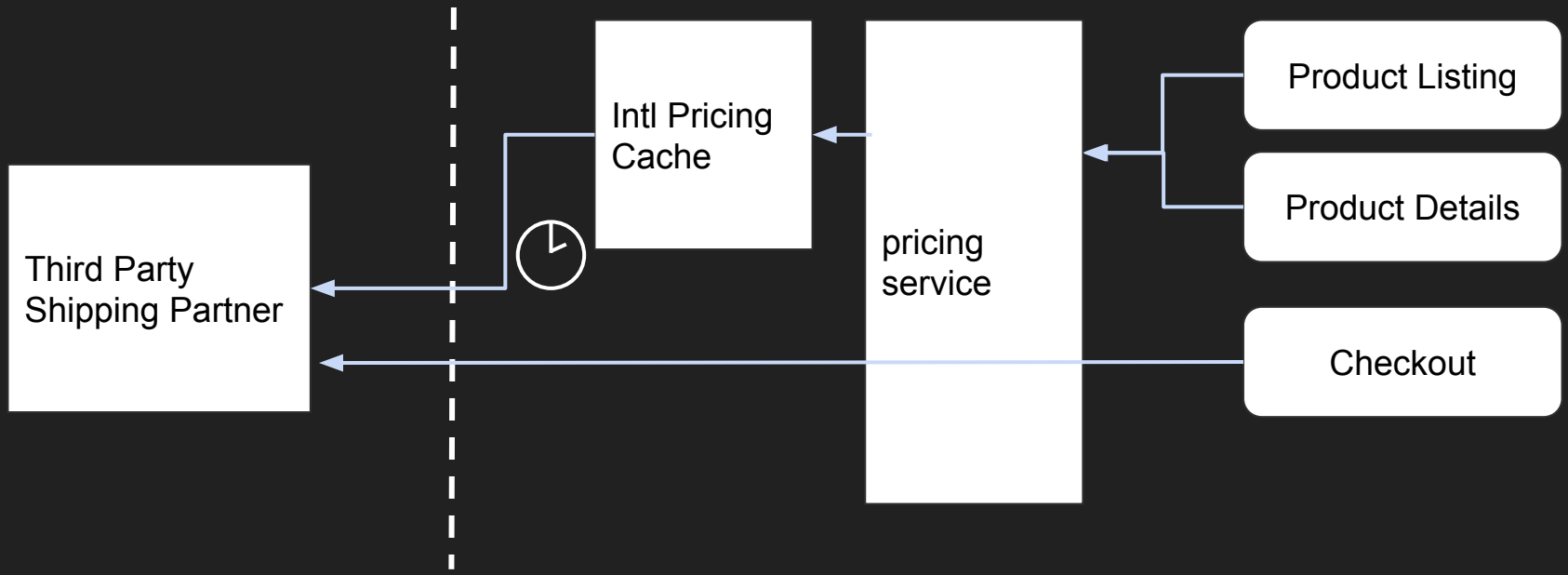
Part 4

In which we learn that the paradigm generalises across third-party boundaries.

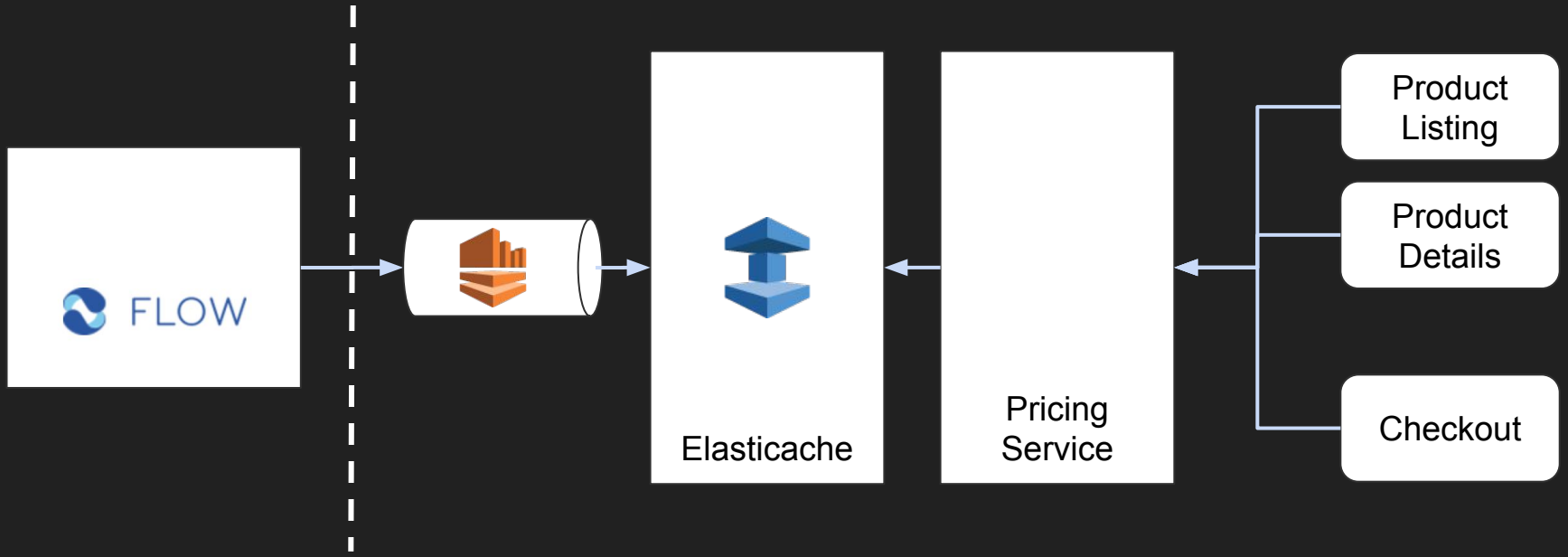




International E-Commerce: Taxes, Shipping & Duty is HARD. Performance is critical!



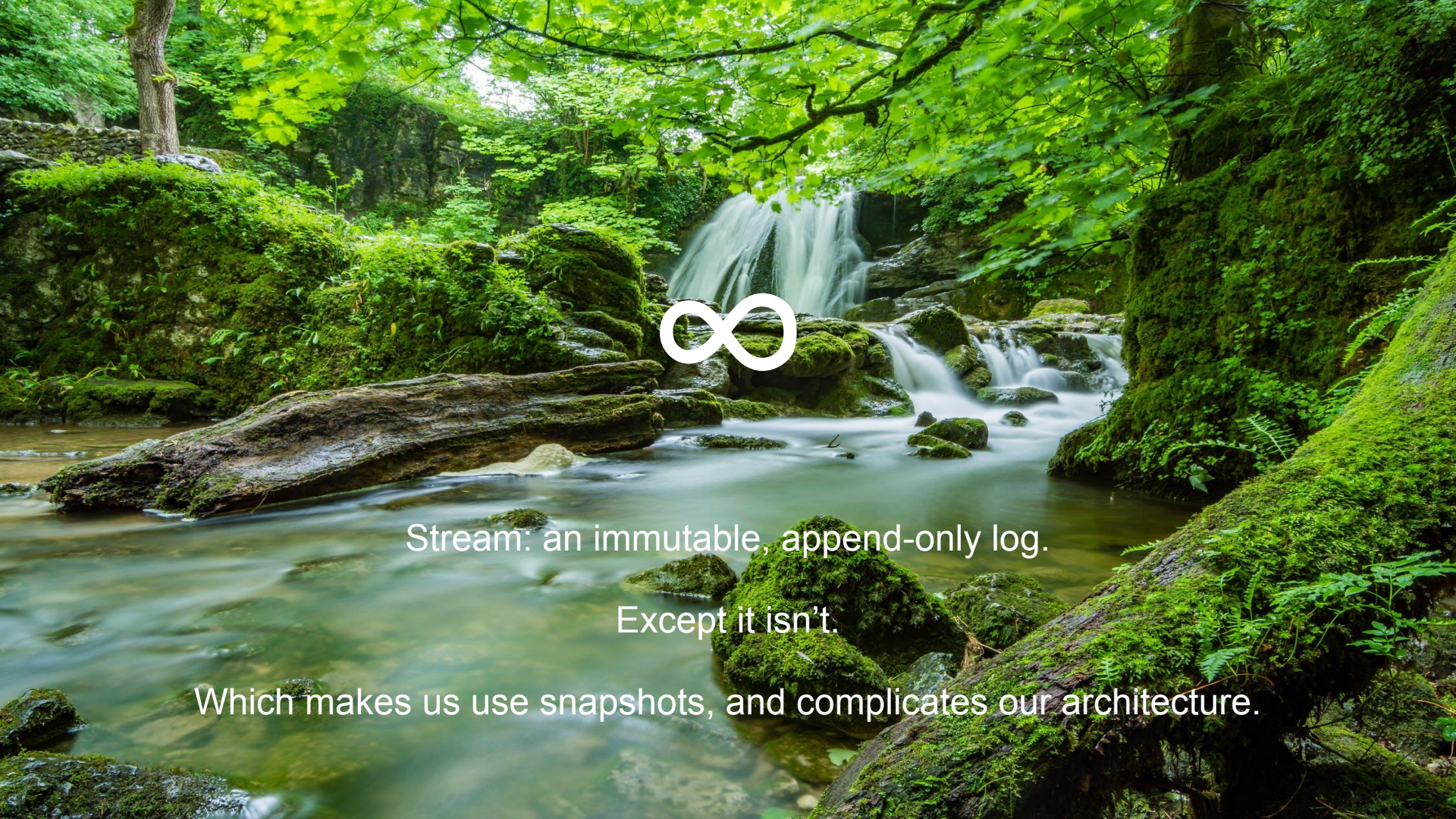
Typical solution: cache for PDP & PA, go direct at checkout. Asymmetric, with chance of sticker-shock.



Stream driven solution with flow.io

Part 5

In which we consider Kafka vs. Kinesis

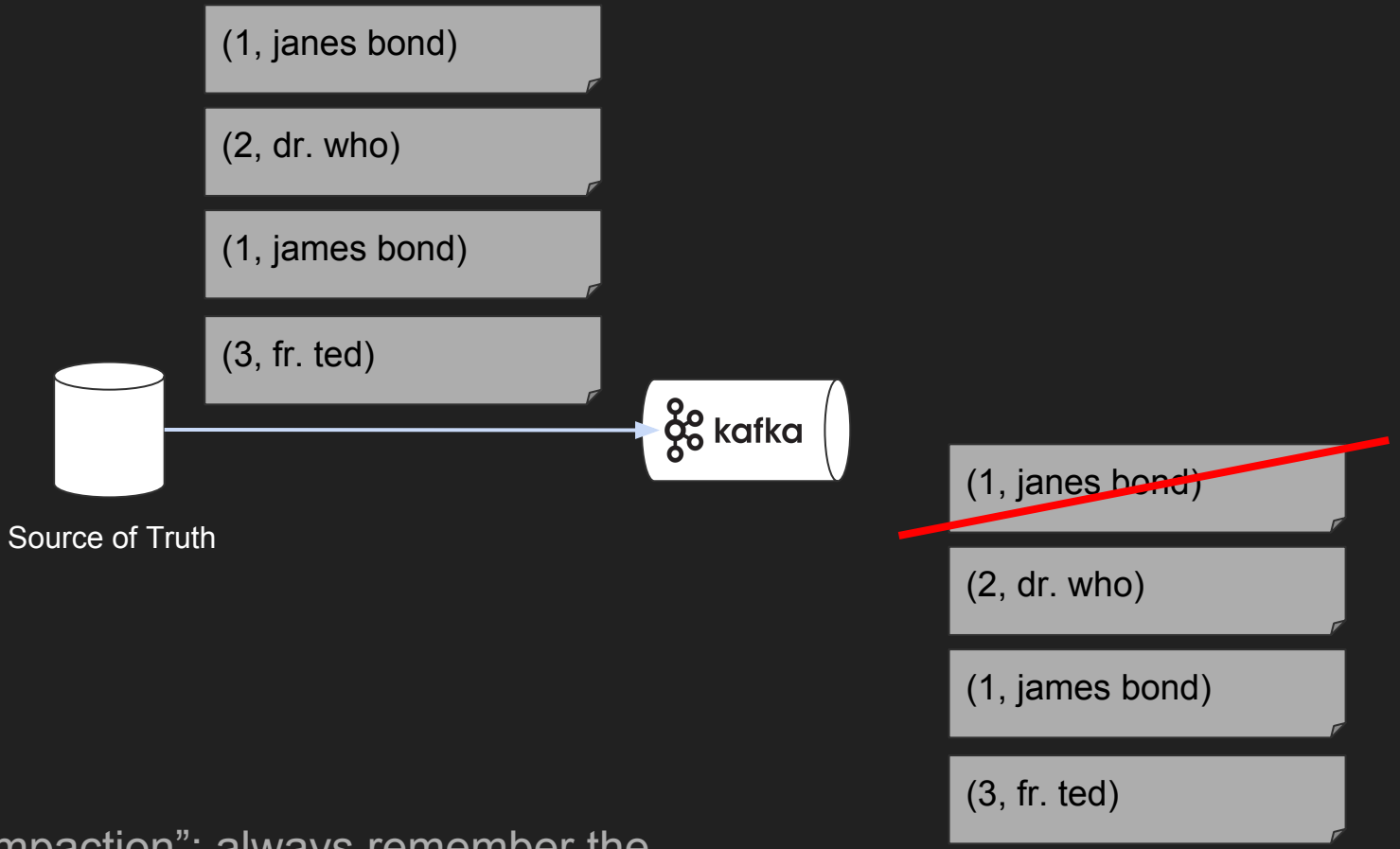


Stream: an immutable, append-only log.

Except it isn’t.

Which makes us use snapshots, and complicates our architecture.


LOG
COMPACTION



“Log compaction”: always remember the latest version of the same object.


TABLE
STREAM
DUALITY

KTable & Kafka Streams Library

K-Table & Kafka Streams...

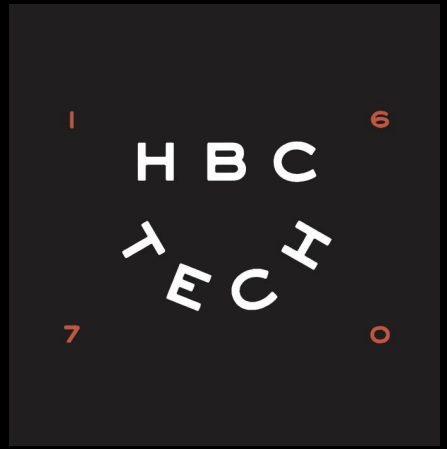
```
3    val builder = new StreamsBuilderS()
4
5    val productTable: KTableS[String, GenericRecord] = builder.table("products")
6    val inventoryTable: KTableS[String, GenericRecord] = builder.table("inventory")
7
8    val groupedInventory = inventoryTable
9      .groupBy(productIdMapper)
10     .reduce(joinInventories, joinInventories)
11
12   productTable
13     .join(groupedInventory, joinProductInventories)
14     .toStream
15     .to("product-inventory")
16
17   val streams = new KafkaStreams(builder.build(), streamsConfiguration)
```

(0) Apply lambda arch to create scalable, reliable offline systems.

(1) Replicate and transform the one source of truth

(2) It’s not all unicorns and rainbows: complex VOT, snapshots

(3) Kinesis is the gateway drug; Kafka is the destination.



#thanks @adrian_trenaman @gilttech @hbcdigital