Beams, Streams & Finance

Neil Stevenson,
Principal Architect, Hazelcast

Agenda

What Is Beam ?

Beam in Finance

Choosing an Implementation Provider

Quick Demo

Hazelcast & Intel

hazelcast

intel

What Is Beam ?

What is Beam ?

https://beam.apache.org/

A standard for batch and stream processing

Origins from Google, became an Apache project in 2016

Based mainly around the dataflow paradigm

SDK available for Python, Java and GoLang

Multiple implementation providers

What is Beam ?

Declarative processing, described as a graph

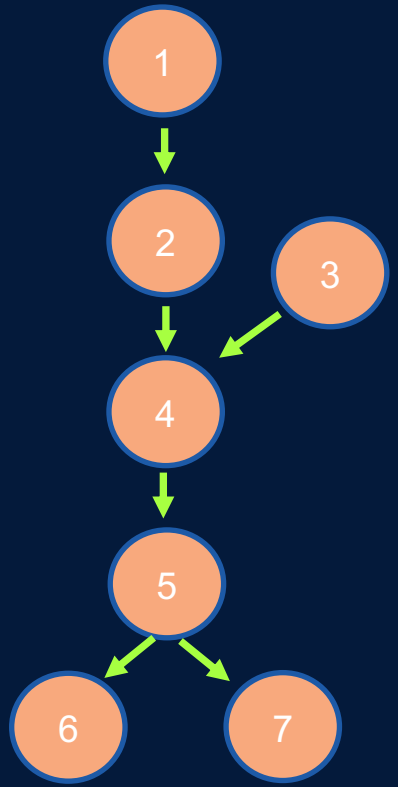
Data flows from stage to stage

Each stage is self-contained, has no knowledge of what is before or after, only sees the data

Stages are stateful or stateless

One input to a stage need not result in one output

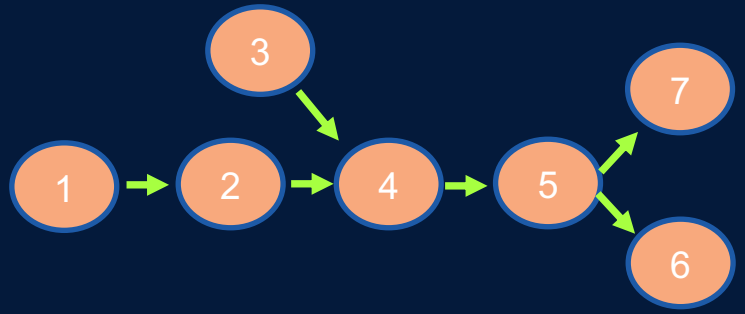
No loops



intel

hazelcast

What is Beam ?



3 building blocks

(a) Source 

Read from database, file, etc

(b) Intermediate 

Filter, compute, enrich, deplete, aggregate

(c) Sink 

Write to database, file, etc

intel

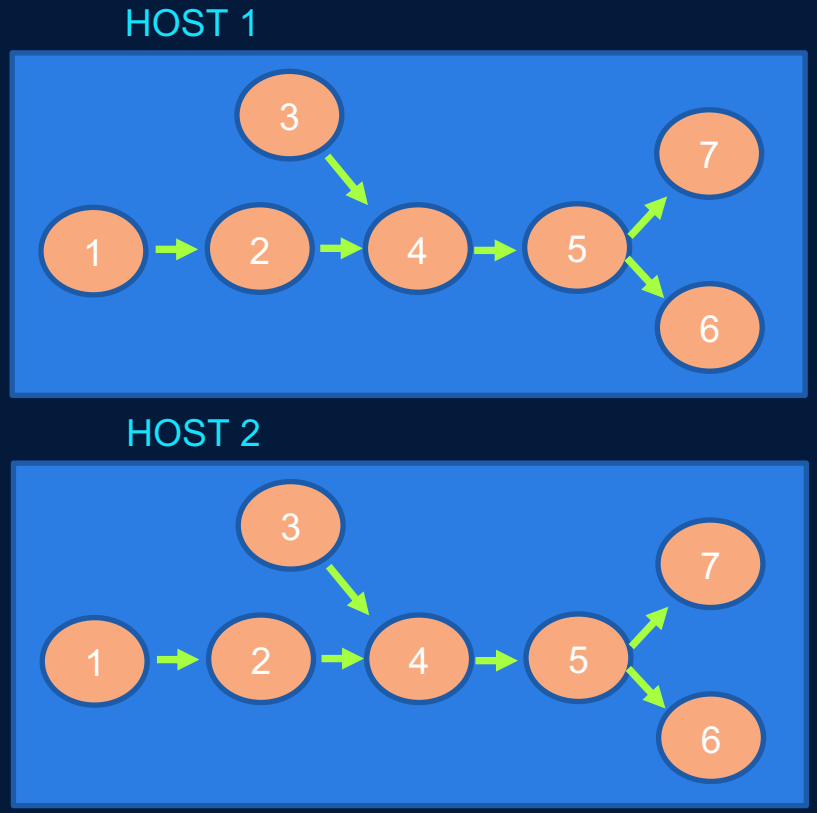
hazelcast

What is Beam ?

Parallelized!

One processing job, two instances

Data can flow from host to host



intel

hazelcast

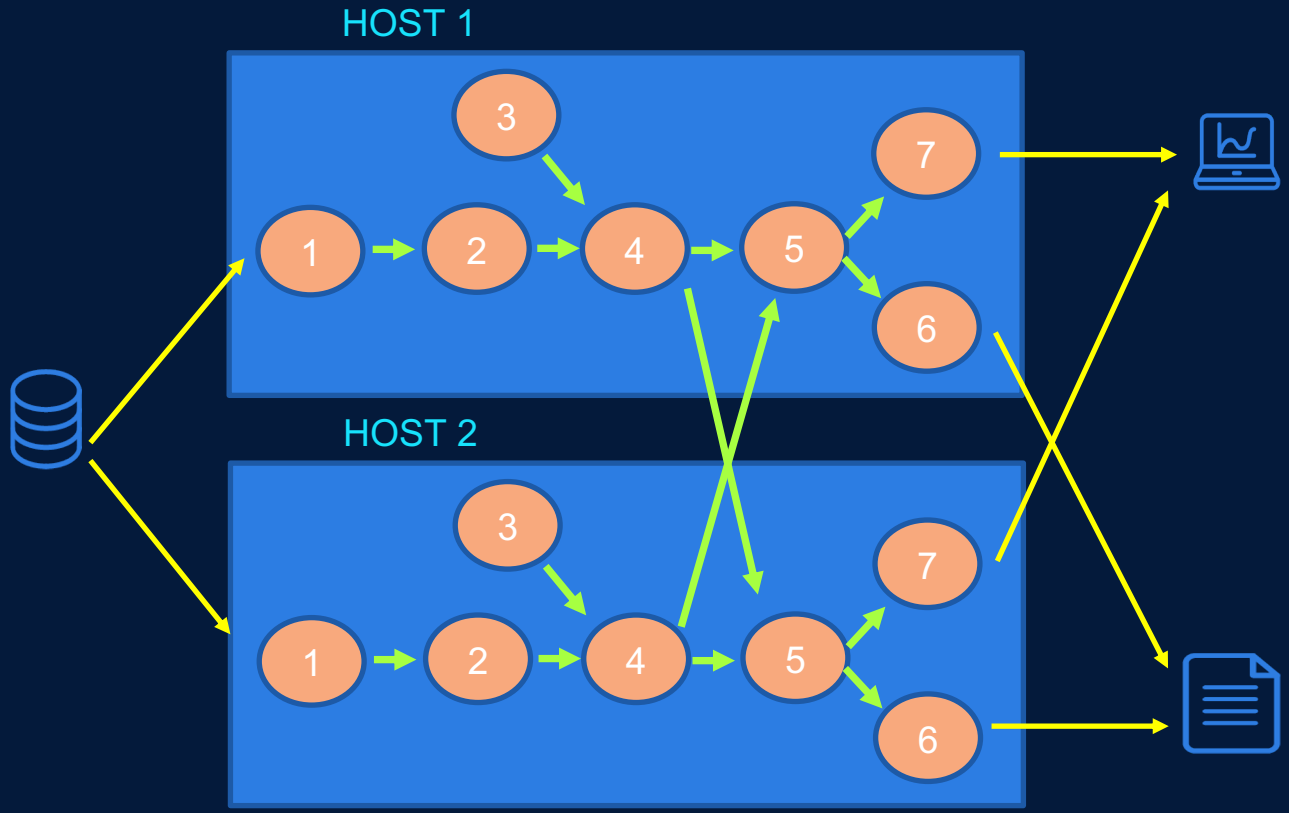
What is Beam ?

Parallelized!

One processing job, two instances

Data can flow from host to host

Stage 4 feeds some or every stage 5



intel

hazelcast

What is Beam ?

Tip:

Some sources are infinite – eg. temperature sensor
Some sources are finite – read a database table

But *think* of a bounded source as unbounded

Then a trigger or CDC on a database table can give a continuous feed of changes

Beam In Finance

Beam in Finance

Reporting

Any type of "*evaluate all X against Y scenario*" type calculation. Partition the *X* across hosts to achieve the SLA. Add more hosts when *Y* goes up, such as month end.

Alerting

Watch the markets to detect trends in prices, eg. Bollinger Bands
=> issue alerts when trending changes from up->down or down->up

Management

Dynamically re-value portfolios as prices & forex rates fluctuate
=> force sell if over-exposed to to one market

Beam in Finance

Credit Value Adjustment

(a) Interest Rate Swap trades
(b) Interest Rate curves

Simplistically:

Calculate the Mark-To-Market for each trade with each curve
Convert to the MTM to a CVA exposure
Discard out-of-the-money
Calculate the average CVA exposure per trade
Sum these per counterparty

Beam in Finance

Credit Value Adjustment

In Beam terms:

Combine every (a) and every (b)
Process these combinations to produce (c)
Transform (c) to (d)
Filter some (d)
Average (d) by primary key giving (e)
Sum (e) by foreign key

Beam in Finance

Credit Value Adjustment

```
PCollection<D> d
    = c.apply(ParDo.of(new TransformCintoD()));
    .apply(ParDo.of(new TransformAintoKV()));

PCollectionView<Map<String, B>> bView
= pipeline
    .apply(TextIO.read().from("./curves.csv"))
    .apply(ParDo.of(new TransformBintoKV()))
    .apply(View.asMap());

PCollection<C> c
    = a.apply(ParDo.of(new BusinessLogic(bView)).withSideInputs(bView));
```

In Beam terms:

Combine every (a) and every (b)
Process these combinations to produce (c)
Transform (c) to (d)
Filter some (d)
Average (d) by primary key giving (e)
Sum (e) by foreign key

intel

hazelcast

Beam in Finance

Credit Value Adjustment

```
PCollection<D> d
    = c.apply(ParDo.of(new TransformCintoD()));
```

In Beam terms:

Combine every (a) and every (b)
Process these combinations to produce (c)
Transform (c) to (d)
Filter some (d)
Average (d) by primary key giving (e)
Sum (e) by foreign key

Beam in Finance

Credit Value Adjustment

```
PCollection<D> d
    = c.apply(ParDo.of(new DoFn<C, D>() {
     @ProcessElement
     public void process(@Element C c, OutputReceiver<D> receiver) {
         D d = new D();
         d.setValue(-1 * c.getValue());
         receiver.output(d);
     }
}));
```

In Beam terms:

Combine every (a) and every (b)
Process these combinations to produce (c)
Transform (c) to (d)
Filter some (d)
Average (d) by primary key giving (e)
Sum (e) by foreign key

Beam in Finance

Credit Value Adjustment

```
PCollection<D> d2
= d.apply(Filter.by(new SerializableFunction<D, Boolean>() {
        public Boolean apply(D d) {
            return d.value > 0;
        }
    }));
```

In Beam terms:

Combine every (a) and every (b)
Process these combinations to produce (c)
Transform (c) to (d)
Filter some (d)
Average (d) by primary key giving (e)
Sum (e) by foreign key

Beam in Finance

Credit Value Adjustment

```
PCollection<KV<String, E>> e =
    d2
    .apply(ParDo.of(new TransformDintoKV()))
    .apply(GroupByKey.create())
    .apply(ParDo.of(new TransformDintoE()));
```

In Beam terms:

Combine every (a) and every (b)
Process these combinations to produce (c)
Transform (c) to (d)
Filter some (d)
Average (d) by primary key giving (e)
Sum (e) by foreign key

Beam in Finance

Credit Value Adjustment

```
e
.apply(ParDo.of(new TransformEintoF()))
.apply(GroupByKey.create())
.apply(ParDo.of(new TransformFintoString()))
.apply(TextIO.write().to("./results.txt"));

return pipeline;
```

In Beam terms:

Combine every (a) and every (b)
Process these combinations to produce (c)
Transform (c) to (d)
Filter some (d)
Average (d) by primary key giving (e)
Sum (e) by foreign key

intel

hazelcast

Beam in Finance

Credit Value Adjustment

BUT

When we did a CVA example at Hazelcast, we discarded Beam in favor of directly coding in Jet

Because

We could optimize it more, for intraday CVA

Choosing An Implementation Provider

Choosing An Implementation Provider

Beam is an SDK or API to define a job in a implementation neutral way

A ”Beam runner” is software that is an execution environment for such a job, dealing with parallelism, data marshalling and general runtime

Using Beam makes it easier

- developers are easier to find, a standard skillset
- no vendor lock-in
- using Beam now allows projects to be developed in parallel to strategic projects
- (change from Beam runner A to Beam runner B)

Choosing An Implementation Provider

Various exist (12):

- Hazelcast Jet
- Google Cloud Dataflow
- Apache Spark
- Apache Flink

etc

All the usual considerations

- Operational familiarity, performance, licensing model, cost, export restrictions, cloud enabled, developer preference

PLUS one major and one minor caveat…….

Choosing An Implementation Provider

The minor caveat

- Input & Output

There are standard connectors for reading files, writing files

- Baked in assumptions on directory paths, cross-mounts may need a little adjustment when changing Beam runner

Some standard connectors for specific set-ups

- BigQueryIO, SpannerIO for Google source/sink

Choosing An Implementation Provider

The major caveat

- all Beam runners do not implement all features

https://beam.apache.org/documentation/runners/capability-matrix

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark (RDD/DStream based)	Apache Spark Structured Streaming (Dataset based)	Apache Hadoop MapReduce	JStorm	IBM Streams	Apache Samza	Apache Nemo	Hazelcast Jet	Twister2
ParDo	✓	✓	✓	✓	~	✓	✓	✓	✓	✓	✓	✓
GroupByKey	✓	✓	✓	~	~	✓	✓	✓	✓	✓	✓	✓
Flatten	✓	✓	✓	✓	~	✓	✓	✓	✓	✓	✓	✓
Combine	✓	✓	✓	✓	~	✓	✓	✓	✓	✓	✓	✓
Composite Transforms	✓	~	~	~	~	✓	✓	~	~	✓	~	~
Side Inputs	✓	✓	✓	✓	~	✓	✓	✓	✓	✓	~	✓
Source API	✓	✓	✓	✓	~	~	✓	✓	✓	✓	✓	✓
Splittable DoFn (SDF)	~	✓	✓	~	×	×	×	×	~	×	×	~
Metrics	~	~	~	~	~	~	~	~	~	×	~	×
Stateful Processing	✓	~	~	~	×	~	~	~	~	×	~	×

Choosing An Implementation Provider

The major caveat

- all Beam runners do not implement all features

https://beam.apache.org/documentation/runners/capability-matrix

You need to know what features you need, now and in the future

Choosing An Implementation Provider

Pragmatic points:

No one ever changes implementation provider

Blurred distinction
- Be careful:
  Developers will put ”optimizations” into Beam SDK code that rely on the execution environment

Performance
- If you can:
  POC without business logic as Beam and without Beam on your runner

Demo – “VaR”

Demo – “VaR” (?)

An outline of VaR

(1) Take all portfolios
(2) Take all prices
(3) For all combinations
(4) Simulate a range of possible movements
(5) Select 5% worst, 1% worst, whatever

Demo – “VaR” (?)

An outline of VaR

(1) Take all portfolios
(2) Take all prices
(3) For all combinations
(4) Simulate a range of possible movements
(5) Select 5% worst, 1% worst, whatever

```
for (int i = 0 ; i < portfolios.length ; i = i+1) {
    double sum = 0;
    for (int j = 0 ; j < prices.length; j = j+1) {
        sum = sum + businessLogic(portfolios[i], prices[j]);
    }
}
```

Demo – “VaR” (?)

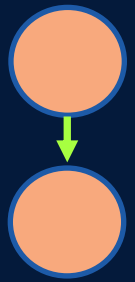
An outline of VaR

(1) Take all portfolios
(2) Take all prices
(3) For all combinations

Demo – “VaR” (?)

An outline of VaR

(1) Take all portfolios
(2) Take all prices
(3) For all combinations

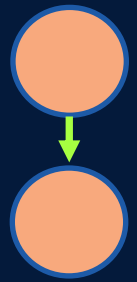


```
PCollection<KV<String, Account>> account
   = pipeline
     .apply("bounded-account-source", Read.from(new MyAccountSource()))
     .apply("account-string-to-entry", ParDo.of(new MyAccountTransform()));
```

Demo – “VaR” (?)

An outline of VaR

(1) Take all portfolios
(2) Take all prices
(3) For all combinations

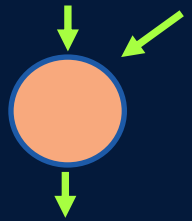


```
PCollection<KV<String, Price>> prices
   = pipeline
     .apply("bounded-price-source", Read.from(new MyPriceSource()))
     .apply("price-string-to-entry", ParDo.of(new MyPriceTransform()));
```

Demo – “VaR” (?)

An outline of VaR

(1) Take all portfolios
(2) Take all prices
(3) For all combinations



```
PCollectionView<Map<String,Price>> pricesView = prices
        .apply("price-entry-viewmap", View.asMap());

PCollection<Valuation> joined
    = account.apply("join",
        ParDo.of(new MyJoinTransform(pricesView)).withSideInputs(pricesView));
```

Like an outer join

intel

hazelcast

Demo – “VaR” (?)

An outline of VaR

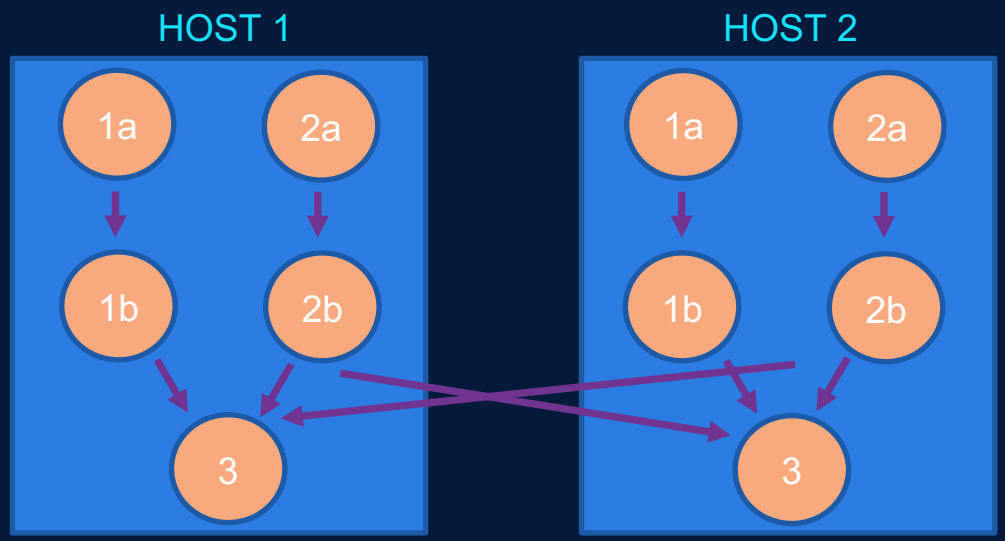
(1) Take all portfolios
(2) Take all prices
(3) For all combinations
(4) Simulate a range of possible movements .. Historical returns, Monte Carlo
(5) Select 5% worst, 1% worst, whatever

This is not a “for loop”, it is a cartesian product
== > RUNNING IN PARALLEL

Demo – “VaR” (?)

Two hosts….

(1) Take all portfolios
(2) Take all prices
(3) For all combinations



This is not a “for loop”, it is a cartesian product
== > RUNNING IN PARALLEL

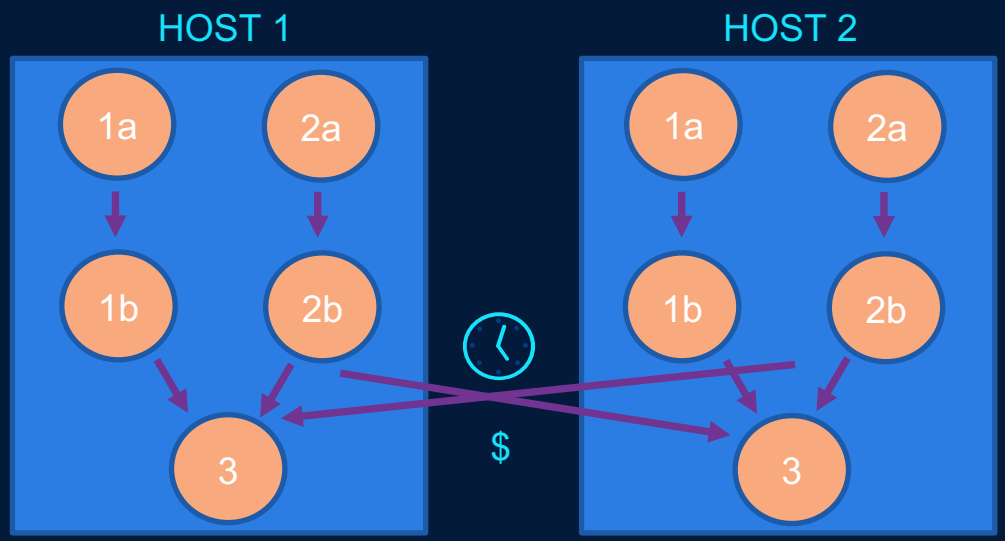
intel

hazelcast

Demo – “VaR” (?)

Two hosts….

(1) Take all portfolios
(2) Take all prices
(3) For all combinations



This is not a “for loop”, it is a cartesian product
== > RUNNING IN PARALLEL

hazelcast

intel

Demo

hazelcast

intel

Demo

Hazelcast & Intel

Hazelcast



An In-Memory Data Grid with a stream processing engine (Jet)

The data grid is collection of Java processes caching data in memory

Hazelcast provides reliability, securability, scalability, performance
Widely used by most of the world's large organisations on cloud or premise
Multiple client connectors – Java, Python, C++, C#, GoLang, Node.js..

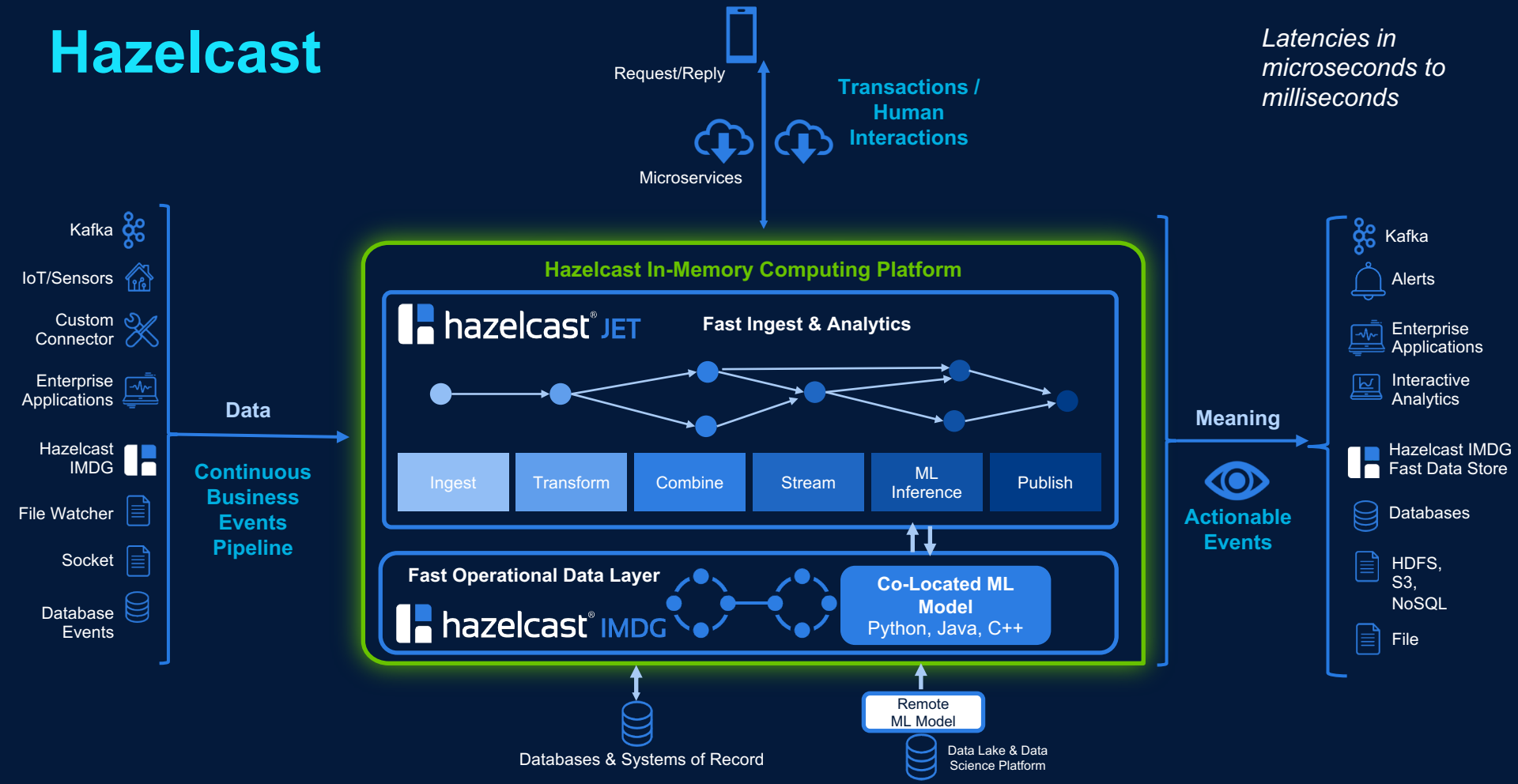
(https://upload.wikimedia.org/wikipedia/commons/9/94/Filing_cabinet_icon.svg)

intel

hazelcast

Hazelcast

Latencies in microseconds to milliseconds



intel

hazelcast

Intel Optane Persistent Memory

Hybrid memory

Packaged as a DIMM, can use as memory
- up to 512GB, larger than typical DRAM cards
- a little slower than DRAM but wins on cost/GB

Can use as storage
- 6x faster than SSD
  - why is this relevant to Hazelcast, an in-memory data grid?

+ Hardware encryption!

Intel Optane Persistent Memory

Using Optane for save/restore file storage

Reload time for data:

87GB in 19 seconds

(5.5 million records, 16Kb each + metadata)

Intel Optane Persistent Memory

intel OPTANE DC PERSISTENT MEMORY

Using Optane for save/restore file storage

Reload time for data:

87GB in 19 seconds

12 node cluster, 1TB of data

Still 19 seconds!

(+ integrity cross-checks)

intel hazelcast

Intel Optane Persistent Memory

intel OPTANE DC PERSISTENT MEMORY

Using Optane for save/restore file storage

In the real world…

A few minutes for off, on, verify, ready
an empty cache
+ however long to reload from legacy
before useable

Now…

Same few minutes, but now can have 1TB of data!

intel hazelcast

Intel Optane Persistent Memory

intel OPTANE DC PERSISTENT MEMORY

Using Optane for save/restore file storage

In the real world…

A few minutes for off, on, verify, ready an empty cache
+ however long to reload from legacy before useable

Now…

Same few minutes, but now can have 1TB of data! Or 2TB!

intel hazelcast

Intel Optane Persistent Memory

intel OPTANE DC PERSISTENT MEMORY

Using Optane for save/restore file storage

In the real world…

A few minutes for off, on, verify, ready an empty cache
+ however long to reload from legacy before useable

Now…

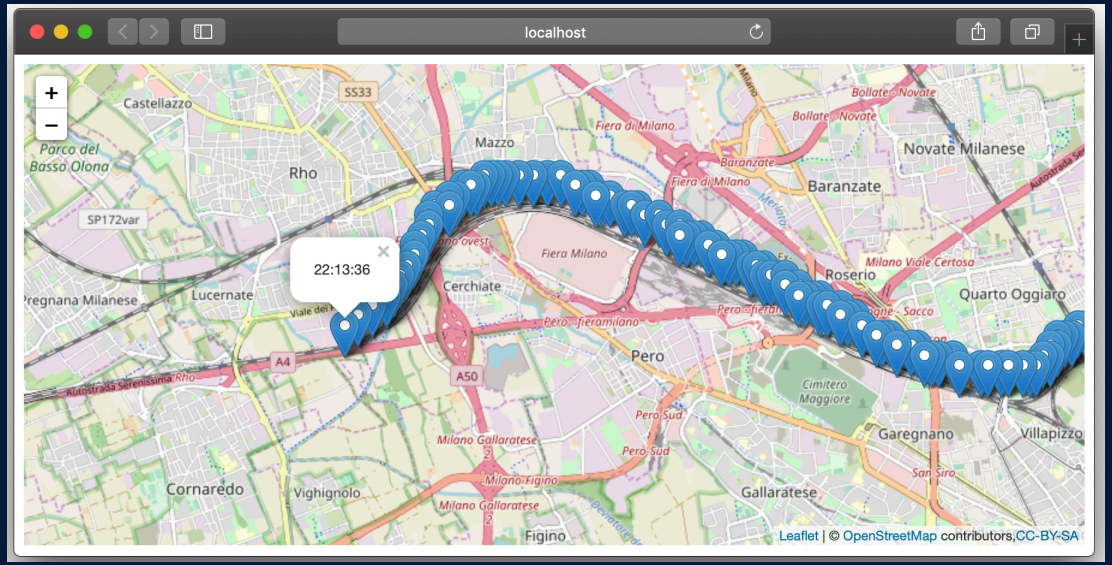
Same few minutes, but now can have 1TB of data! Or 2TB! Or 4TB!...etc

intel hazelcast

Hazelcast

Want to try it ?

A Hazelcast & Beam example, tracking the movement of a train from a stream of GPS points



Look for “Train Track” on https://github.com/hazelcast/hazelcast-jet-demos

intel

hazelcast

hazelcast

intel

Summary

Summary

Imagine your Batch as a Stream, one day it may be

Performance of Beam runners and Beam overhead may matter

Not all Beam runners have complete implementations, may matter

Monitoring/metrics are vital, debugging is difficult

Senior developers only: deplete > enrich

Hazelcast & Intel, other runners are available

Thank you