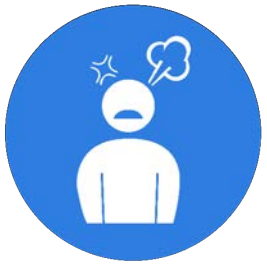
Low Latency Applications

Open Source In Memory Computing Platform

Is Your Business Experiencing These Issues?

Slow websites

Overloaded/crashing web services



Slow customer-facing applications

Large mainframe costs

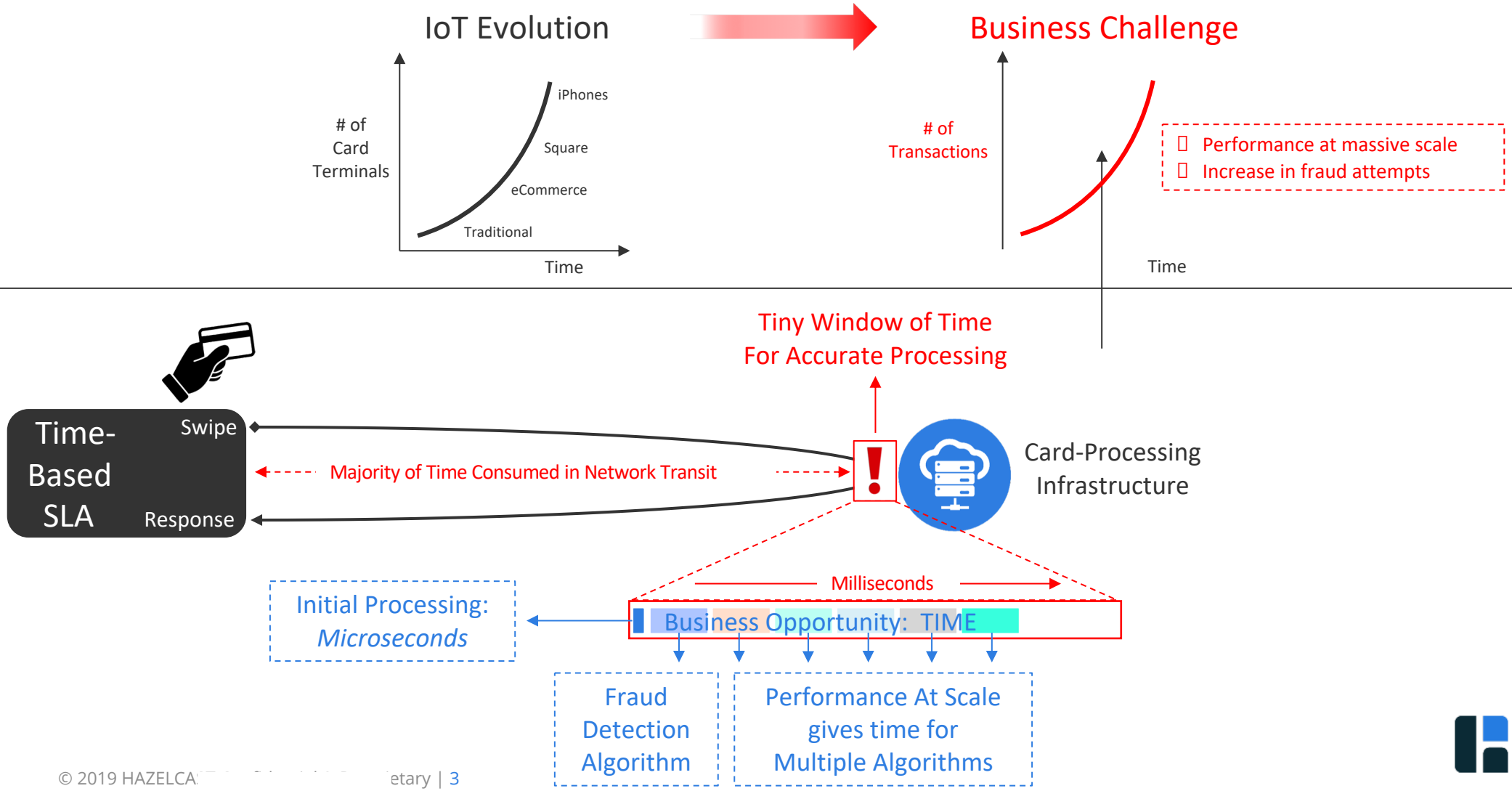
Adding hardware with little effect

New IT architecture projects running slow/missing SLAs

hazelcast

The Hazelcast Difference

Example: Credit Card Processing



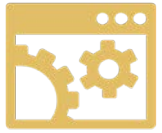
Business Challenges Solved

Latency & Speed

Time is money

Scalability

Hazelcast scales effortlessly responding to peaks, valleys for optimal utilization



Real-Time, Continuous Intelligence

Real-time view of constantly changing operational data

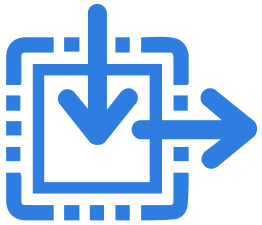
Zero Downtime

Built for high resiliency

hazelcast

Data Grid Use Cases

Caching

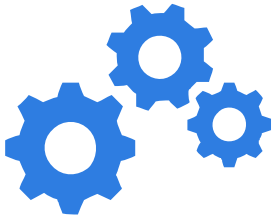


- High-Density Memory Store, client and member
- Full JCache support
- Elastic scalability
- Super fast
- High availability
- Fault tolerant
- Cloud ready

In-Memory Data Grid

- Simple, modern APIs
- Distributed data structures
- Distributed compute
- Distributed clustering
- Object-oriented and non-relational
- Elastic and scalable
- Transparent database integration
- Client-server and/or embedded architecture

Microservices Infrastructure



- Isolation of Services with many, small clusters for easier troubleshooting & maintenance
- Service registry
- Multiple network discovery mechanisms
- Inter-process messaging
- Fully embeddable
- Resilient and flexible

Web Session Clustering

- Seamless failover between user sessions
- High performance
- No application alteration
- Easy scale-out
- Fast session access
- Offload to existing cluster
- Tomcat, Jetty + any Web Container
- Works efficiently with large session objects using delta updates

hazelcast

Hazelcast - High Performance Platform

Mobile

Apps

Social

Commerce

Communities

IMDG In-Memory Data Grid

Integrate APIs, Microservices, Notifications	**Communicate** Serialization, Protocols
Store/Update Caching, CRUD Persistence	**Compute** Query, Process, Execute
Scale Clustering & Cloud, High Density	**Replicate** WAN Replication, Partitioning
Secure Privacy, Authentication, Authorization	**Available** Rolling Upgrades, Hot Restart

Jet In-Memory Streams

Ingest & Transform Events, Connectors, Filtering	**Combine** Join, Enrich, Group, Aggregate
Stream Windowing, Event-Time Processing	**Compute & Act** Distributed & Parallel Computations
Secure Privacy, Authentication, Authorization	**Available** Job Elasticity, Graceful shutdown

Management Center

Secure | Manage | Operate
Embeddable | Scalable | Low-Latency
Secure | Resilient | Distributed

Analytics
Visualization
Data Lake

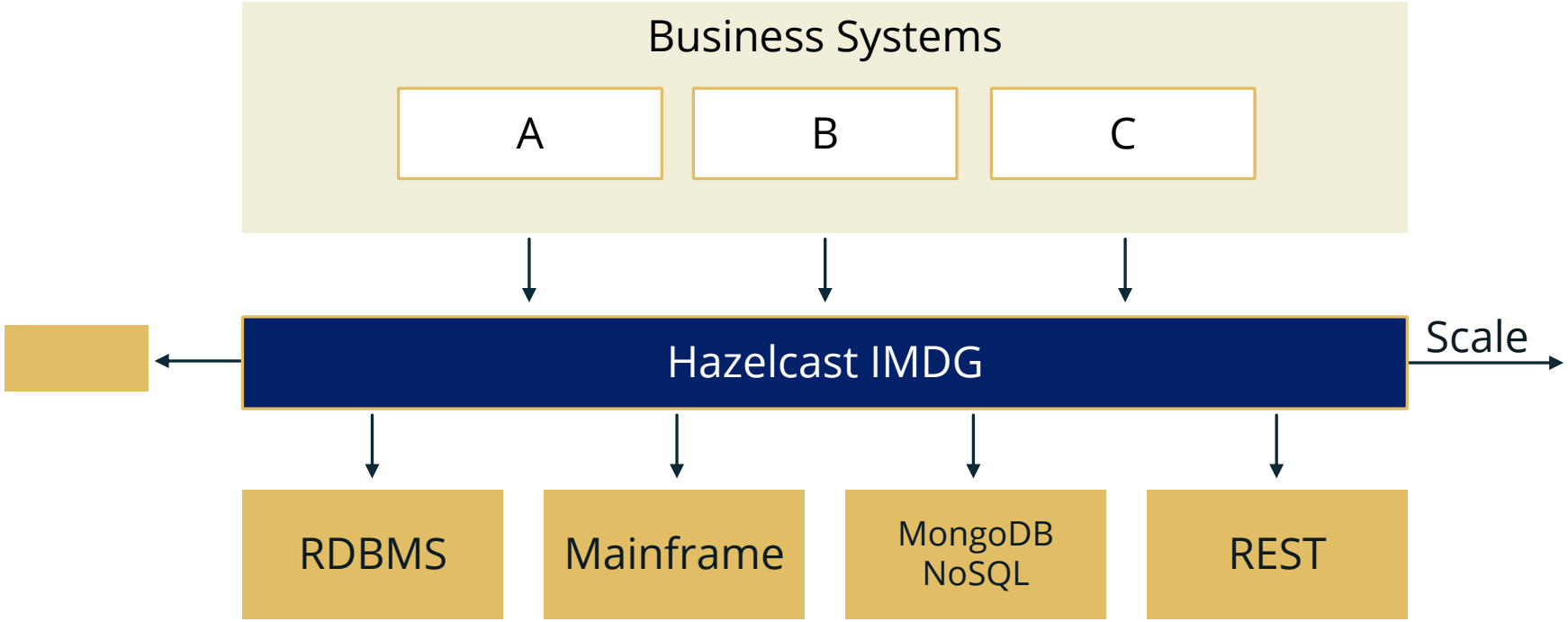
Live Streams
Kafka, JMS,
Sensors, Feeds

Databases
JDBC,
Relational, NoSQL,
Change Events

Files
HDFS, Flat Files,
Logs, File watcher

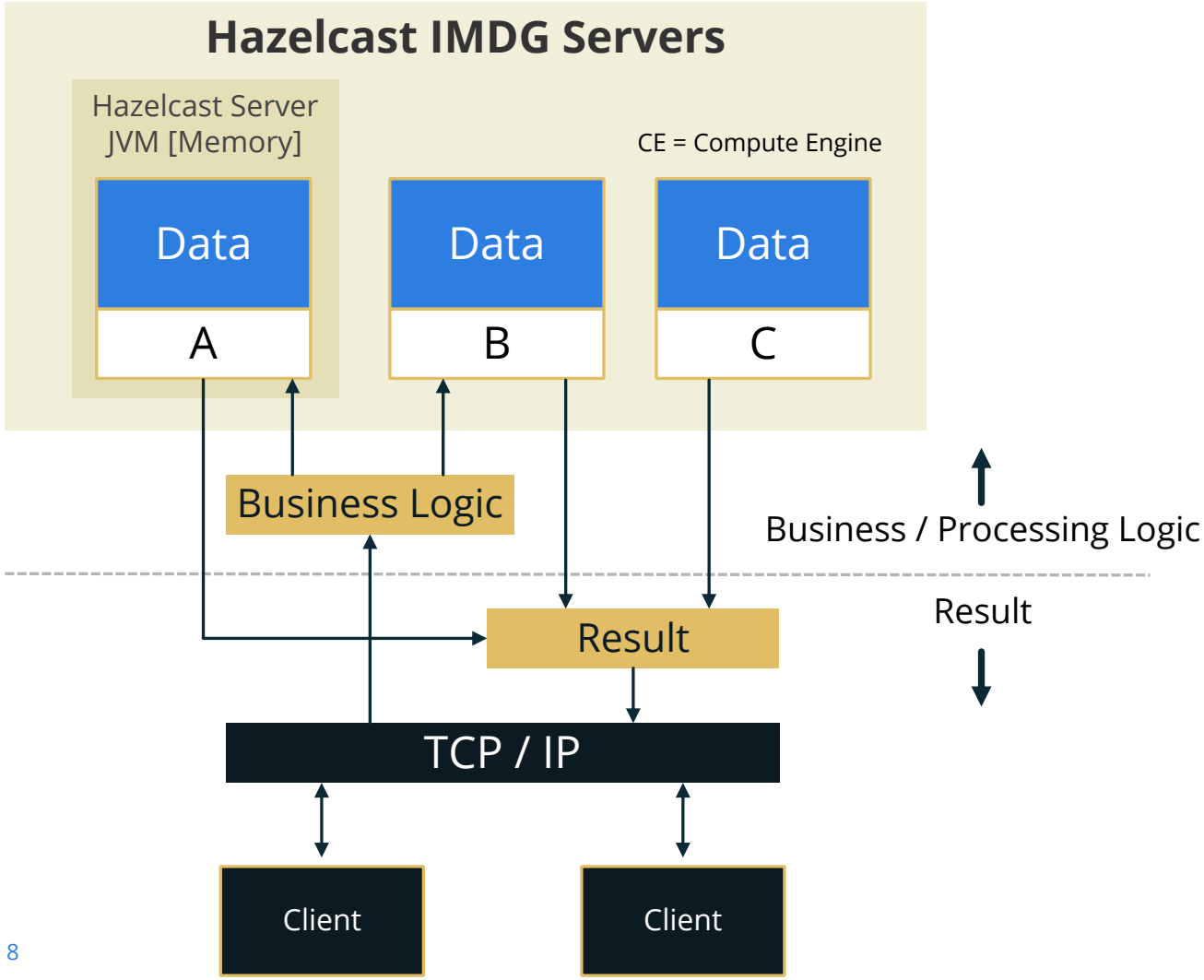
Applications
Sockets

Technical Use Cases: Cache in Front of a Data Store

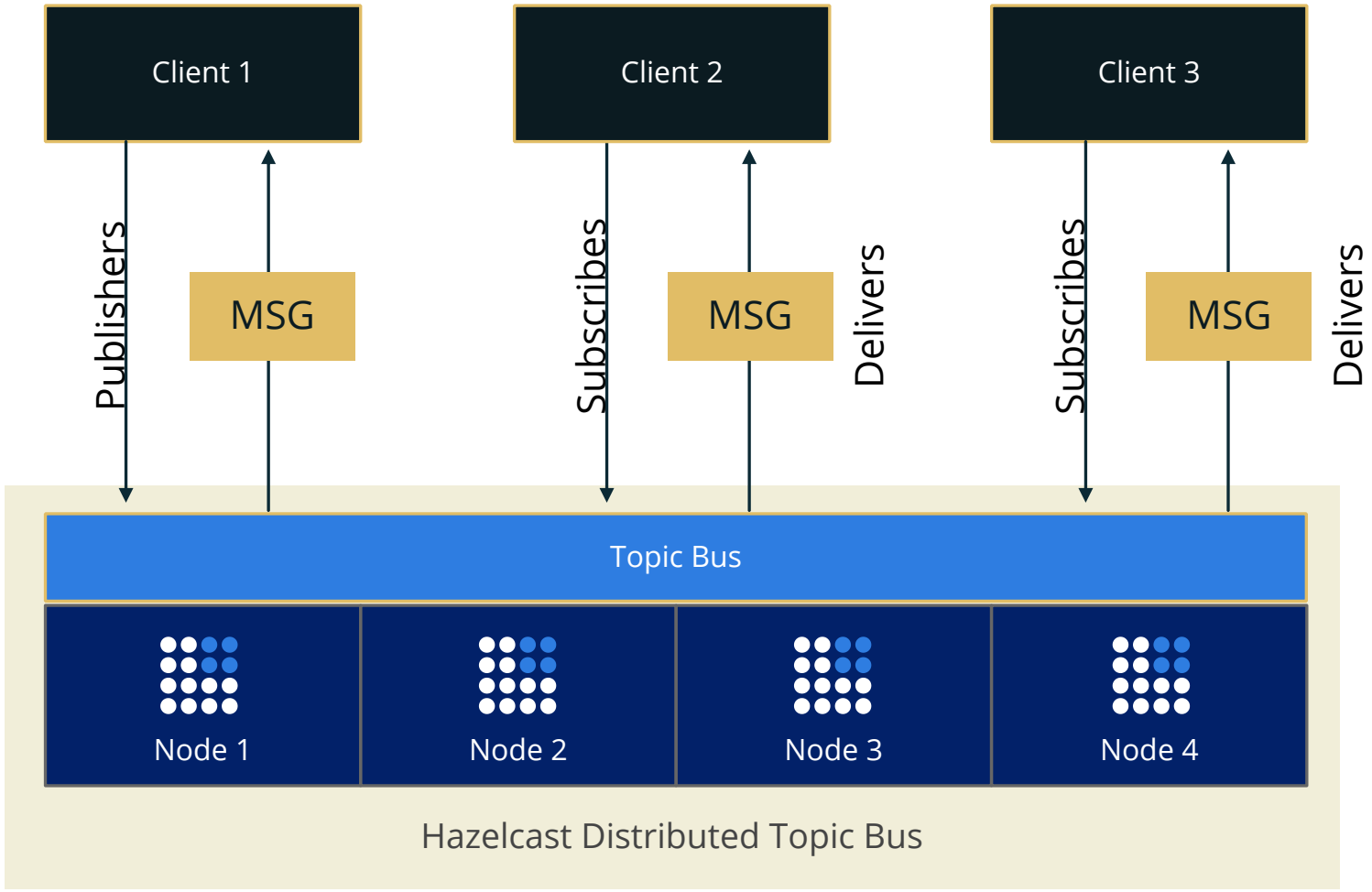


hazelcast

Technical Use Cases: In-Memory Data Grid Compute

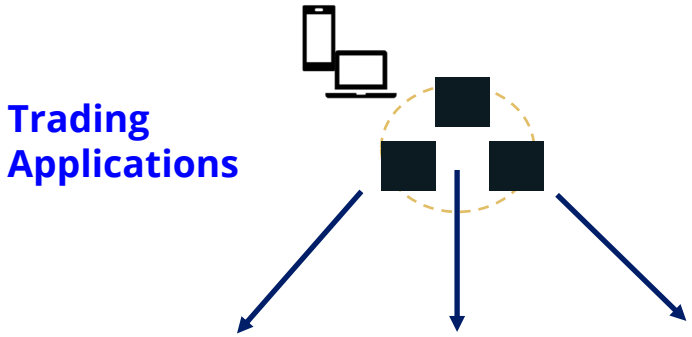


Technical Use Cases: In-Memory Data Grid Messaging

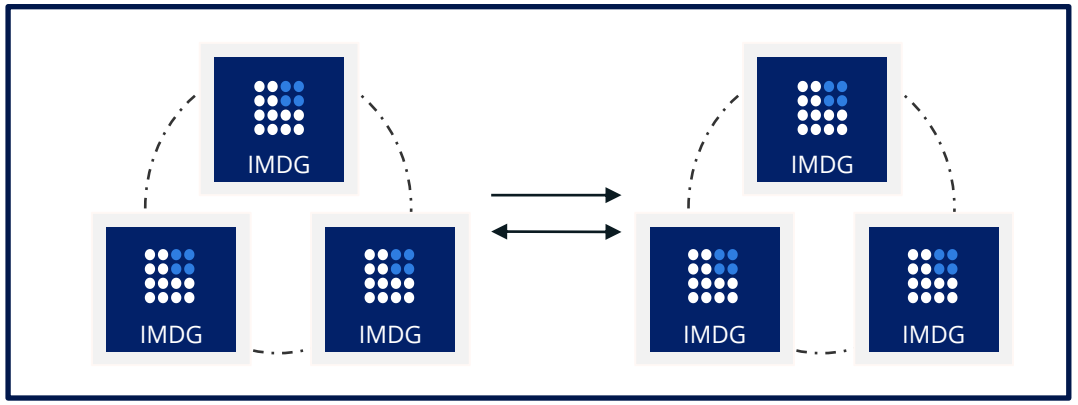


Proof Points – Agile High-Speed Trading

- Low-latency data grid for fast access to market data, positions, etc.
- Low latency, data-aware compute on elastic grid.
- Distributed low-latency calculation of prices, risks, etc.



Unified Data & Compute Grid



MIZUHO Deutsche Bank CREDIT SUISSE

DOW JONES London Stock Exchange JPMORGAN CHASE & CO.

HSBC National Australia Bank

HSBC – FX Quotation Systems

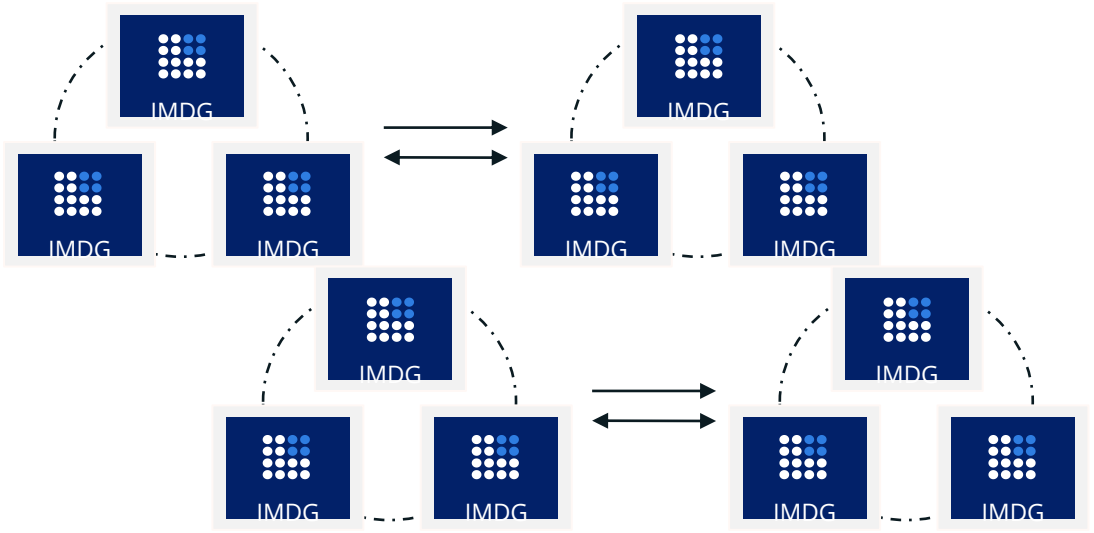
- Sub-millisecond access, off heap data to eliminate garbage collection
- Fast distributed calculations of prices, margins and quotations
- Ensure zero-downtime SLA

National Australia Bank – Financial Market Data

- More predictable/accurate derived calculations with single source of market data
- Stable and always-on gateway access – allowing more concurrent system users, more quickly

Proof Points – Zero-Downtime Business

Cross-cluster replication across geographies
Globally available transaction data with millisecond response
Low-latency data-aware compute on elastic grid
Elastic scalability to support peak loads during extreme spikes

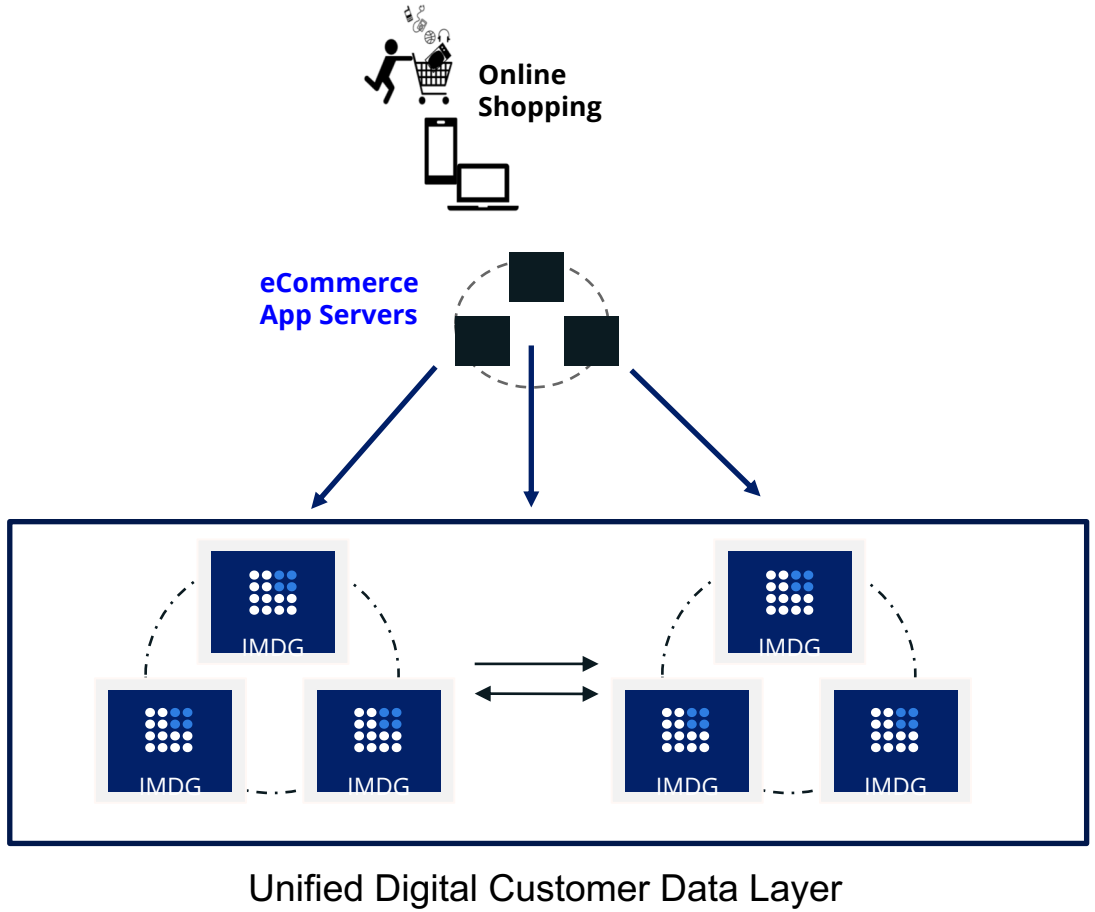


Capital One
Store 2TB of customer data and synch geographically
20K+ tps distributed compute with under 1ms latency
99.999% uptime architecture

Visa
Meets SLA: 10,000 TPS with SSL
99.999% up-time and 2-3X faster than Redis

Proof Points - Online Store - Retail/Tech

Cross-cluster replication across geographies
Globally available online store data with millisecond response
Elastic scalability to support peak loads during extreme spikes
De-couple online store from back-ends for maximum resilience



Unified Digital Customer Data Layer

Apple

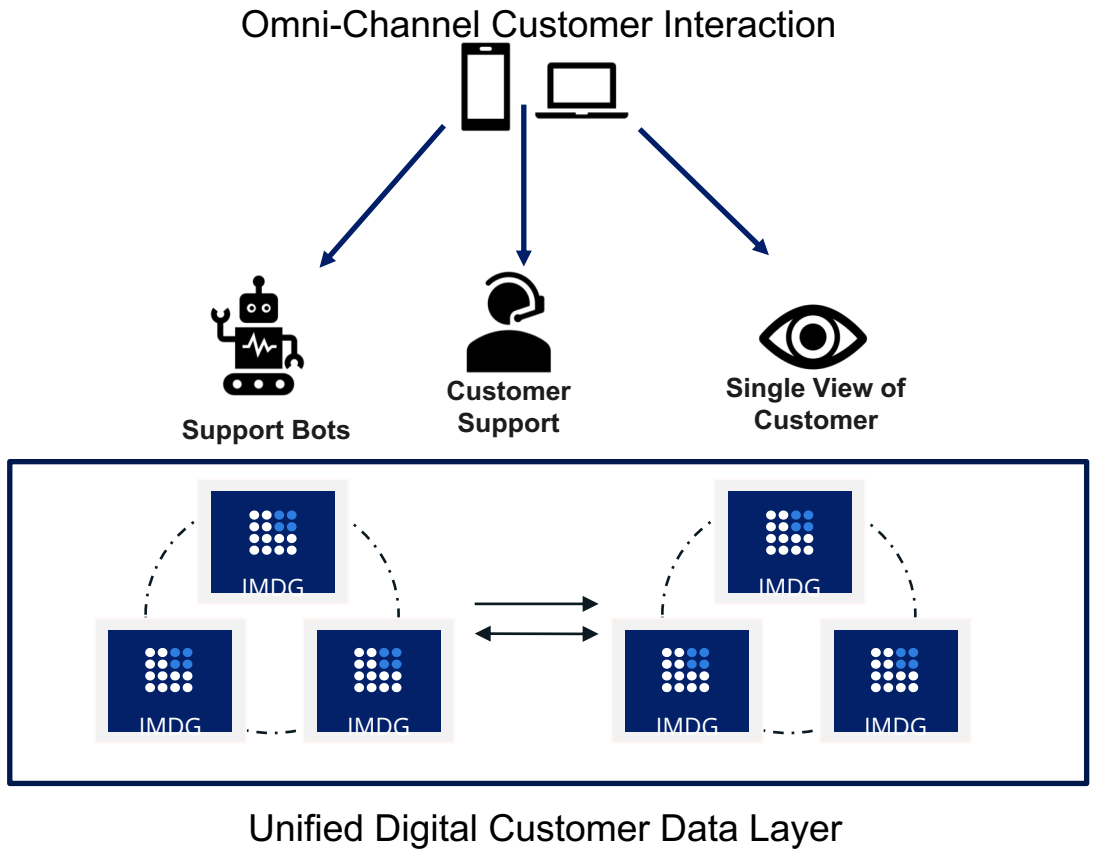
- Time to report accurate order delivery date from 30 mins to 7 secs
- 1.2ms max application latency
- Ensure zero-downtime SLA for new iPhone introductions

Target

- Removed performance bottleneck for Apache Cassandra system of record – latency reduced from 300ms to ~2ms
- Exceeds SLA target of 40ms and scales elastically to meet seasonal events like Black Friday, Cyber Monday

Proof Points - Customer Visibility - Telco/Media

Cross-cluster replication across geographies
Globally available Customer data with millisecond response
Elastic scalability to support peak loads during extreme spikes
De-couple customer sites from back-ends for maximum resilience



Comcast:
Captures viewing and account history, service engagements, location data;
Used to create an integrated enriched view which is the basis for an **AI-driven engagement** on customer call-in

Traditional View of Big Data and Data Science:

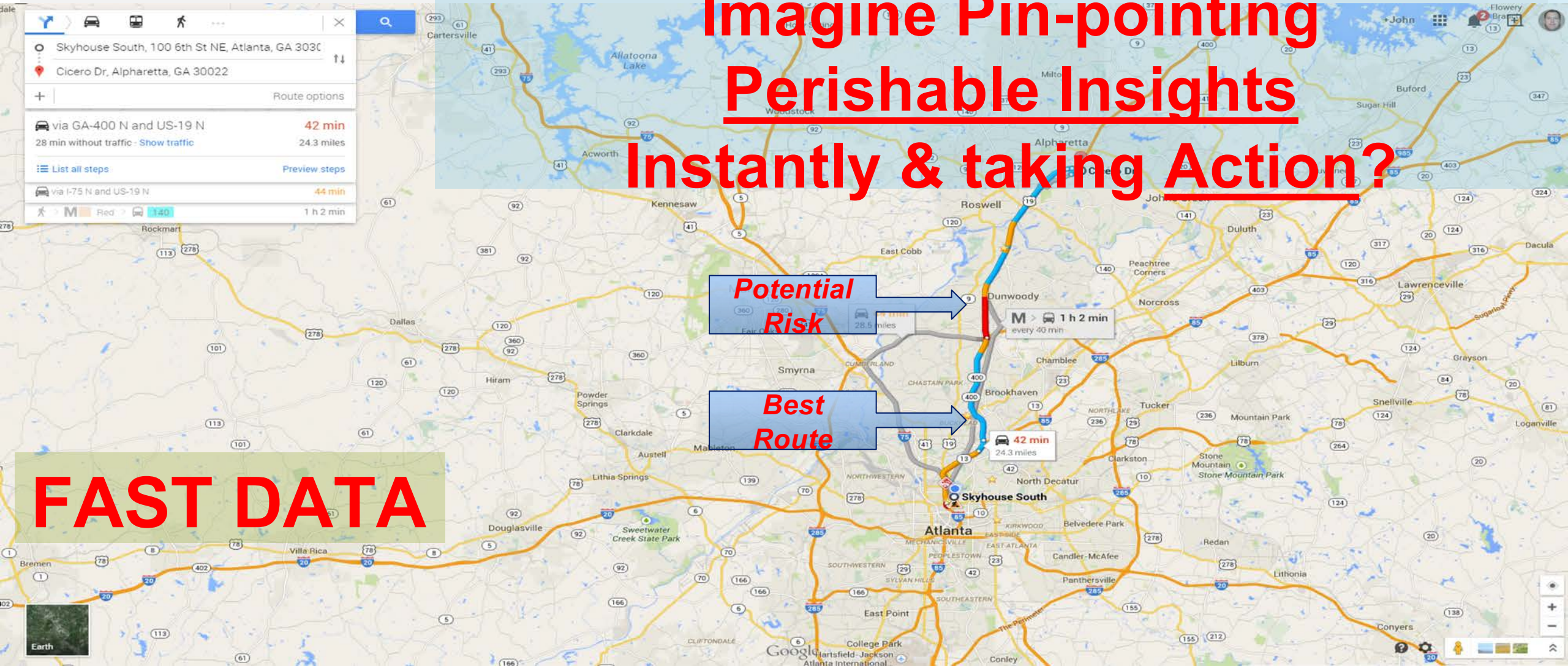
"We Have Mountains of Data"

and: *"There's GOLD in Them Thar Hills!"*

*You just have to dedicate **Massive Computing Resources** & **Teams of Data Scientists** to identify nuggets of insight within a matter of **Days or Hours.***

SLOW DATA

Imagine Pin-pointing Perishable Insights Instantly & taking Action?



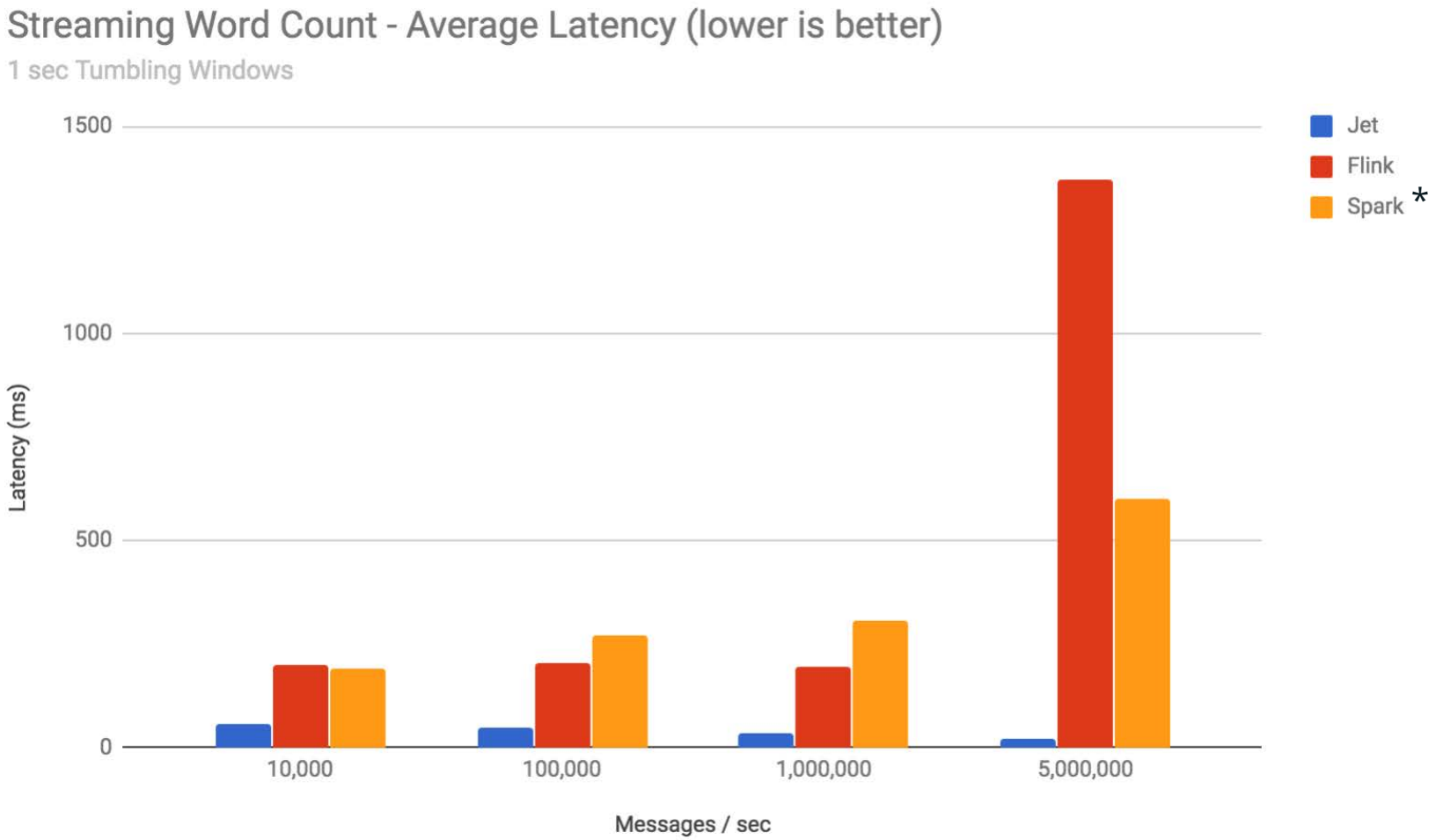
FAST DATA

Evolution of Stream Processing

 1st Gen (2000s) Hadoop(batch) or Apama(CEP) ***hard choices***	**Distributed Batch Compute** – MapReduce – scaled, parallelized, distributed, resilient, - not real-time or **Siloed, Real-time** – Complex Event Processing – specialized languages, not resilient, not distributed(single instance), hard to scale, fast, but brittle, proprietary
 2nd Gen (2014) Spark ***hard to manage***	**Micro-batch distributed** – heavy weight, complex to manage, not elastic, require large dedicated environments with many moving parts, not Cloud-friendly, not low-latency
 3rd Gen (2017 Jet & Flink) **flexible & scalable** **True "Fast Data"**	**Distributed, real-time streaming** – highly parallel, true streams, advanced techniques (Directed Acyclic Graph) enabling reliable distributed job execution Flexible deployment - Cloud-native, elastic, embeddable, light-weight, supports serverless, fog & edge. Low-latency Streaming, ETL, and fast-batch processing, built on proven data grid

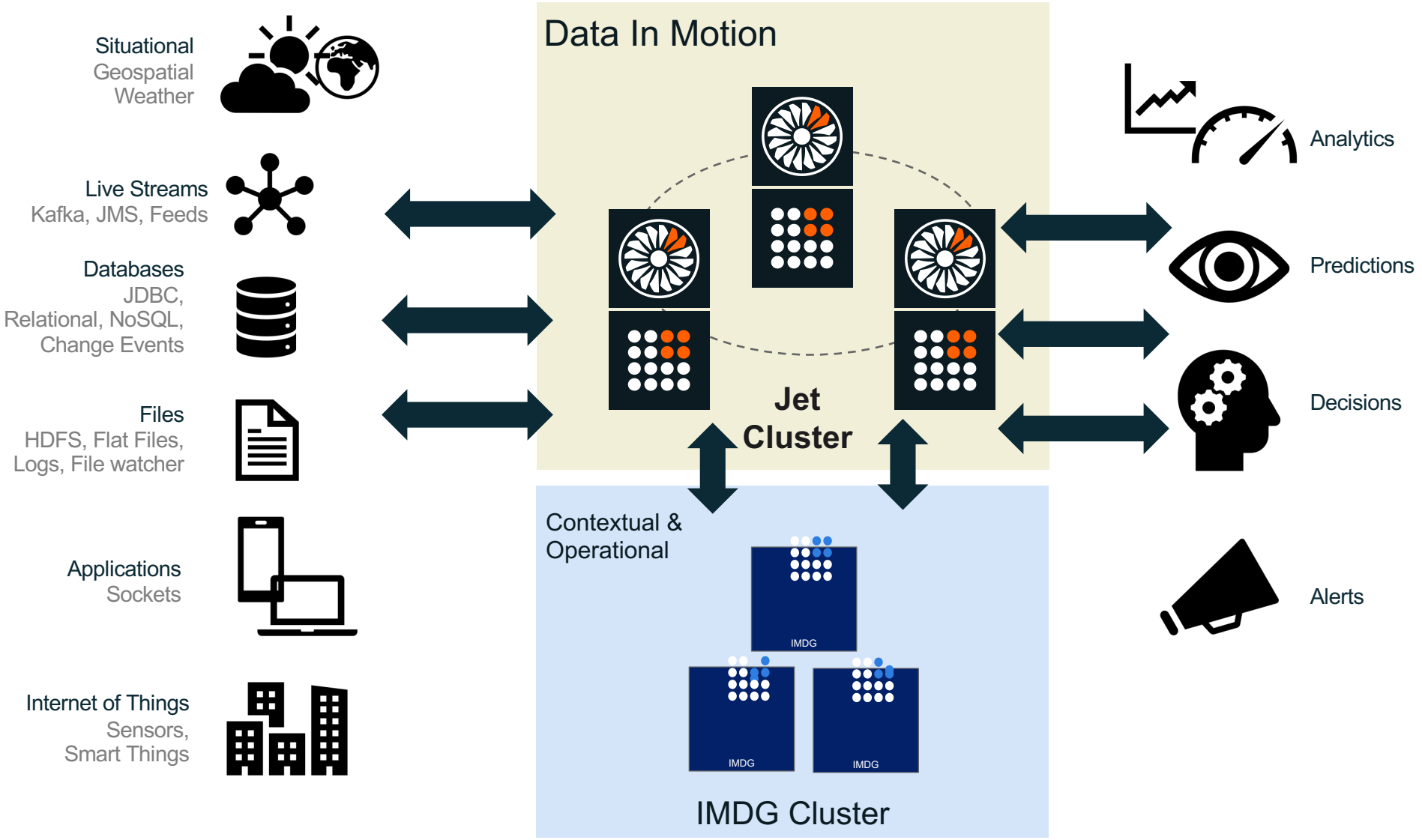
hazelcast

Streaming Performance



* Spark had all performance options, including Tungsten, turned on

Stream Processing



In-Memory Distributed Stream Processing Use-Cases

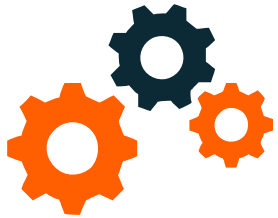
Real-time Stream processing

- Big Data in near real-time
- Distributed, in-memory computation
- Aggregating, joining multiple sources, filtering, transforming, enriching
- Elastic scalability
- Super fast
- High availability
- Fault tolerant

ETL/Ingest

- Supports common sources such as HDFS, File, Directory, Sockets
- Custom sources can be easily created
- Batch and streaming
- Streaming ingest from Oracle, SQL Server, MySQL using Striim
- Sink to Hazelcast or other operational data stores

Data-Processing Microservices



- Data-processing microservices
- Isolation of services with many, small clusters
- Service registry
- Network discovery
- Inter-process messaging
- Fully embeddable
- Spring Cloud, Boot Data Services

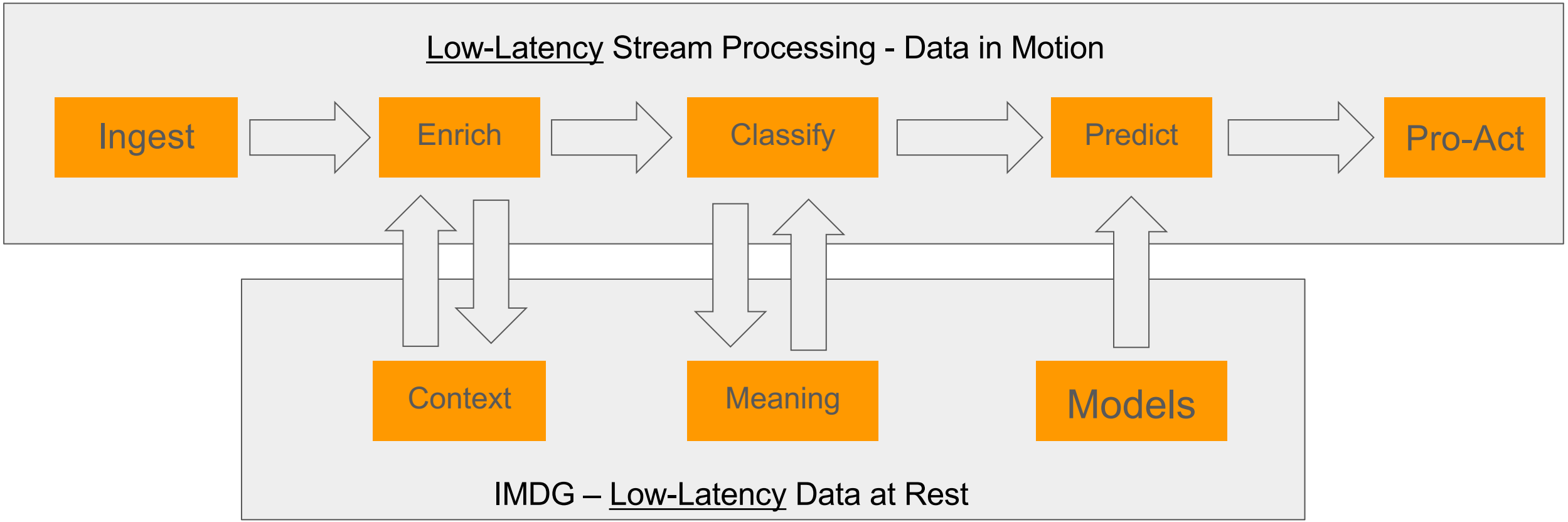
Edge Processing

- Low-latency analytics and decision making
- Saves bandwidth and keeps data private by processing it locally
- Lightweight – runs on restricted hardware
- Both processing and storage
- Fully embeddable for simple packaging
- Zero dependencies for simple deployment

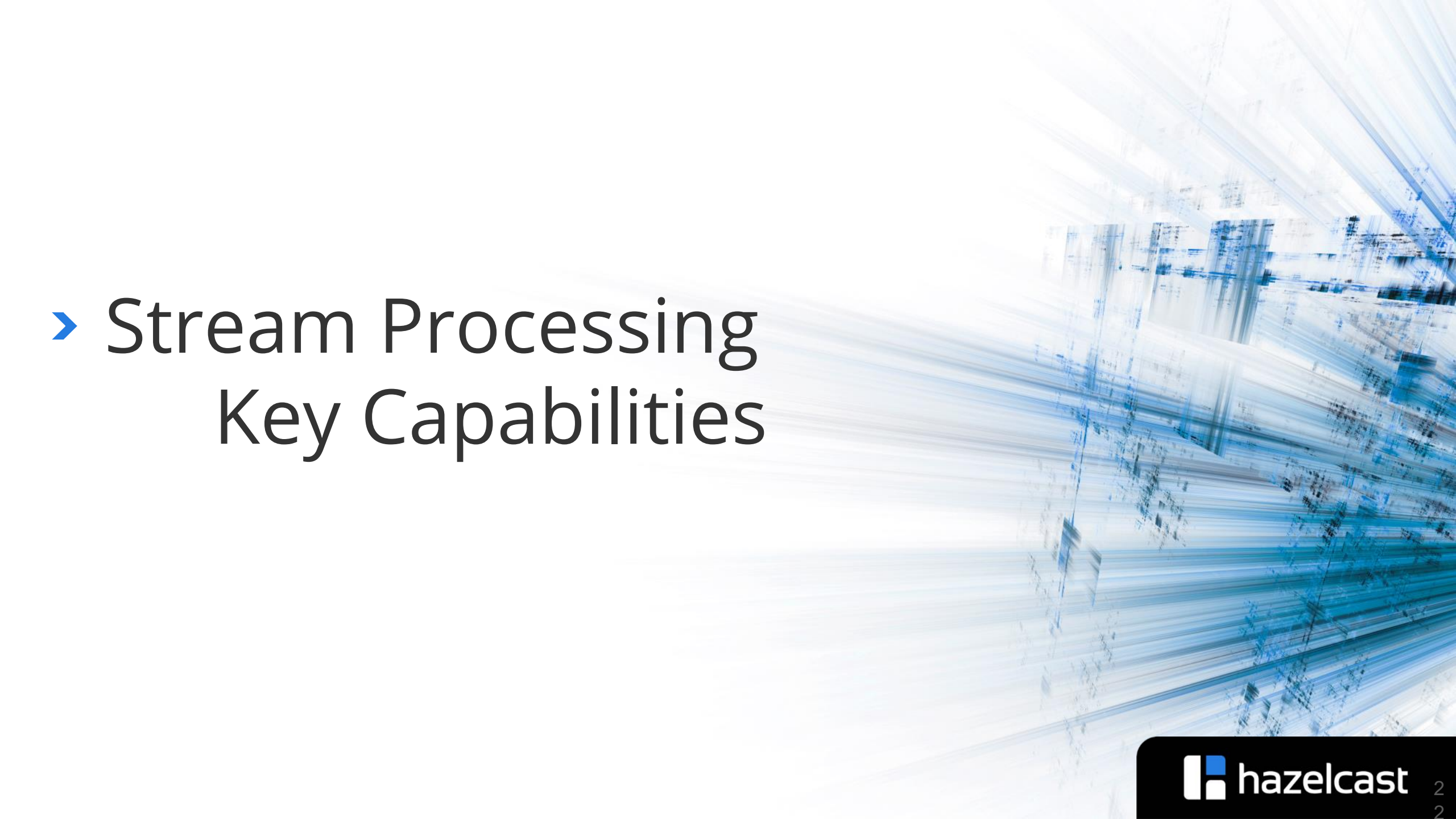
Example - Stream Processing with Machine Learning

Move from Reactive to Pro-Active

Taking Action before negative impact or ahead of opportunity

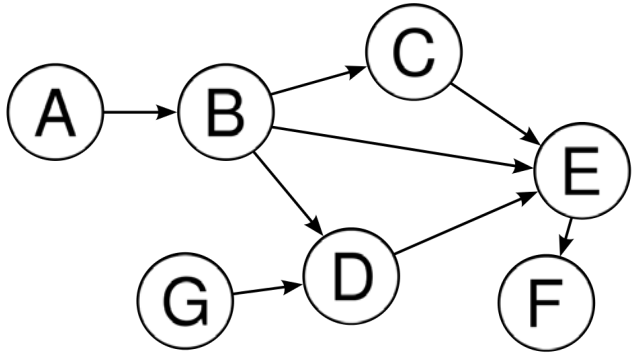


Stream Processing Key Capabilities



Directed Acyclic Graphs

- **Directed Acyclic Graphs** are used to model computations



- Each vertex is a step in the computation
- It is a generalisation of the MapReduce paradigm
- Supports both batch and stream processing
- Other systems that use DAGs: Apache Tez, Flink, Spark, Storm…

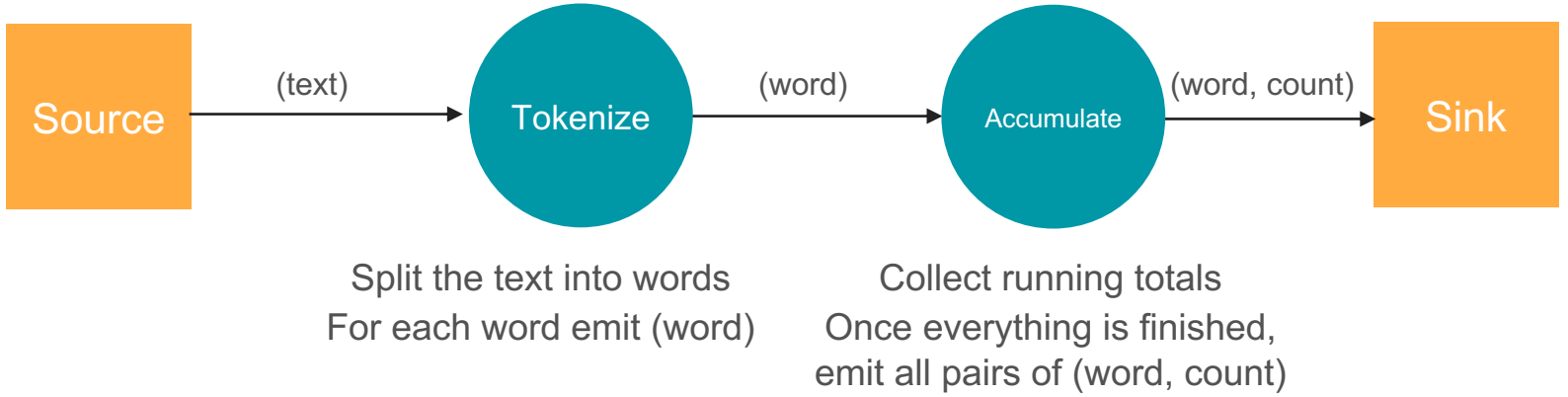
Example: Word Count

- Naïve, single threaded world:

1. Iterate through all the lines
2. Split the line into words
3. Update running total of counts with each word

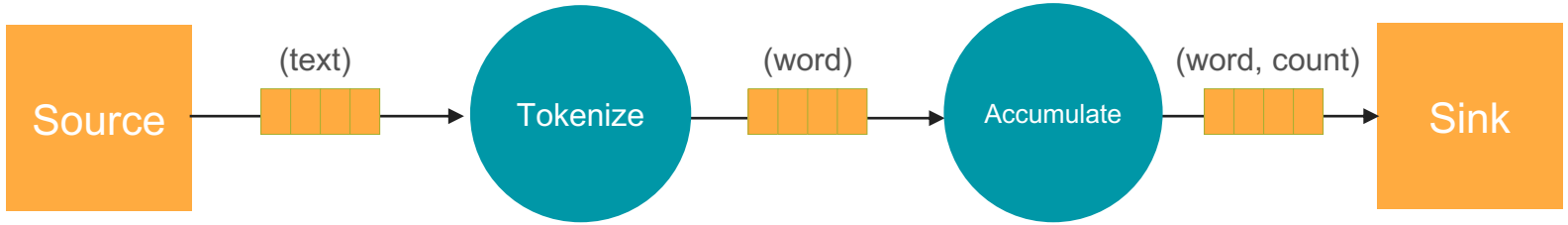
```
final String text = "...";
final Pattern pattern = Pattern.compile("\\s+");
final Map<String, Long> counts = new HashMap<>();

for (String word : pattern.split(text)) {
    counts.compute(word, (w, c) -> c == null ? 1L : c + 1);
}
```

We can represent the computation as a **DAG**



Still single-threaded execution:
each Vertex is executed in turn sequentially,
wasting the CPU cores

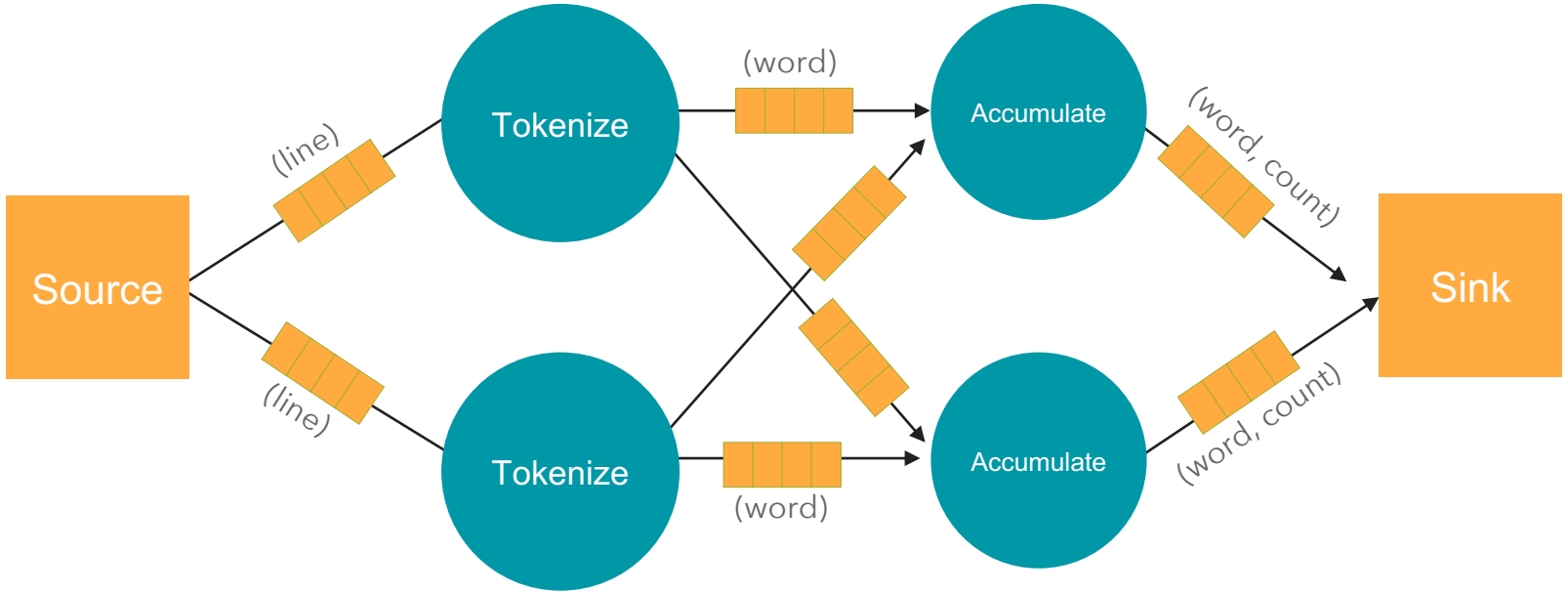
By introducing **concurrent queues** between the vertices we enable each vertex to run concurrently



Split the text into words
For each word emit (word)

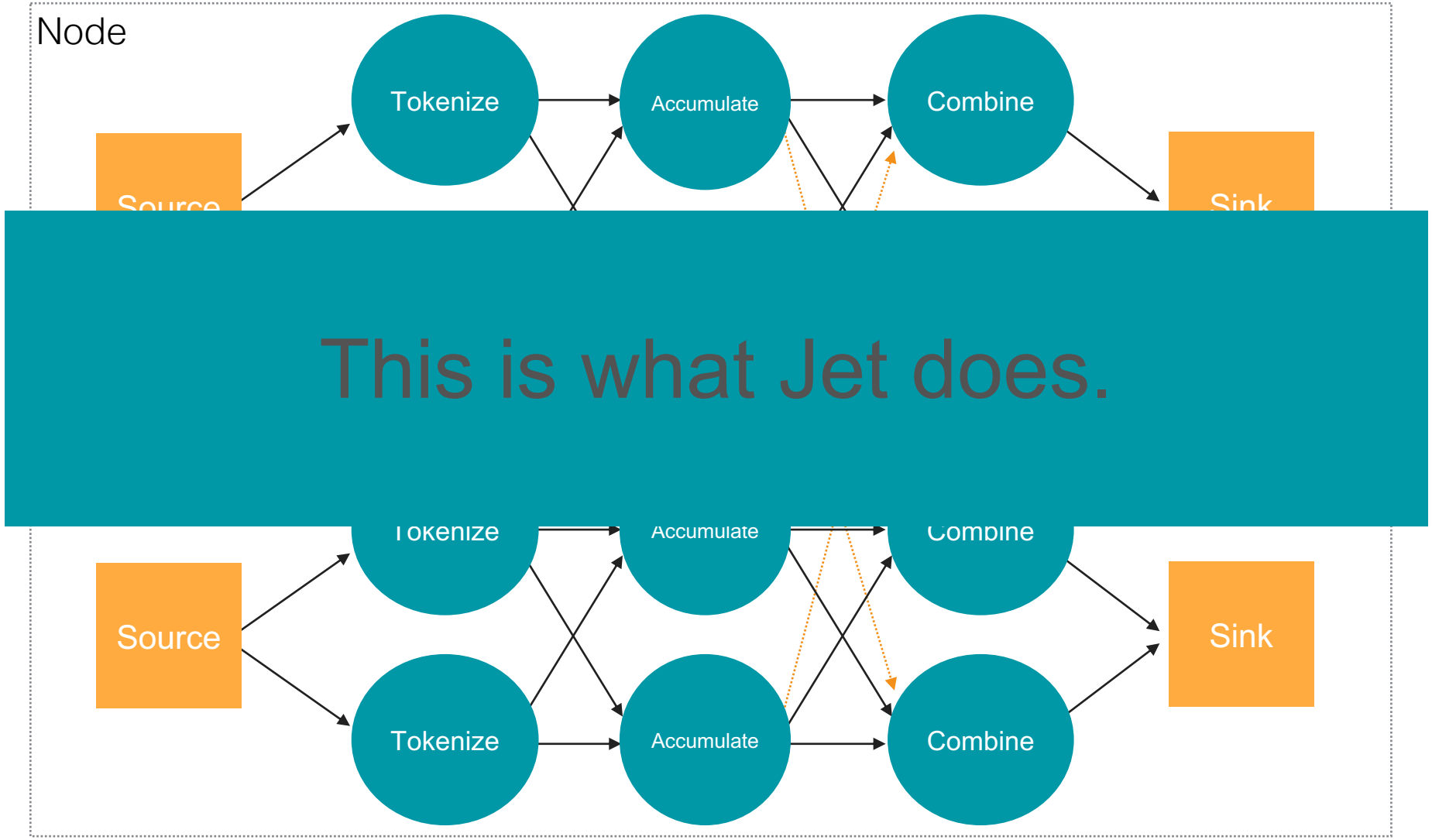
Collect running totals.
Once everything is finished,
emit all pairs of (word, count)

The Accumulator vertex can also be executed in parallel by **partitioning** the accumulation step by the individual words.



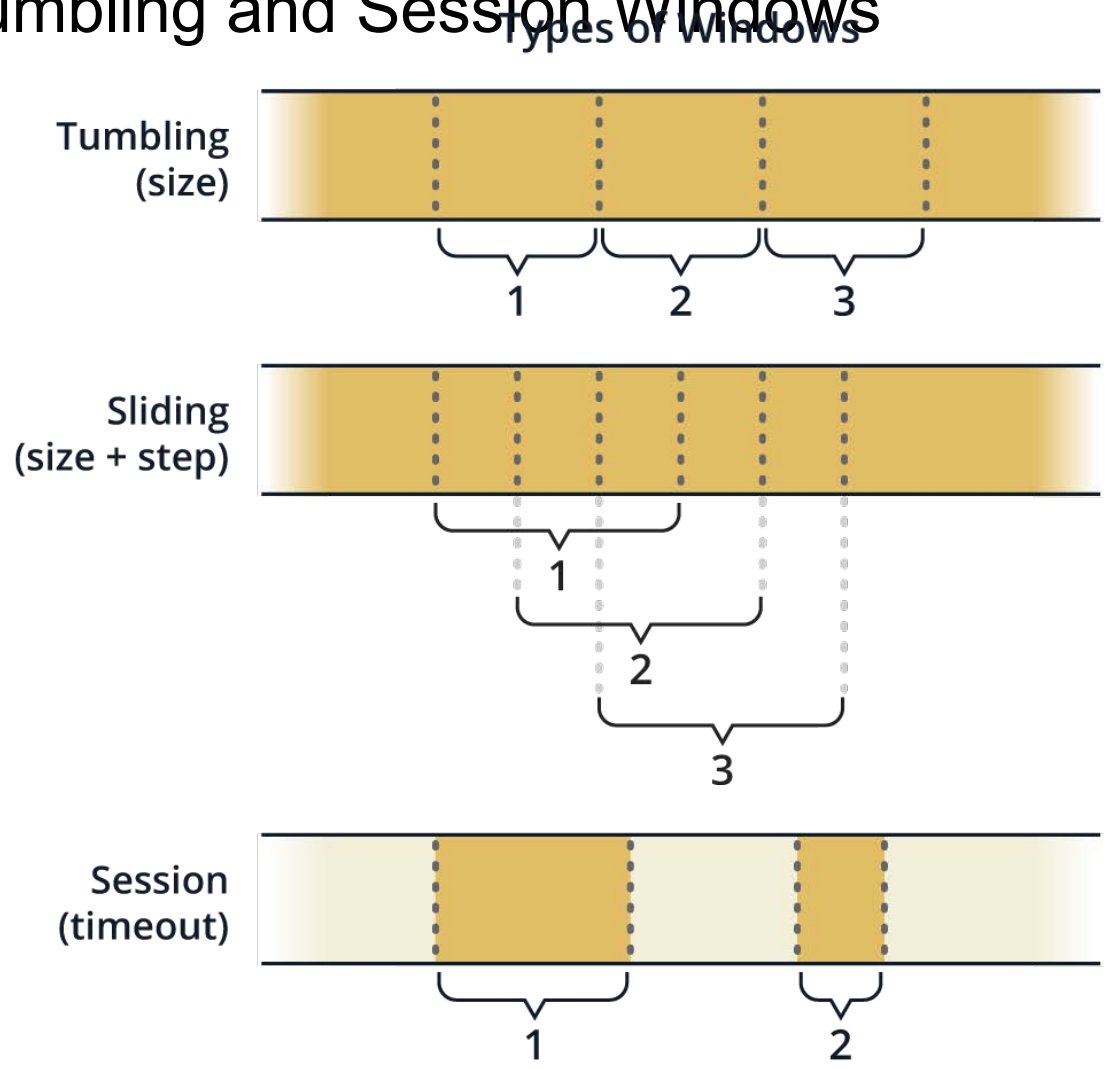
We only need to ensure the **same** words go to the **same** Accumulator.

The steps can also be distributed across multiple nodes. To do this you need a distributed **partitioning** scheme.

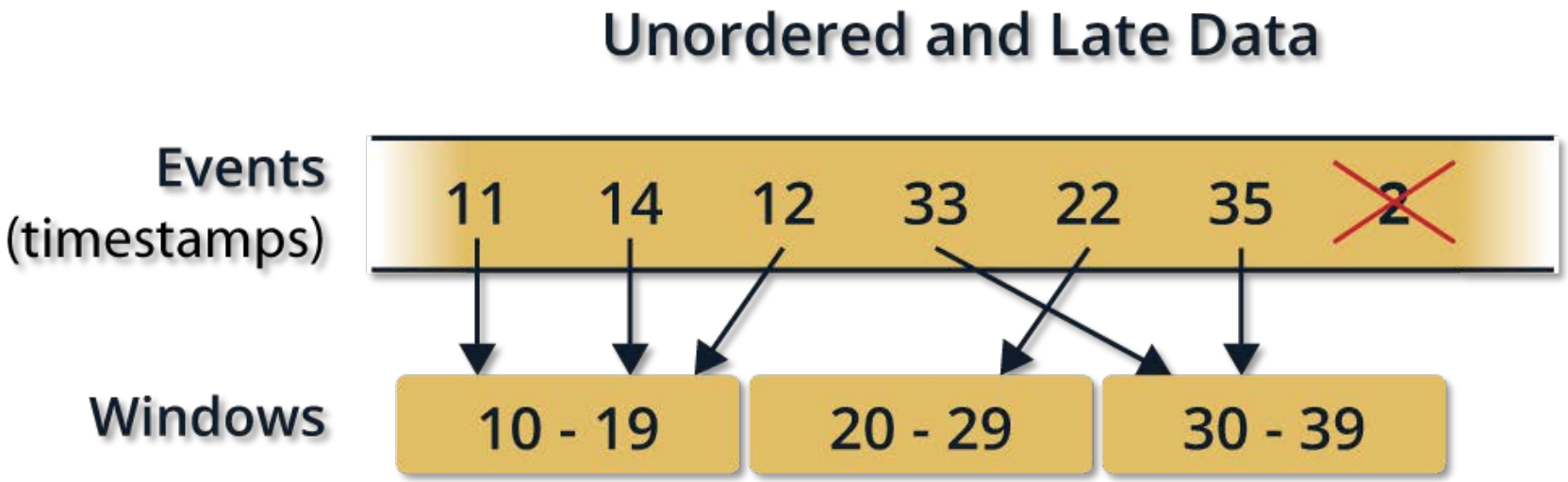


Key to Stream Processing – windows

Sliding, Tumbling and Session Windows



Unordered and Late Data Handling



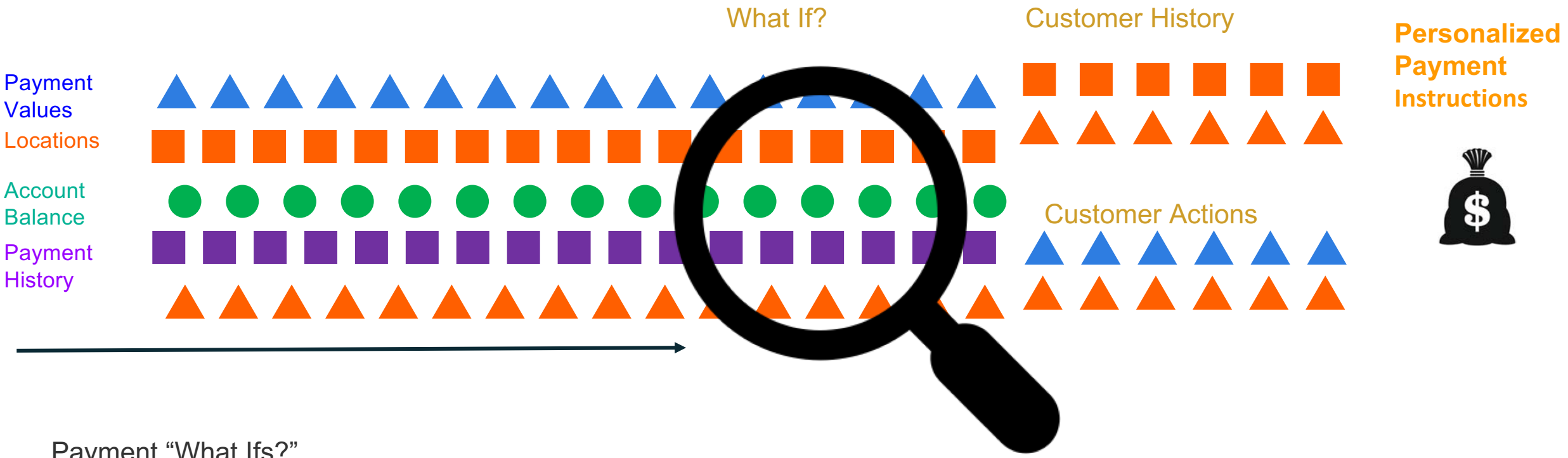
hazelcast

Job Elasticity

- Jobs are elastic – they can dynamically scale to make use of all available members, following cluster topology changes
- Job state and lifecycle are saved to IMDG IMaps and benefit from their performance, resilience, scale and persistence
- Automatic re-execution of part of the job in the event of a failed worker
- Tolerant of loss of nodes; missing work will be recovered from last snapshot and re-executed
- Cluster can be scaled without interrupting jobs – jobs benefit from the increased capacity
- State and snapshots can be persisted to resume after cluster restart (Version 3.0)

hazelcast

Stream Processing Use-Case – Payments Processing



Payment “What Ifs?”

- What are their balances? - Risk > Payment > Identify fraud > Block payment
- What is their history? - Opportunity > Real-time Offers > Upsell

hazelcast

Payment Processing Case Study

Challenge

- Before settling a transaction, payment processing systems check the merchant details by forwarding them to the card's issuing bank or association for verification, and carry out anti-fraud measures
- Each step in this pipeline requires the lowest possible latency to deliver a positive customer experience
- With 24/7 global operations and hard SLAs, resiliency and automatic recovery are a must-have

Solution

- Within the payment processing application, Jet acts as the pipeline for each payment process step
- The payment management application orchestrates XML payment instructions and forwards them to the respective card's issuing bank or association for verification, then carries out anti-fraud measures before settling transactions
- Multiple Jet processing jobs are pipeline components. Hazelcast IMDG distributed IMaps are used for transaction ingestion and messaging

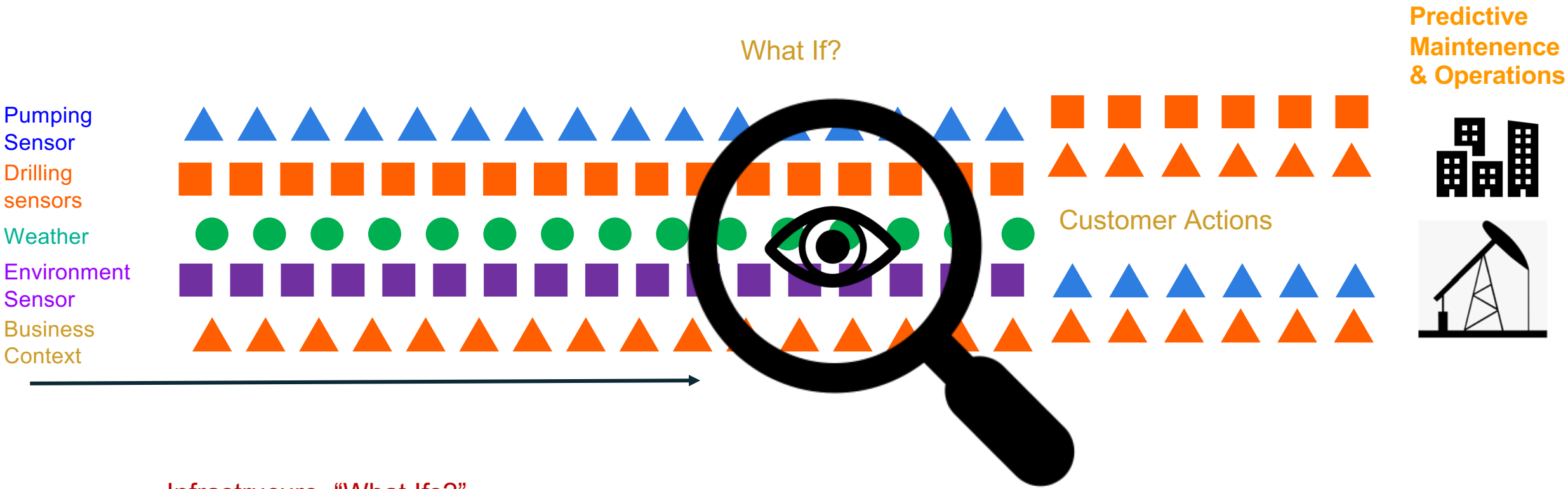
Why Hazelcast Jet

- High-performance connectors between Jet and IMDG enable low-latency operations; consistent low latency of the Hazelcast platform keeps the CGI payment management application within strictest SLA requirements
- Automatic recovery of the Jet cluster achieves high-availability even during failures
- Open source, standards-based avoids vendor lock-in

Customer Success

- A global information technology solutions company
- Processing 10's of 1,000's of payments per second today
- Built-in scalability to support future business

Use-Case - Infrastructure Monitoring

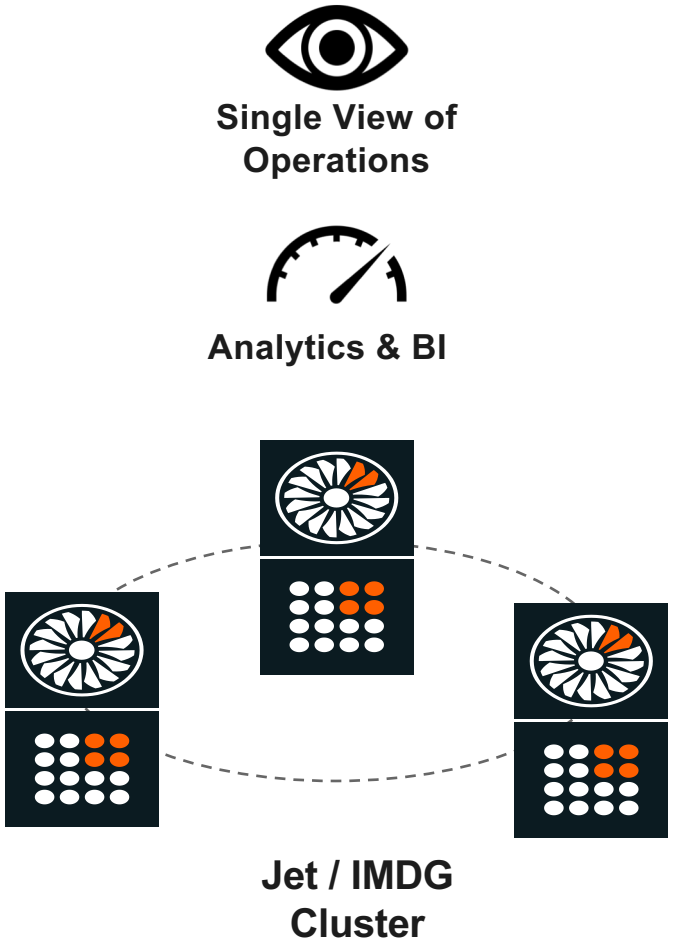


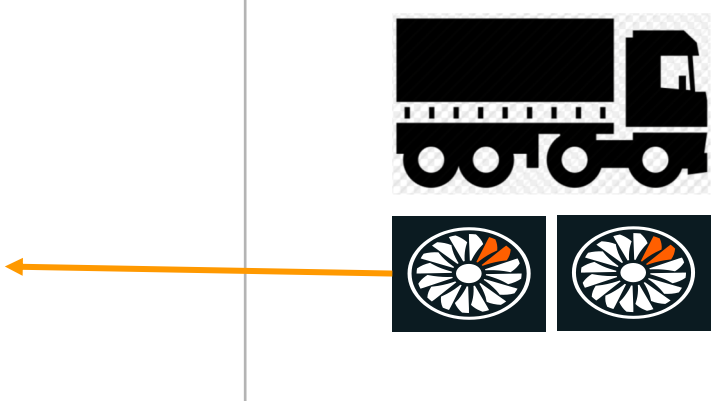
Infrastrucure "What Ifs?"

- What components will fail or require maintenance?
- Should I increase/decrease rate of drilling?
- Can I optimize production?

hazelcast

Use-Case Instance - Oil Infrastructure





Jet adjusts rig settings in real time

Data Center Stream Processing

- Ingest & Consolidation
- Enterprise-Wide Activity Tracking & Scheduling

Operational Site - Edge Processing - Jet uniquely able to run in Edge

- Real-time Low-latency Edge Decisions
- Data Ingest, Filtering, & Aggregation to Feed Data Center (save bandwidth

Edge Processing: Oil & Gas Field Equipment Monitoring

SigmaStream

Challenge

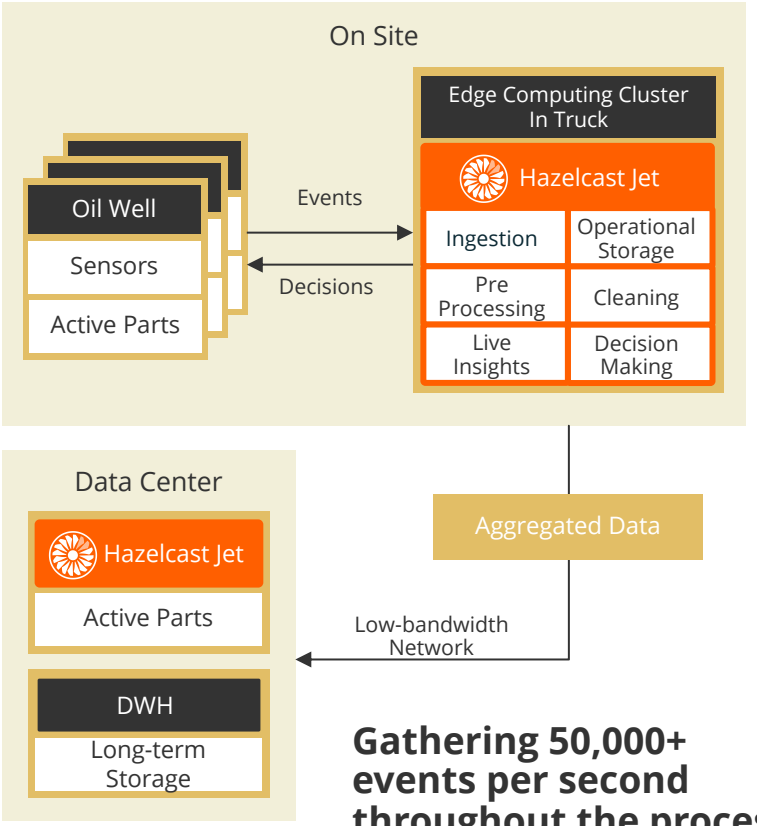
Leading oil & gas system integrator, specializing in acquisition, persistence, secure transportation and dissemination of high-frequency sensor data needed low-latency early issue detection and automated remediation to avoid production loss and optimize well productivity

Solution

- Hazelcast Jet, as processing backbone of application monitoring well sensors with varying formats and frequency, computes data insights to decisions
- Jet adjusts rig settings in real time
- Embedded Hazelcast IMDG as operational data store for easy scaling (bare metal or AWS)

Why Hazelcast Jet

- Embeddability into constrained environments
- No dependencies
- Performance and scalability
- In-memory data store with parallel processing enables scalable, real-time analytics
- Open source, standards-based avoids vendor lock-in



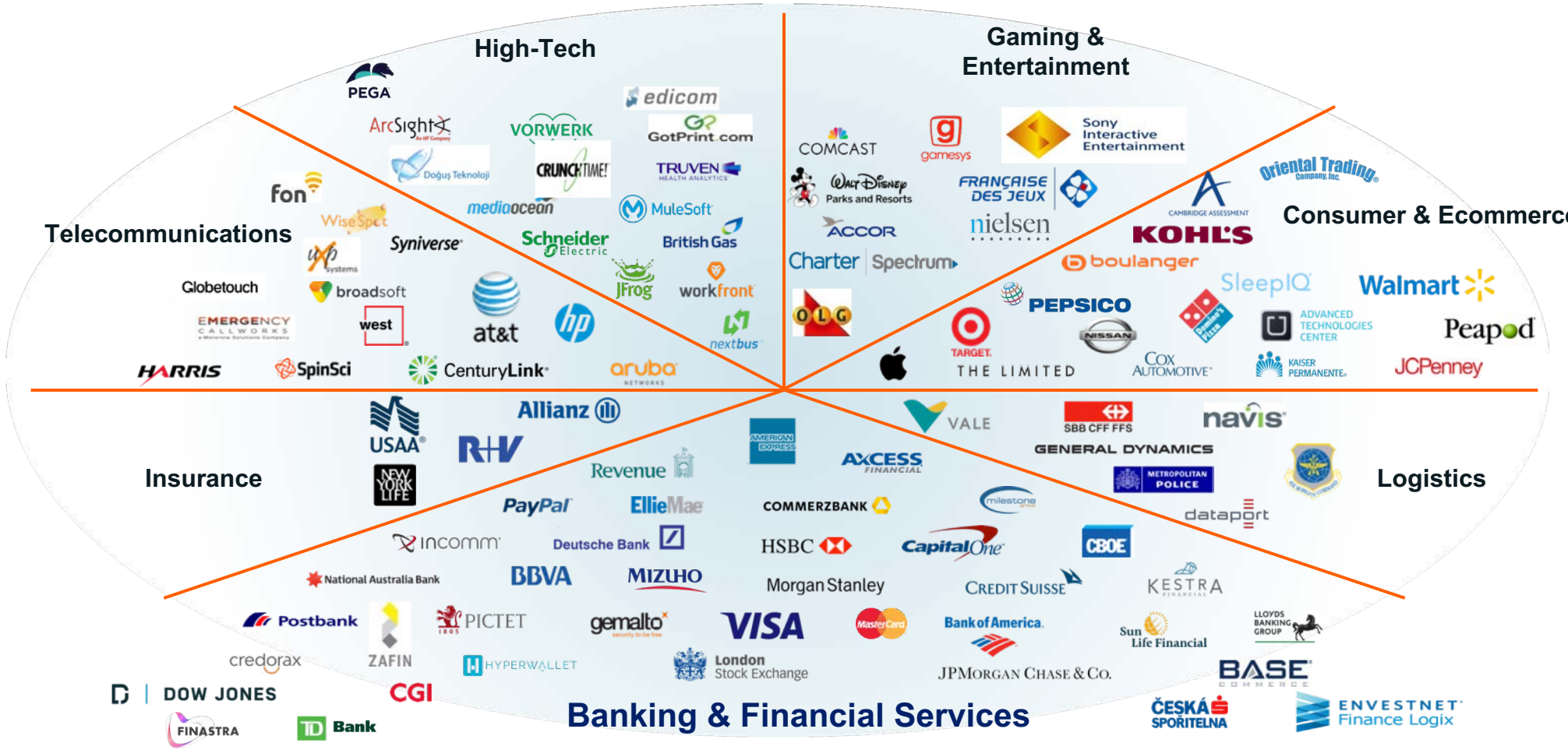
Gathering 50,000+ events per second throughout the process

hazelcast

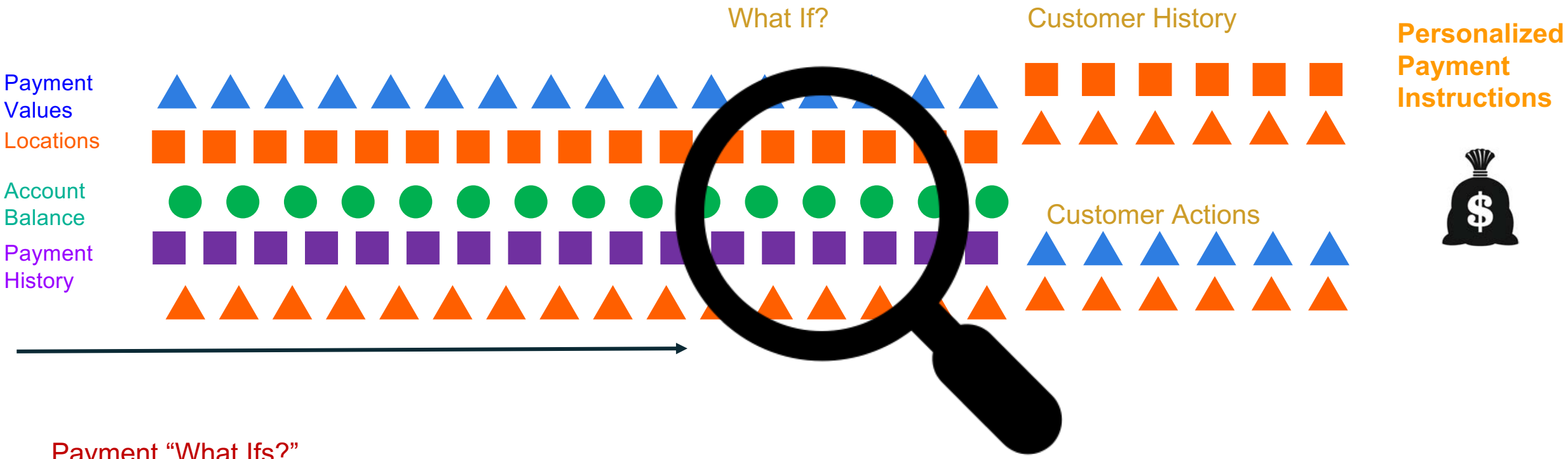

Thank You
hazelcast

Select Customers by Industry

- 50 of the world's largest financial services companies
- 6 of the world's largest e-commerce companies
- 7 of the world's largest communications companies



Use-Case – Payments Processing



Payment “What Ifs?”

- What are their balances? - Risk > Payment > Identify fraud > Block payment
- What is their history? - Opportunity > Real-time Offers > Upsell

hazelcast

ETL Case Study

Finantix

Challenge

- Valuable information such as accounts, portfolios, positions, policies, assets and holdings has to be loaded from multiple sources and systems in order to be analyzed and broken down
- This involves loading, normalizing, reclassifying, combining and aggregating in large scale
- With hard SLAs, high throughput, resiliency and automatic recovery are a must-have

Solution

- Within the analytical application, Jet acts as the ETL pipeline for loading and pre-processing the data
- Data has to be available for analysis as soon as possible. Jet distributes the ETL job across the cluster to reduce the processing time. This allows operations under hard SLAs
- ETL jobs may fail as a result of hardware fault. Restarting the processing would lead to breaking the SLA. Jet brings resilience – the ETL job can resume from where it left off
- Reading from various systems of record is made possible by wide range of connectors in Jet library

Why Hazelcast Jet

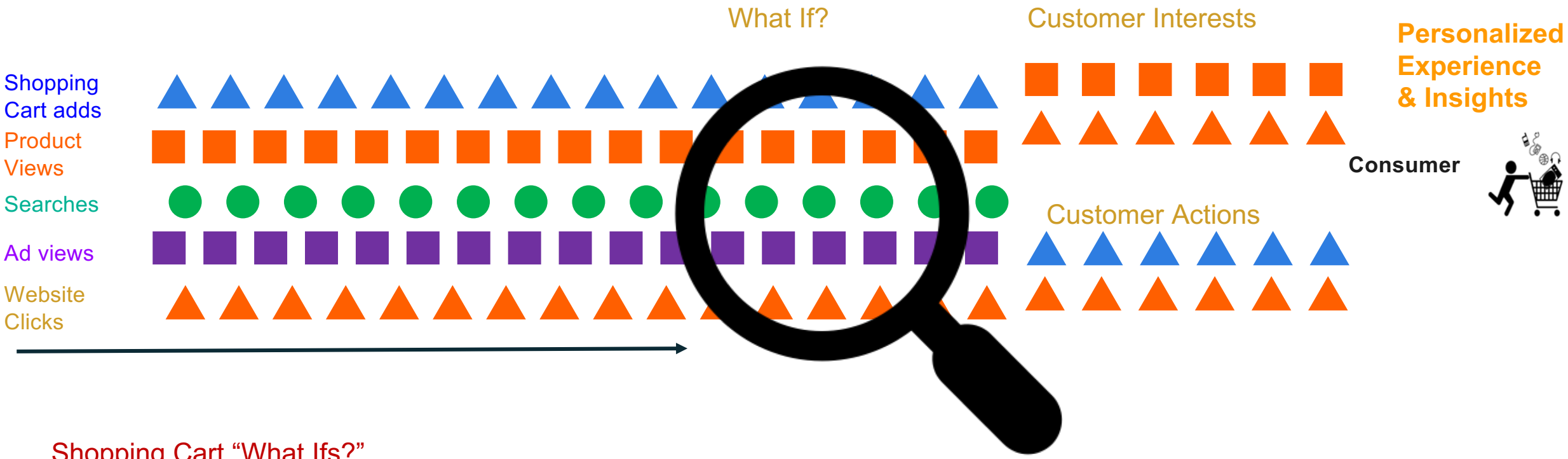
- Embeddable architecture of Jet allows OEMing Jet into Finantix products making deployment into conservative and restrictive banking environments possible
- Automatic recovery of the Jet cluster achieves high-availability even during failures
- Open source, standards-based avoids vendor lock-in

Customer Success

- A global fintech company founded in 1994
- Helps leading financial institutions digitize and transform key processes in the financial services industry
- Built-in scalability to support future business

hazelcast

Use-Case - Personalization - Online Retail

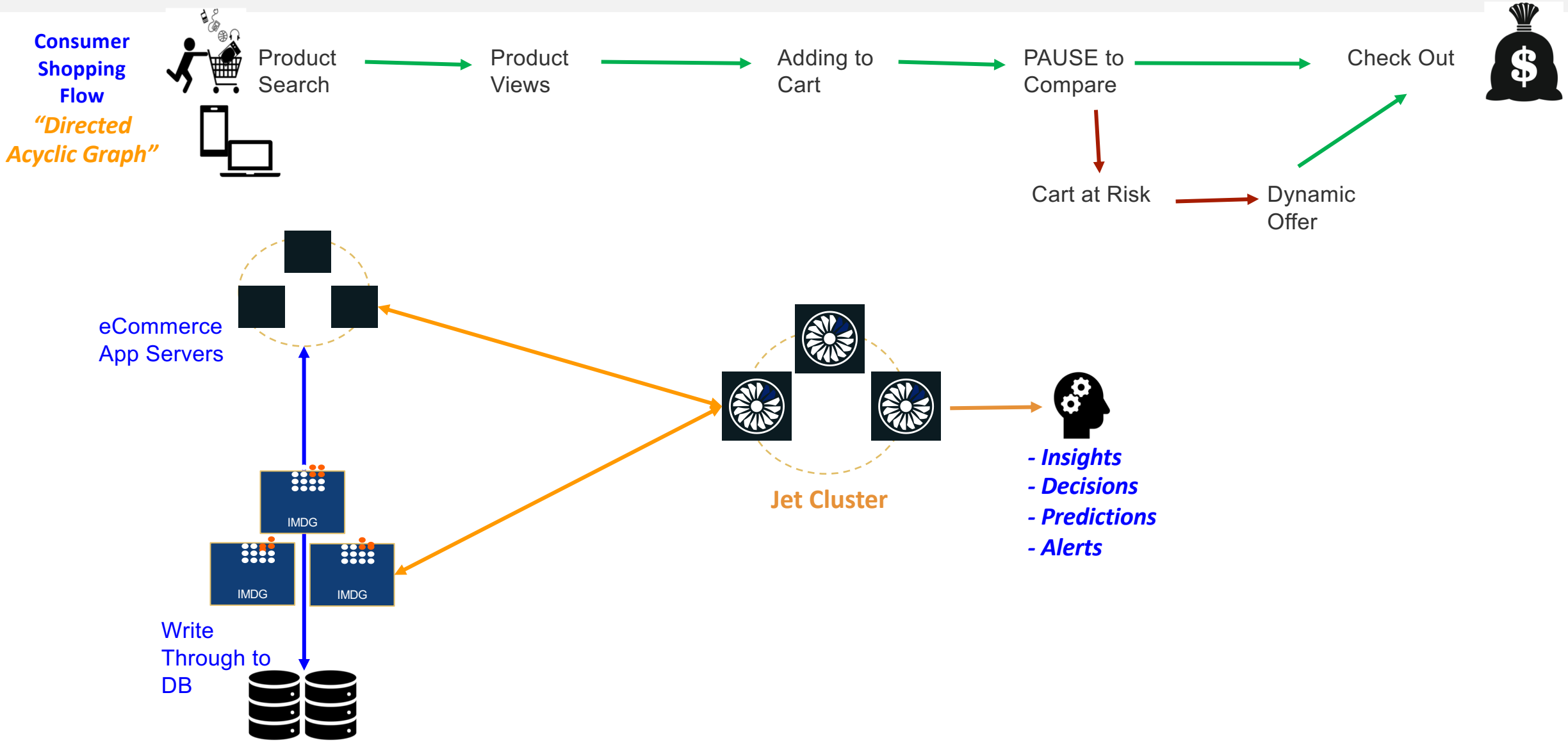


Shopping Cart “What Ifs?”

- Have they paused shopping? - Risk > Cart Abandonment > Offer free shipping > Convert
- Are there offers correlated to their interactions - Opportunity > Real-time Offers > Upsell

hazelcast

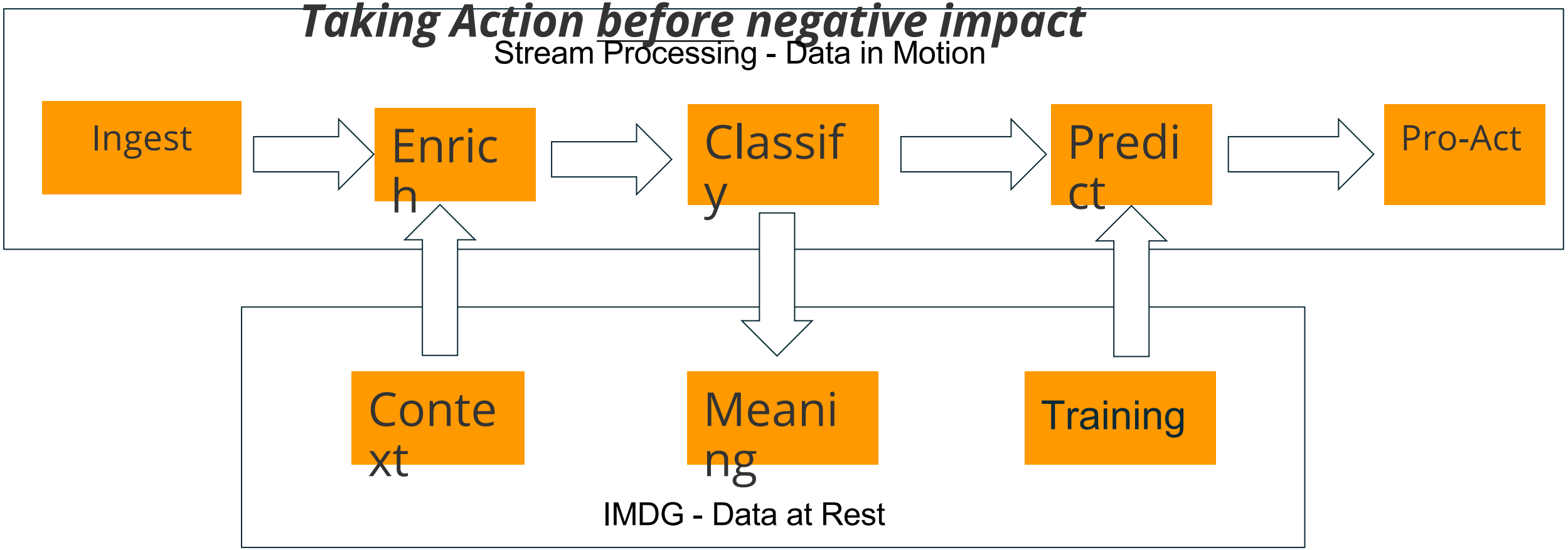
Why Latency Matters - Real-time Offers



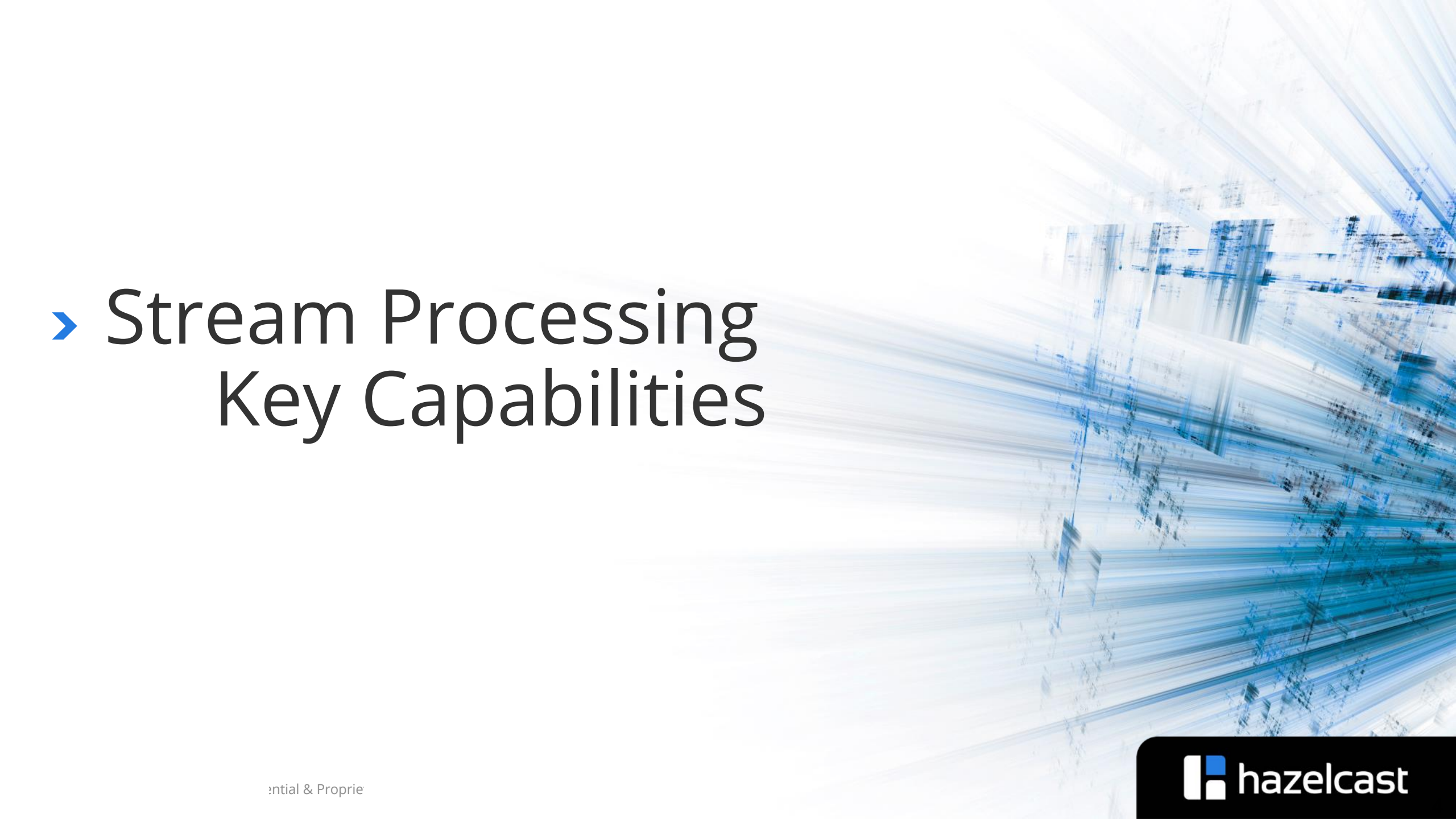
Stream Processing with Machine Learning

Moving Actions from Reactive to Pro-Active

Taking Action before negative impact



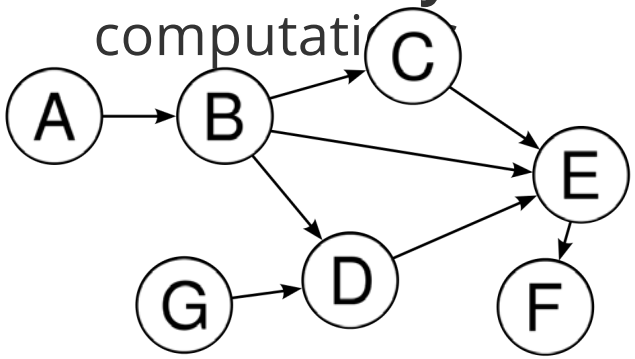
Stream Processing Key Capabilities



hazelcast

Directed Acyclic Graphs

- **Directed Acyclic Graphs** are used to model computati



- Each vertex is a step in the computation
- It is a generalisation of the MapReduce paradigm
- Supports both batch and stream processing
- Other systems that use DAGs: Apache Tez, Flink, Spark, Storm...

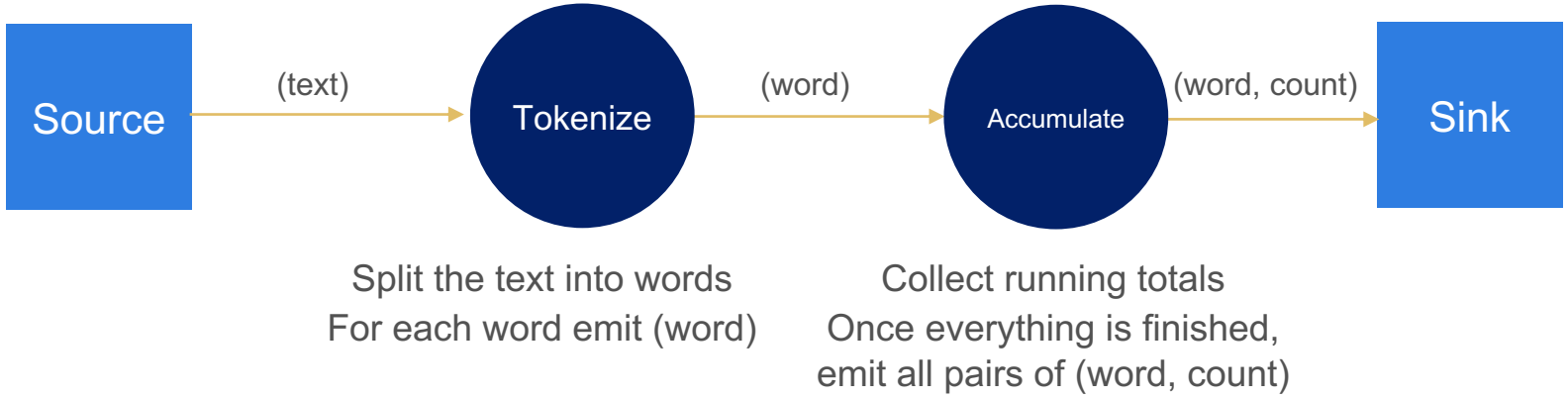
Example: Word Count

- Naïve, single threaded world:

1. Iterate through all the lines
2. Split the line into words
3. Update running total of counts with each word

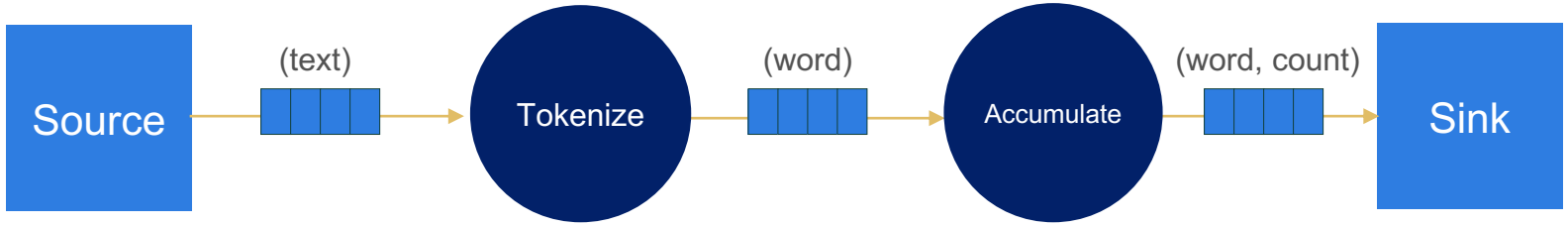
```
final String text = "...";
final Pattern pattern = Pattern.compile("\\s+");
final Map<String, Long> counts = new HashMap<>();

for (String word : pattern.split(text)) {
    counts.compute(word, (w, c) -> c == null ? 1L : c + 1);
}
```

We can represent the computation as a **DAG**



Still single-threaded execution:
each Vertex is executed in turn sequentially,
wasting the CPU cores

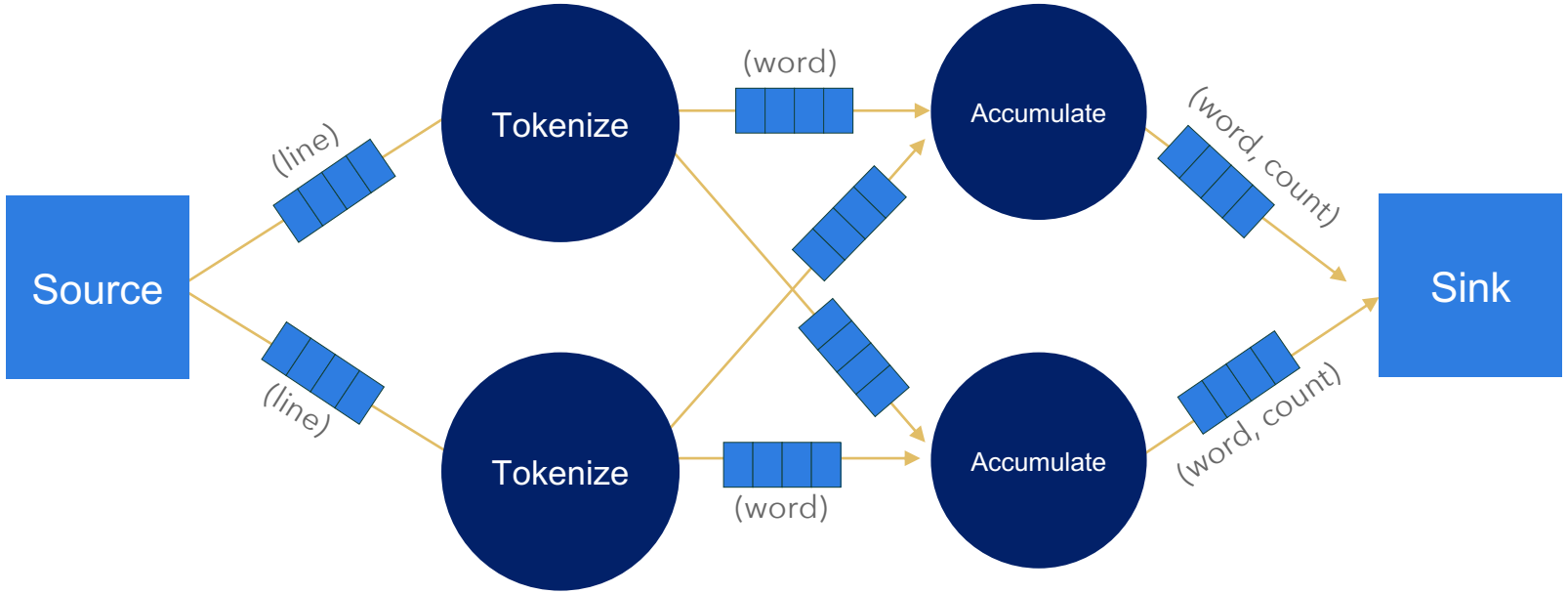
By introducing **concurrent queues** between the vertices we enable each vertex to run concurrently



Split the text into words
For each word emit (word)

Collect running totals.
Once everything is finished,
emit all pairs of (word, count)

The Accumulator vertex can also be executed in parallel by **partitioning** the accumulation step by the individual words.



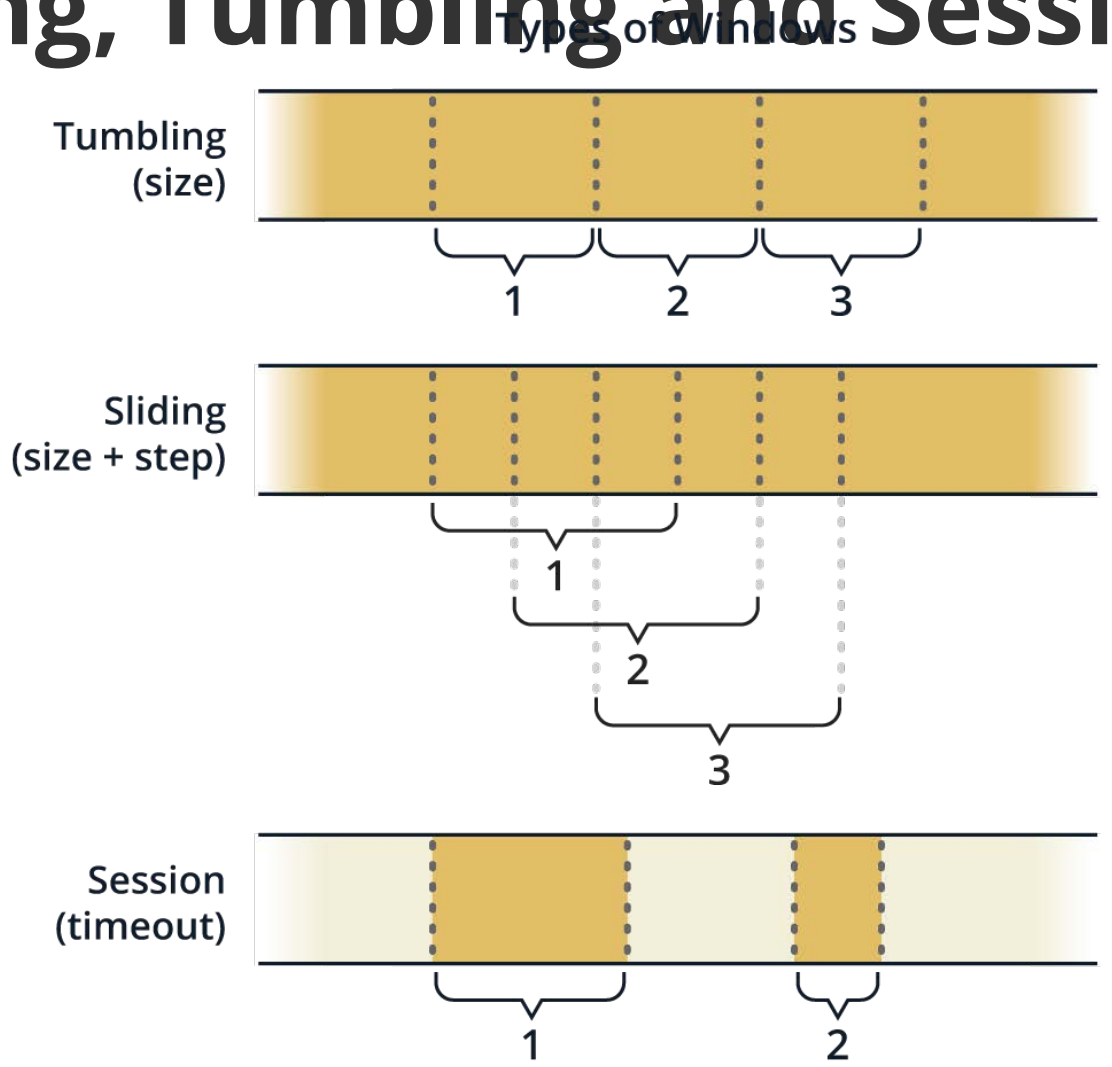
We only need to ensure the **same** words go to the **same** Accumulator.

The steps can also be distributed across multiple nodes. To do this you need a distributed **partitioning** scheme.

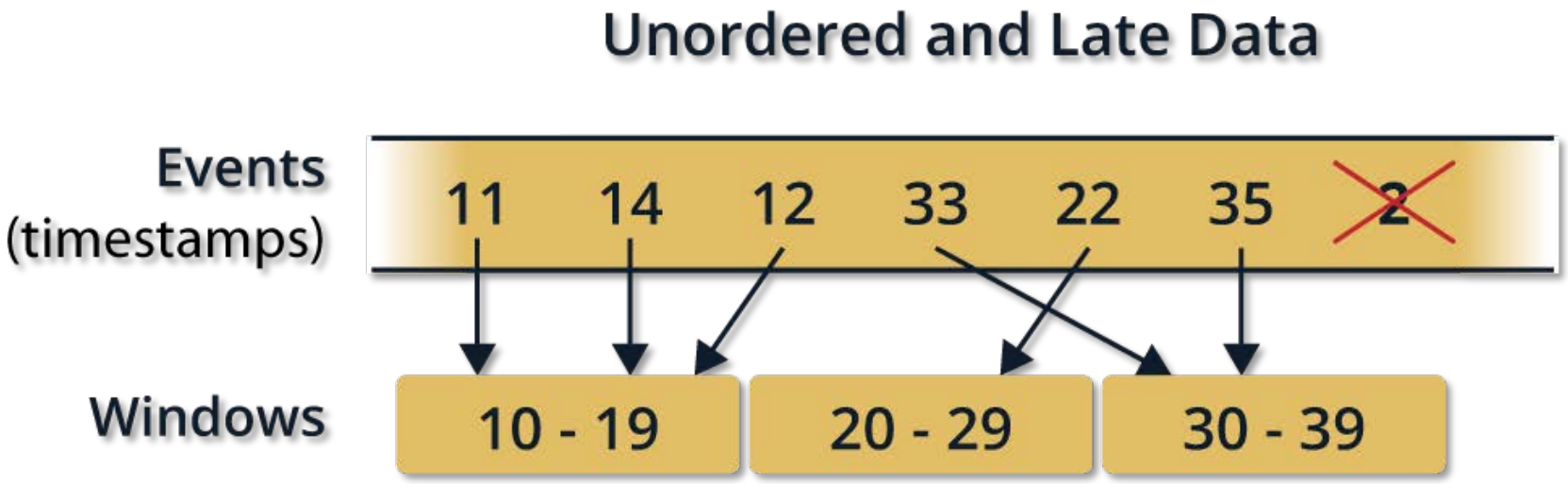
Data Inputs(Sources) and Outputs(Sinks)

- Hazelcast `Icache (JCache)`, (batch and streaming of changes)
- Hazelcast `IMap` (batch and streaming of changes)
- Hazelcast `IList` (batch)
- HDFS (batch)
- Kafka (streaming)
- Socket (text encoding) (streaming)
- File (batch)
- FileWatcher (streaming – as new files appear)
- JDBC (batch)
- NoSQL (Cassandra, MongoDB)
- Time Series (InfluxDB)
- JMS (streaming)
- Custom using simple builders (batch and streaming)

Key to Stream Processing – windows
Sliding, Tumbling and Session Windows



Unordered and Late Data Handling



hazelcast

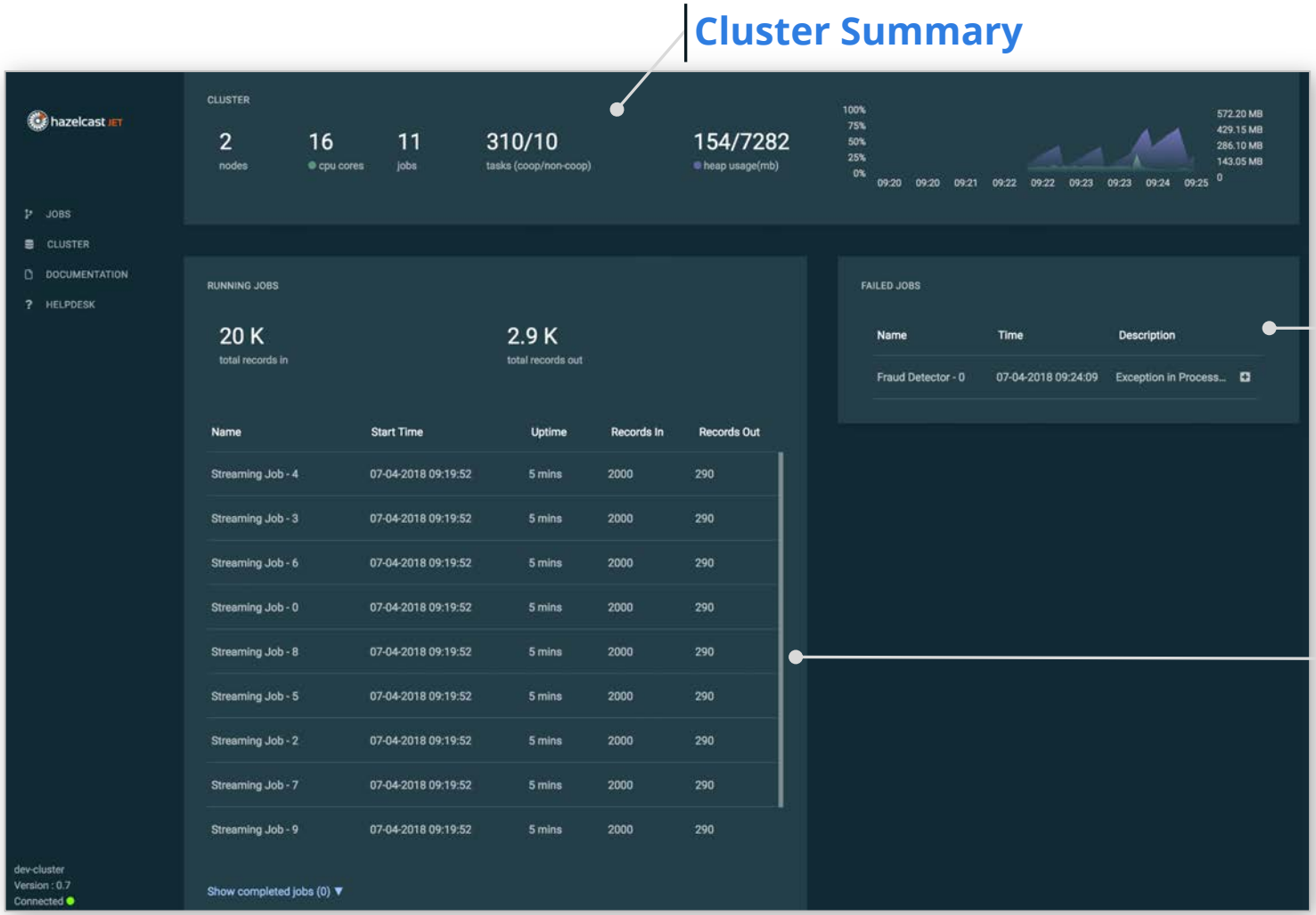
Job Elasticity

- Jobs are elastic – they can dynamically scale to make use of all available members, following cluster topology changes
- Job state and lifecycle are saved to IMDG IMaps and benefit from their performance, resilience, scale and persistence
- Automatic re-execution of part of the job in the event of a failed worker
- Tolerant of loss of nodes; missing work will be recovered from last snapshot and re-executed
- Cluster can be scaled without interrupting jobs – jobs benefit from the increased capacity
- State and snapshots can be persisted to resume after cluster restart (Version 3.0)

hazelcast

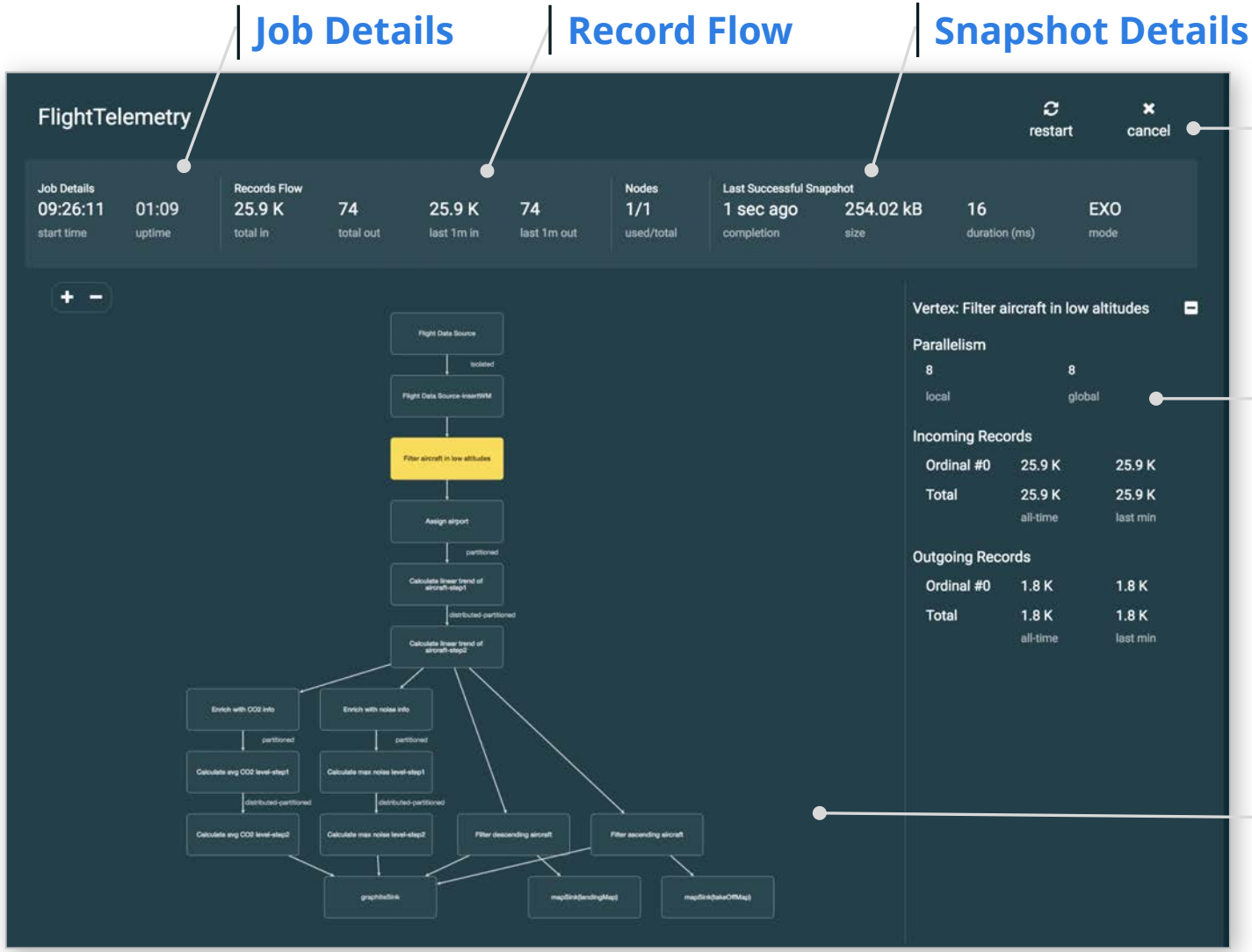
Jet Management Center: Dashboard

EE PS



hazelcast

Jet Management Center: Job Detail



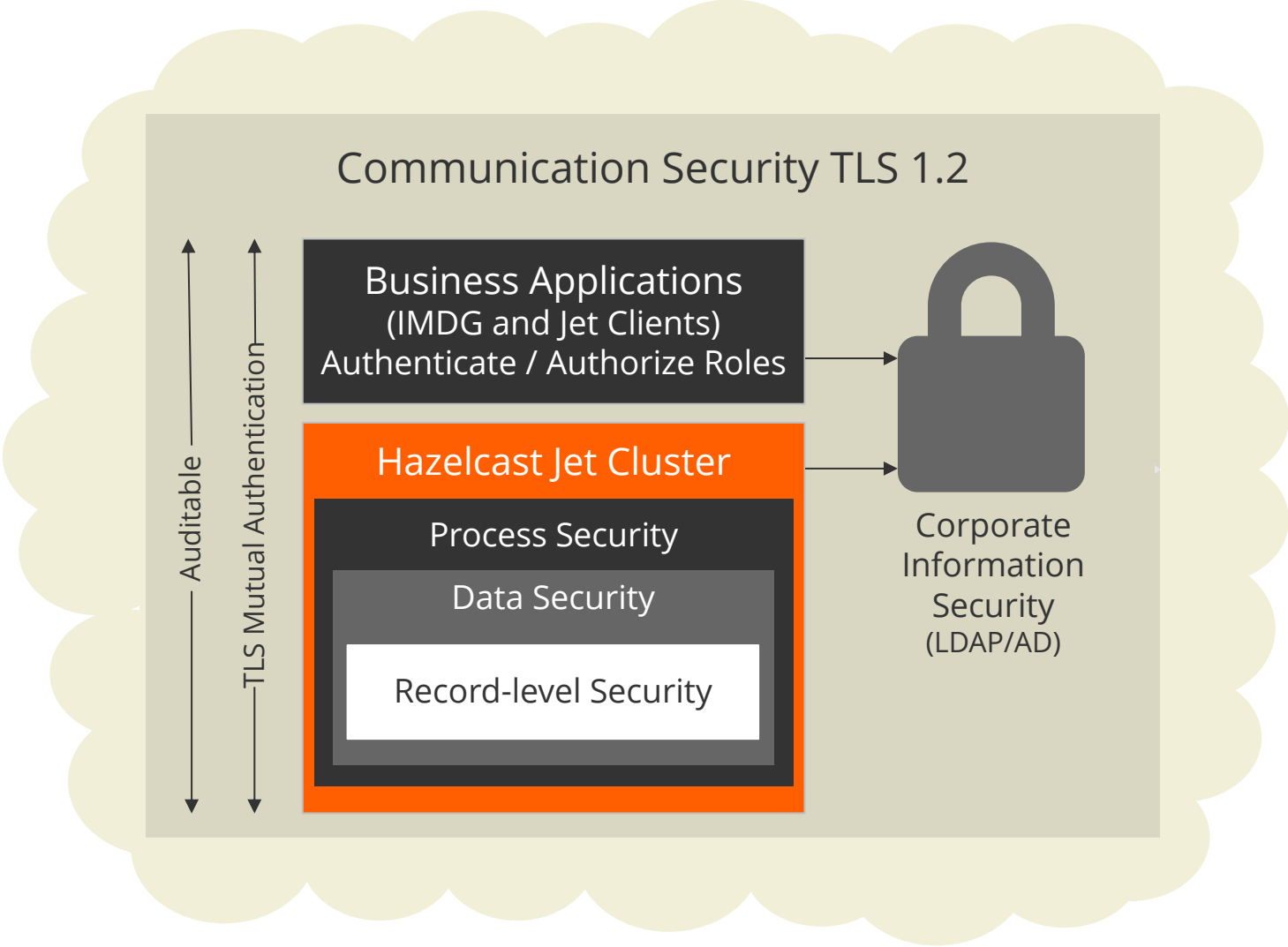
Job Lifecycle

DAG Vertex Details

DAG Visualization

Security Suite Features

EE



Fault Tolerance: Distributed State Snapshots

Exactly-Once, At-Least Once or No Guarantee to optimize between performance and correctness

Distributed State Snapshots to back-up running computations

Resilience with backups distributed and replicated across the cluster to prevent losing data when member fails

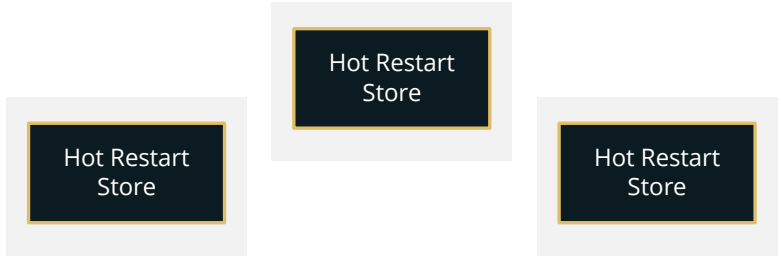
Simplicity as the snapshots are stored in embedded in-memory structures. No further infrastructure is necessary

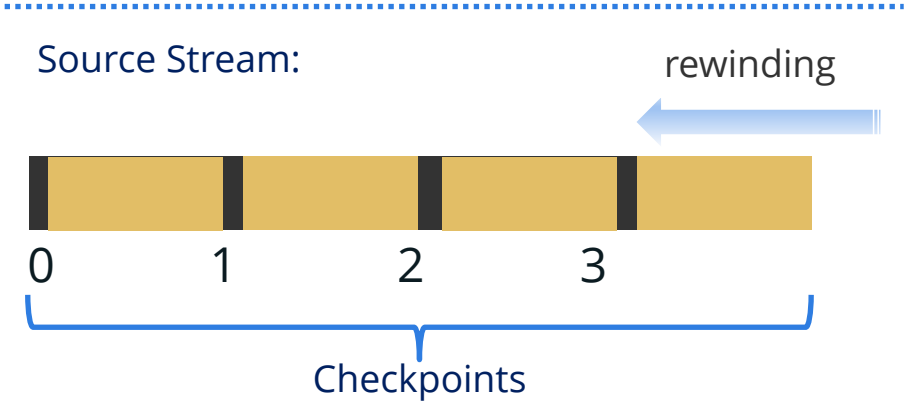
hazelcast

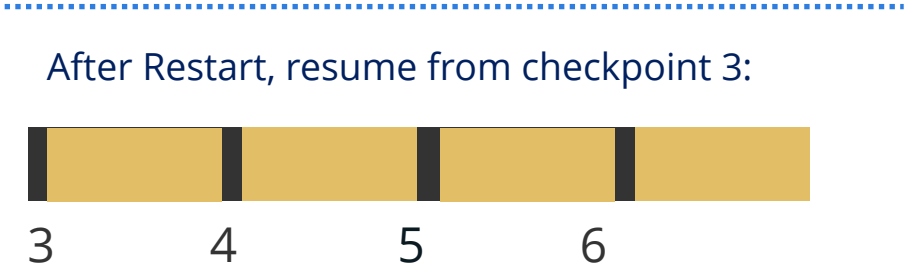
Lossless Recovery: Automatic Job Resumption

EE

Only Hot Restart Stores remain:







- When cluster is restarted, Jet discovers it was shut down with running jobs
- Jet restarts the jobs
- Checkpoints are recovered
- For streaming, rewindable sources are rewound using saved offsets (Kafka, Hazelcast IMap, Hazelcast ICache events). If the source cannot be fully rewound, the job is terminated with error, or continued, depending on configuration
- Batch sources are resumed from last pointer, otherwise from the beginning

hazelcast

Rolling Job Upgrades

EE

- Allow jobs to be upgraded without data loss or interruption
- Rolling upgrades make use of Jet state snapshots
- Via Job API and Man Center

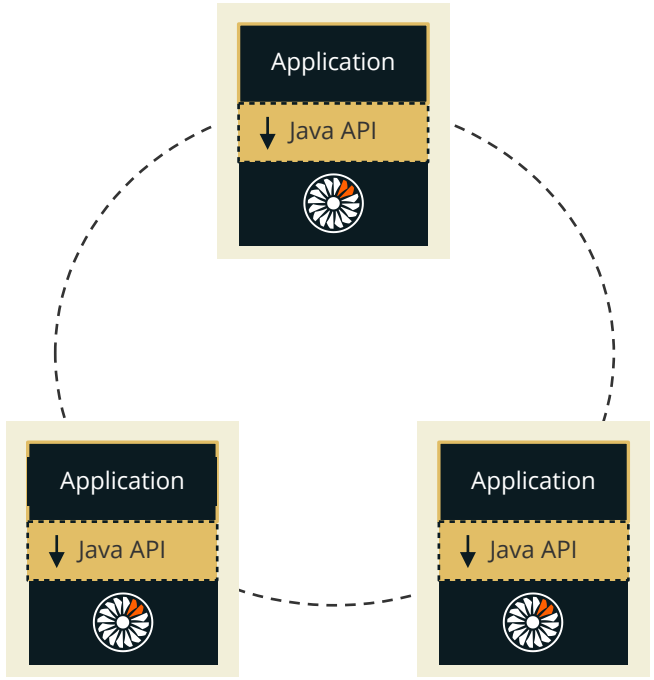
Processing Steps

1. Jet stops the current Job execution
2. It then takes the state snapshot of the current Job and saves it
3. The new classes/jars are distributed to the Jet nodes
4. The job then restarts
5. Data is read from the saved snapshots
6. All of this in a few milliseconds

hazelcast

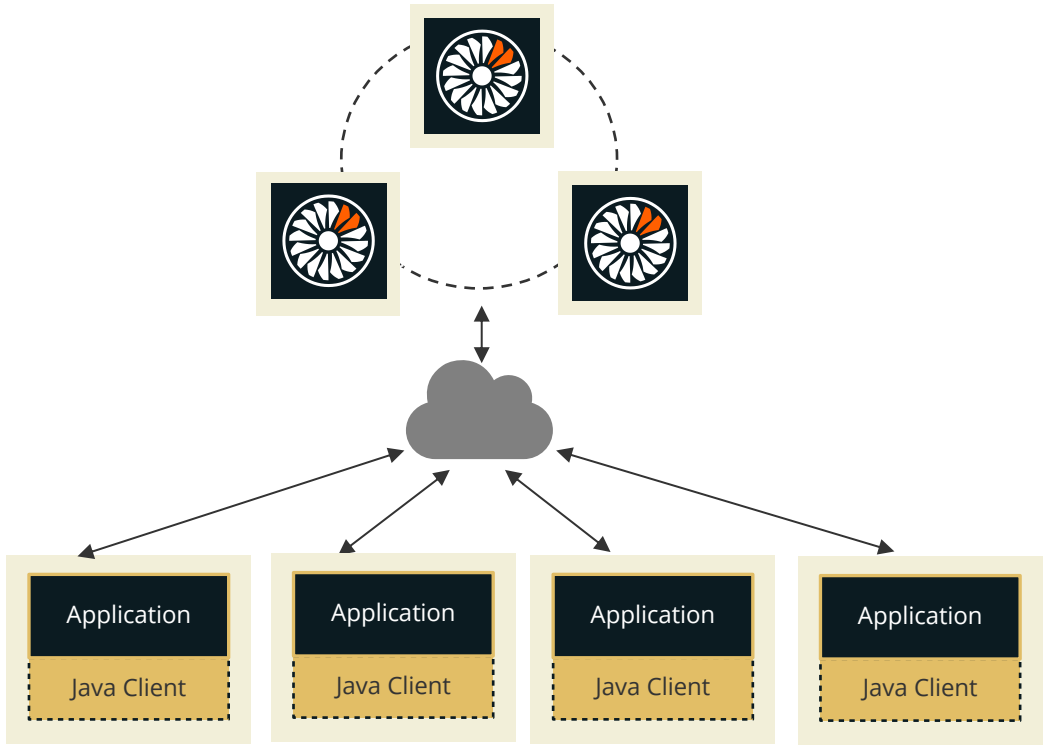
Jet Application Deployment Options

Embedded



- No separate process to manage
- Great for microservices / constrained /Edge
- Great for OEM
- Simplest for Ops – nothing extra

Client-Server

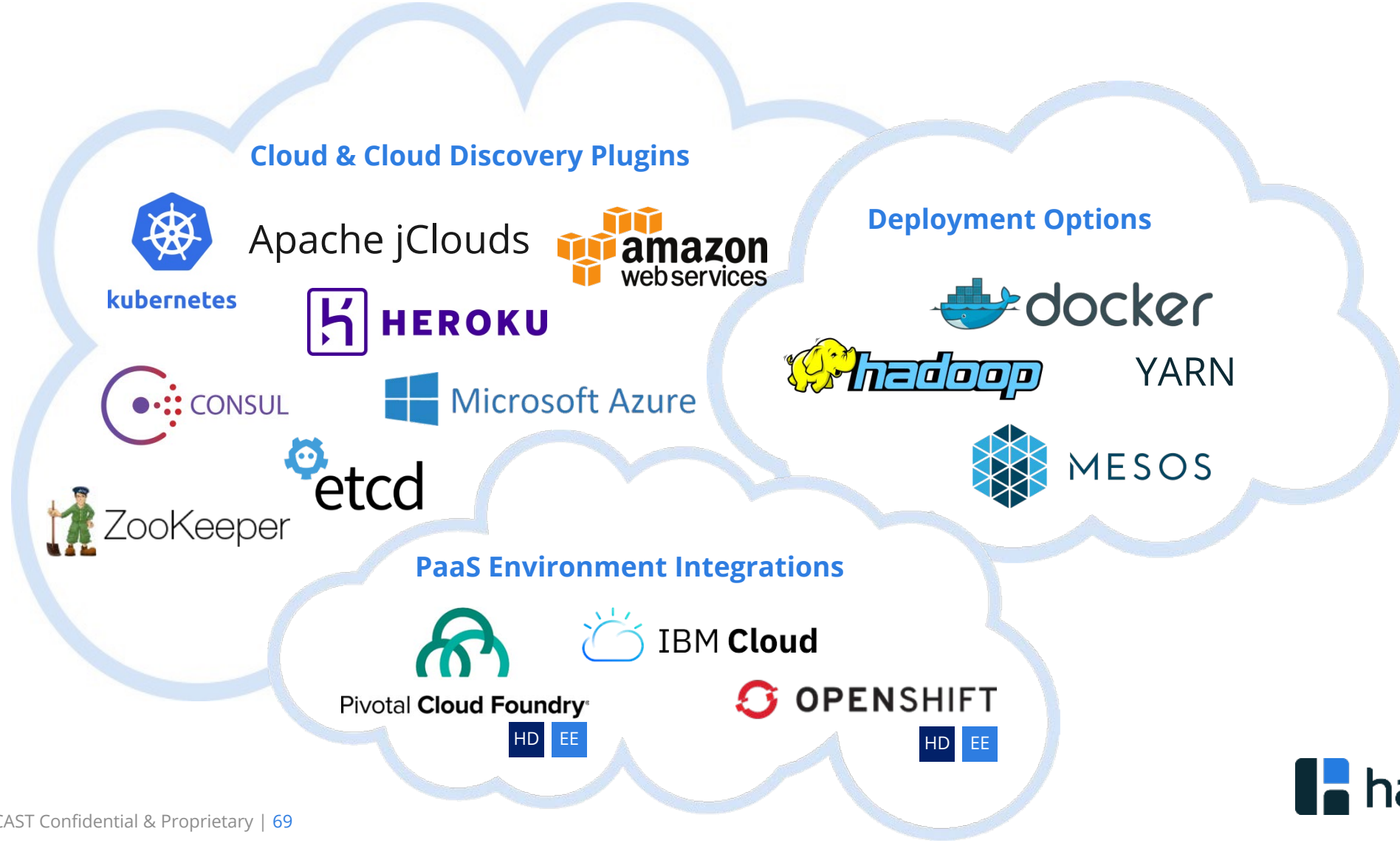


- Separate Jet Cluster
- Scale Jet independent of applications
- Isolate Jet from application server lifecycle
- Managed by Ops

hazelcast

HD EE PS

Hazelcast IMDG Cloud Discovery & Deployment



hazelcast

EE

Hazelcast Jet & IMDG Enterprise for Red Hat OpenShift Container Platform

Simplifies deployment of Jet Enterprise standalone infrastructure, as a certified Red Hat Enterprise Linux based image. Package consists of:

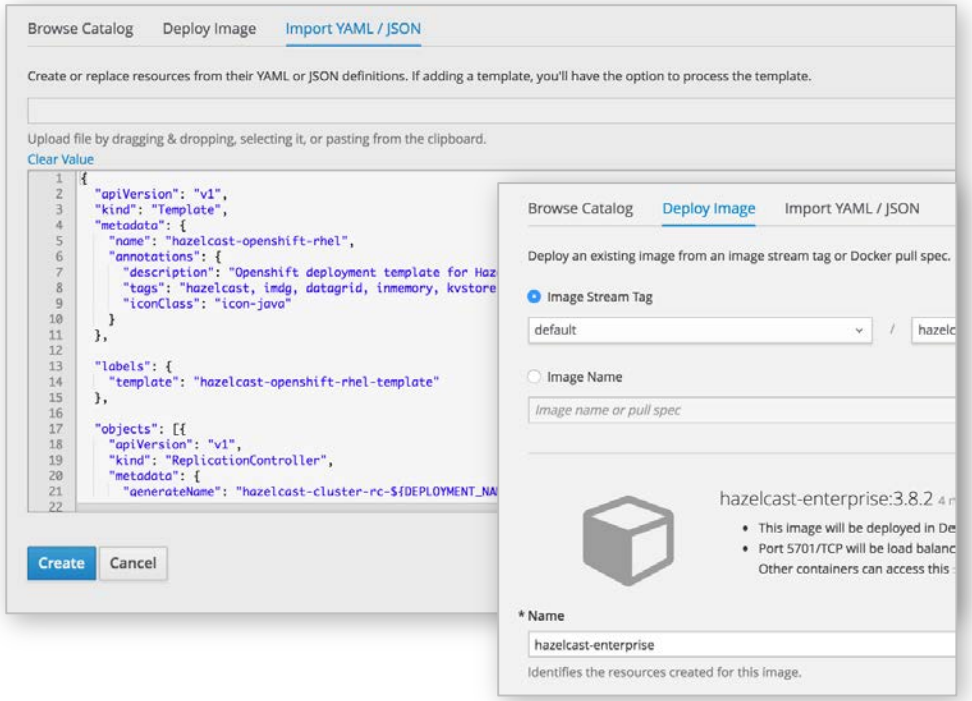
- Hazelcast Jet Enterprise and related dependencies
- Red Hat Enterprise Linux (RHEL) 7.3
- Oracle Java 8
- Health and liveness scripts
- Start and stop scripts

Jet Enterprise for OpenShift Features:

- Hazelcast can be run inside OpenShift, benefiting from its cluster management software, Kubernetes, for discovery of members
- Ability to dynamically pass your Hazelcast configuration in JSON format while creating services

Download at:
https://hazelcast.org/plugins/#hazelcast-openshift-integration

RED HAT® OPENSHIFT Container Platform



hazelcast

Why Hazelcast Jet?

High performance | Industry Leading Performance

Works great with Hazelcast IMDG | Source, Sink, Enrichment

Very simple to program | Leverages existing standards

Very simple to deploy | Embed 12MB jar or Client Server

Works in every Cloud | Same as Hazelcast IMDG

hazelcast