GridGain

Getting Started with Apache Ignite:
Implementing a Digital Integration Hub
Delivering a Digital Business Platform

Glenn Wiebe

September 2, 2020

Inc. 5000

500 Technology Fast 500 2019 NORTH AMERICA Deloitte.

DBTA 100

RED HERRING 100 WINNER GLOBAL

RED HERRING 100 WINNER N. AMERICA

Fast Private 2019 SILICON VALLEY BUSINESS JOURNAL

Bloor MUTABLE AWARD 2019 GOLD

GOLD 2020 STEVIE WINNER AMERICAN BUSINESS AWARDS

Agenda – Delivering a Digital Business Platform

from Frameworks to Facilities with Apache Ignite

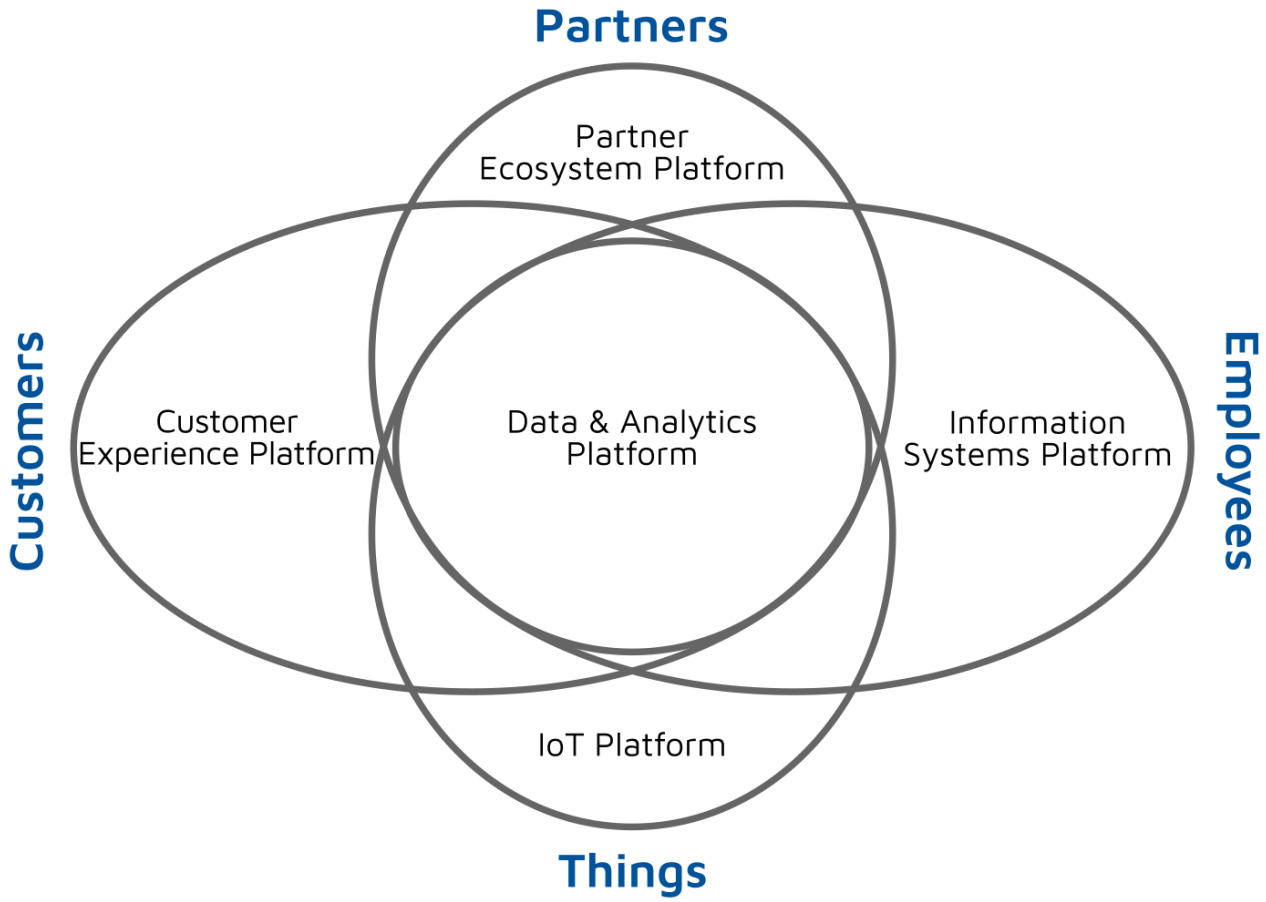
- Digital Integration Hub Architecture…
  - Technical Frameworks
  - Architectures
- DIH Facilities:
  - Storage & Persistence
  - Integration
  - Compute & Event Handling
- Component Implementations:
  - Data Ingest with Ignite (using DataStreamer)
  - Complex Event Processing with Ignite (using Continuous Query)
- Platform Delivery & Management:
  - Development through to Deployment
  - Monitoring & Management Facilities

Digital Integration Hub (DIH) Architecture

Frameworks and architectures for a Digital Business Platform

GridGain

Digital Business Technology Framework

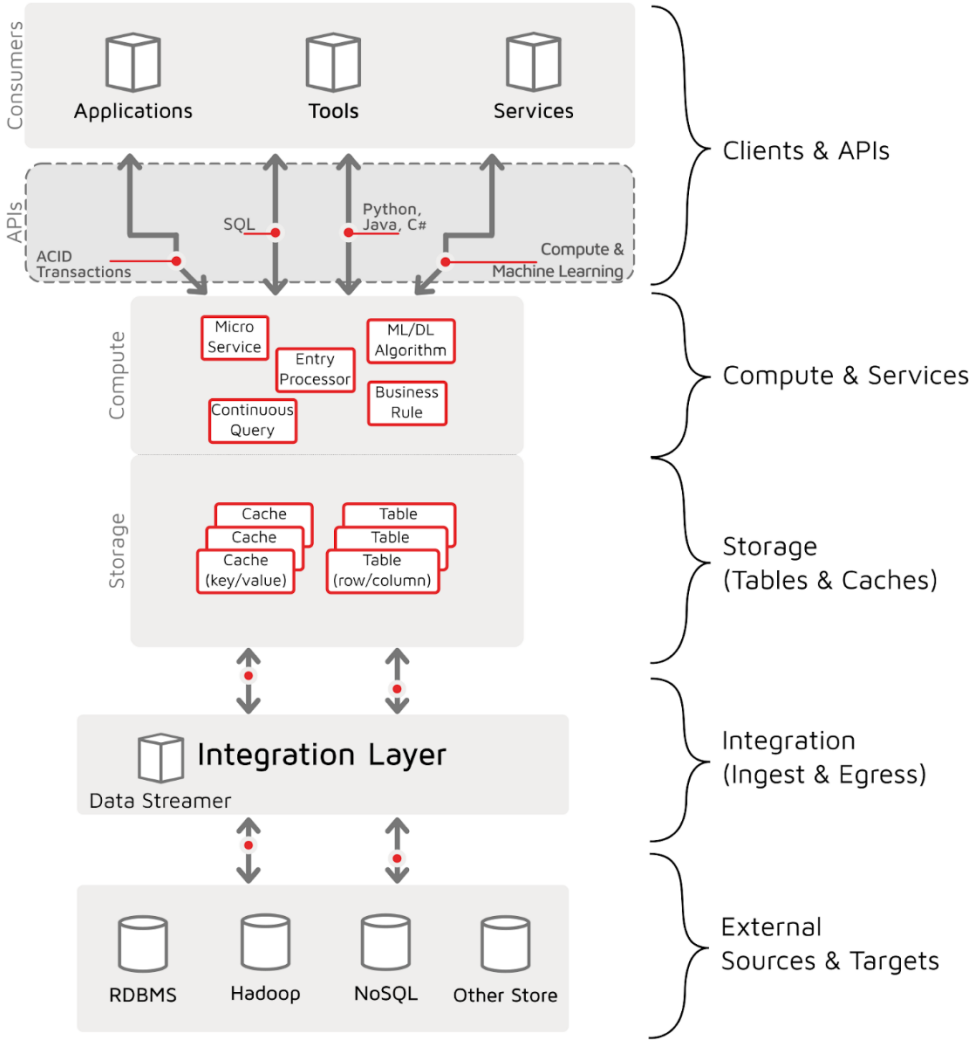


A business technical framework promoted by Gartner

- Describing 4 broad organizations, and their associated technical platforms:
  - Customers – Customer Experience Platform
  - Partners – Partner Ecosystem Platform
  - Employees – Information Systems Platform
  - Things – IoT Platform
- Central Data & Analytics Platform

GridGain

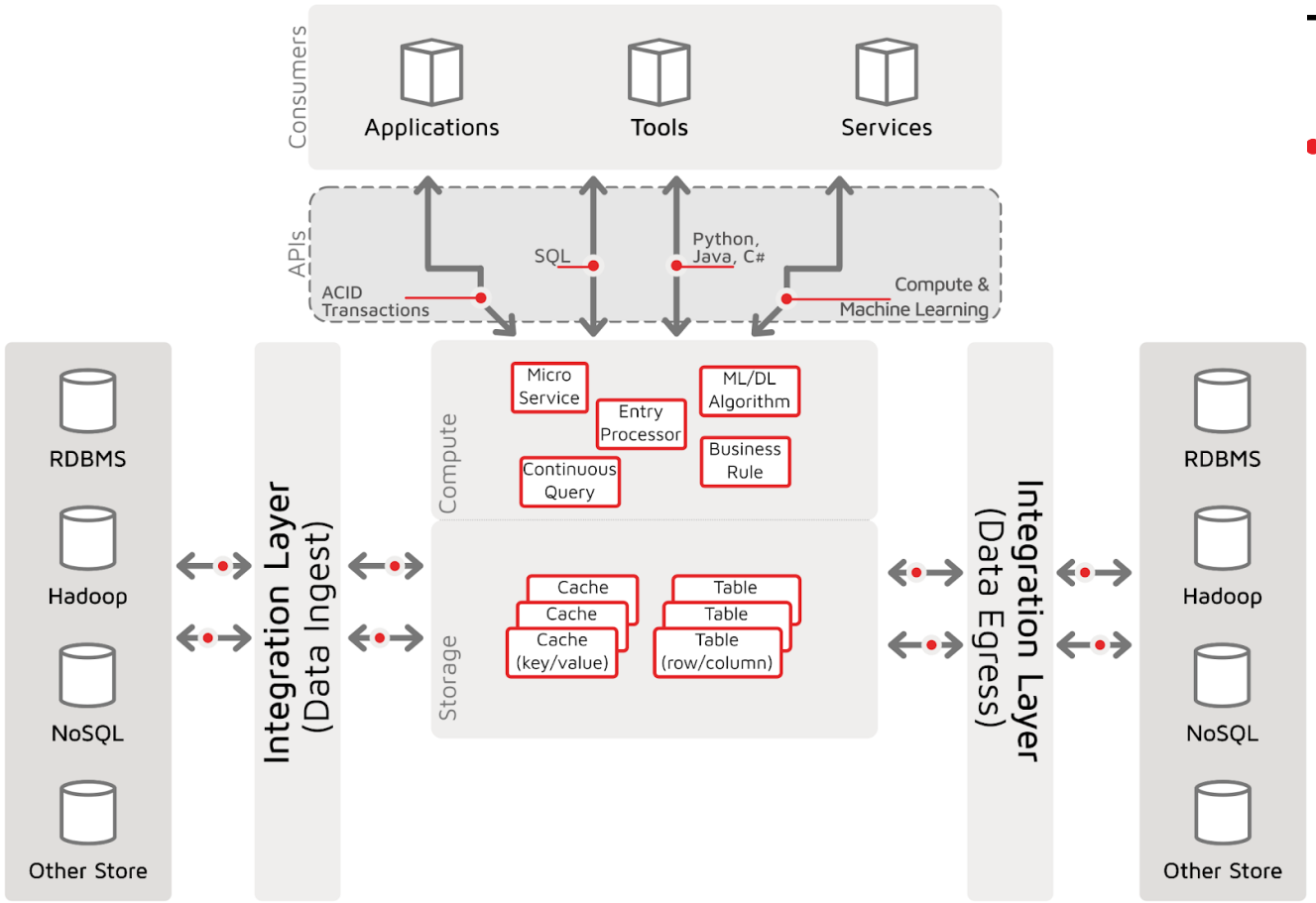
Technology Architecture – Digital Integration Hub



- A modern architecture that supports
  - Storage – Persisting platform needed data
  - Compute – Facilities for building business logic and services to deliver to consumers
  - Integration – Tools, services and facilities for data ingest and data egress
- Multi-model (Relational, NoSQL), Multi-mode (SQL & Key-Value API), Multi-consumption patterns (Tools, Apps, Services)

GridGain

Architecture Depiction – Horizontal DIH



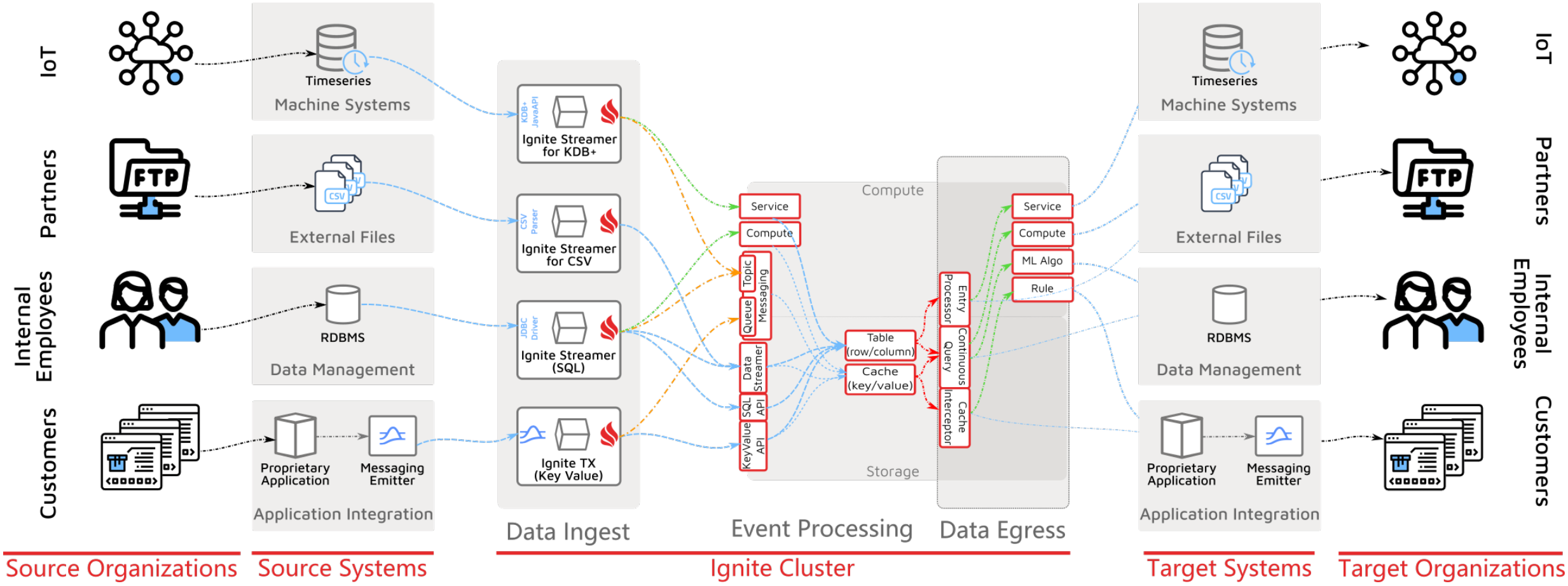
The Horizontal Digital Integration Hub

- Focus on re-factoring lower two levels:
  - Source Data – source systems of varying types and models
  - Inbound Integration – from source systems to central storage & compute facilities
  - Outbound Integration – from central storage & compute facilities to target systems

GridGain

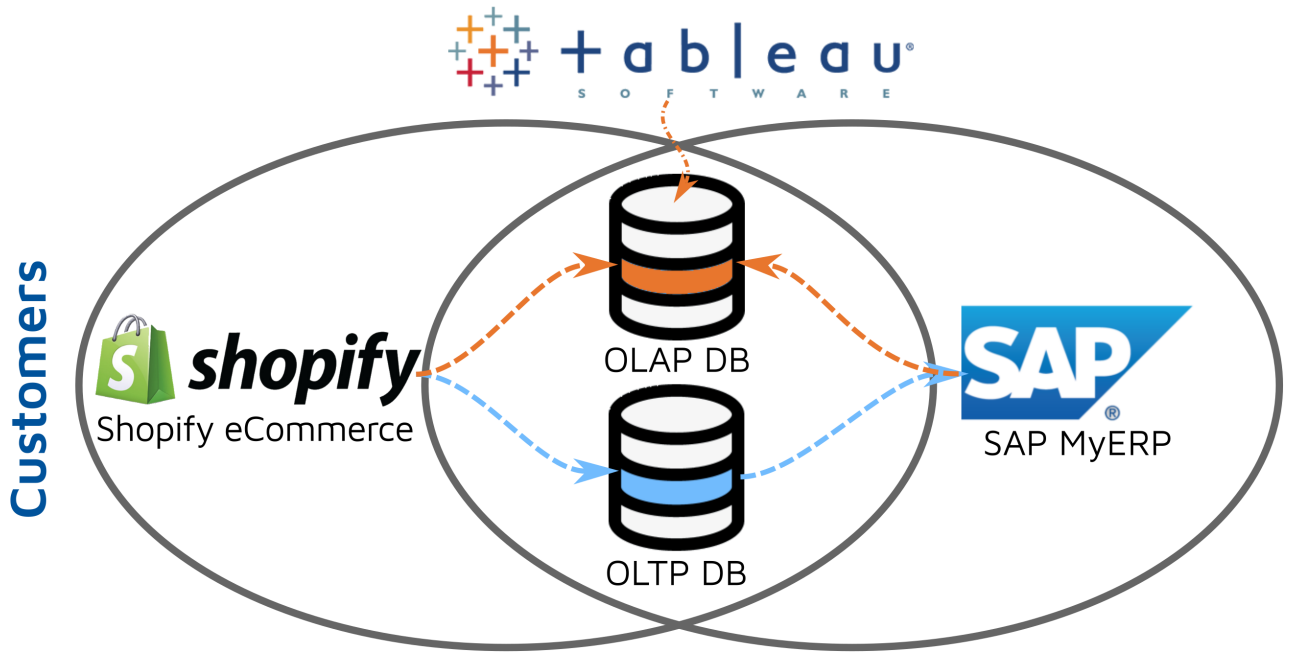
Technology Architecture – Event Stream Processor

ESP Architecture: real-time, message-oriented, event-driven paradigm



GridGain

Technology Framework Usage – Customer to Employee

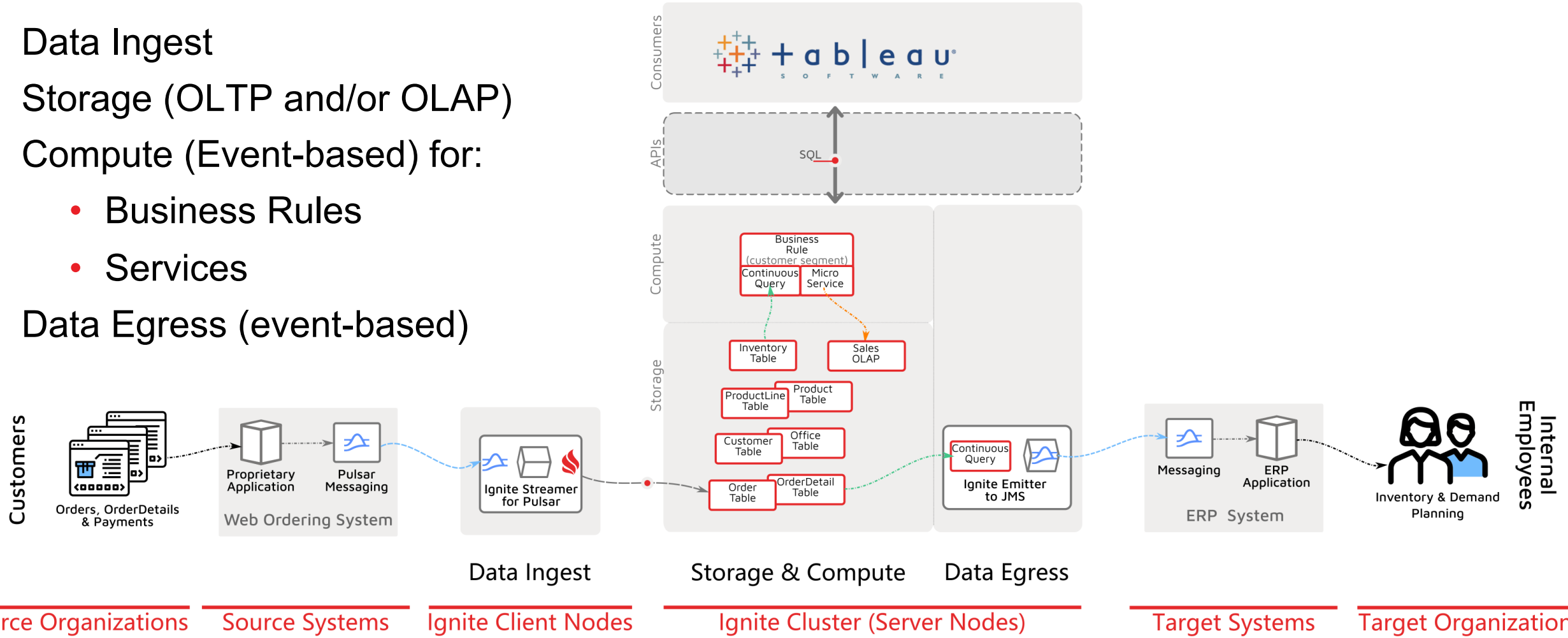


- Customer Experience & Employee Information Systems
- Customer Web sales system feeds internal resource planning system
- shopify eCommerce sales transactions feed inventory & demand planning; Data & Analytics also in landscape
- Data flows through OLTP DB & into OLAP Data Mart

GridGain

Architecture Depiction – DIH: Sales to ERP + Analytics

- Data Ingest
- Storage (OLTP and/or OLAP)
- Compute (Event-based) for:
  - Business Rules
  - Services
- Data Egress (event-based)



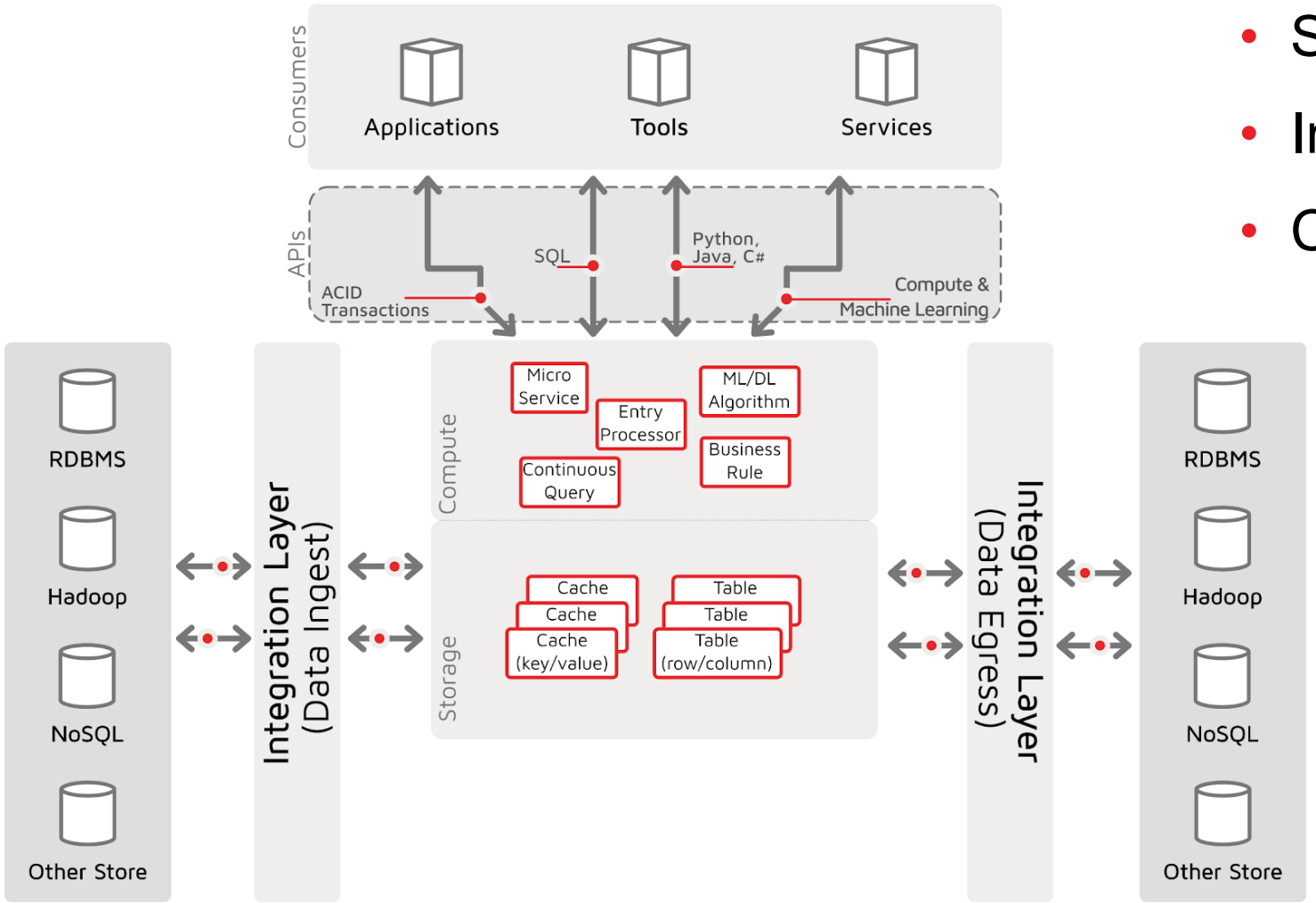
GridGain

Digital Integration Hub (DIH) Facilities

Functional Elements of a Digital Business Platform

GridGain

Digital Integration Hub Facilities



- Storage / Persistence
- Integration
- Compute / Event Handling

GridGain

DIH Facilities – Storage & Persistence

Storage APIs

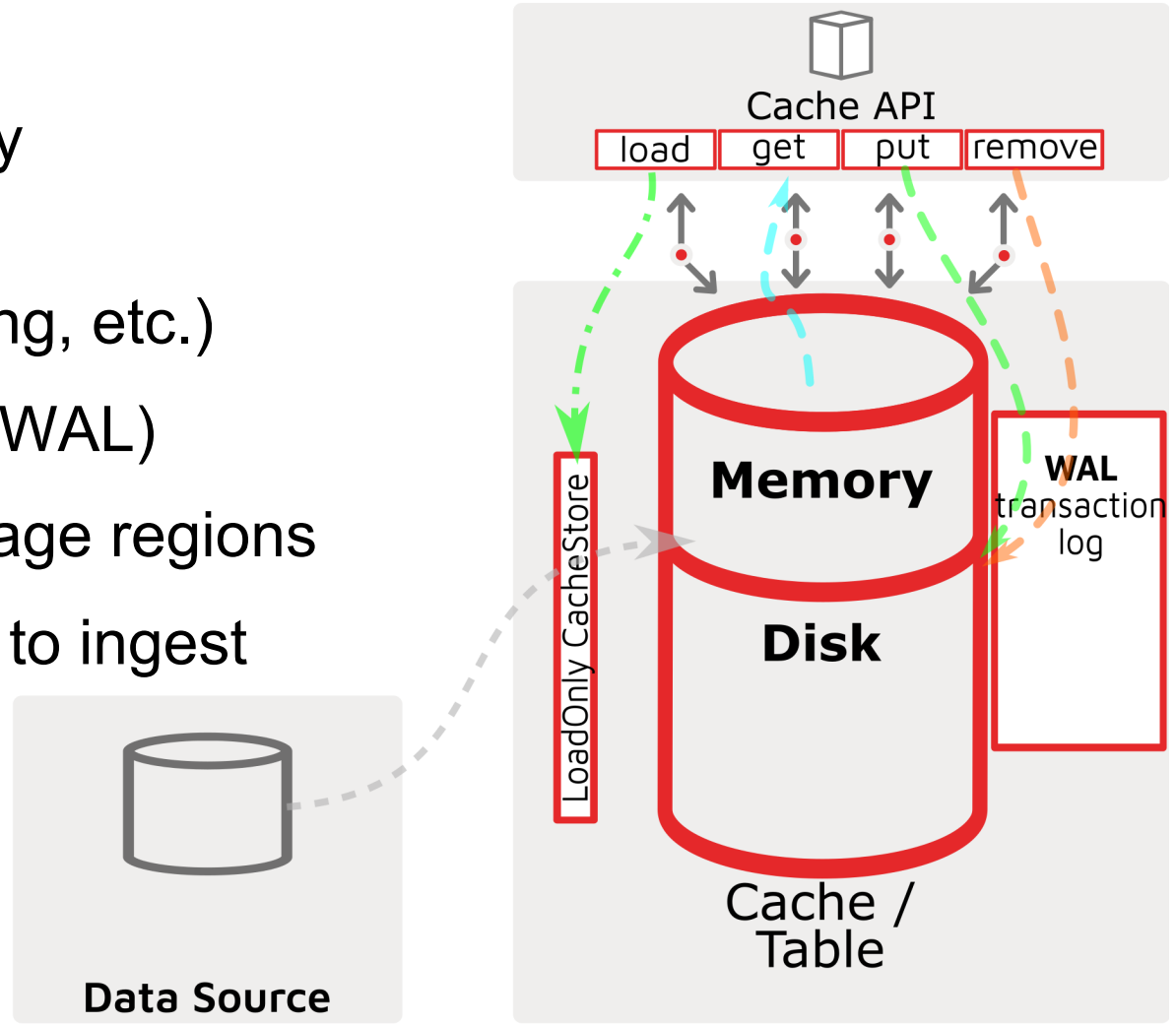
- SQL & NoSQL Key-Value APIs
- Full ACID Transactions

Ignite Storage Options

- Memory only
- Memory with external, 3rd Party Persistence (e.g. Relational, or NoSQL DBs)
- Memory with Ignite Native Persistence

DIH Facilities – Native Persistence

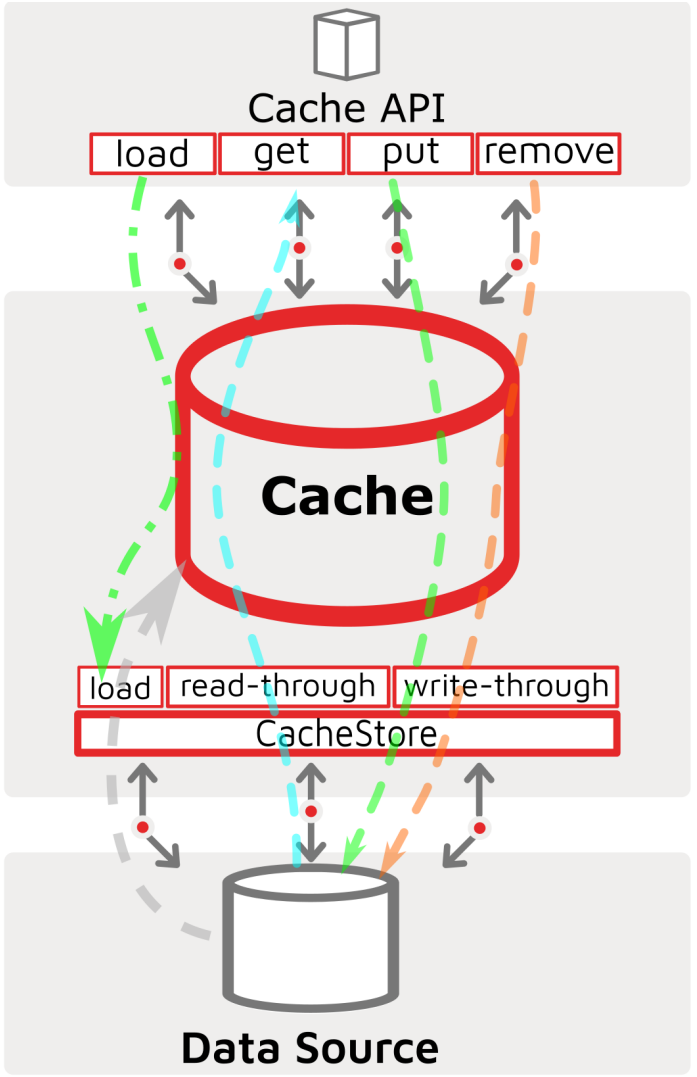
- Memory-centric
- Disk-backed over-flow and durability
- Multi-region, tiered storage
  (e.g. memory, Optane, SSD, Rotating, etc.)
- Transactions via Write-Ahead Log (WAL)
- Full API functionality across all storage regions
- LoadOnly CacheStore can be used to ingest data.



GridGain

DIH Facilities – 3rd Party Persistence

- Implemented via “CacheStore” class
- Out of the box implementations for popular sources like Relational (JDBC/ODBC), NoSQL like Cassandra, etc.
- Supports read-through (missing) and write-through (synchronization)
- Full ACID, 2-Phase Commit synchronization



DIH Facilities – Integration: Data Streamer

Ignite DataStreamer – addData(k,v) or addData(map<k,v>)

- Buffered, Multi-threaded writes
- Topology aware dispatching to minimize network IO and memory impact
- Extensible receiver logic
- At-Least Once loading guarantees
- While it does not support transaction semantics …
  Implicit and user-defined explicit transactionality can be applied (no order guarantee)

  ← Fastest load mechanism supporting both bulk and incremental feeds

DIH Facilities – Event Handling: Continuous Query

Ignite ContinuousQuery – ContinuousQuery class provides the full class lifecycle—from initialization, through execution, to shutdown

- InitialQuery – the query initially invoked to get data & processed prior to steady state
- RemoteFilter – An optional element that may be set on remote node to avoid returning uninteresting data
- LocalListener – The logic to be used with the data returned from the query
- RemoteTransformer – An optional set of transformation logic that can be applied remotely in order to change what data is sent back to listener

DIH Facilities – Compute / Event Handling

Real-Time, Event-trigger Compute can be:

- **Distributed Closures** – broadcast and load-balance closure execution across cluster nodes
- **ComputeTask** – Execute MapReduce and ForkJoin tasks in memory
- **Affinity Call & Run** – Co-locate the computations with the data
- **ML/DL** – Classification, regression and clustering algorithms that can be called with partition-awareness for massive scalability and support continuous learning and model updates
- **Business Rules** – e.g. Drools deployed as a distributed closure to enable complex business rules to be developed by business audience

Ignite Digital Integration Hub

Implementing Data Ingest with Apache Ignite

GridGain

DIH Implement Data Ingest – Load Facilities

1. **Ignite APIs** – Cache put(k,v), putAll(map<k,v>) and SQL INSERT
   - Fine-grained APIs with sync and async semantics
   - Transactional and Ordered processing (for cache APIs, SQL MVCC is beta)

   ← Useful in application interfaces with incremental real-time integration patterns
2. **Ignite DataStreamer** – addData(k,v) or addData(map<k,v>)
   - Buffered, Multi-threaded writes
   - Topology aware dispatching to minimize network IO and memory impact
   - Extensible receiver logic
   - At-Least Once loading guarantees
   - Does not support transaction semantics, but Implicit and user-defined explicit transactionality can be applied (no order guarantee)

   ← Fastest load mechanism supporting both bulk and incremental feeds

DIH Implement Data Ingest – Load Facilities cont.

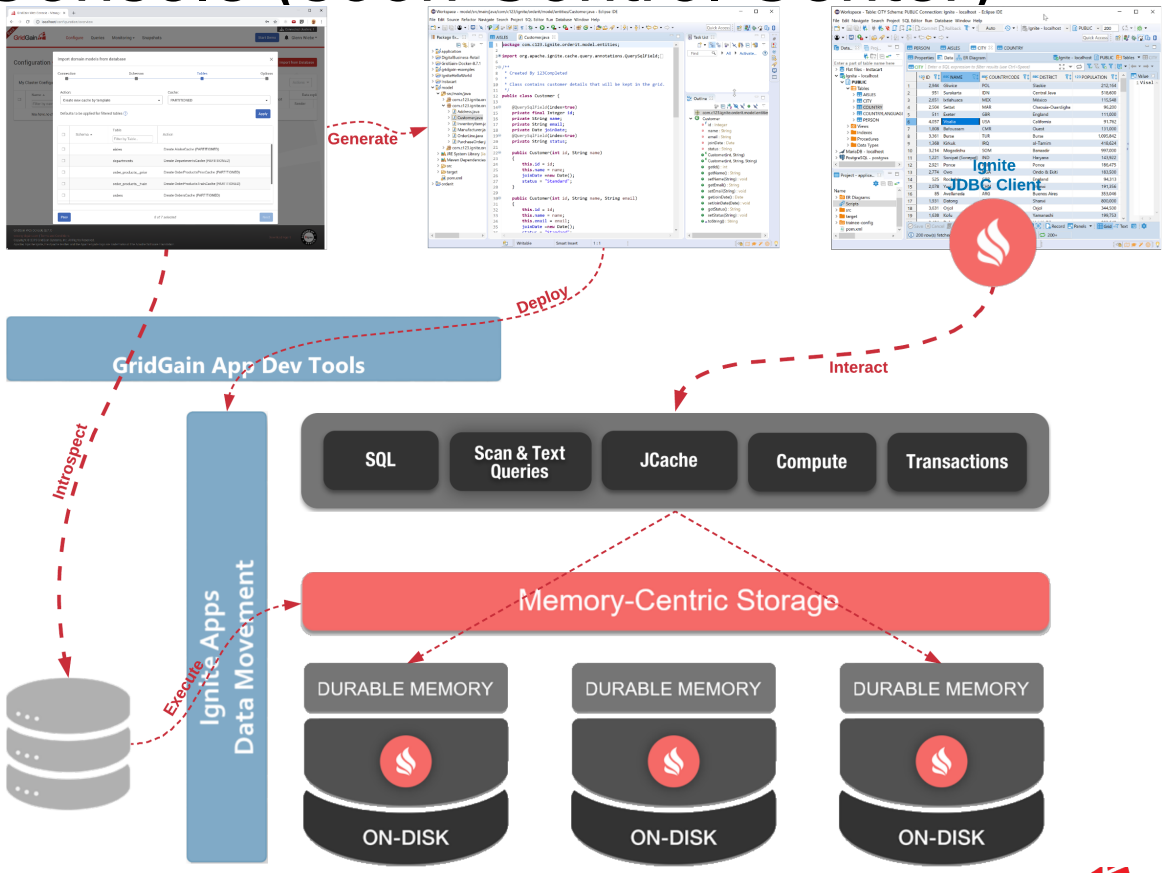
3. **IgniteCache.loadCache()** – cache store implementation to load from source
   - Executed on cache's configured CacheStore on each local node
   - Distributed, node-local loading pattern (i.e. client only initiates load, does no loading)
   - Supports IMDB integration (vs IMDG synchronization) via LoadOnly cache store type

   ← Useful for bulk, initial or batch loads, also event-driven loads (quasi incremental)
4. **Ignite & 3rd Party Tools** – sqlline, DBeaver, Talend, Informatica, etc.
   - Usually SQL JDBC/ODBC based
   - Supports streaming
   - With latest JDBC thin client, supports partition awareness (for directed writes and no intranode data shuffle)

   ← Great for integrating with existing tooling and organizational processes (and no code)

DIH Implement Data Ingest – Project Process

1. **Introspect** existing assets using GridGain Import Wizard
2. **Generate** Maven project using Web Console (soon Control Center)
3. **Customize, Build, Deploy & Run** cluster
4. **Execute** Data Load (i.e. ETL or data ingest)
5. **Interact** with IMDB (e.g. jdbc /odbc SQL client, python, spark, etc.)



DIH Implement Data Ingest – Process

1. Modify standard GridGain Web Console (soon Control Center) Ignite project
2. Adjust POM (e.g. add JDBC, Commons CSV, JSON libraries etc.)
3. Adjust properties file name and content (externalize deployment parameters)
4. Set business-specific class packages for all components
5. Remove Imported CacheStore ← More on this later
6. Create Load & Utility packages
7. LoadCachesFrom***XXX*** *(e.g. for CSV, LoadCachesFromCSV)*
8. ParseTypes (utility class to handle parsing different data types)

DIH Implement Data Ingest – Maven POM

pom.xml

gridgain-solutions > InternalDemos > SalesDataLoaders > pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.apache.ignite</groupId>
    <artifactId>SalesDataLoaders</artifactId>
    <version>0.0.3</version>

    <properties>
        <ignite.version>2.8.0</ignite.version>
        <gridgain.version>8.7.15</gridgain.versi
        <java.version>1.8</java.version>
    </properties>

    <repositories>
        <repository>
            <id>GridGain External Repository</id>
            <url>http://www.gridgainsystems.com/nexus/content/repositories/external</url>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>org.apache.ignite</groupId>
            <artifactId>ignite-core</artifactId>
            <version>${ignite.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.ignite</groupId>
            <artifactId>ignite-spring</artifactI
            <version>${ignite.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.ignite</groupId>
            <artifactId>ignite-indexing</artifac
            <version>${ignite.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.ignite</groupId>
            <artifactId>ignite-rest-http</artifa
            <version>${ignite.version}</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/
        <dependency>
            <groupId>org.apache.commons</groupId
            <artifactId>commons-csv</artifactId>
            <version>1.5</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
        <dependency>
            <groupId>mysql</groupId>
```

```xml
<properties>
    <ignite.version>2.8.0</ignite.version>
    <gridgain.version>8.7.15</gridgain.version>
    <java.version>1.8</java.version>
</properties>
```

```xml
<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-csv -->
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-csv</artifactId>
    <version>1.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.48</version>
</dependency>
```

1. Set Properties (versions)
2. Add CSV, JDBC, other libraries

GridGain

DIH Implement Data Ingest – Properties File

sales.properties

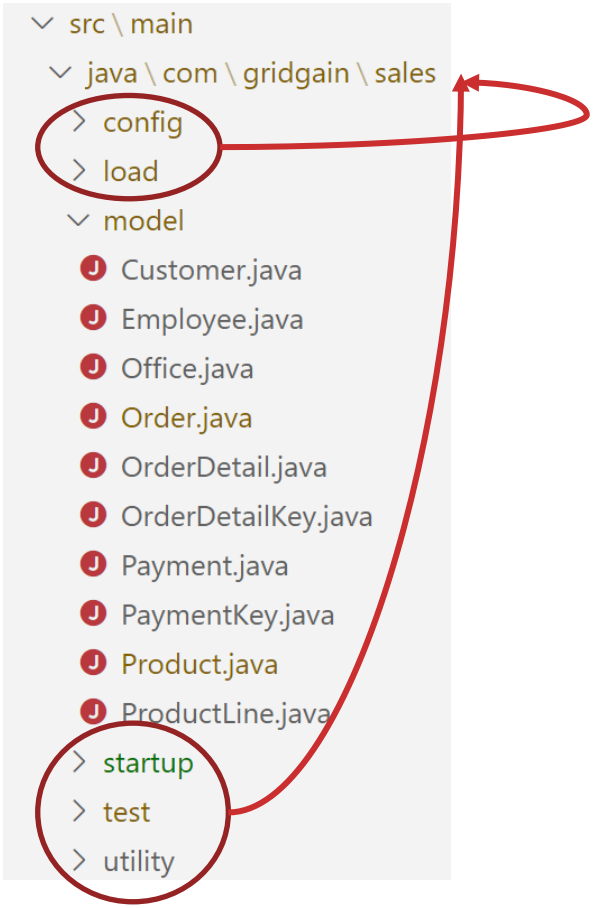
gridgain-solutions > InternalDemos > SalesDataLoaders > src > main > resources > sales.properties

```
1  # This file was generated by Ignite Web Console (04/23/2020, 16:26)
2
3  dsMySQL_Classicmodels.jdbc.url=jdbc:mysql://localhost:3306/classicmodels
4  dsMySQL_Classicmodels.jdbc.username=root
5  dsMySQL_Classicmodels.jdbc.password=Password1
6
7  dataLocation=/code/gridgain-solutions/InternalDemos/SalesDataLoaders/Data
```

Example uses:

1. Data source properties for JDBC
2. Add CSV, JDBC, other libraries

DIH Implement Data Ingest – Java Packaging



Object Model Package:

1. To organize domain specific objects
2. Common facilities (startup, load, config) are not organized that would allow multiple projects to share a runtime libs folder (classes would conflict)
3. We will move these artifacts under the package organized directory structure

DIH Implement Data Ingest – Cache Store

```
CacheJdbcPojoStoreFactory cacheStoreFactory = new CacheJdbcPojoStoreFactory();
cacheStoreFactory.setDataSourceFactory(new Factory<DataSource>() {
    /** {@inheritDoc} **/
    @Override public DataSource create() {
        return DataSources.INSTANCE_dsMySQL_Classicmodels;
    };
});

cacheStoreFactory.setDialect(new MySQLDialect());
cacheStoreFactory.setTypes(jdbcTypeCustomer(ccfg.getName()));

ccfg.setCacheStoreFactory(cacheStoreFactory);
ccfg.setReadThrough(true);
ccfg.setWriteThrough(true);
ccfg.setEagerTtl(true);
```

```
<property name="cacheStoreFactory">
    <bean class="org.apache.ignite.cache.store.jdbc.CacheJdbcPojoStoreFactory">
        <property name="dataSourceBean" value="dsMySQL_Classicmodels"/>
        <property name="dialect">
            <bean class="org.apache.ignite.cache.store.jdbc.dialect.MySQLDialect">
            </bean>
        </property>

        <property name="types">
```

Imported CacheStore is used in IMDG scenario to synchronize data from and to a data source.

Ignite CacheStores implement:

- Insert/Update/Delete Data Sync
- Load (from cache store source)

With IMDB we don't want to synchronize – we want to load.

Remove imported CacheStore from project in Code & SpringBean configs

DIH Implement Data Ingest – Utility

```
1   package com.gridgain.sales.utility;
2
3   import java.math.BigDecimal;···
12
13  public class ParseTypes {
14
15      public static Double parseDouble(String strNumber) {···
26      public static Float parseFloat(String strNumber) {···
37      public static BigDecimal parseBigDecimal(String strNumber) {···
49      public static Integer parseInteger(String strNumber) {···
60      public static Short parseShort(String strNumber) {···
71      public static Date parseDate(String strDate) {···
84      public static Timestamp parseTimestamp(String strDate) {···
97      public static Date parseTimestampToDate(String strDate) {···
111     public static Time parseTime(String strTime) {
112         Time retVal = null;
113         if (strTime != null && strTime.length() > 0) {
114             try {
115                 retVal = java.sql.Time.valueOf(strTime);
116                 return retVal;
117             } catch(Exception e) {
118                 return retVal;
119             }
120         }
121         else return retVal;
122     }
123
124 }
```

Reading from files, data starts as text/string format and needs to be converted to native/binary formats.

Here we have a set of routines that can be used by the many projects and classes to perform that logic.

DIH Implement Data Ingest – DataStreamer CSV Files

LoadCachesFromCsv.java

gridgain-solutions > InternalDemos > SalesDataLoaders > src > main > java > com > gridgain > sales > load > LoadC

```
53  public class LoadCachesFromCsv {
54      /** ...
        Run | Debug
60      public static void main(String[] args) throws IgniteException, IOException {
61          DecimalFormat numFormat = (DecimalFormat)NumberFormat.getCurrencyInstance();
62          String symbol = numFormat.getCurrency().getSymbol();
63          numFormat.setNegativePrefix("-"+symbol);
64          numFormat.setNegativeSuffix("");
65          CSVParser csvParser = null;
66
67          try (Ignite ignite = Ignition.start("sales-client.xml")){
68              System.out.println(">>> CSV Stream Loading caches:");
```

A. Start with a standard Ignite Client:
   `Ignition.start()`

B. Create File Reader & CSV Parser

C. Look at the Object constructor (OrderDetail.java)

D. Create matching Key & Value Object

E. feed to Data Streamer!

```
271             /*
272              * OrderDetail
273              **/
274             System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>
275             reader = Files.newBufferedReader(Paths.g
276             csvParser = new CSVParser(reader,
277                 CSVFormat.DEFAULT
278                 .withEscape('\\')
279                 .withQuoteMode(QuoteMode.NONE)
280                 .withFirstRecordAsHeader()
281                 .withTrim()
282             );
```

```
29      /** Full constructor. **/
30      public OrderDetail(int quantityordered,
31          BigDecimal priceeach,
32          short orderlinenumber) {
```

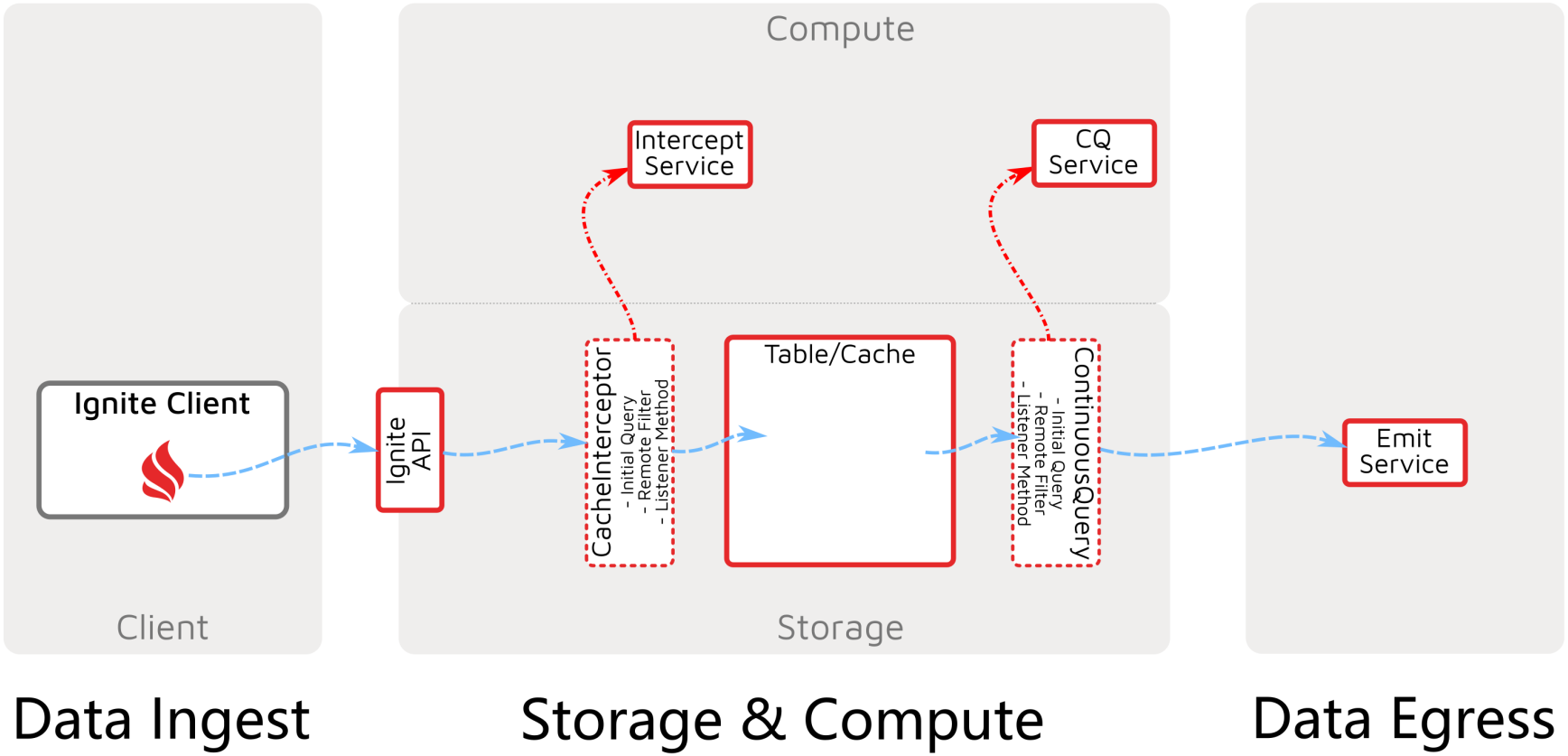
```
284         try (IgniteDataStreamer<OrderDetailKey, OrderDetail> streamer = ignite.dataStreamer("OrderDetailCache")){
285             for (CSVRecord csvRecord : csvParser) {
286                 OrderDetailKey k = new OrderDetailKey (
287                     ParseTypes.parseInteger(csvRecord.get(0)),
288                     csvRecord.get(1)
289                 );
290                 OrderDetail v = null;
291                 try {
292                     v = new OrderDetail(
293                         ParseTypes.parseInteger(csvRecord.get(2)),
294                         ParseTypes.parseBigDecimal(csvRecord.get(3)),
295                         ParseTypes.parseShort(csvRecord.get(4))
296                     );
297                 } catch (NumberFormatException e) {
298                     e.printStackTrace();
299                 }
300                 streamer.addData(k, v);
301             }
302         }
```

Ignite Digital Integration Hub

Real-Time Event Processing with Continuous Query

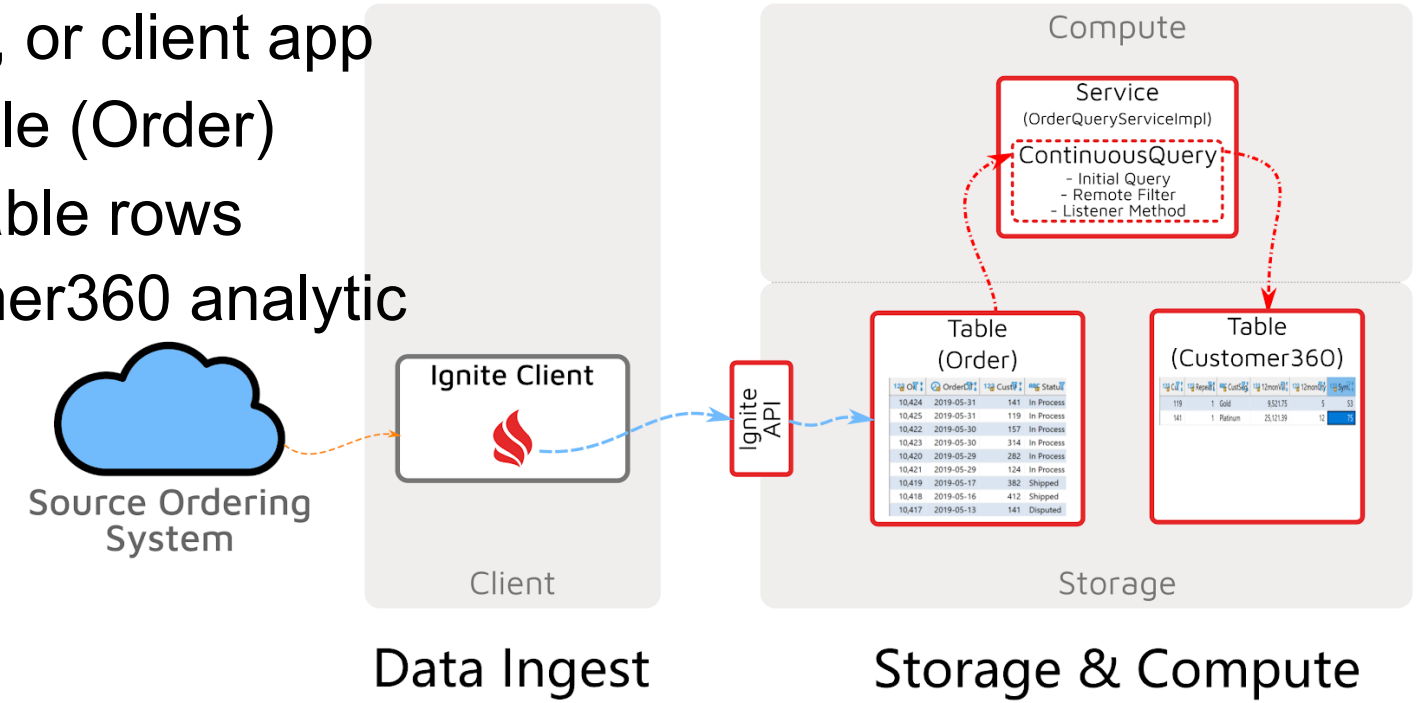
DIH Implement Real-Time Events – Facilities

- Real-Time Data Ingest APIs (e.g. Put, Insert, Stream)
- Real-Time Data Handling:
  - Cache Interceptor
  - Continuous Query
- Real-Time Services



DIH Implement Real-Time Events – Continuous Query

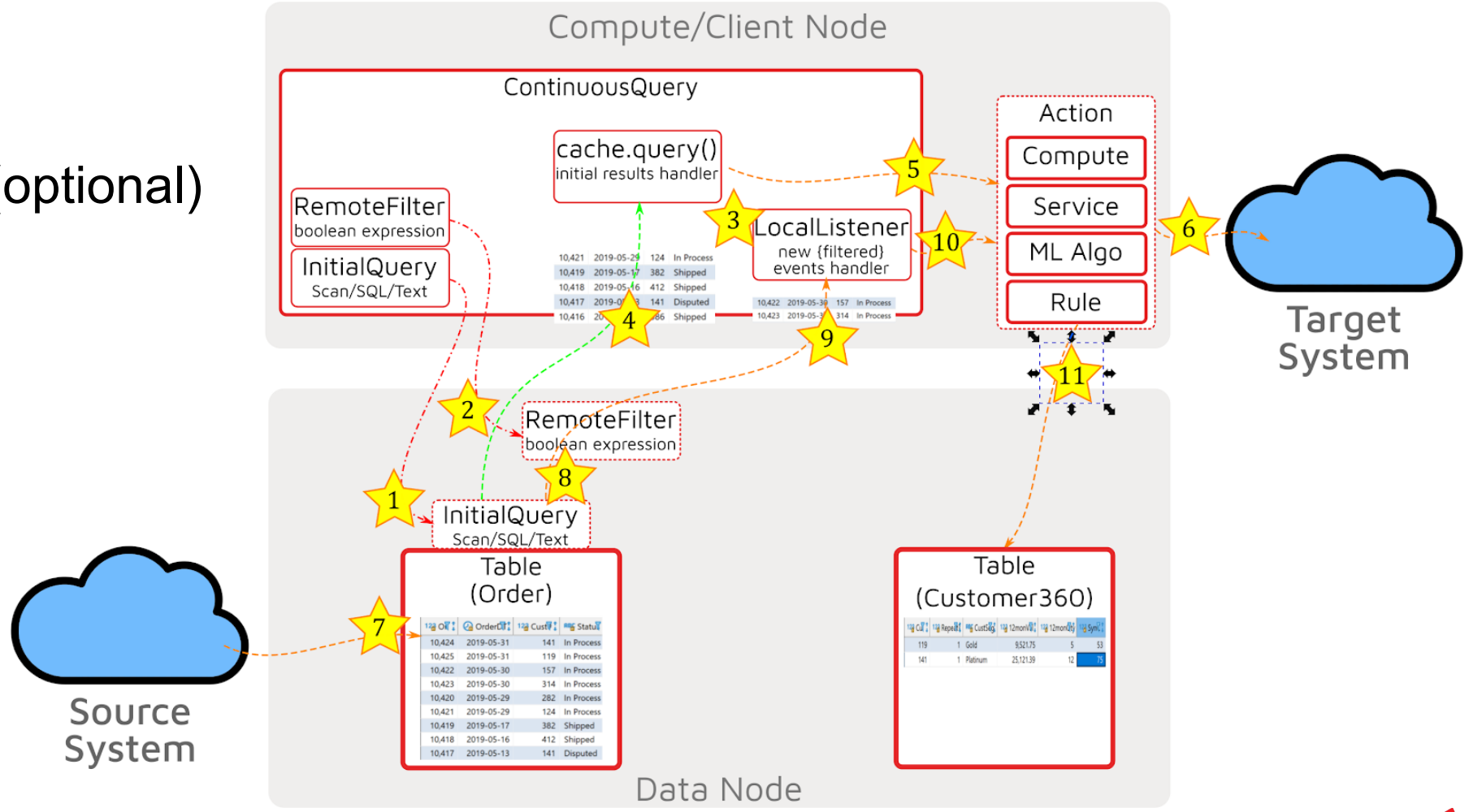
- Order Handling Use-Case
- Continuous Query deployed as a service invoked
- Data Ingest handled by some tool, or client app
- Data loaded/added/inserted to table (Order)
- Continuous Query retrieve applicable rows
- Logic applied, e.g. update Customer360 analytic table.



GridGain

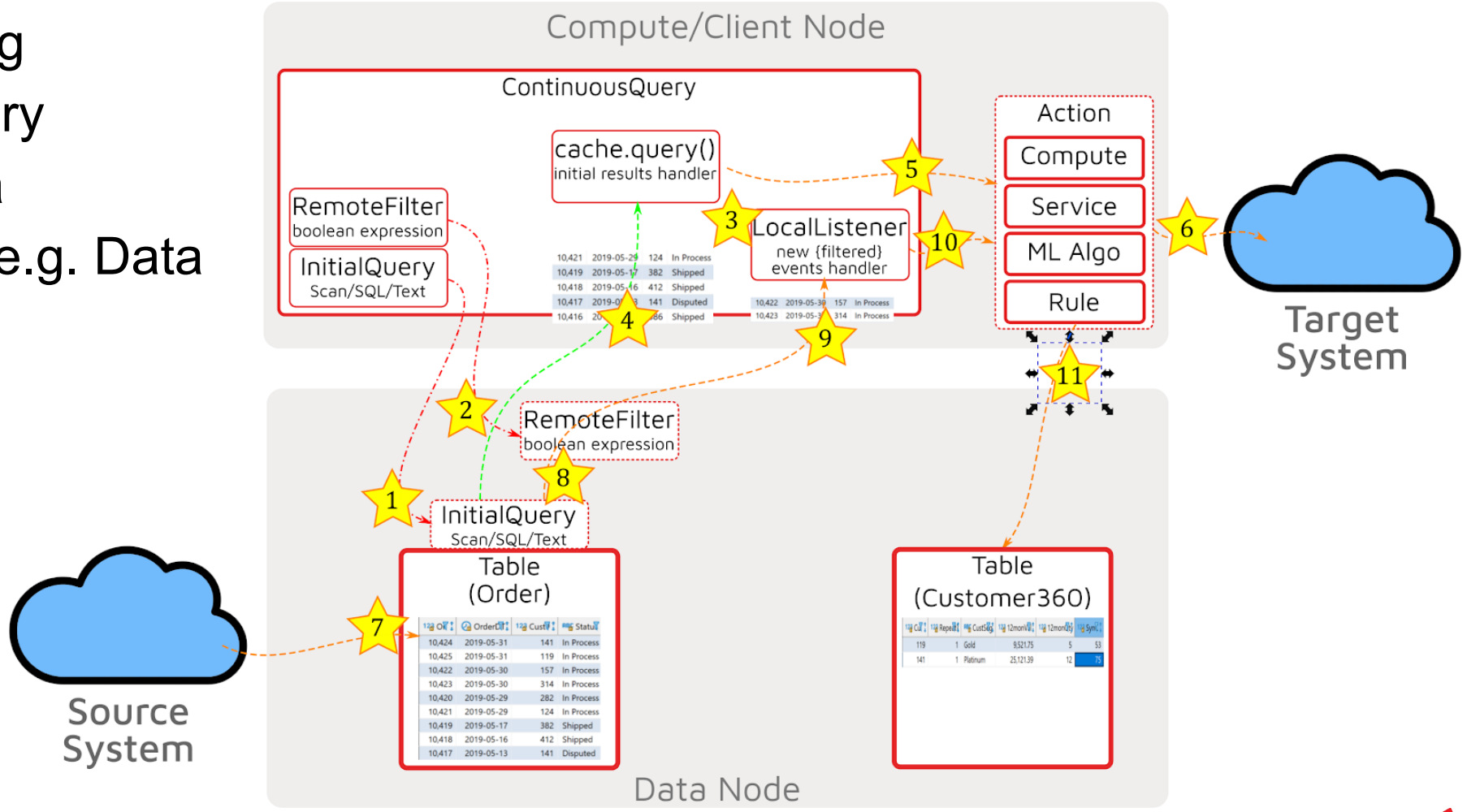
DIH Implement Real-Time Events – CQ Flow: Setup

A. Setup
   1. Set InitialQuery
   2. Set RemoteFilter (optional)
   3. Set LocalListener



DIH Implement Real-Time Events – CQ Flow: Initial Rows

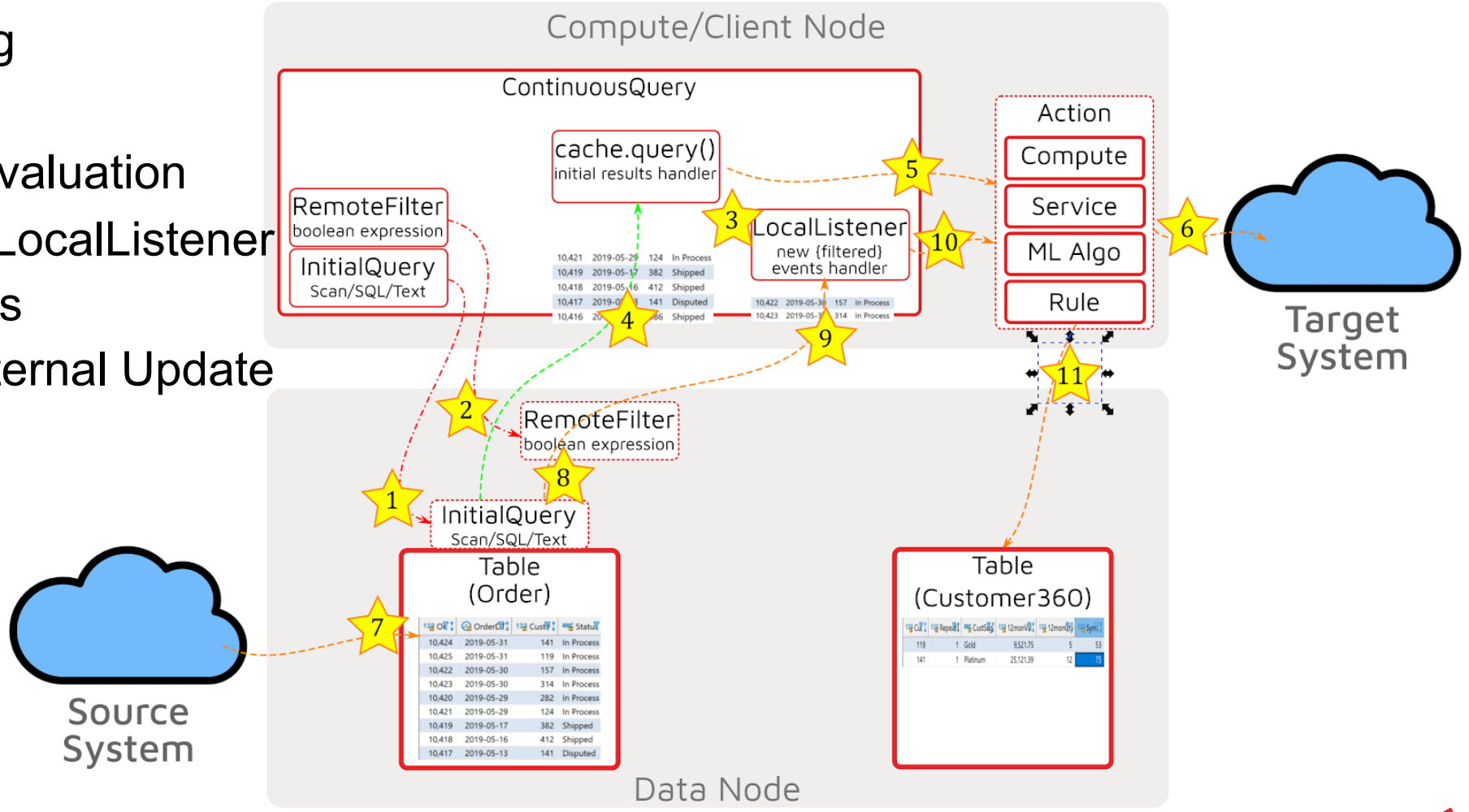
B. Initial Data Handling
   4. Execute InitialQuery
   5. Handle Initial Data
   6. Optional Result – e.g. Data Egress



GridGain

DIH Implement Real-Time Events – CQ Flow: Steady State

C. Steady State Handling
   7. Business Change
   8. Opt. Remote Filter Evaluation
   9. Entries Delivered to LocalListener
   10. LocalListener Actions
   11. Optional Result – Internal Update



DIH Implement Real-Time Events – CQ Coding

```java
/**...
public interface QueryService extends Service {
    /** Service name */
    public static final String SERVICE_NAME = "QueryService";

    /**...
    public void startQuery();

    /**...
    public void stopQuery();

    /**...
    public void adjustFilter(Date
```

```java
public class OrderQueryServiceImpl implements QueryService {

    @IgniteInstanceResource
    private Ignite ignite;

    private ServiceContext svcCtx;

    /* Static Cache Name */
    private static String CACHE_NAME = "OrderCache";

    /* Last order number to continue from */
    private Integer LastOrderNumber = -1;

    /* Last order number to continue from */
    private Date asOfDate = Date.valueOf("2020-07-12");

    /* OrderQuery Listener Active state */
    private String ListenerState = "initializing";

    /** {@inheritDoc} */
    public void init(ServiceContext ctx) throws Exception {...

    /** {@inheritDoc} */
    public void execute(ServiceContext ctx) throws Exception {...

    /** {@inheritDoc} */
    public void cancel(ServiceContext ctx) {...

    @Override
    public void startQuery() {...

    @Override
    public void stopQuery() {...

    @Override
    public void adjustFilter(Date filterDate) {...
```

A. Define ContinuousQuery “QueryService”

B. Implement with OrderQueryServiceImpl

- Execute Method for stateful handling of methods

```java
String initialQuerySQL = "SELECT * FROM SALES.\"ORDER\" WHERE orderdate > ?";
ordQuery.setInitialQuery(
    new SqlQuery<Integer, Order>(Order.class, initialQuerySQL).setArgs(asOfDate)
);

// Callback that is called locally when update notifications are received.
ordQuery.setLocalListener(new CacheEntryUpdatedListener<Integer, Order>() {
    @Override
    public void onUpdated(Iterable<CacheEntryEvent<? extends Integer, ? extends Order>> evts) {
        for (CacheEntryEvent<? extends Integer, ? extends Order> e : evts)
            System.out.println(">>> OrderQueryServiceImpl: execute(): >>>>>> onUpdated: Order: [OrderID=" + e.getKey() + ", Order=" + e.getValue() + ']');
    }
});

// This filter will be evaluated remotely on all nodes.
// Entry that pass this filter will be sent to the caller.
ordQuery.setRemoteFilterFactory(
    new Factory<CacheEntryEventFilter<Integer, Order>>() {
        @Override
        public CacheEntryEventFilter<Integer, Order> create() {
            return new CacheEntryEventFilter<Integer, Order>() {
                @Override
                public boolean evaluate(CacheEntryEvent<? extends Integer, ? extends Order> e) {
                    System.out.format("the value for key [%s] was updated from [%s] to [%s]\n", e.getKey(), e.getOldValue(), e.getValue());
                    return (e.getKey() > LastOrderNumber); // if the entry is greater than the last, then true and return it
                    // return true;
                }
            };
        }
    }
);

// Execute query; inside try with resource, so query closes on end of try block
try (QueryCursor<Cache.Entry<Integer, Order>> cur = cache.query(ordQuery)) {
    System.out.println(">>> OrderQueryServiceImpl: execute(): >>>>>> cache.query() called...");

    // Iterate through existing data.
    for (Cache.Entry<Integer, Order> e : cur) {
        System.out.println(">>> OrderQueryServiceImpl: execute(): >>>>>> Orders since last run:  [key=" + e.getKey() + ", val=" + e.getValue() + ']');
    }

    // place this in a loop checking for status of Quer Listener Service
    while (ListenerState.equalsIgnoreCase("active")) {
        // Wait for a while while callback is notified about remaining puts.
        // System.out.println(">>> OrderQueryServiceImpl: execute(): >>>>>> wait for more orders to come in...");
        Thread.sleep(2000);
        System.out.println(">>> OrderQueryServiceImpl: execute(): >>>>>> check if ListenerState is still \"active\"...");
    }
}
```

Digital Integration Hub (DIH) Platform

Developing, Delivering, Managing & Monitoring an Ignite DIH

Ignite DIH Platform – Development & Delivery

Development

- Full API development in Java, .NET/C#, C++
- Thin Client API development in Java, Python, C#, C++, Node.js, PHP and others
- Cluster and Database Import Wizard config generator (including Dockerfile)

Deployment

- JVM Stand-alone or Embedded deployments (on all operating systems)
- DockerHub-based Docker containers
- Kubernetes deployment with k8s-specific discovery and communication protocols

Ignite DIH Platform – Monitoring & Management

Monitoring

- Command line, REST, and JMX-based monitoring API
- GUI (“Visor”) & Web-based monitoring tools (Web Console and new Control Center)
- New OpenCensus-based monitoring with over 200+ performance metrics
- New Unified Trace facility to consolidate all cluster activity into logical activities

Management & Consumption

- Web-based cluster management for:
- Cache management, node & baseline definition, snapshot scheduling, backup & restore, alert definition, data center replication, etc.
- SQL & Cache metadata exploration & notebooking for zero-install platform consumption

Ignite DIH Platform – Monitoring with Control Center

Comprehensive Tool

- Cluster monitoring, management and interactive developer tool for GridGain and Apache Ignite

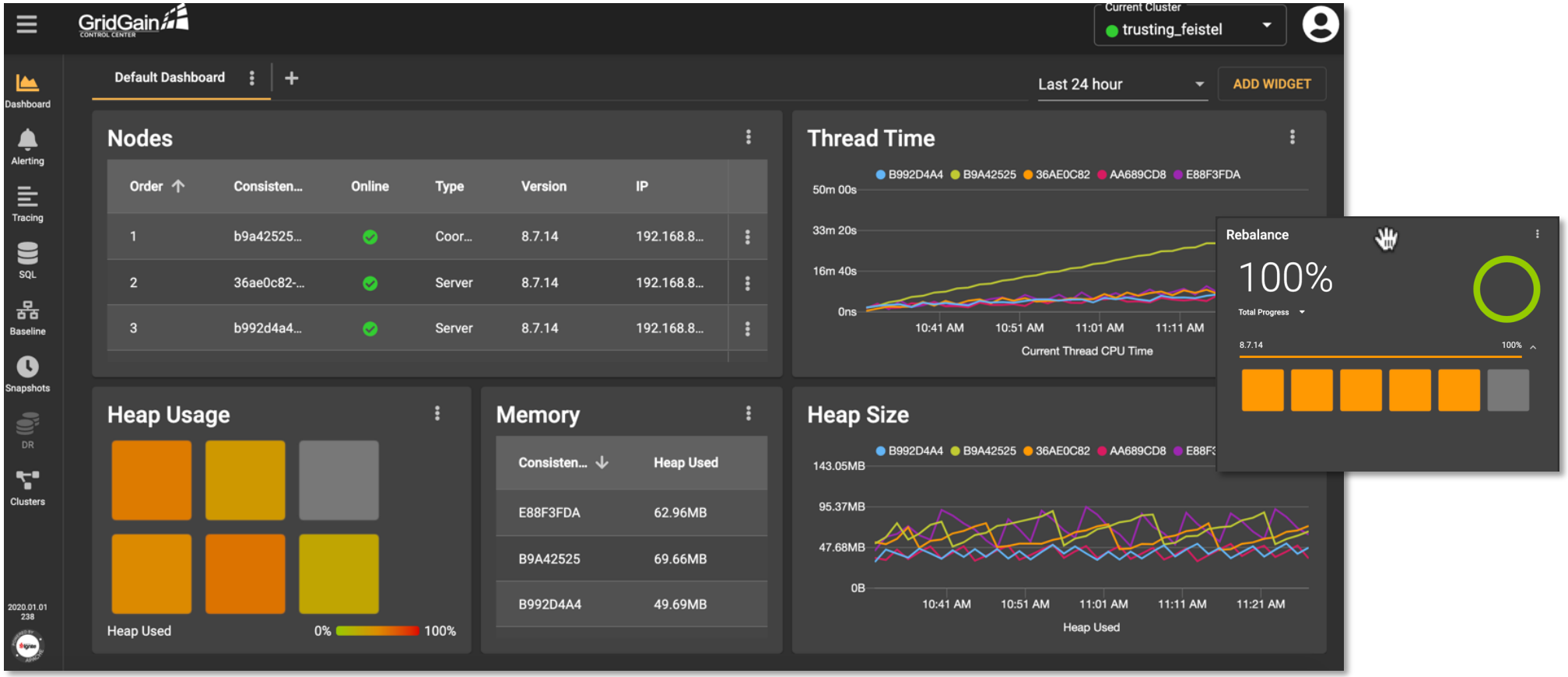
Supported Products

- Built for GridGain 8.7+ and Apache Ignite 2.8+
  - Apache Ignite requires Control Center Agent install

Three Versions

- SaaS version
- Downloadable developer version
- Downloadable on-premises version for comprehensive, shared cluster management

Ignite DIH Platform – Monitoring with Control Center



Summary and Q&A

Summary - Apache Ignite Digital Integration Hub

Apache Ignite can be deployed as a Digital Integration Hub, which will

- Deliver a technical framework for digital organizations
- Provides all core layers of an API system patterned for real-time exchange
- By delivering:
  - Data Storage – to persist data while supporting OLTP & OLAP needs (aka HTAP)
  - Real-Time Compute & Event Handling – to build business logic and services for consumers (users and system agents)
  - Integration – to enable tools, services and facilities to ingest and egress data from DIH

Questions

- Prior Questions during preceding presentation
- New Questions now?

GridGain