High Performance Architectures for IoT

Ultimate Speed and Scale for Distributed Systems

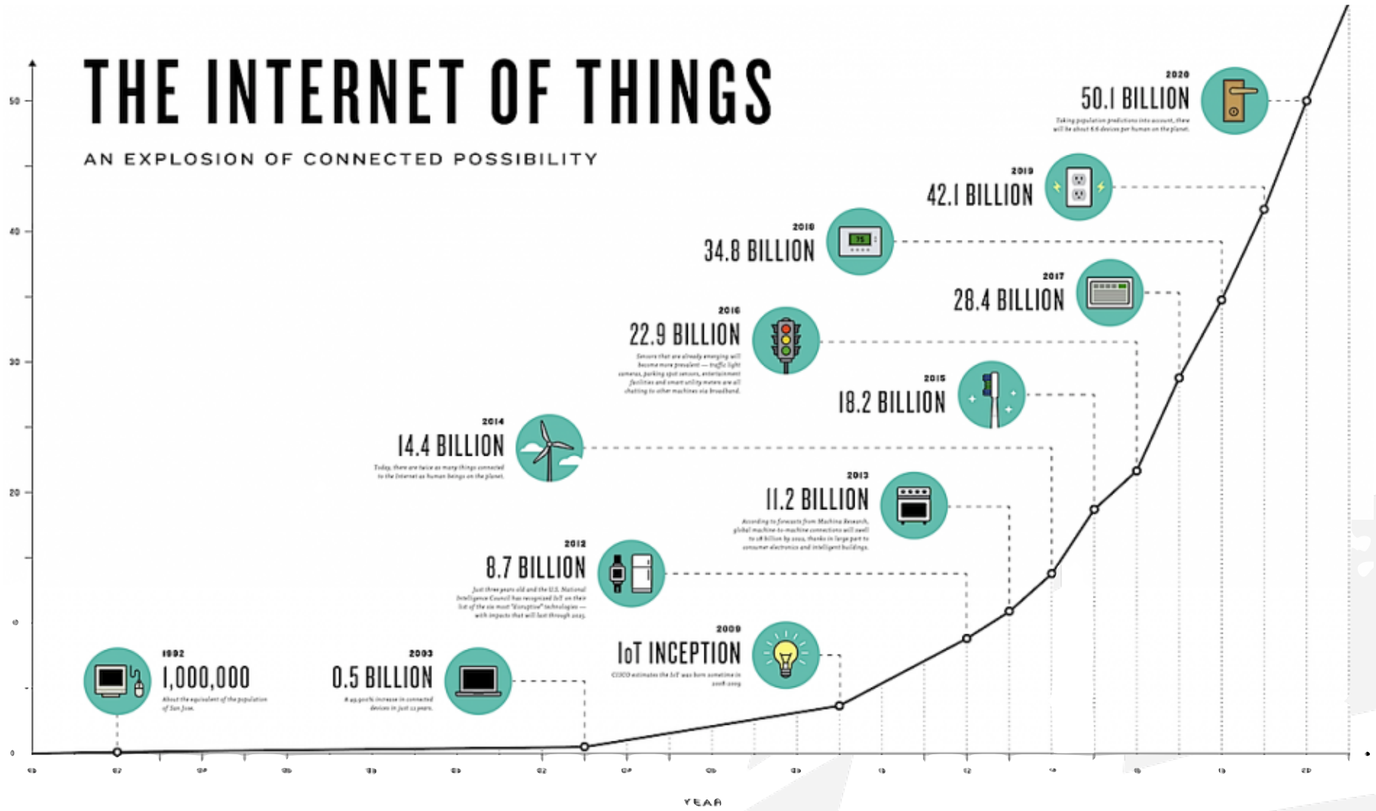
Author: Christos Erotocritou

Agenda

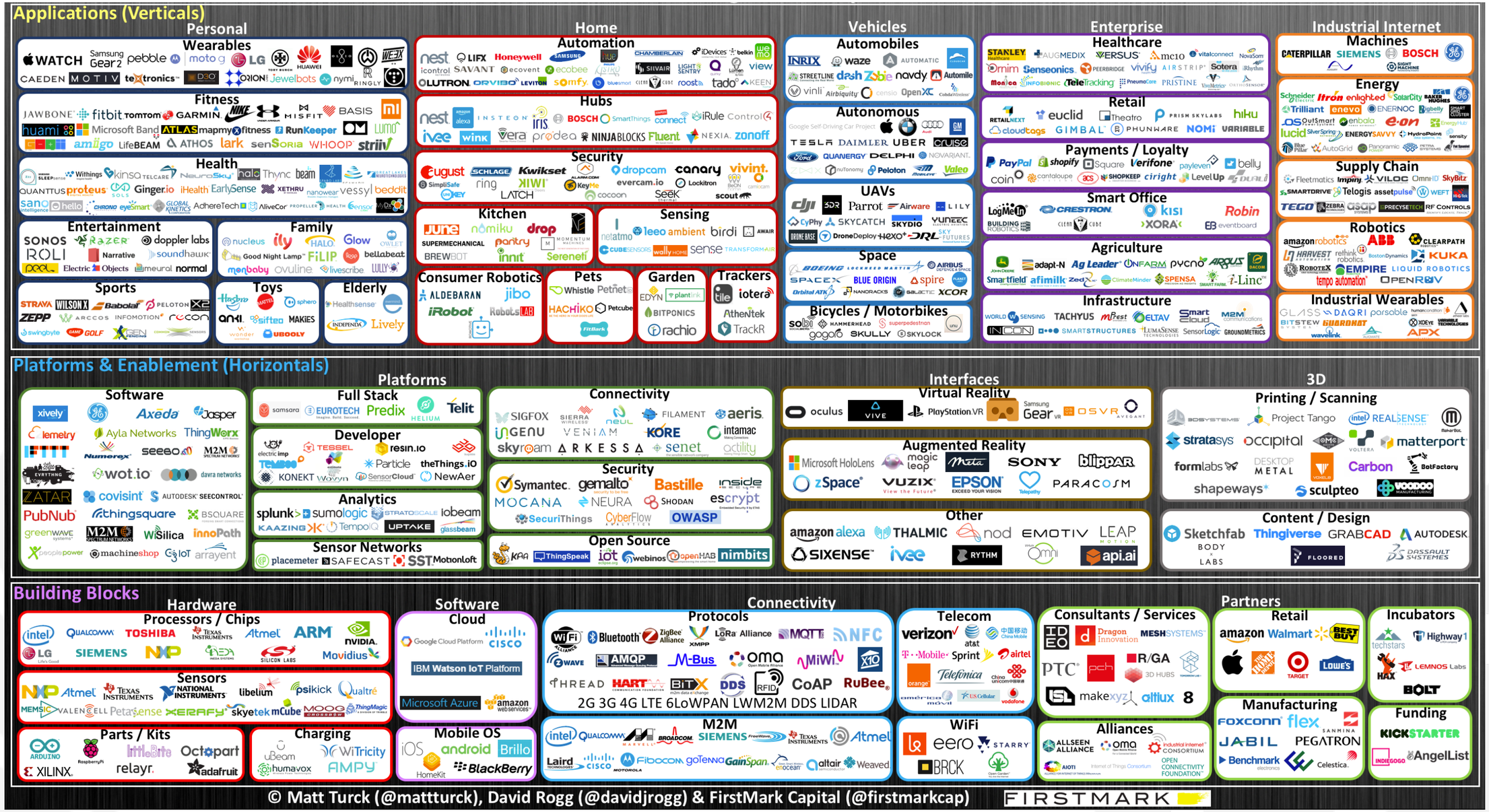
- Current IoT Landscape
- IoT use cases & technology stacks
- About Apache Ignite & GridGain
- GridGain for high-performance IoT
  - Scalability & Resilience
  - Storage & Caching
  - Hybrid Workloads (HTAP)
  - Streaming & CEP
  - Spark Integration
- Open Q & A

The IoT Timeline

- **2010:** Micro-controllers & smart devices
- **2013:** Wearables & home automation
- **2018:** 50% internet traffic from things
- **2018:** 100M Wearable device shipments
- **2020:** Over 30 billion connected things
- **2024:** Internet of flying things
- **2025:** All cars connected to the internet



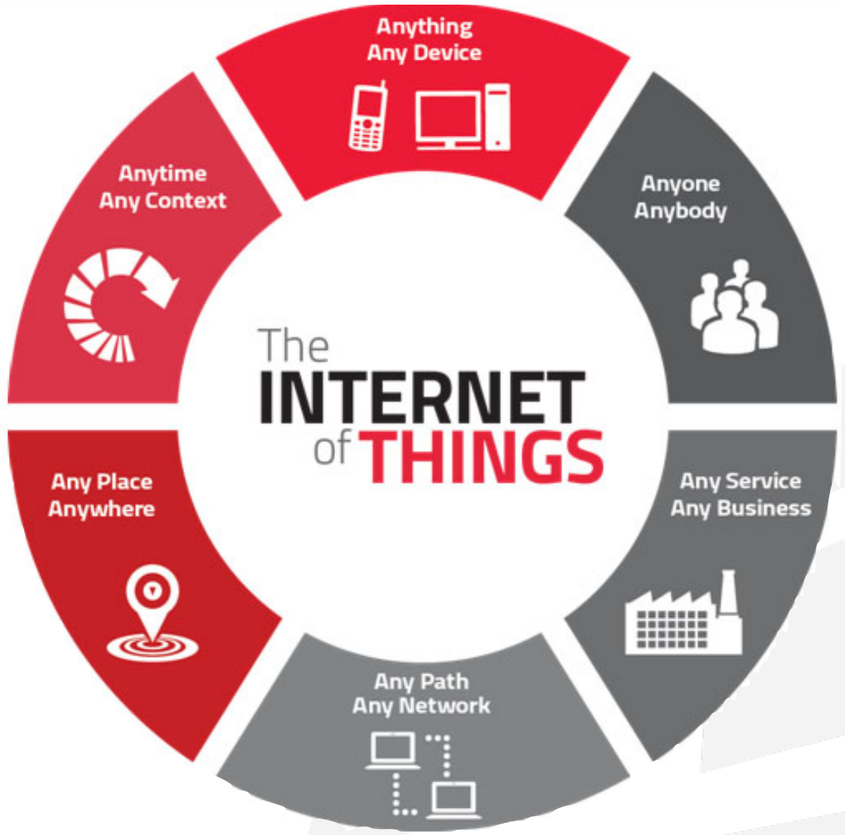
IoT Landscape 2016



Apache Ignite GridGain

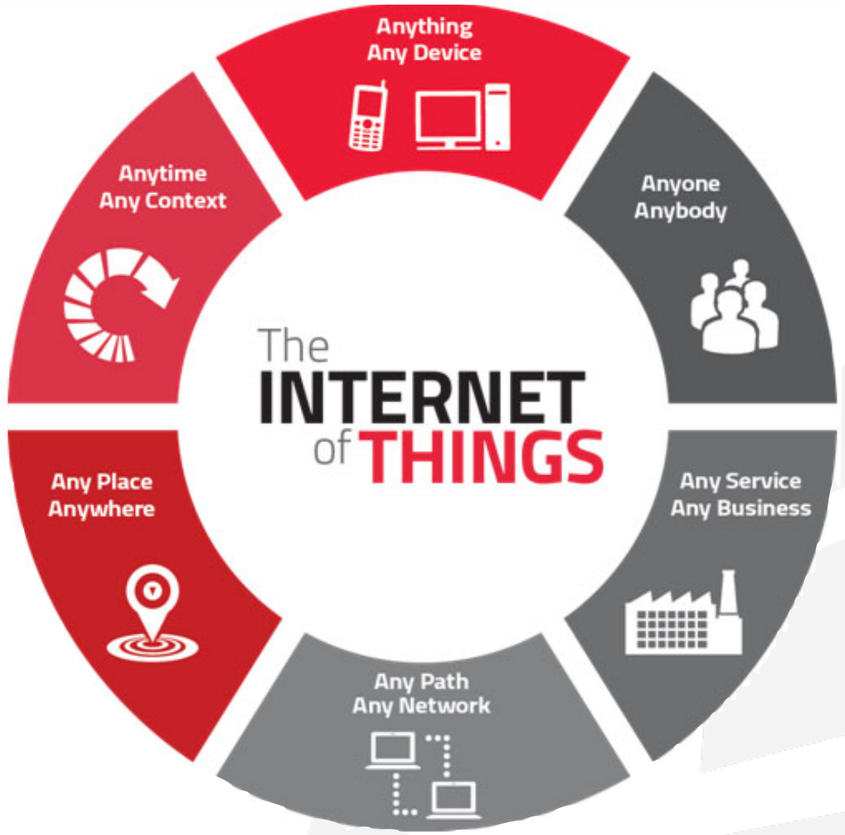
Popular IoT Use Cases

- **Smart Homes:**
  - Smart meters & home automation
- **Wearables:**
  - Clothes, watches, find-my-keys
  - Virtual & Augmented Reality & HCI
- **Manufacturing, Retail & Industrial use:**
  - Sensory tracking for production line monitoring
- **Transportation:**
  - Intelligent traffic control
  - Self-driving vehicles



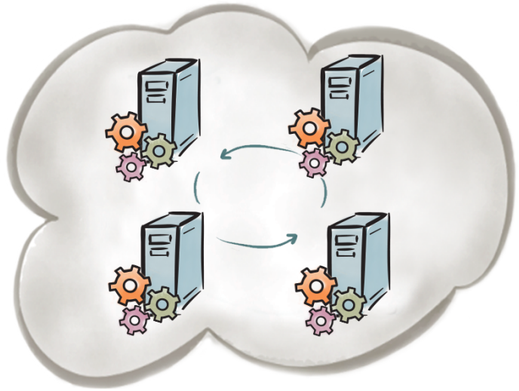
IoT Challenges

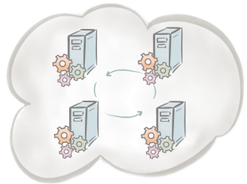
- **Scale:** Billions of connected devices
- **Speed:** High-frequency operations and transactions
- **Distribution:** Any place & anytime (geo-distribution)
- **Security:** Privacy Protection
- **Omni-channel:** Any device & any path (omni-session)
- **Variable Workloads:** Analytical & Transactional



Typical IoT Architecture

High-speed distributed cloud server-side infrastructure





Multi-site deployment

Direct device integration through TCP/UDP

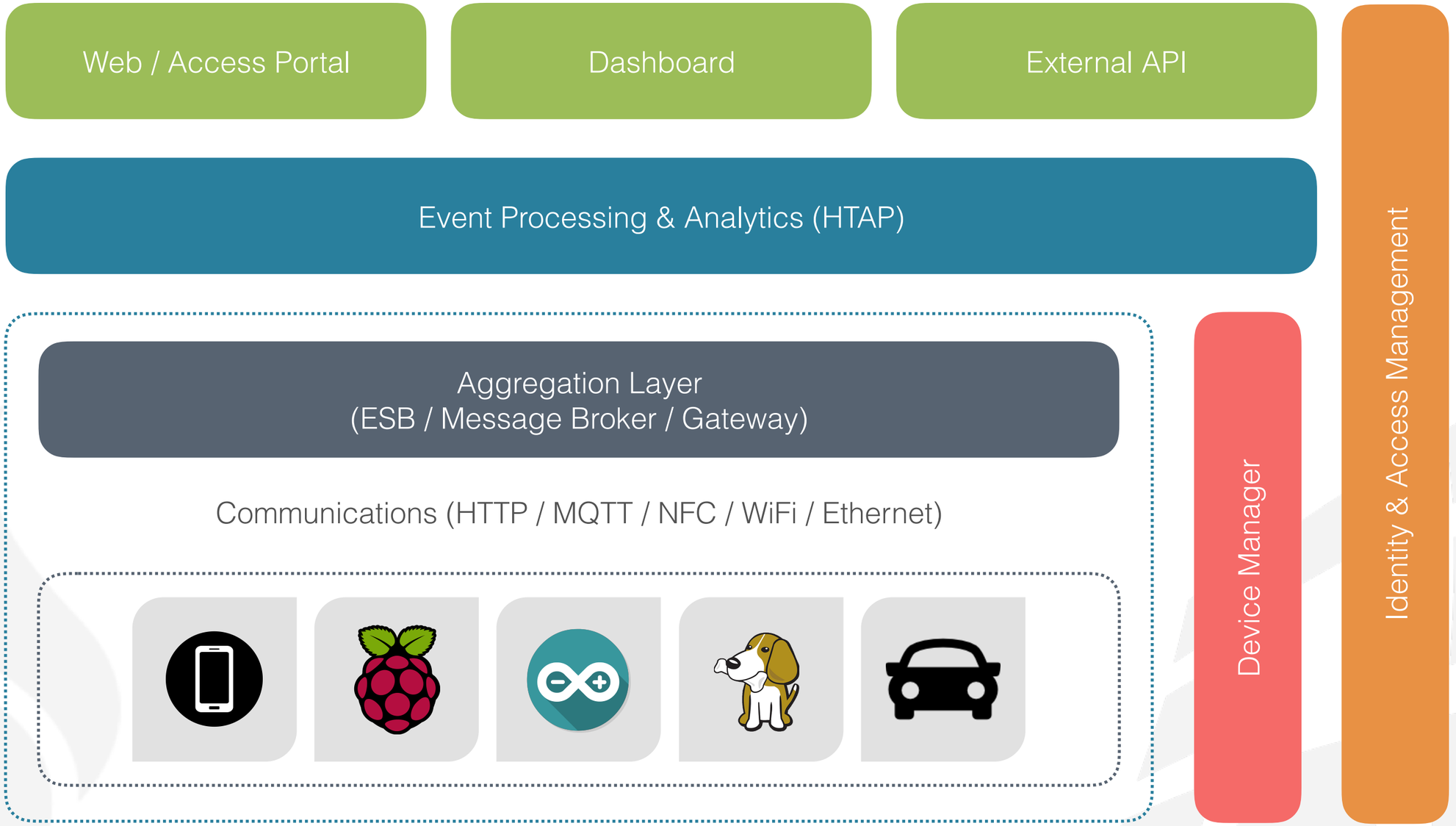
WiFi / Ethernet

High-speed Gateway

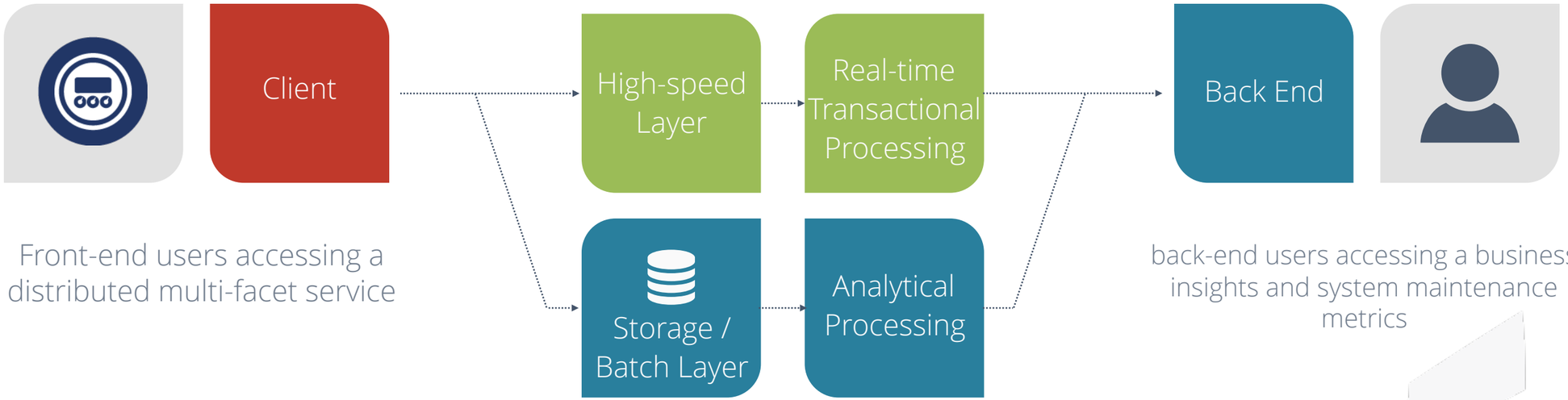
Device integration through gateways using NFC, RFID etc.

"Things" & devices

IoT Reference Architecture



Lambda Architecture

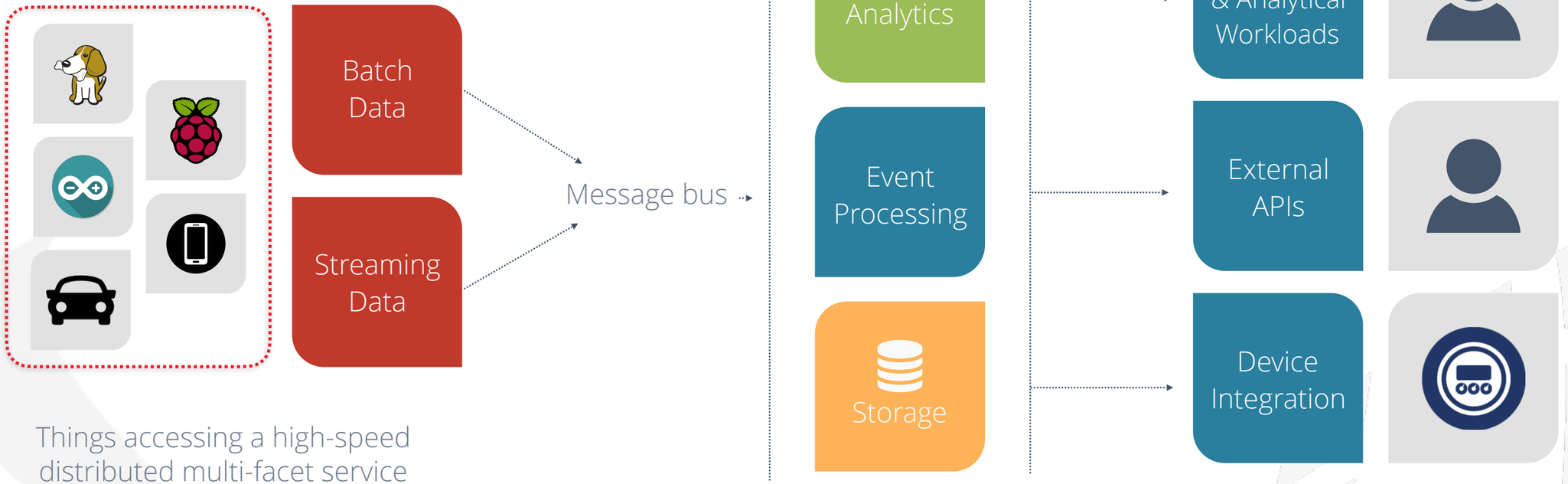


Front-end users accessing a distributed multi-facet service

back-end users accessing a business insights and system maintenance metrics

Transactional & Analytical workloads

Lambda Architecture for IoT

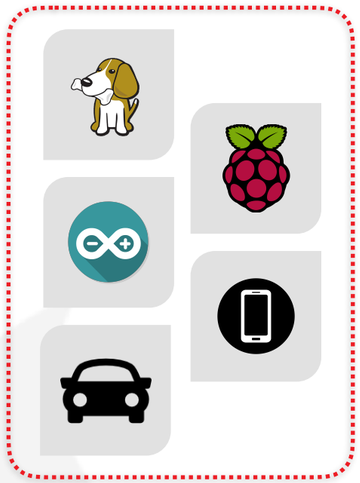


Things accessing a high-speed distributed multi-facet service

Transactional & Analytical workloads

Back-end users, third-party clients and devices

Hybrid Transaction / Analytical Processing (HTAP)



Batch Data

Streaming Data

Things accessing a high-speed distributed multi-facet service

?

- Transactional & Analytical workloads
- Persistency
- Event Processing

Transactional & Analytical Workloads

External APIs

Device Integration

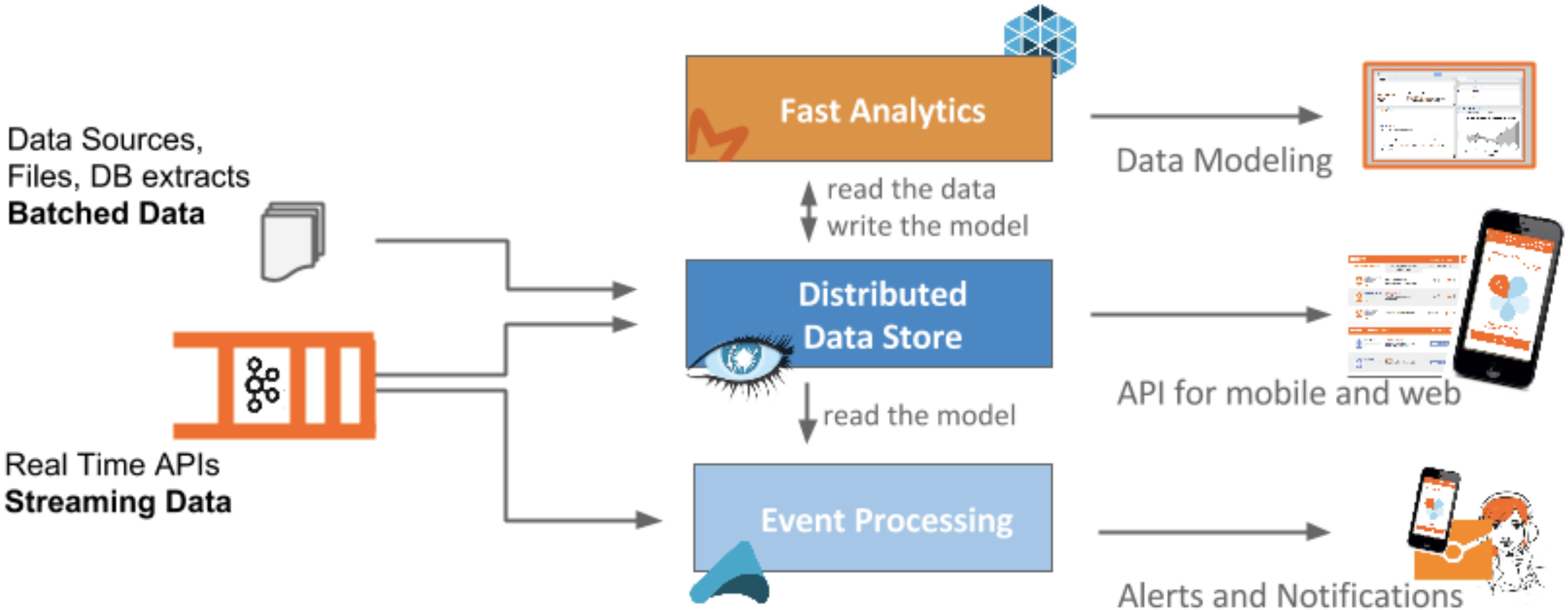
Back-end users, third-party clients and devices

Hybrid Transaction / Analytical Processing (HTAP)

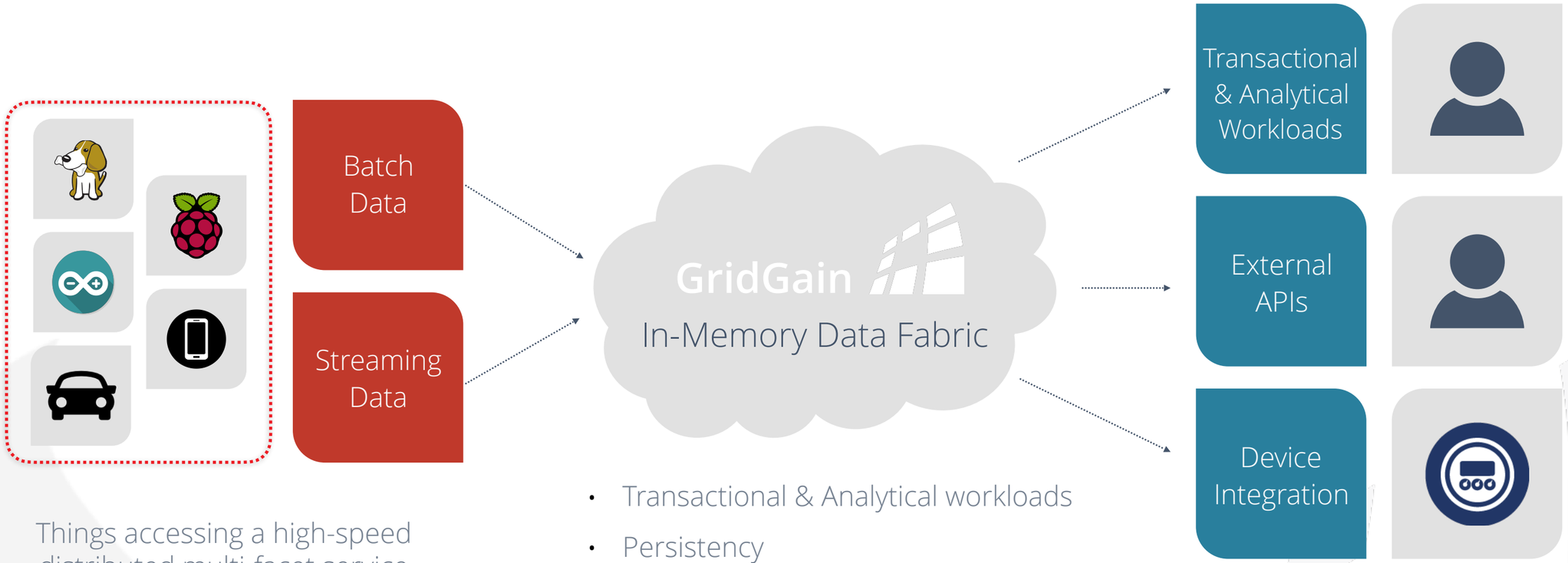
> *Hybrid transaction/analytical processing will empower application leaders to innovate via greater situation awareness and improved business agility. This will entail an upheaval in the established architectures, technologies and skills driven by use of in-memory computing technologies as enablers.*

Gartner 2014

The SMACK Stack



High-Speed Data & Compute Layer

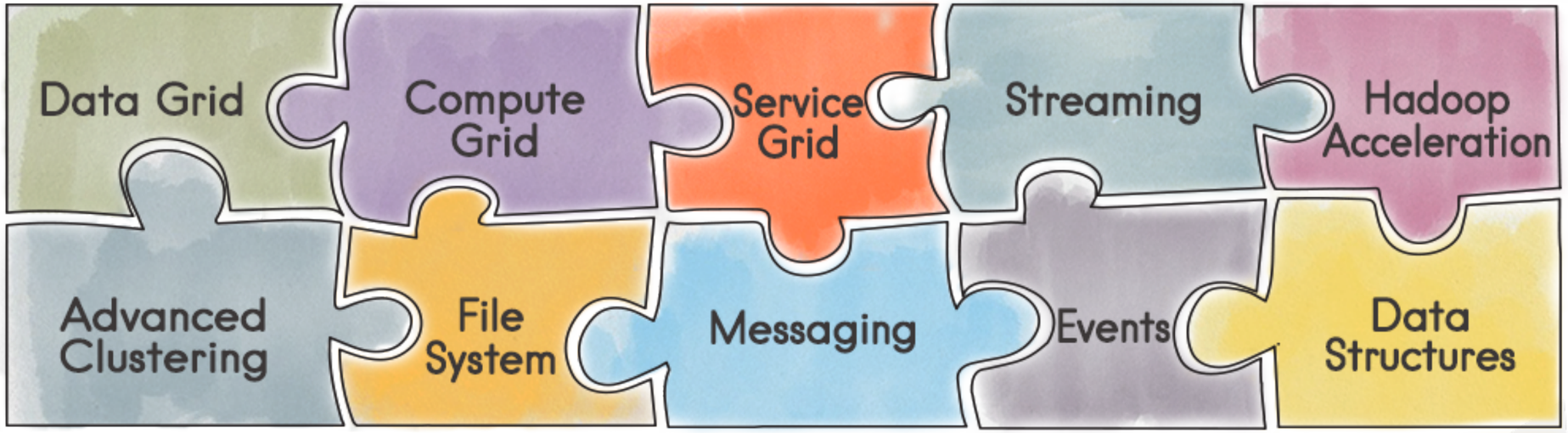


Things accessing a high-speed distributed multi-facet service

- Transactional & Analytical workloads
- Persistency
- Event Processing

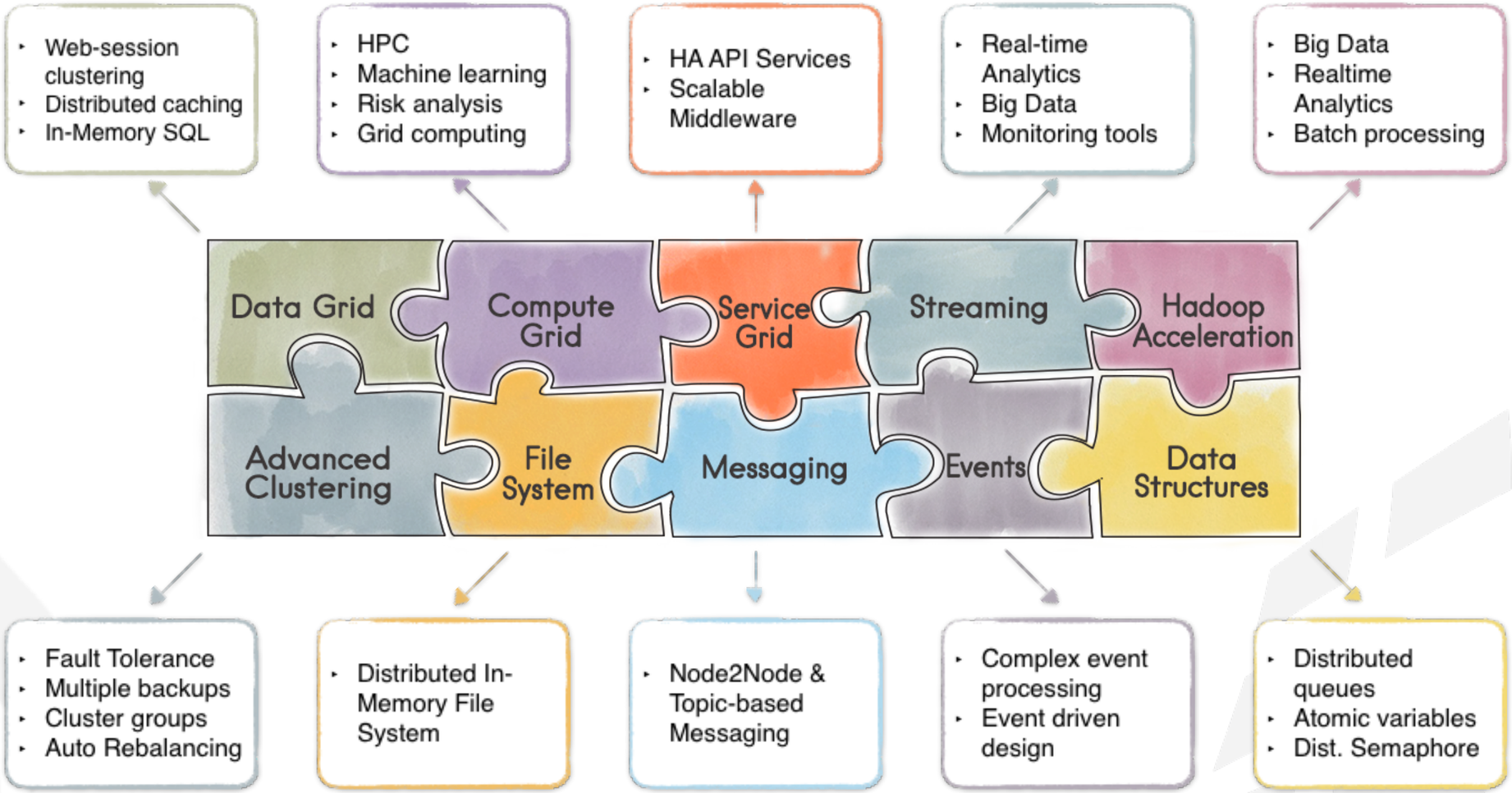
Back-end users, third-party clients and devices

What is an IMDF?

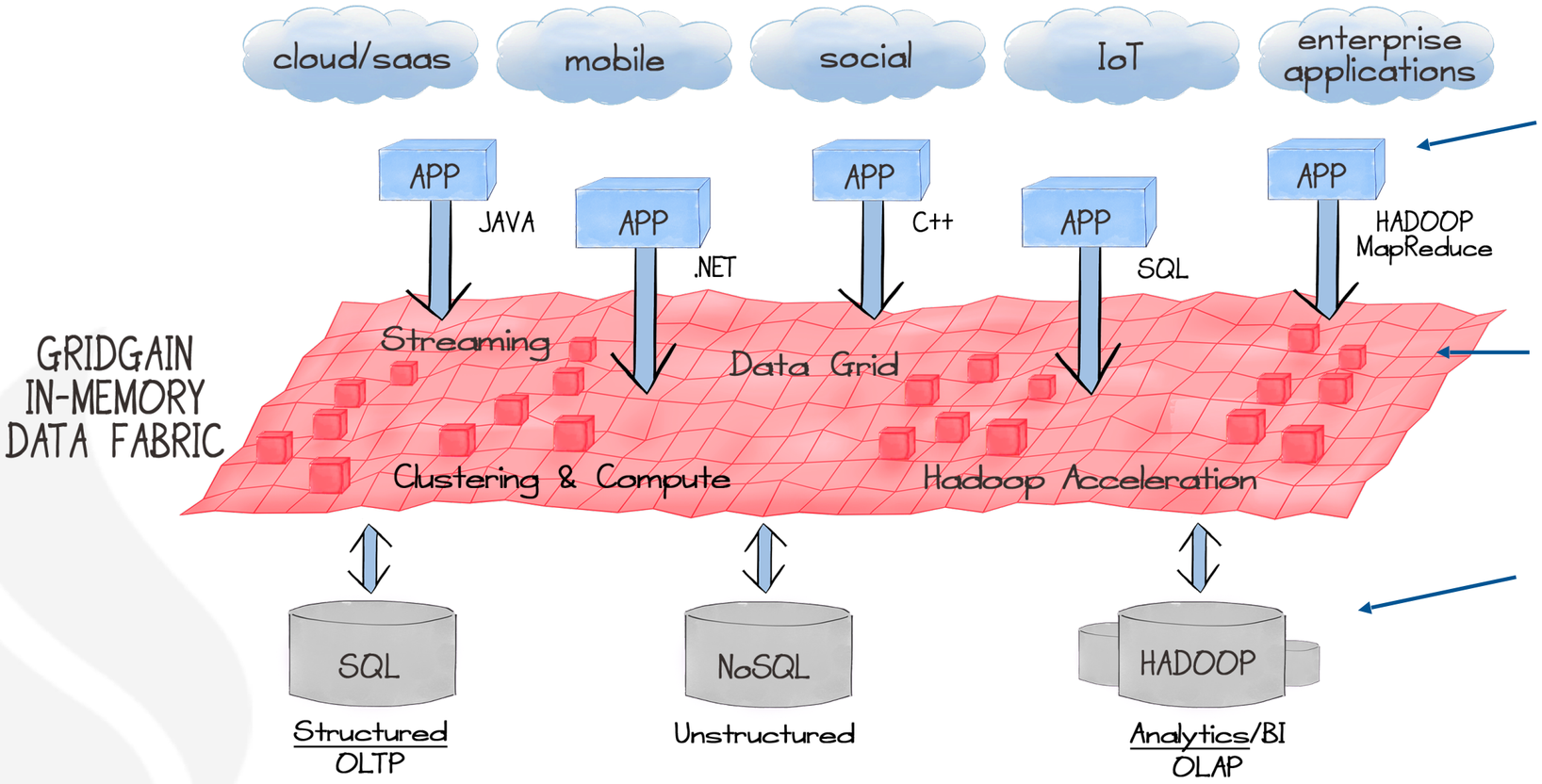


High-performance distributed in-memory platform for computing and transacting on large-scale data sets in near real-time.

IMDF Use Cases



GridGain In-Memory Data Fabric



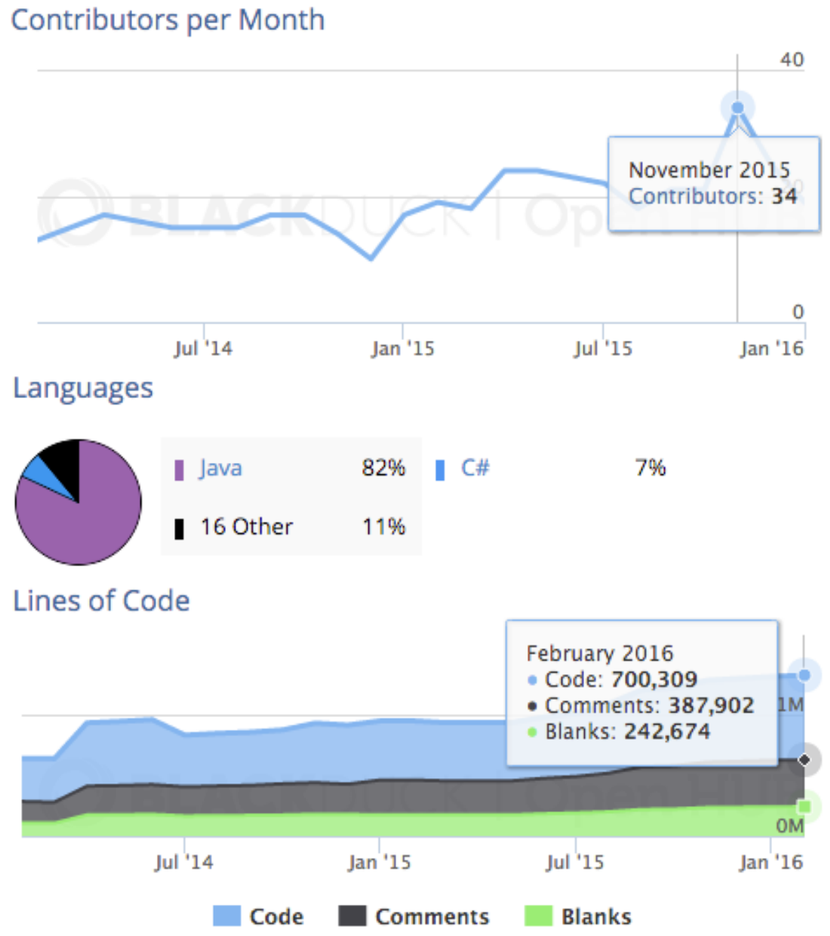
- Supports Applications of various types and languages

- Open Source – Apache 2.0
- Simple Java APIs
- 1 JAR Dependency
- High Performance & Scale
- Automatic Fault Tolerance
- Management/Monitoring
- Runs on Commodity Hardware

- Supports existing & new data sources
- No need to rip & replace

Apache Ignite & GridGain

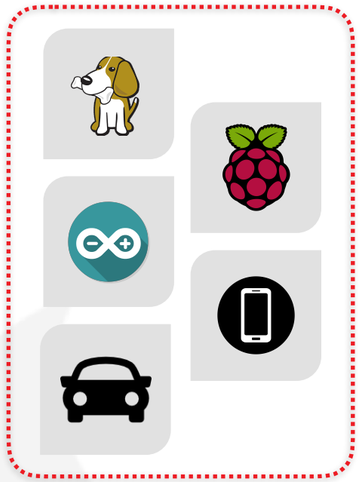
- **2007:** First version of GridGain
- **Oct. 2014:** GridGain contributes Ignite to ASF
- **Aug. 2015:** Ignite is the second fastest project to graduate after Spark
- Today:
  - 60+ contributors and rapidly growing
  - Huge development momentum - Estimated 192 years of effort since the first commit in February, 2014 [Openhub]
  - Mature codebase: 700k+ SLOC & more than 16k commits



GridGain solves the IoT Challenges

- Scale
- Speed
- Distribution
- Security
- Omni-channel
- Variable Workloads

High-Speed Data & Compute Layer



Batch Data

Streaming Data

Message bus

Compute Grid

Data Grid

External Persistency

Transactional & Analytical Workloads

External APIs

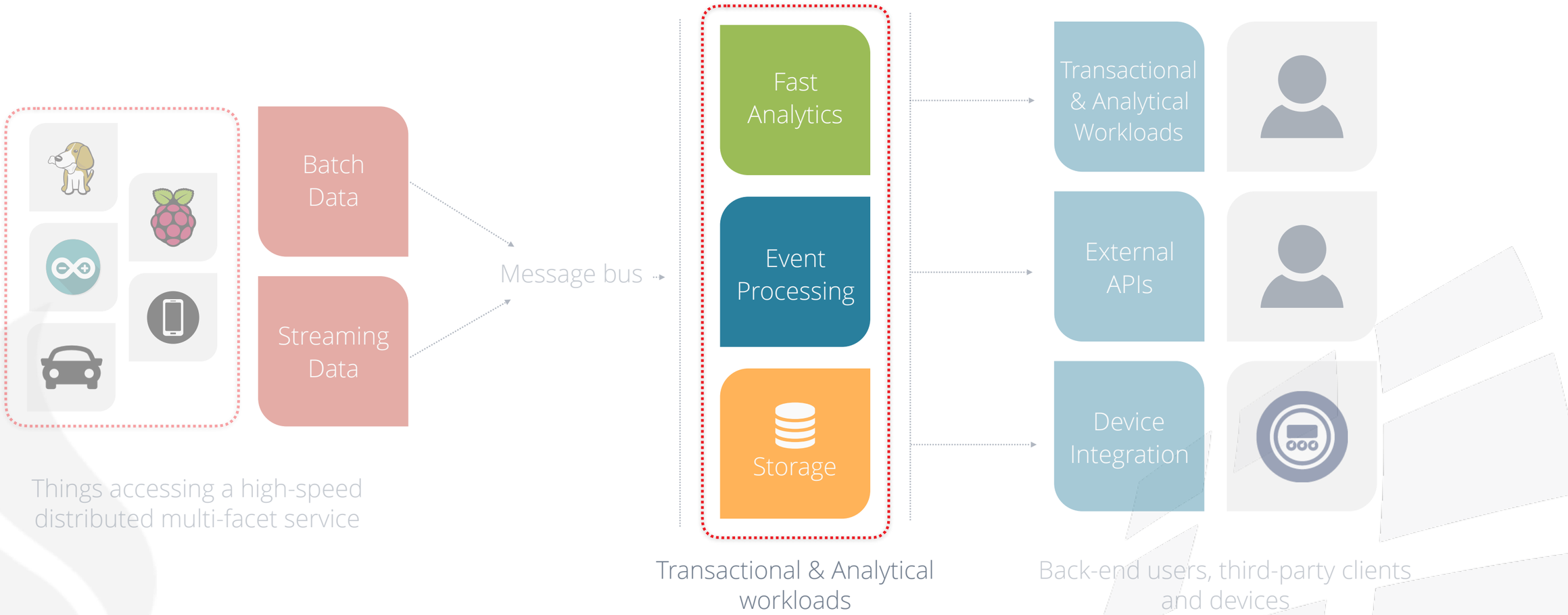
Device Integration

Things accessing a high-speed distributed multi-facet service

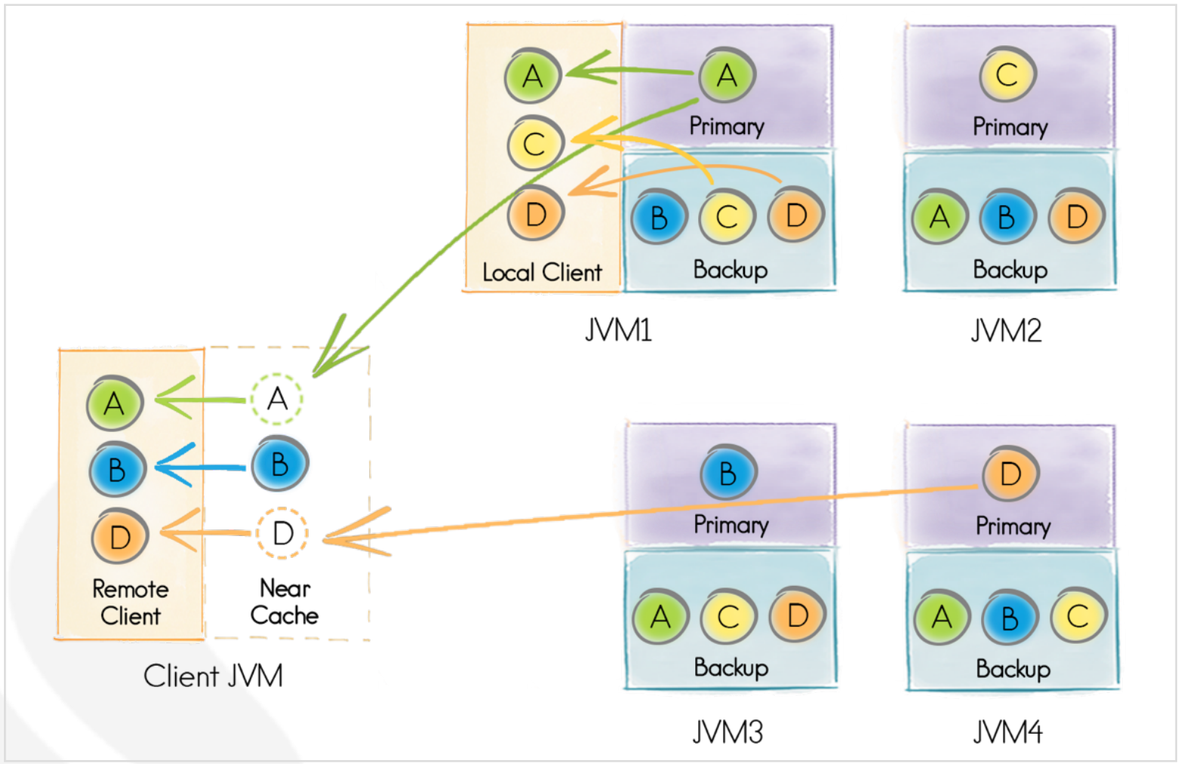
Transactional & Analytical workloads

Back-end users, third-party clients and devices

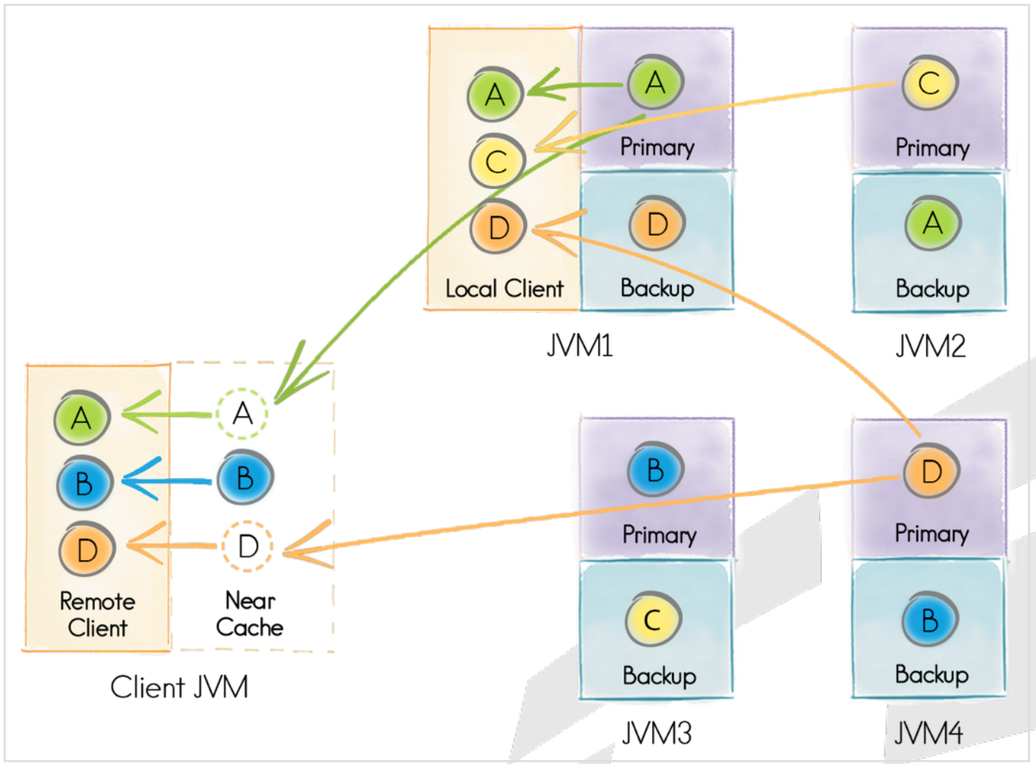
Scalability & Resilience with GridGain



Data Grid: Partitioning, Scalability & Resilience



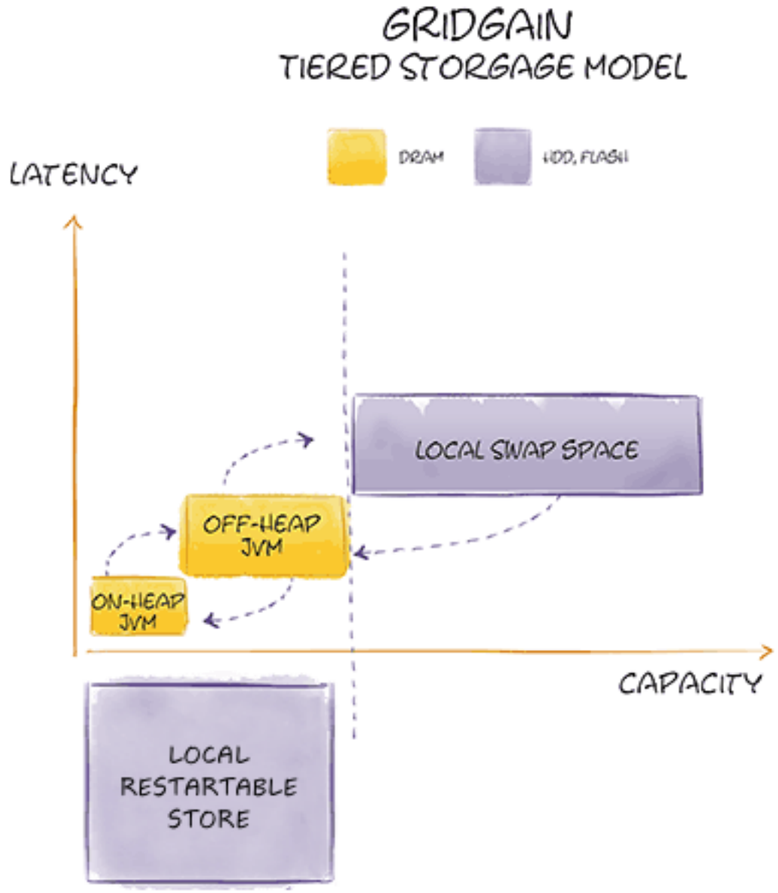
Replicated Cache



Partitioned Cache

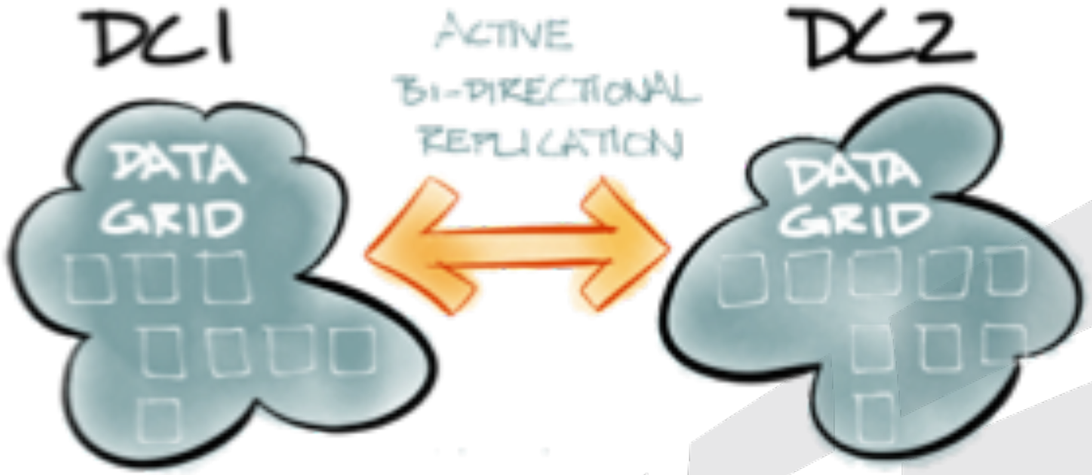
Data Grid: Tiered Memory & Local Store

- Tiered Memory
  - On-Heap -> Off-Heap -> Disk
- Persistent On-Disk Store
- Fast Recovery
- Local Data Reload
  - Eliminate Network and Db impacts when reloading in-memory store

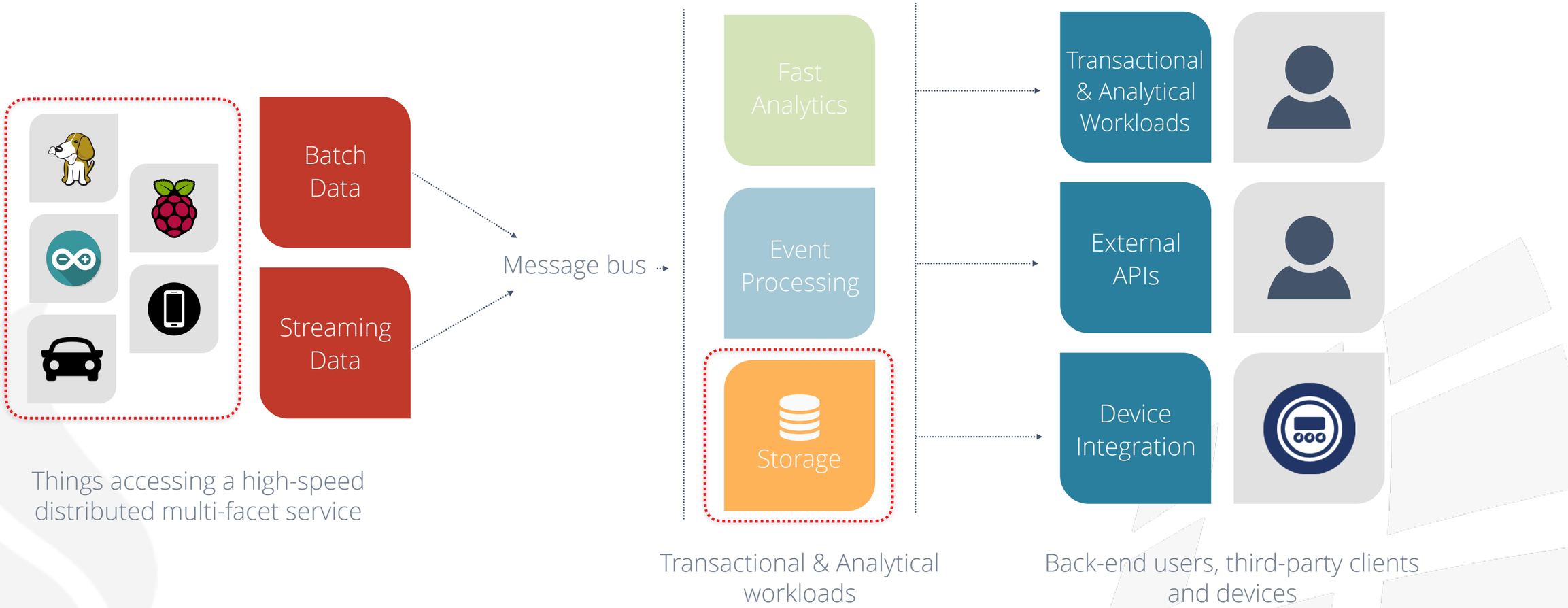


Data Grid: Data Center Replication

- Multiple (up to 32) Data Centers
- Complex Replication Technologies
- Active-Active & Active-Passive
- Smart Conflict Resolution
- Durable Persistent Queues
- Automatic Throttling

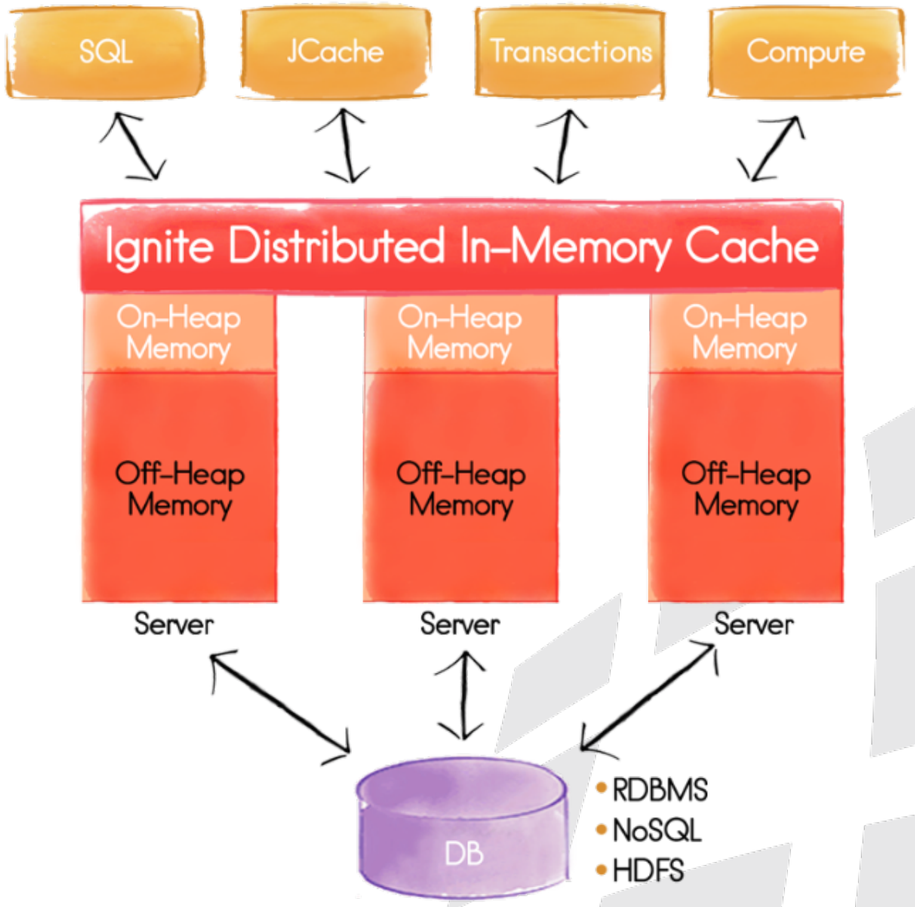


Storage and Caching using GridGain



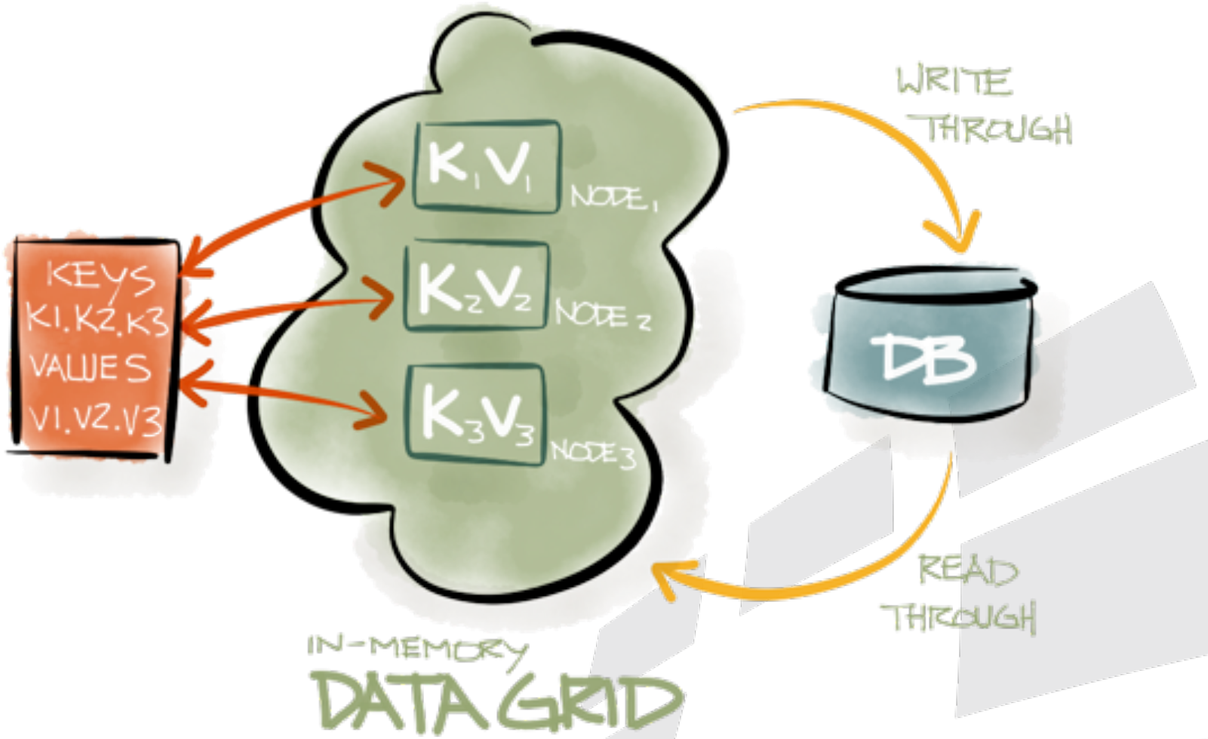
In-Memory Data Grid

- JCache (JSR 107)
  - Basic Cache Operations
  - Concurrent Map APIs
  - Collocated Processing (EntryProcessor)
  - Events and Metrics
  - Pluggable Persistence
- Ignite Data Grid
  - Distributed Key-Value Store
  - Fault Tolerance and Scalability
  - ACID Transactions
  - SQL Queries (ANSI 99)
  - In-Memory Indexes
  - Automatic RDBMS Integration



Data Grid: External Data-store Integration

- RDBMS & NoSQL support
- Read-through & Write-through
- Support for Write-behind
- Configurable eviction policies
- DB schema mapping wizard:
  - Generates all the XML configuration and Java POJOs



Data Grid: Cache APIs & Queries

- Predicate-based Scan Queries
- Text Queries based on Lucene indexing
- Query configuration using annotations, Spring XML or simple Java code
- SQL Queries
- Memcached (PHP, Java, Python, Ruby)
- HTTP REST API
- JDBC

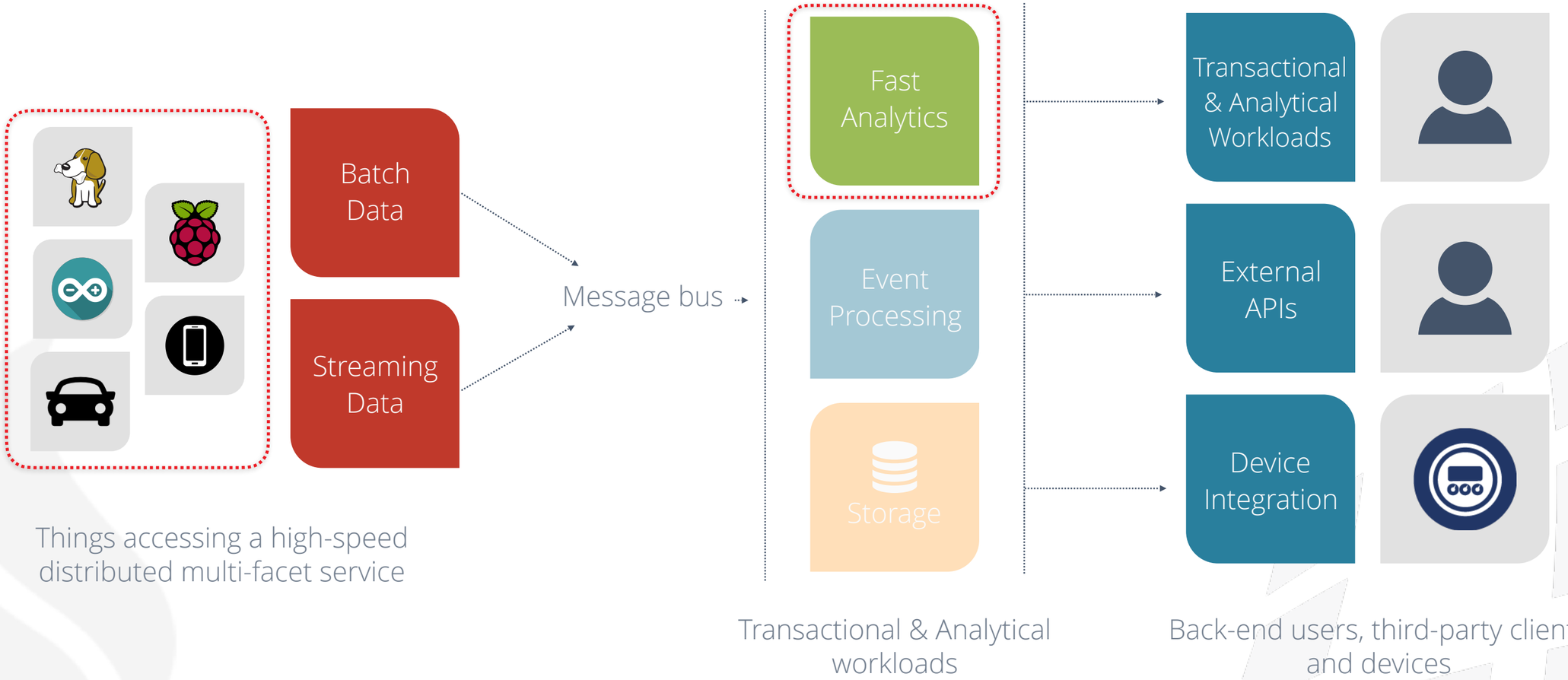
```
IgniteCache<Long, Person> cache = ignite.cache("mycache");

// Find only persons earning more than 1,000.
try (QueryCursor cursor = cache.query(new ScanQuery((k, p) -> p.getSalary() > 1000)) {
  for (Person p : cursor)
    System.out.println(p.toString());
}
```

```
// Query for all people with "Master Degree" in their resumes.
TextQuery txt = new TextQuery(Person.class, "Master Degree");

try (QueryCursor<Entry<Long, Person>> masters = cache.query(txt)) {
  for (Entry<Long, Person> e : cursor)
    System.out.println(e.getValue().toString());
}
```

```
/** Person ID (indexed). */
@QuerySqlField(index = true)
private long id;
```

```
// Listing indexes.
Collection<QueryIndex> indexes = new ArrayList<>(3);

indexes.add(new QueryIndex("id"));
indexes.add(new QueryIndex("orgId"));
indexes.add(new QueryIndex("salary"));

queryEntity.setIndexes(indexes);
```

Fast Analytics & Mixed Workloads using GridGain



Data Grid: Ad-Hoc SQL (ANSI 99)

- ANSI-99 SQL
- Always Consistent
- Fault Tolerant
- In-Memory Indexes (On-Heap & Off-Heap)
- Automatic Group By, Aggregations, Sorting
- Cross-Cache Joins, Unions, etc.
- Ad-Hoc SQL Support

```
IgniteCache<Long, Person> cache = ignite.cache("mycache");

// SQL join on Person and Organization.
SqlQuery sql = new SqlQuery(Person.class,
  "from Person, Organization "
  + "where Person.orgId = Organization.id "
  + "and lower(Organization.name) = lower(?)");

// Find all persons working for Ignite organization.
try (QueryCursor<Entry<Long, Person>> cursor = cache.query
(sql.setArgs("Ignite"))) {
  for (Entry<Long, Person> e : cursor)
    System.out.println(e.getValue().toString());
}
```

Data Grid: SQL Cross-Cache GROUP BY Example

```
IgniteCache<AffinityKey<UUID>, Person> cache = ignite.cache("persons");

// Query to get salaries grouped by organization.
SqlFieldsQuery qry = new SqlFieldsQuery(
    "select org.name, avg(salary), max(salary), min(salary) " +
        "from Person, \"Organizations\".Organization as org " +
        "where Person.orgId = org.id " +
        "group by org.name " +
        "order by org.name");

QueryCursor<List<?>> cursor = cache.query(qry);

List<List<?>> res = cursor.getAll();
```

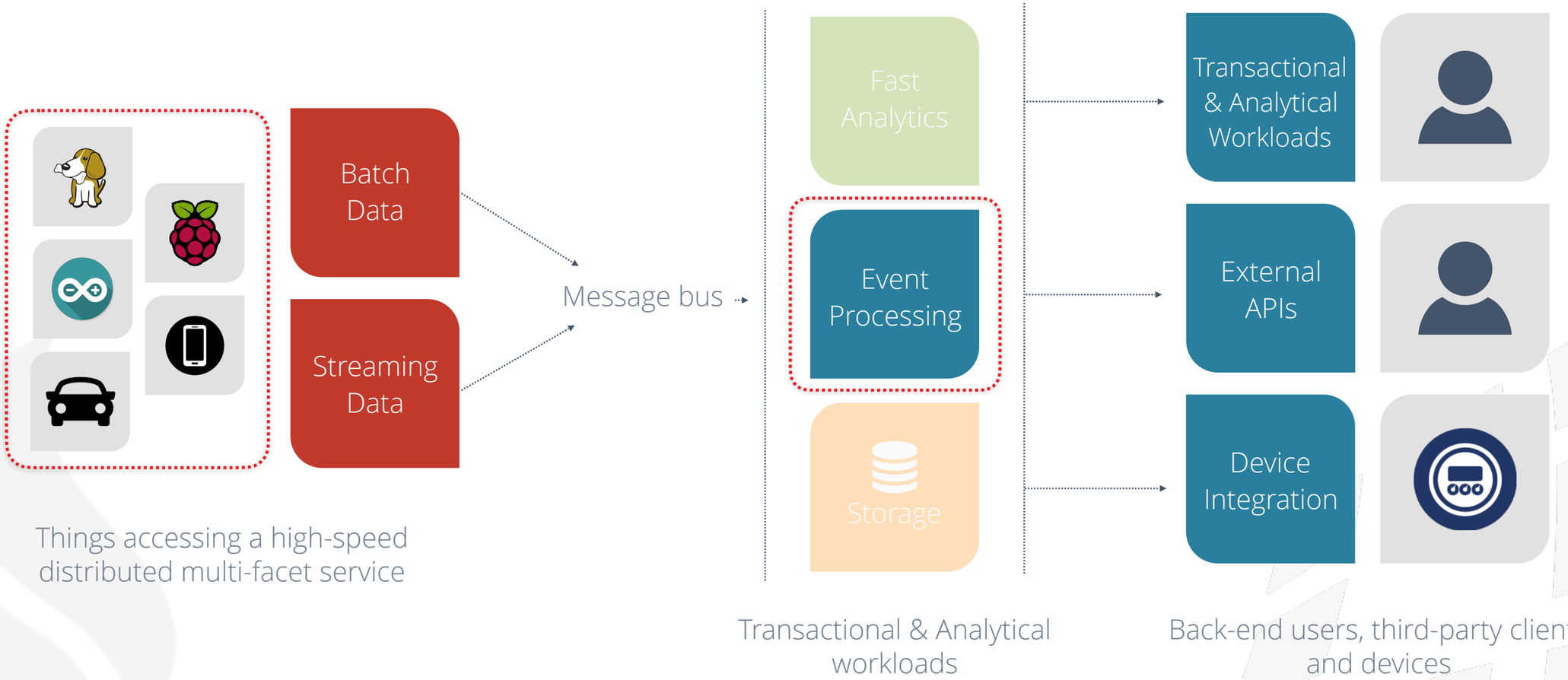
Data Grid: Transactions

- Fully ACID
- Support for Transactional & Atomic
- Cross-cache transactions
- Optimistic and Pessimistic concurrency modes with multiple isolation levels
- Deadlock protection
- JTA Integration

```
try (Transaction tx = transactions.txStart()) {
    Integer hello = cache.get("Hello");

    if (hello == 1)
        cache.put("Hello", 11);

    cache.put("World", 22);

    tx.commit();
}
```

```
IgniteTransactions txs = ignite.transactions();

// Start transaction in optimistic mode with repeatable read is
olation level.
Transaction tx = txs.txStart(TransactionConcurrency.OPTIMISTIC,
TransactionIsolation.REPEATABLE_READ);
```

Event Processing using GridGain

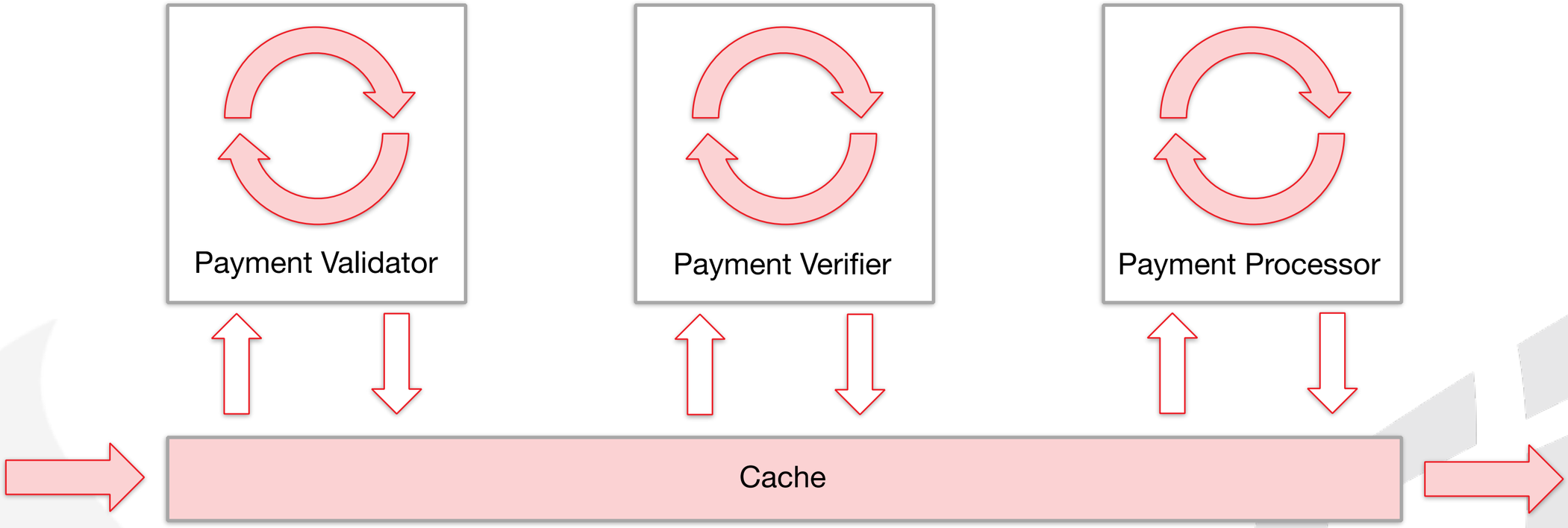


Data Grid: Continuous Queries

- Execute a query and get notified on data changes captured in the filter
- Remote filter to evaluate event and local listener to receive notification
- Guarantees exactly once delivery of an event

```
IgniteCache<Integer, String> cache = ignite.cache("mycache");

// Create new continuous query.
ContinuousQuery<Integer, String> qry = new ContinuousQuery<>();

// Optional initial query to select all keys greater than 10.
qry.setInitialQuery(new ScanQuery<Integer, String>((k, v) -> k > 10));

// Callback that is called locally when update notifications are received.
qry.setLocalListener((evts) ->
  evts.stream().forEach(e -> System.out.println("key=" + e.getKey() + ", val="
+ e.getValue())));

// This filter will be evaluated remotely on all nodes.
// Entry that pass this filter will be sent to the caller.
qry.setRemoteFilter(e -> e.getKey() > 10);

// Execute query.
try (QueryCursor<Cache.Entry<Integer, String>> cur = cache.query(qry)) {
  // Iterate through existing data stored in cache.
  for (Cache.Entry<Integer, String> e : cur)
    System.out.println("key=" + e.getKey() + ", val=" + e.getValue());

  // Add a few more keys and watch a few more query notifications.
  for (int i = 5; i < 15; i++)
    cache.put(i, Integer.toString(i));
}
```

Event Processing using GridGain



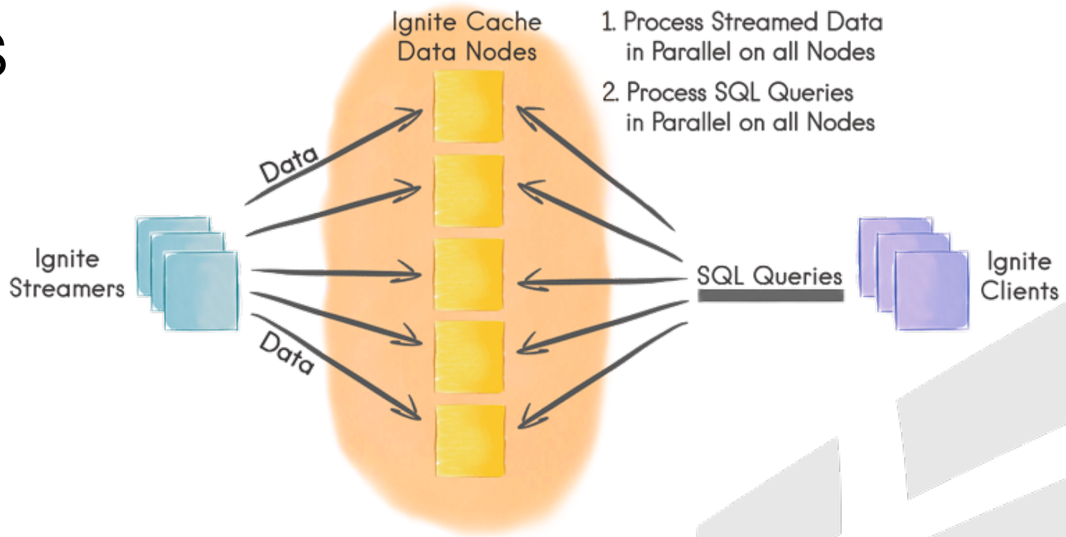
- Create chains of event processors & transform an object through various states
- Synchronous or asynchronous execution of remote filters & listeners with thread control

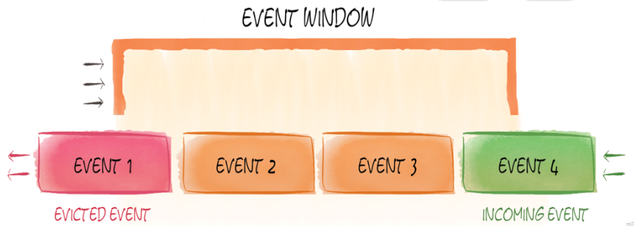
Event Processing using GridGain

```
@IgniteAsyncCallback
private static class NewPaymentsLocalListener implements
    CacheEntryUpdatedListener<int, BinaryObject> {
    @Override public void onUpdated(
        Iterable<CacheEntryEvent<? extends int, ? extends BinaryObject>> evts)
        throws CacheEntryListenerException {
        // Reference to cache.
        IgniteCache<int, Payment> cache = Ignition.ignite().cache("myCache");
        for (CacheEntryEvent<? extends int, ? extends BinaryObject> evt : evts) {
            Payment payment = evt.getValue().deserialize();
            System.out.println("New payment is processed [paymentId=" +
                evt.getKey() + ", payment=" + payment + ']');
            Payment verifiedPayment = new Payment(
                payment.getAmount(),
                PaymentStatus.VERIFIED);
            cache.put(evt.getKey(), verifiedPayment);
        }
    }
}
```

In-Memory Streaming and CEP

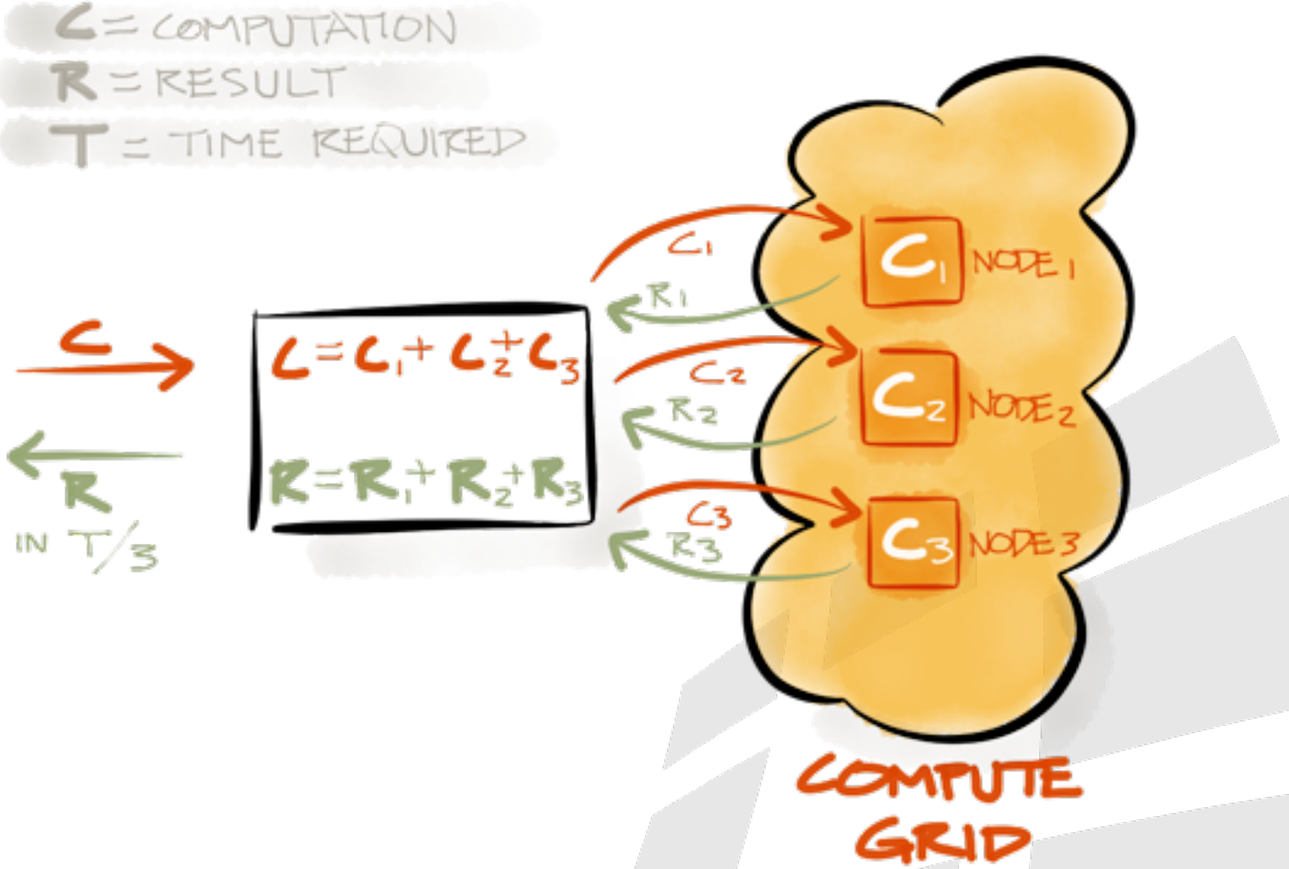
- Streaming Data Never Ends
- Branching Pipelines
- Pluggable Routing
- Sliding Windows for CEP/ Continuous Query
- Real Time Analysis





In-Memory Compute Grid

- Direct API for MapReduce
- Direct API for ForkJoin
- Zero Deployment
- Cron-like Task Scheduling
- State Checkpoints
- Load Balancing
- Automatic Failover
- Full Cluster Management
- Pluggable SPI Design



In-Memory Compute Grid

- Distributed Closures
  - Java lambda expressions (JSR 335)
- Distributed ExecutorService
- Sync or Async
- Task Deployment (GAR)

```
// Execute closure on all cluster nodes.
Collection<Integer> res = compute.apply(
    String::length,
    Arrays.asList("How many characters".split(" "))
);

// Add all the word lengths received from cluster nodes.
int total = res.stream().mapToInt(Integer::intValue).sum();
```

```
// Limit broadcast to remote nodes only.
IgniteCompute compute = ignite.compute(ignite.cluster
().forRemotes());

// Print out hello message on remote nodes in the clust
er group.
compute.broadcast(() -> System.out.println("Hello Node:
" + ignite.cluster().localNode().id()));
```

```
// Iterate through all words and print
// each word on a different cluster node.
for (String word : "Print words on different cluster no
des".split(" "))
    // Run on some cluster node.
    compute.run(() -> System.out.println(word));
```

Data Grid: Distributed Data Structures

- Distributed Map (cache)
- Distributed Set
- Distributed Queue
- CountDownLatch
- AtomicLong
- AtomicSequence
- AtomicReference
- Distributed ExecutorService

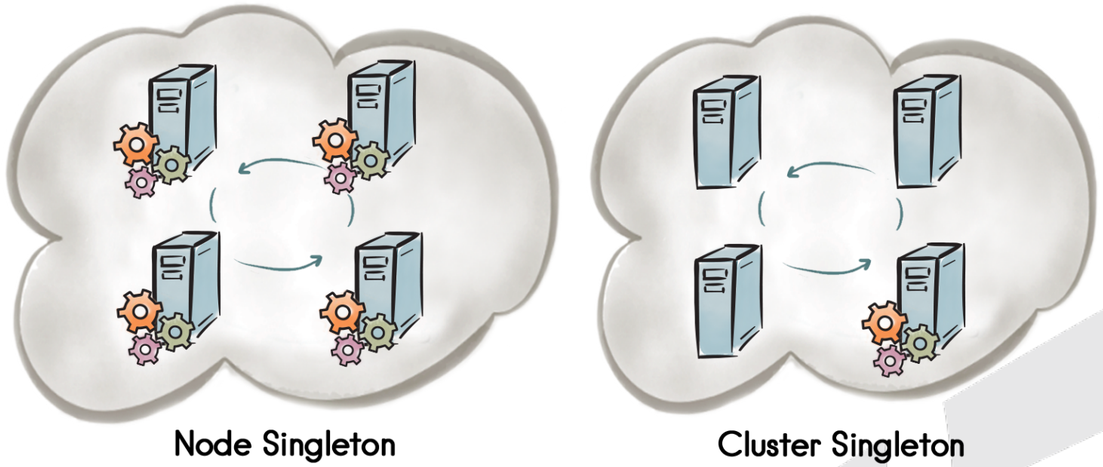
```
Ignite ignite = Ignition.ignite();

CollectionConfiguration colCfg = new Colle
ctionConfiguration();

colCfg.setCollocated(true);

// Create collocated queue.
IgniteQueue<String> queue = ignite.queue
("queueName", 0, colCfg);
```

```
// Initialize atomic long.
final IgniteAtomicLong atomicLong = ignite.atomic
Long("atomicName", 0, true);

// Increment atomic long on local node.
System.out.println("Incremented value: " + atomic
Long.incrementAndGet());
```

```
IgniteAtomicLong atomicLong = ignite.atomicLong(
    "atomicName", // Atomic long name.
    0,            // Initial value.
    false         // Create if it does not exist.
)
```

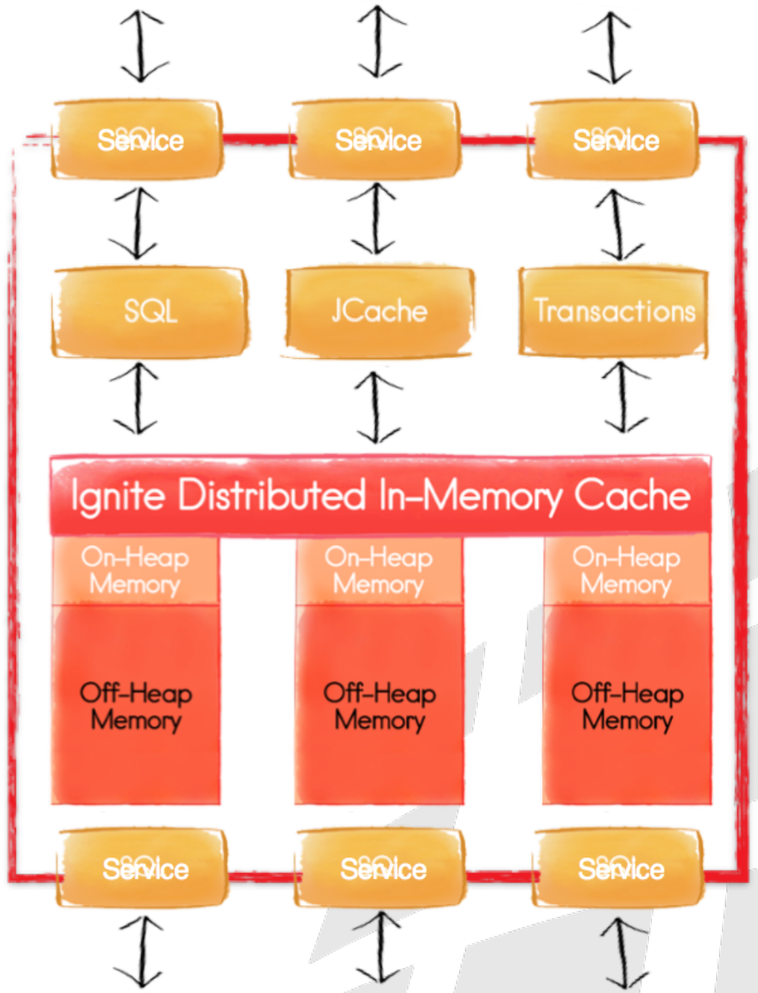
In-Memory Service Grid

- Singletons on the Cluster
  - Cluster Singleton
  - Node Singleton
  - Key Singleton
- Distribute any Data Structure
  - Available Anywhere on the Grid
  - Access Anywhere via Proxies
- Guaranteed Availability
  - Auto Redeployment in Case of Failures



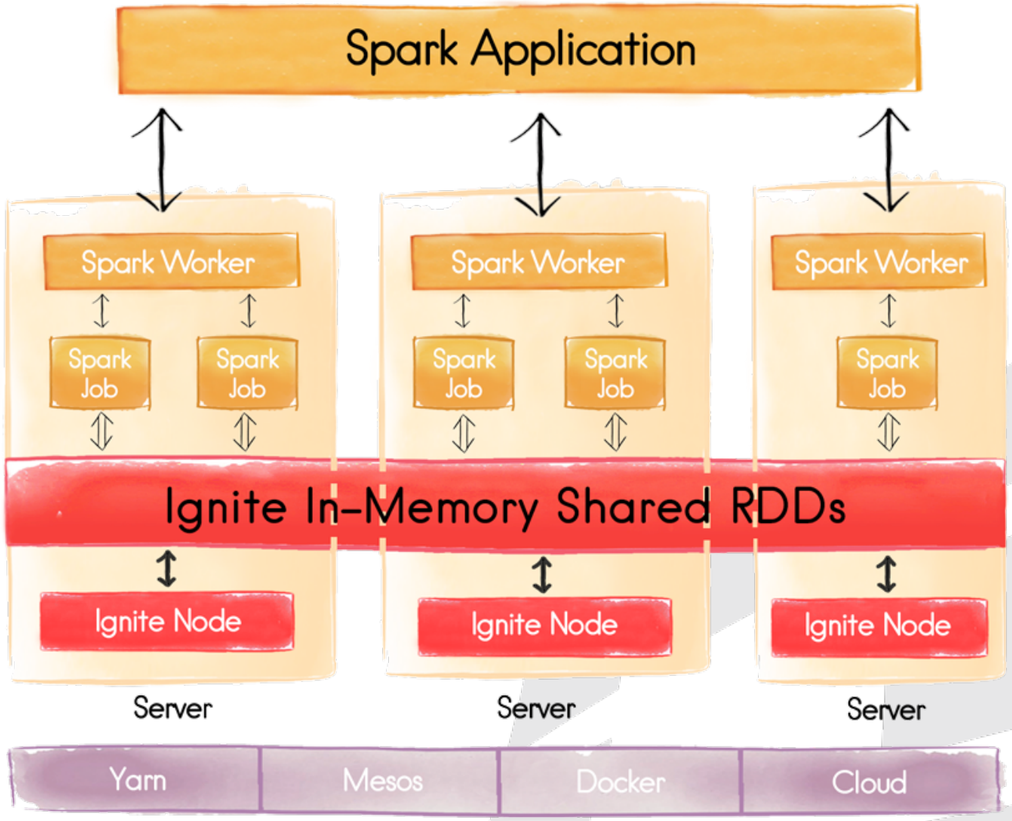
In-Memory Service Grid

- Resilience - Build an in-memory resilient service layer between your client application and the grid
- Encapsulation - Only expose application APIs and not direct grid APIs
- Service Chaining - Call services internally via compute tasks to create service chains



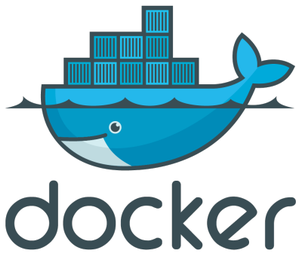
Spark Integration: Shared RDDs & Improved SQL

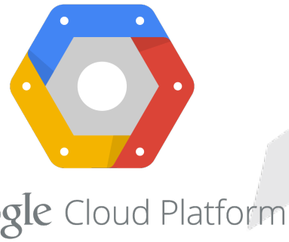
- IgniteRDD
  - Share RDD across jobs on the host
  - Share RDD across jobs in the application
  - Share RDD globally
- Faster SQL
  - In-Memory Indexes
  - SQL on top of Shared RDD



Deployment: Infrastructure Integration

- Docker
- Amazon AWS
- Google Cloud
- Apache JClouds
- Mesos
- YARN
- Apache Karafe (OSGi)





IoT Reference Architecture

GG Visor & WebConsole
Access Portal

GG WebConsole
Dashboards

GG Service Grid
External APIs

GridGain Compute & Data Grid
Event Processing & Analytics (HTAP)

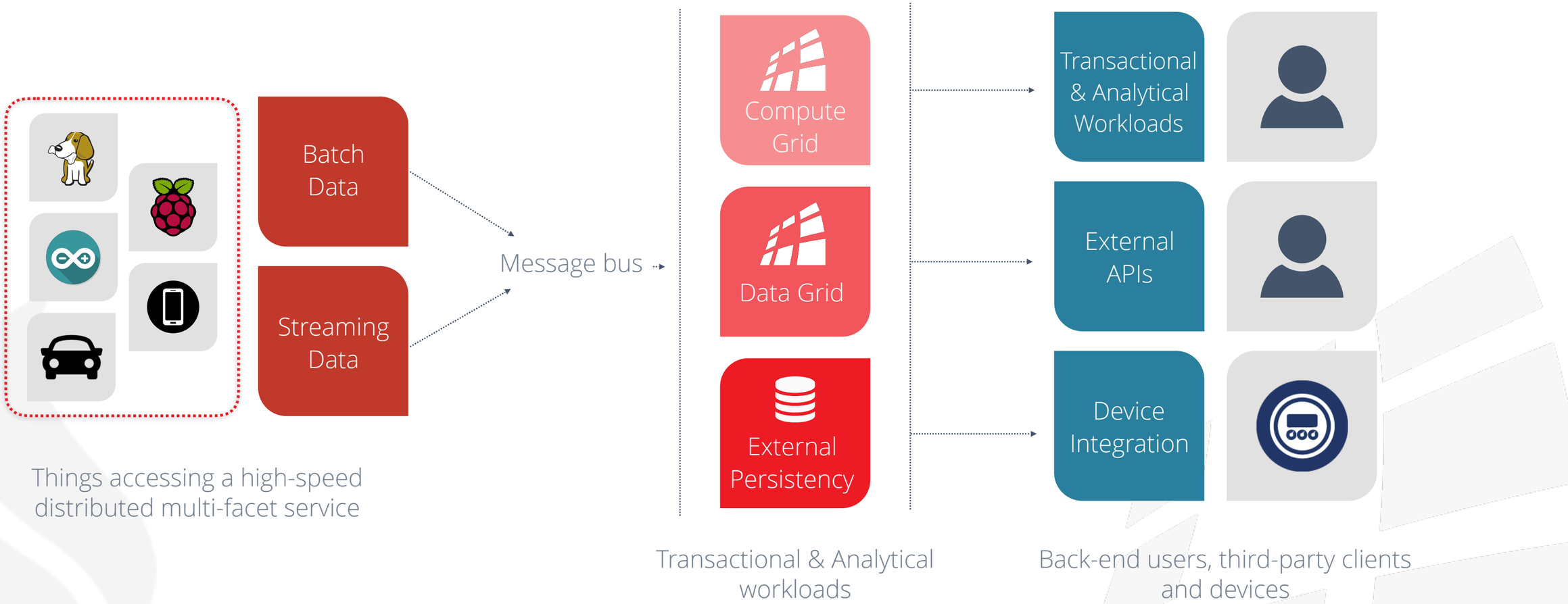
Aggregation Layer
(ESB / Message Broker / Gateway)

Communications (HTTP / MQTT / NFC / WiFi / Ethernet)

Device Manager

GG Security & Auditing
Identity & Access Management

GridGain delivers next generation IoT Platform



GridGain delivers next generation IoT Platform

- Scale
- Speed
- Distribution
- Security
- Omni-channel
- Variable Workloads

Thank You!

Author: Christos Erotocritou

Thank you for joining us. Follow the conversation.

www.gridgain.com

@gridgain
#gridgain