SPARK SUMMIT

Fast Data with Apache Ignite & Apache Spark

Christos Erotocritou, GridGain Systems

#EUstr10

…is a **distributed**, **memory-centric** data platform
with powerful & flexible **processing** APIs

Apache Ignite Memory-Centric Data Platform

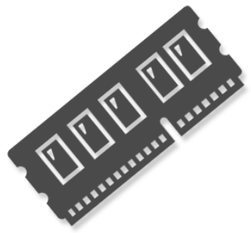
SQL | Key/Value | Transactions | Compute | IgniteRDD | Streaming | ML

Ignite Memory-Centric Storage

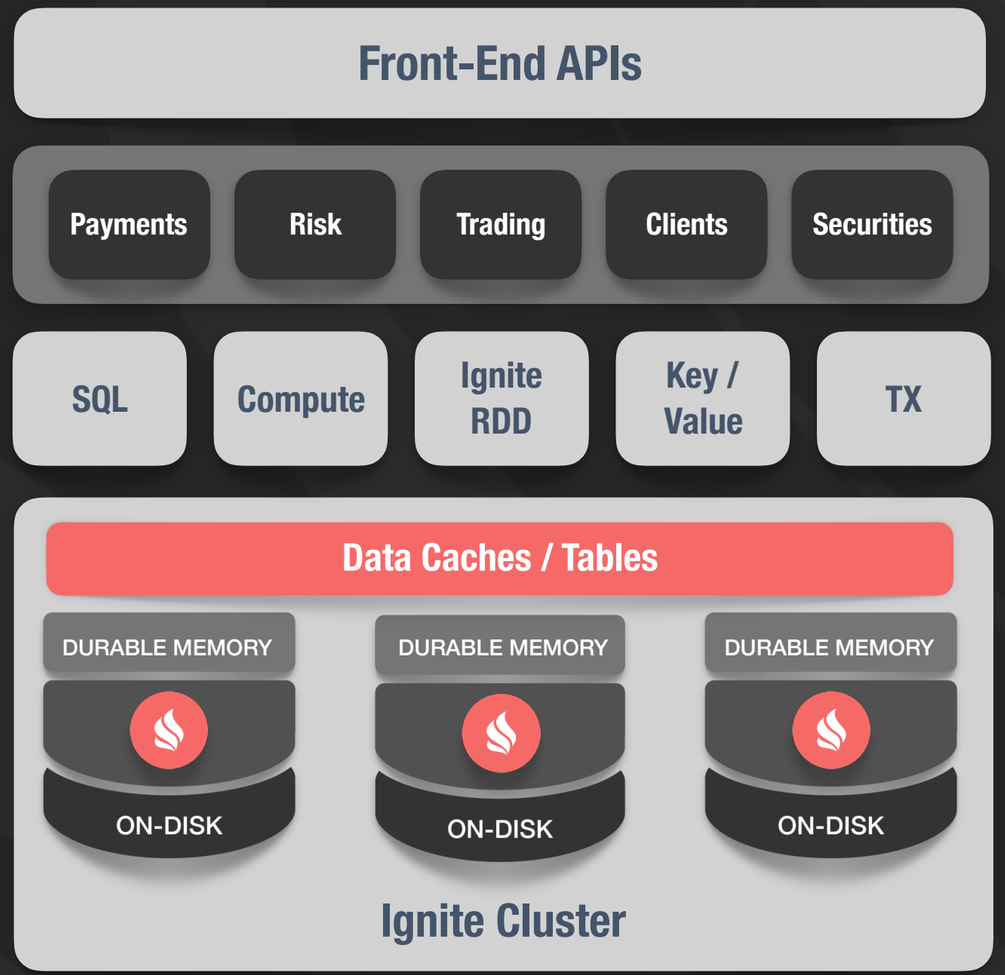
Ignite Native Persistence
(Flash, SSD, Intel 3D XPoint)

Third-Party Persistence
(RDBMS, HDFS, NoSQL)

Memory-Centric Storage

Memory Centric	VS	Disk Centric
		
Memory as a Main Storage		Memory as Caching Layer
Collocated and Client-Server Processing		Client-Server Processing

Pure Ignite Deployment



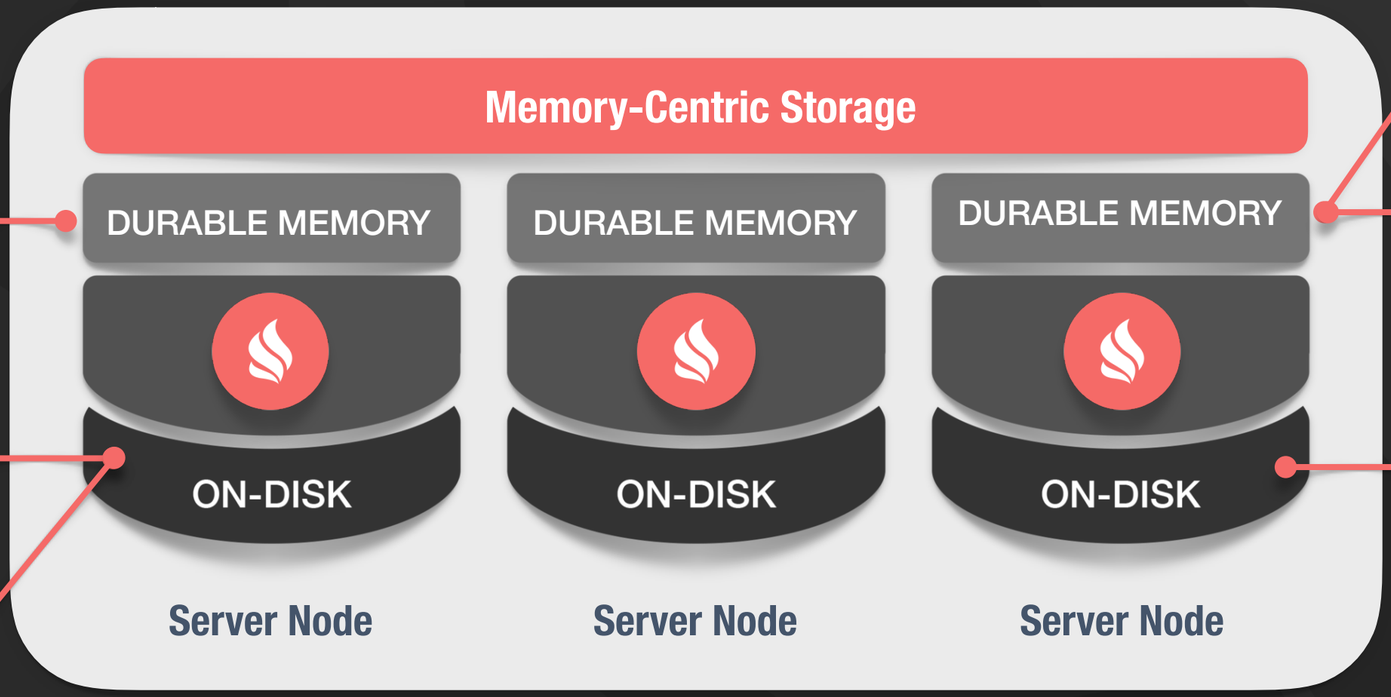
Applications in Java, .NET & C++

Wide Range of Data Access and Processing APIs

Shared Storage across Apps & Support for Multi-Tenancy

Disk & Memory Data Storage

Durable Memory



Predictable memory consumption

Fully Transactional (Write-Ahead Log)

Instantaneous Restarts

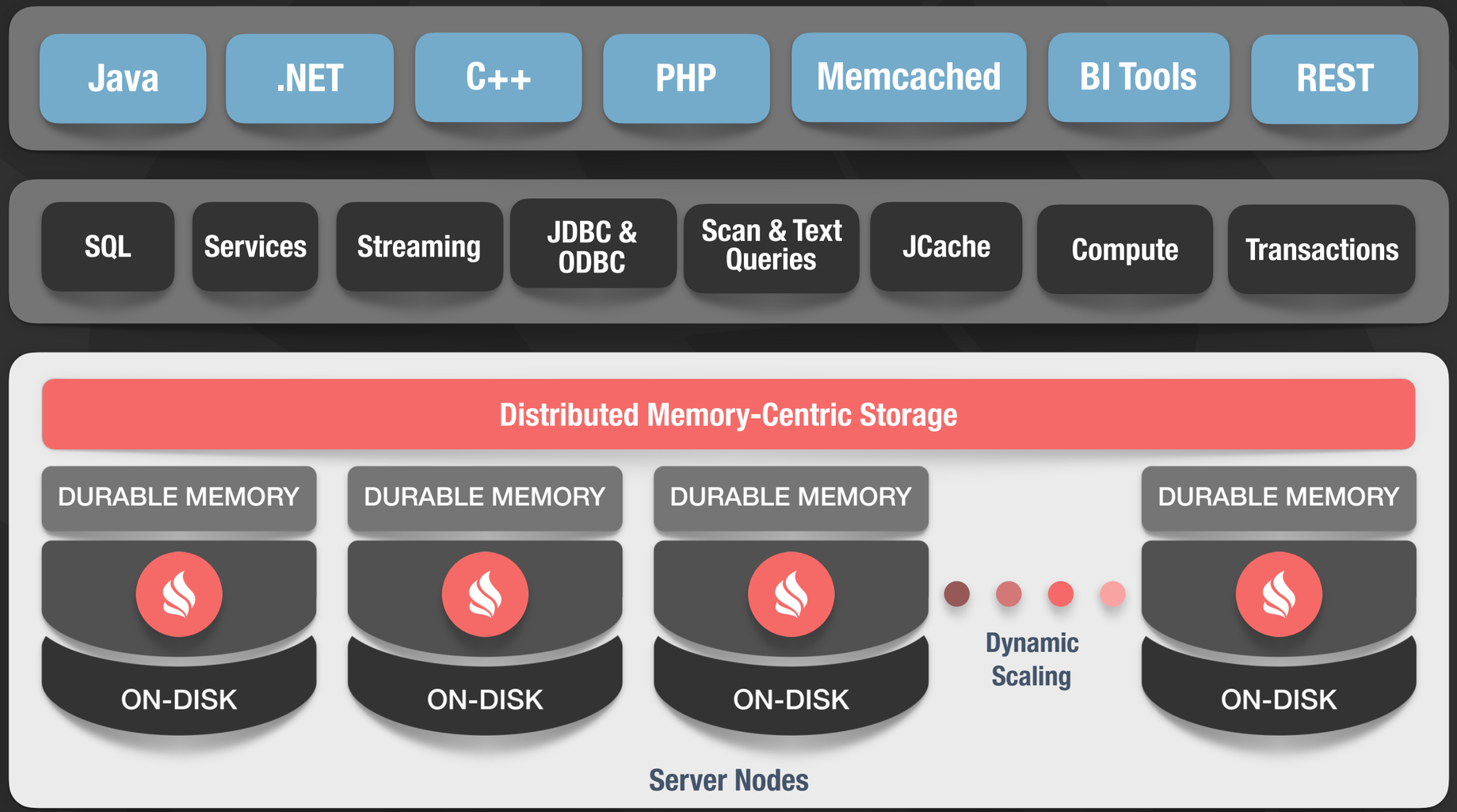
Automatic Defragmentation

Off-heap Removes noticeable GC pauses

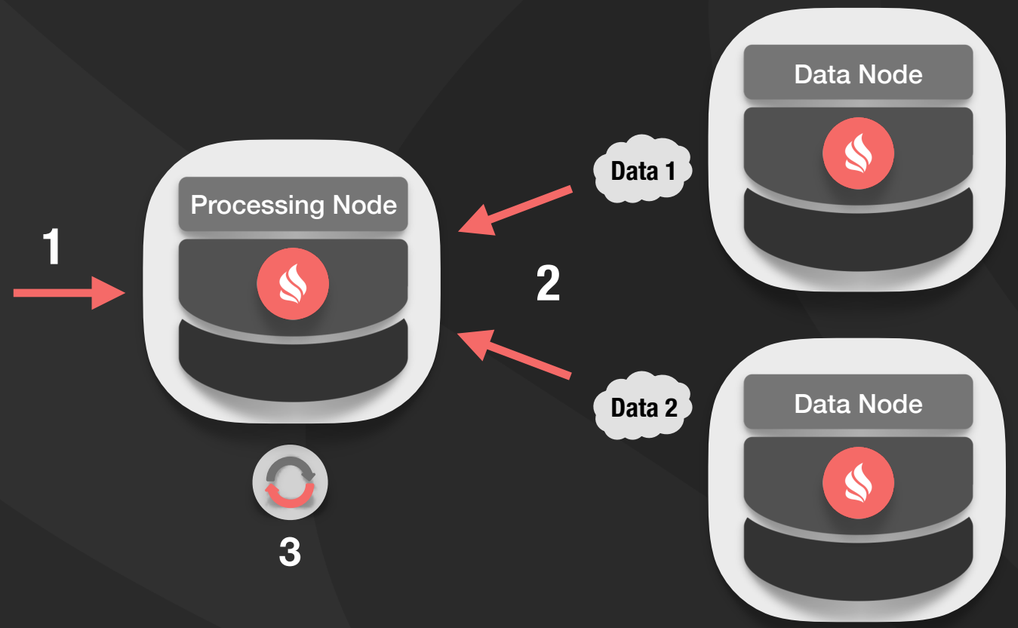
Stores Superset of Data

Ignite Server Cluster

Apache Ignite Features

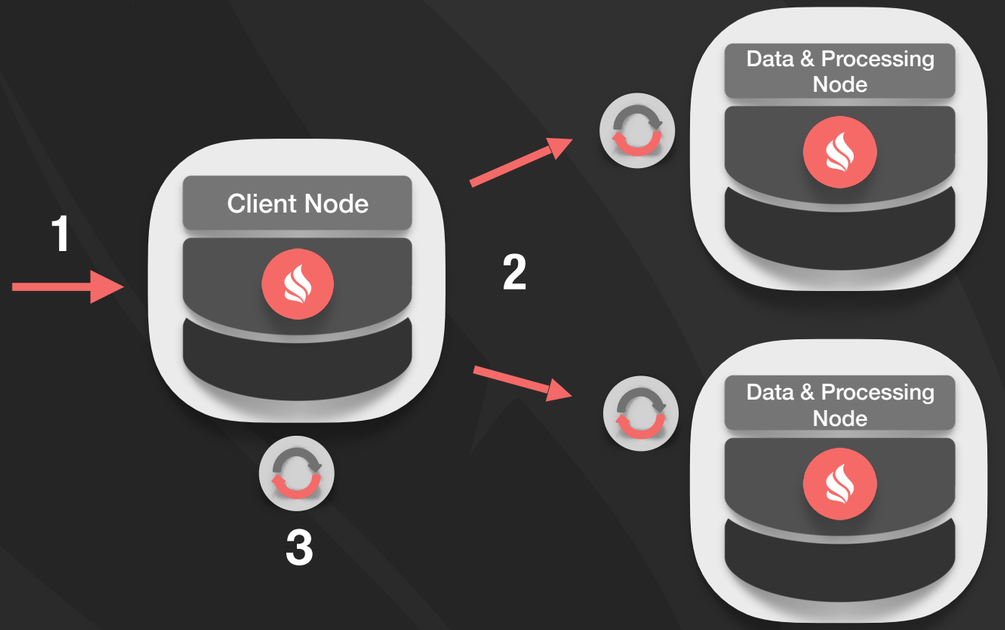


Client-Server Processing



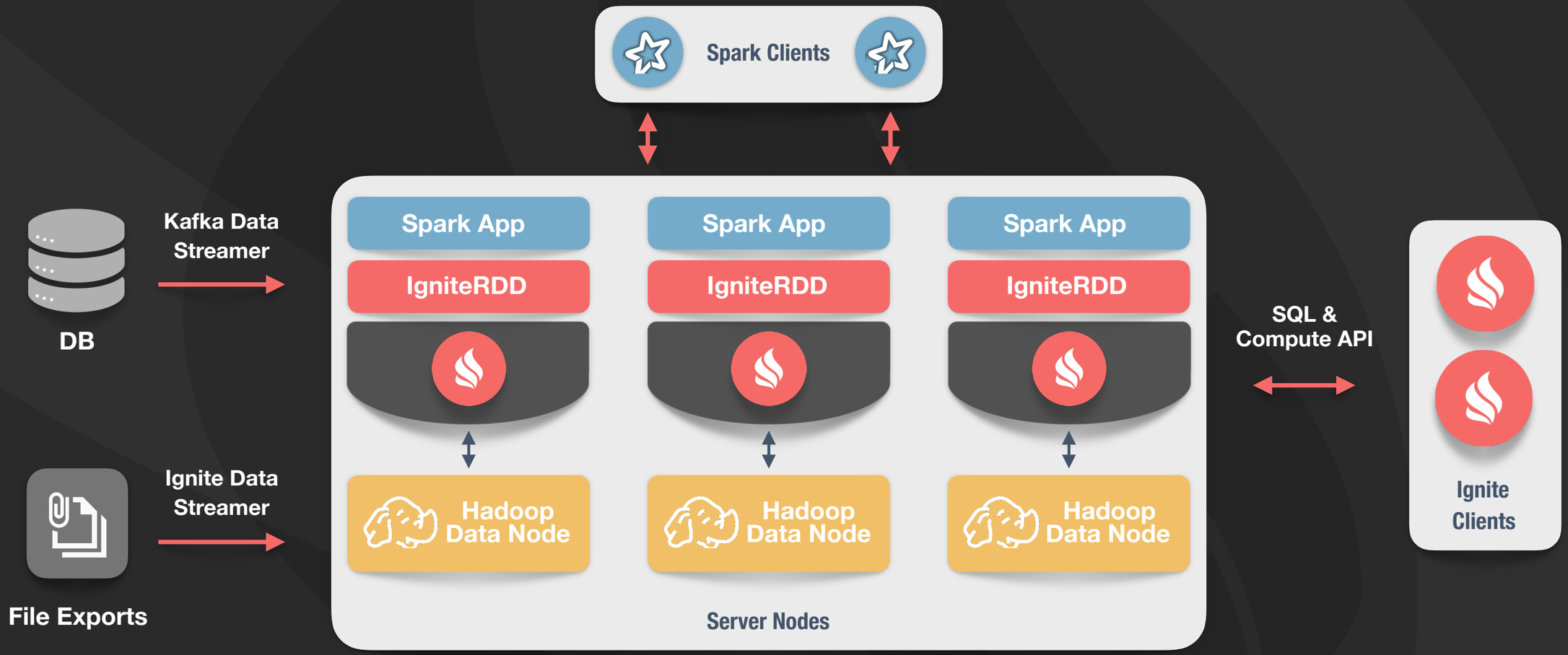
1.Initial Request
2.Fetch data from remote nodes
3.Process the entire data-set

Co-located Processing

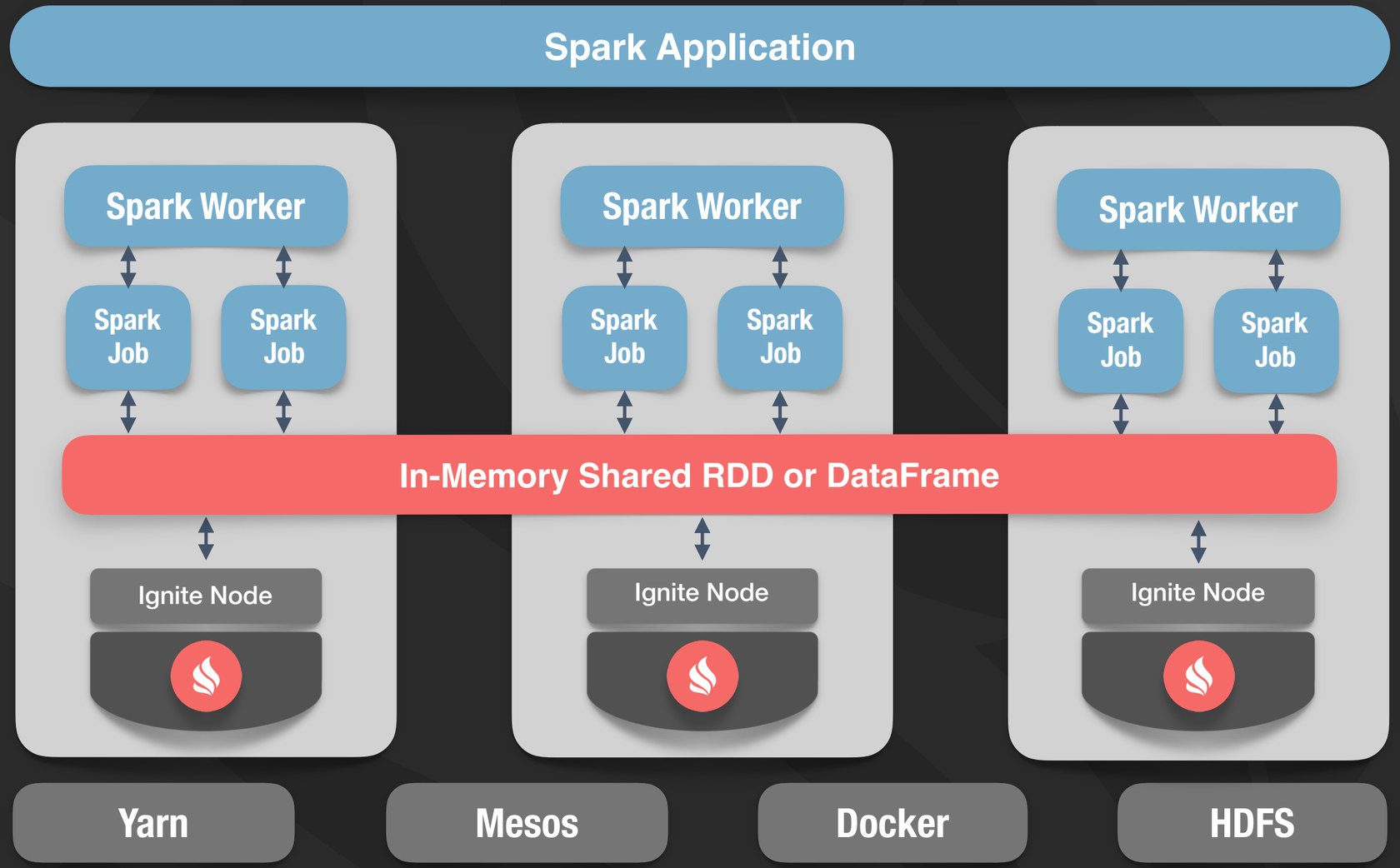


1.Initial request
2.Co-locate processing with data
3.Reduce multiple results into one

Hadoop, Spark & Ignite Deployment



Apache Ignite Spark Integration



Share RDD across jobs on the host

Share RDD Globally

In-Memory Indexes

SQL on top of RDDs

Working with IgniteRDD

- IgniteContext is the main entry point to Spark-Ignite integration:

```
val igniteContext = new IgniteContext[Integer, Integer]
   (sparkContext, () => new IgniteConfiguration())
```

- Reading values from Ignite:

```
val cache = igniteContext.fromCache("myRdd")
val result = cache.filter(_._2.contains("Ignite")).collect()
```

- Saving values to Ignite:

```
val cacheRdd = igniteContext.fromCache("myRdd")
   cacheRdd.savePairs(sparkContext.parallelize(1 to 10000, 10).map(i => (i, i)))
```

- Running SQL queries against Ignite Cache:

```
val cacheRdd = igniteContext.fromCache("myRdd")
val result = cacheRdd.sql
   ("select _val from Integer where val > ? and val < ?", 10, 100)
```

Working with DataFrame API

- Create an IgniteRDD

```
val companyCacheIgnite = new IgniteContext[Int, String](sc, () =>
new IgniteConfiguration()).fromCache("CompanyCache")
```

- Create a “Company” DataFrame

```
val dfCompany = sqlContext.createDataFrame(companyCacheIgnite.map(p=>
Company(p._1, p._2)))
```

- Register DataFrame as a table

```
dfCompany.registerTempTable("company")
```

- Data source agnostic
- Fully fledged compute engine and durable storage
- OLAP & OLTP
- Zero-deployment
- In-Memory SQL support
- Fully ACID transactions across memory and disk
- Less focused on Hadoop
- Early ML Support
- Growing Community

- Ingests data from HDFS or another distributed file system
- Inclined towards analytics (OLAP) and focused on MR-specific payloads
- Requires the creation of RDD and data and processing operations are governed by it
- Basic disk-based SQL support
- Strong ML libraries
- Big community

- **What is GridGain?**
  - Binary build of Apache Ignite™
  - Added enterprise features for enterprise deployments
  - Earlier features and bug fixes by a few weeks
  - Fully certified & tested releases

“We develop and support the worlds leading In-Memory Computing Platform”

Any Questions?

Thank you for joining us. Follow the conversation.
http://ignite.apache.org