GridGain

Cluster Memory Management Techniques

Preventing Out of Memory Incidents

Denis Magda

Apache Ignite PMC; Head of DevRel at GridGain

Agenda

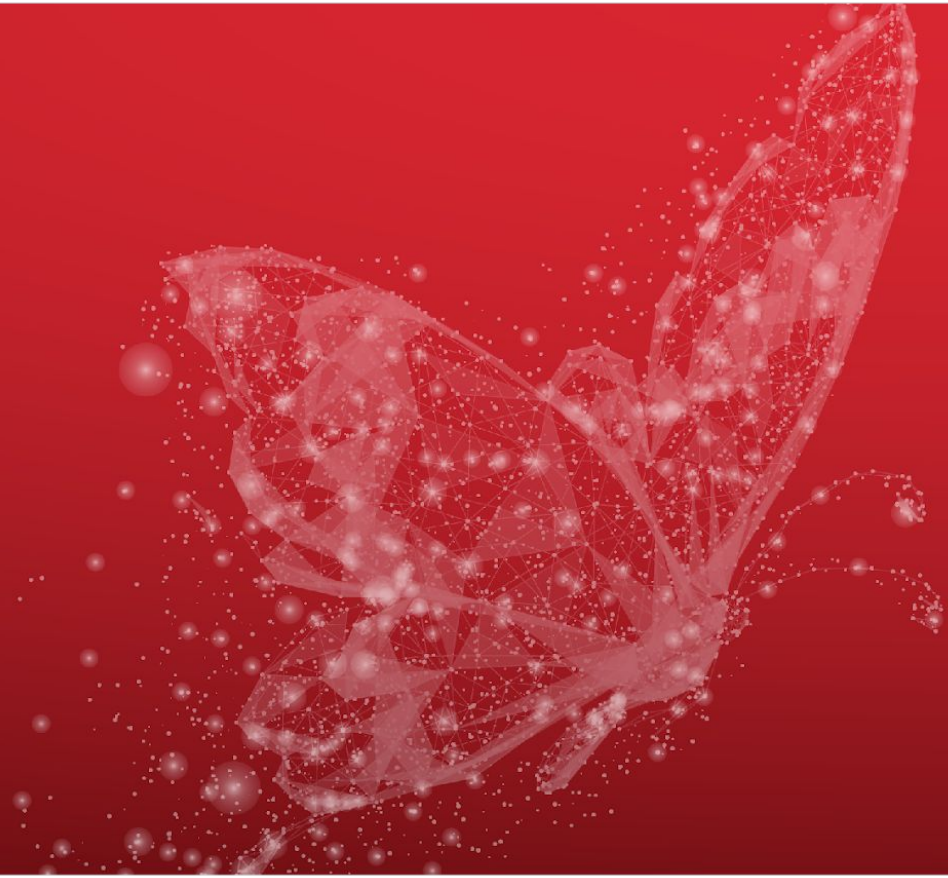
1. Ignite storage engine

2. Generic techniques:
   - Eviction and expiration policies

3. Off-heap specific techniques:
   - Swapping
   - Ignite Native Persistence

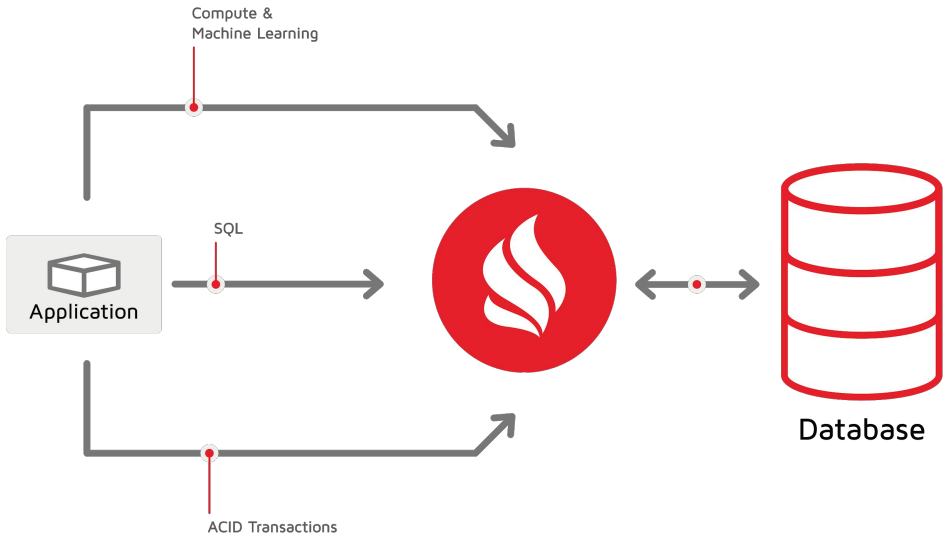
4. Java heap specific techniques:
   - SQL memory quotas

Ignite Storage Engine

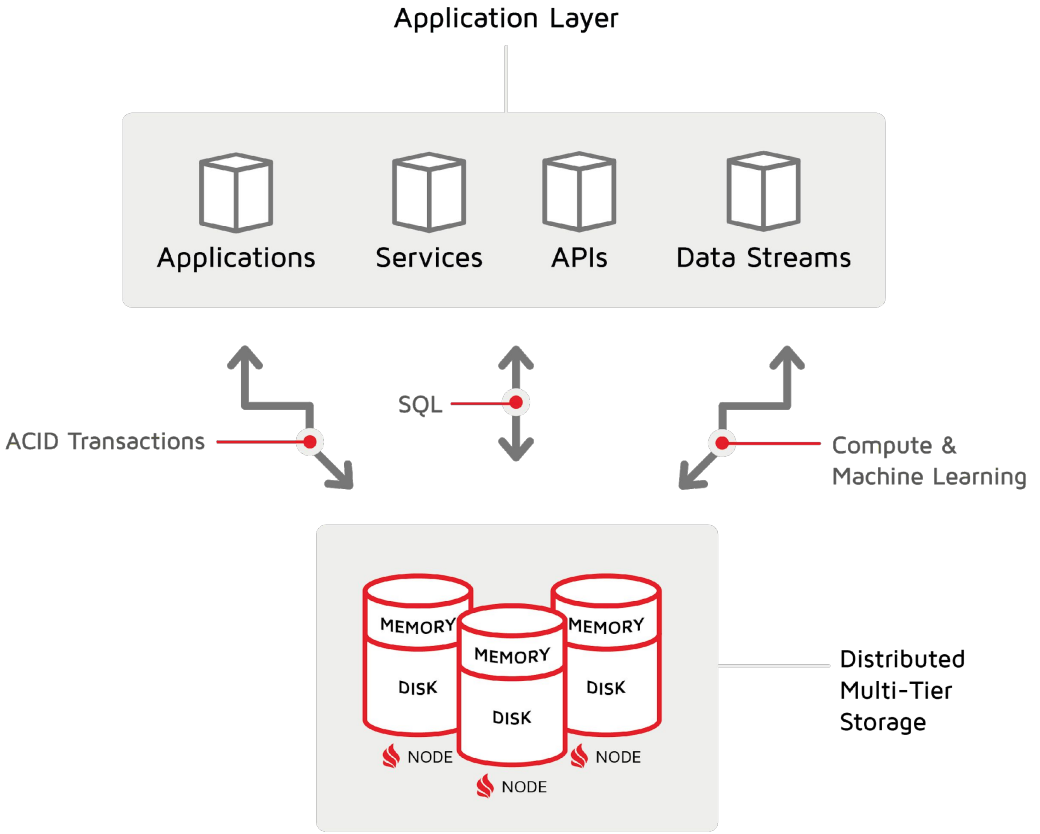


GridGain

Apache Ignite as a Cache or as a Database



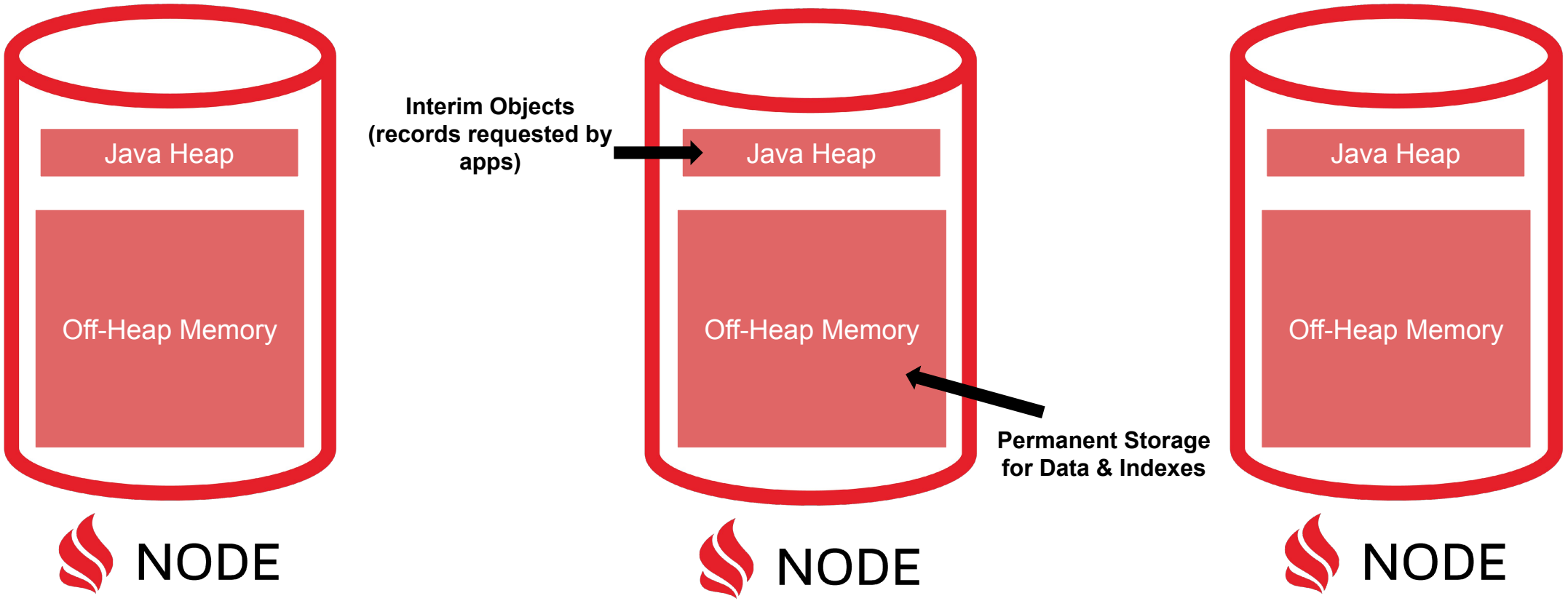
Ignite as a Cache and Data Grid



Ignite as a Database

GridGain

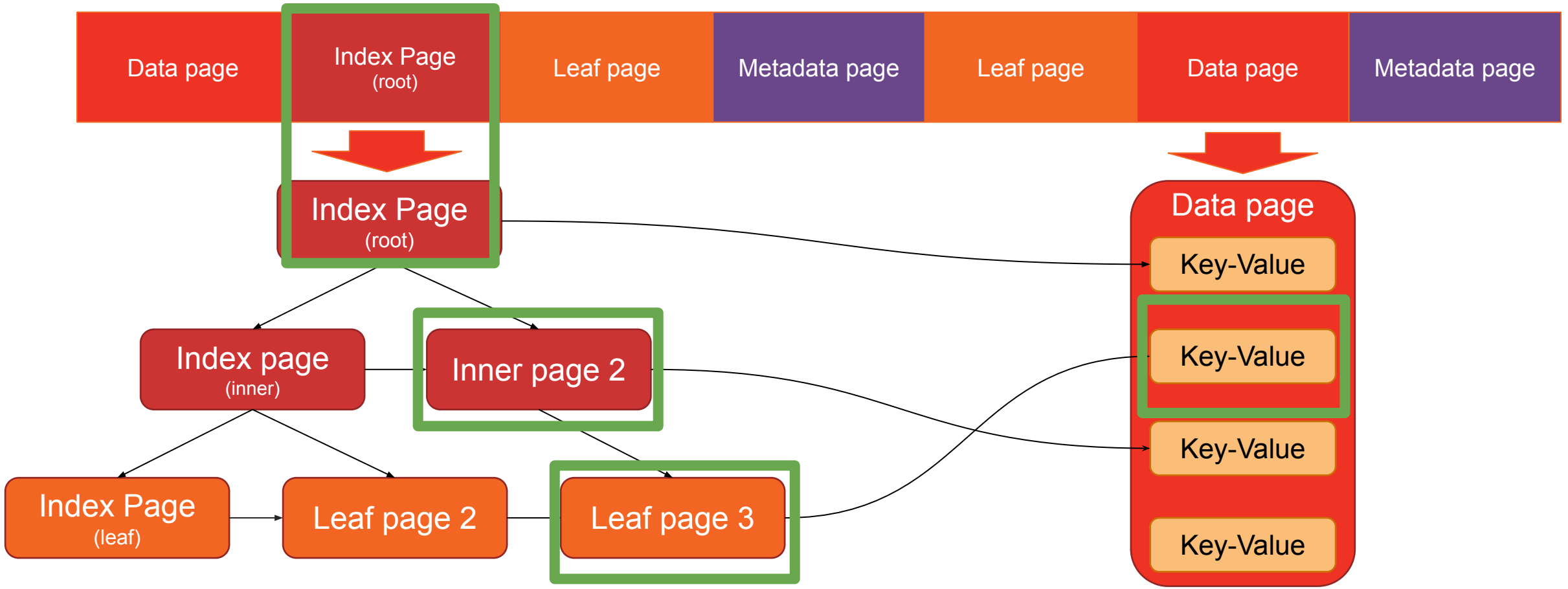
Ignite Memory Tier



GridGain

Ignite B-tree Storage Engine

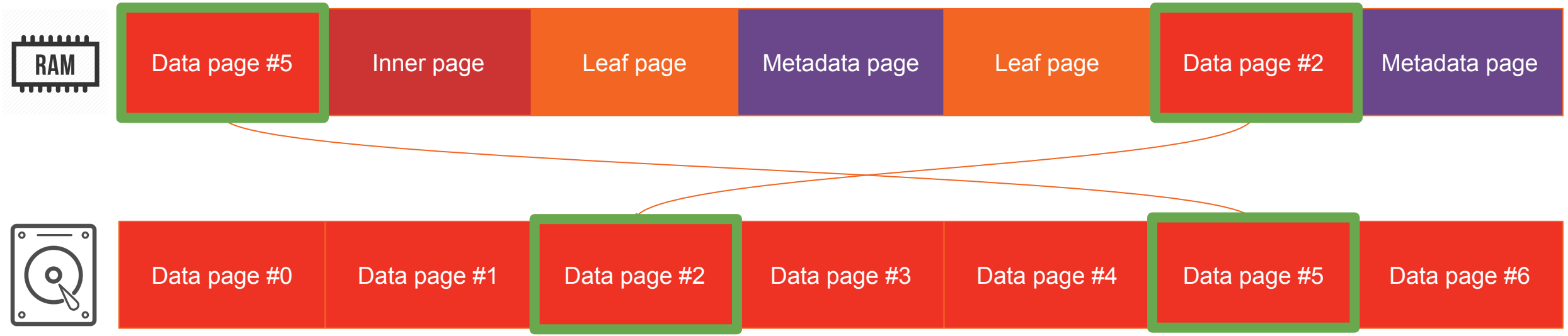
Memory segment



GridGain

Querying Across Memory and Disk

Memory segment



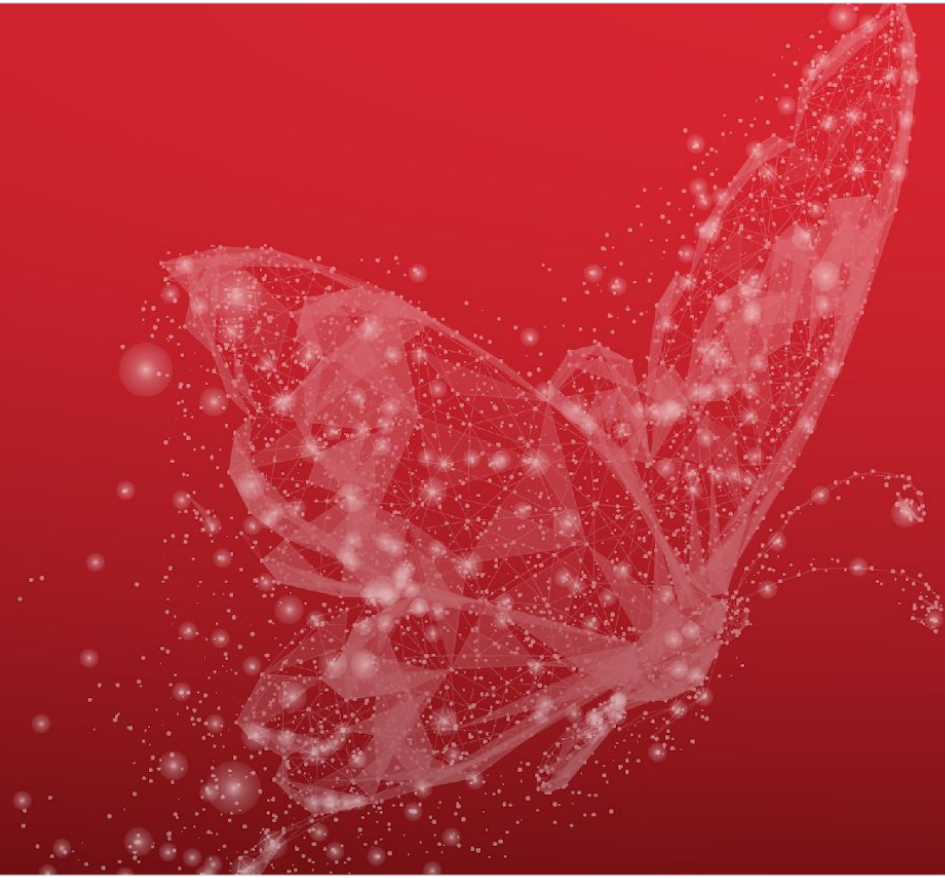
Partition file with Data

PageId → **Pages map** → **Pointer in a memory segment** (read/write ops)

PageId → **Position in a file** (load page/checkpoint)

GridGain

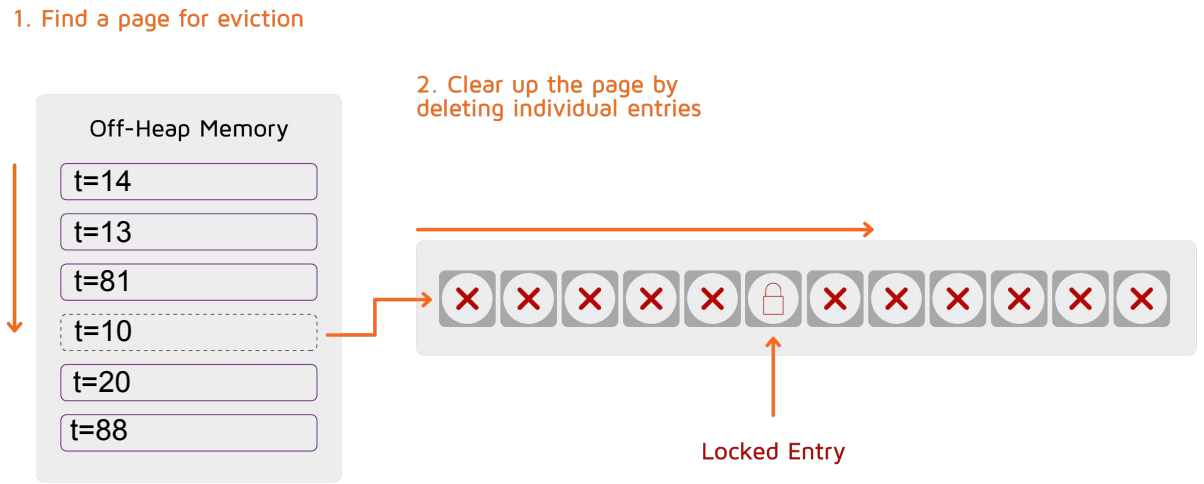
Eviction Policies



GridGain

Eviction Policies for the Off-Heap Memory

- Supported configurations
  - Pure in-memory cluster
  - In-memory cluster with external DBs
- Supported algorithms
  - Random-LRU
  - Random-2-LRU



t - page's last access time

GridGain

Eviction Policies: Configuration

```
1. DataStorageConfiguration storageCfg = new DataStorageConfiguration();
   DataRegionConfiguration regionCfg = new DataRegionConfiguration();
   regionCfg.setName("20GB_Region");

   // 500 MB initial region size (RAM).
2. regionCfg.setInitialSize(500L * 1024 * 1024);

   // 20 GB maximum region size (RAM).
3. regionCfg.setMaxSize(20L * 1024 * 1024 * 1024);

   // Enabling RANDOM_2_LRU eviction for this region.
4. regionCfg.setPageEvictionMode(DataPageEvictionMode.RANDOM_2_LRU);
```

Bringing evicted records back

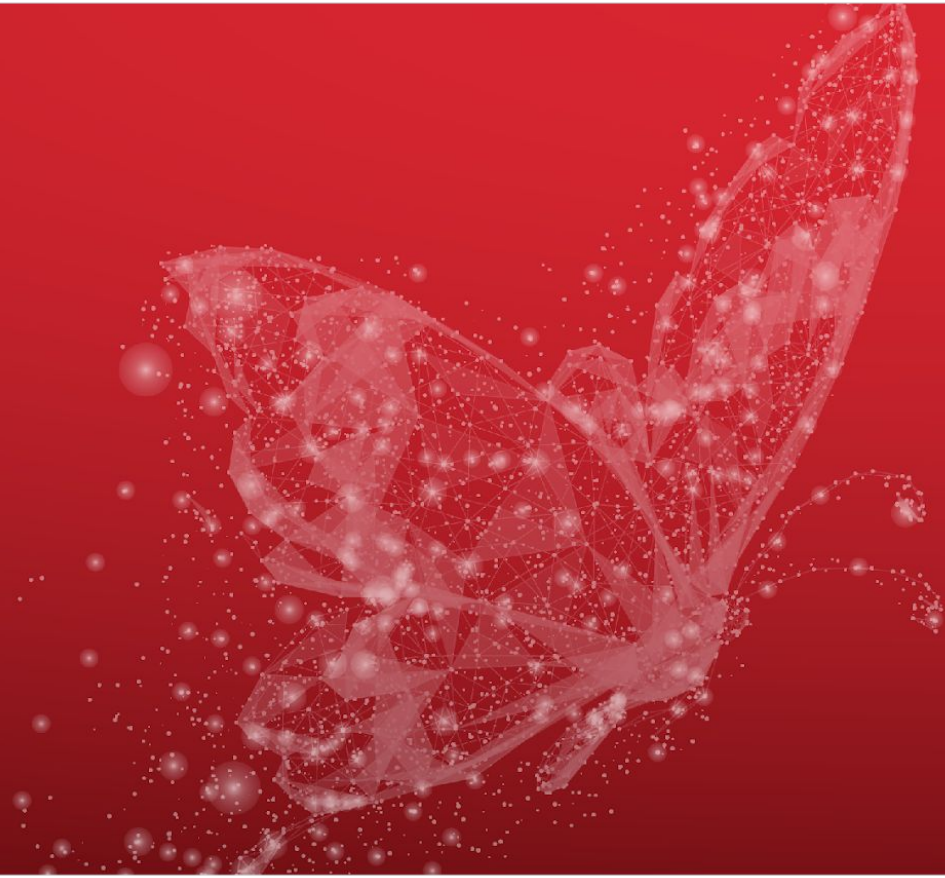
- Ignite + external database
  - Key-value APIs can load missing records from disk
- All other scenarios
  - You need to reload evicted records manually

Eviction Policies for Java Heap

- On-heap caches are **used rarely**
  - https://ignite.apache.org/docs/latest/configuring-caches/on-heap-caching

- Supported for **on-heap caches only**
  - LRU, FIFO, Sorted

Expiration Policies



GridGain

Expiration Policies
Removing unnecessary records proactively

1.
```
CacheConfiguration<Integer, String> cfg =
  new CacheConfiguration<Integer, String>("myCache");
```

2.
```
cfg.setExpiryPolicyFactory(
  CreatedExpiryPolicy.factoryOf(Duration.FIVE_MINUTES));
```

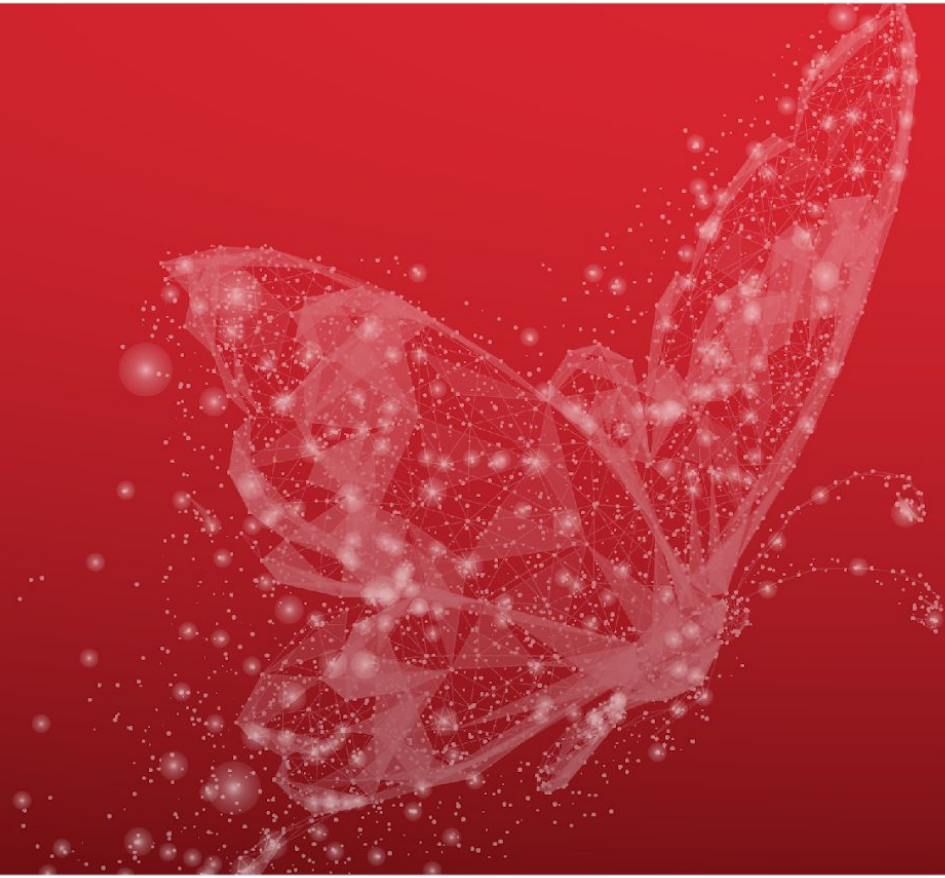
3.
```
cfg.setEagerTtl(true);
```

Different expiration policies for different records

1.
```
IgniteCache cache = ignite.cache("myCache").withExpiryPolicy(
  new CreatedExpiryPolicy(new Duration(TimeUnit.MINUTES, 2)));
```

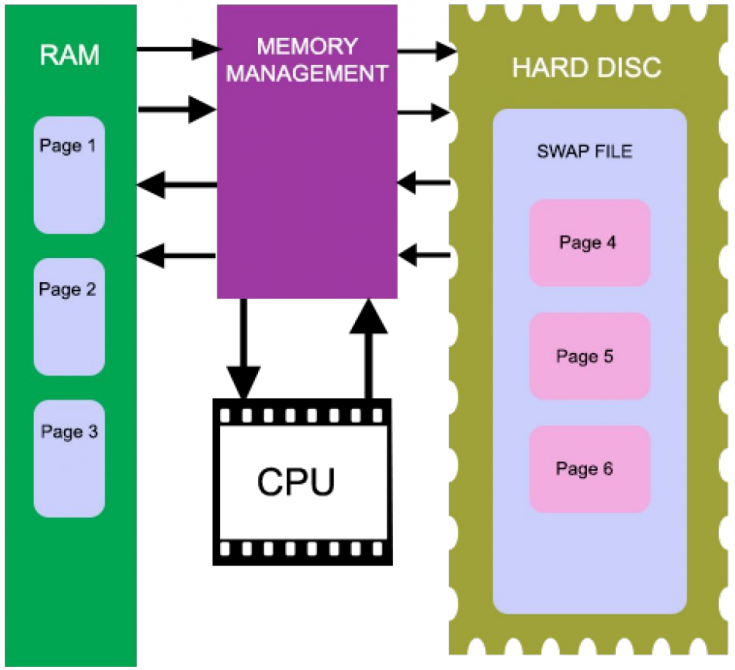
2.
```
cache.put(1, "some value");
```

Swapping



Swapping
As an Out Of Memory Preventive Measure

- Ignite can store data in memory-mapped files
- OS swaps data in/out to balance memory usage
- Swap space is cleared on restarts, records are lost
  - Thus, scale out and get data rebalanced asap
  - Why? To avoid potential data loss
- Swapping might impact latency even if memory is enough
  - See vm.swappinness, vm.extra_free_kbytes, etc.

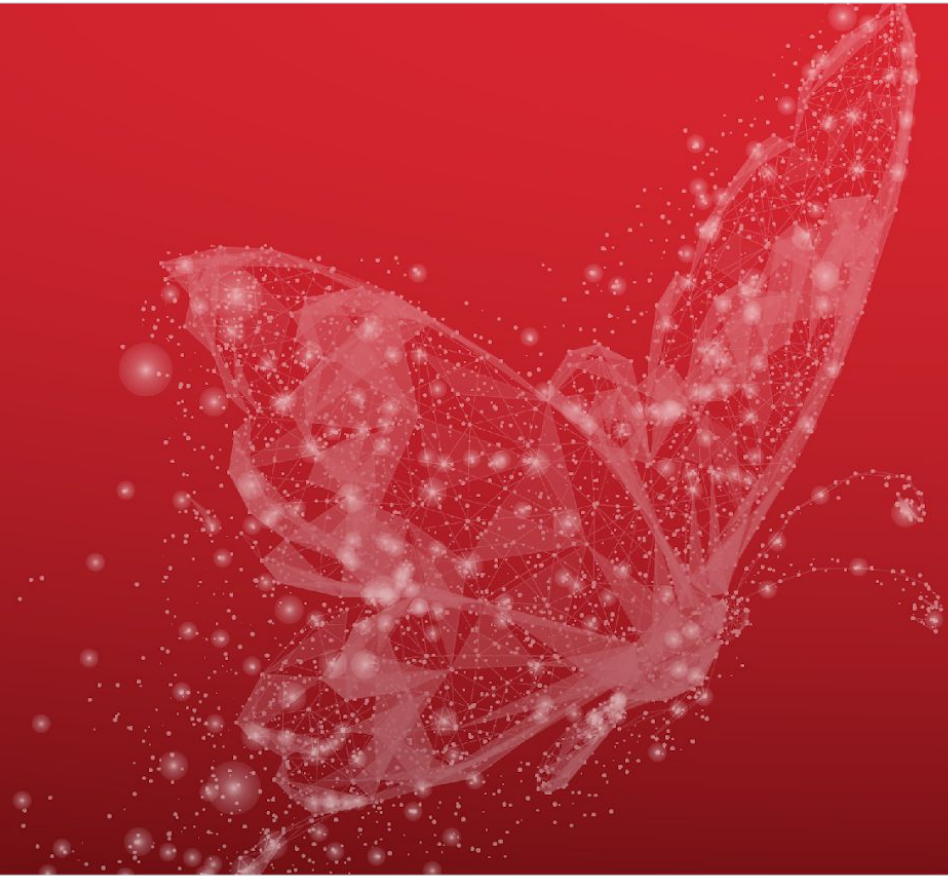


GridGain

Swapping configuration

```
DataStorageConfiguration storageCfg = new DataStorageConfiguration();
DataRegionConfiguration regionCfg = new DataRegionConfiguration();
1. regionCfg.setName("500MB_Region");
regionCfg.setInitialSize(100L * 1024 * 1024);
regionCfg.setMaxSize(5L * 1024 * 1024 * 1024);

// Enable swap space.
2. regionCfg.setSwapPath("/path/to/some/directory");

// Setting the data region configuration.
storageCfg.setDataRegionConfigurations(regionCfg);
```

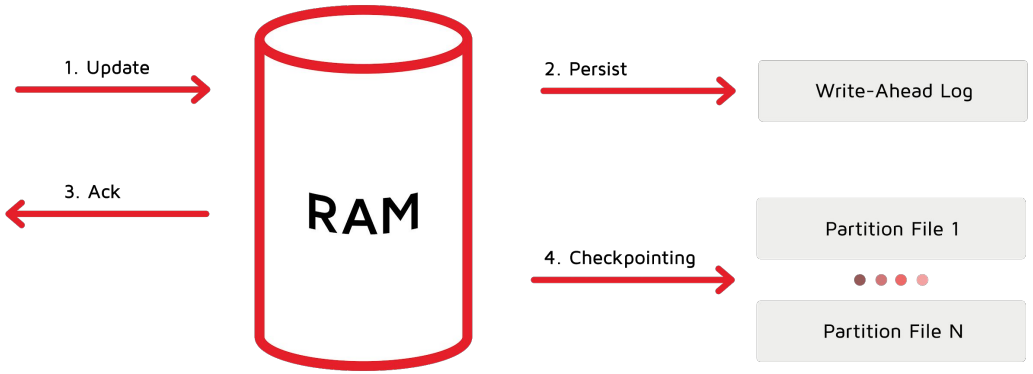
Ignite Native Persistence



GridGain

Ignite Native Persistence

- Distributed Persistence Tier
  - Fully transactional and consistent
  - No need to cache 100% of data in RAM
  - No need to warm-up RAM on restarts
- Performance vs. Cost Tradeoff
  - Cache more for fastest performance
  - Cache less to reduce infrastructure costs

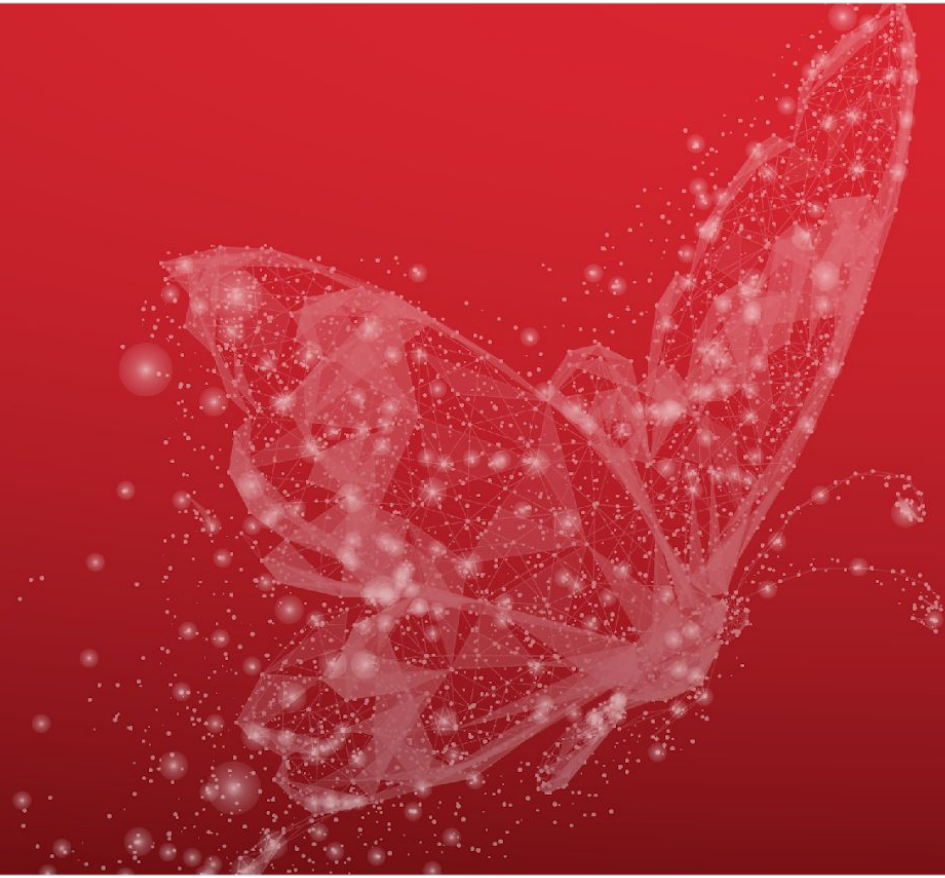


GridGain

Enabling Ignite Native Persistence

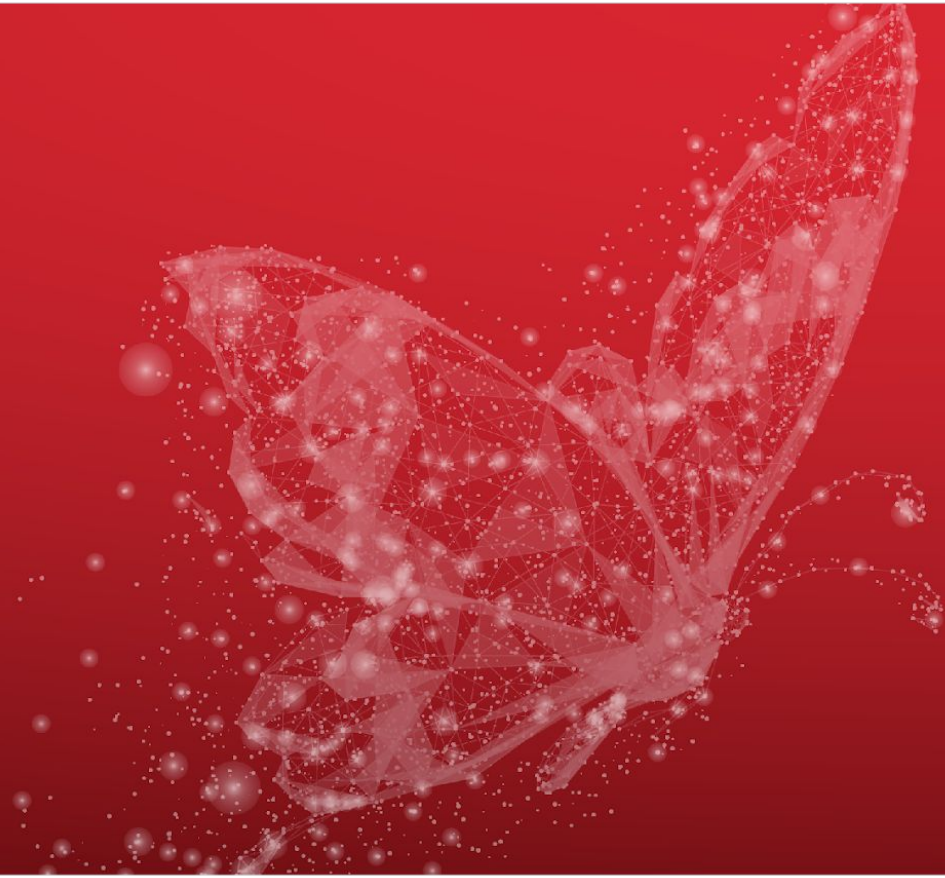
```
DataStorageConfiguration storageCfg = new DataStorageConfiguration();

storageCfg.getDefaultDataRegionConfiguration().setPersistenceEnabled(true);
```

Demo



GridGain

SQL Memory Quotas



GridGain

Query Execution Phases

```
SELECT AVG(population) FROM City
```

=

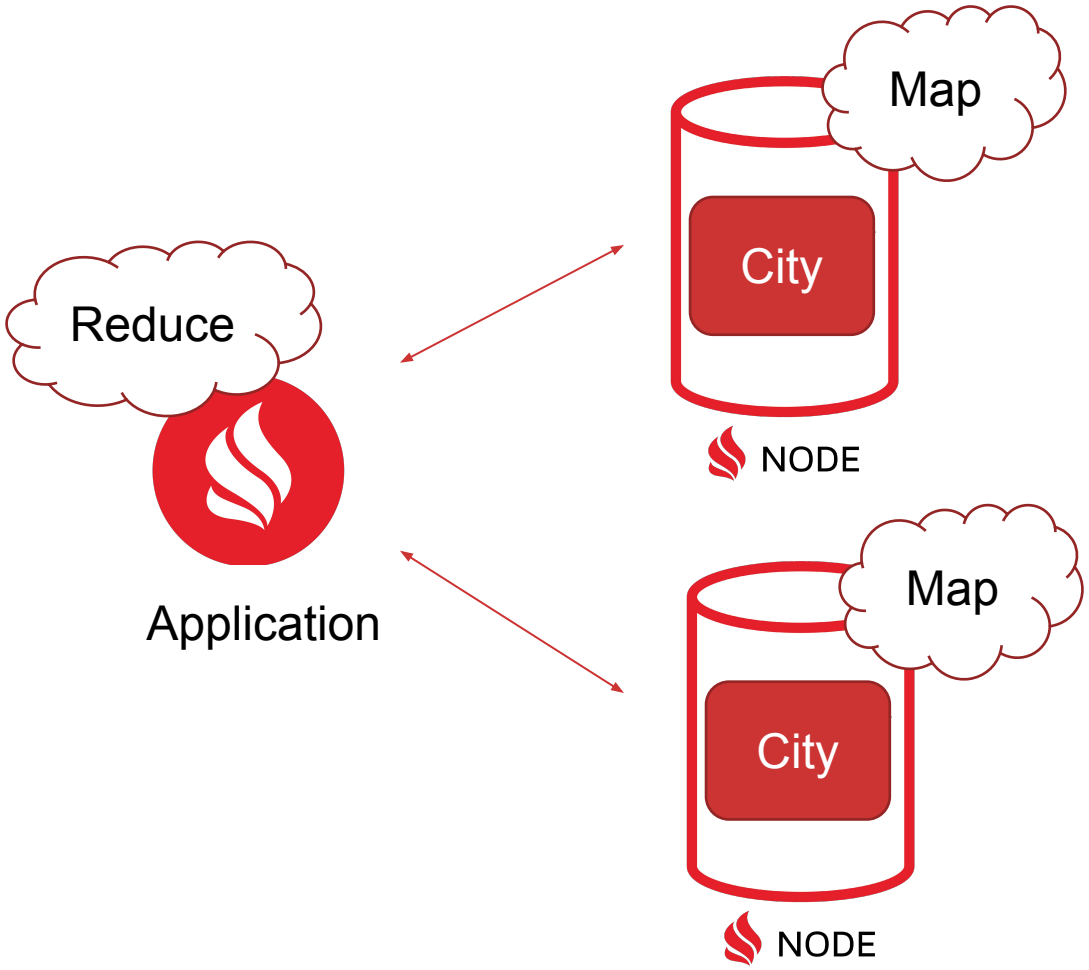
Map

```
SELECT SUM(population) as sum0,
COUNT(population) as count0
FROM City
```

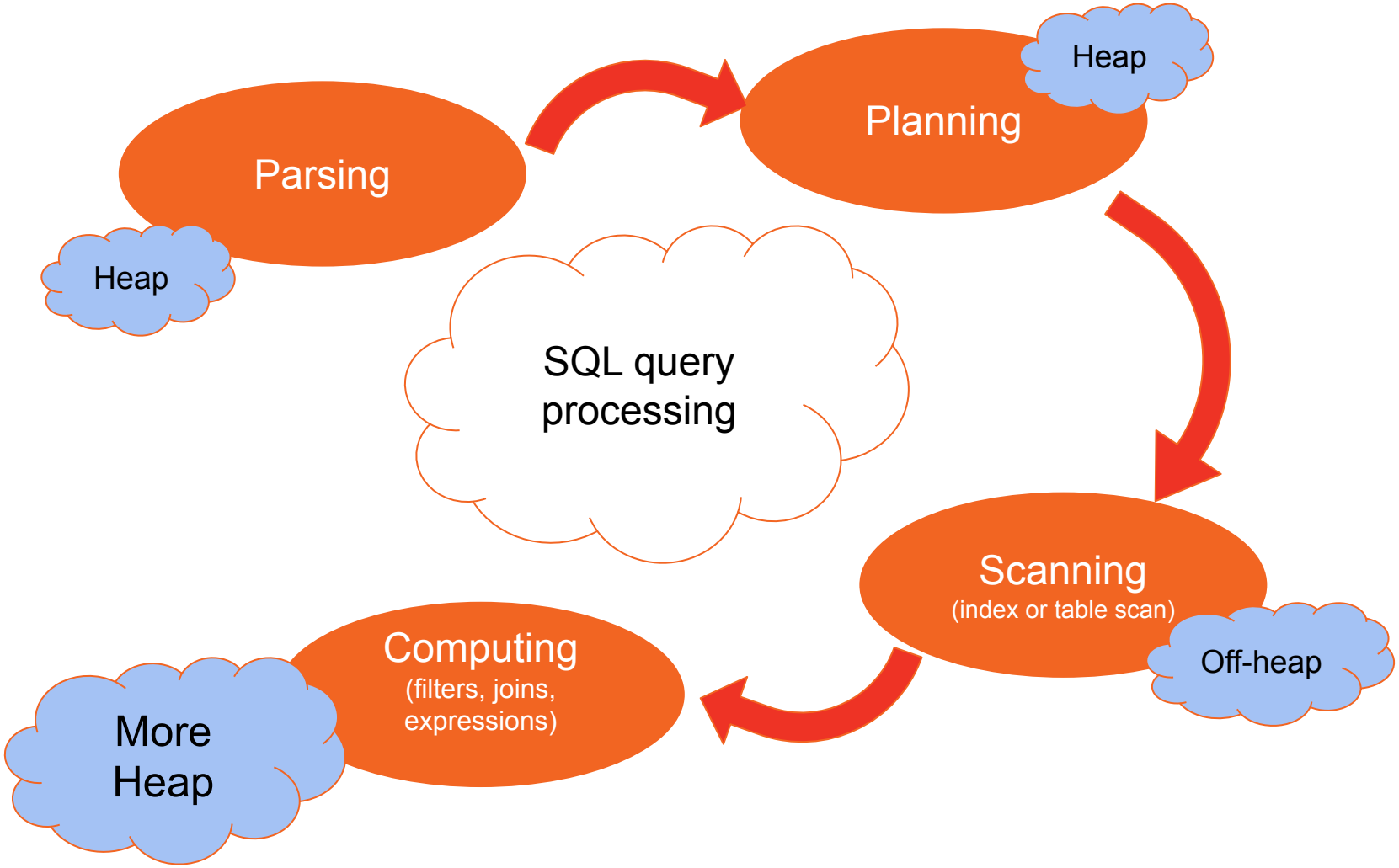
+

Reduce

```
SELECT SUM(sum0)/SUM(count0)
FROM resultTable
```



Java Heap Usage During Query Execution



GridGain

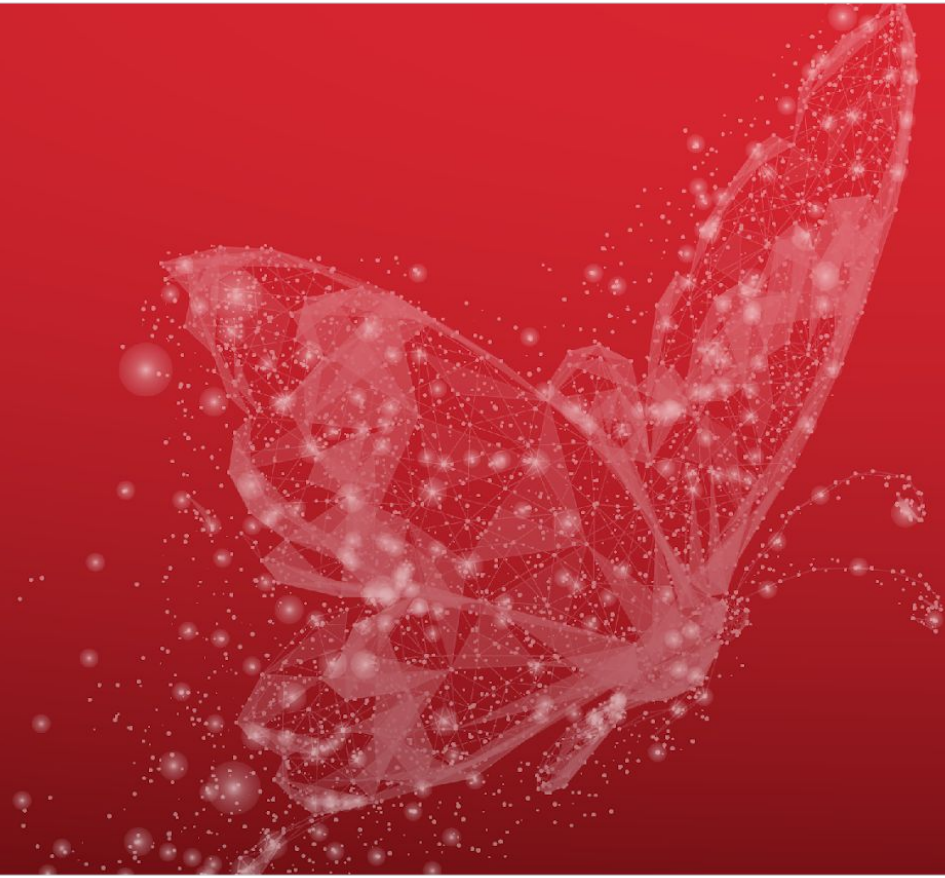
Memory Quotas Configuration

```
1. IgniteConfiguration cfg = new IgniteConfiguration();
   SqlConfiguration sqlCfg = new SqlConfiguration();

   // All running SQL queries combined cannot use more memory as set here.
2. sqlCfg.setSqlGlobalMemoryQuota("500M");

   // A single running SQL query cannot use more memory as set below.
3. sqlCfg.setSqlQueryMemoryQuota("40MB");

   // If any of the quotas is exceeded, a result set will be offloaded to disk.
4. sqlCfg.setSqlOffloadingEnabled(true);

5. cfg.setSqlConfiguration(sqlCfg);
```

When Should You Use Quotas?

- Sorting (ORDER BY)
- Grouping (DISTINCT, GROUP BY)
- Complex subqueries
- Complex analytical queries

Summary

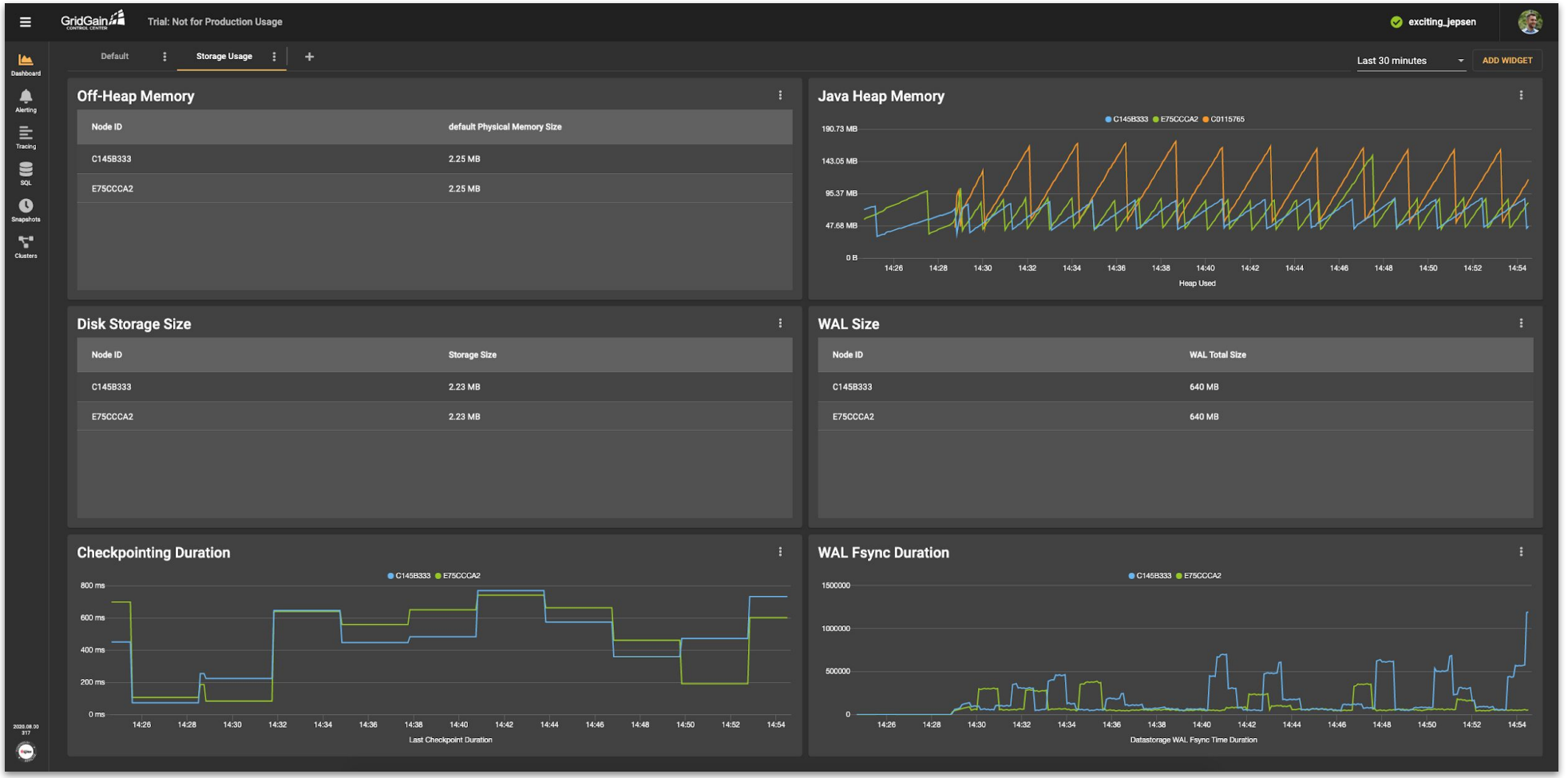


GridGain

Cheatsheet

- Prefer Ignite Native Persistence as OOM preventive measure
  - Otherwise, check eviction and expiration policies, or swapping
- Use memory quotas for SQL (especially for analytics)
  - Don’t forget to enable the offloading to disk feature
- Scale out the cluster when you’re running out of memory
  - To have the best performance characteristics

Monitor Memory Usage With GridGain Control Center



https://www.gridgain.com/docs/tutorials/management-monitoring/overview

GridGain

Don’t miss Ignite talks at the IMCS Summit
October 28-28th

https://www.imcsummit.org/2020/virtual/

GridGain

Stay connected with Apache Ignite users & experts

meetup.com/Apache-Ignite-Virtual-Meetup/