redhat.

HACEP

Highly Available and scalable CEP

Ugo Landini
Senior Solution Architect, Red Hat

versione 1.2.1
14 Jul 2016

Agenda

- The case for HACEP
- Glossary
- High Level Architecture
- Deep Dive & Internals
- Roadmap
- Customers & Use Cases
- Conclusions

DroolsInfinispan

The Case for HACEP

redhat.

DroolsInfinispan

CEP with Drools

- **C**omplex **E**vent **P**rocessing
- Rules + Time
- Adding the concept of **Time** to basic Drools rules
  - Sliding windows
  - Entry points
  - Time operations

Detect events of significance to a business by recognizing **time-based patterns** in one or more real-time data feeds...

Benefits:

'Active decision making': Business makes timely decisions & responds quickly to external events

HA Use Cases

- Out of the box, Drools **doesn't provide** neither an HA architecture nor an horizontal scalability solution
  - Everything must be in a **single** session
- Many use cases needs a solution for that:
  - CEP with **big sliding windows** usually consumes a lot of RAM
  - whenever a single session is not enough

Sample Use Case

- User does something **T** times
- User does something **T** times for **D** consecutive days
- User places **X** actions in **D** days
- User wins/loses more than **X**
- User wins/loses a cumulative **X** amount

User gets rewards, in near real time

Sample Use Case

- **10M** events per day
- **1M** registered users
- CEP sliding window in the **~30** days
- **8k** concurrent users per day
- **90k** unique users in **30** days
- **~200** bytes per event
- **1 second** available (end to end) to run all user rules and process rewards

Some Numbers

- **~200** bytes * **10M** events * **30** days is **~60GB** just for the "raw" facts
- BRMS Sessions contains **much more** than just the events
- **128Gb** heaps at a minimum would be needed to store everything in a single session
  - no HA
  - no scaling out

HACEP 1.0 Features

- **Linearly** scalable from 2 to 100s of nodes
- **Dynamically** scaling up and scaling down
- Survives to multiple node **failures**
- **In-memory** read/write performance for **extreme** throughput
- **Dynamic** CEP rules **recompiling**
- Several **disk** storage options
- **Minimal** footprint
- **Rolling upgrades** support
- Plain **JVM** or **EAP** support

The Case for HACEP

- **HACEP** uses **Infinispan**, **Camel** and **ActiveMq** to make Drools session scalable and highly available
- **HACEP** is a generic solution and impose just a partitioning criteria: no other constraint/limitations

Use cases

- **All CEP** use cases, in particular
  - **Gaming**
  - **Financial** (fraud detection, etc.)
  - **IoT**

HACEP Patterns

- HACEP is **designed** on some **fundamental** patterns
  - **Sharding**/Horizontal Partitioning
  - Data **Affinity**
  - Event **Sourcing**

Sharding

Sharding

- aka **Horizontal Partitioning**
- **splitting** Data in separated nodes in order to improve **performance** and **scalability**

DroolsInfinispan

Data Affinity

redhat.

DroolsInfinispan

Data Affinity

- Data Affinity means **colocating data** together to improve performance and scalability
- Data Affinity means **colocating computing code** with **data** too

Event Sourcing

Event Sourcing

- http://martinfowler.com/eaaDev/EventSourcing.html
- The fundamental idea of Event Sourcing is that of ensuring **every change to the state** of an application is captured in an **event object**, and that these event objects are themselves **stored in the sequence** they were applied for the same lifetime as the application state itself

DroolsInfinispan

High Level Architecture

redhat.

10 thousand foot

- Store everything in the **Data grid** to overcome the single JVM limitation
- Treat Drools CEP as a **stateless** module
- Leverage **Sharding** and **Data Affinity** to optimise network hops and make Drools scalable
- Basically, use Infinispan as Drools **distributed working memory**

Sharding

- Drools sessions partitioned in different nodes on a **particular user defined criteria**
- Gaming: sharding **per player**
- FSI: sharding **per customer/cc**

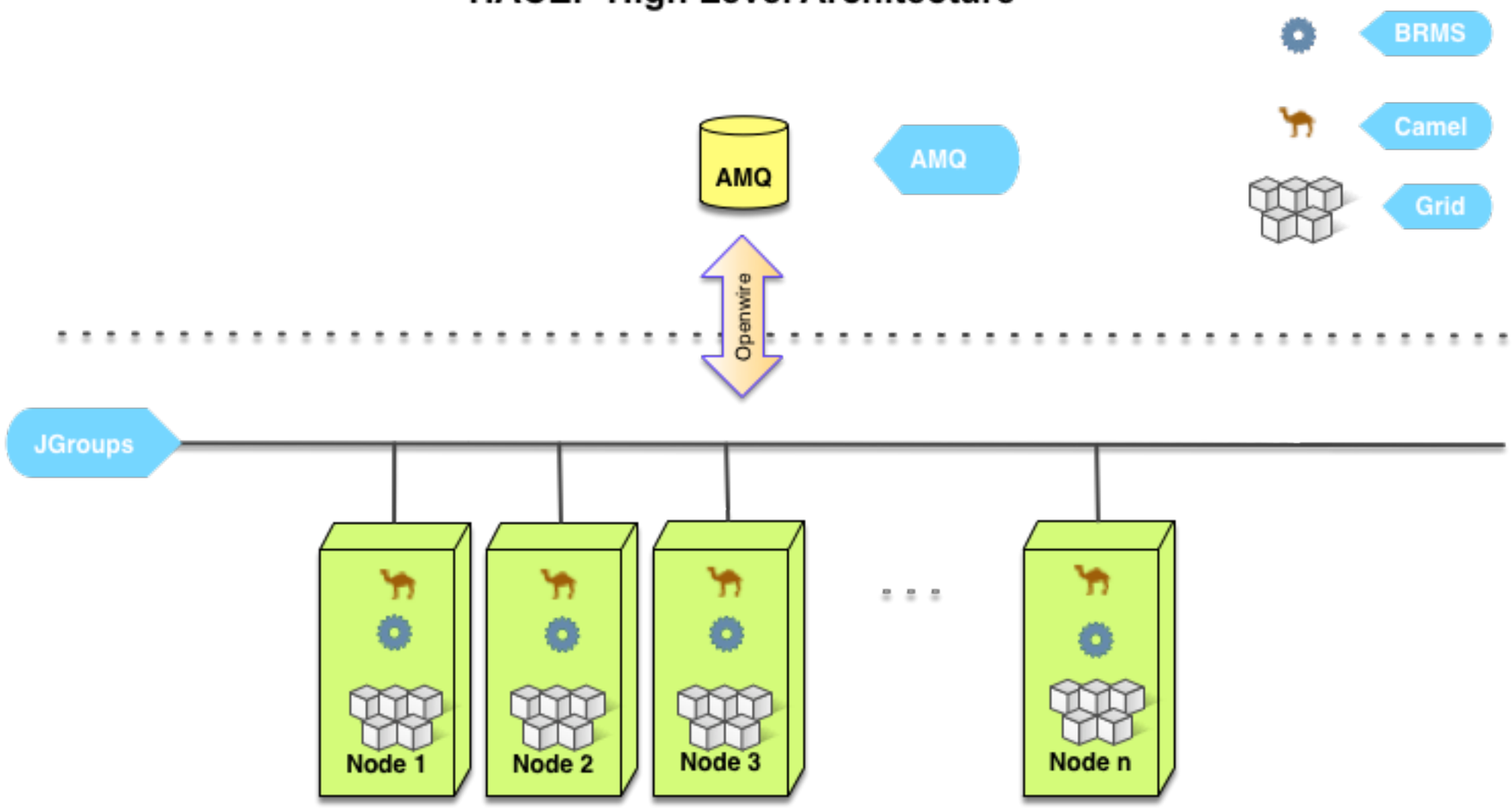
Data Affinity

- Events and Sessions both sharded on **same nodes**
  - i.e. using same **user defined partitioning criteria**
- Drools code **running on** the node containing needed data

Data Affinity

- Events must be related to a **group**, so we can partition them and Data Affinity will be our friend
- a **group** is whatever business criteria we can use to partition (player, cc, location, etc.)
  - *you can't have generic cross-group rules!*

HACEP High Level Architecture



DroolsInfinispan

Deep Dive Architecture

redhat.

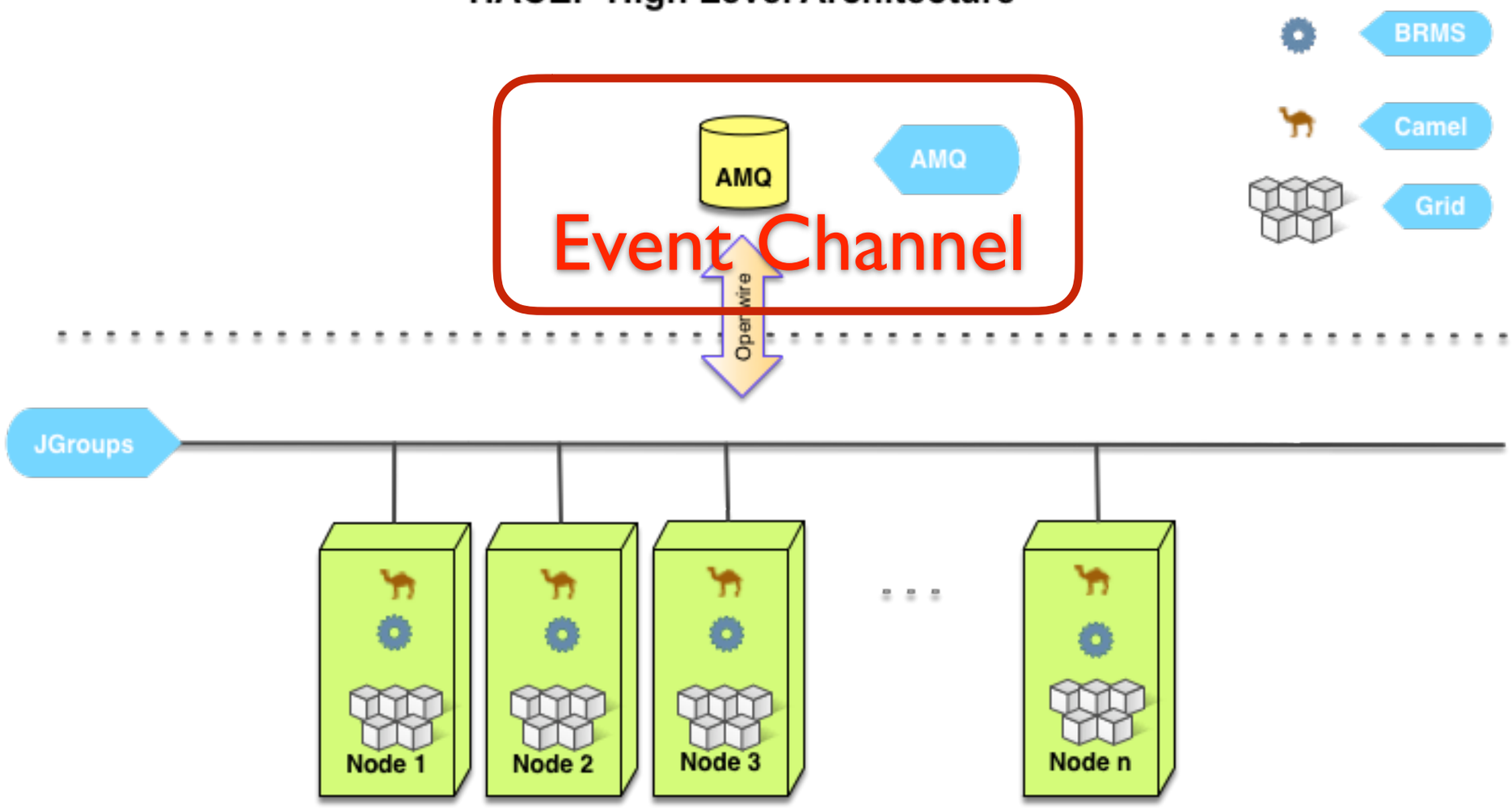
HACEP Nodes

- Each HACEP node is **identical**
- Each node contains:
  - a **Camel** route
  - a portion of the data, in 2 different **Infinispan** caches
  - **Drools** code

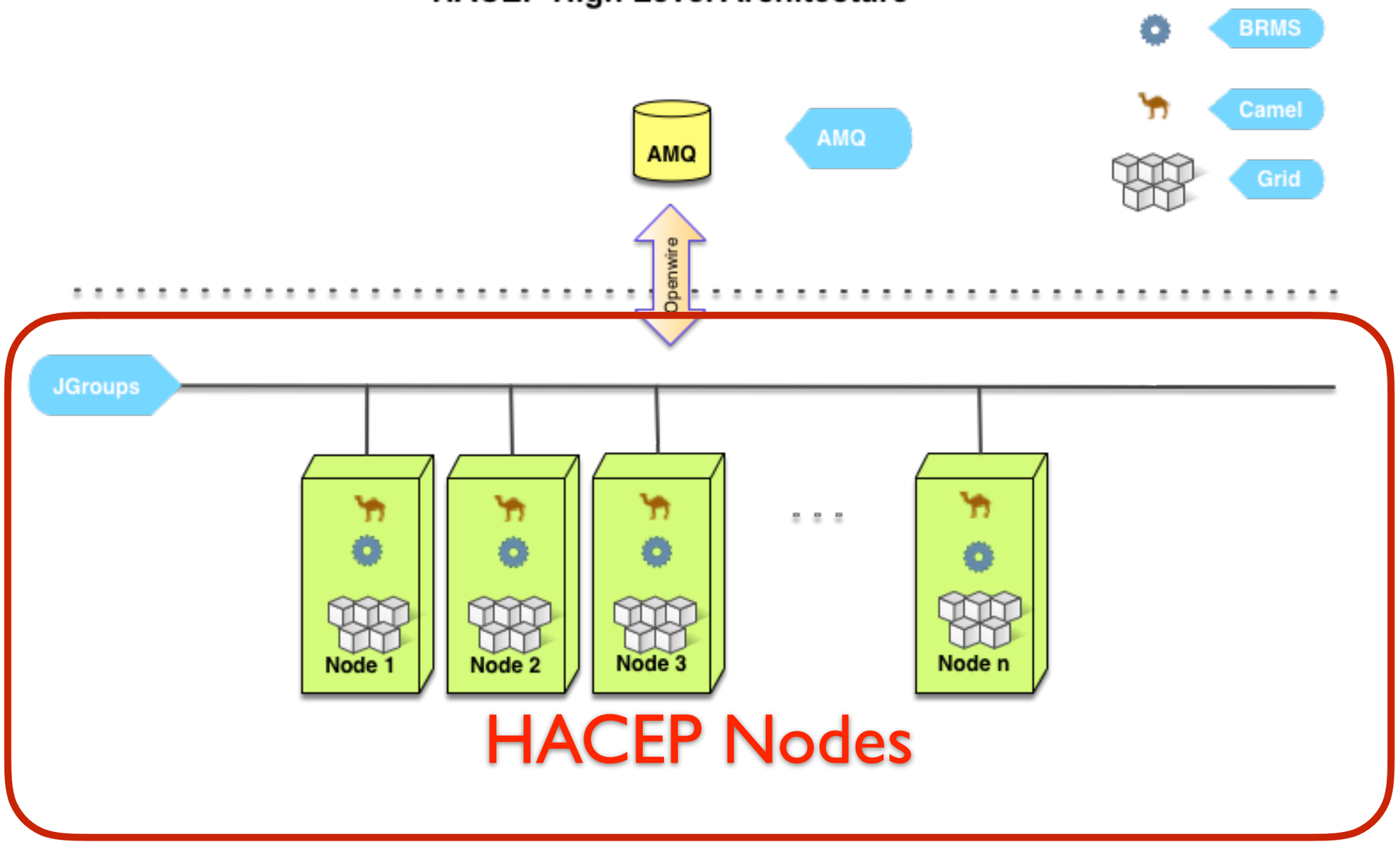
HACEP Nodes

- **Event Channel** is external to HACEP nodes
- Could be anything:
  - typically some kind of **Queueish** software (AMQ, AMQP, JMS, Kafka…)

HACEP High Level Architecture



HACEP High Level Architecture



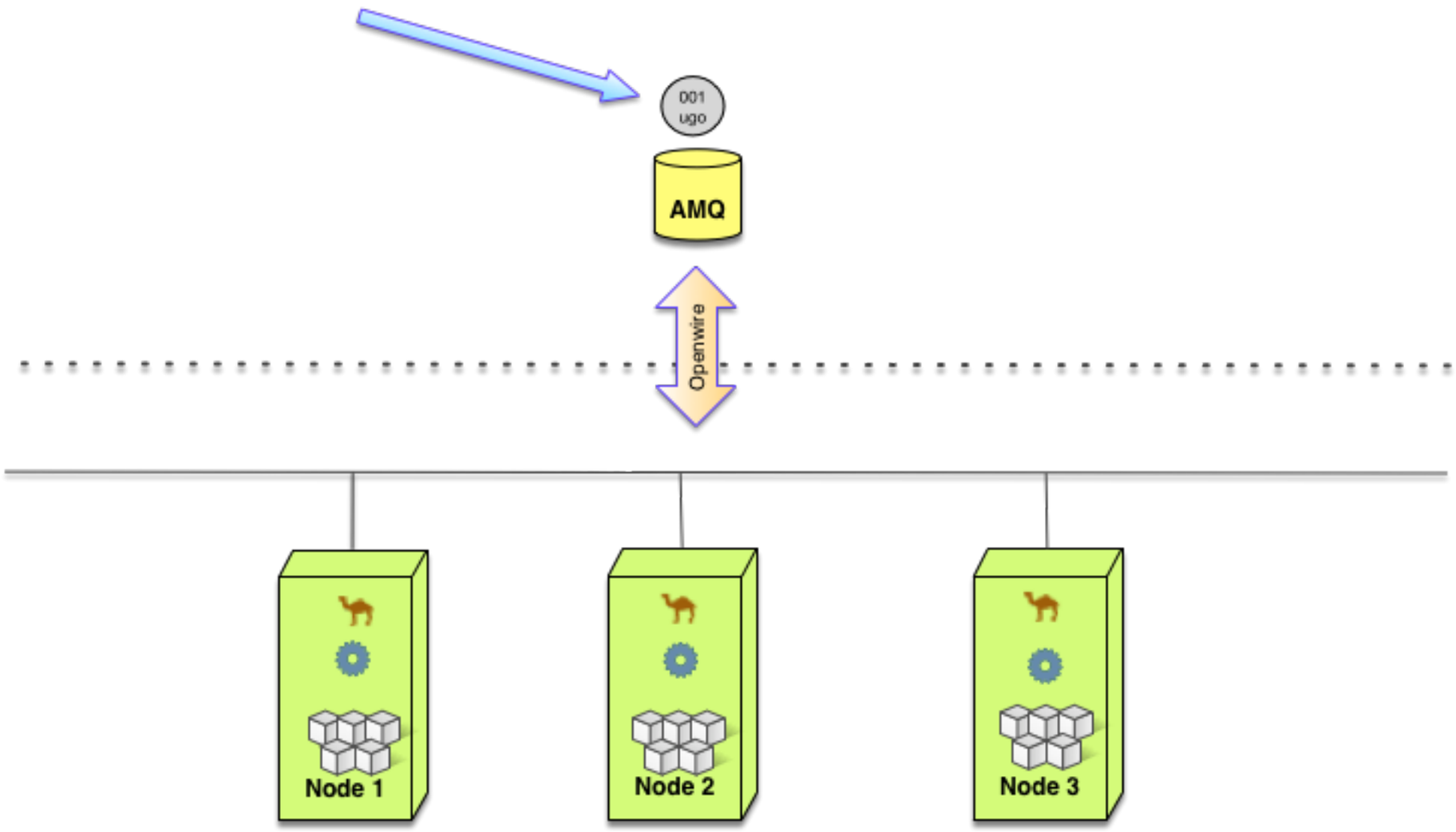
HACEP Nodes

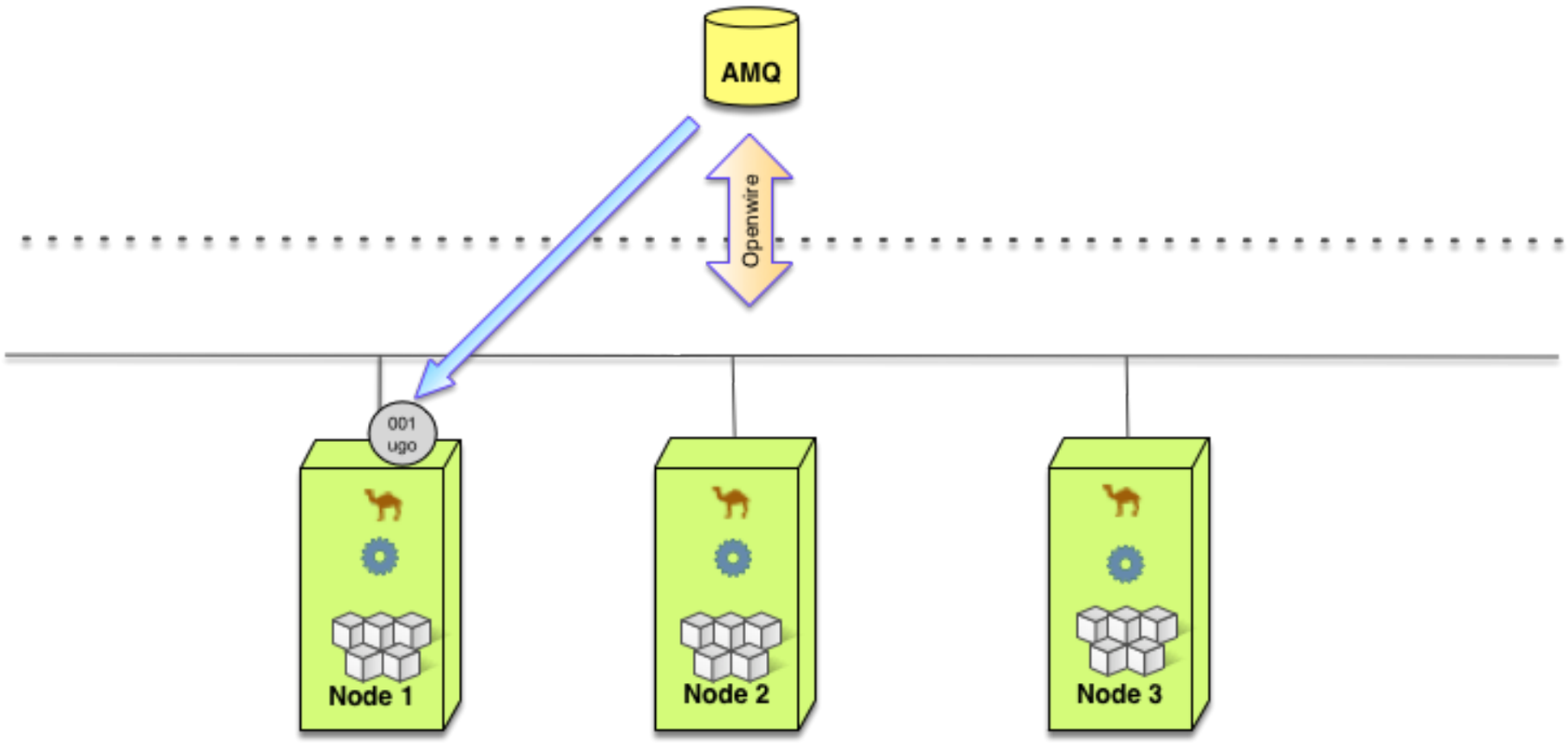
Camel Route

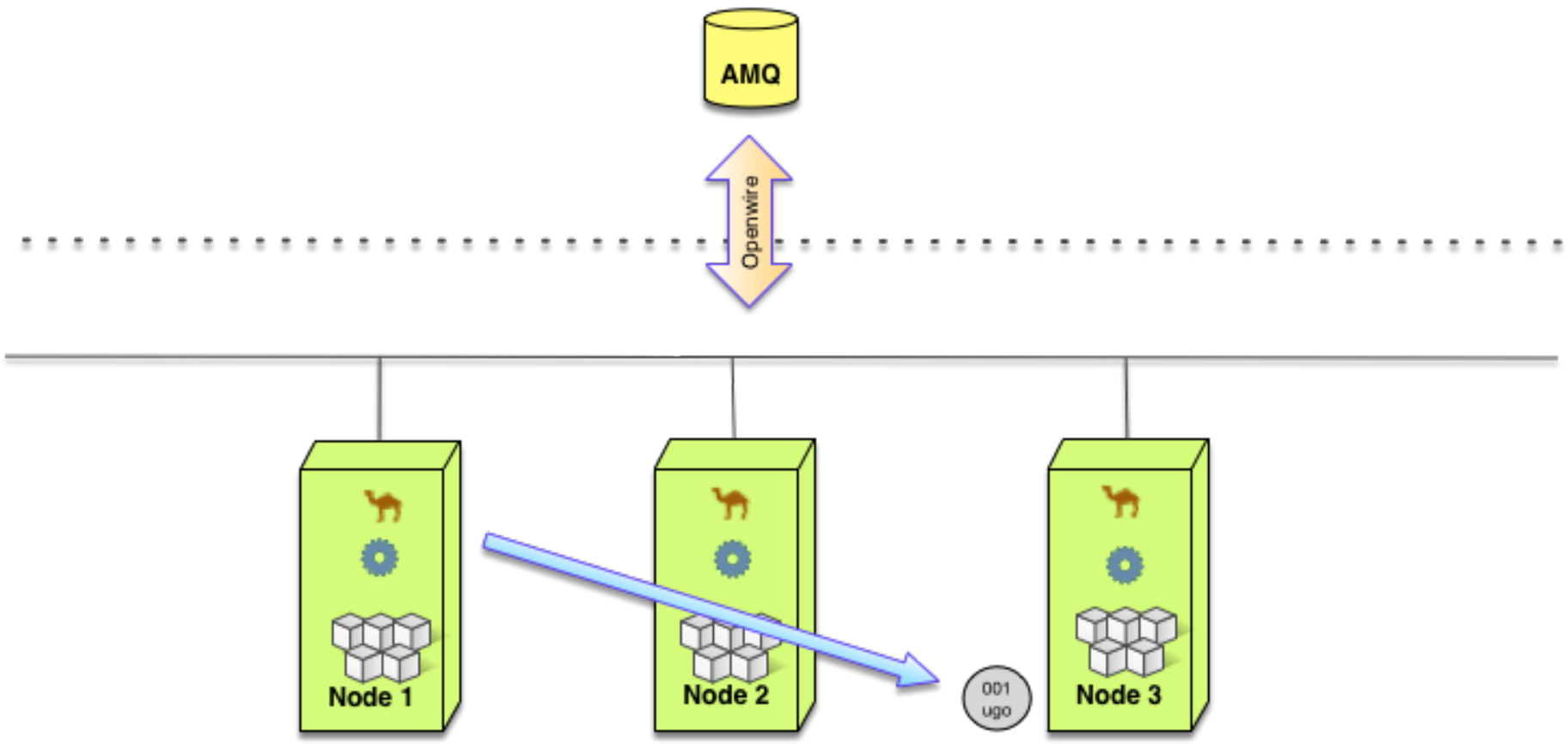
- The Camel route:
  - gets events from an Event Channel
  - puts the events in the **Facts** cache in Infinispan
    - Infinispan is configured with a **Distributed** Topology
  - **Grouping** is enabled
  - Events expires after a few milliseconds idle time

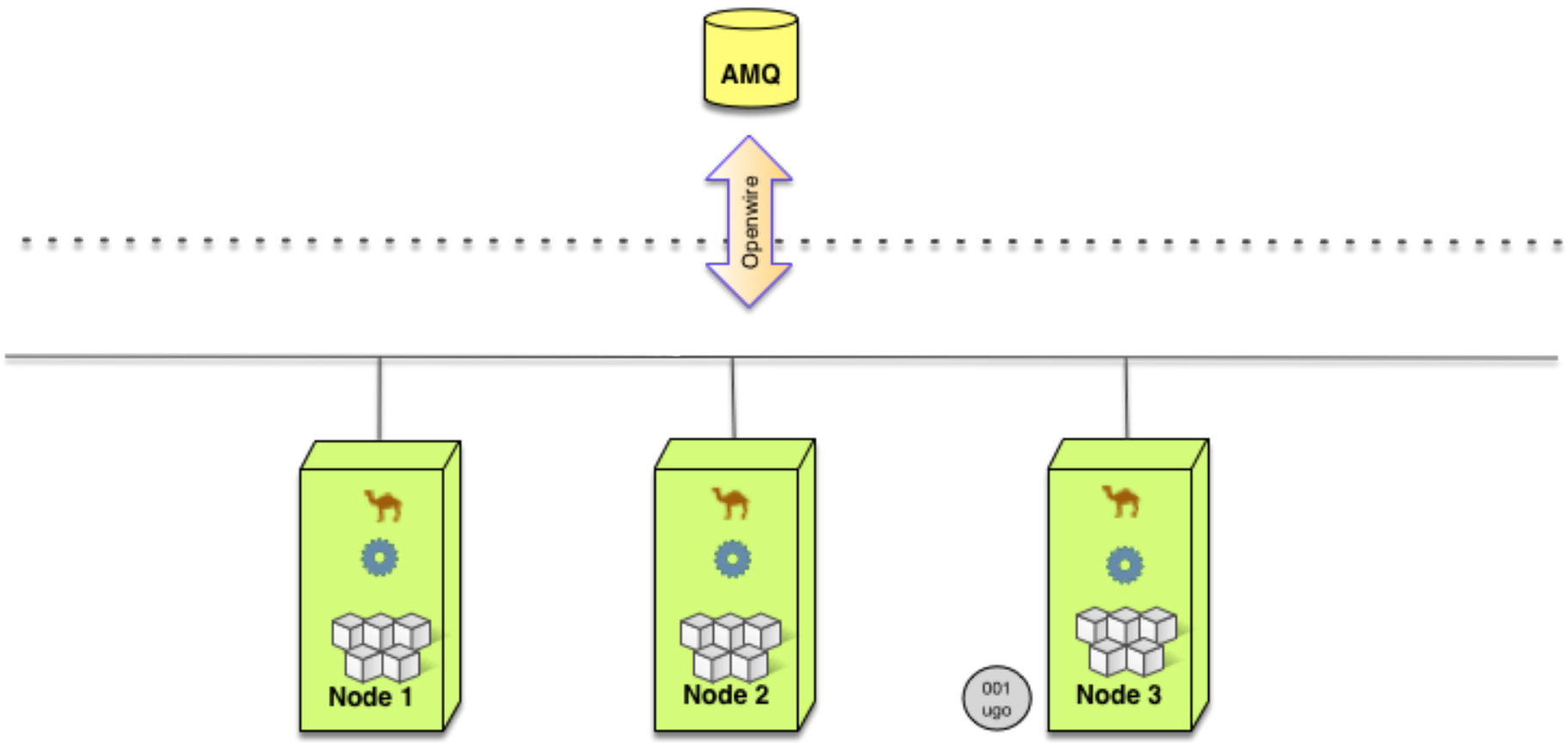
Why Expiration?

- We don't care about immediately **storing** the facts:
  - we put the facts in the Grid just to fire a notification
  - the notification is **synchronous** and happens only on the **primary** node
  - that means that the notification will be fired only on the node which will be (or already is) the primary node for that particular **group**


001
ugo
AMQ
Openwire
Node 1
Node 2
Node 3


AMQ
Openwire
001
ugo
Node 1
Node 2
Node 3


AMQ
Openwire
Node 1
Node 2
001
ugo
Node 3


AMQ
Openwire
Node 1
Node 2
001
ugo
Node 3

Camel Route

- So the **Camel Route** just gets the event and puts it in a special **JDG** cache
  - this cache **doesn't** store the events…
  - … but the notification system gives us the opportunity to find the **pertaining node** (remember Data Affinity?)

Finding the node

- the **synchronous** and **primary** only listener will receive the event
- so, every event with the same group (player id, cc number, etc.) will be **always** notified **on the same node**

Getting the Session

- the listener will **get** the specific session form the Grid
  - or will create an **empty** one if it's not already in the grid
- sessions are **grouped** with same events criteria too, so everything is happening locally
  - Data Affinity again!

…and finally, Drools

- the **event** will be added to the session
- the **pseudoclock** will be advanced accordingly
- the **rules** will be fired
- the updated session will be **saved** again in the Infinispan **Session** cache.
  - remember, everything is local

“Stateless” Drools

- the **Drools state** lives only on JDG
- the Drools session of a single group (player, customer, ecc.) should never be more than a **few megabytes**

Note: events are buffered with DeltaAware. No extra bit will move on the network if not really needed!

DroolsInfinispan

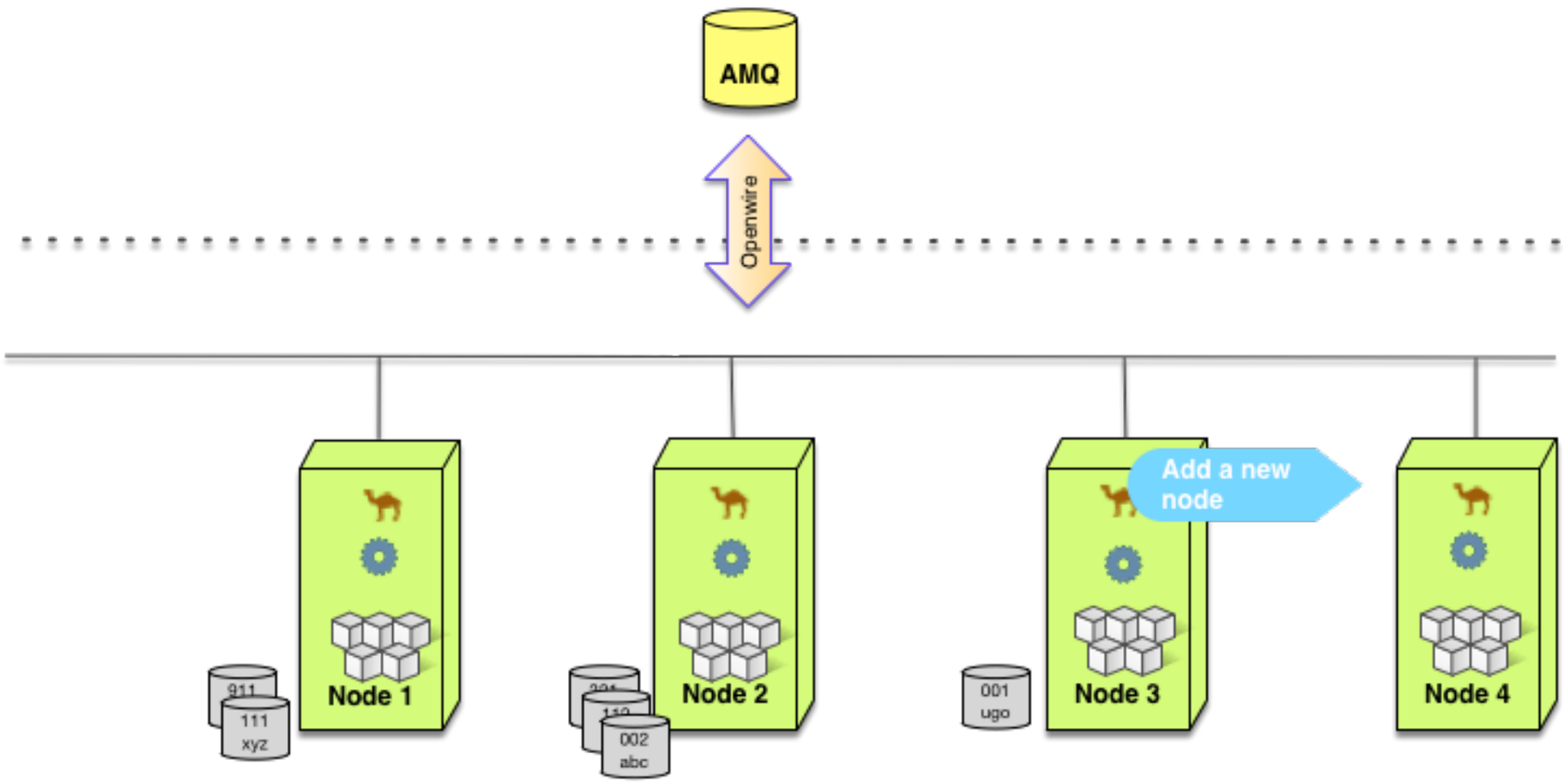
Data Grid Nirvana

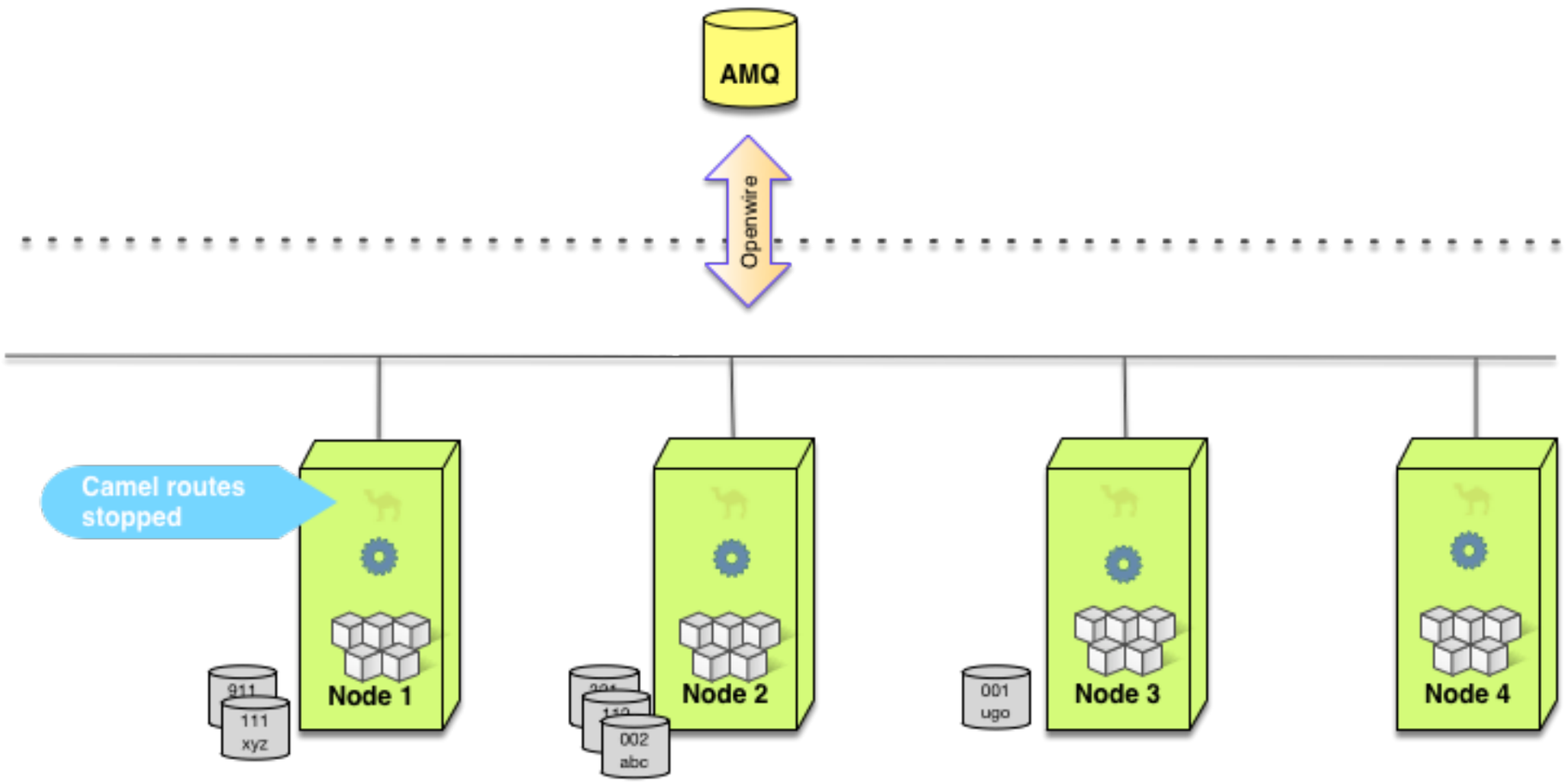
- *“All data are at the distance of a local Java call”*
- HACEP is 100% Nirvana compliant!

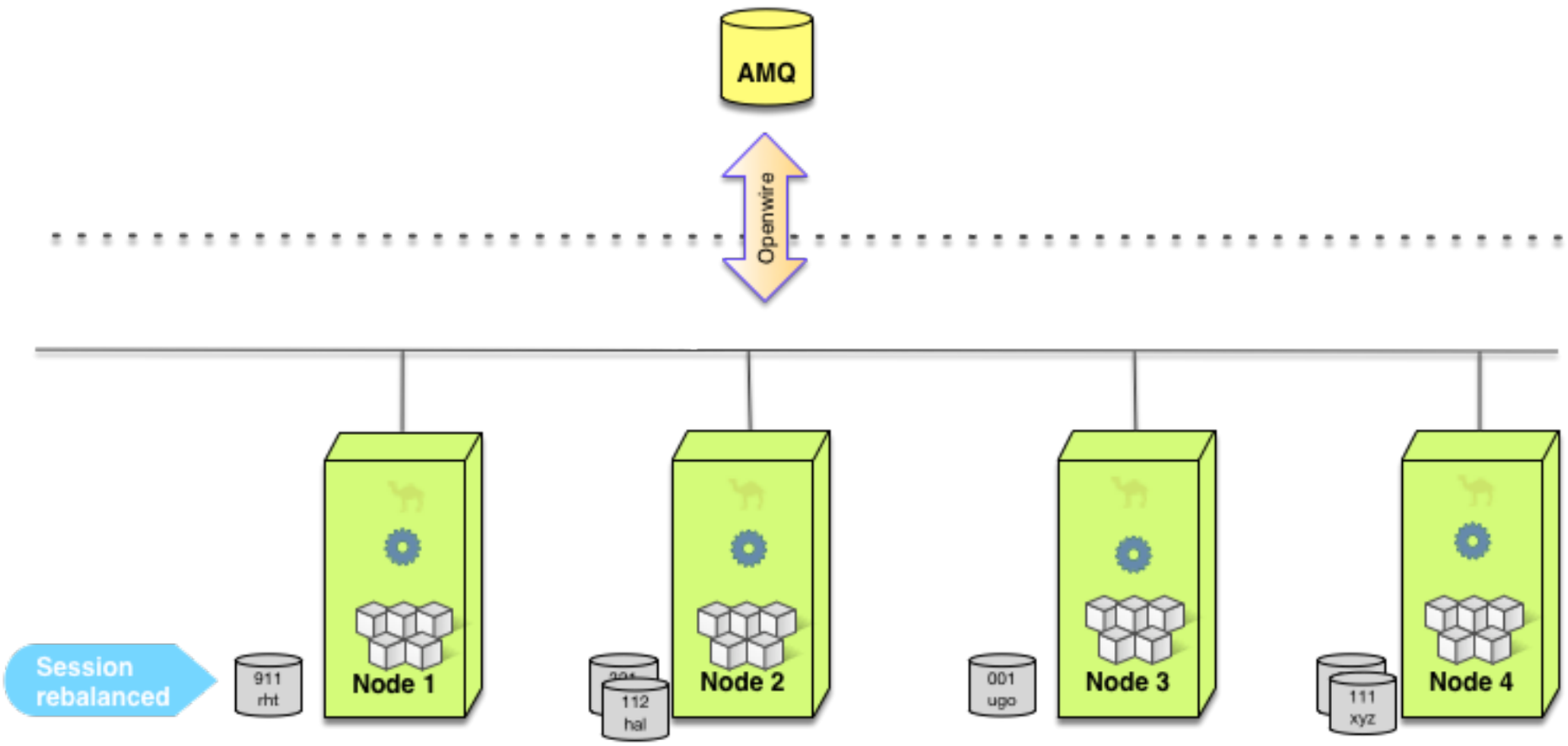
redhat.

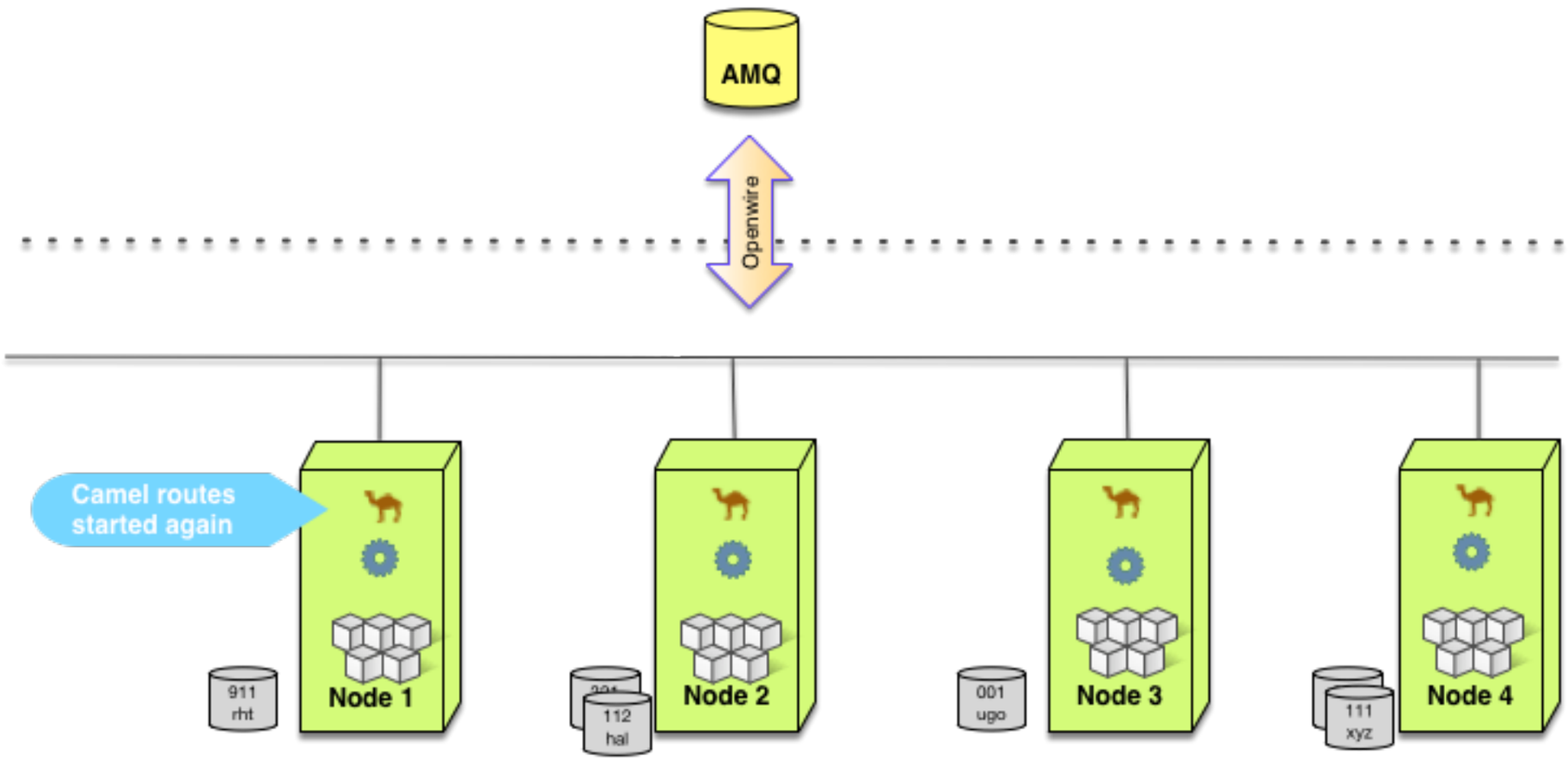
HACEP Topology

- if a node is added/removed to the cluster
  - camel routes are automatically **stopped** when a rehashing event begins in the cluster
  - and **started** again when rehashing finishes


AMQ
Openwire
Node 1
911
111
xyz
Node 2
002
abc
Node 3
001
ugo
Add a new node
Node 4


AMQ
Openwire
Camel routes stopped
Node 1
911
111
xyz
Node 2
002
abc
Node 3
001
ugo
Node 4


AMQ
Openwire
Session rebalanced
911
rht
Node 1
112
hal
Node 2
001
ugo
Node 3
111
xyz
Node 4


AMQ
Openwire
Camel routes started again
911
rht
Node 1
112
hal
Node 2
001
ugo
Node 3
111
xyz
Node 4

HACEP Topology

- Sessions will be redistributed following their **Consistent Hashing**, but always with the **right** Group
- From now on, **Events** will just flow in the **right nodes**

Events Ordering

Events Ordering

- Events Ordering may or may not important for the Customer Use case
- Events Producer is **external** to HACEP
- **HACEP** proposes two different designs
  - JMS **Grouping**
  - **Reordering** component

JMS Grouping

- **JMS Grouping** could be used on Event Source server and is the preferred solution
- JMS Grouping is conceptually similar to Infinispan one and gives us the guarantee that "same group" events **are consumed by the same thread**, thus guaranteeing message ordering (per group)
- Event Source **must be** a JMS server like ActiveMQ and Events **must contain** *JMSXGroupId* metadata

JMS Grouping

- Wouldn't be nice if ActiveMQ could use the **same** Infinispan grouping algorithm so to consume messages directly on the "right" node?
- planned feature for HACEP 1.1

Optional Reordering

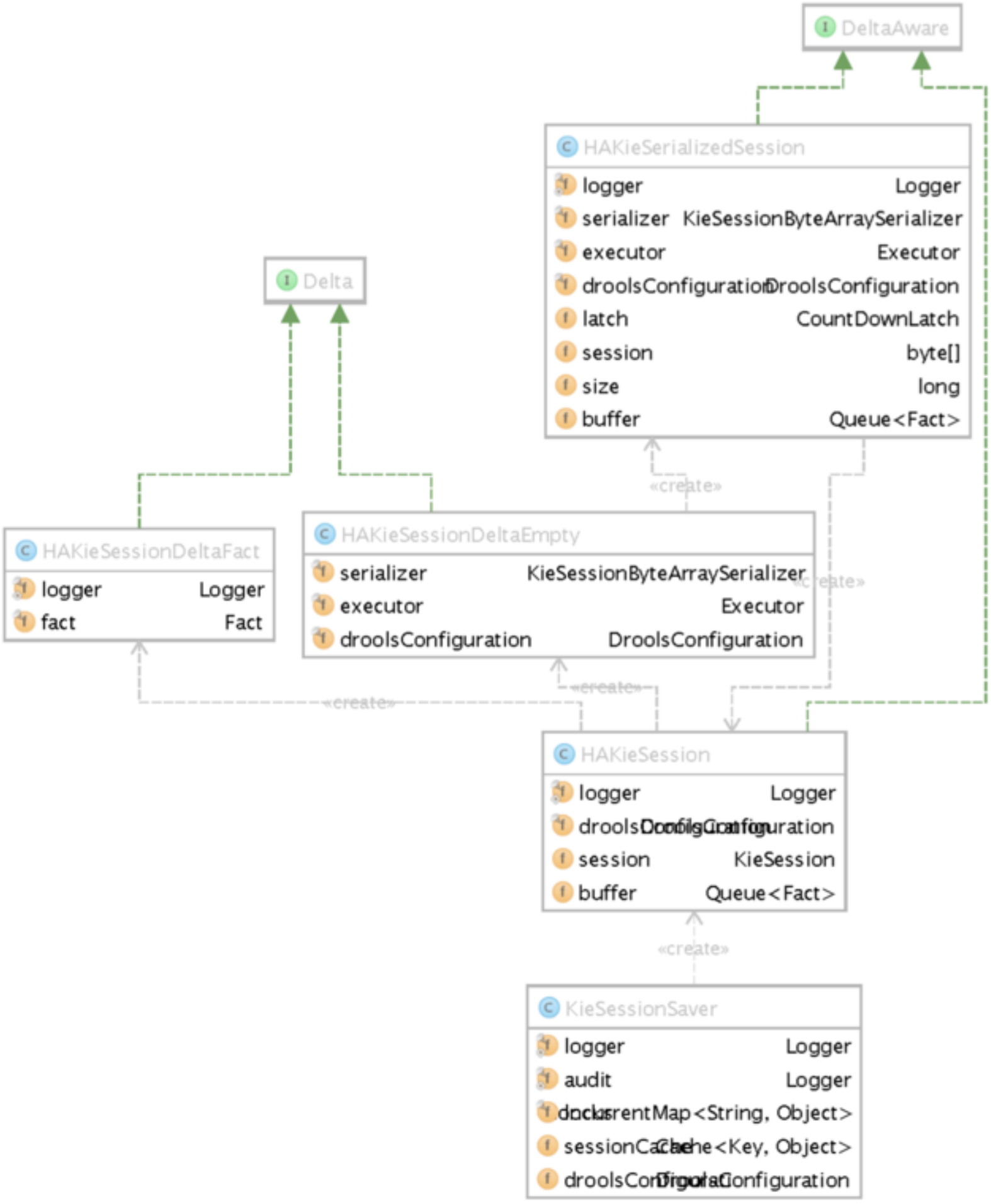
- If **JMS grouping** isn't an option
- HACEP can internally reorder events on the nodes
- Ordering based on a **configurable** Event field
- Could introduce **some latencies** due to buffering and gaps in events

HACEP Internals

DroolsInfinispan

HAKieSession

- **HAKieSession**s are special objects that we save in Infinispan instead of plain Drools KieSessions.
- **HAKieSession** contains a **KieSession** and a **Buffer** of events
- The **Buffer** avoids to load/save the whole BRMS session each time


DeltaAware
HAKieSerializedSession
logger Logger
serializer KieSessionByteArraySerializer
executor Executor
droolsConfiguration DroolsConfiguration
latch CountDownLatch
session byte[]
size long
buffer Queue<Fact>
Delta
«create»
HAKieSessionDeltaFact
logger Logger
fact Fact
HAKieSessionDeltaEmpty
serializer KieSessionByteArraySerializer
executor Executor
droolsConfiguration DroolsConfiguration
«create»
«create»
«create»
HAKieSession
logger Logger
session KieSession
buffer Queue<Fact>
«create»
KieSessionSaver
logger Logger
audit Logger
sessionCache Cache<Key, Object>
droolsConfiguration DroolsConfiguration

Buffers

- Sessions are **not really** loaded and saved in the grid each time an event is received
  - **Event Sourcing** at work
- a **Buffer** of ordered events is saved instead of the whole session, unless the buffer grows more than a configurable limit (Event Sourcing)
- **Snapshots + Buffer** will recreate the state in case of failures
- the buffer uses Infinispan **DeltaAware** apis to minimize network traffic: only the event itself is transmitted to Infinispan

Serialization

- In one of the first releases of the code we found that **serialization/deserialization** of sessions was the most time consuming task
- buffer is an optimization, **but** you can't make it grow indefinitely (though it can be in the range of 1000s thanks to **DeltaAware**)
- being everything local by definition, we are able to **get** pure BRMS sessions from JDG. No session serialization happens, unless you are saving the whole session

Deserialization

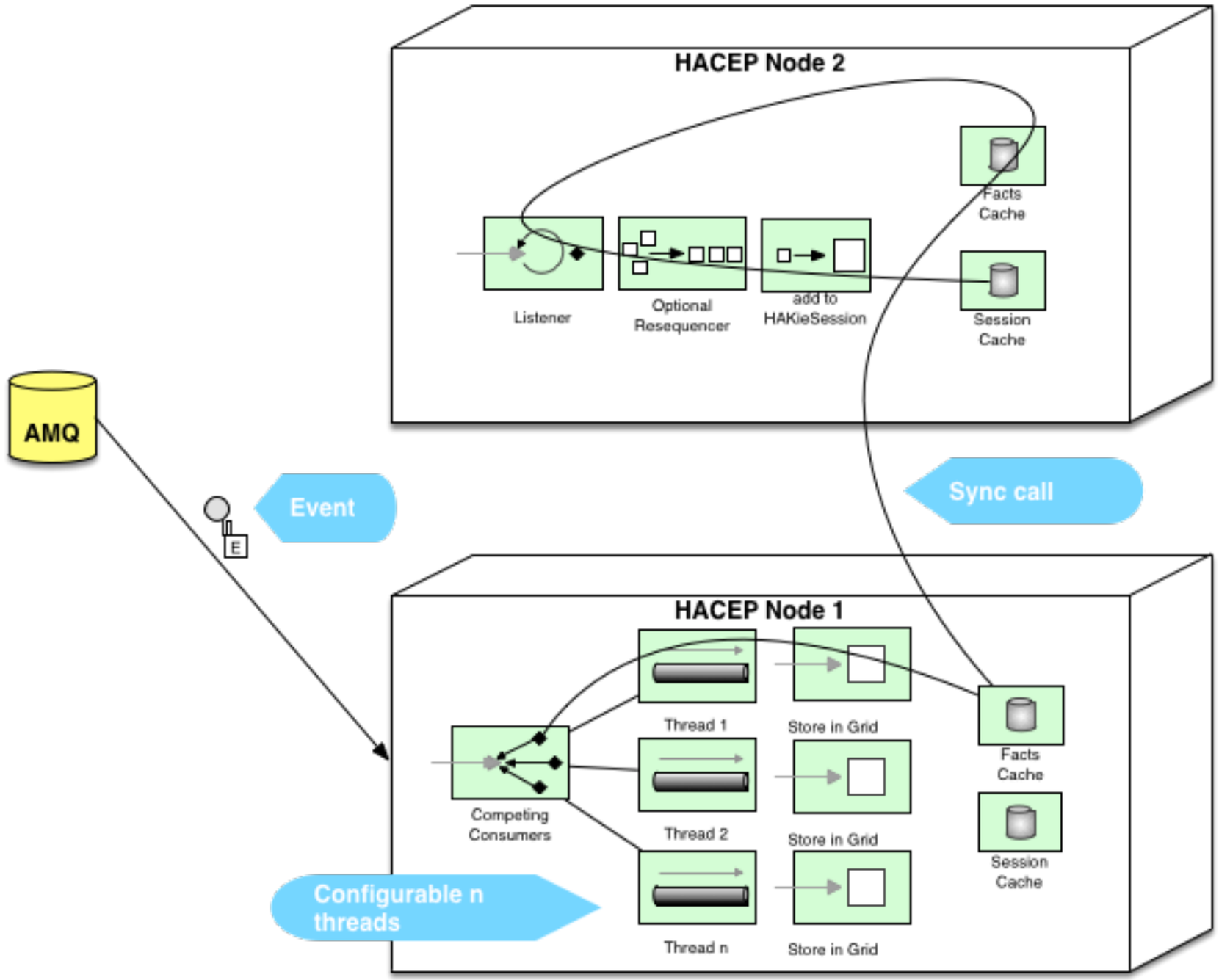
- When replica nodes receive a serialized Object, it's automatically deserialized
  - But it's a **wrapper**, so HACEP avoids the costly operation of deserializing a whole BRMS session
- Sessions are deserialized in replica nodes **only in case** of a failure
  - *if these events are rare, as it should be, no serialization/ deserialization ever happens during the normal flow*

Snapshots

- Sessions **snapshots** are very useful to avoid to replay the whole ever growing buffer in case of failures
- Every node apply **its** buffer to **its** previously stored session, asynchronously
  - at the moment there is only a configurable *size-based policy*
  - more policies are planned
- Again, no big serialized sessions **ever travels on the network**, neither for safe points

Transactions

- Every Hacep node has just a configurable bunch of **ActiveMQ consumers** (threads) serving all its sessions
- Consumers use ActiveMQ **grouping**
- Every consumer is completely **synchronous**
  - Infinispan **notifications** are **synchronous** too
- Messages are consumed **in order** (per group) and won't leave their queue if something goes wrong


HACEP Node 2
Facts Cache
Session Cache
Listener
Optional Resequencer
add to HAKieSession
AMQ
Event
E
Sync call
HACEP Node 1
Thread 1
Store in Grid
Facts Cache
Competing Consumers
Thread 2
Store in Grid
Session Cache
Configurable n threads
Thread n
Store in Grid

Asynchronous Snapshots

- Session snapshots are **asynchronous** operations
- **asynchronous** snapshots enables also a degenerate architecture with no grouping at all, that could be useful for HA (no scaling out) if you can't group sessions
- it basically becomes a simple **active/passive** scenario with 2 nodes

Idempotency

- Different HA architectures must find a solution to replaying events, typically using an **idempotent repository**
- we don't usually need it, because we **always** control the moment when the events need to be replayed
- in case of failures, events in the **Buffer** are replayed and automatically discarded using a special **ReplayChannel**

Idempotent channels

- **Idempotent channels,** due in HACEP 1.1, are useful in cases in which a rule could fire **many** actions on different external systems
- If you need to replay only some of them (let's say a node crashes in the middle of a multi-action) you need to know which actions have been already executed and which haven't
- **Idempotent channels** will be implemented using Infinispan as an **idempotent repository** for command actions

DroolsInfinispan

Customers and Use Cases

redhat.

Lottomatica/IGT

- Not public at the moment (but **can be** informally shared)
- **Gaming** sector
  - Online gaming rewards system
  - part of a bigger **gaming platform** which is sold worldwide

DroolsInfinispan

HACEP Roadmap

redhat.

Planned features

(HACEP 1.1)

- Make ActiveMQ grouping **aware of Infinispan Topology** and therefore plugin Infinispan Consistent Hashing in ActiveMQ
  - with this design even Infinispan notifications will be local in the vast majority of cases
- **Idempotent** replay channels (HACEP 1.1)

Planned features

(HACEP 1.1)

- Pluggable **Policies** to decide when to save the whole session and empty the buffer, for example:
  - cron policies (i.e. when you know the system is idle, or once a day/hour, etc.)
  - only when the rule fire, never
  - Policy scripting (Javascript)

Planned features

(HACEP 1.2)

- Scaling out of **unpartitionable Drools sessions** using **"map/reduce"** like Infinispan collections API
- Migrate from DeltaAware to new **Command-based functional** approach when we'll support it (at the moment it's just in community bits)

HACEP Roadmap

- **HACEP 1.0**
  - September 2016
- **HACEP 1.1**
  - Early 2017
- **HACEP 1.2**
  - TBD

Conclusions

Conclusions

- **HACEP** can easily **scale** horizontally, from 2 nodes to 100s of nodes if needed, even dynamically at runtime
- **HACEP** is **inherently** HA: the minimal HA deployment needs just 2 nodes
- **HACEP** is open to contributions:
  - https://github.com/redhat-italy/hacep
  - http://redhat-italy.github.io/hacep

THANKS!