In-Memory Computing SUMMIT | VIRTUAL EVENT 2020

Patterns of Domain-Driven Design with In-Memory Data Grids

Randy Stafford, Senior Manager
Oracle
October 29, 2020

ORACLE

Hello! (Who’s This Effing Guy?)

- Oracle Coherence Product Manager
- Former Coherence Architect-at-Large
- Former member of Oracle A-Team

- Former Chief Architect of IQNavigator
- Rally Software Technical Advisory Board

- Contributor to architecture literature
- Frequent conference speaker
- An old Smalltalker at heart

Coherence CE

ORACLE

IQNavigator

RALLY SOFTWARE

Patterns of Enterprise Application Architecture

Martin Fowler

97 Things Every Software Architect Should Know

Collective Wisdom from the Experts

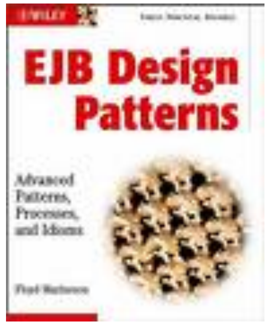
O'REILLY

Collective Wisdom from the Experts

97 Things Every Programmer Should Know

O'REILLY

Edited by Kevlin Henney


EJB Design Patterns

Advanced Patterns, Processes, and Idioms

Agenda

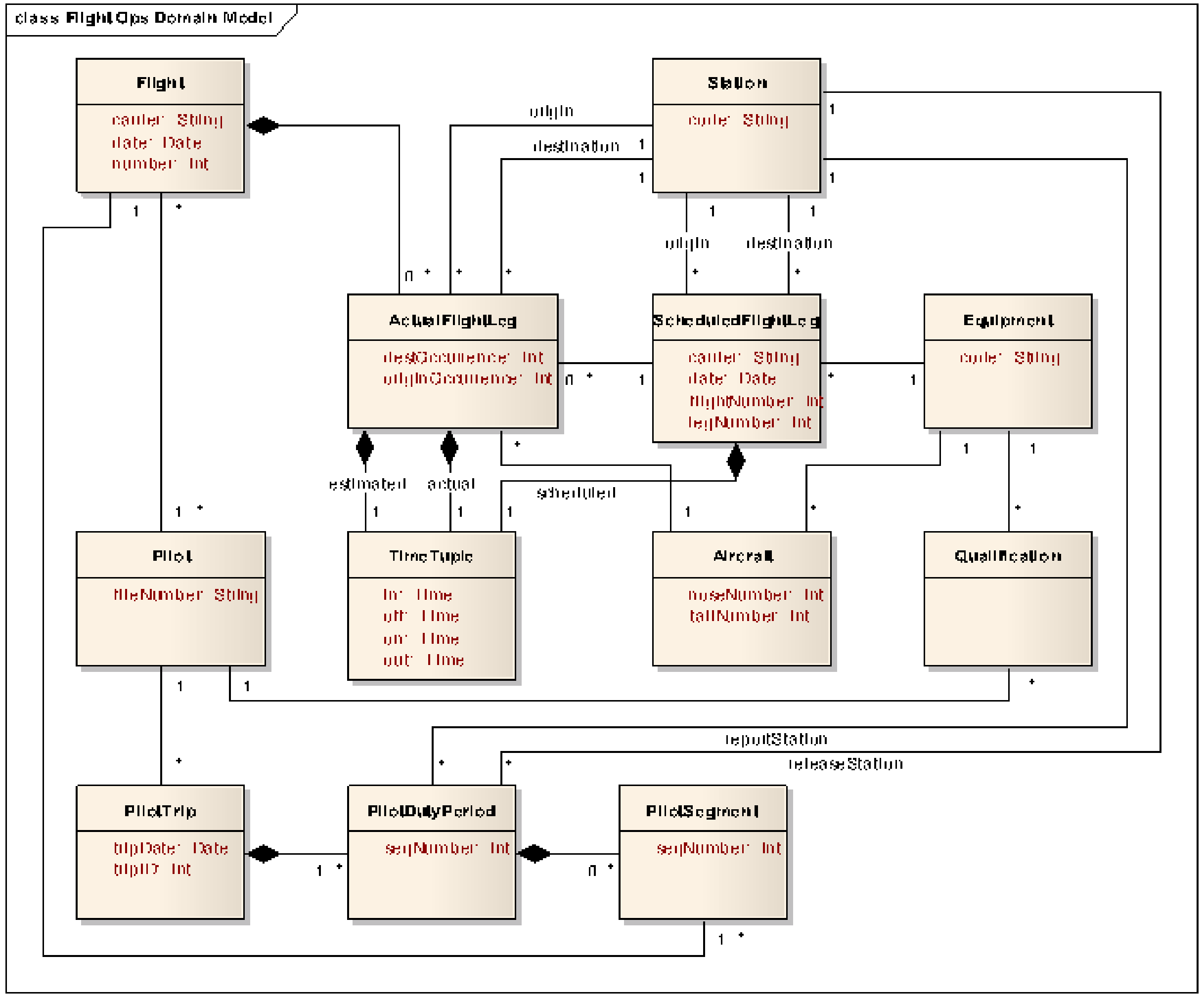
- Domain-Driven Design (DDD)
- In-Memory Data Grids (IMDGs)
- Patterns of DDD with IMDGs
- Cool new stuff
- Summary, Q&A

Domain-Driven Design

- Architectural style at core of OO tradition
- Featuring object model of problem domain
- The architectural style that object persistence technologies have always been designed to support
- State representation changed over time:
    - OOPLs -> objects
    - SOA -> XML
    - HTML5, NoSQL, µservices -> JSON
    - (but where is behavior implemented?)

DDD per Evans: What are the Concepts?

- Domain objects (not DTOs)
- Domain models
- Entities, distinguished by identifier
- Value objects, distinguished by state
- Aggregates
- Aggregate roots
- Relationships
  - Association
  - Composition
- Repositories (not DAOs)



In-Memory Computing SUMMIT VIRTUAL EVENT 2020

DDD Application Runtime Characteristics

- Historically DDD engenders large, highly inter-connected object graphs
  - Domain objects reference each other (by pointer) through fields
  - Collection-typed fields may accumulate many elements over app lifetime
- Object graphs hinder object movement between processes
  - e.g. between remote client and service (hence DTOs)
  - Also between middle tier and persistent store
  - Also between clustered cache servers!
- Application transactions typically involve many Entities or even Aggregates
- May need transaction isolation in domain layer
- Different persistence technologies solve these problems differently

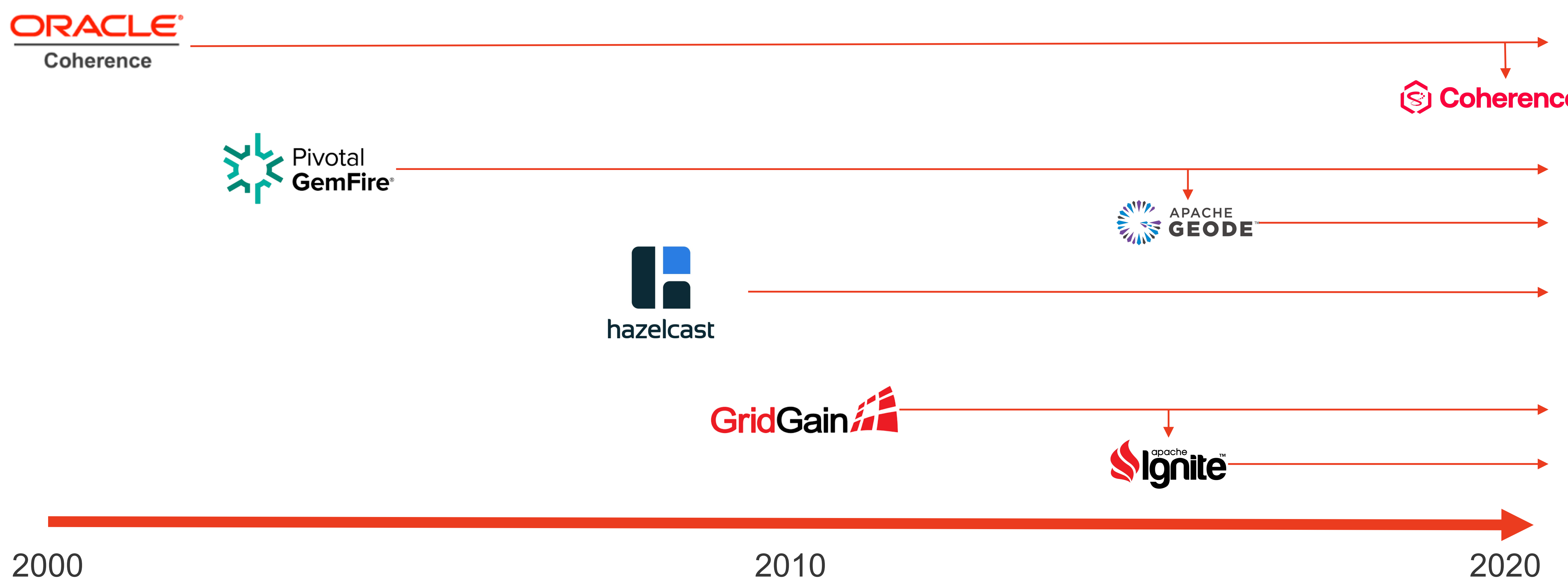
Agenda

- Domain-Driven Design (DDD)
- In-Memory Data Grids (IMDGs)
- Patterns of DDD with IMDGs
- Cool new stuff
- Summary, Q&A

In-Memory Data Grid

- Clustered data management and grid computing software
- Intended to improve performance and scalability of enterprise applications
- Implements key-value (or document) data model; Map interface
- Distinguished from distributed caching platform by more powerful features:
  - Persistence and system-of-record reliability
  - Querying, aggregation, in-place grid computing, transaction support
  - Eventing and messaging, multi-site data federation, change data capture
- Distinction from NoSQL is fuzzy; IMDGs are NoSQL databases+

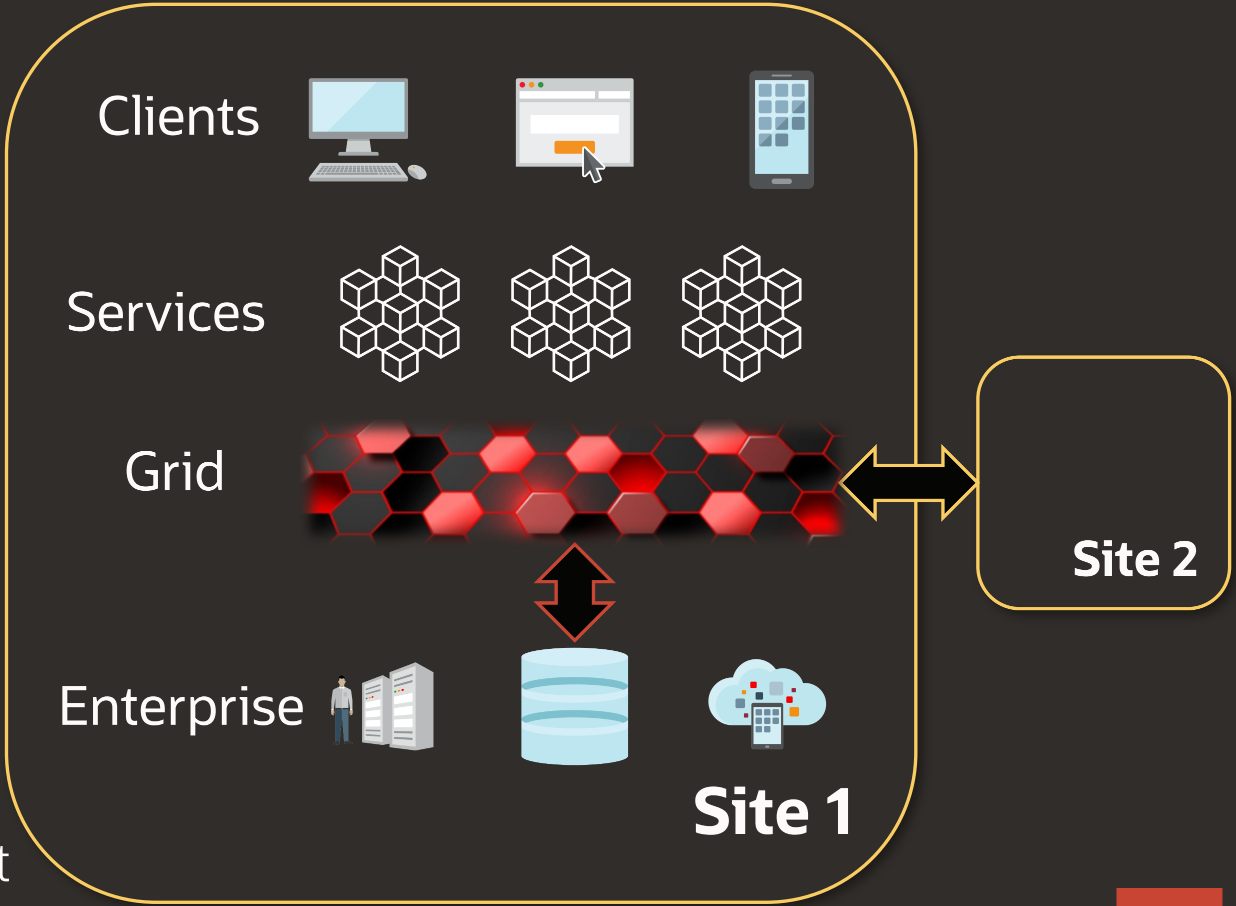
IMDG History



Oracle Coherence Feature Summary

Market-Leading Feature Richness

- Fast key-value store with disk persistence
- Fault-tolerant automatic sharding
- Polyglot and REST client interfaces
- Querying, transactions, eventing
- In-place distributed processing
- HotCache: refresh from database
- Multi-site data federation
- Scalable durable messaging
- Docker, Kubernetes, OpenTracing support



Coherence Community Edition
Launched June 2020

- A free and open-source edition of Coherence
- The core of commercial Enterprise and Grid Editions (EE and GE)
- Hosted on GitHub under Universal Permissive License (UPL)
- Artifacts published to Maven Central; Docker images to GitHub

- Entitles subset of EE features; premium features and support require EE or GE licenses

- Interim YY.MM releases give early access to features in upcoming commercial releases
  - 20.06 release included Helidon MP integration, gRPC proxy server and Java client

- Part of platform for cloud-native microservice apps with Helidon, GraalVM, Verrazzano
- See https://coherence.community, https://github.com/oracle/coherence

Agenda

- Domain-Driven Design (DDD)
- In-Memory Data Grids (IMDGs)
- Patterns of DDD with IMDGs
  - Mapping models to maps
  - Relationships
  - Transactions
  - Domain model caching use cases
- Cool new stuff
- Summary, Q&A

A Question as old as DDD and IMDGs

DDD + No SQL
SYSTEMS SHOULD BE SMALL
OBEY THE TOMATO
REST, DDD & CONTEXT MAPPING
Disposable systems
Grid/Fabric + Domain Events
DDD + No SQL
CAN YOU DO DDD WITH STATE & BEHAVIOR SEPARATE?
AppSec
Business processes
Rules Engine - BPM - BPEL
Map/Reduce + DDD
Datomic
Push DDD Community Forward
Functional DDD
Transactions + NoSQL Mongo, etc
2012
UXD + DDD
How to bring UXD exp. into DDD

http://abdullin.com/journal/2012/5/20/ddd-summit-2012-summary-dddesign.html

Storing Domain Models in IMDGs

- IMDGs have unique programming model
  - Not like ORM programming model
  - A new tier of architecture
  - A new place for behavior
- Choose a model-to-map mapping pattern:
  - Map Per Entity Type
  - Map Per Aggregate Root Type
  - Map Per Object State
- Implement inter-object references in model
  - Per model-to-map mapping pattern
  - Reference By Pointer
  - Reference By Identifier
- Implement Map keys (Entity identifiers)
- Implement serializability
- Implement Repositories
  - Protected Variation pattern
  - Future impls for different IMDGs/APIs

Patterns of Mapping Models to Maps

Pattern	Pros	Cons
Map Per Entity Type	• Well-known precedent from ORM world • Simplest mapping pattern • Very uniform and predictable	• Navigating object graphs requires repeated cache access • Multi-object atomic transactions become challenging • Query by state required for important state models
Map Per Aggregate Root Type	• Fits well with DDD notion of Aggregate • Efficient data access and mutation • Easy to atomically transaction	• Non-uniform; hard to framework (leads to bespoke code) • App transactions may involve multiple Aggregates • Query by state required for important state models
Map Per Object State (e.g. Orders: new, paid, filled)	• Efficient data access for important state models	• Requires moving entries between maps as state changes • May present atomicity challenges

Multiple-Cardinality Relationships (1:M, M:N)

- Serialize objects on M side with object on 1 side
- Separate caches for M side, 1 side objects
  - M side objects hold identifier of 1 side object
    - Requires queries
  - 1 side object holds collection of M side object identifiers
    - Enables use of getAll()
    - May need collection manipulation without deserialization
- Separate cache for Relationship Objects

Transactions

- Single-entry transactions
  - Requires Named Cache per Aggregate Root Type pattern
  - Assumes only one Aggregate per Application Transaction
  - Enterprise application designs skewed for this?
- Partition-level transactions: unique Coherence feature
  - Allows efficient multi-entry, multi-cache transactions
  - Requires data affinity, single service
- Coherence Transaction Framework
  - Full-blown XA / JTA, with attendant performance characteristics
- This is the hardest problem in DDD with IMDGs

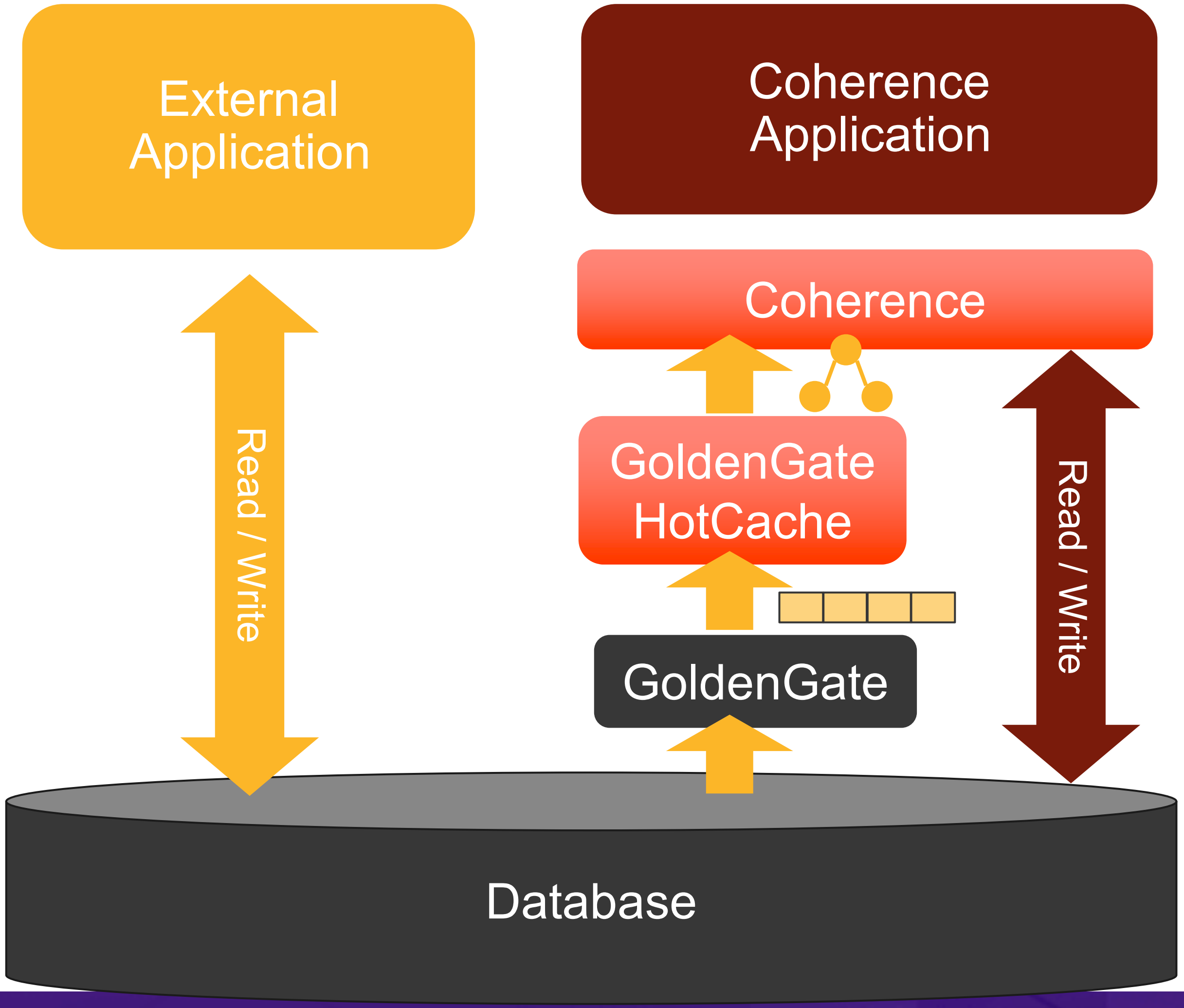
Domain Model Caching Use Cases

Name	Summary
Read Caching	Cache domain objects read from another source, for lower-latency read access, and offloading the source.
Write Buffering	Buffer writes to a data store, to reduce write latency perceived by client, and to avoid exhausting the data store’s write capacity.
Event Processing	Process events affecting the state of stored domain objects, updating their state in a data grid.
Grid Computing	Execute parallel distributed logic algorithms on data in a grid, to minimize execution time or maximize work throughput.
Synchronized Projection	Maintain up-to-date alternative projections of a domain model, as state-mutating events are processed.
Key Mapping	Map secondary keys to primary keys.
Computation Result Caching	Cache results of computations (e.g. Hadoop) for access by live application.

Agenda

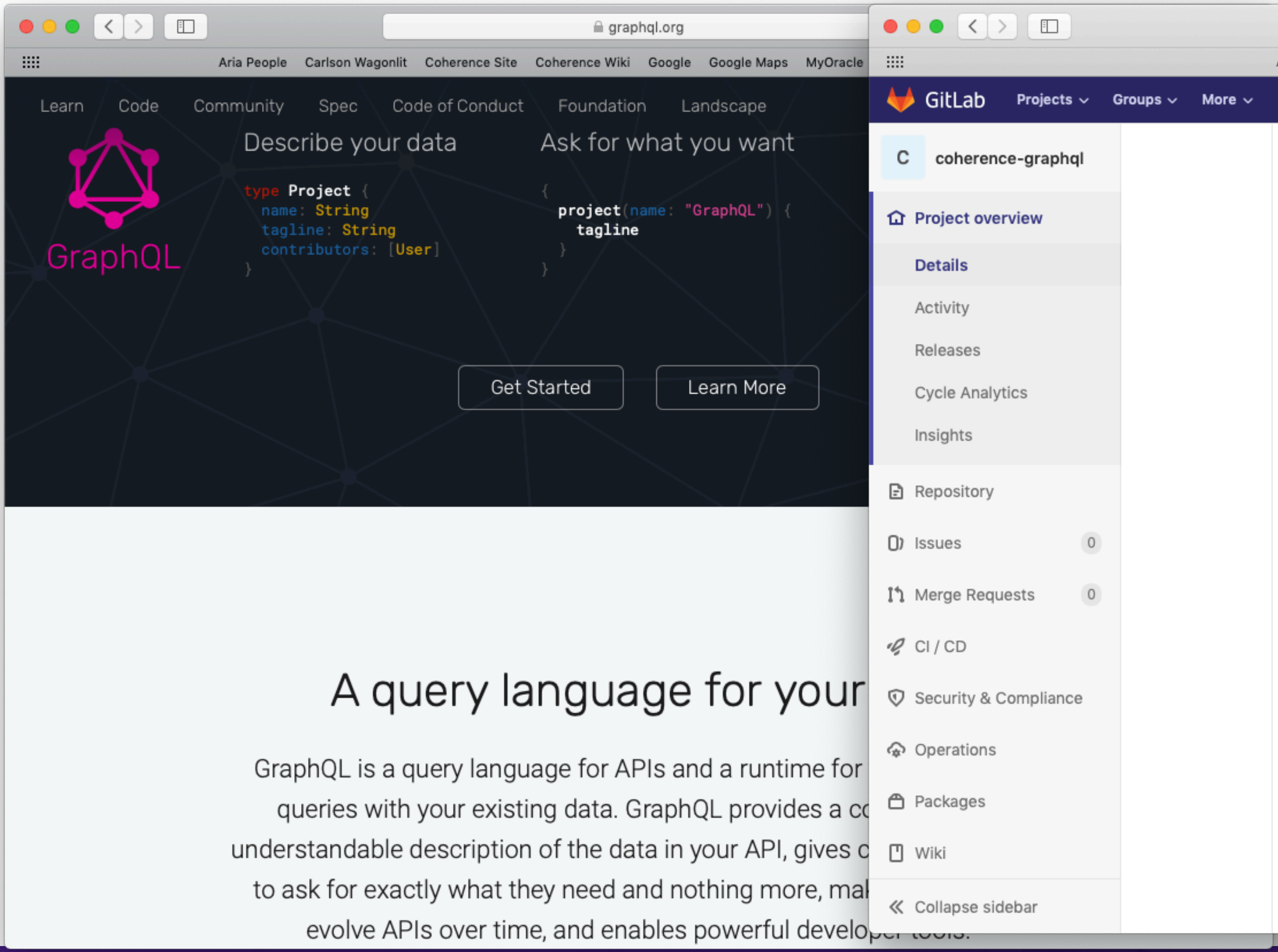
- Domain-Driven Design (DDD)
- In-Memory Data Grids (IMDGs)
- Patterns of DDD with IMDGs
- Cool new stuff
  - HotCache
  - GraphQL
- Summary, Q&A

HotCache: Cache Refresh from DB Txns



- Push DB changes to Coherence
- Via GoldenGate and TopLink JPA
- Tables map to entities, caches
- Event-driven and efficient
- Scale-out tested to 20K writes/sec
- Solves stale cache problem when external apps write to shared DB
- Allows caching to be leveraged in such apps

GraphQL: Object Graph Navigation



Describe your data

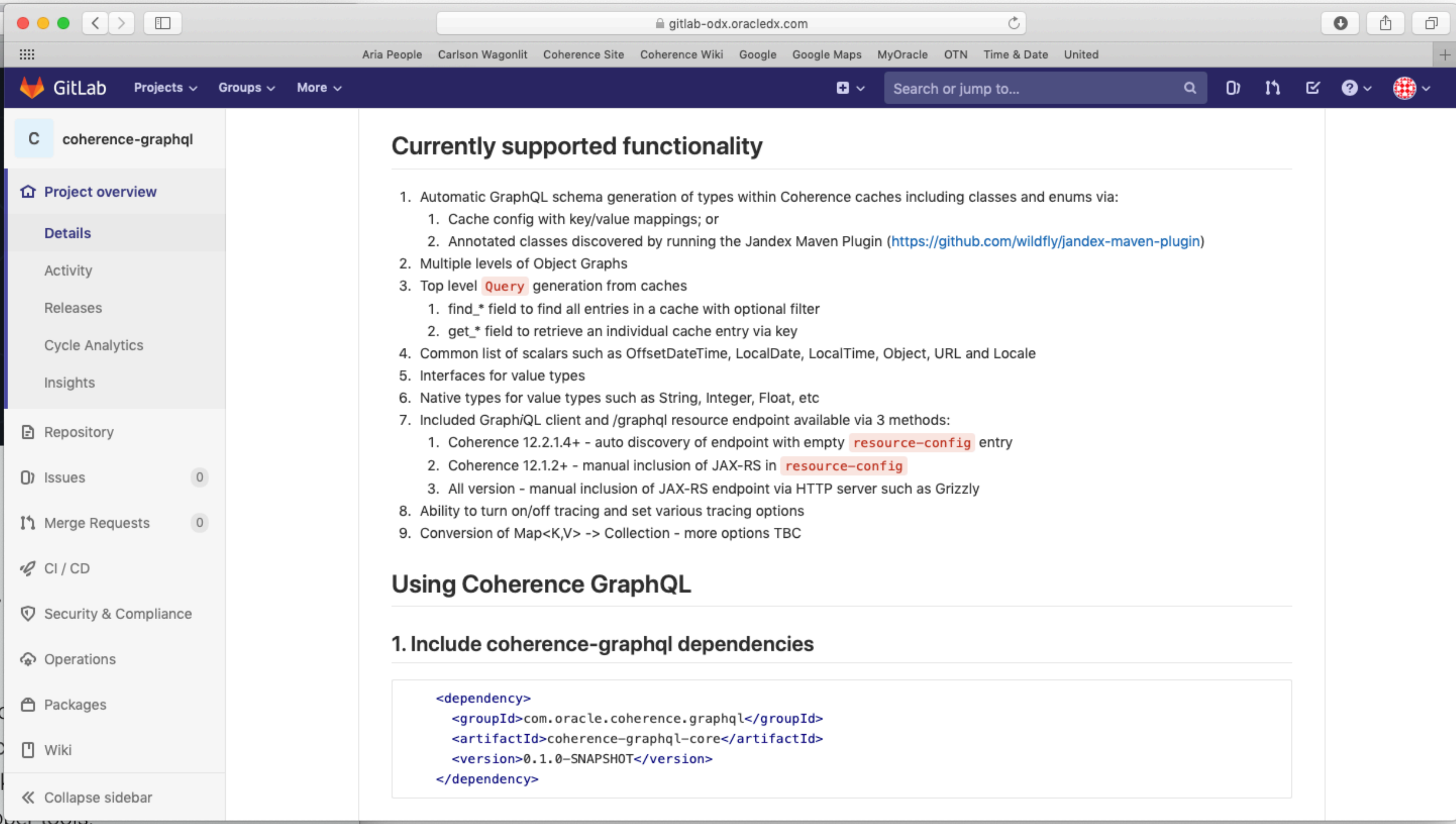
```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Ask for what you want

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

A query language for your

GraphQL is a query language for APIs and a runtime for
queries with your existing data. GraphQL provides a co
understandable description of the data in your API, gives c
to ask for exactly what they need and nothing more, mal
evolve APIs over time, and enables powerful develop



Currently supported functionality

1. Automatic GraphQL schema generation of types within Coherence caches including classes and enums via:
   1. Cache config with key/value mappings; or
   2. Annotated classes discovered by running the Jandex Maven Plugin (https://github.com/wildfly/jandex-maven-plugin)
2. Multiple levels of Object Graphs
3. Top level `Query` generation from caches
   1. find_* field to find all entries in a cache with optional filter
   2. get_* field to retrieve an individual cache entry via key
4. Common list of scalars such as OffsetDateTime, LocalDate, LocalTime, Object, URL and Locale
5. Interfaces for value types
6. Native types for value types such as String, Integer, Float, etc
7. Included GraphiQL client and /graphql resource endpoint available via 3 methods:
   1. Coherence 12.2.1.4+ - auto discovery of endpoint with empty `resource-config` entry
   2. Coherence 12.1.2+ - manual inclusion of JAX-RS in `resource-config`
   3. All version - manual inclusion of JAX-RS endpoint via HTTP server such as Grizzly
8. Ability to turn on/off tracing and set various tracing options
9. Conversion of Map<K,V> -> Collection - more options TBC

Using Coherence GraphQL

1. Include coherence-graphql dependencies

```
<dependency>
  <groupId>com.oracle.coherence.graphql</groupId>
  <artifactId>coherence-graphql-core</artifactId>
  <version>0.1.0-SNAPSHOT</version>
</dependency>
```

Agenda

- Domain-Driven Design (DDD)
- In-Memory Data Grids (IMDGs)
- Patterns of DDD with IMDGs
- Cool new stuff
- Summary, Q&A

In-Memory Computing SUMMIT | VIRTUAL EVENT 2020

Patterns of Domain-Driven Design with In-Memory Data Grids

Randy Stafford, Senior Manager
Oracle
October 29, 2020

ORACLE