ORACLE®

JavaOne™

ORACLE®

JPA In Reverse: Pushing Database Events to Java EE Applications in Real Time

Jean-Philippe Laroche | Owner
Kaféine Consulting Inc.

Randy Stafford | Architect At-Large
Oracle Coherence Product Development

October 27, 2015

JavaOne ORACLE

Session Agenda

1 Context and problem

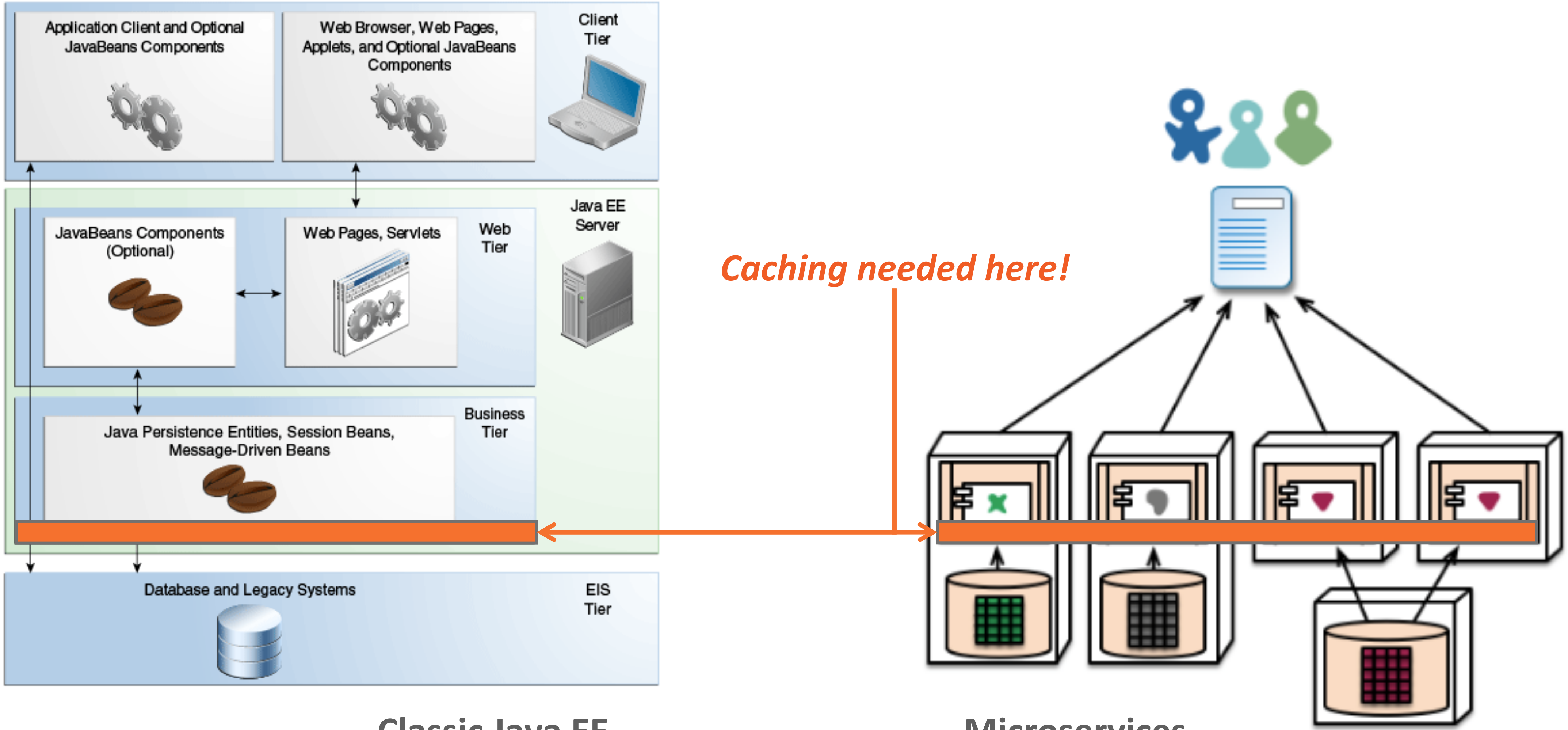
2 Evaluation of known solutions

3 Using database replication technology

4 Consequences and nuances of JPA in reverse

5 Summary, Q&A / discussion

Java Enterprise Applications

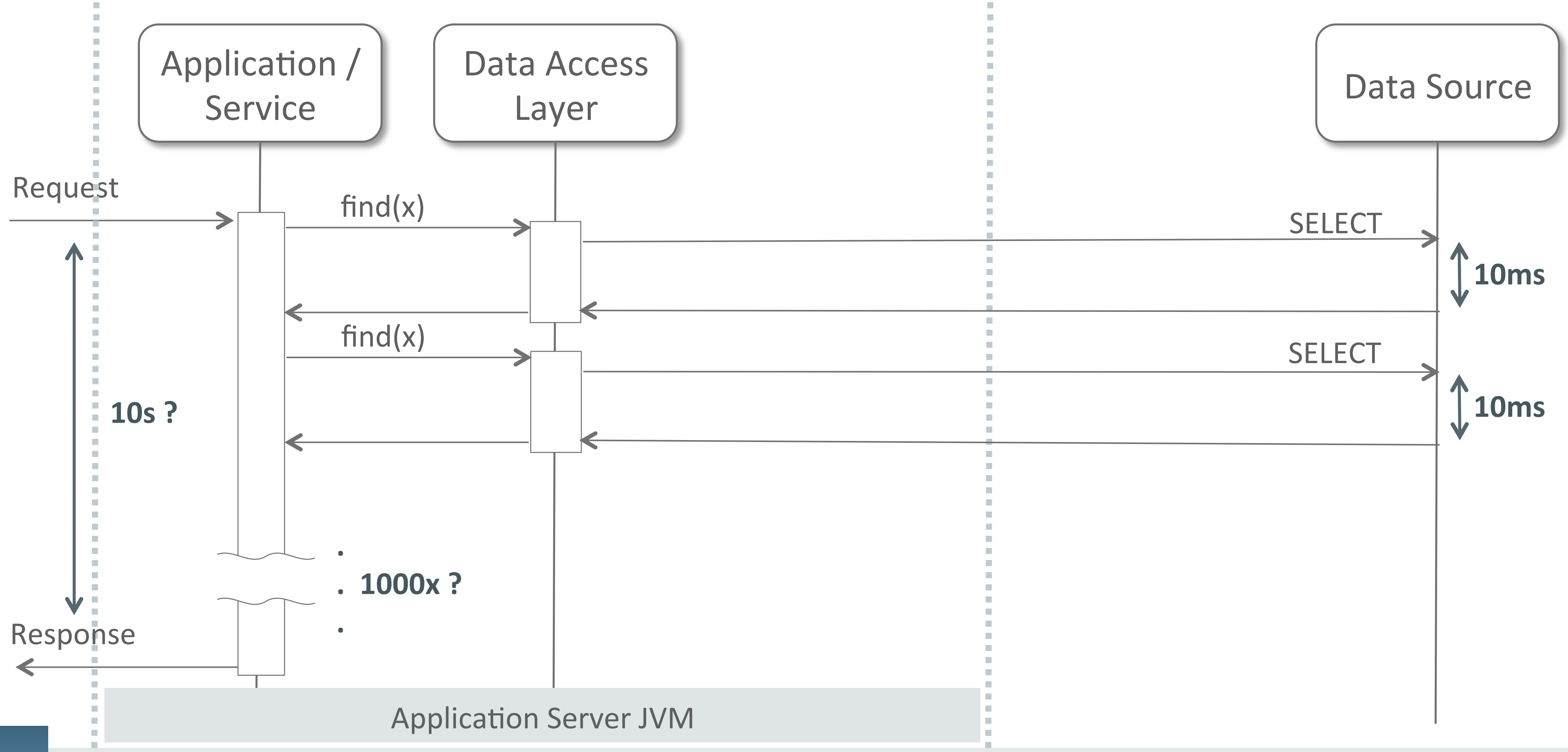

Application Client and Optional JavaBeans Components
Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
JavaBeans Components (Optional)
Web Pages, Servlets
Web Tier
Java EE Server
Java Persistence Entities, Session Beans, Message-Driven Beans
Business Tier
Database and Legacy Systems
EIS Tier

Caching needed here!

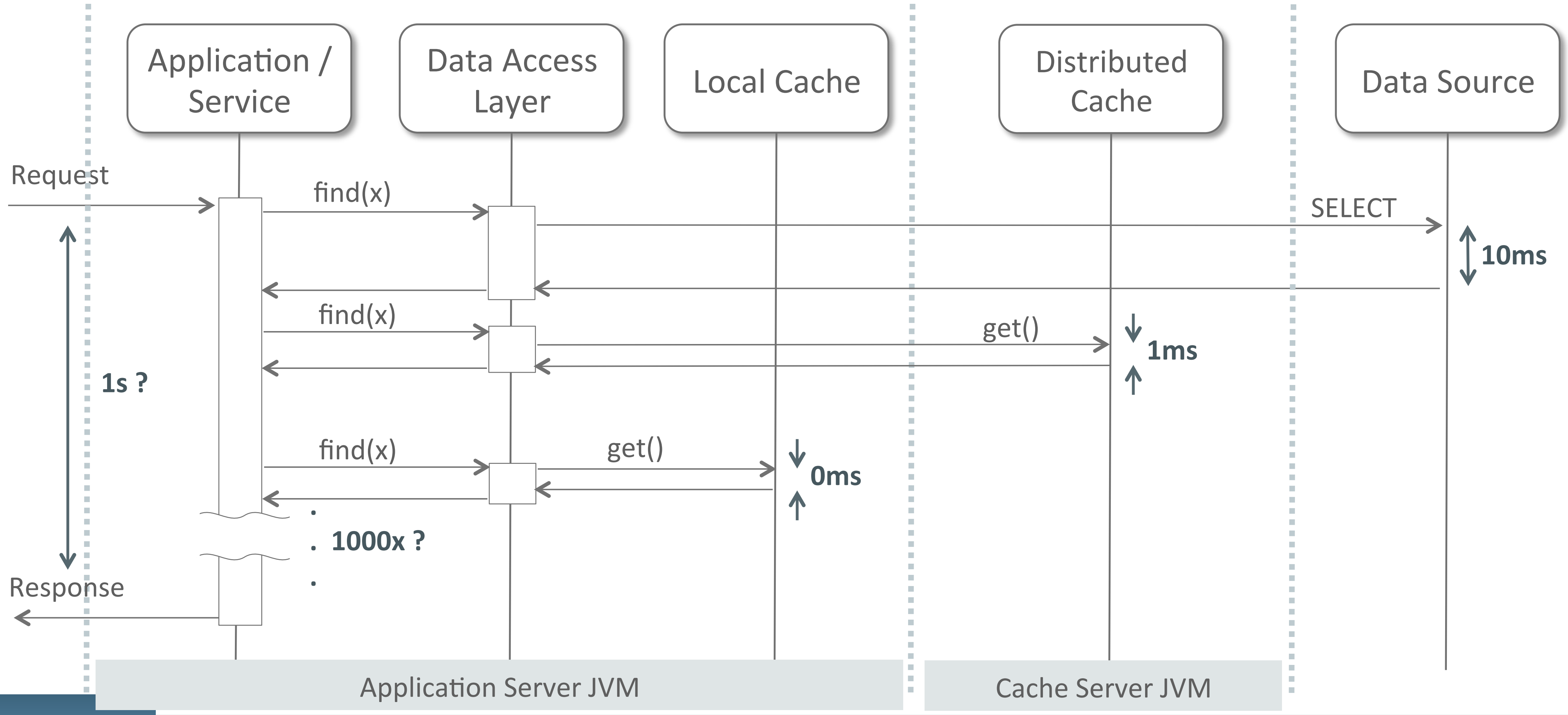
Classic Java EE

Microservices

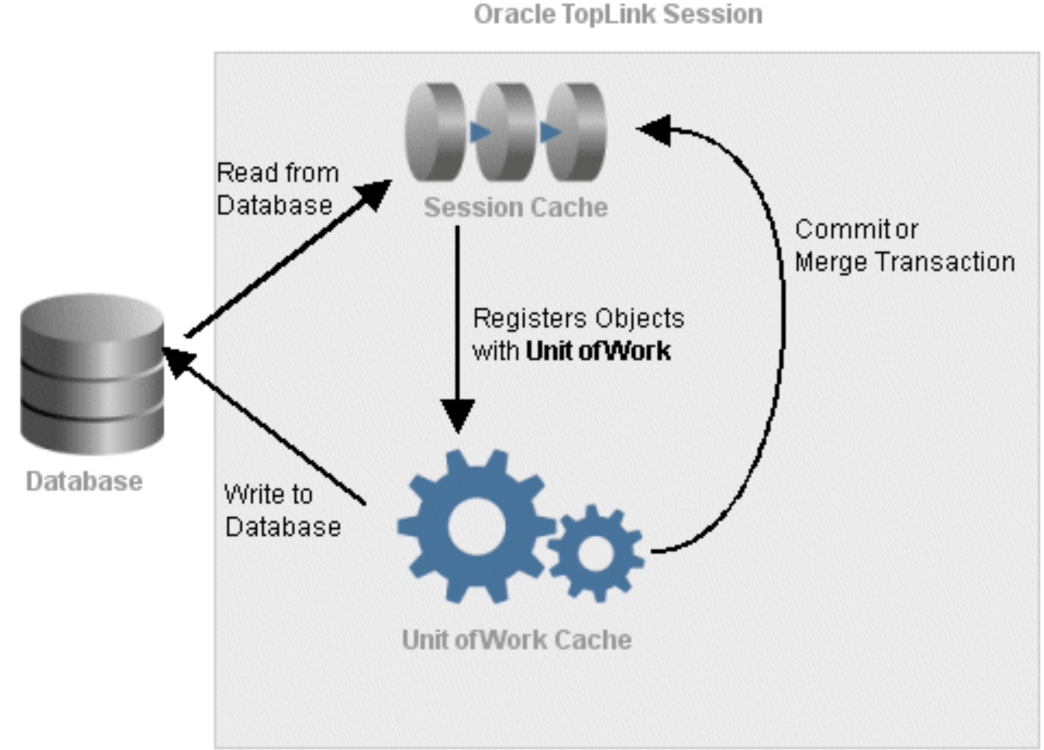
Why Cache? Application Data Access – WITHOUT Caching



Why Cache? Application Data Access – WITH Caching



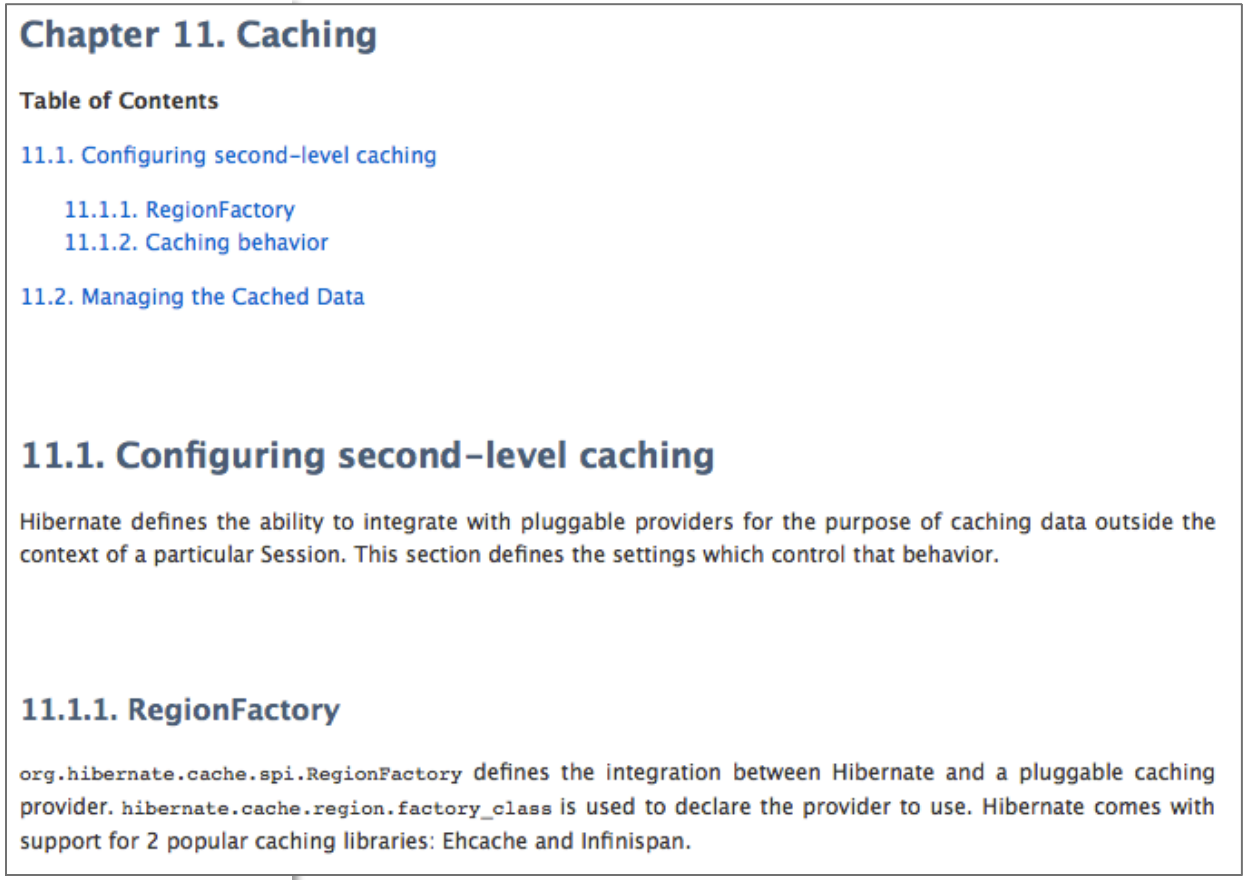
Hence, L2 Caching in Java ORMs – Since 1996



Description of "Figure 8-1 Object Life Cycle and the EclipseLink Caches"

Persistence Unit Cache

The persistence unit cache is a shared cache (L2) that services clients attached to a given persistence unit.



Chapter 11. Caching

Table of Contents

11.1. Configuring second-level caching

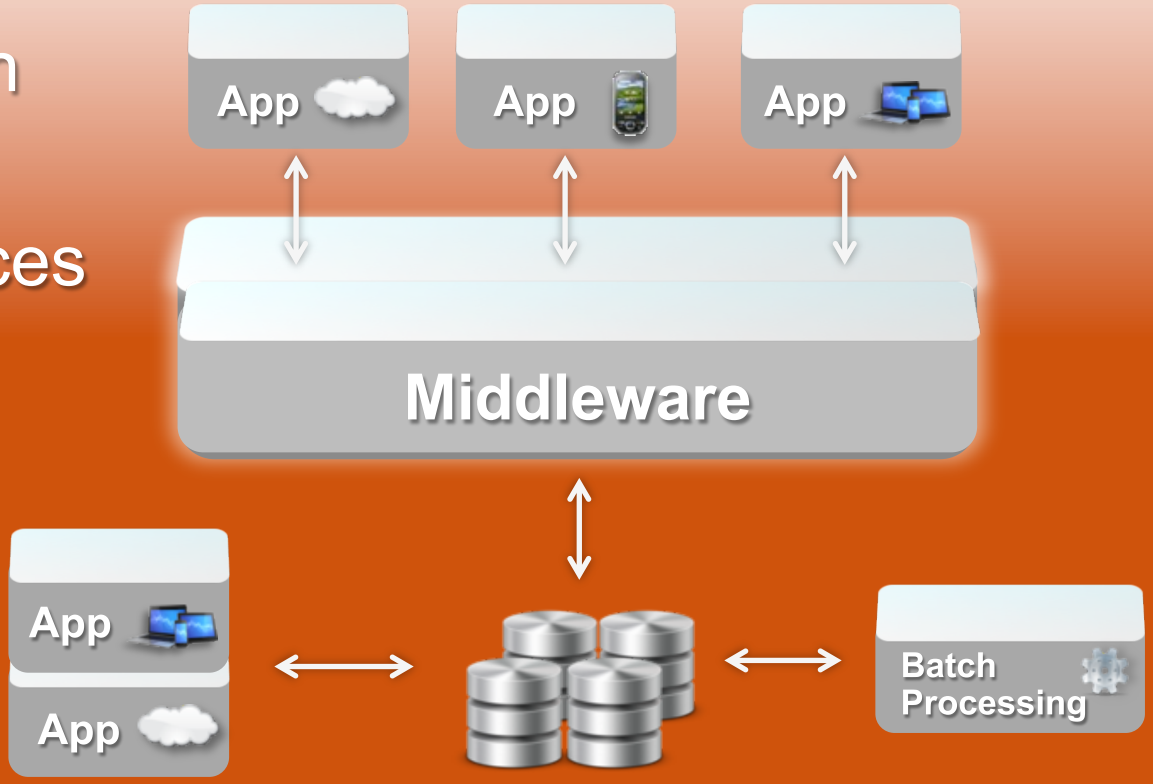
Hibernate defines the ability to integrate with pluggable providers for the purpose of caching data outside the context of a particular Session. This section defines the settings which control that behavior.

11.1.1. RegionFactory

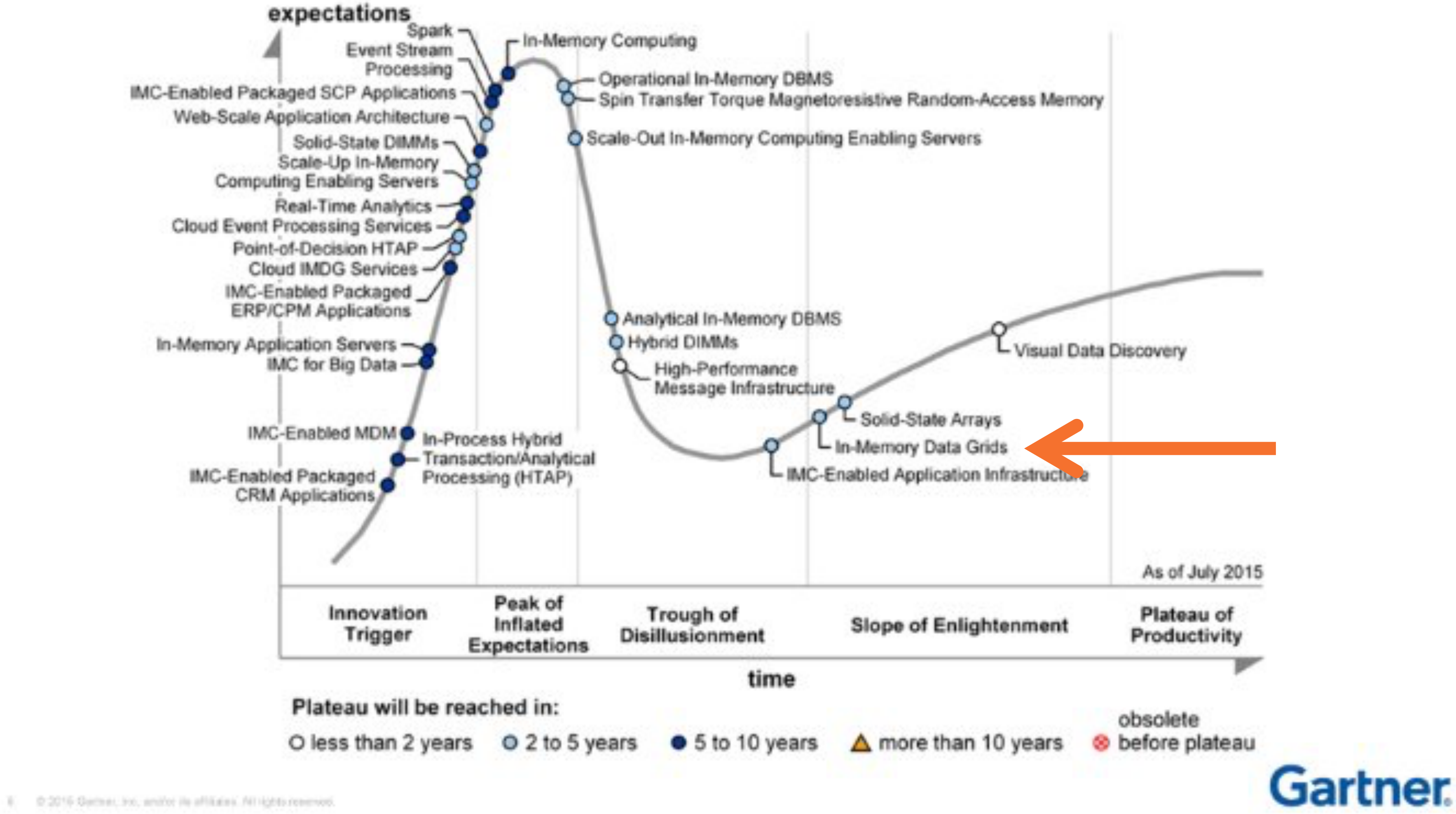
`org.hibernate.cache.spi.RegionFactory` defines the integration between Hibernate and a pluggable caching provider. `hibernate.cache.region.factory_class` is used to declare the provider to use. Hibernate comes with support for 2 popular caching libraries: Ehcache and Infinispan.

Hence, In-Memory Data Grids – Since 2001

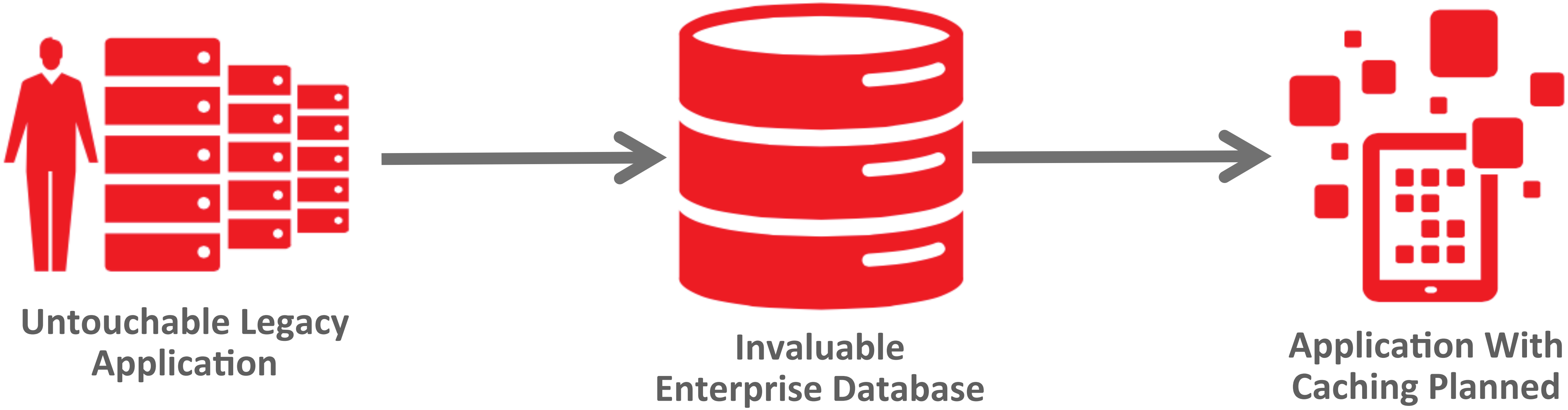
- Scaling applications to support growth
- Offloading / protecting shared resources
- Delivering information in real time



In Fact, IMDGs Have Survived the “Hype Cycle”…



But, A Problem Still Affects The Ability to Cache …


Untouchable Legacy Application
Invaluable Enterprise Database
Application With Caching Planned

How to keep the cache consistent with the database?

Session Agenda

1 Context and problem

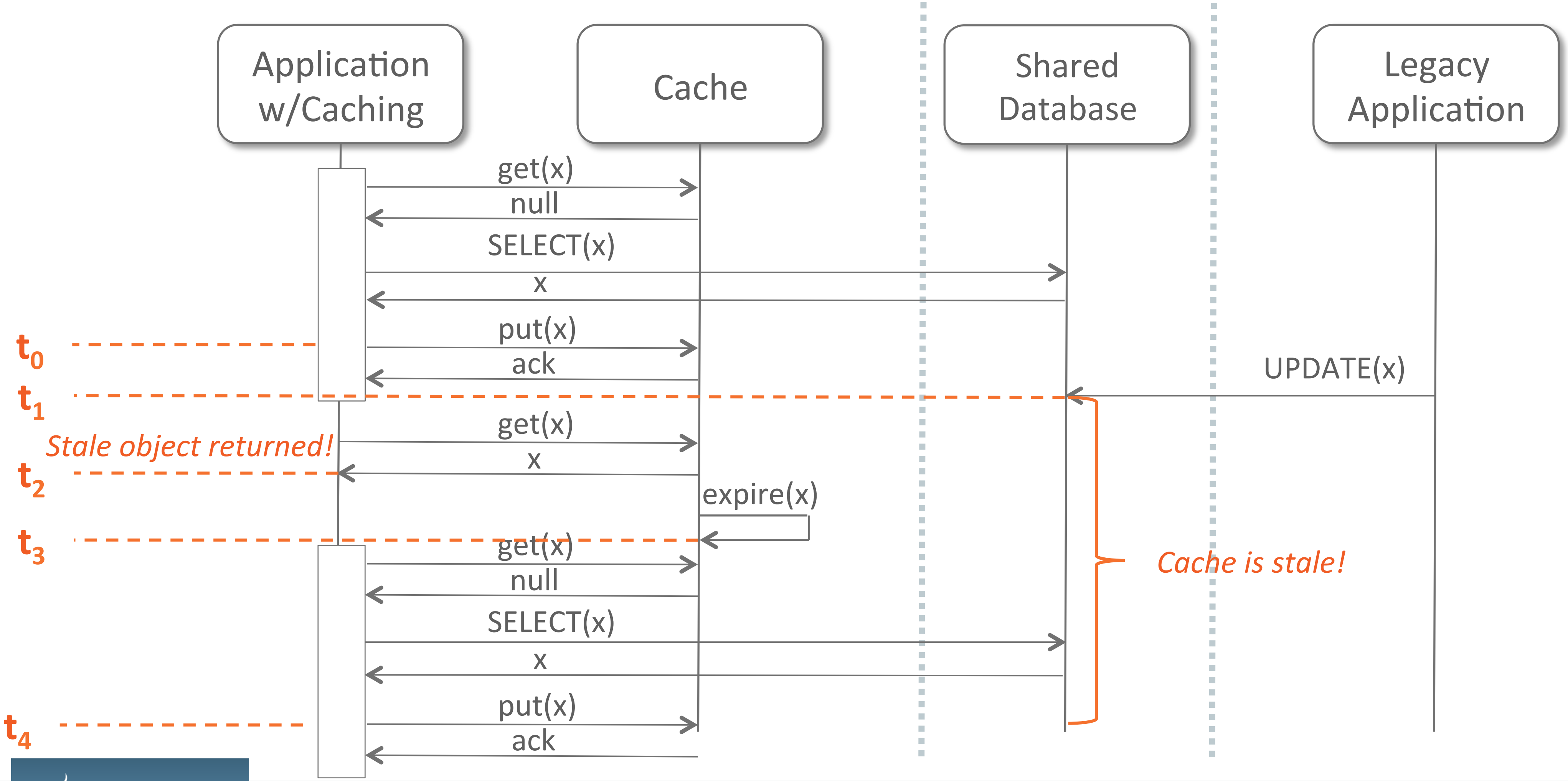
2 Evaluation of known solutions

3 Using database replication technology

4 Consequences and nuances of JPA in reverse

5 Summary, Q&A / discussion

Expiry


Application w/Caching
Cache
Shared Database
Legacy Application
get(x)
null
SELECT(x)
x
put(x)
ack
t_0
UPDATE(x)
t_1
get(x)
Stale object returned!
x
t_2
expire(x)
t_3
get(x)
null
SELECT(x)
x
put(x)
t_4
ack
Cache is stale!

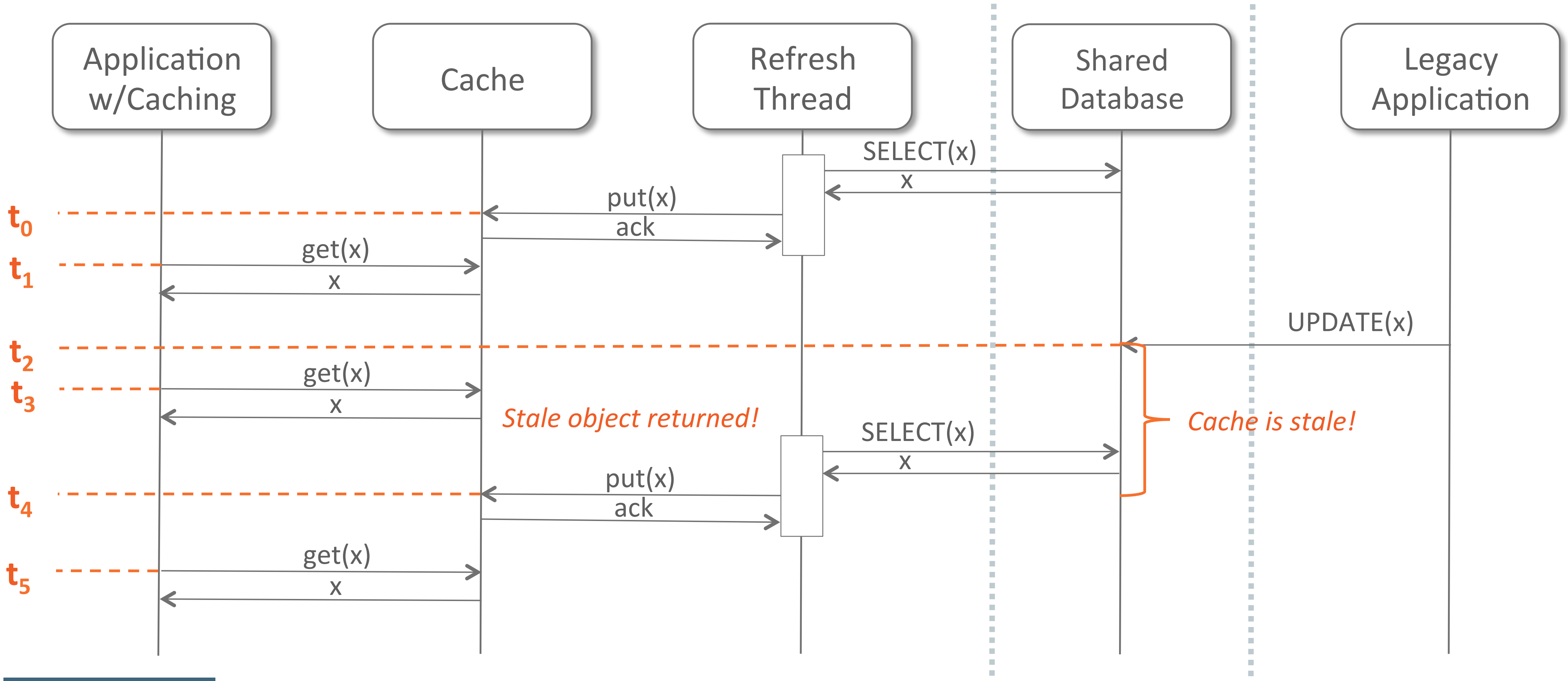
Expiry - Prevalence

IMDGs	ORMs
Generally a standard feature of IMDGs	EclipseLink allows via custom @Cache annotation
Some may have sliding expiry, etc. (reset expiry on get)	Hibernate delegates to second-level cache provider

Expiry - Evaluation

Merits	Demerits
Simplicity	Inconsistency – stale cache values returned to app
Some degree of caching benefit	Shorter expiry => less benefit from caching

Periodic Refresh



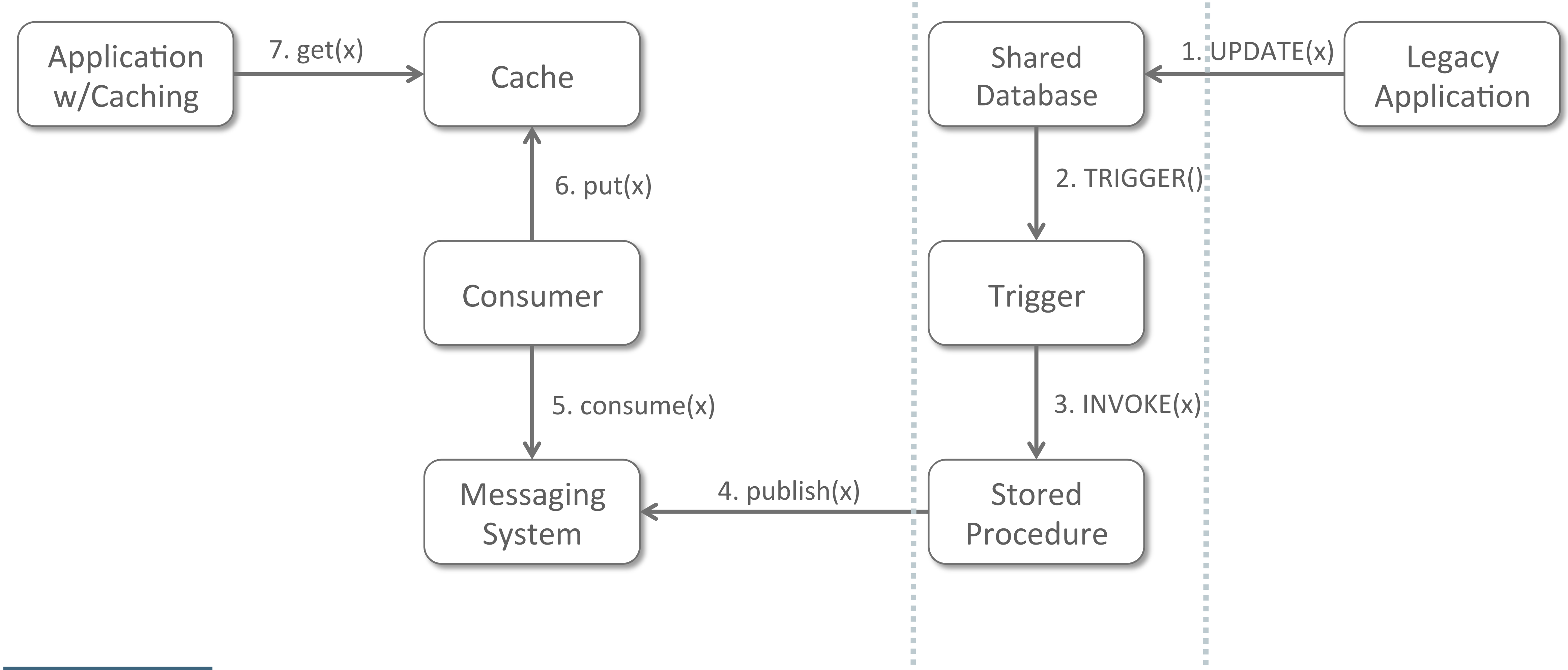
Periodic Refresh - Prevalence

IMDGs	ORMs
Generally not ubiquitously available in IMDGs	Generally not OOTB in ORMs
Some IMDGs have features or integrations that do periodic refresh or variants of it	You could theoretically use this approach, at risk of violating ORM encapsulation / duplicating ORM logic
Have seen custom implementations in the wild	

Periodic Refresh - Evaluation

Merits	Demerits
More benefit from caching than with expiry	Inconsistency – stale cache values returned to app
Short refresh interval => better consistency than expiry	Custom machinery required - refresh thread
Can use “audit tables” for efficiency	Complex to make scalable and highly available

Triggered Messaging



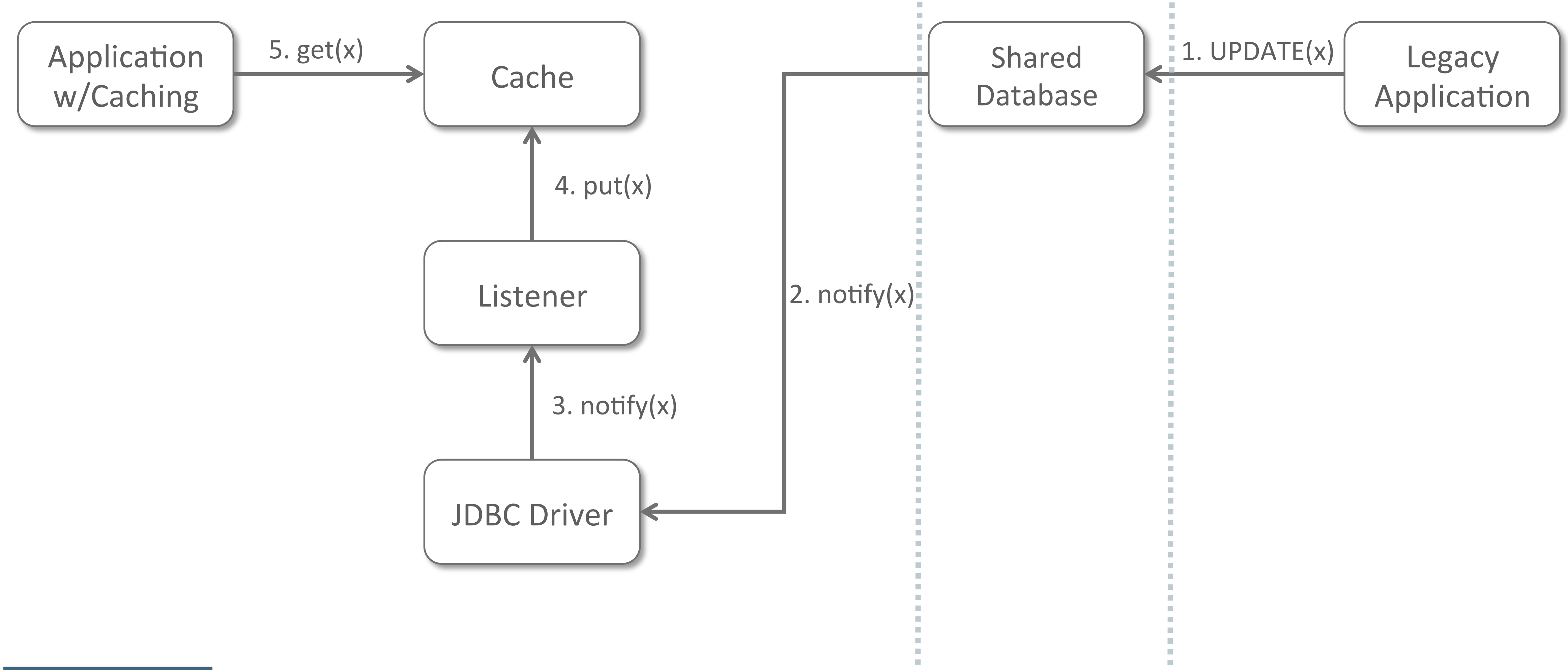
Triggered Messaging - Prevalence

IMDGs	ORMs
Generally not OOTB in IMDGs	Generally not OOTB in ORMs
Requires custom development	You could theoretically use this approach, at risk of violating ORM encapsulation / duplicating ORM logic
Have heard of implementations in the wild	

Triggered Messaging - Evaluation

Merits	Demerits
Event-driven and near real-time	Complexity – many moving parts
Cache stays consistent with database, modulo message delivery latency	Requires messaging system callable from stored procs
Scalability, HA (due to messaging infrastructure)	Custom machinery needed: trigger, producer, consumer
	Requires administering messaging system / destinations

Database Change Notification



Database Change Notification - Prevalence

IMDGs	**ORMs**
Generally not OOTB in IMDGs	OOTB in EclipseLink
Requires custom development	You could theoretically use this approach with others, at risk of violating encapsulation / duplicating logic
Have seen custom implementations in the wild	

Database Change Notification - Evaluation

Merits	**Demerits**
Event-driven, and nearest real-time	Custom machinery – listener and registration
Fewer moving parts than triggered messaging	Listener may need to query database
Cache stays consistent with database, modulo notification latency (lowest of all solution alternatives)	Complex to make scalable and highly available
	Limited to Oracle Database

Session Agenda

1 Context and problem

2 Evaluation of known solutions

3 Using database replication technology

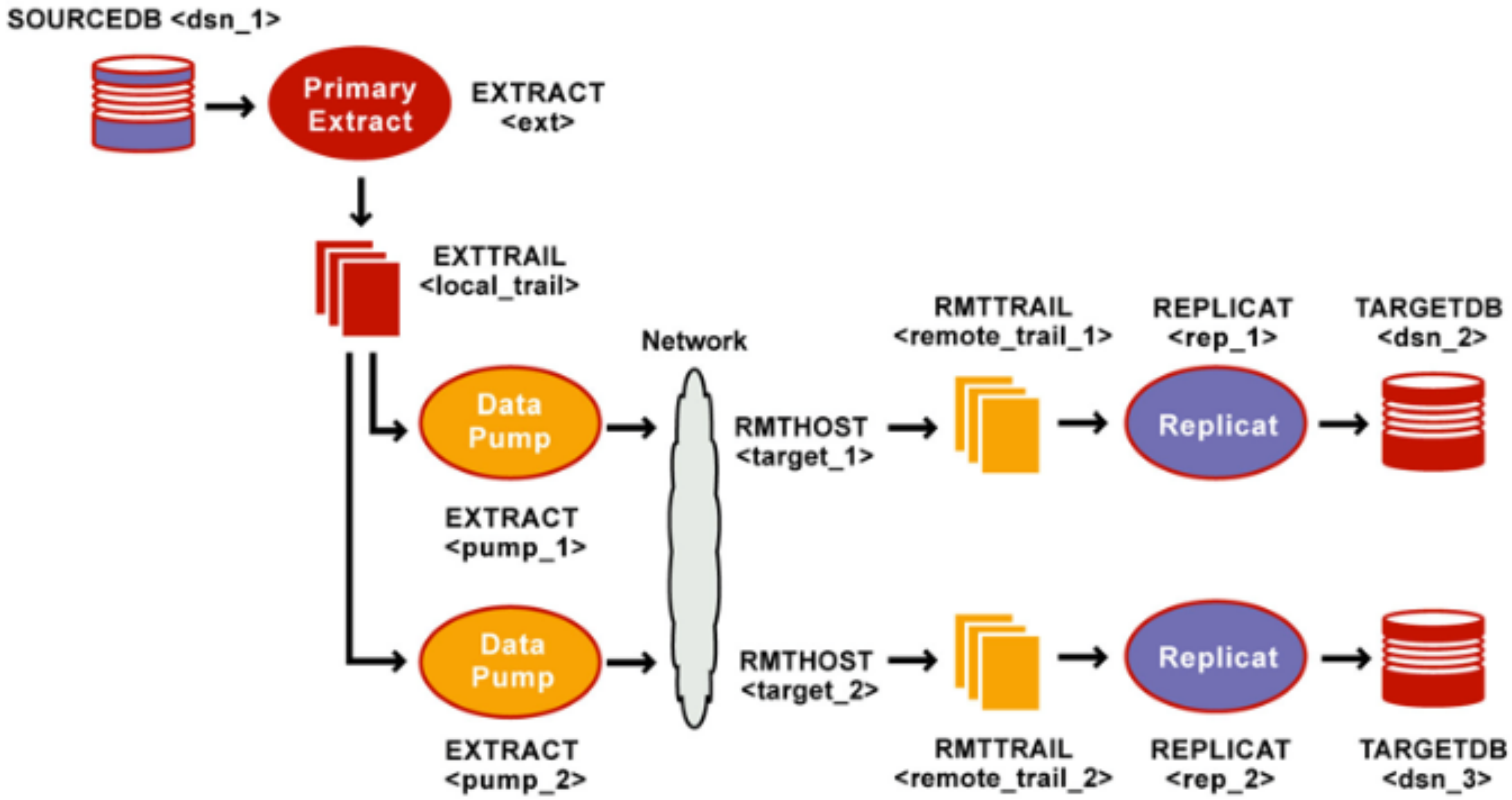
4 Consequences and nuances of JPA in reverse

5 Summary, Q&A / discussion

GoldenGate

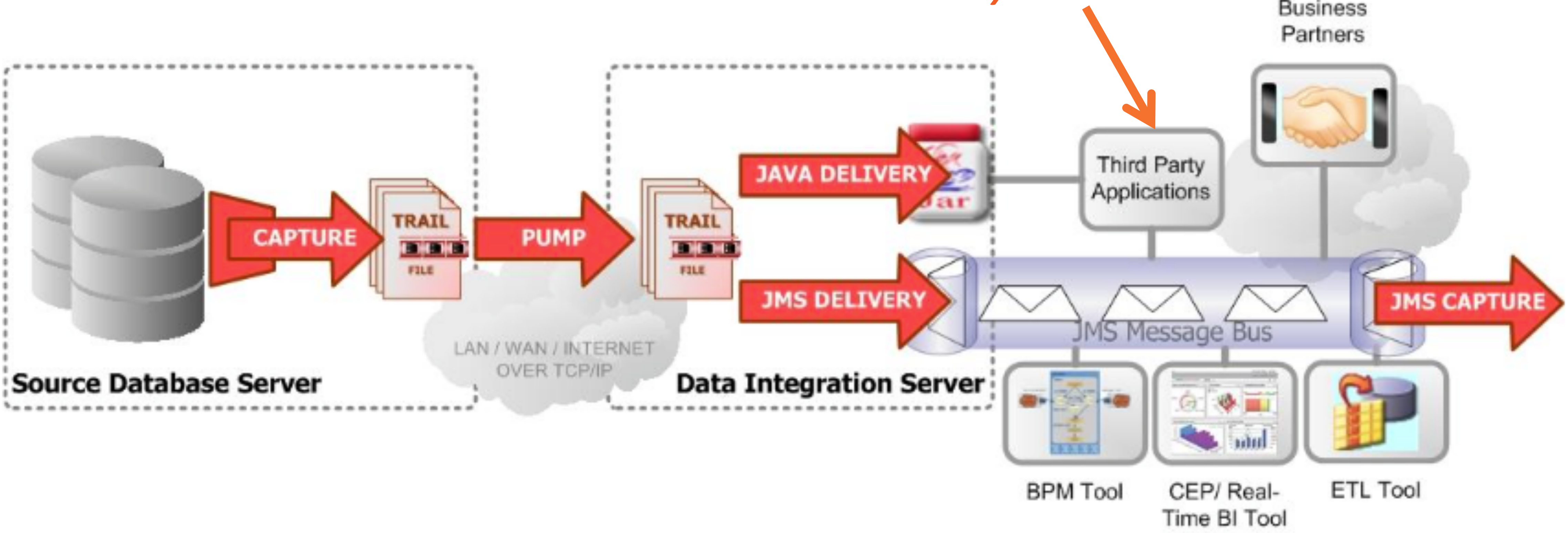
- Real-time database replication technology acquired by Oracle in 2009
- Heterogeneous: sources and targets can be different brands of DBMS
- Conceptually, tails the source DBMS's transaction log
- Transforms source transaction records into neutral-format "trail" file
- Pumps trail files to target hosts over TCP connections
- Robust, proven, industrial-strength, with HA, scaling, monitoring solutions
- Written in C, but has a Java Adapter ☺

GoldenGate Data Distribution Configuration



GoldenGate Adapter for Java

Why not an IMDG?



Oracle Coherence GoldenGate HotCache

- A feature of Oracle Coherence first released in 12.1.2, July 2013
- Specializes the GoldenGate Java Adapter to replicate into Coherence
- Uses JPA for mapping from database rows to cache entries
- Uses JPA "in reverse" – entity hydration driven by database transactions, not by application calls to JPA API
- Real-time, event-driven cache refresh from database transactions, using proven database replication and ORM technology, and the JPA standard
- Simpler and more OOTB than triggered messaging, and equally scalable, highly available, and monitorable

Fundamentally, It’s A Mapping Problem

- What GoldenGate has:

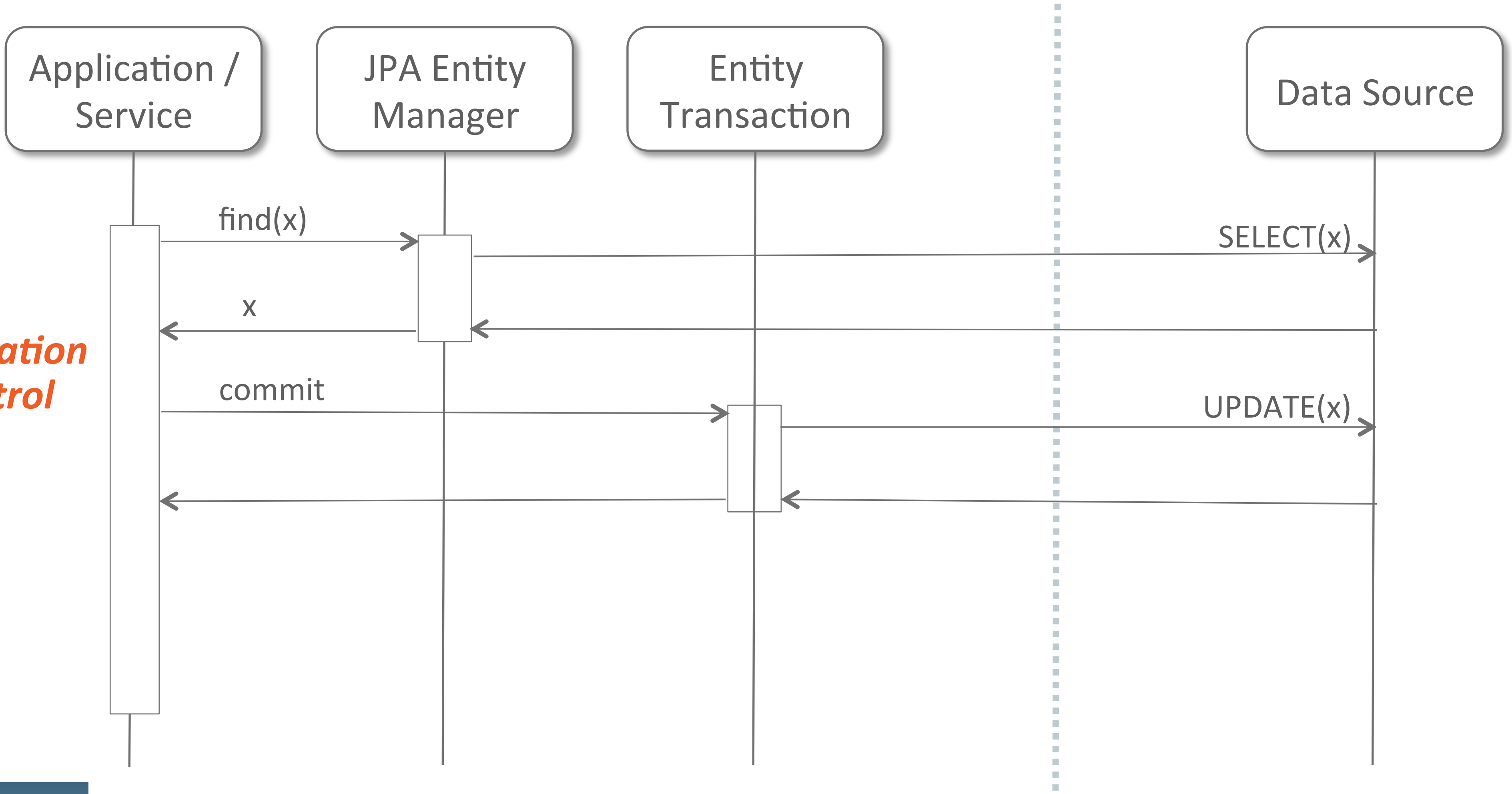
```
<operation table='APPLICATION.CONTACT' type='INSERT' ts='2015-10-24 01:15:03.000000' pos='00000000000000001076' numCols='15'>
 <col name='FIRSTNAME' index='0'>
   <after><![CDATA[Barack]]></after>
 </col>
 <col name='LASTNAME' index='1'>
   <after><![CDATA[Obama]]></after>
 </col>
 <col name='BIRTHDATE' index='2'>
   <after><![CDATA[1961-08-04:00:00:00]]></after>
 </col>
 <col name='HOME_STREET_1' index='3'>
   <after><![CDATA[1600 Pennsylvania Avenue NW]]></after>
 </col>
```

- To which cache and entry should this correspond?

Object-Relational-Cache Mapping

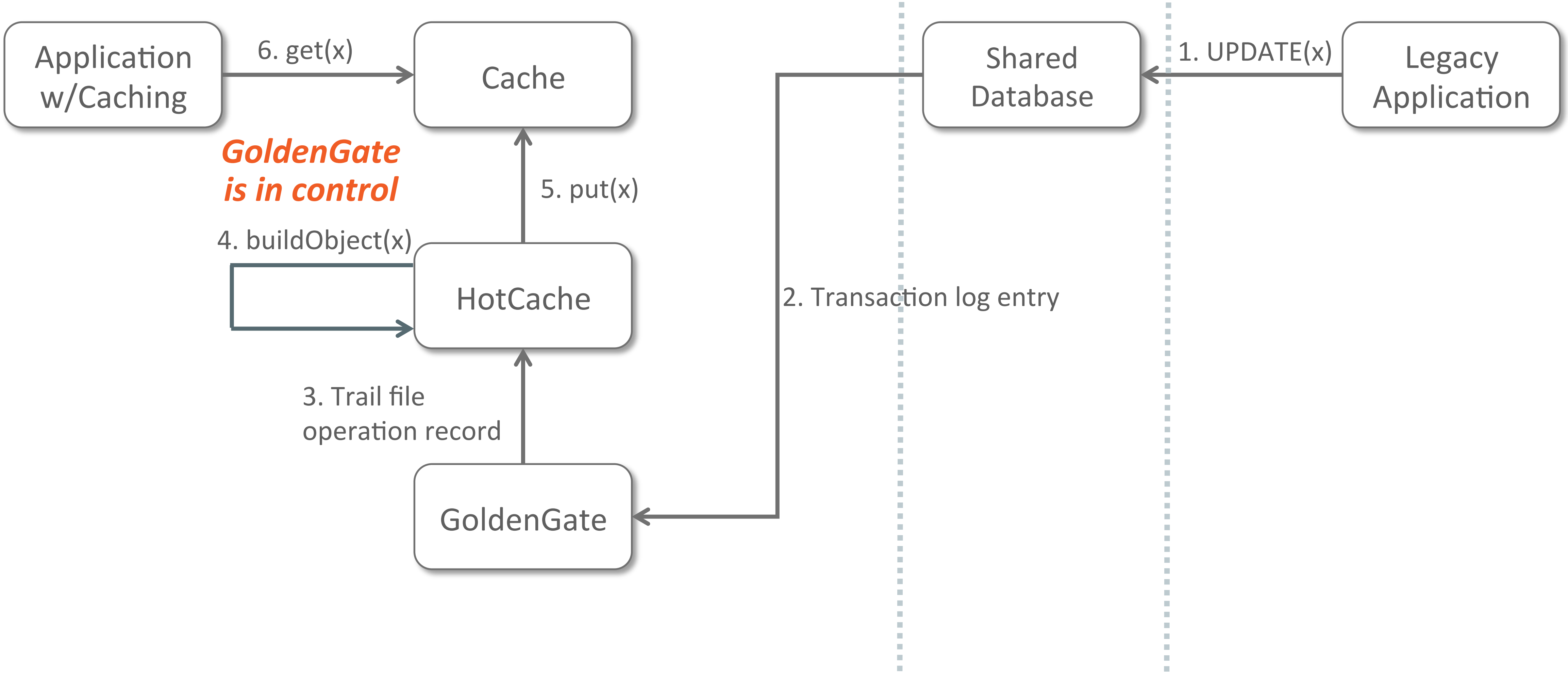
- JPA: (table, PK, row) => (entity class, entity ID, entity state)
- HotCache currently:
  - Entity class => cache name
  - Entity ID => cache key
- Cache Per Entity Type is a longstanding pattern in ORM caching
- Given Cache Per Entity Type, how to handle entity relationships in cache?

Normal JPA Control Flow



The application is in control

JPA In Reverse Control Flow



Session Agenda

1 Context and problem

2 Evaluation of known solutions

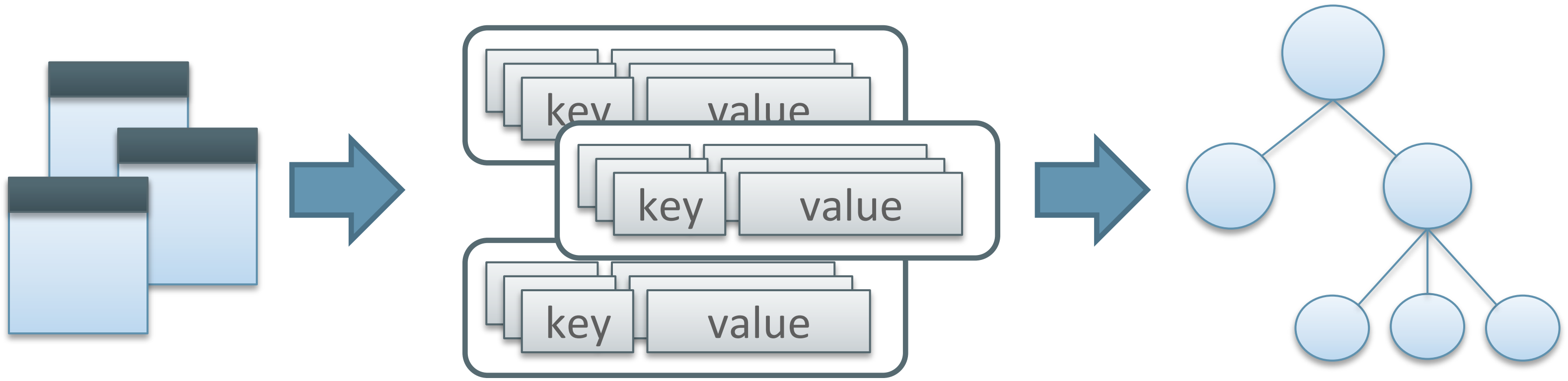
3 Using database replication technology

4 Consequences and nuances of JPA in reverse

5 Summary, Q&A / discussion

Road from Relational model to object model

- Bridging gap between Table model and Object model with reverse JPA not always straightforward
  - Unfriendly/static legacy database model
  - Induced by the intermediate key-value/map structure
  - Intermediate transformations required to fulfill the desired target domain object model



Tables

Maps (caches)

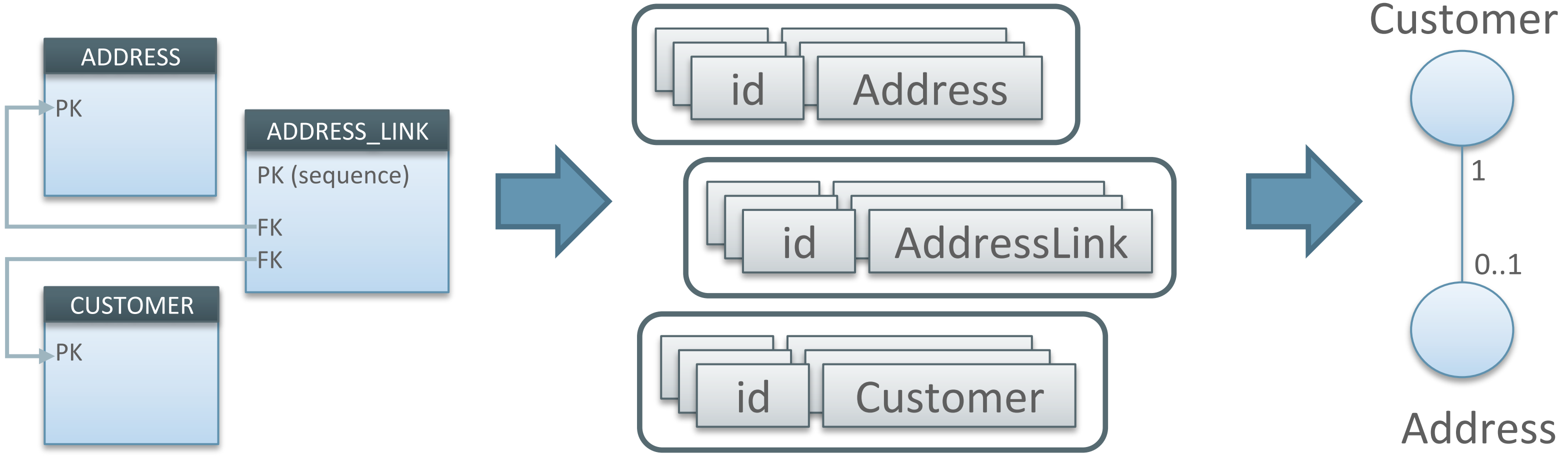
Domain object model

Ternary relationship

- From the database model:
  - ADDRESS, ADDRESS_LINK, CUSTOMER tables
- After reverse JPA*:
  - Maps of Customer, AddressLink and Address objects

- What you might want:
  - Customer object with a reference to their active Address object

* 1:1 table to cache mapping assumed here



Retrieving Customer and Address objects

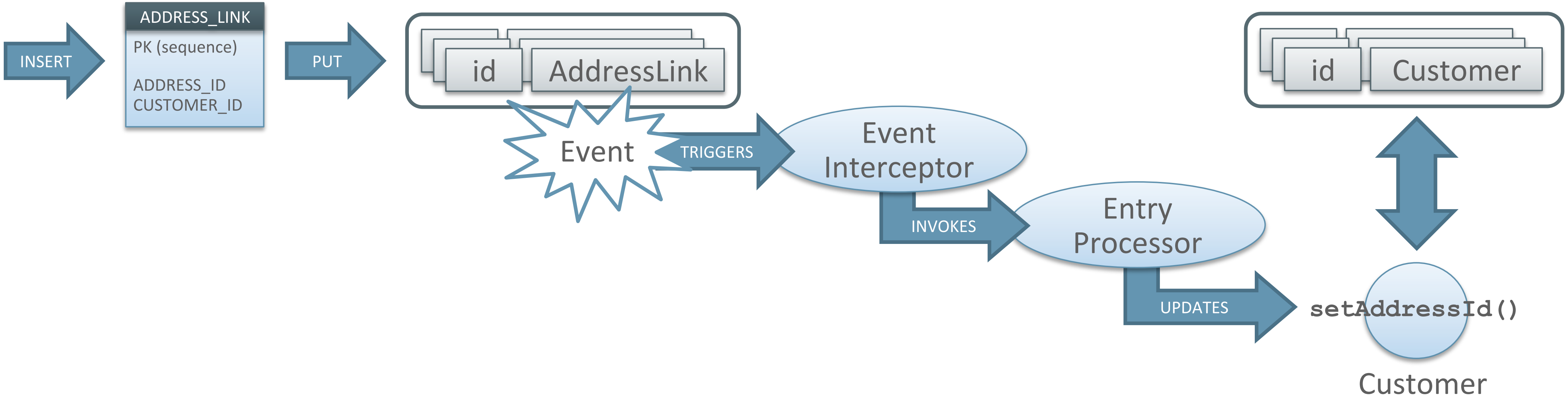
1. Get Customer object from cache
2. Query AddressLink cache with Customer id, keep only highest sequence number
3. Get Address object from cache based on found Address id

- That's three requests to caches…
- Any better option?

➢ Let's get rid of intermediate query by pre-resolving Address association at AddressLink cache insertion time

Coherence Live Events to the rescue

- Use Live Events to intercept insert events on AddressLink cache
- Invoke an Entry Processor from interceptor to do a concurrent safe update on Customer cache



Result

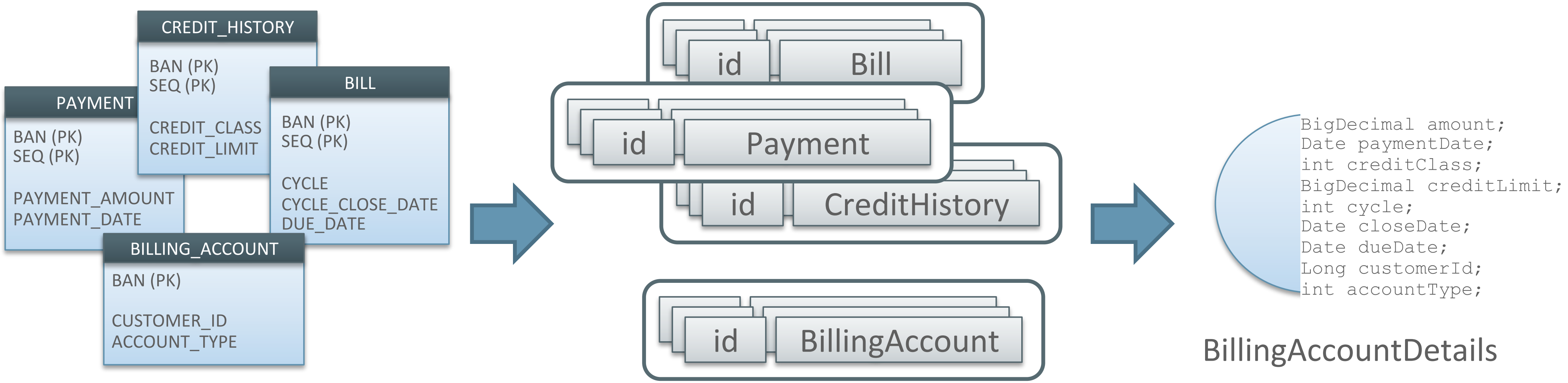
- The Customer object in cache has the Address id which is the cache key to access the Address object
- This field is transient (@Transient), not handled by JPA
- An hypothetical getAddress() method within the Customer object could get the Address object from the Address cache

➢ What if you require non-lazy loading of Address?

  - Address object could be pre-queried at AddressLink insertion time then embedded (replace previous instance) within Customer object
  - Ultimately depends on usage scenario, which one best fits the need

Composite view

- Granularity of source tables might be too fine
- Table view is not an option
- Coarse grained objects are required on consumer/client side
- We want to minimize object assembly done at runtime, have ready to consume objects



CREDIT_HISTORY
BAN (PK)
SEQ (PK)
CREDIT_CLASS
CREDIT_LIMIT

PAYMENT
BAN (PK)
SEQ (PK)
PAYMENT_AMOUNT
PAYMENT_DATE

BILL
BAN (PK)
SEQ (PK)
CYCLE
CYCLE_CLOSE_DATE
DUE_DATE

BILLING_ACCOUNT
BAN (PK)
CUSTOMER_ID
ACCOUNT_TYPE

id Bill

id Payment

id CreditHistory

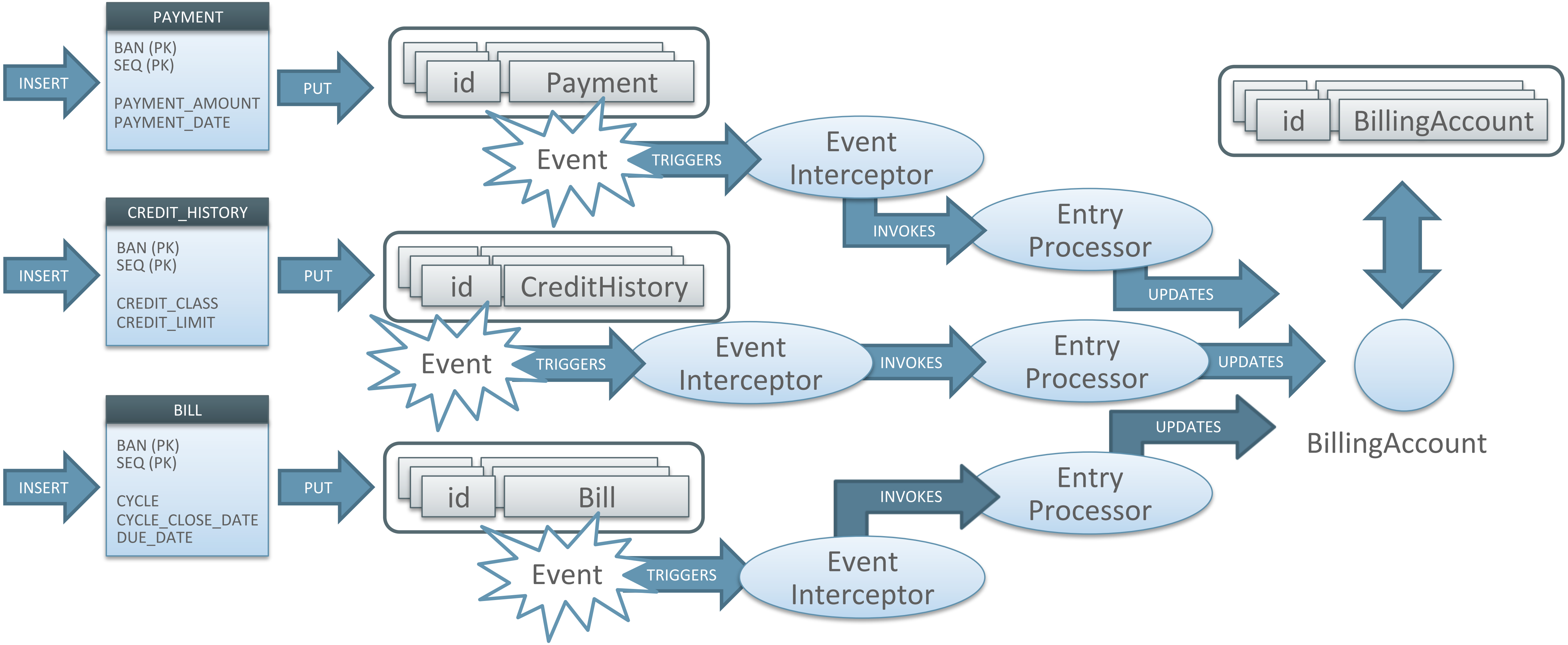
id BillingAccount

```
BigDecimal amount;
Date paymentDate;
int creditClass;
BigDecimal creditLimit;
int cycle;
Date closeDate;
Date dueDate;
Long customerId;
int accountType;
```

BillingAccountDetails

Pushing live data to composite objects

- Use Live Events to intercept insert events on contributing source objects/caches (Payment, Bill, CreditHistory)
- Invoke Entry Processors to do a concurrent safe update on target composite object in cache (BillingAccountDetails)

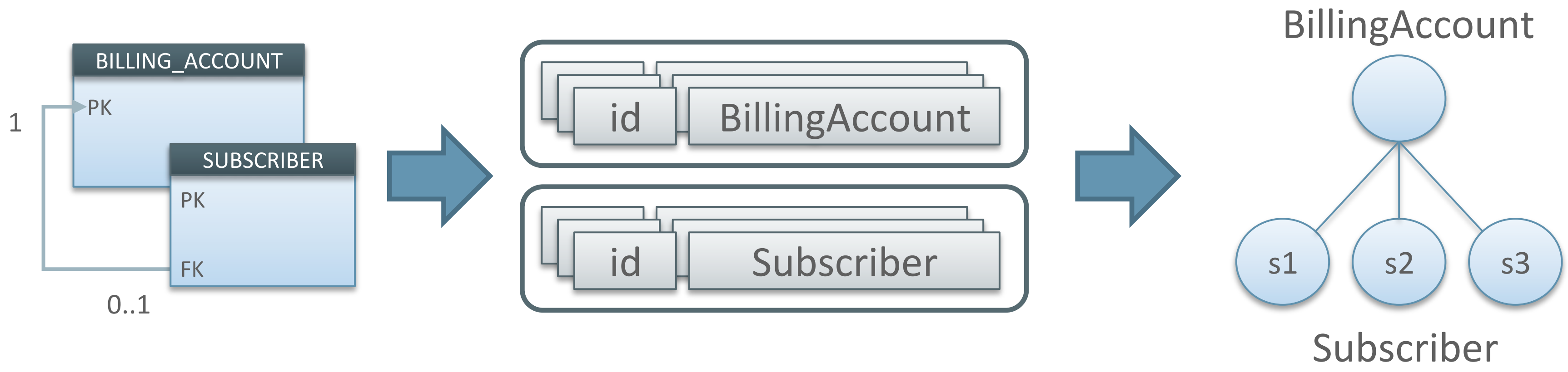

INSERT
PAYMENT
BAN (PK)
SEQ (PK)
PAYMENT_AMOUNT
PAYMENT_DATE
PUT
id
Payment
Event
TRIGGERS
Event Interceptor
INVOKES
Entry Processor
UPDATES
INSERT
CREDIT_HISTORY
BAN (PK)
SEQ (PK)
CREDIT_CLASS
CREDIT_LIMIT
PUT
id
CreditHistory
Event
TRIGGERS
Event Interceptor
INVOKES
Entry Processor
UPDATES
INSERT
BILL
BAN (PK)
SEQ (PK)
CYCLE
CYCLE_CLOSE_DATE
DUE_DATE
PUT
id
Bill
Event
TRIGGERS
Event Interceptor
INVOKES
Entry Processor
UPDATES
id
BillingAccount
BillingAccount

Additional remarks

- Caches holding contributing objects act as staging caches
- Objects in staging caches can be evicted if they are no longer required
- Why not use @Embedded objects as JPA mapping?
  - When 1:1 relationship is simple (no complex conditional logic) and embedded objects do not need to be accessed independently, this is the right choice

Preserving parent-child relationship info

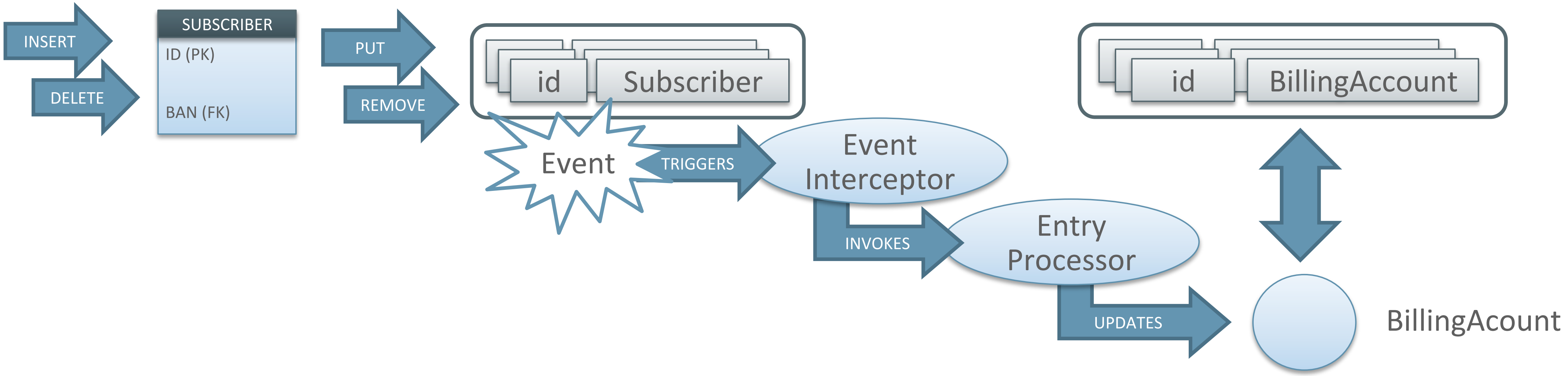
- Object references are not preserved across caches
- To keep relationship information between objects kept in different maps/ caches, the preferred approach is to hold the children’s ids (keys) in the parent object*
- Referential integrity needs to be preserved as child collection gets updated



*This technique is based on Collection-Oriented repositories concept from DDD, you can read more about it in Vaughn Vernon’s book titled “Implementing Domain-Driven Design” from Addison-Wesley

Pushing children's keys to the parent

- Use Live Events to intercept insert events on child cache (Subscriber)
- Invoke Entry Processors to do a concurrent safe update on the Set attribute holding Subscriber keys in parent cache object (BillingAccountInfo)



```
Set<Subscriber> getSubscribers()
setSubscribers(Set<Subscriber> set)
```

Additional remarks

- You can also resolve parent child relationship at runtime using cache queries / filters, assuming required cache indexes are created
- If you use the children key collection in parent approach, keep in mind that parent object and the set is *deserialized* and *serialized* each time it is updated by the EntryProcessor
- If you expect large key sets and children are update in batches (bulk updates) this can become a bottleneck since updates to same cache entry are queued

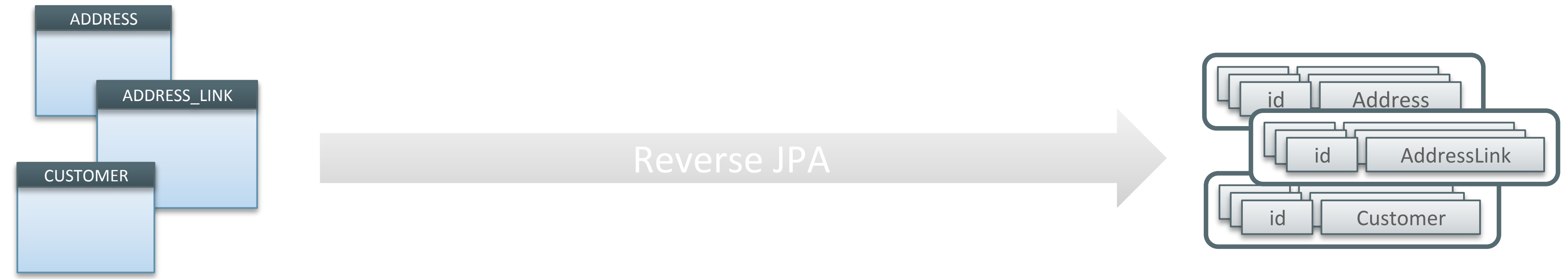
A special odd case

- Change to database PK
  - An update to a database PK should ideally translate to an update to a cache key
  - But a map key cannot be updated, otherwise it would represent another entry
  - It must be handled manually as a cache delete (old key) and put (new key), assuming key rows are mapped within object definition and can be intercepted

General recommendations

- Don’t get stuck at JPA mapping level!
- Only in rare occasions you will be able to modify the source database model, so you have to adapt
- Make objects Evolvable (that table column we forgot to map…)
- Produce an initial simple mapping and test the whole chain (Golden Gate to cache consumer), iterate, refine
- Use Coherence Live Events API to massage the data to suit your final object model if it is half cooked after JPA reverse mapping / HotCache processing
- Don’t forget queries, sometimes it all can be done at data access time with Coherence filters

Demo – Reverse JPA in action



ADDRESS

ADDRESS_LINK

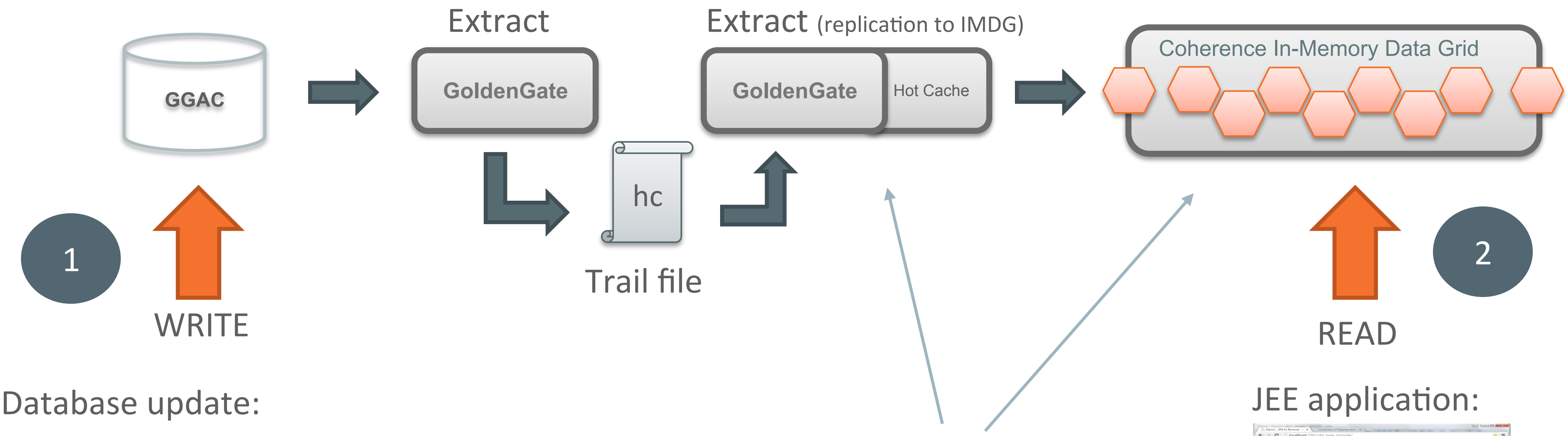
CUSTOMER

Reverse JPA

id | Address

id | AddressLink

id | Customer



GGAC

Extract

GoldenGate

hc

Trail file

Extract (replication to IMDG)

GoldenGate | Hot Cache

Coherence In-Memory Data Grid

1

WRITE

2

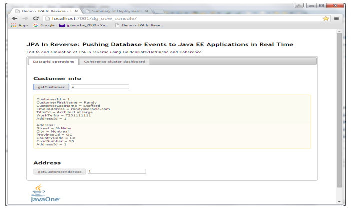
READ

Database update:

```
INSERT INTO ADDRESS …
INSERT INTO ADDRESS_LINK …
```

PEAK at debug files

JEE application:



JPA In Reverse: Pushing Database Events to Java EE Applications in Real Time

End to end simulation of JPA in reverse using GoldenGate/HotCache and Coherence

Customer info

getCustomer 1

```
CustomerId = 1
CustomerFirstName = Randy
CustomerLastName = Stafford
EmailAddress = randy@oracle.com
TitleCd = Architect at large
WorkTelNo = 7201111111
AddressId = 1

Address:
Street = McNider
City = Montreal
ProvinceCd = QC
CountryCode = CA
CivicNumber = 95
AddressId = 1
```

Address

getCustomerAddress 1

Session Agenda

1 Context and problem

2 Evaluation of known solutions

3 Using database replication technology

4 Consequences and nuances of JPA in reverse

5 Summary, Q&A / discussion

Summary

- Cache invalidation remains a real need in Java enterprise applications
- Four different known solutions became prevalent, each with demerits
- Using database replication technology, and JPA in reverse, is a promising new approach
- We have two years' production experience in many mission-critical applications discovering the consequences of applying JPA in reverse, the patterns of working with it, and how the approach can be enhanced

Questions?

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Session Surveys

Help us help you!!

- Oracle would like to invite you to take a moment to give us your session feedback. Your feedback will help us to improve your conference.
- Please be sure to add your feedback for your attended sessions by using the Mobile Survey or in Schedule Builder.

Integrated Cloud

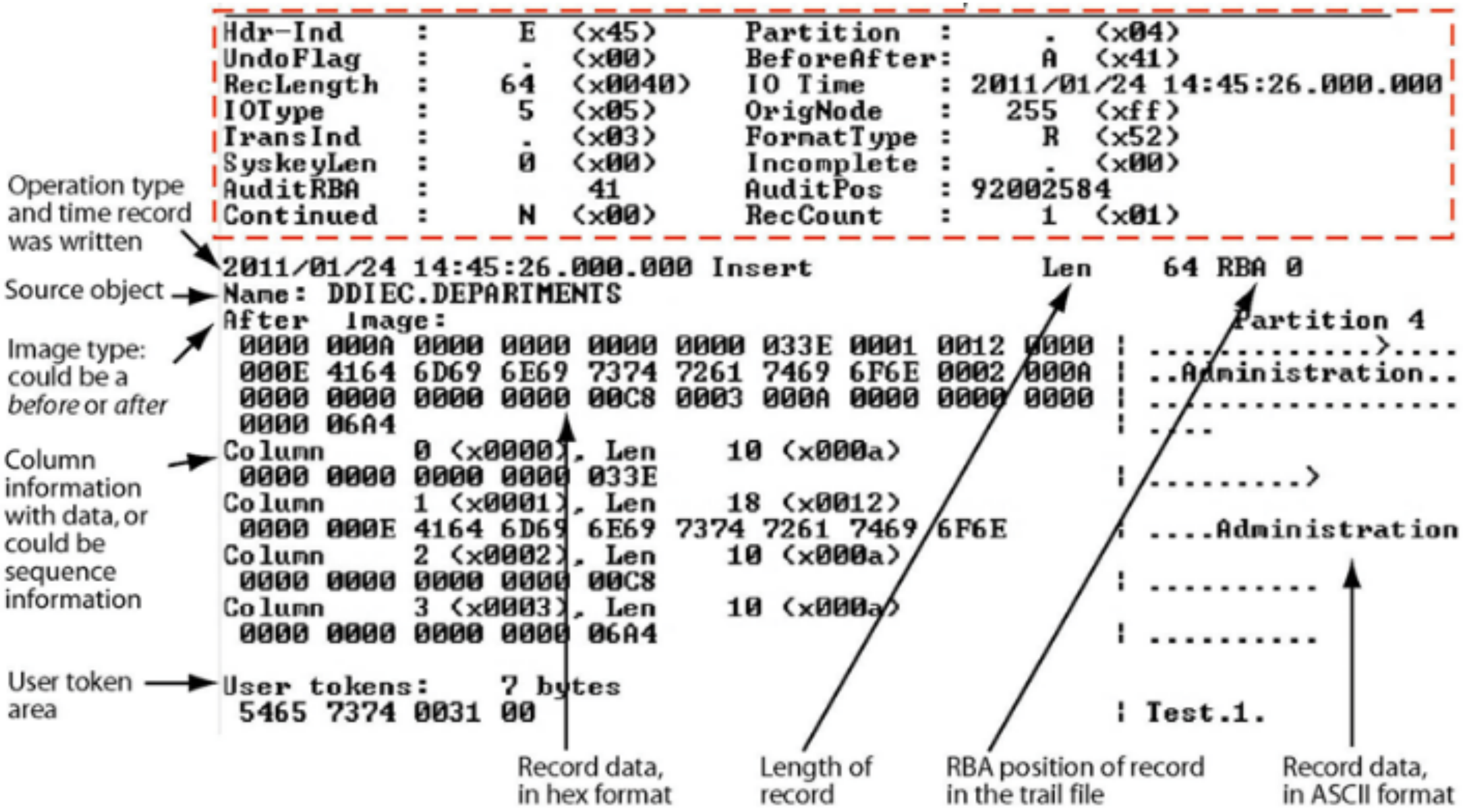
Applications & Platform Services

JavaOne™

ORACLE®

ORACLE®

Golden Gate Trail File Operation



```
Hdr-Ind     :     E  (x45)     Partition  :     .  (x04)
UndoFlag    :     .  (x00)     BeforeAfter:     A  (x41)
RecLength   :    64  (x0040)   IO Time    : 2011/01/24 14:45:26.000.000
IOType      :     5  (x05)     OrigNode   :   255  (xff)
TransInd    :     .  (x03)     FormatType :     R  (x52)
SyskeyLen   :     0  (x00)     Incomplete :     .  (x00)
AuditRBA    :          41      AuditPos   : 92002584
Continued   :     N  (x00)     RecCount   :     1  (x01)

2011/01/24 14:45:26.000.000 Insert              Len    64 RBA 0
Name: DDIEC.DEPARTMENTS
After  Image:                                              Partition 4
 0000 000A 0000 0000 0000 0000 033E 0001 0012 0000 | ..............>....
 000E 4164 6D69 6E69 7374 7261 7469 6F6E 0002 000A | ..Administration..
 0000 0000 0000 0000 00C8 0003 000A 0000 0000 0000 | ....................
 0000 06A4                                         | ....
Column     0 (x0000), Len    10 (x000a)
 0000 0000 0000 0000 033E                          | .........>
Column     1 (x0001), Len    18 (x0012)
 0000 000E 4164 6D69 6E69 7374 7261 7469 6F6E      | ....Administration
Column     2 (x0002), Len    10 (x000a)
 0000 0000 0000 0000 00C8                          | ..........
Column     3 (x0003), Len    10 (x000a)
 0000 0000 0000 0000 06A4                          | ..........

User tokens:    7 bytes
 5465 7374 0031 00                                 | Test.1.
```

Operation type and time record was written

Source object

Image type: could be a *before* or *after*

Column information with data, or could be sequence information

User token area

Record data, in hex format

Length of record

RBA position of record in the trail file

Record data, in ASCII format