Second-Level Cache in JPA Explained

Patrycja Wegrzynowicz
CTO, Yonita, Inc.
JavaOne 2016

About Me

- 15+ professional experience
  - Software engineer, architect, head of software R&D
- Author and speaker
  - JavaOne, Devoxx, JavaZone, TheServerSide Java Symposium, Jazoon, OOPSLA, ASE, others
- Top 10 Women in Tech 2016 in Poland
- Founder and CTO of Yonita
  - Automated detection and refactoring of software defects
  - Trainings and code reviews
  - Security, performance, concurrency, databases
- Twitter @yonlabs

YONITA

About Me

- 15+ professional experience
  - Software engineer, architect, head of software R&D
- Author and speaker
  - JavaOne, Devoxx, JavaZone, TheServerSide Java Symposium, Jazoon, OOPSLA, ASE, others
- Top 10 Women in Tech 2016 in Poland
- Founder and CTO of Yonita
  - Automated detection and refactoring of software defects
  - Trainings and code reviews
  - Security, performance, concurrency, databases
- Twitter @yonlabs

YONITA

Agenda

- Why cacheing is important?
- 1st Level Cache and 2nd Level Cache
- JPA configuration parameters for cache
- JPA API for cache
- Hibernate 2nd Level Cache
- EclipseLink 2nd Level Cache (a bit)

Databases

YONITA

Databases

YONITA

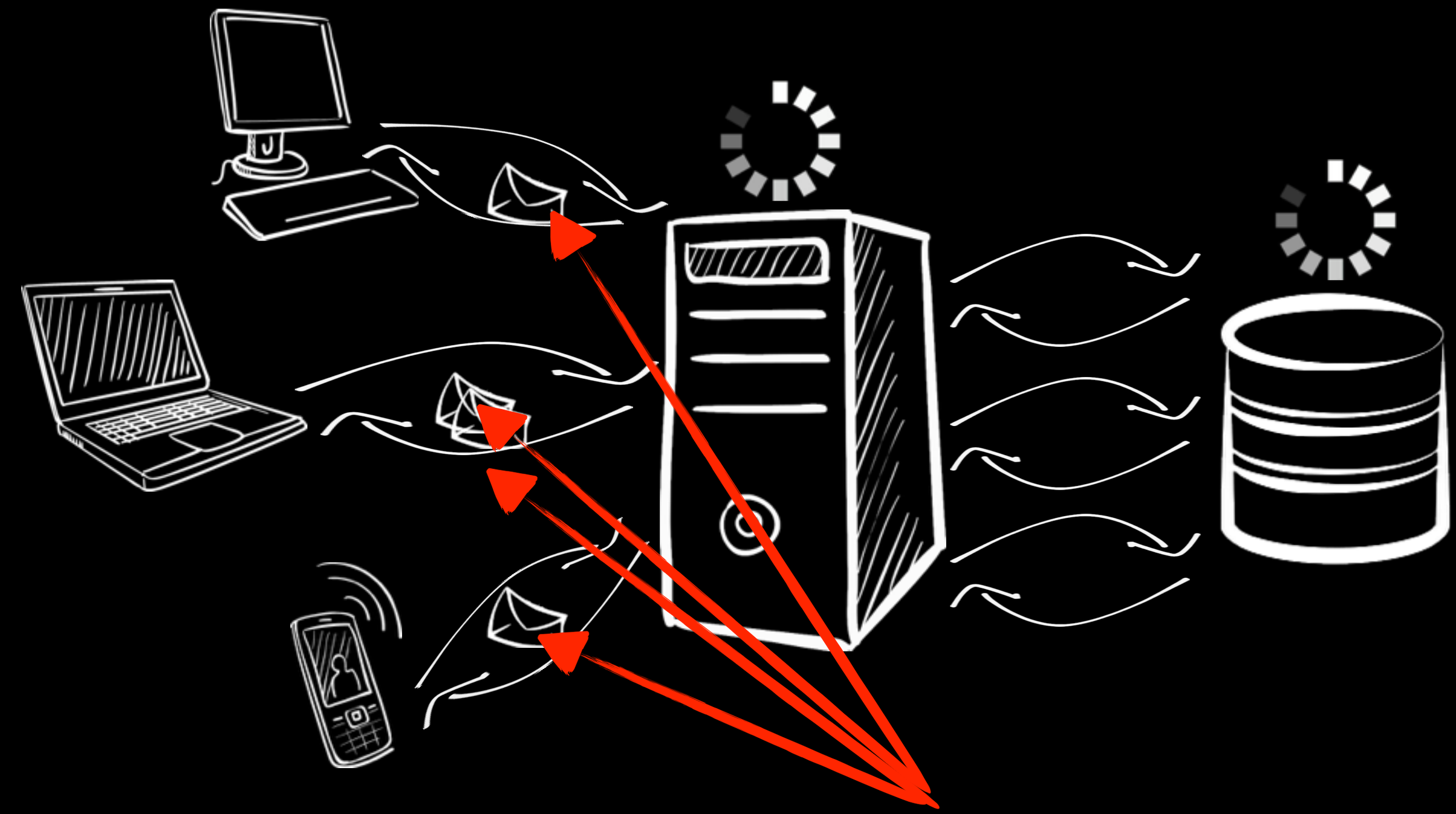
Performance

YONITA

Request Handling

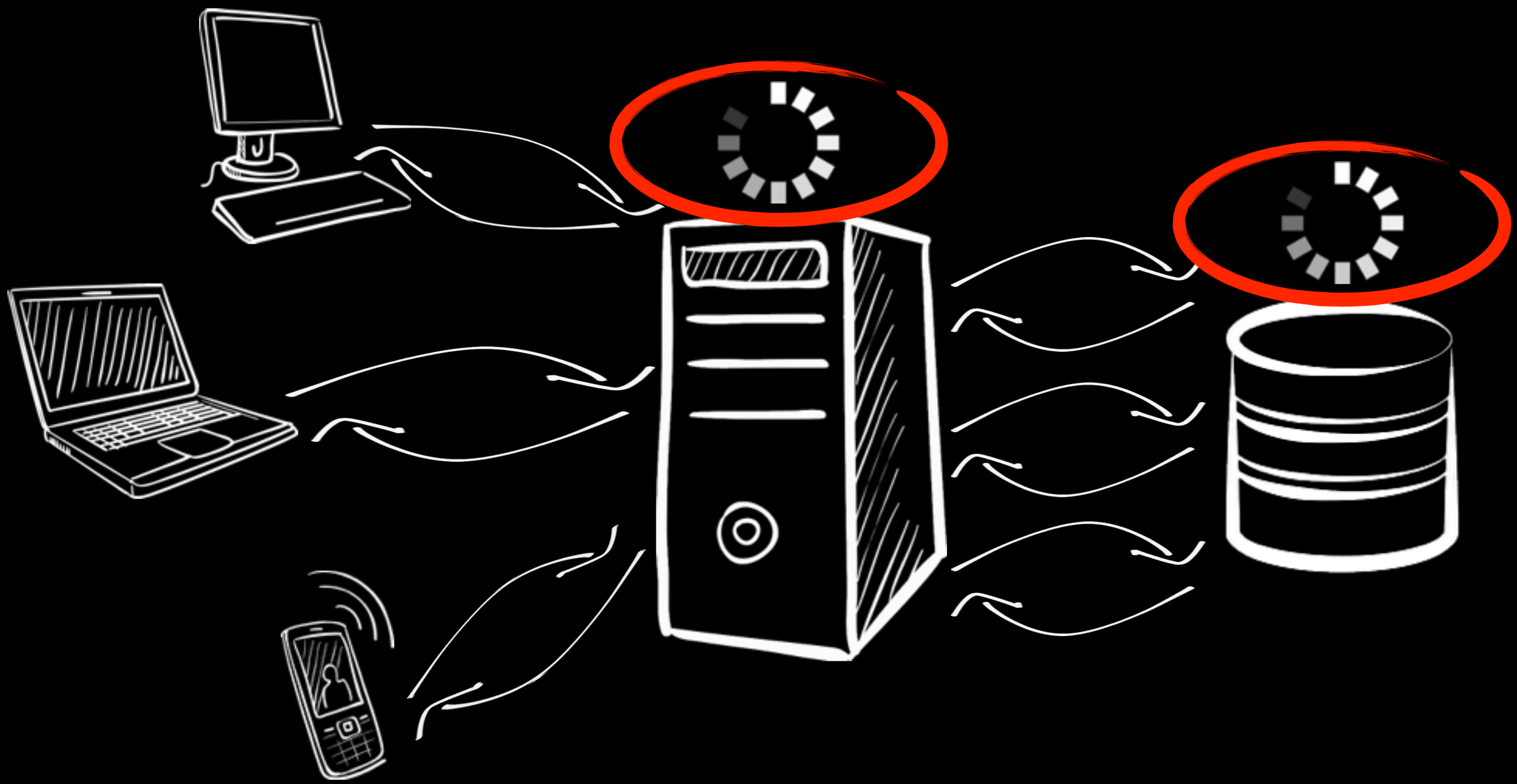
YONITA

Performance: Throughput



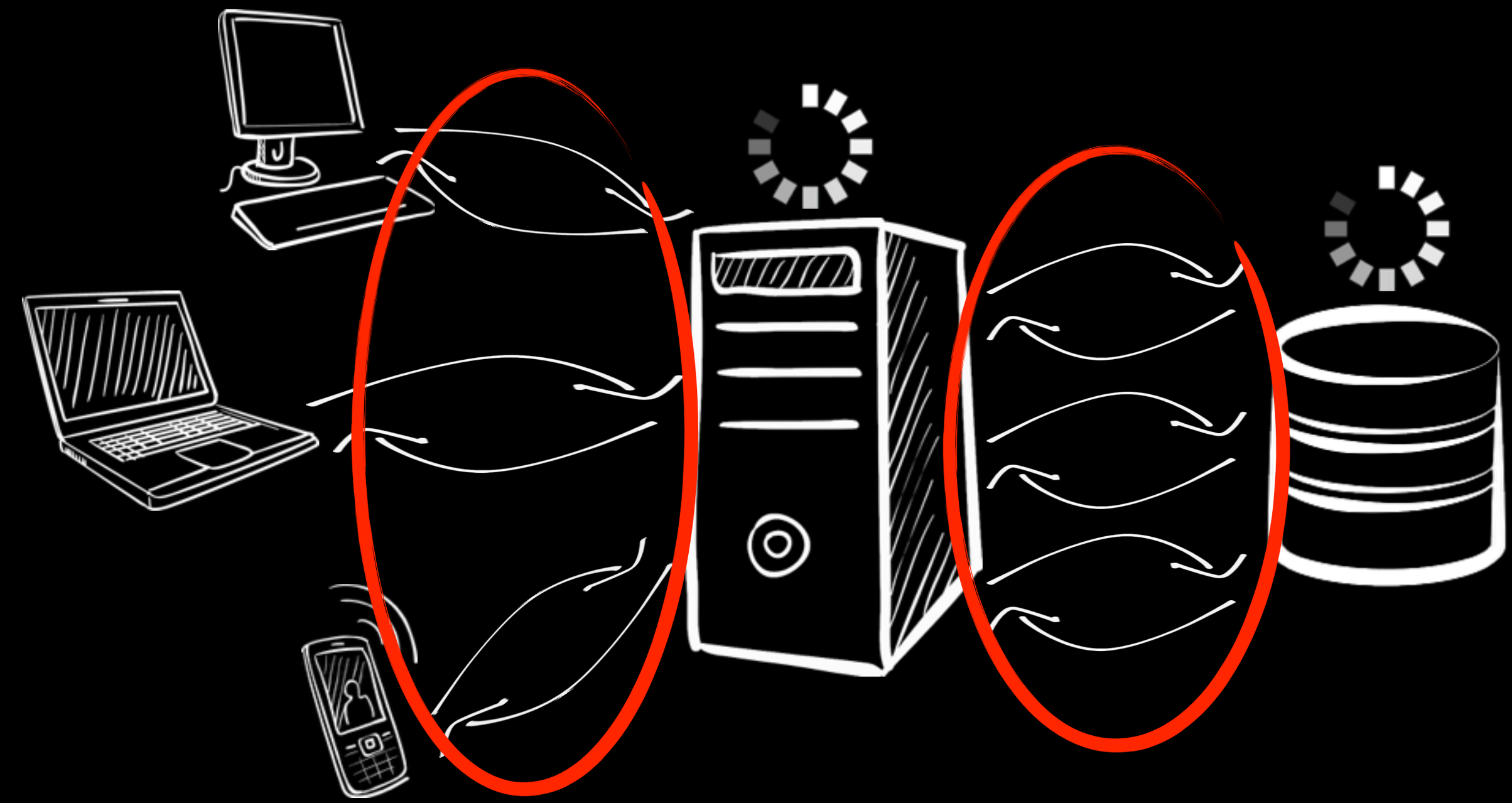
YONITA

Performance: Execution Time



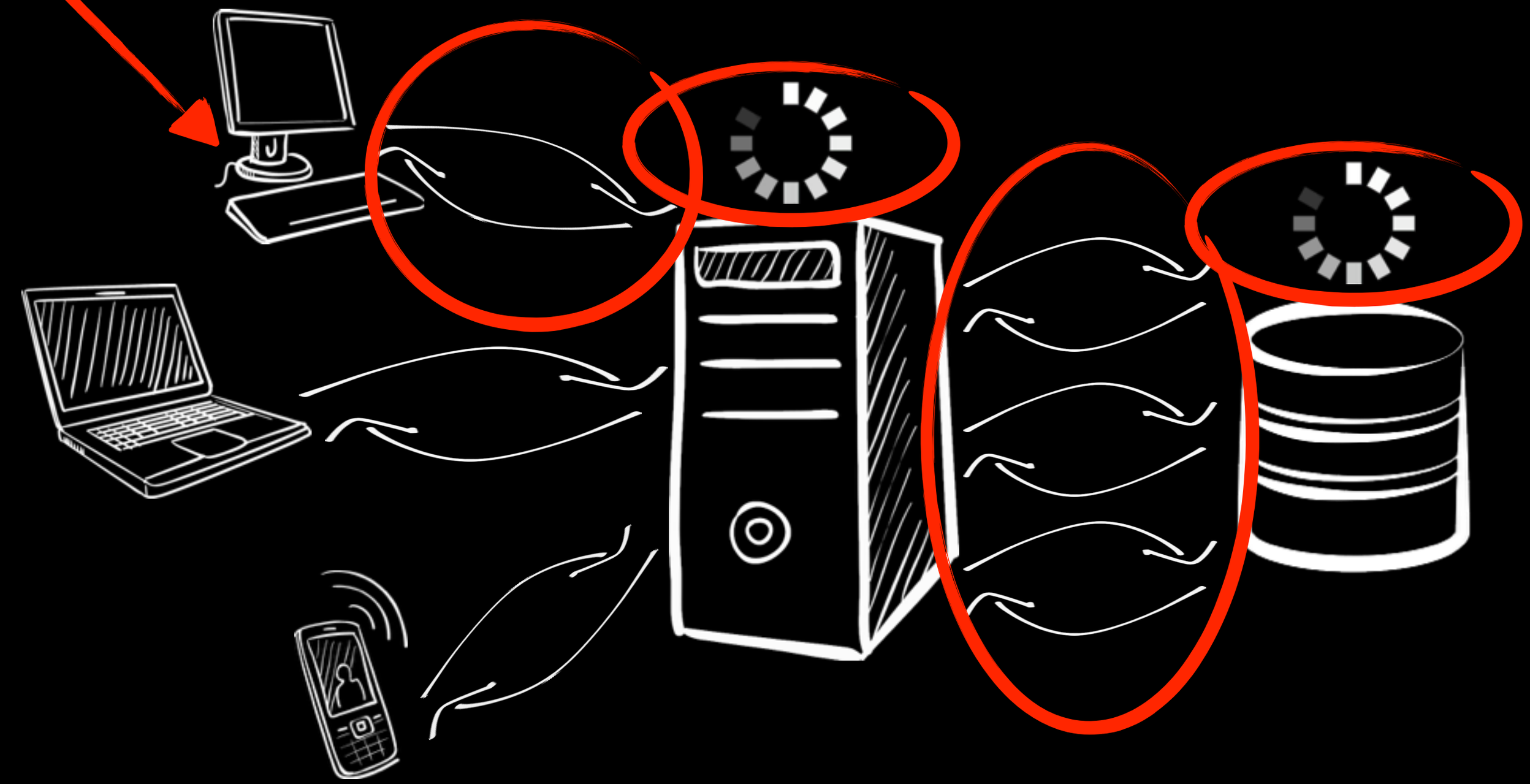
YONITA

Performance: Latency



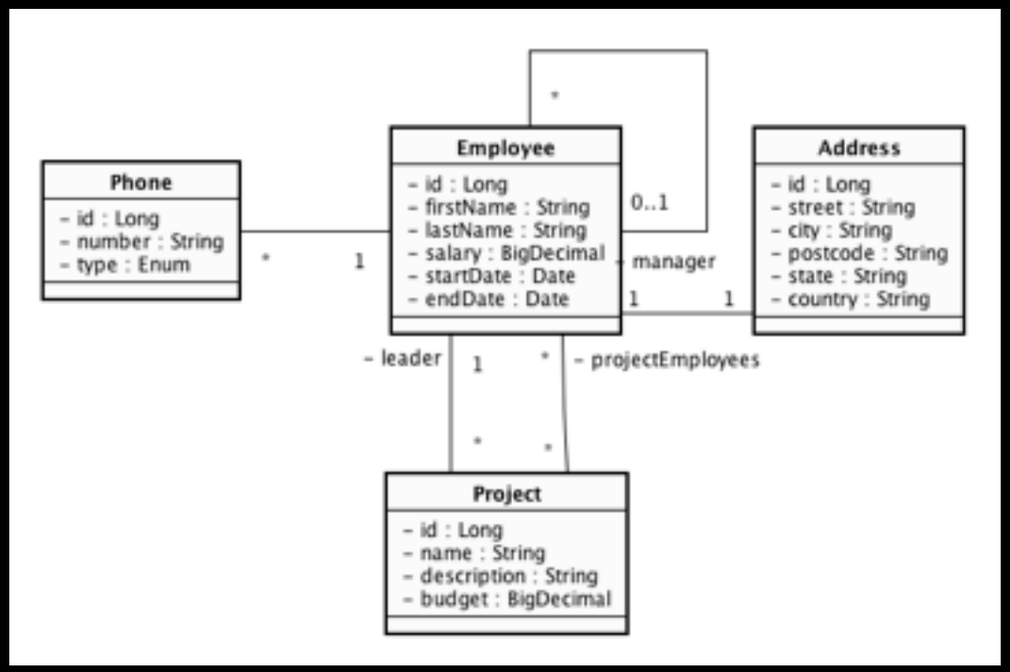
YONITA

Performance: Response Time



YONITA

Example



YONITA

Employee Entity

```
@Entity public class Employee {
  @Id @GeneratedValue
  private Long id;
  private String firstName;
  private String lastName;
  private BigDecimal salary;
  @OneToOne @JoinColumn(name = ”address_id”)
  private Address address;
  @Temporal(TemporalType.DATE)
  private Date startDate;
  @Temporal(TemporalType.DATE)
  private Date endDate;
  @ManyToOne @JoinColumn(name = ”manager_id”)
  private Employee manager;
  // …
}
```

Sum of Salaries By Country Select All (1)

```
TypedQuery<Employee> query = em.createQuery(
    "SELECT e FROM Employee e", Employee.class);
List<Employee> list = query.getResultList();

// calculate sum of salaries by country
// map: country->sum
Map<String, BigDecimal> results = new HashMap<>();
for (Employee e : list) {
    String country = e.getAddress().getCountry();
    BigDecimal total = results.get(country);
    if (total == null) total = BigDecimal.ZERO;
    total = total.add(e.getSalary());
    results.put(country, total);
}
```

Sum of Salaries by Country
Select Join Fetch (2)

```
TypedQuery<Employee> query = em.createQuery(
    "SELECT e FROM Employee e
     JOIN FETCH e.address", Employee.class);
List<Employee> list = query.getResultList();
```

```
// calculate sum of salaries by country
// map: country->sum
Map<String, BigDecimal> results = new HashMap<>();
for (Employee e : list) {
    String country = e.getAddress().getCountry();
    BigDecimal total = results.get(country);
    if (total == null) total = BigDecimal.ZERO;
    total = total.add(e.getSalary());
    results.put(country, total);
}
```

Sum of Salaries by Country Projection (3)

```
Query query = em.createQuery(
    "SELECT e.salary, e.address.country
     FROM Employee e");
List<Object[]> list = (List<Object[]>) query.getResultList();

// calculate sum of salaries by country
// map: country->sum
Map<String, BigDecimal> results = new HashMap<>();
for (Object[] e : list) {
    String country = (String) e[1];
    BigDecimal total = results.get(country);
    if (total == null) total = BigDecimal.ZERO;
    total = total.add((BigDecimal) e[0]);
    results.put(country, total);
}
```

Sum of Salaries by Country Aggregation JPQL (4)

```
Query query = em.createQuery(
    "SELECT SUM(e.salary), e.address.country
     FROM Employee e
     GROUP BY e.address.country");
List<Object[]> list = (List<Object[]>) query.getResultList();
// already calculated!
```

Comparison 1-4 (Hibernate)
100000 Employees, Different DB Locations

	Local DB (ping: ~0.05ms)	North California (ping: ~38ms)	EU Frankfurt (ping: ~420ms)
(1) Select All (N+1)	26756ms	**2-3 hours**	**~1 day**
(2) Select Join Fetch			
(3) Projection			
(4) Aggregation JPQL			

Comparison 1-4
100000 Employees, Different DB Locations

	Local DB (ping: ~0.05ms)	North California (ping: ~38ms)	EU Frankfurt (ping: ~420ms)
(1) Select All (N+1)	26756ms	**2-3 hours**	**~1 day**
(2) Select Join Fetch	4854ms	18027ms	25096ms
(3) Projection			
(4) Aggregation JPQL			

YONITA

Comparison 1-4
100000 Employees, Different DB Locations

	Local DB (ping: ~0.05ms)	North California (ping: ~38ms)	EU Frankfurt (ping: ~420ms)
(1) Select All (N+1)	26756ms	**2-3 hours**	**~1 day**
(2) Select Join Fetch	4854ms	18027ms	25096ms
(3) Projection	653ms	2902ms	5006ms
(4) Aggregation JPQL			

Comparison 1-4
100000 Employees, Different DB Locations

	Local DB (ping: ~0.05ms)	North California (ping: ~38ms)	EU Frankfurt (ping: ~420ms)
(1) Select All (N+1)	26756ms	**2-3 hours**	**~1 day**
(2) Select Join Fetch	4854ms	18027ms	25096ms
(3) Projection	653ms	2902ms	5006ms
(4) Aggregation JPQL	182ms	353ms	1198ms

Performance Tuning: Data

- Get your data in bigger chunks
  - Many small queries => many round-trips => huge extra time on transport => hig
- Move your da sing place
  - Large distance to data => long round-trip => high latency
- Don’t ask about the same data many times
  - Extra processing time + extra transport time

Cache

Cache is Everywhere

Web Cache

Application Cache

RDBMS Cache

DNS Cache

OS Files Cache

-

Second Level Cache in JPA

JPA Spec

- “Persistence providers are not required to support a second-level cache.”
- “Portable applications should not rely on support by persistence providers for a second-level cache.”

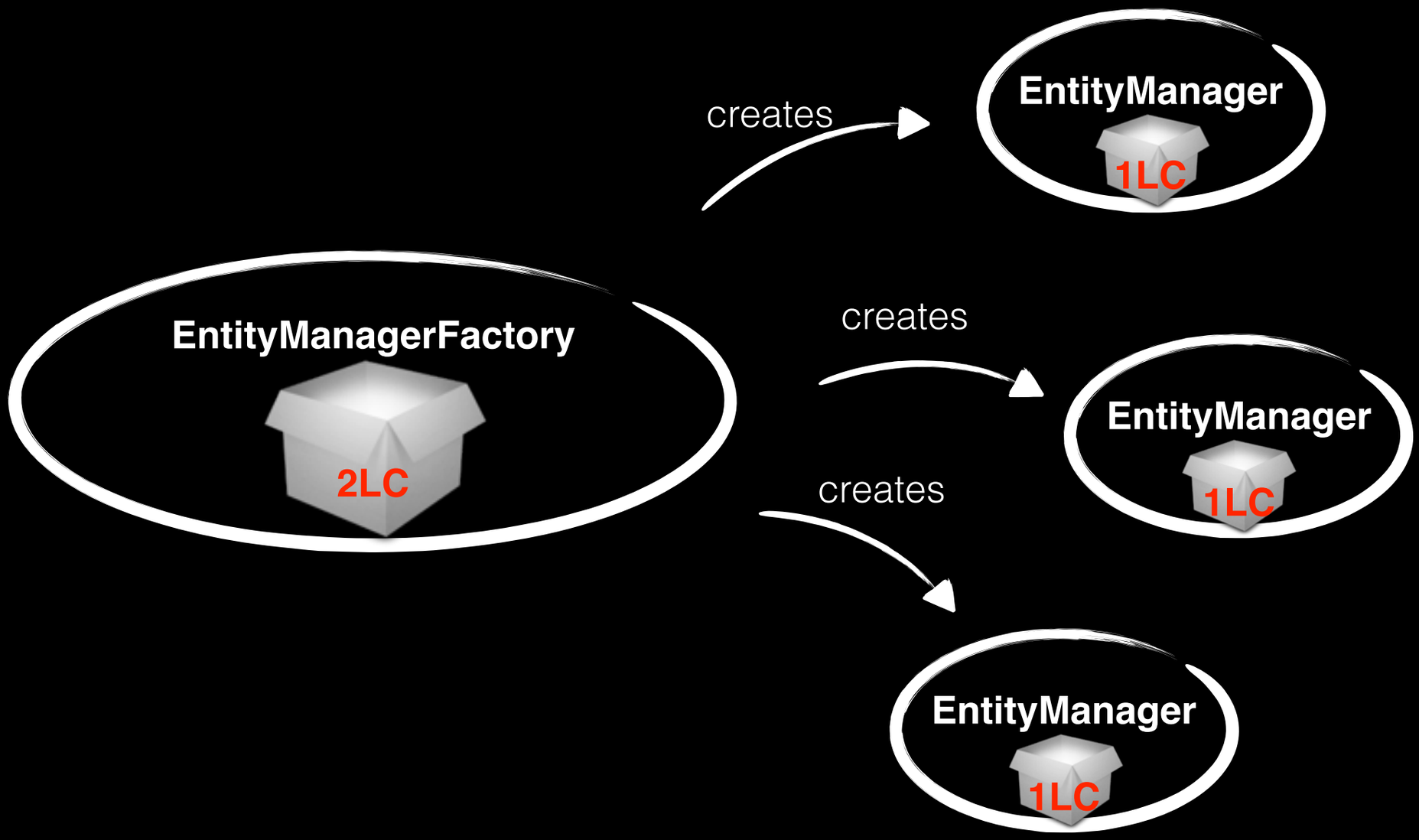
JPA Providers Poll

A. Hibernate
B. EclipseLink
C. OpenJPA
D. DataNuclues
E. Other

JPA Caches

- **First Level Cache**
  - Persistence Context
  - EntityManager
  - Not thread-safe
  - Available for **many transactions** on **one entity manager**
  - Always
- **Second Level Cache**
  - Persistence Unit
  - EntityManagerFactory
  - Thread-safe, shared
  - Available for **many entity managers** from **one entity manager factory**
  - Provider specific support

EntityManagerFactory



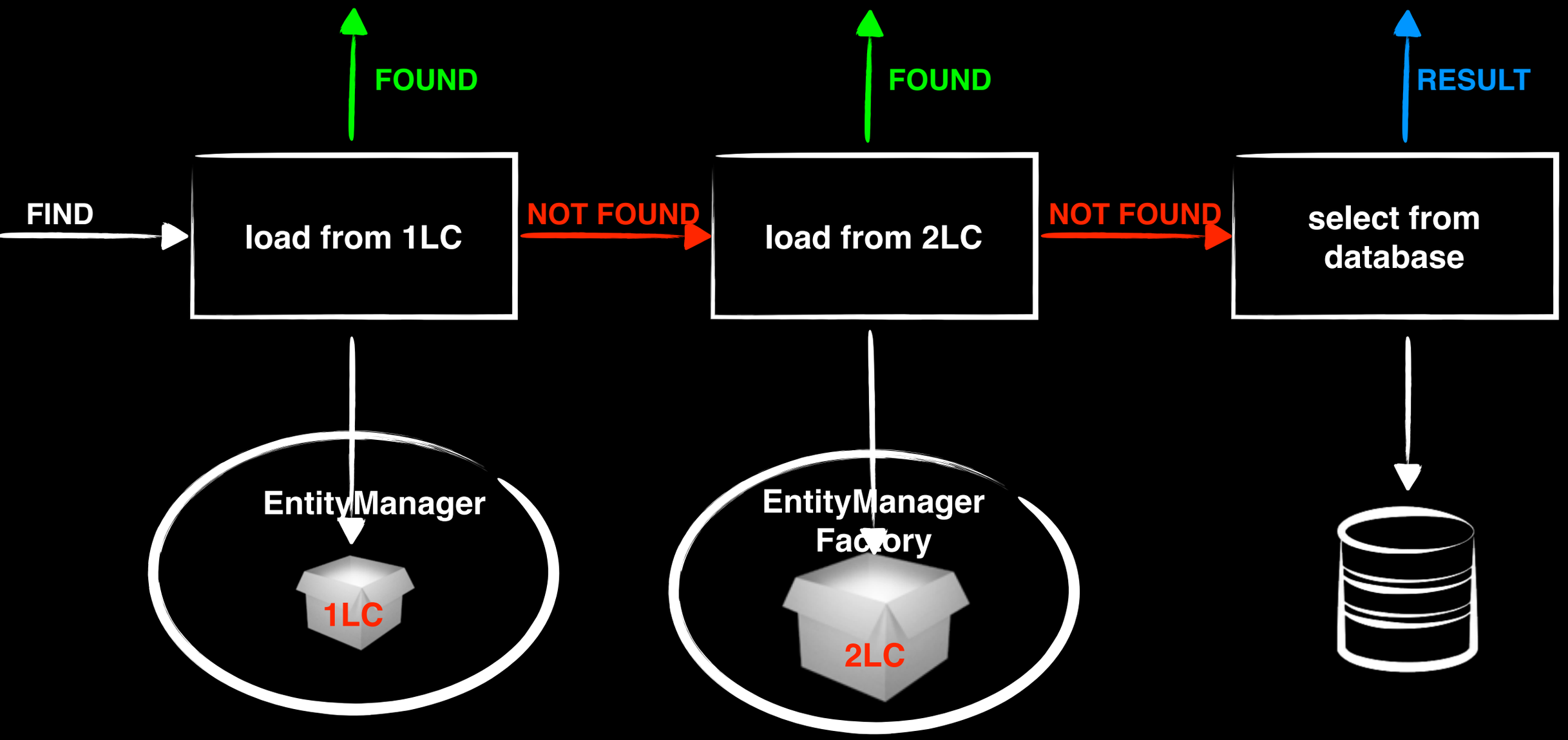
YONITA

Puzzle #1

```
em.getTransaction().begin();
Employee employee = em.find(Employee.class, 2L);
employee.getAddress().size();
Employee another = em.find(Employee.class, 2L);
em.getTransaction().commit();
```

(A) No cache used
(B) First Level Cache Used
(C) Second Level Cache Used
(D) None of the above

Loading



YONITA

Puzzle #1

```
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Employee employee = em.find(Employee.class, 2L);
employee.getAddress().size();
Employee another = em.find(Employee.class, 2L);
em.getTransaction().commit();
```

(A) No cache used
(B) First Level Cache Used
(C) Second Level Cache Used
(D) None of the above

Puzzle #1

```
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Employee employee = em.find(Employee.class, 2L);
employee.getAddress().size();
Employee another = em.find(Employee.class, 2L);
em.getTransaction().commit();
```

(A) No cache used

(B) First Level Cache Used

(C) Second Level Cache Used

(D) None of the above

Puzzle #2

```
EntityManager em1 = emf.createEntityManager();
em1.getTransaction().begin();
Employee employee = em1.find(Employee.class, 2L);
employee.getAddress().size();
em1.getTransaction().commit();
```

```
EntityManager em2 = emf.createEntityManager();
em2.getTransaction().begin();
Employee employee = em2.find(Employee.class, 2L);
employee.getAddress().size();
em2.getTransaction().commit();
```

(A) No cache used
(B) First Level Cache used
(C) Second Level Cache used
(D) None of the above

Puzzle #2

```
EntityManager em1 = emf.createEntityManager();
em1.getTransaction().begin();
Employee employee = em1.find(Employee.class, 2L);
employee.getAddress().size();
em1.getTransaction().commit();
```

```
EntityManager em2 = emf.createEntityManager();
em2.getTransaction().begin();
Employee employee = em2.find(Employee.class, 2L);
employee.getAddress().size();
em2.getTransaction().commit();
```

(A) No cache used
(B) First Level Cache used
(C) Second Level Cache used
(D) None of the above

Puzzle #3 (2LC Configured, Hibernate)

```
EntityManager em1 = emf.createEntityManager();
em1.getTransaction().begin();
Employee employee = em1.find(Employee.class, 2L);
employee.getAddress().size();
em1.getTransaction().commit();
```

```
EntityManager em2 = emf.createEntityManager();
em2.getTransaction().begin();
Employee employee = em2.find(Employee.class, 2L);
employee.getAddress().size();
em2.getTransaction().commit();
```

(A) No cache used
(B) First Level Cache used
(C) Second Level Cache used
(D) None of the above

Puzzle #3 (2LC Configured, Hibernate)

```
EntityManager em1 = emf.createEntityManager();
em1.getTransaction().begin();
Employee employee = em1.find(Employee.class, 2L);
employee.getAddress().size();
em1.getTransaction().commit();
```

```
EntityManager em2 = emf.createEntityManager();
em2.getTransaction().begin();
Employee employee = em2.find(Employee.class, 2L);
employee.getAddress().size();
em2.getTransaction().commit();
```

(A) No cache used

(B) First Level Cache used

(C) Second Level Cache used

(D) None of the above

Puzzle #4 (2LC Configured!)

```
em1.getTransaction().begin();
Employee employee = em1.find(Employee.class, 2L);
employee.getAddress().size();
em1.getTransaction().commit();

em2.getTransaction().begin();
TypedQuery<Employee> q =
   em.createQuery("SELECT e FROM Employee e WHERE e.id=:id", Employee.class);
q.setParameter("id", 21);
Employee employee = q.getSingleResult();
employee.getAddress().size();
em2.getTransaction().commit();
```

(A) No cache used
(B) First Level Cache used
(C) Second Level Cache used
(D) None of the above

Puzzle #4 (2LC Configured!)

```
em1.getTransaction().begin();
Employee employee = em1.find(Employee.class, 2L);
employee.getAddress().size();
em1.getTransaction().commit();

em2.getTransaction().begin();
TypedQuery<Employee> q =
   em.createQuery("SELECT e FROM Employee e WHERE e.id=:id", Employee.class);
q.setParameter("id", 21);
Employee employee = q.getSingleResult();
employee.getAddress().size();
em2.getTransaction().commit();
```

(A) No cache used

(B) First Level Cache used

(C) Second Level Cache used

(D) None of the above

persistence.xml

```
<shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
```

Programmatic

```
Properties props = new Properties()
    .add(“javax.persistence.sharedCache.mode”, “ENABLE_SELECTIVE”);
EntityManagerFactor emf = Persistence
    .createEntityManagerFactory(“test-pu”, props);
```

JPA Cache Modes

- ALL
  - All entity data is stored in the second-level cache for this persistence unit.
- NONE
  - No data is cached in the persistence unit. The persistence provider must not cache any data.
- ENABLE_SELECTIVE
  - Enable caching for entities that have been explicitly set with the @Cacheable annotation.
- DISABLE_SELECTIVE
  - Enable caching for all entities except those that have been explicitly set with the @Cacheable(false) annotation.
- UNSPECIFIED
  - The caching behavior for the persistence unit is undefined. The persistence provider's default caching behavior will be used.

JPA Cache Modes

- ALL
  - All entity data is stored in the second-level cache for this persistence unit.
- NONE
  - No data is cached in the persistence unit. The persistence provider must not cache any data.
- ENABLE_SELECTIVE
  - Enable caching for entities that have been explicitly set with the @Cacheable annotation.
- DISABLE_SELECTIVE
  - Enable caching for all entities except those that have been explicitly set with the @Cacheable(false) annotation.
- UNSPECIFIED
  - The caching behavior for the persistence unit is undefined. The persistence provider's default caching behavior will be used.

@Cachable

```
@Entity
@Cacheable
public class Employee {
}

@Entity
@Cacheable(true)
public class Employee {
}

@Entity
@Cacheable(false)
public class Address {
}
```

@Cacheable

- @Cacheable ignored for ALL or NONE

Cache Retrieval and Store Modes

- Cache Retrieval Modes
  - javax.persistence.CacheRetrieveMode
  - USE (default)
  - BYPASS
- Cache Store Modes
  - javax.persistence.storeMode
  - USE (default)
    - the cache data is created or updated when data is read from or committed to
    - when data is already in the cache, no refresh on read
  - REFRESH
    - forced refresh on read
  - BYPASS

Cache Retrieval and Store

```
EntityManager em = ...;
em.setProperty(”javax.persistence.cache.storeMode”, “BYPASS”);

Map<String, Object> props = new HashMap<String, Object>();
props.put(”javax.persistence.cache.retrieveMode”, ”BYPASS”);
Employee employee = = em.find(Employee.class, 1L, props);
```

```
TypedQuery<Employee> q = em.createQuery(cq);
q.setHint(”javax.persistence.cache.storeMode”, ”REFRESH”);
```

Programmatic Access to Cache

```
EntityManager em = ...;
Cache cache = em.getEntityManagerFactory().getCache();
if (cache.contains(Employee.class, 1L)) {
  // the data is cached
} else {
  // the data is NOT cached
}
```

```
Cache interface methods:
boolean contains(Class cls, Object primaryKey)
void evict(Class cls)
void evict(Class cls, Object primaryKey)
void evictAll()
<T> T unwrap(Class<T> cls)
```

Forget that!

- Don’t use cache retrieval and store modes
- Don’t use programmatic access to 2LC
- Use provider-specific configuration!

Hibernate Caches

Hibernate Caches

- First Level Cache
- Second Level Cache
  - hydrated or disassembled entities: EntityEntry
  - collections
- Query Cache

Hibernate Cache Configuration

- hibernate.cache.use_second_level_cache
  - Enable or disable second level caching overall.
  - Default is true
- hibernate.cache.region.factory_class
  - Default region factory is NoCachingRegionFactory
- hibernate.cache.use_query_cache
  - Enable or disable second level caching of query results.
  - Default is false.
- hibernate.cache.query_cache_factory

Hibernate Cache Configuration

- hibernate.cache.use_minimal_puts
  - Optimizes second-level cache operations to minimize writes, at the cost of more frequent reads. Providers typically set this appropriately.
- hibernate.cache.default_cache_concurrency_strategy
  - In Hibernate second-level caching, all regions can be configured differently including the concurrency strategy to use when accessing that particular region. This setting allows to define a default strategy to be used.
  - Providers specify this setting!

Hibernate Cache Configuration

- hibernate.cache.use_structured_entries
  - If true, forces Hibernate to store data in the second-level cache in a more human-friendly format.
  - Default: false
- hibernate.cache.auto_evict_collection_cache
  - Enables or disables the automatic eviction of a bidirectional association's collection cache entry when the association is changed just from the owning side.
  - Default: false
- hibernate.cache.use_reference_entries
  - Enable direct storage of entity references into the second level cache for read-only or immutable entities.

Cache Concurrency Strategy

- Global cache concurrency strategy
  - hibernate.cache.default_cache_concurrency_strategy
- Hibernate @Cache annotation on an entity level
  - **usage**: defines the CacheConcurrencyStrategy
  - **region:** defines a cache region where entries will be stored
  - **include:** if lazy properties should be included in the second level cache. Default value is "all", so lazy properties are cacheable. The other possible value is "non-lazy", so lazy properties are not cacheable.

Cache Concurrency Strategies

- read-only
  - Application read-only data
  - Allows deletes
- read-write
  - Application updates data
  - Consistent access to a single entity, but not a serializable transaction isolation level
- nonstrict-read-write
  - Occasional stale reads
- transactional
  - Provides serializable transaction isolation level

Example – Entity and Collection Cache

```
@Entity
@Cacheable
@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Employee {
  @OneToMany(mappedBy = ”employee”, cascade = CascadeType.ALL)
  @org.hibernate.annotations.Cache(usage =
                CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
  private Set<Phone> phones = new HashSet<>();
}

// cache used!
Person person = entityManager.find(Employee.class, 1L);

// cache used!
Person person = entityManager.find(Employee.class, 1L);
person.getPhones().size();
```

Example - Query Cache

```
List<Employee> employees = entityManager.createQuery(
    "select e " +
    "from Employee e " +
    "where e.firstName = :firstName", Employee.class)
.setParameter( "firstName", "John")
.setHint("org.hibernate.cacheable", "true")
.getResultList();
```

Puzzle #2

```
EntityManager em1 = emf.createEntityManager();
em1.getTransaction().begin();
Employee employee = em1.find(Employee.class, 2L);
employee.getAddress().size();
em1.getTransaction().commit();
```

```
EntityManager em2 = emf.createEntityManager();
em2.getTransaction().begin();
Employee employee = em2.find(Employee.class, 2L);
employee.getAddress().size();
em2.getTransaction().commit();
```

find, no collection caching

	Local DB (ping: ~0.05ms)	North California (ping: ~38ms)	EU Frankfurt (ping: ~420ms)
No cache	87ms	186ms	1164ms
Cached	62ms	127ms	995ms

YONITA

find, collection caching

	Local DB (ping: ~0.05ms)	North California (ping: ~38ms)	EU Frankfurt (ping: ~420ms)
No cache	82ms	162ms	1178ms
Cached	3ms	98ms	941ms

YONITA

Puzzle #2/P6Spy

```
EntityManager em1 = emf.createEntityManager();
em1.getTransaction().begin();
Employee employee = em1.find(Employee.class, 2L);
employee.getAddress().size();
em1.getTransaction().commit();
```

```
EntityManager em2 = emf.createEntityManager();
em2.getTransaction().begin();
Employee employee = em2.find(Employee.class, 2L);
employee.getAddress().size();
em2.getTransaction().commit();
```

Available Hibernate 2nd Level Cache Implementations

- EHCache
- Infinispan
- Hazelcast

Infinispan Configuration (Local)

```
<!-- This configuration is suitable for non-clustered environments, where only single instance accesses the DB -->
 <cache-container name="SampleCacheManager" statistics="false" default-cache="the-default-cache" shutdown-hook="DEFAULT">
    <jmx duplicate-domains="true"/>

    <local-cache-configuration name="the-default-cache" statistics="false" />

    <!-- Default configuration is appropriate for entity/collection caching. -->
    <local-cache-configuration name="entity" simple-cache="true" statistics="false" statistics-available="false">
        <transaction mode="NONE" />
        <eviction max-entries="10000" strategy="LRU"/>
        <expiration max-idle="100000" interval="5000"/>
    </local-cache-configuration>

    <!-- A config appropriate for query caching. Does not replicate queries. -->
    <local-cache-configuration name="local-query" simple-cache="true" statistics="false" statistics-available="false">
        <transaction mode="NONE" />
        <eviction max-entries="10000" strategy="LRU"/>
        <expiration max-idle="100000" interval="5000"/>
    </local-cache-configuration>

    <local-cache-configuration name="timestamps" simple-cache="true" statistics="false" statistics-available="false">
        <locking concurrency-level="1000" acquire-timeout="15000"/>
        <!-- Explicitly non transactional -->
        <transaction mode="NONE"/>
        <!--  Don't ever evict modification timestamps -->
        <eviction strategy="NONE"/>
        <expiration interval="0"/>
    </local-cache-configuration>

    <!-- When providing custom configuration, always make this cache local and non-transactional.
```

Infinispan Configuration (Clustered)

```
<jgroups>
   <stack-file name="hibernate-jgroups" path="${hibernate.cache.infinispan.jgroups_cfg:default-configs/default-jgroups-tcp.xml}"/>
</jgroups>
<cache-container name="SampleCacheManager" statistics="false" default-cache="the-default-cache" shutdown-hook="DEFAULT">
   <transport stack="hibernate-jgroups" cluster="infinispan-hibernate-cluster"/>
   <jmx duplicate-domains="true"/>

   <local-cache-configuration name="the-default-cache" statistics="false" />

   <!-- Default configuration is appropriate for entity/collection caching. -->
   <invalidation-cache-configuration name="entity" mode="SYNC" remote-timeout="20000" statistics="false" statistics-available="false">
      <locking concurrency-level="1000" acquire-timeout="15000"/>
      <transaction mode="NONE" />
      <eviction max-entries="10000" strategy="LRU"/>
      <expiration max-idle="100000" interval="5000"/>
   </invalidation-cache-configuration>

   <!-- A config appropriate for query caching. Does not replicate queries. -->
   <local-cache-configuration name="local-query" statistics="false" statistics-available="false">
      <locking concurrency-level="1000" acquire-timeout="15000"/>
      <transaction mode="NONE" />
      <eviction max-entries="10000" strategy="LRU"/>
      <expiration max-idle="100000" interval="5000"/>
   </local-cache-configuration>

   <!-- A query cache that replicates queries. Replication is asynchronous. -->
   <replicated-cache-configuration name="replicated-query" mode="ASYNC" statistics="false" statistics-available="false">
      <locking concurrency-level="1000" acquire-timeout="15000"/>
      <transaction mode="NONE" />
      <eviction max-entries="10000" strategy="LRU"/>
      <expiration max-idle="100000" interval="5000"/>
   </replicated-cache-configuration>

   <!-- Optimized for timestamp caching. A clustered timestamp cache
        is required if query caching is used, even if the query cache
        itself is configured with CacheMode=LOCAL. -->
```

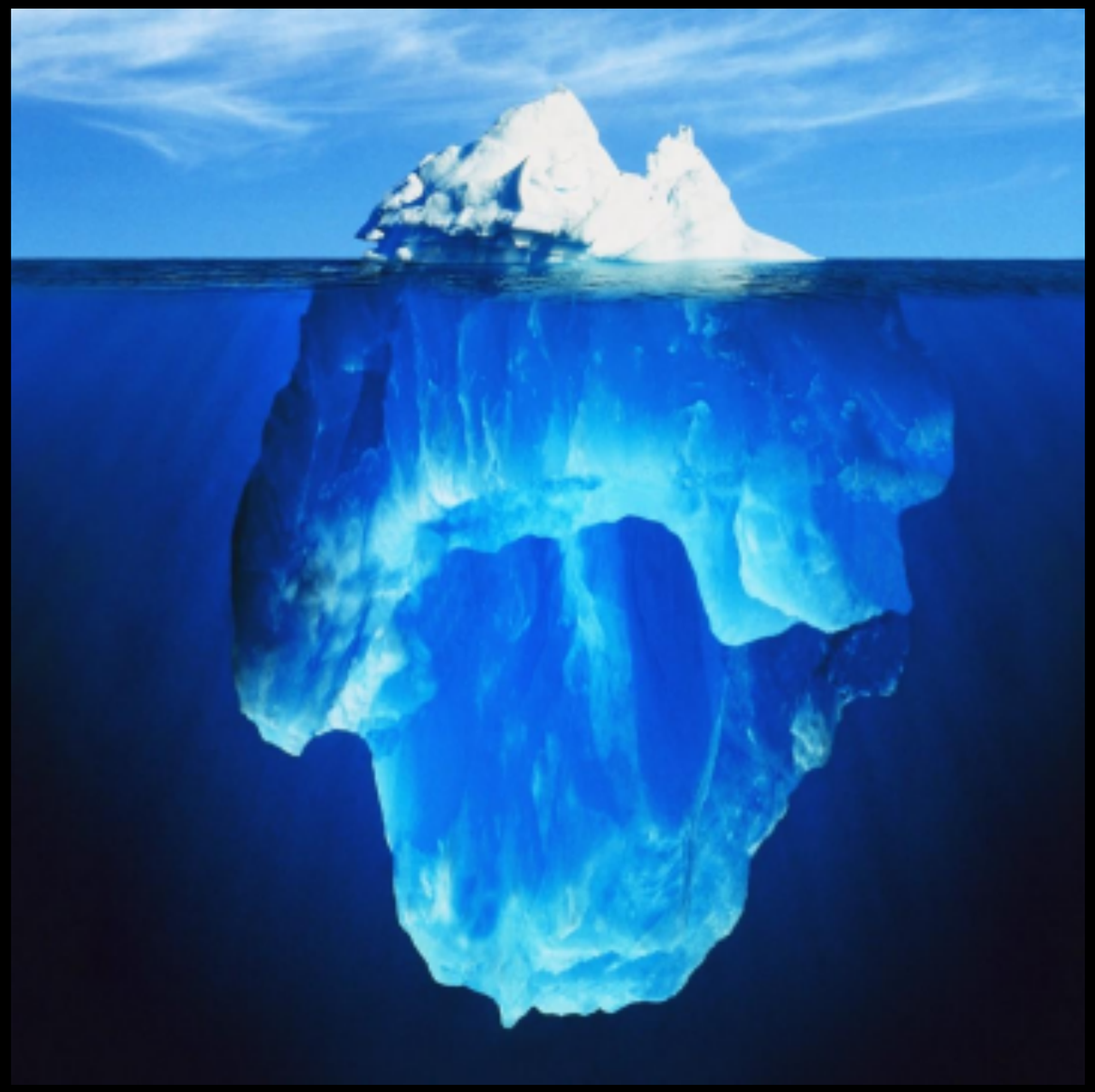
Guidelines

- Cache as much as you can
  - As much RAM you have
- Do it wisely
  - Read-only data
  - Almost read-only data
  - More reads than writes
  - Hit ratio

EclipseLink

- 2LC enabled by default
- Collection caching
- Query caching
- @org.eclipse.persistence.annotations.Cache
  - type: type of the cache (FULL, WEAK, SOFT, SOFT_WEAK, HARD_WEAK)
  - size: number of objects
  - isolation: shared, isolated, protected
  - expiry
  - expiryTimeOfDay
  - alwaysRefresh
  - refreshOnlyIfNewer
  - disableHits
  - coordinationType
  - databaseChangeNotificationType

Conclusion



YONITA

A fool with a tool is only a fool!

YONITA

Continuous Learning

YONITA

Please, vote! :)

YONITA

Q&A

- patrycja@yonita.com
- @yonlabs

YONITA