Writing Performant EJB™ Beans in the Java™ Platform, EE 5 (EJB™ 3.0) Using Annotations

Scott Oaks, Sr. Staff Engineer
Eileen Loh, Staff Engineer
Rahul Biswas, MTS

Sun Microsystems

TS-1624

Goal

Learn to make best use of EJB™ 3.0 specification features to write high performance Enterprise JavaBean™ objects

Agenda

Performance Impact of Deployment

Performance Feature of Session Beans

Performance Features of Persistent Entities

Comparative Performance Data

Conclusion

Agenda

Performance Impact of Deployment

- Developer Performance Effects

Performance Features of Session Beans

Performance Features of Persistent Entities

Comparative Performance Data

Q&A

EJB 3.0 Specification

- New modes of Session and MDB beans
- Java Persistence API replaces Entity Beans
- Lots of good sessions on how to program EJB 3.0 objects
- How EJB 3.0 specification features affect developer performance
  - Deployment annotations
  - Runtime annotations

EJB 3.0 Specification Deployment Annotations

- Deployment annotations
  - @Stateless, @Entity, @TransactionSupport…
  - Entity mappings
  - Can be overridden with deployment descriptors
    - No noticeable performance impact (YMMV)
- Deployment is faster because
  - Less XML parsing
- Deployment is slower because
  - Annotation processing, whether or not they are present

EJB 3.0 Specification Runtime Annotations

- Runtime annotations
  - @EJB, @PersistenceContext, @Resource,…
- Performance considerations
  - Lookup code
  - Stateless session usage

EJB 3.0 Specification Annotations: Lookup Code

```
@EJB MySession ses;
// Logically equivalent to:
public MySession() {
    MySession x = (MySession)
      new
  InitialContext().lookup("java:comp/env/foo");
    Field f = MySession.class.getDeclaredField("ses");
    f.set(x);
}
```

- Reflection adds some overhead (usually only once)

EJB 3.0 Specification Annotations: Session Usage

```
// EJB 2.1 pattern
public void doServletOperation(...) {
    MyStatelessSession ses = sesHome.create();
    ses.doOperation();
    ses.remove();
}
// EJB 3.0 pattern (also valid for 2.1)
@EJB MyStatelessSession ses;
public void doServletOperation(...) {
    ses.doOperation();
}
```

- We'll explore this pattern in a few examples

EJB 3.0 Specification Configuration

- Developer performance on the machine is little affected
- Think of other performance factors
  - Developer productivity
  - Data center maintenance
- How you program the beans is the key

Agenda

Performance Impact of Deployment

Performance Features of Session and MDBs

- Local vs. Remote Interfaces
- One Time Operations
- Interceptor Methods
- Transaction Management
- Transaction Attributes

Performance Features of Persistent Entities

Comparative Performance Data

Conclusion

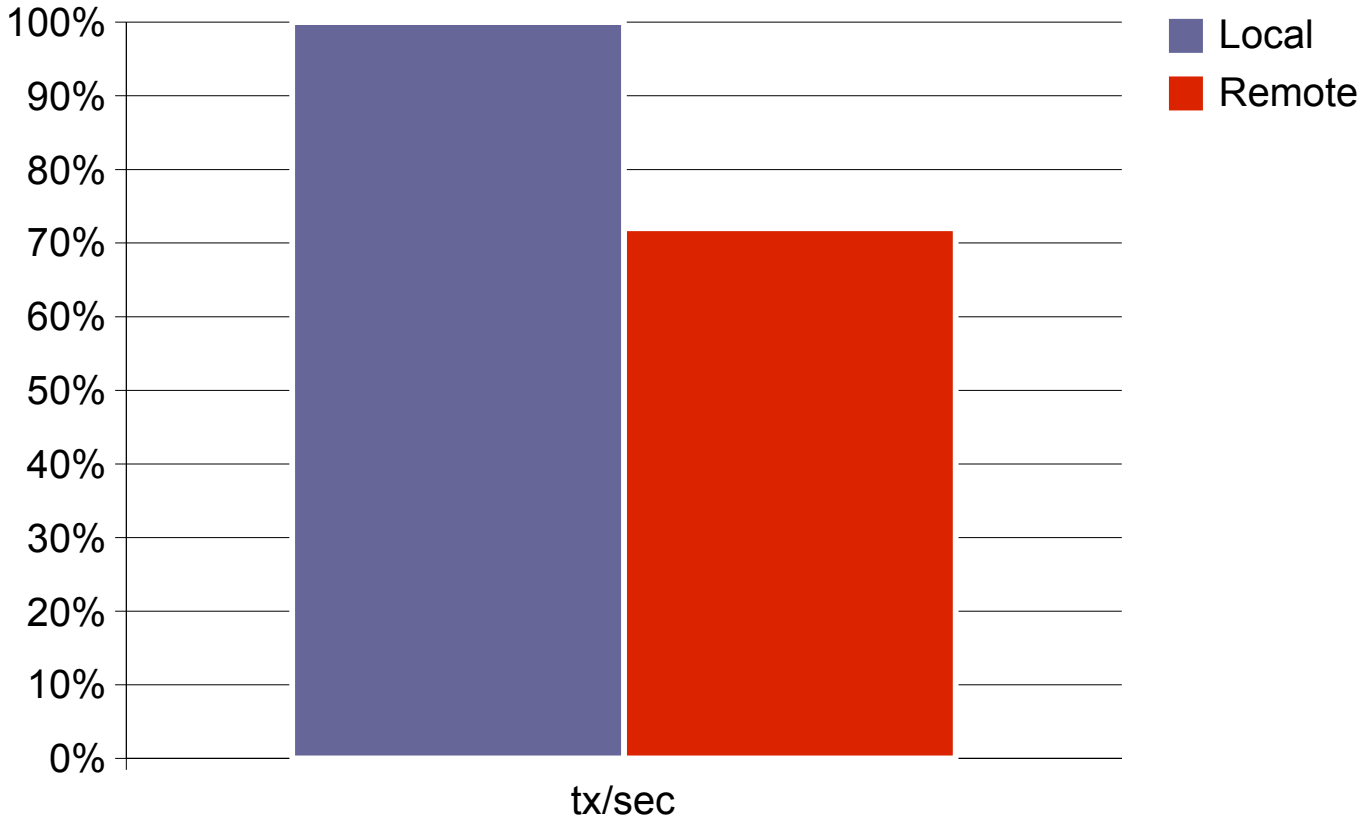
Session Beans

Local vs. Remote Interfaces

- Calls to remote interfaces can be expensive
  - Parameter copy
  - Serialization
  - Network latency
  - Client/server stack overhead
- Use coarse grained methods for remote interfaces
- Use local interfaces for better performance

Session Beans

Local vs. Remote Interface



Source: Internal benchmarks

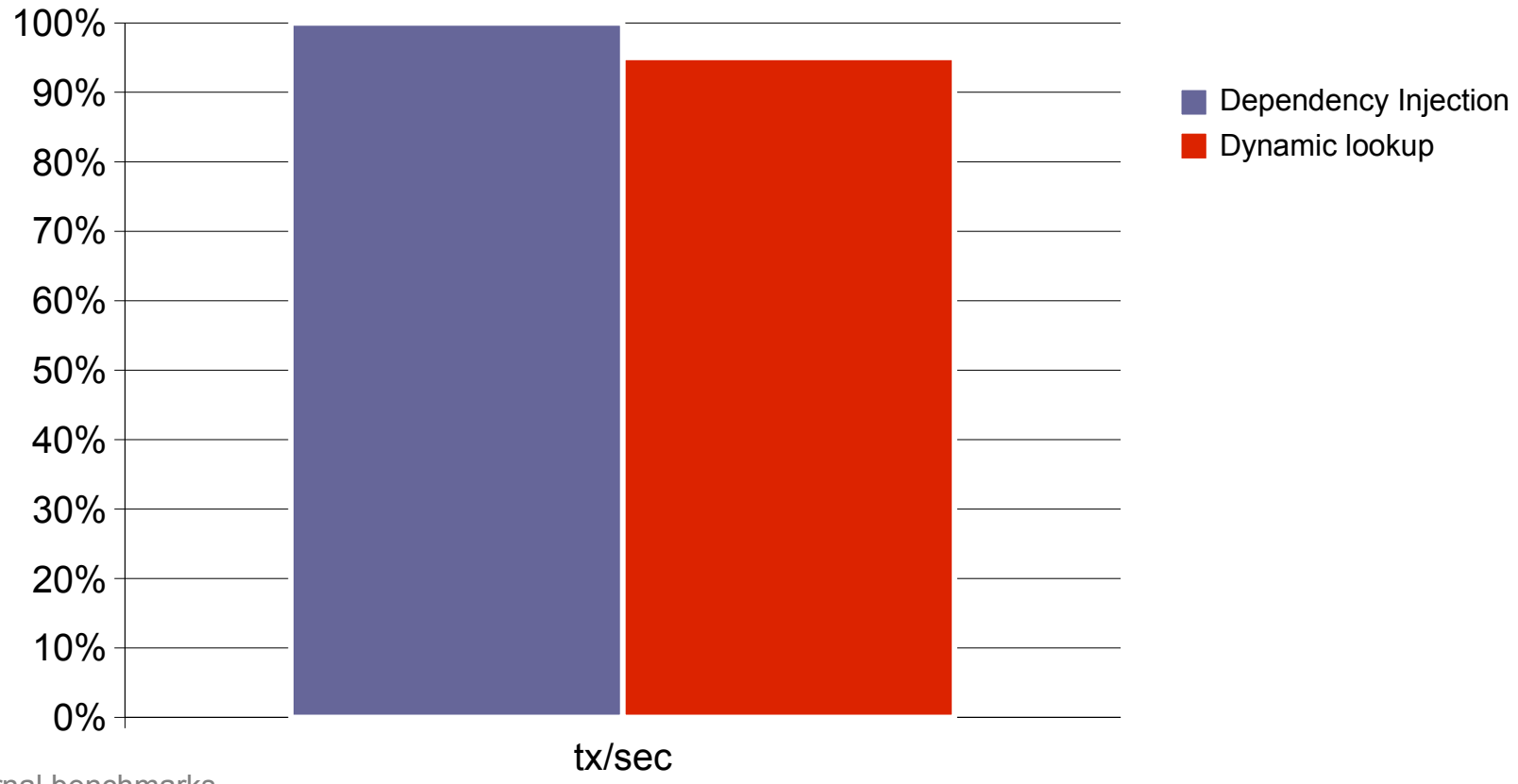
Session and MDBs

One Time Operations

- Resource lookups can incur high overhead
- Cache resources to improve performance
  - EJB object references
  - JDBC™ connections
  - Message/topic queue connections
- Dependency injections make one-time lookups easy
  - Occur after bean is created, but
  - Occur before invocation of business methods

Session and MDBs

One Time Operations



Source: Internal benchmarks

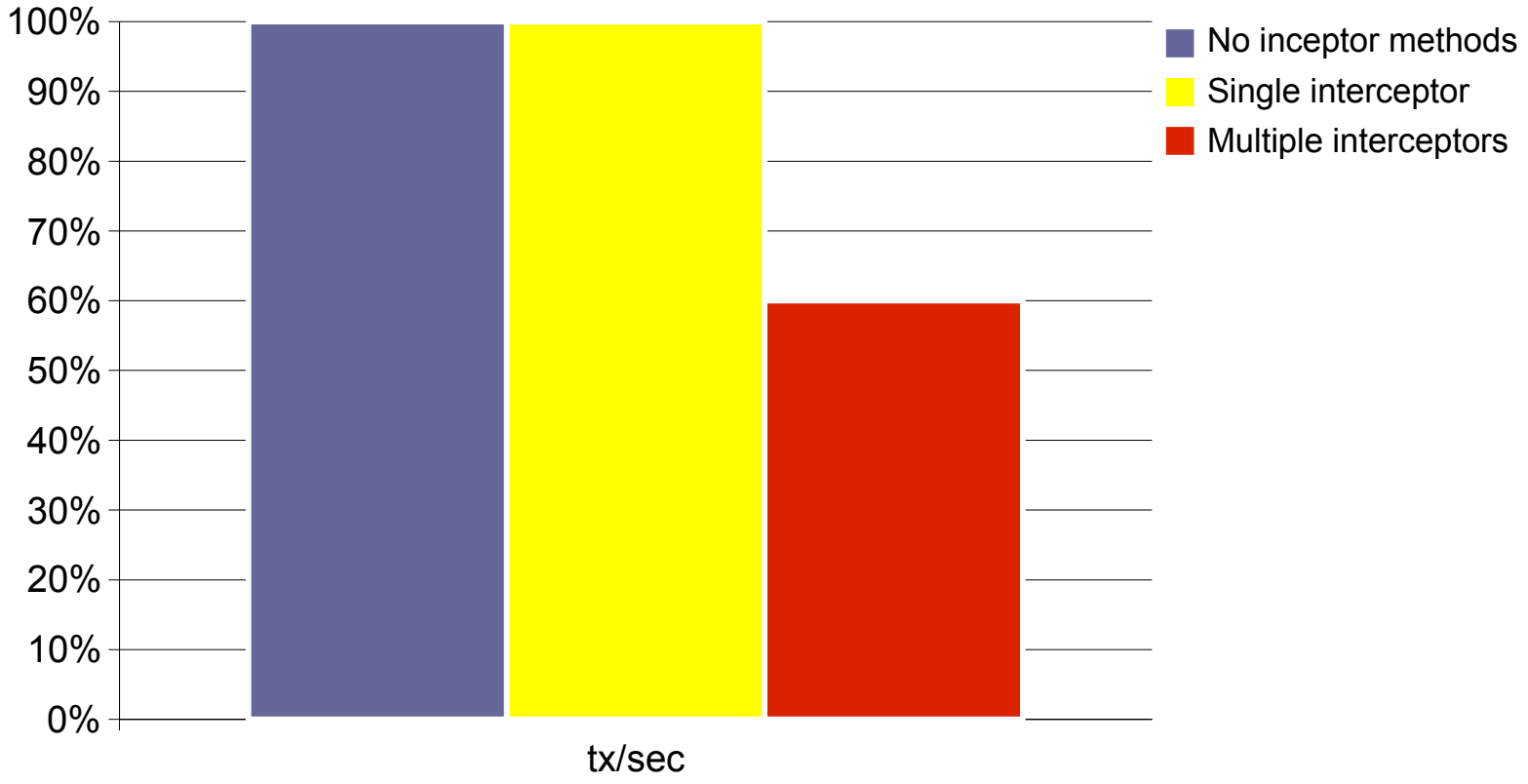
Session and MDBs

Interceptors

- Interceptor overhead
  - Deploy time cost
  - Runtime cost
- Interceptor classes
  - Creation and management overhead
  - Passivation overhead
- Be careful with multiple interceptors

Session and MDBs

Interceptors



Source: Internal benchmarks

Session and MDBs

Transaction Management Type

- Container managed (default)
  - Developer sets transaction attribute
  - Container sets transaction context
- Bean managed
  - Developer sets transaction begin and end
  - Stateful session beans—transaction context may span multiple business methods

Session and MDBs

Bean Transaction Management

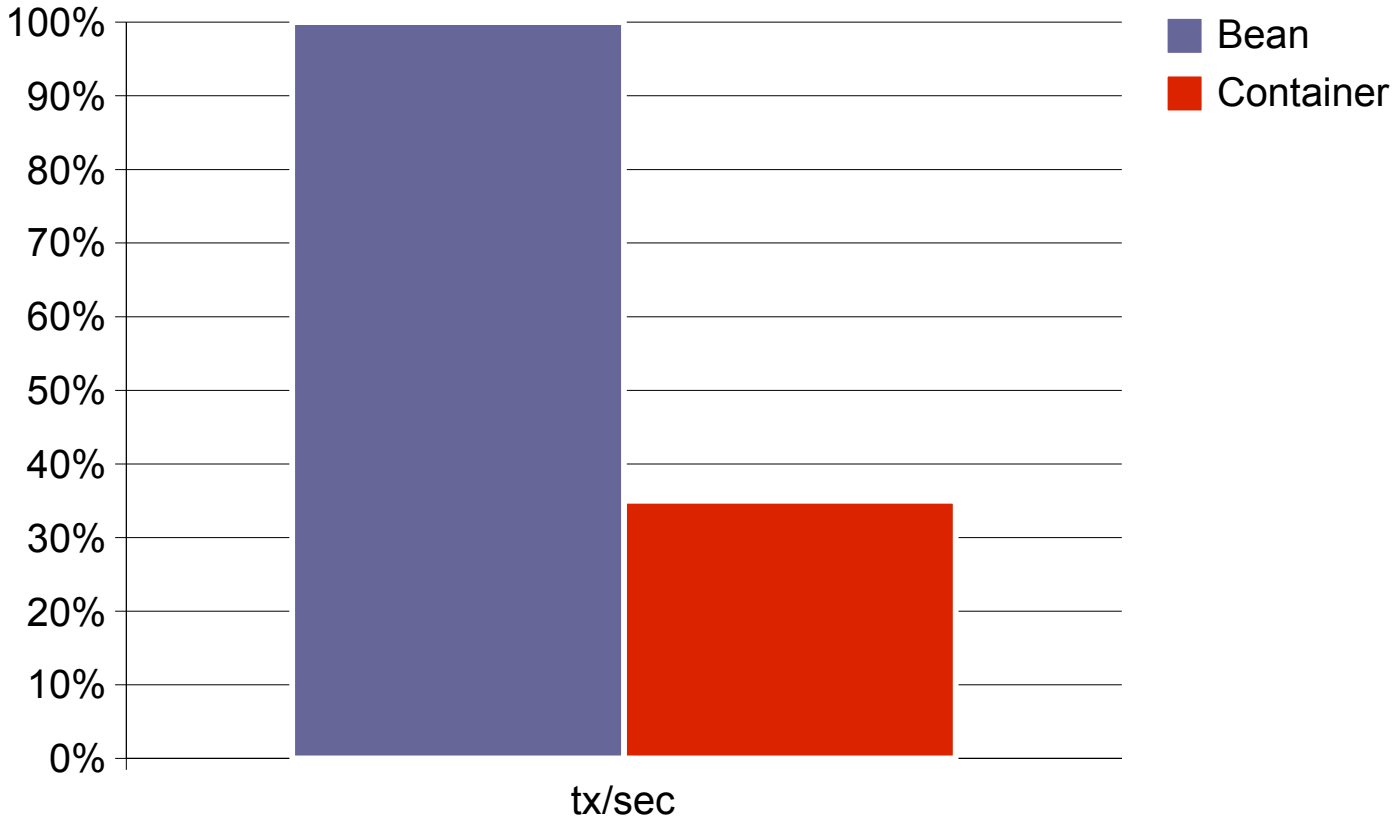
- Helps performance
  - When methods include expensive operations that don't need to be included in a transaction
  - When used to minimize number of transactions on low contention resources
- Hurts performance
  - When long transaction context includes highly contended resources

Bean Managed Example

```
@Stateful
@TransactionManagement(BEAN)
public CartSession {
    @PersistenceContext EntityManager em;
    CartEnt cart;
    @Resource UserTransaction ut;

   @PostConstruct public startCart() {
        ut.begin();
        cart = new CartEnt();
   }

   public addItem (String itemid, int qty) {
        em.persist(new CartItem(itemid, qty, cart.getId());
        cart.setItemQuantity(cart.getItemQuantity() + qty);
        em.merge(cart);
   }

   public checkOut() {
       ut.commit();
   }
}
```

Session and MDBs

Transaction Management Type



Source: Internal benchmarks

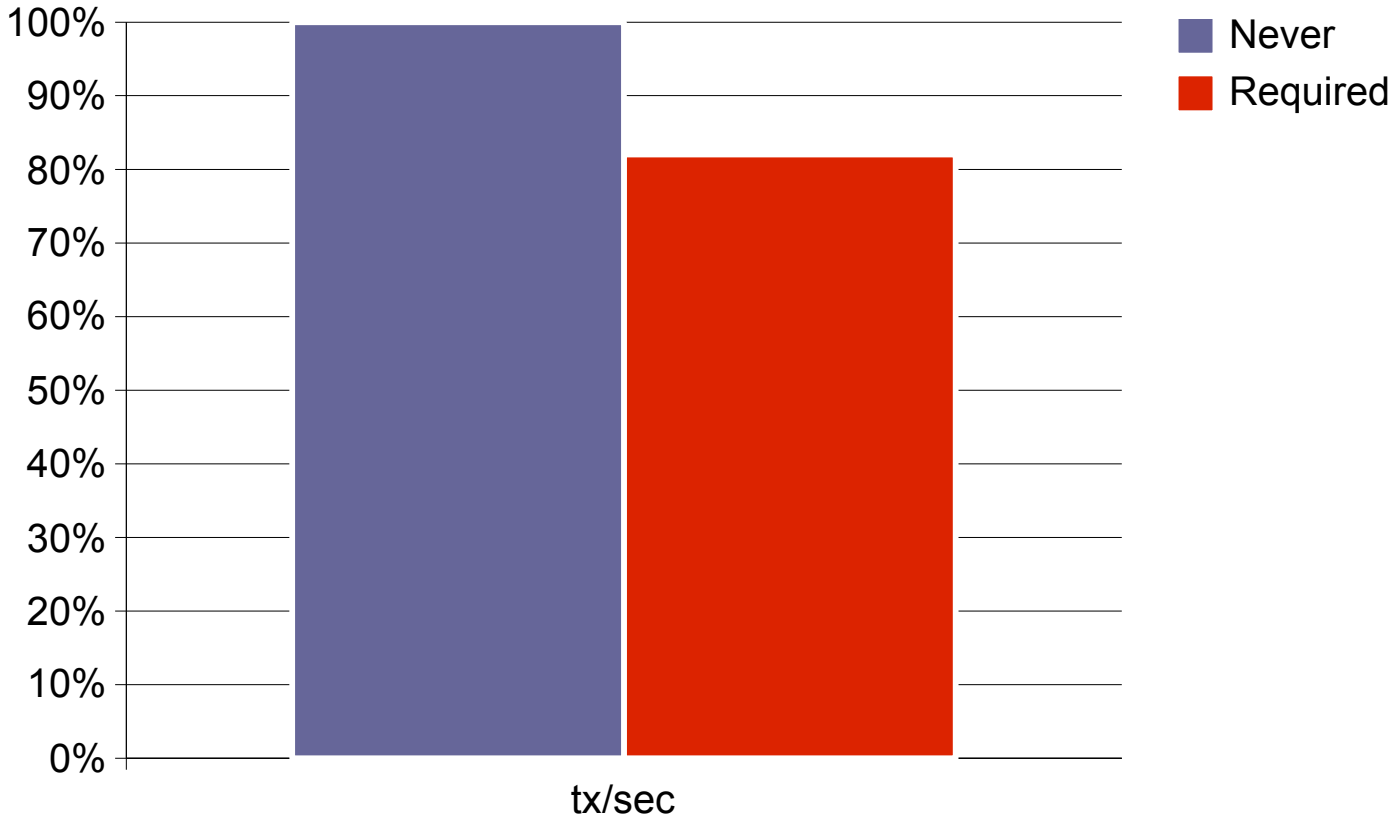
Session and MDBs

Transaction Attributes

- REQUIRED Attribute (default)
  - Business method is run in a valid transaction context
  - Container commits transaction at end of method
  - Overkill for browsing situations
- Use least restrictive level of transaction attribute per method to achieve data correctness
  - SUPPORTS
  - NOT_SUPPORTED
  - NEVER

Session and MDBs

Transaction Management Attribute



Source: Internal benchmarks

Session and MD Bean Performance

- Use remote interfaces only for coarse grained access
- Cache resources to avoid overhead of multiple lookups
- Consider the overhead of multiple interceptors and interceptor classes
- Use bean managed transactions for special cases to improve performance
- Use container managed transactions for ease of programming, but use least restrictive transaction attribute for data correctness

Agenda

Performance Impact of Deployment

Performance Feature of Session Beans

Performance Features of Persistent Entities

- Fetch Type
- Cascade
- Inheritance, Inheritance Strategy
- Flush Mode
- Optimistic Locking, Isolation Levels
- Persistence Context (Transaction vs. Extended)
- Secondary Tables

Comparative Performance Data

Conclusion

Entity Beans

FetchType

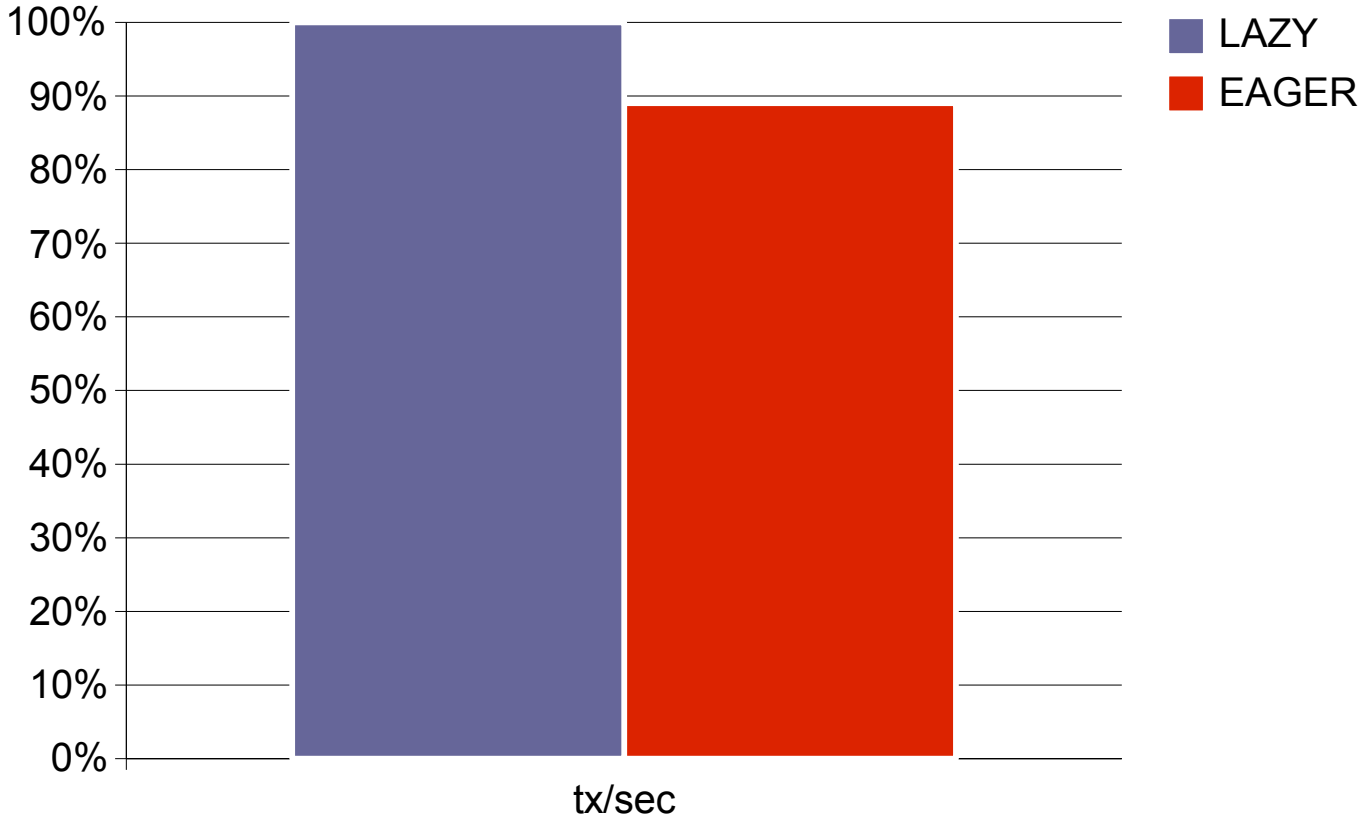
- Data fetching strategy
  - Hint when FetchType is LAZY
  - EAGER required if accessing properties outside a txn
- Used for BasicType, large objects, relationships
  - Default is EAGER except for 1:M and M:N relationships
- FetchType LAZY benefits large objects and relationships with deep hierarchies
  - If property not accessed immediately

Entities—Benchmark

```
@Entity()
public class Order {
    @OneToMany
    public Collection<OrderLineItem> getLineItems(){
       return lineItems;
    }
}

@Entity
public class OrderLineItem {
    @OneToMany
    public Collection<OrderLineItem> getLineItems(){
       return lineItems;
    }
}
```

Entities

FetchType—Relationship



Source: Internal benchmarks

Entities

CascadeType

- Specifies operations cascaded to associated entities
- Used for relationships
  - ALL, PERSIST, MERGE, REMOVE, REFRESH
  - Default is none
- If possible avoid MERGE in relationships with deep hierarchy

Entities—Don't

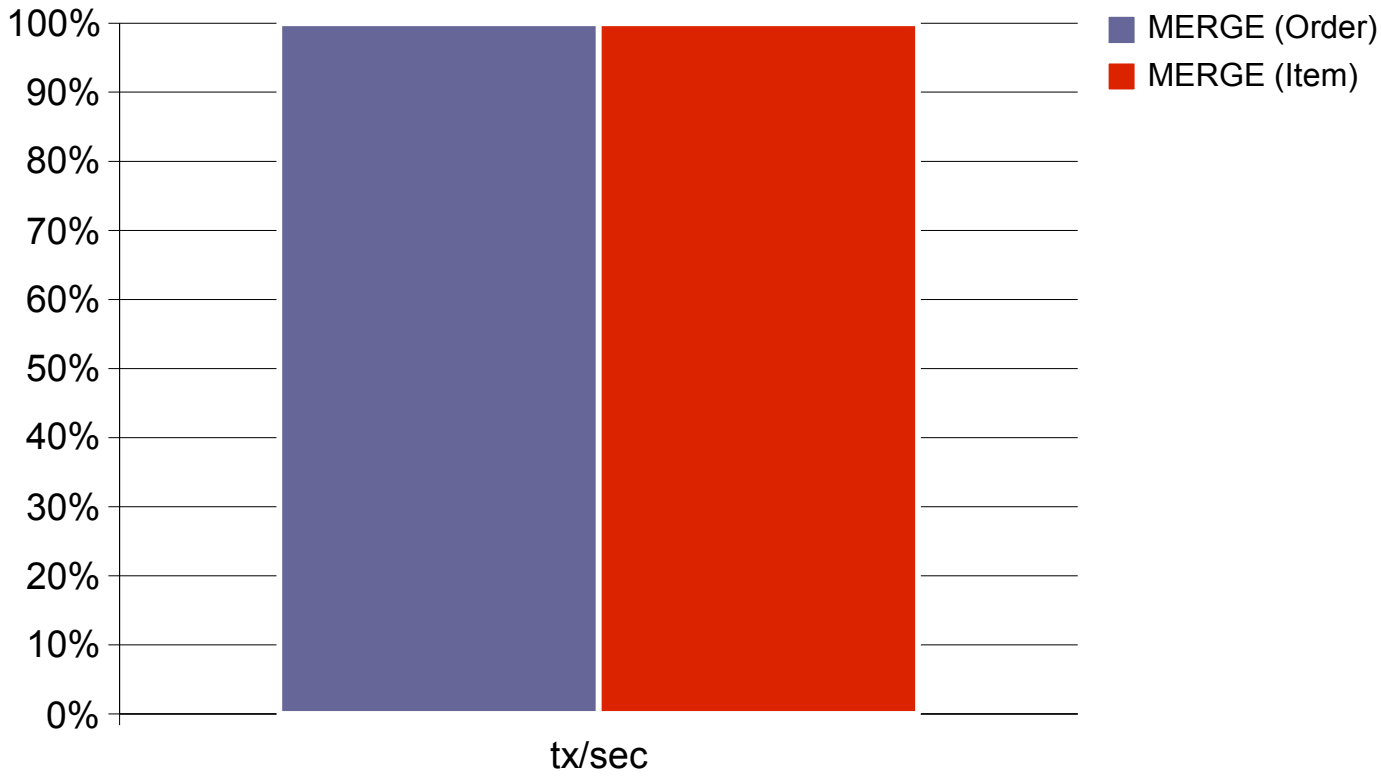
```
@Entity
public class Order {
    @OneToMany(cascade=CascadeType.ALL, ...)
    public Collection<OrderLineItem> getLineItems(){
        return lineItems;
    }
}
@Stateless
public class OrderSessionStateless {
  @PersistenceContext private EntityManager em;

  public void applyDiscount(
            Collection<OrderLineItem> lis, Order order){
    applyDiscount(lis);
    em.merge(order);
  }
}
```

Entities—Do

```
@Entity
public class Order {
    @OneToMany(cascade=CascadeType.ALL, ...)
    public Collection<OrderLineItem> getLineItems(){
        return lineItems;
    }
}
@Stateless
public class OrderSessionStateless {
  @PersistenceContext private EntityManager em;
  public void applyDiscount(
                Collection<OrderLineItem> lis){
    for(OrderLineItem li : lis) {
        applyDiscount(li); em.merge(li);
    }
  }
}
```

Entities

CascadeType



Source: Internal benchmarks

Entities

Inheritance

- Inheritance is possible for entities!
  - Inherit from other entities
  - Inherit from non-entities
    - For behavior
    - For mapping attributes
- All Java Persistence query language queries are polymorphic

Entities

Inheritance Strategy

- Single table per class hierarchy
  - Provides good support for polymorphic queries
- Single table
  - Not required to be supported
  - Need SQL unions
- Joined subclass
  - Need SQL unions

Entities

Inheritance Strategy

```
@Entity()
@Table(name="J1Order")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
    public class Order {    }

@Entity()
public class SmallOrder extends Order{    }

@Entity()
public class MediumOrder extends SmallOrder{    }

@Entity()
public class LargeOrder extends MediumOrder{    }
```

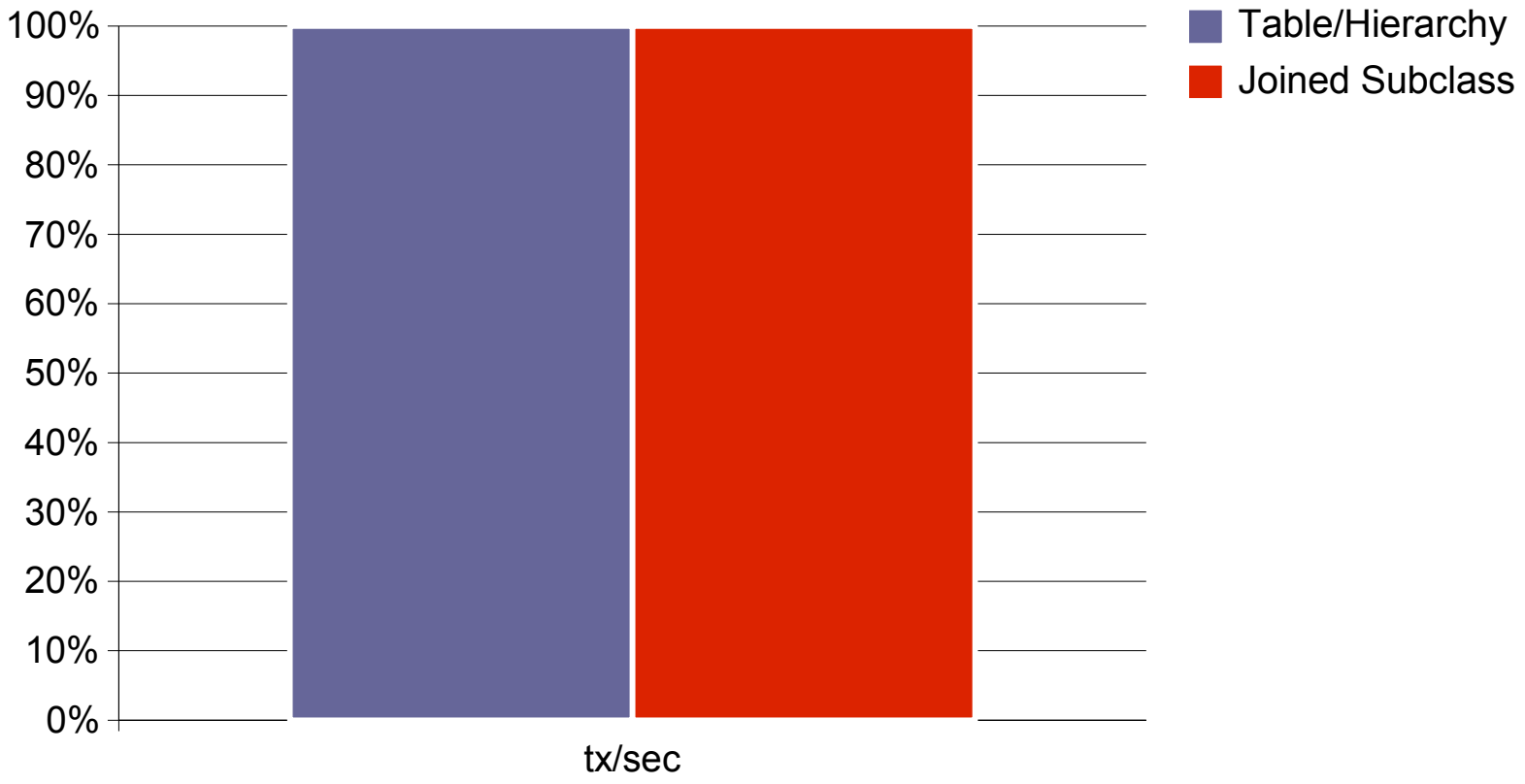
Entities

Inheritance Strategy

```
@Stateless
public class OrderSessionStateless {

  @PersistenceContext private EntityManager em;

  public void queryOrder(String orderID){
    ...
    Order order = em.find(Order.class, orderID);
    ...
  }
}
```

Entities

Inheritance Strategy



Source: Internal benchmarks

Entities

FlushMode

- Control whether state of managed entities is synchronized to the database before a query is executed
- Set on the PersistenceContext level or at a Query level
- Applicable only if transaction is active
- Possible values are AUTO, COMMIT, and NEVER
  - Default is AUTO

Entities

FlushMode—Auto

```
@NamedQuery(name="findLineItemsByOrderID",
query="SELECT OBJECT(li) from OrderLineItem li where
li.order.id=:id")
@Entity public class OrderLineItem{   }

@Stateless
public int addLineItem(String orderID,
                       OrderLineItem li){
    Order order = em.find(Order.class, orderID);
    addLineItem(order, li);
   Query q =
      em.createNamedQuery("findLineItemsByOrderID");
   q.setFlushMode(FlushModeType.AUTO);
    q.setParameter("Id", orderID);
    List list =  q.getResultList();

    return list.size();
}
```

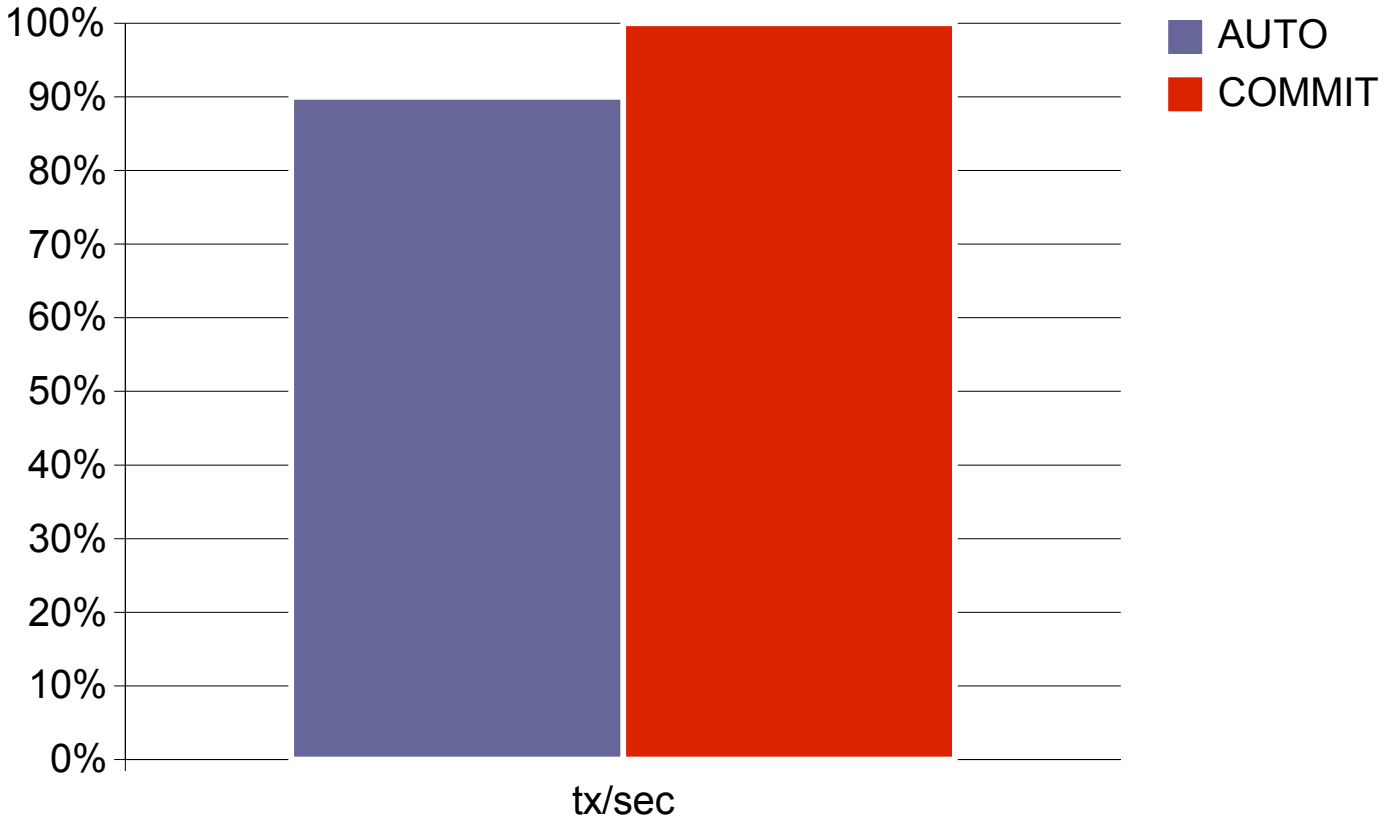
Entities

FlushMode—Commit

```
@Stateless
public void assignCustomerAndCarrier(String customerID,
                String orderID, String carrierID){

   Order order = em.find(Order.class, orderID);
   Customer customer = em.find(Customer.class,customerID);

   order.setCustomer(customer);
   customer.addOrder(order);
   Query q = em.createNamedQuery("findCarrier");
   q.setFlushMode(FlushModeType.COMMIT);
   q.setParameter("Id", carrierID);
   Carrier carrier =  (Carrier)q.getSingleResult();

   order.setCarrier(carrier);

}
```

Entities

FlushMode



Source: Internal benchmarks

Entities

Optimistic Locking, Isolation Level

- Specification assumes
  - Optimistic locking based on version consistency
  - DB access at read-committed isolation level
- Vendor specific support for pessimistic locking
- All relationships with @Version included in check
- At high concurrency, pessimistic locking may be a better option

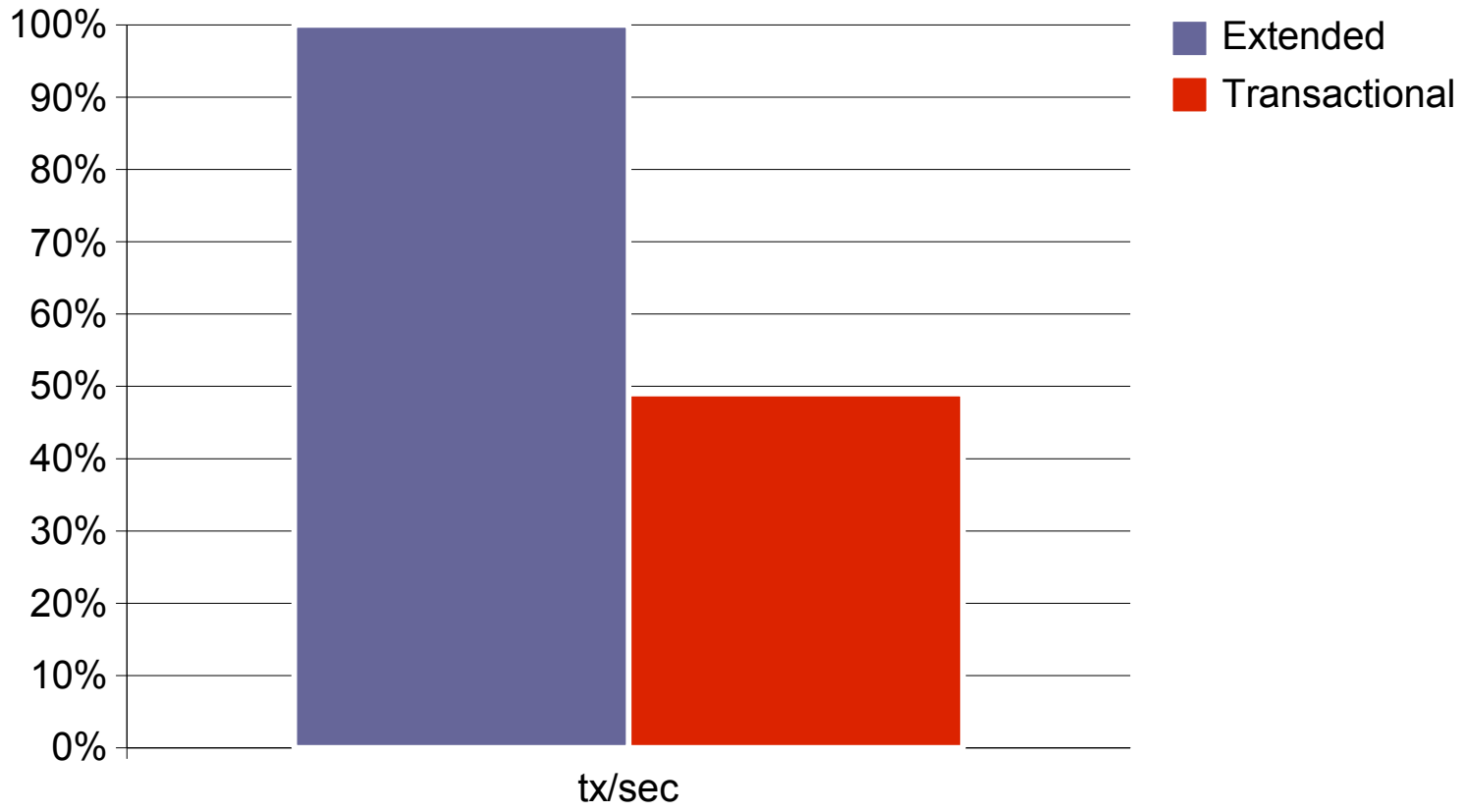
Entities

Persistence Context—Transactional vs. Extended

- Persistence Context: set of managed persistent entities
- Transactional
  - Entities detached at end of transaction
  - Stateless Session Bean
- Extended
  - Entities stay managed beyond transaction
  - Stateful Session Bean
- Does it affect performance?

Entities

Persistence Context



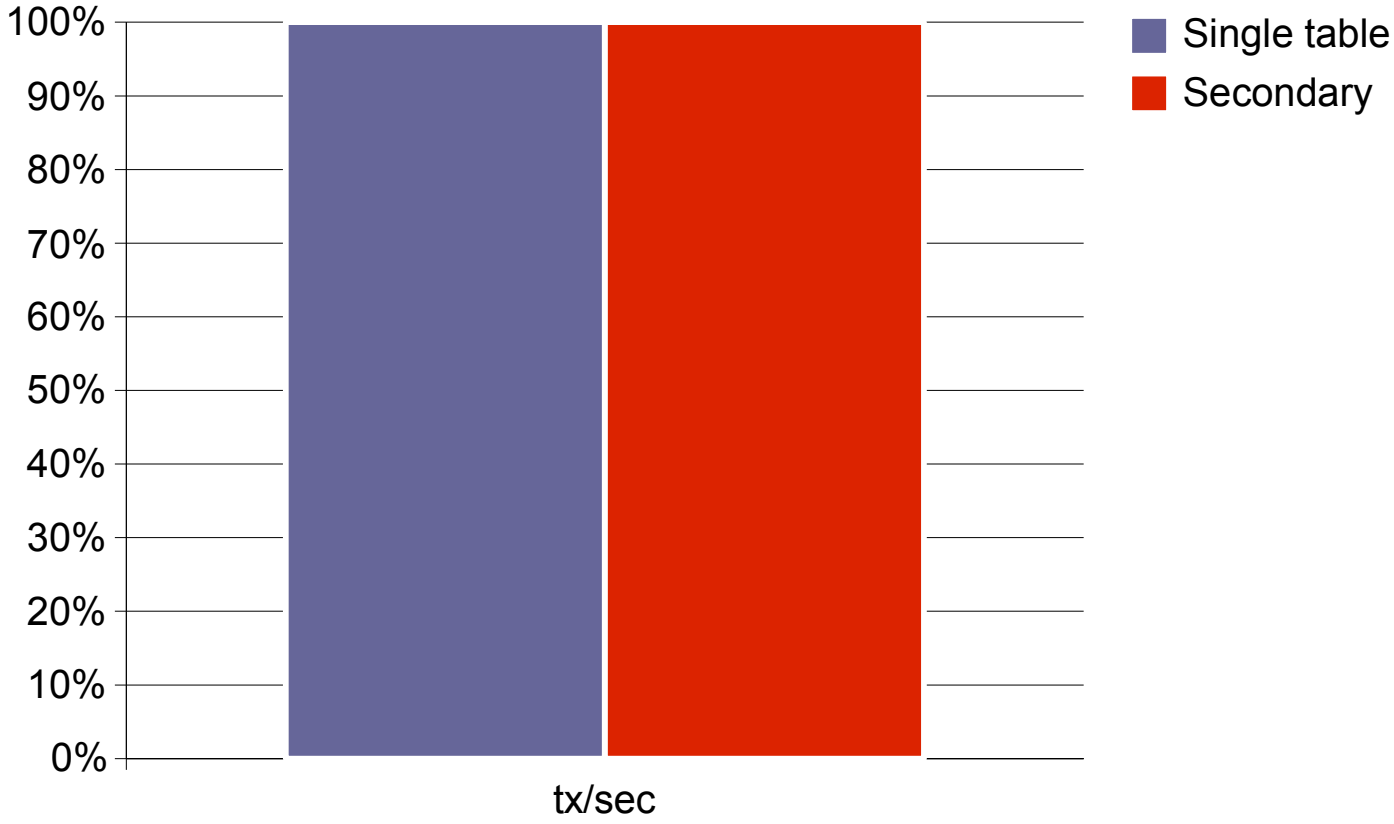
Source: Internal benchmarks

Entities

Secondary Tables

- Entity data is stored across multiple tables
- Result of extensive normalization
- Need SQL unions to retrieve data
- So avoid secondary tables?

Entities
Secondary Table



Source: Internal benchmarks

Entities

Other Performance Effects

- Caching
  - 3.0 Entities are usually cached
  - Cache size is usually limited
- Relationship join
  - Not all vendors fetch relationships in a single SQL statement
  - If a JOIN is absolutely required, use a Query

Entities: Join vs. Find

Other Performance Effects

```
//Owner and Pet have a 1-1 relationship
Owner o = em.find(Owner.class, "me");
// Two SQL statments: select * from owner
// and select * from pets


@NamedQuery(query="SELECT Owner(o) from OWNER o LEFT JOIN
  FETCH o.pet WHERE o.id = :id", name="findOwner")
Query q = em.createNamedQuery("findOwner");
query.setParameter("id", "me");
Owner o =  query.getSingleResult();
// 1 SQL statement
```

Entities

Summary

- Understand your data model
  - Use appropriate lazy/eager loading
    - Understand how properties accessed
  - Understand how best to join tables
    - Single query for eager fetch?
  - Minimize database access
    - Use cache appropriately
    - Use flush mode appropriately
  - Prepare for optimistic locking effects

Agenda

Performance Impact of Deployment

Performance Features of Session Beans

Performance Features of Persistent Entities

Comparative Performance Data

Q&A

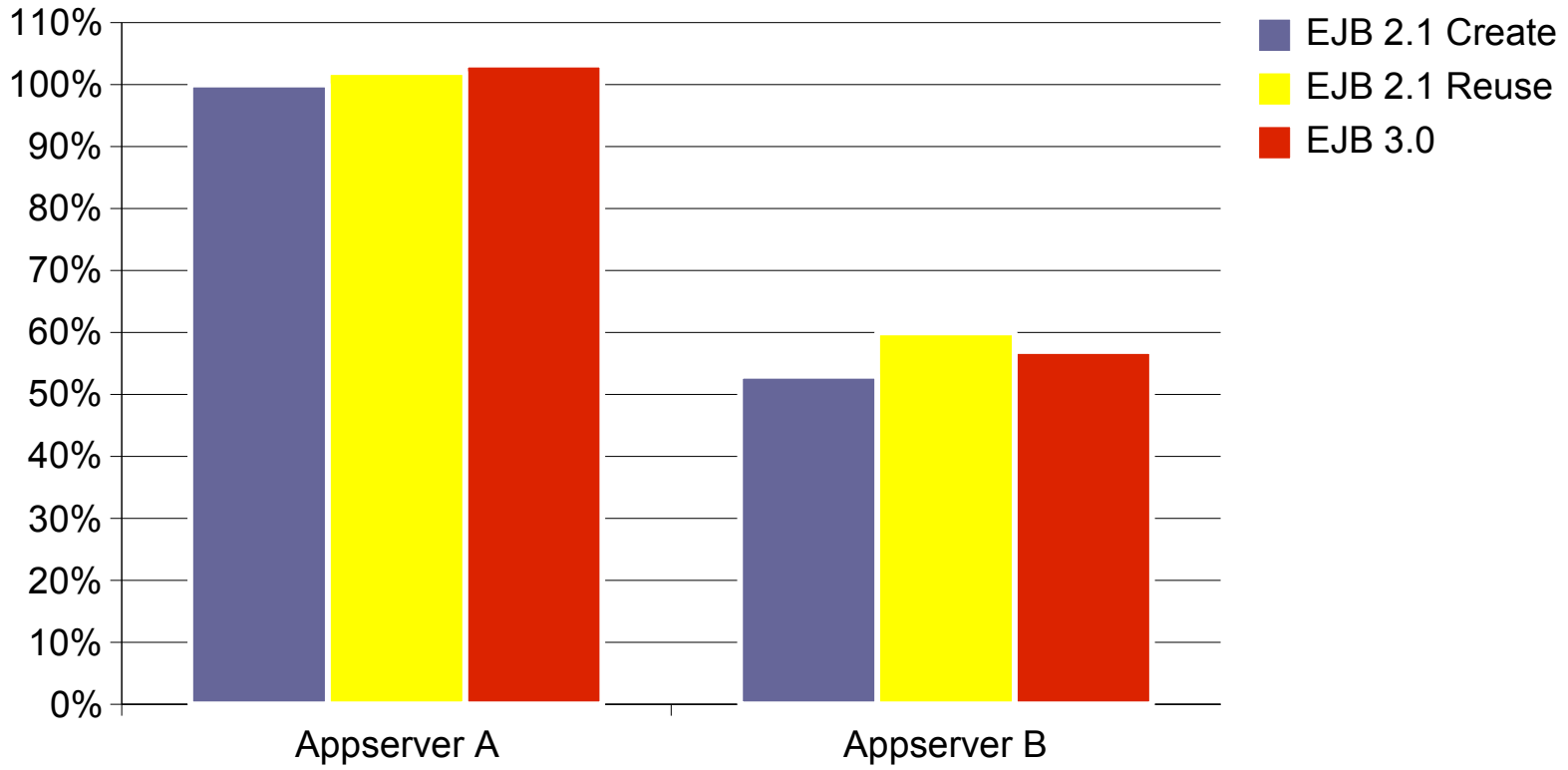
EJB 2.1 Specification vs. EJB 3.0 Specification Performance Data

- Internal microbenchmarks
  - Throughput of N users (~ 10 per CPU)
  - No business logic
  - No think time

Session Bean Microbenchmark

```
public class MyServlet extends HttpServlet {
    private MySession sess;        //2.1
    private MySessionHome sessHome;
    @EJB MySession sess30;         //3.0

    public void doGet(...) {
        if (2.1)
            if (reuseSession)
                s = sess;
            else s = sessHome.create();
        else s = sess30;
        s.doOperation();
        if (!reuseSession)
            s.remove():
    }
}
```

EJB 2.1 Specification vs. EJB 3.0 Specification Performance Data Session Beans



Source: Internal benchmarks

Entity Bean Microbenchmark

- Lookup

```
obj = em.find(MyBean.class, id);        // 3.0
obj = myHome.findByPrimaryKey(id);      // 2.1
```

- Update

```
obj.setField(obj.getField() + 1);       // both
```

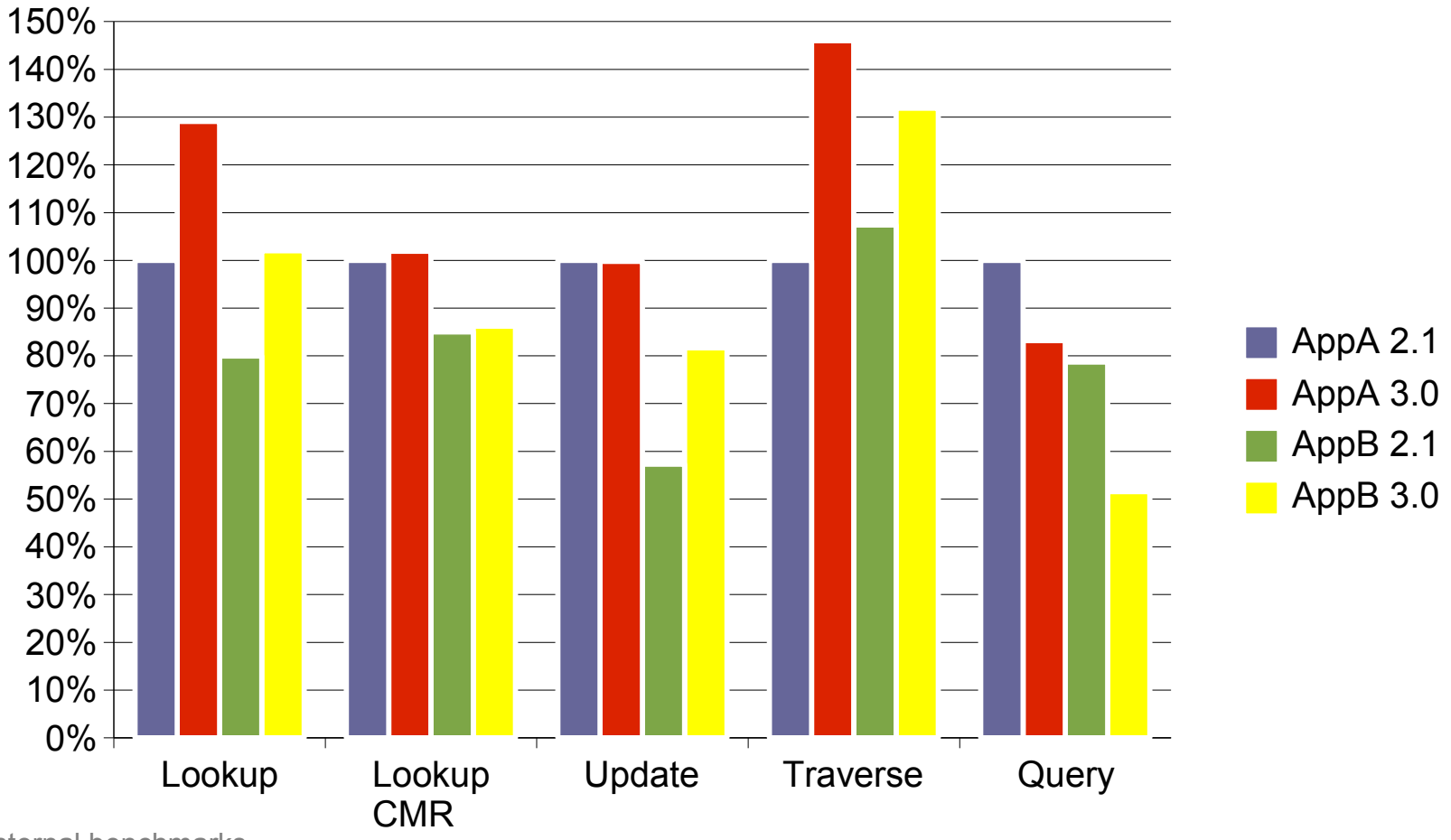
- Traverse

```
Collection c = obj.getRelatedField();  // both
```

- Query

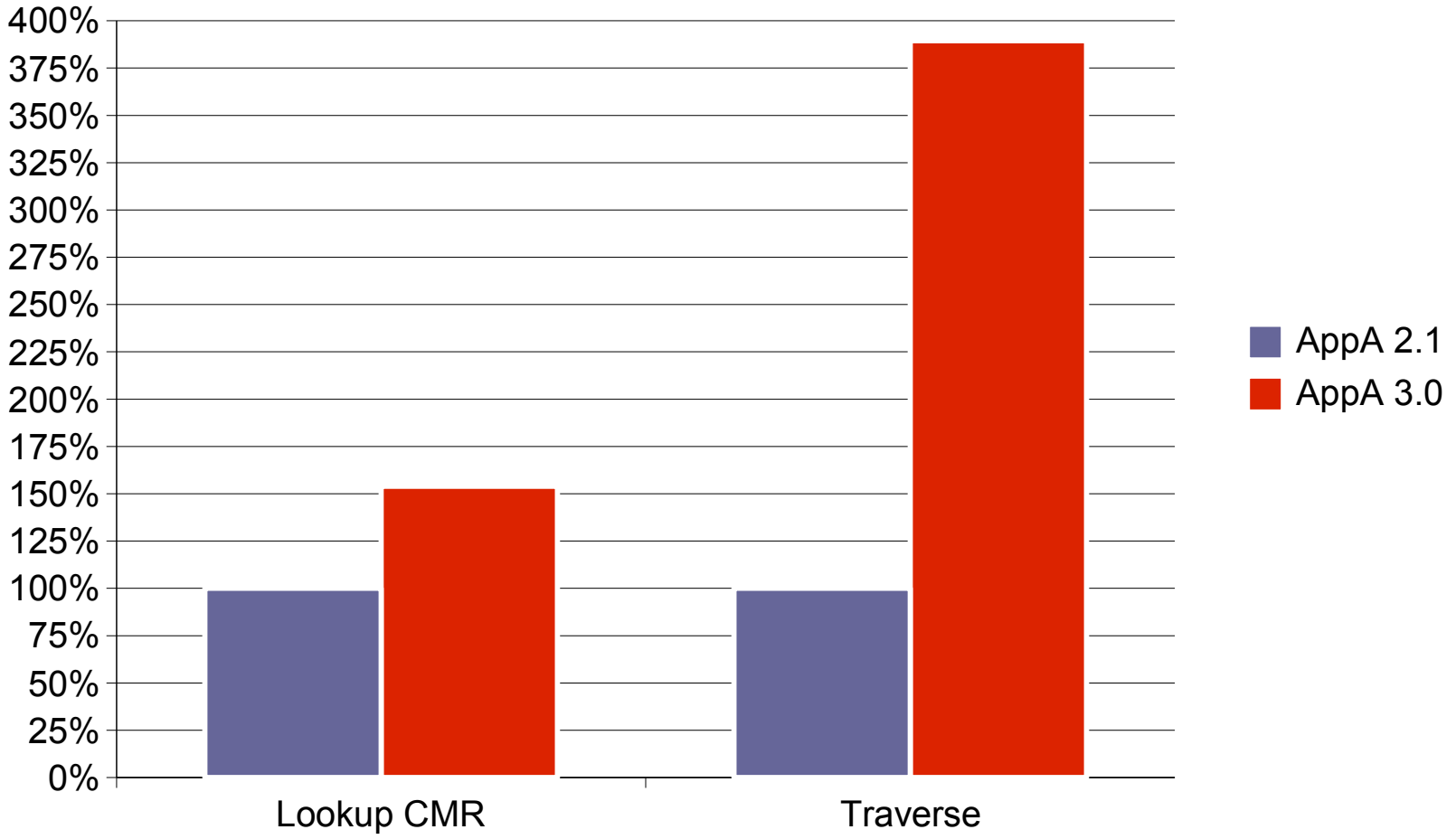
```
Query q = em.findNamedQuery();
List = q.getResultSet();              // 3.0
List = home.findByQuery(...);         // 2.1
```

EJB 2.1 Specification vs EJB 3.0 Specification Performance: Entities



Source: Internal benchmarks

EJB 2.1 Specification vs EJB 3.0 Specification Performance: Cached Entities



Source: Internal benchmarks

Performance Conclusions

- Still early
- Lots of new features enabled by 3.0
  - Cached entities
- Vendor independence lets you be nimble in the performance arena

Q&A

http://performance.dev.java.net/

Writing Performant EJB™ Beans in the Java™ Platform, EE 5 (EJB™ 3.0) Using Annotations

Scott Oaks, Sr. Staff Engineer
Eileen Loh, Staff Engineer
Rahul Biswas, MTS

Sun Microsystems

TS-1624