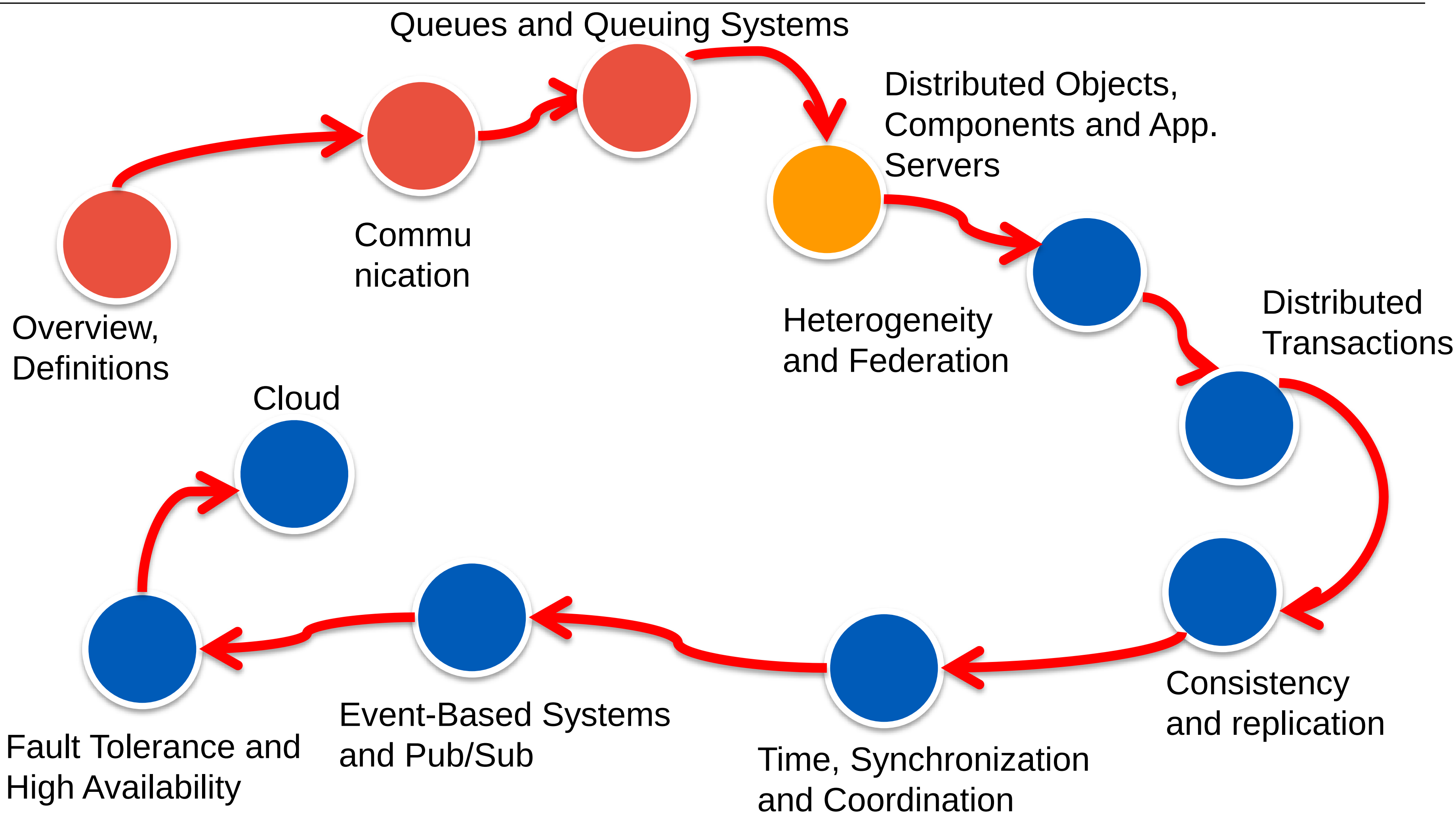
Middleware

4. Distributed Objects and Components, Application Servers, ORM

I. Petrov, A. Buchmann

Wintersemester 2011/2012

Topics


Queues and Queuing Systems
Distributed Objects, Components and App. Servers
Commu nication
Overview, Definitions
Heterogeneity and Federation
Distributed Transactions
Cloud
Consistency and replication
Fault Tolerance and High Availability
Event-Based Systems and Pub/Sub
Time, Synchronization and Coordination

Topics

- CORBA and Distributed Objects
- Components and Containers
- Application Servers
- ORM – Object Relational Mappers

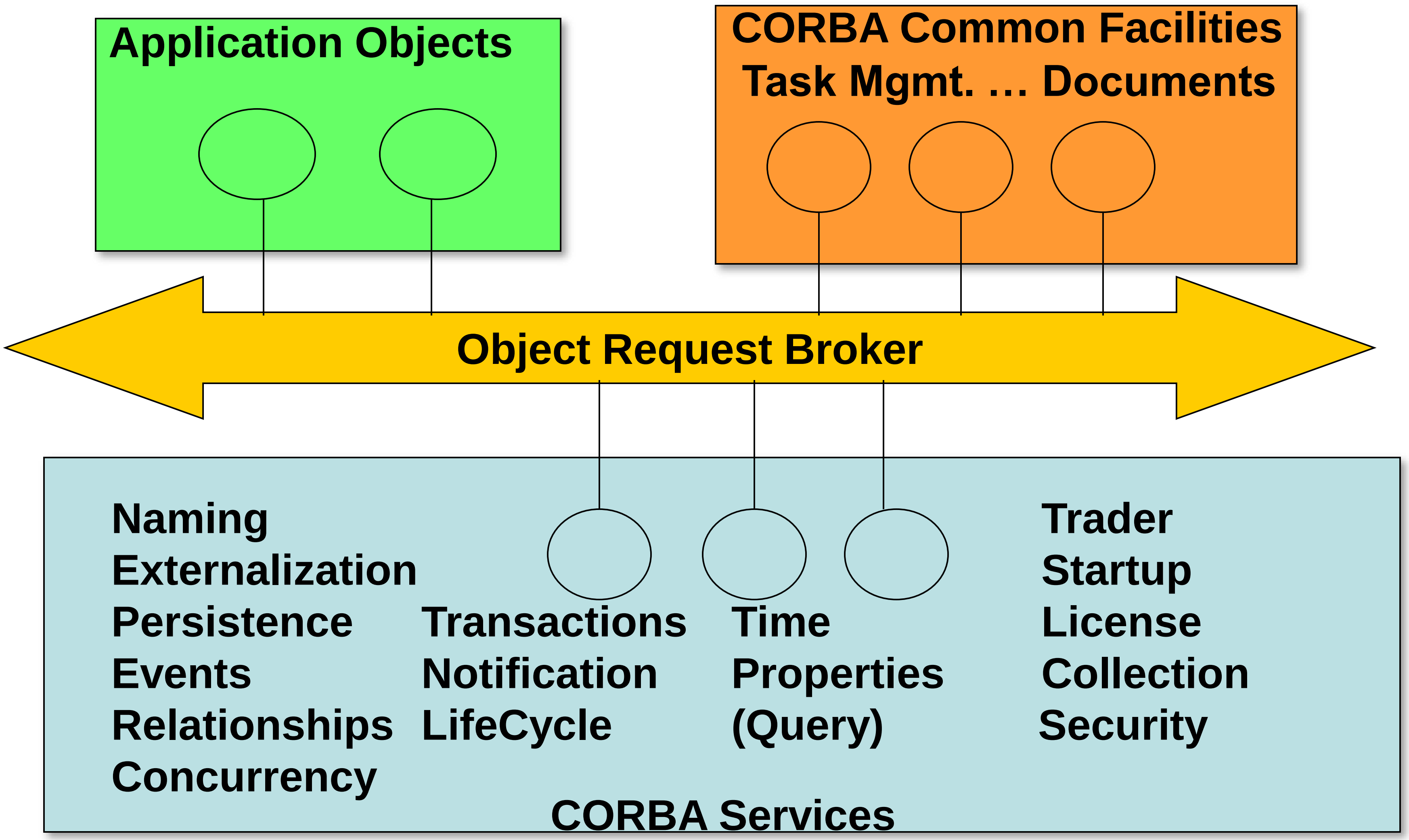
Reading for THIS Lecture

- The slides for the lecture are based on material from:
  - Philip Lewis, Arthur Bernstein, Michael Kifer. 2001.
    Databases and Transaction Processing: An Application-Oriented Approach (1st ed.). Addison-Wesley
    - Chapter 21
  - Orfali, Harkey and Edwards
    CORBA Survival Guide, 3rd Edition, J. Wiley. (dated)
  - http://www.omg.org/gettingstarted/corbafaq.htm (current CORBA FAQ maintained by OMG)
  - R. Monson-Haefel
    Enterprise JavaBeans 3.0, O'Reilly, June 2008.

CORBA

Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

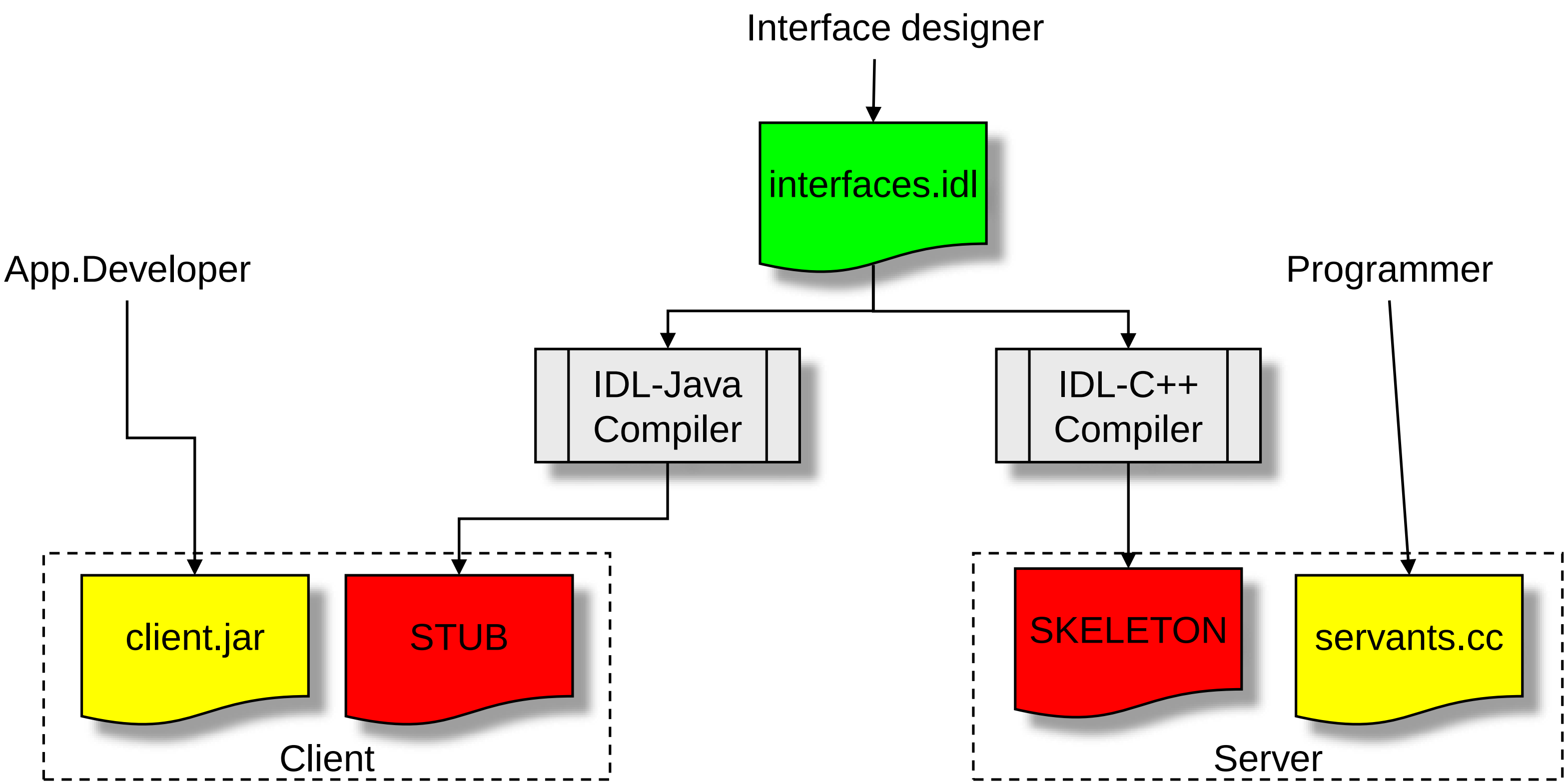
OMG Object Management Arch.



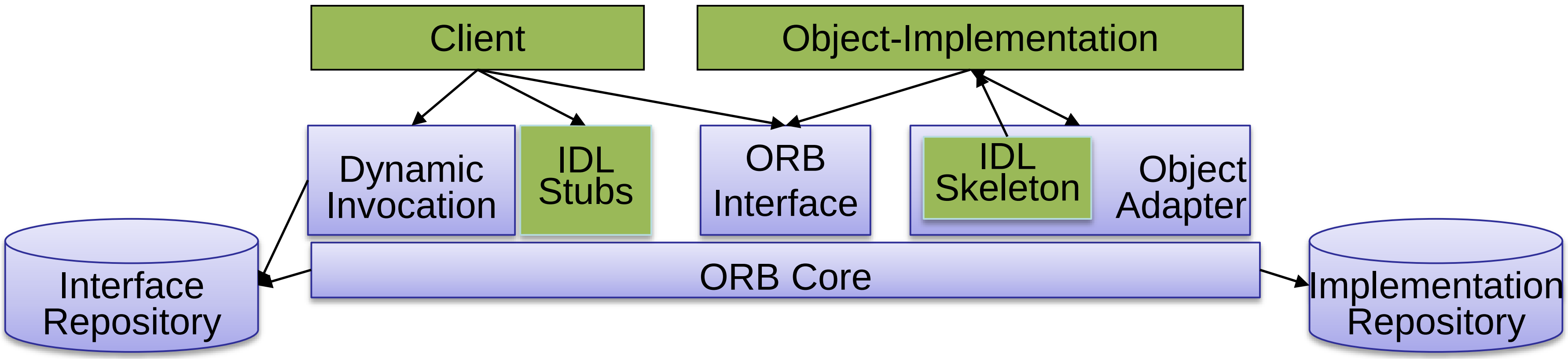
IDL

- Declarative language used to define objects and their contractual interfaces
- IDL provides OS- and PL-independent interfaces to all the services on the CORBA bus
- IDL only for specification of attributes, parent classes, methods and events supported, and exceptions raised, no implementation
- Implementations in any language for which bindings exist (C, C++, Ada, Smalltalk, COBOL, Java)
- To request a service from another object, an object must know the target object's supported interface
- Interface Repository contains the metadata needed to discover other components at run time

Client and Server-side impl.



Simplified CORBA Architecture



Object Request Broker (ORB)

- ORB functions as an object bus
- Static (compile-time) and dynamic (run-time) method invocation
- High-level language bindings: standard interfaces and language-independent types allow invocation of a service independent of the language it is written in
- Self-describing system through metadata for interface definitions, either IDL precompiler or directly from OO-PL compiler (Java bytecode)
- Local/remote transparency: ORB can function on stand-alone machine or interoperate with other ORBs (via IIOP - Internet Inter-ORB Protocol)
  - while user doesn't need to be concerned with location of an object, price is in the performance
- Support for security and transactions
- Polymorphic messaging (target specific)
- Coexistence with legacy systems through encapsulation of legacy code with IDL interface

CORBA Services

- Life Cycle Service
  - defines operations for creating, copying, moving and deleting components on the bus → relationship service
- Persistent State Service
  - provides single interface for storing objects persistently on various storage servers (OODBMSs, RDBMSs, files)
- Event Service
  - allows components to register/unregister interest in specific events. Event channel collects and distributes events among objects
- Naming Service
  - allows components on the bus to locate other components by name. Also allows objects to be bound to existing network directories or naming contexts (ISO X.500, OSF DCE, Sun NIS+, Internet LDAP).
- Concurrency Control Service
  - provides a lock manager that can obtain locks on behalf of transactions or threads

CORBA Services (cont.)

- Transaction Service
  - provides two-phase commit coordination among recoverable components using either flat or nested transactions
- Relationship Service
  - provides a way for creating dynamic links among components and to traverse them. May be used for enforcing referential integrity, containment, ownership etc.
- Externalization Service
  - provides a standard way for getting data into and out of a component using a stream-like mechanism
- Query Service
  - provides query operations on objects. Based on SQL:1999 and ODMG's OQL
- Licensing Service
  - provides metering of a component's use for fair charging (per session, per node, per site, …)

CORBA Services (cont.)

- Properties Service
  - associates named values (properties) with any component.
- Time Service
  - provides interfaces for synchronizing time in a distributed object environment and for defining and managing time-triggered events
- Security Service
  - provides framework for distributed object security (authentication, access control, confidentiality and non-repudiation, delegation of credentials)
- Trader Service
  - provides “yellow pages” allowing objects to publizise their services and bid for jobs
- Collection Service
  - provides interfaces to create and manipulate collections

Reference to services

http://www.dcl.hpi.uni-potsdam.de/LV/Components04/VL3/03a_corba-services.pdf

This is a somewhat more detailed set of slides from the HPI in Potsdam

CORBA 1.0

- CORBA 1.0 specified minimal ORB functionality:
  - Basic ORB
  - IR (Interface Repository, BOA (Basic Object Adaptor)
  - C Bindings
  - Naming
  - Events
  - Life Cycle
  - Persistence

CORBA 2.0

TECHNISCHE UNIVERSITÄT DARMSTADT

- CORBA 2.0 provides for interoperability among ORBs
  - IIOP
  - Federated IR
  - C++ bindings
  - Transactions
  - Security
  - Collections
  - Time

  Licensing
  Compound Documents
  Trader
  Concurrency
  Externalization
  Relationships
  Query

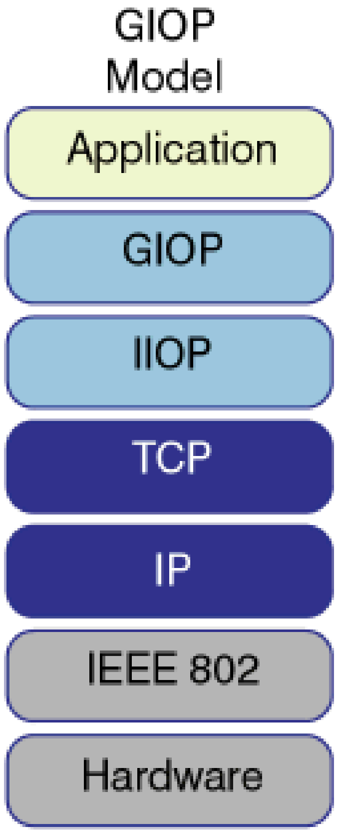
CORBA 3.0

- CORBA 3.0 addresses the issues of portability, vertical integration, life in the object-Web
  - Messaging (MOM)
  - Multiple Interfaces
  - Business Objects/Java Beans
  - Objects-by-Value
  - CORBA/DCOM
  - IIOP Firewall Support
  - Domain-Level Frameworks

Server Portability (POA)
Java Bindings

Mobile Agents
Automatic Persistence
Workflow

Inter-ORB Architecture: GIOP

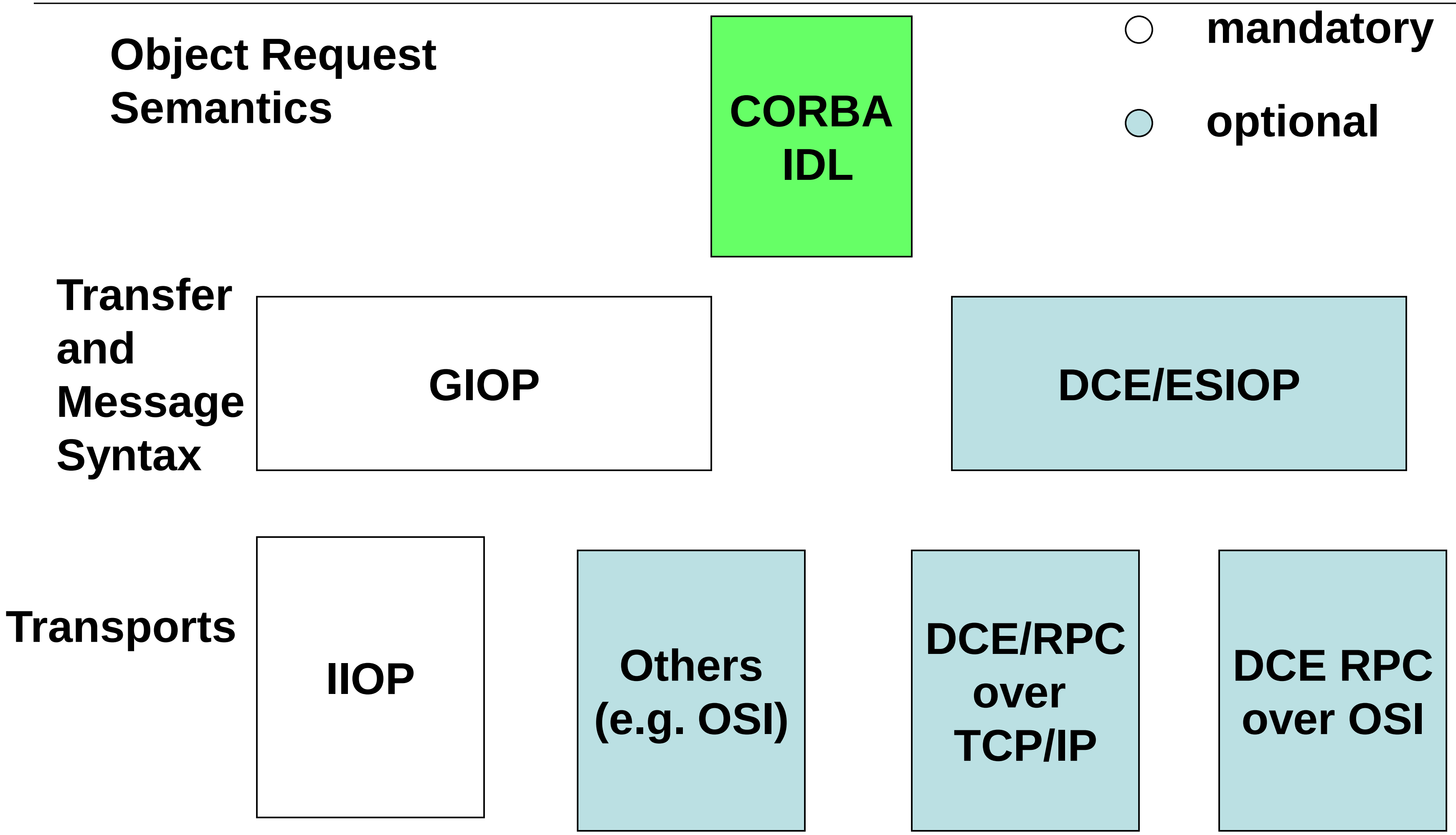


- General Inter-ORB Protocol (GIOP)
  - set of message formats and common data representations for inter-ORB communication
  - works on top of any connection-oriented protocol
  - defines seven message formats that cover all the ORB request/reply semantics
  - no format negotiations are needed
  - Common Data Representation (CDR) maps data defined in IDL to flat message representation
  - CDR takes care of inter-platform issues

Inter-ORB Architecture: IIOP

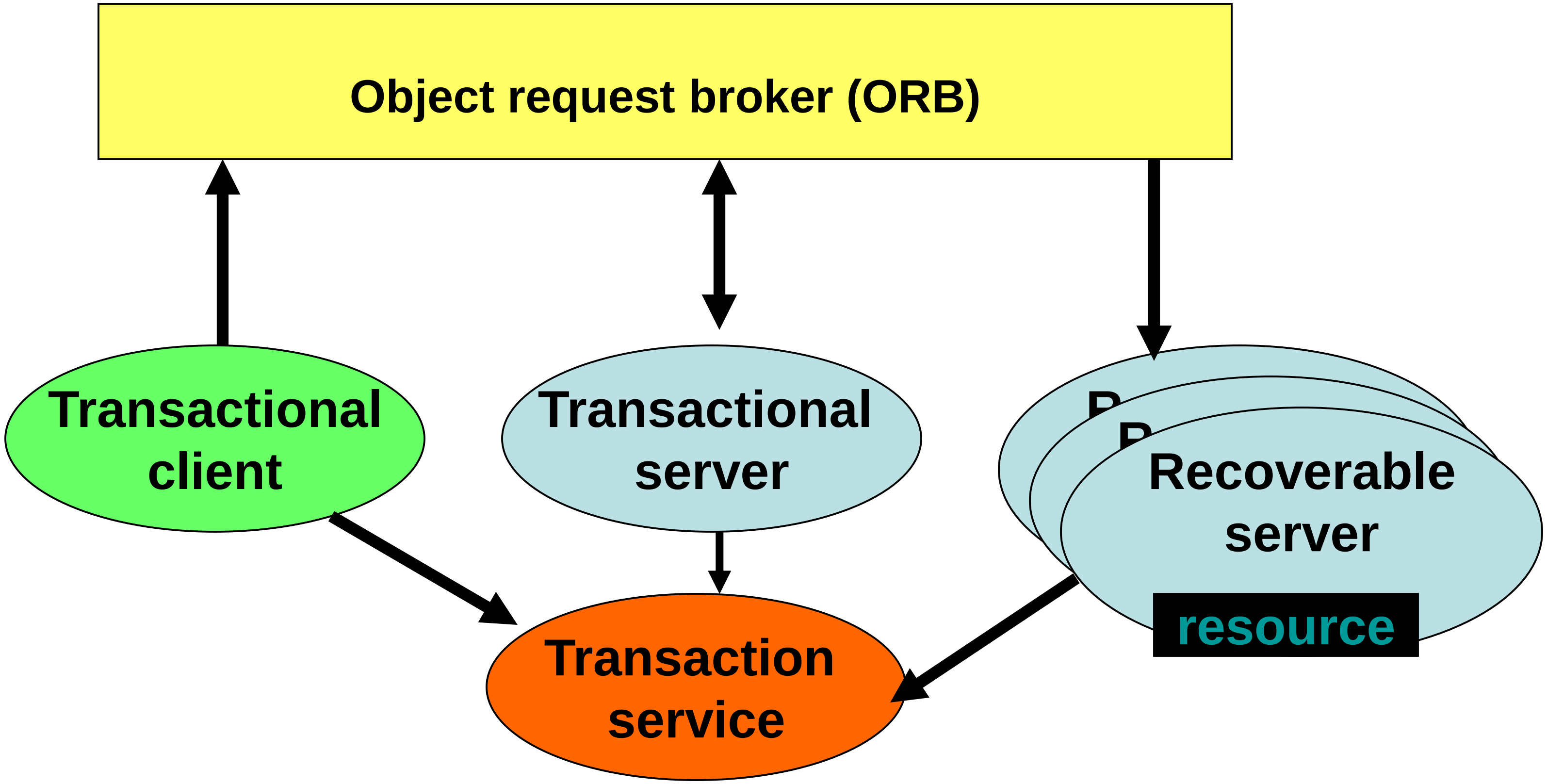
- Internet Inter-ORB Protocol (IIOP)
  - specifies how messages are exchanged over TCP/IP network
  - makes it possible to use Internet as ORB to bridge among other ORBs
  - GIOP may be mapped down to IIOP
  - IIOP provides the mechanisms to transmit implicitly the necessary context information for transaction and security services

CORBA 2.0 Inter-ORB Arch.



OTS

- Object Transaction Service specification defines transactional service based on CORBA for OO programming environment
- Specified by OMG (Object Management Group)



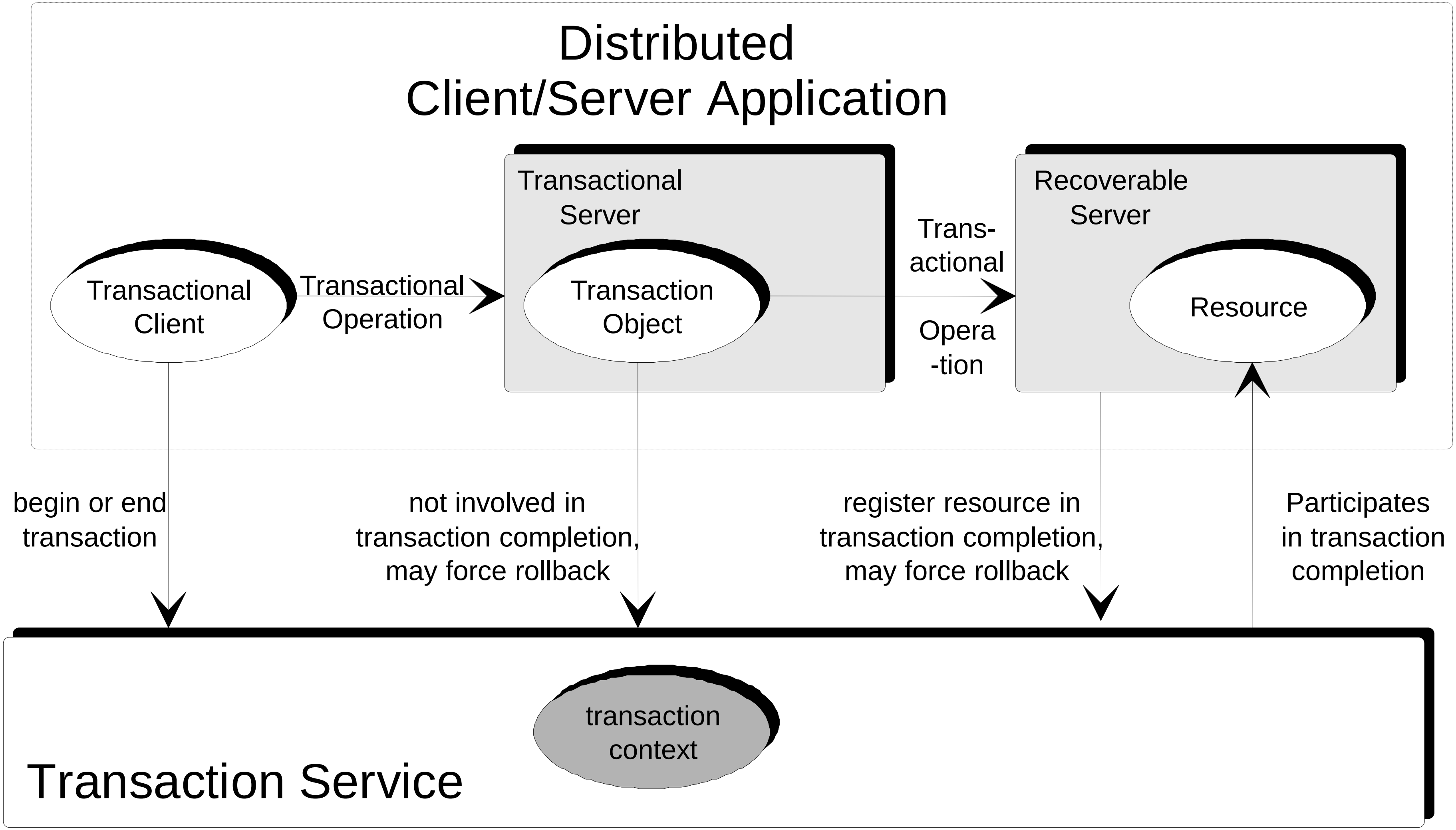
OTS Model: Transactional Objects

- Transactional Object(TO): an object whose methods can be called in a transactional context
- TO is characterized by including some persistent data or pointers to persistent data which can be modified by its methods
- Transactions can be
  - Flat
  - Closed nested

OTS Model

- Recoverable object: a transactional object which is affected by a commit or rollback of a transaction
- Transactional client: program that can invoke operations within a single transaction
- (Transactional client plays role of workflow controller in abstract model of TPM used so far)
- Transactional server: consists of one or more objects involved in a transaction, does not control any resource (transaction server in abstract model)
- Recoverable server: includes at least one recoverable object, plays role of resource manager in abstract TPM model
- Non-transactional clients: everything else
- Presentation server not explicitly called out but realized through non-transactional client objects

Schematic of distributed C/S apps



From CORBA to Application Servers

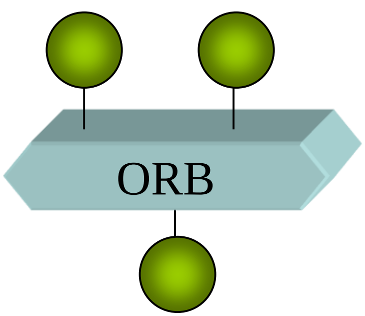
TECHNISCHE UNIVERSITÄT DARMSTADT

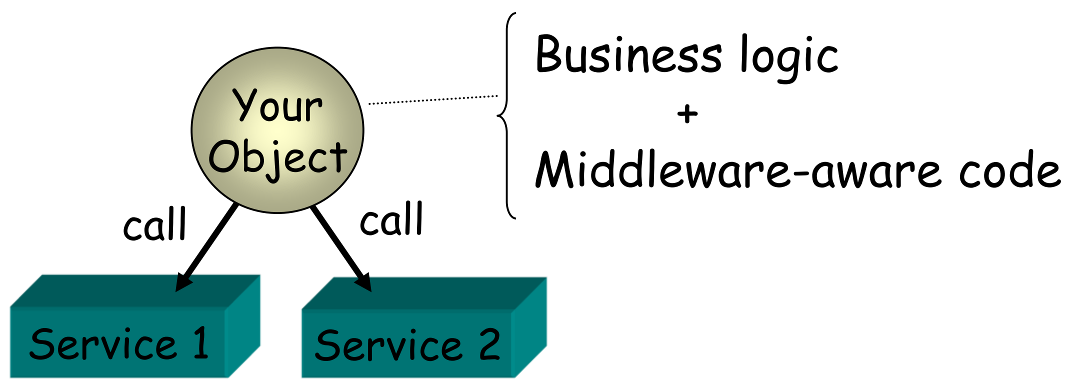
Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

DVS

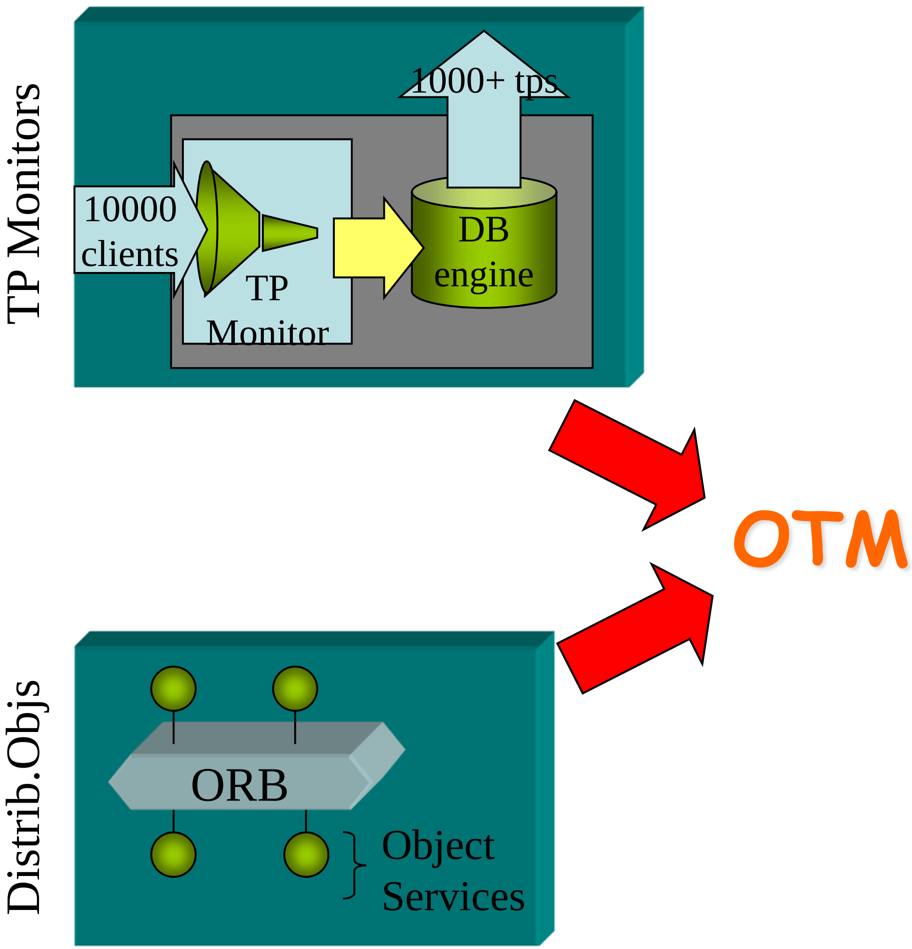
ORBs

- facilitate connectivity between the client application and the distributed objects
  - locate and use distributed objects
  - communication backbone
- let distributed objects interoperate across address spaces, languages, OS, and networks
- not always adequate in high-volume transactional environments





Object Transaction Monitor



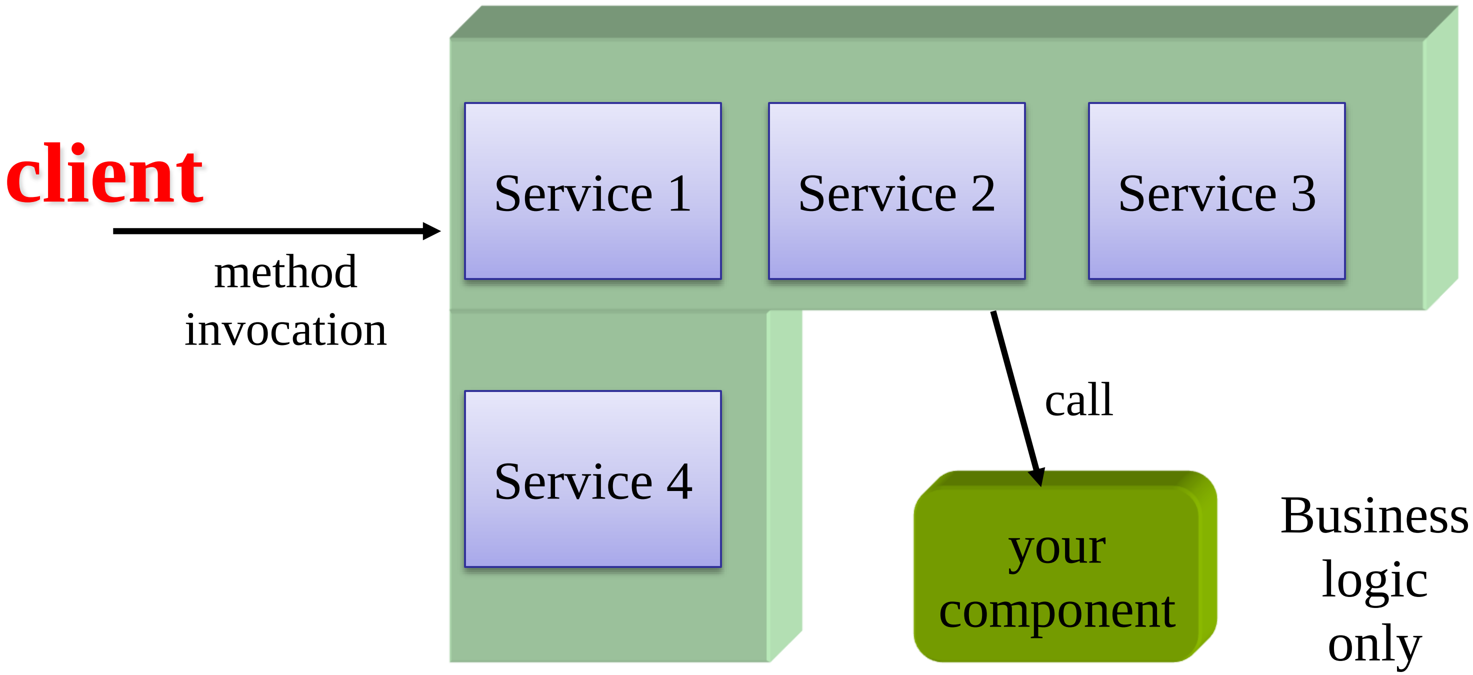
- OTM hybrid of:
  - TP Monitors
  - ORB Technologies
- make easier for developers to create, use and deploy business systems
- capable of handling huge user population and mission-critical work
- provide an infrastructure to manage:
  - transactions, object distribution, concurrency, persistence and resource management

Component Transaction Monitor

- (Application Servers)
- Component Model + (TP-Monitor + ORB)
  - robust server-side component model
- deployers define and administer declaratively the properties of the components by setting their attributes
- resource and service management (monitor)

- CTMs are to business objects what ...
  - RDBMS are to data
  - the railway system is to the trains

CTM Approach



Component Transaction Monitor

- At runtime,
  - it “intercepts” all incoming calls
  - invokes the appropriate callback objects within a container
  - and then passes the request to your object
- Also,
  - pre-starts pools of objects
  - distributes loads
  - provides fault-tolerance
  - coordinates multi-component transactions
- if you play by the CTM’s rules, your objects become managed, transactional, robust, persistent, secure and high-performing

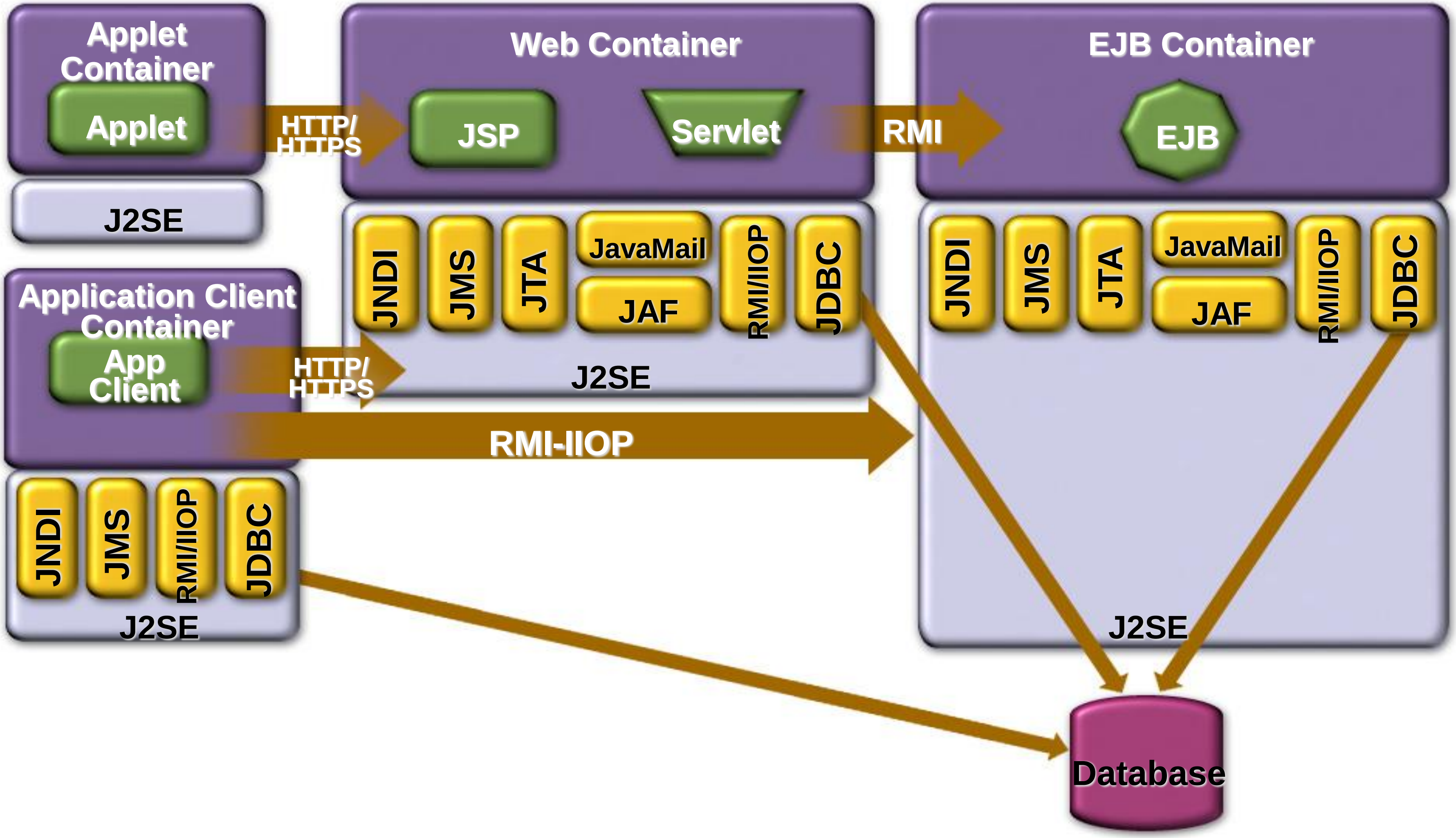
Application Servers (cont.)

- Trend to migrate business logic back to the server and away from the client
  - manageability through thin clients (upgrades, versioning, bug fixes managed on servers)
  - performance (especially of DB intensive apps.)
  - secure network communications
  - minimize downtime
  - reuse of components
- increased reliability through server redundancy
- increased flexibility through multiple tiers
- multi-client support
  - conventional desktops
  - web-clients
  - esoteric devices (smartcards, PDAs, information appliances, cell phones)
- support of variety of middleware services and resource management (multithreading, resource sharing, replication, load balancing)

Java Enterprise Edition (JEE), Enterprise Java Beans (EJB)

Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

JEE – Platform and Application Architecture



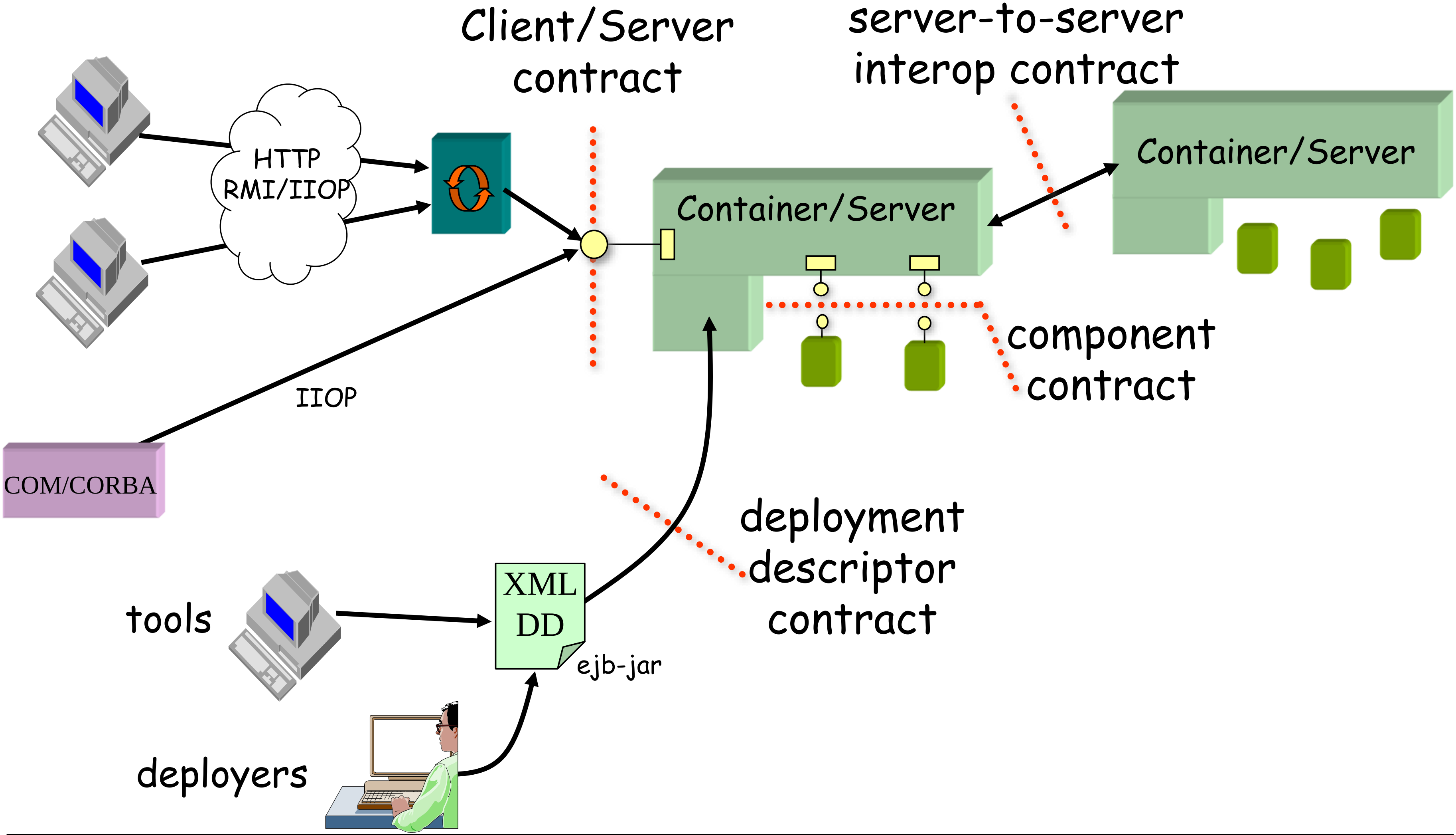
Set of APIs

- Java offers a component model --> Java Beans
- Enterprise Java Platform defines a set of standard Java APIs that provide access to existing infrastructure services (ODBC metaphor)
- EJB specification defines standard model for a Java application server that supports complete portability and implements standard services
- JNDI - Java Naming and Directory Interface (access to DNS, NIS+, NDS, LDAP, etc.)
- RMI - Remote Method Invocation API creates remote interfaces for distributed computing on the Java platform
- Java IDL - creates remote interface to support CORBA communication.
  - Java IDL includes an IDL compiler and a lightweight replaceable ORB that supports IIOP

Set of APIs (cont.)

- Servlets and JSP - Servlets and Java Server Pages support dynamic HTML generation and session management
- JMS - Java Messaging Service supports asynchronous communication through reliable queueing or publish/subscribe
- JTA - Java Transaction API provides a transaction demarcation API
- JTS - distributed transaction service based on CORBA‘s OTS
- JDBC - database access API provides uniform DB access to relational databases

EJB Contracts



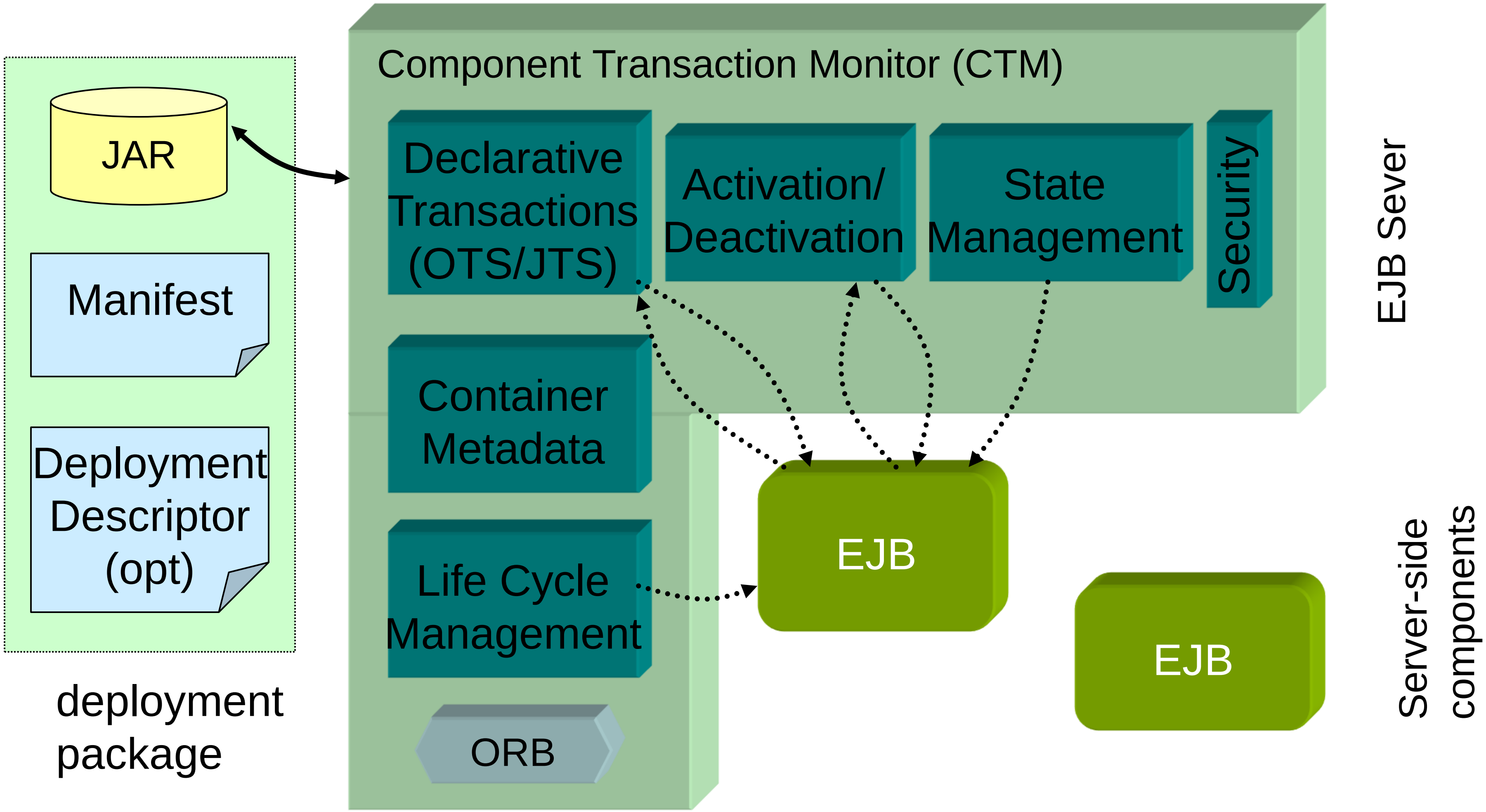
EJBs Specification - Model

- defines an architecture for a transactional, distributed object system based on components
- programming model:
  - conventions or protocols
  - set of classes and interfaces (which make up the EJB API)
- defines the bean-container contract
- Components (Beans) - reusable building block, pre-built piece of encapsulated application
  - Session beans, enterprise beans, message driven beans
- Containers - execution environment for components, provides management and control services for components (i.e. an OS process or thread)

The EJB Container

- Enterprise Beans run in a special environment (Container)
- hosts and manages enterprise beans
- manages every aspect of an enterprise bean at run time:
  - remote access to the bean
  - security
  - persistence
  - transactions
  - concurrency
  - access to and pooling of resources
- isolates the bean from direct access by client applications
- manage many beans simultaneously (reduce memory consumption and processing)
  - pool resources
  - manage lifecycles of all beans
  - Manage bean status
  - the client application is totally unaware of the containers resource management activities

Anatomy of an EJB Container



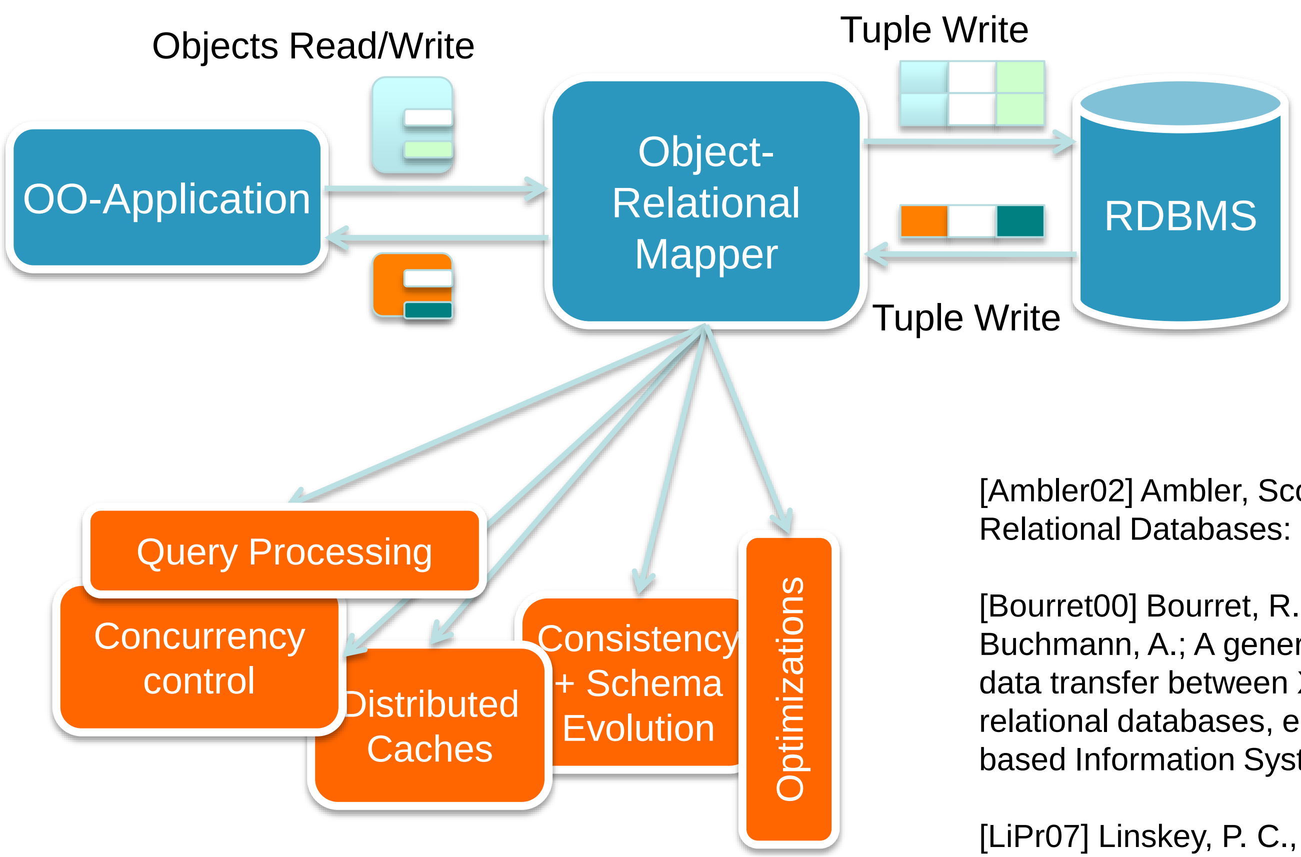
Object Relational Mapping

TECHNISCHE UNIVERSITÄT DARMSTADT

Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

DVS

ORM – Object-Relational Mapper

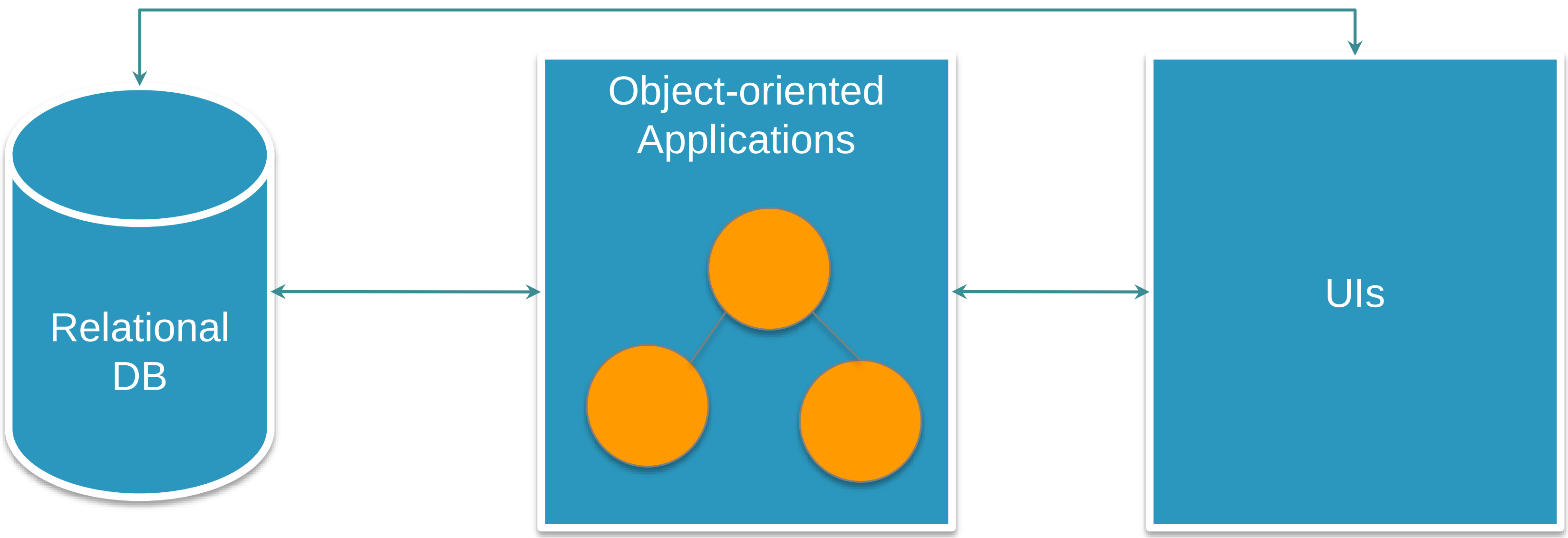


[Ambler02] Ambler, Scott. Mapping Objects to Relational Databases: O/R Mapping In Detail.

[Bourret00] Bourret, R., Bornhoevd, C., Buchmann, A.; A generic load/extract utility for data transfer between XML documents and relational databases, e-Commerce and web-based Information Systems, 2000.

[LiPr07] Linskey, P. C., Prud'hommeaux, M. An in-depth look at the architecture of an object/relational mapper. In Proc. SIGMOD 2007.

Why RDBMS and Objects?



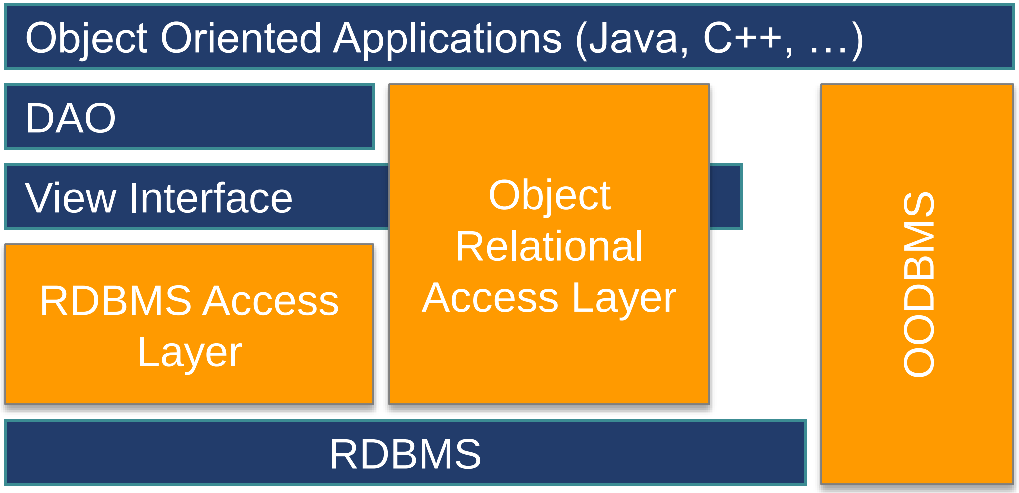
- Proven technology
- Legacy Data

- Leading programming technique
- Flexibility, inheritance, encapsulation, polymorphism, code reuse
- Components

- Different kinds of UIs
- GUI, Web, Mobile, Terminal, …

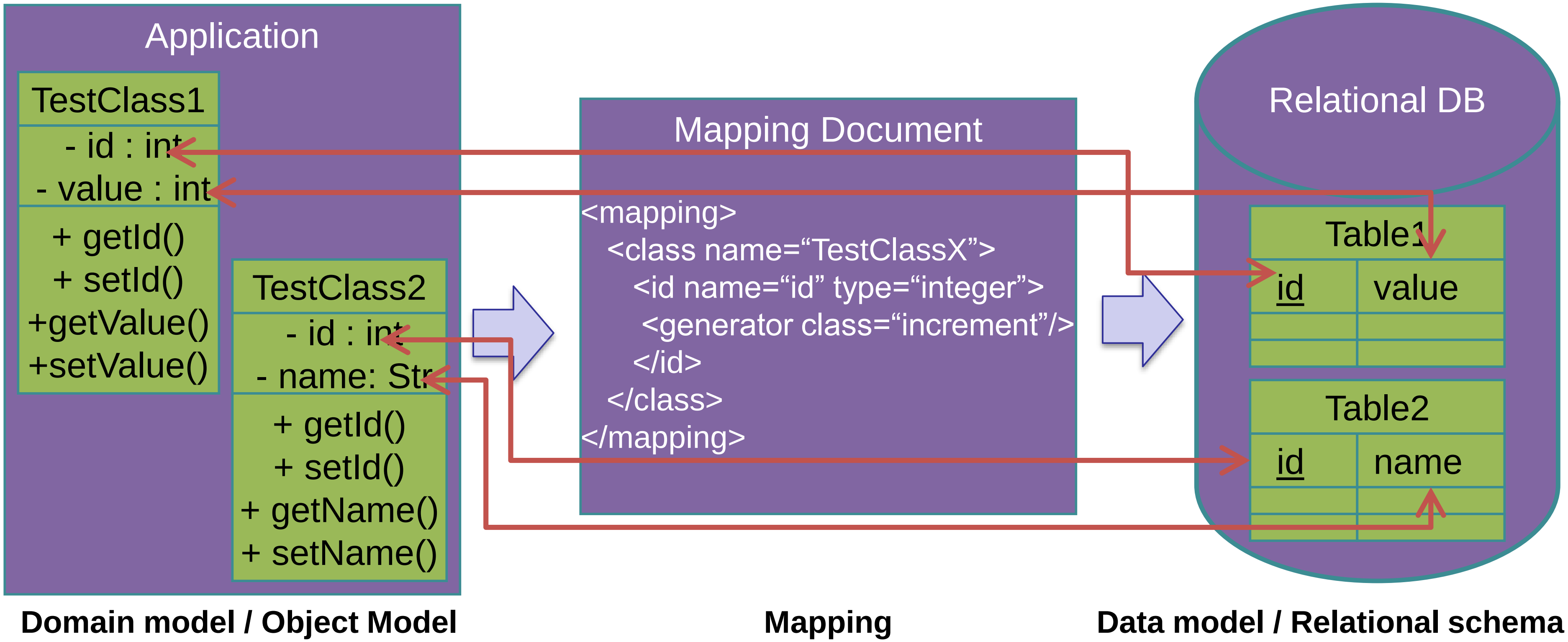
Object Persistence Alternatives

- Standard Persistence (no database)
  - All data in memory for manipulation
  - Custom persistent file organization
  - Not applicable for business applications
- Custom programmed DB persistence
  - ODBC, JDBC, SQLj
  - Data Access Object (DAO) Pattern
- ORM
  - On relational database
  - DB Independent
- OO-database



Mapping / ORM

- Paradigm mismatch
  - Explicit Domain Model/Object Model
  - Explicit Data Model / Relational Schema
  - Explicit mapping Domain Model/Data Model to ObjectModel



Paradigm / Impedance Mismatch

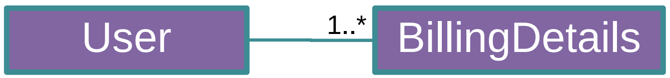
TECHNISCHE UNIVERSITÄT DARMSTADT

- Problem appears in huge software projects with lots of entities
  - Different data representation
  - Different data manipulation
  - Different modeling techniques
  - Semantic gap

- **Impedance mismatch**
  - **Identity**
  - **Complex DataTypes**
  - **OO Features: Inheritance / Encapsulation Polymorphism**
  - **Object Graph Navigation**
- **Mismatch Cost**

Paradigm Mismatch - Example

- Some mappings are straightforward




```
create table USER (
   USERNAME VARCHAR(15) NOT NULL PRIMARY KEY,
   NAME VARCHAR(50) NOT NULL,
   ADDRESS VARCHAR(100)
)
create table BILLING_DETAILS (
   ACCOUNT_NUMBER VARCHAR(10) NOT NULL PRIMARY Key,
   ACCOUNT_NAME VARCHAR(50) NOT NULL,
   ACCOUNT_TYPE VARCHAR(2) NOT NULL,
   USERNAME VARCHAR(15) FOREIGN KEY REFERENCES
   USER
)
```

- How about the mapping of the following diagram
  - Containment of Address

Identity

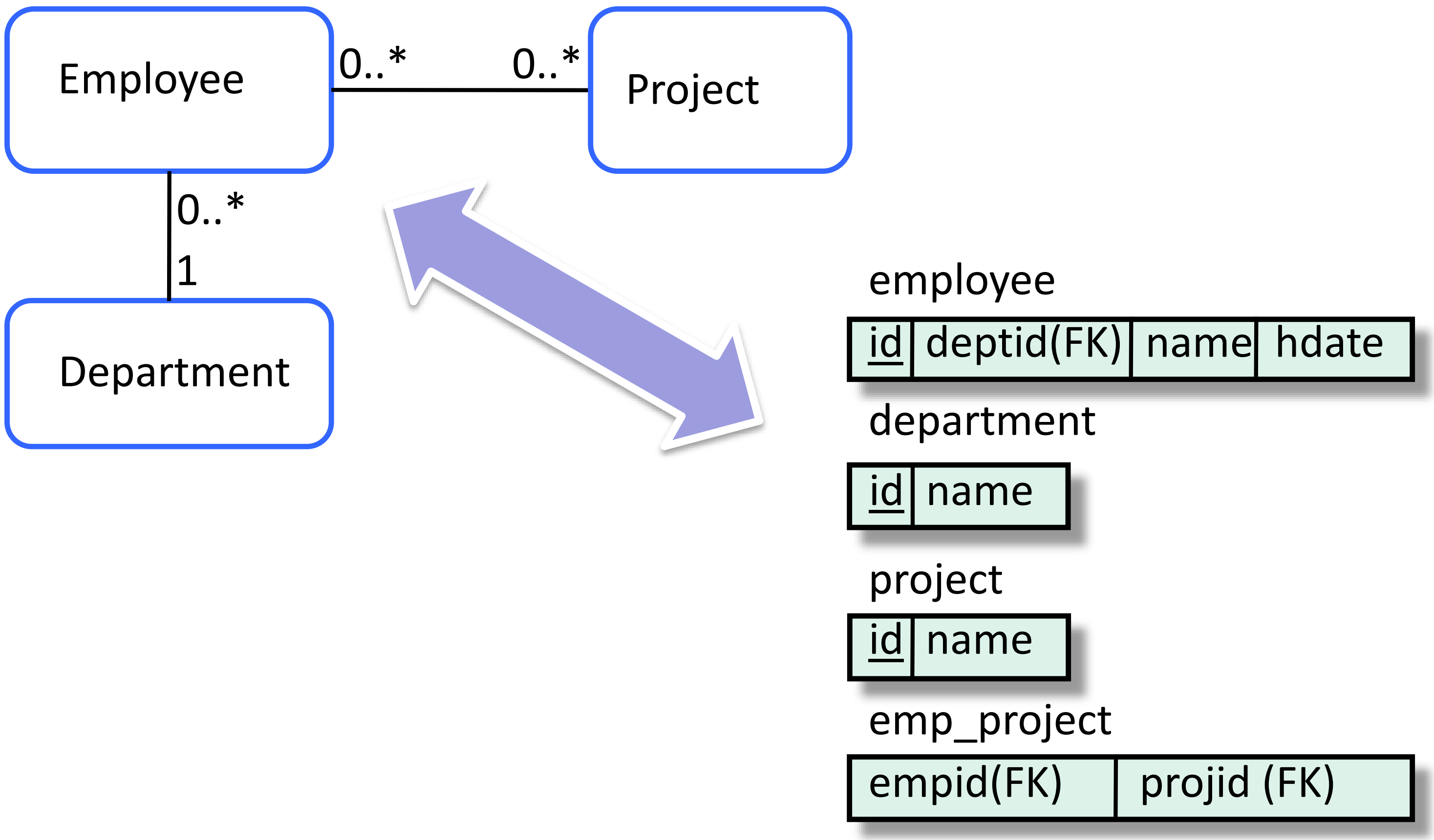
- Object Equality in Java
  - Object identity: memory location → objA == objB
  - Object equality: Implementation of equals() method (equality-by-value/equivalence)
- In RDMBS
  - Two tuples are the same iff they have the same attribute values
- In ORM
  - Every object must have an identity
  - Implementation specific
- Additional Issue: “immutability” of primary keys
  - Recommendation: let ORM use surrogates
  - Natural key also possible, but discouraged unless legacy
    - Practical observation: not immutable on the long run
    - Yields changes to all referencing foreign keys

Data Types / Representation

- Non-matching type systems: OO language <-> DB !
- Character fields → DB: maximum length for character fields | Java: unbounded Strings
- Floating-point numbers → DB: custom representation | Java: IEEE floating-point format
- Fixed-precision numbers → DB: precision and scale same for all values in a column
- Java: each field has specific precision and scale
- Complex DataTypes are widely used in OO design
  - Class clA { **clAddr attrAddr**; ....}; | Class **clAddr**{ int attrZip; String attrStreet;};
- Several possibilities to map on DB features
  - UDT – User-defined Data Types (Part of SQL 99) | Object oriented DB
- ➔ There is no natural mapping on pure relational model.
  - No columns of complex data types
- Use either the same table or a foreign key related table
- Granularity of the object model
  - Fine grained: Many classes mapped on less tables
  - Denormalized schemata → Performance
  - Classes: BillingAddress, HomeAddress
  - Table: USER (BILLING_STREET, BILLING_CITY, BILLING_ZIP, HOME_STREET, HOME_CITY, HOME_ZIP)

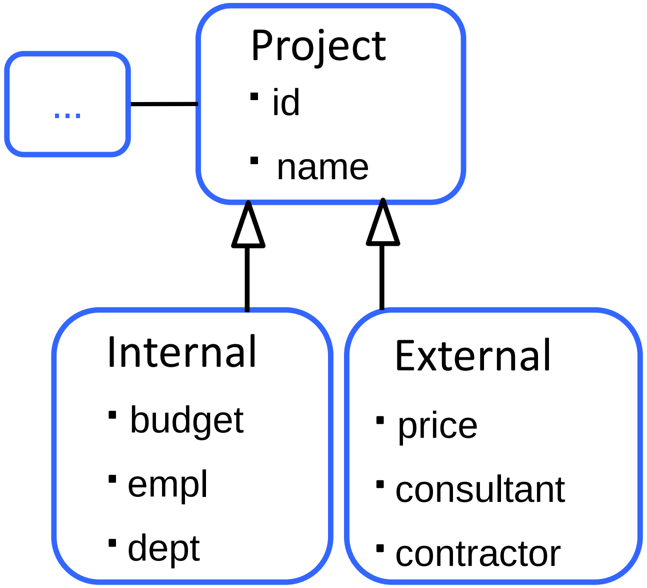
Simple Mapping

- Schema Mapping: map **EntityTypes** onto Tables and vice versa



Inheritance

- ORM support:
  - Inheritance Hierarchies
  - Polymorphic Relationships
- Different Mapping-Alternatives
  - Table per concrete class—Discard polymorphism and inheritance relationships completely from the relational model
  - Table per class hierarchy—Enable polymorphism by denormalizing the relational model and using a type discriminator column to hold type information
  - Table per subclass—Represent “is a” (inheritance) relationships as “has a” (foreign key) relationships



Inheritance – Single Table

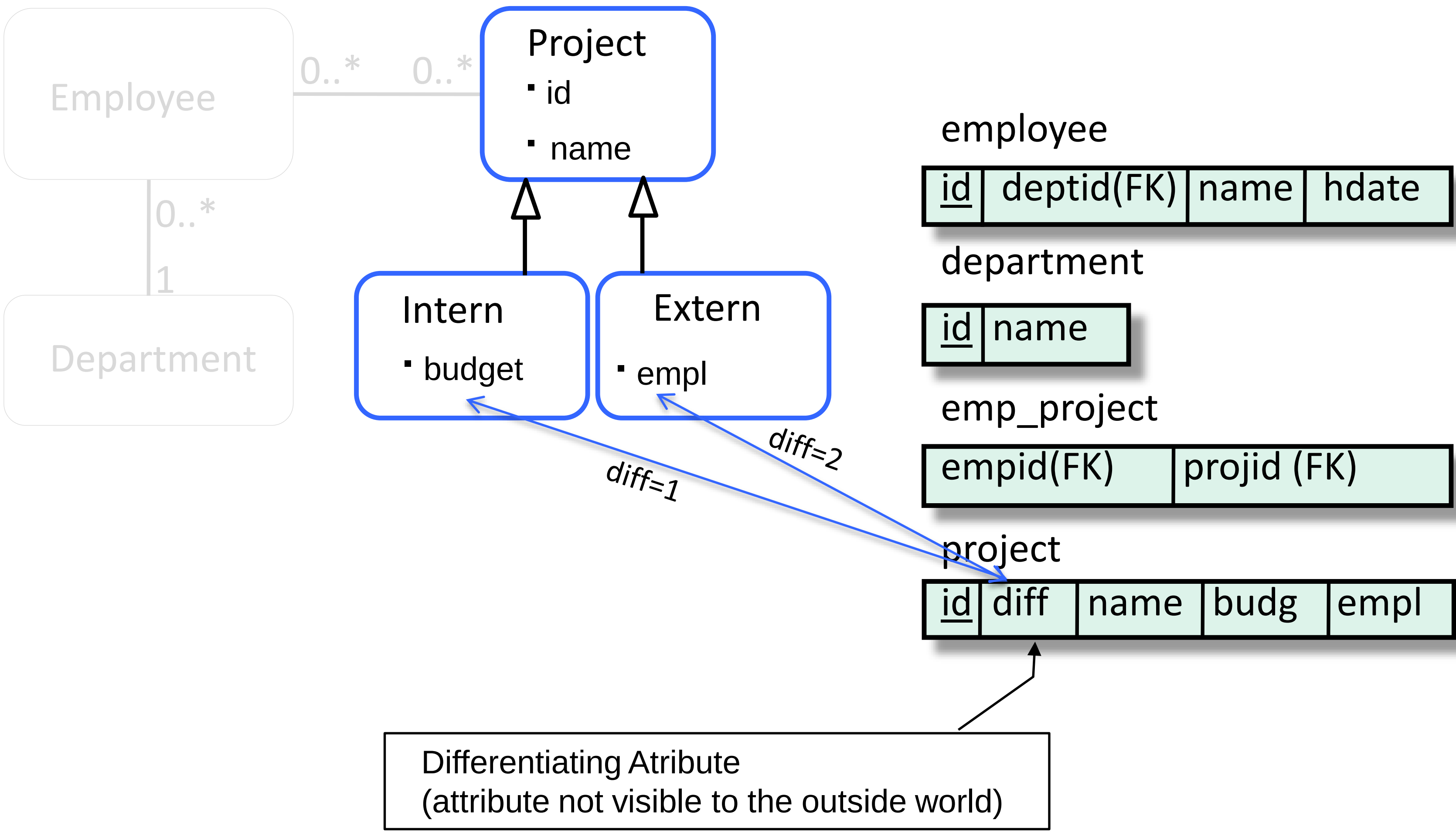
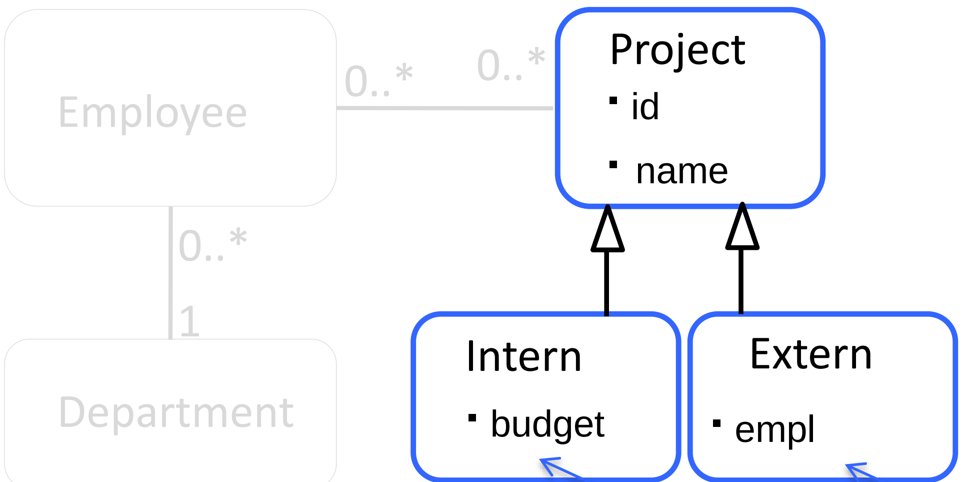
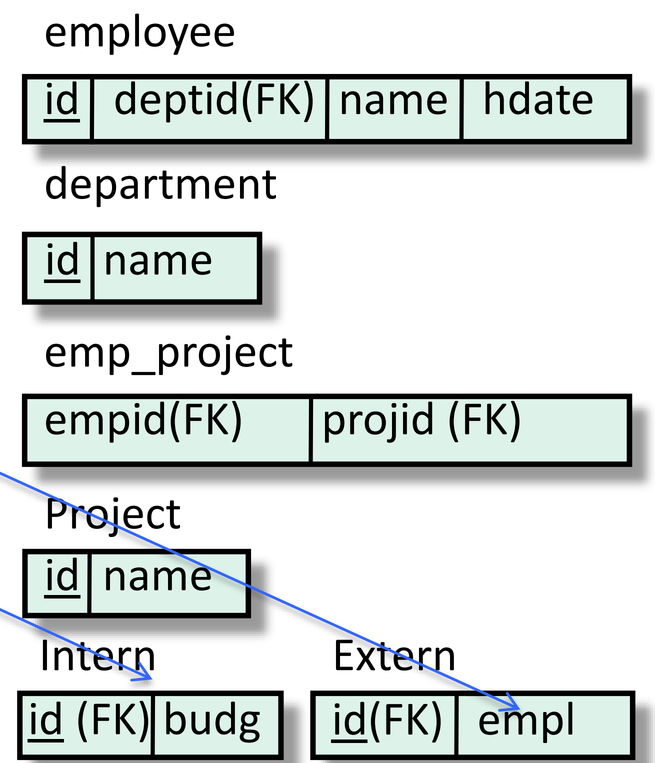


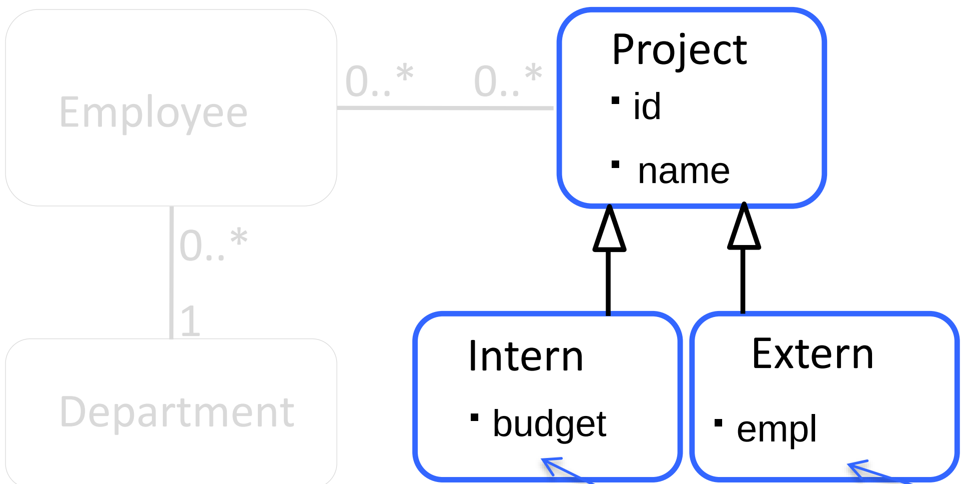
Table per Class





Think about :
- Constraints?
- Disjunctiveness Criterion (DKE)
- Overlapping Criterion (DKE)

(incomplete) Table per sub-class



employee

id	deptid(FK)	name	hdate

department

id	name

emp_project

empid(FK)	projid (FK)

Project

id	diff	name	empl

Intern

id (FK)	budg

Think about :

- Constraints?
- Disjunctiveness Criterion (DKE)
- Overlapping Criterion (DKE)

Mapping Strategy - Rules of Thumb

- Table-Per-Concrete-Class:
  - No polymorphic associations or queries needed
- Table-per-Class-Hierarchy:
  - Polymorphic associations or queries needed and subclasses have few properties
  - Default for simple cases
- Table-Per-Subclass strategy:
  - Polymorphic associations or queries needed and subclasses have many properties
  - For more complex cases or if nullable constraints needed → Decision: consider remodelling the inheritance in the domain model
- Mapping strategies can be applied to abstract classes and interfaces
  - Interfaces → treated as abstract classes (accessor meth.)

Object Graph Navigation – Associations

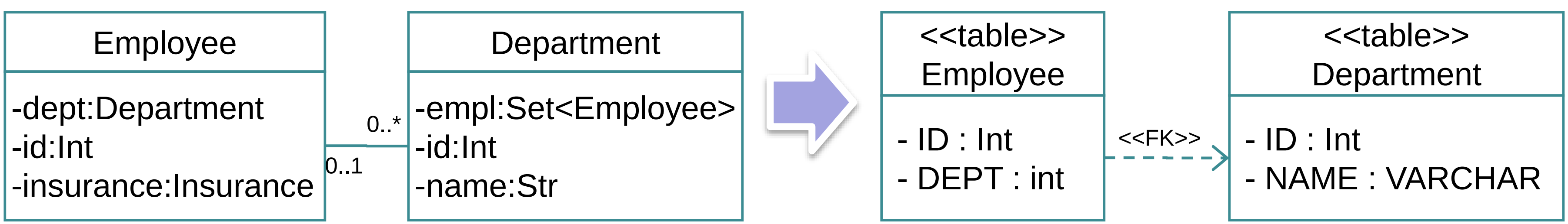
- OO associations are object references or collections of such
  - Have direction / are navigational
  - In general – Bidirectional associations
  - ORM implementations support uni-directional associations (bidirectional implemented as inverse attributes)
  - Multiplicities implemented through sets
- In relational model
  - Primary keys / Foreign Keys
  - Constraints (referential integrity, assertions, check constraints)

```
public class Item {
   private Set bids = new HashSet();
   public void setBids(Set bids) {  this.bids = bids; }
   public Set getBids() { return bids; }
   public void addBid(Bid bid) {
      bid.setItem(this); bids.add(bid); }
}
```

```
<class name="Item“ table="ITEM">
     .....
   <set  name="bids“ inverse="true“ cascade="save-update">
     <key column="ITEM_ID"/>
     <one-to-many class="Bid"/>
   </set>
</class>
```

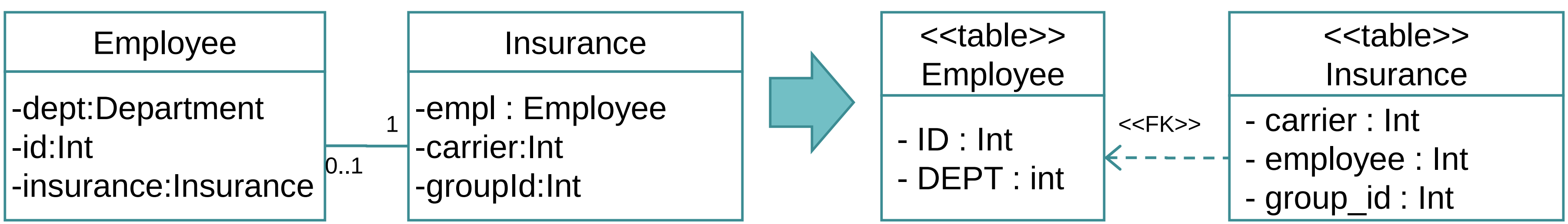
1:N Associations

- 1:N mapping realized through foreign key
  - Bi-directional navigability in DomainModel → Fkey
  - Fkey Constraint → mapped to multiple fields (dept, employees) in the domain model
    - Implicit → Association roles / Explicit → Fields
- Example
  - EMPLOYEE table – as DEPT column
  - DEPT is Fkey in EMPLOYEE table (Constraint – referential integrity)



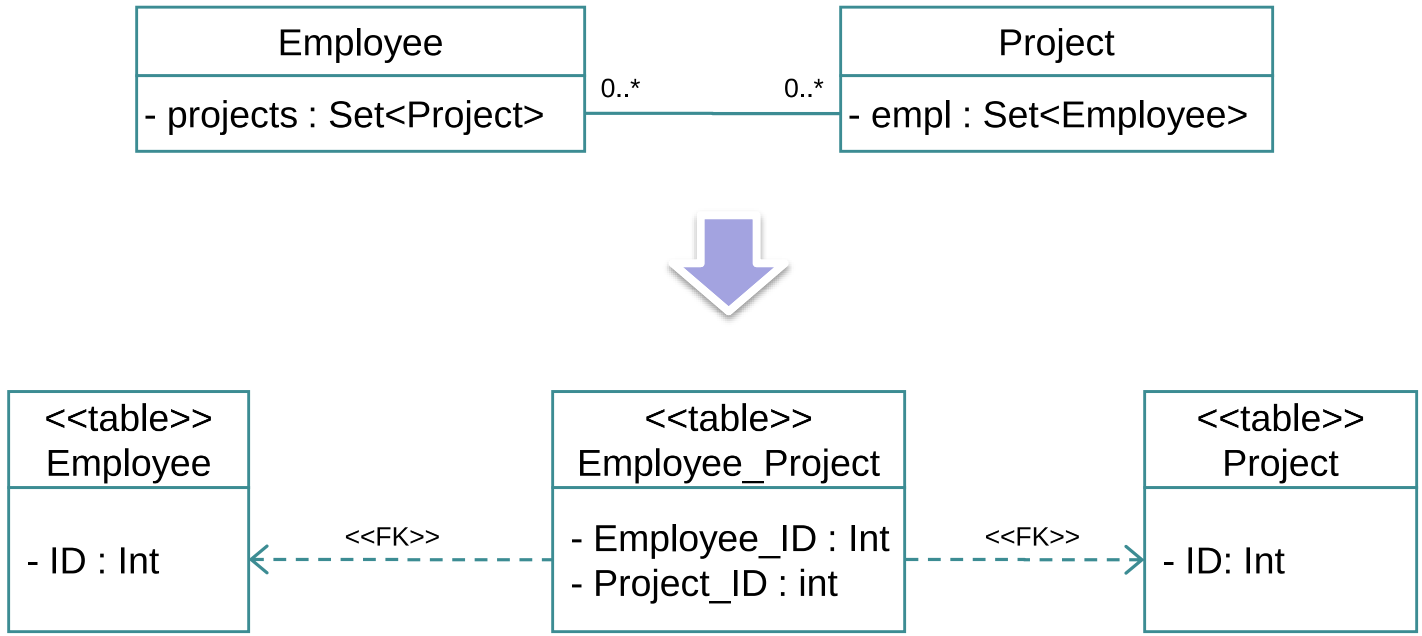
1:1 Associations

- 1:1 mapping realized through foreign key (as 1:N)
  - Additional constraint (difference 1:N): FKey UNIQUE
    - Depending on the optionality UNIQUE, NOT NULL
- Data model
  - Foreign key – Employee column
- Domain mode
  - One reference attribute per class (department, employee)



M:N Associations

- M:N Associations mapped on separate/join tables
  - Example: Employee_Project

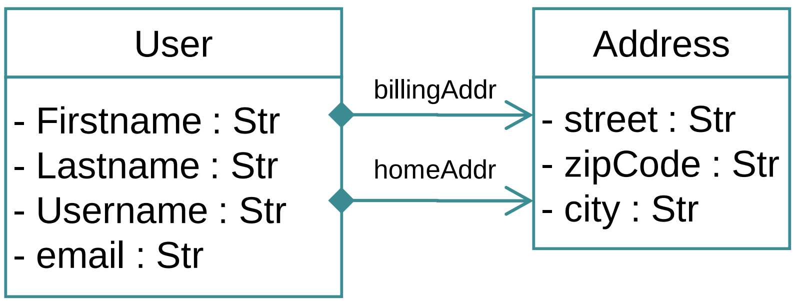


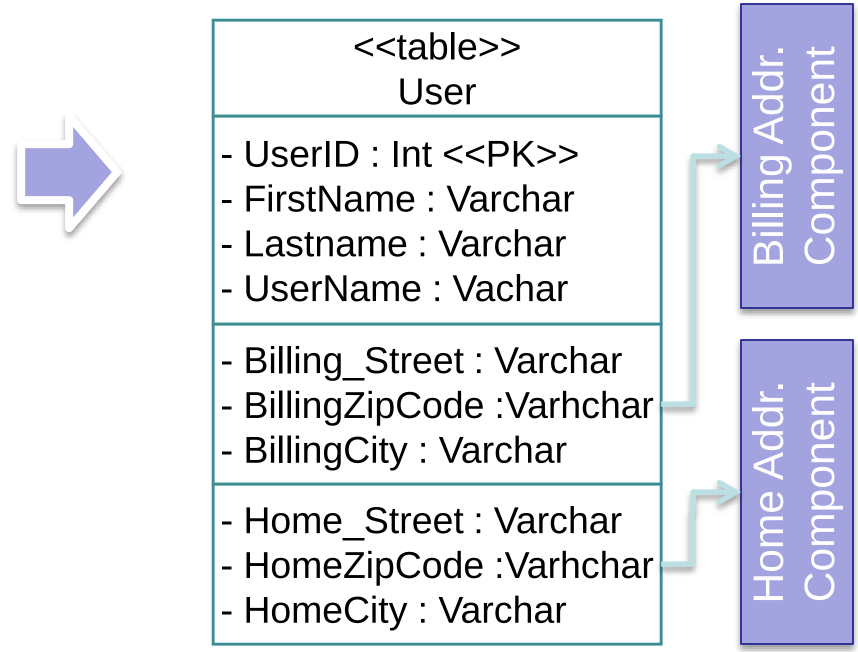
Composition – 1

- Relates to complex data types, and dependent classes
- Lifecycle dependence (in domain model)
  - Lifecycle of an object depends on the existence of another instance
  - Deletion of the containing object → deletion of the component object
- Mapping (in the data model)
  - Single table (dependent and independent objects fall together) → Containment
  - Two Tables (strict foreign key, not NULLABLE, cascading delete) → Aggregation

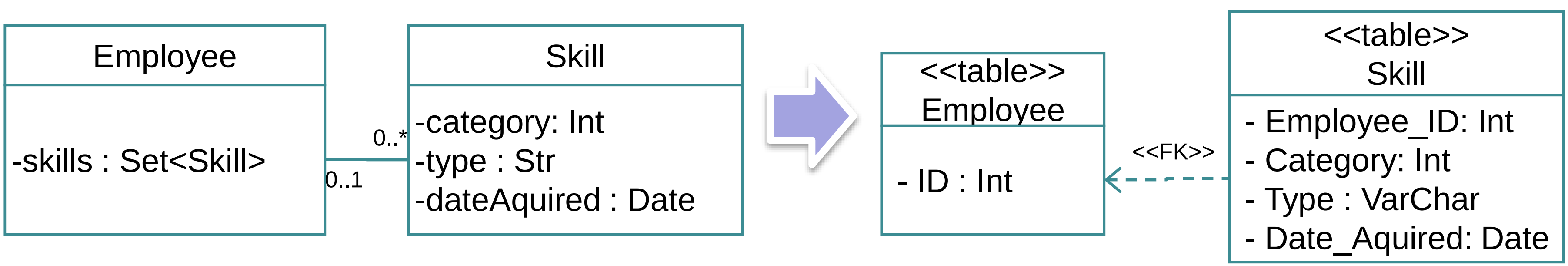
Composition – 2

- Mapping Strategy 1: Single Table





- Mapping Strategy 2: two tables



Data modeling vs. OO modeling

Data Modeling for RDBMS	OO Modeling for OODMBS
Associations computed using joins- Keys have to be developed	Associations are explicit using pointers
Only primitive data stored in columns (characters, numbers)	Structured data of arbitrary complexity can be stored.
Code independently developed and in different places (stored proc. & triggers in DB)	Code found in classes
An ‘object’ can be distributed among tables (problem for complex objects)	An object in one place, so is fast to access
Associations are by default bi-directional	Associations can be unidirectional

Mismatch Cost

- ORM avoids writing complex SQL/JDBC code
  - 30% of the Java application code
- Object modeling → intuitive, easy to maintain
- SQL CLI (ODBC/JDBC) – statement based interface
- Smart caching and query generation/execution yield significant performance increases

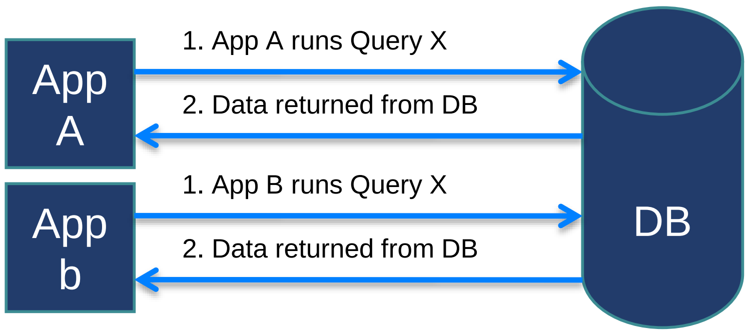
ORM Transactions

TECHNISCHE UNIVERSITÄT DARMSTADT

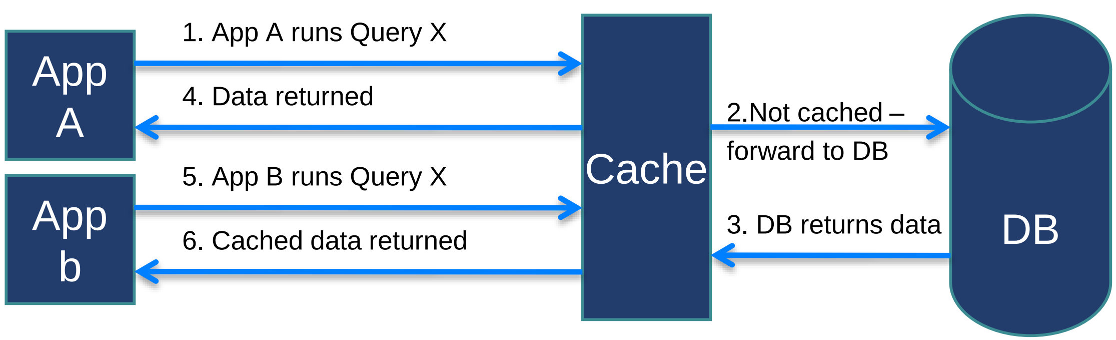
- Optimistic transactions [Linskey et al. 2007]
  - Minimize resource utilization and resource blocking
  - Usable in read-mostly situations
  - Optimistic locking (optimistic concurrency control) [see DB2 course]
    - Execute the operation (assume: no conflicts will occur)
    - No pre-emptive locks → use additional column (numeric/timestamp)
    - Upon write (update, delete) → ensure there are no conflicts
      - Check extra column, WHERE clause of UPDATE/DELETE;
      - Forward, backward validation
    - If WHERE fails → OptimisticLock_Exception raised
- Deadlock prevention
- Caching
  - Effectiveness: Percentage of reads not time between writes

ORM Caching

- ORM without caching



- ORM with caching

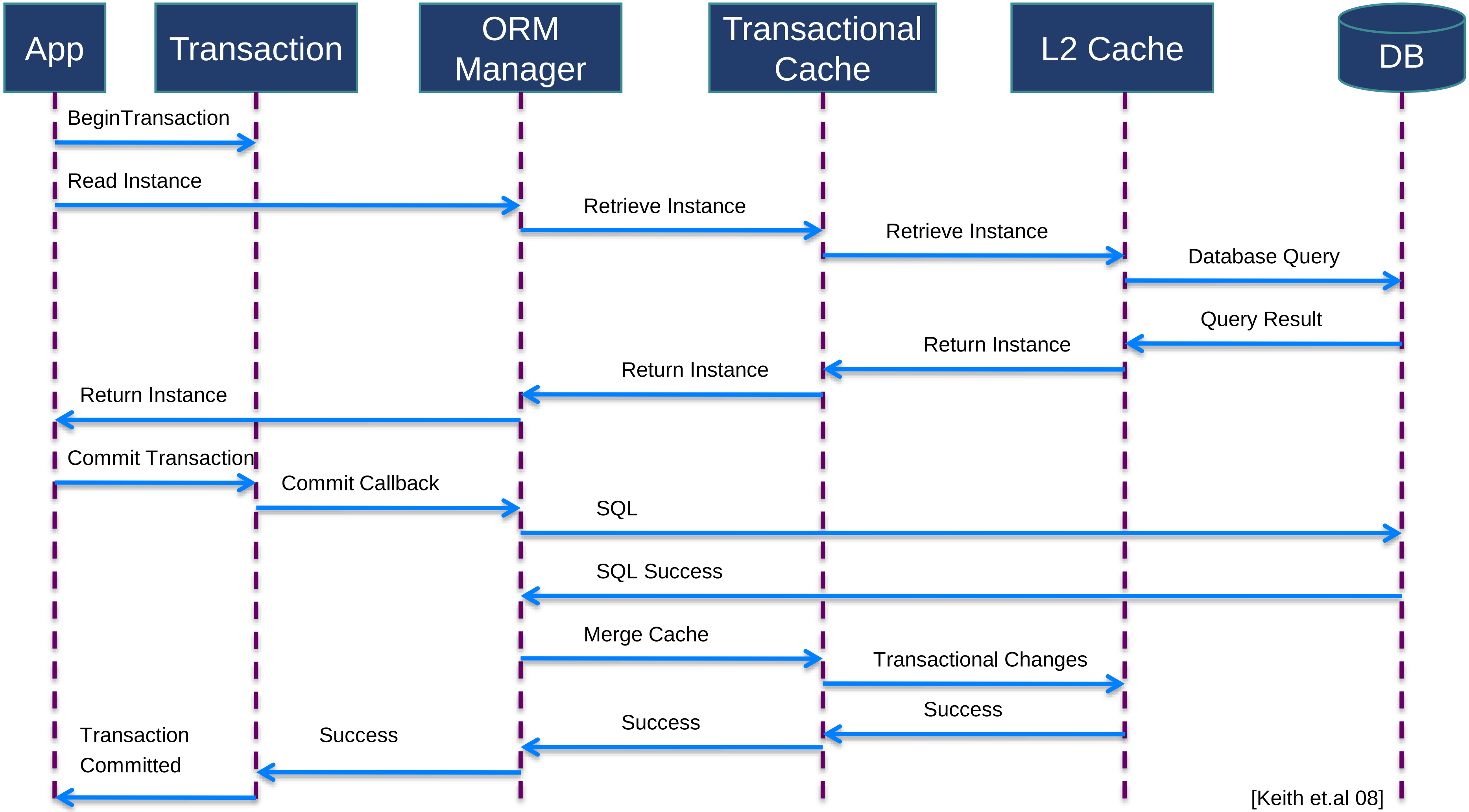


[Linskey et al. 2007]

ORM Caching

- Object identity - identity of object A pointed by N other objects along different paths/relationships must be preserved when loading. Avoid N distinct memory imprints of obj. A.
- Caching Levels
  - Transactional cache – supports transactional objects. Represent the change summary of an obj. from ORM perspective within transactional boundaries.
    - Upon rollback: changes discarded
    - Upon commit: changes → SQL → persisted in DB
  - Shared – globally shared cache for read-only objects. Read-only operations from within a trans. allowed.
  - Query caching
- Caching influences transactions
- Statement ordering and batching

ORM Transactions - Caching

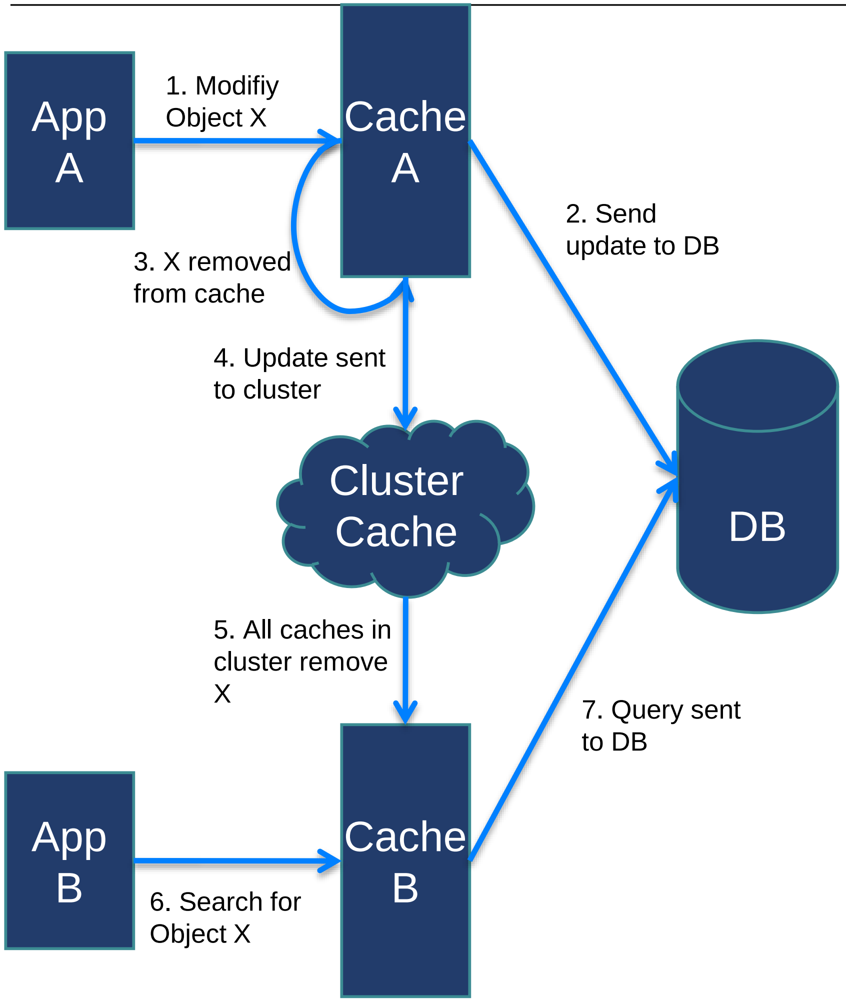


[Keith et.al 08]

Cache Granularity

- What is cached? Different types of ORM cache (structures)
- Object cache
  - Candidate: transactional caches
  - Objects built, preloaded with state from the DB
  - Retrieved data stored in the object aggregate (no other cache)
- Data cache
  - Raw object data cached, no object composition
  - Fragments easily manipulated and stored. No added cost due to object relationships. Allows for extensions.
  - Cost: successful cache requests → 1 or more obj. construction
- Query cache – Principle: held in ORM not a DB → no full query processing!
  - If query on primary Key values and all in cache → execute query in ORM → avoid DB roundtrip
  - If query on non-key values → execute against DB → result: set of identifiers OR complete Data → used to obtain objects from cache
    - Retrieval tradeoffs: Identifiers only <-> complete entity data

Clustered Caches



[Linskey et al. 2007]

- ORM applications run on multiple servers → affects the cache
  - A client updates entity in DB
  - Invalidate cached versions of that entity in all client caches
- Approaches
  - Initiating cache notifies other caches about object changes
  - Every cache acts independently, checks DB for changes
  - Caches rely on notification service to inform them about object changes → evict or refresh objects

ORM Transactions - Isolation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Caching and cache operations (load, merge, evict) influence transaction isolation
  - READ_UNCOMMITTED to be avoided
  - READ_COMMITED (Default)
    - Queries contain no uncommitted data
    - No ORM cache merge until commit
    - Can be realized by (a) DB locking or (b) by using versioning
  - REPEATABLE_READ
    - Reproducibility in query result sets (except for phantoms)
    - Can be realized by (a) DB locking or (b) approximated by versioning and transactional cache
  - SERIALIZABLE seldom used by ORM apps
    - Use pessimistic locking instead

Queries

- Association provide navigational-support
  - Sometimes realized as SQL queries
- Query-mechanism still needed
  - entry points → search-functionality
  - filtering → selections of data with declarative criteria
- Queries should be:
  - Expressed in terms of the domain model
    - statically checked on data model (safety, compile time checking)
    - less complex
  - return objects as specified in domain model
  - select-constraints should be specified as Classes
  - Should allow for navigation along associations
- Different ORMs have own query languages
  - Hibernate → HQL, JDO → JDOQL, JPA → JPQL, EJB→ EJBQL, ...

Queries 2 - Examples

- To find all instances of Employee where the weekly salary is greater than some parameter
  - SELECT E.* FROM EMPLOYEE E, DEPARTMENT D
    WHERE E.WEEKLY_SALARY > ? AND D.NAME = ?
    AND E.DEPARTMENT = D.ID

- The corresponding query using the domain object model would be:
  - SELECT FROM FullTimeEmployee
    WHERE weeklySalary > :salary && dept.name = :dept

Queries 3 – Navigating relationships

- Major differences in query languages
- JDOQL uses query filters as boolean expressions
  - Set.contains(), Set.containsKey(), Set.ContainsValue()
  - SELECT FROM Employee
    WHERE weeklySalary > :salary
    && dept.name == :dptname && projects.contains(p)
    && p.name == :prjname
- Java Persistence API JPQL – explicit Joins
  - SELECT e
    FROM Employee e, JOIN e.projs as p
    WHERE e.weeklySalary > :salary
    AND e.dept.name = :dptname AND p.name = :prjname

ORM Performance

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Factors affecting the performance
  - Change detection
    - Track changes on objects at transaction commit
    - Iterate and compare obj. with DB state or Track as modified
  - Exact data retrieval
    - Persistent state shared among may OO apps
    - Objects retrieved as a whole (no projection)
    - App informs ORM which columns are needed in a use-case
    - Define fields statically → eager fetch
    - Dynamic: App. writes query + navigational path to retrieve obj
    - General: App defines fields of group of classes to be fetched. Groups used in queries to retrieve data per use-case
  - Caching

ORM Optimizations

TECHNISCHE UNIVERSITÄT DARMSTADT

- Pre-fetching - Always fetch all related faults for a given relationship.
- Batch Faulting - On an array of faults always fetch the first n objects.
- Miscellaneous - Stored procedures, raw sql queries, read only optimizations, database optimizations…
- Plan business logic around database roundtrips. Make sure your data is ready to be presented.
- Use optimized object models for different usage. Separate data maintenance from presentation
  - Complex object model for data entry app
  - Simplified model for live site query/display

ORM Advantages

- Productivity
  - Eliminates lots of repetitive code – focus on business logic
  - Database schema is generated automatically
- Maintainability
  - Fewer lines of code – easier to understand
  - Easier to manage change in the object model
- Performance
  - Lazy loading – associations are fetched when needed
  - Caching
- Database vendor independence
  - The underlying database is abstracted away
  - Can be configured outside the application

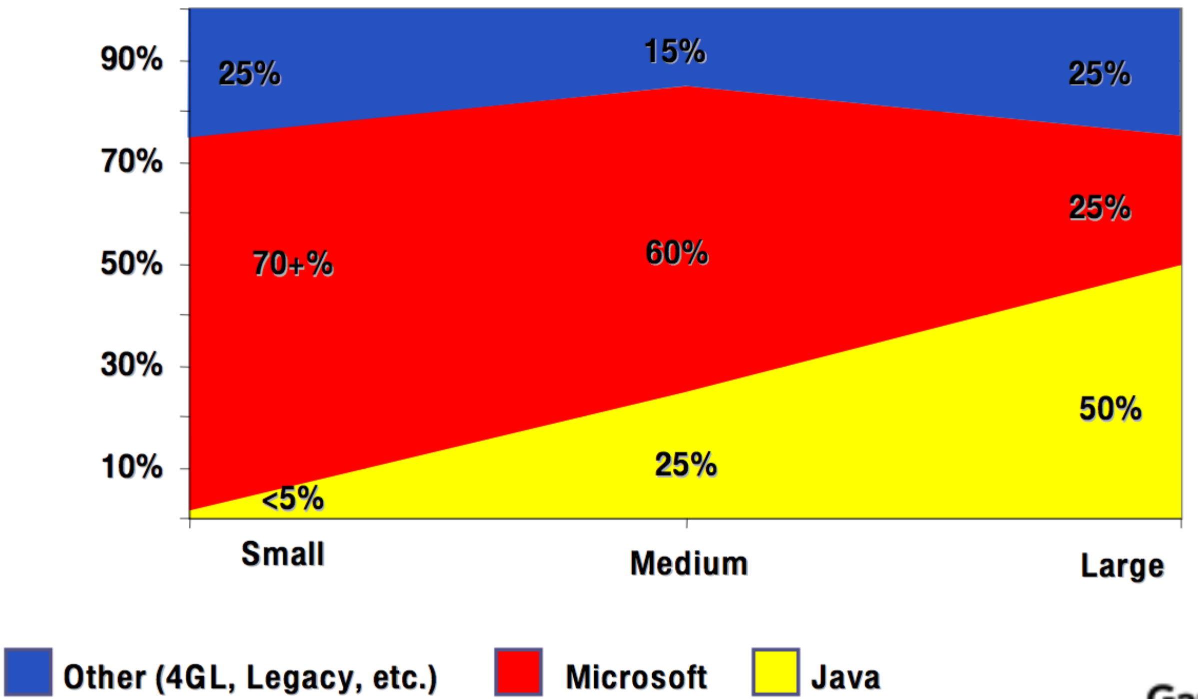
ORM vs. OODBMS

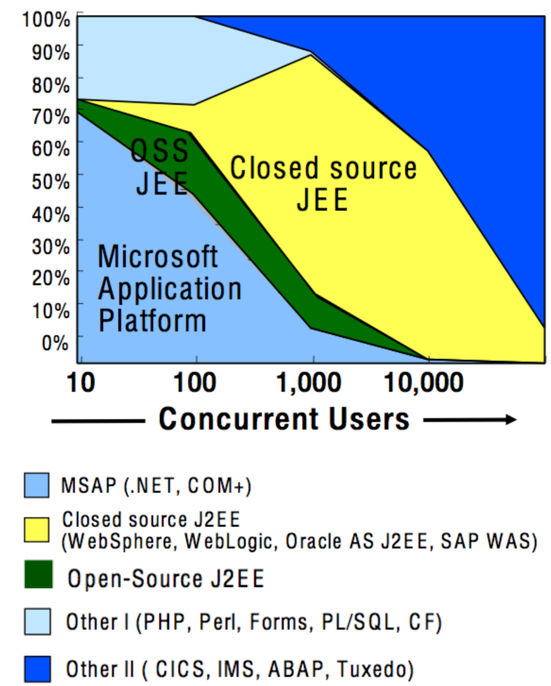
TECHNISCHE UNIVERSITÄT DARMSTADT

- OODMBS ease programming, RDBMS ease report generation and data mining;
- OODBMS - object model navigation, ORM - sequential processing and complex queries;
- If writing an application from scratch, use OODBMS, if you need to integrate to various sources of legacy data - ORM.
- ORM → induce persistence overhead, but allow for flexibility
- Performance reviews: ORM and OODBMS
  - Equally fast in object graph traversal, ORM better on large collections
  - Query performance: OODMS marginally better
  - ORMs perform better on over large data collection and join queries.
  - OODBMS perform faster inserts deletes
  - DB Size disregarded, open source DBMS for ORM used
- [P. van Zyl et al., 2006]

Technological preferences [Gartner]

- Large Enterprise → Java EE Technologies
- Small and middle-sized enterpises → Microsoft Technology (.NET)





Gartner

References

- Linskey, P. C. and Prud'hommeaux, M. 2007. An in-depth look at the architecture of an object/relational mapper. In Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data.
- Scott Ambler “Mapping Objects to Relational Databases”
  - http://www.agiledata.org/essays/mappingObjects.html
- Christian Bauer, Gavin King. Hibernate in Action. Manning Publications. 2004
- Craig Russell. Bridging the Object-Relational Divide. ACM Queue, Vol. 6, No. 3, 2008
- Michael Keith and Randy Stafford. Exposing the ORM Cache. ACM Queue, Vol. 6, No. 3, 2008
- Van Zyl, P., Kourie, D. G., and Boake, A. Comparing the performance of object databases and ORM tools. In SAICSIT 2006.

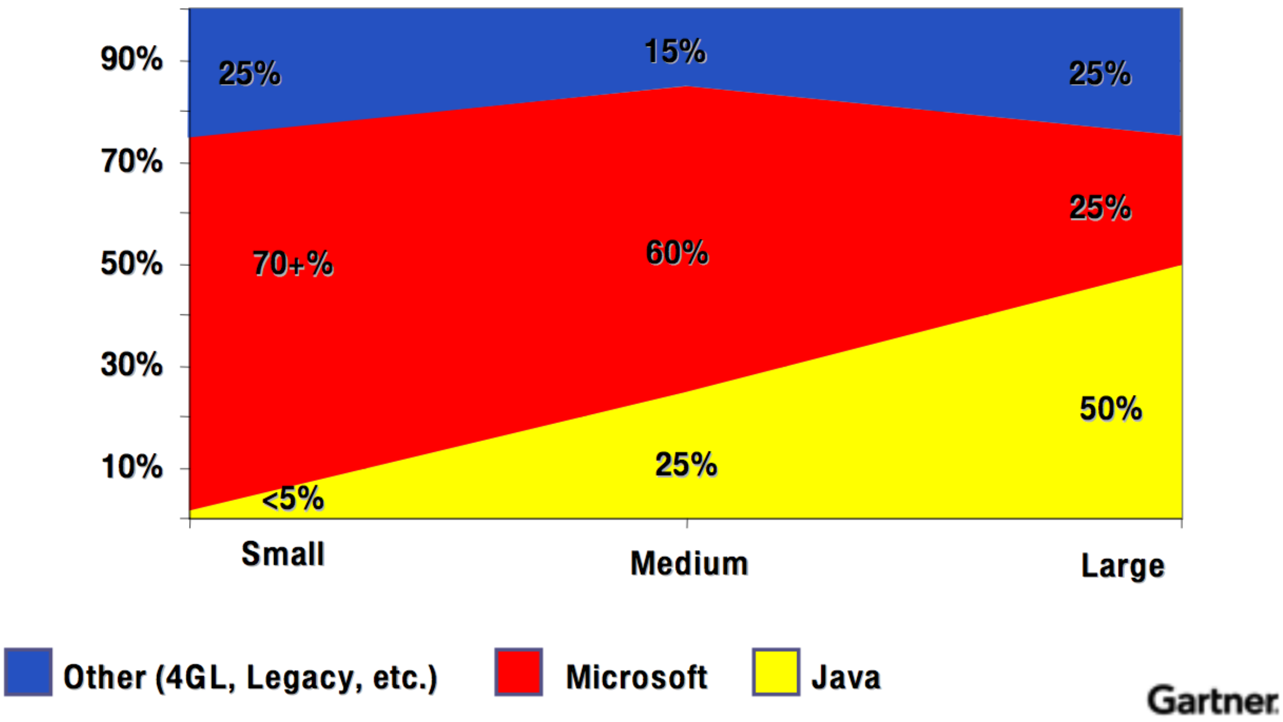
Persistence Frameworks

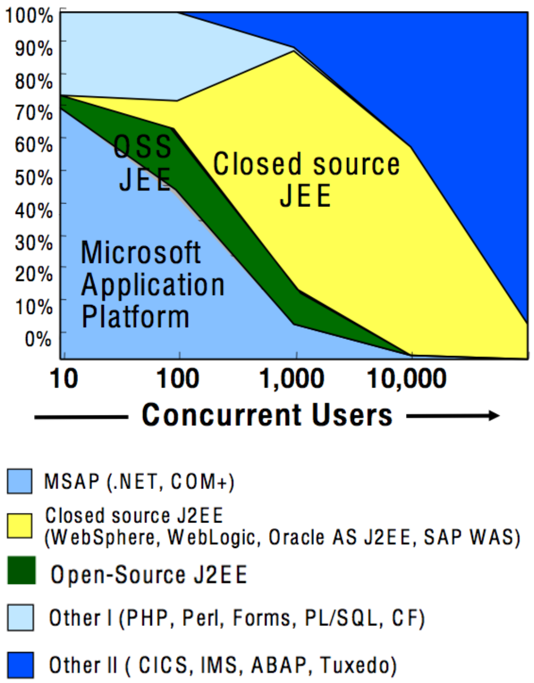
OPTIONAL! Additional Material

Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

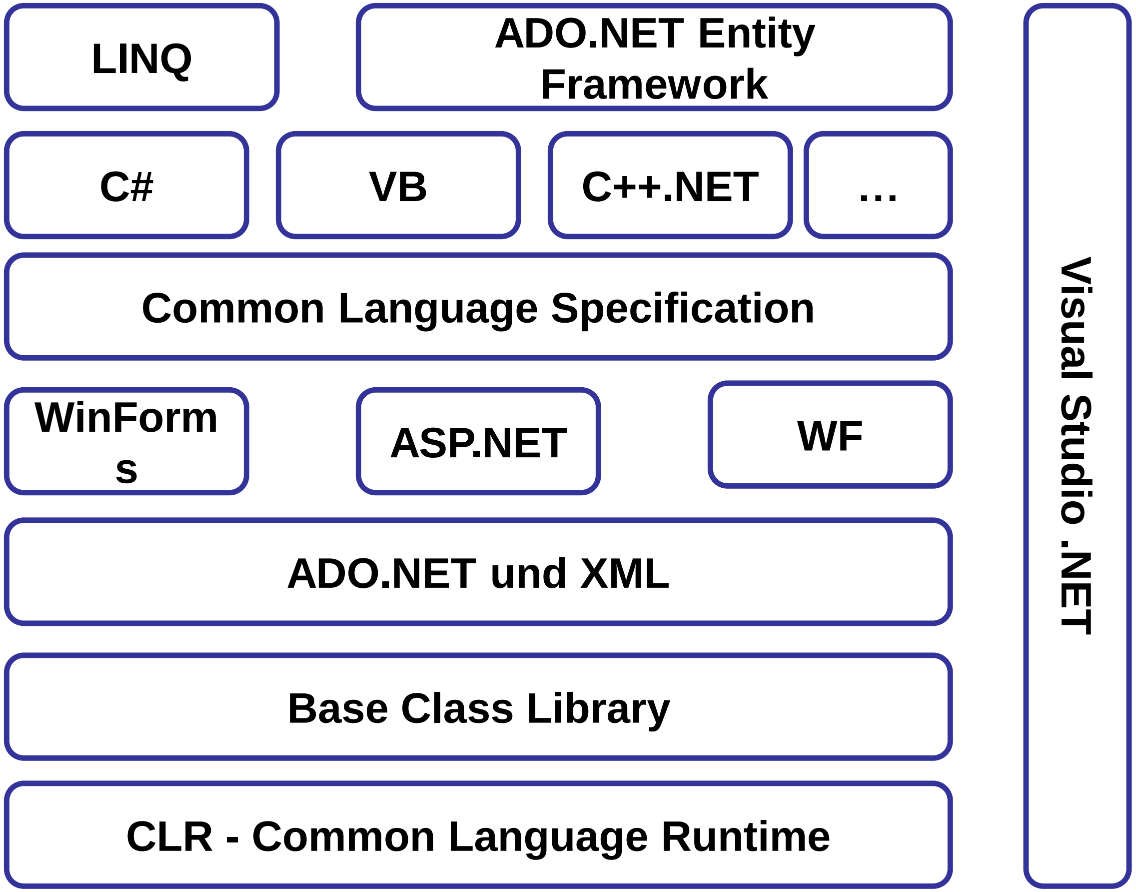
Technological preferences [Gartner]

- Large Enterprise → Java EE Technologies
- Small and middle-sized enterpises → Microsoft Technology (.NET)



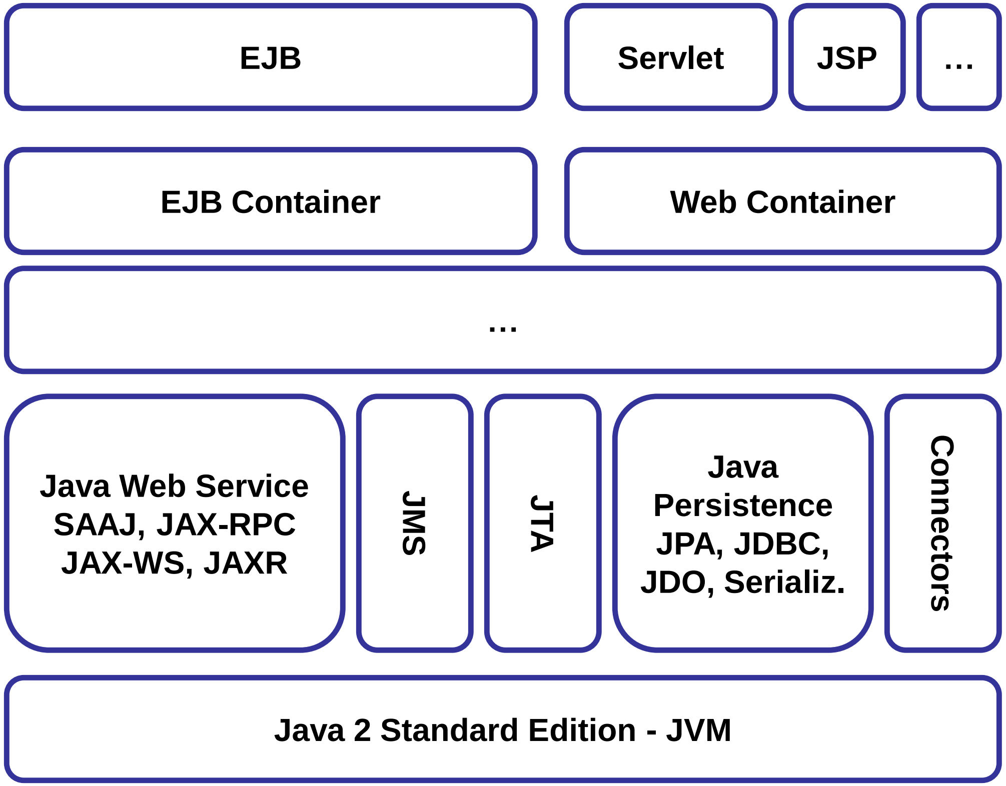


Microsoft .NET – Architecture Diagram



[Microsoft]

Java EE – Architecture Diagram (already discussed)



[Jord04] Jordan, M. 2004 *A Comparative Study of Persistence Mechanisms for the Java™ Platform. Technical Report. UMI Order Number: SERIES13103., Sun Microsystems, Inc.*

[Sun/Oracle]

Comparisons

[Sing03] Singer, J. JVM versus CLR: a comparative study. In *Proceedings of the 2nd international Conference on Principles and Practice of Programming in Java.2003*

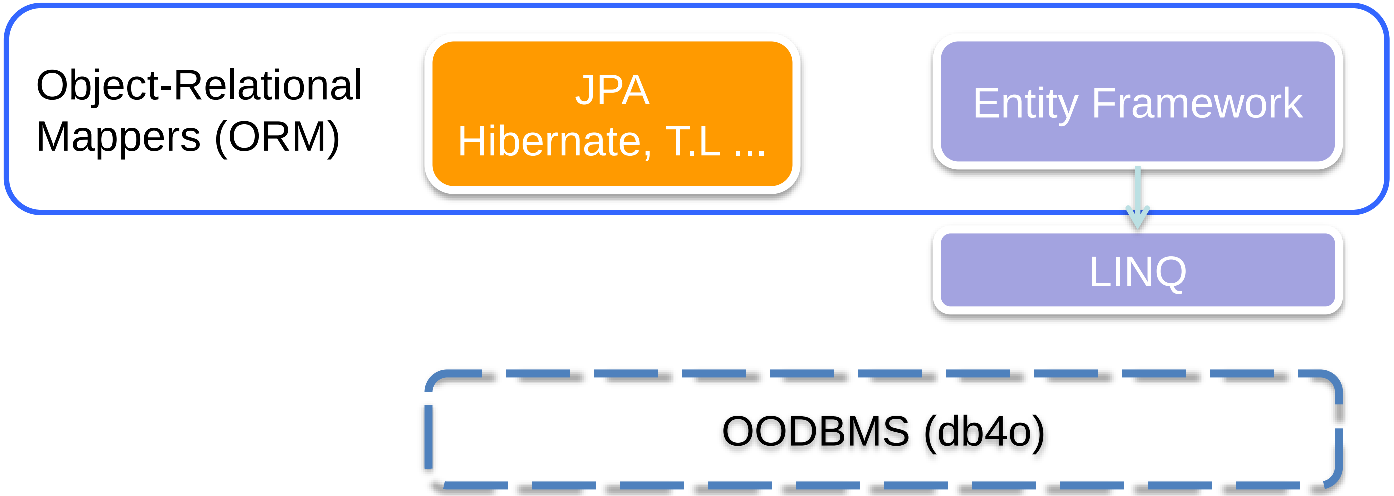
[IBM02] IBM. J2EE vs. .Net . http://www-01.ibm.com/software/smb/na/J2EE_vs_NET_History_and_Comparison.pdf

TECHNISCHE UNIVERSITÄT DARMSTADT

Characteristics	Microsoft .NET	Java EE
Language	C#, VB, C++.NET …	Java
Runtime	CLR	JVM
Server Components	.NET, COM+ Serv.	EJB
Scripting-Lang.	ASP.NET	JSP/Servlet
Data Access	ADO.NET …	JDBC …
	Entity Framework Nhibernate, …	JPA, Hibernate, …
Containers	.NET Runtime	EJB Container
Message Queuing	Sys.Msg. MSMQ	JMS
	COM+ QC	Message Beans

If the frameworks are not all that different how about the persistence?

Persistence Mechanisms



JPA – Java Persistence API

- Transparent object persistence - simple classes
  - EJB – same API with extended functionality
- Management of persistent objects
  - Inheritance, Aggregation, Composition
  - Update and merge
  - Change Tracking
  - Laden
- Query and find objects
- Object Caching
- Transactions

Example – POJO (Plain Old Java Object)

```
public class Employee {

   public Integer id;
   public String name;
   public long salary;
   public Department department;
            public Address address;

   public List<Project> projects;


}
```

- implement standard-constructor (no arguments)
- define Identity– primary key (p.key, id)
- do not use **final** → **Why?**
- Annotate fields or Setter/Getter-Methods

Discussed Example – POJO (Plain Old Java Object)

```
@Entity (name="Employee")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE | JOINED | TABLE_PER_CLASS)
public class Employee {

   @Id  //@GeneratedValue(strategy=TABLE | SEQUENCE | AUTO)
   public Integer id;

   @Column(name="NAME", table="Employee", unique=false, nullable=false, insertable=true,
   updatable=true)
   public String name;
   public long salary;

   @ManyToOne
   @Basic(fetch=FetchType.EAGER | LAZY)
   public Department department;

   @OneToOne(cascade = CascadeType.ALL | MERGE | PERSIST | REFRESH | REMOVE)
   public Address address;

   @ManyToMany(mappedBy = "emps")
   public List<Project> projects; //public class Project{ ... @ManyToMany List<Employee> emps;...}
}
```

Entity Manager

- Manages the lifecycle of entities
  - persist() – create new entity
  - remove() – delete entity
  - merge() – synchronize the state of detached entities
  - refresh() – update the state of an entity (load form DB)

- Managed and lose detached/unmanaged entities

- Persistent Context

Persist-Operation

```
public Order createEmployee( String name ) {

   // create new instance– transient
  Employee employee= new Employee( name );

   // state change to "managed". After the next flush or
   // commit changes are saved in DB
  entityManager.persist( employee );

  return employee;
}
```

Find and Remove Operations

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void removeEmployee(Long emplyoeeId) {

  // find returns null if no Employees are found
  Employee employee=    entityManager.find(Employee.class, employeeId);

   // instanced will be deleted after the next flush or commit
  // Access to deleted entities → undefine result
   if ( employee != null)
          entityManager.remove( employee );
}
```

Update of an Entity

- The persisitence context accepts the changes automatically
- Saves them in the DB

```
public Employee raiseEmployeeSalary(int id, long raise) {

  Employee emp = entityManager.find(Employee.class, id);

  if (emp != null) {
                  emp.setSalary( emp.getSalary() + raise);
  }

  return emp;
}
```

Merge Operation

```
public Employee updateEmployee( Employee emp) {

   // merge of the prsistent and transient state of an entity.
   return entityManager.merge( emp );

}
```

Queries

- Query language:
  - SQL
  - JPQL-Java Persistence Query Language

- Query createNamedQuery(String name)
  Query in JPQL or in native SQL.

- Query createNativeQuery(…)
  Query in native SQL statement: Query, Update Delete.

- Query createQuery(String sqlString)
  Query in JPQL.

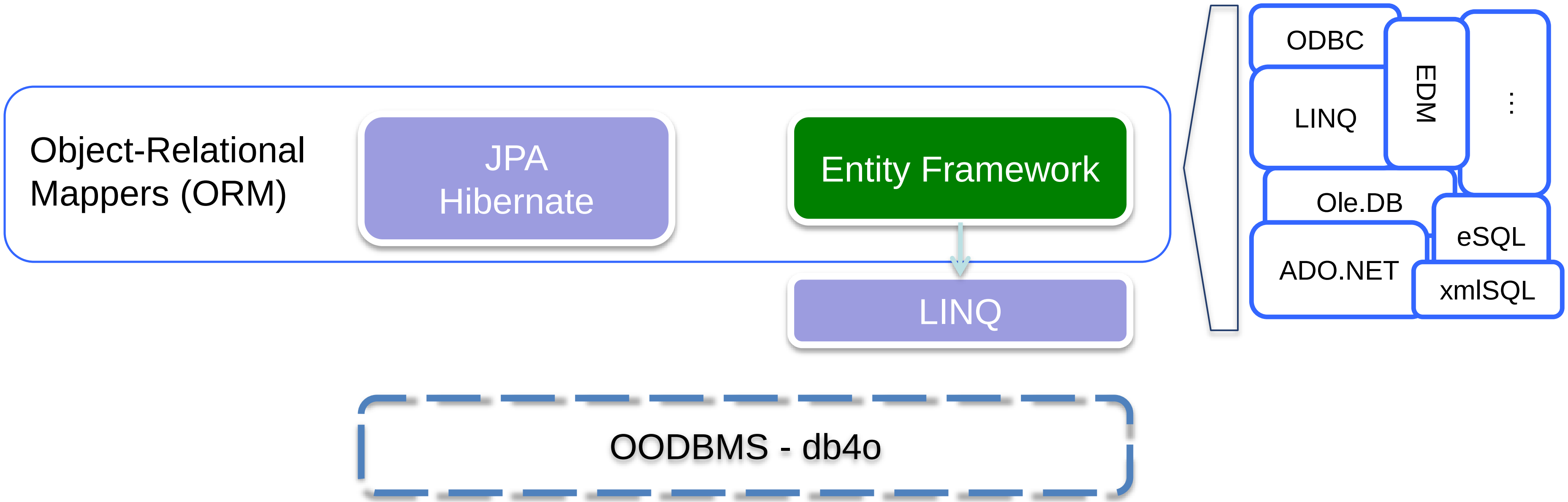
Querying Objects

```
EntityManager em = getEntityManager();

Query query=em.createQuery("SELECT e FROM Employee e");
Collection<Employee> emps = query.getResultList();


for (Iterator i = emps .iterator(); i.hasNext(); ) {
        Employee e = (Employee)i.next();  System.out.println(e);
}
```

Querying Objects

```
EntityManager em = getEntityManager();

Query query=em.createQuery(”SELECT e.name, e.department.name FROM
  Employee e“);


/* similar to :
Query query=em.createQuery("SELECT e.name, d.name FROM Employee e,
  Department d WHERE e.id = d.id“);
Query query=em.createQuery("SELECT e.name, d.name FROM Employee e inner
  join Department d“);
*/

List all = query.getResultList();
for (Iterator i = all.iterator(); i.hasNext();) {
        Object[] res = (Object[]) i.next();
        System.out.println(res[0] + " " + res[1]);
}
```

Persistence Mechanisms

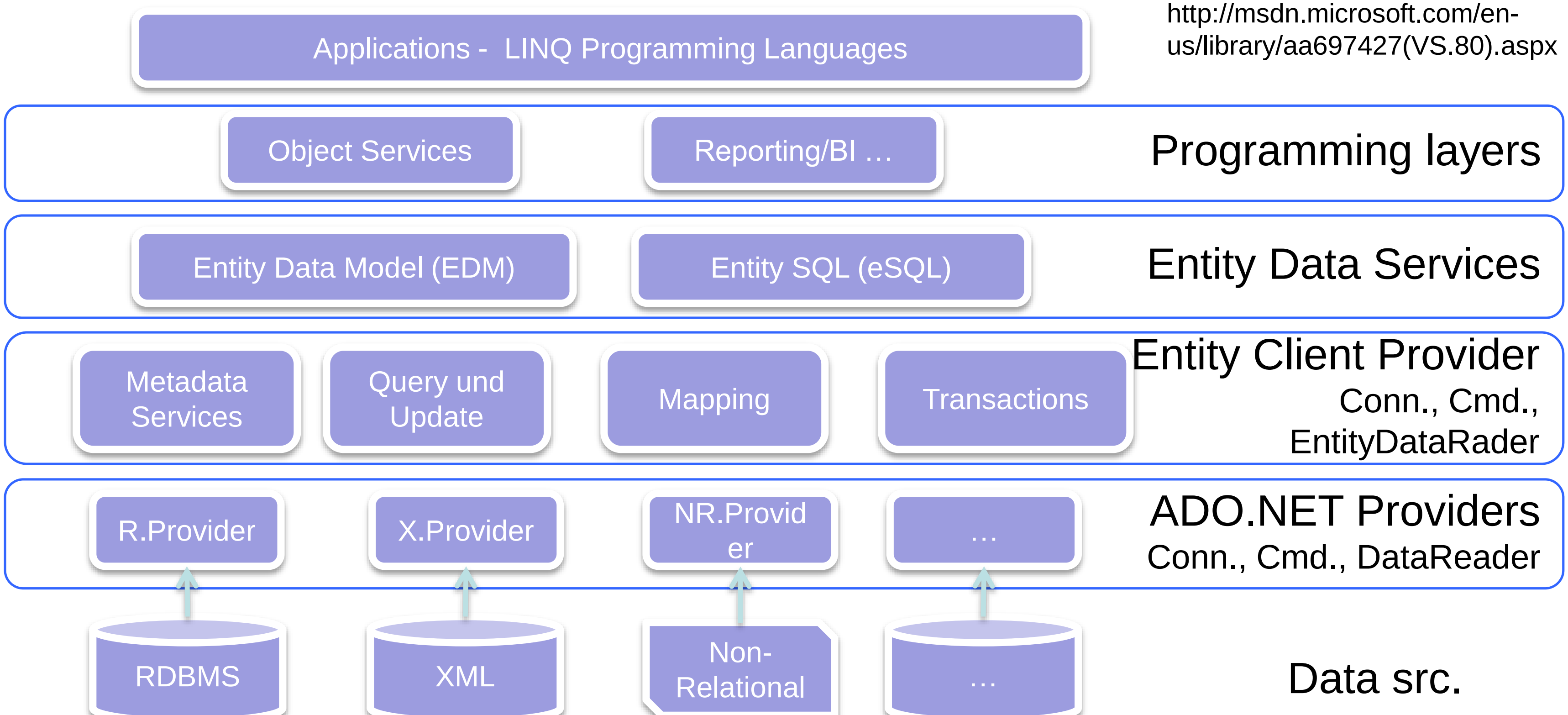


ADO.NET Entity Framework [ABM+07]

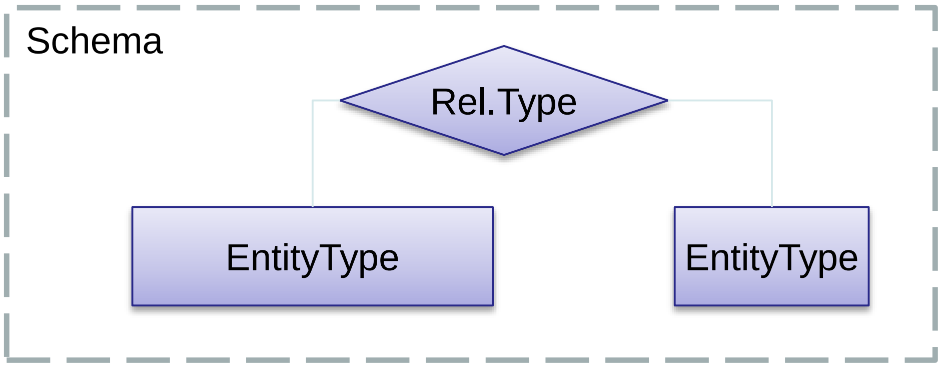
- Services for data intensive applications
  - Cross data source; access mechanisms; QL
  - Placed on conceptual level

[ABM+07] Adya, A., Blakeley, J. A., Melnik, S., and Muralidhar, S. Anatomy of the ADO.NET entity framework. *SIGMOD 2007*

[MS10] Microsoft Corp. Entitiy Framework Overview. http://msdn.microsoft.com/en-us/library/aa697427(VS.80).aspx

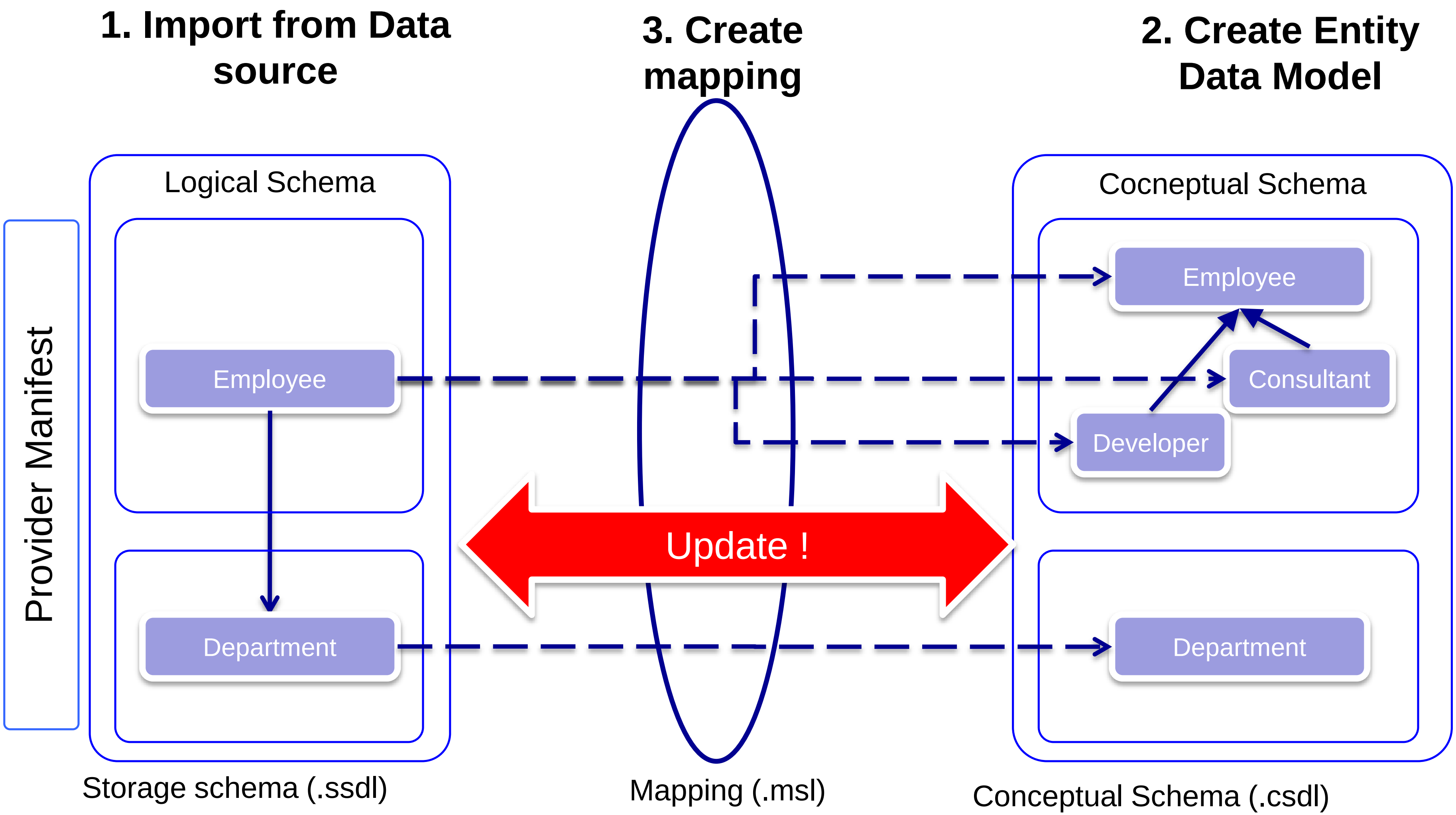


Terminology – Entity Data Model (EDM)



- EntityTypes
  - ID
  - Properies
    - Elementary Datatypes
    - Complex Datatypes
  - Single Inheritance
  - Instance: Entities
  - EntitySet – set of all instances
- Relationship Types (Association Types)
  - Connect 1 or N EntityTypes
  - Instance: Link (Association)
  - AssociationSet: Set of all associations
- Schemata
  - Applications → several Schemata

Mapping



1. Import from Data source
3. Create mapping
2. Create Entity Data Model
Provider Manifest
Logical Schema
Employee
Department
Cocneptual Schema
Employee
Consultant
Developer
Department
Update !
Storage schema (.ssdl)
Mapping (.msl)
Conceptual Schema (.csdl)

POCO – Plain Old Clr Object

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Department
{
        private int _ID;
        private string _Name;

        public int DeptID {
         get  { return this._ID;}
        }

        public string Name    {
      get  { return this._Name;}
      set  { this._Name = value;}
    }
  …
}
```

Private Properties

Getter-Method

Getter und Setter-Methods

Querying Entities – Entity SQL

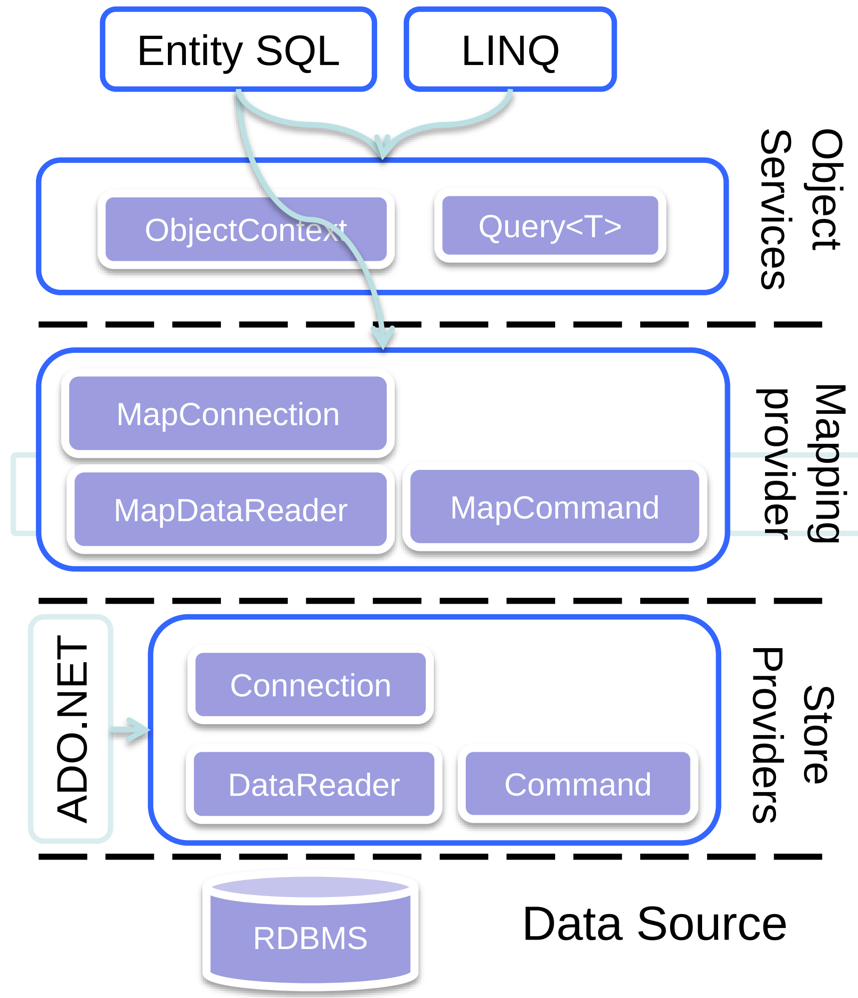
- Independent if Data Source
- Querying Entities (+complex Types)
- Associations, relationships

//over EDM as Entitäten

```
using(MapConnection con = new MapConnection(...) {
con.Open(); MapCommand cmd = con.CreateCommand();
cmd.CommandText="SELECT VALUE e
FROM Employee AS e       WHERE e.Dept.Name LIKE 'UB' ";
DbDataReader r = cmd.ExecuteReader();
 while(r.Read()) { Console.WriteLine("{0}", r["Name"]);} }
```

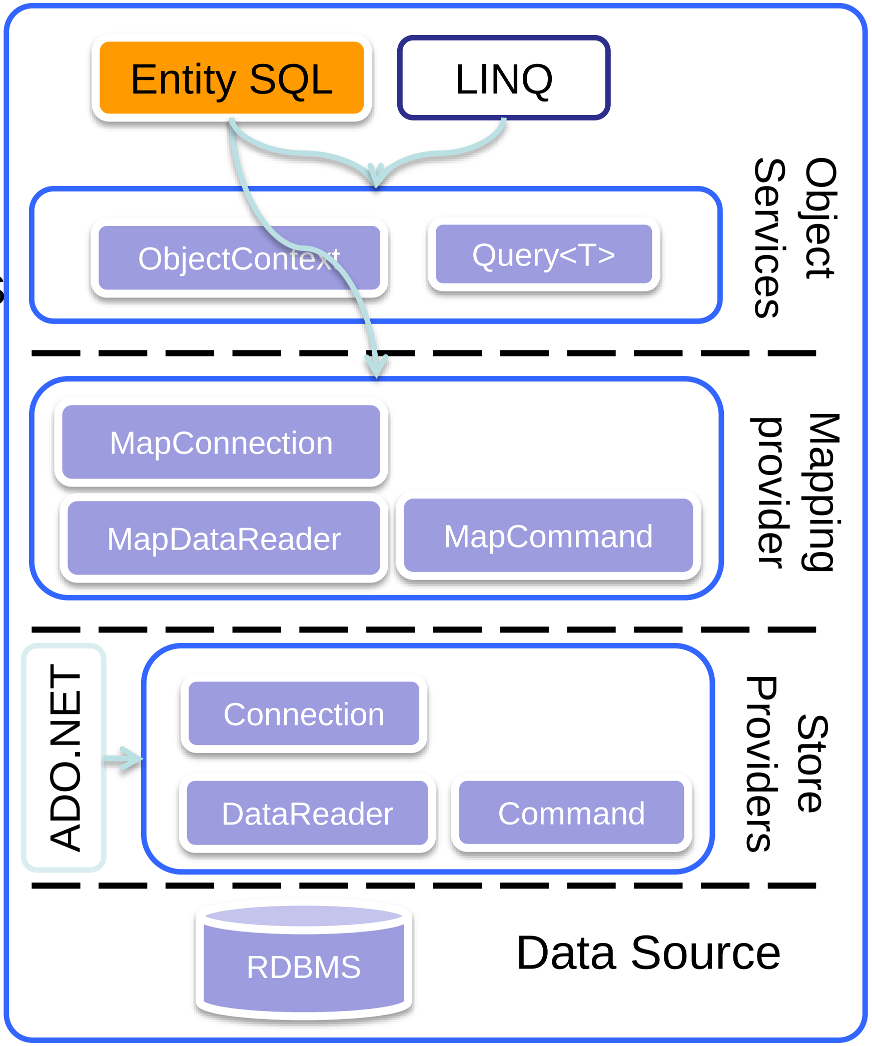
... // with Objectservices as Objectes

```
ObjectContext ctx= new ObjectContext(con);
Query<Employee> qN = ctx.GetQuery<Employee>(
                    "SELECT VALUE e FROM Employees AS e
                    WHERE e.Dept.Name LIKE 'UB'
                    WHERE e IS OF (Consultants));
foreach(Employee e in qN) { ... }
```

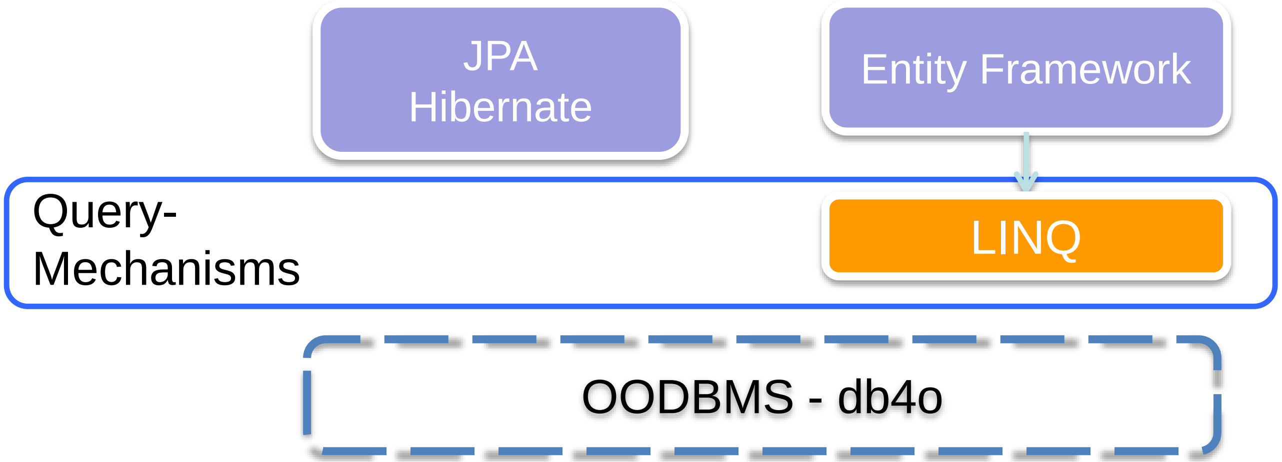


Updating Entities

```
Query<Employee>  emps=
ctx.GetQuery<Employee>(
            "SELECT VALUE e FROM Employees AS e
            WHERE e.Dept.Name LIKE 'Board‘);

 foreach(Employee e in emps) {
            e.Bonus += 10; e.Title = "Senior "+p.Title;
}

 edb.SaveChanges(); // Speichern in DB
}
```



Persistence Mechanisms



Problems with Query Languages

- If queries are defined as string then ...

SELEC * FROM employe;

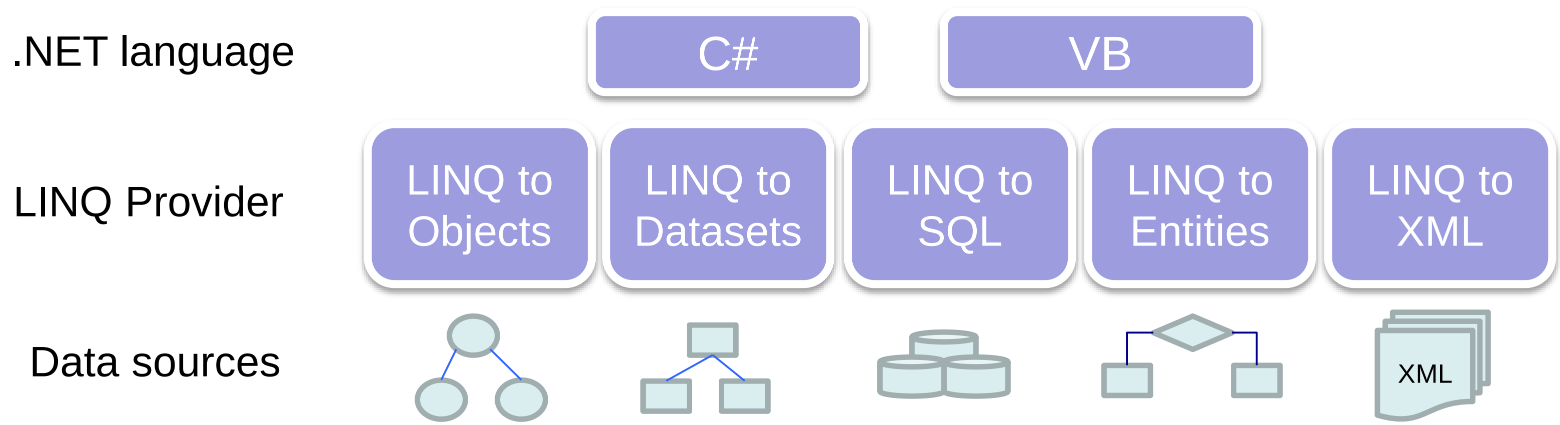
- Type - safety
- Semantic and Syntactic correctness
- Specific for each data source type

LINQ - Language INtegrated Query in .NET 3.5

- LINQ part ot .NET 3.5 (introduced earlier)
  - Java analoges

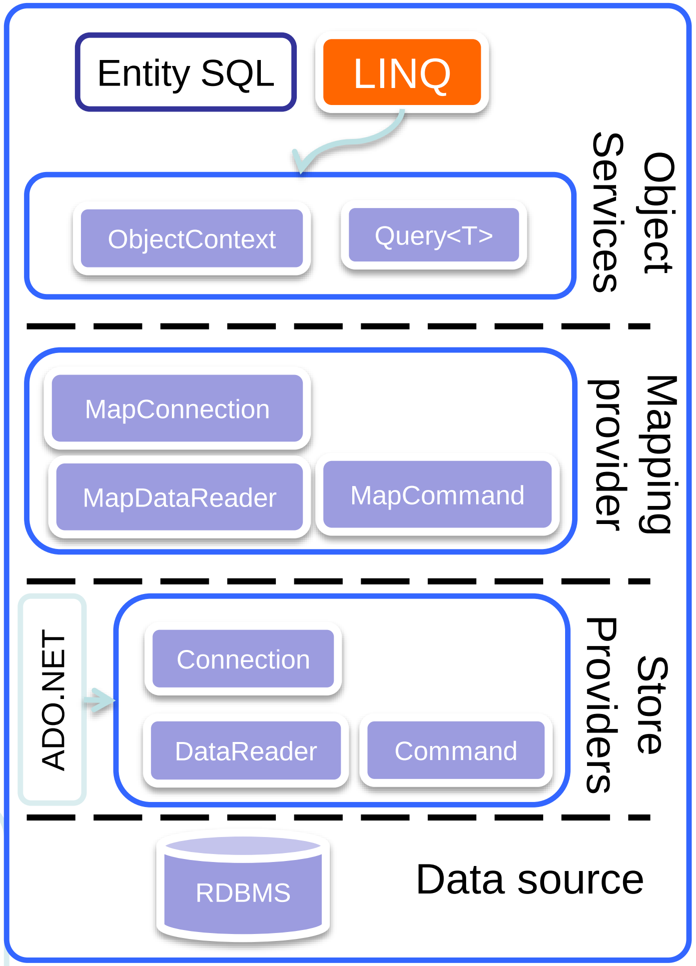
- LINQ creates type-safe queries. Compile time checks.
  - comcosable

- LINQ can be used with different data sources



LINQ to Entities

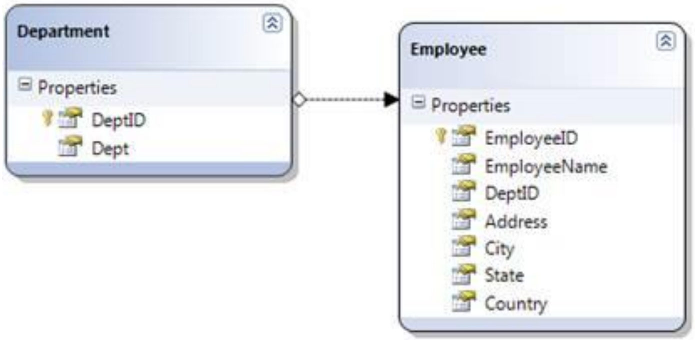
```
String deptName= “Board";
using(EmployeeDB edb = new EmployeeDB()) {

   var emps =          FROM e IN edb.Employee
                                  WHERE e.Dept.Name =
     deptName
                                  SELECT e;

   foreach(Employee e in emps) {
     Console.WriteLine("{0}", e.Name);
   }
}
```




```
//Query<Employee>  emps= ctx.GetQuery<Employee>(
//           "SELECT VALUE e FROM Employees AS e
//           WHERE e.Dept.Name = @deptName“ );
```

LINQ to SQL – create classes

- O/R modelled graphically:
  - Also porgrammable or declared



- Define in the code:

```
[Table(Name = "Department")]
public class Department {
		[Column(IsPrimaryKey
= true)]
	public string DeptID { get; set;
}
	[Column]
		public string Dept { get;
set; }
}
```

```
[Table(Name = "Employee")]
public class Employee {
   [Column(IsPrimaryKey = true)]
   public string EmployeeID; { get; set; }
   //...
   [Column]
   public string DeptID;
   private EntityRef<Department> _Department;
   [Association(Storage = "_Department", ThisKey =
"DeptID")]
   public Department Department {
      get { return this._Department.Entity; }
      set { this._Department.Entity = value; }
   }
}
```

LINQ to SQL

```
using System;
using System.Linq;

namespace App_Test {
    class Class1 {
        static void Main(string[] args) {
            EmployeeDataContext dcEmp = new EmployeeDataContext();

            var queryEmp = dcEmp.Emloyee;
                // Starke Typisierung:   Table<Employee> queryEmp = dcEmp.Employee;

            foreach( Employee emp In queryEmp ) {
                Console.WriteLine("{0} | {1} | {2}", emp.EmployeeID, emp.EmployeeName);
            }
        }
    }
}
```

```
var queryEmp = dcEmp.Emloyee;

SELECT[t0].[EmployeeID], [t0].[EmployeeName], ...
FROM [dbo].[Employee] AS [t0]
```

LINQ to SQL

- SPJ - query

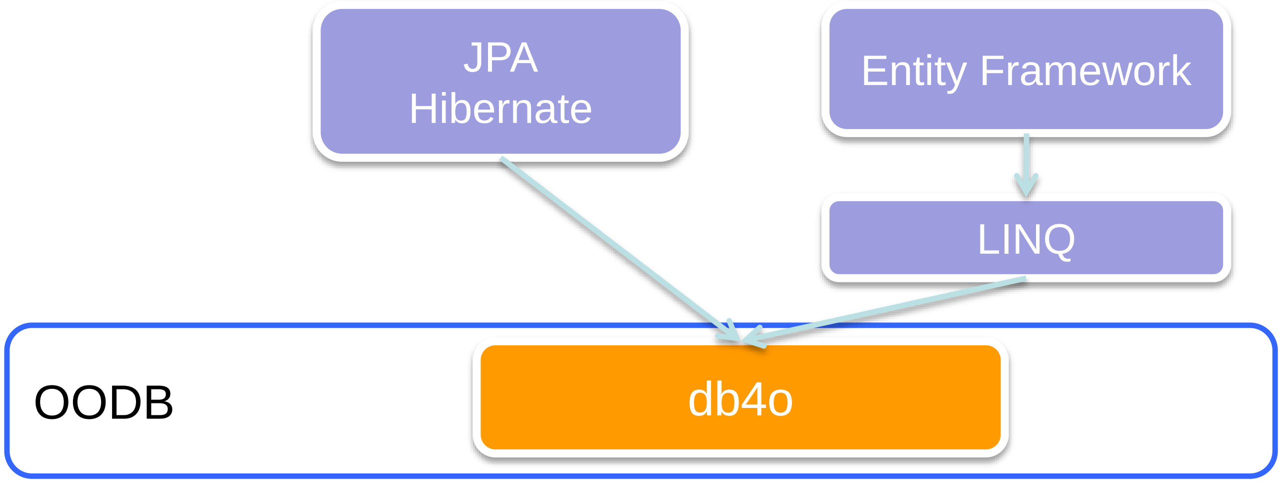
```
var qEmp = from e in dc.Employee
                    where e.EmployeeName.CompareTo("Otto")
                    where e.EmployeeID.StartsWith("O")
                    select e;
```

- join

```
var qEmp2 =         from e in dc.Employee
                    join d in dc.Department on e.DeptID equals d.DeptID
                    orderby e.DeptID
                    select new { e.EmployeeID, e.EmployeeName, d.DeptID, d.Dept};
foreach (var i in qEmp2) { ... }
```

- Supported operators
  - Projection, Selection, exists, groupping, aggregate functions,set-theoretic operators, sorting, ...

Persistence mechanisms



DB4O

- Simple! Java and .NET

- Functions
  - No Mapping required
  - No changes on existing classes → Why?
  - ACID Transactionen
  - Simple API
  - Transparent persistence
  - ...

Object handling

```
ObjectContainer database = Db4o.openFile("test.db");

Employee employee = new Employee();

employee.setName("Ilia");

database.set(employee);
```

Find and Query OBjects

- Three Alternatives

1. Query by Example Employee empPrototype = new Employee(); empPrototype.setName(“Ilia"); Set<Employee> emps = database.get(empPrototype);

2. Native Query Set<Employee> emps = database.query(new Predicate<Employee>() { public boolean match(Employee employee) return employee.getName().equals(“Ilia"); } });

- Simple Object Data Access (SODA)
  Query query = database.query(); query.constrain(Employee.class);
  query.descend("name").constrain(“Ilia").equals();
  Set<Employee> emps = query.execute();

Change Objects

```
ObjectContainer database = Db4o.openFile("test.db");

Employee prototype = new Employee();

prototyp.setName(“Ilia");

Employee employee = (Employee) database.get(prototyp).next();

employee.setName("Peter");

database.set(employee);
```

- Delete

```
database.delete(employee);
```

Cascading persistence and activation

- Cascading operations defined per class
- Update depth configurable
- Global Configuration
- Activation depth configurable in the case of deep hierarchies

Characteristics

- ACID compatible + isolation levels
- Logging
- db4o Core ist threadsafe
- ObjectContainer is always transactional
  - Transaction starts with openning of Container
  - Tx ends with closing the container
  - Commit and Rollback explicit
- Atomicity
- Indices – B-Trees
- Embedded or distributed
- Changes /Schema Evolution
  - Insert and delete field
  - Data is saved not structures
  - Event handler

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- CORBA and Distributed Objects
- Components and Containers
- Application Servers
- ORM – Object Relational Mappers

Thank You!

Questions?