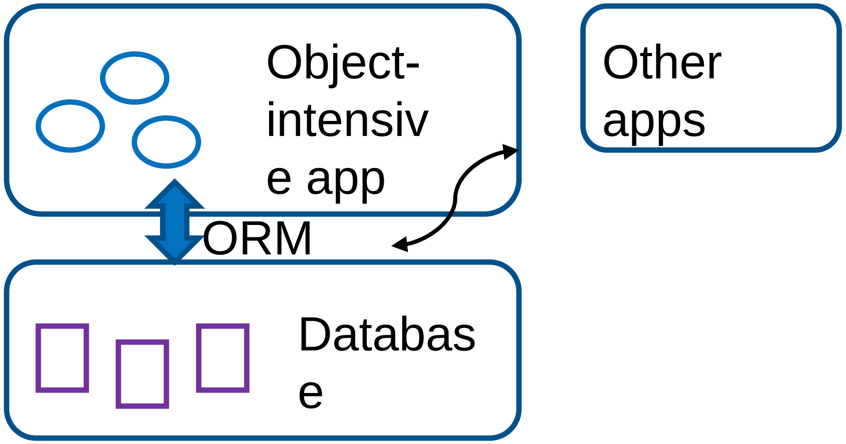
Object/Relational Mapping

Love of Objects ♥ ♥

- Programmers love objects
- Objects are normally ephemeral
- Programming languages provide object persistence, but it is fragile
- Databases provide robust data persistence.
- So… need way to persist object data in the database
- Or think bigger: use data model across object-DB boundaries.

Object-Relational Mapping (ORM)

- A software system that shuttles data back and forth between database rows and objects
- Appears as a normal database user to the database
- Can share the database and tables with other apps



Object-Relational Mapping

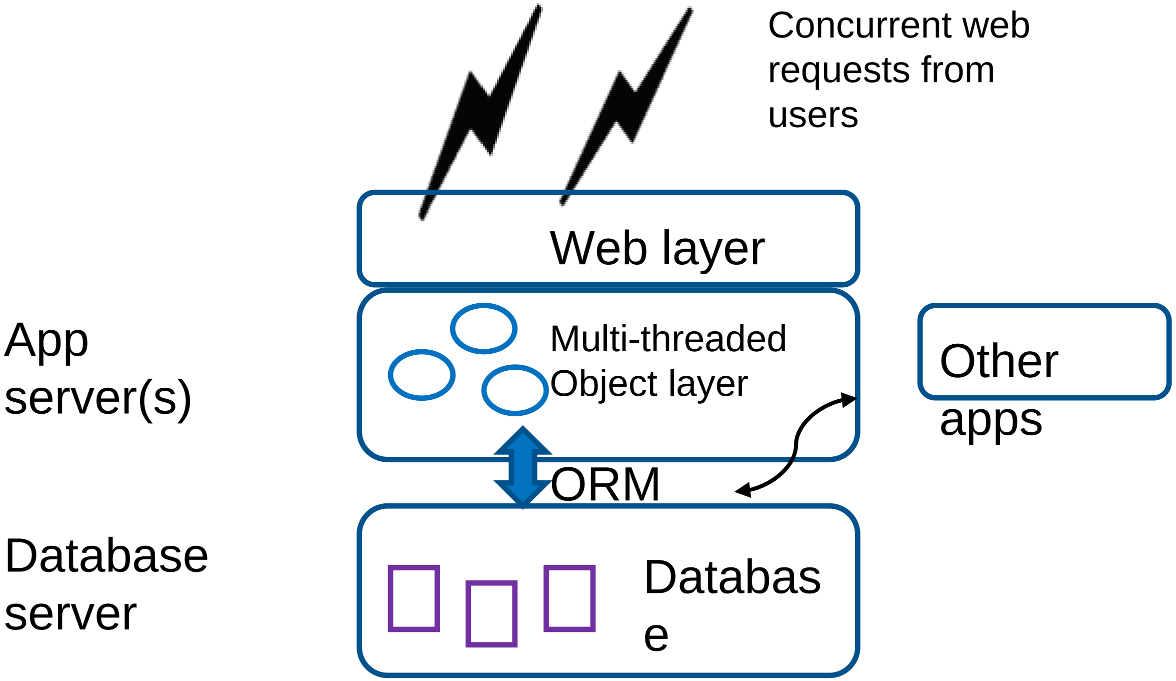
- Has a history, but widely adopted only since open-source Hibernate project started, 2001-2002.
- The Hibernate project [7] was founded and led by Gavin King, a Java/J2EE software developer, now part of Jboss. King wrote an excellent book [3].
- Microsoft has adopted a comparable approach with EDM, Entity Data Model and its Entity Framework, for release this year. [1, 10]
- Both Hibernate and EDM support (or will support) multiple databases: Oracle, DB2, SQL Server, …

Java Persistence Architecture (JPA)

- JPA is part of current JEE (previously J2EE), Sun's Java Enterprise Edition
- JPA is a standardized version of the Hibernate architecture
- EJB 3 (current Entity Java Beans) uses JPA for EJB persistence, i.e., persistence of "managed" objects
- JPA and EJB 3 are now available in major application servers: Oracle TopLink 11g, OpenJPA for WebSphere and WebLogic, Hibernate JPA for JBoss
- JPA can be used outside EJB 3 for persistence of ordinary objects, as Hibernate is used in this talk

Current ORM Impetus: the web app

A web app, with its multi-threaded object layer, particularly needs help with the correct handling of persistent data



Concurrent web requests from users

Web layer

App server(s)

Multi-threaded Object layer

Other apps

ORM

Database server

Databas e

Our Universe of Discourse

- Object-oriented web apps with database backed data
- Let's consider sites with
  - Possibly many application servers, where the objects live
  - A single database server with plenty of CPUs and disks
- Given today's fast machines and databases, this configuration scales up to many 100s of transactions/second (over 1 M Tx/hour)
- We will concentrate on common characteristics of Hibernate and EDM

Data Modeling

- Three modeling methodologies:
  - We all know the venerable Chen E-R models for database schemas; the extended E-R models (EER) incorporate inheritance
  - The object modeling approach uses UML class diagrams, somewhat similar to EER
  - The tables of the database define the “physical” schema, itself a model of underlying resources
- The relationship between these models involves schema mapping, covered in SIGMOD’s keynote talk by Phil Bernstein[9]

Even simple cases need help

- In the simplest case, a program object of class A has fields x, y, z and a table B has columns x, y, z
  - Each instance of A has a row in B and vice versa, via ORM
  - Are we done?
  - If x is a unique id, and x, y, and z are simple types, yes.
  - --Or some unique id in (x, y, z), possibly composite
- If no unique id in (x, y, z), the object still has its innate identity, but corresponding rows involve duplicate rows, against relational model rules
- So in practice, we add a unique id in this case:
- Class A1 has id, x, y, z and table B1 has id, x, y, z

Persistent Objects & Identity

- A “persistent object” is an ordinary program object tied via ORM to database data for its long-term state
- The program objects come and go as needed
- Don’t confuse this with language-provided persistence (Java/C#), a less robust mechanism
- Persistent objects have field-materialized identity
- It makes sense—Innate object identity depends on memory addresses, a short-lived phenomenon
- So long-lived objects (could be years…) have to be identified this way, it’s not the database’s fault

Persistent Objects need tracking

- We want only one copy of each unique object in use by an app, a basic idea of OO programming
- Each persistent object has a unique id
- We can no longer can depend on object location in memory to ensure non-duplication
- So we have a harder problem than before—need an active agent tracking objects
- This agent is part of ORM's runtime system
- The ORM uses hashing to keep track of ids, detect duplicates

ORM Entities

- Like E/R entities, ORM entities model collections of real-world objects of interest to the app
- Entities have properties/attributes of database datatypes
- Entities participate in relationships—see soon (but relationships are not “first class” objects, have no attributes)
- Entities have unique ids consisting of one or more properties
- Entity instances (AKA entities) are persistent objects of persistent classes
- Entity instances correspond to database rows of matching unique id

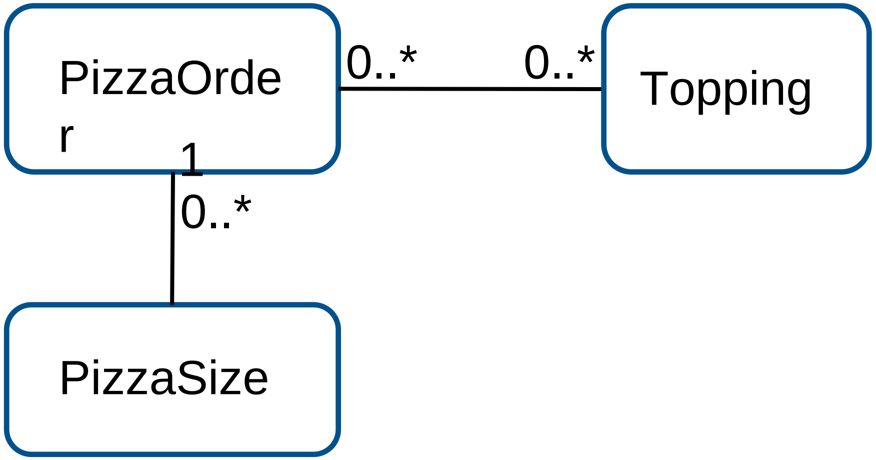
Value Objects

- In fact, persistent objects can be entities or value objects
- Value objects can represent E/R composite attributes and multi-valued attributes
- Example: one address consisting of several address attributes for a customer.
- Programmers want an object for the whole address, hanging off the customer object
- Value objects provide details about some entity, have lifetime tied to their entity, don’t need own unique id
- Value objects are called Hibernate “components”, EDM “complex types”
- We’ll only discuss entities for persistent objects
- For this presentation, persistent object = entity object

Creating Unique IDs

- A new entity object needs a new id, and the database is holding all the old rows, so it is the proper agent to assign it
- Note this can't be done with standard SQL insert, which needs predetermined values for all columns
- Every production database has a SQL extension to do this
  - Oracle's sequences
  - SQL Server's auto-increment datatype
  - …
- The ORM system coordinates with the database to assign the id, in effect standardizing an extension of SQL
- Keys obtained this way have no meaning, are called "surrogate keys"
- Natural keys can be used instead if they are available.

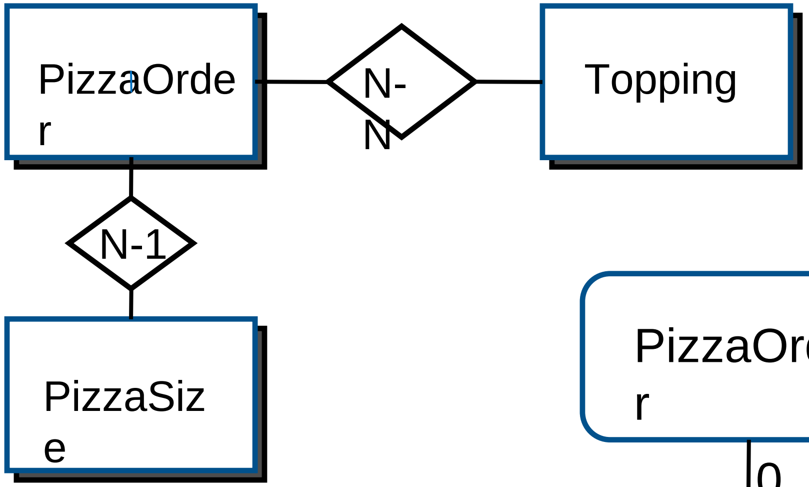
Entity Model



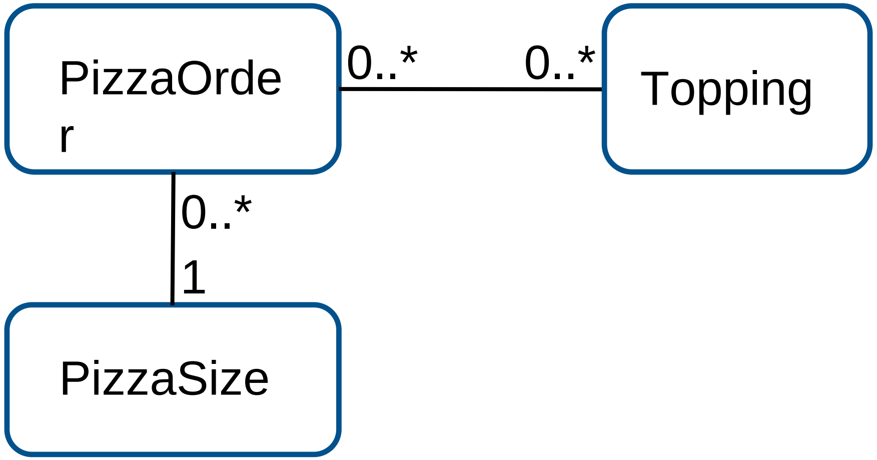
- Uses UML-like diagrams to express object models that can be handled by this ORM methodology
- Currently handles only binary relationships between entities, expects foreign keys for them in database schema
- Has a SQL-like query language that can deliver entity objects and entity object graphs
- Supports updates and transactions

Classic Relationships

A PizzaOrder has a PizzaSize and a set of Toppings



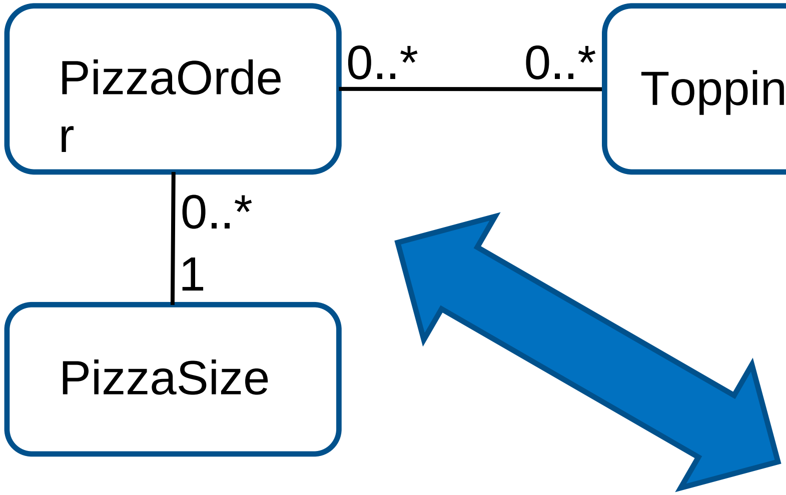
E-R diagram



UML class diagram or entity model: no big diamonds, type of relationship is inferred from cardinality markings

Classic Relationships

Schema mapping, entities to tables and vice versa



pizza_order

id	sizeid (FK)	room	status

pizza_size

id	name

topping

id	name

order_topping

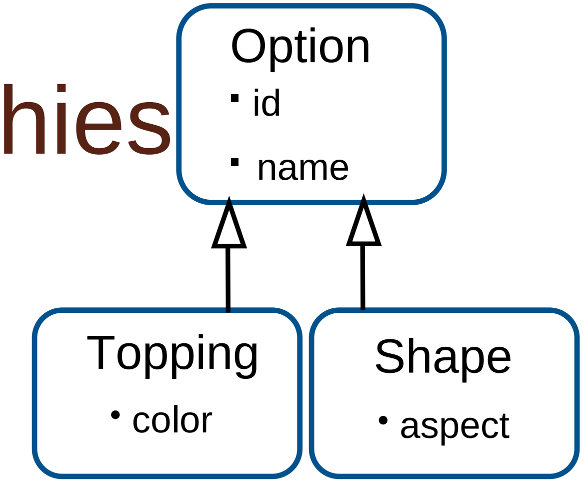
orderid (FK)	toppingid (FK)

Needed database schema: has one table for each entity, plus a link table for N-N relationship

Inheritance

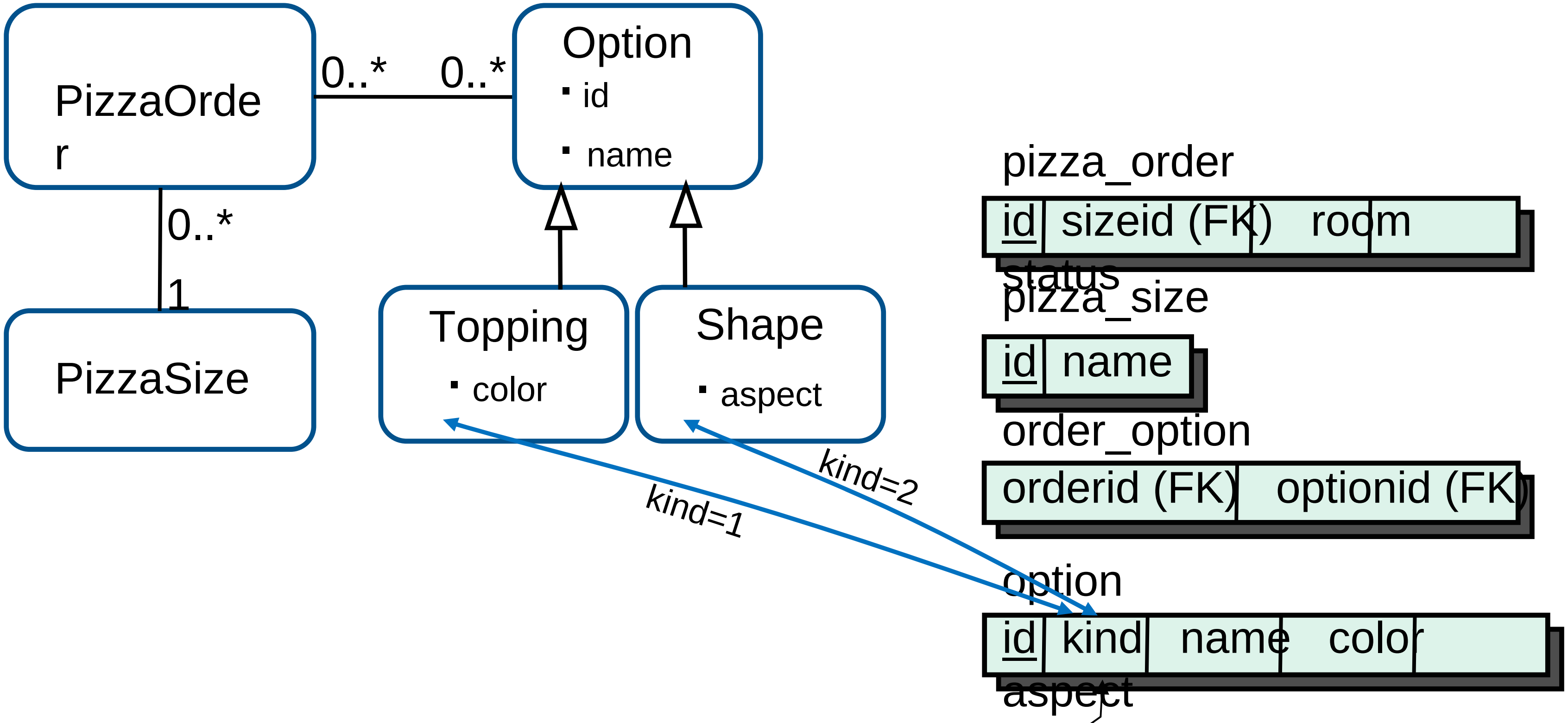
- Example: generalize Topping to PizzaOption, to allow other options in the future:
  - Topping ISA PizzaOption
  - Shape ISA PizzaOption, …
- Then a PizzaOrder can have a collection of PizzaOptions
  - We can process the PizzaOptions generically, but when necessary, be sensitive to their subtype: Topping or Shape
  - It is important to have “polymorphic associations”, such as PizzaOrder to PizzaOption, that deliver the right subtype object when followed.
- Inheritance is supported directly in Java, C#, etc., ISA “relationship”
- Inheritance is not native to RDBs, but part of EER, extended entity-relationship modeling, long-known schema-mapping problem

Inheritance Hierarchies



- Both Hibernate and EDM can handle inheritance hierarchies and polymorphic associations to them
- Both Hibernate and EDM provide single-table and multiple-tables per hierarchy solutions
  - Single-table: columns for all subtypes, null values if not appropriate to row’s subtype
  - Multiple-table: table for common (superclass) properties, table for each subclass for its specific properties, foreign key to top table
  - Also hybrid: common table plus separate tables for some subclasses

Inheritance Mapping (single table)



pizza_order

<u>id</u>	sizeid (FK)	room	status

pizza_size

<u>id</u>	name

order_option

orderid (FK)	optionid (FK)

option

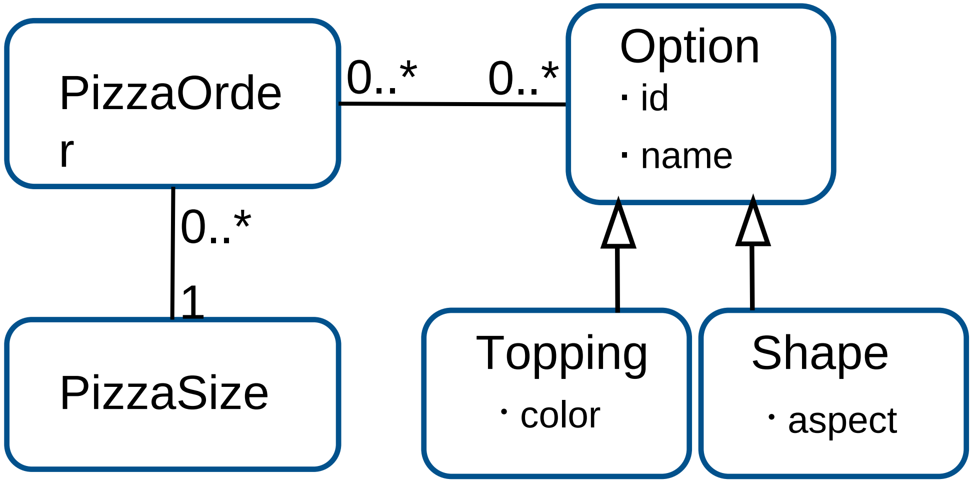
<u>id</u>	kind	name	color	aspect

Discriminator column to specify subtype
(not seen in object properties)

Inheritance using a single table

- The discriminator column (here “kind”) is handled by the O/R layer and does not show in the object properties
- The hierarchy can have multiple levels
- Single-table approach is usually the best performaning way
- But we have to give up non-null DB constraints for subtype-specific properties
- Alternatively, use multiple tables…

Inheritance Mapping (3 tables)



pizza_order

id	sizeid (FK)	room	status

pizza_size

id	name

order_option

orderid (FK)	optionid (FK)

option

id	name

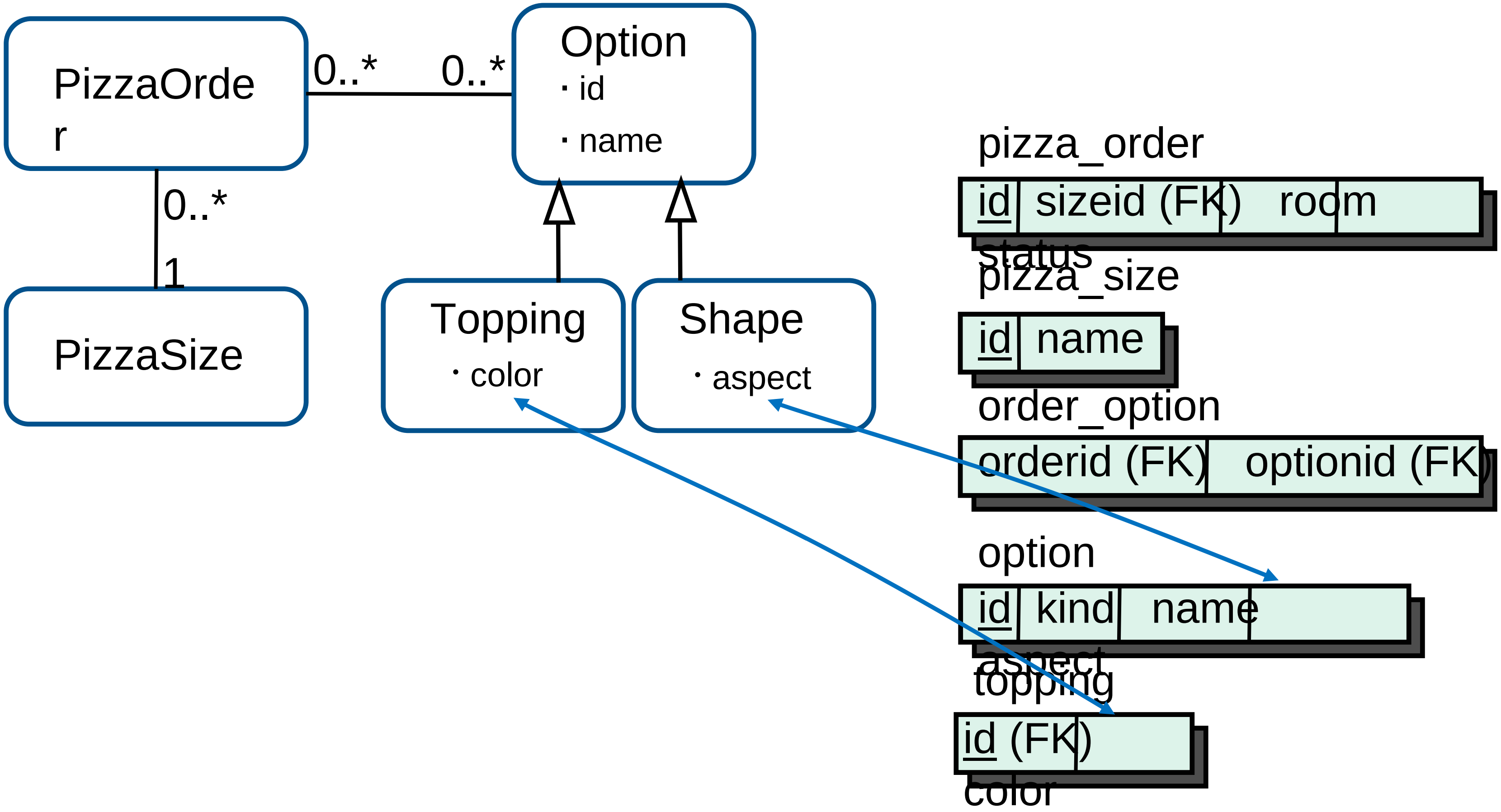
topping

id (FK)	color

shape

id(FK)	aspect

Inheritance Mapping (hybrid)



pizza_order

id | sizeid (FK) | room | status

pizza_size

id | name

order_option

orderid (FK) | optionid (FK)

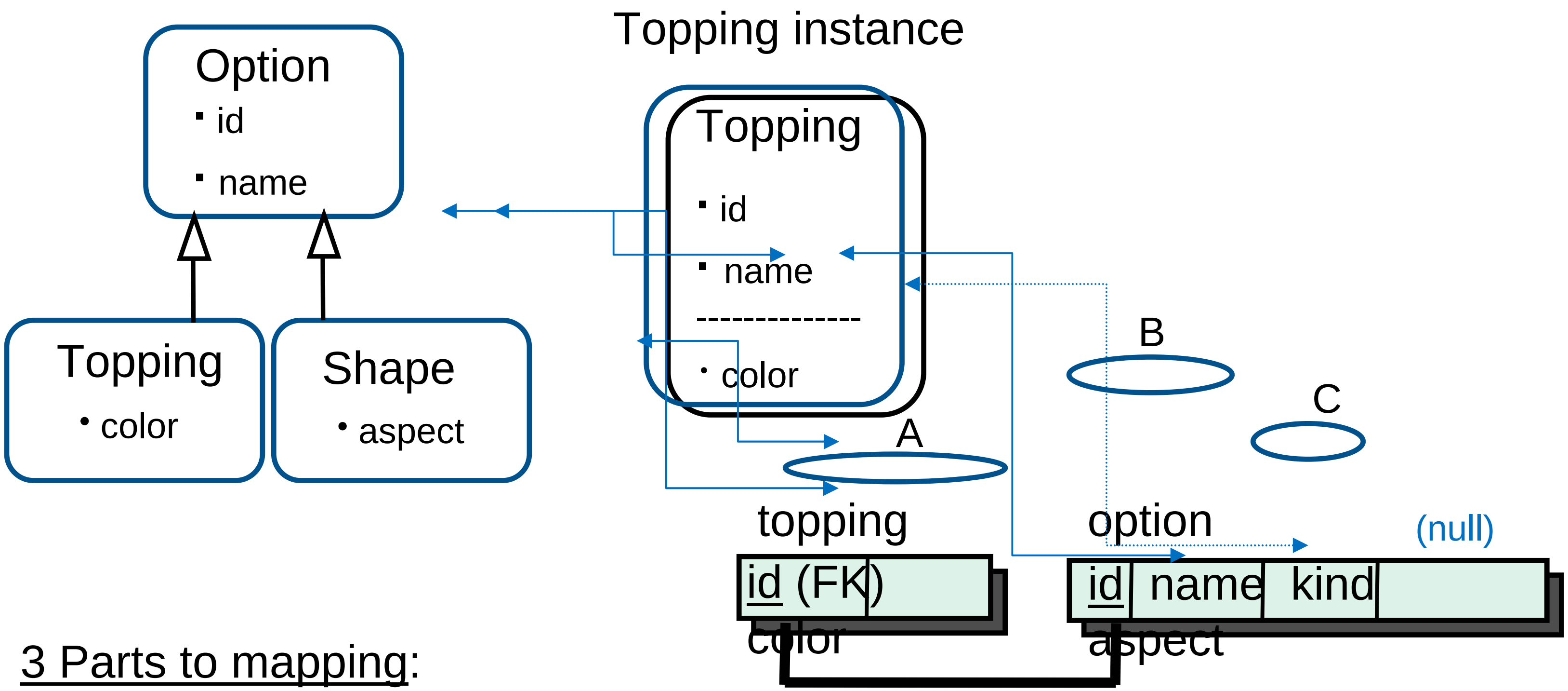
option

id | kind | name | aspect

topping

id (FK) | color

A Mapping dissected



Rows of same key, kind=1

<u>3 Parts to mapping</u>:
B base class mapping
A topping table for Topping
C option table for Topping
Together, 2-way mapping

Queries for views

Query for Topping content is basically

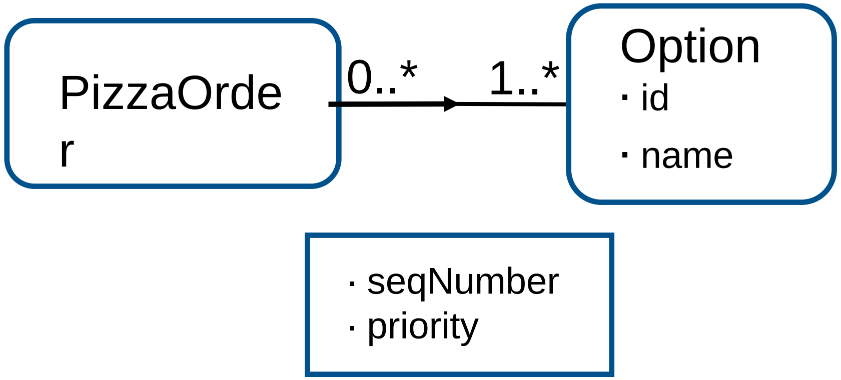
```
select id, name, color
    from (select * from option where kind=1)
        join topping on <unique key match>
```

Query for Option content:

```
select … from option left outer join topping on
                              <unique key match>
```

These could be the bases of updatable views in the database: the FKs are there

Example of an object model that doesn't fit current ORM



- Case of attributes of the N-N relationship
- We know how to do this in the database…
- We can introduce a new entity in the middle but lose crispness
- Related to problem of ternary relations, also missing

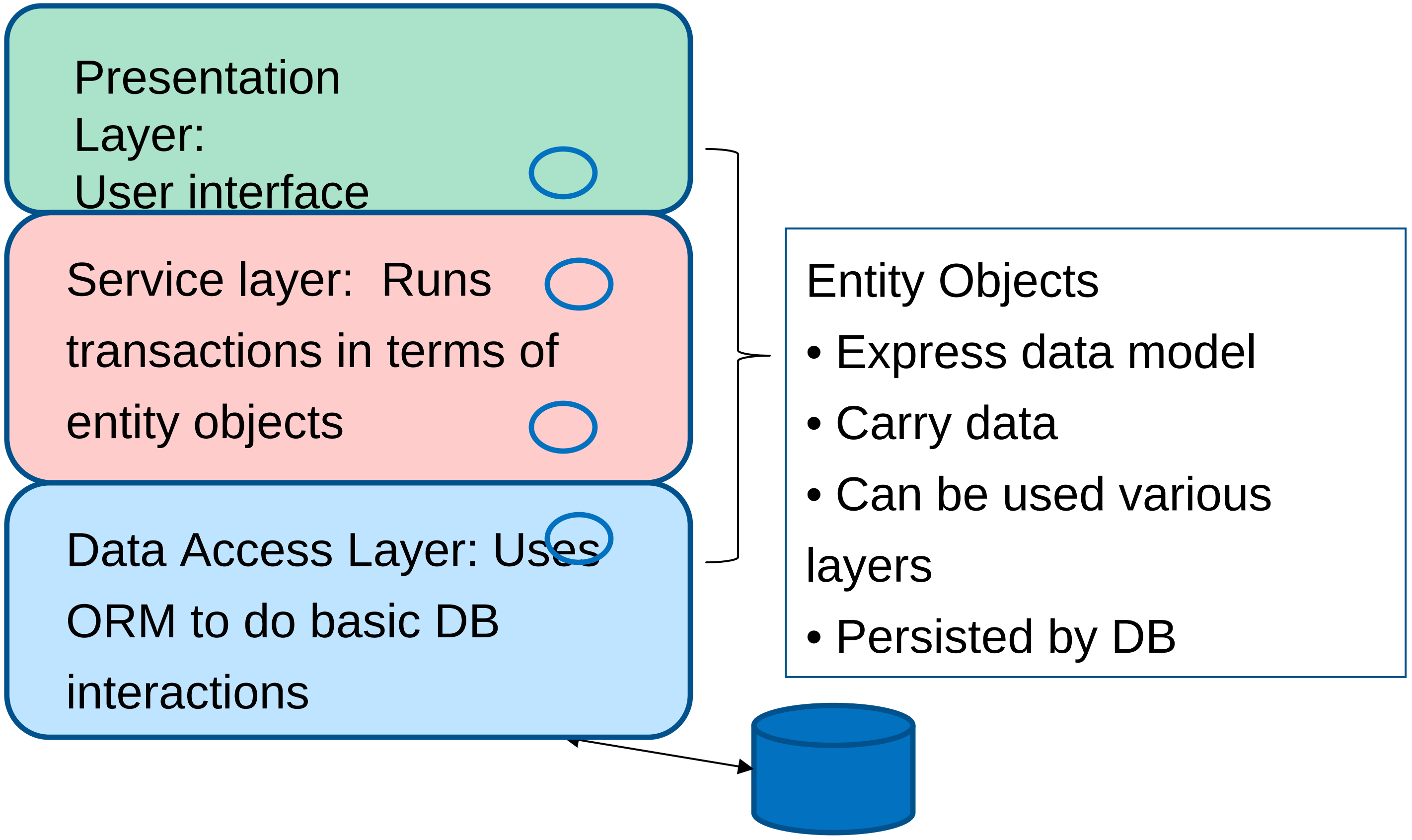
The Pizza Shop Example

- Free pizza for students in a dorm
- Student can:
  - Choose toppings, size, order by room number
  - Check on orders by room number
- Admin can:
  - Add topping choices, sizes
  - Mark the oldest pizza order “done”
- Available at www.cs.umb.edu/~eoneil/orm

The Pizza Shop Example

- Implemented using Hibernate and Microsoft EDM: same architecture, similar code, same database schema
- Implemented as client-server app and web app: only the top-level code changes
  - Client-server means all the app code runs on the client, with network connection to DB
  - Web app means all the app code runs on the web server/app server, clients use browser, DB can be on another server.
- Transactions are explicitly coded, not using container-managed transactions (EJB/COM+ Serviced Components)

The Pizza Shop: layers



Presentation Layer:
User interface

Service layer: Runs transactions in terms of entity objects

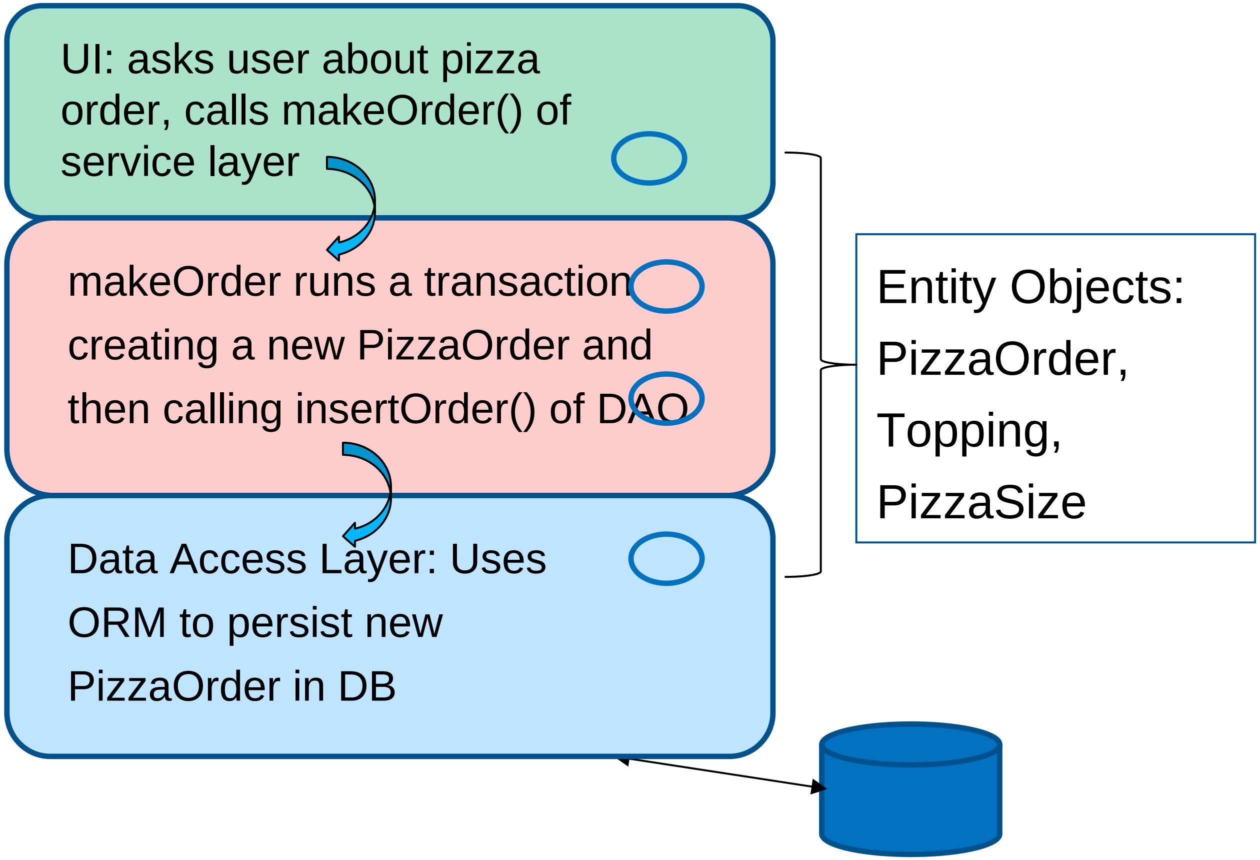
Data Access Layer: Uses ORM to do basic DB interactions

Entity Objects

- Express data model
- Carry data
- Can be used various layers
- Persisted by DB

Note: layers are not required by ORM, they are just a good idea for such apps

The Pizza Shop: objects, calls



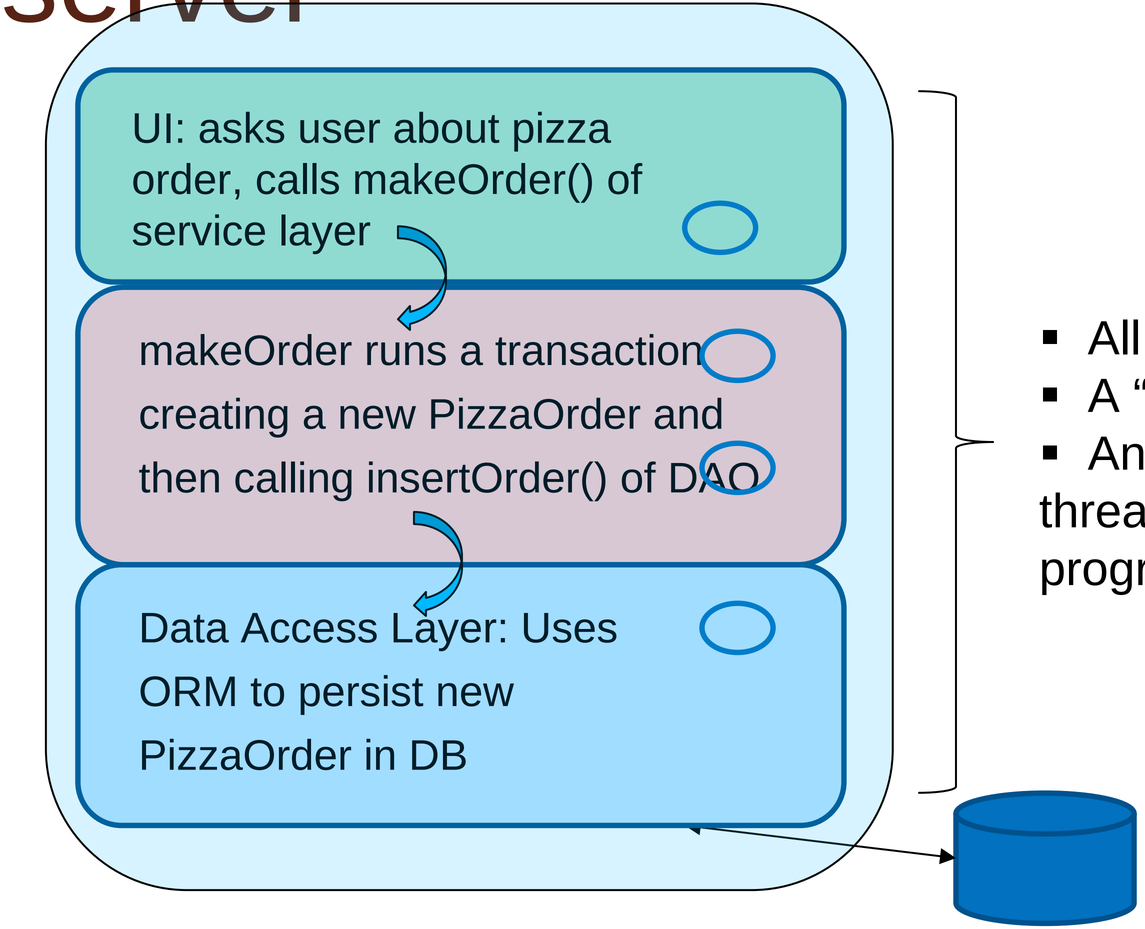
UI: asks user about pizza order, calls makeOrder() of service layer

makeOrder runs a transaction creating a new PizzaOrder and then calling insertOrder() of DAO

Data Access Layer: Uses ORM to persist new PizzaOrder in DB

Entity Objects: PizzaOrder, Topping, PizzaSize

The Pizza Shop: Client-server



UI: asks user about pizza order, calls makeOrder() of service layer

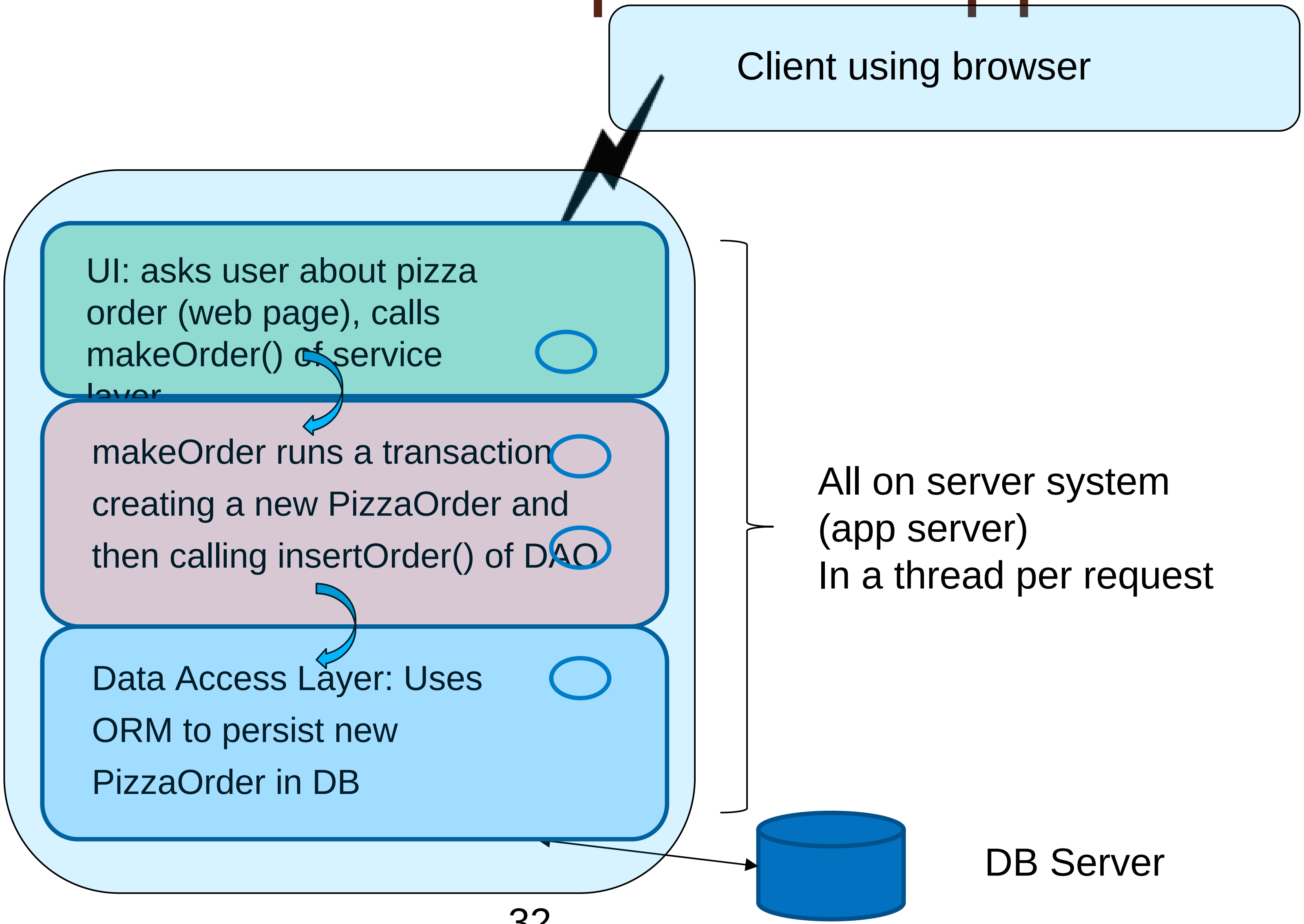
makeOrder runs a transaction creating a new PizzaOrder and then calling insertOrder() of DAO

Data Access Layer: Uses ORM to persist new PizzaOrder in DB

- All on client system
- A “rich client”
- An ordinary (single-threaded) Java/C# program

DB Server

The Pizza Shop: Web app



Client using browser

UI: asks user about pizza order (web page), calls makeOrder() of service layer

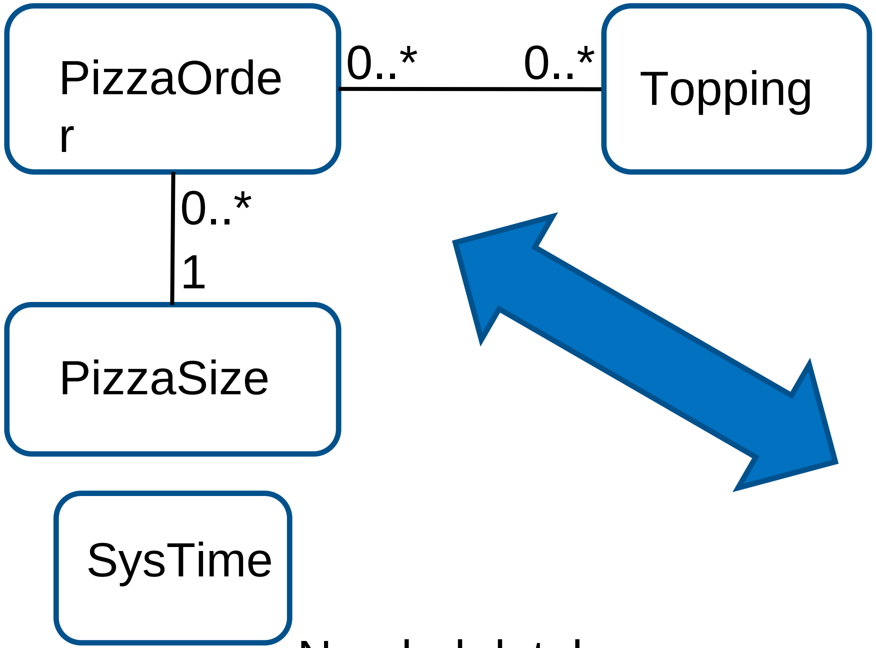
makeOrder runs a transaction creating a new PizzaOrder and then calling insertOrder() of DAO

Data Access Layer: Uses ORM to persist new PizzaOrder in DB

All on server system (app server)
In a thread per request

DB Server

Pizza Shop Entities, Mapping



Needed database schema: has one table for each entity, plus a link table for N-N relationship

pizza_order

id	sizeid (FK)	room	status

pizza_size

id	name

topping

id	name

order_topping

orderid (FK)	toppingid (FK)

sys_time

id	current_day	report_day

Any Question?

Elizabeth (Betty) O’Neil
Dept. of Computer Science
University of Massachusetts
Boston