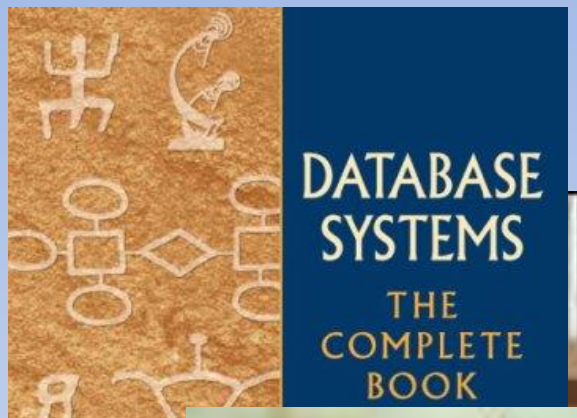
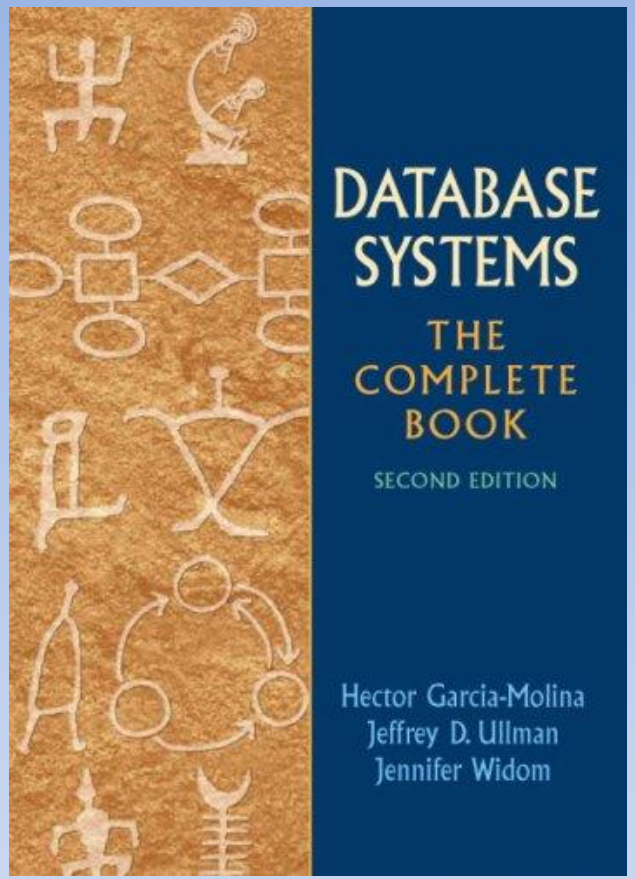
Database Systems: The Complete Book(3rd)

by Hector Garcia-Molina,
Jeffrey D. Ullman & Jennifer Widom





Designing Databases

- Chapter 4: Entity/Relationship Model
  - High-Level Database Models
  - E/R Diagrams

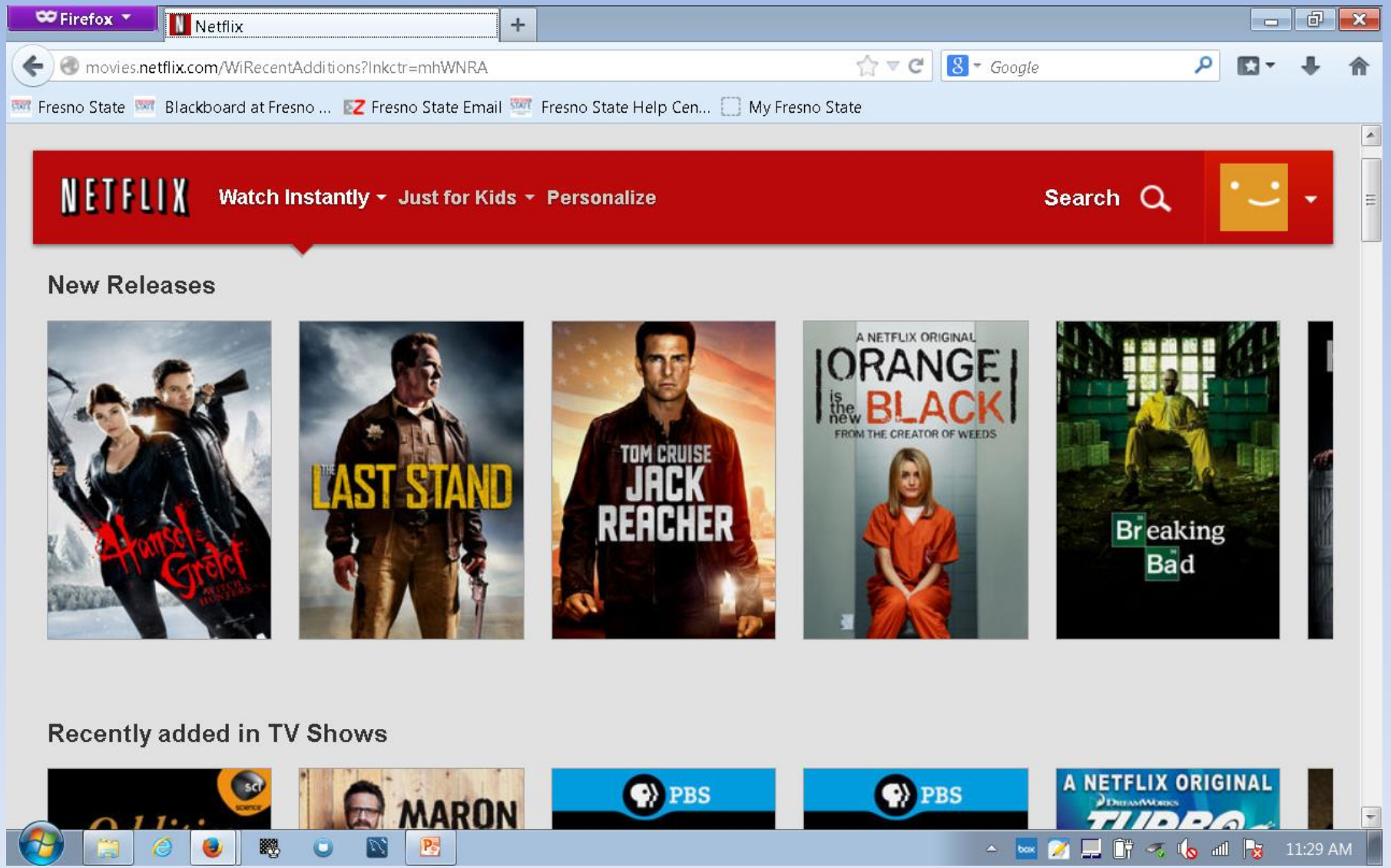
Database Design Process

1. Ideas
2. High-Level Design
3. Relational Database Schema
4. Relational DBMS

Entity-Relationship Model

- Tool for allowing us to sketch out Database Design.
- Designs are pictures called ER Diagrams.
- Easy to convert ER Diagrams to Relational Schema.

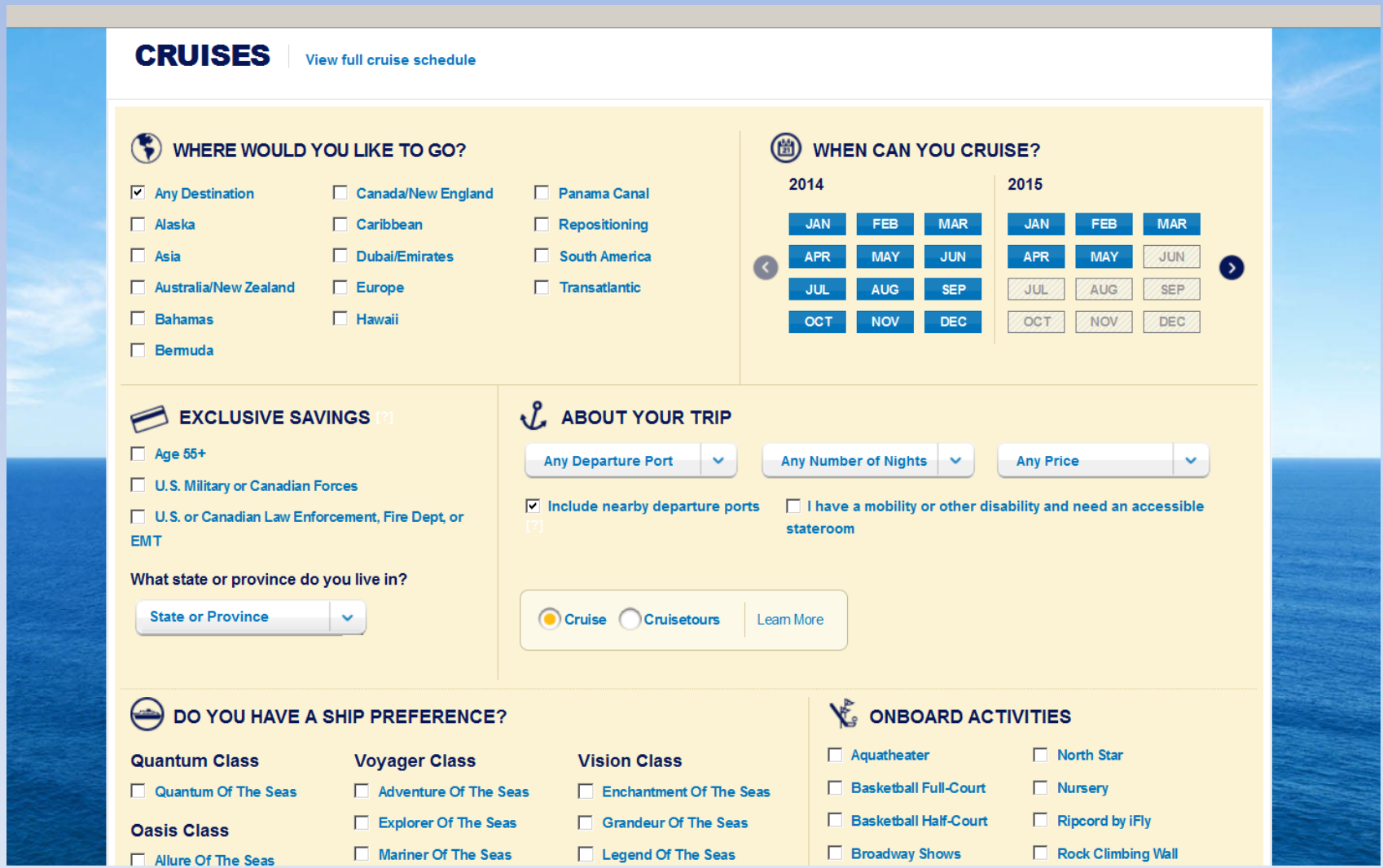
Entity


Firefox
Netflix
movies.netflix.com/WiRecentAdditions?lnkctr=mhWNRA
Google
Fresno State
Blackboard at Fresno ...
Fresno State Email
Fresno State Help Cen...
My Fresno State
NETFLIX
Watch Instantly
Just for Kids
Personalize
Search
New Releases
Hansel & Gretel
THE LAST STAND
TOM CRUISE
JACK REACHER
A NETFLIX ORIGINAL
ORANGE is the new BLACK
FROM THE CREATOR OF WEEDS
Breaking Bad
Recently added in TV Shows
MARON
PBS
PBS
A NETFLIX ORIGINAL
11:29 AM

Entities in Netflix Domain

- Movies
- Users
- Rating
- Reviews
- Categories

Royal Caribbean



Entities in Royal Caribbean

- Ships
- Rooms
- Destinations
- Ports
- Excursions
- Passengers
- Crew

E-R Model

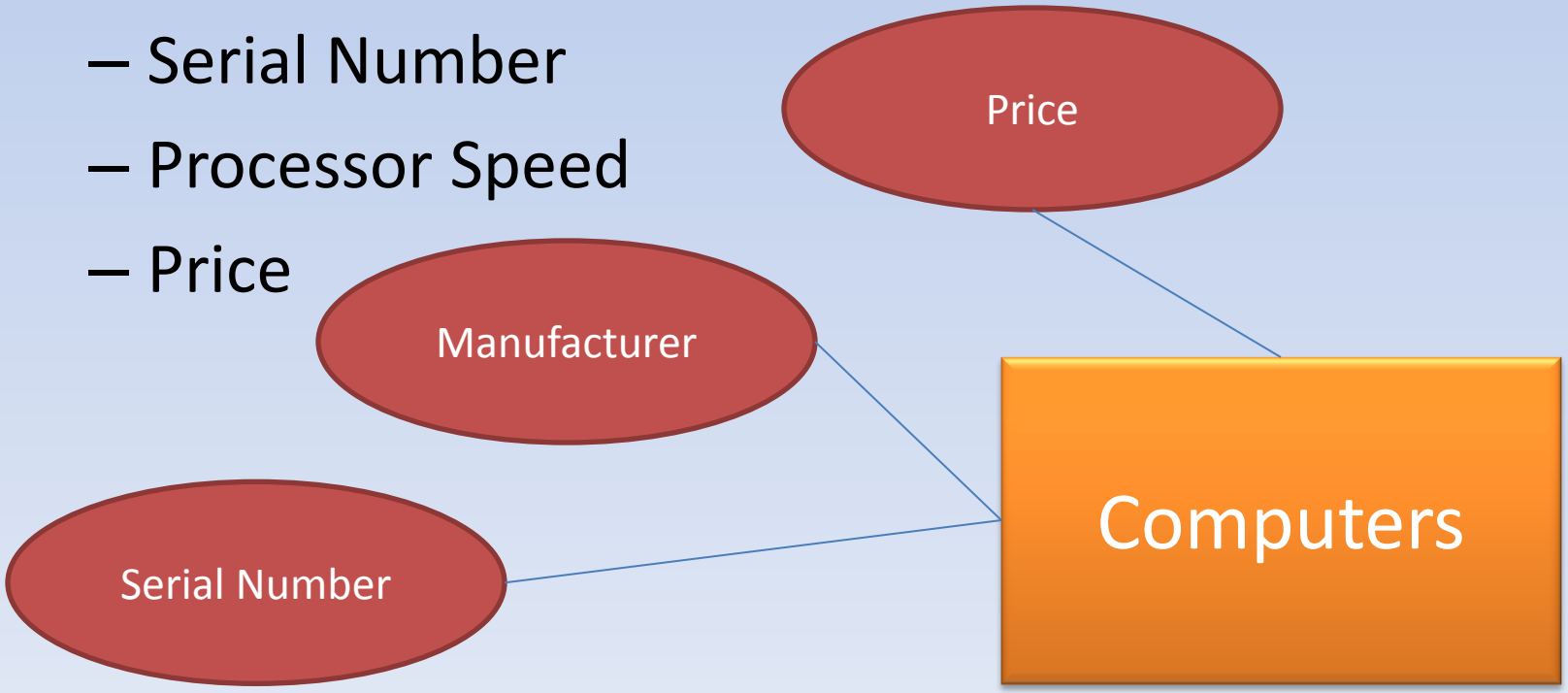
- Entity: Object represented in the database.
- Attributes: Properties of Entities.
  - Computers: Serial Number, Processor Speed, Price, etc…
- Entity Set: A collection of the same type of entities.
  - Computer is an Entity.
  - A Computer Entity Set would contain different computers.

E-R Model - Representation

- Representation of Entity Sets:
  - Entity Sets are represented by Squares
  - Attributes are ovals connected to by lines to their Entity Set.

ER Diagram – Computer Domain

- Entity Set: Computers
- Entity: Computer
- Attributes: A Computer has…
  - Serial Number
  - Processor Speed
  - Price

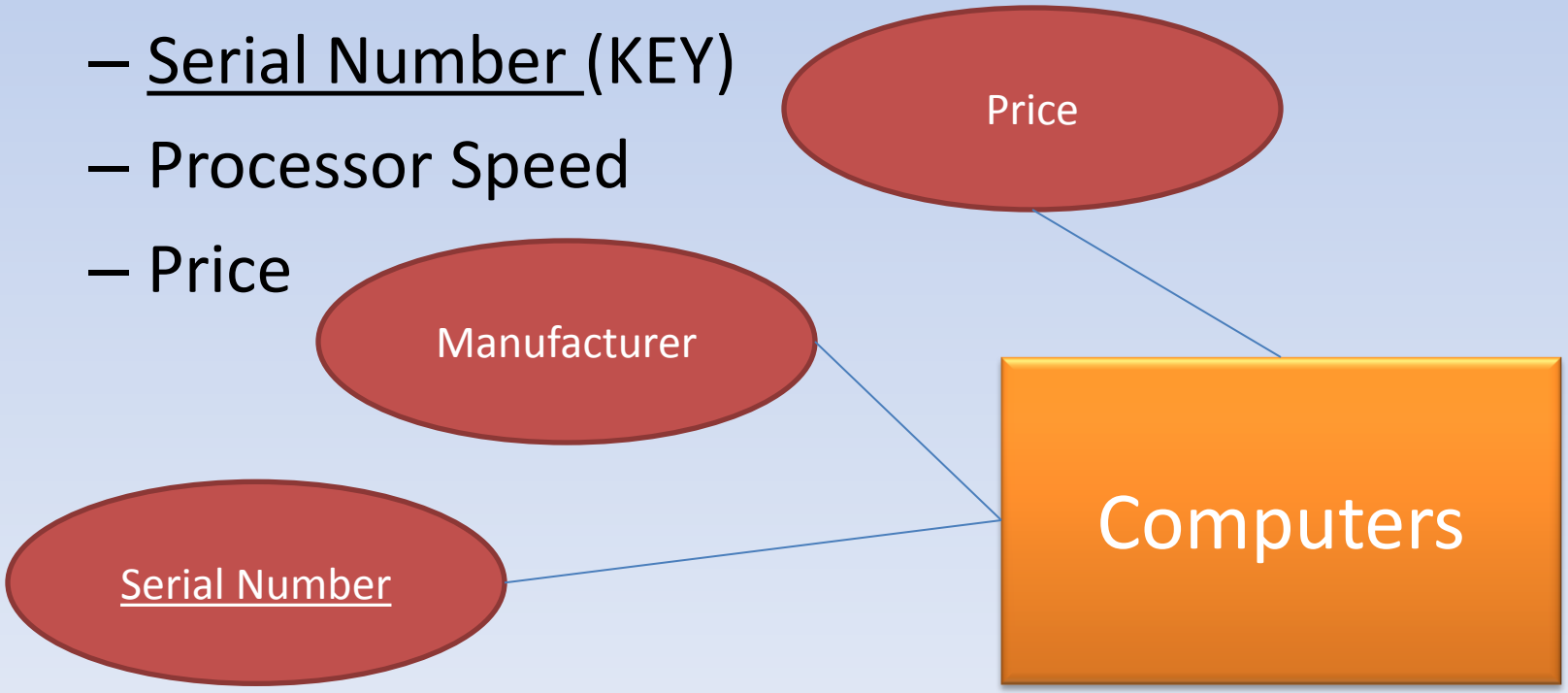


Keys & Entity Sets

- Each Entity Set must have a Key.
- Keys uniquely identify each Entity in an Entity Set.
- Keys are designated as underlined attributes.

ER Diagram – Computer Domain

- Entity Set: Computers
- Entity: Computer
- Attributes: A Computer has…
  - Serial Number (KEY)
  - Processor Speed
  - Price

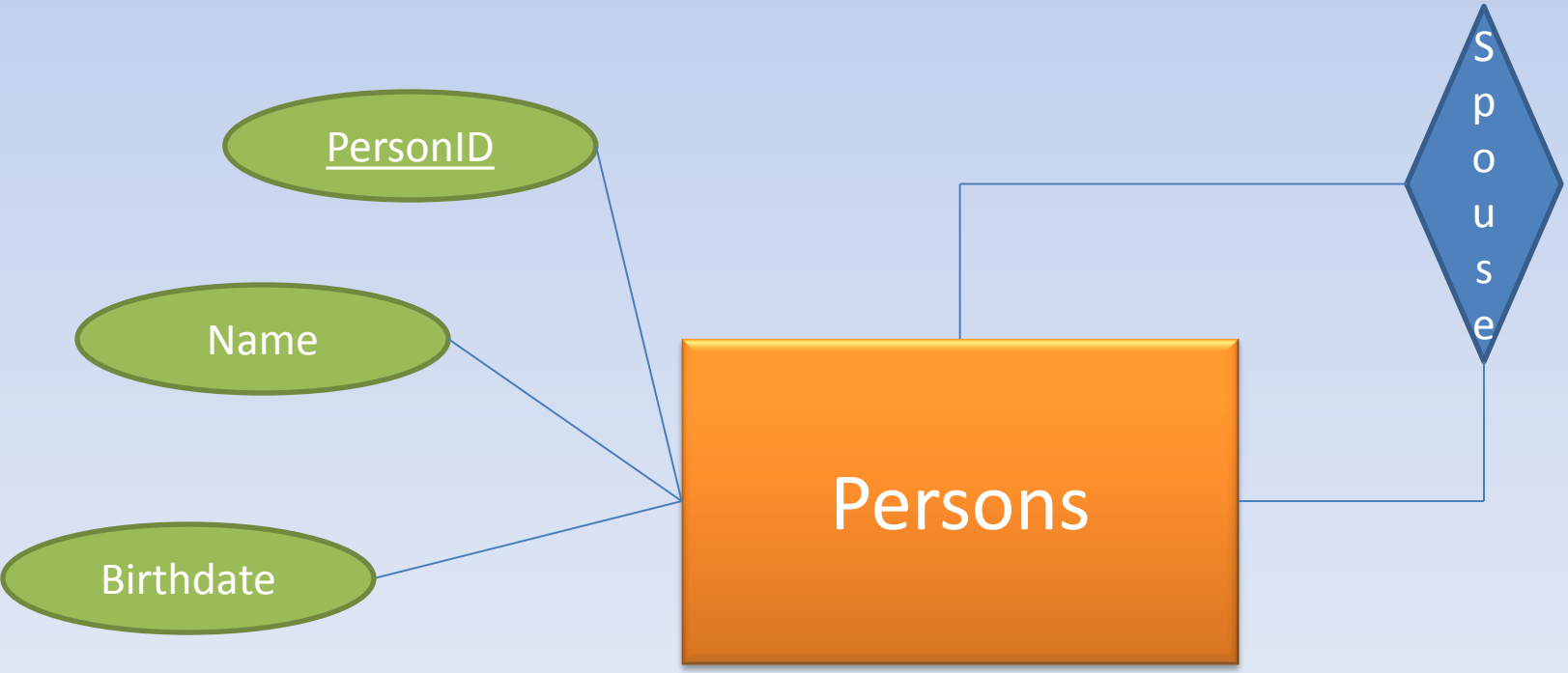


Relationships

- Relationships connect Entity Sets
- Diamonds represent Relationships

Relationship Example

- Persons: Entity Set for Person Entity
- Person: PersonID, Name, Birthdate
- Spouse: Relationship between Persons.

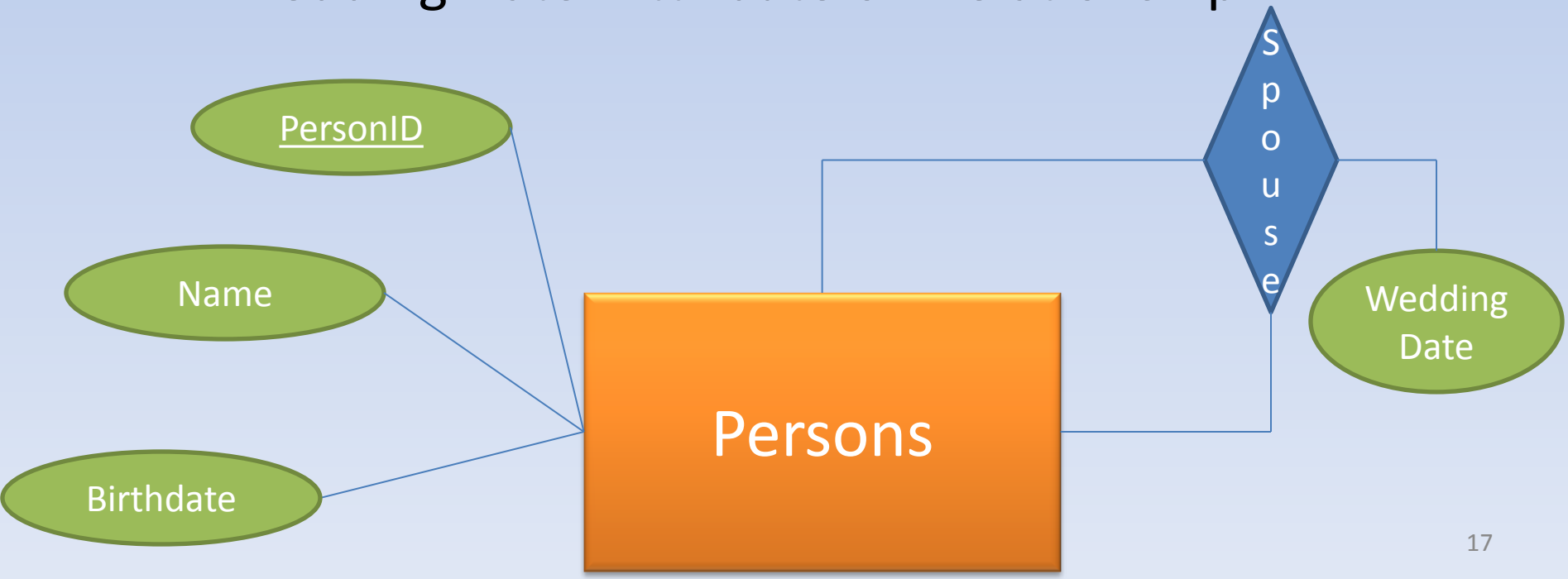


Relationship Attributes

- Relationships can have associated attributes
- Attributes are attached to relationship diamonds.
- Attributes represented as ovals

Relationship Example

- Persons: Entity Set for Person Entity
- Person: PersonID, Name, Birthdate
- Spouse: Relationship between Persons.
  - Wedding Date: Attribute on Relationship



Relationship Value

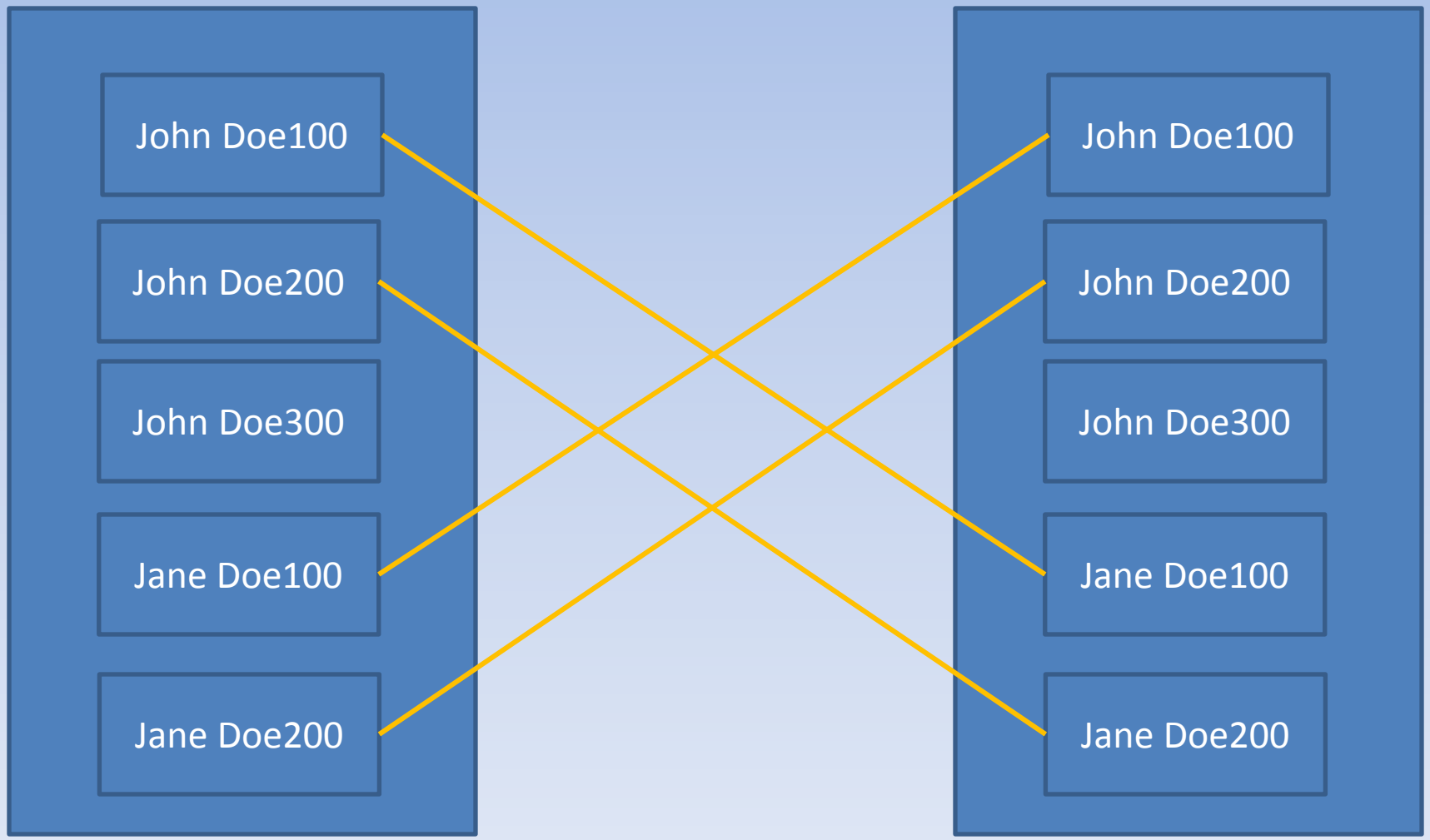
- Entity Set Value: List of entities in set.
- Relationship Value: Created from combining components from each related Entity Set.

PersonID	Wedding Date	PersonID
JohnDoe101	10/10/2010	JaneDoe101
JohnDoe200	2/14/2000	JaneDoe200

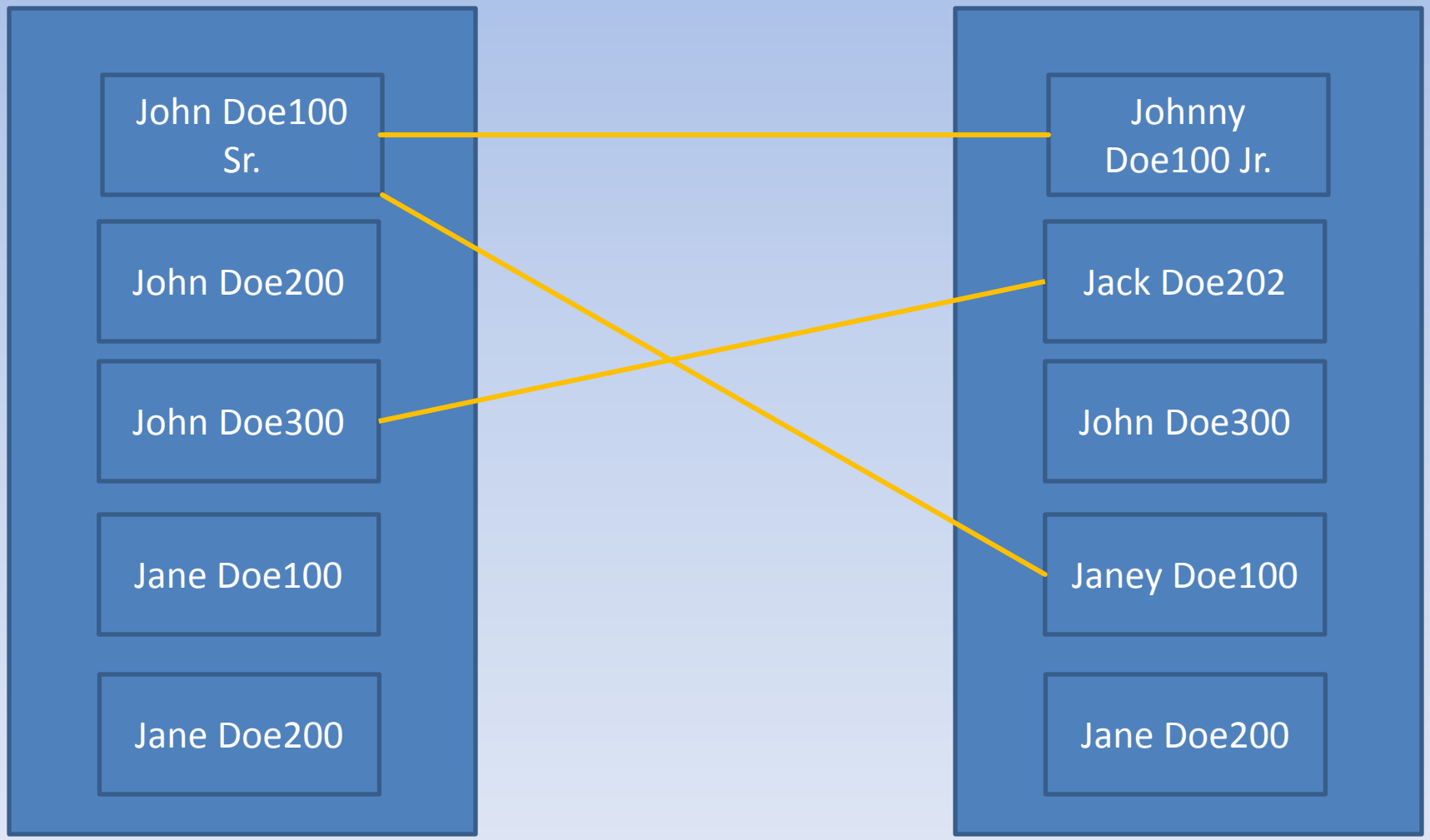
Relationship Types

- One-to-One
  - Spouse
- Many-to-One
  - Father, Mother
- Many-to-Many
  - Movie-Rating
  - Cousin

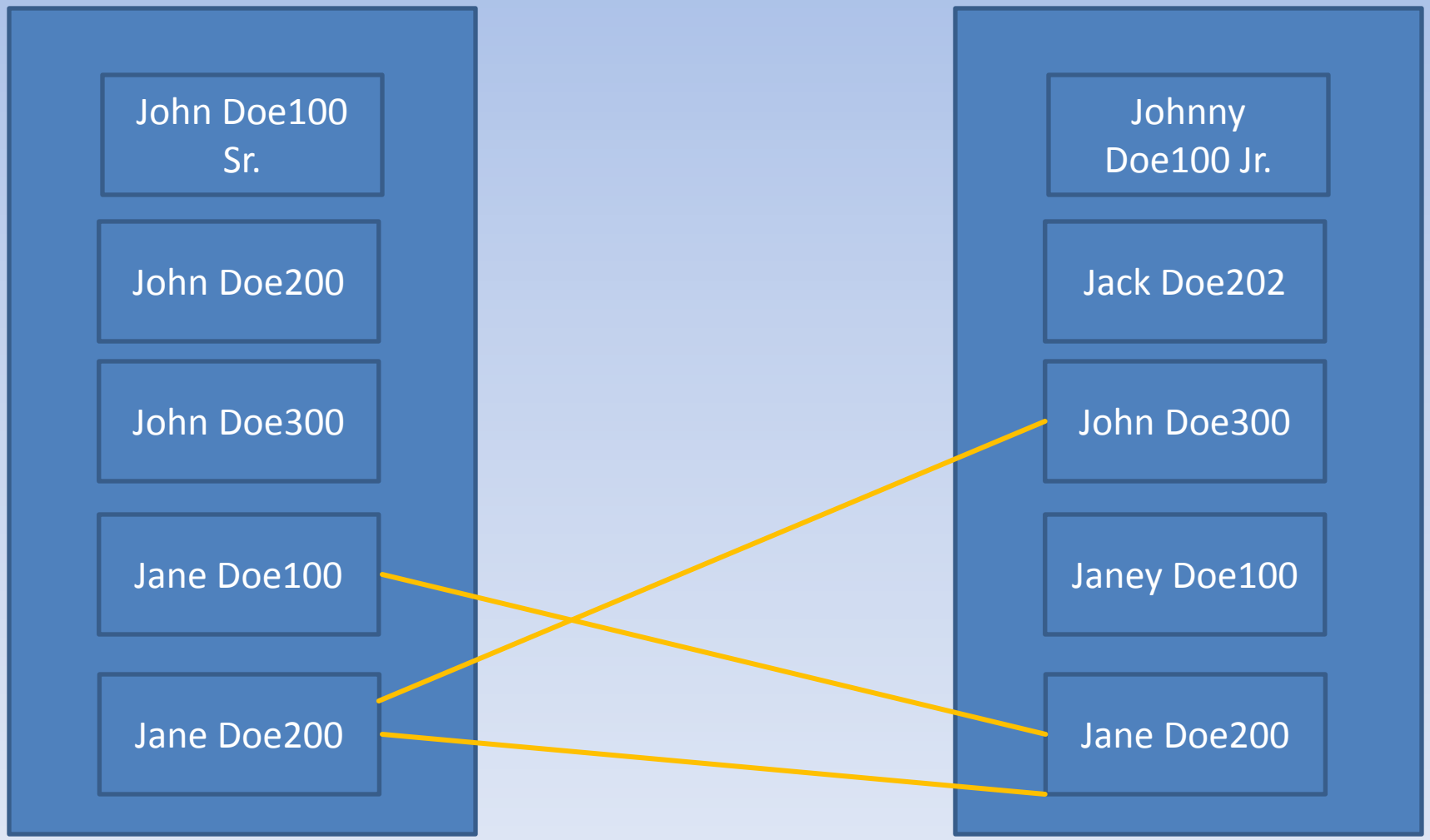
One-to-One Spouse



Many-to-One Father



Many-to-Many
Cousin



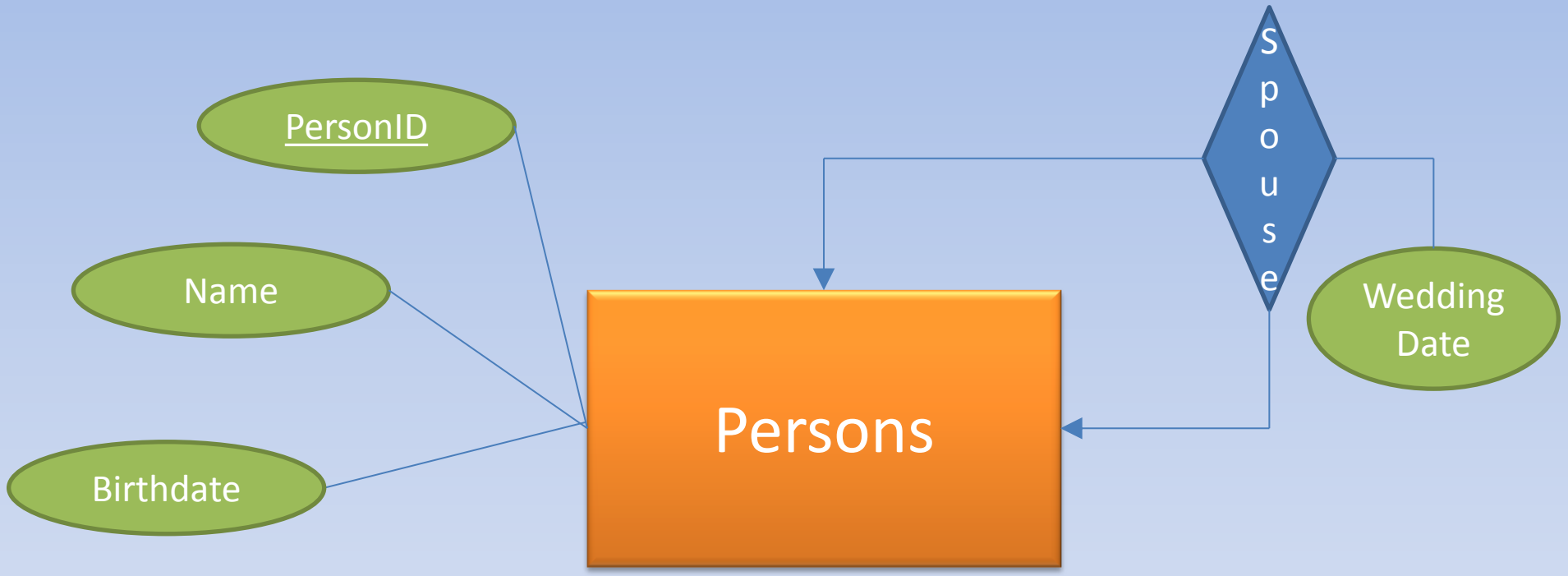
Representing Multiplicity

- Many-to-One: Arrow Head hits the ‘One’ Side.
- One-to-One: Arrow Head on both sides.
- Many-to-Many: No Arrow Head
- NOTE: Rounded (Unfilled) Arrow Head means exactly one (not One or Zero).
  - A child *must* have a Father.
  - A person may be unmarried, but if married must have only One Spouse.

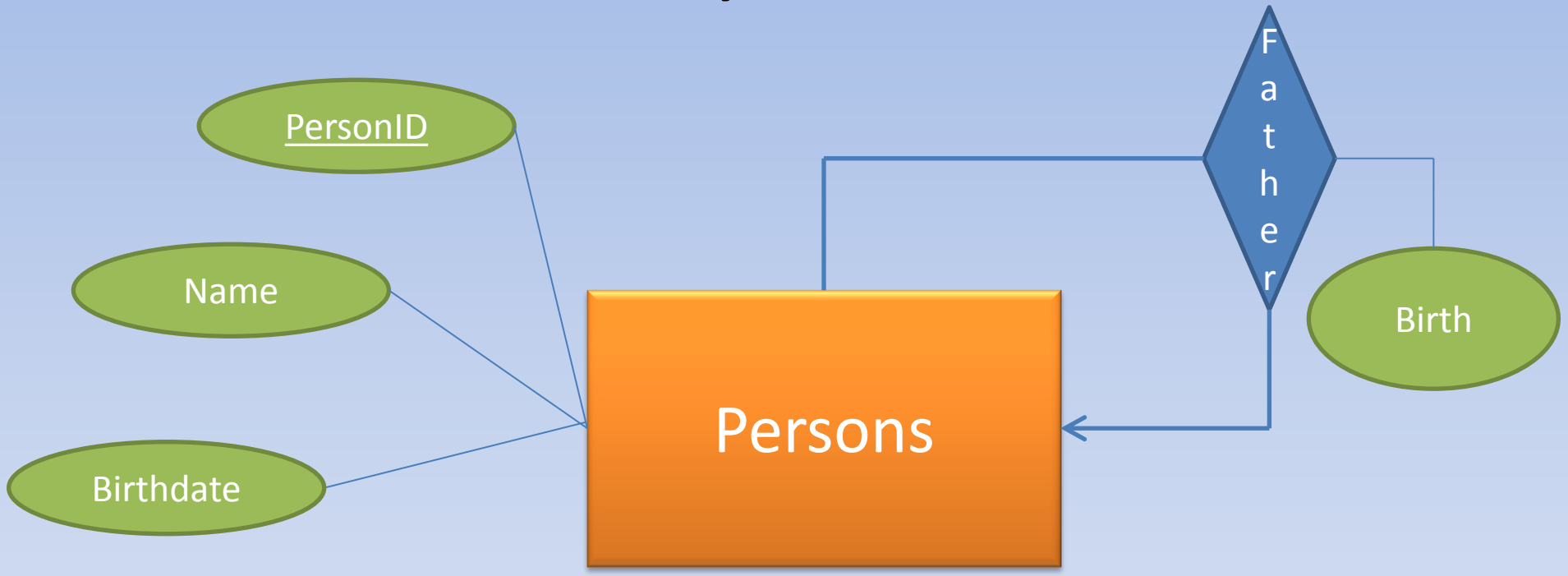
Representing Multiplicity

- Examples
- Spouse
  - One-to-One
- Father
  - Many-to-One

Representing Multiplicity one-to-one



Representing Multiplicity many-to-one



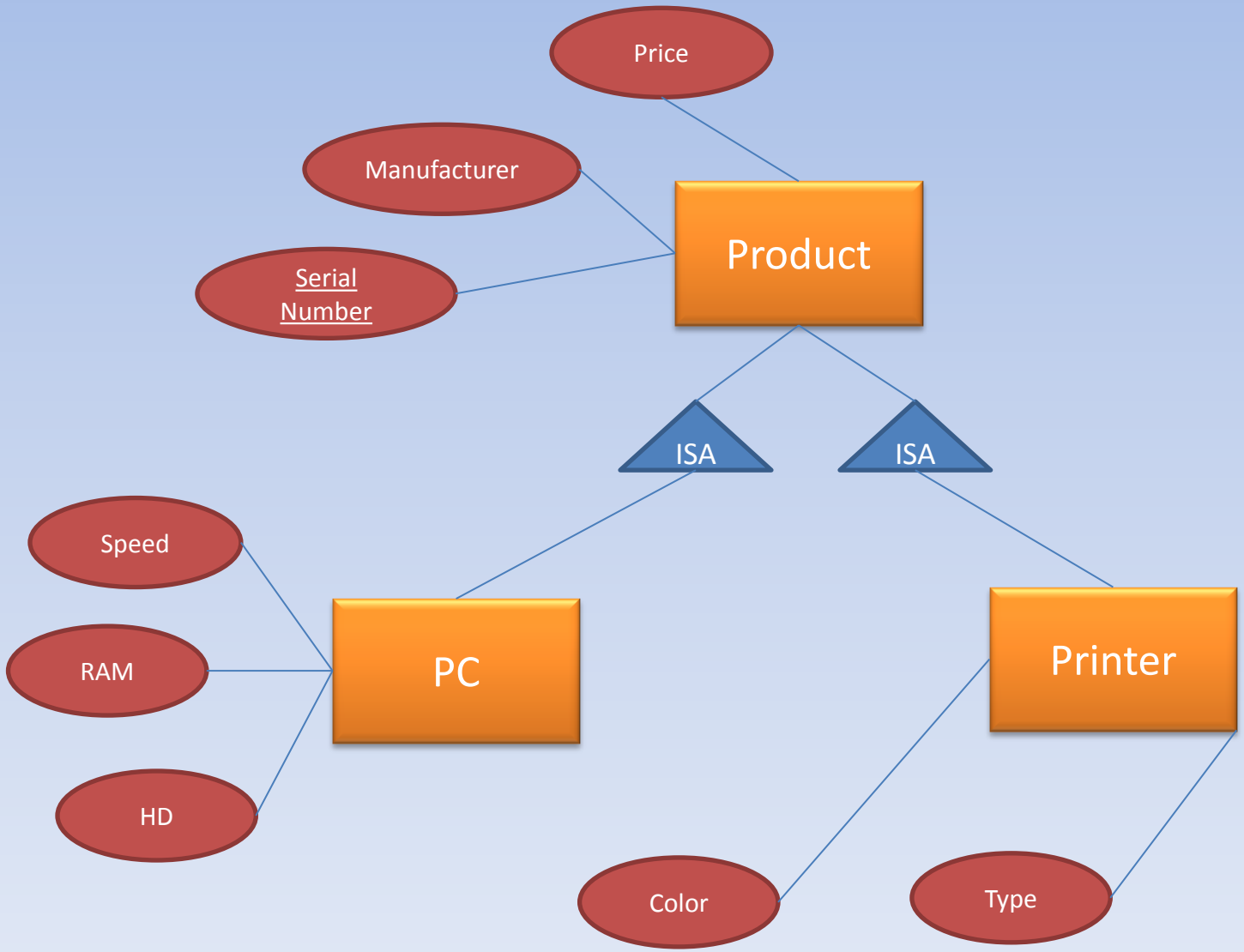
Sub-Classes

- Entity Special Case
  - Fewer example
  - More attributes
- Products
  - All products have a Product Code
  - Some products are PC's and have HD, Memory, …
  - Some products are Printers and have Color, Laser, …

Sub-Classes in ER Diagrams

- Sub-Classes implemented using ISA Hierarchy.
- Assume a tree-structure, with no multiple inheritance (Children have only one parent).

Sub-Classes - Products



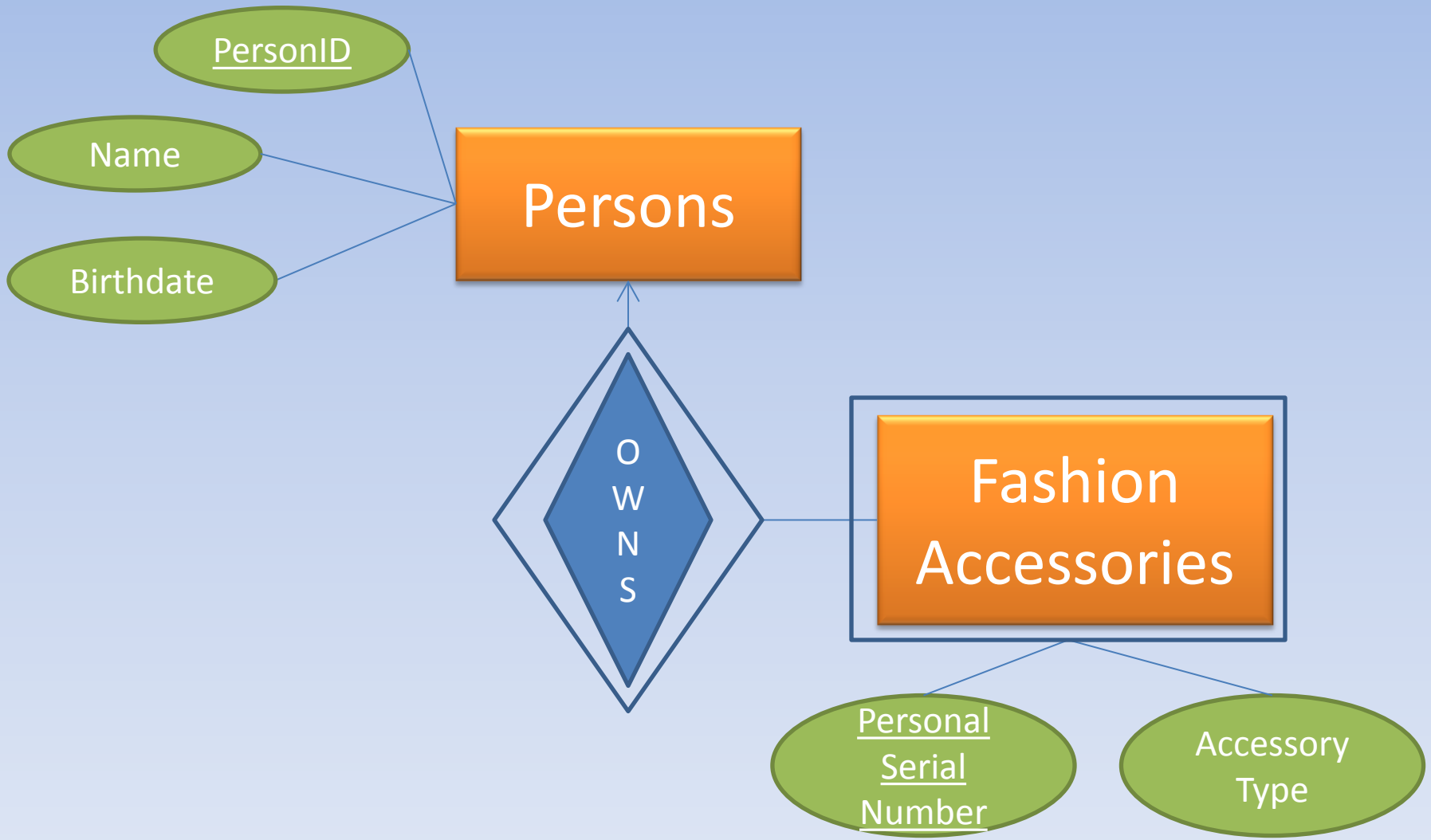
Weak Entity Sets : Who Are You?

- Entities within a set may not be uniquely identified.
  - Jack's Surfboard
  - Surfboard not uniquely identified.
  - Surfboard identified through supporting relationship to Jack (uniquely identified).

Weak Entity Sets : ER Diagrams

- Double Diamonds for Supporting Relationship
- Double Rectangle for Weak Entity Set
- Supporting Relationship must have rounded arrow.

Weak Entity Sets: Example



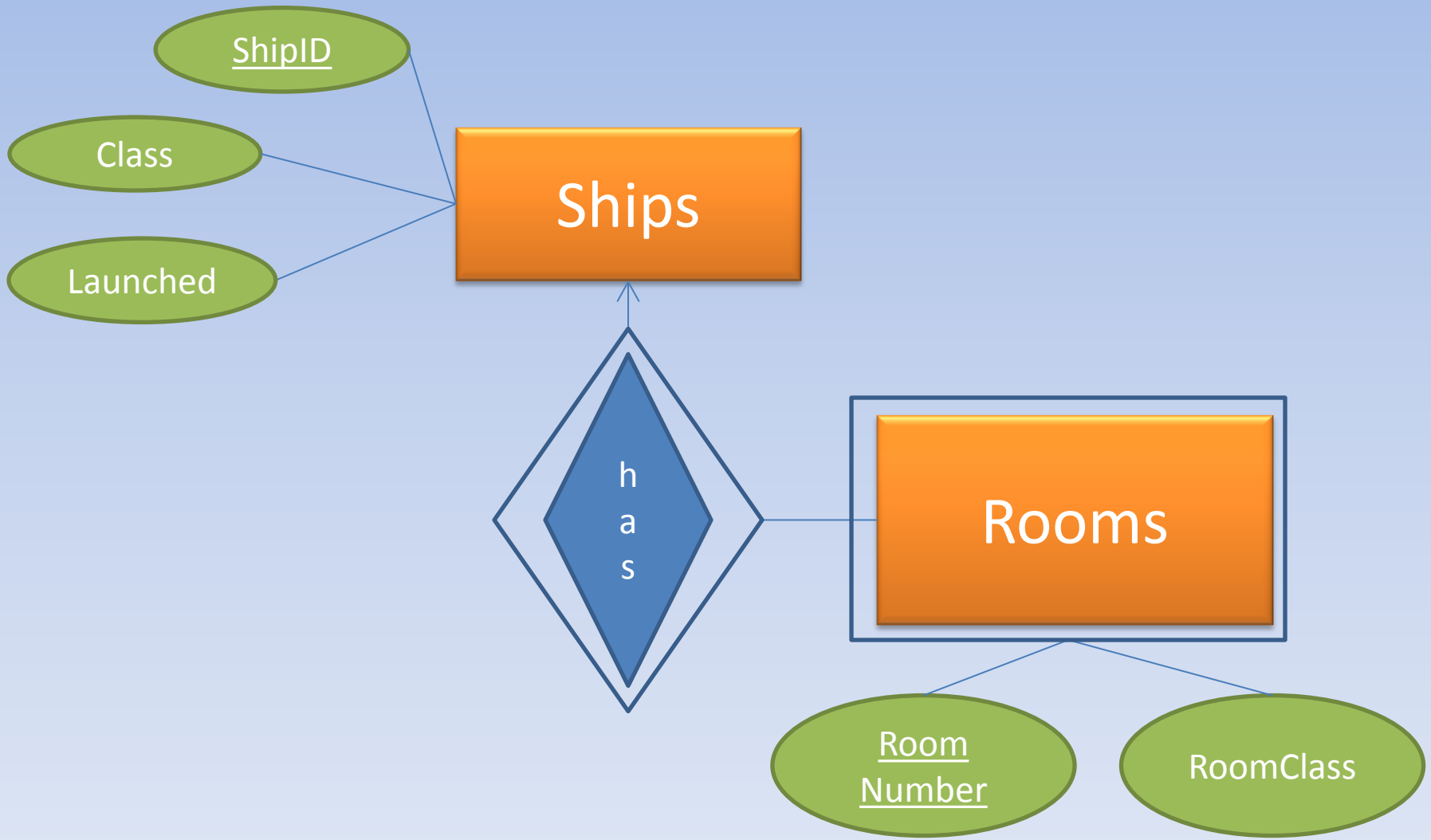
Weak Entities in Royal Caribbean

- Ships
- Rooms
- Destinations
- Ports
- Excursions
- Passengers
- Crew

Weak Entities in Royal Caribbean

- Ships
- Rooms:
  - Rooms have numbers
  - Room numbers not unique
- Destinations
- Ports
- Excursions
- Passengers
- Crew

Weak Entity Sets: RC Example



In-Class Five

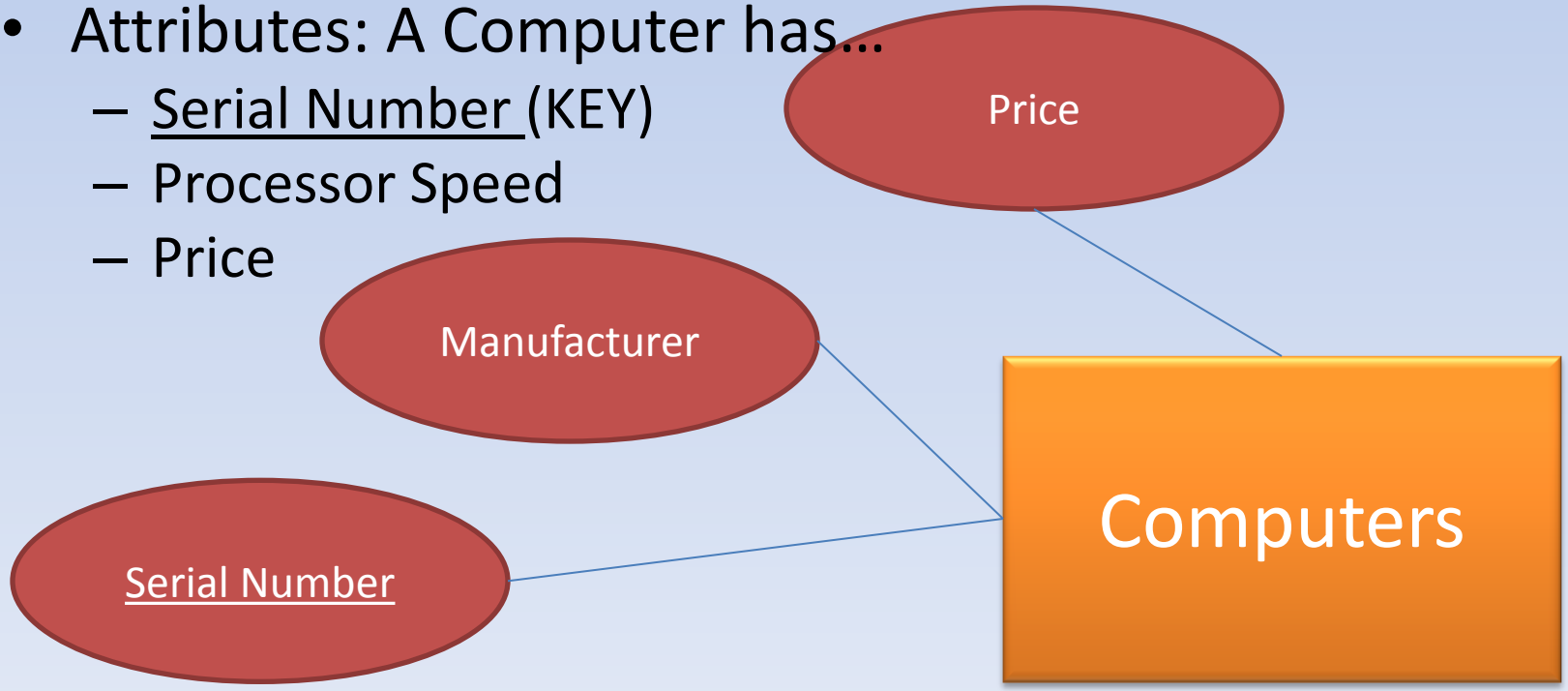
- From your project domain, choose two entities and one relationship between them, and develop an E/R Diagram representing them.
- Give a short description of the entities and relationship, including the multiplicity for the relationship.

Mapping ER Diagrams to Relations Basics

- Entity Sets become Relations
- Relationships Become Relations with/
  - Attributes taken from Key Value of connected Entity Sets
  - Attributes from Relationship itself.

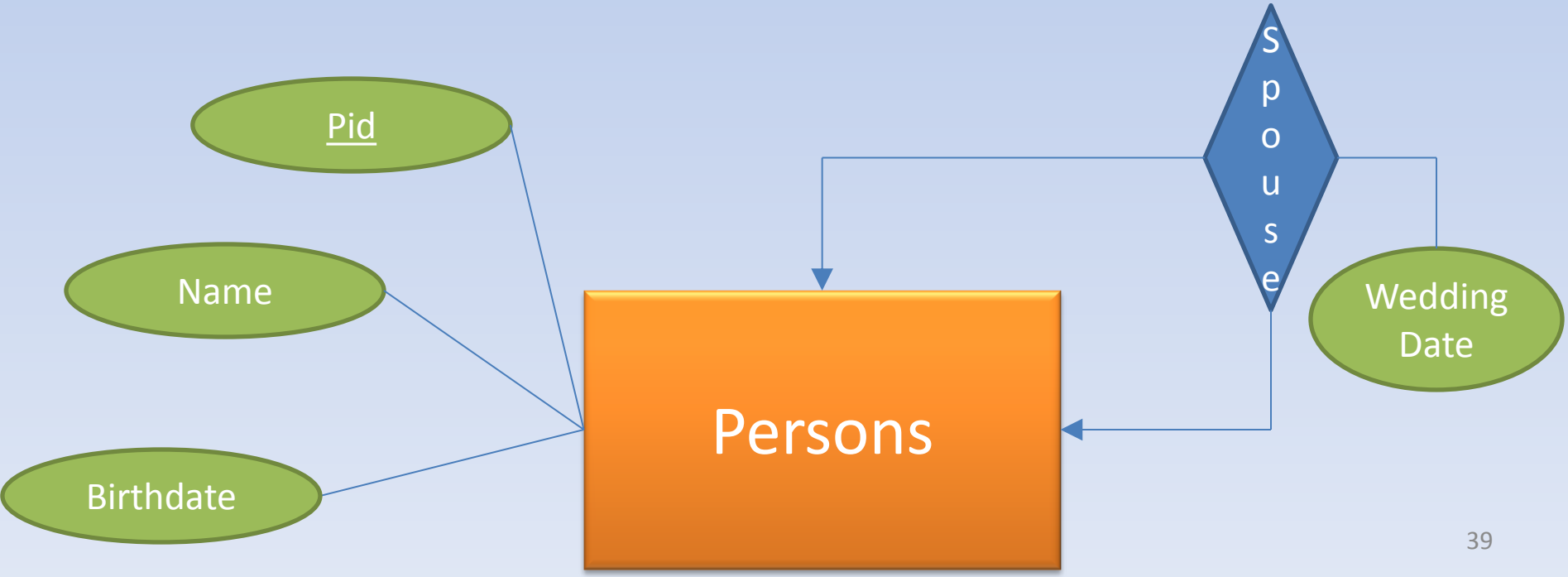
ER Diagram – Computer Domain to Relation

- Relation:
  Computers(
  SerialNumber int key, ProcessorSpeed int, Price int)
- Entity Set: Computers
- Entity: Computer
- Attributes: A Computer has…
  - Serial Number (KEY)
  - Processor Speed
  - Price



Relationship Example
To Relation

- Spouse(Pid1 int, Pid2 int, WeddingDate date)
- Persons: Entity Set for Person Entity
- Person: PersonID, Name, Birthdate
- Spouse: Relationship between Persons.

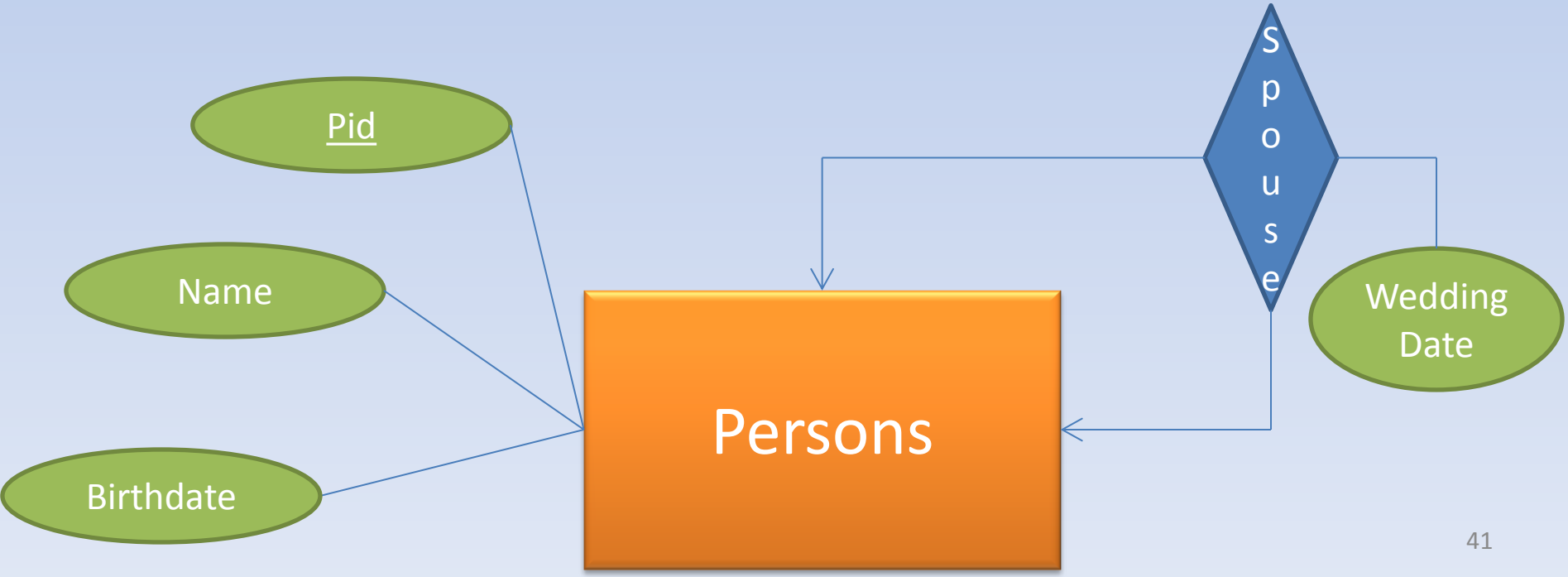


Combining Relations

- Relations can be combined:
  - Relations for Entity Set ‘Entity’
  - Relations for Relationship ‘R’
  - IF:
    - R is a Many-To-One with Entity the Many
    - R is a One-To-One
    - RISKS w/ Many-to-Many

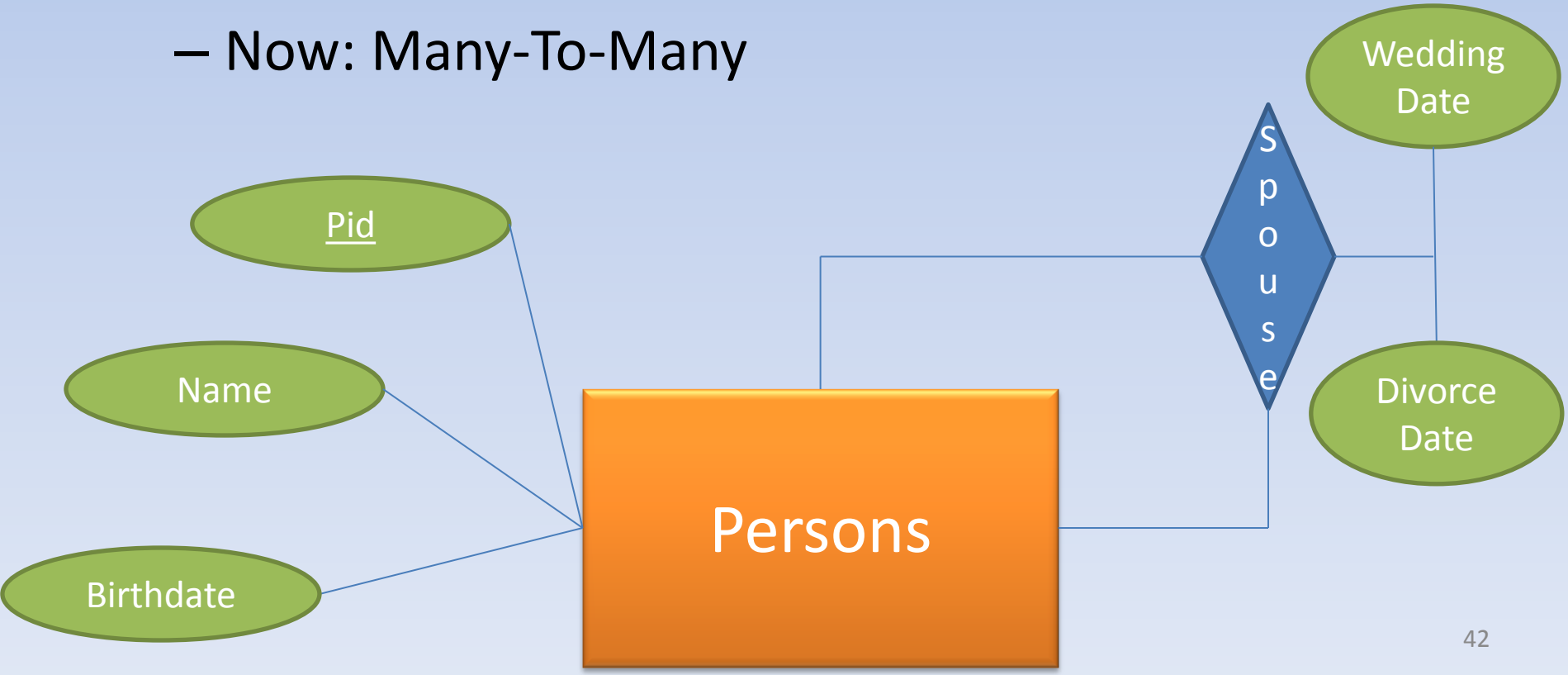
Relationship Example To Relation

- Person(Pid int, Name varchar(20), Birthdate date, Spouse int, WeddingDate date)
- Persons: Entity Set for Person Entity
- Person: PersonID, Name, Birthdate
- Spouse: Relationship between Persons.



Relationship Example To Relation

- Persons: Entity Set for Person Entity
- Person: PersonID, Name, Birthdate
- Spouse: Relationship between Persons.
  - Now: Many-To-Many

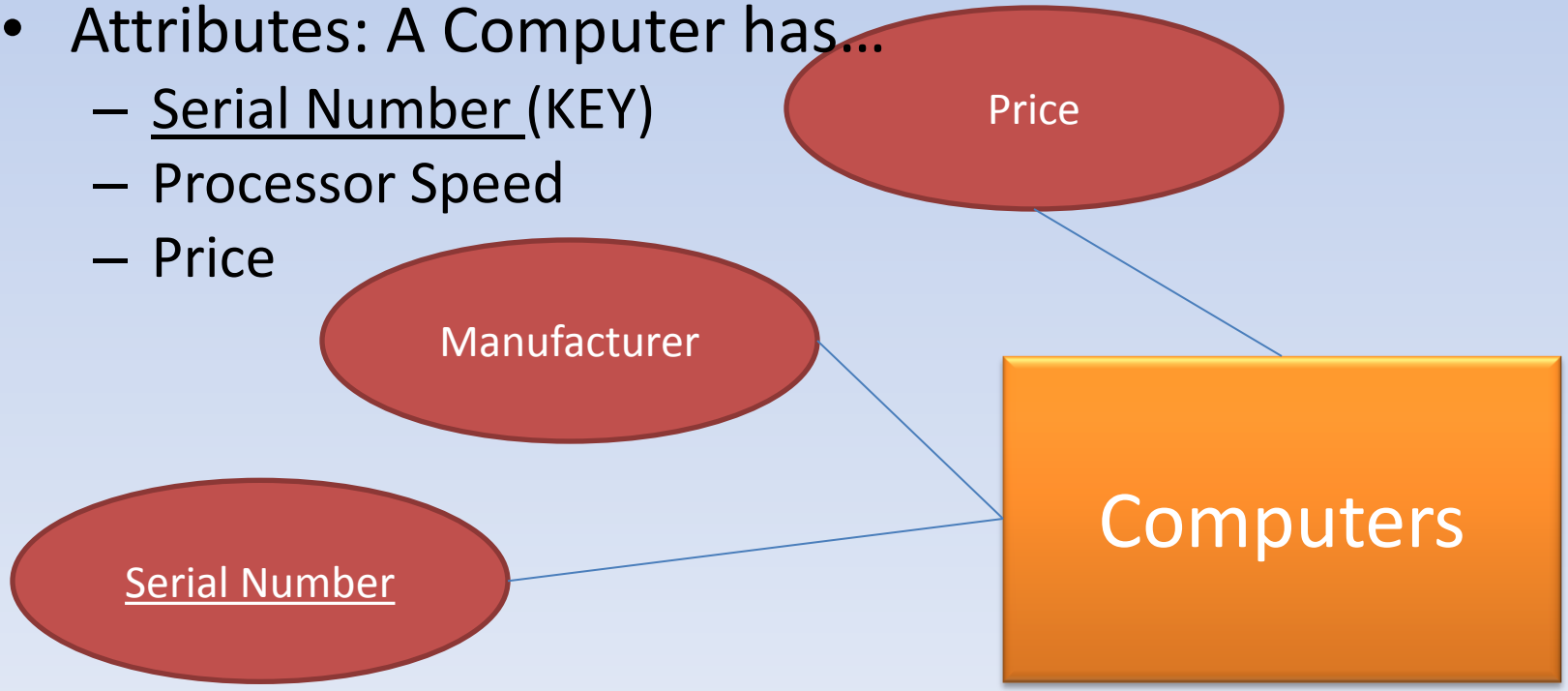


Mapping ER Diagrams to Relations Basics

- Entity Sets become Relations
- Relationships Become Relations with/
  - Attributes taken from Key Value of connected Entity Sets
  - Attributes from Relationship itself.

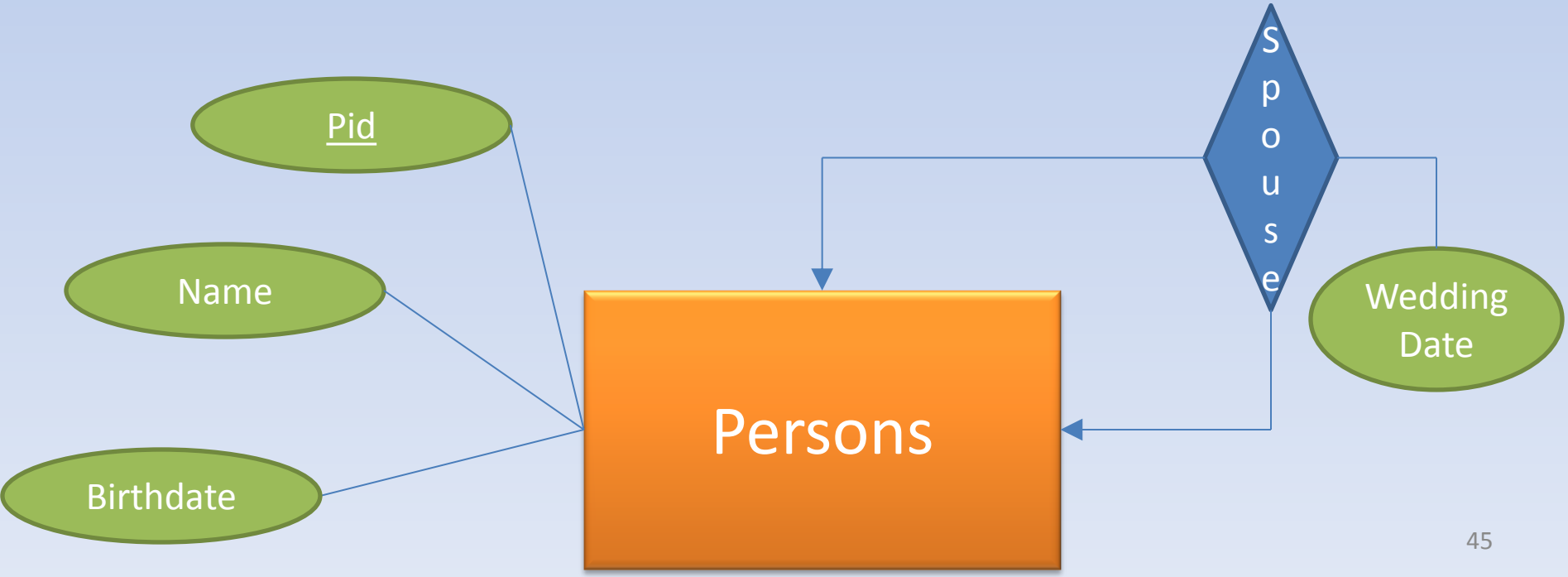
ER Diagram – Computer Domain to Relation

- Relation:
  Computers(
  SerialNumber int key, ProcessorSpeed int, Price int)
- Entity Set: Computers
- Entity: Computer
- Attributes: A Computer has…
  - Serial Number (KEY)
  - Processor Speed
  - Price



Relationship Example
To Relation

- Spouse(Pid1 int, Pid2 int, WeddingDate date)
- Persons: Entity Set for Person Entity
- Person: PersonID, Name, Birthdate
- Spouse: Relationship between Persons.

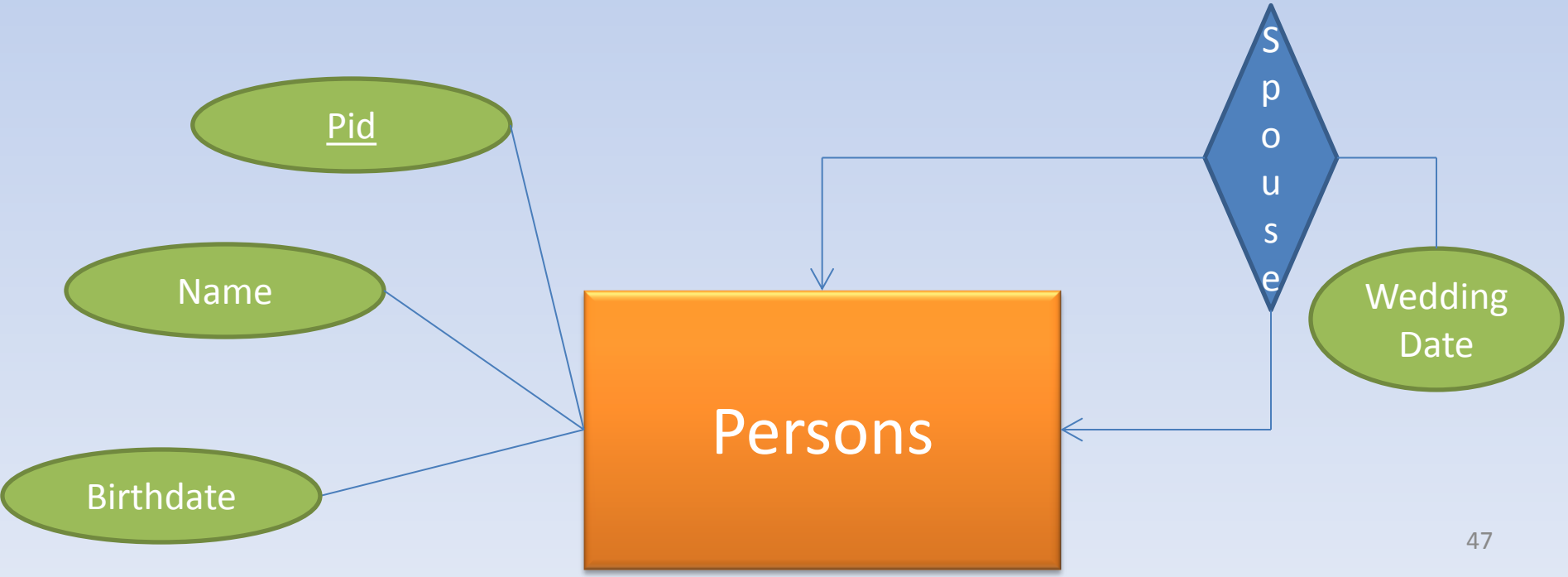


Combining Relations

- Relations can be combined:
  - Relations for Entity Set ‘Entity’
  - Relations for Relationship ‘R’
  - IF:
    - R is a Many-To-One with Entity the Many
    - R is a One-To-One
    - RISKS w/ Many-to-Many

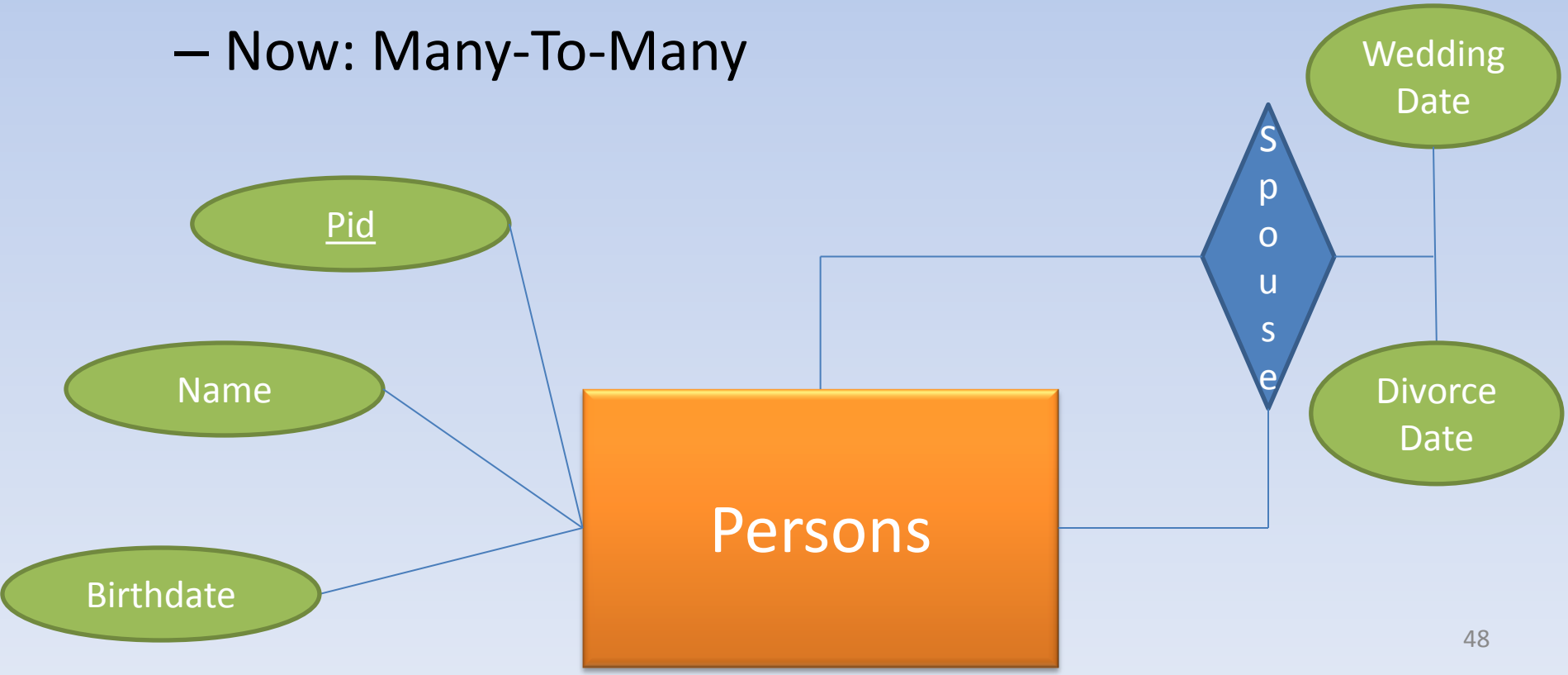
Relationship Example To Relation

- Person(Pid int, Name varchar(20), Birthdate date, Spouse int, WeddingDate date)
- Persons: Entity Set for Person Entity
- Person: PersonID, Name, Birthdate
- Spouse: Relationship between Persons.



Relationship Example To Relation

- Persons: Entity Set for Person Entity
- Person: PersonID, Name, Birthdate
- Spouse: Relationship between Persons.
  - Now: Many-To-Many



Weak Entity Sets to Relations

- Relation for Weak Entity-Sets must use complete key:
  - Key attributes for weak entity set
  - Plus Key attributes from Entity Sets connected by supporting relationship
  - Supporting Relationship not needed unless it adds attributes.

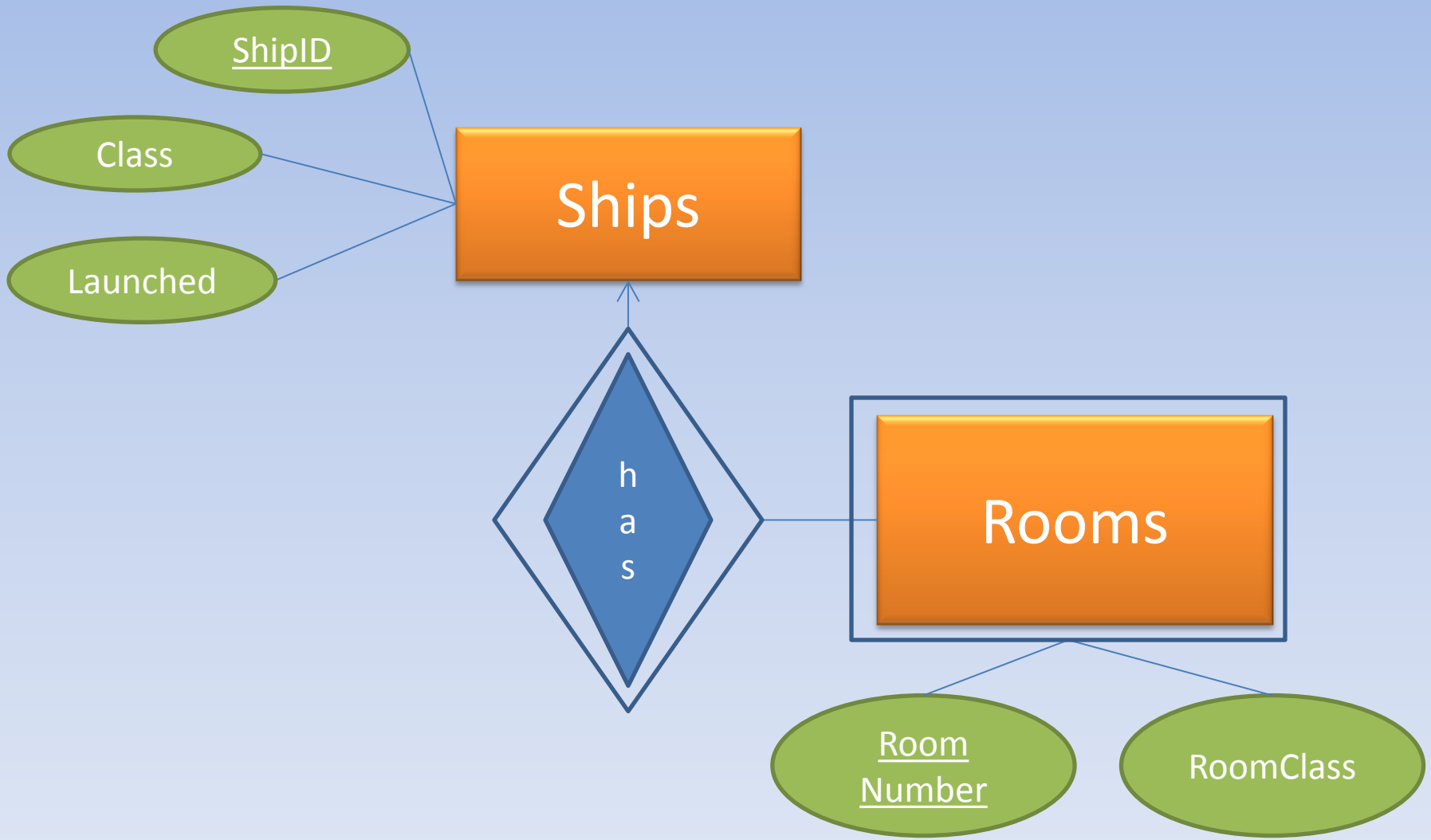
Weak Entities in Royal Caribbean

- Ships
- Rooms
- Destinations
- Ports
- Excursions
- Passengers
- Crew

Weak Entities in Royal Caribbean

- Ships
- Rooms:
  - Rooms have numbers
  - Room numbers not unique
- Destinations
- Ports
- Excursions
- Passengers
- Crew

Weak Entity Sets: RC Example

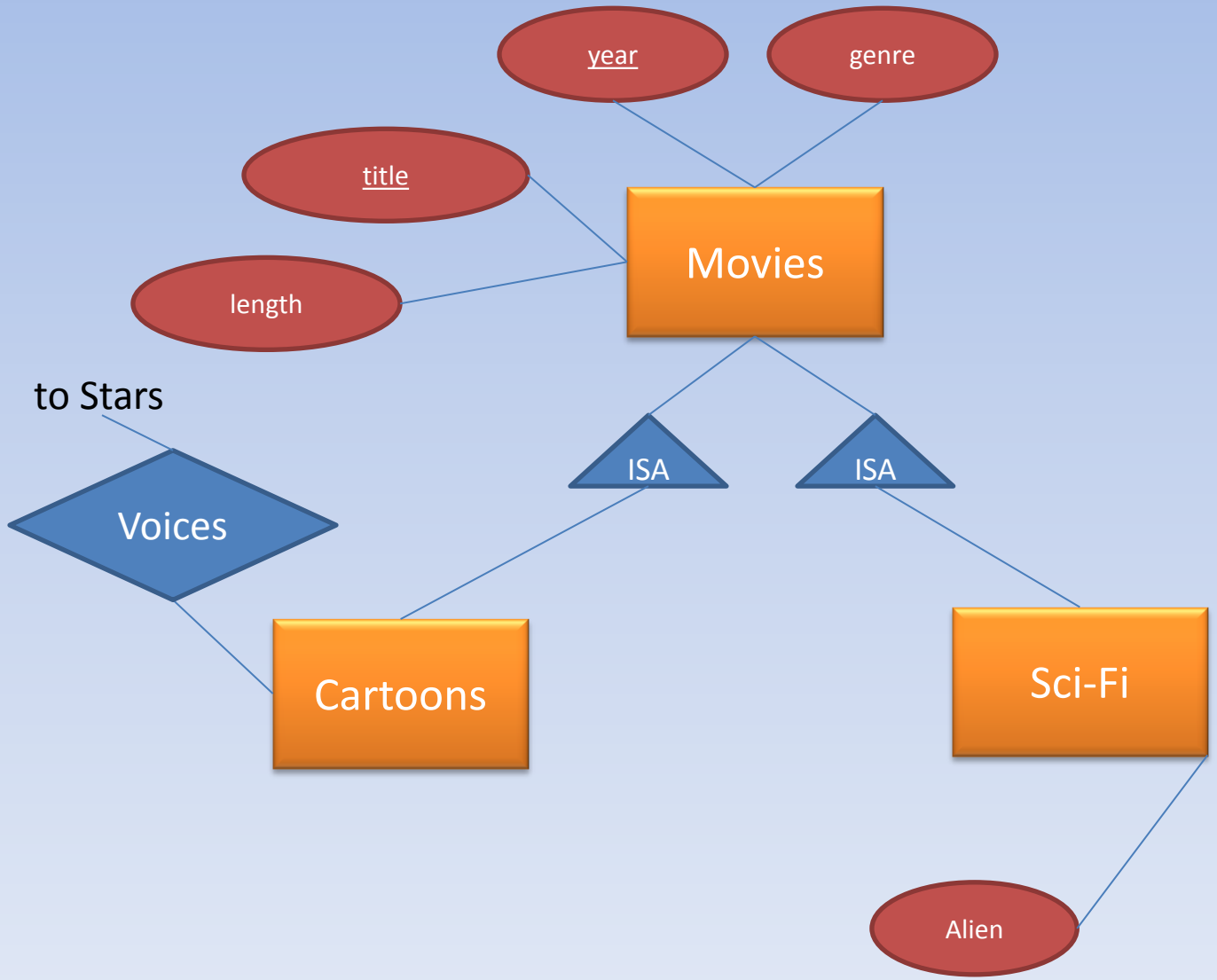


RC Example: Relations

- Ships(shipID, class, launched)
- Rooms(roomNo, roomClass, shipID)

- Don’t need the supporting relationship HAS since the shipID is already contained in Rooms relation.

Movie Hierarchy



Hierarchy Conversion To Relations

- E/R Style
- Object Oriented
- Null Values

E/R Style Conversion

- Movies(title, year, length, genre)
- Sci-Fi(title, year, alien)
  - Also tuple in Movies
- Cartoons(title, year)
  - Also tuple in Movies

Object Oriented Approach

- Movies alone
- Movies and Cartoons only
- Movies and Sci-Fi only
- Movies and Sci-Fi and Cartoons

Object Oriented Approach

- Movies alone
- Movies and Cartoons only
- Movies and Sci-Fi only
- Movies and Sci-Fi and Cartoons

Movies(title, year, length, genre)

MoviesC(title, year, length, genre)

MoviesSciFi(title, year, length, genre, alien)

MoviesSciFiC(title, year, length, genre, alien)

Null Values

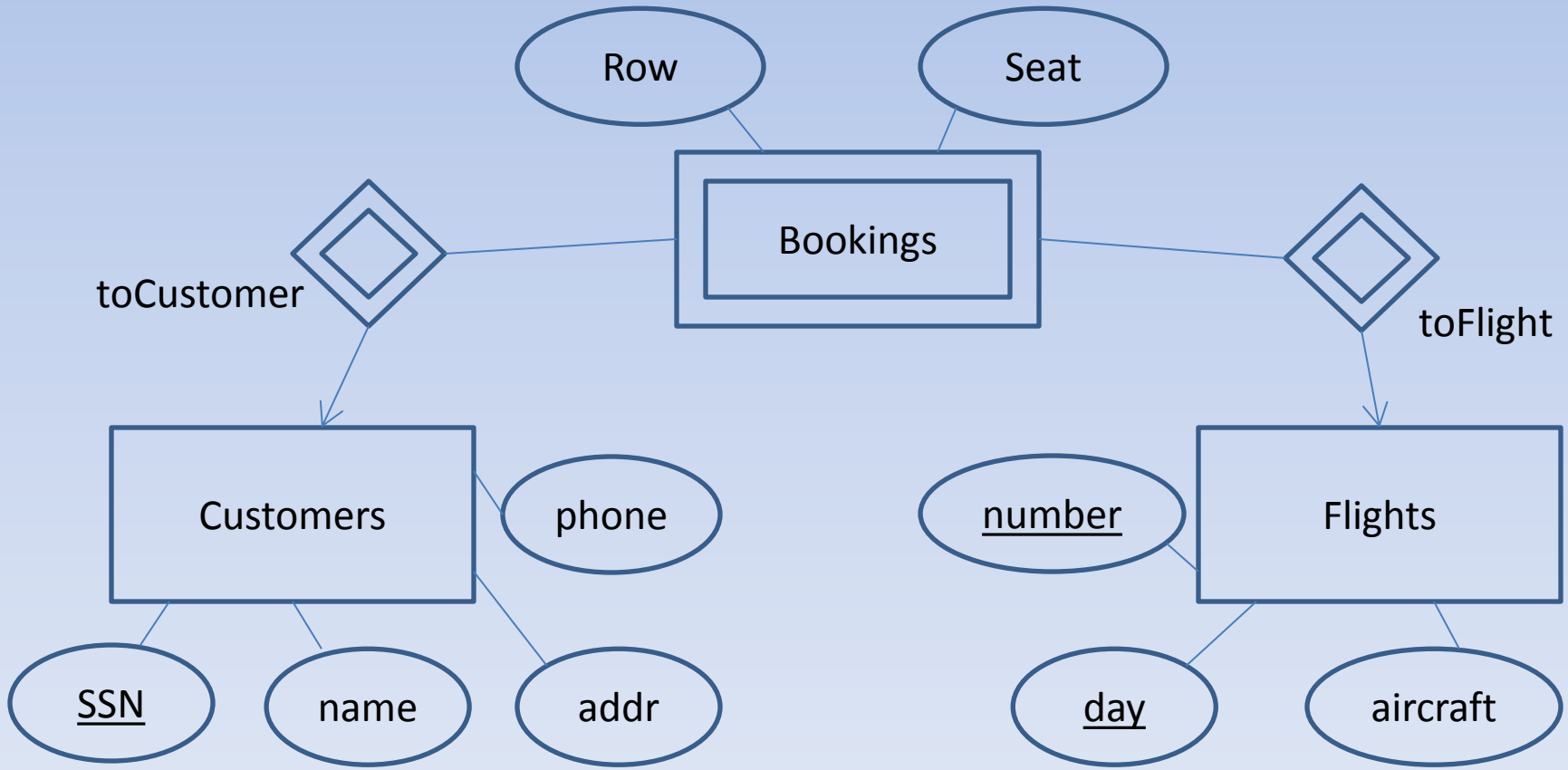
- Movies(title, year, length, genre, alien)

Comparison

- Ease for Queries
- Number of Relations
- Space Required
  - Multiple table with same attributes

Exercise 4.5.1

- Convert to Relations



Relations

- Customers(SSNo, name, addr, phone)
- Flights(number, day, aircraft)
- Bookings(custSSNo, flightNo, flightDay,
  row, seat)

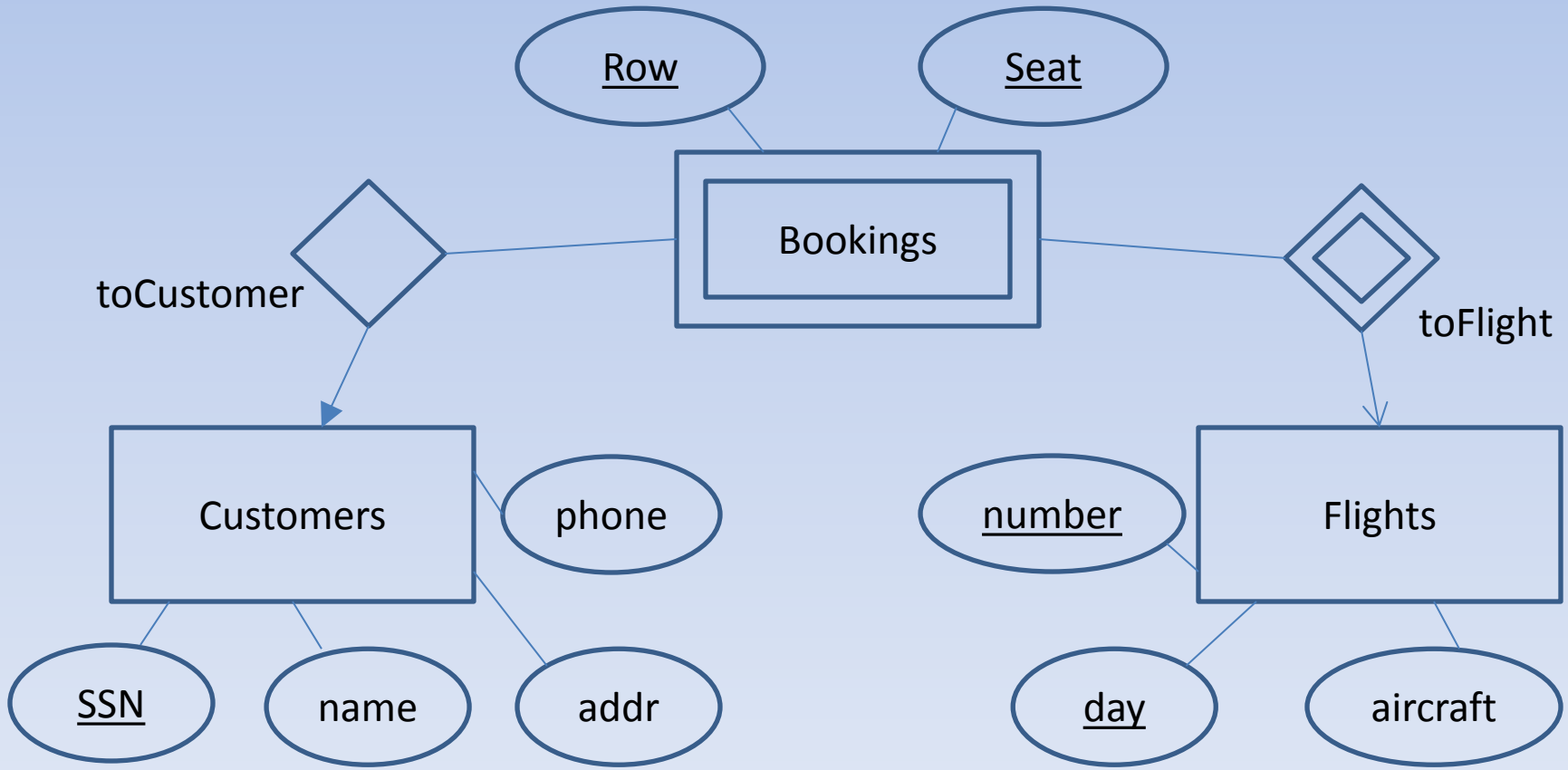
- Relations for toCust and toFlt relationships are not required since the weak entity set Bookings already contains the keys of Customers and Flights.

Exercise 4.5.2

- Bookings uniquely identified by:
  - Flight Number
  - Day of the flight
  - Row
  - Seat
- Customer is not needed to help identify the booking
- Draw an alternative E/R Diagram

Exercise 4.5.2

- Convert to Relations



4.5.2: Changed Schema

- Since toCust is no longer an identifying relationship, SSNo is no longer a part of Bookings relation.

  Bookings(flightNo, flightDay, row, seat)
  ToCust(custSSNO, flightNo, flightDay, row, seat)

- The above relations are merged into

  Bookings(flightNo, flightDay, row, seat, custSSNo)

- However custSSNo is no longer a key of Bookings relation. It becomes a foreign key instead.
  - custSSno in Bookings relation can have NULL value.

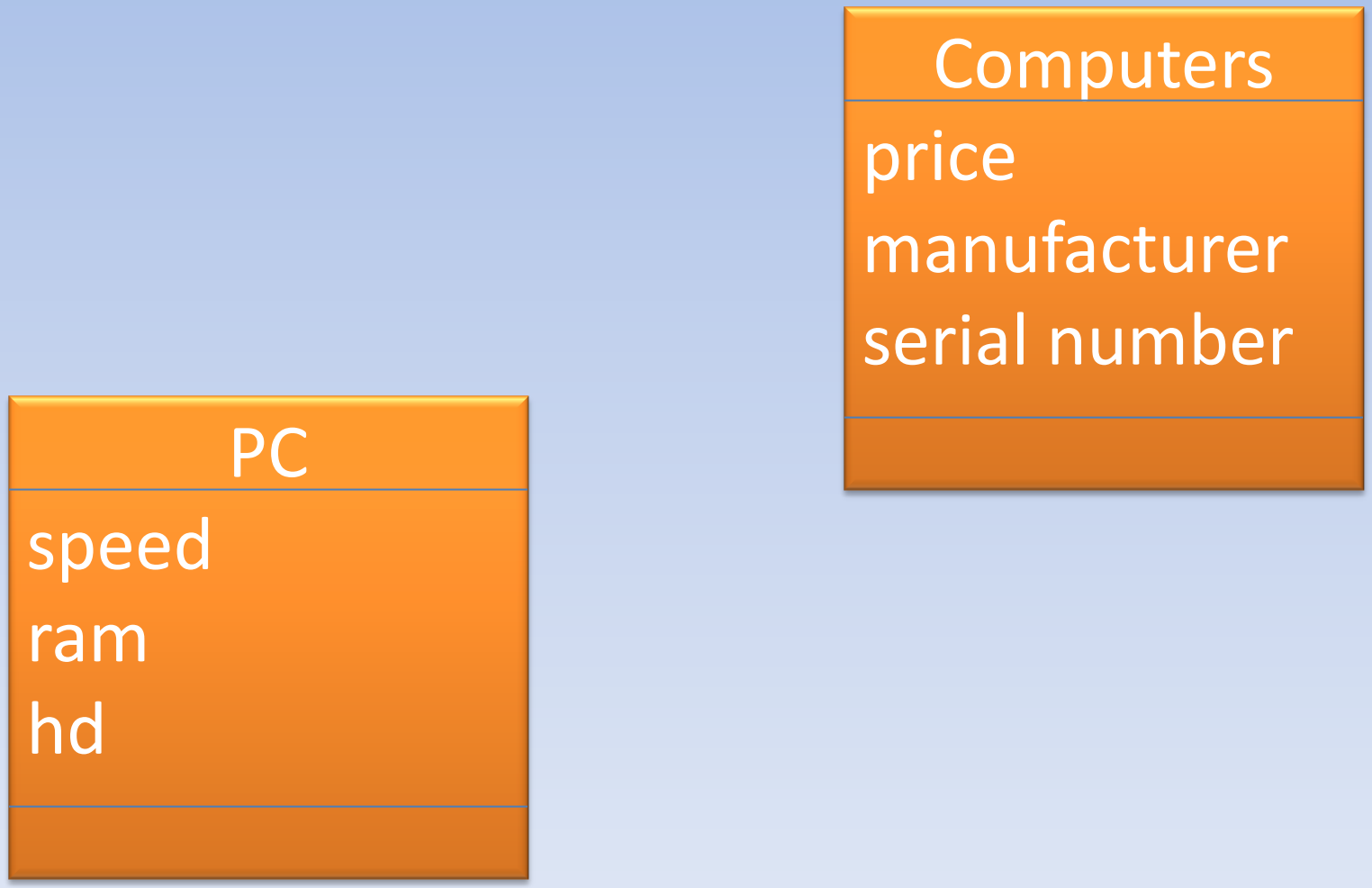
UML – Unified Modeling Language

- Originally developed for Software Design
- Extended & now popular for DB Design.

UML

- UML Classes
  - Objects represented in DB.
  - Defined by set of attributed
  - Similar to E/R Entity

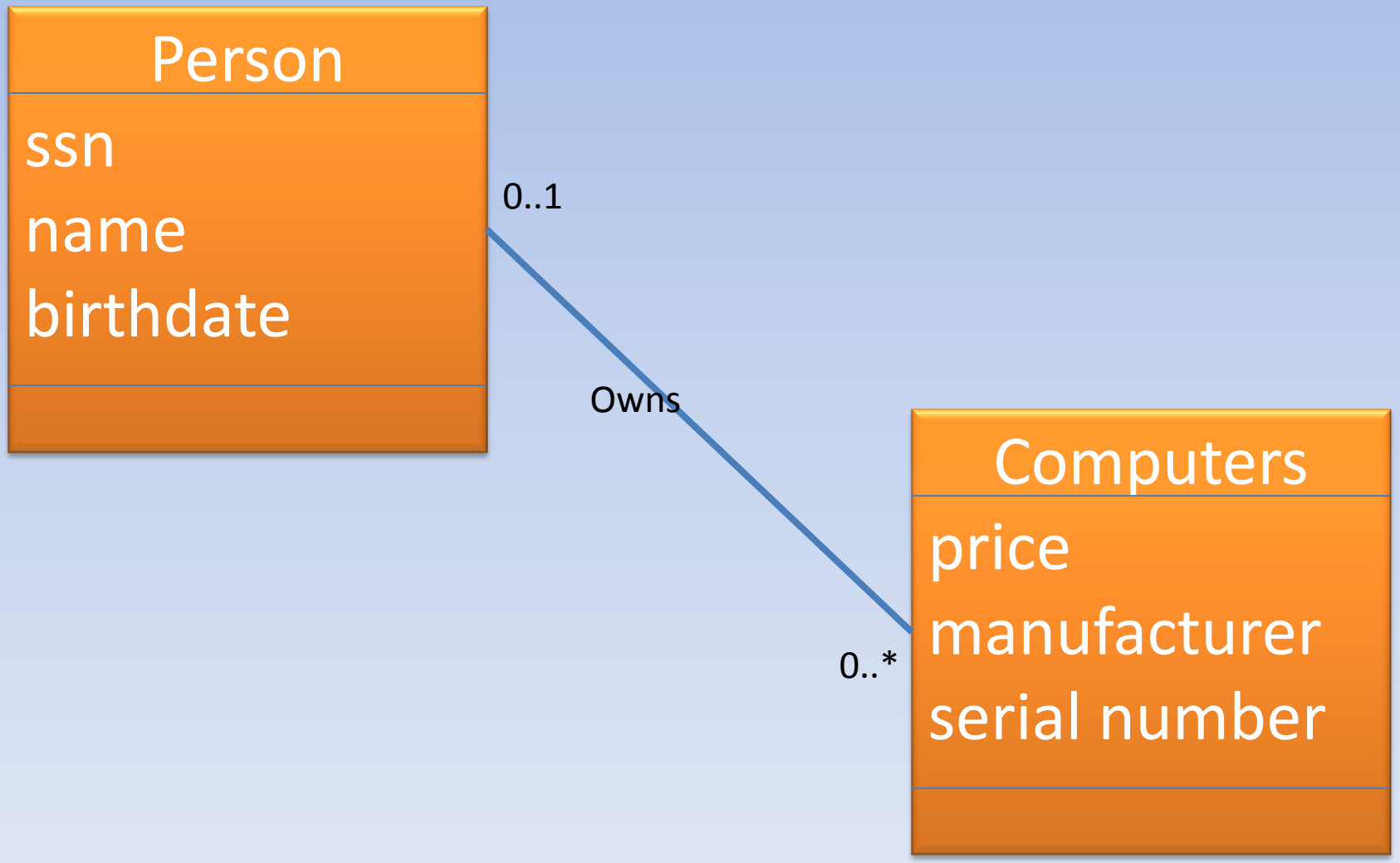
UML - Classes


Computers
price
manufacturer
serial number
PC
speed
ram
hd

Associations

- A Binary relationship between classes.

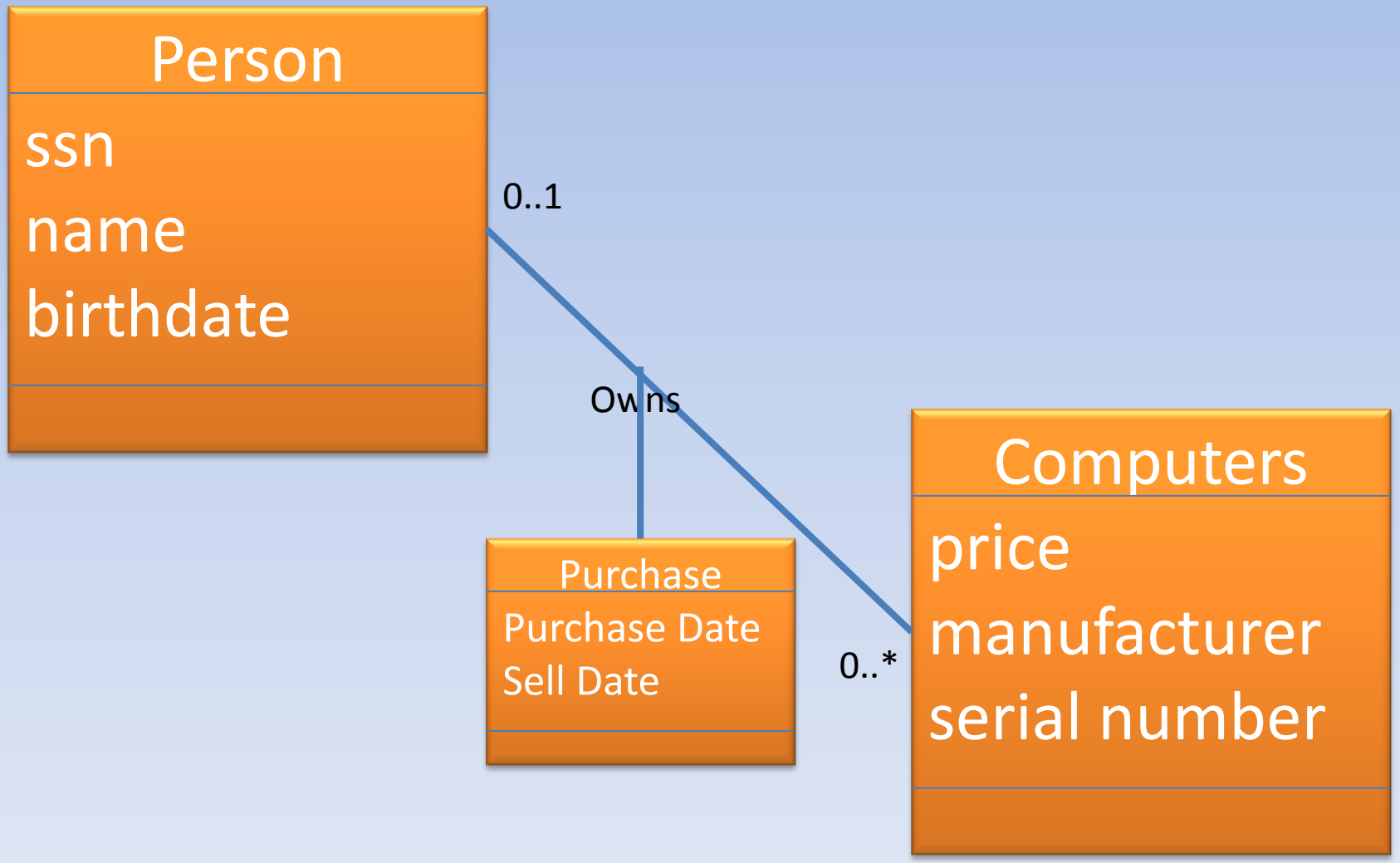
UML - Associations



Association Classes

- Association Class: When attributes attached to an association.

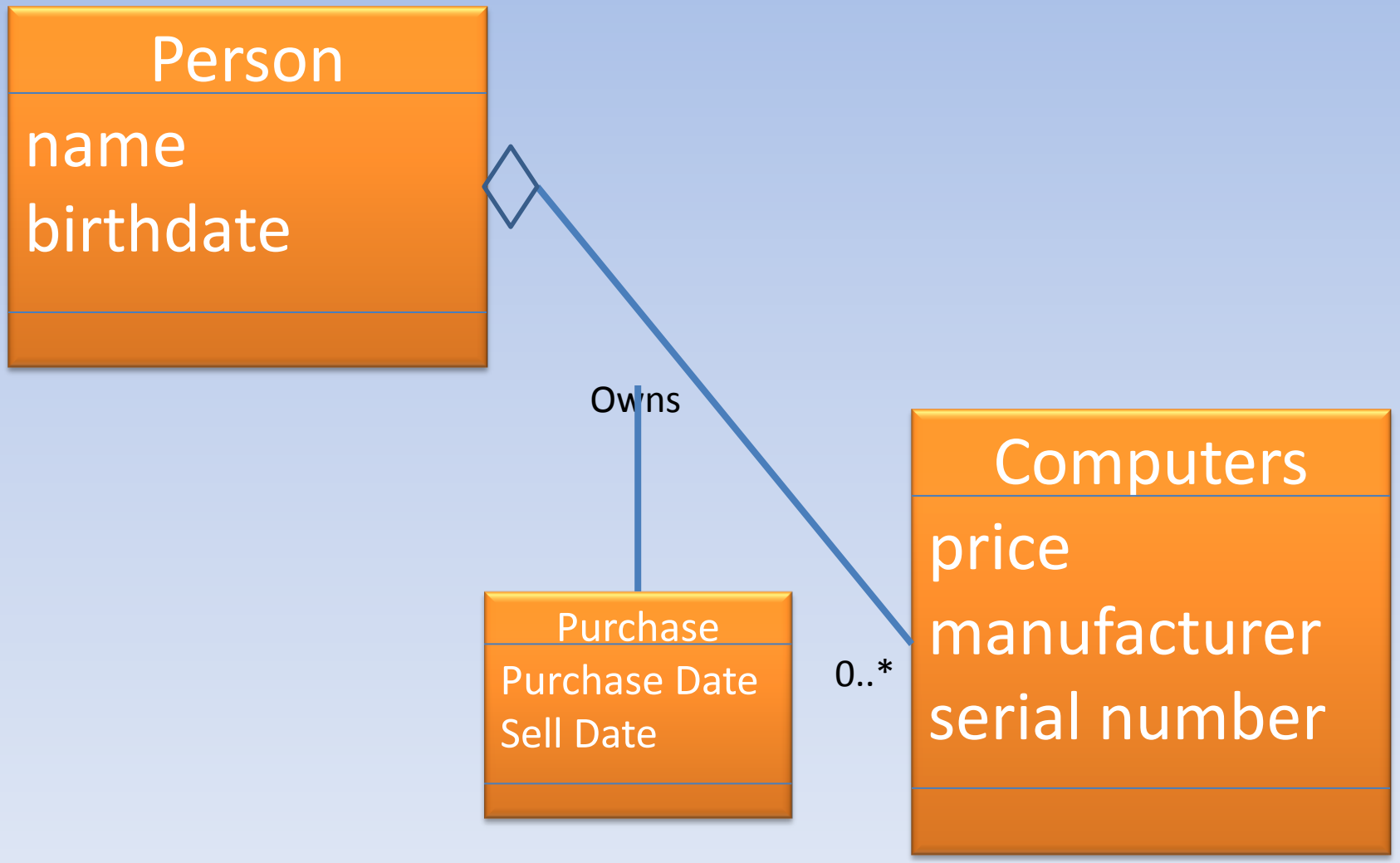
UML – Association Classes



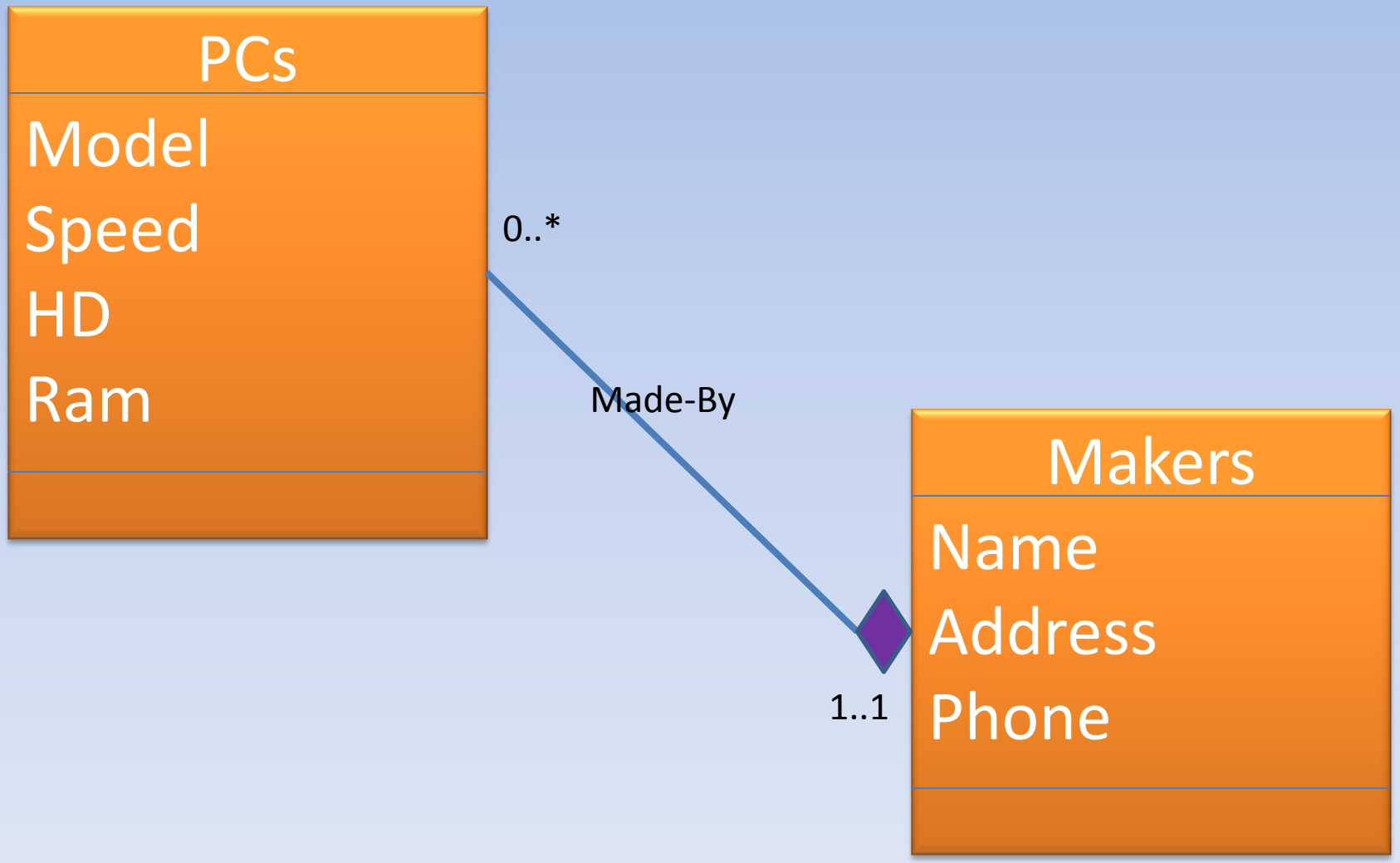
UML: Aggregation & Composition

- Aggregation:
  - An association line between classes that ends with an open diamond.
  - Open diamond label must be 0..1

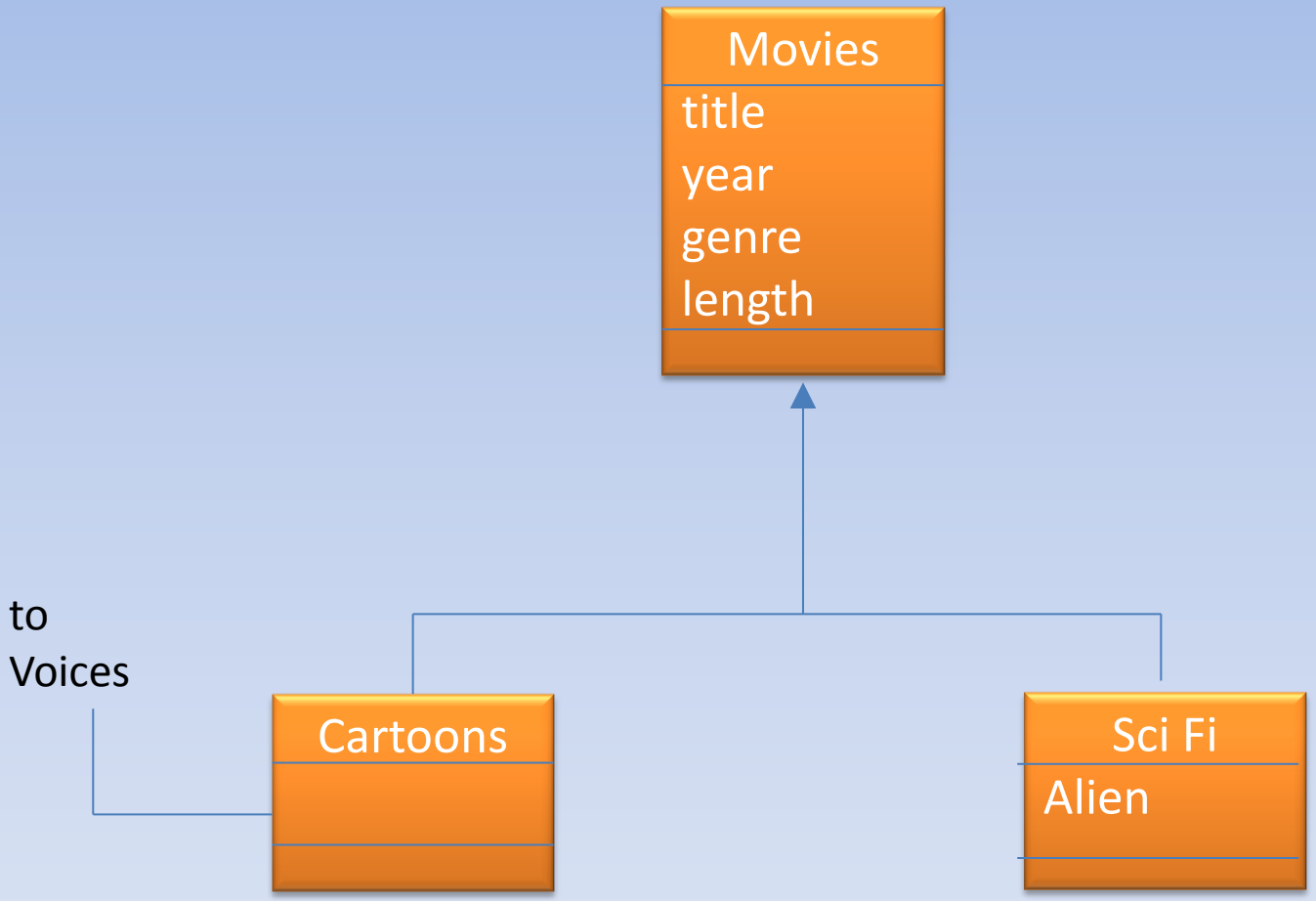
UML – Association Classes
Computer may have owner



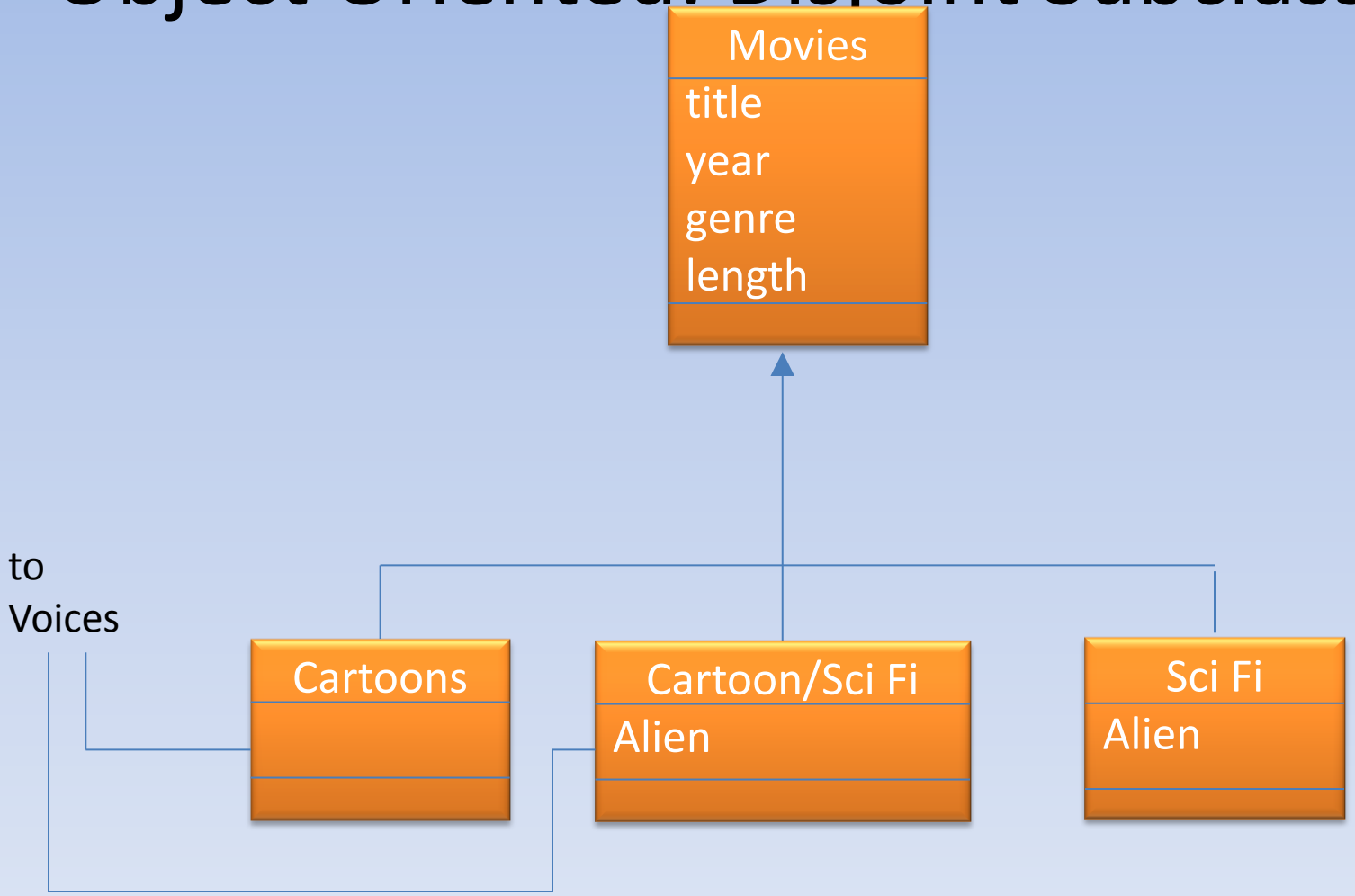
UML – Association Classes
PCs **must-have** Makers



Hierarchy in UML



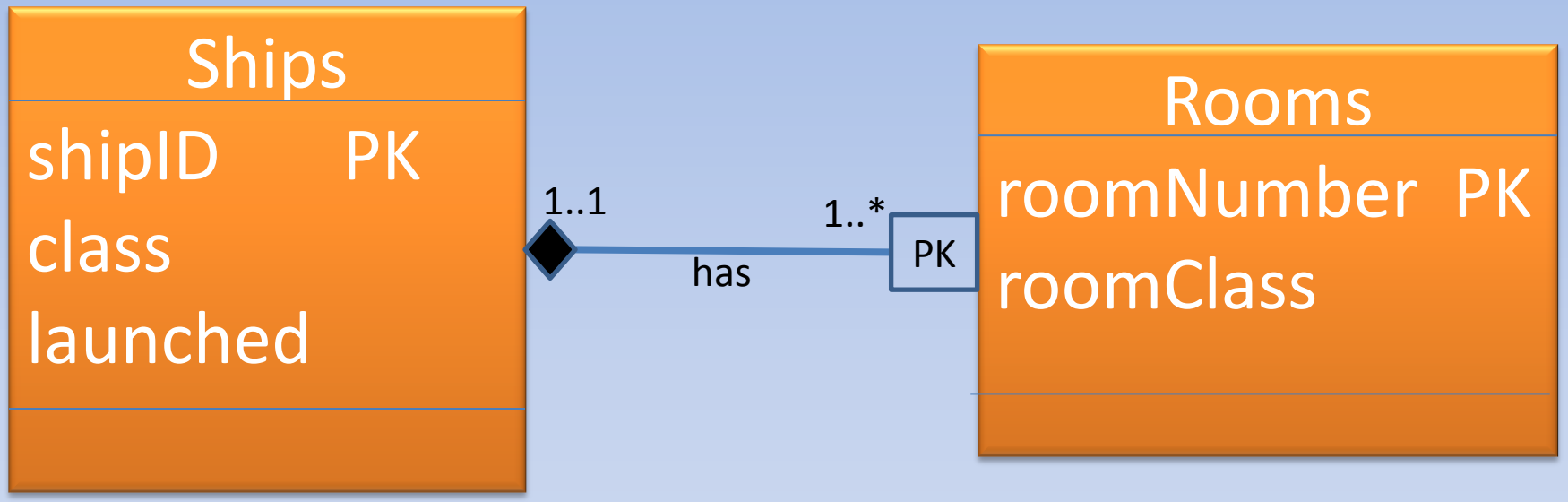
Hierarchy in UML
Object Oriented: Disioint Subclasses



Mapping UML to Relations

- Like with E/R Diagrams
- Classes to Relations
- Associations to Relations
- Combine relations where multiplicity permits.

Weak Entity Set in UML



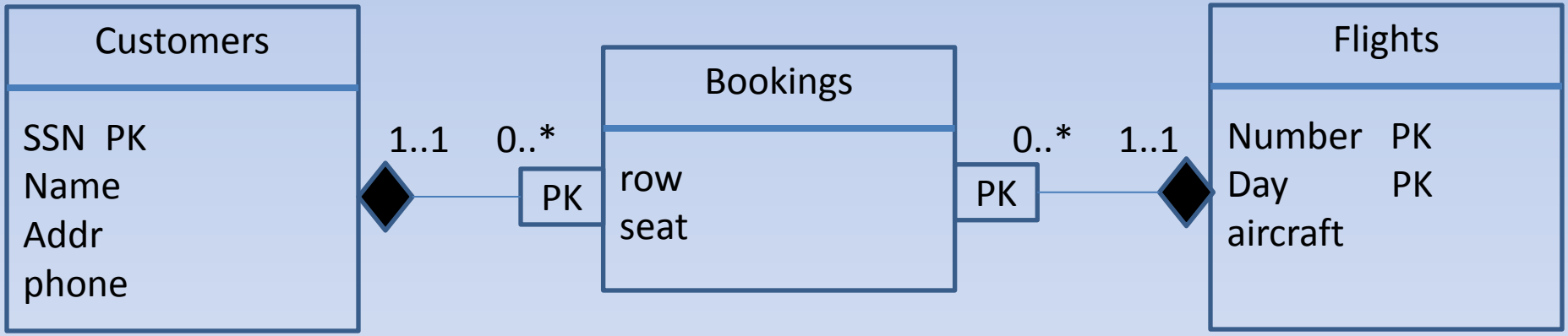
RC Example: Relations

- Ships(shipID, class, launched)
- Rooms(roomNo, roomClass, shipID)

- Don’t need the supporting relationship HAS since the shipID is already contained in Rooms relation.

Exercise 4.8.1

- Need to convert to Relations



Relations Same

- Customers(SSNo, name, addr, phone)
- Flights(number, day, aircraft)
- Bookings(custSSNo, flightNo, flightDay,
  row, seat)

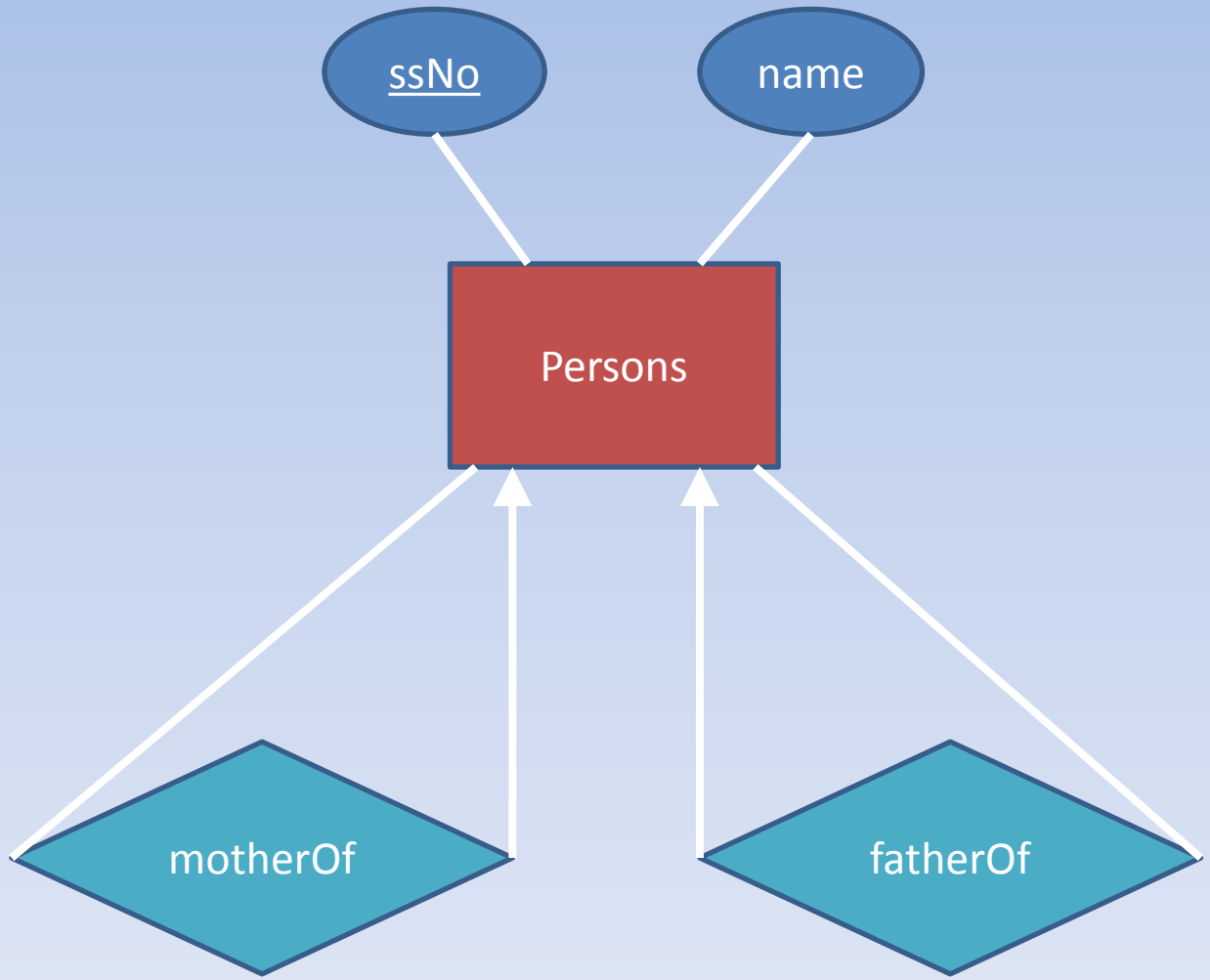
- Relations for toCust and toFlt relationships are not required since the weak entity set Bookings already contains the keys of Customers and Flights.

Genealogy Database

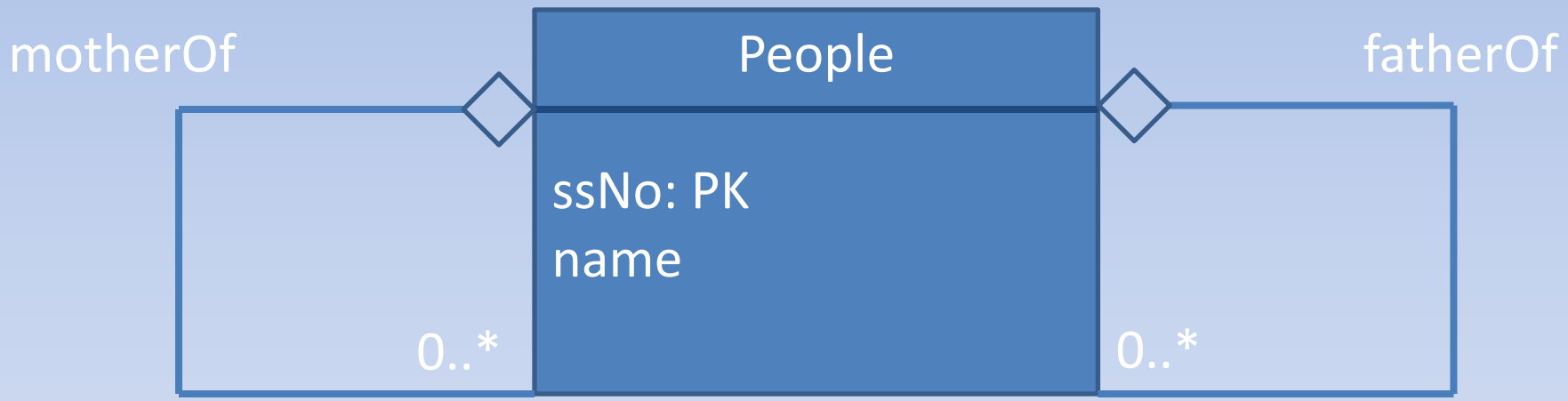
- Design a genealogy database with one entity set: People. The information to record about persons includes their name (an attribute), their mother, father, and children.
  - Children can exist without mother and father (unknown).

- Exercise 4.1.6: Use E/R Notation
- Exercise 4.7.4: Use UML Notation
- Exercise 4.3.1(c): Select and specify keys for both notations.
- Exercise 4.8.2(e): Map to Relations

4.1.6: Solution
E/R Solution



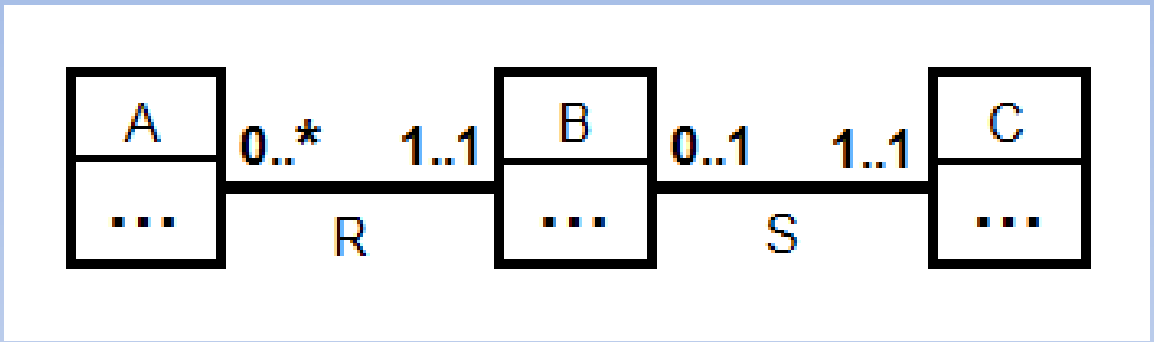
4.7.4
UML Solution



Genealogy Relations

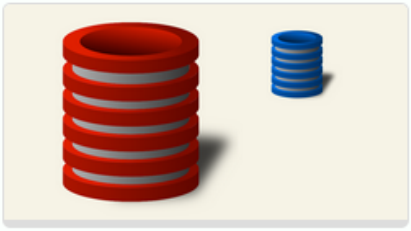
- People(ssNo, name, fatherSSNo, motherSSNo)

UML Constraints



- |A| = 0; |B| = 10; |C| = 0

Is this possible?



Unified Modeling Language

Databases - DB9

Started - Jun 09, 2014

View Course

Your final grade: **100%**.

Download Statement (PDF)

Courseware | Course Info | Discussion | Wiki | Progress | Readings | Extra Problems

- Getting Started
- Unified Modeling Language
  - **UML Data Modeling**
  - **UML to Relations**
  - **UML Quiz**
    Quiz
- Course Completion

UNIFIED MODELING LANGUAGE

You were most recently in UML Quiz. If you're done with that, choose another section on the left.

Object-Oriented DBMS's

- Standards group: ODMG = Object Data Management Group.
- ODL = Object Description Language, like CREATE TABLE part of SQL.
- OQL = Object Query Language, tries to imitate SQL in an OO framework.

Framework – (1)

- ODMG imagines OO-DBMS vendors implementing an OO language like C++ with extensions (OQL) that allow the programmer to transfer data between the database and “host language” seamlessly.

Framework – (2)

- ODL is used to define *persistent* classes, whose objects are stored permanently in the database.
  - ODL classes look like Entity sets with binary relationships, plus methods.
  - ODL class definitions are part of the extended, OO host language.

Object Data Management Group

From Wikipedia, the free encyclopedia

The **Object Data Management Group** (**ODMG**) was conceived in the summer of 1991 at a breakfast with object database vendors that was organized by Rick Cattell of Sun Microsystems. In 1998, the ODMG changed its name from the Object Database Management Group to reflect the expansion of its efforts to include specifications for both object database and object-relational mapping products.

The primary goal of the ODMG was to put forward a set of specifications that allowed a developer to write portable applications for object database and object-relational mapping products. In order to do that, the data schema, programming language bindings, and data manipulation and query languages needed to be portable.

Between 1993 and 2001, the ODMG published five revisions to its specification. The last revision was ODMG version 3.0, after which the group disbanded.

Contents [hide]

Major components of the ODMG 3.0 specification [edit]

- *Object Model.* This was based on the Object Management Group's Object Model. The OMG core model was designed to be a common denominator for object request brokers, object database systems, object programming languages, etc. The ODMG designed a profile by adding components to the OMG core object model.
- *Object Specification Languages.* The ODMG Object Definition Language (ODL) was used to define the object types that conform to the ODMG Object Model. The ODMG Object Interchange Format (OIF) was used to dump and load the current state to or from a file or set of files.
- *Object Query Language (OQL).* The ODMG OQL was a declarative (nonprocedural) language for query and updating. It used SQL as a basis, where possible, though OQL supports more powerful object-oriented capabilities.

TextBook

4.9 Object Definition Language

ODL (*Object Definition Language*) is a text-based language for specifying the structure of databases in object-oriented terms. Like UML, the class is the central concept in ODL. Classes in ODL have a name, attributes, and methods, just as UML classes do. Relationships, which are analogous to UML's associations, are not an independent concept in ODL, but are embedded within classes as an additional family of properties.

ODL Overview

- A class declaration includes:
  1. A name for the class.
  2. Optional key declaration(s).
  3. Element declarations. An *element* is either an attribute, a relationship, or a method.

Class Definitions

```
class <name> {
    <list of element declarations, separated
        by semicolons>
}
```

Attribute and Relationship Declarations

- Attributes are (usually) elements with a type that does not involve classes.

  attribute <type> <name>;

- Relationships connect an object to one or more other objects of one class.

  relationship <type> <name>

    inverse <relationship>;

Inverse Relationships

- Suppose class *C* has a relationship *R* to class *D*.
- Then class *D* must have some relationship *S* to class *C*.
- *R* and *S* must be true inverses.
  - If object *d* is related to object *c* by *R*, then *c* must be related to *d* by *S*.

Example: Attributes and Relationships

```
class Diner {
    attribute string name;
    attribute string addr;
    relationship Set<Dish> serves inverse Dish::servedAt;
}
class Dish {
    attribute string name;
    attribute string manf;
    relationship Set<Dish> servedAt inverse Diner::serves;
}
```

The type of relationship serves is a set of Diner dishes.

The :: operator connects a name on the right to the context containing that name, on the left.

Types of Relationships

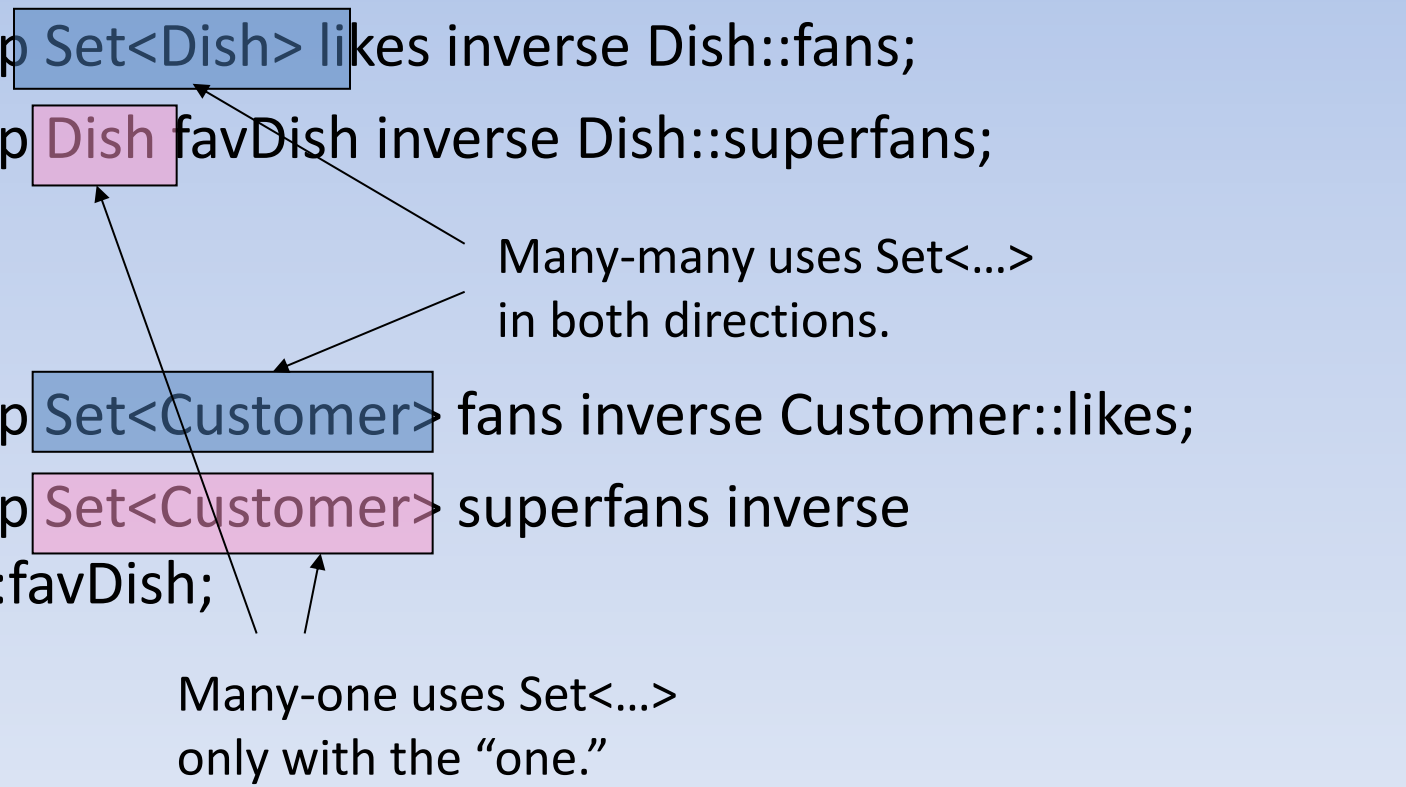
- The type of a relationship is either
  1. A class, like Diner. If so, an object with this relationship can be connected to only one Diner object.
  2. Set<Diner>: the object is connected to a set of Diner objects.
  3. Bag<Diner>, List<Diner>, Array<Diner>: the object is connected to a bag, list, or array of Diner objects.

Multiplicity of Relationships

- All ODL relationships are binary.
- Many-many relationships have Set<…> for the type of the relationship and its inverse.
- Many-one relationships have Set<…> in the relationship of the “one” and just the class for the relationship of the “many.”
- One-one relationships have classes as the type in both directions.

Example: Multiplicity

```
class Customer { …
    relationship Set<Dish> likes inverse Dish::fans;
    relationship Dish favDish inverse Dish::superfans;
}
class Dish { …
    relationship Set<Customer> fans inverse Customer::likes;
    relationship Set<Customer> superfans inverse
    Customer::favDish;
}
```



Another Multiplicity Example

```
class Customer {
    attribute … ;
    relationship Customer husband inverse wife;
    relationship Customer wife inverse husband;
    relationship Set<Customer> buddies
        inverse buddies;
}
```

husband and wife are one-one and inverses of each other.

buddies is many-many and its own inverse. Note no :: needed if the inverse is in the same class.

Coping With Multiway Relationships

- ODL does not support 3-way or higher relationships.
- We may simulate multiway relationships by a "connecting" class, whose objects represent tuples of objects we would like to connect by the multiway relationship.

Connecting Classes

- Suppose we want to connect classes *X*, *Y*, and *Z* by a relationship *R*.
- Devise a class *C*, whose objects represent a triple of objects (*x*, *y*, *z*) from classes *X*, *Y*, and *Z*, respectively.
- We need three many-one relationships from (*x*, *y*, *z*) to each of *x*, *y*, and *z*.

Example: Connecting Class

- Suppose we have Diner and Dish classes, and we want to represent the price at which each Diner sells each dish.
  - A many-many relationship between Diner and Dish cannot have a price attribute as it did in the E/R model.
- One solution: create class Price and a connecting class DDP to represent a related diner, dish, and price.

Example -- Continued

- Since Price objects are just numbers, a better solution is to:
  1. Give DDP objects an attribute price.
  2. Use two many-one relationships between a DDP object and the Diner and Dish objects it represents.

Example -- Concluded

- Here is the definition of DDP:

```
class DDP {
    attribute price:real;
    relationship Diner theDiner inverse Diner::toDDP;
    relationship Dish theDish inverse Dish::toDDP;
}
```

- Diner and Dish must be modified to include relationships, both called toDDP, and both of type Set<DDP>.

Structs and Enums

- Attributes can have a structure (as in C) or be an enumeration.
- Declare with

attribute [Struct or Enum] <name of
struct or enum> { <details> }
<name of attribute>;

- Details are field names and types for a Struct, a list of constants for an Enum.

Example: Struct and Enum

```
class Diner {
    attribute string name;
    attribute Struct Addr
        {string street, string city, int zip} address;
    attribute Enum Lic
        { FULL, Dish, NONE } license;
    relationship …
}
```

Names for the structure and enumeration

names of the attributes

Method Declarations

- A class definition may include declarations of methods for the class.
- Information consists of:
  1. Return type, if any.
  2. Method name.
  3. Argument modes and types (no names).
     - Modes are in, out, and inout.
  4. Any exceptions the method may raise.

Example: Methods

```
real gpa(in string)raises(noGrades);
```

1. The method gpa returns a real number (presumably a student's GPA).
2. gpa takes one argument, a string (presumably the name of the student) and does not modify its argument.
3. gpa may raise the exception noGrades.

The ODL Type System

- Basic types: int, real/float, string, enumerated types, and classes.
- Type constructors:
  - Struct for structures.
  - *Collection types* : Set, Bag, List, Array, and Dictionary (= mapping from a domain type to a range type).
- Relationship types can only be a class or a single collection type applied to a class.

ODL Subclasses

- Usual object-oriented subclasses.
- Indicate superclass with a colon and its name.
- Subclass lists only the properties unique to it.
  - Also inherits its superclass' properties.

Example: Subclasses

- Burgers are a subclass of Dishes:

```
class Burger:Dish {
  attribute string style;
}
```

ODL Keys

- You can declare any number of keys for a class.
- After the class name, add:

  (key <list of keys>)
- A key consisting of more than one attribute needs additional parentheses around those attributes.

Example: Keys

```
class Dish (key name) { …
```

- name is the key for dish.

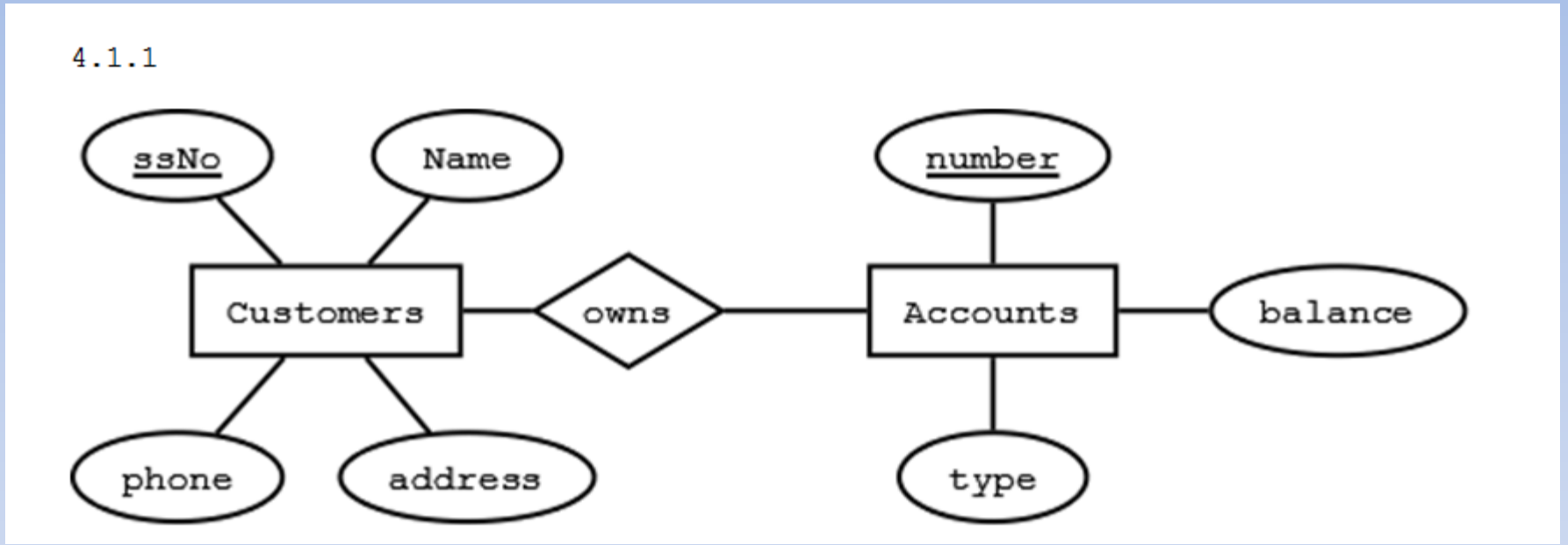
```
class Course (key
  (dept,number),(room, hours)){
```

- dept and number form one key; so do room and hours.

4.1.1

- Design a database for a bank, including information about customers and their accounts.
- Information about a customer includes their name, address, phone, and Social Security number.
- Accounts have numbers, types (e.g., savings, checking) and balances.
- Also record the customer(s) who own an account.
- Draw the E/R diagram for this database. Be sure to include arrows where appropriate, to indicate the multiplicity of a relationship.

4.1.1


4.1.1
ssNo
Name
number
Customers
owns
Accounts
balance
phone
address
type

4.9.1

- In Exercise 4.1.1 was the informal description of a bank database. Render this design in ODL, including keys as appropriate .

4.9.1

```
class Customer (key (ssNo)){
        attribute integer ssNo;
        attribute string name;
        attribute string addr;
        attribute string phone;
        relationship Set<Account> ownsAccts
                inverse Account::ownedBy;
};

class Account (key (number)){
        attribute integer number;
        attribute string type;
        attribute real balance;
        relationship Set<Customer> ownedBy
                inverse Customer::ownsAccts;
```

4.9.1

```
class Customer (key (ssNo)){
        attribute integer ssNo;
        attribute string name;
        attribute string addr;
        attribute string phone;
        relationship Set<Account> ownsAccts
                inverse Account::ownedBy;
};

class Account (key (number)){
        attribute integer number;
        attribute string type;
        attribute real balance;
        relationship Set<Customer> ownedBy
                inverse Customer::ownsAccts;
```