COSC 304
Introduction to Database Systems

Entity-Relationship Modeling

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Conceptual Database Design

Conceptual database design involves modeling the collected information at a high-level of abstraction without using a particular data model or DBMS.

Since conceptual database design occurs independently from a particular DBMS or data model, we need high-level modeling languages to perform conceptual design.

The entity-relationship (ER) model was originally proposed by Peter Chen in 1976 for conceptual design. We will perform ER modeling using Unified Modeling Language (UML) syntax.

Entity-Relationship Modeling

Entity-relationship modeling is a top-down approach to database design that models the data as entities, attributes, and relationships.

The ER model refines entities and relationships by including properties of entities and relationships called *attributes*, and by defining *constraints* on entities, relationships, and attributes.

The ER model conveys knowledge at a high-level (conceptual level) which is suitable for interaction with technical and non-technical users.

Since the ER model is data model independent, it can later be converted into the desired logical model (e.g. relational model).

Example Relation Instances

Emp Relation

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

WorksOn Relation

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

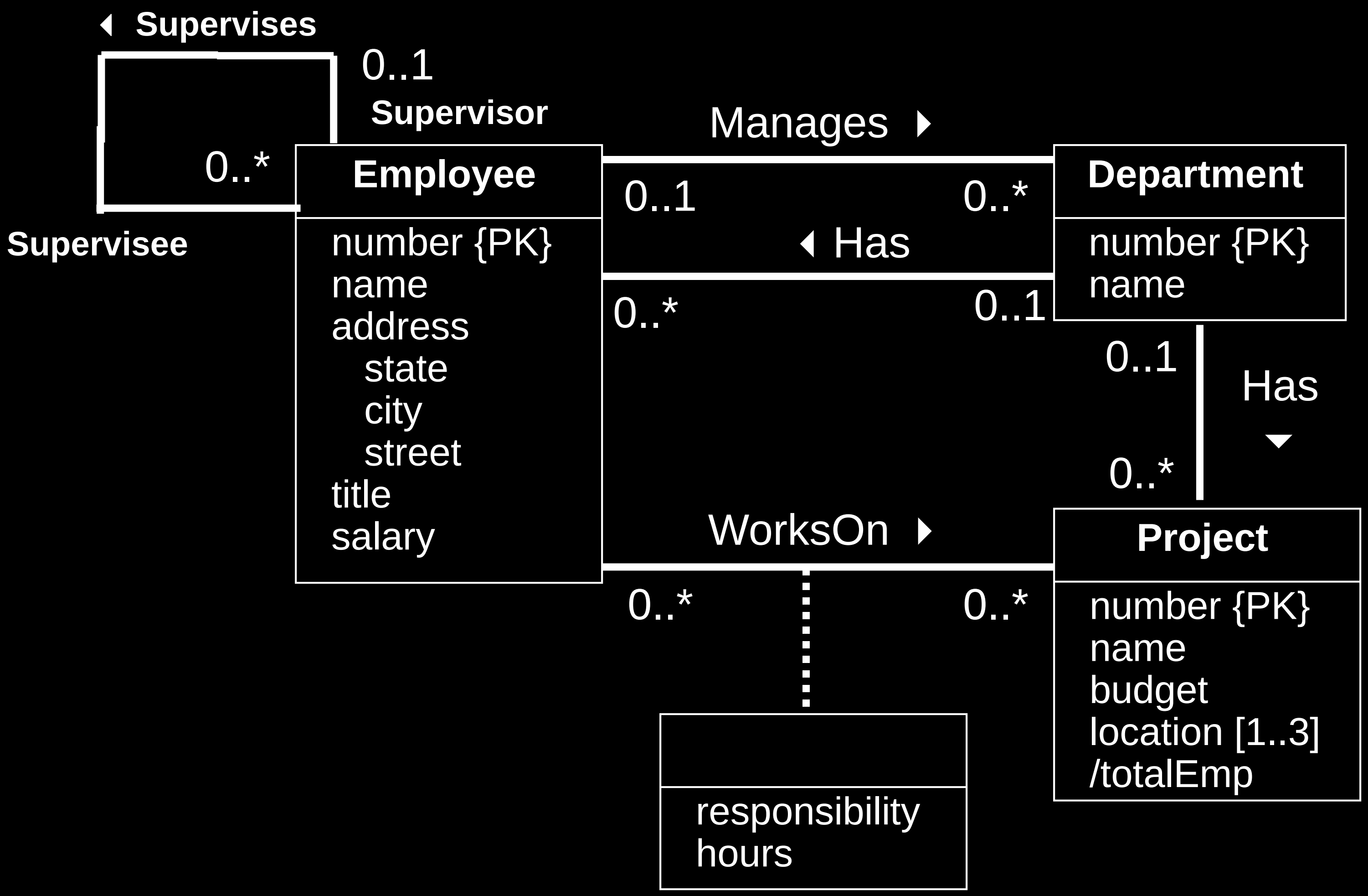
Proj Relation

pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

Dept Relation

dno	dname	mgreno
D1	Management	E8
D2	Consulting	E7
D3	Accounting	E5
D4	Development	null

ER Model Example in UML notation



Entity Types

An ***entity type*** is a group of objects with the same properties which are identified as having an independent existence.

- An entity type is the basic concept of the ER model and represents a group of real-world objects that have properties.
  - Note that an entity type does not always have to be a physical real-world object such as a person or department, it can be an abstract concept such as a project or job.

An ***entity instance*** is a particular example or occurrence of an entity type.

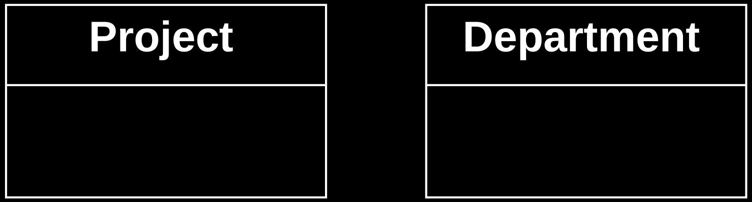
- For example, an entity type is Employee. A entity instance is 'E1 - John Doe'.

An ***entity set*** is a set of entity instances.

Representing Entity Types

Entity types are represented by rectangles with the name of the entity type in the rectangle.

Examples:



Project

Department

- An entity type name is normally a singular noun.
  - That is, use Person instead of People, Project instead of Projects, etc.
- The first letter of each word in the entity name is capitalized.

Entities Question

Question: How many of the following statements are ***true***?

1) Entity types are represented using a rectangle box.

2) An entity is always a physical object.

3) An entity type is named using a plural noun.

4) Employee number is an entity.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Relationship Types

A ***relationship type*** is a set of associations among entity types. Each relationship type has a name that describes its function.

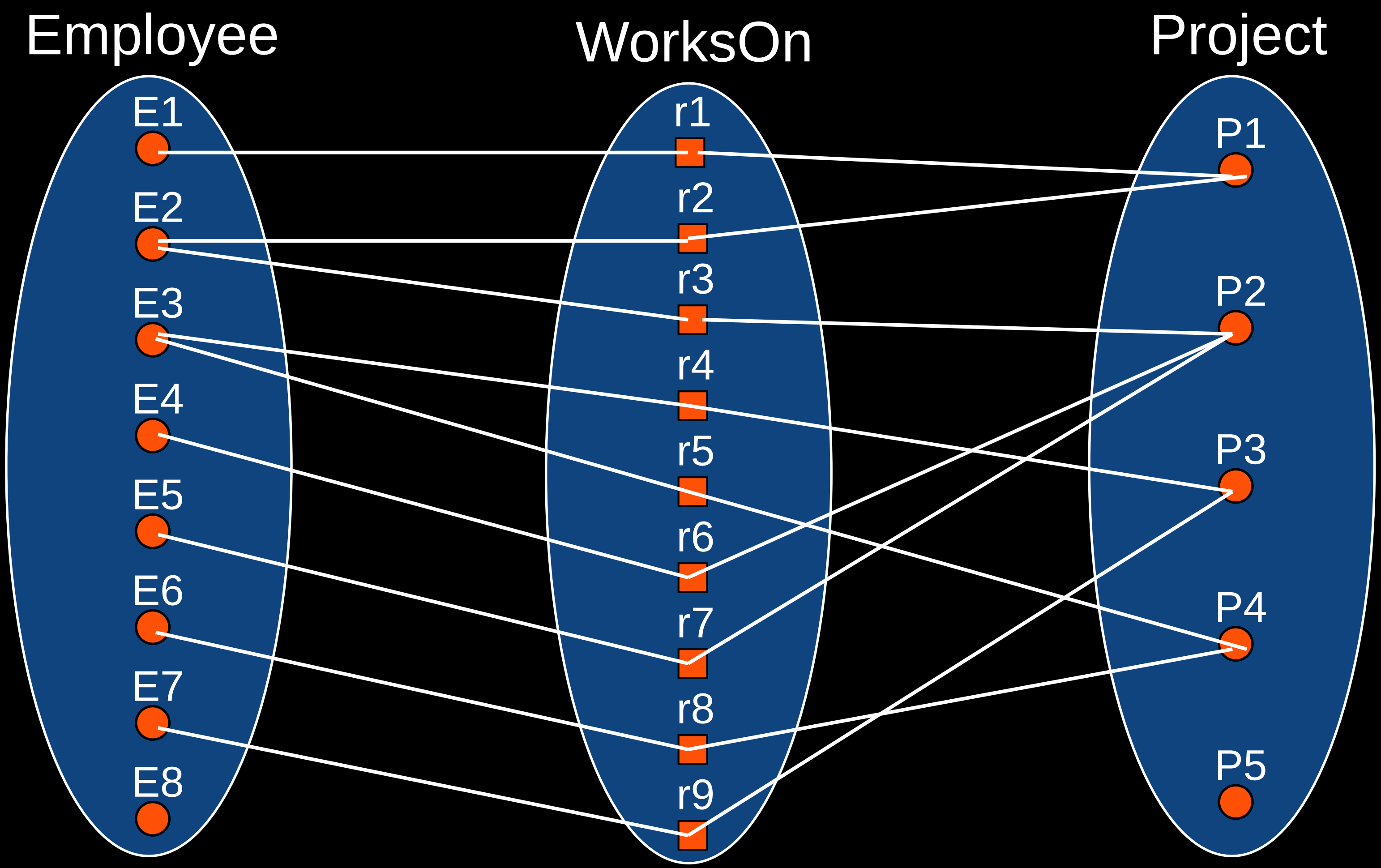
A ***relationship instance*** is a particular occurrence of a relationship type that relates entity instances.

- For example, WorksOn is a relationship type. A relationship instance is that 'E1' works on project 'P1' or (E1,P1).

A ***relationship set*** is a set of relationship instances.

There can be more than one relationship between two entity types.

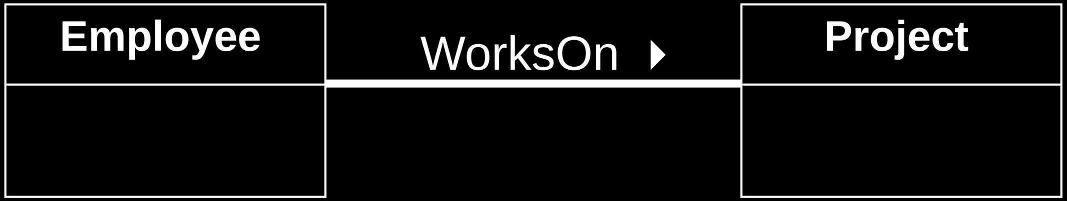
Visualizing Relationships



Note: This is an example of a many-to-many relationship. A project can have more than one employee, and an employee can work on more than one project.

Representing Relationship Types

The relationship type is represented as a labeled edge between the two entity types. The label is applied only in one direction so an arrow indicates the correct way to read it.



- A relationship type name is normally a verb or verb phrase.
- The first letter of each word in the name is capitalized.
- **Do not put arrows on either end of the line.**

Relationship Degree

The ***degree of a relationship type*** is the number of entity types participating in the relationship.

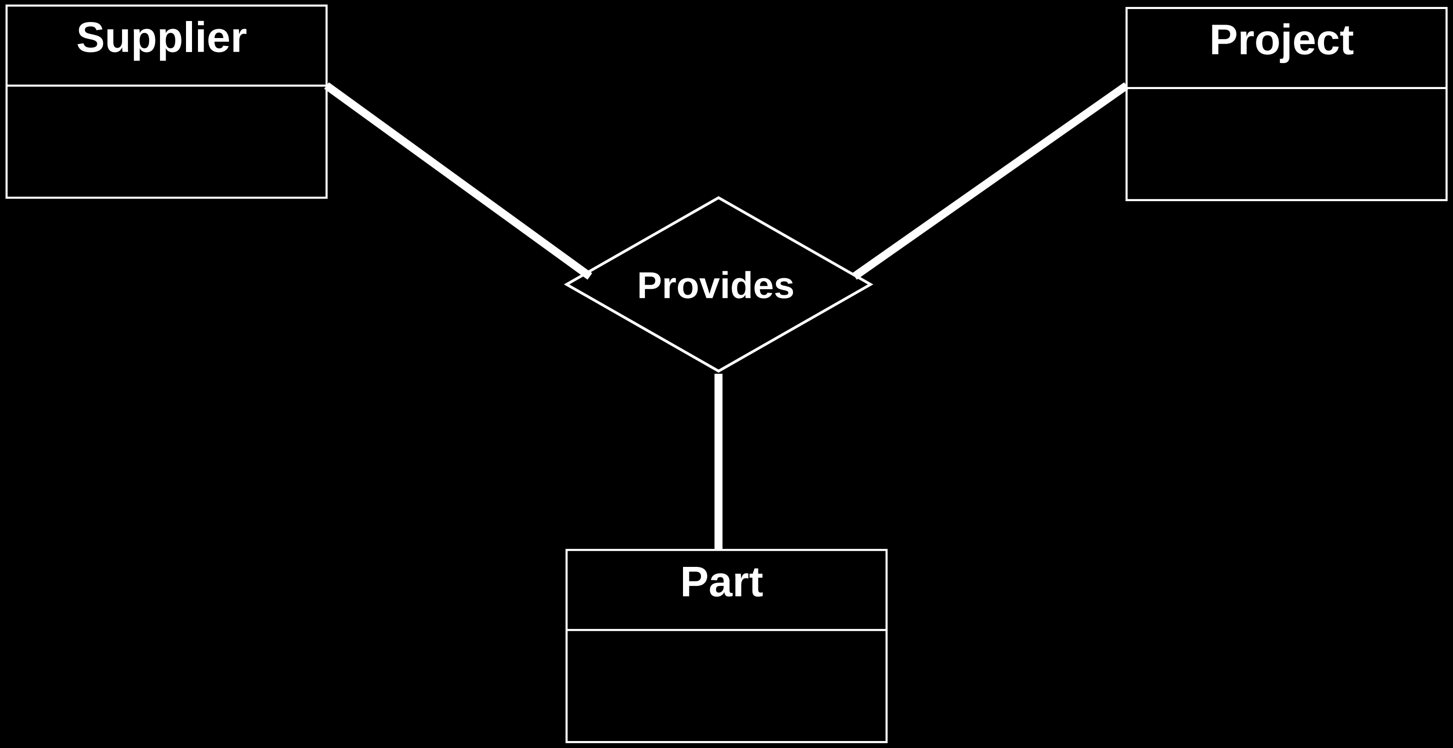
- For example, WorksOn is a relationship type of degree two as the two participating entity types are Employee and Project.
  - Note: This is not the same as degree of a relation which was the number of attributes in a relation.

Relationships of degree two are *binary*, of degree three are *ternary*, and of degree four are *quaternary*.

- Relationships of arbitrary degree N are called *n*-ary.

Use a diamond to represent relationships of degree higher than two.

Ternary Relationship Type Example



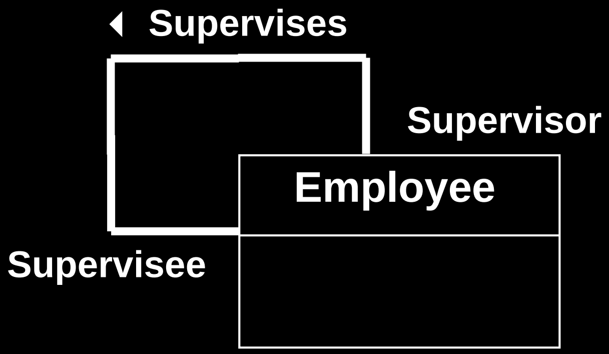
A project may require a part from multiple different suppliers.

Recursive Relationships

A ***recursive relationship*** is a relationship type where the same entity type participates more than once in different roles.

- For example, an employee has a supervisor. The supervisor is also an employee. Each *role* has a *role name*.

Example:



- Note that the degree of a recursive relationship is two as the same entity type participates twice in the relationship.
  - ⇨ It is possible for an entity type to be in a relationship more than twice.

Relationship Question

Question: How many of the following statements are ***true***?

1) Relationships are represented using a directed edge (with arrows).

2) A relationship is typically named using a verb.

3) It is not possible to have a relationship of degree 1.

4) The degree of a relationship is the number of attributes it has.

5) A diamond is used to represent a relationship of degree larger than 2.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Attributes

An ***attribute*** is a property of an entity or a relationship type.

- For example, entity type Employee has attributes name, salary, title, etc.

Some rules:

- By convention, attribute names begin with a lower case letter.
- Each attribute has a *domain* which is the set of allowable values for the attribute.
- Different attributes may share the same domain, but a single attribute may have only one domain.

Simple and Complex Attributes

An attribute is a ***simple attribute*** if it contains a single component with an independent existence.

- For example, salary is a simple attribute.
- Simple attributes are often called *atomic* attributes.

An attribute is a ***composite attribute*** if it consists of multiple components each with an independent existence.

- For example, address is a complex attribute because it consists of street, city, and state components (subattributes).

Question: Is the name attribute of Employee simple or complex?

Single- and Multi-Valued Attributes

An attribute is a ***single-valued attribute*** if it consists of a single value for each entity instance.

- For example, salary is a single-valued attribute.

An attribute is a ***multi-valued attribute*** if it may have multiple values for a single entity instance.

- For example, a telephone number attribute for a person may be multivalued as people may have different phone numbers (home phone number, cell phone number, etc.)

A ***derived attribute*** is an attribute whose value is calculated from other attributes but is not physically stored.

- The calculation may involve attributes within the entity type of the derived attribute and attributes in other entity types.

Attribute Question

Question: How many of the following statements are ***true***?

1) Attributes are properties of either entities or relationships.

2) An attribute may be multi-valued.

3) A composite attribute contains two or more components.

4) Each attribute has a domain representing its data type.

5) A derived attribute is physically stored in the database.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Keys

A ***candidate key*** is a minimal set of attributes that uniquely identifies each instance of an entity type.

- For example, the number attribute uniquely identifies an Employee and is a candidate key for the Employee entity type.

A ***primary key*** is a candidate key that is selected to identify each instance of an entity type.

- The primary key is chosen from a set of candidate keys. For instance, an employee may also have SSN as an attribute. The primary key may be either SSN or number as both are candidate keys.

A ***composite key*** is a key that consists of two or more attributes.

- For example, a course is uniquely identified only by the department code (COSC) and the course number within the department (304).

Key Question

Question: How many of the following statements are ***true***?

1) It is possible to have two candidate keys with different numbers of attributes.

2) A composite key has more than 1 attribute.

3) The computer picks the primary key used in the design.

4) A relationship has a primary key.

5) An attribute has a primary key.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Representing Attributes

In UML attributes are listed in the rectangle for their entity. Tags are used to denote any special features of the attributes.

- primary key: {PK}, partial primary key: {PPK}, alternate key: {AK}
- derived attribute: */attributeName* (e.g. /totalEmp)
- multi-valued attribute: *attributeName* [*minVals*..*maxVals*]
  - e.g. phoneNumber [1..3]

Attributes on Relationships

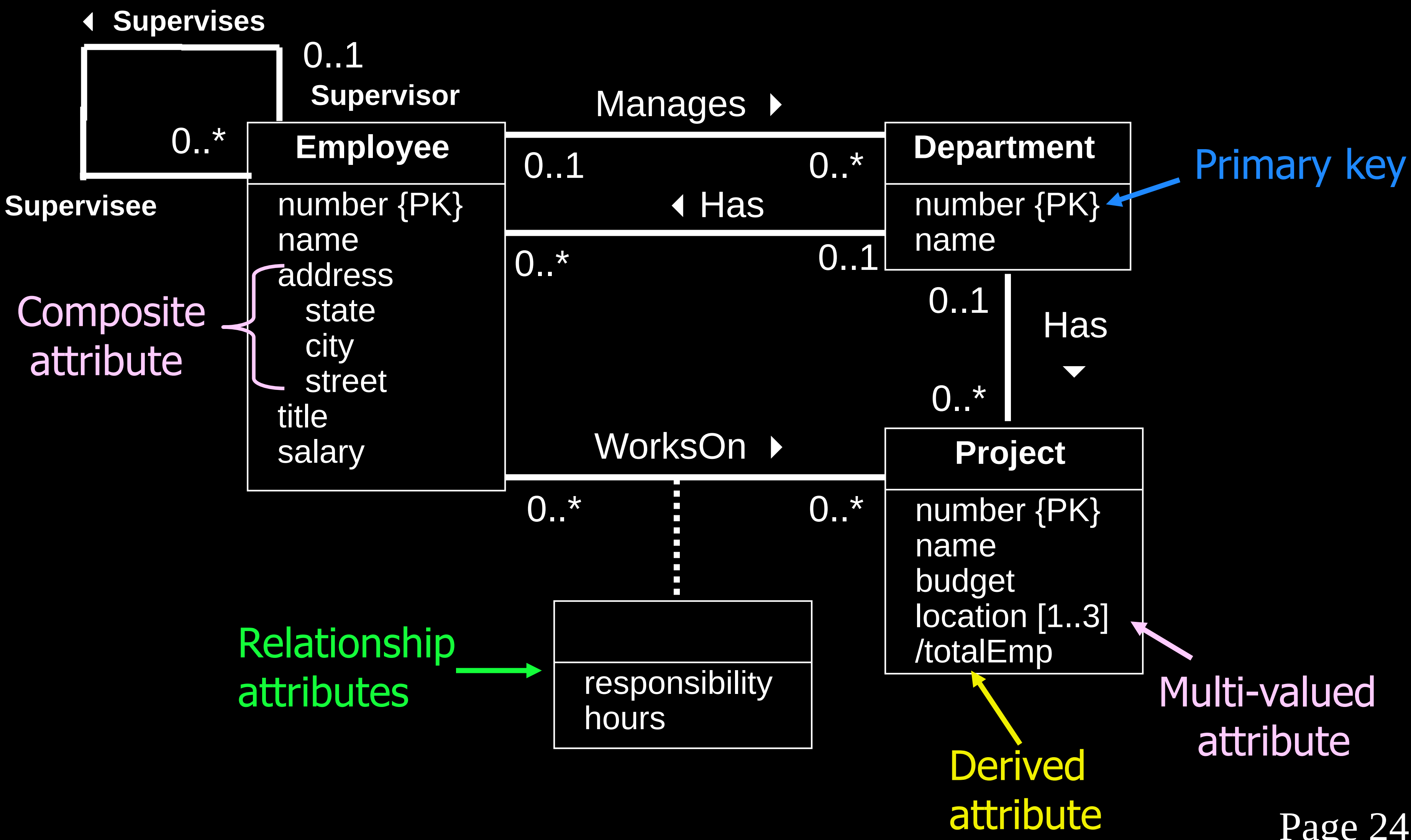
An attribute may be associated with a relationship type.

For example, the WorksOn relationship type has two attributes: responsibility and hours.

Note that these two attributes belong to the relationship and cannot belong to either of the two entities individually (as they would not exist without the relationship).

Relationship attributes are represented as a separate box connected to the relationship using a dotted line.

Attributes in UML Notation



◂ Supervises

0..1

Supervisor

0..*

Supervisee

Employee

number {PK}
name
address
state
city
street
title
salary

Manages ▸

0..1

0..*

◂ Has

0..*

0..1

Department

number {PK}
name

Primary key

0..1

Has ▾

0..*

WorksOn ▸

0..*

0..*

Project

number {PK}
name
budget
location [1..3]
/totalEmp

responsibility
hours

Composite attribute

Relationship attributes

Derived attribute

Multi-valued attribute

ER Design Question #1

Construct a university database where:

- Each student has an id, name, sex, birth date, and GPA.
- Each professor has a name and is in a department.
- Each department offers courses and has professors. A department has a name and a building location.
- Each course has a name and number and may have multiple sections.
- Each section is taught by a professor and has a section number.
- Students enroll in sections of courses. They may only enroll in a course once (and in a single section). A student receives a grade for each of their course sections.

Relationship Cardinalities

Relationship cardinalities or ***multiplicities*** are used to restrict how entity types participate in relationships in order to model real-world constraints.

The ***multiplicity*** is the number of possible occurrences of an entity type that may relate to a single occurrence of an associated entity type through a particular relationship.

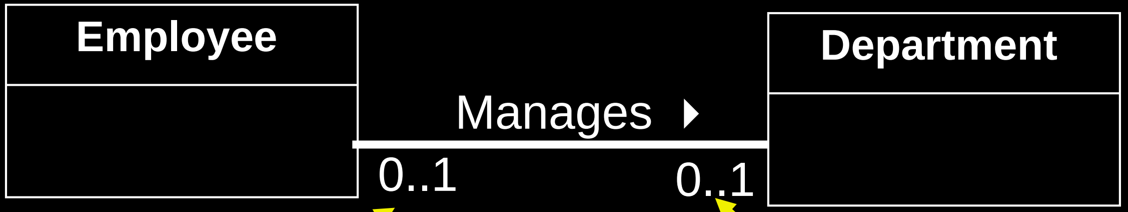
For binary relationships, there are three common types:

- one-to-one (1:1)
- one-to-many (1:* or 1:N)
- many-to-many (*:* or N:M)

One-to-One Relationships

In a one-to-one relationship, each instance of an entity class E1 can be associated with **at most one** instance of another entity class E2 and vice versa.

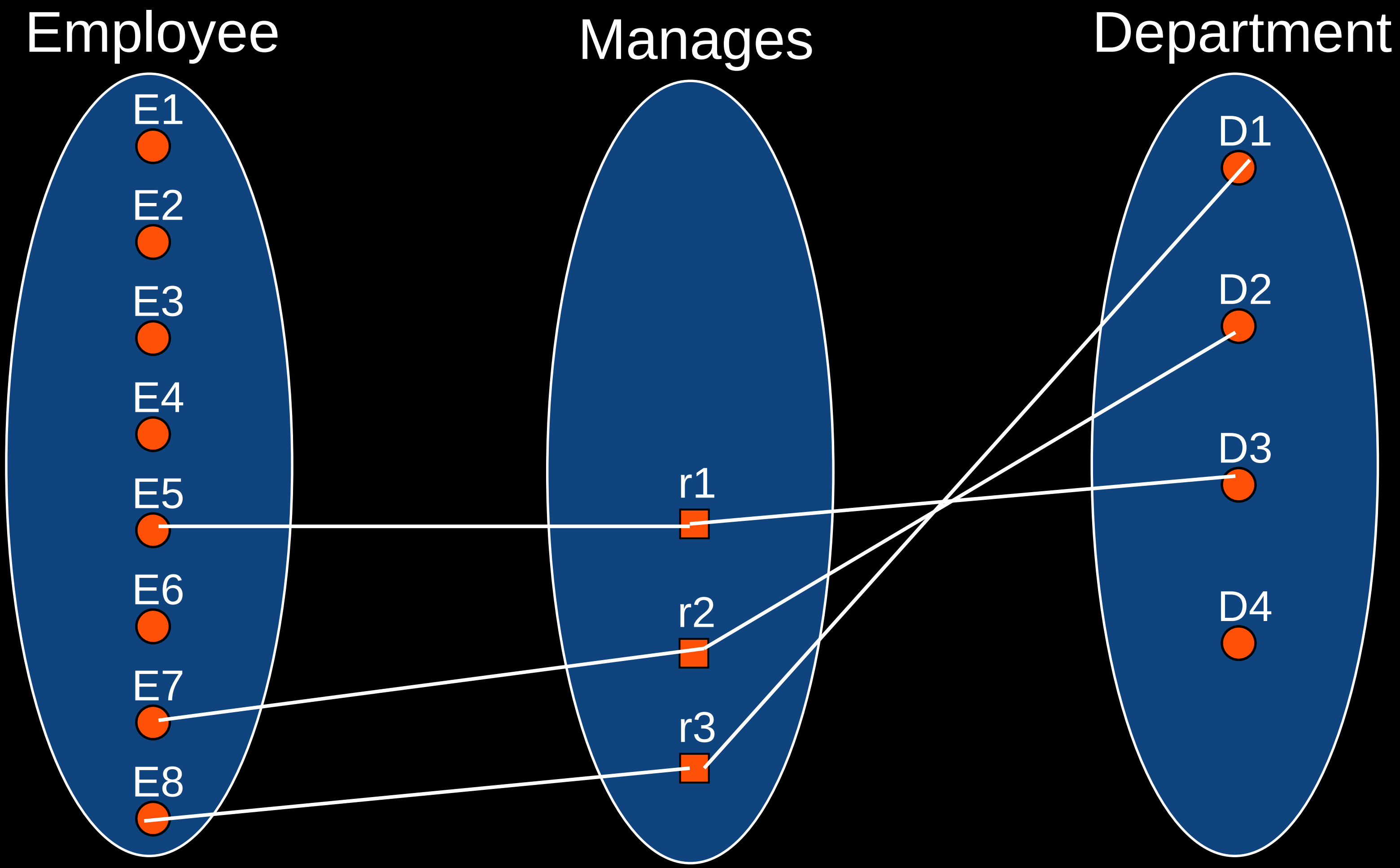
Example: A department may have only one manager, and a manager may manage only one department.



Each department may have a manager.

Each employee manages zero or one department.

One-to-One Relationship Example

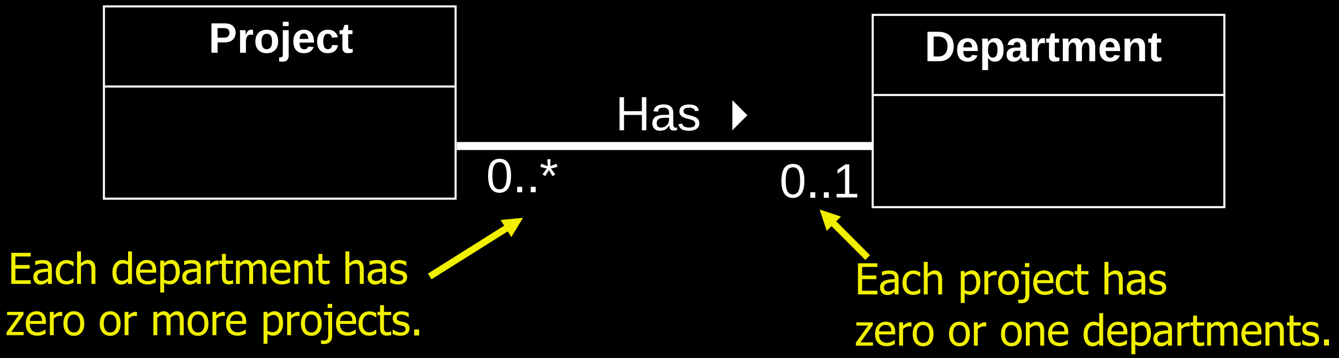


Relationship explanation: A department may have only one manager. A manager (employee) may manage only one department.

One-to-Many Relationships

In a one-to-many relationship, each instance of an entity class E1 can be associated with **more than one** instance of another entity class E2. However, E2 can only be associated with **at most one** instance of entity class E1.

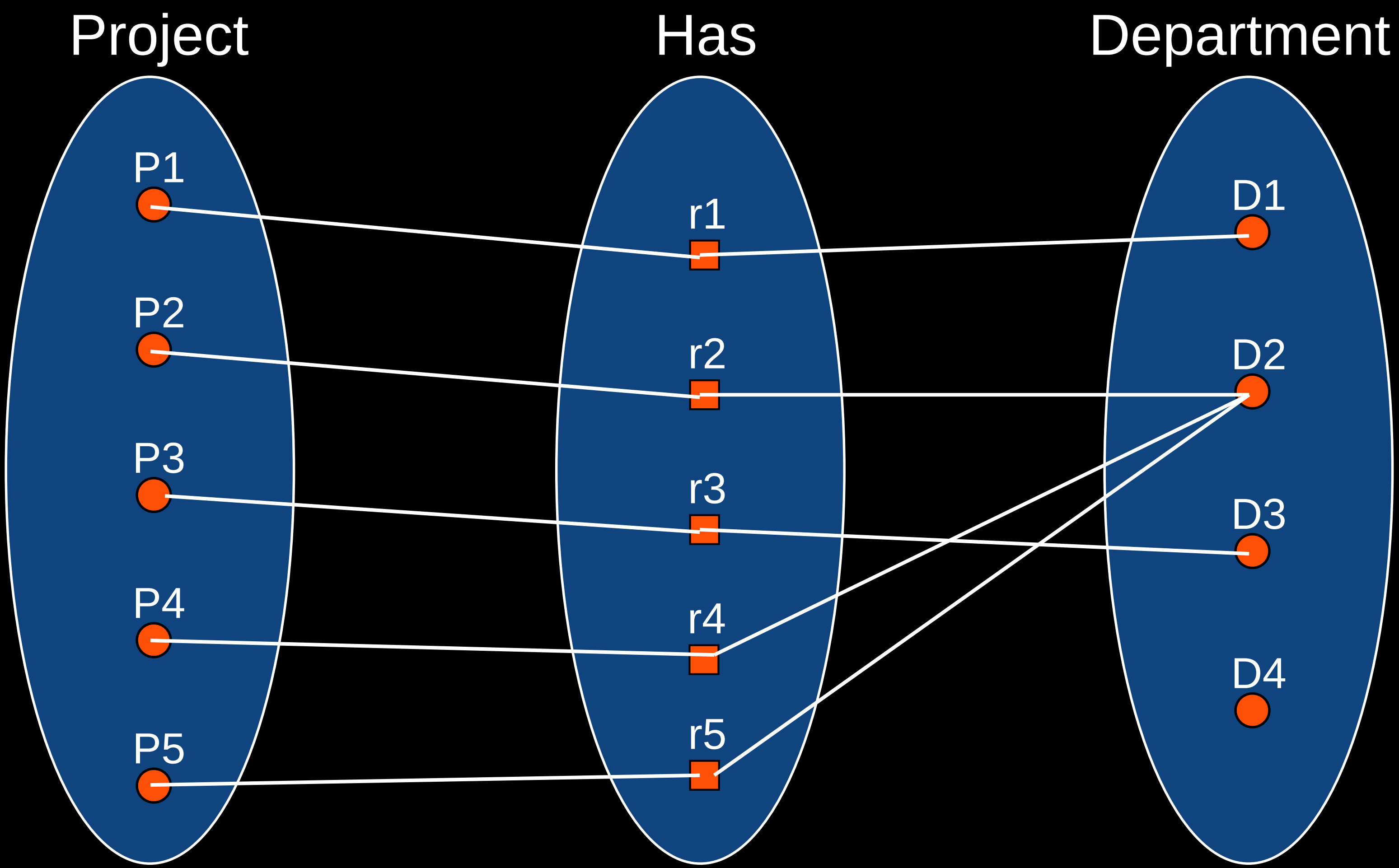
Example: A department may have multiple projects, but a project may have only one department.



Each department has zero or more projects.

Each project has zero or one departments.

One-to-Many Relationship Example

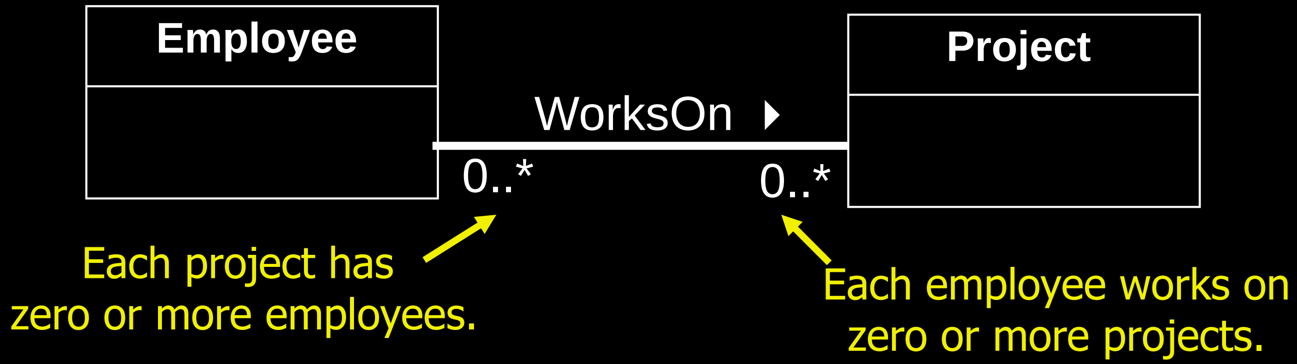


Relationship explanation: A project may be associated with at most one department. A department may have multiple projects.

Many-to-Many Relationships

In a many-to-many relationship, each instance of an entity class E1 can be associated with **more than one** instance of another entity class E2 and vice versa.

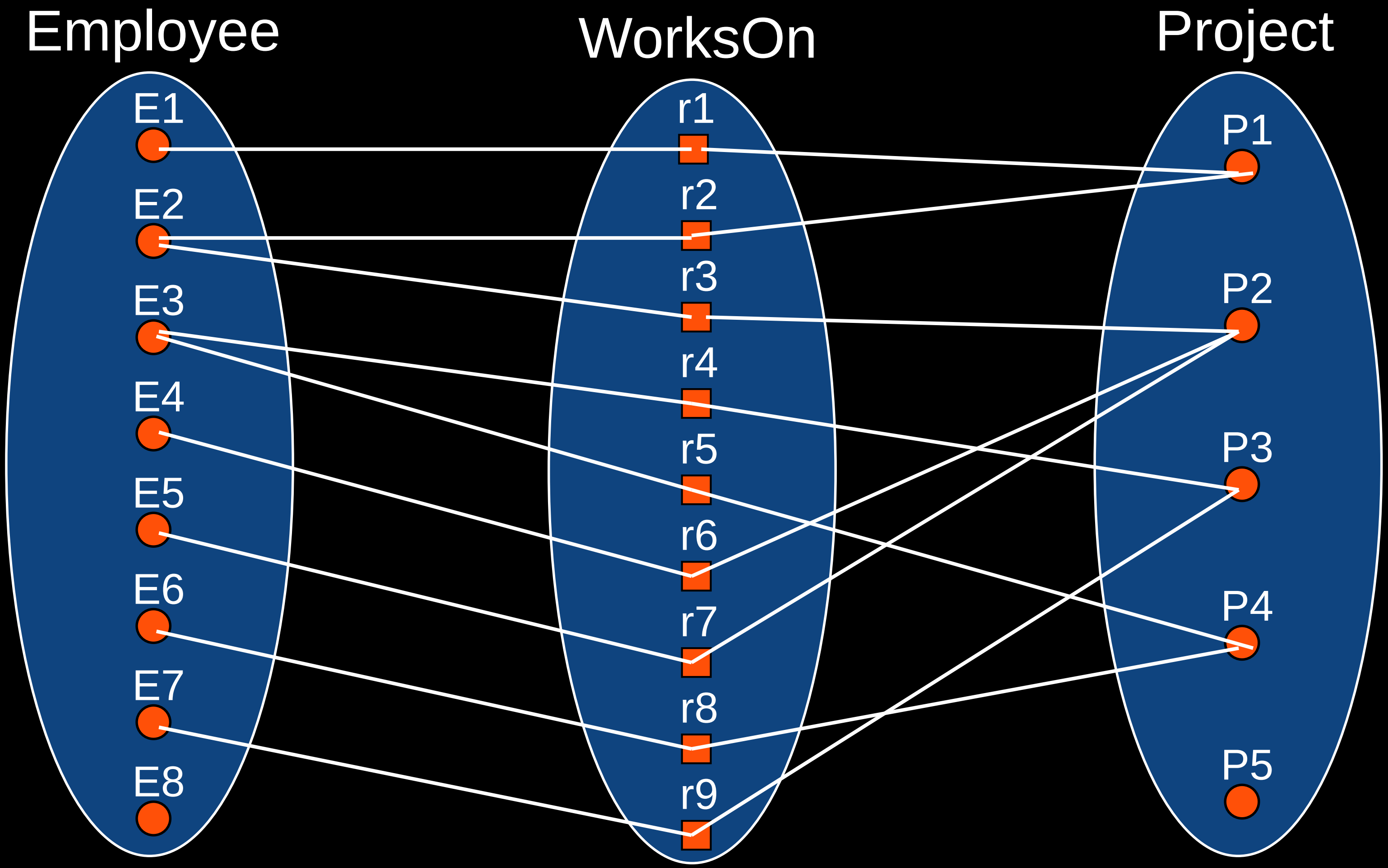
Example: An employee may work on multiple projects, and a project may have multiple employees working on it.



Each project has zero or more employees.

Each employee works on zero or more projects.

Many-to-Many Relationship Example



Participation Constraints

Cardinality is the *maximum* number of relationship instances for an entity participating in a relationship type.

Participation is the *minimum* number of relationship instances for an entity participating in a relationship type.

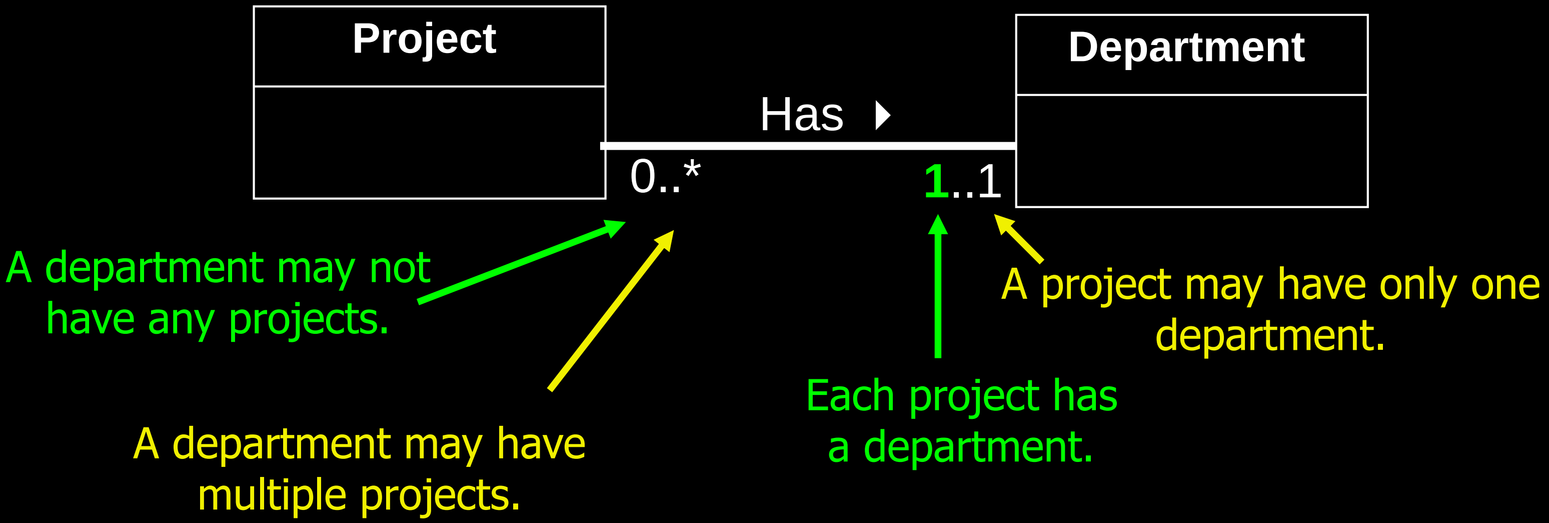
- Participation can be *optional* (zero) or *mandatory (1 or more)*.

If an entity's participation in a relationship is mandatory (also called *total* participation), then the entity's existence depends on the relationship.

- Called an *existence dependency*.

Participation Constraints Example

Example: A project is associated with one department, and a department may have zero or more projects.



A department may not have any projects.

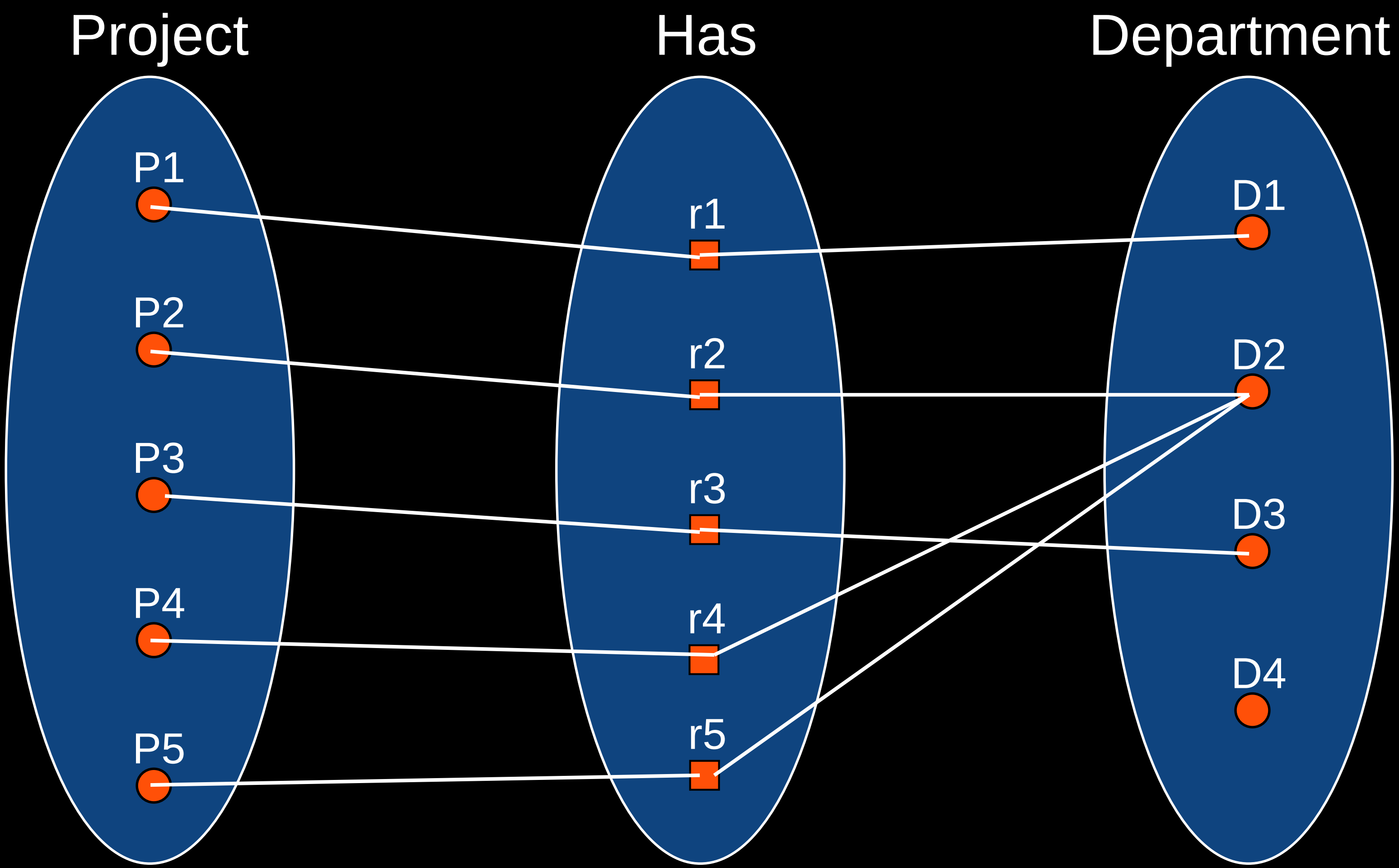
A department may have multiple projects.

Each project has a department.

A project may have only one department.

Note: Every project must participate in the relationship (mandatory).

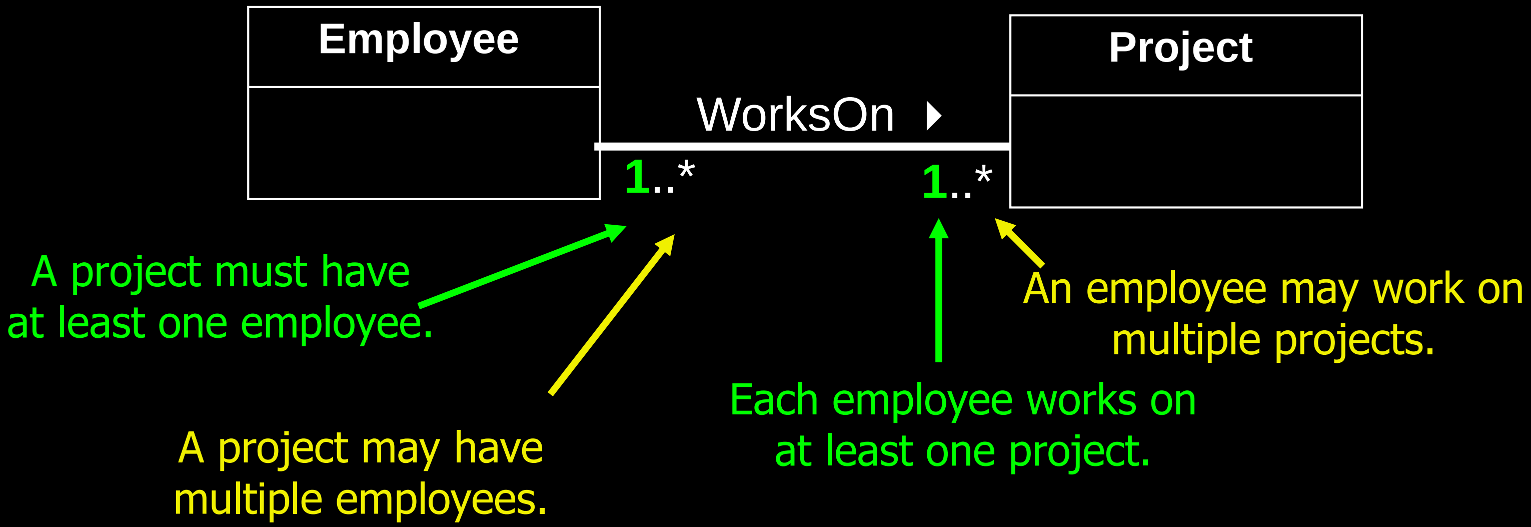
One-to-Many Participation Relationship Example



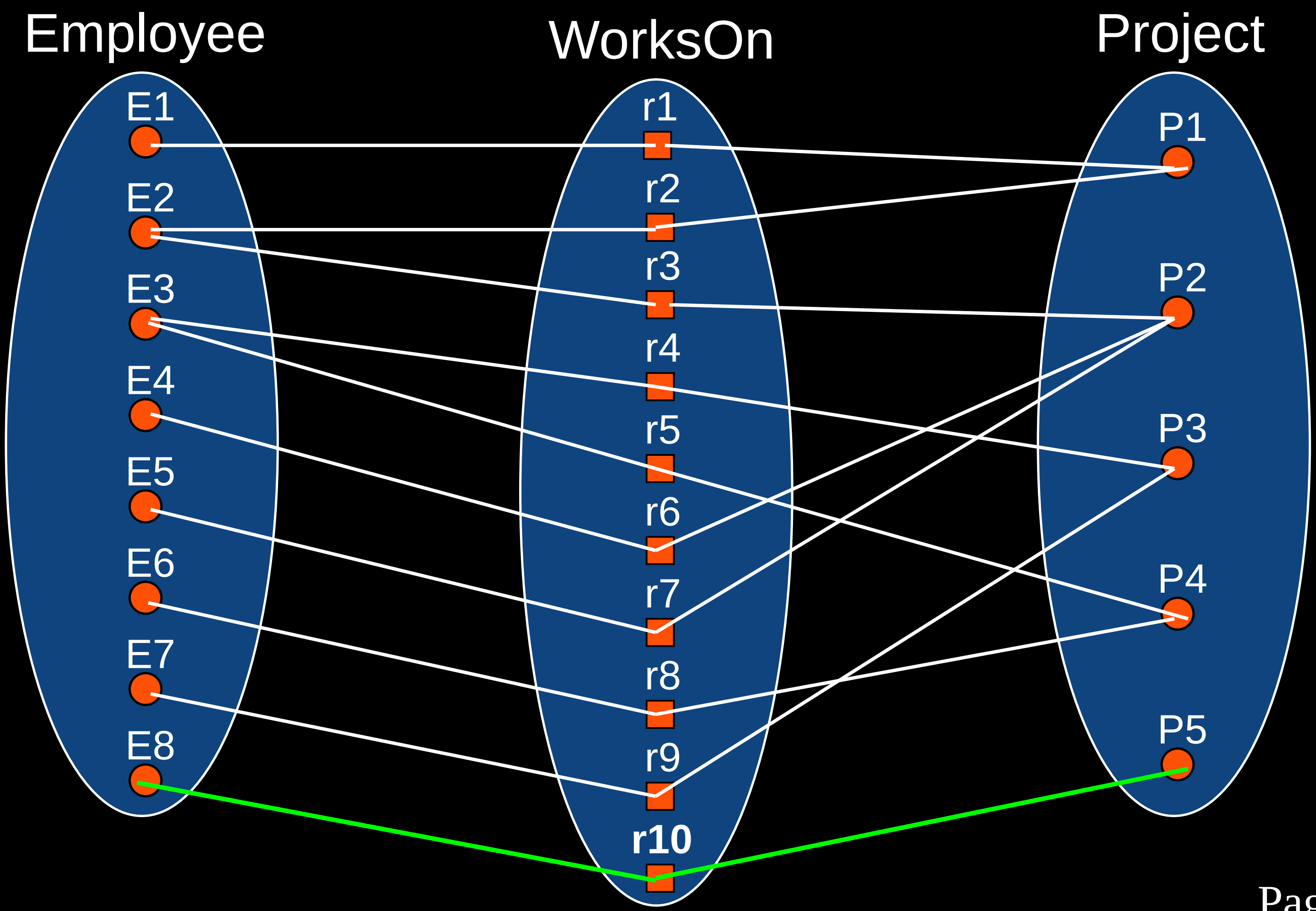
Relationship explanation: A project must be associated with one department. A department may have zero or more projects.

Participation Constraints Example 2

Example: A project must have one or more employees, and an employee must work on one or more projects.


Employee
Project
WorksOn
1..*
1..*
A project must have at least one employee.
A project may have multiple employees.
Each employee works on at least one project.
An employee may work on multiple projects.

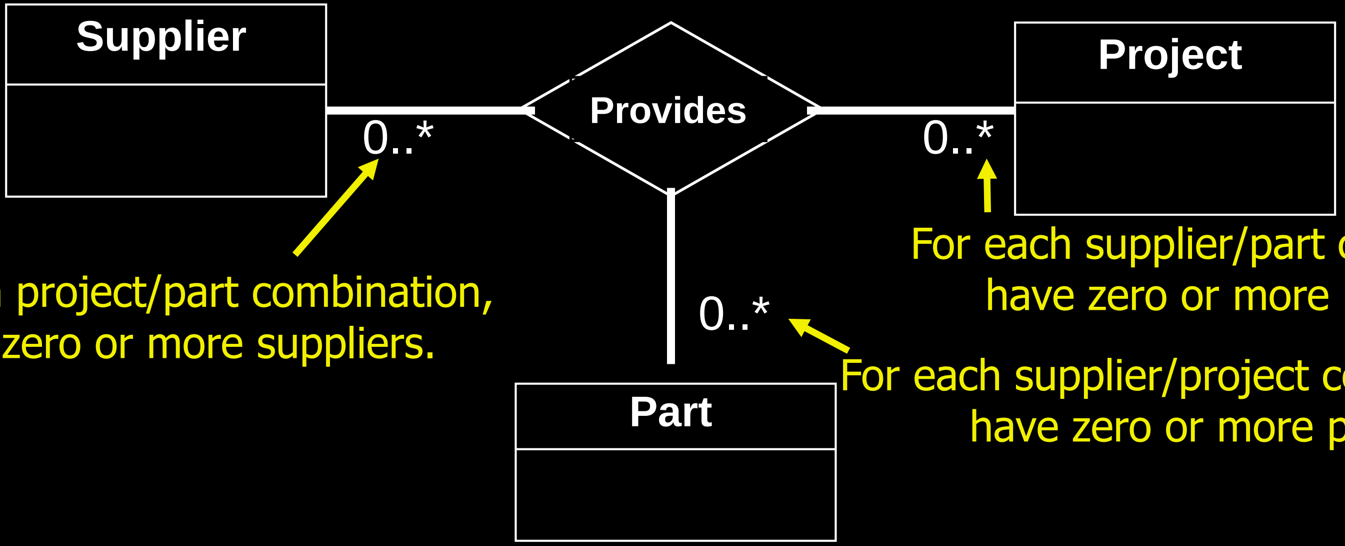
Many-to-Many Relationship Participation Example



Multiplicity of Non-Binary Relationships

The multiplicity in a complex relationship of an entity type is the number of possible occurrences of that entity-type in the *n*-ary relationship when the other (*n*-1) values are fixed.

Example: A supplier may provide zero or more parts to a project. A project may have zero or more suppliers, and each supplier may provide to the project zero or more parts.



For each project/part combination, have zero or more suppliers.

For each supplier/part combination, have zero or more projects.

For each supplier/project combination, have zero or more parts.

Challenges with Multiplicity of Non-Binary Relationships

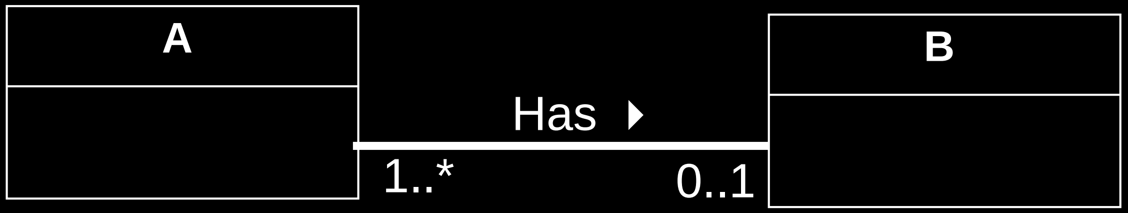
The participation constraint for *N*-ary relationships (minimum cardinality) is ***ambiguous*** in UML notation. Example:

- Constraint: A project has at least 1 supplier per part.
- Initial idea: Multiplicity 1..* beside Supplier should force their to be a Supplier for each Part/Project combination.
- Problem: Two different interpretations of Part/Project combination results in unforeseen consequences:
  - Actual tuple: All entities must always participate (as have actual tuples) so minimum values for all entities would be 1.
  - Potential tuple: 1 beside Supplier implies always a Supplier for every Part/Project combination. Not true in practice.

Bottom line: We will avoid problem of specifying participation constraints for *N*-ary relationships. One way to avoid it is convert a relationship into an entity with *N* binary relationships.

Relationship Cardinality Question

Question: How many of the following statements are ***true***?



1) An entity of type A must be related to an entity of type B.

2) An entity of type A may be related to more than one entity B.

3) This is a 1-to-many relationship between A and B.

4) An entity of type B must be related to an entity of type A.

5) An entity of type B must be related to more than one entity of type A.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Multiplicity Practice Question

Consider the university database developed before. Write multiplicities into the ER diagram given that:

- A department must offer at least 2 courses and no more than 20 courses. Courses are offered by only one department.
- A course may have multiple sections, but always has at least one section.
- A student may enroll for courses (but does not have to).
- A professor may be in multiple departments (at least 1), and a department must have at least 3 professors.
- A section is taught by at least one professor, but may be taught by more than one. A professor does not have to teach.

Strong and Weak Entity Types

A ***strong entity type*** is an entity type whose existence is not dependent on another entity type.

- A strong entity type always has a primary key of its own attributes that uniquely identifies its instances.

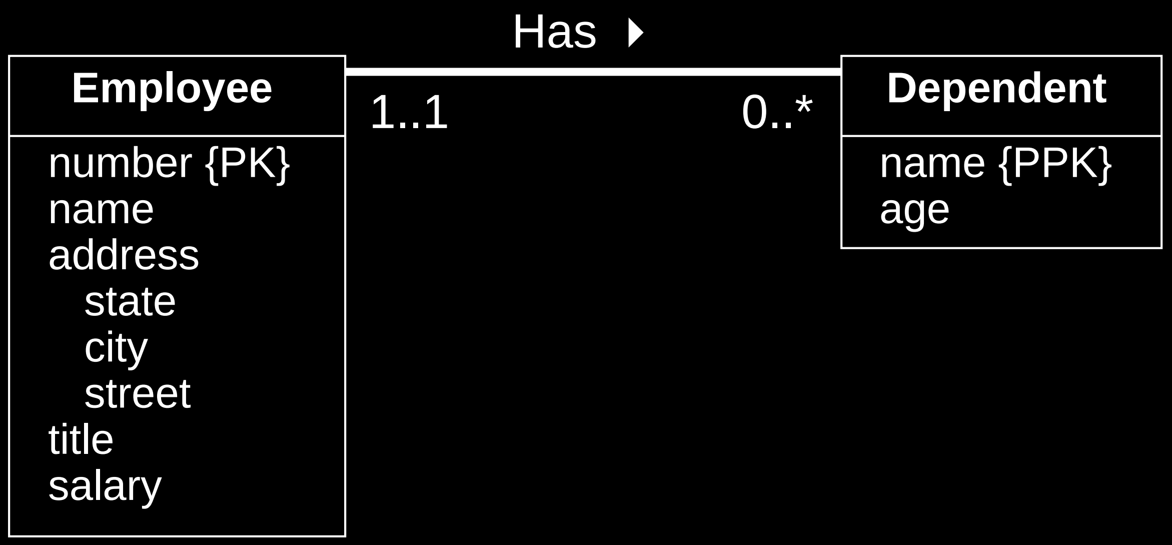
A ***weak entity type*** is an entity type whose existence is dependent on another entity type.

- A weak entity type does not have a set of its own attributes that uniquely identifies its instances.

A common example of strong and weak entity types are employees and their dependents:

- An employee is a strong entity because it has an employee number to identify its instances.
- A dependent (child) is a weak entity because the database does not store a key for each child, but rather they are identified by the parent's employee number and their name.

Weak Entities in UML



Has ▸

1..1

0..*

Employee

number {PK}
name
address
  state
  city
  street
title
salary

Dependent

name {PPK}
age

Strong and Weak Entity Question

Question: How many of the following statements are ***true***?

1) A weak entity has its own primary key.

2) A strong entity has its own primary key.

3) A weak entity must be associated (identified) by a strong entity.

4) A weak entity can have a relationship with another entity besides its identifying strong entity.

5) The attribute(s) of a weak entity used to identify it with its associated strong entity are noted as `{PPK}` in the UML model.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Problems with ER Models

When modeling a Universe of Discourse (UoD) using ER models, there are several challenges that you have.

The first, basic challenge is knowing when to model a concept as an entity, a relationship, or an attribute.

In general:

- Entities are nouns.
  - You should be able to identify a set of key attributes for an entity.
- Attributes are properties and may be nouns or adjectives.
  - Use an attribute if it relates to one entity and does not have its own key.
  - Use an entity if the concept may be shared by entities and has a key.
- Relationships should generally be binary.
  - Note that non-binary relationships can be modeled as an entity instead.

Modeling Traps

There are several different "modeling traps" that you can fall into when designing your ER model.

Two connection traps that we will look at are:

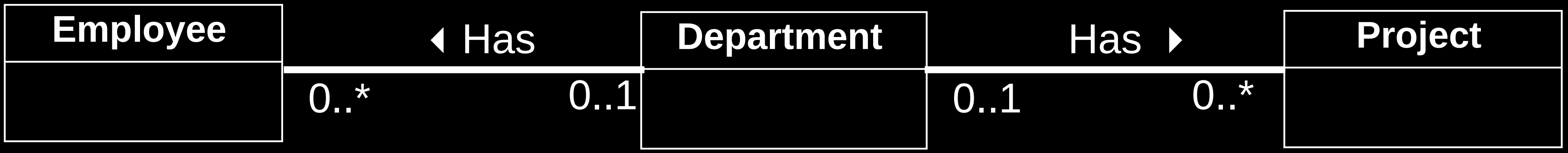
- Fan traps
- Chasm traps

Fan Traps

A ***fan trap*** is when a model represents a relationship between entity types, but the pathway between certain entity instances is ambiguous.

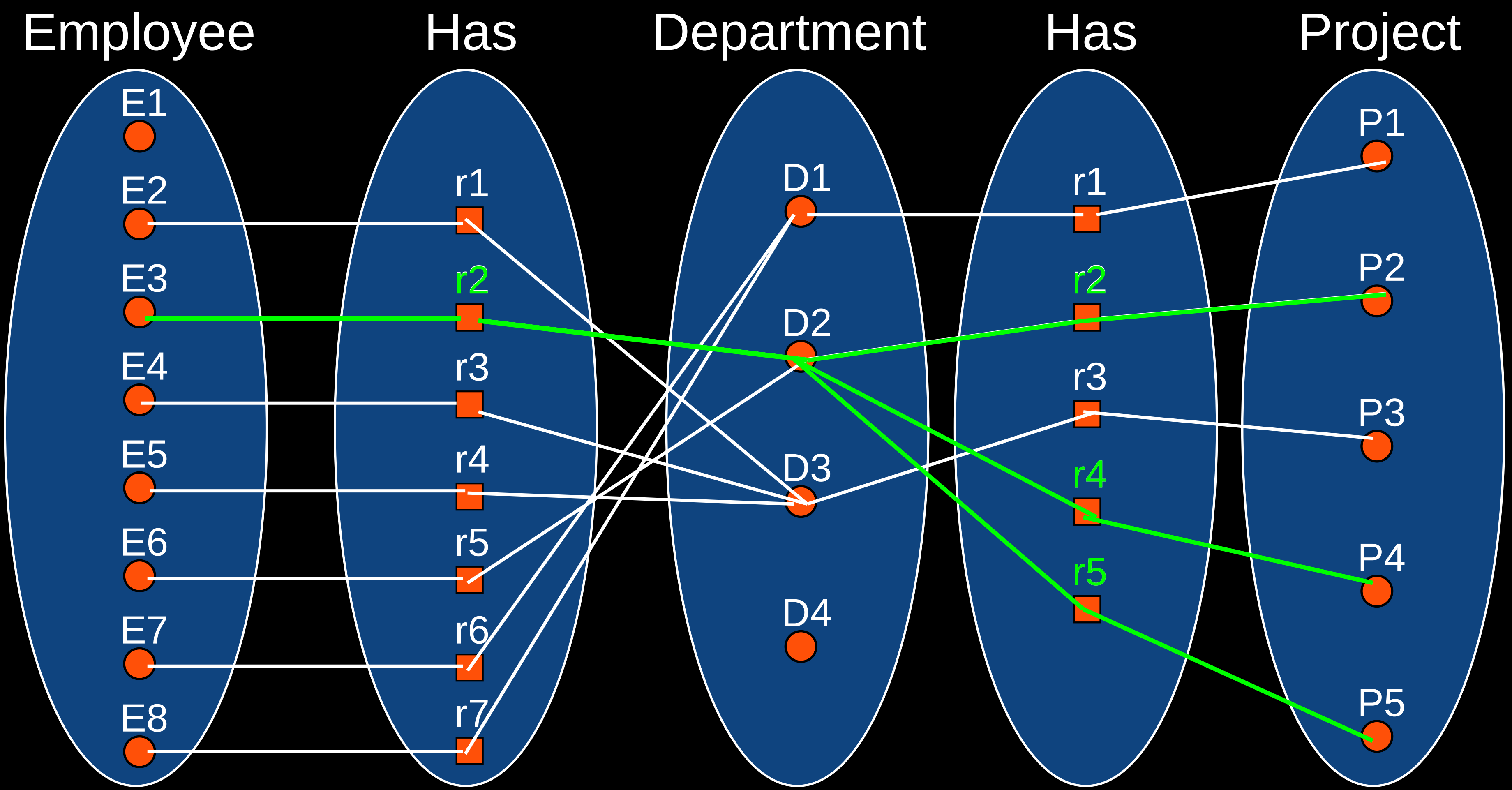
- Often occurs when two or more one-to-many relationships fan out (come from) the same entity type.

Example: A department has multiple employees, a department has multiple projects, and each project has multiple employees.



Now answer the question, which projects does employee E3 work on?

Fan Trap Example



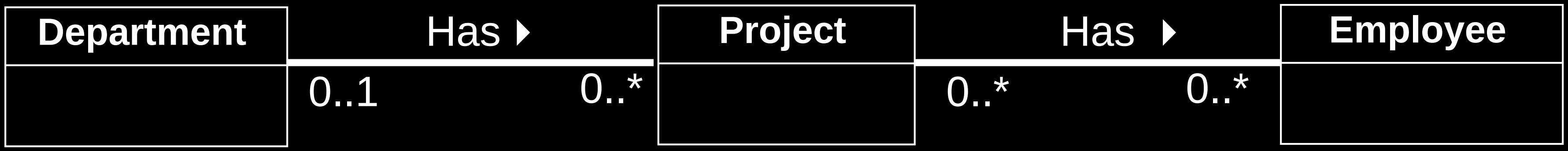
Which projects does employee E3 work on?

Chasm Traps

A ***chasm trap*** occurs when a model suggests that a relationship between entity types should be present, but the relationship does not actually exist. (*missing* relationship)

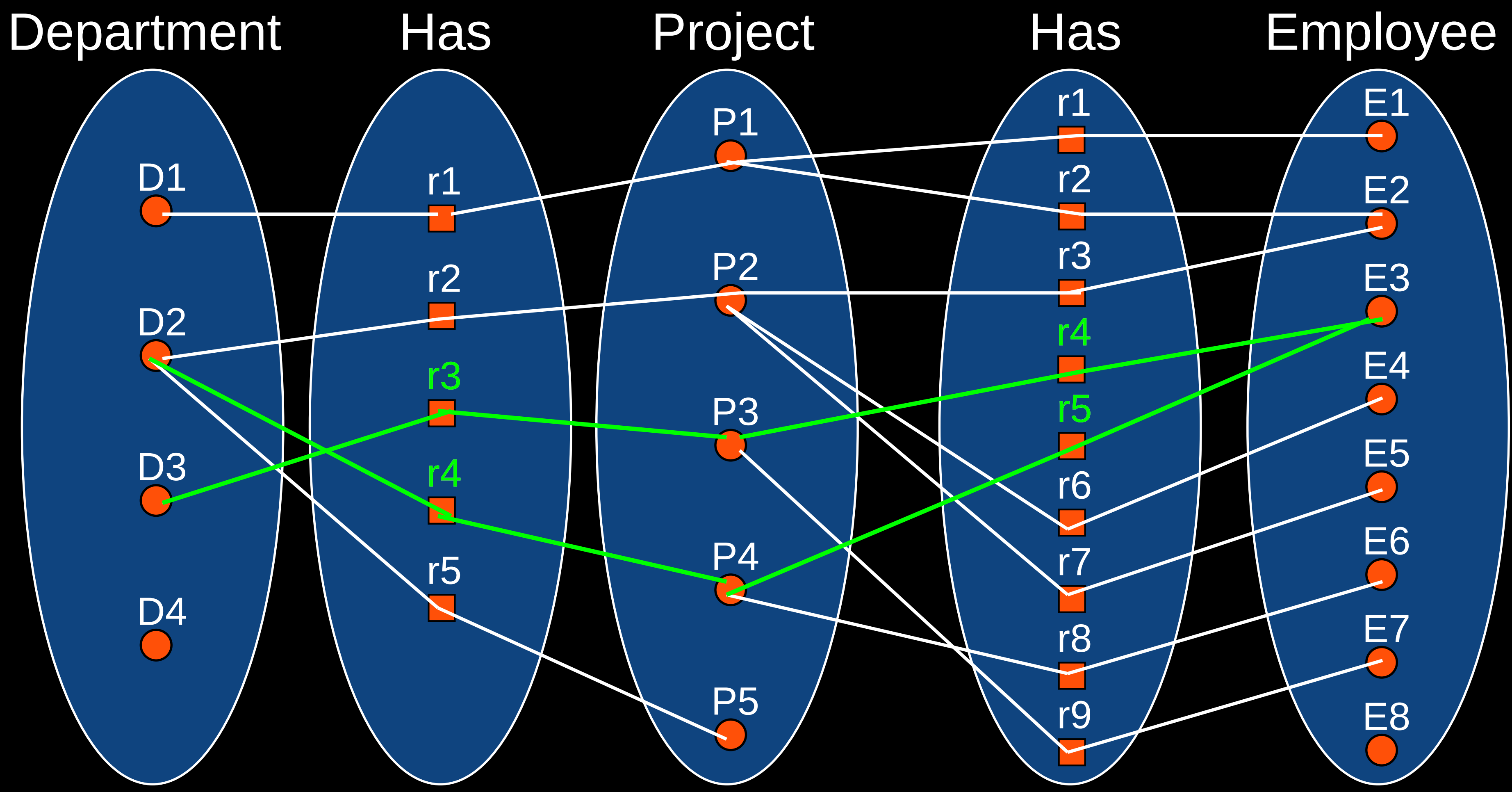
- May occur when there is a path of optional relationships between entities.

Example: A department has multiple employees, a department has multiple projects, and each project has multiple employees.



Now answer the question, what department is employee E3 in?

Chasm Trap Example



Which department is employee E3 in?

What are the employees of department D2?

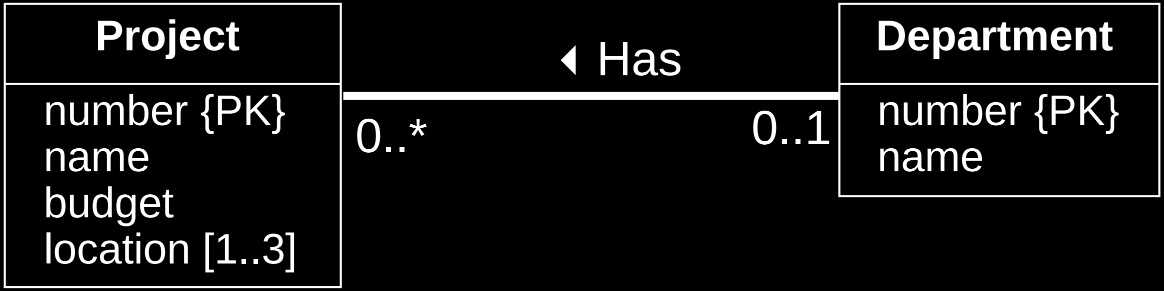
Good Design Practices

When designing ER models, there are several things that you should consider:

- 1) Avoid redundancy - do not store the same fact more than once in the model.
- 2) Do not use an entity when you can use an attribute instead.
- 3) Limit the use of weak entity sets.

Good Design Practices Avoiding Redundancy Example

Good:



Project

number {PK}
name
budget
location [1..3]

◀ Has

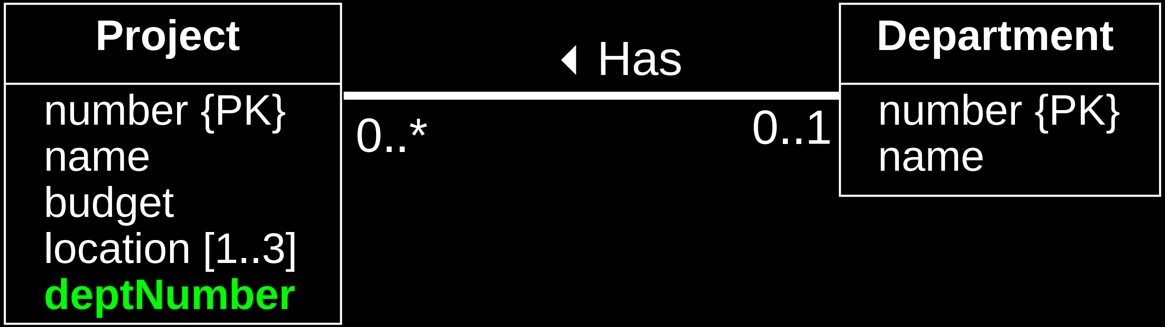
0..*

0..1

Department

number {PK}
name

Wrong:



Project

number {PK}
name
budget
location [1..3]
deptNumber

◀ Has

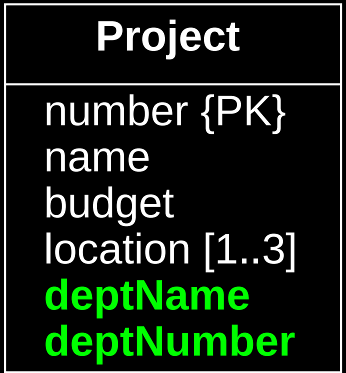
0..*

0..1

Department

number {PK}
name

Bad:



Project

number {PK}
name
budget
location [1..3]
deptName
deptNumber

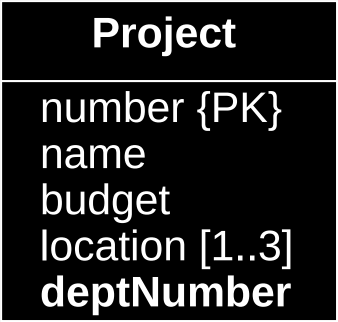
Good Design Practices Entity versus Attribute Example

An entity should only be used if one of these conditions is true:

- 1) The entity set should contain at least one non-key attribute.
- 2) It is the many side in a many-to-one relationship.

Example: Projects have a department. A department only has a number.

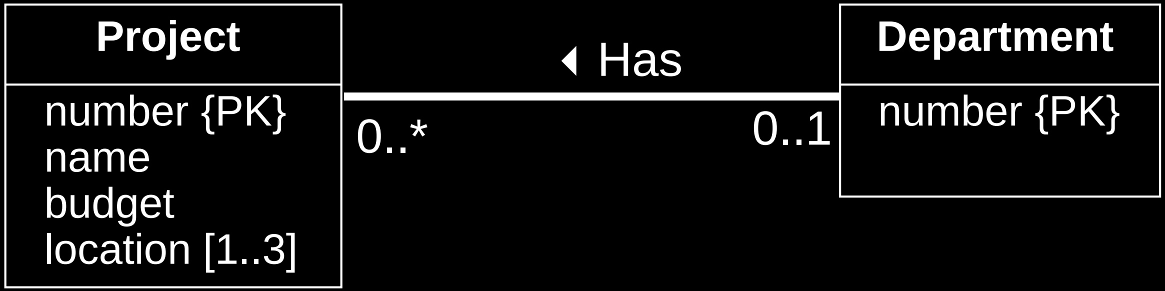
Acceptable:



Project

number {PK}
name
budget
location [1..3]
deptNumber

Most general:



Project

number {PK}
name
budget
location [1..3]

◀ Has

0..*

0..1

Department

number {PK}

Good Design Practices
Weak Entity Sets

Avoid the use of weak entity sets in modeling. Although at first glance there are very few natural keys based on the properties of objects (name, age, etc.), there are many good human-made keys that can be used (SSN, student#, etc.)

Whenever possible use a global, human-made key instead of a weak entity. Note that sometimes a weak entity is required if no such global key exists.

- For example, a database storing the students of multiple universities could not use a single student# as a key as it is unlikely that universities could agree on a global numbering system. Rather, student becomes a weak entity with partial key student# and identifying entity the university the student attends.

Simplifications

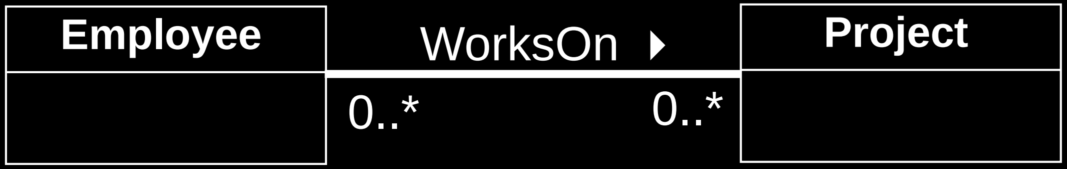
Two common relationship simplifications:

- Many-to-many relationship ⇨ Two one-to-many relationships
- Higher order relationships ⇨ binary relationships

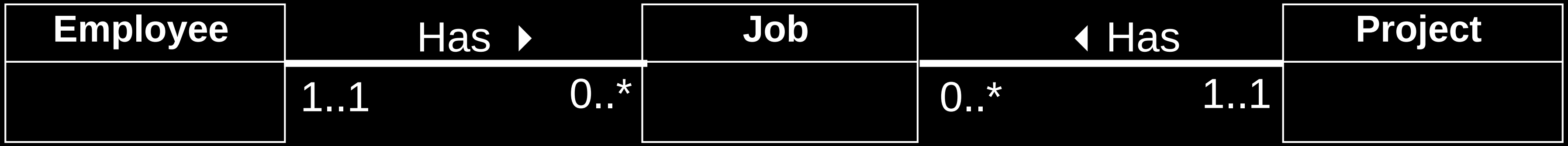
Many-to-Many Relationship Simplification

A many-to-many relationship can be converted into one entity with two 1:N relationships between the new entity and the original entities participating in the relationship.

Original:

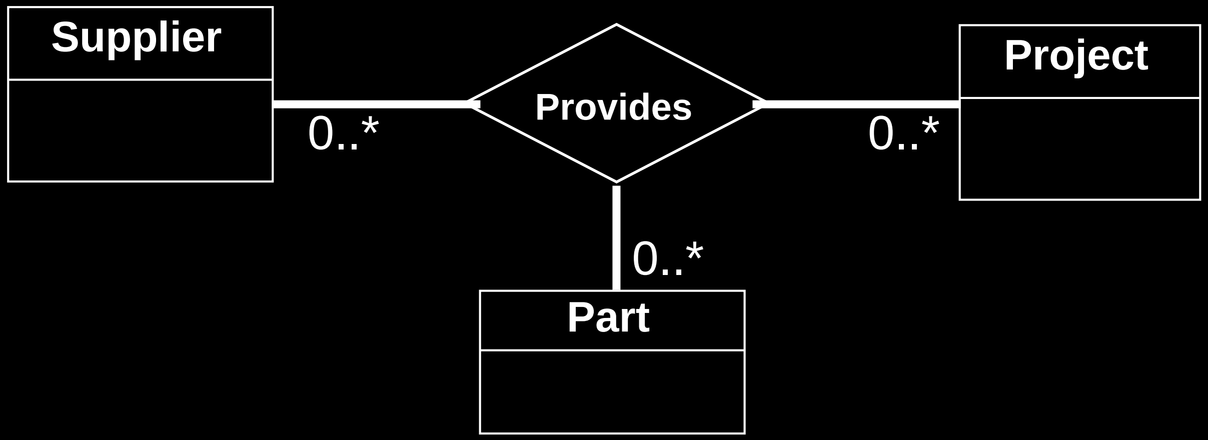


Simplified:

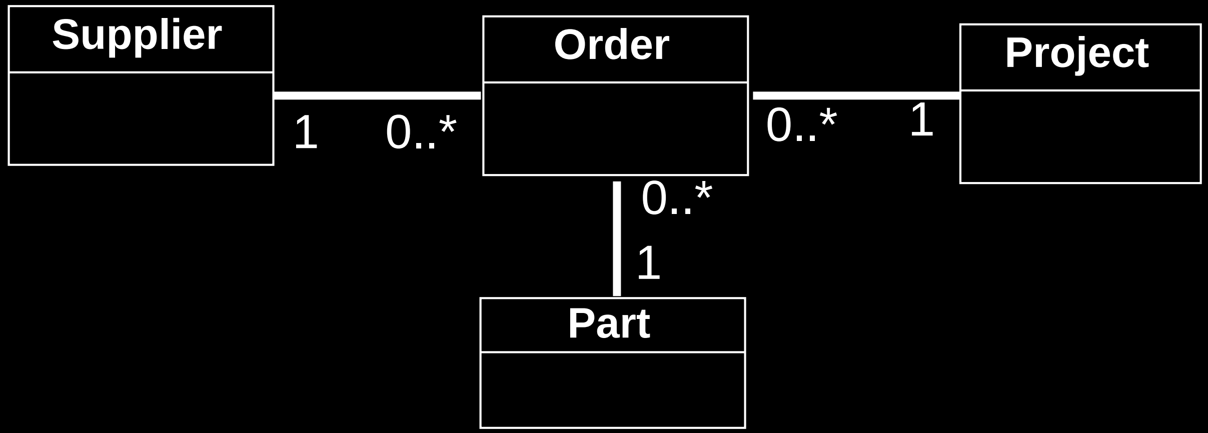


Higher Degree Relationships Simplified using a Weak Entity

Original:



With weak entity:

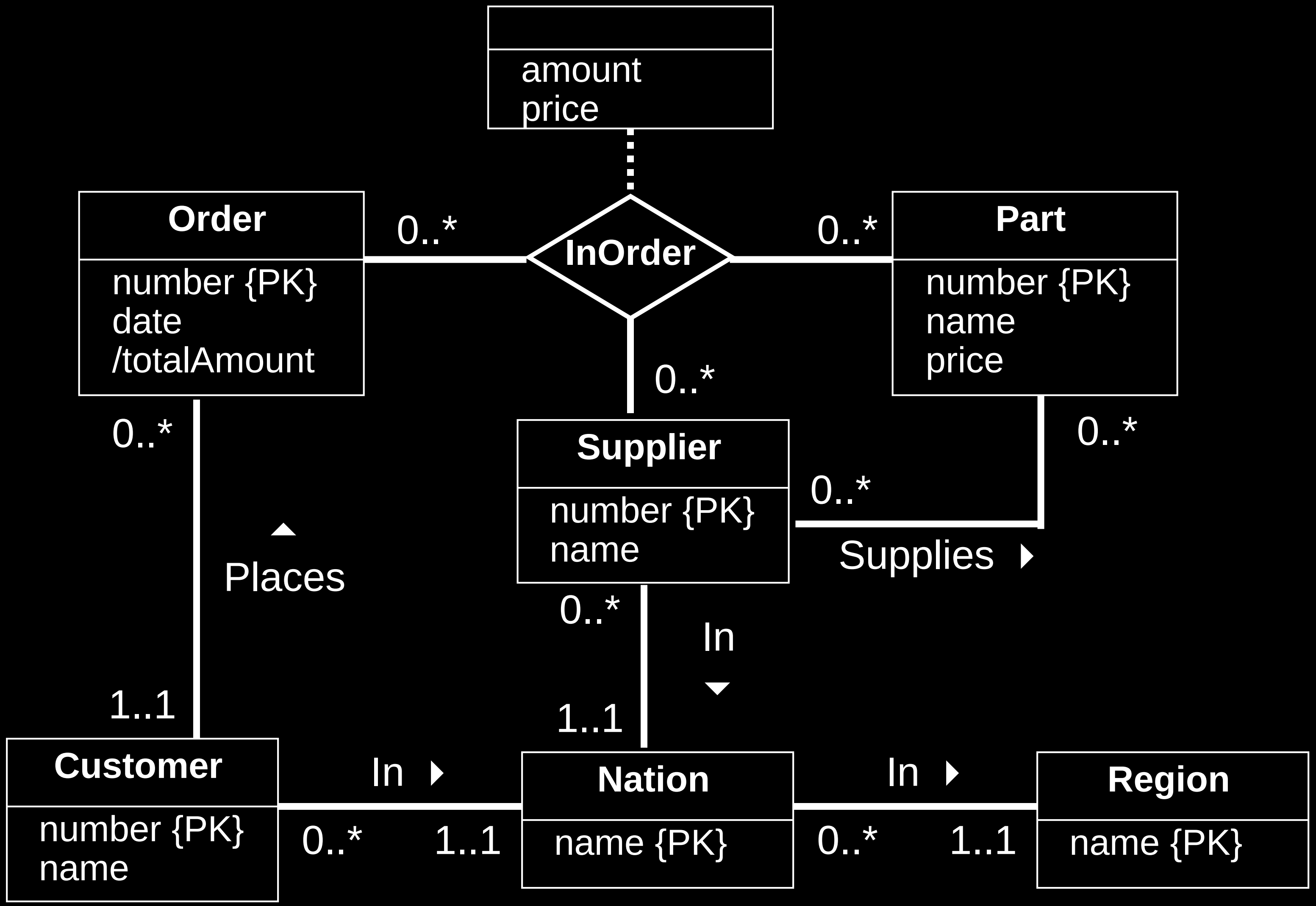


ER Design Example TPC-H Standard Schema

The database will store order information:

- Each order has a numeric key, a customer, a date, a total order amount, and a list of parts.
- A part in an order has an amount and a price paid and is supplied by a certain supplier.
- Each part has a numeric key, a name, and a price and may be supplied by multiple suppliers.
- A supplier has a key and a name and may supply multiple parts.
- A customer has a key and a name.
- Each supplier and customer is located in a nation.
- Each nation is located in a region (continent).

ER Design Example
TPC-H Standard Schema Answer



ER Design Question #2

Construct a fish store database where:

- A fish store maintains a number of aquaria (tanks), each with a number, name, volume and color.
- Each tank contains a number of fish, each with an id, name, color, and weight.
- Each fish is of a particular species, which has a id, name, and preferred food.
- Each individual fish has a number of events in its life, involving a date and a note relating to the event.

ER Design Question #3

Construct an invoice database where:

- An invoice is written by a sales representative for a single customer and has a unique ID. An invoice has a date and total amount and is comprised of multiple detail lines, containing a product, price and quantity.
- Each sales representative has a name and can write many invoices, but any invoice is written by a single representative.
- Each customer has a unique id, name, and address and can request many invoices.
- Products have descriptions and weights and are supplied by vendors. Each product has a unique name for a particular vendor. A product is supplied by only one vendor.
- A vendor has an id and an address.

Conclusion

Conceptual design is performed at a high-level of abstraction involving entities, relationships, and attributes.

- An entity type is a group of entities with the same properties.
  - Entities may be strong (have unique key) or weak (no unique key).
- A relationship type is an association between entities.
  - A relationship may involve two or more entities and may be recursive.
  - A relationship has two types of constraints:
    - Participation - minimum # of times an entity must be involved in relationship
    - Cardinality - maximum # of times an entity can be involved in relationship
  - Common relationship multiplicities are: 1:1, 1:*, *:*.
- Attributes are properties of entities or relationships.

The ER model using UML syntax is used for database design.

It is possible to fall into design traps, so practice is necessary to become a good conceptual designer.

Objectives

- Explain the relationship between conceptual design and ER modeling/UML.
- Be able to define: entity type, relationship type, degree of a relationship, recursive relationship, attribute, attribute domain, simple attribute, composite attribute, single-valued attribute, multi-valued attribute, derived attribute
- Be able to define: candidate key, primary key, composite key
- Be able to explain the difference between a strong entity type and a weak entity type.
- Be able to explain multiplicity and participation and how they are used in modeling.
- Be able to describe fan traps and chasm traps.

Be able to model a domain explained in an English paragraph in an ER diagram using UML notation.