E/R Modelling

Databases

Ira Assent

ira@cs.au.dk

Data-Intensive Systems group, Department of Computer Science, Aarhus University, DK

Intended learning outcomes

- Be able to
  - build a database design using the ER model
  - Follow design principles
  - Map ER model to relational schema

Recap: DBMS makes our data handling easier!

- DBMS takes care of all data handling
  - No need to handle in program / applications
  - Users are blissfully unaware it exists!
  - Flexible
    - Storage structures and efficient search techniques
    - Backup and recovery
    - Multiple user access
    - Controlling redundancy
    - Restricting unauthorized access
    - Representing complex data relationships
    - Business rules / semantic rules

- Relational Model:
  - simple data structure: the table
  - Effective (captures many situations)
  - leads to useful yet not too complex query languages

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

ira@cs.au.dk

Make a database

So, your boss says

"The meeting went great, we get to build the system. We need to manage all data for this chain of shops, and the web shop, of course. They sell wanyatas, which is so hyped right now. They have so many different varieties, and they keeping adding producers as we are speaking. I hear they even might open shops abroad next week already. Come to my office in 5 minutes and show me what we will build for them."

ira@cs.au.dk

How to get started?

- So, how do we know which tables we need?
  - Which columns and rows?
    - Design of databases
- Structured process to create well-formed databases
  - Why?
    - A poor database can be
      - Slow
      - Redundant
      - Contain faulty or inconsistent data
      - Miss relevant data
      - …
  - Time spent on design is time spent well
    - Saves a lot of hassle in fixing issues in a (more or less) working system!

ira@cs.au.dk

Entity/Relationship Models

- People want a database, but do not know what they want in it
- Sketching the key components is an efficient way to develop a working database
  - Includes some constraints, but not operations
  - Designs are pictures called *entity-relationship diagrams*
- **Entity/relationship model** de-facto standard for most relational databases
  - Provides graphical notation in **ER diagrams**
    - Enable high-level design of databases
    - Can be used to discuss with customers / stakeholders
    - Ease of use
    - Serves as documentation
    - Different models exist; we cover one notation
- Rules of thumb for good designs
  - More formal analysis in normalization later
- Algorithmic conversion into relational schemas

ira@cs.au.dk

What is the purpose of the database?

- We start from the **miniworld**
  - Which describes the aspect of the real world for which to store data
  - E.g. we want a database to maintain information about a company
    - We need to represent all relevant information about the company
      - "relevant" depends on what the user/programs will need
    - We should represent it correctly
      - If we make assumptions that are invalid, such that employees only belong to a single department, this can be an issue later
    - We should represent properties and relationships
      - If every project needs a manager that needs to be ensured
    - We need to identify what the database is to be used for
      - E.g. mostly queries relating to the company's organization, its employees, its projects

ira@cs.au.dk

Entity / relationship example

- Identify
  - things / objects called **entities**
    - *departments*, *projects*, *employees*, etc
  - descriptive information / properties called **attributes**
    - e.g. projects may have a project *duration* or project *name*
  - And **relationships** between entities
    - E.g. how projects *belong to* departments or employees
- Hence the name entity / relationship model
- Visual representation / notation in entity / relationship diagram

ira@cs.au.dk

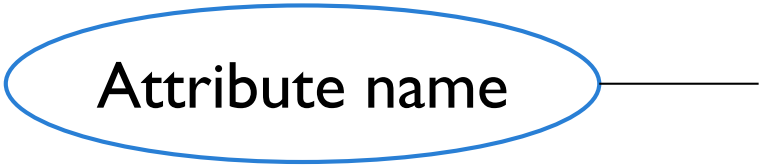
Entity sets and entities

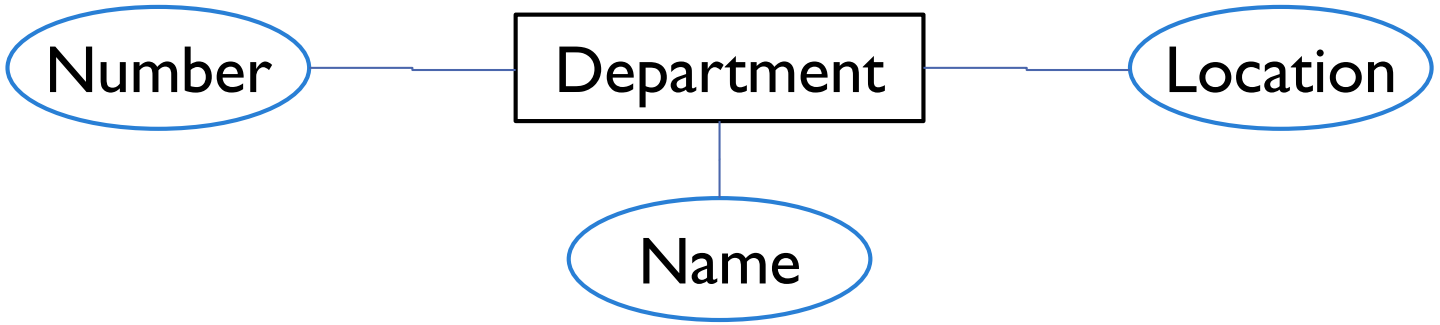
- Things or objects
- Graphical notation: rectangle
- **Entity Set** is a collection of entities with same attributes
  - E.g. Department with number, name, location
- **Entity** is one such instance
  - E.g. the department with number 5, name IT, location Aarhus
  - Another entity of the same entity set is number 7, name Bookkeeping, location Herning

ira@cs.au.dk

Attributes

- Properties / features of entities
  - Simple values
  - E.g. the name of a department
  - Graphical notation: oval, line to corresponding entity set

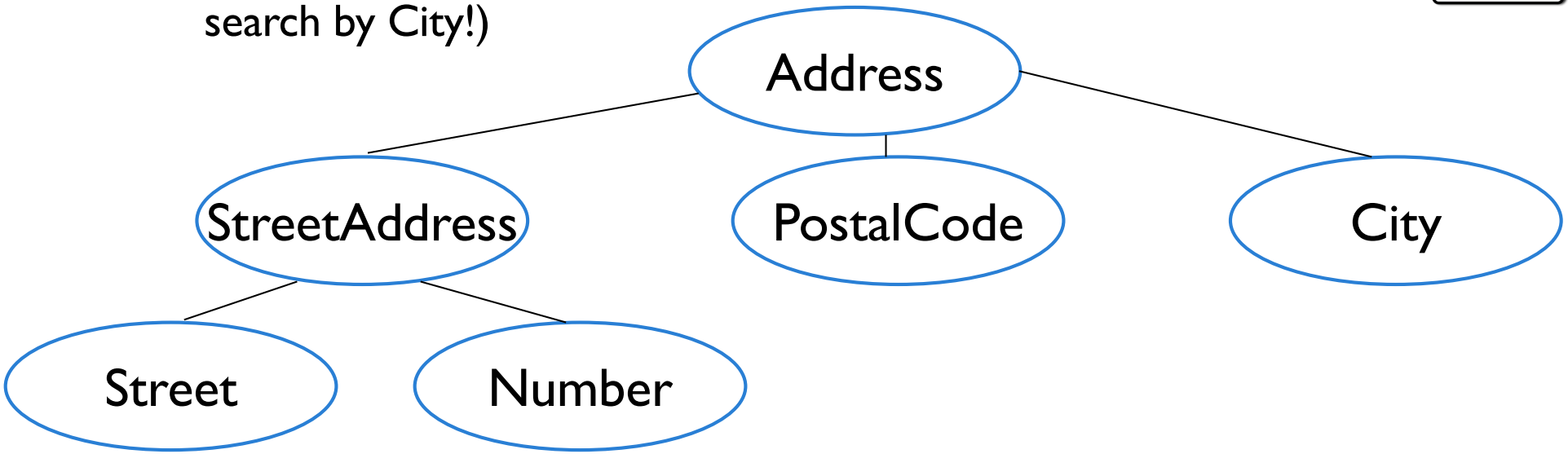




ira@cs.au.dk

Composite Attributes

- Complex attributes can be composed of simpler attributes
  - E.g. Address composed of StreetAddress, PostalCode, City
  - May still be treated as one attribute if not accessed individually (e.g. search by City!)

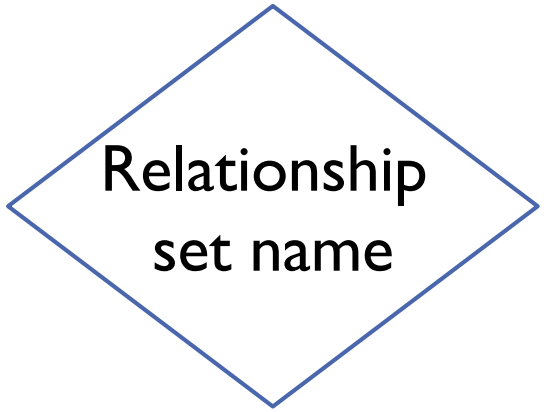


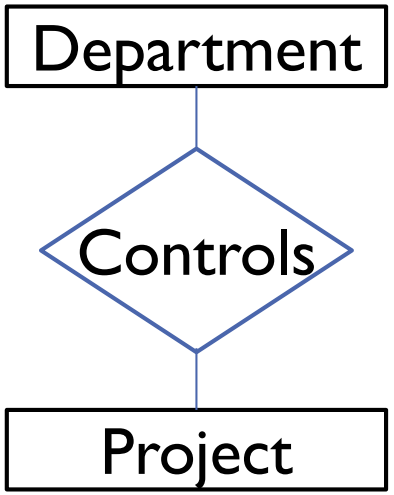
- For the purpose of this course, we work with simple attributes
- We show composite attributes here to avoid confusion arising from examples (and inconsistent notation) in the textbook

ira@cs.au.dk

Relationship sets and relations

- How entities are connected / relate
- Graphical notation: diamond with lines to corresponding entity sets
- Relationship Set is a collection of relations between entities in these entity sets
  - E.g. Controls between Departments and Projects
- Relationship is one such connection
  - E.g. department 5 controls project 1
- The value of a relationship is a set of tuples with one component for each related entity set

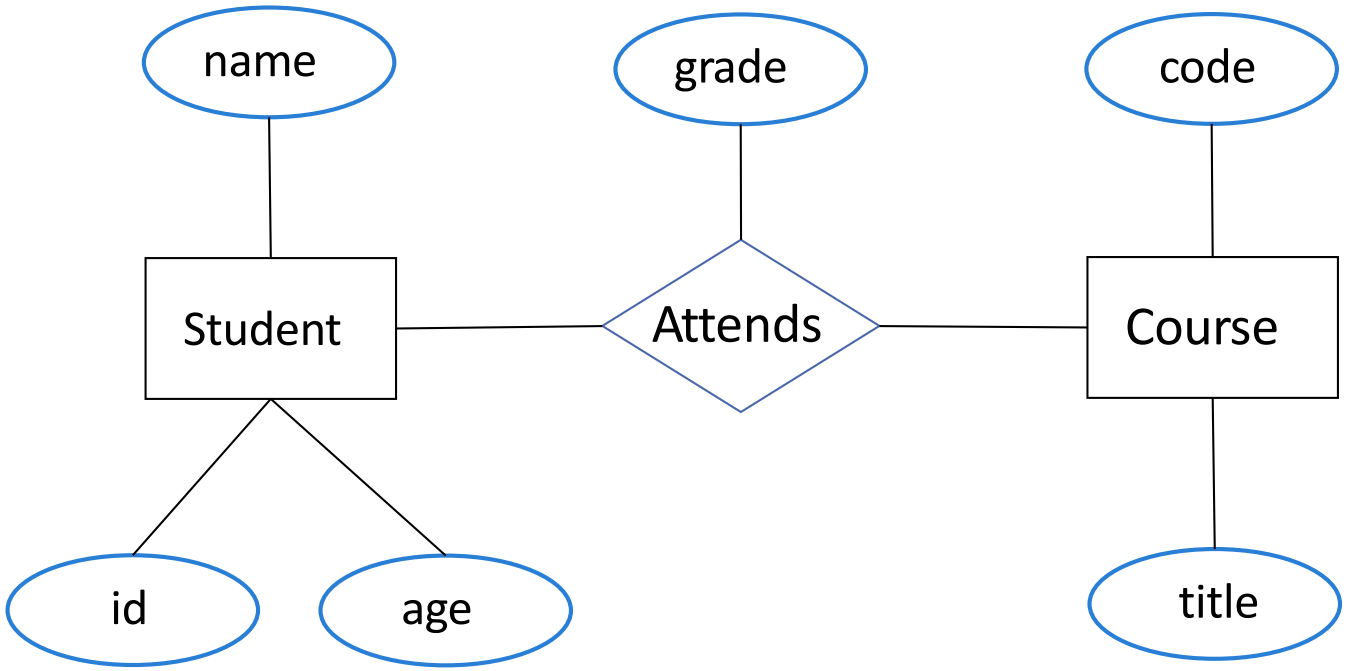




ira@cs.au.dk

Attributes on Relationships

- Relationships may have their own attributes

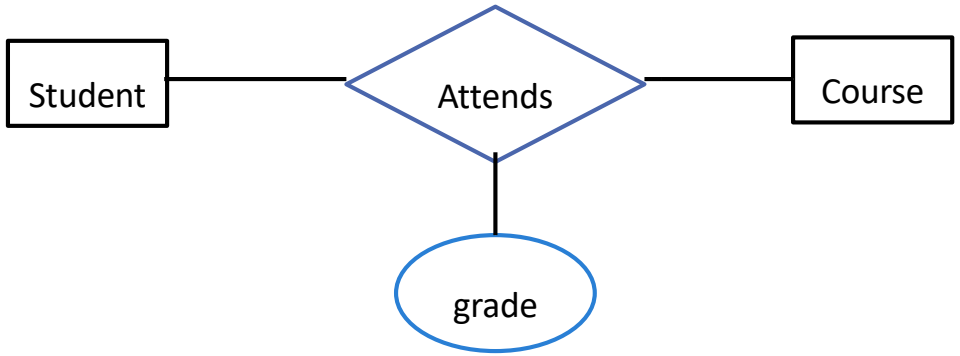


ira@cs.au.dk

Attributes on Relationships – doing without

- Where should grade go if we want to do without attributes on relationships?

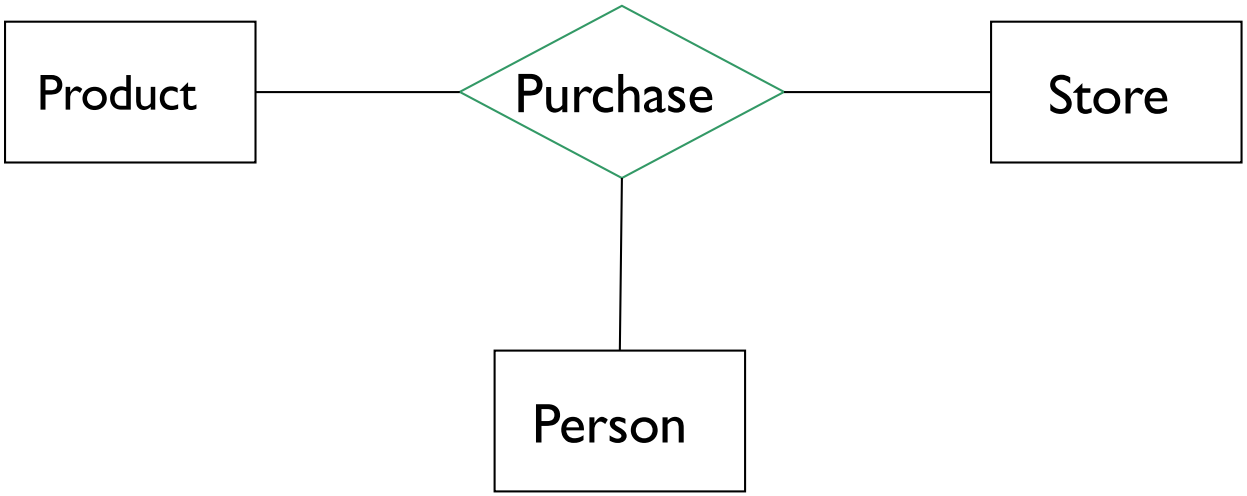
1. Student
2. Course
3. Both Student and Course
4. A new entity



ira@cs.au.dk

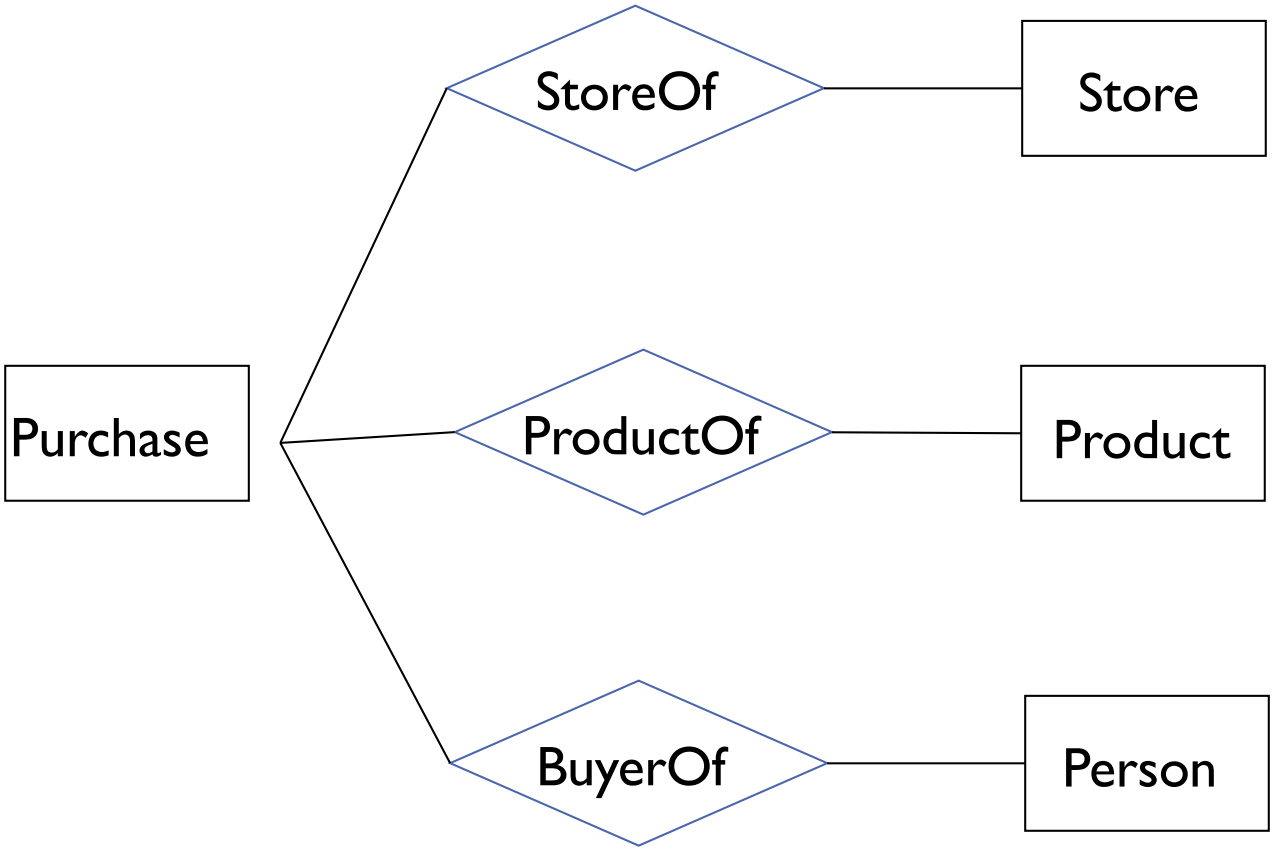
Multiway Relationships

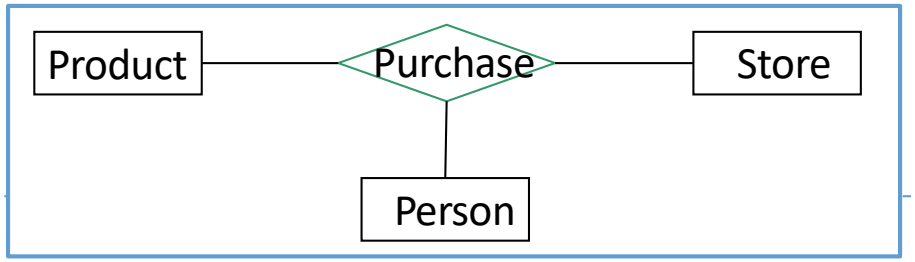
- Relationships are generally (but rarely) *n*-ary
  - So, can connect *n* entity sets
  - Example for tertiary relationship set



ira@cs.au.dk

Reducing to Binary Relationships



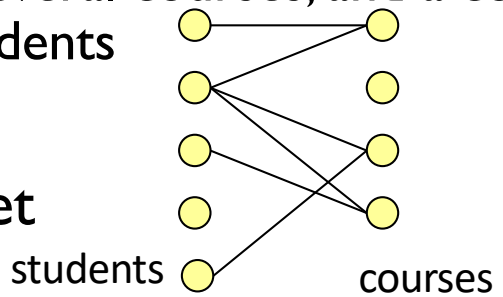


ira@cs.au.dk

Cardinality Ratio in relationship sets

- Very important design consideration
  - How many entities *can be* part of a relationship set in *one relation*?
  - Has decisive impact on tables in relational database
  - Also called multiplicity
- **Many-to-many** (denoted **M:N**)
  - Each entity is connected to zero or many entities

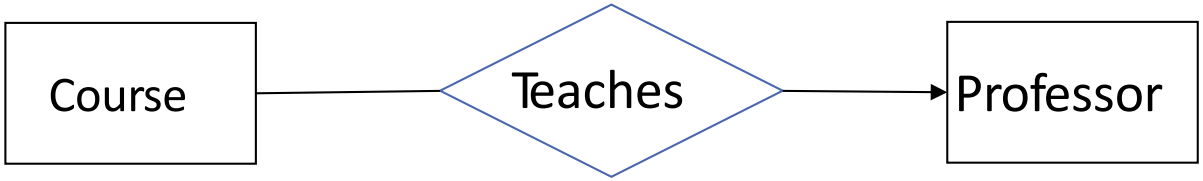
    - E.g. a student may attend no, a single, or several courses, and a course is attended by zero, a single or several students
  - Indicated by ordinary lines
  - Possible instances in the relationship set



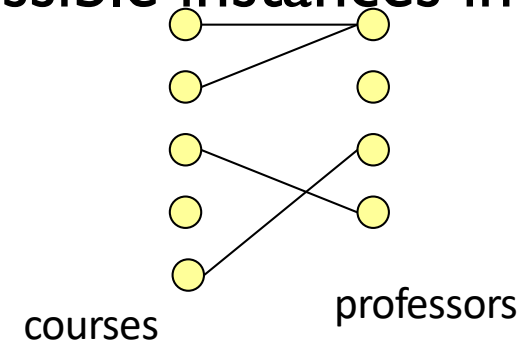
ira@cs.au.dk

Many-One Relationships

- The first entity determines the second entity
  - Denoted as **N:1** (or 1:N the other way around)



  - E.g. a course is taught by one professor, so once we know the course, we also know the professor ("determined")
- Indicated by an arrow line to the "One" part
- Possible instances in the relationship set

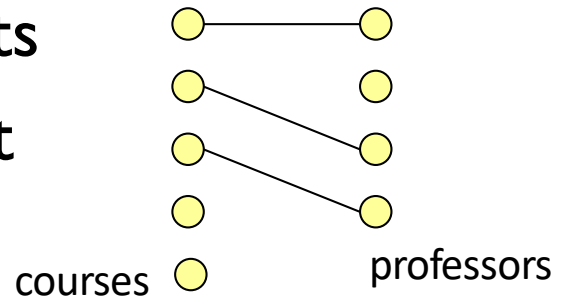


ira@cs.au.dk

One-One Relationships

- Each entity in the relationship determines the other
  - Denoted as **1:1**

- E.g. each course is the favorite of one professor, but no two professors have the same favorite course, and no professor has two favorite courses
- Indicated by arrow lines to both parts
- Possible tuples in the relationship set

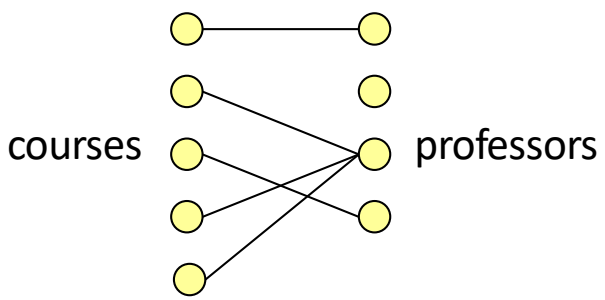


ira@cs.au.dk

Total participation

- Sometimes the connection is to *exactly* one entity
  - Called total participation or existence dependency
  - Essentially means a minimum cardinality
  - Otherwise, partial participation
  - Both for many-one and one-one (N:1 or 1:1)

- Each course is taught by a professor, i.e., there is no course without a professor (minimum one professor)
  - Indicated by a double line between that entity and the relationship
- Possible tuples in the relationship set



ira@cs.au.dk

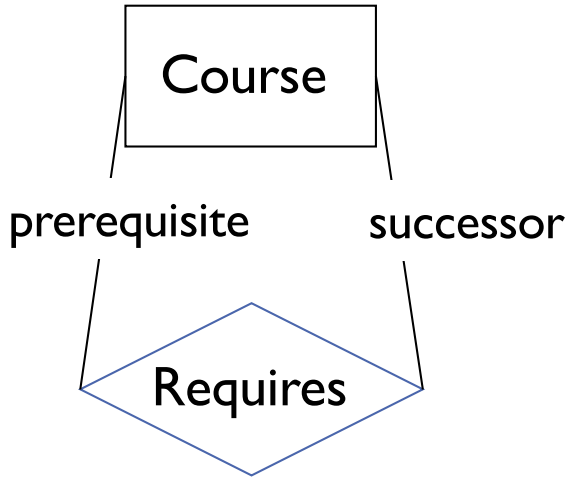
What is the cardinality ratio?

1. Many-many
2. Many-one
3. One-many
4. Undefined

ira@cs.au.dk

Roles

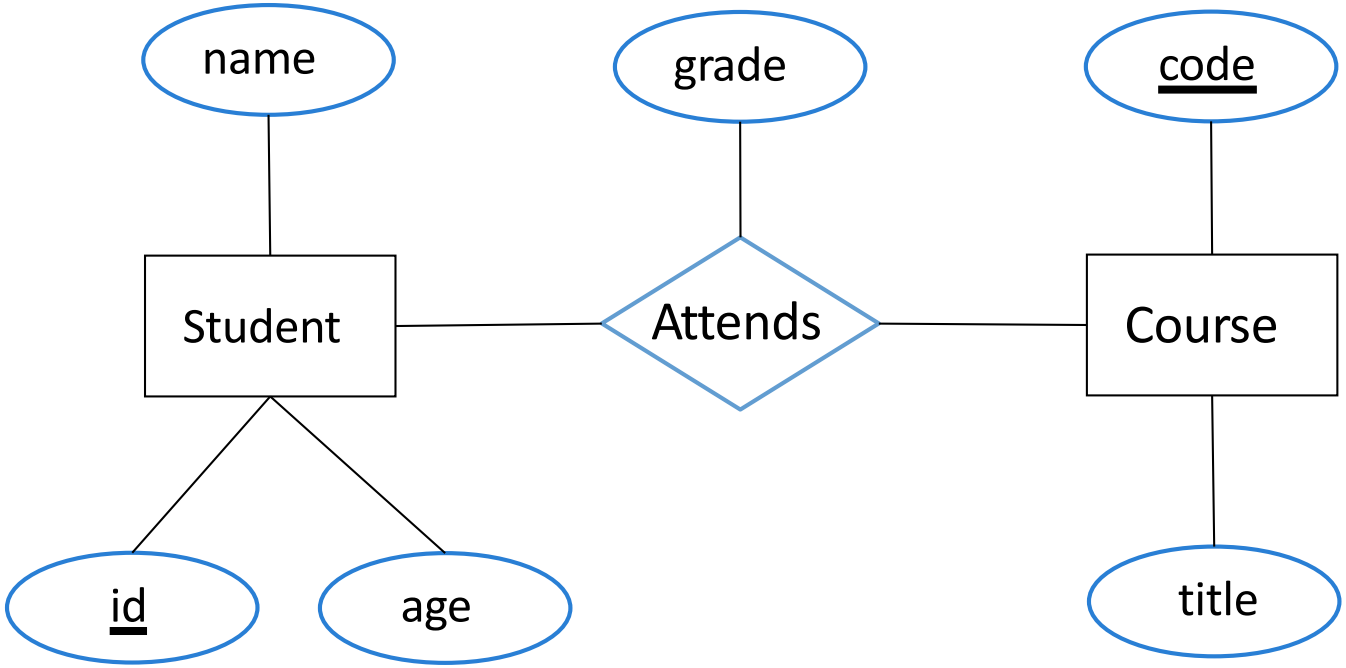
- An entity set may be in several relationships
  - Called **recursive** or **self-referencing**
  - E.g. Requires is a relationship between two courses
- For clarity, label the edges with *role* names
  - helpful for reading / understanding which of the two courses has which role in the relationship
  - Used in mapping to schema
  - E.g. for relationship Requires, one course may be the prerequisite for another
    - Labeled prerequisite
    - Other course then labeled successor
    - E.g. Databases has prerequisite Programming



ira@cs.au.dk

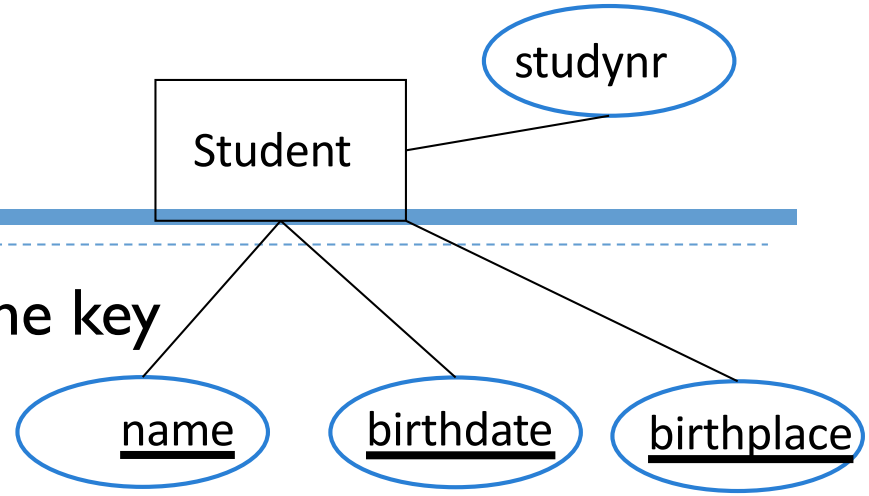
Keys

- A set of attributes that identifies entities
- Every entity set *must have a key*
- Indicated by underlining the attribute names
  - E.g. courses are uniquely identified by a course code



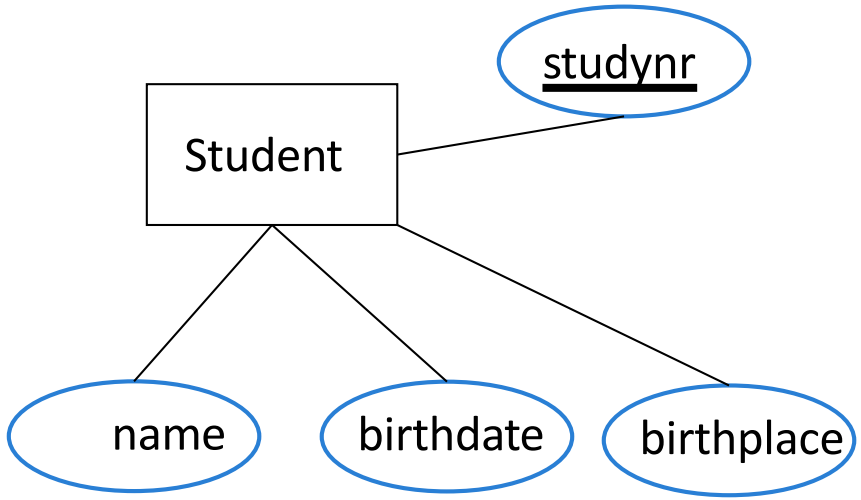
ira@cs.au.dk

More Keys



- There may be several options for the key
- E.g. a student is identified by
  - Studynr
  - Name, birthdate, birthplace
- Prefer
  - Fewer
  - Existing (= not artificial)

  attributes

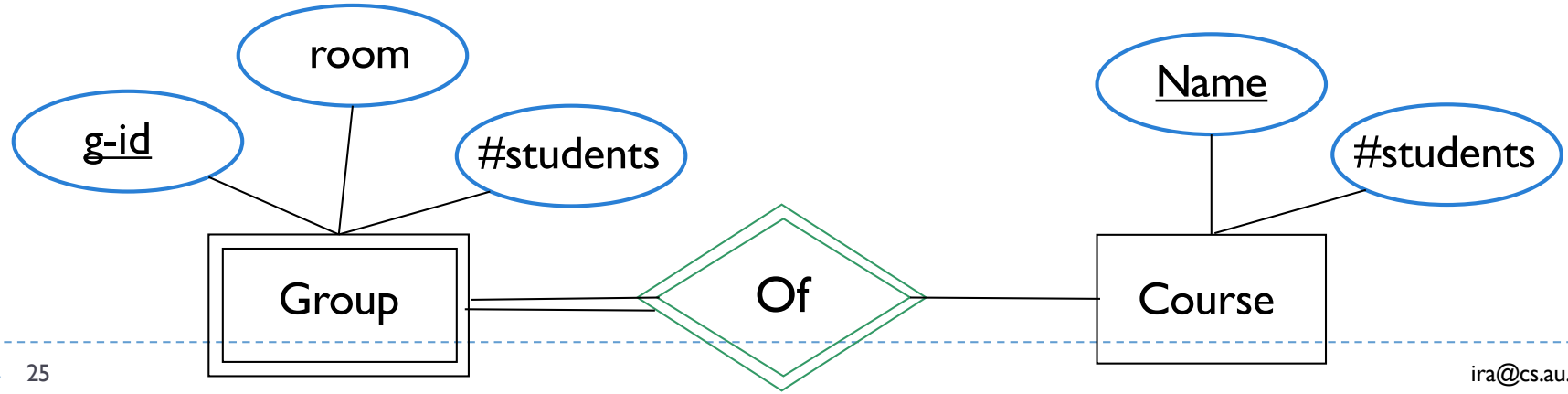


- All attributes of the chosen key are underlined, and are all part of the key for the entity set
  - the textbook is inconsistent here, unfortunately, and states that a key of two or more attributes should be a composite attribute instead

ira@cs.au.dk

Weak Entity Sets

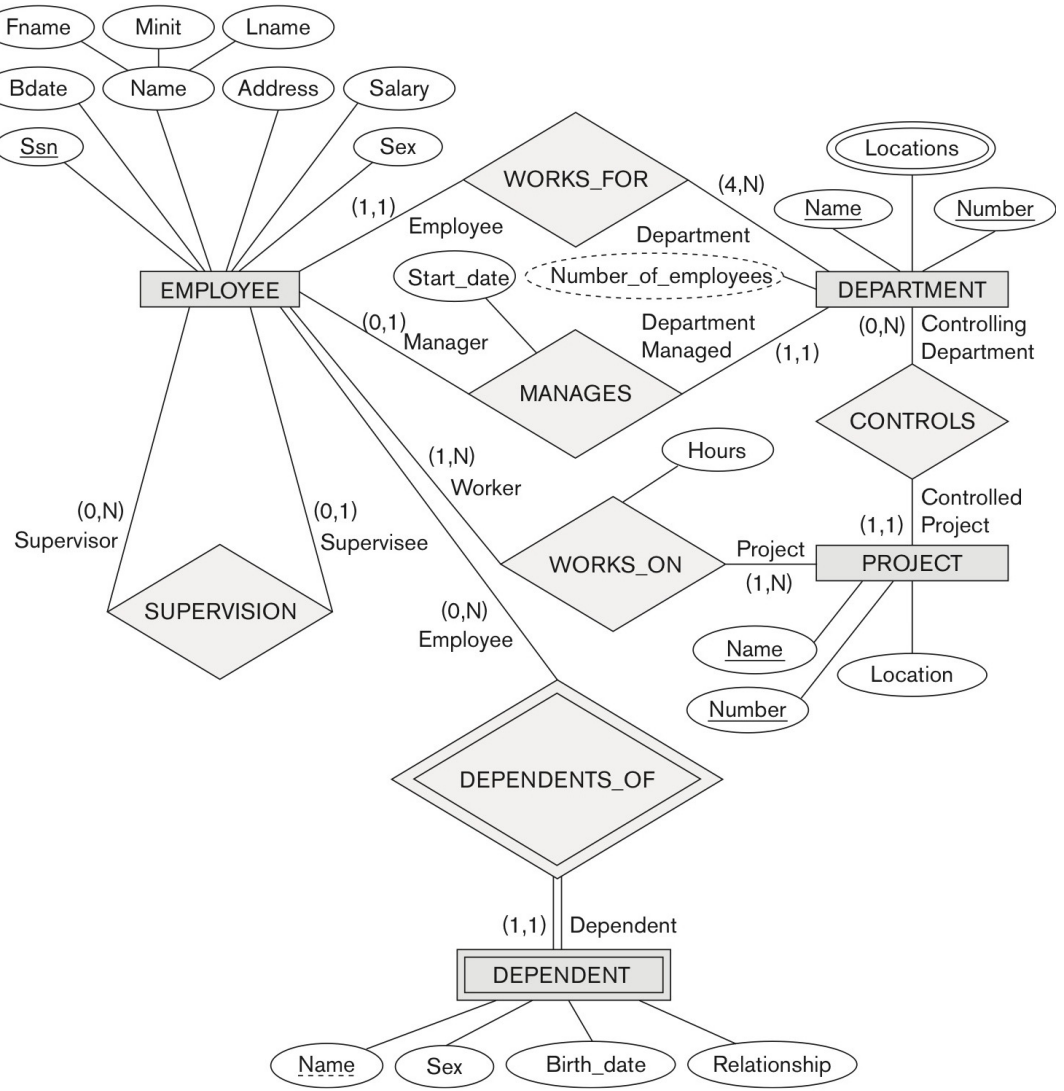
- If an entity set has no key, it is a **weak entity set**, and it must get its key partially from another **(supporting) entity set** through a many-to-exactly-one relationship
- Notation: Double rectangle, double diamond
  - A group does not have a unique group-id (DA1 may be a group in Databases and in Programming)
    - If we know the supporting entity (e.g. Databases), then it is unique (only one DA1 in our course)
    - Please note that this implies total participation of group in relationship to course also



ira@cs.au.dk

Alternative Notations for cardinalities

- Structural constraints on relationships
  - Replace cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints
  - Note (min, max) for each participation of an entity set in a relationship set
    - where $0 \leq \min \leq \max$ and $\max \geq 1$
    - E.g. a department may may be controlling zero or several projects, where each project is being controlled by one department at any point in time
    - E.g. an employee may be a supervisor in none or many supervision relationships, an employee may be supervised by one or no supervisors at any point in time
- In this course, adopt M:N notation for cardinality, but some exercises on (min, max)



Which of the following statements about keys is **not** true?

1. A key is a constraint on the possible attribute values.
2. There must be exactly one key for every entity set.
3. A key is always a set of attributes.
4. A partial key is always a single attribute.

ira@cs.au.dk

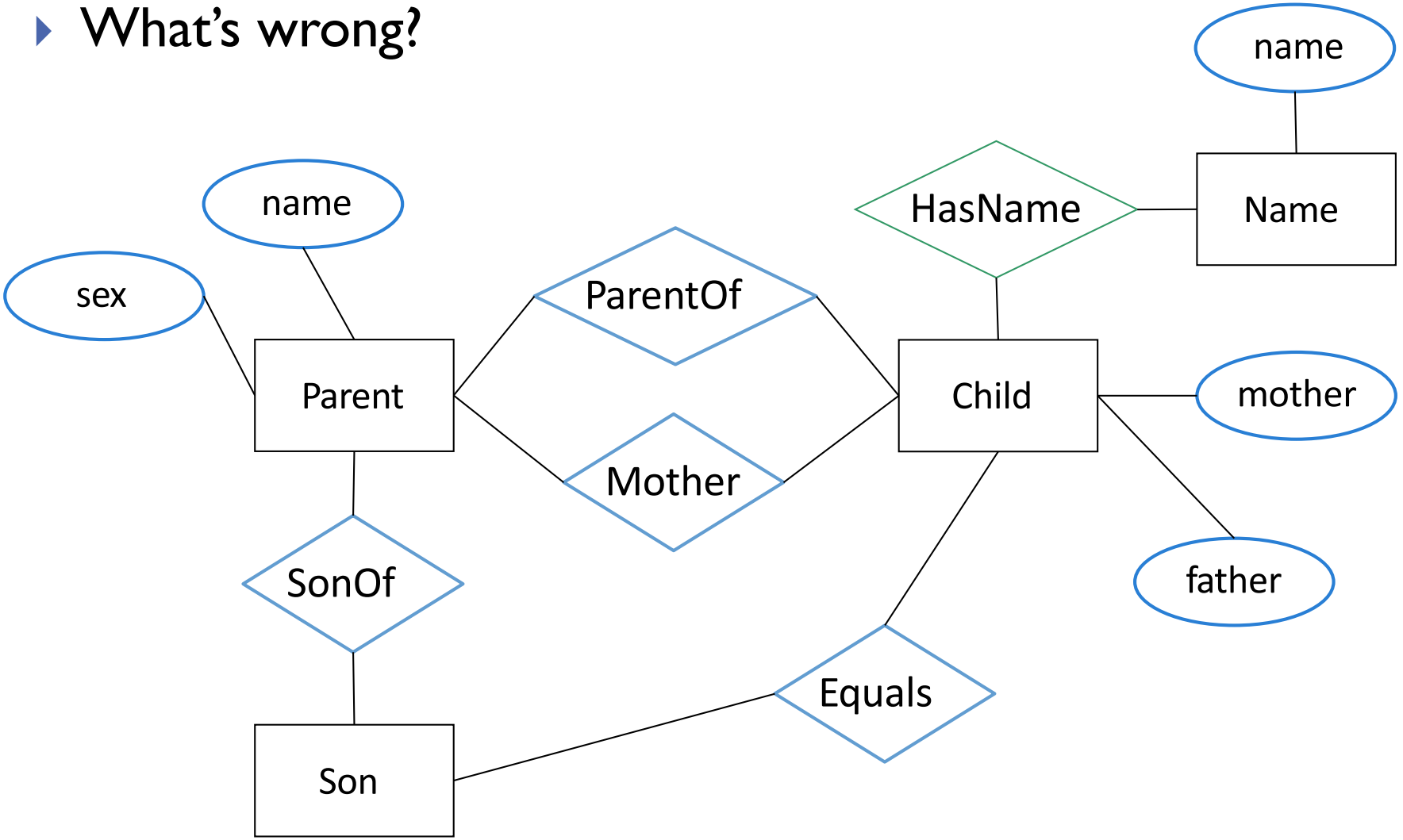
Some Design Principles

- Avoid redundancy
  - Do not introduce the same entity twice
  - Do not add relationships that can be inferred
- Don't use a key from another entity as attribute
  - Use another relationship instead
- Use attributes instead of entity sets unless:
  - It has at least one non-key attribute
  - It is the many-part in a many-many or many-one relationship

ira@cs.au.dk

A Terrible Design

- What's wrong?



ira@cs.au.dk

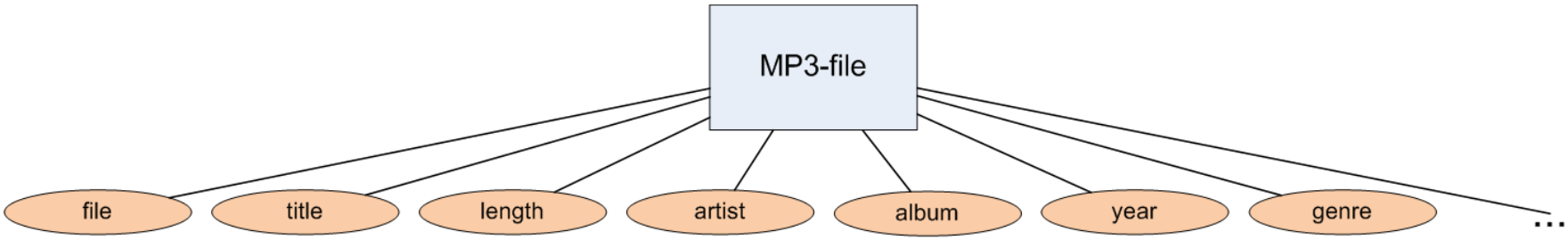
Level of Details

- A model can be made ever more detailed
- Do not try to make the most detailed model of the phenomenon in question
- Think about which queries you want to answer

ira@cs.au.dk

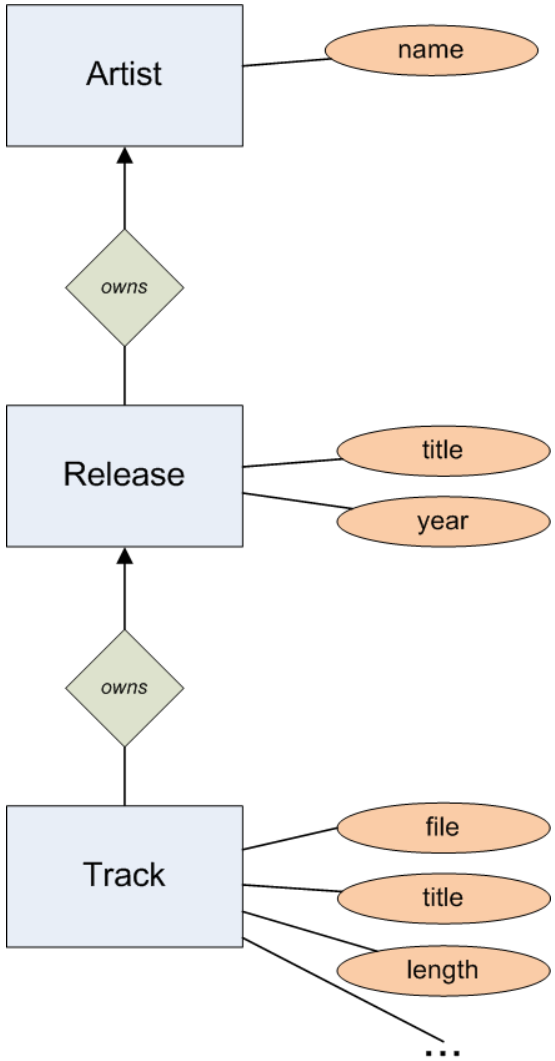
A Naive Model

- Like iTunes and MediaPlayer



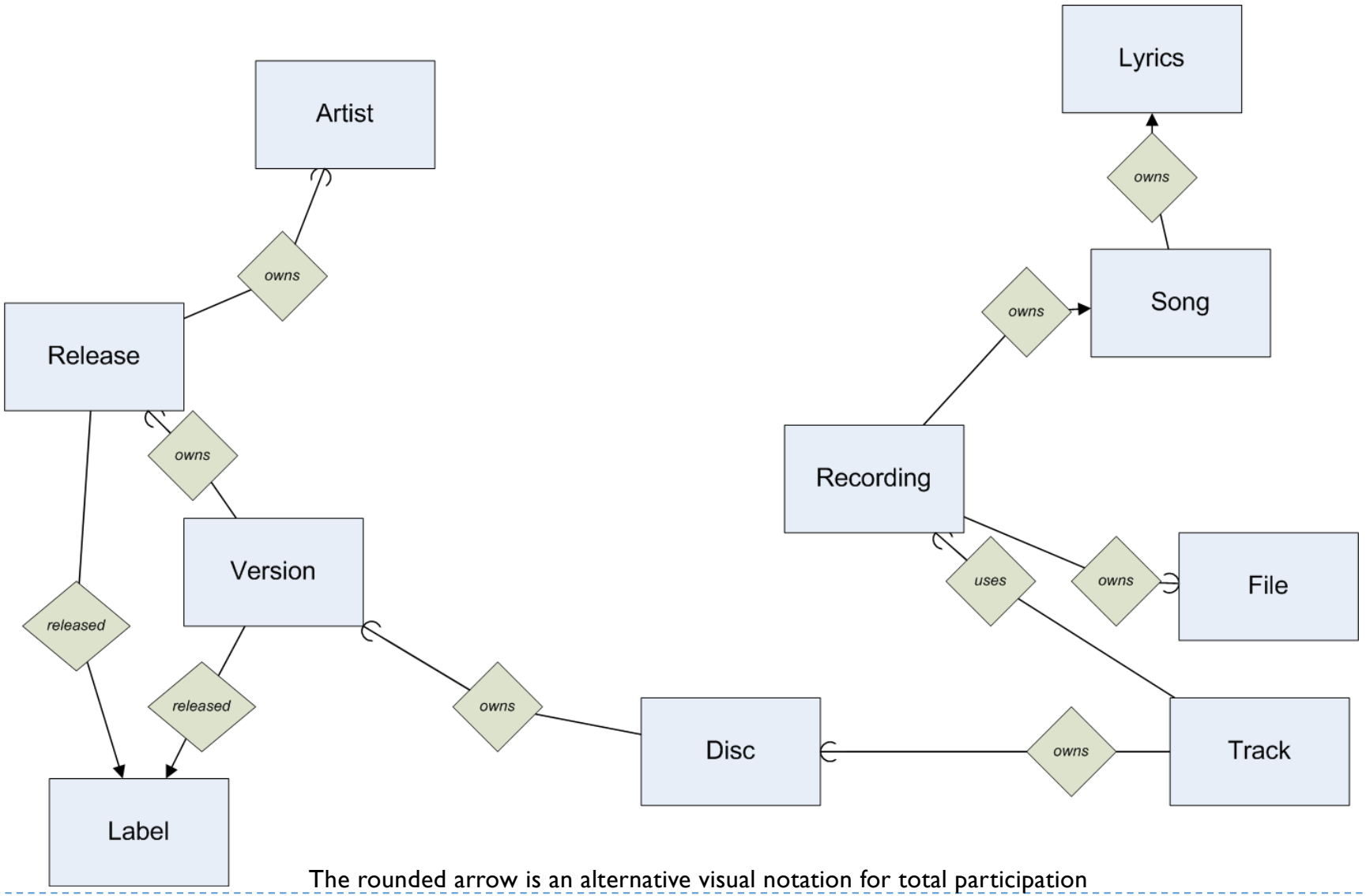
ira@cs.au.dk

More Detail



ira@cs.au.dk

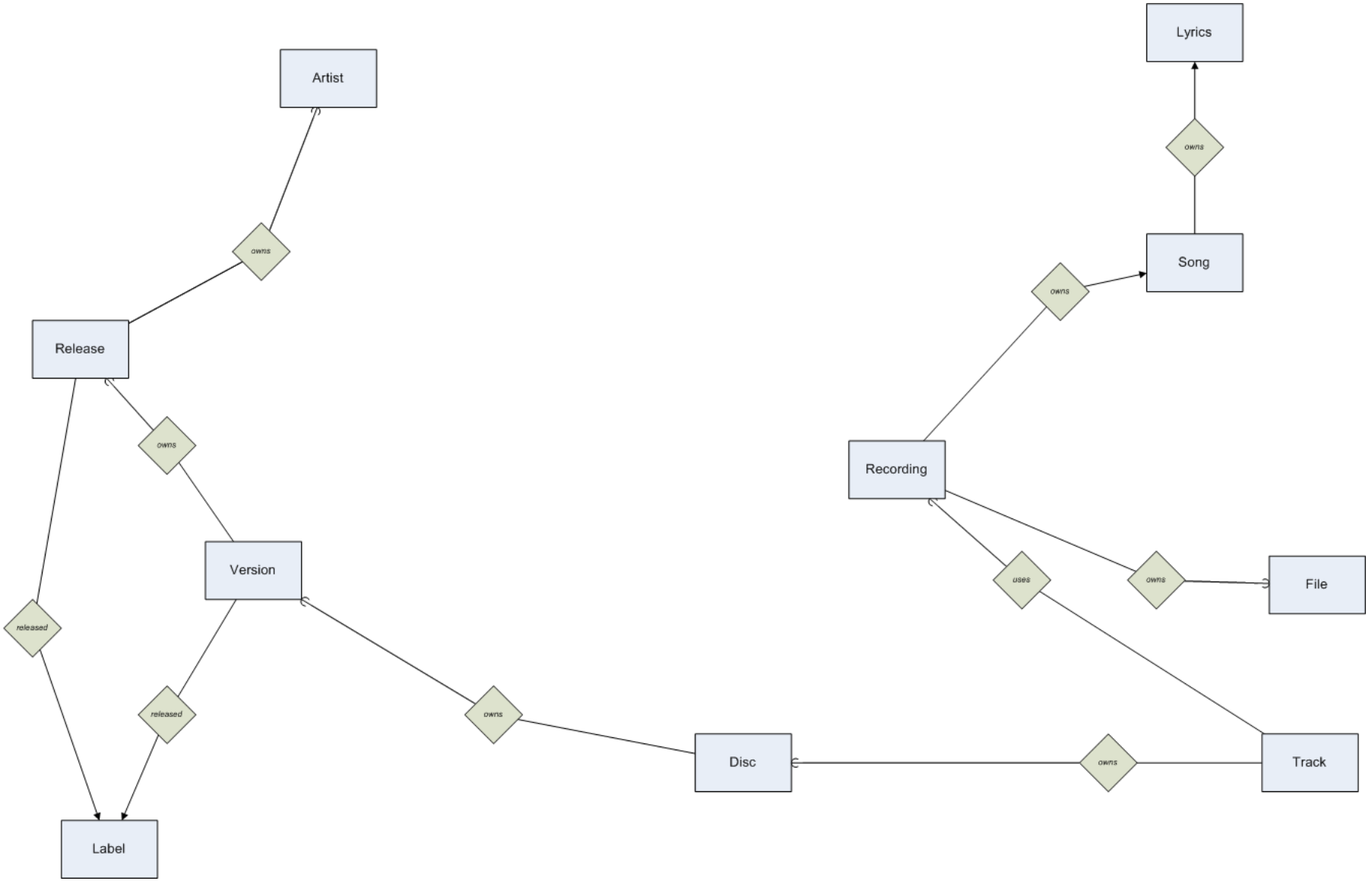
Even More Detail



The rounded arrow is an alternative visual notation for total participation (instead of double line, e.g. artist)

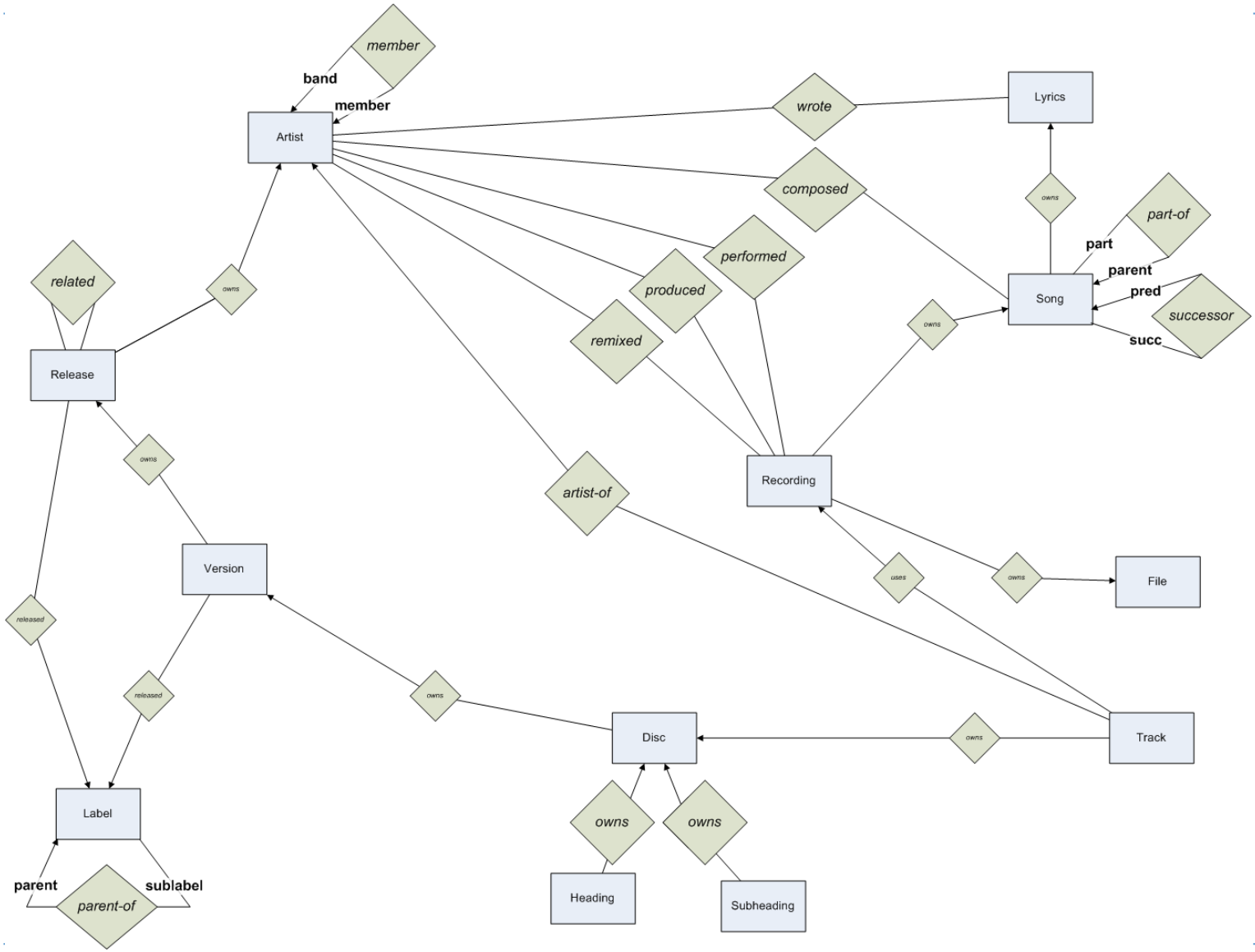
ira@cs.au.dk

Zooming Out



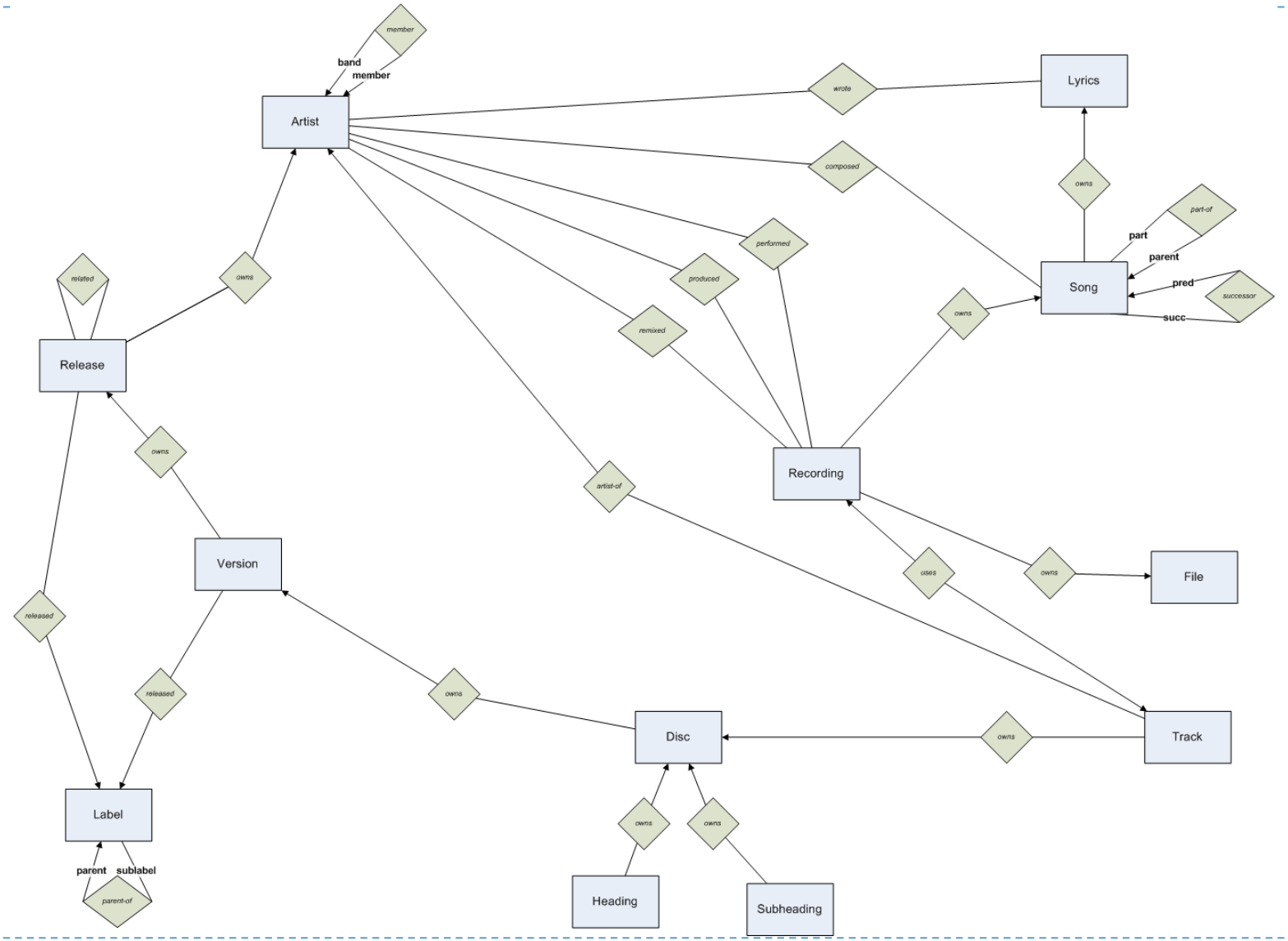
ira@cs.au.dk

Still More Details



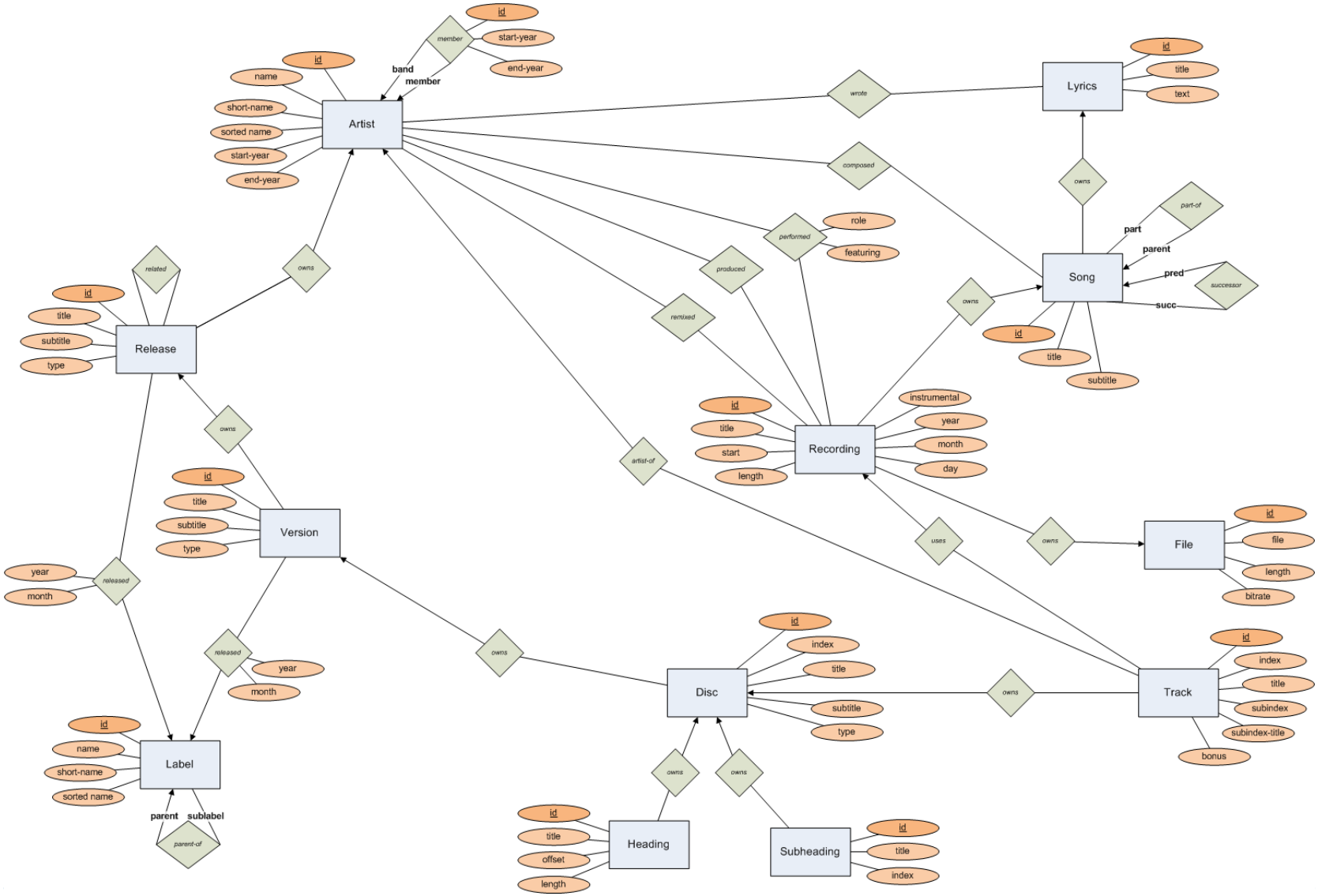
ira@cs.au.dk

Zooming Out



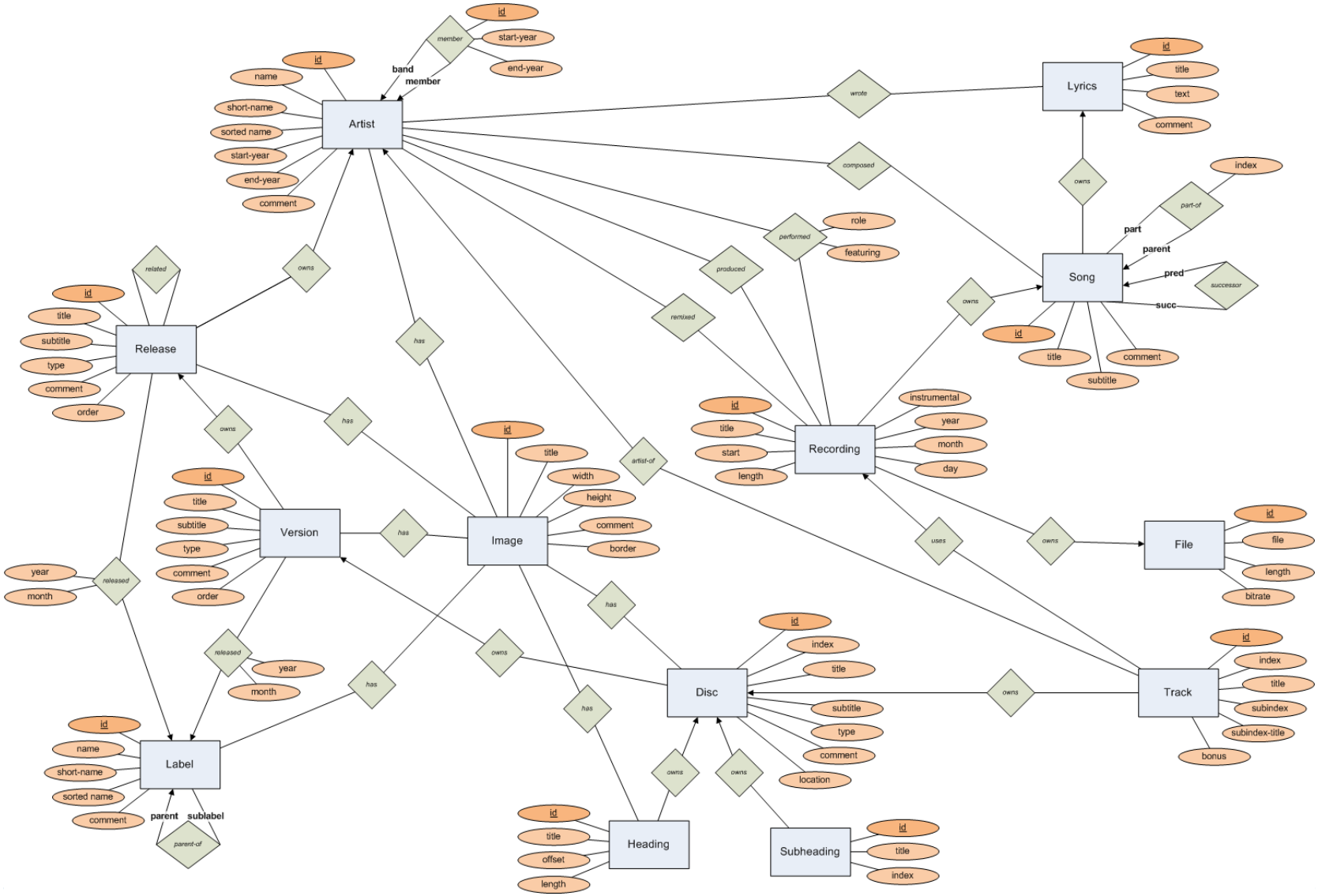
ira@cs.au.dk

Adding Attributes



ira@cs.au.dk

And Then Yet More Detail



ira@cs.au.dk

From E/R to Schemas

- A simple algorithm for generating relational schemas

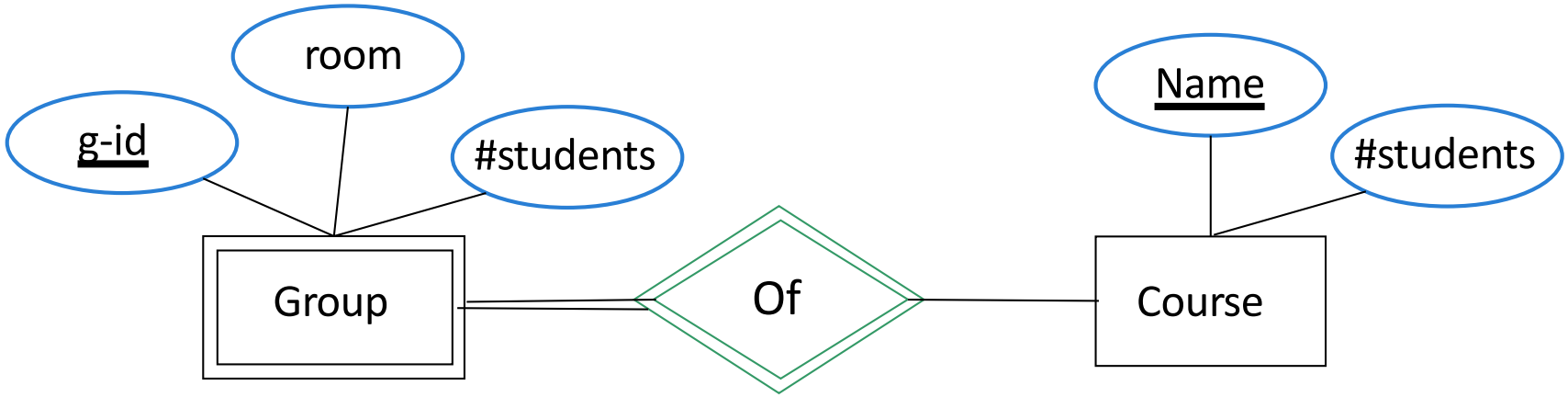
- Entity set → table
- Attribute → attribute
- Relationship → table
  - where attributes are keys or relationship attributes
- Then use cardinality ratios to simplify
  - Tables with single participation may be combined

- A good E/R design yields a reasonable schema
  - but further quality considerations are necessary
    - In part additional considerations here
    - In part when we turn to normalization

ira@cs.au.dk

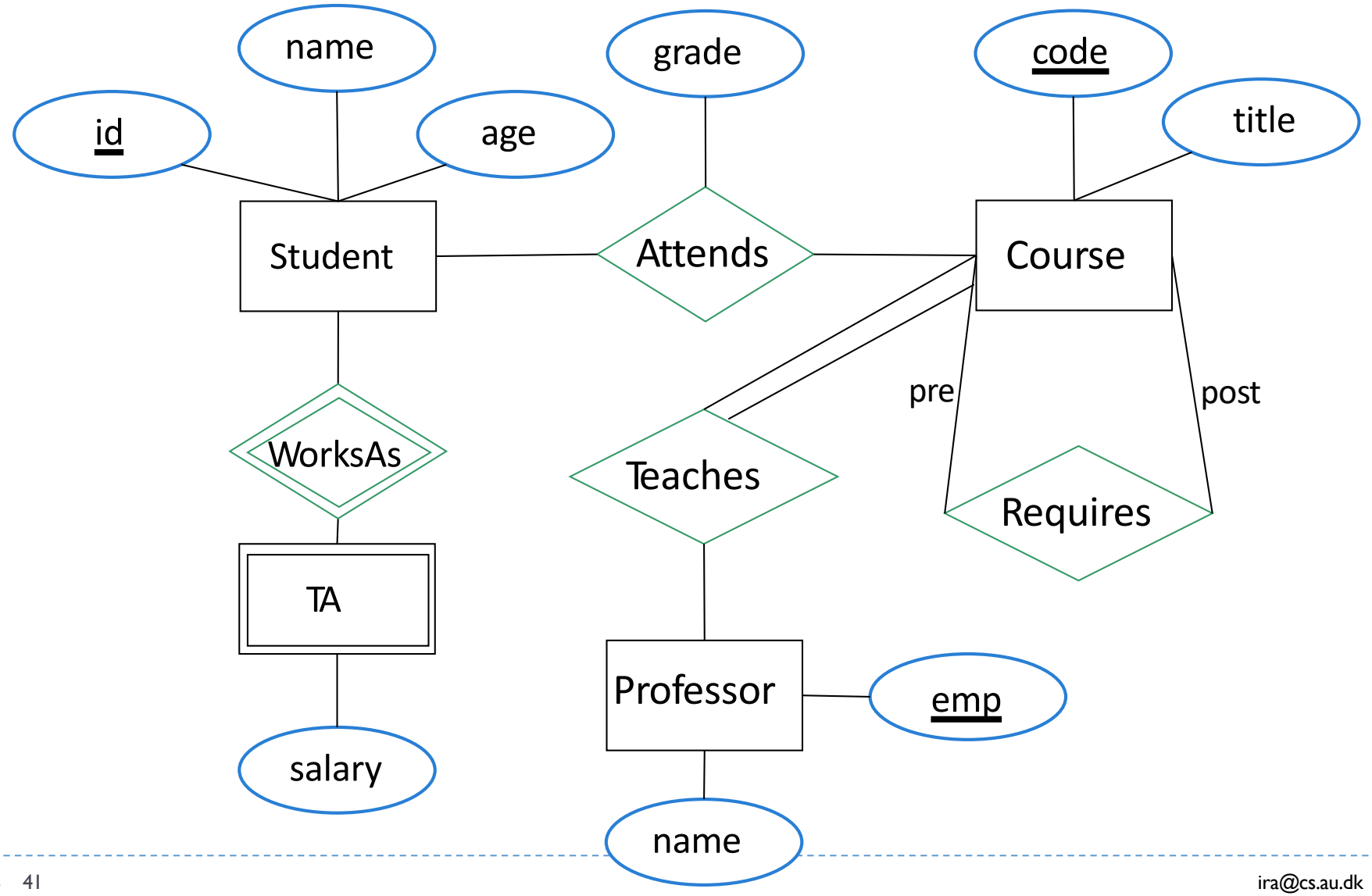
Weak entity sets

- Handling weak entity sets
  - As regular entity sets, but include also the key of the supporting entity set
  - E.g. Group(g_id, Name, room, #students)



ira@cs.au.dk

Example



ira@cs.au.dk

Generating Tables…

Student

id	name	age

Course

code	title

Requires

code	code

Attends

id	code	grade

TA

?	salary

Teaches

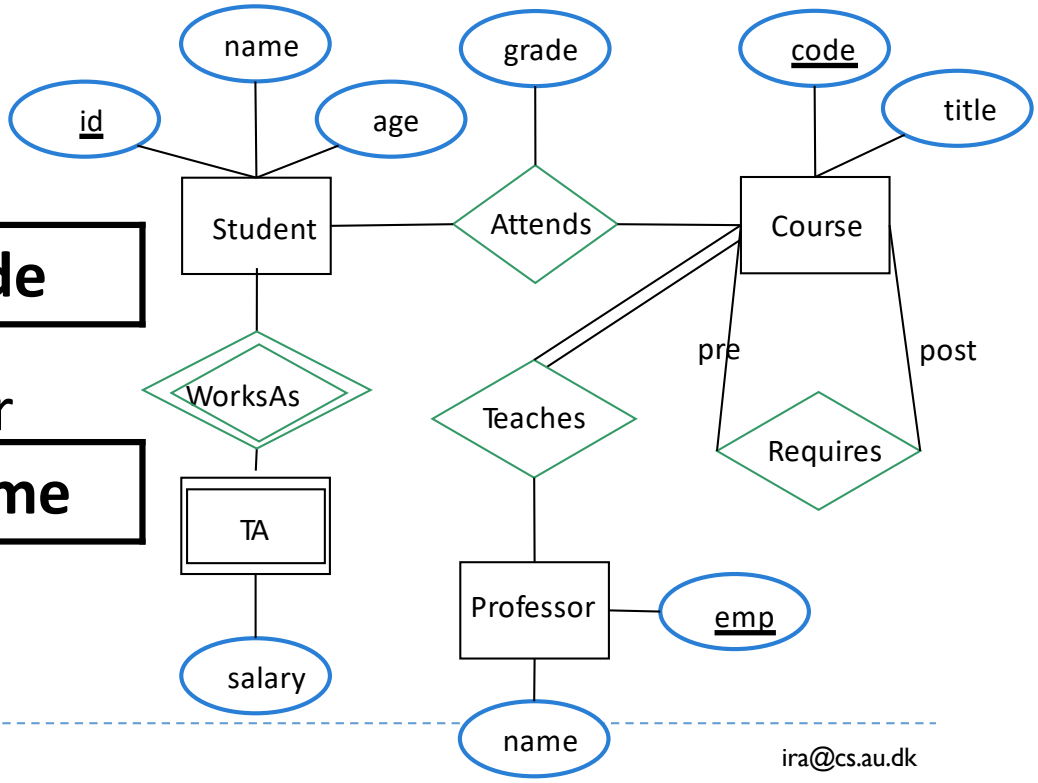
emp	code

WorksAs

id	?

Professor

emp	name



ira@cs.au.dk

Generated Tables

Student

id	name	age

Course

code	title

Requires

pre	post

Attends

id	code	grade

TA

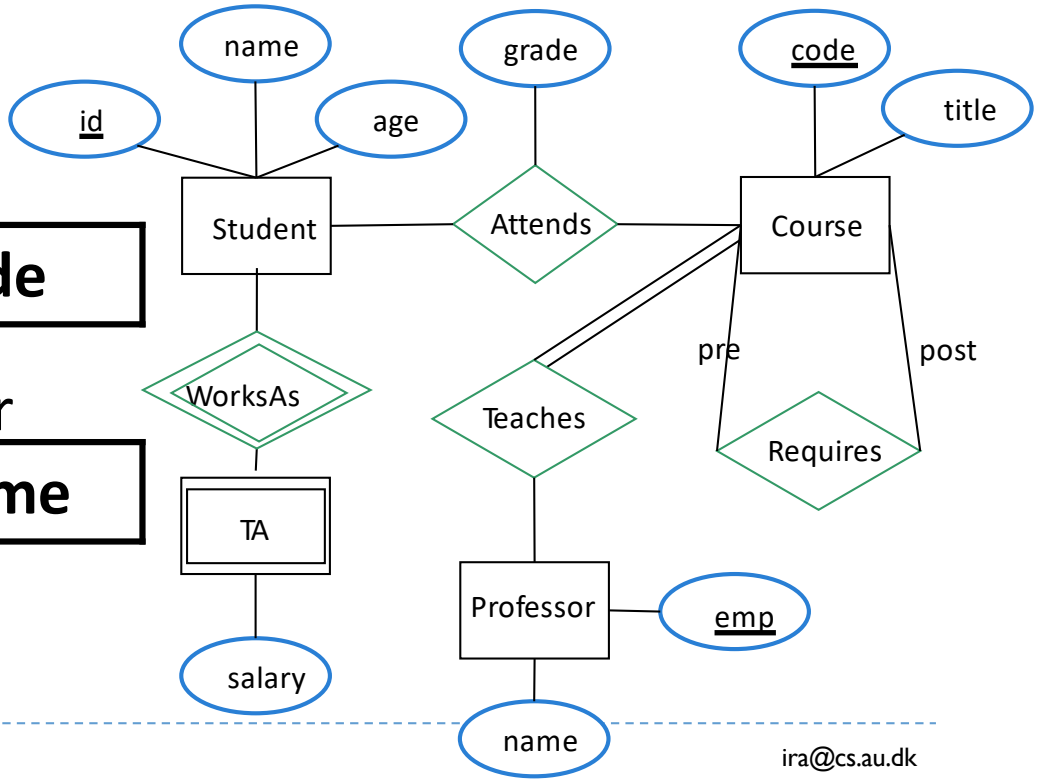
id	salary

Teaches

emp	code

Professor

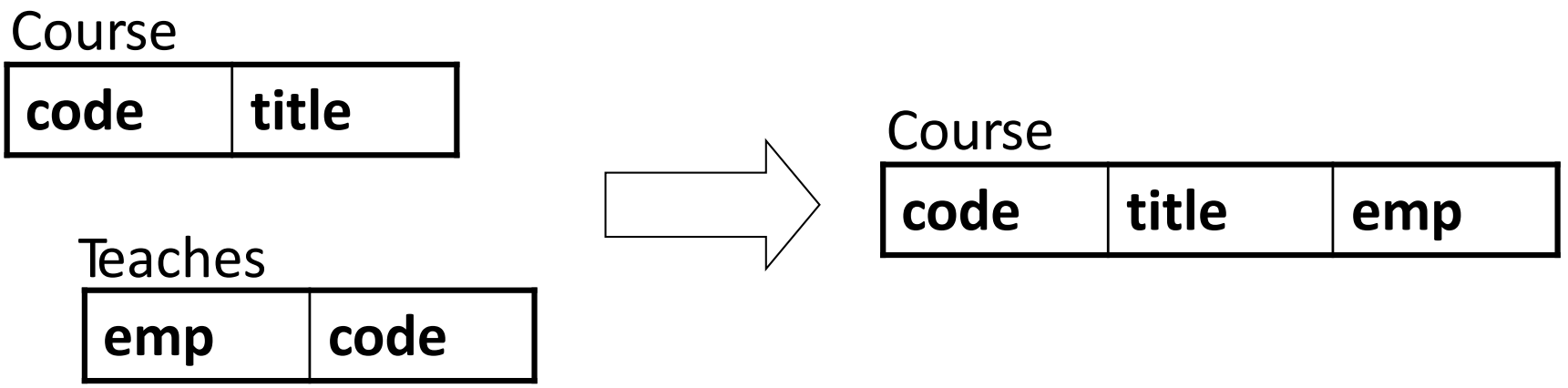
emp	name



ira@cs.au.dk

Combining Tables

- Simplify the generated schema by combining
  - the table for an entity set with
  - the tables for many-one relationships where it is the many part
    - E.g. Teaches relationship is M:1 where Course is the many-part and teacher the one-part
    - Combine into a new table where the single teacher is added to the course information



ira@cs.au.dk

When not to combine tables

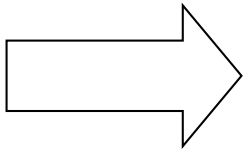
- Cannot simplify the generated schema by combining
  - the table for an entity set with the tables for many-many relationships
    - E.g. if Teaches relationship is M:N where Course and teacher both may have multiple participating entities (a teacher may teach several courses, a course may be taught by several teachers)
    - Would mean that we have to repeat information!! (will get back to redundancy when talking normalization)

Course

code	title

Teaches

emp	code



Course

code	title	emp
Red	DB	Ira
Red	DB	ChatGPT
Red	DB	Unicorn
…	…	…

ira@cs.au.dk

Where to go from here?

- So, how do we tell the database about our schema?
- How do we get data in?
- And how out again?
- The SQL language coming up next
  - Typical language for relational databases
  - Allows us to define the schema
  - Allows us to enter data
  - Allows us to query the data

ira@cs.au.dk

Summary

- Design
  - Crucial for good database performance
    - Ensure you store the right data in the right way
- E/R modeling
  - Capture entities (objects/things), their attributes (properties) and how they relate (connect)
  - Represent visually, supports discussion with stakeholders
  - Cardinality important for later mapping to schema
- Mapping to relational schema
  - Entity sets and relationship sets as well as their attributes translate mostly in a straightforward manner
  - Considerations based on cardinality

Conclusion

- Intended learning outcomes
  - Be able to
    - build a database design using the ER model
    - Follow design principles
    - Map ER model to relational schema
- Exercises:
  - Build ER models, convert to schema
  - Preparation for hand-in 1

What was this all about?

Guidelines for your own review of today's session

- ER models are used to…
- An ER diagram consists of…
  - We note information on…
- Keys are very important because…
  - They are defined as…
- We convert an ER diagram into a relation schema by…
  - Paying special attention to…
- We can further simplify the schema by…

- What is a good model / schema?

ira@cs.au.dk