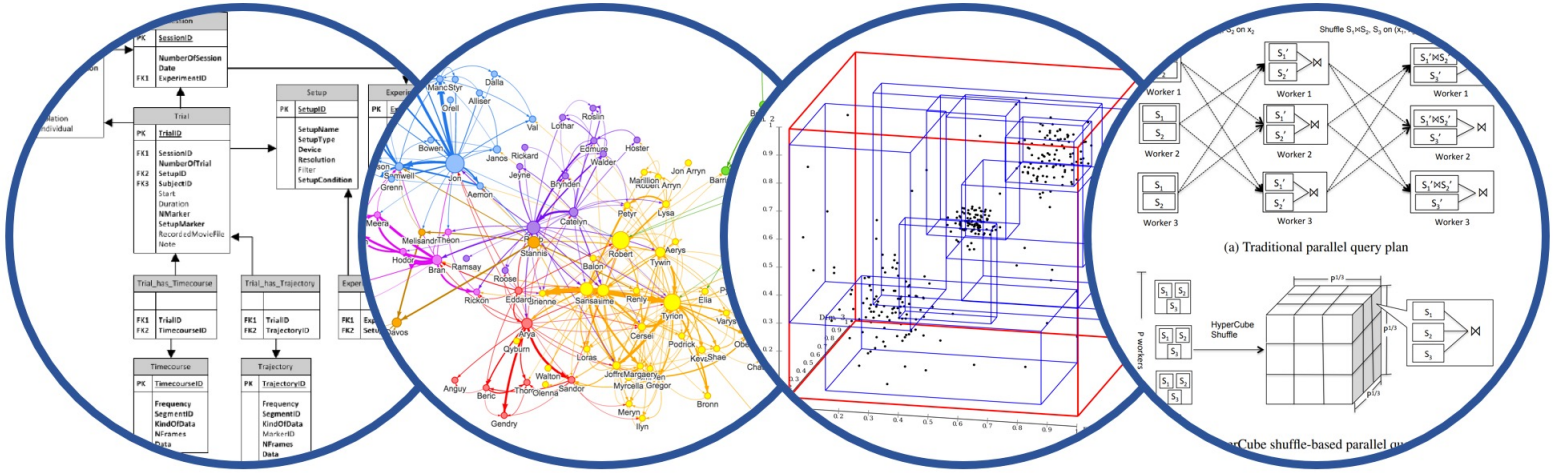


Introduction to Data Management

RA and ER Diagrams

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW3 due on Friday

- Midterm on Friday, 4/26 in class
  - Closed books, no cheat sheet (you won't need it)
  - Some practice midterms on the course website

Recap: Relational Algebra

- SQL: declarative language; we say **what**
- RA: an algebra for saying **how**
- Optimizer converts SQL to RA

Recap: Relational Algebra

1. Selection $\sigma_{\text{condition}}(S)$
2. Projection $\Pi_{\text{attrs}}(S)$
3. Join $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
4. Union $\cup$
5. Set difference $-$

- Rename ρ

Recap: Relational Algebra

1. Selection $\sigma_{\text{condition}}(S)$
2. Projection $\Pi_{\text{attrs}}(S)$
3. Join $R \bowtie_\theta S = \sigma_\theta(R \times S)$
4. Union $\cup$
5. Set difference $-$

Monotone

Non-monotone

- Rename ρ Monotone, but doesn't do anything

Query Plans

Relational Algebra Plan, or Query Plan

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Relational Algebra Plan, or Query Plan

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

$$\Pi_{\text{Name}}(\sigma_{\text{Job}='\text{TA}'}(\text{Payroll} \bowtie \text{Regist}))$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

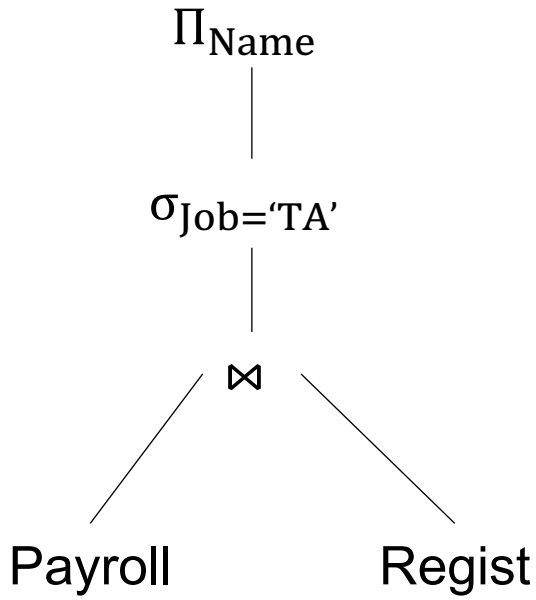
Relational Algebra Plan, or Query Plan

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

$$\Pi_{\text{Name}}(\sigma_{\text{Job}='\text{TA}'}(\text{Payroll} \bowtie \text{Regist}))$$

We write it as

a query plan



Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

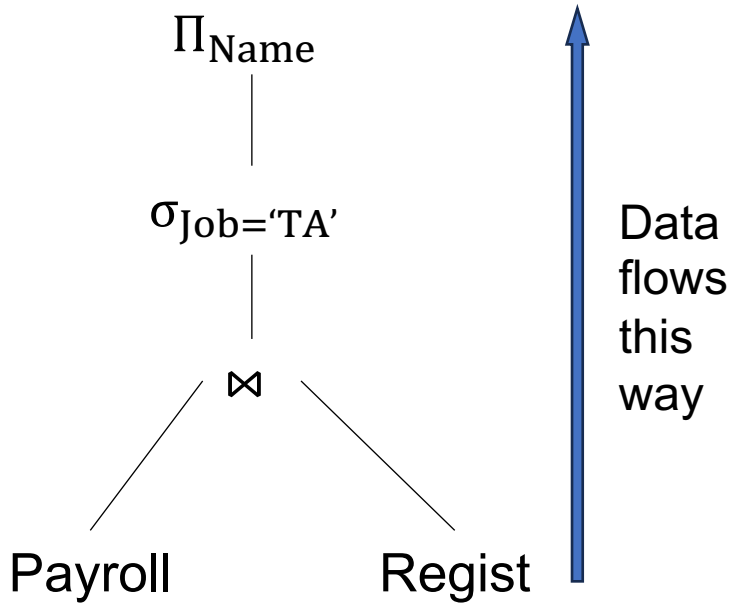
Relational Algebra Plan, or Query Plan

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

$\Pi_{\text{Name}}(\sigma_{\text{Job}='\text{TA}'}(\text{Payroll} \bowtie \text{Regist}))$

We write it as

a query plan



Payroll

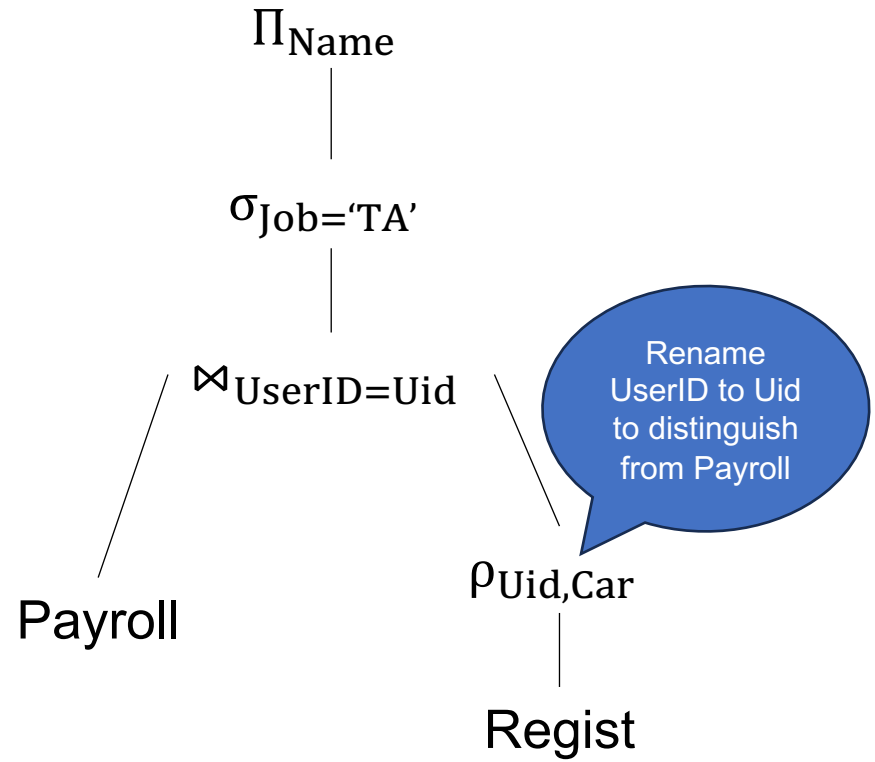
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

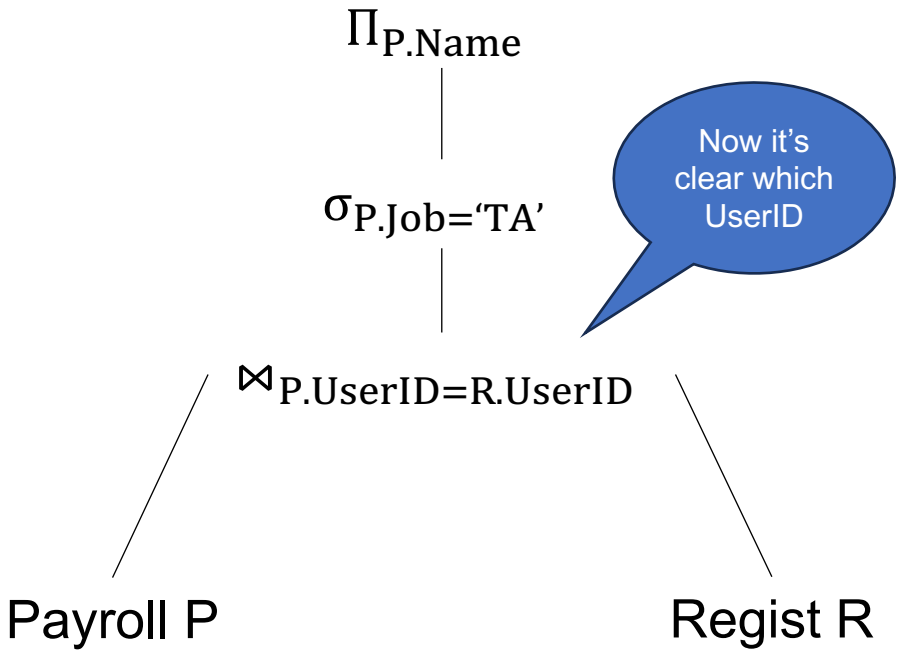
UserID	Car
123	Charger
567	Civic
567	Pinto

Query Plan: Attribute Names

Managing attribute names correctly is tedious



Better: use aliases, much like in SQL



Query Plan: Execution Order

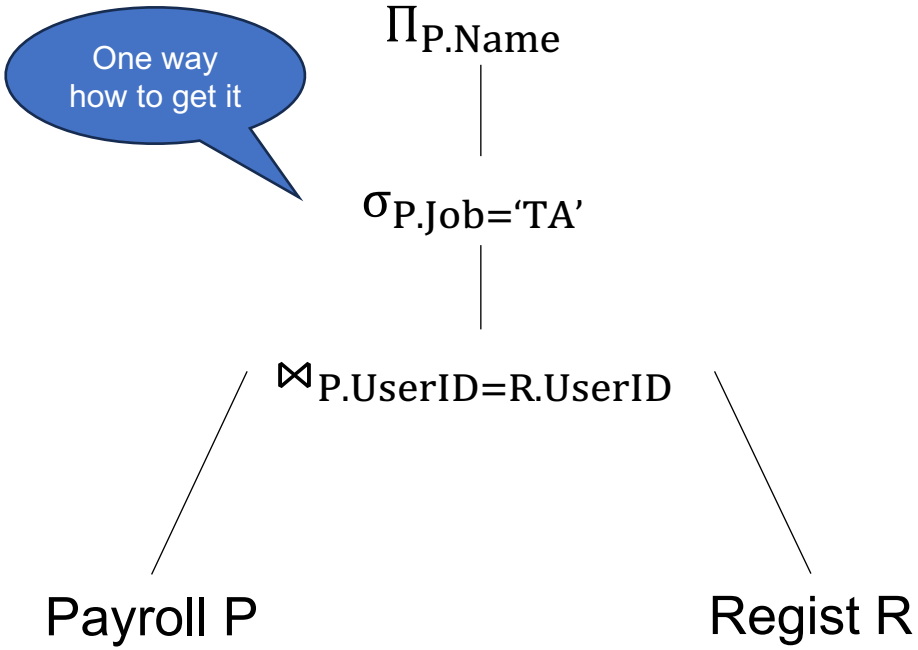
```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

We say what we want,
not how to get it

Query Plan: Execution Order

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

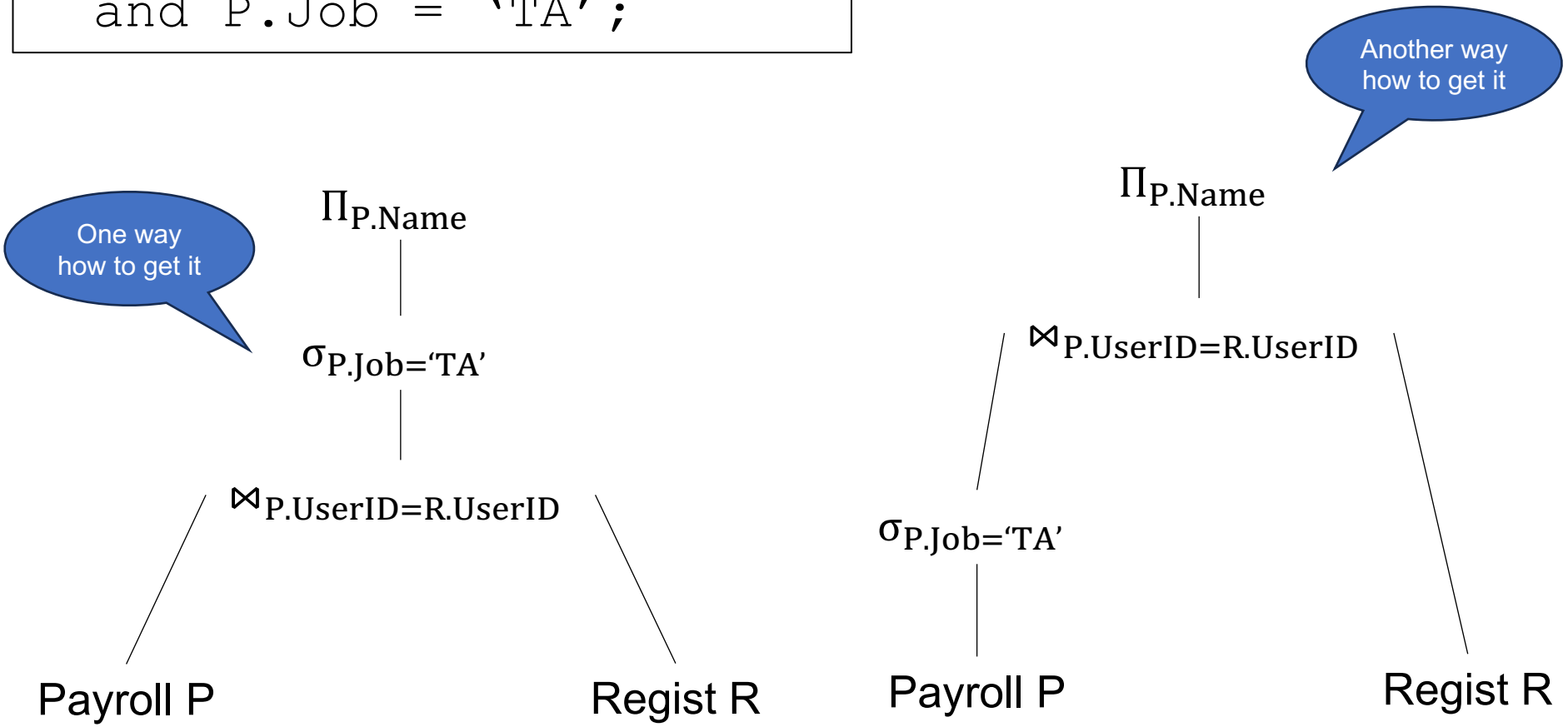
We say what we want,
not how to get it



Query Plan: Execution Order

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

We say what we want,
not how to get it

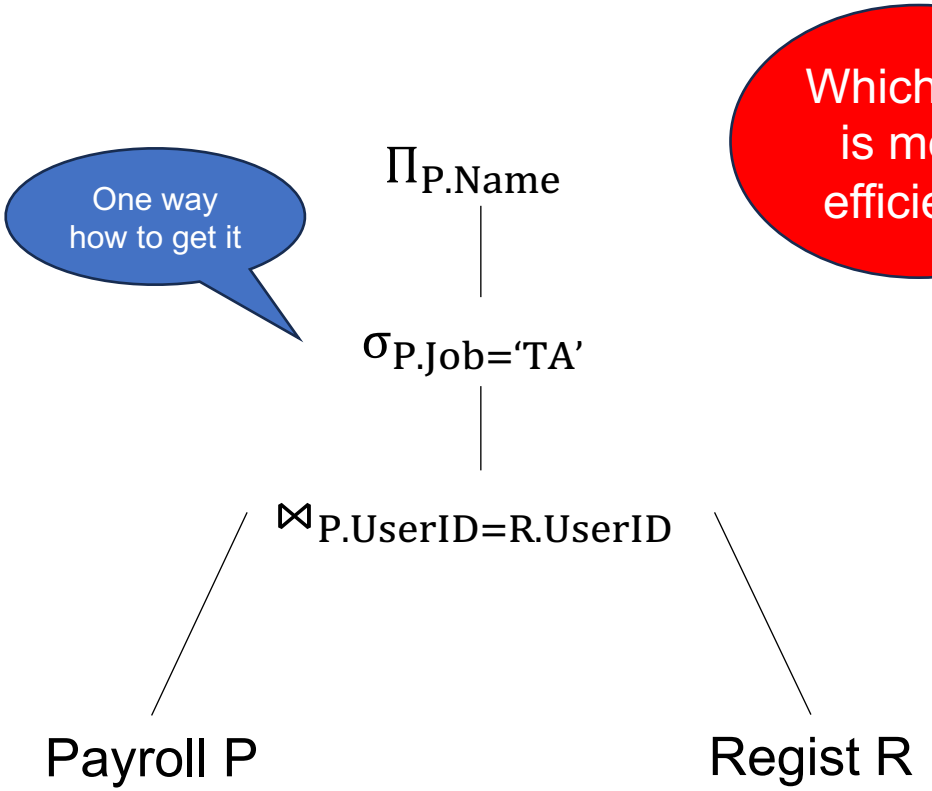


Query Plan: Execution Order

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

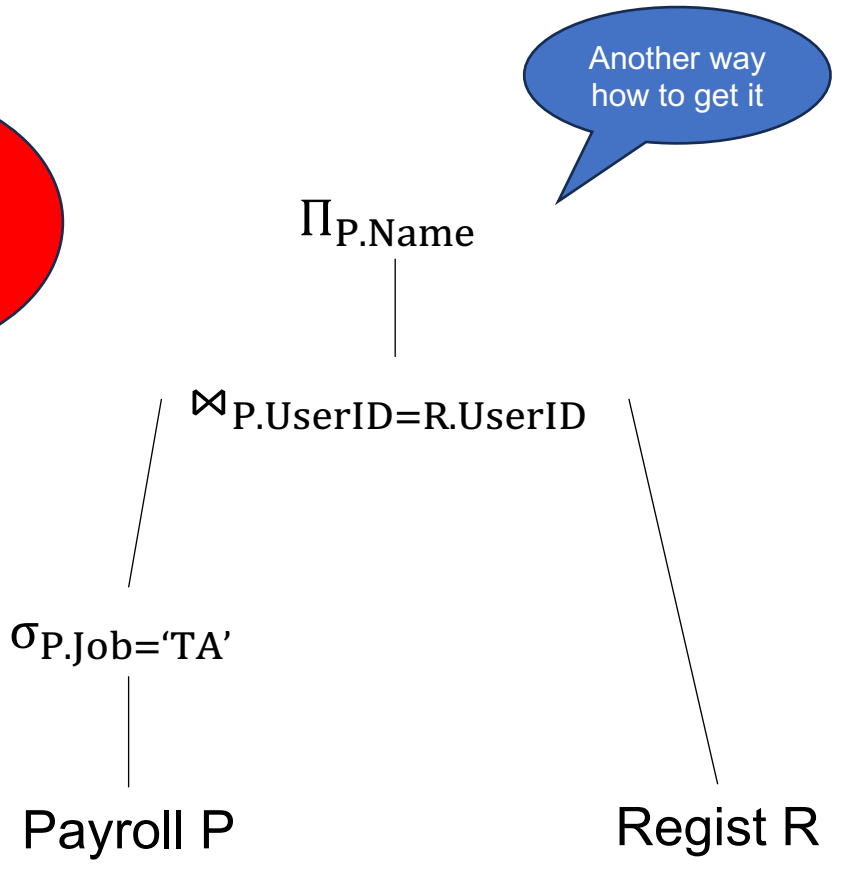
We say what we want,
not how to get it

One way
how to get it



Which one
is more
efficient?

Another way
how to get it



Query Plan: Execution Order

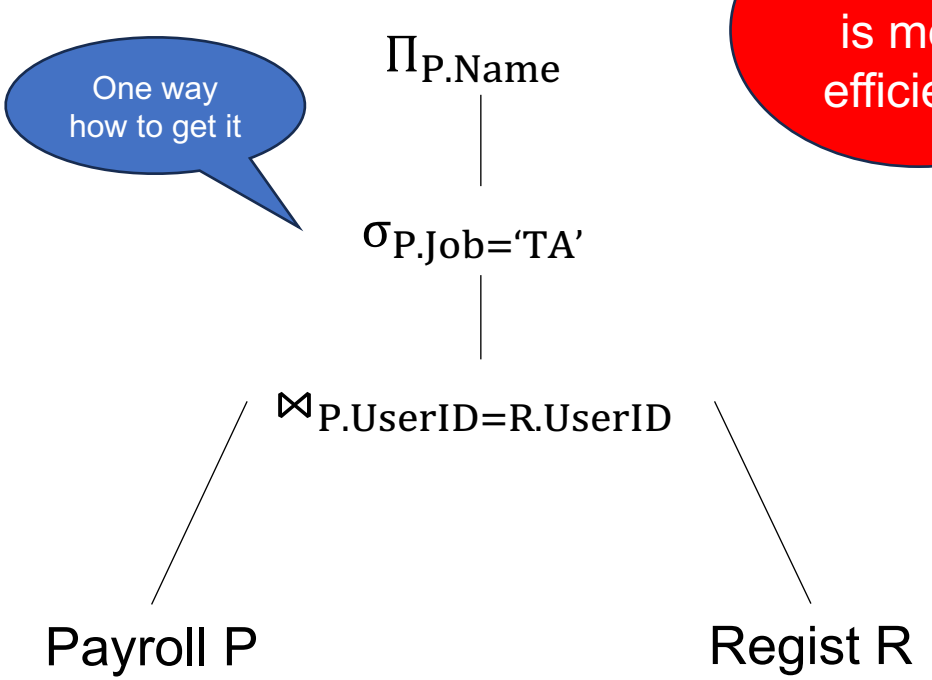
```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

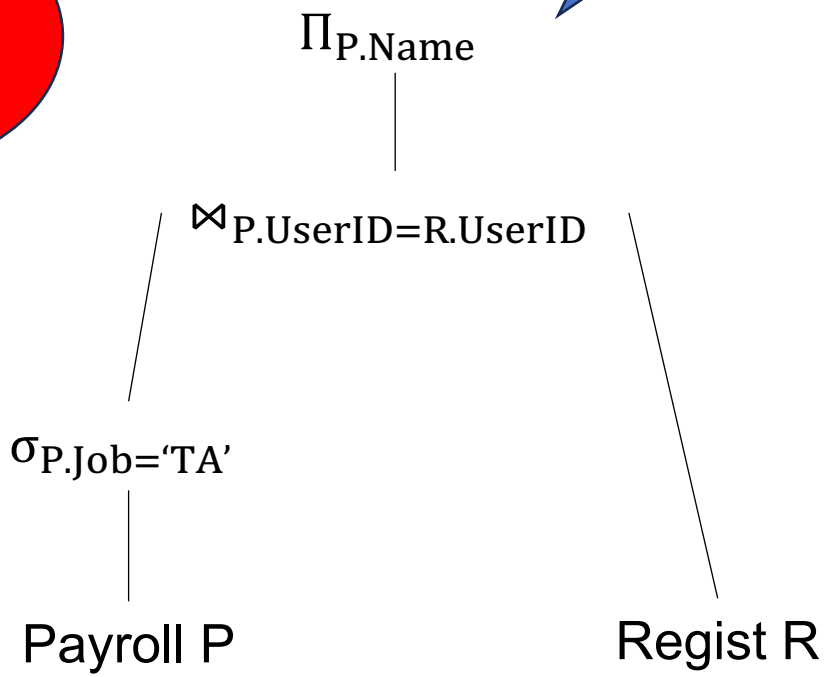
Most likely this one, and optimizer choose this

Another way how to get it

Which one is more efficient?

One way how to get it





Discussion

- Database system converts a SQL query to a Relational Algebra Plan

Discussion

- Database system converts a SQL query to a Relational Algebra Plan

- Then it optimizes the plan by exploring equivalent plans, using simple algebraic identities:

$$R \bowtie S = S \bowtie R$$
$$R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$
$$\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S$$
… many others*

*over 500 rules in SQL Server

Discussion

- Database system converts a SQL query to a Relational Algebra Plan

- Then it optimizes the plan by exploring equivalent plans, using simple algebraic identities:

$$R \bowtie S = S \bowtie R$$
$$R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$
$$\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S$$

… many others*

- Next: how to convert SQL to RA plan

*over 500 rules in SQL Server

Simple SQL to RA

SQL to RA

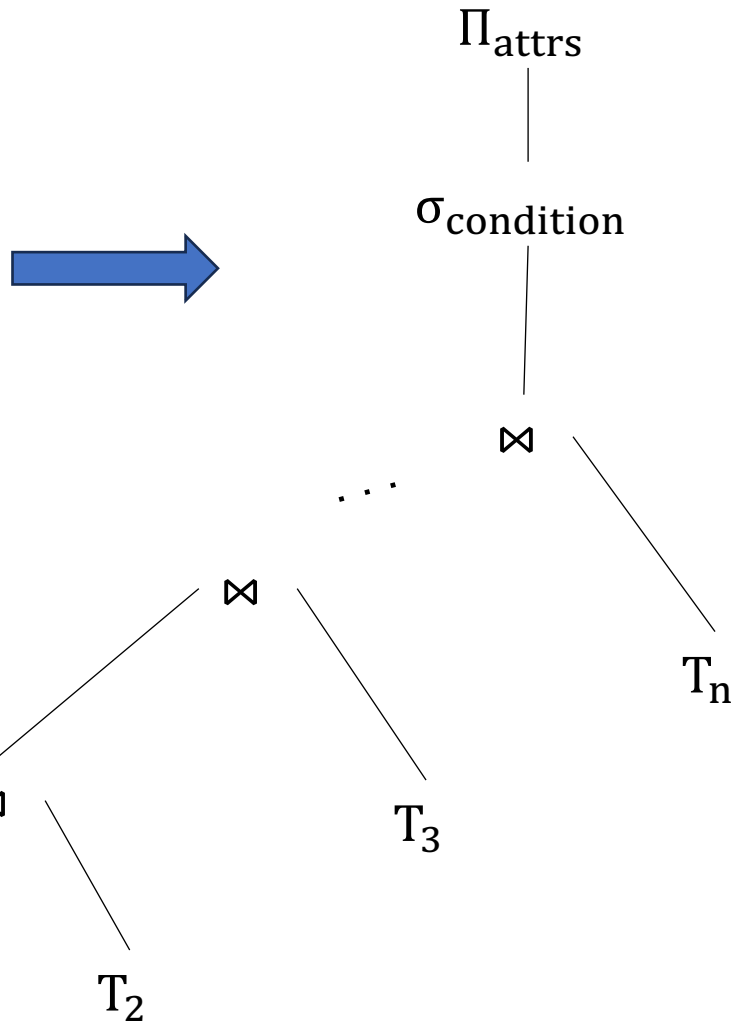
Single SELECT-FROM-WHERE query:

```
SELECT attrs
FROM T1,T2,...,Tn
WHERE condition;
```

SQL to RA

Single SELECT-FROM-WHERE query:

```
SELECT attrs
FROM T1,T2,...,Tn
WHERE condition;
```



Next: to convert group-by we need to extend RA

Extended Relational Algebra

- Duplicate elimination δ

- Group-by aggregate $\gamma_{\text{attr1,attr2,...,agg1,...}}$

Duplicate Elimination

$\delta(\mathrm{T})$

Eliminates duplicates from the bag T

```
SELECT DISTINCT *
FROM T;
```

Duplicate Elimination

$\delta(\mathrm{T})$

Eliminates duplicates from the bag T

$\delta(\mathrm{R}) =$

```
SELECT DISTINCT *
FROM T;
```

R

A	B
1	10
2	10
2	10
2	20
1	10

Duplicate Elimination

$\delta(\mathrm{T})$

A	B
1	10
2	10
2	20

Eliminates duplicates from the bag T

$\delta(\mathrm{R}) =$ 

```
SELECT DISTINCT *
FROM T;
```

R

A	B
1	10
2	10
2	10
2	20
1	10

GroupBy-Aggregate

$$\gamma_{attr1,attr2,\ldots,agg1,\ldots}(\mathrm{T})$$

Group-by, then aggregate

```
SELECT attr1,...,agg1,...
FROM T
GROUP BY attr1,...;
```

GroupBy-Aggregate

$$\gamma_{attr1,attr2,\ldots,agg1,\ldots}(\mathrm{T})$$

Group-by, then aggregate

$$\gamma_{\mathrm{Job},\mathrm{avg}(\mathrm{Salary})\rightarrow \mathrm{S}}(\mathrm{R}) =$$

```
SELECT attr1,...,agg1,...
FROM T
GROUP BY attr1,...;
```

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

GroupBy-Aggregate

$\gamma_{attr1,attr2,\ldots,agg1,\ldots}(\mathrm{T})$

Job	S
TA	55000
Prof	95000

Group-by, then aggregate

$\gamma_{\mathrm{Job,avg(Salary)}\rightarrow \mathrm{S}}(\mathrm{R}) =$ 

```
SELECT attr1,...,agg1,...
FROM T
GROUP BY attr1,...;
```

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the
average salary of employees
earning over 55000
is < 70000

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the
average salary of employees
earning over 55000
is < 70000

```
SELECT Job
FROM Payroll
WHERE Salary > 55000
GROUP BY Job
HAVING avg(Salary)<70000;
```

Payroll

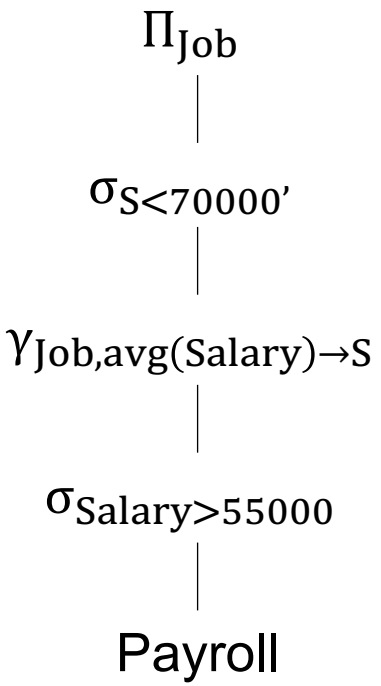
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the average salary of employees earning over 55000 is < 70000

```
SELECT Job
FROM Payroll
WHERE Salary > 55000
GROUP BY Job
HAVING avg(Salary)<70000;
```



Payroll

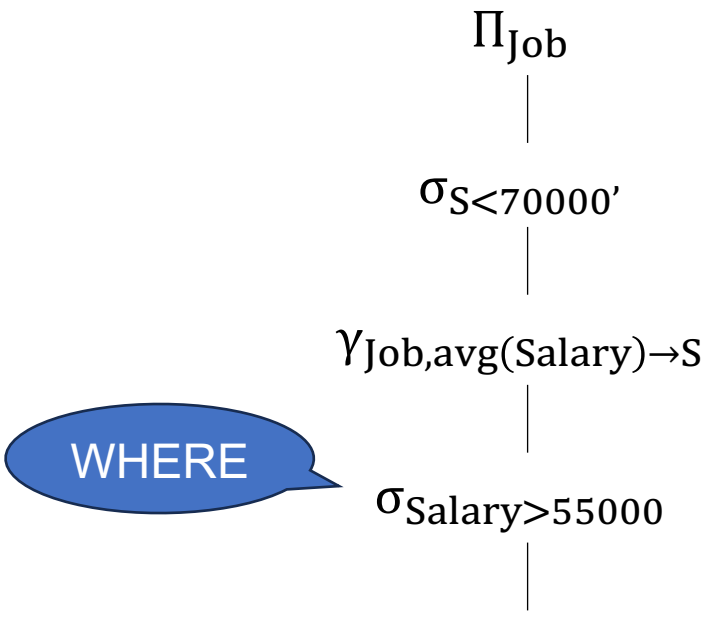
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the average salary of employees earning over 55000 is < 70000

```
SELECT Job
FROM Payroll
WHERE Salary > 55000
GROUP BY Job
HAVING avg(Salary)<70000;
```



Payroll

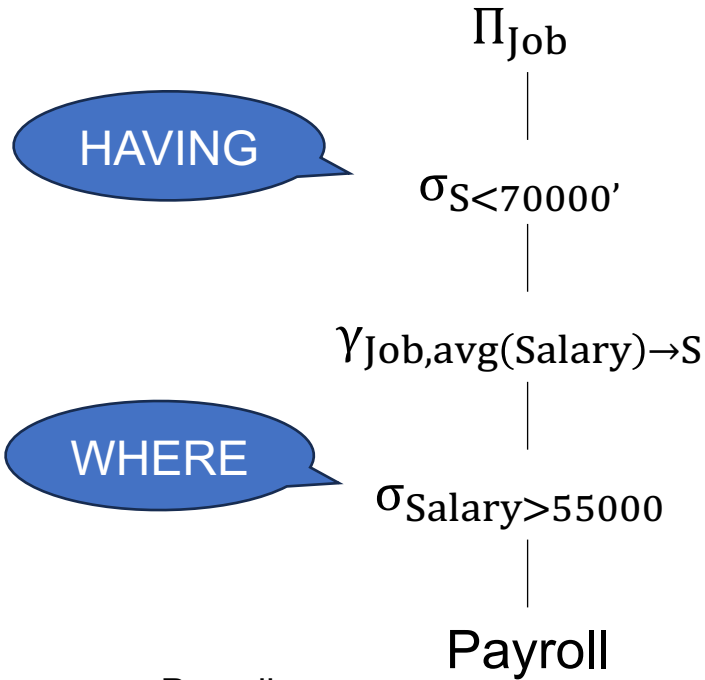
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the average salary of employees earning over 55000 is < 70000

```
SELECT Job
FROM Payroll
WHERE Salary > 55000
GROUP BY Job
HAVING avg(Salary)<70000;
```



Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Discussion

The Greek alphabet soup:

- $\sigma, \Pi, \delta, \gamma$

- They are standard RA symbols, get used to them

Next: converting nested SQL queries to RA

Nested SQL to RA

Nested Queries to RA

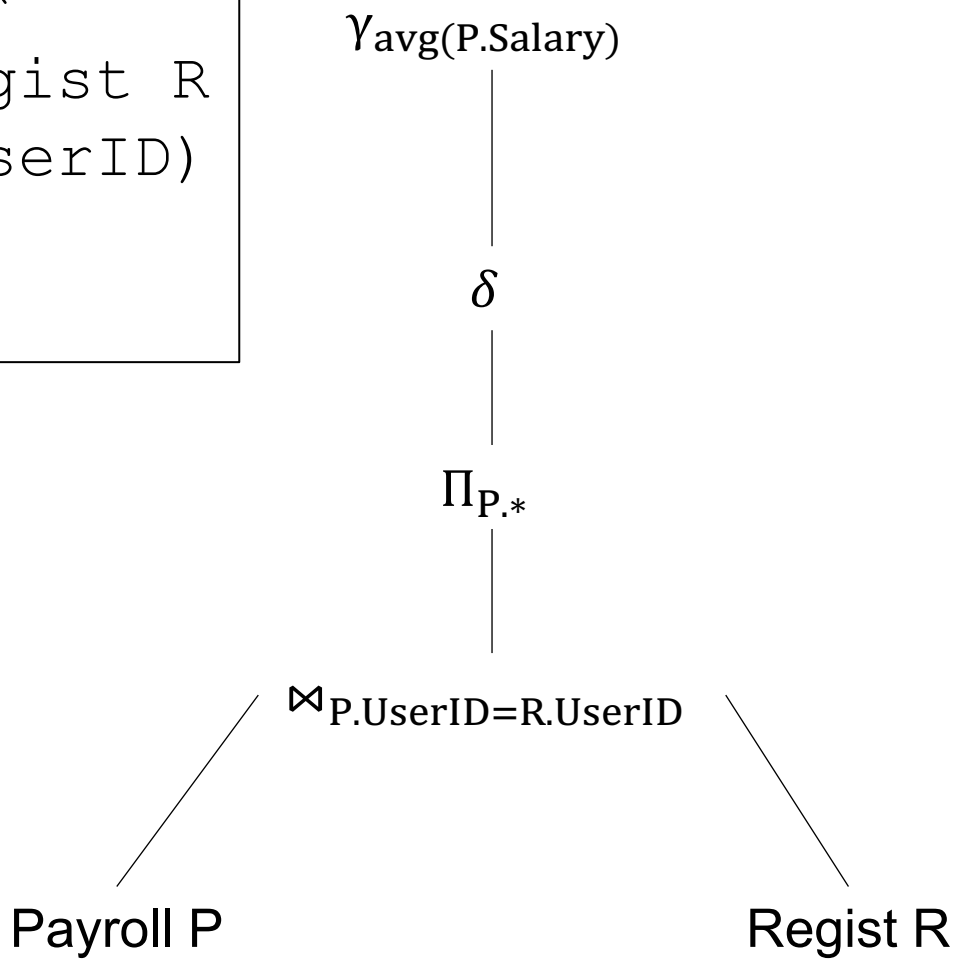
- RA is an algebra: has no nested expressions
- We cannot write EXISTS or NOT EXISTS in σ
- First unnest SQL query, then convert to RA

A Simple Case: the WITH Clause

```
WITH Cardrivers AS
   (SELECT DISTINCT P.*
    FROM Payroll P, Regist R
    WHERE P.UserId=R.UserID)
SELECT avg(Salary)
FROM Cardrivers;
```

A Simple Case: the WITH Clause

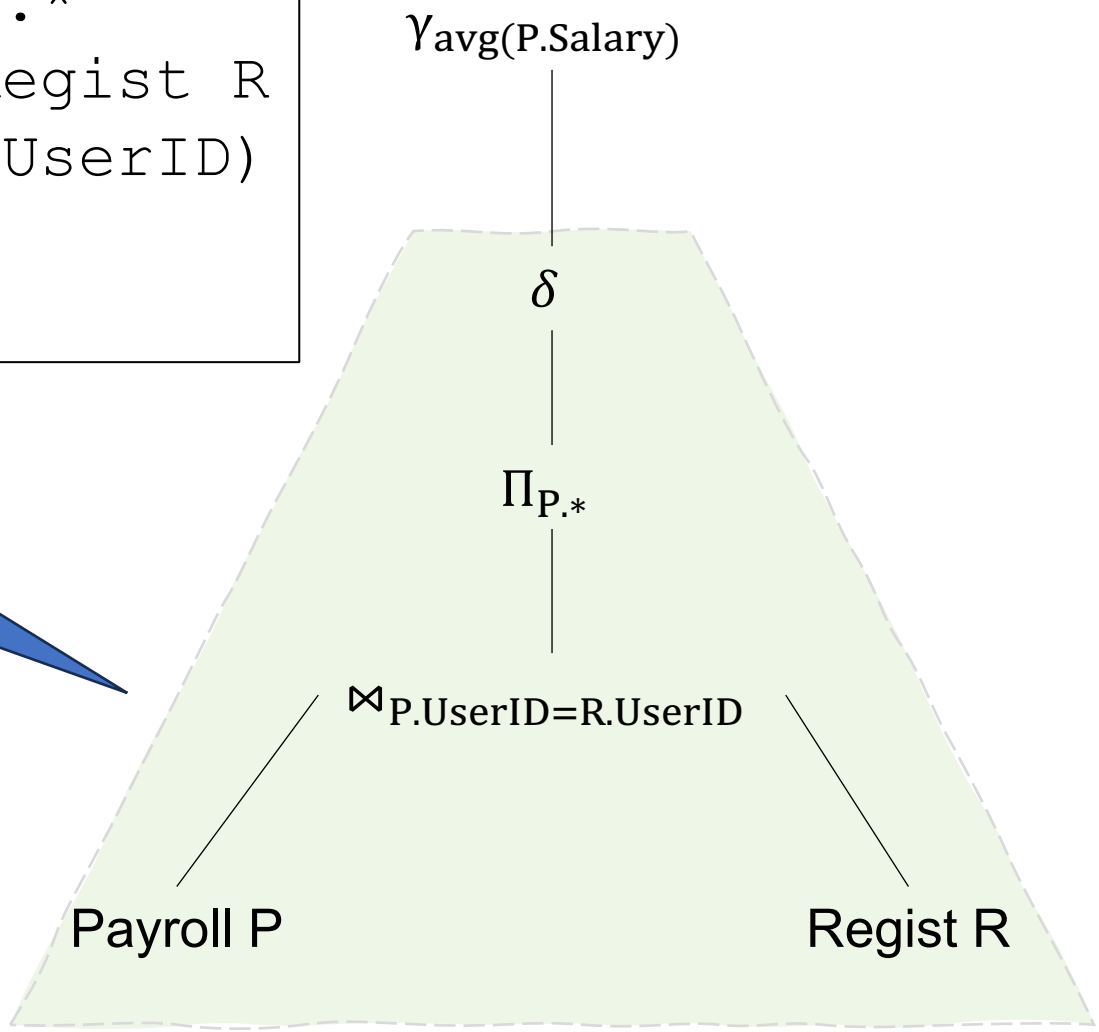
```
WITH Cardrivers AS
   (SELECT DISTINCT P.*
    FROM Payroll P, Regist R
    WHERE P.UserId=R.UserID)
SELECT avg(Salary)
FROM Cardrivers;
```



A Simple Case: the WITH Clause

```
WITH Cardrivers AS
   (SELECT DISTINCT P.*
    FROM Payroll P, Regist R
    WHERE P.UserId=R.UserID)
SELECT avg(Salary)
FROM Cardrivers;
```

Computes Cardrivers

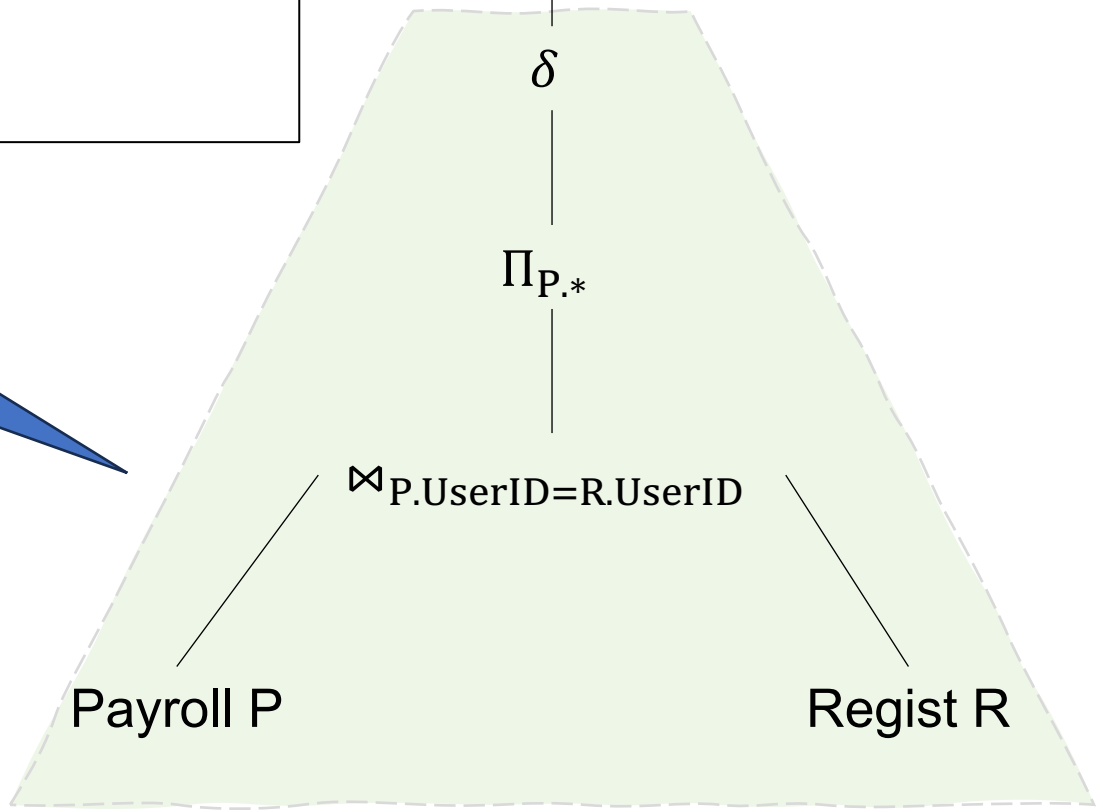


A Simple Case: the WITH Clause

```
WITH Cardrivers AS
   (SELECT DISTINCT P.*
    FROM Payroll P, Regist R
    WHERE P.UserId=R.UserID)
SELECT avg(Salary)
FROM Cardrivers;
```

Does the rest

Computes Cardrivers



A Simple Case: a Monotone Query

```
SELECT P.UserID, P.Name
FROM Payroll P
WHERE exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

A Simple Case: a Monotone Query

```
SELECT P.UserID, P.Name
FROM Payroll P
WHERE exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

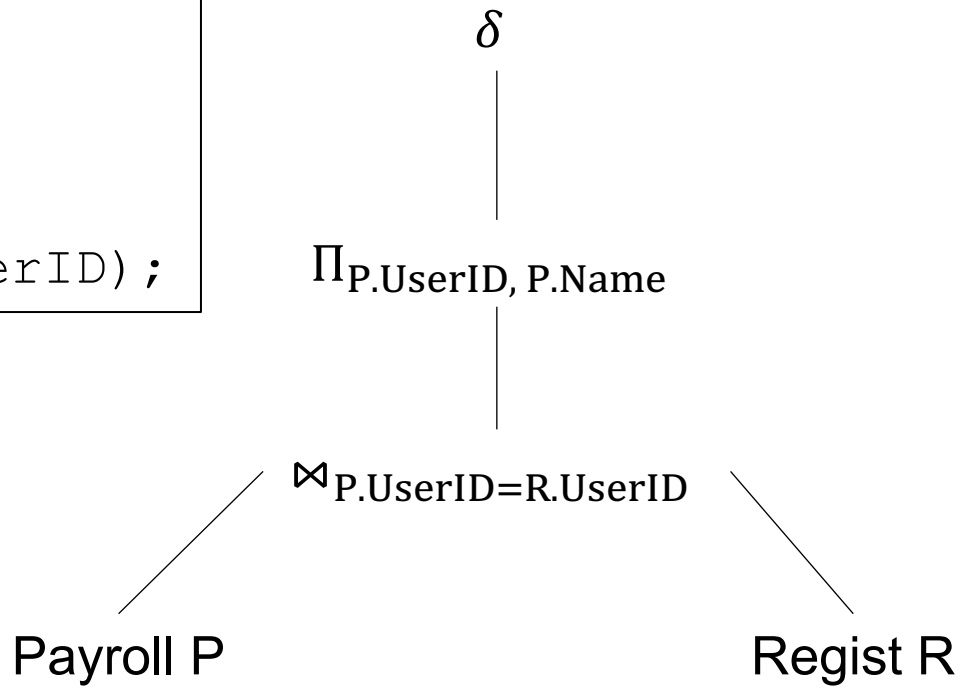
First unnest

```
SELECT DISTINCT P.UserID, P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID;
```

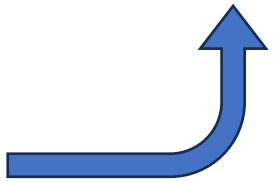
A Simple Case: a Monotone Query

```
SELECT P.UserID, P.Name
FROM Payroll P
WHERE exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

First unnest




```
SELECT DISTINCT P.UserID, P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID;
```

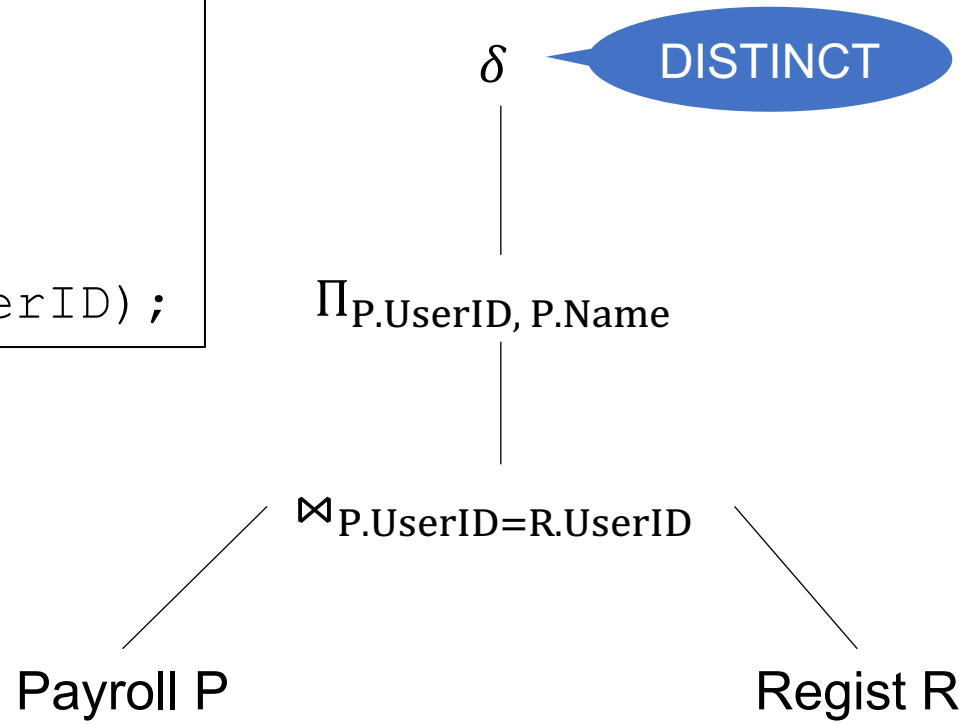


The convert to RA

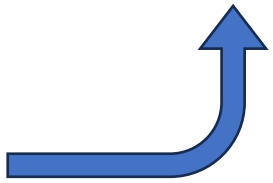
A Simple Case: a Monotone Query

```
SELECT P.UserID, P.Name
FROM Payroll P
WHERE exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

First unnest




```
SELECT DISTINCT P.UserID, P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID;
```



The convert to RA

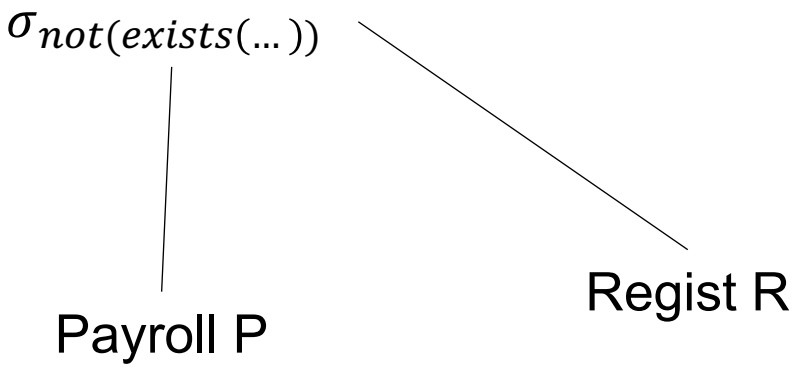
A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

Totally, totally wrong!

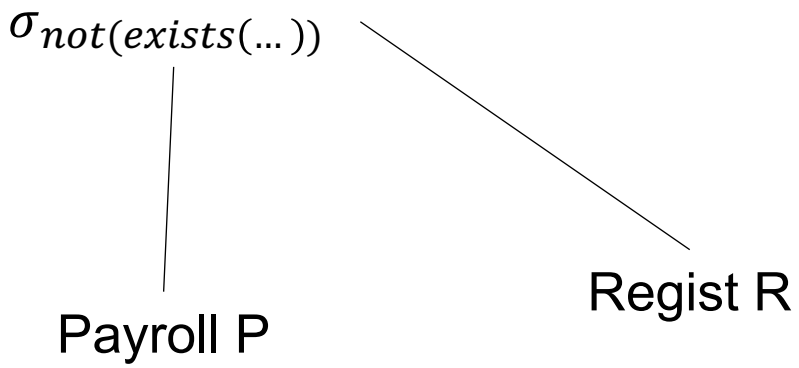


A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

Totally, totally wrong!

There are no subqueries in RA.

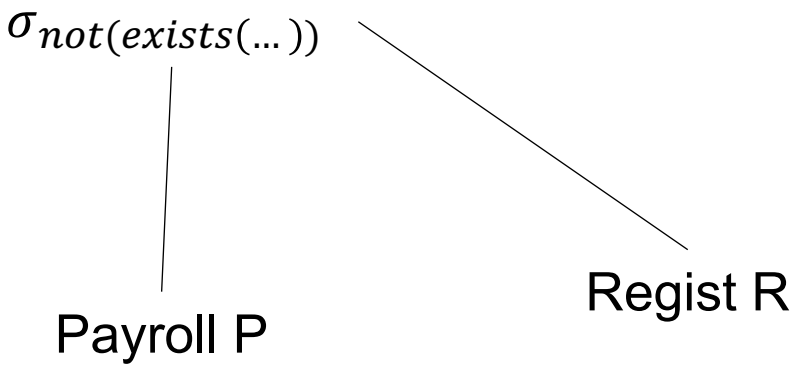


A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

Totally, totally wrong!

There are no subqueries in RA.



Need to unnest,
but first need to de-correlate.

A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

First
de-correlate

```
SELECT P.UserID
FROM Payroll P
WHERE P.UserID not in
       (SELECT R.UserID
        FROM Regist R);
```

A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

First
de-correlate

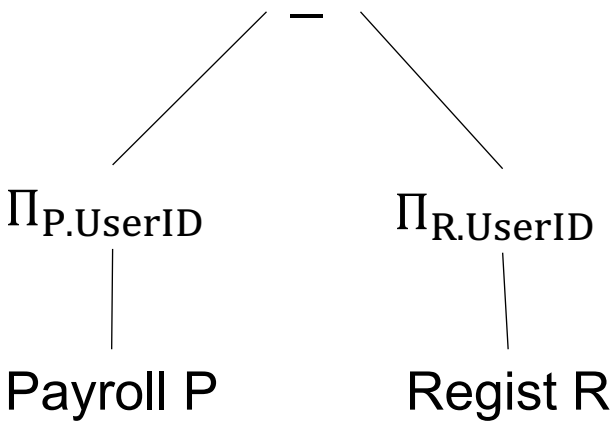
```
SELECT P.UserID
FROM Payroll P
WHERE P.UserID not in
       (SELECT R.UserID
        FROM Regist R);
```

Then unnest
using set difference

```
SELECT P.UserID
FROM Payroll P
   EXCEPT
SELECT R.UserID
FROM Regist R;
```

A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
        (SELECT *
         FROM Regist R
         WHERE P.UserID = R.UserID);
```



First
de-correlate

Finally,
rewrite to RA

```
SELECT P.UserID
FROM Payroll P
WHERE P.UserID not in
        (SELECT R.UserID
         FROM Regist R);
```

Then unnest
using set difference

```
SELECT P.UserID
FROM Payroll P
    EXCEPT
SELECT R.UserID
FROM Regist R;
```

Discussion

- SQL = declarative language; what we want
  RA = an algebra; how to get it

- We write in SQL, optimizers generates RA

- Some language resemble RA more than SQL, e.g. Spark

Next topic: how to design a database from scratch

Database Design

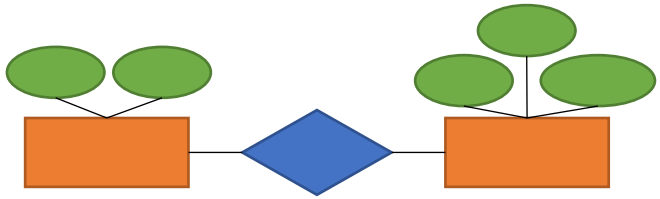
Database Design

- New application needs persistent database.
- The database will persist for a long period of time. We need a good design from day 1.
- Incorporate feedback from many stakeholders
  - Programmers, business teams, analysts, data scientists, product managers, …

The Database Design Process

Today

Conceptual Model



Relational Model

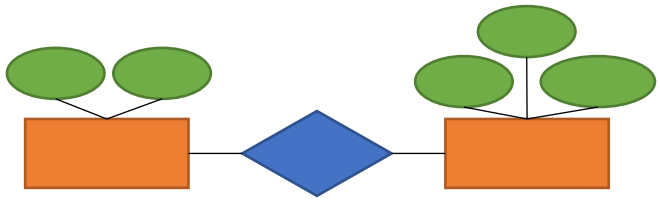
+ Schema

+ Constraints

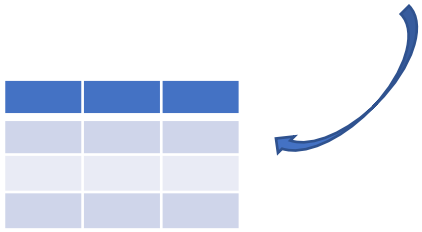
The Database Design Process

Today

Conceptual Model

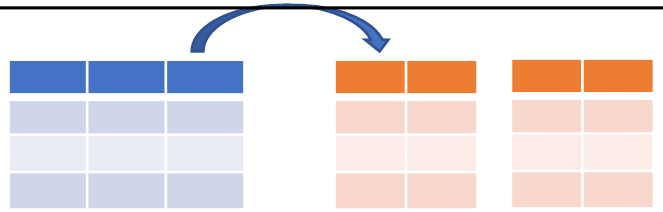


Relational Model
+ Schema
+ Constraints



Next Lectures

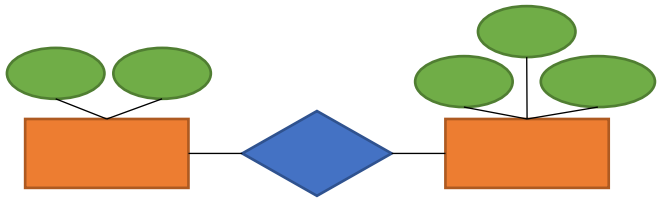
Conceptual Schema
+ Normalization



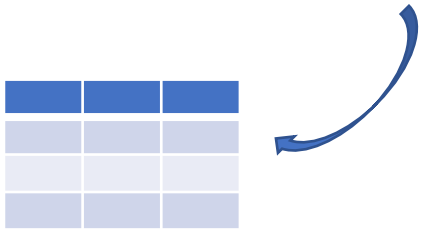
The Database Design Process

Today

Conceptual Model

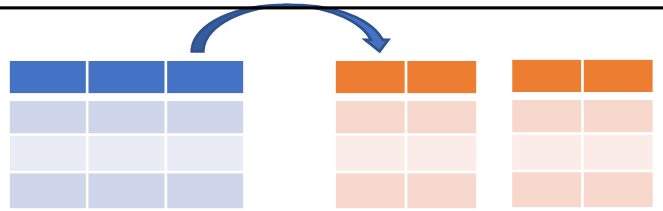


Relational Model
+ Schema
+ Constraints



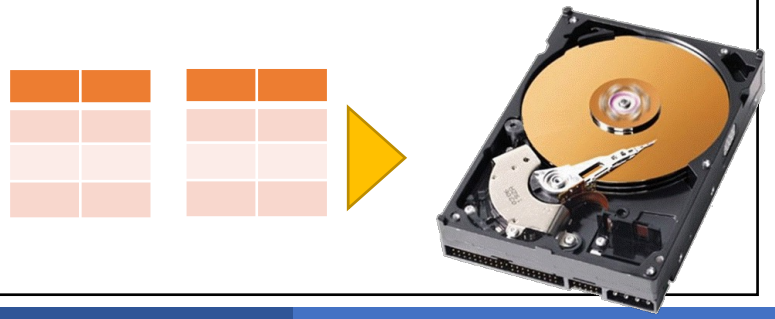
Next Lectures

Conceptual Schema
+ Normalization



Later…

Physical Schema
+ Partitioning
+ Indexing



ER Diagrams

Entity-Relationship (ER) Diagrams

- A visual way to describe the schema of a database

- Language independent: may implement in SQL, or some other data model

Example

Application to track the lifetime of products

- Keep information about Products: name, price, …
- Who manufactures them? Company name, address, their workers, …
- Who buys them? Customers with their names, …

Example: designing the Entity Sets

Example: designing the Entity Sets

Product

Company

Worker

Example: designing the Entity Sets

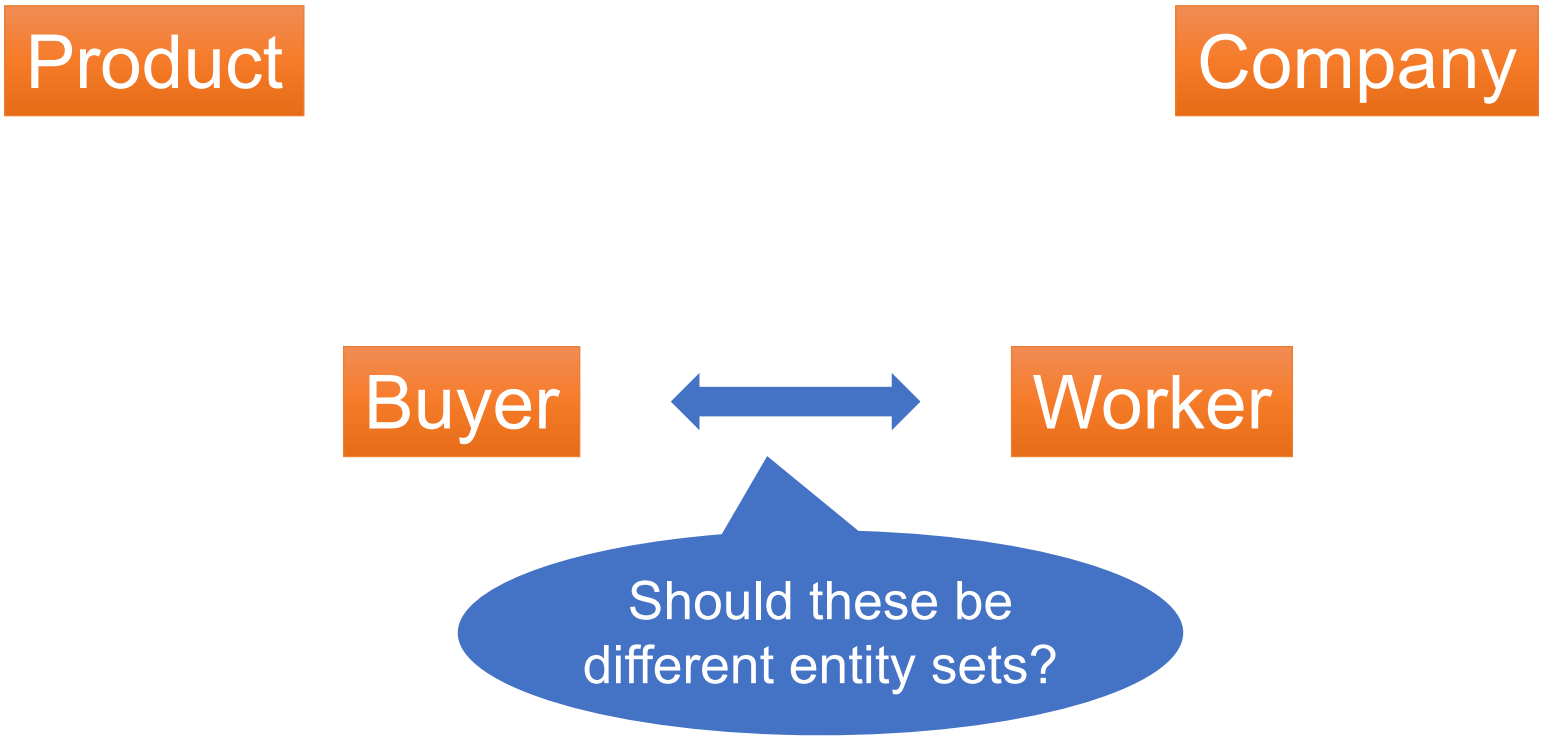
Product

Company

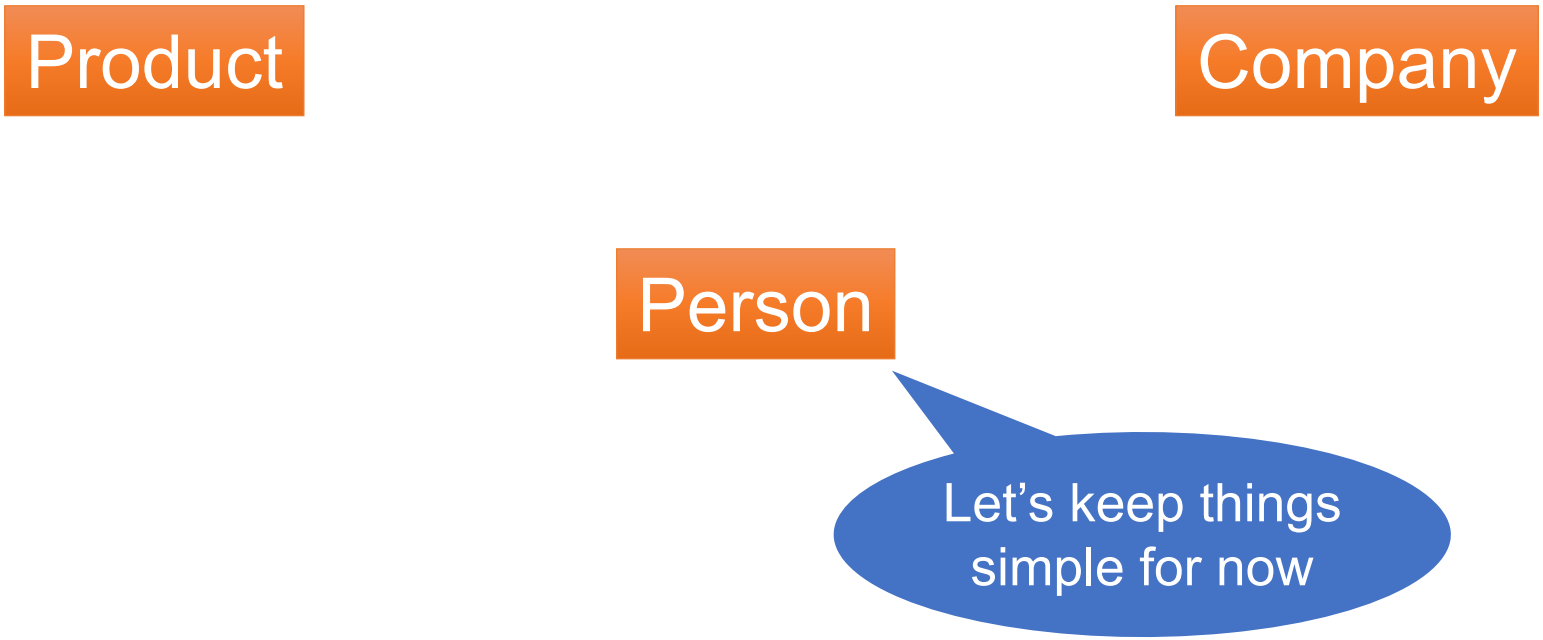
Buyer

Worker

Example: designing the Entity Sets



Example: designing the Entity Sets



Example: adding Attributes

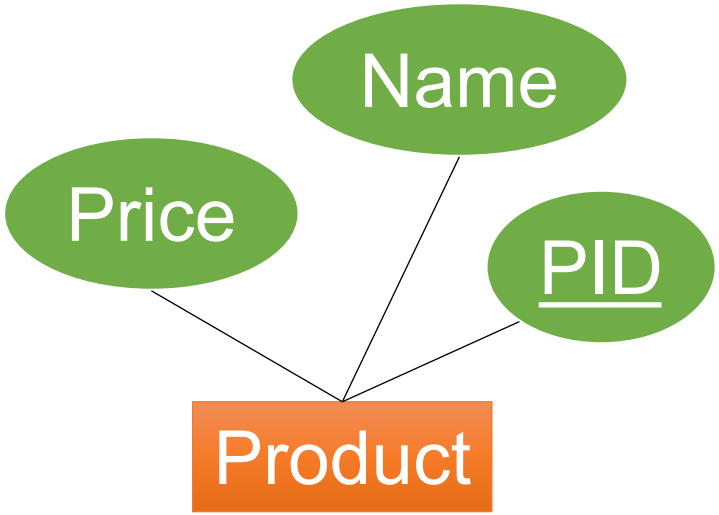
Next, let's design their attributes

Product

Company

Person

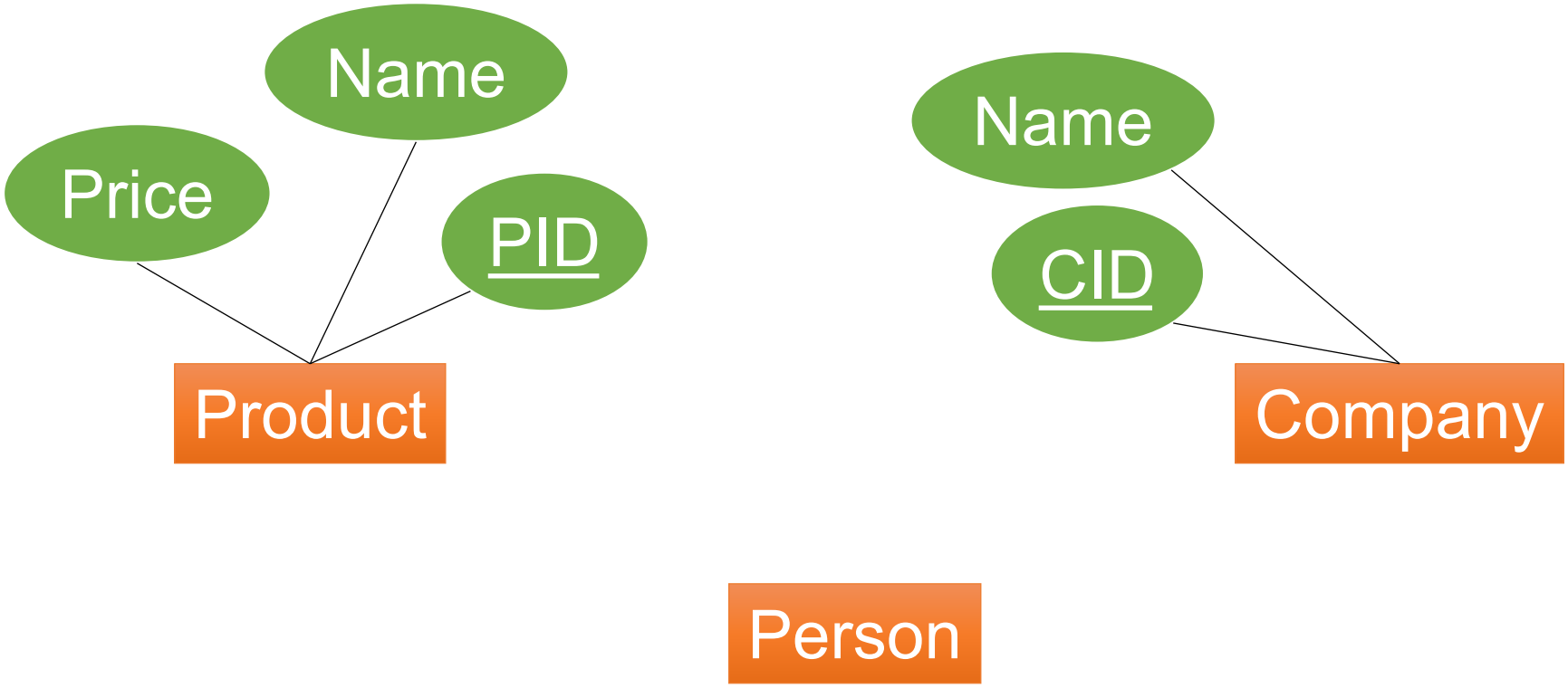
Example: adding Attributes



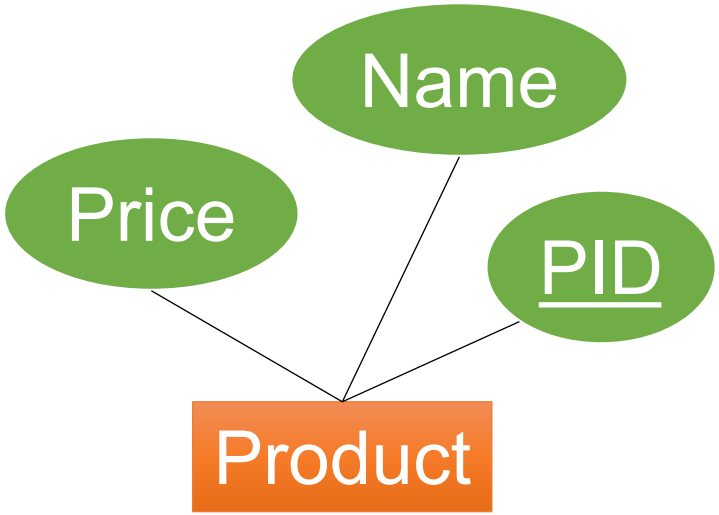
Company

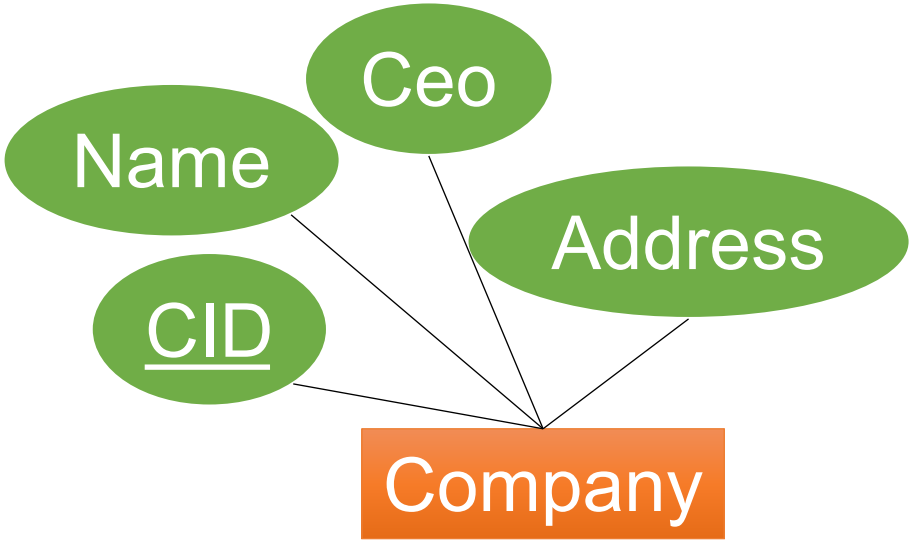
Person

Example: adding Attributes



Example: adding Attributes

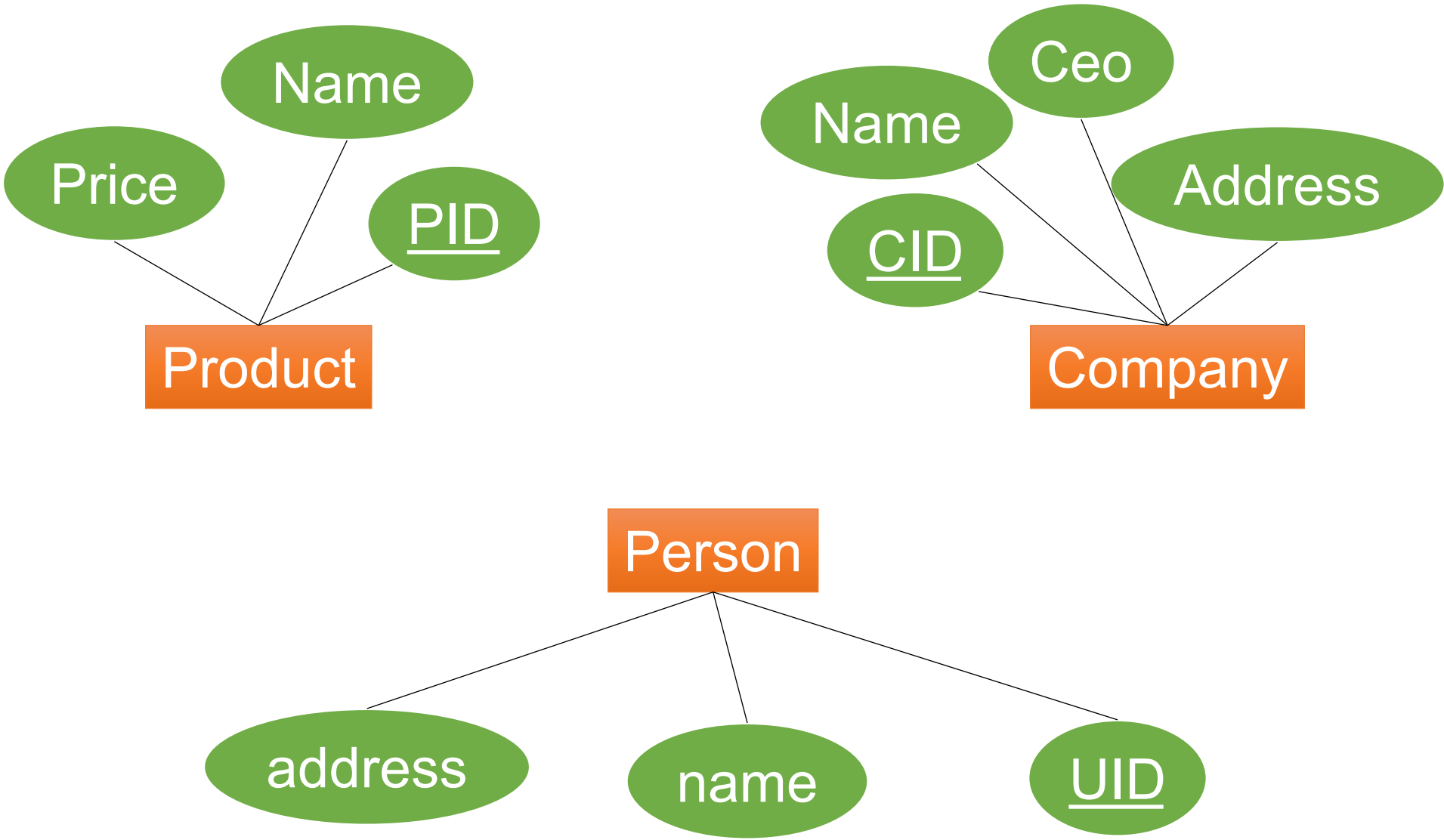




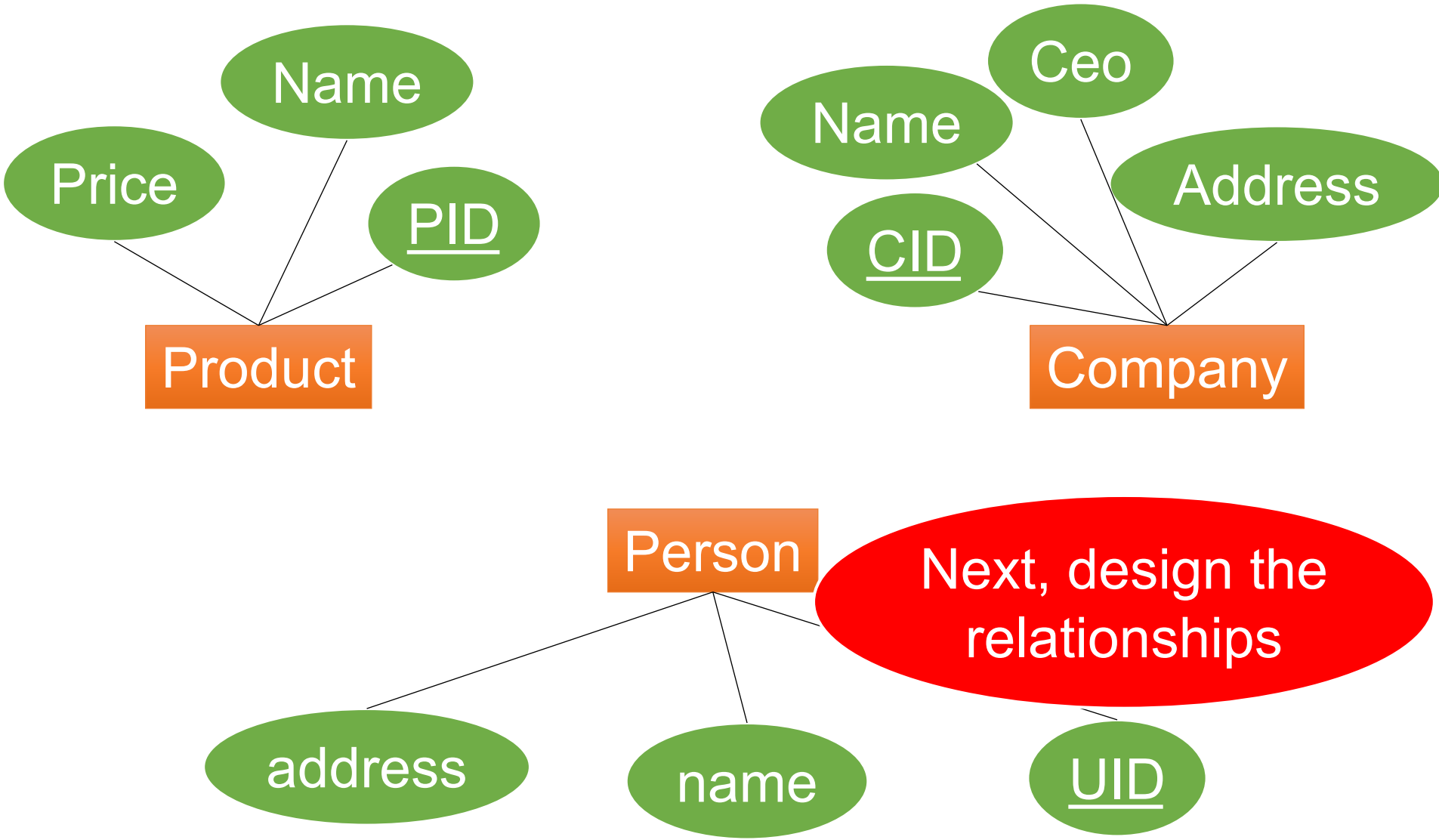
Person

Determine ALL attributes that your application needs

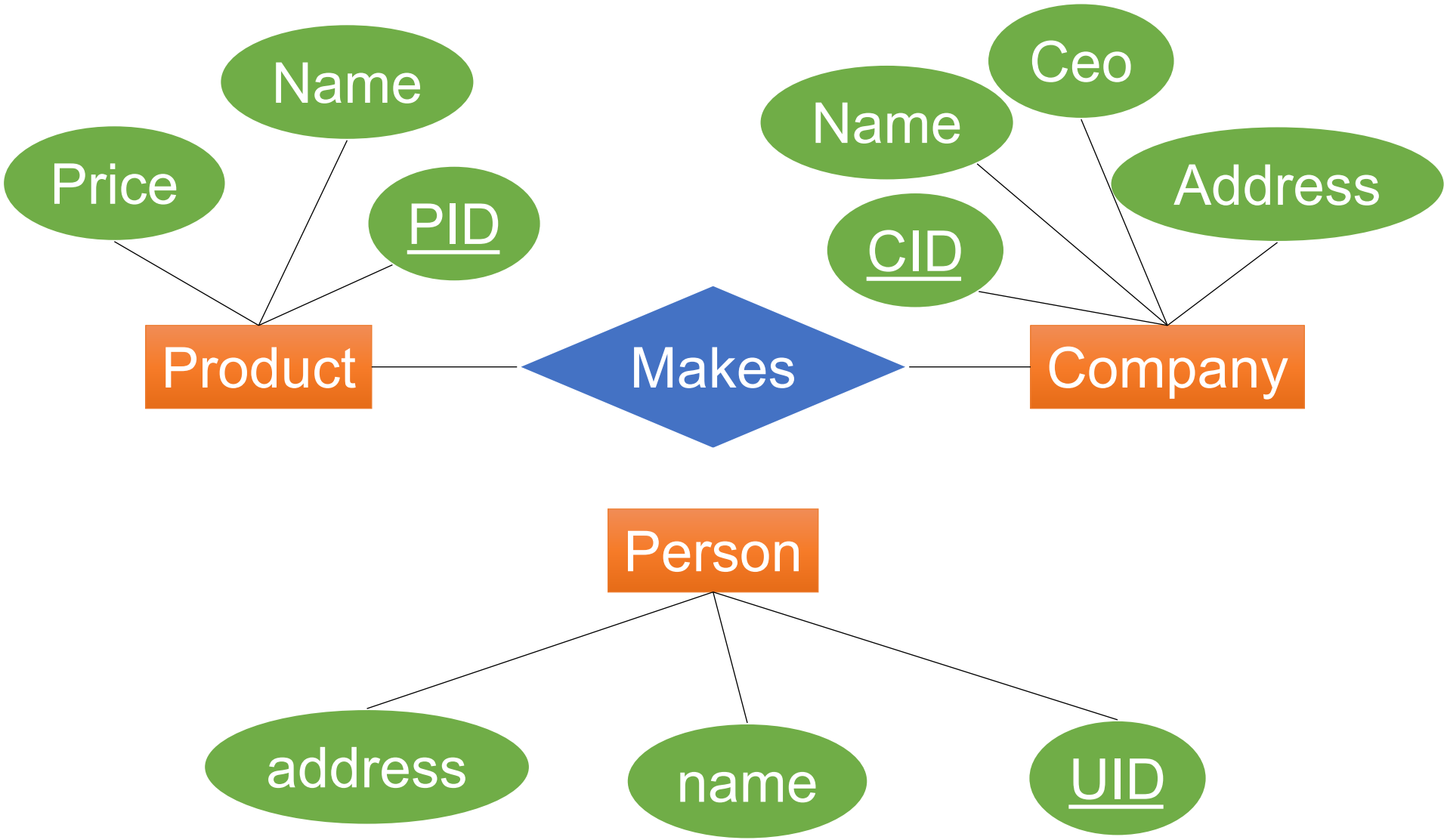
Example: adding Attributes



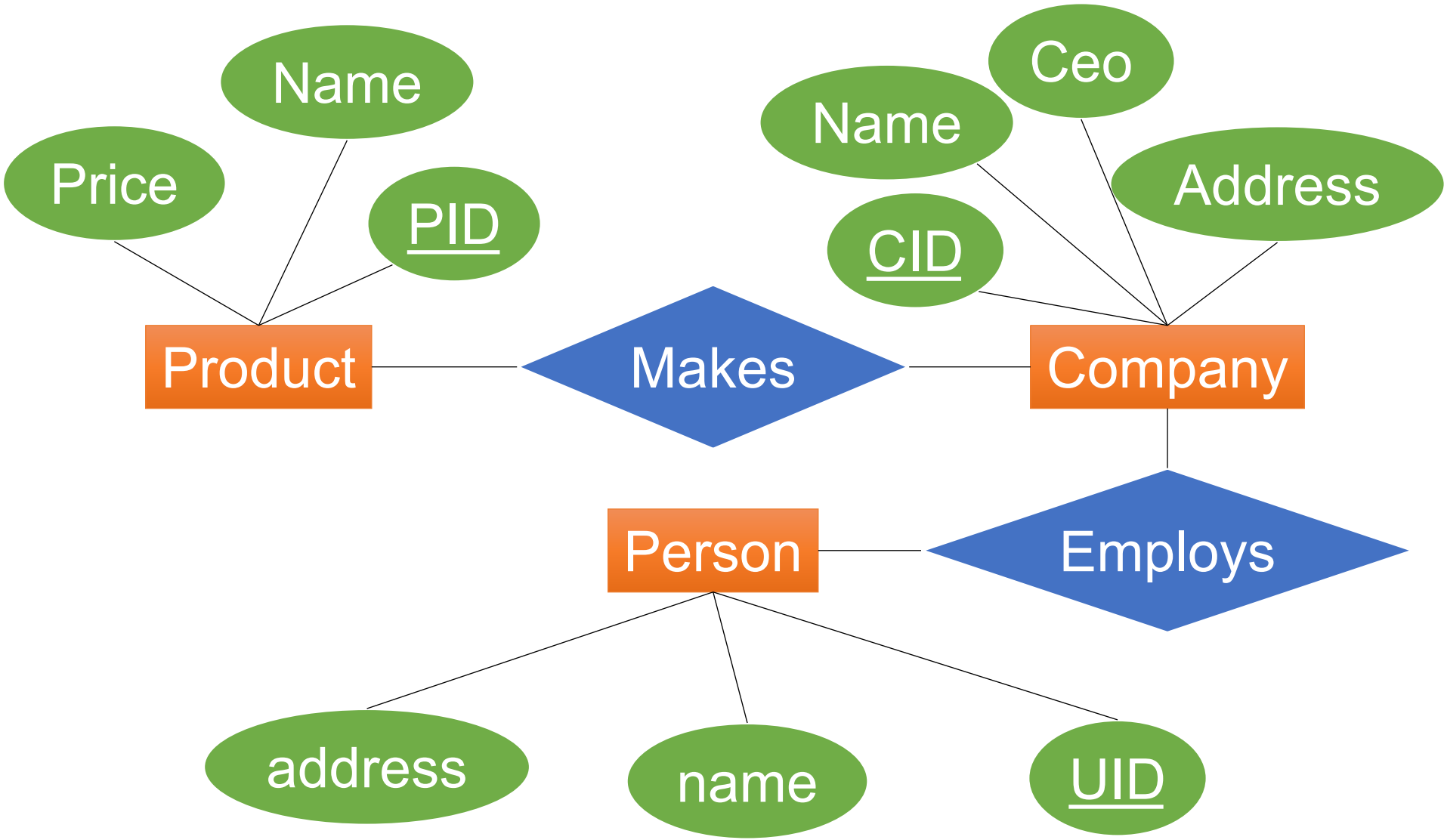
Example: adding Relationships



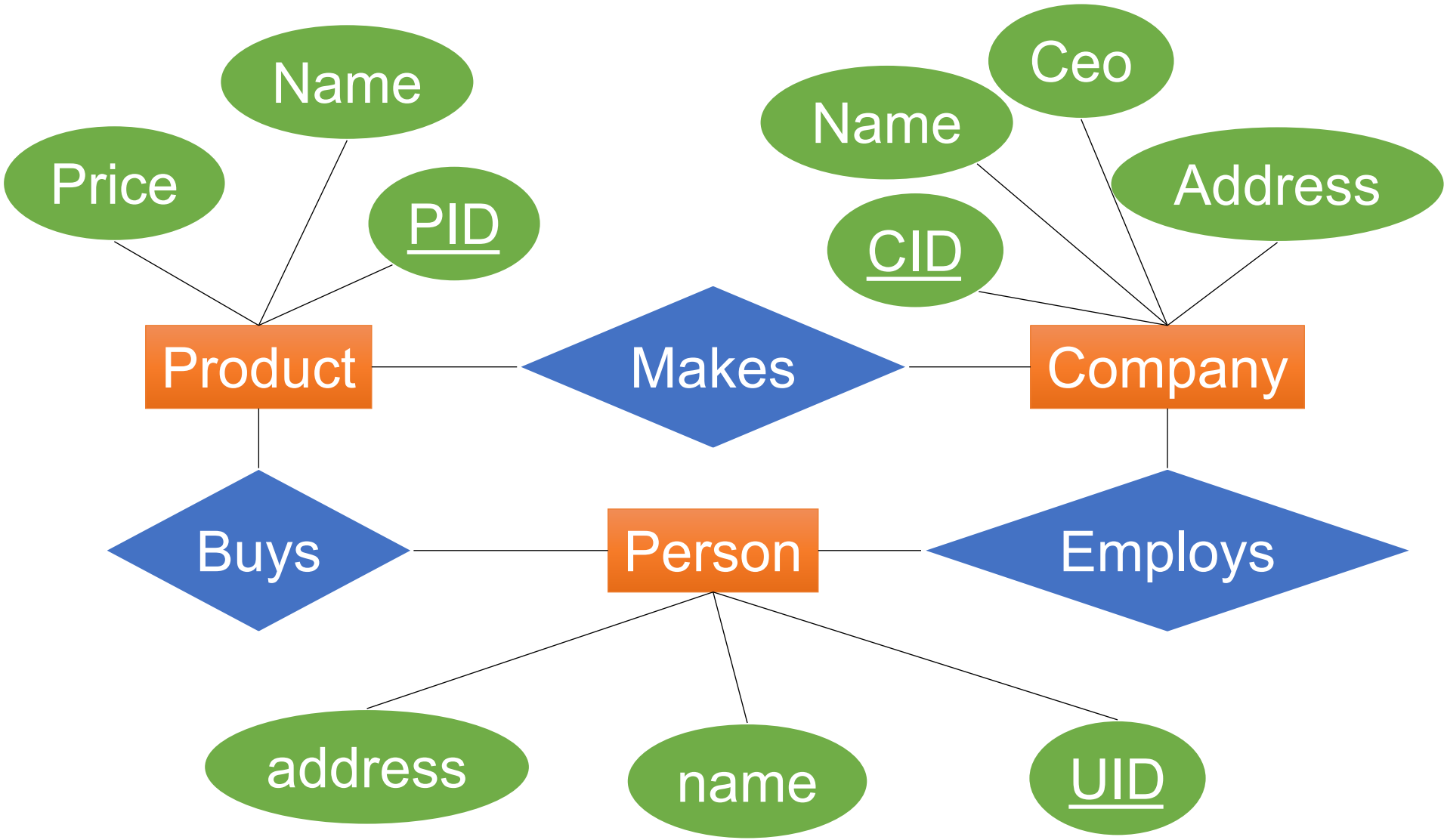
Example: adding Relationships



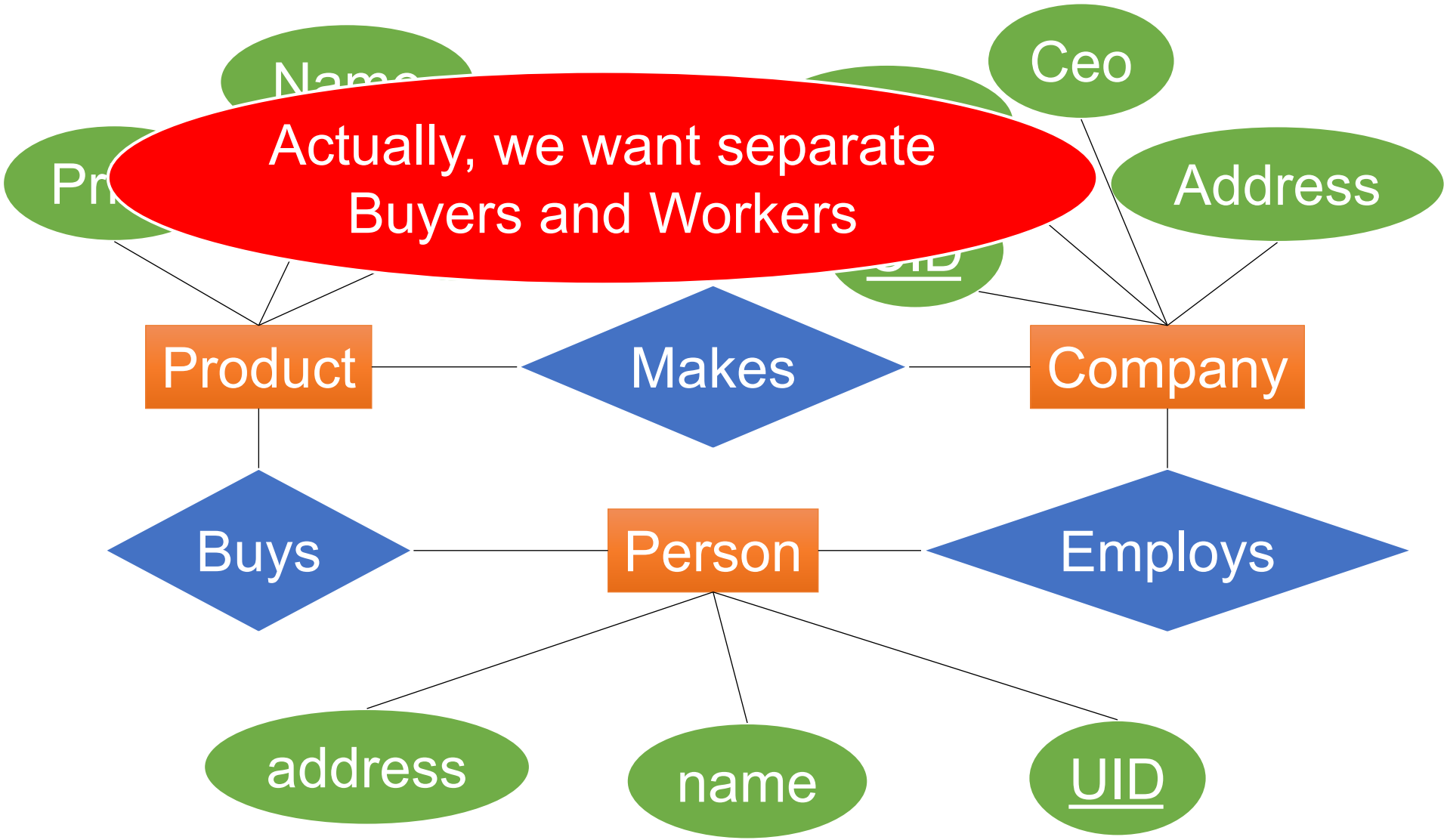
Example: adding Relationships



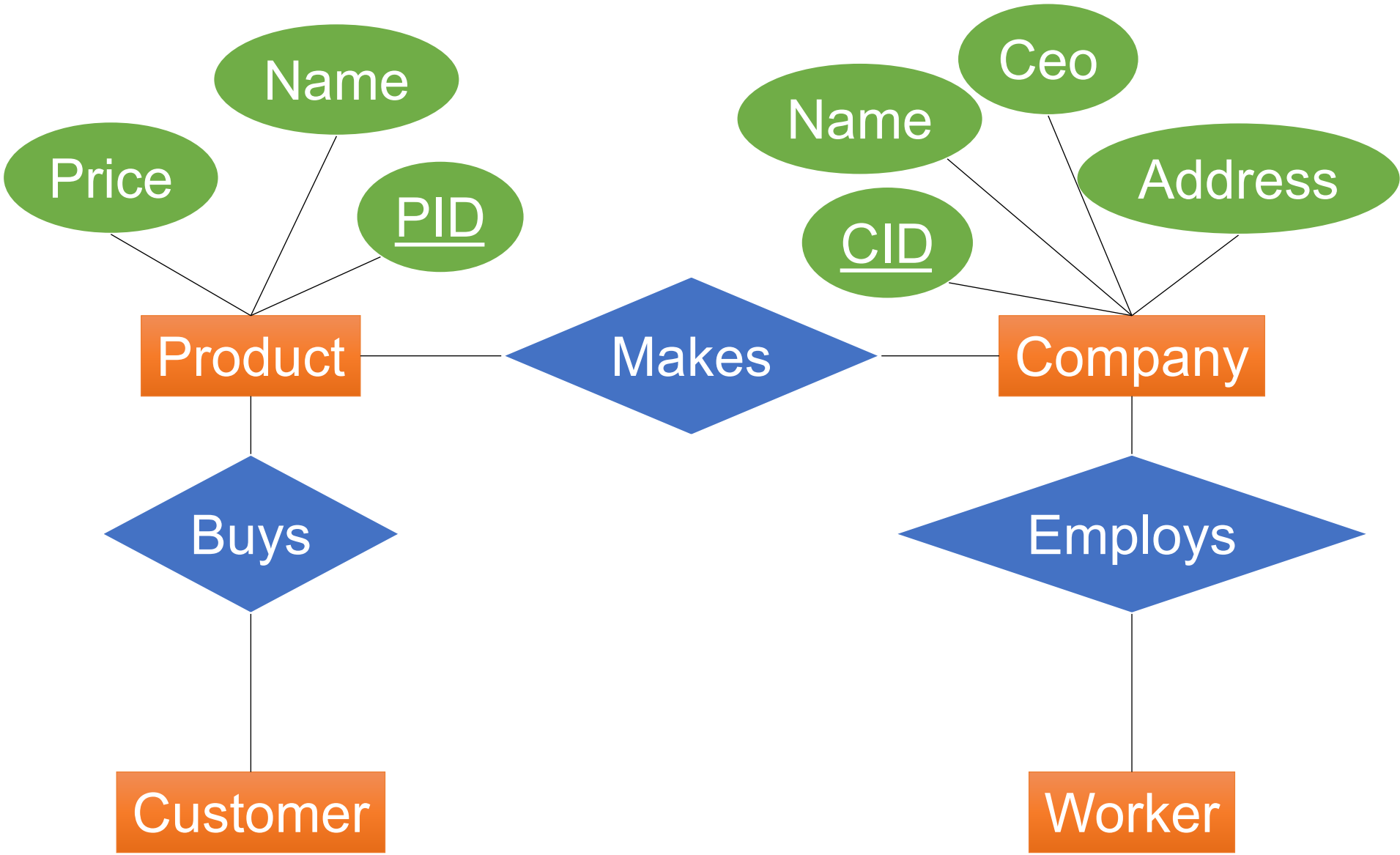
Example: adding Relationships



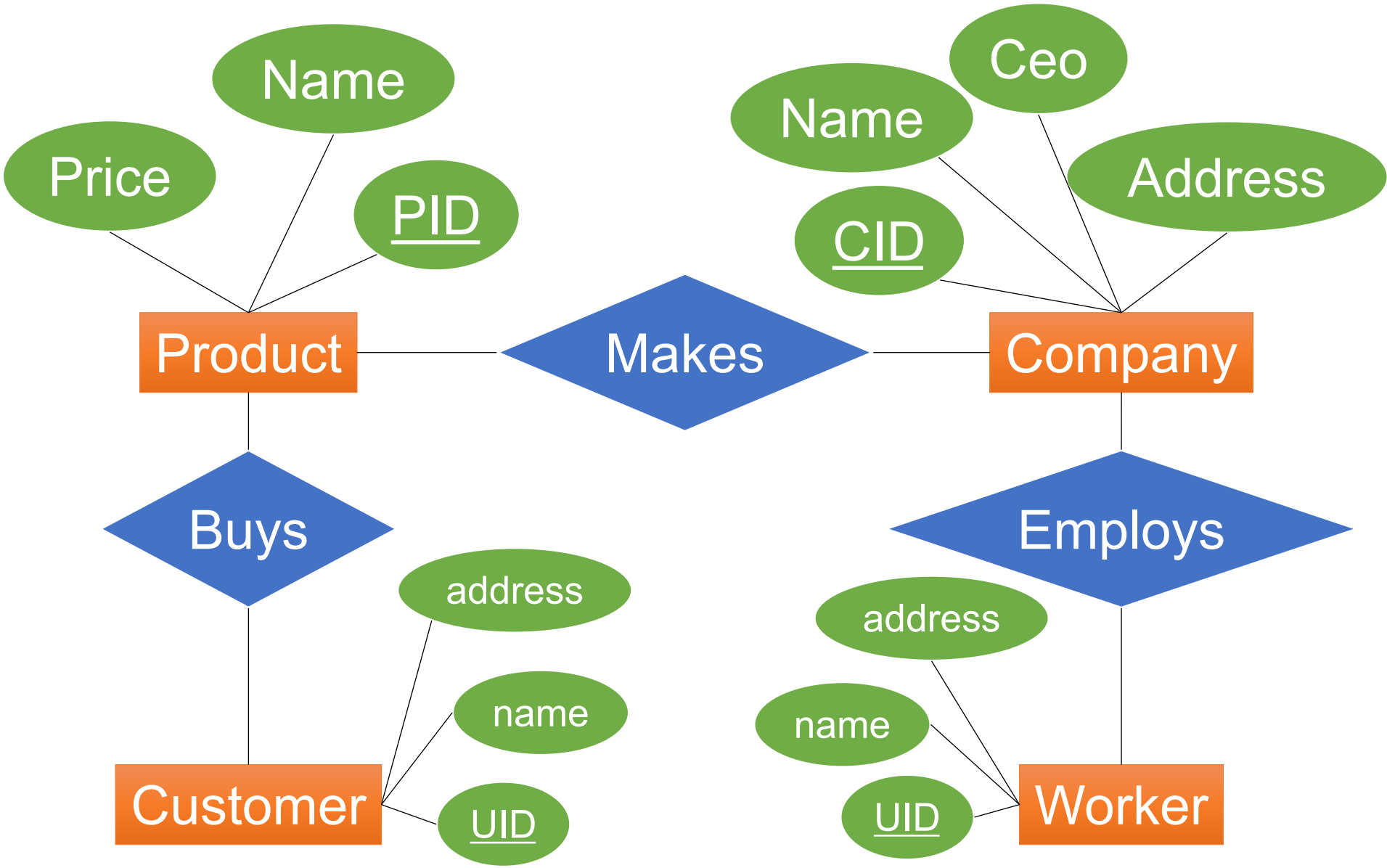
Example: Refining the Schema



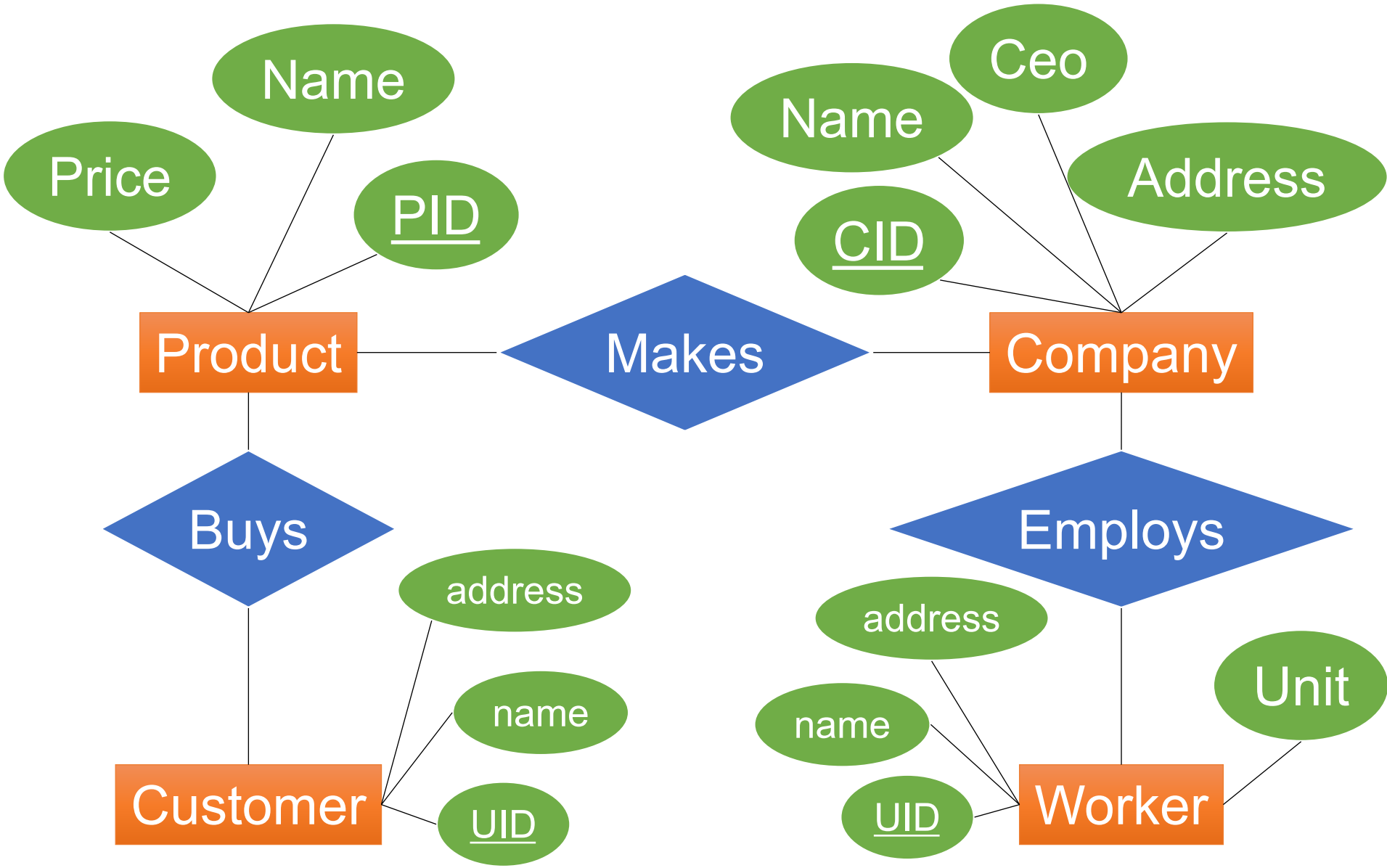
Example: Refining the Schema



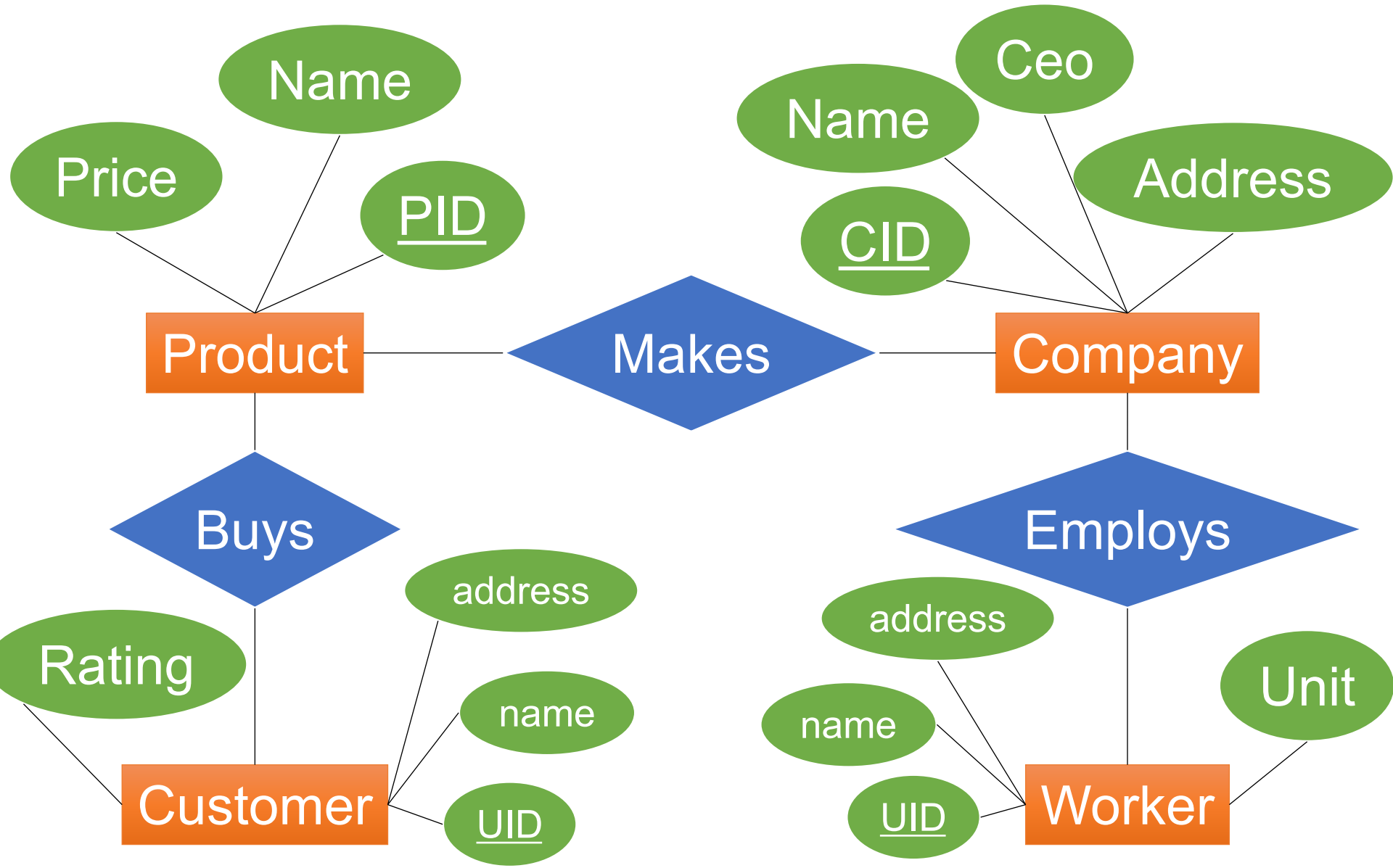
Example: Refining the Schema



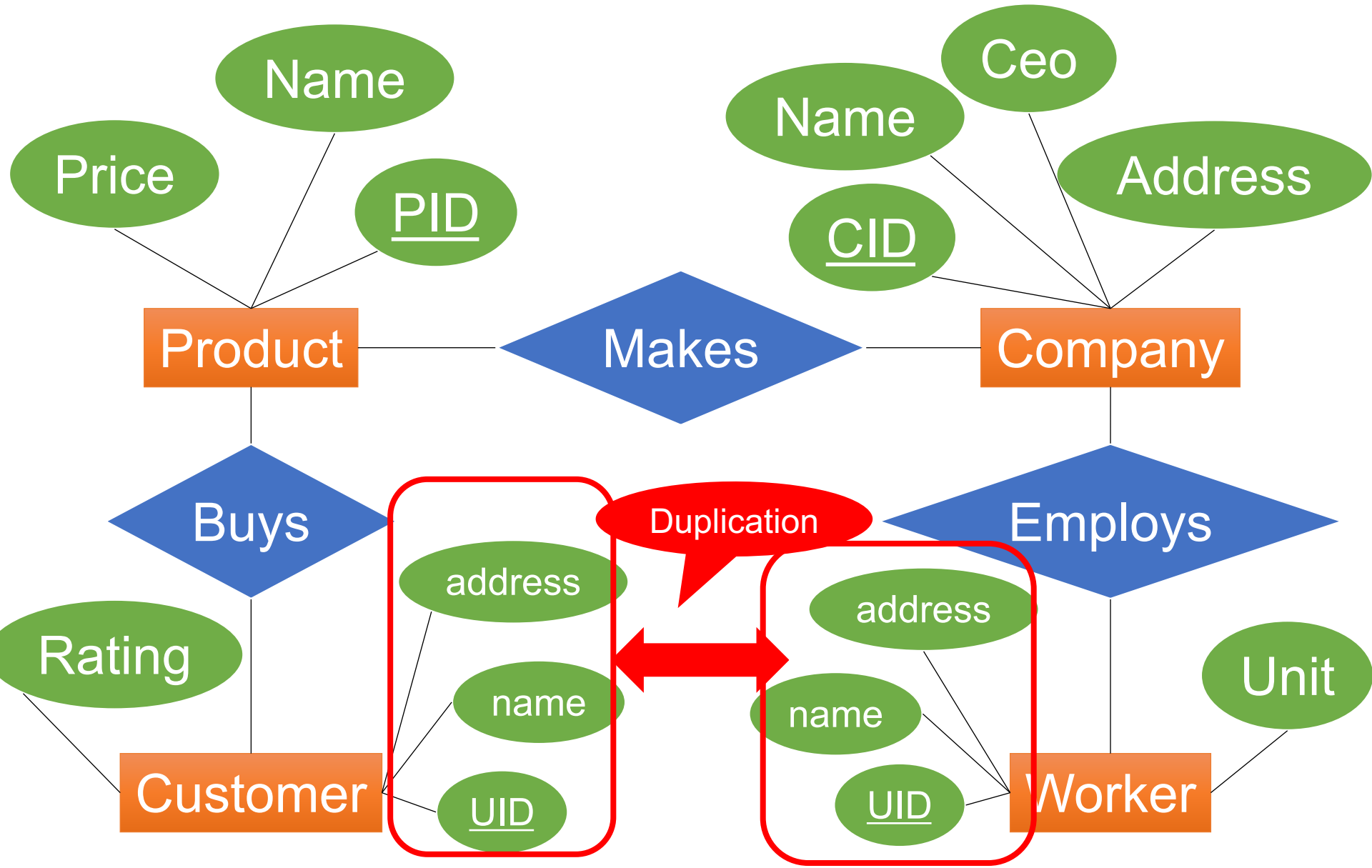
Example: Refining the Schema



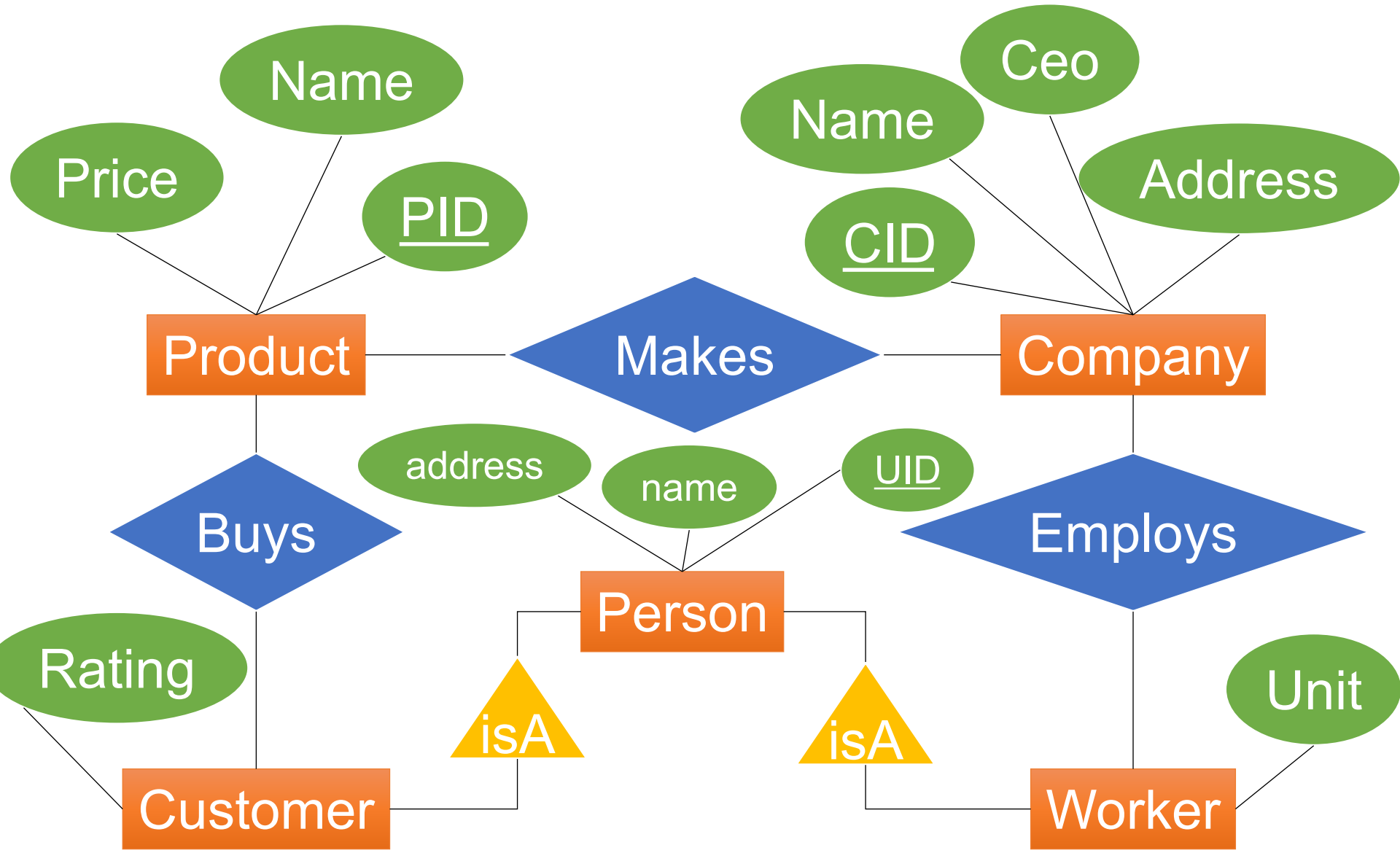
Example: Refining the Schema



Example: Refining the Schema



Example: Refining the Schema



Discussion

- ER diagram are easy to design,
  yet rigorous enough to convert to SQL

- Lots of ER diagram "dialects"
  - Textbook use rectangles/diamonds/ovals
  - Industry uses other standards

- In class we use the textbook version

Next lecture: E/R diagrams in detail