CSE544Database Management Systems

Lecture 4: Data Models

References

- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed.

Data Model Motivation

- Applications need to model real-world data
- User somehow needs to define data to be stored in DBMS
- Data model enables a user to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk

Outline

- Early data models
- Physical and logical independence in the relational model
- Conceptual design
- Data models that followed the relational model

Early Proposal 1: IMS*

- What is it?

* IBM Information Management System

Early Proposal 1: IMS*

- **Hierarchical data model**

- **Record**
  - **Type**: collection of named fields with data types
  - **Instance**: must match type definition
  - Each instance has a **key**
  - Record types arranged in a **tree**

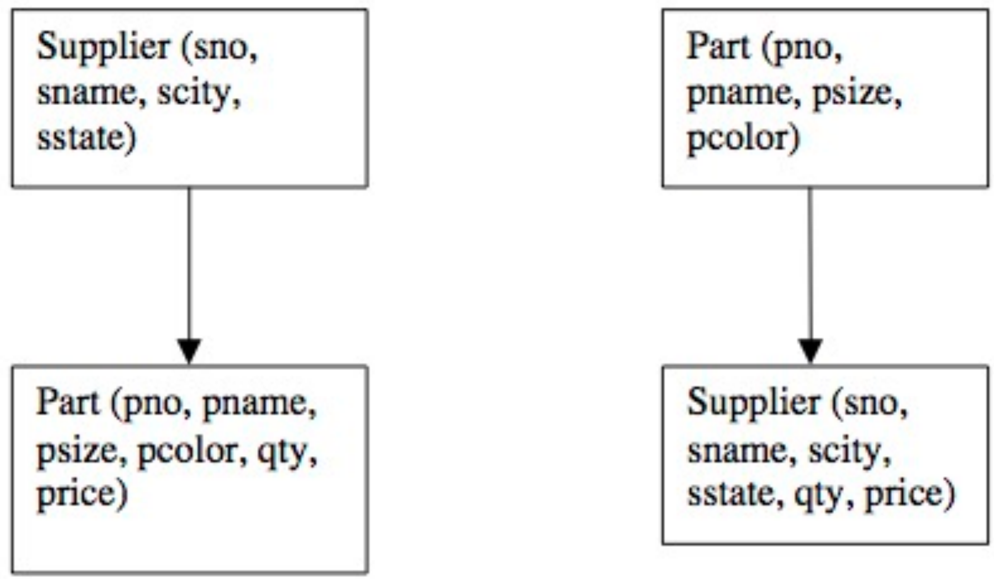
- **IMS database** is collection of instances of record types organized in a tree

* IBM Information Management System

IMS Example

- Figure 2 from “What goes around comes around”

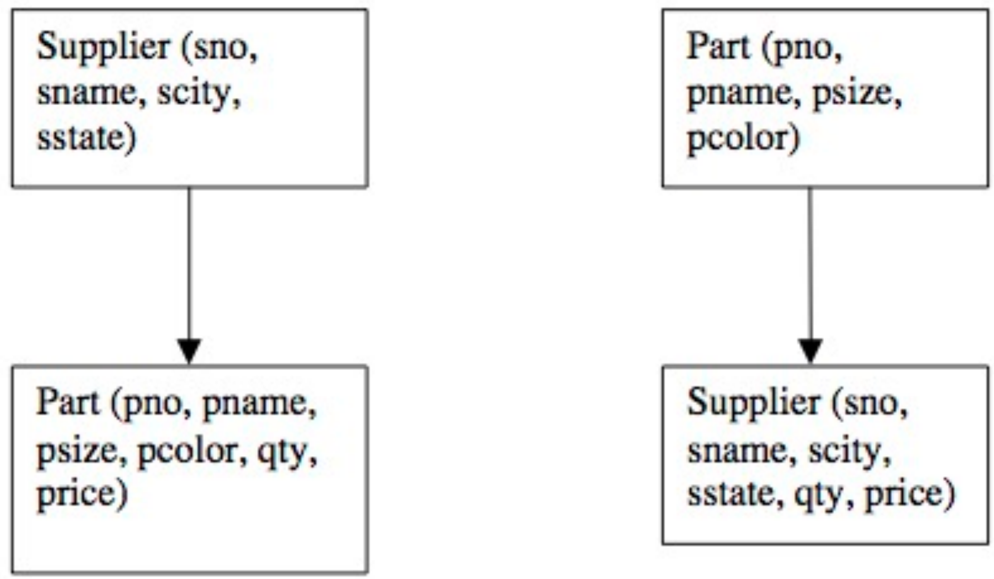
What does this mean?



IMS Example

- Figure 2 from “What goes around comes around”

What does this mean?

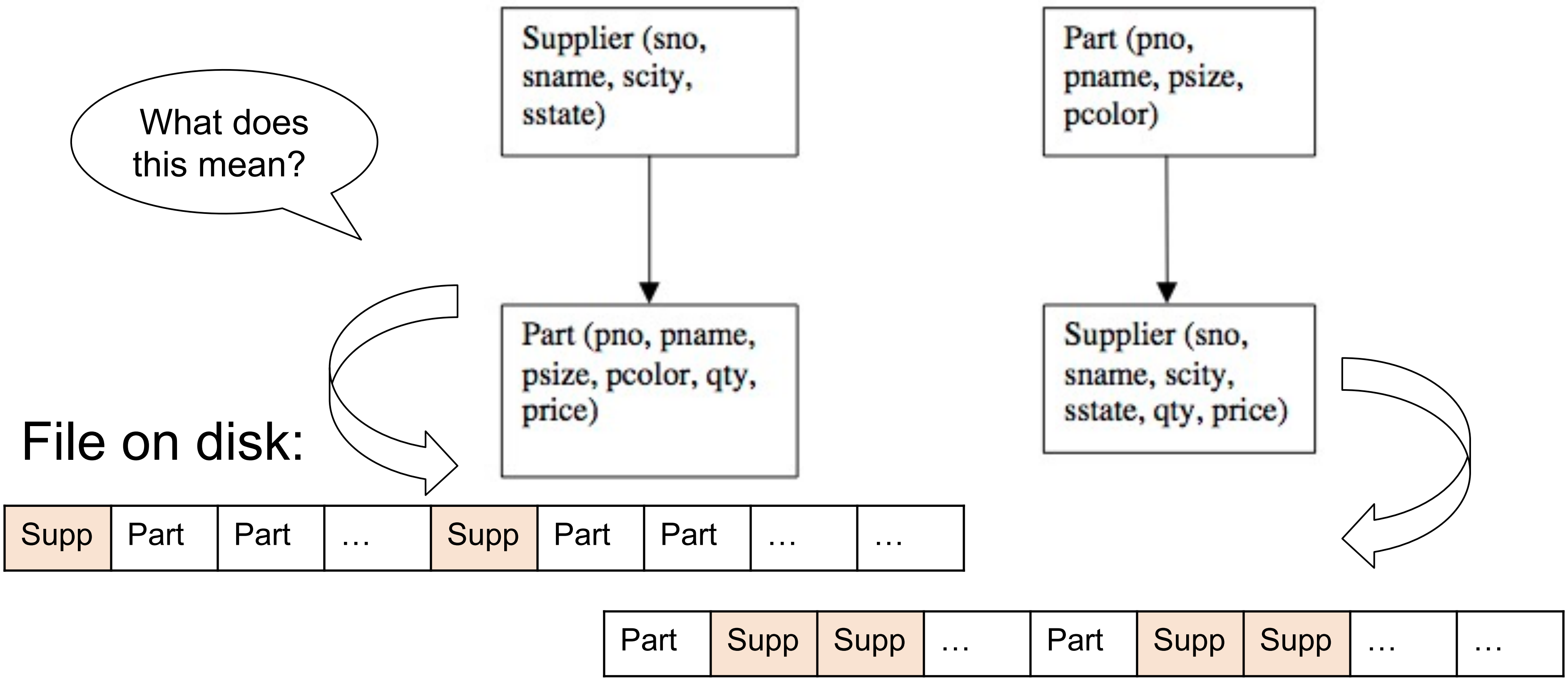


File on disk:

Supp	Part	Part	…	Supp	Part	Part	…	…

IMS Example

- Figure 2 from "What goes around comes around"



IMS Limitations

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent
- **Record-at-a-time** user interface
  - User must specify algorithm to access data

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent
- **Record-at-a-time** user interface
  - User must specify algorithm to access data
- **Very limited physical independence**
  - Phys. organization limits possible operations
  - Application programs break if organization changes
- **Some logical independence but limited**

Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

- Each record has a hierarchical sequence key (HSK)
- HSK defines semantics of commands:
  - get_next; get_next_within_parent
- **DL/1 is a record-at-a-time language**
  - Programmers construct algorithm, worry about optimization

Data storage

How is data physically stored in IMS?

Data storage

How is data physically stored in IMS?

- Root records
  - Stored sequentially (sorted on key)
  - Indexed in a B-tree using the key of the record
  - Hashed using the key of the record
- Dependent records
  - Physically sequential
  - Various forms of pointers
- Selected organizations restrict DL/1 commands
  - No updates allowed due to sequential organization
  - No “get-next” for hashed organization

Data Independence

What is it?

Data Independence

What is it?

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**

Lessons from IMS

- Physical/logical data independence needed
- Tree structure model is restrictive
- Record-at-a-time programming forces user to do optimization

Early Proposal 2: CODASYL

What is it?

Early Proposal 2: CODASYL

What is it?

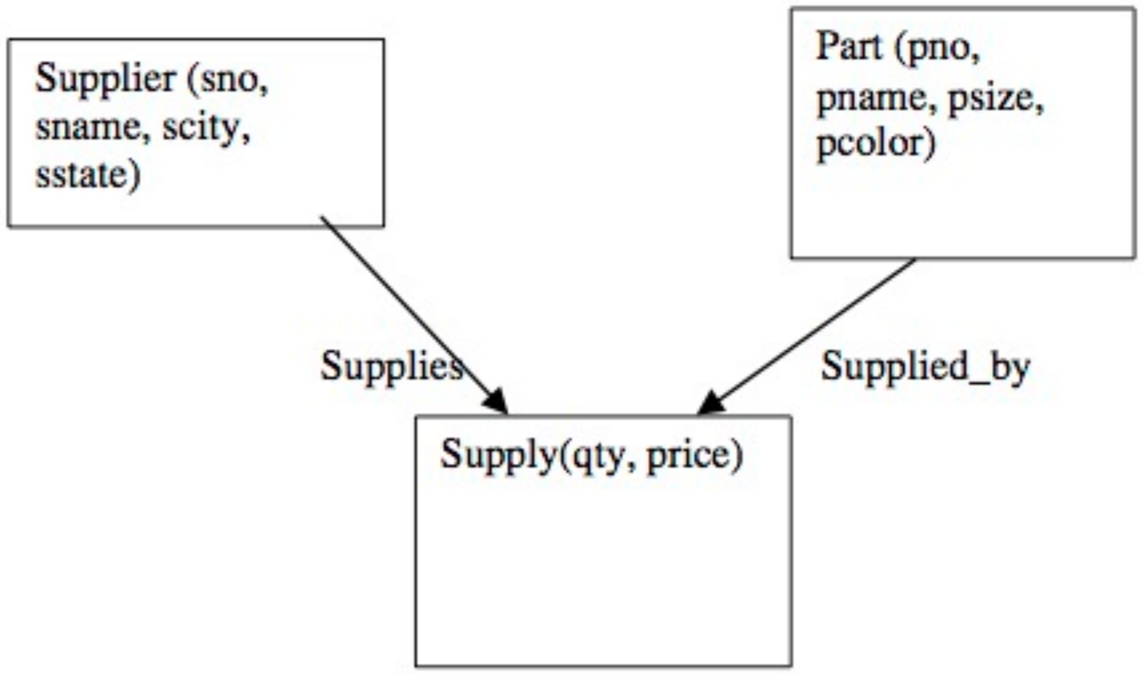
- **Networked data model**

- Primitives are also **record types** with **keys**
- Record types are organized into **network**
- Multiple parents; arcs = "sets"
- More flexible than hierarchy

- **Record-at-a-time** data manipulation language

CODASYL Example

- Figure 5 from "What goes around comes around"



CODASYL Limitations

- No data independence: application programs break if organization changes
- Record-at-a-time: “navigate the hyperspace”

The Programmer as Navigator

by Charles W. Bachman

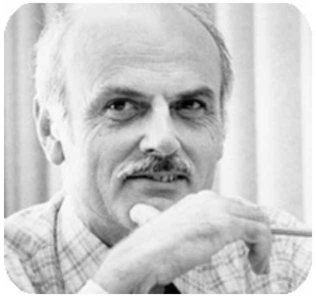
Outline

- Early data models
- Physical and logical independence in the relational model
- Conceptual design
- Data models that followed the relational model

Relational Model Overview

Ted Codd 1970

- What was the motivation? What is the model?



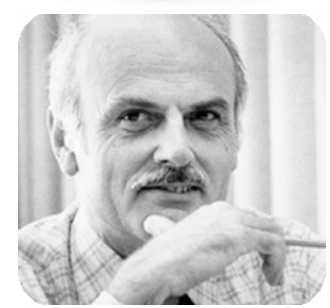
Relational Model Overview

Ted Codd 1970

- Motivation: logical and physical data independence

- Store data in a **simple data structure** (table)
- Access data through **set-at-a-time** language
- **No physical storage proposal**

Relational Database: A Practical Foundation for Productivity



Great Debate

- Pro relational
  - What were the arguments?
- Against relational
  - What were the arguments?
- How was it settled?

Great Debate

- Pro relational
  - CODASYL is too complex
  - No data independence
  - Record-at-a-time hard to optimize
  - Trees/networks not flexible enough

- Against relational
  - COBOL programmers cannot understand relational languages
  - Impossible to implement efficiently

- Ultimately settled by the market place

Data Independence

How it is achieved today:

- Physical independence: SQL to Plan
- Logical independence: Views in SQL

Physical Data Independence

- In SQL we express *What* data we want to retrieve
- The optimizers figures out *How* to retrieve it

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Query Plan

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
          x.price > 100 and z.city = ‘Seattle’
```

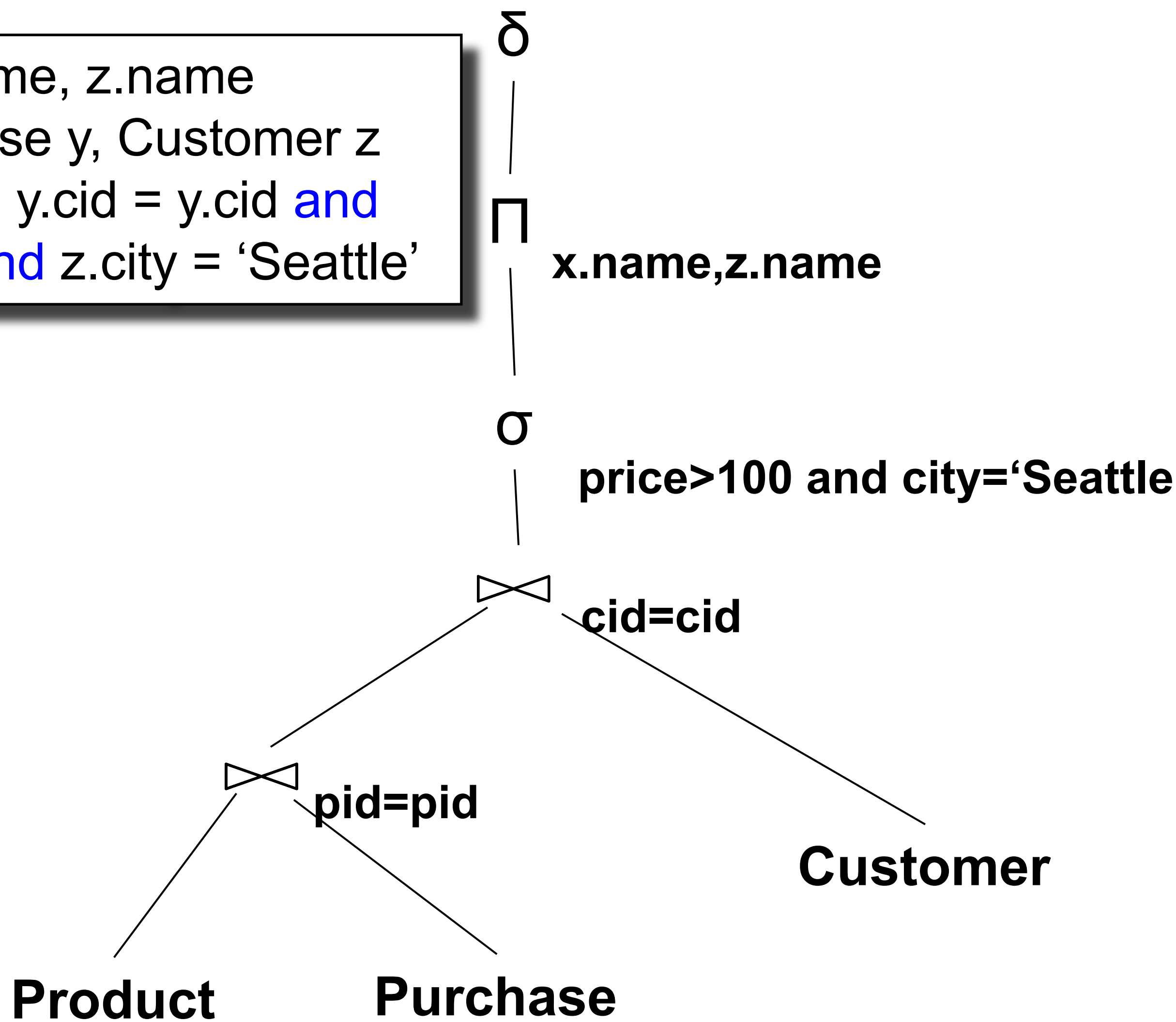
We say *What*
we want

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Logical Query Plan

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
         x.price > 100 and z.city = 'Seattle'
```

We say *What* we want

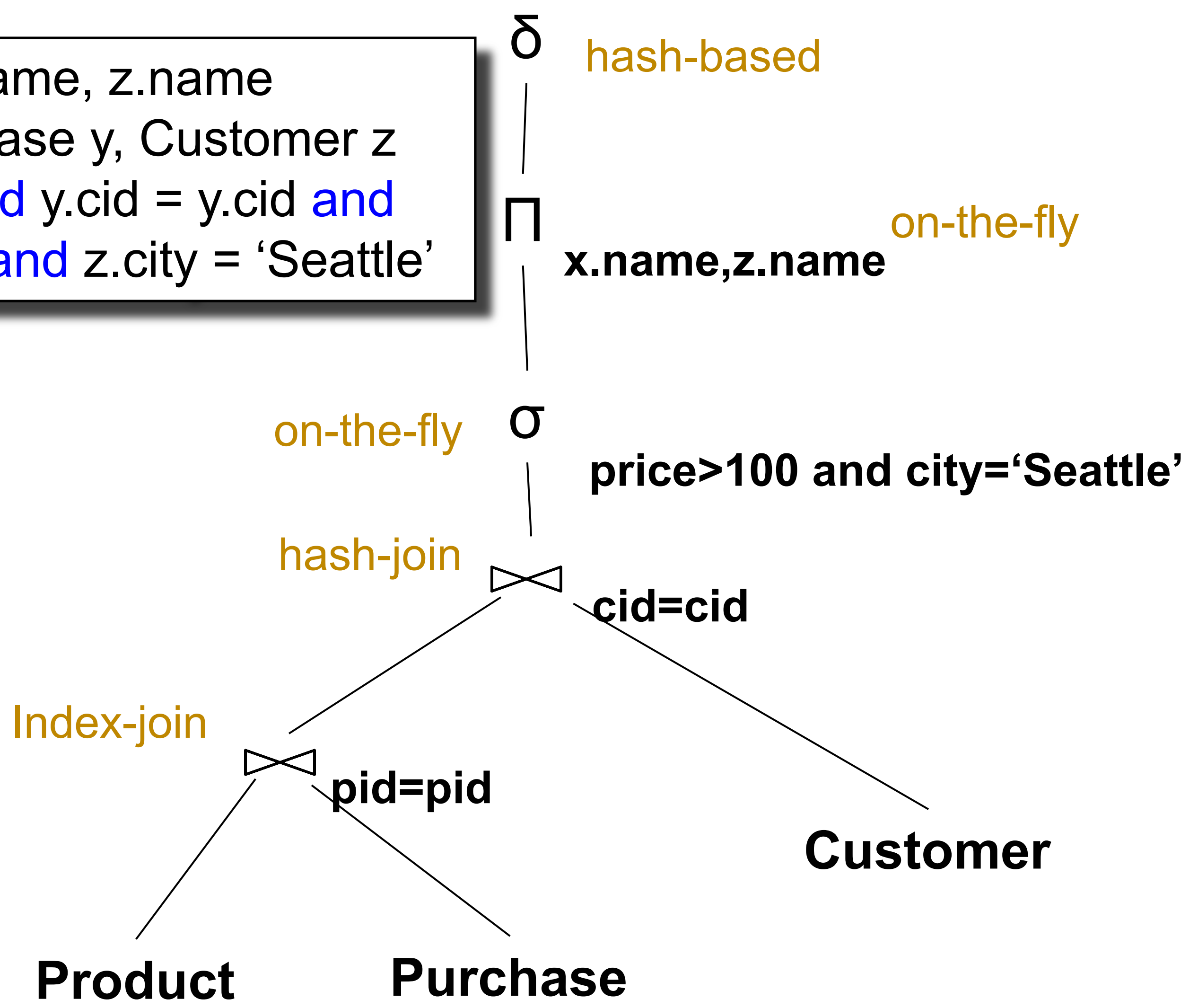


Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Physical Query Plan

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
          x.price > 100 and z.city = 'Seattle'
```

We say *What* we want



Says *How* to get it

Query Optimizer

- Rewrite one relational algebra expression to a better one

Logical Data Independence

A View is a Relation defined by a SQL query

It can be used as a normal relation

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

Virtual table: Big_Parts(pno,pname,psize,pcolor)

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

Virtual table: Big_Parts(pno,pname,psize,pcolor)

Querying the view:

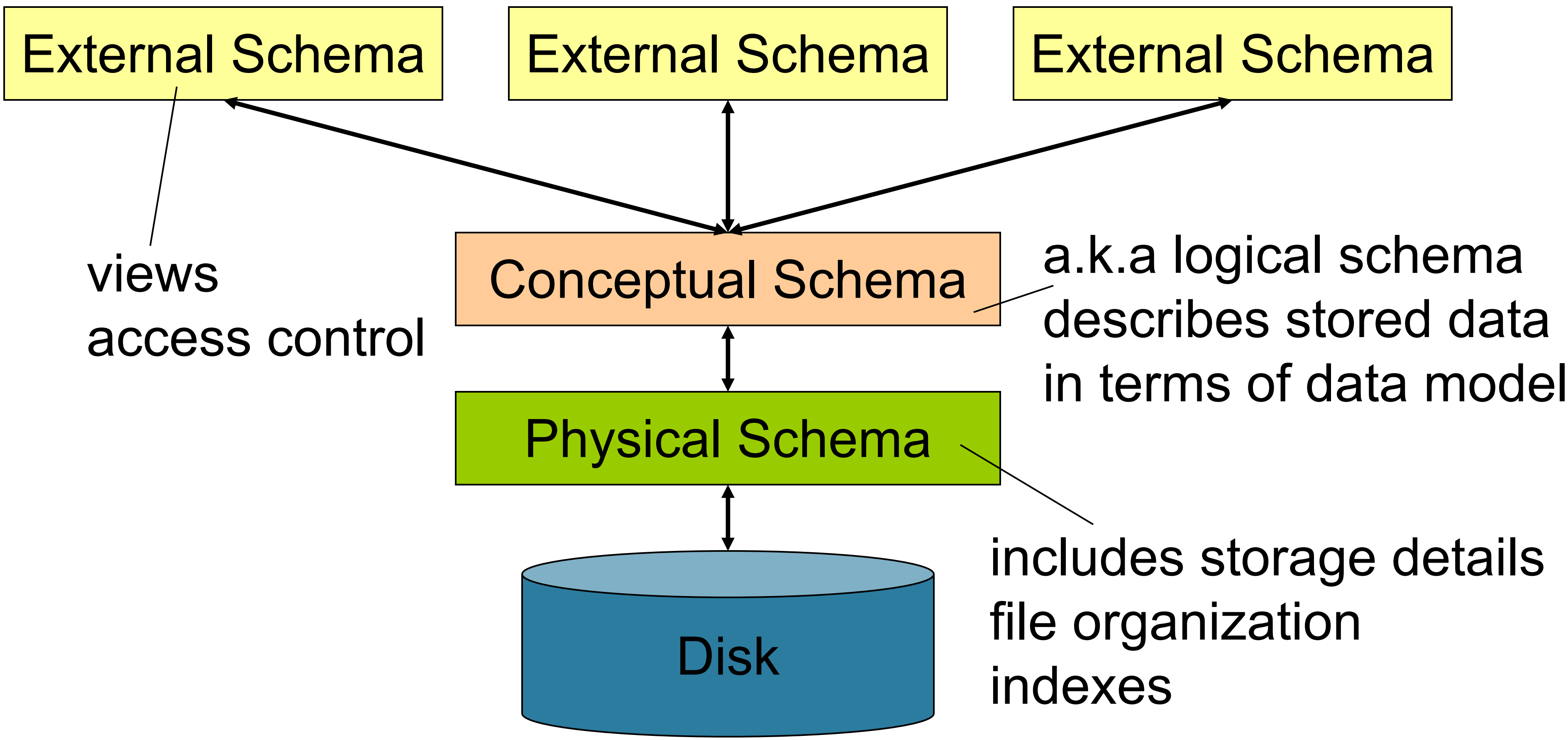
```
SELECT *
FROM Big_Parts
WHERE pcolor='blue';
```

Two Types of Views

- Virtual views:
  - Default in SQL, and what Stonebraker means in the paper
  - CREATE VIEW xyz AS …
  - Computed at query time

- Materialized views:
  - Some SQL engines support them
  - CREATE MATERIALIZED VIEW xyz AS
  - Computed at definition time

Levels of Abstraction



Recap: Data Independence

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

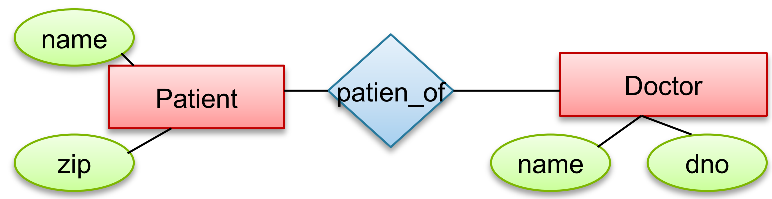
- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**

Outline

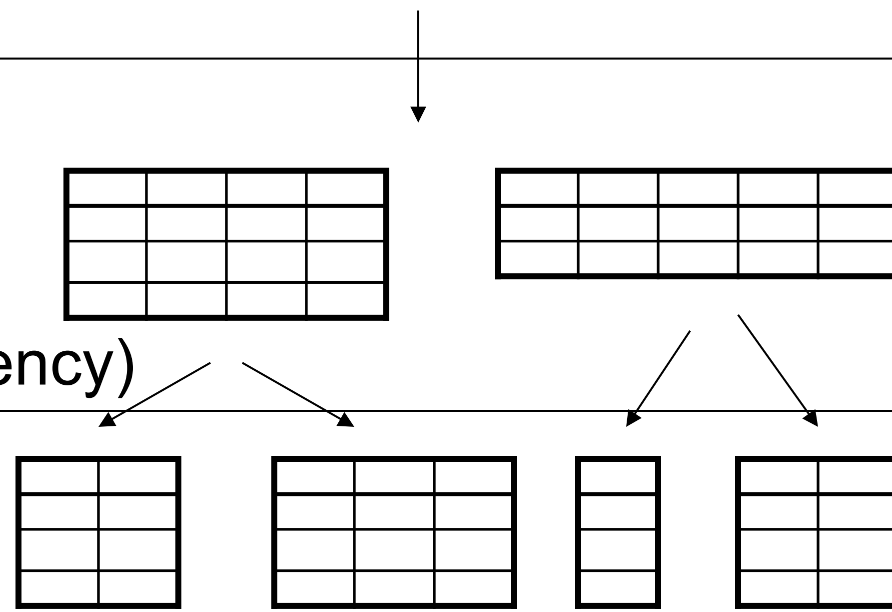
- Early data models
- Physical and logical independence in the relational model
- Conceptual design
- Data models that followed the relational model

Conceptual Schema Design

Conceptual Model:



Relational Model:
plus FD's
(FD = functional dependency)



Normalization:
Eliminates anomalies

Entity-Relationship Diagram

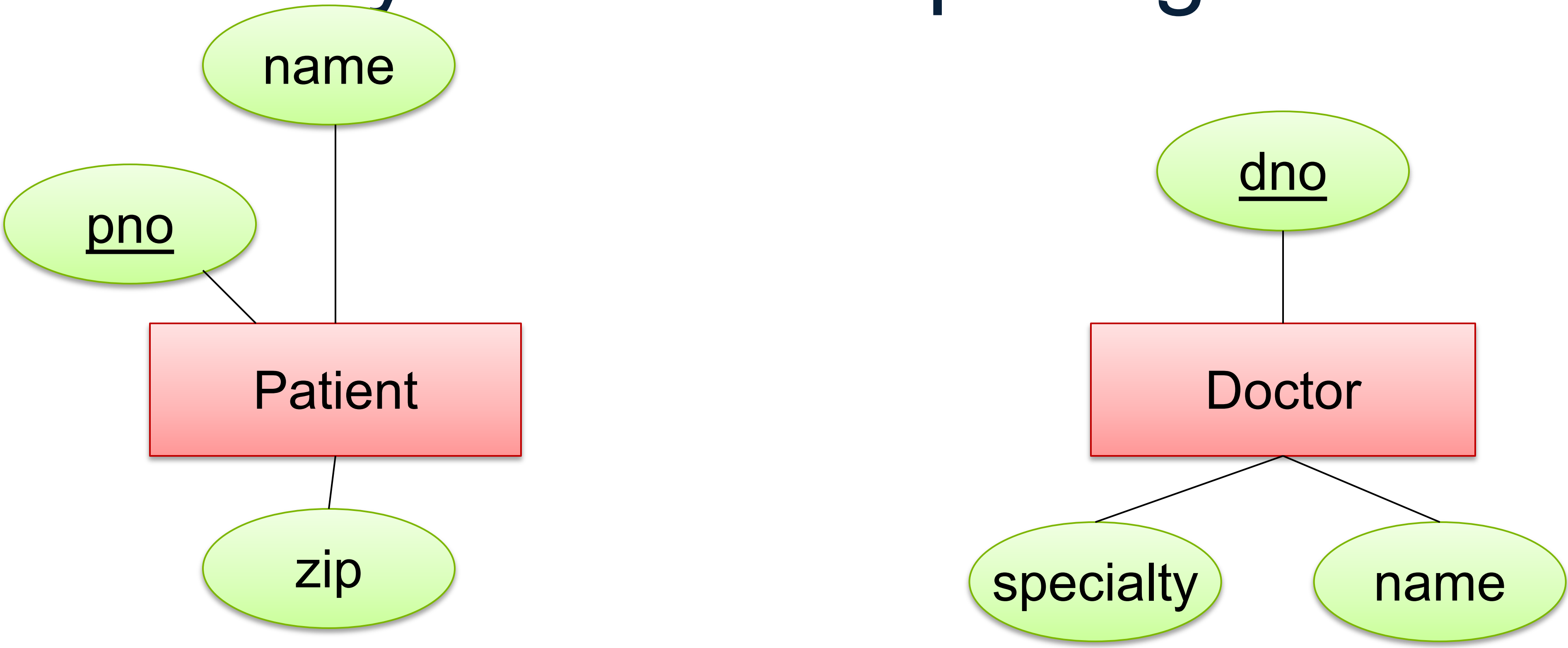
Attributes

Entity sets

Relationship sets

Entity-Relationship Diagram

Entity-Relationship Diagram

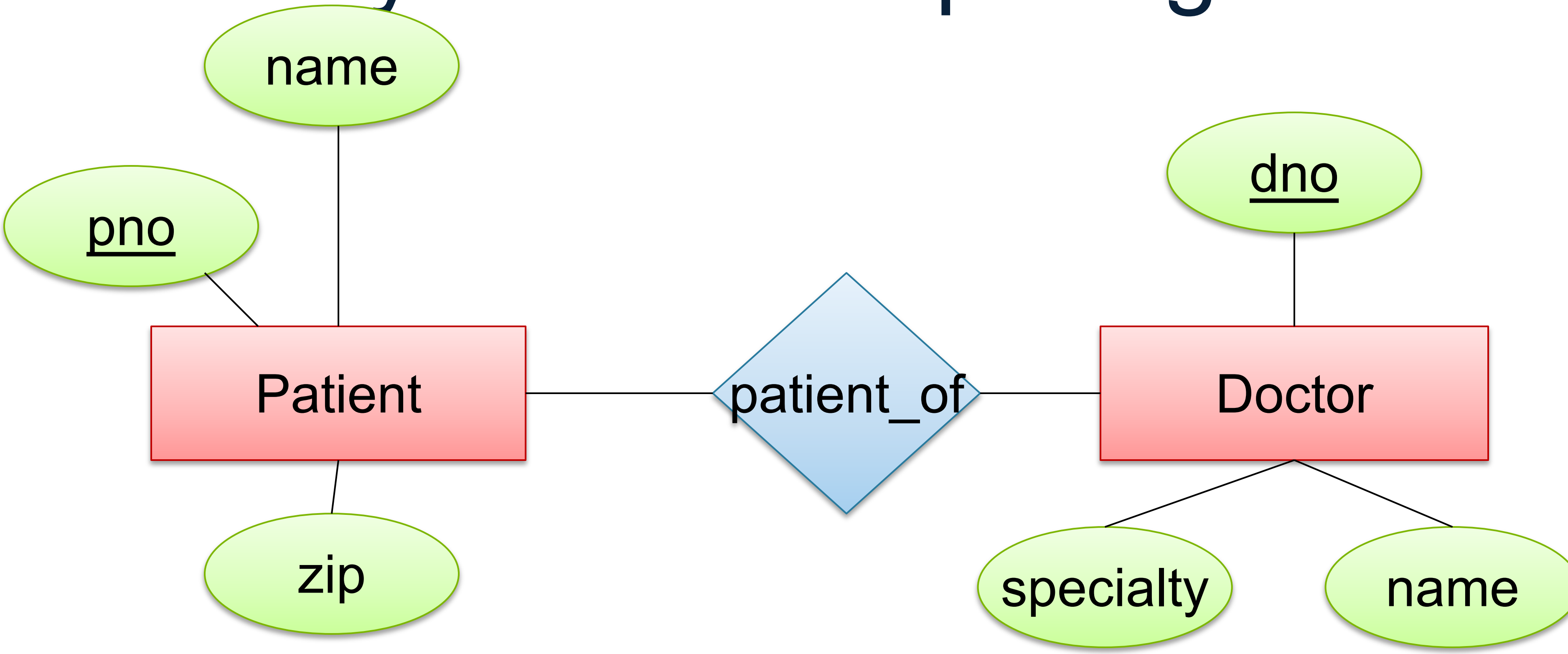


Attributes

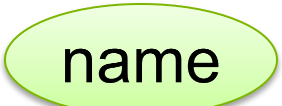
Entity sets

Relationship sets

Entity-Relationship Diagram

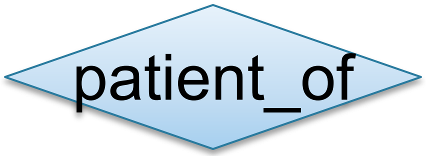


Attributes

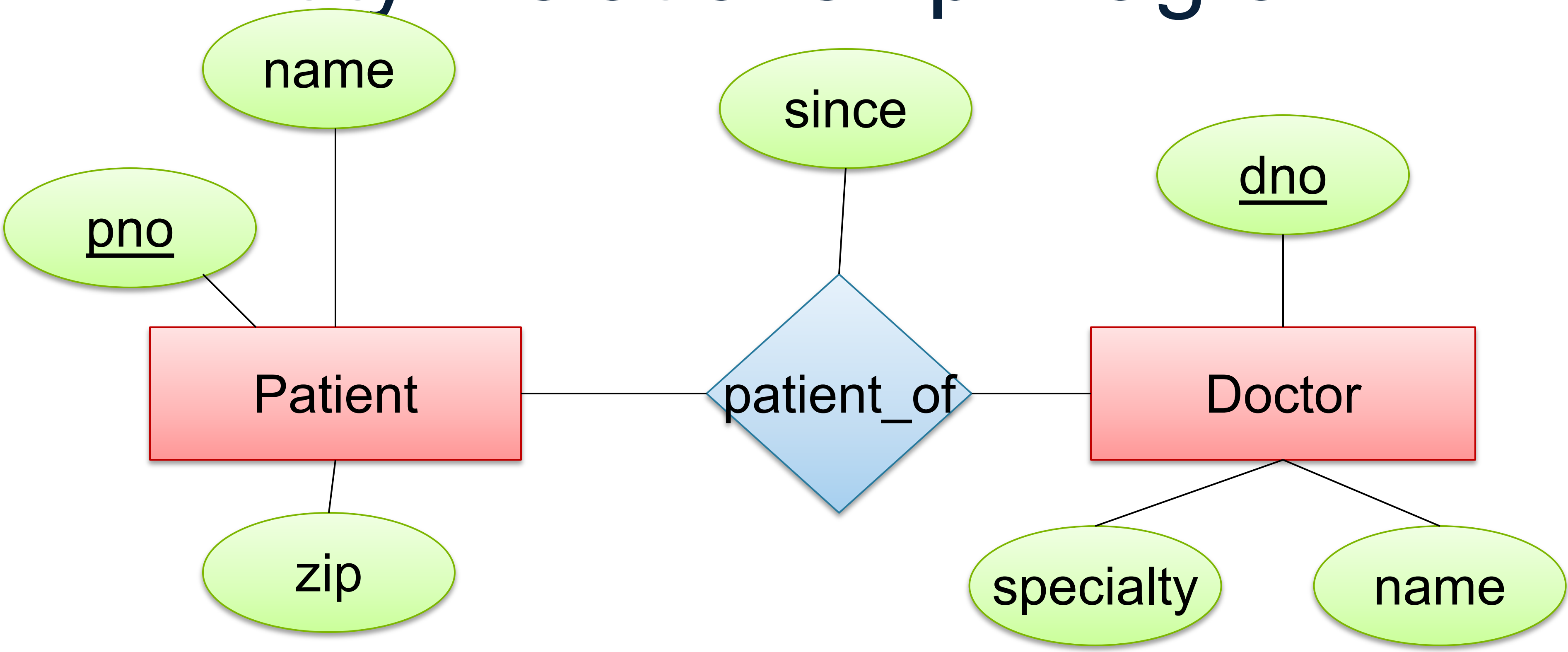


Entity sets

Relationship sets

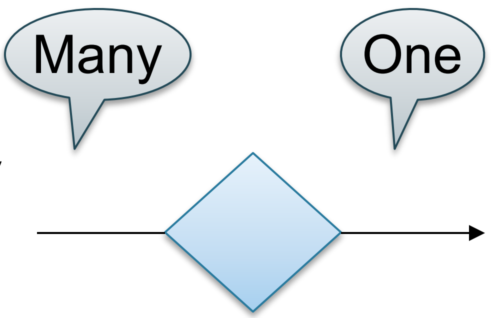


Entity-Relationship Diagram

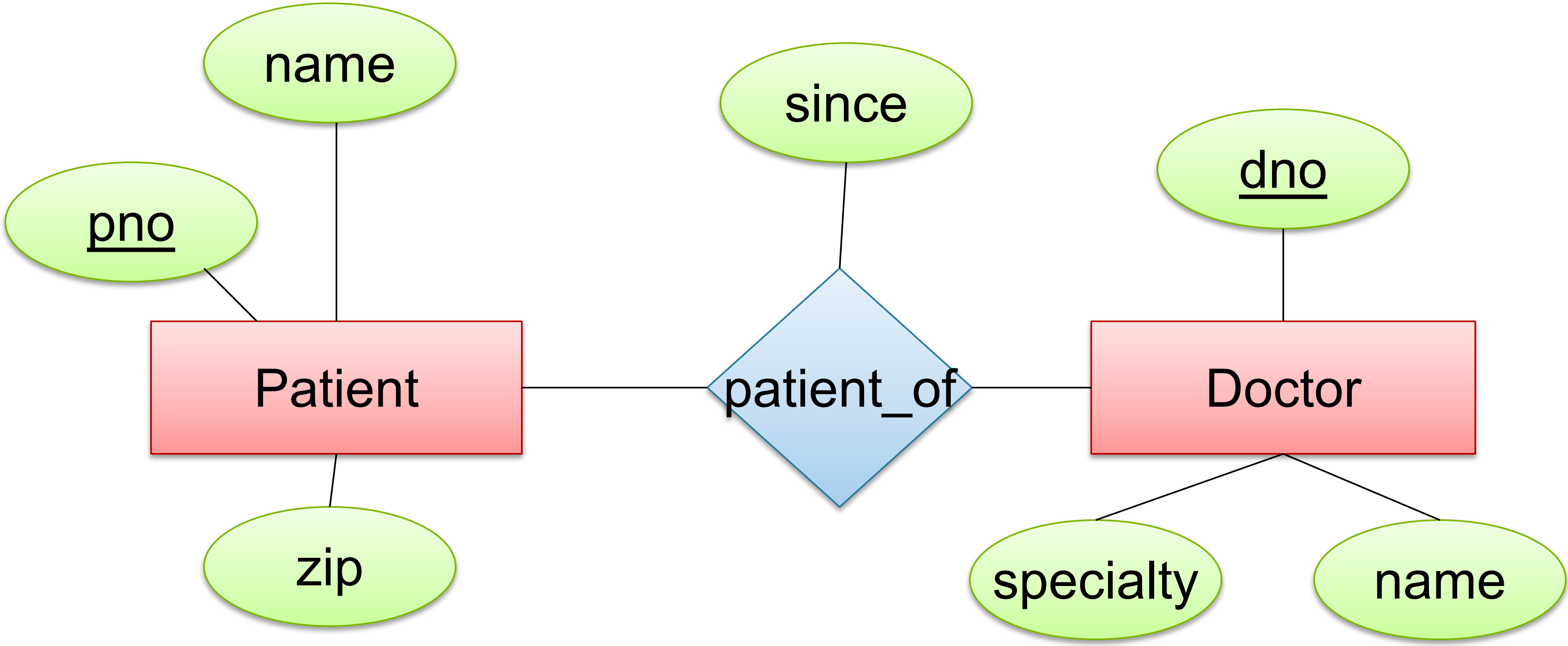


Entity-Relationship Model

- Typically, each entity has a key
- ER relationships can include multiplicity
  - One-to-one, one-to-many, etc.
  - Indicated with arrows
- Can model multi-way relationships
- Can model subclasses
- And more...



E/R To Relations



Patient

pno	name	zip
P311	Alice	98765
…		

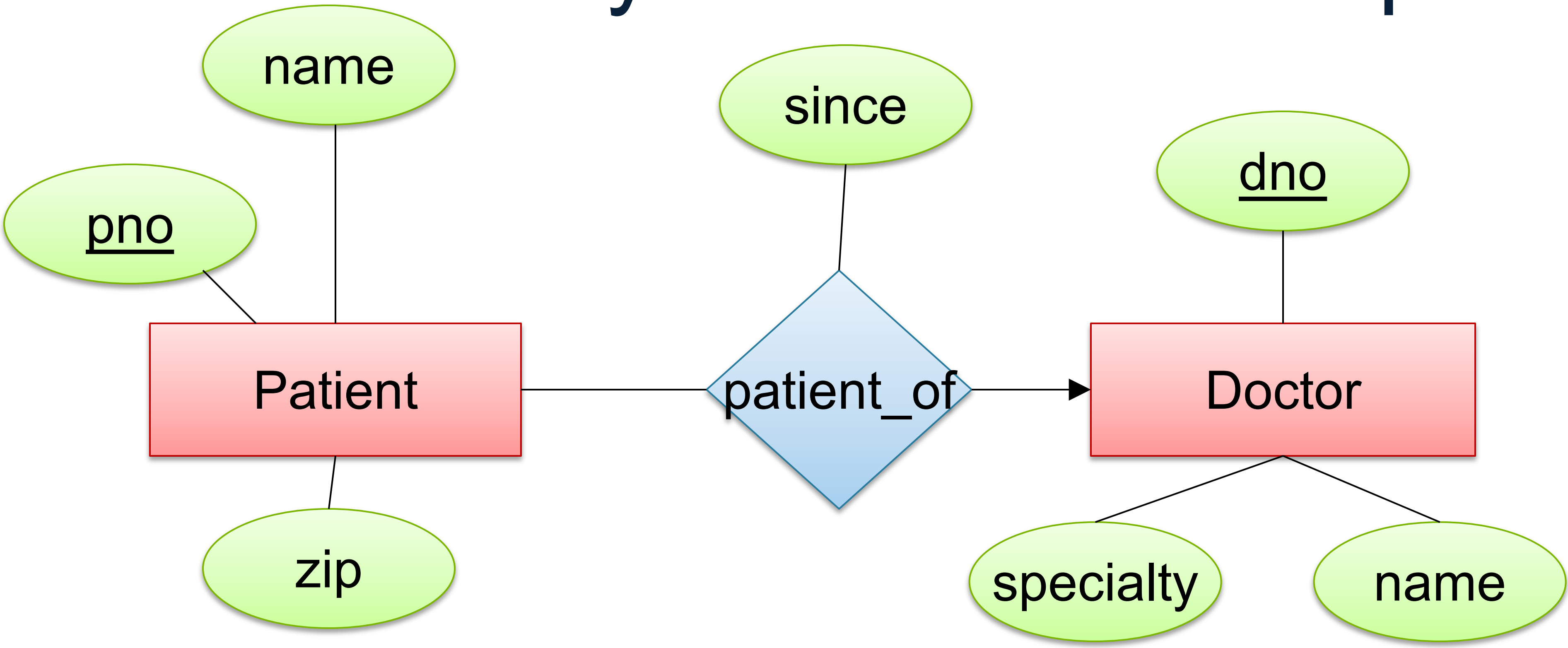
Patient_of

pno	dno	since
P311	D007	2001
…		

Doctor

dno	name	spec
D007	Bob	cardio
…		

Notice Many-One Relationship



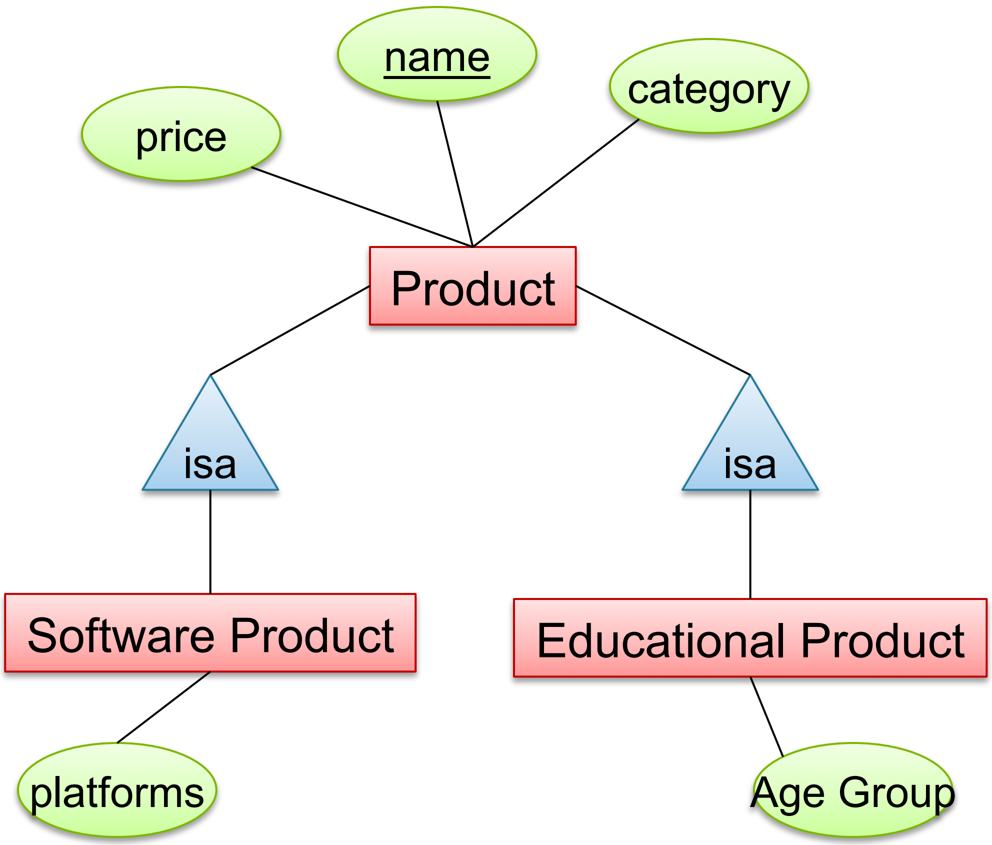
Patient

pno	name	zip	dno	since
P311	Alice	98765	D007	2001
…				

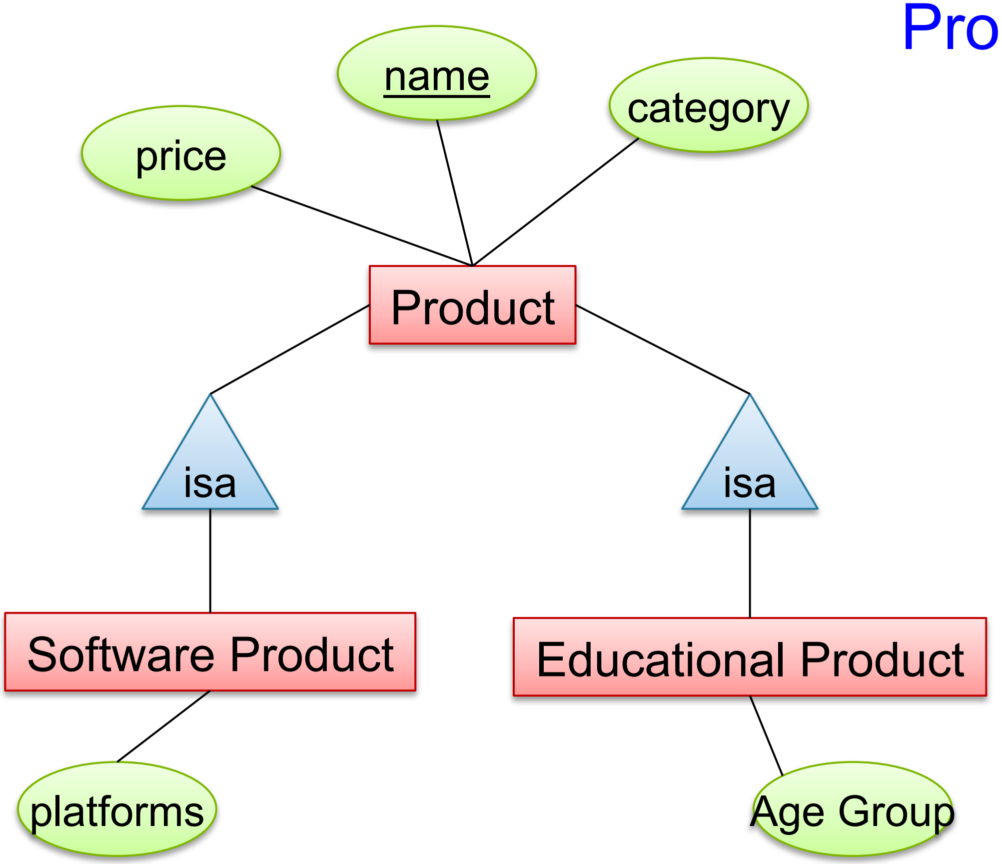
Doctor

dno	name	spec
D007	Bob	cardio
…		

Subclasses to Relations



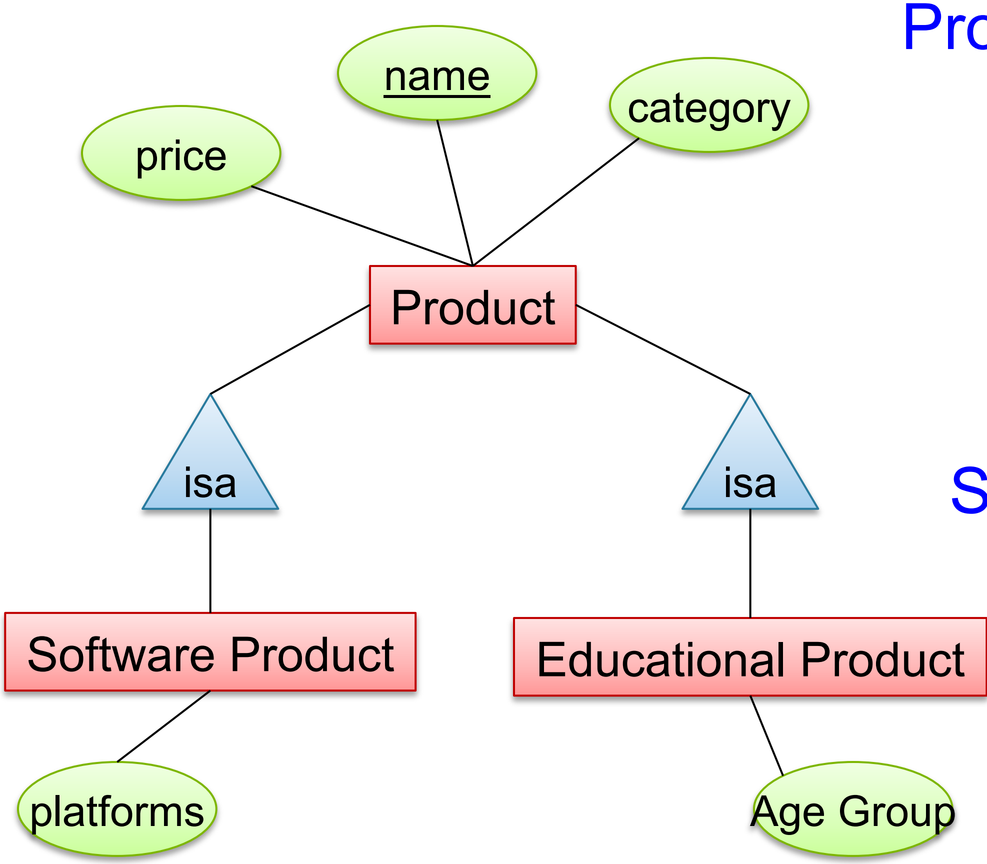
Subclasses to Relations



Product

Name	Price	Category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Subclasses to Relations



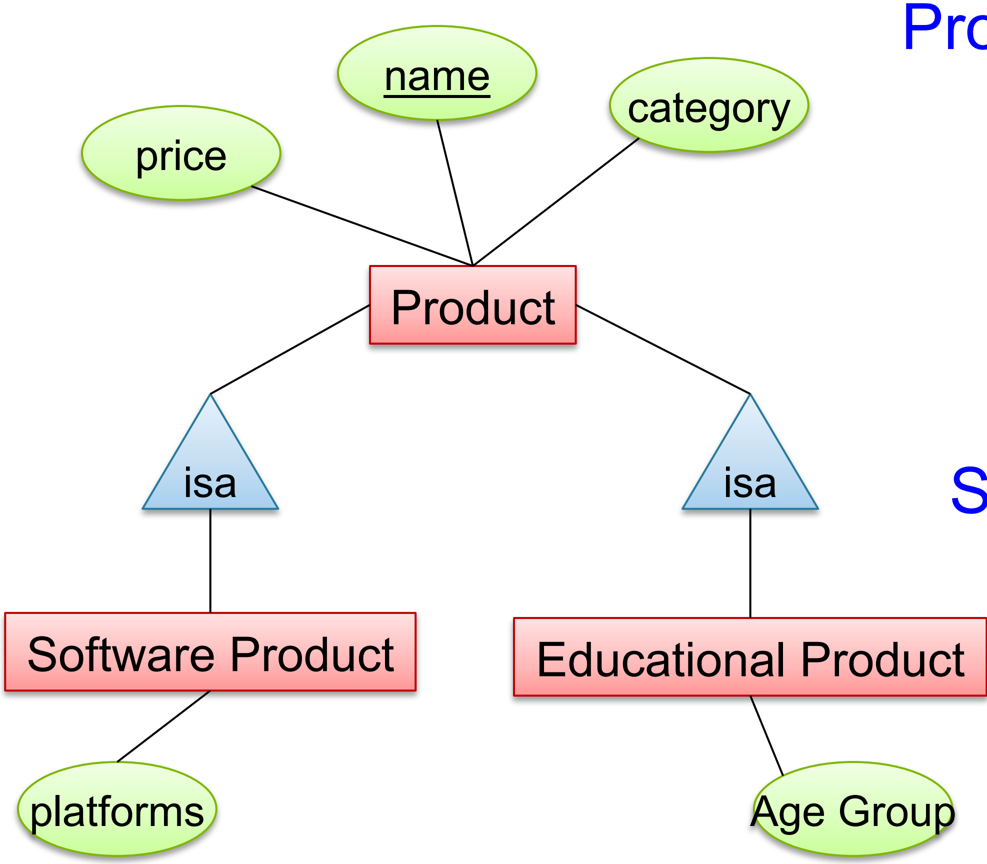
Product

Name	Price	Category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Sw.Product

Name	platforms
Gizmo	unix

Subclasses to Relations



Product

Name	Price	Category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Sw.Product

Name	platforms
Gizmo	unix

Ed.Product

Name	Age Group
Gizmo	toddler
Toy	senior

E/R Diagram to Relations

- Each entity set becomes a relation with a key
- Each relationship set becomes a relation *except* many-one relationships: just add the fk
- Each isA relationship becomes another relation, with both a key and foreign key

Outline

- Early data models
- Physical and logical independence in the relational model
- Conceptual design
- Data models that followed the relational model

Other Data Models

- Entity-Relationship: 1970's
  - Successful in logical database design
- Extended Relational: 1980's
- Semantic: late 1970's and 1980's
- Object-oriented: late 1980's and early 1990's
  - Address impedance mismatch: relational dbs ←→ OO languages
  - Interesting but ultimately failed (several reasons, see references)
- Object-relational: late 1980's and early 1990's
  - User-defined types, ops, functions, and access methods
- Semi-structured: late 1990's to the present
- Key-value pairs: the NoSQL databases since 2000s

Semistructured vs Relational

- Relational data model
  - “Schema first”
- Semistructured data model: XML, Json, Protobuf
  - ”Schema last”
  - Hierarchical (trees)

XML Syntax

```xml
<article mdate="2011-01-11" key="journals/acta/GoodmanS83">
   <author>Nathan Goodman</author>
   <author>Oded Shmueli</author>
   <title>NP-complete Problems Simplified on Tree Schemas.</title>
   <pages>171-178</pages>
   <year>1983</year>
   <volume>20</volume>
   <journal>Acta Inf.</journal>
   <url>db/journals/acta/acta20.html#GoodmanS83</url>
   <ee>http://dx.doi.org/10.1007/BF00289414</ee>
</article>
```

Semistructured, self-describing schema

JSon

Example from: http://www.jsonexample.com/

```
myObject = {
  "first": "John",
  "last": "Doe",
  "salary": 70000,
  "registered": true,
  "interests": [ "Reading", “Biking”, "Hacking" ]
}
```

Semistructured, self-describing schema

Discussion

- Stonebraker (circa 1998)
  - “schema last” is a niche market
- Today (circa 2020)
  - Major vendors scramble to offer efficient schema discovery while ingesting Json
- Why? What changed?

Discussion

- Stonebraker (circa 1998)
  - “schema last” is a niche market
- Today (circa 2020)
  - Major vendors scramble to offer efficient schema discovery while ingesting Json
- Why? What changed?
  - Today datasets are available in text format, often in Json; ingest first, process later

NoSQL Data Model(s)

- Web boom in the 2000's created a scalability crises
  - DBMS are single server and don't scale; e.g. MySQL
- NoSQL answer:
  - "Shard" data, i.e. distribute on a cluster
  - Simple data mode: key/value pairs

Key-Value Pair Data Model

- **Data model**: (key,value) pairs
- **Operations:** `get(key)`, `put(key,value)`
- **Distribution / Partitioning** – w/ hash function

No physical data independence!

Conclusion

- Data model: a formalism to describe/query the data
- Relational data model: tables+relational language; no description of physical store
- Data independence: efficiency needs to be realized separately, by the query optimizer
- Many competing "more efficient" data models have been proposed, and will be proposed, but fail because of lack of data independence