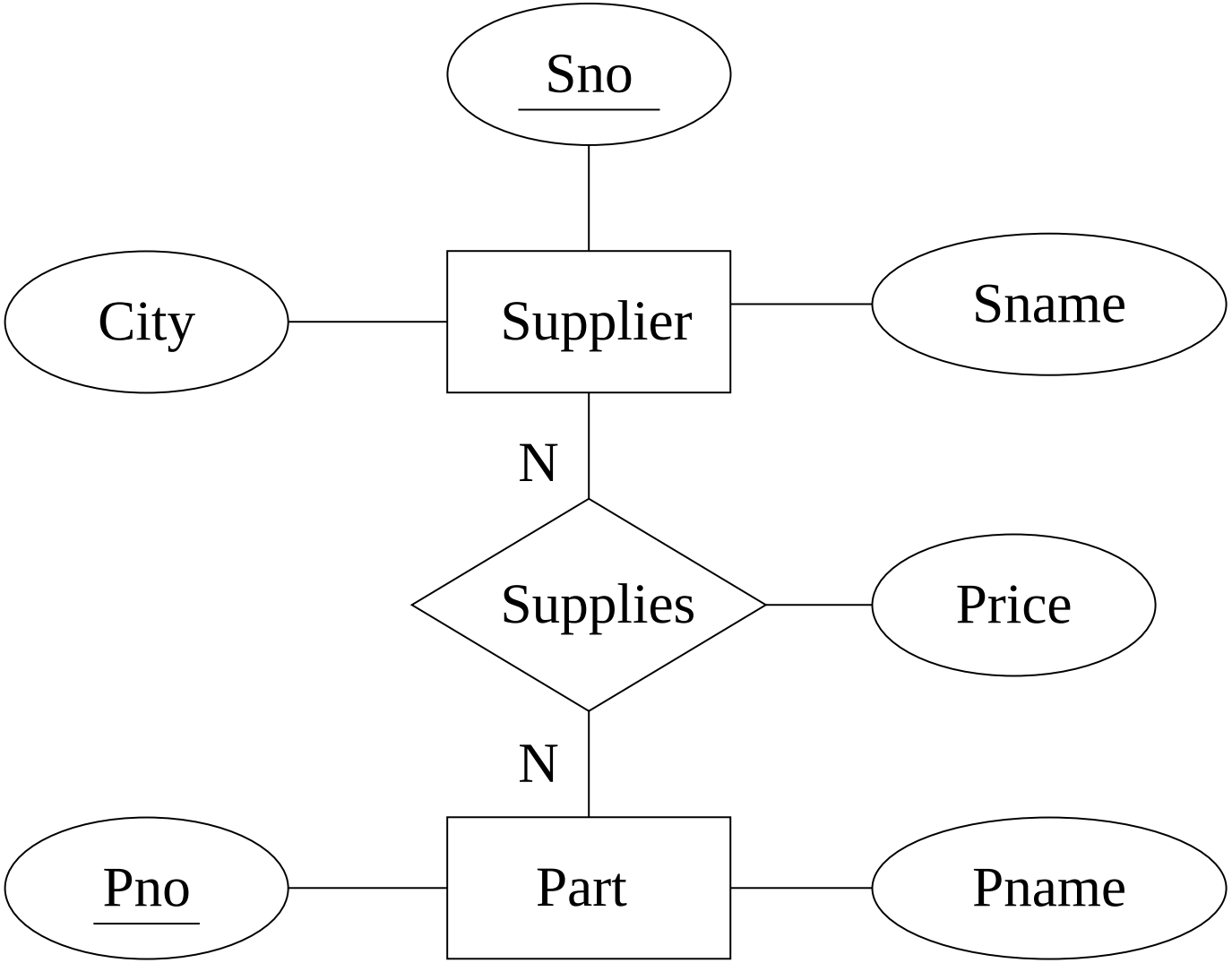
A Parts/Suppliers Database Example

- Description of a parts/suppliers database:
  - Each type of part has a name and an identifying number, and may be supplied by zero or more suppliers. Each supplier may offer the part at a different price.
  - Each supplier has an identifying number, a name, and a contact location for ordering parts.

Parts/Suppliers Example (cont.)



An E-R diagram for the parts/suppliers database.

Parts/Suppliers Example (cont.)

Suppliers

Sno	Sname	City
S1	Magna	Ajax
S2	Budd	Hull

Parts

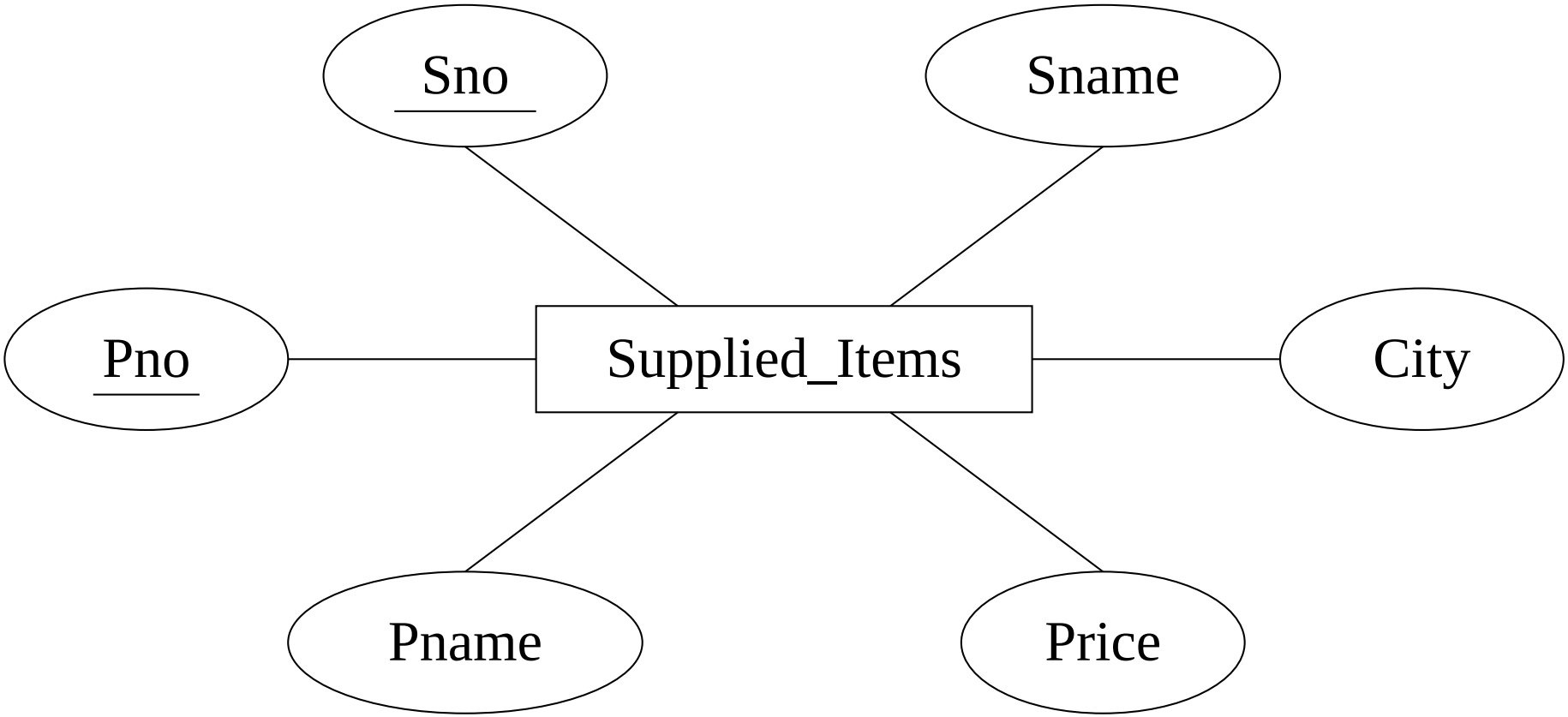
Pno	Pname
P1	Bolt
P2	Nut
P3	Screw

Supplies

Sno	Pno	Price
S1	P1	0.50
S1	P2	0.25
S1	P3	0.30
S2	P3	0.40

An instance of the parts/suppliers database.

Alternative Parts/Suppliers Database



An alternative E-R model for the parts/suppliers database.

Alternative Example (cont.)

Supplied_Items

Sno	Sname	City	Pno	Pname	Price
S1	Magna	Ajax	P1	Bolt	0.50
S1	Magna	Ajax	P2	Nut	0.25
S1	Magna	Ajax	P3	Screw	0.30
S2	Budd	Hull	P3	Screw	0.40

A database instance corresponding to the alternative E-R model.

Change Anomalies

- Some questions:
  - How do these alternatives compare?
  - Is one schema better than the other?
  - What does it mean for a schema to be good?
- The single-table schema suffers from several kinds of problems:
  - Update problems (e.g. changing name of supplier)
  - Insert problems (e.g. add a new item)
  - Delete problems (e.g. Budd no longer supplies screws)
  - Likely increase in space requirements
- The multi-table schema does not have these problems.
- Goals:
  - A methodology for evaluating schemas.
  - A methodology for transforming bad schemas into good schemas.

Designing Good Databases

- What makes a relational database schema good?
  - One criterion: independent facts in separate tables
- Functional dependencies (among attributes) are used to determine which attributes are mutually independent.

Functional Dependencies

- The schema of relation will be represented by a list R's attribute names. Each attribute name will often be designed by a single letter:
  - Example: if R is the Suppliers relation, with attributes Sno, Sname, City, we may simply write $R = SNC$
- The notation $X \subseteq R$ will be used to mean that X represents some subset of the attributes of R, e.g., $X = S$, or $X = SC$.
- Let R be a relation schema, and $X, Y \subseteq R$. The **functional dependency**

  $$X \rightarrow Y$$

  holds on R if no legal instance of R contains two tuples t and u with $t.X = u.X$ and $t.Y \neq u.Y$
- The notation $t.X$ means the values of the attributes X in tuple t.

Functional Dependencies Are Contraints

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Al-Nour
Hall	Compilers	Hoffman
Brown	Data Structures	Augenthaler

Functional dependencies are constraints on *all* instances of a schema. A single instance can confirm that a functional dependency does not hold. It cannot confirm that a functional dependency alway holds.

Functional Dependencies and Keys

- Keys (again)
  - A *superkey* is a set of attributes such that no two tuples (in an instance) agree on their values for those attributes.
  - A (candidate) *key* is a *minimal* superkey.
- Functional dependencies generalize the notion of superkey.

Saying that $K \subseteq R$ is a superkey for relation schema R is the same as saying that the functional dependency $K \rightarrow R$ holds on R

Boyce-Codd Normal Form (BCNF) - Informal

- BCNF formalizes the idea that in a good database schema, independent relationships are stored in separate tables.
- Given a database schema and a set of functional dependencies for the attributes in the schema, we can determine whether the schema is in BCNF. A database schema is in BCNF if each of its relation schemas is in BCNF.
- Informally, a relation schema is in BCNF if and only if any group of its attributes that functionally determines *any* others of its attributes functionally determines *all* others, i.e., that group of attributes is a superkey of the relation.

BCNF and Redundancy

- Why does BCNF avoid redundancy? Consider:

Supplied_Items

Sno	Sname	City	Pno	Pname	Price

- The functional dependency

$$\text{Sno} \rightarrow \text{Sname, City}$$

  holds for Supplied_Items

- This implies that a supplier's name and city must be repeated each once for each part supplied by that supplier.
- Now, assume this FD holds over a schema R that is in BCNF. This implies that:
  - Sno is a superkey for R
  - each Sno value appears on one row only
  - no need to repeat Sname and City values

Formal Definition of BCNF

- Let R be a relation schema and F a set of functional dependencies. A functional dependency $X \to Y$ is **trivial** if $Y \subseteq X$.
- Schema R is in **BCNF** if and only if whenever $(X \to Y) \in F^+$ and $XY \subseteq R$, then either
  - $(X \to Y)$ is trivial, or
  - X is a superkey of R
- A database schema $\{R_1, \ldots, R_n\}$ is in BCNF if each relation schema R_i is in BCNF

Closure of FD Sets

- The definition of BCNF refers to F^+. This is called the **closure** of the set of functional dependencies F.
- Informally, F^+ includes all of the dependencies in F, plus any dependencies they imply.
- For example, suppose that F consists of the two dependencies

  $$\text{A} \rightarrow \text{B}$$
  $$\text{B} \rightarrow \text{C}$$

  If a relation satisfies these two dependencies, then it must also satisfy the dependency

  $$\text{A} \rightarrow \text{C}$$

  This means that $\text{A} \rightarrow \text{C}$ should be included in F^+.
- Note that $F \subseteq F^+$

Armstrong's Axioms

- Logical implications of a set of functional dependencies can be derived by using inference rules called **Armstrong's axioms**
  - (reflexivity) $Y \subseteq X \Rightarrow X \rightarrow Y$
  - (augmentation) $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
  - (transitivity) $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$
- Additional rules can be derived from the three above:
  - (union) $X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$
  - (decomposition) $X \rightarrow YZ \Rightarrow X \rightarrow Y$
- These axioms are
  - sound (anything derived from F is in F^+)
  - complete (anything in F^+ can be derived)

Using Armstrong's Axioms

- Let F consist of:
  SIN, PNum $\rightarrow$ Hours
  SIN $\rightarrow$ EName
  PNum $\rightarrow$ PName, PLoc
  PLoc, Hours $\rightarrow$ Allowance
- A derivation of: SIN, PNum $\rightarrow$ Allowance
  1. SIN, PNum $\rightarrow$ Hours ($\in F$)
  2. PNum $\rightarrow$ PName, PLoc ($\in F$)
  3. PLoc, Hours $\rightarrow$ Allowance ($\in F$)
  4. SIN, PNum $\rightarrow$ PNum (reflexivity)
  5. SIN, PNum $\rightarrow$ PName, PLoc (transitivity, 4 and 2)
  6. SIN, PNum $\rightarrow$ PLoc (decomposition, 5)
  7. SIN, PNum $\rightarrow$ PLoc, Hours (union, 6, 1)
  8. SIN, PNum $\rightarrow$ Allowance (transitivity, 7 and 3)

Computing Attribute Closures

- There is a more efficient way of using Armstrong's axioms

```
function Compute X+ (X, F) {
    X+ = X;
    while there exists (Y → Z) ∈ F such that
                          Y ⊆ X+ and Z ⊈ X+ do {
        X+ = X+ ∪ Z;
    }
    return (X+);
}
```

Computing Attribute Closures (cont'd)

- Let R be a relational schema and F a set of functional dependencies on R. Then

  Theorem: $X \rightarrow Y \in F^+$ if and only if

  $$Y \subseteq ComputeX^+(X, F)$$

  Theorem: X is a superkey of R if and only if

  $$ComputeX^+(X, F) = R$$

Attribute Closure Example

- Let F consists of:
  - SIN→EName
  - Pnum→Pname,Ploc
  - PLoc,Hours→Allowance
- ComputeX^+({Pnum,Hours},F):

FD	X^+
initial	Pnum,Hours
Pnum→Pname,Ploc	Pnum,Hours,Pname,Ploc
PLoc,Hours→Allowance	Pnum,Hours,Pname,Ploc,Allowance

Computing a Normal Form

- What to do if a given relational schema is not in BCNF?
- Strategy: identify undesirable dependencies, then *decompose* the schema.
- Let R be a relation schema. A collection $\{R_1, \ldots, R_n\}$ of relation schemas is a **decomposition** of R if

$$R = R_1 \cup R_2 \cup \cdots \cup R_n$$

- A good decomposition does not
  - lose information
  - complicate checking of constraints

Lossless-Join Decompositions

- Consider decomposing

Marks

Student	Assignment	Group	Mark
Ann	A1	G1	80
Ann	A2	G3	60
Bob	A1	G2	60

into two tables

SGM

Student	Group	Mark
Ann	G1	80
Ann	G3	60
Bob	G2	60

AM

Assignment	Mark
A1	80
A2	60
A1	60

Lossless-Join Decompositions (cont'd)

- Computing the natural join of SGM and AM produces

Student	Assignment	Group	Mark
Ann	A1	G1	80
Ann	A2	G3	60
Ann	A1	G3	60
Bob	A2	G2	60
Bob	A1	G2	60

- The join result contains spurious tuples. Information would be lost if we were to replace Marks by SGM and AM.
- If re-joining SGM and AM would **always** produce exactly the tuples in Marks, then SGM and AM is called a **lossless-join decomposition**.

Another Lossless-Join Decomposition Example

Consider the following (BCNF) decomposition of the relation from the parts/suppliers database.

Snos

Sno
S1
S2

Snames

Sname
Magna
Budd

Cities

City
Ajax
Hull

Pnum

Inum
I1
I2
I3

Pname

Iname
Bolt
Nut
Screw

Price

Price
0.50
0.25
0.30
0.40

Identifying Lossless-Join Decompositions

- Since schemas, not schema instances, are decomposed, how to be sure that a decomposition is lossless?
- A decomposition $\{R_1, R_2\}$ of R is lossless if and only if the common attributes of R_1 and R_2 form a superkey for either schema, that is

$$R_1 \cap R_2 \rightarrow R_1 \quad \text{or} \quad R_1 \cap R_2 \rightarrow R_2$$

- Example:
  R = {Student, Assignment, Group, Mark}
  F = {(Student, Assignment $\rightarrow$ Group, Mark),
  (Assignment, Group $\rightarrow$ Mark) }
  R_1 = {Student, Group, Mark} R_2 = {Assignment, Mark}

- Decomposition $\{R_1, R_2\}$ is lossy because $R_1 \cap R_2$ is not a superkey of either SGM or AM

Computing a Lossless-Join BCNF Decomposition

```
function ComputeBCNF(𝒟, F) {
   Result = 𝒟;
   while some R ∈ Result and X → Y ∈ F+
                violate the BCNF condition do {
      remove R from Result;
      insert R − (Y − X) into Result;
      insert X ∪ Y into to Result;
   }
    return(Result);
}
```

No efficient procedure to do this exists.

Decomposition - An Example

- $R = \{$Sno,Sname,City,Pno,Pname,Price$\}$
- functional dependencies:
  Sno $\rightarrow$ Sname,City
  Pno $\rightarrow$ PName
  Sno,Pno $\rightarrow$ Price
- This schema is not in BCNF because, for example, Sno determines Sname and City, but is not a superkey of R.

Decomposition - An Example (cont.)

- Using the dependency

$$\text{Sno} \rightarrow \text{Sname,City}$$

  R can be decomposed into

$$R_1 = \{\text{Sno,Pno,Pname,Price}\}$$
$$R_2 = \{\text{Sno,Sname,City}\}$$

- R_2 is now in BCNF (why?) but R_1 is not, and must be further decomposed.

Decomposition - An Example (cont.)

- Using the dependency

$$\text{Pno} \rightarrow \text{Pname}$$

  R_1 can be decomposed into

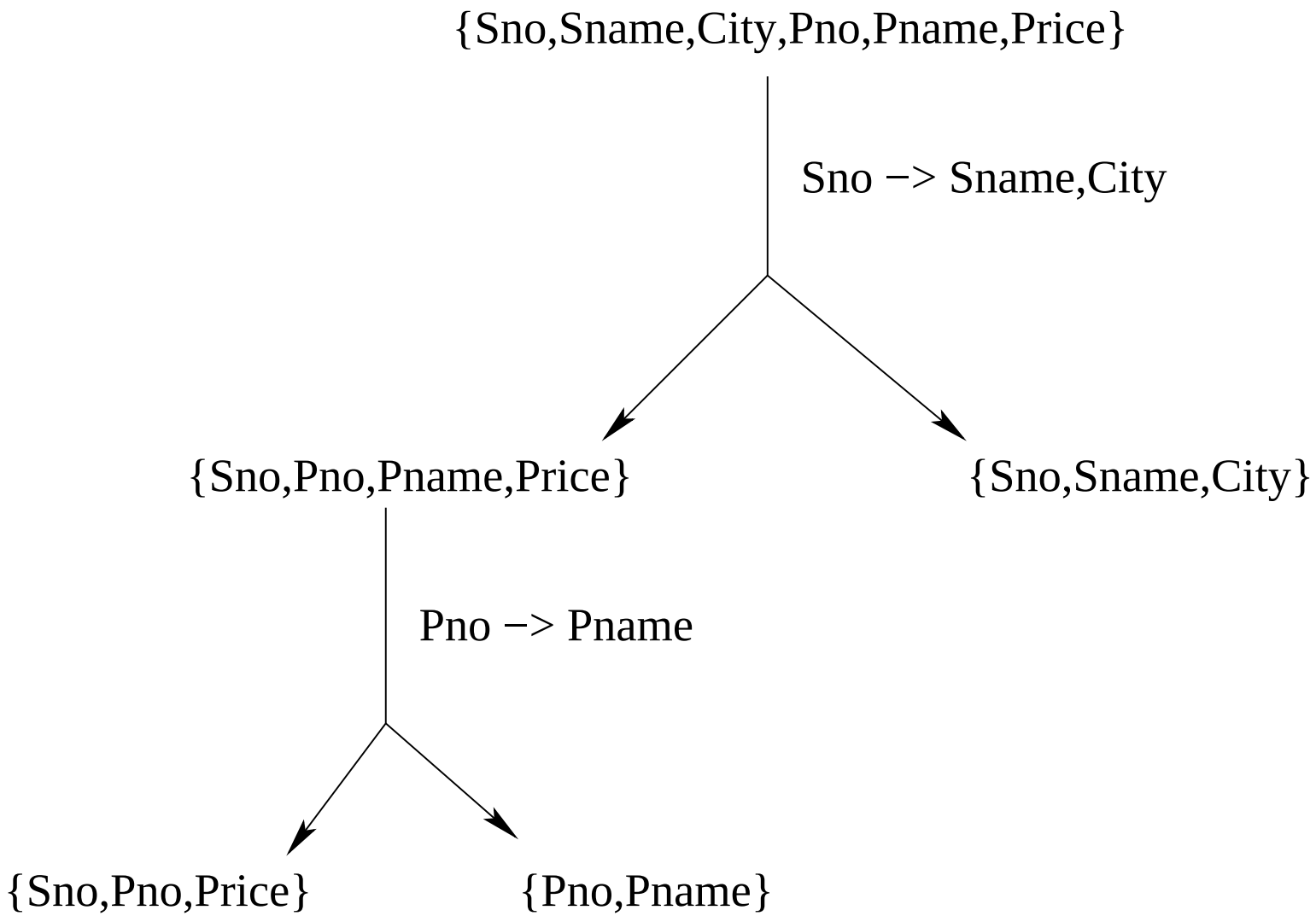
$$R_3 = \{\text{Sno,Pno,Price}\}$$
$$R_4 = \{\text{Pno,Pname}\}$$

- The complete schema is now

$$R_2 = \{\text{Sno,Sname,City}\}$$
$$R_3 = \{\text{Sno,Pno,Price}\}$$
$$R_4 = \{\text{Pno,Pname}\}$$

- This schema is a lossless-join, BCNF decomposition of the original schema R.

Decomposition Diagram



Dependency Preservation

- Ideally, a decomposition would be dependency preserving as well as lossless.
- Dependency preserving decompositions allow for efficient testing of constraints on the decomposed schema.
- Consider a relation schema `R = {Proj, Dept, Div}` with functional dependencies
  FD1: Proj $\rightarrow$ Dept
  FD2: Dept $\rightarrow$ Div
  FD3: Proj $\rightarrow$ Div
- Consider two decompositions
  D_1 = {R1 = {Proj, Dept}, R2 = {Dept, Div}}
  D_2 = {R1 = {Proj, Dept}, R3 = {Proj, Div}}
- Both are lossless. (Why?)

Dependency Preservation (cont'd)

- Decomposition D_1 lets us test FD1 on table R1 and FD2 on table R2; if they are both satisfied, FD3 is automatically satisfied
- In decomposition D_2 we can test FD1 on table R1 and FD3 on table R3. Dependency FD2 is an **interrelational constraint**: testing it requires joining tables R1 and R3
- Let R be a relation schema and F a set of functional dependencies on R. A decomposition $D = \{R_1, \ldots, R_n\}$ of R is **dependency preserving** if there is an equivalent set F' of functional dependencies, none of which is interrelational in D

It is possible that no dependency preserving BCNF decomposition exists. Consider $R = ABC$ and $F = \{\text{AB} \rightarrow \text{C}, \text{C} \rightarrow \text{B}\}$.
