Distribution Design

Data Management for Big Data

2018-2019 (spring semester)

Dario Della Monica

These slides are a modified version of the slides provided with the book

Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

The original version of the slides is available at: extras.springer.com

Outline (distributed DB)

- Introduction (Ch. 1) ★
- Distributed Database Design (Ch. 3) ★
  - ➡ Fragmentation
  - ➡ Data distribution (allocation)
- Distributed Query Processing (Ch. 6-8) ★
- Distributed Transaction Management (Ch. 10-12) ★

★ Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

Outline (today)

- Distributed DB design (Ch. 3) *
  - Introduction
  - Top-down (vs. bottom-up) design
  - Distribution design issues
    - Fragmentation
    - Allocation
  - Fragmentation
    - Horizontal Fragmentation (HF)
      - ✓ Primary Horizontal Fragmentation (PHF)
      - ✓ Derived Horizontal Fragmentation (DHF)
    - Vertical Fragmentation (VF)
    - Hybrid Fragmentation (HyF)
  - Allocation
  - Data directory

* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

Design Problem

- In the general setting:

    Making decisions about the placement of data and programs (control) across the sites of a computer network as well as possibly designing the network itself

- In Distributed DBMS, the placement of applications entails
    - ➡ placement of the distributed DBMS software; and
    - ➡ placement of the applications that run on the database

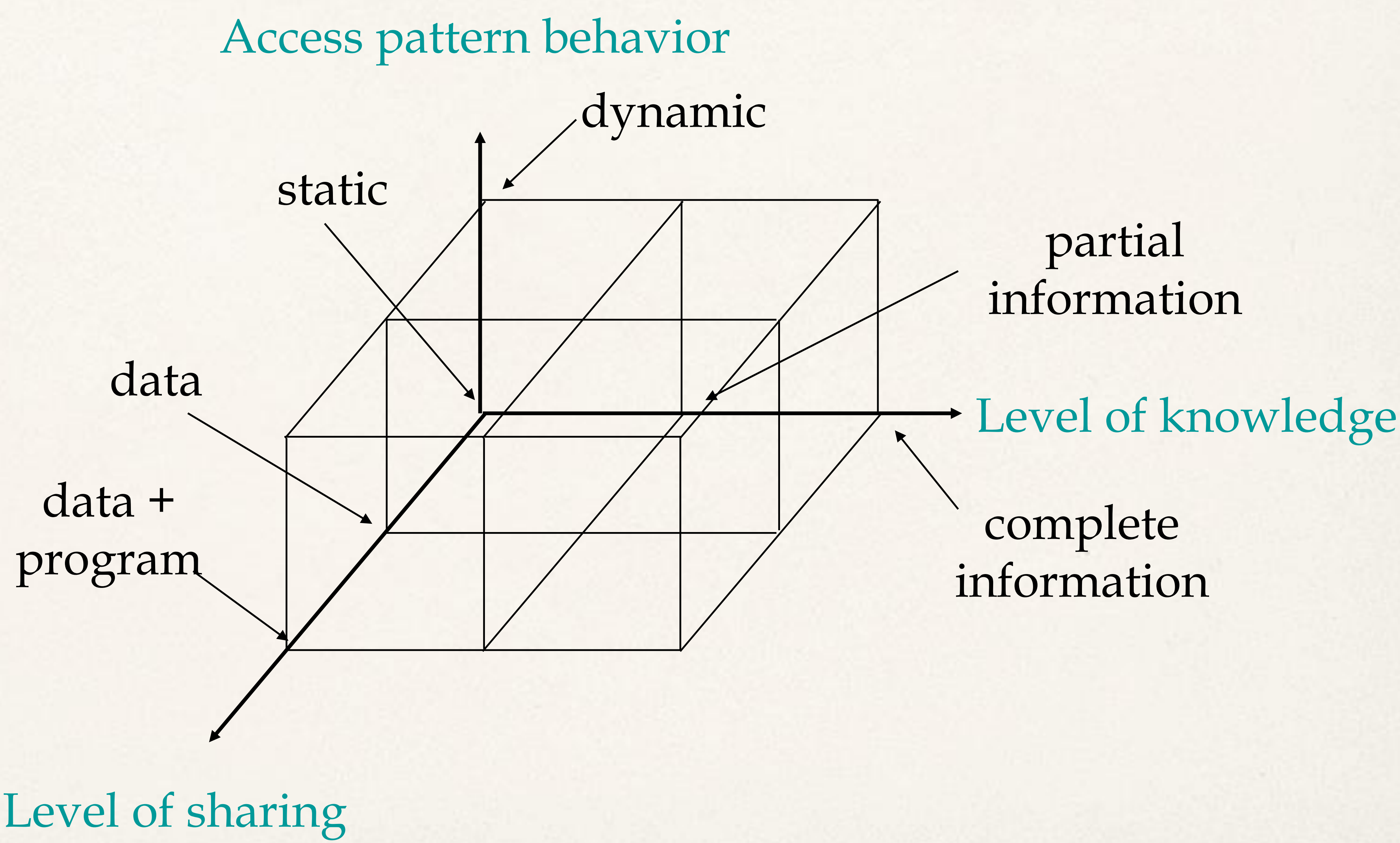
Forms of Distribution

- Architecture classification (in previous class)
  - Client/server
  - **Fully distributed**
  - MDBS

  based on distribution degree of DBMS (control) among nodes of the network

- Another classification (based on distribution degree of data)
  - Dimensions of the classification
    - Level of sharing (*no sharing* vs. *data sharing* vs. *data-plus-program sharing*)
    - Behavior of access pattern (*static* vs. *dynamic*)
    - Level of knowledge on access pattern behavior (*no information* vs. *complete information* vs. *partial information*)

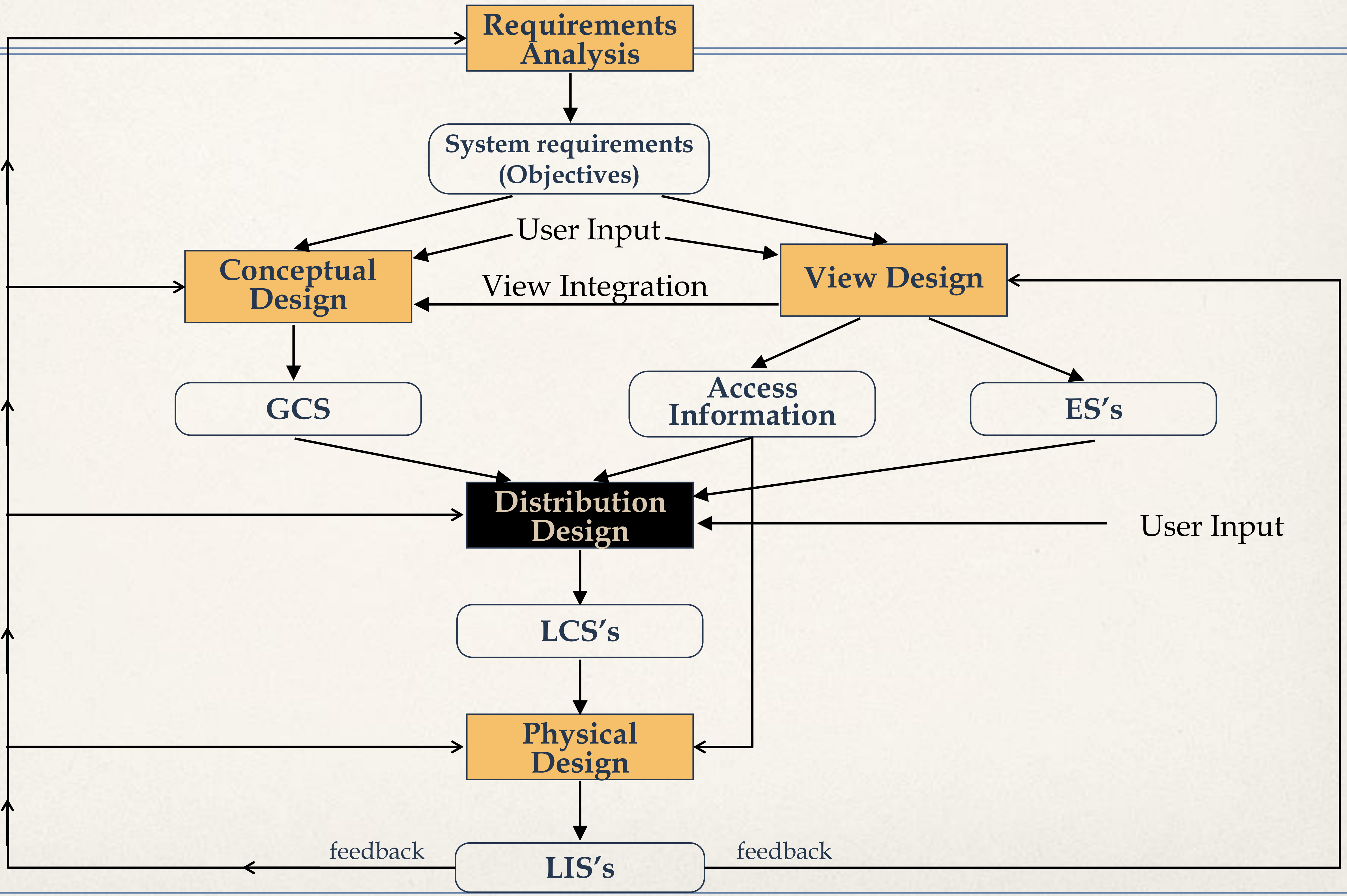
Dimensions of the Problem



Distribution Design

- Top-down
  - → mostly in designing systems from scratch
  - → mostly in homogeneous systems
  - → applies to fully distributed DBMS (a logical view of the whole DB exists)
- Bottom-up
  - → when the databases already exist at a number of sites
  - → applies to MDBS (we will not treat them)

Top-Down Design



Distribution Design Issues

Distribution design activity boils down to *fragmentation* and *allocation*

❶ Why fragment at all?	[reasons for fragmentation]
❷ How to fragment?	[fragmentation alternatives]
❸ How much to fragment?	[degree of fragmentation]
❹ How to test correctness?	[correctness rules of fragmentation]
❺ How to allocate?	[allocation alternatives]
❻ Information requirements?	[for both fragmentation and allocation]

Reasons for Fragmentation

- Can't we just distribute relations (no intrinsic reason to fragment)?
  - ➡ distributed file systems are not fragmented (i.e., distr. unit is the file)
- What is a reasonable unit of distribution?
  - ➡ advantages of fragmentation (why isn't relation the best choice?)
    - ✦ application views are subsets of relations ➔ **locality** for application accesses on subsets of relations
      - ✓ 2 applications accessing different portion of a relation: without fragmentation, either unnecessary data replication or loss of locality (extra communication)
    - ✦ without fragmentation, no room for **intra-query parallelism**
  - ➡ disadvantages of fragmentation
    - ✦ might cause queries to be executed on more than one fragment (performance degradation, especially when fragments are not mutually exclusive)
    - ✦ semantic data control (especially integrity enforcement) more difficult and costly

Fragmentation Alternatives

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

Horizontal fragmentation

- $PROJ_1$: projects with budget less than $200,000
- $PROJ_2$: projects with budget greater than or equal to $200,000

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

Vertical fragmentation

- PROJ1: information about project budgets
- PROJ2:information about project names and locations

$PROJ_1$

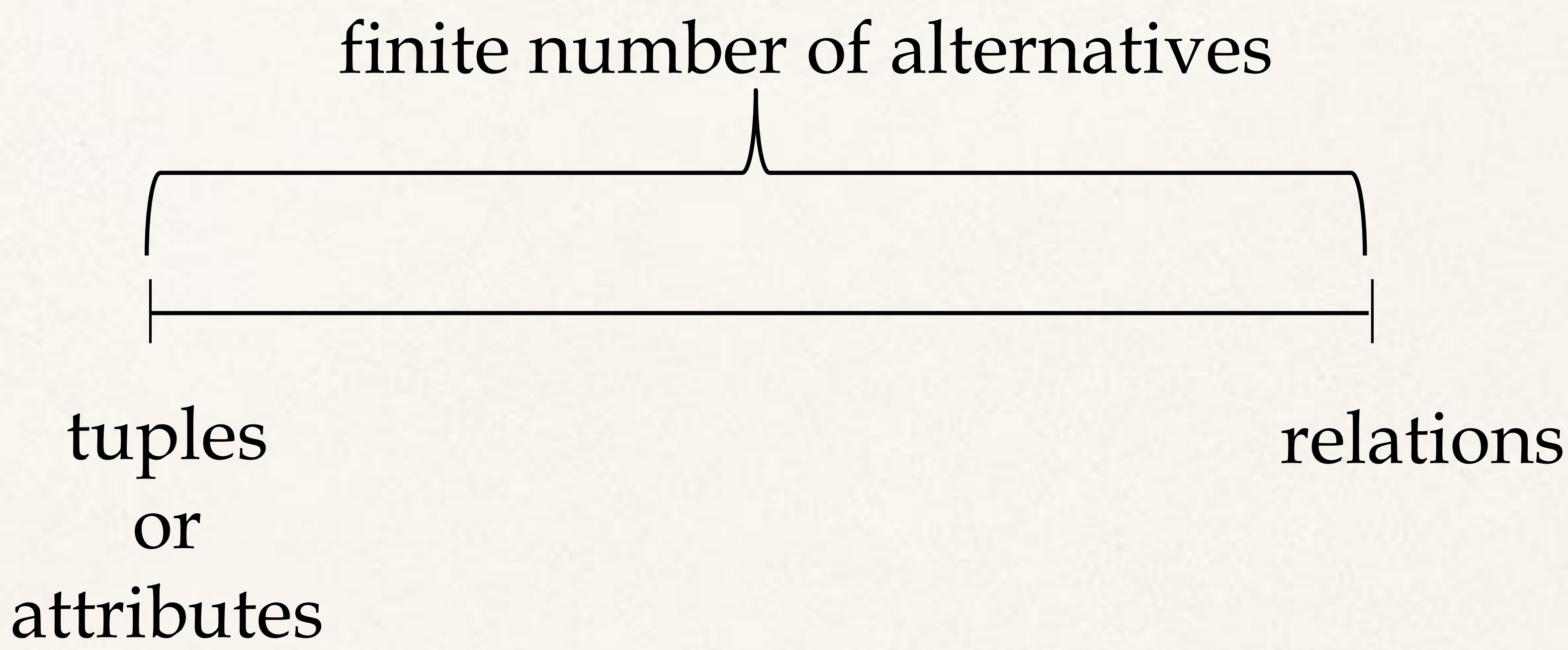
PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris

Hybrid fragmentation: obtained by nesting horizontal and vertical fragmentation

Degree of Fragmentation



- Finding the suitable level of partitioning within this range
- It depends on the applications that will use the DB
- This is the real difficulty of fragmentation

Correctness of Fragmentation

- Completeness
  - Decomposition of relation R into fragments $R_1, R_2, ..., R_n$ is complete if and only if each data item in R can also be found in some R_i
- Reconstruction
  - If relation R is decomposed into fragments $R_1, R_2, ..., R_n$, then there should exist some relational operator ∇ such that

$$R = \nabla_{1 \le i \le n} R_i$$

- Disjointness
  - If relation R is decomposed into fragments $R_1, R_2, ..., R_n$, and data item d_i is in R_j, then d_i should not be in any other fragment R_k ($k \neq j$).

Allocation Alternatives

- It is more about whether or not to replicate a fragment
  - *Partitioned* (aka *non-replicated*): each fragment resides at only one site
  - *fully replicated*: each fragment at each site
  - *partially replicated*: each fragment at some of the sites
- Rule of thumb:

  If $\frac{\text{read-only queries}}{\text{update queries}} >> 1$, replication is advantageous,

  otherwise replication may cause problems
- In case of partially replicated DDBS, the number of copies of replicated fragments can either be an input to the allocation algorithm or a decision variable to be computed by the algorithm

Information Requirements

- The difficulty of the distributed DB design problem is that too many factor affect the choices towards an optimal design
  - ➡ Logical organization of the DB
  - ➡ Location of DBMS applications
  - ➡ Characteristics of user applications (how they access the DB)
  - ➡ Properties of (computers at) network nodes
  - ➡ …
- Those can be grouped into four categories:
  - ➡ Database information
  - ➡ Application information
  - ➡ Communication network information
  - ➡ Computer system information

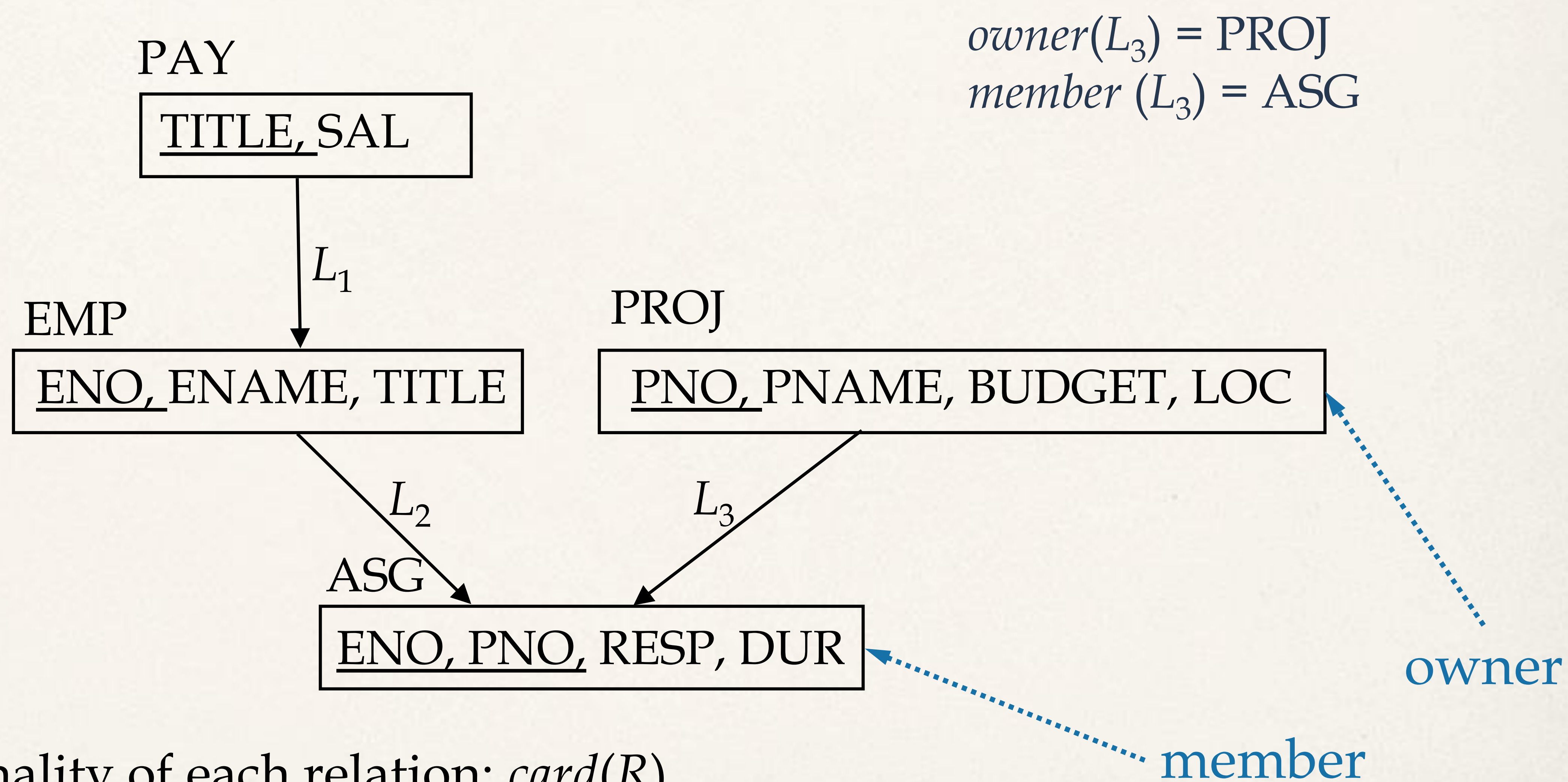
} quantitive information, mostly used for allocation, we will not treat them

Fragmentation

- Horizontal Fragmentation (HF)
  - Primary Horizontal Fragmentation (PHF)
  - Derived Horizontal Fragmentation (DHF)
- Vertical Fragmentation (VF)
- Hybrid Fragmentation (HyF)

HF – Information Requirements (DB Info)

- Database Information
  - relationship



  - cardinality of each relation: *card*(*R*)

HF – Information Requirements (Qualitative Application Info)

- (qualitative) application information (for fragmentation)
  - Predicate used in queries
    - 80/20 rule: the most active 20% of user applications account for 80% of accesses
    - **simple predicates**: Given $R[A_1, A_2, \ldots, A_n]$, a **simple predicate** p_j over R is

      $$p_j : A_i \,\theta\, Value$$

      where $\theta \in \{=,<,\leq,>,\geq,\neq\}$, $Value \in D_i$ and D_i is the domain of A_i.

      Example:

      PNAME = "Maintenance"

      BUDGET ≤ 200000
    - **minterms**: Given a set $Pr = \{p_1, p_2, \ldots, p_m\}$ of simple predicates over a relation R, a **minterm** (induced by Pr) is a conjunction

      $$\bigwedge_{p_j \in Pr} p_r^*$$

      where $p_j^* \in \{ p_j , \neg p_j \}$, for all $p_j \in Pr$

      We let $M_{Pr} = \{m_1, m_2, \ldots, m_r\}$ be the set of all minterms induced by a set of simple predicates Pr

HF – Information Requirements (Example of Qualitative Application Info)

Example

Pr = { PNAME="Maintenance" , BUDGET < 200000 }

$M_{Pr} = \{ m_1 , m_2 , m_3 , m_4 \}$

Where

- m_1: PNAME="Maintenance" ∧ BUDGET < 200000
- m_2: ¬(PNAME="Maintenance") ∧ BUDGET < 200000
- m_3: PNAME= "Maintenance" ∧ ¬(BUDGET < 200000)
- m_4: ¬(PNAME="Maintenance") ∧ ¬(BUDGET < 200000)

HF – Information Requirements (Quantitative Application Info)

- (quantitative) application information (for allocation)
  - **minterm selectivity** of a minterm m_i (over relation R):

    $sel(m_i)$

    The rate of tuples of R that satisfy m_i
    - Example: $sel(m_1) = 0$, $sel(m_3) = 0.25$
  - **access frequency** of a query q_i:

    $acc(q_i)$

    The frequency with which a user application (query) q_i accesses data
  - **access frequency** for a minterm m_i:

    $acc(m_i)$

    can be derived from all $acc(q_j)$ s.t. m_i occurs in q_j

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

m_1: PNAME="Maintenance“ ∧ BUDGET < 200000

m_2: ¬(PNAME="Maintenance") ∧ BUDGET < 200000

m_3: PNAME= "Maintenance“ ∧ ¬(BUDGET < 200000)

m_4: ¬(PNAME="Maintenance") ∧ ¬(BUDGET < 200000)

Primary Horizontal Fragmentation

- Primary horizontal fragmentation (**PHF**) is induced by a set of minterm.
- **Definition:** A set M of minterm induces the fragmentation

$$F = \{ R_j \mid R_j = \sigma_m(R),\ m \in M \}$$

- Therefore, a horizontal fragment R_i of relation R consists of all the tuples of R which satisfy a minterm predicate m_i

⬇

Given a set of minterm predicates M, there are as many horizontal fragments of relation R as there are minterm predicates (some fragments might be empty)

- Set of horizontal fragments also referred to as minterm fragments.

PHF – Example (1)

- Assume there is an application **Q: find projects with budget less than 200 000 €**
- Then, it makes sense to consider the set of simple predicates S = { BUDGET < 200000 }

  which induces the set of minterms M_S = { BUDGET < 200000, ¬(BUDGET < 200000) }

  which, in turn, induces fragmentation F = { $PROJ_1$, $PROJ_2$ }
- $PROJ_1$ and $PROJ_2$ are called **minterm fragments defined on *S***

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

PHF – Example (2)

- Assume there is ALSO another application **Q′: find projects at each location**
- Then, it makes sense to consider the set of simple predicates

$$S' = \{LOC = \text{“Montreal”}, LOC = \text{“New York”}, LOC = \text{“Paris”}, \}$$

which induces the set of minterms (use abbreviations **L_M**: LOC = “Montreal”, **L_N**: LOC = “New York”, **L_P**: LOC = “Paris”)

$M_{S'} = \{$ ~~$L_M \wedge L_N \wedge L_P$,~~ ~~$L_M \wedge L_N \wedge \neg L_P$,~~ ~~$L_M \wedge \neg L_N \wedge L_P$,~~ $L_M \wedge \neg L_N \wedge \neg L_P$,
~~$\neg L_M \wedge L_N \wedge L_P$,~~ $\neg L_M \wedge L_N \wedge \neg L_P$, $\neg L_M \wedge \neg L_N \wedge L_P$, ~~$\neg L_M \wedge \neg L_N \wedge \neg L_P$~~ $\}$

which reduces to $\{ L_M \wedge \neg L_N \wedge \neg L_P, \quad \neg L_M \wedge L_N \wedge \neg L_P, \quad \neg L_M \wedge \neg L_N \wedge L_P, \}$

which, in turn, induces fragmentation $F' = \{ PROJ'_1, PROJ'_2, PROJ'_3 \}$

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

$PROJ'_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ'_2$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York

$PROJ'_3$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

Completeness

- Sets of simple predicates (and thus sets of minterms) should be complete and minimal
- Intuitively, *complete* means that all applications (queries) are taken into account
- Definition: a set of simple predicates *Pr* is said to be complete if and only if any two tuples of the same minterm fragment defined on *Pr* have the same probability of being accessed by any application

Completeness – Examples

Definition: a set of simple predicates *Pr* is said to be complete if and only if any two tuples of the same minterm fragment defined on *Pr* have the same probability of being accessed by any application

- S′ is not complete (wrt. applications Q and Q′)
  - it produces $F' = \{ PROJ'_1 , PROJ'_2 , PROJ'_3 \}$
  - Q (find projects with budget less than 200 000 €) only accesses project P2 in fragment $PROJ'_2$
- S is not complete (wrt. applications Q and Q′)
  - it produces $F = \{ PROJ_1 , PROJ_2 \}$
  - Q′ instantiated with "New York" (find projects based in New York) only accesses project P2 in fragment $PROJ_1$
- $S'' = S \cup S'$
  - it produces

$$M_{S''} = \{ BUDGET < 200000 \wedge L_M ,\ \neg BUDGET < 200000 \wedge L_M ,\ BUDGET < 200000 \wedge L_N ,\ \neg BUDGET < 200000 \wedge L_N ,\ BUDGET < 200000 \wedge L_P ,\ \neg BUDGET < 200000 \wedge L_P \}$$

$PROJ'_2$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

Minimality

- Sets of simple predicates (and thus sets of minterms) should be complete and minimal
- Intuitively, *minimal* means that all predicates should be relevant
- Definition: a set of simple predicates *Pr* is said to be minimal if and only if a predicate $p \in Pr$ produces a fragment, i.e., p divides fragment F into F_1 and F_2, only if F_1 and F_2 are accessed differently by at least one application
- S′′ is not minimal (wrt. application Q′)
  - It produces F = { PROJ′$_1$, PROJ′′$_2$, PROJ′′′$_2$, PROJ′$_3$ }
  - Q′ (find projects at each location) cannot distinguish between PROJ′′$_2$ and PROJ′′′$_2$:

    Q′ accesses PROJ′′$_2$ iff Q′ accesses PROJ′′′$_2$

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

PROJ′$_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

PROJ′′$_2$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

PROJ′′′$_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

PROJ′$_3$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

PHF – Algorithm (Intuition)

Input: a relation R and a set of simple predicates Pr over R

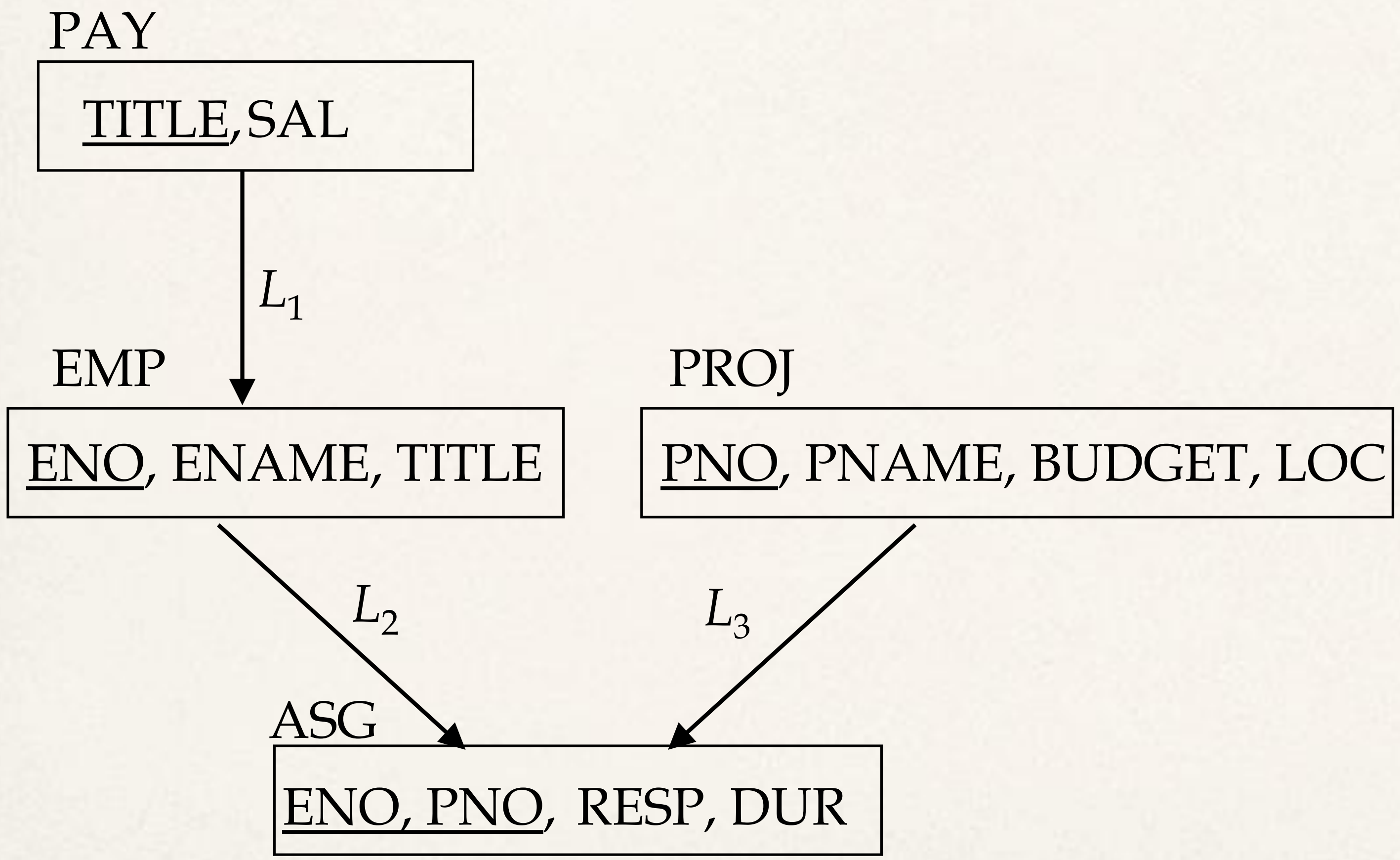
Output: a *complete* and *minimal* set of simple predicates Pr' over R

Minimality rule (relevant predicates): a predicate $p \in Pr$ that produces a fragment, i.e., p divides fragment F into F_1 and F_2, is **relevant** if and only if F_1 and F_2 are accessed differently by at least one application

```
repeat
     select a relevant (for R) predicate p ∈ Pr
     P := P \ { p }
     P' := P' ∪ { p }
     P' := P' \ {p ∈ Pr | p is not relevant }
until P' is complete
determine set M of minterms
eliminate contradictory minterms from M
return fragmentation F = { F_m | m ∈ M }
```

Derived Horizontal Fragmentation

- Derived Horizontal Fragmentation (DHF) is defined on a member relation of a link according to a selection operation specified on its owner (propagated from owner to member)



owner(L_1) = PAY
member (L_1) = EMP

owner(L_2) = EMP
member (L_2) = ASG

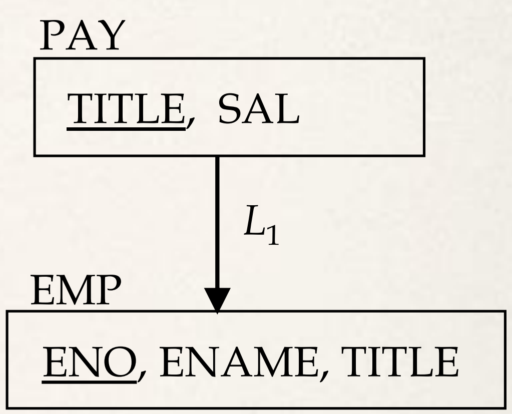
owner(L_3) = PROJ
member (L_3) = ASG

DHF – Definition

Given

- a relation S fragmented into $F_S = \{ S_1, S_2, \ldots, S_w \}$ and
- a link L where $owner(L)=S$ and $member(L)=R$,

the derived horizontal fragments of R are defined as $R_i = R \ltimes S_i \; (S_i \in F_S)$



PAY

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

PAY_1

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000

PAY_2

TITLE	SAL
Mech. Eng.	27000
Programmer	24000

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

EMP_1

ENO	ENAME	TITLE
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E7	R. Davis	Mech. Eng.

EMP_2

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E8	J. Jones	Syst. Anal.

$PAY_1 = \sigma_{SAL<30000}(PAY)$
$PAY_2 = \sigma_{SAL<30000}(PAY)$

$EMP_1 = EMP \ltimes PAY_1$
$EMP_2 = EMP \ltimes PAY_2$

ASG could be fragmented either into $ASG_i = ASG \ltimes EMP_i$ or $ASG_i = ASG \ltimes PROJ_i$ – the choice depends on applications

HF – Correctness

- Completeness (info is entirely preserved)
  - PHF: completeness follows from the way minterms are built (**exaustively**)
    - NOTICE: The book says something different
- Reconstruction
  - If relation R is fragmented into $F_R = \{R_1, R_2, \ldots, R_r\}$

$$R = \cup_{\forall R_i \in FR} R_i$$

- Disjointness
  - PHF: minterms are **mutually exclusive** by construction
- Completeness and disjointness for DHF
  - Both come from **integrity constraints** of foreign keys and from completeness/disjointness of PHF
    - fragmentation propagates from *owner* to *member* following one-to-many associations; thus, each tuple of *member* is associated with exactly 1 tuple of *owner* (a NOT NULL constraint must be defined on the foreign key in the *member* relation that refer to the *owner* relation); by disjointness and completeness of PHF, such tuple of owner appears in exactly 1 fragment of owner

Vertical Fragmentation

- Has been studied within the centralized context
  - design methodology
  - physical clustering
- Choose a partition $P = \{ P_1, P_2, \ldots, P_n \}$ of the set of attribute of relation. Then,

$$F = \{ R_i \mid R_i = \Pi_{P_i \cup key}(R) \text{ and } P_i \in P \}$$

  where *key* is the set of key attributes: they are replicated in each fragment
- More difficult than horizontal, because more alternatives exist (more than exponentially many)
- The problems boils down to finding the best partition
  - Number of elements of the partition
  - Distribution of attributes among elements of the partition

  Two approaches :
  - Grouping (bottom-up) – attributes to fragments
  - Splitting (top-down) – relation to fragments
    - preferable for 2 reasons
      - close to the design approach
      - optimal solution is more likely to be close to the full relation than to the fully fragmented situation

VF – The General Idea

- The idea is to group together attributes that are accessed together by queries
- Partition is guided by a measure of affinity (“togetherness”)

VF – Information Requirements

- Application Information
  - Attribute usage values
    - Given a set of queries $Q = \{q_1, q_2, \ldots, q_q\}$ that will run on the relation $R[A_1, A_2, \ldots, A_n]$,
      - (the 80/20 rule can be used here, too: select the most active 20% of queries only)

$$use(q_i, A_j) = \begin{cases} 1 & \text{if attribute } A_j \text{ is referenced by query } q_i \\ 0 & \text{otherwise} \end{cases}$$

$use(q_i, \bullet)$ can be defined accordingly

VF – Definition of *use*(q_i,A_j)

Consider the following 4 queries for relation PROJ

q_1: **SELECT** BUDGET
FROM PROJ
WHERE PNO=Value

q_2: **SELECT** PNAME,BUDGET
FROM PROJ

q_3: **SELECT** PNAME
FROM PROJ
WHERE LOC=Value

q_4: **SELECT** **SUM**(BUDGET)
FROM PROJ
WHERE LOC=Value

Let A_1= PNO
A_2= PNAME
A_3= BUDGET
A_4= LOC

	A_1	A_2	A_3	A_4
q_1	1	0	1	0
q_2	0	1	1	0
q_3	0	1	0	1
q_4	0	0	1	1

VF – Affinity Measure $aff(A_i,A_j)$

The attribute affinity measure between two attributes A_i and A_j of a relation $R[A_1, A_2, \ldots, A_n]$ with respect to the set of applications $Q = (q_1, q_2, \ldots, q_q)$ is defined as follows :

$$aff(A_i, A_j) = \sum_{\substack{\text{all queries } q \\ \text{that access} \\ \text{both } A_i \text{ and } A_j}} \; \sum_{\text{all sites } s} ref_s(q) * acc_s(q)$$

where

- $acc_s(q)$ = # execution of q at s in a given period
- $ref_s(q)$ = # of accesses for each execution of q at site s

VF – Calculation of *aff*(A_i, A_j)

Assume each query in the previous example accesses the attributes once during each execution

- Also assume the access frequencies
- Then, *aff*(A_1, A_3) = 15*1 + 20*1+10*1 = 45
- *aff*(. , .) is stored in the **attribute affinity matrix** *AA*
- Any clustering algorithm based on the attribute affinity values
  - Bond energy algorithm
  - Neural network
  - Machine learning
  - **(no details here)**

	S_1	S_2	S_3
q_1	15	20	10
q_2	5	0	0
q_3	25	25	25
q_4	3	0	0

	A_1	A_2	A_3	A_4
A_1	45	0	45	0
A_2	0	80	5	75
A_3	45	5	53	3
A_4	0	75	3	78

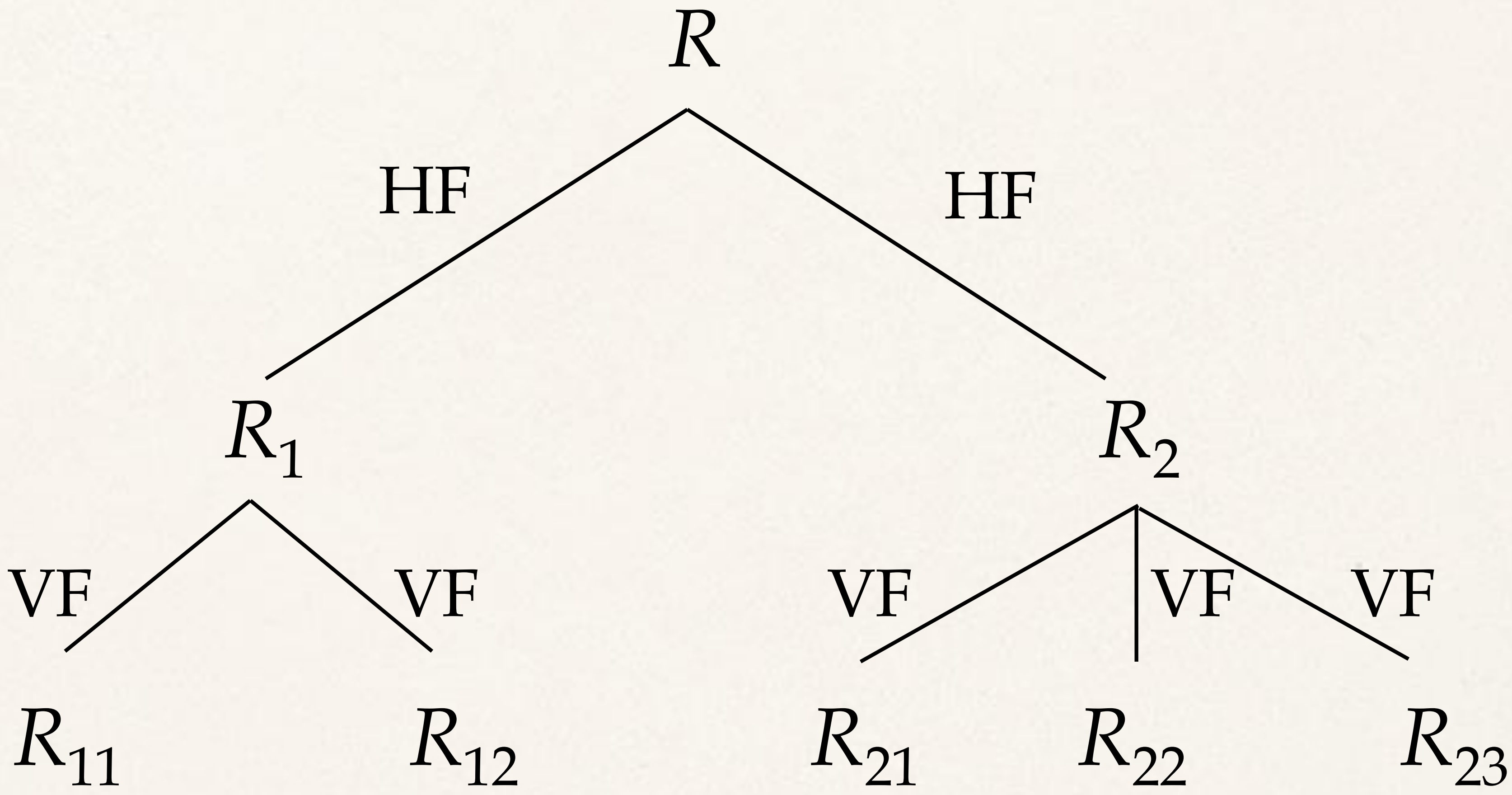
VF – Correctness

- Completeness and disjointness follow from the completeness and disjointness of the clustering algorithm
- Reconstruction can be achieved by joining the fragments
  - ➡ Let $F_R = \{R_1, R_2, \ldots, R_r\}$ be the vertical fragmentation obtained for R

$$R = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_r$$

Hybrid Fragmentation

Hybrid fragmentation, aka *mixed* or *nested fragmentation*



To reconstruct ***R***: start from the leaves and move upward applying fragmentation reconstruction methods depending on fragmentation types

Fragment Allocation

- Fragment allocation concerns distribution of resources across network nodes
  - Assignment (possibly with replications) of fragments to sites
- Problem formaliation
  - Given

    $F = \{F_1, F_2, \ldots, F_n\}$ fragments

    $S = \{S_1, S_2, \ldots, S_m\}$ network sites

    $Q = \{q_1, q_2, \ldots, q_q\}$ applications

    Find the best ("optimal") distribution of fragments in F among sites in S
- Optimality
  - Minimal cost
    - Communication, Storage (of F_i at site s_j), Querying (F_i at site s_j , from site s_k), Updating (F_i at all sites where it is replicated, from site s_k)
  - Performance
    - Response time and/or throughput
  - Can be formulated as an operations research problem

```
min (tot. cost)
s.t. response time, storage, and processing constraints
```

    - techniques and heuristics from the field apply (no optimal solution, NP-hard)

Data directory

- Data directory (aka. data dictionary or catalog)
- Both in classic (centralized) and distributed DB, it stores metadata about DB
  - Centralized context
    - Schema (relation metadata) definitions
    - Usage statistics
    - Memory usage
    - ...
  - Distributed context
    - Info to reconstruct global view of whole DB
    - What relation/fragment is stored at which site
    - ...
- It is itself part of the DB, so considerations about fragmentation and allocation issues apply