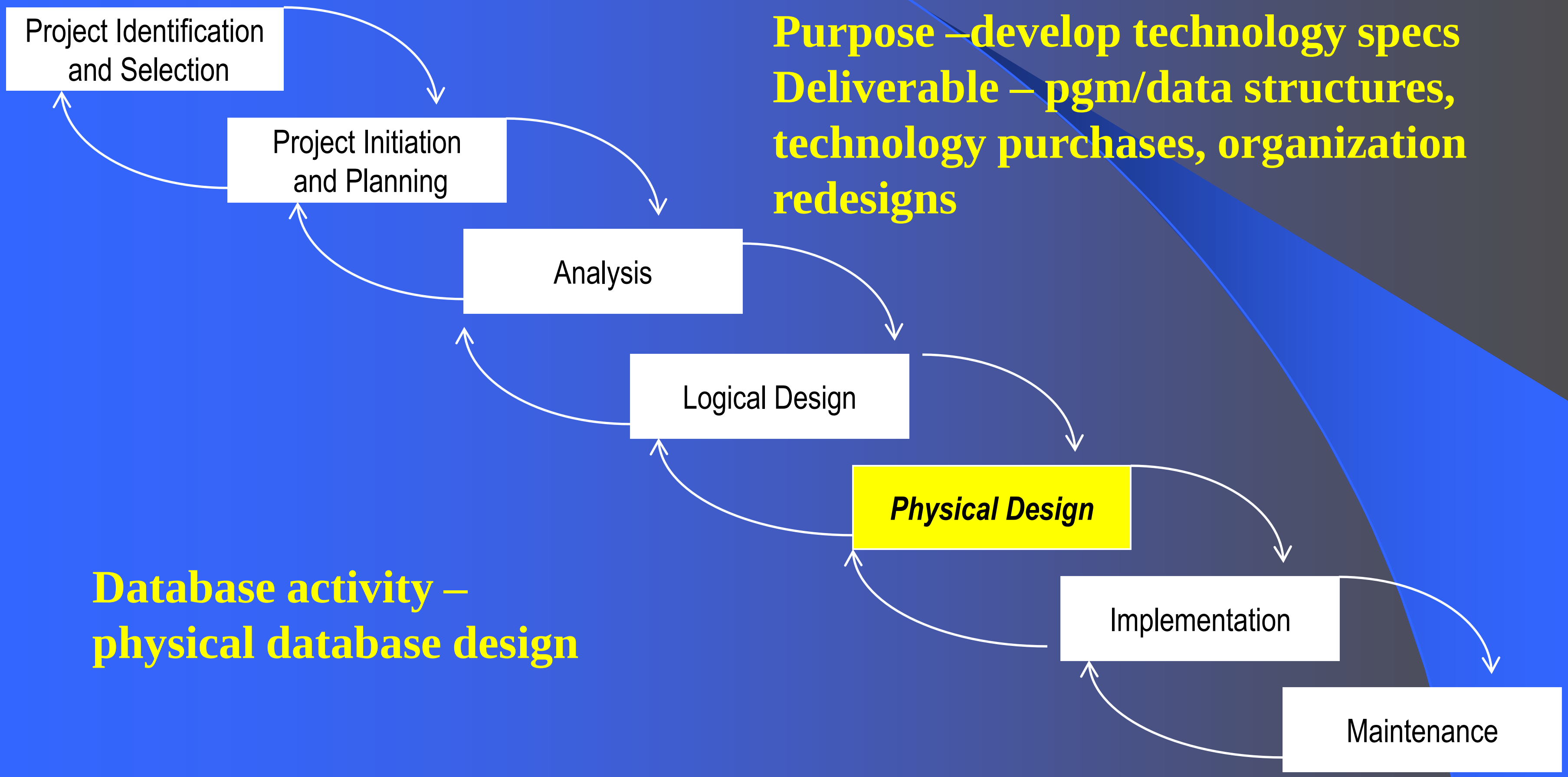
Chapter 6: Physical Database Design and Performance

Modern Database Management

6th Edition

Jeffrey A. Hoffer, Mary B. Prescott, Fred R. McFadden

The Physical Design Stage of SDLC (figures 2.4, 2.5 revisited)



Purpose –develop technology specs
Deliverable – pgm/data structures, technology purchases, organization redesigns

Database activity – physical database design

Physical Database Design

- Purpose - translate the logical description of data into the *technical specifications* for storing and retrieving data
- Goal - create a design for storing data that will provide *adequate performance* and insure *database integrity*, *security* and *recoverability*

Physical Design Process

Inputs

- Normalized relations
- Volume estimates
- Attribute definitions
- Response time expectations
- Data security needs
- Backup/recovery needs
- Integrity expectations
- DBMS technology used

Leads to

Decisions

- Attribute data types
- Physical record descriptions (doesn’t always match logical design)
- File organizations
- Indexes and database architectures
- Query optimization

Figure 6.1 - Composite usage map (Pine Valley Furniture Company)

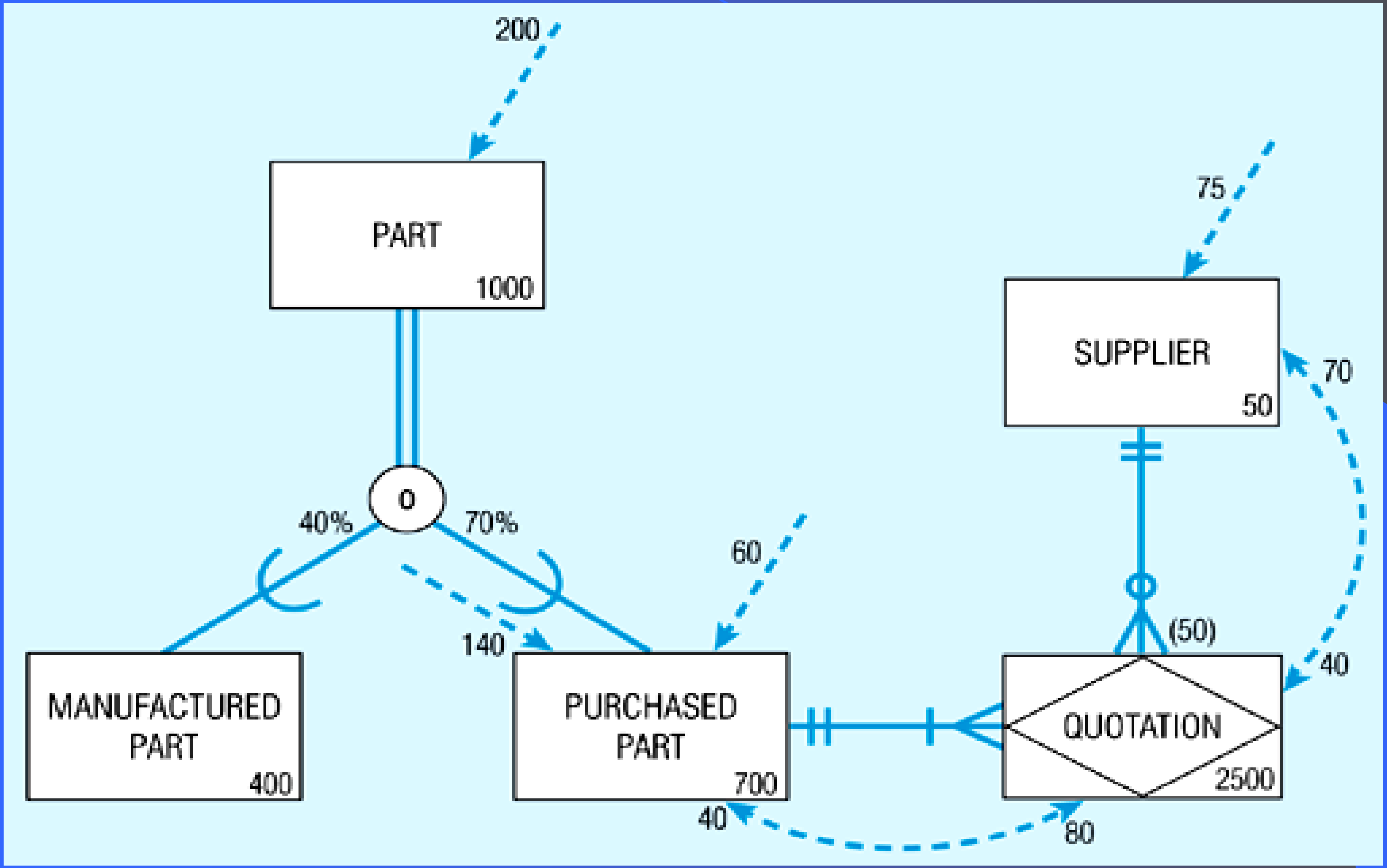


Figure 6.1 - Composite usage map (Pine Valley Furniture Company)

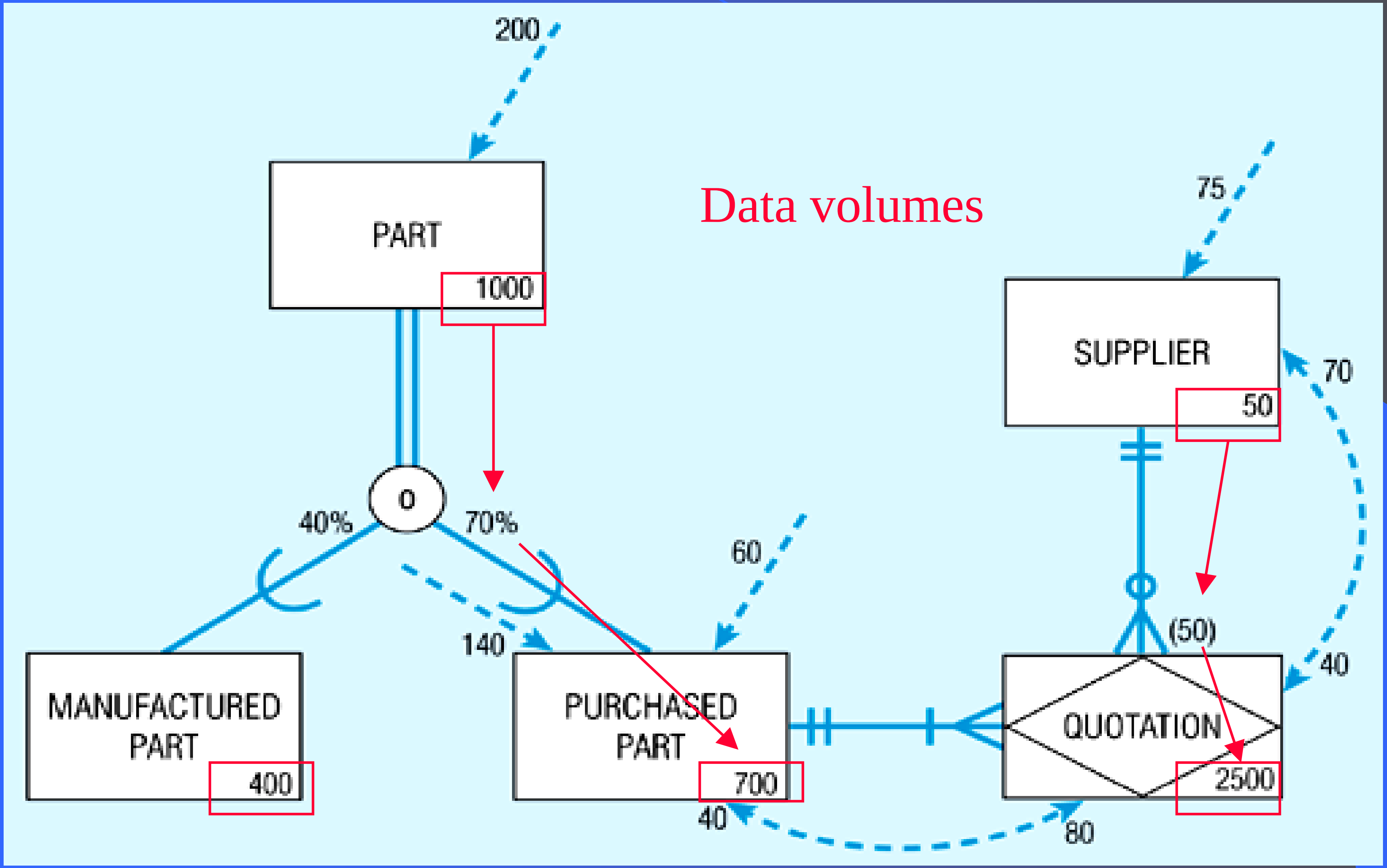


Figure 6.1 - Composite usage map (Pine Valley Furniture Company)

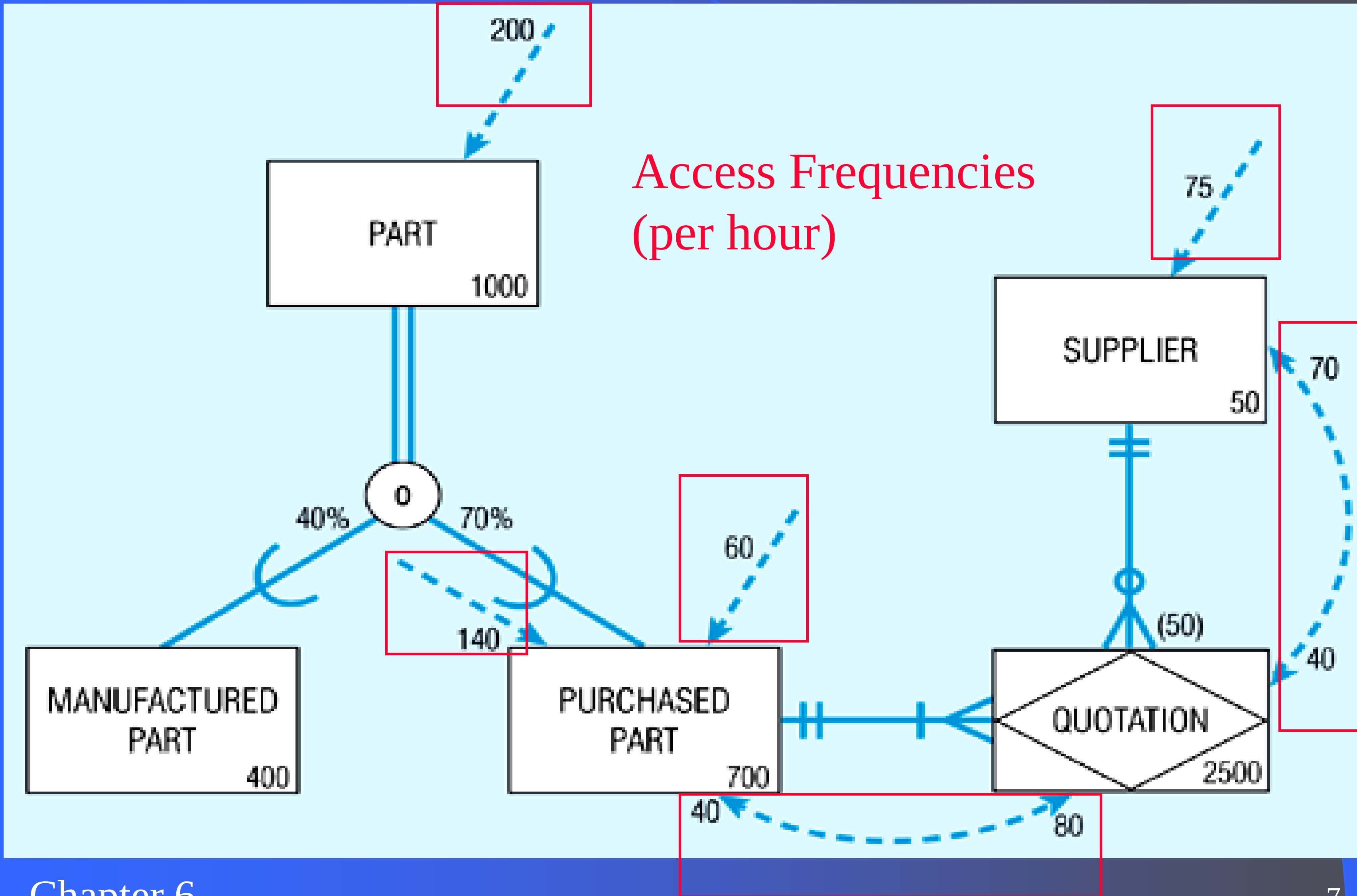
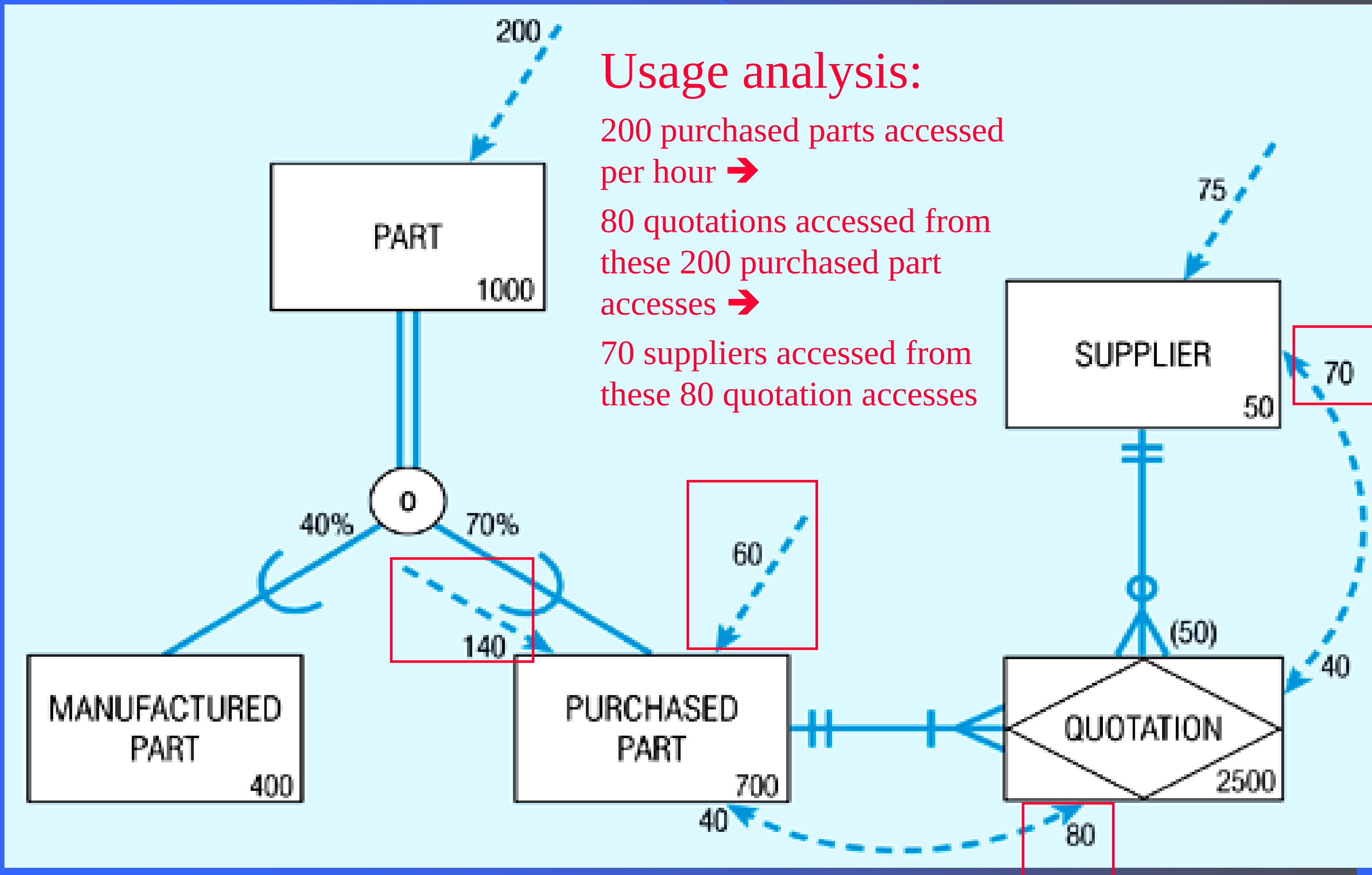


Figure 6.1 - Composite usage map (Pine Valley Furniture Company)



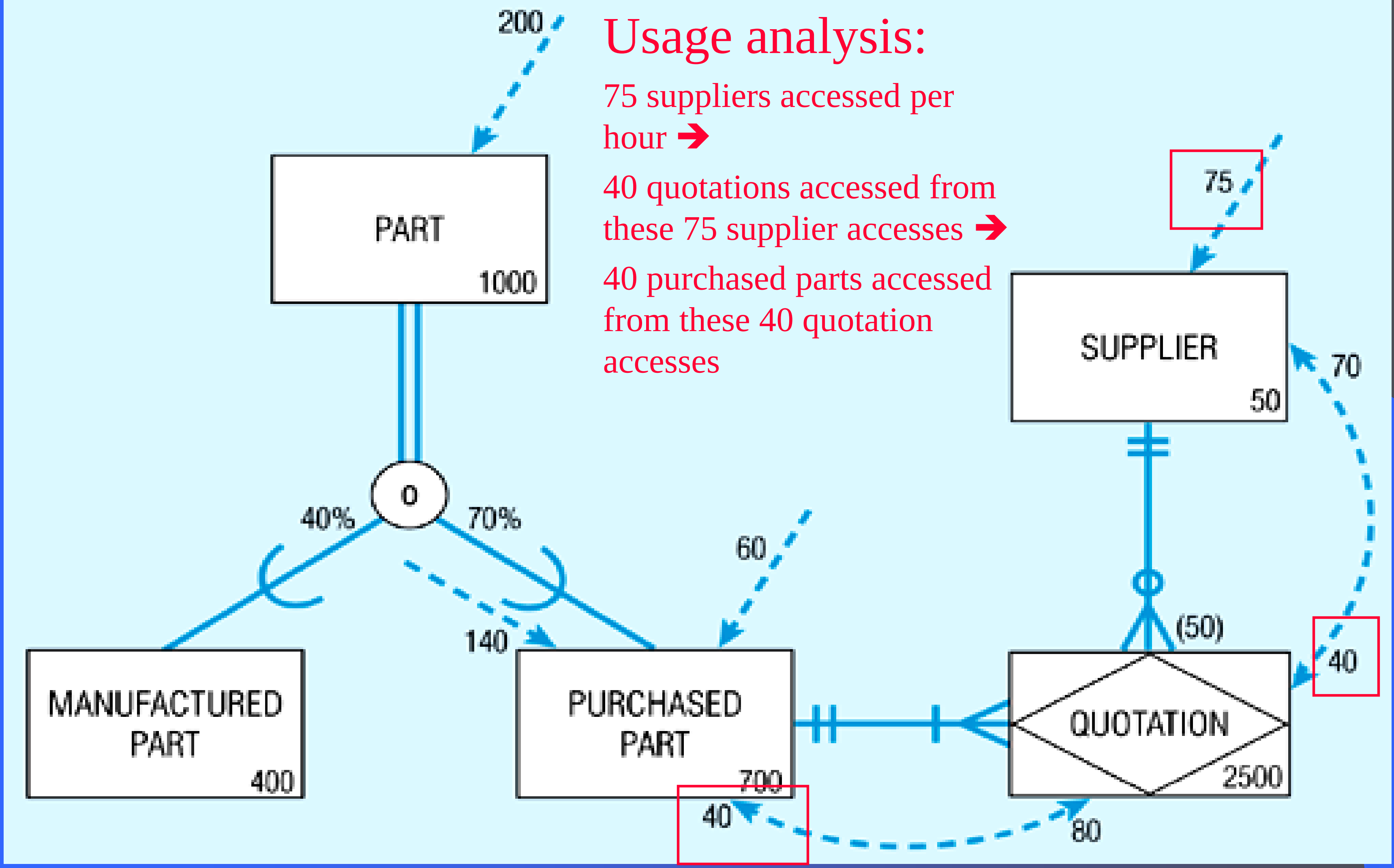
Usage analysis:

200 purchased parts accessed per hour ➔

80 quotations accessed from these 200 purchased part accesses ➔

70 suppliers accessed from these 80 quotation accesses

Figure 6.1 - Composite usage map (Pine Valley Furniture Company)



Usage analysis:

75 suppliers accessed per hour ➔

40 quotations accessed from these 75 supplier accesses ➔

40 purchased parts accessed from these 40 quotation accesses

Designing Fields

- Field: smallest unit of data in database
- Field design
  - Choosing data type
  - Coding, compression, encryption
  - Controlling data integrity

Choosing Data Types

- CHAR – fixed-length character
- VARCHAR2 – variable-length character (memo)
- LONG – large number
- NUMBER – positive/negative number
- DATE – actual date
- BLOB – binary large object (good for graphics, sound clips, etc.)

Figure 6.2
Example code-look-up table (Pine Valley Furniture Company)

PRODUCT File

Product_No	Description	Finish	…
B100	Chair	C	
B120	Desk	A	
M128	Table	C	
T100	Bookcase	B	
…	…	…	

FINISH Look-up Table

Code	Value
A	Birch
B	Maple
C	Oak

Code saves space, but costs an additional lookup to obtain actual value.

Field Data Integrity

- Default value - assumed value if no explicit value ➔ reduce data entry
- Range control – allowable value limitations (constraints or validation rules) ➔ careful, ex: causing Year 2000 problem (year in 00 to 99 only)
- Null value control – allowing or prohibiting empty fields
- Referential integrity – range control (and null value allowances) for foreign-key to primary-key match-ups

Handling Missing Data

- Substitute an estimate of the missing value (e.g. using a formula: mean/interpolation) but give mark
- Construct a report listing missing values
- In programs, ignore missing data unless the value is significant

Triggers can be used to perform these operations

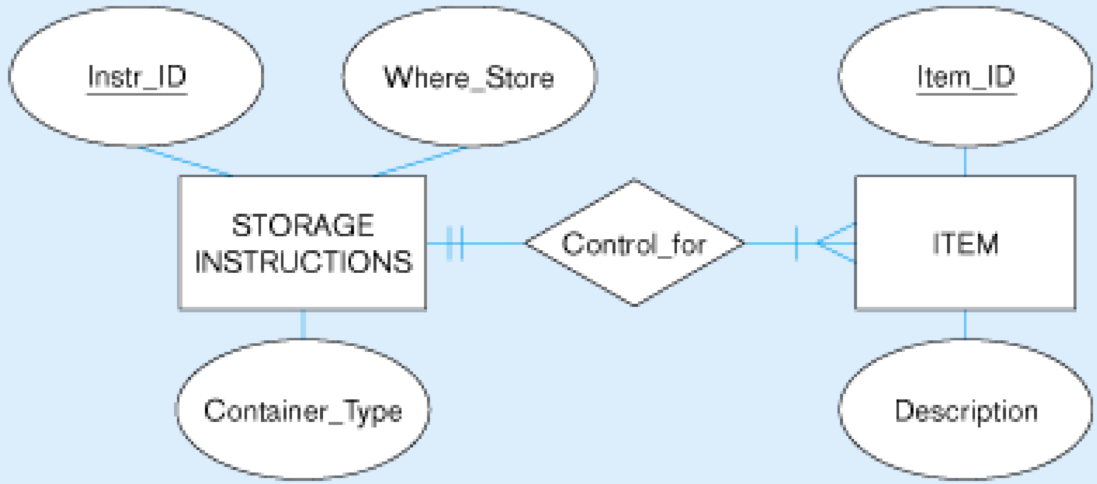
Physical Records

- Physical Record: A group of fields stored in adjacent memory locations and retrieved together as a unit
- Page: The amount of data read or written in one I/O operation
- Blocking Factor: The number of physical records per page

Denormalization

- Transforming ***normalized*** relations into ***unnormalized*** physical record specifications
- Benefits:
  - Can improve performance (speed) be reducing number of table lookups (i.e *reduce number of necessary join queries*)
- Costs (due to data duplication)
  - Wasted storage space
  - Data integrity/consistency threats
- Common denormalization opportunities
  - One-to-one relationship (Fig 6.3)
  - Many-to-many relationship with attributes (Fig. 6.4)
  - Reference data (1:N relationship where 1-side has data not used in any other relationship) (Fig. 6.5)

Fig 6.5 – A possible denormalization situation: reference data



Normalized relations:

STORAGE

Instr_ID	Where_Store	Container_Type

ITEM

Item_ID	Description	Instr_ID

Extra table access required

Denormalized relation:

ITEM

Item_ID	Description	Where_Store	Container_Type

Data duplication

Partitioning

- Horizontal Partitioning: Distributing the rows of a table into several separate files
  - Useful for situations where different users need access to different rows
  - Three types: Key Range Partitioning, Hash Partitioning, or Composite Partitioning
- Vertical Partitioning: Distributing the columns of a table into several separate files
  - Useful for situations where different users need access to different columns
  - The primary key must be repeated in each file
- Combinations of Horizontal and Vertical

Partitions often correspond with User Schemas (user views)

Partitioning

- Advantages of Partitioning:
  - Records used together are grouped together
  - Each partition can be optimized for performance
  - Security, recovery
  - Partitions stored on different disks: contention
  - Take advantage of parallel processing capability
- Disadvantages of Partitioning:
  - Slow retrievals across partitions
  - Complexity

Data Replication

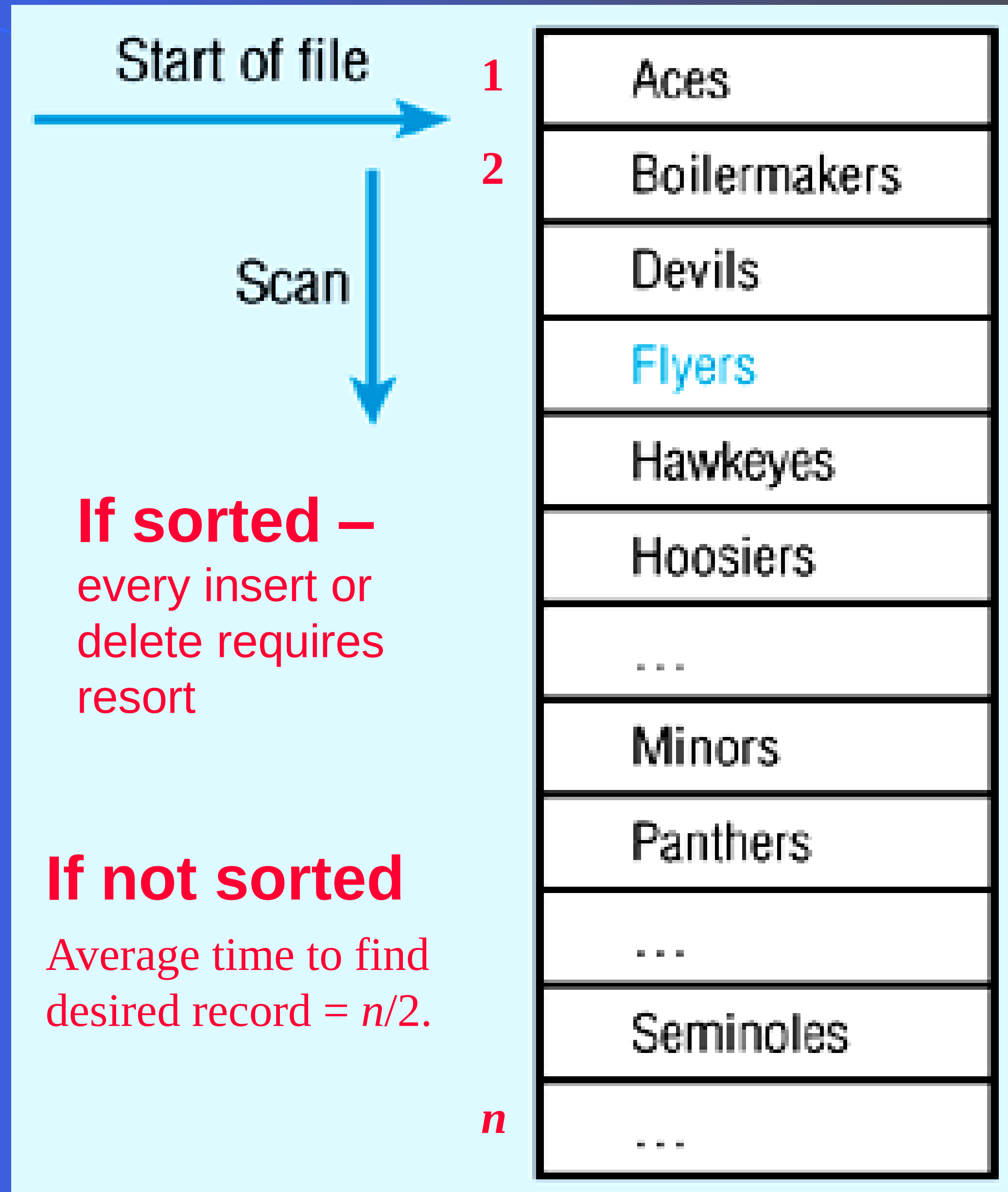
- Purposely storing the same data in multiple locations of the database
- Improves performance by allowing multiple users to access the same data at the same time with minimum contention
- Sacrifices data integrity due to data duplication
- Best for data that is not updated often

Designing Physical Files

- Physical File:
  - A named portion of secondary memory allocated for the purpose of storing physical records
- Constructs to link two pieces of data:
  - Sequential storage.
  - Pointers.
- File Organization:
  - How the files are arranged on the disk.
- Access Method:
  - How the data can be retrieved based on the file organization.

Figure 6-7 (a)
Sequential file organization

Records of the file are stored in sequence by the primary key field values.



If sorted – every insert or delete requires resort

If not sorted

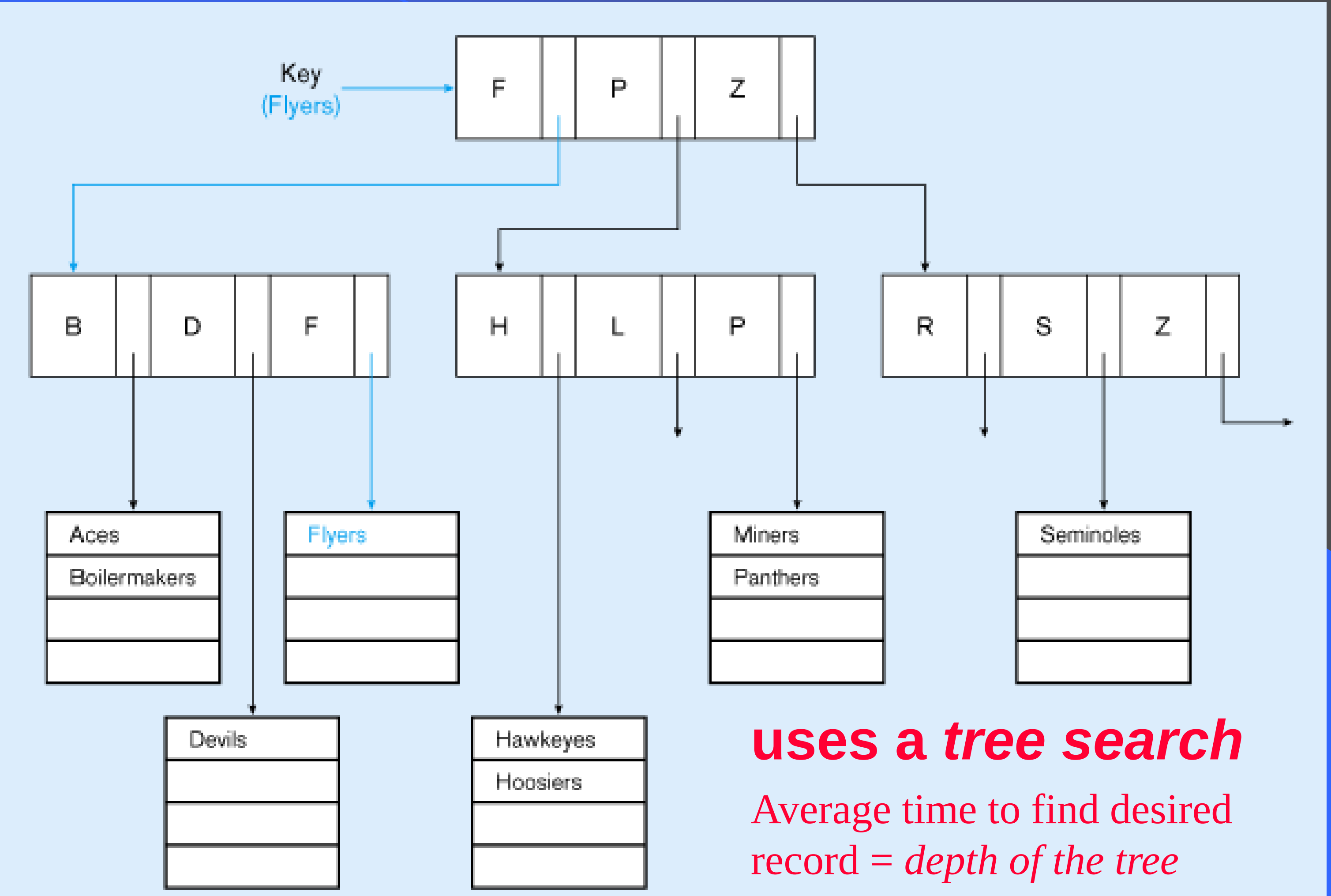
Average time to find desired record = $n/2$.

Indexed File Organizations

- Index – a separate table that contains organization of records for quick retrieval
- Primary keys are automatically indexed
- Oracle has a CREATE INDEX operation, and MS ACCESS allows indexes to be created for most field types
- Indexing approaches:
  - B-tree index, Fig. 6-7b
  - Bitmap index, Fig. 6-8
  - Hash Index, Fig. 6-7c
  - Join Index, Fig 6-9

Fig. 6-7b – B-tree index

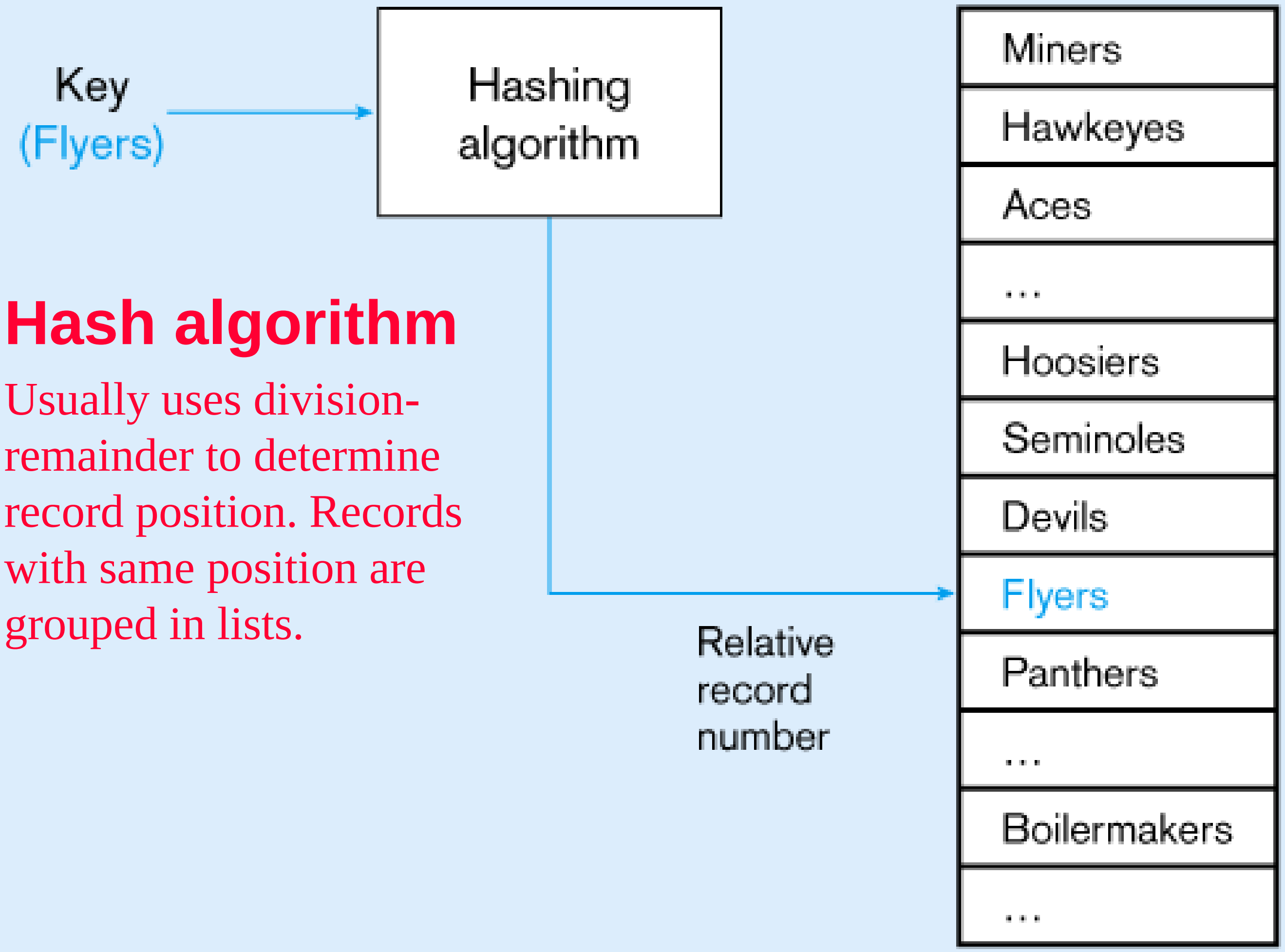
Leaves of the tree are all at same level → consistent access time



uses a *tree search*

Average time to find desired record = *depth of the tree*

Fig 6-7c
Hashed file or index organization



Hash algorithm

Usually uses division-remainder to determine record position. Records with same position are grouped in lists.

Fig 6-8

Bitmap index index organization

Bitmap saves on space requirements

Rows - possible values of the attribute

Columns - table rows

Bit indicates whether the attribute of a row has the values

Price	Product Table Row Numbers									
	1	2	3	4	5	6	7	8	9	10
100	0	0	1	0	1	0	0	0	0	0
200	1	0	0	0	0	0	0	0	0	0
300	0	1	0	0	0	0	1	0	0	1
400	0	0	0	1	0	1	0	1	1	0

Products 3 and 5 have Price $100
Product 1 has Price $200
Products 2, 7, and 10 have Price $300
Products 4, 6, 8, and 9 have Price $400

Fig 6-9 **Join** Index – speeds up join operations

Customer

RowID	Cust#	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1026	Baines	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
. . .				

Store

RowID	Store#	City	Size	Manager
20001	S4266	Dayton	K2	E2166
20002	S2654	Columbus	K3	E0245
20003	S3789	Dayton	K4	E3330
20004	S1941	Toledo	K1	E0874
. . .				

Join Index

CustRowID	StoreRowID	Common Value*
10001	20001	Dayton
10001	20003	Dayton
10002	20002	Columbus
10003	20002	Columbus
10004	20004	Toledo
. . .		

Order

RowID	Order#	Order Date	Cust#(FK)
30001	O5532	10/01/2001	C3861
30002	O3478	10/01/2001	C1062
30003	O8734	10/02/2001	C1062
30004	O9845	10/02/2001	C2027
. . .			

Customer

RowID	Cust#(PK)	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1026	Baines	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
. . .				

Join Index

CustRowID	OrderRowID	Cust#
10001	30004	C2027
10002	30002	C1062
10002	30003	C1062
10004	30001	C3861
. . .		

Clustering Files

- In some relational DBMSs, related records from different tables can be stored together in the same disk area
- Useful for improving performance of join operations
- Primary key records of the main table are stored adjacent to associated foreign key records of the dependent table
- e.g. Oracle has a CREATE CLUSTER command

Rules for Using Indexes

1. Use on larger tables
2. Index the primary key of each table
3. Index search fields (fields frequently in WHERE clause)
4. Index fields in SQL ORDER BY and GROUP BY commands
5. When there are >100 values but not when there are <30 values

Rules for Using Indexes

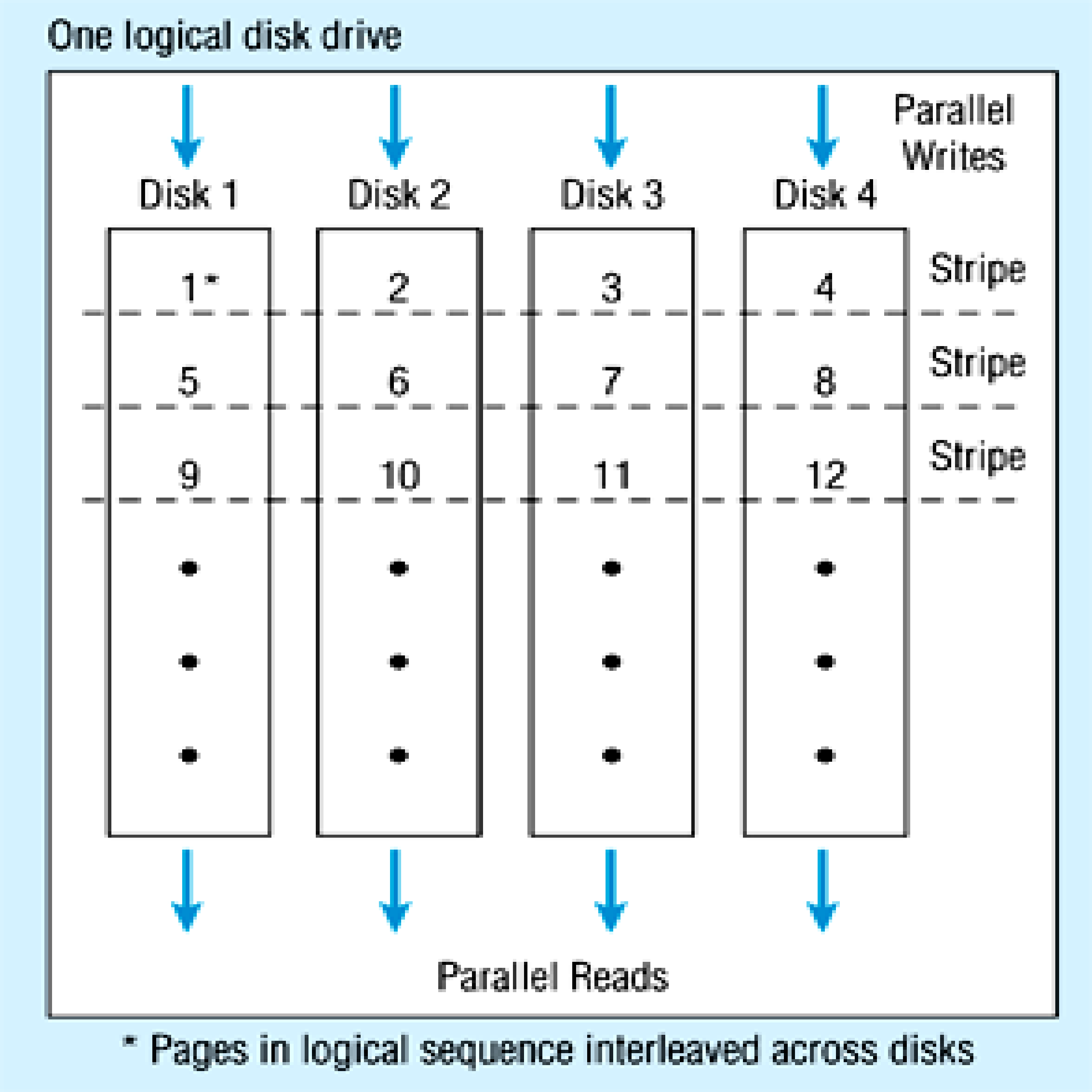
6. DBMS may have limit on number of indexes (mostly 16) per table and number of bytes per indexed field(s)
7. Null values will not be referenced from an index
8. Use indexes heavily for non-volatile databases; limit the use of indexes for volatile databases

    Why? Because modifications (e.g. inserts, deletes) require updates to occur in index files

RAID

- Redundant Array of Inexpensive Disks
- A set of disk drives that appear to the user to be a single disk drive
- Allows parallel access to data (improves access speed)
- Pages are arranged in **stripes**

Figure 6-10 – RAID with four disks and striping

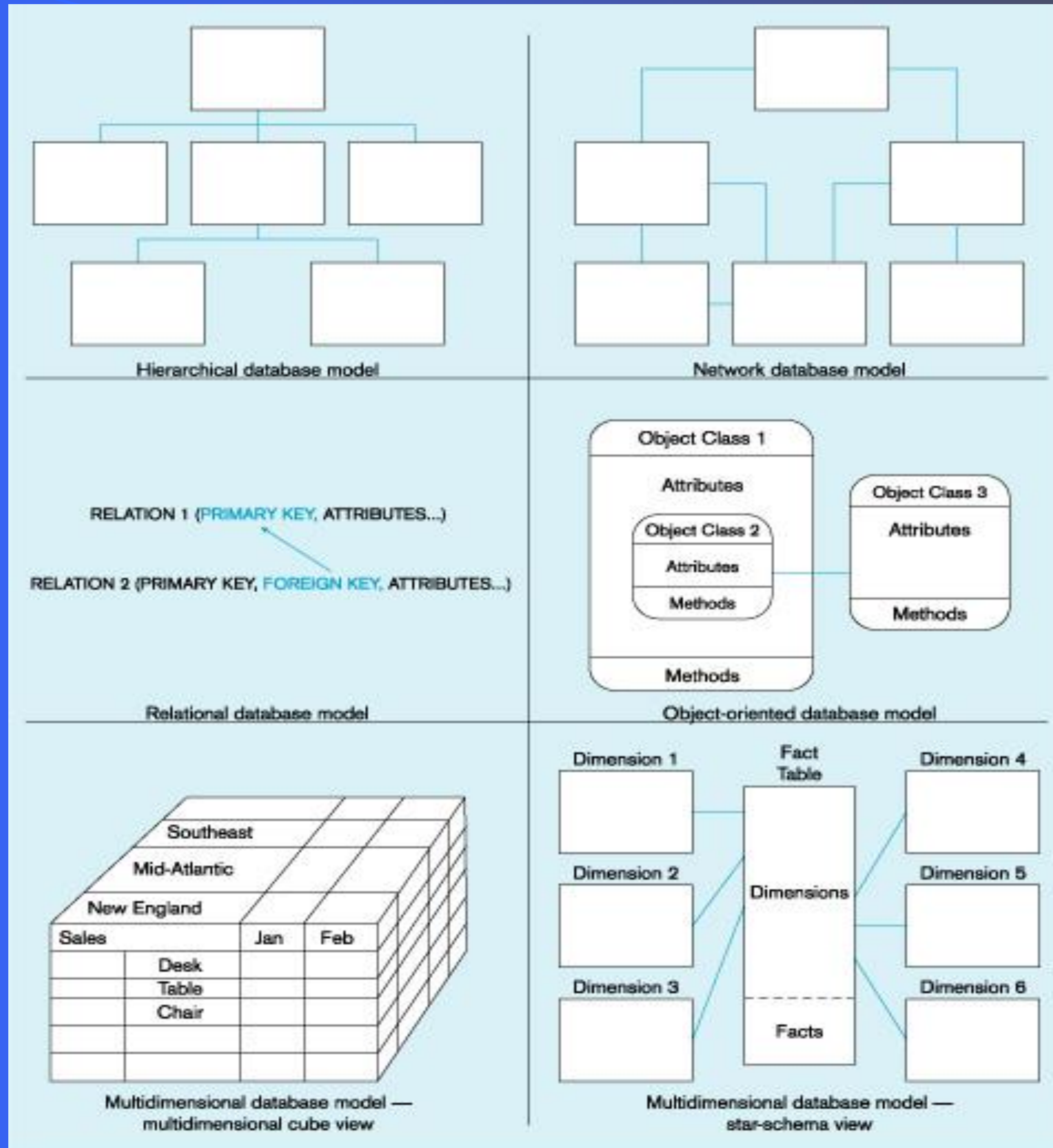


Here, pages 1-4 can be read/written simultaneously

Raid Types (Figure 6-11)

- Raid 0
  - Maximized parallelism
  - No redundancy
  - No error correction
  - no fault-tolerance
- Raid 1
  - Redundant data – fault tolerant
  - Most common form
- Raid 2
  - No redundancy
  - One record spans across data disks
  - Error correction in multiple disks– reconstruct damaged data
- Raid 3
  - Error correction in one disk
  - Record spans multiple data disks (more than RAID2)
  - Not good for multi-user environments,
- Raid 4
  - Error correction in one disk
  - Multiple records per stripe
  - Parallelism, but slow updates due to error correction contention
- Raid 5
  - Rotating parity array
  - Error correction takes place in same disks as data storage
  - Parallelism, better performance than Raid4

Database Architectures (figure 6-12)



Legacy Systems

Current Technology

Data Warehouses

Query Optimization

- Parallel Query Processing
- Override Automatic Query Optimization
- Data Block Size -- Performance tradeoffs:
  - Block contention
  - Random vs. sequential row access speed
  - Row size → match block size with physical table row size
  - Overhead → small block size produce more overhead
- Balancing I/O Across Disk Controllers

Query Optimization

- Wise use of indexes
- Compatible data types
- Simple queries
- Avoid query nesting
- Temporary tables for query groups
- Select only needed columns
- No sort without index