The logical answer: IMS Logical Relationships

Aurora Emanuela Dell'Anno

Engineering Services Architect

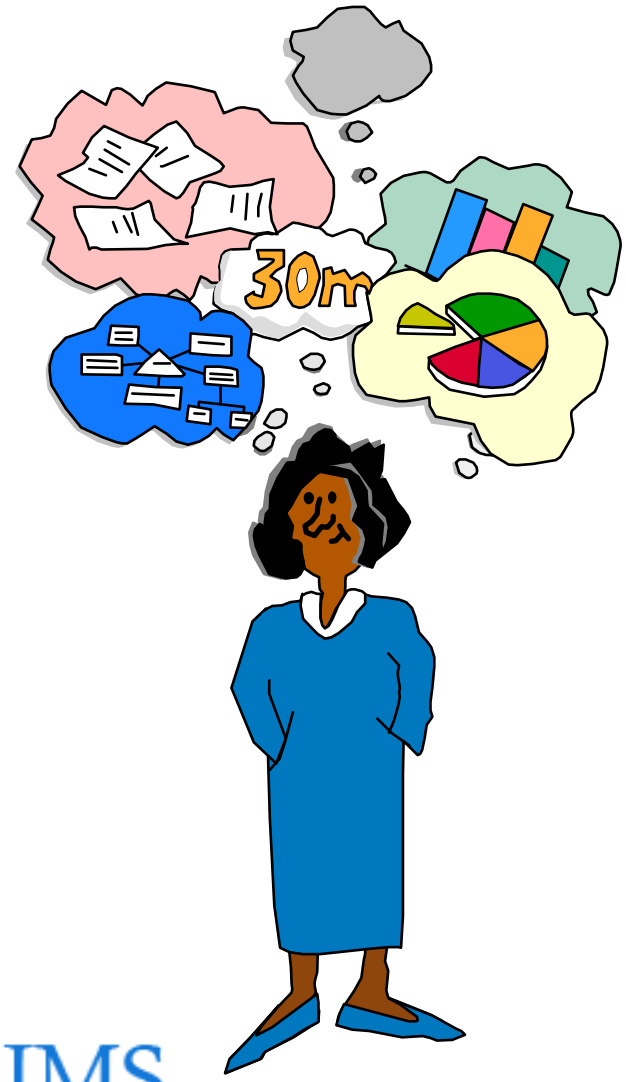
CA MSC

Virtual IMS CONNECTION

ca

Session Agenda

- Logical Relationships in IMS?
  - Why IMS DB needs Logical Relationships
- This is how we do it…
  - How to define Logical Relationships
- The way there
  - Pointers, paths and Physical and Logical DBD
- I've changed my mind…
  - Alter rules for Logical Relationships
- And what about Performance?
  - Performance Considerations
- Things that make you go UHM…



Logical Relationships in IMS? WHY?

Hierarchical data structure

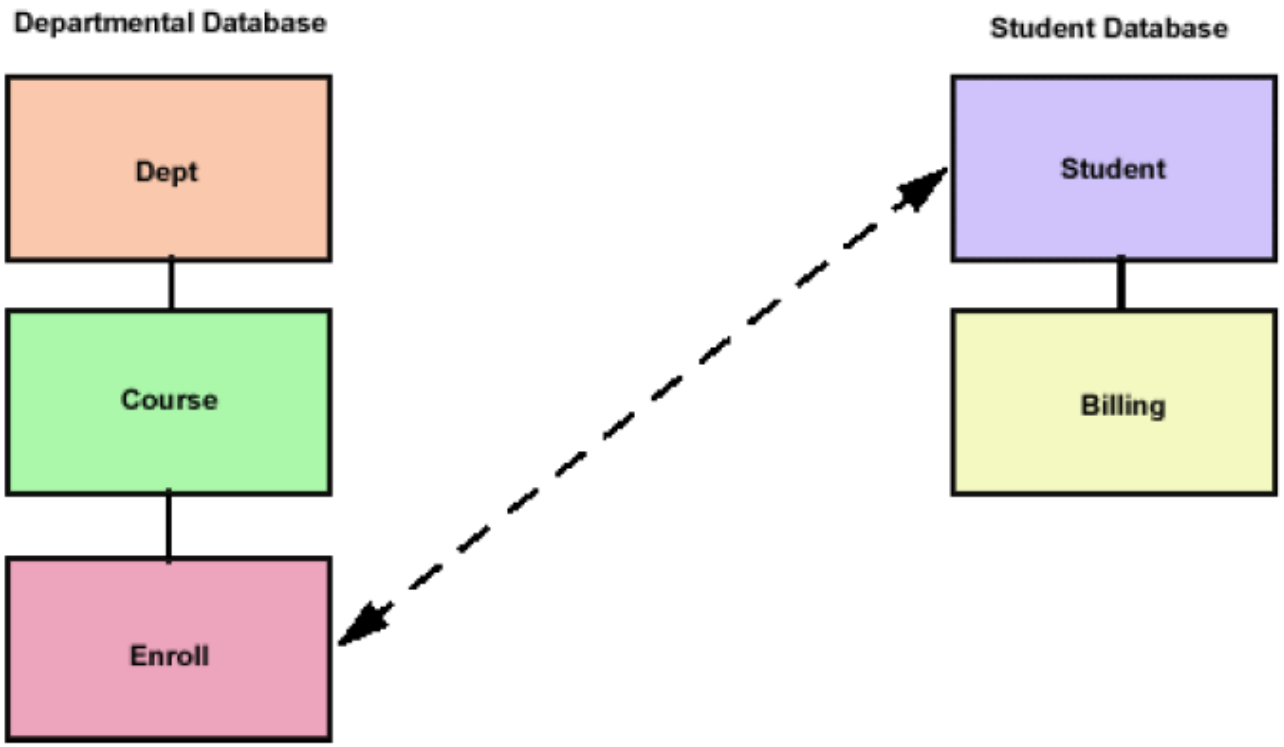


Figure 1-2: Sample hierarchical databases for department and student.

Relational data structure

Course No.	Course Title	Description	Instructor	Dept ID
HI-445566	History 321	Survey course	J. R. Jenkins	HIST
MH-778899	Algebra 301	Freshman-level	A.L. Watson	MATH
BI-112233	Biology 340	Advanced course	B.R. Sinclair	BIOL

Table 1-1: Course database in relational table format.

Student ID	Student Name	Address	Major
123456777	Jones, Bill	1212 N. Main	History
123456888	Smith, Jill	225B Baker St	Physics
123456999	Brown, Joe	77 Sunset St	Zoology

Table 1-2: Student database in relational table format.

Dept ID	Dept. Name	Chairman	Budget Code
HIST	History	J. B. Hunt	L72
MATH	Mathematics	R. K. Turner	A54
BIOL	Biology	E. M. Kale	A25

Table 1-3: Department database in relational table format.

Logical Relationships in IMS? WHY?

To access segments by a field other than the one chosen as the key

OR

To associate segments in two different dbs or hierarchies

IMS provides two very useful tools to resolve these data requirements:

- secondary indexes
- logical relationships

Database Design Considerations

- Normalization of Data
  - Helps break data into naturally associated groupings that can be stored collectively in segments in a hierarchical database
    - break individual data elements into groups based on the processing functions
    - group data based on inherent relationships between data elements

Supported databases

The following db types support logical relationships:

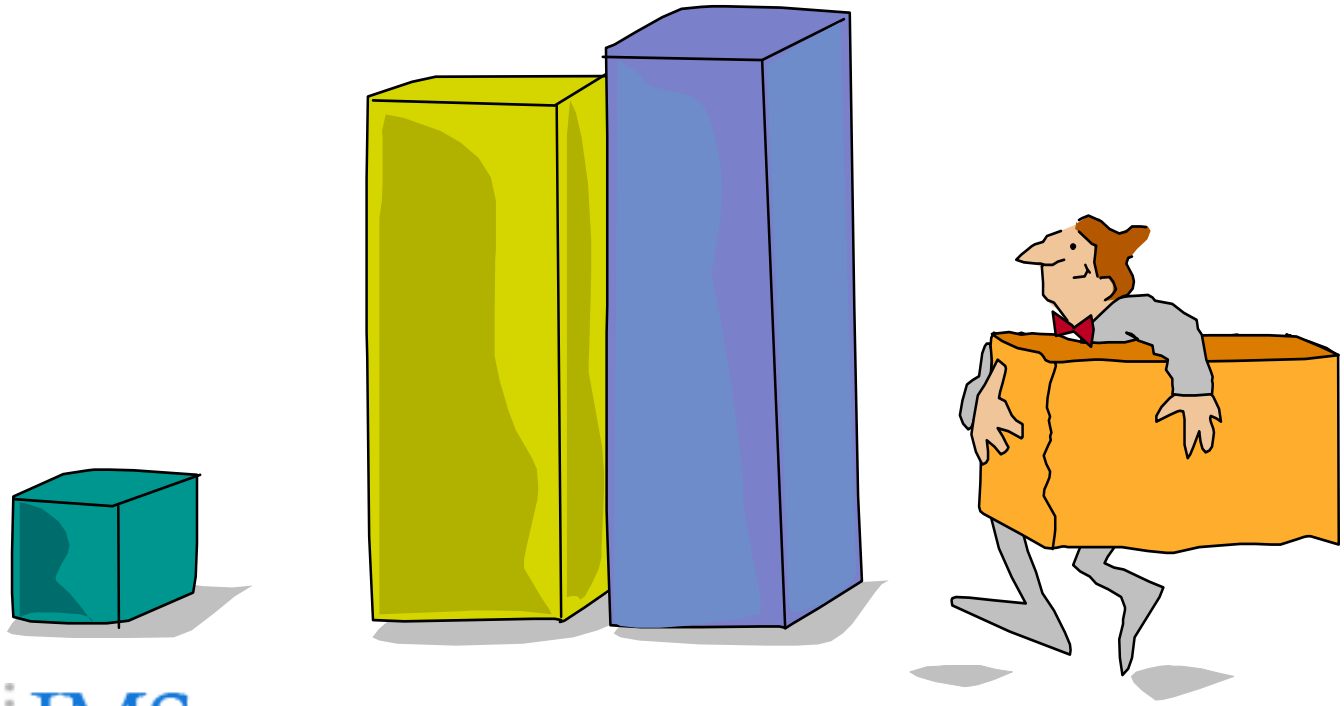
- HISAM
- HDAM
- PHDAM
- HIDAM
- PHIDAM

No logical relationships with
Fast Path DEDB or MSDB databases

Virtual IMS CONNECTION

ca

This is how we do it…

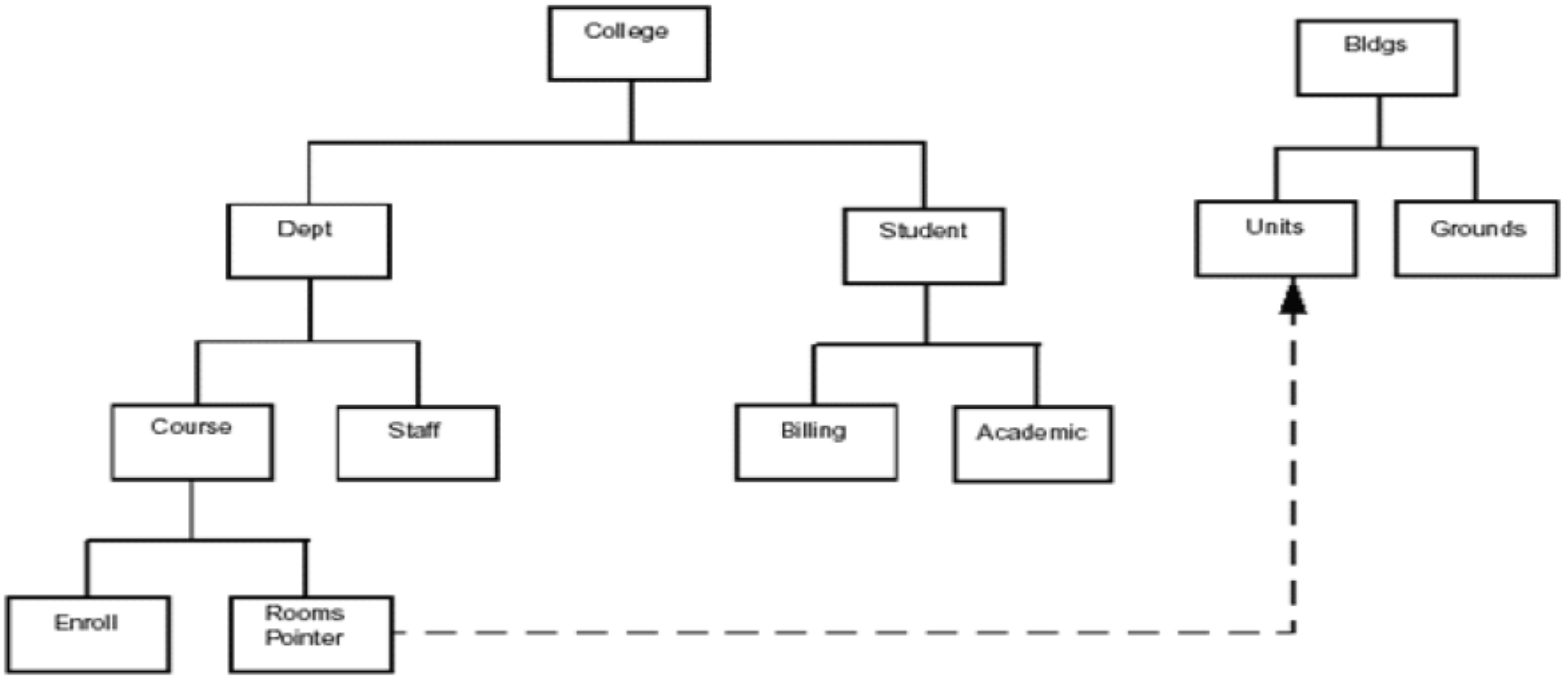


Virtual IMS CONNECTION

This is how we do it…

We create special segments that use pointers to access data in other segments

The path between the logical child and the segment to which it points is called a logical relationship



Virtual IMS CONNECTION

Logical Relationship Types

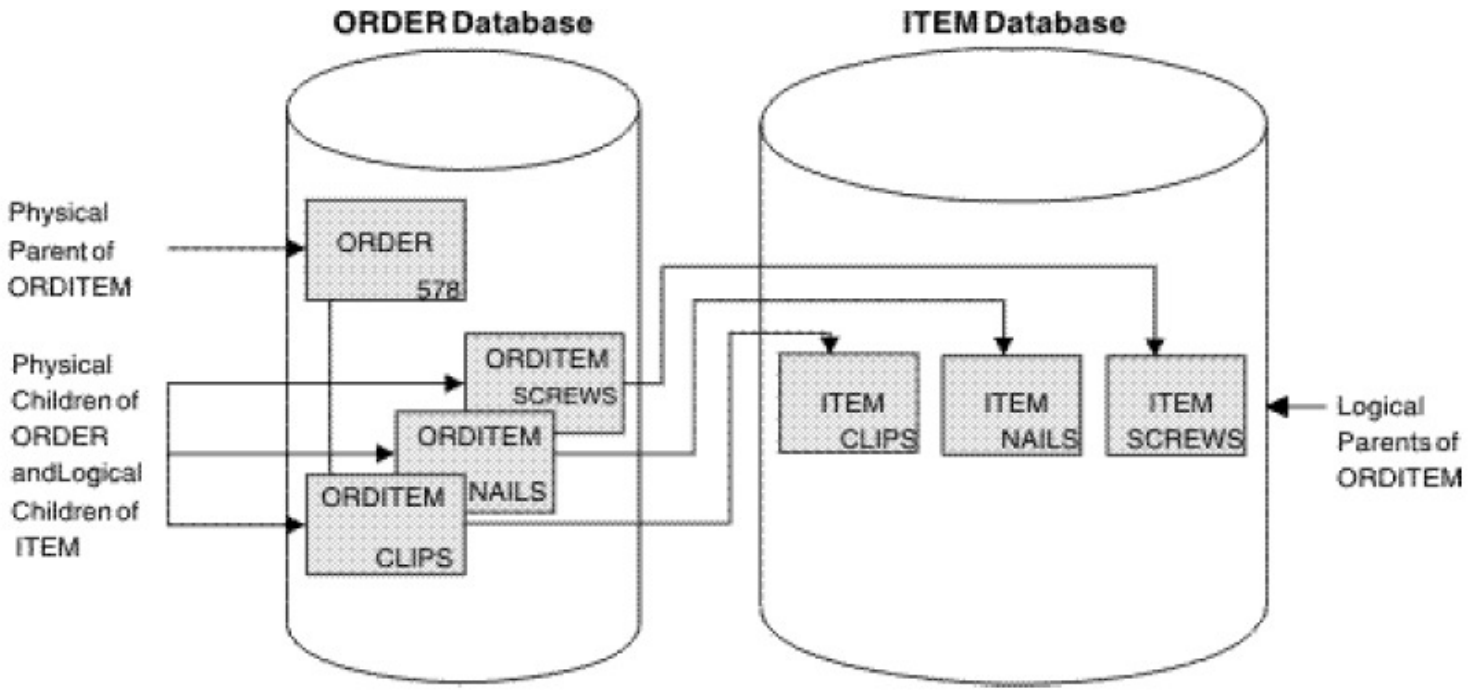
There are three types of logical relationships:

- Unidirectional
- Bidirectional physically paired
- Bidirectional virtually paired

Unidirectional relationship

Links two segment types (logical child and its logical parent) in one direction

- A one-way path is established using a pointer in the logical child

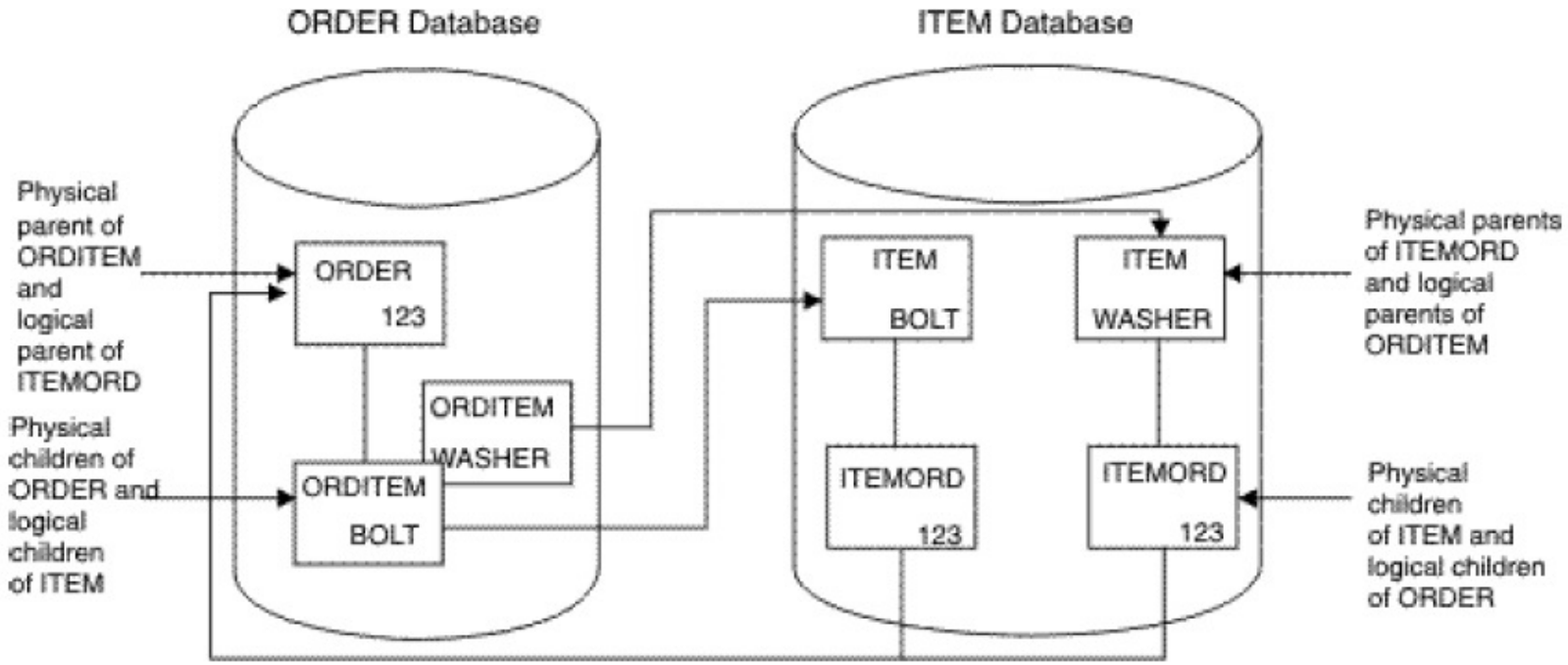


Virtual IMS CONNECTION

Bidirectional Physically Paired Logical Relationship

Links two segment types, a logical child and its logical parent, in two directions

- A two-way path is established using pointers in the logical child segments



Virtual IMS CONNECTION

Bidirectional Physically Paired Logical Relationship

Physical Pairing

Suppose that we want to be able to access data in the College database as if the data were in a segment of the Buildings database. When we access the Units segment of the Buildings database, we want to know which courses will be taught in each room. We can do this by storing the Rooms logical child segment in the Buildings database and pointing to the Course segment in the College database. As a room is assigned to a course, the Rooms logical child will be updated in both databases. We will have a bidirectional path that lets us determine what room a course will be taught in and what courses will be taught in any room. The Rooms segment is physically stored in both databases and the segments are said to be physically paired. IMS automatically updates both logical child segments when a logical parent segment occurrence is added

Bidirectional Virtually Paired Logical Relationship

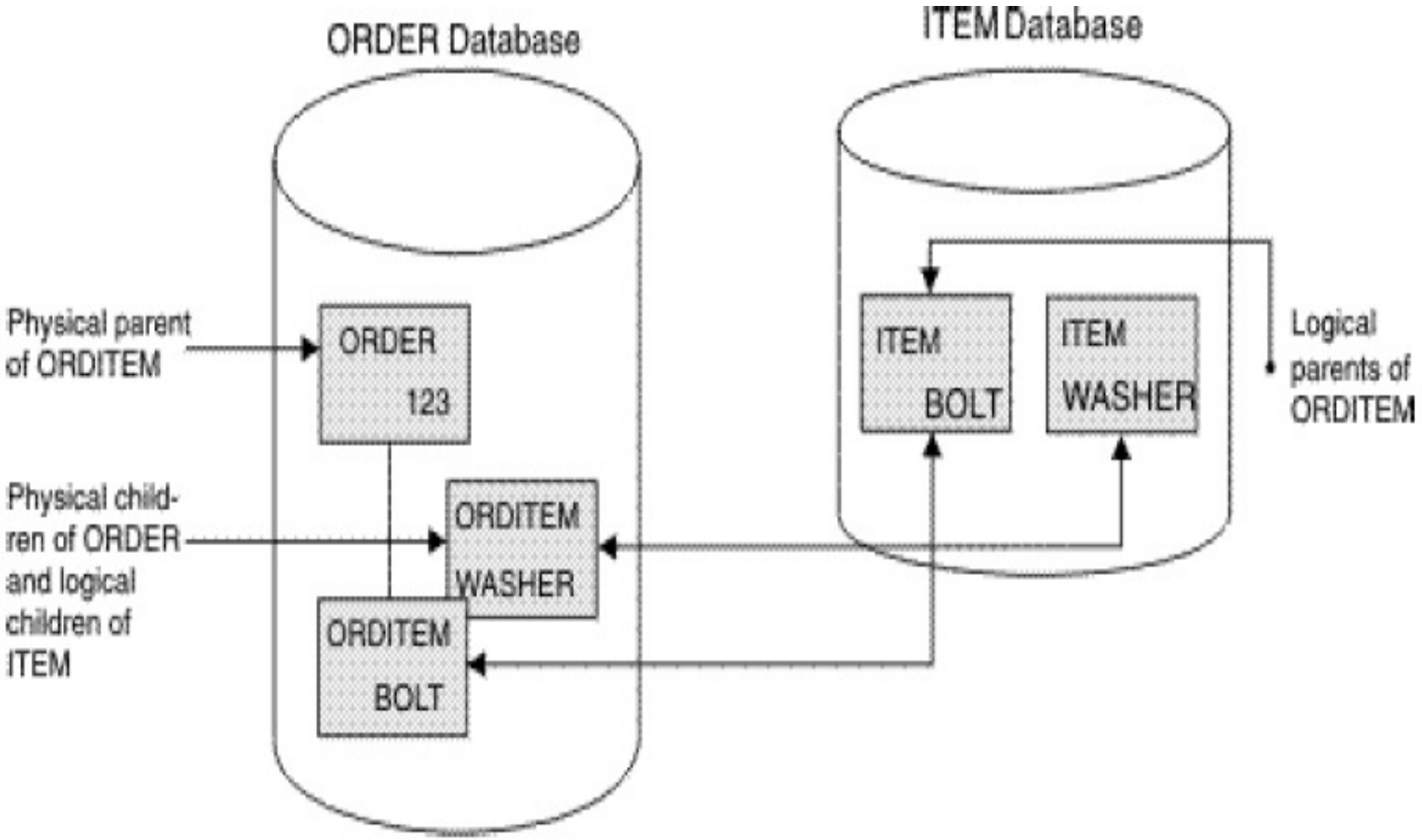
Like a bidirectional physically paired relationship:

- links two segment types, a logical child and its logical parent, in two directions, establishing a two-way path
- can be established between two segment types in the same or different dbs

A logical child segment exists only in the one db

- Going from db A to db B, IMS uses the pointer in the logical child segment
- Going from db B to db A, IMS uses the pointer in the logical parent, as well as the pointer in the logical child segment

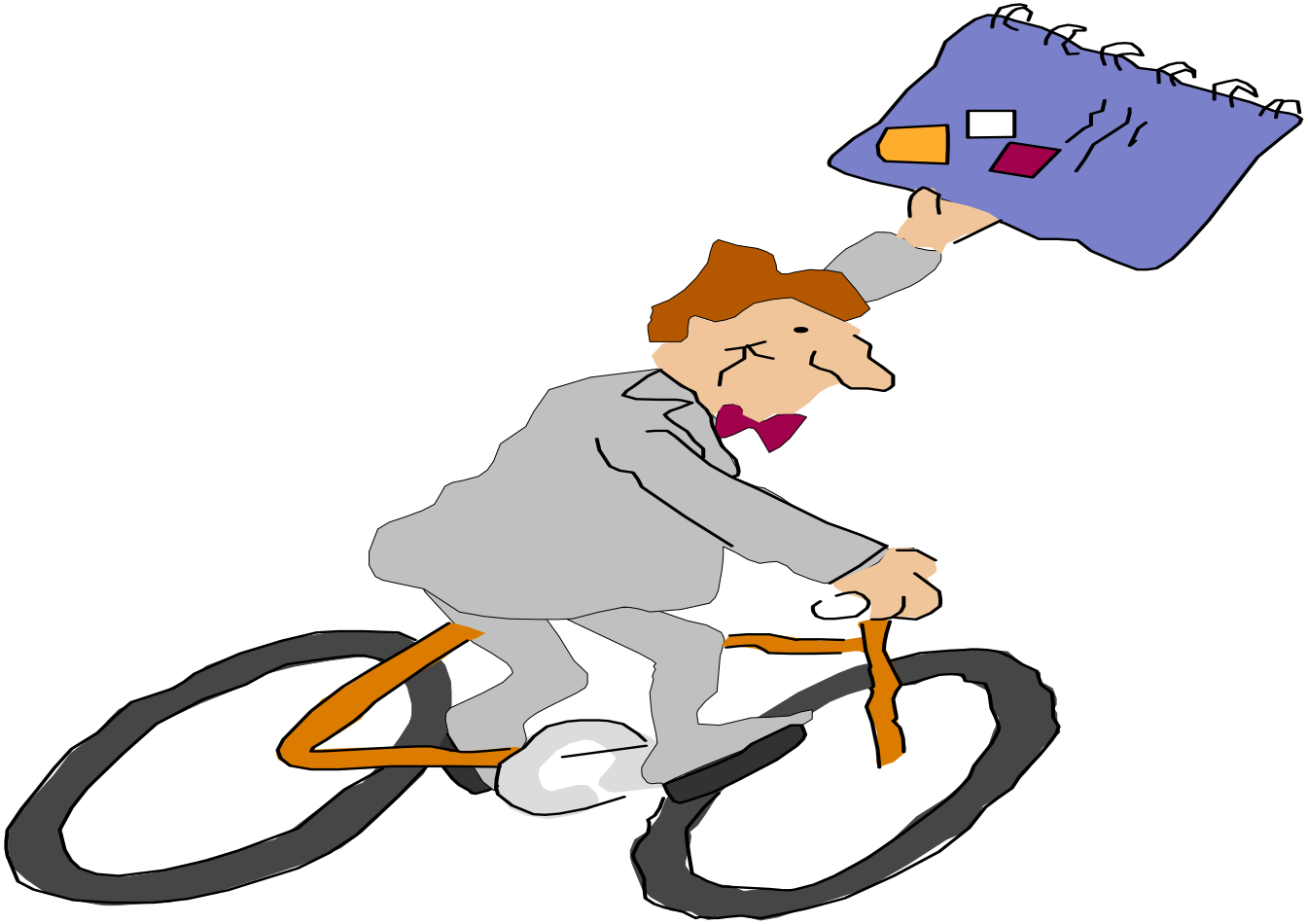
Bidirectional Virtually Paired Logical Relationship



Virtual IMS CONNECTION

ca

the way there…



Virtual IMS CONNECTION

Segment types

To establish a logical relationship, three segment types are always defined:

- physical parent
- logical parent
- logical child

Use of Pointers

Pointers used in logical relationships fall into two categories - direct and symbolic

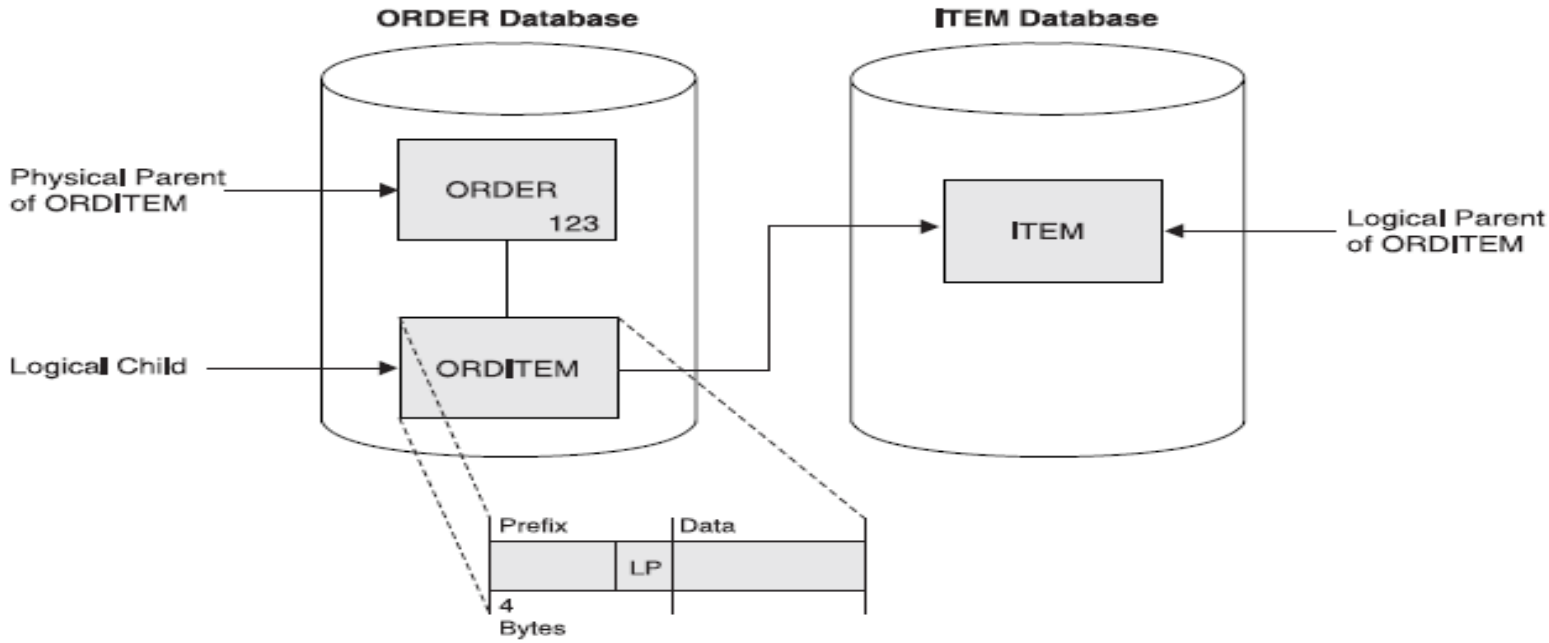
- We can implement logical relationships using both types
- A direct pointer is a true pointer

There are four types of pointers:

- logical parent (LP)
- logical child (LC)
- logical twin (LT)
- physical parent (PP)

Direct LP Pointer

- LP pointers point from the logical child to the logical parent
  - an LP pointer is in the prefix of the logical child and consists of the 4-byte direct address of the logical parent



Logical child pointers

Logical child pointers are used only for logical relationships that use virtual pairing. With virtual pairing, only one logical child exists on DASD, and it contains a pointer to a logical parent. The logical parent points to the logical child segment. Two types of logical child pointers can be used:

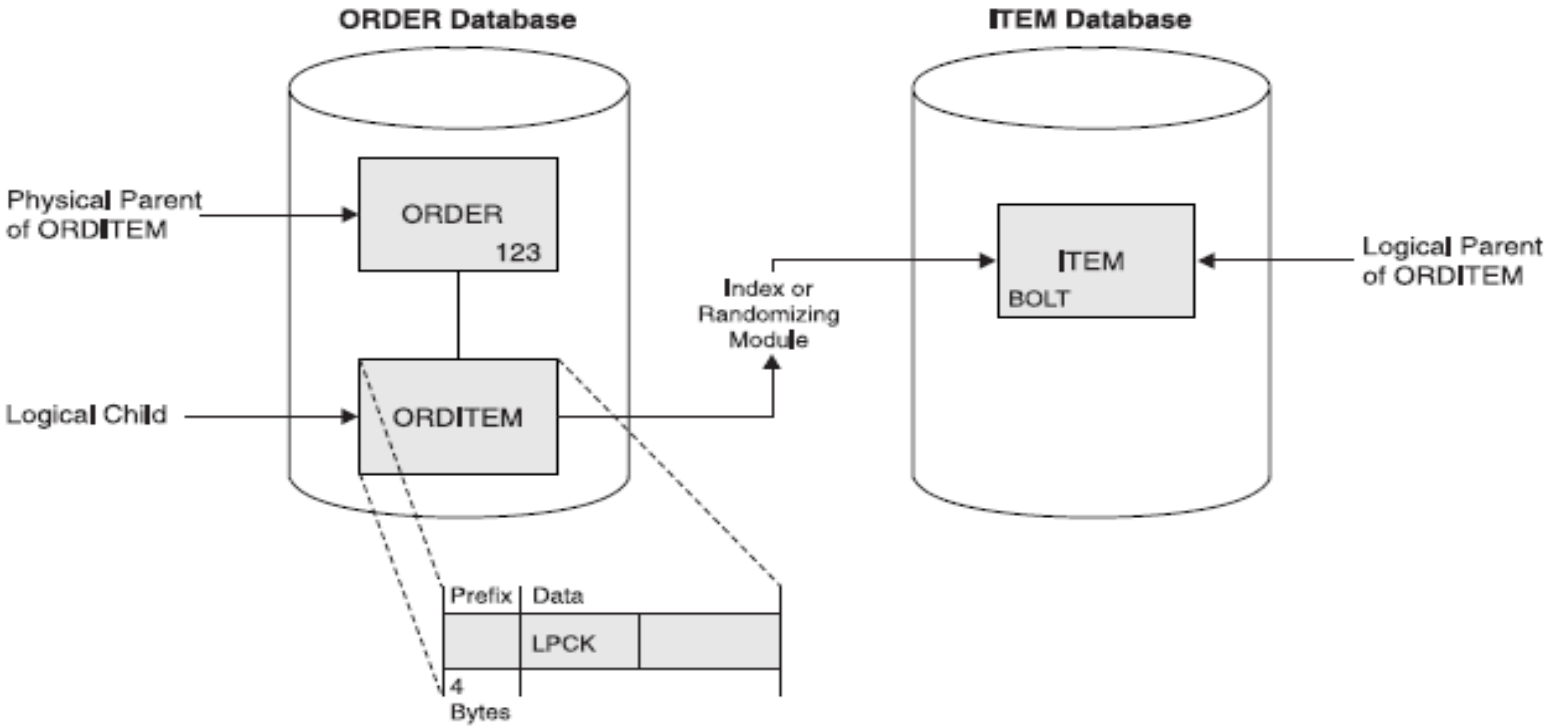
- logical child first (LCF)
- a combination of LCF and logical child last (LCL) pointers

Because LCF and LCL pointers are direct pointers, the segment they are pointing to must be in an HD database. The logical parent (the segment pointing to the logical child) must be in a HISAM or HD database. If the parent is in a HISAM database, the logical child must use a symbolic pointer to point to the parent.

Symbolic LP pointer

A symbolic LP pointer consists of the logical parent's concatenated key (LPCK)

- it can be used to point into a HISAM or HD database

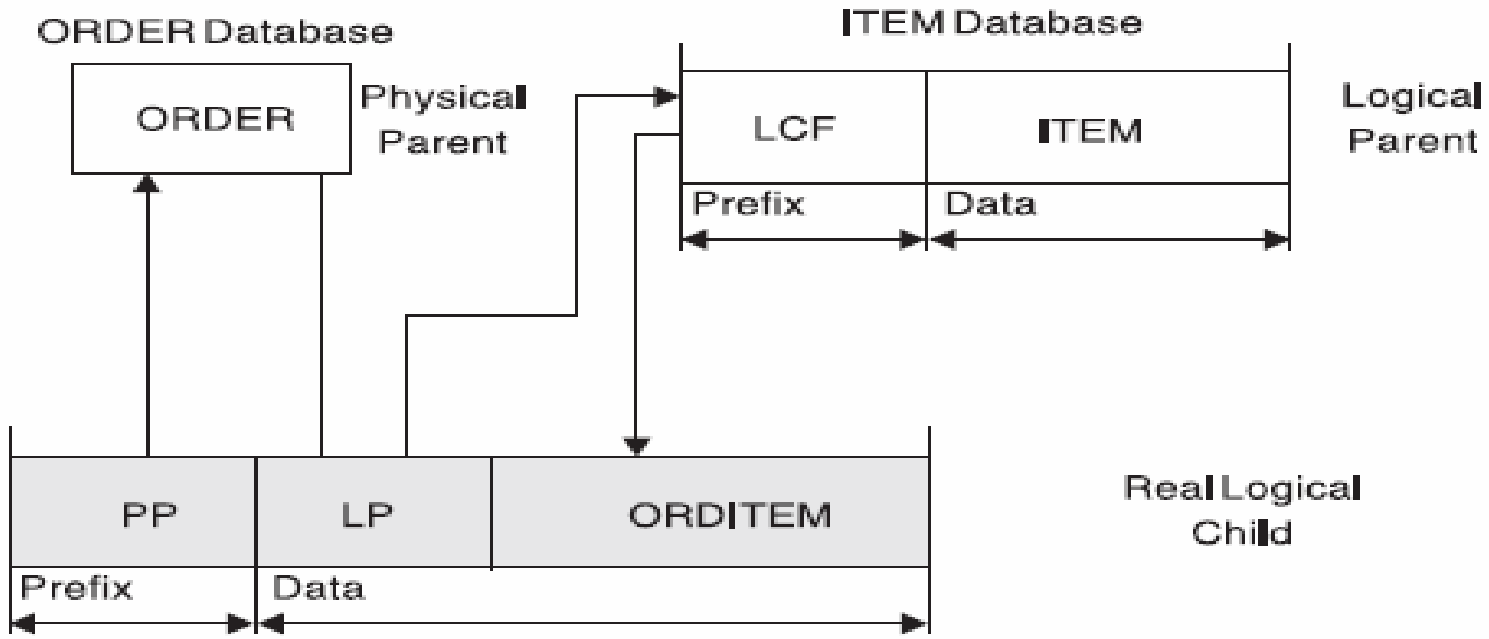


Virtual IMS CONNECTION

Physical Parent (PP) Pointer

Physical parent (PP) pointers point from a segment to its physical parent

- IMS generates PP pointers automatically for HD databases involved in logical relationships



Logical Child Pointers

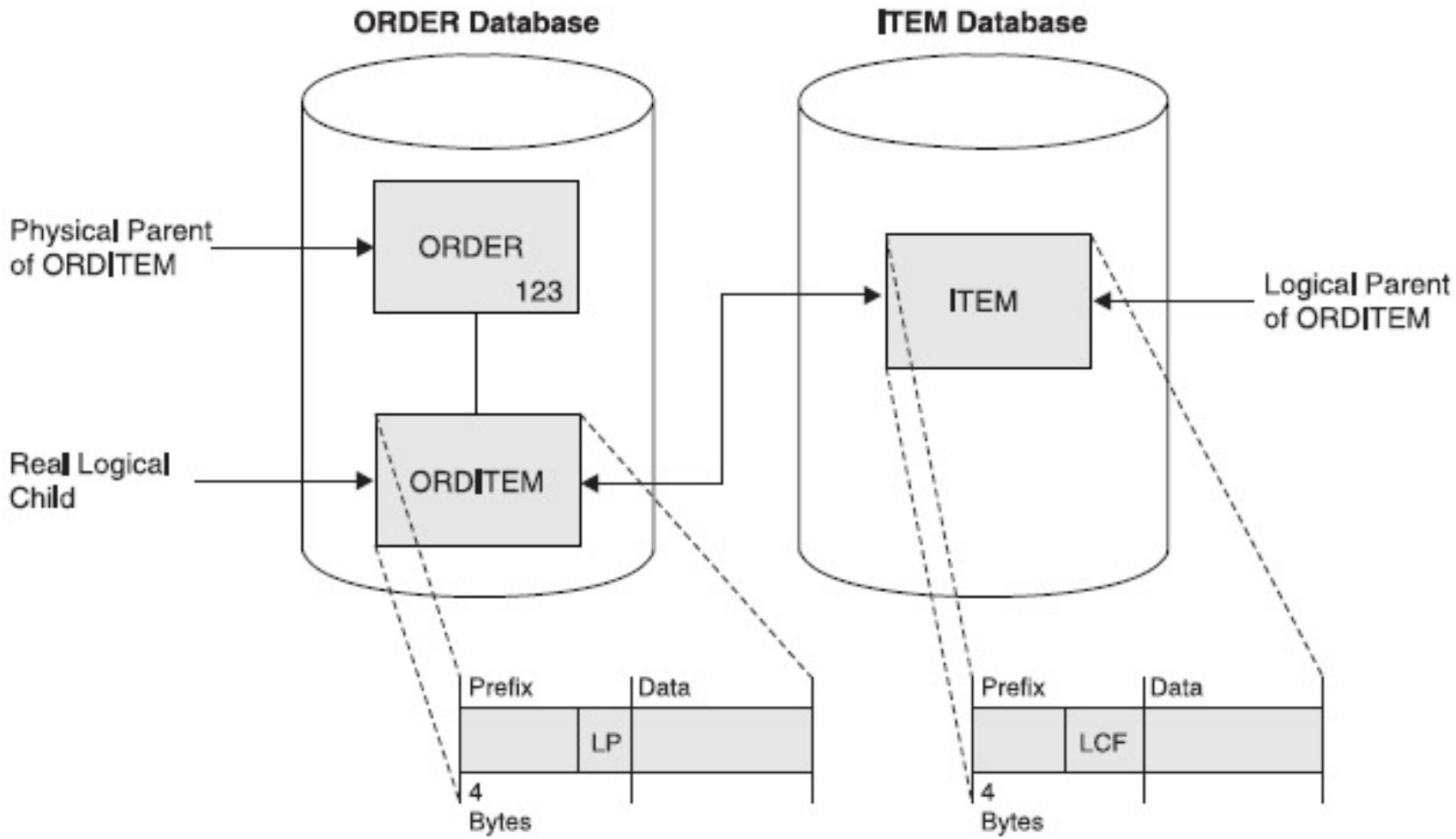
Logical child pointers are only used in logical relationships with virtual pairing

When virtual pairing is used, there is only one logical child on DASD, called the real logical child. This logical child has an LP pointer. The LP pointer can be symbolic or direct

Two types of logical child pointers can be used:

- Logical child first (LCF) pointers
- A combination of logical child first (LCF) and logical child last (LCL) pointers

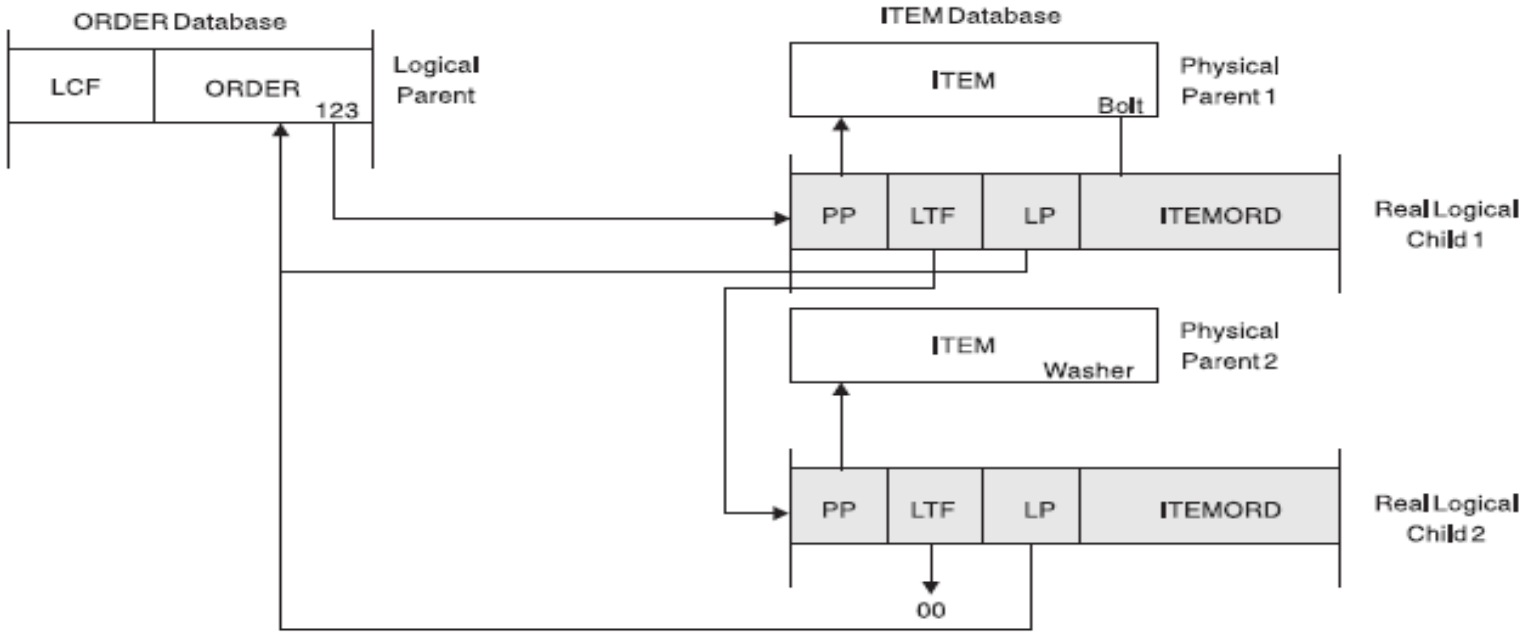
LCF Pointer



Logical Twin Pointer

Logical twins are multiple occurrences of logical child segments that point to the same occurrence of a logical parent segment. Two types of logical twin pointers can be used:

- logical twin forward (LTF)
- a combination of LTF and logical twin backward (LTB)



Virtual IMS CONNECTION

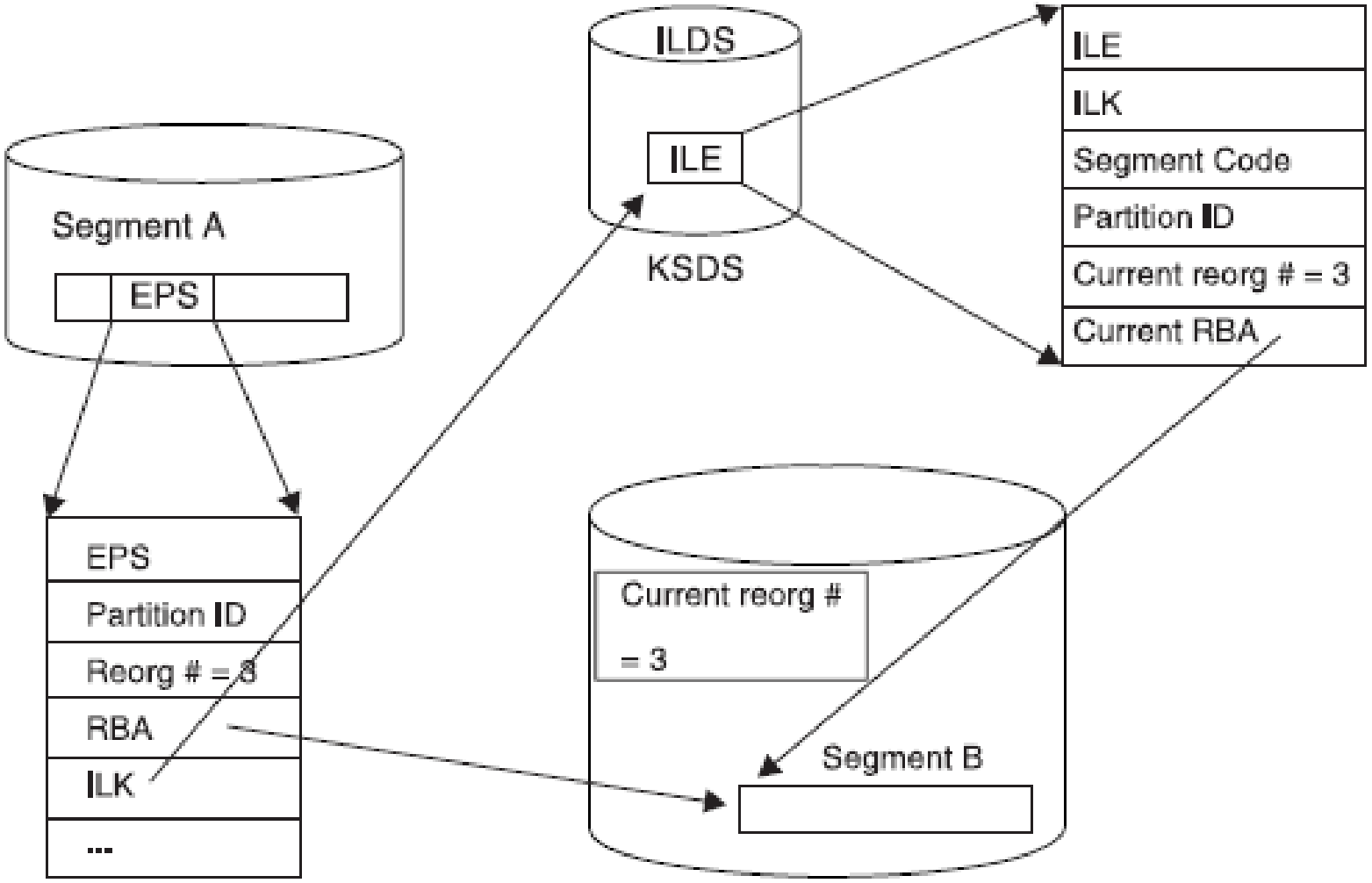
Indirect Pointers

HALDBs (PHDAM, PHIDAM, and PSINDEX databases) use direct and indirect pointers for pointing from one database record to another database record

The use of indirect pointers prevents the problem of misdirected pointers that would otherwise occur when a database is reorganized

- The repository for the indirect pointers is the indirect list data set
- The misdirected pointers after reorganization are self-healing using indirect pointers

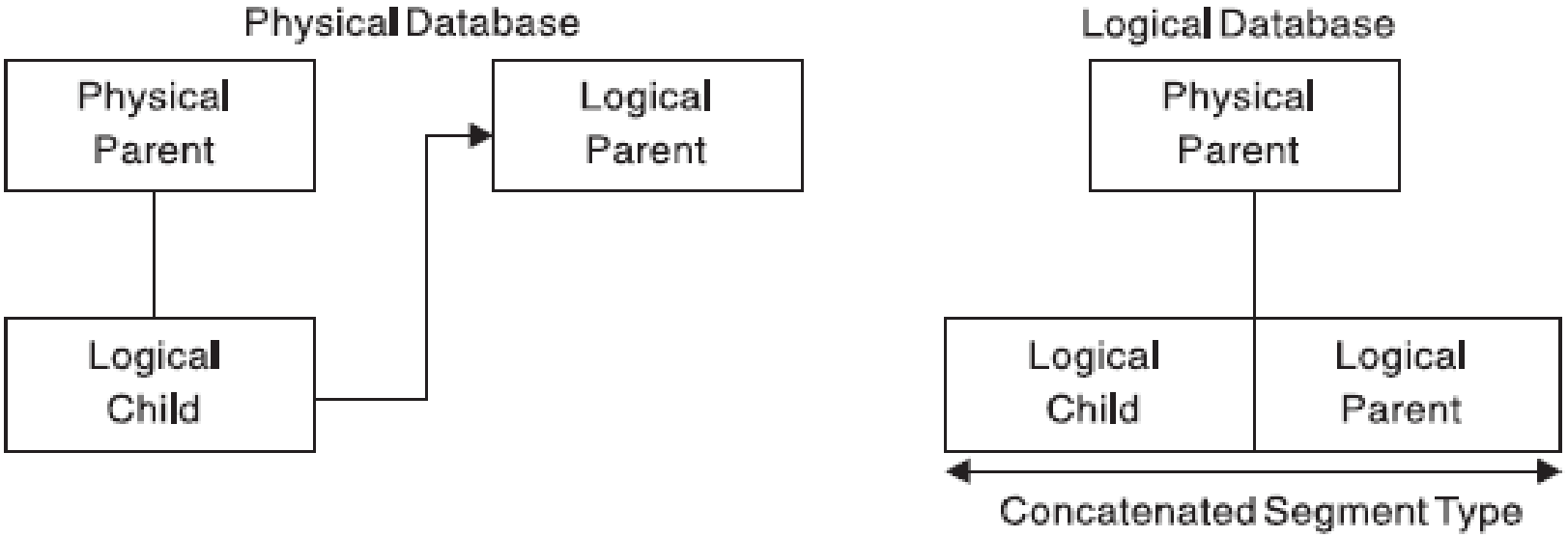
Self Healing Pointers



Physical Parent to Logical Parent Path

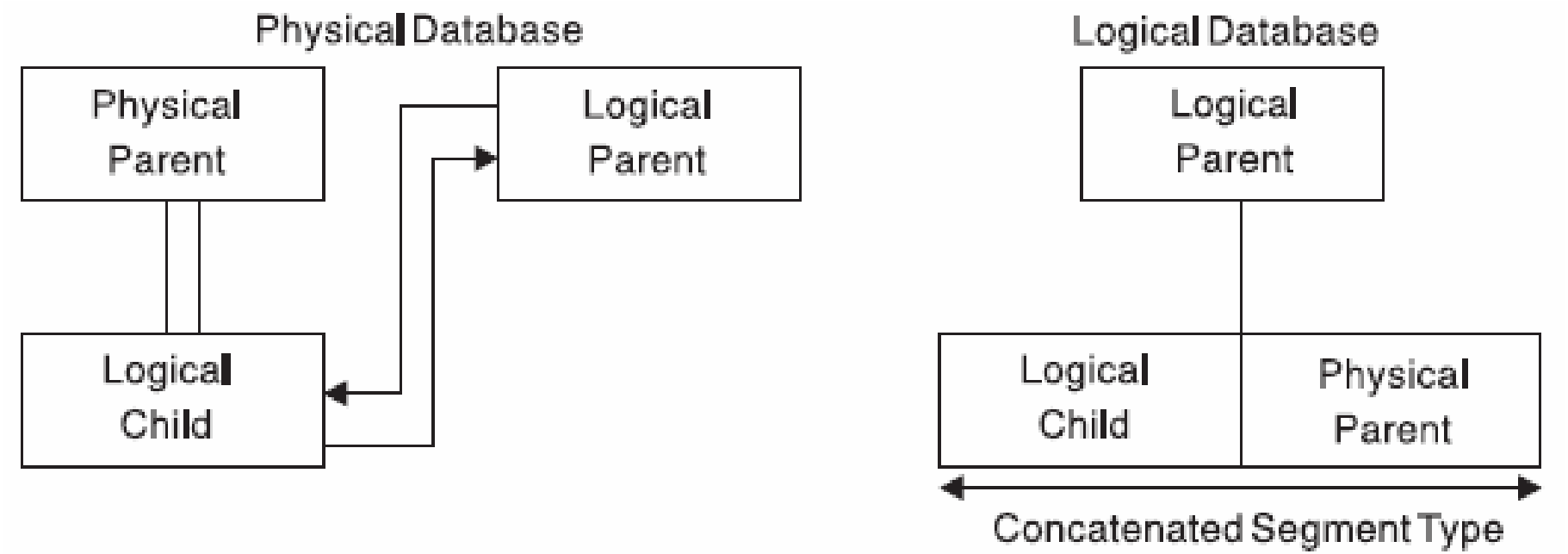
The relationship between physical parent and logical child in a physical database and the LP pointer in each logical child creates a physical parent to logical parent path

For a physical parent to logical parent path, the logical parent is the destination parent in the concatenated segment



Logical Parent to Physical Parent Path

For a logical parent to physical parent path, the physical parent is the destination parent in the concatenated segment



Paths in Logical Relationships

When use of a physical parent to logical parent path is defined, the physical parent is the parent of the concatenated segment type. When an application program retrieves an occurrence of the concatenated segment type from a physical parent, the logical child and its logical parent are concatenated and presented to the application program as one segment. When use of a logical parent to physical parent path is defined, the logical parent is the parent of the concatenated segment type. When an application program retrieves an occurrence of the concatenated segment type from a logical parent, an occurrence of the logical child and its physical parent are concatenated and presented to the application program as one segment.

In both cases, the physical parent or logical parent segment included in the concatenated segment is called the *destination parent*.

The Logical Child Segment

When defining a logical child in its physical database, the length specified for it must be large enough to contain the concatenated key of the logical parent. Any length greater than that can be used for intersection data

To identify which logical parent is pointed to by a logical child, the concatenated key of the logical parent must be present

Each logical child segment must be present in the application program's I/O area when the logical child is initially presented for loading into the database. However, if the logical parent is in an HD database, its concatenated key might not be written to storage when the logical child is loaded

If the logical parent is in a HISAM database, a logical child in storage must contain the concatenated key of its logical parent. For logical child segments, you can define a special operand on the PARENT= parameter of the SEGM statement. This operand determines whether a symbolic pointer to the logical parent is stored as part of the logical child segment on the storage device. If PHYSICAL is specified, the concatenated key of the logical parent is stored with each logical child segment. If VIRTUAL is specified, only the intersection data portion of each logical child segment is stored. When a concatenated segment is retrieved through a logical database, it contains the logical child segment, which consists of the concatenated key of the destination parent, followed by any intersection data. In turn, this is followed by data in the destination parent.

The concatenated key of the destination parent is returned with each concatenated segment to identify which destination parent was retrieved. IMS gets the concatenated key from the logical child in the concatenated segment or by constructing the concatenated key. If the destination parent is the logical parent and its concatenated key has not been stored with the logical child, IMS constructs the concatenated key and presents it to the application program. If the destination parent is the physical parent, IMS must always construct its concatenated key.

Logical Relationships in the Physical DBD

- One physical DBD for each of the dbs in a logical relationship
  - all statements are coded with the same format used when a logical relationship is not defined, except for the SEGM and LCHILD statements
  - the SEGM statement includes the new types of pointers
  - the LCHILD statement is added to define the logical relationship between the two segment types
  - the pointers for use with HD dbs must be explicit in the PTR= parameter

Bidirectional Logical Relationships

- For a bidirectional relationship with physical pairing, you include an LCHILD statement under both logical parents
  - you need to include the PAIRED operand on the POINTER= parameter of the SEGM statements for both logical children
- when defining a bidirectional relationship with virtual pairing, you need to code an LCHILD statement only for the real logical child
  - on the LCHILD statement, you code POINTER=SNGL or DBLE to get logical child pointers
  - the PAIR= operand indicates the virtual logical child that is paired with the real logical child
  - in the SEGM statement for the real logical child, the PARENT= parameter identifies both physical and logical parents
  - specify logical twin pointers (in addition to any other pointers) on the POINTER= parameter
  - define a SEGM statement for the virtual logical child even though it does not exist
    - on this SEGM statement, specify PAIRED on the POINTER= parameter
    - specify a SOURCE= parameter with SEGM name and DBD name of the real logical child
    - DATA must always be specified when defining SOURCE= on a virtual logical child SEGM statement.

Defining Logical Relationships in Physical dbs

Logical Child Rules:

- A logical child must have a physical and a logical parent
- A logical child can have only one physical and one logical parent
- A logical child is defined as a physical child in the physical database of its physical parent
- A logical child is always a dependent segment in a physical database, and can, therefore, be defined at any level except the first level of a database
- A logical child in its physical database cannot have a physical child defined at the next lower level in the database that is also a logical child
- A logical child can have a physical child. However, if a logical child is physically paired with another logical child, only one of the paired segments can have physical children.

Defining Logical Relationships in Physical dbs

Logical Parent Rules:

- A logical parent can be defined at any level in a physical database, including the root level
- A logical parent can have one or more logical children. Each logical child related to the same logical parent defines a logical relationship
- A segment in a physical database cannot be defined as both a logical parent and a logical child
- A logical parent can be defined in the same physical database as its logical child, or in a different physical database.

Physical Parent Rules:

- A physical parent of a logical child cannot also be a logical child.

Logical Relationships in the Logical DBD

To identify which segment types are used in a logical data structure, you must code a logical DBD

- When defining a segment in a logical database, you can specify whether the segment is returned to the program's I/O area by using the KEY or DATA operand on the SOURCE= parameter of the SEGM statement
  - DATA returns both the key and data portions of the segment to the I/O area
  - KEY returns only the key portion, and not the data portion of the segment to the I/O area
  - When the SOURCE= parameter is used on the SEGM statement of a concatenated segment, the KEY and DATA parameters control which of the two segments, or both, is put in the I/O area on retrieval calls
- In other words, you define the SOURCE= parameter twice for a concatenated segment type, once for the logical child portion and once for the destination parent portion
  - It is implemented with a unidirectional logical relationship using symbolic pointing

Defining Logical Databases

A logical DBD is needed only when an application program needs access to a concatenated segment or needs to cross a logical relationship.

Crossing a Logical Relationship:

- A logical relationship is considered crossed when it is used in a logical database to access a segment that is:
  - A physical parent of a destination parent in the destination parent's database
  - A physical dependent of a destination parent in the destination parent's physical database.

If a logical relationship is used in a logical database to access a destination parent only, the logical relationship is not considered crossed.

Definition of First and Additional Logical Relationships Crossed:

- More than one logical relationship can be crossed in a hierarchic path in a logical database
  - If either logical relationship or both is crossed, each is considered the first logical relationship crossed

Defining Logical Databases

-The root segment in a logical database must be the root segment in a physical database

-A logical database must use only those segments and physical and logical relationship paths defined in the physical DBD referenced by the logical DBD

-The path used to connect a parent and child in a logical database must be defined as a physical relationship path or a logical relationship path in the physical DBD referenced by the logical DBD

-Physical and logical relationship paths can be mixed in a hierarchic segment path in a logical database

-Additional physical relationship paths, logical relationship paths, or both paths can be included after a logical relationship is crossed in a hierarchic path in a logical database. These paths are included by going in upward directions, downward directions, or both directions, from the destination parent. When proceeding downward along a physical relationship path from the destination parent, direction cannot be changed except by crossing a logical relationship. When proceeding upward along a physical relationship path from the destination parent, direction can be changed

-Dependents in a logical database must be in the same relative order as they are under their parent in the physical database. If a segment in a logical database is a concatenated segment, the physical children of the logical child and children of the destination parent can be in any order. The relative order of the children or the logical child and the relative order of the children of the destination parent must remain unchanged

Defining Logical Databases

- The same concatenated segment type can be defined multiple times with different combinations of key and data sensitivity. Each must have a distinct name for that view of the concatenated segment. Only one of the views can have dependent segments
  - A PCB for the logical database can be sensitive to only one of the views of the concatenated segment type
    - **LC** Logical child segment type
    - **DP** Destination parent segment type
    - **K** KEY sensitivity specified for the segment type
    - **D** DATA sensitivity specified for the segment type

DBDs and PCBs

When a logical relationship is used, you must define the physical databases involved in the relationship to IMS

- *physical* DBD

Also, many times you must define the logical structure of IMS since this is the structure the application program perceives

- *logical* DBD
  - needed because the application program's PCB references a DBD
  - physical DBD does not reflect the logical data structure

Also, the application program needs a PSB

- one or more PCBs
  - the PCB used when processing with a logical relationship points to logical DBD (if defined)
  - this PCB indicates which segments in the logical db the application program can process
  - also indicates what type of processing the application program can perform on each segment

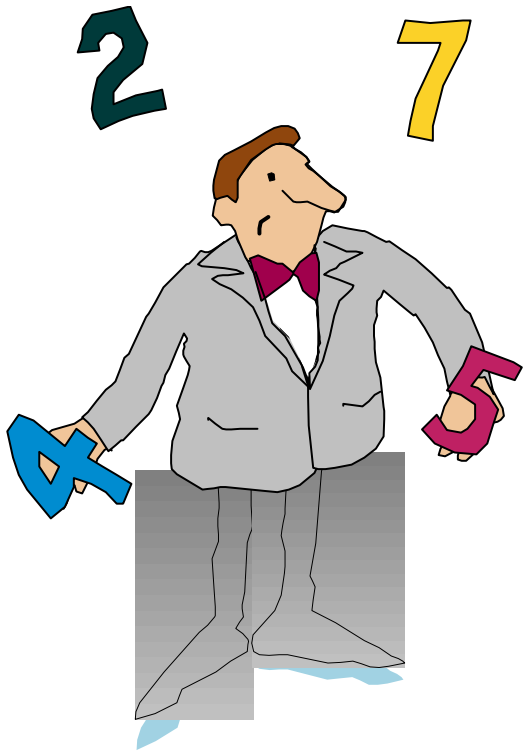
HALDB Databases

With HALDB dbs, bidirectional logical relationships must be implemented with physical pairing

- when loading a new partitioned db with logical relationships, the logical child segments cannot be loaded as part of the load step
- IMS adds logical children by normal update processing after db has been loaded

HALDBs use an indirect list data set (ILDS) to maintain logical relationship pointers when logically related db are reorganized

I’ve changed my mind…



Choosing Replace, Insert, and Delete Rules

- Insert, delete, and replace rules must be specified when a segment is involved in a logical relationship, because such segments can be updated from two paths: a physical path and a logical path

- You must first determine your application processing requirements and then the rules that support those requirements

Converting for support of partitioned dbs

Converting Logical Relationships to Support Partitioned Databases

- William N. Keene, *NEON Enterprise Software, Inc.*

This white paper provides guidance and examples for converting a set of logically related databases from bidirectional, virtual pairing using direct pointers to bidirectional, physical pairing using symbolic pointers

Virtual IMS CONNECTION

ca

and what about performance?

Performance Considerations

Advantages of direct pointers:

- usually shorter than symbolic pointers (4 bytes long)
- less DASD space generally required to store
- usually give faster access to LP segments
  - except possibly HDAM or PHDAM LP segments, which are roots

Performance Considerations

Advantages of symbolic pointers:

- stored as part of the logical child segment on DASD
  - can save resources needed to format LC segment in the user's I/O area
  - Can REORG logical parent dbs without the logical child db having to be reorganized
    - unidirectional and bidirectional physically paired relationships (when symbolic pointing is used)

Symbolic pointing **must** be used:

- for HISAM logical parent database
- to sequence LC segments (except virtual logical children) on any part of the symbolic key

KEY/DATA

When including a concatenated segment as part of a logical DBD, you can control how it appears in the user's I/O area

- specify either KEY or DATA on the SOURCE= keyword of the SEGM statement for the concatenated segment
  - a concatenated segment is a logical child followed by logical (or destination) parent
    - You CAN specify KEY or DATA for both parts
- choosing KEY or DATA carefully you could retrieve a concatenated segment more cheaply

do not automatically choose DATA sensitivity for both logical child and logical parent parts of a concatenated segment

Logical Twin Chains sequence

With virtual pairing, we can sequence the real logical child on physical twin chains and the virtual logical child on logical twin chains

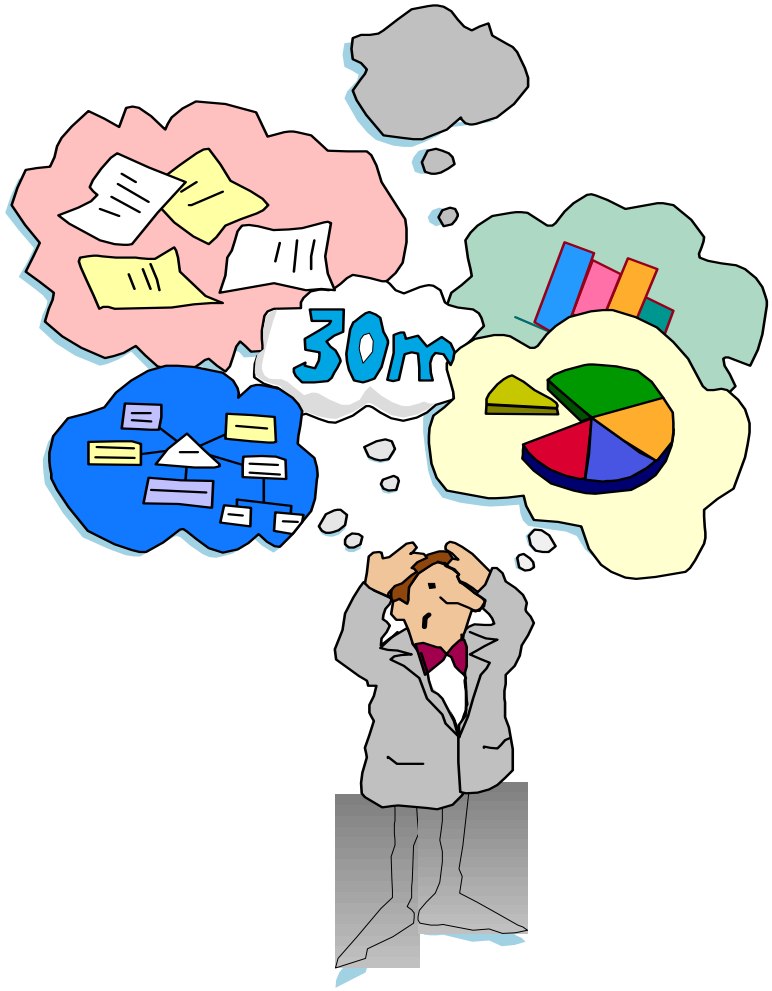
Try to avoid operations that need sequencing logical twins:

- when a logical twin chain is followed, DL/I usually has to access multiple db records
  - this increases the resources needed to process the call

Placement of Real LC in Virtually Paired Relationship

- if you need the LC sequenced in only one of the logically related dbs put the real LC in that db
- if you must sequence the LC in both logically related dbs, put the real LC in the database from which it is most often retrieved
- place the real LC so logical twin chains are as short as possible
    - decrease the number of db records that must be read to follow a logical twin chain

Things that make you go UHM…



Segment Prefix

1. IMS places pointers in the prefix in a specific sequence
2. IMS places a counter in the prefix for logical parents that do not have logical child pointers

   Multiple PCF and PCL pointers can exist in a segment type but more than one of the other types of pointers cannot

Segment Prefix

With segments with more than one type of pointer in a logical relationship, pointers in the segment's prefix are in the following sequence:

1. HF
2. HB
3. PP
4. LTF
5. LTB
6. LP

or

1. TF
2. TB
3. PP
4. LTF
5. LTB
6. LP
7. PCF
8. PCL

or

1. TF
2. TB
3. PP
4. PCF
5. PCL
6. EPS

Counters

IMS puts a 4-byte counter in all logical parents that do not have logical child pointers

- it is stored in the logical parent's prefix
- contains a count of the number of logical children pointing to this logical parent
- it is maintained by IMS
- it is used to handle delete operations properly

If count >0 the logical parent cannot be deleted

Intersection Data

With two logically related segments, there can be data that is unique to only that relationship - this type of data is called intersection data

- it has meaning only for the specific logical relationship

Two types:

- fixed intersection data (FID)
  - any data stored in the logical child
  - with direct pointing, FID is the only data in the logical child segment
  - in symbolic pointing, FID is stored in the data portion of the segment after the LPCK
- variable intersection data (VID)
  - used when several occurrences of intersection data for the same logical relationship
  - stored as a dependent of the logical child
  - there can be as many occurrences per logical child as needed

Recursive Structures: Same db Logical Relationships

Logical relationships can be established:

- between segments in two or more physical databases
- between segments in the same database
  - the logical data structure is called a *recursive structure*
  - most often defined in manufacturing for bill-of-materials type applications

Logical Parent Sequence Fields

Define unique sequence fields in all LP segments

- avoid potential problems in processing databases using logical relationships
- a logical parent is dependent on in its physical database
- Without unique sequence fields defined in all segments on the path to and including a logical parent, multiple logical parents in a database can have the same concatenated key
- problems can arise at and after initial db load

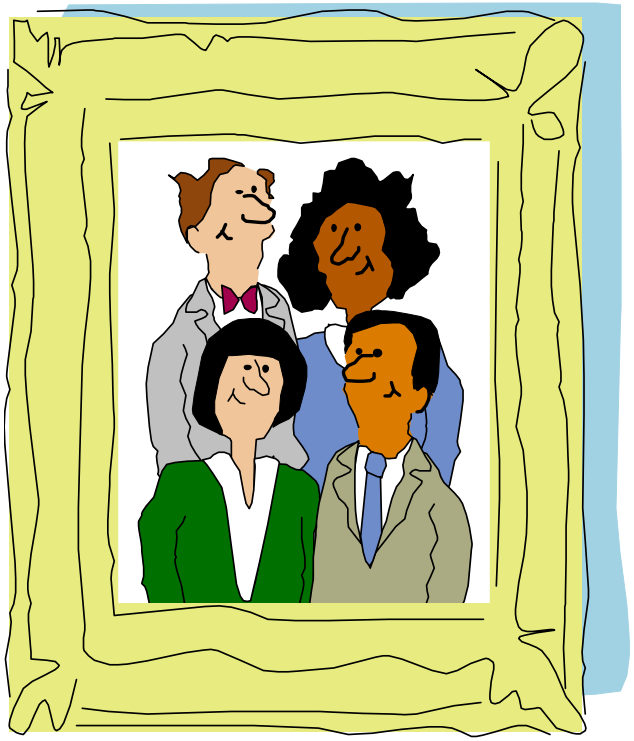
Logical Children Sequence Fields

Real Logical Children Sequence Fields

If the sequence field of a real logical child consists of any part of the logical parent's concatenated key, PHYSICAL must be specified on the PARENT= parameter in the SEGM statement for the logical child. This will cause the concatenated key of the logical parent to be stored with the logical child segment.

Virtual Logical Children Sequence Fields

As a general rule, a segment can have only one sequence field. However, in the case of virtual pairing, multiple FIELD statements can be used to define a logical sequence field for the virtual logical child. A sequence field must be specified for a virtual logical child if, when accessing it from its logical parent, you need real logical child segments retrieved in an order determined by data in a field of the virtual logical child as it could be seen in the application program I/O area. This sequence field can include any part of the segment as it appears when viewed from the logical parent (that is, the concatenated key of the real logical child's physical parent followed by any intersection data). Because it can be necessary to describe the sequence field of a logical child as accessed from its logical parent in non-contiguous pieces, multiple FIELD statements with the SEQ parameter present are permitted. Each statement must contain a unique fldname1 parameter.



In short - a review

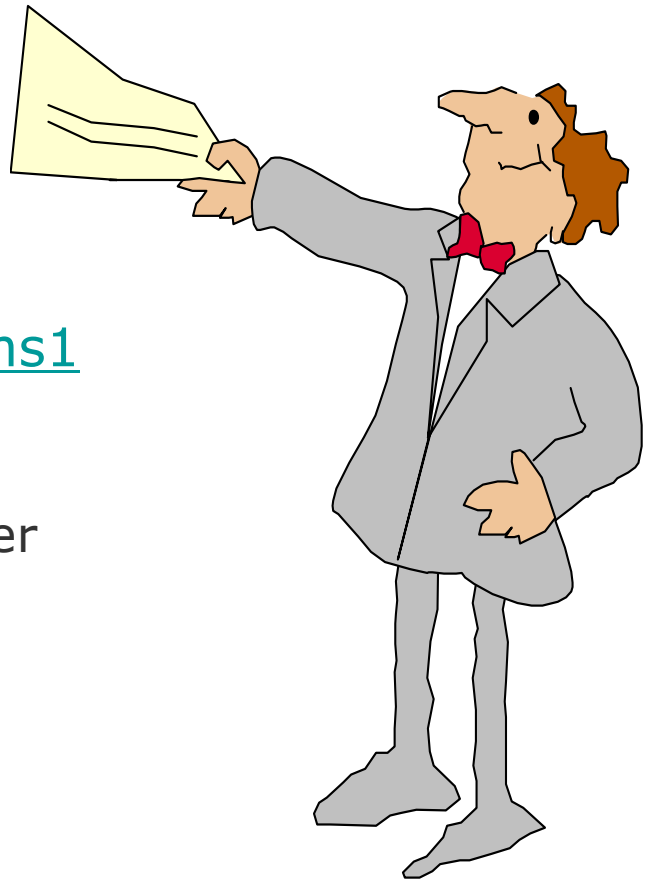
Virtual IMS CONNECTION

ca

Logical Relationships in IMS

- Logical relationships resolve conflicts in the way application programs need to view segments in the database
- With logical relationships, application programs can access:
    - Segment types in an order other than the one defined by the hierarchy
    - A data structure that contains segments from more than one physical database

Read the books!!!!!



- www.ibm.com/ims
- www.redbooks.ibm.com
- www.dbazine.com/ofinterest/oi-articles/ims1

- books:
  - IMS Administration Guide: Database Manager
  - IMS Performance and Tuning Guide

  - Redbook:
  - IMS Primer
    - SG24-5352-00

If you have no questions, I had thought of somewhere to spend the rest of the day...

Virtual IMS CONNECTION

The logical answer: IMS Logical Relationships

Aurora Emanuela Dell'Anno

CA

aurora.dellanno@ca.com

Virtual IMS CONNECTION