A Beginner's Guide to IMS Databases
E01

Charles Ryan

IBM Silicon Valley Lab

St. Louis, MO   **Sept. 30 - Oct. 3, 2002**

Contents

IBM Software Group

IBM

Contents

IBM Software Group

IBM

Contents

Secondary Indices

IBM Software Group

IBM

Database Basics

TOPIC

Database Basics

What is a Database

- A collection of interrelated data items organized in a form for easy retrieval
  - ► The collection of data is stored in a computer system
  - ► The retrieval is done by application programs
  - ► Each item of data only needs to be stored once
    - – Shared among the programs and users
- An IMS database is organized as a hierarchy
  - ► Levels of data
  - ► Data at lower levels depends on data at higher levels for its context
    - – You cannot understand the lower level without knowing the higher levels

IBM

The IMS Database

- A database is a group of related database records
- A database record is a single hierarchy of related segments
- A segment is a group of related fields
- A field is a single piece of data
  - ▶ It can be used as a key for ordering the segments
  - ▶ It can be used as a qualifier for searching
  - ▶ It may only have meaning to the applications
- IMS database always look like hierarchies

IBM Software Group

IBM

The Hierarchy

Level 1

ROOT
(1)

Level 2

Segment A
(2)

Segment B
(7)

Segment C
(9)

Level 3

Segment D
(3)

Segment E
(4)

Segment F
(8)

Segment G
(10)

Segment H
(12)

Level 4

Segment I
(5)

Segment J
(6)

Segment K
(11)

Segment Rules

- Root
    - ► One and only one root for each database record
    - ► No higher level segments
        - − Everything depends on the information in the root
- Other Segment Types
    - ► Up to 254 different segment types
        - − 255 including the root
    - ► Any number of occurrences of each segment type
    - ► Each segment, except the root, is related to one and only one segment at the next higher level

Segment Relationships

- Parent
  - ► All segments which have dependent segments at the next lower level are parents of those segments
  - ► A parent may have any number of dependent segments
- Child
  - ► A segment which depends on a segment at a higher level is a child of that segment
  - ► Every child segment has one and only one parent
- Twins
  - ► All occurrences of a segment type under the same parent are twins
  - ► There may be any number of twins and they are still called twins
- Siblings
  - ► Segments of different types with the same parent are siblings

Hierarchic Sequence

- Top to Bottom
- Left to Right
- Front to Back (for twins)
  - ► Each segment TYPE has a code which is its number in hierarchic sequence
    - – Segment codes numbers do not take twins into account
  - ► Sequential processing of a database record is in hierarchic sequence
    - – All segments of a database record are included so twins do have a place in hierarchic sequence
  - ► Segments may contain sequence fields which will determine the order in which they are stored and processed

IBM Software Group

Hierarchic Sequence ...

IBM Software Group
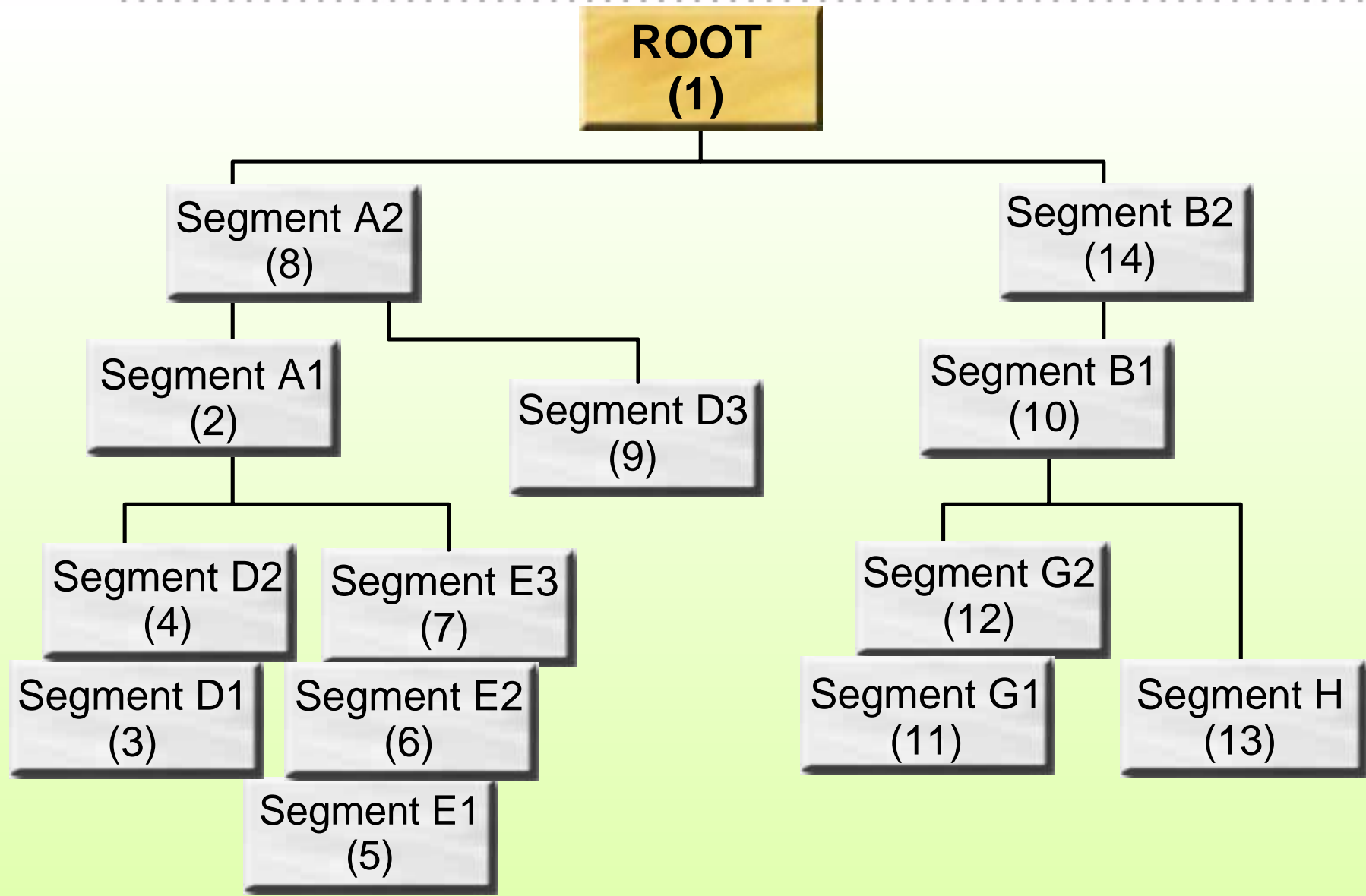
Access to Segments

- Retrieval
  - ► Get Unique (GU)
    - − Read a particular segment as determined by sequence or search fields
  - ► Get Next (GN)
    - − Read the next segment in hierarchic sequence
  - ► Get Next Within Parent (GNP)
    - − Read the next segment in hierarchic sequence under a particular parent segment
- Update
  - ► Insert (ISRT)
    - − Insert a new occurrence of a segment
  - ► Delete (DLET)
    - − Delete a segment
  - ► Replace (REPL)
    - − Update a segment with a new data, except for the sequence field

IBM Software Group

IBM

Segments in Storage

- Segments are stored with a prefix and a data portion
  - ► The prefix is used only by IMS
  - ► The data is what the application program sees

Prefix	Data
◄— 2+(4 x n) —►	◄— Variable —►

SC	DB	Pointer	Pointer	. . . .	Pointer
1	1	4	4		4
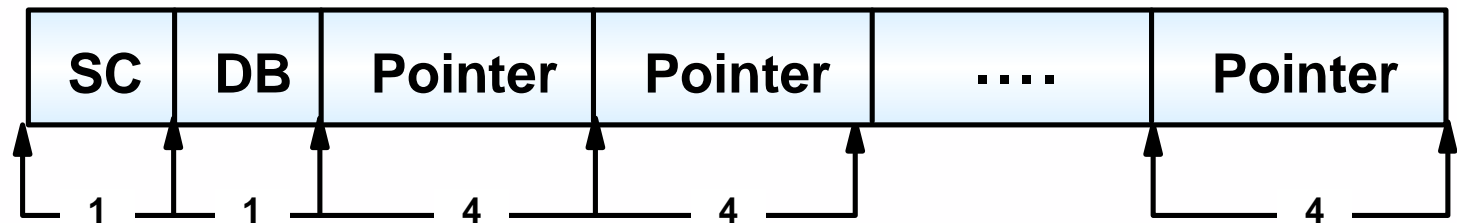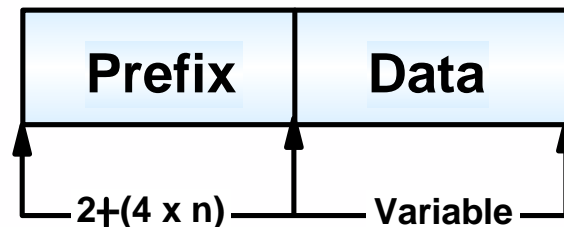
- The prefix contains:
  - ► SC = segment code, 1 byte
  - ► DB = delete byte, 1 byte
  - ► 0 to n pointers, 4 bytes each

Sequential Organization

TOPIC

Sequential Organization

IBM

Sequential Organization

- The data is physically stored in hierarchic sequence
  - ► Database records are stored in a root key sequence
    - − If no root key, they are stored as presented
  - ► Segments in a record are stored in hierarchic sequence
- Sequential Database Types
  - ► Hierarchical Sequential Access Method (HSAM)
  - ► Simple Hierarchical Sequential Access Method (SHSAM)
    - − Root-only HSAM
  - ► Hierarchical Indexed Sequential Access Method (HISAM)
  - ► Simple Hierarchic Indexed Sequential Access Method (SHISAM)
    - − Root-only HISAM using VSAM
  - ► Generalized Sequential Access Method (GSAM)
    - − No hierarchy, no database records, no segments

IBM Software Group

HSAM

- Tape or DASD
- BSAM or QSAM
  - ► QSAM if online or PROCOPT=GS
- Fixed-Length, Unblocked format
  - ► RECFM=F, logical record length=physical block size
- Cannot Delete or Replace
  - ► Update by rewriting the database
- Insert allowed when loading the database
- Restrictions
  - ► No pointers in prefix - SC and DB only
    - − Delete byte is not used
  - ► No multiple data set groups (MSDG)
  - ► No logical relationships or secondary indices
  - ► No variable length segments
  - ► No edit/compression or data capture
  - ► No logging, recovery, or reorganization

IBM Software Group

IBM

HSAM Storage

ROOT 1

Segment A2

Segment A1

Segment D3

Segment B2

Segment B1

Segment D2

Segment E3

Segment D1

Segment E2

Segment E1

Segment G2

Segment G1

Segment H1

Root 1	A1	D1	D2	E1	00

Block 1

E2	E3	A2	D3	B1	G1	0

Block 2

G2	H1	B2	Root 2	...	00

Block 3
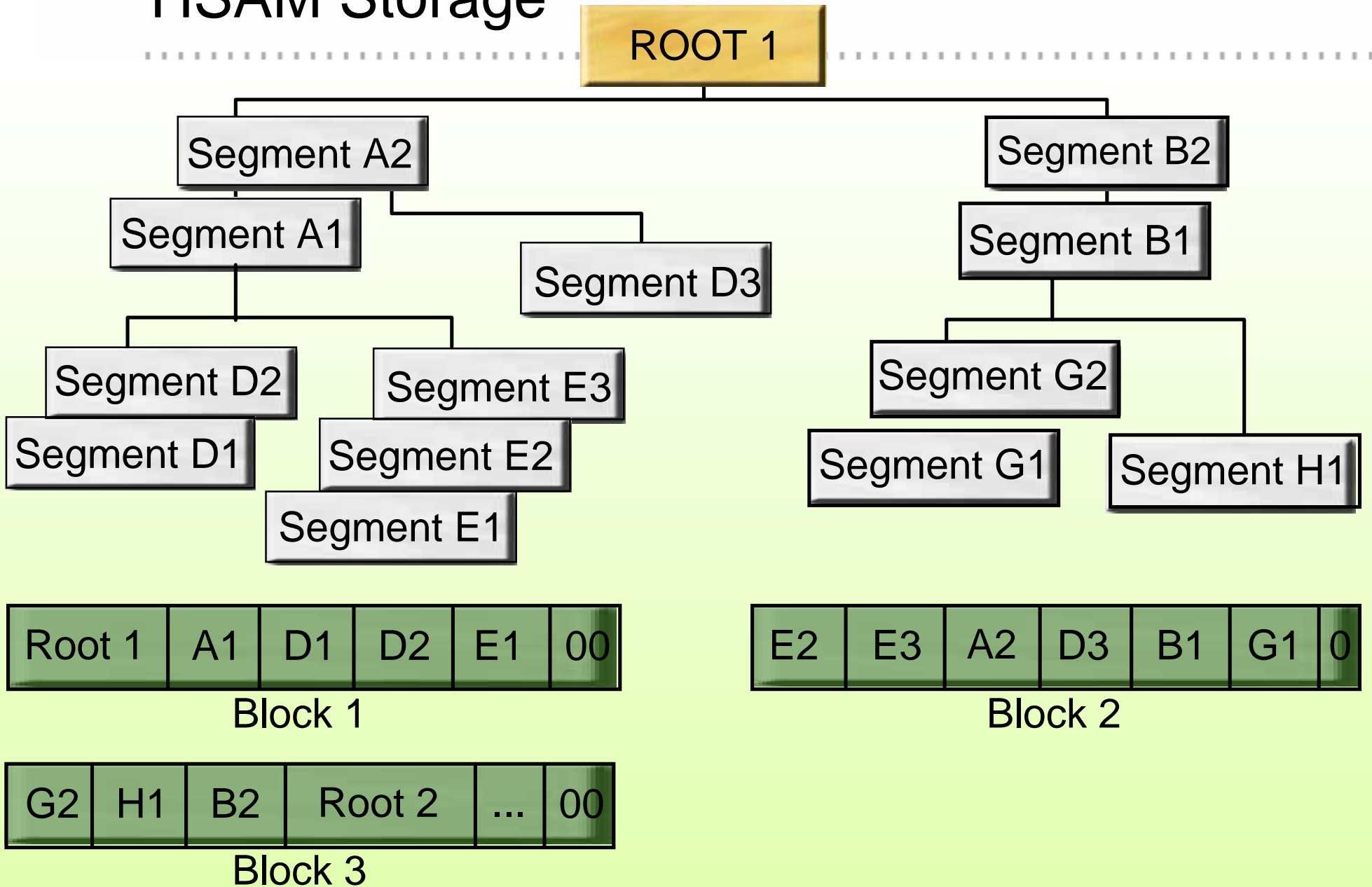
HSAM Processing

- Retrieval

```
GU CALL IS ISSUED
        |
IS THERE A ROOT ──YES──> IS KEY WANTED GREATER ──YES──┐
SEQUENCE FIELD            THAN CURRENT KEY             |
        |                        |                     |
        |                       NO                     |
        |              GS        |                     |
        |<───────────── PROCOPT                        |
        |                        |                     |
        |                        G                     |
        v                        v                     v
Scan from start of DB    Back 2, Forward 1      Scan From Here
```

- Update

```
EXISTING HSAM ──> UPDATING ──> NEW (UPDATED HSAM
DB                PROGRAM       DB
```
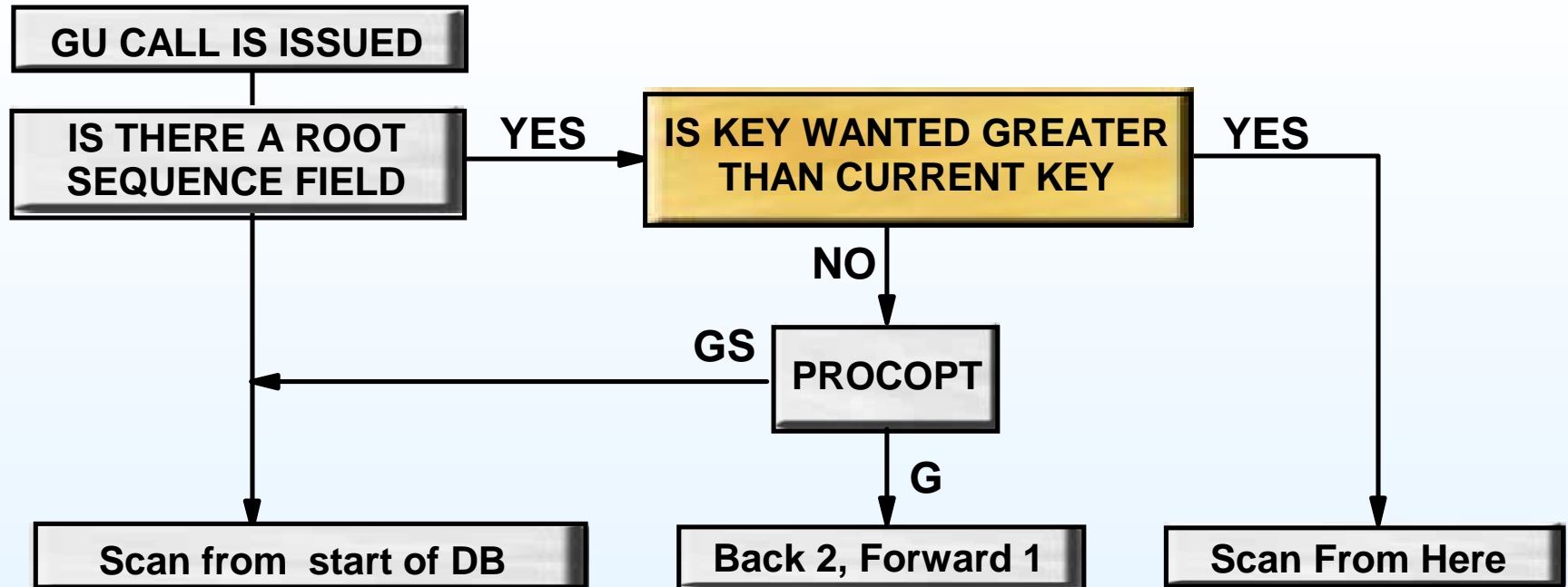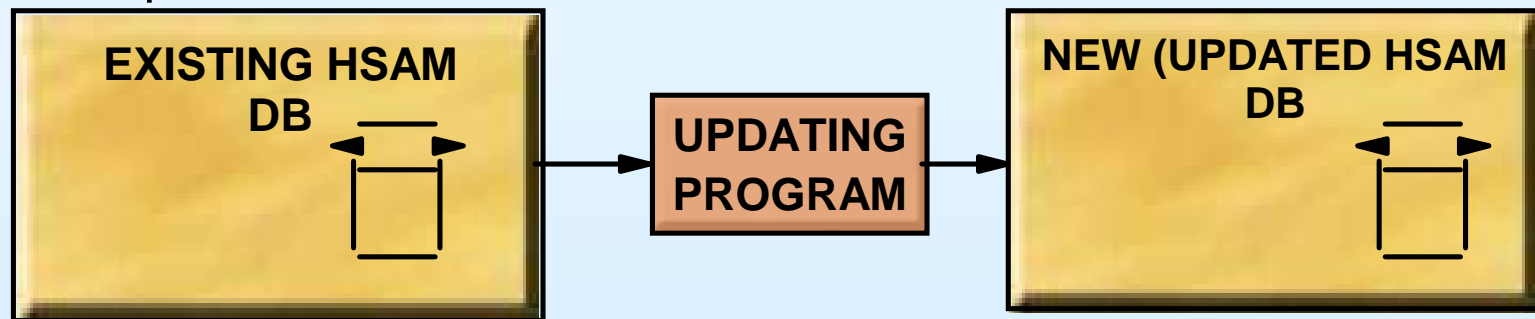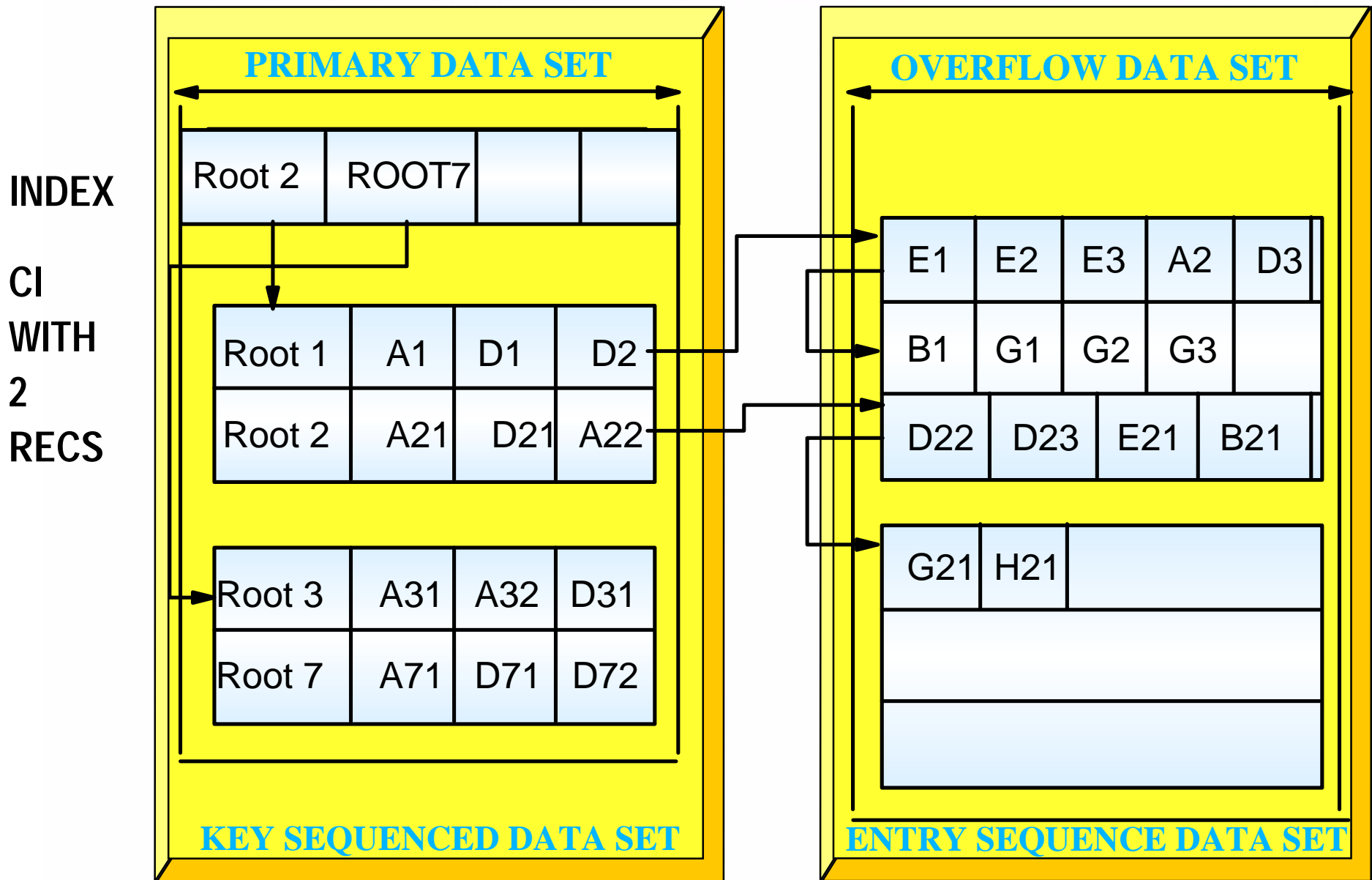
SHSAM

- HSAM with only one segment type (root-only)
  - ► No prefix is used
    - – No SC because only one segment type
- Same restrictions and processing as HSAM
- Fully equivalent to plain QSAM or BSAM file
  - ► Communication with non-IMS systems
  - ► Passing large amounts of data

IBM Software Group

HISAM

- DASD only
- VSAM
  - ► KSDS for the primary data set
  - ► ESDS for the overflow data set
- Each root must have a unique key
- A database record is stored as 1 record in the primary data set and 0 to N records in the overflow data set
- All calls are allowed
- Prefix consists of Segment Code (SC) and Delete Byte (DB)
- HSAM restriction do not apply
- HISAM works better when
  - ► Applications randomly access the records and then read the segments sequentially
  - ► Most of the database records are the same size
  - ► Relatively few dependents per root
  - ► Very low insert/delete activity

IBM Software Group

IBM

HISAM Storage

INDEX

CI

WITH

2

RECS

PRIMARY DATA SET

Root 2	ROOT7		

Root 1	A1	D1	D2
Root 2	A21	D21	A22

Root 3	A31	A32	D31
Root 7	A71	D71	D72

KEY SEQUENCED DATA SET

OVERFLOW DATA SET

E1	E2	E3	A2	D3
B1	G1	G2	G3	
D22	D23	E21	B21	

G21	H21		

ENTRY SEQUENCE DATA SET

IBM Software Group

IBM

22

HISAM VSAM Logical Record

Relative Byte Address	Segment	Segment	. . . . .	Segment	Segment Code of 0	Unused Space

◄—— 4 ——► ◄———————— Varies ————————► ◄— 1 —► ◄ Varies ►
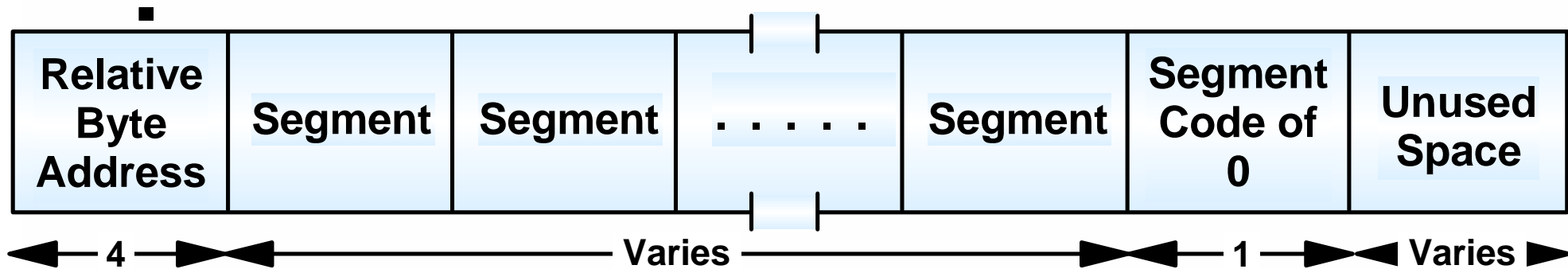
- RBA pointer to the next logical record for this database record
- Last logical record for DB record has zeros
- Segments are stored in hierarchic sequence
- SC of zero indicates end of segments in this logical record
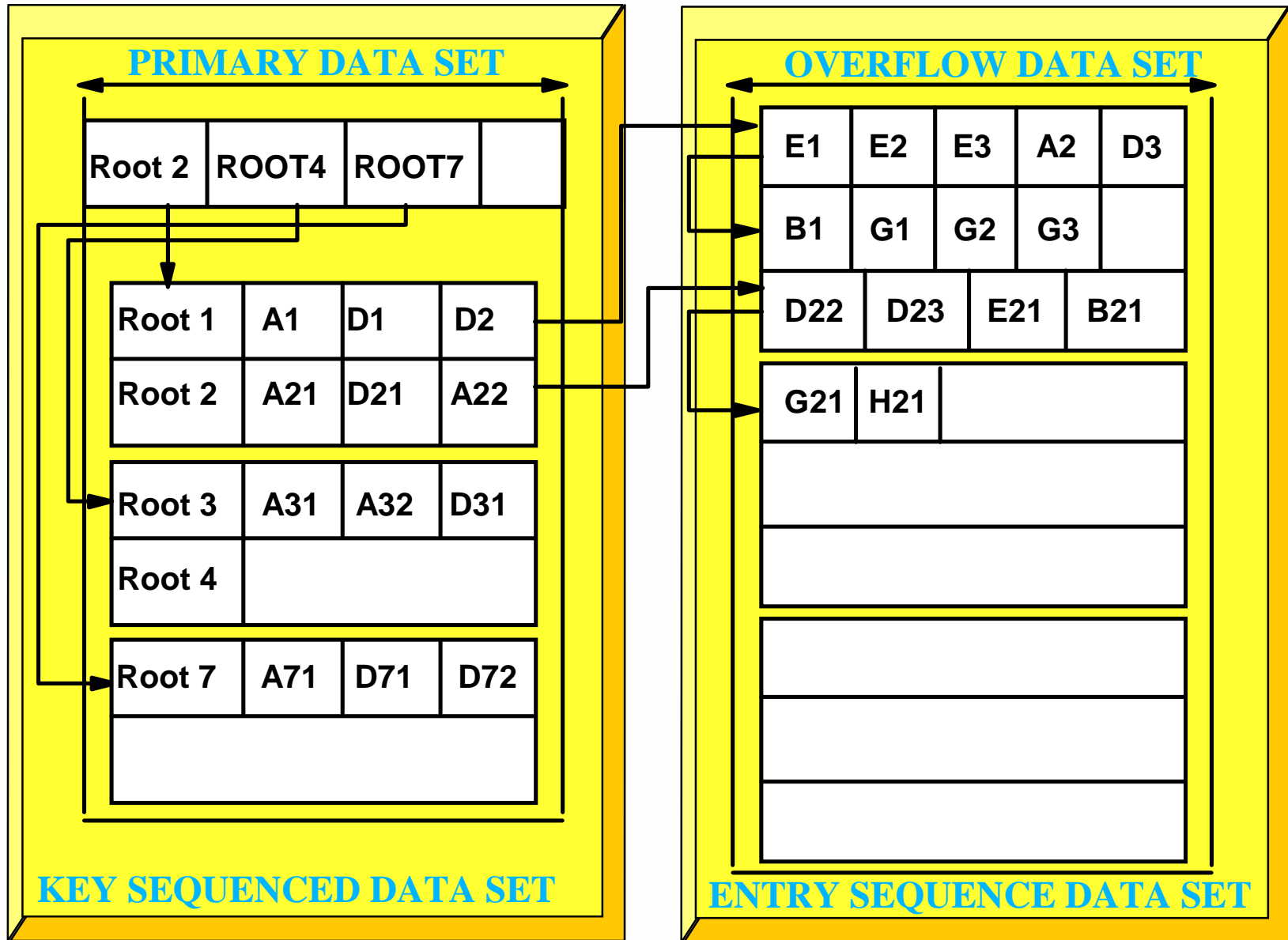- Unused space can have any data in it

IBM Software Group

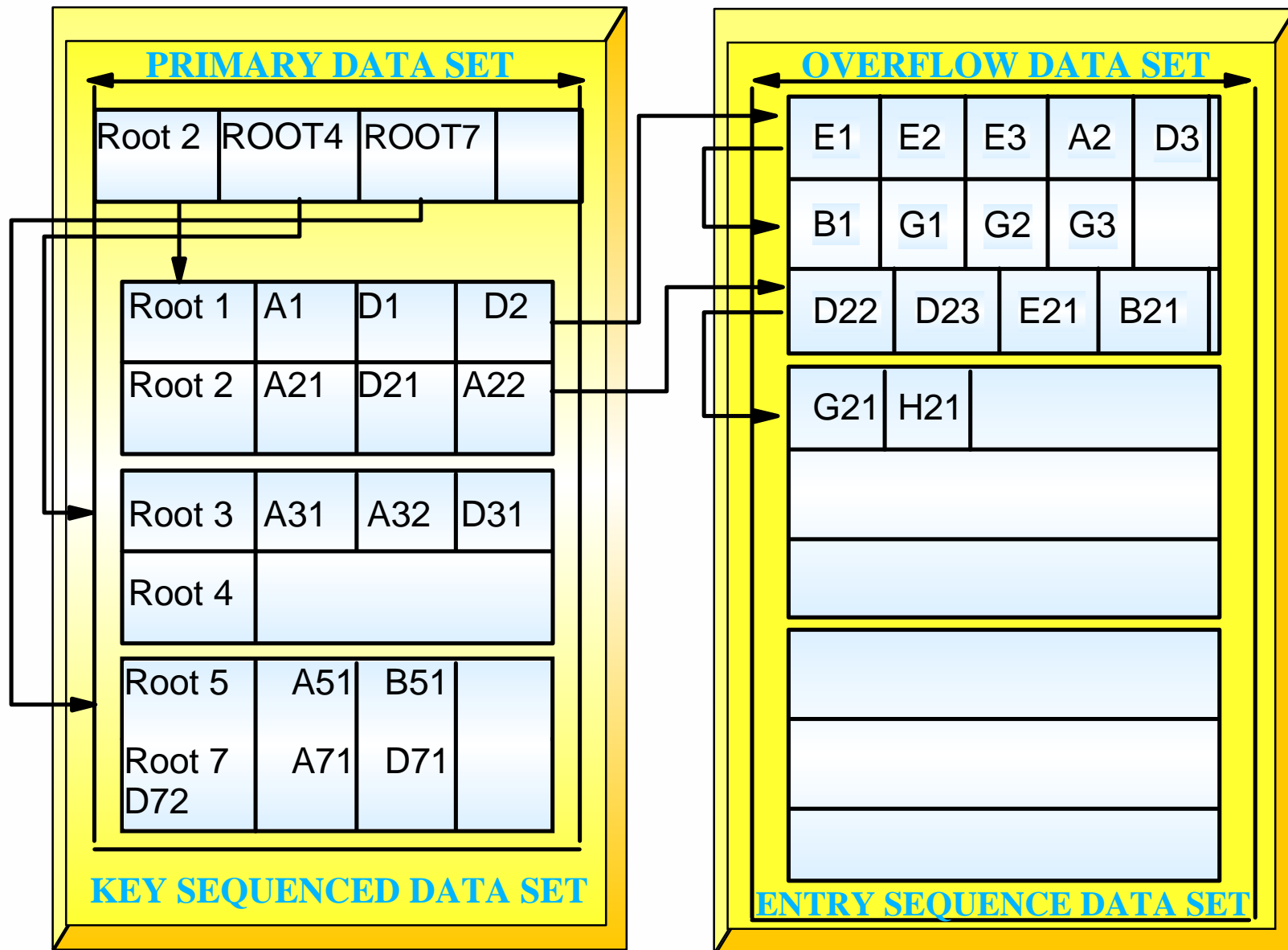IBM

23

HISAM Inserts

- HISAM Roots are always inserted into the Primary Data Set (KSDS)
  - ► If there is an free record in the VSAM Control Interval (CI)
    - − Inserted in root key sequence
    - − Higher keys are 'pushed down' to make space
  - ► If there is no free record in the CI
    - − CI is split - some of the records moved to a new CI
      - Split at midpoint or insert point by INSERT = in DFSVSAMP
    - − After split, same as free record case
- Dependents are inserted in their place in hierarchic sequence
  - ► If there is room in the logical record
    - − Following are 'pushed down' to make space
  - ► If there is not enough room
    - − All following segments are moved to a new overflow record
    - − Overflow records chain is updated
    - − Segment is inserted

IBM Software Group

IBM

Insert Root 4

PRIMARY DATA SET

| Root 2 | ROOT4 | ROOT7 | |

| Root 1 | A1 | D1 | D2 |
| Root 2 | A21 | D21 | A22 |

| Root 3 | A31 | A32 | D31 |
| Root 4 | | | |

| Root 7 | A71 | D71 | D72 |
| | | | |

KEY SEQUENCED DATA SET

OVERFLOW DATA SET

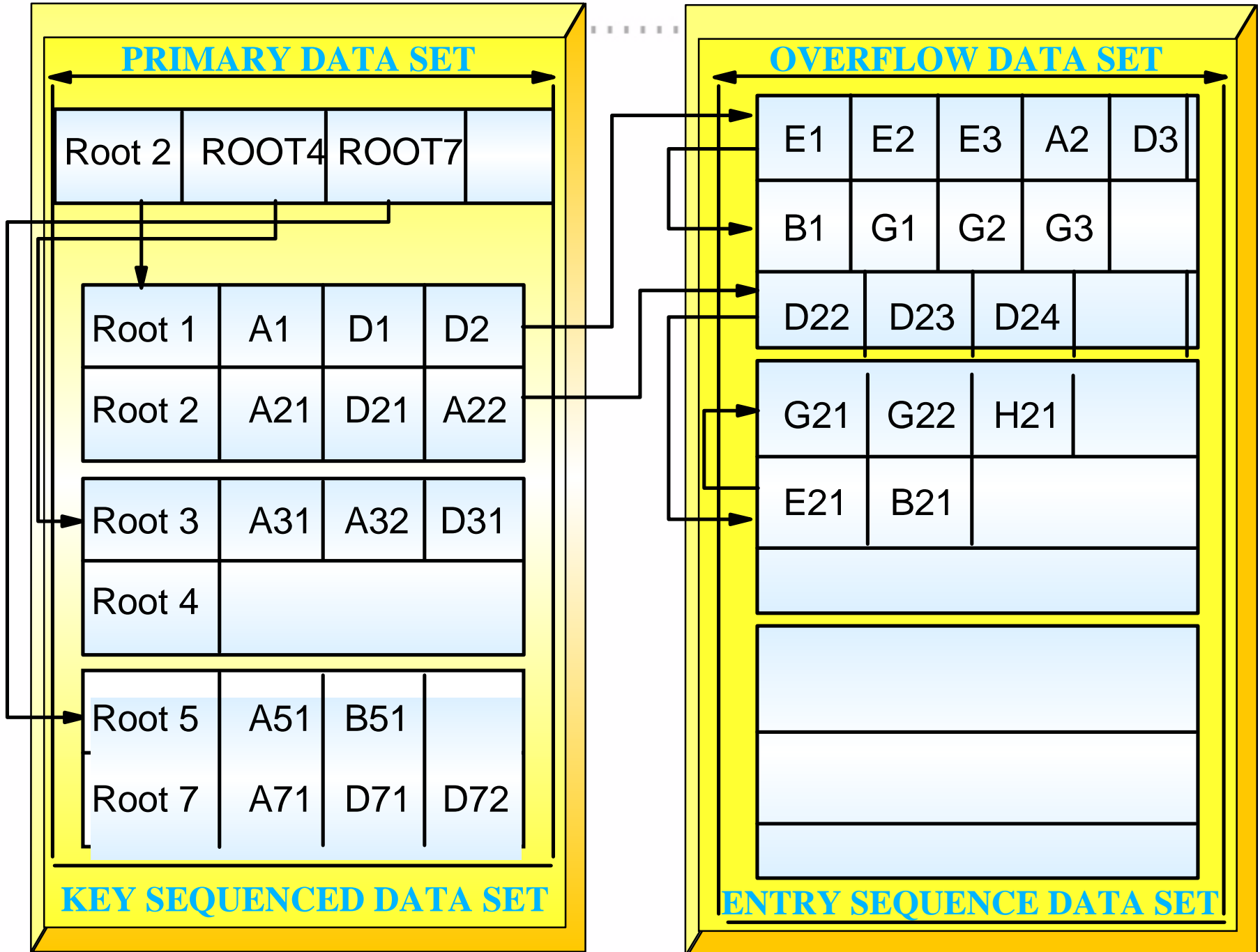| E1 | E2 | E3 | A2 | D3 |
| B1 | G1 | G2 | G3 | |
| D22 | D23 | E21 | B21 | |

| G21 | H21 | | |

ENTRY SEQUENCE DATA SET

IBM Software Group

IBM

Insert Root 5

Insert Dependents G22 and D24

HISAM Delete and Replace

- Delete
  - ► Marked as deleted in the Delete Byte in prefix
    - – Dependents are not flagged but can't be accessed (parent segment marked)
  - ► Continue to take up space
    - – Unload/Reload to reclaim space
  - ► If the root is deleted and no logical relationship exists
    - – The record is deleted from the primary data set
    - – Overflow records continue to exist in the overflow
- Replace
  - ► Fixed length or same length
    - – Overwrite previous data
  - ► Variable length
    - – Other segments in the record move to make space
    - – Displaced segments will go to a new overflow record

IBM Software Group

SHISAM

- HISAM with only one segment type (root-only)
  - ► No prefix is used
    - – No SC because only one segment type
    - – No DB because logical record is deleted (VSAM erase)
- Restrictions
  - ► No logical relationships or secondary indices
  - ► No multiple data set groups
  - ► No variable length segments
  - ► No edit/compression
- Fully equivalent to a VSAM KSDS
  - ► No ESDS because no dependent overflow
  - ► Can be accessed by native VSAM programs

IBM Software Group

IBM

GSAM

- Compatible with MVS data sets
  - ► No hierarchy
  - ► No database records
  - ► No segments and no keys
- GSAM VSAM
  - ► ESDS on DASD
  - ► Fixed or variable length records
- GSAM QSAM/BSAM
  - ► Physical sequential (DSORG=PS) on DASD or Tape
  - ► Fixed, variable, or undefined length records
- GSAM Processing
  - ► No Delete or Replace
  - ► Insert only at the end of the data set
  - ► Gets by sequential scan

IBM Software Group

IBM

GSAM ...

- Restrictions
  - ► No multiple data set groups
  - ► No logical relationships or secondary indices
  - ► No edit/compression or data capture
  - ► No field level sensitivity
  - ► No logging or reorganization
- Checkpoint and Restart
  - ► IMS symbolic checkpoint supports GSAM
  - ► Can restart from checkpoint instead of reprocessing
  - ► Restart repositions in the GSAM data set

IBM Software Group

Direct Organization

TOPIC

Direct Organization
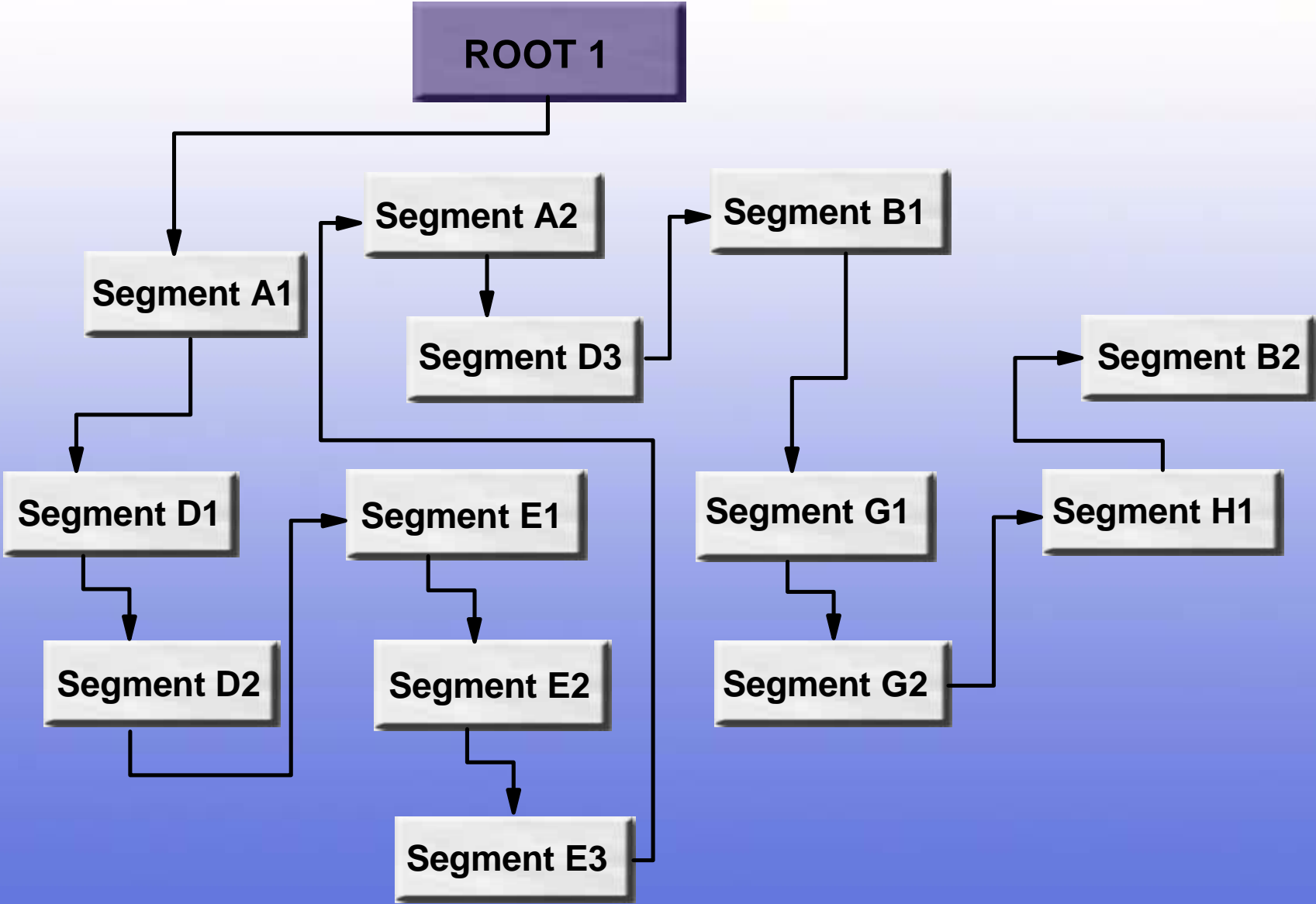
Direct Organization

- Physical storage is independent of hierarchic sequence
  - ► Pointers are used to maintain segment relationships
    - – Pointers are in the segment prefix
    - – Segments can be stored 'anywhere'
    - – Segments are not physically moved
  - ► Space from deleted segments can be reused
- Direct Database Types
  - ► Hierarchic Direct Access Method (HDAM)
    - – Uses a randomizing module for direct access to root
  - ► Hierarchic Indexed Direct Access Method (HIDAM)
    - – Searches an index to find the root
- High Availability Large Database (HALDB)
  - ► HDAM and HIDAM databases partitioned using the HALDB Partition Definition Utility (DSPXPDDU) become
    - – Partitioned Hierarchic Direct Access Method (PHDAM)
    - – Partitioned Hierarchic Indexed Direct Access Method (PHIDAM)
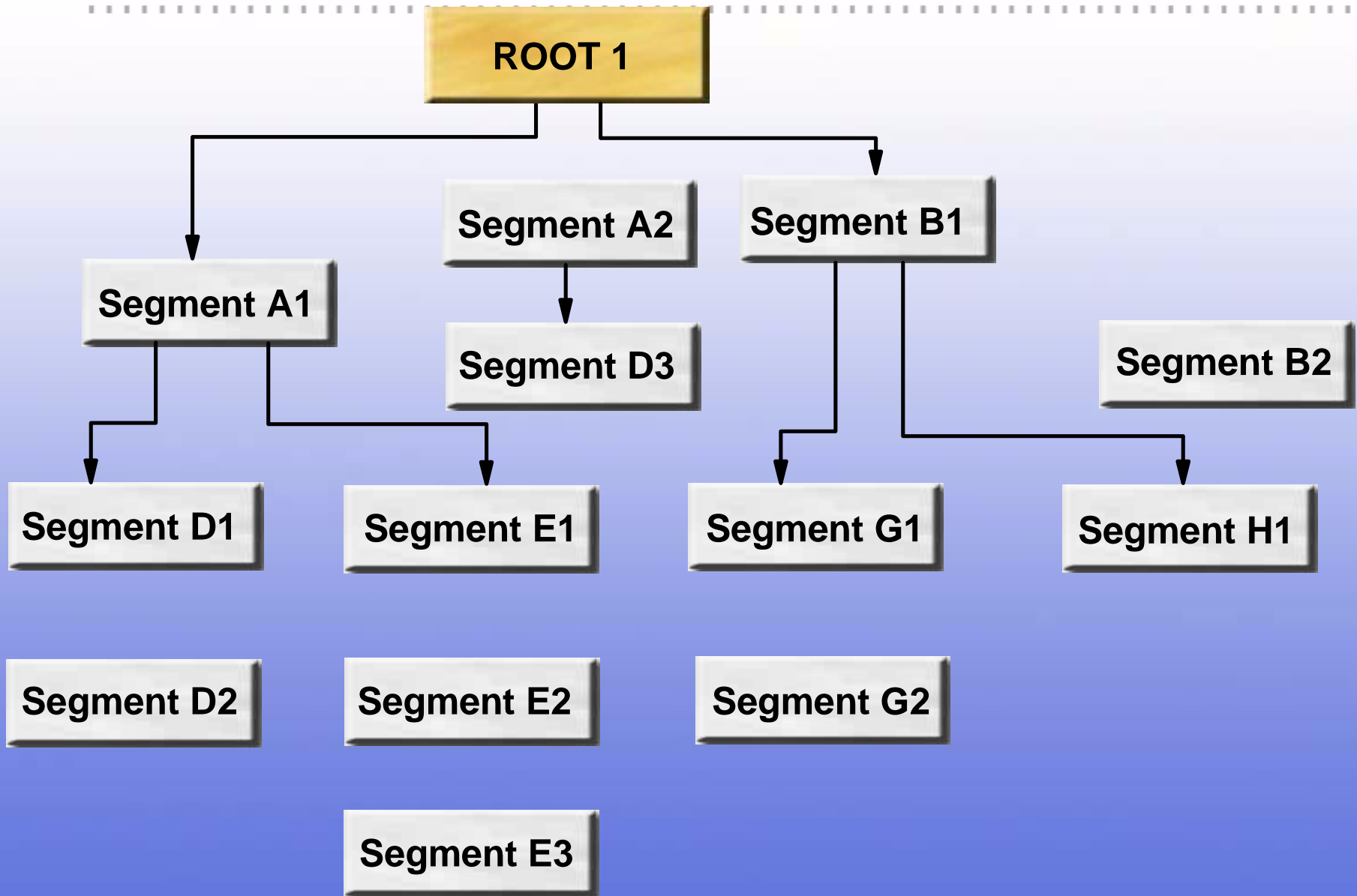   - – See manuals for further information.

Pointer Types

- Hierarchic
  - ► May be present in all segment types
  - ► Forward (HF)
    - − Points to next segment in hierarchic sequence
  - ► Backward (HB)
    - − Points to previous segment in hierarchic sequence
    - − Must also have HF pointers
- Physical Child
  - ► Found only in the prefix of a parent segment
  - ► First (PCF)
    - − Points to the first occurrence of a child segment type
    - − Must also have PCF pointer
- Twin
  - ► Forward (PTF)
    - − Points to the next twin in key or entry sequence
  - ► Backward (PTB)
    - − Points to the previous twin
    - − Must also have PTF pointer

IBM Software Group

IBM

Hierarchic Forward Pointers

IBM Software Group

Physical Child First Pointers

IBM Software Group

Physical Twin Pointers

ROOT 1

Segment A1 → Segment A2

Segment B1 → Segment B2

Segment D3

Segment D1 → Segment D2

Segment E1 → Segment E2 → Segment E3

Segment G1 → Segment G2

Segment H1
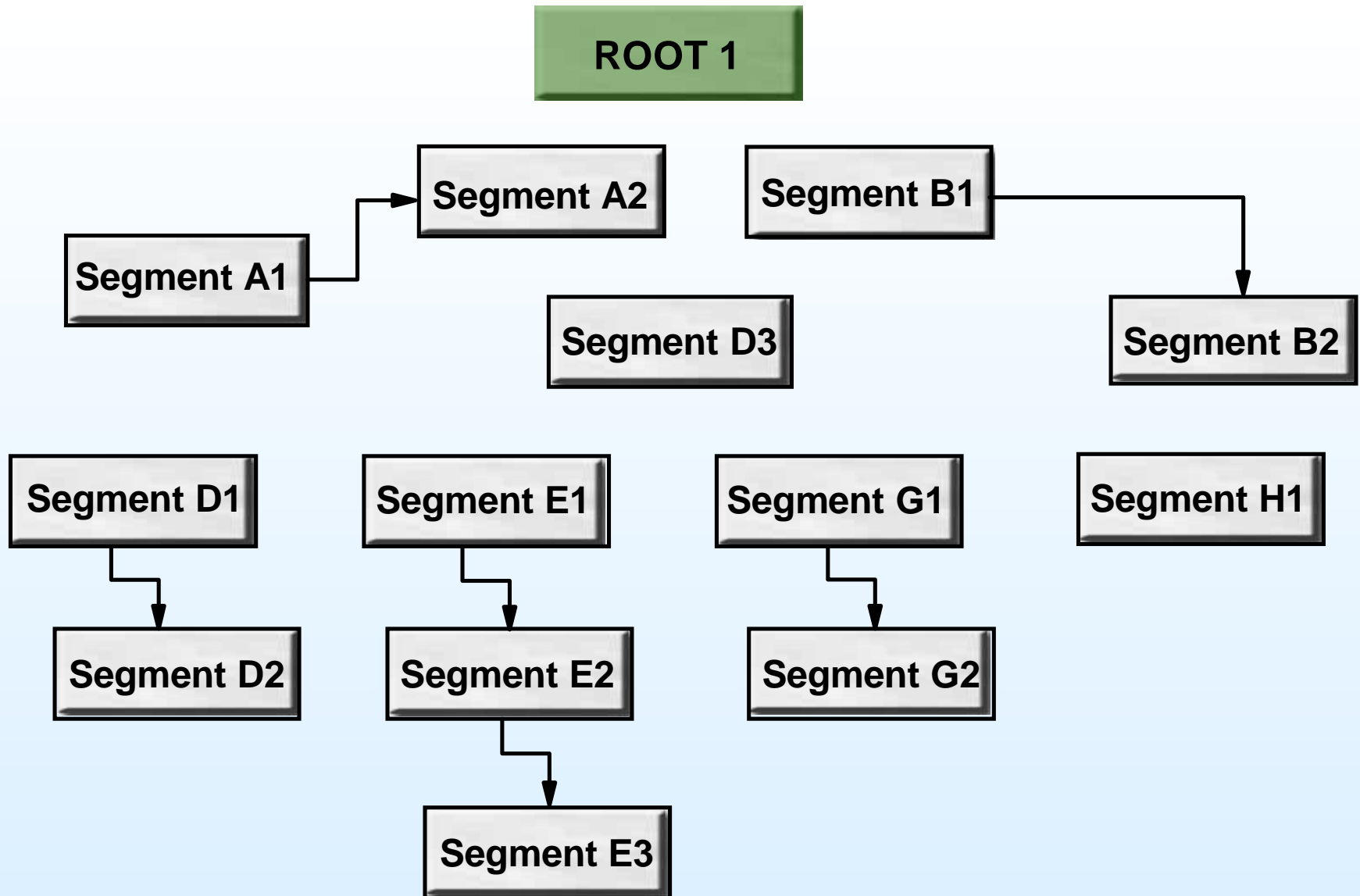
Physical Child Last Pointers

IBM Software Group
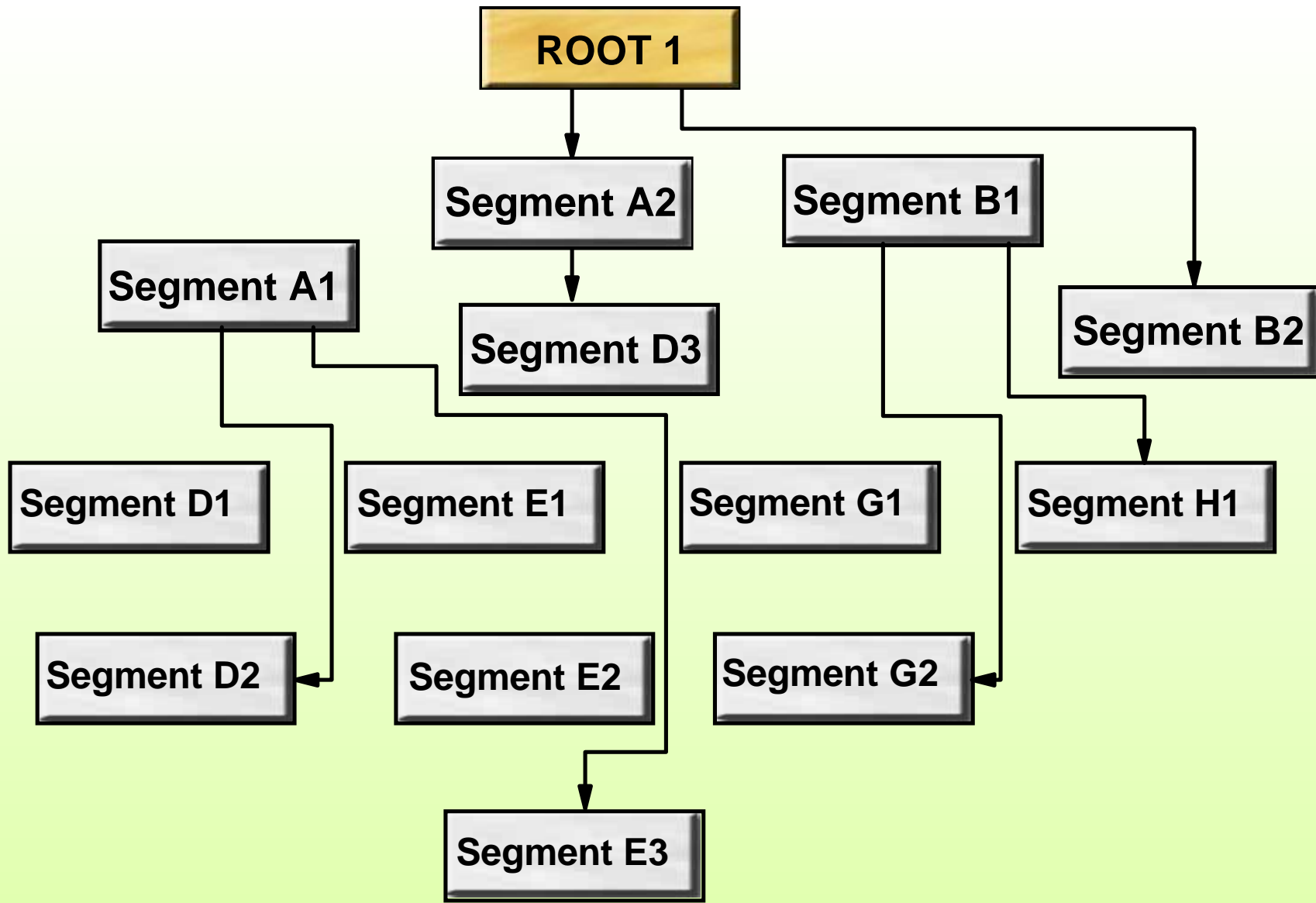
Pointer Uses

- **Hierarchic Forward**
  - ▶ Primary processing is in hierarchic sequence
- **Hierarchic Backward**
  - ▶ Delete activity via a logical relationship or secondary index
- **Physical Child First**
  - ▶ Random processing
  - ▶ Sequence field or insert rule FIRST or HERE
- **Physical Child Last**
  - ▶ No sequence field and insert rule LAST
  - ▶ Use of *L command code
- **Physical Twin Forward**
  - ▶ Random processing
  - ▶ Needed for HDAM roots
  - ▶ Poor choice for HIDAM roots
- **Physical Twin Backward**
  - ▶ Improves delete performance
  - ▶ Processing HIDAM roots in key sequence

IBM Software Group

IBM

Pointers in the Prefix

| HF | HB | or | PTF | PTB | PCF | PCL |

- Cannot have Hierarchic and Physical in the same prefix
  - ► PTR=H will cause PCF specification to be ignored
- If a parent has PTR=H, children cannot use backward pointers
- If a parent has PTR=HB, children must use backward pointers
- Child pointers will behave like the parent specification
  - ► Parent hierarchic, last twin pointer goes to sibling, not 0
  - ► Parent twin, last hierarchic pointer in twins is 0

IBM Software Group

IBM

HD Storage

VSAM ESDS OR OSAM DATA SET

- All HD data is in a single ESDS or OSAM data set
- The logical records are unblocked
  - Logical record length = block size for OSAM
  - Logical record length = block size -7 for VSAM
- All segments are stored as an even number of bytes

Reserved CI If VSAM - Not Present If OSAM				
FSAP	ANCHOR POINT AREA	BITMAP		
FSAP	ANCHOR POINT AREA	SEGMENTS	FSE	FREE SPACE
FSAP	ANCHOR POINT AREA	FSE	FREE SPACE SEGMENTS FSE	.
FSAP	ANCHOR POINT AREA	SEGMENTS FSE FREE SPACE		SEGMT

IBM Software Group
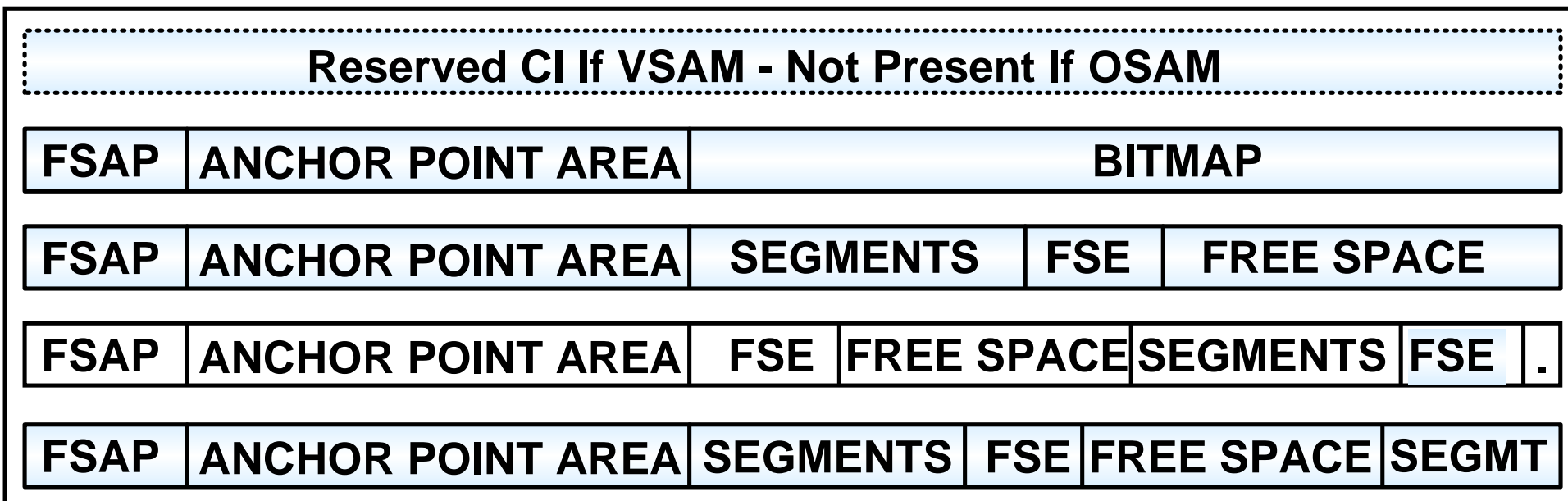
Special HD Fields

- Bitmap
  - ► One bit per block or CI
    - – First bit corresponds to the bitmap itself
  - ► 1 = enough space to store the LONGEST segment in the database
  - ► 0 = not enough space for the LONGEST segment
  - ► If bitmap has N bits, block or CI N + 1 is a new bitmap
- Free Space Anchor Point (FSAP)
  - ► Two 2-byte fields
    - – First the offset from in bytes to first FSE
    - – Second is a flag indicating if this block is a bitmap
      0 = this is not a bitmap
- Anchor Point Area
  - ► Contains one or more 4-byte Root Anchor Points (RAP)
    - – 1 RAP in HIDAM if the root has PTF or HF pointer
    - – RMNAME parameter specifies number of RAPs in HDAM
  - ► Each RAP contains the address of a root segment or 0
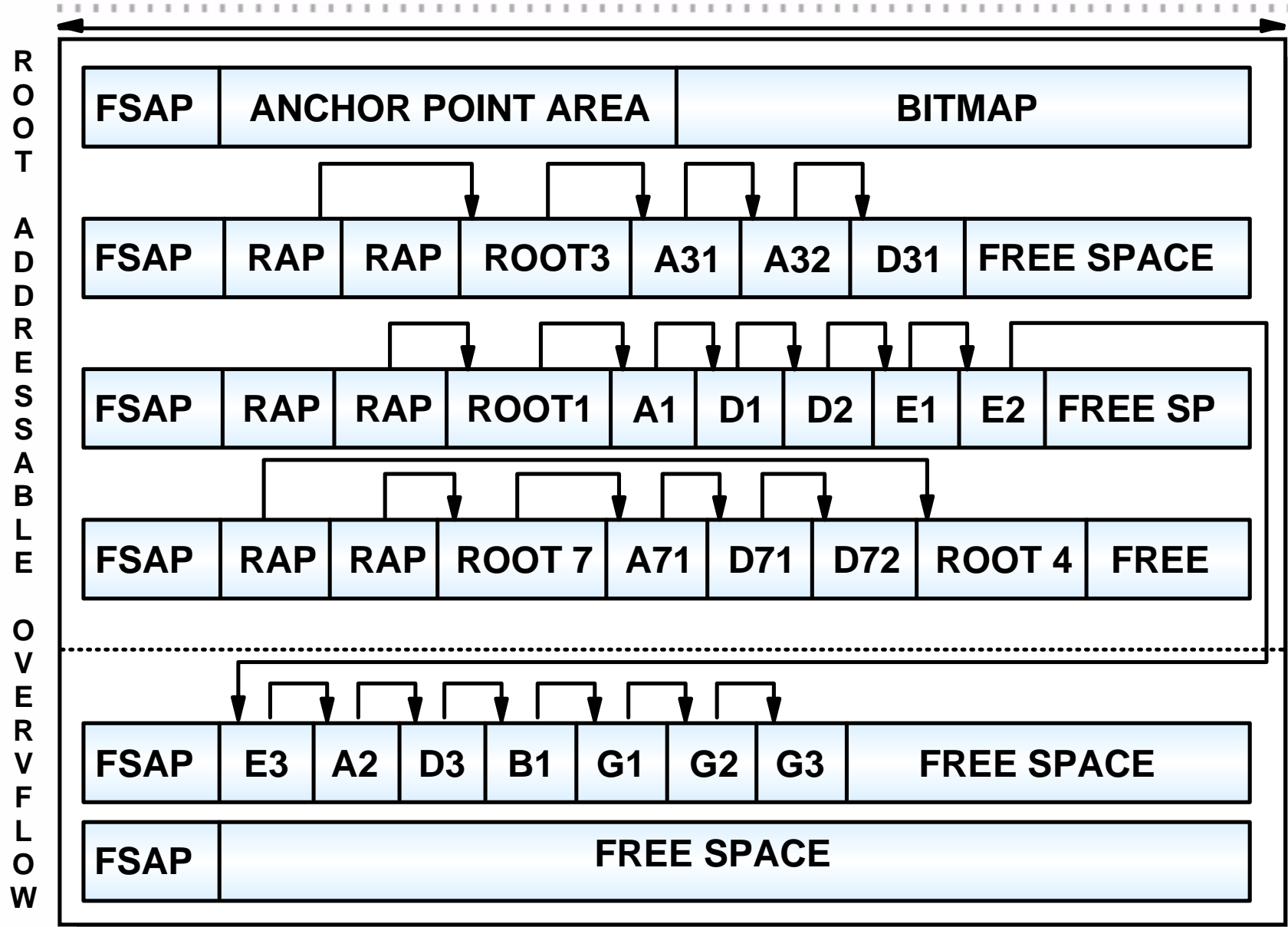
Special HD Fields ...

- Free Space Element

FSE CHAIN	AVAILABLE LENGTH	TASK ID
◄——— 2 ———►	◄——— 2 ———►	◄——— 4 ———►

  - ► First 2 bytes are offset, in bytes, to next FSE
    - − Zero if this is the last FSE in the block or CI
  - ► Second 2 bytes are length of free space, including FSE
    - − No FSE is created if free space is less than 8 bytes long
  - ► Last 4 bytes is the task ID of the program that freed the space
    - − Allows a program to free and reuse the same space without contention
   - − Useful in determining who freed the space

HDAM Storage

R	FSAP	ANCHOR POINT AREA		BITMAP	
O					
O					
T					

A	FSAP	RAP	RAP	ROOT3	A31	A32	D31	FREE SPACE

| D |
| R |
| E | FSAP | RAP | RAP | ROOT1 | A1 | D1 | D2 | E1 | E2 | FREE SP |
| S |
| S |

| A |
| B | FSAP | RAP | RAP | ROOT 7 | A71 | D71 | D72 | ROOT 4 | FREE |
| L |
| E |

| O |
| V |
| E | FSAP | E3 | A2 | D3 | B1 | G1 | G2 | G3 | FREE SPACE |
| R |
| V |
| F | FSAP | FREE SPACE |
| L |
| O |
| W |

IBM Software Group

IBM

44

HDAM Storage ...

- Root Addressable Area (RAA)
  - ► Number of blocks or CIs defined in RMNAME parameter
  - ► Primary storage area for roots and dependents
    - – Number of dependents at initial load is limited by RMNAME
    - – Insert until specified bytes limit would be exceeded
  - ► All RAPs are in the RAA
  - ► Location is determined by Randomizer specified in RMNAME
    - – Randomizer input is the root segment's key
    - – Randomizer output is a block number and RAP number
    - – Keys that randomize to same block and RAP are synonyms
    - – Synonyms are chained using PTF pointers
    - – Chain is ascending key sequence or by insert rules
- Overflow Area
  - ► For segments that do not fit in the RAA
 - ► No RAPs are present in the overflow area

HIDAM Storage

IBM Software Group
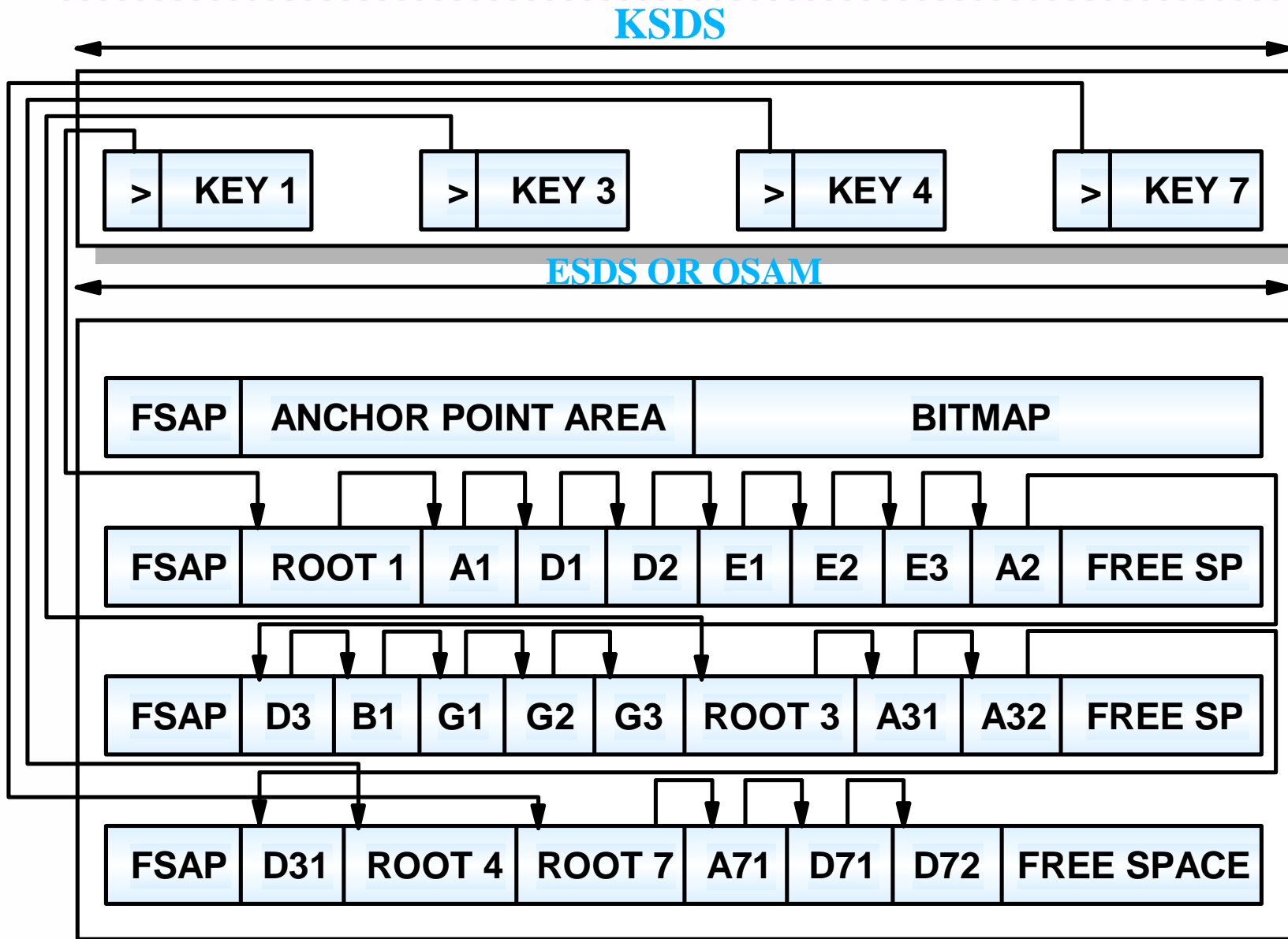
HIDAM Storage ...

- Data Component
  - ► A VSAM ESDS or OSAM data set
  - ► No RAA or Overflow portions
  - ► Database records are stored in key sequence
  - ► Roots must have unique keys
  - ► Segments in hierarchic sequence
  - ► You can specify that free space be left after loading
    - – A percentage in each block or CI
    - – Every Nth block or CI
- Index Component
  - ► VSAM KSDS
  - ► The index is a root-only database
  - ► One index segment for each database root

DB	Pointer to Root	Root Key
◄─ 1 ─►	◄──── 4 ────►	◄─ Varies ─►
◄──────── Prefix ────────►		◄─ Data ─►

IBM
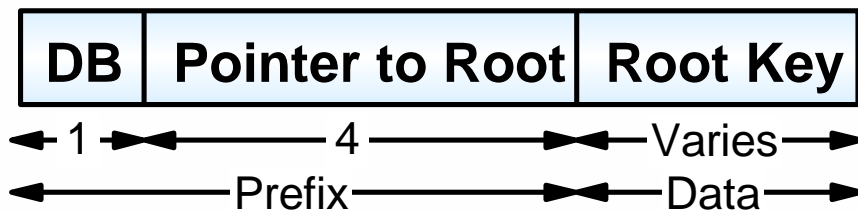
HIDAM RAP

- One RAP per block or CI if PTR=T or PTR=H for the root
  - ► No RAP is generated if PTR=TB or PTR=HB
  - ► No RAP is generated if PTR=NOTWIN

- Roots are chained from RAP in reverse order of insertion
  - ► RAP points to most recently inserted root
  - ► Each root points to previously inserted root
  - ► First root inserted has a zero pointer

- Index must be used to process roots sequentially
  - ► Index must also be used if NOTWIN is specified

- Remember that TWIN is the default
  - ► Specify something useful!
  - ► Use backward pointers if you process roots sequentially
  - ► Use NOTWIN if you only do random processing

IBM Software Group

Processing HD Databases

- Delete
  - ► The segment and all of its dependents are removed
  - ► FSE is used indicate the space is free
    - − Create a new FSE and update the FSAP/FSE Chain
    - − Update length field of preceding FSE
  - ► Pointers are updated
- Replace
  - ► No change in length or fixed-length
    - − Overwrite old segment with updated segment
  - ► Shorter segment
    - − Space previously occupied is freed
    - − FSE created if at least 8 bytes shorter
  - ► Longer segment
    - − If adjacent free space lets it fit, store in original location
    - − If no space available, separated data
      Data part goes to overflow with prefix of SC and DB=x'FF'
      Bit 4 of DB in original prefix is turned on
      Pointer to data in overflow is built after prefix
      Remainder of space is freed

Processing HD Databases ...

- Insert
  - ► Store in the Most Desirable Block (MDB)
    - − HDAM root MDB

      The one which is selected by the randomizer

      The one containing its previous synonym
    - − HIDAM root MDB

      If no backward pointer, same as the next higher key root

      If backward pointer, same as the next lower key root
    - − Dependents

      If Physical, same as parent or previous twin

      If Hierarchic, same as previous segment in hierarchy
  - ► Second most desirable block
    - − Nth Block or CI left free during loading

      If in buffer pool or bitmap shows space available
    - − Specified by FRSPC parameter

      If not specified, then no second MDB

HD Space Search Algorithm

- In the MDB (this will be in the buffer pool)
- In the second MDB
- Any block in the buffer pool on the same cylinder
- Any block on the same track
  - ► If the bitmap shows space available
- Any block on the same cylinder
  - ► If the bitmap shows space available
- Any block in the buffer pool within +/- SCAN cylinders
- Any block within +/- SCAN cylinders
  - ► If the bitmap shows space available
- Any block at the end of the data set is in the pool
- Any block at the end of the data set
  - ► If the bitmap shows space available
  - ► Extend the data set if necessary
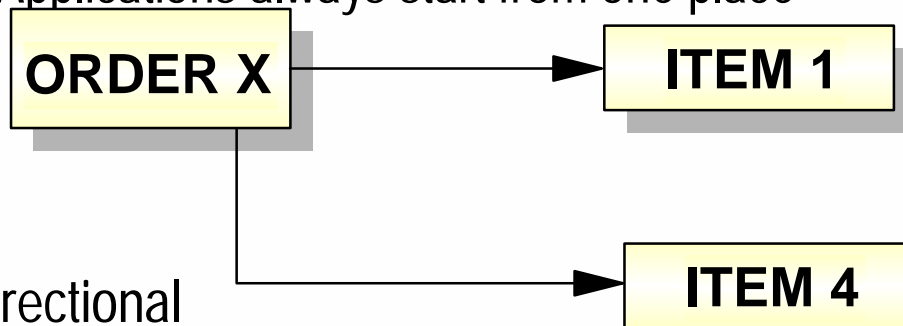- Any block where the bitmap shows space
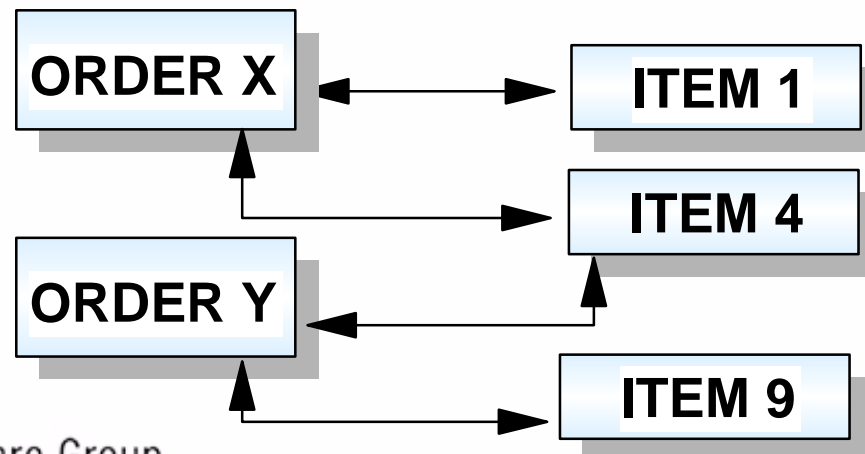
Logical Relations

Logical Relations

Types

- Unidirectional
    - ► A one-way relationship from one database record to another
    - ► Applications always start from one place
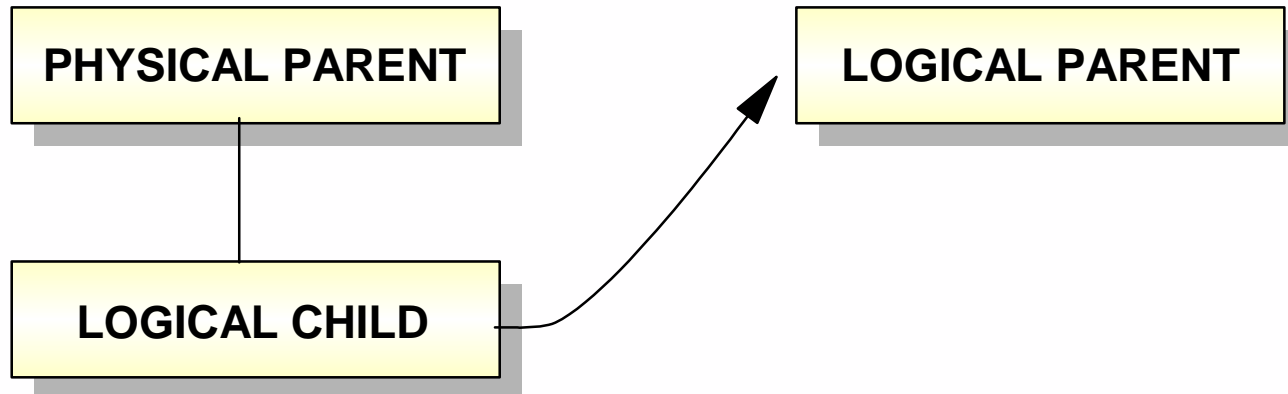
| ORDER X | → | ITEM 1 |

→ ITEM 4

- Bi-directional
    - ► A two-way relationship between database records
    - ► Applications may start on either side
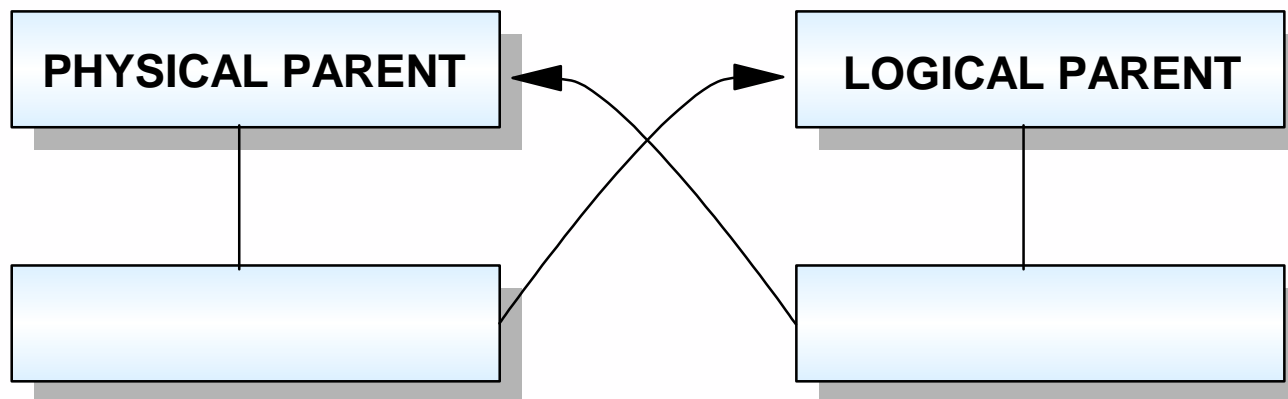   - ► IMS maintains both sides of bi-directional relationships

ORDER X ←→ ITEM 1

ORDER Y → ITEM 4

ORDER Y ← ITEM 9

IBM

How Logical Relationships are Implemented

Unidirectional

PHYSICAL PARENT	LOGICAL PARENT

LOGICAL CHILD

Bi-directional

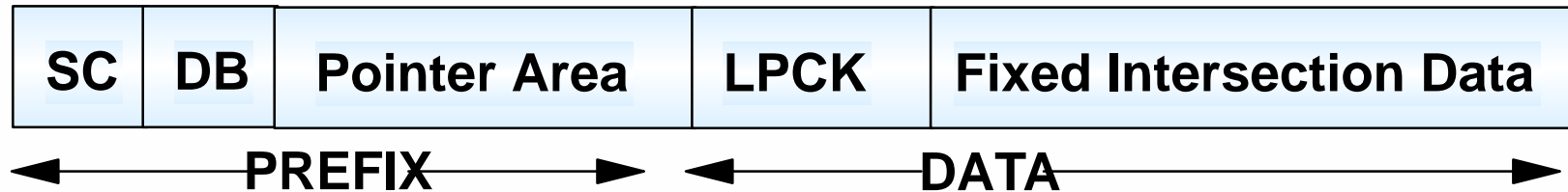| PHYSICAL PARENT	LOGICAL PARENT

- Logical Parent (LP) Pointer
  - ► In the Logical Child segment
  - ► points to logical parent

The Logical Child

| SC	DB	Pointer Area	LPCK	Fixed Intersection Data

◄——————— PREFIX ——————► ◄——————— DATA ———————►

- **Logical Parent Concatenated Key**
  - ► Sequence fields of all segments from root to logical parent
  - ► Always appears to the application program
  - ► May or may not be physically stored with logical child
    - – If not stored, IMS generates it on retrieval

- **Logical Parent Pointer**
  - ► The LPCK if it is physically stored
    - – Must be used if logical parent database is HISAM
    - – This is called a symbolic pointer
  - ► A 4-byte pointer in the segment prefix
    - – May only be used if logical parent database is HD
    - – The only kind of pointer that can exist in HISAM

- **Fixed Intersection Data**
  - ► Data that is dependent on the logical relation
  - ► Maintained on both sides of a bi-directional relation
  - ► Variable intersection data is in dependents of the logical child
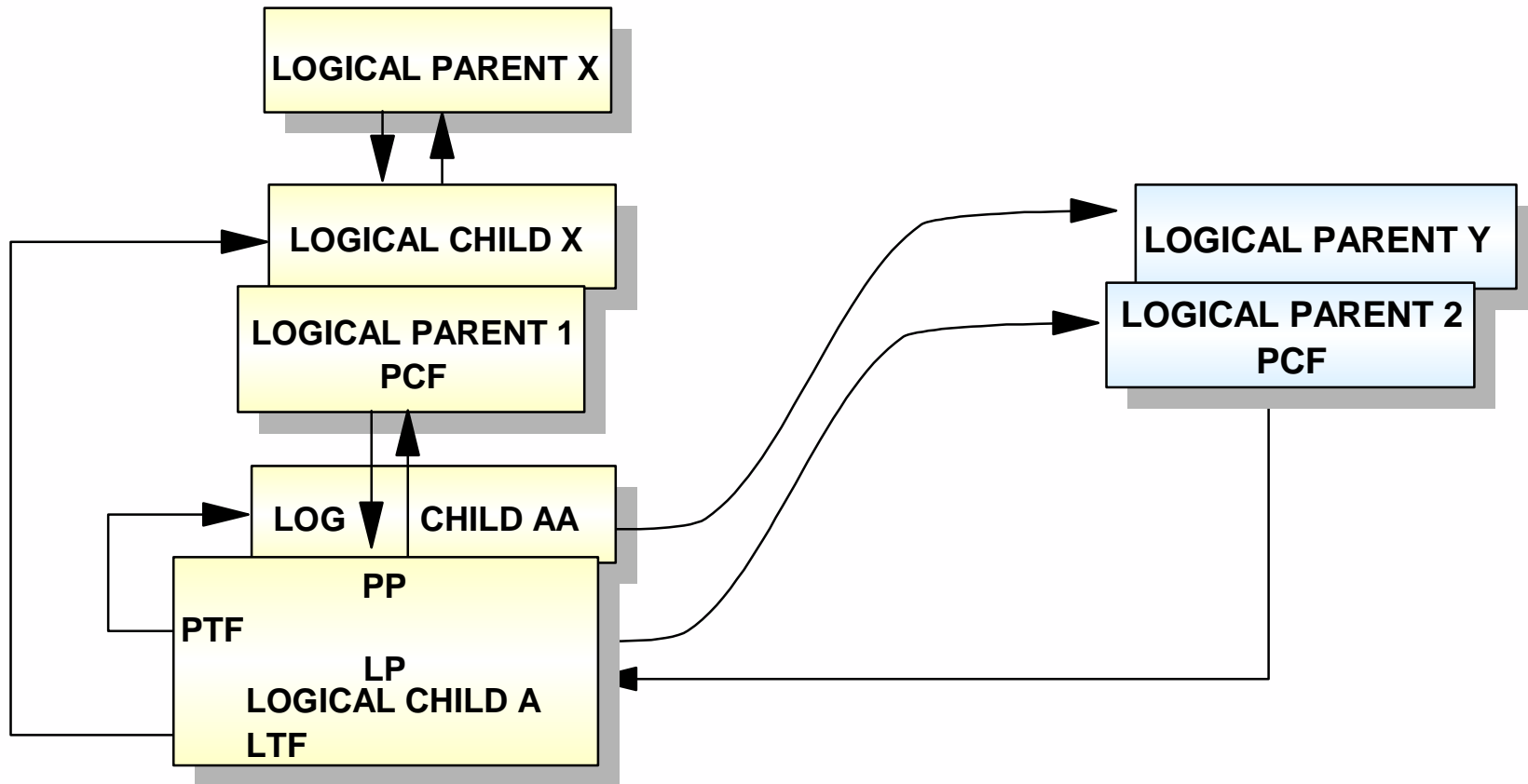
IBM Software Group

IBM

Bi-directional Physical Pairing
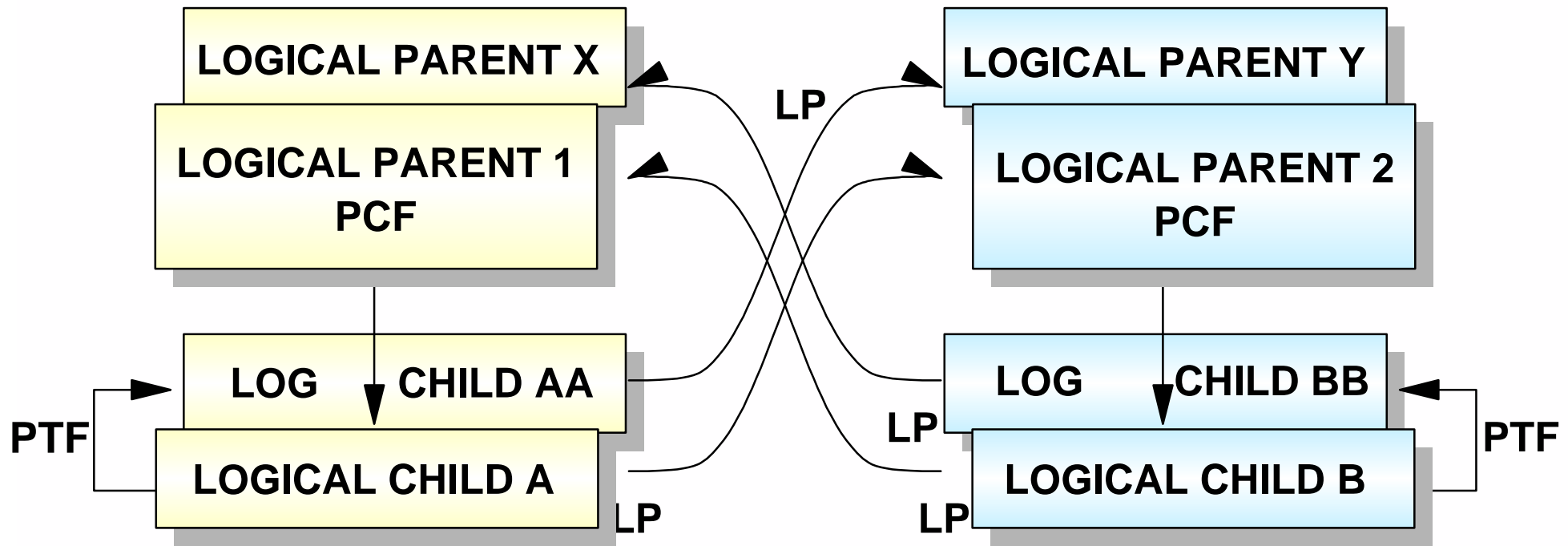
- Physical or Hierarchic relate Physical Parent and Logical Children
- Logical Parent relates Logical Child to Logical Parent
- Requires a physical segment on both sides of the relation

IBM Software Group

Bi-directional Virtual Pairing

- Logical Child First (LCF) replaces PCF
- Logical Twin Forward (LTF) replaces PTF
- Physical Parent (PP) replaces (LP)
- Physical segment only exists on one side of relation
- Real Logical Child must be in HD database

Logical Relation Prefix

- Logical Child Prefix
  - ► PP, LTF and LTB only present if virtual pairing

SC	DB	HF	HB	PP	LTF	LTB	LP

OR

SC	DB	PTF	PTB	PP	LTF	LTB	LP	PCF	PCL

- Logical Parent Prefix
  - ► PP only if a lower level segment is a logical parent

SC	DB	HF	HB	PP	LCF	LCL

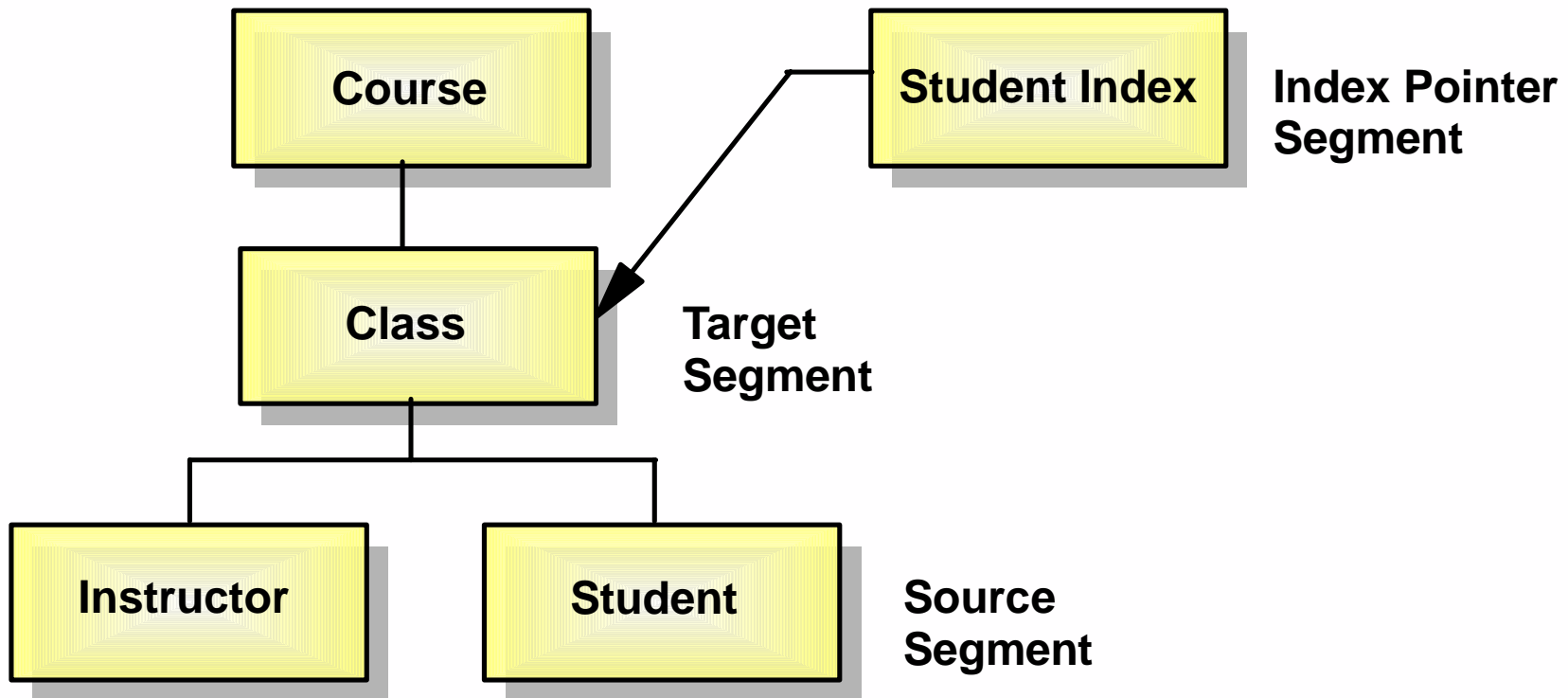| SC	DB	PTF	PTB	PP	PCF	PCL	LCF	LCL

IBM Software Group

Secondary Indices

Topic

Secondary Indices

Why Secondary Indices

- Processing sequence other than root key
    - ▶ Avoid scan for non-key field
- Direct access to lower level segments
    - ▶ Faster processing

```
        ┌─────────────┐                    ┌──────────────────┐
        │   Course    │────────            │  Student Index   │   Index Pointer
        └─────────────┘        \           └──────────────────┘   Segment
               │                \
        ┌─────────────┐          \▼
        │    Class    │          Target
        └─────────────┘          Segment
               │
        ┌──────┴──────┐
┌─────────────┐  ┌─────────────┐
│ Instructor  │  │   Student   │   Source
└─────────────┘  └─────────────┘   Segment
```
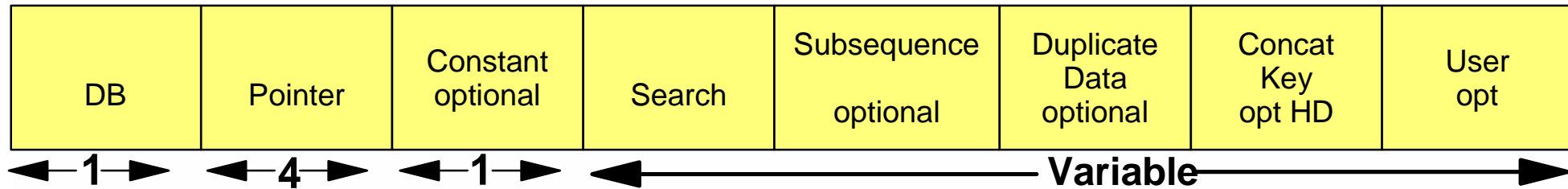
IBM Software Group

IBM

Secondary Index (SI)

- Can be based on HISAM, HDAM, or HIDAM
- Is a separate database
  - ► Can be processed on its own
- Uses fields from the source segment to create a key
- Access via a secondary index is to the target segment
- Invisible to the application
  - ► PROCSEQ = on PCB tells IMS to use the secondary index
  - ► Application must use XDFLD name in the SSA
- Limits on secondary indices
  - ► 32 secondary indices onone segment type
  - ► 1000 secondary indices for a database
- Secondary index is a special kind of logical relation

Fields in the Index Pointer

DB	Pointer	Constant optional	Search	Subsequence optional	Duplicate Data optional	Concat Key opt HD	User opt

←1→ ←4→ ←1→ ←———————————————————— Variable ——————————————————→

- **Pointer is used when target is in HD database**
- **Constant is used for shared secondary indices**
  - ► More than one SI in the same database
- **Search is made up of up to 5 fields form the source**
  - ► This is the key of the secondary index
- **Subsequence is up to 5 fields from source or IMS-generated values**
  - ► Used to make the secondary index key unique
- **Duplicate Data is up to 5 fields from the source**
  - ► Only used when processing the SI as a database
- **Concatenated Key is the symbolic pointer to the target**
  - ► Required when the target is in HISAM database
- **User Data is anything you want to stick in there**
  - ► Only used when processing the SI as a database

IBM Software Group

IBM

Where to look for further information...

- IMS Manuals
  - ► Administration Guide: Database Manager
    - – Chapter 4 - Designing a Full-Function Database, Choosing a Database Type
    - – Chapter 5 - Choosing Additional Database Functions
  - ► Utilities Reference: System
    - – Chapter 1 - Database Description (DBD) Generation
- IMS Redbooks
  - ► IMS Primer (SG24-5352) - Part 3 - IMS Database Manager
    - – Four Chapters with very good information on the topics covered in this presentation
- Introduction to HALDB - Session E04 (Vern Watts)

IBM Software Group

IBM