E11

Application Design and Programming with HALDB

Rich Lewis

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

RETURN TO INDEX

Abstract

IMS Version 7 adds High Availability Large Database (HALDB) capabilities. HALDB supports very large databases. By splitting a database into multiple partitions, HALDB allows it to contain up to 40 terabytes! In spite of this, an application program continues to see one database. That is, the database is addressed by one PCB.

In general, application programs do not have to be modified when a database is migrated to HALDB. On the other hand, there are cases where application programs must be changed. Also, an installation may want to take advantage of some capabilities which may require application changes.

This session describes changes that you may need to make or want to make when a database is migrated to HALDB. Considerations for processing partitions in parallel, processing secondary indexes as databases, initially loading HALDB databases, handling unavailable partitions, and converting from user partitioning are explained.

A version of this presentation which includes extensive notes is available at:
http://www.ibm.com/support/techdocs/atsmastr.nsf/PubAllNum/PRS490

Agenda

- **Introduction**
  - Highlights of HALDB
  - Partitioning
- **Application Considerations**
  - Initial Loads
  - Processing Partitions in Parallel
  - Restricting a PCB to One Partition
  - Handling Unavailable Partitions
  - Processing Secondary Indexes as Databases
  - Converting from User Partitioning

Highlights of HALDB

HALDB (High Availability Large Database)

Large Database

Up to 10,010 data sets per database!

- Databases are partitioned — Greater than 40 terabytes
  - Up to 1001 partitions per database
  - Partitions have up to 10 data set groups

High Availability Database

- Partition independence
  - Allocation, authorization, reorganization, and recovery are by partition
- Simplified and shortened reorganization process
  - Partitions may be reorganized in parallel
  - Reorganization of partition does not require changes to secondary indexes or logically related databases which point to it
  - Prefix Resolution, Prefix Update, and secondary index rebuilds are eliminated

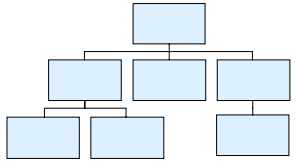
Highlights

New database types

- PHDAM - partitioned HDAM
- PHIDAM - partitioned HIDAM
  - Index is also partitioned
- PSINDEX - partitioned secondary index

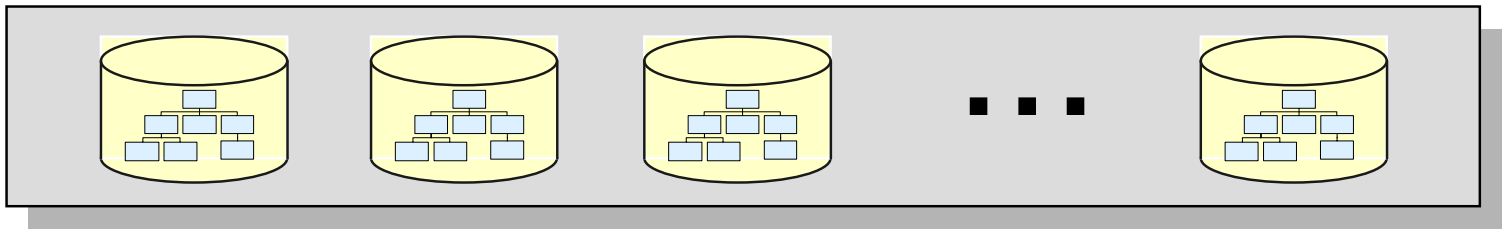
Hierarchic structure is maintained

- A database record resides in one partition



Partition selection (deciding in which partition a record resides)

- By key range or by user exit routine



Highlights

Logical relationships and secondary indexes are supported

- Secondary indexes may be partitioned

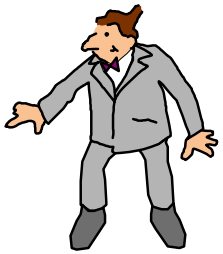
Parallel Processing

- Reorganizations
  - Partitions may be reorganized independently
  - Partitions may be reorganized in parallel
- Application processing
  - Partitions may be processed in parallel
  - DBRC authorization is by partition (not entire database)

The Application Programming News

The good news:

- In general, application programs do not have to be changed when databases are migrated to HALDB



The OK news:

- You may want to change application programs to take advantage of new opportunities with HALDB

The not so good news:

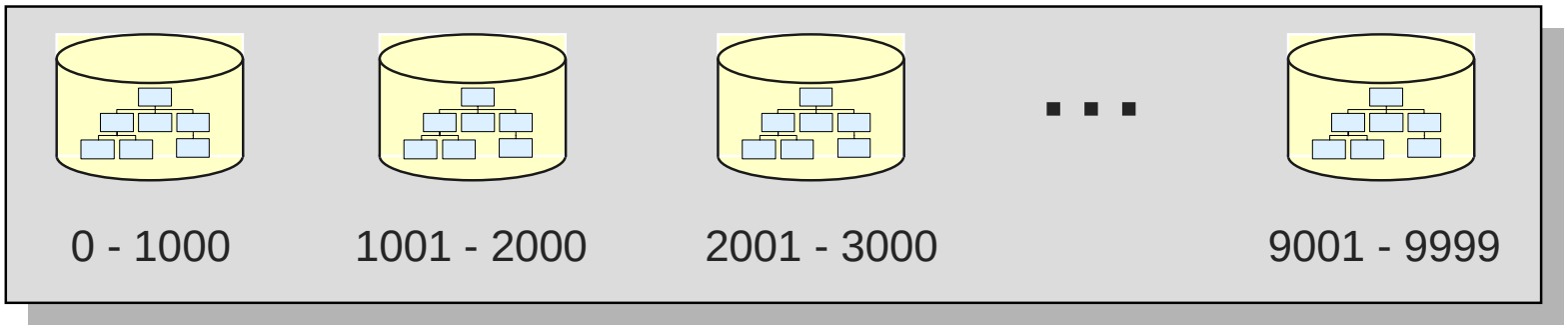
- You may have to change a small number of application programs when databases are migrated to HALDB

Partitioning

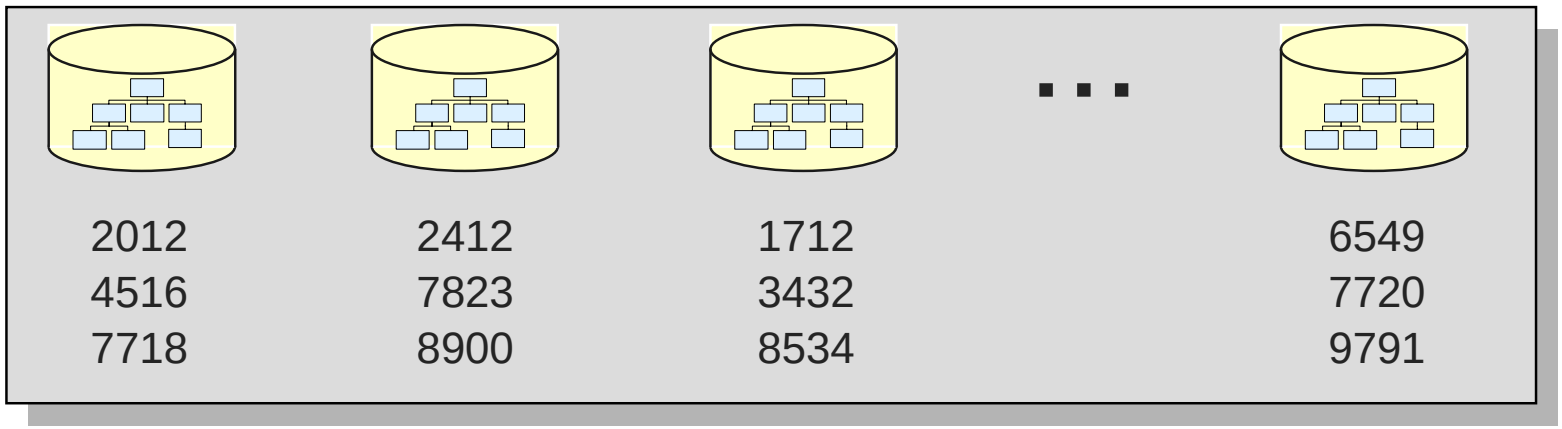
Partitioning Choices

Two methods of partitioning

- Key range
  - Each partition is assigned a range of root segment keys



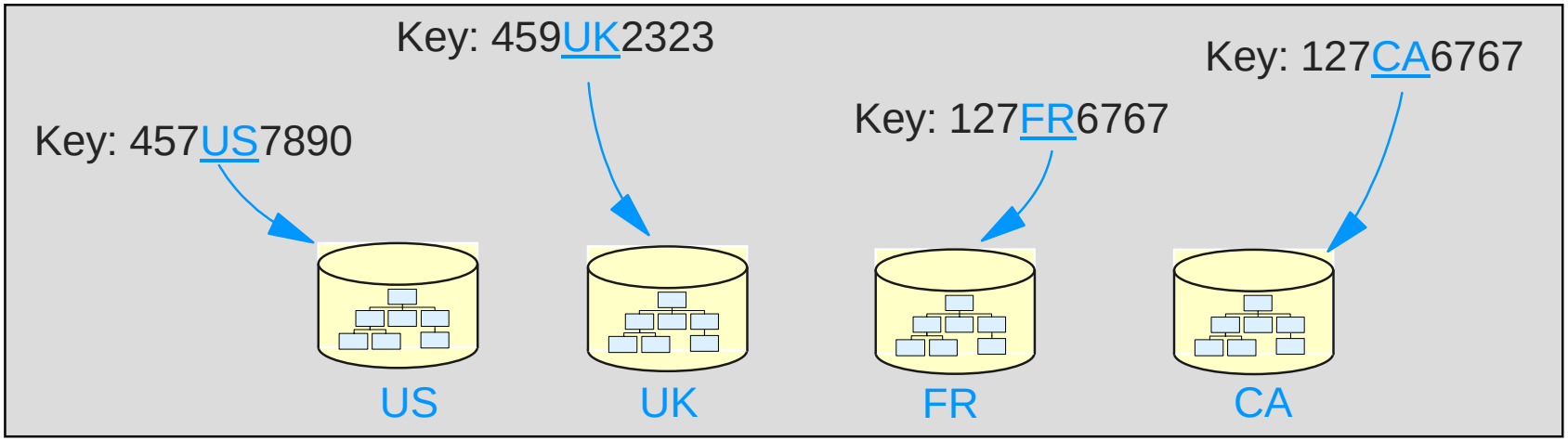
- Partition Selection Exit routine
  - The exit routine assigns a root segment to a partition based on its key



Partitioning Choices

Partition Selection Exit routine example:

- Assign records by country code which is in the root key



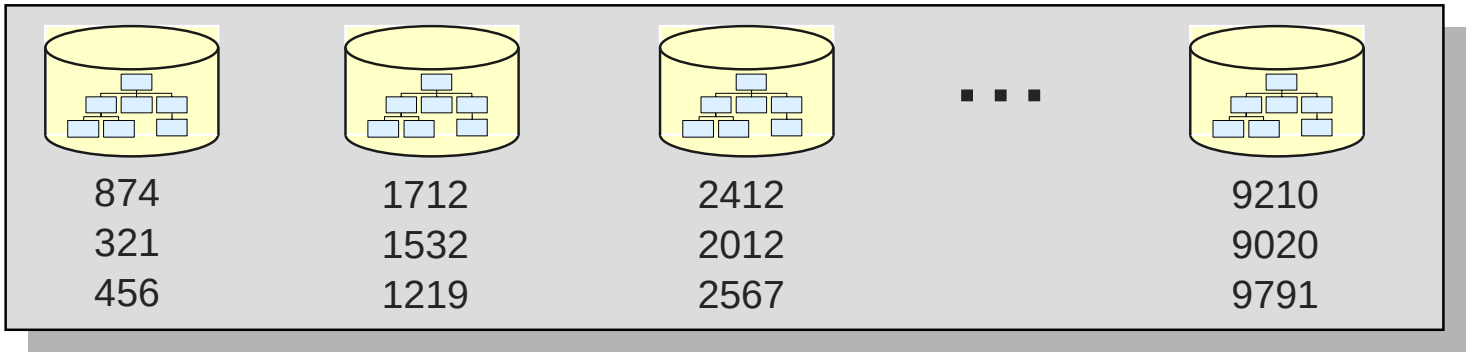
Order of Segments in a Partition

- **PHDAM - Partitioned HDAM**
  - Roots are in random order within a partition
- **PHIDAM - Partitioned HIDAM**
  - Roots are in sequential order within a partition
- **PSINDEX - Partitioned Secondary Index**
  - Entries are in sequential order within a partition

Order of Segments in a PHDAM Database

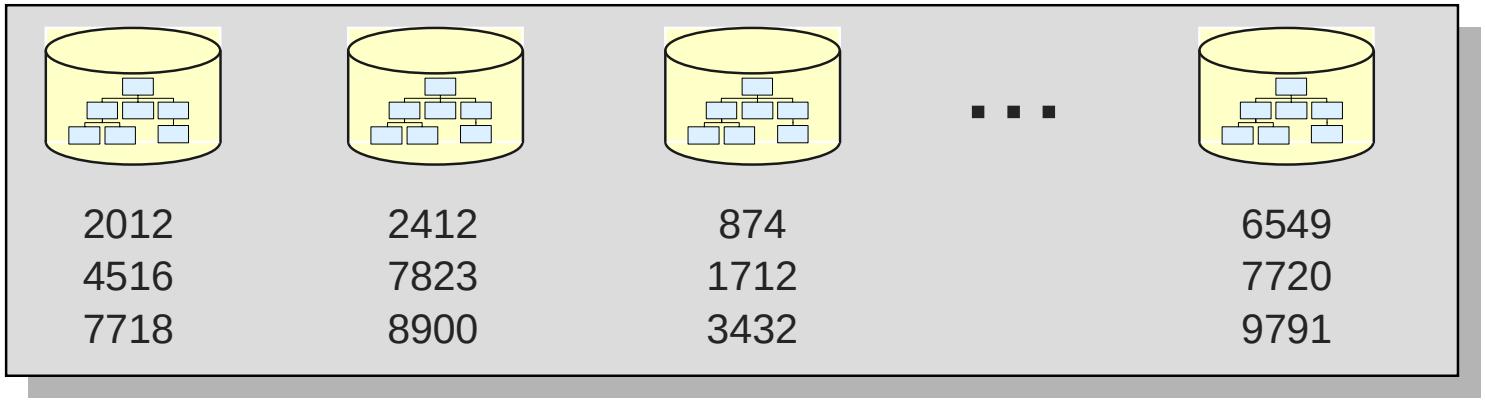
PHDAM with key range partitioning

- Keys are sequential between partitions, random within a partition



PHDAM with Partition Selection Exit routine

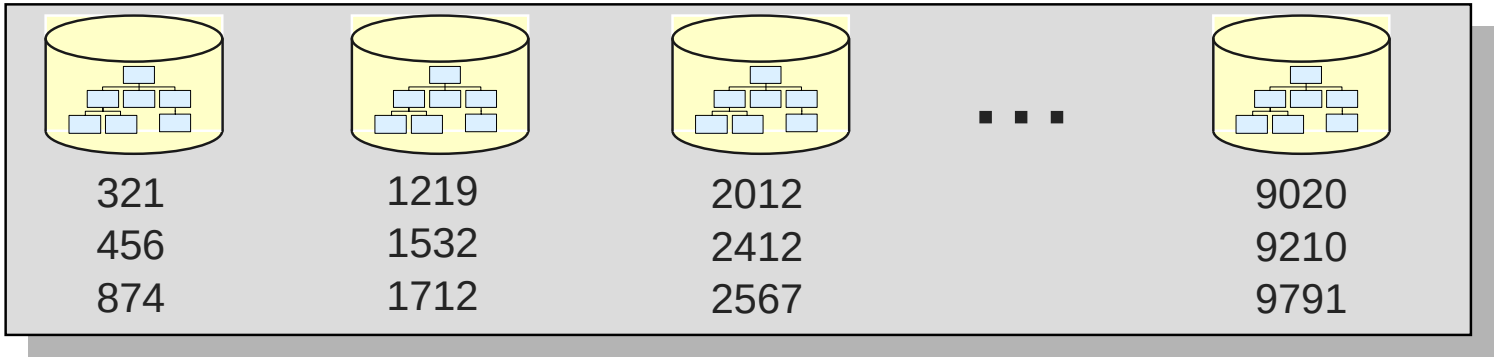
- Keys are not sequential between partitions, random within a partition



Order of Segments in a PHIDAM Database

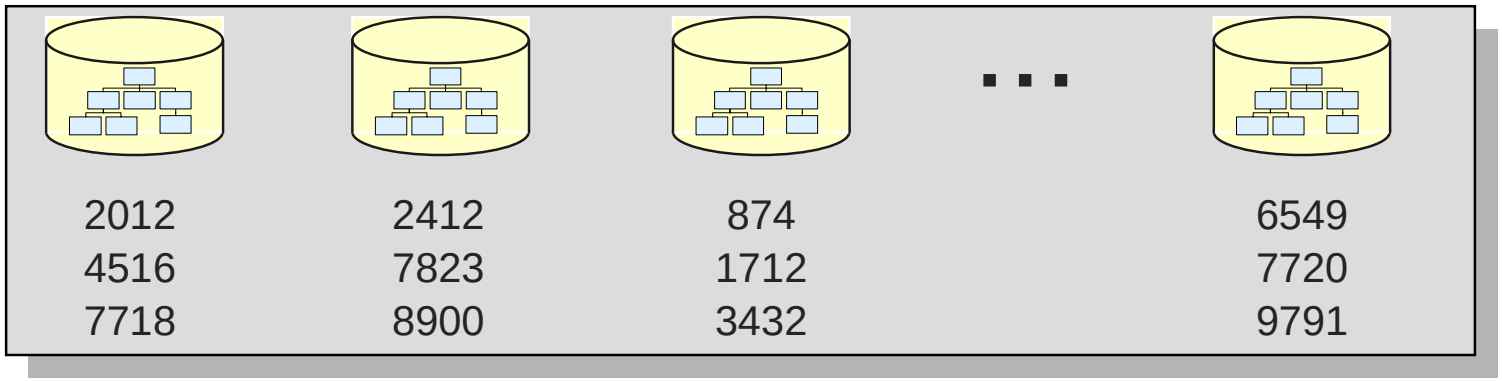
PHIDAM with key range partitioning

- Keys are sequential between partitions, sequential within a partition



PHIDAM with Partition Selection Exit routine

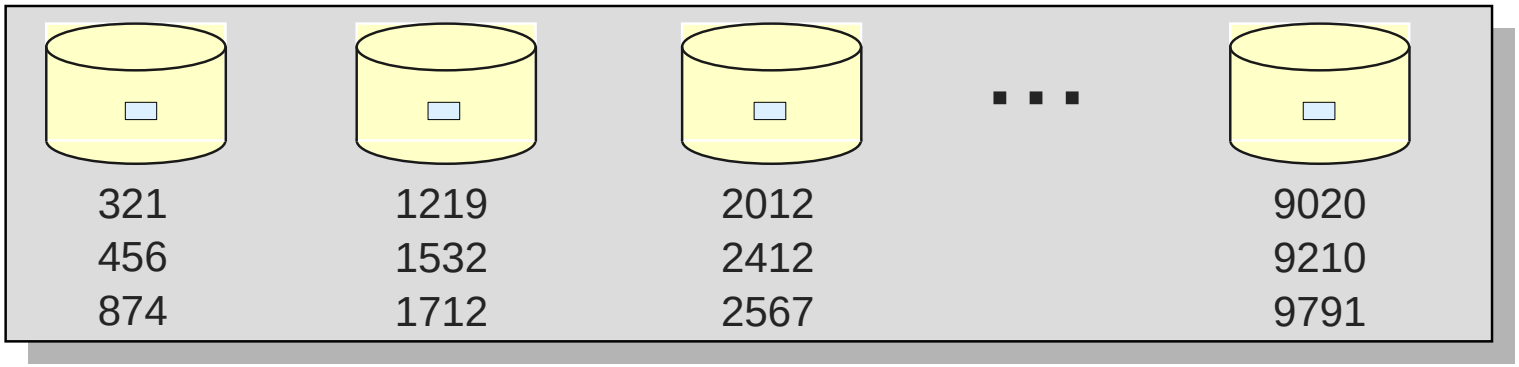
- Keys are not sequential between partitions, sequential within a partition



Order of Segments in a PSINDEX Database

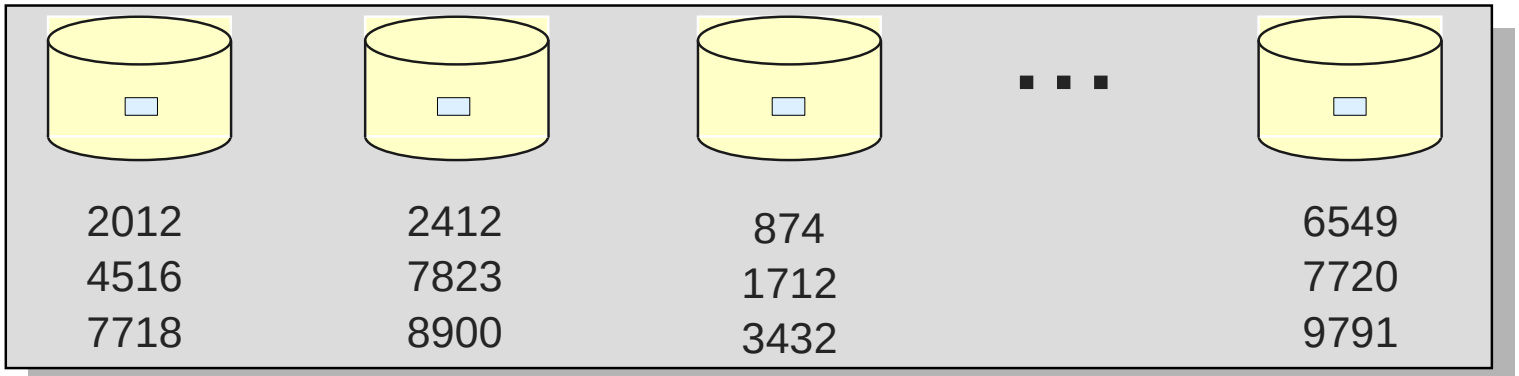
PSINDEX with key range partitioning

- Keys are sequential between partitions, sequential within a partition



PSINDEX with Partition Selection Exit routine

- Keys are not sequential between partitions, sequential within a partition



Application Considerations

- Initial loads
- Processing Partitions in Parallel
- Restricting a PCB to One Partition
- Handling Unavailable Partitions
- Processing Secondary Indexes as Databases
- Converting from User Partitioning

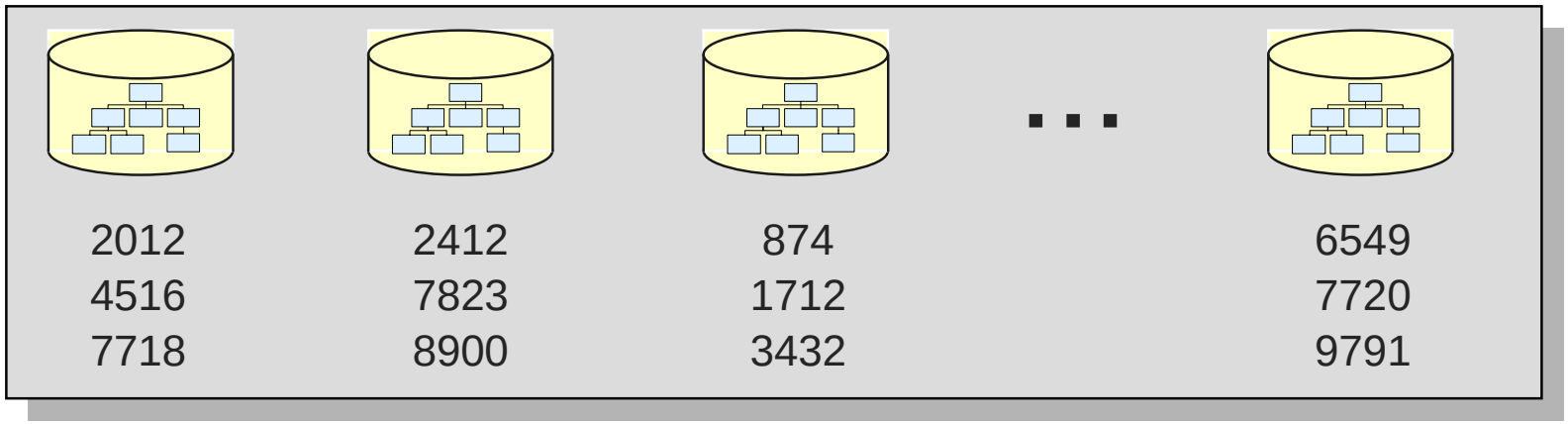
Initial Loads

Initial Loads

Initial load uses PROCOPT=L or PROCOPT=LS in PCB

- Same as non-HALDB databases

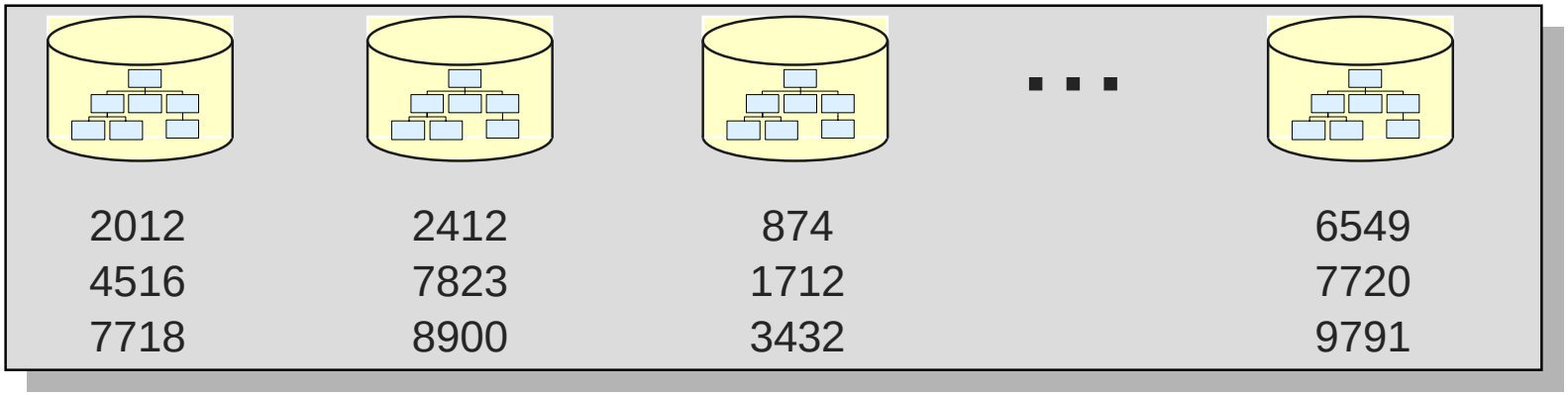
Loads of PHDAM roots may be in any key sequence



- Could load 874, 1712, 2012, 2412, 4516, 6549, 7718, 7720, ...
- Could load 8900, 2012, 7823, 4516, 7718, 2412, 874, 1712, ...
- No changes required for load program
  - If you sort by RAP sequence for faster loads, you will want to sort by RAP sequence within partitions
    - Phys. Seq. Sort for Reload (PSSR) in High Perf. Load will do this

Initial Loads

- **Loads of PHIDAM roots must be in key sequence within a partition**
  - May be in key sequence across the entire database

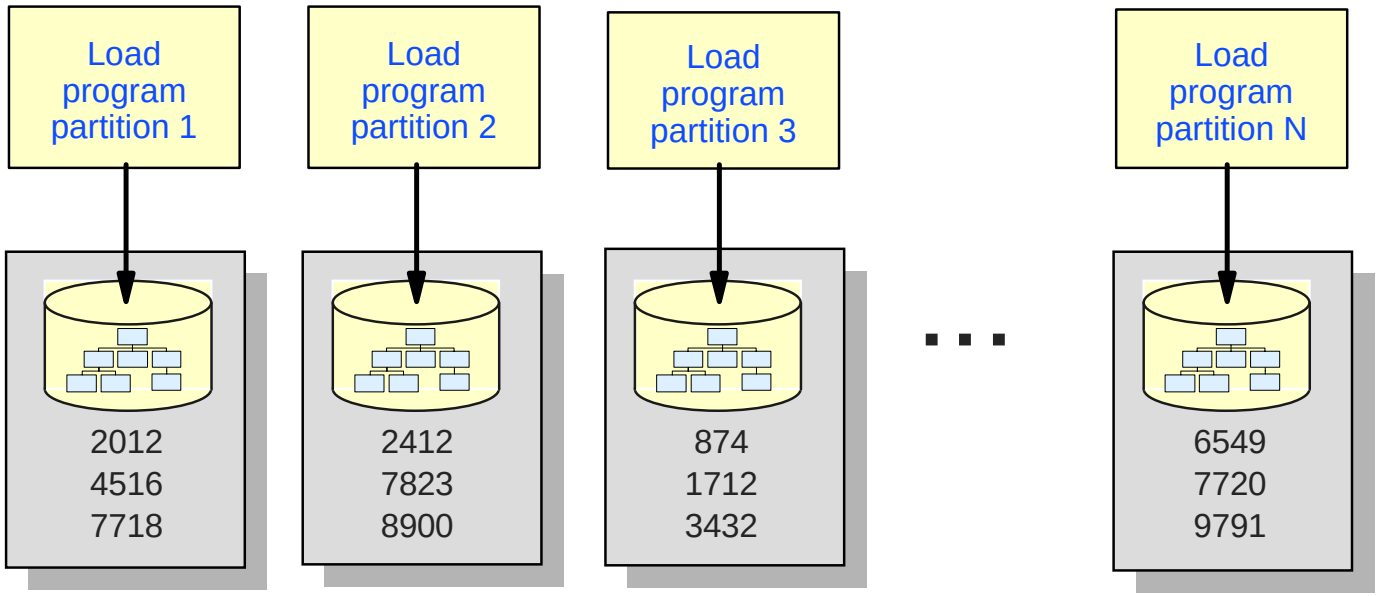


  - Could load 874, 1712, 2012, 2412, 4516, 6549, 7718, 7720, ...
    - This is like HIDAM database order
    - No changes required for load program
  - Could load 2012, 4516, 7718, 2412, 7823, 8900, 874, 1712, ...

Initial Loads in Parallel

Partitions may be initially loaded in parallel

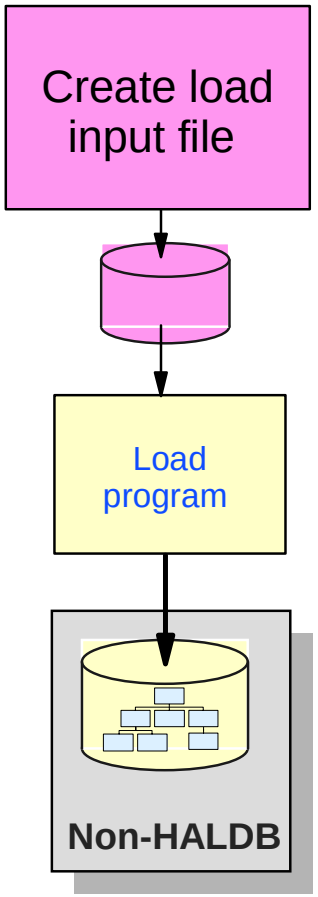
- Separate jobs for each partition
  - Multiple jobs cannot load records in the same partition
- Could make load of database much faster
- Current load programs may work without change
  - Input to program would have to be split by partition



Initial Loads in Parallel

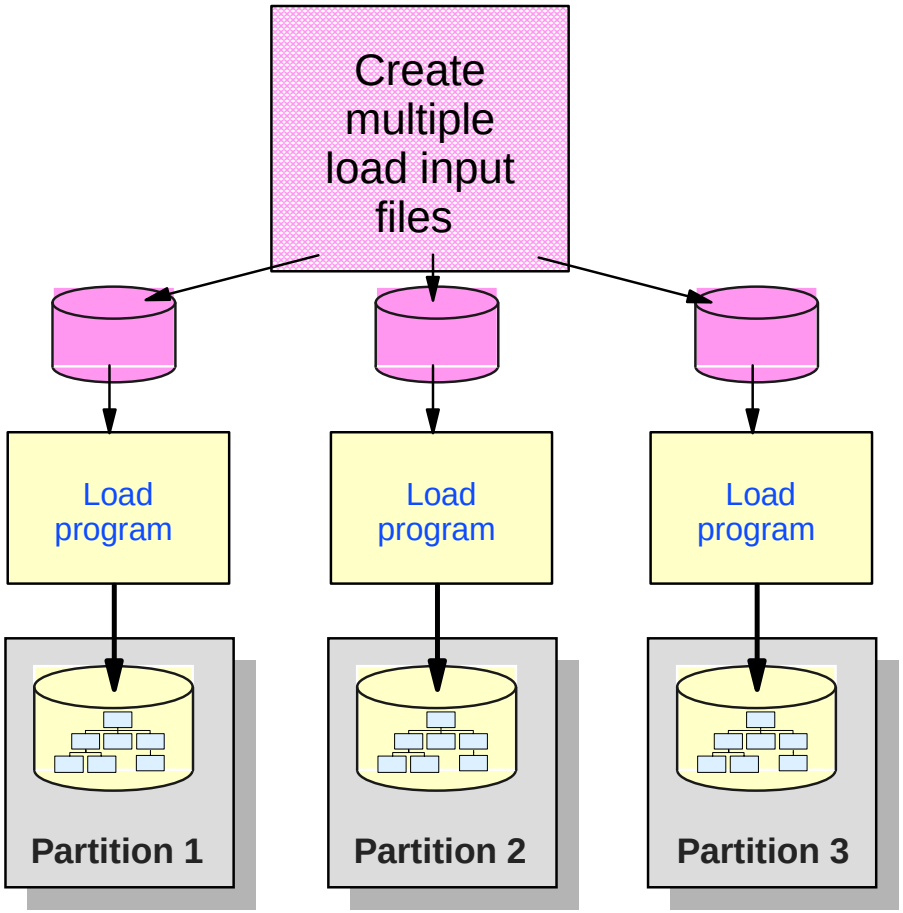
Non-HALDB process

- Step 1 creates file read by load program



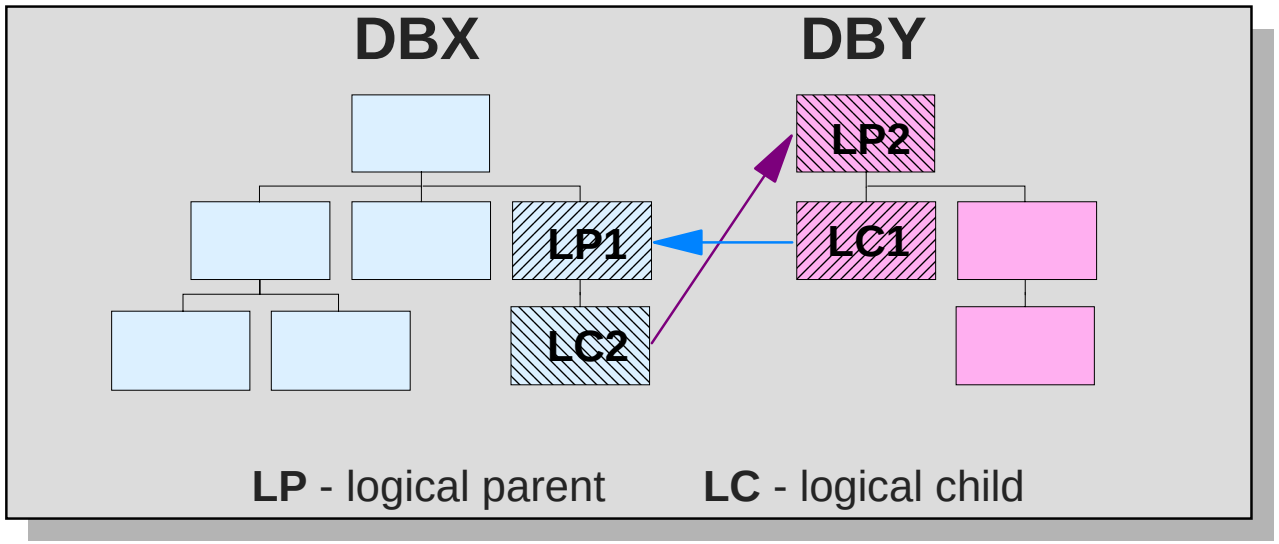
HALDB parallel load process

- Step 1 creates a file for each partition
- Load program is unchanged



Initial Loads with Logical Relationships

Logical relationships



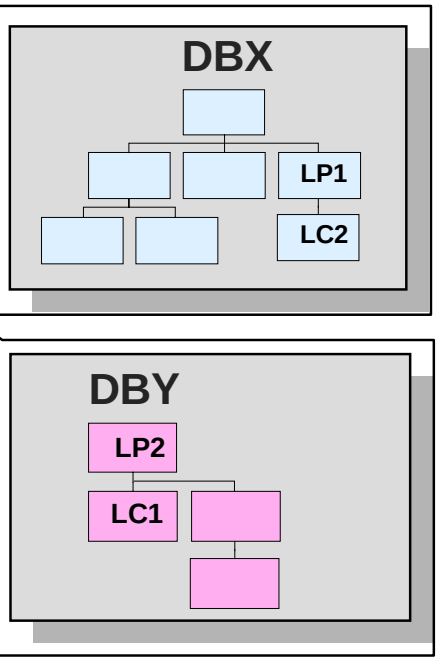
Logical children cannot be loaded in HALDB

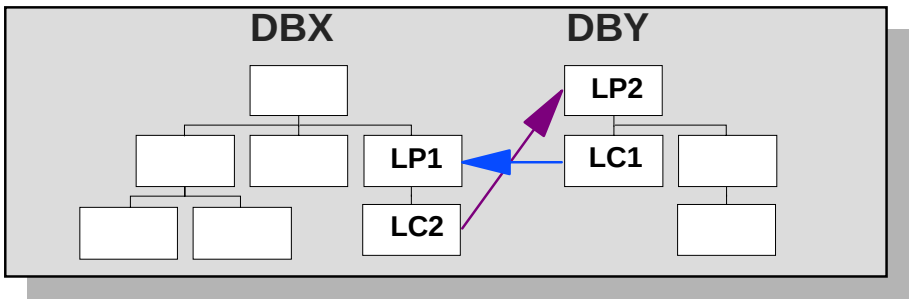
- Attempts to load logical children receive 'LF' status code
- They must be added by update programs (PROCOPT=I or A)
  - Cannot insert logical child without logical parent

Initial Loads with Logical Relationships

Non-HALDB process:

- Step 1: Prereorganization utility
- Step 2: Initial load DBX including logical children
  - Creates work file for logical relationships
- Step 3: Initial load DBY including logical children
  - Creates work file for logical relationships
- Step 4: Prefix Resolution utility
  - Sorts work file records
- Step 5: Prefix Update utility
  - Updates prefixes (pointers and counters) used for logical relationships

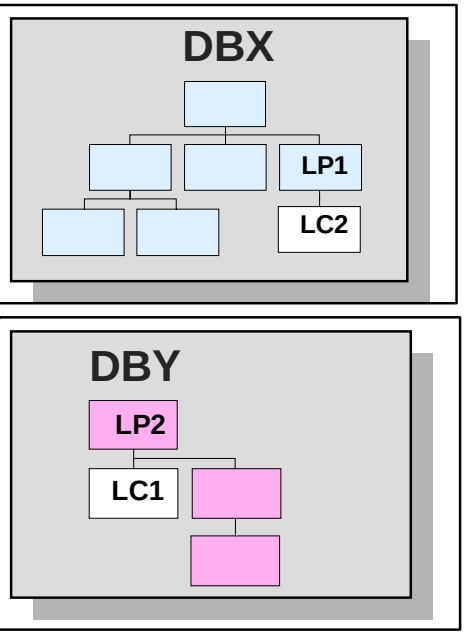




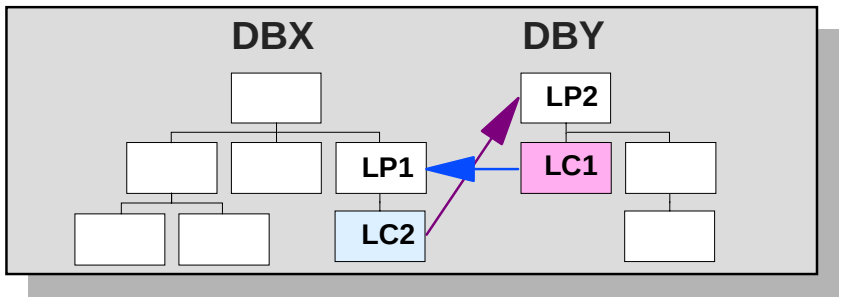
Initial Loads with Logical Relationships

Initial load programs for HALDB with logical children must be split

- Step 1: Initialize DBX partitions with Prereorg.
- Step 2: Initialize DBY partitions with Prereorg.
- Step3 : Initial load DBX without logical children
  - This is a modification to the existing load program
- Step 4 : Initial load DBY without logical children
  - This is a modification to the existing load program



- Step 5: Insert (PROCOPT=I or A) logical children in DBX or DBY
  - New program to insert only the logical children in one database
  - Insert will create paired logical child in other database



Processing Partitions in Parallel

Online, Batch, and Data Sharing

Online

- All regions or threads have concurrent access to all partitions
- Parallel processing is easily done (e.g. BMPs)

Data sharing

- All sharing subsystems have concurrent access to all partitions
  - Subsystems may be online subsystems or batch jobs
- Parallel processing is easily done (e.g. DLI batch jobs)

Batch (without data sharing)

- Multiple batch jobs may have read authorization for the same partitions
- Only one batch job can have update authorization for a partition
- Different batch jobs may concurrently update different partitions

Two Styles of Batch Programs

Random and Skip Sequential

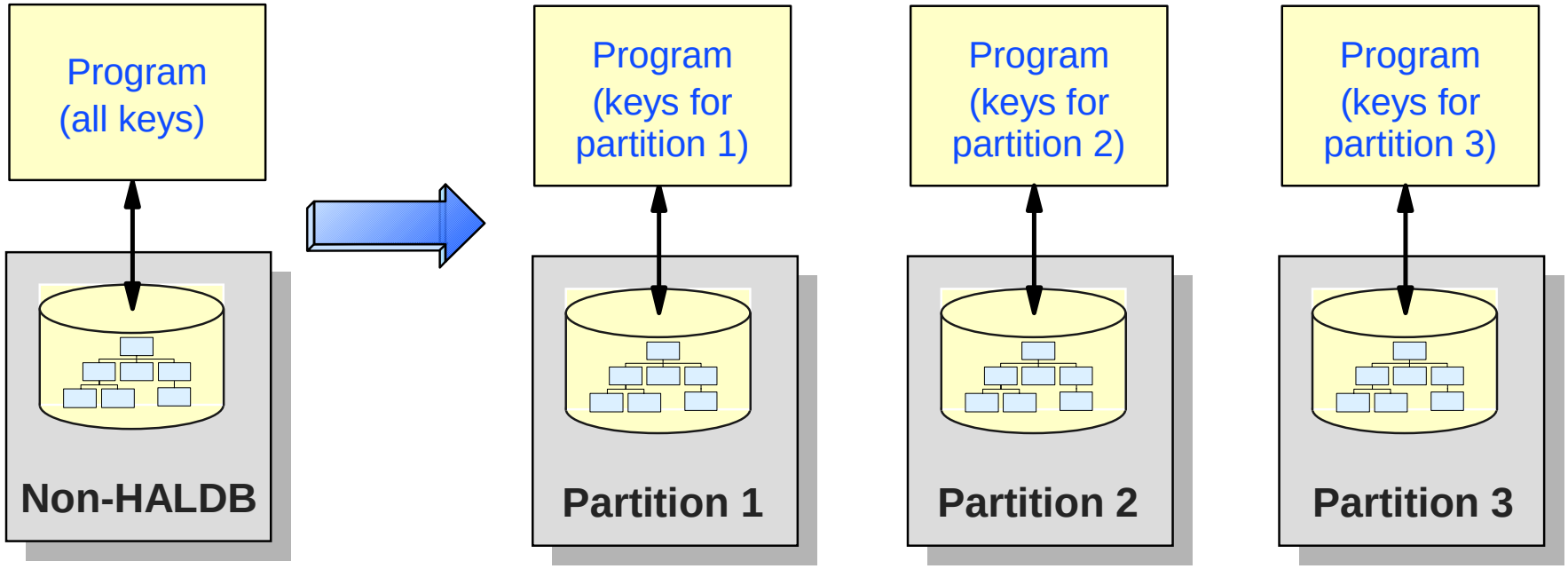
- Records are accessed by key
- Get calls are qualified on keys
  - Keys are known before calls are made
  - Program accesses only some of the records in the database
  - Program may be able to determine the partition before the call is made

Sequential

- Records are accessed by Get Next calls
- Get calls are not qualified on keys
  - Keys are not known before calls are made
  - Program typically accesses all of the records in the database
  - Program cannot know if the next record is in the same partition

Random or Skip Sequential

- Processing within a partition
- Program could be modified to restrict accesses to one partition
  - Restrict to the set of keys held by the partition



Sequential Processing within Partitions

- **Current program sequentially accesses database**
  - Accesses all records by doing GN calls
    - Begins with unqualified GN
    - Ends when 'GB' status code is returned (end of database)
- **Either**
  - Program must be modified to restrict accesses to one partition
    - Must start with first record in partition
    - Must recognize the end of the partition
- **Or**
  - HALDB Control Statement may be used
    - Restricts a PCB to a single partition

Restricting a PCB to One Partition

The HALDB Control Statement

- **Enhancement to IMS V7 and V8**
  - APAR PQ57313 for IMS V7
  - APAR PQ58600 for IMS V8
- **Control statement to limit PCB access to one partition**
  - Batch (DLI or DBB), BMP, or JBP region
  - Supported with PHDAM, PHIDAM, and PSINDEX
- **DFSHALDB DD statement:**

```
HALDB PCB=(nnn,pppppppp)

   nnn - DBPCB number
   pppppppp - partition name
```

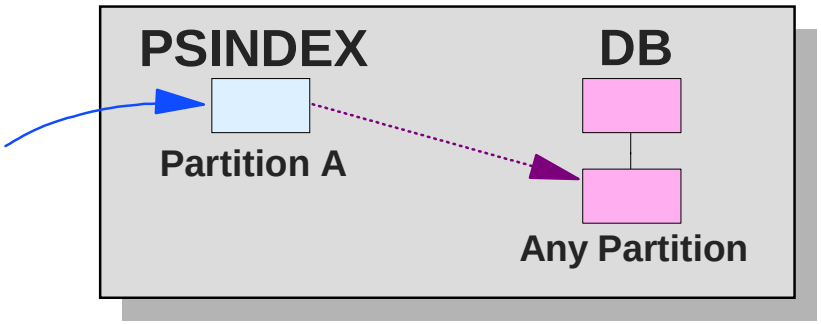
Using the HALDB Control Statement

- **Request for first segment in database, returns first segment in partition**
  - Example:
    - Unqualified GN call with no previous position
- **GN request which reaches end of the partition, returns 'GB' status code**
  - Example:
    - Unqualified GN call with position on the last database record in the partition
- **Request for segment in another partition returns 'FM' status code**
  - Example:
    - Restriction is to partition with keys 1,000,000 to 1,999,999
    - GU call qualified with 'Root Key = 2,500,000'

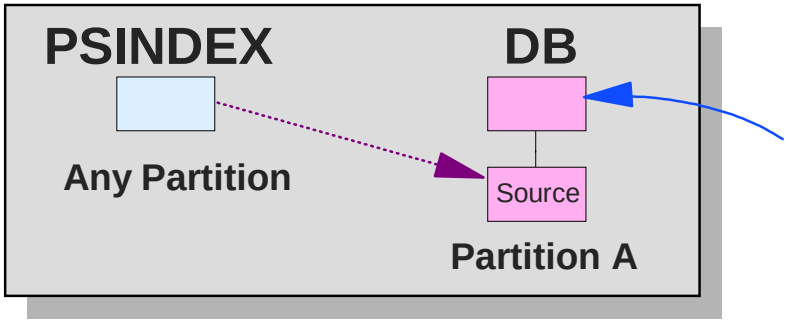
Using the HALDB Control Statement

Secondary Index usage

- When PCB specifies PROCSEQ,
  - PSINDEX partition is specified in the control statement
  - Limitation is to the PSINDEX partition, not to an indexed database partition



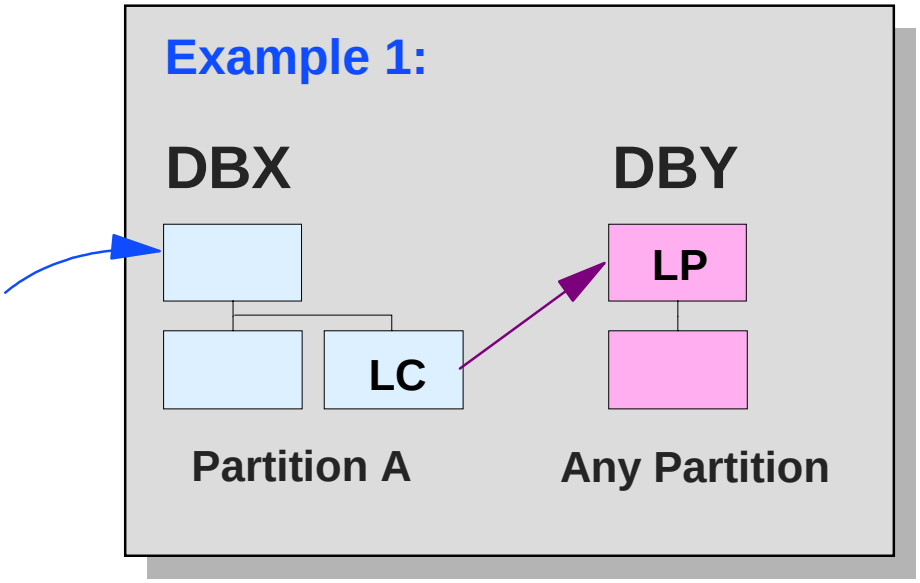
- When PCB does not specify PROCSEQ,
  - Database (not sec. index) partition is specified in the control statement
  - Updates to source segments may cause updates to any PSINDEX partition

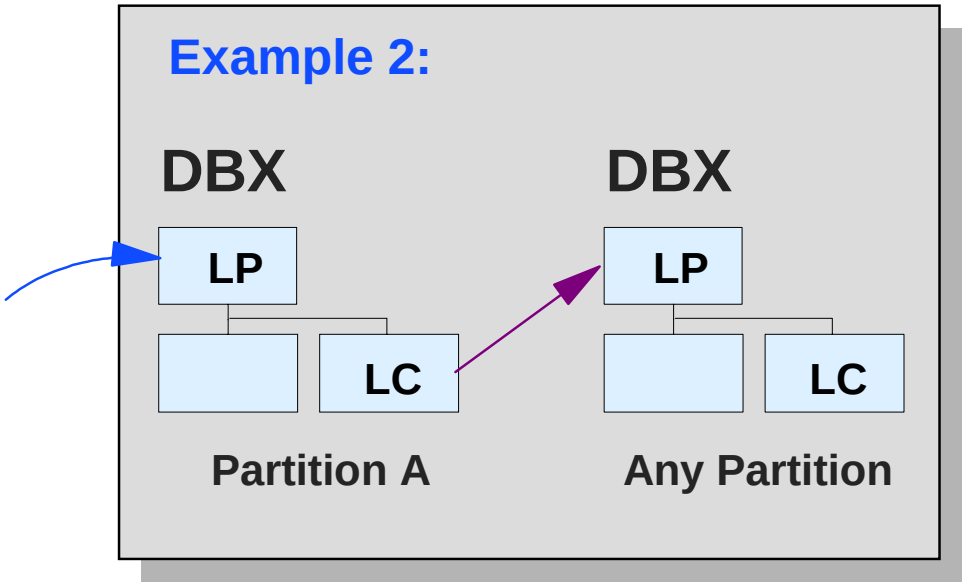


Using the HALDB Control Statement

Logical relationship usage

- Restricts access to partition in which LC resides
- Does not restrict access to partitions in which LPs reside





Using the HALDB Control Statement

- **HALDB control statement may be used with any batch job**
  - DBB, DLI, BMP, or JBP
- **HALDB control statements may be used for multiple PCBs**
  - One statement per HALDB PCB
- **Sequential processing**
  - May not require any application program changes
  - Could require processing to consolidate information, such as reports, from multiple executions
- **Random or skip sequential**
  - Probably requires application program changes to handle 'FM' status code

Handling Unavailable Partitions

Unavailable Partitions

Causes of unavailable partitions

- Partition stopped in online system
  - Typically, /DBR or /STOP command has been issued for the partition
- Partition authorized to another system without data sharing
  - Typically, batch job is processing the partition
- Partition has flag set in DBRC RECONs
  - Typically, partition needs to be recovered or image copied

Operational Options

Operate as we do with non-HALDB databases

- Do not /DBR or /STOP partitions
  - Issue these commands only for the database
- Do not attempt concurrent access to different partitions from different online systems and batch jobs without data sharing
- If any database data set is unavailable, stop all access to database

➢ Do not have to handle unavailable partitions

- Use HALDB for:
  - Large databases
  - Shortened database maintenance windows

Operational Options

Operate differently with HALDB databases

- Process partitions in parallel by different systems without data sharing
- /DBR or /STOP partitions
- If any database data set is unavailable, keep other partitions available

Add INIT STATUSGROUP calls

- Add to programs to avoid U3303 ABENDs
  - Attempts to access unavailable partitions cause U3303 unless this call is issued

Code for 'BA' status code

- Must be able to react to different reasons for 'BA'
  - Attempts to access an unavailable partition result in 'BA' status code
  - Other conditions may cause 'BA'
- React to unavailable partitions, databases, or records

DB PCB Status Code Priming

When program is scheduled,

- DB PCB status code indicates status of database
  - 'blank blank' - database is available
  - 'NA' - database is not available
  - 'NU' - database is not available for update

INIT DBQUERY call

- Restores the DB PCB status code fields to settings at schedule time
  - Same meanings for 'blank blank', 'NA', and 'NU'

No indication of status of partitions

- Database may be available, but some or all partitions may not be available
- Application program does not know which partitions are available

Reasons for 'BA' Status Code

Unavailable partition or database

- Stopped due to /DBR, /STOP, or /LOCK command
- Not available for update due to /DBD or access intent of RD or RO
- DBRC authorization failed
  - Authorized to another subsystem which is incompatible with this subsystem or
  - Flag set in RECONs (Prohibit further authorization, IC Needed, ...)

Unavailable record

- Lock held by failed IMS subsystem
- Lock held by in-doubt UOW in failed commit manager
  - CICS, DB2 Stored Procedure, other ODBA connector

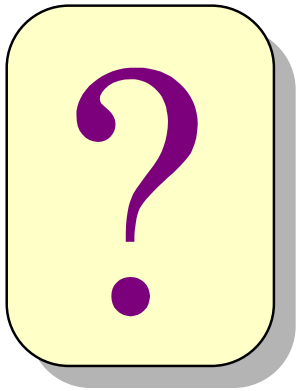
Unavailable Data Conditions

How can we know which unavailable data condition exists?

- Partition Not Available

  or

- Record Not Available



If we get a 'BA' status code, what should we do?

Assume an entire partition is not available?

Assume one record is not available?

Handling a 'BA' Status Code

Unqualified GN call after 'BA' for unavailable partition

- Retrieves first record in next available partition or gets 'GB' status code
  - Indicates that a partition was unavailable

Unqualified GN call after 'BA' for unavailable record

- Attempts to access same unavailable record
- Receives 'BA' status code
  - Indicates that a record was unavailable

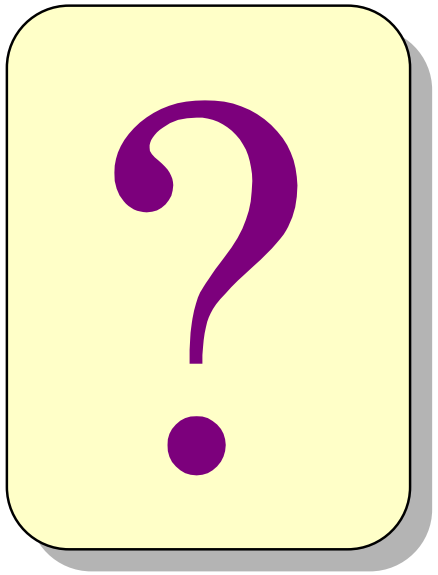
Potential problem:

- If first call gets 'BA' due to unavailable partition and
- First record in next available partition is unavailable (lock reject)
- Second call (unqualified GN) will get 'BA'

Programming for 'BA'

What should your application do when a partition is unavailable?

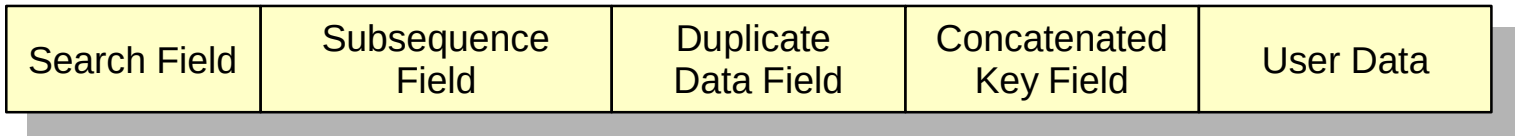
- Inform the terminal operator?
  - Is there a terminal operator?
- Quit?
  - How do you restart the process at the right time?
- Skip this partition?
  - What effect does this have on the application?
- Cause this partition to be processed later?
  - How do you implement this?



Processing Secondary Indexes as Databases

Secondary Indexes

Fields in a secondary index segment:



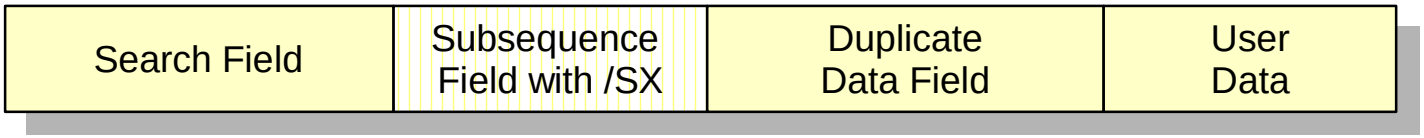
- Search field - key of the secondary index segment
- Subsequence field - optional fields to make non-unique keys unique
  - Copied from source segment
    or
  - System-related unique key created by use of '/SX...' field name
  - Concatenated key of source segment created by use of /CK...' field name
- Duplicate data - optional fields copied from source segment
- Concatenated key field - optional field used for symbolic pointing with non-HALDB
- User data - optional fields maintained by user, not IMS
  - Rarely used

Processing a Secondary Index as a Database

Subsequence field size increased when using /SX

- /SX field increased from 4 to 8 bytes for HALDB
  - KEYLEN in PCB must be increased by 4 bytes
  - Duplicate data fields and user data are offset by 4 bytes
    - IO-area must be adjusted

Program I/O Area for Non-HALDB Secondary Index Segment



Program I/O Area for HALDB Secondary Index Segment

Are You Affected by /SX?

- If you have PSB which specifies DBDNAME as a secondary index name:

```
PCB   TYPE=DB,DBDNAME='sec. index name',KEYLEN=n
```

- Find LCHILD in the DBD for this secondary index:

```
LCHILD NAME=(segmentname,databasename),INDEX=xdfldname
```

- Match INDEX value from secondary index LCHILD with NAME value from XDFLD statement in indexed database:

```
XDFLD NAME=xdfldname,SRCH=list,SUBSEQ=/SXzzzzz,DDATA=list
```

- If "/SX" appears in SUBSEQ value, KEYLEN must be increased in PSB and size of IO area must be increased
- If DDATA is also specified, reference to dup. data fields must be adjusted

Processing a Secondary Index as a Database

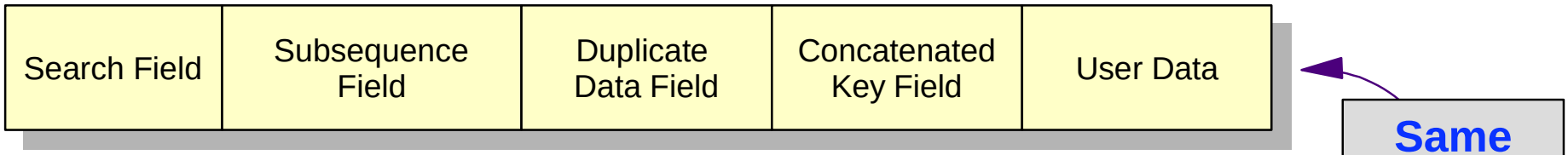
Symbolic pointing not used with HALDB

- Concatenated key field not present
  - Application may not react correctly

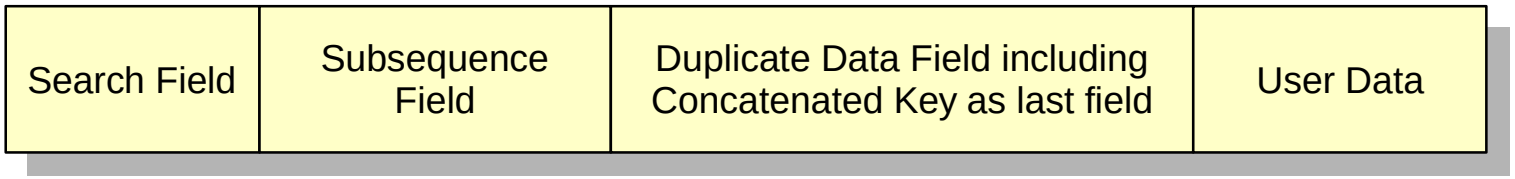
Solution:

- Concatenated key may be retained as duplicate data field
  - No changes required to application programs

Program I/O Area for Non-HALDB Sec. Index Segment using Symbolic Pointing



Program I/O Area for HALDB Sec. Index Segment with Concat. Key as Duplicate Data



Are You Affected by Symbolic Pointing?

- If you have PSB which specifies DBDNAME as a secondary index name:

```
PCB   TYPE=DB,DBDNAME='sec. index name' ,KEYLEN=n
```

- Find LCHILD in the DBD for this secondary index:

```
LCHILD NAME=(segmentname,databasename),PTR=SYMB
```

- If "PTR=SYMB" appears on LCHILD, you have symbolic pointing

- Keep concatenated key in I/O area by adding "/CKxxxxx" field to DDATA

```
XDFLD NAME=xdfldname,SRCH=list,DDATA=(list,/CKxxxxx)
```

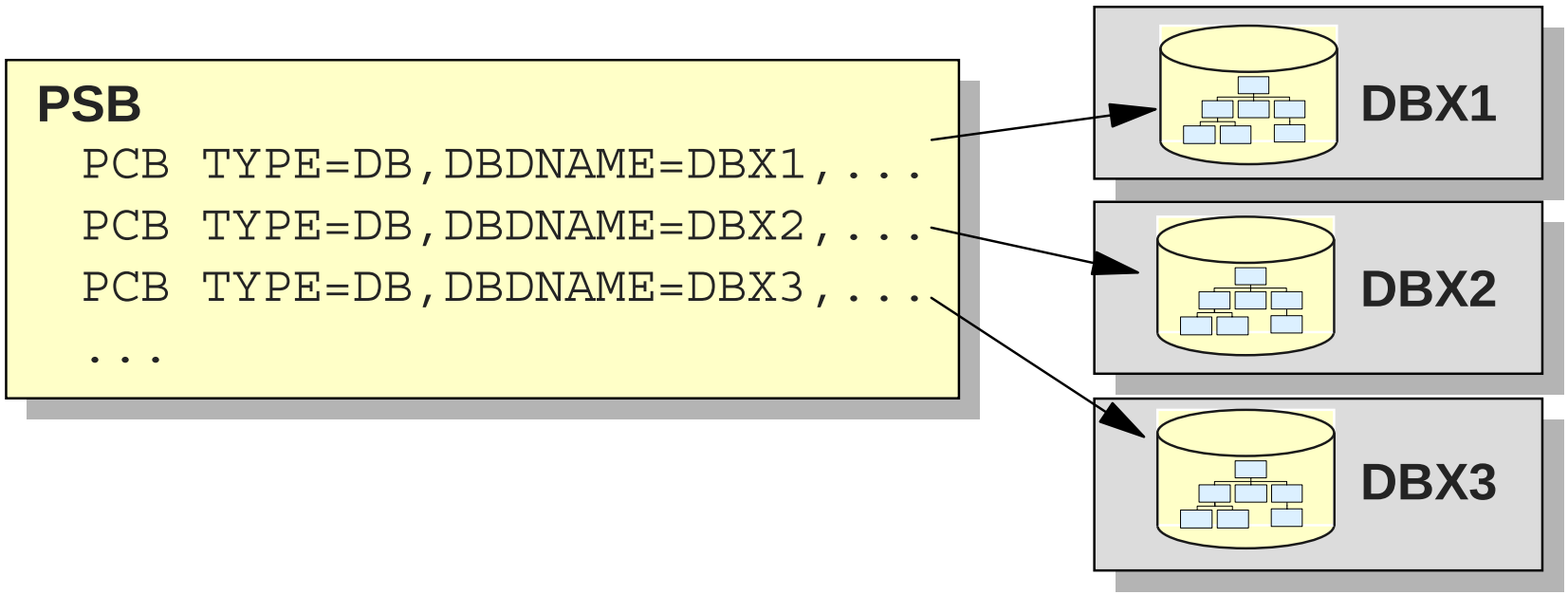
  - /CKxxxxx field must also be defined with FIELD statement in indexed database

Converting from User Partitioning

User Partitioning

Some installations have done their own partitioning

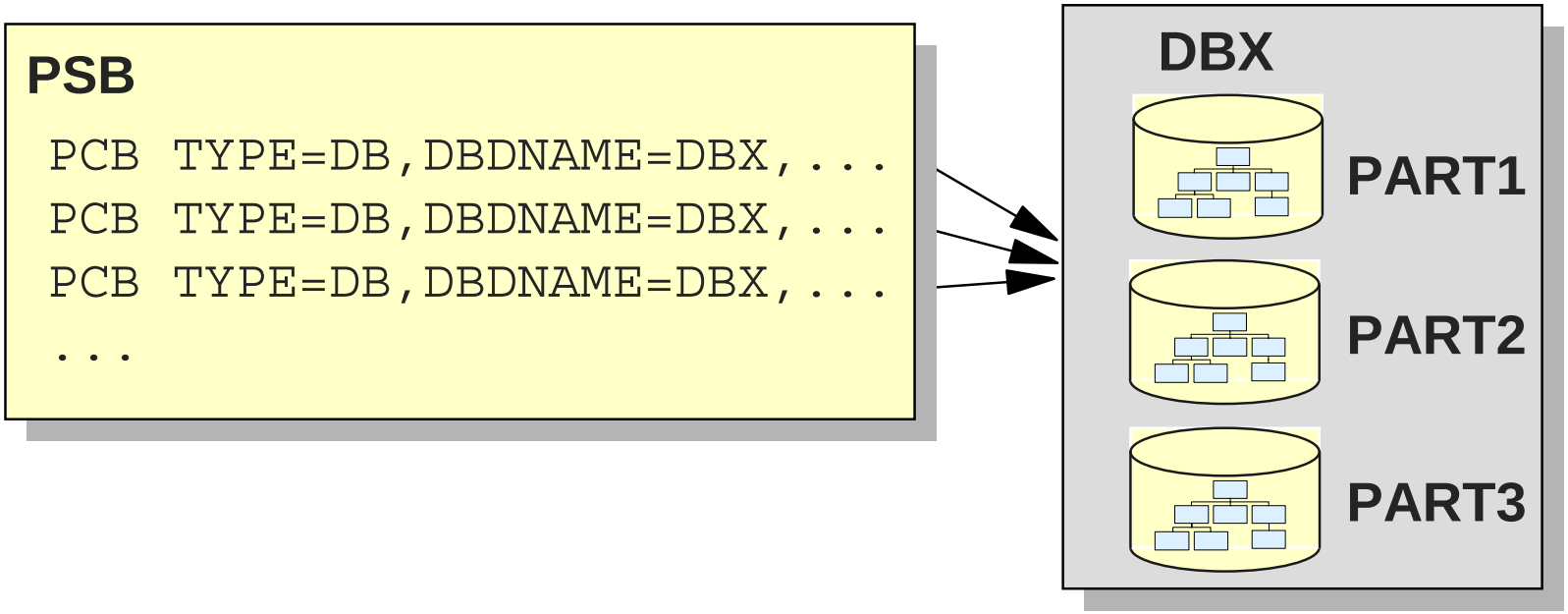
- Database split into multiple databases
- Application selects which database to use
  - Based on key of root segment
  - Database selection may be done by subroutine or modification to language interface module
  - PSB has PCB for each database



User Partitioning

Converting from user partitioning

- Multiple databases become one HALDB database with multiple partitions
- Application does not need to select which database to use
  - Could use one PCB - would require application changes
- Alternative:
  - PSB is changed so that all PCBs reference the same HALDB database
  - Application program continues to select PCBs - no application changes



Summary

Partitioning Options

- Key range vs. partition selection exit routine

Initial Loads

- Current programs work except for loading logical children
- Can load partitions in parallel

Processing Partitions in Parallel

- Use HALDB control statement to limit a PCB to a partition

Handling Unavailable Partitions

- Understand 'BA' status code use

Processing Secondary Indexes as Databases

- May require changes to application programs, especially with duplicate data and /SX field

Converting from User Partitioning

- PSB change will probably be sufficient

RETURN TO INDEX