A16

Application Design and Programming with HALDB

Rich Lewis

Miami Beach, FL October 22-25, 2001

RETURN TO INDEX

The world depends on it

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.

The world depends on it

Agenda

- Highlights of HALDB
- Overview of Partitioning
  - Partitioning Choices
  - Order of segments in a database and its partitions
- Initial Loads
- Processing Partitions in Parallel
- Handling Unavailable Partitions
- Processing Secondary Indexes as Databases
- Converting from User Partitioning

HALDB (High Availability Large Database)

Large Database

Up to 10,010 data sets per database!

Greater than 40 terabytes

- Databases are partitioned
  - 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
  - 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

The world depends on it

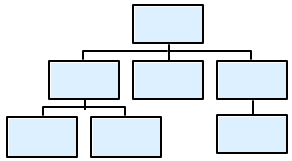
Highlights

New database types

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

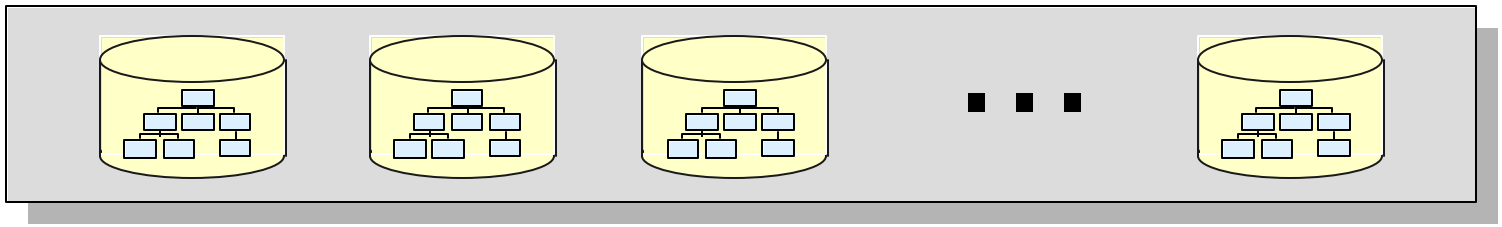
Hierarchic structure is maintained

- A database record resides in one partition



Partition selection (deciding in which partition a record resides)

- By key range or by user exit routine



The world depends on it

Highlights

Logical relationships and secondary indexes are supported

- Secondary indexes may be partitioned

Parallel Processing

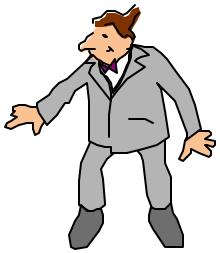
- Reorganizations
  - Partitions may be reorganized independently
  - Partitions may be reorganized in parallel
- Application processing
  - Partitions may be processed in parallel
  - DBRC authorization is by partition (not entire database)

The Application Programming News

The world depends on it

The good news:

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

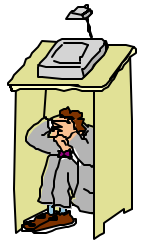


The OK news:

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

The bad news:

- You may have to change some application programs when databases are migrated to HALDB

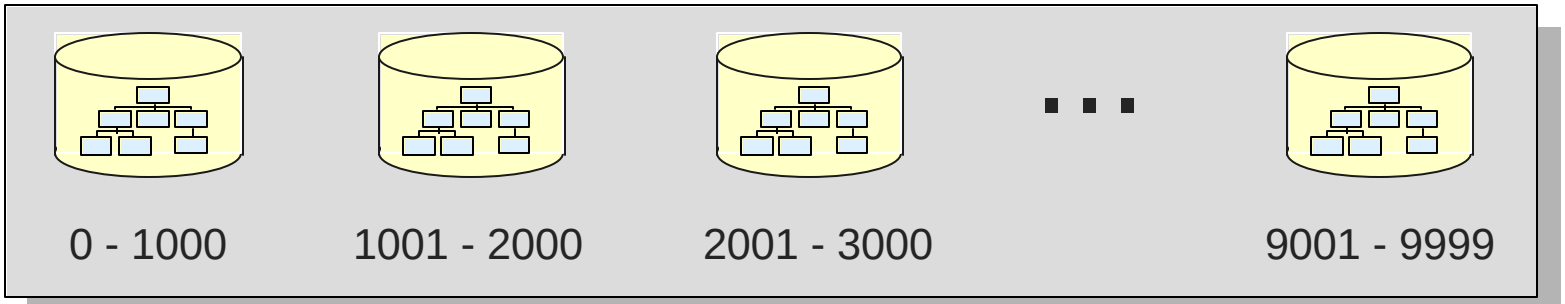


The world depends on it

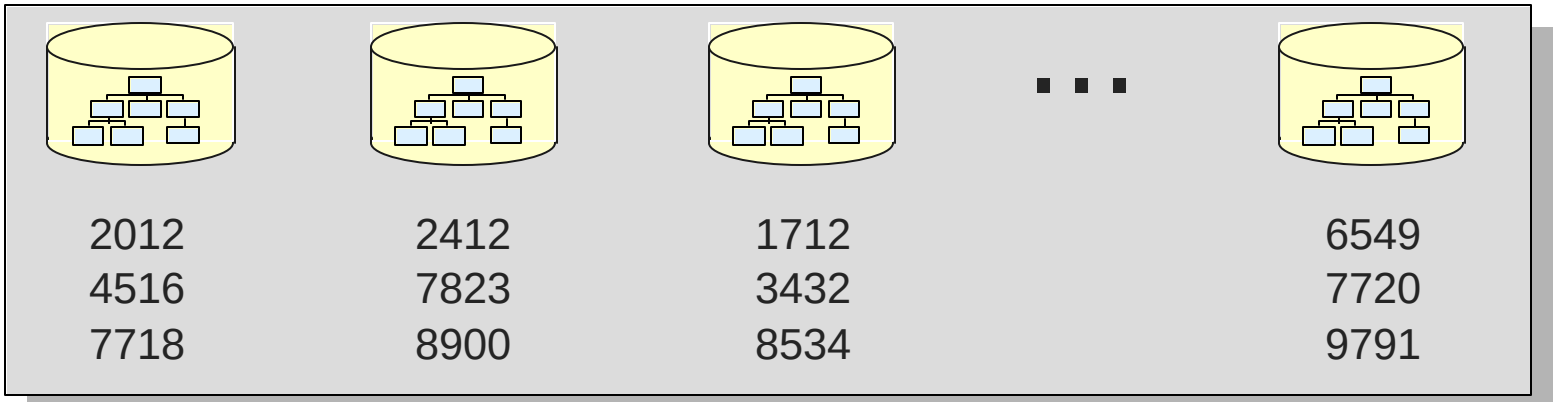
Partitioning Choices

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

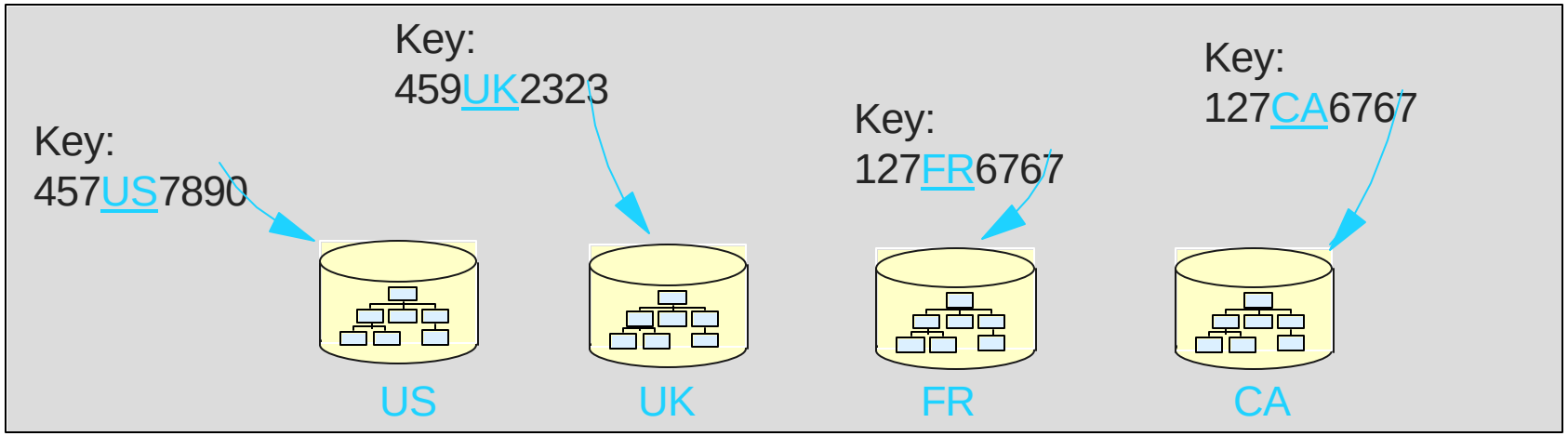


The world depends on it

Partitioning Choices

Partition Selection Exit routine example:

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



The world depends on it

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

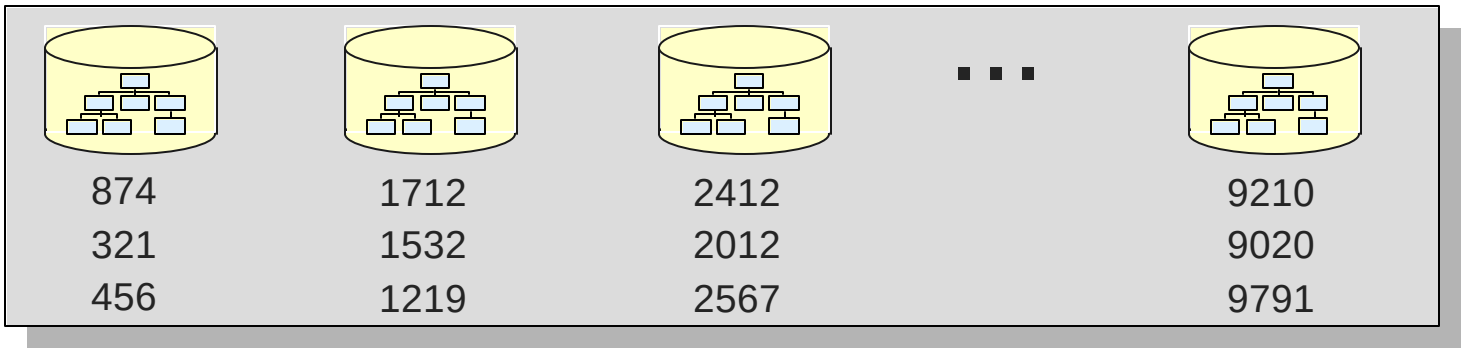
PSINDEX - Partitioned Secondary Index

- Entries are in sequential order within a partition

Order of Segments in a PHDAM Database

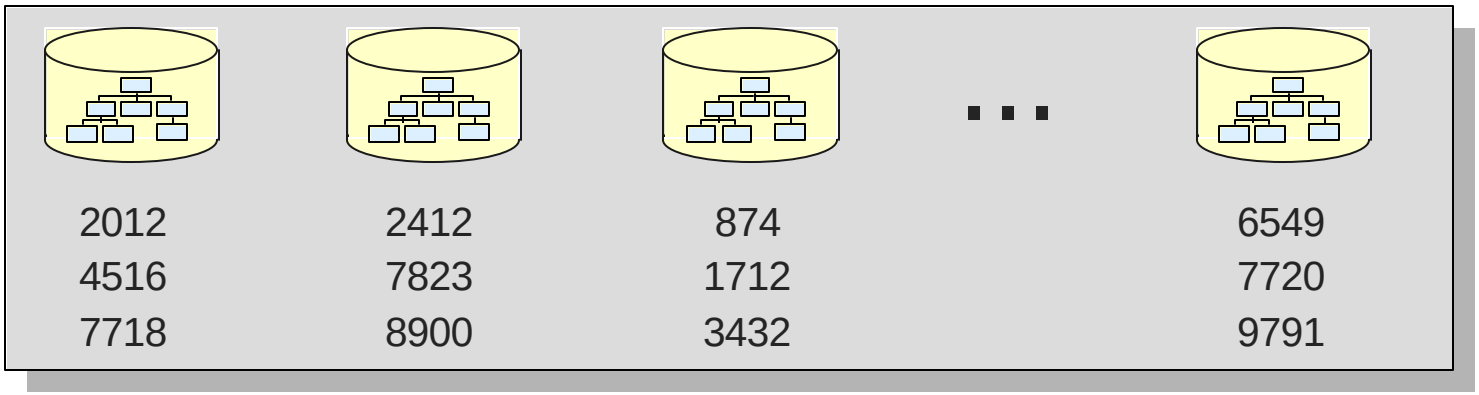
PHDAM with key range partitioning

- Keys are sequential between partitions, random within a partition



PHDAM with Partition Selection Exit routine

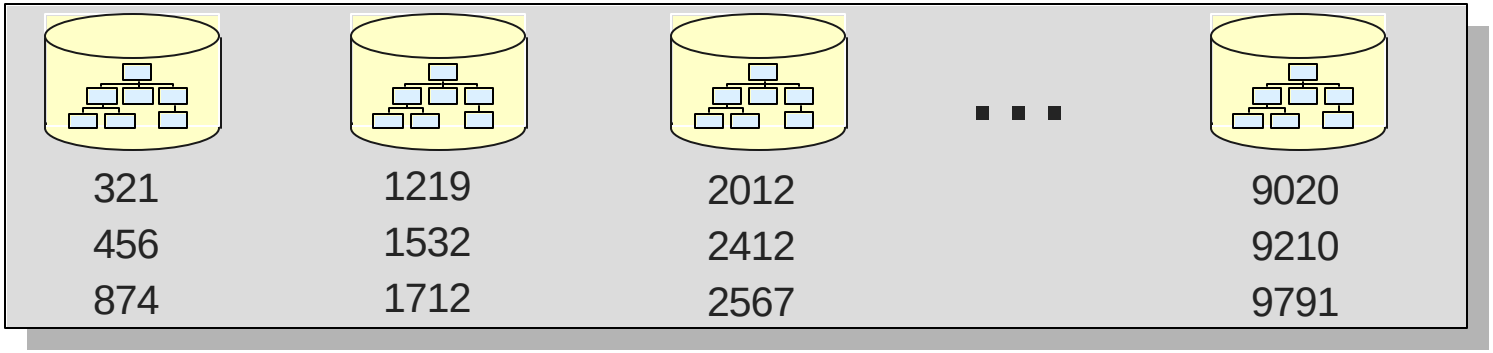
- Keys are not sequential between partitions, random within a partition



Order of Segments in a PHIDAM Database

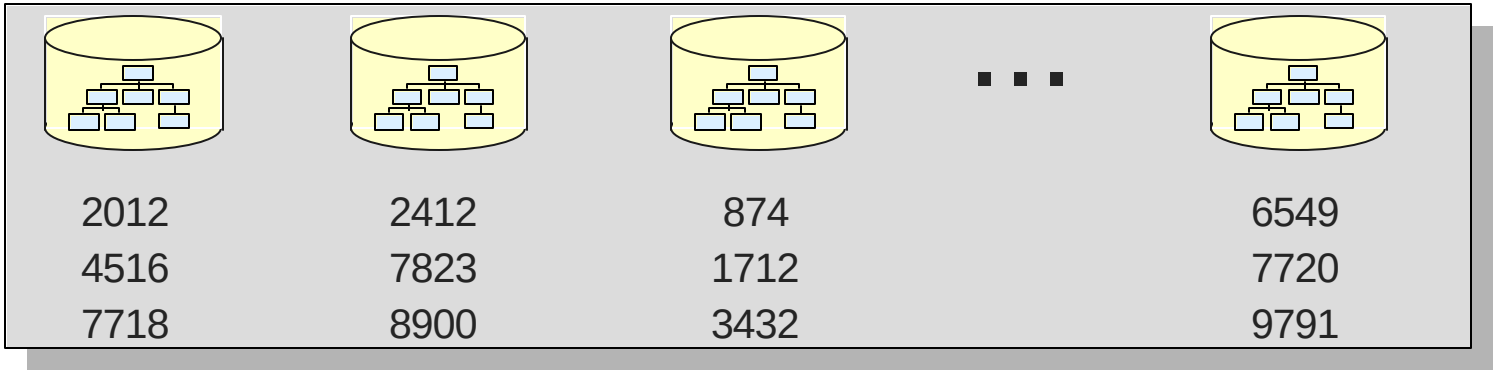
PHIDAM with key range partitioning

- Keys are sequential between partitions, sequential within a partition



PHIDAM with Partition Selection Exit routine

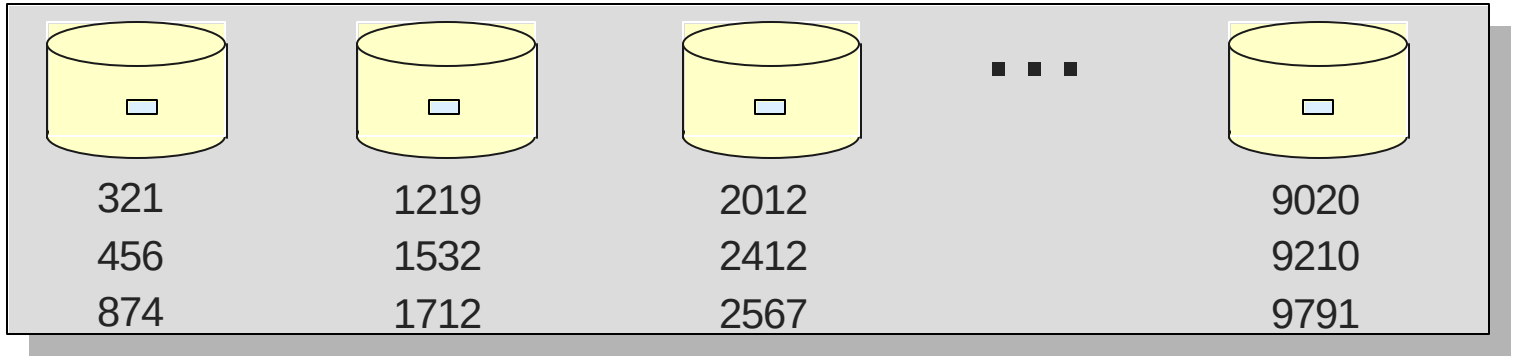
- Keys are not sequential between partitions, sequential within a partition



Order of Segments in a PSINDEX Database

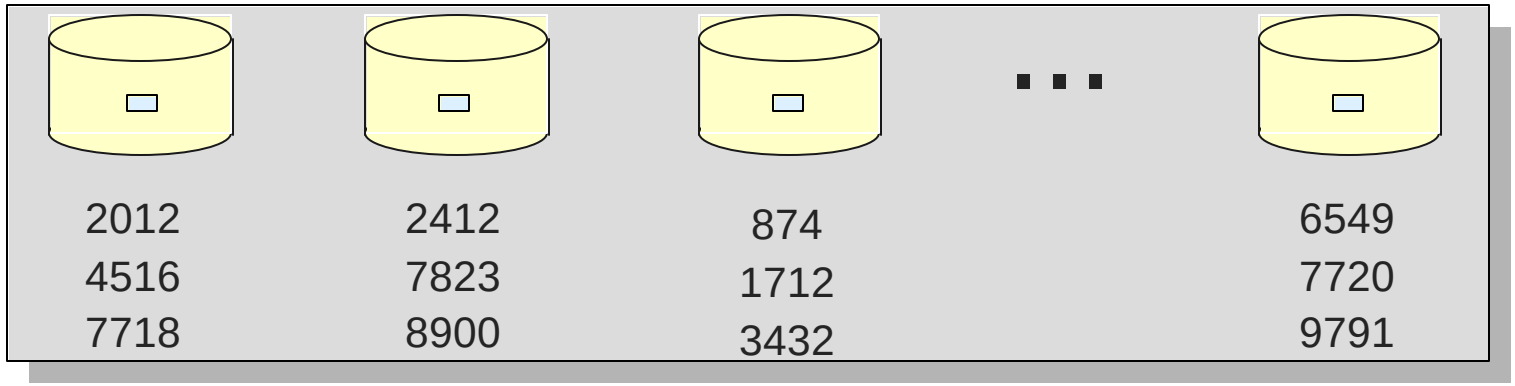
PSINDEX with key range partitioning

- 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



The world depends on it

Application Considerations

- **Initial loads**
  - Order of records in the load
  - Loading partitions in parallel
  - Logical children may not be loaded
- **Processing partitions in parallel**
- **Handling unavailable partitions**
- **Processing secondary indexes as databases**
- **Converting from 'user partitioning'**

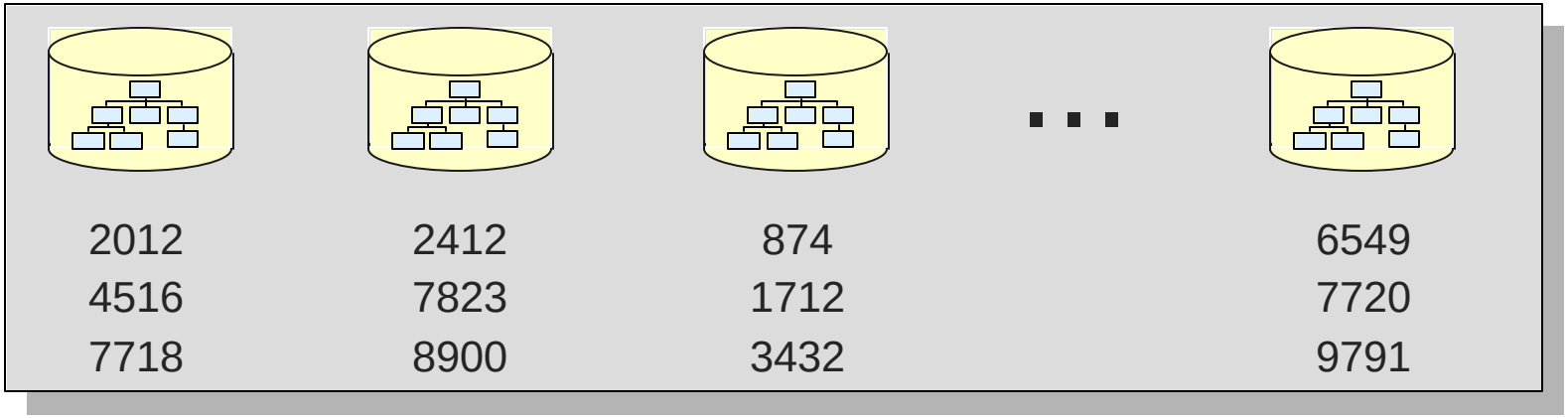
Initial Loads

Initial Loads

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

- Same as non-HALDB databases

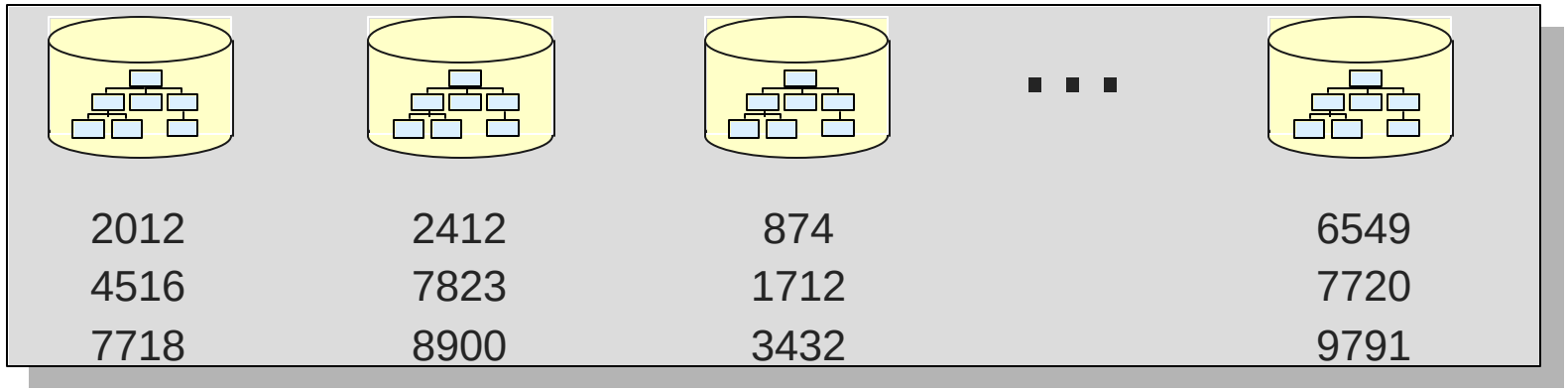
Loads of PHDAM roots may be in any key sequence



- 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

Initial Loads

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



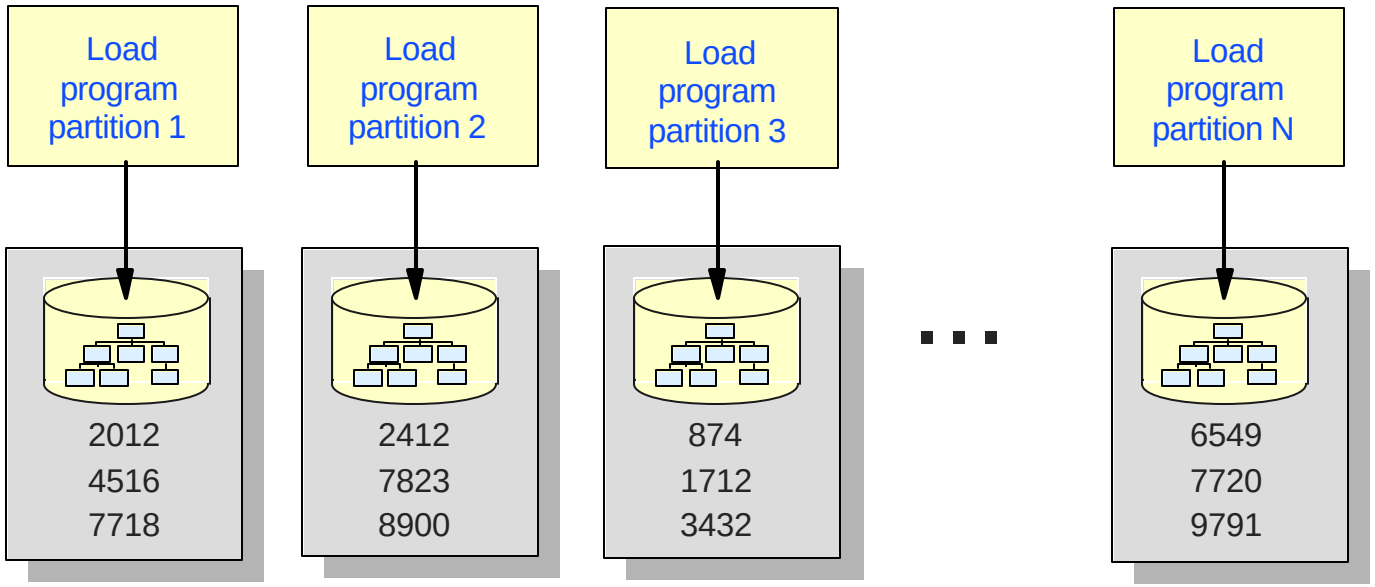
  - Could load 874, 1712, 2012, 2412, 4516, 6549, 7718, 7720, ...
    - This is like HIDAM database order
    - No changes required for load program
  - Could load 2012, 4516, 7718, 2412, 7823, 8900, 874, 1712, ...

The world depends on it

Initial Loads in Parallel

Partitions may be initially loaded in parallel

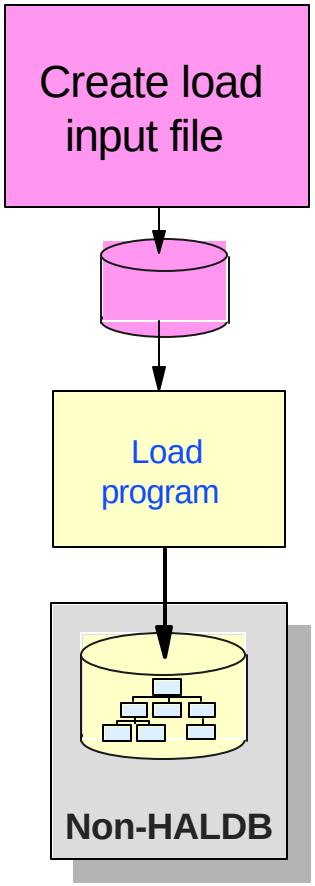
- Separate jobs for each partition
  - Multiple jobs cannot load records in the same partition
- Could make load of database much faster
- Current load programs may work without change
  - Input to program would have to be split by partition



Initial Loads in Parallel

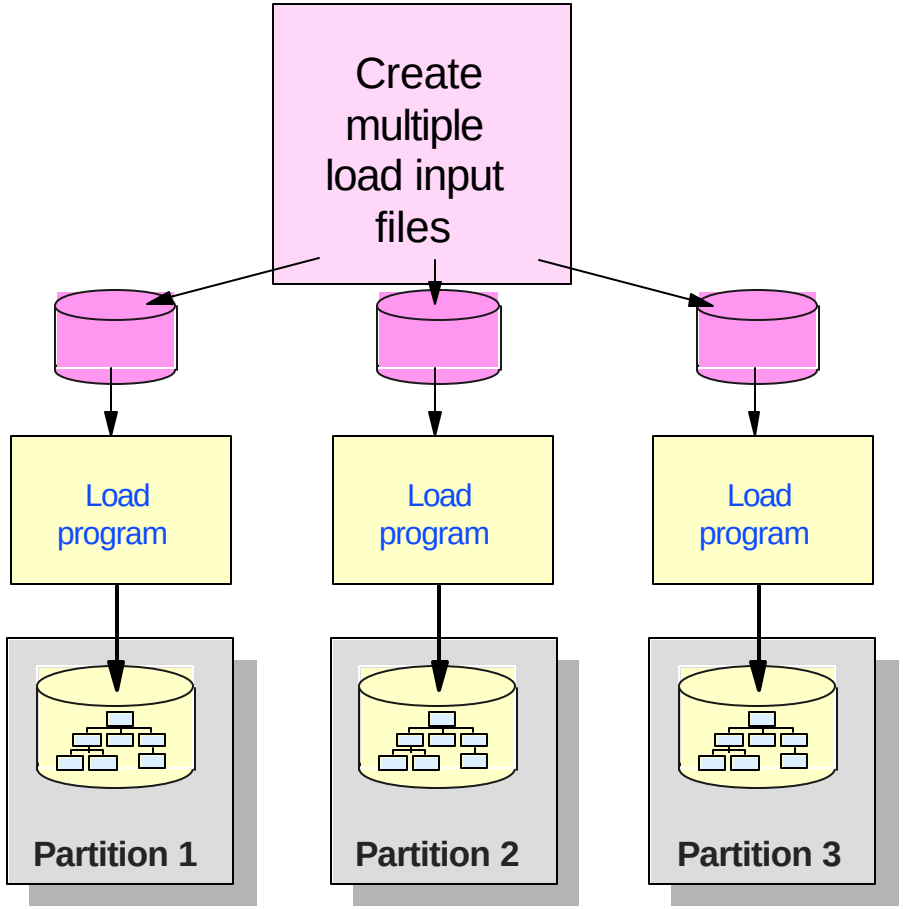
Non-HALDB process

- Step 1 creates file read by load program



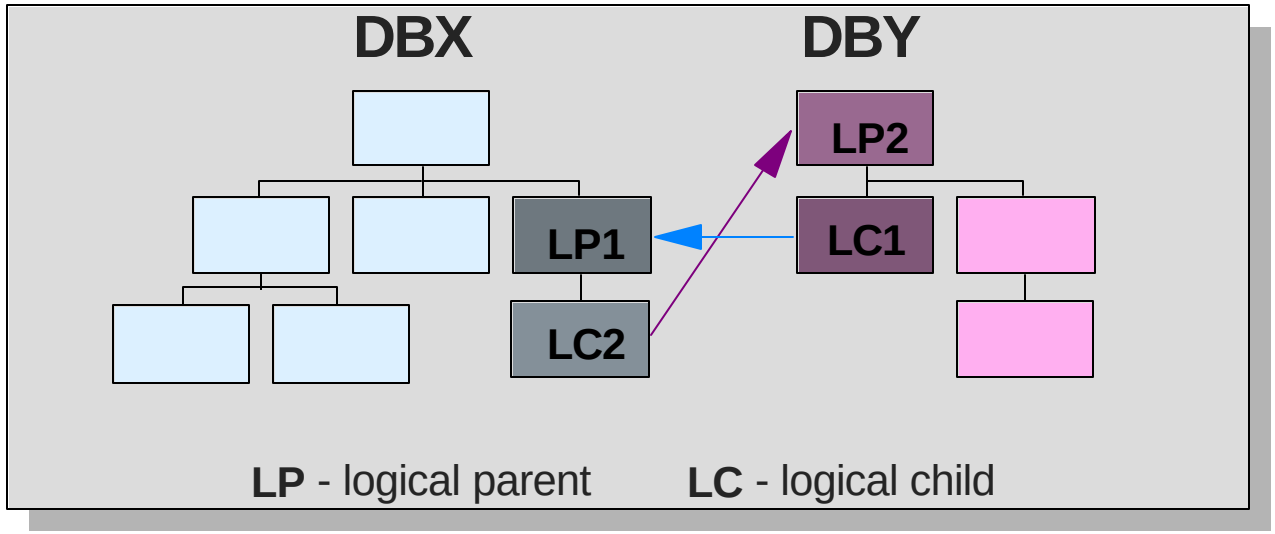
HALDB parallel load process

- 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

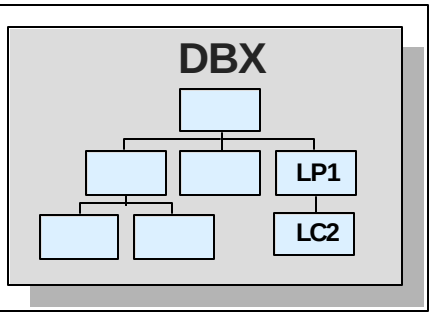
- Attempts to load logical children receive 'LF' status code
- They must be added by update programs (PROCOPT=I or A)
  - Cannot insert logical child without logical parent

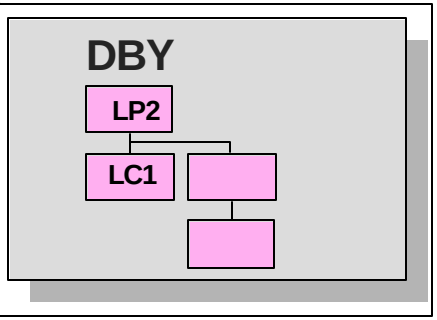
Initial Loads with Logical Relationships

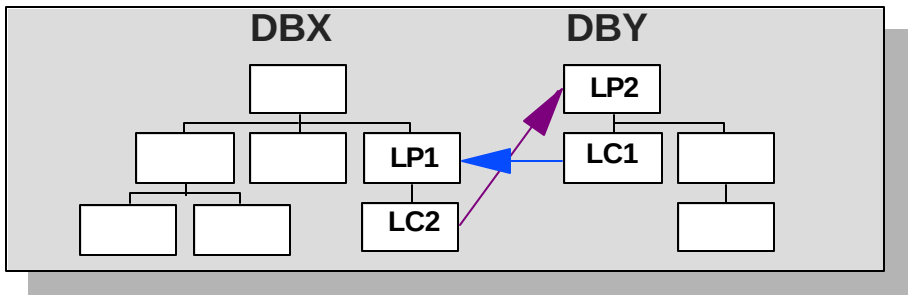
The world depends on it

Non-HALDB process:

- Step 1: Prereorganization utility (optional)
- 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



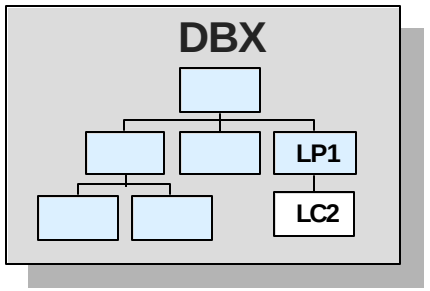




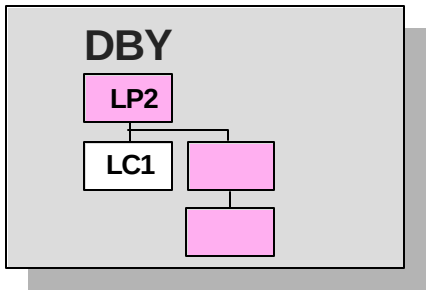
Initial Loads with Logical Relationships

Initial load programs for HALDB with logical children must be split

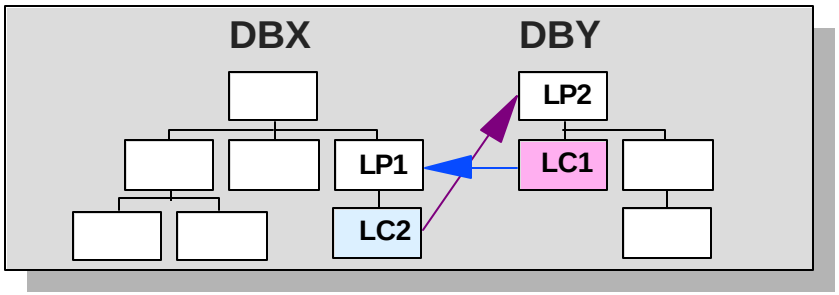
- Step1 : Initial load (PROCOPT=L or LS) DBX without logical children
  - This is a modification to the existing load program



- Step 2 : Initial load (PROCOPT=L or LS) DBY without logical children
  - This is a modification to the existing load program



- Step 3: 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

The world depends on it

Processing Partitions in Parallel

- **Online, batch, and data sharing considerations**
- **Random and skip sequential processing**
- **Sequential processing**
  - Finding the first record in a partition
  - Finding the last record in a partition

The world depends on it

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

The world depends on it

Two Styles of Batch Programs

Random and Skip Sequential

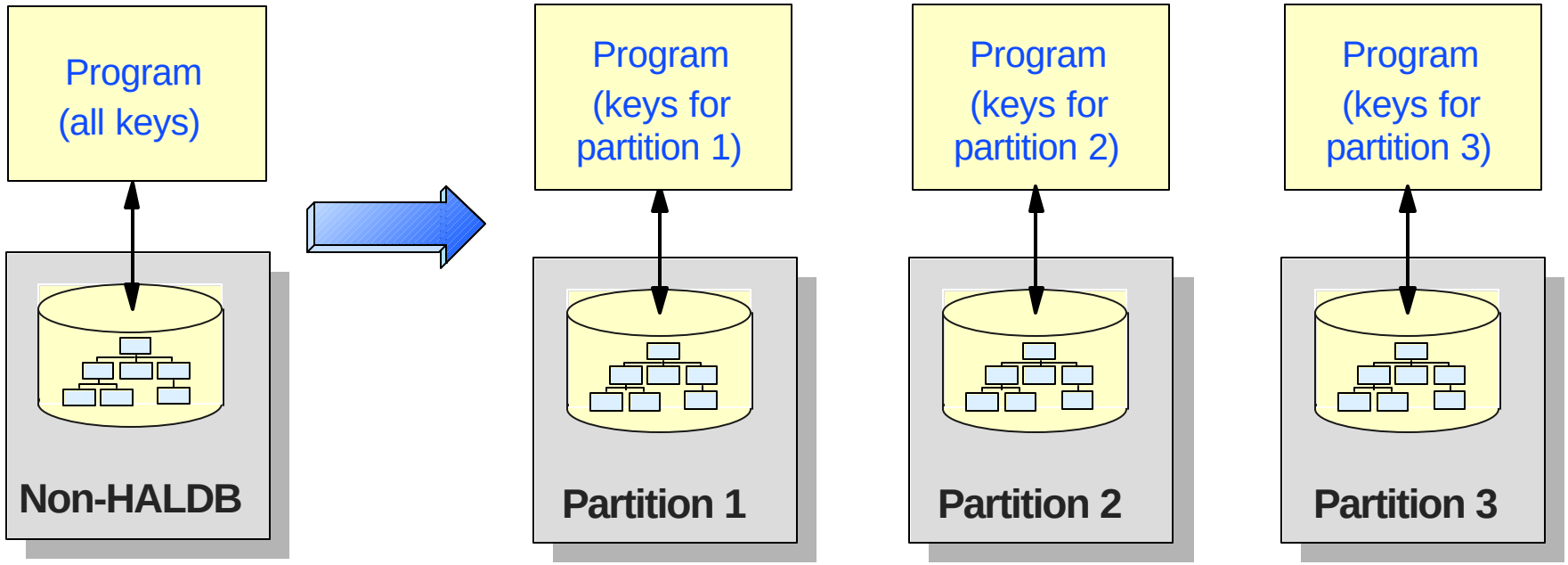
- Records are accessed by key
- Get calls are qualified on keys
  - Keys are known before calls are made
  - Program accesses only some of the records in the database
  - Program can determine the partition before the call is made

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



The world depends on it

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)

Program must be modified to restrict accesses to one partition

- Must start with first record in partition
- Must recognize the end of the partition

The world depends on it

Finding First Record in Partition

PHIDAM with key range partitioning

- GU (root key = lowest key in partition)
  - Lowest possible key is high key for previous partition + 1

PHIDAM with Partition Selection Exit routine

- GU (root key = lowest key in partition)
  - Lowest possible key is determined by exit routine decision

PHDAM with key range partitioning

- GU (root key = lowest key that randomizes to first RAP in partition)
  - Requires a special randomization routine*

PHDAM with Partition Selection Exit routine

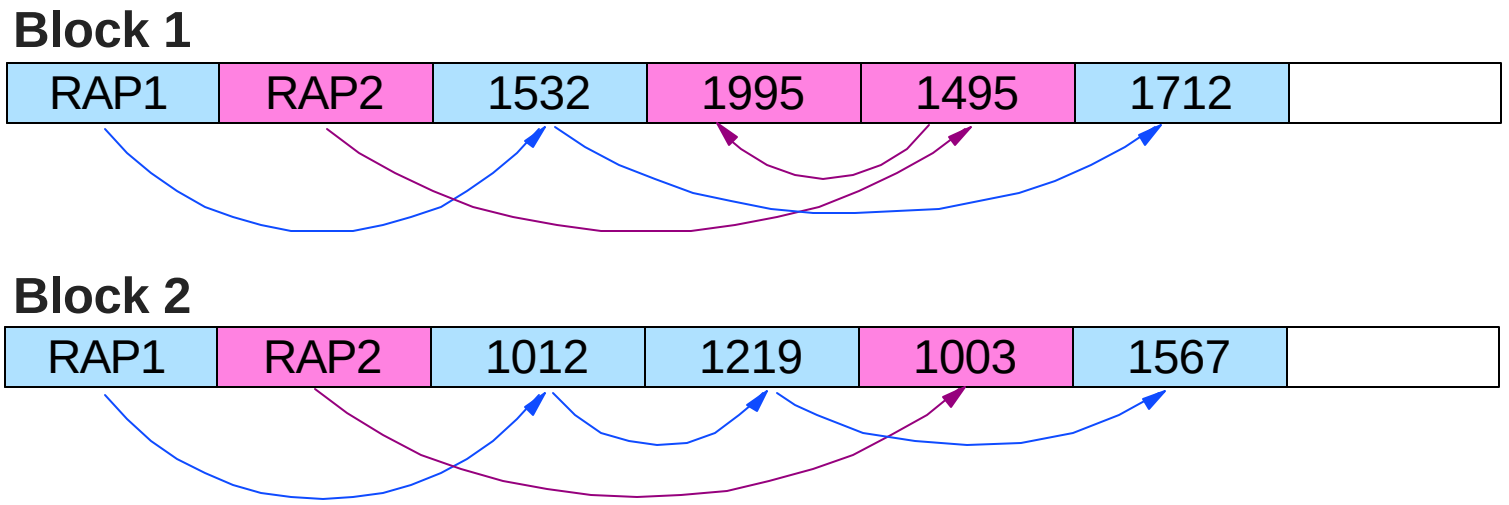
- GU (root key = lowest key that randomizes to first RAP in partition)
  - Requires a special randomization routine*

* Seq. Subset Randomizer function of IBM IMS HP Unload may be used to create rand. routines

The world depends on it

Finding First Record in Partition - PHDAM

- PHDAM randomization example for partition with keys 1000 - 1999

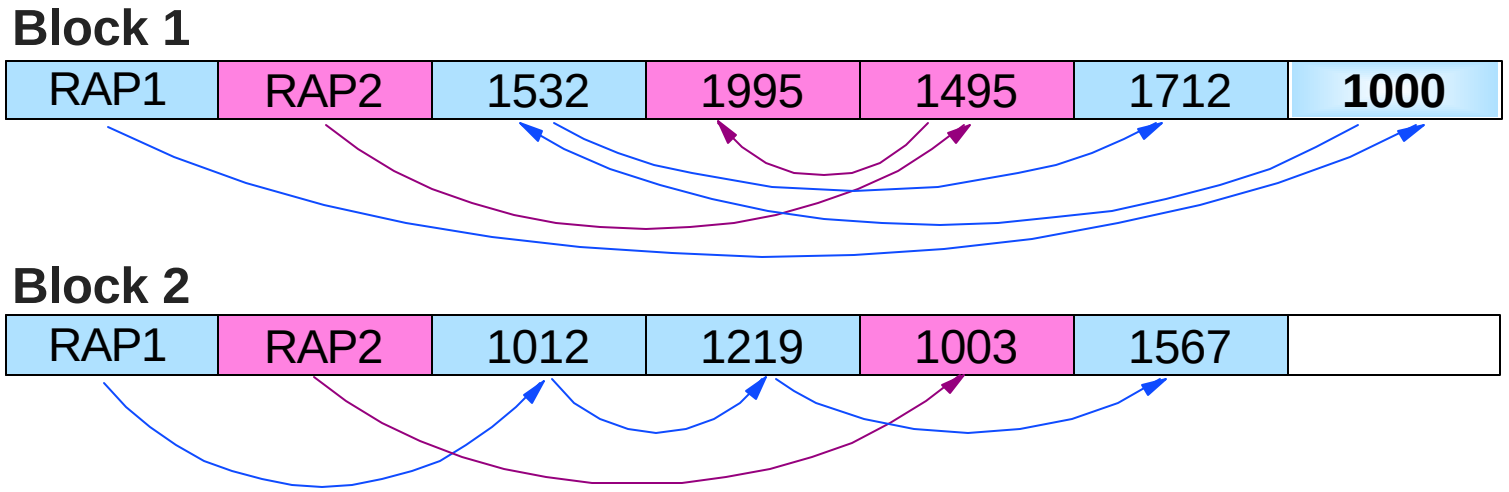


- Application must begin at beginning of chain from RAP1 in Block 1
  - Keys are in sequence on chain from a RAP
  - Keys are not in sequence between RAPs or blocks
  - Application program must know to begin with key=1532
    - Or key=n where n randomizes to RAP1 in Block 1 and is lower than 1532

Finding First Record in Partition - PHDAM

Solution for PHDAM randomization example:

- Randomize the lowest possible key in partition to first RAP
  - Application program must know to begin with key=1000
  - Key=1000 does not have to exist



- Unique randomization routine for each partition

  or

- Routine would have to recognize lowest key in each partition

➢ Randomization routines are not aware of the partitions for which they are called

The world depends on it

Recognizing Last Record in Partition

PHIDAM

- If highest possible key for partition is present,
  - End of partition when it is returned by a call
- If highest possible key for partition does not exist,
  - End of partition when key from next partition, 'GB', or 'BA' is returned
    or
  - 3303 ABEND

PHDAM

- When highest possible key chained from last RAP is present,
  - End of partition when it is returned by a call
- If highest possible key chained from last RAP does not exist,
  - End of partition when key from next partition, 'GB', or 'BA' is returned
    or
  - 3303 ABEND

The world depends on it

If Last Possible Key Does Not Exist

- **Key from next partition returned**
  - Indicates program has crossed to next partition
- **'GB' status code returned**
  - Indicates end of database
- **'BA' status code returned**
  - Indicates unavailable data condition
    - Only returned when program has issued INIT STATUSGROUP call
- **3303 ABEND**
  - Unavailable data condition exists
    - Only occurs when program has not issued INIT STATUSGROUP call

The world depends on it

Unavailable Data Conditions

Record Not Available

- Data sharing lock reject condition
  - Lock for database record or block is held by a failed subsystem
- Failed in-doubt unit-of-work lock reject condition
  - Typically, CICS failed with a transaction in-doubt
    - The transaction holds a lock on an IMS database record or block
  - Could be similar condition for DB2 stored procedure or other ODBA user

Partition or Database Not Available

- Stopped due to /DBR, /STOP, or /LOCK command for partition
- 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, ...)

The world depends on it

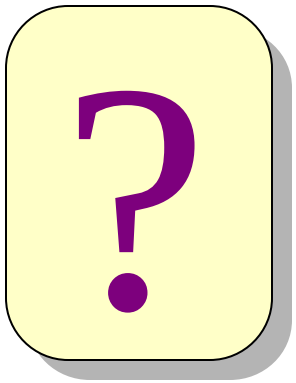
Unavailable Data Conditions

How can we know which unavailable data condition exists?

- Record Not Available

  or

- Partition 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?

The world depends on it

Programming for 'BA'

Making another call after receiving a 'BA' status code

- If record not available
  - Unqualified GN attempts to access same locked record
    - Receives 'BA' again
      - We know it's a lock reject due to unavailable record
- If unavailable partition
  - Unqualified GN accesses next available partition
    - Retrieves first record in next available partition
      - We know there's an unavailable partition

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'

The world depends on it

3303 ABEND

3303 ABEND

- Application cannot react
  - ABEND terminates the program
- Updates are backed out
  - BMP or online transaction
    - IMS backs out to last sync point
  - Batch job
    - Batch Backout must be run to back out to last sync point
      - Dynamic backout occurs if BKO=Y specified with DASD log

If access to unavailable partition might be attempted

- 3303 ABEND must be avoided by using INIT STATUSGROUP call

The world depends on it

Problems with Parallel Sequential Processing

Knowing first and last records in PHIDAM partition

- Requires knowledge of key ranges or partition selection exit logic

Knowing first and last records in PHDAM partition

- Requires knowledge of key ranges or partition selection exit logic
- Requires special randomizers
  - May be generated by Seq. Subset Randomizer function of IBM IMS HP Unload

Reaching end of partition

- Attempts to access next partition
  - May result in unavailable data condition ('BA' or 3303 ABEND)
- May require INIT STATUSGROUP call and special programming
  - 'BA' status code has multiple meanings (record or partition not available)

The Solution

The world depends on it

The Solution for Parallel Processing

- **Enhancement planned for IMS V7**
  - APAR number: ?
- **Control statement to limit access to one partition**
  - Batch (DLI or DBB)
  - IMS TM dependent region (BMP, IFP, or MPP)
  - CICS region

The world depends on it

The Solution for Parallel Processing

- **Unqualified GN call without position retrieves first record in partition**
  - Solves the PHIDAM and PHDAM first record problem
    - Does not require knowledge of key ranges or exit routine logic
    - Does not require a special randomizer for PHDAM
- **GN call which reaches end of partition returns 'GB' status code**
  - Solves recognizing the end of partition problem
- **Warning:**
  - Logical relationships may cause another partition or database to be accessed
  - Updates may cause a secondary index to be accessed (updated)

Handling Unavailable Partitions

The world depends on it

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

The world depends on it

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

The world depends on it

Operational Options

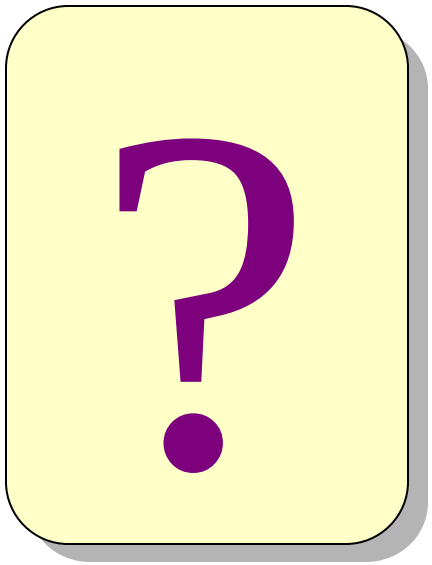
- **Operate differently with HALDB databases**
  - Process partitions in parallel by different systems
  - /DBR or /STOP partitions
  - If any database data set is unavailable, keep other partitions available
- **Add INIT STATUSGROUP calls**
  - Add to programs to avoid 3303 ABENDs
- **Code for 'BA' status code**
  - React to unavailable partitions, databases, or records

The world depends on it

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

The world depends on it

Secondary Indexes

Fields in a secondary index segment:

Search Field	Subsequence Field	Duplicate Data Field	Concatenated Key Field	User Data

- 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

Search Field	Subsequence Field with /SX	Duplicate Data Field	User Data

Program I/O Area for HALDB Secondary Index Segment

Search Field	Subsequence Field with /SX	Duplicate Data Field	User Data

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

The world depends on it

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

Search Field	Subsequence Field	Duplicate Data Field	Concatenated Key Field	User Data

Same

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

Search Field	Subsequence Field	Duplicate Data Field including Concatenated Key as last field	User Data

The world depends on it

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)
```

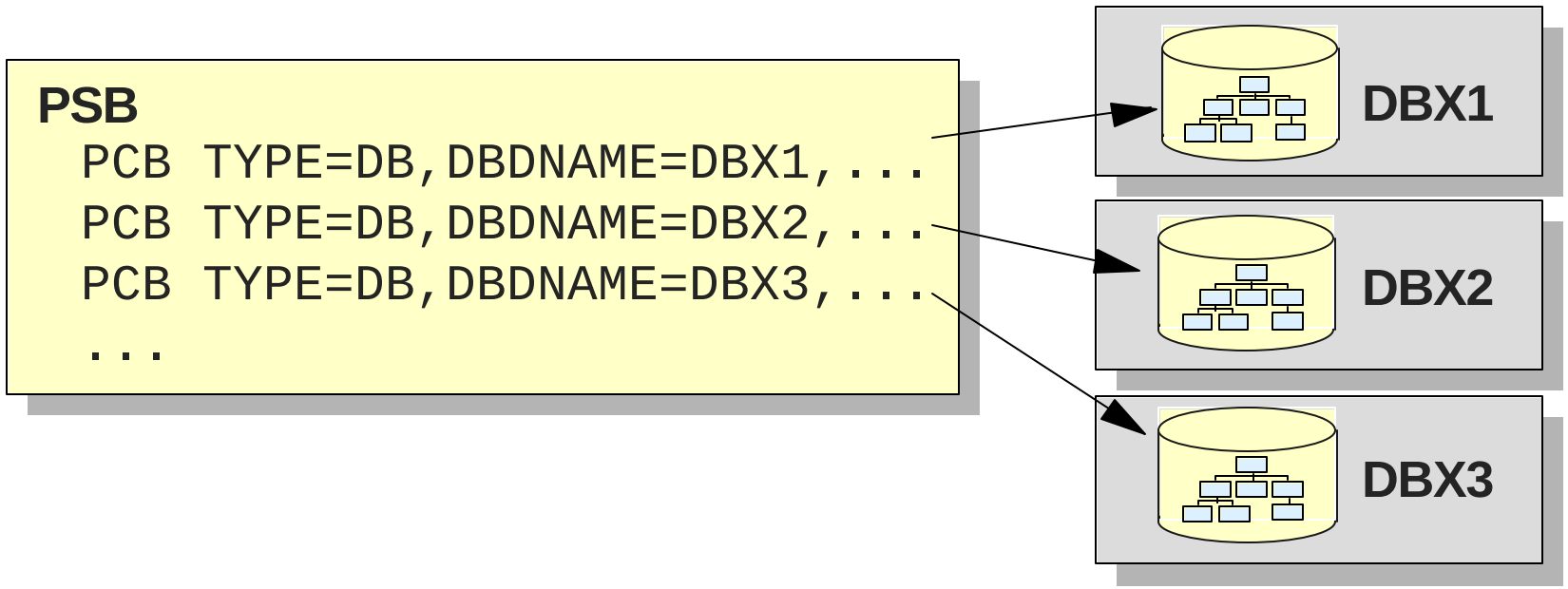
  - /CKxxxxx field must also be defined with FIELD statement in indexed database

Converting from User Partitioning

User Partitioning

Some installations have done their own partitioning

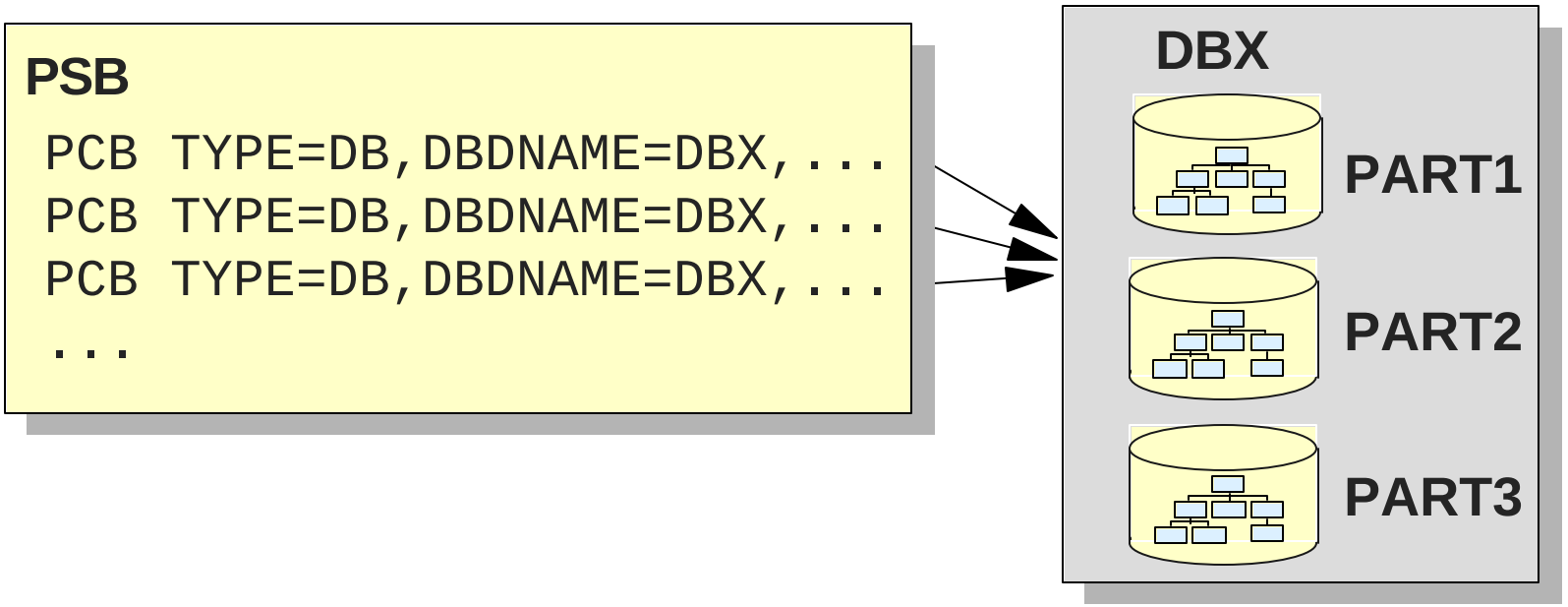
- Database split into multiple databases
- Application selects which database to use
  - Based on key of root segment
  - Database selection may be done by subroutine or modification to language interface module
  - PSB has PCB for each database



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



The world depends on it

Summary

- **Database Design Affects Application Possibilities**
  - Partitioning method
  - PHDAM randomization
- **Initial Loads**
- **Processing Partitions in Parallel**
- **Handling Unavailable Partitions**
- **Processing Secondary Indexes as Databases**
- **Converting from User Partitioning**

RETURN TO INDEX