DB2® for z/OS™ and OS/390® Performance Update - Part 1

Akira Shibamiya

Orlando, Florida October 1 - 5, 2001

M15a

NOTES

- **Abstract: The highlight of major performance enhancements in V7 of DB2 for OS/390 or z/OS, as well as recent performance enhancements made to V6, are covered in this presentation.**

- **Because of time constraints, only major performance enhancement highlights are covered. For more details, please see the redbook "DB2 for z/OS and OS/390 V7 Performance Topics" SG24-6129.**

Acknowledgment and Disclaimer

- **Measurement data included in this presentation are obtained by the members of the DB2 performance department at the IBM® Silicon Valley Laboratory.**
- **The materials in this presentation are subject to**
  - **enhancements at some future date,**
  - **a new release of DB2, or**
  - **a Programming Temporary Fix**
- **The information contained in this presentation has not been submitted to any formal IBM review and is distributed on an "As Is" basis without any warranty either expressed or implied. The use of this information is a customer responsibility.**

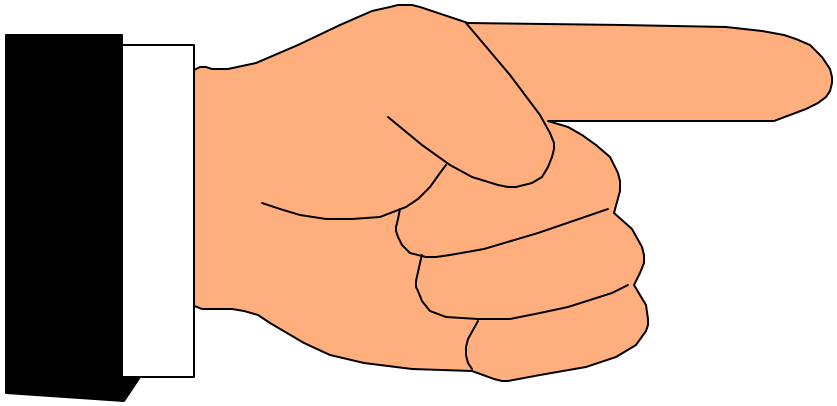
Outline

- Transaction Performance
  - INSERT, UPDATE, DELETE SQL
- Utility Performance
- Dataspace Buffer Pool with zSeries™ processor and OS/390 2.10 or z/OS

4

Transaction Performance

- Performance of
  - INSERT, UPDATE, DELETE SQL



NOTES

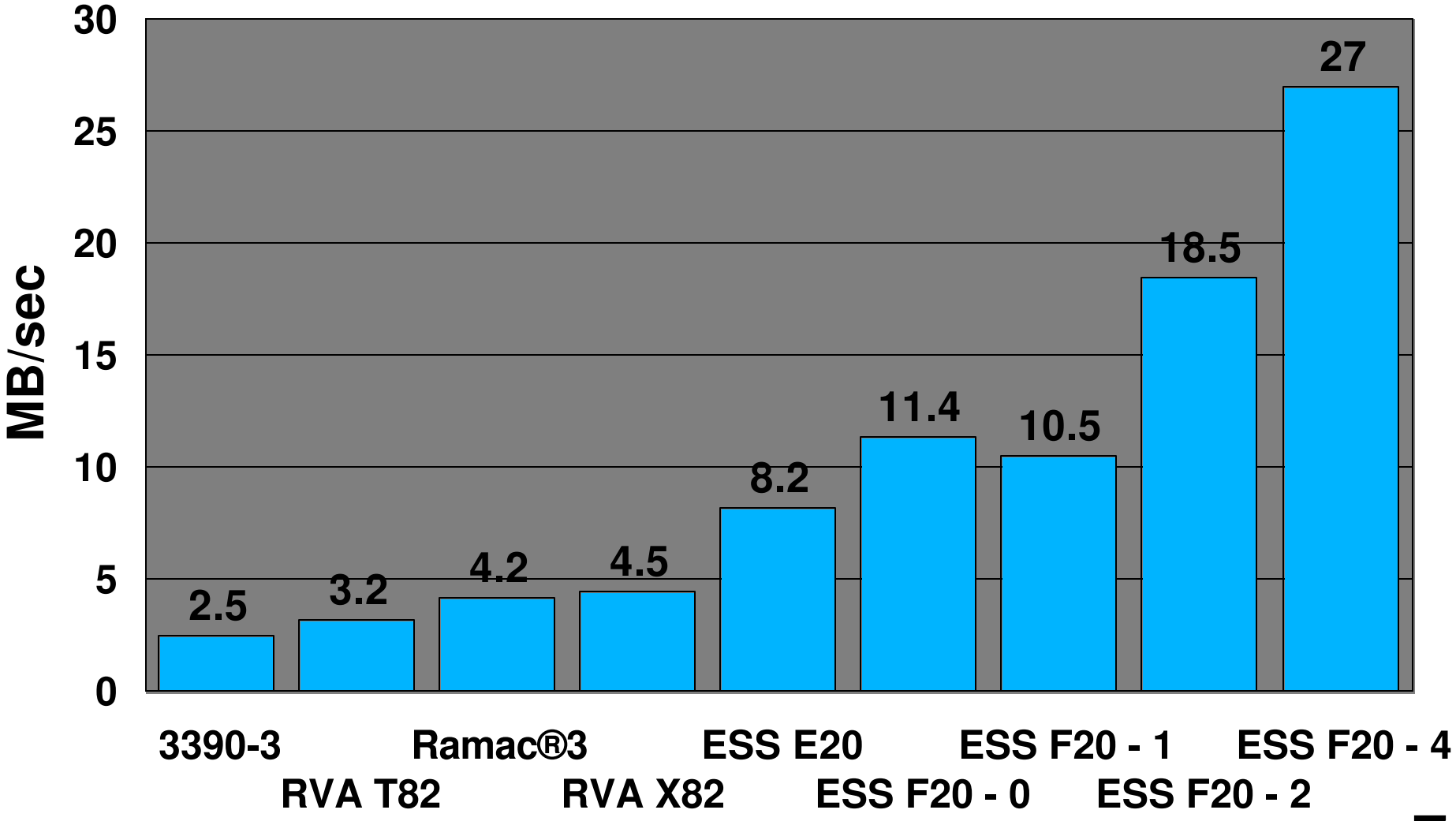
OUTLINE

- DB2 logging performance
- Asynchronous preformat
- Insert/Update performance enhancement
- Insert/Update performance tuning recommendation
- IRLM deadlock detection

6

VSAM I/O Striping for DB2 Log

Max observed rate of writing to active log



7

NOTES

- **Ramac1 and Ramac2 are similar to 3390-3.**
- **In "ESS F20-x", x indicates the number of stripes. This measurement from Ying Chang of IBM SSD, San Jose.**
- **For VSAM I/O striping, a data set must be SMS-managed, EF (Extended Format), and needs OS/390 2.10.**
- **8% less throughput from non EF log to a single striped EF log but no difference in CPU time.**
- **ESS F20 numbers include V6 PQ28857 8/99 for improved log write**

Log Statistics

Log activity/second

	OLTP	Batch
Log Records Created	3229	13600
Log CIs Created	70	1550
NEW Output Log Buffer Paged In	0	27.7
NEW Log Write I/O Request (Log 1 and 2)	417	35.6
NEW Log CIs Written (Log 1 and 2)	428	3172
NEW Log rate for 1 log in MB/sec	0.86	6.34

- 2 examples
  - OLTP (Online Transaction) insert with frequent commit
  - Batch insert with infrequent commit

NOTES

- **New Log statistics (V6 PQ28857 8/99 and DB2PM V7)**
  - **Average log record size = (log Control Intervals created * 4KB) / Log records created**
    - **So for OLTP, 70*4KB/3229 = 88 bytes average**
  - **Log Write I/O Request = Media Manager calls = sum of IFCID 38/39 pairs, should correspond to active log write I/O in RMF**

Log Statistics - continued

- LOG CIs (Control Intervals) Written represents a precise MB/sec log write rate as it counts each write of the same CI separately.
  - When more than 1MB/sec to one log, an attention should be paid on the log write performance.
  - Previously available Log CIs Created has been used to estimate the MINIMUM MB/sec log write rate as it does not account for the rewrite of the same CI.

NOTES

- Log CIs created vs Log CIs written
  - OLTP: 70 vs 428 or 214 per dataset
    - So each CI created is written 3 times per dataset
  - Batch: 1550 vs 3172 or 1586 per dataset
    - So each CI created is written once per dataset
  - In terms of active log dataset write performance, CIs written counter is a more meaningful measure than CIs created
- Log rate for 1 log = (Log CIs written * 4KB)/2 if dual log
  - So for OLTP, 428*4KB/2 = 0.86MB/sec

Log Statistics - continued

- Output Log Buffer Paged In
  - Can indicate insufficient real storage to back up log buffers (V6 PQ39000 6/00)
  - If class 19 log latch contention is also high (>200to1000/sec),
    - Reduce output log buffer size
    - Get more real storage if <2GB
    - Else use OS/390 2.10 or z/OS and zSeries processor with >2GB real storage

NOTES

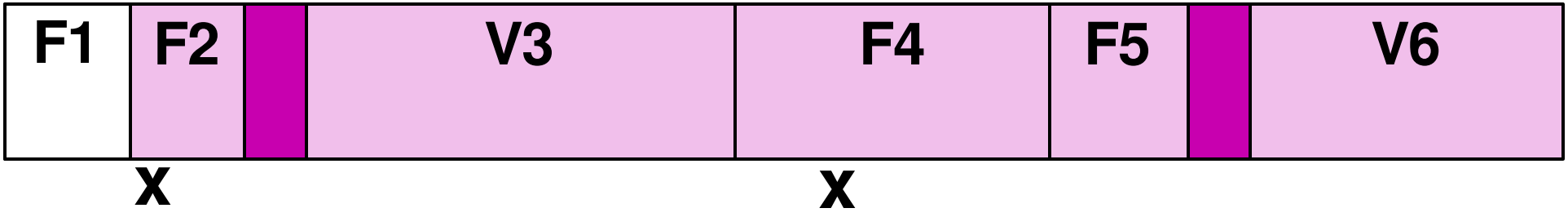
- **Maximum output log buffer size =**
  - **4MB in V5**
  - **400MB in V6**
  - **Watch out for real storage availability (Output Log Buffer Paged In counter) when increasing output log buffer size**
- **OS/390 2.10 or z/OS and zSeries processor enable exploitation of >2GB real storage**
- **Maximum active and archive log dataset size increased from 2GB to 4GB (V6 PQ48126 5/01)**

Reduction in the size of log data created

- Only log from the first changed byte to the last changed column in variable-length row update without length change, instead of from the first changed byte to the end of the row
  - -35% log CI created in one customer measurement
  - -8% log CI created in another
  - Greater improvement observed in less tuned environment

NOTES

- Let F1, F2, F4, and F5 be fixed length columns, and V3 and V6 variable length columns. F2 and F4 columns are updated as indicated by 'x'.

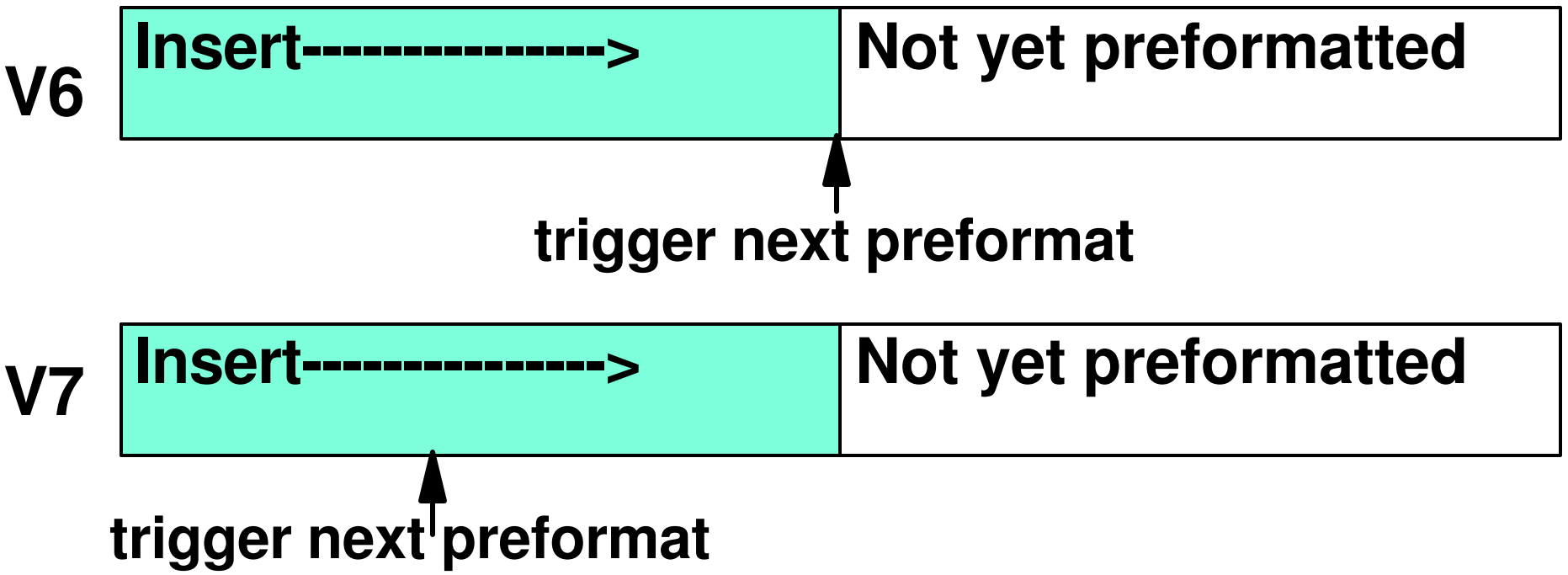


  - 2 dark areas indicate 2 byte length field preceding variable length columns.

- If fixed-length row update, then from first changed column to last changed column is logged.
- Log CI = 4KB log Control Interval

Asynchronous Preformat

- Preformat engine scheduled ahead to reduce wait time for preformat



  - Maximum possible reduction in elapsed time with one inserter can be estimated from Dataset Extend wait time in class 3 accounting

17

NOTES

- The concept of asynchronous preformat is similar to that of asynchronous prefetch by triggering preformat by anticipating the near-future need.
- In an Insert-heavy workload to one or small number of data sets and I/O-bound, asynchronous preformat may not result in the net elapsed time reduction equal to the dataset extend wait time, as other wait times may increase.
  - The net elapsed time reduction ranges from 0 to Dataset Extend wait time in accounting class 3.
- With multiple inserters to the same dataset, both Dataset Extend wait and Lock wait time can go down.

Insert/Update Performance Enhancement

- OS/390 2.10
  - VSAM I/O striping to remove log I/O bottleneck
  - Dataspace buffer pool for index, at least to avoid nonleaf page read I/O (with zSeries processor)
- DB2 V7
  - Less log data created in Update
  - Asynchronous preformat to reduce elapsed time
  - Avoid unnecessary dynamic prefetch and read I/O in non-segmented tablespace insert by multiple agents
- ESS PAV to reduce I/O contention, especially for non partitioning indexes

NOTES

- Rule-of-Thumb to keep nonleaf pages in buffer pool
  - Buffer pool size > 2*(number of nonleaf pages)
- V7 change in sequential detection algorithm in Insert to track read I/O instead of pages accessed without read I/O can reduce unnecessary scheduling of dynamic prefetch and read I/O
  - Also for insert to end of non-segmented tablespace or segmented mass deleted pages (V6 PQ42372 12/00)
- ESS PAV = Parallel Access Volume feature of ESS to enable concurrent I/O's to the same volume at the same time

Insert/Update Performance Tuning Recommendation

- For a large table with non partitioning indexes, I/O typically becomes the most critical performance bottleneck.
- If index I/O-bound,
  - ESS with PAV
  - Non partitioning index piece
  - Hiperpool, or Dataspace buffer pool on zSeries processor
  - Default freespace

21

NOTES

- PAV = Parallel Access Volume to allow multiple concurrent I/O's to the same volume

- If data I/O-bound,
  - ESS, with PAV if possible
  - V7 asynchronous preformat or Load/Reorg with preformat option
  - 0 freespace

- If log or non DB2 data I/O-bound,
  - ESS, with PAV if possible
  - I/O striping (VSAM, BSAM)

Insert/Update Performance Tuning Recommendation - continued

- Deferred write threshold tuning for update-intensive data or index
- Segmented tablespace for variable-length record insert or update, and mass delete
- V7 Online Load Resume to reduce CPU time while minimizing concurrency impact
- Data sharing
  - MEMBER CLUSTER tablespace to minimize spacemap and data page P-lock contention
  - V7 CURRENT MEMBER special register to enable insert by tablespace or partition by member to reduce data sharing overhead

23

NOTES

- **For recommendations on deferred write threshold tuning, please see Insert Performance Considerations in DB2 for OS/390 in 2000 DB2 Tech Conference**

Deadlock detection interval

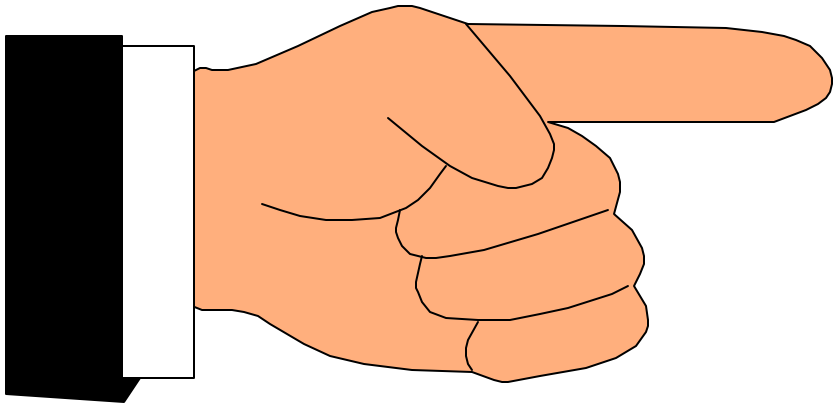
- Deadlock detection interval reduction (IRLM PQ44791 5/01)
  - Before Apar: 1 to 5sec interval
  - After Apar: 0.1 to 5sec interval
  - To keep pace with ever-faster processors
  - Negligible performance impact

NOTES

- 0.1sec versus 1sec deadlock detection interval
  - +0.44% IRWW (IBM Relational Warehouse Workload) transaction CPU time because of IRLM CPU time increase
- New MODIFY irlmproc,SET,DEADLOK= command to dynamically change deadlock frequency
- Dynamic change of timeout value without shutting down and restarting DB2 subsystem
  MODIFY irlmproc,SET,TIMEOUT=1to3600sec
  via IRLM PQ38455 6/00

26

Utility Performance



27

NOTES

OUTLINE

- Faster and more available Online Reorg
- Online Load Resume
- Parallel Partition Load from multiple input data sets in one job
- Load performance tuning recommendations
- Fast Unload
- Modify Recovery

Faster and more available Online Reorg

- Data Availability during Online Reorg

Online Reorg Phase	Data Access Allowed
Unload	Read/Write
Reload	Read/Write
Sort and Build Index	Read/Write
Log processing iterations	Read/Write
Last log iteration	Read Only (1)
Switch	None (2)
Build2	None to affected area (3)

V7

V7

29

NOTES

- **3 phases in Online Reorg do not allow full Read/Write access of data.**

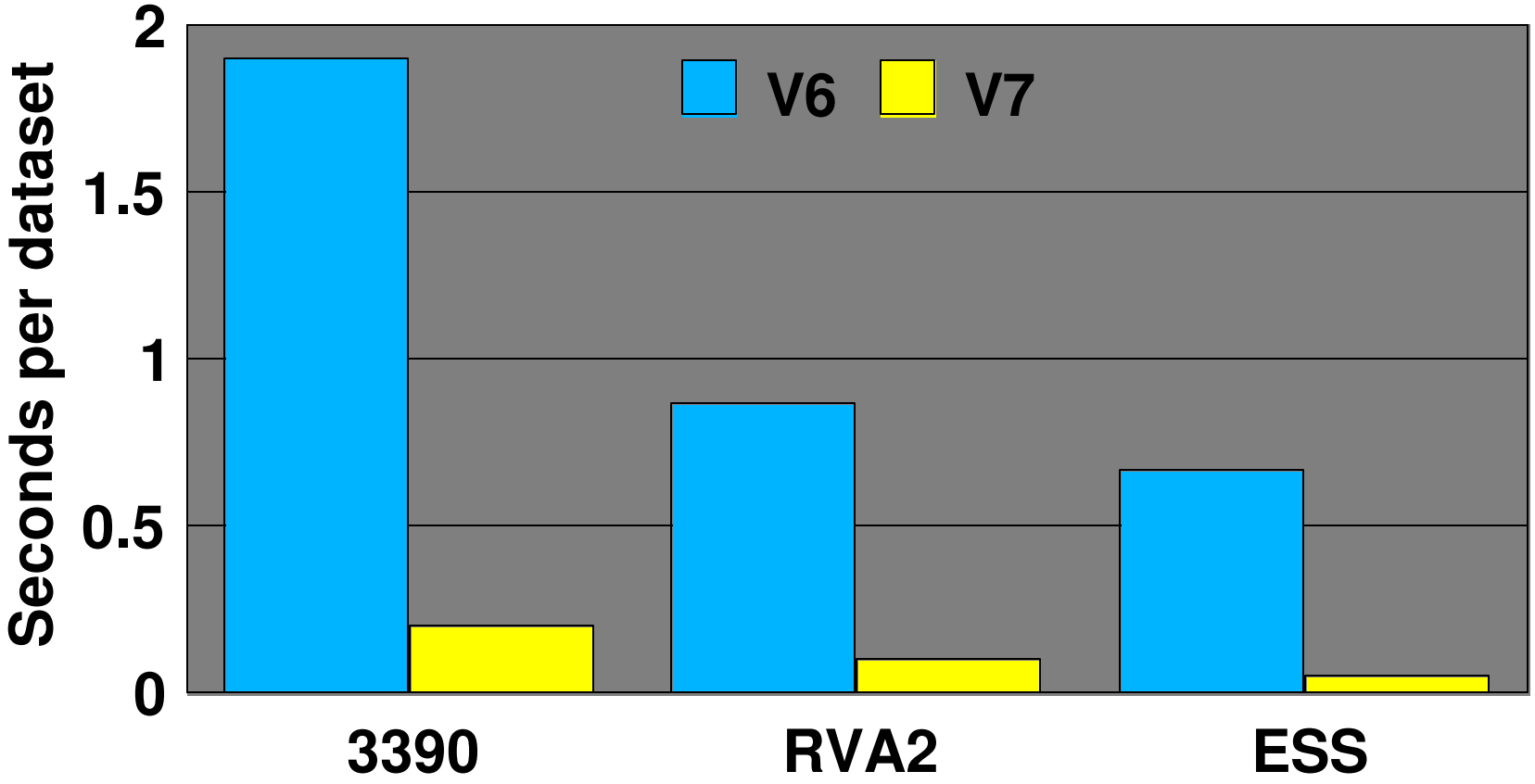
1. **Last Log Iteration phase supports read only access. However, this phase is typically very short and also the duration can be user-controlled by REORG MAXRO parameter.**

2. **Switch phase time which is a function of the number of data sets to be renamed can take in the range of 0.7 to 2 seconds per dataset. So, for a tablespace with 100 partitions, no access is allowed for up to 400 seconds. (100 datasets for data and 100 datasets for partition index)**

3. **Build2 phase which takes place in Online Reorg of selected partitions with non partitioning indexes (NPI) stops access to Reorg'd partitions as well as entire NPIs.**

Online Reorg - continued

- Fast switch phase by avoiding dataset RENAME
  - About 10 times faster switch phase
    - ➡ 10 times availability improvement



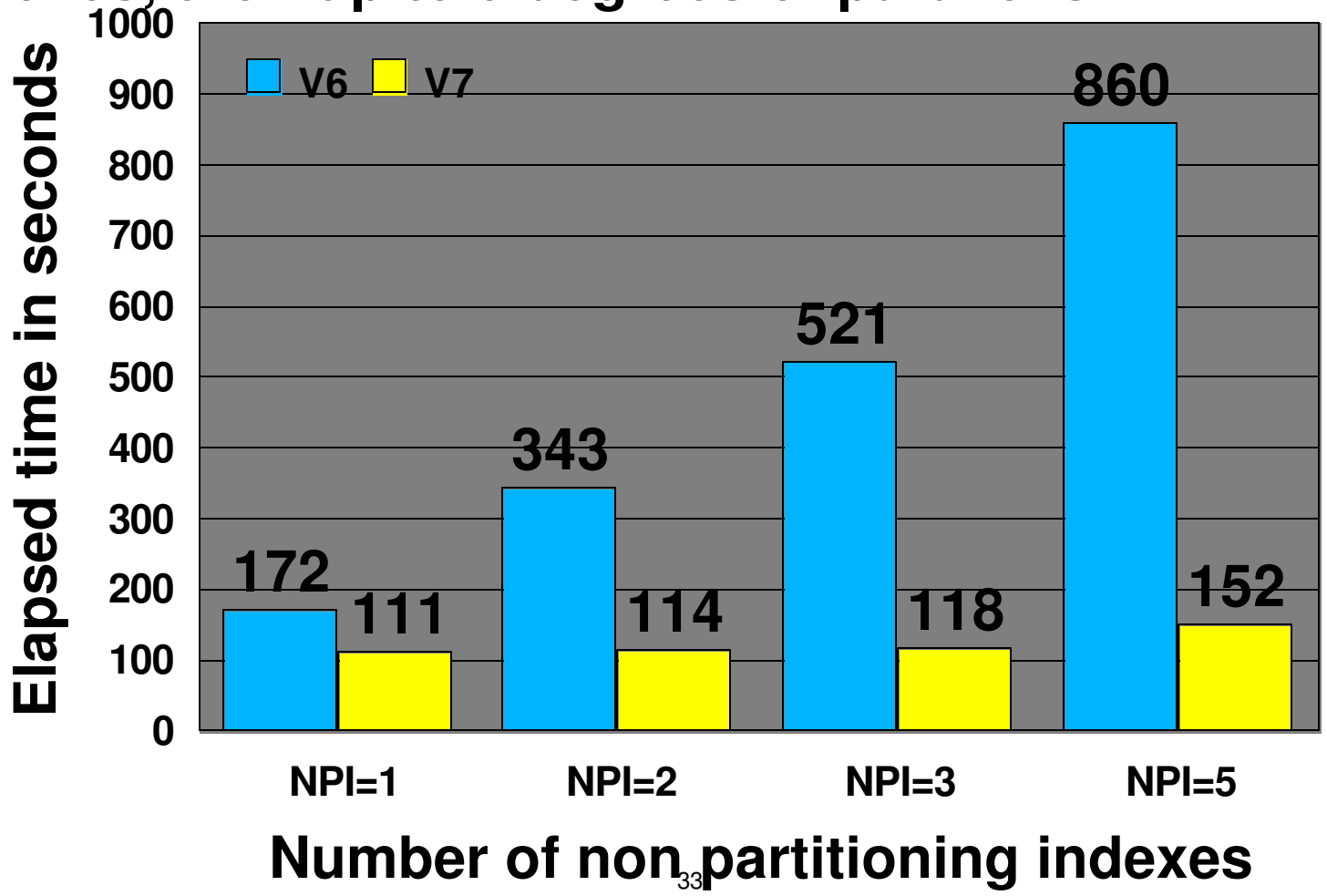
NOTES

- **Switch phase time of**
  - **0.7 to 2 seconds per dataset in V6**
  - **0.05 to 0.2 seconds per dataset with V7 Fast Switch**

depending on the device type

Online Reorg - continued

- **Parallel Build2 phase**
  - **If Partition Reorg of partition(s) with 5 non partitioning indexes, then up to 5 degrees of parallelism**



NOTES

- **Measurement of Reorg of 3 out of 10 partitions in 20 million row tablespace**

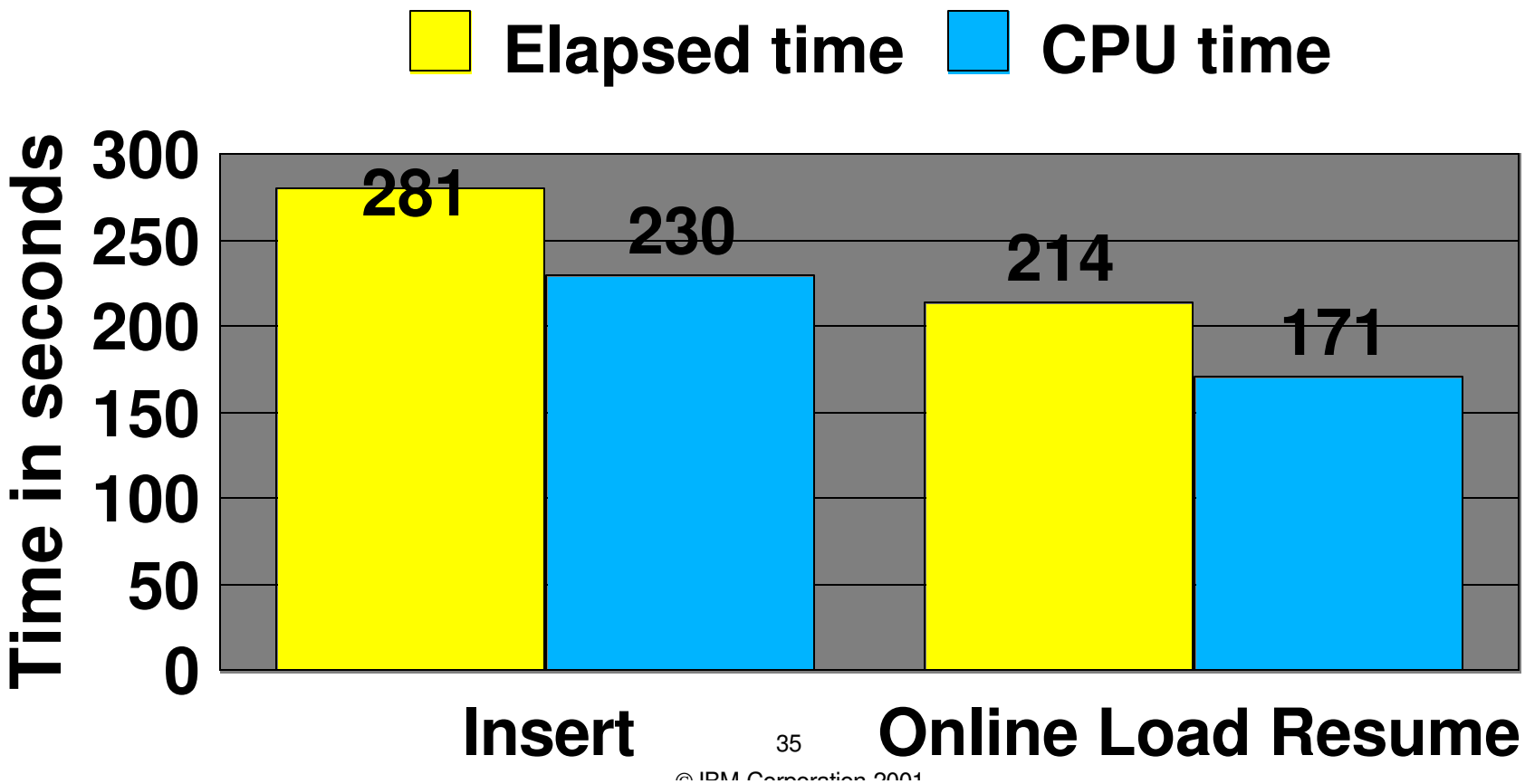
Number of non partitioning index	Elapsed time reduction
1	1.55 times (35%)
2	3 times (67%)
3	4.4 times (77%)
5	5.66 times (82%)

  - **Improvement even for one NPI because of elimination of unnecessary logging for each index entry**

34

Online Load Resume

- Functionally operate like SQL Insert
  - But up to 37% less CPU time by avoiding Insert API overhead
  - Corresponding elapsed time reduction also possible



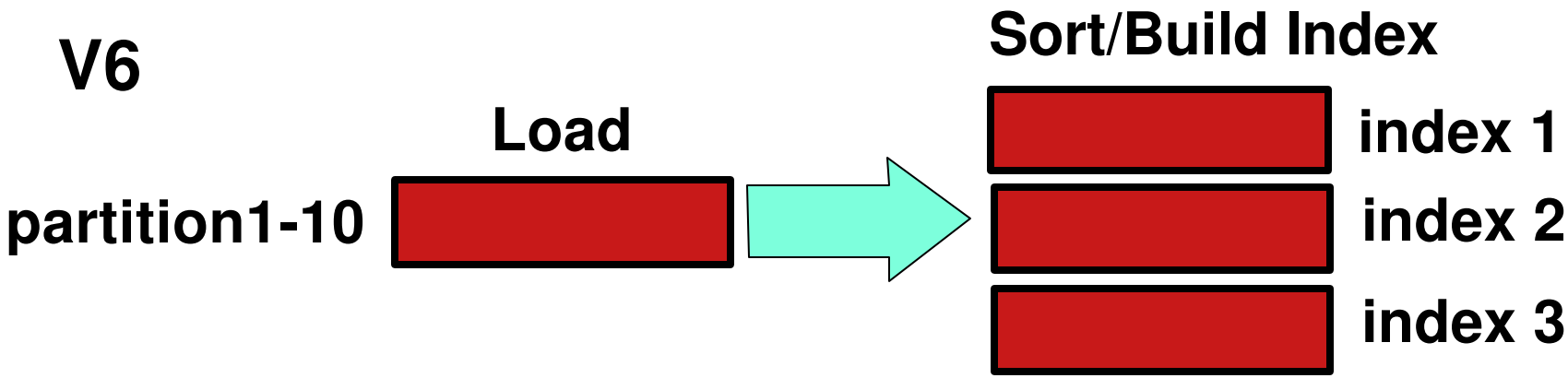
35

NOTES

- Unlike normal Load Resume, Online Load Resume provides the same level of concurrency as Insert
- API = Application Program Interface. There is a fixed overhead of invoking an SQL call such as Select, Open, Fetch, Close, Insert, Update, Delete, etc.
- Online Load Resume vs Insert measurement example
  - 2 million rows added to 8 million row table with 2 indexes, resulting in 10 million row table
  - 24% reduction in elapsed time
  - 26% reduction in CPU time
- 15 to 37% elapsed time and CPU time reduction for 1 to 3 indexes, with 1 index getting the best improvement.

Parallel Partition Load in one job

- By up to N tasks for N partitions with N input datasets (used to be sequential here in V6)
  - followed by parallel sort and build for M indexes



- About 30% faster compared to one Load job for the whole table

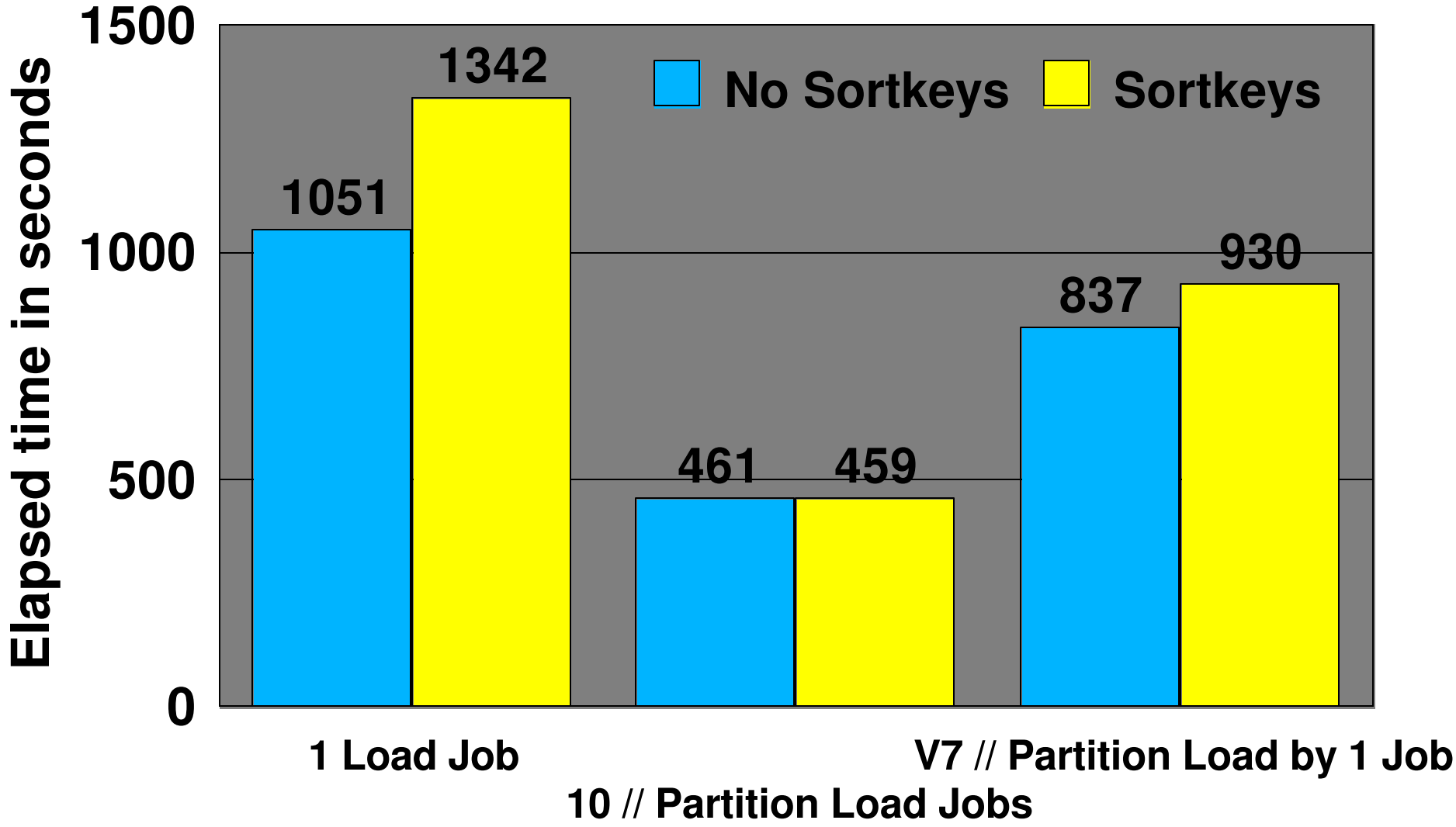
37

NOTES

- In the following pages, 50 million row Load into a 10 partition tablespace was measured using 12way G6 turbo and ESS.
- A bigger elapsed time reduction possible for parallel processing if well-tuned I/O configuration, also BSAM i/o striping
- A similar result was obtained for 20 partition tablespace containing 50 million rows

Parallel Partition Load - continued

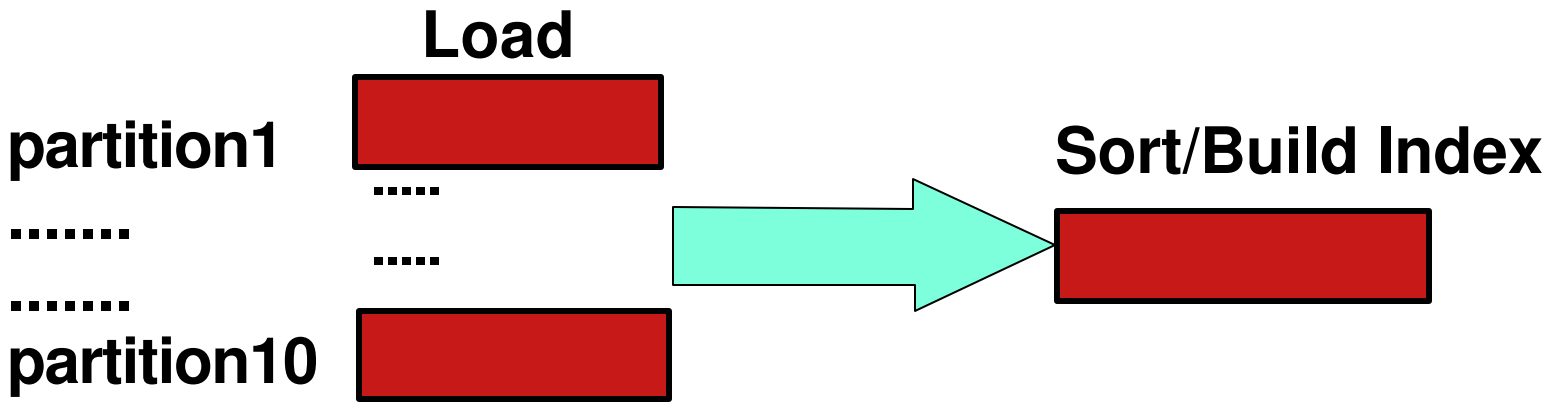
- Load into table with 1 index



39

NOTES

- When there is only one index on a table to be loaded,
  - 10 // (Parallel) Partition Load Jobs the fastest.
  - V7 // Partition Load by 1 Job is faster than V6 1 Load Job, but slower than 10 // Partition Load Jobs because of sequential index build.

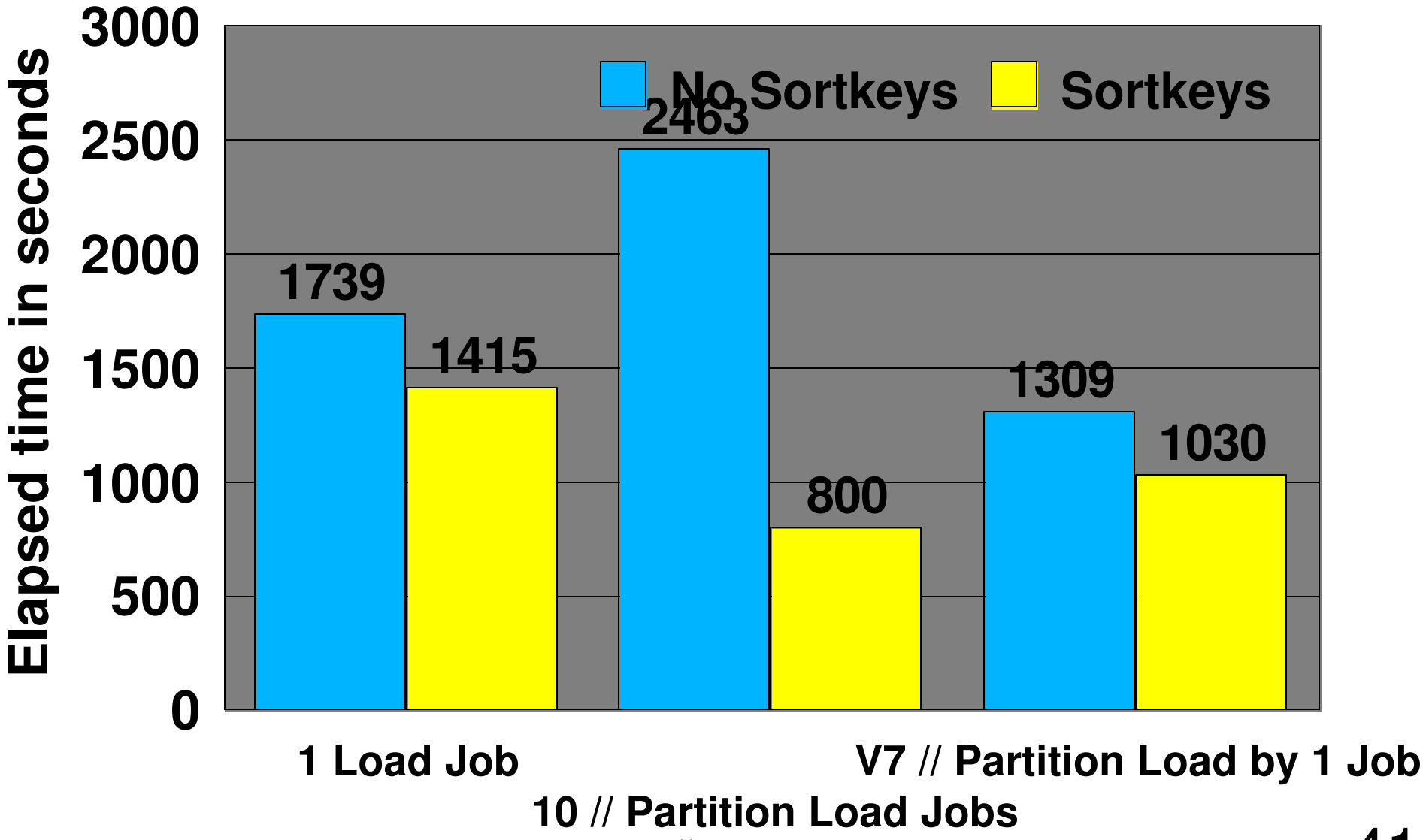


  - SORTKEYS option, which always calls DFSORT, tends to be slower if input data is already in key sequence.

40

Parallel Partition Load - continued

- Load into table with 2 indexes



41

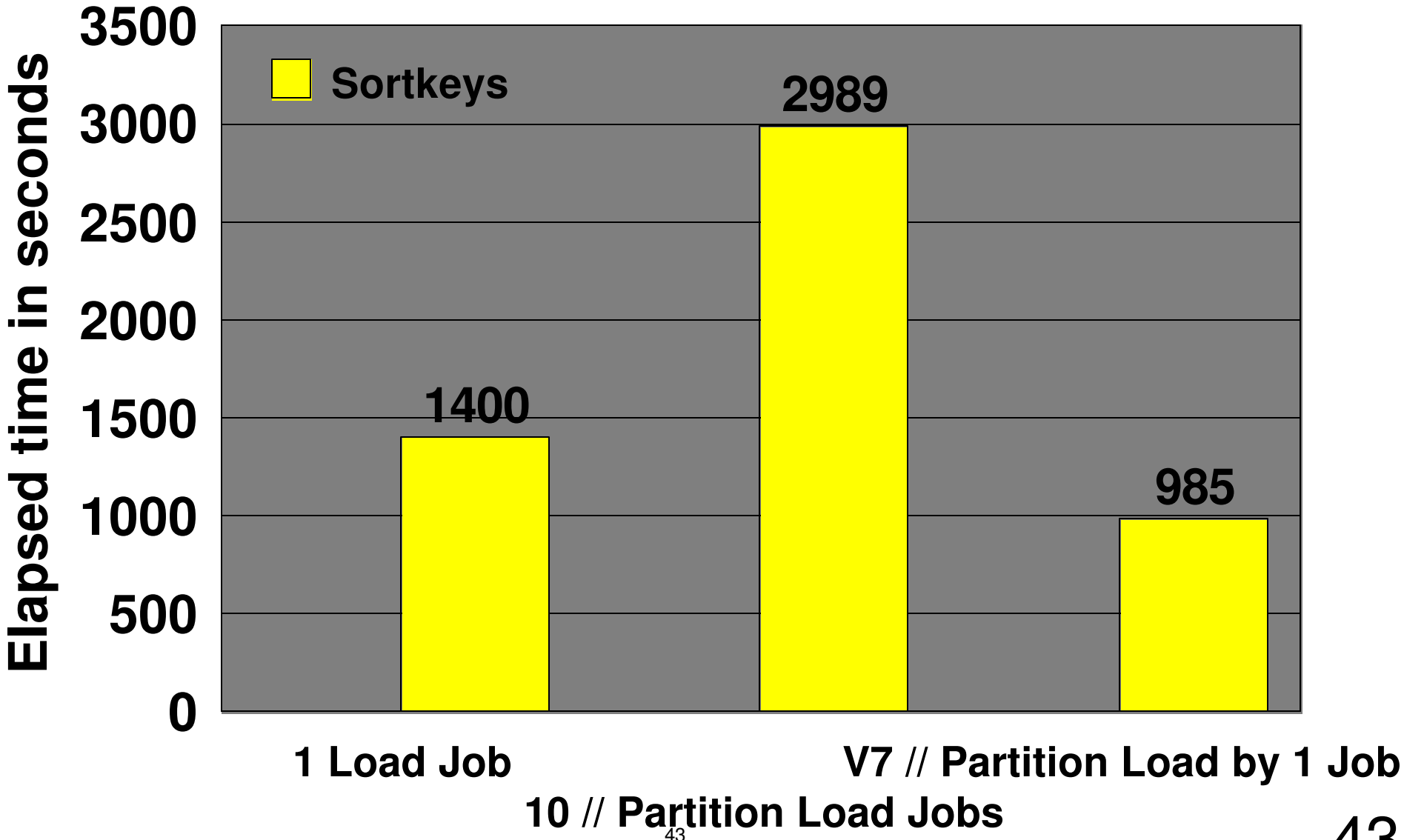
NOTES

- When there are 2 indexes on a table to be loaded,
  - SORTKEYS option is clearly better.
    - No DASD Read/Write for SYSUT1 and SORTOUT work files for indexes (V5)
    - Parallel sort and build of indexes (V6)
    - SORTKEYS option always recommended if more than 1 index
  - 10 // (Parallel) Partition Load Jobs is the fastest, followed by V7 // Partition Load by 1 Job. However, since the latter uses about 20% less CPU time, it is considered to be a toss up.

42

Parallel Partition Load - continued

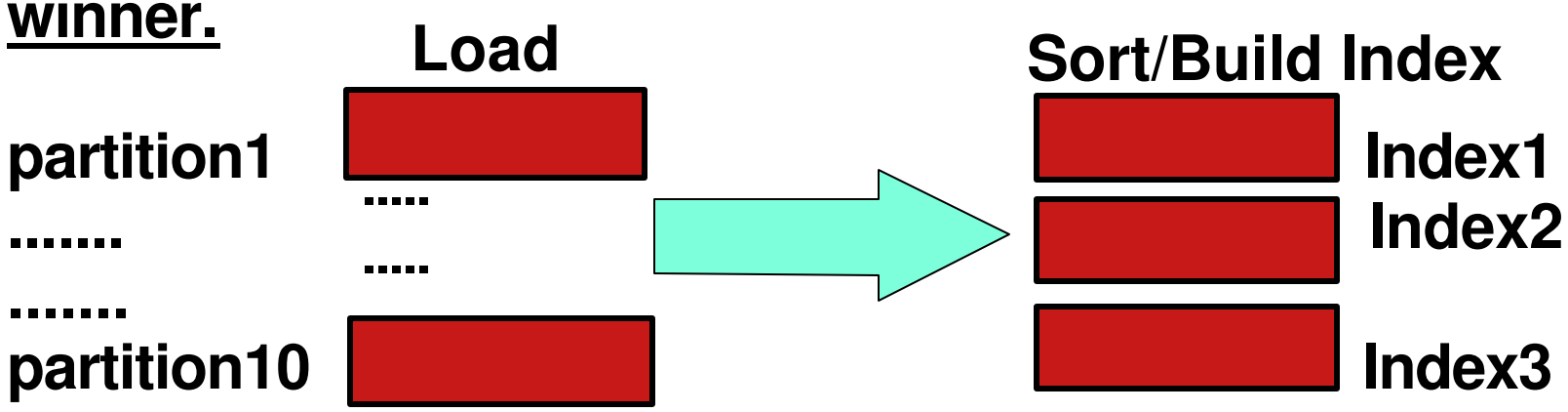
- Load into table with 3 indexes



43

NOTES

- When there are 3 indexes on a table to be loaded,
  - As a result of multiple jobs contending for non partitioning index pages, // Partition Load Jobs deteriorate rapidly as the number of indexes increases because of much higher CPU time for non-partitioning index processing.
  - V7 // Partition Load by 1 job with SORTKEYS option is a clear winner.



  - The same story for more than 3 indexes

Load Performance Tuning Recommendations

- If one index, parallel partition Load jobs
- If multiple indexes, V7 parallel partition Load in 1 job with SORTKEYS option

- If I/O-bound,
  - ESS with PAV
  - I/O striping (BSAM, VSAM)

- If Load Resume Yes, similar to Insert recommendation

- If DB2 data compression, zSeries processor

45

NOTES

- Use SORTKEYS option unless there is only one index and data is in index key sequence
- PAV = Parallel Access Volume to allow multiple concurrent I/O's to the same volume
- Other possibilities
  - V7 Online Load Resume for high concurrency
  - V7 Union Everywhere and Parallel Load of small tables

Fast Unload

- More function and 15 to 18% less CPU time compared to Reorg Unload External
  - Reorg Unload External, which was made available in V6 and V5 apar PQ19897 11/98, uses up to 4 to 9 times less CPU time compared to DSNTIAUL.
- 2 times faster with partition parallelism in one measurement (similar to user-controlled parallelism in V6)

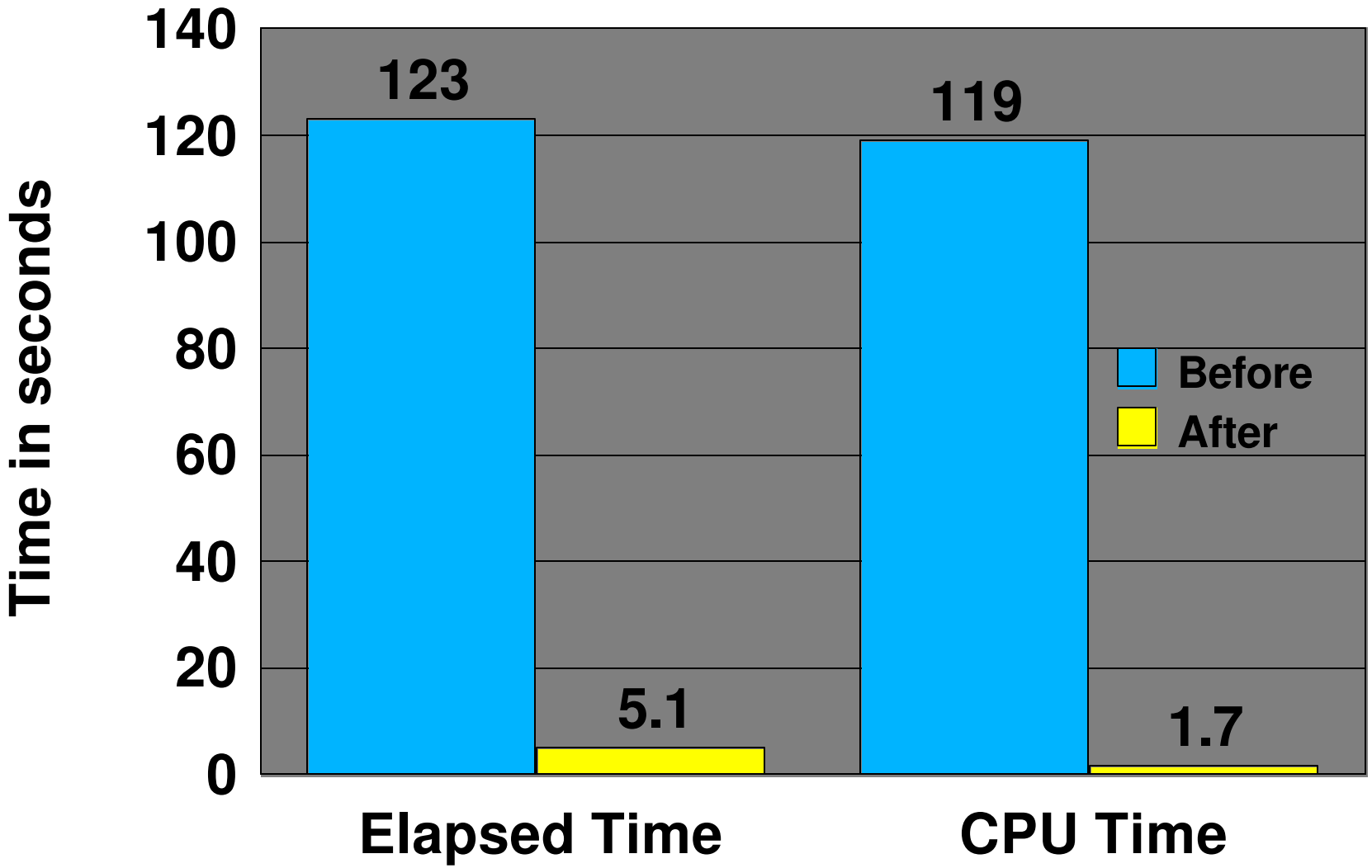
47

NOTES

- **V7 Unload utility, which is independent of Reorg utility, is an enhanced version of Reorg Unload External**

48

Modify Recovery Performance



49

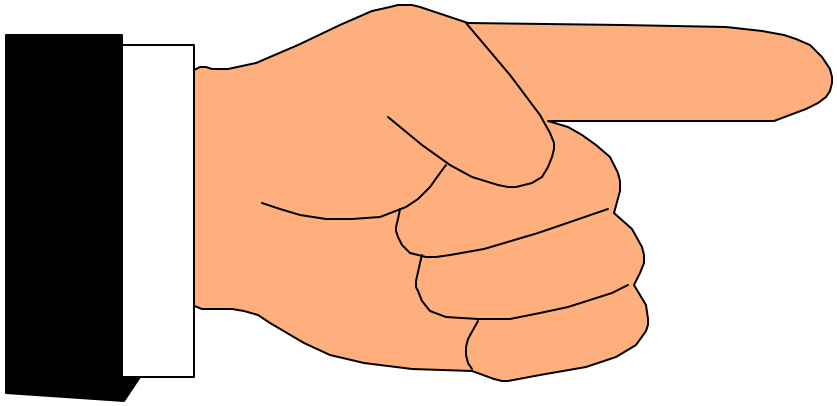
© IBM Corporation 2001

NOTES

- MODIFY to delete from SYSCOPY, SYSLGRNX, and DBD
- Example of MODIFY DELETE AGE(*) for one tablespace
  - Deleting 7500 rows out of 1 million SYSCOPY rows
  - 24 times less Elapsed Time and 70 times less CPU time
  - V5/V6/V7 PQ45184 5/01

	Before Apar	After Apar
Lock Request	7,400,000	8,000
Unlock Request	7,400,000	37
Catalog Getpage	400,000	100,000
Catalog Update	42,000	42,000

Dataspace Buffer Pool with zSeries Processor and OS/390 2.10 or z/OS



51

NOTES

52

Dataspace Buffer Pool

- With zSeries processor and OS/390 2.10 or z/OS enabling exploitation of >2GB real storage, Dataspace Buffer Pool becomes a powerful force in
  - DBM1 virtual storage constraint relief
  - Bigger buffer pool to reduce DASD I/O rate, promoting performance scalability for bigger and faster processors
    - Doubling the throughput when the processor power doubles

NOTES

- **zSeries processor became available 12/00**
- **Dataspace buffer pool was introduced in V6 DB2, but its full exploitation was not possible until >2GB real storage is supported.**

54

Dataspace Buffer Pool - continued

- **Dataspace buffer pool with sufficient real storage**

1. **Typically 0 to 10% additional CPU time compared to virtual pool. However, CPU and elapsed time reduction possible compared to storage constrained virtual pool.**
2. **Up to 8 million buffers total (32GB with 4K pages)**
3. **Direct I/O support - more concurrent parallel prefetch I/O's can be supported**
4. **Can contain updated pages not yet written to DASD - better for update-intensive data**
5. **Reduced class14 latch contention because no VP-HP page movement**

NOTES

- For sufficient storage, make sure PAGE IN FOR READ or WRITE is less than 5% of pages read or written in steady state
- Higher %CPU overhead for page-processing intensive application, such as a query simply scanning a large amount of data
- Relevant apars
  - OS/390 R10 preconditioning 64bit support to allow dataspace BP and global dynamic statement cache to be fixed above 2GB real storage (V6 PQ25914 4/99)
  - Same as above for virtual pool and log buffers (V6 PQ36933 4/00)
  - Both apars remove the need to move the page to be fixed below 2GB
- VP = Virtual Pool, HP = Hiper Pool

56

Dataspace Buffer Pool - continued

- **Hiperpool comparison**

1. **More overhead compared to dataspace buffer pool with sufficient real storage**
2. **Up to 8GB total**
3. **No direct I/O to/from HP - less concurrent I/O's can be supported**
4. **Can not contain updated pages not yet written to DASD**
5. **High class14 latch contention possible even with FIFO buffer steal option**

NOTES

- Support of HP (Hiperpool) via Fast Synchronous Data Mover Facility without ADMF on G5 and up (V4/V5/V6 PQ38174 6/00)
  - ADMF = Asynchronous Data Mover Facility, which has been a prerequisite since DB2 V3 for HP support.
- FIFO = First In First Out buffer steal option, recommended for data or index which are entirely resident in buffer pool (100% hit) or practically never resident (nearly 100% miss)
- IFCID 198 indicates where buffer is found
  - VP (Virtual Pool)
  - HP (Hiper Pool)
  - GBP (Group Buffer Pool)
  - Dataspace Buffer Pool

58

Dataspace Buffer Pool - continued

- **Conclusion: With zSeries processor and OS/390 2.10 or z/OS, dataspace buffer pool is recommended over hiperpool.**

NOTES

- **Prior to zSeries processor and OS/390 2.10, hiperpool recommended over dataspace buffer pool**

60

Dataspace Buffer Pool - continued

- Other performance benefits of zSeries for DB2
  - 15 to 30% faster on single engine and 30 to 75% more throughput than G6 turbo
    - on non-compressed data
  - 3 to 4 times more efficient hardware compression than G6 turbo
    - So 30 to 60% faster on compressed data
- Therefore, more than doubling of throughput possible with zSeries compared to G6 turbo

61

NOTES

- **Maximum of 12 engines in G6 turbo, 16 in zSeries processor**

References

- Redbooks at www.redbooks.ibm.com
  - DB2 for z/OS and OS/390 V7 Performance Topics SG24-6129
  - DB2 UDB Server for OS/390 V6 Technical Update SG24-6108
  - DB2 UDB for OS/390 V6 Performance Topics SG24-5351
- DB2 UDB for OS/390 Administration Guide, Performance Monitoring and Tuning Section, SC26-9003 for V6, SC26-9931 for V7
- More information on DB2 UDB for OS/390 at www.ibm.com/db2

DB2® for z/OS™ and OS/390® Performance Update - Part 2

Akira Shibamiya

Orlando, Florida October 1 - 5, 2001

M15b

NOTES

- **Abstract: The highlight of major performance enhancements in V7 of DB2 for OS/390 or z/OS, as well as recent performance enhancements made to V6, are covered in this presentation.**

- **Because of time constraints, only major performance enhancement highlights are covered. For more details, please see the redbook "DB2 for z/OS and OS/390 V7 Performance Topics" SG24-6129.**

Acknowledgment and Disclaimer

- **Measurement data included in this presentation are obtained by the members of the DB2 performance department at the IBM® Silicon Valley Laboratory.**
- **The materials in this presentation are subject to**
  - **enhancements at some future date,**
  - **a new release of DB2, or**
  - **a Programming Temporary Fix**
- **The information contained in this presentation has not been submitted to any formal IBM review and is distributed on an "As Is" basis without any warranty either expressed or implied. The use of this information is a customer responsibility.**

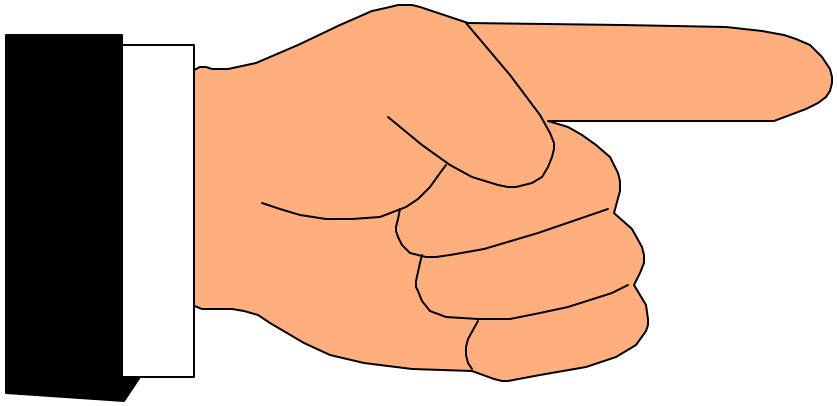
Outline

- Query Performance
  - More efficient access path
  - Query parallelism
  - More efficient bind of many-table join
- V5 to V6 Performance Regression
- Fetch First N Rows
- Backward Dynamic Prefetch
- Catalog Migration Performance

4

Query Performance

- **Performance of SELECT SQL scanning many rows**



NOTES

OUTLINE

- More efficient access path
- Query parallelism
- More efficient bind of many-table-join

More Efficient Access Path

- Matching index access in "column comparison-operator column expression" in join

```
Select from A,B where a1=SUBSTR(b1,1,8) ...
```

```
Select from A,B where a1=b1+x ...
```

  - Use index on a1 if table A accessed after table B
  - Particularly useful for large table A

NOTES

- **Exploitation of available index(es) has always been one of the top performance requirements in DB2 since V1.**

  V7 continues to further exploit available indexes for performance advantage.

More Efficient Access Path - continued

- For compatible column types such as numeric to numeric and target column having equal or bigger size and/or precision
  - float a1 = cast(decimal b1)
  - integer a1 > smallint b1 + x
  - date a1 between date(b1) and date(b1) + 10 days
  - char a1 = char b1||char b2

NOTES

- Join examples here show an index on a1 can be used if the table A is access after the table B
- Related enhancements in review
  - Scalar and noncolumn expression indexable in V5
    - a1 = date(host-variable)
    - a1 = x+y
  - Char/Varchar of different size indexable in equi-join with V5 PQ22046 and PQ24933 3/99
    - 10byte char a1 = 20byte varchar b1

More Efficient Access Path - continued

- Indexable non-correlated IN subquery in Select/Update/Delete
  - Introduced by V5 PQ23243 3/99 and V6

```
where a1 IN (Select b1 from ...)
```

  - Extended in V7 to
    - =ANY, =SOME
    - Multi-column (row expression)

```
where a1 =ANY (Select b1 from ...)

where (a1,a2) IN (Select b1,b2 from ...)
```

NOTES

- **AccessType = N**
  - **i.e. no join in Plan Table**
  - **Index on a1 or a1,a2 can be used**

More Efficient Access Path - continued

- Correlated subquery transformation to join

```
Select a1 from A where a1 in
                    (Select b1 from B where b2=a2) ...
```

transformed into

```
Select a1 from A,B where a1=b1 and b2=a2 ...
```

  - IN, =ANY, =SOME, EXISTS correlated subquery can be processed as stage 1 and transformed to join in Select, Update, Delete
  - Typical 2 to 10 times faster and less CPU time possible if large outer and small subquery results

NOTES

- Both correlated and non correlated subquery transformed to join prior to V7 if single table and single column guaranteed by unique index to have unique values. In V7 correlated subquery,
  - searched Update and Delete supported
  - subquery result does not have to be unique
  - EXISTS subquery supported

- In general, join can be more efficient than subquery because Optimizer can choose the best table join sequence (A to B, or B to A).
  - In subquery, sequence of table processing is fixed (A to B).

- Explain shows join if applicable

More Efficient Access Path - continued

- Avoid unnecessary sort when equal predicate on Order By key

```
Select from A where a1=x and a3=y order by a1,a2,a3
```

equivalent to

```
Select from A where a1=x and a3=y order by a2
```

➡ Thus, index on a2 could avoid sort in V7.

NOTES

- **Optimizer still relies on cost estimate to determine the best access path**
  - **Therefore, sort avoidance occurs only if that results in the lowest cost estimate.**
  - **For example, if an index on a1 and another on a2,**
    - **a2 index can avoid sort**
    - **but a1 index may be preferable, depending on what % of rows satisfy a1=x, how many rows are to be fetched (Optimize for N Rows), cluster ratio, etc.**

More Efficient Access Path - continued

- Avoid unnecessary sort when equal predicate on Order By key - join example

```
Select from A,B where a1=x and ... order by
a1,b2
```

equivalent to

```
Select from A,B where a1=x and ... order by b2
```

➡ Index on b2 could avoid sort in V7

NOTES

- **Prior to V7, it was not possible to avoid sort by creating an index, when a sort key consists of columns from different tables.**
  - **In V7, it is now possible to avoid sort even when sort key columns come from multiple tables.**

More Efficient Access Path - continued

- Immediately go to end of file for predicate "not null column is null" (V6 PQ39261 8/00)

```
Create Table A
        a1 INTEGER NOT NULL
             ....
Select from A where a1 is null
```

  - Previously, all rows in table A scanned to find no qualifying rows.

NOTES

- **If V6 apar number is specified in parentheses, an enhancement is retrofitted to V6. An apar closing date is also shown if already closed.**

More Efficient Access Path - continued

- Direct access for both MAX and MIN can now be supported with only one index

```
Select MAX(a1) from A where a1<X

Select MIN(a1) from A where a1>X
```

  - In V6, descending index on a1 and ascending index on a1 are required for MAX and MIN, respectively, to avoid a possibly long index sequential scan
  - In V7, only one index, either ascending or descending, is needed to avoid a scan

21

NOTES

- **AccessType of I1 instead of I indicates an efficient one-fetch index access**

	V6	V7
MAX(a1) with ascending index on a1	I	I1
MAX(a1) with descending index on a1	I1	I1
MIN(a1) with ascending index on a1	I1	I1
MIN(a1) with descending index on a1	I	I1

More Efficient Access Path - continued

- Index-only access for VARCHAR when results won't change, e.g.
  - VARCHAR in where clause
  - No VARCHAR in scalar function, LIKE, Group By, Select list
  - Example: Select a1,a2 from A where a3=x with index on a1,a2,a3
    - Index only in V7 but not V6 if a1 and a2 CHAR and a3 VARCHAR
    - No index only if a1 or a2 VARCHAR

NOTES

- **Less restrictive but more error-prone index-only VARCHAR zparm option in V5 not recommended in general**
- **Index-only access can benefit transaction performance also.**

More Efficient Access Path - continued

- Minimize premature elimination of multi-index access path, ie index ANDing/ORing (V5 PQ39587 8/00)
- Optimizer cost formula change to account for index I/O cost reduction for prefetch to enable optimal access path selection (V5/V6 PQ38106 7/00)
  - Enabled via OPTPREF zparm in V5/V6, no zparm in V7

NOTES

- **A RID pool threshold will be applied AFTER considering more index matching predicates**

  Example: SELECT FROM A WHERE a1=? AND (a2=? OR a3=?), with one index on a1,a2 and another on a1,a3

  - **If 100M(million) row table with filter factors FFa1=10%, FFa2=6%, FFa3=4%,**
    then estimated number of rids to be processed =

    - **Before: 100M*10% = 10M rids**
    - **After: 100M*(10%*6% + 10%*4%) = 1M rids**

  - **So if 20MB RID pool, RID processing disabled before apar, enabled after apar.**

Query Parallelism

- **IN-list index access in parallel**

  Select from A where a1 IN (-, -, -, -)

  - **V6 supports parallel IN-list index access for inner table in join**
  - **V7 support for both outer table and single table access**
  - **Typical 2 to 5 times faster possible for IN-list index access**

NOTES

- **IN-list index access is indicated by AccessType 'N' in Plan table**
- **Number of degrees determined by smaller of**
  - **Number of values in IN-list or number of partitions if I/O-bound query**
  - **Number of values in IN-list or number of processors if CPU-bound query**

Query Parallelism - continued

- Disable unnecessary dynamic and list prefetch in a query parallelism when VPSEQT=0 (V6 PQ41128 11/00)
  - Just like in sequential execution
  - Can reduce DBM1 SRB time and class 24 latch contention, especially in many concurrent prefetch
  - V6 eliminated unnecessary prefetch scheduling for sequential, but not dynamic or list, prefetch, independent of VPSEQT value.

29

NOTES

- **VPSEQT = Maximum percentage of Virtual Pool buffers that can be used for sequential, dynamic, and list prefetch (default = 80%)**
  - **When VPSEQT is set to 0, prefetch is usually disabled. This can result in up to 10% reduction in query CPU time when the requested pages are all in buffer pool and possibly even bigger improvement when high class 24 latch contention due to many concurrent prefetch.**
  - **Under query parallelism, prefetch was scheduled even when VPSEQT set to 0 prior to this enhancement.**

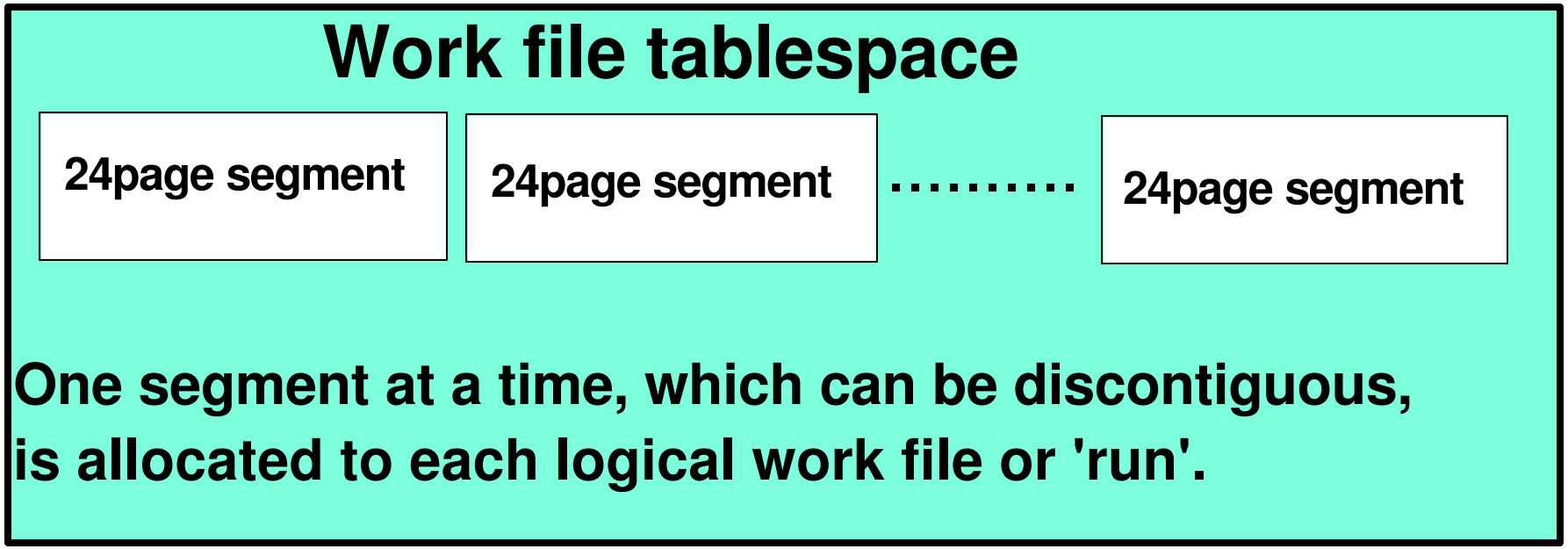
30

Query Parallelism - continued

- Work file prefetch quantity of 32 in parallel query brought down to 8, just like sequential query
  - Can reduce unnecessary page reads, especially when many concurrent work file activities, such as concurrent sort and Star Join
  - 10 to 20% reduction in the number of pages read for a single query in one measurement
  - Bigger reduction expected for multiple concurrent work file activities

NOTES

- A work file consists of a set of segments, each containing 24 pages. So reading 32 pages can cause pages belonging to a segment in a different logical work file to be read unnecessarily.



Query Parallelism - continued

- **Default maximum parallel degree reduction from 254 to [10*number of CPUs] (V6 PQ47914 6/01)**
  - **To minimize excessive use of DBM1 virtual storage, buffers, CPU time, etc. from over-parallelism**
  - **Overriding the default may be beneficial for very I/O-bound long-running query**

33

NOTES

- **Recommendation to have number of physical work files at least equal to the maximum parallel degree**

More Efficient Bind of Many-Table-Join

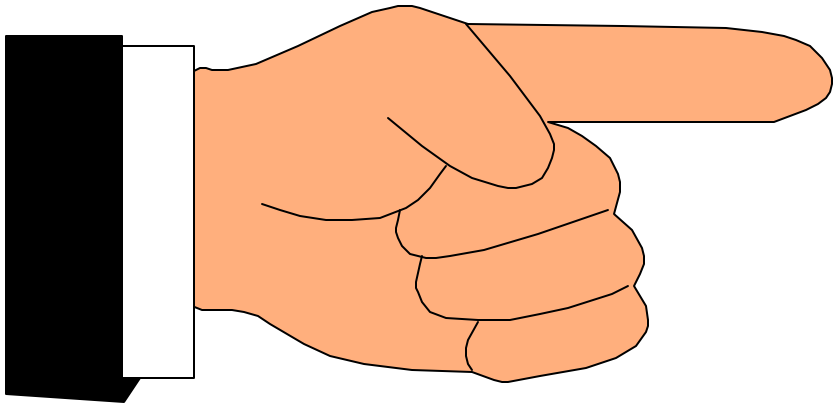
- Bind CPU and DBM1 storage reduction
  - A new algorithm enabled for 9 or more tables in join
  - Up to 6 times reduction in DBM1 storage and CPU time observed for complex queries with 15 to 20 tables joined

35

NOTES

- **Bind storage reduction extended to Outer Join queries via V7 PQ48306 5/01**
  - **Up to 25 times reduction in virtual storage usage for outer join queries**
  - **Up to 2 times further reduction in virtual storage usage for inner join queries**

Miscellaneous Topics



NOTES

OUTLINE

- V5 to V6 performance regression
- Fetch First N Rows
- Scrollable Cursor Performance Considerations
- Catalog migration (CATMAINT)

38

V5 to V6 performance regression

- Stored procedure call with stored procedure name in host variable results in incremental bind at runtime
- Short-running SQL with degree any
  - especially those repeatedly executed in a loop

39

NOTES

- **Stored Procedure regression fix**
  - **Reduce incremental bind overhead by saving the executable form of the bound CALL statement (V6 PQ35329 3/00)**
  - **Reduce excessive class28 (stored procedure) and class32 (storage pool) latch contention when many active threads are issuing frequent stored procedure calls, also minimize storage get/free in incremental bind (V6 PQ38744 10/00)**

Performance regression - continued

- Query parallelism threshold SPRMPTH zparm with nonzero default to avoid parallelism overhead for short-running queries (V6 PQ45820 3/01)
  - More important in each new release/version in trying to avoid performance regression as more queries become enabled for parallelism
  - Can dramatically reduce high 'Other service task wait' in class 3 accounting when degree any

41

NOTES

- **Some examples of V5 to V6 parallelism enablement**
  - **Non partitioned tablespace**
  - **Outer join**
  - **Non partitioning index**
- **Service task wait breakdown in V6 accounting class 3**
  - **To quickly identify those with significant wait time**
  - **'Other service task wait' is supposed to be small**

Fetch First N Rows

- Limits the number of rows fetched to avoid fetching unwanted rows
- Singleton Select (or SELECT INTO) can be used with Fetch First 1 Row even if multiple rows qualify
  - Avoids -811 SQLCODE
  - Up to 30% CPU time saving compared to Open Cursor/Fetch/Close
  - Bigger improvement possible for CICS attach

43

NOTES

- Can result in a significant elapsed time reduction in a distributed environment by reducing the message traffic
- Existence check without the use of EXISTS subquery
  - SELECT FROM T WHERE COLUMN = ...
    - Fetches all qualifying rows
    - Adding FETCH FIRST 1 ROW ONLY terminates a query execution as soon as the first qualifying row is returned

Fetch First N Rows - continued

- Access path selection (V7 PQ49458)
  - Use M in OPTIMIZE FOR M ROWS if specified
  - Else use N in FETCH FIRST N ROWS if specified
  - Example of M=1, N=50, and 100 qualifying rows
    - Optimizer picks the best access path for 1 row fetch. 50 rows are fetched
  - Example of N=10 and 5 qualifying rows
    - Optimizer picks the best access path for 10 row fetch. 5 rows are fetched

NOTES

- **Example of 25 (estimated) qualifying rows with no N or M specified**
  - **Optimizer picks the best access path for 25 row fetch. All qualifying rows are fetched.**

46

Backward Dynamic Prefetch

- Introduced with scrollable cursor support but applies to all tables and indexes
- Sequential detection enabled for both forward and backward direction
  - Forward only in V6
  - With or without scrollable cursor

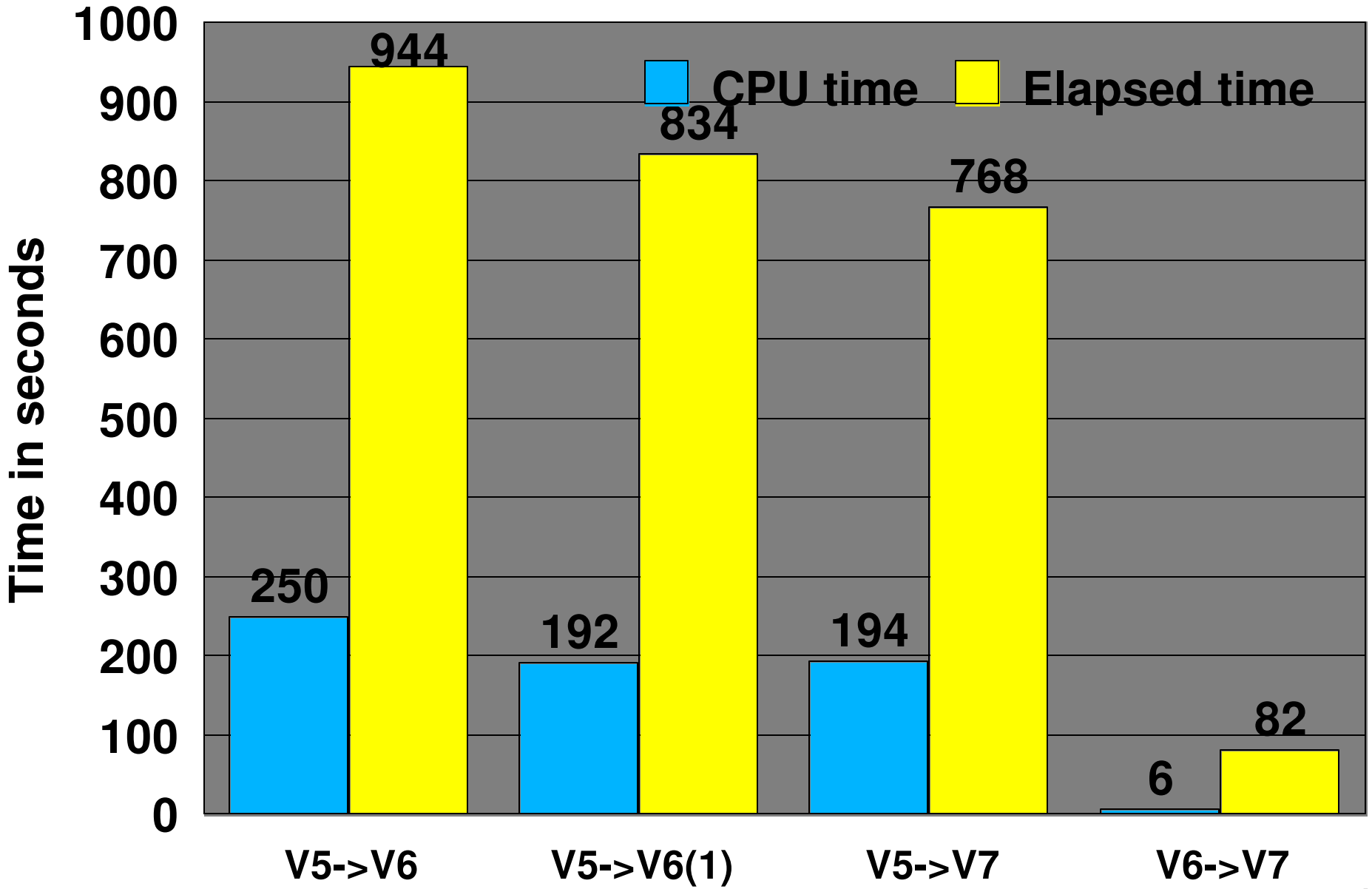
NOTES

- **Example**
  - **Do loop of Select * from T where c1=host-variable resulting in 1 million data pages accessed backward.**

V6	**1 million synchronous read I/O's**
V7	**31250 dynamic prefetch I/O's reading 1 million pages asynchronously**

48

Catalog Migration Utility (CATMAINT)



49

NOTES

- (1) With V6 PQ38035 2/01 and PQ44985 4/01
- For 5GB catalog with 7million rows
- Recommended buffer pool size for high performance catalog migration
  - Catalog buffer pool = 10000 buffers
  - Workfile buffer pool = 10000 in V5->V6, 1000 in V6->V7
- More than 10x elapsed time and 40x CPU time reduction in V6->V7 compared to V5->V6
  - 7x reduction in catalog Getpage
  - 21000x reduction in workfile Getpage as DB2 sort practically eliminated

References

- Redbooks at www.redbooks.ibm.com
  - DB2 for z/OS and OS/390 V7 Performance Topics SG24-6129
  - DB2 UDB Server for OS/390 V6 Technical Update SG24-6108
  - DB2 UDB for OS/390 V6 Performance Topics SG24-5351
- DB2 UDB for OS/390 Administration Guide, Performance Monitoring and Tuning Section, SC26-9003 for V6, SC26-9931 for V7
- More information on DB2 UDB for OS/390 at www.ibm.com/db2

NOTES

- **A comprehensive set of performance tuning recommendations for JDBC/SQLJ in V7 Performance Redbook**

52