DB2 for z/OS V8 New Function Performance

ON DEMAND BUSINESS™

Akira Shibamiya
Silicon Valley Laboratory

May 2007

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**

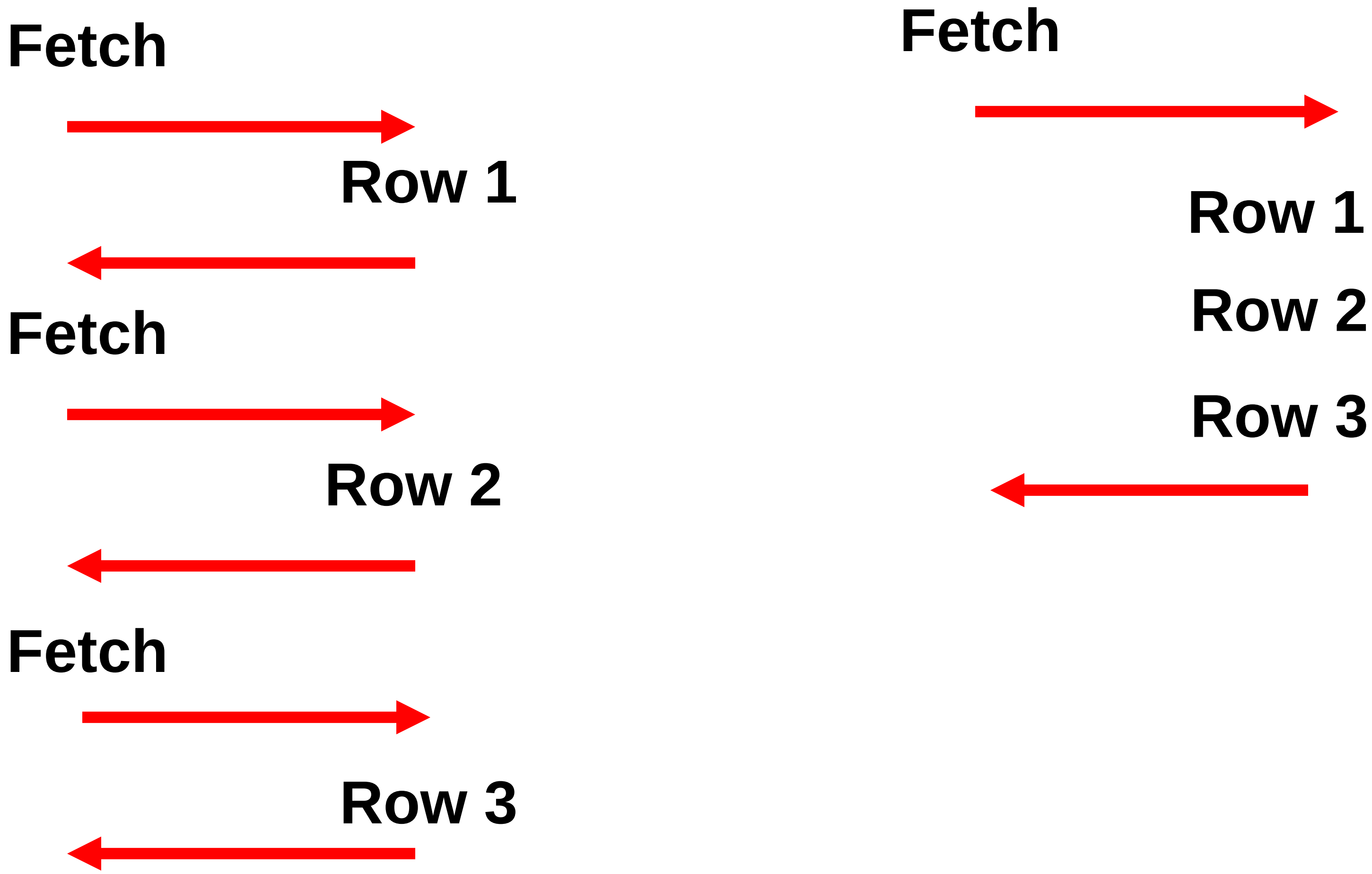
- **The information contained in this presentation has not been submitted to any formal IBM review and is distributed on an "As Is" basis without any warranty either expressed or implied. The use of this information is a customer responsibility.**

Agenda

- **Multi-row Fetch and Insert in local and distributed**
- **Index enhancements**
- **Query performance enhancements**
- **Miscellaneous enhancements**
- **Synergy with Processor and I/O Hardware**

Multi-Row Operation

Single row fetch versus Multi row fetch



Fetch

Row 1

Fetch

Row 2

Fetch

Row 3

Fetch

Row 1

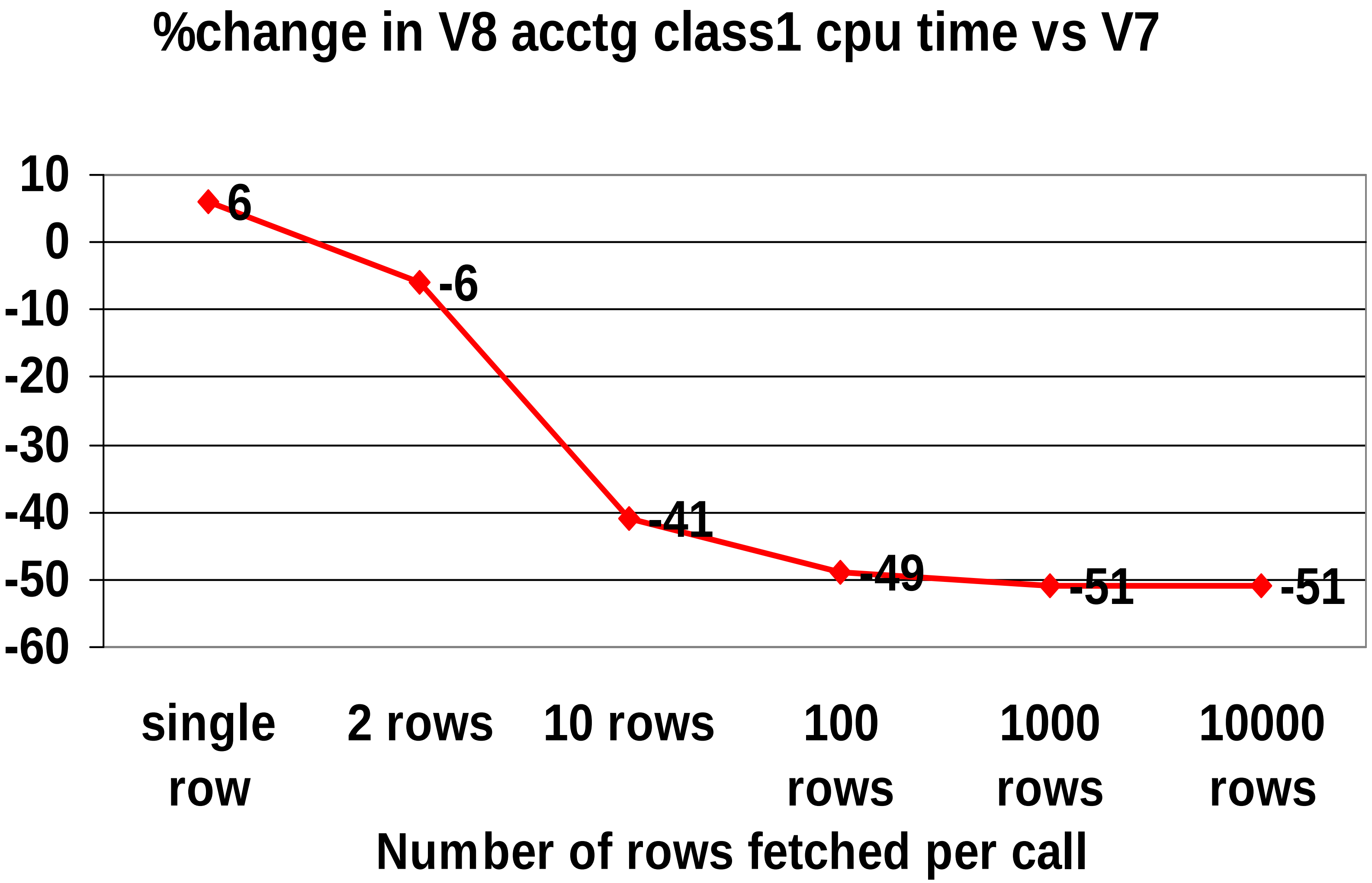
Row 2

Row 3

Multi-row Fetch

- FETCH NEXT ROWSET FROM cursor FOR N ROWS INTO hva1,hva2,hva3
- Up to 50% cpu time reduction by avoiding API (Application Program Interface) overhead for each row fetch
  - Lower %improvement if more columns and/or fewer rows fetched per call
  - Higher %improvement if
    - acctg class 2 on
    - No thread-safe option in Open Transaction Environment for DB2 with CICS/TS 2.2

20column 100000row Fetch CPU Time



Notes

- The graph clearly shows that the percentage improvement goes up as more rows are fetched per Fetch call.
  - With 1 row fetch, V8 cpu is 6% higher than V7.
  - However, with 2 row fetch, V8 becomes faster by 6%.
  - Beyond 100 rows, about 50% improvement continues.
  - Similarly for elapsed time and class 2 cpu time.
- The measurement shown is for a very simple fetch via tablespace scan fetching 20 columns
  - Less %improvement for more complex Fetch involving join, sort, index access, more than 20 column fetch
  - More %improvement for less than 20 column fetch

Multi-row Insert

- Insert into Table for N Rows Values (:hva1,:hva2,…)
- Up to 40% cpu time reduction by avoiding API overhead for each Insert call
  - %improvement lower if more indexes, more columns, and/or fewer rows inserted per call
- ATOMIC (default) is better from performance viewpoint as multiple SAVEPOINT log records can be avoided
- Similarly for multi-row cursor Update/Delete

Notes

- Hva = host variable array
- API = Application Program Interface overhead for each SQL call
- Atomic (default) specifies that if insert of any row fails, then all changes made are undone.
  - Atomic requires one SAVEPOINT, contributing less than 5% overhead with 2 row insert and completely negligible for many row insert.
  - Atomic always for Update and Delete
- Non Atomic: V8 PK30906 11/06 SAVEPOINT log record written for each row inserted to keep successfully inserted rows
- Up to 32767 rows can be inserted in 1 call
- Support for C, C++, Cobol, PL/I, Assembler, Java and T4 driver
  - For both static and dynamic SQL calls

Multi-row in Distributed

- Dramatic reduction in network traffic and response time possible
  - By avoiding message send/receive for each row in
    - Non read-only Fetch
    - Update and/or Delete with cursor
    - Insert
  - Up to 8 times faster response time and 4 times cpu time reduction

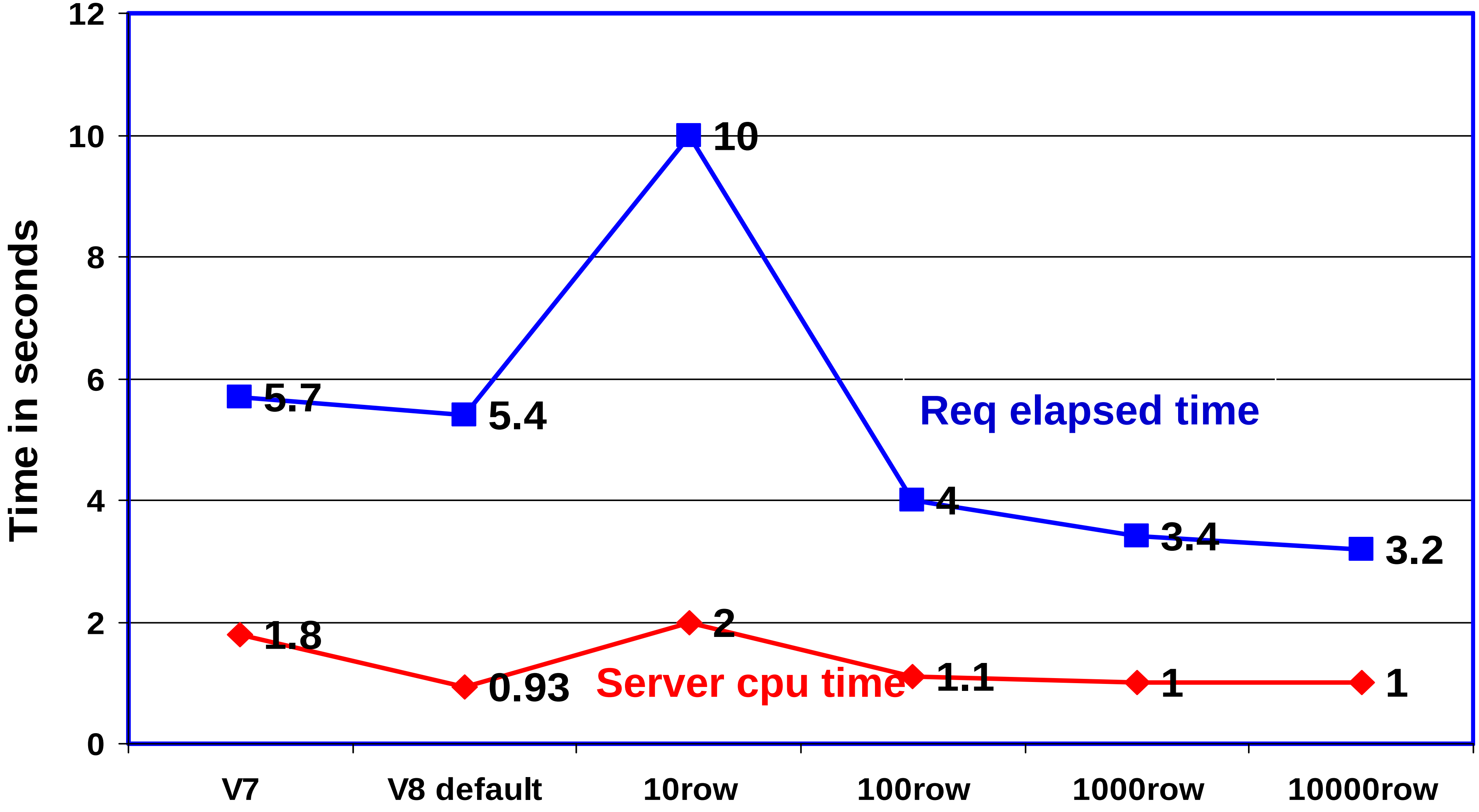
Notes

- If Fetch with read-only or [CD NO and ambiguous cursor], multi-row Fetch is automatically enabled in DRDA, resulting in
  - CPU time saving of up to 50%
  - But less difference in message traffic compared to V7 with Block Fetch
    - Note that multi-row Fetch is unblocked; ie if 10 Fetch calls are issued for 10 rows each, 10 blocks are sent, compared to only 1 block in implicit multi-row Fetch
    - V7 PQ49458 8/03
      - OPTIMIZE FOR access path and network blocking
      - FETCH FIRST for access path but not network blocking when no OPTIMIZE FOR clause

Multi-row Fetch Host-to-Host

- DB2 for z/OS V8 acting as a DRDA application server accessed by another DB2 for z/OS V8 acting as a DRDA application requestor
- Fetching 100,000 20 column rows
- V8 default = implicit multi-row Fetch
  - -5% requestor elapsed time because of V7 block fetch
  - -47% server cpu time
- V8 explicit multi-row Fetch
  - Up to -43% requestor elapsed time due to blksize increase from 32KB to 10MB max
  - Up to -43% server cpu time also
- % impact depends on network performance, #columns fetched, #rows fetched in one Fetch call, rowsize, complexity of SQL call

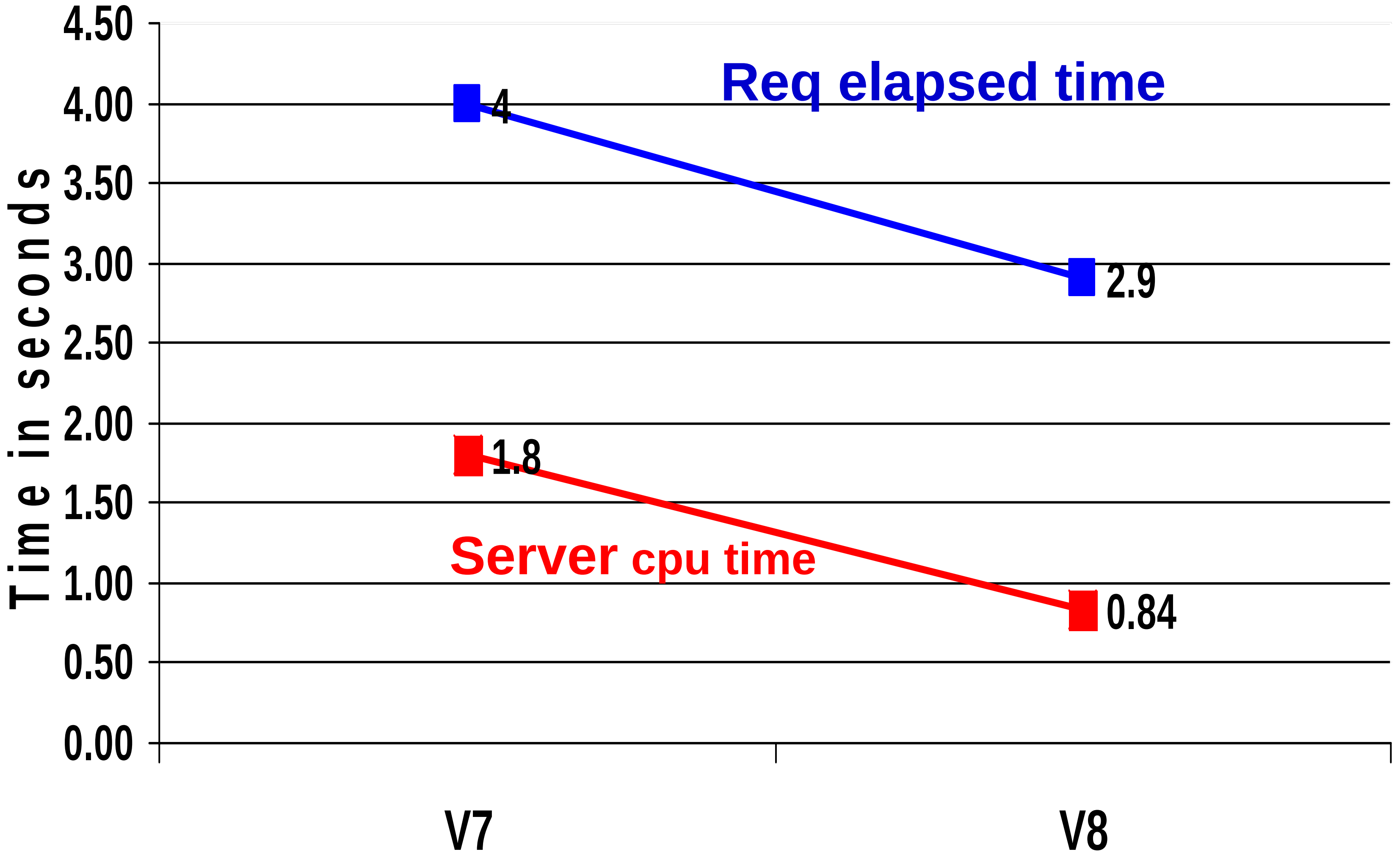
Host-to-Host Fetch



Multi-row Fetch Workstation-to-Host

- DB2 for z/OS V8 acting as a DRDA application server, accessed from a DB2 Connect Client running on Linux/Unix/Windows as a DRDA application requestor
- Fetching 100,000 20-column rows
- V8 default = implicit multi-row Fetch
  - -26% client elapsed time
  - -53% server cpu time
  - Up to 64KB blksize
    - 64K blk via “DB2 update database manager configuration using rqrioblk 65535”, default=32K
  - No explicit multi-row Fetch supported by DB2 Connect V8 clients

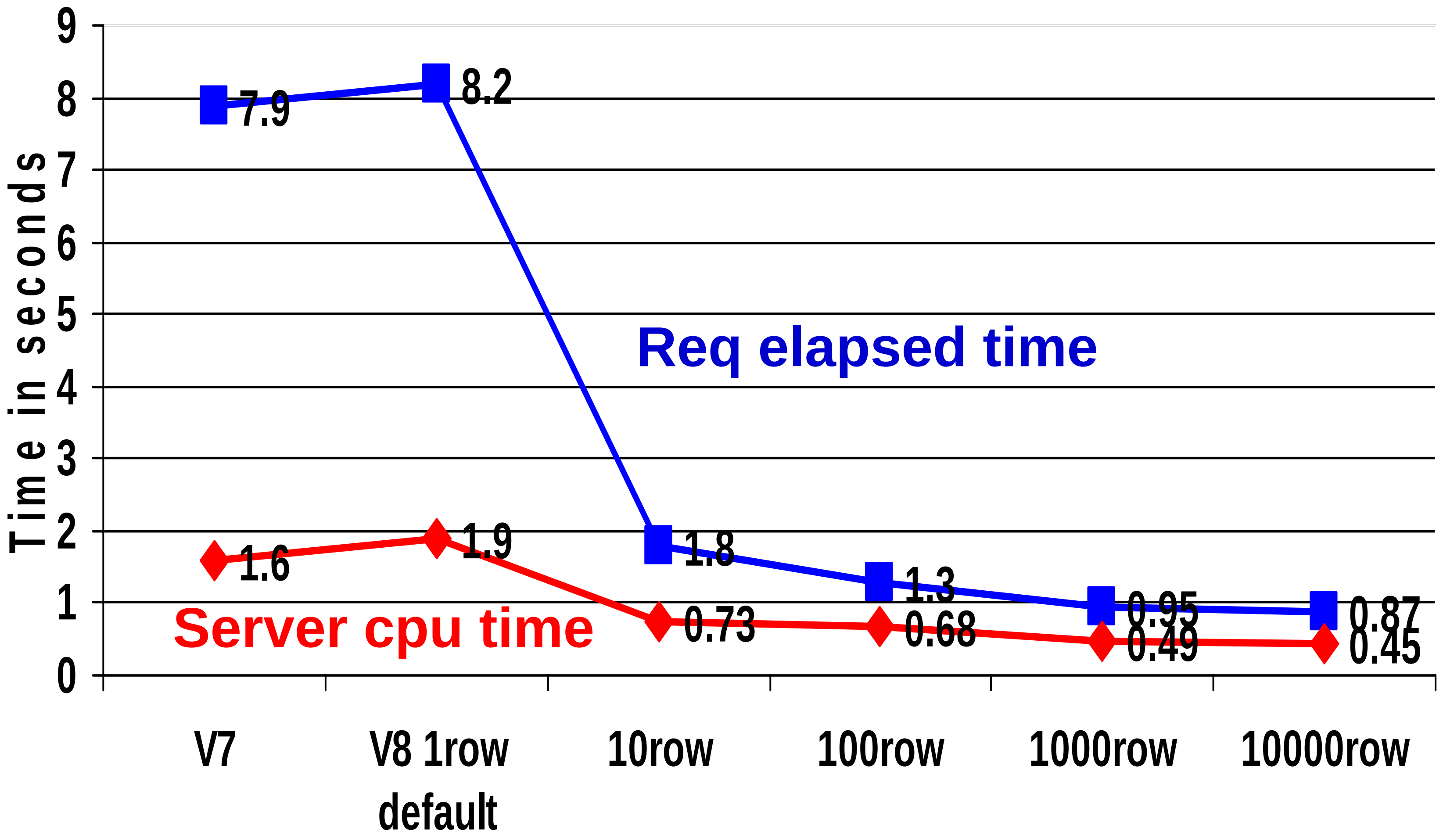
Workstation-to-Host Fetch



Multi-row Insert Host-to-Host

- DB2 for z/OS V8 acting as a DRDA application server accessed by another DB2 for z/OS V8 acting as a DRDA application requestor
- Total of 10000 20-column rows inserted
  - -77% elapsed time and -54% cpu time for 10row/Insert call
  - -83% elapsed time and -58% cpu time for 100row/Insert call
- % impact depends on network performance, #indexes, #columns, rowsize, #rows inserted per Insert call

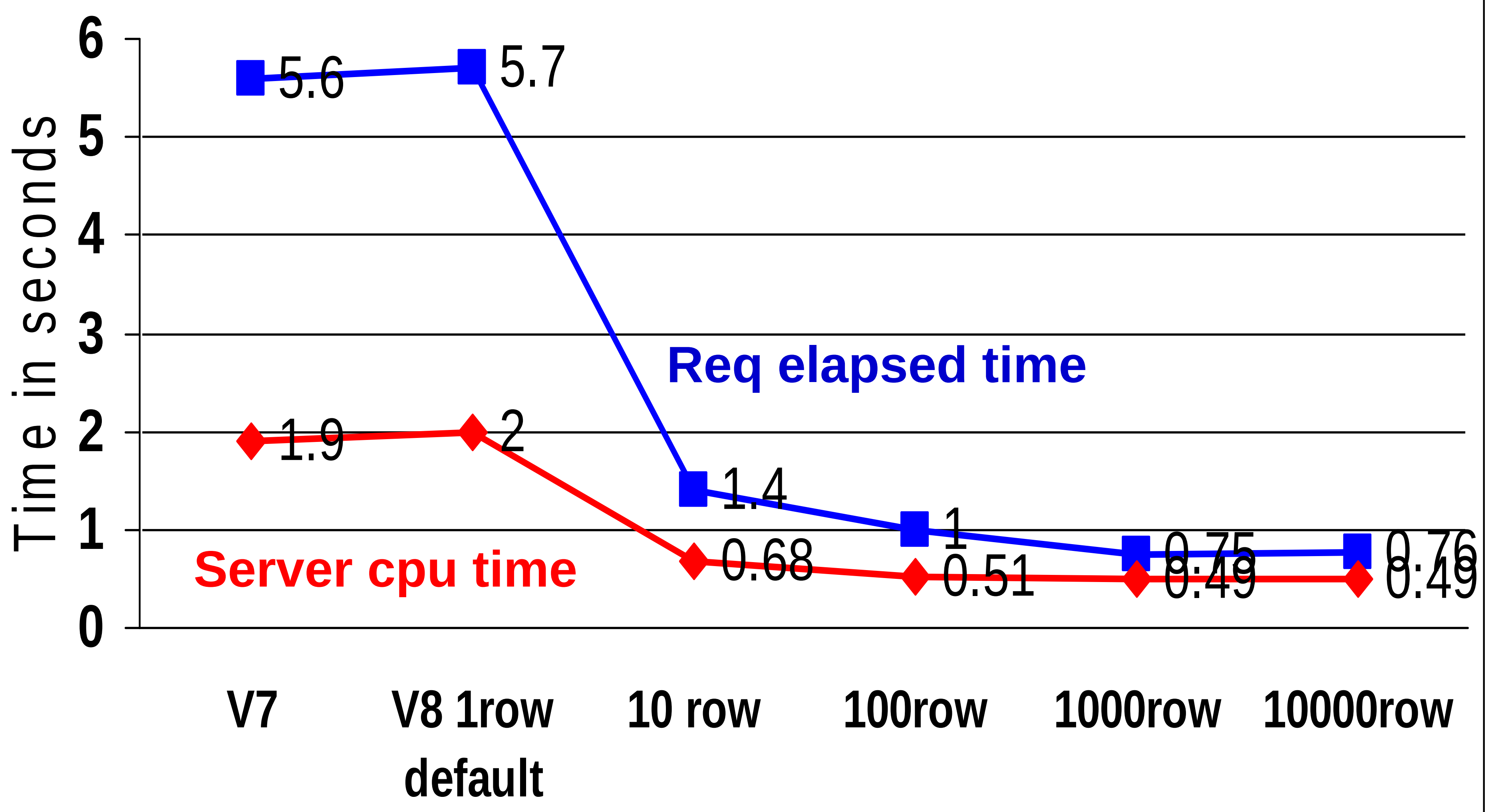
Host-to-Host Insert



Multi-row Insert Workstation-to-Host

- DB2 for z/OS V8 acting as a DRDA application server, accessed from a DB2 Connect Client running on Linux/Unix/Windows as a DRDA application requestor
- 10000 20-column rows inserted
- 10row/Insert call
  - -76% elapsed time and -63% cpu time compared to V7
  - -30% elapsed time and -38% cpu time compared to V7 array input
- 100row/Insert call
  - -82% elapsed time and -63% cpu time compared to V7
  - -33%elapsed time and -49% cpu time compared to V7 array input

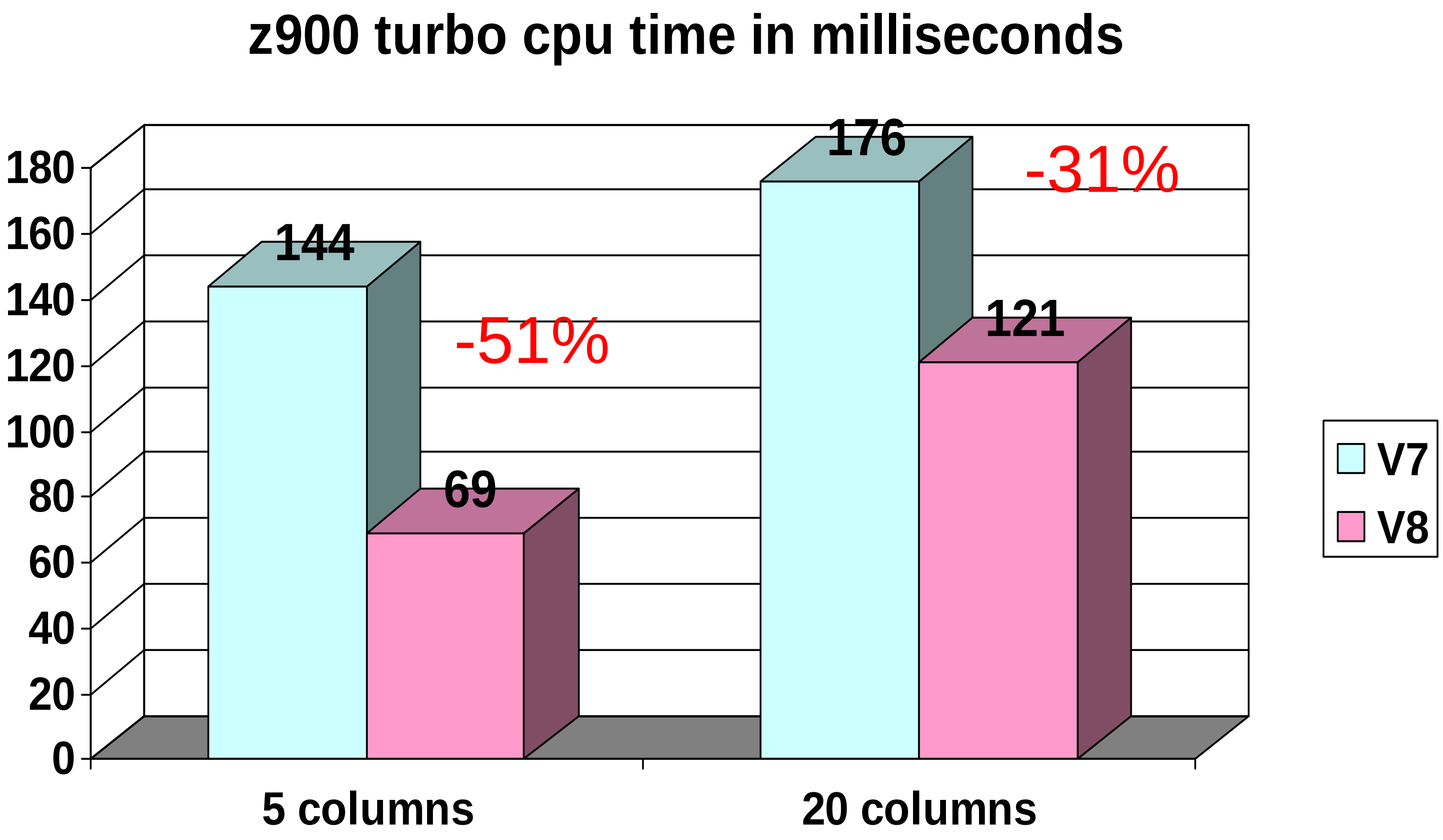
Workstation-to-Host Insert without array input



Automatic exploitation of multi-row Operation

- **DRDA as described already**
- **DSNTEP4=DSNTEP2 with automatic multi-row fetch**
  - **Up to 35% cpu reduction in fetching 10000 rows with 5 and 20 columns**
- **DSNTIAUL (sample Unload utility)**
  - **Up to 50% cpu reduction in fetching 10000 rows with 5 and 20 columns**
- **QMF multi-row fetch and insert V8 PQ99482 9/05**

DSNTIAUL fetching 10000 rows with 5 and 20 columns



Index Enhancements in V8

- **Biggest set of index enhancements in DB2 history since V4 when type 2 index was introduced**

Update of partitioning key columns

- When an update causes a row to be moved to another partition, timeouts can occur as the range of affected data and index partitions and all NPIs are drained before update.
- This is one unexpected surprise for some customers migrating from a segmented tablespace to a partitioned tablespace.
- This problem is eliminated in V8 as there is no more drain lock.

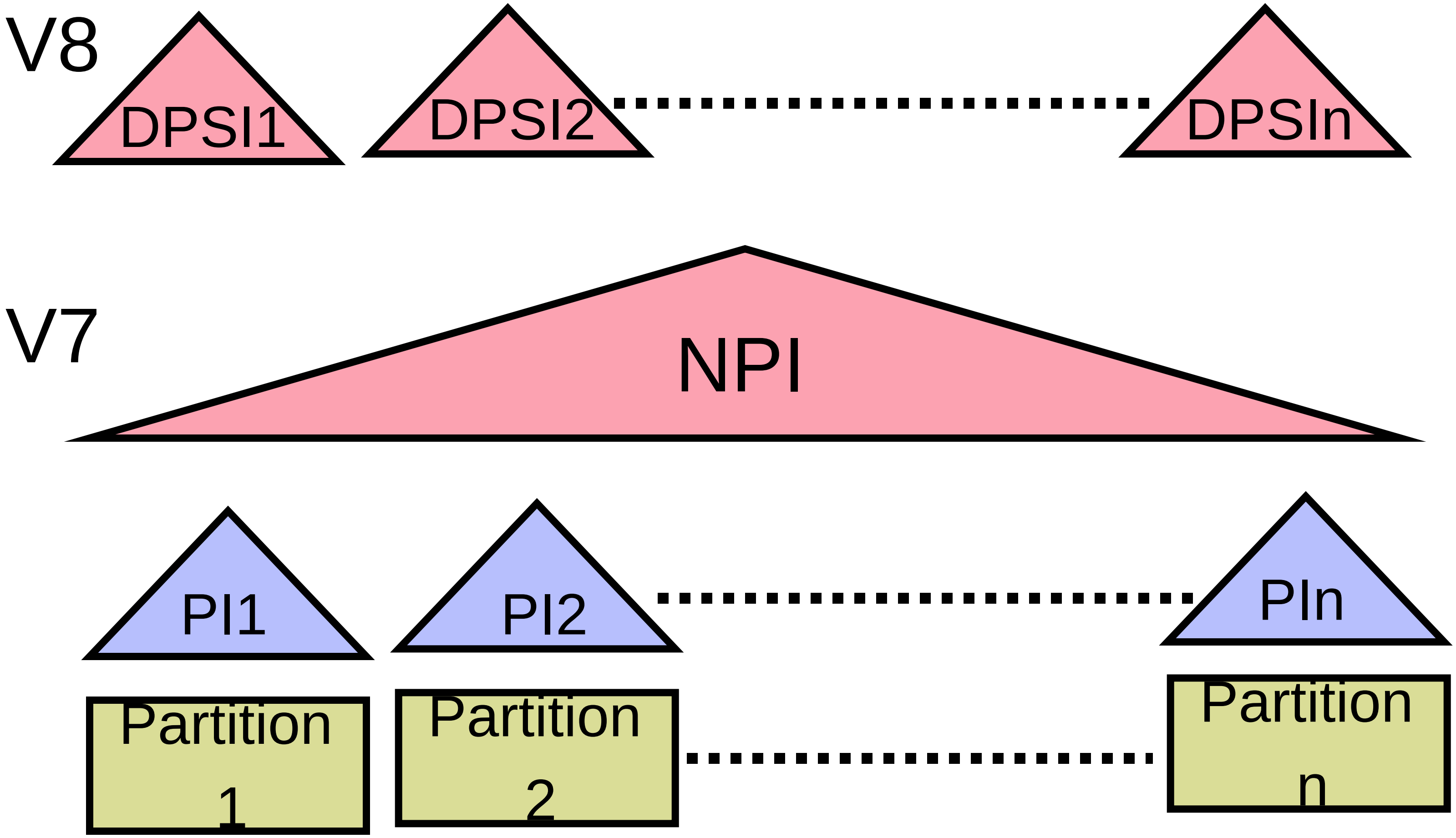
Variable length index key

- VARCHAR index key no longer needs to be padded to maximum
  - V7: Always padded to maximum length
  - V8: Option of either padded or not
  - Especially useful for a large VARCHAR, eg DB2 catalog with 128byte VARCHARs
  - In such a case, more index entries per index page, resulting in fewer index pages and index levels, and less DASD space and buffer pool
- Further enablement of index-only access
  - SELECT varchar1 FROM table WHERE char1=x
    - With index on char1.varchar1

Notes

- Rule-of-Thumb: If <18byte varchar columns on average, use padded key, because of
  - Extra cpu time for non-padded key processing
  - 2 extra bytes per varchar column in each non-padded key
  - Cost depends on # and size of VC columns in index key
- DEFIXPD zparm with default of
  - PADDED in migrating to V8
  - NOT PADDED in new V8 install
  - Meaningful if variable-length columns; else padded index always
- Maximum key length increased to 2000 from 255
  - Partition key is limited to 255
- CHAR(8) or VARCHAR(18) columns in catalog changed to VARCHAR(128) to support long names

Parallel Partition Load/Reorg/Rebuild with DPSI



Notes

- PI = Partitioning Index
  - One data set per partition
  - Example: unique ACCOUNT#
- DPSI = Data Partitioned Secondary Index
  - One data set per partition
  - Example: non unique CUSTNAME
- NPI = Non Partitioning Index
  - One, or multiple if PIECESIZE, data set(s) per tablespace
  - Possible contentions by concurrent partition utilities
- A single table may have a mix of NPI and DPSI

DPSI - continued

- Much faster partition-level operation when multiple indexes present
  - Up to N times faster where N is the number of partitions
    - Avoids accessing entire index in single partition utility
    - Avoids contention and insert mode processing in parallel partition utilities such as Load

Notes

- Additional benefits
  - Reduces data sharing overhead if partition affinity by member
    - Reduced GBP dependency
  - Avoids Online Reorg Build2 phase, invoked when partition utility with NPI present
    - Build2 typically results in the longest period of unavailability for selected partitions and logical partitions of NPIs during Online Reorg

DPSI Usage Considerations

- Not for unique index
  - Insert of each row must check all DPSI partitions to make sure it is unique, if unique index is to be supported
- Query performance impact
  - Depending on PI predicates available, some or all DPSI partitions may have to be scanned because the same DPSI key value may be in multiple partitions
    - Example: SELECT FROM TABLE WHERE CUSTNAME=x
      - Unique ACCOUNT# as PI key
      - Non-unique CUSTNAME as DPSI key
    - More %overhead with fewer rows scanned and/or more partitions scanned

DPSI Usage Considerations - continued

- ORDER BY or DISTINCT on DPSI column may require extra processing
  - Example: SELECT FROM table WHERE CUSTNAME BETWEEN x AND y ORDER BY CUSTNAME

- Also some difference in index-only access, index lookaside, and parallel query

- Trade-off between partition tablespace utility and some query performance

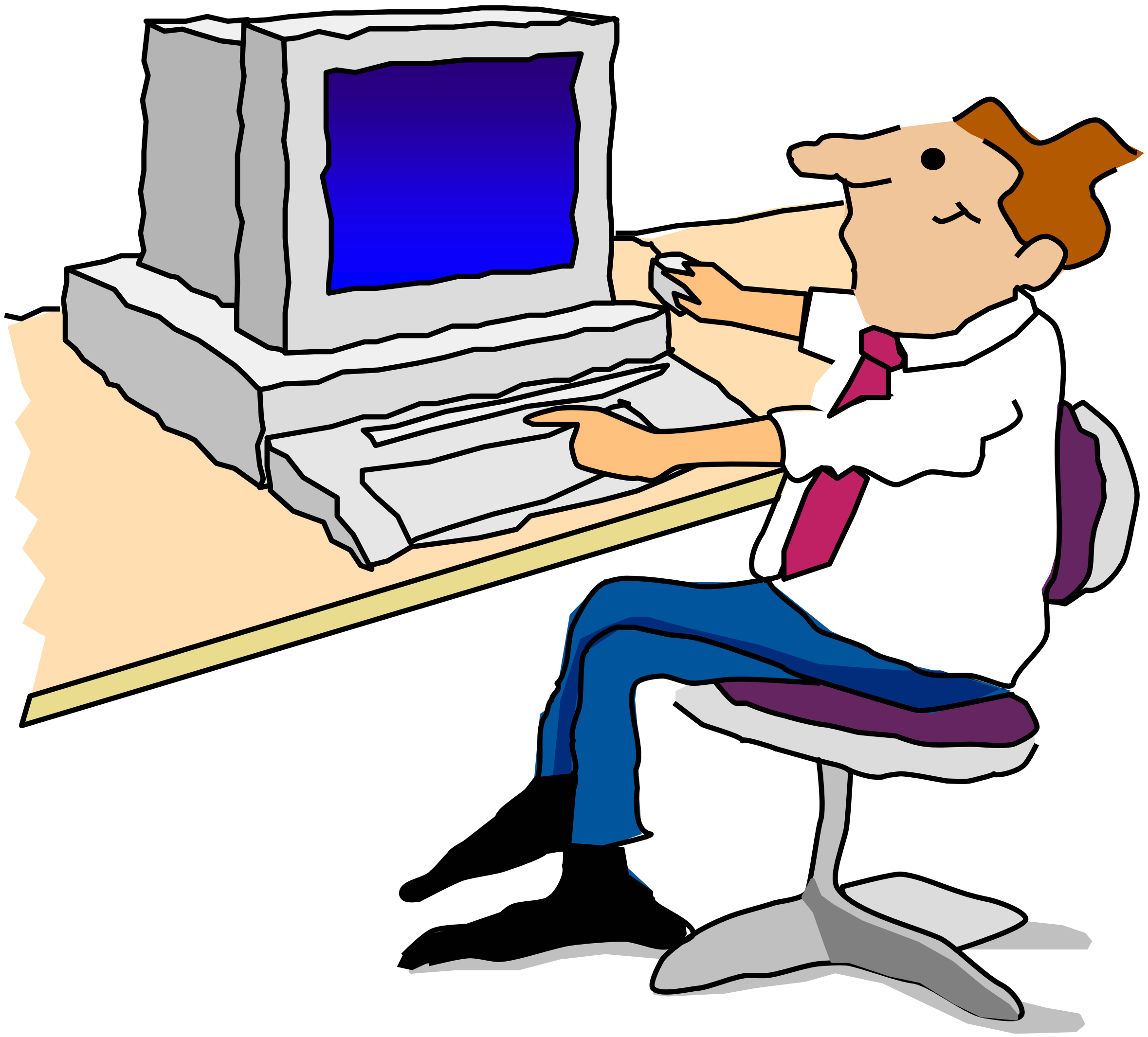
Reading Index Backward

- Read multiple rows via index backward to avoid sort
  - SELECT FROM table … ORDER BY c1 DESCENDING
    - With an ascending index on c1
    - Dynamic prefetch of index to make backward scan almost as efficient as forward scan
  - Supported with or without scrollable cursor

Notes

- Other index related enhancements
  - Partitioned tablespace without index
    - Useful when PI created just for partitioning purpose and not for predicates
    - CPU and i/o reduction in Insert, Update, Delete
  - When no index is defined as clustering, the first created index is made clustering, making queries which reference this index potentially more efficient.
    - Compatible with insert behavior
  - Clustering index separate from partitioning index

Query Performance



NOTES

- Materialized query table
- Distribution statistics on non-indexed columns
- Star join
- More indexable predicates
- Non-correlated EXISTS subquery
- Prepare/Bind performance
- Others

Materialized Query Table

- Pre-selected and/or pre-computed results from large table(s) saved in much smaller MQT for fast subsequent access
  - ►Example: Avg Income, Height, NetAssetValue, ... of 300 million US residents grouped by 50 states
  - ►10 to 1000 times faster possible for some queries
- Automatic query rewrite for dynamic SQL to take advantage of relevant MQT
  - ►Summary table can be used directly by both static and dynamic SQL

NOTES

- MQT = Materialized Query Table, sometimes called Automatic Summary Table
- Existing tables can be registered as MQT via ALTER TABLE
- If MQT is used, Plan Table TABLE_TYPE='M'

Materialized Query Table - continued

- **Large MQT performance considerations for maximum exploitation**
  - **Use segmented tablespace because of almost instantaneous mass delete in REFRESH TABLE**
    - **REFRESH TABLE deletes current data and then inserts new data. MQT is locked out during this process.**
  - **Runstats after REFRESH for good access path selection**
    - **Especially useful in join involving MQT**

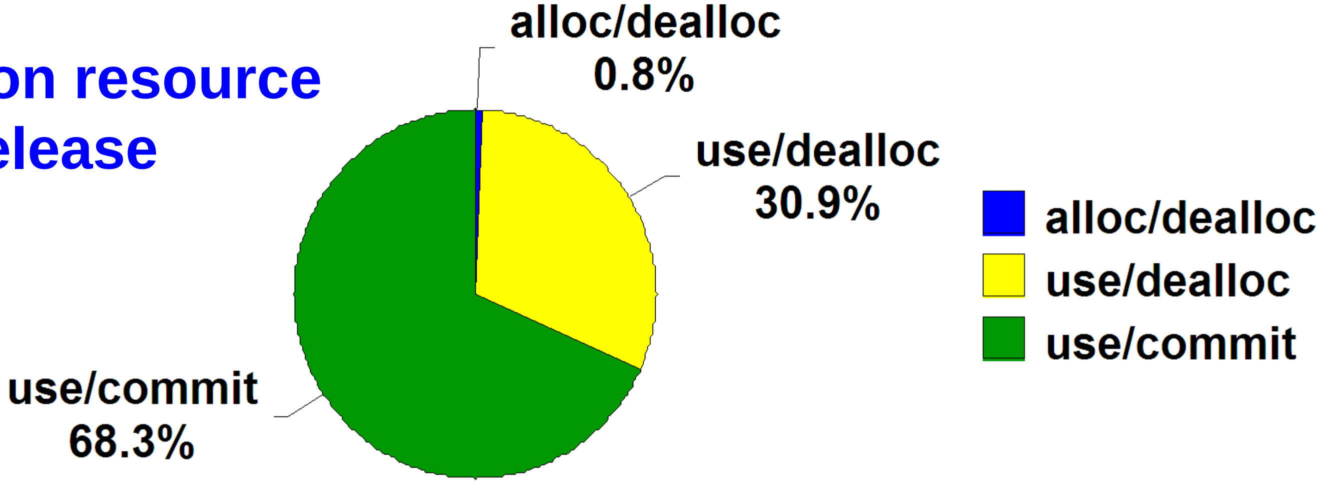
NOTES

- System-maintained MQT can be used just like any other table except for some restrictions
  - No Insert, Update, or Delete allowed
- User-maintained MQT supports both REFRESH TABLE and Insert, Update, Delete

Distribution stats on single and multiple columns

- Top N highest, and/or lowest, frequency of values and cardinality

Bind option resource acquire/release example



SELECT FROM A, SYSIBM.SYSPLAN B WHERE B.ACQUIRE='A' AND B.RELEASE='D' ...

Better join sequence from more precise filter factor estimation of combined predicates

NOTES

- DSTATS (Distribution stats for DB2 for OS/390)
  - A down-loadable tool available prior to V8
  - http: //www-1.ibm.com/support/docview.wss?uid=swg24001598

- Fixes the most typical access path selection problems encountered today

  - Optimizer unable to come up with the best access path because of a lack of distribution stats on non-indexed columns which are referenced in predicates
    - ➔Can cause performance degradation due to access path change in a new release or after access-path-related maintenance

Star Join Performance

- Use of sparse index on work file to reduce workfile scan
  - Sparse index up to 240KB in memory
  - Binary search of index followed by sequential scan of workfile subset with nest loop rather than merge join
  - 2 to 5 times faster for some star join queries
  - Also in V7 PQ61458 6/02
- Use of memory above 2GB rather than workfile when available
- Other star join enhancements

NOTES

- Normal index: 1 index entry for each row
- Sparse index: 1 index entry for every N rows

- Sparse index first used in DB2 V4
  - 16KB sparse index in memory for non-correlated IN subquery for up to 100 times performance improvement

- Accesstype='T' in Plan Table for sparse index access or in-memory work file

- Star join in-memory work file can also prevent performance disruption on other threads using work files, such as sort, merge join, trigger, created temp table, non-correlated subquery, table UDF, outer join, materialization of nested table expression and/or view, ...

More Indexable Predicates

- **For column comp-op value with unlike type or length**
  - ►**4byte char column = 8byte host variable**
  - ►**Integer column = decimal host variable**

  - ►**Stage 2 and non indexable in V7**
  - ►**Stage 1 and indexable in V8**
    - **So index on char or integer column here can be used in V8 but not in V7**

  - ►**Also useful where a programming language does not support all SQL data types. For example,**
    - **No decimal type by C/C++ on non mainframe platform, no fixed-length char by Java**

NOTES

- Stage 1 and indexable predicate in
  - V6: Column comp-op non column expression such as SELECT FROM A WHERE a1=x+y
    - also char/varchar of different size in equi-join such as
      SELECT FROM A,B
      WHERE 10byte char a1=20byte varchar b1
  - V7: Column comp-op column expression in join such as SELECT FROM A,B WHERE a1=b1+x, if table B joined to A

- But generally only if left side column has equal or bigger size and precision
- V8 removed this restriction for both local and join predicates

Indexable Predicates - continued

- SELECT FROM Unicode table U, Ebcdic table E WHERE u1=e1 …

  - In V7, join of U and E tables not allowed
  - In V8, multiple CCSID sets per SQL statement supported
    - Useful in joining with catalog tables

  - In join of E to U, stage 1 and indexable. An index on u1 can be used.
  - If join of U to E (E is inner table), stage 1 but no indexable.

PK04107 8/05 Bidirectional indexability between unicode and ebcdic tables

Non-correlated EXISTS Subquery Improvement

- **Stop evaluating non-correlated EXISTS subquery as soon as a qualifying row is found**

  - **Prior to V8, all qualifying rows are retrieved and stored in the work file**
    Example: SELECT FROM table
    WHERE EXISTS (SELECT FROM
    SYSIBM.SYSTABLES WHERE TYPE=‘A’)
    …..

    - **In this example, a typical medium to large V7 DB2 system can contain an average of 25,000 rows in SYSTABLES and 18%, or 4500 rows, represent Type Alias**
      - **Thus 4500 rows are retrieved and stored in a work file in V7 but not V8.**

Prepare/Bind Performance

- Up to 225 (default) tables to be joined in a single FROM clause
  - ➢Increasingly more important as more complex queries can be supported in V8 without -101 SQLCODE
  - ➢Control CPU time, elapsed time, and storage usage with an internal threshold to speed up optimization process when necessary for >15 table non-star join

NOTES

- Bind option REOPT(NONE), (ALWAYS), or (ONCE)
  - V7 REOPT(VARS) and DSC result in REOPT(VARS) but no DSC
    - DSC = Dynamic Statement Caching
  - V8 NONE equivalent to NOREOPT(VARS)
  - ALWAYS equivalent to REOPT(VARS)
  - ONCE = REOPT(VARS) only once for DSC

- Improved global DSC ("short prepare")
  - V7: thread-based pool with frequent Getmain/Freemain at commit due to storage contraction
  - V8: 30 shared pools with best-fit algorithm
    - 4 to 5% improvement in transaction rate in one measurement

Other Query Performance Enhancement

- More parallel sort enablement
- Cost-based parallel sort
- Multi-column merge join parallelism
- Intelligent Visual Explain
- Numerous access path selection enhancements
- Stats Advisor for better access path selection as well as reduced chance for performance regression

NOTES

- **Examples of IN-list performance enhancement**

  - **Dynamic instead of sequential prefetch in IN-list index access to data if not contiguous (V6 PQ71925 5/03)**
  - **IN-list predicate pushdown into materialized view or table expression**
  - **Cross query block transitive closure for IN-list**
  - **Correlated subquery transformation with IN-list**

Miscellaneous Enhancements

INSERT Performance

- V8 skip option uncommitted insert for row lock
- Fast insert at end of data set by always searching forward for freespace when 0 PCTFREE and FREEPAGE for member cluster tablespace V7 PQ86037 4/04
  - V7/V8 PQ87381 8/05 to try to reuse available space while minimizing any overhead to avoid the need for Reorg to reclaim deleted space
- V8 PK05644 11/05 Preformat 1trk, 2trk, 1cyl, or 2cyl whichever is bigger
  - Useful if small Priqty and increasingly larger Secqty
- V8 PK30160 for non segmented 9/06, PK36717 for segmented tablespace 1/07 to avoid excessive conditional lock failures for page locking when many inserters to the end

- SELECT FOR READ ONLY KEEP UPDATE LOCKS, instead of FOR UPDATE, to reduce message traffic by enabling block fetch

- Allow ORDER BY in SELECT INTO statement

  - Enables SELECT INTO to get the top row based on a specified ordering

    - Example: SELECT INTO … ORDER BY
      ANNUAL_INCOME <u>FETCH FIRST 1 ROW ONLY</u>

    - More efficient than Open/Fetch/Close

- **Row-level Multi-Level Security cpu overhead roughly in the same ballpark as DB2 data compression**
  - **Requires z/OS1.5 RACF macro**
  - **Lower (<5%) for online transaction**
  - **Higher for cpu-bound sequential scan**

- **Row-level encryption tool compared to V8 column level encryption**
  - **With PQ94822 1/05 and OA08172 12/04 on z890 or z990, tool faster even when only 1 (out of 20) column encrypted**
  - **Performance characteristics similar to DB2 data compression but somewhat more expensive**

Online Schema Evolution – Highly Available Online Alter

- Instead of Drop/Create of Table, Tablespace, Index
- When some ALTER completes,
  - No existing data converted to new version format
  - Object placed in Advisory Reorg Pending (AREO) state
    - With <u>some</u> performance degradation
    - Shown in Display Database

ALTER TABLE ALTER COLUMN

- **Measured example**
  - **Char to varchar**
  - **Integer to decimal**
  - **Char(8) to char(10)**
- **10 to 30% cpu increase, depending on the number of columns processed, in Fetch because**
  - **Fast column processing disabled by ALTER**
  - **Possible conversion in Fetch**

ALTER TABLE - continued

- CPU time increased brought down to 0 to 5% after Reorg

  ➲ALTER VC to C could save 0 to 5% CPU

  - V5 Alter Varchar length, but Varchar no longer necessary to alter length

- Change allowed for longer length, precision, scale

- Alter Table Add Column
  - Supported prior to V8
  - No performance difference before and after Reorg

ALTER INDEX

1. CREATE INDEX PADDED, Reorg, Runstats
2. SELECT using padded index <base case>
3. ALTER INDEX NOT PADDED, Rebuild Index, Runstats
4. SELECT using not padded index

   a. Can be faster or slower depending on the number and size of varchar columns in index key

   - As the difference between maximum and average varchar gets bigger, NOT PADDED index becomes better, eg varchar(128) with an average of 8

   - Bigger %impact in long index sequential scan

ALTER INDEX - continued

b. Padded index more efficient for small varchar columns

c. Significant improvement possible if Alter to NOT PADDED index enables index-only access

5. ALTER PADDED, Rebuild Index, Runstats

6. SELECT using padded index again

a. No difference from the base

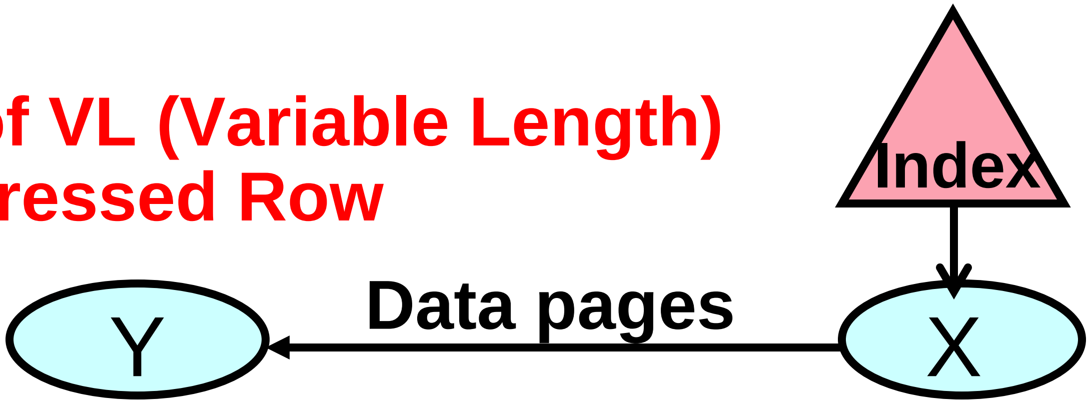
➲ No cumulative performance overhead

Performance improvement for 8K, 16K, 32K page

- **CI (Vsam Control Interval) size equals page size by default**
  - **Eg 16K instead of 4 4K CIs for 16K page**
  - **Bigger data read/write rate for 8K, 16K, 32K page**
    - **16K page measurement with 16K instead of 4K CI**
      - **+70% throughput for EF datasets**
      - **+40% for non EF (Extended Format) datasets**
  - **Enables Vsam I/O striping for 8K, 16K, 32K page**

- **Good for LOB, XML-like data, or data primarily processed sequentially**

Update of VL (Variable Length) or Compressed Row



- When an updated VL, or compressed, row can not fit in page X,
  - New row stored in a different page Y
  - Its pointer stored in page X to avoid index update
  - If updating later with small row, it can be put back on home page X, again without index update (overflow row is deleted)
- Problems
  - Potential doubling of data I/O, Getpage, and locking
  - No lock avoidance for overflow record or page in query
- Prevalent problem today as majority of data are either compressed or contain varchar columns.

VL Row Update - continued

- Reorg Tablespace will eliminate these problems
  - Consider Reorg when (Far+NearIndref)>10% (5% if data sharing) of rows in tablespace
  - Far and NearIndref in SYSTABLEPART catalog, or RTS REORGNEAR/FARINDREF, indicates the number of rows relocated from the home page due to VL row update
  - %Freespace more effective than Freepage in reducing Far and NearIndref

Overflow Locking in Read

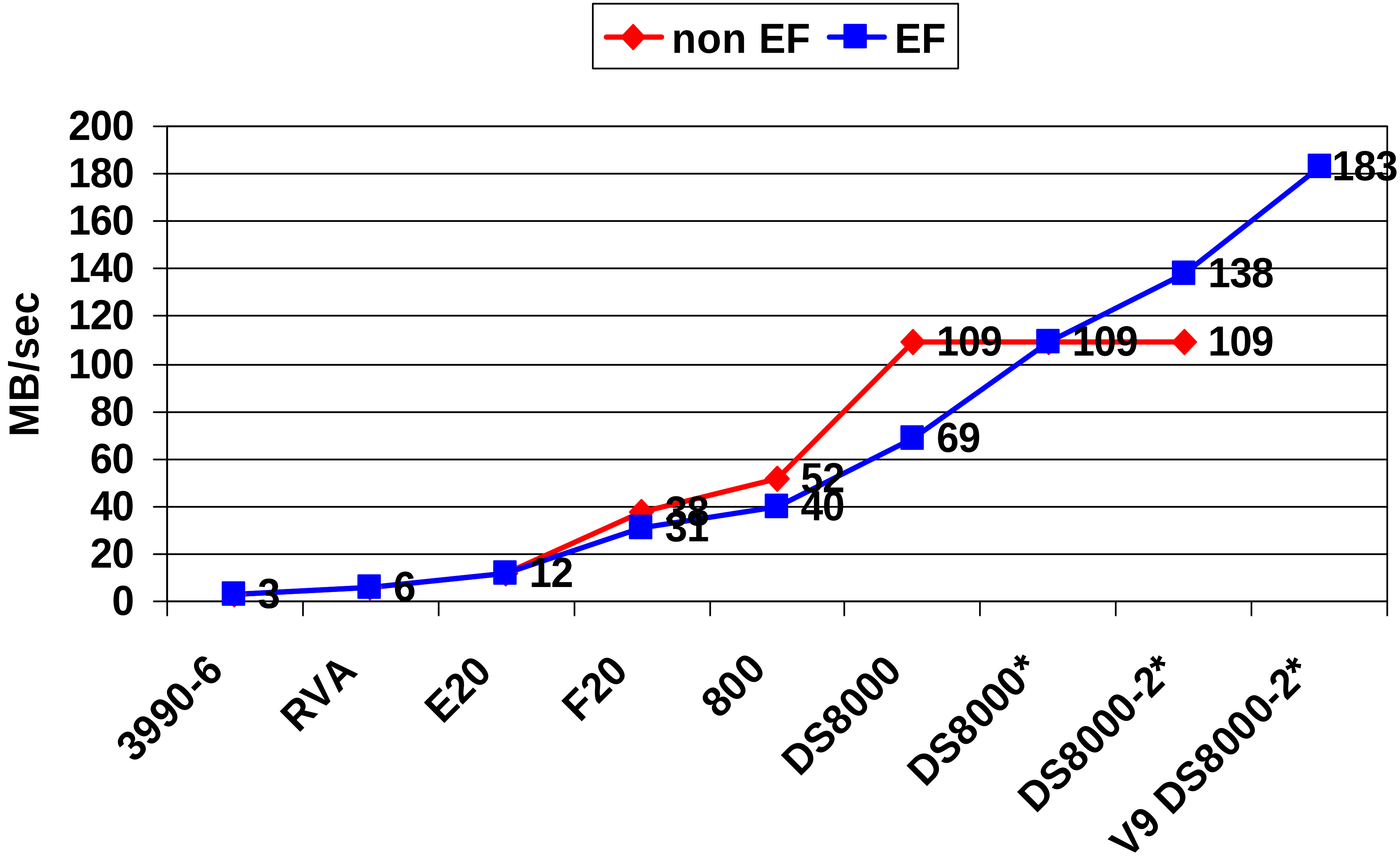
Isolation CS CD No or Yes	V7	V8
Lock/Unlock Pointer	Yes	Yes
Lock/Unlock Overflow	Yes	No

- V7: Lock on both pointer and overflow
- V8: Lock on pointer but not overflow
- All lock/unlocks here disappear after Reorg

Synergy with New I/O Hardware

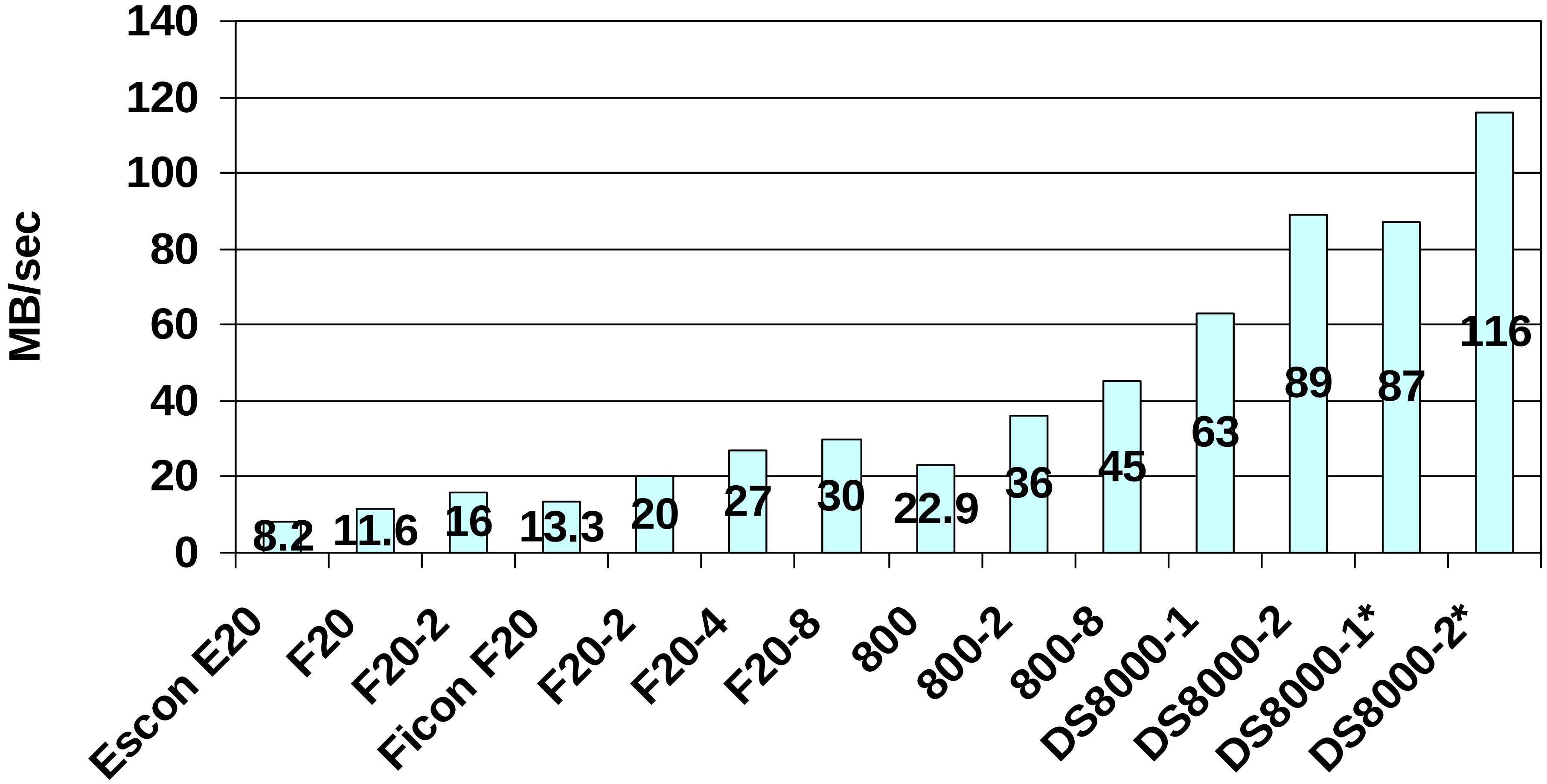
- **DS8000 with Ficon Express and MIDAW (Modified Indirect Data Address Word)**
  - **MIDAW requires z9 (2094) and z/OS1.6 OA10984 8/05, 13324/13384 9/05**
  - **Sequential read throughput**
    - **40MB/sec on ESS 800**
    - **69MB/sec with DS8000**
    - **109MB/sec with DS8000 and MIDAW**
    - **138MB/sec with 2 stripes**
  - **Bigger read, write, preformat quantity**
    - **183MB/sec in sequential read with 2 stripes**
  - **Similarly for write**
  - **Performance gap between EF(Extended Format) and nonEF datasets practically gone**

MB/sec in sequential prefetch from cache (*MIDAW)



Maximum observed rate of active log write

- First 3 use Escon channel, the rest is Ficon.
- -N indicates N i/o stripes; * MIDAW



Synergy with New Hardware

- **In addition to the raw speed improvement per engine, there are more engines (up to 54 for z9) and special performance improvement tied to a given hardware**
- **Data compression**
  - **Z900 (2064-1) up to 5 times faster than G6 turbo (9672), instead of normal 1.15 to 1.3 times, in compression and decompression**
  - **Z990(2084) 1.4 times additional speed up compared to z900 turbo in decompression**
    - **Z990 1.5 times faster than z900 turbo on average**
    - **But decompression is 1.5x1.4=2.1x faster**

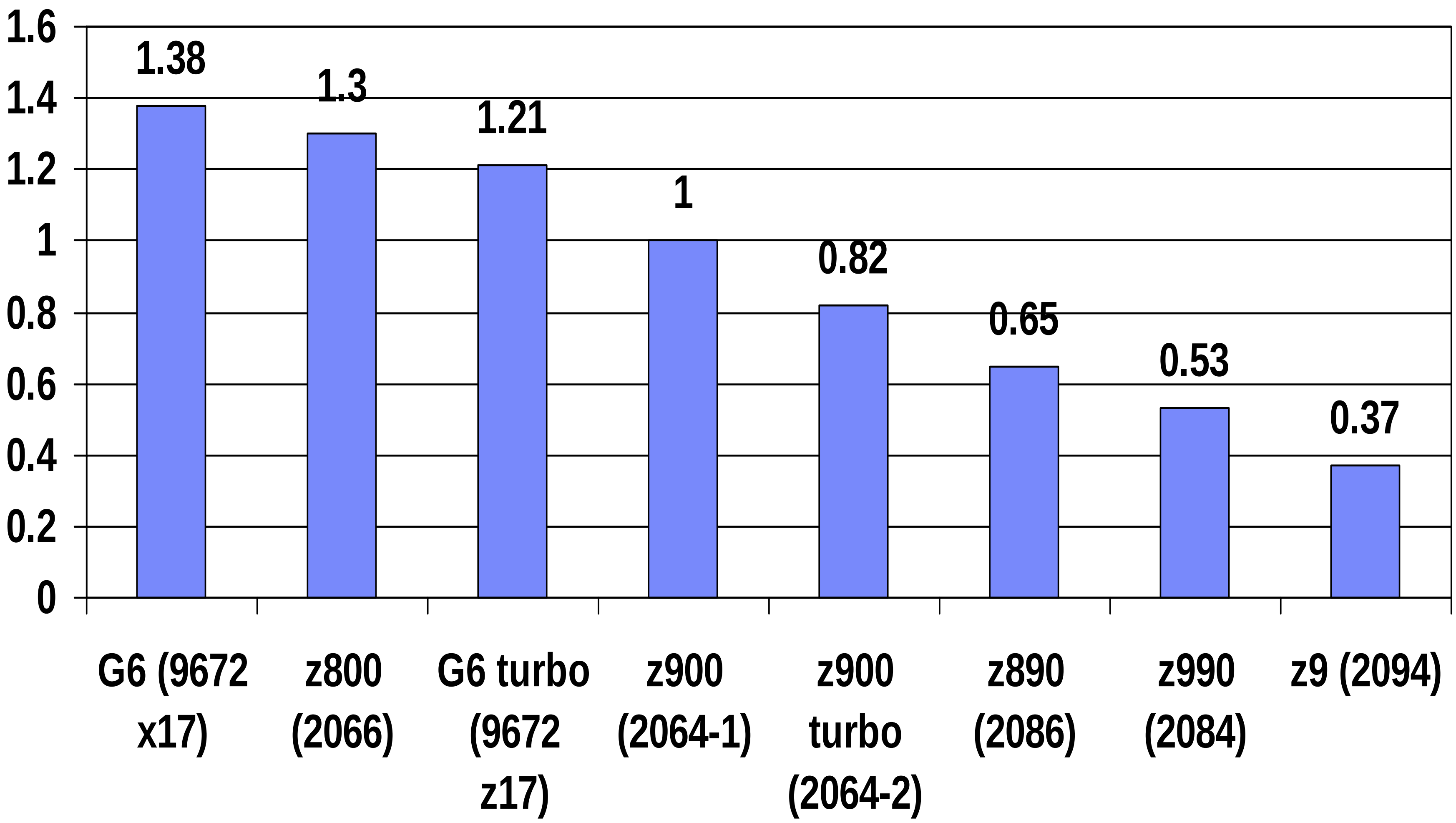
Synergy with CPU Hardware - continued

- Faster Unicode conversion with z900, and more with z990

- Z990 (2084)
  - More than 2 times faster row-level encryption vs z900

- Z9 (2094)
  - MIDAW to improve I/O performance
  - zIIP off-load to reduce total cost of ownership

CPU Time Multiplier for various processor models



Reference

- V8 manuals, especially Performance Monitoring and Tuning section of Administration Guide
- Redbooks at www.redbooks.ibm.com
  - DB2 UDB for z/OS V8 Everything you ever wanted to know… SG24-6079
  - DB2 UDB for z/OS V8 Performance Topics SG24-6465
- More DB2 for z/OS information at www.ibm.com/software/db2zos
  - E-support (presentations and papers) at www.ibm.com/software/db2zos/support.html