Platform: DB2 UDB for z/OS

What's New in DB2 UDB for z/OS Locking and Concurrency

Session: C9

Dr. Jim Teng
IBM Silicon Valley Laboratory

May 25, 2005 - 10:00 am to 11:10 am

Disclaimer

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 express or implied. The use of this information is a customer responsibility.

In addition, the material in this presentation may be subject to enhancements or Programming Temporary Fixes (PTFs) subsequent to general availability of the code.

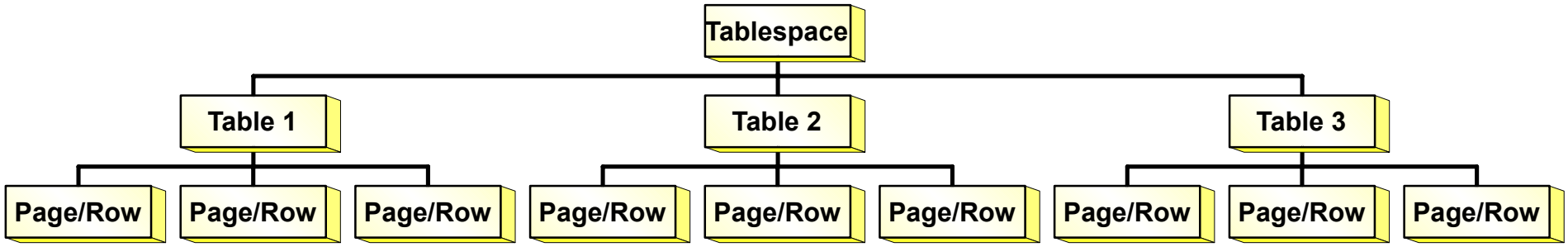
Agenda

- Locking - the basics
- Optimistic Locking for Static Scrollable Cursor
- ZPARM Knobs for Locking
- LOB locks
- Locking Enhancements in V8
  - Deadlock Avoidance between REORG and SQL
  - Data Sharing Locking Enhancement
  - ...

Terminology

- CF - Coupling Facility
- ISO - Lock isolation level
  - RR - Repeatable Read
  - RS - Read Stability
  - CS - Cursor Stability
  - UR - Uncommitted Read
- PI - Partitioned Index
- NFM - V8 New Function Mode
- NPI - Non-Partitioned Index
- XES - z/OS System Lock Manager
- WLM - z/OS Workload Manager

DB2 Locking Hierarchy



DB2 allows 4 levels of locking

High Performance

Tablespace

Partition (for partitioned table spaces)

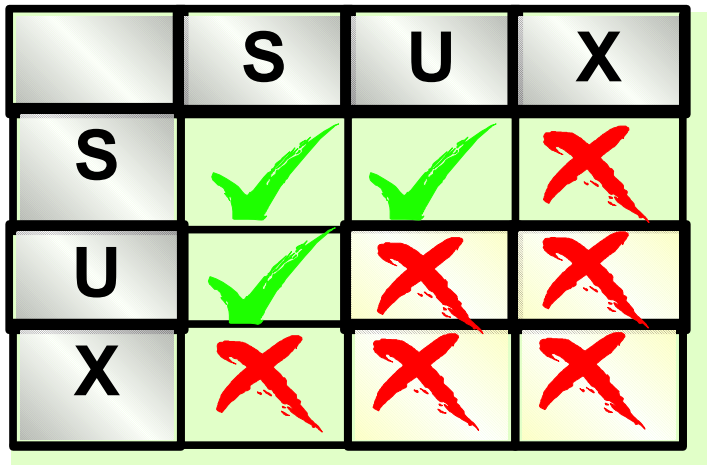
Table (for segmented table spaces)

Page

Row

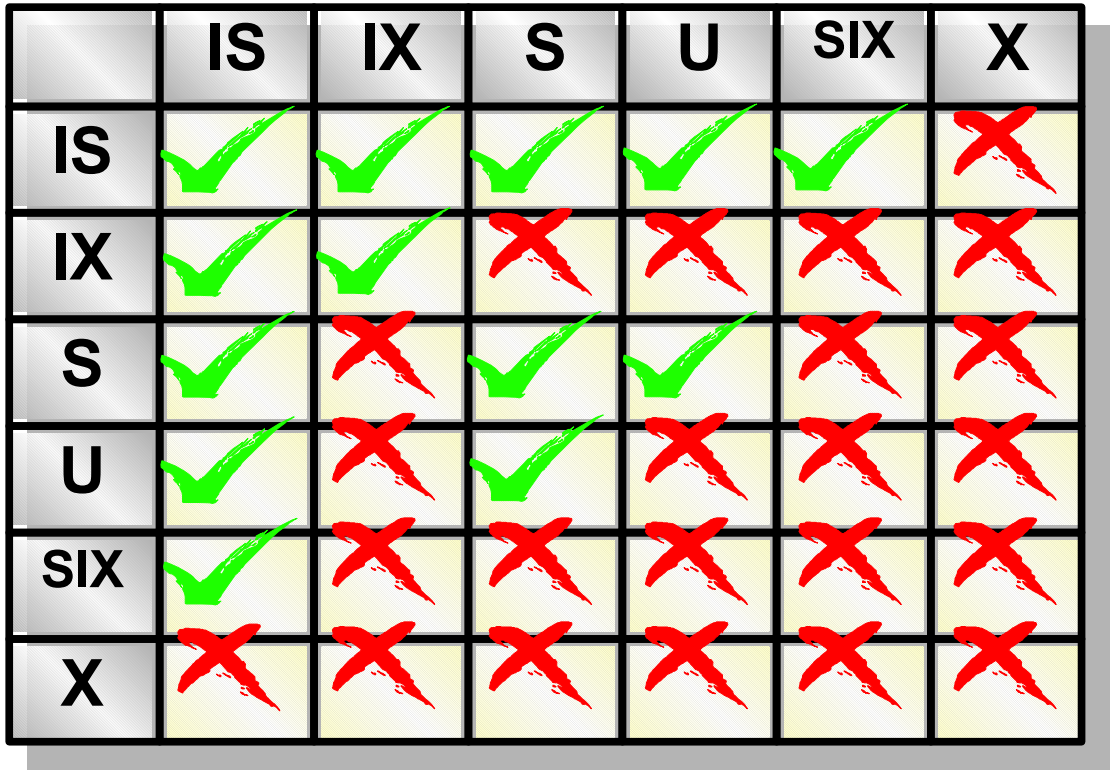
High Concurrency

What makes locking work ?



	S	U	X
S	✓	✓	✗
U	✓	✗	✗
X	✗	✗	✗

Page/Row locks

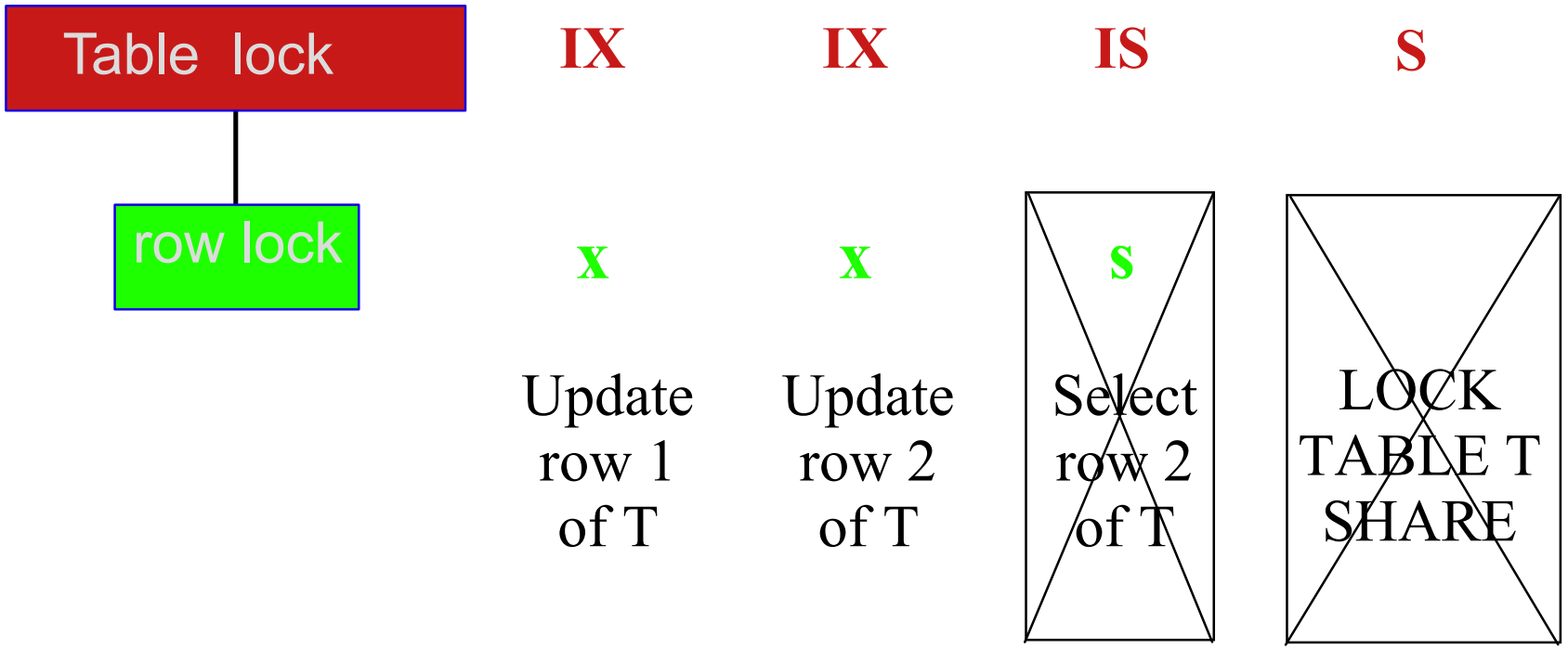


	IS	IX	S	U	SIX	X
IS	✓	✓	✓	✓	✓	✗
IX	✓	✓	✗	✗	✗	✗
S	✓	✗	✓	✓	✗	✗
U	✓	✗	✓	✗	✗	✗
SIX	✓	✗	✗	✗	✗	✗
X	✗	✗	✗	✗	✗	✗

Tablespace/Table/Partition locks

Example

- *CREATE TABLESPACE **TS1** ... LOCKSIZE(ROW)*
- *CREATE TABLE **T** IN **TS1***



What is Drain/Claim ?

- *Drains/Claims are used to serialize between Utilities and SQL*
- A CLAIM registers an SQL agent's use of an object
  - Claim can be acquired on individual partitions
  - Acquired at first access to data or an index
  - CLAIMs are released at commit (except for held cursor)
  - Supports 3 claim classes: RR, CS, and WRITE
- Utilities detect claimers are present and wait
- **Drain Write** waits for all write claims to be released
- **Drain All** waits for claims on all classes to be released
- SHRLEVEL(CHANGE) Utilities are CLAIMers

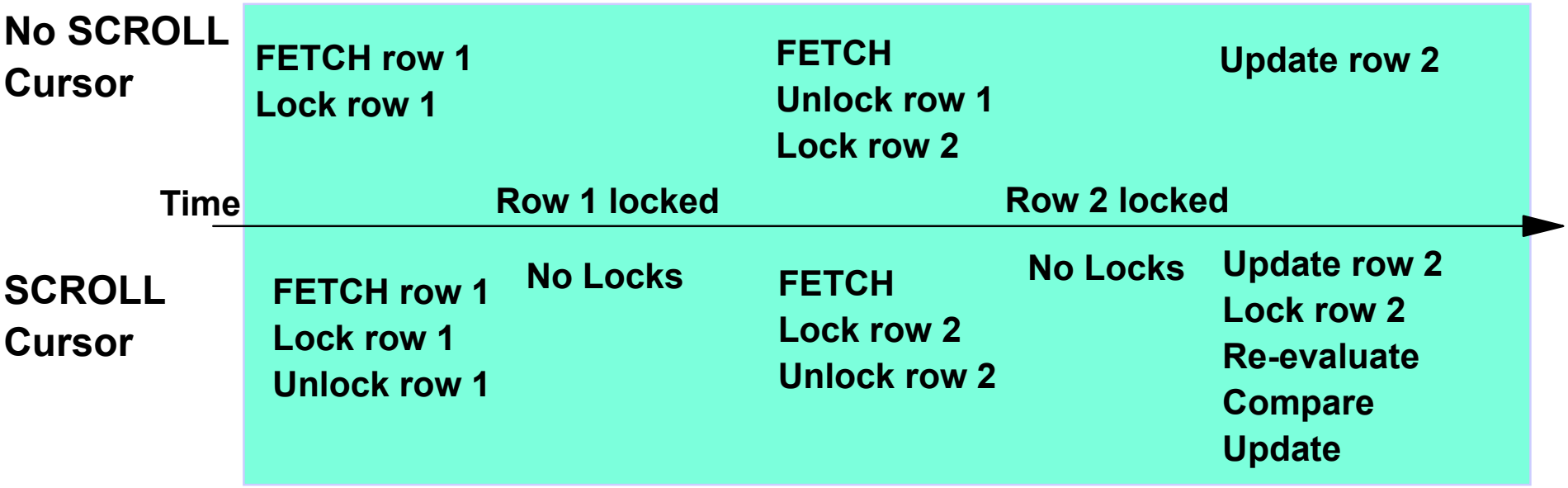
Optimistic Locking for Static Scrollable Cursor

RR - lock every page/row as it is read
RS - lock every row that qualifies stage 1 predicate
CS - lock each row fetched(currentdata(YES))
- lock only if cursor FOR UPDATE OF if (currentdata(NO))
- no rows remain locked after OPEN
UR - No rows locked

Optimistic Locking Mechanism
If not RR/RS, rows are not locked at the end of OPEN
When positioned updates/deletes requested,
-DB2 locks the row,
-Re-evaluates the predicate and
-Compares by Column Value (cols in SELECT list)
to determine if the update/delete can be allowed

Optimistic Locking for Static Scrollable Cursor ...

- Release locks after fetch
- Because we are optimistic that no positioned update or delete will occur
- Even if it does, we are optimistic that the pertinent values will not have changed
- So we compare by value under a new lock to ensure data integrity



ZPARM Knobs for Locking

- *EVALUNC - Applies to ISO(CS or RS)*
  - YES - Defer locking until after predicate evaluation
  - NO - If lock avoidance fails, lock row before predicate evaluation (default)
- RRULOCK - Applies to ISO(RS or RR) on SELECT FOR UPDATE
  - YES - Acquires U locks on FETCH
    - If no update, U is demoted to S on next fetch
    - If update, U is promoted to X in COMMIT duration
  - NO - Acquires S locks on FETCH (default)
- XLKUPDLT - Applies to searched UPDATE or DELETE
  - YES/TARGET - Acquires X locks on qualified rows based on stage-1 predicates (PQ98172 introduces the TARGET option)
  - NO - Acquires S/U first, then promote from S/U to X (default)

ZPARM Knobs for Locking ...

- *RELCURHL - Applies to Cursor Hold*
  - YES - Release row/page lock on the positioned row (default)
  - NO - Keep lock on row/page of position (if any) across commit
    - Only need for this would come from some application dependency
- SKIPUNCI (V8) - Applies to ISO(CS or RS)
  - YES - Skip uncommitted insert
  - NO - Wait until uncommitted insert commits or rolls back (default)

LOB Locks

- *Used to determine whether space can be reused for deleted LOBs*
  - Not for concurrency control
- Locks are held until commit
  - Insert/Update - X LOB lock
  - Select/Delete - S LOB lock
    - No lock avoidance
    - ISO UR skips uncommitted inserts
      - If acquired, will keep S locks until commit
  - Held across commit for held cursor or held locator
- Increase LOCKMAX value to avoid lock escalation
- Frequent commits, free locators and release held cursors to reduce number of held LOB locks

DDL and DML Concurrency

- Unable to support parallel DDLs against the same database due to X-DBD lock
- How to improve DDL concurrency ?
  - Reducing the number of objects within a database
    - Also helps to reduce the EDM Pool Size and
    - Logging volume
  - Avoid mixing DDLs and DMLs within the same COMMIT
  - Group all DDLs within the same database in the same commit scope
    - The deletion and re-insertion of DBD will be done once, instead of each DDL statement
  - Don't delay commit after DDLs
- How to improve DDL and DML concurrency ?
  - Dynamic SQL will not acquire S DBD locks if ZPARM CACHEDYN = YES

Agenda

- Locking - the basics
- Optimistic Locking for Static Scrollable Cursor
- ZPARM Knobs for Locking
- LOB locks
- **Locking Enhancements in V8**
  - **Deadlock Avoidance between REORG and SQL**
  - **Data Sharing Locking Enhancement**
  - **...**

Deadlock Avoidance between Utilities and SQL

- *Problem: Drain/Claim deadlocks between REORG and SQL*
  - Draining/claiming data/indexes in opposite order
  - Draining/claiming partitions in opposite order
  - Drainng/claiming writers and readers in opposite order
  - Could also happen for QUIESCE, RECOVER, CHECK INDEX, LOAD, COPY, ...
  - Deadlocks will not be detected by IRLM
    - Typically, application gets a -911 lock time-out SQL code
    - Utilities could end with RC = 8 and reason code = 00C200EA

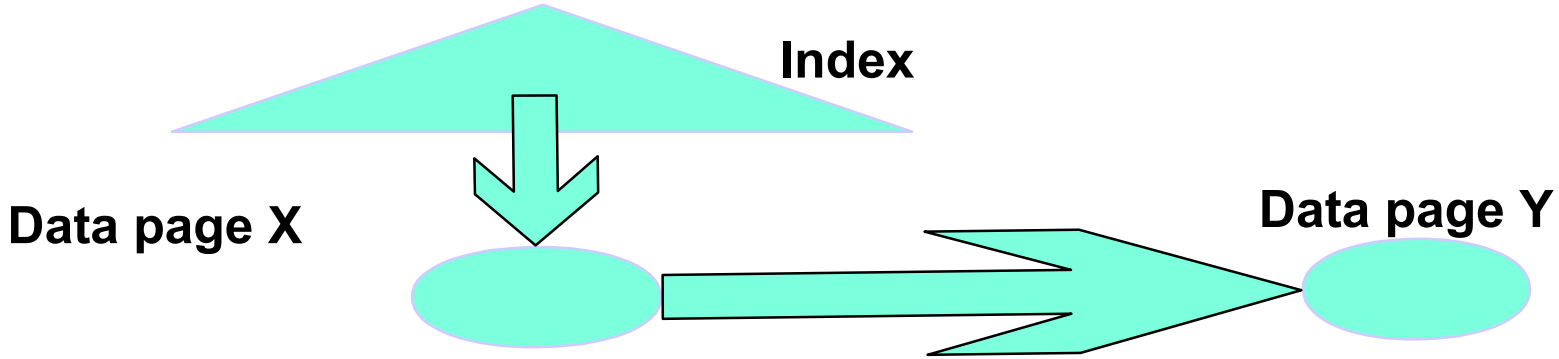
Deadlock Avoidance between Utilities and SQL ...

- Solution:
  - SQL will always claim data first before claiming indexes
  - For Partitioned Tables
    - Introduces an extra claim at the table space level
    - Claim table space first before claiming partition(s)
    - Drain table space first before draining partition(s) for a table space level utility
  - DISPLAY DATABASE ... will show table space level claims and drain locks
  - For Online REORG, could use DRAIN ALL to alleviate drain/claim writers and readers in opposite order
    - But, may impact availability to R/O applications during the last log apply phase
  - Available in V7 via PQ96628 with ZPARM CLAIMDTA=YES

Partitioning Key Update Enhancement

- *Problem in V7: When a row is updated to move across partition boundary (e.g. from PART 2 to Part 6)*
  - Needs to quiesce applications from accessing data between Part 2 and Part 6
    - Drain table and PI from partitions 2 through 6
    - Drain all NPIs
  - Not usable because deadlock/timeout
  - Disable by setting ZPARM PARTKEYU = NO or SAME
- V8 will no longer need to quiesce applications when moving rows across partition boundary
  - Will only need to acquire S page/row lock on the parent row until commit if the update is done against a dependent table in Referential Integrity relationship

Overflow Lock Avoidance



- When an update of variable length row in data page X results in new row which can not fit in that page,
  - New row stored in a different page Y
  - Its pointer stored in old 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.
- Problem: Potential doubling of data I/O and Getpage and more lock/unlock requests

Overflow Lock Avoidance ...

- V7: no lock avoidance on both pointer and overflow
- V8: lock on pointer only

Isolation CS CD NO or YES	V7	V8
Lock/Unlock for pointer	Yes	Yes
Lock/Unlock for overflow	Yes	No

➡ All locks/unlocks here disappear after REORG
- Consider Reorg when (FAR+NEARINDREF) exceeds 5 to 10% range

Increase Lock Holder WLM priority

- *Problem: A high WLM priority application needs to access/update a row that was X locked by a low priority job*
  - Needs to wait until the low priority job issues a commit
- V8 will temporary increase the lock holders priority using the waiter's priority to reduce the lock wait time
  - Invokes WLM to raise the holder's priority when the waiter exceeds 1/2 of the lock time-out value
  - limited to lock holders on the same DB2 member
  - Resume lock holder's original WLM service class when lock is released
  - Prereq z/OS 1.4 (WLM Enqueue Management)

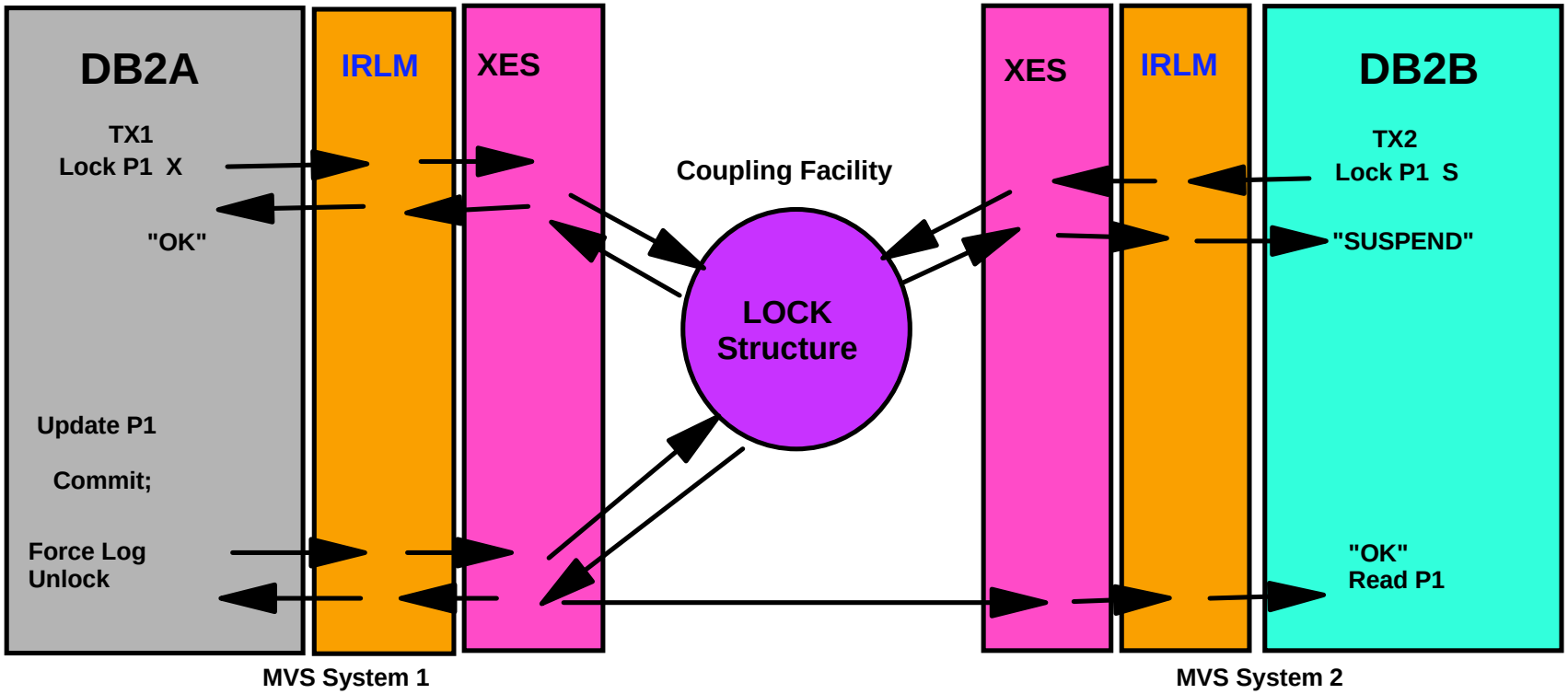
Miscellaneous Enhancements

- *Lock avoidance for singleton SELECT*
  - In V7, ISO(CS) CD(YES) acquires S page/row lock on the qualified row
  - In V8, DB2 will no longer acquire and hold S page/row lock on the qualified row for ISO(CS) CD(YES or NO)
    - Internal cursor is closed after the singleton SELECT is processed
- V8 writes IFCID 337 record when lock escalation occurs
  - Allows performance monitors to report on lock escalation
  - Need to enable statistics trace class 3 or performance trace class 6
  - The DSNI031I message will continue to be written when lock escalation occurs

Miscellaneous Enhancements ...

- *CLI "release locks at close cursor" attribute*
  - *Release S/U row/page locks at close cursor*
  - *Applicable only to ISO (RR or RS)*
  - *In V7, locks are held until commit*
- *SELECT ...WITH ISO(RR or RS) USE AND KEEP(EXCLUSIVE) LOCKS*
  - Acquires X locks on FETCH
- *SELECT ...WITH ISO(RR or RS) USE AND KEEP(SHARE/UPDATE) LOCKS*
  - Acquires S/U locks on FETCH
- SELECT ... KEEP UPDATE LOCKS (V7 syntax)
  - Acquires X locks on FETCH
- Timeout message (DSNT318I) for P locks (PQ87877)

Data Sharing Locking Overview

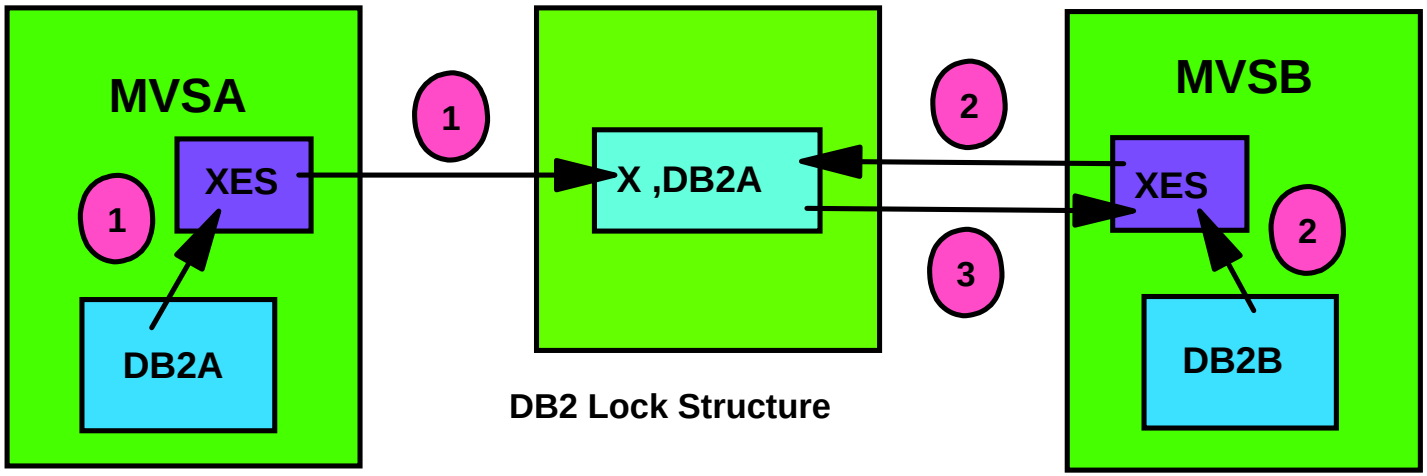


- When "No contention", Global lock granted synchronously for execution of the transaction
- No need to "Suspend" the transaction task (measured in Microseconds)
- "Lock contention" is detected quickly

Data Sharing Global Lock Contention Problem

- *IRLM relies upon the z/OS System Lock Manager (i.e. XES) to handle*
  - Inter-system locking services
- Various IRLM lock levels can ONLY map to one of two XES lock levels
  - IRLM IS and S locks map to XES S lock
  - IRLM U, IX, SIX, and X locks map to XES X lock
- When two members requesting IX lock on the same TS
  - Both IRLM IX locks map to XES X locks - lock conflict
  - XES detects contention and invokes IRLM to perform global contention processing
    - Determine if IX is really compatible with IX
    - Grant lock request

Global L-lock Contention - Example



1. A program on DB2A requests an IX lock on tablespace TS123, and XES propagates this as an X lock on the lock table entry

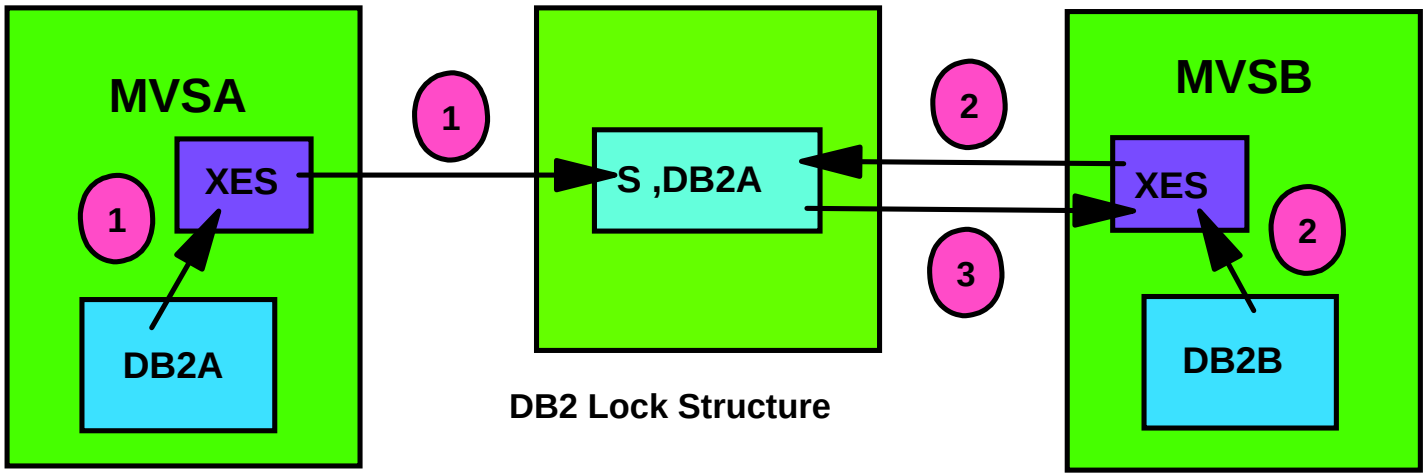
2. A program on DB2B requests an IX lock on tablespace TS123, and XES propagates this as an X lock on the lock table entry

3. MVSB gets a contention response from the coupling facility which is an XES lock contention; XES invokes IRLM to resolve that IX - IX is compatible

Avoid Global L-lock Contention

- *IRLM now maps IX L-locks to XES S*
  - Grant IX lock locally when only IS or IX L-locks held on the object
  - Parent S L-lock still sent to CF without XES contention
- How do we ensure that IX remains incompatible with S ?
  - Gross S parent L-lock now maps to XES X
  - XES global lock contention to verify that a S table L-lock is compatible with another S table L-lock
    - Rare
- Majority cases are IS - IS, IS - IX, and IX - IX
  - Hence performance benefits
- IRLM U, SIX, and X L-locks continue maps to XES X

Avoid Global L-lock Contention - Example



1. A program on DB2A requests an IX lock on tablespace TS123, and XES propagates this as an S lock on the lock table entry

2. A program on DB2B requests an IX lock on tablespace TS123, and XES propagates this as an S lock on the lock table entry

3. MVSB gets a OK response from the coupling facility and IX lock on TS123 is granted immediately (in microseconds)

Child Locks Propagation based on Pageset P-lock

- *Child L-lock propagation no longer based on parent L-lock*
- *Now based on cached(held) state of the pageset P-lock*
  - If pageset P-lock negotiated from X to SIX or IX, then child L-locks propagated
  - Reduced volatility
  - If P-lock not held at time of child L-lock request, child lock will be propagated
  - "Index-only" scan (if any locks taken) must open table space
- Parent L-lock no longer need to be held in cached state after DB2 failure
  - A pageset IX L-lock no longer held as a retained X lock
    - Important availability benefit in data sharing

LOCKPART NO is no longer supported

- *The parent L-lock and P-lock for partitioned tables will always be managed at the partition level*
- *LOCKPART NO semantic is no longer supported*
  - Change to LOCKPART YES internally
  - Data sharing and non-data sharing
  - ONLY lock the partitions as we need them
  - May see additional partition locks acquired even if LOCKPART NO is specified
- All data sharing locking enhancements are enabled only in V8 NFM after
  - A group-wide shutdown and restart

Benefits with Data Sharing Locking Enhancement

- *Faster lock processing for IX and IS parent L-locks*
  - *IX - IX, IX - IS, IS - IS*
  - *Parent L-lock still sent to CF Lock Structure*
    - *But, there will never be XES-level lock contention for the above common conditions*
- *Child locks propagated based on pageset P-lock*
  - Less volatility
- Avoids the cost of global lock contention whenever possible
- Improved availability by reducing X "retained lock" on the entire table space following a system failure
- Improved data sharing performance, especially for OLTP
- Reduce the need for RELEASE(DEALLOCATE)

Possible Future Enhancements

- Avoid commit duration LOB locks
- Option to skip locked rows for ISO(CS or RS)
- Avoid child lock propagation during group restart
- Support Postponed URs for Restart Light

Reference

- *IBM RedBook at www.redbooks.ibm.com*
  - DB2 for z/OS and OS/390 V7 Performance Topics SG24-6129
  - DB2 UDB for z/OS Version 8 Performance Topics SG24-6465
- DB2 UDB for OS/390 V7 Administration Guide, SC26-9931
- *DB2 UDB for z/OS V8 Administration Guide, SC18-7413-01*

Session Title: What's New in DB2 UDB for z/OS Locking and Concurrency

Session: C9

Dr. Jim Teng
IBM Silicon Valley laboratory
jteng@us.ibm.com

International DB2 Users Group