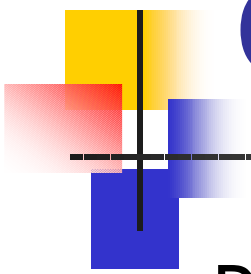
MS SQL Server 7.0 Locking, Logging & Recovery Architecture

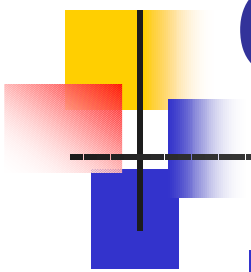
David Campbell

Microsoft Corp.

Overview of 6.x architecture



- Page locking architecture
  - Recovery protocols predicated on page locking
  - Tx undo takes pages back in time
  - Transactional locks cover page "access", no latches.
- Fairly old code base
  - Design target 10's MB of RAM, 1's GB of disk.
  - Small footprint, tightly coupled, code base.
- Old disk/record layout
  - "Device" model – file system in a file.
  - Optimized for space, inflexible
  - No way to represent NULL's, etc.

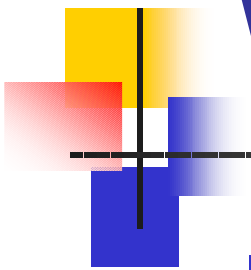


Goals & Constraints

- Architect system for next 10-15 years.
  - No "architectural headroom" in 6.x design
  - Didn't want to compromise new system
- Keep system running
  - Kept system running during entire implementation cycle – major effort; major payoff
- Compatibility
  - Retain application compatibility
- Ease of use
  - Make code smarter rather than add knobs

Architectural Philosophy

- "Clean sheet"
  - Pretend like we're doing it from scratch.
  - Then apply constraints
- Separable components
  - Can be tuned/optimized independently
- Clean interfaces
- The design team enabled this
  - Real world experience from multiple products



What we changed…

- Complete redesign of on-disk architecture
  - New database structure
    - Files instead of former "device" model
  - New page structure
    - 2K -> 8K page
    - Header from 32 -> 96 bytes
    - Free space mgmt. and other row locking elements
    - Mechanism to detect torn-writes
  - New row format
    - Extensible – complex types, schema version, etc.
    - Can natively represent NULL's

Core SE architecture…

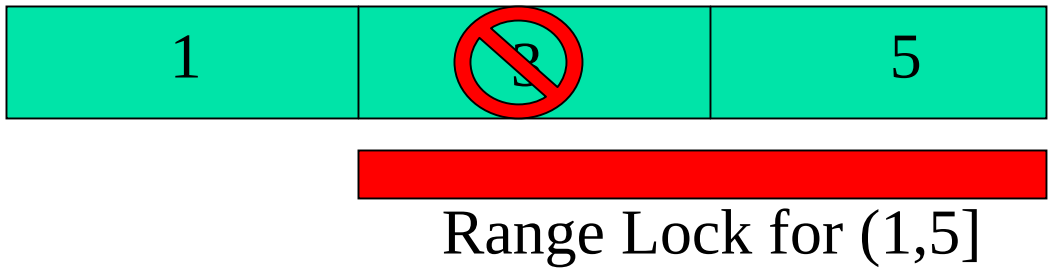
- Rewrote lock manager
- Rewrote recovery manager
- Rewrote log manager
- Rewrote allocation manager
- Rewrote DB and file I/O interface
- Rewrote page access interface
- Reworked access methods for row locking
- (You get the idea…)

Architecture on a slide

- Key-range index locking, (as opposed to data or RID locking).
- Concurrent Table, Page, and Row locking granularities
- ARIES based recovery protocol
- Logical undo, (navigate indices to perform undo).
- Multi-level recovery
  - Ensures physical index consistency for logical undo phase.

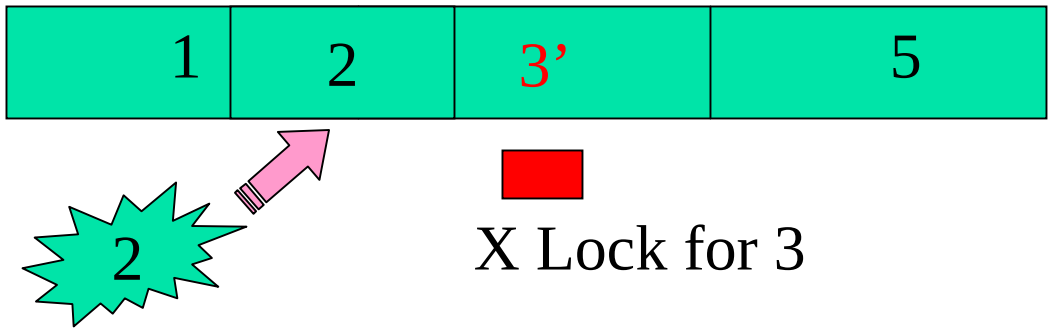
Key Range Locking

- How the "ghost" record was born…
- Initially implemented straight range locking for deleted index keys.



Key Range Locking

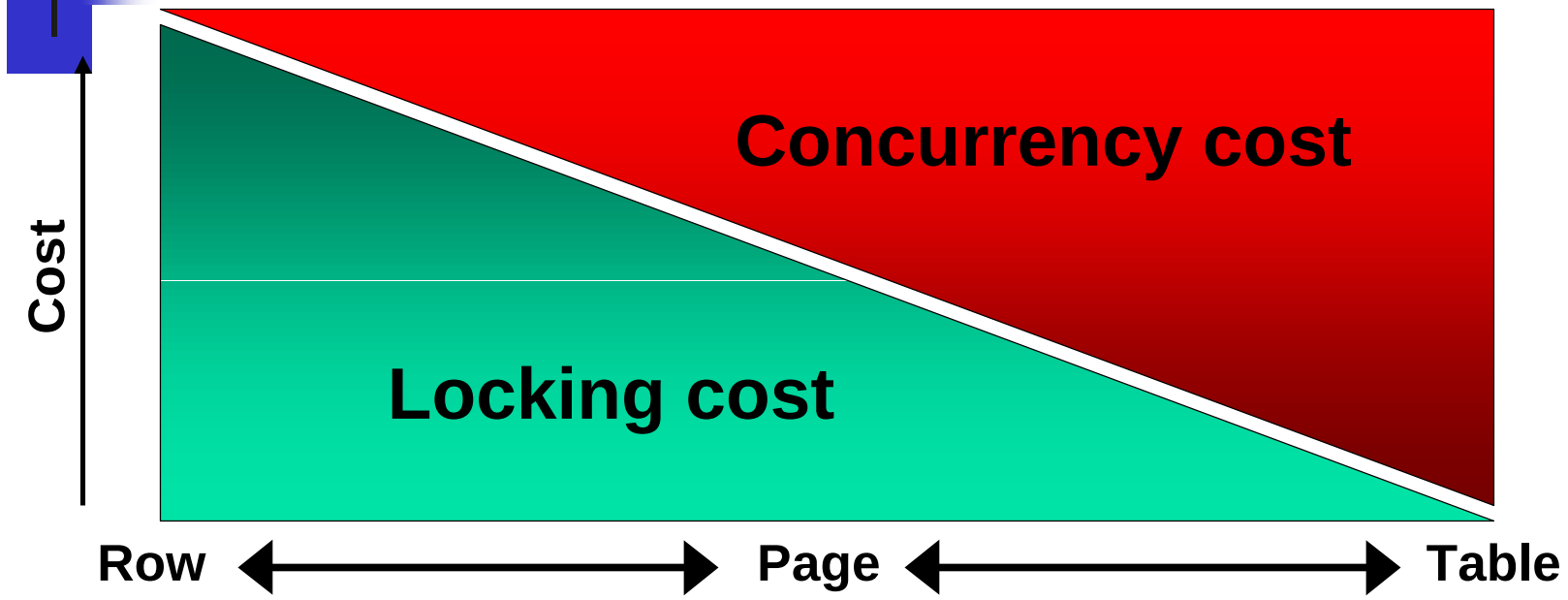
- Locking of entire resultant delete range too prohibitive
  - Solution was to implement deletes as updates.
  - Allows concurrent inserts into range formed by delete



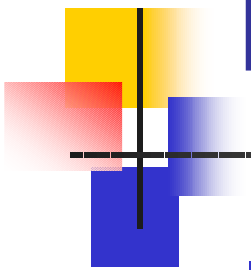
Key Range Locking

- Cleaning up ghosted keys
  - Can be removed when X covering delete is released.
  - Can be removed en-masse when page LSN < oldestActiveLSN
  - Harvested when we need room for an insert
  - Added a harvester thread for “sliding” data

Dynamic Locking



- Row locks are great for concurrency but require lots of bookkeeping and lock manager calls
- Table locks don't allow much concurrency but are cheaper to acquire and manage
- Each has its place

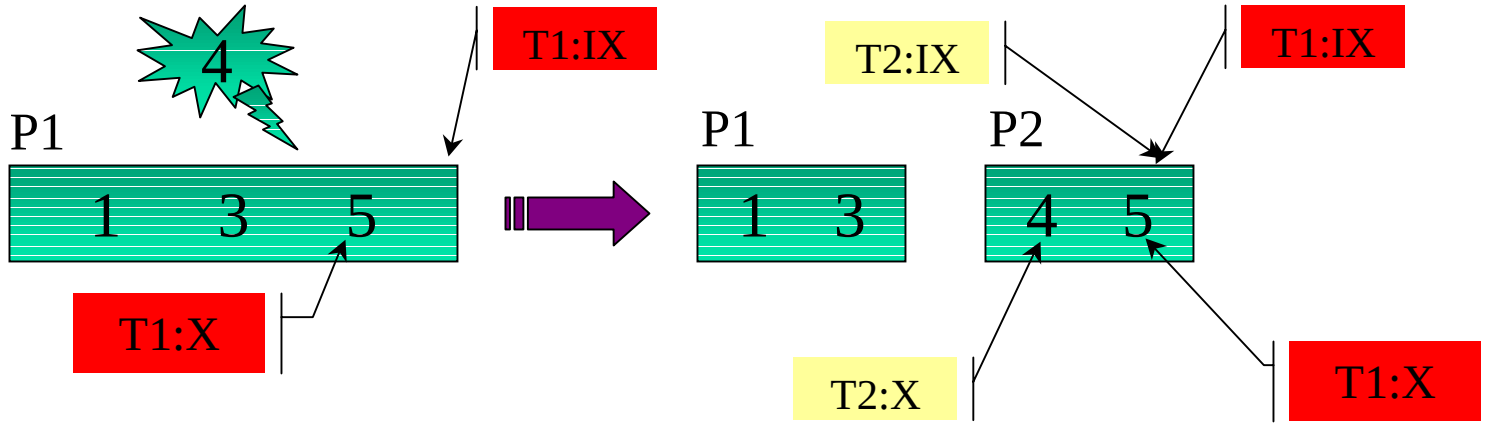


Page locking has its place

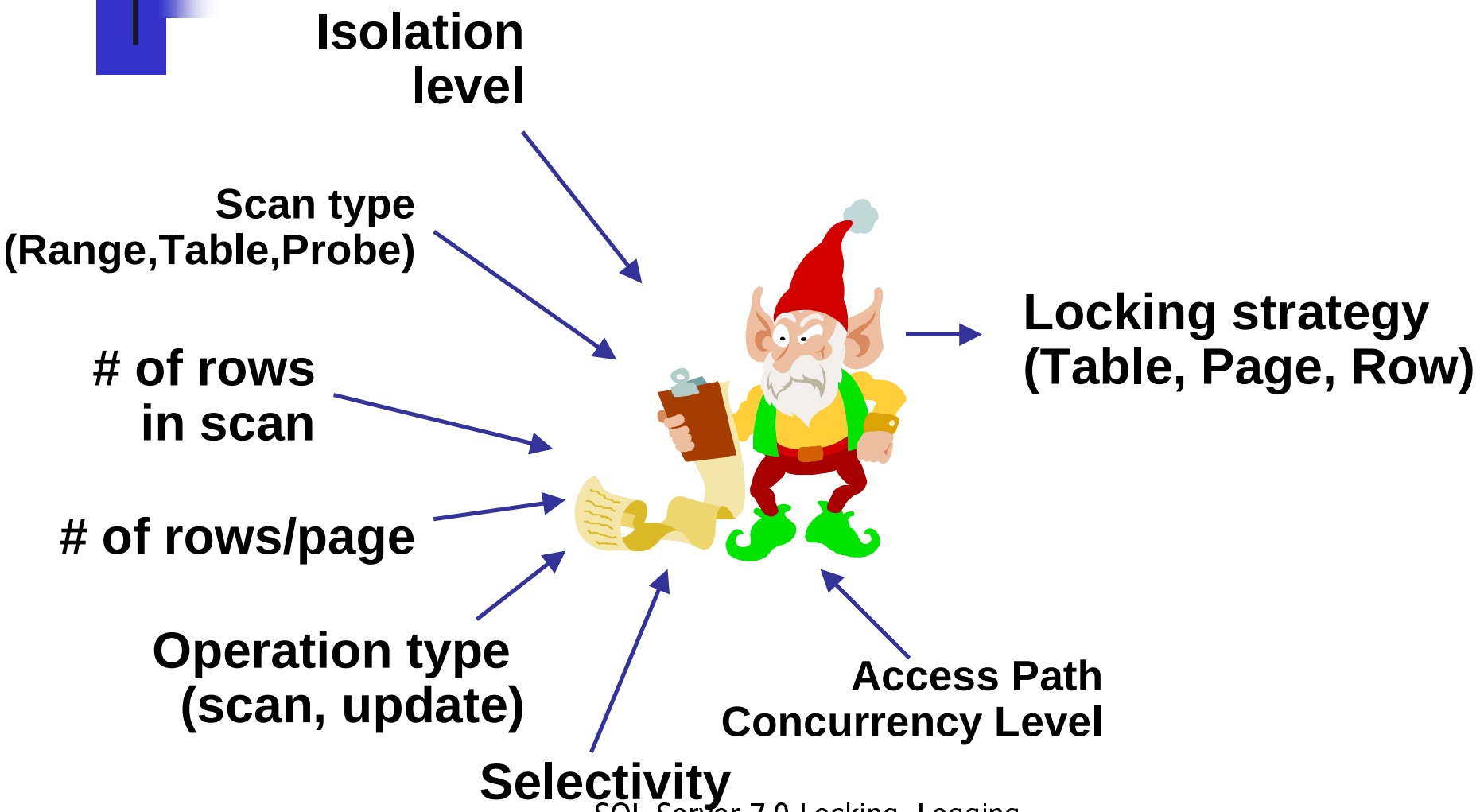
- Concern: cost of locking and unlocking each row for a scan – particularly index range scans, which are dense
- Having pages in the lock hierarchy requires intent locks for multi-granular locking.
- The trick is how to handle page splits.

Pages in the lock hierarchy

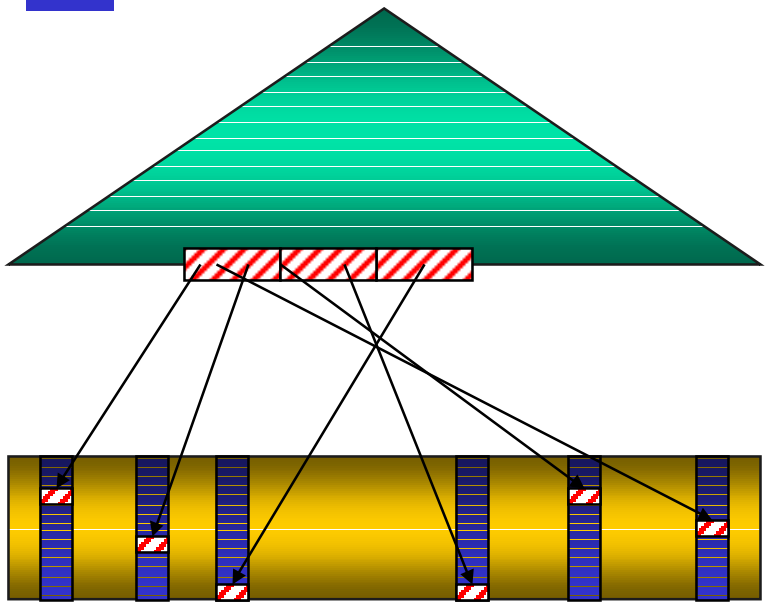
- When a Tx splits a page, it must move all intent locks corresponding to the moved keys – even those for other transactions.



Dynamic Locking



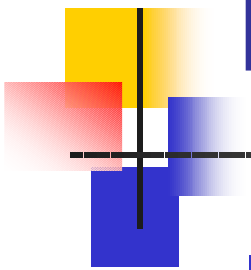
Dynamic Locking Example



- Scan index and then lookup rows based upon scan:
  - Use page locks to scan index, (contiguous compact range)
  - Use row locks when reading from fixed-RID base table
- Minimizes locking cost, maximizes concurrency
- All done dynamically at run-time

Latching hurts

- Had to add latches to handle concurrent access to pages – previously covered by page locks.
- Naïve implementation
  - Latch/unlatch for each row in scan
  - Save off scan position (key) for each row
  - This can cost 100's instructions/row



Latching hurts

- How we handled this:
  - Don't release latch for each row
  - Rather than releasing latch when leaving page, hold the latch in "lazy" mode with notification
  - When notified that your lazy latch is blocking some other requestor:
    - Save off scan key
    - Release lazy latch
  - Typical cost:
    - 1 Latch acquire/release per page, don't have to save off keys

Multi-level recovery

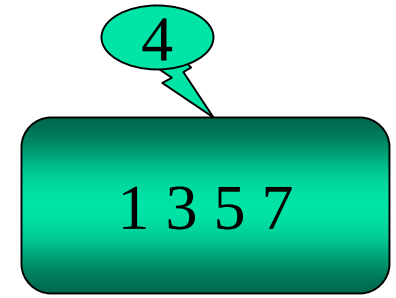
- Undo is logical: i.e. undo of index delete is normal index insert operation.
- Requires physically consistent index structure to perform undo operation.
- Solution: "System" transactions and multi-level recovery protocol
  - *MLR: a recovery method for multi-level systems* David B. Lomet, Sigmod '92

Multi-level recovery

- Index splits performed under latches to provide isolation.
- Tree is inconsistent during split
- Database has 3 consistency states:
  - Inconsistent
  - Physically consistent
  - Transactionally consistent

Multi-level recovery

- Crash during split



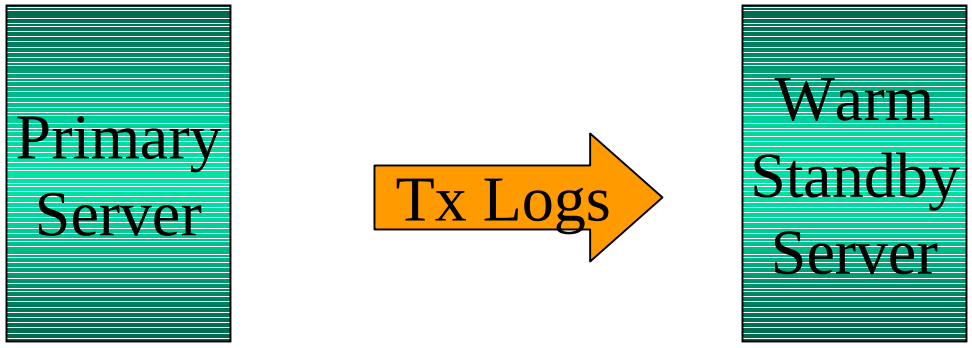
Redo phase - inconsistent

Undo system Xacts – physically consistent

Undo user Xacts – X-actional consistency

Warm Standby Server

- Copy and apply log to remote server
  - Disaster recovery, reporting, etc.



Warm standby server

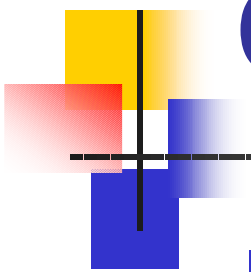
- 6.x algorithm
  - Copy log
  - Perform recovery
    - Redo
    - Identify losers
    - Undo losers by taking pages “back” in time
  - Copy log… (New log dump contains data for previous loser Tx’s)

Warm standby server – 7.0 Issue

- The problem:
  - Copy log
  - Perform redo
  - Undo generates compensating action and takes pages to point where subsequent redo will fail!
- Thought about physical undo
  - Complicated
  - Not running same code as "real" time

Warm standby server – 7.0 issue

- The solution: Copy on first undo reference
- Algorithm:
  - Copy log
  - Copy pages from side file
  - Perform redo, record redoLastLSN
  - Perform undo
    - If pageLSN < redoLastLSN copy page to side file
  - Back to step 1

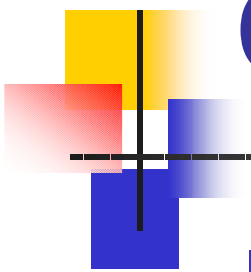


QA/Testing

- Fail Fast
  - If something's wrong don't continue with bad state
- Code Reviews
  - Peer reviews of all checkins.
  - More detailed reviews of new subsystems
- Assertions/Assumption checks
  - More than parameter validation.
  - Latch enforcement (memory protect unlatched pages)
  - Assert that proper locks held on all modifications

QA/Testing

- "SE Stress"
  - Highly concurrent workload with stress induction
  - Clients randomly canceling queries and being killed.
- RAGS
  - Random query generator
  - Results can be compared with old release
  - Ran in single user and concurrent mode
- SQL "Killer"
  - OLTP workload with backup and log backup
  - Randomly kill and recover
  - Restore and recover and compare results
  - Runs for weeks…



QA/Testing

- Failpoints
  - Induce failures at interesting points:
    - In middle of B-Tree split
    - At interesting points in 2-phase commit protocol
- Exception induction
  - Throws low level exceptions, (out of memory, log full, lock request cancelled, I/O failure), from every reachable call path.
  - Finds hard bugs that otherwise wouldn't be found in house.
- Playback testing
  - 100's of real-life concurrent customer workloads