MemScale Part 1:

Technical Overview of XOLTP Features in ASE 160 SP03
In-Memory Row Store, Data Row Caching, Hash Caching

Aditya P. Gurajada, SAP
6th Sept, 2017

EXTERNAL

Agenda

Motivation

Btrim Architectural Overview

XOLTP Performance Improvements

IMRS Data Row Caching

Hash Cache Btree Indexes

Summary

Motivation

Performance bottlenecks in traditional disk / buffer caching DB engines

- High cost per instruction due to lack of cache conscious data structures (memory-stalls)
  - E.g. Lots of random memory accesses, multiple data / index / log page accesses within transactions
- Lot of [un-useful] code execution in transaction execution code-path
  - E.g. Write-ahead logging, buffer pinning, locking, large index traversals for point queries
- Scale-up issues (contention, spinlocking …) as number of cores & concurrency increased
  - Contention for various data / index page accesses

Leverage Hardware Trends

- Availability of inexpensive multicore hardware
- High computation power
- Large memory systems
- Increased in-memory processing power (processor speed, number of cores) at lower cost

Objective

- To deliver scalable performance for high-volume transactional workloads
- To overcome concurrency bottlenecks with page based storage
- To be able to scale better
- Provide extreme transaction processing using modern hardware capabilities

XOLTP Terminology

- **ISUDs** (Workload consisting of)
  Inserts Selects Updates Deletes
- **IMRS – In-Memory Row Store**
  Storage substratum to host “hot” rows in-memory
- **DRC – Data Row Caching**
  Performance oriented feature leveraging IMRS to store “hot” transactional rows in-memory
- **HCB Indexes – Hash Cache Btree Indexes**
  On IMRS-enabled tables with unique indexes
- **MVCC – Multi-version Concurrency Control**
  Snapshot Isolation feature, using in-memory versioning created in the IMRS
- **ODMVCC – On-Disk Multi-version Concurrency Control**
  Snapshot Isolation feature, using on-disk versioning created in some tempdb storage

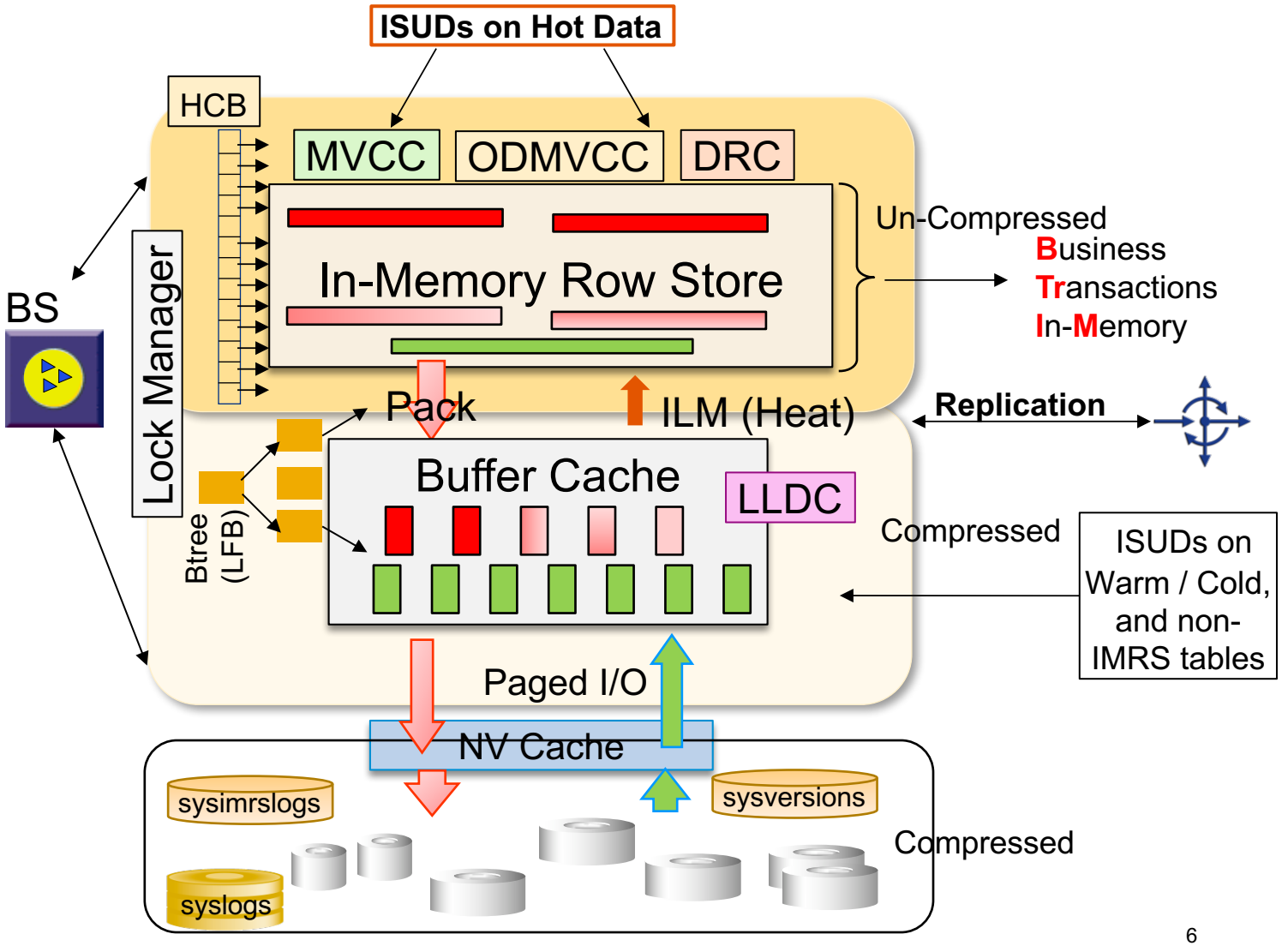
- **IMRS-enabled database / table**
  Database or table that is configured for IMRS use
  Either for DRC or MVCC
- **ODRS – On-Disk Row Store** – aka IMRS log
  Collection of one or more devices of imrslog type providing durable storage to IMRS contents
  - sysimrslogs – System catalog-oriented storage for committed rows in IMRS; ~syslogs for page store
  - imrslogsegment – System segment assigned to imrslog devices; ~logsegment for syslogs
- **Page Store database / table**
  Pre-SP03 / SP03 version of ASE database and table
  Uses existing features; not configured for xOLTP features

Architectural Overview

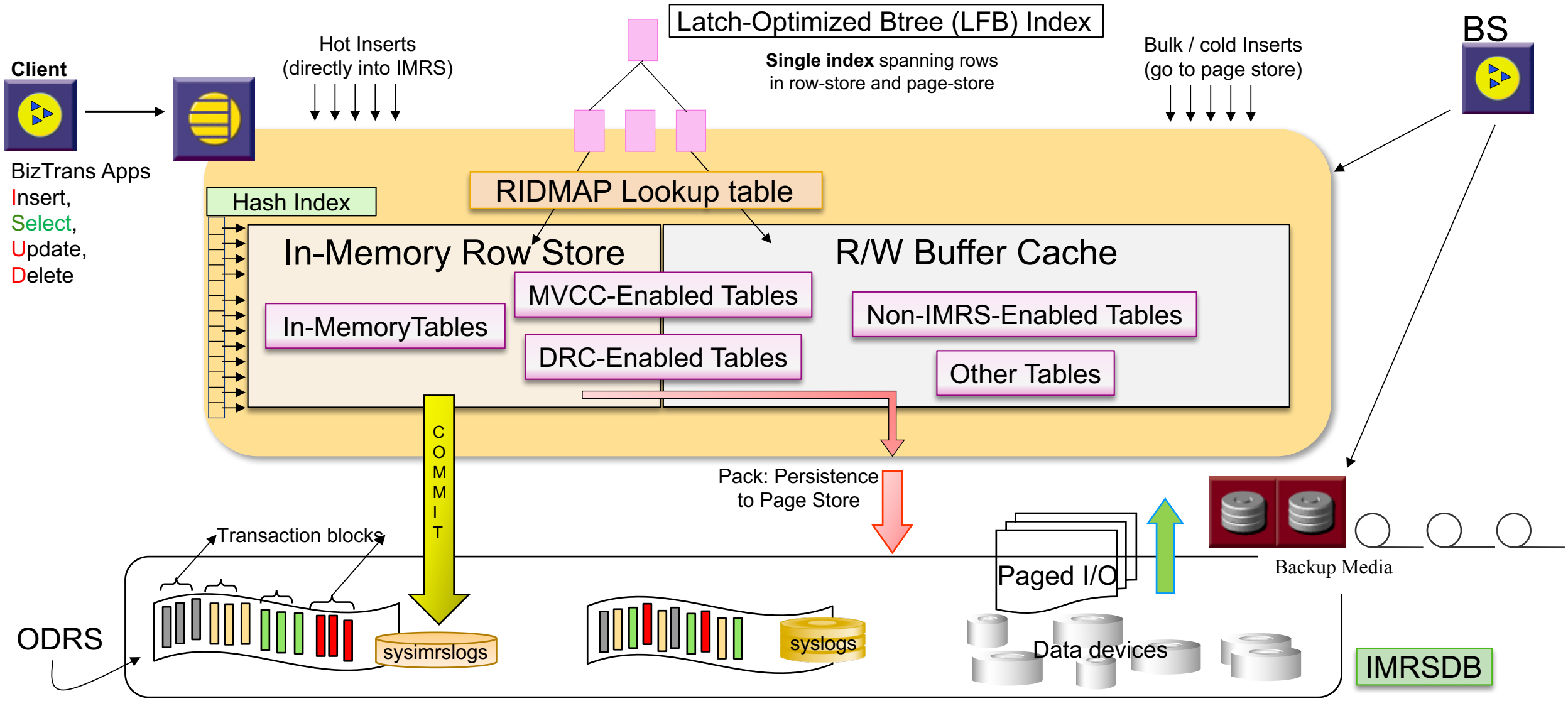
BTrim – In-Memory Row Store, Data Row Caching, MVCC, Hash Cache Indexes

ASE BTRIM Architecture – Business Transactions In-Memory

- In-Memory Row Store
- Data Row Cache (DRC)
- MVCC Snapshot Isolation
- Hash Indexes on rows in IMRS
- Latch-free BTree indexes
- Persistent Store for in-memory rows
- **Tightly-integrated into existing architecture**
- Full support for dump / load backup and Replication
- IMRS-GC, LOB-GC, IMRS-Pack operations, Utility support, DBCC …
- IMRS-Memory manager, Kernel Bucket Fragment Pool manager



ASE BTRIM Architecture – Business Transactions In-Memory



Customer Usage: Data Row Caching? MVCC ?

DRC – Purely performance oriented feature

Guideline: Enable row_caching for few “hot” tables experiencing contention in page store

Use recommendations from Workload Profiler toolkit as initial set of tables to enable DRC

Monitor IMRS performance, sp_imrs metrics and adjust table’s property as needed

Snapshot Isolation – Mainly application compatibility, non-blocking feature

MVCC is in effect only if all tables involved in statement / transaction are SI-enabled

Guideline: Likely start with all tables enabled for snapshot_isolation

IMRS cache requirements

Smaller for DRC. Server adjusts DRC usage based on workload pattern and available cache

Larger for SI as in-memory versioning always requires IMRS memory

IMRS Memory Availability

For DRC, OOM is avoided by redirecting inserts to page store

Pack dynamics attempt to keep sufficient IMRS memory available

XOLTP Performance: Delivered!

Micro-benchmarking across versions

E2E performance gains across versions

E2E scalability across engines

Customer Testing Results with 160SP03

MemScale Feature Options And Benefits

		Performance Improvements	
Feature	Release	Code path	Scaling
Lockless Data Cache (LLDC) Latch Free Btree (LFB) Hardware Transaction Memory	16.0 SP02		✓
Simplified Native Access Plan (SNAP)	16.0 SP02	✓	
Non-Volatile Cache	16.0 SP02		
Data Row Caching (DRC)	16.0 SP03	✓	✓
Hash Cache Btree (HCB)	16.0 SP03	✓	
MVCC	16.0 SP03		✓

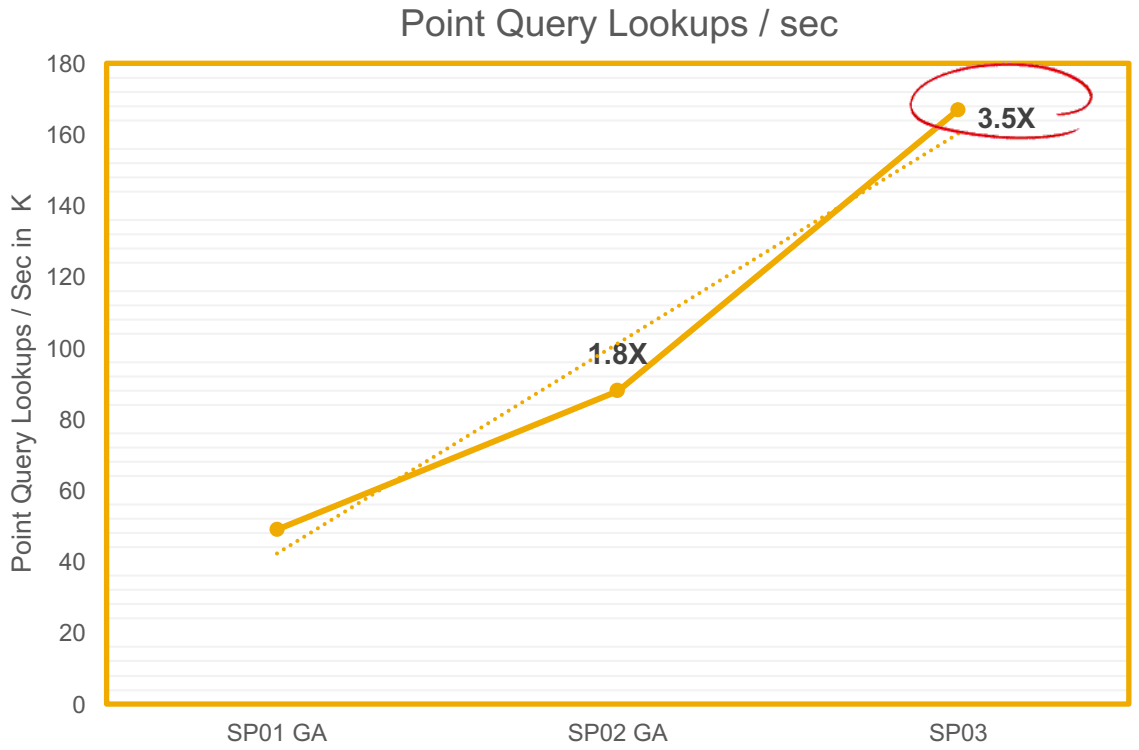
Code path improvements with XOLTP features

Point Query Lookups

- 1 million rows in table
- Each lookup picks one row, unique index scan
- Single client, running lookups in a loop

Improvements from code-path reduction

- Compiled Queries (SP02)
- IMRS / DRC + HCB (SP03)



Multi-core Scaling with IMRS + HCB – 16.0 SP03

Comparison of ASE Performance on 160 SP03, across multiple engines / cores: Page Store v/s IMRS

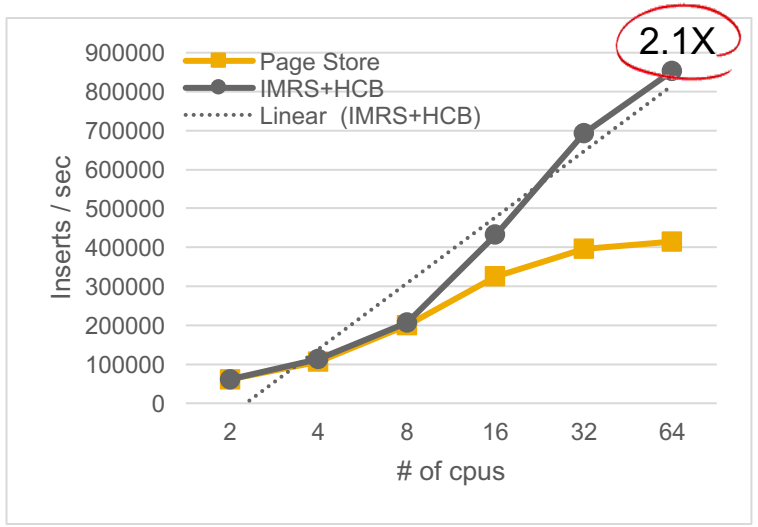
Concurrent Inserts Micro-bm

- 250 clients concurrently insert
- Into single table, with one index
- Page-allocation completely avoided
- Page-level latch conflicts avoided for data pages
- Rows inserted directly in-memory

Concurrent Updates Micro-bm

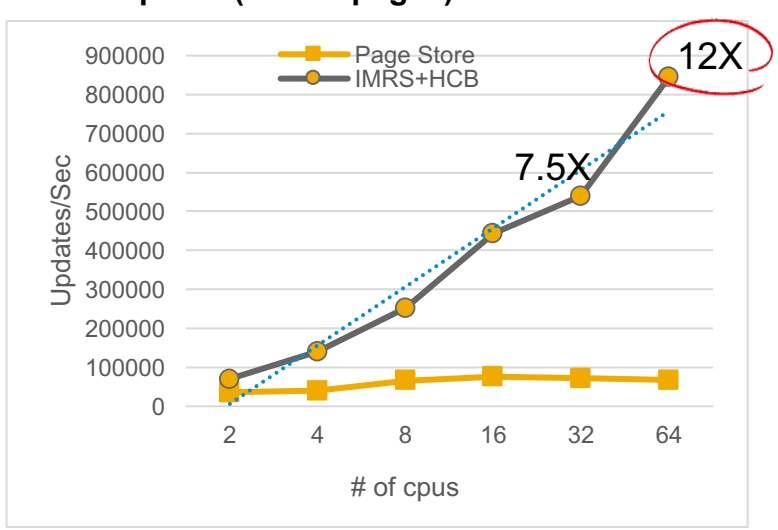
- 250 clients concurrently update
- Multiple updates in one transaction
- Each client updates its own rows
- Updates performed in-memory
- Scaling improvements by avoiding data page-latching overheads

Conc Insert Micro-benchmark



Run at 64 cpus	CPU busy	Inserts/ sec	Gain
Page Store	40%	415,081	
IMRS + HCB	91%	852,879	2.1X

Conc Update (on hot pages) Micro-benchmark



Run at 64 cpus	CPU busy	Updates / sec	Gain
Page Store	16%	68,474	
IMRS + HCB	63%	845,070	12X

End-to-End OLTP* Performance : Minimal Tunings – Feature-Wise

(*) 250 clients, concurrently executing ISUD transactions, on 64-engines

ASE version	Transactions*/ min	Gain	CPU Utilization	Analysis
15.7 SP64	42K	Baseline	99%	Cache manager spinlock contention
16.0 SP01 GA	42K		99%	Same as above
16.0 SP02 GA (with LFB+LLDC+SNAP)	282K	**6.5X**	45%	Buffer unpinning on data pages, leading to latch conflicts resulting in syslogs semaphore contention
16.0 SP03 with same features	315K	**7.5X**	44%	Improvements made in LFB. Data page latch contention remains
+ DRC	719K	**17X**	95%	Data page latch contention eliminated. In-Memory transaction processing
+ HCB	771K	**18X**	96%	Further code path improvements with HCB

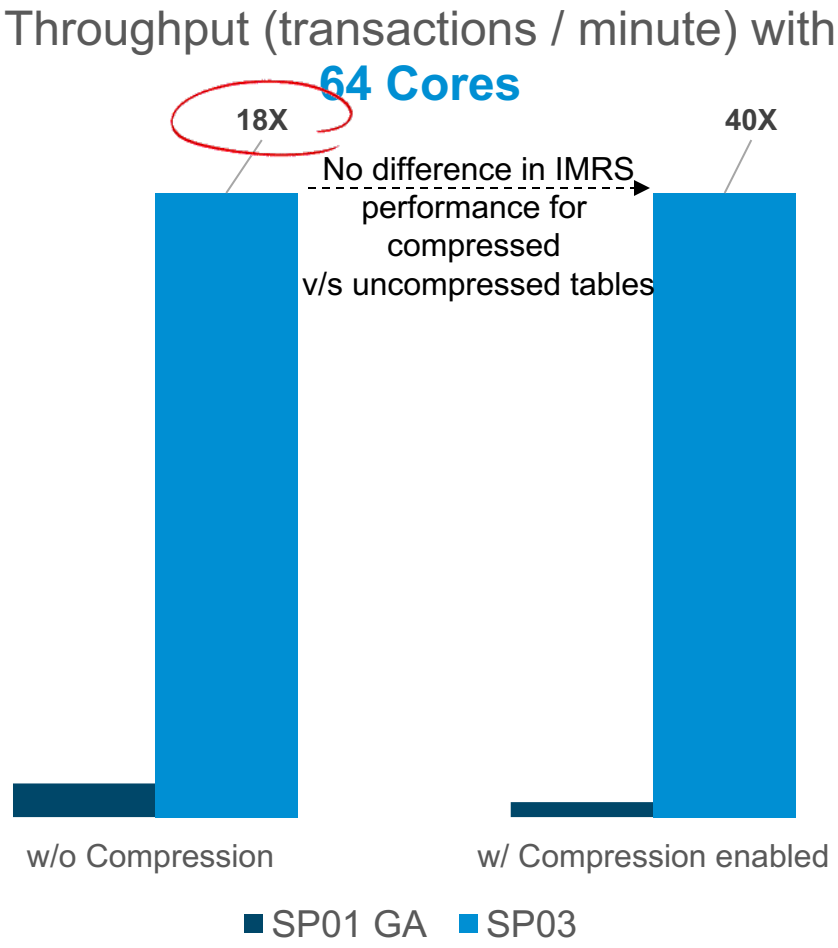
End-to-End throughput for an OLTP workload

Scaling Improvements with minimal tuning – TCO – 64 engines
- Mostly only configuration changes; e.g. lock hash table size, named cache for syslogs, procedure / statement cache sizing…
- No table- / object-level tuning included (e.g. partitioned tables, named caches for tables & indexes … not done)

SP02 scaling improvements:
- Latch Free Btree (LFB)
- Lockless Data Cache (LLDC)

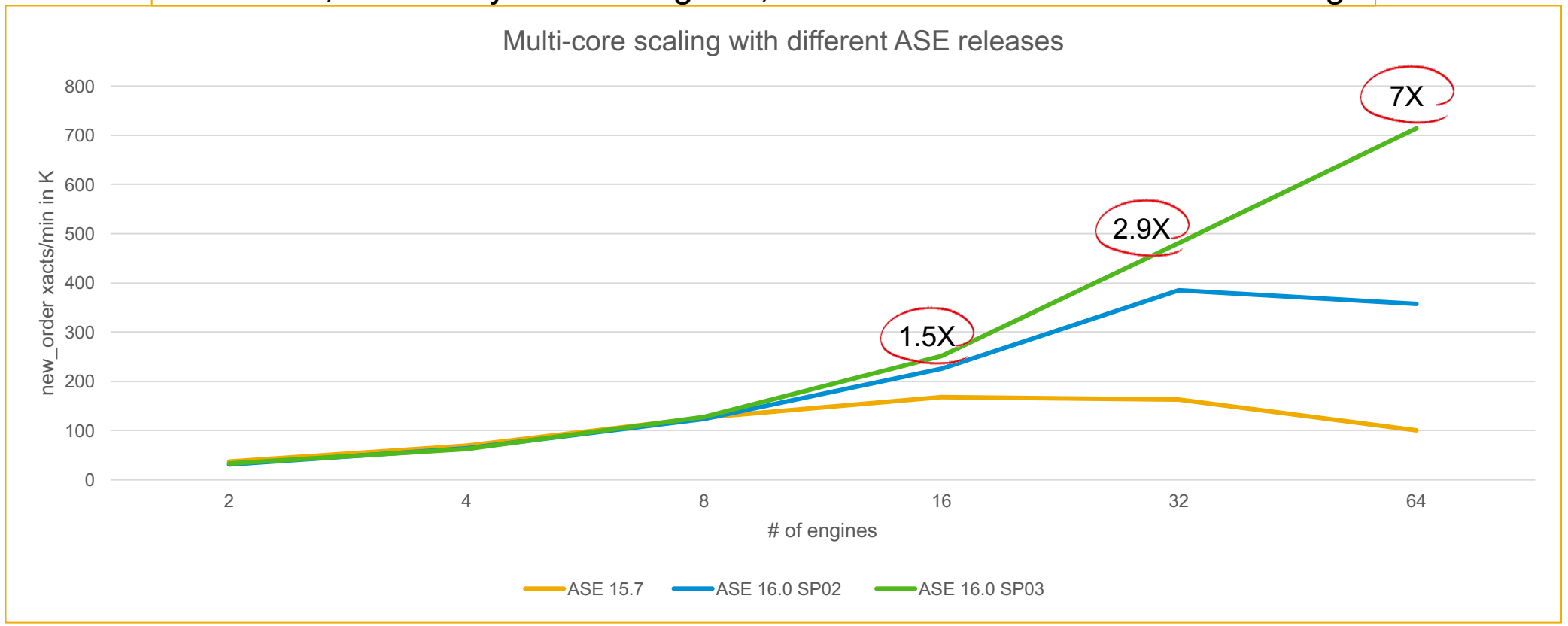
SP03 – Further scaling improvements:
- In-Memory Row Cache
- Hash Indexing
- Compression code-path improvements

E2E throughput for OLTP workload w/ minimal tuning:
- From SP02 (LFB + LLDC + SNAP) to SP03 (SP02 features + DRC + HCB)
- 2.5X performance gains in transactions / minute



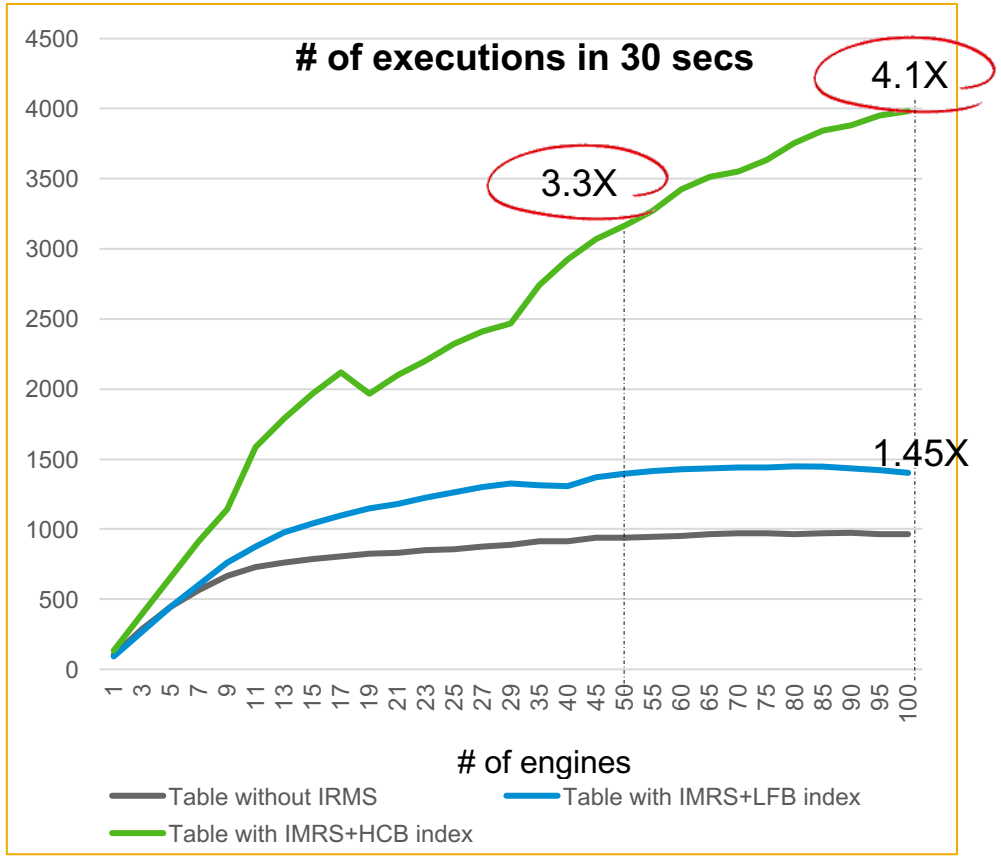
160 SP03 Multi-core Scaling – E2E OLTP Workload

250 clients, Scalability across engines, OLTP workload with minimal tunings



With IMRS, ASE throughput for highly concurrent workloads scales well even at higher core counts

Concurrent Select Performance – Beta Customer Scenario



Small # of rows selected by 100 clients
Client selects its own unique row
Scalable Performance across multiple cores

2-table nested-loop join. Hot rows cached in IMRS
Outer: 20K rows, Inner: 10 million rows
Query returns about 1200 rows
Scalable Performance across multiple cores

When to use which XOLTP feature?

Resolving performance / scaling issues with multiple engines

Performance Issue	Example Workload	Feature
Cache manager spinlock contention for search, keep, etc.	Most concurrent workloads, running with single / few caches	LLDC
Cache manager spinlock contention for latch, unlatch of index pages; Benefits index reads also. Real SH-EX latch conflicts on index pages	Frequently accessed index, causing latches to be frequently taken on root page. Benefits indexes on small & large tables.	LFB
Latch conflicts on data pages	Few pages of a table frequently updated; Highly concurrent inserts / updates on a DOL table	DRC
Logical lock blocking for selects due to concurrent updates to shared data	Reporting applications running concurrently with update-intensive applications	MVCC

Improving response times with each engine

Example Workloads	Feature
Query plans involving tight loops; Scalar aggregate queries	SNAP
Point query lookups using unique index – common in almost any OLTP workload	HCB

IMRS – Data Row Caching

Customer Usage –Top-Level Interfaces

Entity	New Installation	Migration	Downgrade
IMRS	sp_cacheconfig row_storage	→	0
Installation	DISK INIT type = imrslog	→	sp_dropdevice
Database	CREATE DATABASE	ALTER DATABASE	→
Tables	CREATE TABLE	ALTER TABLE	→
Indexes (LFB / Hash Index)	CREATE INDEX	CREATE / ALTER INDEX	DROP INDEX

Indexes on IMRS-enabled tables

Btree indexes are fully and transparently supported

Data row can be either in page store or in-memory

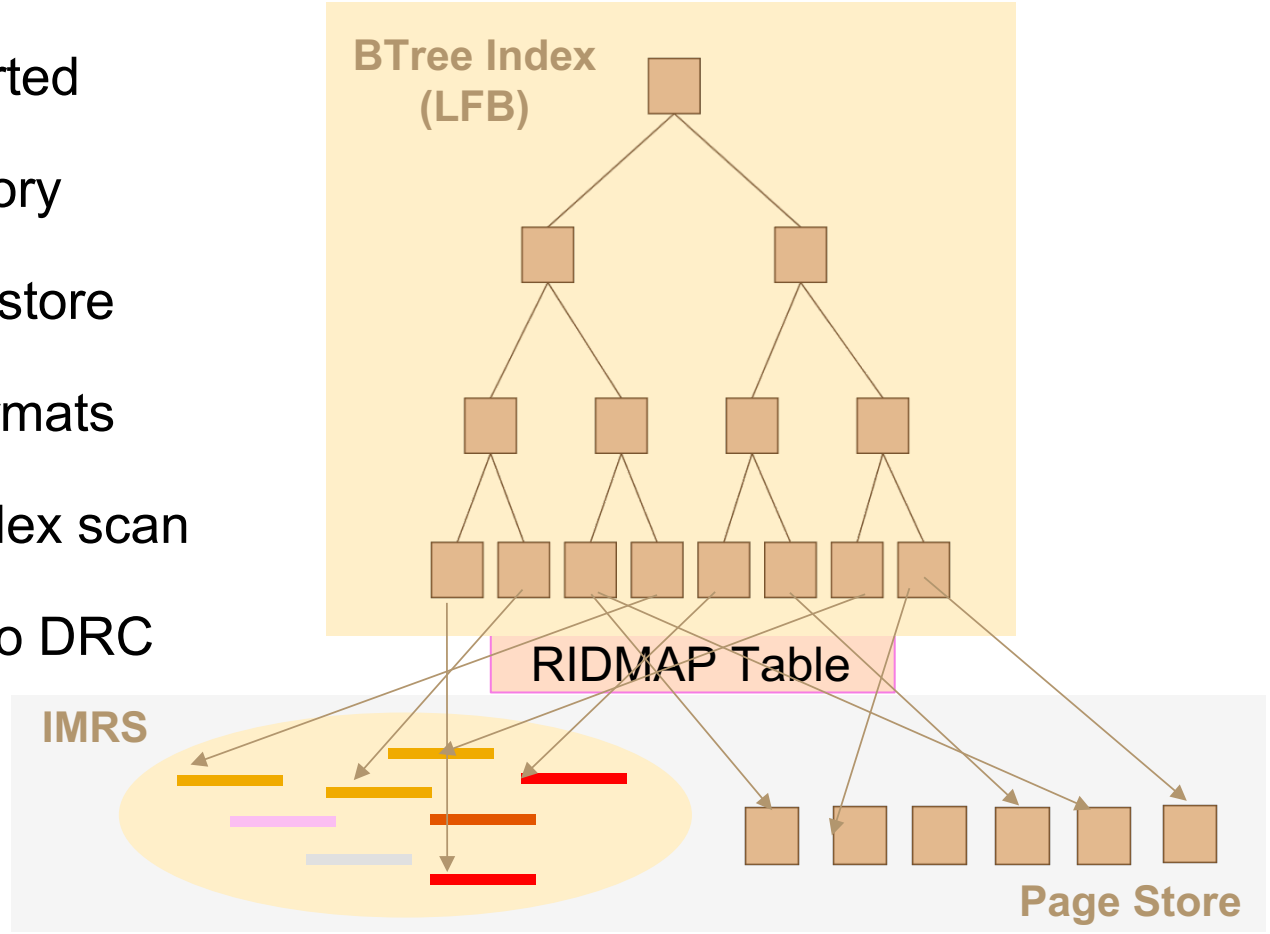
Index leaf row can “point to” data rows in either store

No changes to existing Btree leaf-row / page formats

Internal mapping table to locate data row via index scan

Easy migration for existing databases to move to DRC

- No need to drop and recreate indexes



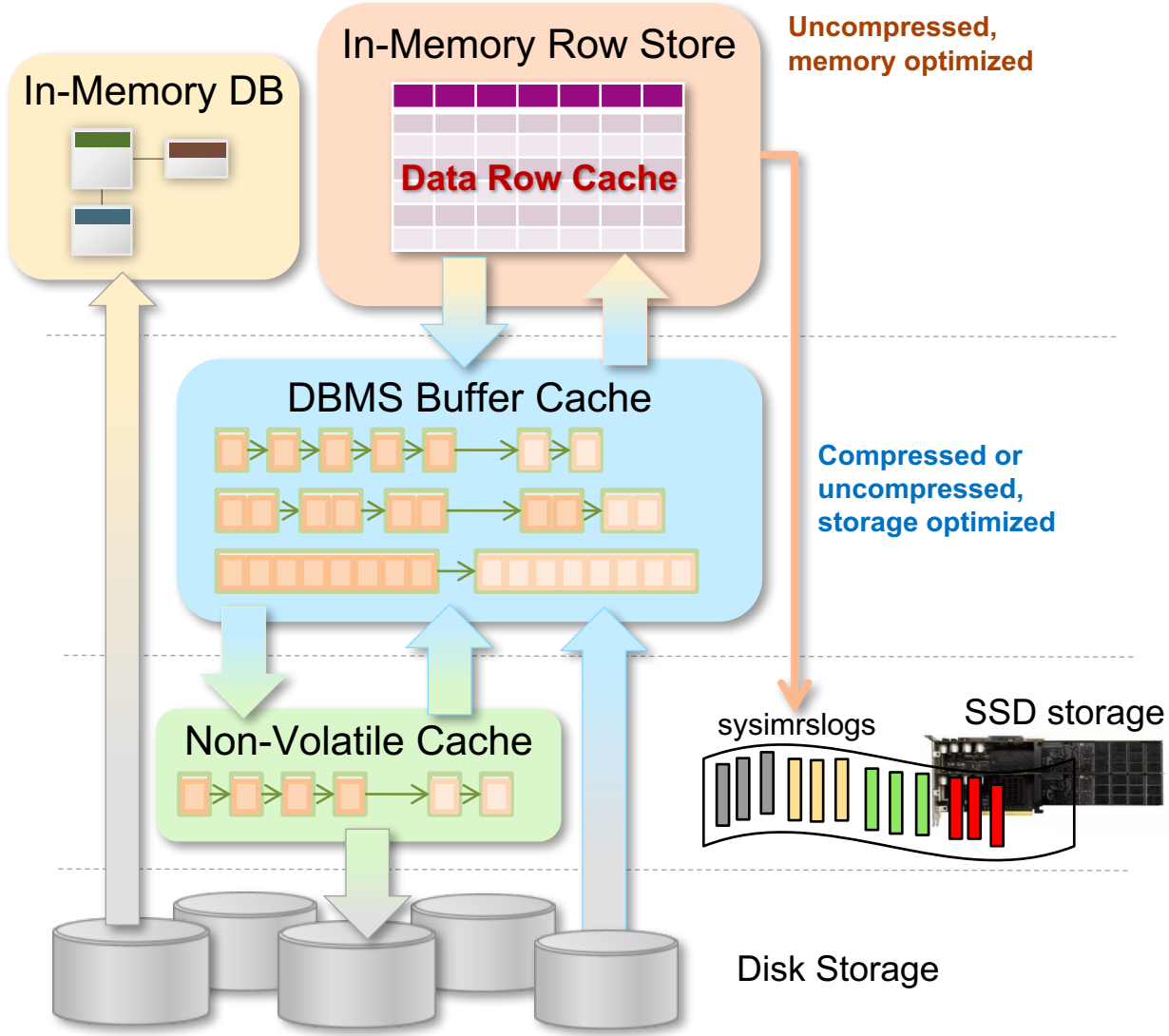
Data Row Caching via the In-Memory Row Store

IMRS as an ILM

- Some of the most active tables are also the largest
  – For example, transaction tables, etc.
- It all won't fit in memory
- Today's data hotter than yesterdays, ….

Data Row Caching

- Frequently accessed data is promoted to DRC
  – Newly inserted rows
  – Frequently selected or frequently updated
- Uses memory optimized cache techniques
  – Row versioning for updates
  – Data is uncompressed for frequent reads/updates
  – Column-oriented layout for frequently read data rows
- Optimized IMRS log for ACID durability/recovery
- As data "cools", it is packed back to normal DBMS buffer cache
- Works in conjunction with MVCC and HCB



ILM / Pack Sub-systems – Optimize Cache Utilization

ILM Objectives

- Keep hot data in IMRS
- Keep cache utilization stable

Moving (possibly) useful data to IMRS

- ILM – ISUD rule engine
- Hot / Cold partition awareness

Moving cold data out of IMRS

- Taxing (cold, fat) partitions by Pack
- Data rows are removed from IMRS and placed back in page store
  – Frees up valuable memory for new hot rows
- Hot Rows filtering by Pack

ILM ISUD Rules Engine:

Every row operation makes storage choice

- Use IMRS or perform op in page-store

Rule to keep possibly Cold data on page store

- Query type analysis
- Result set size

Rules to move hot data to IMRS – Buffer hotness

IMRS Resource – Cache utilization

- Disable migration and new inserts to IMRS if available cache drops

IMRS log free space available

- Disable migration and new inserts to IMRS if log free space <= 10%

Auto Partition Tuning – Efficient IMRS Usage

DB-option – ON by default

Tuning is workload and partition aware

Various partition and workload characteristics considered before disabling / re-enabling IMRS-usage for partition

Decision is performed for (partition, row_type)

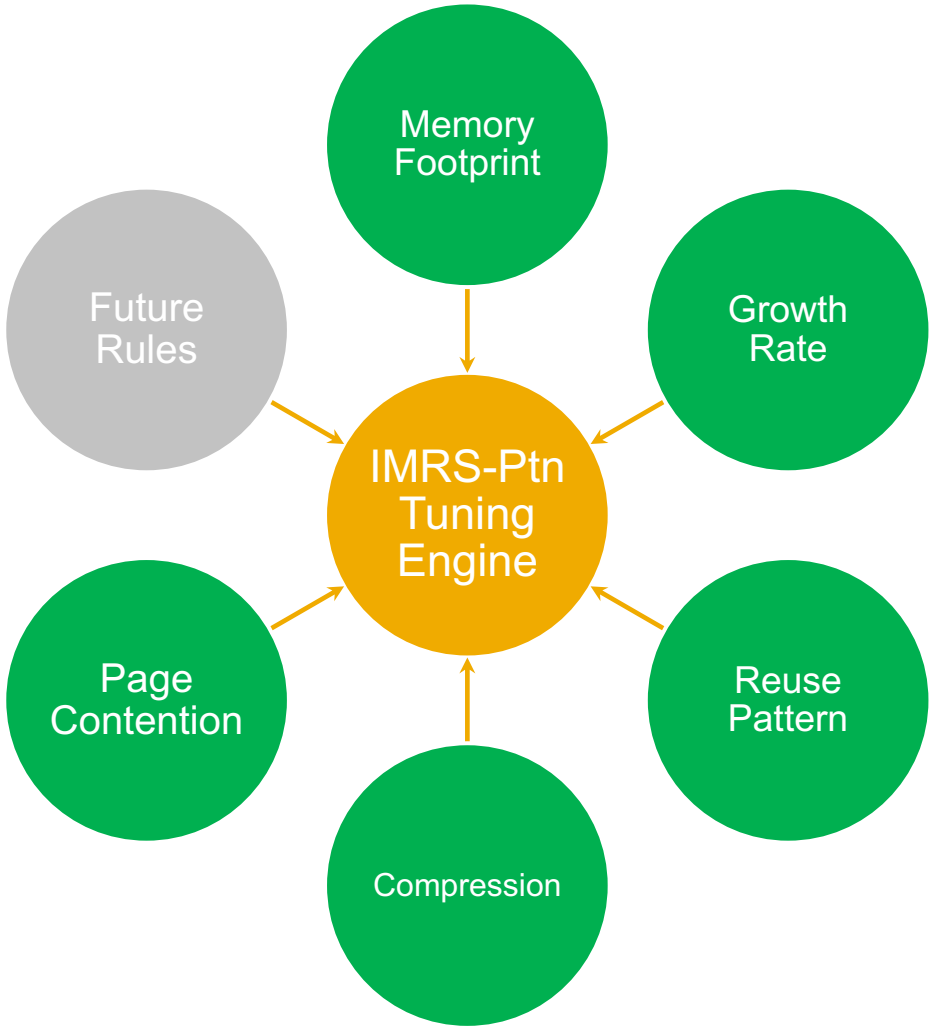
- Row Types: Inserted, Migrated (Updated), Cached

Tuning performed by IMRS Pack threads

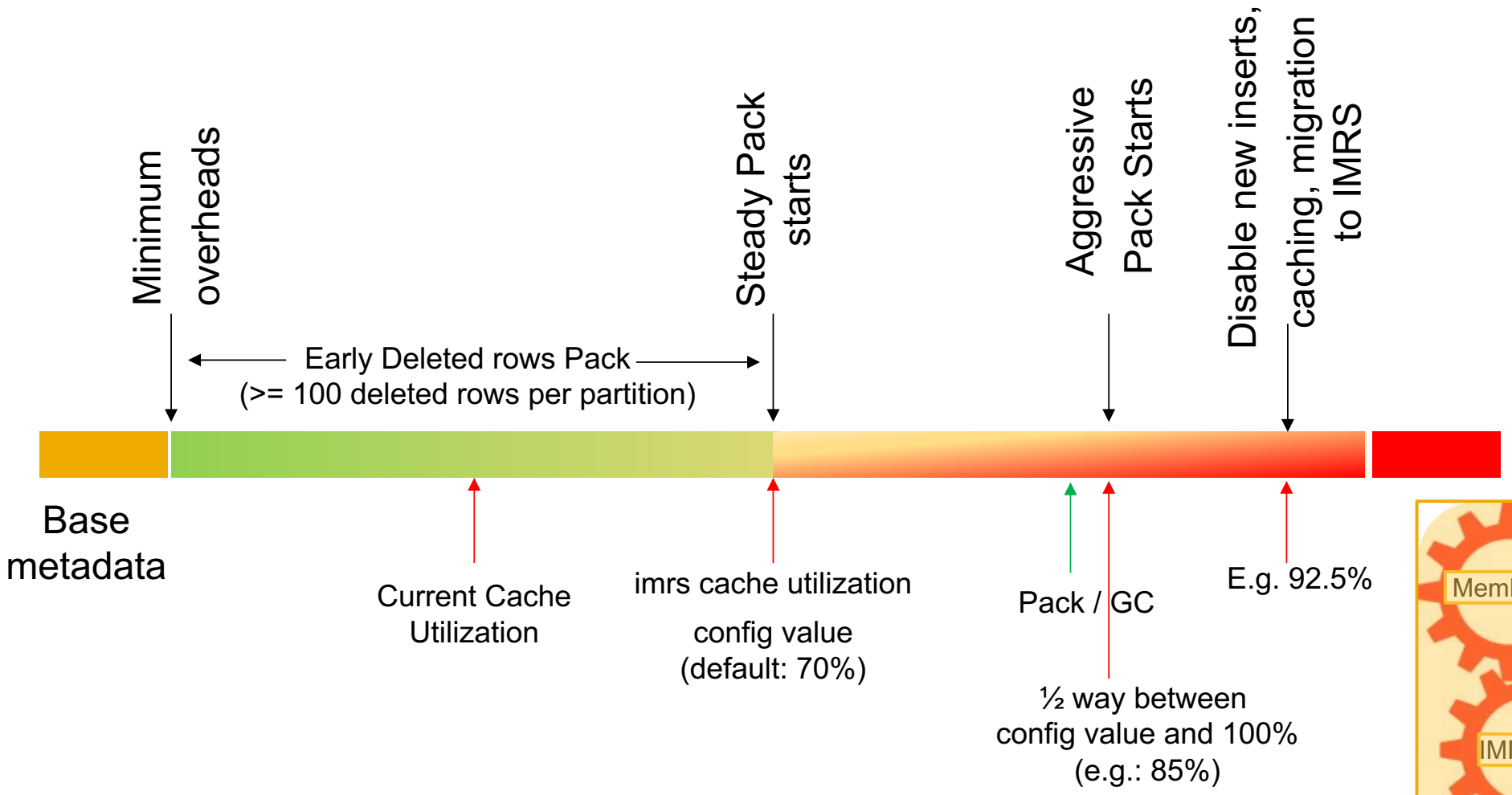
Useful to enable IMRS for all tables in db

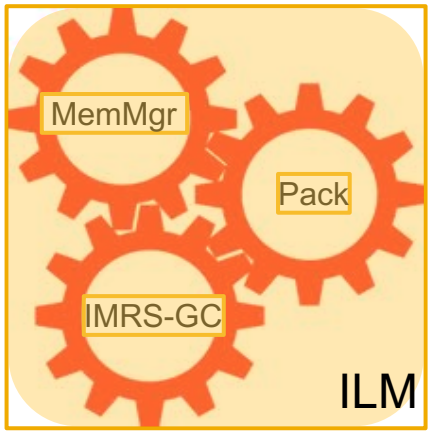
- Less-used tables disabled for IMRS usage

Useful to keep cache utilization optimal



Configuration – Cache Utilization – ILM / Pack dynamics





Hash-Cache BTree (HCB) Indexes

Btree Performance and Scalability Issues

B-Tree (most common)

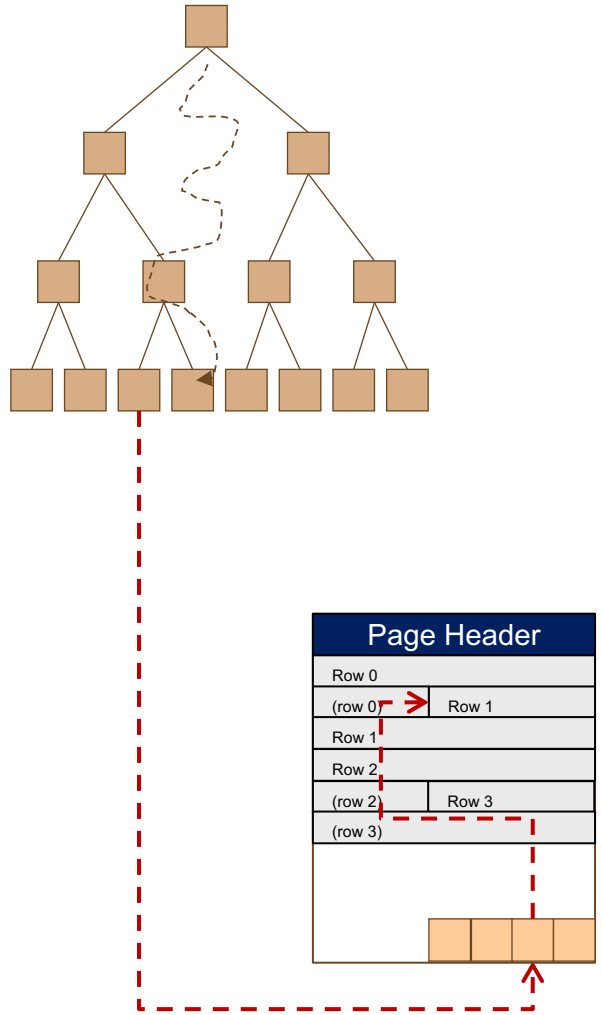
- Typical tree structure
- Nodes are composed of index key values + RID

Index Scans

- On VLDB tables, multi-level tree structures
  – Adds I/O costs during index scans
- Jump from index leaf to data page
  – Navigating inside data page row-formats

Concurrency & Scalability Issues

- Latch conflicts on index root and non-leaf pages
- Updates cause cache-line invalidation
- Latch-Free Btree indexes address issues to large extent



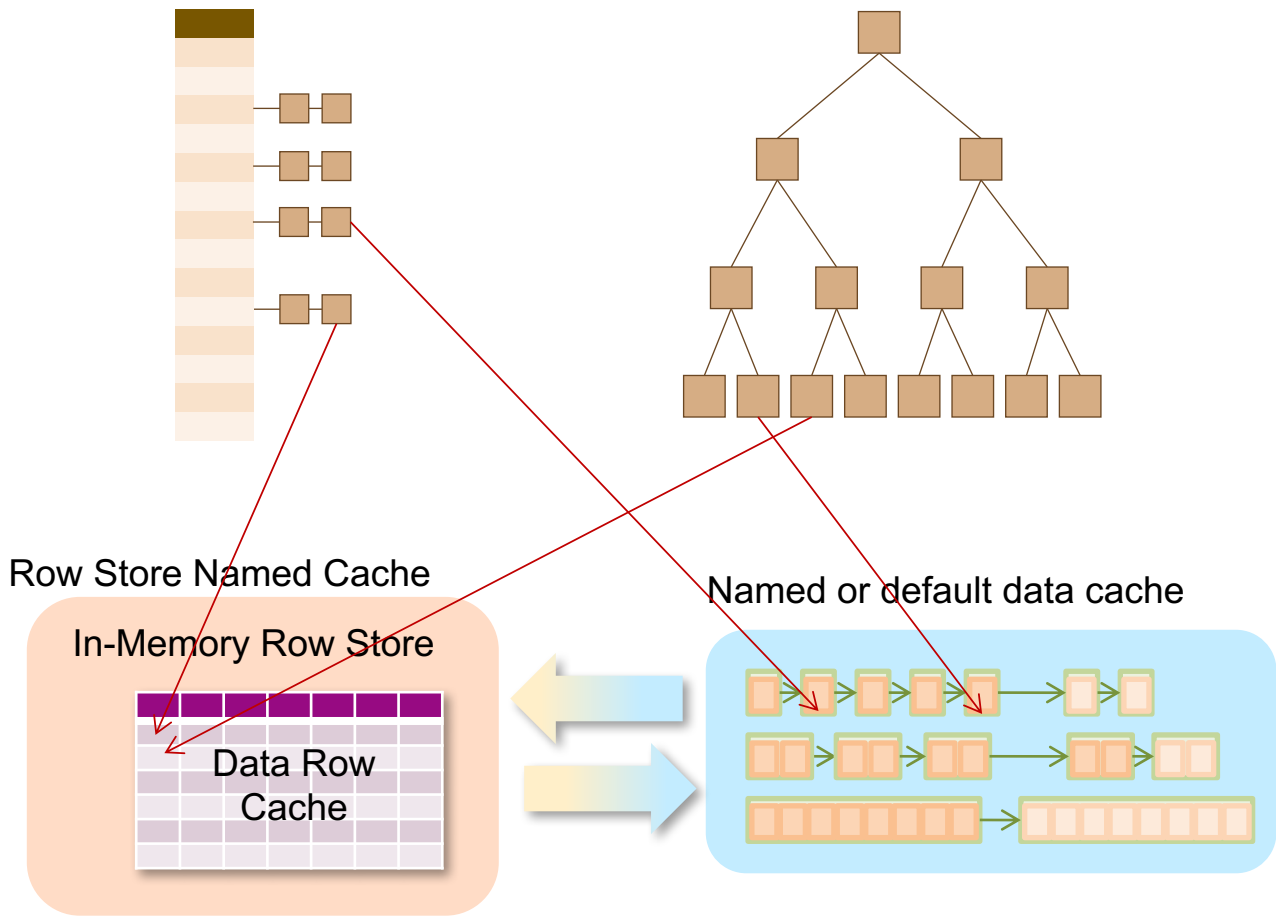
Best of Both Worlds: Hash Cache B-Tree (1)

Hash Cache B-Tree

- Only on unique indexes for IMRS enabled tables
- Rows promoted to IMRS will have unique index keys hashed and added to HCB
  - Hash node points to row in IMRS
- If row is removed from IMRS, hash node is removed from hash table:
  - Occurs when row is deleted
    Or, when row is Packed back to page store

How it works

- If query uses Equi-sargs & unique index
  - Index keys are first checked in HCB; if found, row in either IMRS or page cache is returned
  - This includes Selects, Updates, Deletes & JOINS!!!
- Otherwise, query uses b-tree index
  - For non-unique indexes, this also could point to rows in the IMRS



HCB Index is a transparent, well-tuned, fast-path accelerator under Btree scans

Best of Both Worlds: Hash Cache B-Tree (2)

Advantages over full-table native Hash Indexing

- Since only “hot” rows in IMRS are hash indexed, the number of hash buckets can be smaller yet still have short hash chains for fast access
  - Efficient memory resource requirement
  - For VLDB tables, expect a small “working-set” of hot rows to be in the IMRS
  - Hash chains tend to be small and practically manageable
- Advantages of B-tree index are retained eliminating need for duplicate indexes on key columns
  - Think datetime cols in pkey index and range scans or min/max queries

Advantages over Btree index scans

- Scanning through multiple Btree levels is avoided – Removes index page concurrency issues
- Built with CAS-based lockless hash tables – highly concurrent & scalable hashing machinery

Disadvantages

- Since cache volumes may fluctuate, more difficult to predict the number of hash buckets

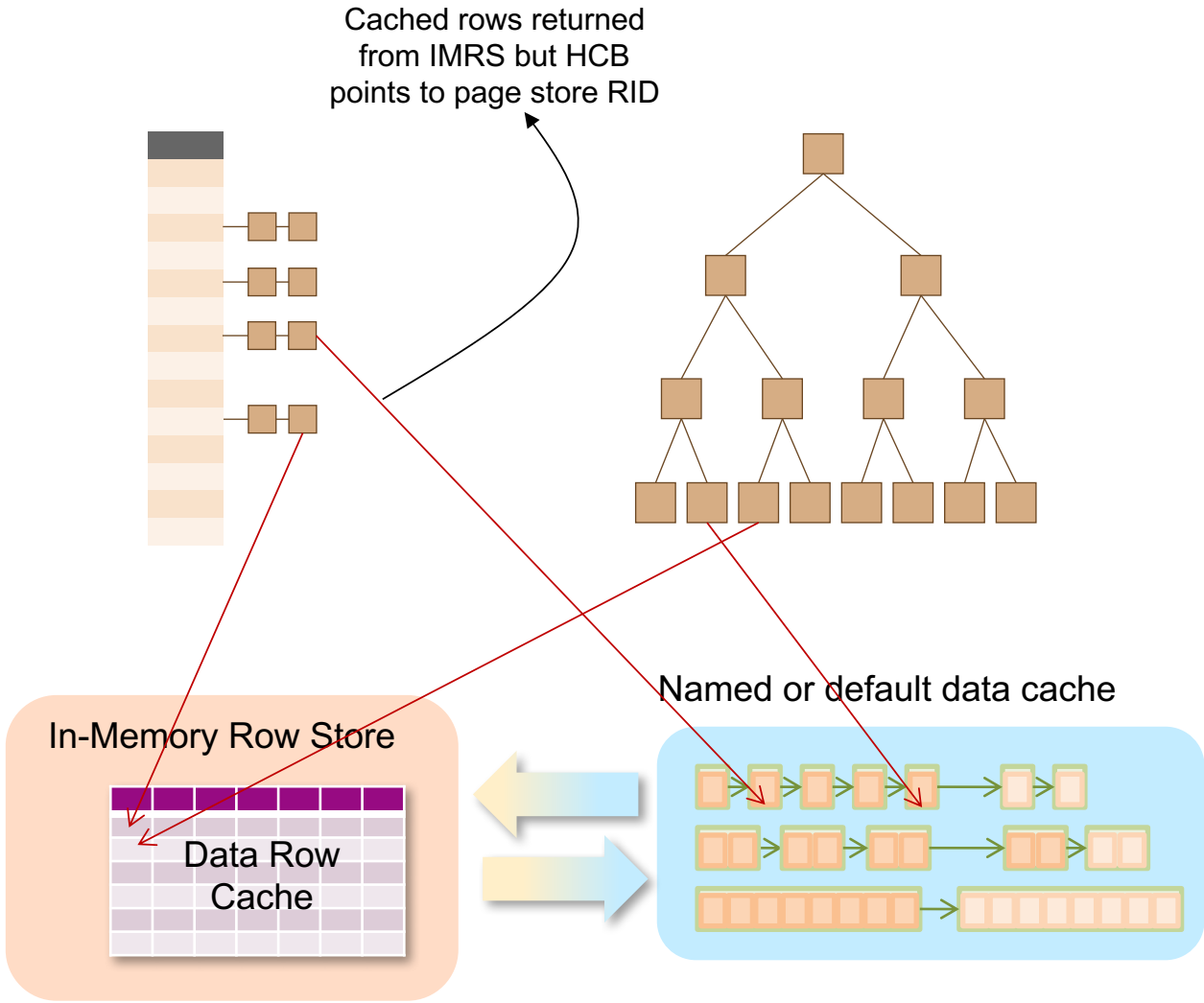
Hash Cache B-Tree Advantages

Reduces Read/Write conflicts

- As there is no concept of a page, it avoids the latch contention between reads & writes

Faster query execution

- Typically, a primary key lookup must traverse the full B-Tree
  - For a single row fetch, this can be 5-6 LIO's
    - Adds to the cache contention
    - If there is index latch contention, query is impacted
  - For joins, it could be 100's or 1000's of iterations
    - One full traversal for each outer row that qualifies
- With HCB
  - Hash operation on keys (no LIO)
  - Hash chain scan
    - Sized correctly, this would be 1-2 at the most



Summary

ASE 160 MemScale option delivers Extreme OLTP Performance

Collection of innovative features addressing different performance & scalability aspects

New technologies seamlessly integrated into existing product base – Easy migration process

- Furthers the low TCO capabilities of ASE

Complementary feature set – Allows for incremental roll-out of specific features

- DRC suited for highly concurrent insert / update workloads
- HCB suited for unique index lookup access
- LFB suited for highly-concurrent index access – Benefits almost all applications
- MVCC suited for mixed workload, concurrent reporting & OLTP applications

New offerings make efficient use of large memory and multi-core hardware

Significant performance gains can be realized depending on your workload & concurrency issues

Upcoming SAP ASE Events

Upcoming webcasts:

Register here:
https://event.on24.com/eventRegistration/EventLobbyServlet?target=reg20.jsp&referrer=&eventid=1480117&sessionid=1&key=EC65E8FCB99D96463A86466A9157D459&regTag=150696&sourcepage=register

September 12, 2017: MemScale Part 2: Multi-Version Concurrency Control, Migration Considerations
October 3, 2017: SAP ASE 16 Cloud Strategy and Offerings Available Today
October 9, 2017: The Debut of SAP Replication Server 16 and its Future Direction
October 24, 2017: The value proposition of SAP HANA Accelerator for ASE 2.0 SP02
November 7, 2017: SAP ASE Workload Analyzer Option 16 SP03 for upgrade, migrations and diagnostics

Upcoming in-person SAP ASE events:

SAP TowerTalk Frankfurt, October 10
Register here: http://events.sap.com/de/sap-tower-talk/de/home

TechSelect New York, October 12
Register here: http://my.isug.com/p/cm/ld/fid=1200

TechSelect London, October 17
Register here: http://www.uksug.com/events/techselect-london-2017-sap-ase-sp03-launch-party

Thank you.

Contact information:

Aditya P. Gurajada
Development Expert, SAP ASE
SAP India, Pune

© 2017 SAP SE or an SAP affiliate company. All rights reserved.