Column-Oriented Database Systems

VLDB 2009 Tutorial

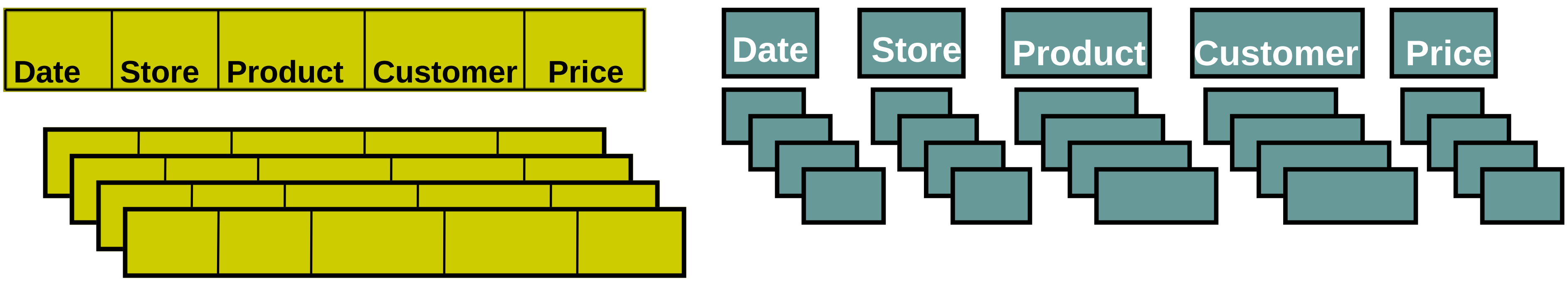
Part 1: Stavros Harizopoulos (HP Labs)

~~Part 2: Daniel Abadi (Yale)~~

~~Part 3: Peter Boncz (CWI)~~

What is a column-store?

row-store | **column-store**



+ easy to add/modify a record

- might read in unnecessary data

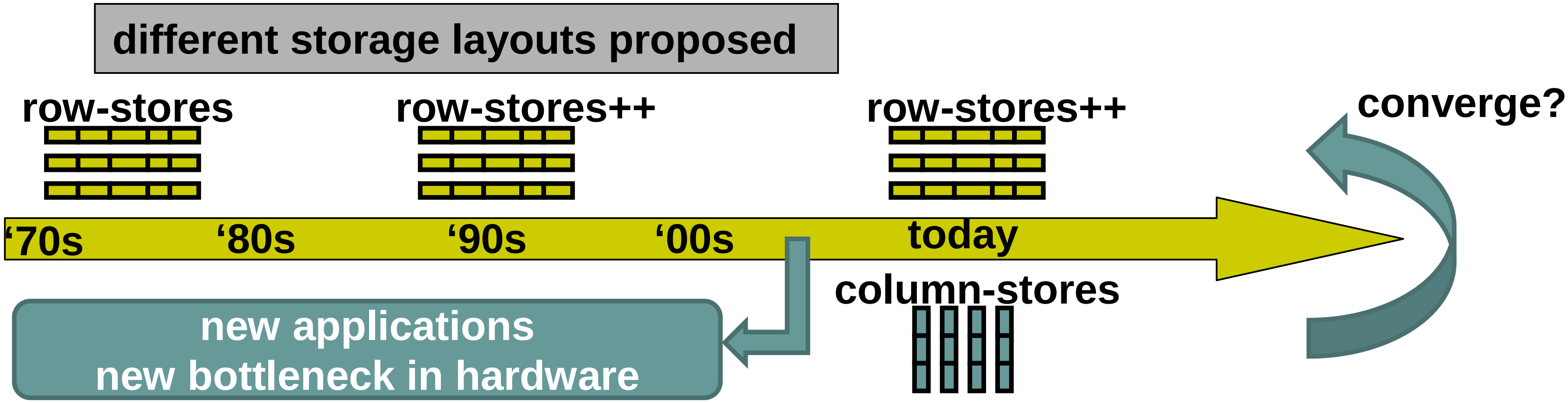
+ only need to read in relevant data

- tuple writes require multiple accesses

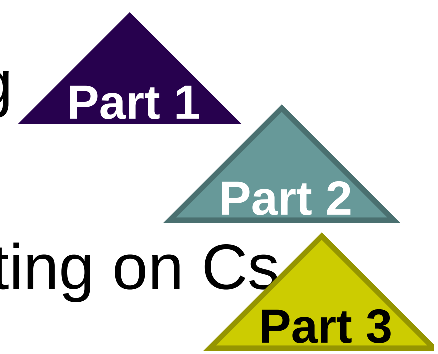
=> suitable for read-mostly, read-intensive, large data repositories

Are these two fundamentally different?

- The only fundamental difference is the storage layout
- However: we need to look at the big picture



- How did we get here, and where we are heading
- What are the column-specific optimizations?
- How do we improve CPU efficiency when operating on Cs

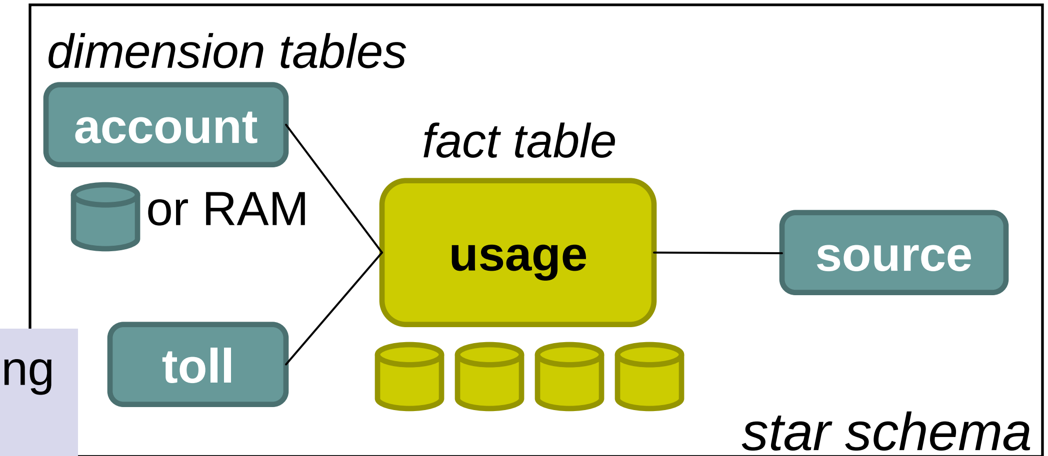


Outline

- Part 1: Basic concepts — *Stavros*
  - Introduction to key features
  - From DSM to column-stores and performance tradeoffs
  - Column-store architecture overview
  - Will rows and columns ever converge?
- Part 2: Column-oriented execution — *Daniel*
- Part 3: MonetDB/X100 and CPU efficiency — *Peter*

Telco Data Warehousing example

- Typical DW installation
- Real-world example



"One Size Fits All? - Part 2: Benchmarking Results" Stonebraker et al. CIDR 2007

QUERY 2

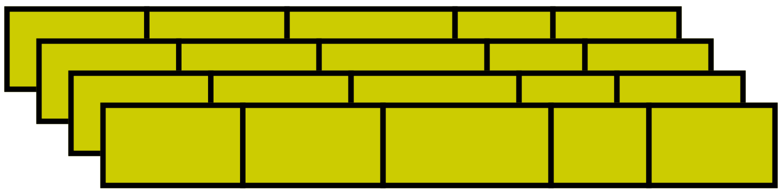
```
SELECT account.account_number,
sum (usage.toll_airtime),
sum (usage.toll_price)
FROM usage, toll, source, account
WHERE usage.toll_id = toll.toll_id
AND usage.source_id = source.source_id
AND usage.account_id = account.account_id
AND toll.type_ind in ('AE'. 'AA')
AND usage.toll_price > 0
AND source.type != 'CIBER'
AND toll.rating_method = 'IS'
AND usage.invoice_date = 20051013
GROUP BY account.account_number
```

	Column-store	*Row-store*
Query 1	*2.06*	*300*
Query 2	*2.20*	*300*
Query 3	*0.09*	*300*
Query 4	*5.24*	*300*
Query 5	*2.88*	*300*

Why? Three main factors (next slides)

Telco example explained (1/3): *read efficiency*

row store



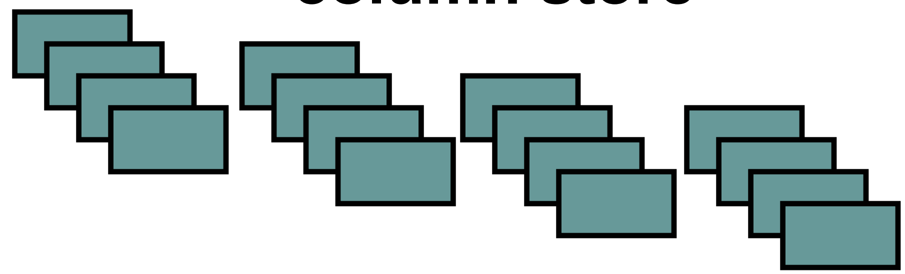
read pages containing entire rows

one row = 212 columns!

is this typical? (it depends)

What about vertical partitioning? (it does not work with ad-hoc queries)

column store



read only columns needed

in this example: 7 columns

caveats:

- “select * ” not any faster
- clever disk prefetching
- clever tuple reconstruction

Telco example explained (2/3): *compression efficiency*

- Columns compress better than rows
  - Typical row-store compression ratio 1 : 3
  - Column-store 1 : 10
- Why?
  - Rows contain values from different domains
    => more entropy, difficult to dense-pack
  - Columns exhibit significantly less entropy
  - Examples:

    Male, Female, Female, Female, Male
    1998, 1998, 1999, 1999, 1999, 2000

  - Caveat: CPU cost (use lightweight compression)

Telco example explained (3/3): *sorting & indexing efficiency*

- Compression and dense-packing free up space
  - Use multiple overlapping column collections
  - Sorted columns compress better
  - Range queries are faster
  - Use sparse clustered indexes

What about heavily-indexed row-stores?
(works well for single column access,
cross-column joins become increasingly expensive)

Additional opportunities for column-stores

- Block-tuple / vectorized processing
  - Easier to build block-tuple operators
    - Amortizes function-call cost, improves CPU cache performance
  - Easier to apply vectorized primitives
    - Software-based: bitwise operations
    - Hardware-based: SIMD

Part 3

- Opportunities with compressed columns
  - *Avoid* decompression: operate directly on compressed
  - *Delay* decompression (and tuple reconstruction)
    - Also known as: *late materialization*

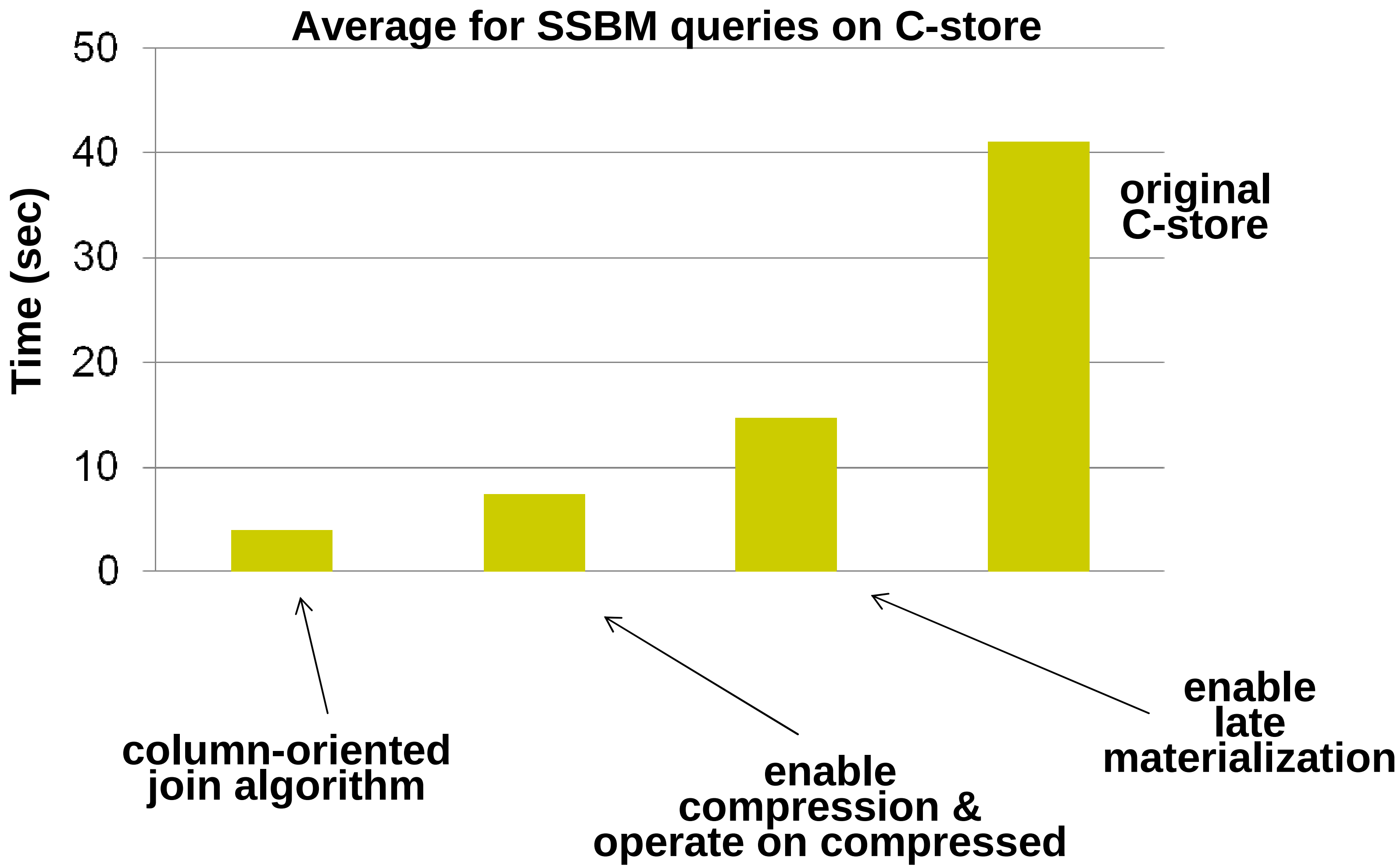
more in Part 2

- Exploit columnar storage in other DBMS components
  - Physical design (both static and dynamic)

See: *Database Cracking*, from CWI

“Column-Stores vs Row-Stores: How Different are They Really?” Abadi, Hachem, and Madden. SIGMOD 2008.

Effect on C-Store performance



Summary of column-store key features

- Storage layout
  - columnar storage (Part 1)
  - header/ID elimination (Part 1)
  - compression (Part 2, Part 3)
  - multiple sort orders
- Execution engine
  - column operators (Part 1, Part 2)
  - avoid decompression (Part 2)
  - late materialization (Part 2)
  - vectorized operations (Part 3)
- Design tools, optimizer

Outline

- Part 1: Basic concepts — *Stavros*
  - Introduction to key features
  - From DSM to column-stores and performance tradeoffs
  - Column-store architecture overview
  - Will rows and columns ever converge?
- Part 2: Column-oriented execution — *Daniel*
- Part 3: MonetDB/X100 and CPU efficiency — *Peter*

From DSM to Column-stores

70s -1985:

TOD: Time Oriented Database – Wiederhold et al. "A Modular, Self-Describing Clinical Databank System," *Computers and Biomedical Research, 1975*
More 1970s: Transposed files, Lorie, Batory, Svensson

"An overview of cantor: a new system for data analysis"
Karasalo, Svensson, SSDBM 1983

1985: DSM paper

"A decomposition storage model"
Copeland and Khoshafian. SIGMOD 1985.

1990s: Commercialization through SybaseIQ

Late 90s – 2000s: Focus on main-memory performance

- DSM "on steroids" [1997 – now] — CWI: MonetDB
- Hybrid DSM/NSM [2001 – 2004] — Wisconsin: PAX, Fractured Mirrors

Michigan: Data Morphing | CMU: Clotho

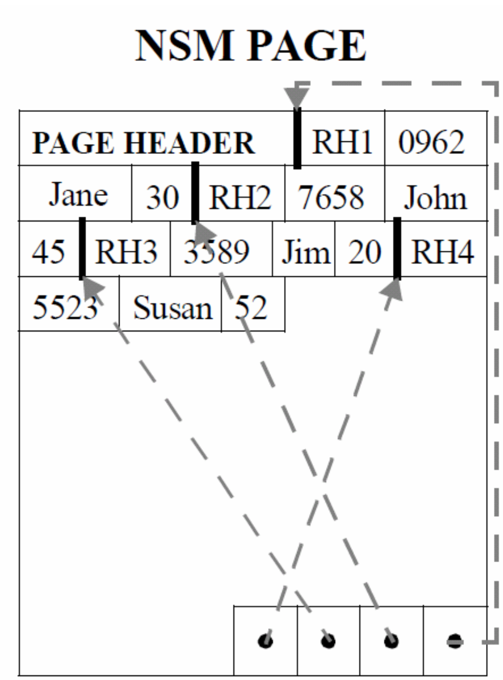
2005 – : Re-birth of read-optimized DSM as "column-store"

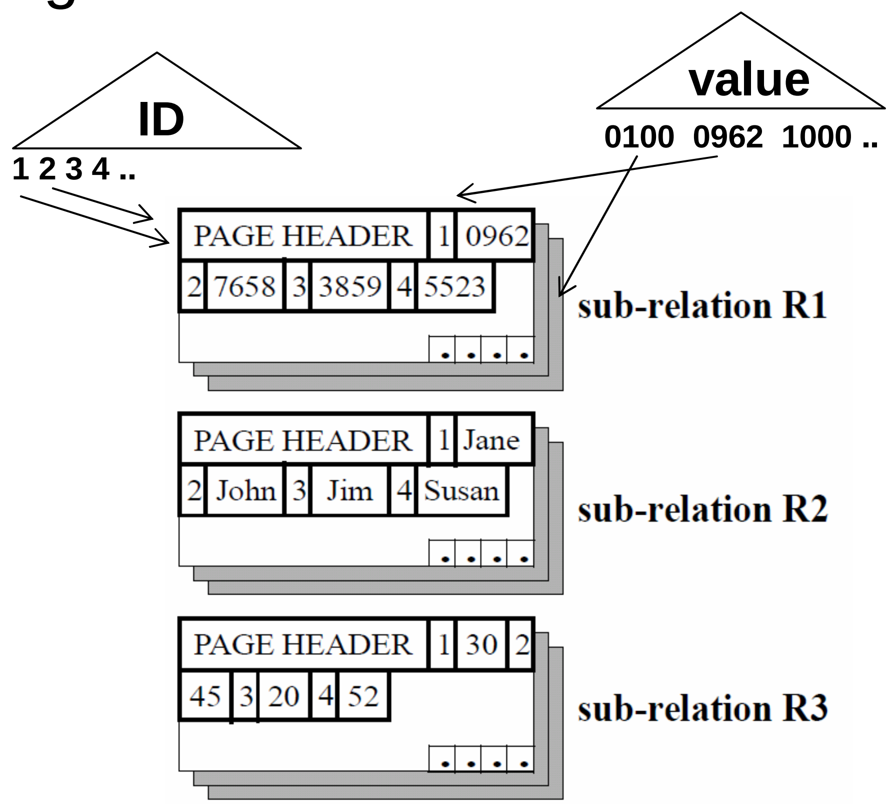
MIT: C-Store | CWI: MonetDB/X100 | 10+ startups

The original DSM paper

"A decomposition storage model" Copeland and Khoshafian. SIGMOD 1985.

- Proposed as an alternative to NSM
- 2 indexes: clustered on ID, non-clustered on value
- Speeds up queries projecting few columns
- Requires more storage





Memory wall and PAX

- 90s: Cache-conscious research

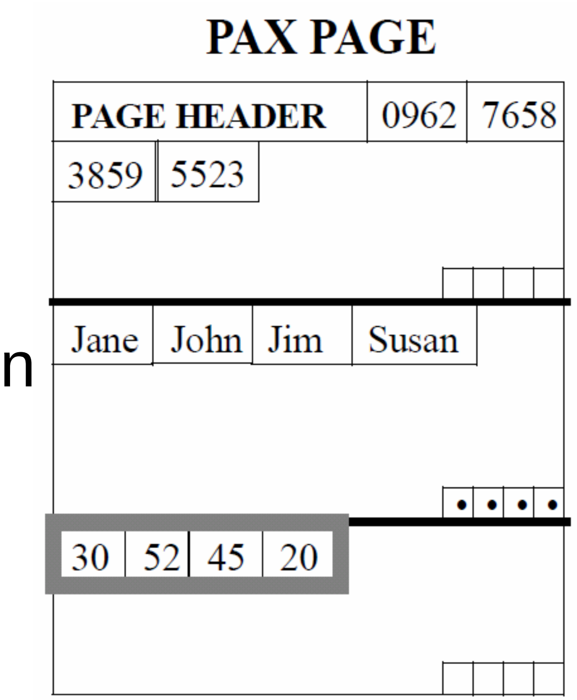
from: "Cache Conscious Algorithms for Relational Query Processing." Shatdal, Kant, Naughton. VLDB 1994.

to: "Database Architecture Optimized for the New Bottleneck: Memory Access." Boncz, Manegold, Kersten. VLDB 1999.

and: "DBMSs on a modern processor: Where does time go?" Ailamaki, DeWitt, Hill, Wood. VLDB 1999.

- PAX: Partition Attributes Across
  - Retains NSM I/O pattern
  - Optimizes cache-to-RAM communication

"Weaving Relations for Cache Performance." Ailamaki, DeWitt, Hill, Skounakis, VLDB 2001.



More hybrid NSM/DSM schemes

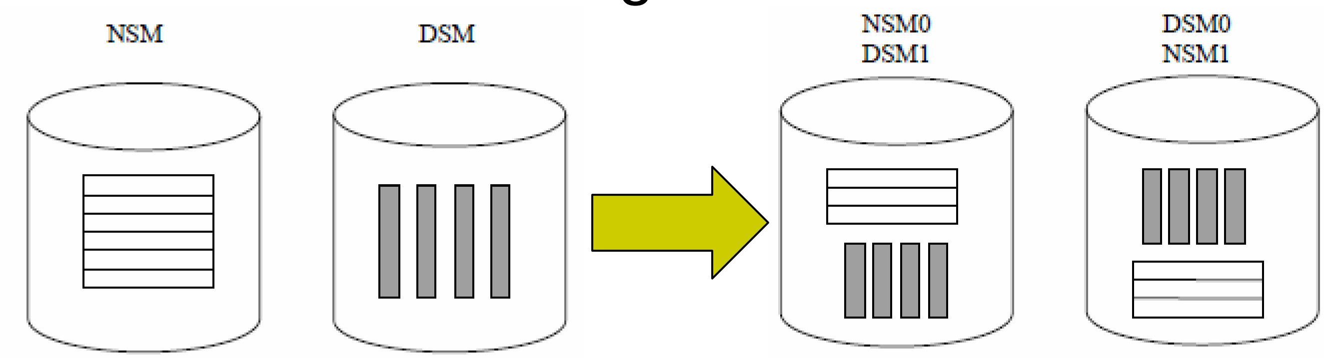
- Dynamic PAX: Data Morphing

"Data morphing: an adaptive, cache-conscious storage technique." Hankins, Patel, VLDB 2003.

- Clotho: custom layout using scatter-gather I/O

"Clotho: Decoupling Memory Page Layout from Storage Organization." Shao, Schindler, Schlosser, Ailamaki, and Ganger. VLDB 2004.

- Fractured mirrors
  - Smart mirroring with both NSM/DSM copies



"A Case For Fractured Mirrors." Ramamurthy, DeWitt, Su, VLDB 2002.

MonetDB (more in Part 3)

- Late 1990s, CWI: Boncz, Manegold, and Kersten
- Motivation:
  - Main-memory
  - Improve computational efficiency by avoiding expression interpreter
  - DSM with virtual IDs natural choice
  - Developed new query execution algebra
- Initial contributions:
  - Pointed out memory-wall in DBMSs
  - Cache-conscious projections and joins
  - …

2005: the (re)birth of column-stores

- New hardware and application realities
  - Faster CPUs, larger memories, disk bandwidth limit
  - Multi-terabyte Data Warehouses
- New approach: combine several techniques
  - Read-optimized, fast multi-column access, disk/CPU efficiency, light-weight compression
- C-store paper:
  - First comprehensive design description of a column-store
- MonetDB/X100
  - “proper” disk-based column store
- Explosion of new products

Performance tradeoffs: columns vs. rows

DSM traditionally was not favored by technology trends
How has this changed?

- Optimized DSM in "Fractured Mirrors," 2002
- "Apples-to-apples" comparison

"Performance Tradeoffs in Read-Optimized Databases" Harizopoulos, Liang, Abadi, Madden, VLDB'06

- Follow-up study

"Read-Optimized Databases, In-Depth" Holloway, DeWitt, VLDB'08

- Main-memory DSM vs. NSM

"DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing" Boncz, Zukowski, Nes, DaMoN'08

- Flash-disks: a come-back for PAX?

"Fast Scans and Joins Using Flash Drives" Shah, Harizopoulos, Wiener, Graefe. DaMoN'08

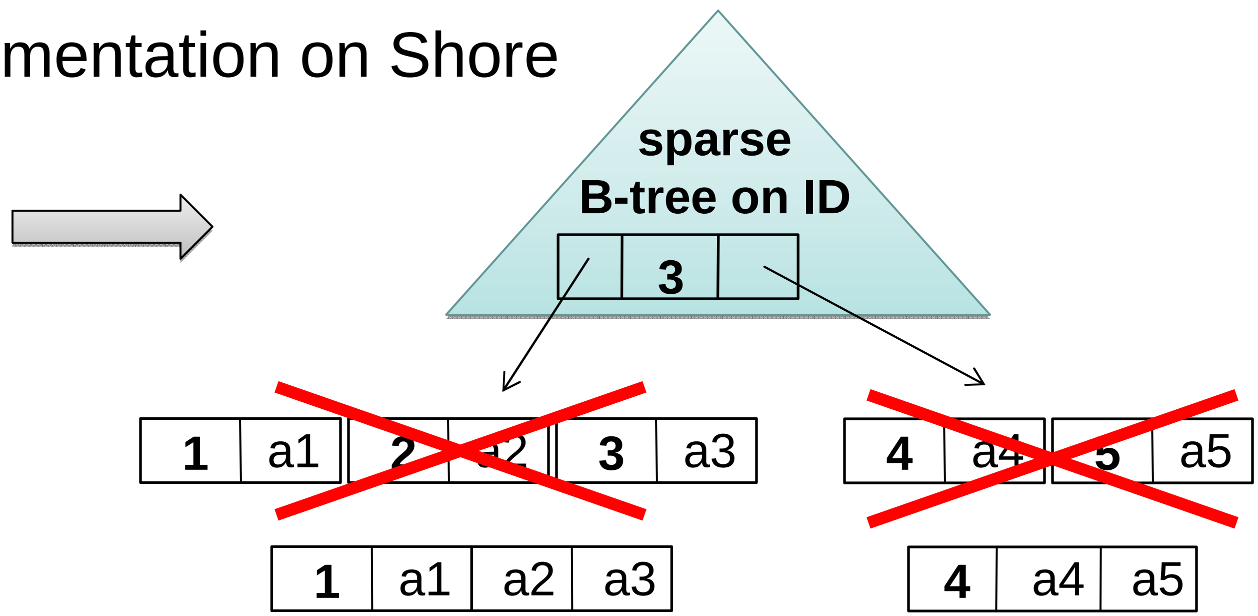
"Query Processing Techniques for Solid State Drives" Tsirogiannis, Harizopoulos, Shah, Wiener, Graefe, SIGMOD'09

Fractured mirrors: a closer look

- Store DSM relations inside a B-tree
  - Leaf nodes contain values
  - Eliminate IDs, amortize header overhead
  - Custom implementation on Shore

"A Case For Fractured Mirrors" Ramamurthy, DeWitt, Su, VLDB 2002.

Tuple Header	TID	Column Data
	1	a1
	2	a2
	3	a3
	4	a4
	5	a5

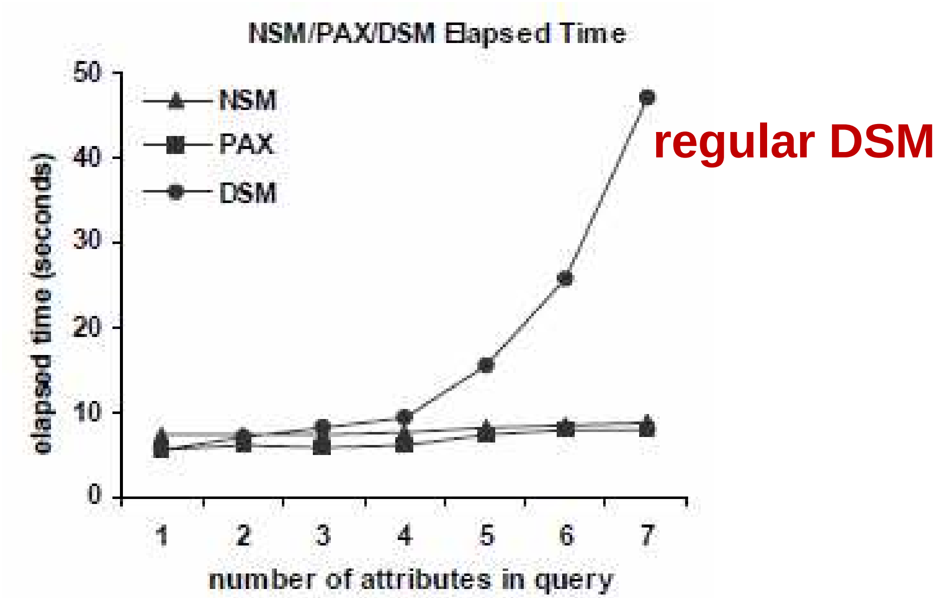


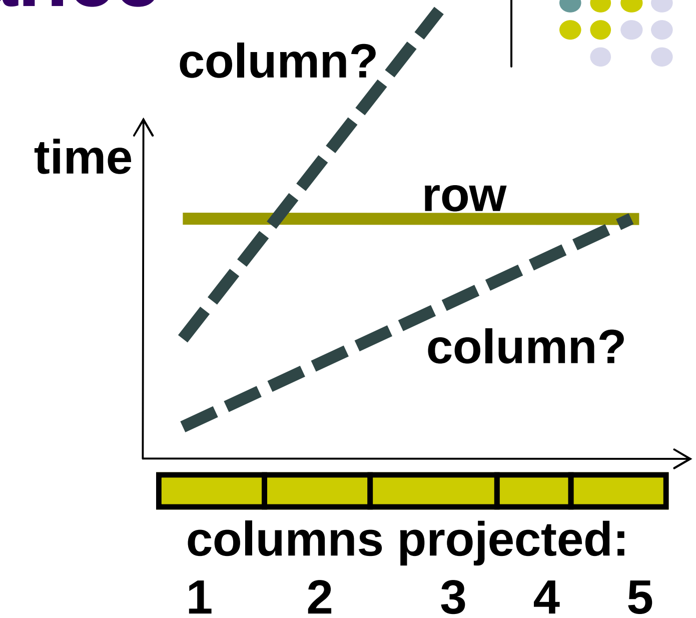
Similar: storage density comparable to column stores

"Efficient columnar storage in B-trees" Graefe. Sigmod Record 03/2007.

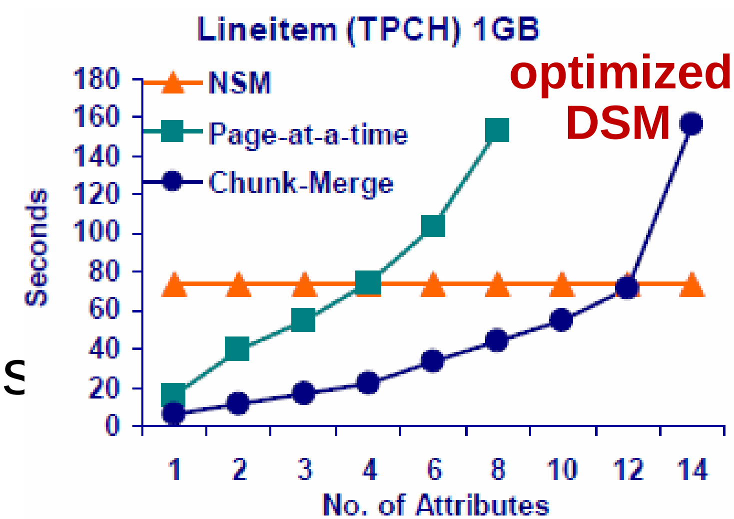
Fractured mirrors: performance

From PAX paper:



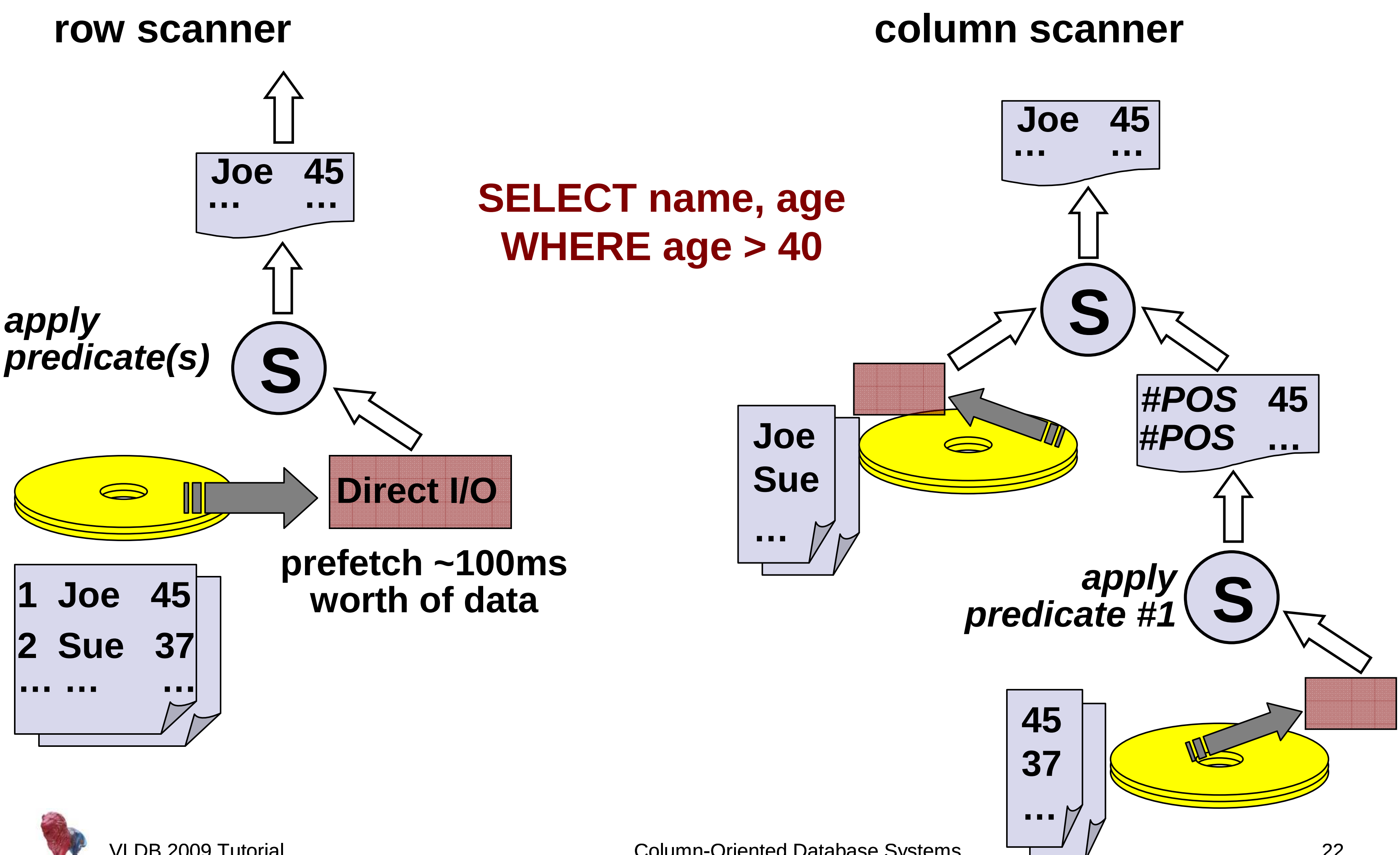


- Chunk-based tuple merging
  - Read in segments of M pages
  - Merge segments in memory
  - Becomes CPU-bound after 5 pages



Column-scanner implementation

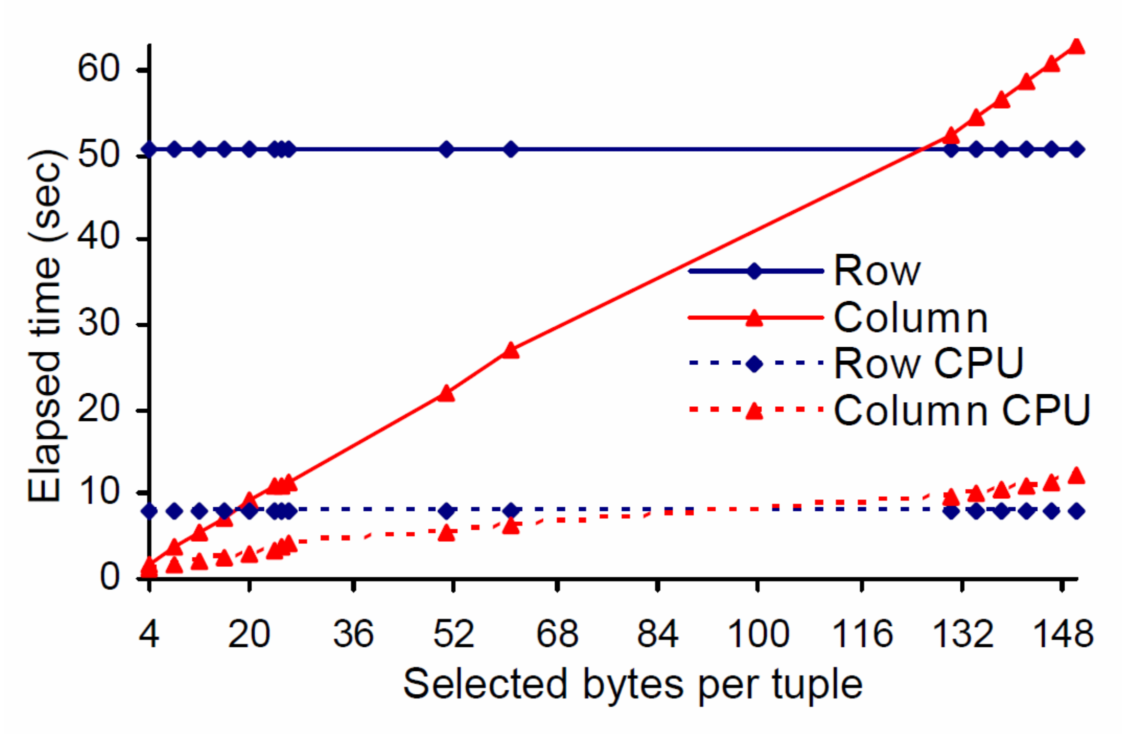
"Performance Tradeoffs in Read-Optimized Databases"
Harizopoulos, Liang, Abadi, Madden, VLDB'06



Scan performance

- Large prefetch hides disk seeks in columns
- Column-CPU efficiency with lower selectivity
- Row-CPU suffers from memory stalls
- Memory stalls disappear in narrow tuples
- Compression: similar to narrow

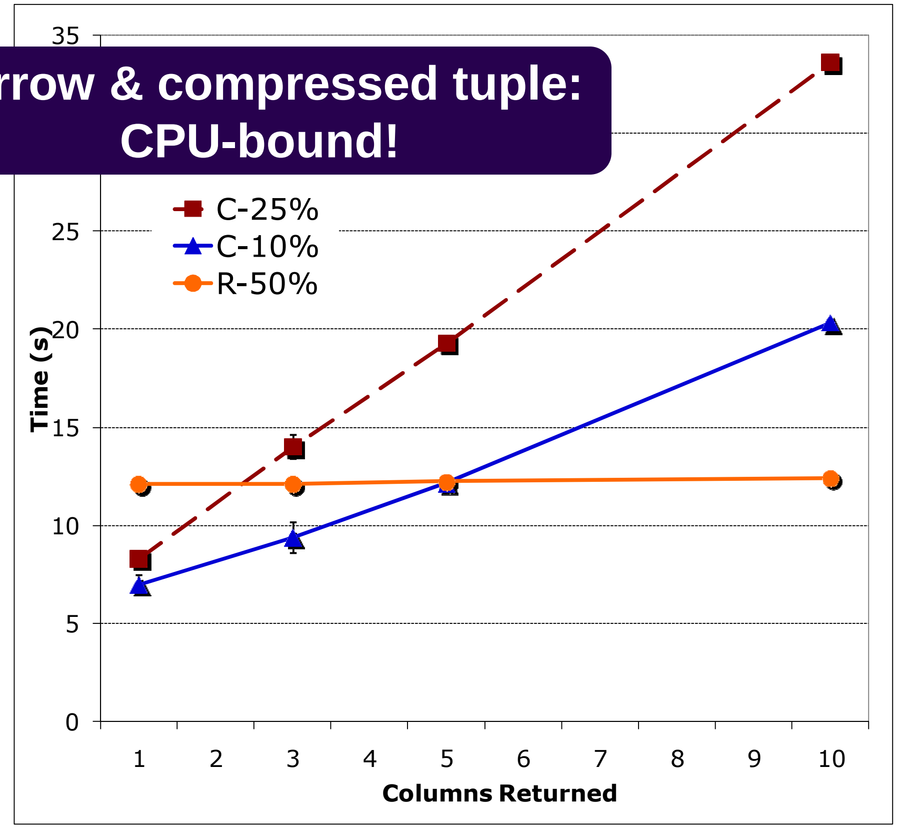
not shown, details in the paper



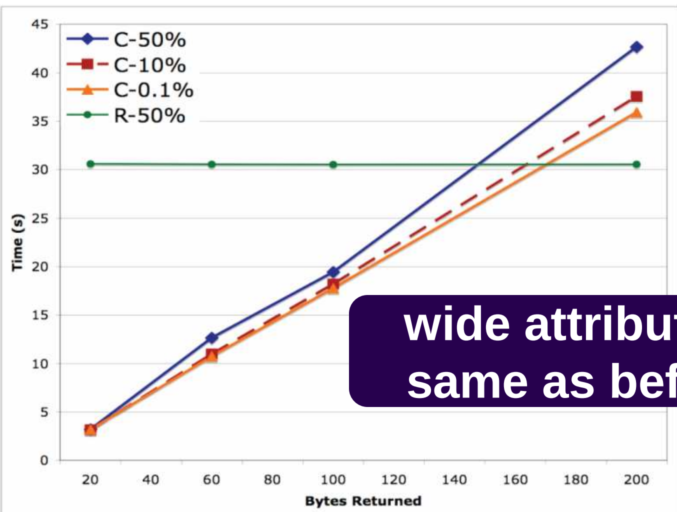
Even more results

"Read-Optimized Databases, In-Depth" Holloway, DeWitt, VLDB'08

- Same engine as before
- Additional findings



narrow & compressed tuple: CPU-bound!



wide attributes: same as before

Non-selective queries, narrow tuples, favor well-compressed rows

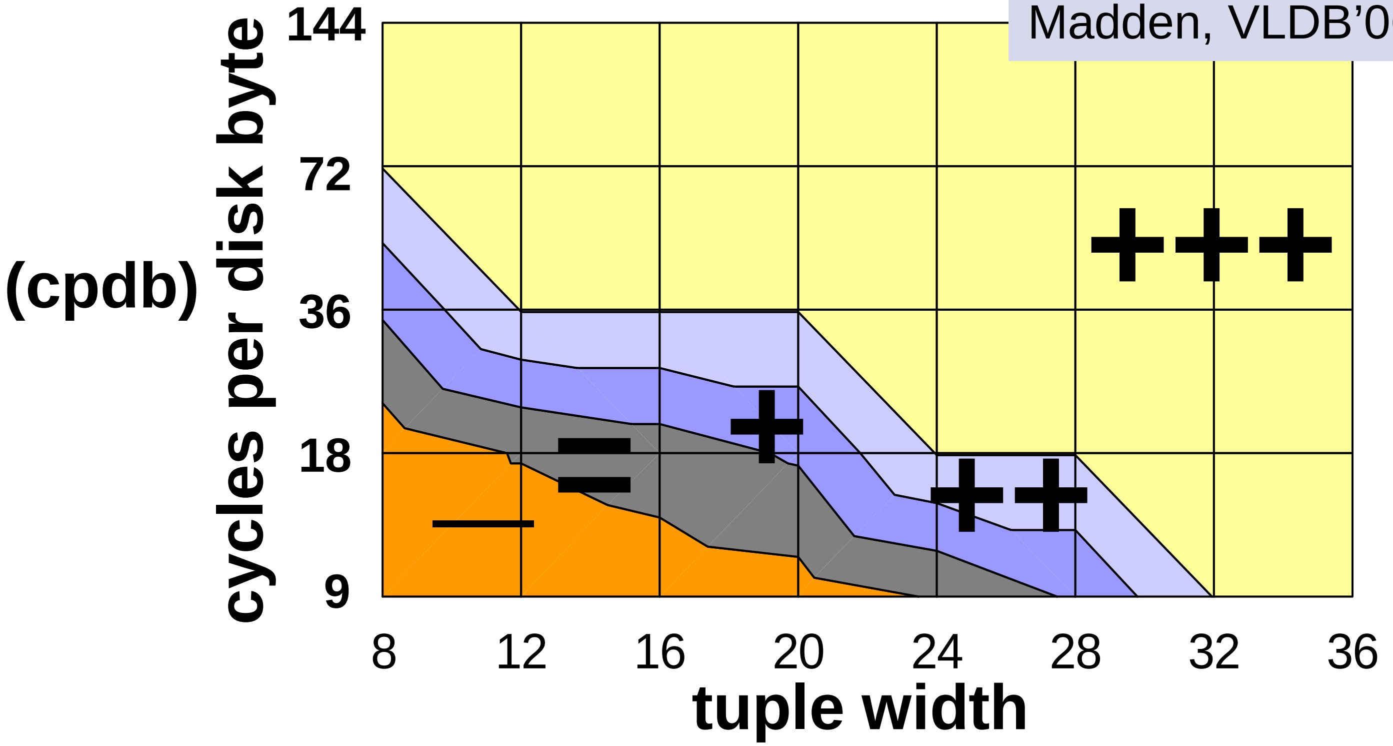
Materialized views are a win

Scan times determine early materialized joins

Column-joins are covered in part 2!

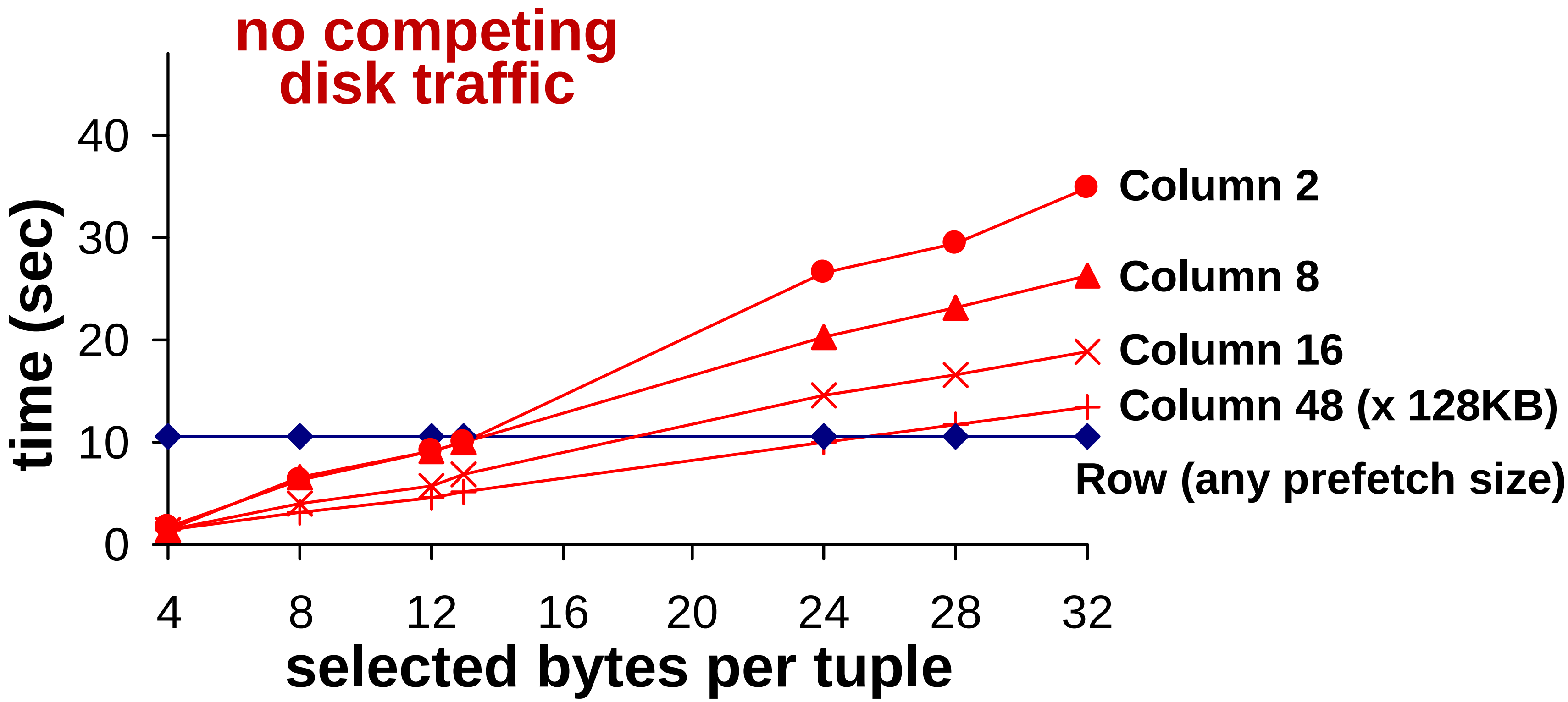
Speedup of columns over rows

"Performance Tradeoffs in Read-Optimized Databases"
Harizopoulos, Liang, Abadi, Madden, VLDB'06



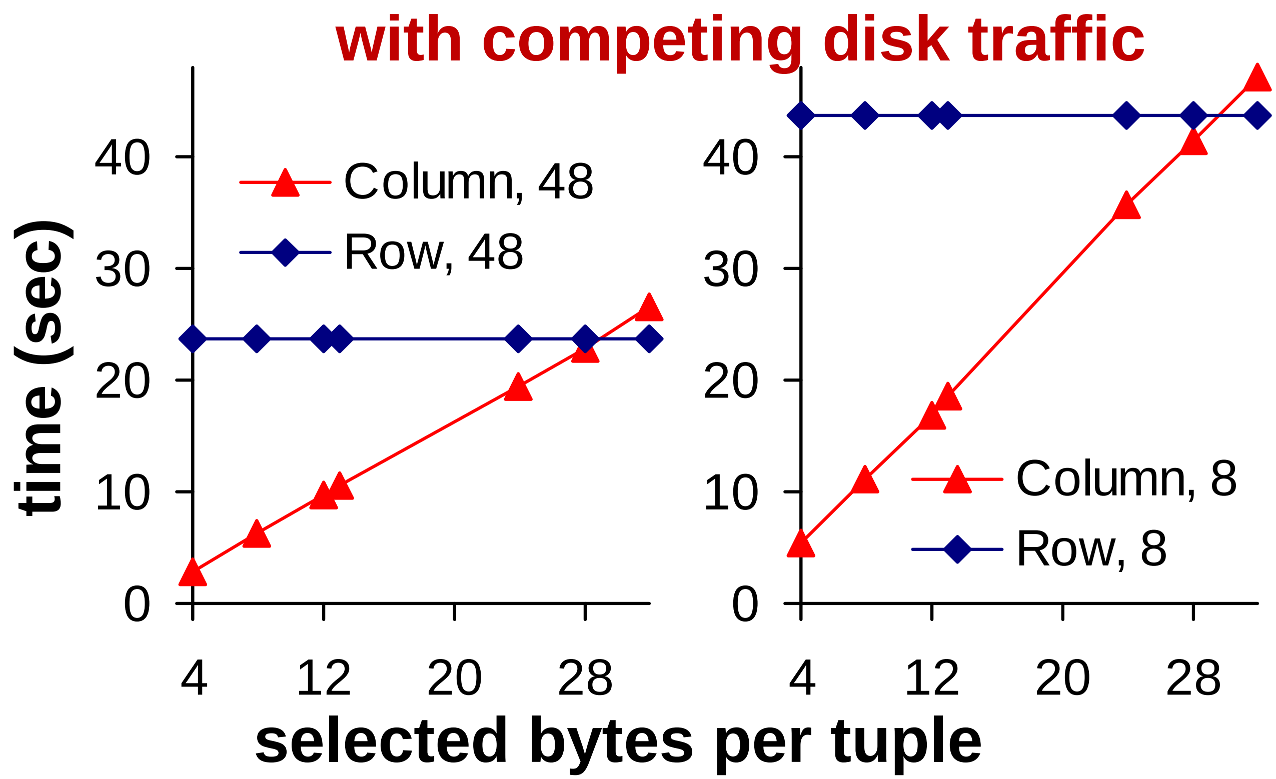
- Rows favored by narrow tuples and low *cpdb*
  - Disk-bound workloads have higher *cpdb*

Varying prefetch size



- No prefetching hurts columns in single scans

Varying prefetch size



- No prefetching hurts columns in single scans
- Under competing traffic, columns outperform rows for any prefetch size

CPU Performance

"DSM vs. NSM: CPU performance trade offs in block-oriented query processing"
Boncz, Zukowski, Nes, DaMoN'08

- Benefit in on-the-fly conversion between NSM and DSM
- DSM: sequential access (block fits in L2), random in L1
- NSM: random access, SIMD for grouped Aggregation

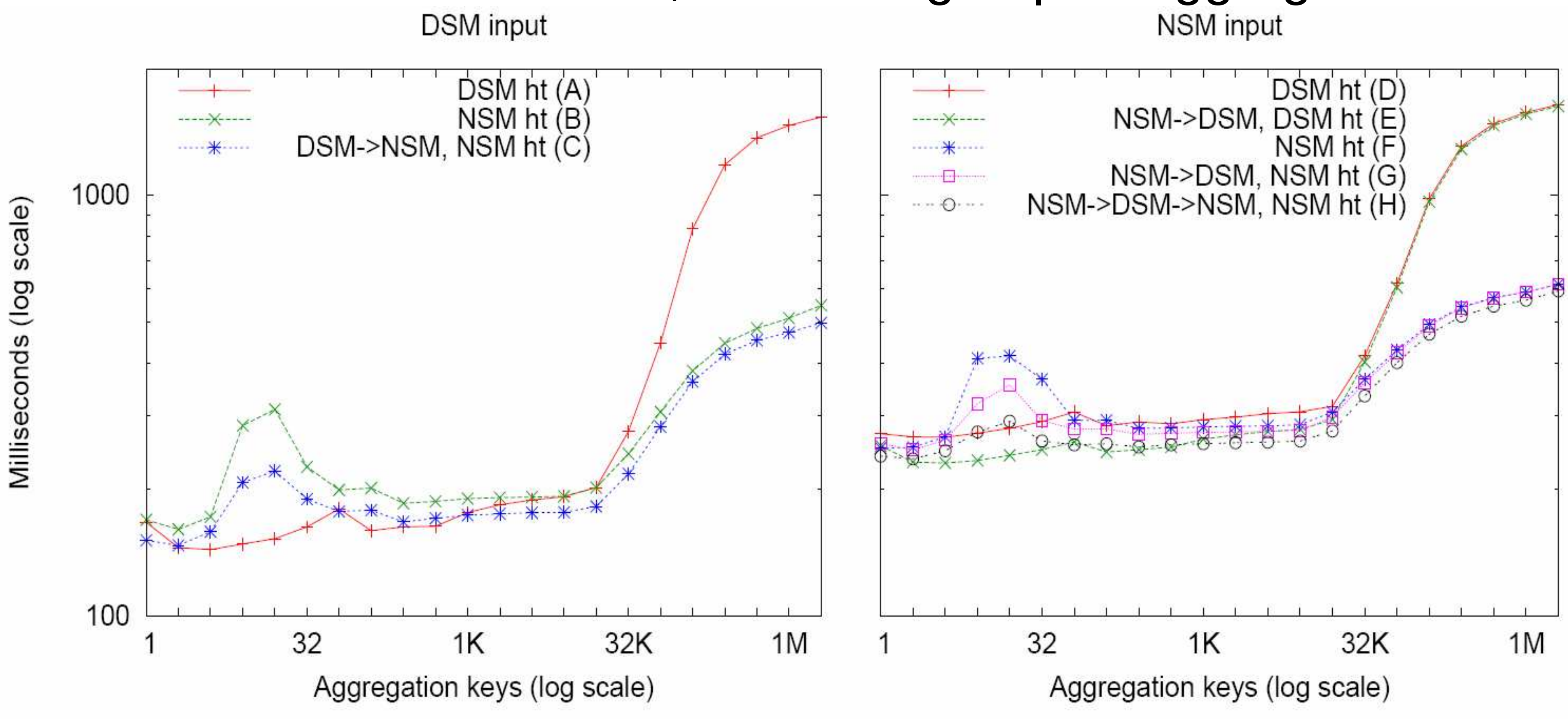


Figure 5: TPC-H Q1, with a varying number of keys and different data organizations (ht – hash table)

New storage technology: Flash SSDs

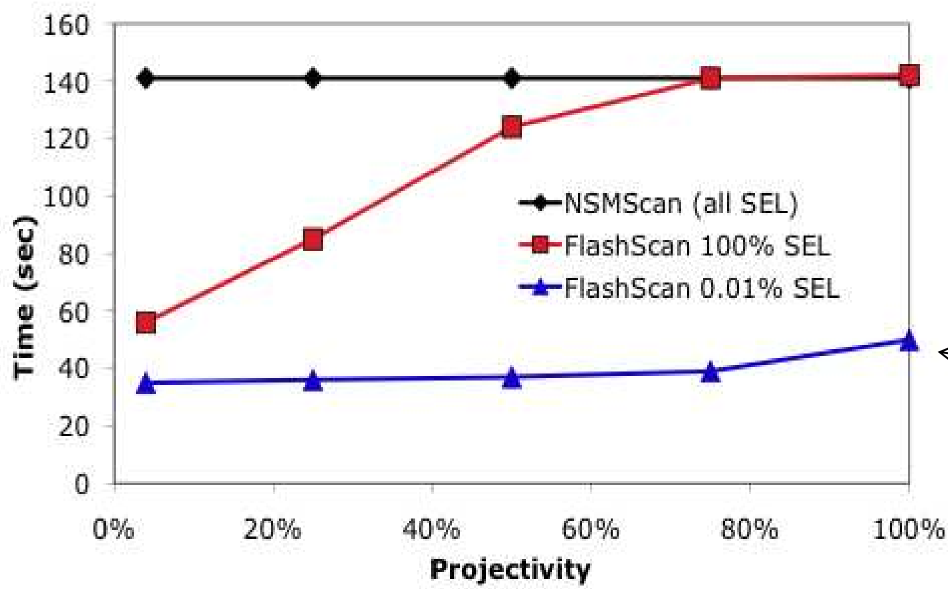
- Performance characteristics
  - very fast random reads, slow random writes
  - fast sequential reads and writes
- Price per bit (capacity follows)
  - cheaper than RAM, order of magnitude more expensive than Disk
- Flash Translation Layer introduces unpredictability
  - avoid random writes!
- Form factors not ideal yet
  - SSD (Ł small reads still suffer from SATA overhead/OS limitations)
  - PCI card (Ł high price, limited expandability)

- Boost Sequential I/O in a simple package
  - Flash RAID: very tight bandwidth/cm^3 packing (4GB/sec inside the box)
- Column Store Updates
  - useful for delta structures and logs
- Random I/O on flash fixes unclustered index access
  - still suboptimal if I/O block size > record size
  - therefore column stores profit mush less than horizontal stores
- Random I/O useful to exploit secondary, tertiary table orderings
  - the larger the data, the deeper clustering one can exploit

Even faster column scans on flash SSDs

30K Read IOps, 3K Write Iops
250MB/s Read BW, 200MB/s Write

- New-generation SSDs
  - Very fast random reads, slower random writes
  - Fast sequential RW, comparable to HDD arrays
- No expensive seeks across columns
- FlashScan and Flashjoin: PAX on SSDs, inside Postgres

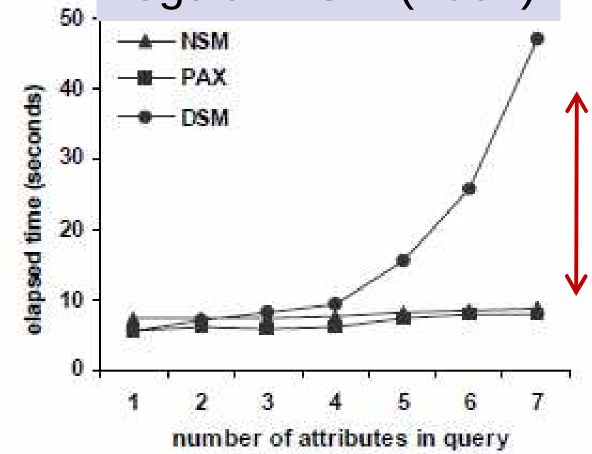


"Query Processing Techniques for Solid State Drives" Tsirogiannis, Harizopoulos, Shah, Wiener, Graefe, SIGMOD'09

mini-pages with no qualified attributes are not accessed

Column-scan performance over time

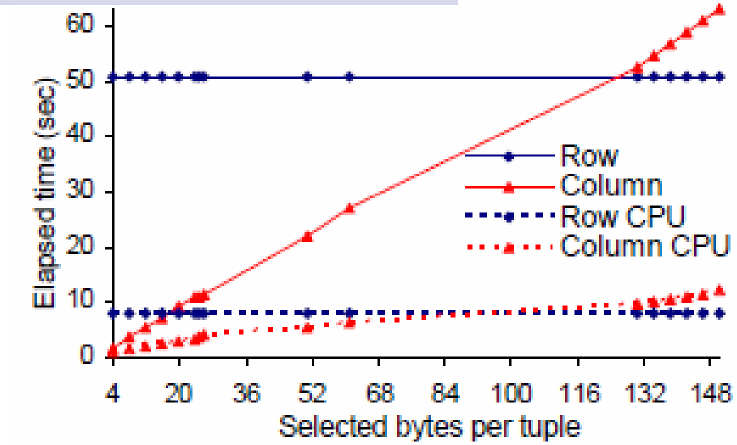
regular DSM (2001)

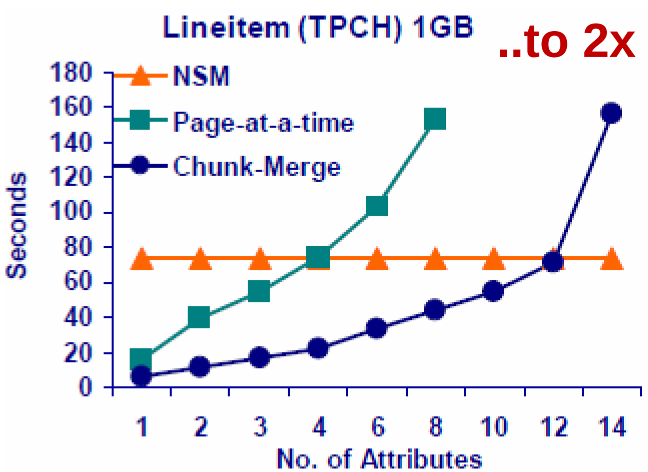


from 7x slower

column-store (2006)

..to 1.2x slower



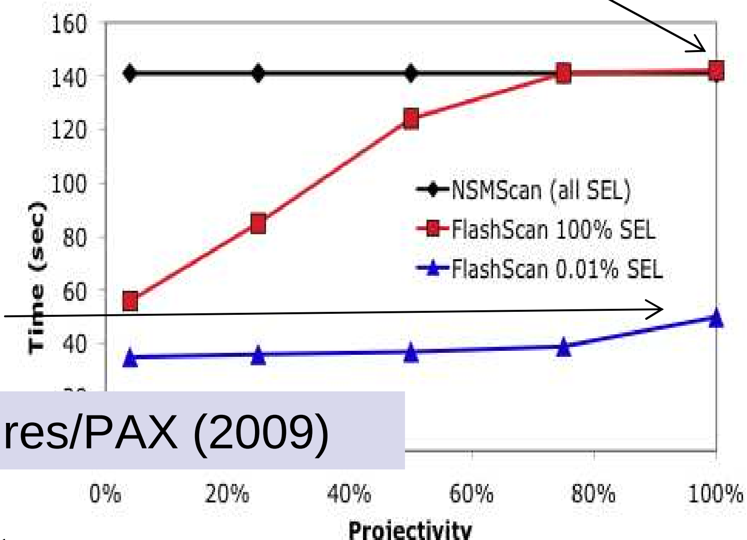


..to 2x slower

optimized DSM (2002)

..to same

and 3x faster!



SSD Postgres/PAX (2009)

Outline

- Part 1: Basic concepts — *Stavros*
  - Introduction to key features
  - From DSM to column-stores and performance tradeoffs
  - Column-store architecture overview
  - Will rows and columns ever converge?
- Part 2: Column-oriented execution — *Daniel*
- Part 3: MonetDB/X100 and CPU efficiency — *Peter*

Architecture of a column-store

storage layout

- read-optimized: dense-packed, compressed
- organize in extends, batch updates
- multiple sort orders
- sparse indexes

engine

- block-tuple operators
- new access methods
- optimized relational operators

system-level

- system-wide column support
- loading / updates
- scaling through multiple nodes
- transactions / redundancy

C-Store

“C-Store: A Column-Oriented DBMS.” Stonebraker et al. VLDB 2005.

- Compress columns
- No alignment
- Big disk blocks
- Only materialized views (perhaps many)
- Focus on Sorting not indexing
- Data ordered on anything, not just time
- Automatic physical DBMS design
- Optimize for grid computing
- Innovative redundancy
- Xacts – but no need for Mohan
- Column optimizer and executor

C-Store: only materialized views (MVs)

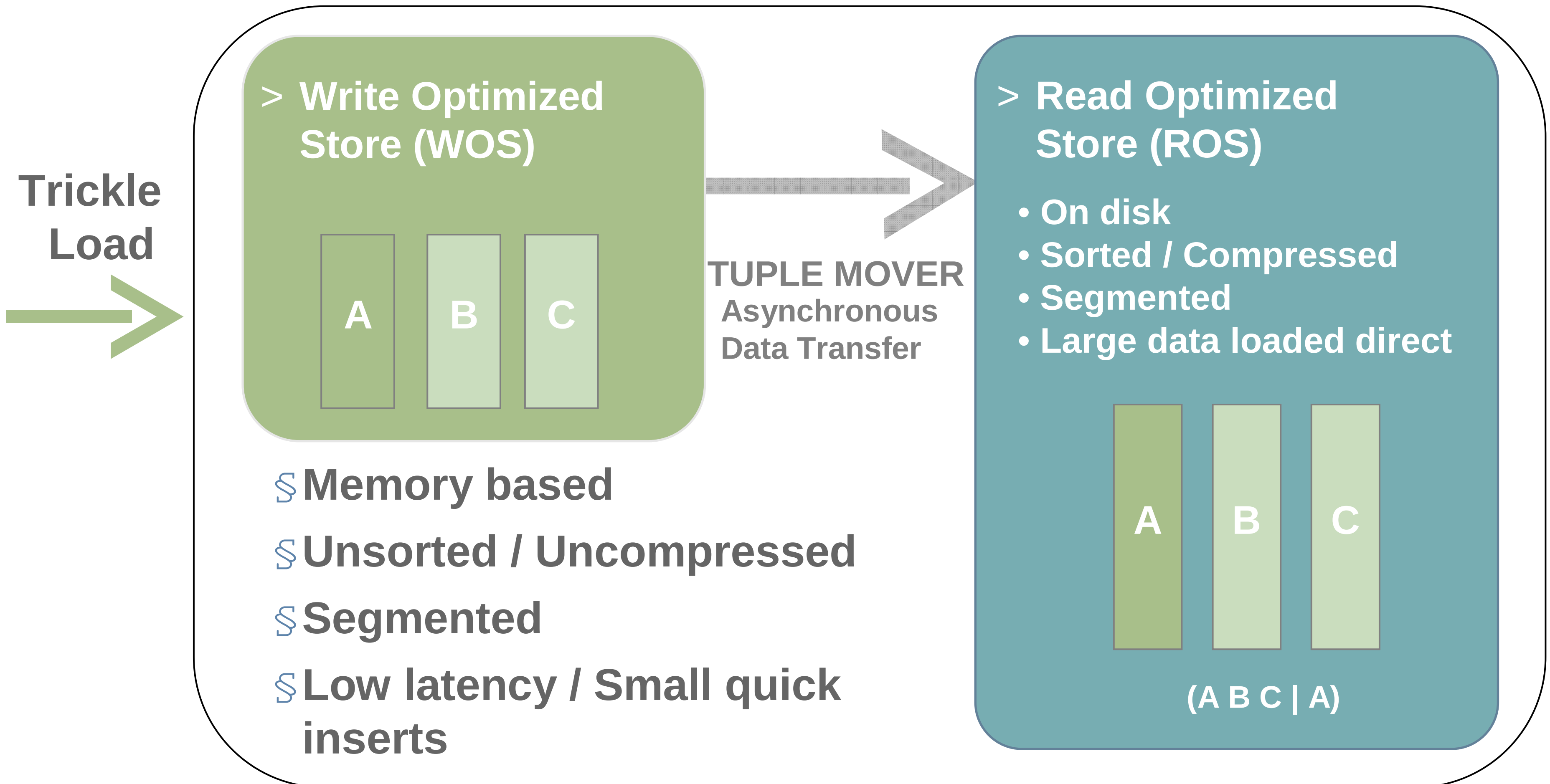
- Projection (MV) is some number of columns from a fact table
- Plus columns in a dimension table – with a 1-n join between Fact and Dimension table
- Stored in order of a storage key(s)
- Several may be stored!
- With a permutation, if necessary, to map between them
- Table (as the user specified it and sees it) is not stored!
- No secondary indexes (they are a one column sorted MV plus a permutation, if you really want one)

User view:
EMP (name, age, salary, dept)
Dept (dname, floor)

Possible set of MVs:
MV-1 (name, dept, floor) in floor order
MV-2 (salary, age) in age order
MV-3 (dname, salary, name) in salary order

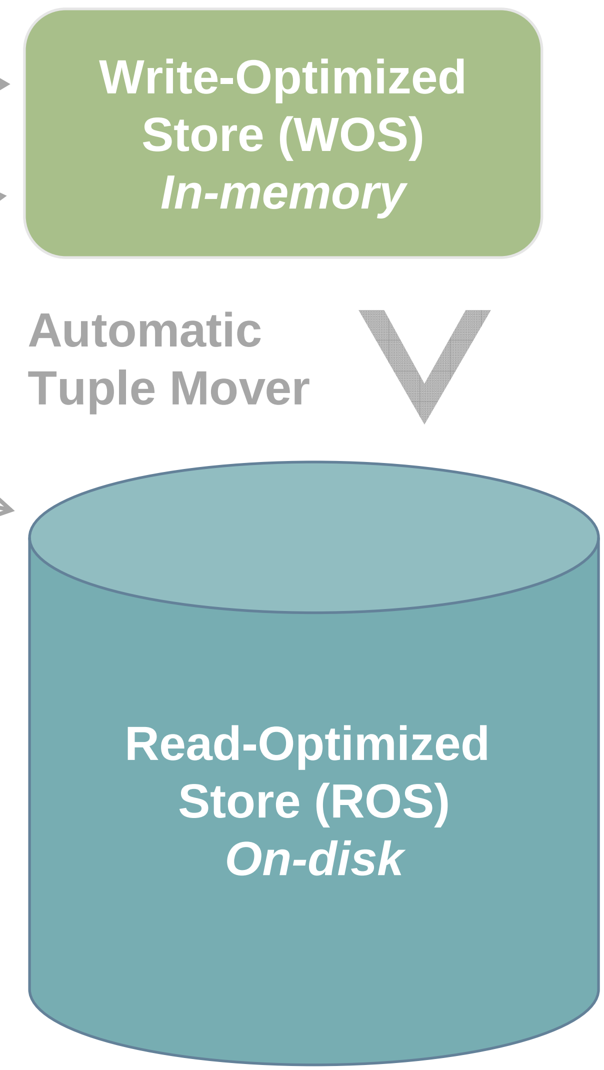
Continuous Load and Query (Vertica)

Hybrid Storage Architecture



Loading Data (Vertica)

- INSERT, UPDATE, DELETE
- Bulk and Trickle Loads
  - COPY
  - COPY DIRECT
- User loads data into logical Tables
- Vertica loads atomically into storage



Applications for column-stores

- Data Warehousing
  - High end (clustering)
  - Mid end/Mass Market
  - Personal Analytics
- Data Mining
  - E.g. Proximity
- Google BigTable
- RDF
  - Semantic web data management
- Information retrieval
  - Terabyte TREC
- Scientific datasets
  - SciDB initiative
  - SLOAN Digital Sky Survey on MonetDB

List of column-store systems

- Cantor (history)
- Sybase IQ
- SenSage (former Addamark Technologies)
- Kdb
- 1010data
- MonetDB
- C-Store/Vertica
- X100/VectorWise
- KickFire
- SAP Business Accelerator
- Infobright
- ParAccel
- Exasol

Outline

- Part 1: Basic concepts — *Stavros*
  - Introduction to key features
  - From DSM to column-stores and performance tradeoffs
  - Column-store architecture overview
  - Will rows and columns ever converge?
- Part 2: Column-oriented execution — *Daniel*
- Part 3: MonetDB/X100 and CPU efficiency — *Peter*

Simulate a Column-Store inside a Row-Store

Date	Store	Product	Customer	Price
01/01	BOS	Table	Mesa	$20
01/01	NYC	Chair	Lutz	$13
01/01	BOS	Bed	Mudd	$79

Option A: Vertical Partitioning

Date

TID	Value
1	01/01
2	01/01
3	01/01

Store

TID	Value
1	BOS
2	NYC
3	BOS

Product

TID	Value
1	Table
2	Chair
3	Bed

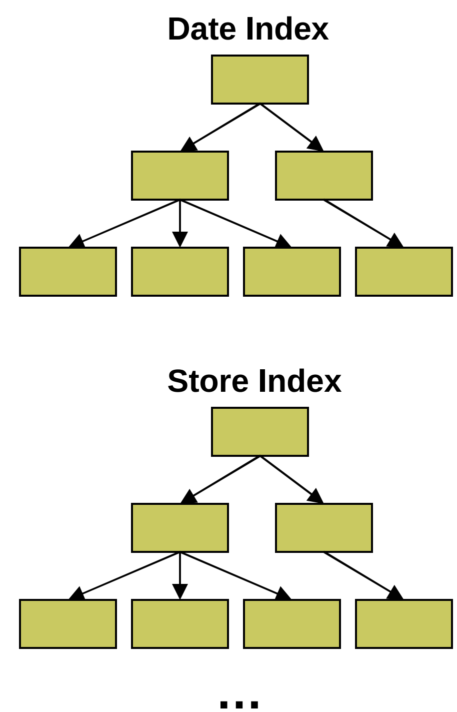
Customer

TID	Value
1	Mesa
2	Lutz
3	Mudd

Price

TID	Value
1	$20
2	$13
3	$79

Option B: Index Every Column



Simulate a Column-Store inside a Row-Store

Date	Store	Product	Customer	Price
01/01	BOS	Table	Mesa	$20
01/01	NYC	Chair	Lutz	$13
01/01	BOS	Bed	Mudd	$79

Option A: Vertical Partitioning

Date

Value	StartPos	Length
01/01	1	3

Store

TID	Value
1	BOS
2	NYC
3	BOS

Product

TID	Value
1	Table
2	Chair
3	Bed

Customer

TID	Value
1	Mesa
2	Lutz
3	Mudd

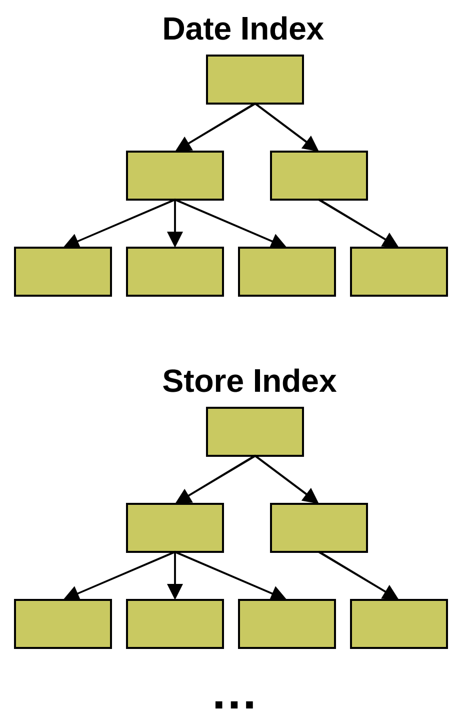
Price

TID	Value
1	$20
2	$13
3	$79

Can explicitly run-length encode date

"Teaching an Old Elephant New Tricks."
Bruno, CIDR 2009.

Option B: Index Every Column

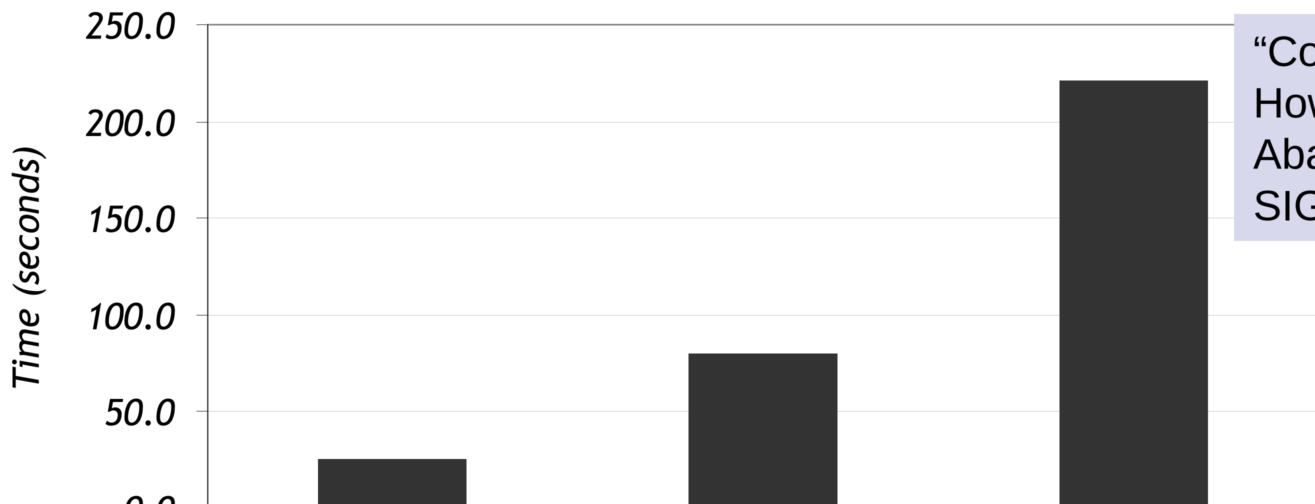


Experiments

- Star Schema Benchmark (SSBM)

Adjoined Dimension Column Index (ADC Index) to Improve Star Schema Query Performance”. O’Neil et. al. ICDE 2008.

- Implemented by professional DBA
- Original row-store plus 2 column-store simulations on same row-store product



	Normal Row-Store	Vertically Partitioned Row-Store	Row-Store With All Indexes
Average	25.7	79.9	221.2

“Column-Stores vs Row-Stores: How Different are They Really?” Abadi, Hachem, and Madden. SIGMOD 2008.

What's Going On? Vertical Partitions

- Vertical partitions in row-stores:
  - Work well when workload is known
  - ..and queries access disjoint sets of columns
  - See automated physical design

Tuple Header	TID	Column Data
	1	
	2	
	3	

- Do not work well as full-columns
  - TupleID overhead significant
  - Excessive joins

"Column-Stores vs. Row-Stores: How Different Are They Really?" Abadi, Madden, and Hachem. SIGMOD 2008.

Queries touch 3-4 foreign keys in fact table, 1-2 numeric columns

Complete fact table takes up ~4 GB (compressed)

Vertically partitioned tables take up 0.7-1.1 GB (compressed)

What's Going On? All Indexes Case

- Tuple construction
  - Common type of query:

```
SELECT store_name, SUM(revenue)
FROM Facts, Stores
WHERE fact.store_id = stores.store_id
    AND stores.country = "Canada"
GROUP BY store_name
```

  - Result of lower part of query plan is a set of TIDs that passed all predicates
  - Need to extract SELECT attributes at these TIDs
    - BUT: index maps value to TID
    - You really want to map TID to value (i.e., a vertical partition)

  Tuple construction is SLOW

So….

- All indexes approach is a poor way to simulate a column-store
- Problems with vertical partitioning are NOT fundamental
  - Store tuple header in a separate partition
  - Allow virtual TIDs
  - Combine clustered indexes, vertical partitioning
- So can row-stores simulate column-stores?
  - Might be possible, BUT:
    - Need better support for vertical partitioning at the storage layer
    - Need support for column-specific optimizations at the executer level
    - Full integration: buffer pool, transaction manager, ..
  - When will this happen?
    - Most promising features = soon

See Part 2, Part 3 for most promising features

    - ..unless new technology / new objectives change the game (SSDs, Massively Parallel Platforms, Energy-efficiency)

End of Part 1

- Basic concepts — *Stavros*
  - Introduction to key features
  - From DSM to column-stores and performance tradeoffs
  - Column-store architecture overview
  - Will rows and columns ever converge?

- ~~Part 2: Column-oriented execution — *Daniel*~~

- ~~Part 3: MonetDB/X100 and CPU efficiency — *Peter*~~