Centrum Wiskunde & Informatica

The Story Of VectorWise

Peter Boncz

Senior Research Scientist @ CWI

Senior Lecturer @ Vrije Universiteit Amsterdam

Architect & Co-founder MonetDB

Architect & Co-founder VectorWise

CWI Centrum Wiskunde & Informatica

vectorwise

Getting To Be the TPC-H Champ

- Fastest non-MPP analytical database system

www.tpc.org/tpch/results/tpch_perf_results.asp?resulttype=noncluster

TPC Transaction Processing Performance Council

SEARCH

Advanced Search

The TPC defines transaction processing and database benchmarks and delivers trusted results to the industry.

- **Home**
- **Results**
  - TPC-C
  - TPC-E
  - TPC-H
- **Benchmarks**
  - TPC-C
  - TPC-E
  - TPC-H
  - Pricing Spec
  - TPC Energy
  - Obsolete
  - - TPC-A
  - - TPC-B
  - - TPC-D
  - - TPC-R
  - - TPC-W
  - - TPC-App
- **Technical Articles**
- **Related Links**
- **Press**
- **About the TPC**
  - What is the TPC
  - Mailing List
  - Applications
  - Documentation
- **Who We Are**
  - Members
  - Affiliates
- **Member Login**
- **Contact Us**

100 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating
1	INGRES	HP ProLiant DL380 G7	251,561	.38 USD	NR	03/31/1	VectorWise 1.5	RedHat Enterprise Lin
2	hp invent	HP ProLiant DL380 G7	73,974	.58 USD	5.93	07/02/10	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Se Enterprise Edition
3	hp invent	HP ProLiant DL385 G7	71,438	.51 USD	6.48	07/14/10	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Se Enterprise Edition

300 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operat
1	DELL	Dell PowerEdge R910 using VectorWise 1.6	400,931	.35 USD	2.38	06/30/1	VectorWise 1.6	RedHat Enterprise
2	hp invent	HP ProLiant DL580 G7	121,345	.65 USD	10.33	09/14/10	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Enterprise Edition
3	hp invent	HP ProLiant DL585 G7	107,561	1.08 USD	9.58	06/21/10	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Enterprise Edition

1,000 GB Results

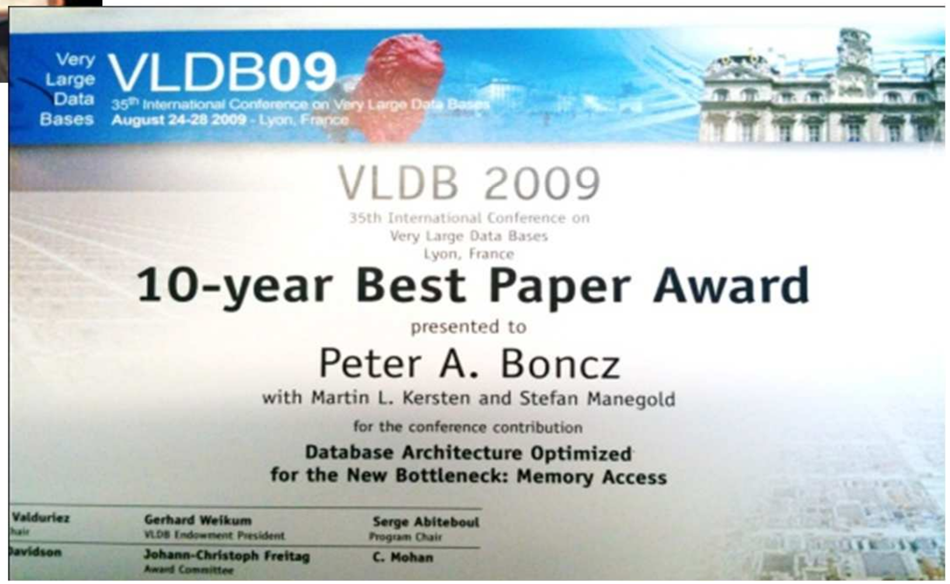
Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Oper
1	DELL	Dell PowerEdge R910 using VectorWise 1.6	436,788	.88 USD	NR	06/30/11	VectorWise 1.6	RedHat Enterp

CWI Centrum Wiskunde & Informatica

vectorwise

Winning an Award

- For research contribution on column stores and architecture-conscious database architecture



Driving Some Fancy Cars

- VectorWise acquisition celebrations

Stories to tell

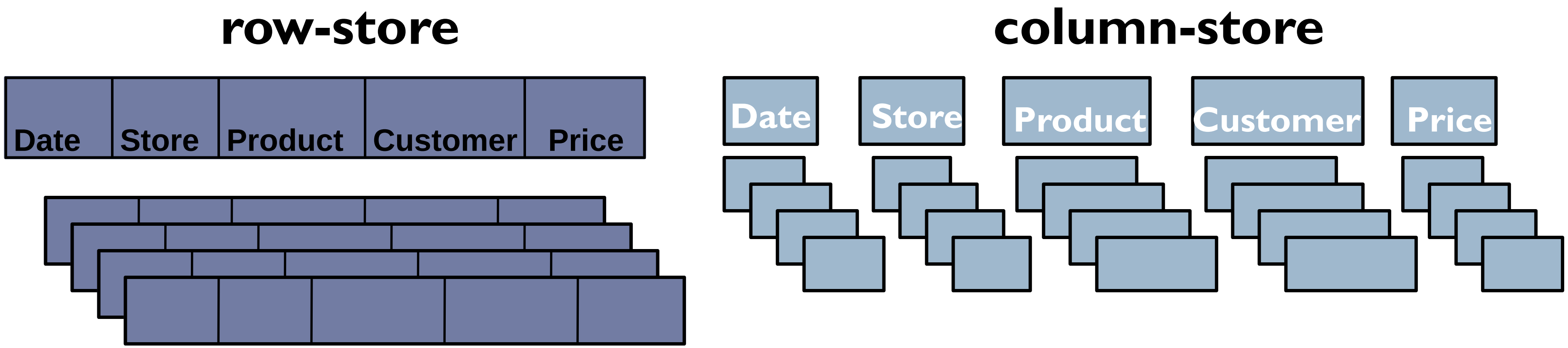
- The Technical story
  - Column Store re-cap
  - History of VectorWise: MonetDB, X100, Ingres (!)
  - short VectorWise technical Highlights
  - TPC-H war stories (if time permits)
- The Spin-off Story
  - how the company got founded, matured, sold
  - perspectives on doing scientific spin-offs

what is a

Column-Store

anyway?

What is a column-store?



+ easy to add/modify a record

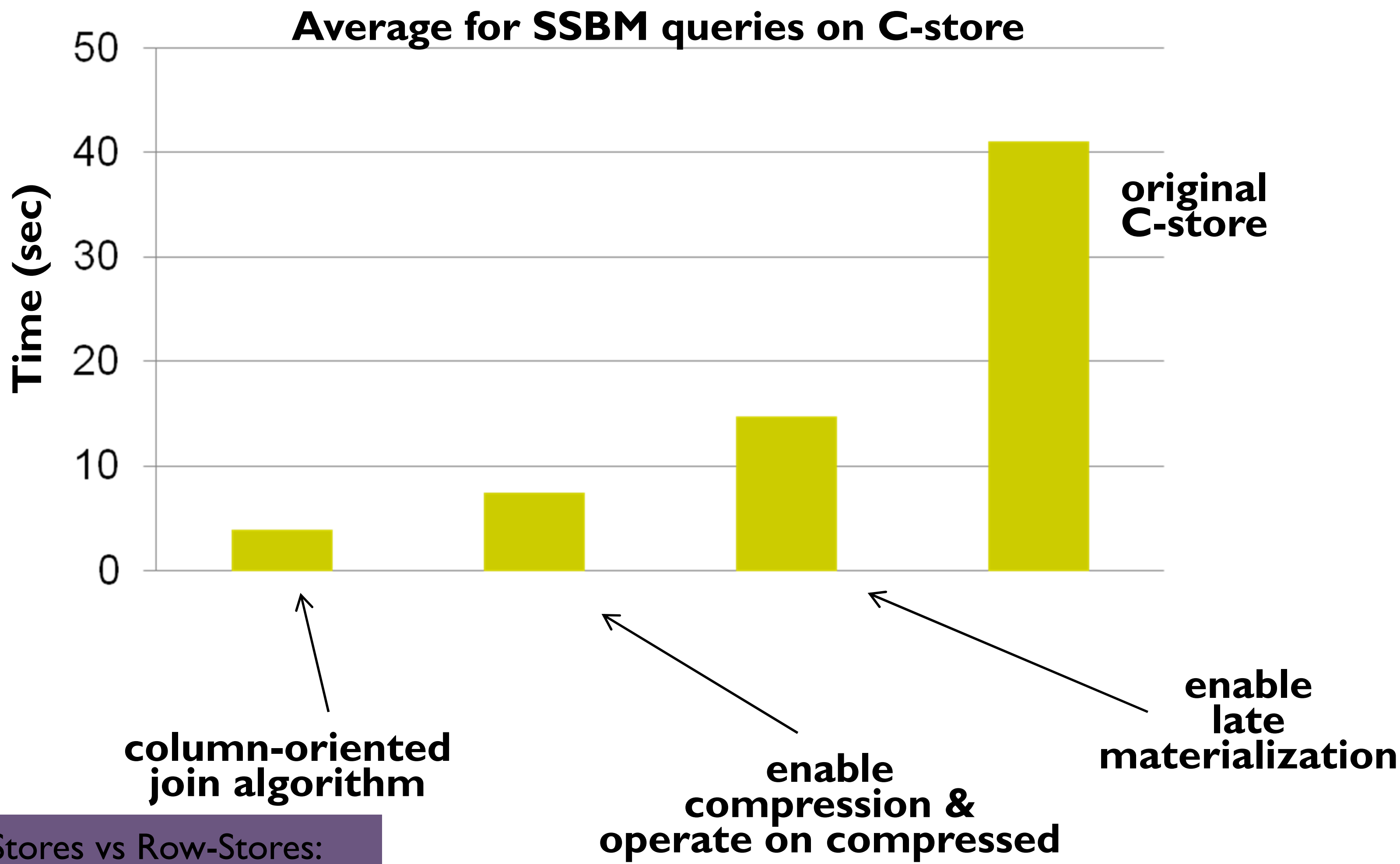
- might read in unnecessary data

+ only need to read in relevant data

- tuple writes require multiple accesses

➔ *suitable for read-mostly, read-intensive, large data repositories*
➔ *OLAP, not OLTP*

Just Different Storage Model?



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

Some Architectural Differences

storage system

- read-optimized: dense-packed, compressed
- batch & differential updates
- multiple sort orders instead of secondary indexes
- deep prefetching in scans

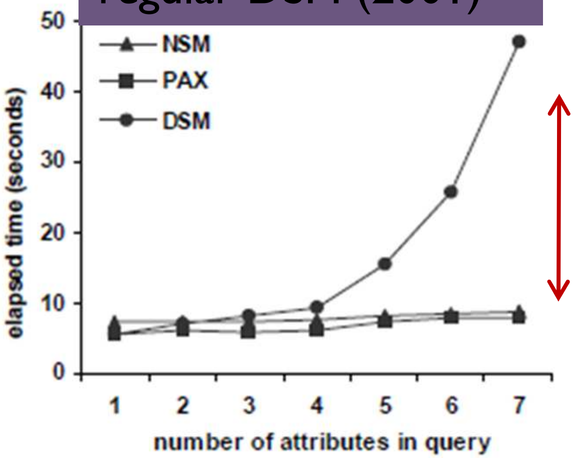
execution engine

- vectorized operators
- compressed execution
- optimized relational operators

Scans: Deep Prefetching

- Problem: multiple columns read in parallel ➔ IO thrashing
- Solution: large prefetch buffer for each column

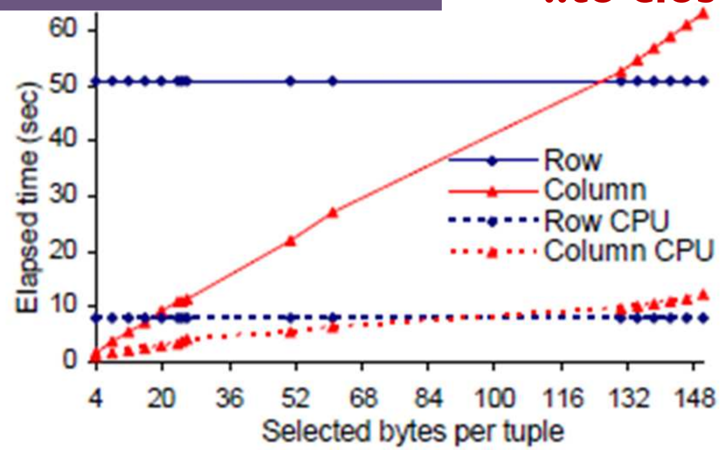
regular DSM (2001)



from 7x slower

column-store (2006)

..to close



Compression: Run-length Encoding

Quarter	Product ID	Price
Q1	1	5
Q1	1	7
Q1	1	2
Q1	1	9
Q1	1	6
Q1	2	8
Q1	2	5
…	…	…
Q2	1	3
Q2	1	8
Q2	1	1
Q2	2	4
…	…	…

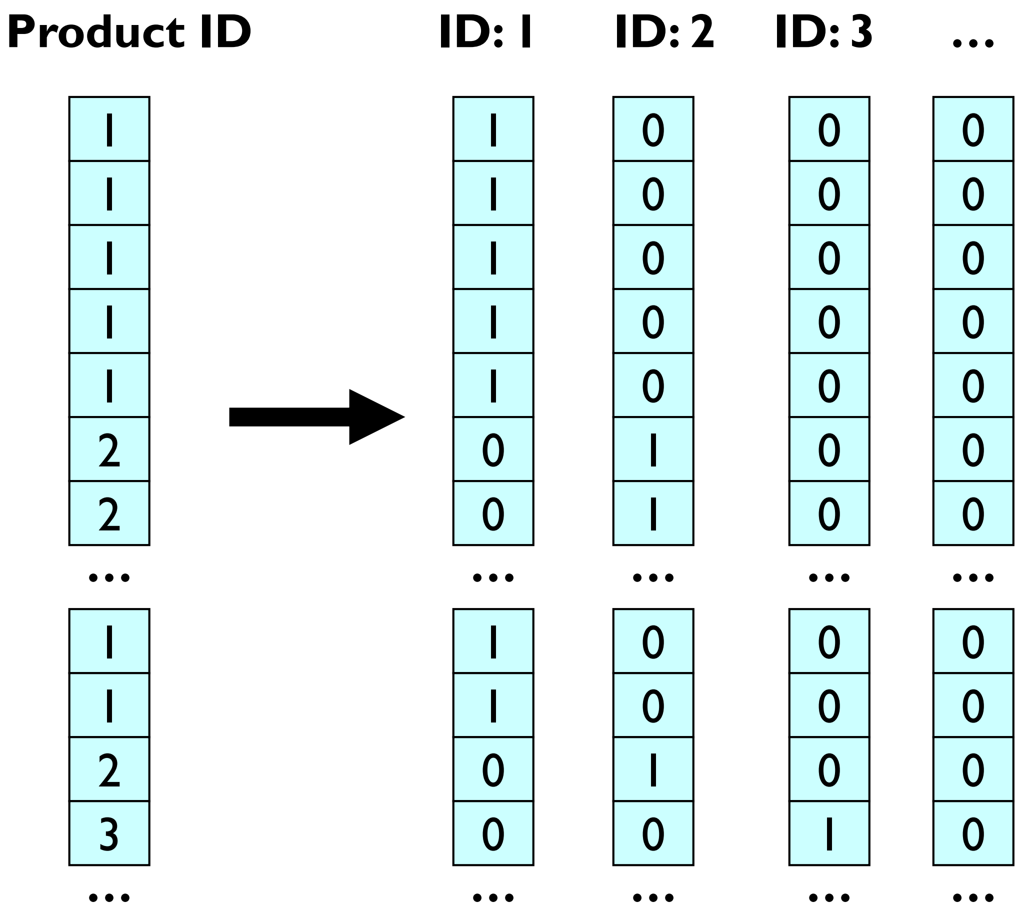
→

Quarter (value, start_pos, run_length)	Product ID (value, start_pos, run_length)	Price
(Q1, 1, 300)	(1, 1, 5)	5
(Q2, 301, 6)	(2, 6, 2)	7
(Q3, 307, 500)	…	2
(Q4, 807, 600)	(1, 301, 3)	9
	(2, 304, 1)	6
	…	8
		5
		…
		3
		8
		1
		4
		…

“Integrating Compression and Execution in Column-Oriented Database Systems” Abadi et. al, SIGMOD ’06

Bit-vector Encoding

- **For each unique value, v, in column c, create bit-vector b**
  - **b[i] = 1 if c[i] = v**
- **Good for columns with few unique values**
- **Each bit-vector can be further compressed if sparse**



Operating Directly on Compressed Data

Quarter

(Q1, 1, 300)
(Q2, 301, 6)
(Q3, 307, 500)
(Q4, 807, 600)

301-306

Product ID

1	2	3
…	…	…
1	0	0
0	0	1
0	1	0
1	0	0
1	0	0
0	0	1
0	1	0
1	0	0
0	0	1
0	1	0
0	0	1
…	…	…

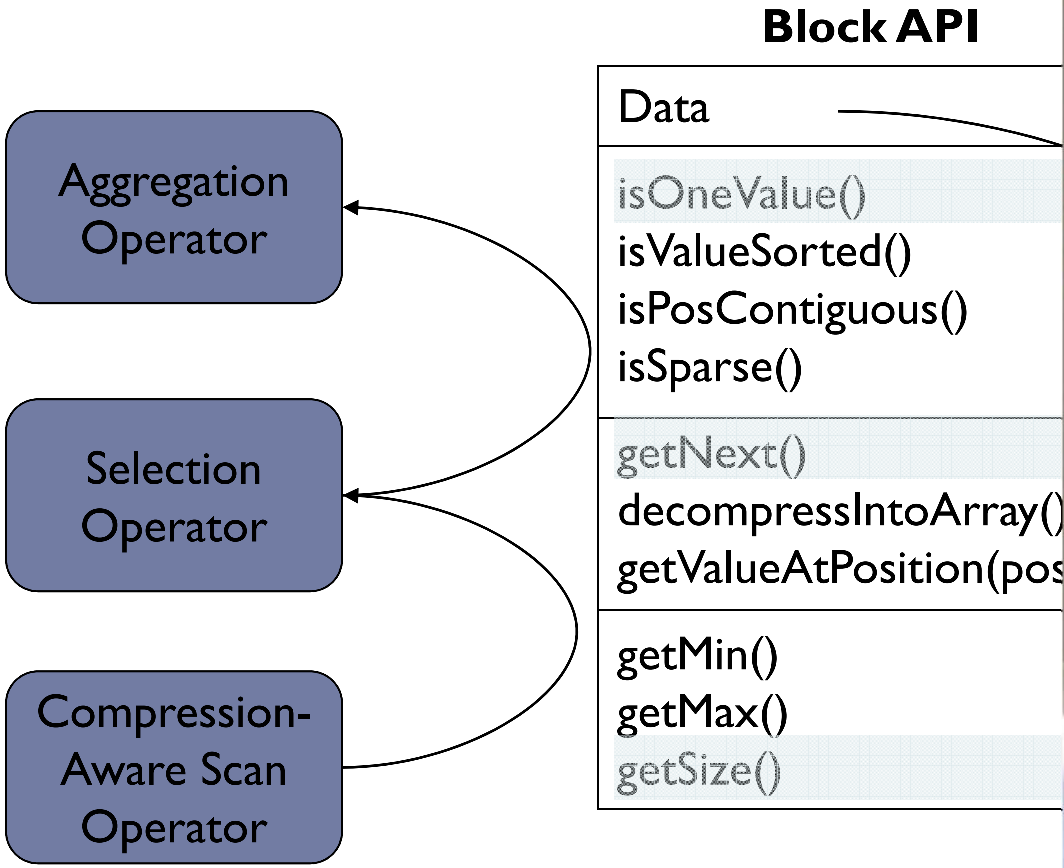
Index Lookup + Offset jump

ProductID, COUNT(*))

(1, 3)
(2, 1)
(3, 2)

SELECT ProductID, Count(*)
FROM table
WHERE (Quarter = Q2)
GROUP BY ProductID

Operating Directly on Compressed Data



vrije Universiteit amsterdam CWI Centrum Wiskunde & Informatica

The History Of

vectorwise

=

monetdb + INGRES

??

Centrum Wiskunde & Informatica

MonetDB

- ~~"save disk I/O when scan-intensive queries only need a few columns"~~
- "avoid an expression interpreter to improve computational efficiency"

Centrum Wiskunde & Informatica

DBMS Computational Efficiency

TPC-H 1GB, query 1

- selects 98% of fact table, computes net prices and aggregates all
- Results:
  - C program: **0.2s**
  - MySQL: 26.2s
  - DBMS "X": 28.1s

"MonetDB/X100: Hyper-Pipelining Query Execution " Boncz, Zukowski, Nes, CIDR'05

RISC Database Algebra

CPU ☺? Give it "nice" code !

- few dependencies (control,data)
- CPU gets out-of-order execution
- compiler can e.g. generate SIMD

One loop for an entire column

- no per-tuple interpretation
- arrays: no record navigation
- better instruction cache locality

```
{
    for(i=0; i<n; i++)
        res[i] = col[i] - val;
}
```

Simple, hard-coded semantics in operators

MATERIALIZED intermediate results

MonetDB

- ~~"save disk I/O when scan-intensive queries only need a few columns"~~
- ~~"avoid an expression interpreter to improve computational efficiency"~~

How?

- RISC query algebra: hard-coded semantics
  - Decompose complex expressions in multiple operations
- Operators only handle simple arrays
  - No code that handles slotted buffered record layout
- Relational algebra becomes Array manipulation language
  - Often SIMD for free

- Plus:
  - use of ***cache-conscious*** algorithms for Sort/Aggr/Join
  - Run-time query optimization: ***recycling*** and ***cracking***, etc
  - Liberal open-source license (monetdb.cwi.nl)

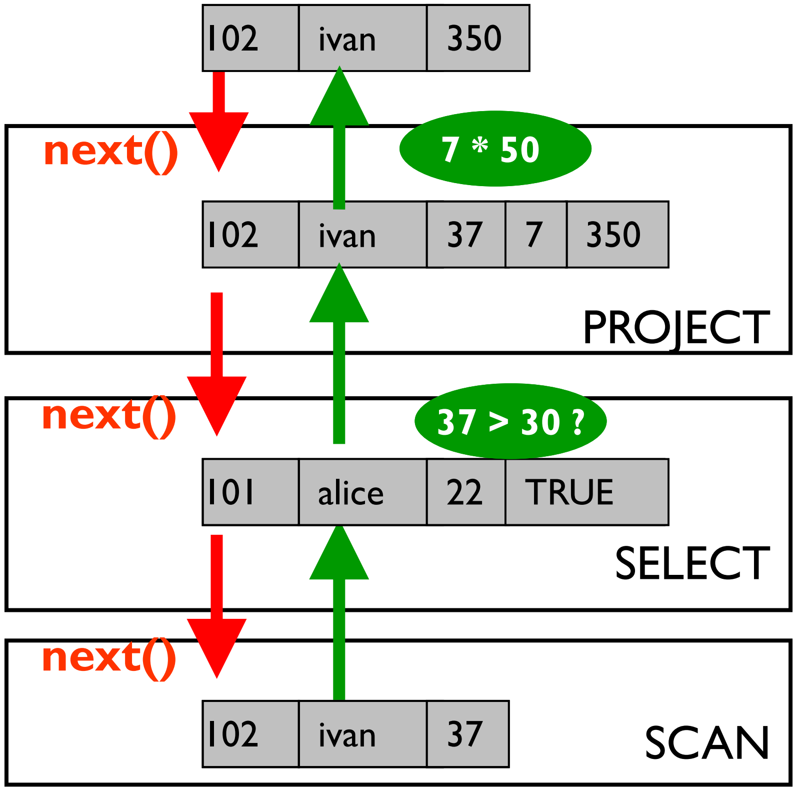
A pact with the devil

- You want efficiency
  - Simple hard-coded operators
- I take scalability
  - Result materialization

C program:	0.2s
MonetDB:	3.7s
MySQL:	26.2s
DBMS "X":	28.1s

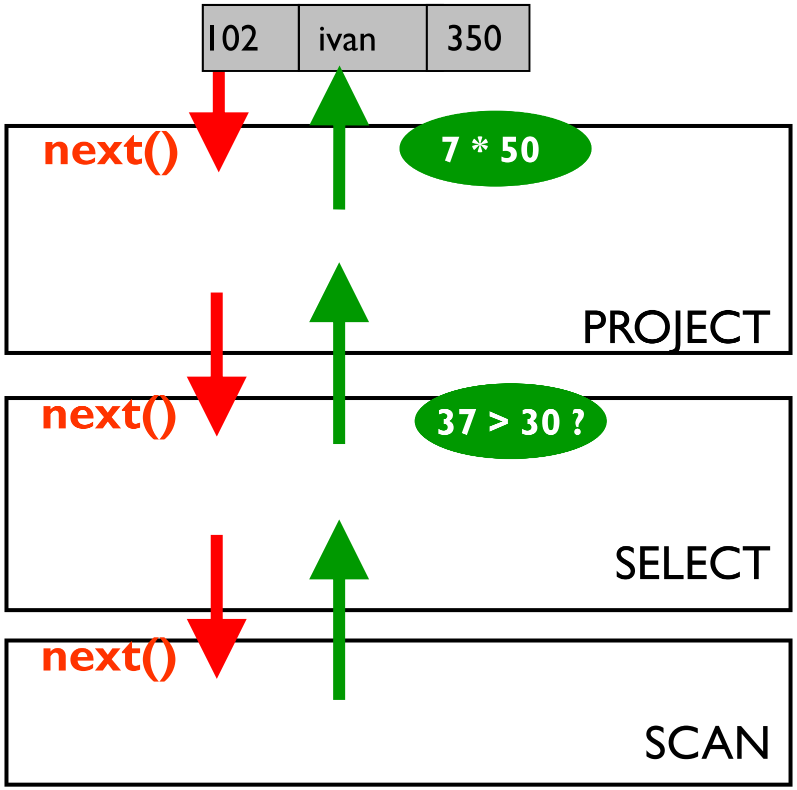
Technical Highlights

A Look at the Query Pipeline



SELECT id, name
(age-30)*50 AS bonus
FROM employee
WHERE age > 30

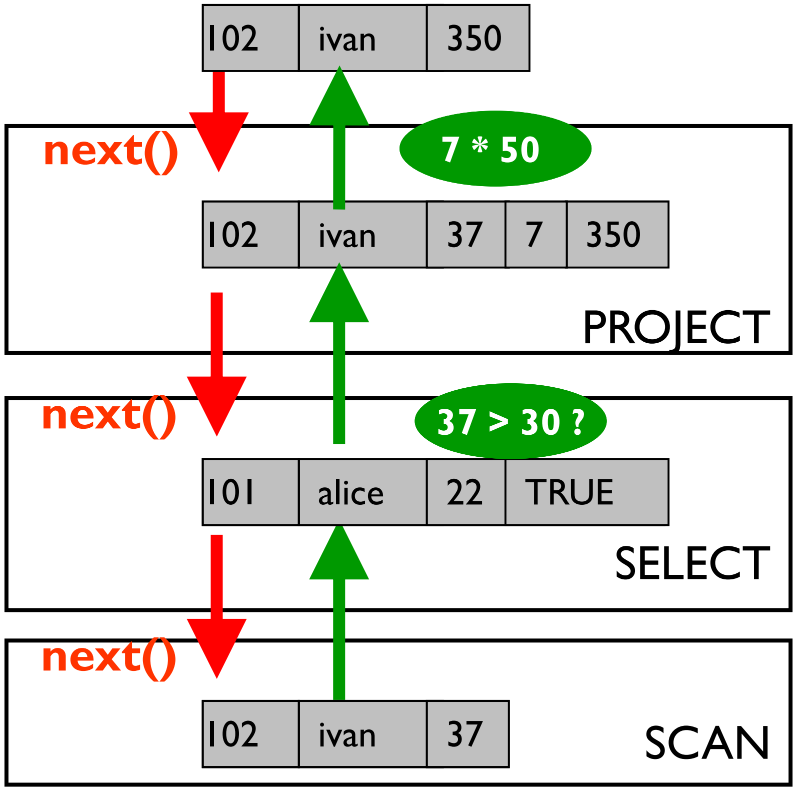
A Look at the Query Pipeline



Operator
s

Iterator interface
-open()
-next(): tuple
-close()

A Look at the Query Pipeline



Primitives

Provide computational functionality

All arithmetic allowed in expressions,
e.g. Multiplication

`mult(int,int) ➔ int`

"Vectorized In Cache Processing"

vector = array of ~100

processed in a tight loop

CPU cache Resident

- automatic SIMD

> 30 ?

FALSE
TRUE
TRUE
FALSE

```
for(i=0; i<n; i++)
   res[i] = (col[i] > x)
```

- 30

7
15

next()

```
for(i=0; i<n; i++)
   res[i] = (col[i] - x)
```

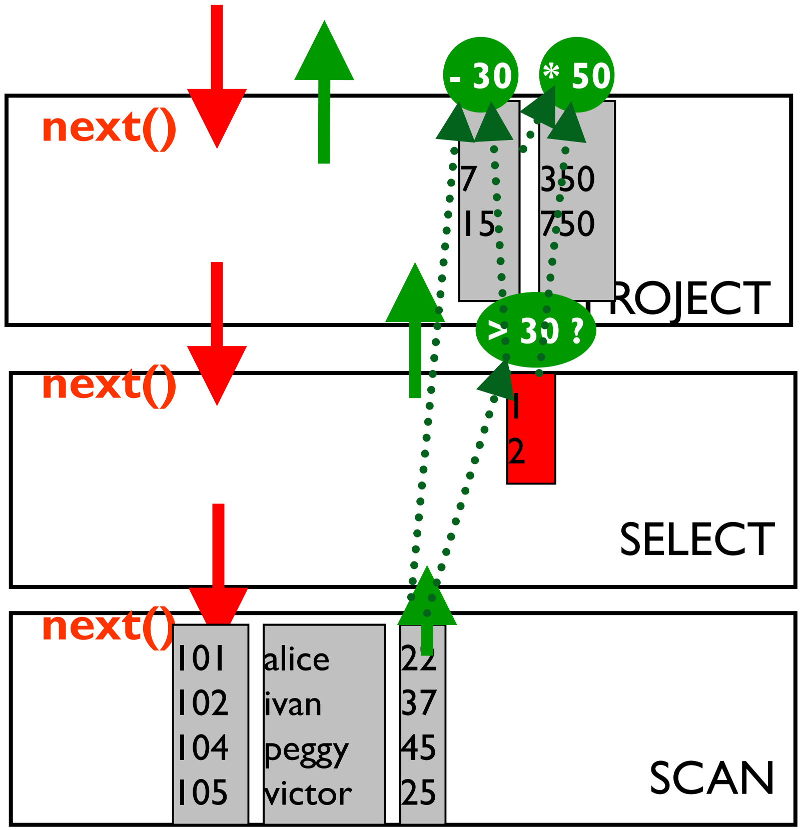
* 50

350
750

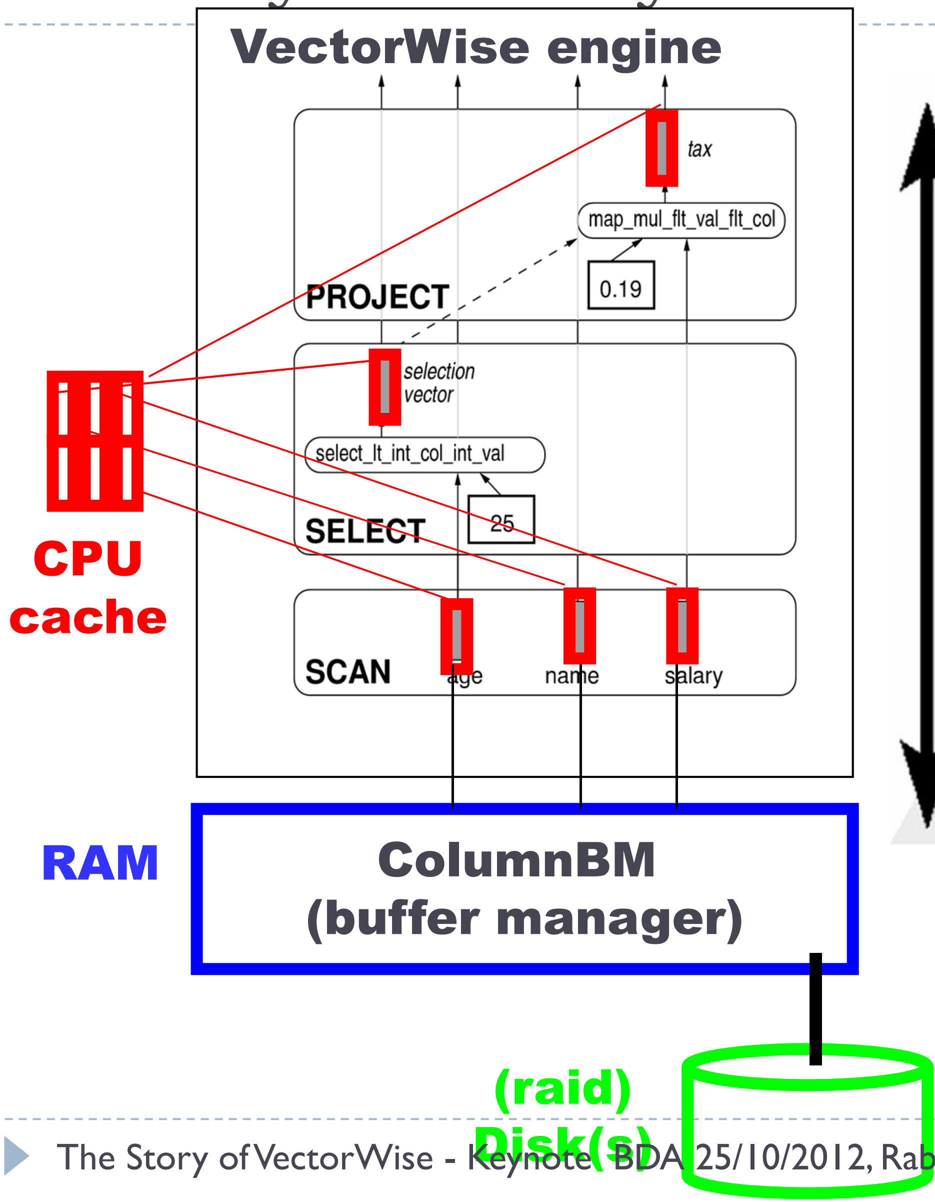
```
for(i=0; i<n; i++)
   res[i] = (col[i] * x)
```

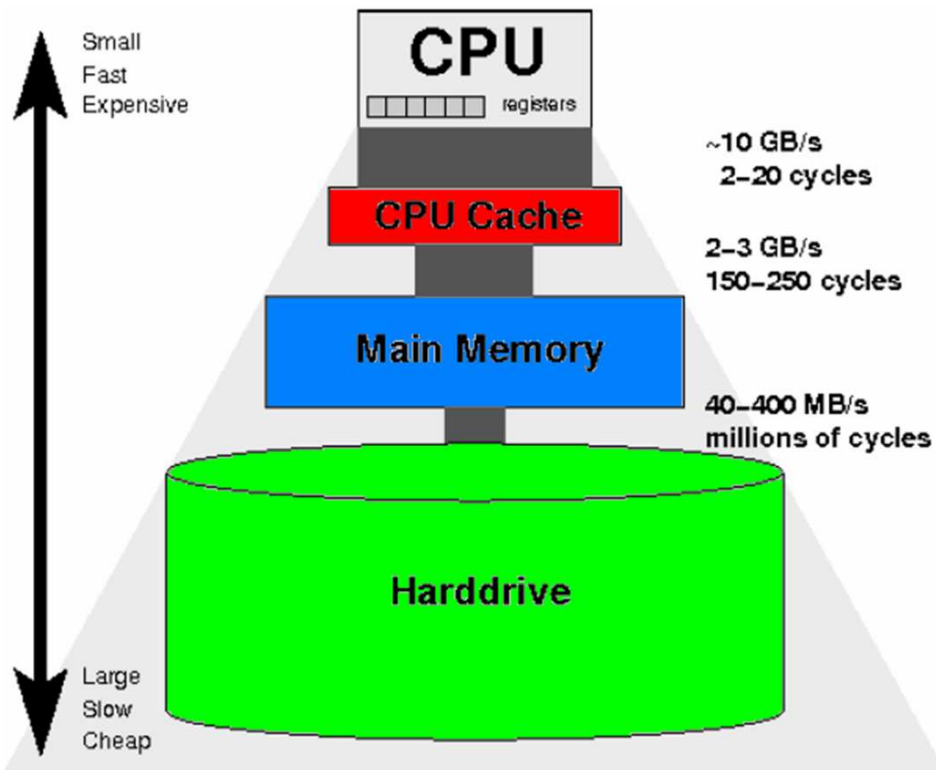
```
map_mul_flt_val_flt_col(
    float *res,
    int*   sel,
    float  val,
    float *col, int n)

{
    if (many selected and narrow)
      for(int i=0; i<n; i++) // SIMD!!
            res[i] = val * col[i];
    else
       for(int i=0; i<n; i++)
            res[i] = val * col[sel[i]];
}
```

“push selections up” to get SIMD

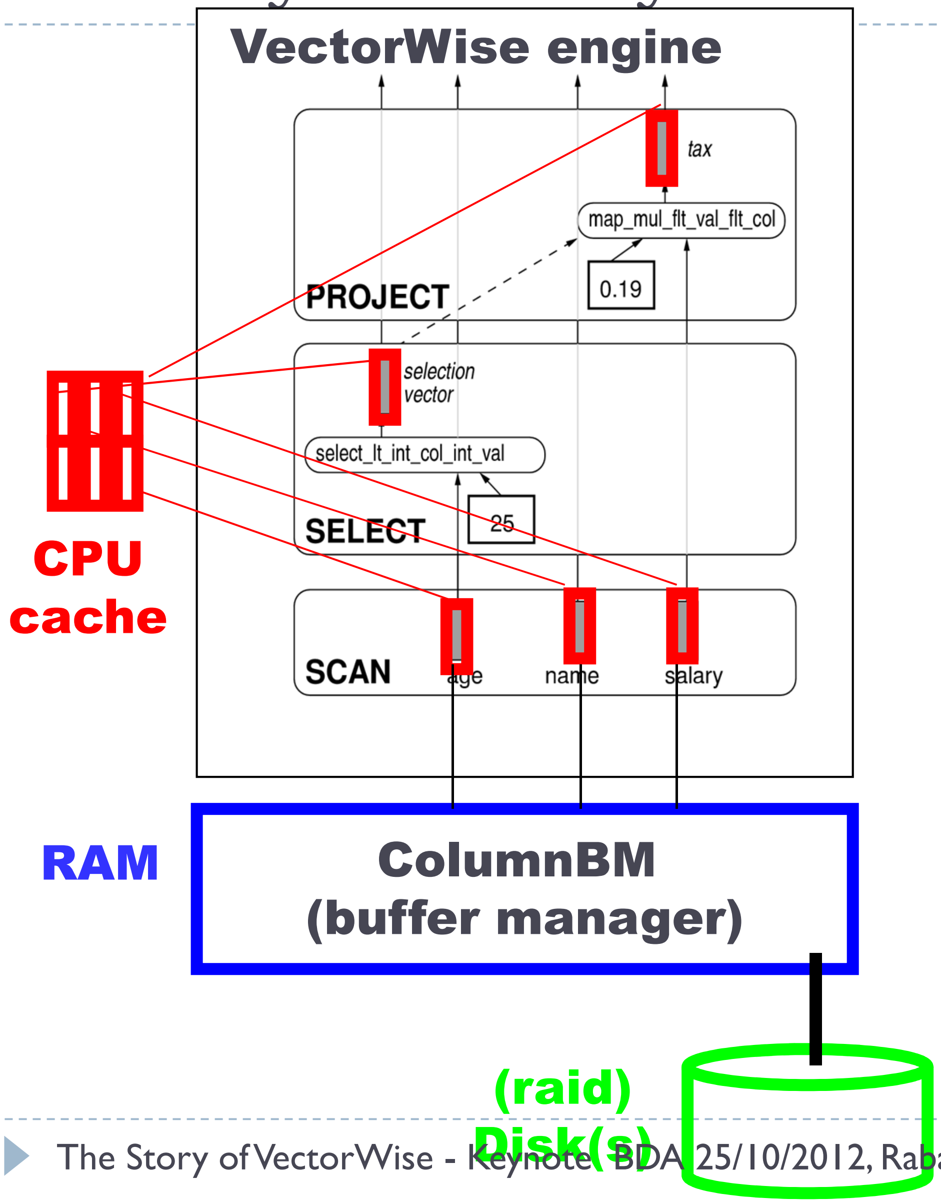


Memory Hierarchy





Memory Hierarchy



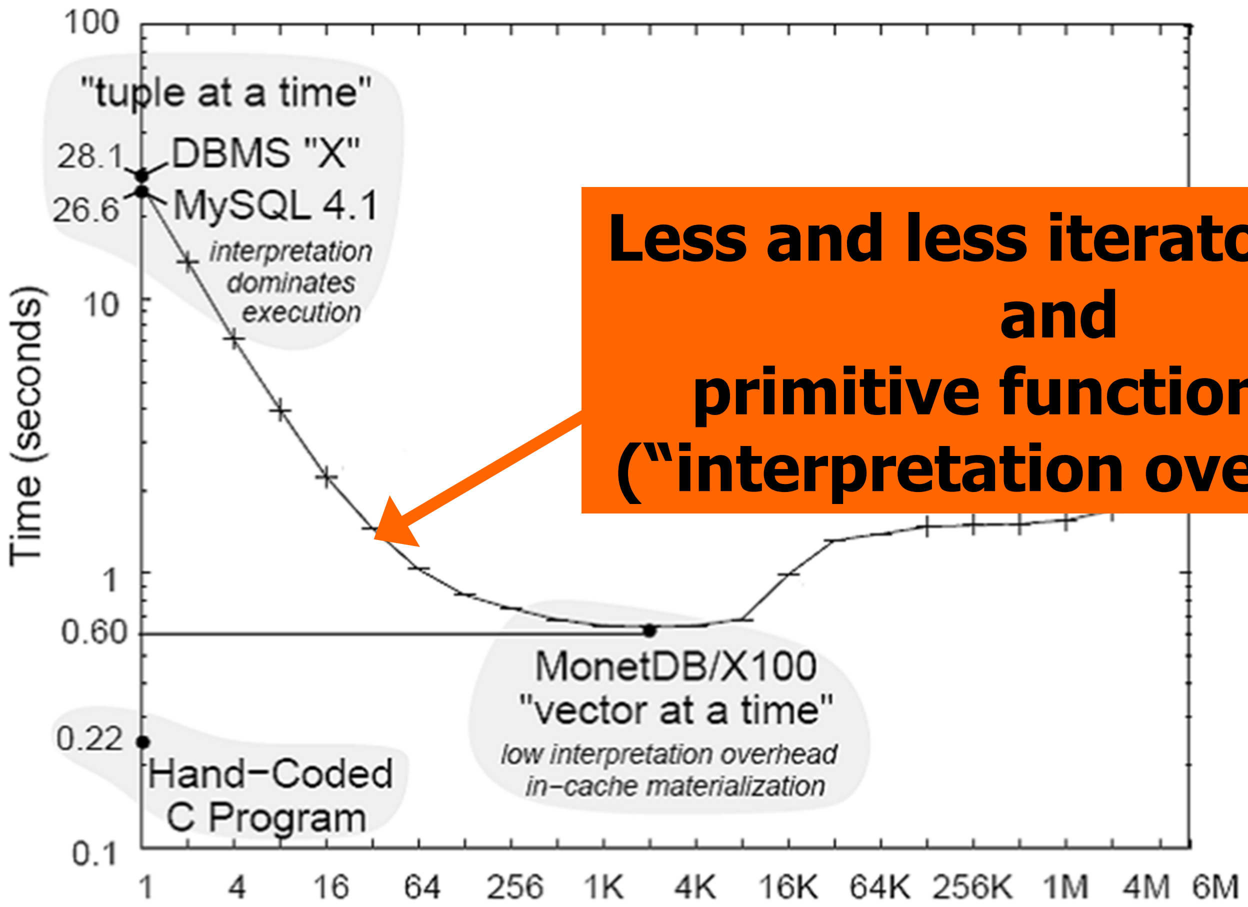
Vectors are only the in-cache representation

RAM & disk representation might actually be different

(vectorwise uses both PAX & DSM)

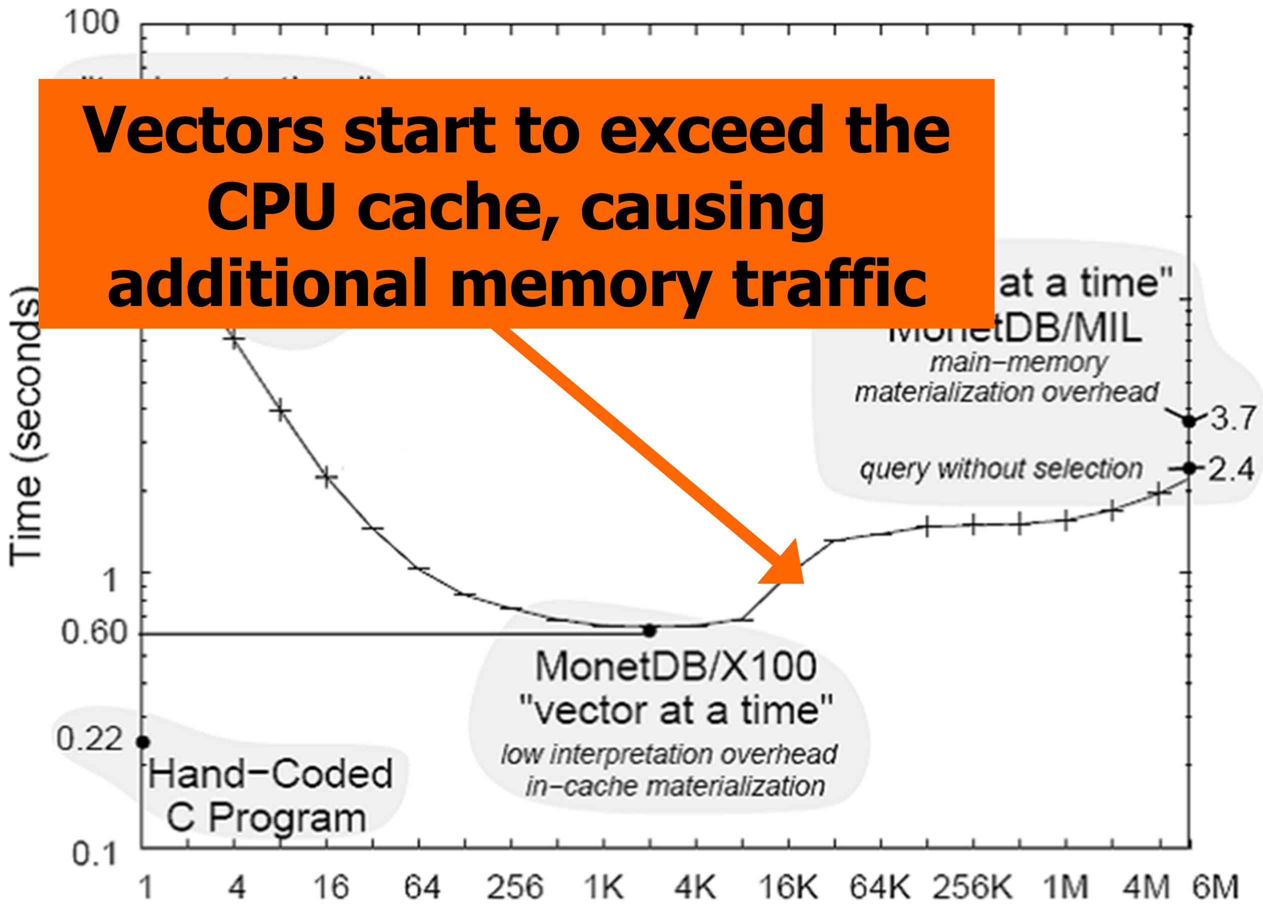
"MonetDB/X100: Hyper-Pipelining Query Execution"
Boncz, Zukowski, Nes, CIDR'05

Varying the vector size



Less and less iterator.next() and primitive function calls ("interpretation overhead")

Varying the Vector size

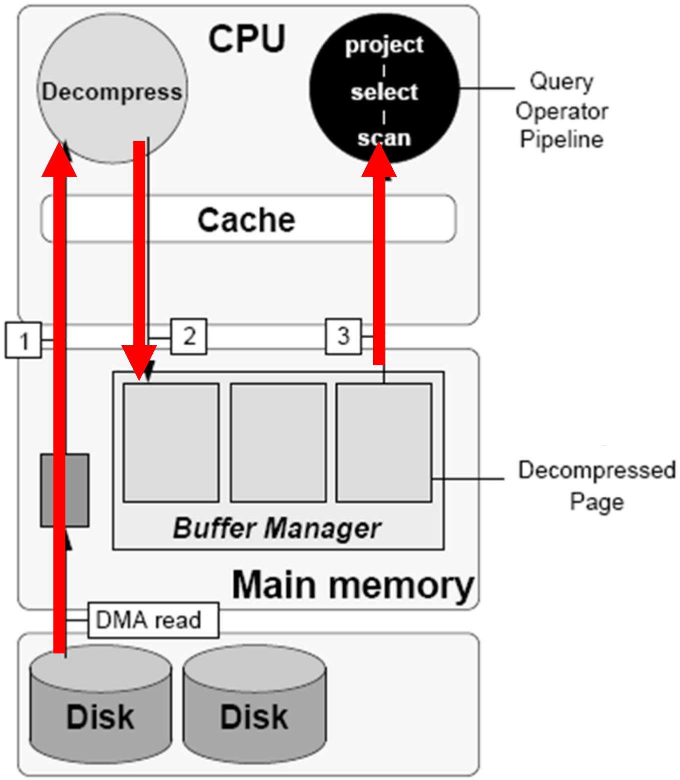


"MonetDB/X100" (VectorWise)

- Both efficiency
  - Vectorized primitives
- and scalability..
  - Pipelined query evaluation

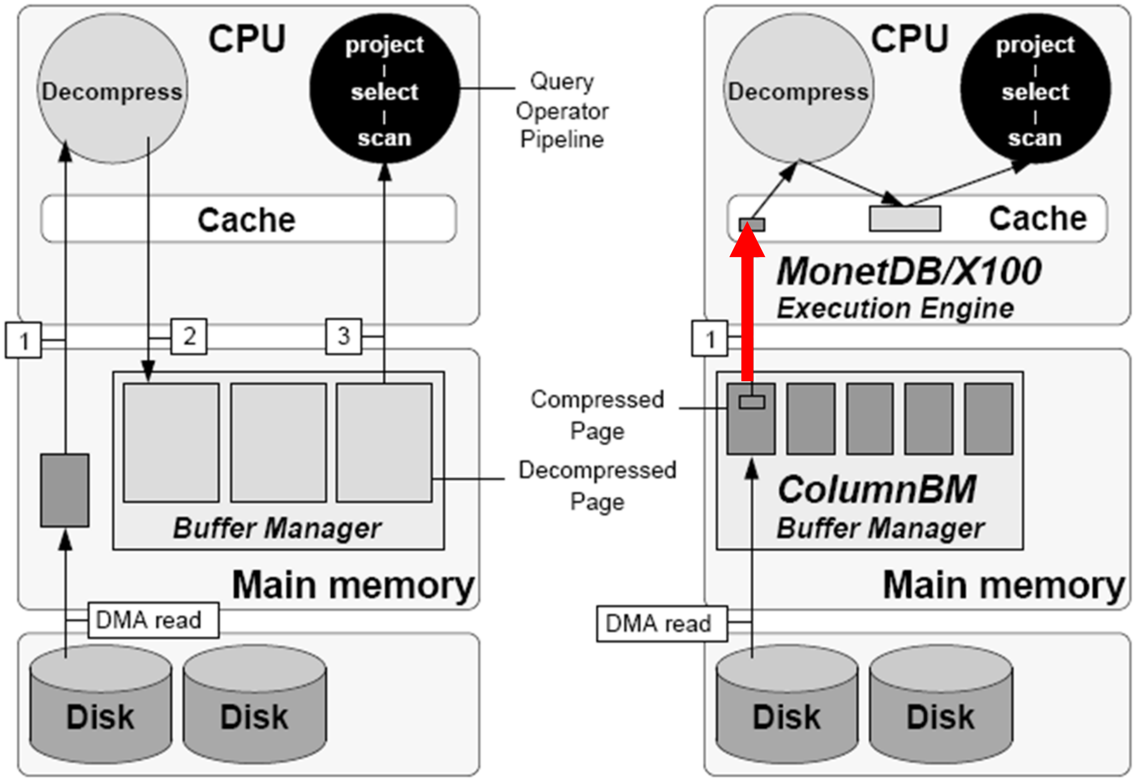
C program:	0.2s
VectorWise:	0.6s
MonetDB:	3.7s
MySQL:	26.2s
DBMS "X":	28.1s

RAM-Cache Decompression



- Naïve way:
  decompress disk block
  before processing
- RAM / CPU-Cache
  boundary crossed
  three times

RAM-Cache Decompression



- Decompress vectors on-demand into the cache
- RAM-Cache boundary only crossed once
- More (compressed) data cached in RAM

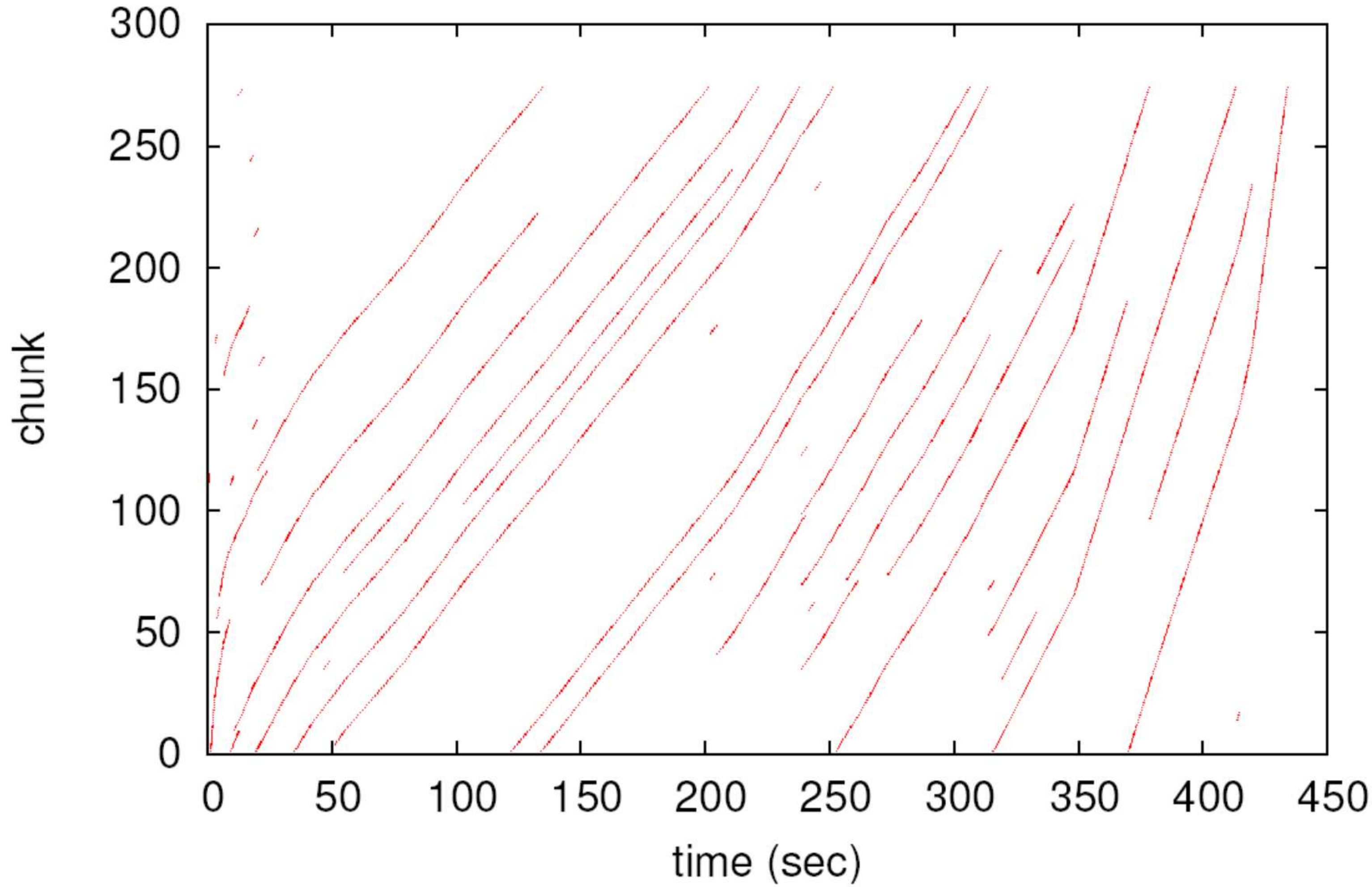
New Compression Schemes

- Goal: improve disk access by accessing less
  - Decompression must be very fast to get benefit

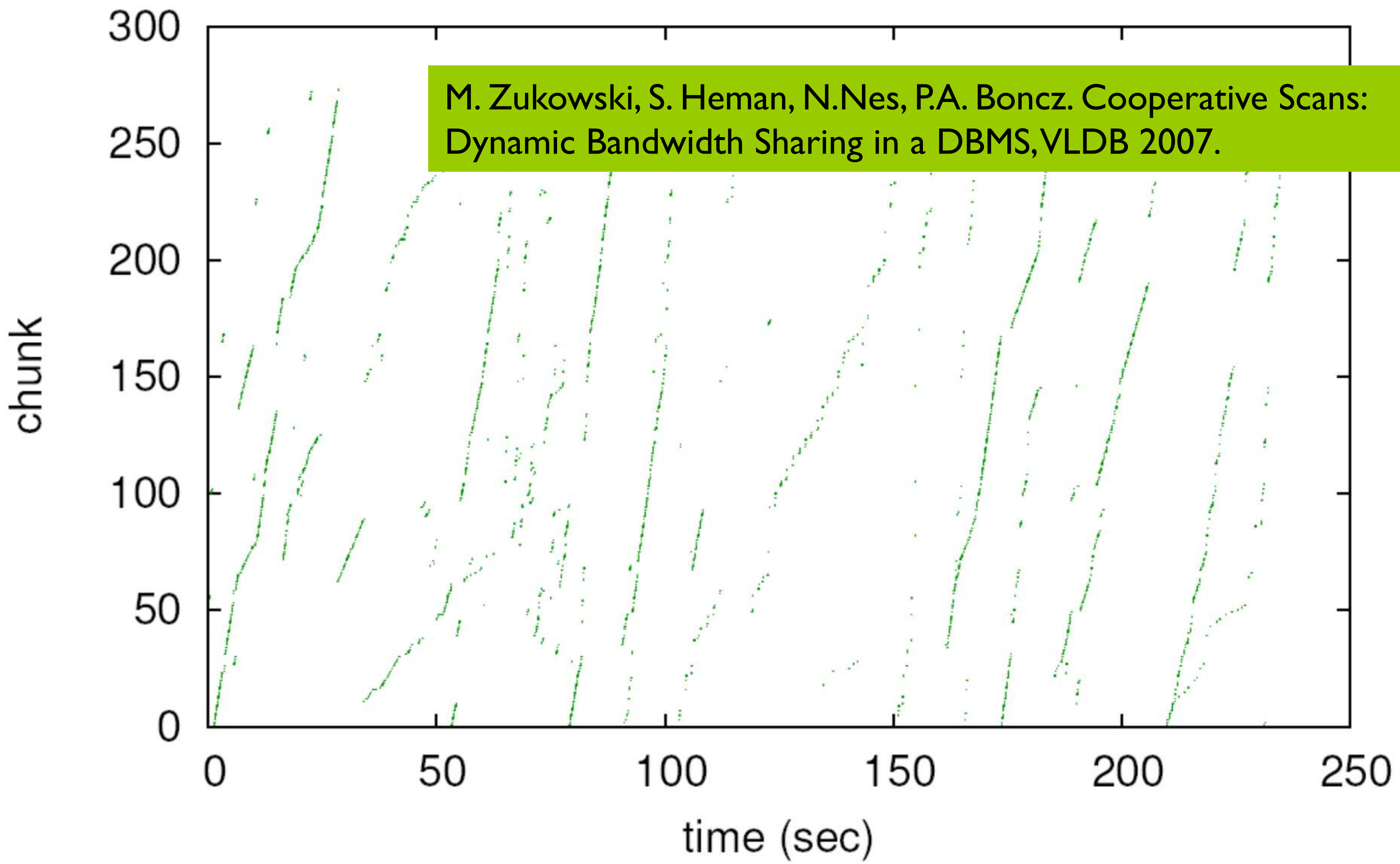
M. Zukowski, S. Heman, N. Nes, P.A. Boncz. Super-Scalar RAM-CPU Cache Compression. ICDE 2006.

- Generic compression spends 5-10 cycles per byte
  - Slower than a good disk system (.5GB/sec)
  - CPU Branch mispredictions slow them down
- New "Patching" family of compression schemes
  - Decompression without IF-THEN-ELSE
  - Achieve 1 byte per cycle (e.g. 3GB/sec)
  - **PFOR, PFOR-DELTA, PDICT**

OLAP: Scan Thrashing



Cooperative scans



M. Zukowski, S. Heman, N.Nes, P.A. Boncz. Cooperative Scans: Dynamic Bandwidth Sharing in a DBMS, VLDB 2007.

Positional Differential Updates

- Remember the position of an update rather than its Sort Key (SK) values
  - Merge once at write ➔ Read-Optimized approach
  - No need to scan SK columns
  - Scan can skip ➔ less CPU overhead

M. Zukowski, S. Heman, N.Nes, P.A. Boncz. Positional Update Handling In Column Stores, SIGMOD 2010.

Notation:

- TABLE_x state of TABLE at time x
- SID(t): StableID
  - Position of tuple *t* in immutable base TABLE_0 ← Stable
- RID_x(t): RowID
  - Position of **visible** tuple *t* at time x ←VOLATILE!
  - SID(t) = RID_0(t)

SID/RID Example

SID	STORE	PROD	NEW	QTY	RID
0	London	chair	N	30	0
1	London	stool	N	10	1
2	London	table	N	20	2
3	Paris	rug	N	1	3
4	Paris	stool	N	5	4

TABLE_0

INSERT INTO inventory VALUES('Berlin', 'table', Y, 10)
INSERT INTO inventory VALUES('Berlin', 'cloth', Y, 5)
INSERT INTO inventory VALUES('Berlin', 'chair', Y, 20)

SID	STORE	PROD	NEW	QTY	RID
0	Berlin	chair	Y	20	0
0	Berlin	cloth	Y	5	1
0	Berlin	table	Y	10	2
0	London	chair	N	30	3
1	London	stool	N	10	4
2	London	table	N	20	5
3	Paris	rug	N	1	6
4	Paris	stool	N	5	7

TABLE_1

SIDs and RIDs

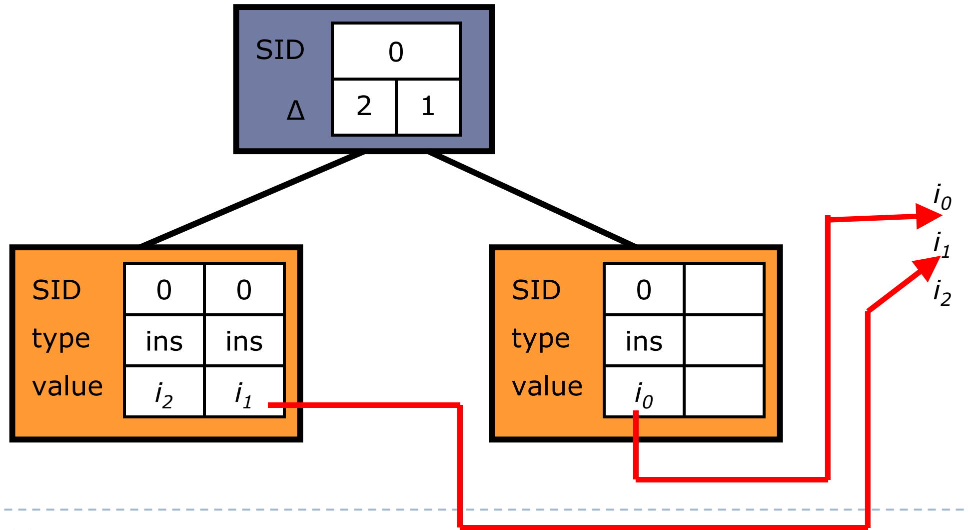
- RID(t) = SID(t) + Δ(t)
- Δ(t) = #inserts *before* t – #deletes *before* t
- SID and RID are monotonically increasing →
  - *organize positional updates on SID in a counting B-Tree that keeps track cumulative deltas (Δ)*
    - **Positional Delta Tree (PDT)**
      - SIDs are stable
      - Only need to maintain cumulative Δ on path root → leaf

PDT Example

SID	STORE	PROD	NEW	QTY	*RID*
0	London	chair	N	30	0
1	London	stool	N	10	1
2	London	table	N	20	2
3	Paris	rug	N	1	3
4	Paris	stool	N	5	4

TABLE_0

INSERT INTO inventory VALUES('Berlin', 'table', Y, 10)
INSERT INTO inventory VALUES('Berlin', 'cloth', Y, 5)
INSERT INTO inventory VALUES('Berlin', 'chair', Y, 20)



Insert Value Table

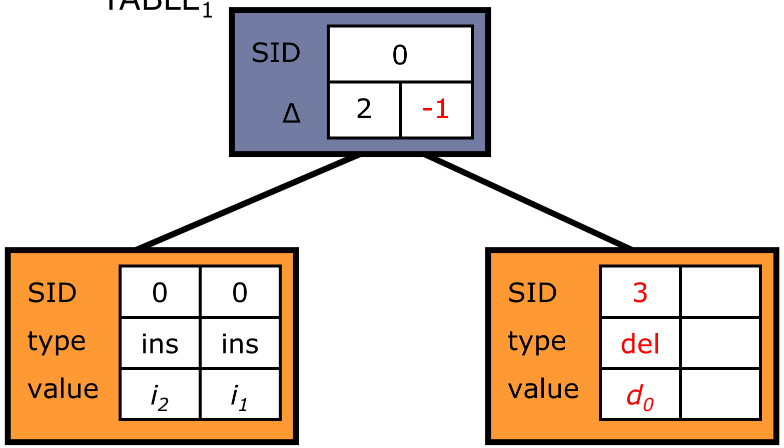
STORE	PROD	NEW	QTY
Berlin	table	Y	20
Berlin	cloth	Y	5
Berlin	chair	Y	10

PDT Example

SID	STORE	PROD	NEW	QTY	RID
0	*Berlin*	*chair*	*Y*	*20*	*0*
0	*Berlin*	*cloth*	*Y*	*5*	*1*
0	*Berlin*	*table*	*Y*	*10*	*2*
0	London	chair	N	30	3
1	London	stool	N	10	4
2	London	table	N	20	5
3	Paris	rug	N	1	6
4	Paris	stool	N	5	7

DELETE FROM inventory WHERE
store = 'Berlin' AND prod = 'table'
DELETE FROM inventory WHERE
store = 'Paris' AND prod = 'rug'

$TABLE_1$



Insert Value Table

	STORE	PROD	NEW	QTY
i_0	Berlin	table	Y	20
i_1	Berlin	cloth	Y	5
i_2	Berlin	chair	Y	10

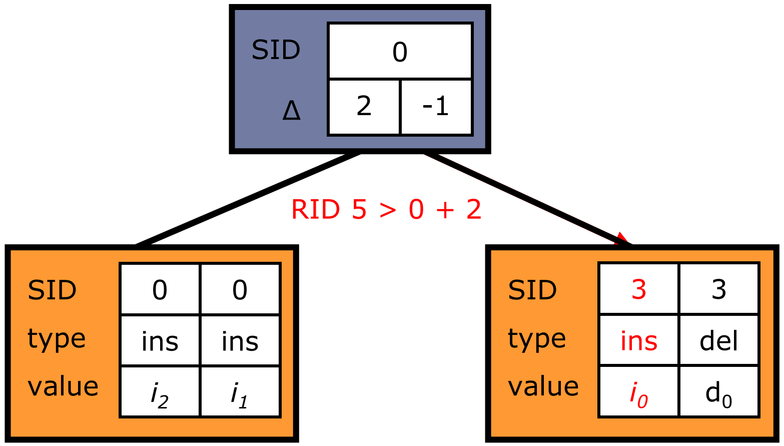
PDT Example

SID	STORE	PROD	NEW	QTY	RID
0	*Berlin*	*chair*	*Y*	*20*	*0*
0	*Berlin*	*cloth*	*Y*	*5*	*1*
0	London	chair	N	30	2
1	London	stool	N	10	3
2	London	table	N	20	4
4	Paris	stool	N	5	5

$TABLE_2$

INSERT INTO inventory VALUES
('Paris', 'rack', Y, 4)

Insert at RID = 5

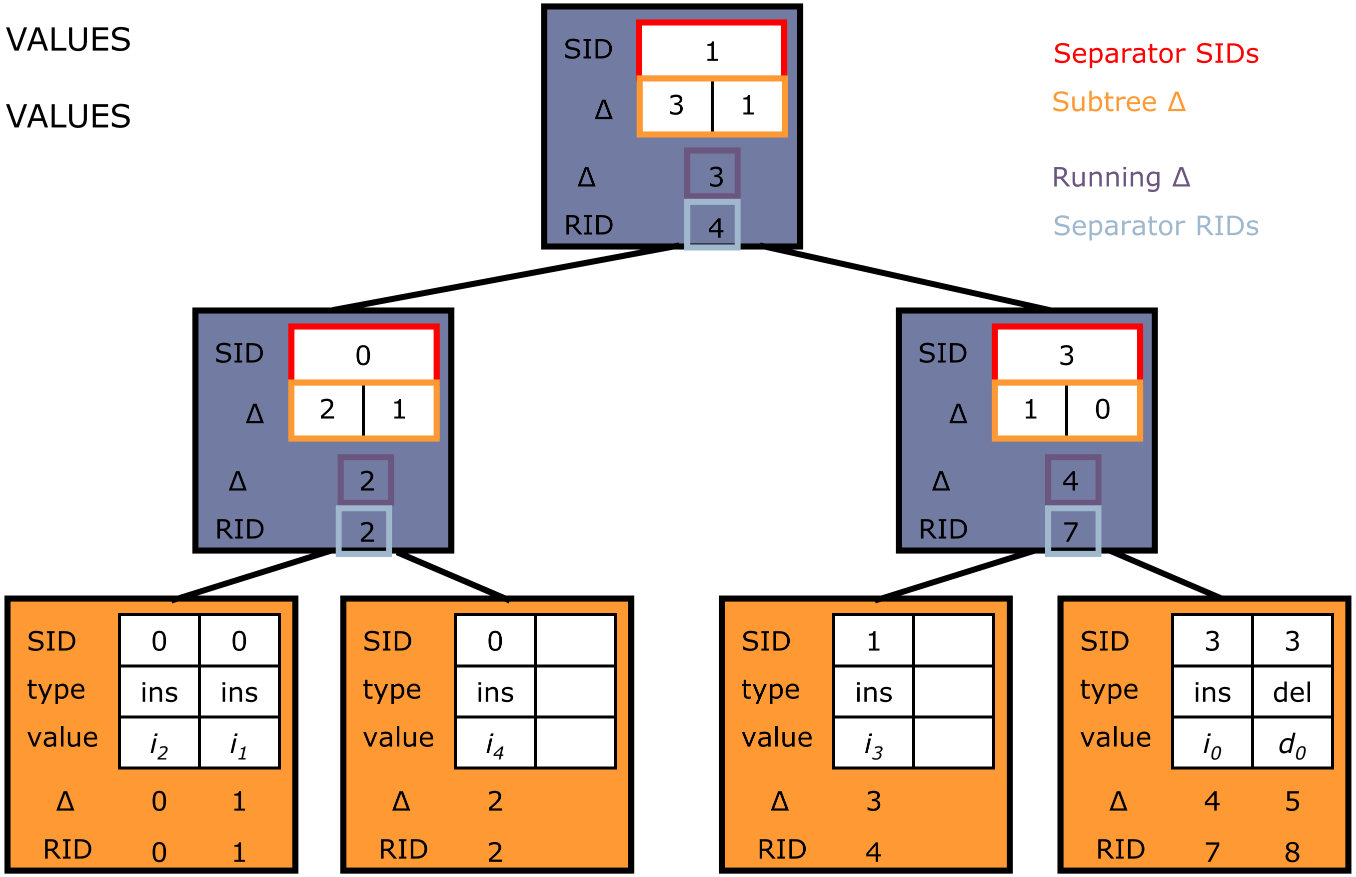


Insert Value Table

	STORE	PROD	NEW	QTY
i_0	Paris	Rack	Y	4
i_1	Berlin	cloth	Y	5
i_2	Berlin	chair	Y	10

PDT Example

INSERT INTO inventory VALUES
('London', 'rack', Y, 4)
INSERT INTO inventory VALUES
('Berlin', 'rack', Y, 4)

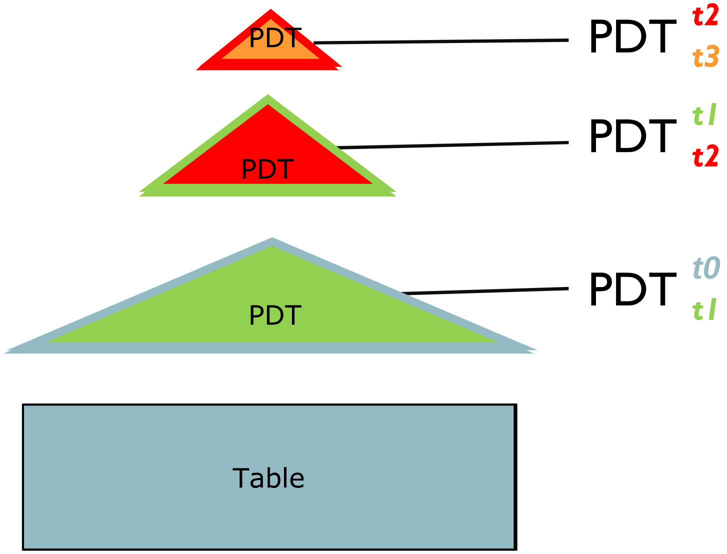


Stacking PDTs

- Arbitrary number of layers: "deltas on deltas on .."
  - *RID domain of child PDT = SID domain of parent PDT*

PDT^{t2}_{t3} vs PDT^{t0}_{t1} are

consecutive ⇔ t2=t1



Stacking PDTs

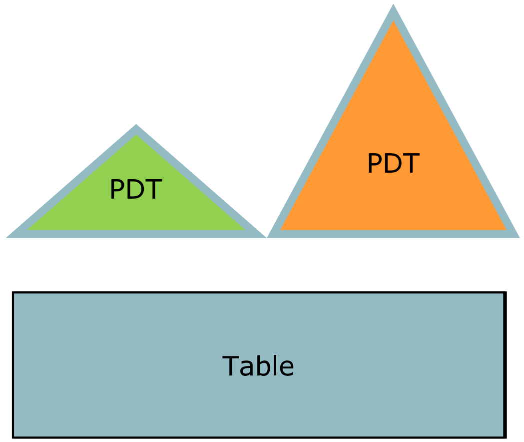
- Arbitrary number of layers: "deltas on deltas on .."
  - *RID domain of child PDT = SID domain of parent PDT*

PDT^{t2}_{t3} vs PDT^{t0}_{t1} are

consecutive ⇔ **t2=t1**

aligned ⇔ **t2=t0**

"same base"



Stacking PDTs

- Arbitrary number of layers: "deltas on deltas on .."
  - *RID domain of child PDT = SID domain of parent PDT*

PDT $^{t2}_{t3}$ vs PDT $^{t0}_{t1}$ are

consecutive ⇔ t2=t1

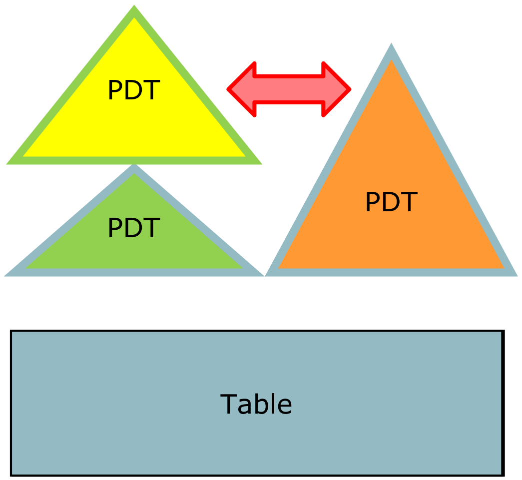
aligned ⇔ t2=t0

"same base"

overlapping ⇔

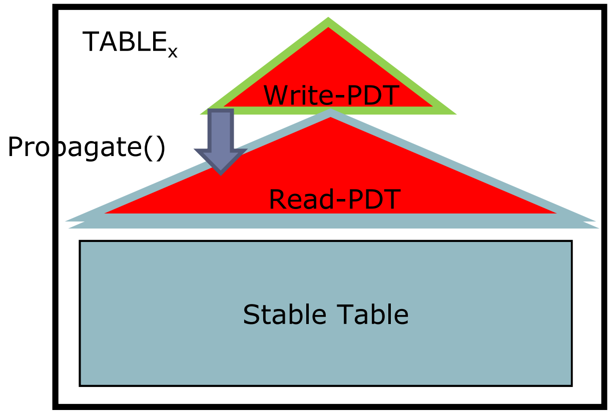
[t2,t3] overlaps [t0,t1]

"uncomparable" / "incompatible"

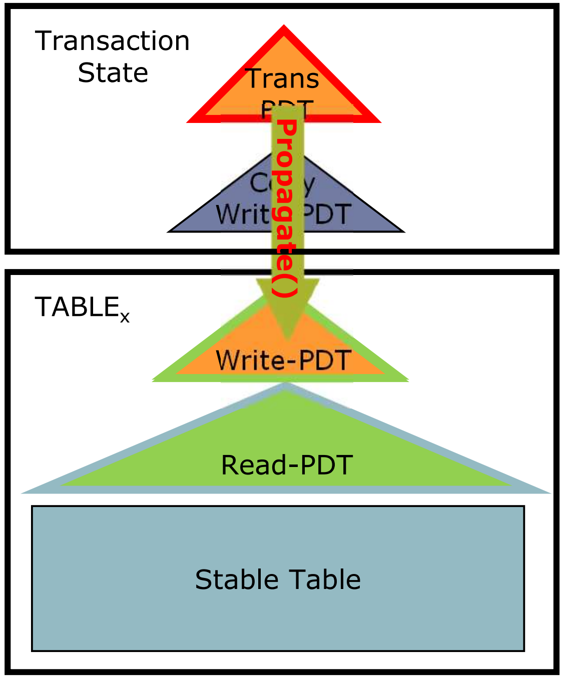


Stacking for Isolation

- 'lock' PDT down for further updates
  - Immutable *read-PDT* ➔ *BIG: main memory resident*
- 'stack' empty PDT on top
  - Updateable *write-PDT* ➔ *SMALL: L2 cache resident*
  - Note: PDTs are *consecutive*
- once in a while changes are propagated
  - Propagate() operation
    - Requires consecutive PDTs



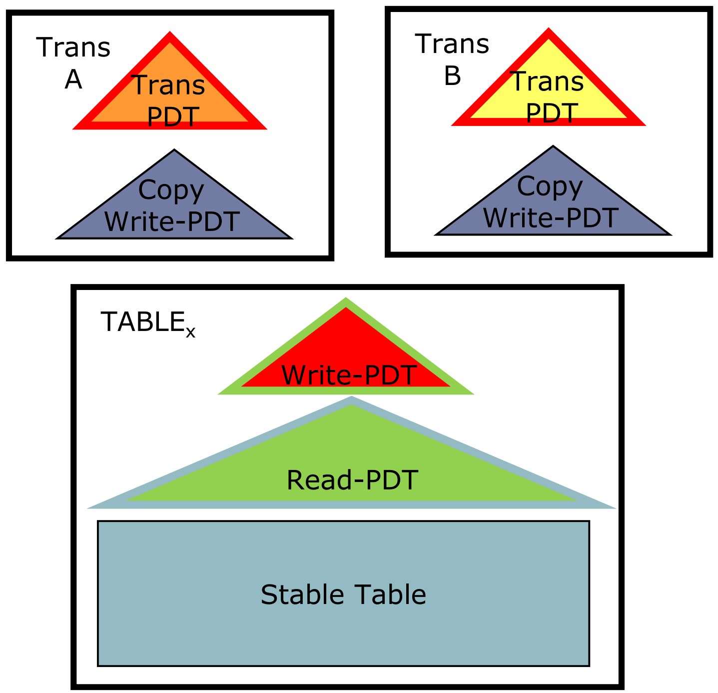
Snapshot Isolation



- Transaction creates snapshot copy of write-PDT
- Updates go into *trans-PDT*
- On commit, *Propagate()* trans-PDT into write-PDT

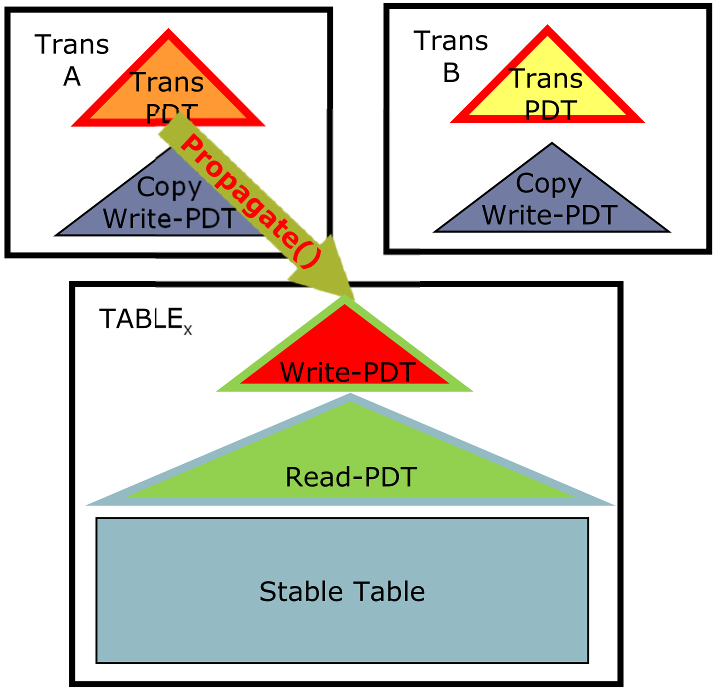
Optimistic Concurrency Control

- Two concurrent transactions

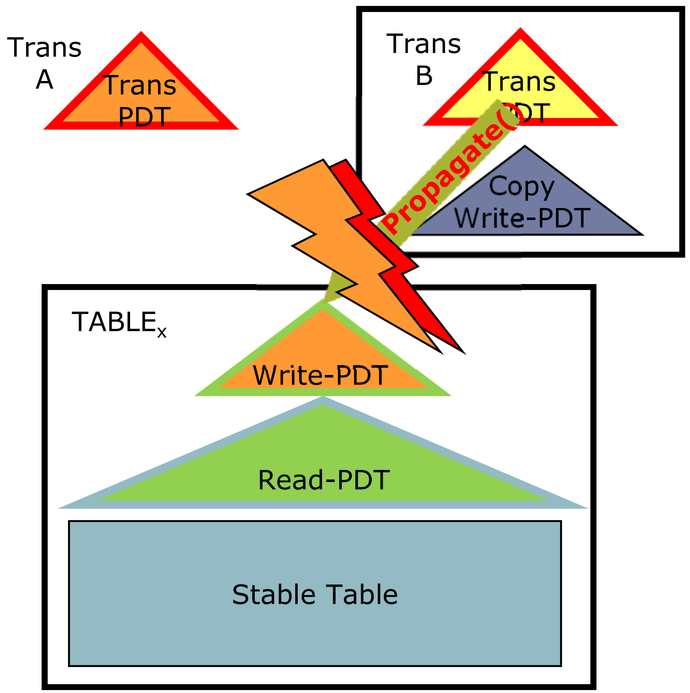


Optimistic Concurrency Control

- Two concurrent transactions
- A commits before B

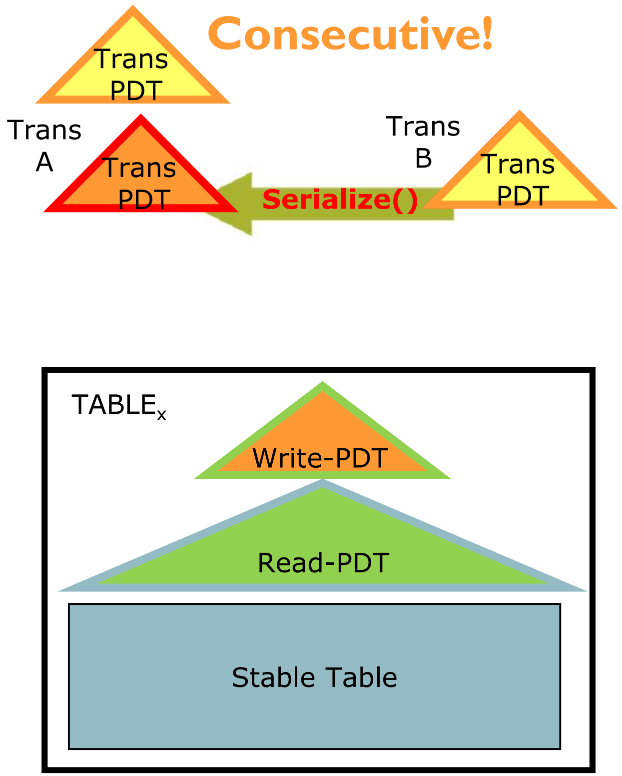


Optimistic Concurrency Control



- Two concurrent transactions
- A commits before B
- Can not commit B into modified write-PDT!
  - A changed RID enumeration

Optimistic Concurrency Control



- Two concurrent transactions
- A commits before B
- Can not commit B into modified write-PDT!
  - A changed RID enumeration
- Serialize(A, B)
  - Makes *aligned* PDTs *consecutive*
  - *MAY FAIL!!* ➔ *trans abort*

    = *succeeds if no conflict*

    = *write set intersection*

Summary

- Vectorized execution
  - Is what makes it blindingly fast
  - + Just-In-Time compilation (DaMoN 2011)
  - + Multi-Core Parallelism

Keeping I/O in balance with CPU

- Columnar Storage
  - Saves I/O bandwidth
- New lightweight compression schemes (PFOR,PDICT,..)
  - 10x faster than fastest Zipf
- MinMax Indices (not discussed)
- Cooperative Scans
- Lots of RAID/SSD experiments (not dicussed)
- Multi-table clustering (not discussed)

TPC-H Stories

Getting To Be the TPC-H Champ

- Fastest non-MPP analytical database system

www.tpc.org/tpch/results/tpch_perf_results.asp?resulttype=noncluster

TPC Transaction Processing Performance Council

SEARCH

Advanced Search

The TPC defines transaction processing and database benchmarks and delivers trusted results to the industry.

- Home
- Results
  - TPC-C
  - TPC-E
  - TPC-H
- Benchmarks
  - TPC-C
  - TPC-E
  - TPC-H
  - Pricing Spec
  - TPC Energy
  - Obsolete
  - - TPC-A
  - - TPC-B
  - - TPC-D
  - - TPC-R
  - - TPC-W
  - - TPC-App
- Technical Articles
- Related Links
- Press
- About the TPC
  - What is the TPC
  - Mailing List
  - Applications
  - Documentation
- Who We Are
  - Members
  - Affiliates
- Member Login
- Contact Us

100 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating
1	INGRES	HP ProLiant DL380 G7	251,561	.38 USD	NR	03/31/11	VectorWise 1.5	RedHat Enterprise Lin
2	hp invent	HP ProLiant DL380 G7	73,974	.58 USD	5.93	07/02/10	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Se Enterprise Edition
3	hp invent	HP ProLiant DL385 G7	71,438	.51 USD	6.48	07/14/10	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Se Enterprise Edition

300 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operat
1	DELL	Dell PowerEdge R910 using VectorWise 1.6	400,931	.35 USD	2.38	06/30/11	VectorWise 1.6	RedHat Enterprise
2	hp invent	HP ProLiant DL580 G7	121,345	.65 USD	10.33	09/14/10	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Enterprise Edition
3	hp invent	HP ProLiant DL585 G7	107,561	1.08 USD	9.58	06/21/10	Microsoft SQL Server 2008 R2 Enterprise Edition	Microsoft Windows Enterprise Edition

1,000 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Oper
1	DELL	Dell PowerEdge R910 using VectorWise 1.6	436,788	.88 USD	NR	06/30/11	VectorWise 1.6	RedHat Enterp

TPC-H

- political situation in TPC
  - H➔DS transition
  - hardware companies rule

- 02/2011: first official results 100GB ➔ 251K

  (compared to 71K SQLserver)
  - HP DL380 144GB, self-financed
  - auditor cost: >$20K, two site visits needed..
- 04/2011: Exasol clustered results
- 05/2011: Dell partnership
  - 100GB, 300GB, 1TB ➔ 303K, 401K, 436K

    (1TB compared to 171K SQLserver @ 80core)
  - Provided 1TB 32-core R910 machine ($50K)

TPC-H

Power Run: 1 stream = 22 queries+ updates on idle system

- challenge: update performance without index support
- challenge: **multi-core speedup beyond 16 cores**
  - (bandwidth limitations, affinity, cache coherence traffic, TLB misses)

Throughput Run: many streams in parallel

- 100GB: 1 stream/core best (12 streams)
- 1TB:
  - 1 stream/core for 32 cores consumed too much RAM
  - run 8x4 cores instead (automatic parallelism tuning)
  - performance improvements in PDTs

The Spin-Off Story

Ingredients For a Spin-Off

- cool technology in prototype state
  - MonetDB/X100 aka VectorWise
- a team with diverse capabilities

- (moral/legal) support from your scientific employer
  - permission for time off
  - reasonable terms for IP
- money, business case
  - Ingres funds development (and donates experts)
  - 2-year option period

INGRES

- A true forefather of our field (“legacy”)
  - founded by Stonebraker 1987
  - small market share but loyal customer base
- 1978 RTI → ASK → CA → Ingres → Actian
  - 2006: bought out by venture fund
  - 2011: renamed Actian
- very distributed organization
  - Redwood City, Ottawa, Ilmenau, London, NY State, Sydney, ..
- focus on OLTP
  - Plus application support (e.g. OpenROAD 4GL)

Why Ingres?

- We had second thoughts ourselves…
  - Hey, do they still exist?
  - Lost the RDBMS war to Oracle

but

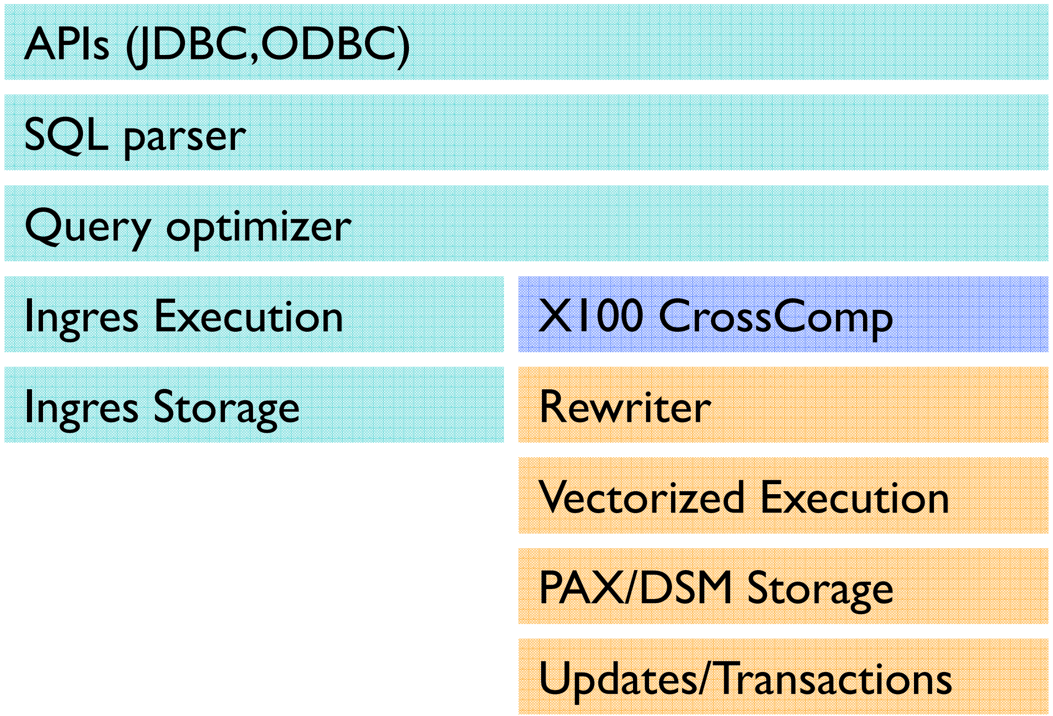
- DBMS market is very mature
  - Very tough for non-MPP startups (impossible)
    - VectorWise is for the mass market (up to few TBs)
    - dominated by…
  - Oracle, IBM, Microsoft, SAP
    - do not want to get a free hand
    - large organizations, political minefield
  - Ingres and VectorWise complemented each other

Timeline Of Ingres VectorWise

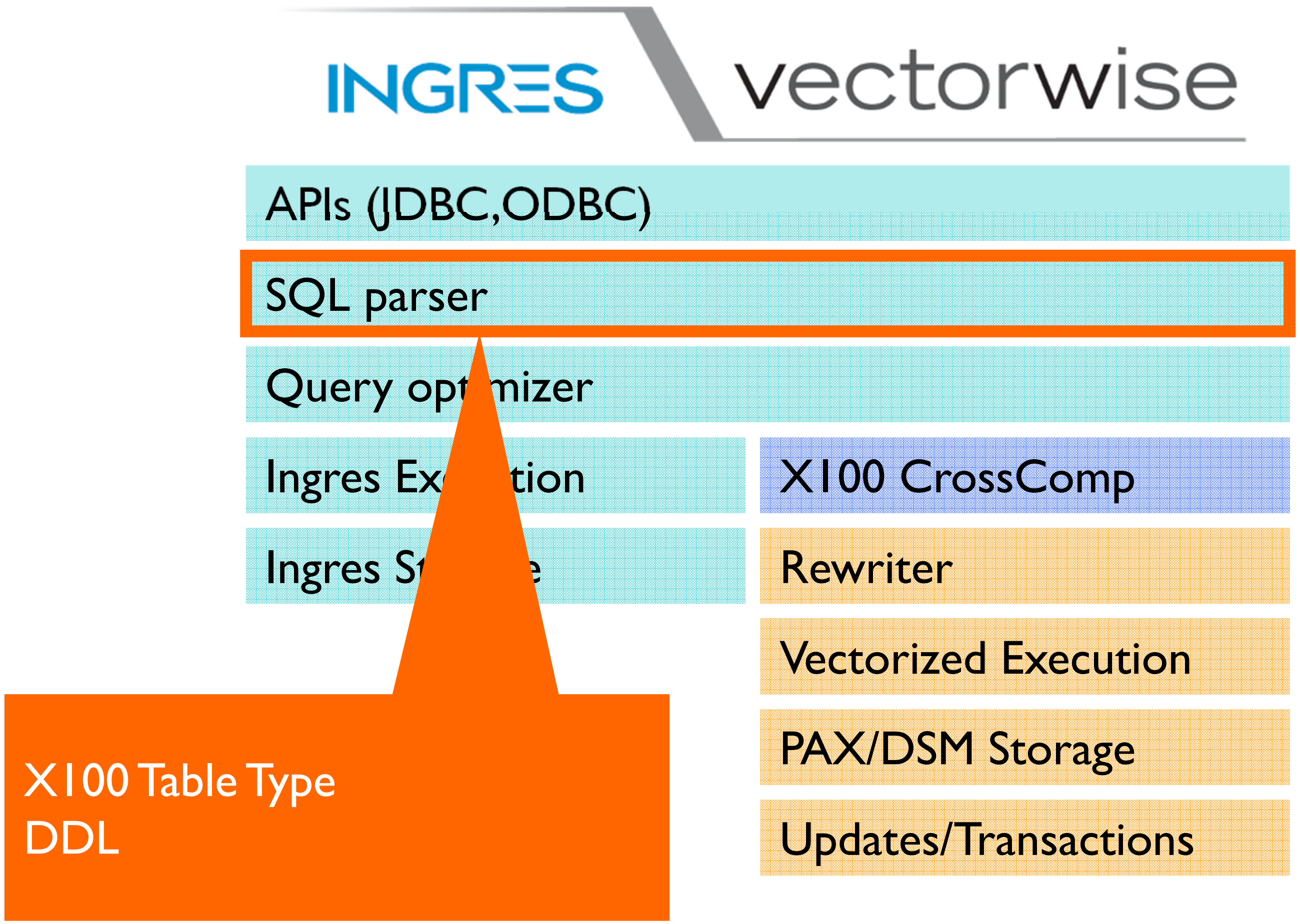
- first Ingres contacts late 07
- negotiations 02/08 → 09/08
- founded 08/08/08
- announced 29/7/09
- demo at IDF fall 09
- alpha in 12/09
- beta in 02/10
- first released in 07/10

(more juice details, not in sheets)

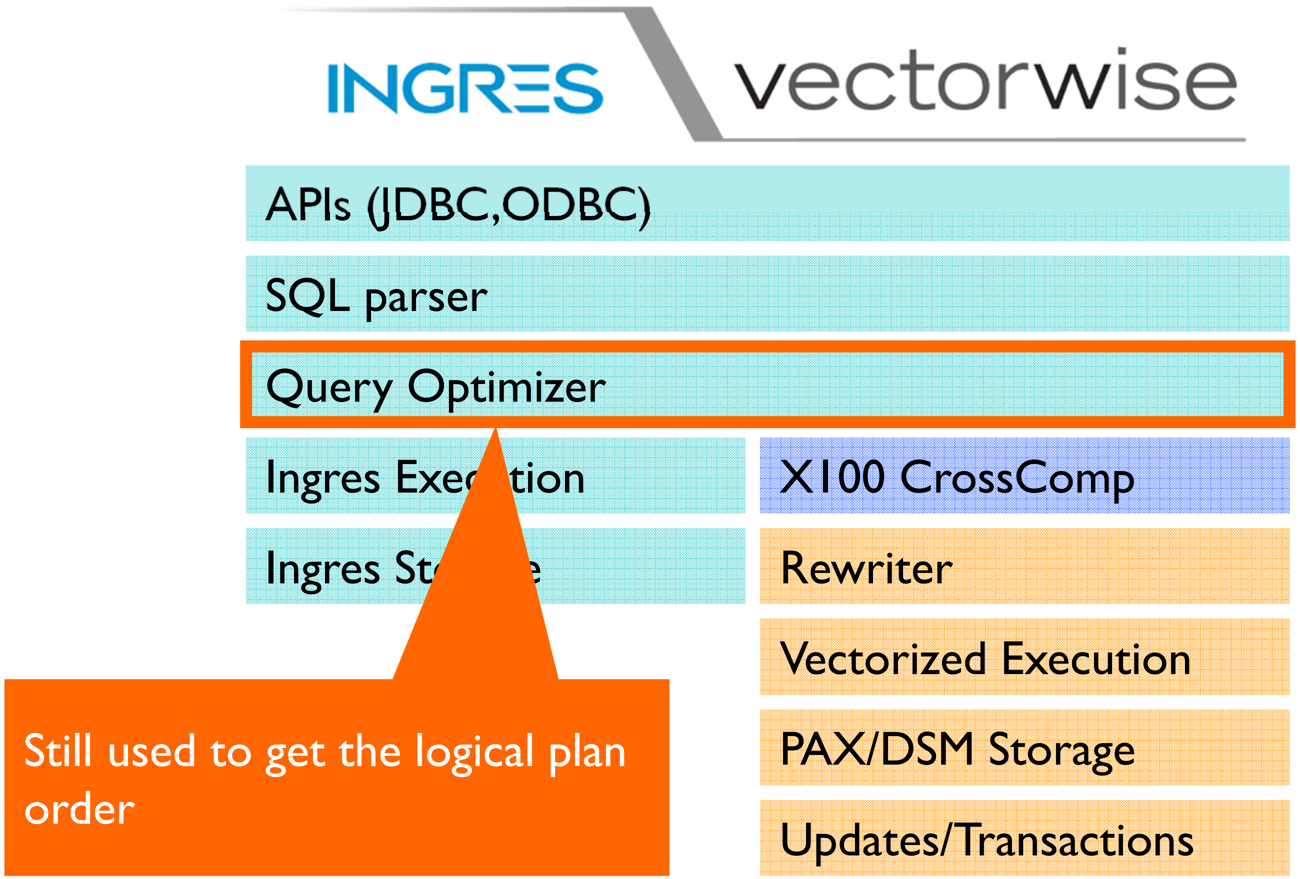
Architecture



Architecture

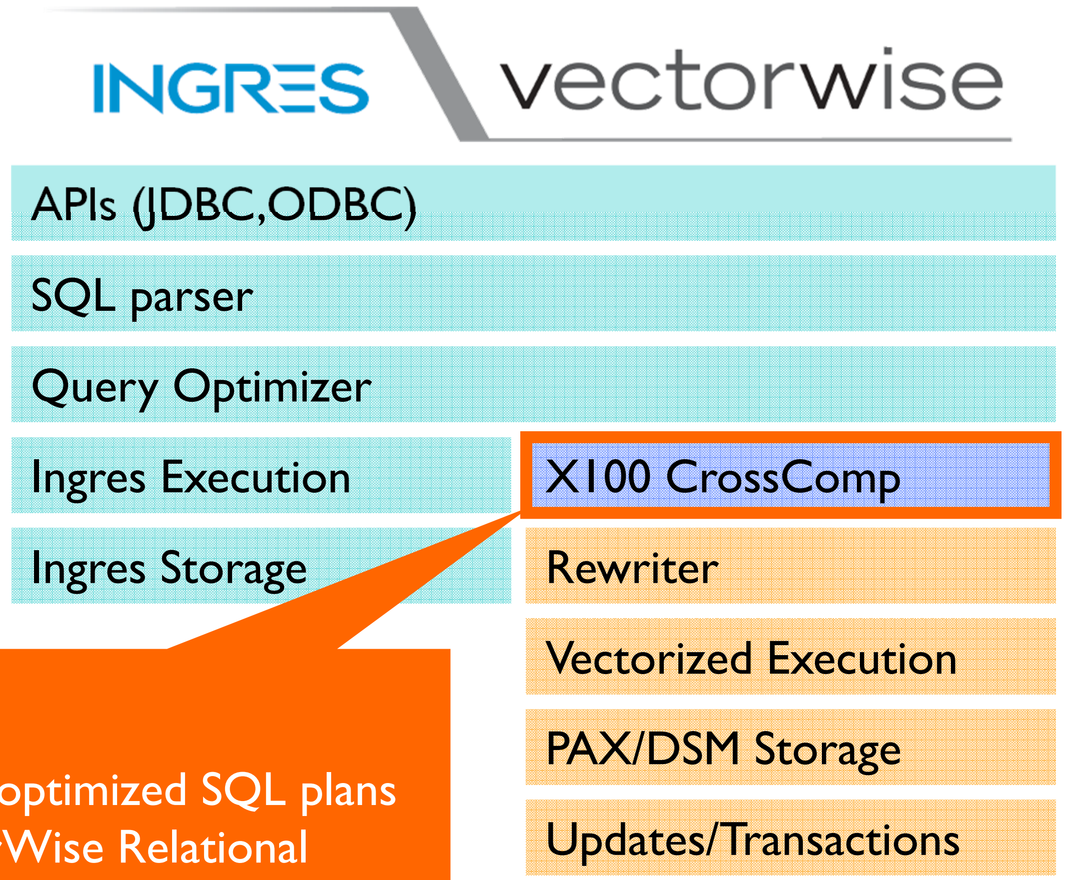


Architecture



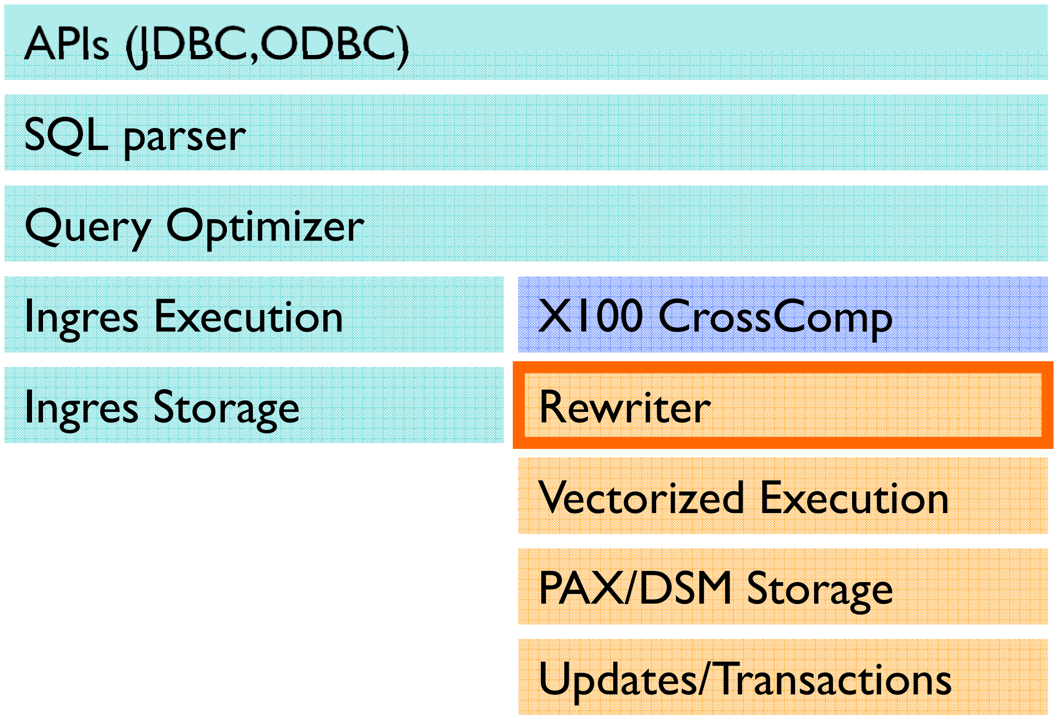
Centrum Wiskunde & Informatica

Architecture



Translates optimized SQL plans into VectorWise Relational Algebra

Architecture



How Can I Get it?

- Binary Releases
  - Download from www.ingres.com/vectorwise

- Source: **Academic Licensing Program**
  - CWI
  - University Ilmenau
  - University Edinburg
  - University Tuebingen
  - Yale, ETH, Barcelona Supercomputerin

  - contact me for details (boncz@cwi.nl)

Active Research Topics

Multi-Core Parallelism
Recycling intermediates
Multi-Dim Clustering
Just-in-time Compilation
Predictive Buffer Manager
Compressed Execution
…

Thanks! Questions?

- www.actian.com/vectorwise