The Story Of

Peter Boncz
Senior Research Scientist @ CWI
Senior Lecturer @ Vrije Universiteit Amsterdam
Architect & Co-founder MonetDB
Architect & Co-founder VectorWise

The Story of VectorWise - Keynote
BDA 25/10/2012, Rabat Morocco
Getting To Be the TPC-H Champ

- Fastest non-MPP analytical database system

---

<table>
<thead>
<tr>
<th>Rank</th>
<th>Company</th>
<th>System</th>
<th>QPH</th>
<th>Price/QPH</th>
<th>Watts/KQPH</th>
<th>System Availability</th>
<th>Database</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INGRES</td>
<td>HP ProLiant DL380 G7</td>
<td>231.561</td>
<td>.38 USD</td>
<td>NR</td>
<td>03/31/2012</td>
<td>VectorWise 1.5</td>
<td>RedHat Enterprise Linux</td>
</tr>
<tr>
<td>2</td>
<td>HP Invent</td>
<td>HP ProLiant DL380 G7</td>
<td>73.974</td>
<td>.58 USD</td>
<td>5.93</td>
<td>07/02/2010</td>
<td>Microsoft SQL Server 2008 R2 Enterprise Edition</td>
<td>Microsoft Windows 7 Enterprise Edition</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Rank</th>
<th>Company</th>
<th>System</th>
<th>QPH</th>
<th>Price/QPH</th>
<th>Watts/KQPH</th>
<th>System Availability</th>
<th>Database</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DELL</td>
<td>Dell PowerEdge R910 using VectorWise 1.6</td>
<td>400,931</td>
<td>.35 USD</td>
<td>2.38</td>
<td>06/30/2010</td>
<td>VectorWise 1.6</td>
<td>RedHat Enterprise Linux</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Rank</th>
<th>Company</th>
<th>System</th>
<th>QPH</th>
<th>Price/QPH</th>
<th>Watts/KQPH</th>
<th>System Availability</th>
<th>Database</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DELL</td>
<td>Dell PowerEdge R910 using VectorWise 1.6</td>
<td>436,788</td>
<td>.88 USD</td>
<td>NR</td>
<td>06/30/2011</td>
<td>VectorWise 1.6</td>
<td>RedHat Enterprise Linux</td>
</tr>
</tbody>
</table>
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?

**row-store**

<table>
<thead>
<tr>
<th>Date</th>
<th>Store</th>
<th>Product</th>
<th>Customer</th>
<th>Price</th>
</tr>
</thead>
</table>

+ easy to add/modify a record
- might read in unnecessary data

**column-store**

<table>
<thead>
<tr>
<th>Date</th>
<th>Store</th>
<th>Product</th>
<th>Customer</th>
<th>Price</th>
</tr>
</thead>
</table>

+ only need to read in relevant data
- tuple writes require multiple accesses

➤ suitable for read-mostly, read-intensive, large data repositories
➤ OLAP, not OLTP
“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 $\rightarrow$ IO thrashing
- Solution: large prefetch buffer for each column
### Compression: Run-length Encoding

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Product ID</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Q1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Q1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Q1</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>Q1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Q1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Q2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Q2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Q2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Product ID</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1, 1, 300</td>
<td>(1, 1, 5)</td>
<td>5</td>
</tr>
<tr>
<td>Q2, 301, 6</td>
<td>(2, 6, 2)</td>
<td>7</td>
</tr>
<tr>
<td>Q3, 307, 500</td>
<td>(1, 301, 3)</td>
<td>9</td>
</tr>
<tr>
<td>Q4, 807, 600</td>
<td>(2, 304, 1)</td>
<td>6</td>
</tr>
</tbody>
</table>

...
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

```
Product ID     ID: 1    ID: 2    ID: 3    ...
1             1       0       0       0
1             1       0       0       0
1             1       0       0       0
2             0       1       0       0
2             0       1       0       0
2             0       1       0       0
3             0       0       1       0
3             0       0       1       0
3             0       0       1       0
...           ...     ...     ...     ...
```

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

The Story of VectorWise - Keynote BDA 25/10/2012, Rabat Morocco
Operating Directly on Compressed Data

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

Index Lookup + Offset jump

The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco
Operating Directly on Compressed Data

Block API

Data
- isOneValue()
- isValueSorted()
- isPosContiguous()
- isSparse()
- getNext()
- decompressIntoArray()
- getValueAtPosition(pos)
- getMin()
- getMax()
- getSize()

Aggregation Operator
Selection Operator
Compression-Aware Scan Operator

Daniel Abadi
C-Store
ACM Dissertation Award 2010
founder Hadapt

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

The Story of VectorWise - Keynote BDA 25/10/2012, Rabat Morocco
The History Of

\[ \text{vectorwise} = \text{monetdb} + \text{INGRES} \]
MonetDB

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

The Story of VectorWise - Keynote BDA 25/10/2012, Rabat Morocco
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

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

Simple, hardcoded semantics in operators
MATERIALIZED intermediate results
“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
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

\[ 7 \times 50 \]

\[ \text{mult}(\text{int},\text{int}) \rightarrow \text{int} \]
“Vectorized In Cache Processing”

vector = array of ~100

generated 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
13

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

VectorWise engine

CPU cache

RAM

ColumnBM
(buffer manager)

(raid)

Disk(s)

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

The Story of VectorWise - Keynote BDA 25/10/2012, Rabat Morocco
Vectors are only the in-cache representation

RAM & disk representation might actually be different

(vectorwise uses both PAX & DSM)
Varying the vector size

Less and less iterator.next() and primitive function calls ("interpretation overhead")
Varying the Vector size

Vectors start to exceed the CPU cache, causing additional memory traffic.
“MonetDB/X100” (VectorWise)

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

<table>
<thead>
<tr>
<th></th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C program</td>
<td>0.2</td>
</tr>
<tr>
<td>VectorWise</td>
<td>0.6</td>
</tr>
<tr>
<td>MonetDB</td>
<td>3.7</td>
</tr>
<tr>
<td>MySQL</td>
<td>26.2</td>
</tr>
<tr>
<td>DBMS “X”</td>
<td>28.1</td>
</tr>
</tbody>
</table>

“MonetDB/X100: Hyper-Pipelining Query Execution”
Boncz, Zukowski, Nes, CIDR’05
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

- 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

Positional Differential Updates

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

Notation:
- \(\text{TABLE}_x\): state of TABLE at time \(x\)
- \(\text{SID}(t):\) StableID
  - Position of tuple \(t\) in immutable base \(\text{TABLE}_0\) \(\leftarrow\) Stable
- \(\text{RID}_x(t):\) RowID
  - Position of visible tuple \(t\) at time \(x\) \(\leftarrow\) VOLATILE!
  - \(\text{SID}(t) = \text{RID}_0(t)\)

### SID/RID Example

<table>
<thead>
<tr>
<th>SID</th>
<th>STORE</th>
<th>PROD</th>
<th>NEW</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>London</td>
<td>chair</td>
<td>N</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>London</td>
<td>stool</td>
<td>N</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>London</td>
<td>table</td>
<td>N</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Paris</td>
<td>rug</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Paris</td>
<td>stool</td>
<td>N</td>
<td>5</td>
</tr>
</tbody>
</table>

### TABLE₀

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

### TABLE₁

<table>
<thead>
<tr>
<th>SID</th>
<th>STORE</th>
<th>PROD</th>
<th>NEW</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Berlin</td>
<td>chair</td>
<td>Y</td>
<td>20</td>
</tr>
<tr>
<td>0</td>
<td>Berlin</td>
<td>cloth</td>
<td>Y</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>Berlin</td>
<td>table</td>
<td>Y</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>London</td>
<td>chair</td>
<td>N</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>London</td>
<td>stool</td>
<td>N</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>London</td>
<td>table</td>
<td>N</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Paris</td>
<td>rug</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Paris</td>
<td>stool</td>
<td>N</td>
<td>5</td>
</tr>
</tbody>
</table>
SIDs and RIDs

- $\text{RID}(t) = \text{SID}(t) + \Delta(t)$
- $\Delta(t) = \#\text{inserts before } t - \#\text{deletes before } t$

- SID and RID are monotonically increasing
  - organize positional updates on SID in a counting B-Tree that keeps track cumulative deltas ($\Delta$)
  - **Positional Delta Tree (PDT)**
    - SIDs are stable
    - Only need to maintain cumulative $\Delta$ on path root ➔ leaf
PDT Example

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

TABLE$_0$

<table>
<thead>
<tr>
<th>S ID</th>
<th>STORE</th>
<th>PROD</th>
<th>NEW</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>London</td>
<td>chair</td>
<td>N</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>London</td>
<td>stool</td>
<td>N</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>London</td>
<td>table</td>
<td>N</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Paris</td>
<td>rug</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Paris</td>
<td>stool</td>
<td>N</td>
<td>5</td>
</tr>
</tbody>
</table>

The Story of VectorWise - Keynote  BDA
25/10/2012, Rabat Morocco
The Story of VectorWise - Keynote BDA
25/10/2012, Rabat Morocco
The Story of VectorWise - Keynote  BDA
25/10/2012, Rabat Morocco
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 ..”
  - \( \text{RID domain of child PDT} = \text{SID domain of parent PDT} \)

\[
\begin{array}{c|c|c}
\text{PDT} & t_2 & t_3 \\
\hline
\text{PDT} & t_0 & t_1
\end{array}
\]

\( t_2 = t_1 \), \( t_0 \) vs are \( t_3 \) consecutive

The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco
Stacking PDTs

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

\[
\text{PDT}_{t_2}^{t_3} \quad \text{vs} \quad \text{PDT}_{t_0}^{t_1}
\]

- consecutive
- aligned

\[ t_2 = t_1 \]
\[ t_2 = t_0 \]

“same base”
Stacking PDTs

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

<table>
<thead>
<tr>
<th>PDT $t_2$ vs PDT $t_0$</th>
<th>are</th>
</tr>
</thead>
<tbody>
<tr>
<td>consecutive</td>
<td>$t_2 = t_1$</td>
</tr>
<tr>
<td>aligned</td>
<td>$t_2 = t_0$</td>
</tr>
<tr>
<td>overlapping</td>
<td>“same base”</td>
</tr>
<tr>
<td>$[t_2, t_3]$ overlaps $[t_0, t_1]$</td>
<td>“uncomparable” / “incompatible”</td>
</tr>
</tbody>
</table>

Table

The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco
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, \texttt{Propagate()} trans-PDT into write-PDT
Optimistic Concurrency Control

- Two concurrent transactions
Optimistic Concurrency Control

- Two concurrent transactions
- A commits before B

The Story of VectorWise - Keynote BDA
25/10/2012, Rabat Morocco
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!* $\Rightarrow$ *trans abort*
  - $\Rightarrow$ *succeeds if no conflict*
  - $\Rightarrow$ *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 discussed)
- Multi-table clustering (not discussed)
TPC-H Stories
Getting To Be the TPC-H Champ

- Fastest non-MPP analytical database system
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)

*The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco*
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

The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco
A true forefather of our field ("legacy")
- founded by Stonebraker 1987
- small market share but loyal customer base
- 1978 RTI ➔ ASK ➔ CA ➔ Ingres ➔ 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

<table>
<thead>
<tr>
<th>INGRES</th>
<th>vectorwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIs (JDBC, ODBC)</td>
<td></td>
</tr>
<tr>
<td>SQL parser</td>
<td></td>
</tr>
<tr>
<td>Query optimizer</td>
<td></td>
</tr>
<tr>
<td>Ingres Execution</td>
<td>X100 CrossComp</td>
</tr>
<tr>
<td>Ingres Storage</td>
<td>Rewriter</td>
</tr>
<tr>
<td>Vectorized Execution</td>
<td></td>
</tr>
<tr>
<td>PAX/DSM Storage</td>
<td></td>
</tr>
<tr>
<td>Updates/Transactions</td>
<td></td>
</tr>
</tbody>
</table>

The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco
Architecture

APIs (JDBC, ODBC)
- SQL parser
- Query optimizer
- Ingres Execution
- Ingres Storage
- X100 CrossComp
- Rewriter
- Vectorized Execution
- PAX/DSM Storage
- Updates/Transactions

X100 Table Type
DDL
Architecture

APIs (JDBC, ODBC)

SQL parser

Query Optimizer

Ingres Execution

Ingres Storage

X100 CrossComp

Rewriter

Vectorized Execution

PAX/DSM Storage

Updates/Transactions

Still used to get the logical plan order

The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco
Architecture

APIs (JDBC, ODBC)
SQL parser
Query Optimizer
Ingres Execution
Ingres Storage
Rewriter
Vectorized Execution
PAX/DSM Storage
Updates/Transactions

Translates optimized SQL plans into VectorWise Relational Algebra
Architecture

APIs (JDBC, ODBC)
SQL parser
Query Optimizer
Ingres Execution
Ingres Storage
X100 CrossComp
Rewriter
Vectorized Execution
PAX/DSM Storage
Updates/Transactions

The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco
How Can I Get it?

- **Binary Releases**
  - Download from [www.ingres.com/vectorwise](http://www.ingres.com/vectorwise)

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

- 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

---

The Story of VectorWise - Keynote  BDA 25/10/2012, Rabat Morocco
Thanks!

www.actian.com/vectorwise

Questions?