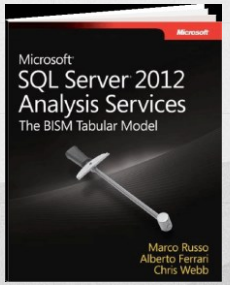
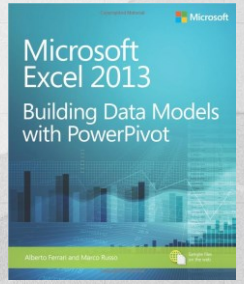
Alberto Ferrari
alberto.ferrari@sqlbi.com

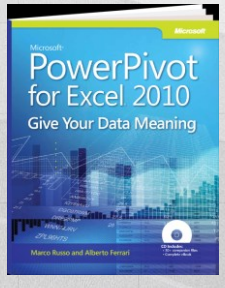
Inside xVelocity in-memory Engine

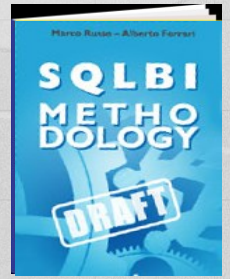
Who's speaking Spaghetti English?

- BI Expert and Consultant
- Founder of www.sqlbi.com
  - Problem Solving
  - Complex Project Assistance
  - DataWarehouse Assesments and Development
  - Courses, Trainings and Workshops
- Book Writer
- Microsoft Business Intelligence Gold Partners
- SSAS Maestro – MVP – MCP

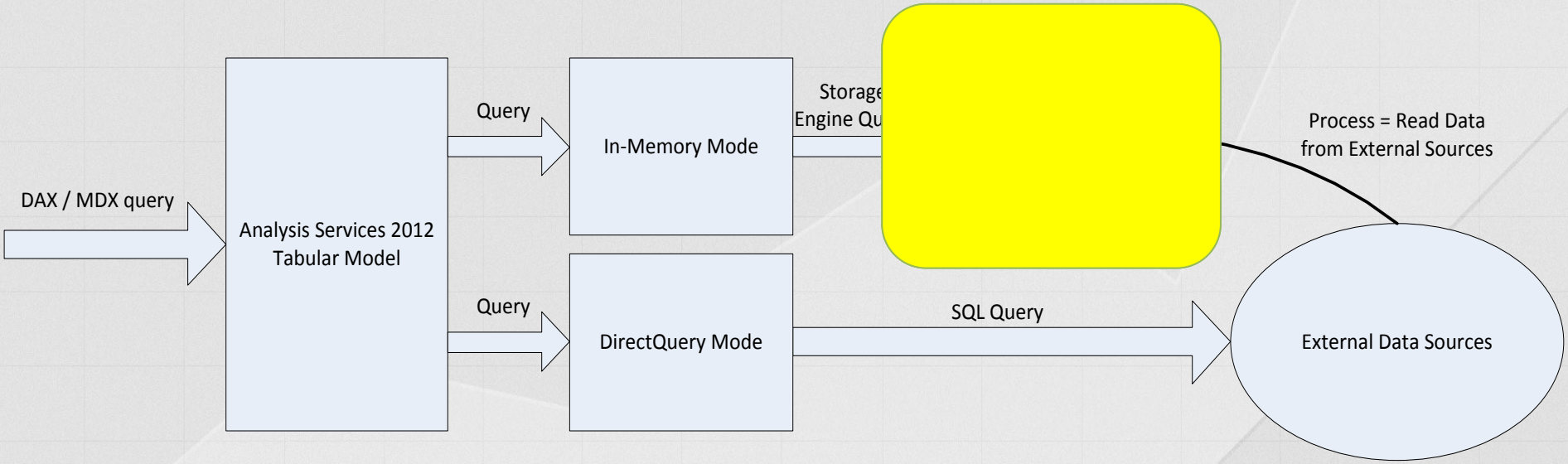








Tabular Query Architecture



sqlbi

Different Query Handling

- In-Memory mode
  - DAX Formula Engine
  - xVelocity in-memory analytics Engine (Vertipaq)
  - Full Tabular options available
- DirectQuery mode
  - DAX to SQL translation
  - SQL Server queries
  - Many limitations (we will see them later)

Agenda

- How xVelocity stores data
- Memory usage and monitoring
- Some basic optimization techniques
- A few best practices

What is xVelocity in-memory?

- It is an in-memory database
- Based on the relational methodology
- Column oriented database

Row Storage Layout

Customers Table

ID	Name	Address	City	State	Bal Due
1	Bob	…	…	…	3,000
2	Sue	…	…	…	500
3	Ann	…	…	…	1,700
4	Jim	…	…	…	1,500
5	Liz	…	…	…	0
6	Dave	…	…	…	9,000
7	Sue	…	…	…	1,010
8	Bob	…	…	…	50
9	Jim	…	…	…	1,300

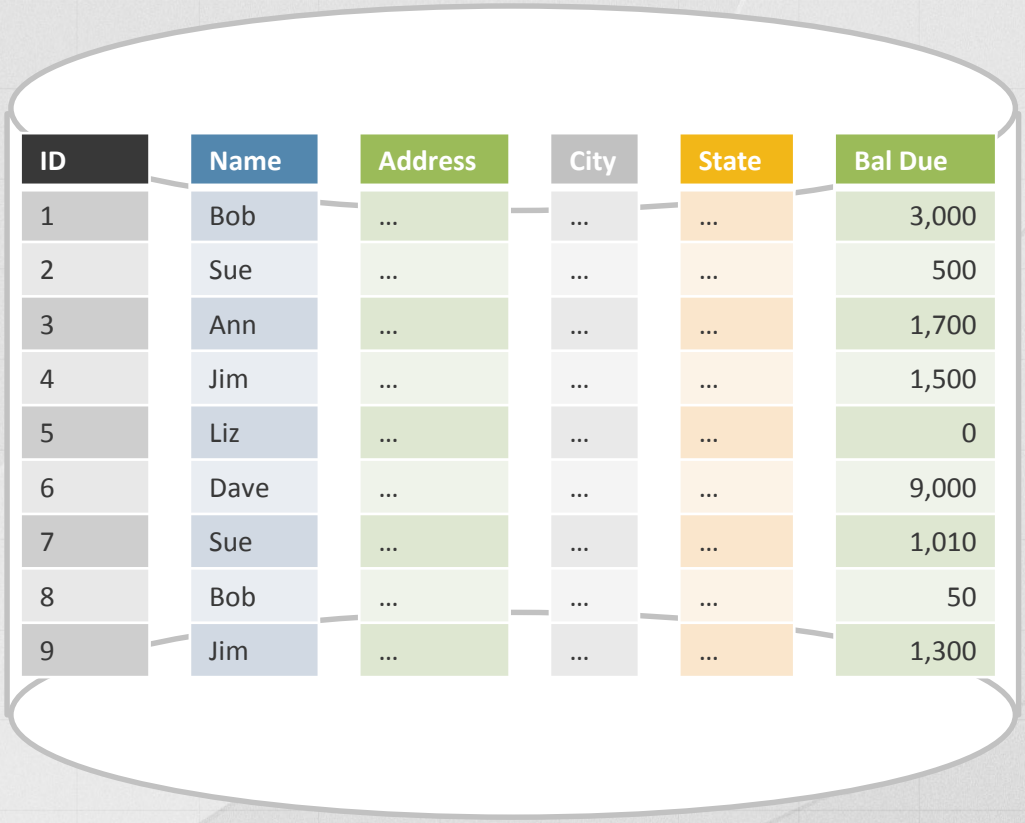
Nothing special here.
This is the standard way database systems have been laying out tables on disk since the mid 1970s.
Technically, it is called a “row store”

sqlbi

Column Storage Layout

Customers Table

ID	Name	Address	City	State	Bal Due
1	Bob	…	…	…	3,000
2	Sue	…	…	…	500
3	Ann	…	…	…	1,700
4	Jim	…	…	…	1,500
5	Liz	…	…	…	0
6	Dave	…	…	…	9,000
7	Sue	…	…	…	1,010
8	Bob	…	…	…	50
9	Jim	…	…	…	1,300



Tables are stored “column-wise” with all values from a single column stored in a single block

sqlbi

Column vs Row Storage

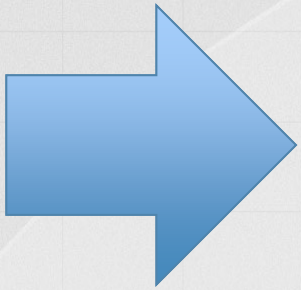
- Column Storage
  - Quick access to a single column
  - Time needed to materialize rows
  - Trade CPU vs I/O
- Row Storage
  - Quick access to a single row
  - No materialization needed
  - Trade I/O vs CPU

Run Length Encoding (RLE)

Quarter
Q1
Q1
Q1
Q1
Q1
...
Q2
Q2
Q2
Q2
Q2
Q2
Q2
Q2
...

ProdID
1
1
1
1
1
2
2
2
...
1
1
1
1
1
2
2
2

Price
100
120
315
100
315
198
450
320
320
150
256
450
192
184
310
251
266



Quarter	Start	Count
Q1	1	310
Q2	311	290
...	...	...

ProdID	Start	Count
1	1	5
2	6	3
...	...	...
1	51	5
2	56	3

Price
100
120
315
100
315
198
450
320
320
150
256
450
192
184
310
251
266

RLE Compression applied only when size of compressed data is smaller than original

sqlbi

Dictionary Encoding

Quarter
Q1
Q1
Q1
Q1
Q2
Q2
…
Q2
Q3
Q3
Q3
Q3
Q4
Q4
Q4
Q4
…

DISTINCT

Q.ID
0
0
0
0
1
1
…
1
2
2
2
2
4
4
4
4
…

R.L.E.

Q.ID	Quarter
0	Q1
1	Q2
2	Q3
3	Q4

Only 4 values.
2 bits are enough to represent it

xVelocity Store

Quarter	Start	Count
0	1	4
1	5	10
2	11	4
3	15	15

sqlbi

xVelocity in-memory Compression

- Dictionary Encoding
  - Happens when necessary
    - Very large ranges of integers are not encoded
  - Makes tables datatype-independent
- RLE Encoding
  - Only if compressed data is smaller than original
  - Strongly depends on data order
  - SSAS automatically chooses best sorting

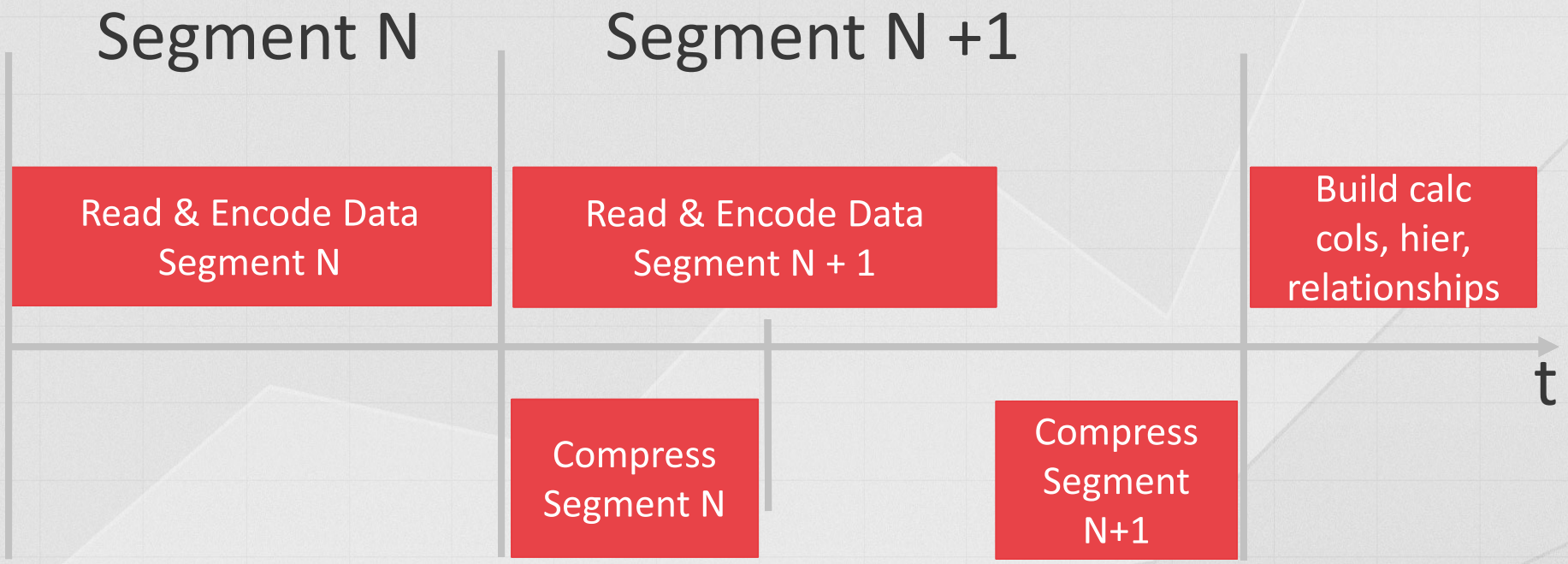
xVelocity in-memory Compression

- Compression comes from
  - Column Store
  - Dictionary Encoding
  - RLE Encoding
- Less RAM used for the in-memory database
- Faster column scans
- 10x is a good average compression ratio
  - Against non-compressed SQL database

Segmentation

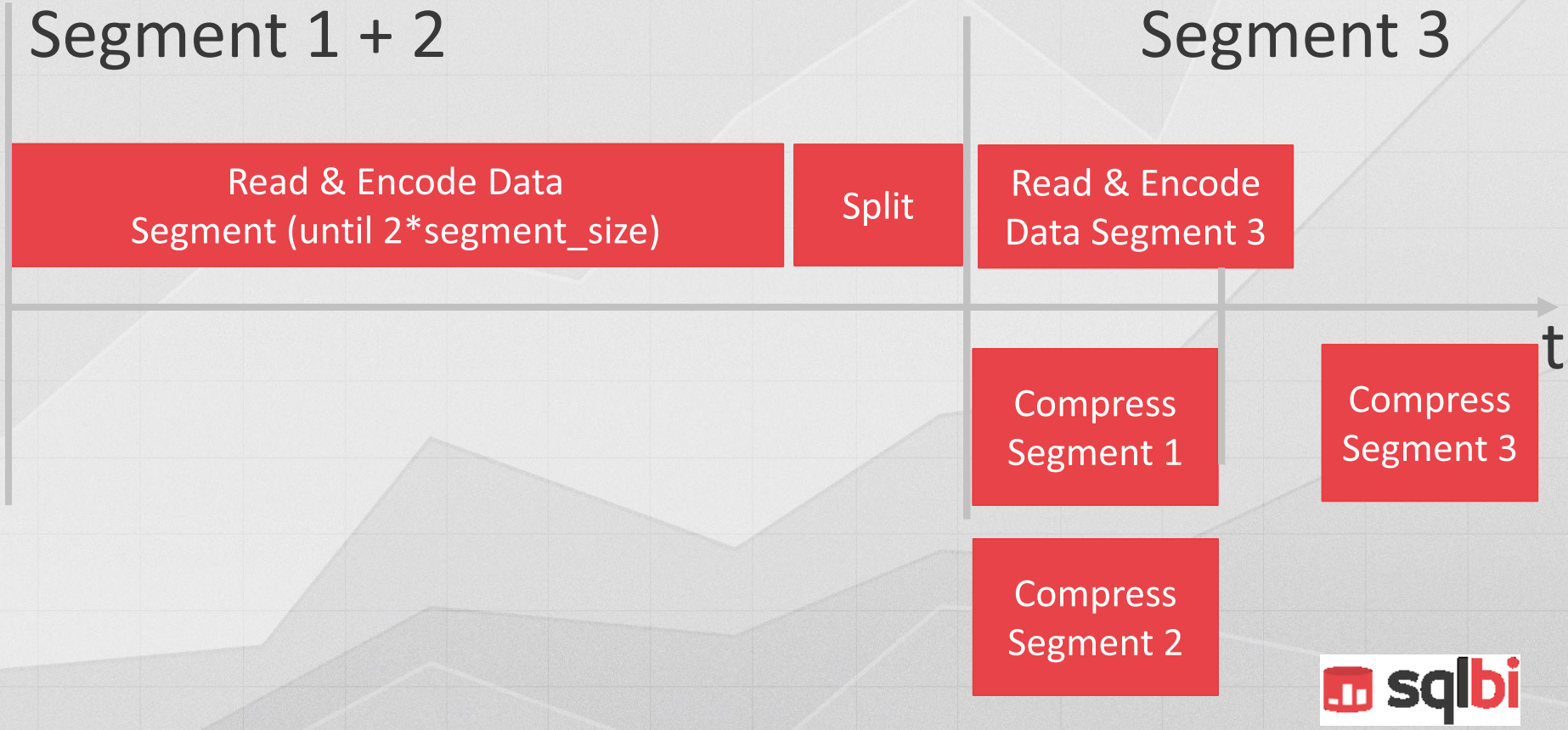
- Each table is divided in segments
  - 8 million rows for each segment in SSAS
  - 1 million rows in PowerPivot
- Dictionary is global to the table
- Bit-sizing is local to the segment
  - Column «Date» uses 4 bits in segment 1
  - Only 2 bits in segment 2
  - ...
- DMV available to query that info

Processing Phases



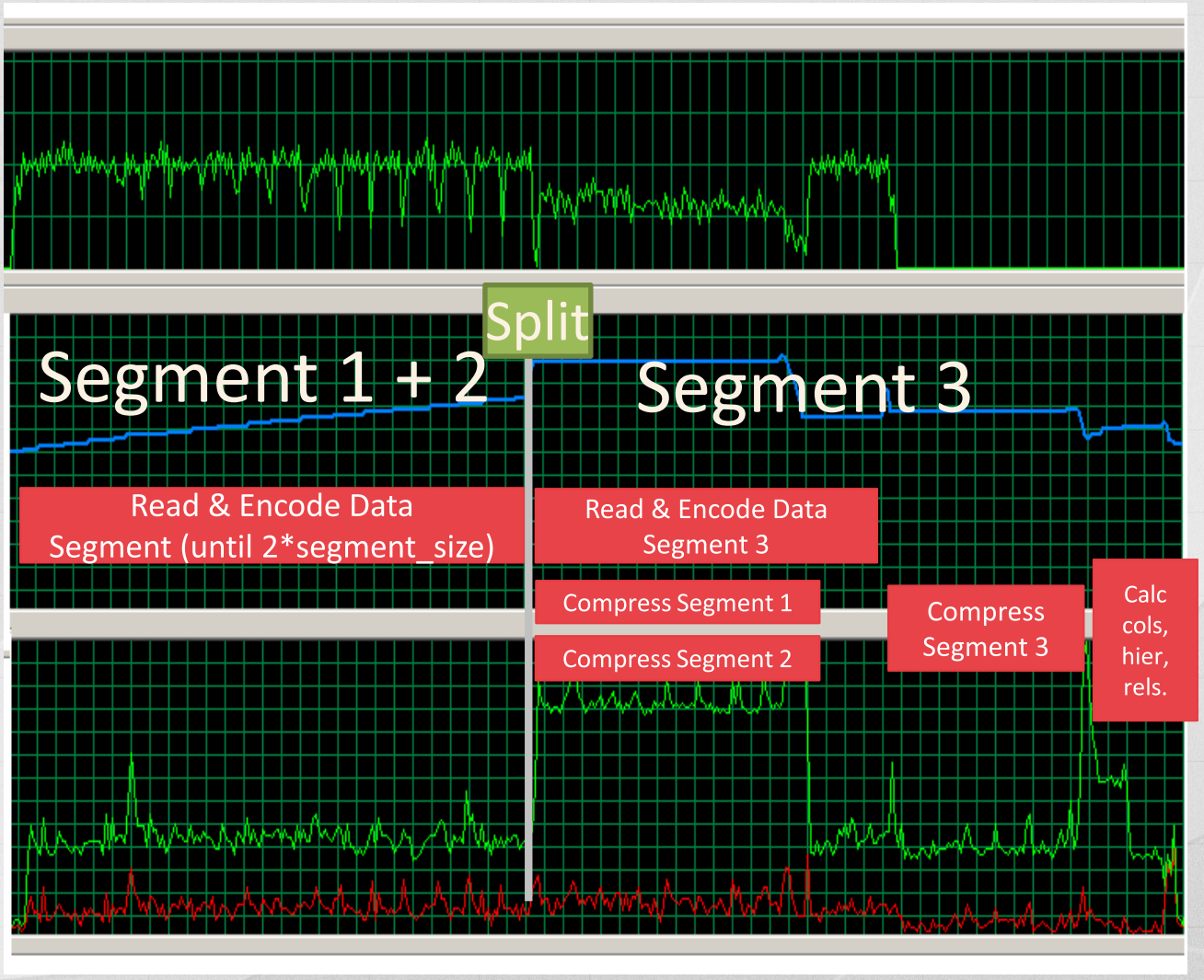
Special case of 3rd segment

- First segment can “stretch” to be twice as large
- Optimizes for smaller lookup tables



sqlbi

Processing - Memory & CPU usage



Network

Memory

CPU

sqlbi

Data Memory Usage

- Memory usage depends on
  - Number of columns
  - Cardinality of each column
  - Data type
  - Number of rows
- Strings
  - Average size is relevant for dictionary size
- No easy formula can be applied

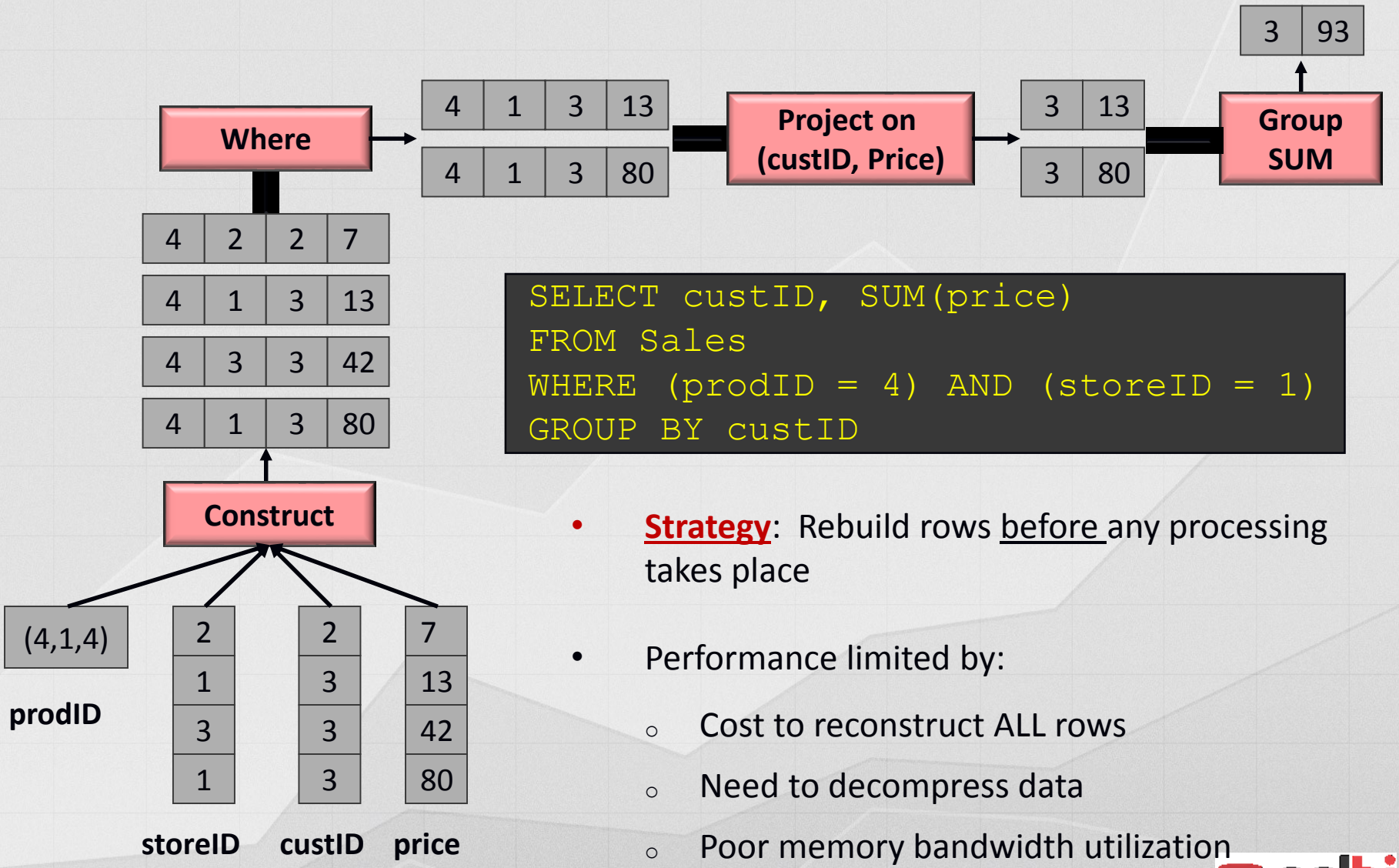
Processing Memory Usage

- Each table is sequential
  - No parallelism on partitions
- Many tables can be loaded in parallel
- Each table
  - Divided in segments (8 million rows each)
- For each segment
  - Load
  - Compress
  - Store
  - Parallelism at the column level

Query Memory Usage

- Simple queries requires some memory
- Complex queries require more memory
  - Spooling of temporary values
  - Materialization of datasets
- Cache requires memory
- Materialization is the big issue

Early Materialization



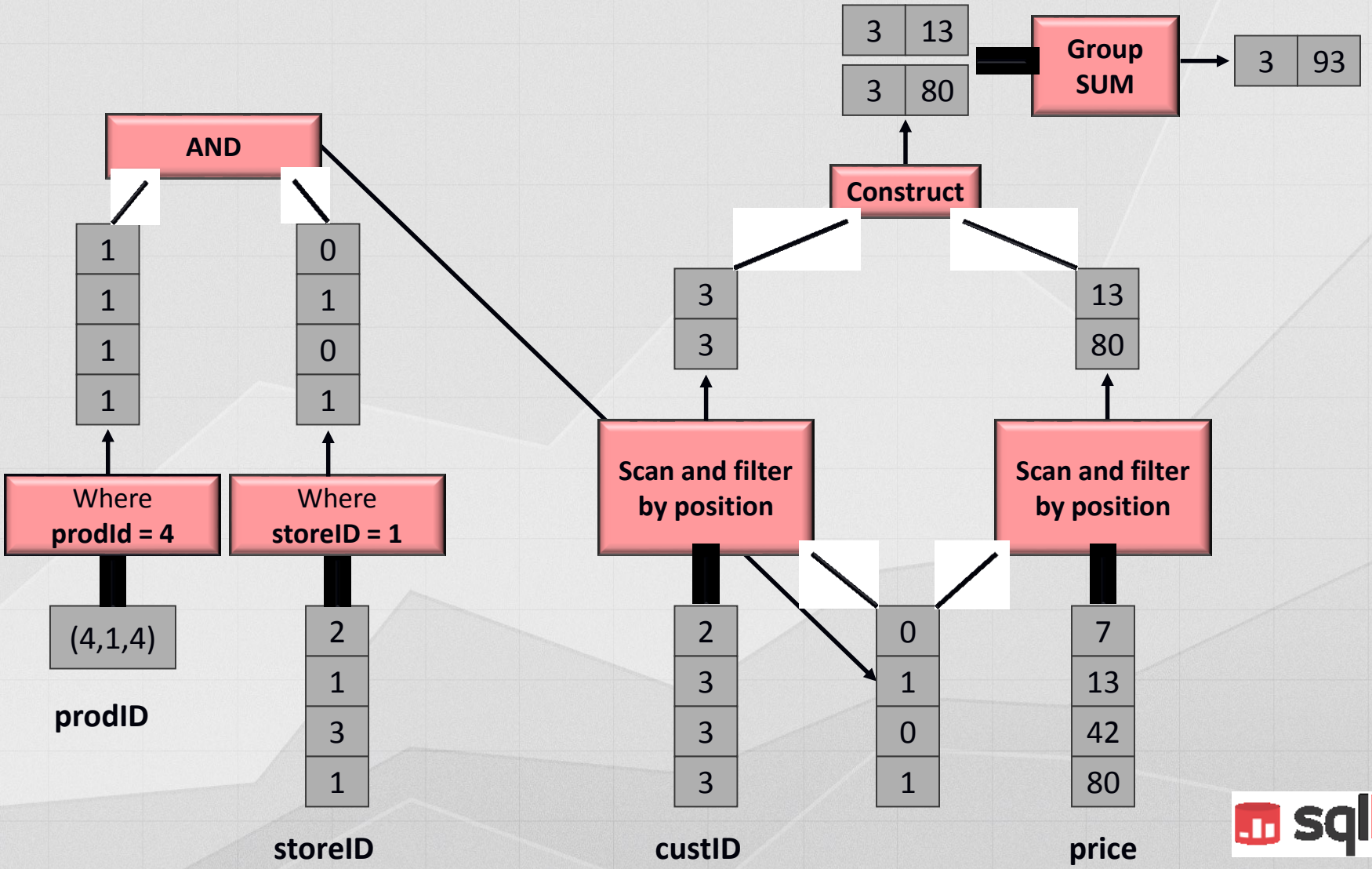

```
SELECT custID, SUM(price)
FROM Sales
WHERE (prodID = 4) AND (storeID = 1)
GROUP BY custID
```

- **Strategy**: Rebuild rows before any processing takes place
- Performance limited by:
  - Cost to reconstruct ALL rows
  - Need to decompress data
  - Poor memory bandwidth utilization

sqlbi

Late Materialization

```
SELECT custID, SUM(price)
FROM Sales
WHERE (prodID = 4) AND (storeID = 1)
GROUP BY custID
```



sqlbi

Materialization

- Materializations happens for
  - Complex Joins
  - Complex Iterators
  - Temporary data spooled for further processing
- Memory requirements
  - Might be more than the whole database
  - Spooled data is not compressed

Storage Internals

- Files in DataDir folder, one folder per database
- File types & file extensions / names
  - **Dictionary**: .DICTIONARY
  - **Data**: .IDF
  - **Indexes**: .IDF
    - POS_TO_ID, ID_TO_POS
  - **Relationships**: GUID + .HIDX
  - **Hierarchies**: .IDF
    - CHILD_COUNT, FIRST_CHILD_POS
    - MULTI_LEVEL_ID, PARENT_POS

sqlbi

Available DMV

You can use DMV to query the server and discover the size of each object

```
--
-- Returns all the DMV
--
select * from $system.discover_schema_rowsets

--
-- Discover memory usage of all objects
--
select * from
    $system.discover_object_memory_usage
```

sqlbi

Available DMV

You can use DMV to query the server and discover the size of each object

```
--
-- Discover details of individual columns
--
select * from
    $system.discover_storage_table_columns


--
-- Discover details of segments
--
select * from
    $system.discover_storage_table_column_segments
```

sqlbi

If you don't like DMV…

- You can avoid typing and remembering DMV
- Using PowerPivot
  - Kasper De Jonge wrote a beautiful PowerPivot data model
  - http://www.powerpivotblog.nl/what-is-using-all-that-memory-on-my-analysis-server-instance
- Just…
  - Open the Excel file
  - Refresh
  - Browse the model

sqlbi

Reduce Dictionary Size

- Reduce number of distinct values
  - DateTime: split in two columns
    - Date
    - Time
  - Floating Point Values: fix precision
    - 10.231 → 10.2
- Reduce strings length
- All this should be done in source data (ie SQL views), not in calculated columns

Reduce Table Size

- Remove useless columns
- Avoid partial results in calculated columns
  - They tend to have many distinct values
  - They increase the number of columns
- Beware of Junk Dimensions
  - Five TinyInt are better than one int
  - Less distinct values
  - Better columnstore data structure

Optimize Degenerate Dimensions

- Storing an ID for DrillThrough is expensive
  - One different value for every row
  - Large dictionary in large fact table
- Consider splitting in more columns
  - Every column has a smaller dictionary
- Impact on query performance
  - Good for drillthrough or single lookup
  - Bad for distinct count / filters
    - Slow response time
    - Requires memory for spooling

Split String Column

Split a 10-character length string into two 5-character strings

```
SELECT
    LEFT( TransactionID, 5 )
        AS TransactionHighID,
    SUBSTRING(
        TransactionID,
        6,
        LEN( TransactionID ) - 5
    ) AS TransactionLowID,
    Quantity,
    Price
FROM Fact
```

Split Integer Column

Split 100 million range in two 10.000 ranges
Beware of possible materialization later on

```
SELECT
    TransactionID / 10000 AS TransactionHighID,
    TransactionID % 10000 AS TransactionLowID,
    Quantity,
    Price
FROM Fact
```

Split Column Optimization

- Splitting saves memory but increases process time
- Query performance penalty for materialization

Number of Columns	Process Time	Cores Used	Disk Size
1 (original)	02:48	1	2,811 MB
2	03:21	up to 8	191 MB
3	03:49	up to 8	129 MB
4	04:01	up to 8	97 MB
8	05:32	up to 8	105 MB

Processing Steps

- Process Data
  - Load, Compress, Store
- Process Other Structures
  - Calculated Columns
  - Indexes
  - Relationships
  - Hierarchies

Memory Usage During Process

- Transactional Process
  - Old data is still in memory
  - New data is processed
  - New data is switched in
- During processing
  - Memory for old data (1x)
  - Memory for processing (2x)
- A total of 3x is needed to process an object

Reduce Processing Memory

- Split processing in steps
  - Divide tables
  - Divide processing steps (data – recalc)
- Issue a ProcessClear in advance
  - Data will not be available
  - A lot of memory will be freed
  - Issued in a different transaction

Weighted Aggregation

In order to compute Mean Price we need to follow a weighted average pattern, using the quantity as the weight

$$MeanPrice = \frac{\sum(Qty \times Price)}{\sum(Qty)}$$

Classical Weighted Avg Solution

Like we do in Multidimensional: add a column to the fact table and use SUM to leverage aggregations and max scanning speed

```
MeanPrice :=

SUM ( [PriceMultipliedByQuantity] )
/
SUM ( [OrderQuantity] )
```

sqlbi

Performance Impact

- Column PriceMultipliedByQuantity
  - Huge number of distinct values
  - Much greater than the source columns
- On a production database
  - Query speed: pretty good
    - Test query: 13 seconds
    - Using many-to-many on a 4 billion rows table
  - Column size: 9GB (RAM!)
- DAX requires a different approach

Weighted Averages: Naïve Formula

- Use SUMX
- Multiplication pushed down to xVelocity
  - Runs in parallel on all cores
  - Does not require table spooling
  - Reduced memory usage during query
- On the same production database
  - Test query: 3 seconds
  - i.e. 4 times faster
- As often: simpler is faster and easier

What to Store in the Cube?

- Tabular has different priorities than Multidimensional
- Distinct Count of values is the top priority

OrderId	ProductId	Quantity	Price	Discount	SalesAmount
1	12	10	2.55	1.5	24.00
2	12	8	2.55	0.4	20.00
…	…	…	…	…	…
…	…	…	…	…	…
847	12	9	2.55	0.95	22.00

sqlbi

What to Store in the Cube?

Sales Amount should be computed at leaf level at query time

These columns are needed

Small number of Distinct Values

Higher Cardinality

sqlbi

Let's play together...

- Now we know the theory
- Time to work with some real data
- Source: Contoso Retail Analysis 2013
- Steps:
  - Restore the database
  - Investigate on the database content
  - Define an optimization plan

sqlbi

xVelocity – Conclusions

- Columnar databases are different
- Dictionary size
  - Fixed amount of RAM
  - Larger for strings
- Segment size
  - Grows with the number of rows
  - Depends on number of distinct values
- Segmentation
  - Drives parallelism at query time
  - Reduces bit usage for many columns

sqlbi

Coming up…

Speaker	Title	Room
Christina E. Leo	Why APPLY?	Theatre
Jennifer Stirrup	Advanced Data Visualisation in Reporting Services 2012	Exhibition B
Denny Cherry	Optimizing SQL Server Performance in a Virtual Environment	Suite 3
Christian Wade	MDX vs. DAX: Currency Conversion Faceoff	Suite 1
Thomas LaRock	Database Design: Size Does Matter!	Suite 2
Peter ter Braake	SSIS 2012 logging and monitoring	Suite 4

twitter #SQLBITS

Consulting

Assessment

Outsourcing

Technical Fellowship

Find out more on

www.sqlbi.com/consulting