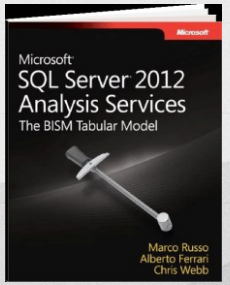
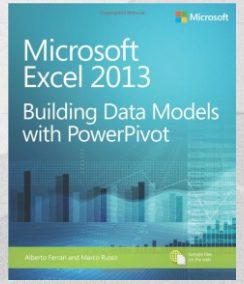
Alberto Ferrari
alberto.ferrari@sqlbi.com

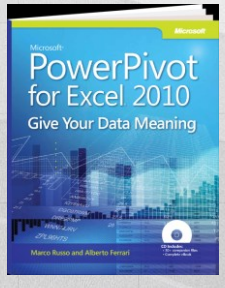
DAX Query Engine Internals

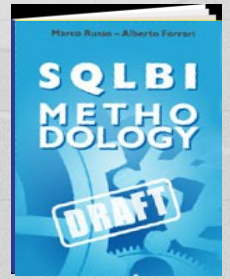
Who's speaking Spaghetti English?

- 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

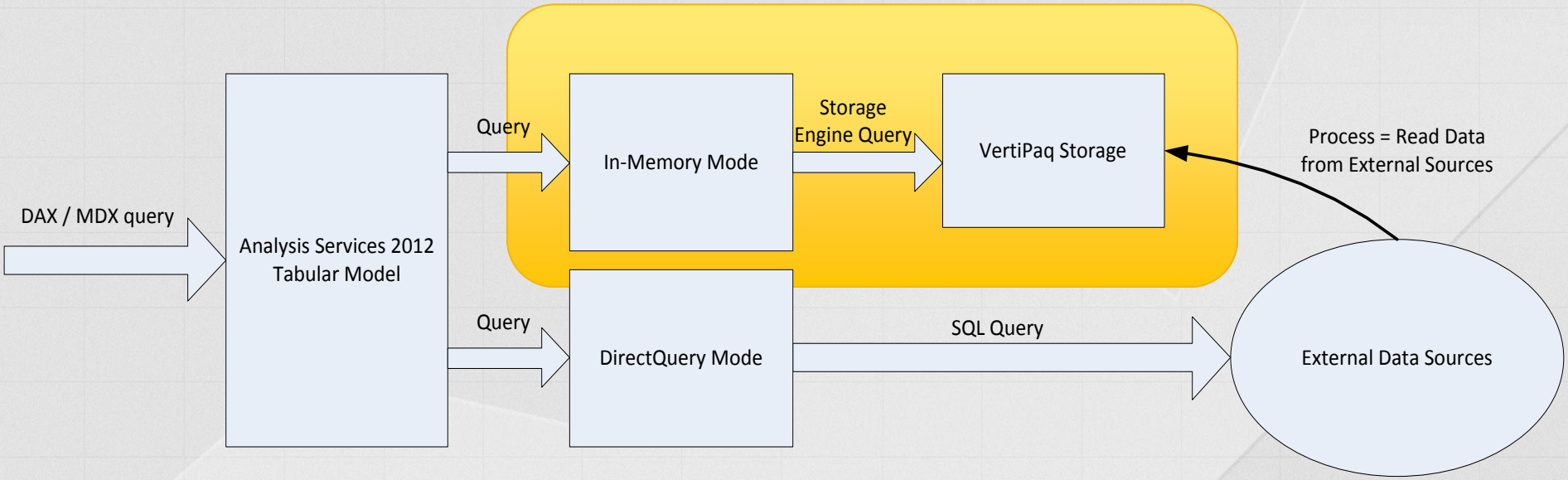








Tabular Query Architecture



Agenda

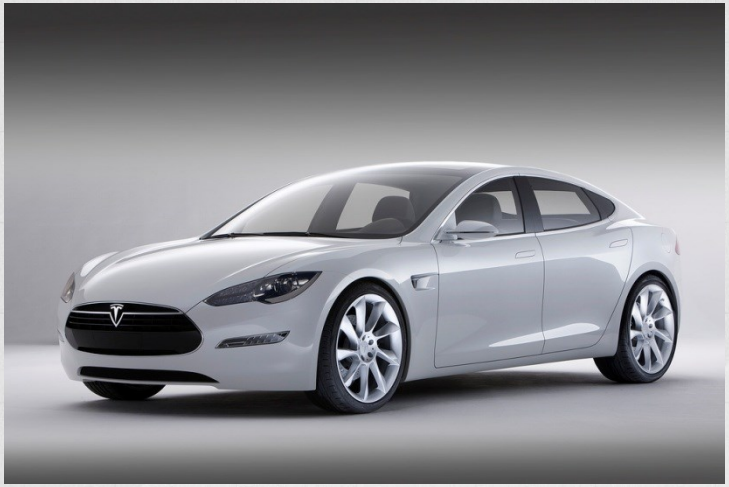
- Tabular Query Architecture
- Monitoring and Query Plans
- Optimization Examples
- Query Examples

Tabular Two Engines

- Formula Engine
  - Handles complex expressions
  - Single threaded
- Storage Engine (VertiPaq / xVelocity)
  - Handles simple expressions
  - Executes queries against the database
  - Multithreaded

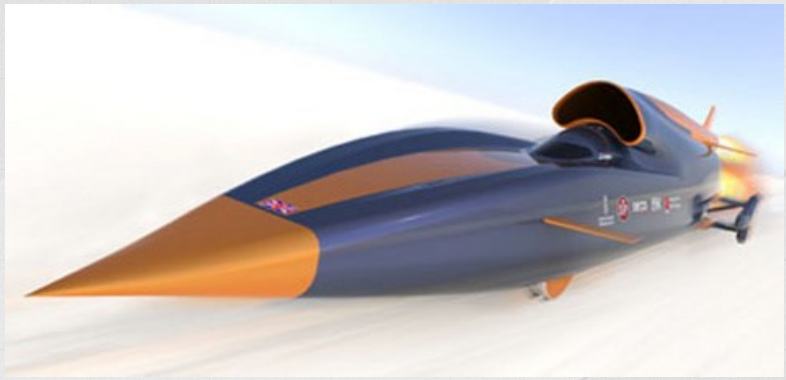
Tabular: Rich & Fast

DAX



- Rich
- Single threaded per query
- Designed for expressivity

VertiPaq Query



- Simple
- One core per segment
- Optimized for speed

sqlbi

Formula Engine

Formula Engine

Vertipaq

Trust the Rain Man

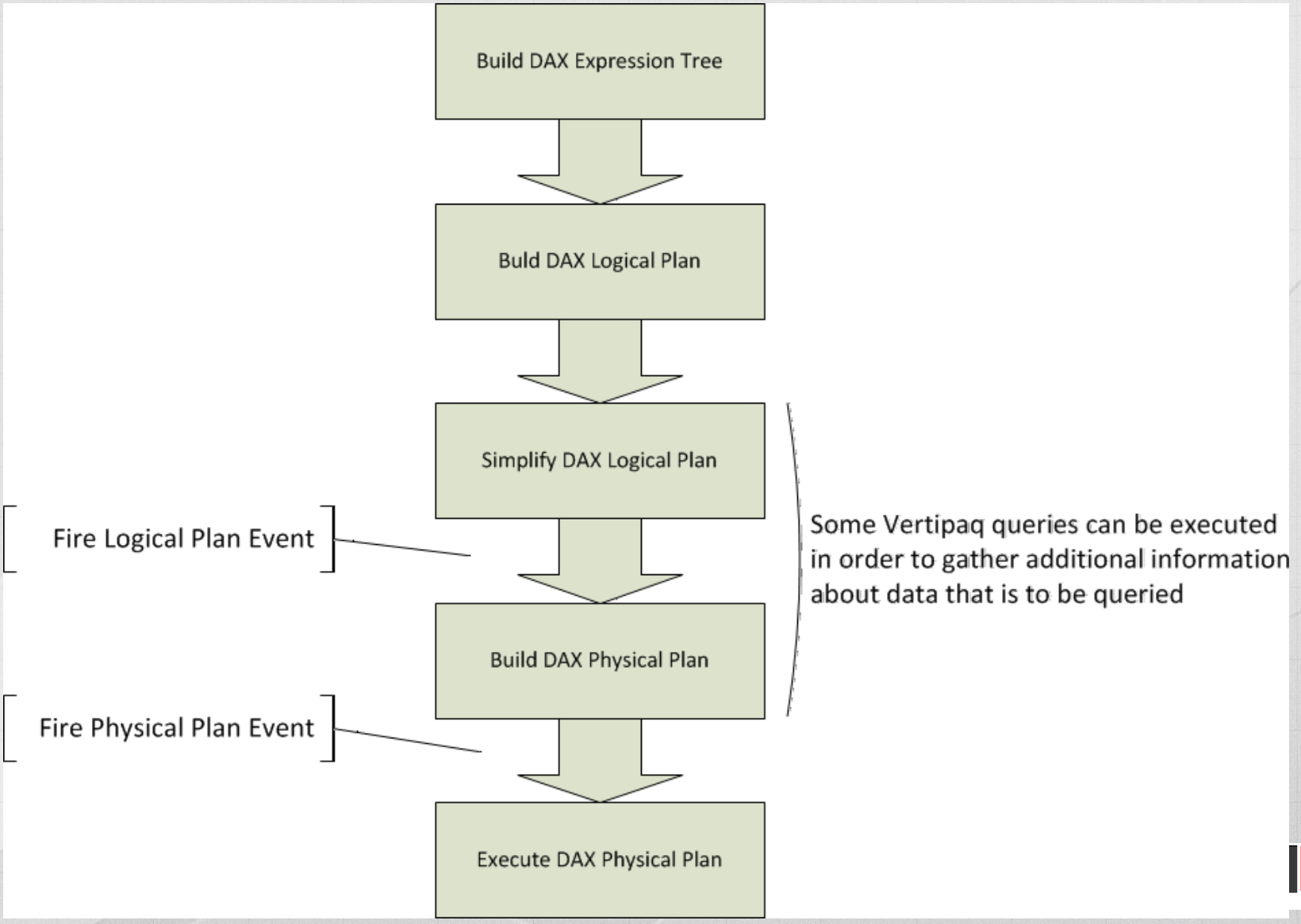
- Optimizing DAX means:
  - Reduce FE usage
  - Increase SE usage
- At the end, it is very easy, we only need to understand who processes what ☺
  - What is computed in FE?
  - And what is computed in SE?
  - How to move computation in SE?
- Time to dive deeper in the DAX engine

sqlbi

Understanding Query Plans

- Logical Query Plan
  - It is the logical flow of the query
  - Fired as standard text
  - Pretty hard to decode
- Physical Query Plan
  - The logical query plan executed by the engine
  - Can be very different from the logical one
  - Uses different operators
- VertiPaq Queries
  - Queries executed by the xVelocity engine

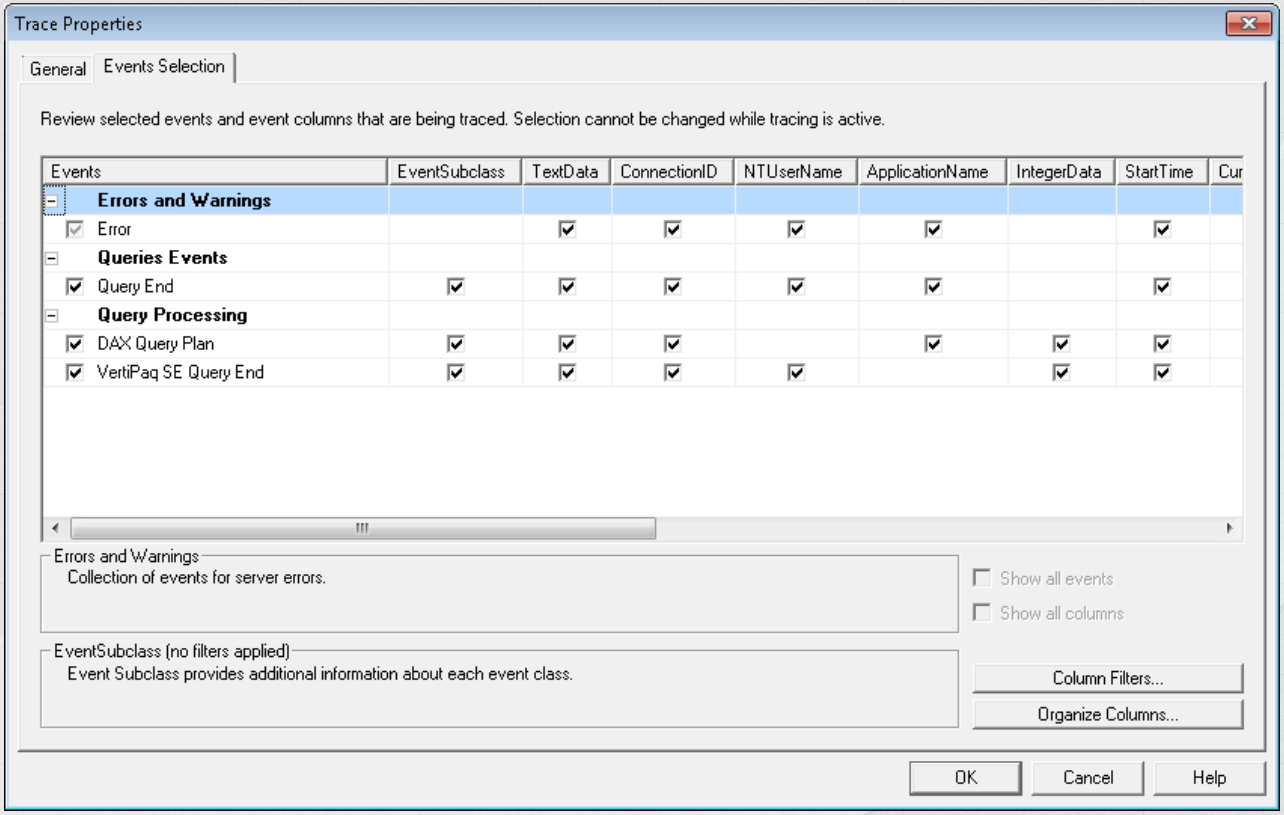
DAX Query Optimization Flow



DAX is not cost-based

- Unlike SQL Server
- Data is gathered to analyze
  - When to perform filtering
  - How to resolve joins
  - Materialization needs
- But the query plan does not change with different row counts

SQL Server Profiler



- Catches events from SSAS
  - Queries Events
  - Query Processing

sqlbi

Monitoring a Query

Trace of a simple query

```
EVALUATE

CALCULATETABLE(
    SUMMARIZE(
        'Internet Sales',
        Geography[State Province Code],
        "Sales", SUM( 'Internet Sales'[Sales Amount] )
    ),
    FILTER(
        Customer,
        Customer[Last Name] = "Anand"
    )
)
```

sqlbi

1° VertiPaq Query

Note the usage of DATAID to filter «Anand»

```
SELECT
    [Customer].[CustomerKey]
FROM
    [Customer]
WHERE
   ( PFDATAID( [Customer].[LastName] ) = 81 )
```

sqlbi

2° VertiPaq Query

Query using JOINS inside xVelocity. These joins do not require materialization and can be executed very fast in parallel

```
SELECT
    [Geography].[StateProvinceCode]
FROM [Internet Sales]
    LEFT OUTER JOIN [Customer]
         ON [Internet Sales].[CustomerKey] = [Customer].[CustomerKey]
    LEFT OUTER JOIN [Geography]
         ON [Customer].[GeographyKey] = [Geography].[GeographyKey]
WHERE
    [Customer].[CustomerKey] IN
      (11096, 11989, 17005, 22513, 28899, 15054,
       19626, 20344, 25918, 27141...
      [74 total values, not all displayed]);
```

sqlbi

DAX Query Plan

Simplifed text of the query plan, gives a good idea of what the FE is going to execute. In red the parts executed by xVelocity

```
CalculateTable
    AddColumns
        Scan_VertiPaq
        GroupBy_VertiPaq
            Scan_VertiPaq
        Sum_VertiPaq
            Scan_VertiPaq
           'Internet Sales'[Sales Amount]
    Filter_VertiPaq
        Scan_VertiPaq
        'Customer'[Last Name] = Anand
```

sqlbi

Query Plans side by side

Simplifed text of the query plan, gives a good idea of what the FE is going to execute. In red the parts executed by xVelocity

```
CalculateTable
    AddColumns
        Scan_VertiPaq
        GroupBy_VertiPaq
            Scan_VertiPaq
        Sum_VertiPaq
            Scan_VertiPaq
            [Sales Amount]
    Filter_VertiPaq
        Scan_VertiPaq
        'Customer'[Last Name] = Anand
```

```
EVALUATE
CALCULATETABLE(
    SUMMARIZE(
        'Internet Sales',
        Geography[State Province Code],
        "Sales", SUM( [Sales Amount] )
    ),
    FILTER(
        Customer,
        Customer[Last Name] = "Anand"
    )
)
```

sqlbi

Query Running: Step 1

This query is the first one used to really execute the DAX query.
Note that SUM is executed inside xVelocity, not in the Formula Engine

```
SELECT
    [Geography].[StateProvinceCode],
    SUM([Internet Sales].[SalesAmount])
FROM
    [Internet Sales]
    LEFT OUTER JOIN [Customer]
        ON [Internet Sales].[CustomerKey]=[Customer].[CustomerKey]
    LEFT OUTER JOIN [Geography]
        ON [Customer].[GeographyKey]=[Geography].[GeographyKey]
WHERE
    [Customer].[CustomerKey] IN
        (11096, 11989, ...[74 total values, not all displayed]) VAND
    [Geography].[StateProvinceCode] IN
        ('VIC', 'BC', ...[21 total values, not all displayed]);
```

Query Running: Step 1

This query is identical to the second one computed during optimization and will hit the cache

```
SELECT
    [Geography].[StateProvinceCode]
FROM [Internet Sales]
    LEFT OUTER JOIN [Customer]
         ON [Internet Sales].[CustomerKey] = [Customer].[CustomerKey]
    LEFT OUTER JOIN [Geography]
         ON [Customer].[GeographyKey] = [Geography].[GeographyKey]
WHERE
    [Customer].[CustomerKey] IN
      (11096, 11989, 17005, 22513, 28899, 15054,
       19626, 20344, 25918, 27141...
      [74 total values, not all displayed]);
```

Simple Query Plan

- 4 VertiPaq Queries
  - 2 before execution
  - 2 during execution
    - 1 performed GROUPBY and SUM in VertiPaq
    - 1 hit the cache
- Intermediate results are spooled
- Final JOIN performed by Formula Engine
- This is a very good query plan

Is SUMX the Evil?

This SUMX is resolved inside VertiPaq because it is a simple operation that VertiPaq know how to handle. No iteration happens here.

```
EVALUATE
    ROW (
        "Sum",
        SUMX(
            'Internet Sales',
            'Internet Sales'[Sales Amount] / 'Internet Sales'[Order Quantity] )
    )
```

```
SELECT
    SUM( [Internet Sales].[SalesAmount] / [Internet Sales].[OrderQuantity] )
FROM
    [Internet Sales];
```

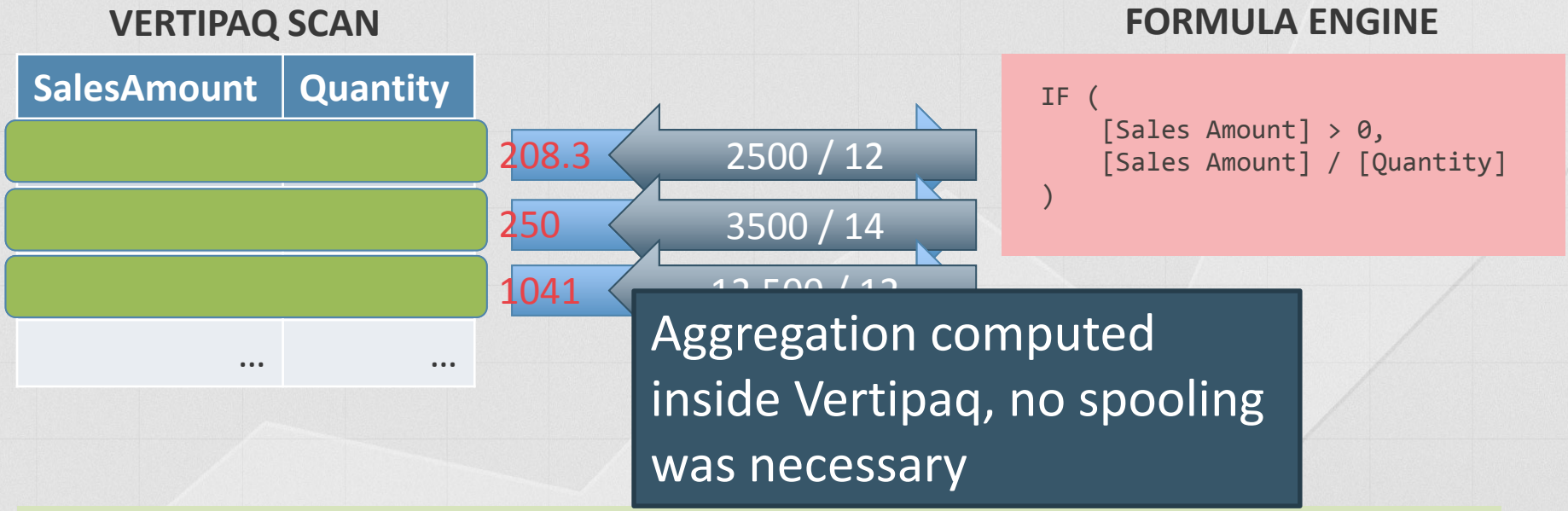
Is SUMX the Evil?

If the expression is too complex, CallbackDataID appears, meaning a call back to formula engine during the VertiPaq scan.

```
EVALUATE
    ROW (
        "Sum",
        SUMX (
            'Internet Sales',
            IF (
                'Internet Sales'[Sales Amount] > 0,
                'Internet Sales'[Sales Amount] / 'Internet Sales'[Order Quantity]
            )
        )
    )
```

```
SELECT
    SUM(
        [CallbackDataID(
            IF (
                'Internet Sales'[Sales Amount] > 0,
                'Internet Sales'[Sales Amount] / 'Internet Sales'[Order Quantity]]
            )
        )]
        (
            PFDATAID( [Internet Sales].[OrderQuantity] ),
            PFDATAID( [Internet Sales].[SalesAmount] )
        )
    )
FROM [Internet Sales];
```

CallBackDataID in Action




```
SELECT
    SUM(
        [CallbackDataID(
            IF (
                'Internet Sales'[Sales Amount]] > 0,
                'Internet Sales'[Sales Amount]] / 'Internet Sales'[Order Quantity]]
            )
        )]
        (
            PFDATAID( [Internet Sales].[OrderQuantity] ),
            PFDATAID( [Internet Sales].[SalesAmount] )
        )
    )
FROM [Internet Sales];
```

CallBackDataID Performance

- Slower than pure Vertipaq
- Faster than pure Formula Engine
  - Highly parallelized
  - Works on compressed data
- Does not require materialization
  - No spooling of temporary results
  - Less memory used
- Materializes a single row

SQL Server Profiler

- There are other useful info
  - CPU Time
  - Duration
  - Event Subclass
- CPU Time >= Duration
  - Many cores run in parallel
  - Only during VertiPaq scans
  - Formula Engine is still single-threaded

Monitor MDX and DAX

- MDX Queries
  - Generate DAX Query Plans
  - Not translated to DAX
- SQL Queries
  - Useful if you cannot monitor SQL Server
  - Optimize SQL to optimize SSAS

Clear the Cache

Always remember to clear the cache prior to execute any performance test, otherwise numbers will be contaminated

```
<Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
   <ClearCache>
      <Object>
         <DatabaseID>Adventure Works DW Tabular</DatabaseID>
      </Object>
   </ClearCache>
</Batch>
```

sqlbi

Currency Conversion

Two SUMX are converted to two iterations

```
[FirstCurrencyAmount] :=

SUMX(
    DimCurrency,
    SUMX(
        DimDate,
        CALCULATE(
            VALUES( CurrencyRate[AverageRate] )
             * SUM( FactInternetSales[SalesAmount] )
        )
    )
)
```

sqlbi

Currency Conversion

Simple SUMX over CrossJoin is resolved as a single VertiPaq scan with CallbackDataID to compute the multiplication.

```
[SecondCurrencyAmount] :=

SUMX(
    CROSSJOIN(
        DimCurrency,
        DimDate
    ),
    CALCULATE(
       VALUES( CurrencyRate[AverageRate] )
         * SUM( FactInternetSales[SalesAmount] )
    )
)
```

sqlbi

Filter as Soon as You Can

This query computes YTD and QTD inside a loop and then removes empty values.

```
DEFINE
    MEASURE 'Internet Sales'[Sales] =
        CALCULATE( ROUND( SUM( 'Internet Sales'[Sales Amount] ), 0 ) )
    MEASURE 'Internet Sales'[YTD Sales] =
        TOTALYTD( [Sales] , 'Date'[Date] )
    MEASURE 'Internet Sales'[QTD Sales] =
        TOTALQTD( [Sales] , 'Date'[Date] )
EVALUATE

FILTER(
    ADDCOLUMNS(
        CROSSJOIN(
            VALUES( 'Date'[Calendar Year] ),
            VALUES( 'Date'[Month] ),
            VALUES( 'Date'[Month Name] )
        ),
        "Sales", [Sales],
        "YTD Sales", [YTD Sales],
        "QTD Sales", [QTD Sales]
    ),
    NOT ISBLANK( [Sales] )
)
ORDER BY 'Date'[Calendar Year], 'Date'[Month]
```

sqlbi

Filter as Soon as You Can

This second query removes empty rows before computing expensive measures. It runs 7 times faster even if it computes [Sales] twice.

```
DEFINE
    MEASURE 'Internet Sales'[Sales] =
        CALCULATE( ROUND( SUM( 'Internet Sales'[Sales Amount] ), 0 ) )
    MEASURE 'Internet Sales'[YTD Sales] =
        TOTALYTD( [Sales] , 'Date'[Date] )
    MEASURE 'Internet Sales'[QTD Sales] =
        TOTALQTD( [Sales] , 'Date'[Date] )
EVALUATE

ADDCOLUMNS(
    FILTER(
        CROSSJOIN(
            VALUES( 'Date'[Calendar Year] ),
            VALUES( 'Date'[Month] ),
            VALUES( 'Date'[Month Name] )
        ),
        NOT ISBLANK( [Sales] )
    ),
    "Sales", [Sales],
    "YTD Sales", [YTD Sales],
    "QTD Sales", [QTD Sales]
)
ORDER BY 'Date'[Calendar Year], 'Date'[Month]
```

sqlbi

Use Relationships

Relationships can be pushed down to VertiPaq, CALCULATE conditions cannot

```
EVALUATE
ADDCOLUMNS(
    DimProduct,
    "SumOfSales", CALCULATE(
        SUM( FactInternetSales[SalesAmount] ),
        USERELATIONSHIP( DimProduct[ProductKey], FactInternetSales[ProductKey] )
    )
)
```

```
EVALUATE
ADDCOLUMNS (
    DimProduct,
    "SumOfSales", CALCULATE(
        SUM( FactInternetSales[SalesAmount] ),
        FILTER(
            FactInternetSales,
            DimProduct[ProductKey] = CALCULATE( VALUES( FactInternetSales[ProductKey] ) )
        )
    )
)
```

Use Relationships

The first VertiPaq query resolves the JOIN, the second one is a simple scan and the JOIN will be performed inside Formula Engine

```
SELECT
    DimProduct.ProductKey,
    ...
    SUM (FactInternetSales.SalesAmount)
FROM FactInternetSales
     LEFT OUTER JOIN DimProduct
    ON FactInternetSales.ProductKey = DimProduct.ProductKey
WHERE
    ...
```

```
SELECT
    FactInternetSales.ProductKey,
    SUM (FactInternetSales.SalesAmount)
FROM FactInternetSales
WHERE
    ...
```

Optimizations: Conclusions

- Optimizing DAX is not a black art
  - Like it was with MDX
- Query plan greatly help understanding what is happening
- Optimizing DAX is your only choice to improve the database responsiveness
- We scratched the surface… experience is the key of being effective

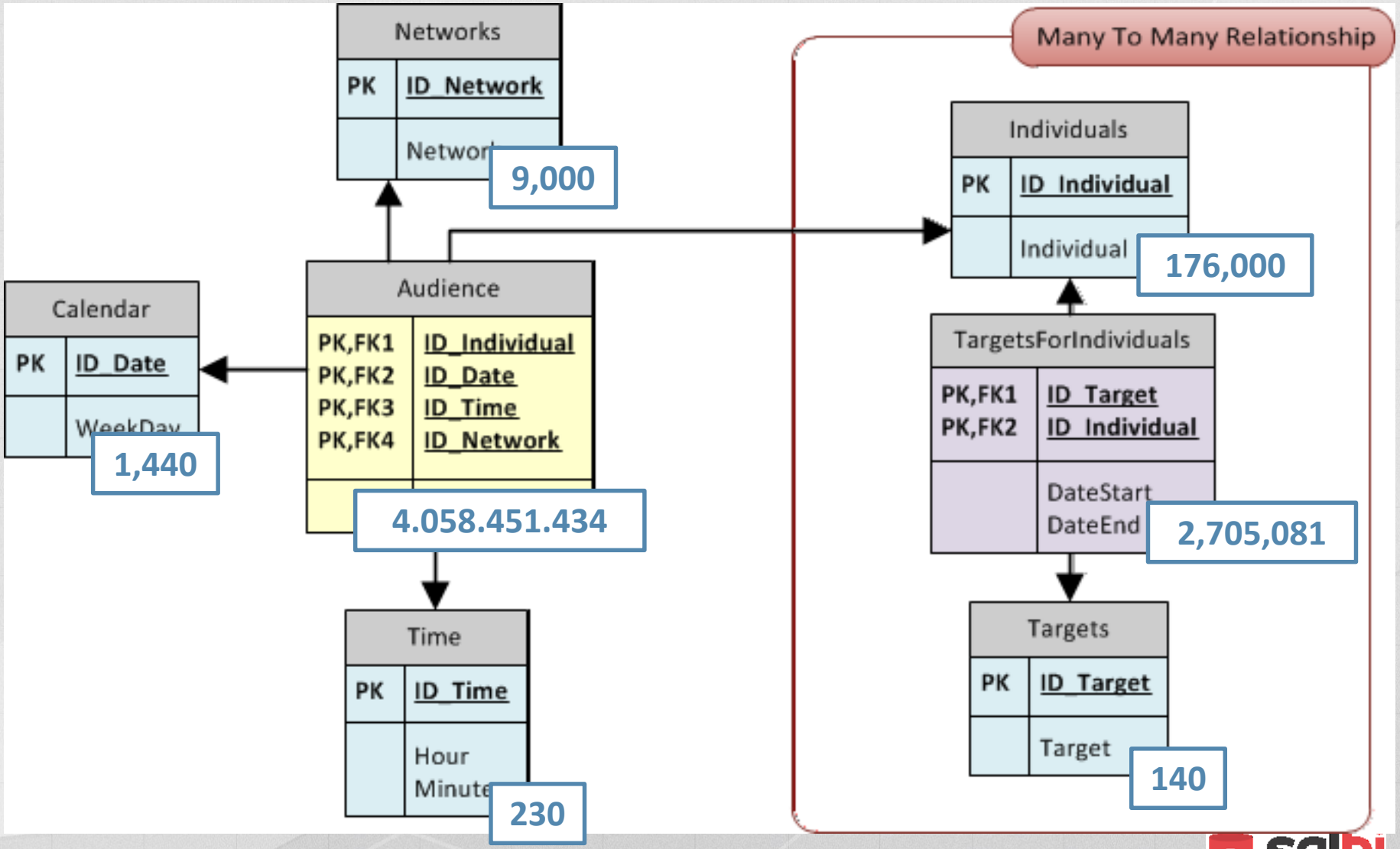
sqlbi

General Rules

- Use the simplest formulas
  - Prefer ADDCOLUMNS to SUMMARIZE
  - Avoid complex calculations in measures
- Don't use error-handling functions extensively
  - They are slooooooow
  - Good only in measures
- Build a correct data model
  - Reducing distinct count of values for columns is the main target
  - Keep it simple, relational, clear
- Push calculations down to the VertiPaq engine
  - Not an easy task to do looking at query plans
  - But, after some experience, you will learn it

sqlbi

The Test Data Model



sqlBI

Top Three Fears for UDM Developers

- DISTINCT COUNT
  - «Do you REALLY need DISTINCT COUNT of Customer»?
- Many-To-Many Relationships
  - «Well… I can try to optimize them in some way»
  - «Let me search that whitepaper…»
- Leaf-Level Calculations
  - «It would be better to compute this during ETL»
  - «Oh… I see… do you need it dynamic? … »

Conclusions

- DAX is a simple language
  - Effective in expressing complex queries
  - Looks strange, but easier than MDX
- Optimizing Tabular means optimizing DAX
  - Pushing calculations to VertiPaq
  - Trust the Rain Man, he's your best friend!
  - Sometimes, need to change the data model
- **DAX is simple, it is not easy**... but this is the fun part of it!

Coming up…

Speaker	Title	Room
Alex Whittles	Data Modeling for Analysis Services Cubes	Theatre
Allen White	Maintain SQL Server System and Performance with PowerShell	Exhibition B
Hugo Kornelis	Everything you always wanted to know about MERGE	Suite 3
Stephan Stoltze	Excel 2013 - Whats new beside PowerPivot and Power View?	Suite 1
Christian Bolton	Advanced SQL Server 2012 HA and DR Architectures	Suite 2
Niko Neugebauer	Dynamic MSBI content generation	Suite 4

twitter #SQLBITS

Consulting

Assessment

Outsourcing

Technical Fellowship

Find out more on

www.sqlbi.com/consulting