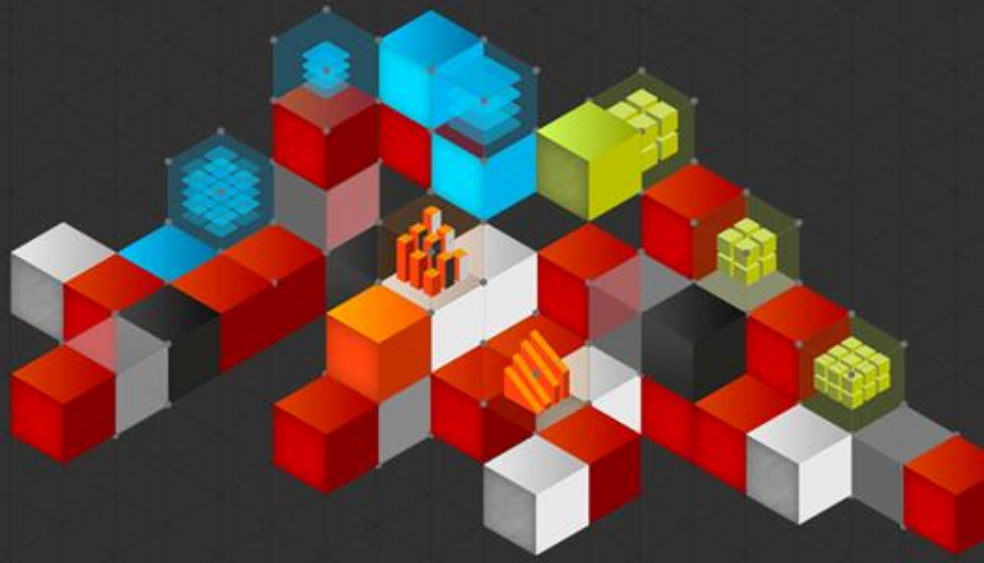
SQL SERVER 2012 XVELOCITY COLUMNSTORE INDEX

Conor Cunningham
Principal Architect
SQL Server Engine

Microsoft® SQL Server® 2012



TODAY'S TALK

- Today we're going to look "under the hood" on one of the key new features in SQL Server 2012
- We will focus on ColumnStore indexes and how they are implemented
- For this talk, I am going to cover:
  - ***How*** SQL Server executes queries using this index
  - ***Why*** this functionality speeds performance

- Note: There are other talks on this index at SQLBits – they are all useful and focus on different parts of the technology

DATA WAREHOUSES INTRODUCTION

- What is a Data Warehouse?
  - Data Warehouses are databases constructed in a special way
  - They support reporting and business intelligence operations in organizations
  - They are often BIG – lots of data
  - Their size forces some tradeoffs and special use patterns
- Example: Show me the sales totals for each department in my company by month for the past 3 years

STAR SCHEMAS

Raw Data

SaleDate	SalesPerson	ItemName	SaleAmt
2012-2-3	Bob	Paper	12.34
2012-2-4	Sally	Paper	25.00
2012-2-4	Bob	Toner	65.23

FactID	DateID	SalesPID	ItemID	Amt
1	1	1	1	12.34
2	2	2	1	25.00
3	2	1	2	65.23

"Fact" Table

"Dimension" Tables

ItemID	Name
1	Paper
2	Toner

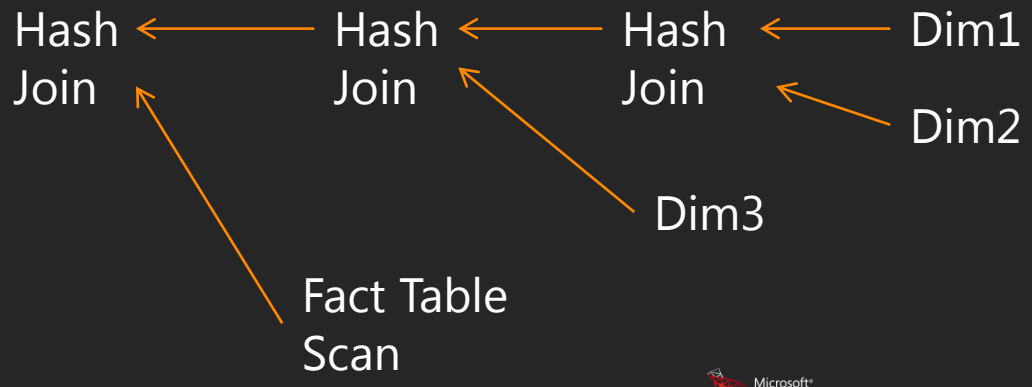
SalesPID	Name
1	Bob
2	Sally

DateId	Date
1	2012-2-3
2	2012-2-4

STAR JOIN PLANS

- Fact Tables are often BIG
  - So big that indexing is often impractical
  - You only want to scan it once per query
- SQL Server builds special query plans for aggregate queries over star schemas
  - These are “Star Join” Plans

1. Hash Joins
2. Dimensions First, then Scan Fact Table
3. Often Parallel



DATA WAREHOUSE PERFORMANCE

- DW Perf originally limited by IO Scan Speed
- Over time, IO speed got faster
  - Many disks, faster interfaces
- Eventually DW Perf was CPU bound
  - Even with Parallel Query, eventually you got stuck on the performance of the CPU
- Example – An analyst runs a BI report against SQL (ROLAP mode)
  - Load Report
  - Click on report to drill down
  - Go get some coffee
  - Click again
  - Go get more coffee

WHAT IF…

- …We could make those report queries run ten times faster?
- …We changed the user paradigm from batch processing to an interactive experience?

- Businesses could navigate their data much more easily
- Everyone wins
  - (…except perhaps coffee shops)

DEMO

HOW DOES IT WORK?

- Making something 10x faster is HARD
- Imagine making a car or a plane 10x faster
  - Bugatti Veyron – 252 mph
  - SR-71 Blackbird – 2070 mph
- If you DO happen to make it 10x faster, you're likely not called a "car" or a "plane" anymore

- To build the xVelocity technology in ColumnStore, we had to do start over

HOW TO SPEED UP STORAGE AND IO

- How do we cut down IO by enough to get 10x overall speedup?
- Step 1: Column-Orientation
  - Store data vertically instead of per-row
  - Exploit that most Star Join Queries touch only some of the columns
- Step 2: Dictionaries for variable-length data
- Step 3: Compress Repeated Values
  - Segment data into groups (1 million rows/group)
  - Star Schemas have lots of repeated values
- Space savings of 1.5x-2x vs. a row-based page-compressed equivalent

IO Patterns for
(CI Scan, Column-based scan of 3 cols, Column-based w/Compression)

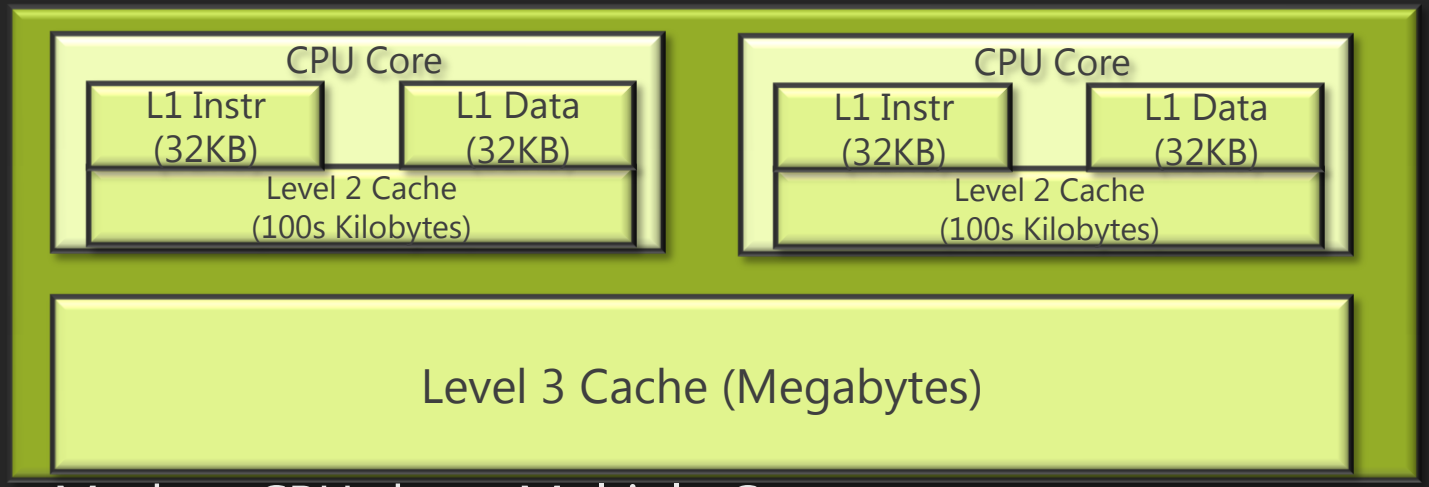
WHAT ELSE CAN WE DO?

- New systems are all 64-bit
- When compressed, sometimes (parts of) fact tables are starting to get small enough to fit into memory
- Memory gets exponentially cheaper over time
- So buy enough to fit that compressed fact table in memory
- We biased the memory management code to keep more data in memory in this scenario
- Now we can get the data into our star join potentially with no IO…

IMPROVING QUERY EXECUTION

- How do you make any program go 10x faster?
- Cut out 90% of the instructions?
  - Unfortunately not…
- Buy 10x the number of processors?
  - Maybe not…
- Throw out the book?

AN ASIDE…HOW CPUS WORK



- Modern CPUs have Multiple Cores
- Cache Hierarchies: L1, L2, L3
  - Small L1 and L2 per core; L3 shared by all cores on die
  - L1 is faster than L2, L2 faster than L3
  - CPUs can stall waiting for caches to load

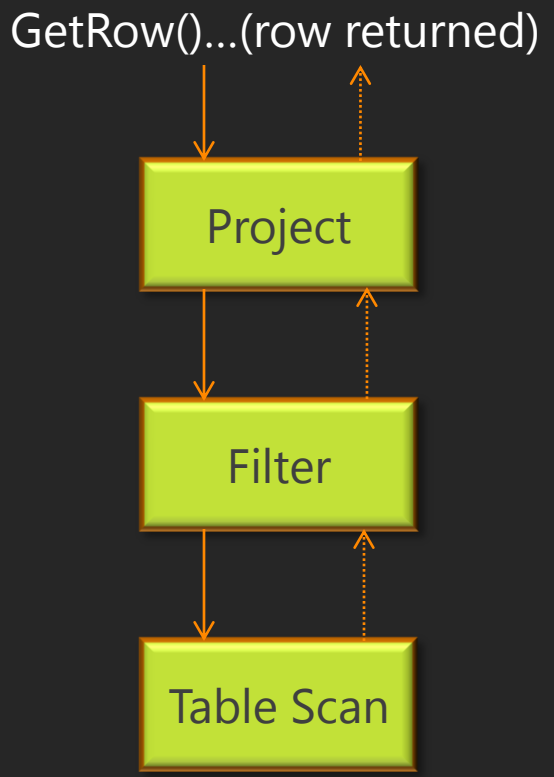
Time to Access Increases each level you need to touch!

KEY LESSONS

- Trend: CPUs speeds increase faster than memory
  - Cost of going to next cache level goes up each year
- Code and Data both cause cache misses
  - How you touch memory impacts program speed
- So we need to write programs to:
  - Use fewer instructions
  - Avoid cache misses
- We implemented a new execution model called "Batch Mode"
  - The existing model is called "Row Mode"

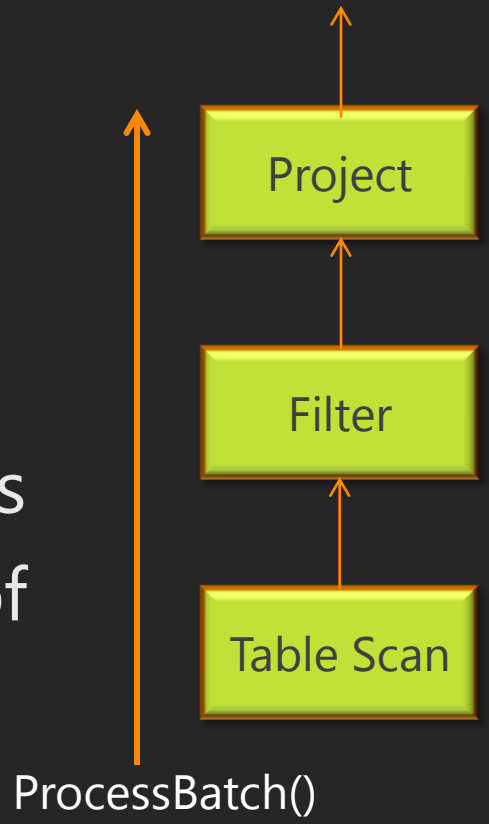
HOW ROW MODE WORKS

- Each operator calls child for each row to "pull" the next row
- Works fine for smaller queries
- Often each operator transition causes L2 cache misses to load instructions/data
- When databases were new, the cost of IO was MUCH larger than CPU speed and this never mattered
- Now the equation has changed



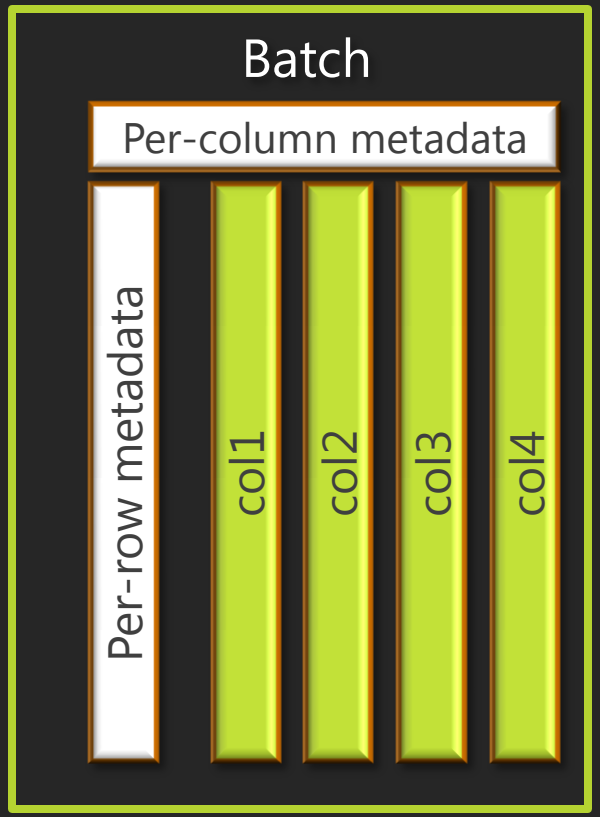
BATCH MODEL

- Move from “pull” model to “push”
- Group rows into batches
  - Re-use instructions while in cache
  - Touch all “close” data in each operator
- This model reduces L2 cache misses
- It works best for queries with lots of rows being processed



BATCHES IN DETAIL

- Column-Oriented
- Sized to fit within L2 cache
- Reduce average per-tuple instruction cost
  - Run-Length Compression Per Column
  - Probabilistic data representations (in-row vs. out-of-row)
  - Probabilistic operator execution algorithms (in-row first)
- This gets us to 10x+ faster (average)



PERFORMANCE

- New model makes queries go a LOT faster
- However, not all queries get the benefits
  - Data types: core types <=64 bits (CPU register size)
  - Operators: Star Join Pattern (Join, Group By, Filter)
- Some restrictions we will relax over time
- If your application follows the star join pattern, you can get great speedups

CONCLUSION

- Please try out the new xVelocity ColumnStore Index
- It can give you significant performance improvements over prior technologies

- Thank You for Attending!