SQL Server

Data Loading Performance Guide

Joyful Craftsmen

Business Intelligence Crafted to Joy

BIO

- SQL Server Support Engineer @Joyful Craftsmen
- Former Lead SQL Server DBA from Česká spořitelna
- Czech PASS Leader & SQL Saturday co-organizer
- Father of two

@malekpav

malekpav@gmail.com

https://joyfulcraftsmen.com/

Czech PASS

SQL Saturday Prague #889

CZECH PASS CHAPTER

meetup.com/**czechpass**

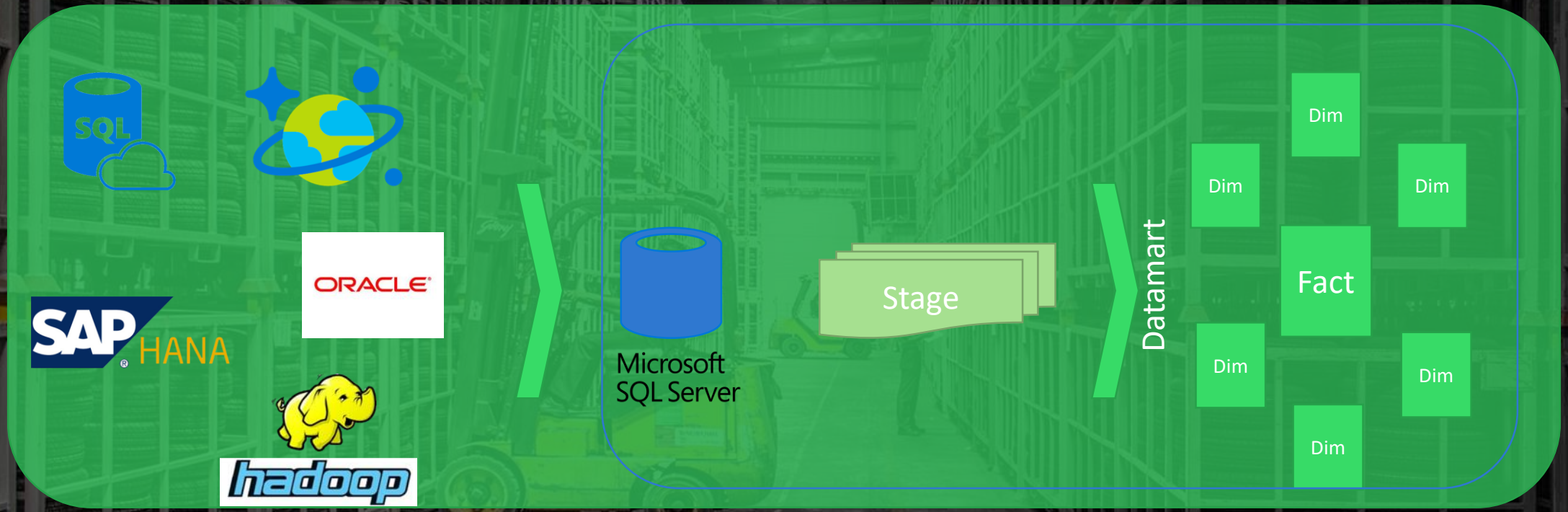
czechpass.slack.com

community.powerbi.com

Thanks to the sponsors and organizers!

Typical story – Datawarehouse Load

- We insert huge amount of data
- Performance at stage/dm level is not always ideal



Bonus Material

Other story – Application Inserts

- Many users loads same data into same data structure

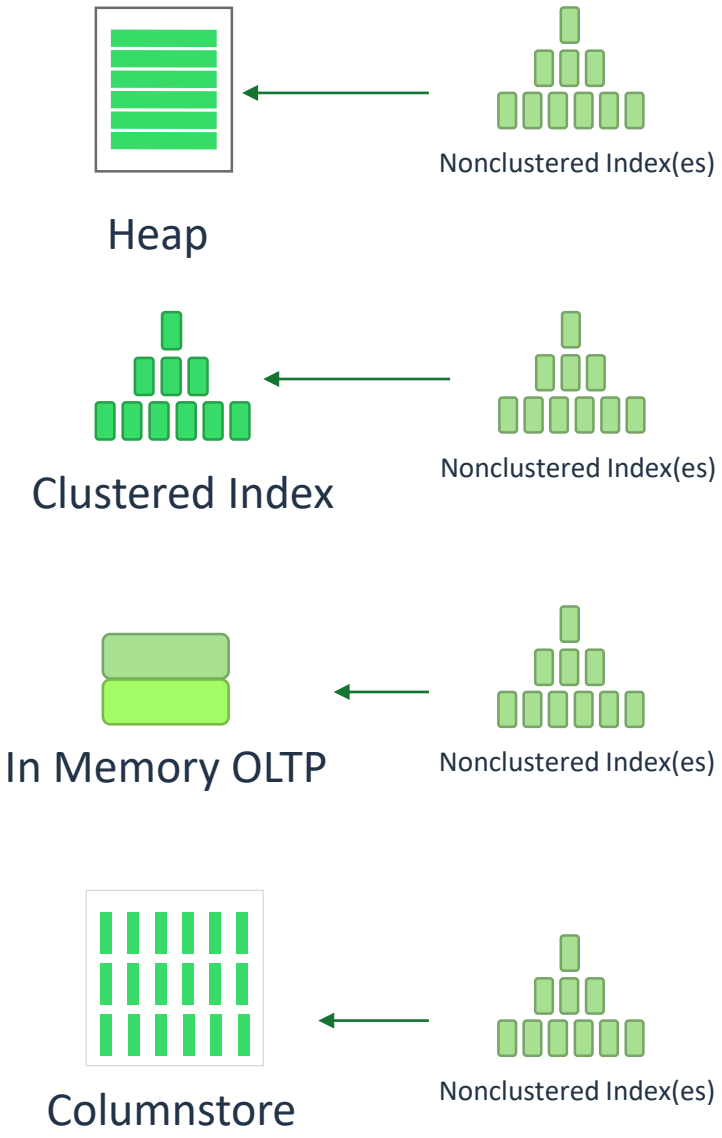
What, where & how

- What data we want to load
  - Source
- Where we want to load them
  - Process
- What tool are we going to use
  - Landing area

Landing Area

- Traditinal row store
  - Heap
  - Clustered Index
- Untraditional Store
  - Memory optimized tables
  - Columnstore

- With(out) other non clustered indexes
- Partitioned
- Compressed
- Empty
- Durable, delayed durable or just schema durable
- Constraints
- Identity / Sequences

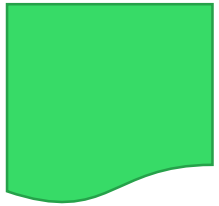


Process

- INSERT SELECT
- SELECT INTO
- bcp
- SSIS
- BULK INSERT

Source

Large Datasets

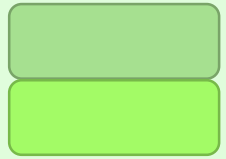


CSV

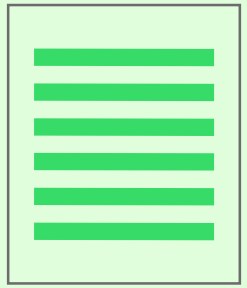
Data in tables

Stream of data

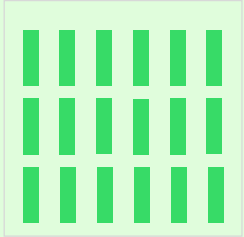
Many small batches



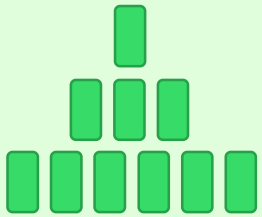
In Memory OLTP



Heap

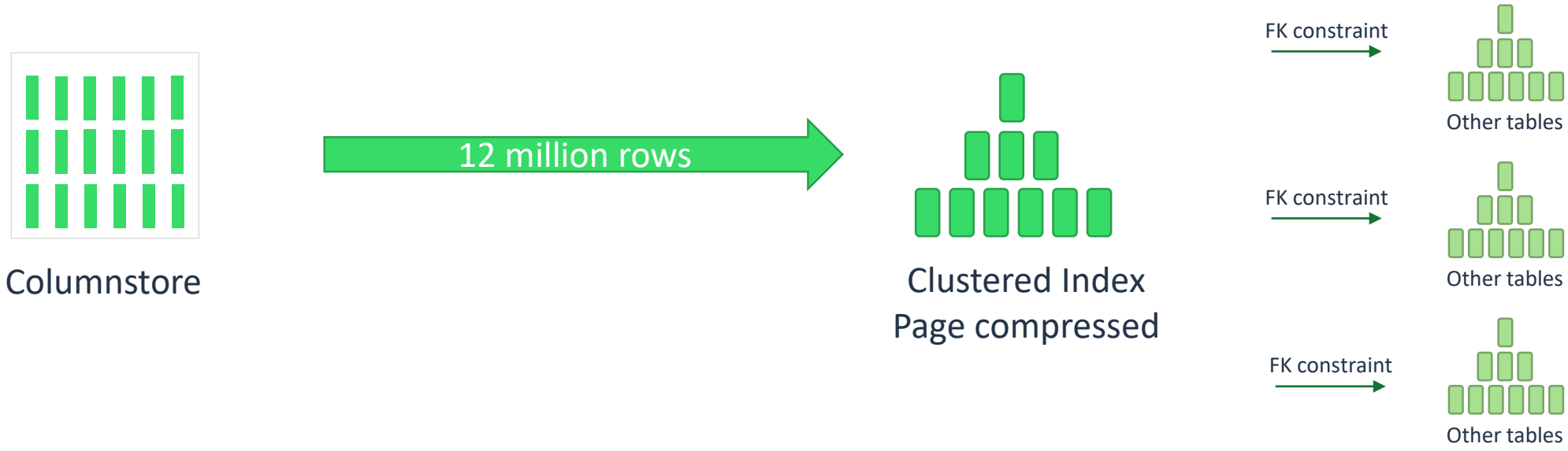


Columnstore



B-Tree Indexes

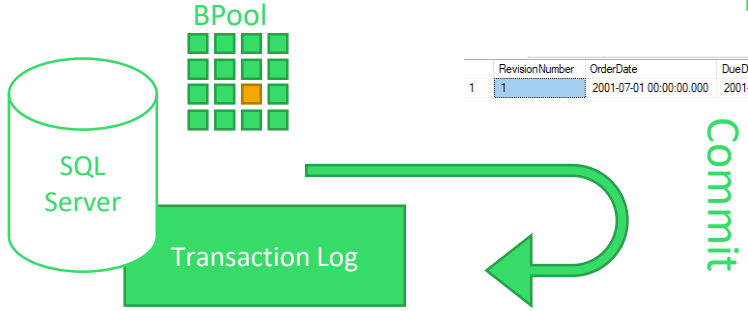
Baseline – CCI to traditional Row Store Fact



Bit of a theory – Write Ahead Logging

Phase 1 - Sync

```
INSERT Sales.SalesOrderHeader
(
    RevisionNumber,
    OrderDate,
    ...
    ModifiedDate
)
VALUES
(   1, GETDATE(), ..., GETDATE())
```

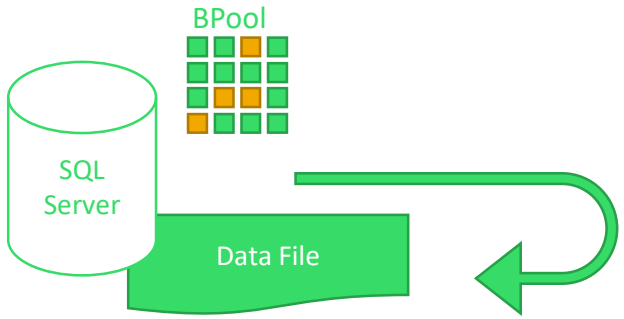


Full uncompressed* record

	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	PurchaseOrderNumber
1	1	2001-07-01 00:00:00.000	2001-07-13 00:00:00.000	2001-07-08 00:00:00.000	5	0	PO522145787

Commit

Phase 2 - Async



Checkpoint

How to speed it up?

Don’t log full records

- Make your inserts minimally logged

Run it in parallel

- Stop using parallelism inhibitors

Don’t wait for commit

- Delayed Durability
- Schema durable data only

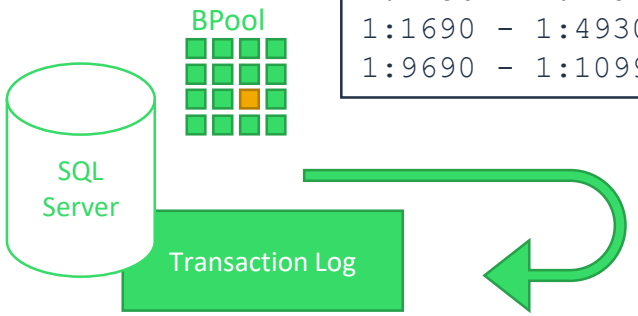
Don’t log full records

Phase 1a - Sync

What pages have been changed

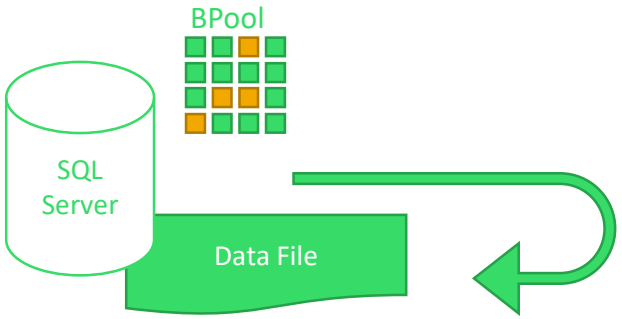
```
1:1236 - 1:1290
1:1690 - 1:4930
1:9690 - 1:10996
```

```
INSERT Sales.SalesOrderHeader
(
    RevisionNumber,
    OrderDate,
    ...
    ModifiedDate
)
VALUES
(   1, GETDATE(), ..., GETDATE()),
...
```



Commit

Phase 1b - Sync

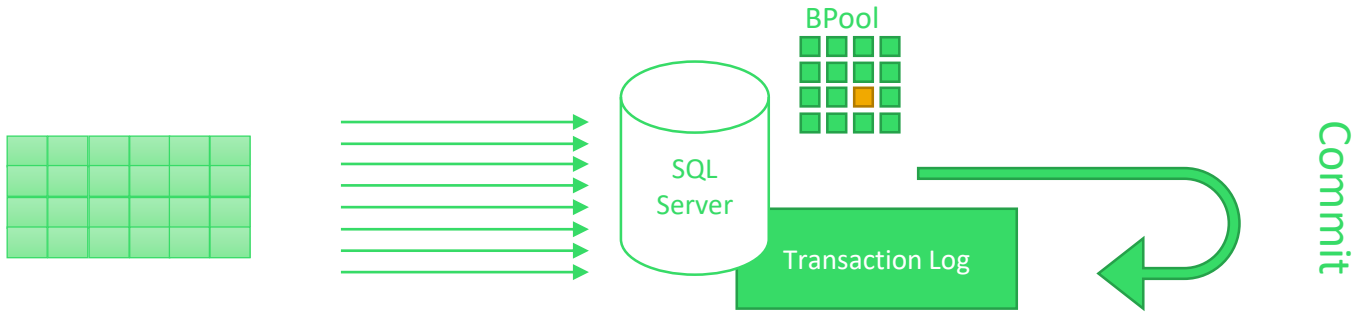


Don’t log full records - tradeoffs

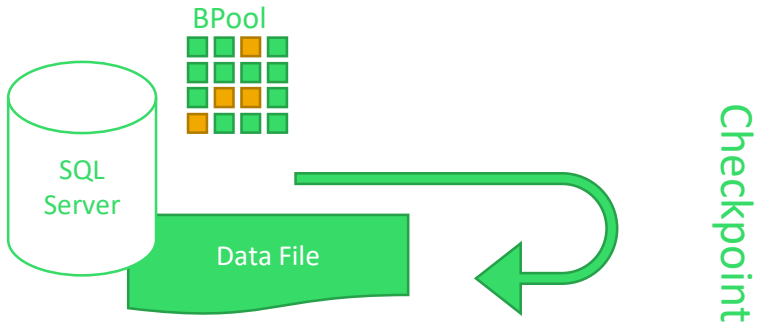
- Simple or bulk logged recovery model
  - No Point in Time Restore
  - No Always On Availability Groups or Database Mirroring
  - No Transactional Replication

- Usually acceptable for data analytics databases

Run it in parallel or concurrently



Phase 2 - Async



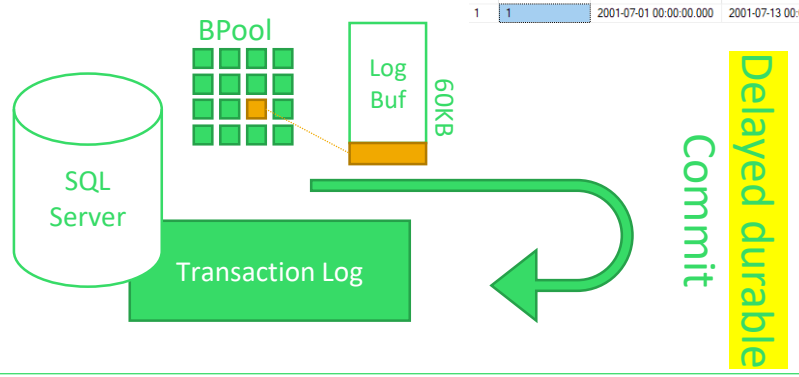
Don’t wait for commit #1

Phase 1 - Async

Full uncompressed record

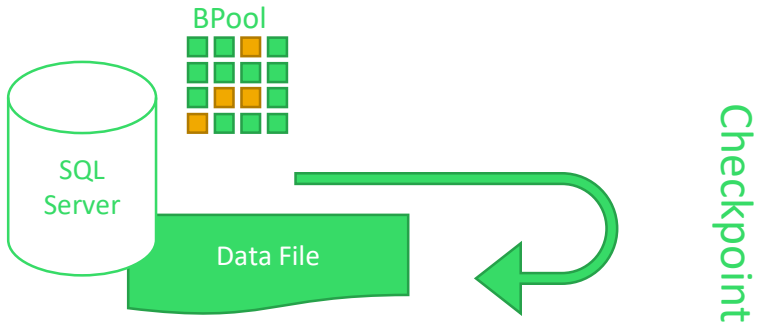
	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	PurchaseOrderNumber
1	1	2001-07-01 00:00:00.000	2001-07-13 00:00:00.000	2001-07-08 00:00:00.000	5	0	PO522145787

```
INSERT Sales.SalesOrderHeader
(
    RevisionNumber,
    OrderDate,
    ...
    ModifiedDate
)
VALUES
(   1, GETDATE(), ..., GETDATE())
```



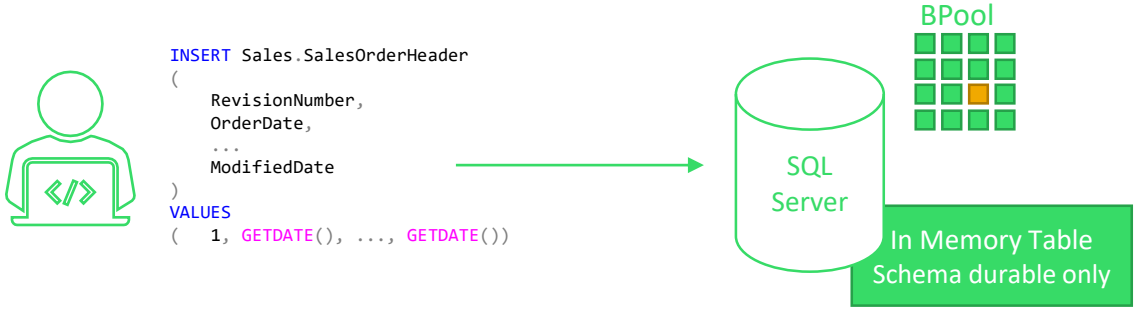
For small transactions only!

Phase 2 - Async



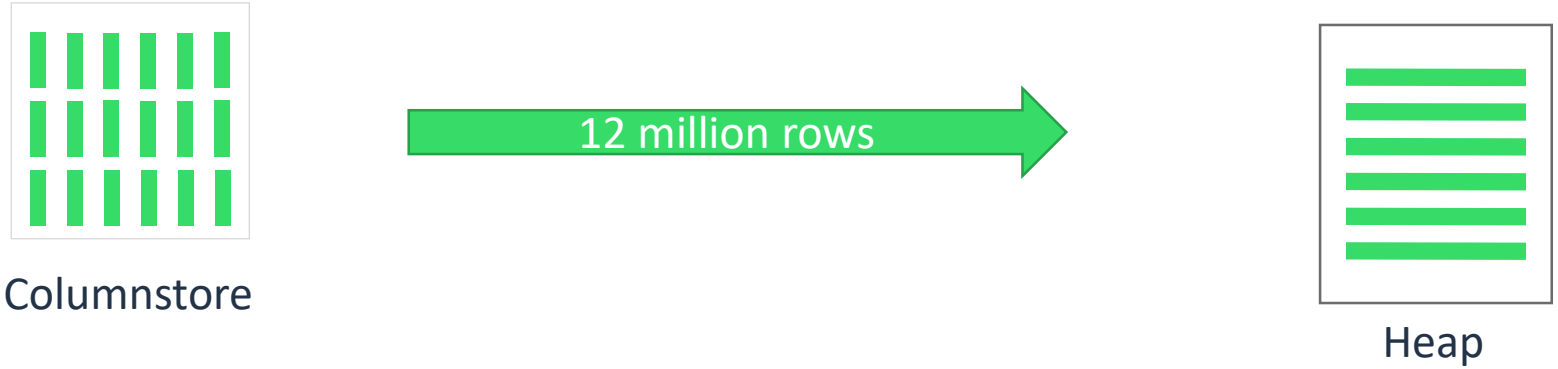
Don’t wait for commit #2

Phase 1 - Async



No Phase 2

Scenario 1 – Load into Heap



Some notes

- Concurrent load is possible and recommended but it is not working with INSERT SELECT
  - Concurrent vs parallel
- Use tab lock everywhere you can
  - TF715 (global || session)
  - sp_tableoption
- Use TF610 if you are on SQL Server 2014 or older

- SQL Server must expect **more than 250 rows** with a total size of **at least one extent** (8 pages).
  - It multiplies the estimated number of rows by a calculated maximum row size.
  - SQL Server **counts internal columns** present in the insert stream when computing the row size.

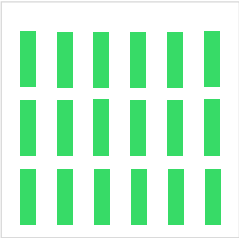
Src: https://sqlperformance.com/2019/05/sql-performance/minimal-logging-insert-select-heap

Scenario 2 – Load into CIX

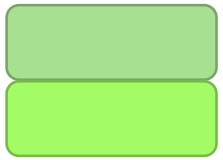
Load into CIX

- To achieve parallelism you need to go via heap
- Minimal logging is possible with or without TABLOCK but the performance is different
  - TABLOCK = RowSetBulk optimization
  - No TABLOCK = FastLoadContext optimization (for nonempty indexes)
- You can fake stats and get parallelism where you need
  - Not really recommended
- It matters if index(es) is full or not*
- Input order has severe performance impact

Scenario 3 – Load into In Memory OLTP



Columnstore

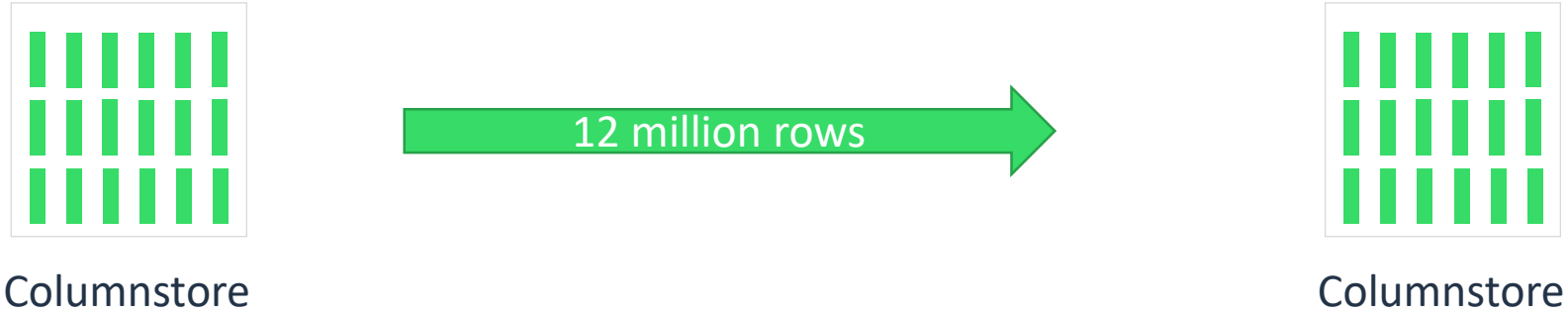


In Memory OLTP
durable / not durable

“BULK” Load into Memory Optimized Tables

- No parallelism
- No WAL but for bulk load data similar mechanism
  - Data hardened to TLOG at COMMIT
- Easily parallelized
- Indexes are more lightweight and hence easier to create
- Pretty good performance in certain scenarios

Scenario 4 – Load into CCI

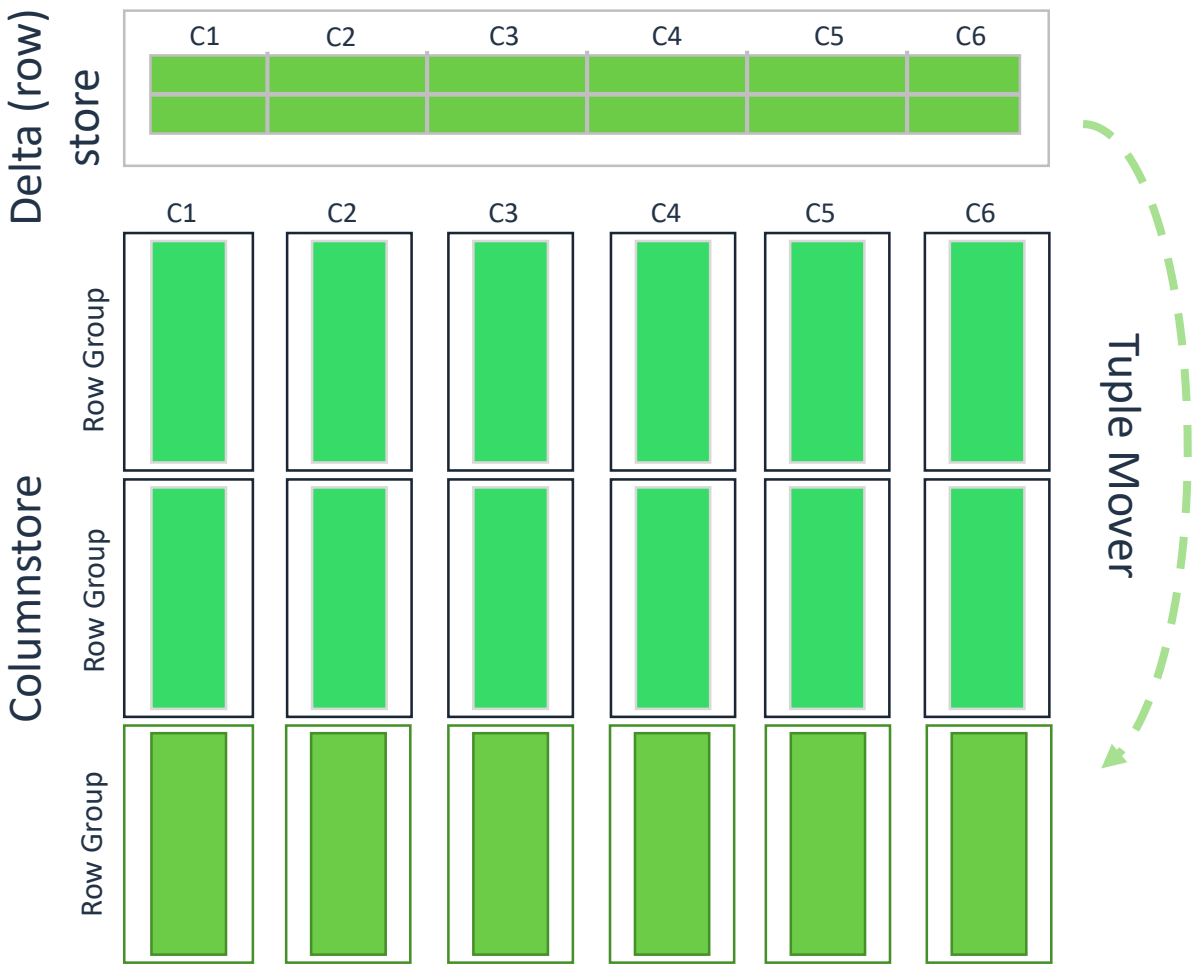


Columnstore Index

- (trickle) INSERT of single values
  - Inserted to delta store*
- BULK INSERT
  - if batch < 102400 rows
    - Lands in delta store
  - if batch >= 102400 rows
    - Lans in compressed row group
- SELECT
  - Select data from row group and delta store

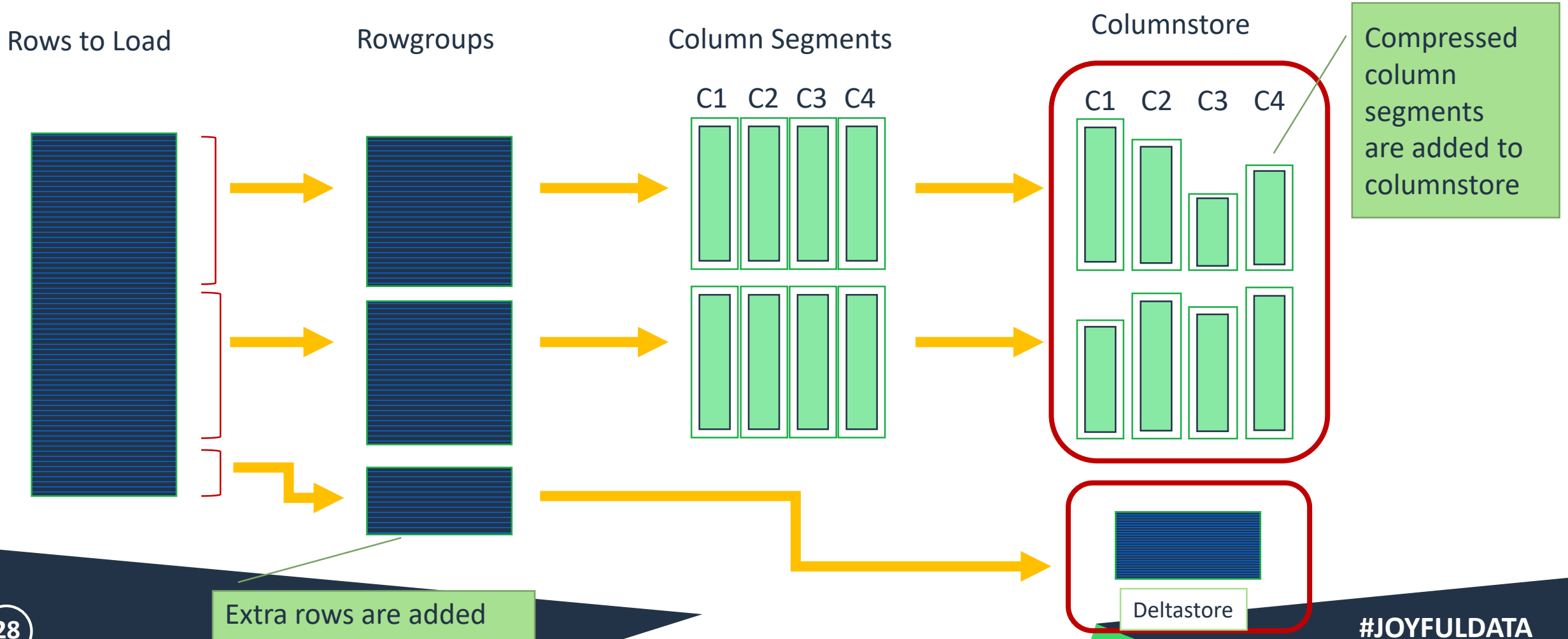
Deltastore data to Compressed Row Group

- Performed by tuple mover if segment is full (1M of rows)
- REORGANIZE statement is run
- Index Rebuild is performed



MINIMAL LOGGING
Changed between 2014 and 2016
Delta store in 2016 is no longer compressed
Before 2016
TABLOCK HINT
Trace Flag 610

BULK Insert Concept Overview
Loading Data into a Columnstore Index



Bulk Insert Optimizations

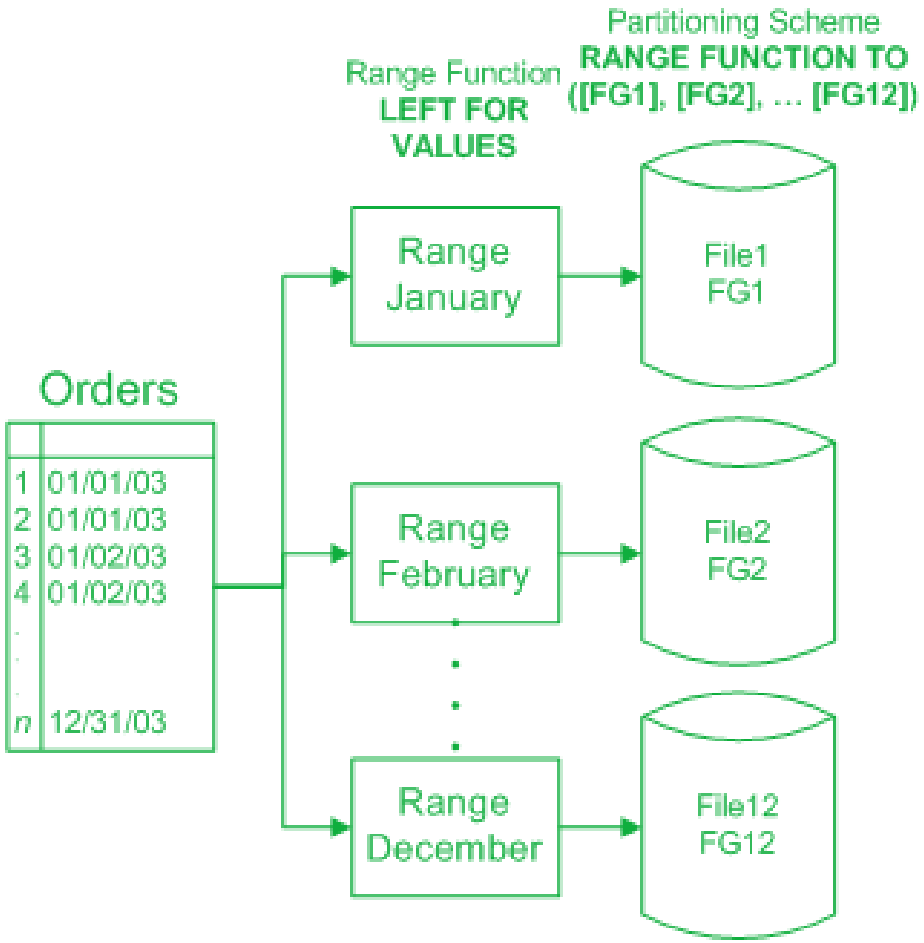
Threshold which bypasses Delta Store is 102,400 rows

- < 102,400 Rows inserted into delta store
- >= 102,400 rows directly into columnstore
- Max Size of one Row Group = 1,048,576
  - Additional rows are moved to separate Row Group

Optimal Batch Size Impact

- Batches of 90K row inserts: Go over delta store but creates larger Row Groups once the delta store is full
- Batches of 120K row inserts: Creates smaller Row Groups which may be suboptimal
- Index rebuild recreates Row Groups but the order is lost (segment elimination issue)

Partitioning

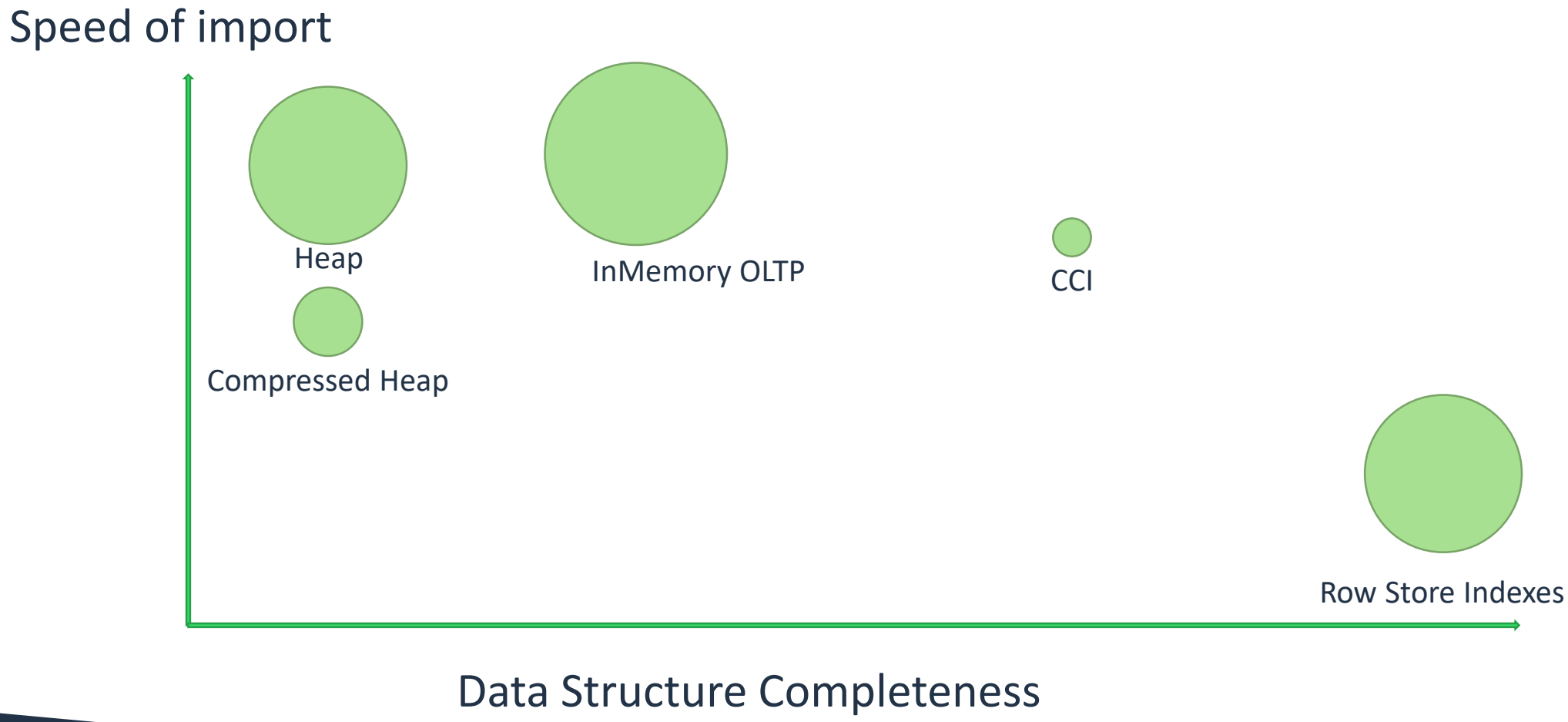


Partitioning

- Don’t load data directly
- Sort operator can be really expensive
- Recommended approach
  - Use stage heaps
  - Create

Review

Data Loading Factors



Think about import tools as well
…under certain conditions

- Bulk import operations (bcp, BULK INSERT, and INSERT… SELECT)
- SELECT INTO operations
- Partial updates to large value data types, using the .WRITE clause
- WRITETEXT and UPDATETEXT statements
- CREATE INDEX

Rowstore Overview

+ Clustered and Nonclustered indexes can be created in parallel and minimally logged when conditions are met

Table Indexes	Rows in table	Hints	Without TF 610	With TF 610 (only for SQL2014 or lower)	Concurrent possible*	Parallel
Heap	Any	TABLOCK	Minimal	Minimal	Yes	Yes
Heap	Any	None	Full	Full	Yes	No
Heap + Index	Any	TABLOCK	Full	Depends (3)	No	No
Cluster	Empty	TABLOCK, ORDER (1)	Minimal	Minimal	No	No
Cluster	Empty	None	Full	Minimal	Yes (2)	No
Cluster	Any	None	Full	Minimal	Yes (2)	No
Cluster	Any	TABLOCK	Full	Minimal	No	No
Cluster + Index	Any	None	Full	Depends (3)	Yes (2)	No
Cluster + Index	Any	TABLOCK	Full	Depends (3)	No	No
InMem data durable	Any	Can't be	Full	Full	Yes	No
InMem schema only	Any	Can't be	None	None	Yes	No

Columnstore Overview

Table Indexes	Rows in structure	Hints	Logging	Concurrent possible*	Parallel
CCI compressed rowgroup	Any	TABLOCK	Minimal	No	Yes
CCI compressed rowgroup	Any	None	Minimal*	Yes	No
CCI compressed rowgroup **+ Indexes**	Any	TABLOCK/NONE		No	No
DeltaStore	Any	TABLOCK	Minimal	No	Yes
DeltaStore	Any	None	Minimal in 2016+	Yes	Yes

Segment MIN/MAX Impact

Don’t use Azure SQL Database

Not great with VM + premium storage either

…for big data load

It’s super limited!

```
SELECT *
INTO dbo.Posts2
FROM dbo.Posts;
```

- 8 cores, $1,991 per month: 64 minutes
- **16 cores, $3,555 per month: 32 minutes**
- **80 cores, $18,299 per month: 32 minutes**

- 8-core AWS EC2 i3.2xl VM, $1,424: **2 minutes**

Business Critical service tier: Generation 5 compute platform (part 2)

Compute size	BC_Gen5_16	BC_Gen5_18	BC_Gen5_20	BC_Gen5_24	BC_Gen5_32	BC_Gen5_40	BC_Gen5_80
H/W generation	5	5	5	5	5	5	5
vCores	16	18	20	24	32	40	80
Memory (GB)	81.6	91.8	102	122.4	163.2	204	408
Columnstore support	Yes	Yes	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	15.768	18.139	20.51	25.252	37.936	52.22	131.64
Max data size (GB)	3072	3072	3072	4096	4096	4096	4096
Max log size (GB)	922	922	922	1229	1229	1229	1229
TempDB size (GB)	384	384	384	384	384	384	384
Storage type	Local SSD	Local SSD	Local SSD	Local SSD	Local SSD	Local SSD	Local SSD
IO latency (approximate)	1-2 ms (write) 1-2 ms (read)	1-2 ms (write) 1-2 ms (read)	1-2 ms (write) 1-2 ms (read)	1-2 ms (write) 1-2 ms (read)	1-2 ms (write) 1-2 ms (read)	1-2 ms (write) 1-2 ms (read)	1-2 ms (write) 1-2 ms (read)
Target IOPS (64 KB)	40000	45000	50000	60000	80000	100000	200000

Final recommendation – row store

Use partitioning in final table. To get new load there:

1. Load data to empty or nonindexed heap (with constraints to avoid sort)
   - Avoid key generation or any other parallelism inhibitor
   - Use TABLOCK hint to get parallelism
2. Create Indexes or other (partitioning unrelate) constraints afterwards
3. Switch data to final table

Consider In Memory OLTP if

- You have a lot of memory
- You need many indexes
- You don’t need parallelism

Final recommendation – column store

If you use indexes and constraints, use partitioning and have a same approach as with heaps.

Otherwise you can skip it.

- Consider skipping tablock, if you can parallelize by your own
- Always watch for batch row size (and MAXDOP) to avoid delta store

Review of review

- Use columnstore or heap if possible
- Use tablock if possible
- Use appropriate batch size
- Avoid other indexes or constraints or create them afterwards
- Avoid identity or sequences
- If your input data are ordered use it to your advantage

Resources

- CAT Team
  - The Data Loading Performance Guide (2008 R2)
- Tiger Team
  - Update for 2016 (not complete☹)
- Denzil Ribeiro
  - Data Loading performance considerations with Clustered Columnstore indexes
- Paul White
  - Miniseries about minimal logging
- Gavin Draper
  - SQL Server Minimally Logged Inserts (Assume Nothing!)

Thank you!

Questions?

Bonus material:

Application inserts into the same data structure

- Log, analytics and non-critical data data
  - Delayed durability
- Meaningful data with increasing clustered index defined or single hot page
  - Memory optimized tables

Joyful Craftsmen

Business Intelligence Crafted to Joy