PASS SUMMIT 2013

Table Partitioning:

Secret Weapon for Big Data Problems

October 15-18, 2013
Charlotte, NC

John Sterrett, Sr. Database Admin Advisor
DELL

About John

How to contact me?

http://johnsterrett.com/go/partition

Blog: http://JohnSterrett.com/
Twitter: @JohnSterrett
LinkedIn: http://linkedin.com/in/johnsterrett

Agenda

Big Data starting to slow you down? Data growth putting your maintenance tasks in danger of not meeting your SLAs? Wish you could archive old data with minimal impact to your tables during the archive process or that you could eliminate most of the data in your tables when you query them? If so, it's time you consider implementing table partitioning to help with general performance and reduce your window for completing maintenance tasks.

- Learn if Partitioning can help you manage big data.
- Understand how partitioning works.
- Learn how to maintain table partitioning with an automated sliding window.

Assumptions…

- **You are not familiar with Table Partitioning and want to see how it works.**
- You have Enterprise Edition
- Highly Transactional Tables with over 50 GB of data
- General Maintenance Tasks are in danger of not meeting your SLA
- Purge and/or Archive process is slowing you down.

The Big Question…

HOW CAN TABLE PARTITIONG MAKE MY LIFE EASIER?

How partitioning helps me?

- Reduce Maintenance Tasks

How partitioning helps me?

- Reduce Maintenance Tasks
- Improves Purging and/or Archiving

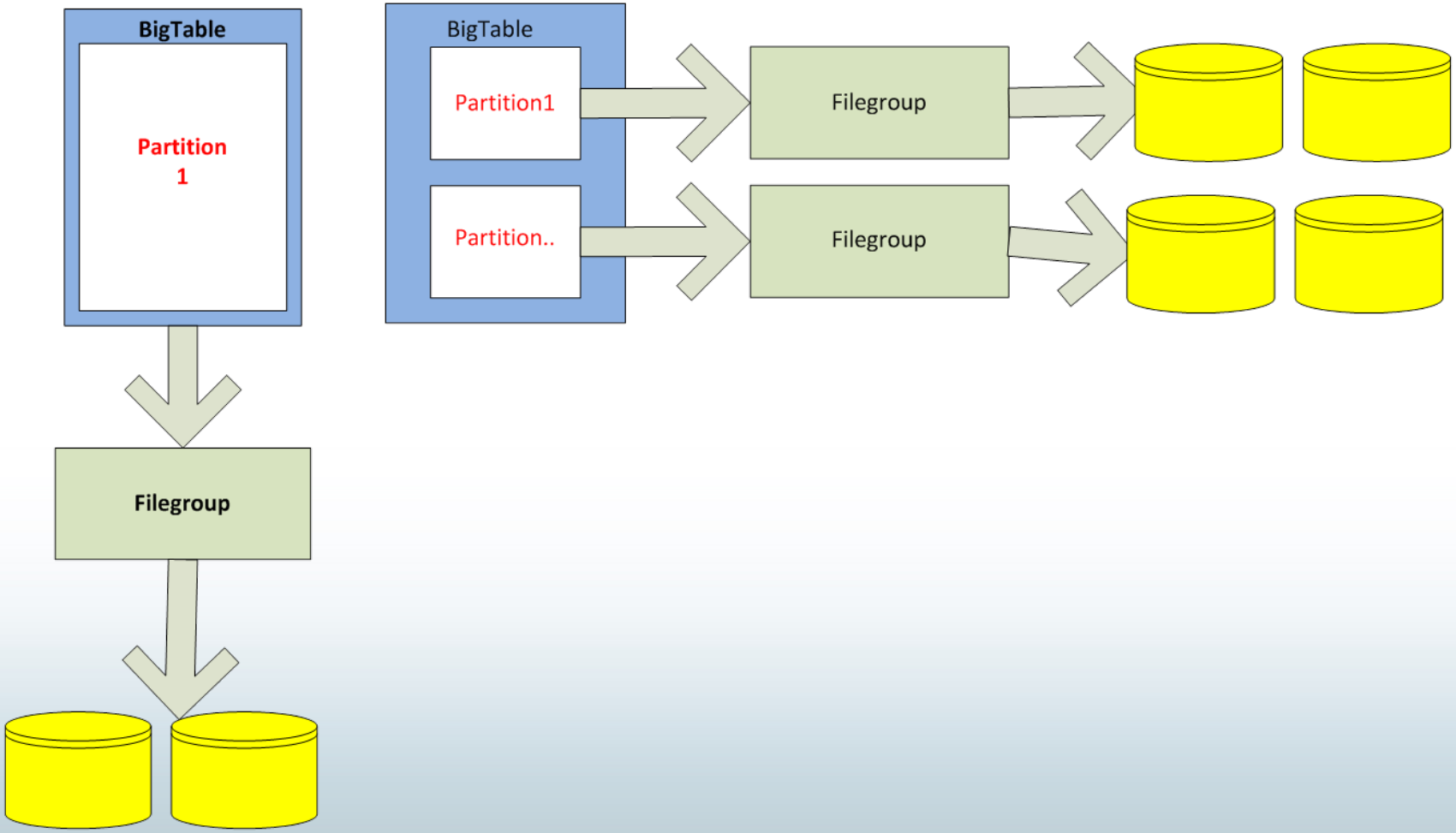
How partitioning helps me?

- Reduce Maintenance Tasks
- Improves Purging and/or Archiving
- Improves Performance

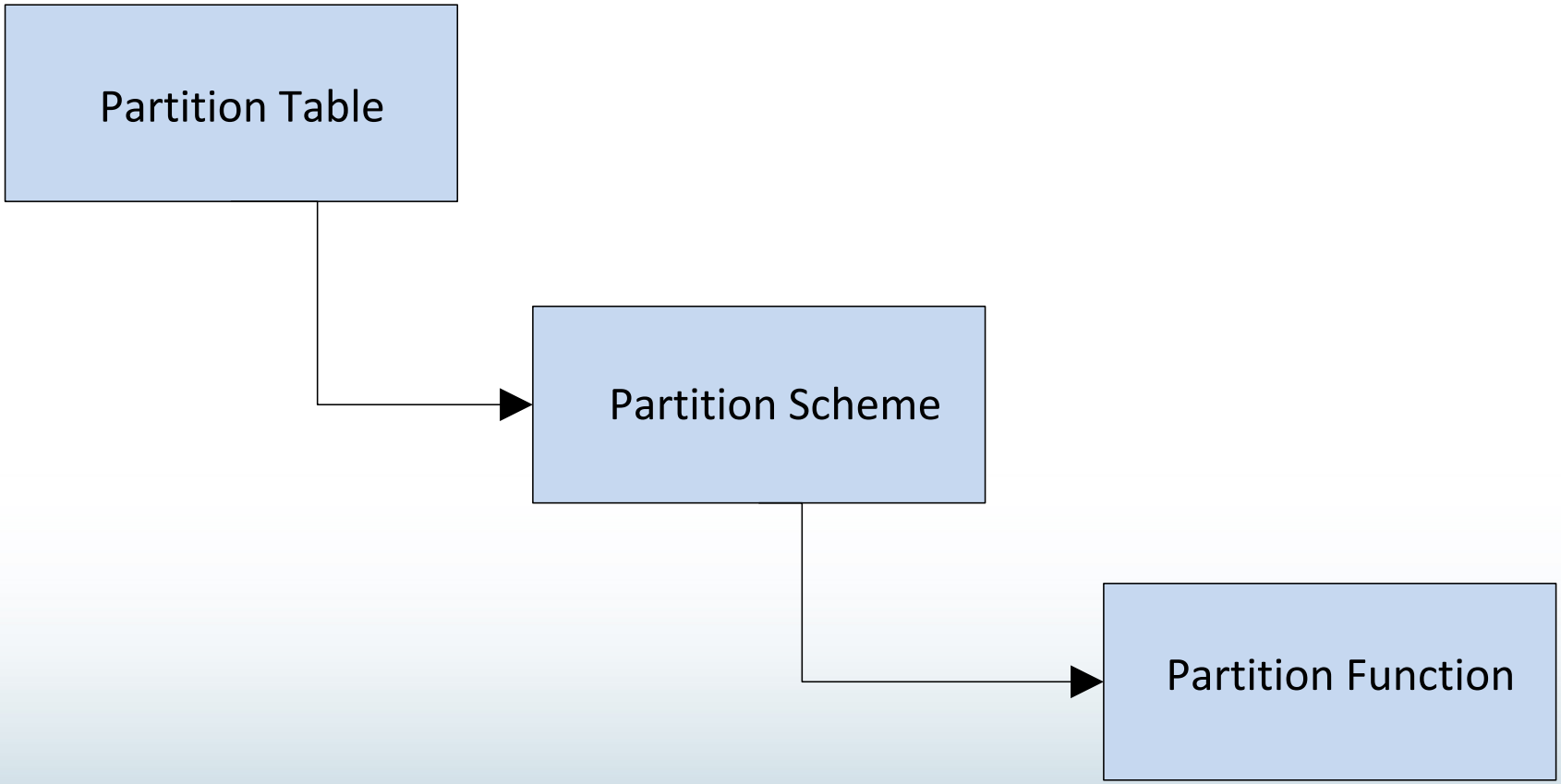
Big Question…

HOW DOES TABLE PARTITIONING WORK?

High Level…


BigTable
Partition
1
Filegroup
BigTable
Partition1
Partition..
Filegroup
Filegroup

High Level…



Selecting The Partition Column

Only get one column, use it wisely!

Column should be a highly used filter.

- Review index usage statistics
- Review Missing index statistics
- Review Queries and Talk to developers ;-)

Column must be part of clustered index

Partition Function

```
CREATE PARTITION FUNCTION Demo1_FN (int)
AS RANGE LEFT FOR VALUES (100,200,300);
```

Defines the data type used to distribute data into partitions.

Assigns boundary values to split data between partitions.

Assigns the RANGE for boundary values

Partitions = Boundary values + 1

NULL values go to left most partition

Partition Function

Partition functions **are not static**. They can change over time with SPLIT and MERGE statements.

Range LEFT is used by default.

Partition Function - Range

```
CREATE PARTITION FUNCTION Demo1_FN (int)
AS RANGE LEFT FOR VALUES (100,200,300);
```

{min….100}, {101…200}, {201...300}, {301…max}

```
CREATE PARTITION FUNCTION Demo1_FN (int)
AS RANGE RIGHT FOR VALUES (100,200,300);
```

{min…99}, {100…199}, {200…299}, {300…max}

Partition Scheme

```
CREATE PARTITION SCHEME Demo3_Scheme
AS PARTITION Demo3_Function
TO ([FG1], [FG2], [FG3], [FG4],[FG5],[FG6],[FG7]);
```

Assigns a partition function to a partition scheme

Assigns filegroups to partitions

Partitioning

DEMO

Big Question…

HOW DOES TABLE PARTITIONING IMPROVE MY MAINTENANCE TASKS?

Improving Maintenance Tasks

Backup and restore filegroups based on business priorities.

Index Maintenance by partition

New Features in SQL 2014

- Rebuild index online by partition
- Incremental Statistics by partition **(SQL 2014 CTP 2)**

Incremental Statistics

Just added in SQL Server 2014 CTP2

From Books Online:

“When ON, the statistics are recreated as per partition statistics.”

Cannot be used for the following:

- Statistics created with indexes that are not partition-aligned with the base table
- Filtered Indexes

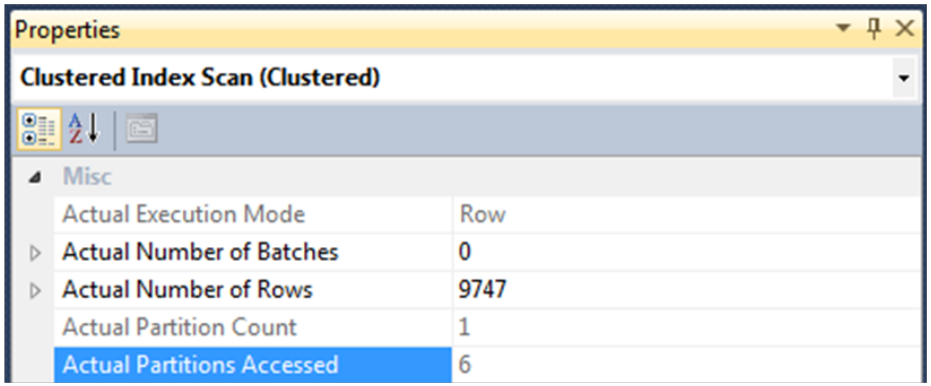
HOW DOES TABLE PARTITIONING IMPROVE PERFORMANCE?

Partition Elimination

Clustered Index Scan (Clustered)
Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	9747
Actual Number of Batches	0
Estimated I/O Cost	3.15223
Estimated Operator Cost	3.35511 (100%)
Estimated Subtree Cost	3.35511
Estimated CPU Cost	0.202881
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	9960.2
Estimated Row Size	19 B
Actual Rebinds	0
Actual Rewinds	0
Partitioned	True
Actual Partition Count	1
Ordered	True
Node ID	0

Predicate
[AdventureWorks2012].[Sales].[SalesOrderHeaderEnlarged_PT].[OrderDate]>'2011-01-01 00:00:00.000' AND [AdventureWorks2012].[Sales].[SalesOrderHeaderEnlarged_PT].[OrderDate]<'2011-12-01 00:00:00.000'
Object
[AdventureWorks2012].[Sales].[SalesOrderHeaderEnlarged_PT].[PK_SalesOrderHeaderEnlarged_PT_SalesOrderID]
Output List
[AdventureWorks2012].[Sales].[SalesOrderHeaderEnlarged_PT].CustomerID
Seek Predicates
Seek Keys[1]: Prefix: PtnId1000 = Scalar Operator((6))




```
(9747 row(s) affected)
Table 'SalesOrderHeaderEnlarged_PT'. Scan count 1, logical reads 230,

(1 row(s) affected)

(9747 row(s) affected)
Table 'SalesOrderHeaderEnlarged'. Scan count 5, logical reads 30014,
```

Skip-Scan: Seek keys

Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	1148
Actual Number of Batches	0
Estimated I/O Cost	0.021875
Estimated Operator Cost	0.0241924 (97%)
Estimated CPU Cost	0.0023174
Estimated Subtree Cost	0.0241924
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	1107.65
Estimated Row Size	15 B
Actual Rebinds	0
Actual Rewinds	0
Partitioned	True
Actual Partition Count	7
Ordered	True
Node ID	1

Object
[AdventureWorks2012].[Sales].[SalesOrderHeaderEnlarged_PT].[idx_SalesOrderHeaderEnlarged_PT_CustomerID]

Output List
[AdventureWorks2012].[Sales].[SalesOrderHeaderEnlarged_PT].OrderDate

Seek Predicates
Seek Keys[1]: Start: PtnId1000 >= Scalar Operator((1)), End: PtnId1000 <= Scalar Operator((7)), Seek Keys[2]: Prefix: [AdventureWorks2012].[Sales].[SalesOrderHeaderEnlarged_PT].CustomerID = Scalar Operator((11091))

Seek Predicates
Seek Keys[1]: Start: PtnId1000 >= Scalar Operator((1)), End: PtnId1000 <= Scalar Operator((7)), Seek Keys[2]: Prefix: [AdventureWorks2012].[Sales].[SalesOrderHeaderEnlarged_PT].CustomerID = Scalar Operator((11091))

Index Seek (NonClustered)

Misc	
Actual Execution Mode	Row
Actual Number of Batches	0
Actual Number of Rows	1148
Actual Partition Count	7
Actual Partitions Accessed	1..7

Database Compression by Partition

```sql
ALTER INDEX idx_SalesOrderHeaderEnlarged_PT_rowguid
ON Sales.SalesOrderHeaderEnlarged_PT
REBUILD Partition = 6
WITH (DATA_COMPRESSION = PAGE);

ALTER INDEX idx_SalesOrderHeaderEnlarged_PT_rowguid
ON Sales.SalesOrderHeaderEnlarged_PT
REBUILD Partition = 7
WITH (DATA_COMPRESSION = ROW);
```

Partitioning

DEMO

HOW DOES PARTITIONING IMPROVE ARCHIVING AND PURGING?

Real World Example:

runDate	Hours	Minutes	Seconds	name
2013-06-18 00:00:00.000	3	29	37	EVENT_LOG Purge Job

runDate	Hours	Minutes	Seconds	name
2013-08-20 00:00:00.000	0	1	0	EVENT_LOG - SlidingWindow
2013-08-20 00:00:00.000	0	1	3	EVENT_LOG - SlidingWindow
2013-08-21 00:00:00.000	0	1	3	EVENT_LOG - SlidingWindow

Quickest way to move billions of rows

```
ALTER TABLE [Sales].[SalesOrderHeaderEnlarged_PT]
    SWITCH PARTITION 2 TO [Sales].[SalesOrderHeaderEnlarged_Staging];
```

Meta data swap is quickest way to move billions of rows **<u>assuming you can get a schema lock.</u>**

Sliding Window Goals

SPLIT and MERGE with empty partitions

Use SWITCH to do meta-data swaps

Minimize all physical data movement

Range Right - MERGE

```sql
ALTER PARTITION FUNCTION Demo1_FN()
MERGE RANGE (100)
```

PartitionSchemeName	RangeType	PartitionID	BoundaryValue	FileGroup
Demo1_Scheme	RIGHT	1	NULL	FG1
Demo1_Scheme	RIGHT	2	100	FG2
Demo1_Scheme	RIGHT	3	200	FG3
Demo1_Scheme	RIGHT	4	300	FG4

PartitionSchemeName	RangeType	PartitionID	BoundaryValue	FileGroup
Demo1_Scheme	RIGHT	1	NULL	FG1
Demo1_Scheme	RIGHT	2	200	FG3
Demo1_Scheme	RIGHT	3	300	FG4

Range Left - MERGE

```
ALTER PARTITION FUNCTION Demo1_FN()
MERGE RANGE (200)
```

PartitionSchemeName	RangeType	PartitionID	BoundaryValue	FileGroup
Demo1_Scheme	LEFT	1	100	FG1
Demo1_Scheme	LEFT	2	200	FG2
Demo1_Scheme	LEFT	3	300	FG3
Demo1_Scheme	LEFT	4	NULL	FG4

PartitionSchemeName	RangeType	PartitionID	BoundaryValue	FileGroup
Demo1_Scheme	LEFT	1	100	FG1
Demo1_Scheme	LEFT	2	300	FG3
Demo1_Scheme	LEFT	3	NULL	FG4

Range Right - SPLIT

```
ALTER PARTITION FUNCTION Demo1_FN()
SPLIT RANGE (350)
```

PartitionSchemeName	RangeType	PartitionID	BoundaryValue	FileGroup
Demo1_Scheme	RIGHT	1	NULL	FG1
Demo1_Scheme	RIGHT	2	200	FG3
Demo1_Scheme	RIGHT	3	300	FG4

Range Right - SPLIT

PartitionSchemeName	RangeType	PartitionID	BoundaryValue	FileGroup
Demo1_Scheme	RIGHT	1	NULL	FG1
Demo1_Scheme	RIGHT	2	200	FG3
Demo1_Scheme	RIGHT	3	300	FG4
Demo1_Scheme	RIGHT	4	350	FG2

Range Left - SPLIT

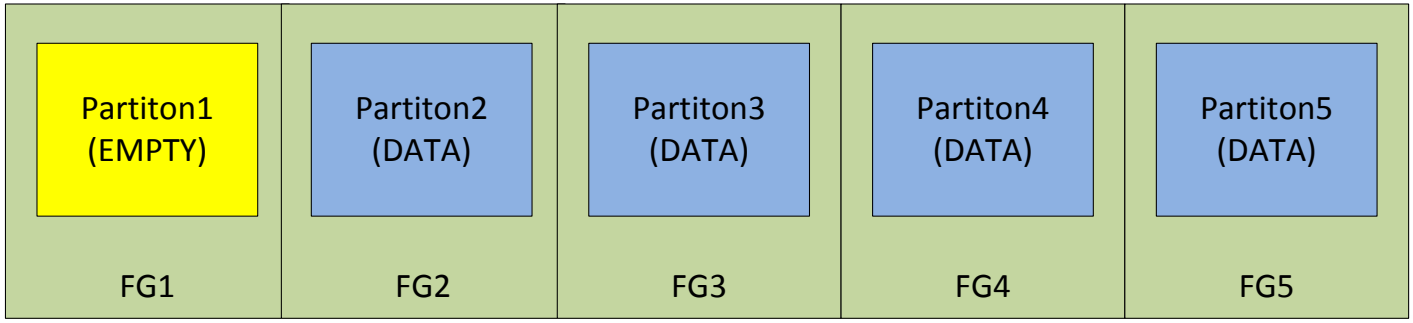
```
ALTER PARTITION FUNCTION Demo1_FN()
SPLIT RANGE (350)
```

PartitionSchemeName	RangeType	PartitionID	BoundaryValue	FileGroup
Demo1_Scheme	LEFT	1	100	FG1
Demo1_Scheme	LEFT	2	300	FG3
Demo1_Scheme	LEFT	3	NULL	FG4

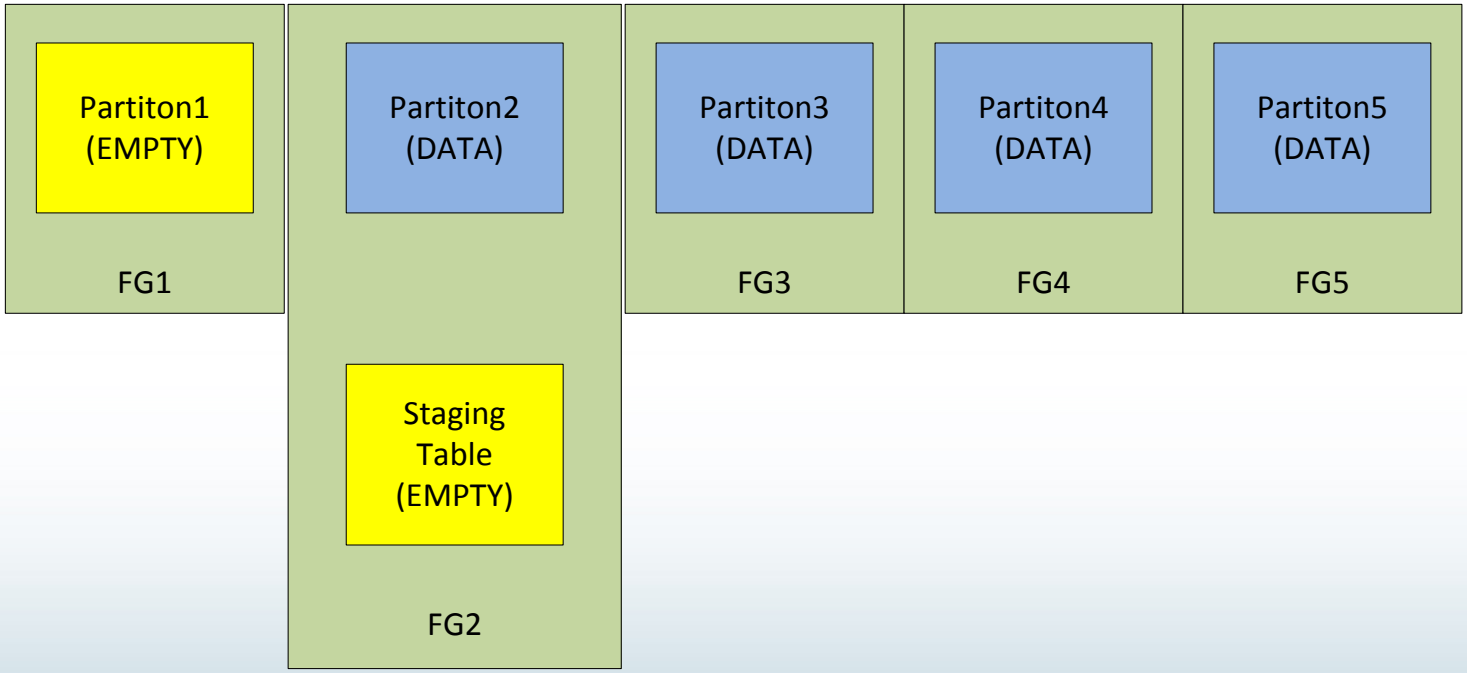
PartitionSchemeName	RangeType	PartitionID	BoundaryValue	FileGroup
Demo1_Scheme	LEFT	1	100	FG1
Demo1_Scheme	LEFT	2	300	FG3
Demo1_Scheme	LEFT	3	350	FG2
Demo1_Scheme	LEFT	4	NULL	FG4

Visual Sliding Window Example

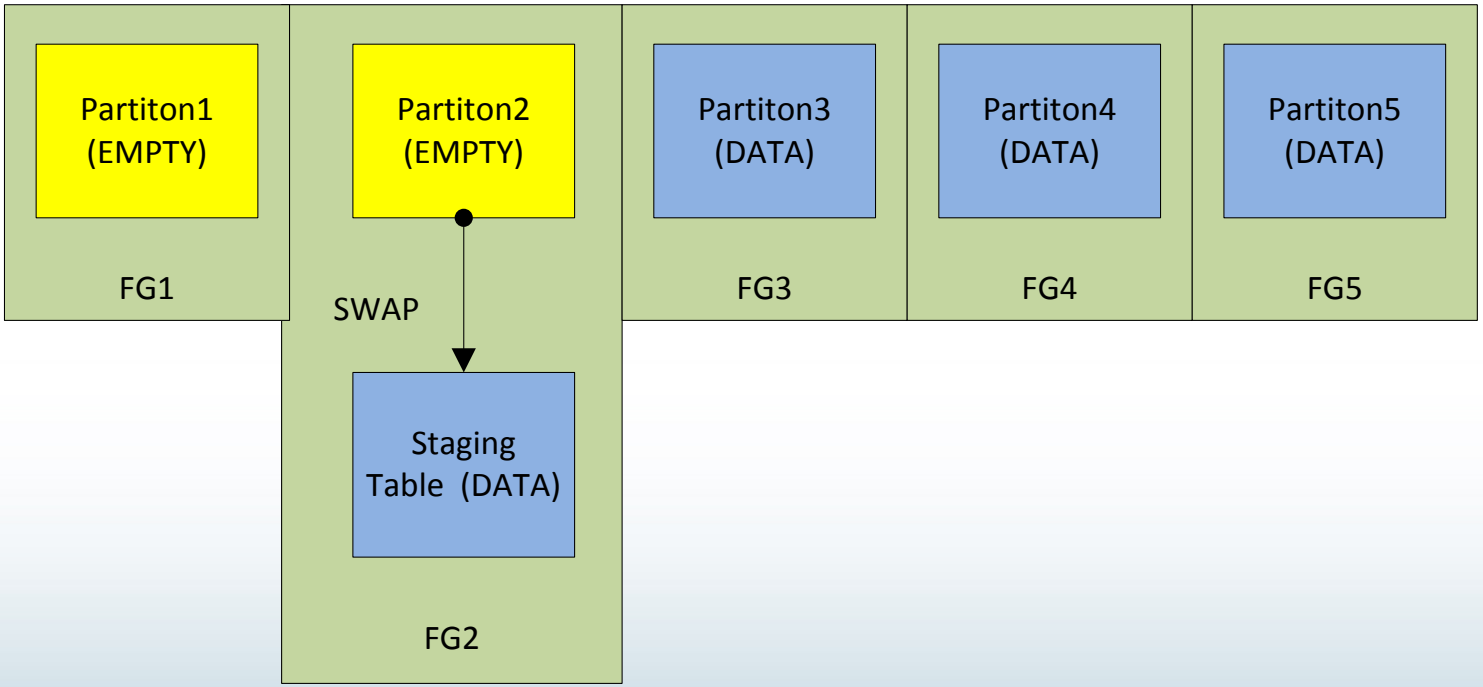
Visual Sliding Window Example



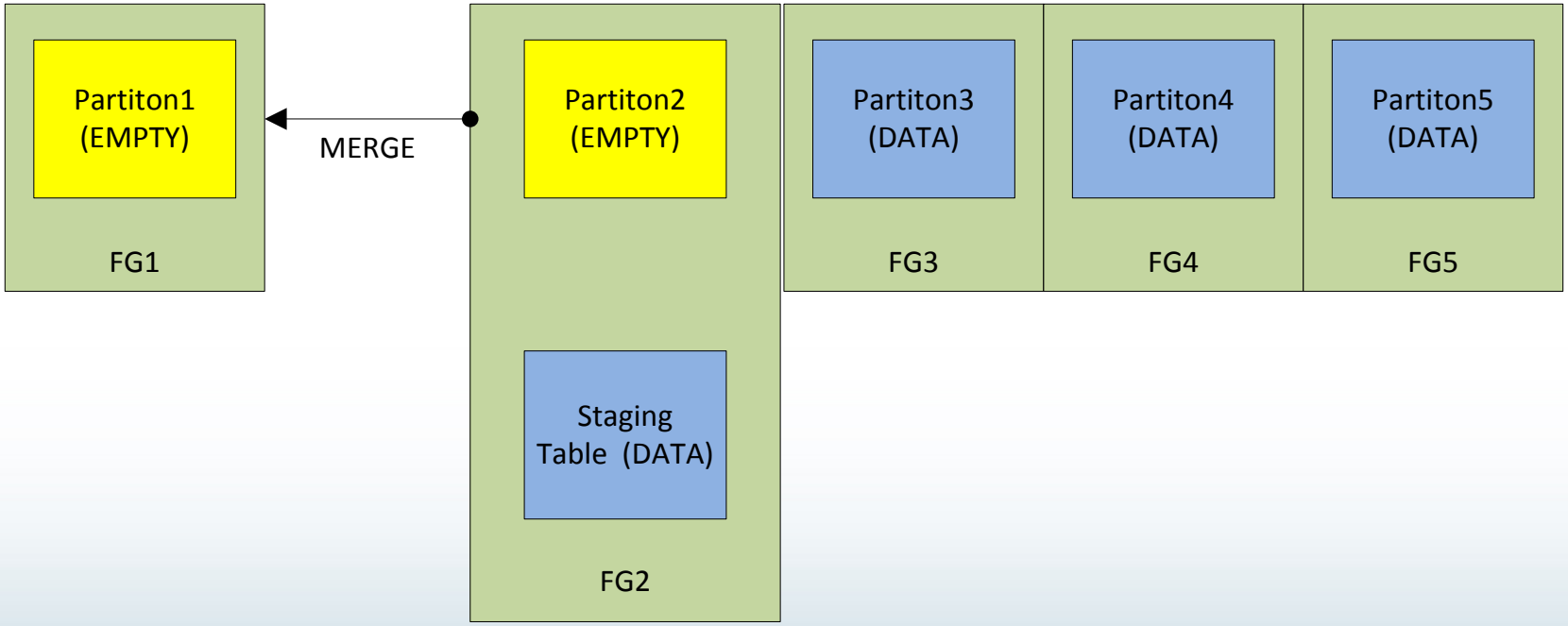
Visual Sliding Window Example



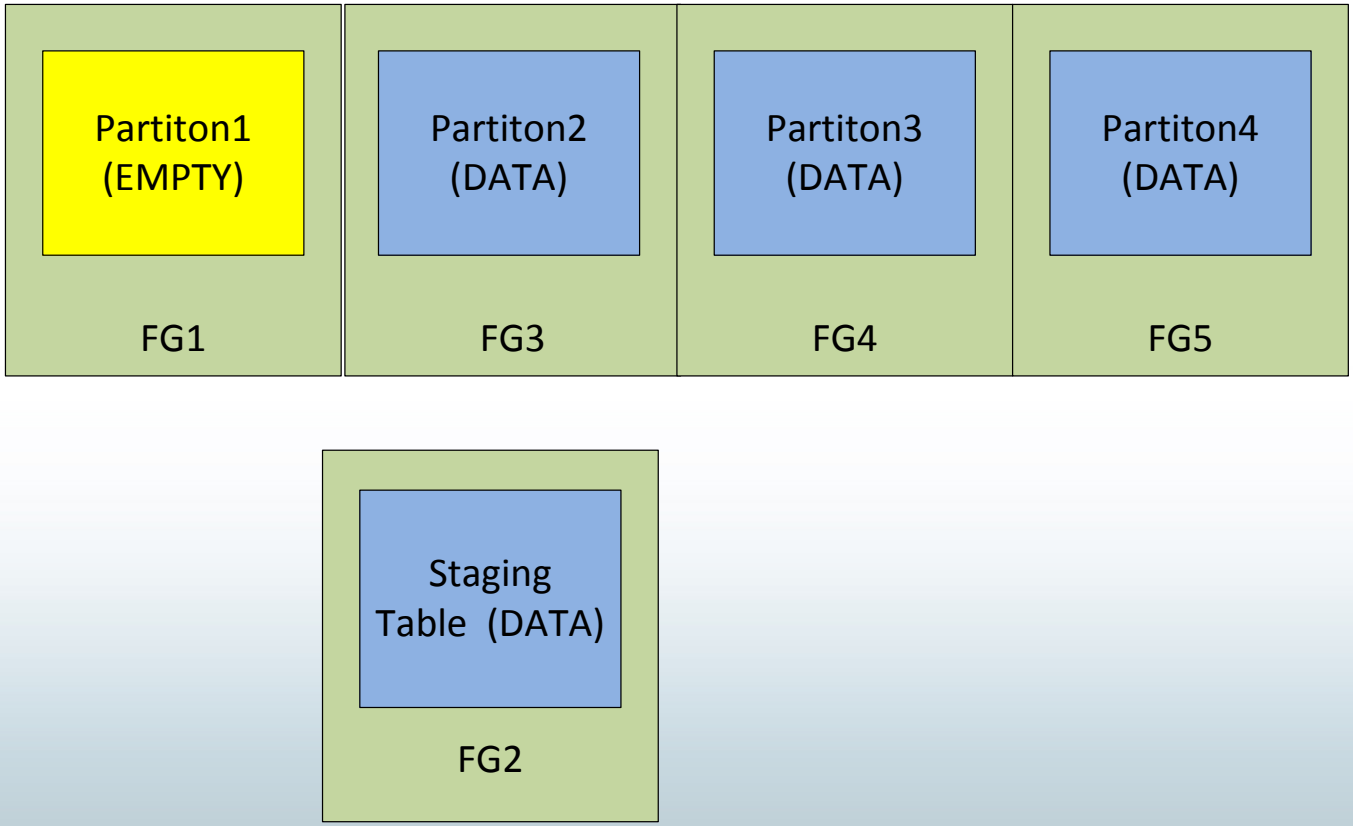
Visual Sliding Window Example



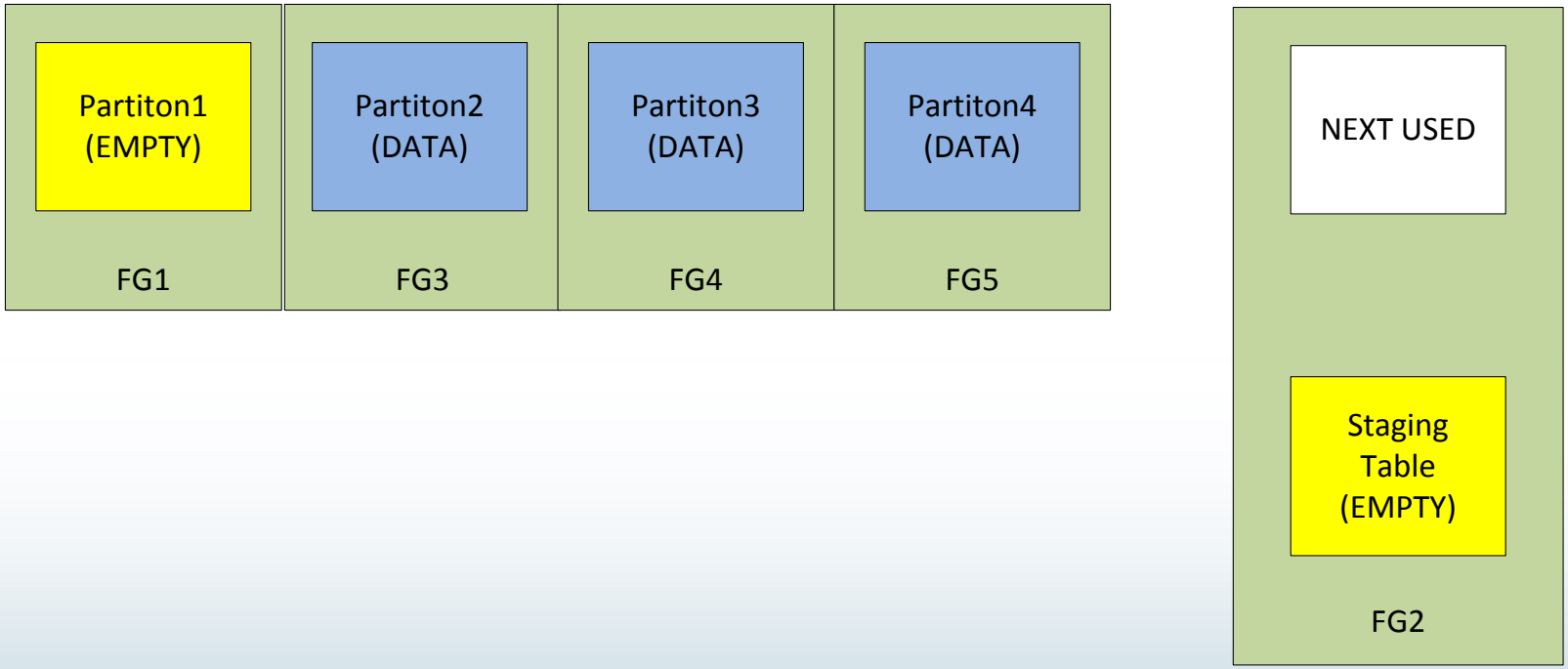
Visual Sliding Window Example



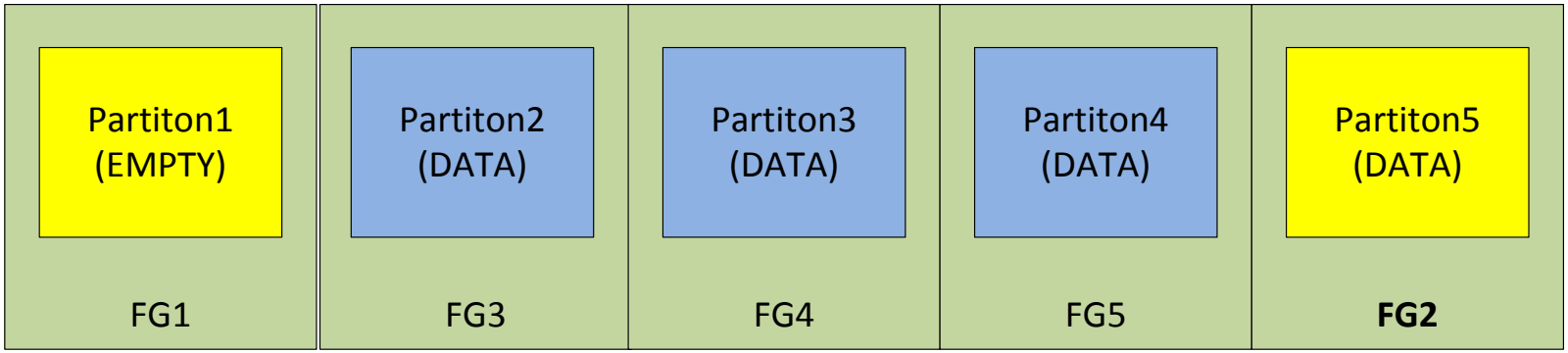
Visual Sliding Window Example



Visual Sliding Window Example



Visual Sliding Window Example



Sliding Window Steps...

Create partition swap table (if it doesn't exist)

1. Insert partition swap meta data
2. Create staging table
3. Meta-data swap (partition 2 with staging table)
4. Merge Partitions #1 and #2
5. Mark next used partition
6. Split to create new partition
7. Update processed partition swap meta data.

Partitioning

DEMO

Questions…

http://johnsterrett.com/go/partition

Blog: http://JohnSterrett.com/
Twitter: @JohnSterrett
LinkedIn: http://linkedin.com/in/johnsterrett

Thank you

for attending this session and the
2013 PASS Summit in Charlotte, NC