



IDUG*VIRTUAL*

2021 Australasia **Db2** Tech Conference

Db2 reorg
unleash the full power of the utility

Markus Fraune, ITGAIN



Db2

Agenda

- Basics (why and how to reorg)
- Classic vs inplace
- How to run a reorg in parallel
- Alternative to reorg (amt)
- Performance Comparison

Reorgs are recommended, because (1|2)

- Someone executed a certain alter table command
- The following is the full list of REORG-recommended ALTER statements that cause a version change and place the table into a REORG-pending state:
 - DROP COLUMN
 - ALTER COLUMN SET NOT NULL
 - ALTER COLUMN DROP NOT NULL
 - ALTER COLUMN SET DATA TYPE, except in the following situations:
 - Increasing the length of a VARCHAR or VARGRAPHIC column
 - Decreasing the length of a VARCHAR or VARGRAPHIC column without truncating trailing blanks from existing data, when no indexes exist on the column
- Two (or more) alter commands in one UOW increase the counter by 1

Reorgs are recommended, because (2|2)

- **up** to 3 single reorg recommended operations allowed
- some commands possible if table in reorg pending state:
 - Drop table
 - Rename table
 - Truncate table
 - Reorg table (offline and full table only)
- Check for current counters: SYSIBMADM.ADMINTABINFO (use a few columns and a where clause to filter for specific table(s) or schema)

Reorgs are needed (reorg pending), because

- Reorg recommended counter is >0 / reorg pending state
- Regular tablespace has reached max pages limit and has to be converted to large (you will need a reorg indexes to „activate“ large RIDs and make table „insertable“ again)

Reorgs are really recommended, because (1|2)

- Reorgcheck shows at least 2* on a table or 3* on an index (data is really bad fragmented)
- Performance is slow (having overflow records), like after adding new columns or heavy updates on variable columns with larger data



Reorgs are really recommended, because (2|2)

- compression is now inplace (generate compression dictionary)
 - Generate a dictionary because of high amount of new data that is compressed badly
 - After a load operation as records used to generate initial dictionary are a bad sample
 - After changing from dictionary compression to adaptiv compression
- Everytime you convert tablespace from regular to large

how execute a reorg (1|3)

- via command line
 - db2 „reorg table myschema.mytable“
- Via SQL
 - CALL SYSPROC.ADMIN_CMD ('REORG TABLE myschema.mytable')
 - CALL SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS('table', 'MYSCHEMA', 'MYTABLE')

only for tables in reorg pending state, will execute and runstats for those

how execute a reorg (2 | 3)

- Can be executed for a table
 - db2 „reorg table myschema.mytable“
- Can be executed for a data partition of a table (range parted)
 - db2 „reorg table myschema.mytable on data partition part001“
- Can be executed for a database partition of a table (DPF)
 - db2 „reorg table myschema.mytable on DBPARTITIONNUM (1)“

how execute a reorg (3 | 3)

- Can be executed for all indexes of a table
 - db2 „reorg indexes all for table myschema.mytable“
- Can be executed for a given index of a table
 - db2 „reorg index myindex for table myschema.mytable“
 - Only supported for:
 - Nonpartitioned indexes on a data partitioned table that are not block indexes
 - Any index on any permanent table if CLEANUP ALL is specified and RECLAIM EXTENTS is not specified

Pitfalls when executing a reorg (1 | 3)

- Table reorg always recreates all indexes as well
 - Maybe better to drop idx / reorg table / create idx (especially when reorg gets logged in hadr -> avoid log full)
- Reorg a table partition with non parted indexes will lead to a reorg of the non parted idx for the whole table
- If partitioned table is in reorg pending and is having non partitioned indexes you will have to reorg whole table at once (not per partition)
- Inplace / online reorg is not allowed when in reorg pending state

Pitfalls when executing a reorg (2 | 3)

- By default lobs are not reorganized (use parameter LONGLOBDATA)
- By default db2 uses the same tablespace for the working copy (if partitioned it will use the tablespace of that partition)
 - At least twice space is needed in the same tablespace or specify a system temp space to write working copy in that tablespace
- Reorg on the complete partitioned table will result in 1 partition reorg at a time (at least twice space if largest partition is needed)
- „The original table might be available for queries until the replace operation starts, depending on the access clause“ (classic/offline reorg)

Pitfalls when executing a reorg (3 | 3)

- Allow read access on partitioned table only supported when having no non partitioned indexes (otherwise it will be allow no access) - all other partitions would be read and writeable
- When having a non partitioned index the whole table will be set in no access mode (not only the given partition)
- By default no automatic parallelism for a table (it will always use a single thread, one partition after another)

How to avoid reorgs

- Use more alter table statements in one OUW to avoid getting from reorg recommended to reorg pending status
- Minimize the usage of VARCHAR, VARGRAPHIG to avoid overflows
- Use ITC (Insert Time Clustering) tables - data and indexes will be stored and deleted in blocks - no reorg needed (and not supported) and having a new not usable column and index to store the insert time
- Use admin move table before it comes to reorg pending situation

Monitoring a reorg

- GET SNAPSHOT FOR TABLES command
 - db2 get snapshot for tables on sample
- db2pd -reorg command
 - db2pd -db sample -reorg -repeat 60
- List History Reorg:
 - db2 list history reorg all for sample
- Select from the SYSIBMADM.SNAPTAB_REORG
- INPLACE_REORG_STATUS in SYSPROC.ADMIN_GET_TAB_INFO
- Use third party monitoring tool

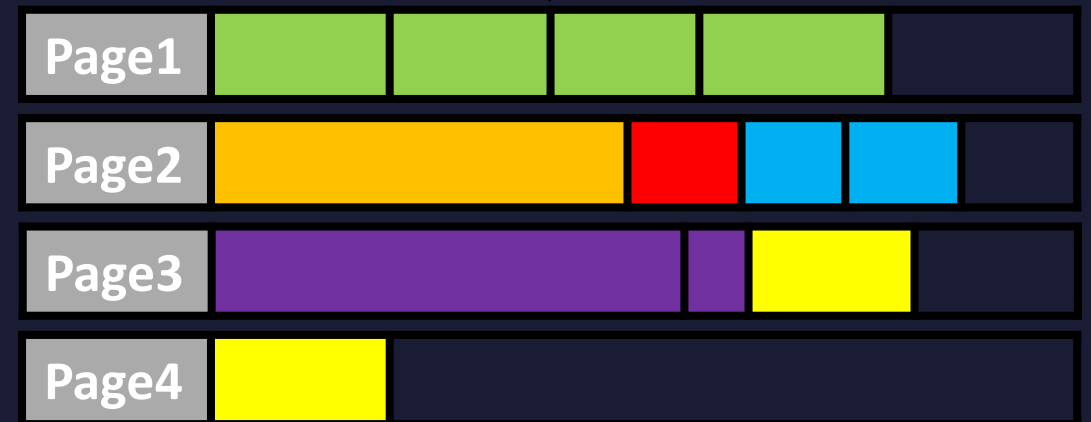
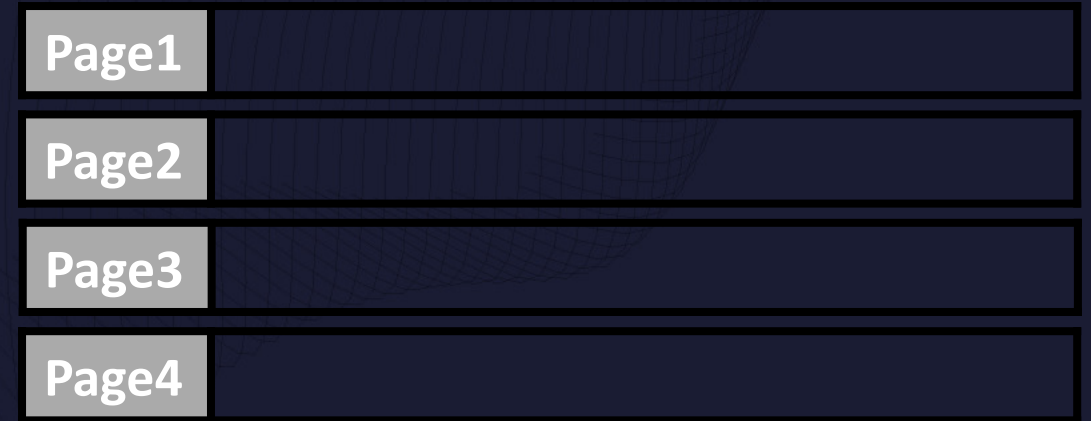
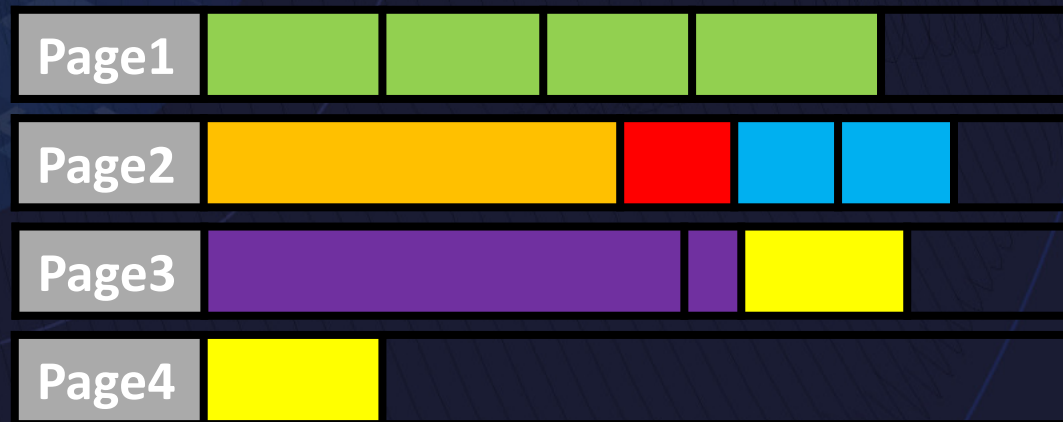
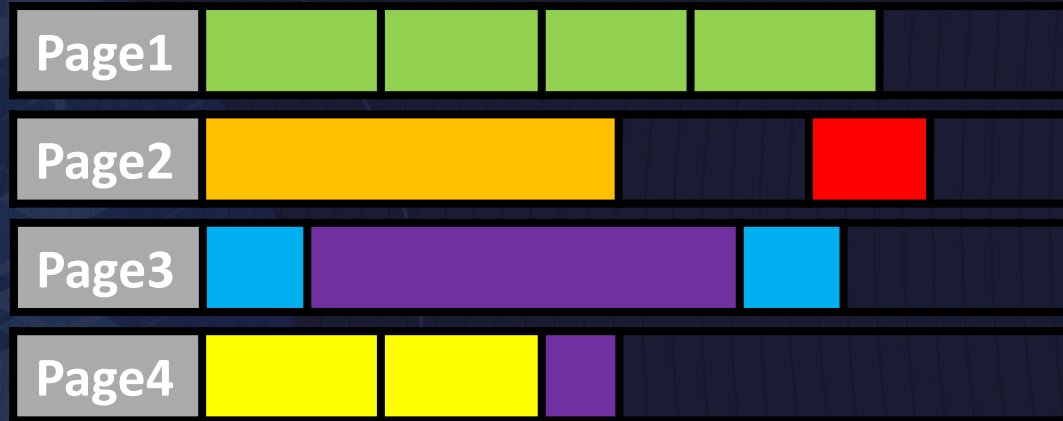
Classic vs. inplace (offline vs. online) (1 | 2)

- Classic
 - „No“ access to the table
 - Uses a copy approach, building a full copy of the table
 - Fast
 - Perfectly clustered data once finished
 - Indexes gets rebuild automatically
 - Can use temp tablespace to reduce needed storage in tablespace
- Inplace
 - Full access to the table
 - Rows are moved within the table, sequentially
 - Slow
 - Maybe imperfect clustering depending on sql during reorg
 - Indexes maintained but not rebuild
 - Low storage requirement as using source object to move data

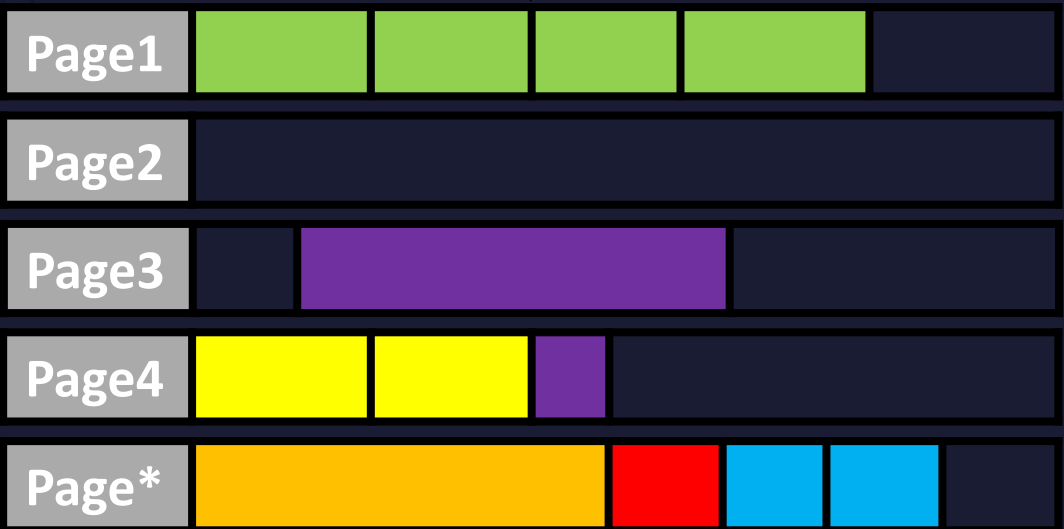
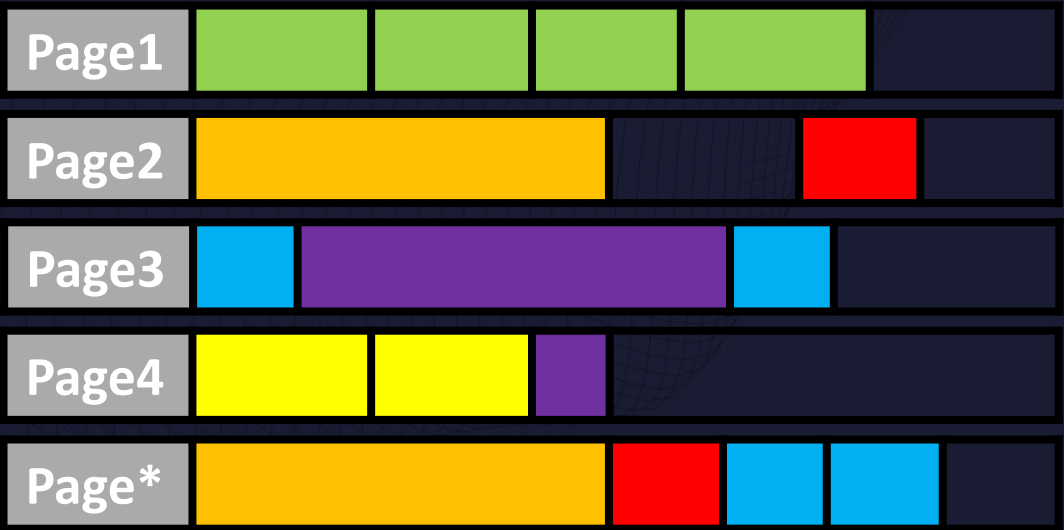
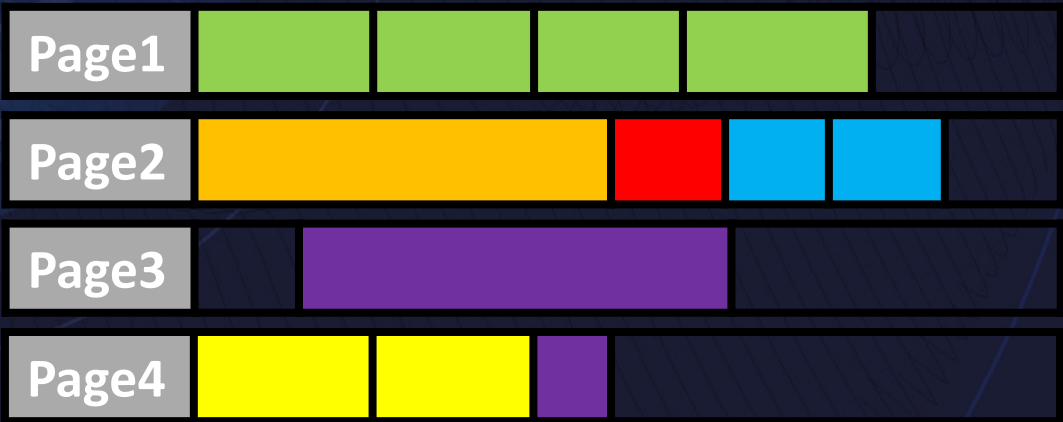
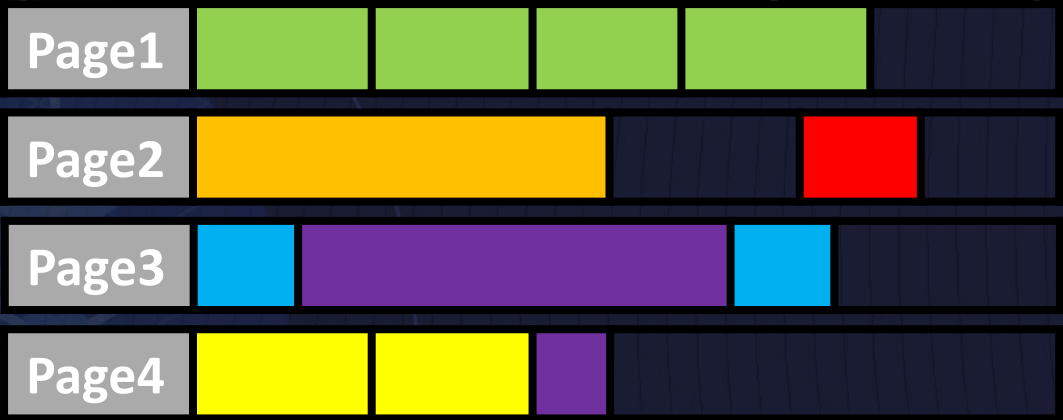
Classic vs. inplace (offline vs. online) (2 | 2)

- Classic
 - Large space requirement (shadow copy)
 - No control during execution (has to be restarted if it stops)
 - Not recoverable
 - Needs fewer TRX Log
 - Benefit once the reorg completes
 - Suitable for large tables
- Inplace
 - Small space requirement (moving data in existing object)
 - Good control during the execution (pause, resume, start, stop)
 - Fully recoverable
 - Needs more TRX Log
 - Direct benefit right after reorg starts
 - Not suitable for large tables (may never finish)

Classic workflow (basic)



Inplace workflow (basic)



Reorg options for partitioned Tables (1|4)

- A reorg can run on a single partition of range partitioned table, allowing access on all other partitions during the execution
- Multiple partition reorgs can run in parallel with following criteria:
 - Each REORG command must specify a different partition with the ON DATA PARTITION clause.
 - Each REORG command must use the ALLOW NO ACCESS mode to restrict access to the data partitions.
 - The partitioned table must have only partitioned indexes if issuing REORG TABLE commands. No nonpartitioned indexes (except system-generated XML path indexes) can be defined on the table.

Reorg options for partitioned Tables (2|4)

- Restrictions for an index to be partitionable
 - To be partitioned the index will have to include the distribution key
`create index parted_1 on myschema.mytable (distrib_col,col1) partitioned`
 - If distribution key is/can not be included, the index can be partitioned (if index is not unique) and the column will be included (hidden) -> maybe this index will be used differently in SQL queries (Data-partitioned secondary index (DPSI) on z/OS)
`create index parted_2 on myschema.mytable (col1) partitioned`

Reorg options for partitioned Tables (3|4)

- Examples:

- Table is partitioned by quarters of a year, reorg per quarter or in parallel

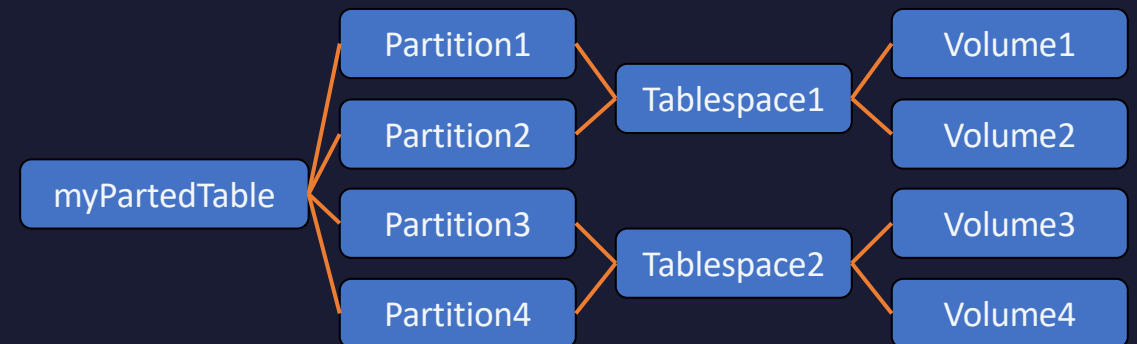
- REORG TABLE myschema.mytable ALLOW NO ACCESS ON DATA PARTITION P1
- REORG TABLE myschema.mytable ALLOW NO ACCESS ON DATA PARTITION P2
- REORG TABLE myschema.mytable ALLOW NO ACCESS ON DATA PARTITION P3
- REORG TABLE myschema.mytable ALLOW NO ACCESS ON DATA PARTITION P4

- What also is possible to be excuted in parallel

- REORG INDEXES ALL FOR TABLE myschema.mytable ALLOW NO ACCESS ON DATA PARTITION P1
- REORG TABLE myschema.mytable ALLOW NO ACCESS ON DATA PARTITION P2
- REORG INDEXES ALL FOR TABLE myschema.mytable ALLOW NO ACCESS ON DATA PARTITION P3

Reorg options for partitioned Tables (4|4)

- Recommendations:
 - Use separate/dedicated tablespaces for every partition or group partitions if having several disks/volumes per path
 - Parallelism should not be higher than number of (free) Cores and number of disks/Volumes (one reorg will use full power of one core and underlying storage)



Admin Move Table - Basics (1 | 3)

- System Procedure officially introduced in v9.7
- Basically developed to move on table from one tablespace to another without blocking the table for UID
- Easy to use but flexibel for advanced usage (one command for the complete workflow vs. one command per step and more customization options)
- Columns can be added/dropped -> minize risk of reorg pending
- Improvements over last versions, getting less restrictions and more options

Admin Move Table - Basics (2 | 3)


- Admin Move Table has basically 4 needed steps:
 - INIT: Verifies requirements, creates target and staging tables and triggers
 - COPY: Copies data from source to target table (Insert or Load)
 - REPLAY: Move Data from staging to target (repeat this step if first execution took long time)
 - SWAP: Rename the target table to source table, rename indexes, remove staging table, remove source table (optionally keep it)
- And just one to go complete through, to terminate and to clean:
 - Move: Performs INIT, COPY, REPLAY, and SWAP operations in one step
 - Term: Terminates a running or killed table move
 - CANCEL: clears up all intermediate data

Admin Move Table - Basics (3 | 3)

- Two Procedure Methods (easy and more options, non-optional: tabschema, tabname and operation)


Method 1:

► ADMIN_MOVE_TABLE (— tabschema — , — tabname — , — data_tbsp — , — index_tbsp — , — lob_tbsp — , —
 — organize_by_clause — , — partkey_cols — , — data_part — , — coldef — , — options — , — operation —) ►



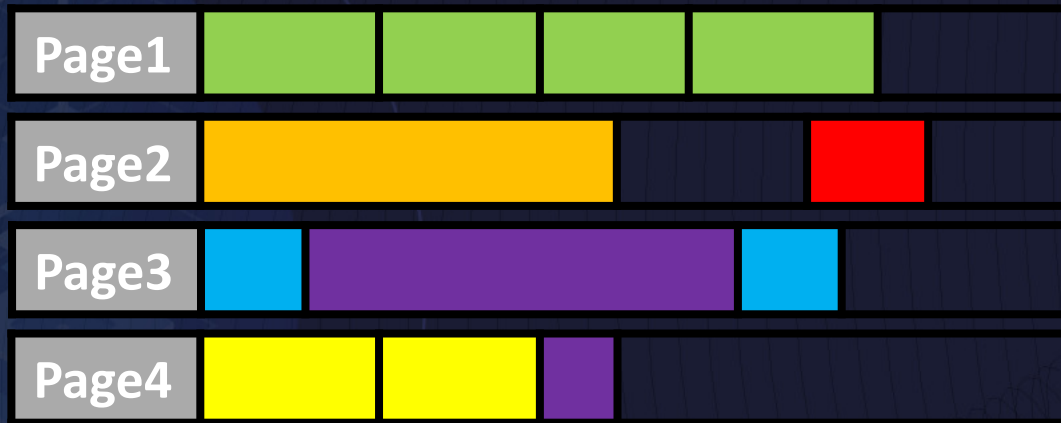
Method 2:

► ADMIN_MOVE_TABLE (— tabschema — , — tabname — , — target_tabname — , — options — , — operation —
) ►



Admin Move Table - Init Phase

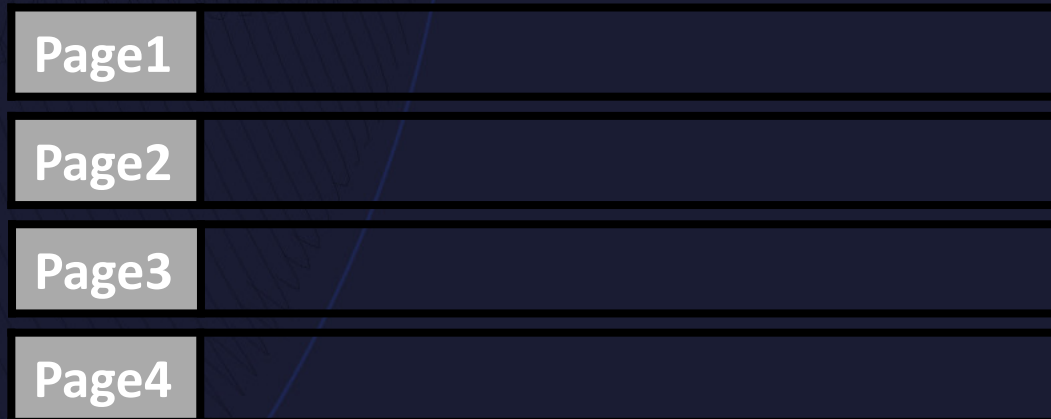
Source



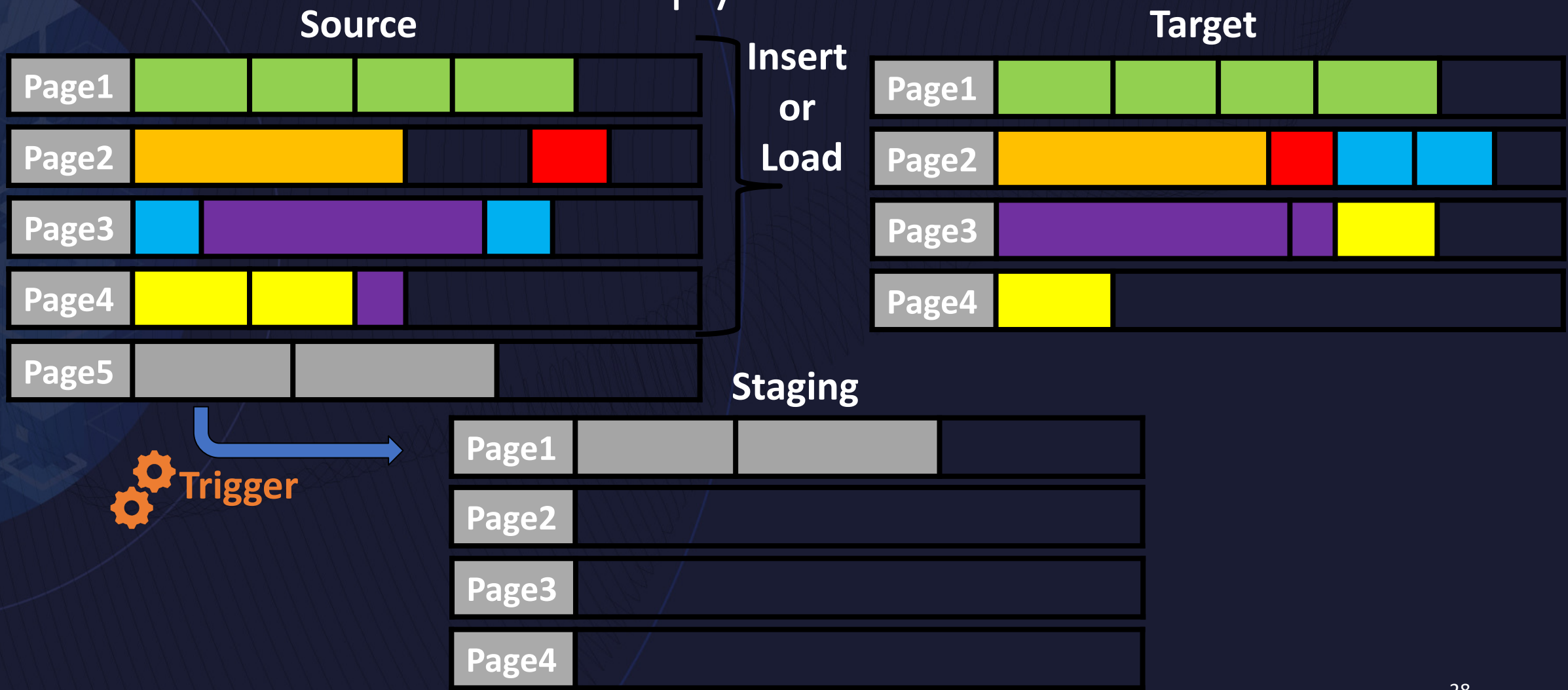
Target



Staging



Admin Move Table - Copy Phase



Admin Move Table - Replay Phase

Source

Page1					
Page2					
Page3					
Page4					
Page5					

Target

Page1					
Page2					
Page3					
Page4					

Staging

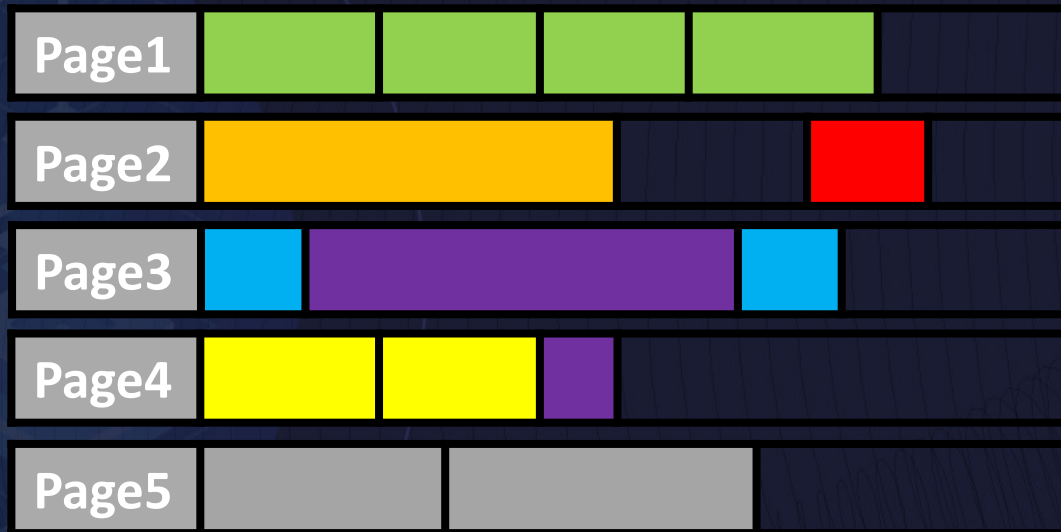
Page1			
Page2			
Page3			
Page4			

Insert

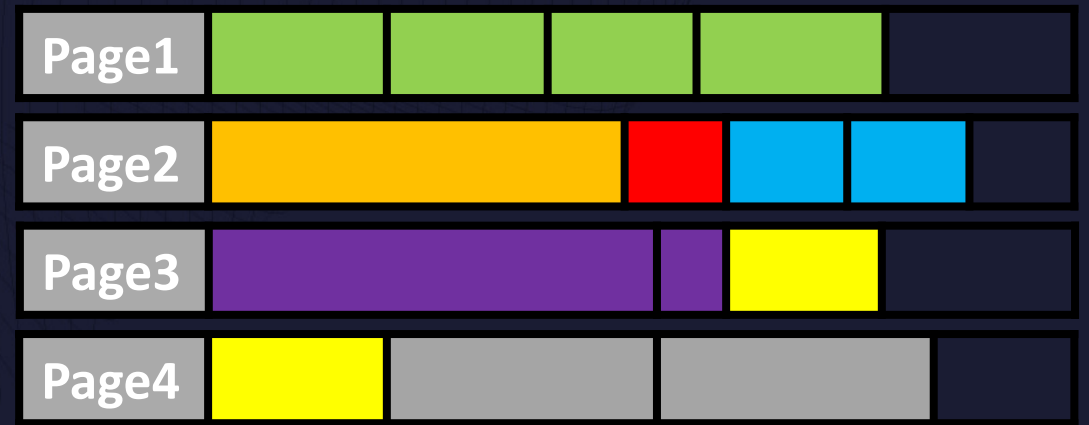


Admin Move Table - Swap Phase

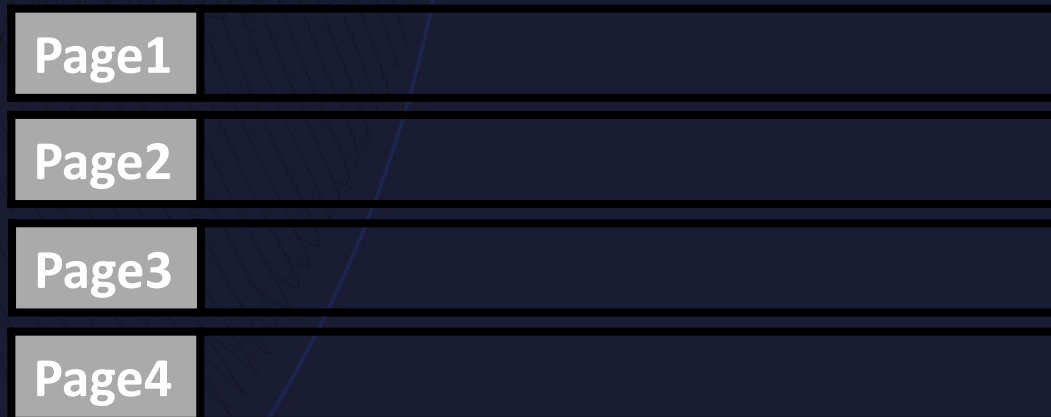
Source → Gets deleted



Target → Rename to final name



Trigger --> get deleted



Staging → gets deleted

Admin Move Table - alternative to reorg

- Can only be used for „really“ reorg recommended **not** for reorg pending tables (use it before having reorg pending status)
- More or less fully online - during swap phase X lock needed on source, rebinds needed after the swap
- Needs at least twice the space as source (maybe less if compression rate improves or had a lot of gaps in source)
- Reads data from source by default with order by if a clustering index exist or cluster option is specified (and clustering index, uniq index or primary key exists or an alternative index to use has been specified with ADMIN_MOVE_TABLE_UTIL)

Admin Move Table - with an extra reorg

- Admin Move Table has the option to explicitly perform a reorg on target table during the swap step right before swap
- Only method to get an optimal compression dictionary for xml columns
- Set the REORG option at any point up to and including the SWAP phase
- Not recommended for „normal“ or every day use (e.g. to compact/compress data/remove gaps/sort etc. it is not needed)

Admin Move Table - to avoid reorg pending (1|3)

- Use admin move table for your DDL changes (change the target)
 - Drop / add column
 - Change data type (when types are compatible and column name persists)
 - Change nullable options
 - ~~Primary Key~~
- Option1
 - Create your target table in forehand with structure of next version (only table!)
- Option2
 - Use parameter coldef of admin move table to define future structure of the table

Admin Move Table - to avoid reorg pending (2 | 3)

```
[[db2inst1@db2_test ~]$ db2 "create table testschema.testtable (col1 int not null primary key, col2 int,col3 int,col4 char(50)) "
```

DB20000I The SQL command completed successfully.

```
[[db2inst1@db2_test ~]$ db2 "create index testschema.clus_idx on testschema.testtable (col2) cluster"
```

DB20000I The SQL command completed successfully.

```
[[db2inst1@db2_test ~]$ db2 "insert into testschema.testtable values (1,2,3,'FOUR')"
```

DB20000I The SQL command completed successfully.

```
[(reverse-i-search)`create': db2 "^Ceate index testschema.clus_idx on testschema.testtable (col2) cluster"
```

```
[[db2inst1@db2_test ~]$ db2 "create table testschema.testtable_new (col1 int not null primary key, col2 int,col4 char(50)) "
```

DB20000I The SQL command completed successfully.

```
[[db2inst1@db2_test ~]$ db2 "call SYSPROC.ADMIN_MOVE_TABLE('TESTSCHEMA','TESTTABLE','TESTTABLE_NEW','','MOVE')";
```

KEY	VALUE
AUTHID	DB2INST1
CLEANUP_END	2020-07-31-12.35.58.608997
CLEANUP_START	2020-07-31-12.35.58.512877
COPY_END	2020-07-31-12.35.58.142783
COPY_INDEXNAME	CLUS_IDX
COPY_INDEXSCHEMA	TESTSCHEMA
COPY_OPTS	OVER_INDEX,ARRAY_INSERT,CLUSTER_OVER_INDEX
COPY_START	2020-07-31-12.35.58.114566
COPY_TOTAL_ROWS	1
INDEXNAME	SQL200731123507230
INDEXSCHEMA	SYSIBM
INDEX_CREATION_TOTAL_TIME	0
INIT_END	2020-07-31-12.35.58.037016
INIT_START	2020-07-31-12.35.57.702908
ORIGINAL_TBLSIZE	512
PAR_COLDEF	using a supplied target table so COLDEF could be different
REPLAY_END	2020-07-31-12.35.58.437970
REPLAY_START	2020-07-31-12.35.58.143856
REPLAY_TOTAL_ROWS	0
REPLAY_TOTAL_TIME	0
STATUS	COMPLETE
SWAP_END	2020-07-31-12.35.58.468269
SWAP_RETRIES	0
SWAP_START	2020-07-31-12.35.58.460685
UTILITY_INVOCATION_ID	01000000B800000008000000000000000002073112355803812400000000
VERSION	11.05.0400

```
[db2inst1@db2_test ~]$ db2 "select * from testschema.testtable"
```

COL1	COL2	COL4
1	2	FOUR

Admin Move Table - to avoid reorg pending (3 | 3)

('TESTSCHEMA', 'TESTTABLE', "", "", "", "",
'col1 int not null primary key,
col2 int not null, col4 varchar(255)', '', 'MOVE')";

```
[db2inst1@db2_test ~]$ db2 "call SYSPROC.ADMIN_MOVE_TABLE('TESTSCHEMA','TESTTABLE','', '', '', '', '', '', '', 'col1 int not null primary key,col2 int not null, col4 varchar(255)', '', 'MOVE')";
```

Result set 1

KEY	VALUE
AUTHID	DB2INST1
CLEANUP_END	2020-07-31-12.26.49.964842
CLEANUP_START	2020-07-31-12.26.49.854156
COPY_END	2020-07-31-12.26.49.255086
COPY_INDEXNAME	CLUS_IDX
COPY_INDEXSCHEMA	TESTSCHEMA
COPY_OPTS	OVER_INDEX,ARRAY_INSERT,CLUSTER_OVER_INDEX
COPY_START	2020-07-31-12.26.49.225587
COPY_TOTAL_ROWS	1
INDEXNAME	SQL200731122629560
INDEXSCHEMA	SYSIBM
INDEX_CREATION_TOTAL_TIME	1
INIT_END	2020-07-31-12.26.49.132874
INIT_START	2020-07-31-12.26.48.647045
ORIGINAL_TBLSIZE	512
PAR_COLDEF	col1 int not null primary key,col2 int not null, col4 varchar(255)
REPLAY_END	2020-07-31-12.26.49.736263
REPLAY_START	2020-07-31-12.26.49.255982
REPLAY_TOTAL_ROWS	0
REPLAY_TOTAL_TIME	0
STATUS	COMPLETE
SWAP_END	2020-07-31-12.26.49.795169
SWAP_RETRIES	0
SWAP_START	2020-07-31-12.26.49.783091
UTILITY_INVOCATION_ID	01000000AE0000000800000000000000002020073112264913404900000000
VERSION	11.05.0400

26 record(s) selected.

Return Status = 0

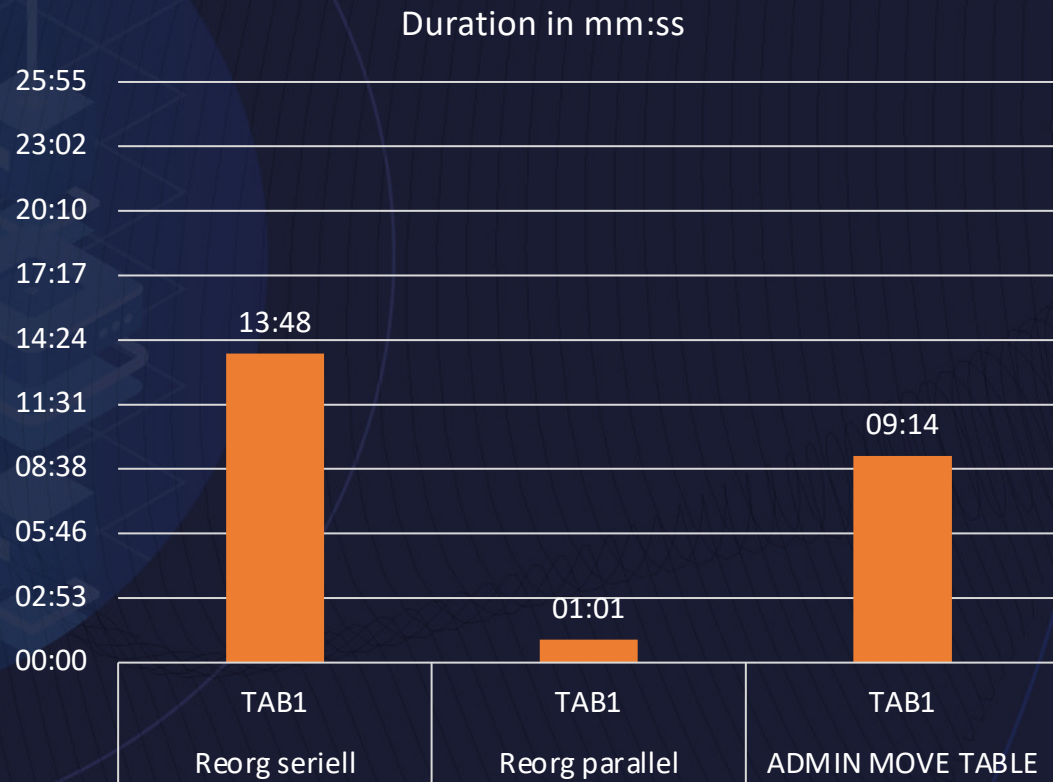
Performance Comparison - Background (1|2)

- Project goal: verify reorg performance for the Db2 LUW database
- Test-System
 - AWS EC2 instance of the r5.12xlarge type
 - 48 vCPU
 - 384 GiB Memory
 - 8 EBS Volumes with 1TB and limited to 10.000 IOPS for db2 data
- Test-Tables (partitioned)
 - Tab1 1.201.894 pages (32k) → ~36GB | 142.572.100 Rows
 - Tab2 698.711 pages (32k) → ~21GB | 98.459.199 Rows
- Roughly gone through 150 test steps in total

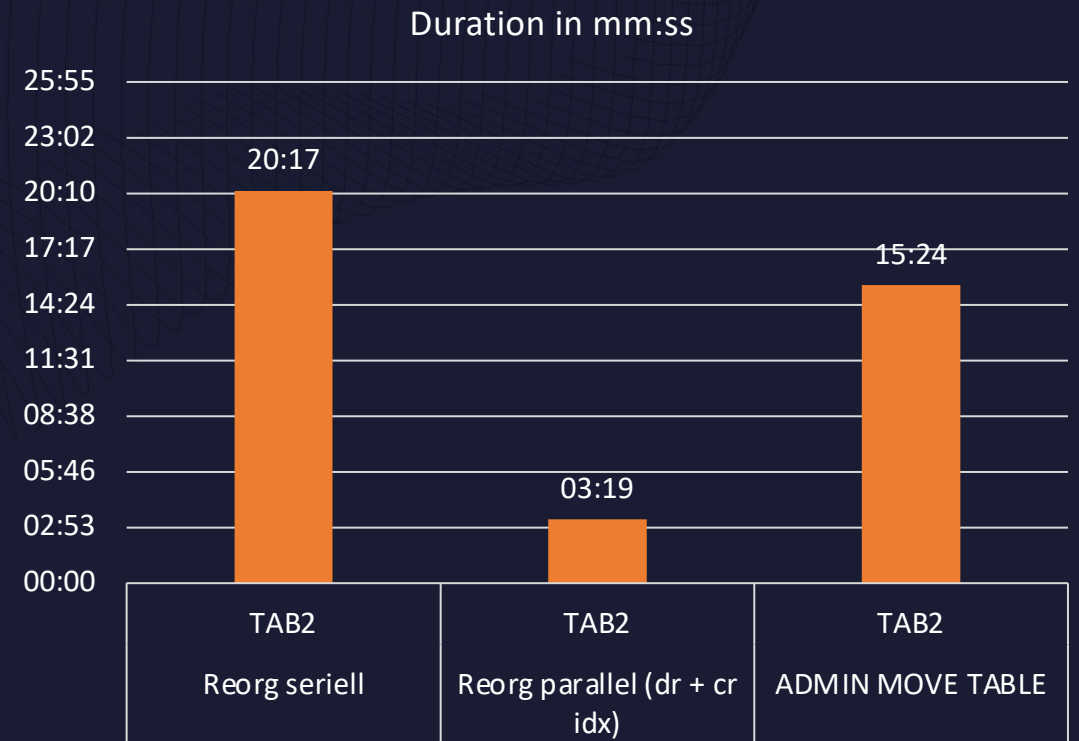
Performance Comparison - Background (2 | 2)

- Recommended final solution has to be simple and out-of-the-box
- Data had the size of a test-system first, increased over time to see production like runtimes
- Both tested tables are the biggest in production, are having a clustered index and are partitioned
- Tab1 has no nonpartitioned index
- Tab2 has some nonpartitioned index (→ drop/create approach)
- Goal: Achieve maximum resource consumption with high parallelism
- → Parallel: all 16 Partitions at once in parallel
- → Test: normal reorg vs. parallel (per partition) vs. admin move table

Performance Comparison - default vs. parallel reorg vs. admin_move_table

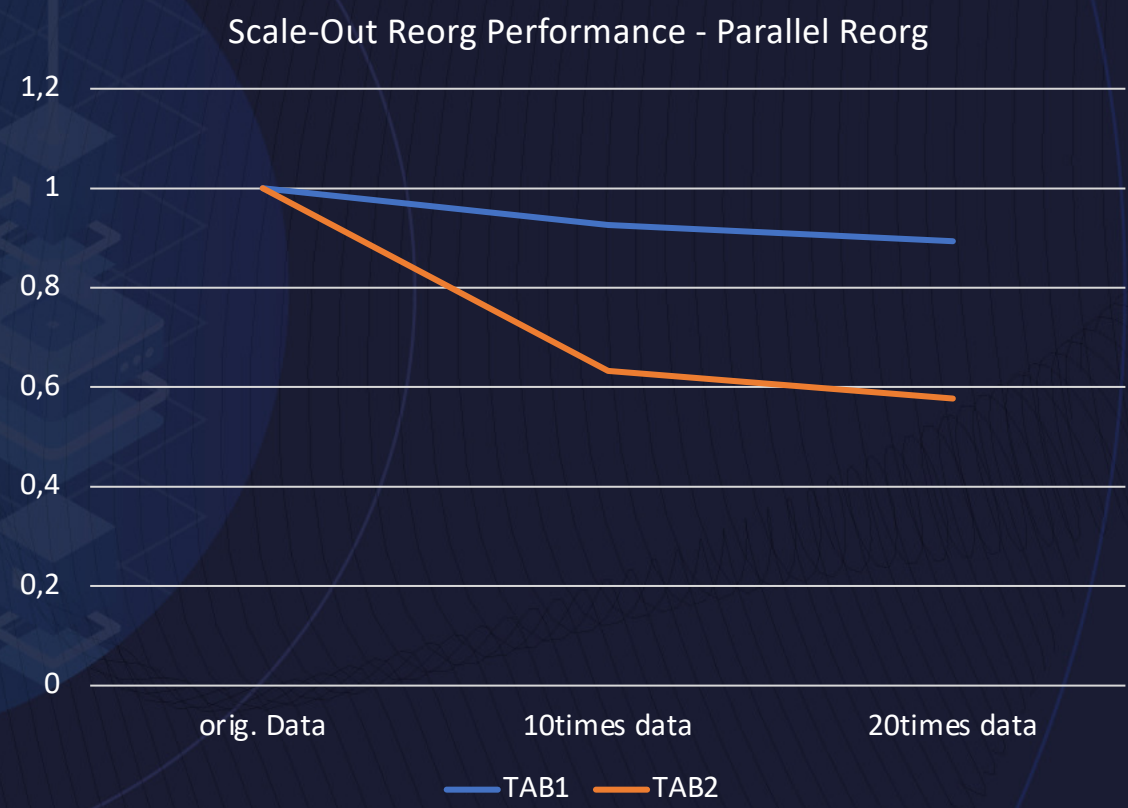


142Mio Rows, 36GB, all partitioned



98Mio Rows, 21GB, not all partitioned

Performance Comparison - scale-out parallel reorg



TAB1: 142Mio Rows, 36GB, all partitioned

Duration parallel reorg mm:ss

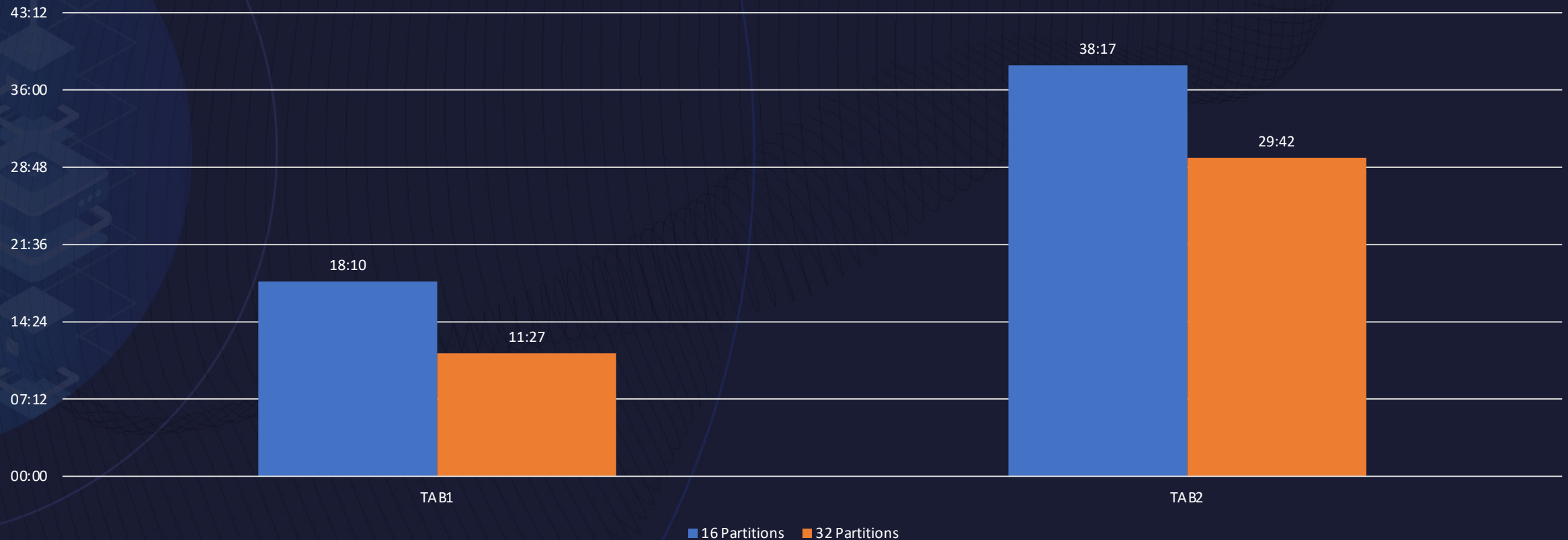
	orig. Data	10times data	20times data
TAB1	01:01	09:25	18:10
TAB2	03:19	20:58	38:17

It gets faster when data increases!

TAB2: 98Mio Rows, 21GB, not all partitioned

Performance Comparison - increase parallelism

Test 16 vs 32 Partitions (20times data) duration in mm:ss



142Mio Rows, 36GB, all partitioned

98Mio Rows, 21GB, not all partitioned

Summary

- If you need fast reorg performance, use range partitioned tables (with a well balanced distribution key) and the reorg utility in parallel
- Reorg as many partitions in parallel as possible (rule of thumb when activity is low: Number of Cores -1, else: number of “free” cores)
- Avoid nonpartitioned indexes (or drop/recreate those)
- Use Admin Move Table for most of your every day reorgs (reorg recommended / improve performance / remove gaps)
- Avoid Reorg Pending, use admin move table instead!

Sources

- <https://www.sap.com/documents/2015/07/3080d083-5b7c-0010-82c7-eda71af511fa.html>



IDUG DB2 Tech Conference
Prague, Czech Republic | November 2011

ADMIN_MOVE_TABLE() The New DBA Swiss Army Knife

Philip Nelson
Lloyds Banking Group / ScotDB Limited

Session Code: D1902



IDUG DB2 Tech Conference
Prague, Czech Republic | November 2011

Deep Dive in DB2 LUW offline table reorg and index reorg

Saeid Mohseni
Folksam Insurance Co.

Session Code: D11

Speaker: Markus Fraune

Company: ITGAIN

Email Address: markus.fraune@itgain.de

Twitter: [@maggus_f](https://twitter.com/maggus_f)

Don't forget to fill out a session evaluation!