

# SQLskills Immersion Event

## IEPTO1: Performance Tuning and Optimization

### Module 7: Index Fragmentation

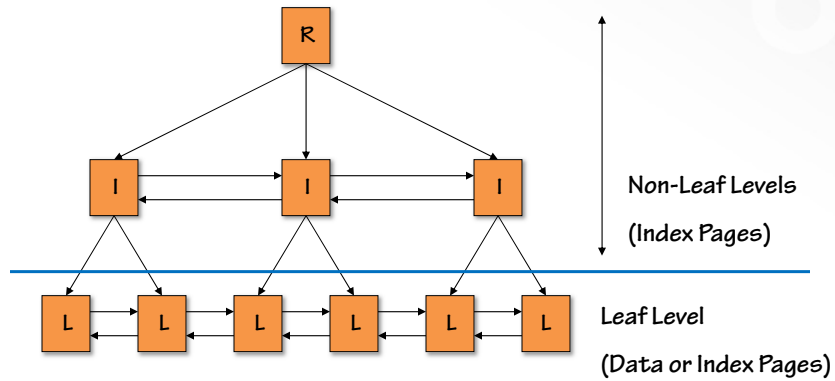
Paul S. Randal  
Paul@SQLskills.com



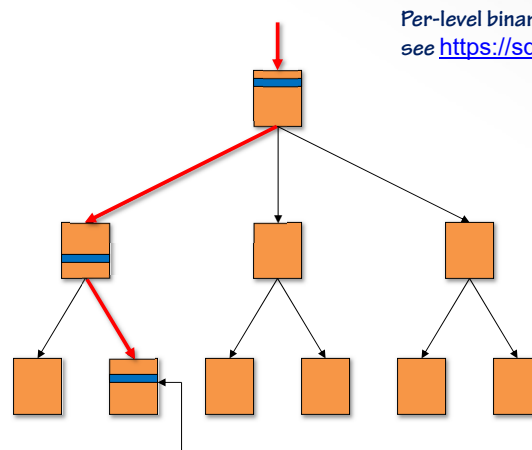
### Overview

- Data access methods
- What is index fragmentation?
- How does index fragmentation happen?
- Detecting index fragmentation
- Avoiding index fragmentation
- Removing index fragmentation
  
- Beware of people stating that fragmentation is not a problem any longer, or not a problem with SSDs
- Not true!

## Index Structure



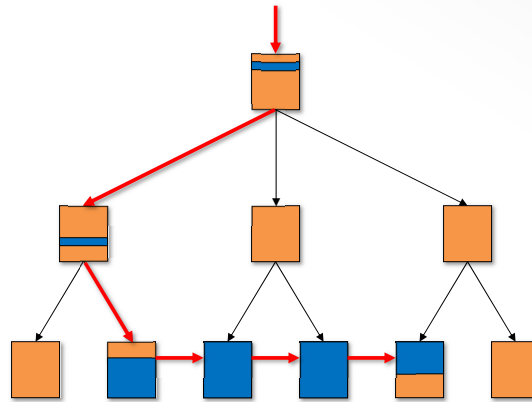
## Single-record Seek



Per-level binary search cost –  
see <https://sqlskills.com/p/068>

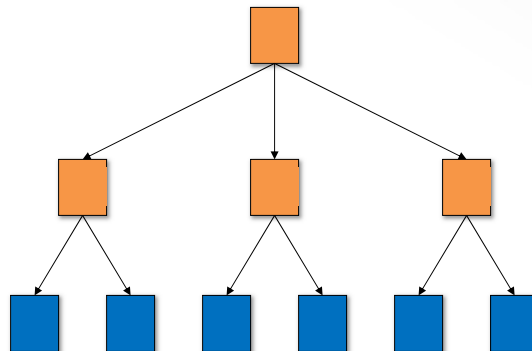
Matching record

## Multi-record Seek/Scan



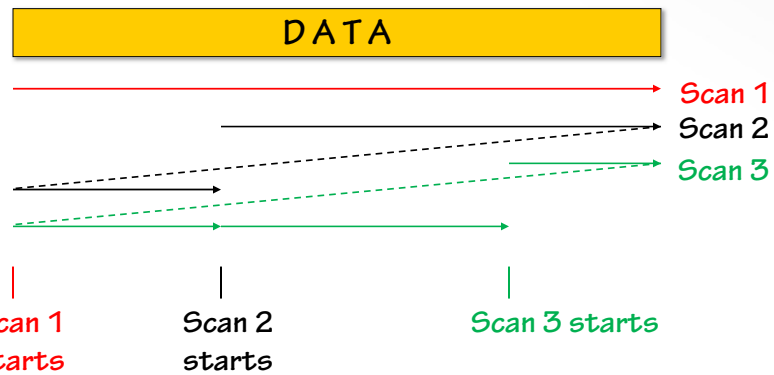
Matching records (in blue)

## Allocation Order Scan

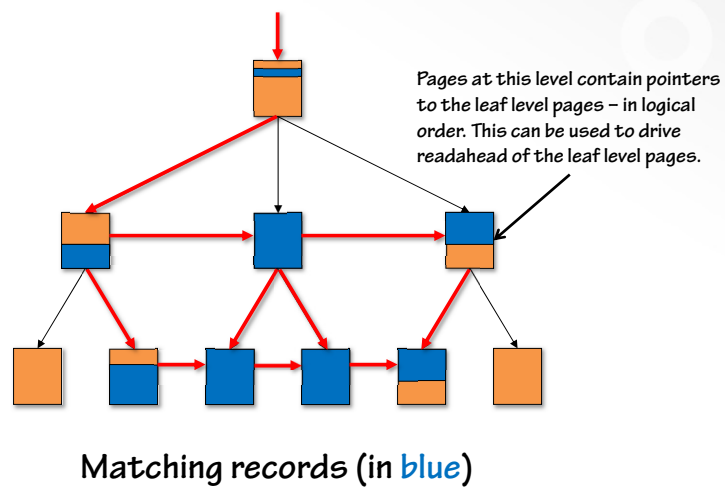


Matching records (in blue)

## Side Note: Merry-Go-Round Scans



## Readahead



## Readahead

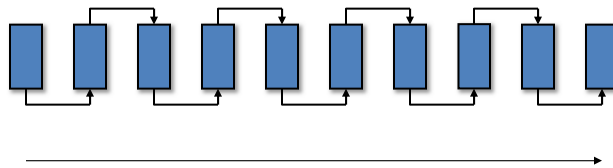
- **Why use readahead?**
  - Keep the CPUs busy, maximize throughput, avoid I/O waits
  - More efficient to issue 1 x 8-page read than 8 x 1-page reads
- **Feedback mechanism to avoid going too far ahead of scan point**
  - Maximum 1,000 pages ahead
- **Driven from parent level during scans**
  - Parent level pages contain logically-ordered links to the leaf level
- **Uses variable read sizes, up to 4MB read in 2016+**
  - Larger reads only possible with contiguous pages
  - Better contiguity = bigger reads = better performance
- **Possible to disable using trace flag 652**
- **Problem: fragmentation causes lower-performing scans**

## Overview

- Data access methods
- **What is index fragmentation?**
- How does index fragmentation happen?
- Detecting index fragmentation
- Avoiding index fragmentation
- Removing index fragmentation

## Fragmentation in Action

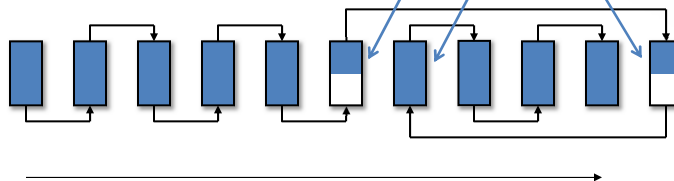
Index leaf level of newly built index



Long arrow is the allocation order  
Short arrows are following the logical order

## Fragmentation in Action

And now with fragmentation!




Long arrow is the allocation order  
Short arrows are following the logical order

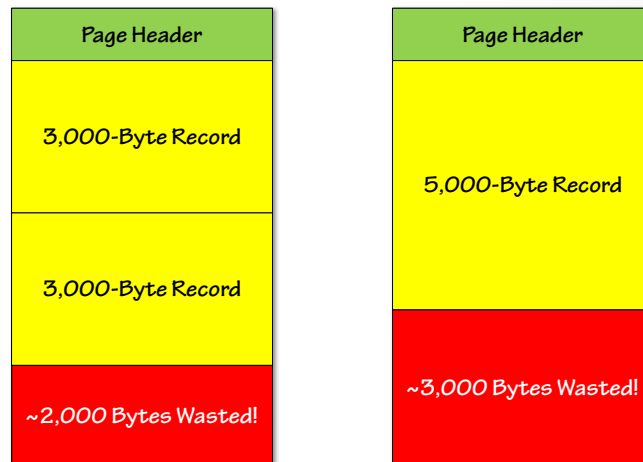
## Logical Fragmentation Defined

- (Sometimes called “external” fragmentation)
- Occurs when the next logical page is not the next physical page
- Prevents optimal readahead
  - Reduces seek/scan performance
- Does not affect pages that are already in cache
  - Smaller indexes cause less of a performance hit (e.g. 1-5000 pages or less)
- Reported as `avg_fragmentation_in_percent` for indexes in the `sys.dm_db_index_physical_stats` DMV
- **This is what most people consider ‘fragmentation’**
  - “Index fragmentation affects scan performance”
  - There is *\*so much more\** to it than that!

## Extent Fragmentation Defined

- Old concept, no longer reported for indexes
  - Occurs when the extents in an index are not contiguous
- 
- Also affects readahead performance but not as much
    - When writing the DMV for 2005, we decided to remove it to avoid confusion from too many measures of ‘fragmentation’
  - Reported as `avg_fragmentation_in_percent` in the `sys.dm_db_index_physical_stats` DMV for heaps ONLY
  - (2000: extent fragmentation algorithm in DBCC SHOWCONTIG is documented as not working for multiple files)

## Low Page Density in Action

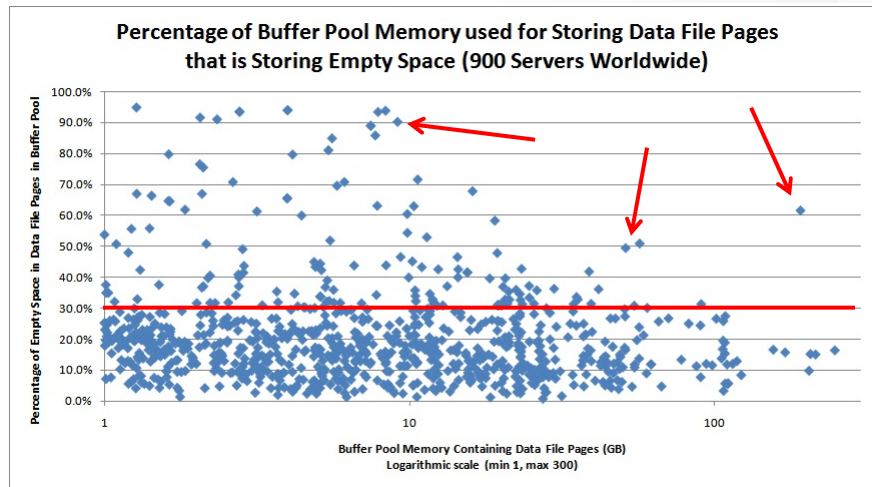


## Page Density Defined

- (Sometimes called “physical” or “internal” fragmentation)
- Page fullness is below the optimal level so lots of wasted space
- Effect is:
  - Increased disk space (more pages required to hold same number of rows)
  - Increased I/Os to read the same amount of data, leading to I/O subsystem pressure and overall performance degradation
  - Greater memory usage if most of the index is memory resident, leading to increased I/Os from \*other\* workloads, and so on...
  - More pages in the index unnecessarily can mean the Query Optimizer doesn't pick that index, leading to inefficient query plans
- This means ‘fragmentation’ can affect your performance even if you don't do index scans
- Hardware does not fix this
- Reported as avg\_page\_space\_used\_in\_percent in the DMV



## Increased Buffer Pool Usage



Source: my blog at <https://sqlskills.com/p/069>

## Overview

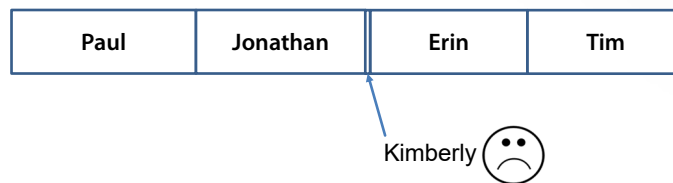
- Data access methods
- What is index fragmentation?
- **How does index fragmentation happen?**
- Detecting index fragmentation
- Avoiding index fragmentation
- Removing index fragmentation

## What Causes Fragmentation?

- **Schemas/workloads that cause page splits on full pages**
  - GUID as high-order key (or any other random key)
    - Can even affect nonclustered indexes
  - Updates to variable-length columns
  - Badly configured fill factor (more in a few slides)
- **Clustered index is likely the only one you can make the key not cause fragmentation by picking an ascending order key (e.g. bigint identity)**
- **Wide schemas that only fit a few records per page**
  - E.g. a fixed-size 5000 byte row = 3000 bytes lost per page!
- **Real-world example:**
  - Social networking site that has a homepage comments table with the member ID as the high-order key
  - Patient check-in company using GUID as clustering key

## Real-World Examples

- **MySpace**



- **Patient check-in company using GUID as clustering key**

## Can DML Cause Fragmentation?

- **Yes, data modifications can lead to fragmentation**
- **INSERT**
  - YES – if key value is not ever increasing/decreasing (e.g. GUID)
  - NO – if key is ever increasing/decreasing (e.g. INT IDENTITY)
- **UPDATE**
  - YES – if updates make variable-length columns wider on full pages
  - NO – if columns are fixed width or columns have 'place holder' values (i.e. DEFAULT values) to minimize row expansion on update
- **DELETE**
  - YES – if deletes are singleton deletes (Swiss-cheese problem – page density issues)
  - NO – if deletes are range deletes for archival purposes

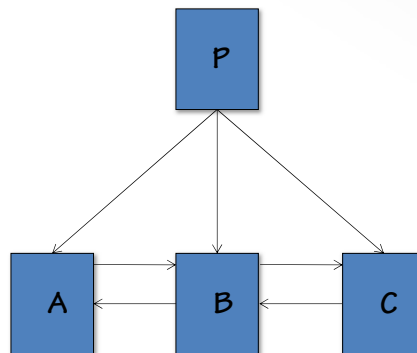
## What is a Page Split?

- **This is the primary cause of fragmentation, and is itself a performance problem when it occurs**
- **Occurs when a record must be inserted onto (or expanded on) a specific page in the index and there is not enough space**
  - Could be caused by a new record or an updated record that is now longer than it was before
  - Could also be caused by enabling snapshot isolation, which makes updated records 14-bytes longer
    - Also from enabling readable availability group secondaries in SQL Server 2012+
- **The page has to 'split' to make room**
  - Split point is usually as close to 50/50 as possible, but may be skewed if Storage Engine can determine an obvious split point

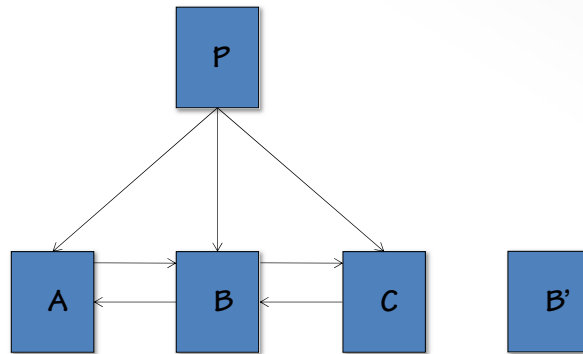
## Page Split Mechanism

- **For every page split:**
  - A new page is allocated to the index
  - All records after the split point are moved to the new page
  - New page is linked into the leaf level
  - A new record must be inserted into index level above the leaf
    - Could also cause a page split, cascading upwards to the root page
- **All steps are fully logged and performed by a system transaction**
  - **Very expensive, and hardware does not fix this!**
  - Detailed study of log records generated shown in demo towards end of Module 4 of the Pluralsight course *SQL Server: Logging, Recovery, and the Transaction Log*
- **After page split is committed, insert/update can take place**
- **Page split is never rolled back**

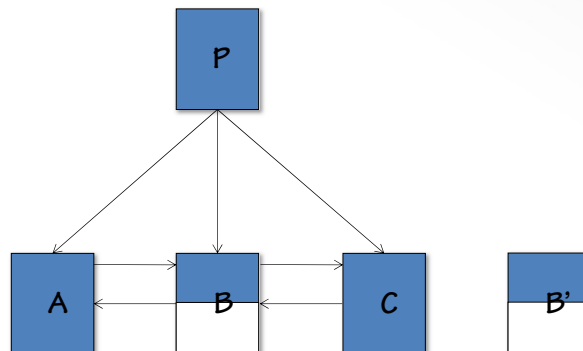
## Page Split Mechanism



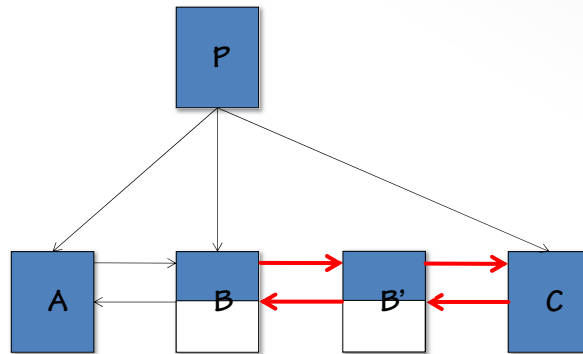
## Page Split Mechanism



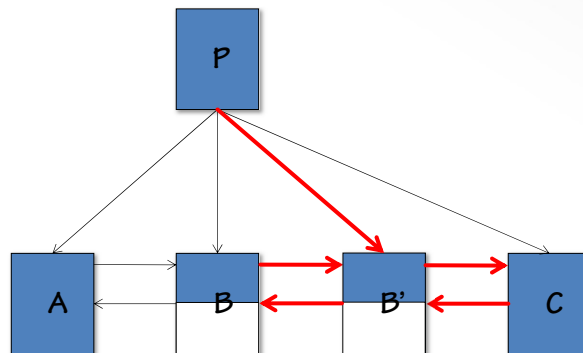
## Page Split Mechanism



## Page Split Mechanism



## Page Split Mechanism



## Demo

Increased logging during page splits

## Overview

- Data access methods
- What is index fragmentation?
- How does index fragmentation happen?
- **Detecting index fragmentation**
- Avoiding index fragmentation
- Removing index fragmentation

## Tracking Page Splits

- **There are 'good' and 'nasty' page splits...**
  - 'Good' split is when a page is allocated as part of an append-only insert pattern
  - 'Nasty' split is when a real page split occurs
- **Unfortunately, all documented methods of tracking page splits prior to SQL Server 2012 do not allow differentiation between 'good' and 'nasty' page splits**
  - Perfmon counter
  - sys.dm\_db\_index\_operational\_stats
  - Extended event (possibly with post-processing)
- **Either use log/log backup scanning or 2012+ Extended Events**
  - Both methods track the LOP\_DELETE\_SPLIT log record
  - See my blog post at <https://sqlskills.com/p/070>

## Symptoms of Fragmentation

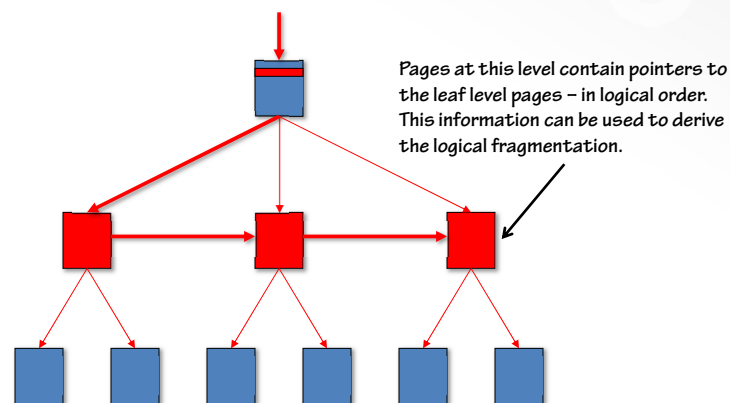
- **Poor/degrading query performance over time**
  - Longer run-times
  - More disk activity
    - SET STATISTICS IO ON
    - More frequent checkpoints occurring
  - Increased logging (from page split activity)
    - Depending on the average record length and the split point, a page split could log up to 50 times more than a regular insert!
  - Increased buffer pool usage
- **Worsening results from the sys.dm\_db\_index\_physical\_stats DMV**
  - Keys to success are knowing which indexes to look at and how to interpret the results



## sys.dm\_db\_index\_physical\_stats

- Replacement for DBCC SHOWCONTIG since SQL Server 2005
  - `select * from sys.dm_db_index_physical_stats (dbid, objectid, indexid, partitionid, samplemode)`
- No need to insert/exec to analyze/process DBCC SHOWCONTIG results
  - DMVs are programmatically “composable”
  - However, this is a DMF, not a true DMV so must do work for results
- Ability to control how much data is read using sample mode (LIMITED, SAMPLED, DETAILED)
  - LIMITED (default) does not read the leaf level so is fastest mode
    - This is good enough for most people
  - SAMPLED reads 1% of the leaf-level pages if the index/partition has more than 10000 pages
  - DETAILED reads everything and is the slowest mode

## How the LIMITED Scanning Mode Works



## Interpreting the DMV Output

- **What you need to look at:**
  - Logical fragmentation
    - avg\_fragmentation\_in\_percent (should be low)
  - Page density
    - avg\_page\_space\_used\_in\_percent
      - Should be high for data warehouse
      - Should have some free space for OLTP
  - Number of pages in the index
- **Other counters exist (e.g. fragments, avg. fragment size) but these were only invented to be more accessible to users – somewhat unsuccessfully**

## Demo

Detecting fragmentation using sys.dm\_db\_index\_physical\_stats

## Overview

- Data access methods
- What is index fragmentation?
- How does index fragmentation happen?
- Detecting index fragmentation
- **Avoiding index fragmentation**
- Removing index fragmentation

## How to Avoid Fragmentation?

- **Avoid 'random' index keys**
  - Almost impossible to do for nonclustered indexes
  - For clustered indexes, be careful about moving to (BIG)INT IDENTITY as small row size combined with many concurrent inserters could lead to an 'insert hotspot' performance issue
- **Implement index fill factors and periodically remove fragmentation**
  - Coming up next...
- **There is nothing you can do in hardware that means you can ignore index fragmentation**
  - Don't fall for the advice that SSDs mean you can ignore it
  - SSDs don't stop page splits, extra logging, wasted space, plan changes

## Contiguity When (Re)Building

- Consider using –E startup parameter for very large indexes that support very large scans
  - <http://support.microsoft.com/kb/329526>
  - During index build/rebuild (and all other operations):
    - SQL Server 2008+: 64 extents allocated before round-robin (4MB)
    - I.e. 64 single-extent allocations, not one 64-extent allocation
    - Combine with large RAID stripe size
- For best contiguity and readahead I/O size, use MAXDOP = 1 when building or rebuilding indexes
  - Otherwise multiple (re)build threads building the leaf level, leading to extent interleaving (essentially extent fragmentation), and reduced readahead
- Note: this is not relevant for OLTP systems

## Fill Factors

- Setting a fill factor makes the Storage Engine leave space on each leaf-level page to allow inserts/expansions to not cause page splits
- Specified at index creation or rebuild time
  - NOT maintained during regular DML
- Use during index create/rebuild/reorganize
- Can specify with sp\_configure for entire instance
  - Not recommended – specify it per index
- Use PAD\_INDEX to use fill factor for upper levels of the index
  - Rarely used
- 0 = 100 = default value with special meaning of ‘leave no space’
  - Excellent for data warehouse, but not ideal for OLTP
- For OLTP, which value to use?

## Picking a Fill Factor to Use

- **Balancing act between how often page splits occur and how often you can rebuild/defrag the index**
- **What is going to cause page splits in your schema?**
  - UPDATES to variable-width data types?
  - Random INSERTs?
    - The more volatile ⇒ lower FILLFACTOR
- **How often can you rebuild/defrag?**
  - The more frequent ⇒ higher FILLFACTOR
- **Pick a value, try it, monitor fragmentation, tweak it**
  - Use DMVs to see how fast the fragmentation increases
  - The faster fragmentation occurs ⇒ lower FILLFACTOR or decreased time between rebuilds/defrag
  - 70% or 80% are common first guesses

## Setting a Fill Factor

- **Can be set when creating or rebuilding an index**
  - Stores the fill factor in the index metadata
- **Can also be set using Object Explorer in SSMS**
- **Cannot be set directly with ALTER INDEX ... REORGANIZE**
- **REBUILD and REORGANIZE use the metadata-stored fill factor, if there is one, otherwise they will use the instance-wide fill factor**
  - Unless a fill factor is specified on the REBUILD
  - I.e. REBUILD-specified fill factor overrides metadata-stored fill factor, which overrides instance-wide fill factor

## Additional: Are Your Indexes Being Used?

- There are lots of bad practices around index strategy, including creating extra indexes
  - E.g. an index for each column in the table
- Extra, unused indexes waste resources as they must be maintained by DML operations
- Use the `sys.dm_db_index_usage_stats` DMV to tell if an index is being used at all during the business cycle
  - Beware of indexes not being used but enforcing unique constraints
  - Beware that in 2012 and 2014 the stats are reset for indexes rebuilt online
    - Fixed in SQL Server 2016+, and latest builds of 2012 and 2014

## Overview

- Data access methods
- What is index fragmentation?
- How does index fragmentation happen?
- Detecting index fragmentation
- Avoiding index fragmentation
- Removing index fragmentation

## How to Remove Fragmentation?

- **2 realistic choices**
  - Rebuild the index: ALTER INDEX ... REBUILD
    - Create a brand new index structure
  - Reorganize the index: ALTER INDEX ... REORGANIZE
    - Shuffle the existing pages allocated to the index
- **Also CREATE INDEX ... WITH (DROP\_EXISTING = ON)**
  - Commonly used to move or (re)partition an index
- **Can also choose not to remove fragmentation**
  - If the index isn't used for scans, and page density isn't an issue, why spend the resources?
- **Don't just rebuild all indexes every day**
- **Synchronous mirroring or AGs may force REORGANIZE to be used**

## ALTER INDEX ... REBUILD

- **Pros**
  - Can use multiple CPUs, and control MAXDOP (lower DOP = better contiguity)
  - Rebuilds index statistics (with equivalent of full scan, or sampled if partitioned index)
  - Can rebuild a single partition (online from 2014) or all partitions (online from 2005)
  - Can be performed online
    - 2012+: Indexes with **non-legacy** LOB columns (plus clustered index on table with **non-legacy LOB/FILESTREAM** column)
    - 2017+: ability to pause and resume an online-index rebuild, resume starts from last position
  - Can be minimally-logged (but log backup will be the same size)
  - SORT\_IN\_TEMPDB reduces logging + perf boost in 2014+ (<https://sqlskills.com/p/071>)
    - Not available with resumable online index rebuild
- **Cons**
  - Atomic operation – potentially long rollback on interrupt, all or nothing semantics
  - Must create new index before dropping old one, up to 125% extra space required
  - When offline – SCH-M table lock for nonclustered or clustered index rebuild
  - When online – blocking potential, but can be resolved in SQL 2014 onward
    - Resumable online rebuild of clustered with LOB columns = SCH-M table lock for duration!

## ALTER INDEX ... REORGANIZE

- Replaced DBCC INDEXDEFRAG in SQL Server 2005 onward
- **Pros**
  - ALWAYS online – only requires table IX lock
  - Interruptible with no loss of work – stops instantly
  - Has progress reporting in sys.dm\_exec\_requests / percent\_complete
  - Compacts LOB storage (on by default, see <https://sqlskills.com/p/072> for bug fixes)
  - Usually faster for a lightly fragmented index
  - Can reorganize one or all partitions
  - Does not require any extra disk space
  - In SQL Server 2016+, works on columnstore indexes too (i.e. online columnstore ops)
- **Cons**
  - Usually slower for a heavily fragmented index
  - Always fully-logged, single CPU only, does not update statistics
  - Does not do as good a job as removing fragmentation
  - Does not increase free space on pages!! (so may be better with a rebuild)
  - Possible problem with cached query plans if # of pages drastically changes



47

© SQLskills, All rights reserved.  
<https://www.sqlskills.com>

## CREATE INDEX ... WITH (DROP\_EXISTING=ON)

- Don't use this if you just want to rebuild the index with no changes
- **Pros**
  - Same as ALTER INDEX ... REBUILD
  - Can move the index to a new location
  - Can rebuild the index with a new partitioning scheme
  - Can change the index schema (keys, sort order, etc)
  - Can do all of this online (with same limitations as regular index rebuild)
- **Cons**
  - Same as ALTER INDEX ... REBUILD
  - Need to know the index schema



48

© SQLskills, All rights reserved.  
<https://www.sqlskills.com>



## Comparison Points: REBUILD vs. REORGANIZE

- **Space required**
  - This may force you to do REORGANIZE
- **Log generated**
  - This may force you to do 'staggered index maintenance' using REORGANIZE
- **Algorithm speed on amount of fragmentation**
- **Lots of pages above fill factor? Possibly REBUILD**
- **Locks required (i.e. online or not)**
  - This may force you to do REORGANIZE
- **Interruptible or not**
- **Progress reporting or not**

## When To Rebuild vs. Reorganize

- **Much debate on this, basically it depends!**
- **I had to come up with numbers for Books Online so I chose:**
  - < 5-10% do nothing
  - 5-10% <> 30% defrag/reorganize
  - 30%+ rebuild
  - And don't do anything if the index has < 1-5000 pages
- **Your mileage may (and will) vary**

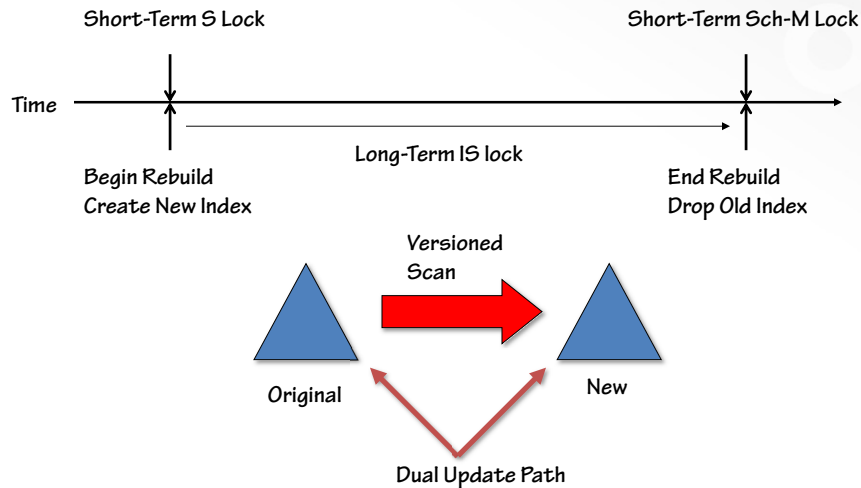
## Demo

Removing fragmentation and index rebuild options

## Paul's Method...

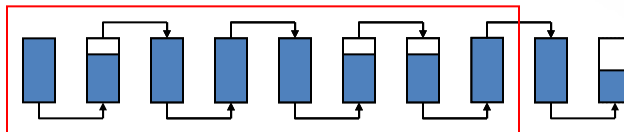
- Create a table with one row per index you want to work on
  - I call it the 'driver table'
- Call the DMV for the indexes listed in the driver table
- Use per-index fragmentation thresholds to determine whether to rebuild, reorganize, or do nothing
- Log what you decide to do for future reference
- Optional: keep a counter of how many times in succession an index is rebuilt and programmatically reduce fill factor
- Much easier: use code someone's already written...
  - <http://ola.hallengren.com> – the gold standard

## Inside Online Index Operations



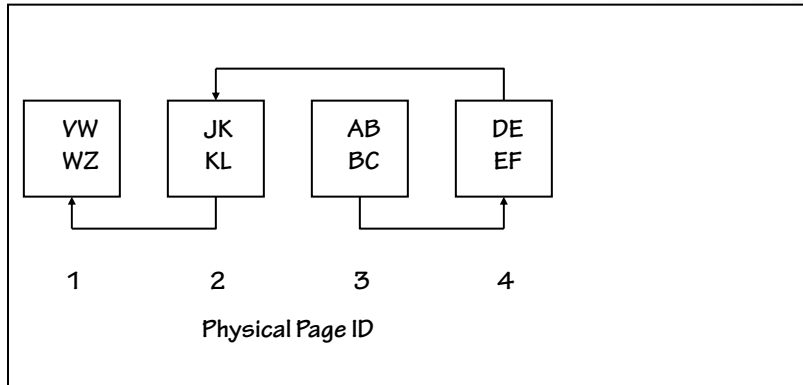
## Inside REORGANIZE: Phase One

- Uses a 'sliding window' compaction algorithm
- Deletes ghosted rows

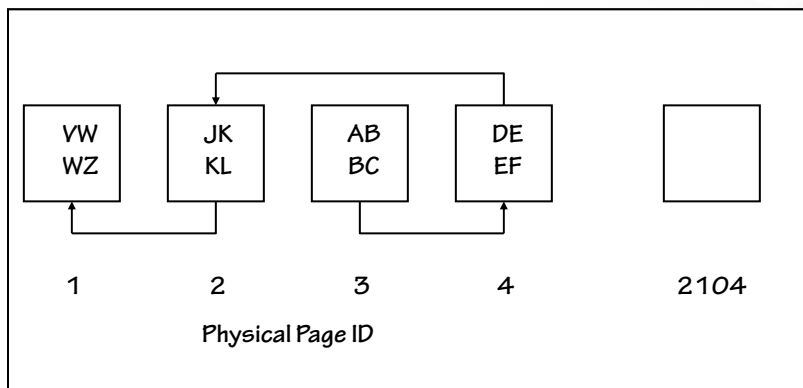


- This algorithm only compacts if enough space over 8-pages to remove one page
  - Earlier algorithm from DBCC INDEXDEFRAG in SQL Server 2000 ran into pathological cases with some applications

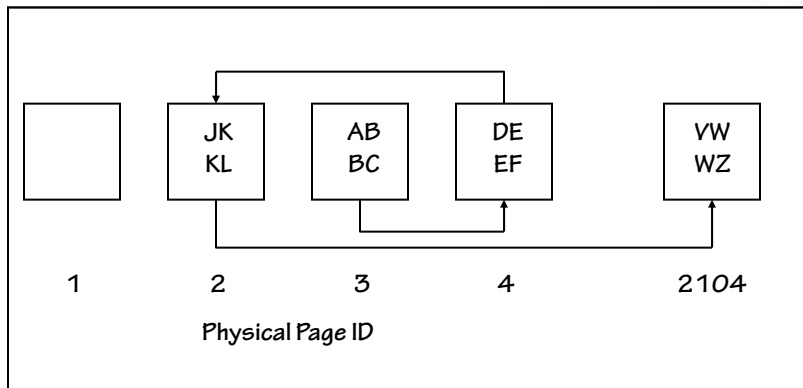
## Inside REORGANIZE: Phase Two



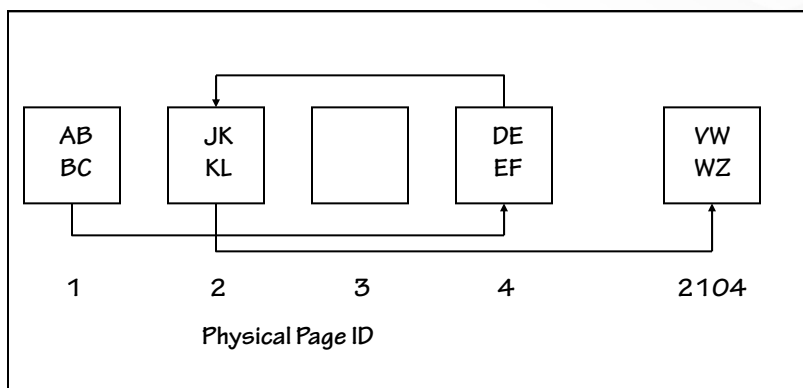
## Inside REORGANIZE: Phase Two



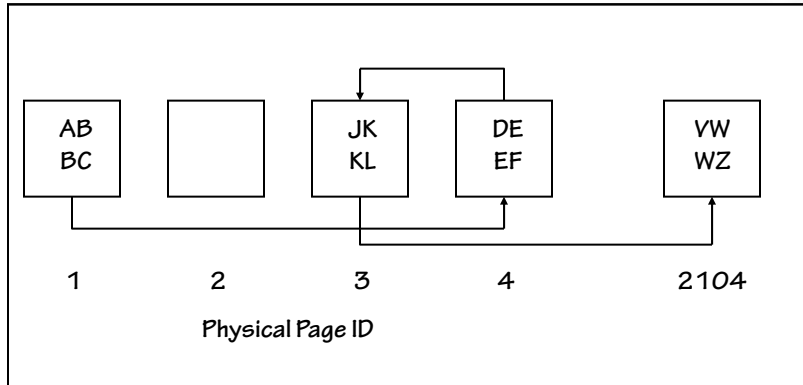
## Inside REORGANIZE: Phase Two



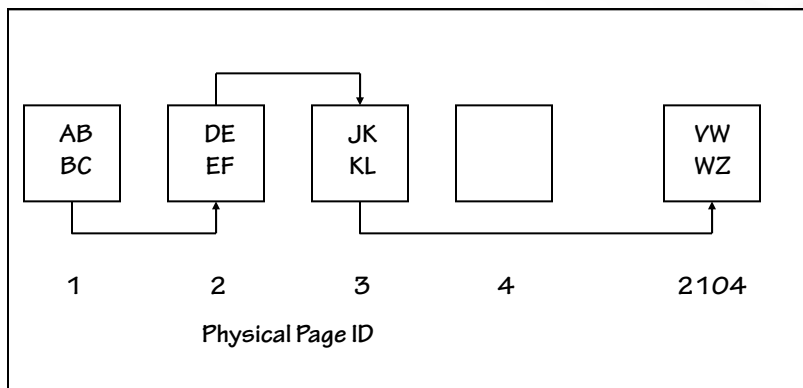
## Inside REORGANIZE: Phase Two



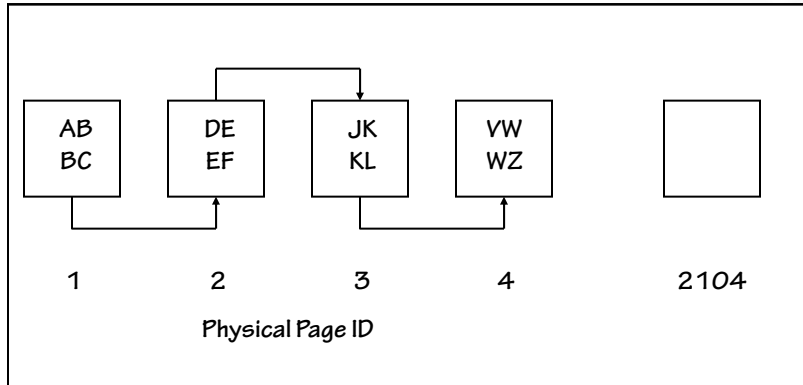
## Inside REORGANIZE: Phase Two



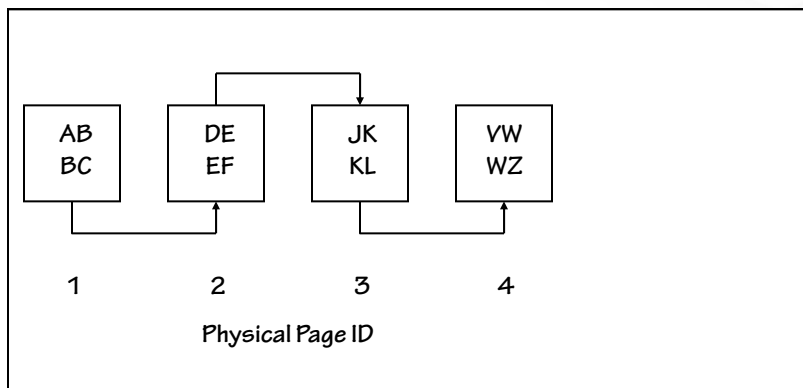
## Inside REORGANIZE: Phase Two



## Inside REORGANIZE: Phase Two



## Inside REORGANIZE: Phase Two



## Key Takeaways

- As you can see, fragmentation is very expensive when it happens
- Many people say not to bother about fragmentation
  - They're WRONG!
  - Lots of wasted storage space and extra I/Os
  - Lots of wasted buffer pool memory
  - Lots of extra log to back up, ship, mirror, scan...
  - Performance hit of the page splits happening
- Still a problem even when using SSDs
  - SSDs don't stop fragmentation from happening
- Set appropriate fill factors for indexes that get heavily fragmented
  - Start with FILLFACTOR = 70 and tweak as needed
- Consider changing index keys (carefully)

## Resources

- My blog category on index fragmentation
  - <https://sqlskills.com/p/076>
- Pluralsight course
  - <https://sqlskills.com/p/074>
- Free index maintenance (and more!) tool
  - <http://ola.hallengren.com/>
- WP: Microsoft SQL Server 2000 Index Defragmentation Best Practices
  - <https://sqlskills.com/p/073>
  - Based on SQL Server 2000, so discusses DBREINDEX vs. INDEXDEFRAG
- WP: Online Indexing Operations in SQL Server 2005
  - <https://sqlskills.com/p/075>



## Review

- Data access methods
- What is index fragmentation?
- How does index fragmentation happen?
- Detecting index fragmentation
- Avoiding index fragmentation
- Removing index fragmentation

# Questions!