

The background of the slide is a deep blue with a high-contrast image of a water droplet hitting a surface, creating concentric ripples that spread outwards. The droplet is captured at the moment of impact, appearing as a clear, spherical bubble in the center of the ripples.

Module 3

Backup Strategy and Internals

SQL
skills

B/R

Online

non det. time of
restore

Only get "used"
Space

Rolling/online/upgrade

Detach

Offline

attach is exactly
when taken offline

potentially lots of
free space to move

Update stats?

update stats
TDE? neither will compress well

This was from a side discussion on backup/restore vs. detach/attach. The most important two items (IMO) for why B/R is better – you have a backup ☺ and you can do this without taking the database offline. Be careful using detach/attach as it might be a one way street.

This is a pretty good pros/cons list: <http://strictlysql.blogspot.com/2011/10/backuprestore-vs-detach-attach.html> and Aaron Bertrand also did an interesting post here:

http://sqlblog.com/blogs/aaron_bertrand/archive/2011/01/21/downgrading-a-database-you-can-t-get-there-from-here.aspx.

Backward Compatibility Levels

The following drawings discussed our tangent about backward compatibility levels.

The key points:

- The compat mode of the database when backed up is what will get restored. If the database compatibility mode is set to 6.5 (for example) on a 7.0 server and then backed up – the database's compat mode is 6.5. If this backup (a 7.0 database backup) is restored on SQL Server 2000 (8.0) then it will be restored and the database itself will be converted to an 8.0 database. However, the database compatibility mode will still be 6.5.
- Compat modes do not impact features, they impact parsing and keyword use. They are meant as a way to allow you to upgrade to the new release while having time to “fix” code – to remove keywords, etc.
- Setting compat modes can be done through the UI or the syntax. Often, the UI is limited in the modes available. For example, even on SQL Server 2005 (9.0) you can still set 6.5 and 6.5 – but not through the UI. Always check the syntax for setting compat modes:
 - Use `ALTER DATABASE` to set compatibility mode
 - For backward compatibility, you can still use `sp_dbcmtlevel`.

Compat

2008

B/R

2008 R2 / 2012

Compat. 2005

→ doesn't change

Upgrade advisor

Upgrade assistant

trace trace

=

Tools to help you upgrade: ?

Microsoft Upgrade Advisor: (found within the feature pack) <http://www.microsoft.com/en-us/download/details.aspx?id=29065>

Upgrade Assistant Tool for SQL Server 2012 (en-US):

<http://social.technet.microsoft.com/wiki/contents/articles/2558.upgrade-assistant-tool-for-sql-server-2012-en-us.aspx>

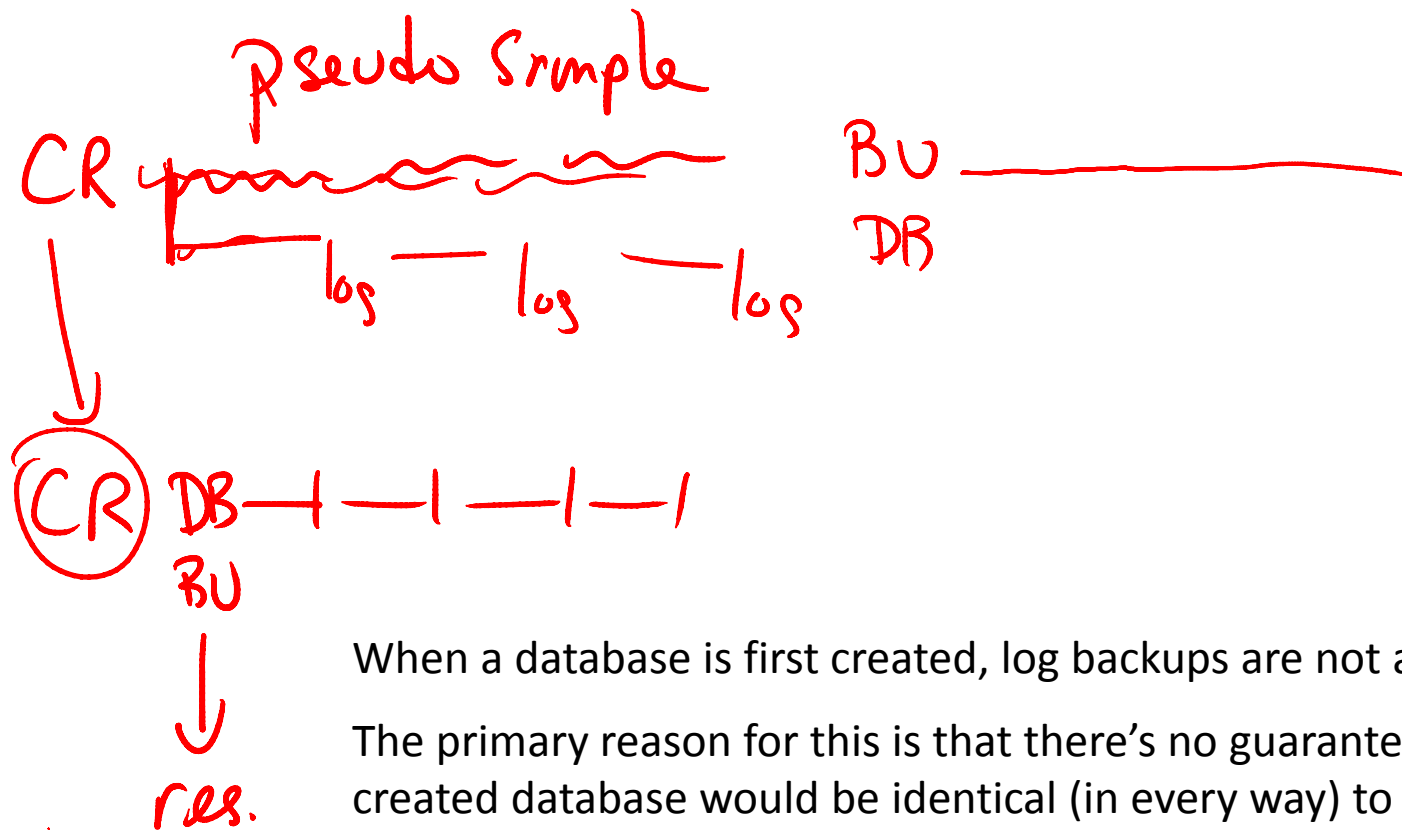
Backup/restore is supported “up-level” only and only 2 versions. However, 2008 R2 is not counted in the two version restriction. For example, you can restore from:

- 2000 to 2005/2008/2008R2
- 2005 to 2008/2008R2/2012

But, not from 2000 to 2012.

The compat mode of the backed up database is the compat mode that’s restored (unless no longer supported).

From SQL 2012 BOL: When a database is upgraded to SQL Server 2012 from any earlier version of SQL Server, the database retains its existing compatibility level **if it is at least 90**. Upgrading a database with a compatibility level below 90 sets the database to compatibility level 90. <http://technet.microsoft.com/en-us/library/bb510680.aspx>



When a database is first created, log backups are not allowed.

The primary reason for this is that there's no guarantee that the newly created database would be identical (in every way) to the database as it was originally created. So, when a database is created it runs in the pseudo-simple recovery model. This is where the log is cleared on checkpoint. There's no reason to keep it, it cannot be backed up.

However, once you do your first database backup, the full recovery model can be leveraged and therefore the log no longer clears...

This is the reason that some folks end up with a GIANT transaction log when they've only been performing full backups.

10GB DATA \ HUGE LOG (961GB)

In Full what to do?

if BU 1TB?

Switch to simple (clears the log)
— [Full backup (only backs up data not free space)
Switch back Full?

Fix log size (shrinkfile)

BU 10GB \ 20GB log

What do you do if you have a transaction log that's grown wildly out of control?

Switch to simple to clear the log

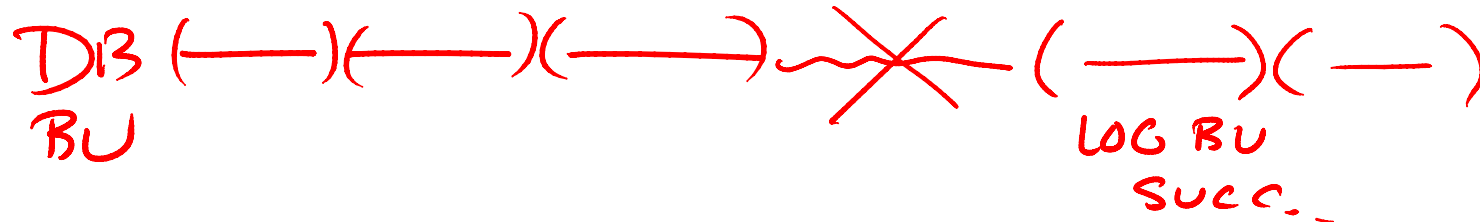
Perform a full database backup

Consider switching back to FULL (if you're going to perform log backups) OR stay in SIMPLE.

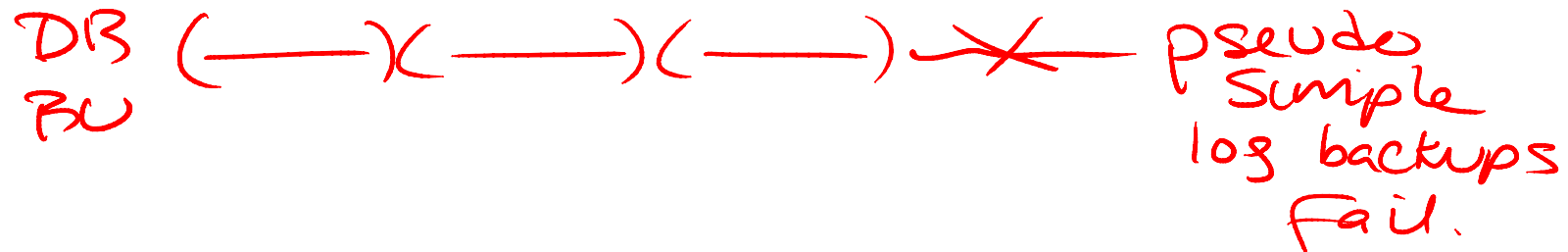
Fix the log size (DBCC SHRINKFILE)

Do another backup? Or, at least a log backup (if you're in FULL)

2000



2005



SQL Server 2000 allows the transaction log to be backed up even after the continuity of the log has been broken... Don't let this happen use trace flag 3231 to disable NO_LOG and TRUNCATE_ONLY for the log.

SQL Server 2005 sets the database back to a pseudo-simple recovery model after the continuity of the log has been broken. But, they still ALLOW the continuity of the log to be broken. Use trace flag 3231 OR 3031 to stop this from happening.

SQL Server 2008 does not allow manual clearing of the log. The NO_LOG and TRUNCATE_ONLY syntax has been deprecated.



Here we were discussing how log backups can occur concurrently with full database backups. They can even run numerous times during a single full. Each log backup will run but it will NOT clear the log. When the backup completes the data copy phase, the necessary log information will be backed up as well. This will allow the no-longer-needed log information to be cleared but it is not the full backup that caused the log to clear – it's the log backups that have already been done!

Also, if a full backup becomes damaged, SQL Server can use the logs as if the full backup had never occurred. The restore process would be:

- Use the most-recent (but working 😊) full database backup

- Use the last differential based on that full backup

- Use any logs from the last differential and until current (up-to-the-minute)

VLF Fragmentation

We briefly discussed VLFs (covered in IE1), here are the relevant links:

- Kimberly’s blog post *“8 Steps to better Transaction Log throughput”*
 - <http://www.sqlskills.com/blogs/kimberly/post/8-Steps-to-better-Transaction-Log-throughput.aspx> (<http://bit.ly/67DFtV>)
- Beware of exactly 4GB growth/auto-growth bug
 - <http://www.sqlskills.com/BLOGS/PAUL/post/Bug-log-file-growth-broken-for-multiples-of-4GB.aspx> (<http://bit.ly/c3QQLr>)
 - Fixed in 2008 R2: <http://support.microsoft.com/kb/2633151>

2000

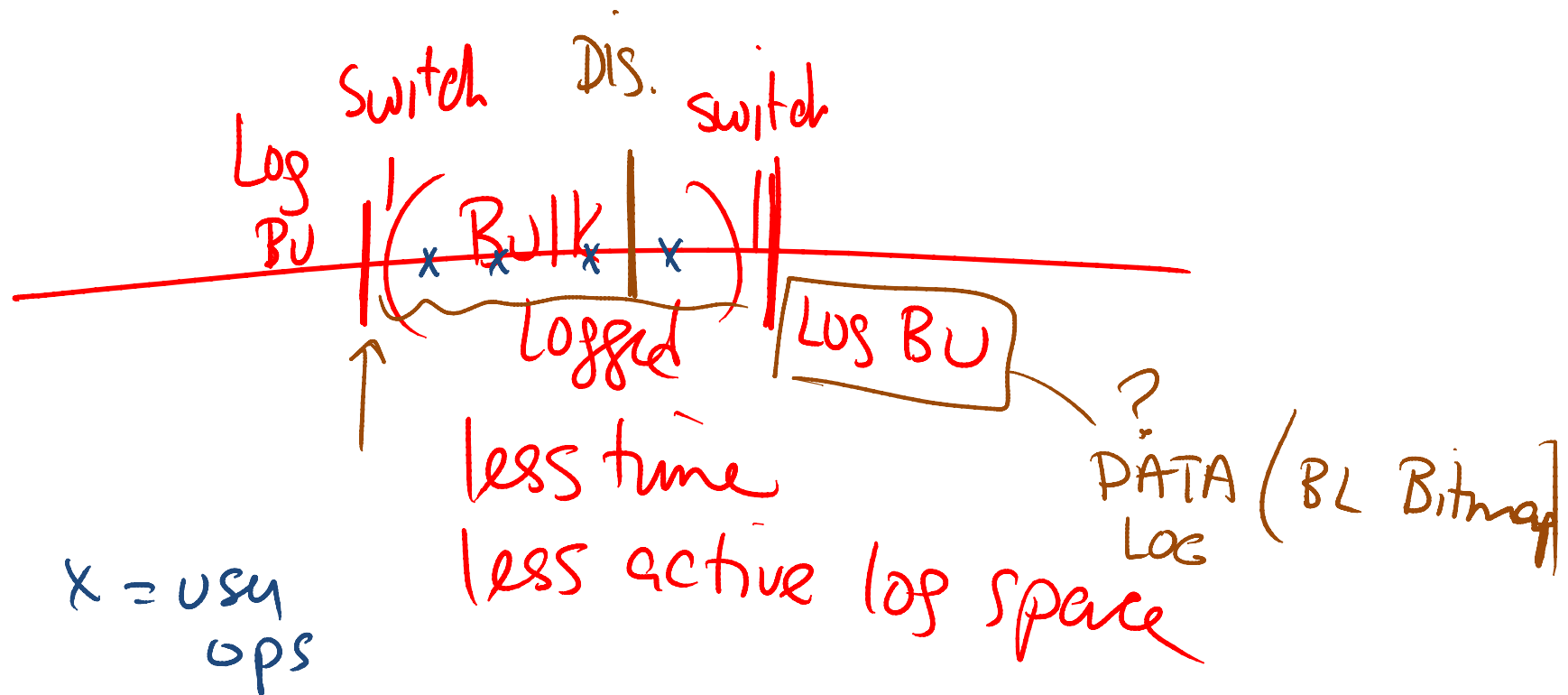
~~NO RECOVERY~~
=

file

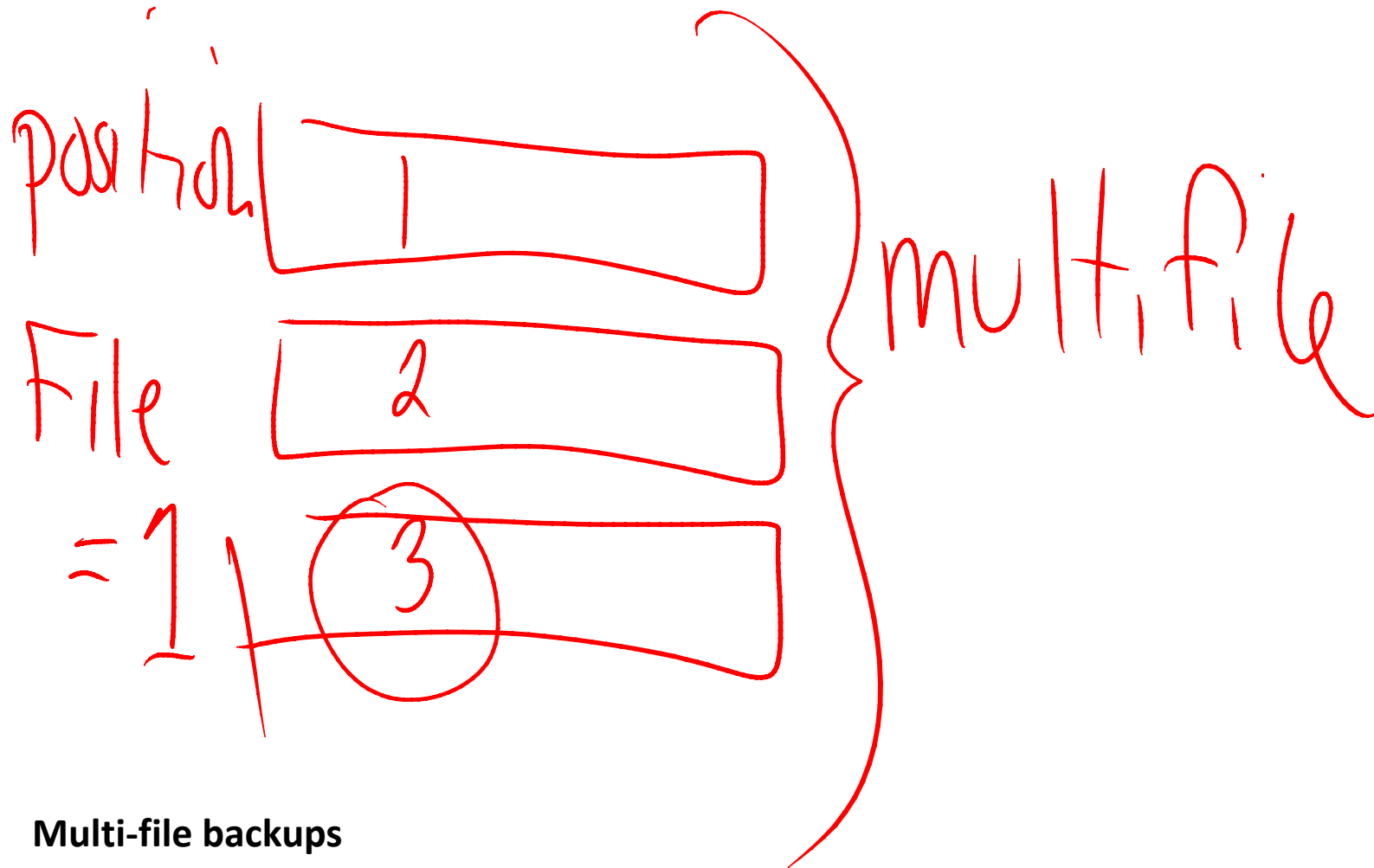


LSN
MIN

In SQL Server 2000, log backups could NOT occur concurrently with full database backups. However, log backups COULD occur concurrently with FILE or FILEGROUP backups. As a result, you could replace your full database backup strategy with FILEGROUP backups (and you have to backup ALL files/filegroups) and never have to stop your log backup job. This also meant that your secondary sites would NOT fall far behind (because log backups would NOT be blocked during FILE/FILEGROUP backups).



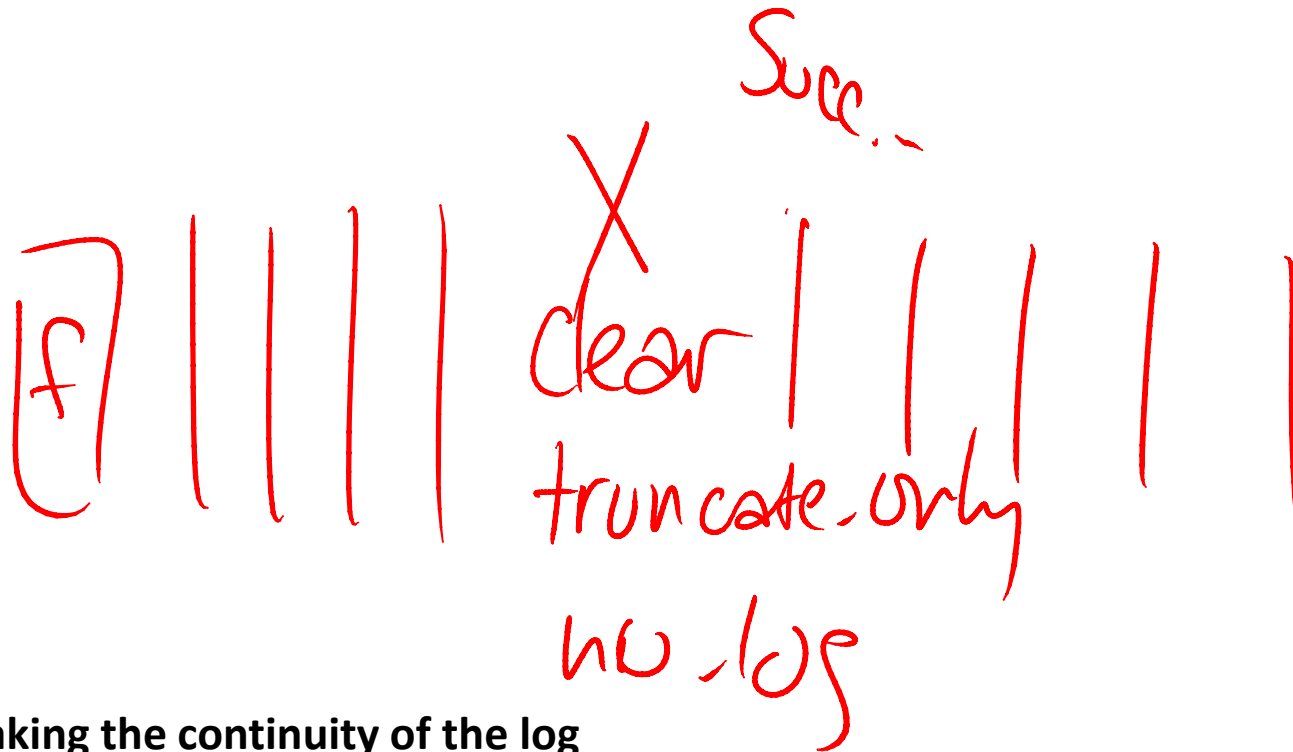
If you plan to use the BULK_LOGGED recovery model during certain batch/bulk operations, you should be sure to minimize the potential data loss by performing log backups before the change TO bulk_logged as well as AFTER the change back to the FULL recovery model. Remember, if a disaster were to occur while SQL Server is in BULK_LOGGED you cannot access the tail of the log (if any bulk operations have occurred). This is what you would lose if there were a disaster during the batch process.



Multi-file backups

Are multiple backups to one or more devices (known as a media set). Once a media set has been defined, all backups to that media set must match in number as well as compression setting. If you want to change compression or use a different number of backup devices – you must use the WITH FORMAT option. When specified the media set is overwritten and any remaining “fragments” are completely and totally unusable.

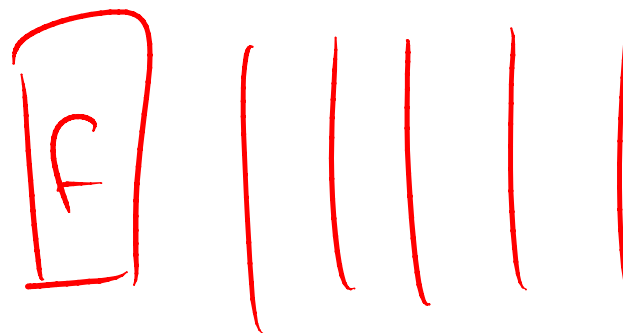
2000



Breaking the continuity of the log

In SQL Server 2000, you can continue to successfully perform log backups even after the continuity of the log has been broken (YIKES!). To stop this possibility of this happening when someone uses `BACKUP LOG WITH TRUNCATE_ONLY` or `NO_LOG`, use trace flag 3231 (see the slides for more details). These commands are turned into a NOOP (null operation) in SQL Server 2000 and 2005 with 3231. On SQL Server 2005, 3230 turns them into a CHECKPOINT (which has no [negative] effect on databases in the FULL or BULK_LOGGED recovery models).

2005 +



fail fail fail
Simple
log
X



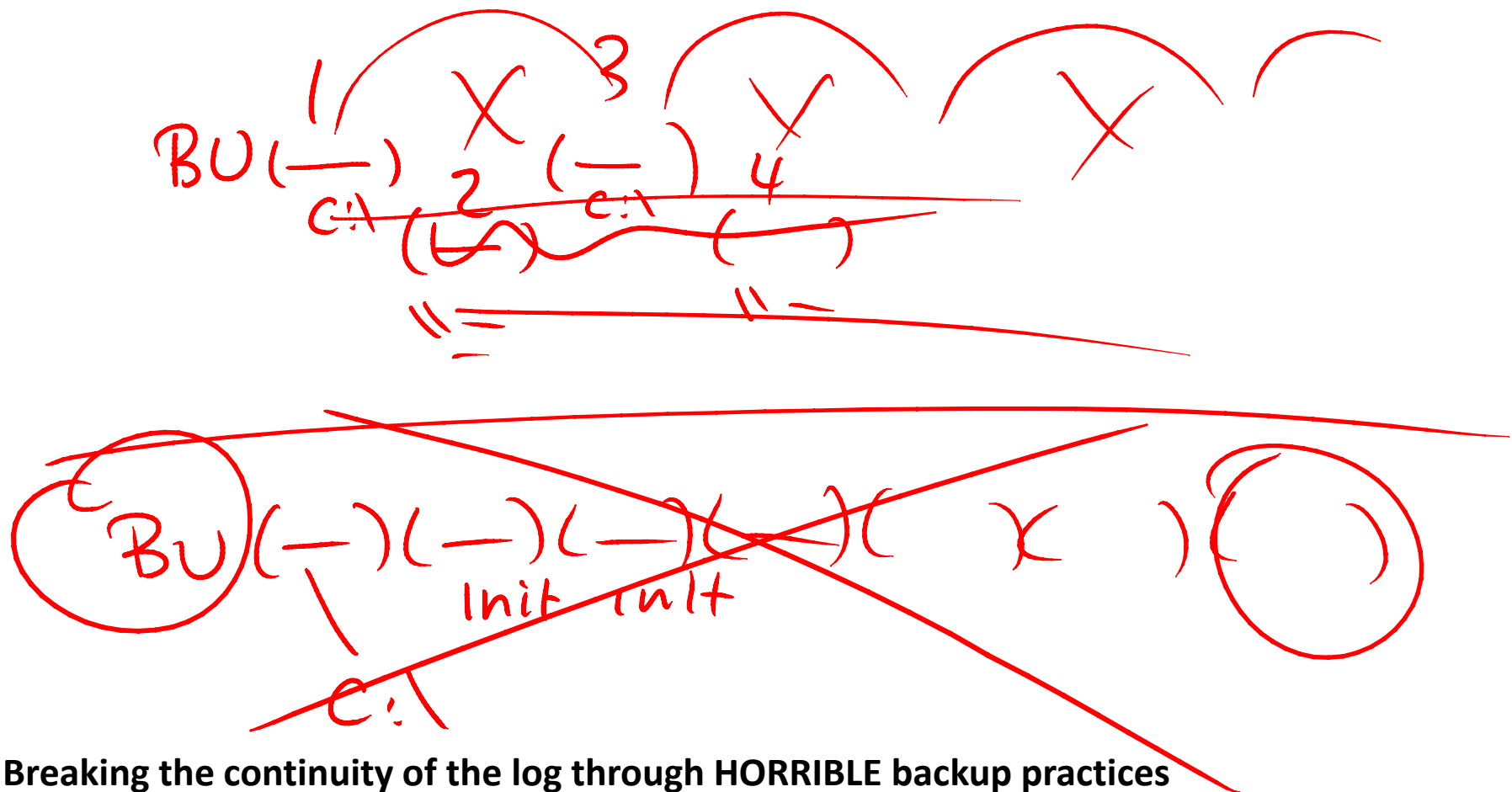
pseudo Simple



2005
no log
truncate only

Breaking the continuity of the log

In SQL Server 2005, breaking the log chain breaks the ability to backup the log without first doing some other “data” backup (full database, database differential, full filegroup, filegroup differential, full file or file differential). Your database is put into a pseudo SIMPLE recovery model where you log is cleared on checkpoint. Transaction log backups will fail.

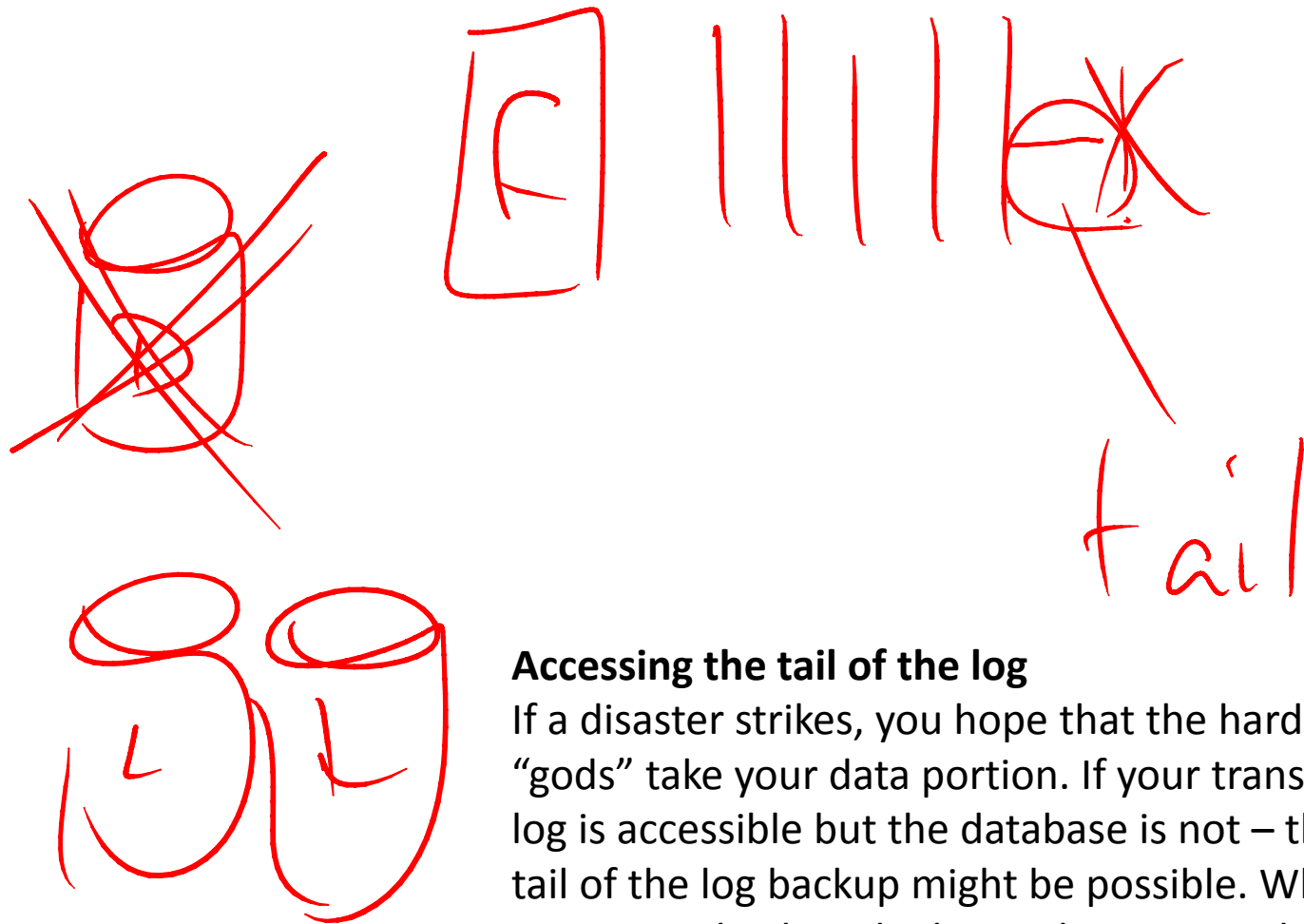


Breaking the continuity of the log through HORRIBLE backup practices

The first “strategy” was one that Edward had experienced. His customer had been backing up every OTHER backup to the network. So, local then network, then local, then network. This is a recipe for disaster!

The second “strategy” is one where the customer did NOT understand log backups AT ALL. They were backing up every log with INIT. If they had had a disaster they would have had the backup plus ONLY the last few transactions with NO WAY of getting the log to a useable point.

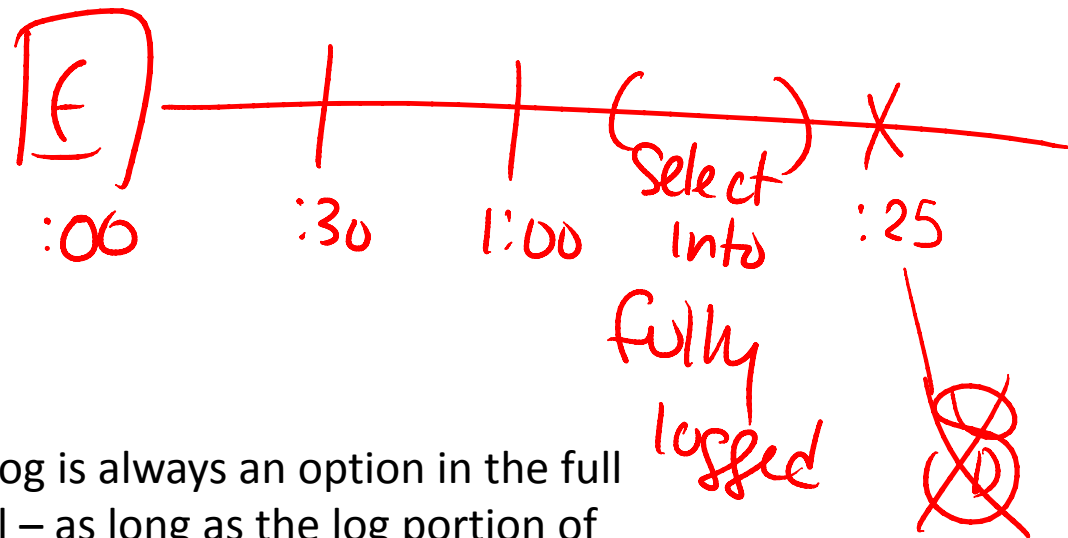
Key Point: TEST YOUR BACKUP/RESTORE STRATEGY



Accessing the tail of the log

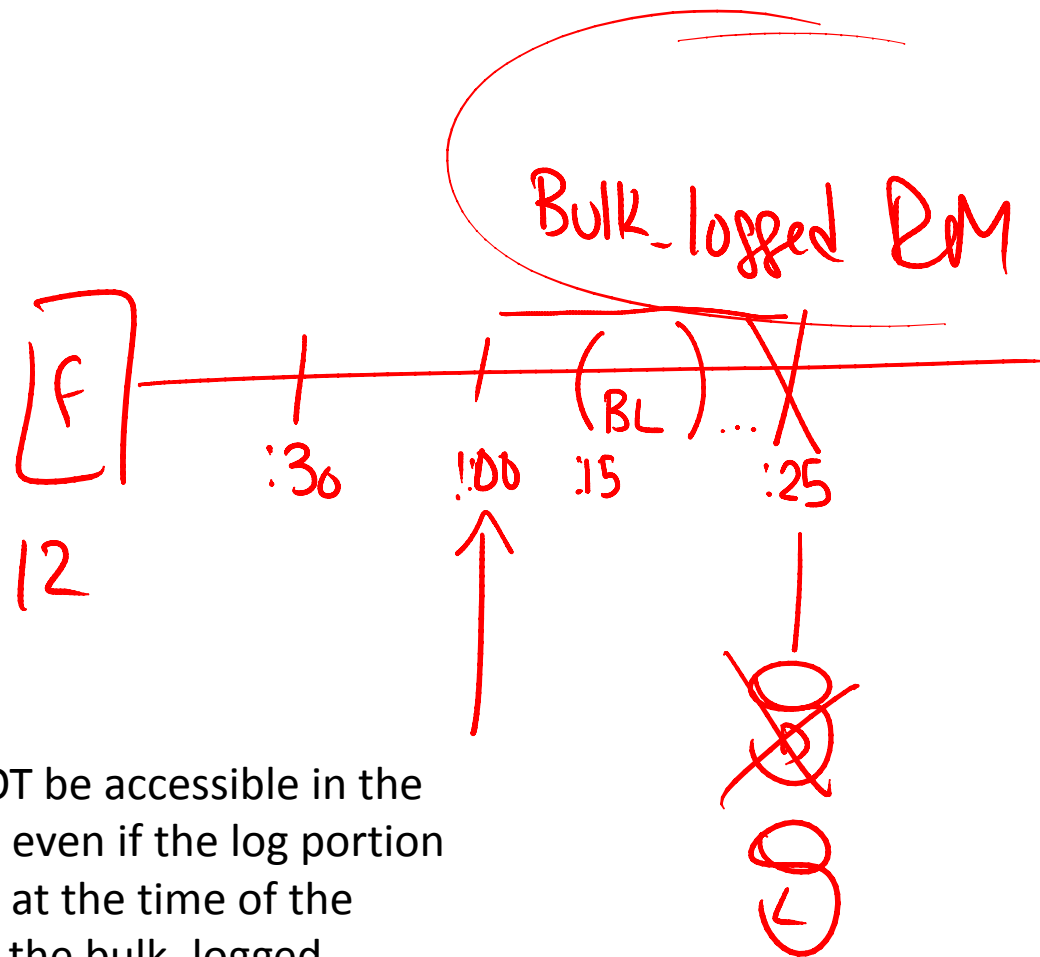
If a disaster strikes, you hope that the hard drive “gods” take your data portion. If your transaction log is accessible but the database is not – then a tail of the log backup might be possible. When it is, you can backup the log and get up-to-the-minute recovery of your database (with ZERO data loss). This is why redundancy for the log is the most important – even more so than for the data portion. However, you need redundancy there as well to minimize downtime.

Full RM



The tail of the log is always an option in the full recovery model – as long as the log portion of the database is accessible at the time of the disaster. If the database is in the FULL recovery model then ALL operations are fully logged (even those that have the ability to be minimally logged; they are ONLY minimally logged if the database recovery model is NOT FULL).

Ⓢ Tail is possible

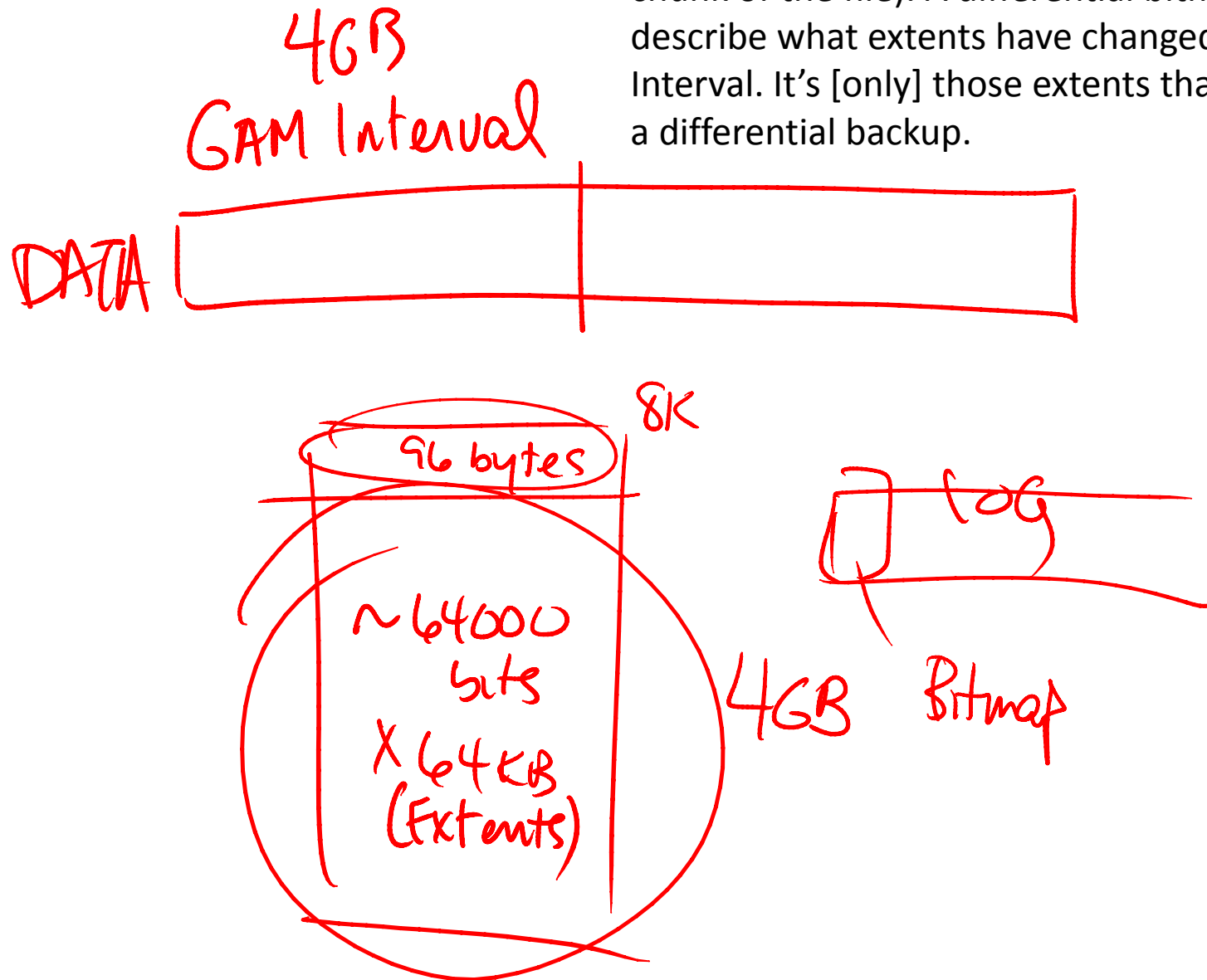


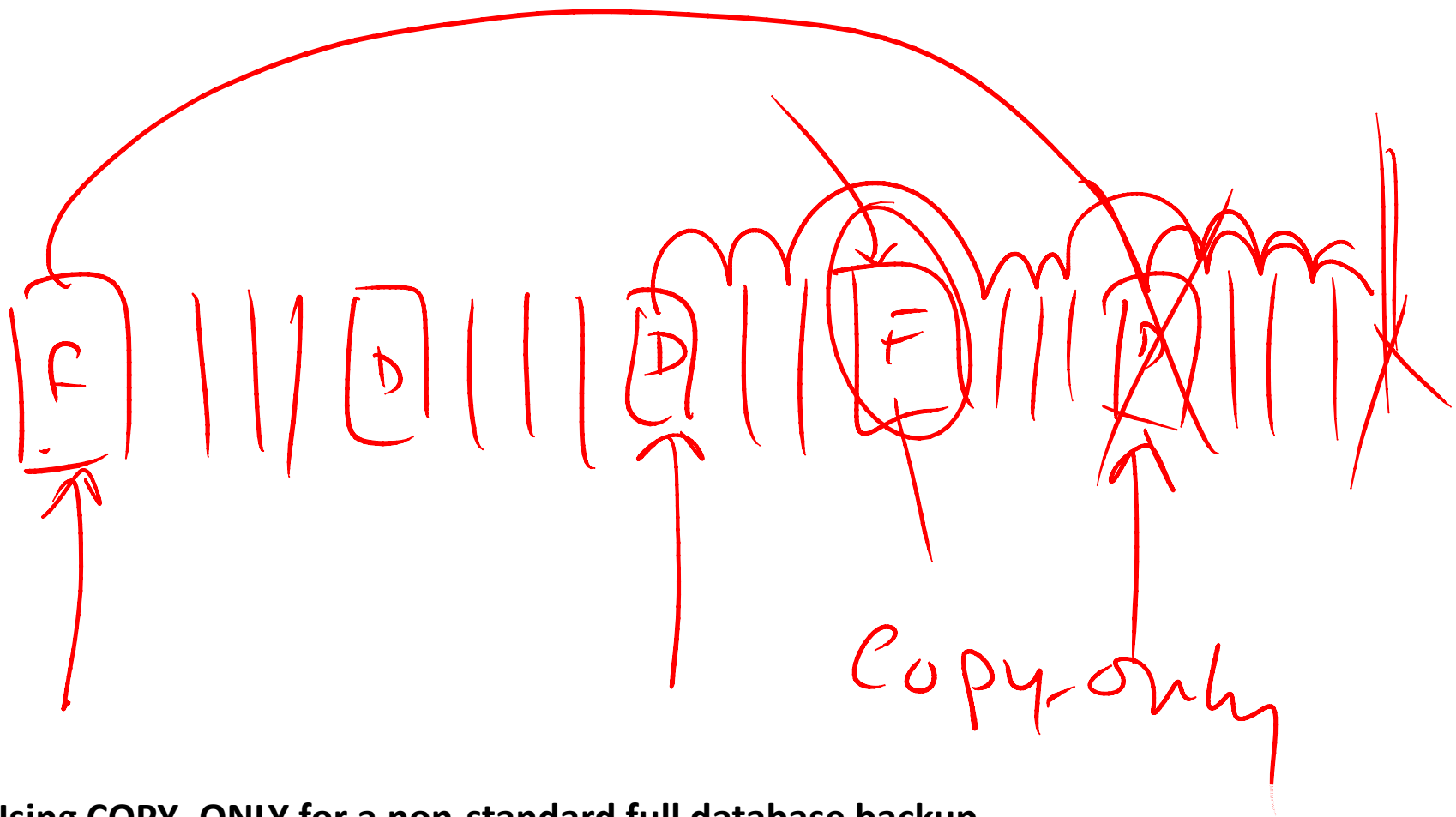
The tail of the log MIGHT NOT be accessible in the bulk_logged recovery model even if the log portion of the database is accessible at the time of the disaster. If the database is in the bulk_logged recovery model AND a bulk operation HAS occurred then the tail of the log cannot be backed up.

End result – be sure to protect the log by restricting the time that a database is in the bulk_logged recovery model and do not allow users into the database during this time.

IE1 Internals Review

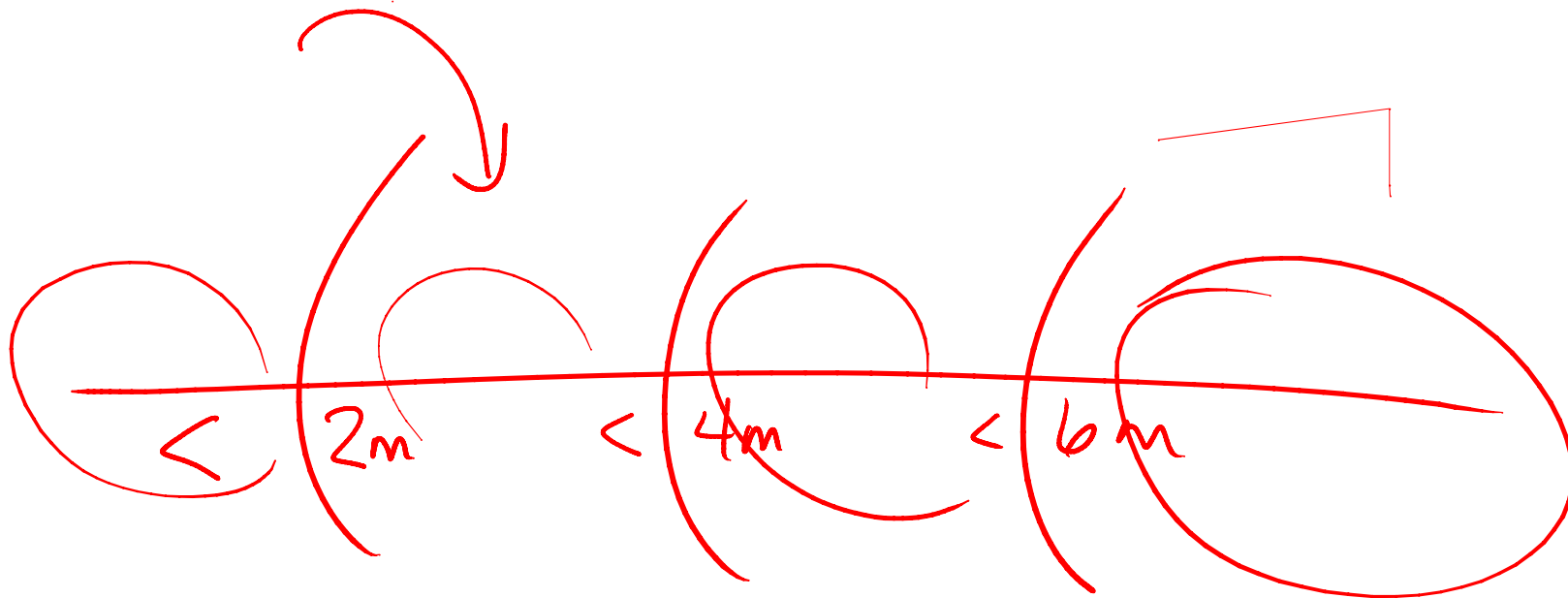
“Bitmap” pages in SQL Server use the 8K page as a map to describe *something* about a GAM Interval (4GB chunk of the file). A differential bitmap uses a page to describe what extents have changed within that GAM Interval. It’s [only] those extents that are backed up on a differential backup.





Using COPY_ONLY for a non-standard full database backup

If a developer/tester or another team needs a backup of a database, you do not just want to back it up and give it to them. Performing a full database backup resets the differential bitmap. All differentials would then be tied to that most recent backup and not the one that was performed during your regularly scheduled backups. As a result, your restore process might take quite a bit longer (and require quite a few more log backups to be applied). Use COPY_ONLY instead and the diff bitmap will not get reset!



Online operations – SalesDB partitioning

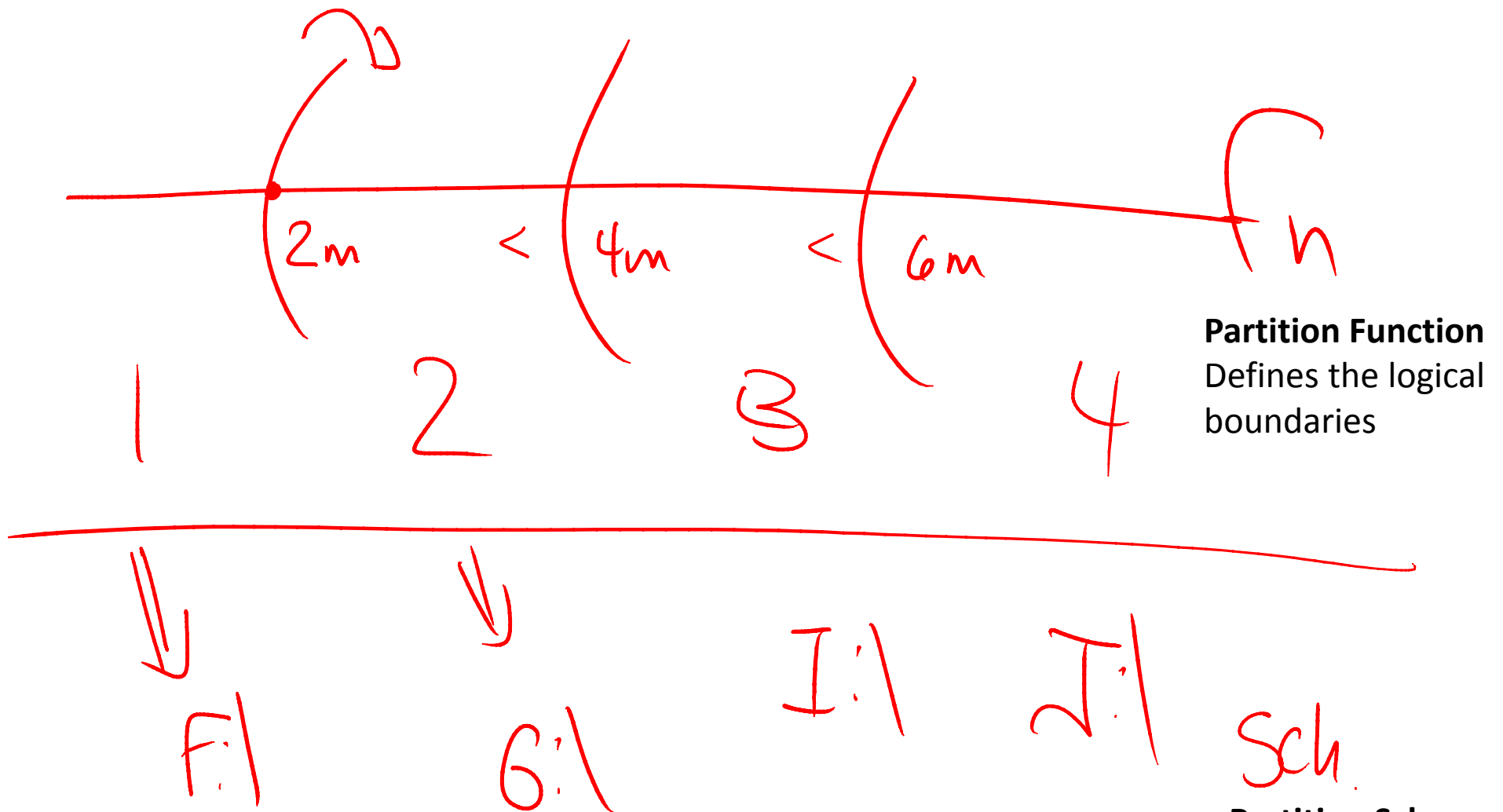
The sales table was partitioned with 3 boundary points: 2 million, 4 million and 6 million. It's a RIGHT-based partition function so the rows that match the boundary point will be to the RIGHT side. Specifically this translates into:

In partition 1: all rows less than 2 million

In partition 2: all rows equal to or greater than 2 million and less than 4 million

In partition 3: all rows equal to or greater than 4 million and less than 6 million

In partition 4: all rows equal to or greater than 6 million



Online operations – SalesDB partitioning

When I partitioned the sales data, I ended up writing the data to the USB drives.

Drive F:\ (partition 1): all rows less than 2 million

Drive F:\ (partition 1): all rows equal to or greater than 2 million and less than 4 million

Drive F:\ (partition 1): all rows equal to or greater than 4 million and less than 6 million

Drive F:\ (partition 1): all rows equal to or greater than 6 million

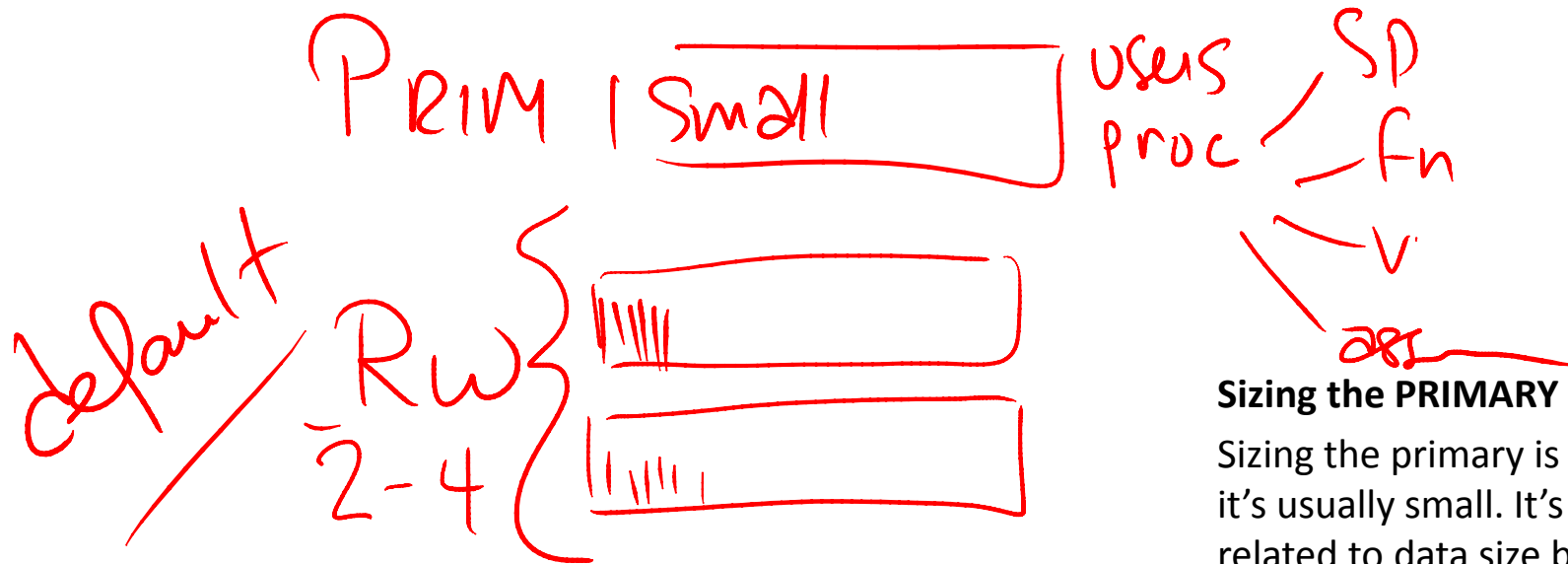
Partition Scheme

Defines the physical location for the data



Primary filegroup, Primary file?

The first file associated with a database is the MDF (main data file) which is also known as the PRIMARY DATA FILE (but Adobe beat us to PDF ☺). This file is critical for a database to be up and running. If you add files to a database without specifying a filegroup they are going to end up in the PRIMARY filegroup. Because the primary file is a member and because allocations can come from any file of the filegroup – ALL of the files of the primary filegroup end up being critical. Generally, it's important to isolate the primary filegroup (with just one mdf) and protect it (as much as you would the transaction log).



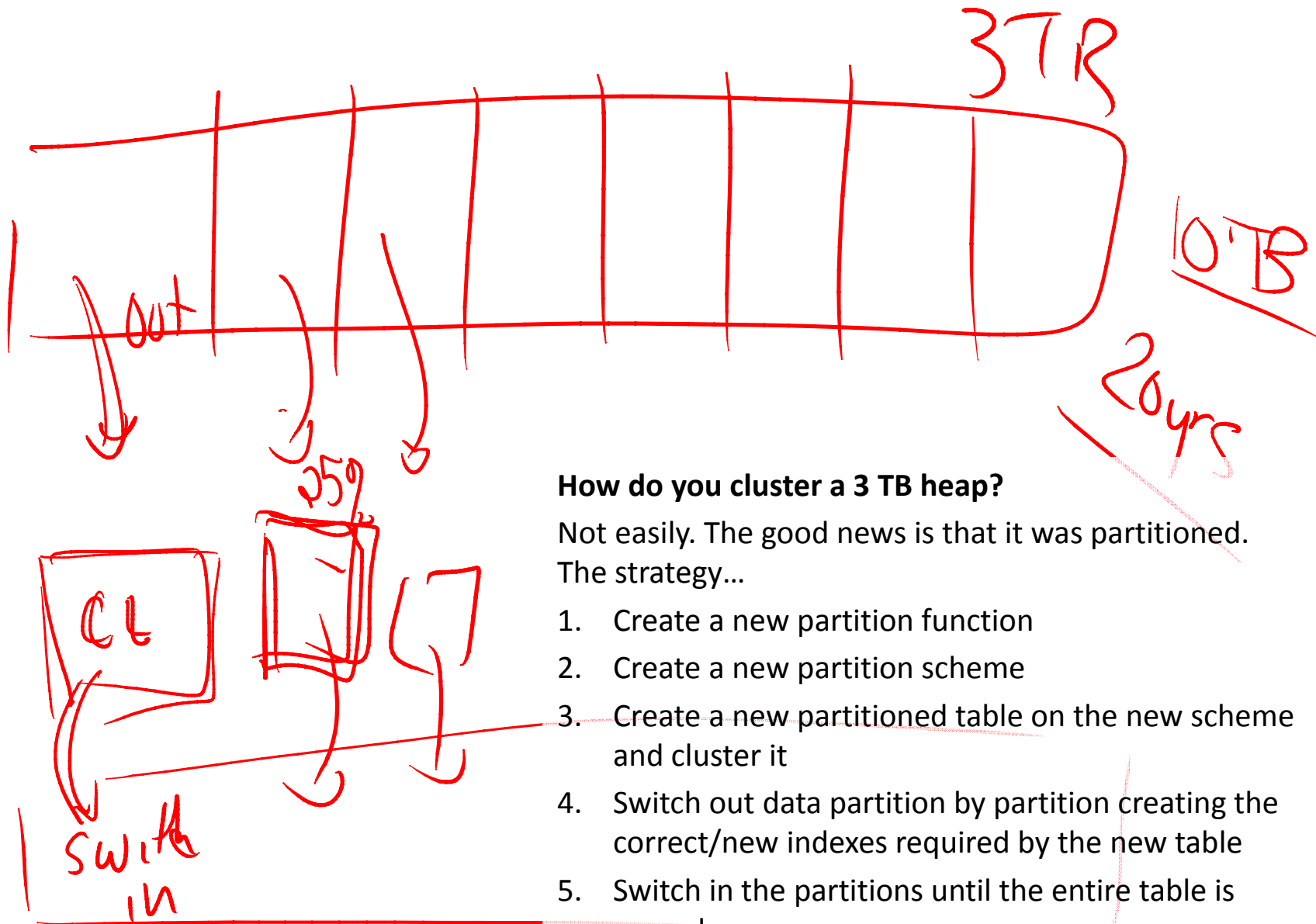
Sizing the PRIMARY filegroup

Sizing the primary is challenging but it's usually small. It's not directly related to data size but instead related to metadata complexity. Users/permissions as well as code (functions, procedures, assemblies) are what's stored in the "primary" portion of the database.

RO 2011 → R10
 RO 2010 → R5
 RO 2009 → R0

Isolating the primary filegroup

More specifically, isolate primary and separate this from your user-defined data. Create a filegroup for readwrite data that has at least 2 files and then read-only data can be in one or more filegroups (depending on availability requirements, backup/restore SLAs and hardware options/costs)



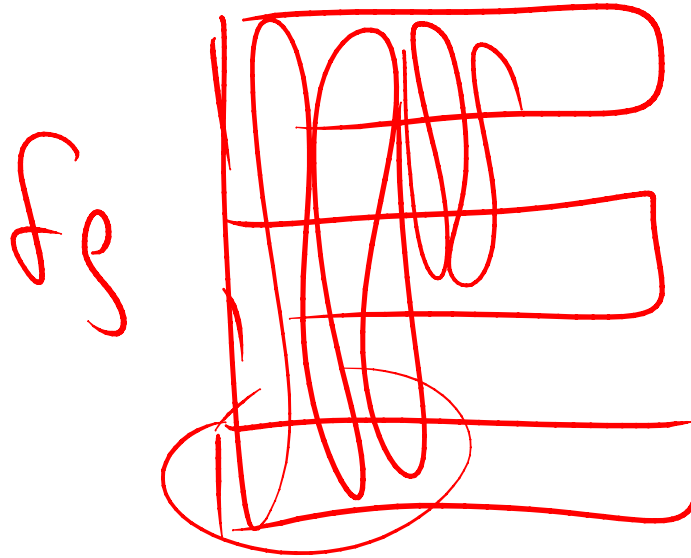
How do you cluster a 3 TB heap?

Not easily. The good news is that it was partitioned.
The strategy...

1. Create a new partition function
2. Create a new partition scheme
3. Create a new partitioned table on the new scheme and cluster it
4. Switch out data partition by partition creating the correct/new indexes required by the new table
5. Switch in the partitions until the entire table is moved over

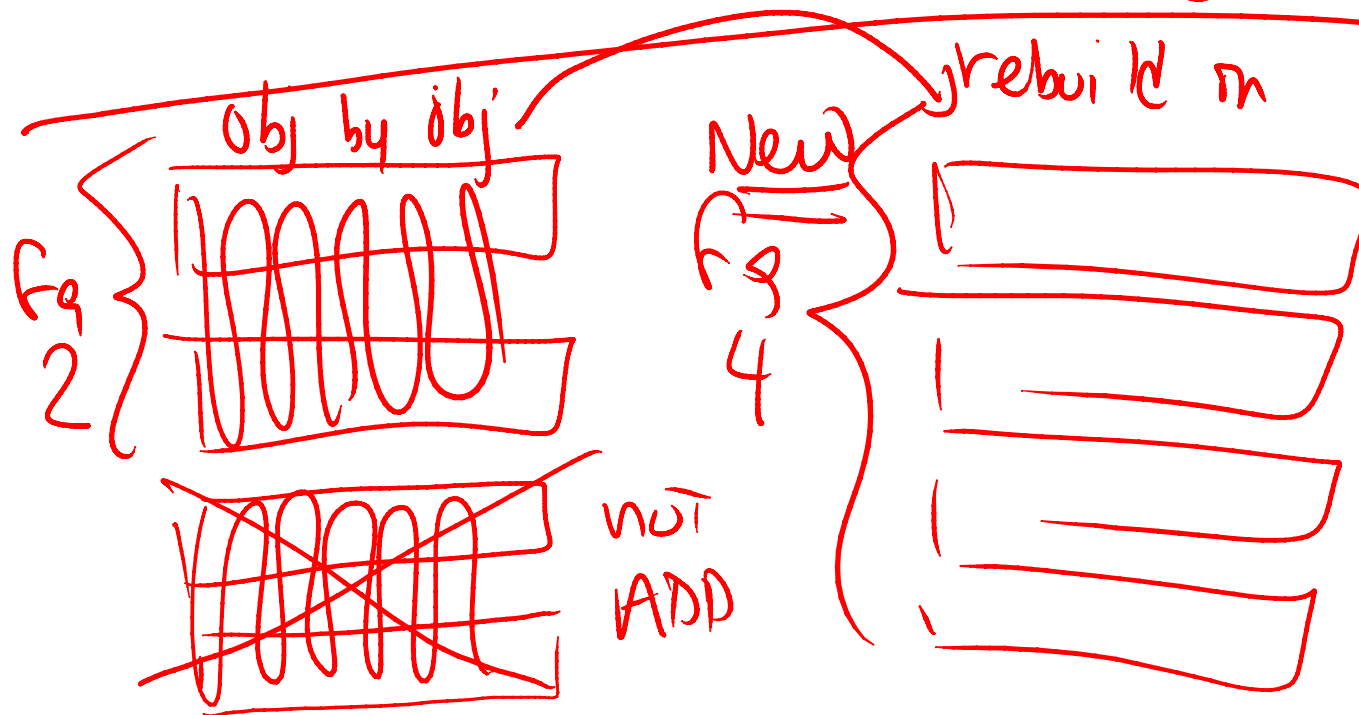
Concerns:

- * primary/foreign keys
- * downtime



Removing does
balance

Remove Empty file



Changing the
number of files in
a filegroup

See the next slide
for details.

Changing the number of files in a filegroup

Removing a file must be done through EMPTYFILE (which DOES rebalance the data among the remaining files in the filegroup). However, it is not object specific.

Adding a file to a filegroup does NOT rebalance. If you want to change a filegroup from two or three files to something like four or five, your best bet is to create a NEW filegroup and then rebuild all of your indexes into it.

Use CREATE with DROP_EXISTING to MOVE an index to another filegroup.

Now, as for moving LOB data... this is the most interesting and it DOES work if you use DROP_EXISTING and rebuild on a scheme or conversely from a scheme to just a single filegroup. However, going from one filegroup to another (even when using DROP_EXISTING) does NOT move the LOB data. There's a blog post in that information for sure. I might try and do this as a SQLServerPro post in my series on Partitioning (Partitioned Tables vs. Partitioned Views).

Here are the links to the current posts:

Q1: [Partitioned Tables v. Partitioned Views—Why are they even still around?](#)

Q2: [Solutions to VLT concerns around statistics and maintenance!](#)

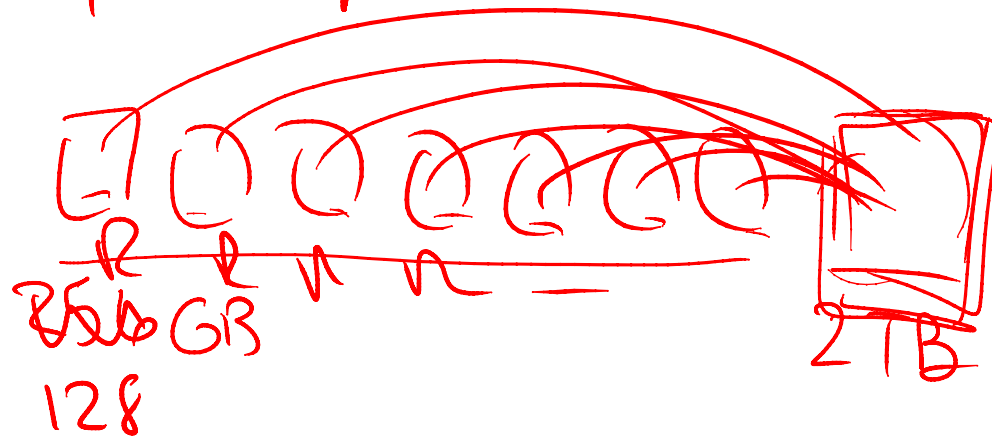
Q3: [Did SQL Server eliminate any partitions?](#)

Q4: [How about Filtered Indexes instead of Partitioning?](#)

Q5: Is there a way to move LOB data? (or something like that)

The final consideration is availability. See the next slide for Brad's (@SQLPhilosopher's) trick!

SQL philosopher . com



As an alternative, Brad (@SQLPhilosopher) came up with a way to move LOB data to a different number of files – and, do this as an ONLINE operation.

He described it fully here: <http://www.sqlphilosopher.com/wp/2012/02/moving-a-filegroup-to-a-new-disk-array-with-no-downtime/>.

It's not a different filegroup but the end result (restructuring the filegroup) is the same!

COPY_ONLY vs. NO_TRUNCATE

NOTE: This is a new slide that I added to the Backup module and I thought you might want to have the info as a reminder from class! fyi, kt ☺

- ◆ Sound similar... but, they're not the same!
 - ◆ COPY_ONLY will only work if the backup is OK
 - ◆ NO_TRUNCATE
 - ◆ A backup of the log without clearing the log
 - ◆ Designed for situations where the data portion is unavailable...
 - ◆ Works even if the log backup is damaged but you won't know this because NO_TRUNCATE does a CONTINUE_AFTER_ERROR even if you don't ask
- + NO_TRUNCATE should always work
- You won't really know if there's an error until you restore. You might be forced to use CONTINUE_AFTER_ERROR

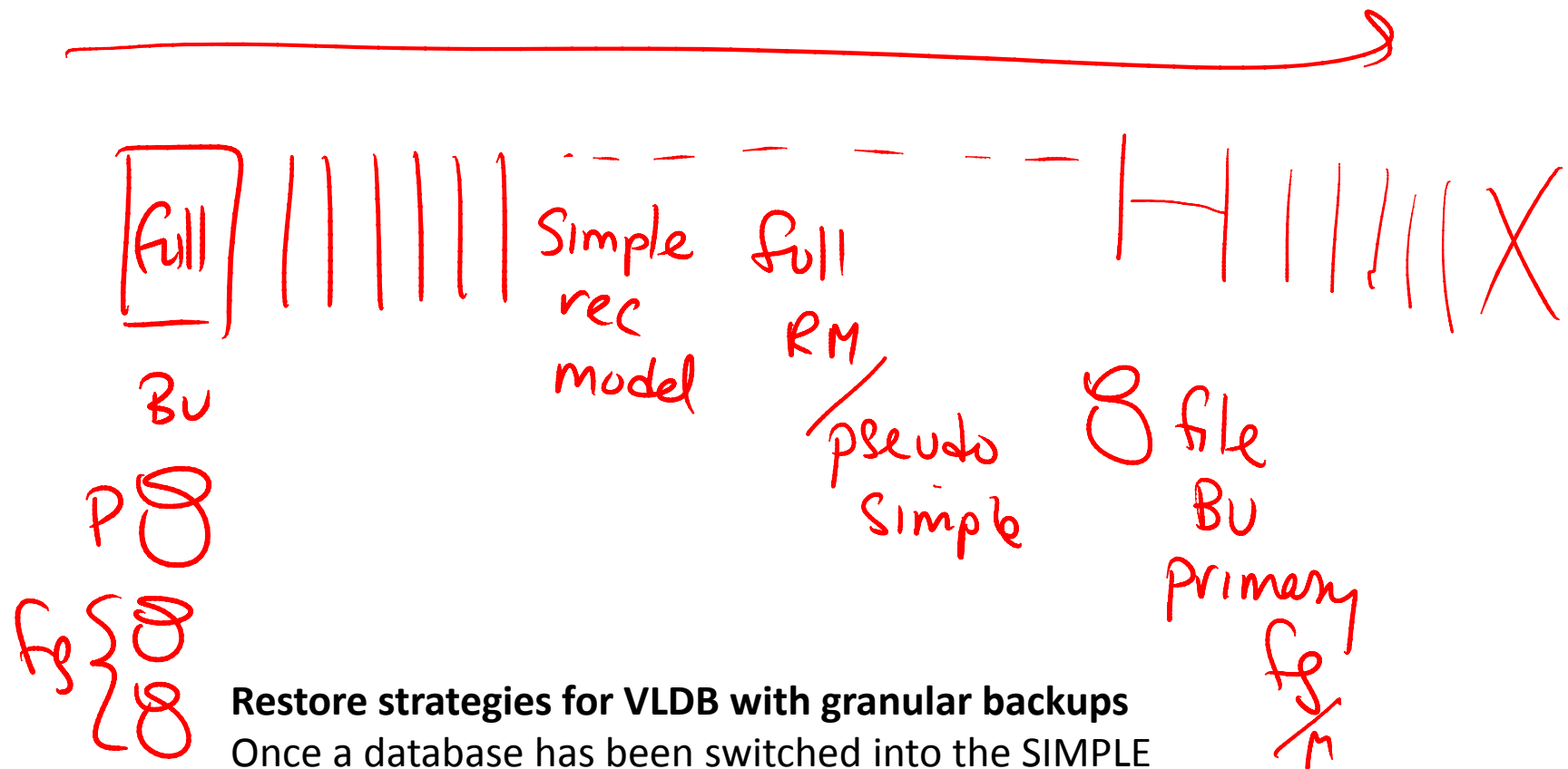


Module 4

Restore Internals and Procedures

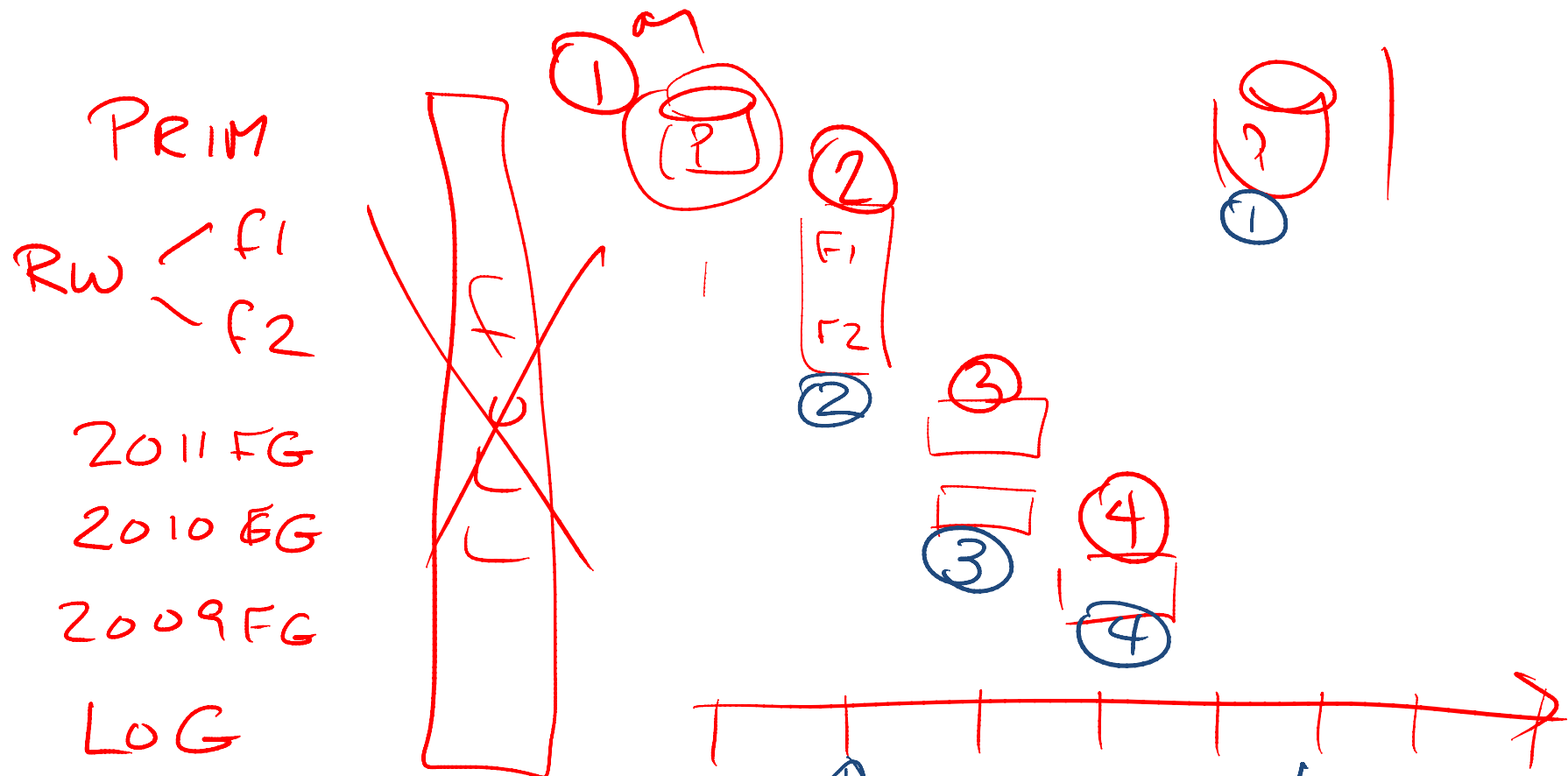
SQL
skills

This shows a multi-file database and a series of different types of backups. What was most important here is the understanding of what happens when a filegroup is taken OFFLINE. This is the point in time when SQL Server knows that no new transactions will occur. This is the point in time to which this filegroup must be restored (reconciled) in order for this filegroup to be brought online.



Restore strategies for VLDDB with granular backups

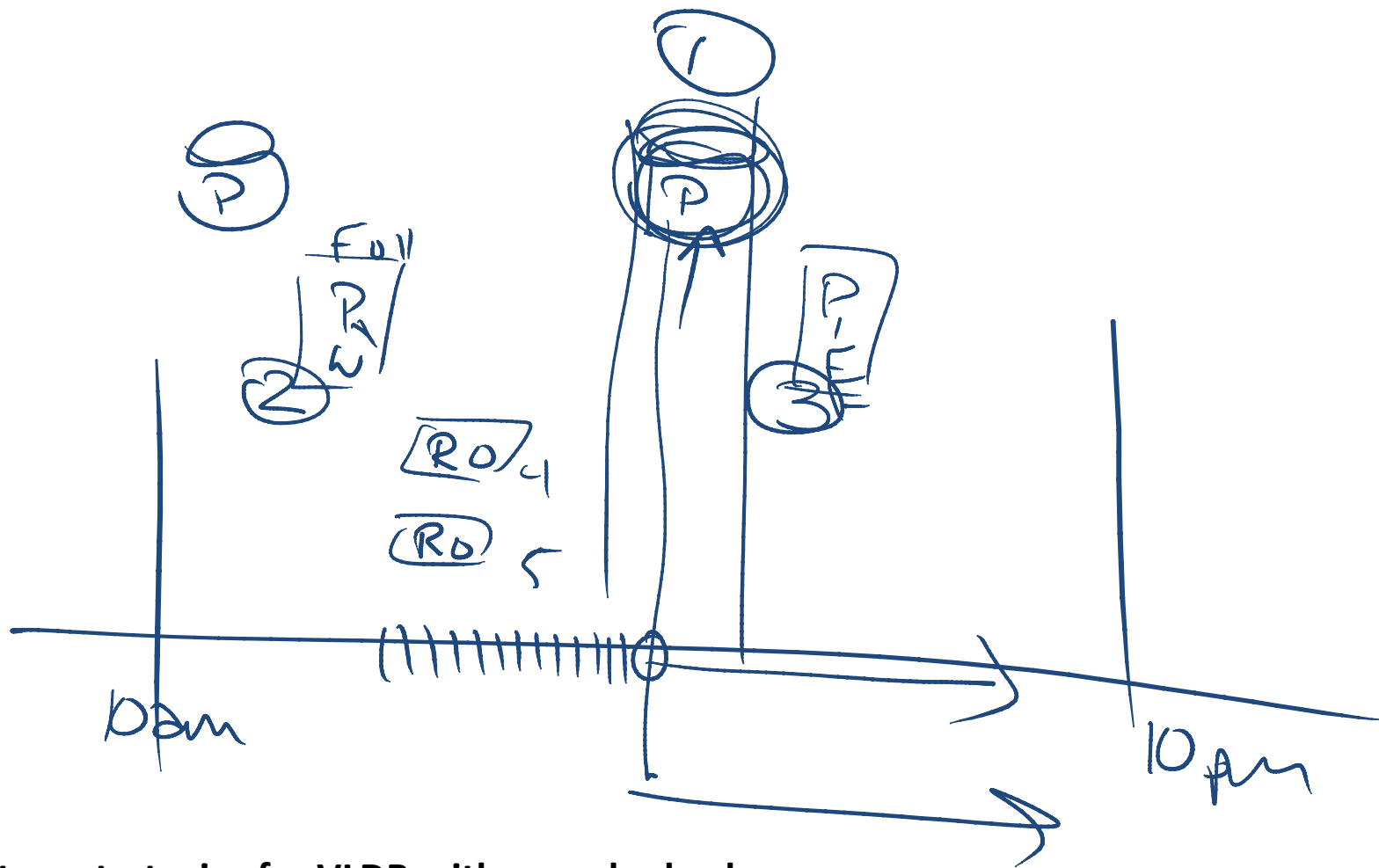
Once a database has been switched into the SIMPLE recovery model then the continuity of the log has been broken. You do NOT need to do a full database backup to begin the process of log backups again. You only need to "begin" your backup strategy. If you perform any type of backup (full database, full filegroup, full file OR even a differential database, differential filegroup or differential file) then you have begun your strategy. However, you are still VERY vulnerable until ALL of your files have been backed up in some way. Always a good idea to do a database-level backup (full or differential) after breaking the continuity.



Restore strategies for VLDB with granular backups

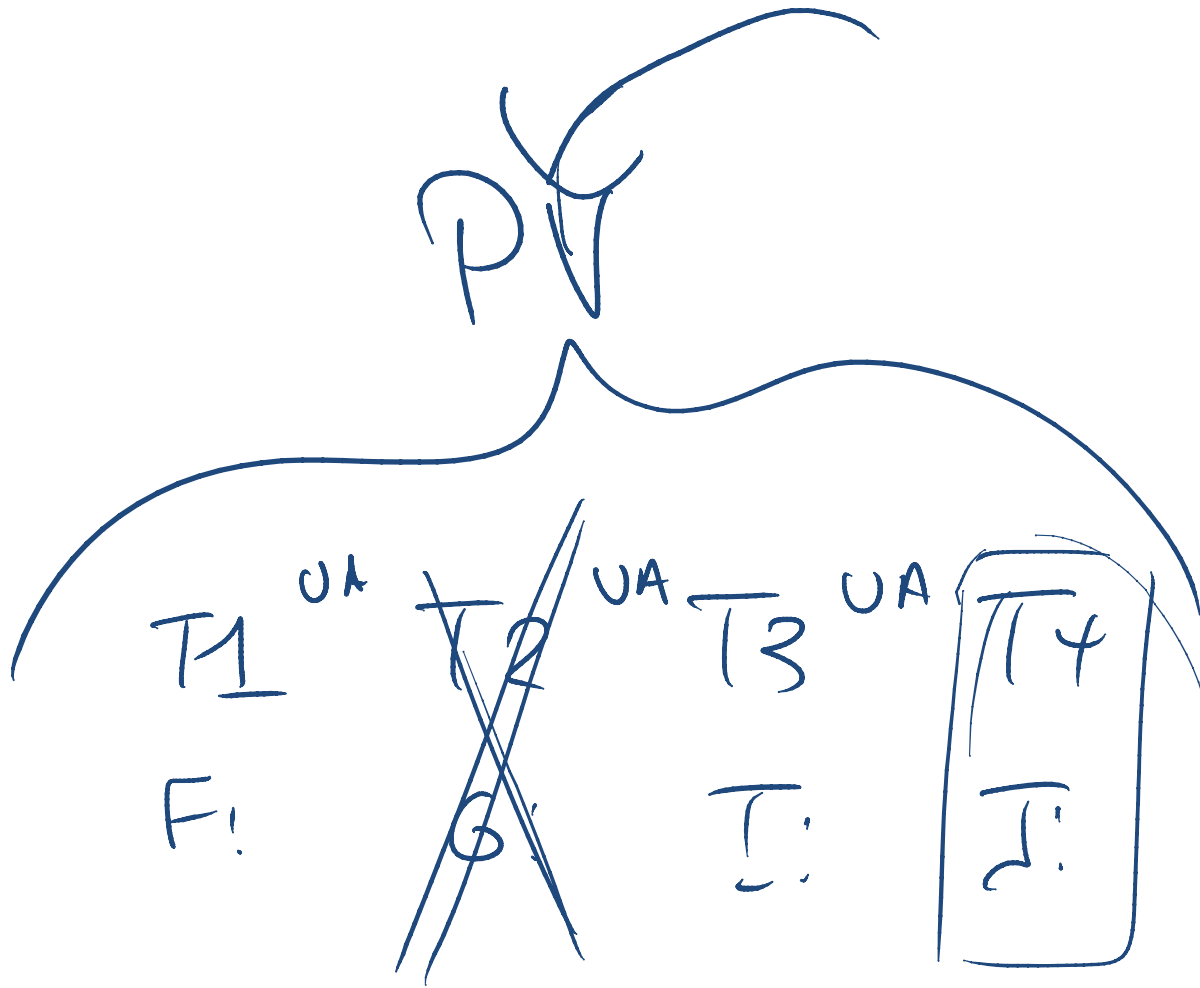
To reconstruct a database from granular backups requires that ALL pieces are restored.

However, since they were backed up at different times – a transaction log series that starts with the oldest filegroups minimum (or effective) LSN must be used. If you have a current primary and all of the filegroups (labeled in blue as 2, 3, 4) then you only need logs starting with the bottom blue arrow. If the blue backup #1 did not exist and you had to go back to the red #1 then you'd need all of the logs back to that point. The KEY point is that the database will be completely offline until the entire set of filegroups being restored are recovered to the SAME point in time. If you restore PARTIAL then you could bring it online more slowly/granularly.



Restore strategies for VLDB with granular backups

This is a continuation of the last picture. If you have a primary filegroup backup (#1) then a RWFilegroup backup (#2) then you could restore that with logs to bring the entire database up to current (restoring the RO filegroups at virtually any time – possibly a lot later if you really just want to get the RW up to current). If you have a differential of the RWFilegroup then you could have restored it and then you only would have needed the logs from the backup time of the Primary filegroup.



Querying data when a filegroup is damaged

If you query a partitioned table when a filegroup is damaged then you'll only receive an error if you (try to) select from the damaged partition. This is also NOW true of a partitioned view. It was not always true (in 2005 a view that was damaged would fail). I **think** this was fixed in 2008 but for sure in 2008 R2.



The concept is this... if a database uses column-level encryption and therefore stores encrypted data, then you'll need to re-associate the DMK (database master key) with the new server's SMK (service master key) when you restore. Otherwise, all of the data will be inaccessible!

```
USE database;
```

```
go
```

```
-- Open the DMK with the SAME password used when created:
```

```
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'original password';
```

```
go
```

```
-- Reassociate the DMK with the SMK of the new instance:
```

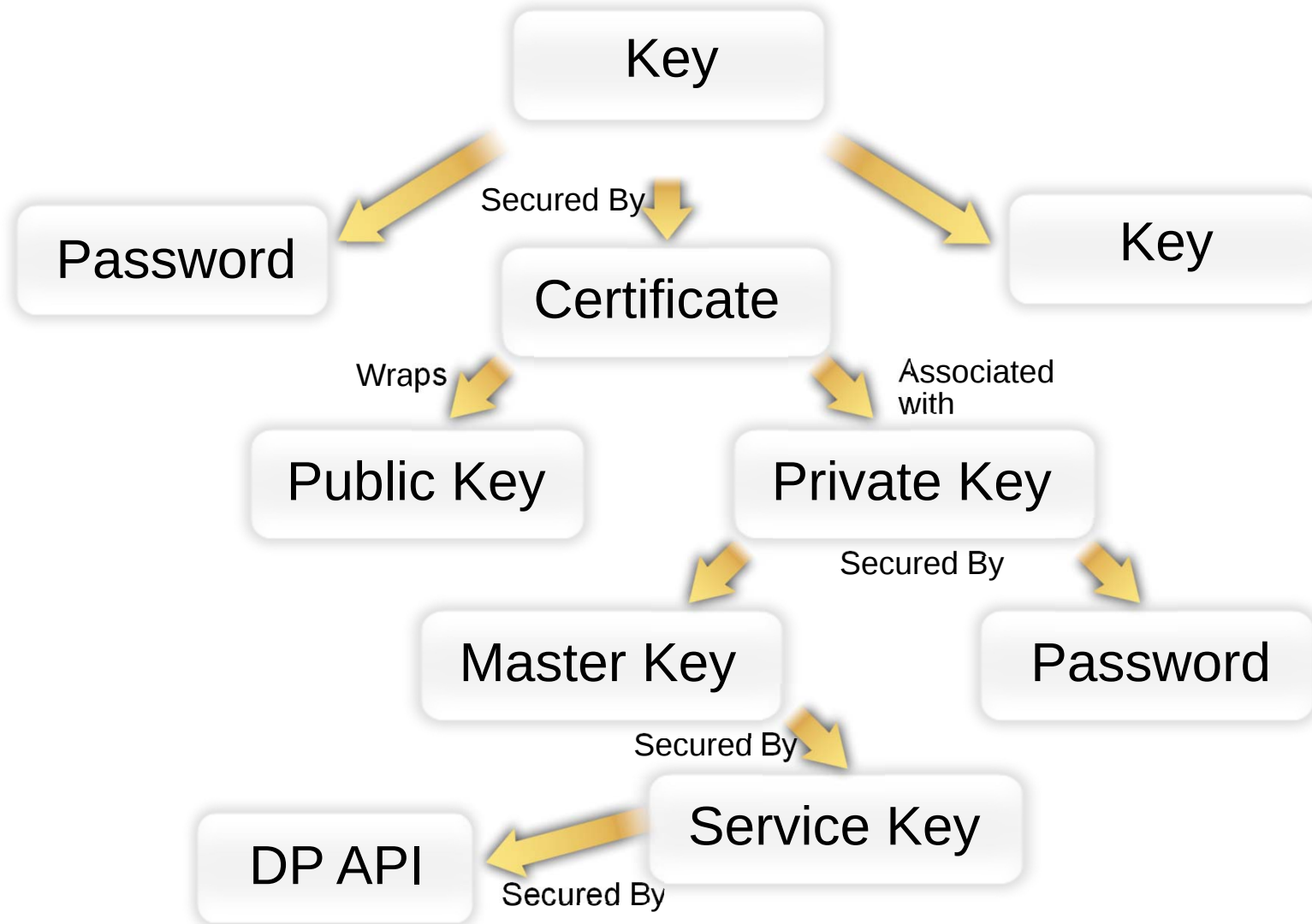
```
ALTER MASTER KEY ADD ENCRYPTION BY SERVICE MASTER KEY;
```

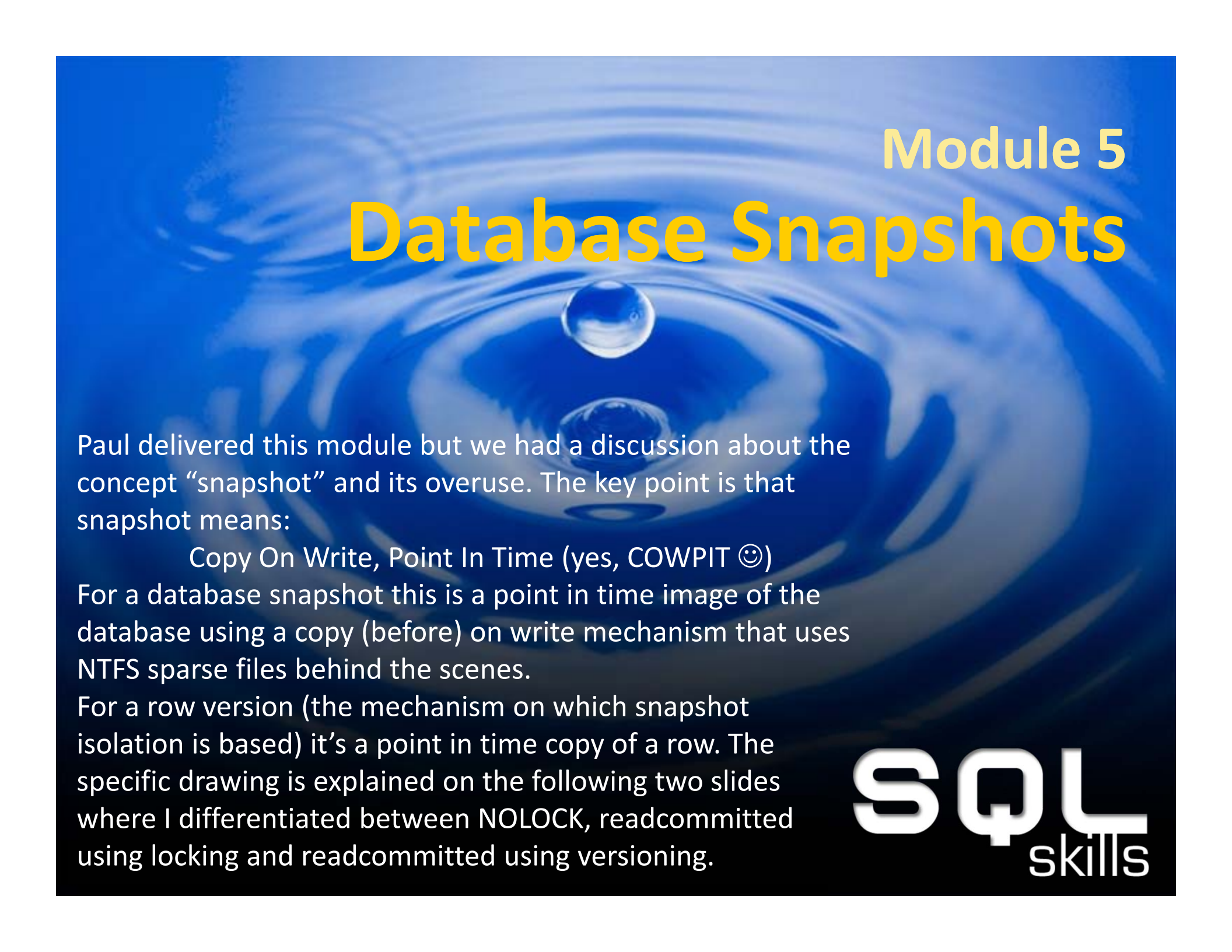
```
go
```

Encryption Hierarchy

- SQL Server 2005 implements a framework to help protect encryption keys by using an encryption key hierarchy, as shown in the figure on this slide. In this hierarchy, each layer encrypts the layers that are below it.
- The Data Protection API (DPAPI) is at the top of this encryption key hierarchy. DPAPI is a pair of function calls that provide operating system–level data protection services to user and system processes. Because this API is part of the Microsoft Windows® operating system, applications can use DPAPI to encrypt data but do not have to use any cryptographic code other than the code that calls the DPAPI functions. DPAPI is a password-based data protection service. Therefore, DPAPI requires a password to encrypt the data.
- The SQL Server 2005 Service Master Key is a symmetric key that the **cryptGenKey** function automatically generates when an administrator installs SQL Server 2005. DPAPI uses the password of the account under which SQL Server 2005 runs to generate a key with which DPAPI encrypts the Service Master Key. Therefore, if an administrator changes the service account under which SQL Server 2005 runs, he or she must decrypt the Service Master Key by using the original credentials. Then, he or she must encrypt the Service Master Key by using the new credentials. We strongly recommend that administrators change the service account under which SQL Server 2005 runs only by using the SQL Server 2005 Computer Manager tool. This tool automatically performs the required steps to decrypt the Service Master Key, and to then re-encrypt the Service Master Key.
- The Service Master Key encrypts the Database Master Key. The Database Master Key is required in each database where an organization encrypts data. This key provides the same functionality as the Service Master Key. However, the functionality occurs at a database level instead of a SQL Server 2005 instance level.
- The Database Master Key is a symmetric key. It is not automatically created when a new SQL Server 2005 database is created. Therefore, the developer must explicitly create this key to implement encryption in a database.
- The Database Master Key encrypts the user keys that a developer creates. These user keys include certificates and asymmetric keys. In turn, certificates and asymmetric keys encrypt symmetric keys that the developer creates.
- **Note:** A developer can also use certificates and asymmetric keys to encrypt data directly. However, certificates and asymmetric keys are best suited to encrypt only small amounts of data.
- A developer can use symmetric keys to encrypt other symmetric keys that he or she creates or to encrypt data. This encryption key hierarchy is especially important to consider when the developer determines the type of key to use to encrypt newly created keys. The SQL Server 2005 encryption key framework gives a developer a large amount of flexibility to create a simple or complex encryption key hierarchy for each database that he or she creates. However, we recommend that a developer create as simple a key structure as his or her organization allows.

Encryption Hierarchy





Module 5

Database Snapshots

Paul delivered this module but we had a discussion about the concept “snapshot” and its overuse. The key point is that snapshot means:

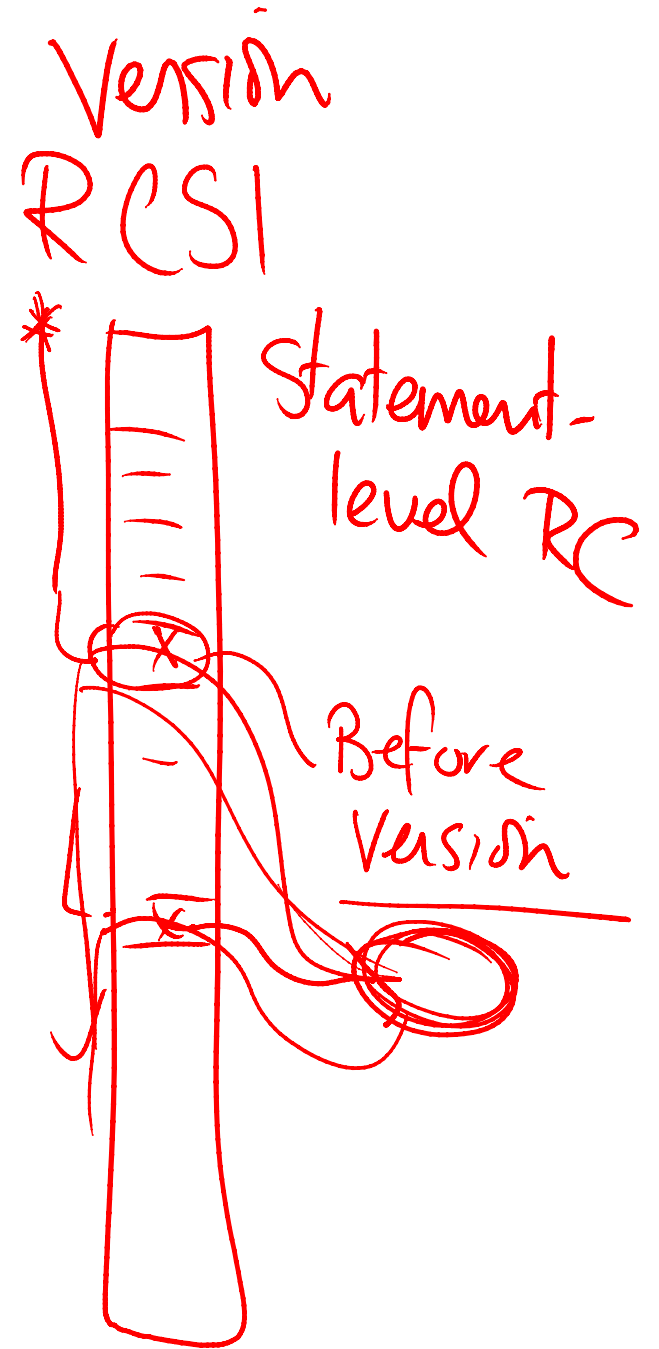
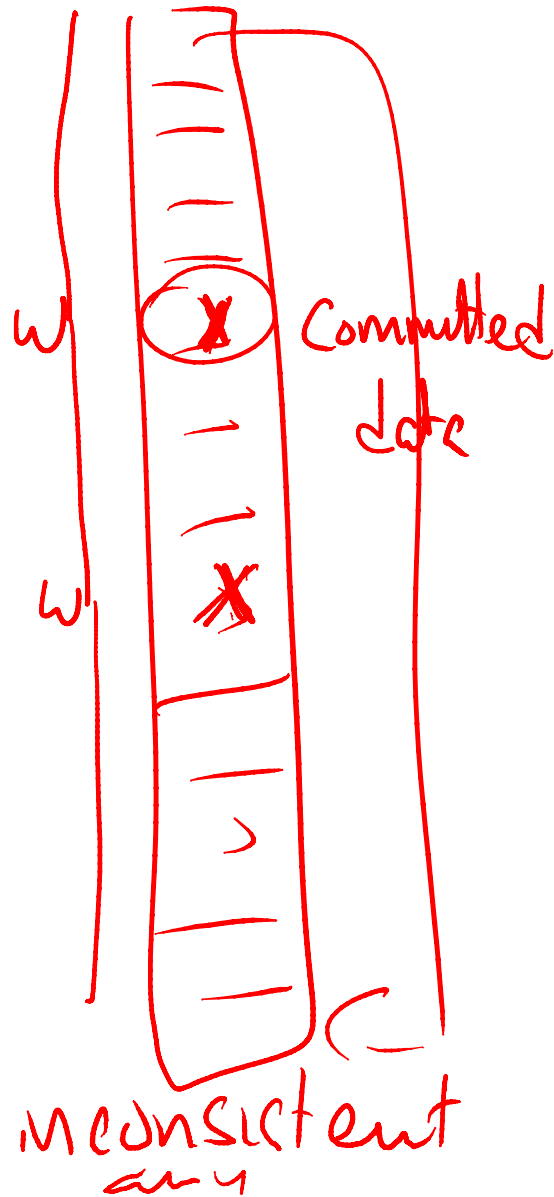
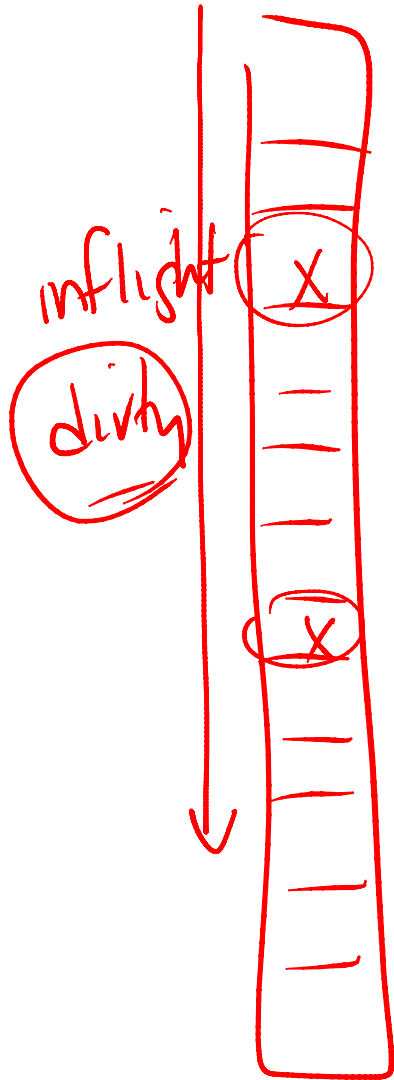
Copy On Write, Point In Time (yes, COWPIT 😊)

For a database snapshot this is a point in time image of the database using a copy (before) on write mechanism that uses NTFS sparse files behind the scenes.

For a row version (the mechanism on which snapshot isolation is based) it's a point in time copy of a row. The specific drawing is explained on the following two slides where I differentiated between NOLOCK, readcommitted using locking and readcommitted using versioning.

SQL
skills

select nowait ~~RC~~ Locking



The prior slide showed a table vertically as a set of pages. The idea was to compare/contrast the behaviors between:

(1) A forced NOLOCK hint

The key thing to remember here is that nologlock allows the reader to read quickly – not stopping for locked rows. However, these rows may be “in-flight” and their modified data might end up getting rolled back or even be in a mid-flight state. If you’re looking for only an estimate – this might be fine.

(2) The default – READ COMMITTED (using locking)

The key things to remember from the picture is that (1) has “hiccups” along the way as they encounter rows that are locked. They read... wait... read... wait. This is *partially* what slows down a statement. However, it’s worse than this. It’s possible that AFTER a row is read, it will appear again (if the record is relocated) and we will read it twice (non-repeatable reads). Also, it’s possible that another transaction would modify a row that we’ve NOT yet seen (before we get there) as well as a row that we have seen (after we’ve read it) such that the end result of our statement has rows that aren’t really transactionally consistent (with another multi-statement transaction). This is also a problem... The default (1) environment is prone to a few inconsistencies (aka. Inconsistent analysis). Only way to solve – increase isolation (or [as of 2005] consider using versioning).

(3) Versioning (and it was implied that it was statement-level RCSI: read_committed_snapshot)

When a versioned query runs the reader will not stop but will have to go to the version store for any row that’s in flight. This is a tiny bit slower but guarantees consistency of the ENTIRE read to the point in time when the statement started. This gives you better concurrency as well as a definable point in time to which your statement reconciles. Of course, it isn’t free – the overhead of versioning is for every writer and it occurs within tempdb.