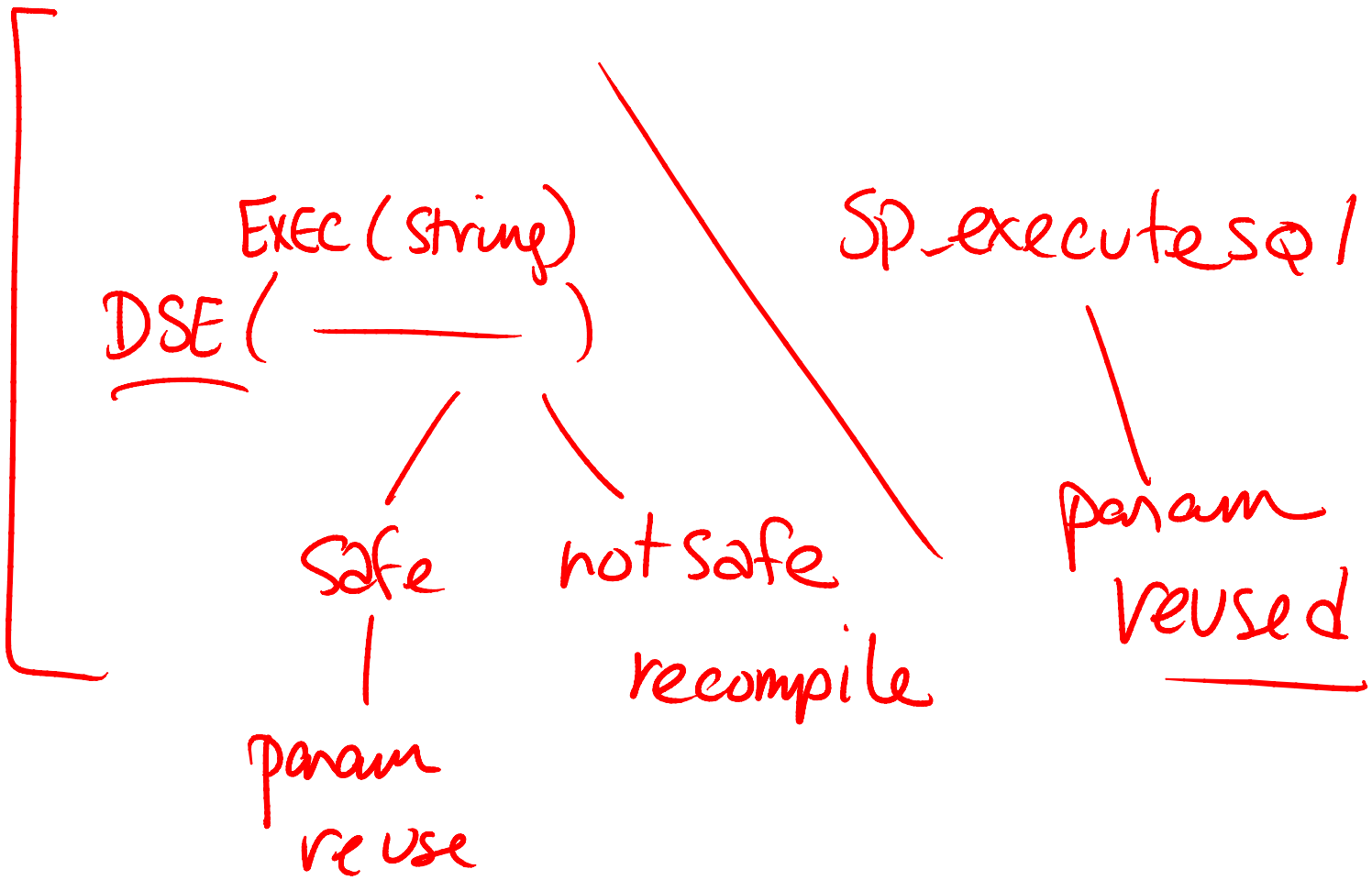
SQLskills Immersion Event
IE2: Performance Tuning

Module 9: Plan Cache & Index Analysis

Kimberly L. Tripp
Kimberly@SQLskills.com

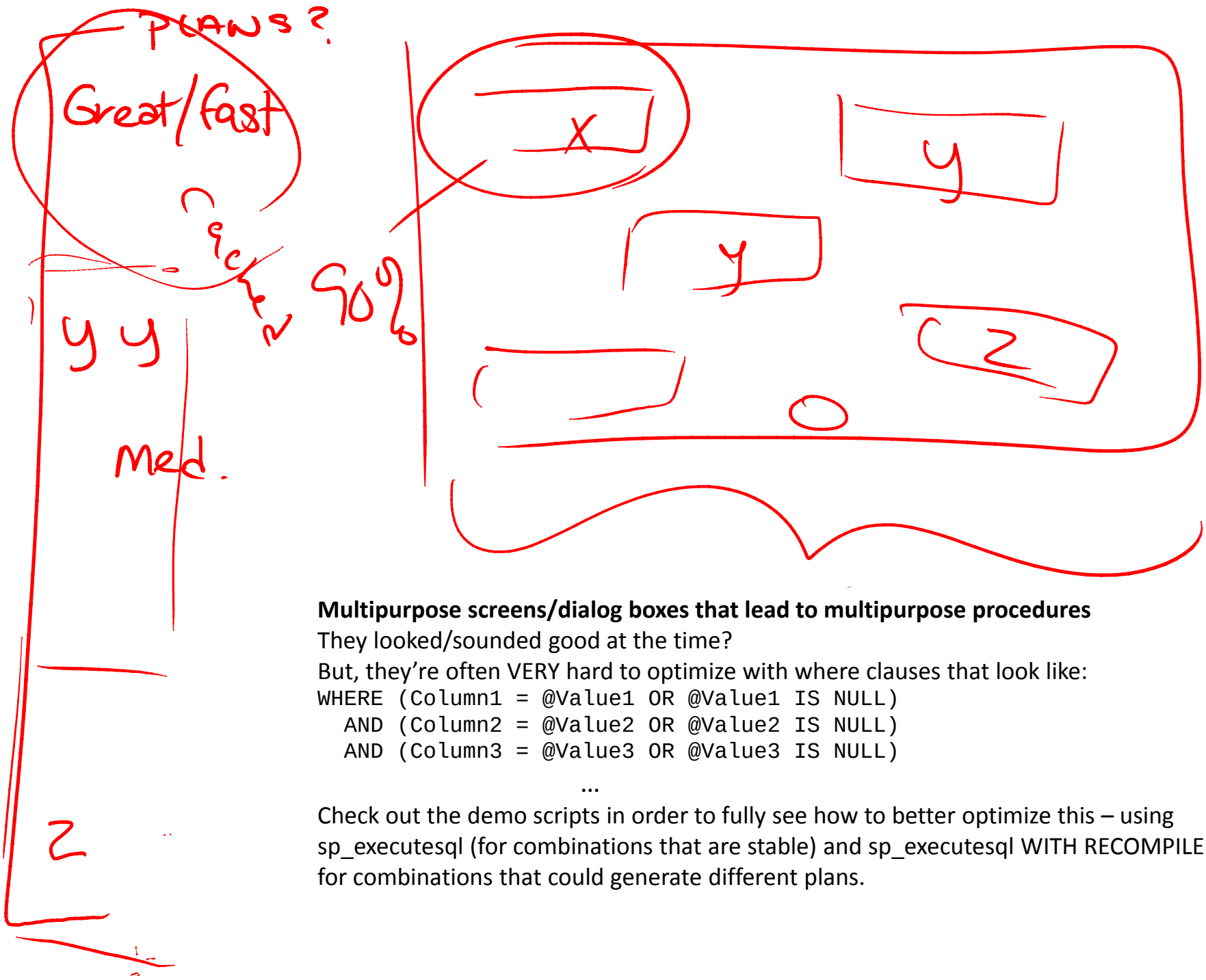


Using EXEC (string) [DSE] and/or sp_ExecuteSql [ES] inside of stored procedures

DSE and ES are not the same.

DSE is unknown until runtime. At that point the statement is treated as though it's adhoc. Because most statements are **not** going to be safe, the statement will end up being recompiled (and optimized) for each execution.

ES is forced parameterization. When a statement is stable you can use this to reduce compilation/CPU costs.



Multipurpose screens/dialog boxes that lead to multipurpose procedures

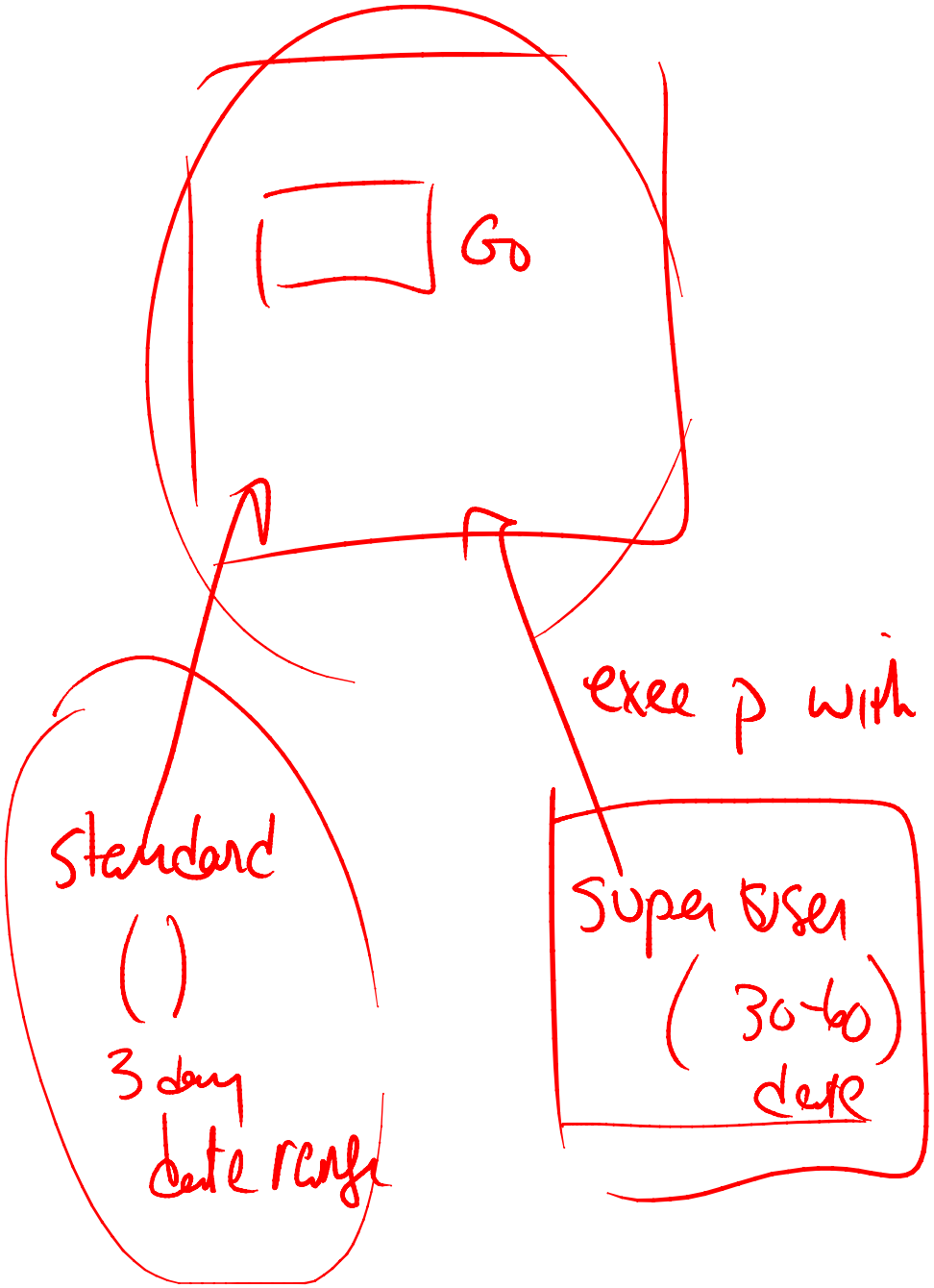
They looked/sounded good at the time?

But, they're often VERY hard to optimize with where clauses that look like:

```
WHERE (Column1 = @Value1 OR @Value1 IS NULL)
  AND (Column2 = @Value2 OR @Value2 IS NULL)
  AND (Column3 = @Value3 OR @Value3 IS NULL)
```

...

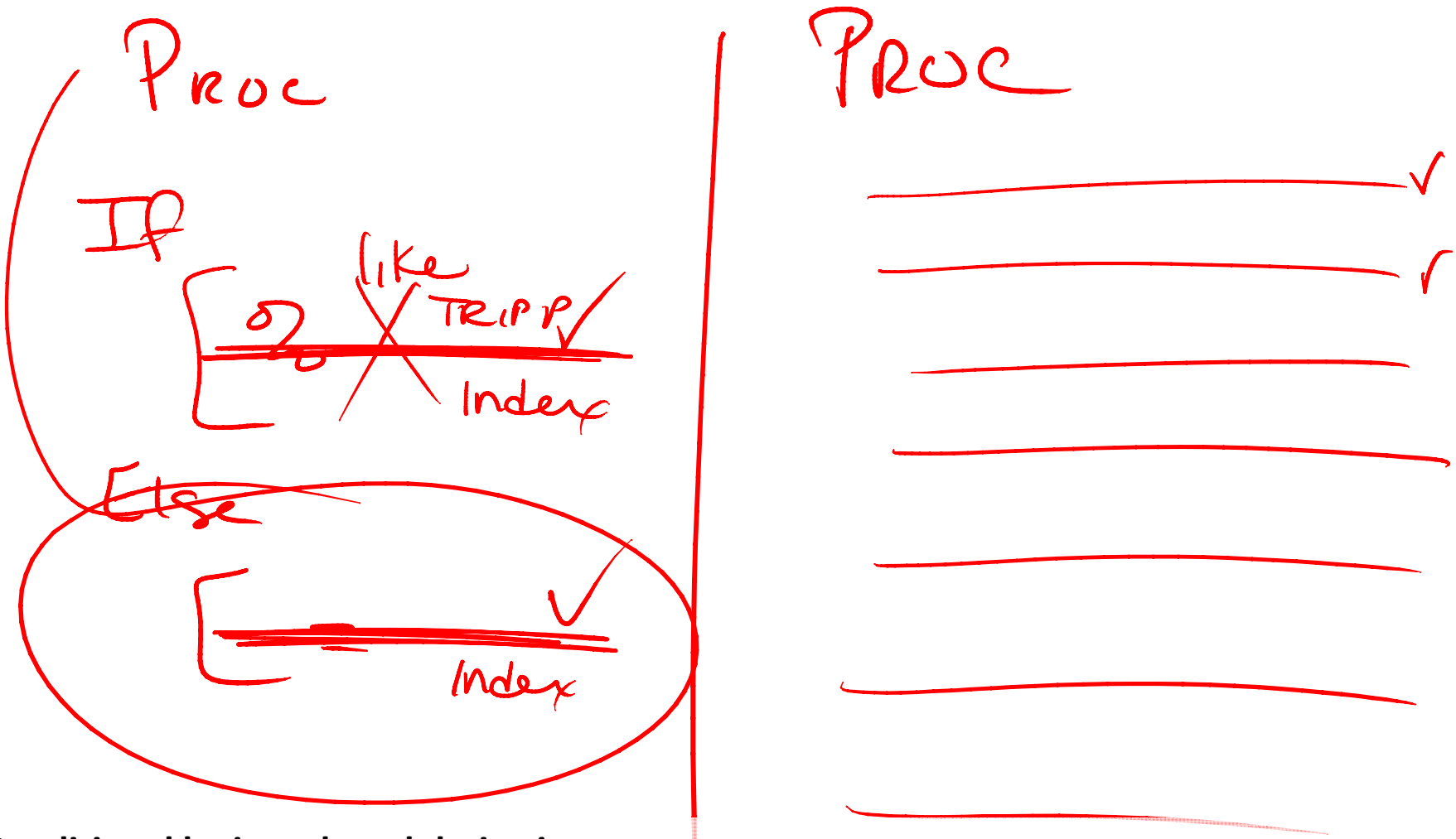
Check out the demo scripts in order to fully see how to better optimize this – using sp_executesql (for combinations that are stable) and sp_executesql WITH RECOMPILE for combinations that could generate different plans.



Options for different executions

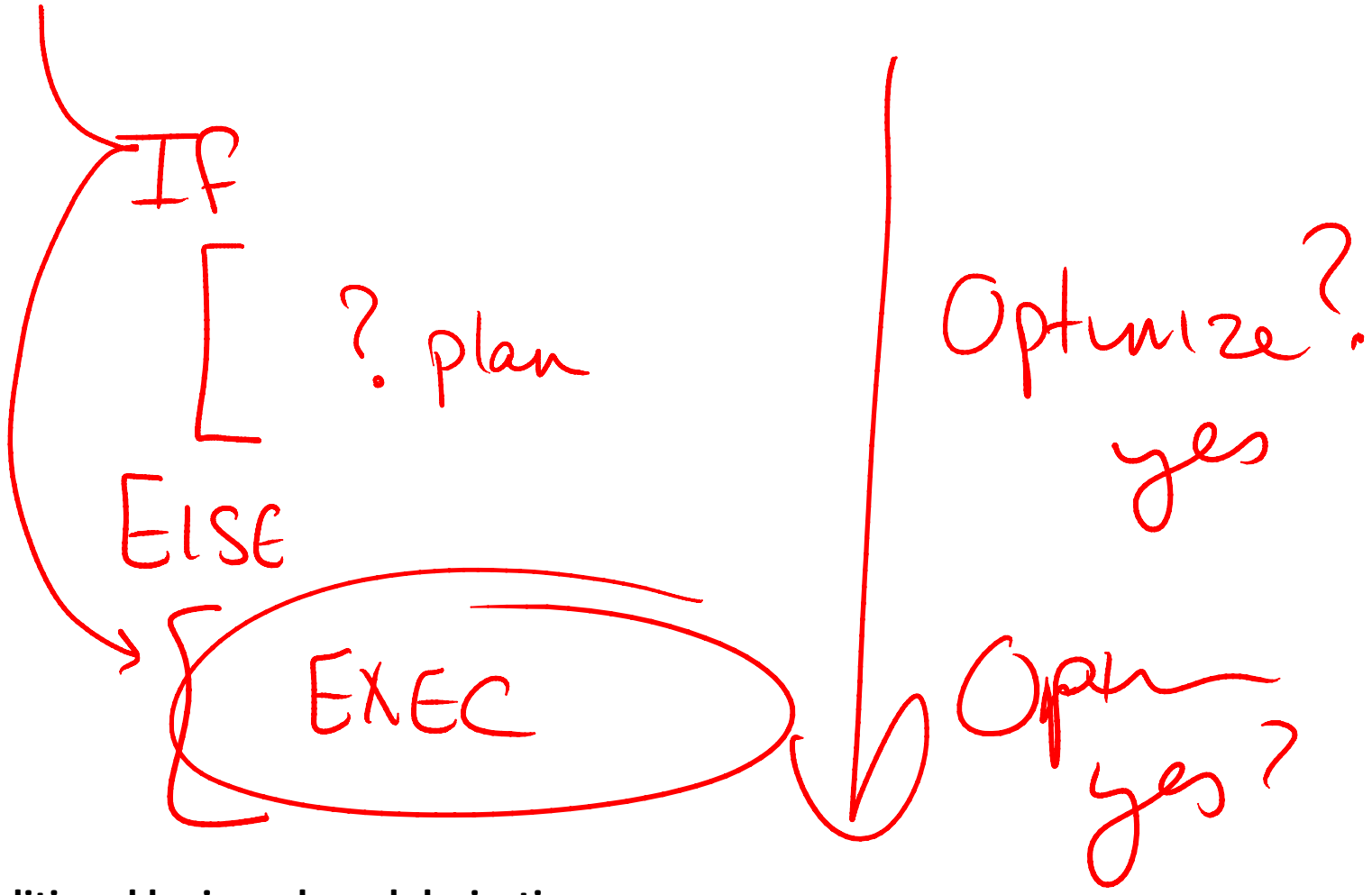
If we largely execute with "standard" users and we absolutely want THAT plan to be in cache and to be re-used then we could:

1) Use option (optimize FOR…) and use a literal that will generate a plan that's great for the "typical" scenario. However, the performance won't be great for the atypical (super user) scenario.
2) We could use EXEC for the typical scenario and EXEC proc WITH RECOMPILE for the super user scenario. The benefit of this is:
    1) We get a cached plan for the typical user
    (no CPU/plan stability)
    2) We get a specific plan for the atypical user (the super user) as opposed to a possibly inefficient plan (so, we have to compile but we're only compiling this one type of plan)



Conditional logic and modularization

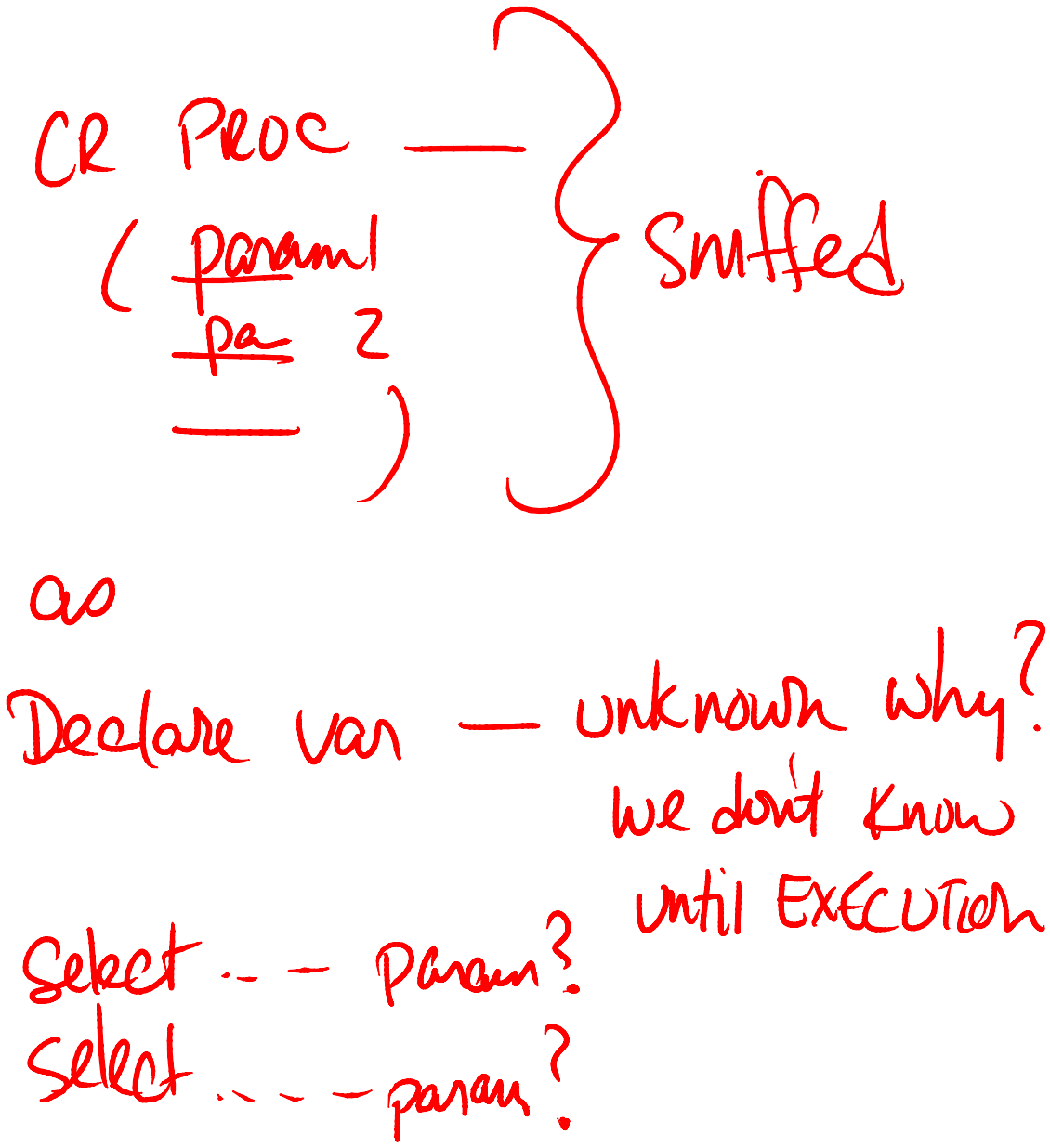
This is another way of looking at the same thing from the prior diagram. Really, don't think of the "logic" in your procedure; think only of the procedure as a list of statements (regardless of conditional logic). SQL Server tries to optimize ANY statement that's "*optimizable*" with the parameters that were supplied (regardless of those specific parameters apply for *that* execution). This is another case where you can end up with an inefficient plan because of parameter sniffing.



Conditional logic and modularization

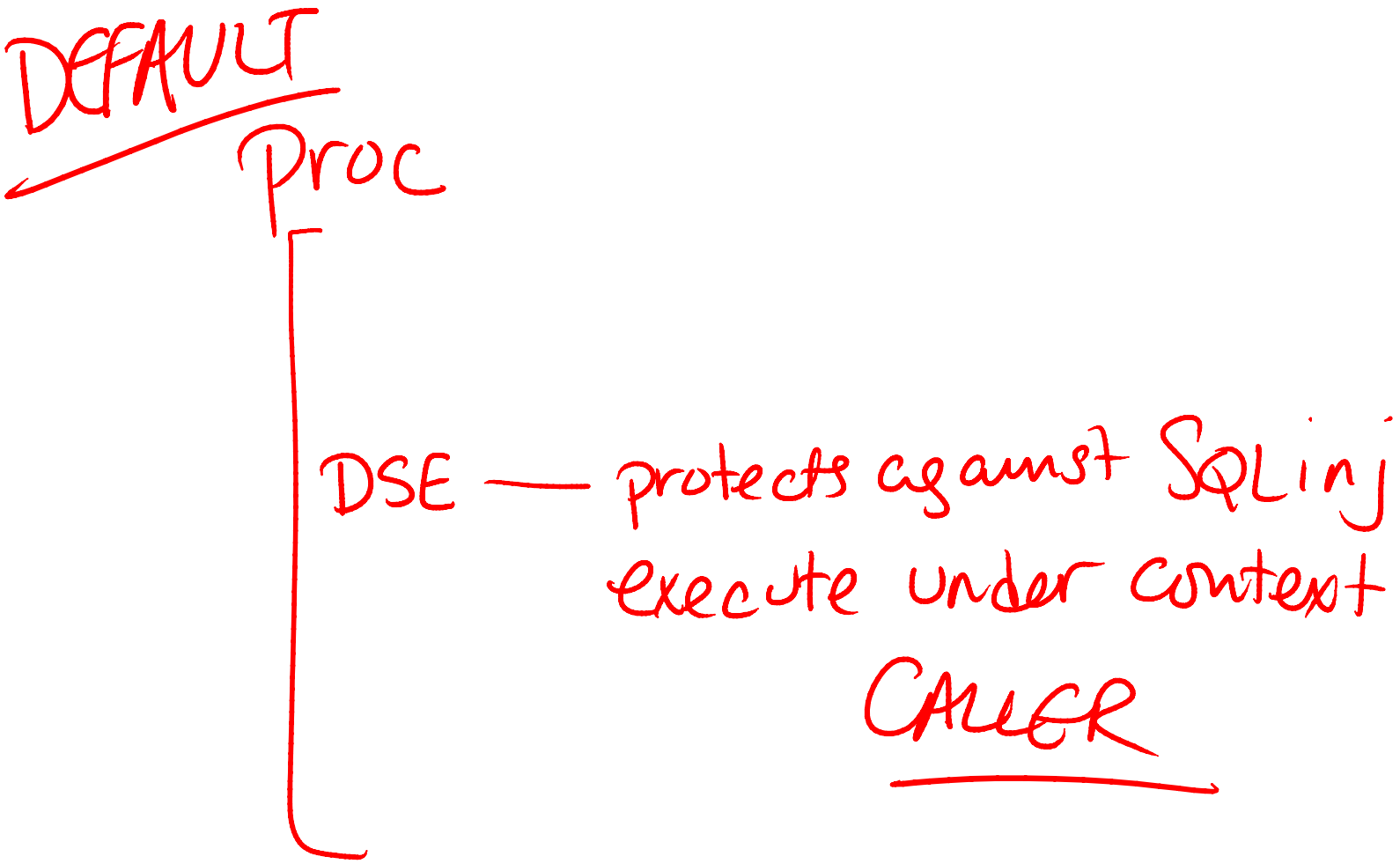
Creating branching logic for different types of parameters seems logical but because SQL Server optimizes the process of optimization (where they try to optimize anything that's optimizable), you can often end up with a plan that's not ideal.

You are much better off breaking that code into smaller subprocedures – SQL Server will never step into a subprocedure unless it's executed and in this case it will only be executed with exactly the right parameters.



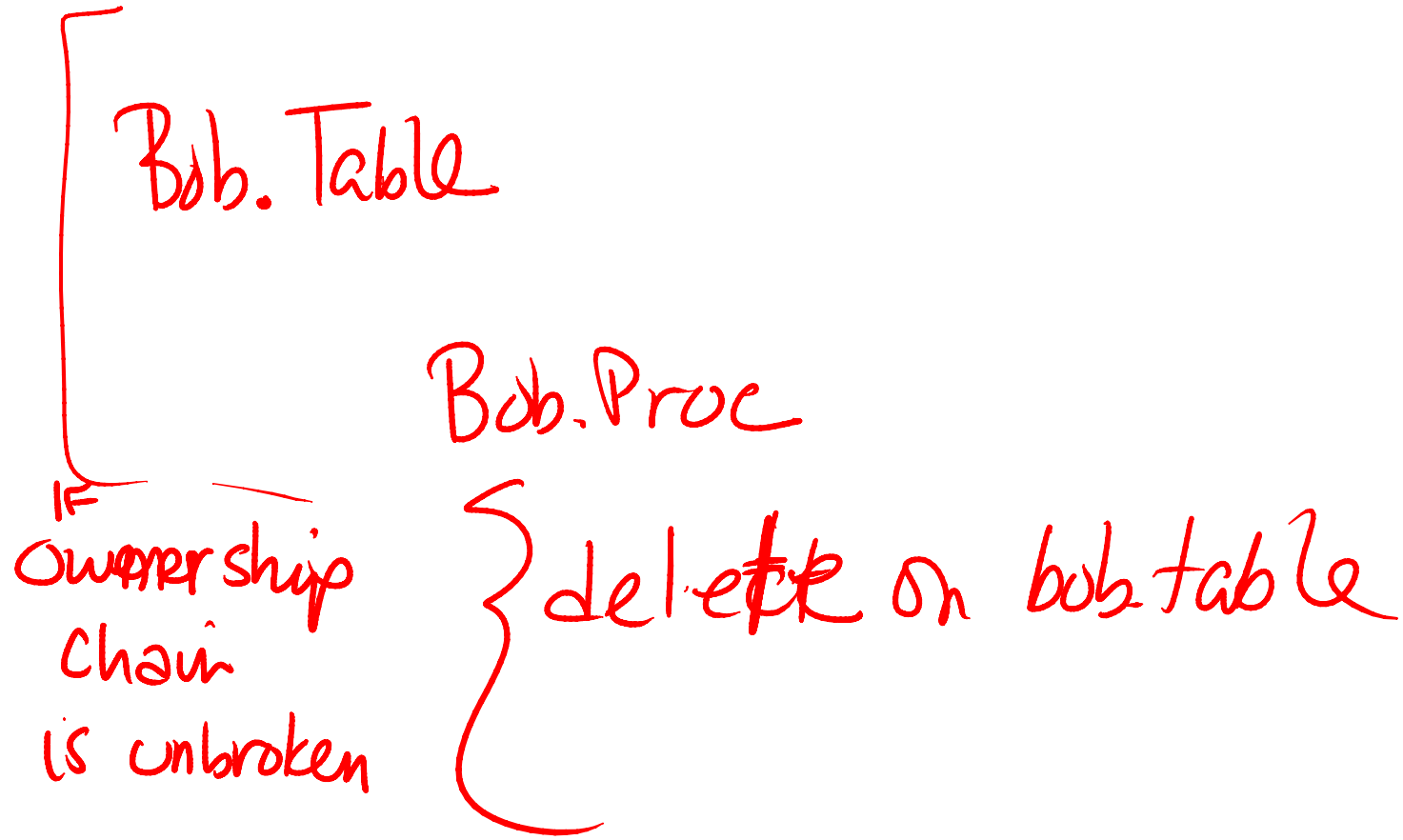
Parameters are "sniffed"
SQL Server uses parameters to optimize. Sniffing implies that SQL Server uses the histogram to evaluate the data selectivity. For the FIRST execution parameter sniffing is great. It's the subsequent executions that can suffer from parameter sniffing (and therefore end up with PSP = parameter sniffing problems).

Variables are "unknown"
Variables are only known at runtime as the statements execute and the variables are assigned. As a result, they are unknown during optimization. With an actual value, SQL Server cannot use the histogram. Instead SQL Server uses the density vector for the "average" numbers of rows that meet that criteria.

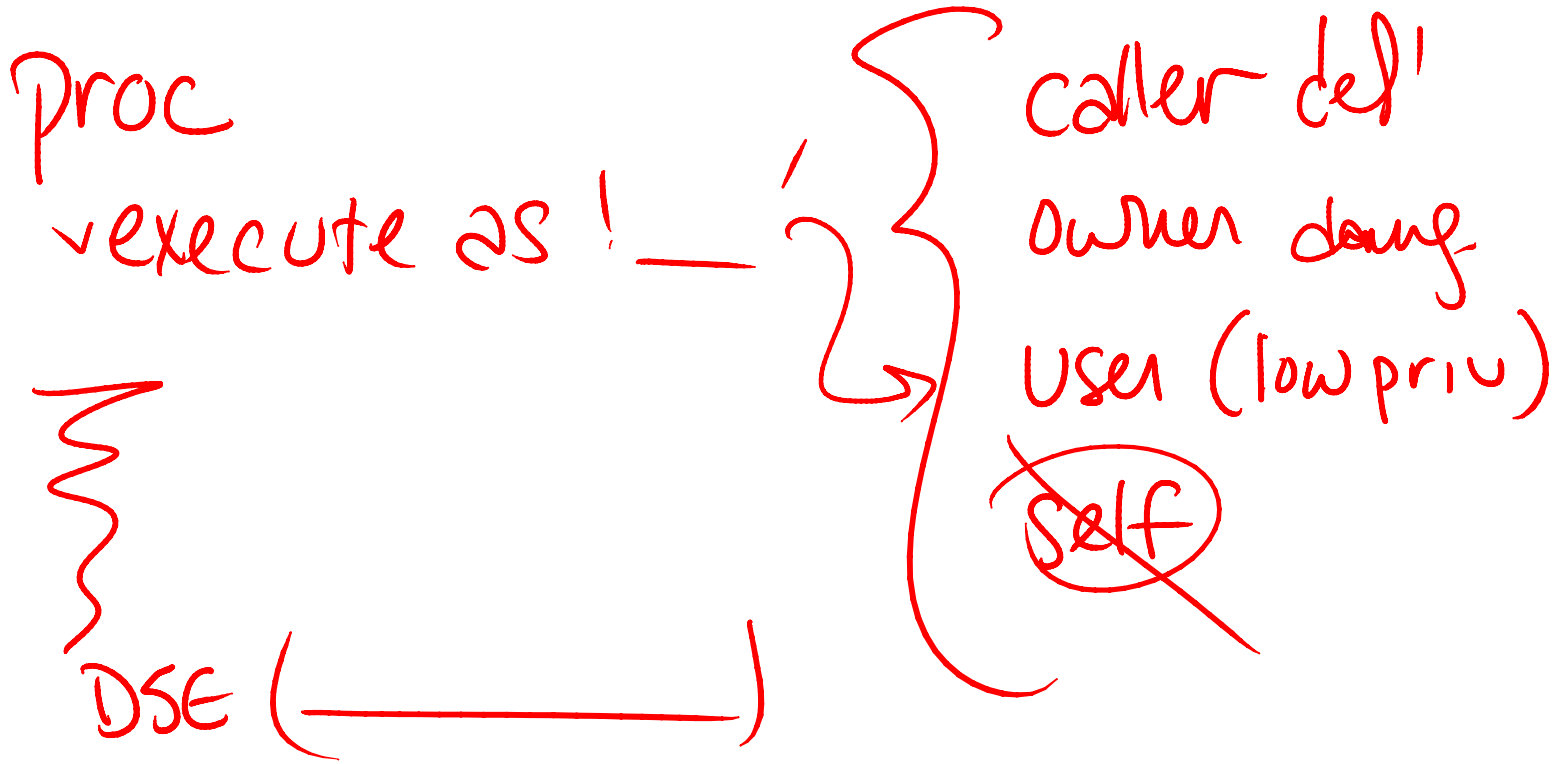


Dynamic String Execution and SQL Injection

TO prevent SQL Injection, SQL Server requires that strings execute under the context of the caller.



Without Dynamic String Execution permissions flow through the ownership chain
The idea is that when the ownership chain is unbroken then direct permissions (for example, to do a delete) are not necessary. As an owner you are giving rights to a process; this procedure can be protected with more business rules/logic and even reduced/isolated to specific rows. The end user does NOT need direct delete permissions to be able to execute the procedure as long as they've been given execute rights on the sproc.



Dynamic String Execution, EXECUTE AS and SQL Injection

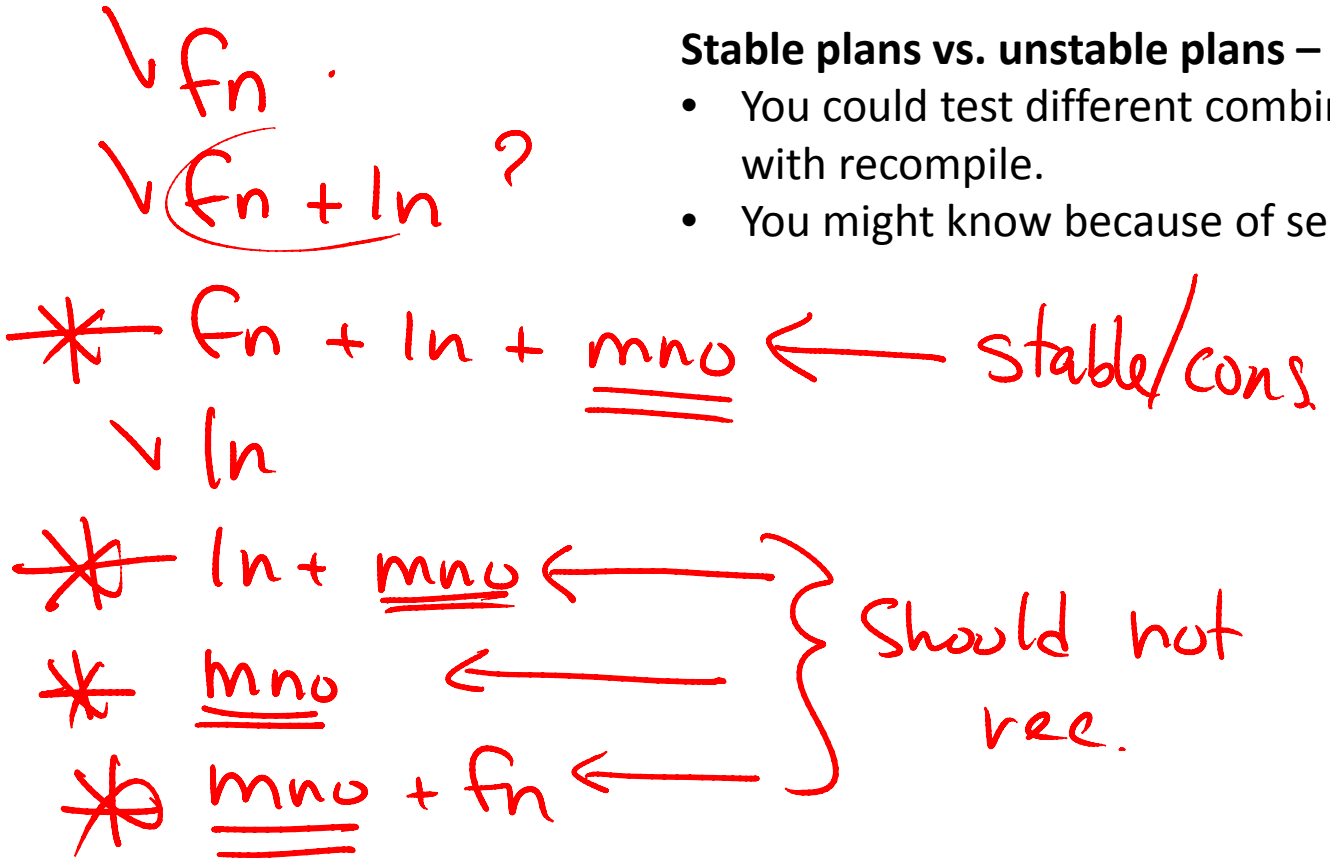
People complained that granting permissions directly to the user was a problem… so, they wanted an alternative. Enter: EXECUTE AS.

While it solves the problem of permissions, it adds more potential for SQLInjection.

So…….. Check out these posts:

Little Bobby Tables, SQL Injection and EXECUTE AS

"EXECUTE AS" and an important update your DDL Triggers (for auditing or prevention)

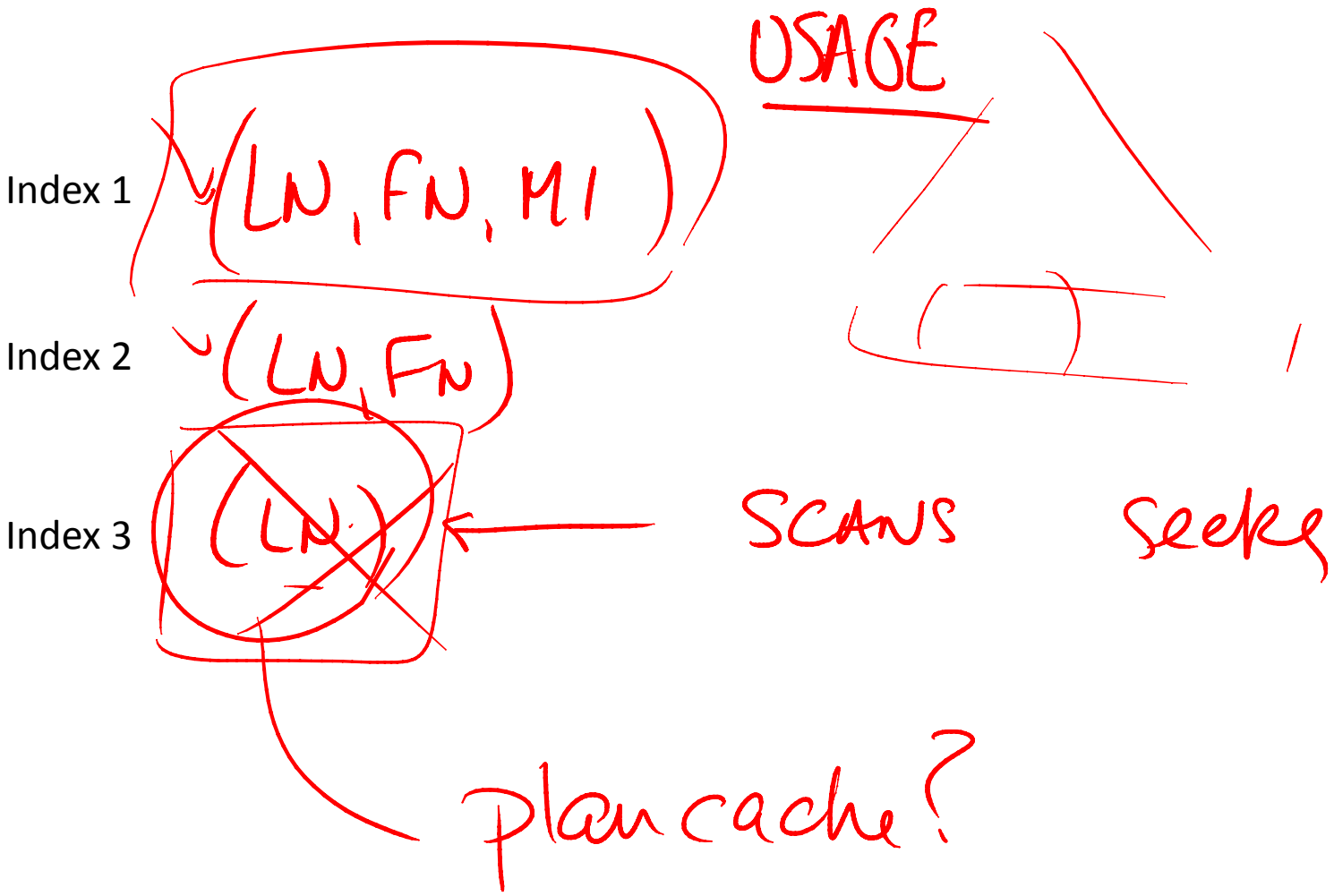


Stable plans vs. unstable plans – how do you know?

- You could test different combinations using execute with recompile.
- You might know because of selectivity?

M9s38 – options that better balancing recompiling too much vs. not recompiling enough

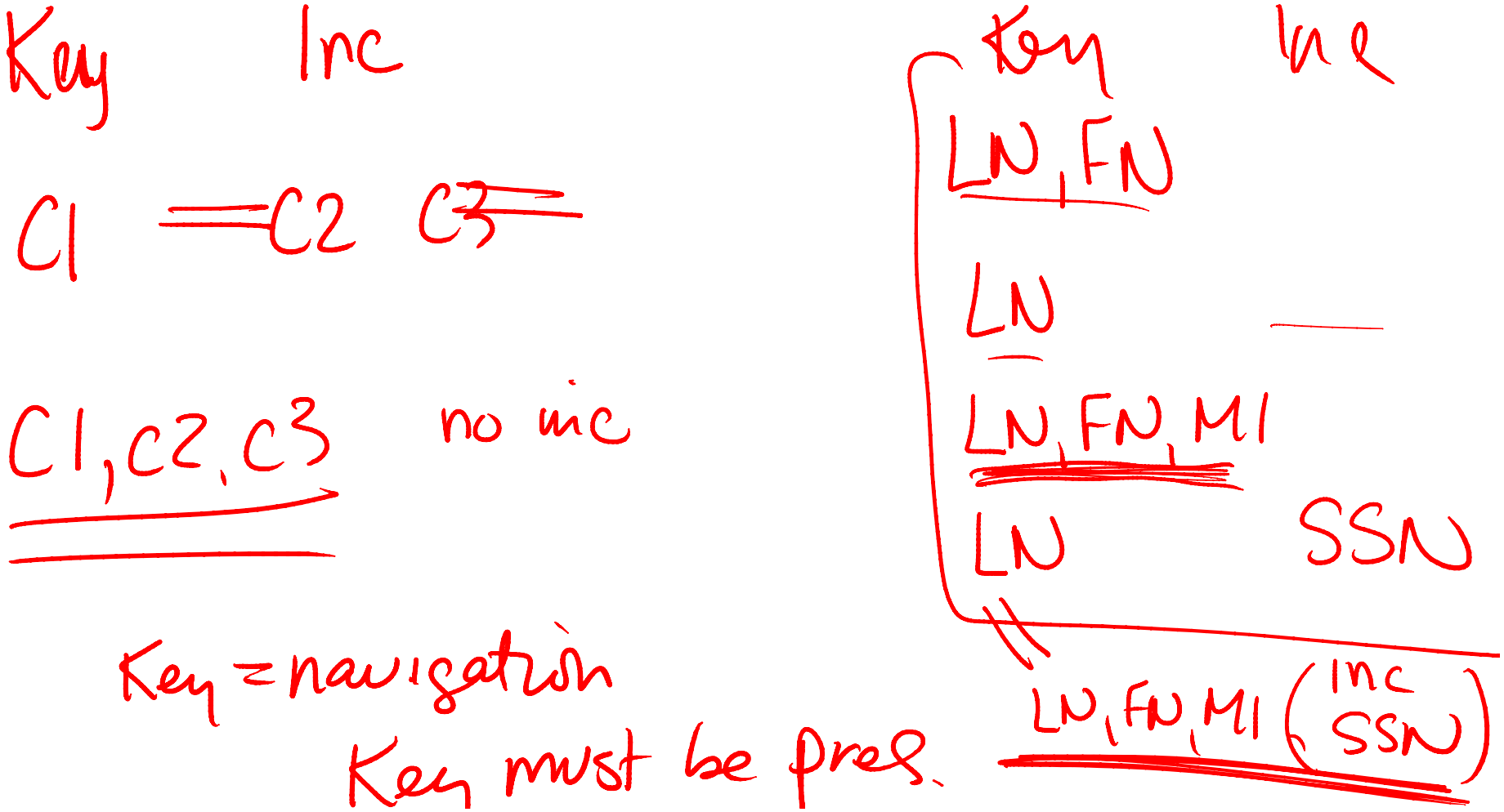
This is from the multipurpose procedure demo. There are 3 parameters and 7 possible combinations that can be submitted. Instead of adding OPTION (RECOMPILE) to the statement (which doesn't always work [remember, it has a very checkered past]) you can build the statement dynamically and then programmatically (by setting a RECOMPILE flag to 0 or 1) decide whether or not OPTION(RECOMPILE) should be added to the dynamically constructed statement. Then, you can use sp_executesql to place ALL seven statements in cache. Those without the OPTION(RECOMPILE) clause will be cached. Those with – will get a plan on each execution.



Should you drop redundant indexes? It depends. ☺

It's TRUE that query that uses a left-based subset of one index (let's say that a query uses index 2) that is COULD use a wider version of the same (left-based) index. It could use Index 1 instead of 2.

The problem is that there could be specific uses that warrant the smaller version. Questions to ask: Is the smaller version used by scans? Are they executed VERY frequently? How important are they?

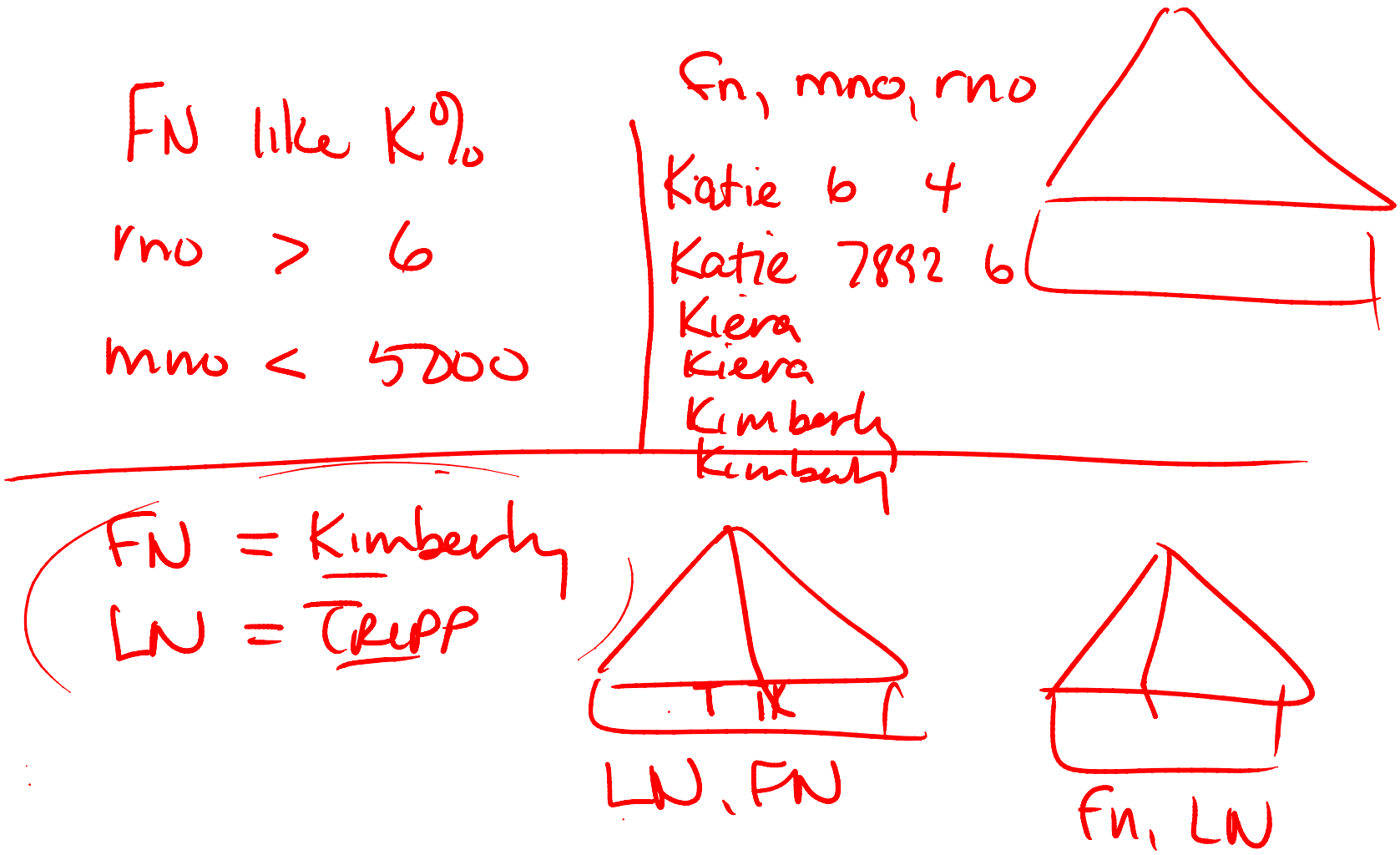


Consolidation?

You can CONSIDER removing indexes that are left-based subsets of others (in terms of the key). However, be sure to preserve the key as that's used for navigation/seeking. The columns that are included can be combined in any order.

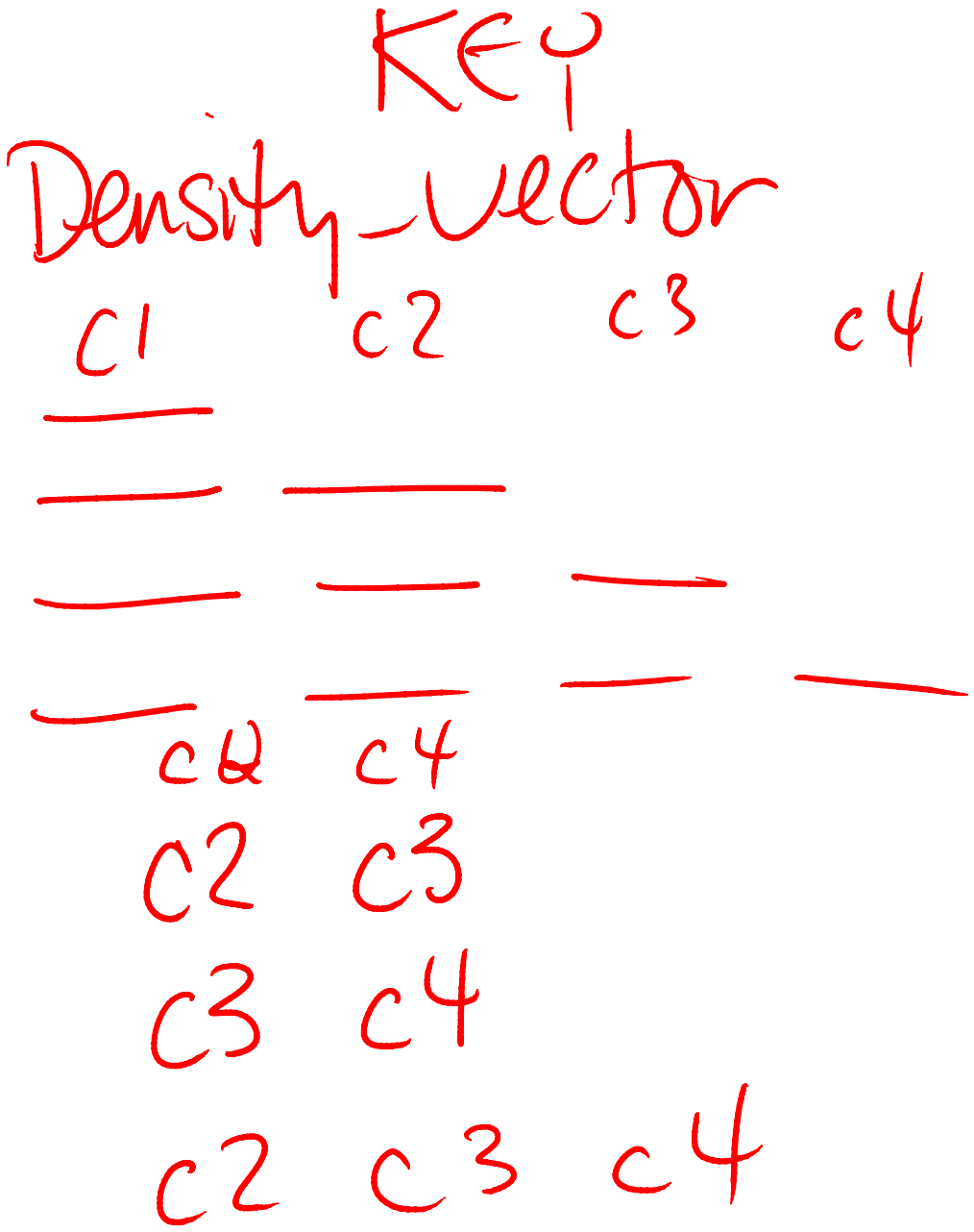
Index Health – Checking for fragmentation

For a limited mode query against sys.dm_db_index_physical_stats, SQL Server uses a scan of the first intermediate level (index level 1) to analyze for the value: avg_fragmentation_in_percent. If your analysis/rebuild scripts use this to determine fragmentation than you'd be better off just doing a limited scan. What SAMPLED and DETAILED offer are page-specific details (page density, forwarding pointers, etc.) and if you're NOT using those (most people aren't using these during their automated maintenance process). So, don't waste time with a SAMPLED or DETAILED can during your maintenance window!



Adding more indexes

Here we were discussing the order of columns. If there are range-based predicates than it doesn't usually matter which column is second. You tend to want to put the most selective first (in terms of the PREDICATES supplied). If all of the predicates are equality-based then it doesn't really matter. You're best ordering them by the LEFT-based combination that's used most commonly.



Multi-column, column-level statistics
Only DTA recommends multi-column, column-level statistics…
If you are about to trust (and create) a "green hint" recommendation (which comes from the Missing Index DMVs) then you might want to first run the query through DTA. If you end up creating the index that DTA recommends then you should also create the recommended stats (for that same table). These statistics can be helpful to the optimizer to better understand non-left-based subsets of the index for other possible uses.

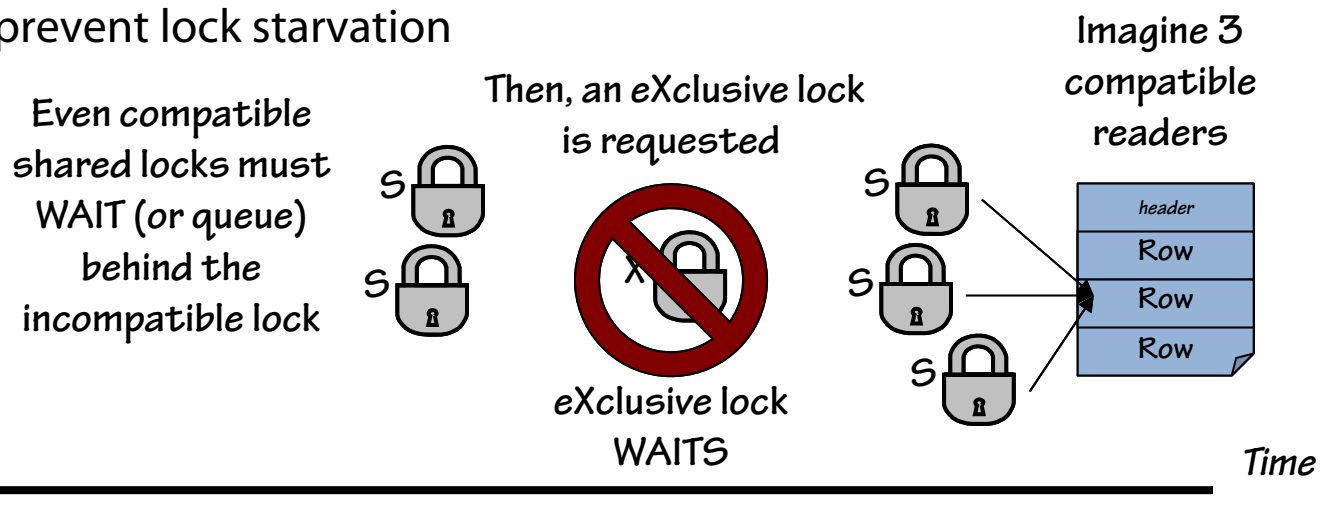
SQLskills Immersion Event
IE1: Internals and Performance
(reference slides)

Module 3: Locking and Blocking

Kimberly L. Tripp
Kimberly@SQLskills.com

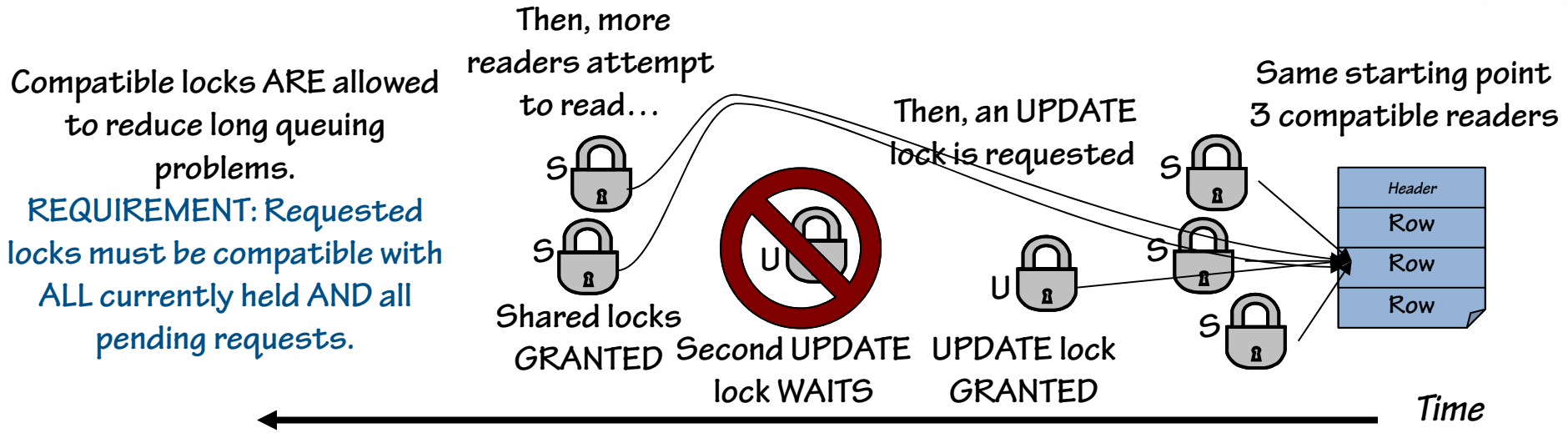
Blocking – Is it Really a Problem? It depends…

- **Locks guarantee consistency**
- **First incompatible lock request waits**
- **Once an incompatible lock request is made then pending locks will wait even if they're compatible with locks currently held**
  - Special case WHEN the requested lock IS compatible WITH ALL pending requests…
- **Primary reason for the locks to WAIT**
  - Why? To prevent lock starvation



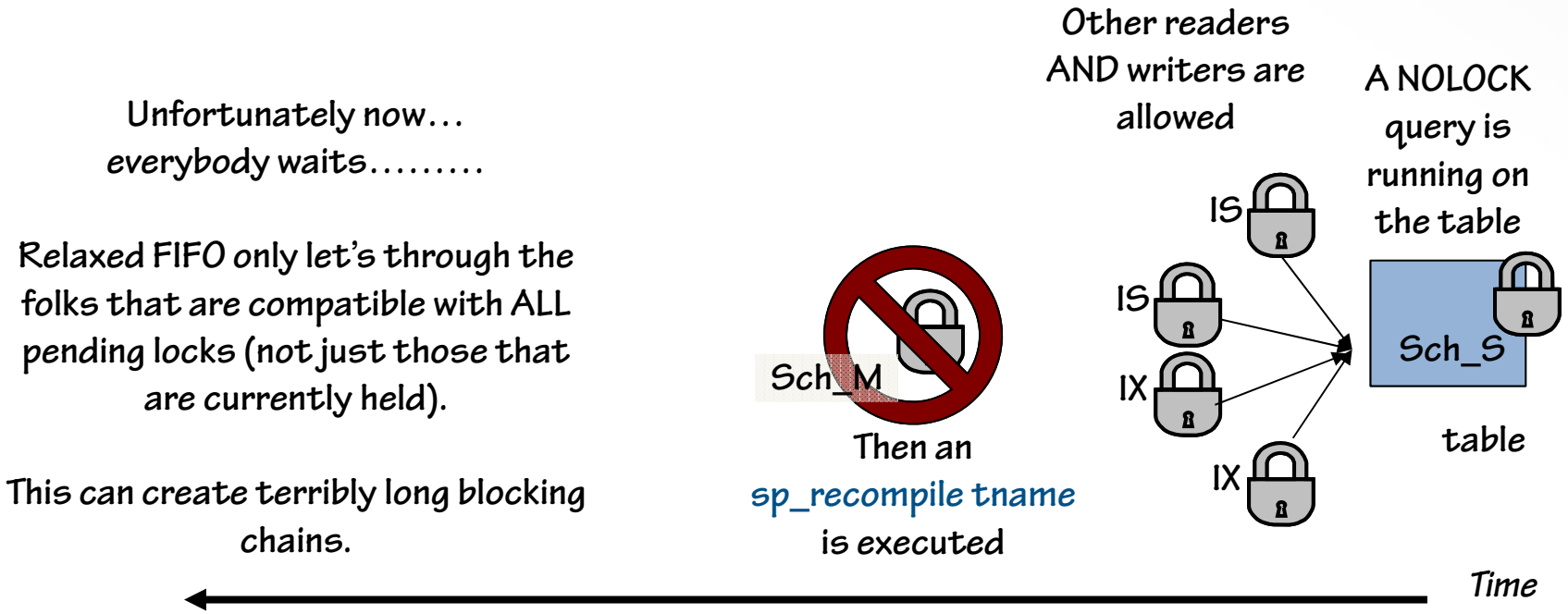
Relaxed FIFO
To Reduce Really Long Blocking Chains

- http://blogs.msdn.com/b/psssql/archive/2009/06/02/sql-server-lock-manager-and-relaxed-fifo.aspx (http://bit.ly/NWAyVC)
- An update is incompatible with another update
- The requested (second) update will wait behind the current update…
- What about shared locks if ONLY update locks?
  - ALLOWED



Nasty Blocking CHAINS
Relaxed FIFO isn't Always Enough

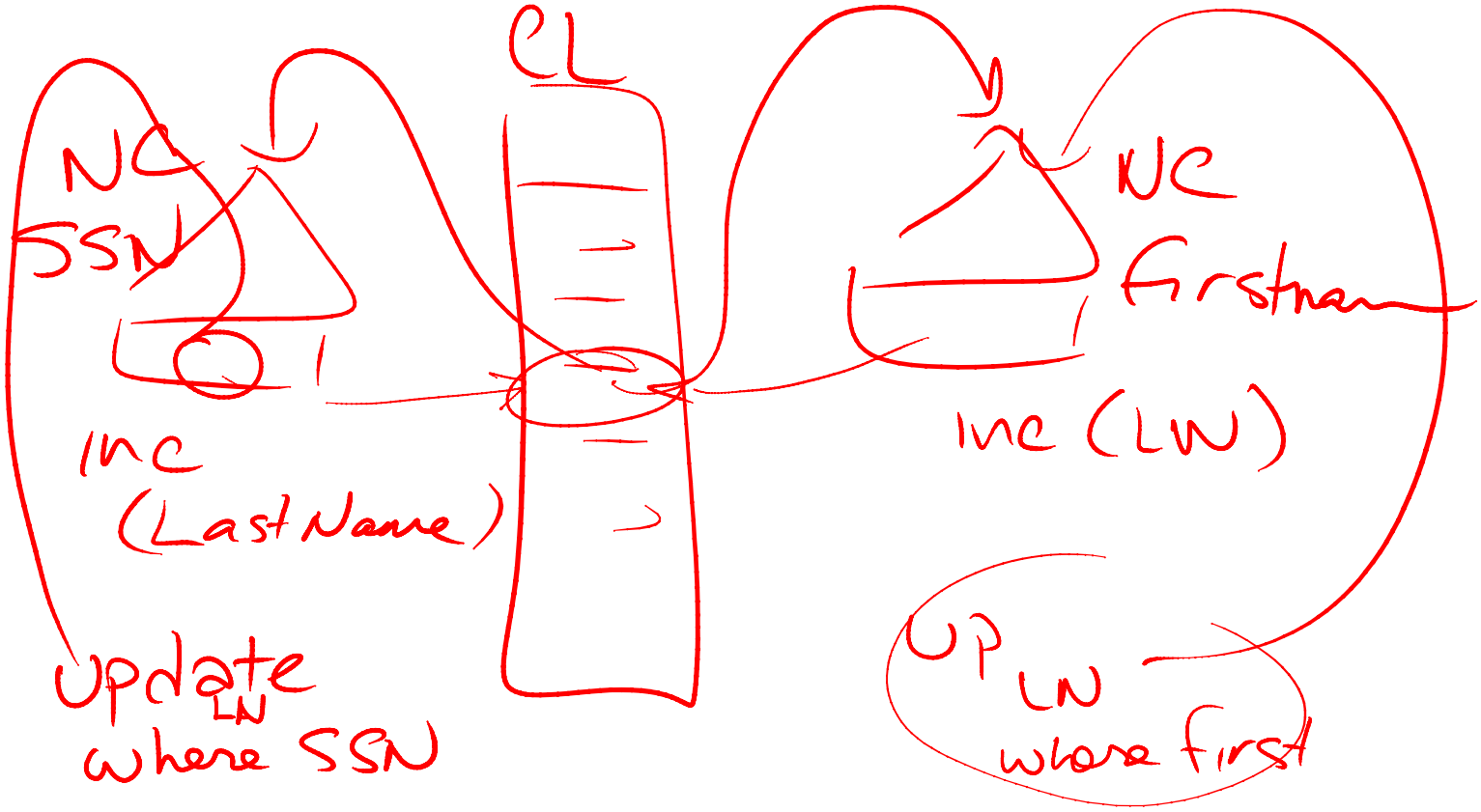
- Imagine someone's running a NOLOCK query against a very large table (the query takes 4 minutes to run [ok, not that nasty])
- This query's running off-hours but the standard is to use NOLOCK
- An `sp_recompile tname` is requested



SQLskills Immersion Event
IE2: Performance Tuning

Module 12: Collecting Performance Data

Jonathan Kehayias
Jonathan@SQLskills.com



M3s55

This picture described a deadlock that can occur when multiple indexes exist on a single table and they INCLUDE columns that are being updated.

```
CL = clustered table
Left side = index on SSN INCLUDE (lastname)
Right side = index on firstname INCLUDE (lastname)
```

Imagine an update to firstname that uses WHERE SSN = running at the same time as an update to firstname that uses WHERE Lastname = (let's not debate how bad this latter update is ☺).

These updates are MUCH more likely to create deadlocks because of how locking works within indexes. To reduce the potential here – I often use sp_tableoption.