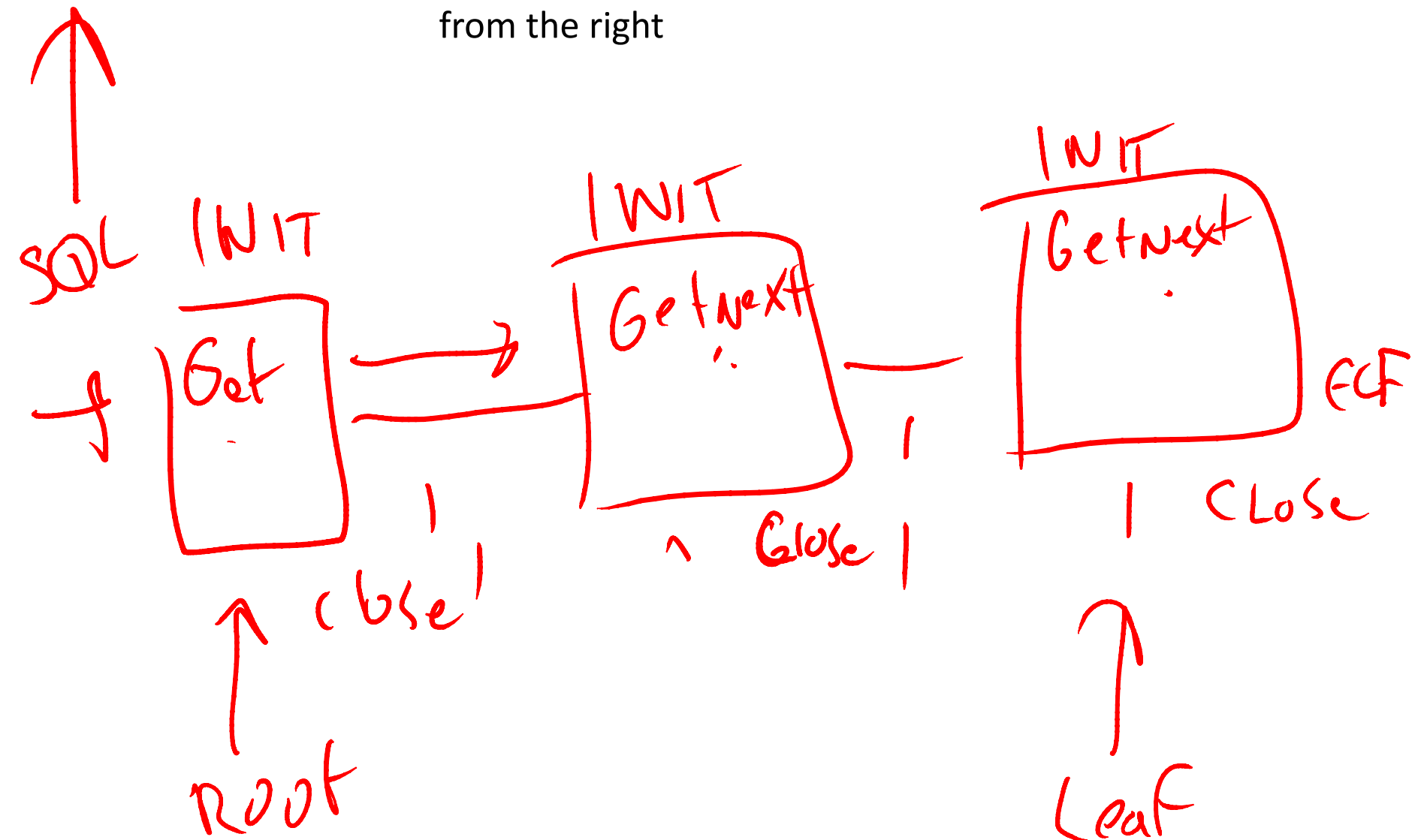


## Module 7: Query Plan Analysis

Row-Execution Mode – showing the flow of one row at a time within a tree, controlled from the left, data flowing from the right



## Module 7: Query Plan Analysis

Ideally cardinality estimates come from statistics and constraint information.

In absence of this, the QO has to “guess” and that means heuristics need to be used.

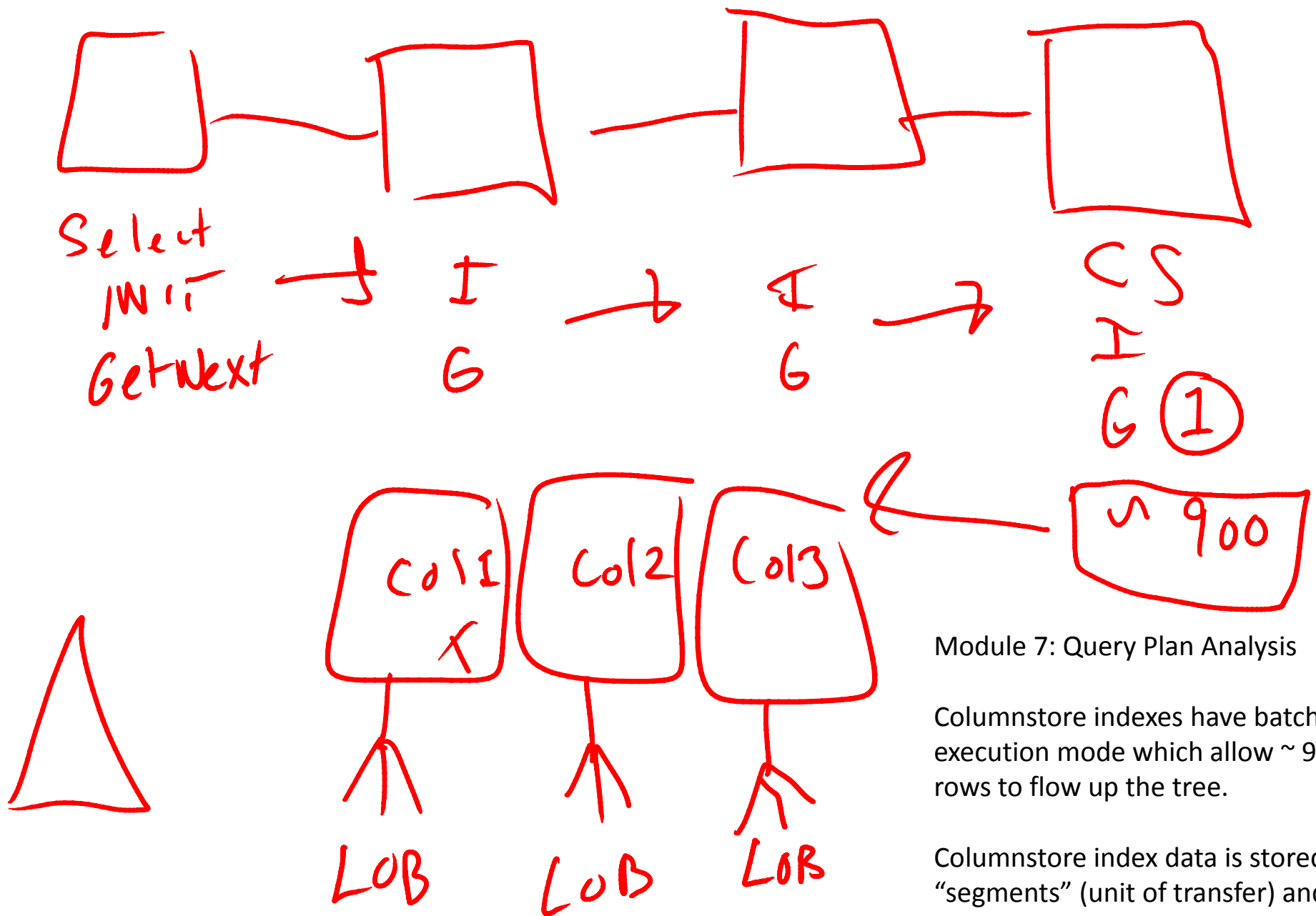
Statistics

Unique constraints

FKs

Heuristics

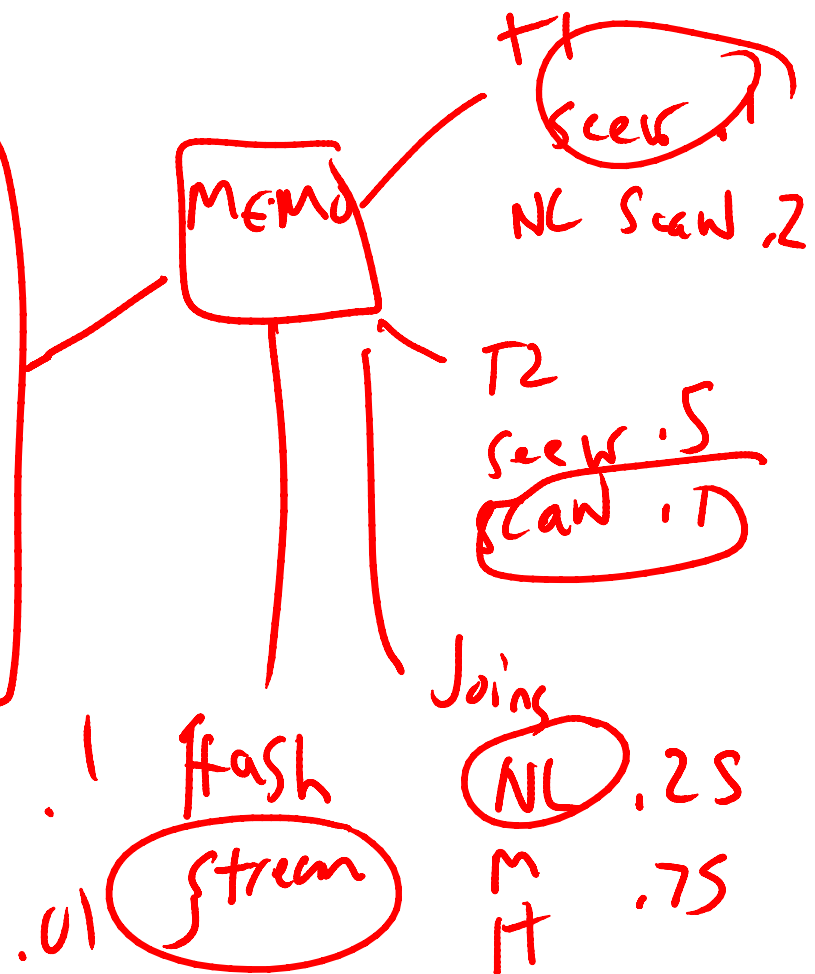
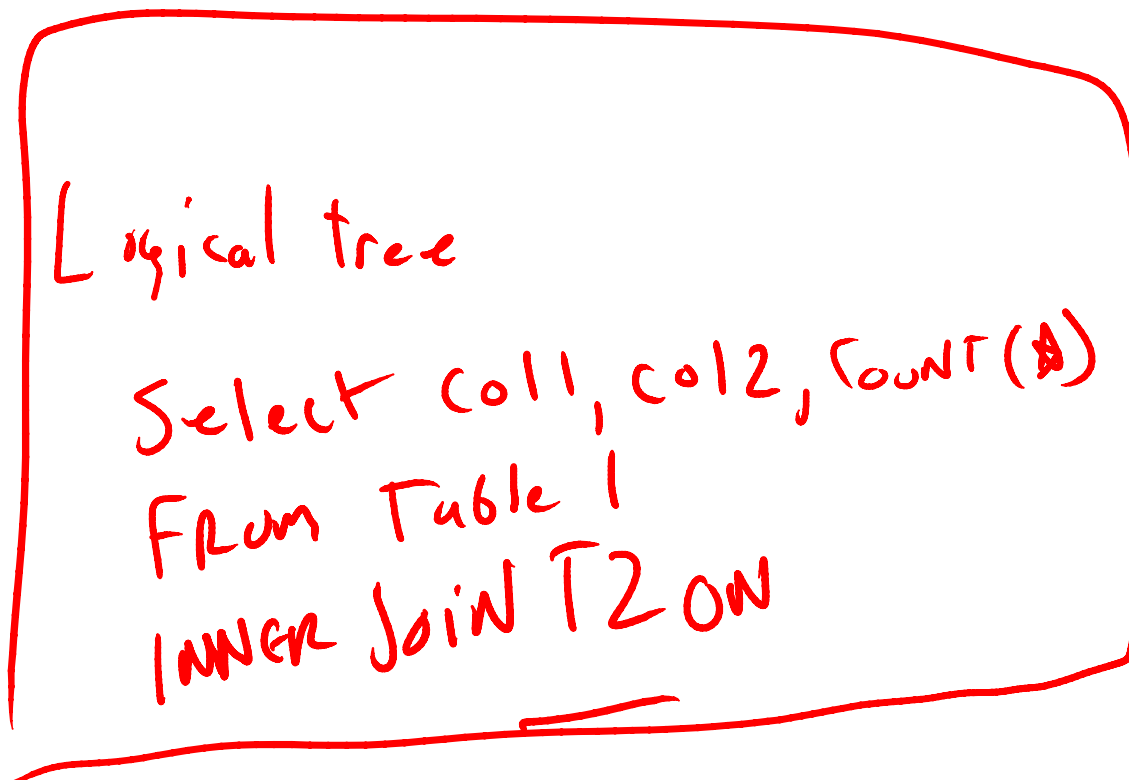
S



## Module 7: Query Plan Analysis

Columnstore indexes have batch-execution mode which allow ~ 900 rows to flow up the tree.

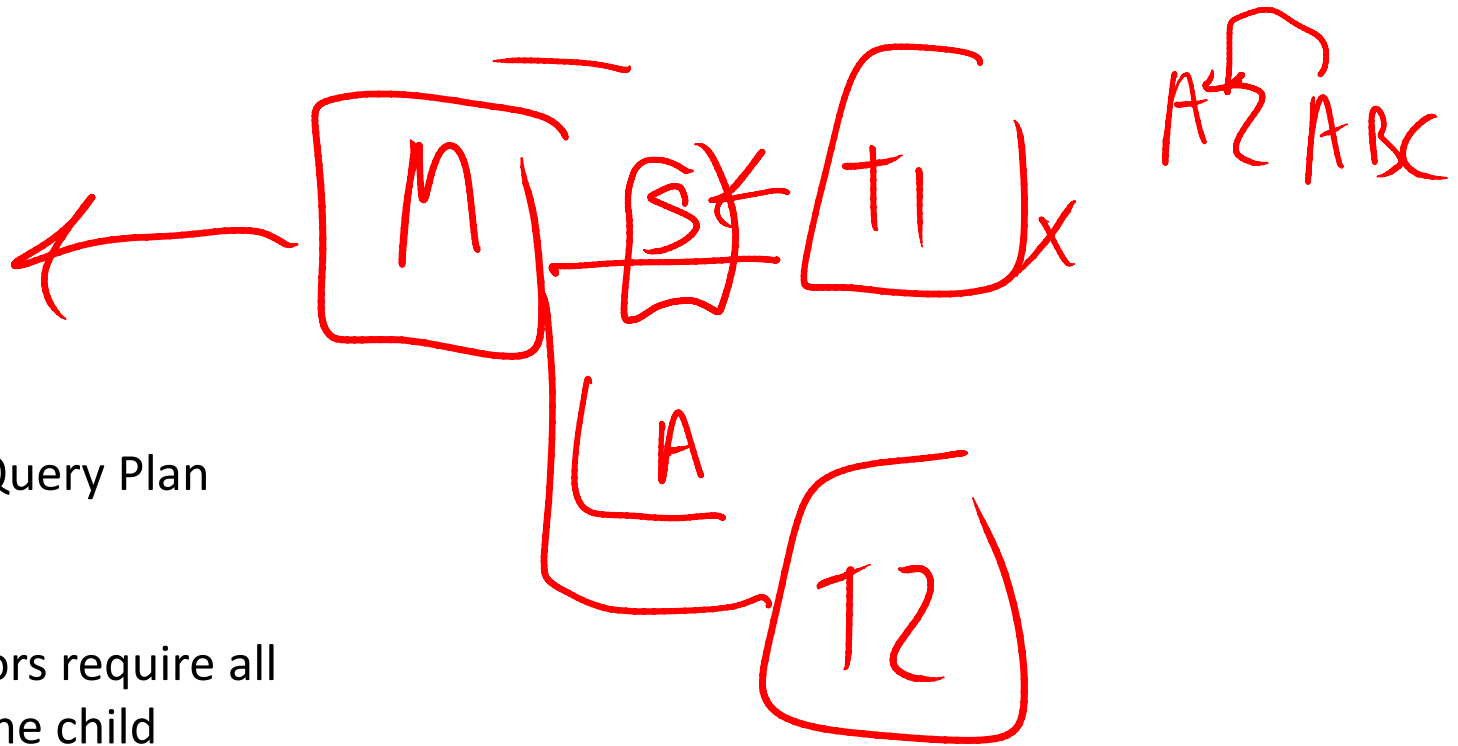
Columnstore index data is stored in "segments" (unit of transfer) and that associates to multiple LOB pages per segment.



Final  
plan

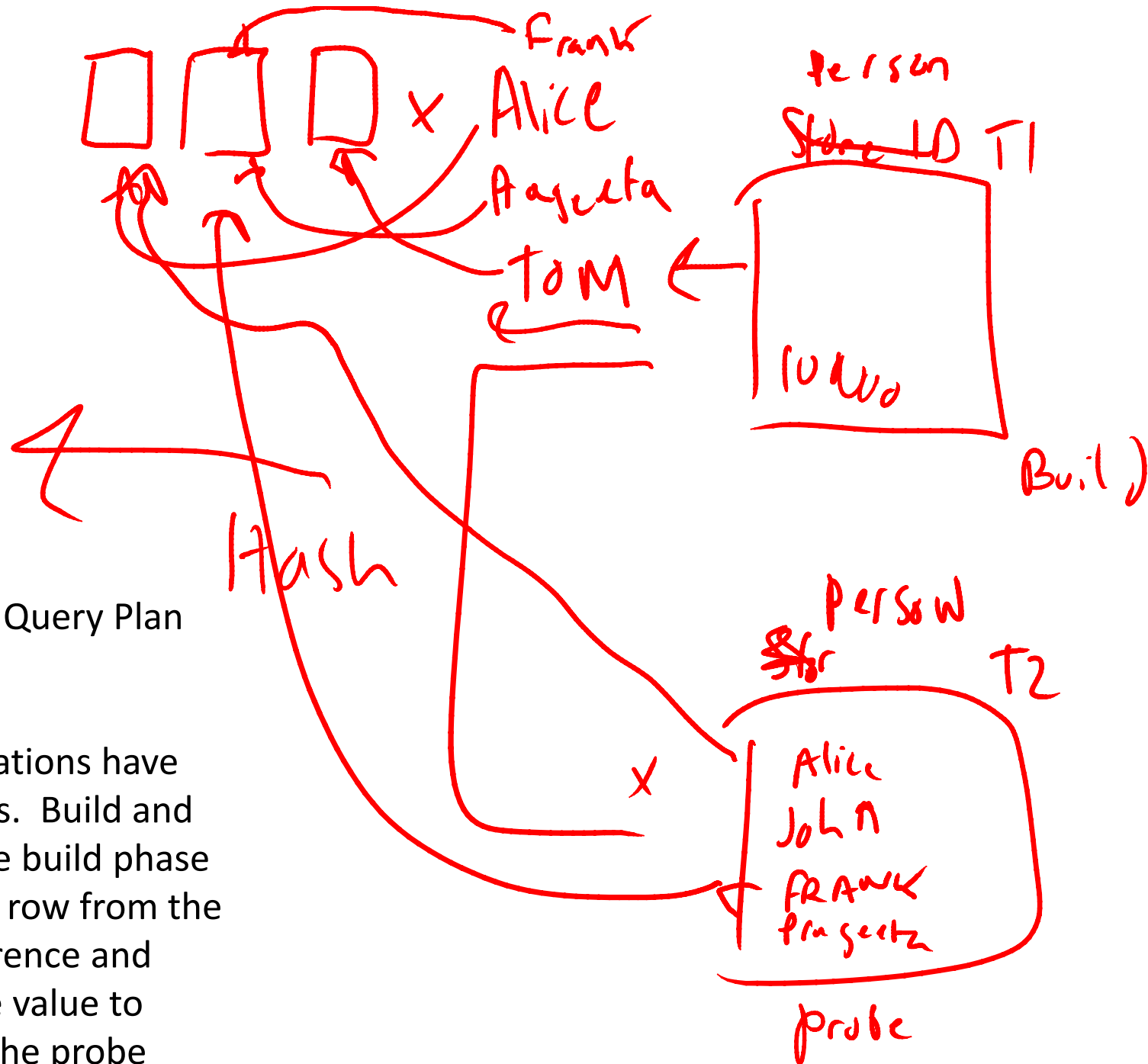
Module 7: Query Plan Analysis

During the optimization process,  
the QO tracks plan subtree  
alternatives using a "memo"  
structure



## Module 7: Query Plan Analysis

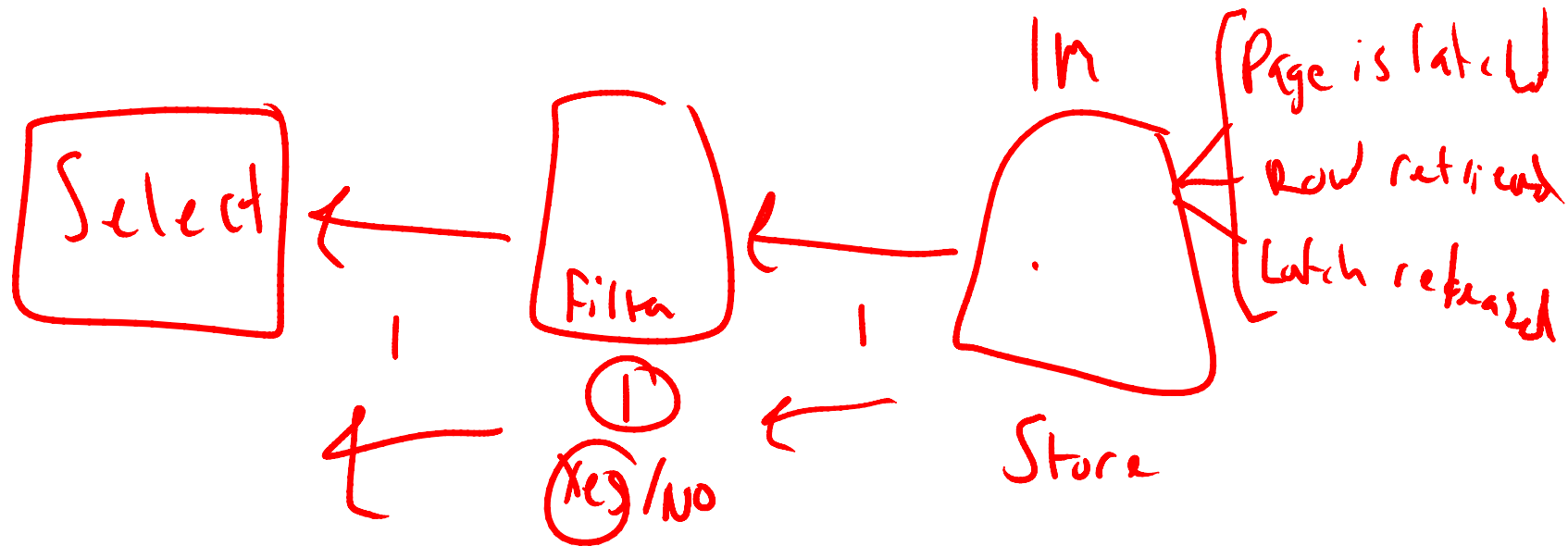
Sort operators require all rows from the child operator before rows can begin flowing to it's own parent operator. For example – values A, Z, A, B, C, Z – need to be sent to the Sort first before flowing up the tree



## Module 7: Query Plan Analysis

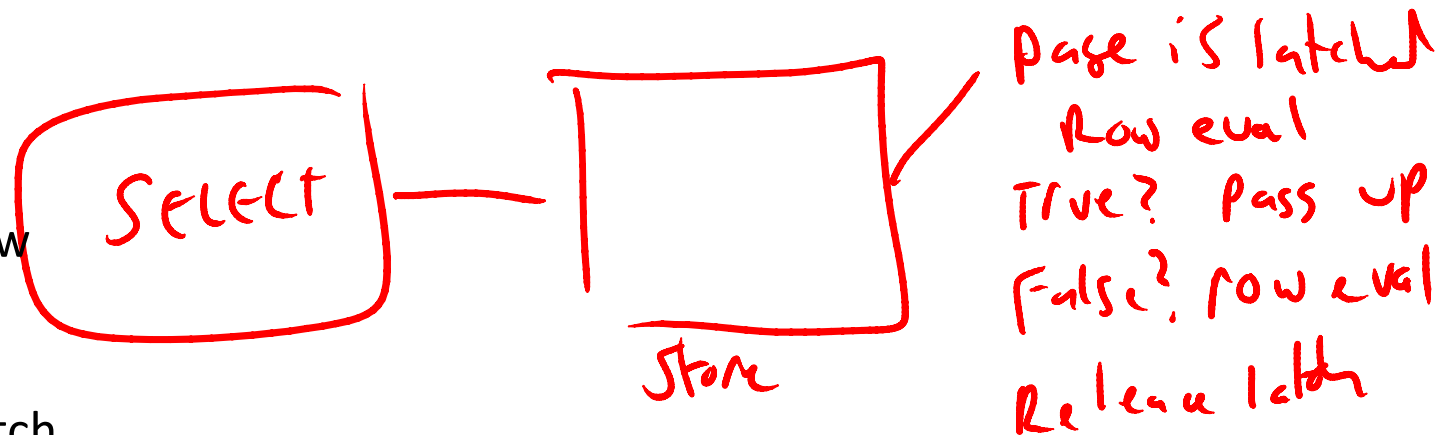
Hash operations have two phases. Build and Probe. The build phase takes each row from the outer reference and hashes the value to buckets. The probe phase doesn't begin until the build phase is complete.

StoreID = 1



## Module 7: Query Plan Analysis

Pushed predicates allow for efficient latching – compared to separate filters which require latch operations per row, even when the row doesn't pass the predicate



## Module 7: Query Plan Analysis

You might see a bitmap operator prior to a Hash Match operation. The Bitmap operator leverages the “build” phase, creating a filter which can then be pushed to the probe phase inner input, narrowing out rows that have no chance of matches

