Executing nested queries

Goetz Graefe
Microsoft SQL Server

Executing nested queries

- Motivation – scalability
- Speeding I/O – asynchronous I/O
- Avoiding I/O – caching, merged indexes
- Data flow – batches, parallelism
- Control flow – spool iterator, iterator methods
- Summary & conclusions

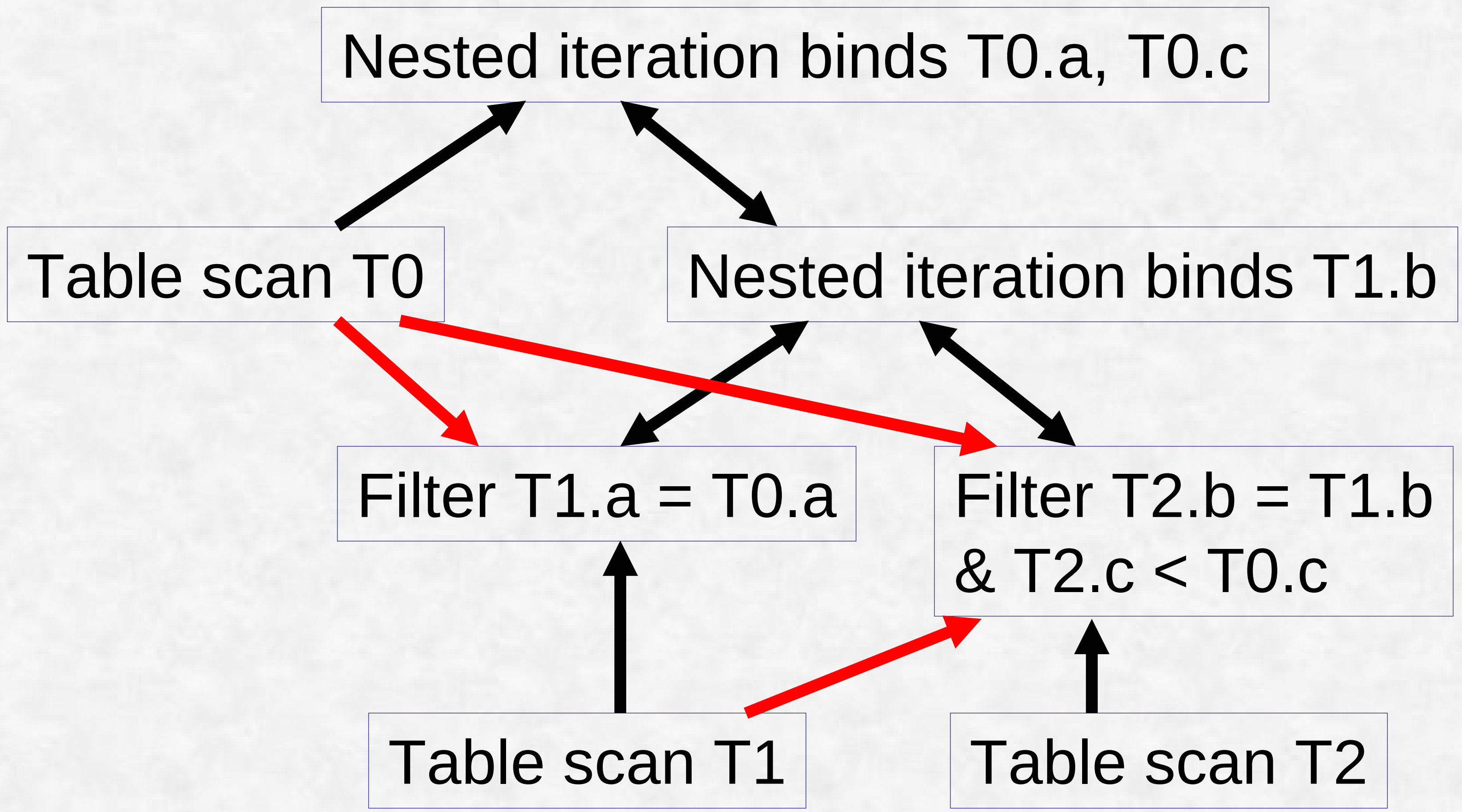
Motivation: scalability

- Disk capacities grow, database sizes grow
- Bandwidths grow more slowly
- Set-based algorithms get slower!
  - E.g., sort, merge join, hash join
- Need algorithms that scale with results size
  - Human attention does not grow
  - Processing capacity grows slowly
- Future requires row-to-row index navigation
  - Nested iteration!

Nested execution plans

- Naïve nested loops, block nested loops
  - Useful only for guaranteed small files
- Fetch full row using record identifier
  - Also search using key of clustered index
- Naïvely execute nested query
  - Multiple levels of nesting
  - Multiple branches at any level
  - Memory-intensive operations: sort, hash, bitmap
- Index navigation plan created by optimizer

Example right-deep nested plan



Asynchronous I/O

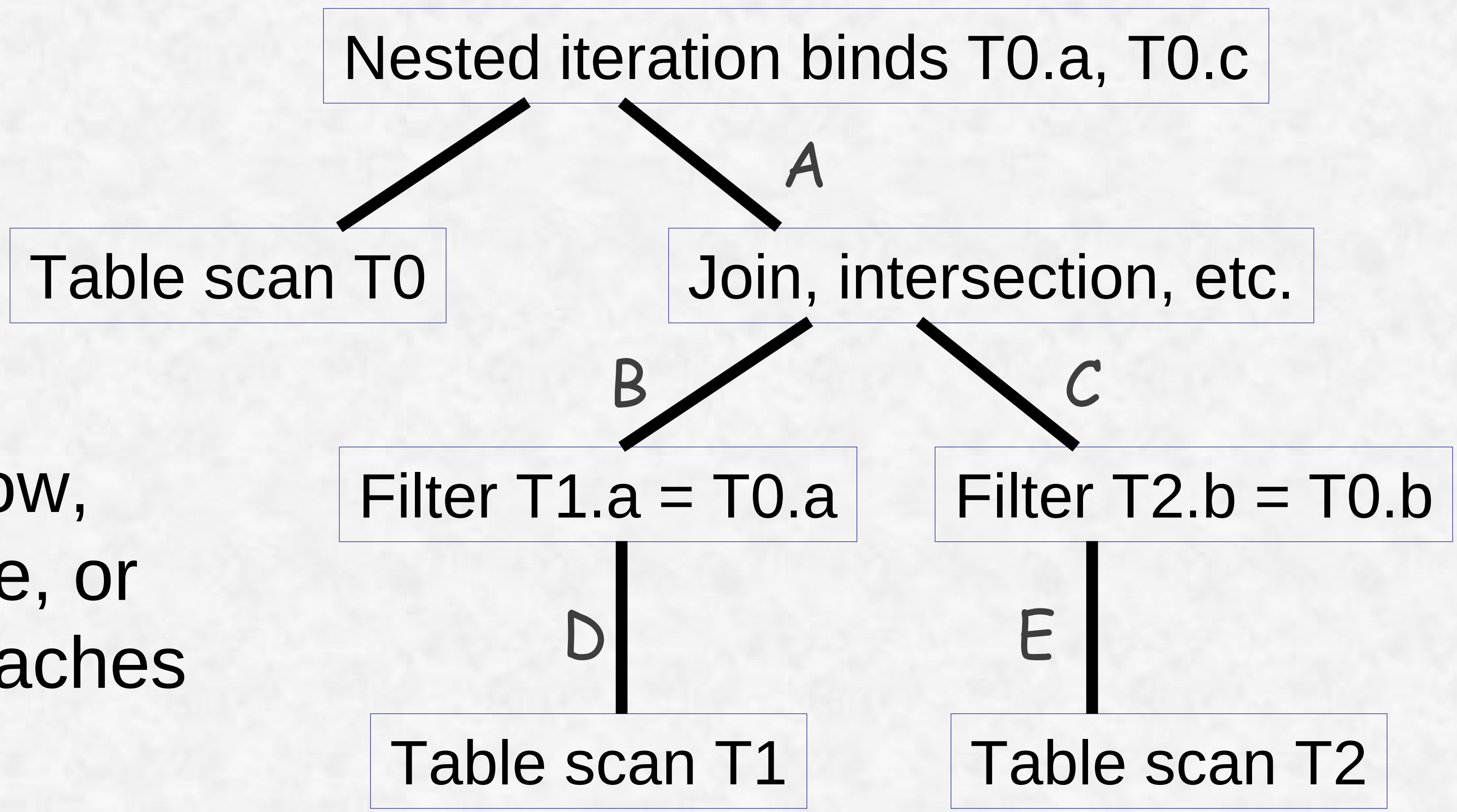
- Read-ahead in sequential scans
- Read-ahead in nested queries?
  - – One thread per disk? – Effect on CPU caches
  - – Fetch twice: separate *hint* from *absolute* request
    - Asynchronous read for first buffer fault or for index leaf
  - – Fetch using a list or a steady-state FIFO queue

Avoiding I/O: caching

- Cache one inner result – sort outer input
  - Opportunistic sort: run generation only
  - “Poor man’s merge join” due to access pattern
- Look-up structure: hash, B-tree, any other
  - Search by parameter value
- Two separate indexes
  - Prior outer values + frequency, LRU info, etc.
  - Prior inner results, if not empty

Cache locations – any or all

- Caches at D and E dominated by caches at B and C
- Cache at A might complement caches at B and C



- Single-row, fixed-size, or infinite caches

Avoiding I/O: merged indexes

- Aka “master-detail clustering”
- Very rigid version:
  - Full rows only – clustered indexes
  - Hashing – no range queries
- Very flexible version:
  - Any index in any B-tree
  - Sort order & search key use domain tags
  - Special tag for table/view & index identifiers
- Merged index for outer & inner values

Most flexible merged indexes

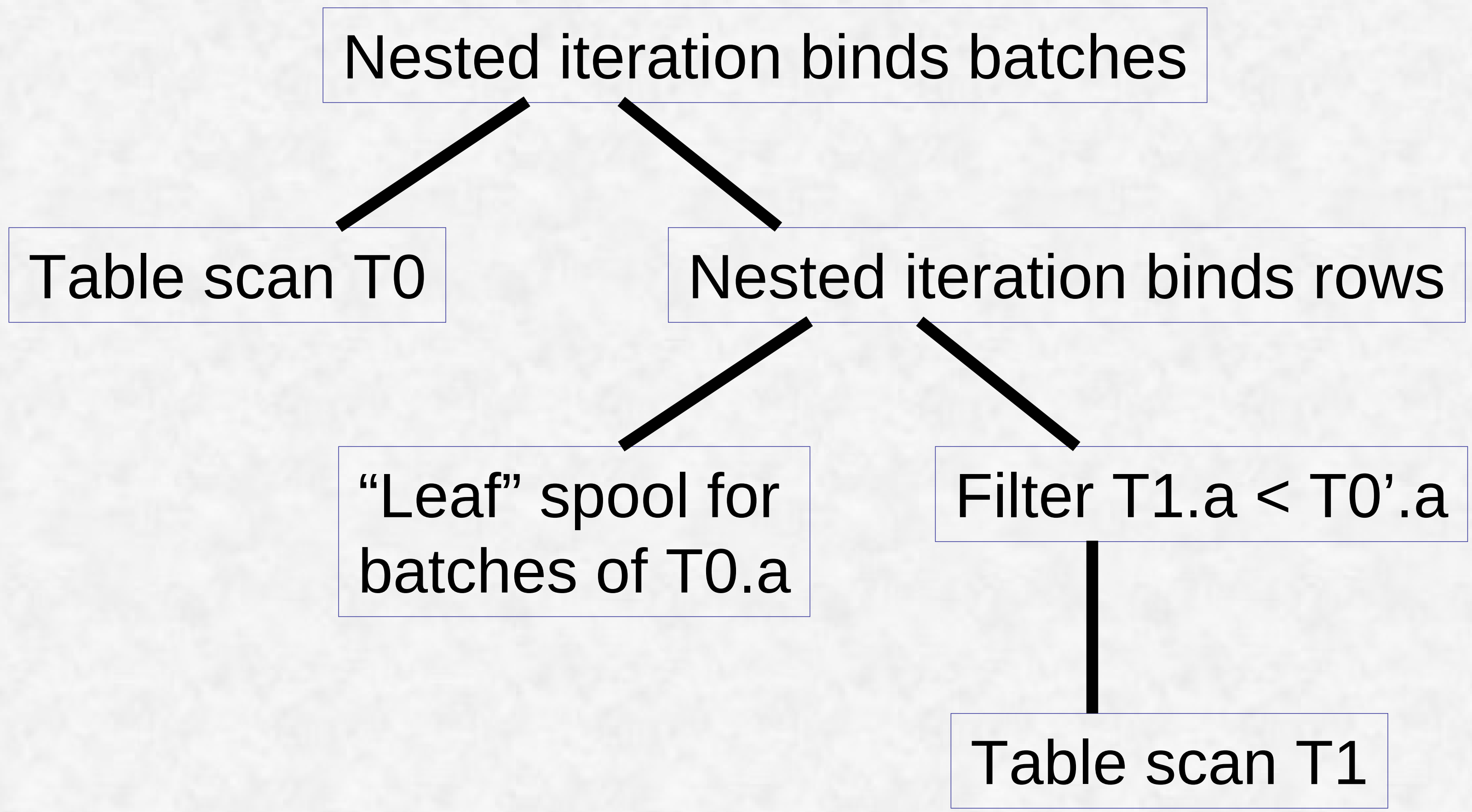
Field value	**Field type**
"Customer #"	Domain identifier
Customer #, e.g., 4711	Actual value
"Order #"	Domain identifier
Order #, e.g., 1234	Actual value
"Table & index identifier"	Fixed domain identifier
Orders table, customer-order index	References to entries in index catalog
Order date, e.g., '2/2/02'	Actual value
…	More actual values

Data flow: batches

- Exploit “economy of scale” in inner executions
  - Shared computations, shared searches
- Retain outer rows to match with inner results
  - Or have inner query regurgitate outer rows
- Accumulate outer rows in inner plan
  - Hash join with single input (+ parameters)
  - Sort & hash distinct with no input (+ parameters)
  - Spool with no input (“leaf”)

Mixed batched & non-batches

- Disassemble batches using another nested iteration



Data flow: parallelism

- Must cross boundaries in batches
  - Thread, process, machine boundaries
  - Batches of parameters, batches of results
- Disassemble on the producer side
  - If & where required

Control flow: spool iterator

- Standard modes of operation:
  - Single input, single output
  - Demand-driven interfaces
  - Filling store eagerly & lazily
- Creating batches in an outer input
  - Batch or “sliding window” mode
    - FIFO or priority queue (i.e., opportunistic sort)
- Managing batches in the inner plan
  - Leaf mode (retain parameter bindings)

Control flow: iterator methods

- Open, next-row, close
- Rewind
- Bind & unbind parameters
  - Boolean result to invalidate cached results
- Pause & resume
  - To manage resources, e.g., memory

Control flow: parallelism

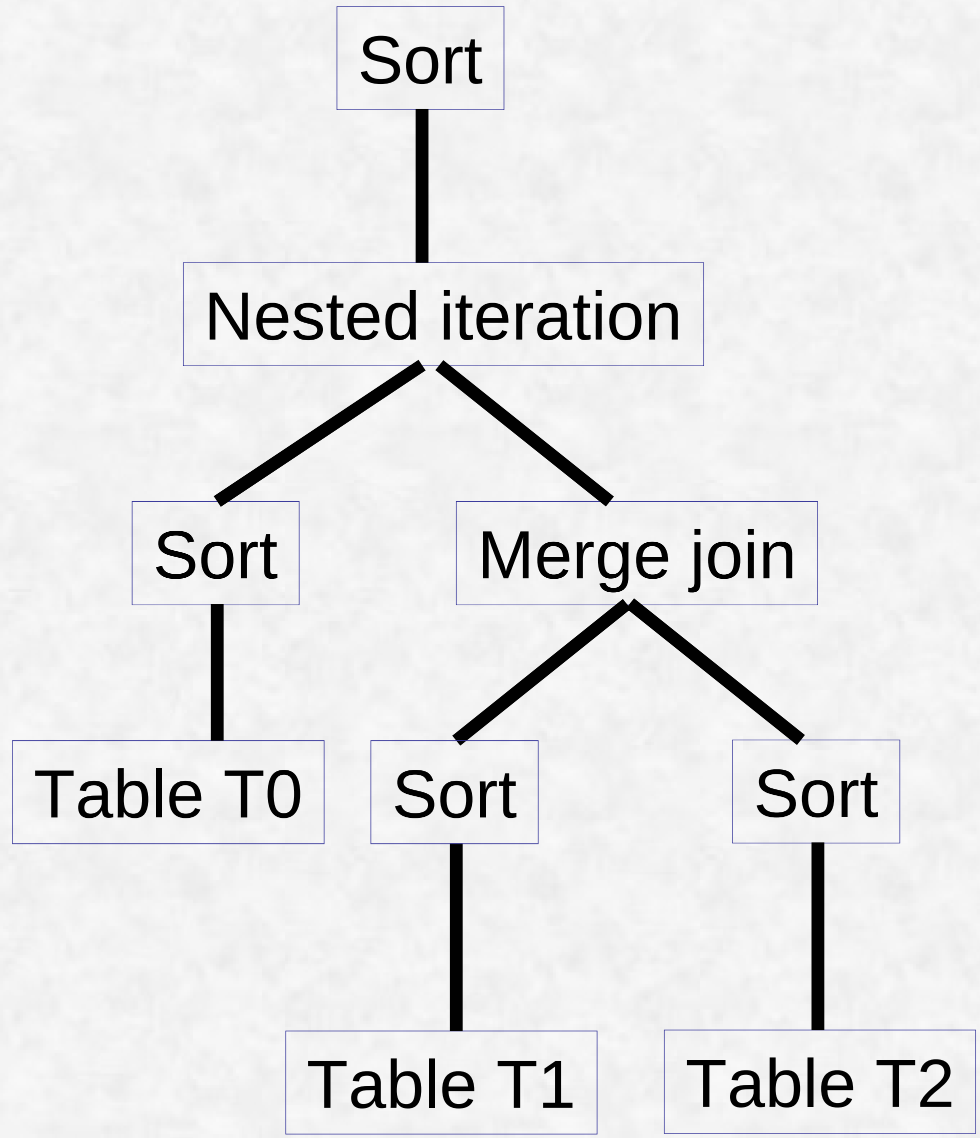
- Invoke inner using batches of parameters
- Share inner threads among all outer threads
  - Bind & unbind for one consumer at a time
  - Pause & result: aggregate over all consumers

Research issues: policies

- Memory management
  - Sort in outer input & inner input & output
  - Multiple levels & branches of nesting
- Batch sizes
  - In single-thread query execution
  - In parallel query execution
- Thread scheduling
  - Assignment of producer threads to consumers
- Cost calculations prior to setting policies?

Memory management

- Nested sorts compete with each other
- Outer sort pauses during inner sort
- Result sort may for a pipeline with the inner
- Inner size might vary for different outer bindings



Summary and conclusions

- Execution of nested plans is not trivial!
  - Attempt to summarize existing technology: caching, batching, iterators, parallelism
  - Provide implementation blueprint for researchers
- Resource policies & mechanisms
  - Memory & threads
  - Multiple levels & branches of nesting
  - Sort, hash, & bitmap operations
  - Hard & practical research!