

Synchronized Sequential Scan

*by
Jeff Davis,
Laika, Inc.*

*thanks to
Heikki Linnakangas and Simon Riggs,
EnterpriseDB*

Introduction

- first patch committed to PostgreSQL
- functional prototype, proof-of-concept patch completed in 2005
- didn't have time to follow up, then went back to the patch for inclusion during 8.3 dev cycle
- patch was committed this year

Sequential Scans

- Sequential scan is one of many plans for finding tuples
- reads entire relation file, a.k.a. “the heap”
- start at block 0, read to end, returning any matching (and visible) tuples

Useful

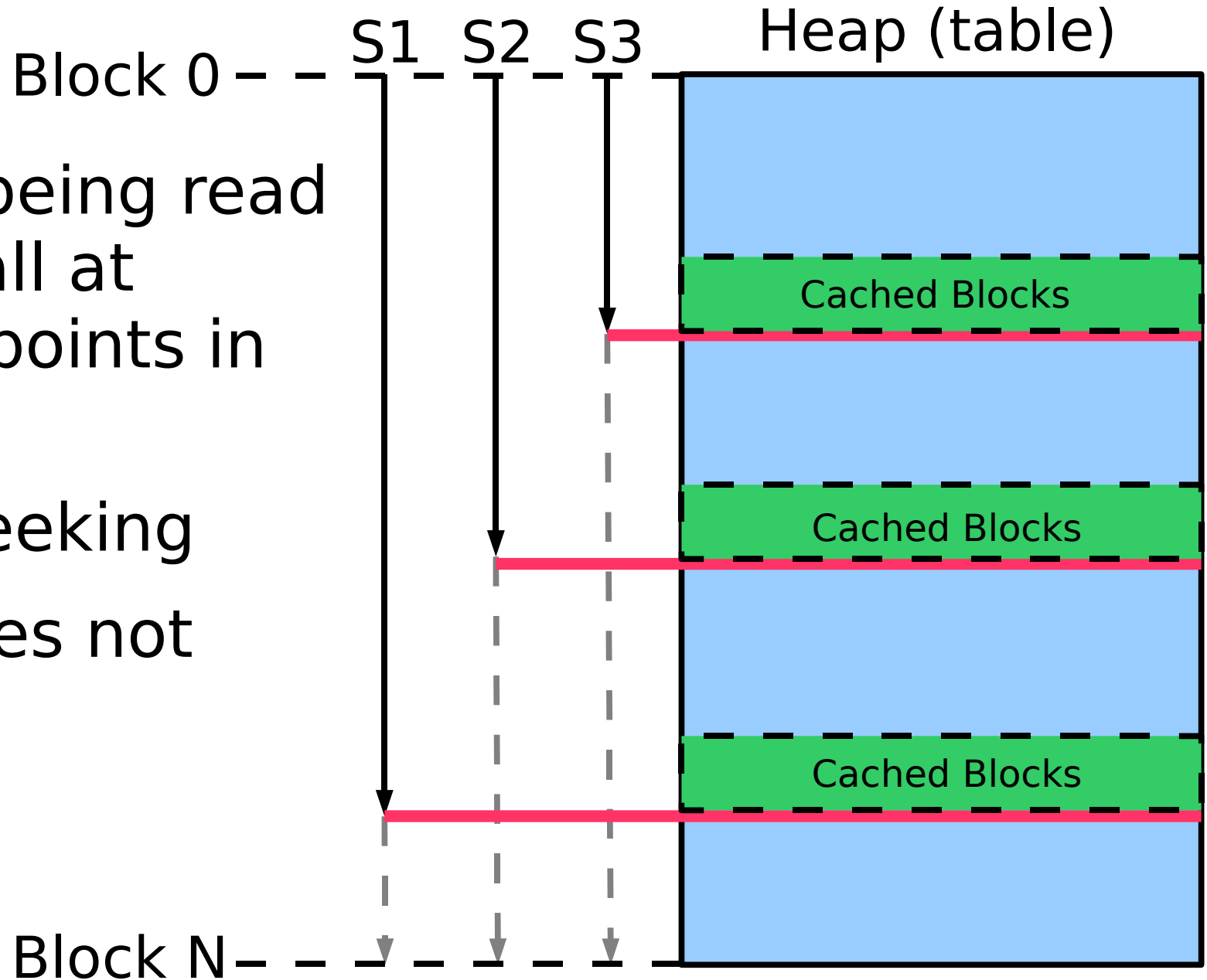
- sometimes sequential scans are necessary, such as when no indexes are available
- most efficient when a large fraction of the tuples are returned
- sometimes more efficient than an index even for a **sort**

Two Big Problems

- When multiple I/O-bound sequential scans are happening on the same table
- And the table is large compared to the cache
- First Problem: inefficient use of cache
- Second Problem: disk seeking

Current Behavior

- 3 blocks being read at once, all at different points in the file
- Causes seeking
- Cache does not overlap



Current Behavior

- Very bad performance when the “Two Big Problems” conditions are met.
- Cache is useless. By the time S2 reads a block that has already been read by S1, that block has already been evicted due to memory pressure
- Seeking. Amplifies problem, and causes **unpredictable** spikes in response time on other queries for extended periods of time.

Observation

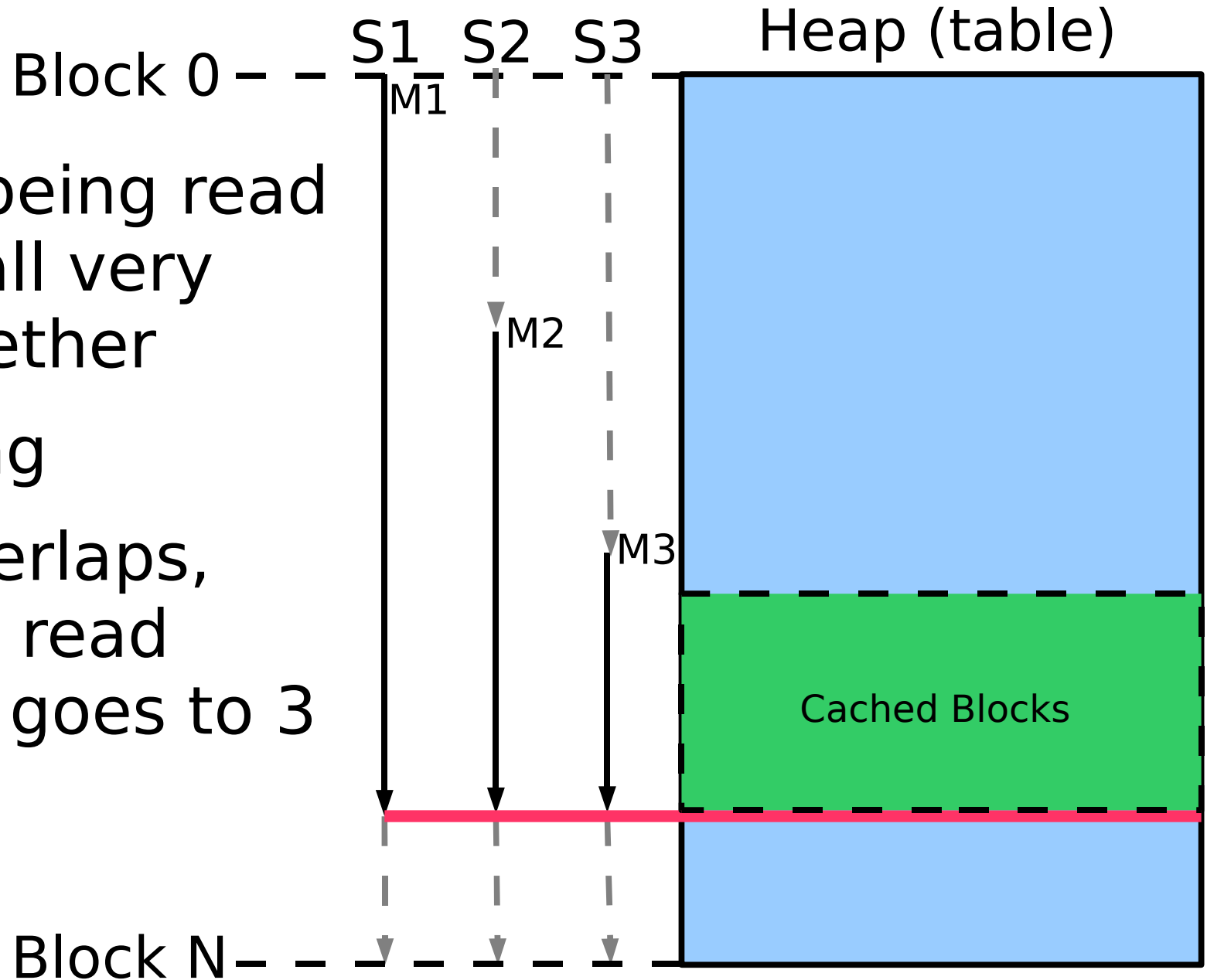
- The current behavior is based on reading from block 0...N.
- Starting at block 0 is arbitrary. The blocks can be read in any order
- Where the scan starts will affect performance a great deal

Solution

- Let's be smarter about where we choose to start.
- For simplicity, I generalized $0...N$ to be: $M...N; 0...M-1$. In effect, “wrap around”.
- The old behavior is equivalent to $M=0$.
- Now we choose a better M : we choose M to be a block closely behind the current block being read by another concurrent scan.

Sync Scan Behavior

- 3 blocks being read at once, all very close together
- No seeking
- Cache overlaps, one block read from disk goes to 3 scans



Solved!

- Now, add 10 more scans. Each will run as though it was the only scan on that table!
- No seeking.
- A block is read once, and provided to all concurrent scans on that table.

Nondeterministic Results

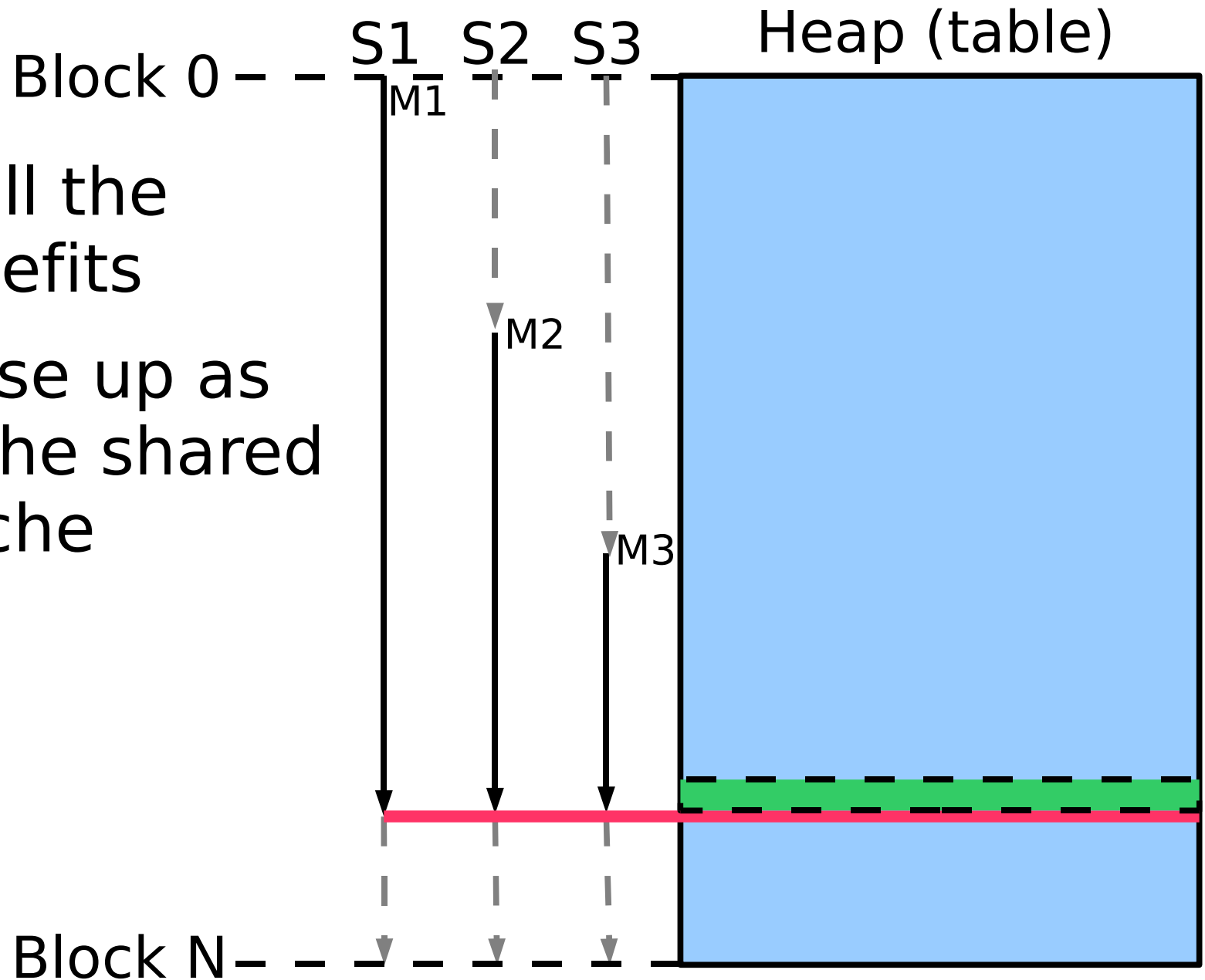
- Because we start scanning at a different point in the heap depending on what time the scan starts, two scans may return the same tuples in different order.
- Always use ORDER BY if you care about the order. You can't count on the tuples to be returned in heap order.

Scan Recycle Buffers

- Simon Riggs wrote a patch to reduce the buffer cache pollution of a sequential scan.
- With his patch, every sequential scan uses a small ring cache rather than the general shared buffers.
- Had to make sure our patches were *effective together*.

Sync Scan + Recycle Buffers

- Still has all the great benefits
- Doesn't use up as much of the shared buffer cache



I/O Schedulers

- Linux-Anticipatory/NOOP/deadline: Very good. Ideal, actually. My tests showed that scans had no measurable interference with each other as more scans were added to the same table.
- Linux-CFQ: Very bad in my tests. If you are using this at all, I recommend you analyze other schedulers yourself.
- ZFS: good, but more testing is needed.

Conclusion

- Use case is somewhat narrow. DBAs generally avoid running many sequential scans on the same large table at the same time.
- If you need sync scans, you REALLY need them. And you don't want to be hit with unpredictable performance problems.
- DBAs no longer have to work around the problem.