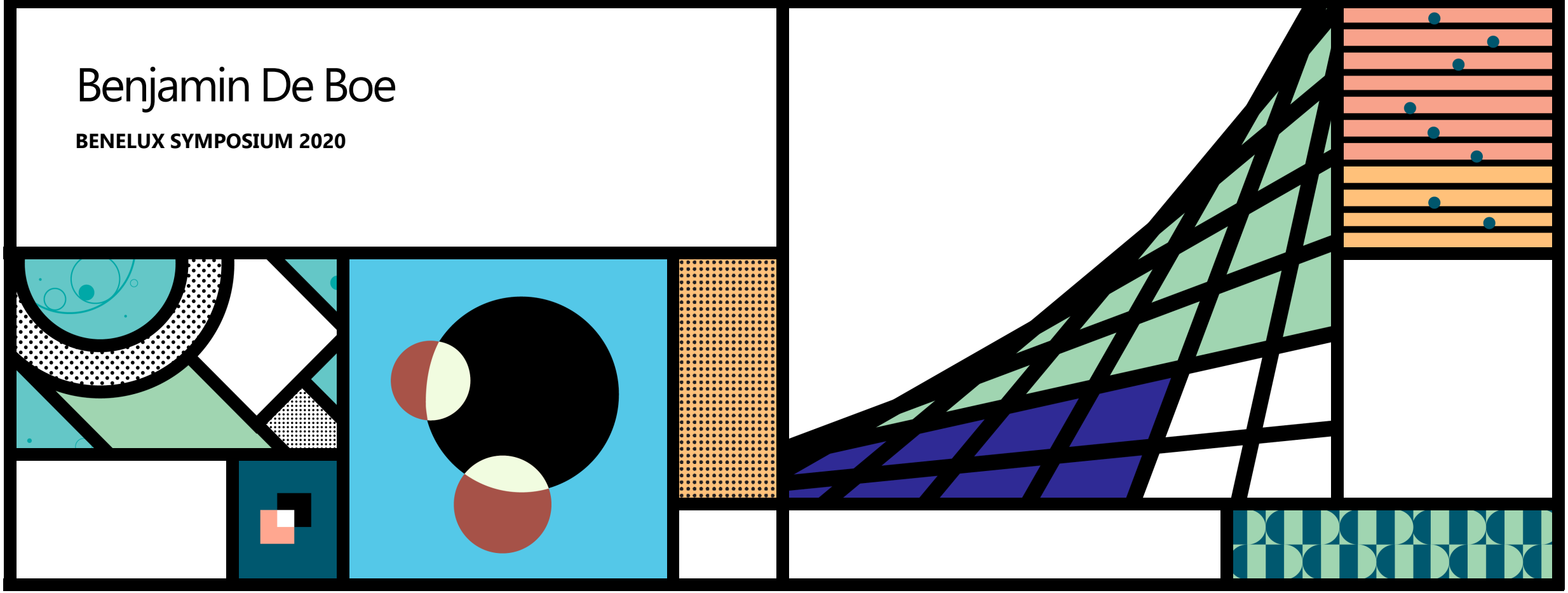
SQL Performance: Need for Speed

InterSystems®
Health | Business | Government

Benjamin De Boe

BENELUX SYMPOSIUM 2020



SQL Performance: Need for Speed

InterSystems
Health | Business | Government

SQL Performance: Need for Speed

InterSystems
Health | Business | Government

SQL Performance: Need for Speed

InterSystems®
Health | Business | Government

We made SQL 2x faster Here's how

SQL Performance Need for Speed

InterSystems®
Health | Business | Government

Outline

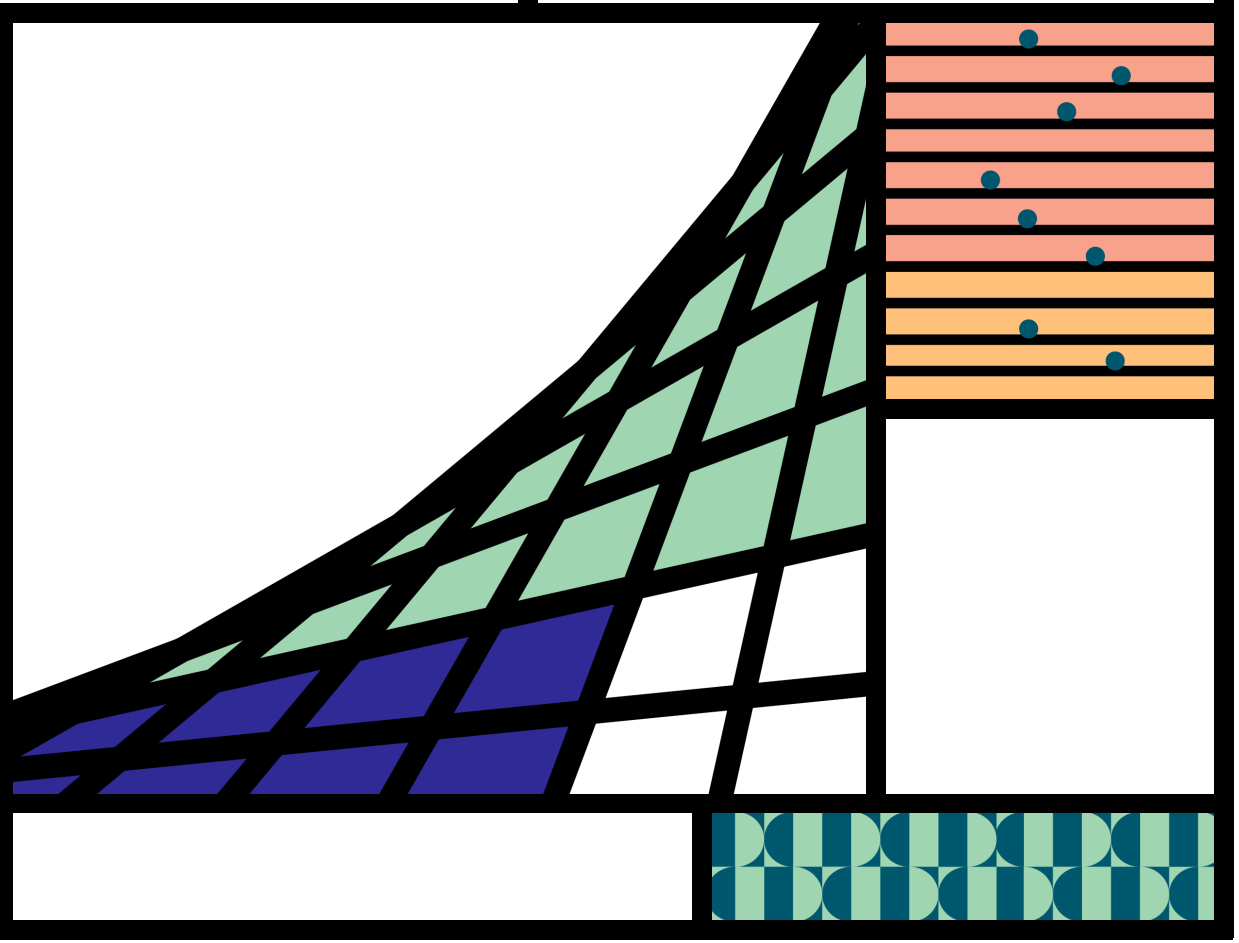
Architected for Speed

A-Team Sport

Accelerating through 2019

- Data Engine
- Distributed SQL

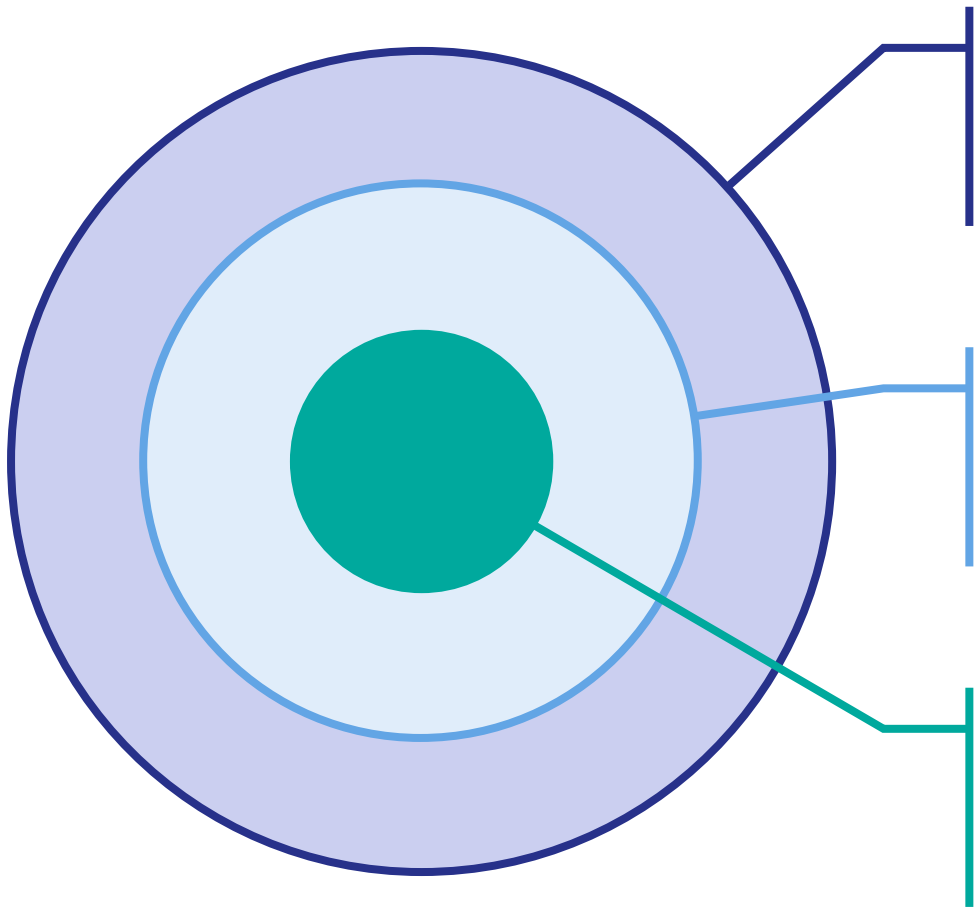
The Finishing Line



Architected for Speed

SQL Performance – Need for Speed

Platform | Building on the Right Foundations



Add Data Models:

a fast, distributed query engine for SQL, Objects, Documents, …

Add Scale:

a distributed, coherent cache

Data Engine:

a fast, ordered store for data and code

A-Team Sport

SQL Performance – Need for Speed

A-Team Sport | The Team

A-Team Sport | Performance Engineering

Regression tests

Continuous, semi-automated benchmarking of basic metrics

- Yields timely alerts for unexpected performance regressions

Deep dive

Regular, more pervasive stress testing with comprehensive test suites

- Deeper analysis of cumulative code enhancements or evaluating new hardware
- Includes industry benchmarks such as YCSB, TPC-H and TPC-DS

A-Team Sport | Extra Muros

Partner Testing

Leverage customer benchmarks for measuring specific benefits

- e.g. ahead of major upgrade
- Nurtures long-term partnerships

Competitive Benchmarks

Bake-off vs competing or incumbent product in select sales opportunities

- Has yielded some of the more interesting innovations

A-Team Sport | Extra Muros

Sample Partner Benchmark

Highly complex all-SQL benchmark on anonymized customer data touching a variety of advanced features of the SQL engine

- Statistics & outliers
- Parallelization opportunities

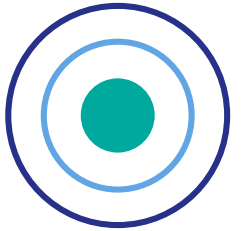
Tested on every major release since Caché 2017.2

 |

Accelerating through 2019 – Data Engine

SQL Performance – Need for Speed

Data Engine | Block Search Optimization



Context

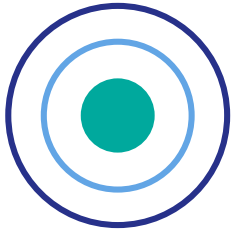
Like every database, InterSystems IRIS spends the majority of CPU time scanning blocks to locate records requested by users

- InterSystems' Globals and subscripts add an important dimension vs classic databases
- On InterSystems IRIS, this data access function is called gsearch4k()
- Block structure is typically highly optimized for sequential lookup

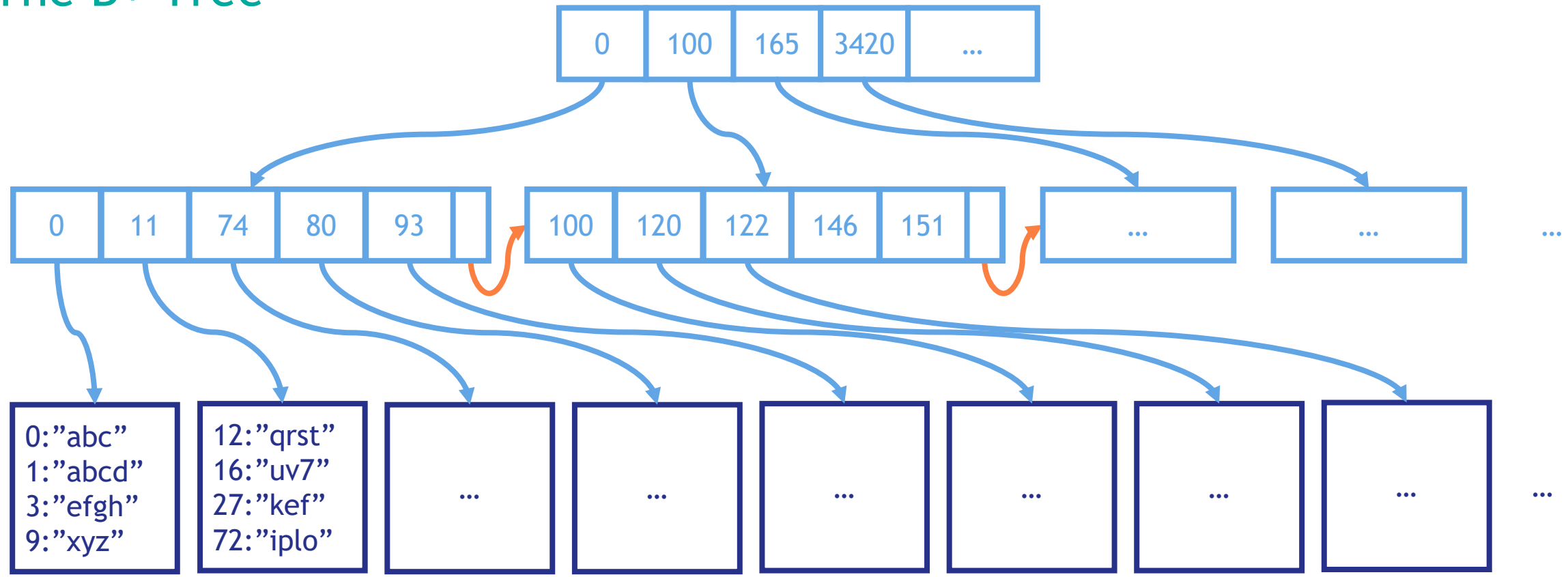
However, not all lookups are sequential…

Crosshair on: gsearch4k() for nonsequential lookups

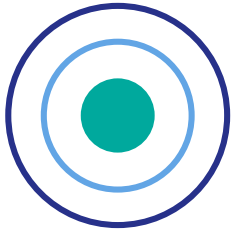
Data Engine | Talking Trees



The B+ Tree



Data Engine | Block Structure



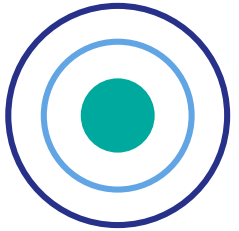
Simplified IRIS block structure

```
^myGlo("ABCD") = $lb(123,"some data")
^myGlo("ABKEF") = $lb(456, "significantly more data")
^myGlo("XYZ") = $lb(789, "much more data so it's really hard to keep track")
^myGlo("XYZZ") = $lb(0, "short")
```

```
:"ABCD":18:[123,"some data"]
```

 | |

Data Engine | Block Structure



Advantages

- Efficient storage for **arbitrary-length** subscripts and data values
- Highly **efficient forward search** within a block

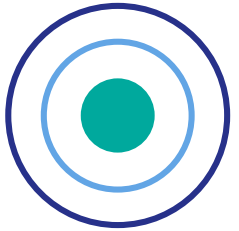
However, this is less efficient for **backward or random search** within a single block

- In SQL: WHERE and ORDER BY x DESC clauses

```
:"ABCD":18:[123,"some data"];2:"KEF":33:[456,"
significantly more data"];0:"XYZ":58:[789,"muc
h more data so it's really hard to keep track
of where we were"];3:"Z":11:[0,"short"]
```

 |

Data Engine | Node Tables



Simplified Node Table Contents

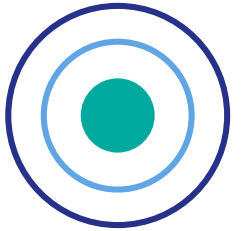
Build an internal index of all nodes in the block and their starting offset. For each node, track **prior node with lower CCC**

- Allows quickly rebuilding full subscript
- Enables naïve binary search

Also track earliest **next node with lower CCC**

- This subscript will collate after the search string if current CCC is fully matched.
- Enables smart binary search through establishing an upper limit

Data Engine | Node Tables



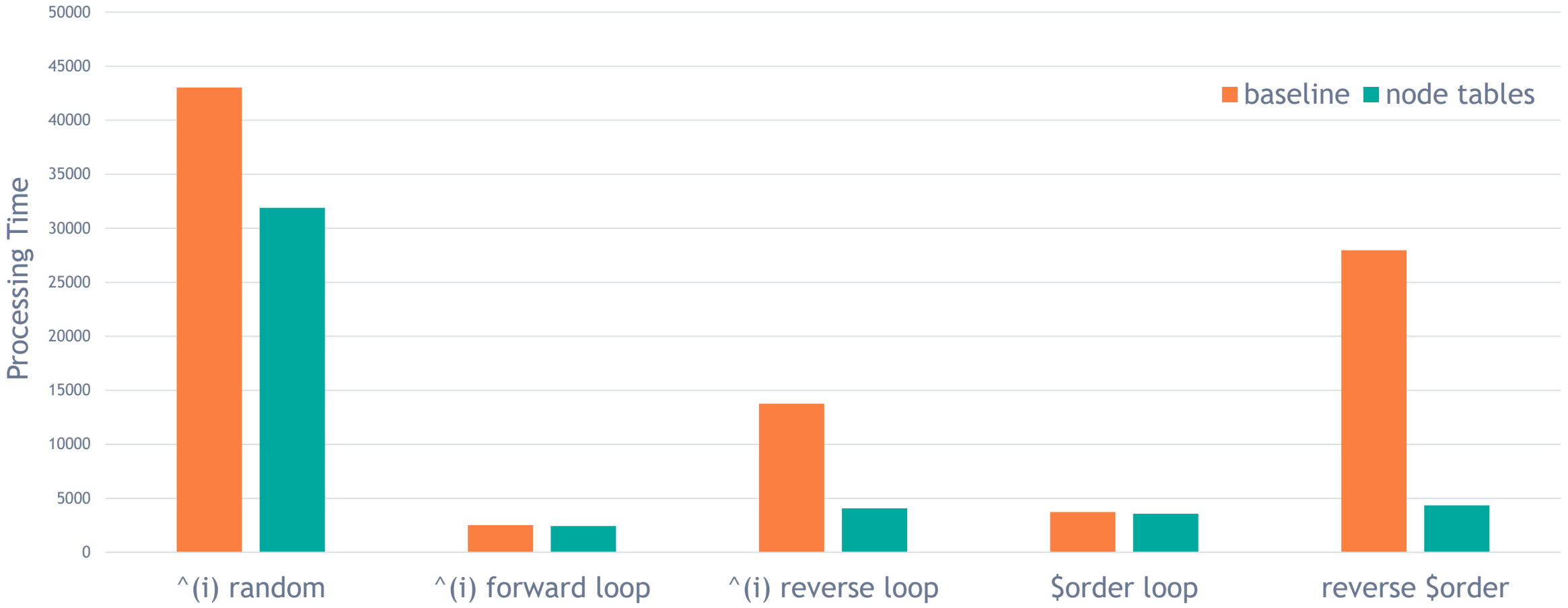
Node Table Lifecycle

Node tables are built and maintained in-memory for pointer blocks and infrequently changing data blocks that have enough free space at the end

- Consume less than 1% of overall memory
- Automatically discarded when a block is updated

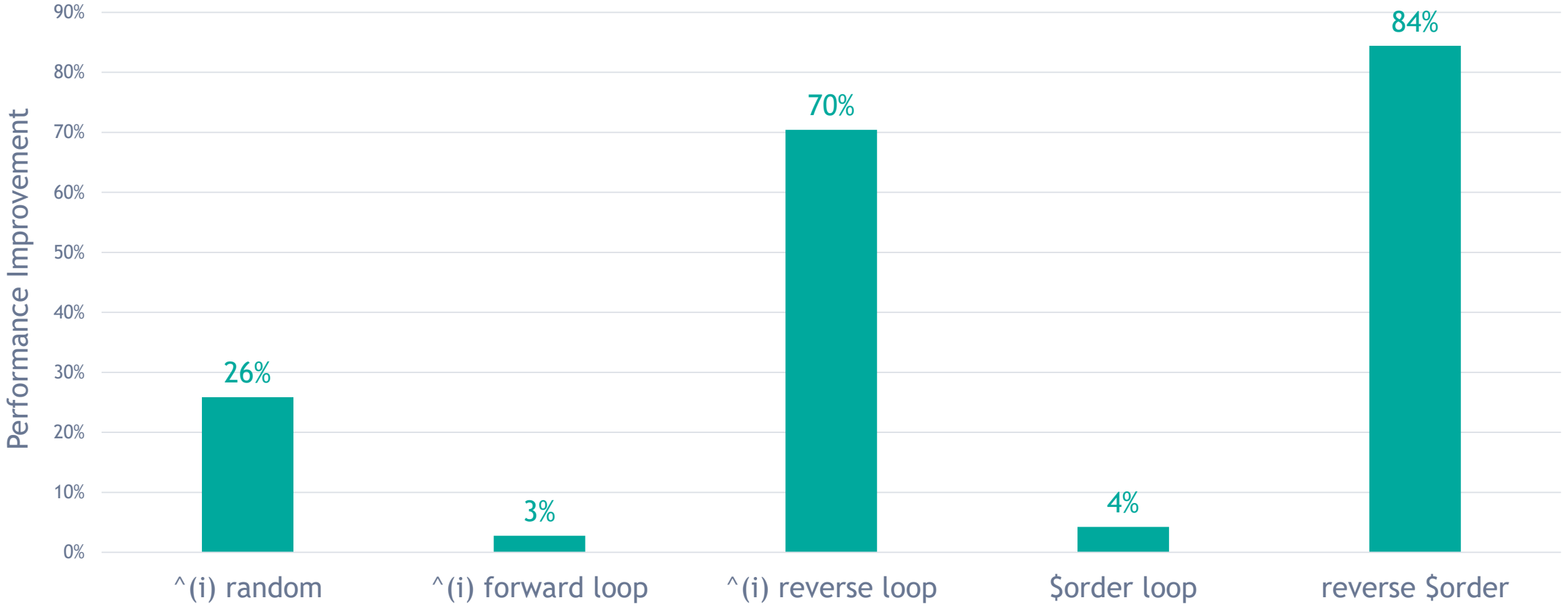
Fully automated mechanism, 100% transparent to application code

Data Engine | Benefits – Global Access



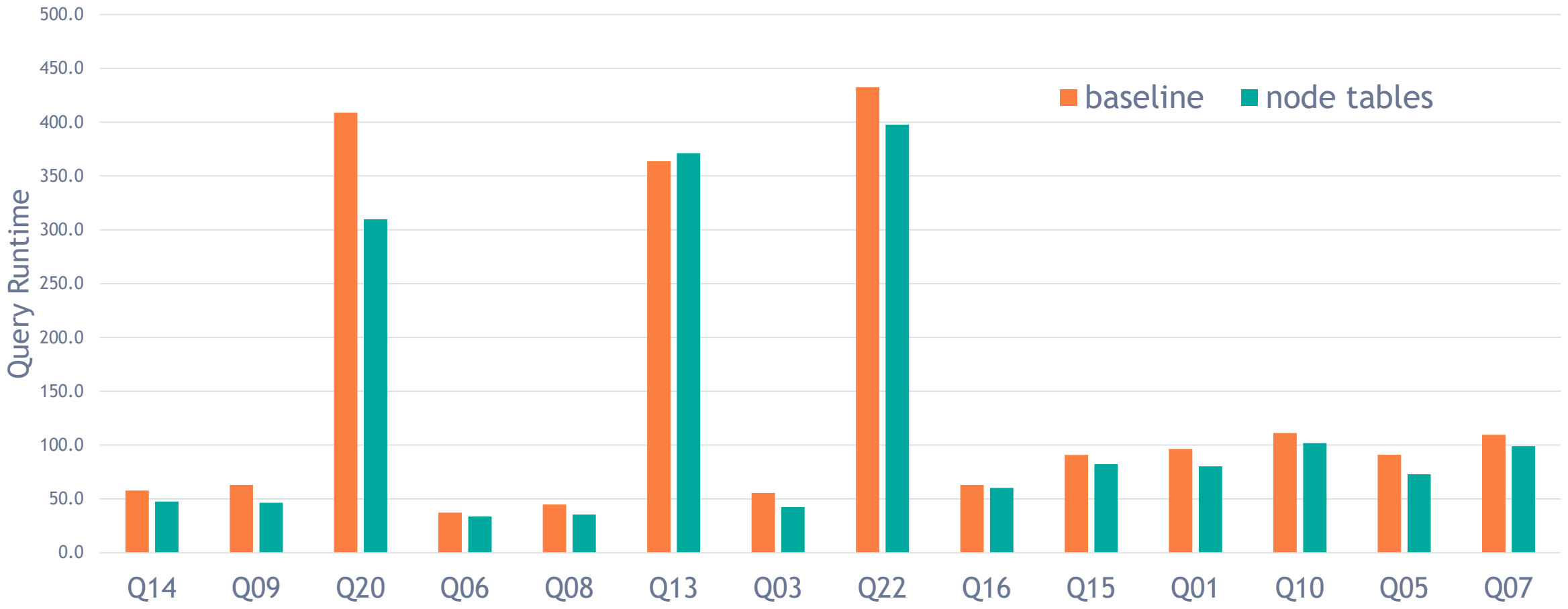
 | |

Data Engine | Benefits – Global Access

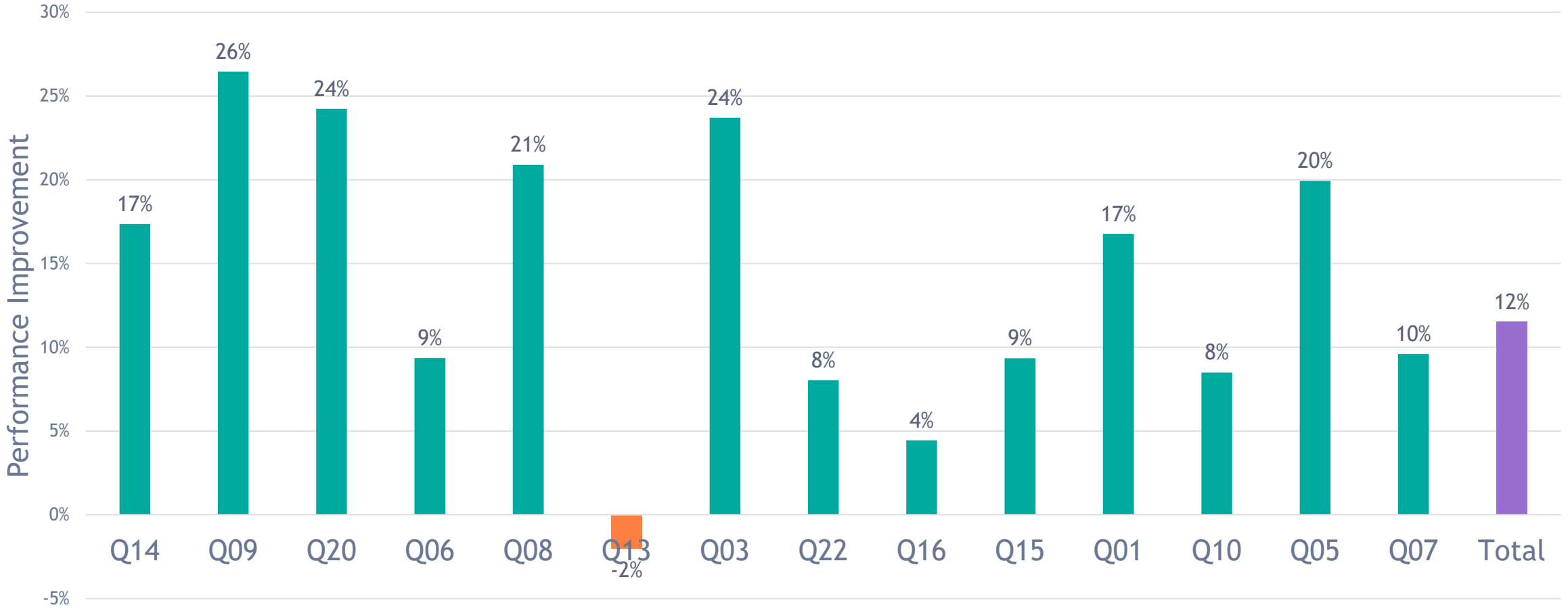


 | |

Data Engine | Benefits - SQL

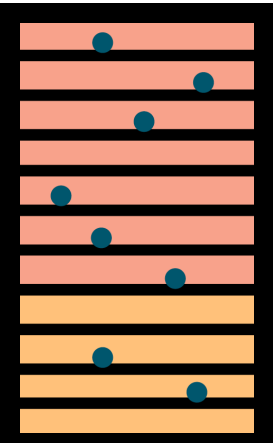


Data Engine | Benefits - SQL



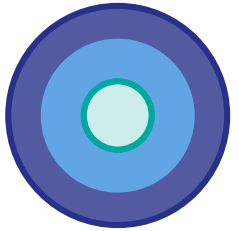
 | | Confidential

Accelerating through 2019 – Distributed SQL



SQL Performance – Need for Speed

Distributed SQL | CPU Usage



Context

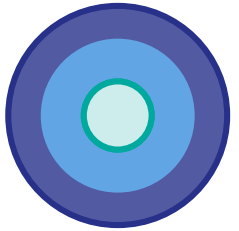
Running a highly demanding benchmark in a competitive sales opportunity indicated high CPU usage

- Using perf tool for process analysis

Crosshair on: memory allocation

CPU Used	Module	Function
9.11%	irisdb	[.] gsearch4k
2.34%	libc-2.17.so	[.] __memmove_ssse3
2.05%	irisdb	[.] xmbudretnoscope
1.98%	irisdb	[.] xmallocbud
1.74%	irisdb	[.] gblkrd
1.72%	libc-2.17.so	[.] __memcpy_ssse3
1.72%	irisdb	[.] Ldlist_elem
1.68%	irisdb	[.] gloformat
1.56%	irisdb	[.] Cmlsetlstbld

Distributed SQL | CPU Usage



Low-level CPU load analysis and a debug kernel build's bookkeeping pointed at this line getting executed 2.5M times during the sample:

```
set row = "" if $data(^globalName(ID),row)
```

What's happening here?

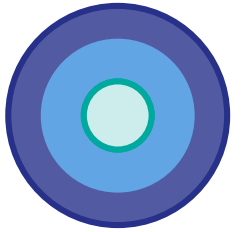
1a. Free memory for `row`

1b. Allocate smallest memory size for `row`

2. Allocate 4k memory for `row`

 | |

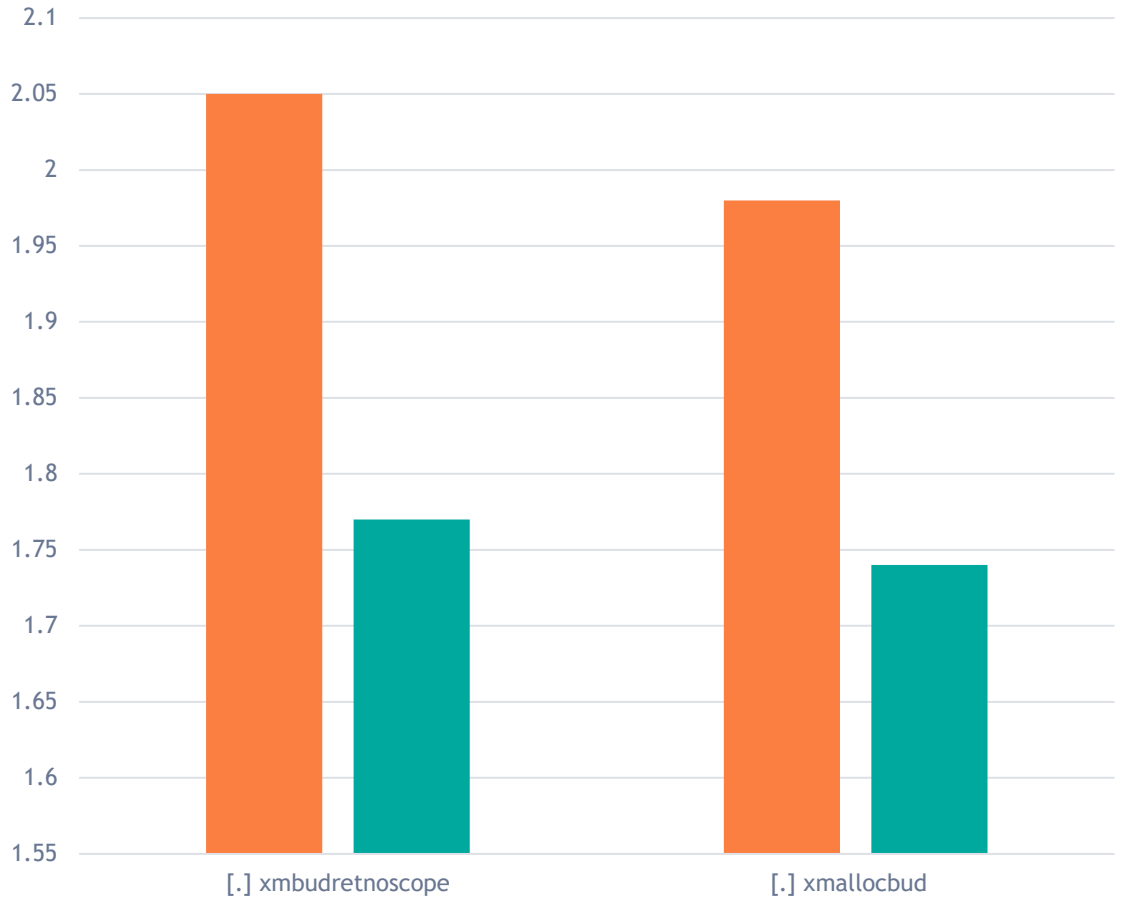
Distributed SQL | Code Generation



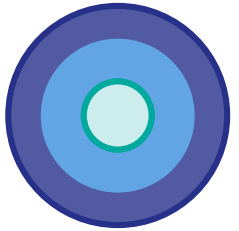
Solution

Code generation tweaks managed to avoid this and yielded a generous 2.5% decrease in CPU consumption

Updated SQL code generation is in 2019.3



Distributed SQL | CPU Overhead



Context

Still in the same opportunity, looking for further opportunities to decrease CPU load and leave room for spikes in user activity

Noted sharp increase in CPU consumption when scaling up user count

- Very high number of processes
- Lots of time spent in context switches

Crosshair on: decreasing process overhead / count

 | |

Distributed SQL | CPU Overhead

Decreasing process overhead

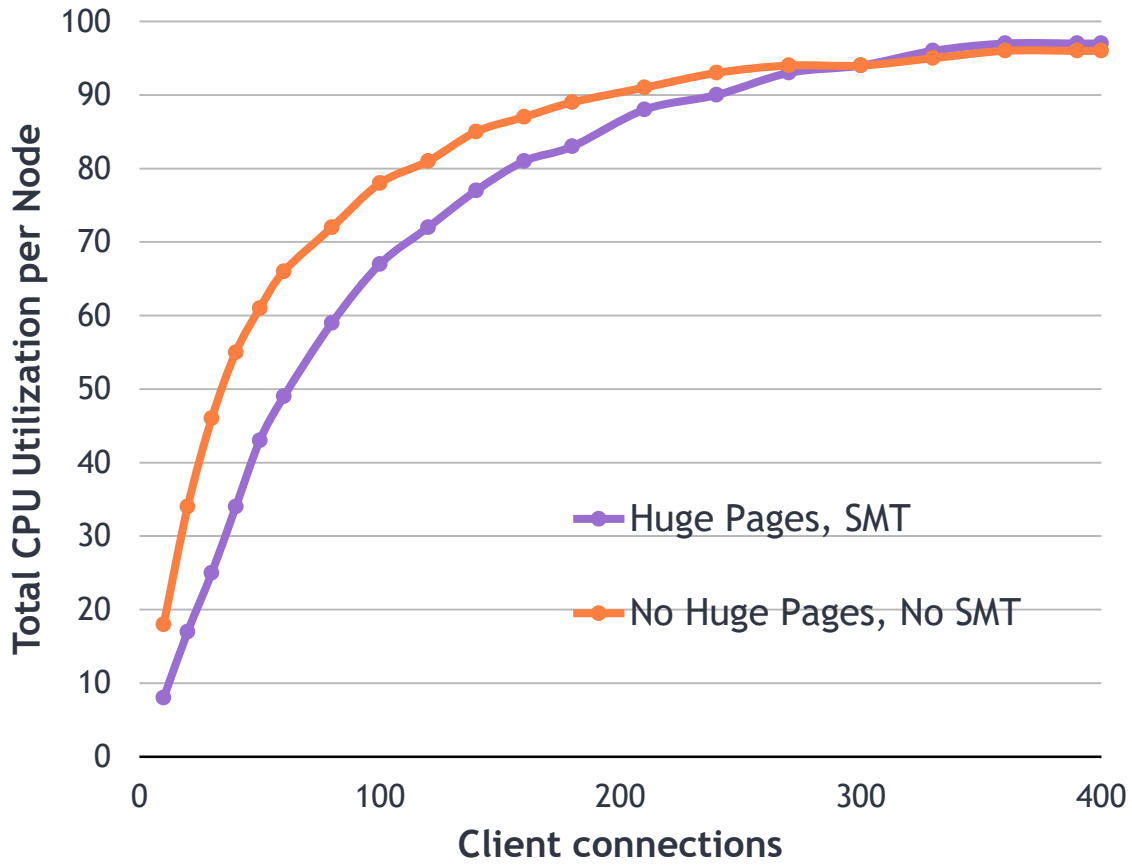
Standard, documented tuning already decreased process overhead somewhat:

Huge Pages

- Decreases CPU overhead related to memory management

Simultaneous Multi-Threading

- Lowers the frequency of process-process context switching and cache invalidation



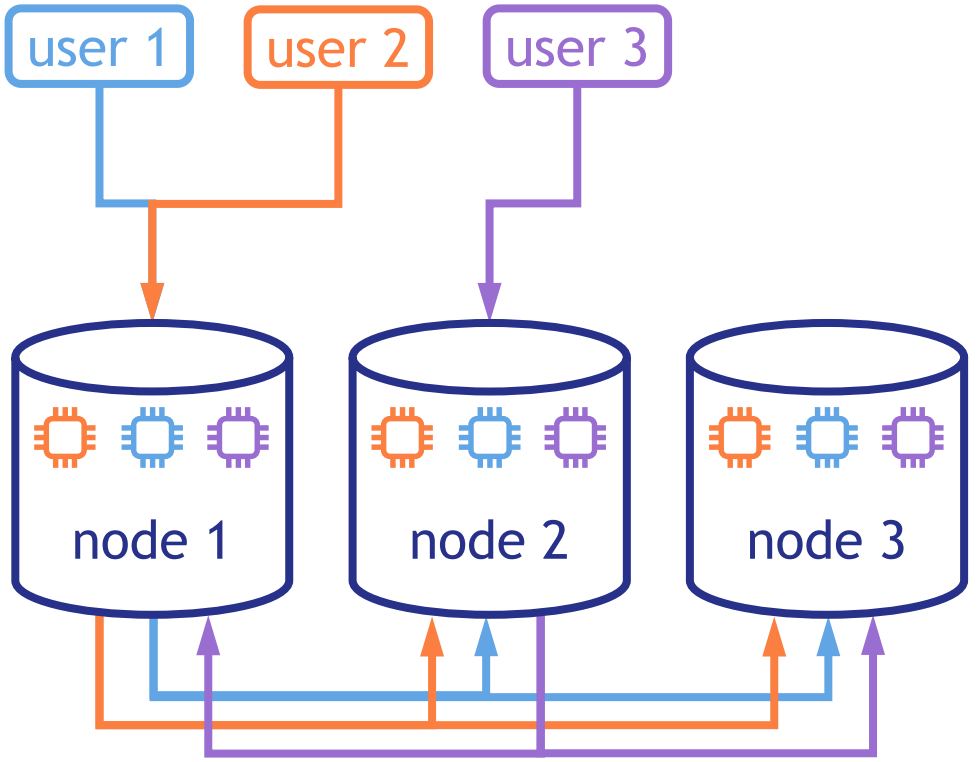
 | |

Distributed SQL | Inter-Shard Communication

Decreasing process count

When a user issues a query to a sharded cluster node, that node opens a connection to all other nodes to obtain shard-local results

- Leveraged existing xDBC mechanisms, with one process per connected user

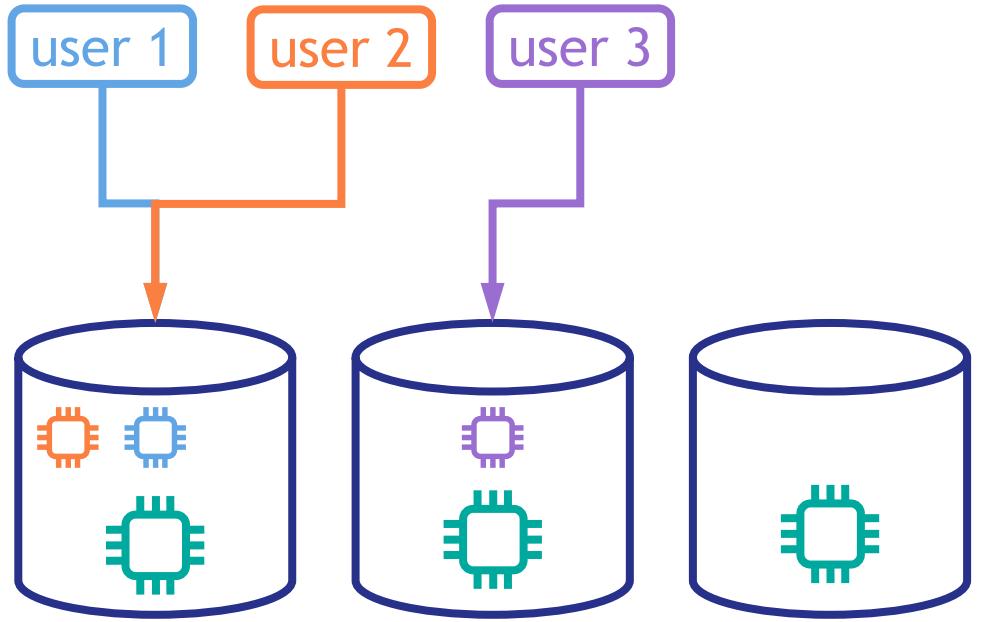


Distributed SQL | Shard Queue Manager

The **Shard Queue Manager** replaces the initial xDBC-based mechanism for intra-shard connections with a single work queue per node

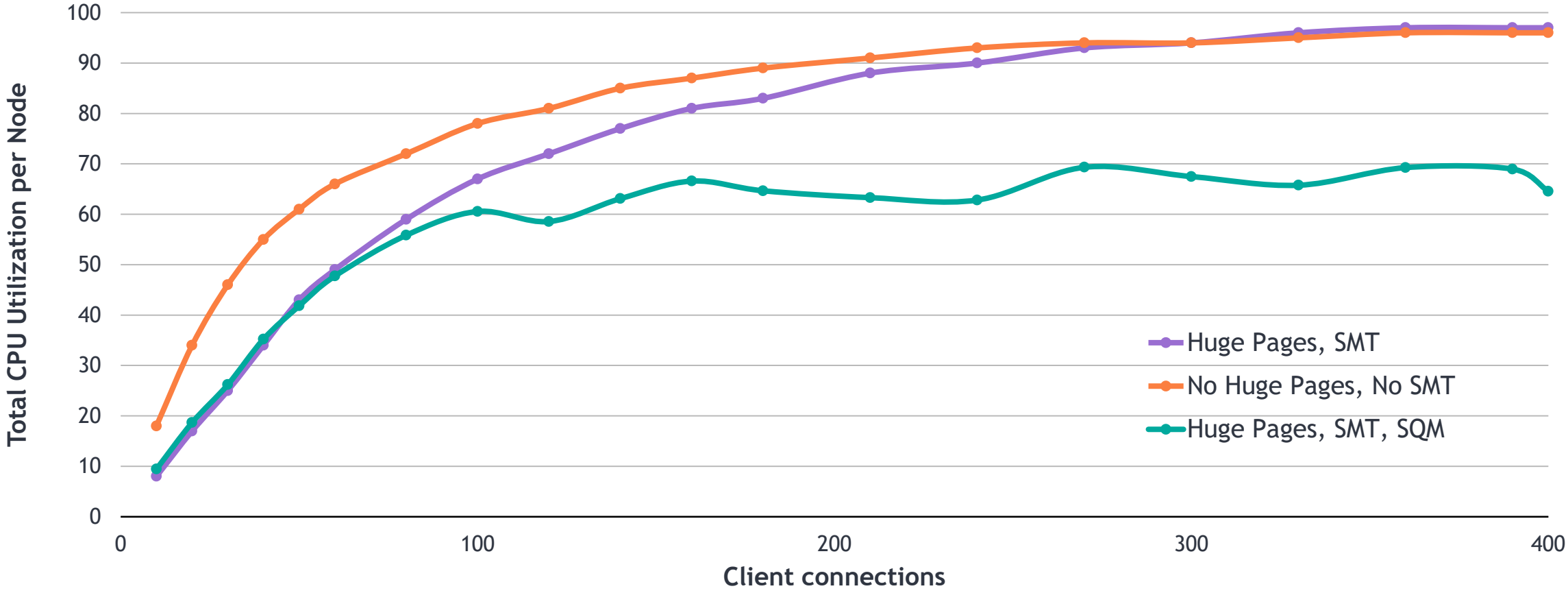
- Auto-scaling number of worker processes
- Builds on proven Work Queue Manager infrastructure

Included with 2019.4



 | |

Distributed SQL | Shard Queue Manager



 | |

The Finishing Line

SQL Performance – Need for Speed

Free Lunch

SQL Performance | Conclusion

These are all 100% transparent benefits

Upgrading to the latest release is always highly recommended

- Unfreeze query plans & recompile classes for maximum benefits
- Your mileage may vary but the engine *is* faster

Thinking of in-place conversion…?

This is not just about SQL Performance

SQL Performance | Outlook

We're never done!

Many exciting things ahead, both dev-driven and sales-driven

- Better use of statistics & outlier info in cost function
- Improved scalability for ECP on high-end platforms
- Further improvements to parallel & sharded query plans
- Columnar storage format for relational data
- SQL window functions

Thank You

InterSystems®
Health | Business | Government