Minimizing CPU Utilization Requirements to Monitor an ATLAS Data Transfer System

Georgios Leventis - CERN

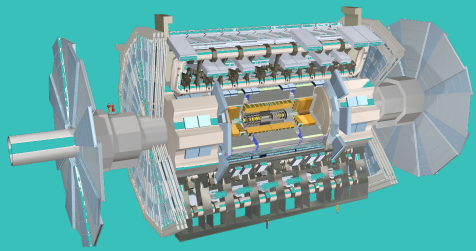
Hello!

I AM GEORGIOS LEVENTIS

Technical Student at CERN

geoleven@outlook.com

ATLAS



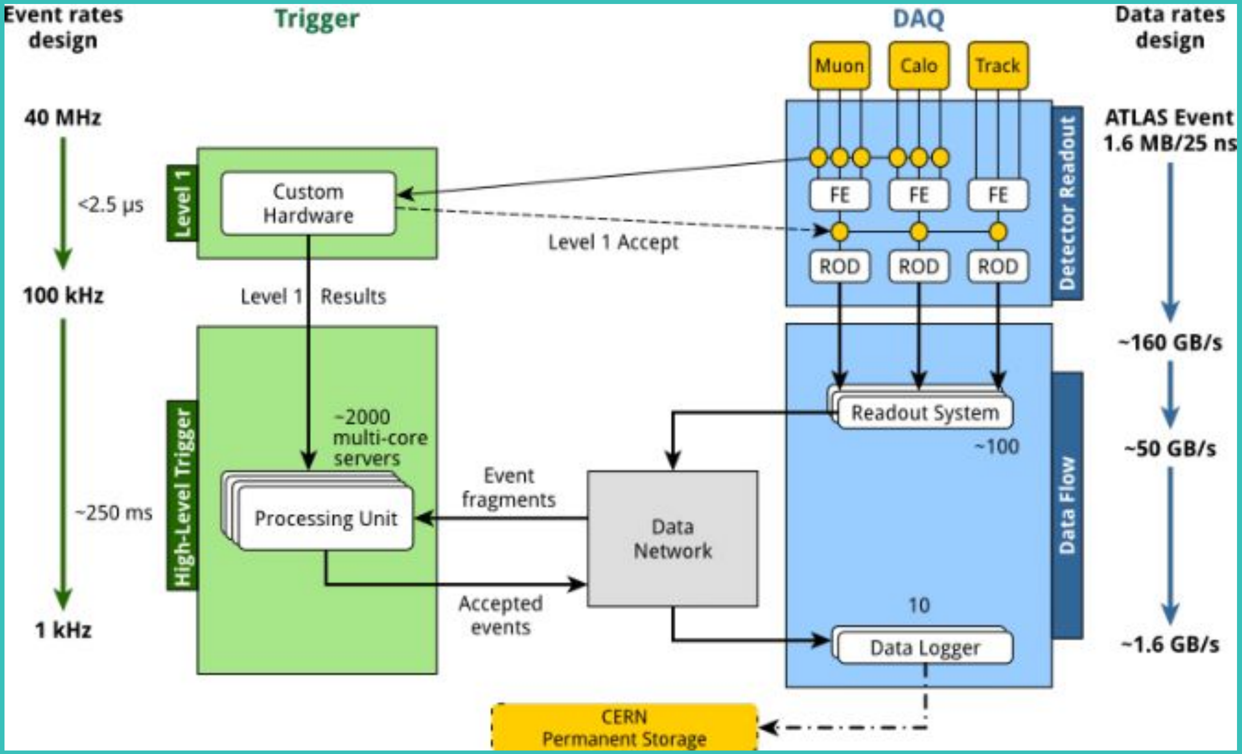
ATLAS Experiment at CERN

LHC general purpose detector

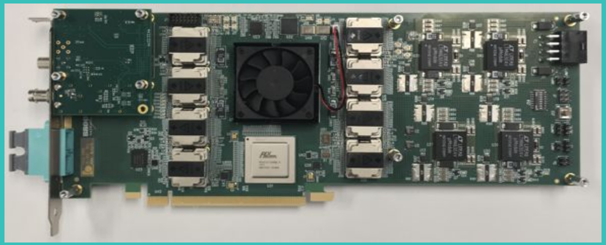
- Constant stream of data
- Data reduced by 2 level of triggers
- Currently the readout system uses custom electronics

Current TDAQ

The ATLAS Trigger and
Data Acquisition System
during Run 2



FELIX

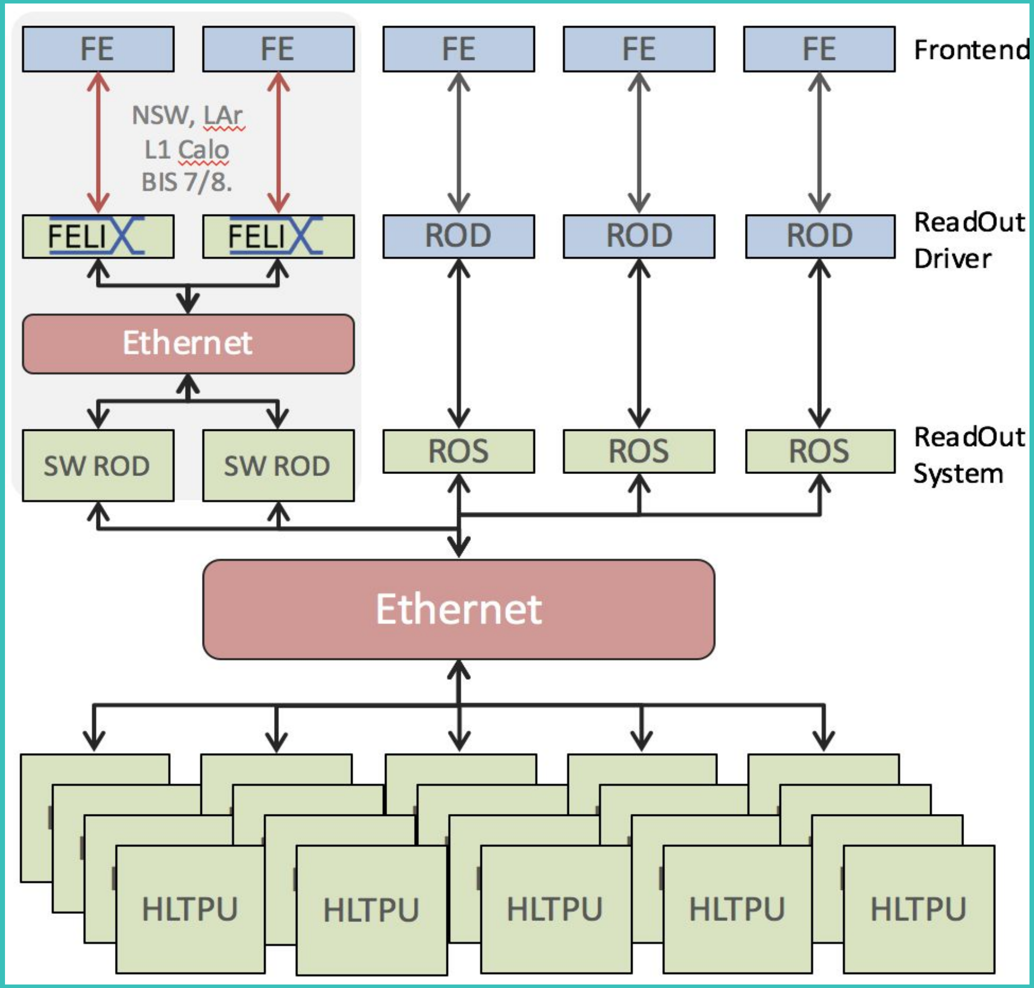


FELIX is the new readout system for the ATLAS experiment

FELIX routes data coming from the ATLAS detector front-end electronics to the DAQ network

- Servers with custom FPGA-based PCIe cards
- Critical infrastructure:
  - Control of ATLAS detector
  - Data flow
- Every server - system is driven by the **FelixCore** software application

TDAQ after FELIX integration as foreseen for Run 3



FelixCore

FelixCore is the application that routes the data from the FELIX cards to commodity network.

- Needs to be able to handle up to ~1.5GiB/s or ~5GiB/s of constant data streams (mode depended)
- Needs to be able to monitor the FELIX machine it runs on as these machines are single point of failure components

1

The problem

Efficiently monitoring ***FelixCore***

- A single FELIX computer receives up to 40MHz of data fragments
- Routing within FelixCore is a CPU-intensive task
- For efficient routing **parallel threads** are used
- Statistics from these threads have to be combined to be meaningful

What are we monitoring?

- Data counter:
  - Data packet rates
  - Throughput
  - Error rates
- FELIX variables:
  - Global buffer memory
  - Thread buffer memory
  - Queue sizes
  - Interrupts
  - Polls
- Etc

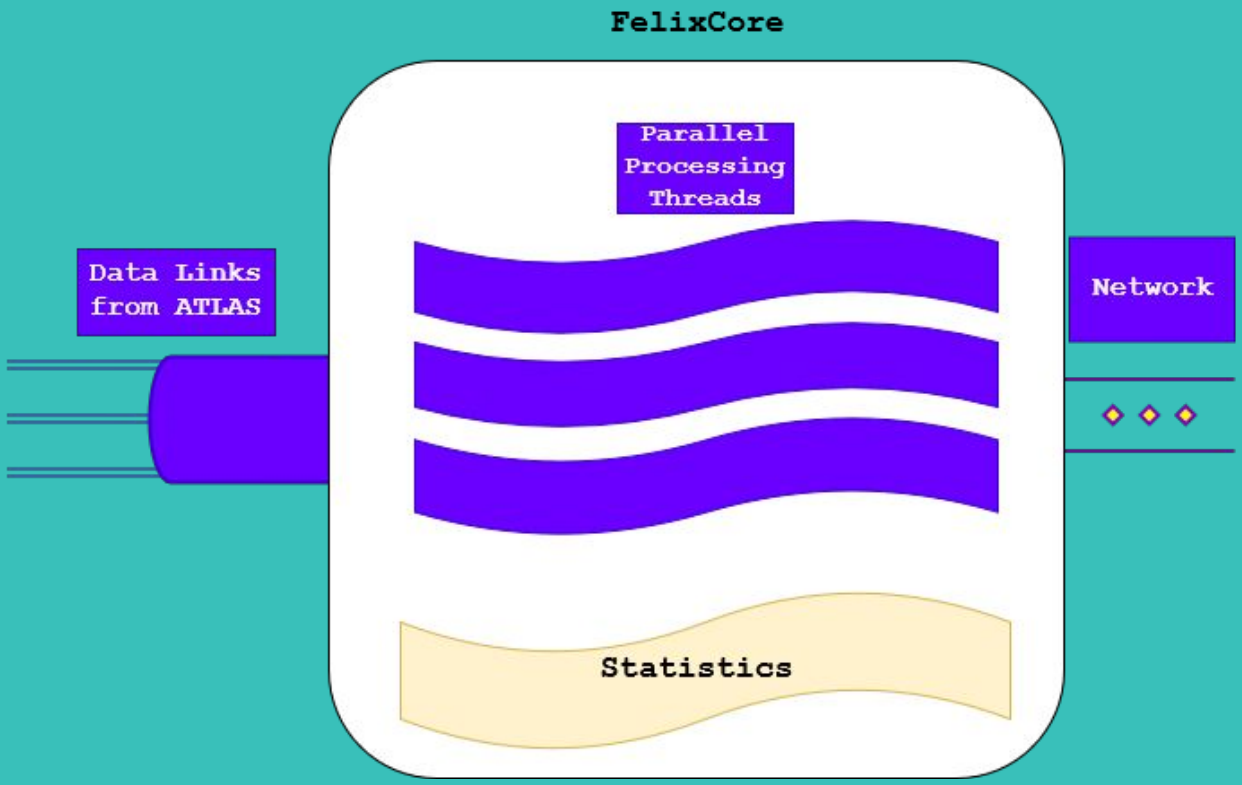
Statistics?

For the statistics to be useful, individual thread's statistics must be:

- Gathered
- Processed / Combined

The data routing threads have to communicate with the statistics thread.

Statistics


FelixCore
Parallel Processing Threads
Data Links from ATLAS
Network
Statistics

Info | Configuration | Statistics | Bus

General

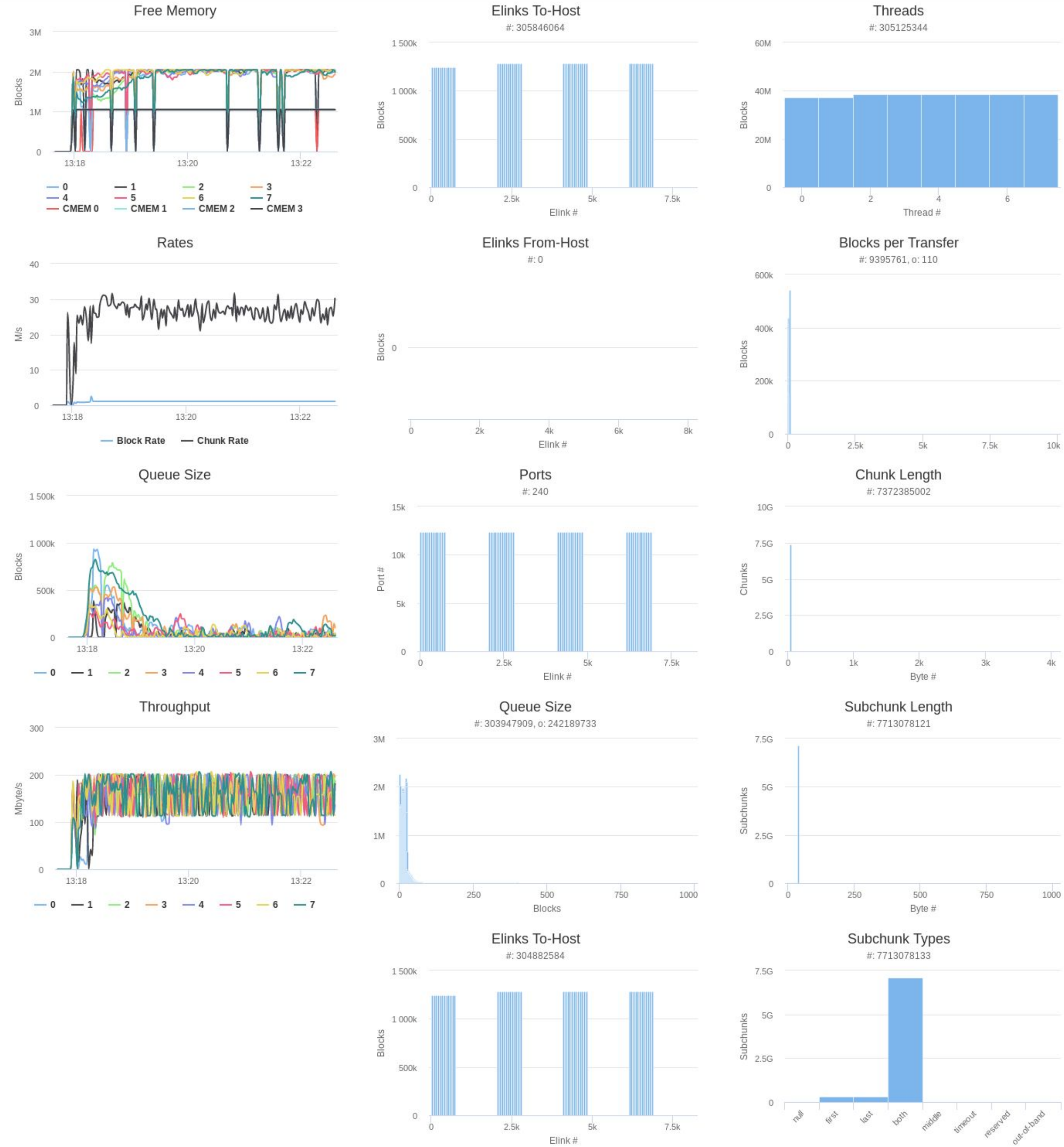
timestamp	1543321356	
blocks	300	MBlocks
chunks	290	MChunks
shortchunks	7.1	GChunks
block_rate	1.1	MBlocks / s
chunk_rate	31	MChunks / s
throughput	1500	Mbyte / s
max_queue_size	110	kBlocks
avg_queue_size	44	kBlocks

Card

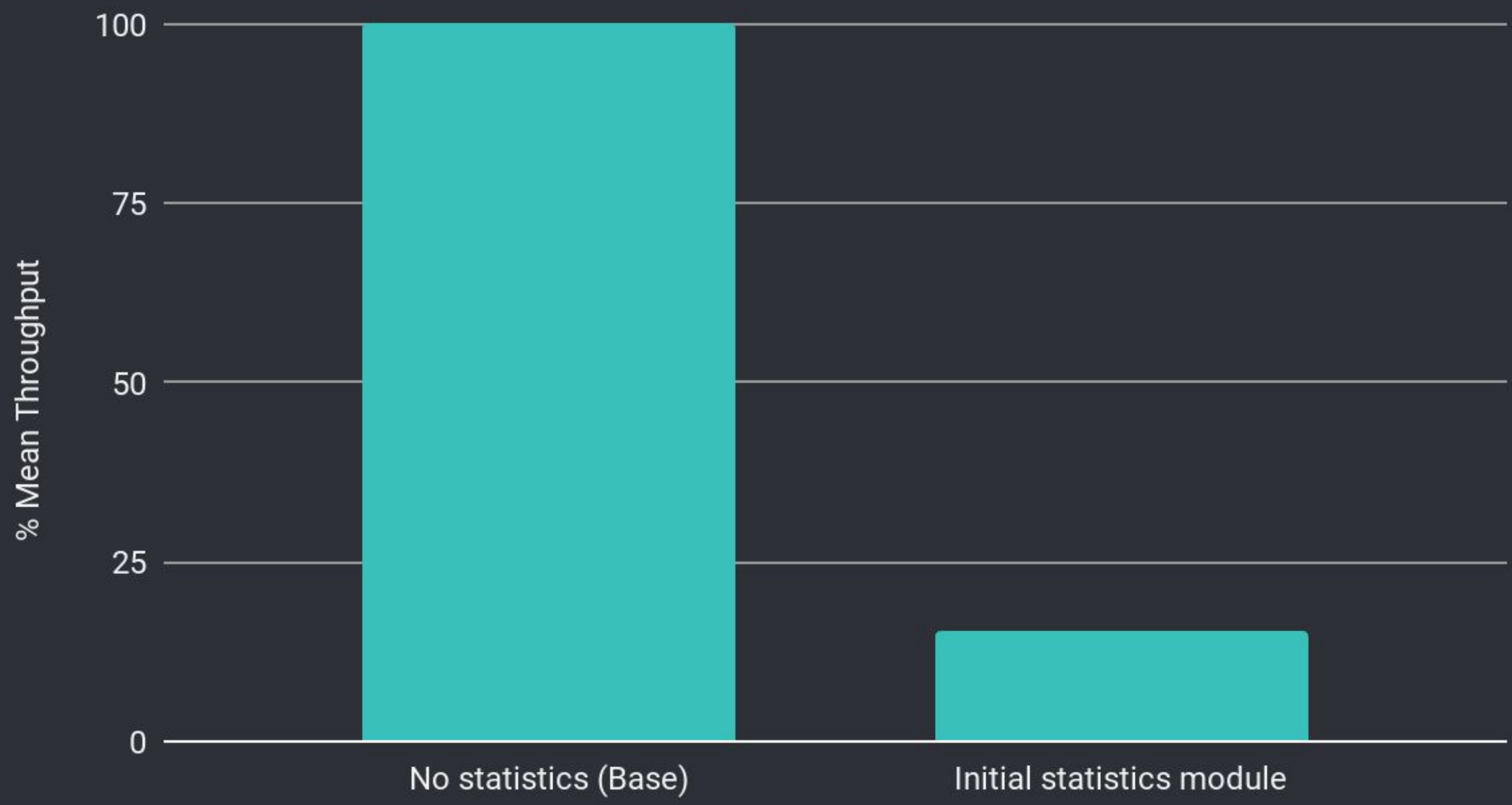
blocks_free 0	1	MBlocks
blocks_free 1	1	MBlocks
blocks_free 2	1	MBlocks
blocks_free 3	1	MBlocks

Errors

malformedSubchunks	0	Chunks
malformedChunks	0	Chunks
malformedBlocks	0	Blocks
size_error_chunks	0	Chunks
error_chunks	0	Chunks
truncated_chunks	0	Chunks



Achievable throughput relative to no statistics baseline

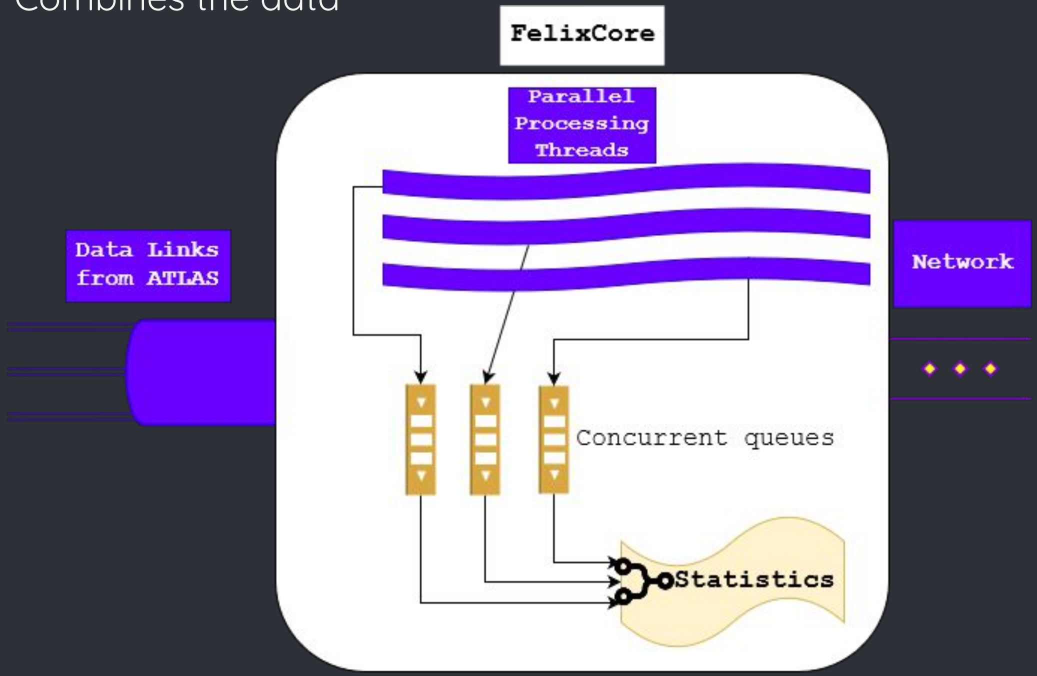


THE IMPACT

Extracting statistics has ~**85%** performance hit!

The ***initial*** implementation

- Data routing threads:
  - Push their individual monitoring data to concurrent queues
  - Pushing at a rate multiple to the data rate
- Statistics thread:
  - Retrieves monitoring data from the queues in set intervals
  - Combines the data



2

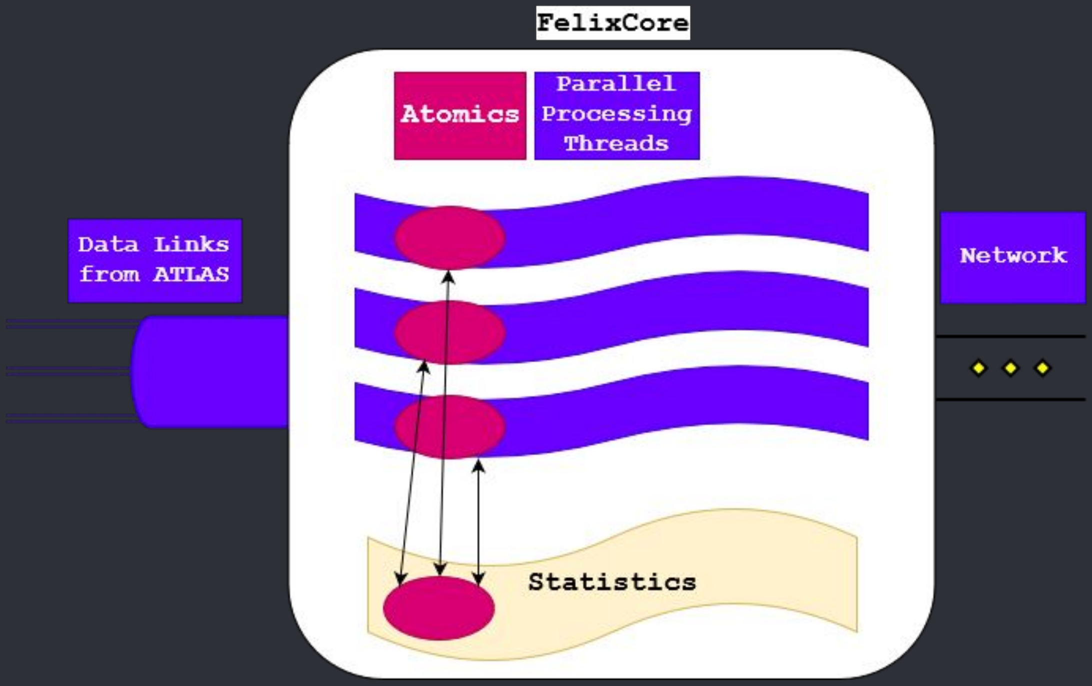
The solution

Atomics

Efficiently synchronizing data from the routing threads

The new statistics module:

- Hardware supported (x86/amd64) ***Atomic Operations***
- Atomic variables

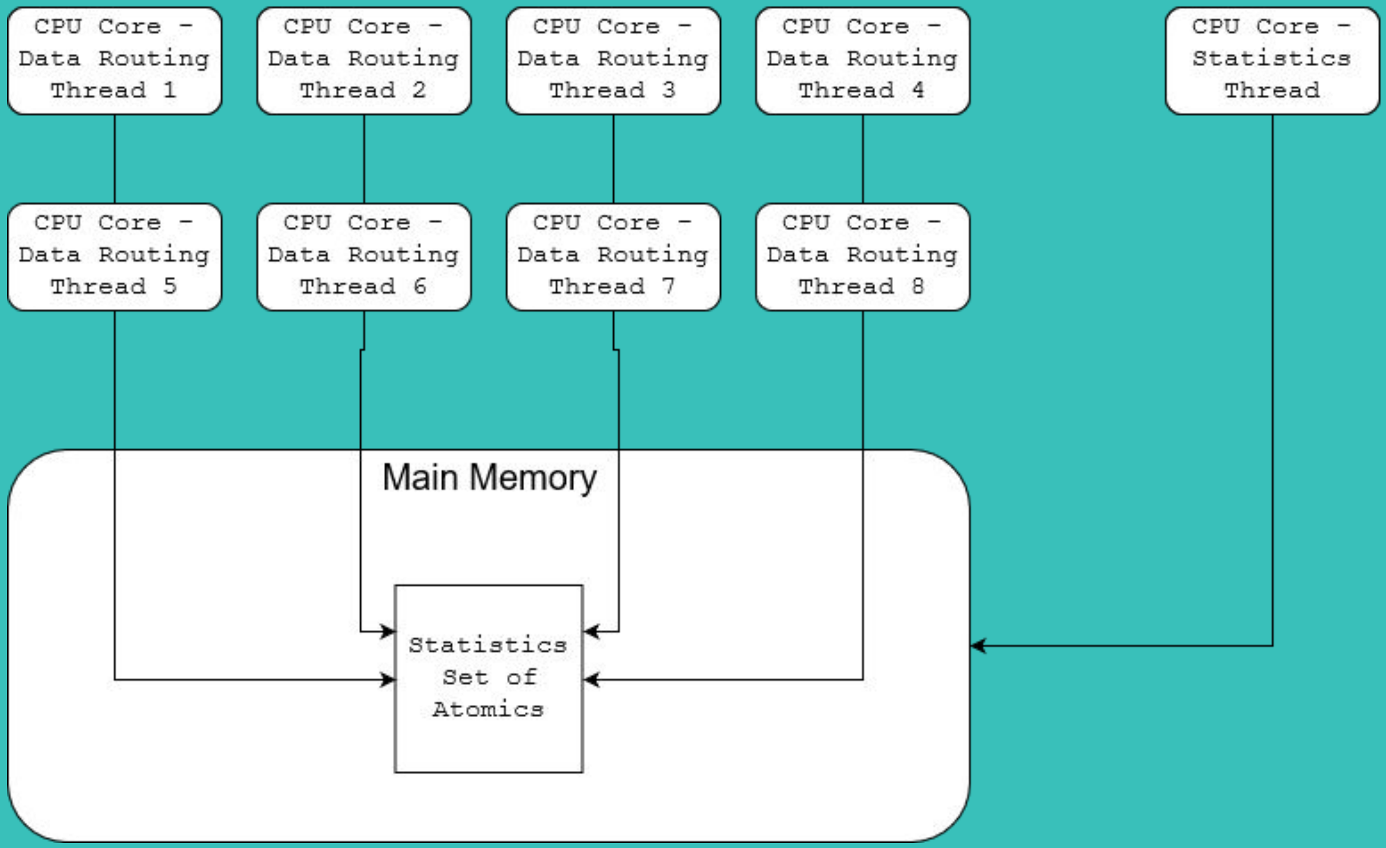


Three implementations

1) Central Atomics

A set of atomic variables accessible from **all** the parallel routing threads.

1) Central Atomics



Three implementations

1) Central Atomics

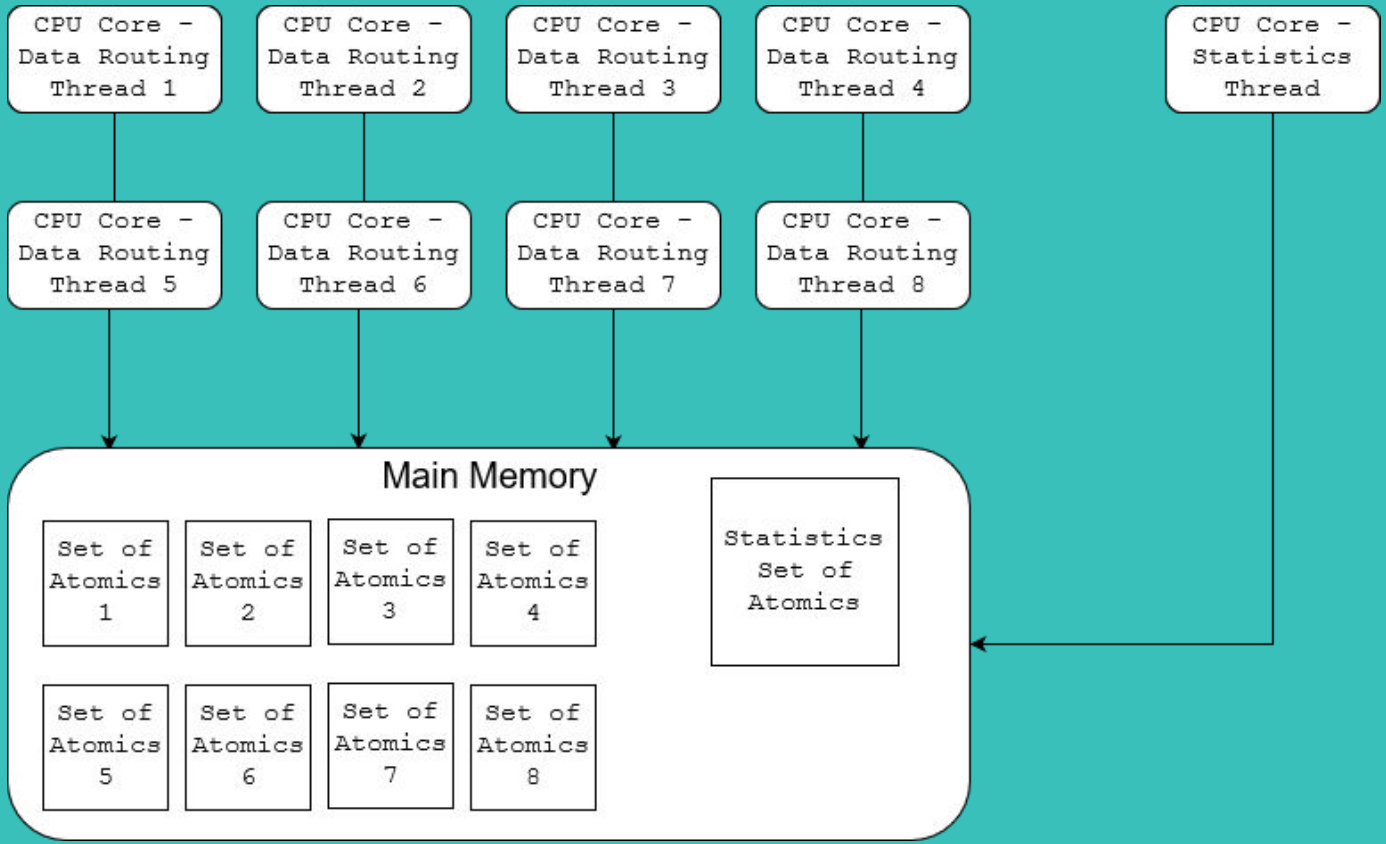
A set of atomic variables accessible from **all** the parallel routing threads.

2) Separate Atomics - Push Config

A set of atomic variables for **each** thread.
The accumulated values are held by the statistics thread.

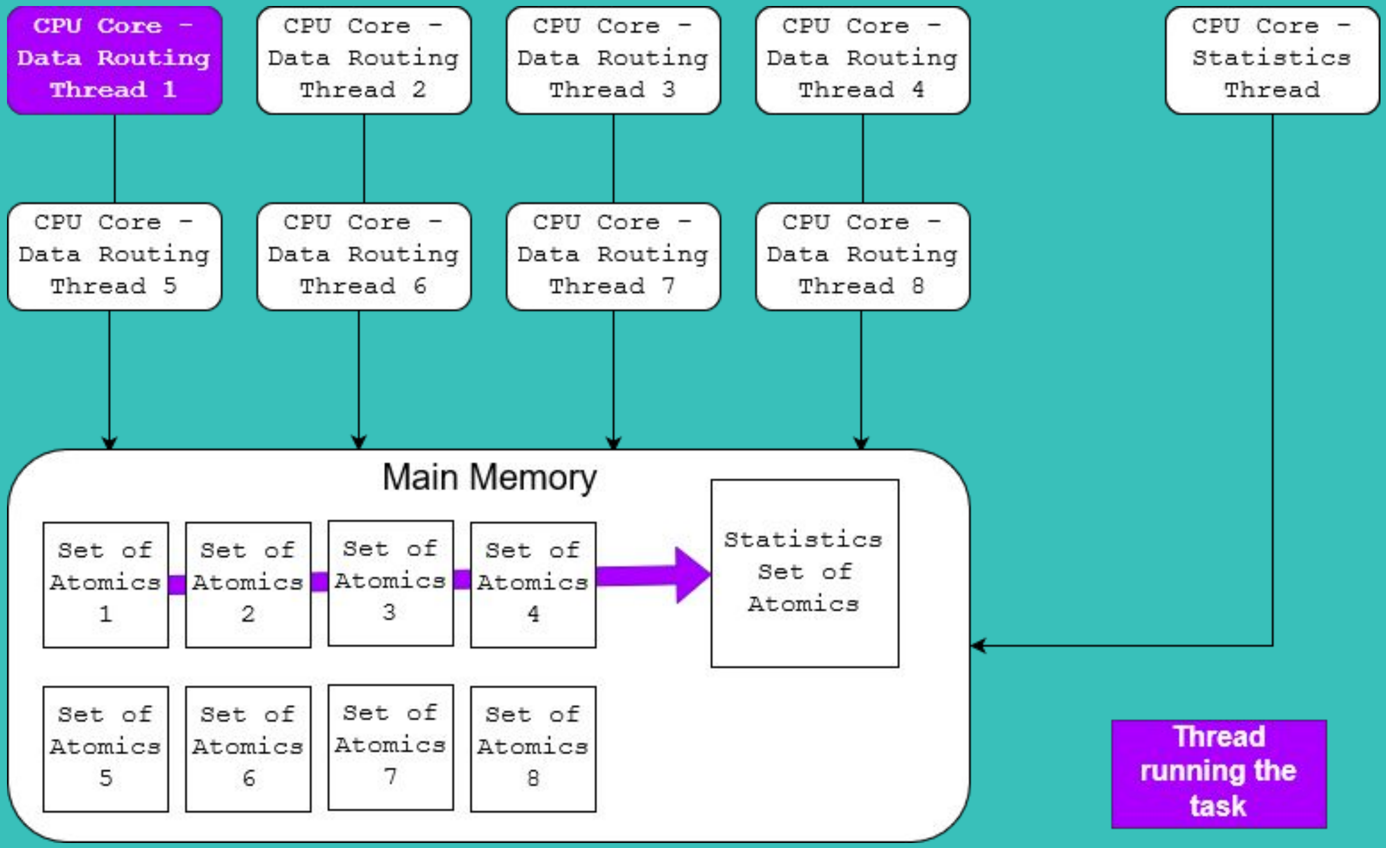
The routing threads push their own set to the statistics one.

2) Separate Atomics - Push Config



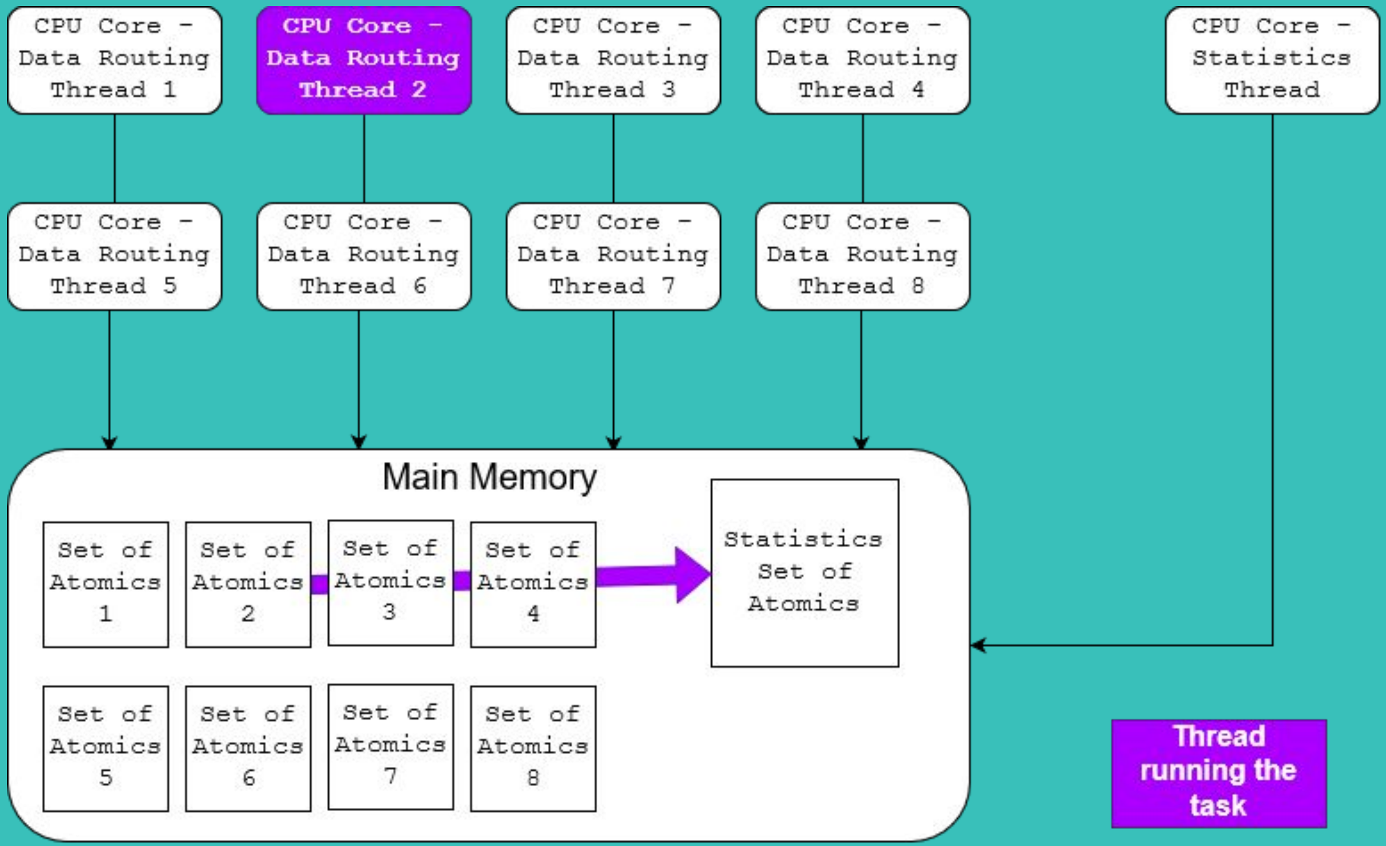
2) Separate Atomics - Push Config

Expected



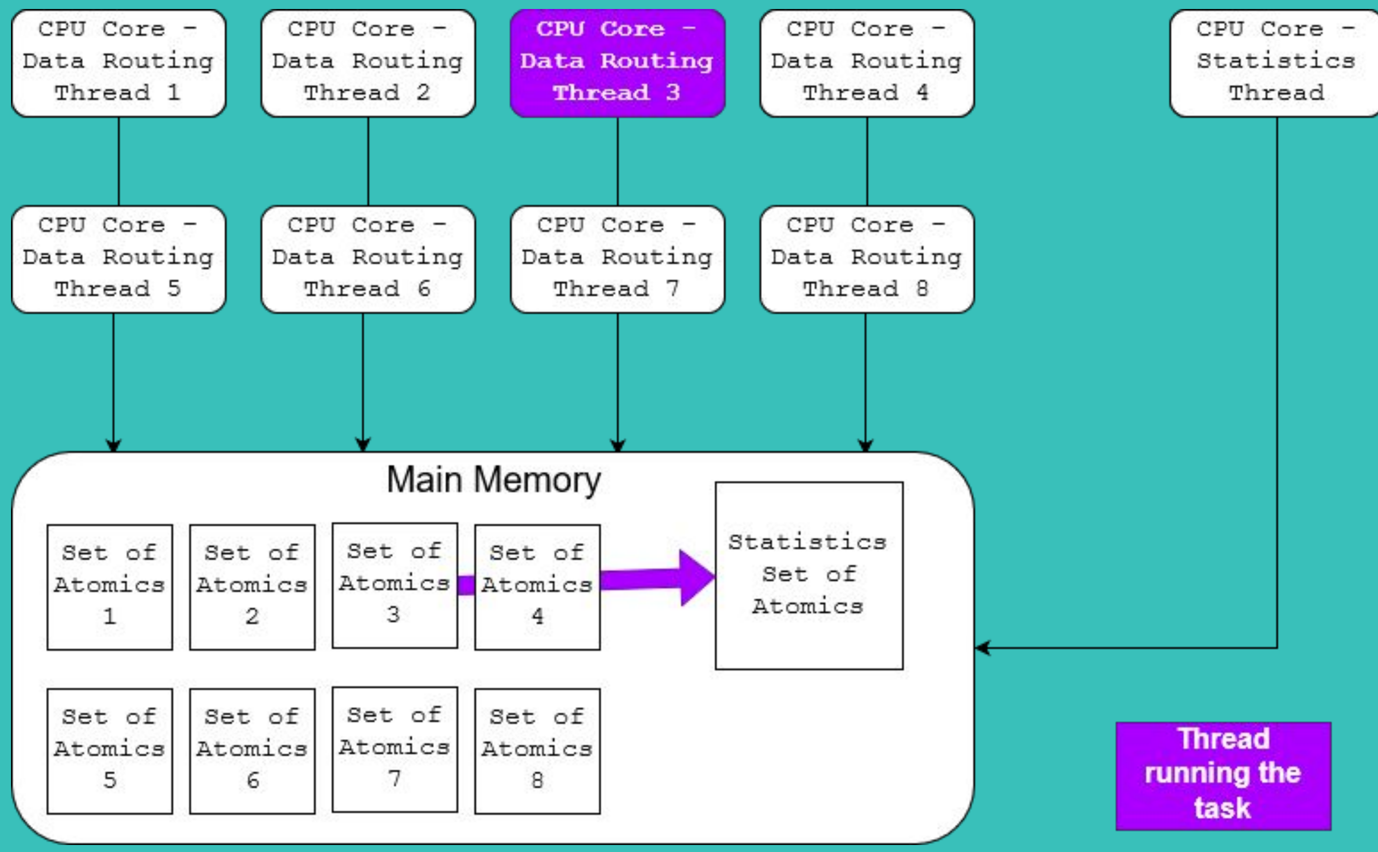
2) Separate Atomics - Push Config

Expected



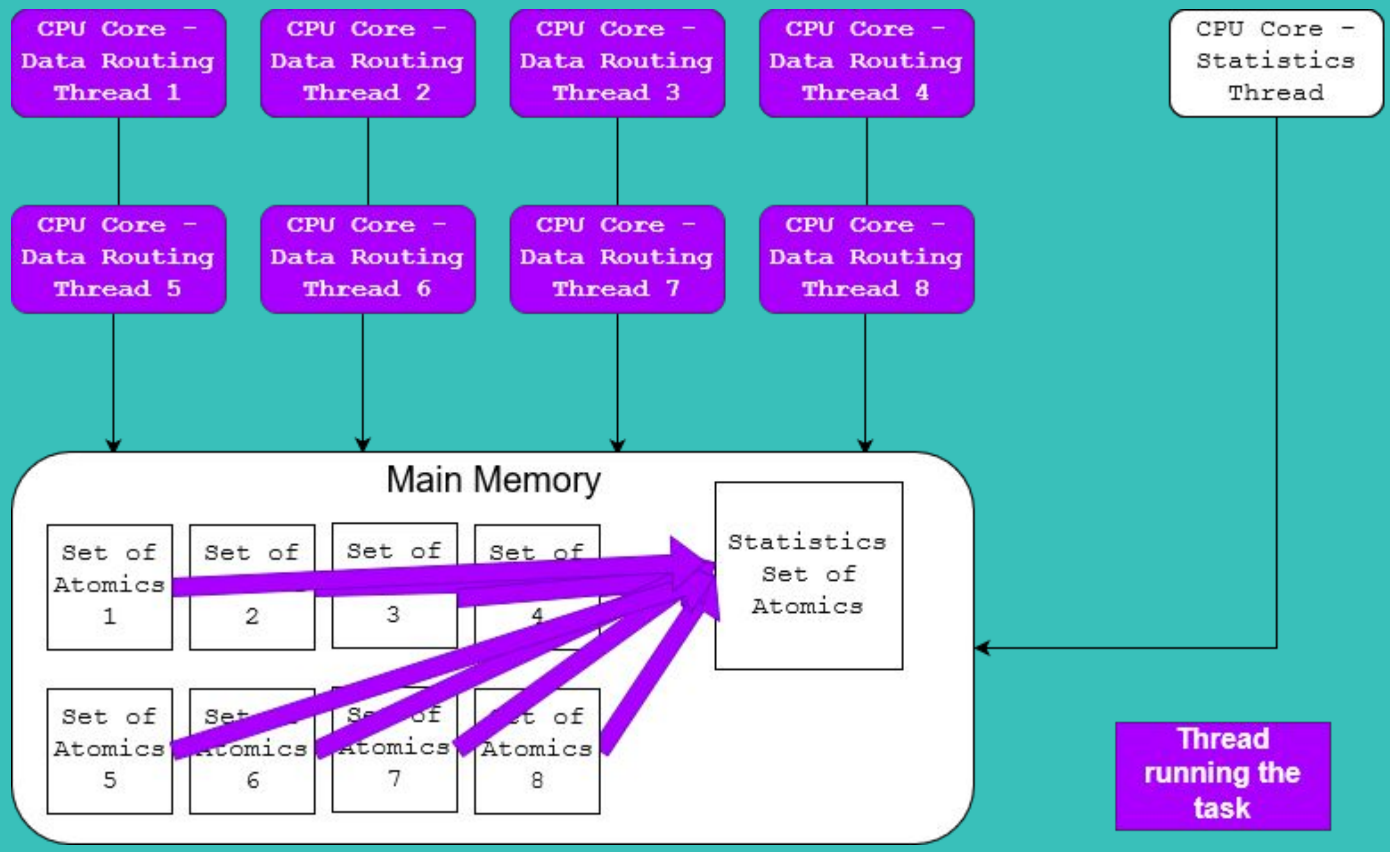
2) Separate Atomics - Push Config

Expected



2) Separate Atomics - Push Config

Worst Case Scenario



Three implementations

1) Central Atomics

A set of atomic variables accessible from **all** the parallel routing threads.

2) Separate Atomics - Push Config

A set of atomic variables for **each** thread.
The accumulated values are held by the statistics thread.

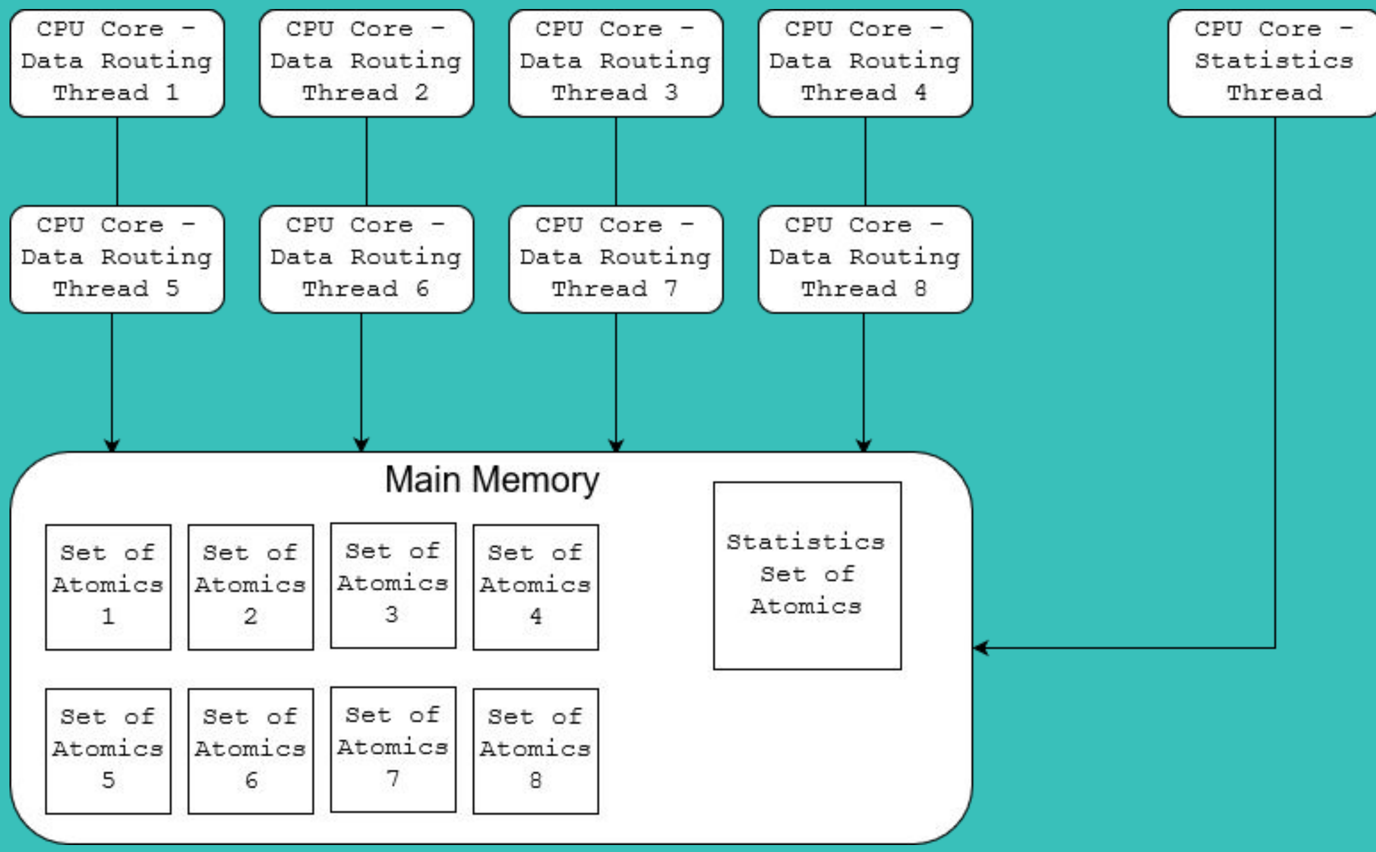
The routing threads push their own set to the statistics one.

3) Separate Atomics - Pull Config

A set of atomic variables for **each** thread.
The accumulated values are held by the statistics thread.

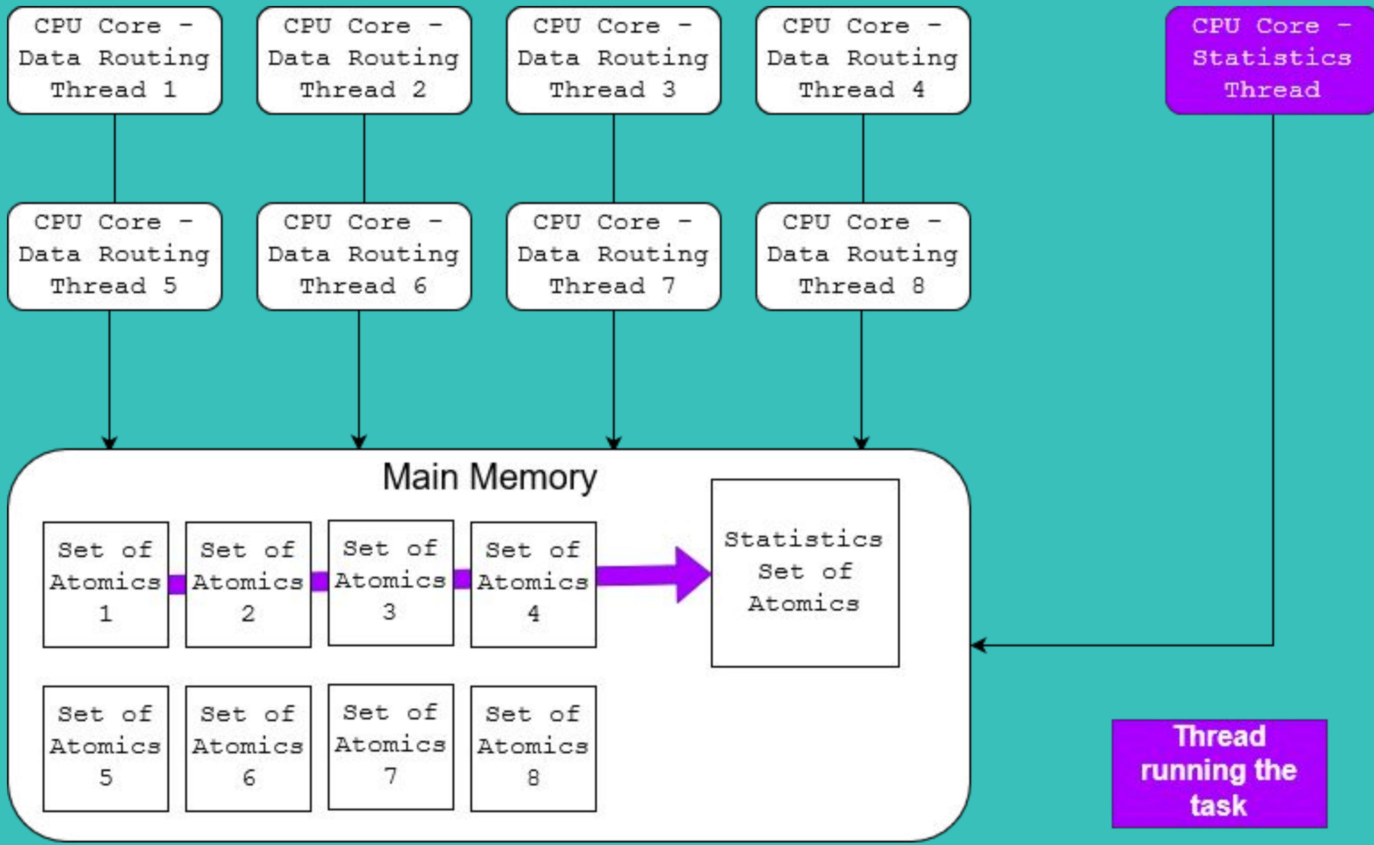
The statistics thread pulls the partial sets from the routing threads.

3) Separate Atomics - Pull Config



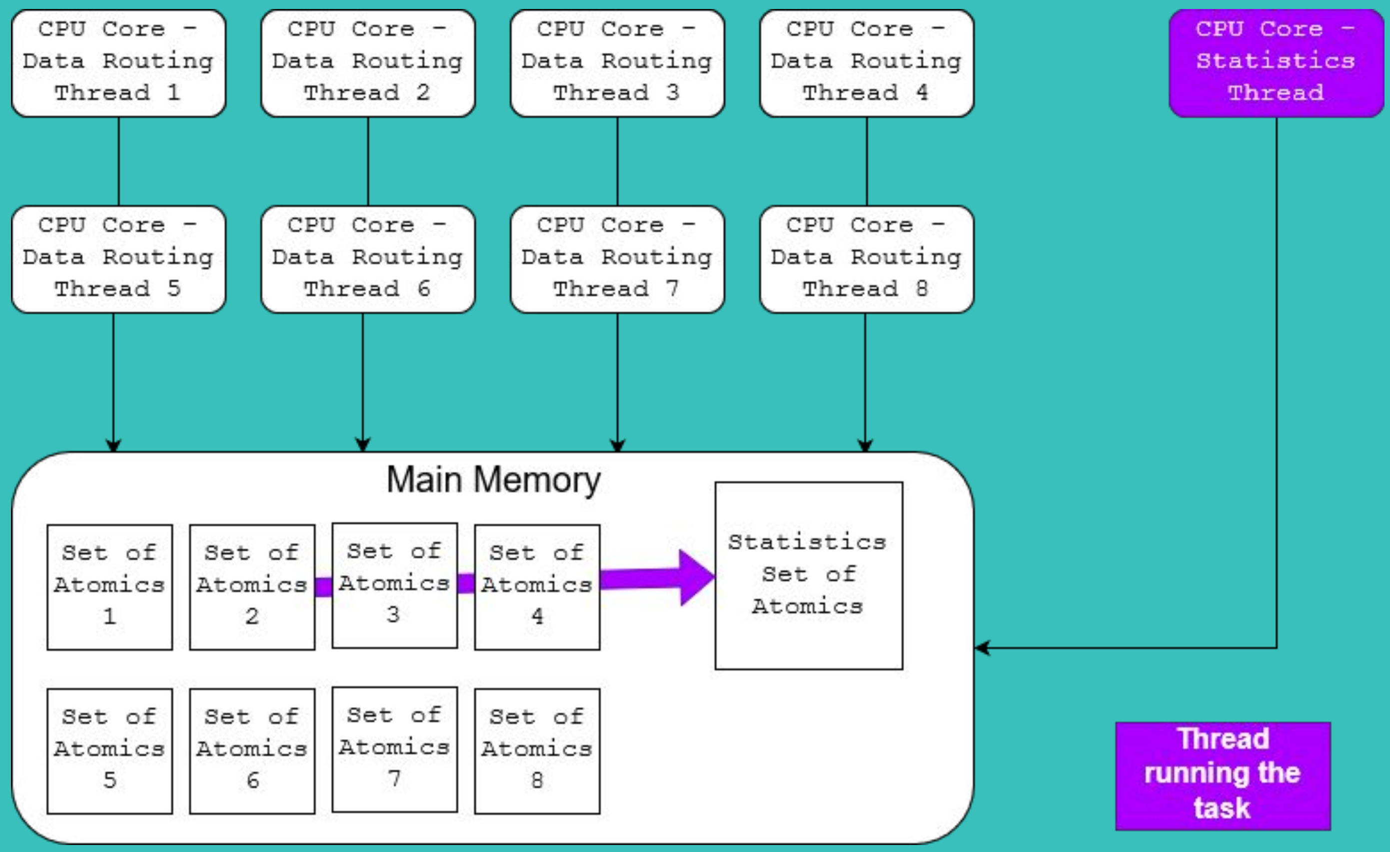
3) Separate Atomics - Pull Config

Always executed serially



3) Separate Atomics - Pull Config

Always executed serially



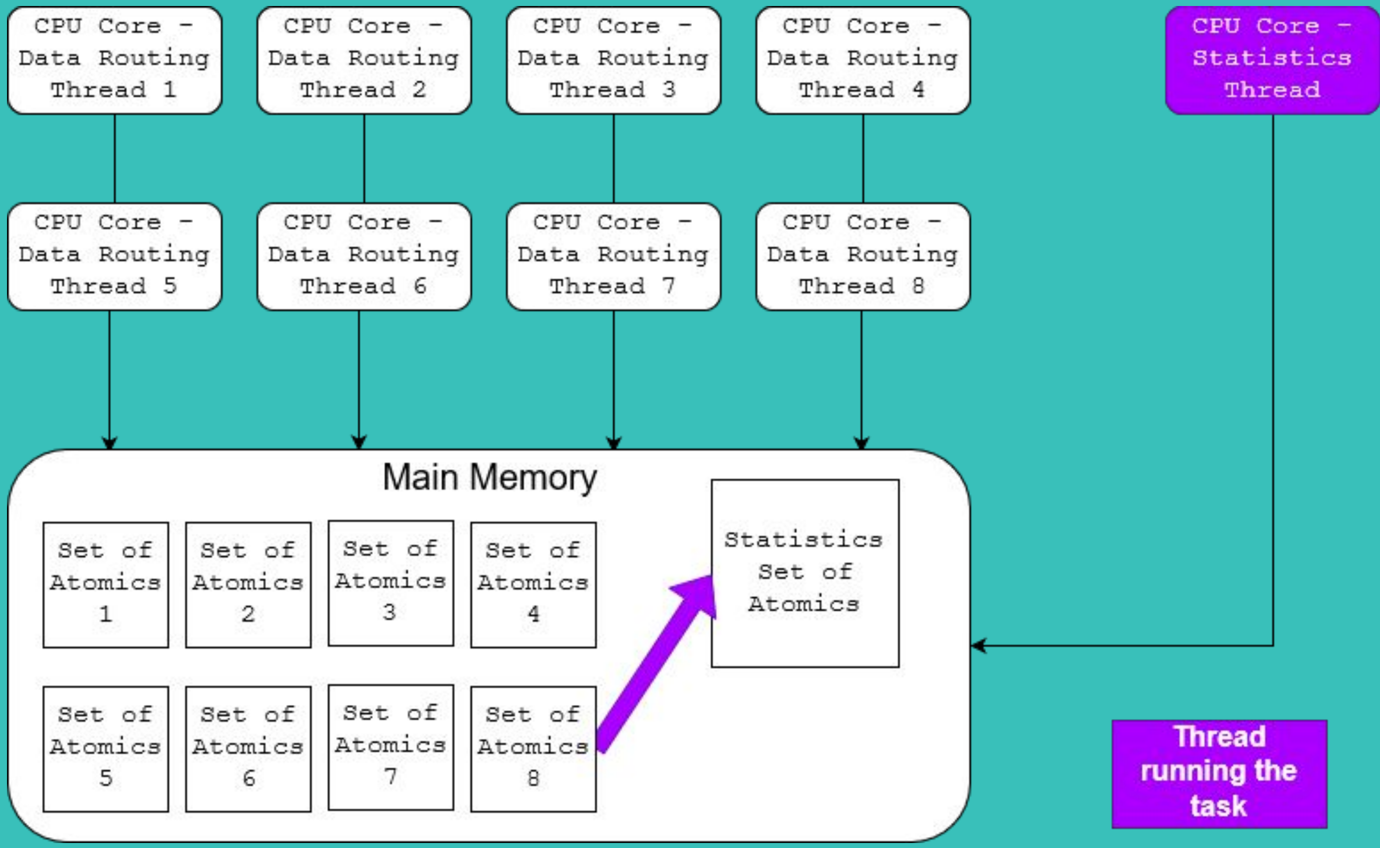
3) Separate Atomics - Pull Config

Always executed serially

...

3) Separate Atomics - Pull Config

Always executed serially



The results of the three implementations

1) Central Atomics

Negligible performance gains (< 5%).

2) Separate Atomics - Push Config

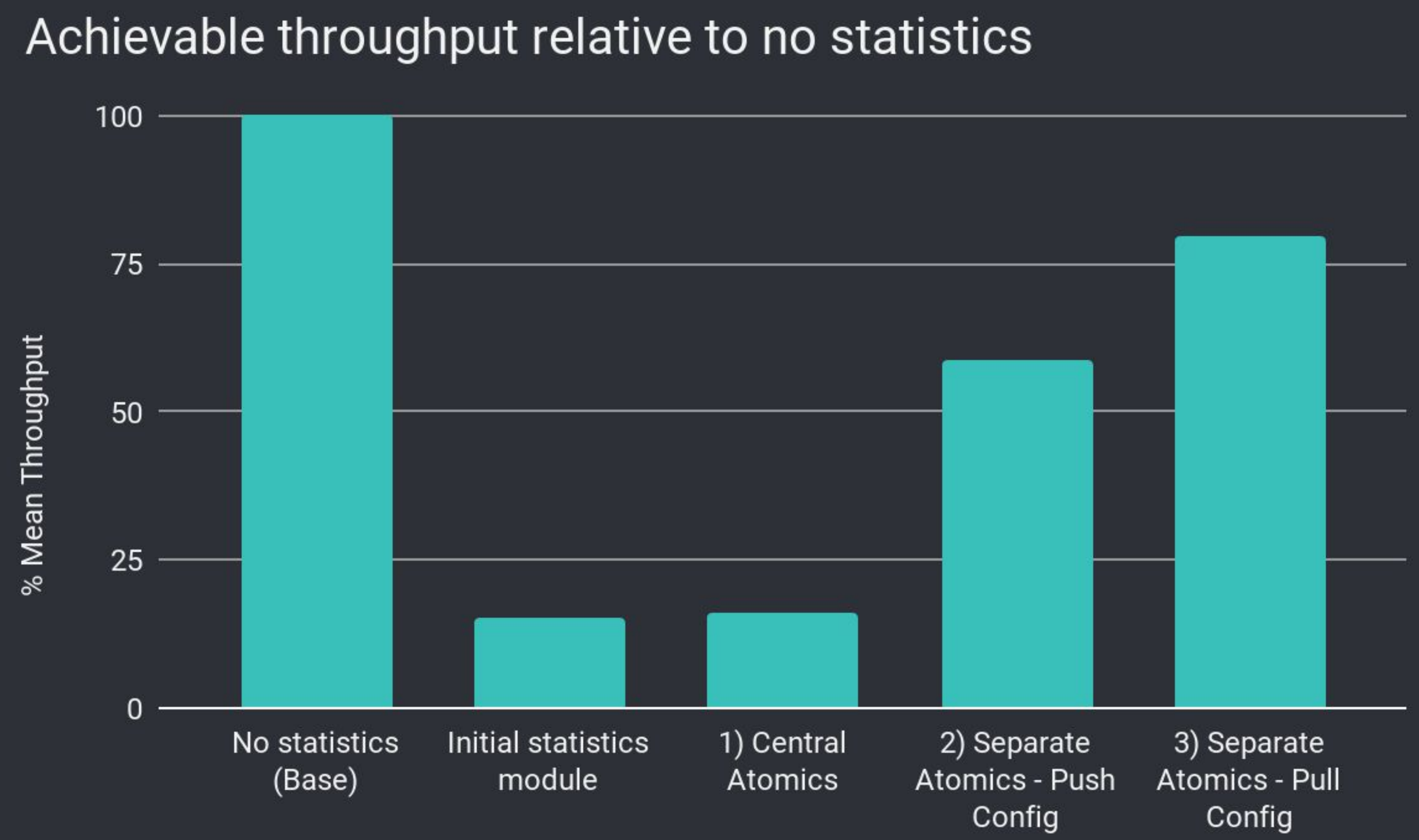
~400% of performance compared to the initial statistics module. 60% to 70% of target performance.

3) Separate Atomics - Pull Config

~**500%** of performance compared to the initial statistics module.

~5x

the initial throughput performance.



Comparison of the final implementation to the initial one

3

Conclusions

Parallel processing: Concurrency optimization matters

Cache invalidation

Measurements through Intel® VTune™ Amplifier suggested that the “1) Central Atomics” implementation was suffering from performance issues which were manifested as cache invalidation.

What we did and what we learned.

Result

The performance gains were enough for us to meet our internal target.

Lesson

Change on the concurrency while using atomics could yield totally different results.

Thanks!

ANY QUESTIONS?

Summary

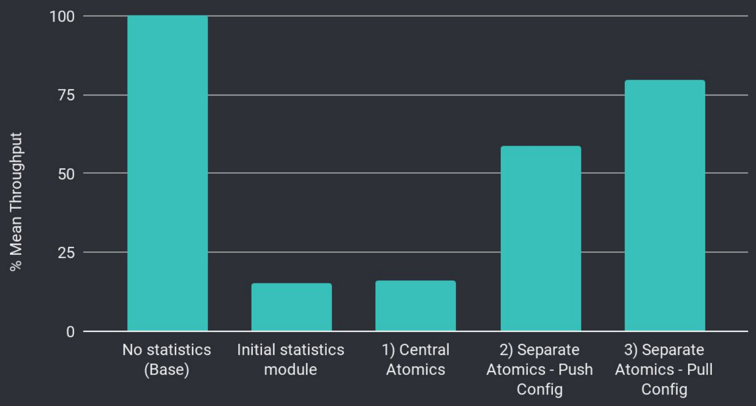
~500%

FELIX data throughput performance gain when using the final statistics module

Concurrency Levels

A significant difference in results

Achievable throughput relative to no statistics



CREDITS

Special thanks to all the people who wrote the original software and helped with my work:

- Jörn Schumacher
- Mark Dönszelmann