Making Sense of Multi-threaded Application Performance at Scale with *NonSequitur*

Augustine Wong,Paul Bucci,
Ivan Beschastnikh
Alexandra Fedorova

THE UNIVERSITY
OF BRITISH COLUMBIA

The Impact of Performance Bugs

Compromised but operational system

Degradation in:

Energy

Latency

R. Krishna, M. S. Iqbal, M. A. Javidian, B. Ray, and P. Jamshidi. Cadet: Debugging and fixing misconfigurations using counterfactual reasoning.arXiv preprint arXiv:2010.06061, 2020.

The Cost of Fixing Performance Bugs

More difficult to fix than other types of bugs

[W]e found that performance bugs take more time to fix, are fixed by more experienced developers and require more changes to more lines than non-performance bugs [1].

[1] S. Zaman, B. Adams, and A. E. Hassan. A qualitative study on performance bugs. In 2012 9th IEEE working conference on mining software repositories (MSR), pages 199–208. IEEE, 2012.

A Case Study of Performance Comprehension

We performed an empirical case study to examine the nature of *performance comprehension* [2]

Information

Tools

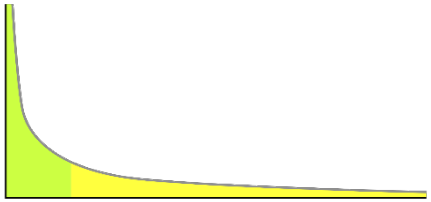
Processes

[2] Alexandra Fedorova, Craig Mustard, Ivan Beschastnikh, Julia Rubin, Augustine Wong, Svetozar Miucin, and Louis Ye. 2018. Performance comprehension at WiredTiger. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 83–94.

Analyzing Performance Comprehension

We analyzed 44 JIRA tickets that tracked performance issues at WiredTiger

Key Findings on Performance Comprehension



WiredTiger developers spent the majority of their performance debugging time investigating latency spikes

Developers were interested in knowing what threads did over time

Developers collected and analyzed large amounts of information

Performance Analysis Tool Requirements

Requirements	Description
R_Latency	Find when latency spikes occur
R_Time	Display thread behavior over time
R_Large	Process large execution traces

Current State of Performance Debugging

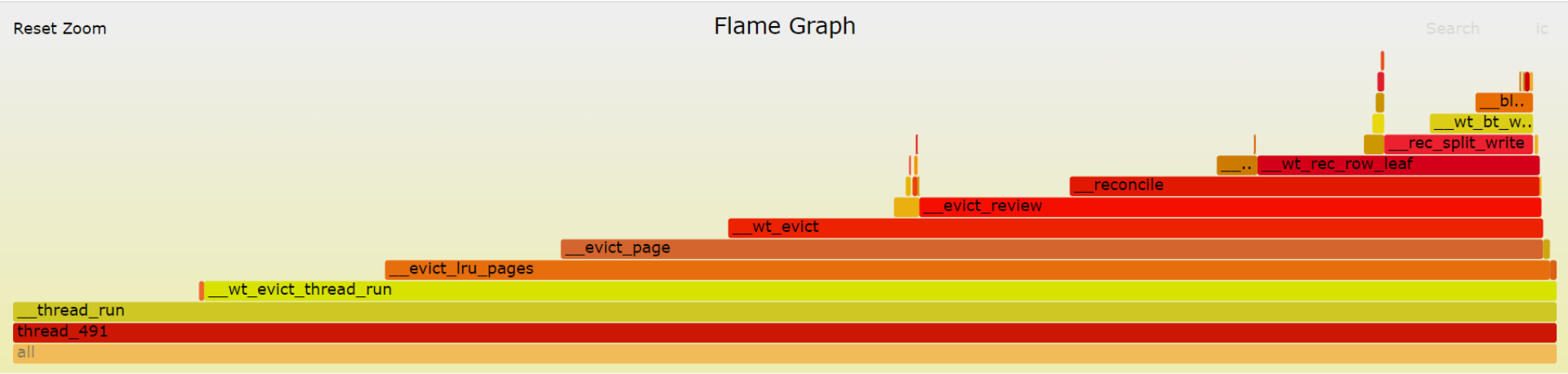
Sue Doe is a software developer tasked with investigating latency spikes captured in a large execution trace.

RocksDB trace

Thread ID	Size	
	# Function Calls	# Different Functions
1	2,466,295	74
2	2,464,045	75
3	194,042	247

What can she do?

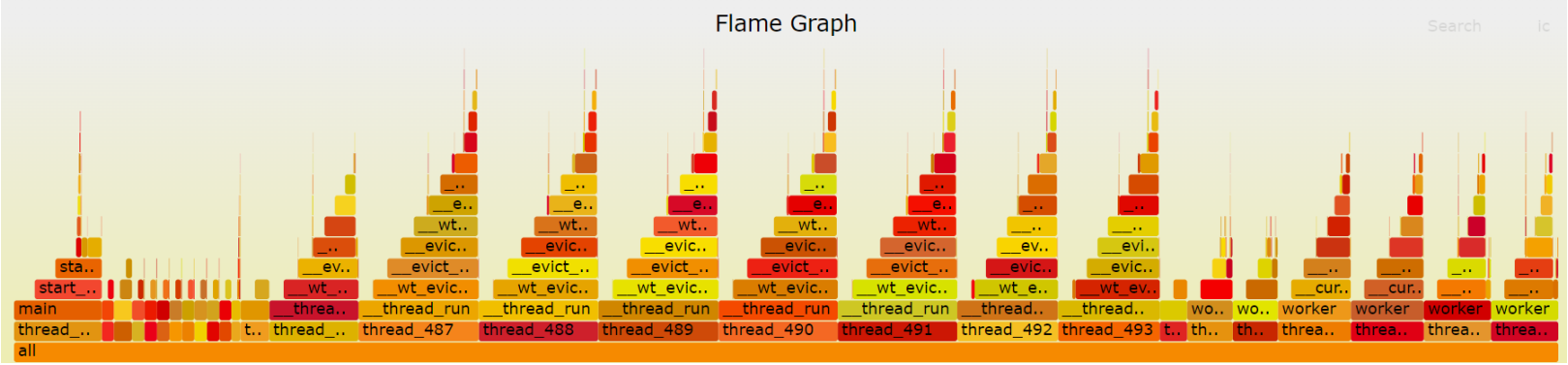
Option 1: Use FlameGraphs?



Merges callstack samples together and sorts them alphabetically

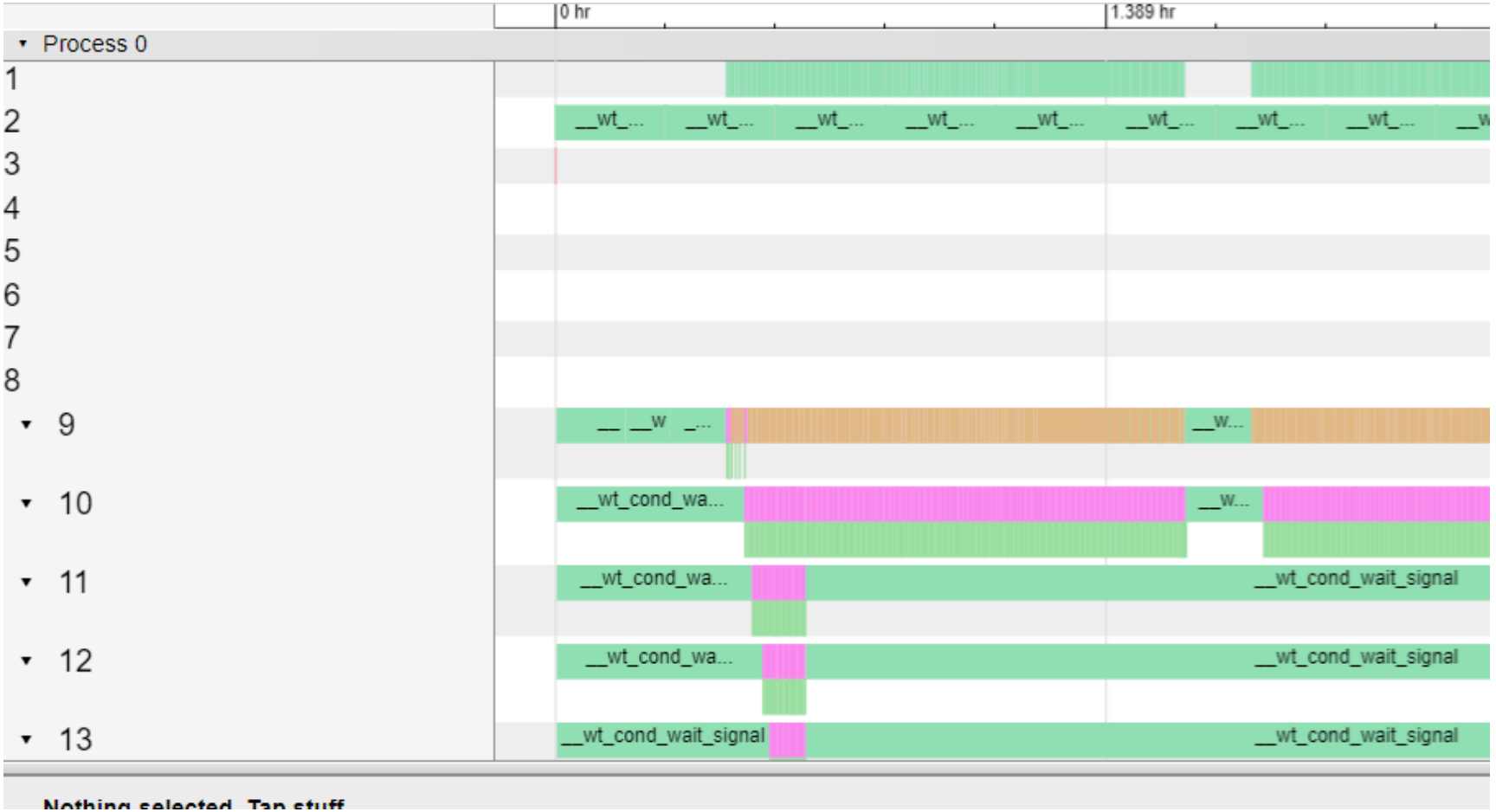
Does not show time 

Option 1: Use FlameGraphs?

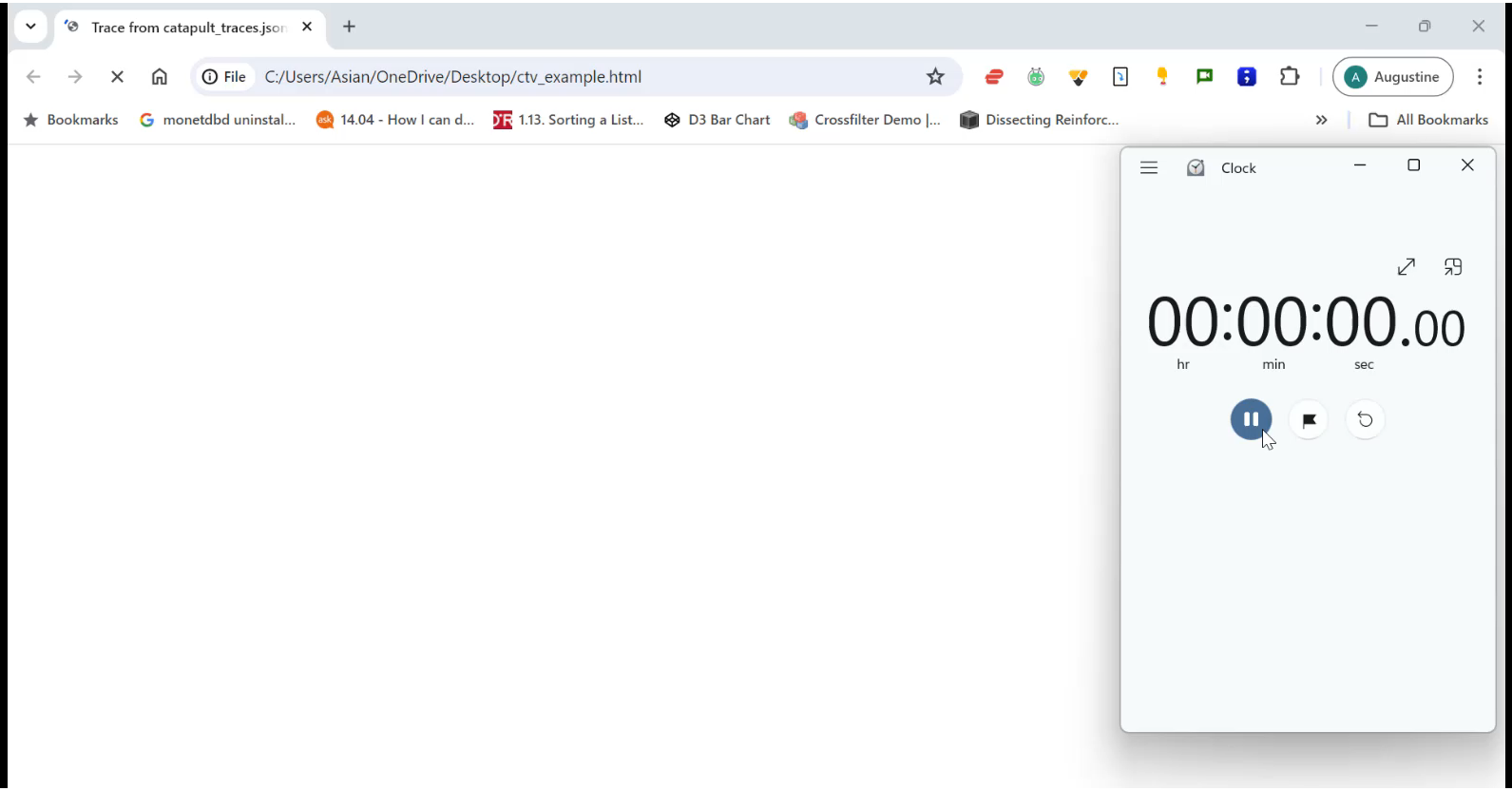


Tool Requirements		
R_Latency	R_Time	R_Large
✗	✗	✓

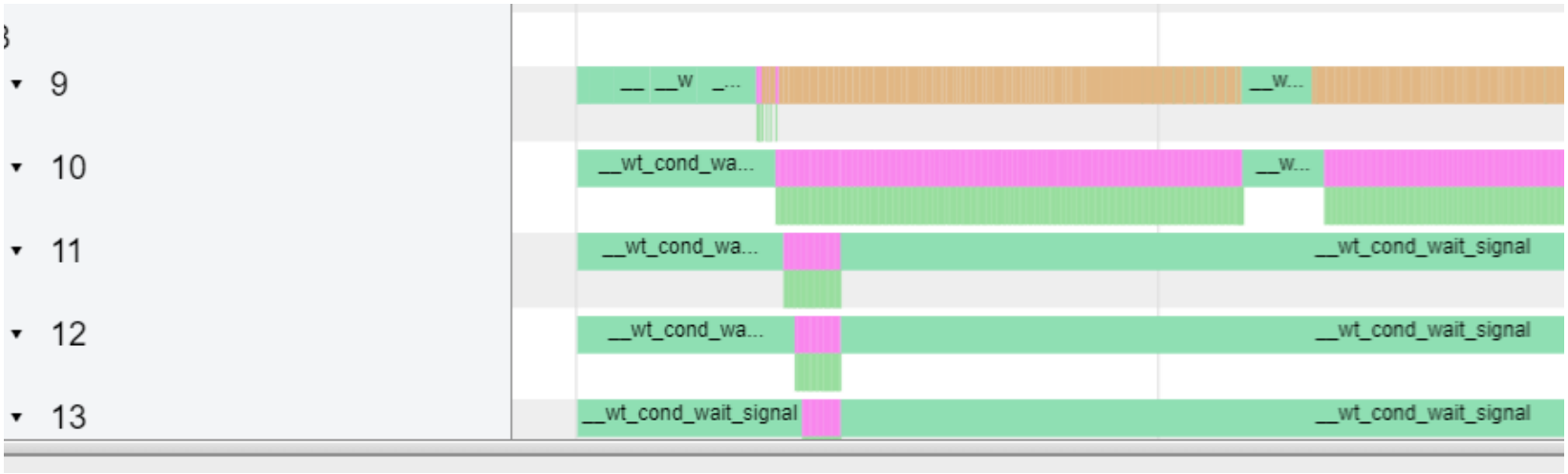
Option 2: Chrome TraceViewer?



Option 2: Chrome TraceViewer?



Option 2: Chrome TraceViewer?



Tool Requirements		
R_Latency	R_Time	R_Large
✓	✓	✗

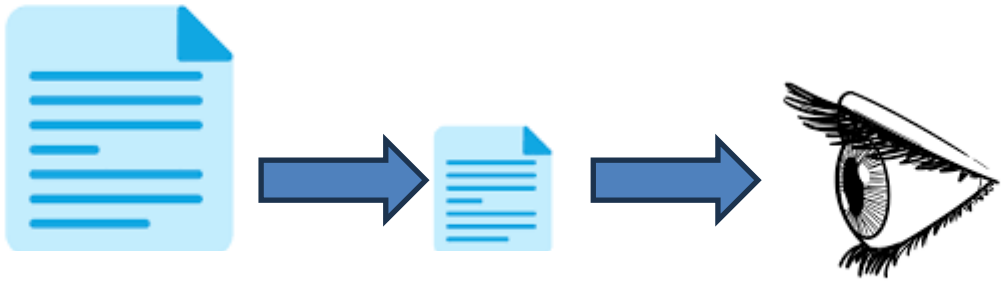
Tries to visualize everything in an execution trace!

Other Tools?

Tools	Tool Requirements		
	R_Latency	R_Time	R_Large
Zinsight [3]	✗	✓	✓
TraceViz [4]	✗	✓	✓
SyncTrace [5]	✓	✓	✗
ExtraVis [6]	✗	✓	✗
TraceDiff [7]	✗	✓	✗
Compress [8]	✗	✗	✓
Phase Finder [9]	✗	✗	✓
Sabalan [10]	✗	✗	✓

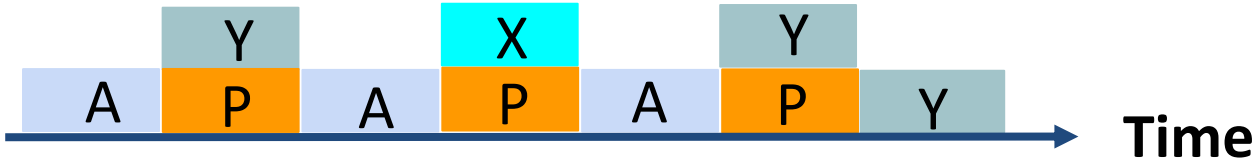
[3] De Pauw et al. (2010)
[4] Dautriche et al. (2016)
[5] Karran et al. (2013)
[6] Cornelissen et al. (2008)
[7] Trumper et al. (2013)
[8] Hamou-Lhadj et al. (2002)
[9] Pirzadeh et al. (2010)
[10] Alimadadi et al. (2018)

Our Solution: Compress + Visualize



RegTime NonSequitur

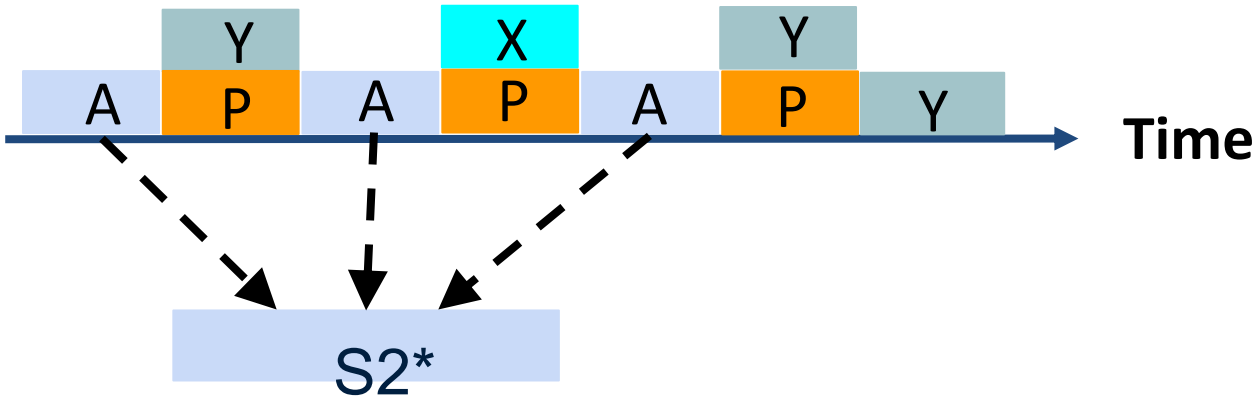
Our Solution: Compression with RegTime



Function Sequence

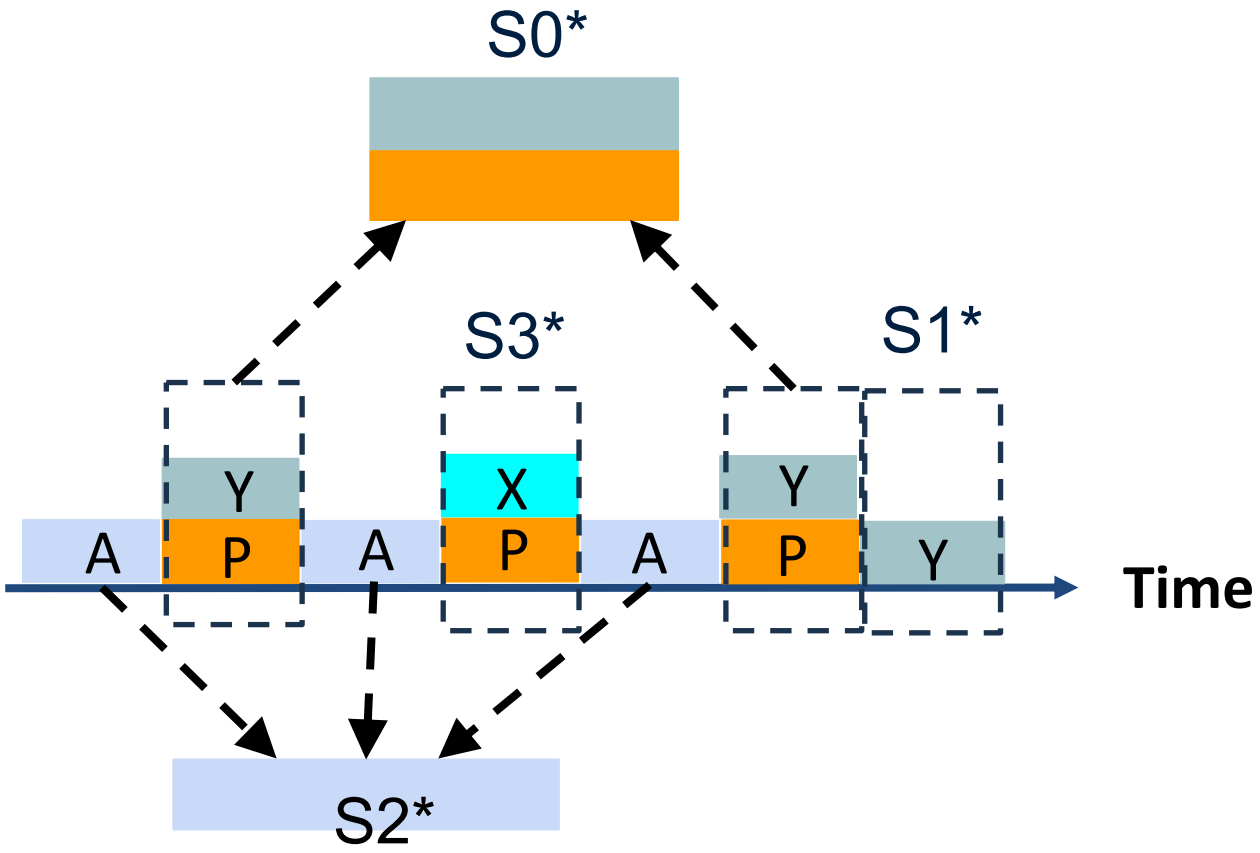
RegTime Expression?

Our Solution: Compression with RegTime



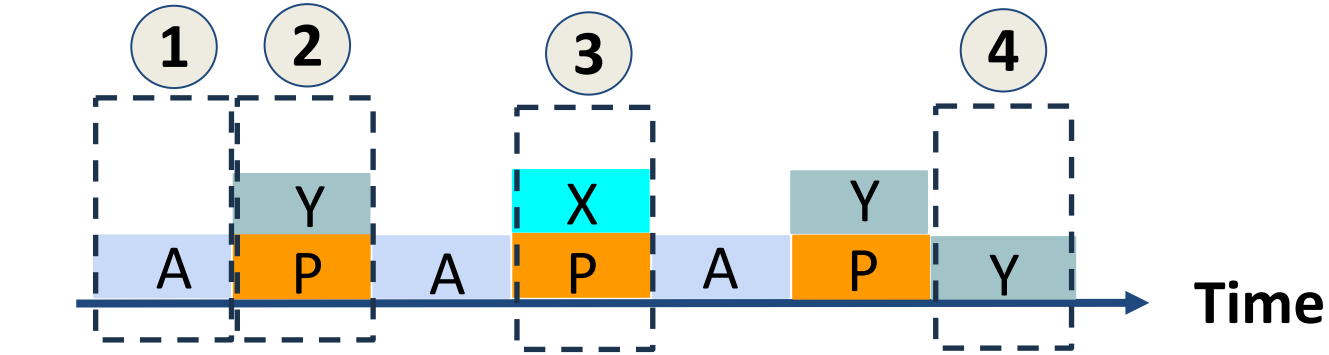
1) Merge same callstacks together

Our Solution: Compression with RegTime

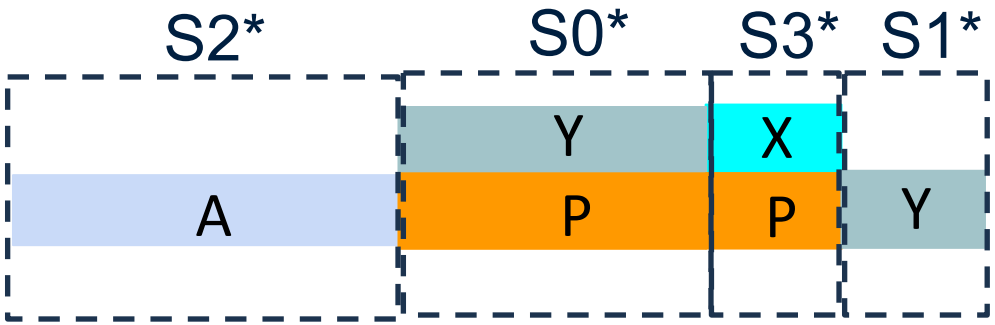


1) Merge same callstacks together

Our Solution: Compression with RegTime

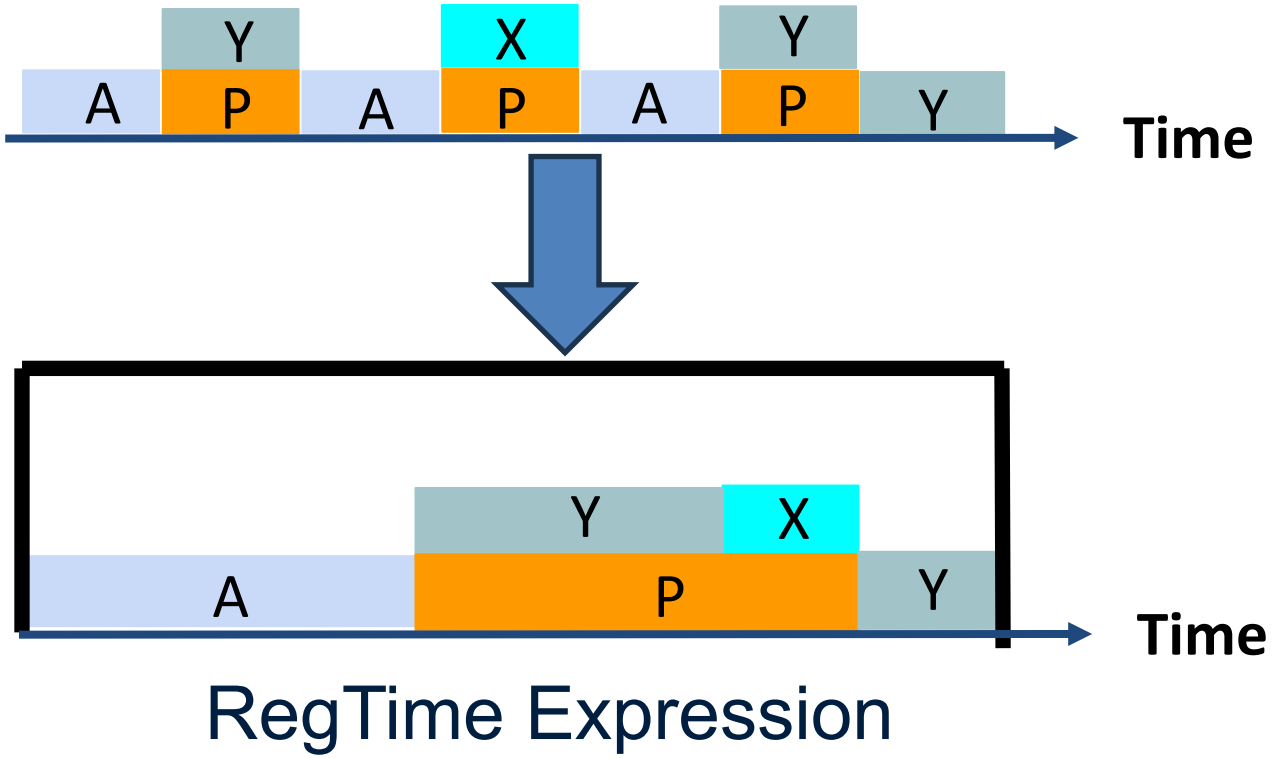


First Occurrences of each unique callstack



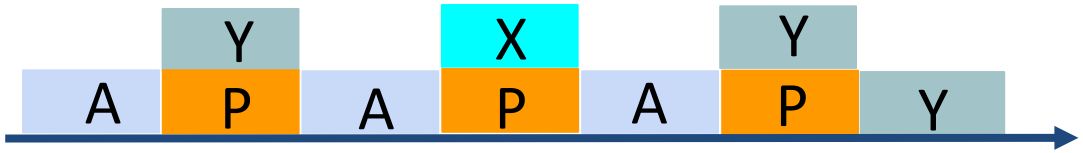
2) Arrange groups of merged callstacks

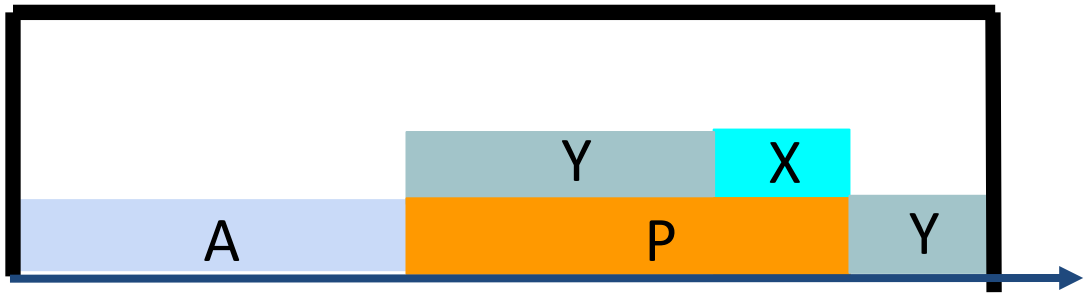
Visualizing RegTime Expressions



RegTime Expression

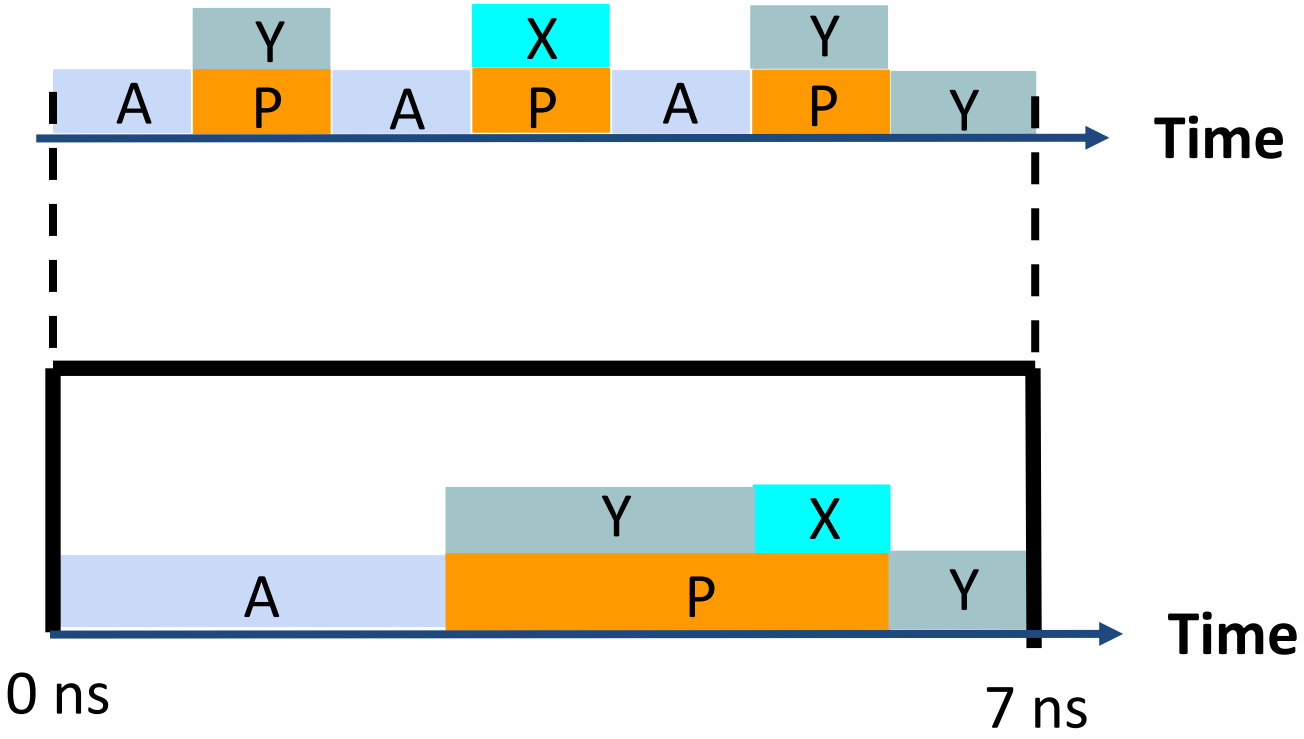
Visualizing RegTime Expressions





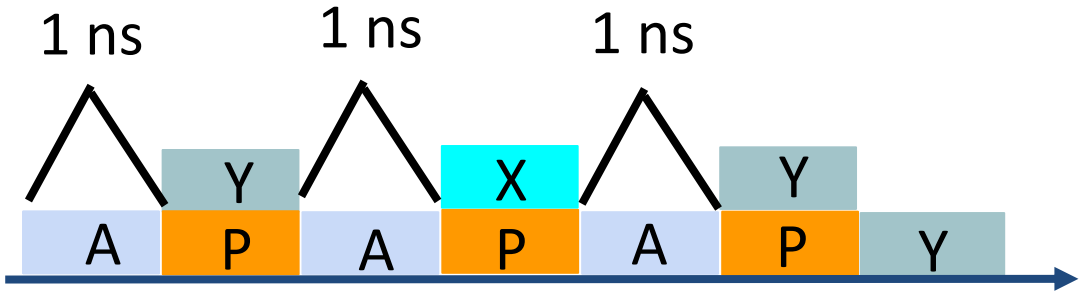
❌ **Precise ordering of function calls**

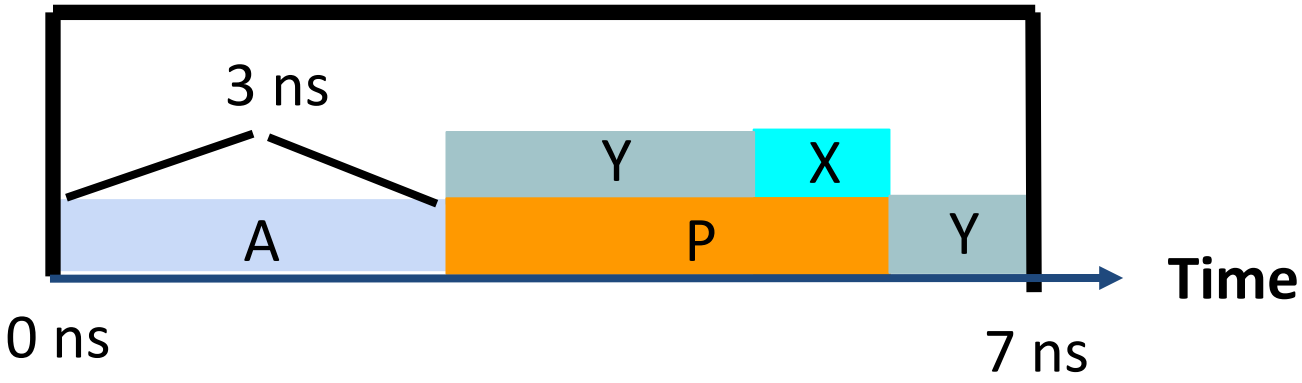
Visualizing RegTime Expressions



A box encodes start and end time of the function sequence

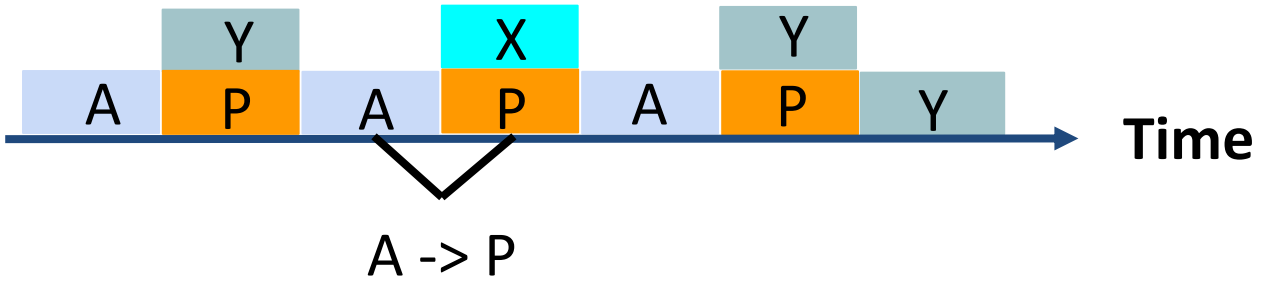
Visualizing RegTime Expressions

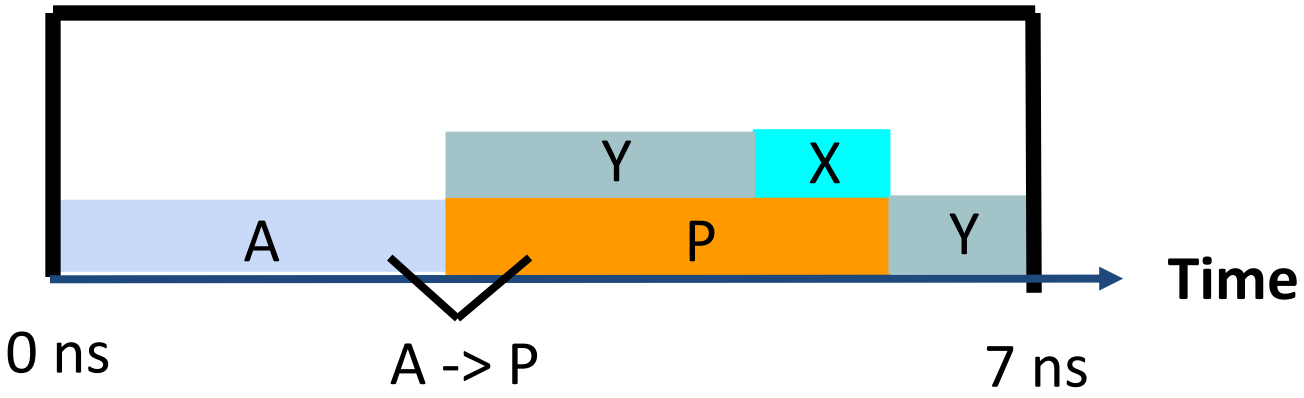




✓ **Cumulative durations of functions**

Visualizing RegTime Expressions





✓ **Loose causal relationships**

Let's Visualize This Trace with NonSequitur

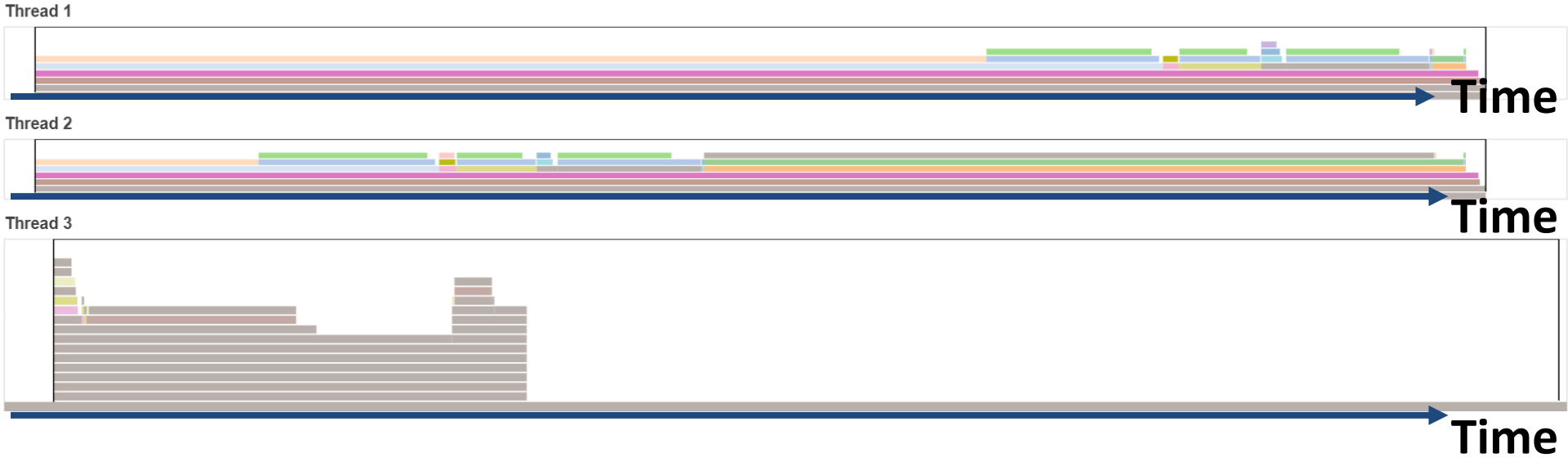
RocksDB

Thread ID	Size	
	# Function Calls	# Different Functions
1	2,466,295	74
2	2,464,045	75
3	194,042	247

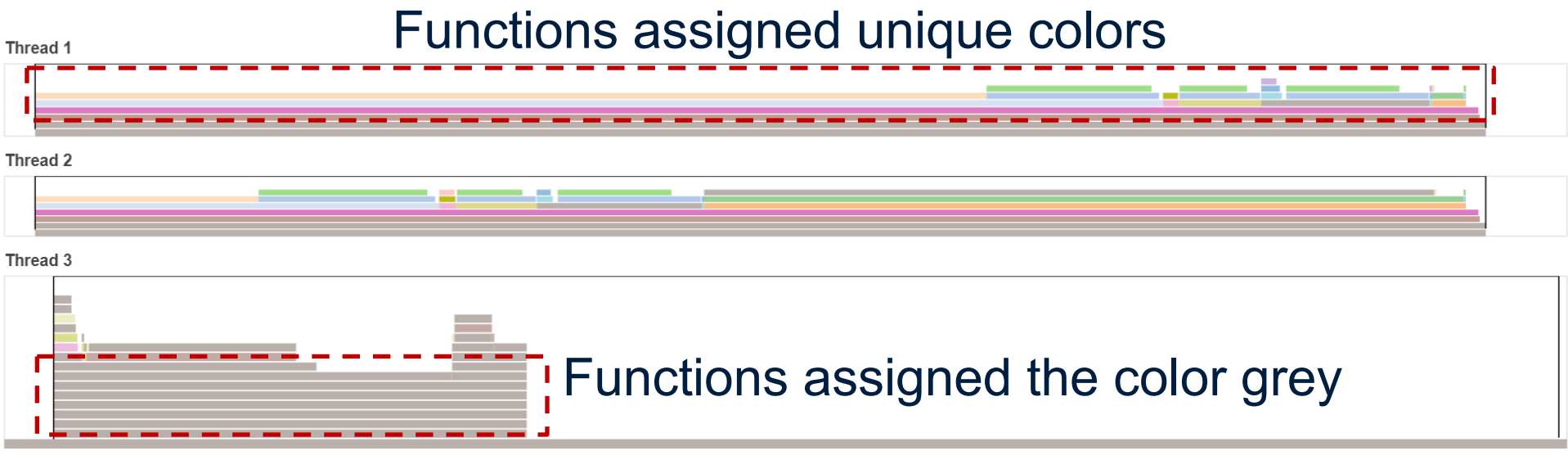
NonSequitur Version 1

One RegTime Expression Per Thread

NonSequitur Version 1



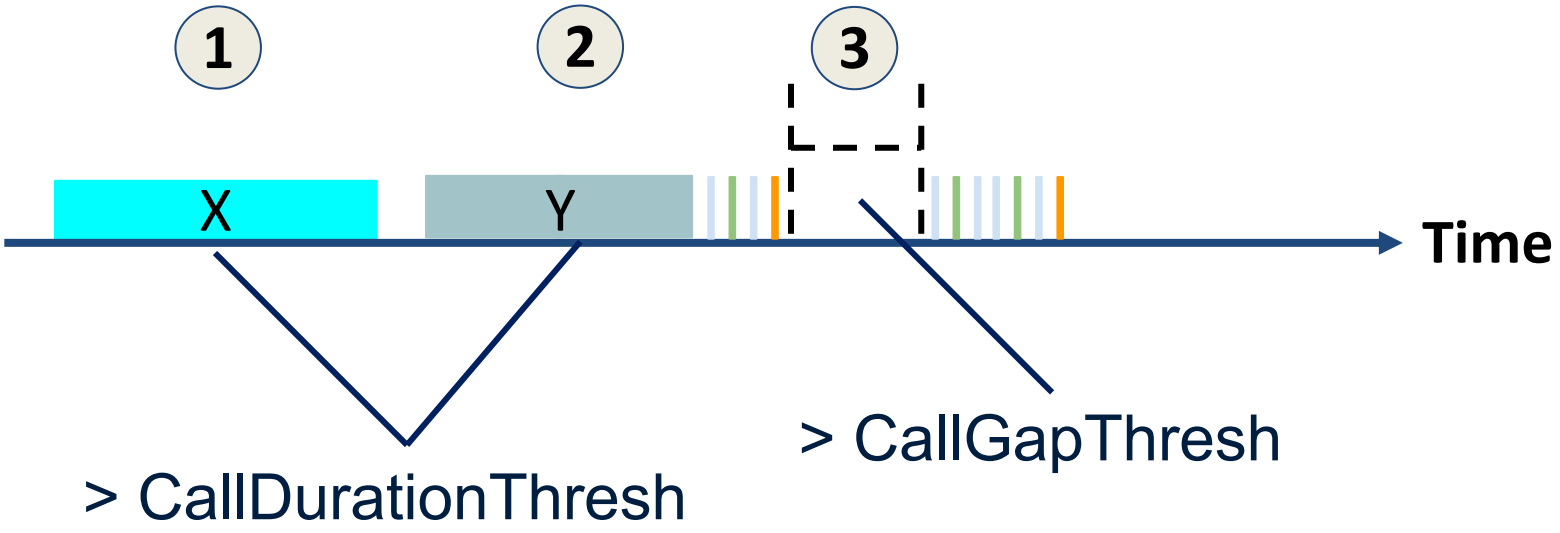
NonSequitur Version 1



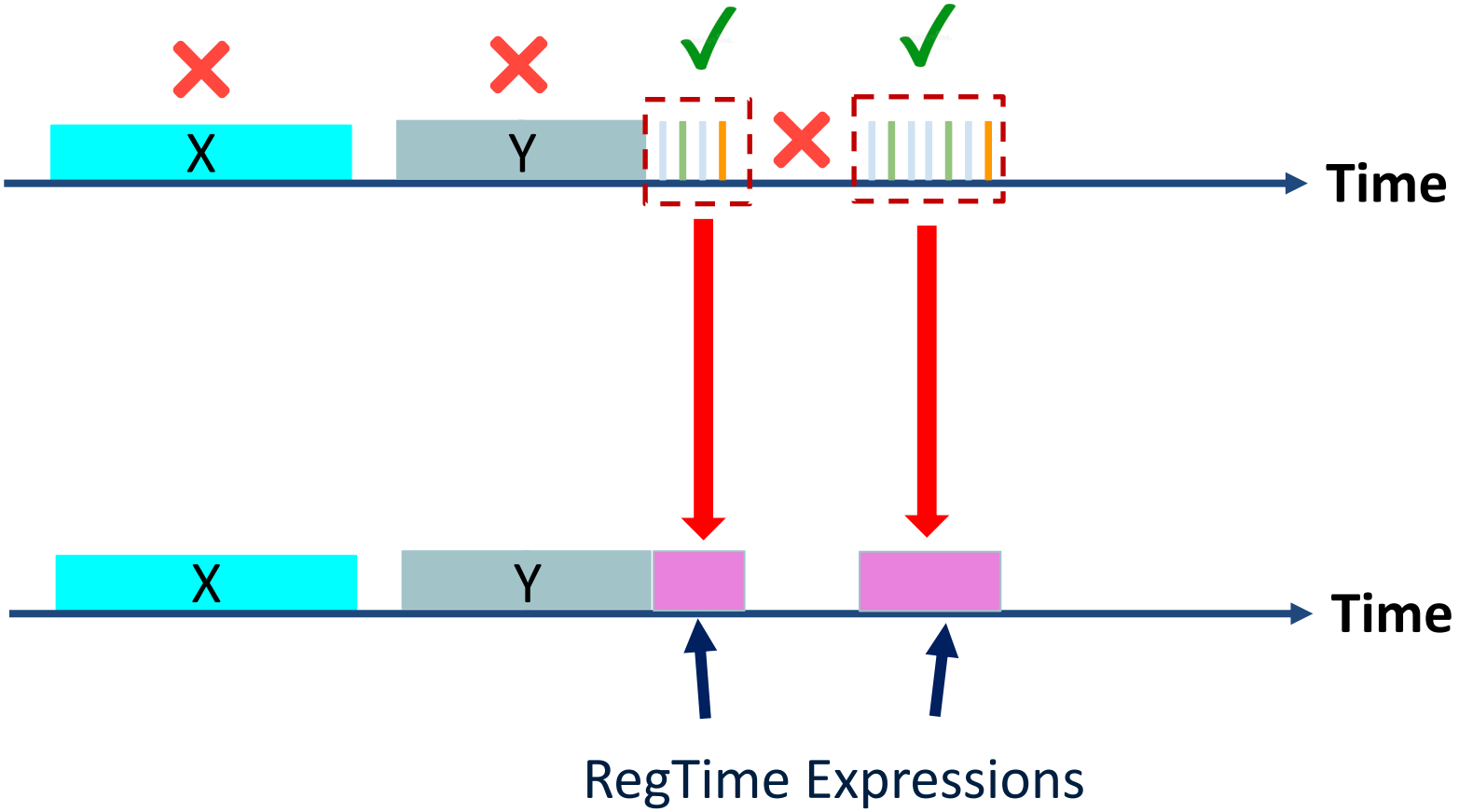
NonSequitur Version 1

Tool Requirements		
R_Latency	R_Time	R_Large
☹	☹	☺

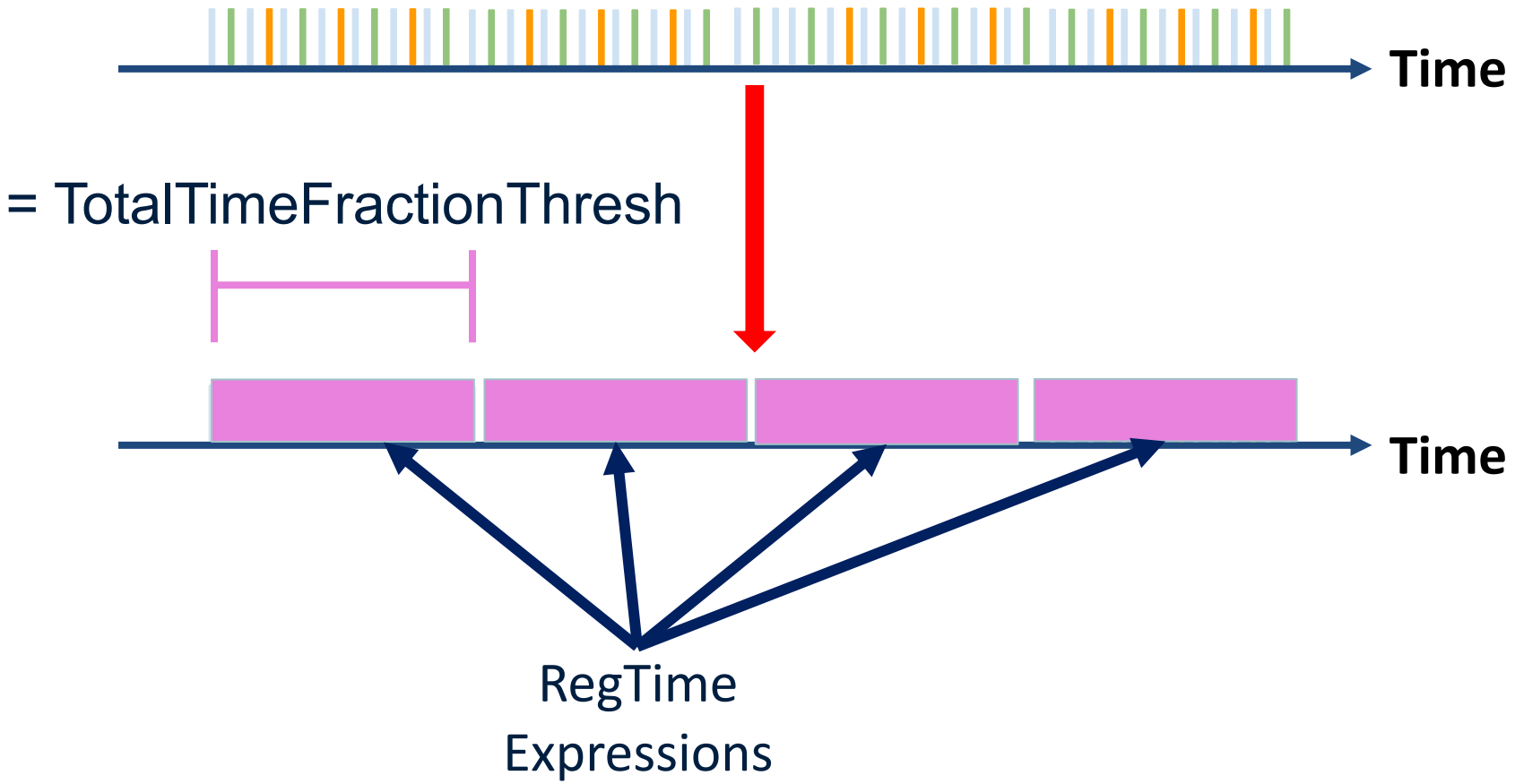
Leave Long Events Alone



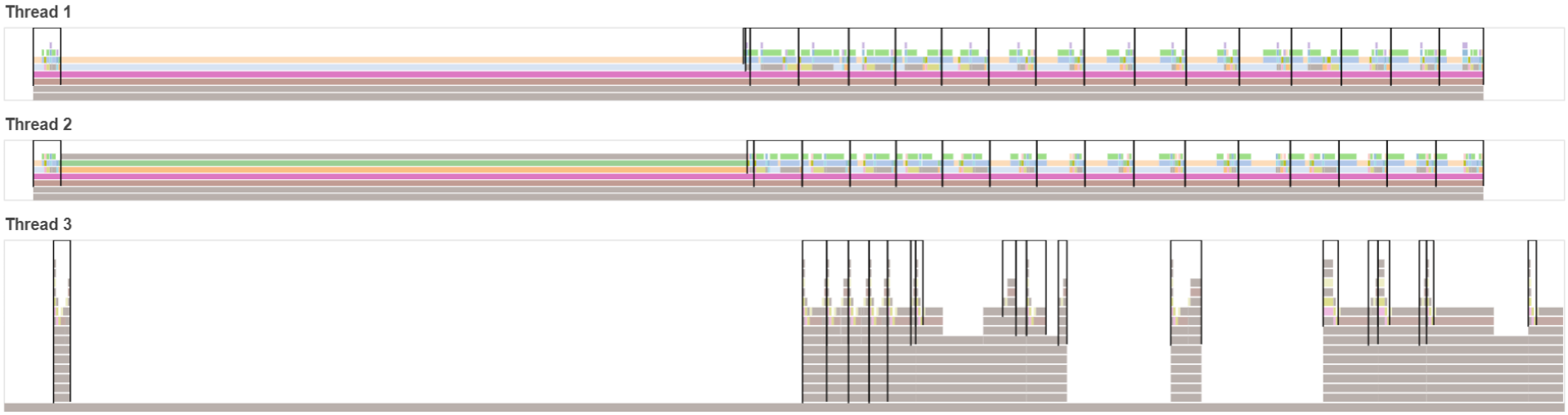
Leave Long Events Alone



Use Multiple RegTime Expressions

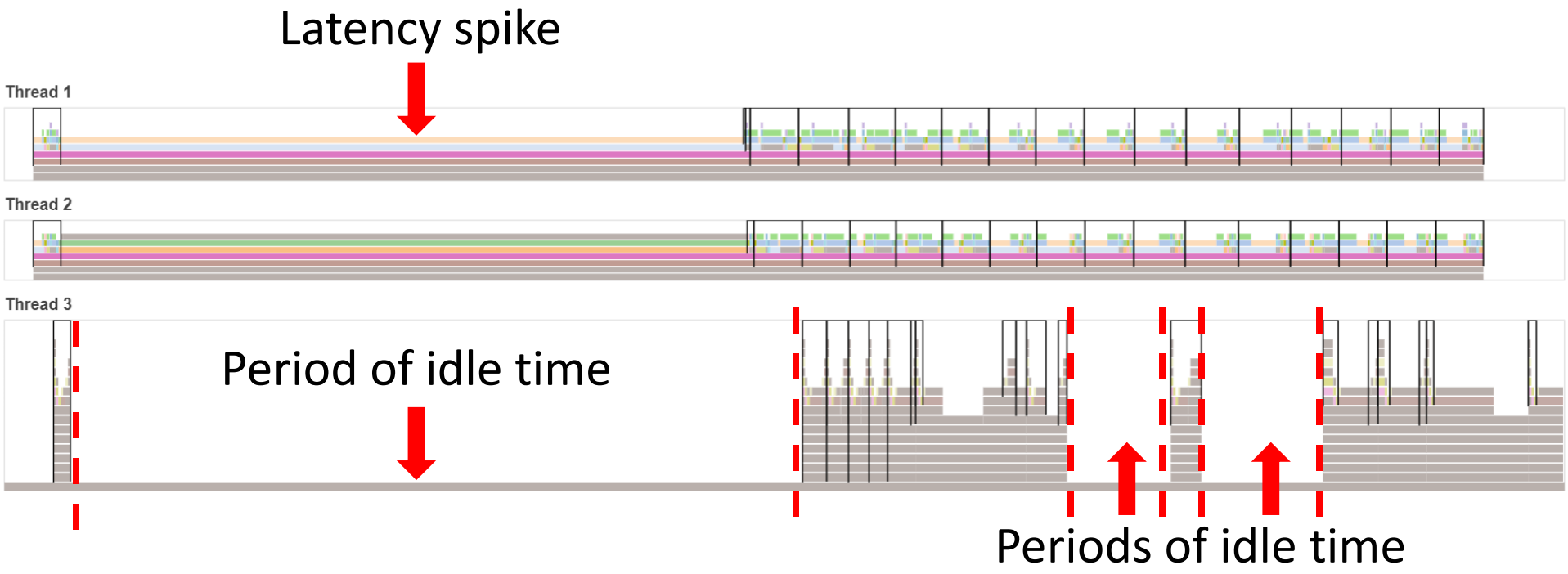


NonSequitur Version 2



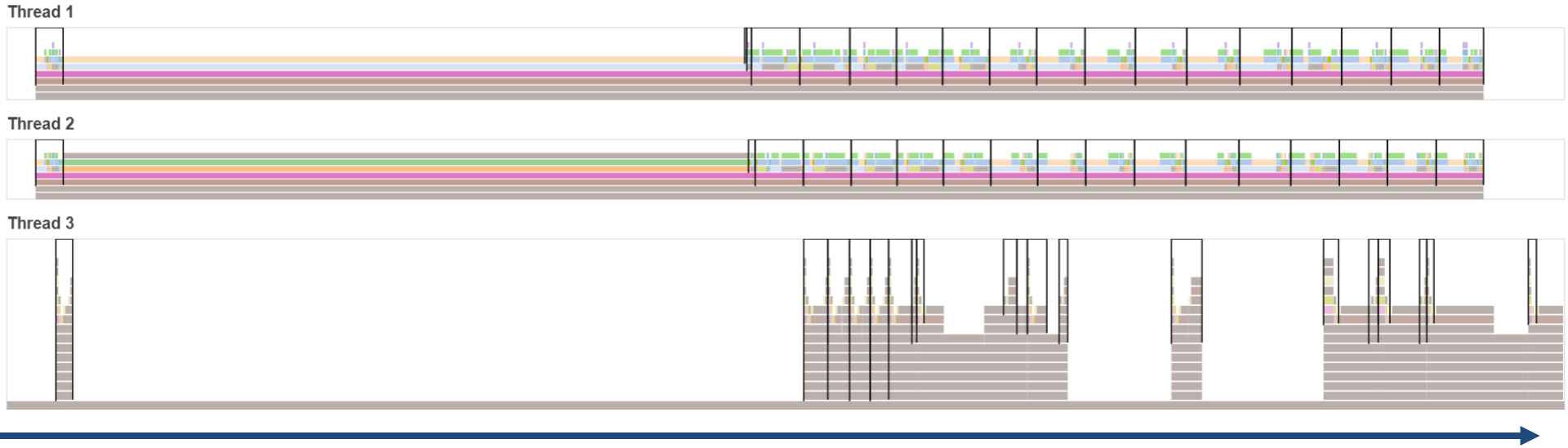
CallDurationThresh = 1% of a thread's execution time
CallGapThresh = 0.1% of a thread's execution time
TotalTimeFractionThresh = 3.2% of a thread's execution time

NonSequitur Version 2



CallDurationThresh = 1% of a thread's execution time
CallGapThresh = 0.1% of a thread's execution time
TotalTimeFractionThresh = 3.2% of a thread's execution time

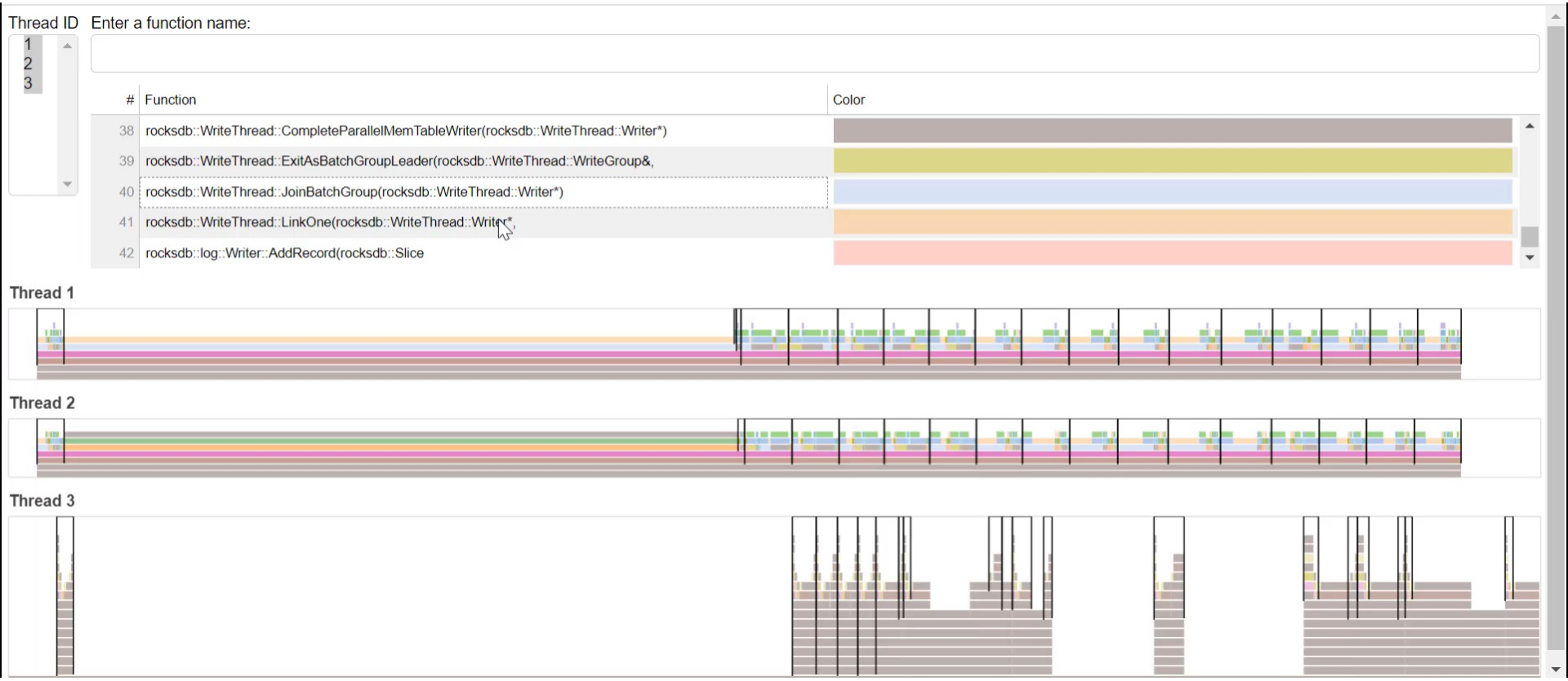
NonSequitur Design Drawbacks



X-axis not linear

Each callstack has a minimum horizontal pixel width

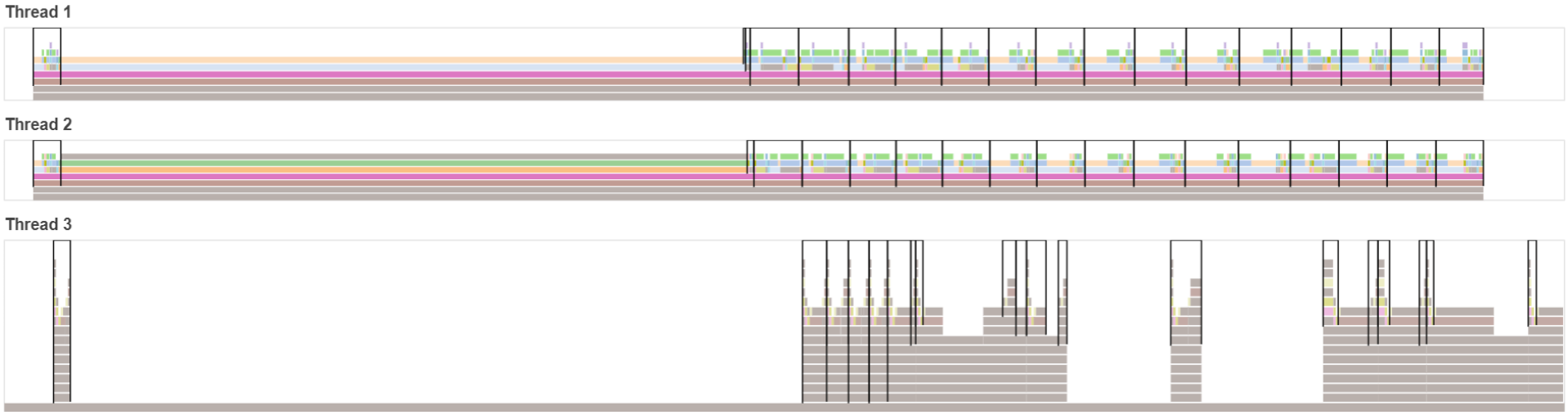
Navigating NonSequitur: Highlight Functions



Navigating NonSequitur: Linked Highlighting

Linked highlighting to help users find correlations in activities *across* threads.

NonSequitur Version 2



Tool Requirements		
R_Latency	R_Time	R_Large
☺	☺	☺

NonSequitur Eval: The Research Questions

RQ1: Does NonSequitur help analyze large execution traces to understand what threads are doing over time?

RQ2: Does NonSequitur help analyze large execution traces to find when unusually long function latencies occur?

RQ3: Does NonSequitur help analyze large execution traces to learn what one thread was doing during the time when another thread was blocked or delayed?

RQ4: For small traces that can be visualized with other tools, does NonSequitur lead to false conclusions in questions regarding thread activity over time as compared to other tools that do not drop information from the trace?

NonSequitur Eval: The Research Questions

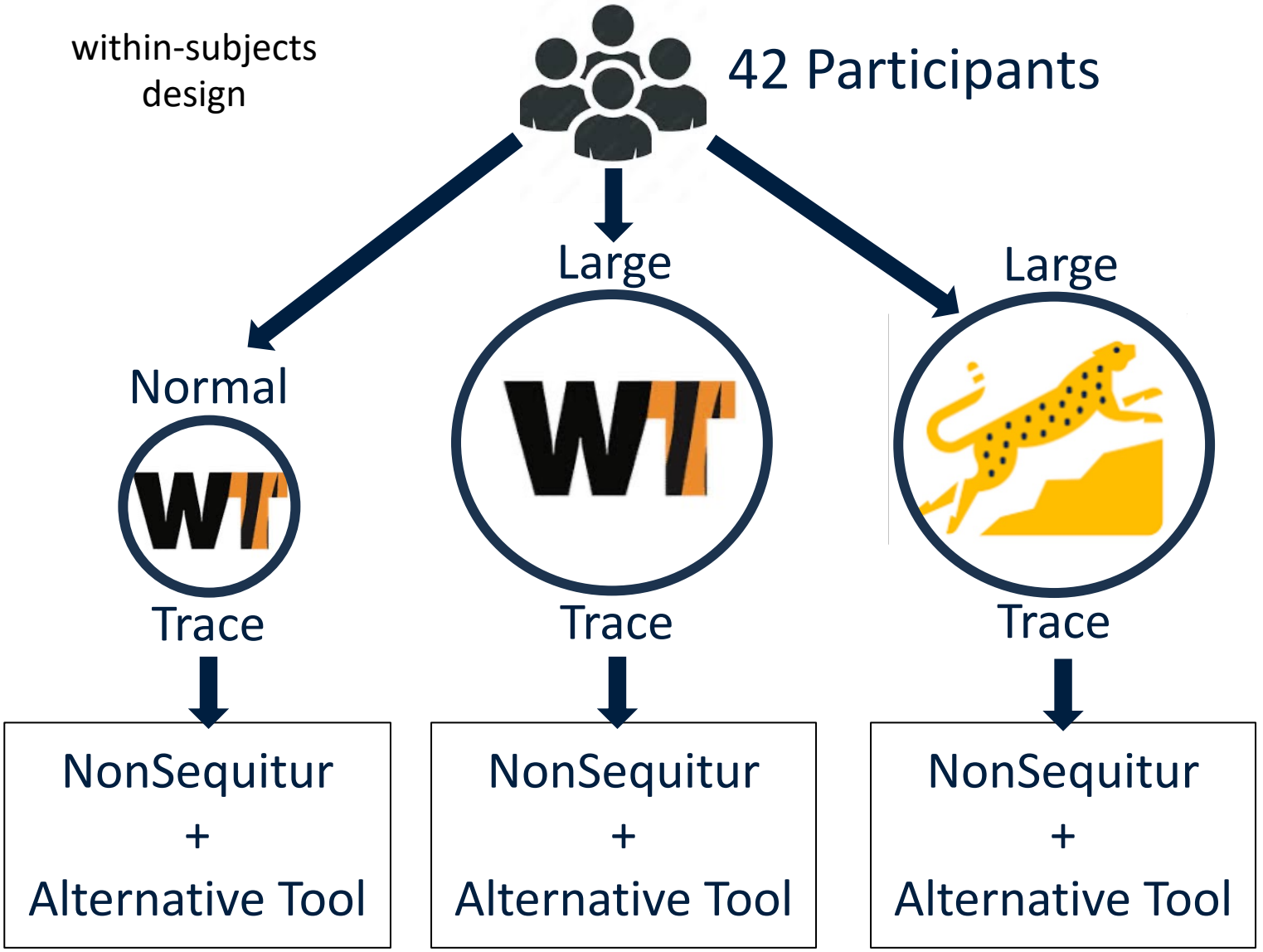
RQ1: Does it help explain what threads are doing over time?

RQ2: Does it help find unusually long function latencies?

RQ3: Does it help with correlating thread activities?

RQ4: Does it mislead on small traces?

NonSequitur Eval: The User Study



NonSequitur Eval: Trace Sizes

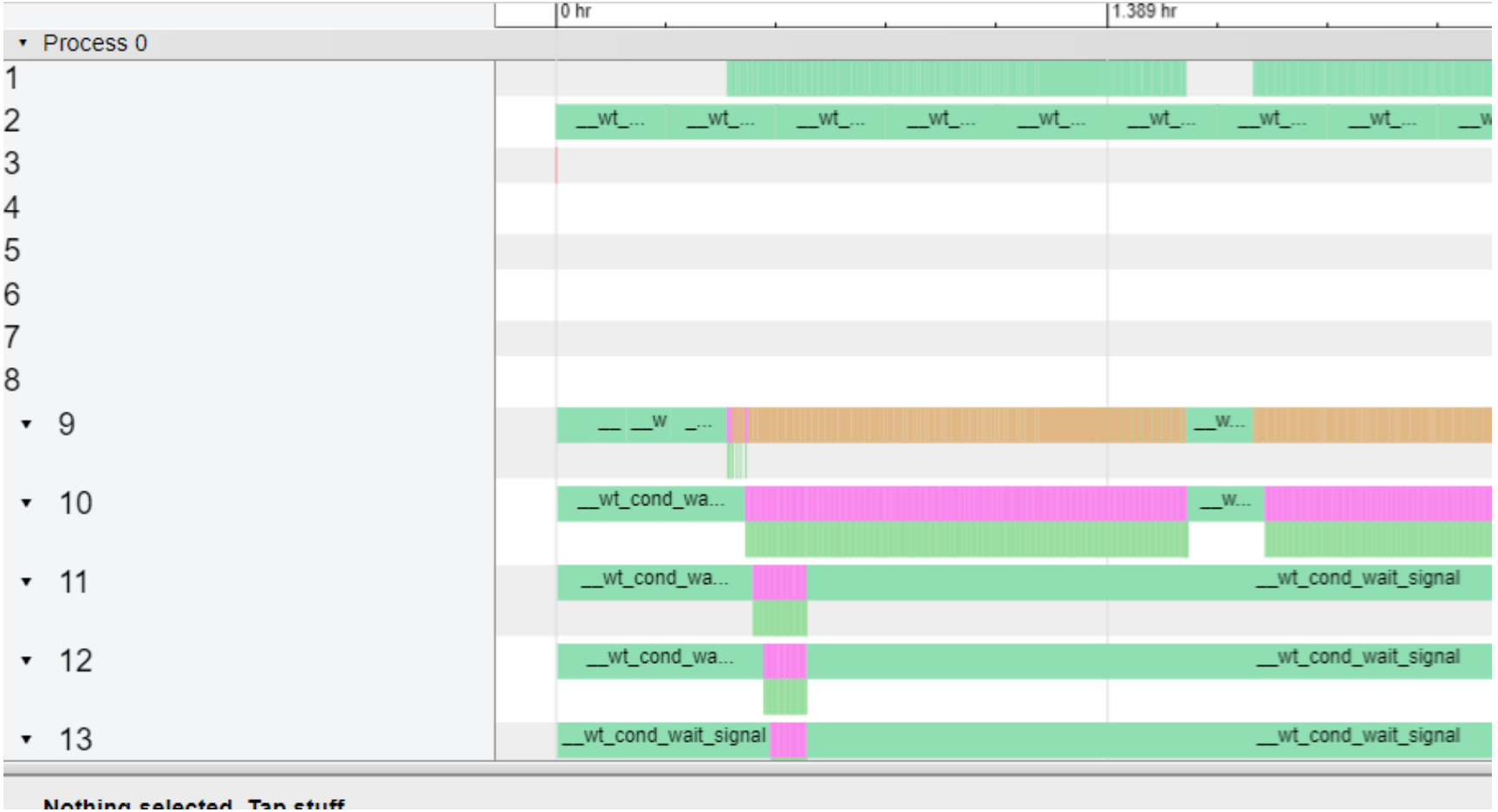
Normal WiredTiger Trace
0.8M function calls across 24 threads

Large WiredTiger Trace
50M function calls across 28 threads

Large RocksDB Trace
39M function calls across 32 threads

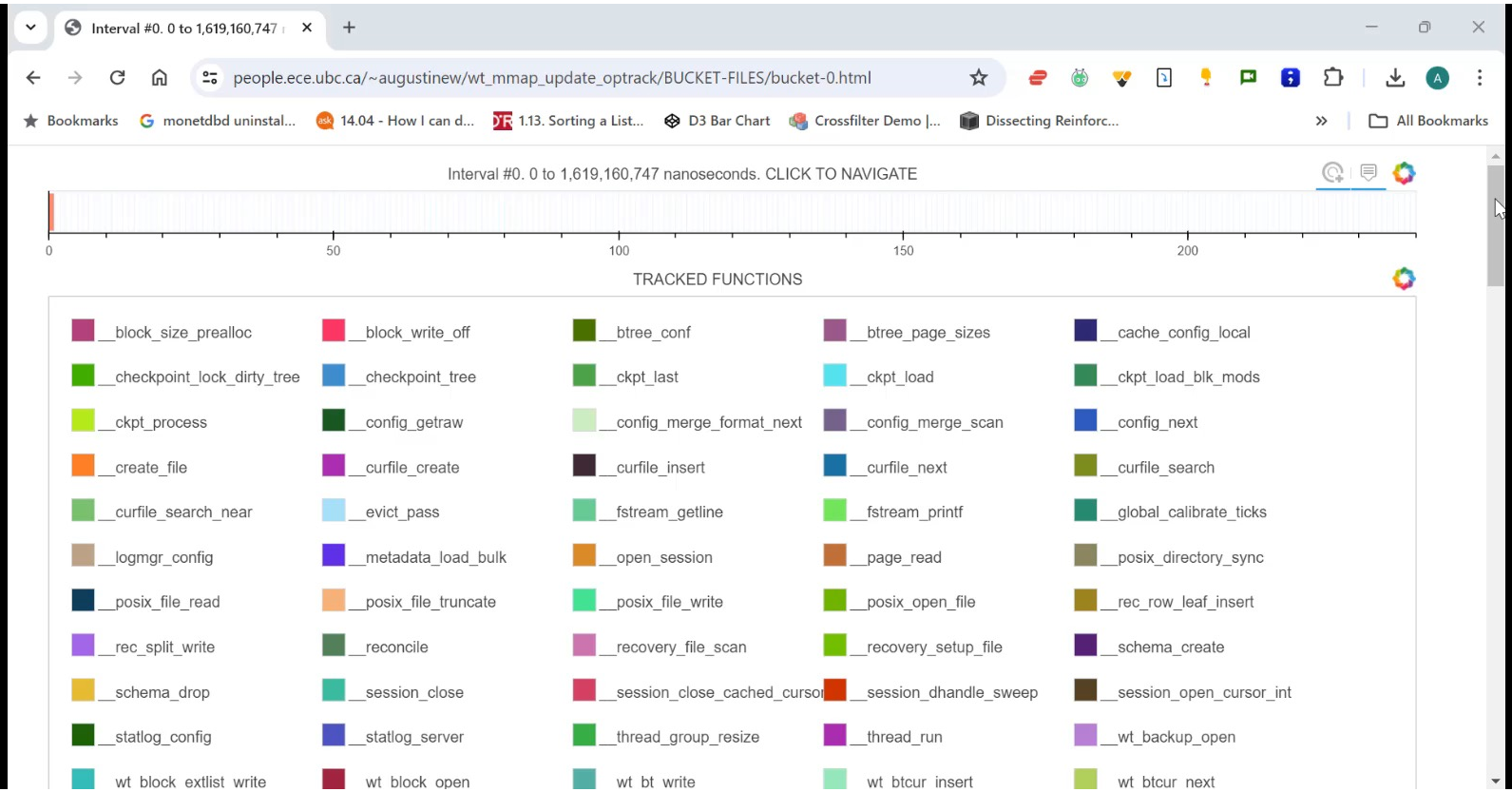
NonSequitur Eval: The Alternative Tools

Chrome TraceViewer for the normal WiredTiger Trace



NonSequitur Eval: The Alternative Tools

OpTrack for the large traces



NonSequitur Eval: The Tasks

Tasks	Description
T1	Understand what threads are doing over time
T2	Find when outlier events occur
T3	Understand what one thread was doing during the time when another thread was blocked or delayed.

NonSequitur Eval: Difference Scores

Difference Score > 0 

Difference Score < 0 

Difference Score = 0 

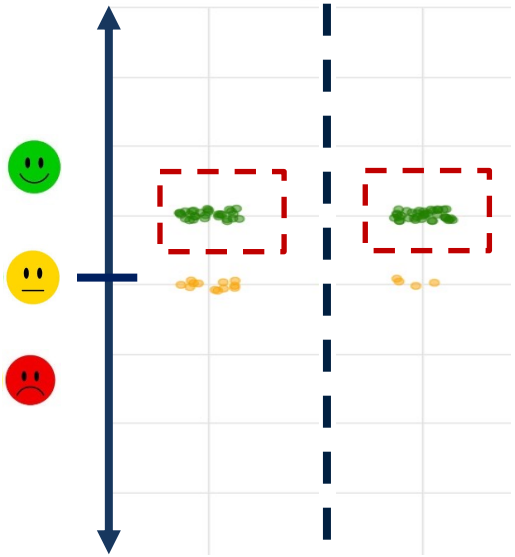
NonSequitur Eval: RQ1 Finding

Does it help explain what threads are doing over time?

Large WiredTiger Trace
50M function calls across
28 threads

Large RocksDB Trace
39M function calls across
32 threads

NonSequitur
vs
OpTrack



T1 Task

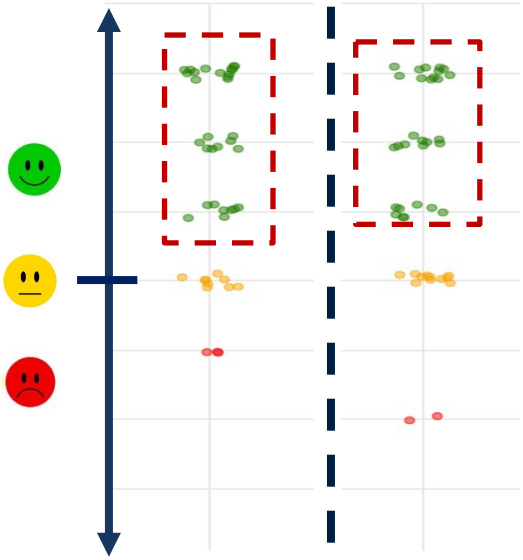
NonSequitur Eval: RQ2 Finding

Does it help find unusually long function latencies?

Large WiredTiger Trace
50M function calls across
28 threads

Large RocksDB Trace
39M function calls across
32 threads

NonSequitur
vs
OpTrack



T2 Task

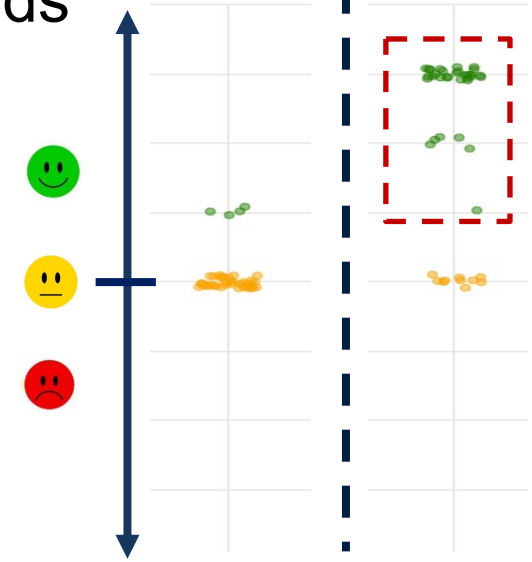
NonSequitur Eval: RQ3 Finding

Does it help with correlating thread activities?

Large WiredTiger Trace
50M function calls across
28 threads

Large RocksDB Trace
39M function calls across
32 threads

NonSequitur
vs
OpTrack



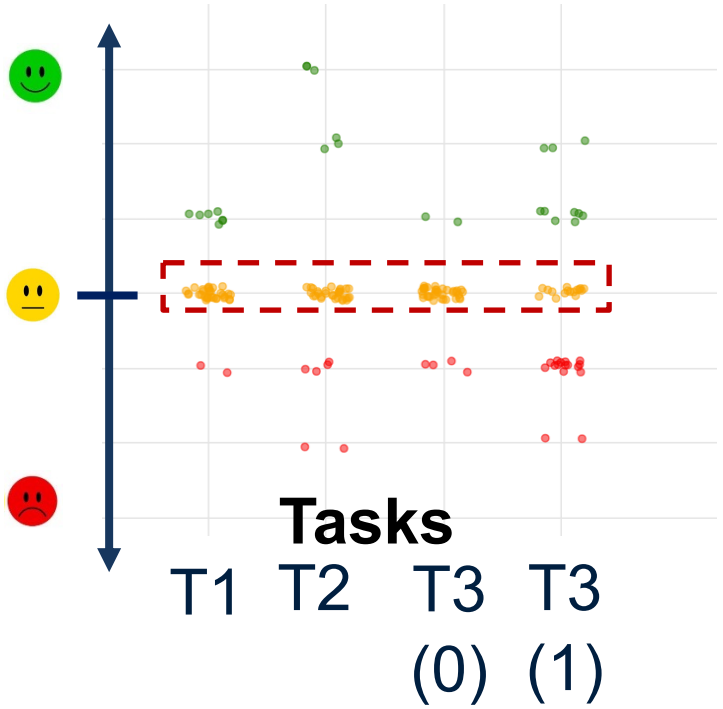
T3 Task

NonSequitur Eval: RQ4 Finding

Does it mislead on small traces?

Normal WiredTiger Trace
0.8M function calls across 24 threads

NonSequitur
vs
Chrome TraceViewer



Eval: Three case studies

Study 1: Reproduce slow LSM Performance in WiredTiger

Study 2: Slow Performance with the mmap Option in WiredTiger

Study 3: RocksDB memtable Concurency

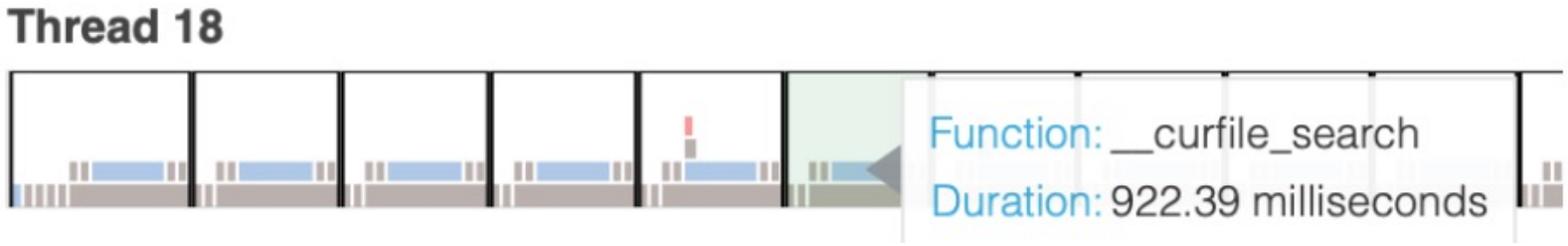
Eval: Three case studies

Study 1: Reproduce slow LSM Performance in WiredTiger

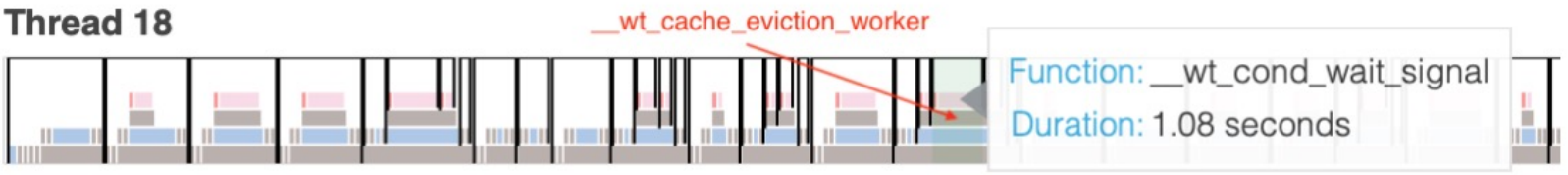
Study 2: Slow Performance with the mmap Option in WiredTiger

Study 3: RocksDB memtable Concurency

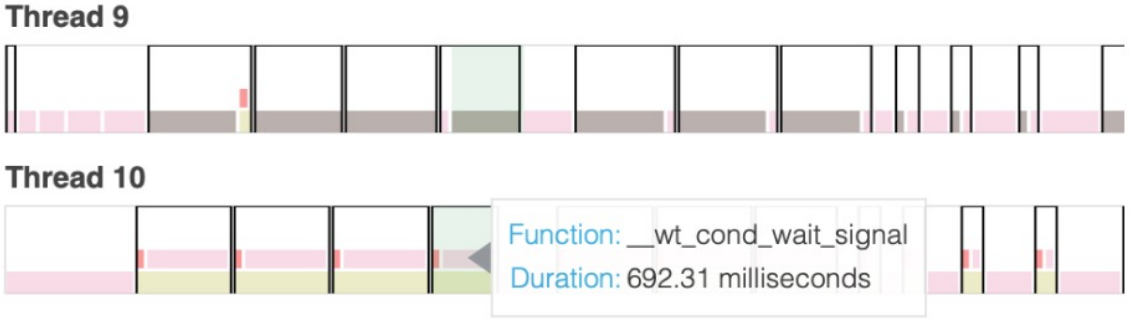
Good run:



Bad run:



Background threads during the bad run:



Blocking eviction is the root cause in the background threads

Eval: compression ratio

High compression ratios with RegTime!

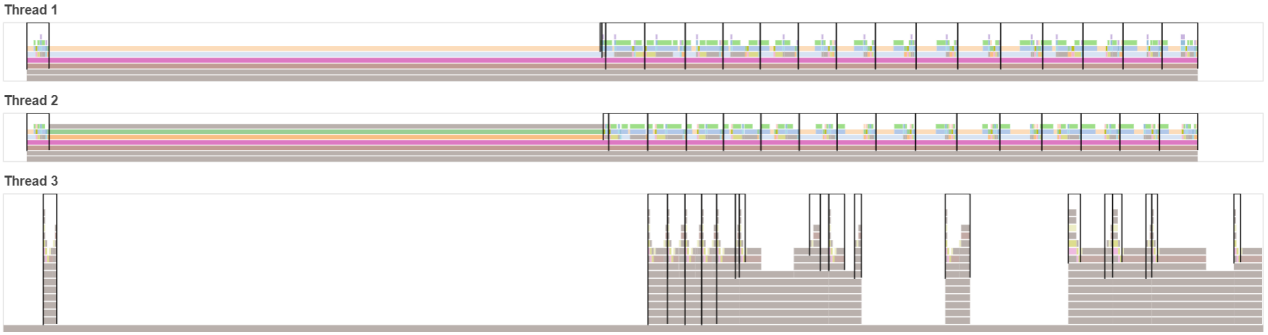
System	Thread	# unique events	n_{before}	n_{after}	Compression Ratio
RocksDB	0	74	2,466,295	323	7,635
RocksDB	1	316	2,762,774	2,485	1,111
WiredTiger	0	115	7,784,936	546	14,258
WiredTiger	1	81	6,935,791	209	33,185
Chrome	0	34	274,361	96	2,857
Chrome	1	8	46,464	56	829

Contributions

*Making Sense of Multi-threaded Application Performance with **NonSequitur***

The lossy compression performed by RegTime does not negatively impact a developer's ability to understand what threads are doing over time.

NonSequitur can help developers perform performance analysis tasks on large execution traces more accurately.



https://github.com/auggywonger/nonsequitur_vis/