COZ : Finding Code that Counts with Causal Profiling

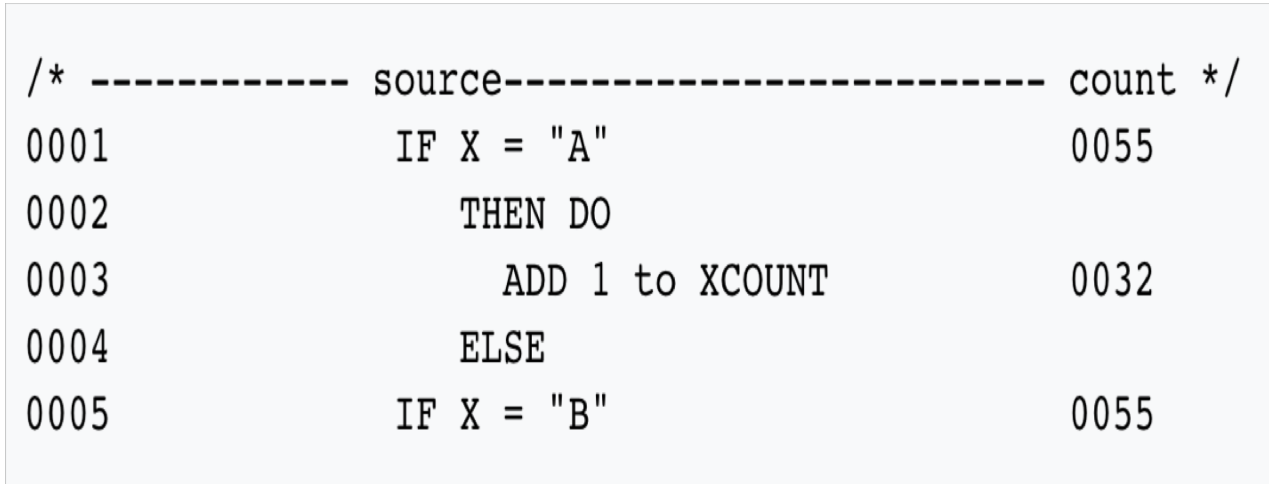
Anuja Golechha

Agenda

- Profiling
- Issues with current profilers
- Causal profiling
- COZ – Overview and Implementation
- COZ Evaluation
- Comparison with Pivot Tracing

Profiling

- Profiler Types
  - Instrumentation
  - Sampling



```
/* ------------ source------------------------- count */
0001                 IF X = "A"                    0055
0002                    THEN DO
0003                      ADD 1 to XCOUNT          0032
0004                    ELSE
0005                 IF X = "B"                    0055
```

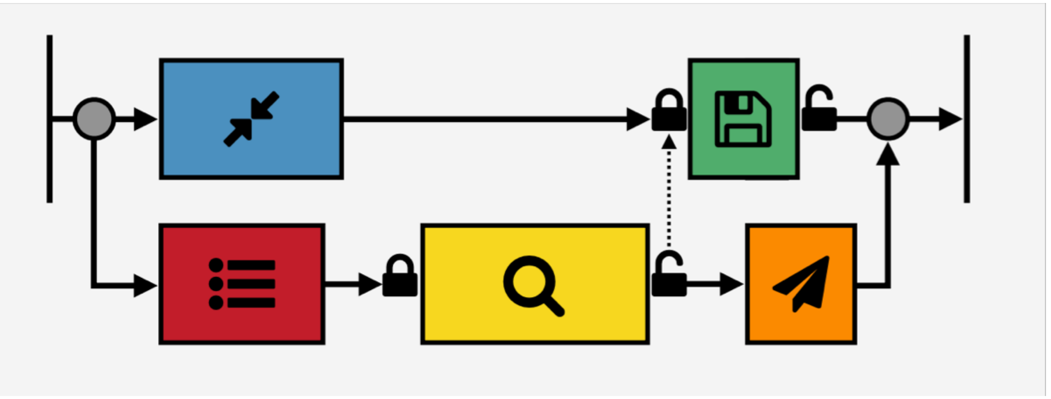
Issues with current profilers

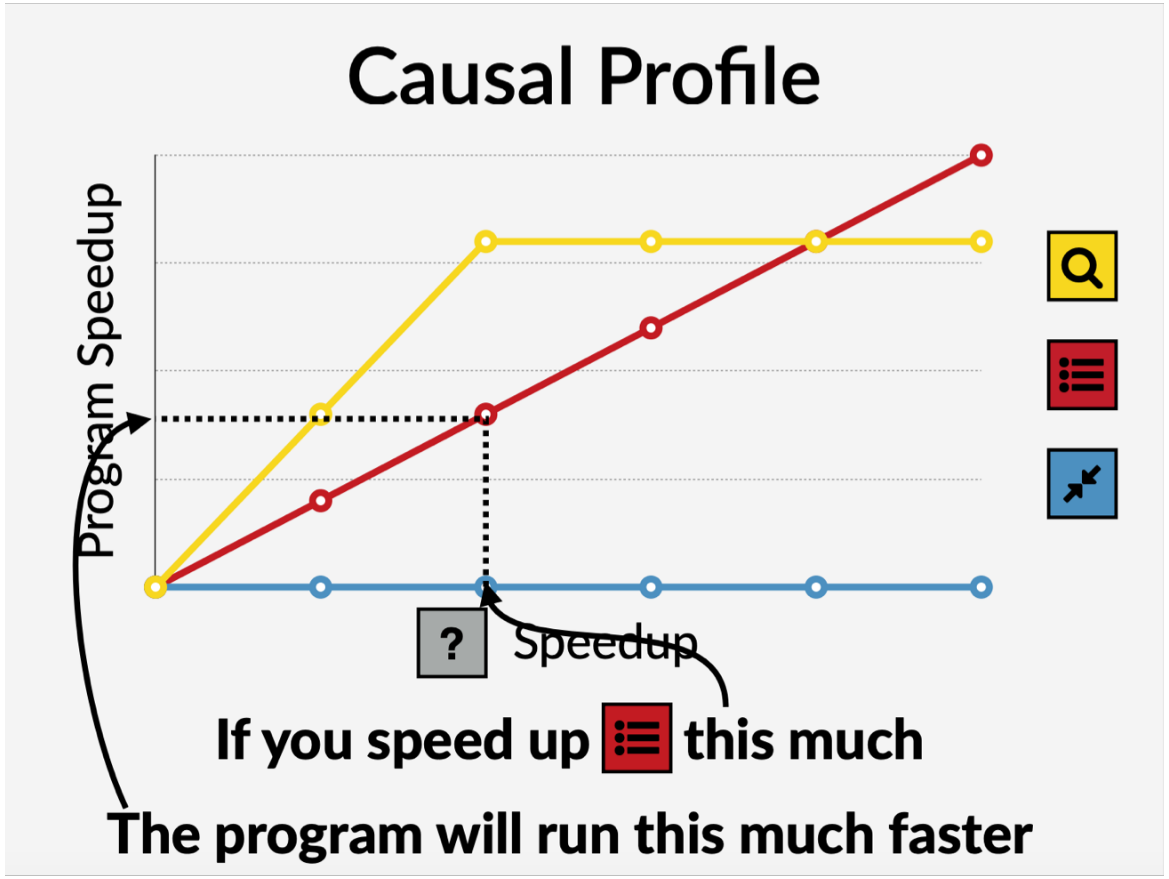
- Only report how long code runs for

- Code that runs for a long time might not be the best choice for optimization
  - Example – loading animation during file download

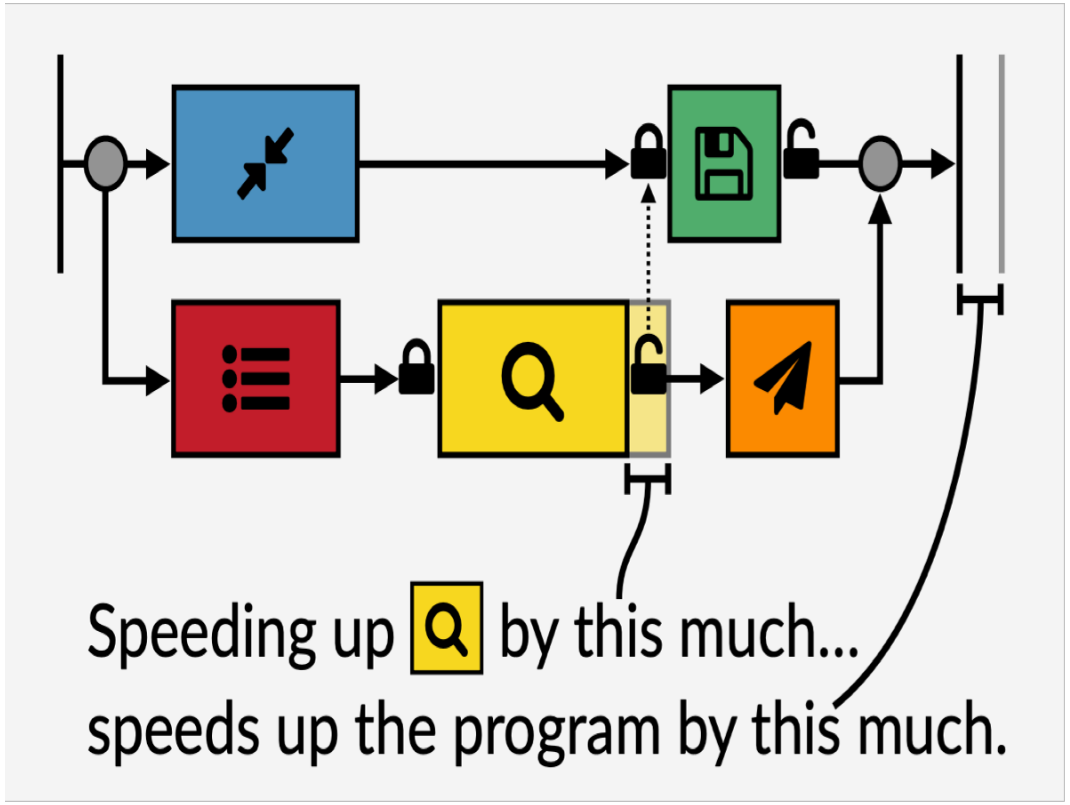
- Do not report potential impact of optimization

Example Application



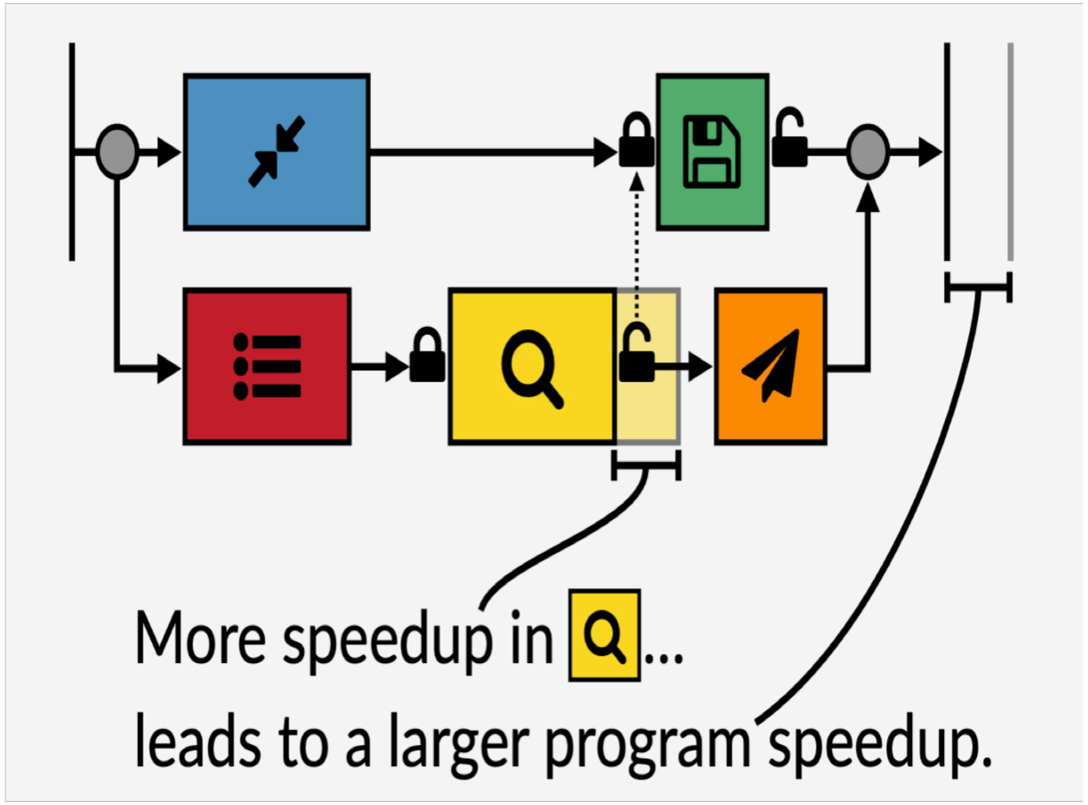

Causal Profile
Program Speedup
?
Speedup
If you speed up this much
The program will run this much faster

Example Application – Speed up Search



Speeding up 🔍 by this much...

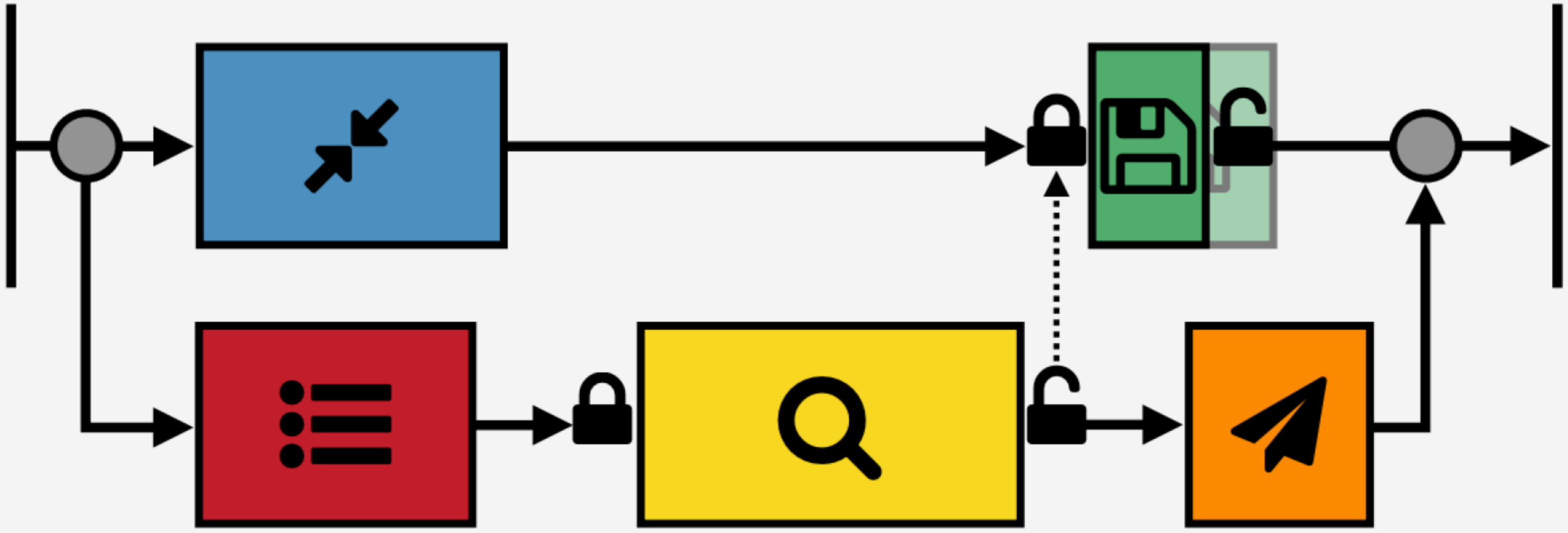
speeds up the program by this much.



More speedup in 🔍...

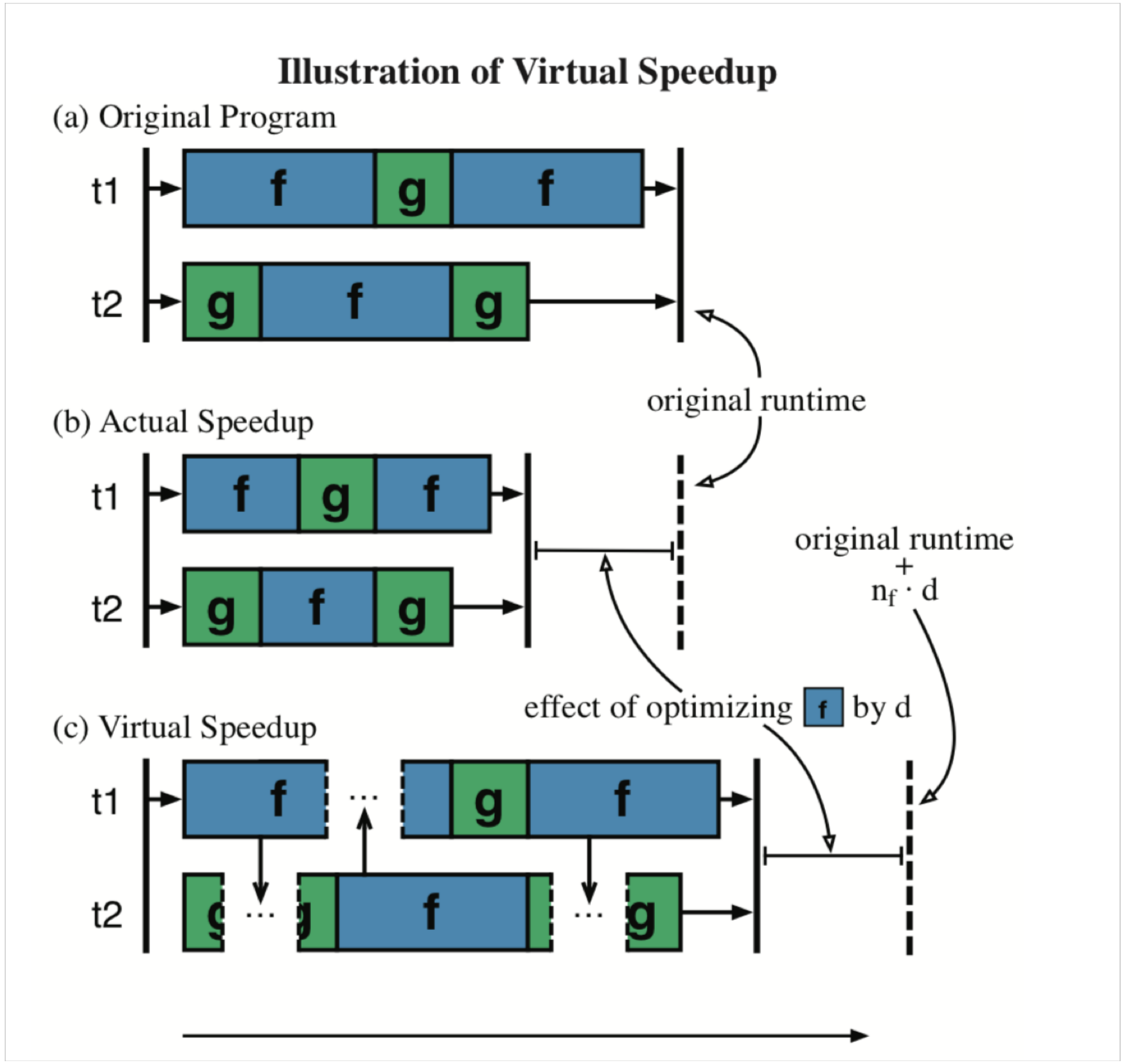
leads to a larger program speedup.

Example Application – Speed up Save

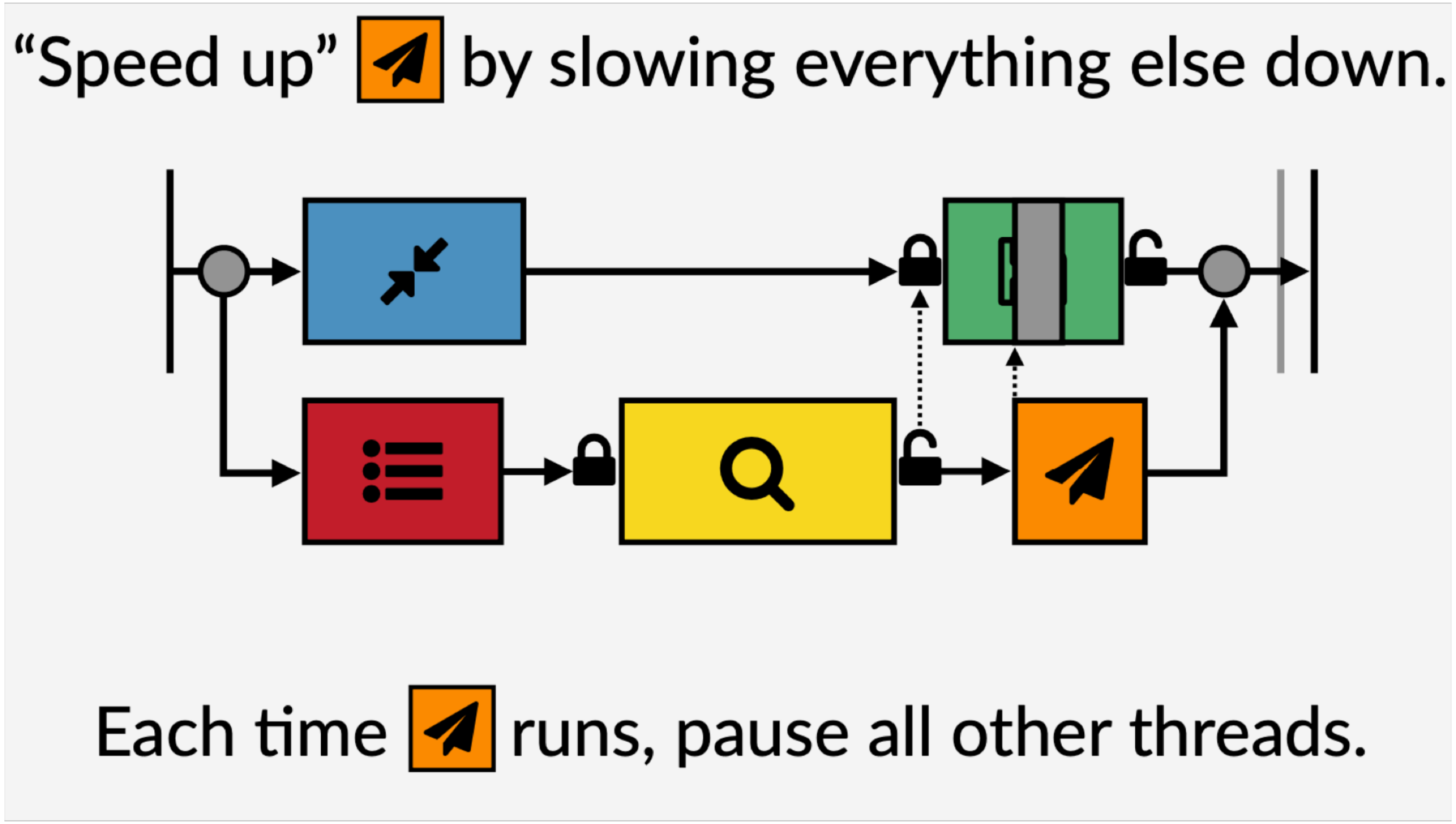


No program speedup

Causal Profiling – Virtual Speedup



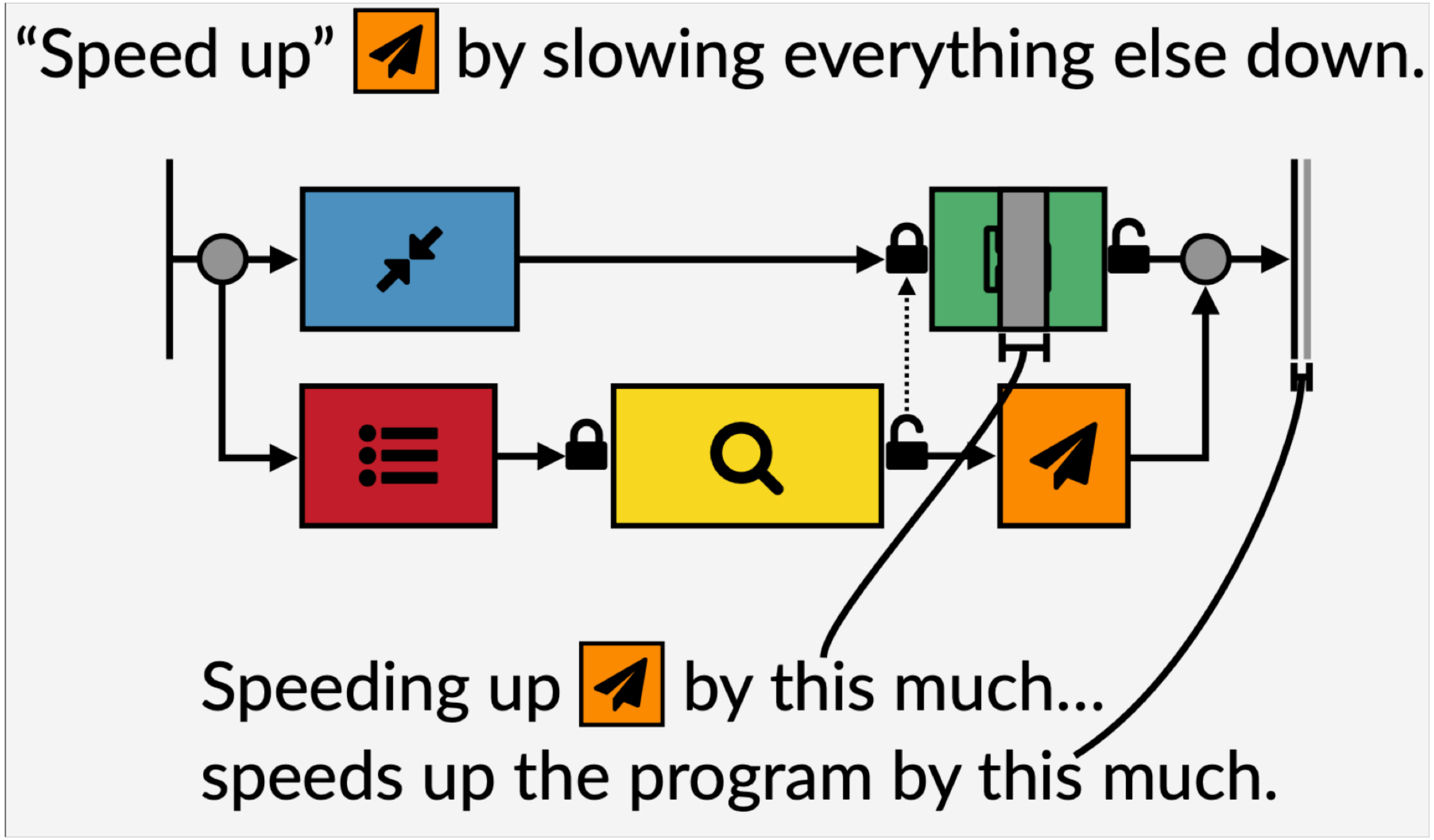
Example Application – Virtual Speedup Send



“Speed up” by slowing everything else down.

Each time runs, pause all other threads.

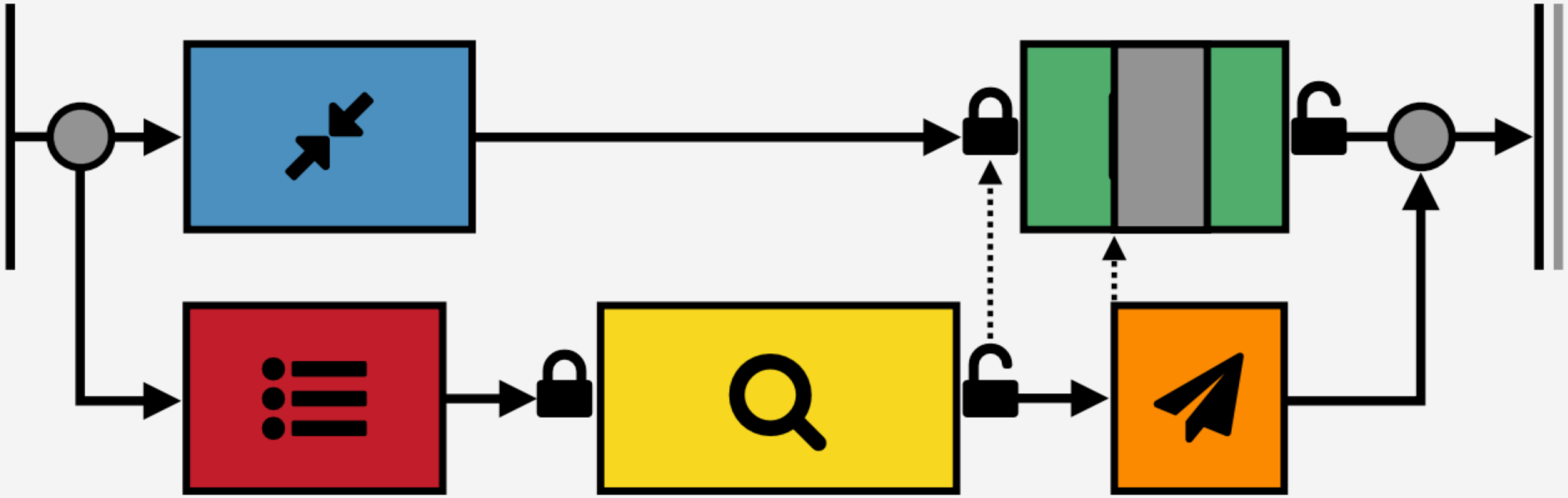
Example Application – Virtual Speedup Send



"Speed up" ✈ by slowing everything else down.

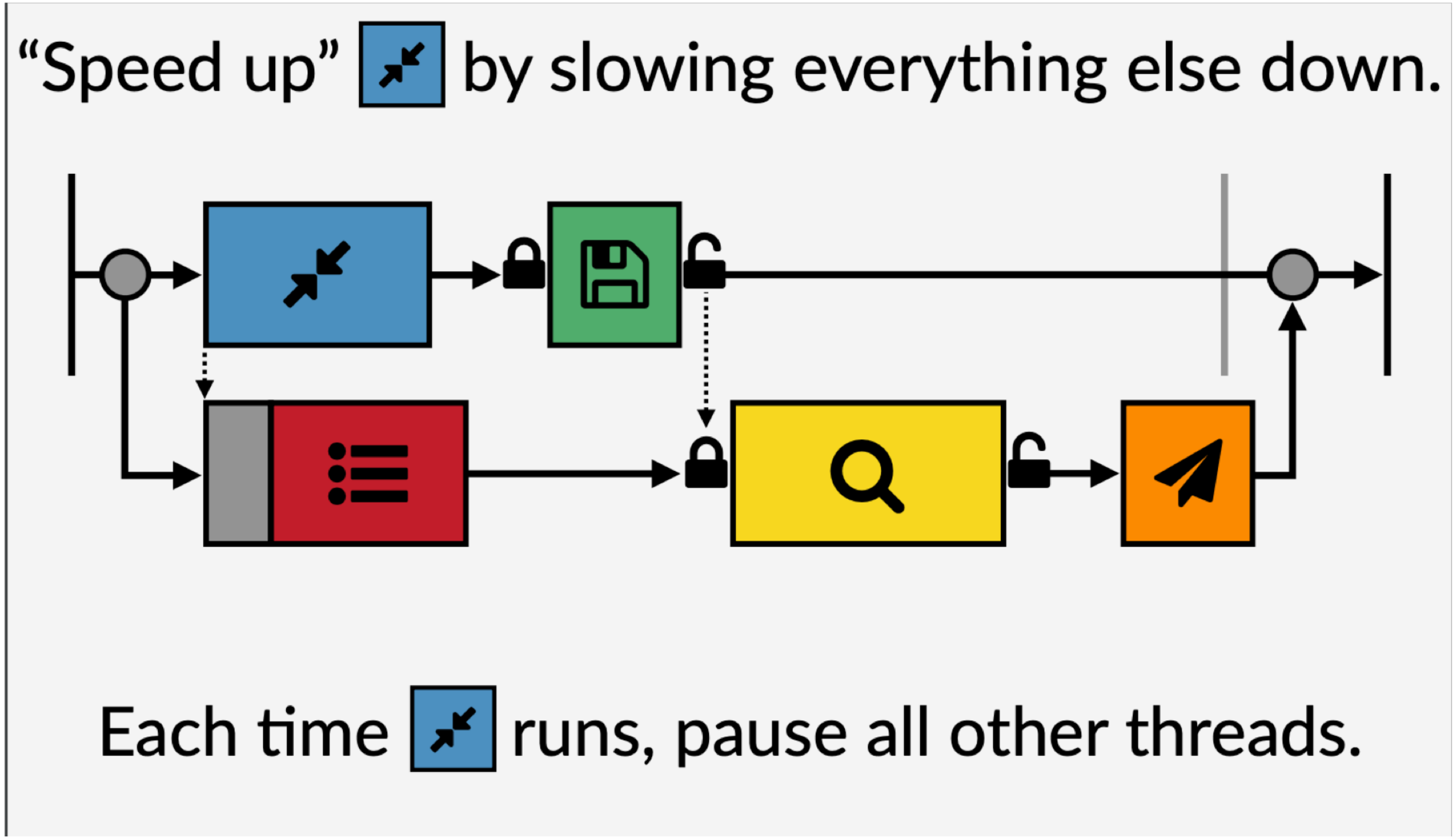
Speeding up ✈ by this much...
speeds up the program by this much.

Example Application – Virtual Speedup Send



A larger speedup has no additional effect

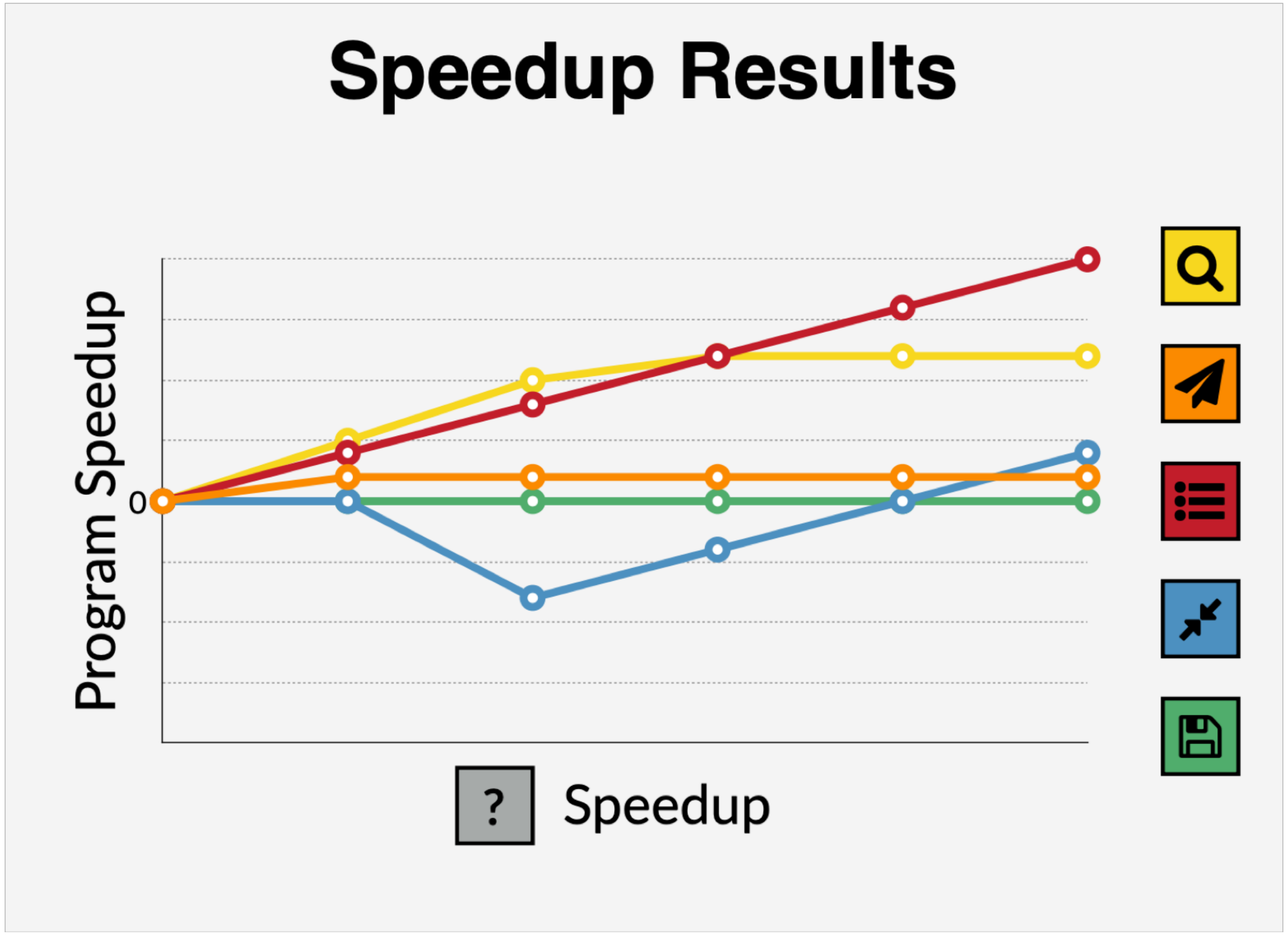
Example Application – Virtual Speedup Compress



“Speed up” by slowing everything else down.

Each time runs, pause all other threads.

Example Application



Causal Profiling

- Performance experiments
  - Associated with a line of code and a percent speedup value
- Progress Points – View effect of optimization on both throughput and latency
  - Progress point – a line of code indicating the end of a unit of work
  - Throughput – measured by rate of visits to each progress point
  - Latency – use two progress points
    - Difference between counts at start and end points gives how many requests are currently in progress
    - Rate of visits to the start point gives the arrival rate
    - Little’s Law – average latency = number of requests in progress / arrival rate

COZ

- Prototype for Linux

- Implementation Details
  - Dedicated profiler thread
  - Flexibility – User can specify a scope to control which lines are considered for potential optimizations

COZ - Causal Profiling Overview

- Profiler Startup
  - Map instructions to source code using the program’s debug information
  - Create profiler thread
- Performance Experiment Initialization
  - Randomly choose a line and a percent speedup
- Apply Virtual Speedup
  - Pause other threads if sample belongs to selected line of code
- Experiment end
  - Pre-determined time
  - Cooloff period

COZ Virtual Speedup Implementation

Uses Sampling

$$s \approx \frac{n \cdot \bar{t}}{P}$$

s – number of samples of selected line

P – sampling period

n – number of times selected line is executed

d - delay

$$\bar{t}_e = \frac{(n-s) \cdot \bar{t} + s \cdot (\bar{t} - d)}{n}.$$

$$\Delta\bar{t} = 1 - \frac{\bar{t}_e}{\bar{t}} = \frac{d}{P}.$$

COZ Virtual Speedup Implementation

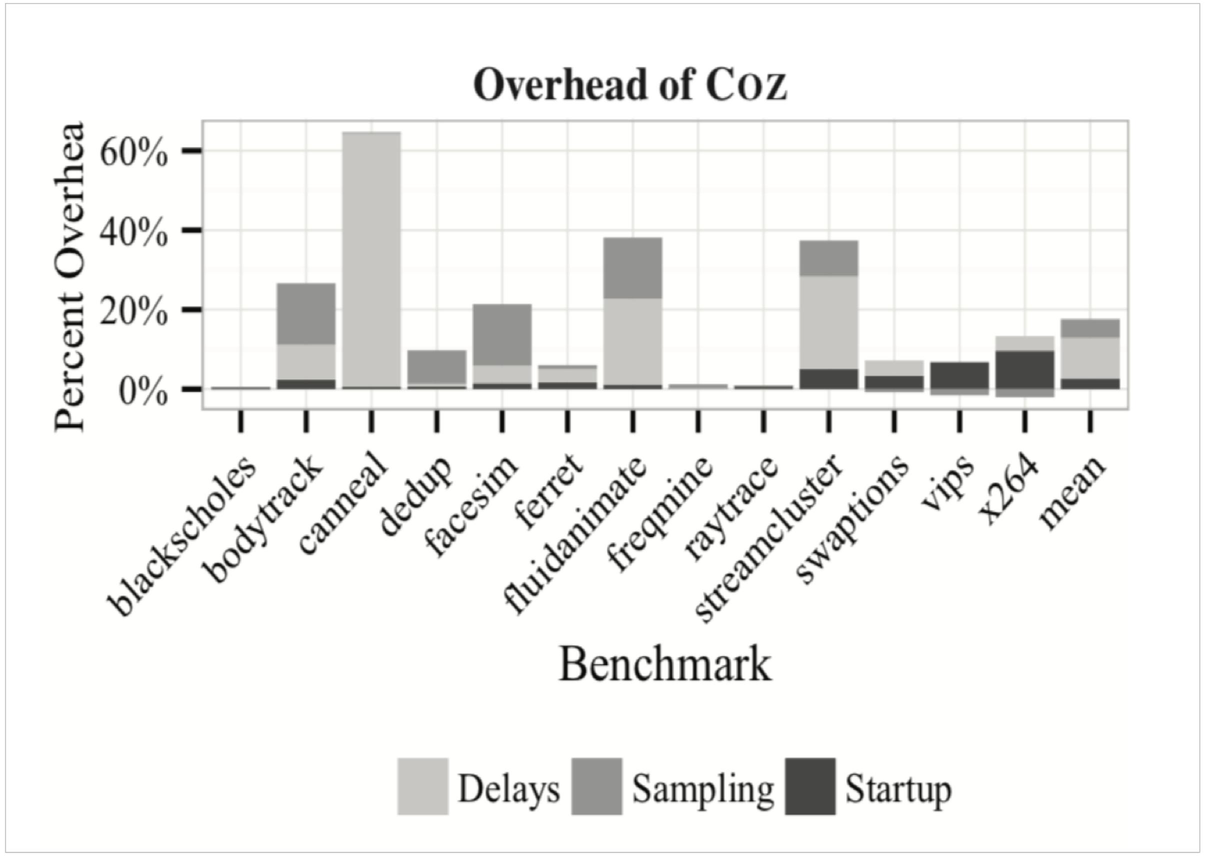
- Pauses other threads using counters
- Global counter – the number of times each thread should have paused
- Local counter – the number of times a thread has already paused
- Thread must pause and increment local counter if local < global
- Suspended threads – Thread must execute all required delays before a potential blocking operation or waking up another thread

COZ Evaluation – Types of Optimizations

- Identifying bottleneck
  - Dedup – hash bucket traversal (8.9 % actual, 9% predicted)
  - SQLite – overhead of indirect function calls (25 %)
- Reallocation of resources based on COZ's predicted impact
  - Ferret – reallocation of threads across stages (21.2 % actual, 21.4% predicted)
- Points of Contention – downward sloping causal profile
  - Fluidanimate – replaced custom barrier by default (37 %)
  - Memcached – removed lock while updating reference counts (9 %)

COZ Evaluation – Overhead

- Average – 17.6 % overhead
- Possible optimizations to improve overhead –
  - Collect and process debug information lazily to reduce startup overhead
  - Amortize sampling cost by sampling globally instead of per-thread
  - Reduce delay overhead by allowing normal execution between experiments for some time



Comparison with Pivot Tracing

- Type
  - Sampling vs Dynamic Instrumentation
- Causality
  - COZ – Effect of optimization on total runtime / throughput / latency
  - PT – Correlation between events (abstraction of happened-before joins)
- PT – For distributed systems
- COZ – Focuses on CPU usage

References

- https://www.sigops.org/s/conferences/sosp/2015/current/2015-Monterey/printable/090-curtsinger.pdf
- https://www.usenix.org/node/196222
- https://github.com/plasma-umass/coz
- http://sigops.org/s/conferences/sosp/2015/current/2015-Monterey/printable/122-mace.pdf
- http://pivottracing.io/
- https://en.wikipedia.org/wiki/Profiling_(computer_programming)

Thank You