This presentation uses the following non-standard fonts:

Lato (http://www.latofonts.com/lato-free-fonts/)

FontAwesome (https://fortawesome.github.io/Font-Awesome/)

You will need to download and install the TTF versions of these fonts for the slides to work correctly. A PDF version with embedded fonts is available at http://www.cs.grinnell.edu/~curtsinger/

Coz: Finding Code that Counts with Causal Profiling

Charlie Curtsinger

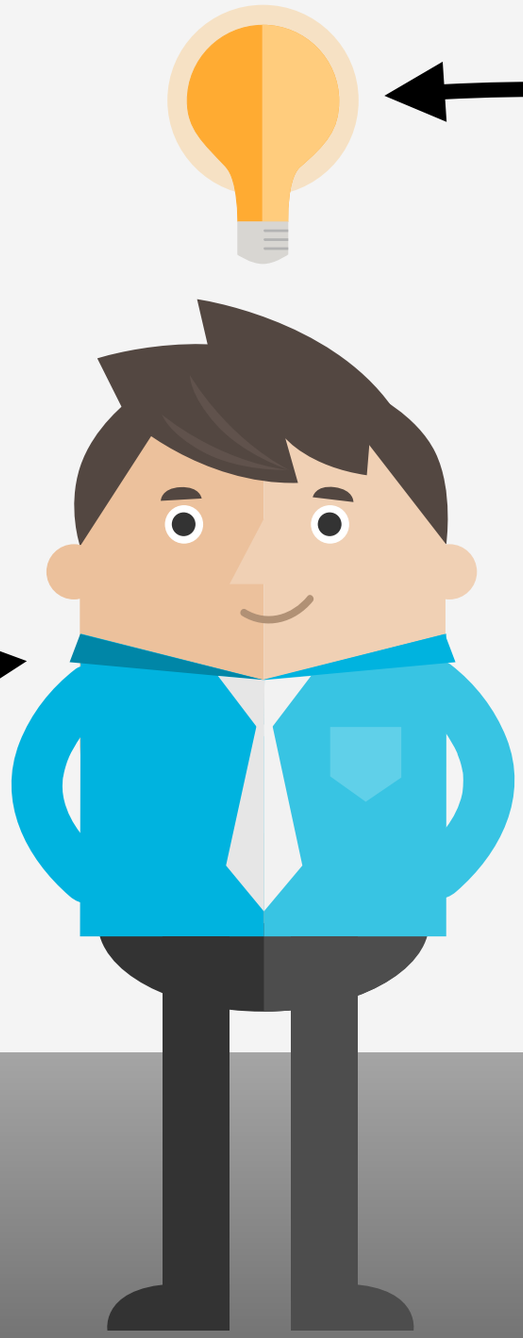
Emery D. Berger

GRINNELL COLLEGE

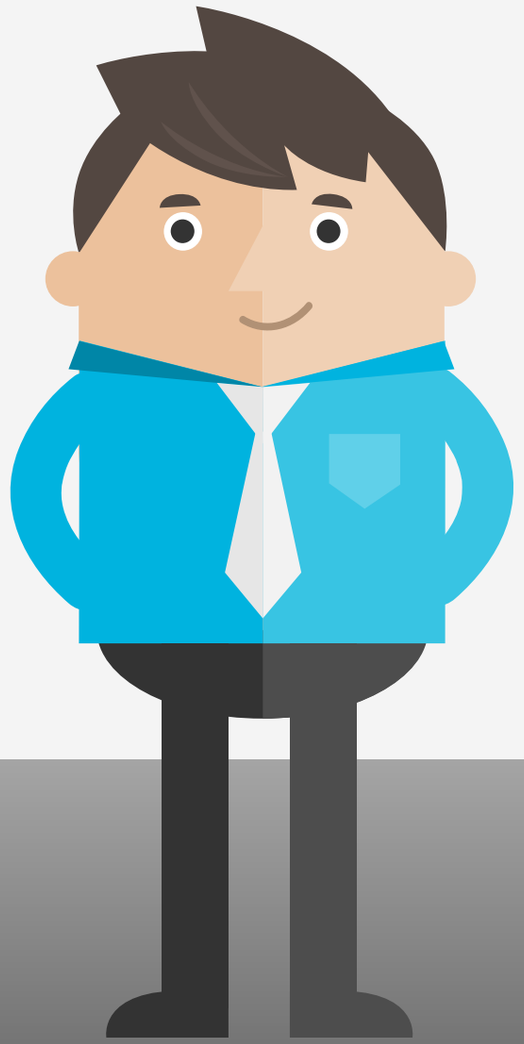
UMassAmherst

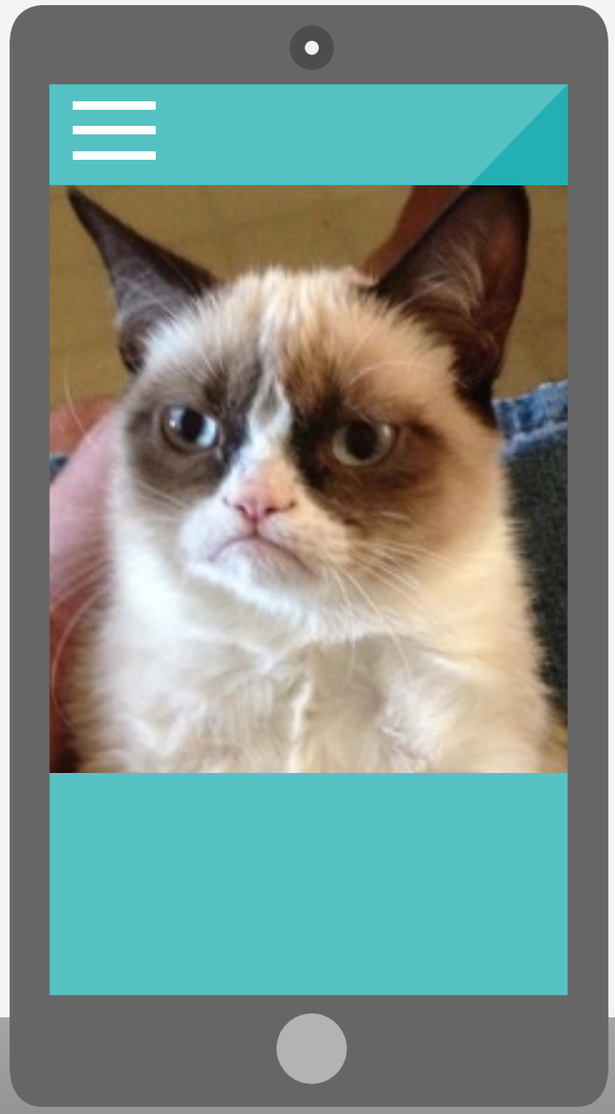
A few months ago,
in a valley not so far away...

This is Bob.

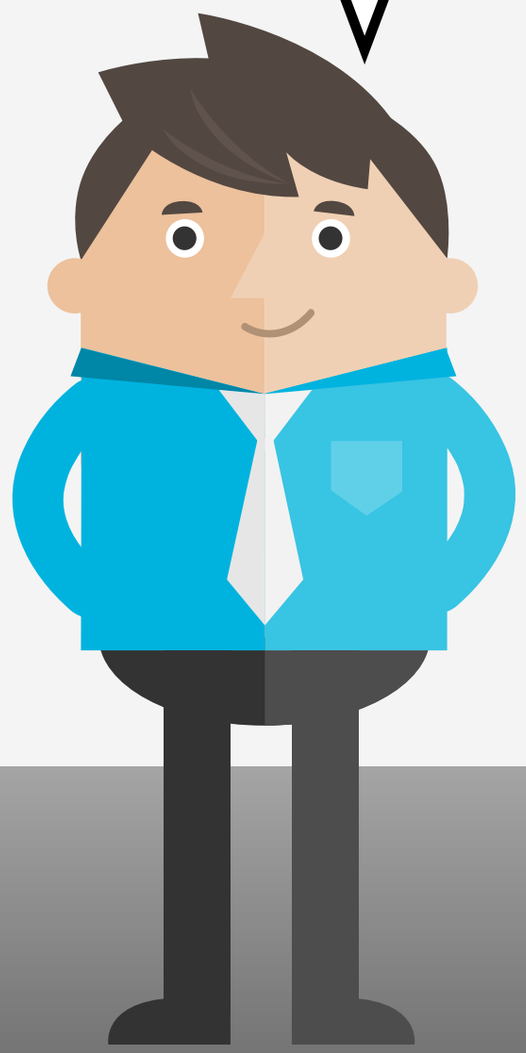


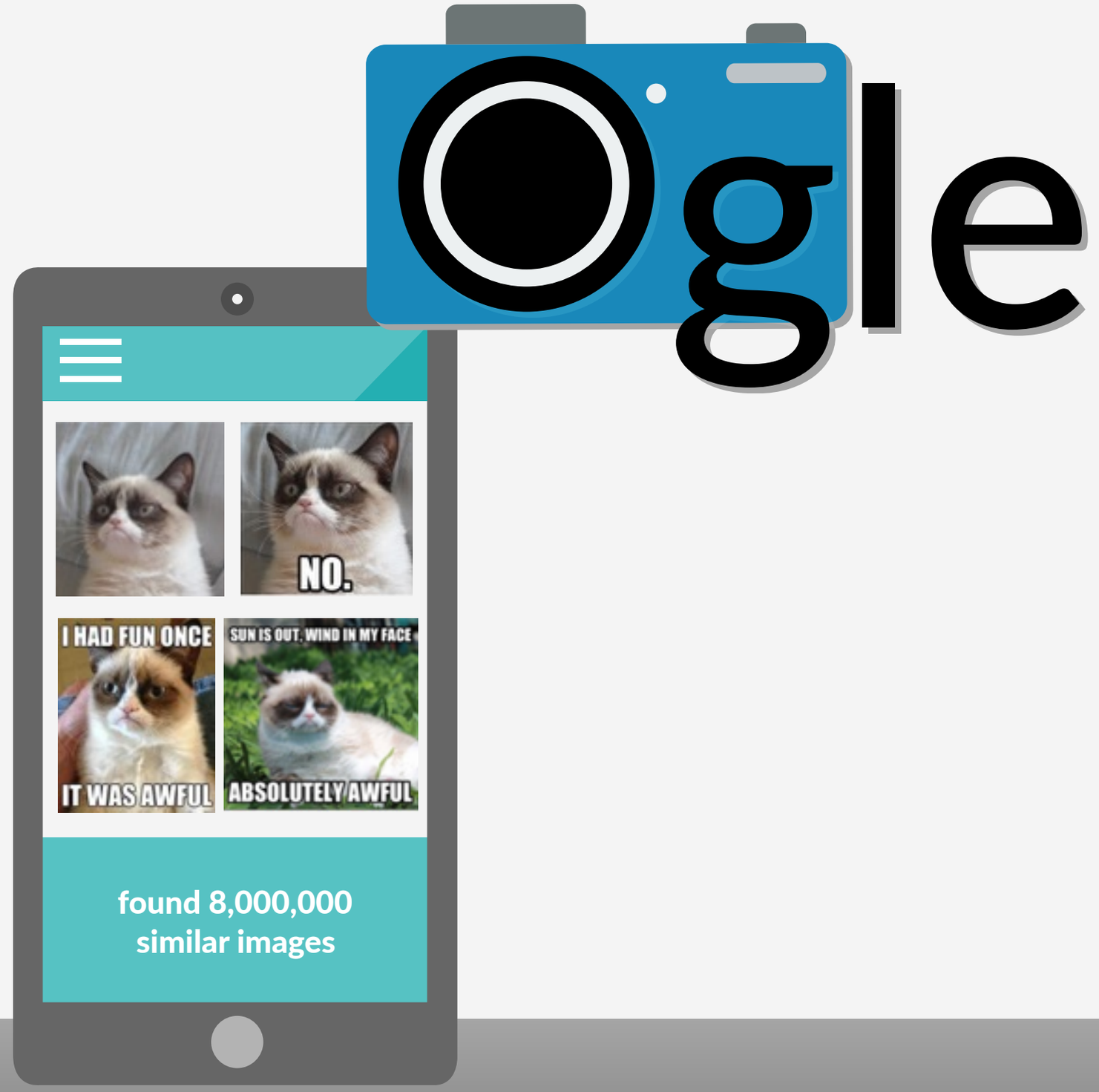
Bob has an idea for a startup.





It’s going to totally disrupt image search.



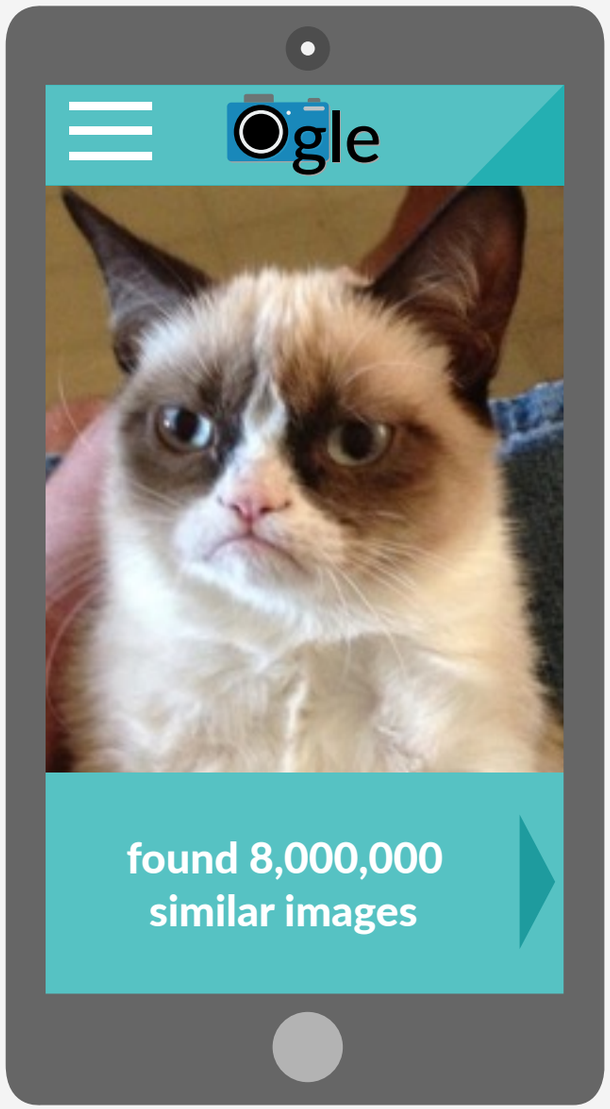


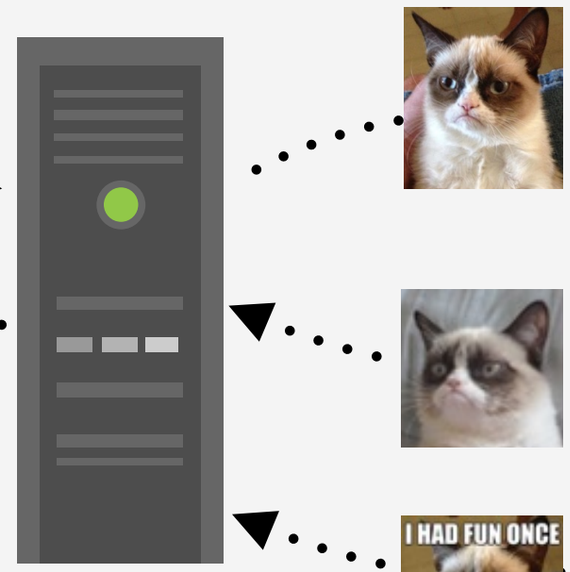
The Prototype Ogle

Take a picture

Send it to Ogle

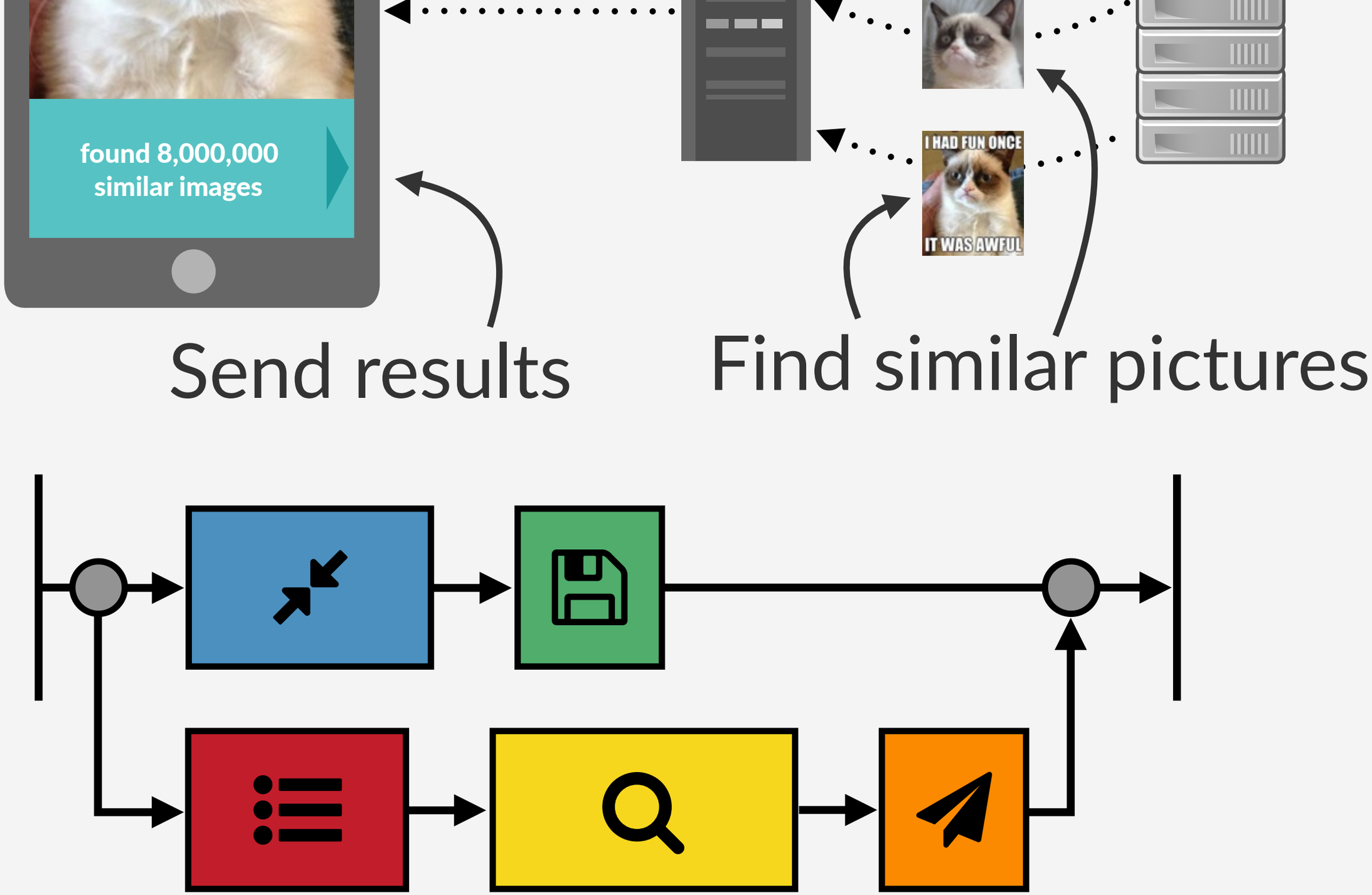
Add it to the database





Send results

Find similar pictures


found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

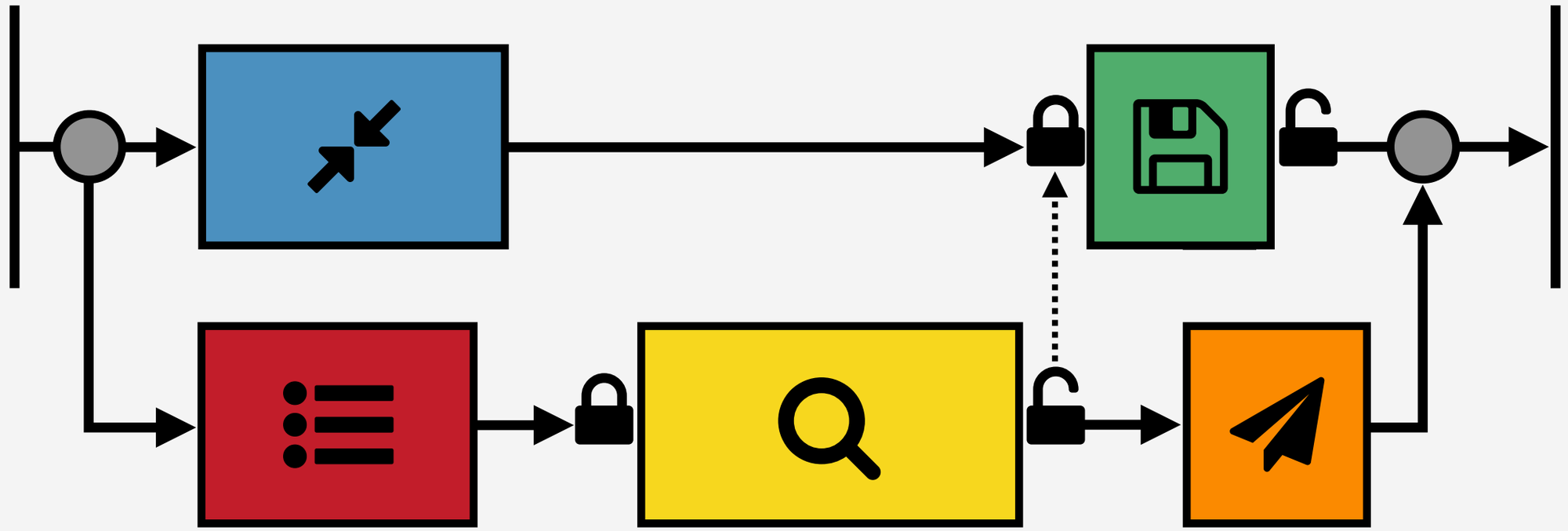
found 8,000,000
similar images

I HAD FUN ONCE

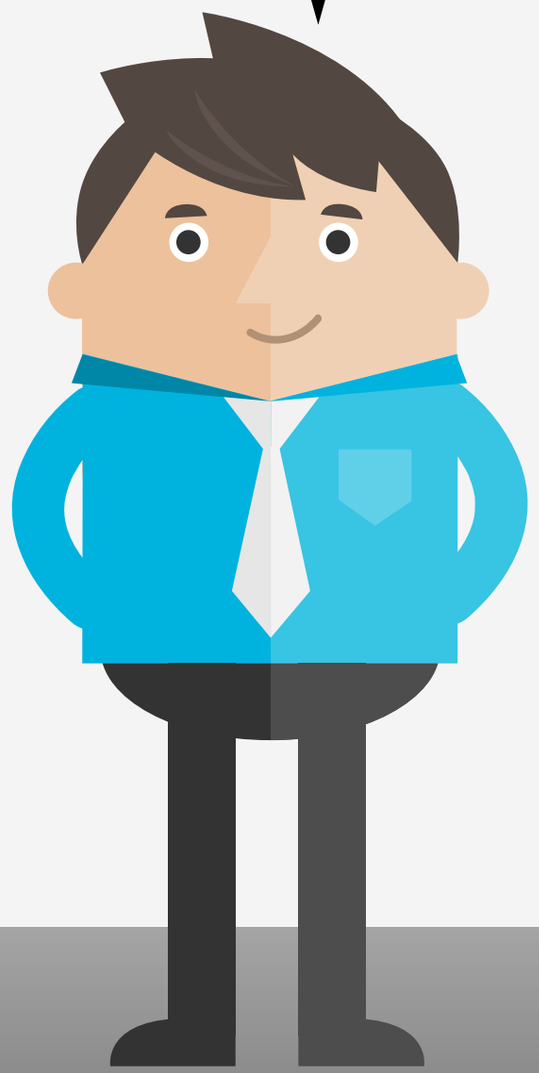
IT WAS AWFUL

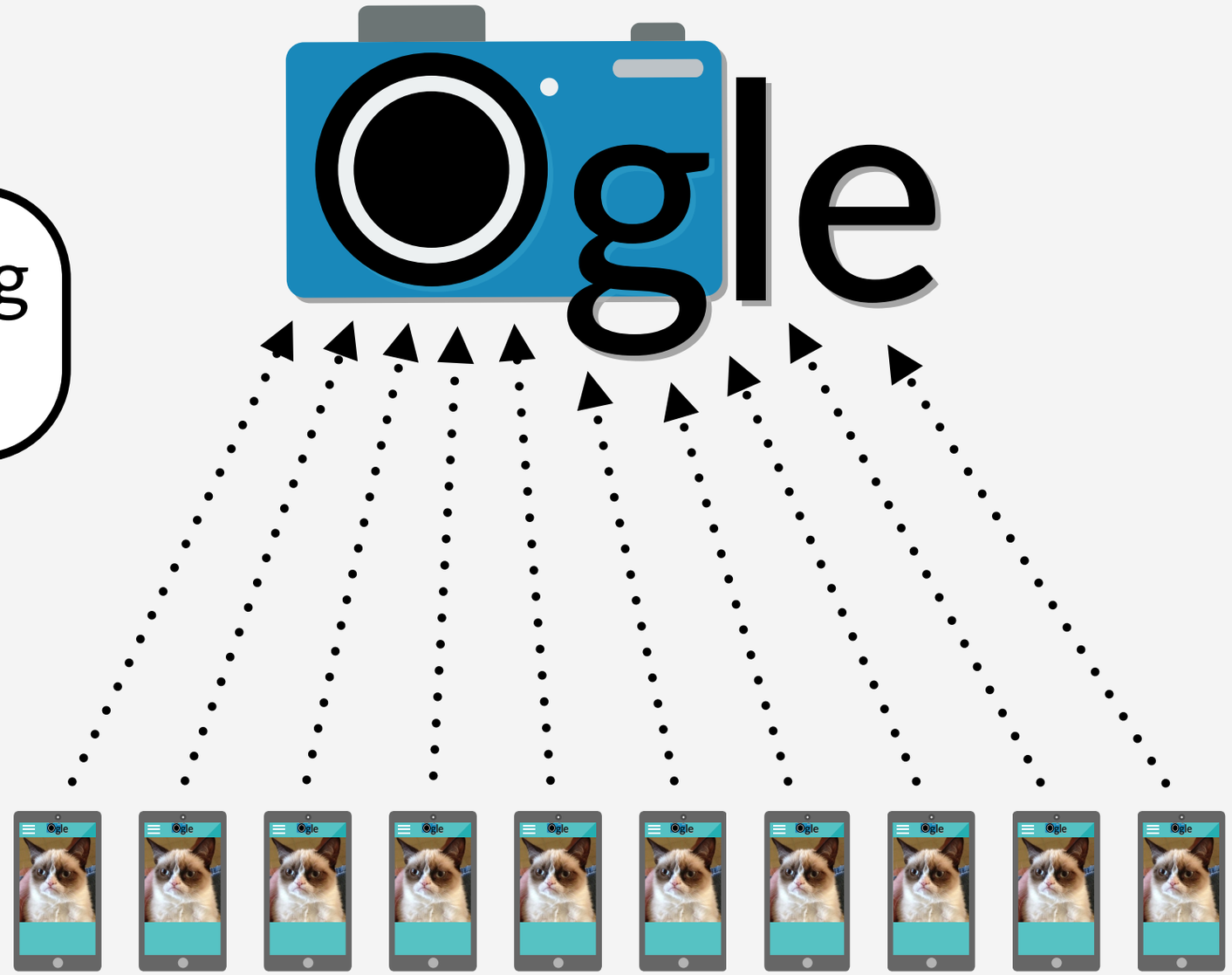
Send results

Find similar pictures

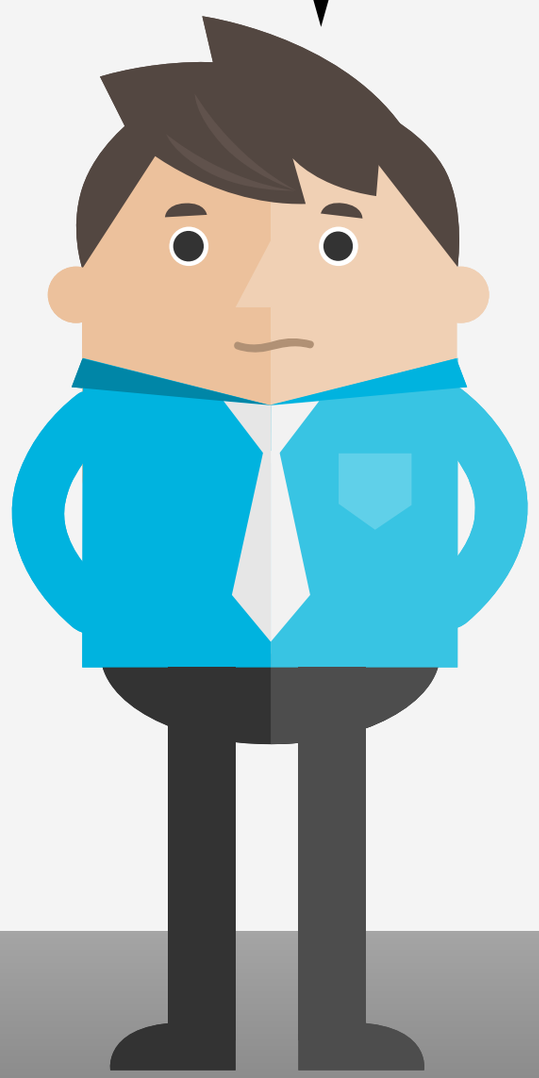


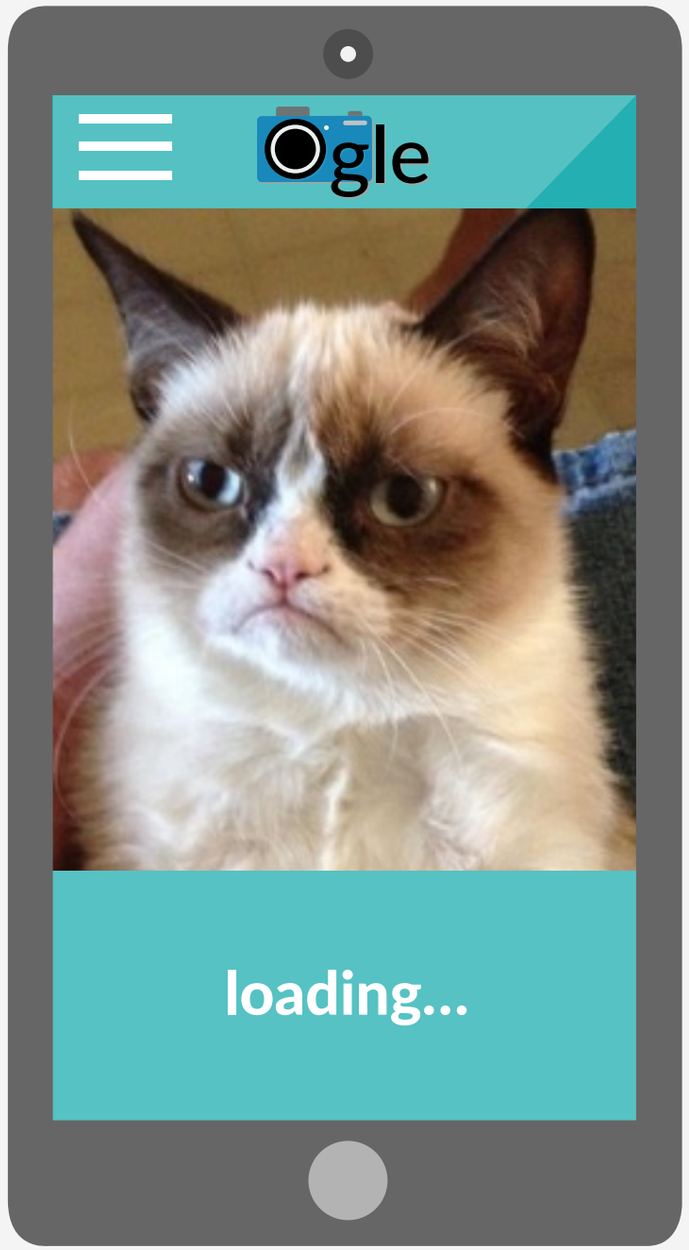
Ogle is totally disrupting image search.



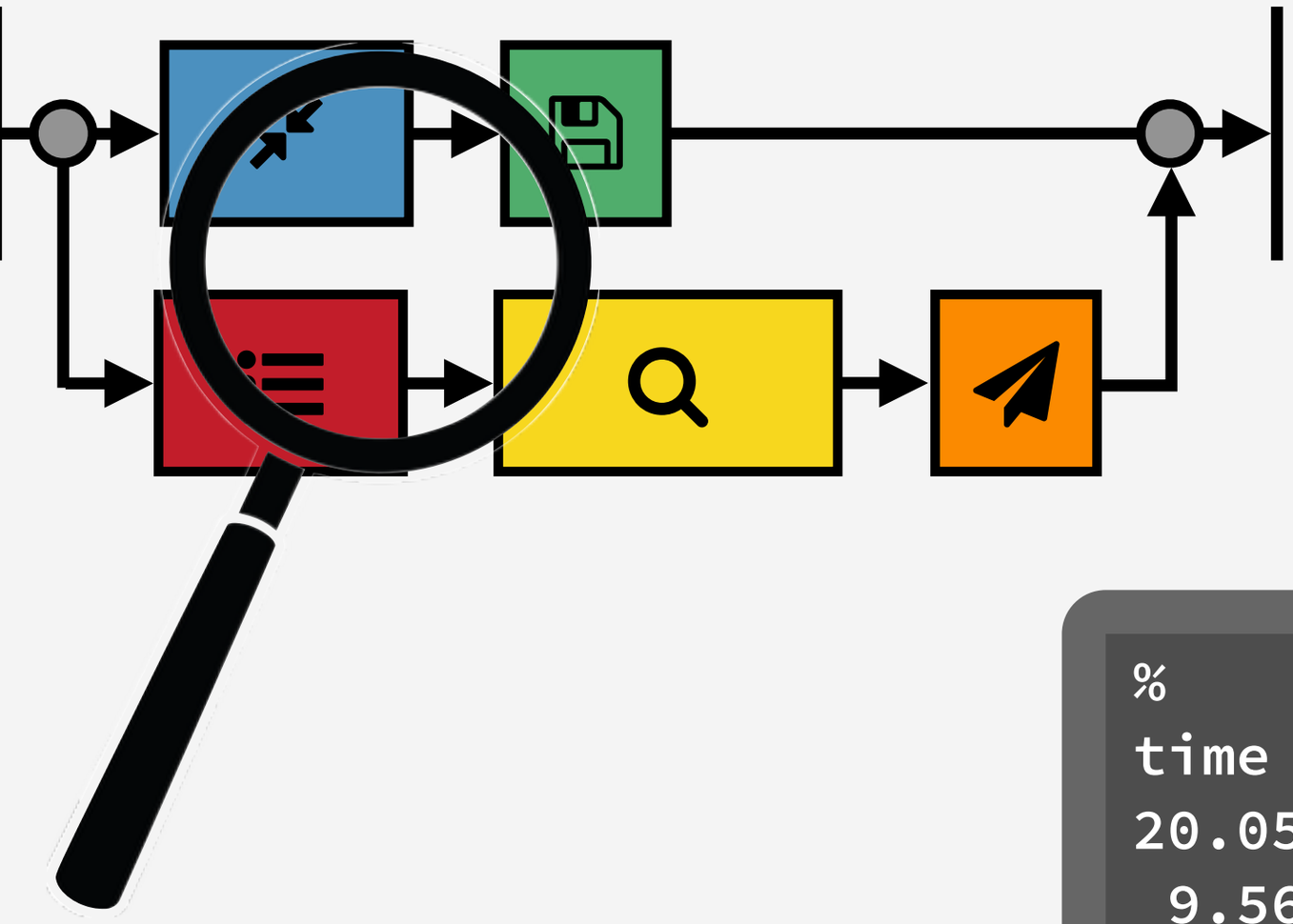


Ogle is too slow!



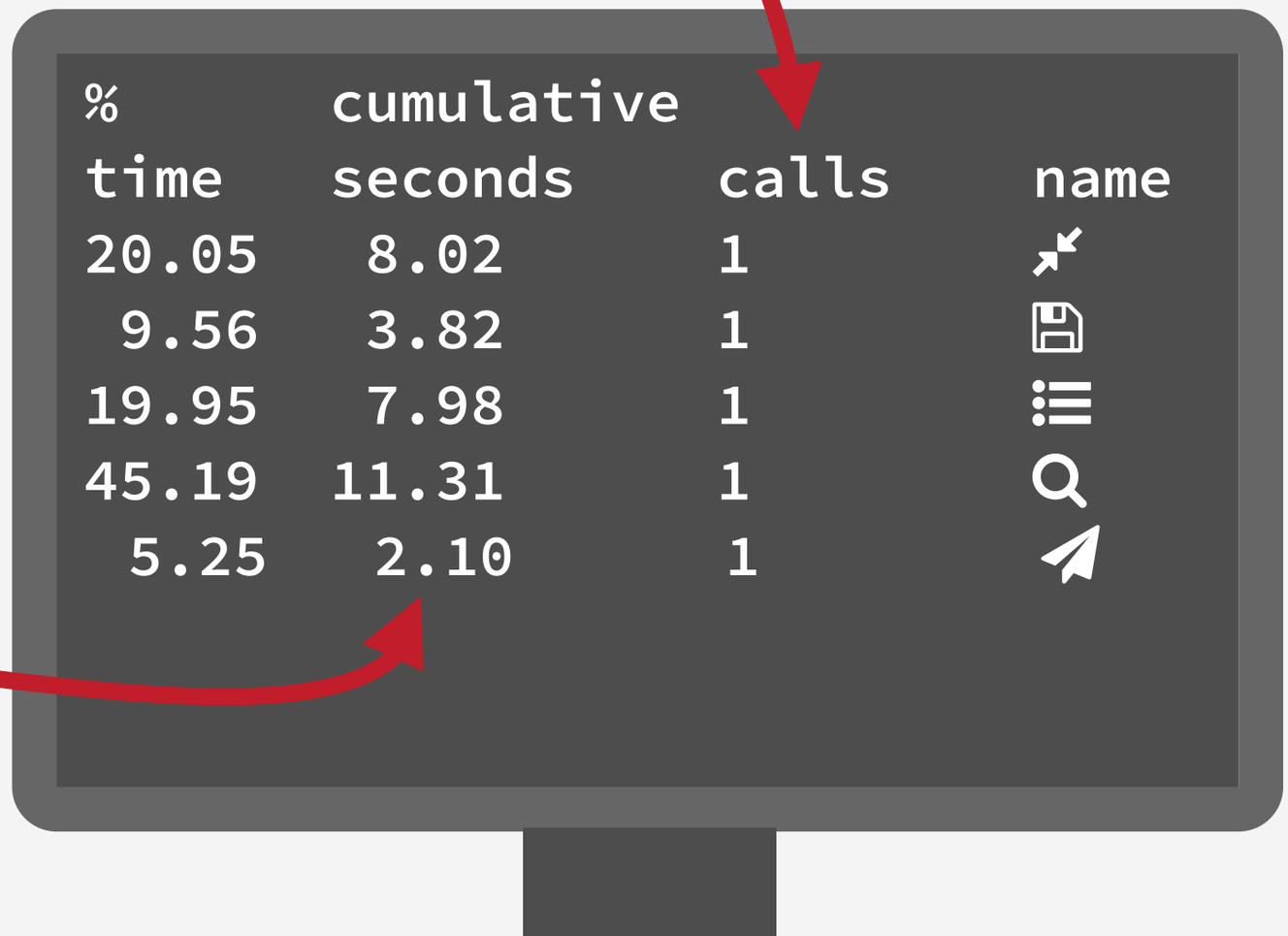


Software Profilers

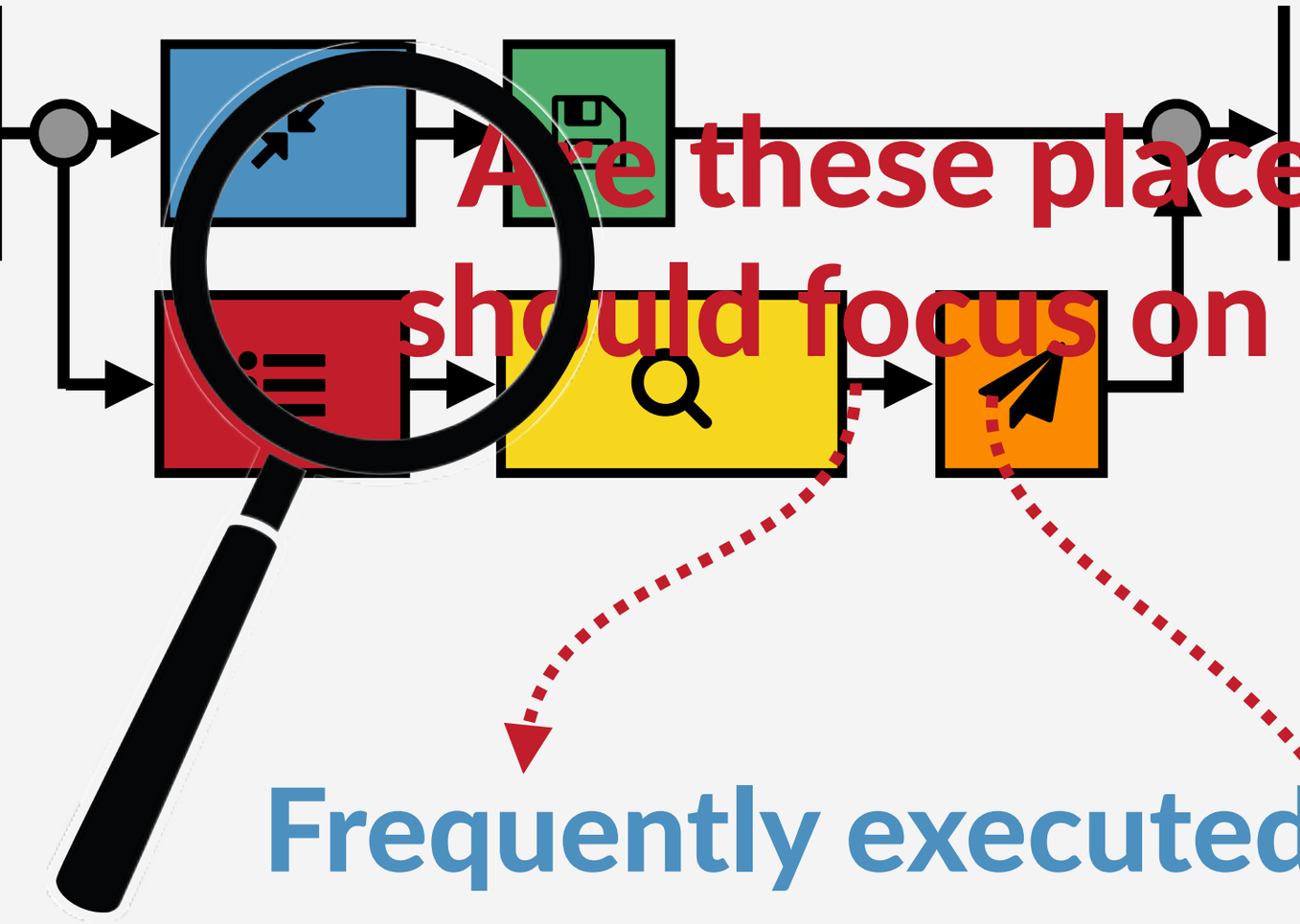


Number of calls to each function

Runtime for each function



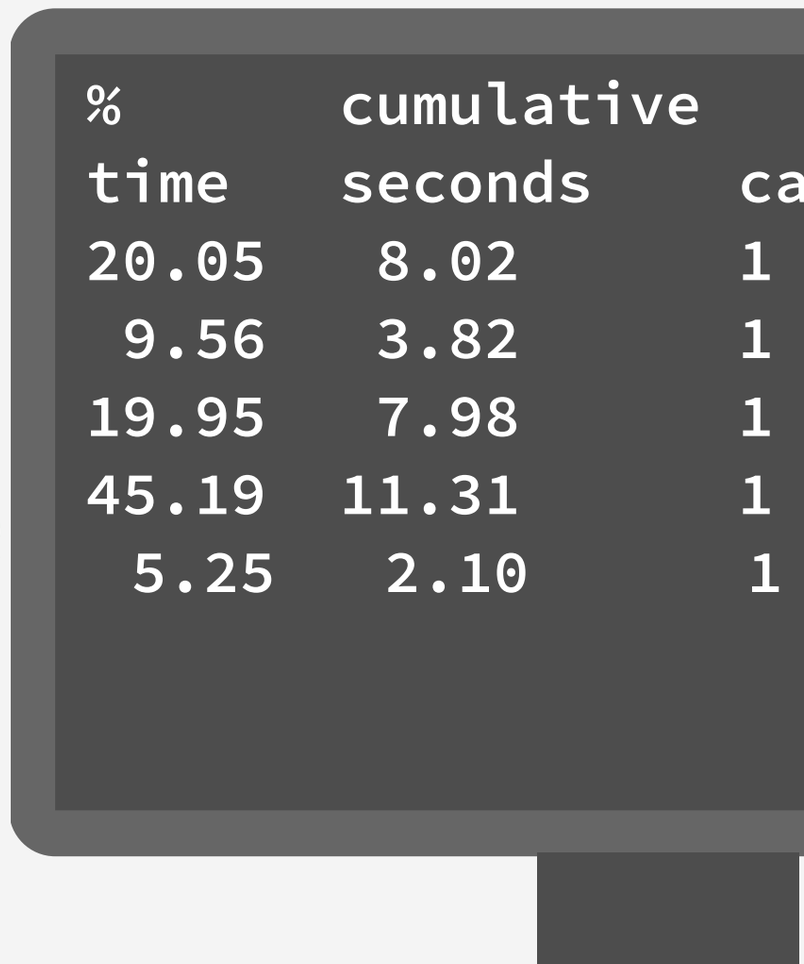
Software Profilers



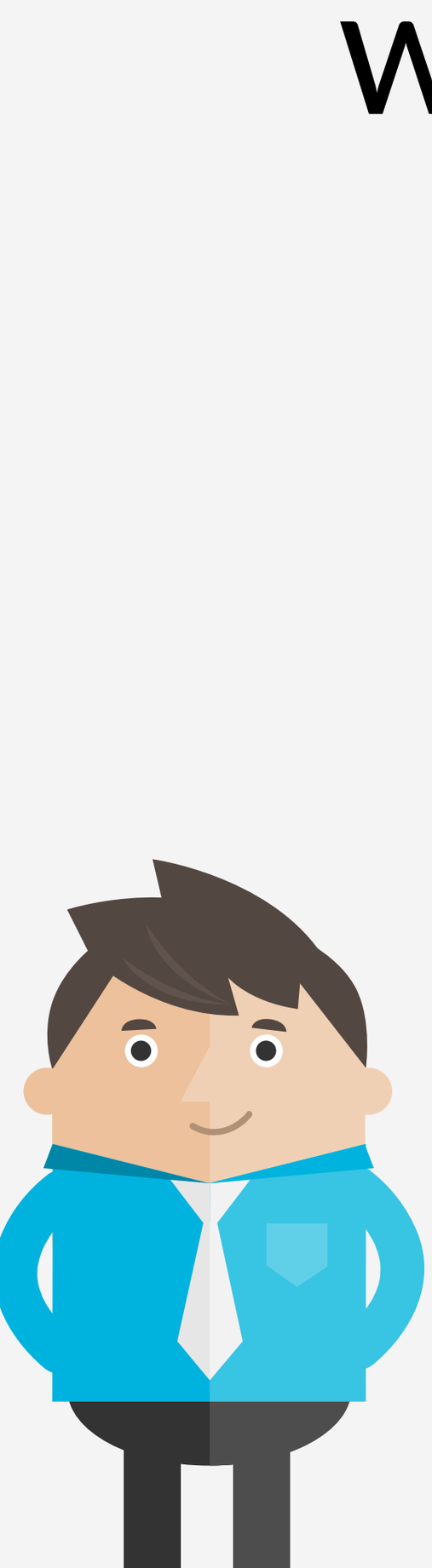
Are these places where Bob should focus on performance?

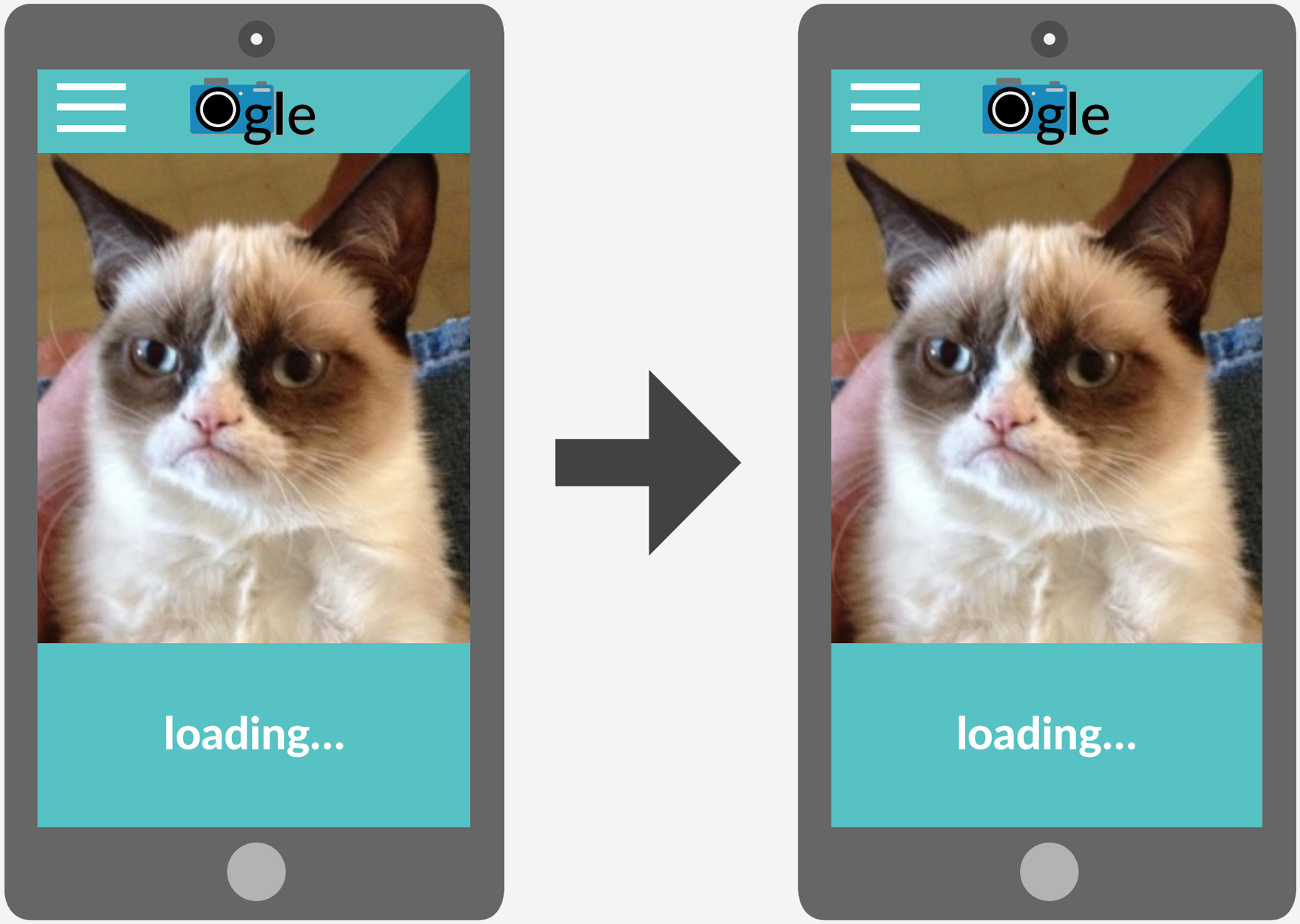
Frequently executed code

Code that runs for a long time



Would this speed up Ogle?



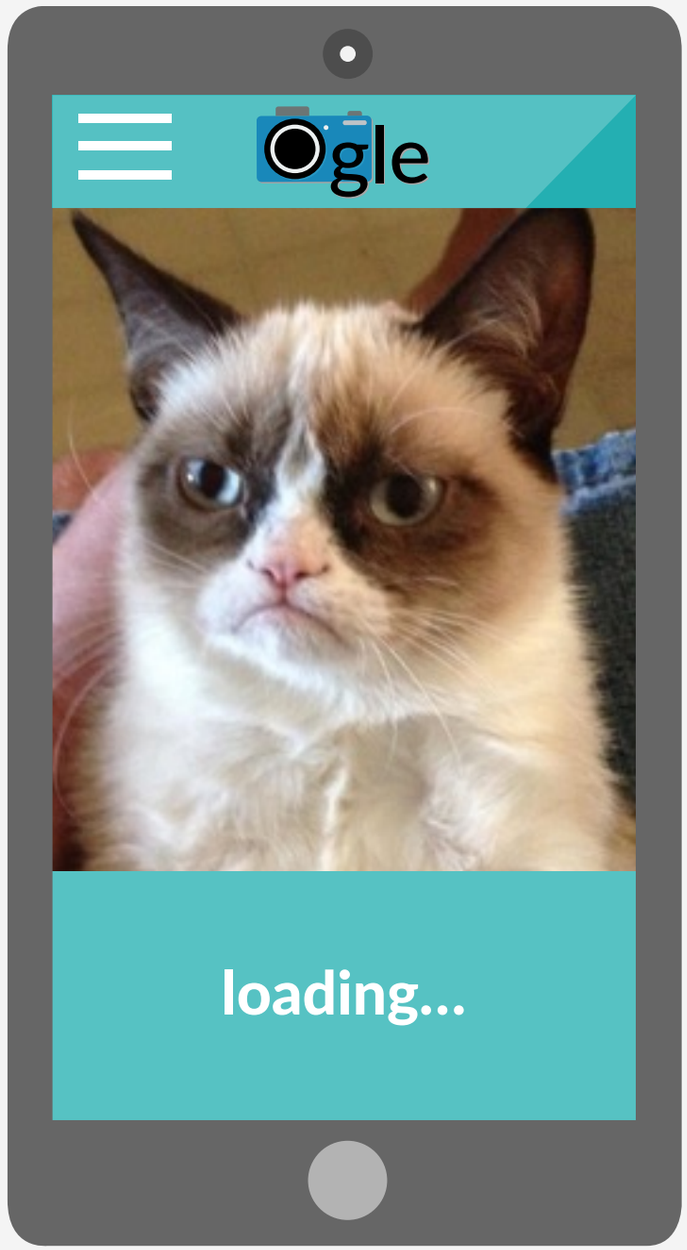


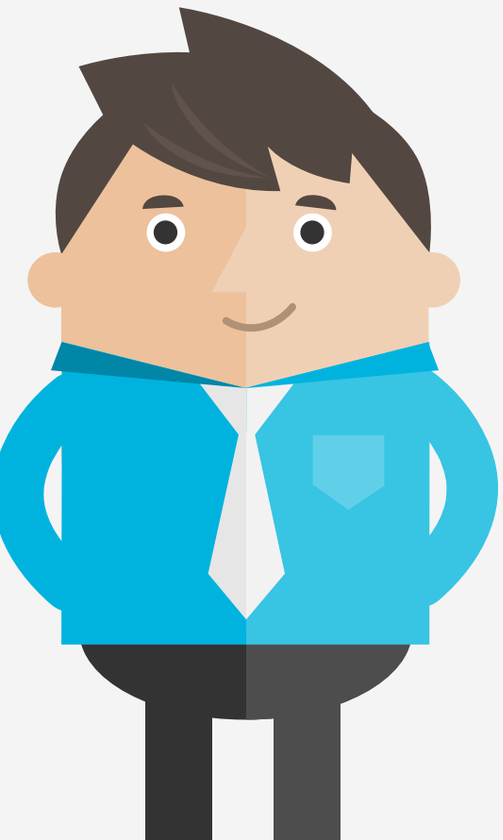
Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

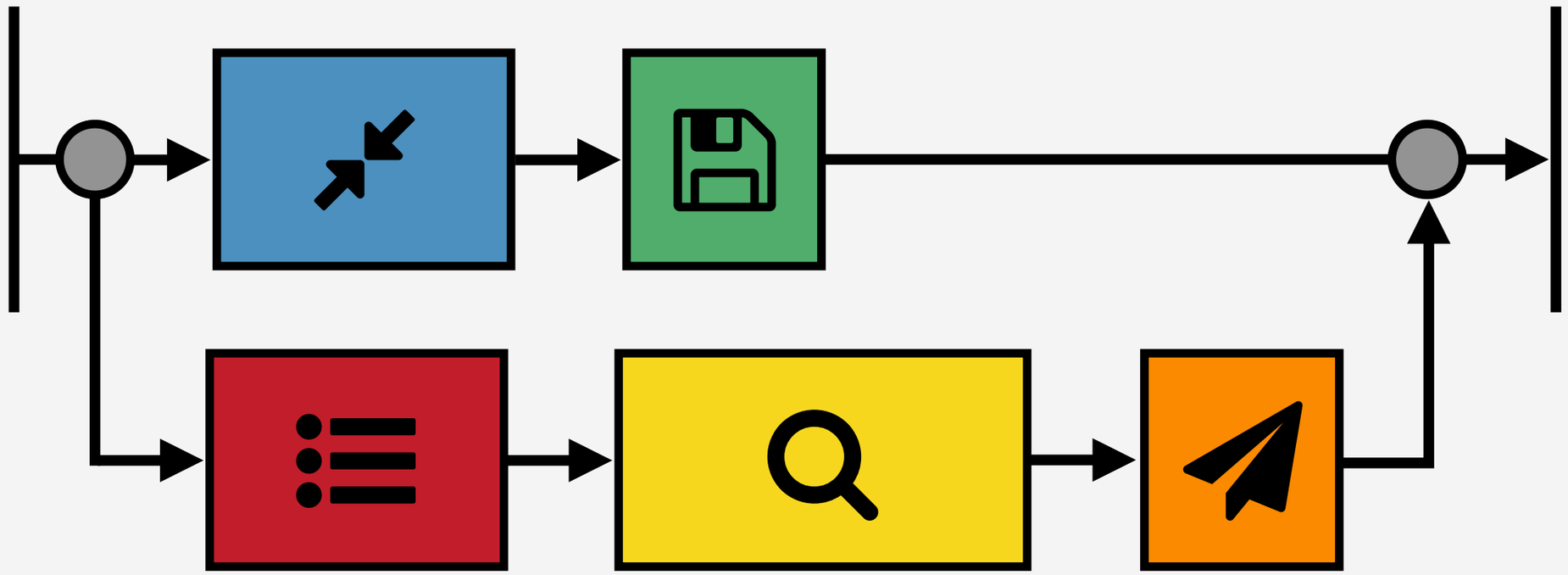
Profilers do a bad job finding important code in parallel programs.





We need better tools.

What *would* speed up Ogle?

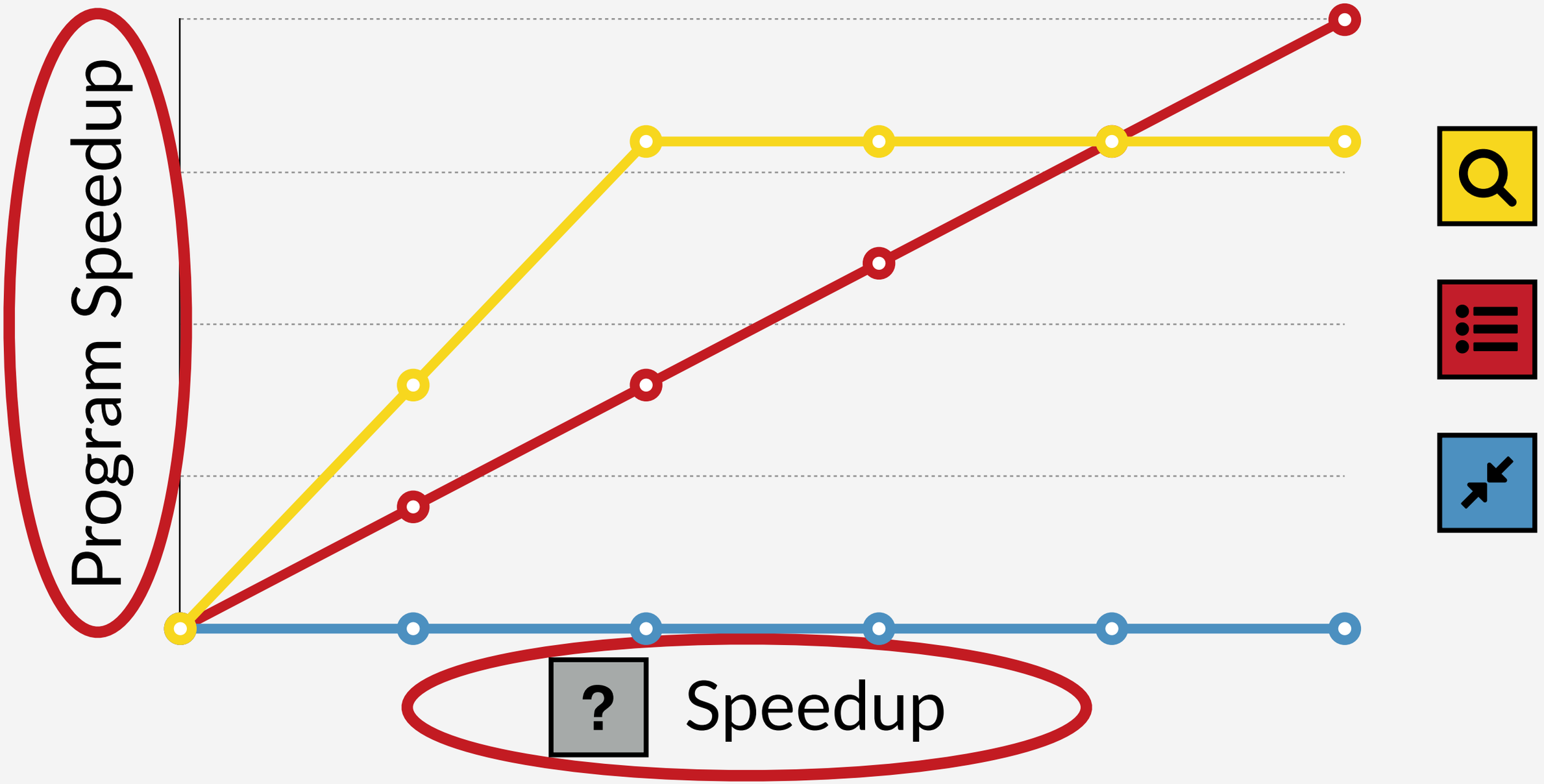


Hey, look over here

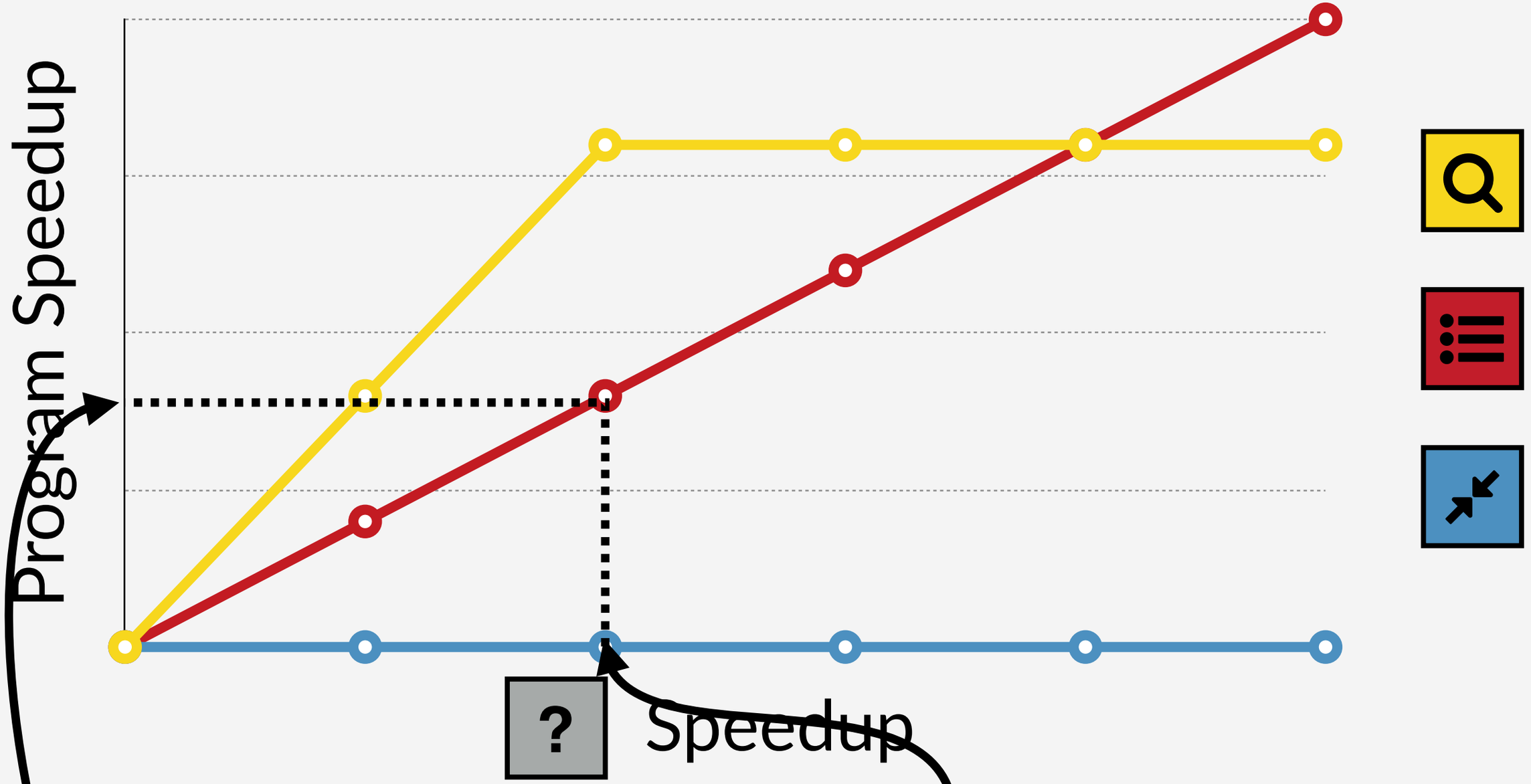
What would this information look like?

Causal Profile

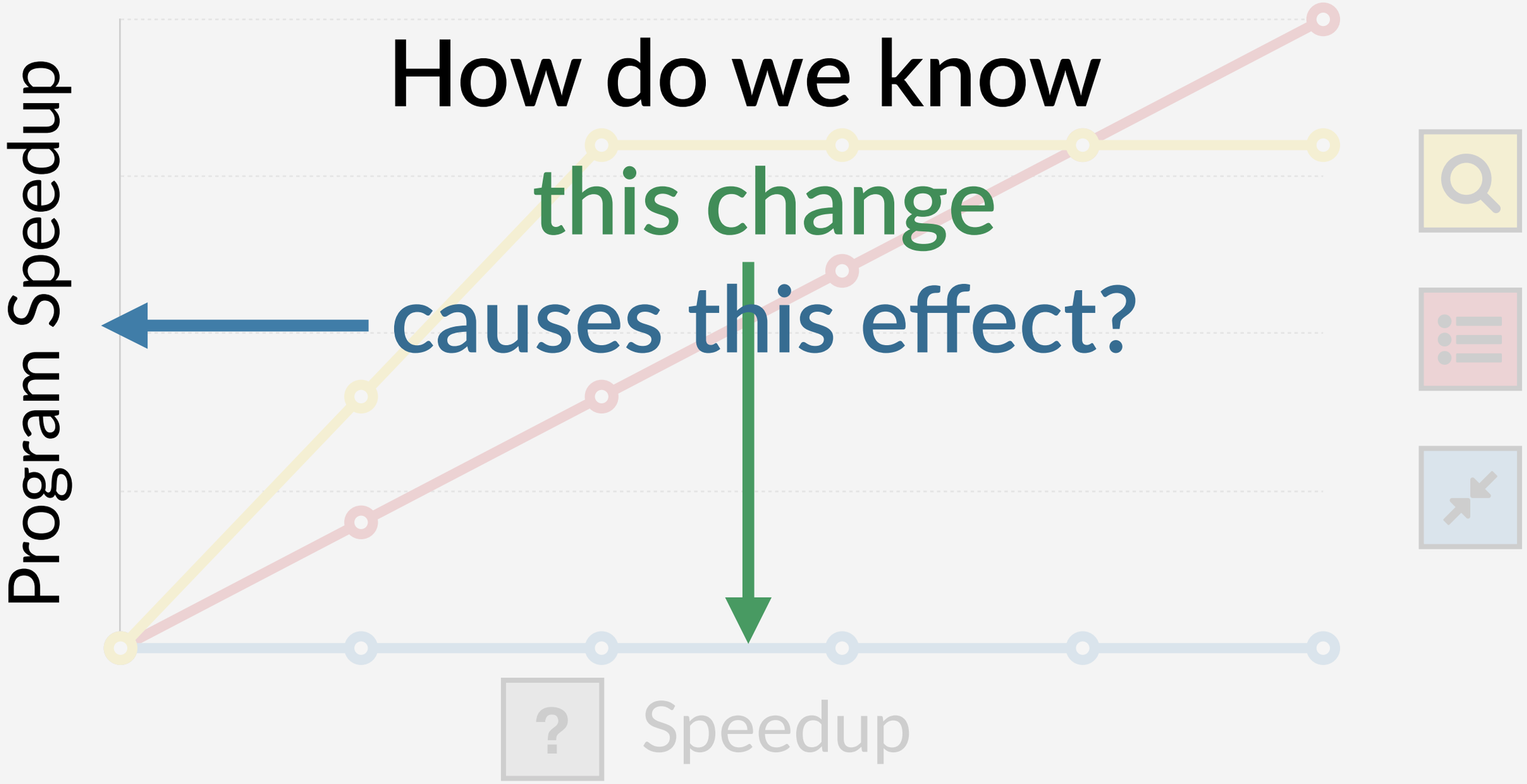
Tells you where optimizations will make a difference



Causal Profile



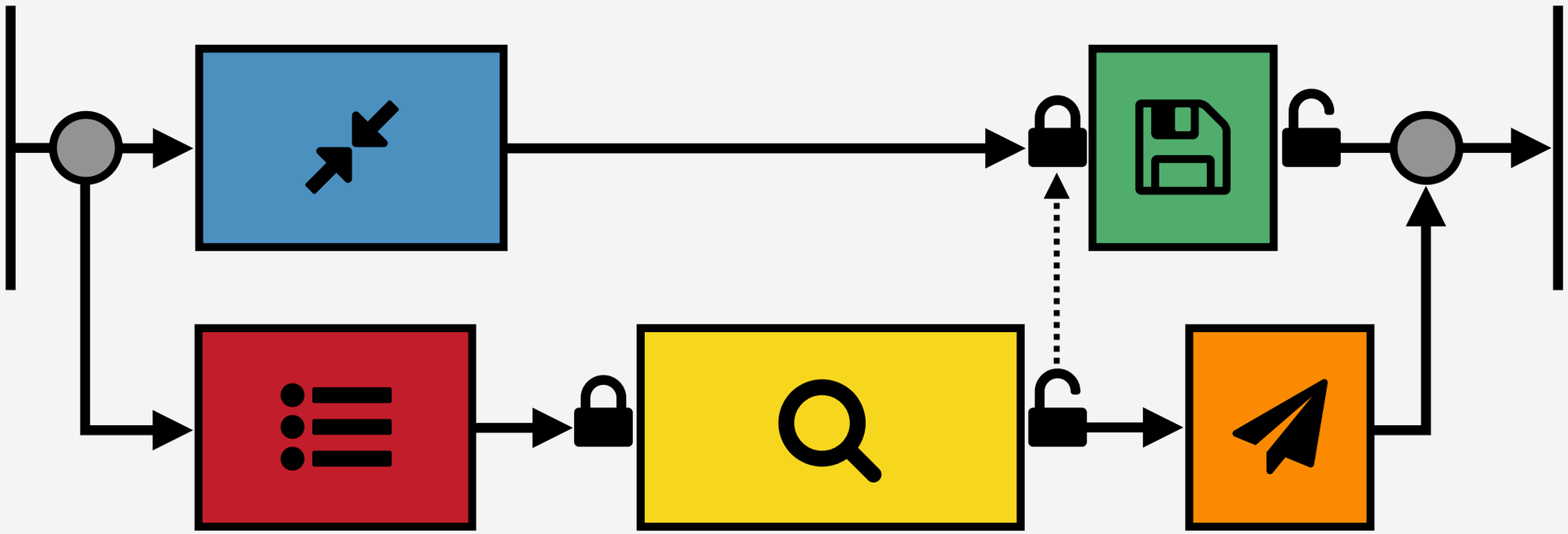
Causal Profile



Run an experiment

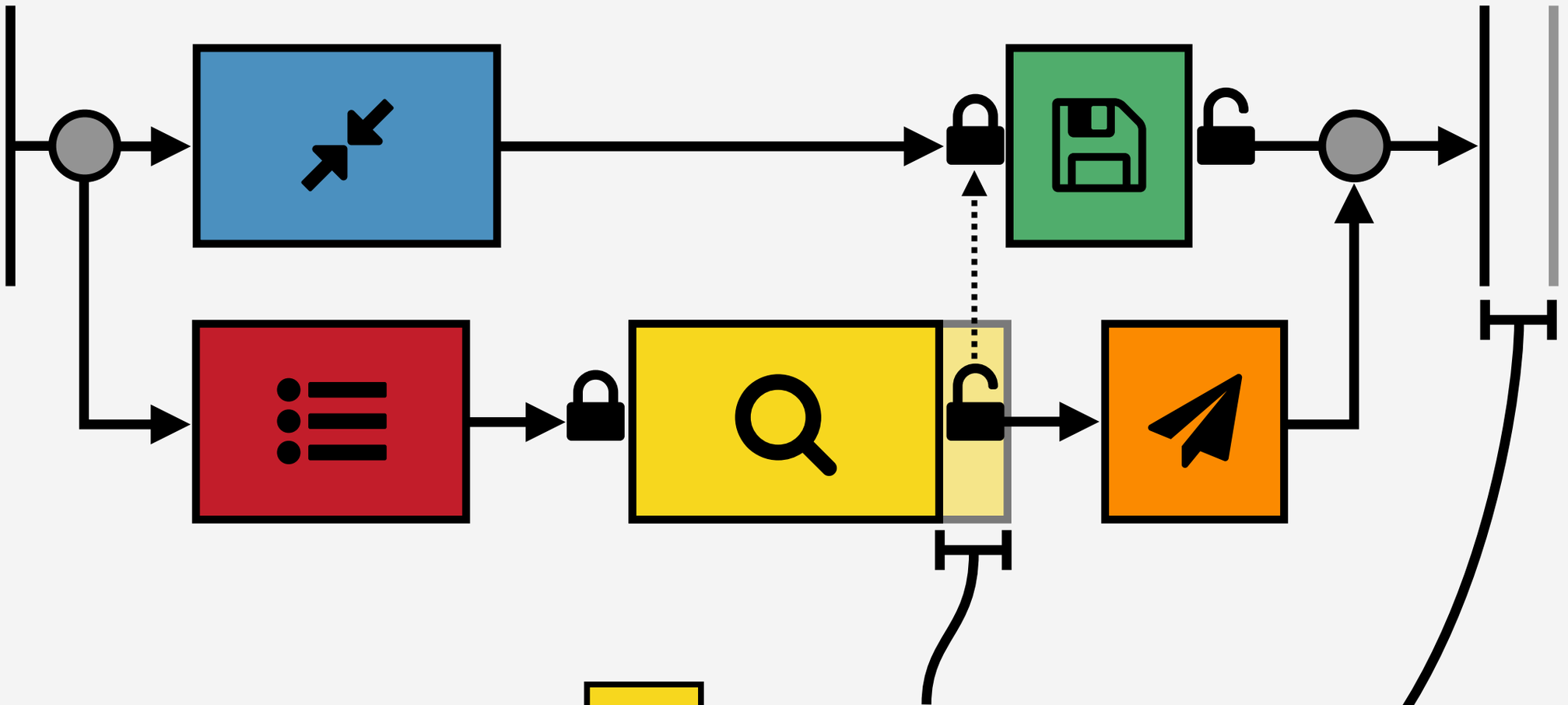
Performance Experiments

If we could magically speed up 🔍 ...



Performance Experiments

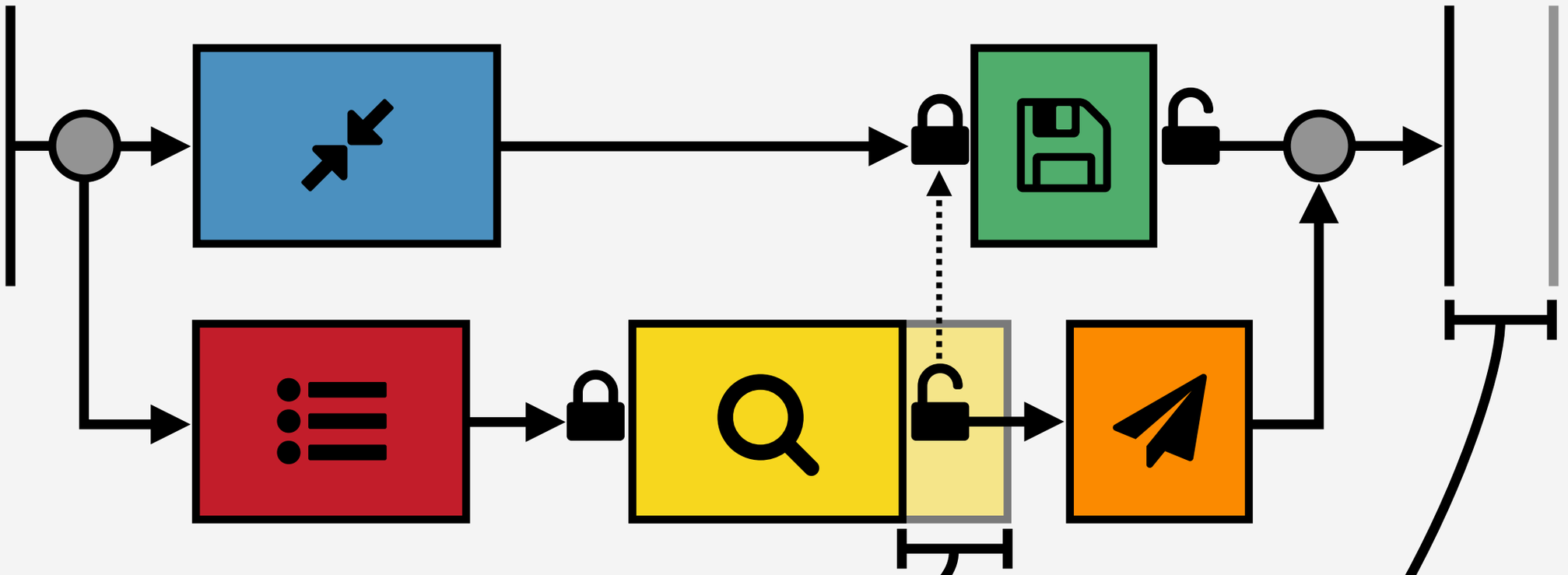
If we could magically speed up 🔍 ...



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

Performance Experiments

If we could magically speed up 🔍 …

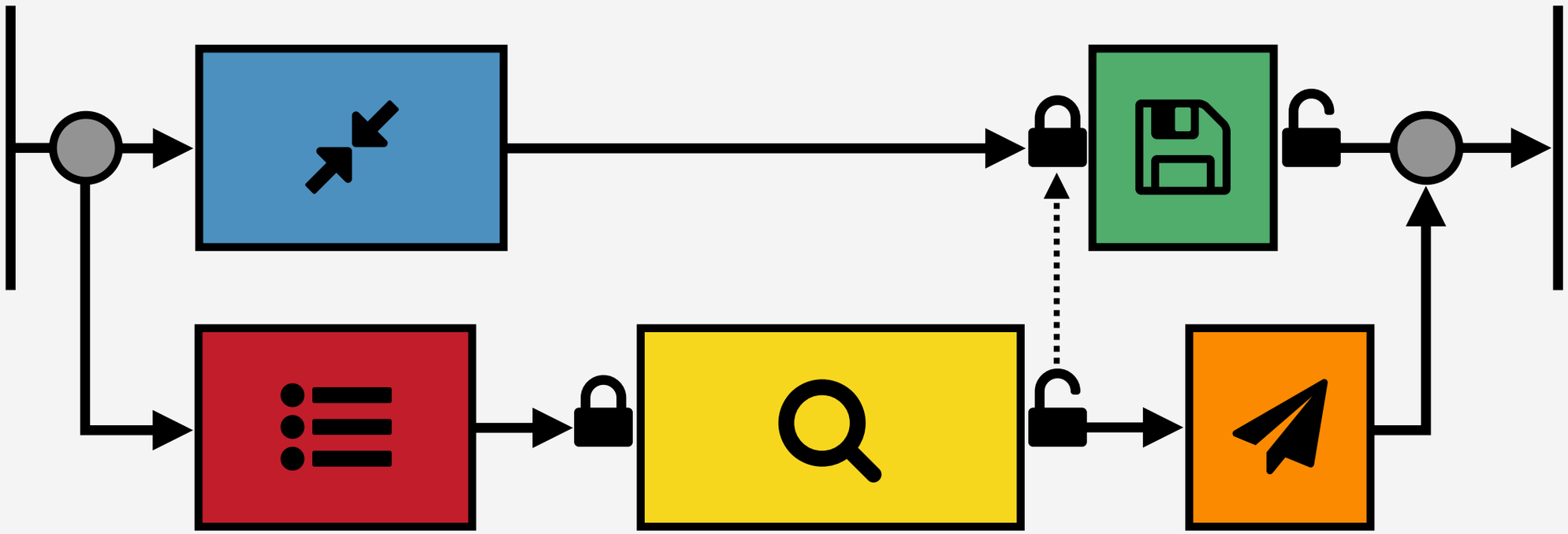


More speedup in 🔍 …

leads to a larger program speedup.

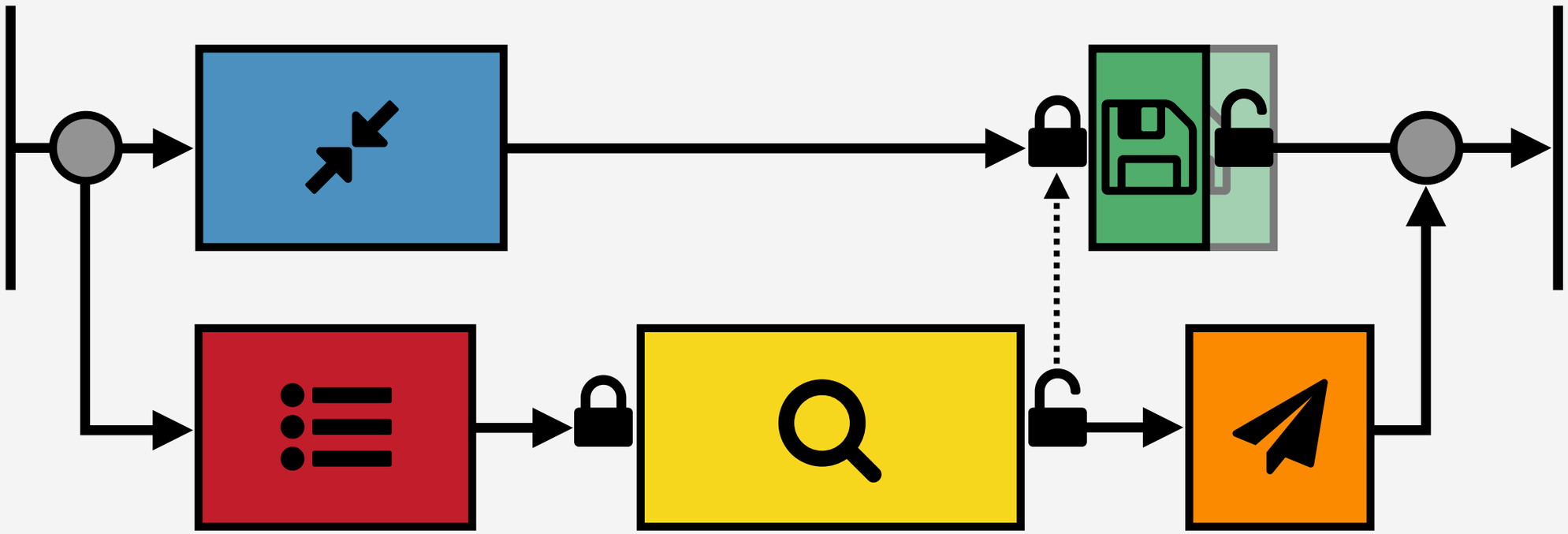
Performance Experiments

If we could magically speed up  ...



Performance Experiments

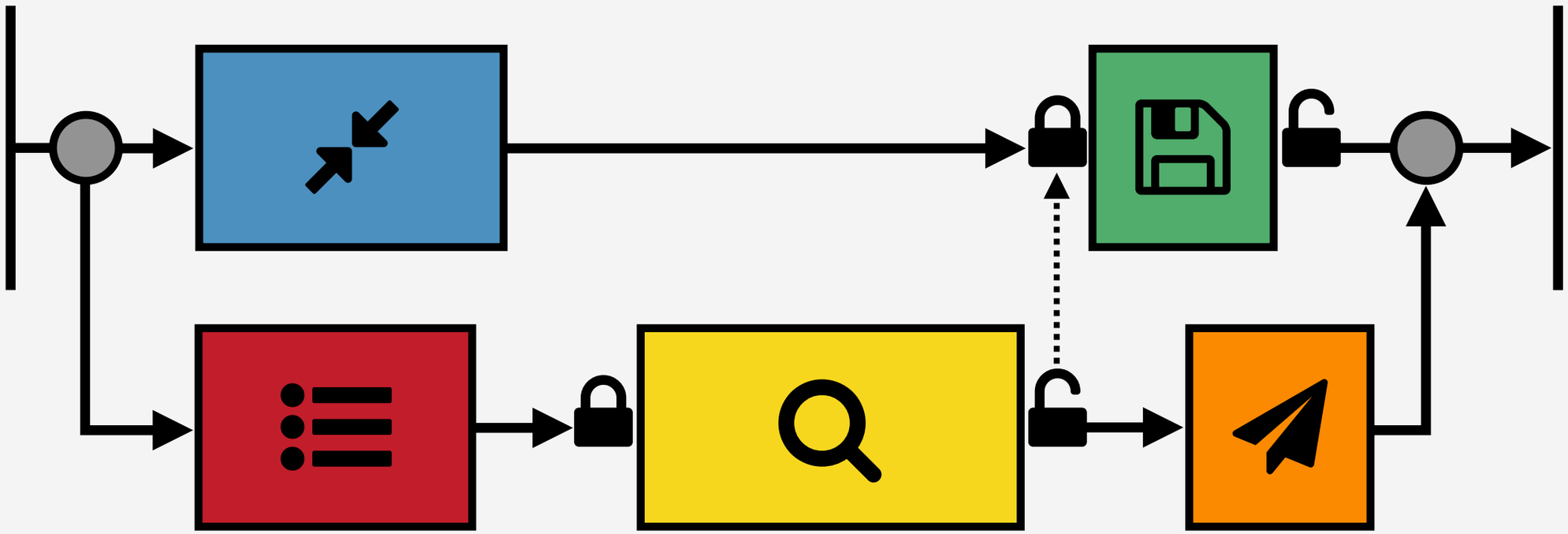
If we could magically speed up  ...



No program speedup

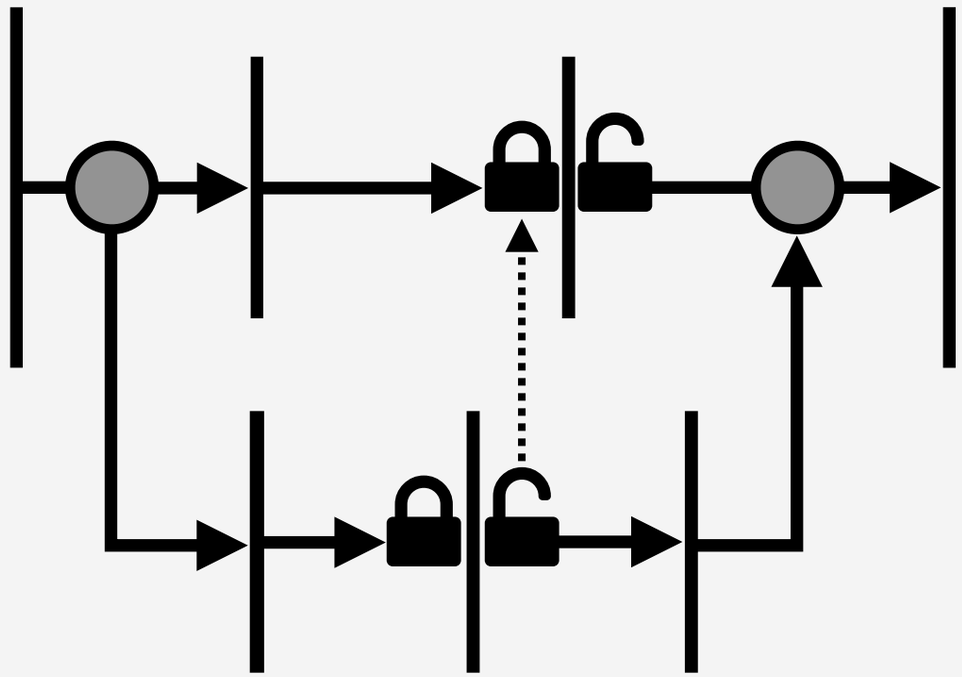
Performance Experiments

We’re going to have to do this without magic.



Performance Experiments

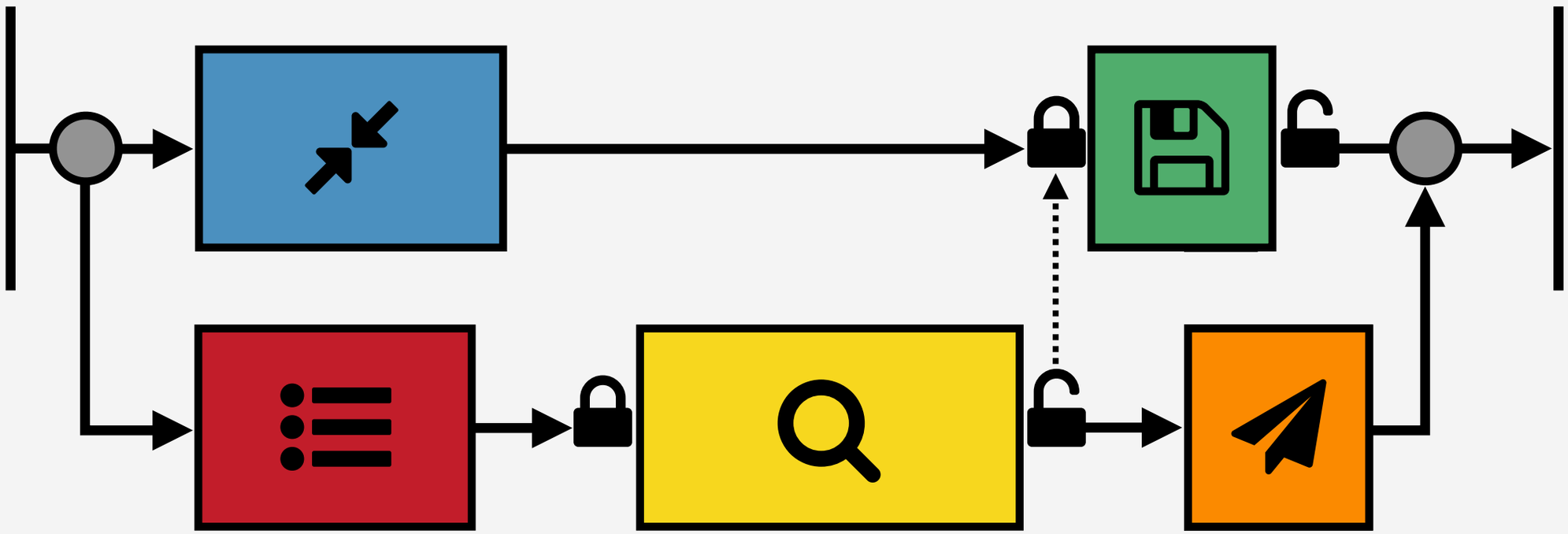
We’re going to have to do this without magic.



Otherwise we’d just do this.

Virtual Speedup

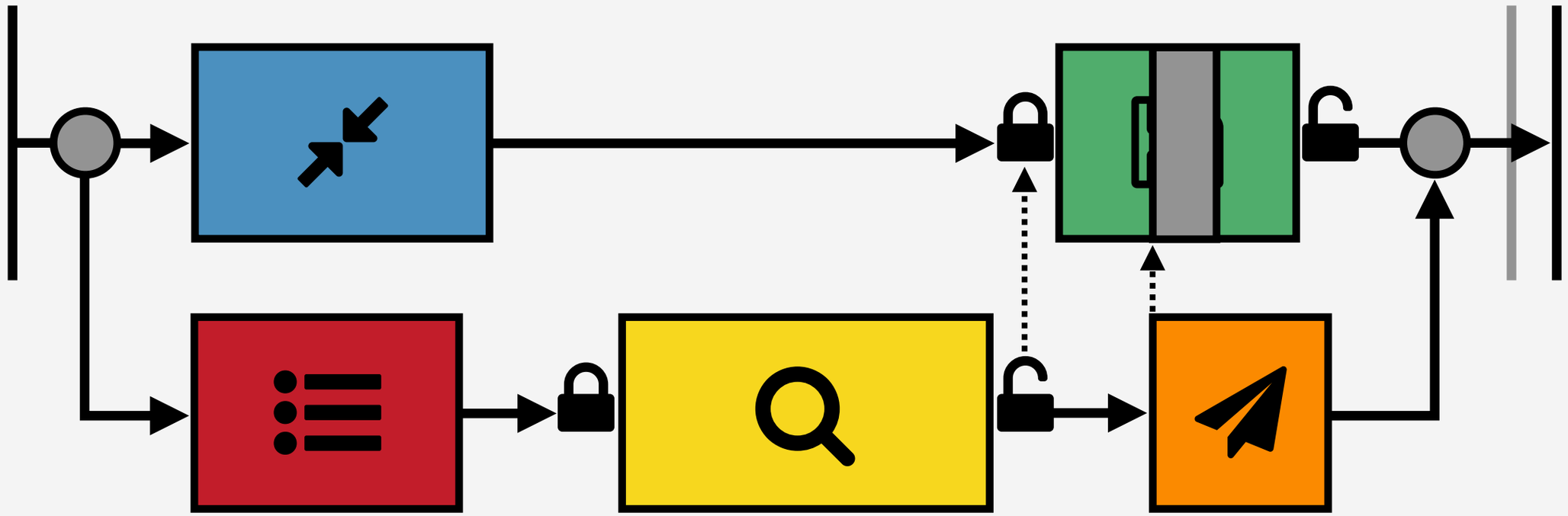
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

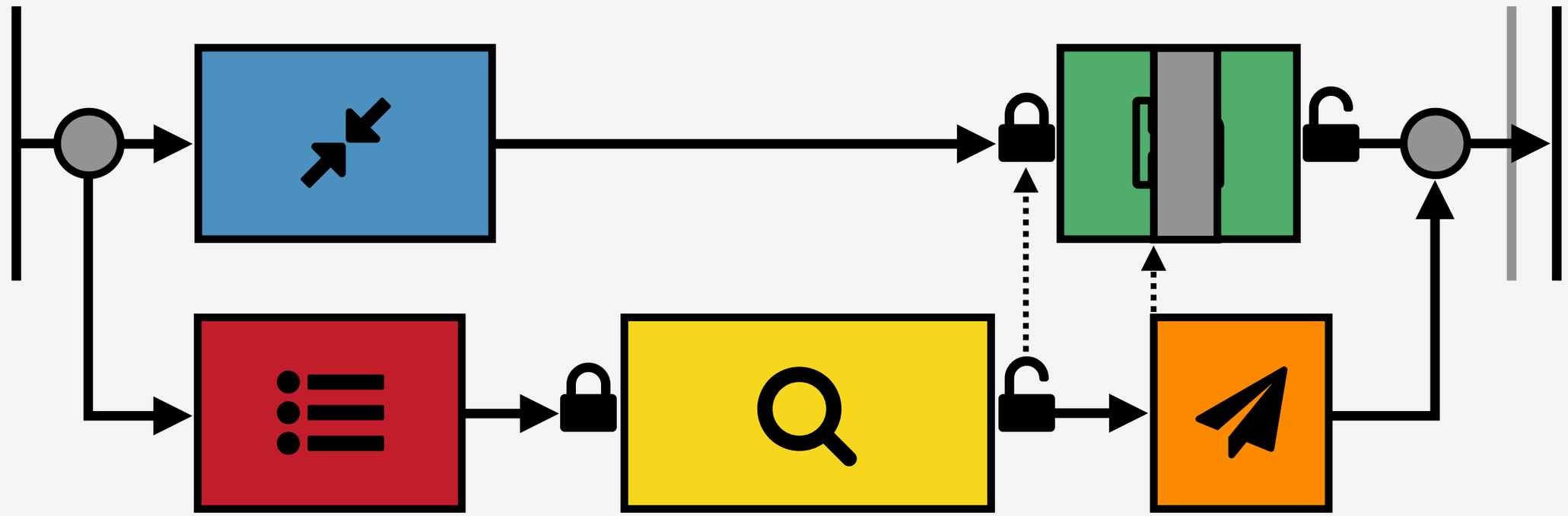
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

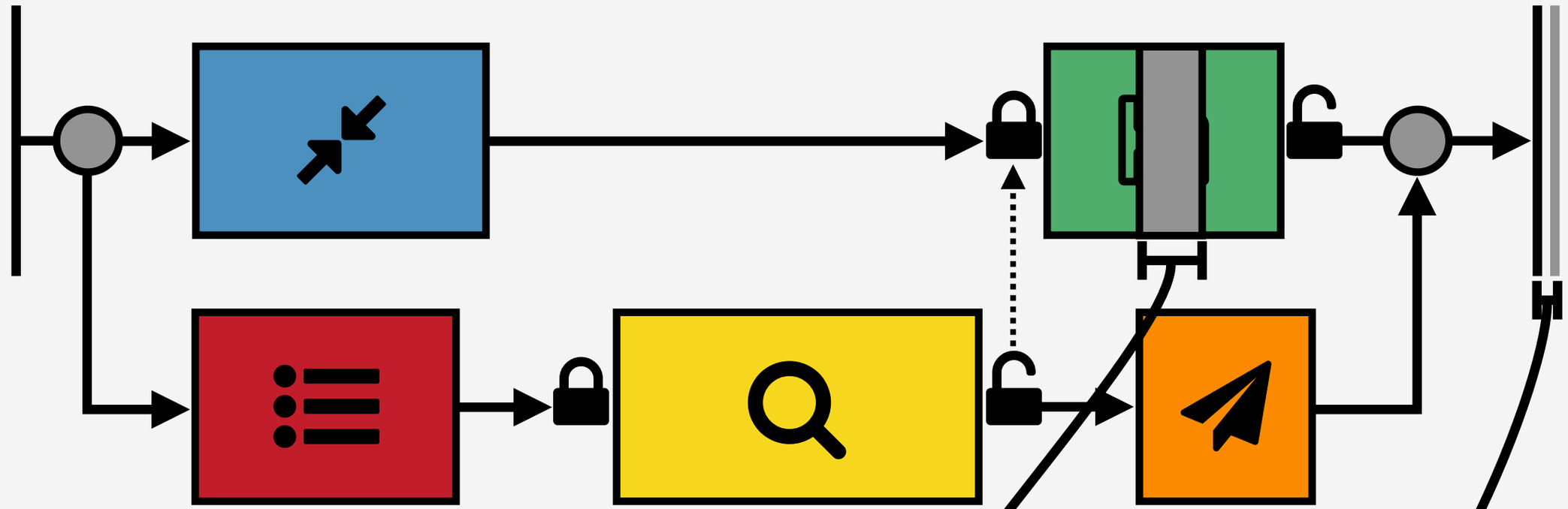
“Speed up”  by slowing everything else down.



To account for the size of the delay…

Virtual Speedup

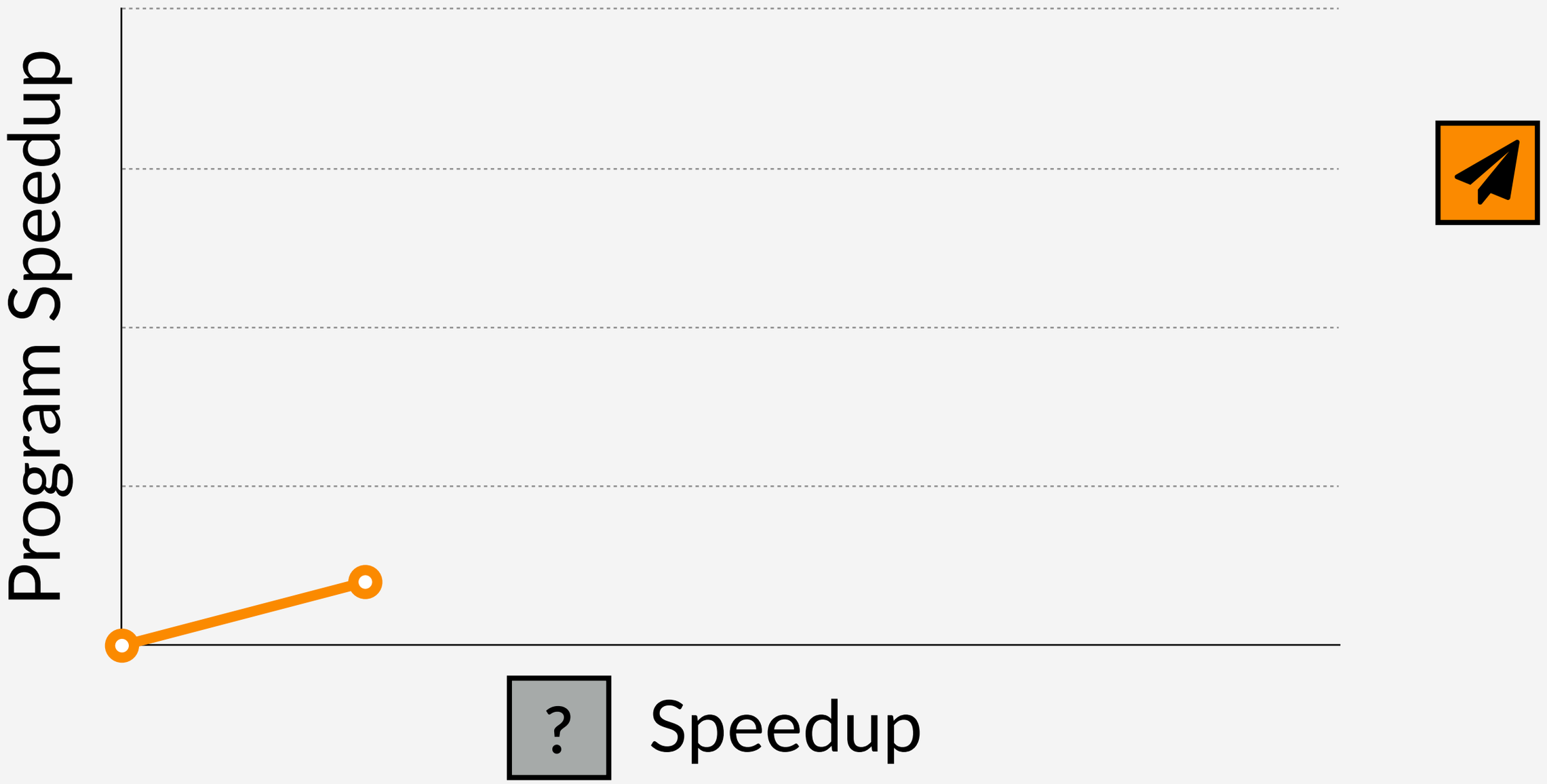
“Speed up” by slowing everything else down.



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

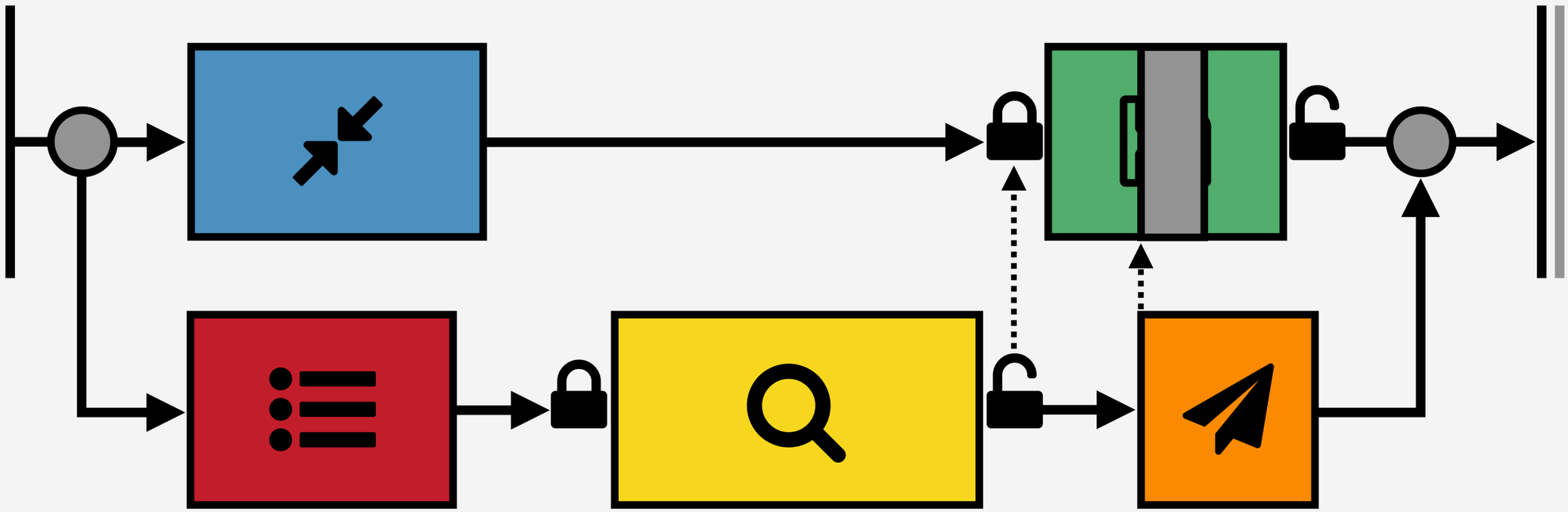
Speedup Results

Speedup Results



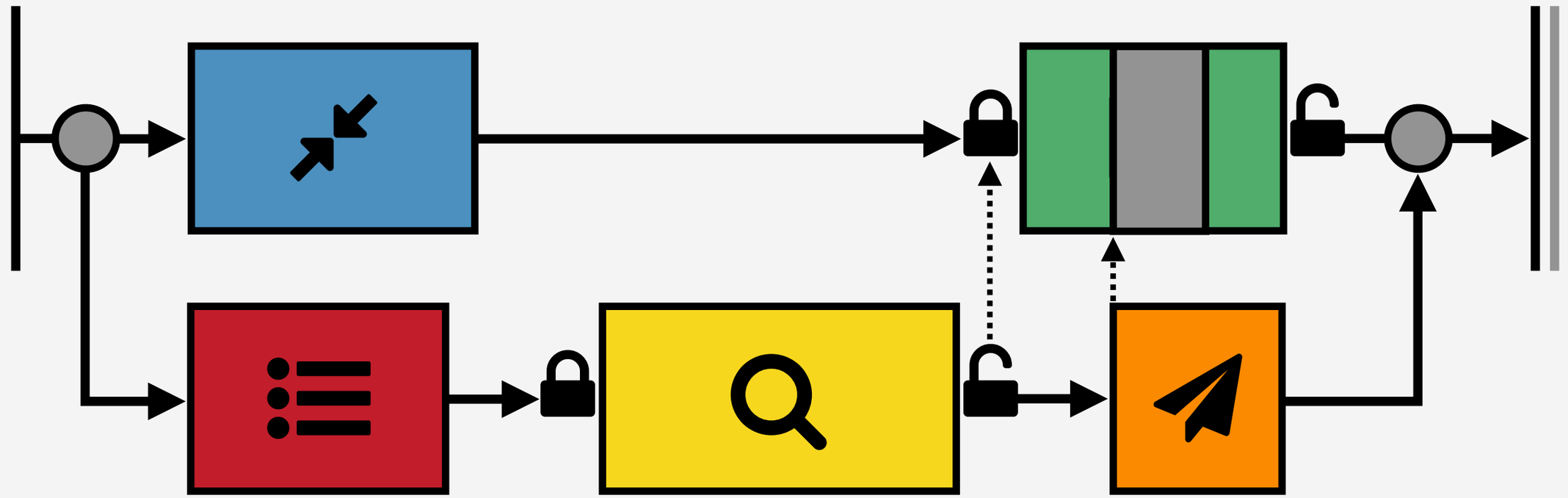
Virtual Speedup

“Speed up”  by slowing everything else down.



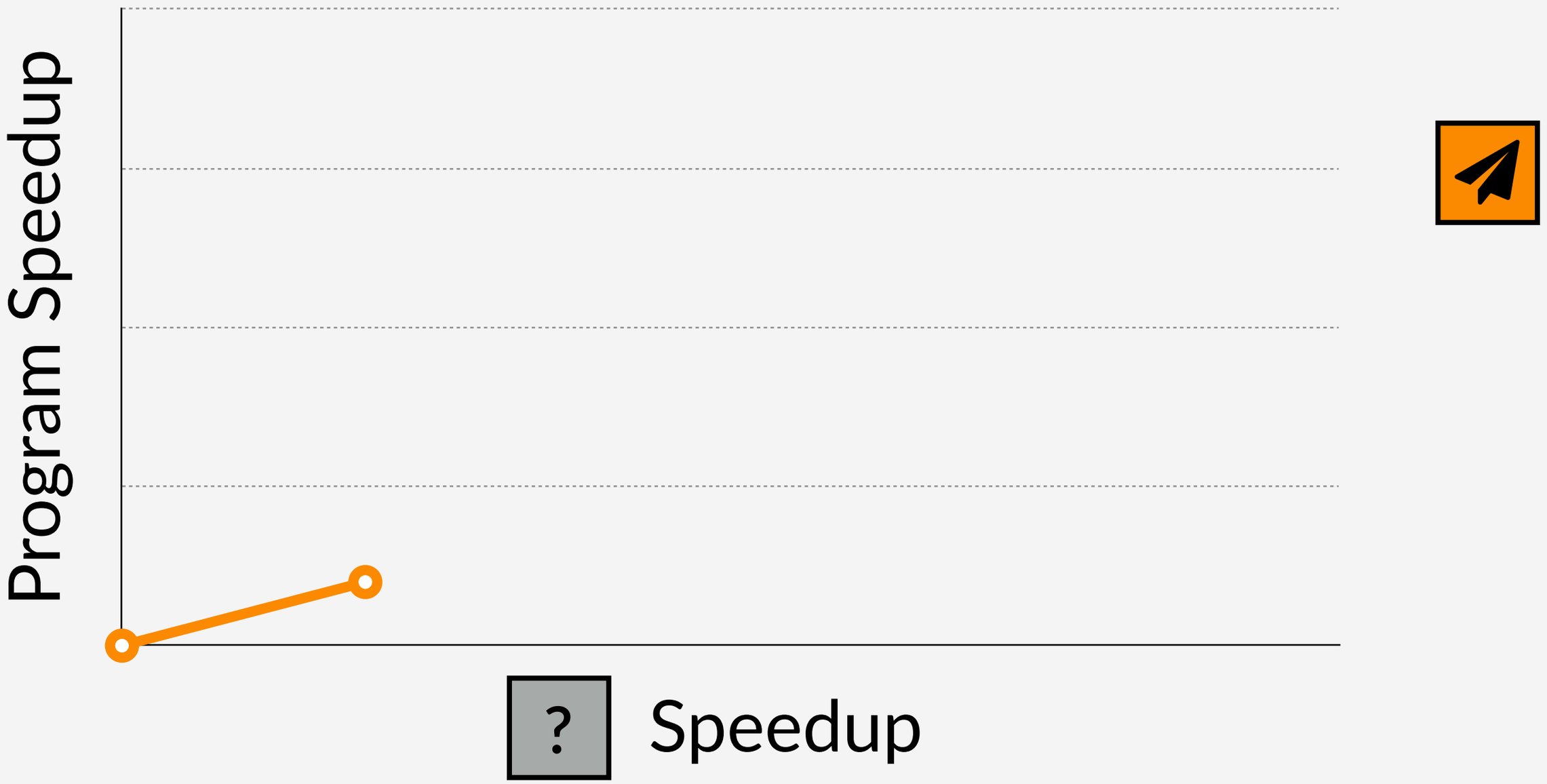
Virtual Speedup

“Speed up”  by slowing everything else down.

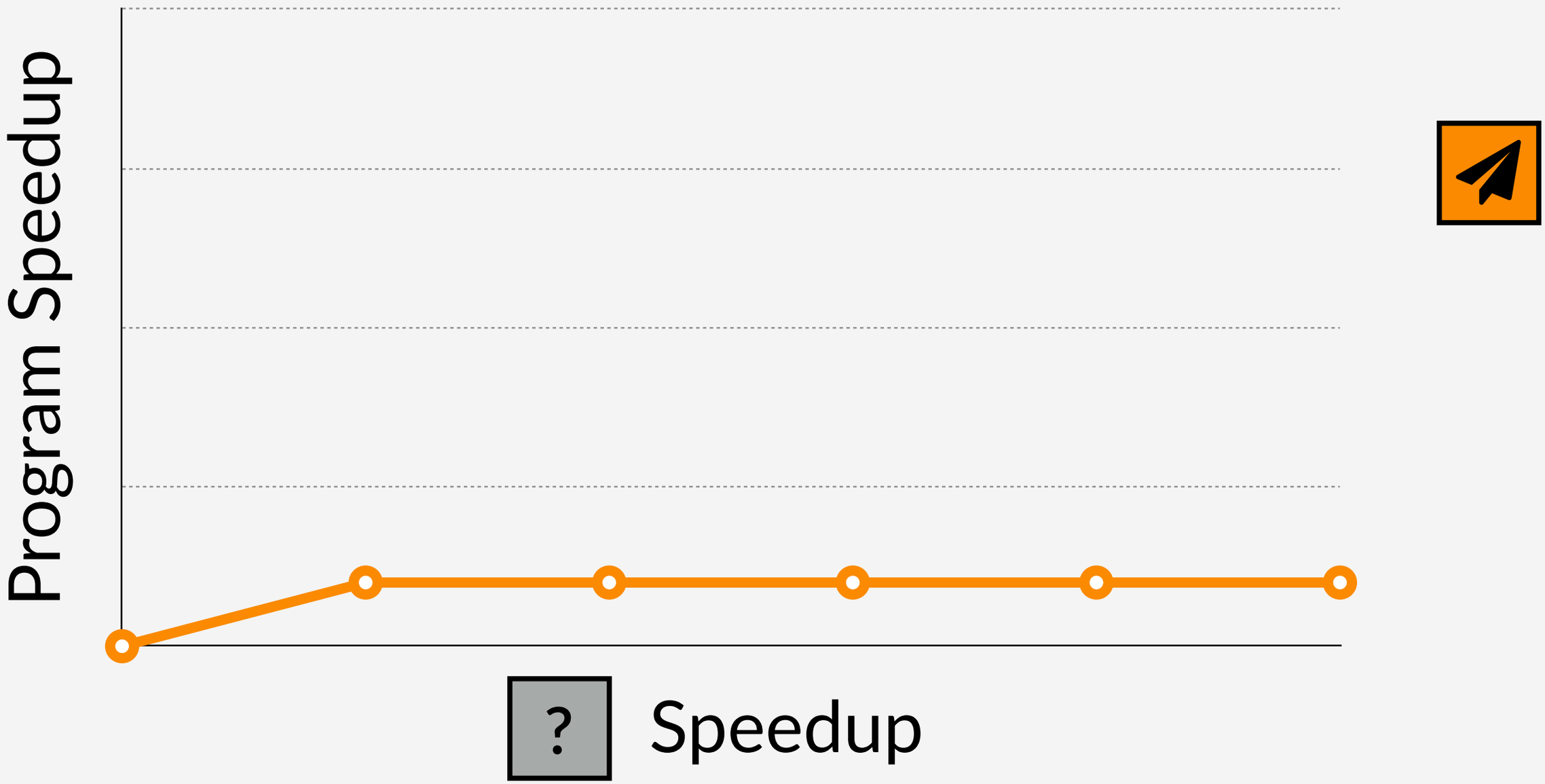


A larger speedup has no additional effect

Speedup Results

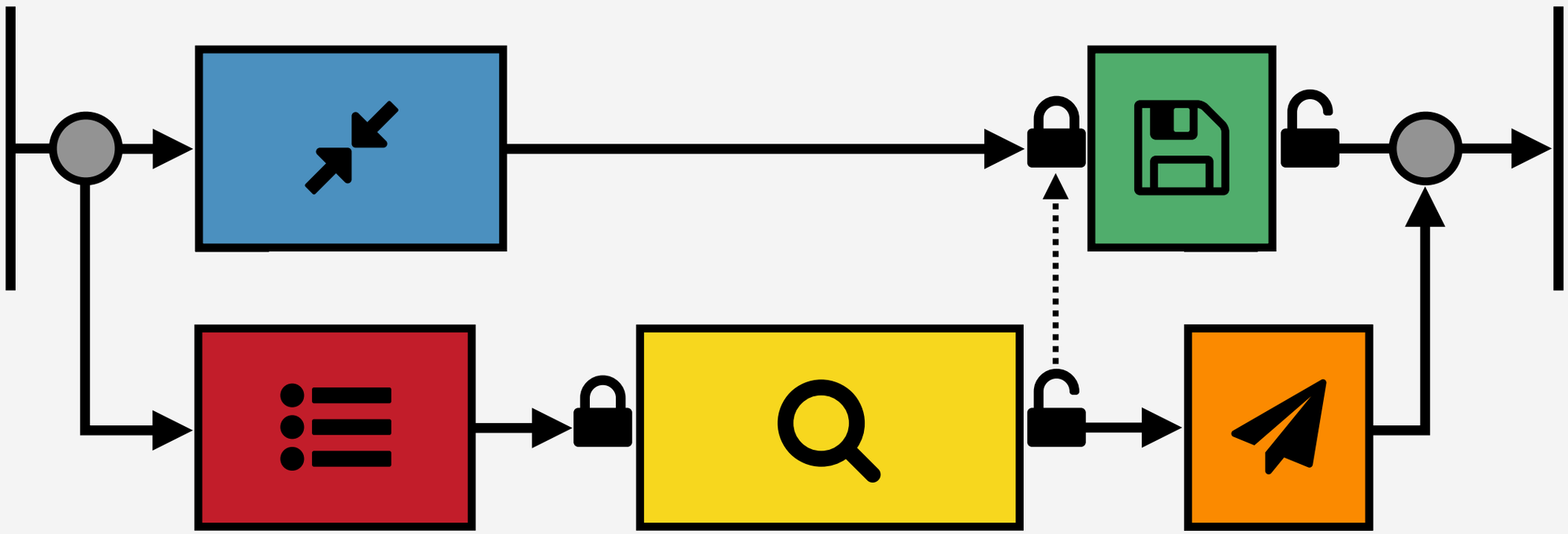


Speedup Results



Virtual Speedup

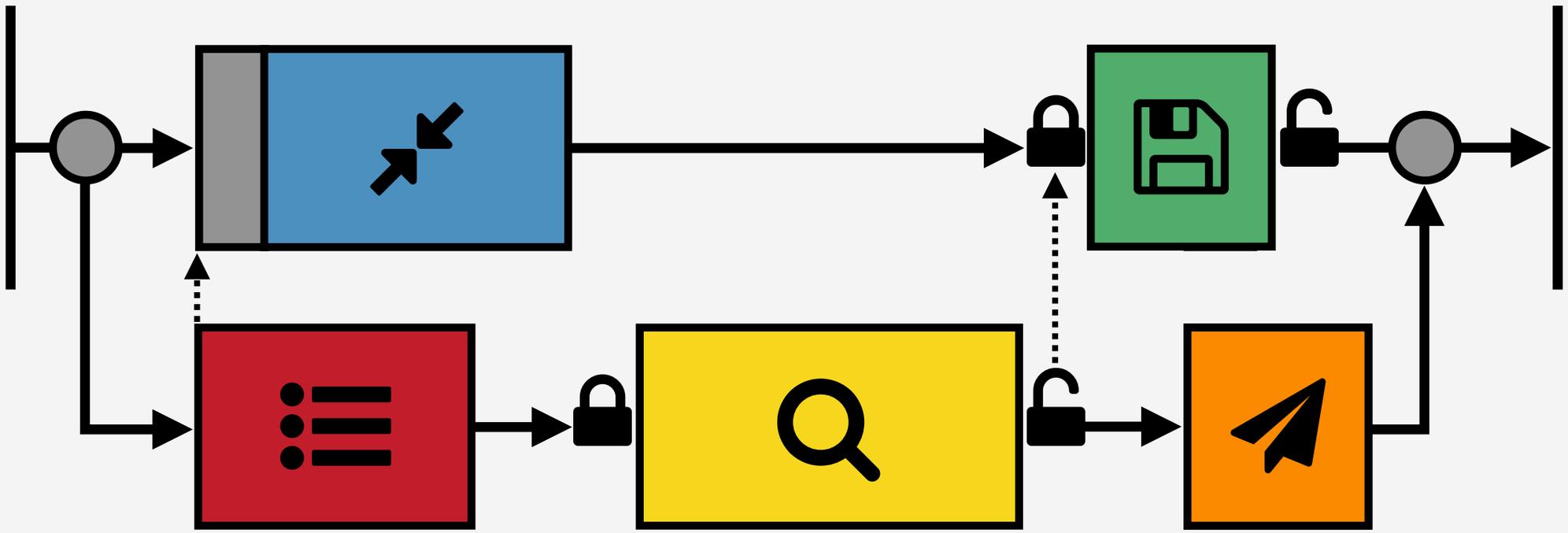
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

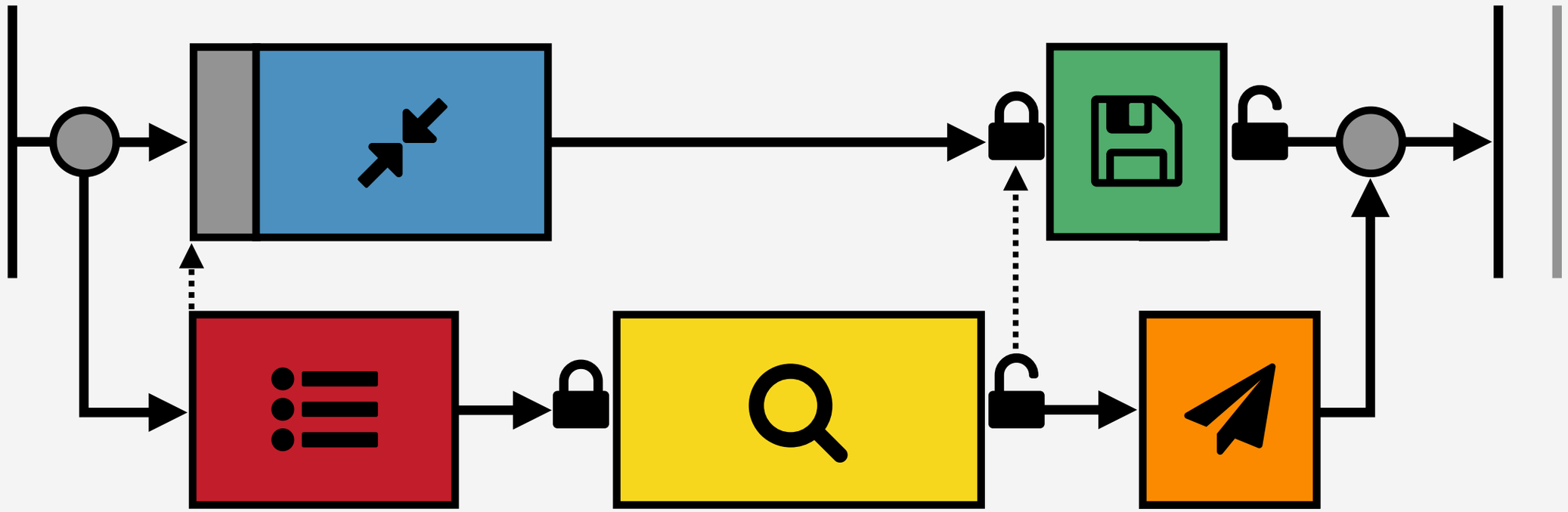
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

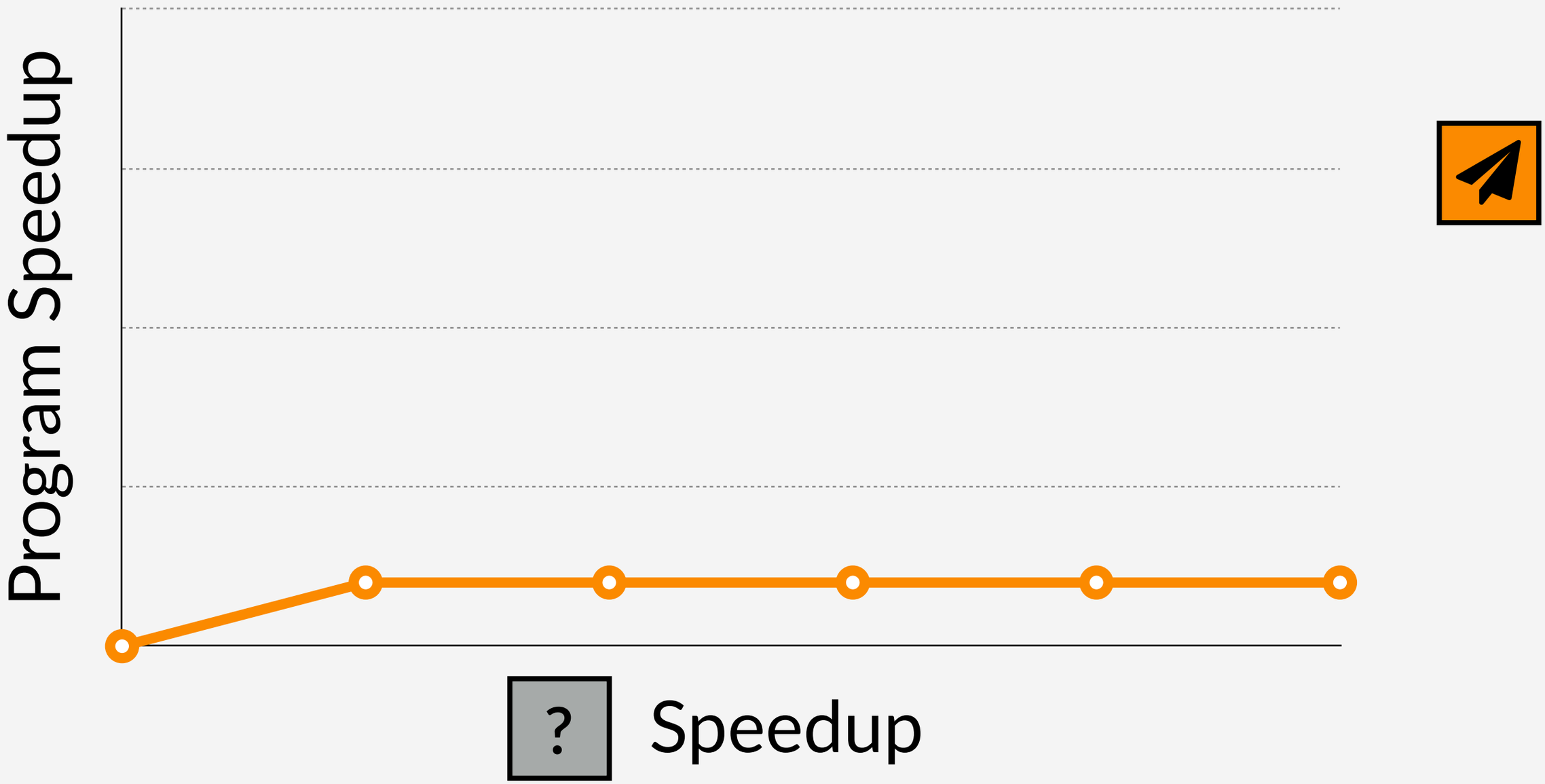
Virtual Speedup

“Speed up”  by slowing everything else down.

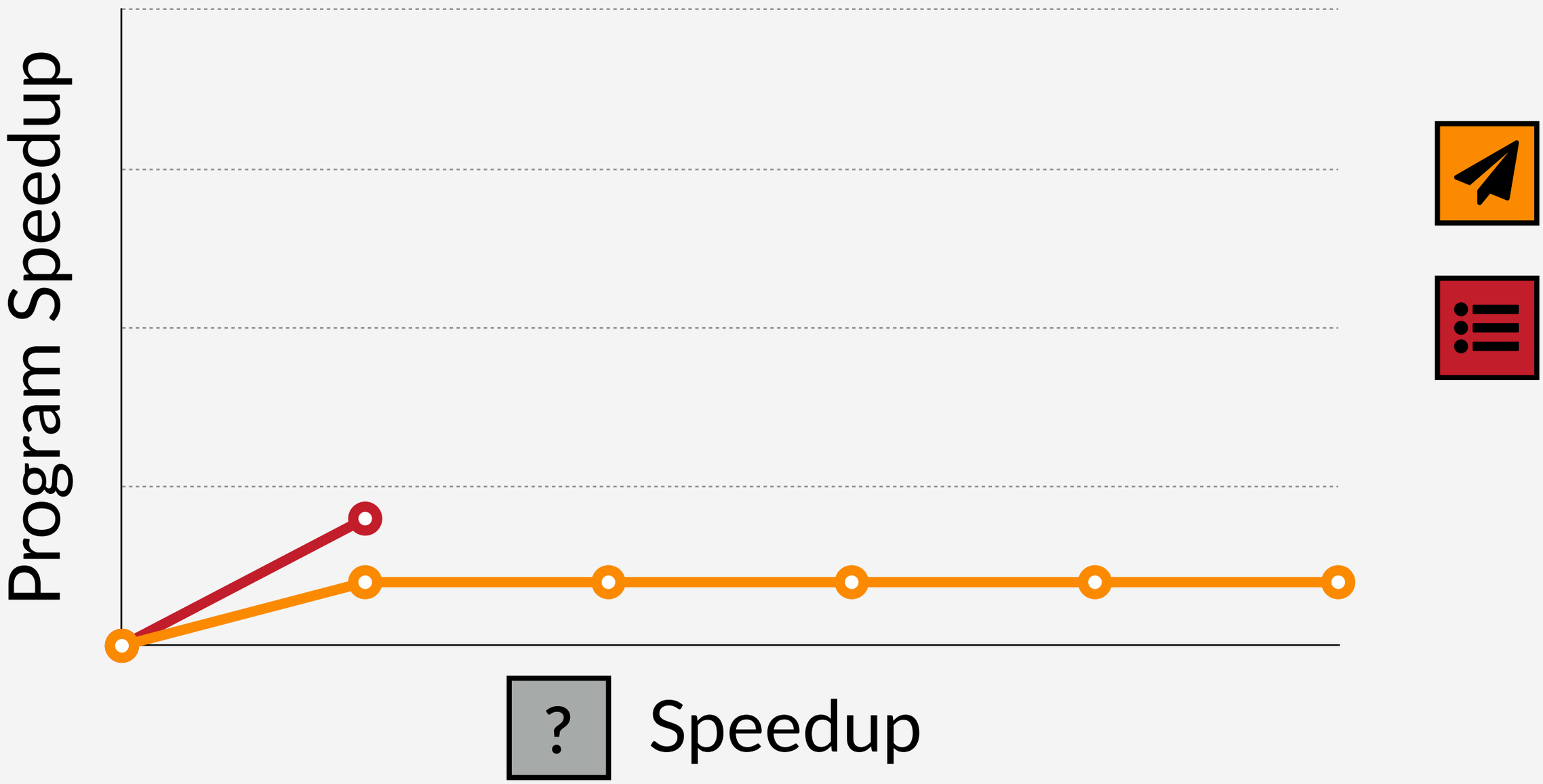


Each time  runs, pause all other threads.

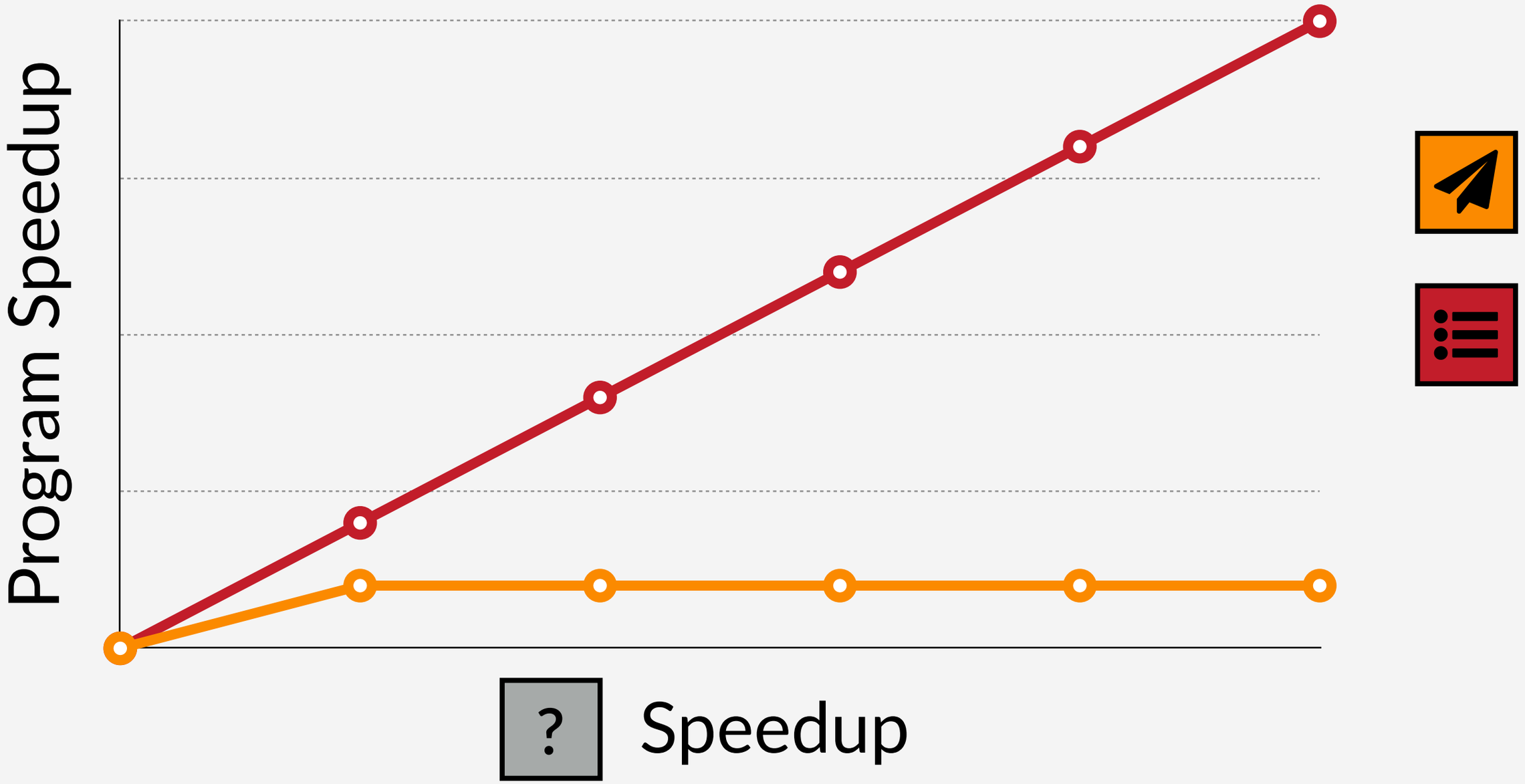
Speedup Results



Speedup Results

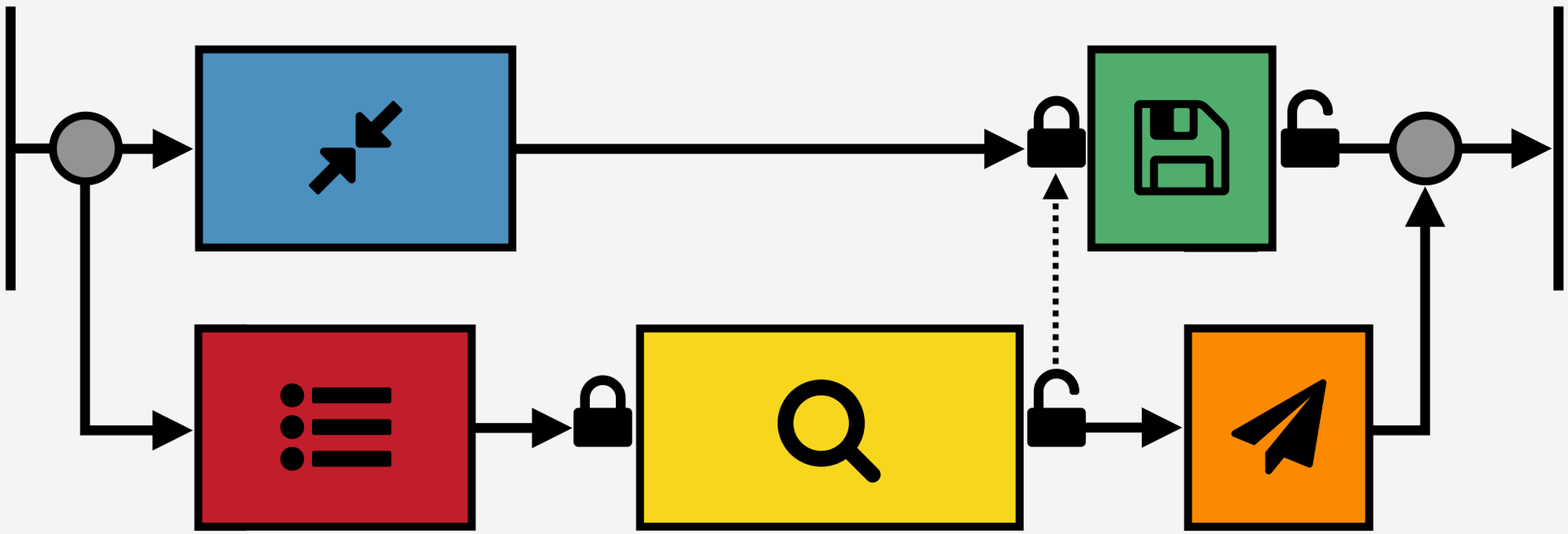


Speedup Results



Virtual Speedup

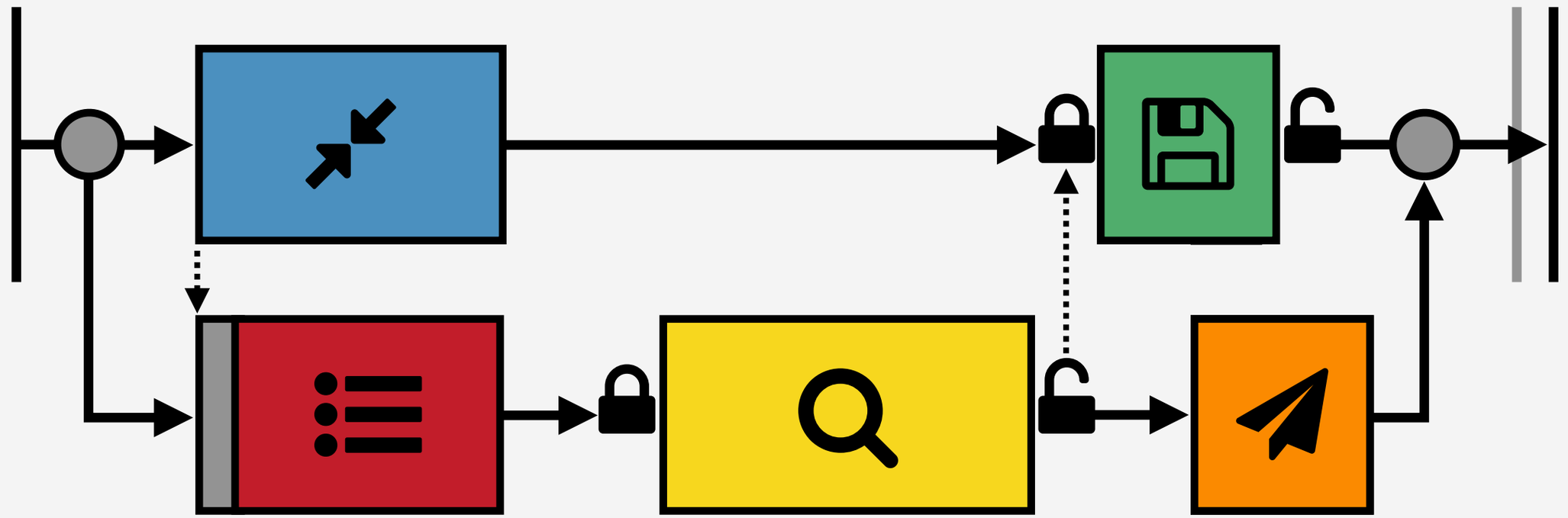
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

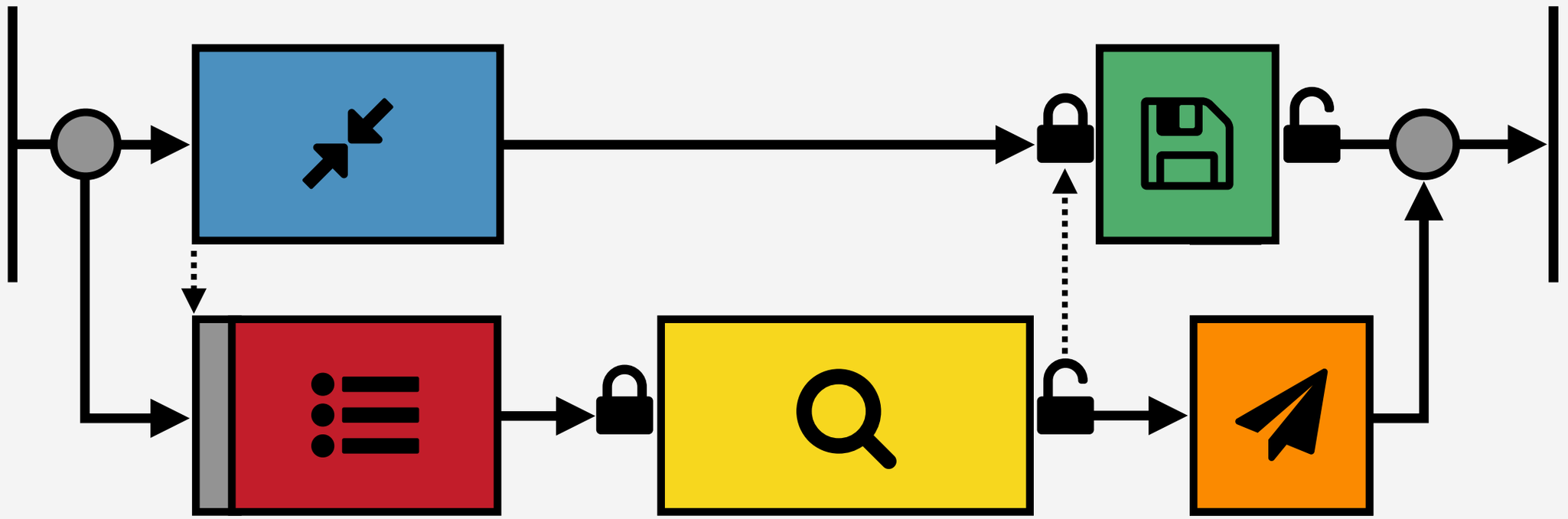
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

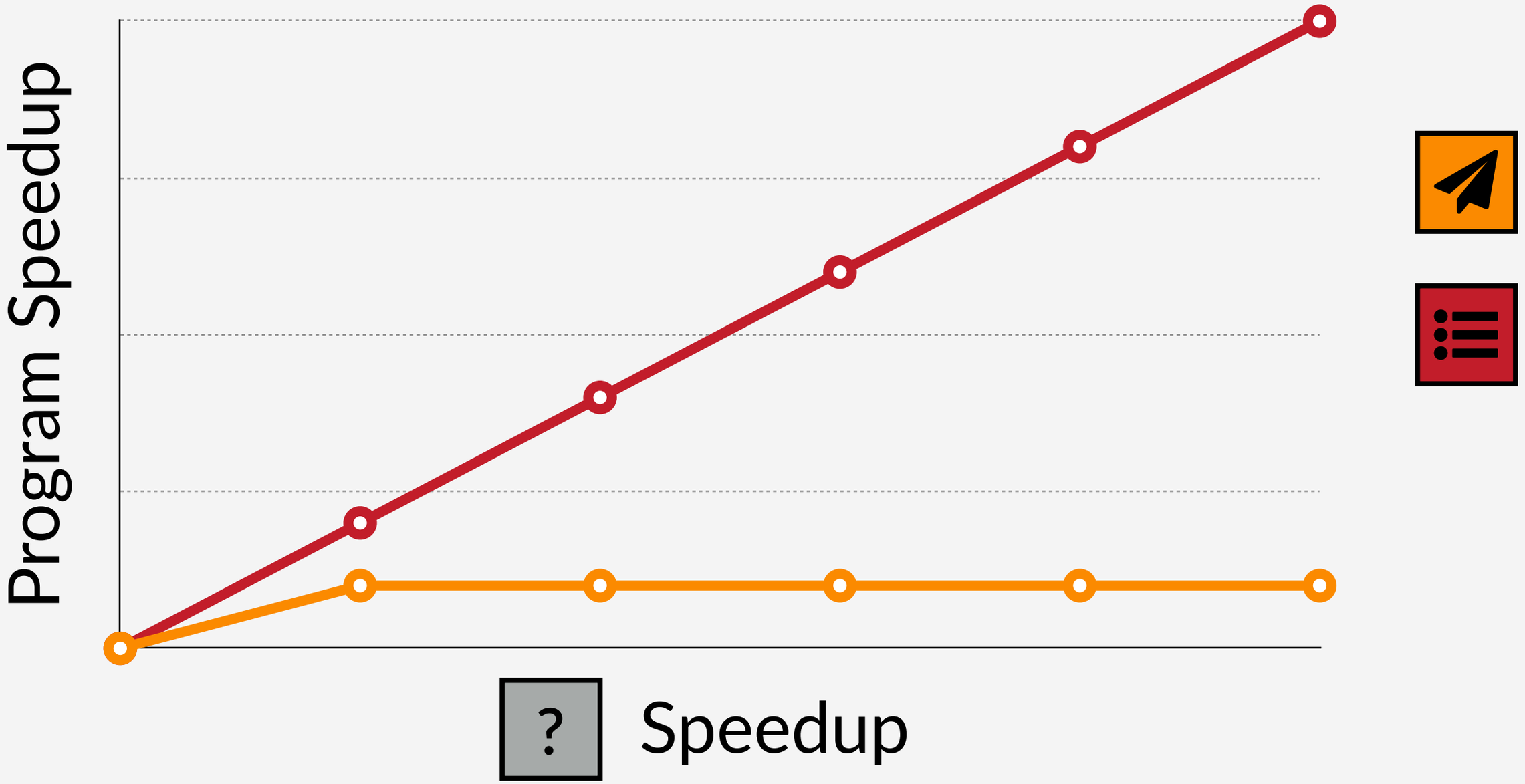
Virtual Speedup

“Speed up”  by slowing everything else down.

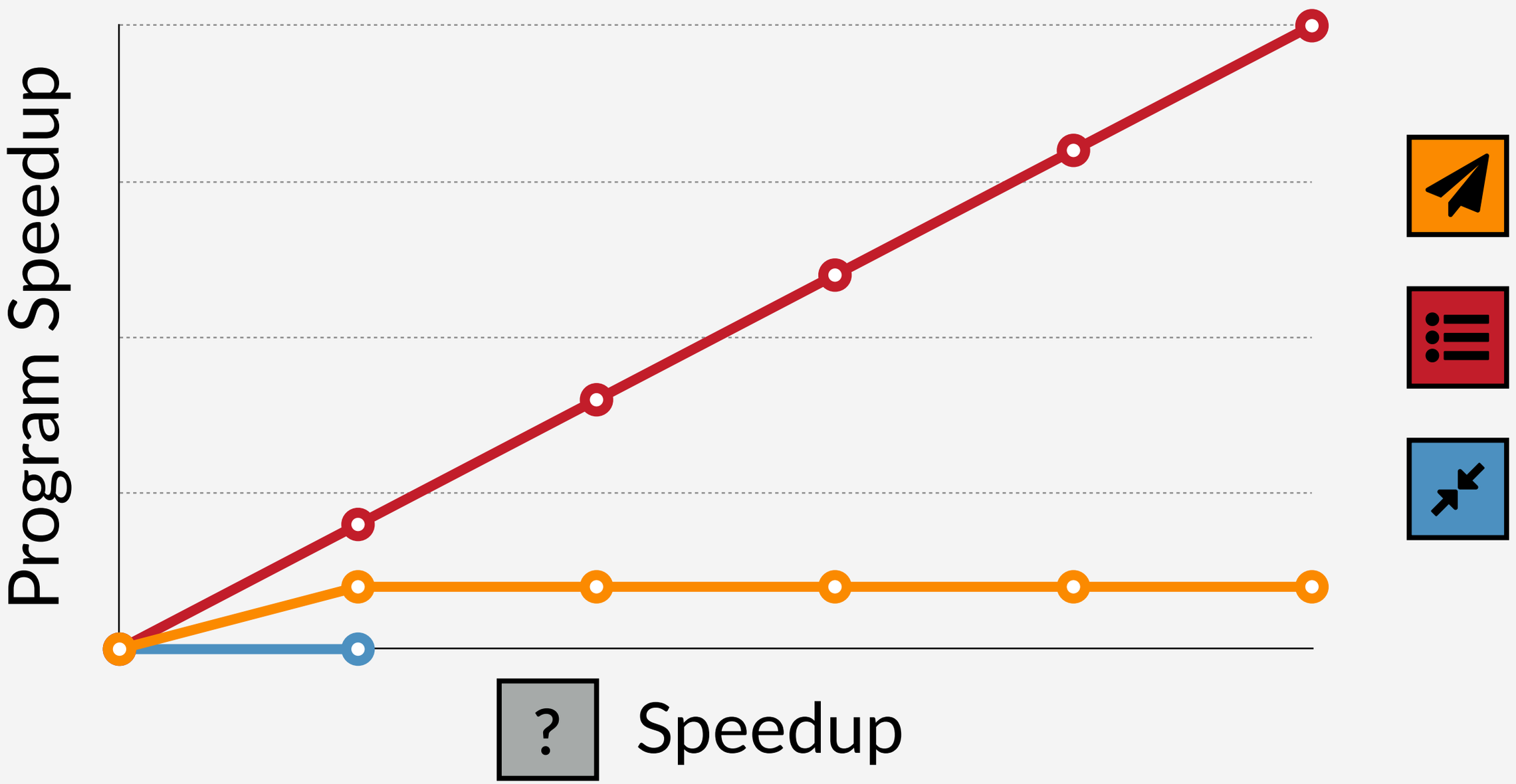


Each time  runs, pause all other threads.

Speedup Results

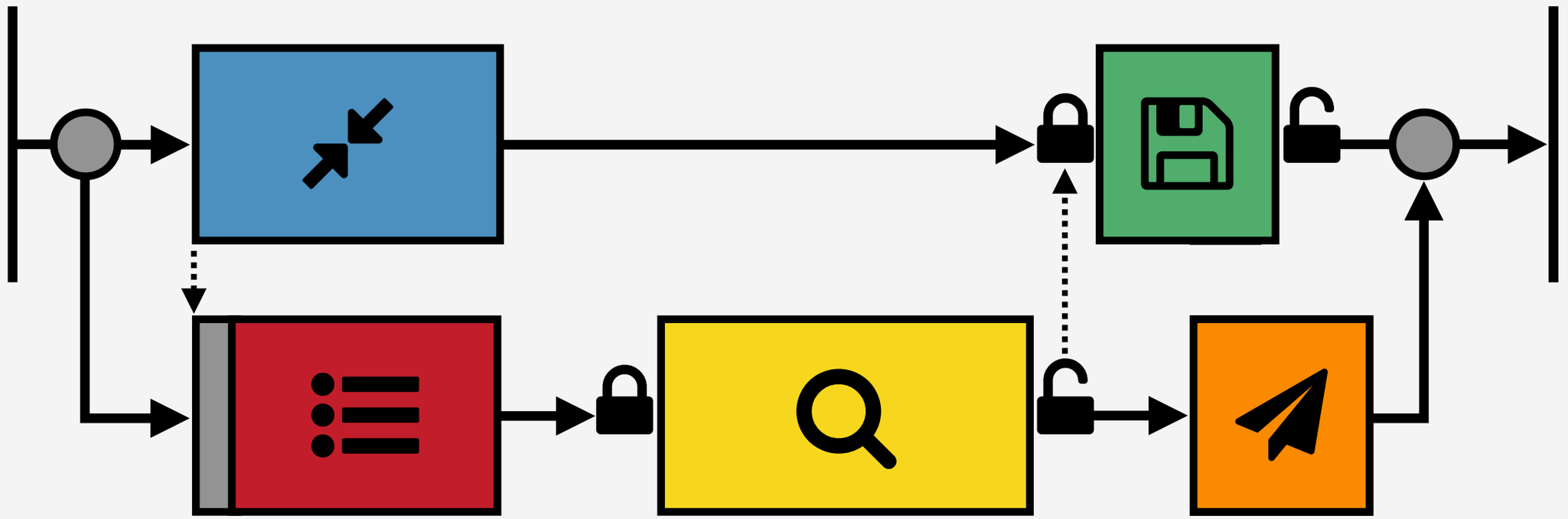


Speedup Results



Virtual Speedup

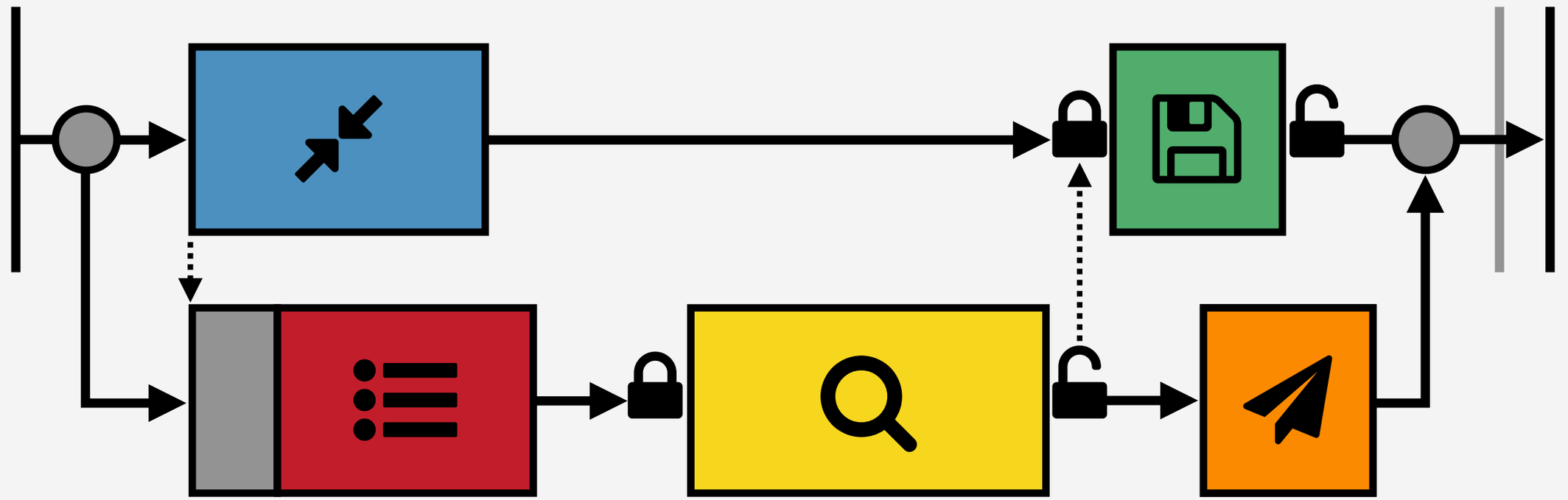
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

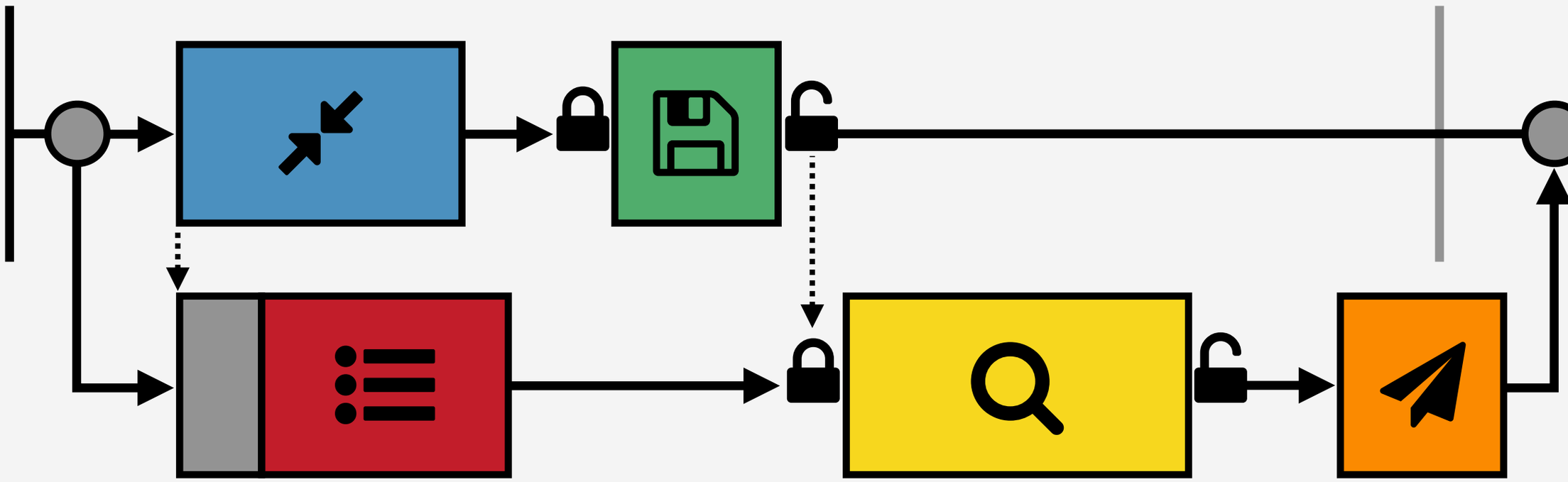
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

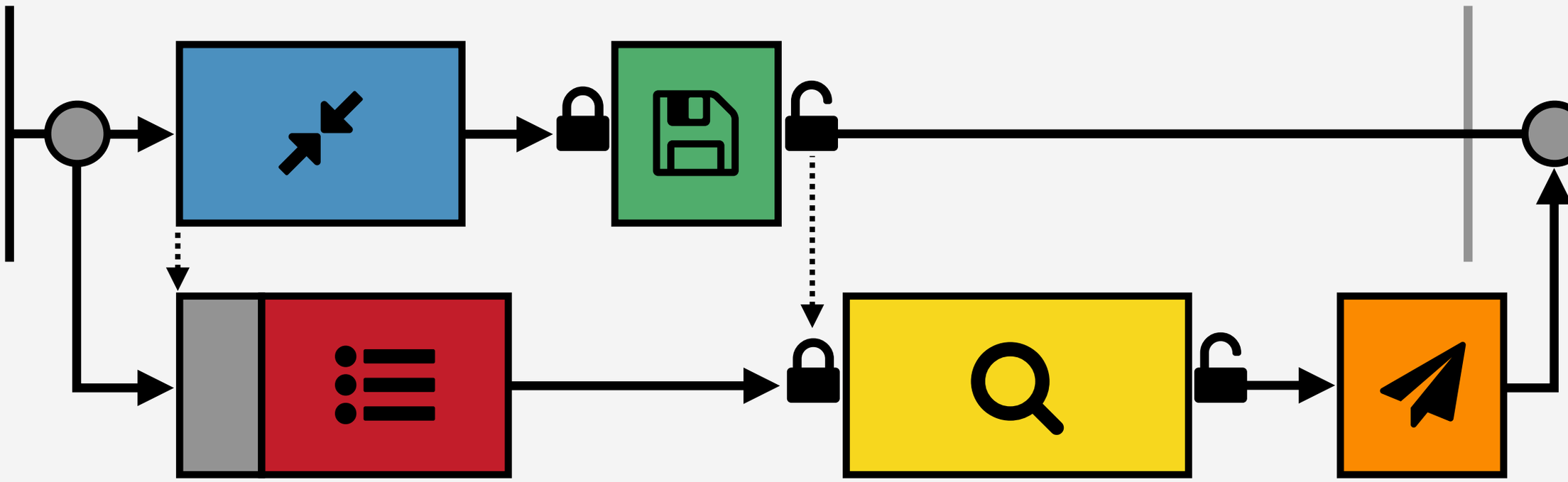
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

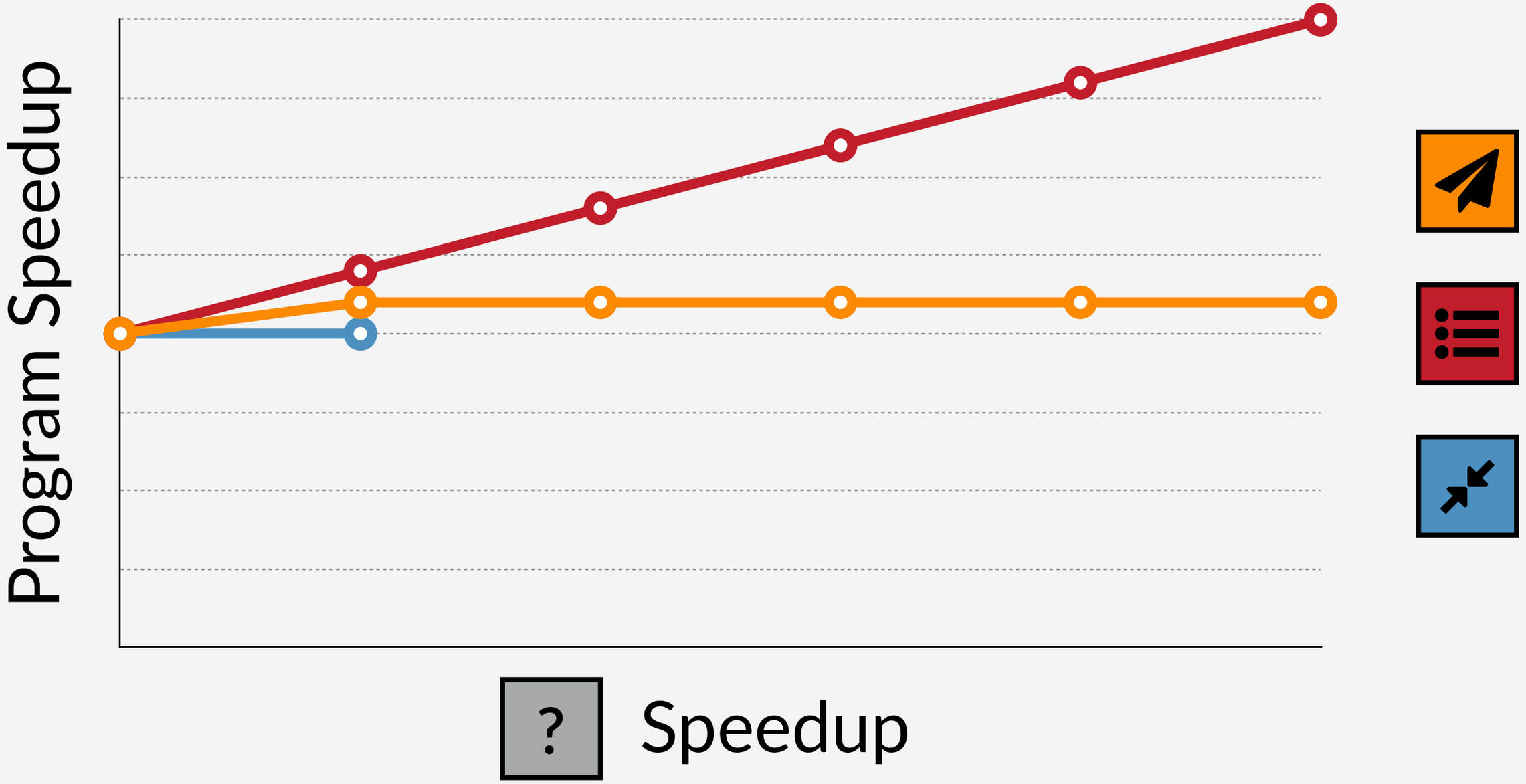
Virtual Speedup

“Speed up”  by slowing everything else down.



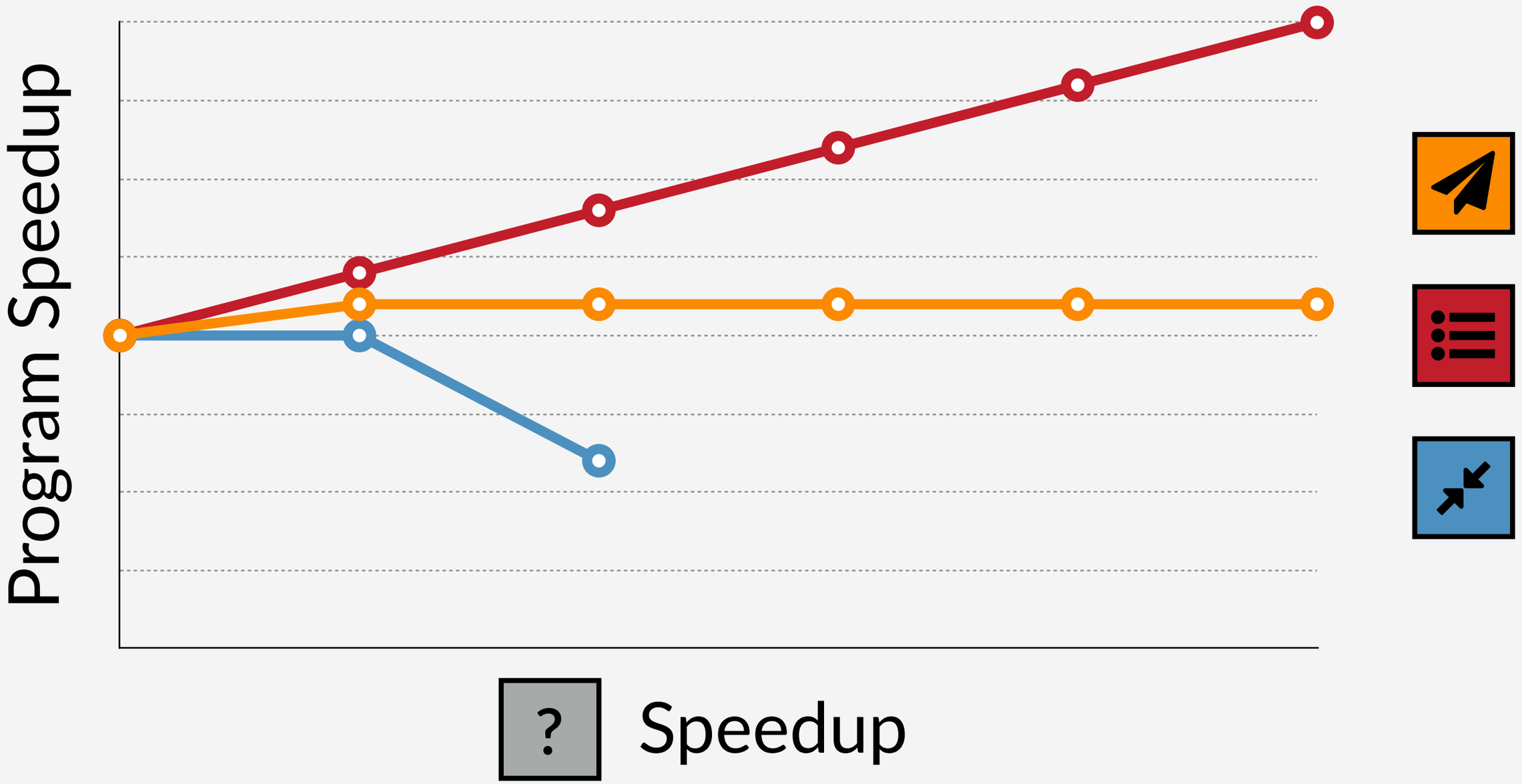
Each time  runs, pause all other threads.

Speedup Results



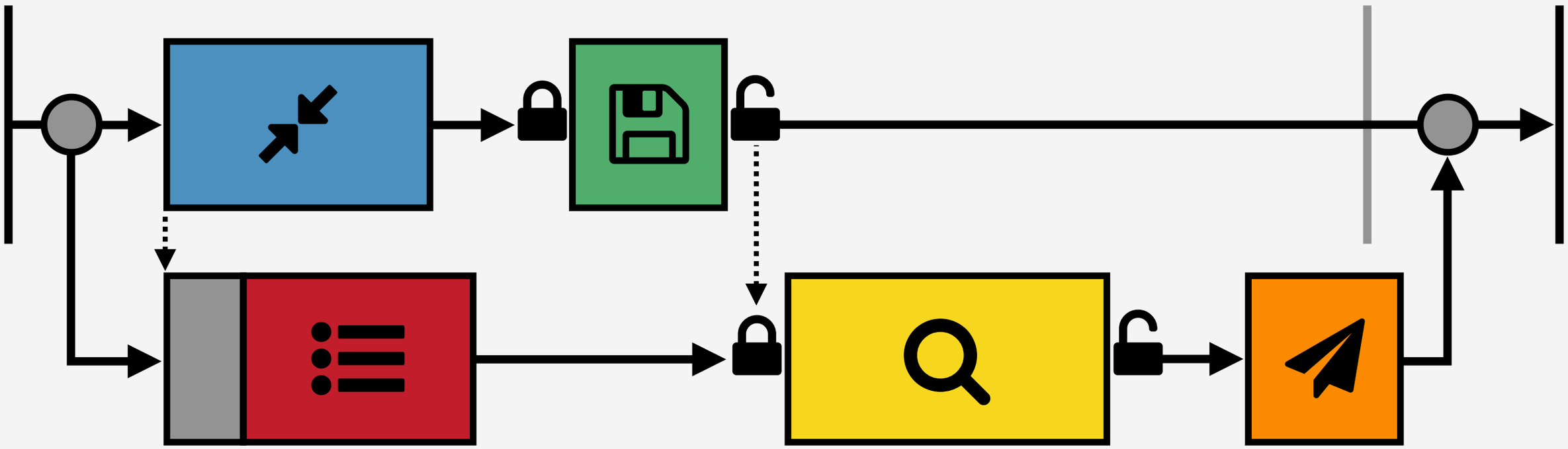
Speedup Results

Speeding up [↙↗] slows the program down!



Virtual Speedup

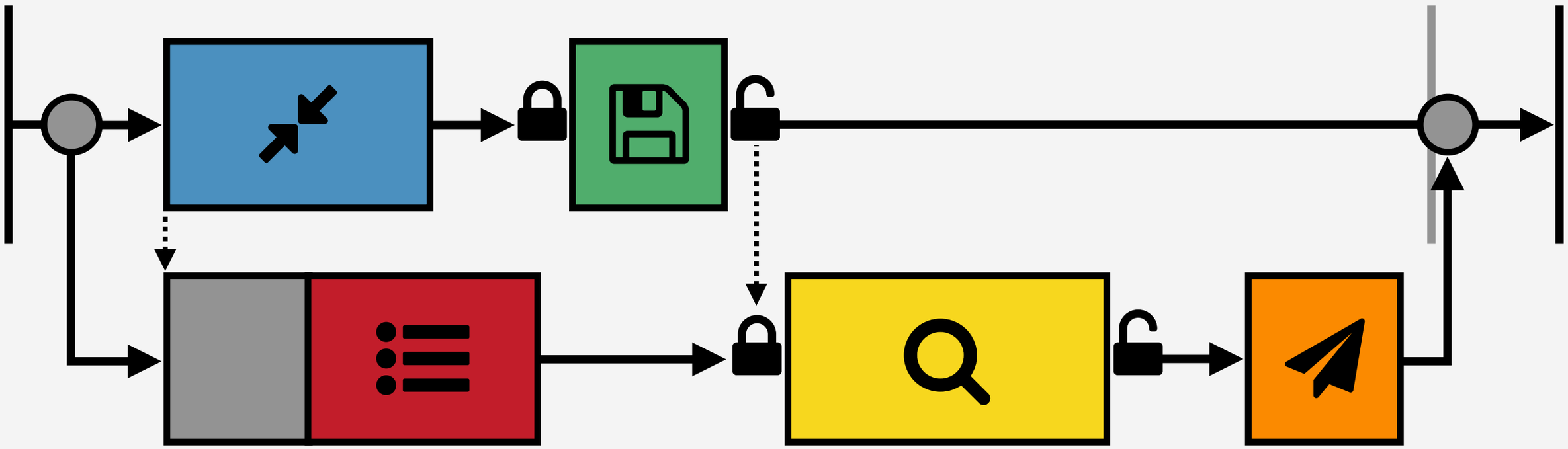
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

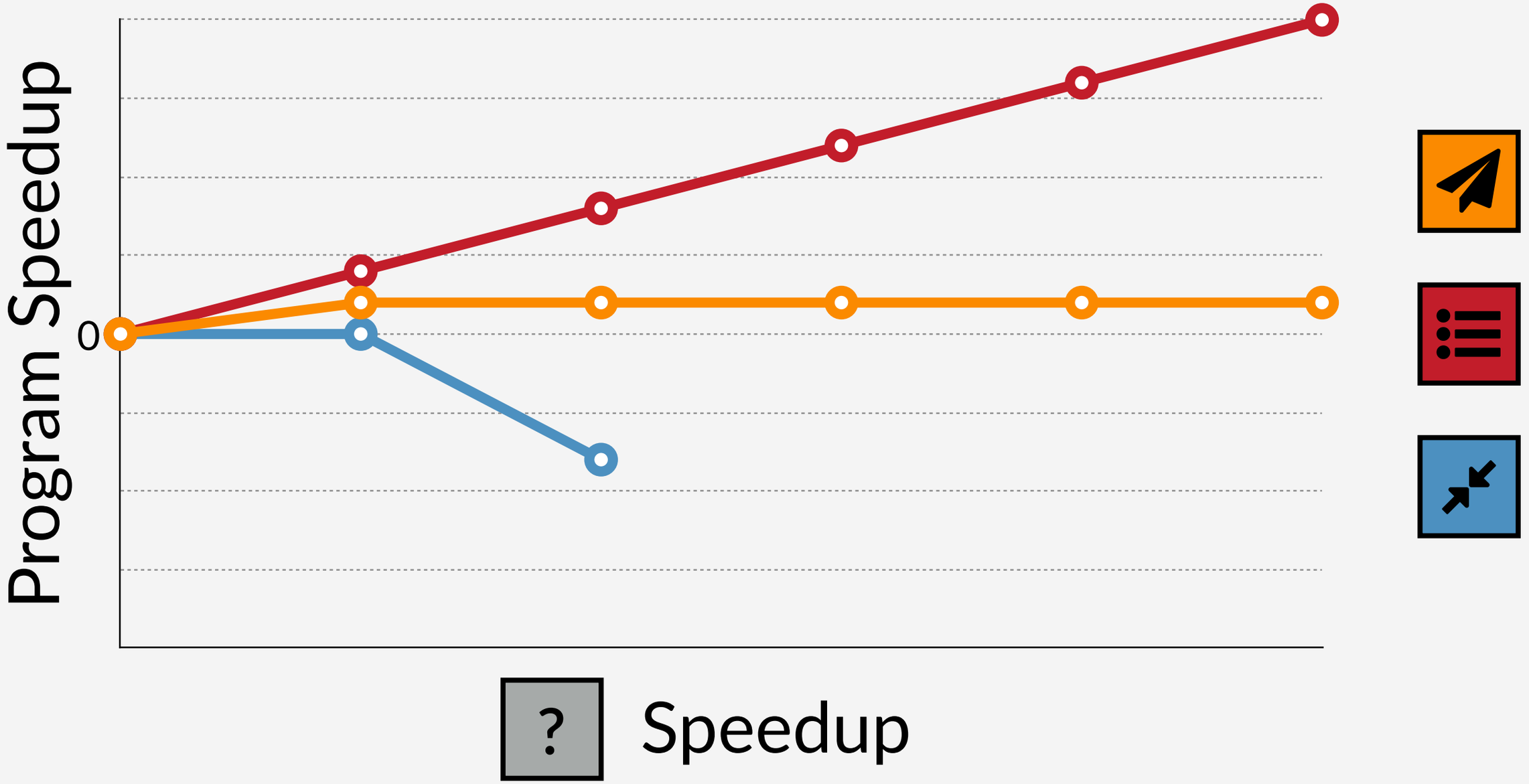
Virtual Speedup

“Speed up”  by slowing everything else down.

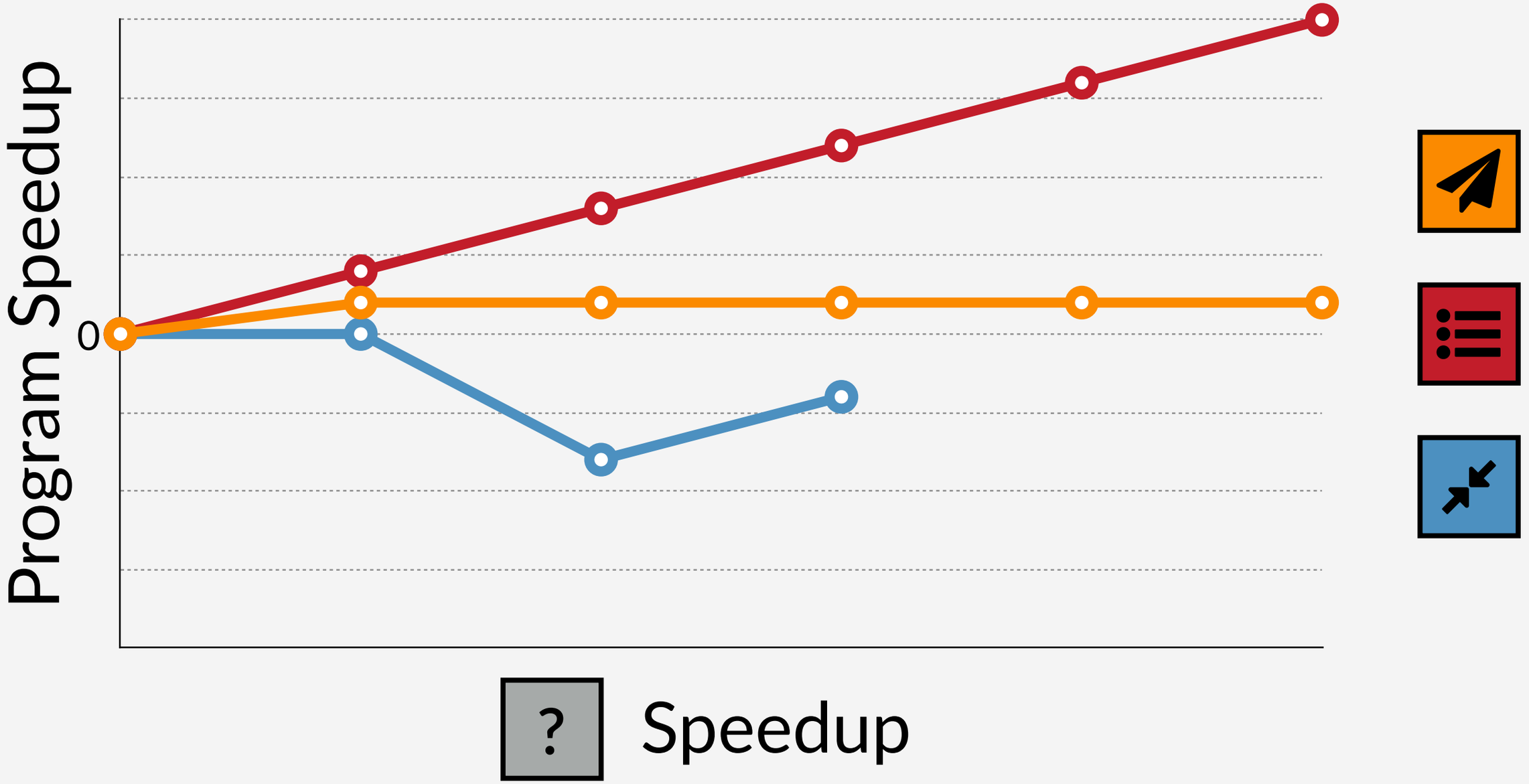


Each time  runs, pause all other threads.

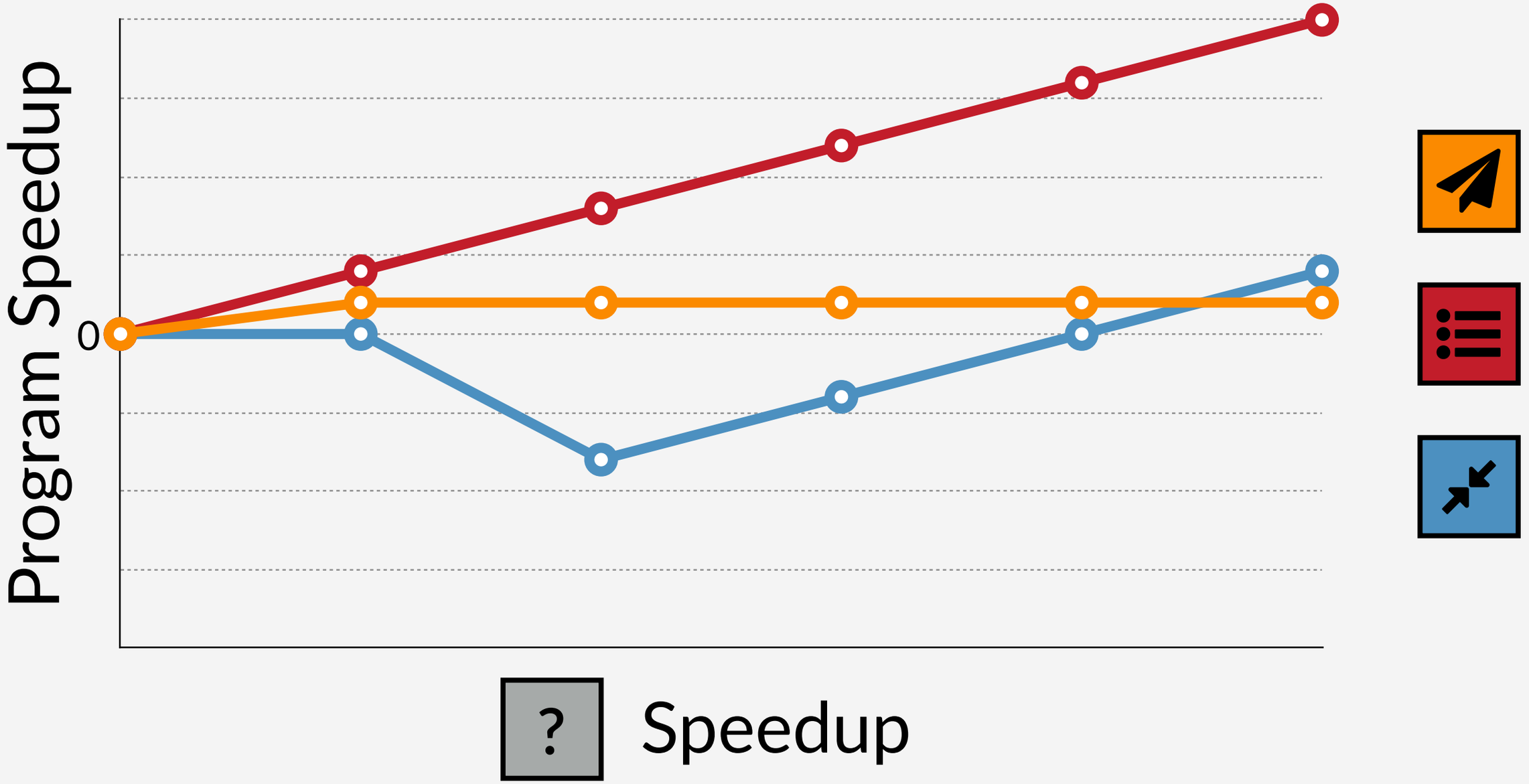
Speedup Results



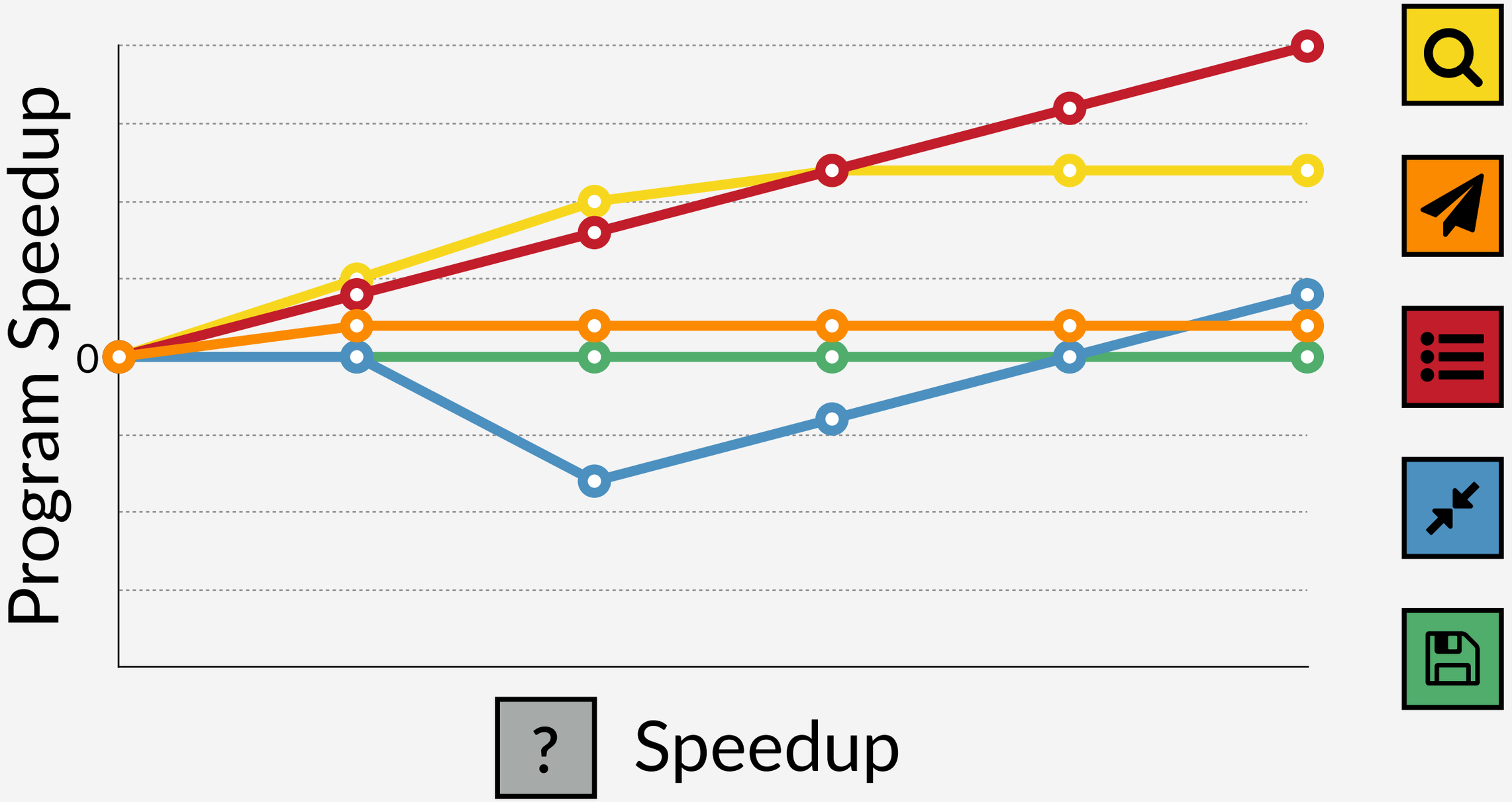
Speedup Results



Speedup Results



Speedup Results



Is runtime meaningful?

Take a picture

Send it to Ogle

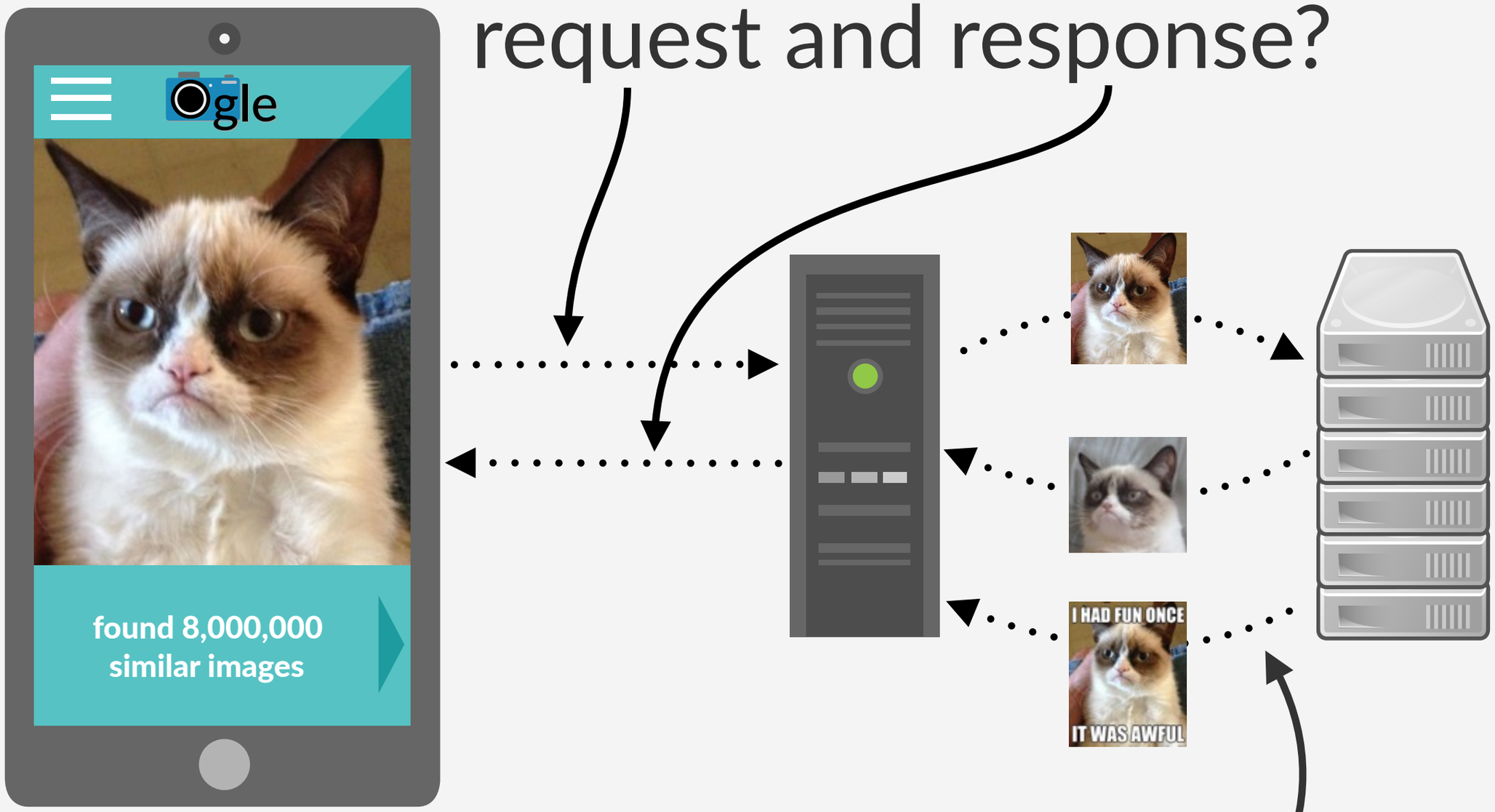
Add it to the database

found 8,000,000 similar images

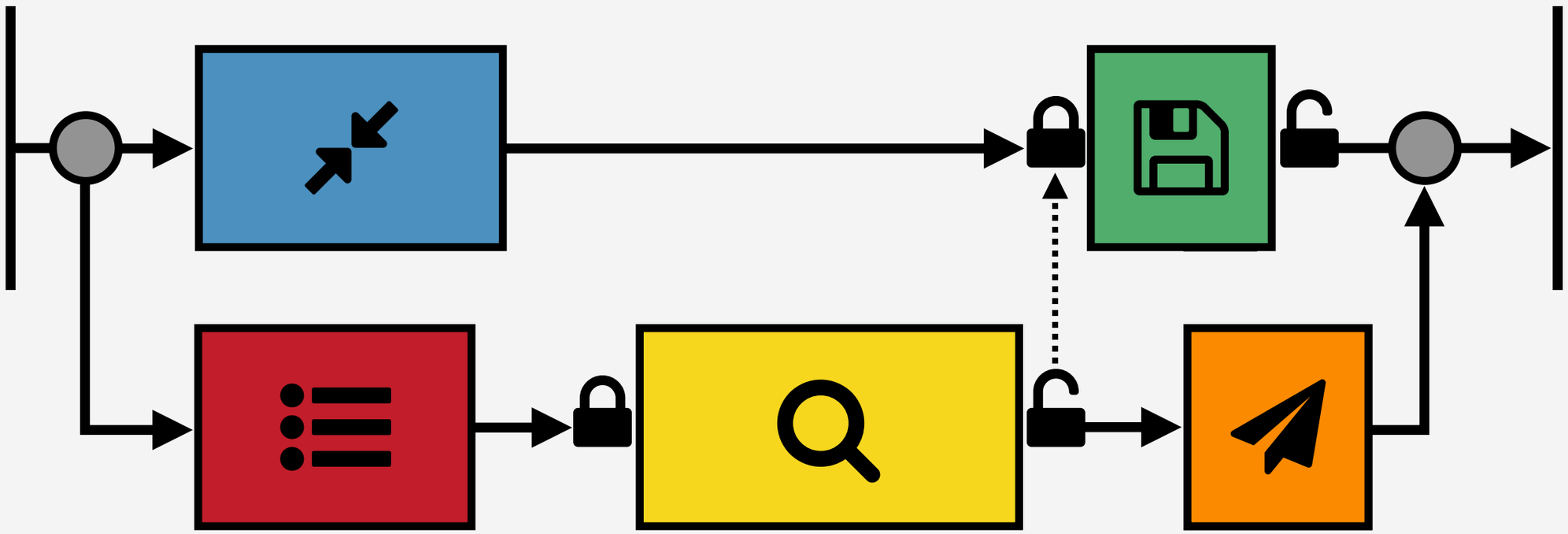
Send results

Find similar pictures

Is runtime meaningful?



Progress Points



Bob wants to send responses faster.

He marks  as a *progress point*.

Each time this code runs…

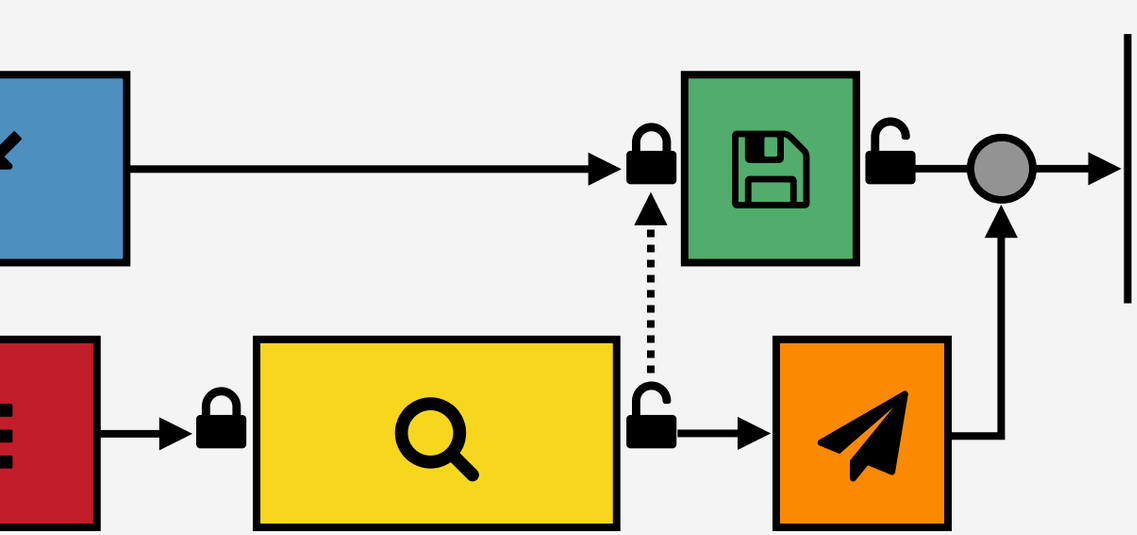
Progress Points

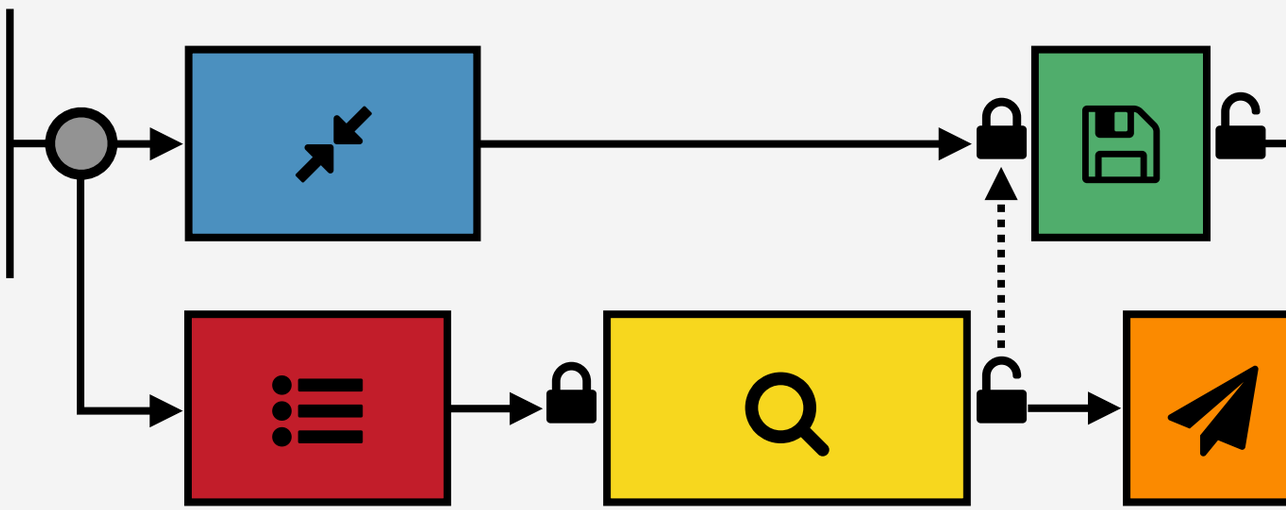
Many requests running for many users.

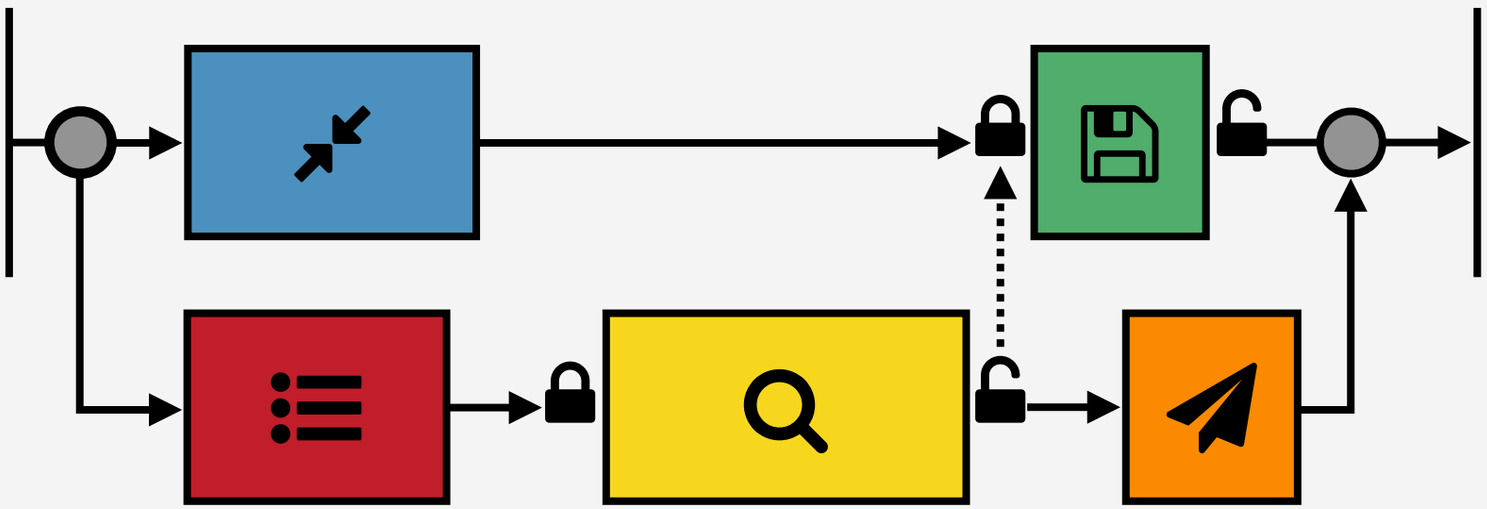
Progress Points

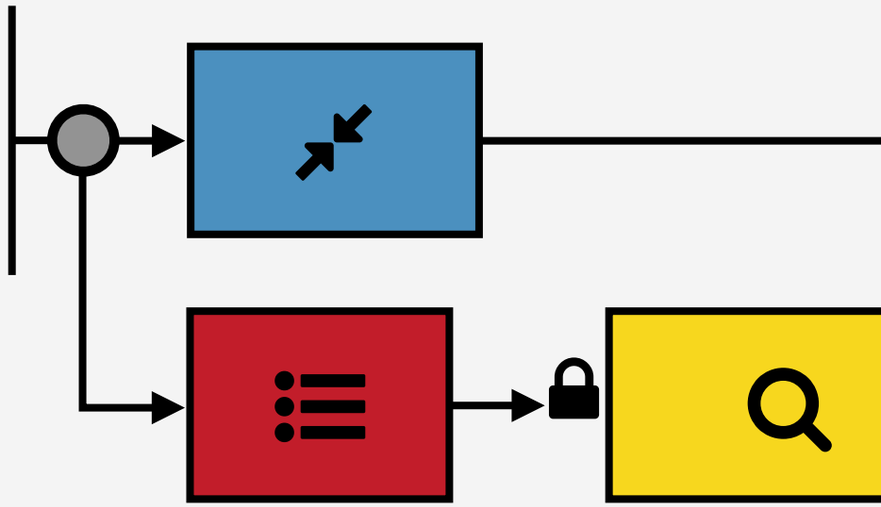
One progress point measures throughput.

If I speed up [?], how much faster do I run  ?

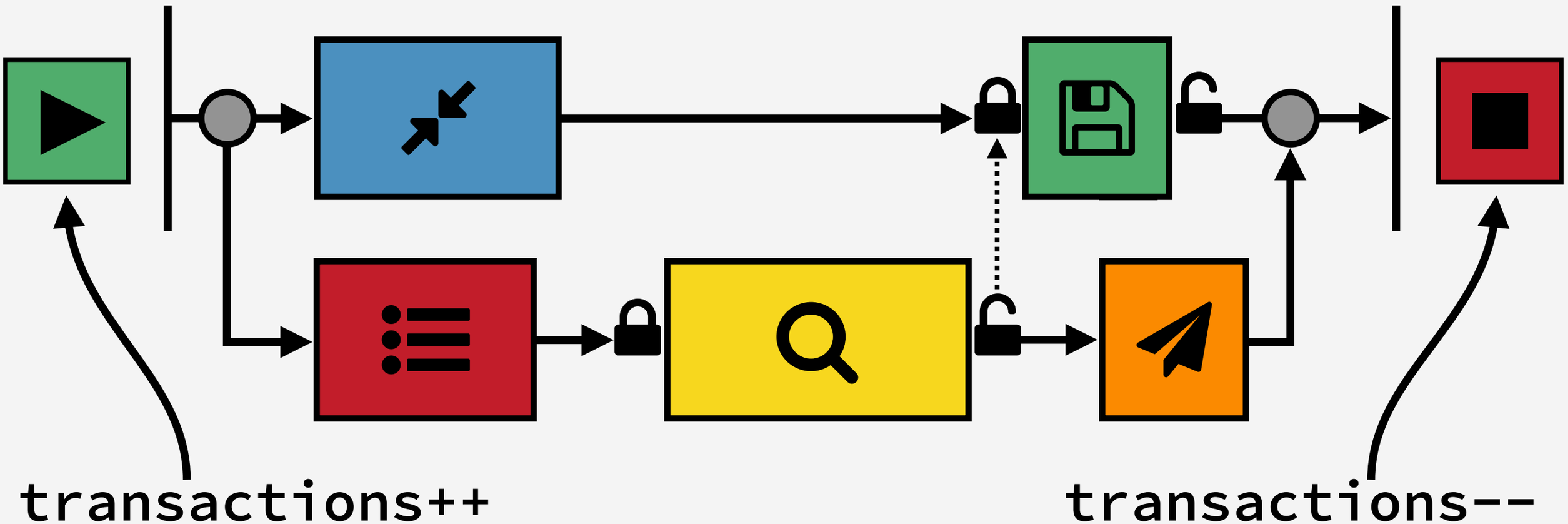








Progress Points



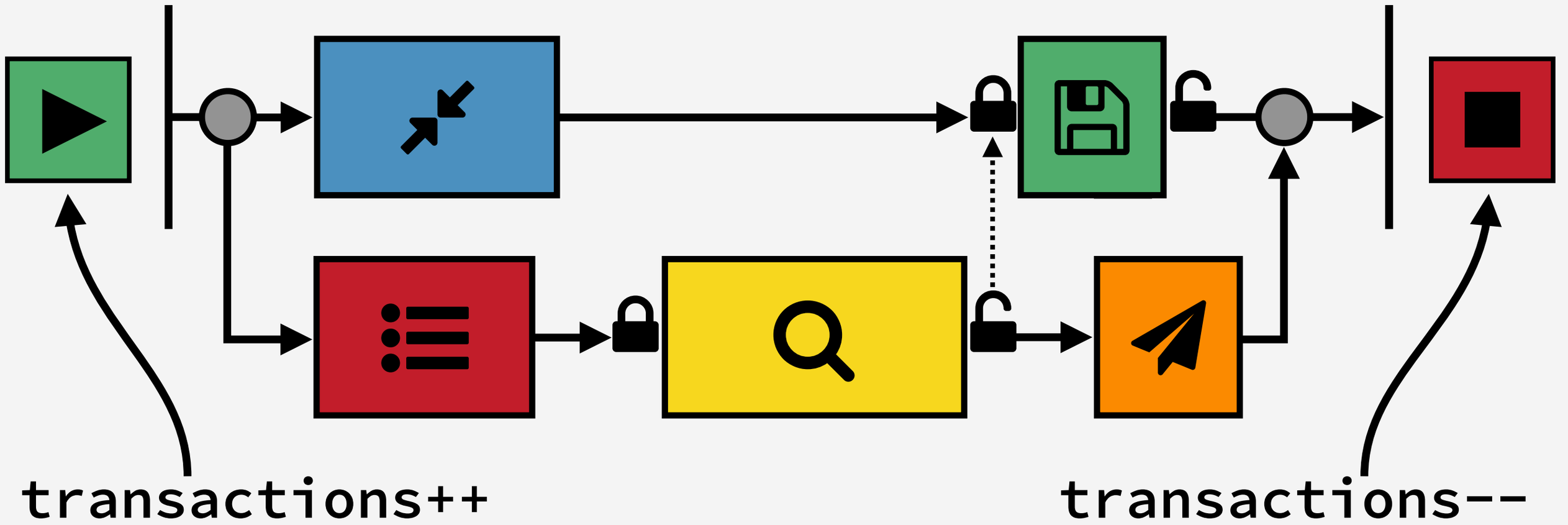
Bob wants to minimize response time.

He adds *latency progress points*.

Progress Points

Little's Law: W = L / λ

latency = `transactions` / throughput



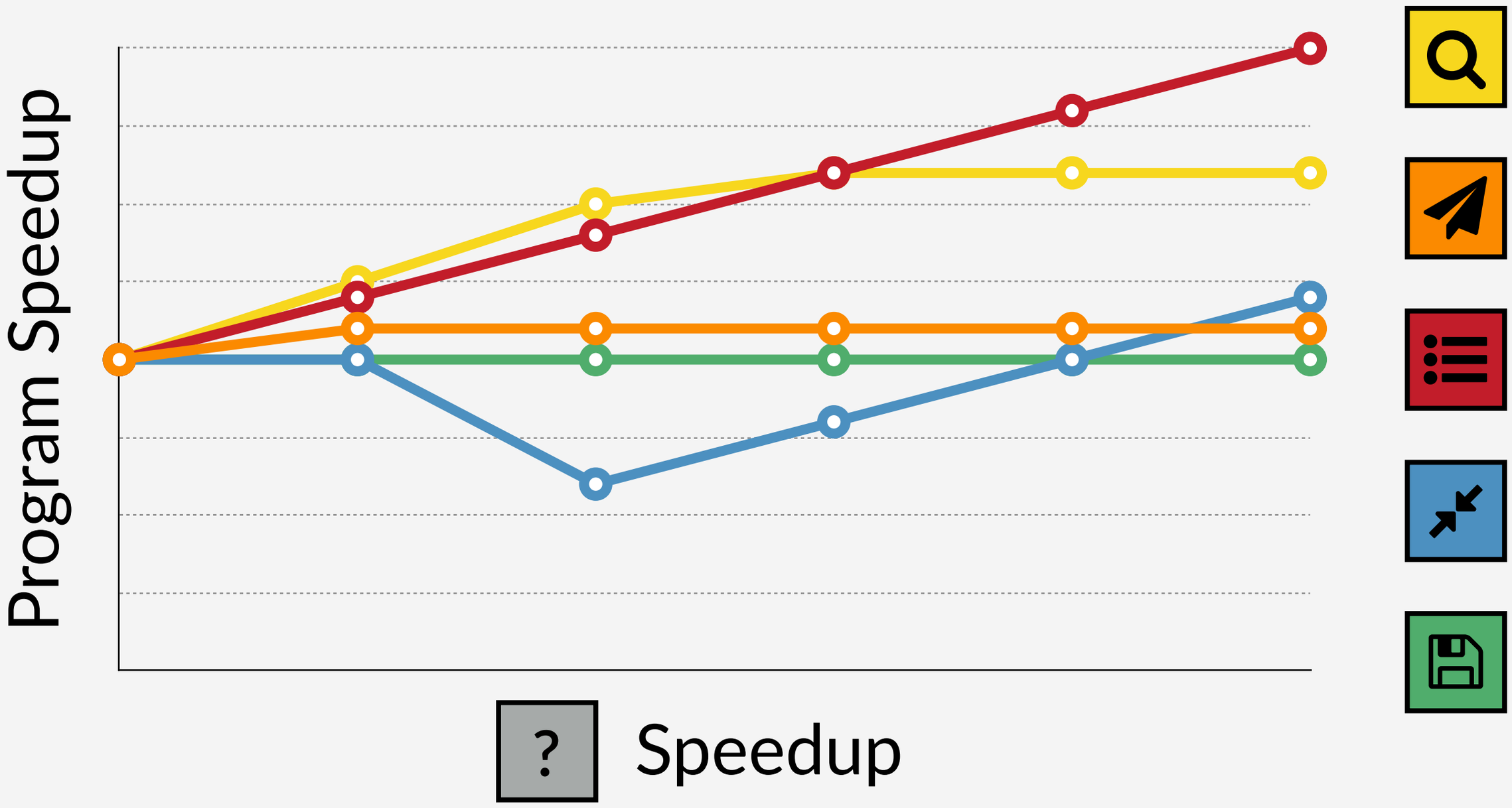
Bob wants to minimize response time.

Coz: a Causal Profiler for Linux

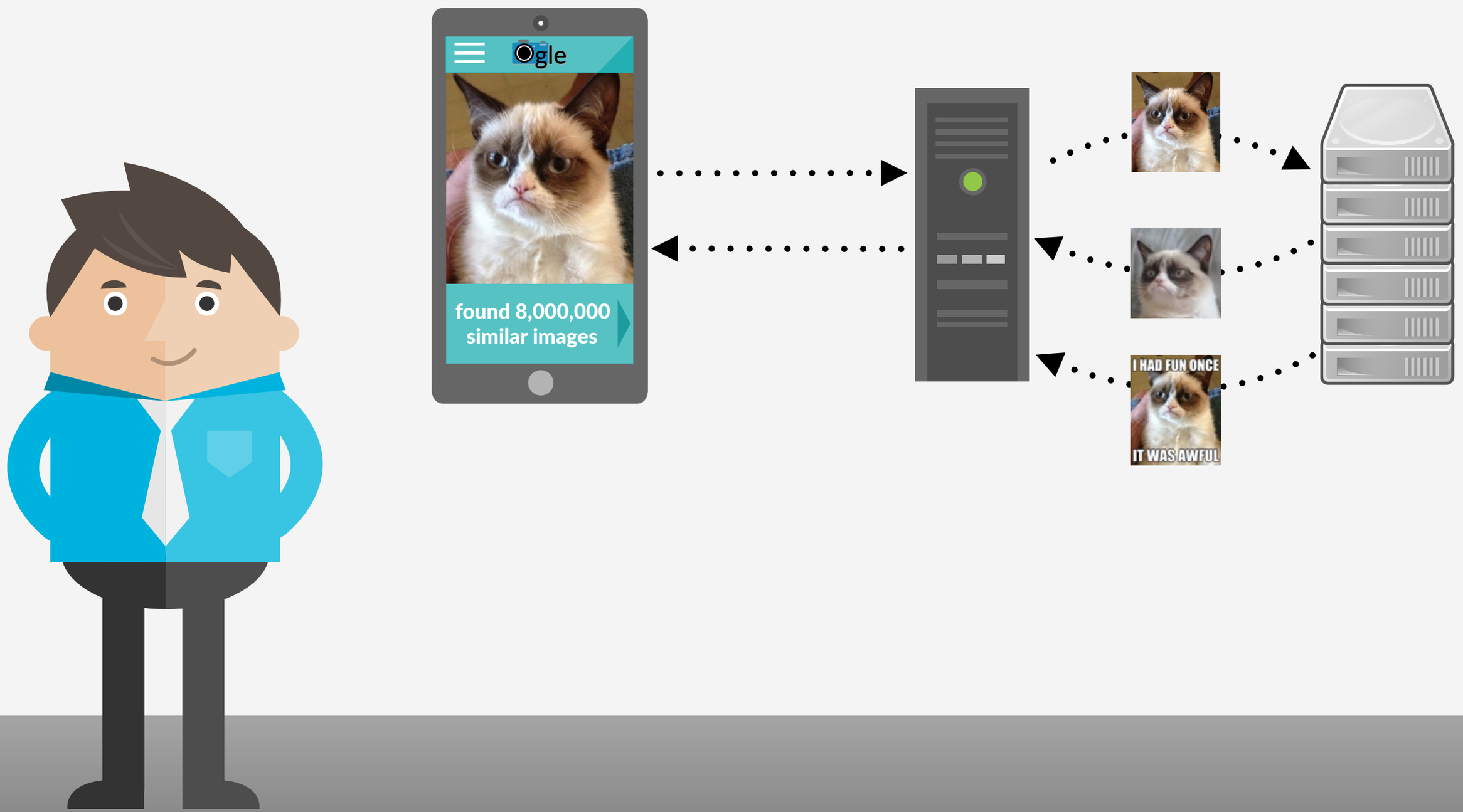
```
> coz run --- ./ogle_server deploy
```

Coz Produces Causal Profiles

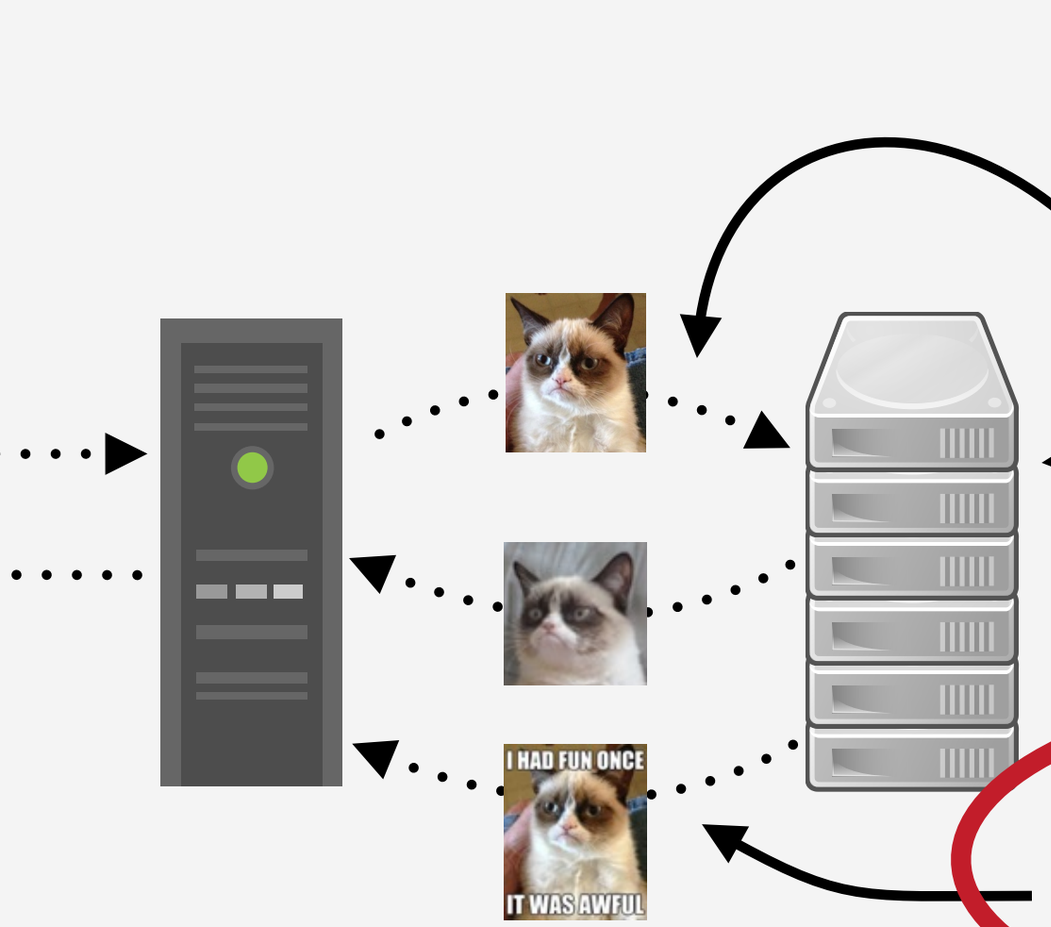
Let's use it to improve Ogle



Using Causal Profiling on Ogle



Using Causal Profiling on Ogle



SQLite

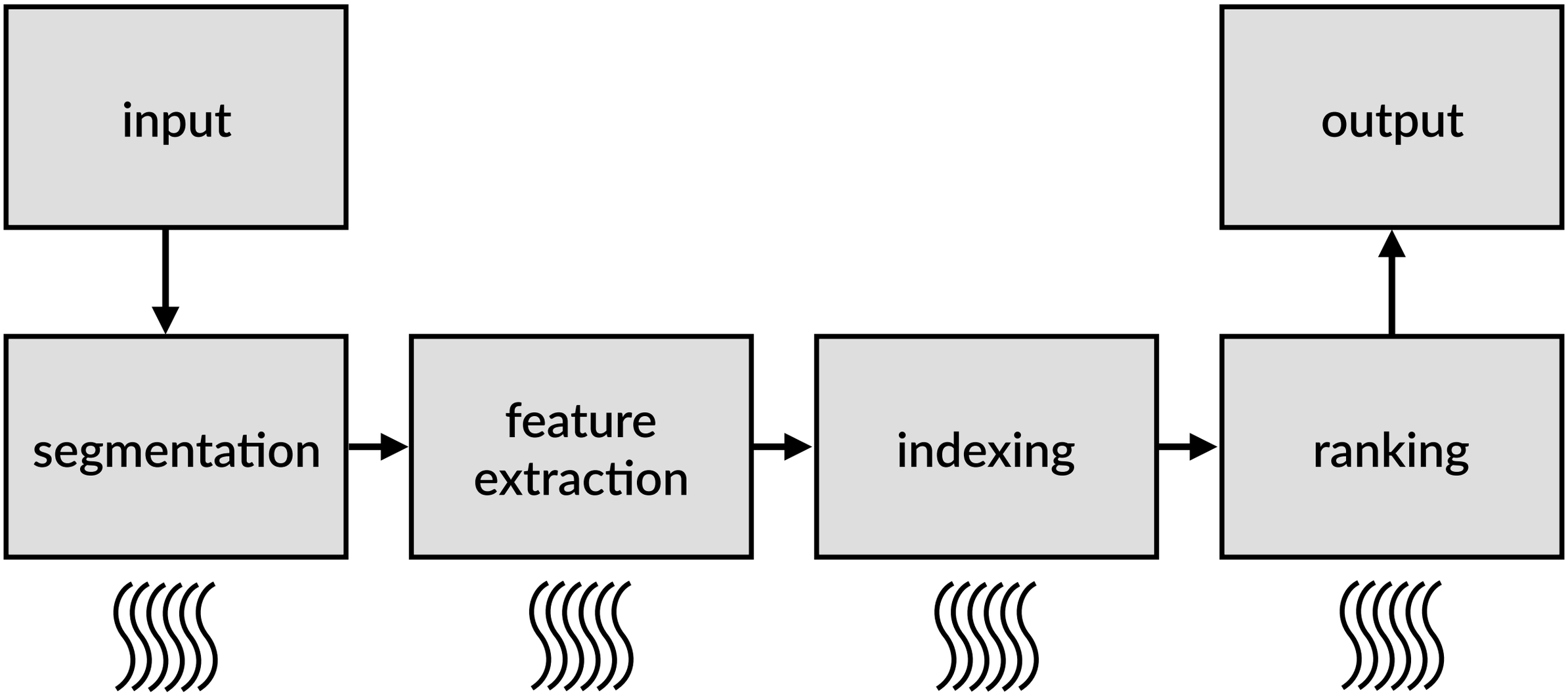
dedup
compression

ferret
image comparison

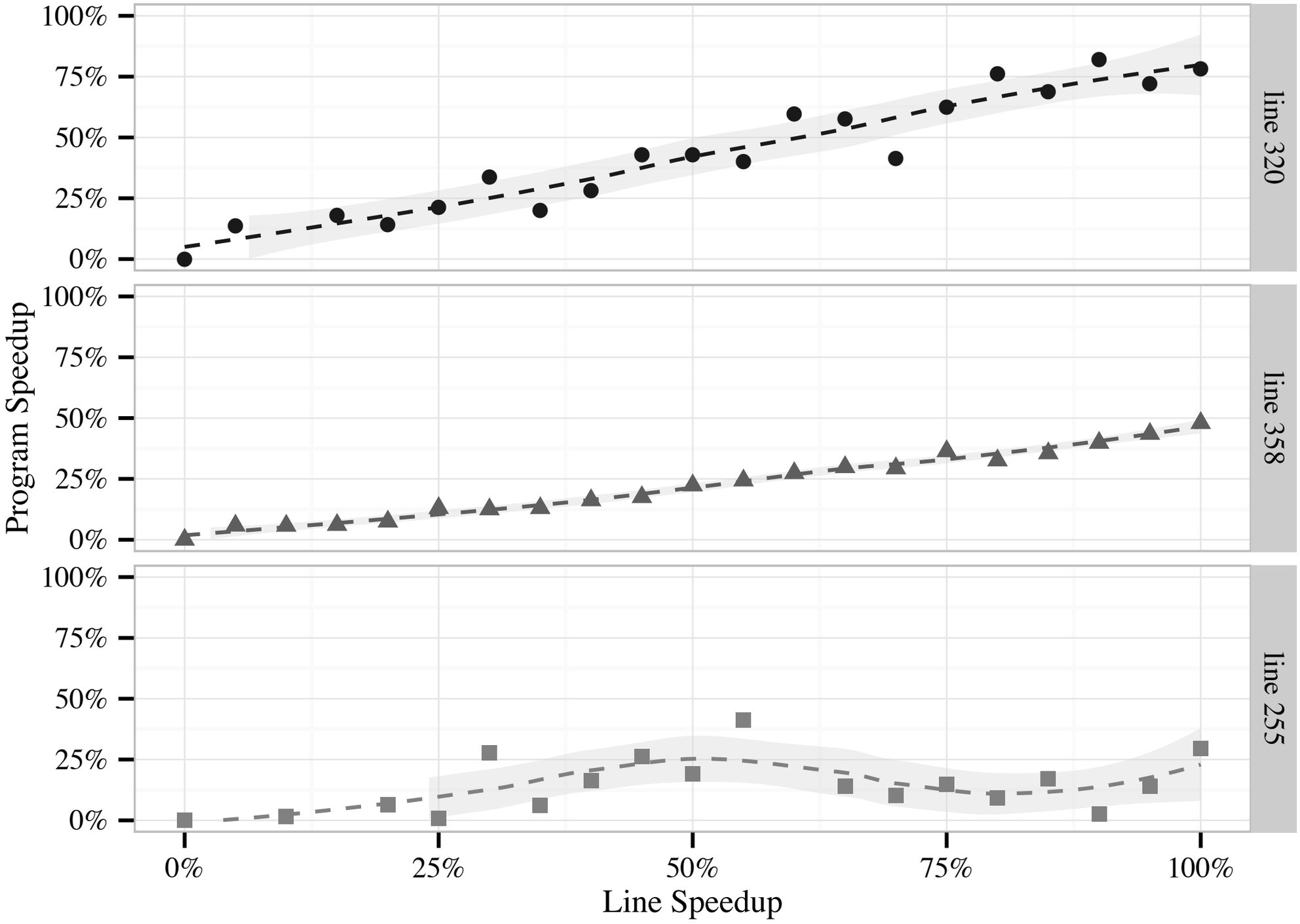
PARSEC
['pär-''sek] A unit of measure

Ferret

image comparison



Ferret

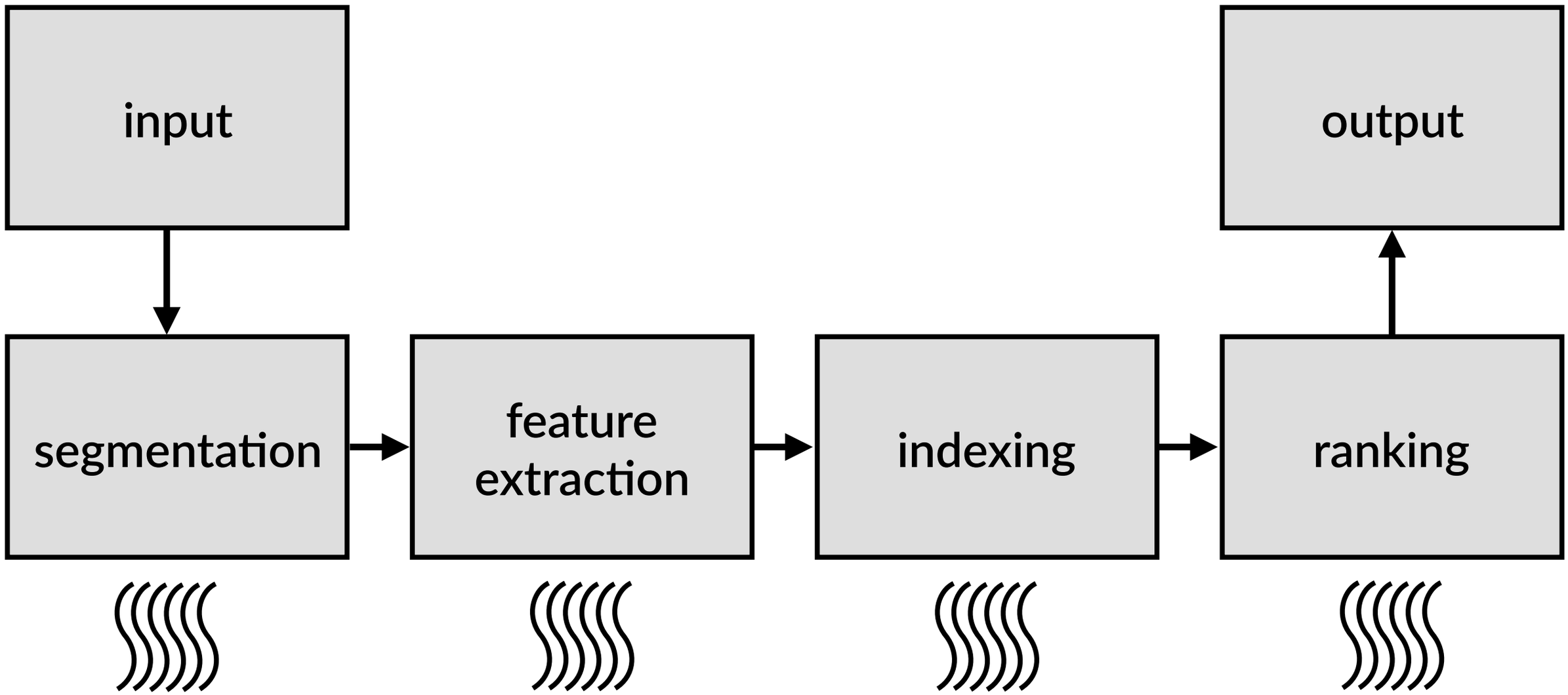


ranking

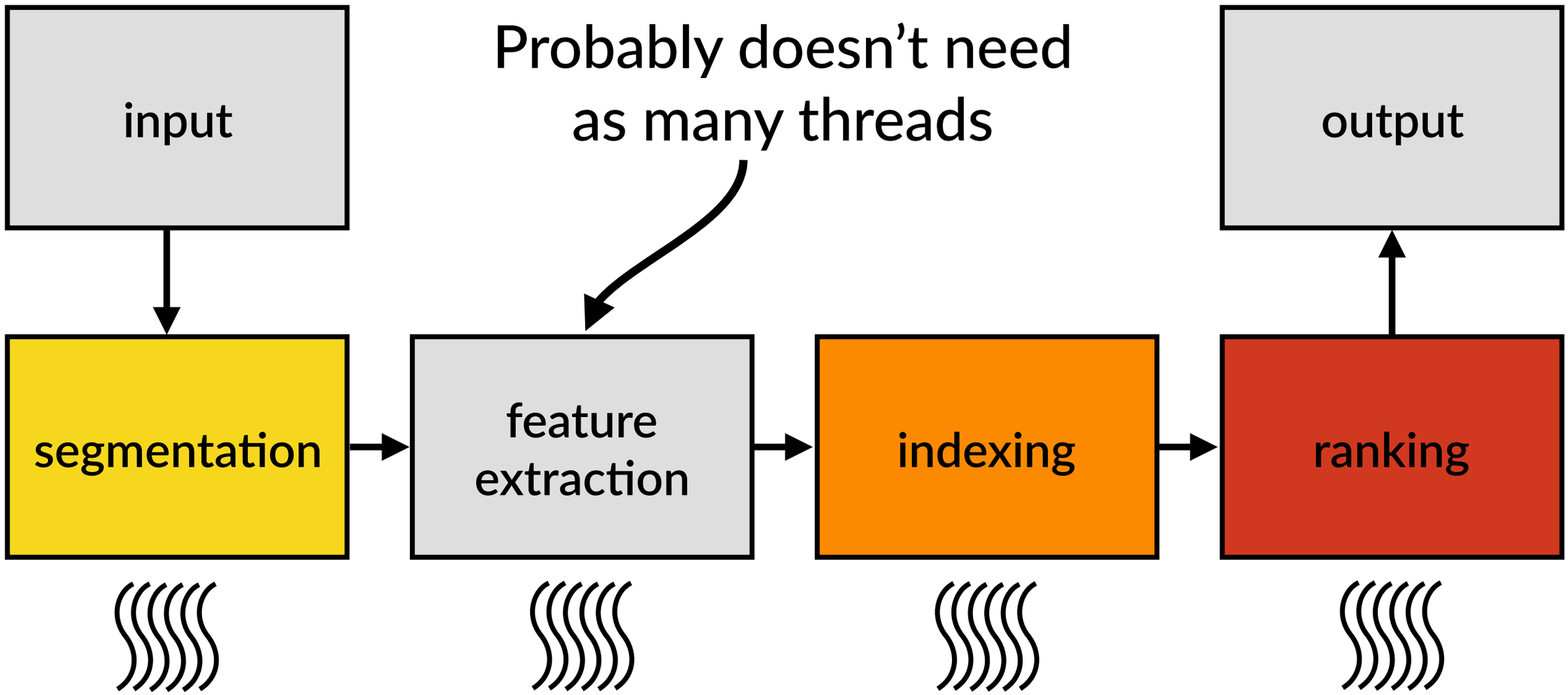
indexing

segmentation

Ferret

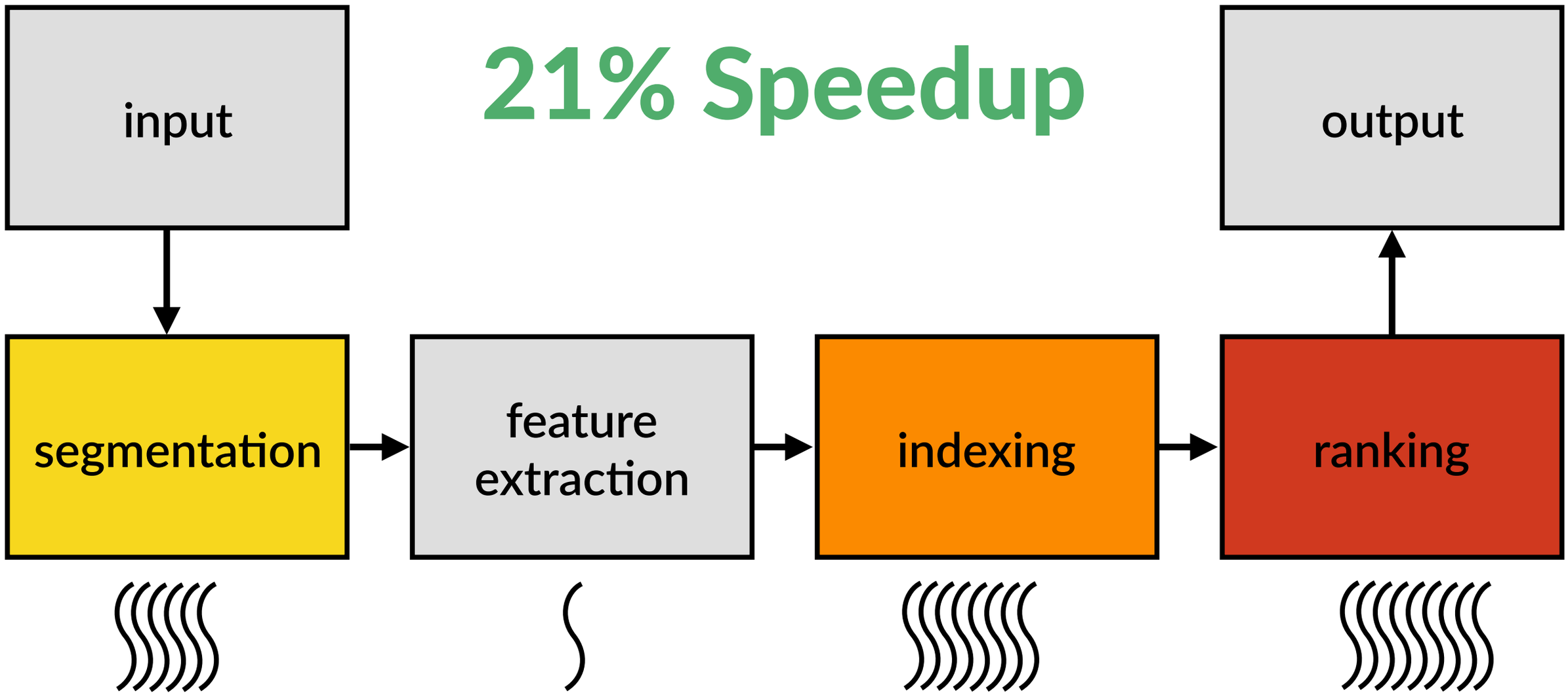

input
segmentation
feature extraction
indexing
ranking
output

Ferret


input
Probably doesn't need as many threads
output
segmentation
feature extraction
indexing
ranking

Ferret

21% Speedup

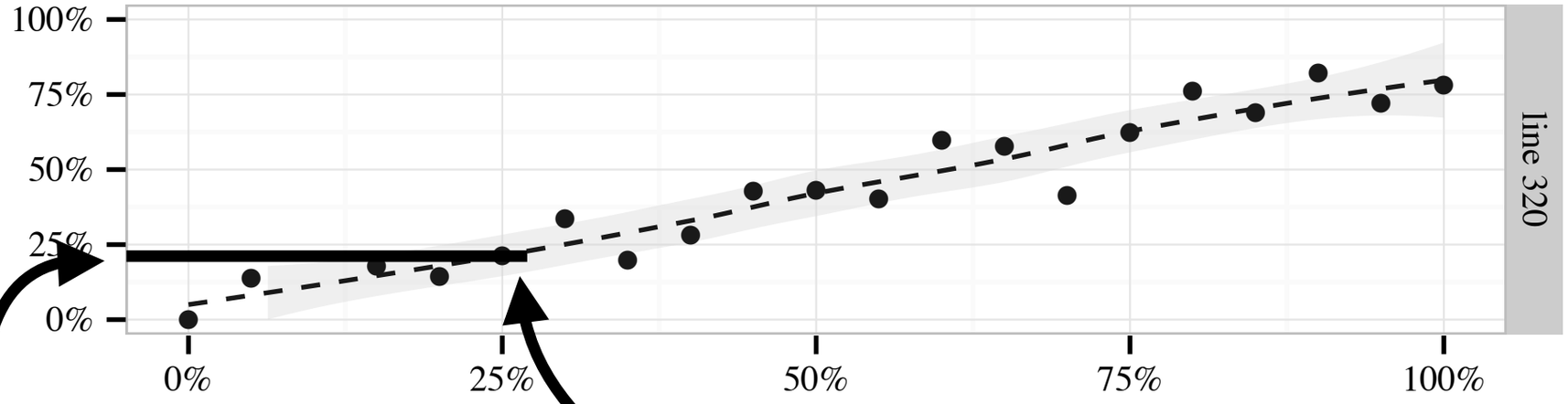


What did Causal Profiling predict?

Increased from 16 to 22 threads

27% increase in ranking throughput

What did Causal Profiling predict?



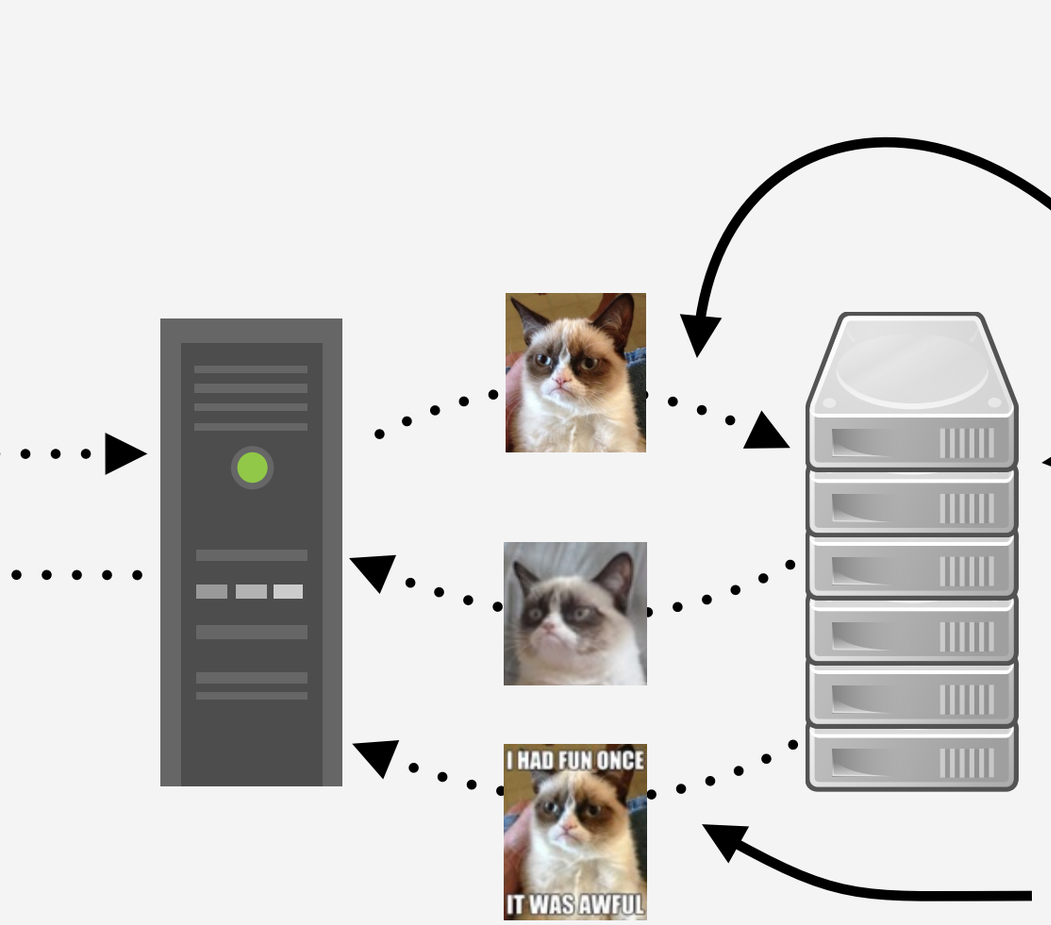
ranking

27% increase in ranking throughput

Causal Profiling predicted a 21% improvement

Exactly what we observed

Using Causal Profiling on Ogle



SQLite

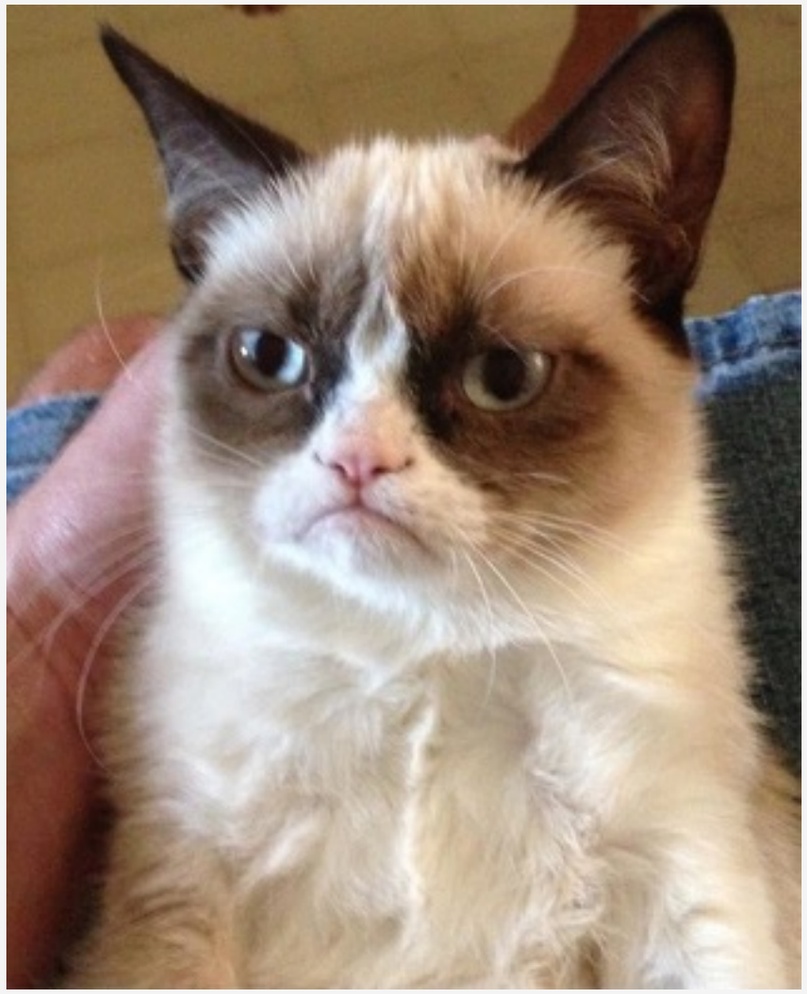
dedup
compression

ferret
image comparison

PARSEC
['pär-''sek] A unit of measure

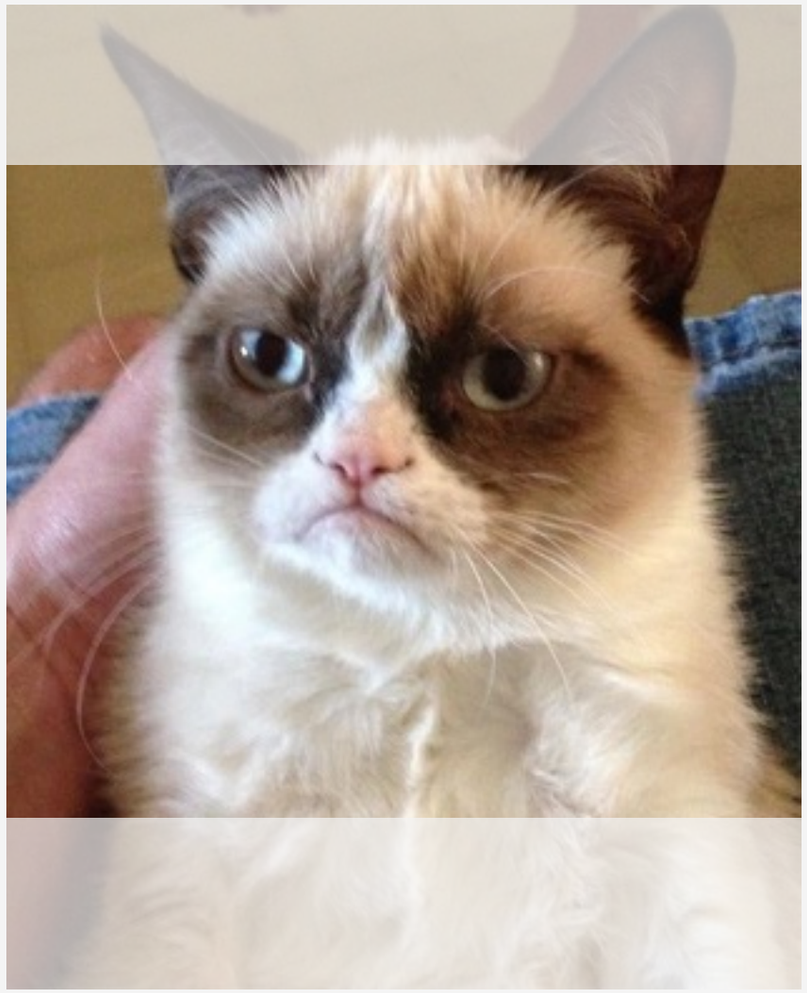
Dedup

Compression via deduplication



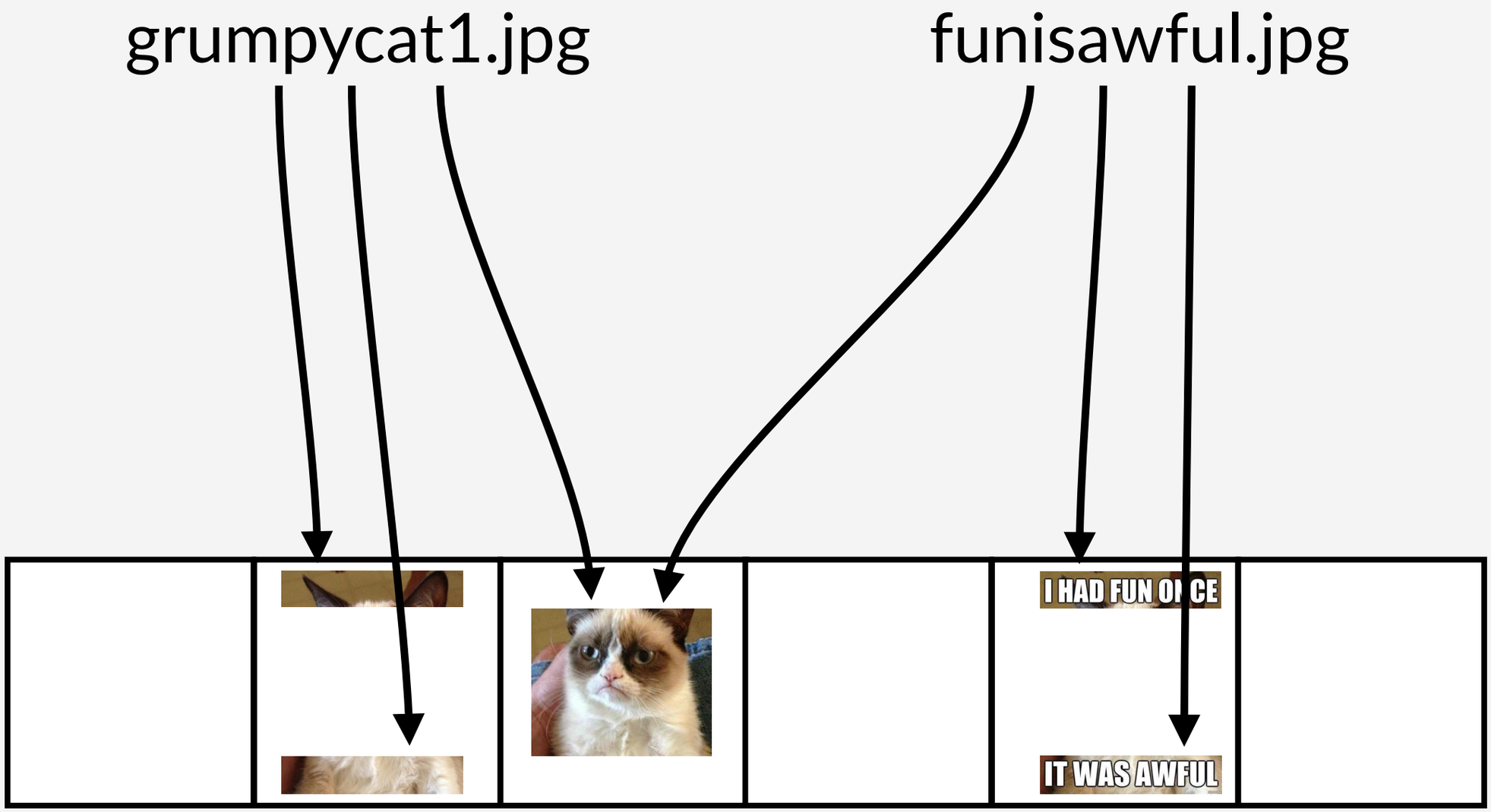
Dedup

Compression via deduplication



Dedup

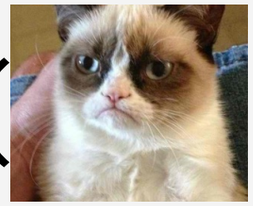
Compression via deduplication



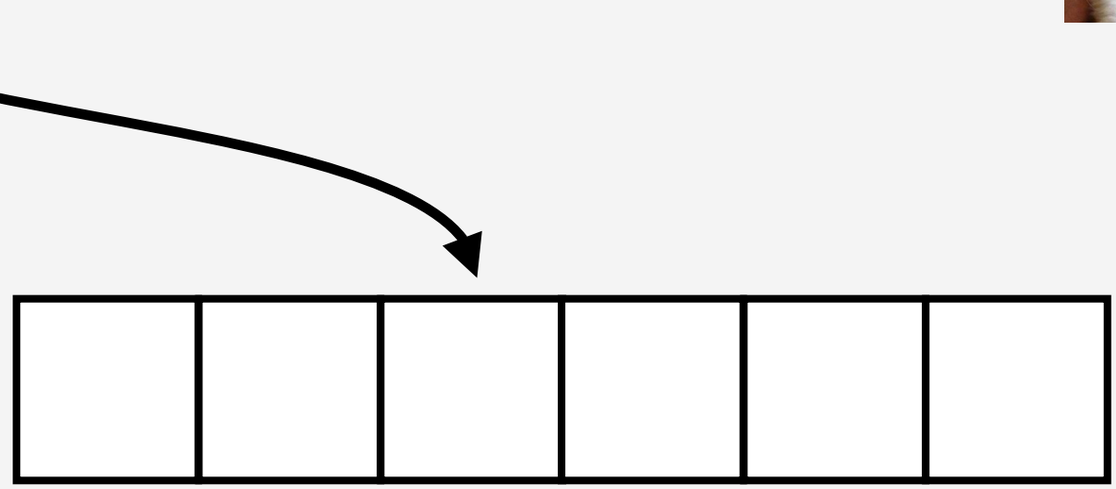
Dedup

Compression via deduplication

```
i = hash_function(
```



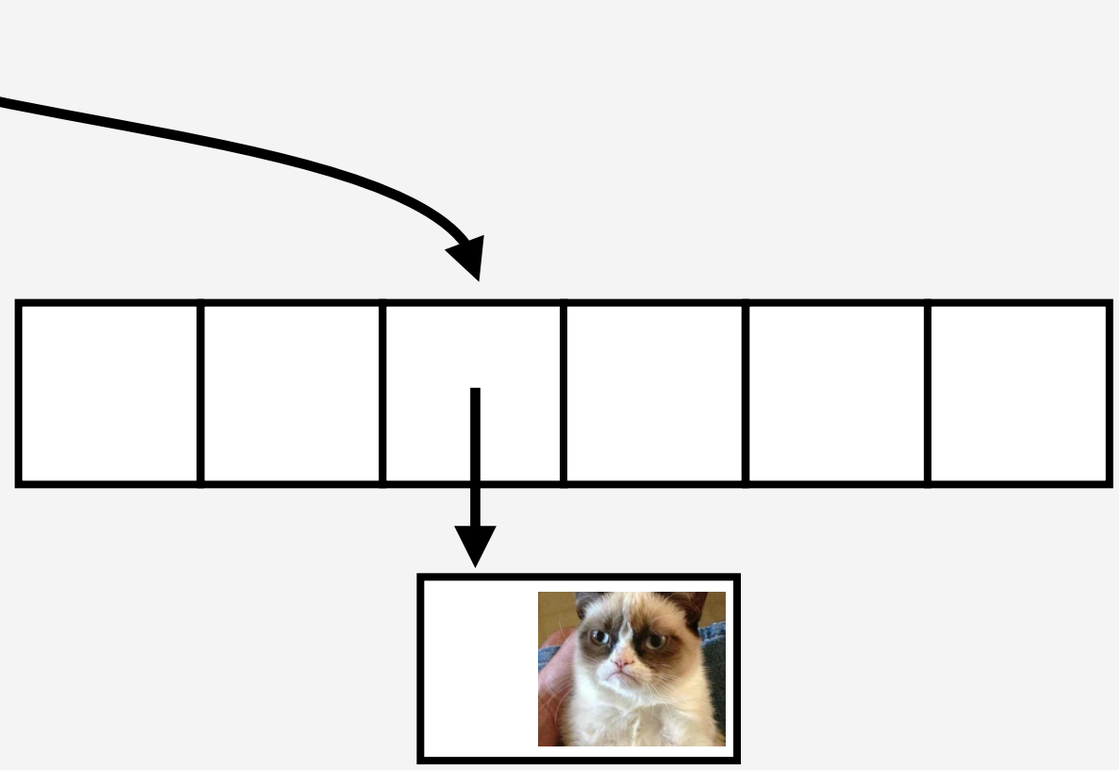
)



Dedup

Compression via deduplication

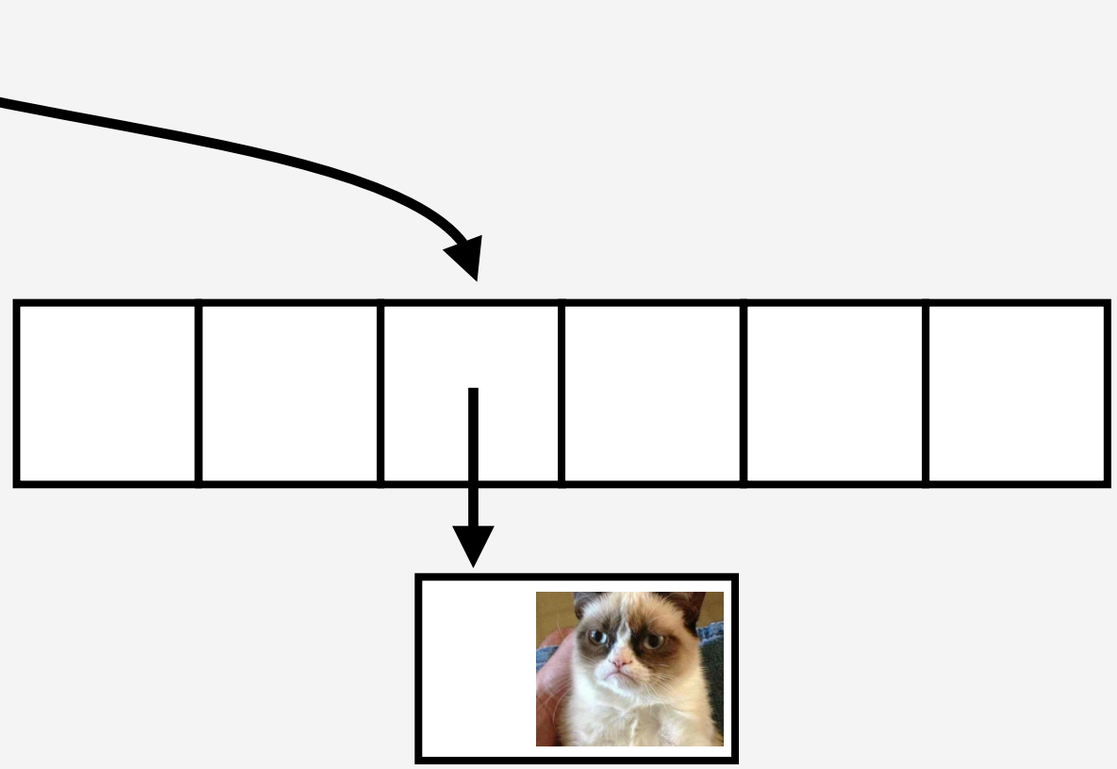
```
i = hash_function(    )
```



Dedup

Compression via deduplication

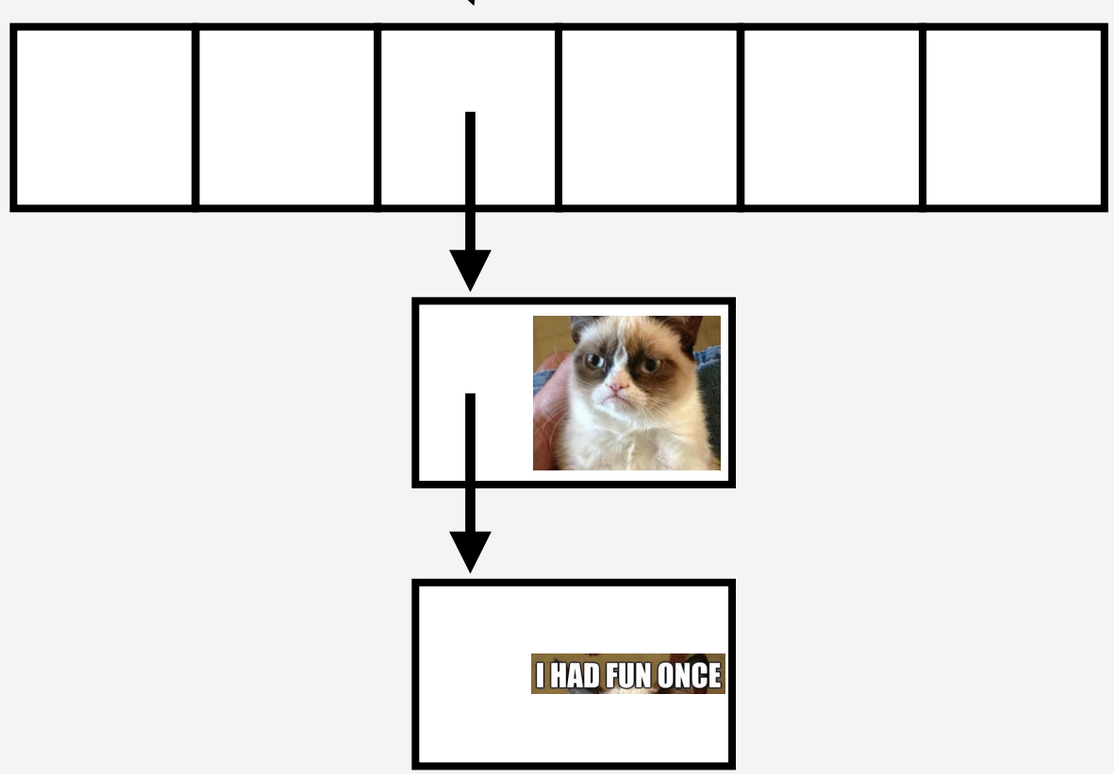
```
i = hash_function( I HAD FUN ONCE )
```



Dedup

Compression via deduplication

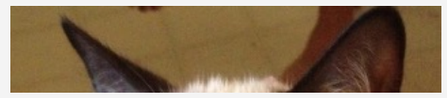
```
i = hash_function(          )
```

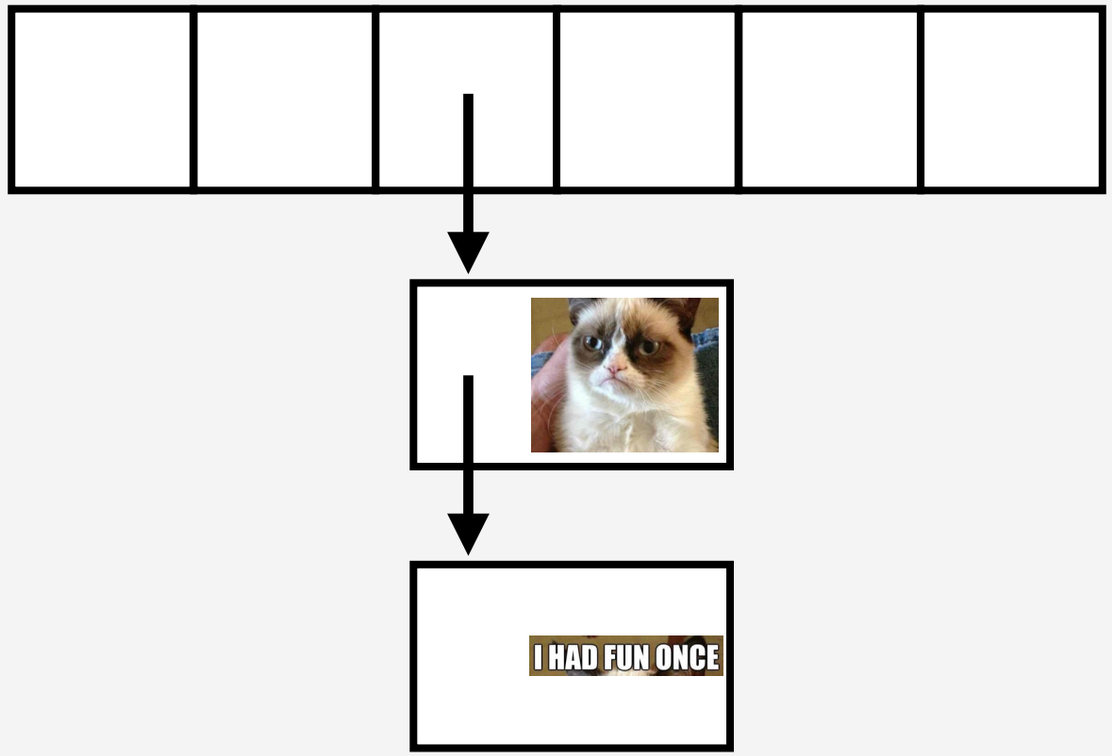


Dedup

Compression via deduplication

```
i = hash_function(
```

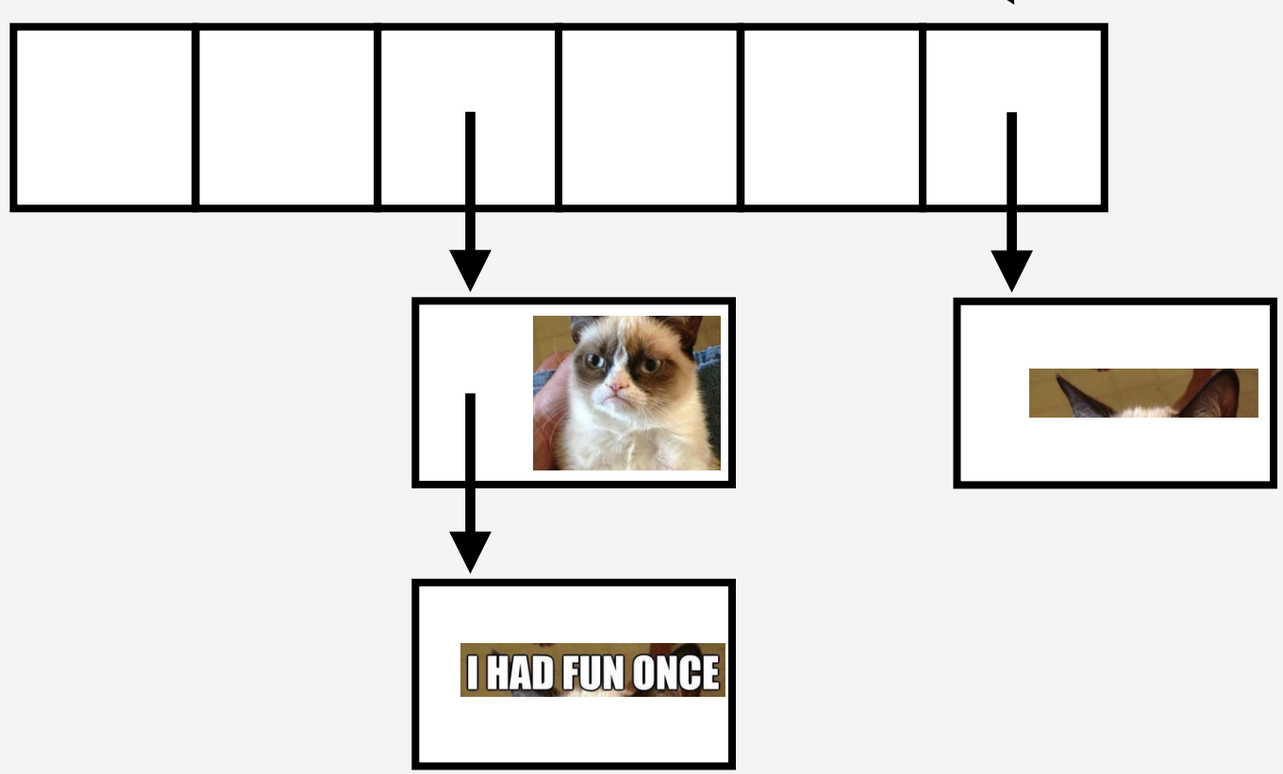




Dedup

Compression via deduplication

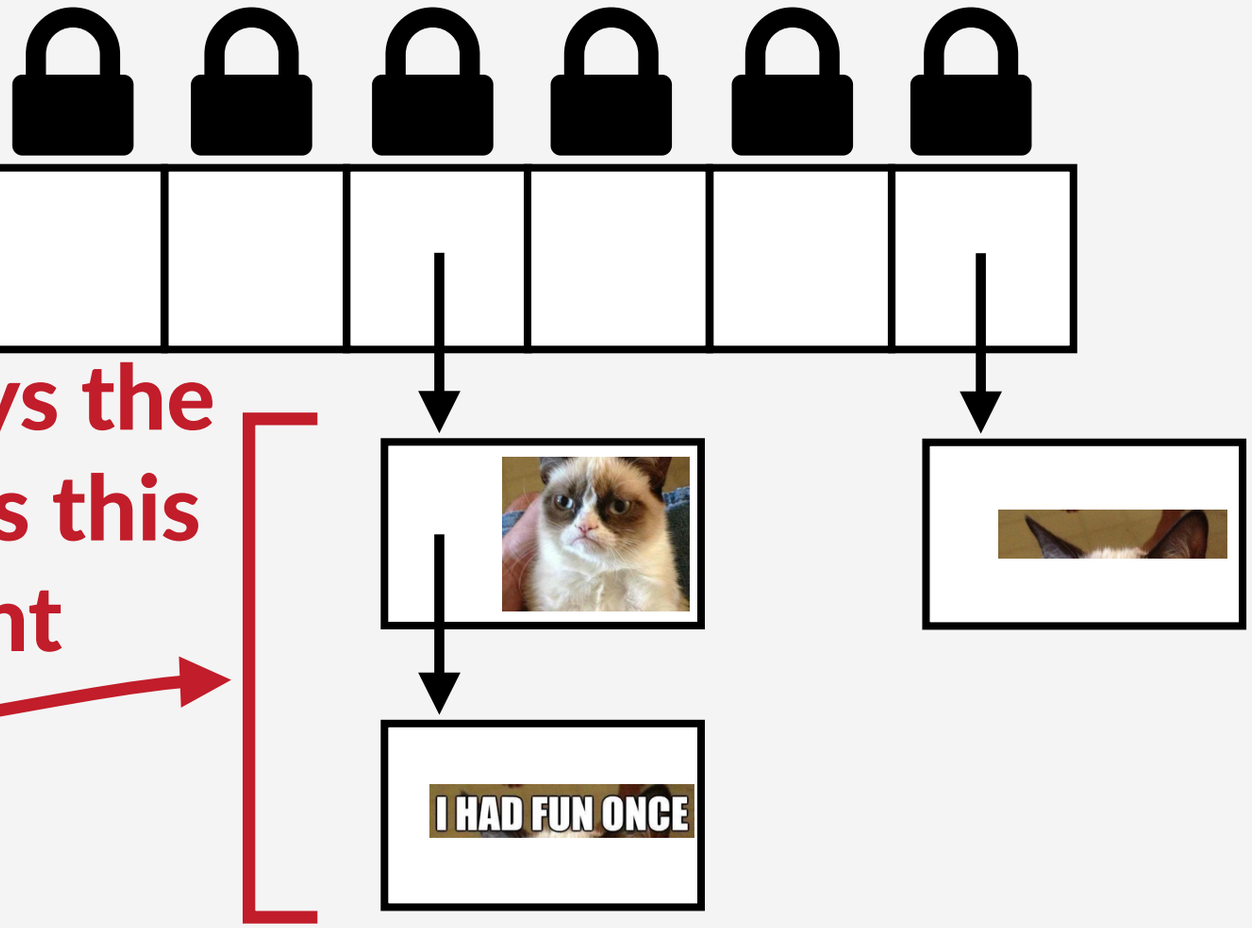
```
i = hash_function(          )
```



Dedup

Compression via deduplication

Hash table is accessed concurrently by many threads

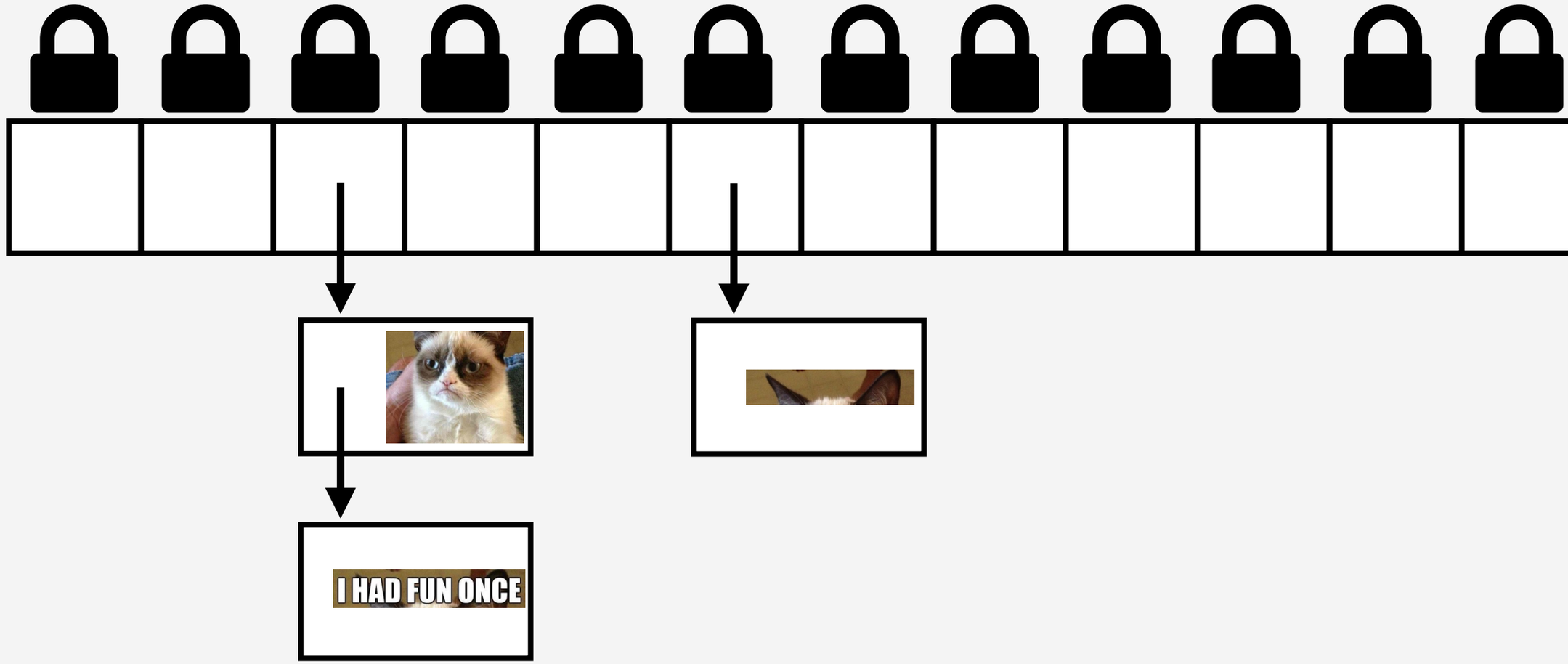


Causal Profiler says the loop that accesses this list is important

Dedup

Compression via deduplication

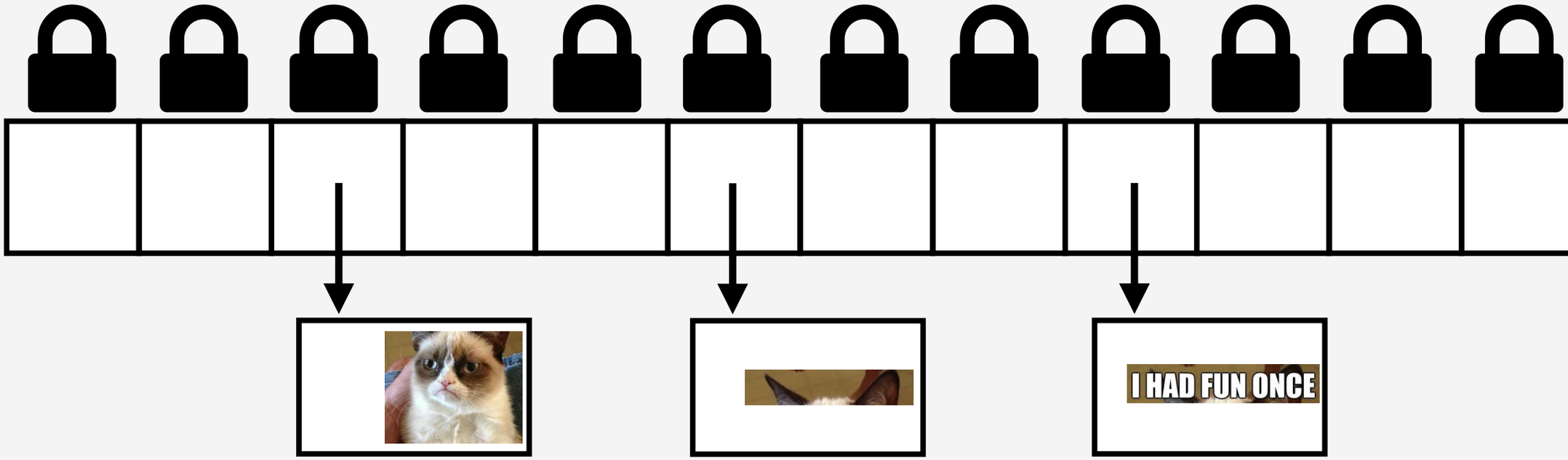
More hash buckets should lead to fewer collisions



Dedup

Compression via deduplication

More hash buckets should lead to fewer collisions

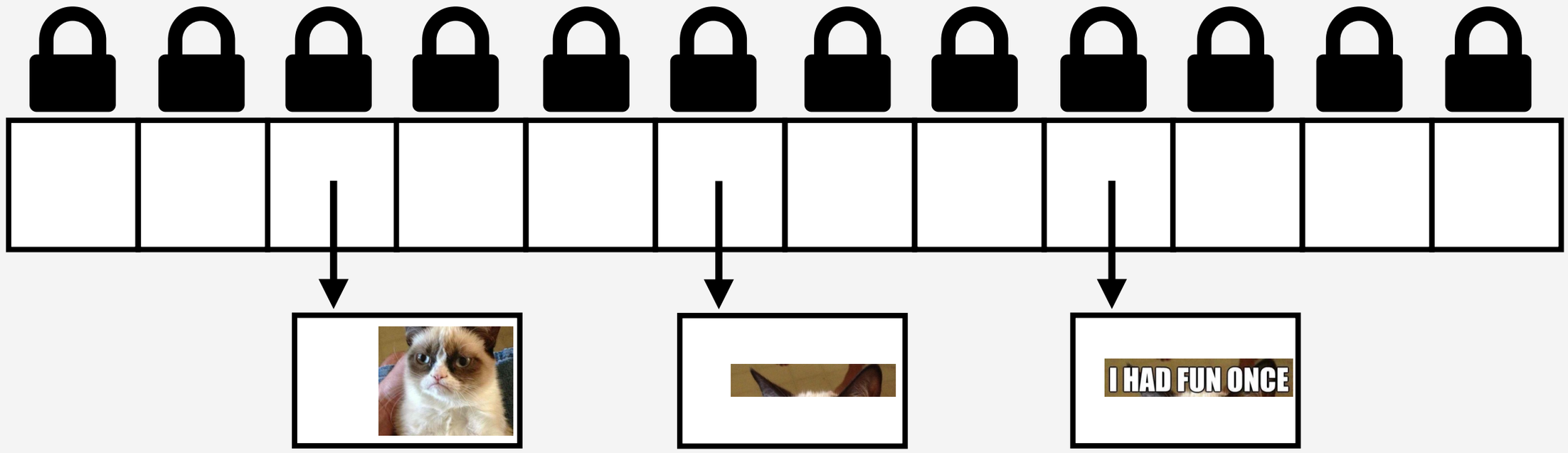


Dedup

Compression via deduplication

More hash buckets should lead to fewer collisions

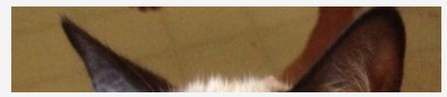
No performance improvement

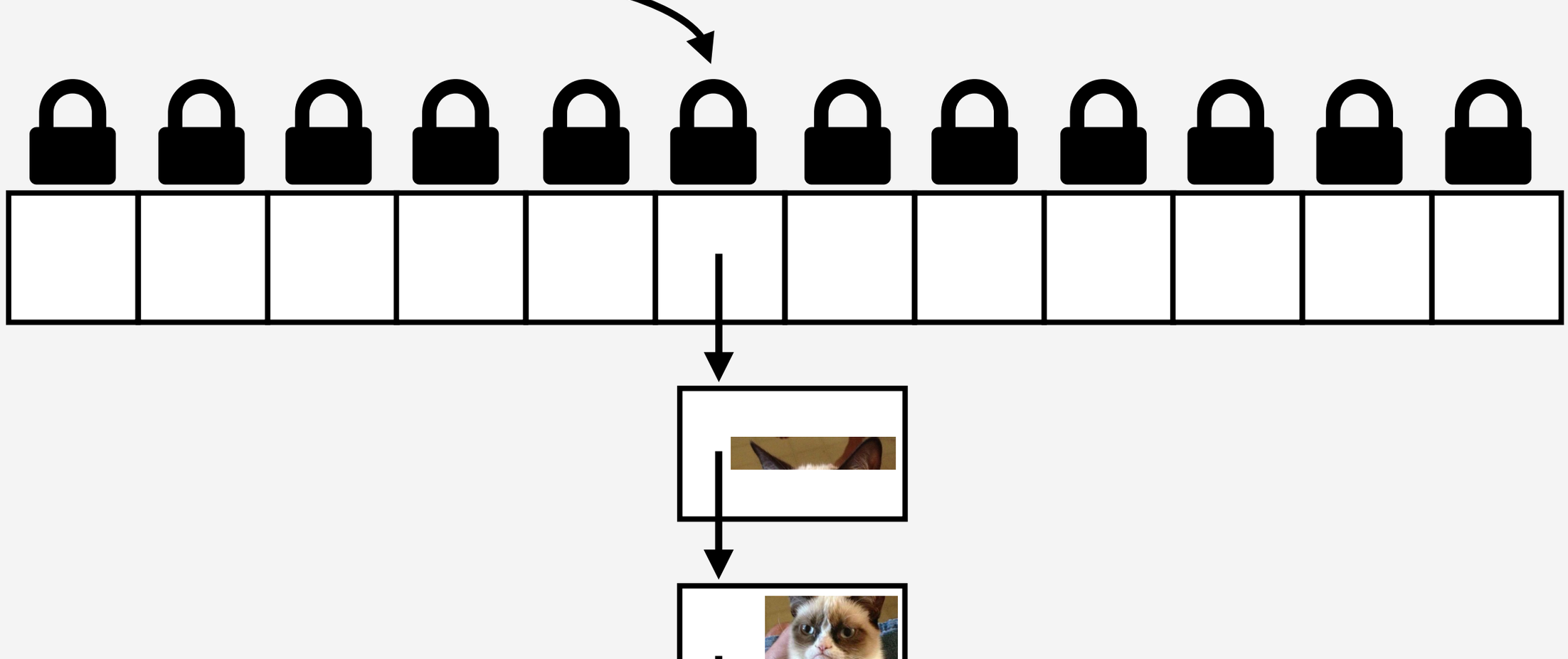


Dedup

Compression via deduplication

What else could be causing collisions?

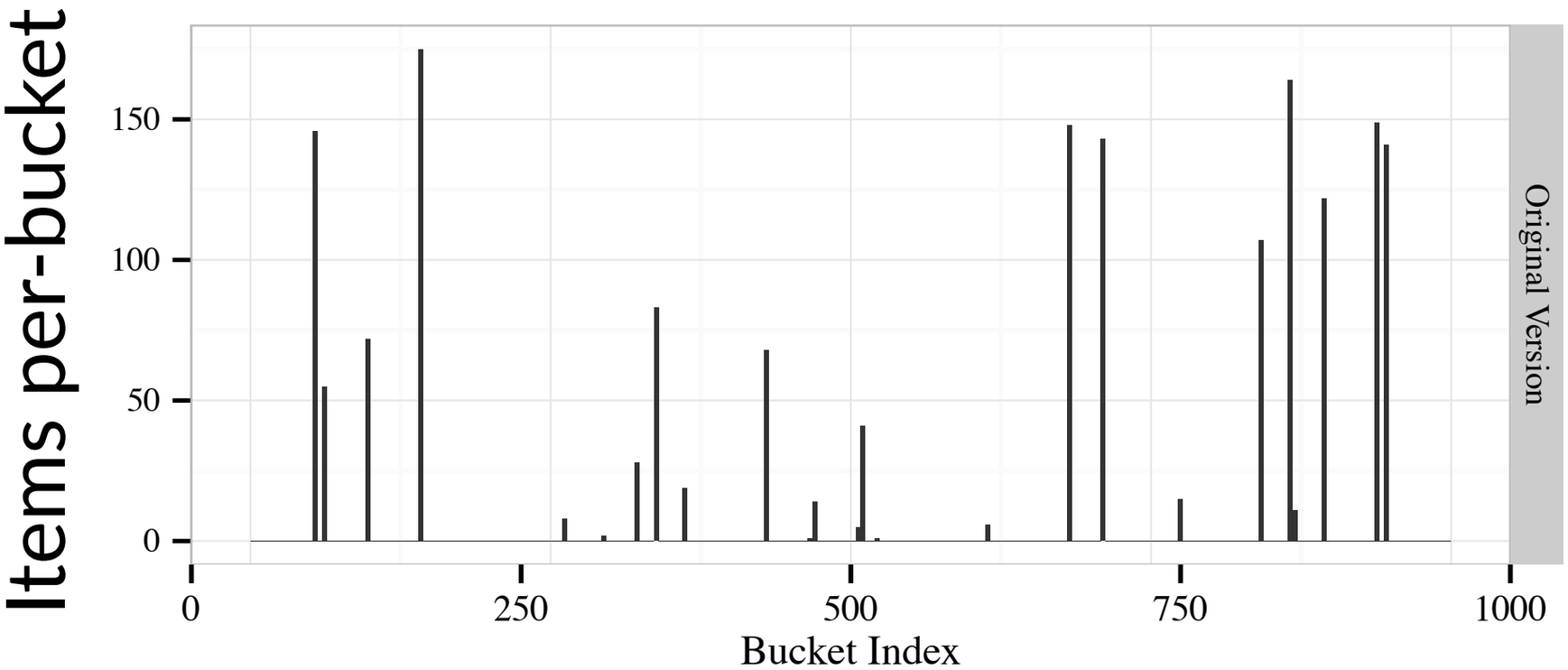
`i = hash_function(`  `)`



Dedup

Compression via deduplication

Horrible hash function!

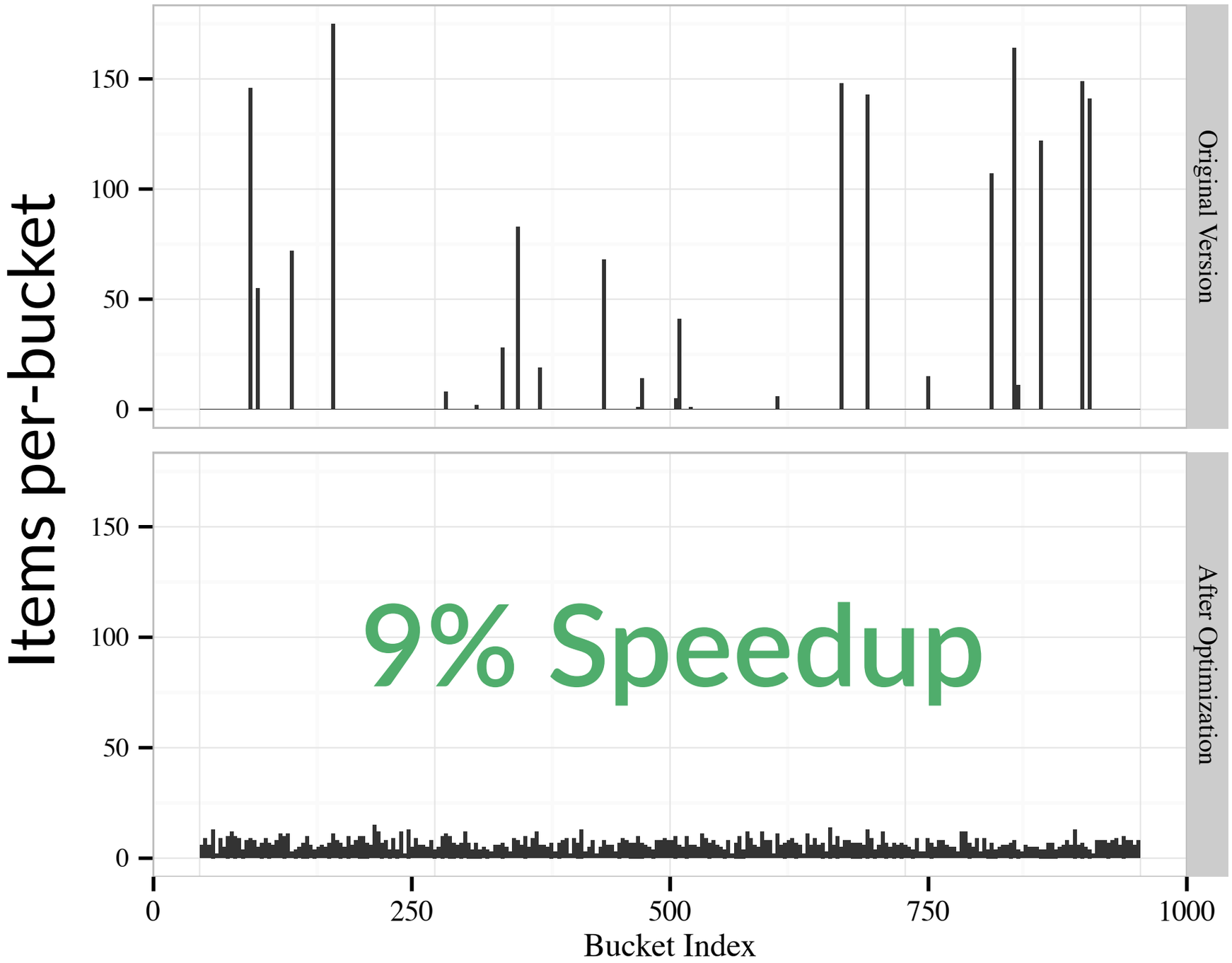


Bin Utilization

2.3%

Dedup

Compression via deduplication



Bin Utilization

2.3%

82%

Dedup

Compression via deduplication

What did Causal Profiling predict?

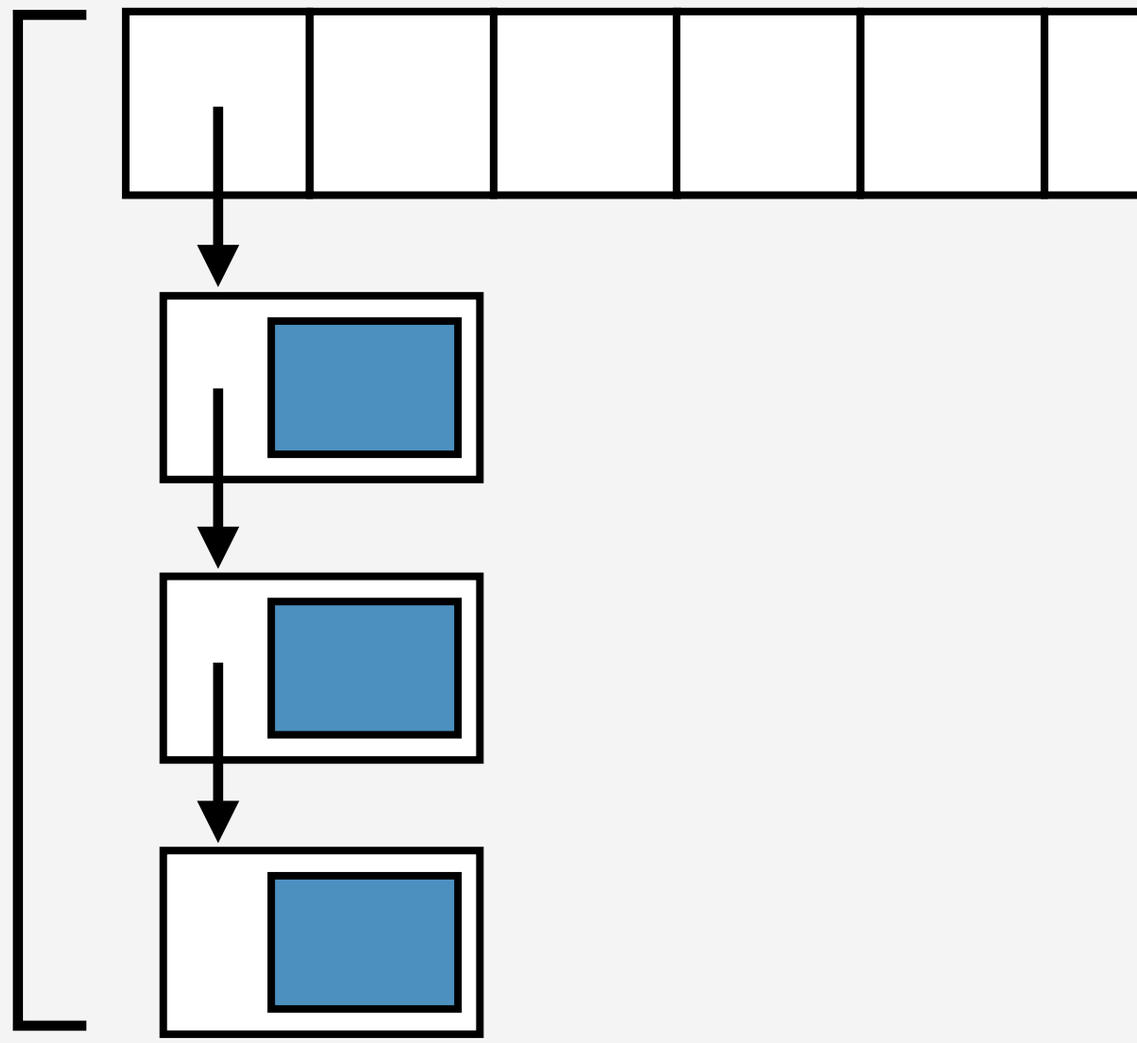
Blocks per-bucket

Before: 76.7

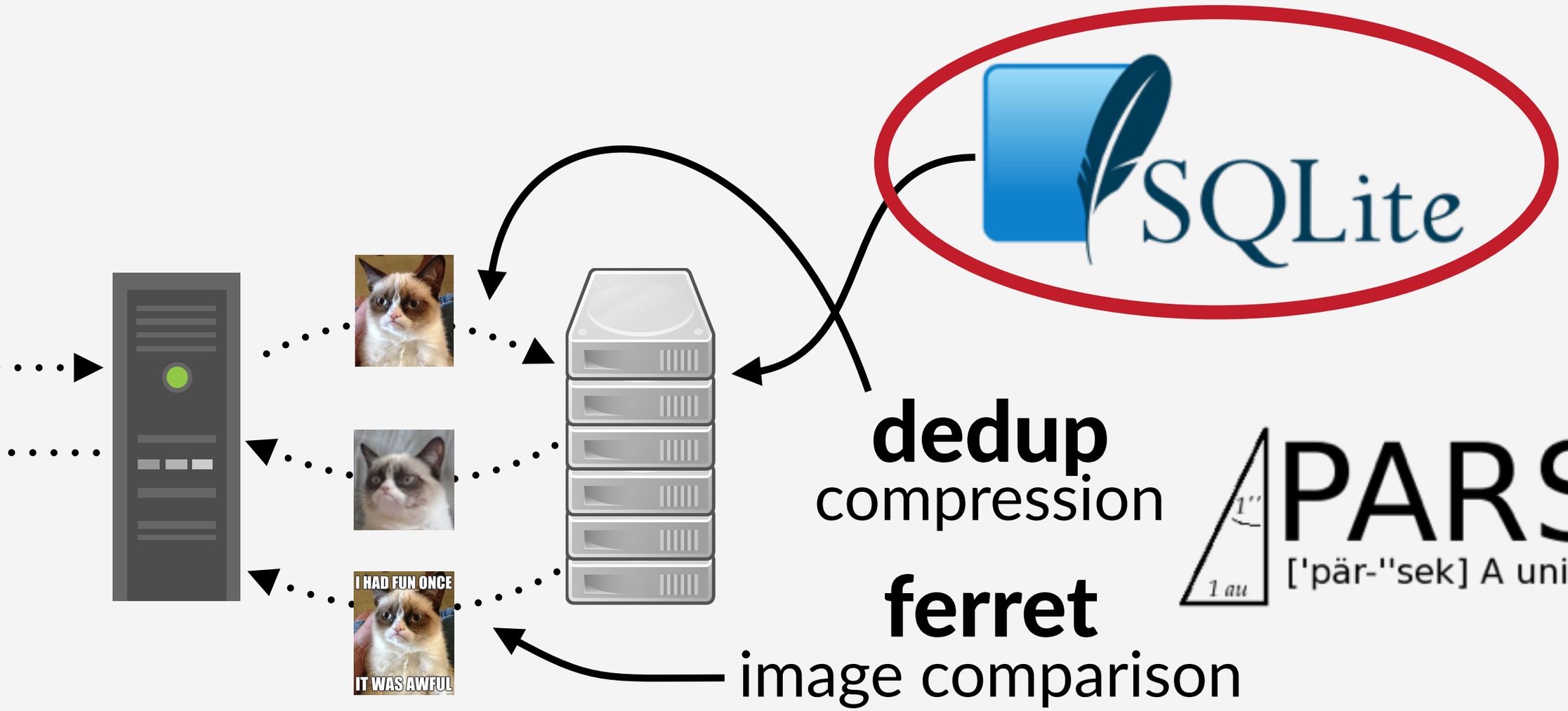
After: 2.09

96% traversal speedup

9% predicted speedup, exactly what we observed



Using Causal Profiling on Ogle



dedup
compression

ferret
image comparison

PARSEC
['pär-''sek] A unit of measure

Simple SQL Database

```
#if THREAD_SAFE
config_t global_config = {
  ...
    .unlock = pthread_mutex_unlock,
    .getsize = sqlite_usable_size,
    .nextitem = sqlite_pagecache_next,
  ...
};
#endif
```

Simple SQL Database

```
void pthreadMutexLeave(lock* l) {
   global_config.unlock(l);
}
```

Indirect Call

Cheap, but almost the same cost as `pthread_mutex_unlock`

Simple SQL Database

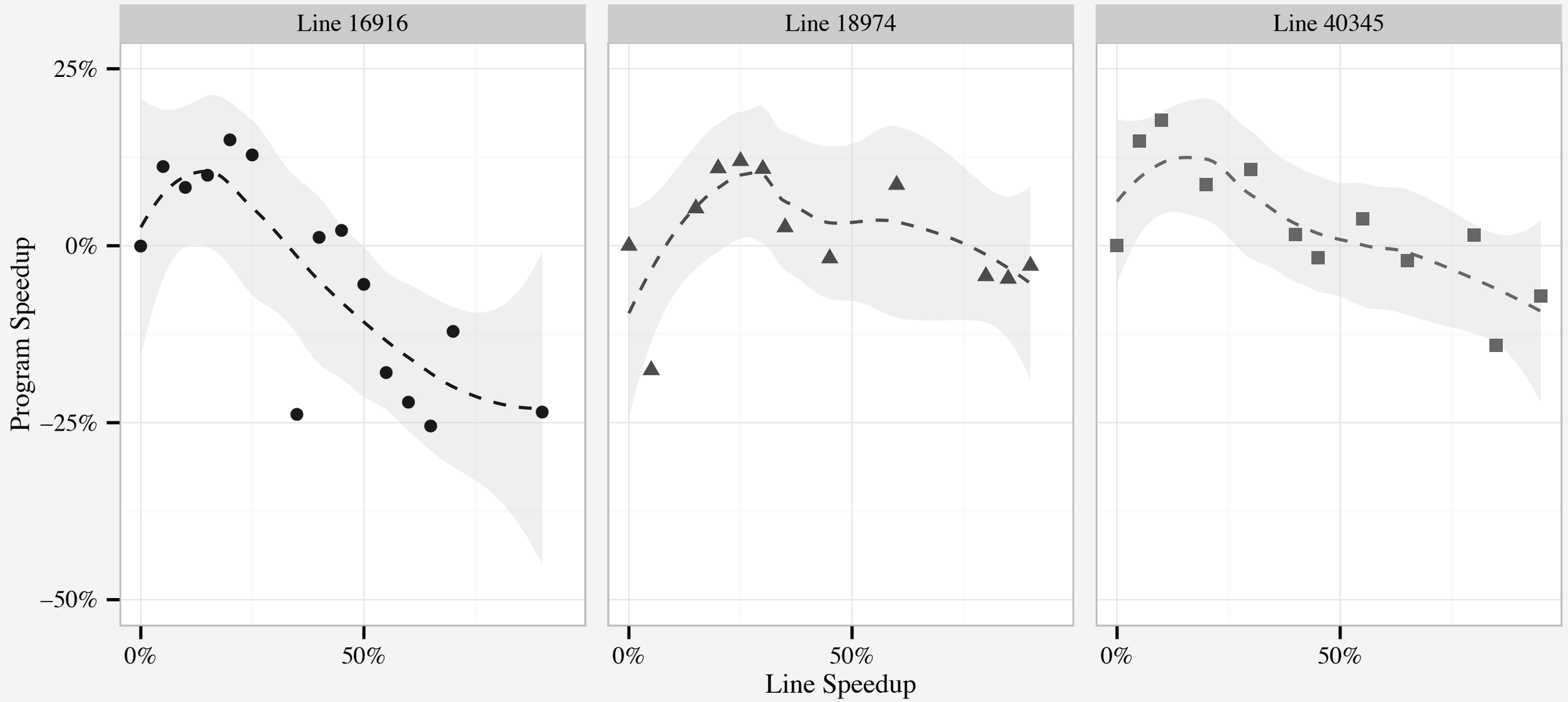
Coz highlights these lines

```
void pthreadMutexLeave(lock* l) {
  global_config.unlock(l);
}

void sqlite3MemSize(void* p) {
  global_config.getsize(p);
}

void pcache1Fetch(item* i) {
  global_config.nextitem(i);
}
```

Simple SQL Database



Simple SQL Database

```
void pthreadMutexUnlock(lock* l) {
  global_config.unlock(l);
}

void sqlite3MemSize(void* p) {
  global_config.getsize(p);
}

void pcache1Fetch(item* i) {
  global_config.nextitem(i);
}
```

Simple SQL Database

```
void pthreadMutexUnlock(lock* l) {
  pthread_mutex_unlock(l);
}
```

```
void sqlite3MemSize(void* p) {
  sqlite_usable_size(p);
}

void pcache1Fetch(item* i) {
  sqlite_pagecache_next(i);
}
```

25% Speedup

What do other profilers say?

```
% Runtime     Symbol
85.55%         _raw_spin_lock
1.76%          x86_pmu_enable_all

   ... 30 lines …

0.10%          rcu_irq_enter
0.09%          sqlite3MemSize
0.09%          source_load

   ... 26 lines …

0.03%          __queue_work
0.03%          pcache1Fetch
0.03%          kmem_cache_free
0.03%          update_cfs_rq_blocked_load
0.03%          pthreadMutexLeave
0.03%          sqlite3MemMalloc
```

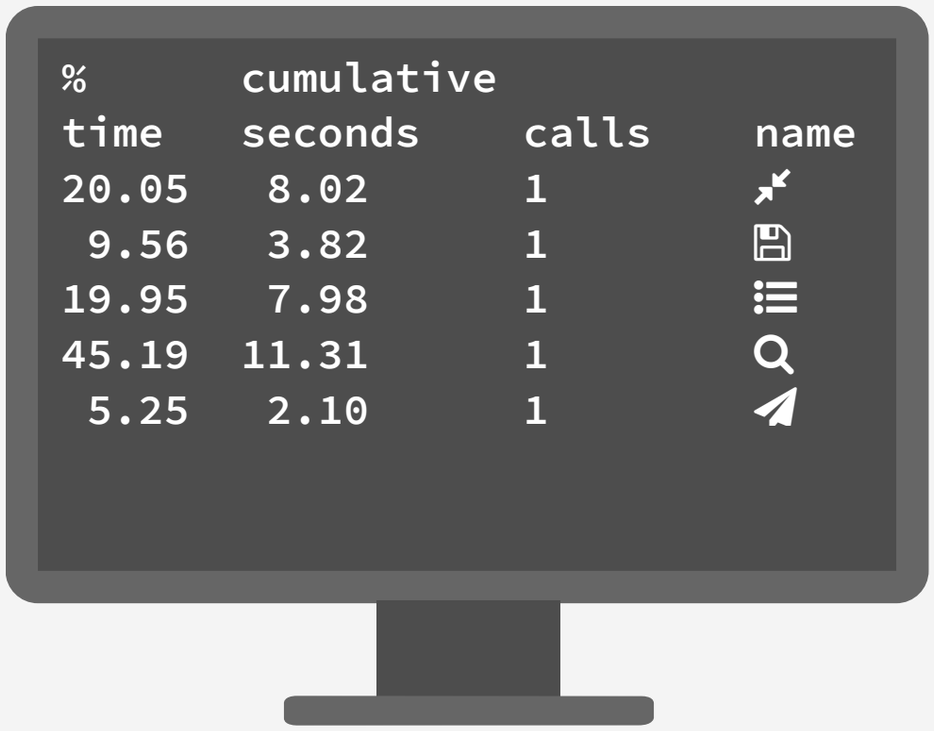
Just 0.15% of total runtime

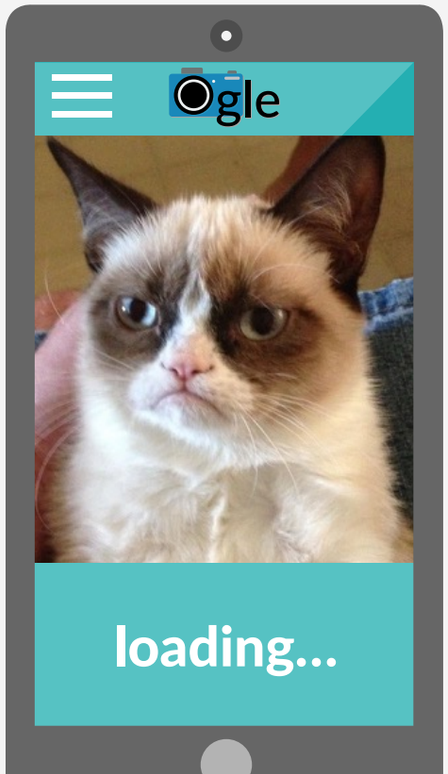
Using Causal Profiling on Ogle

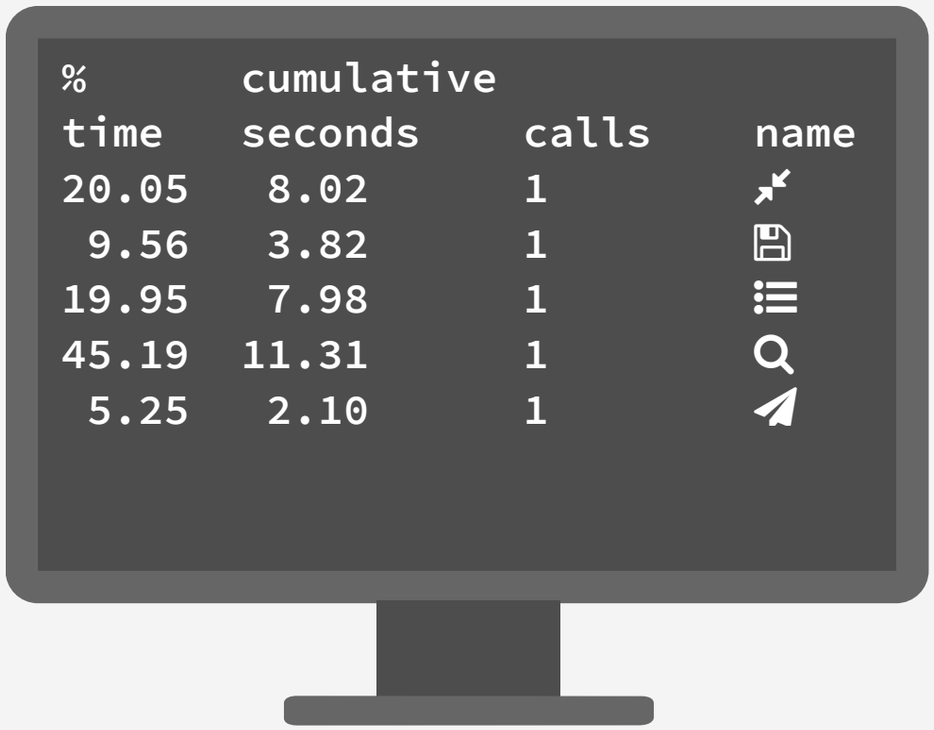
	speedup with Coz	% runtime
	25%	0.15%
dedup compression	9%	14.38% *(whole function)*
ferret image comparison	21%	0.00%

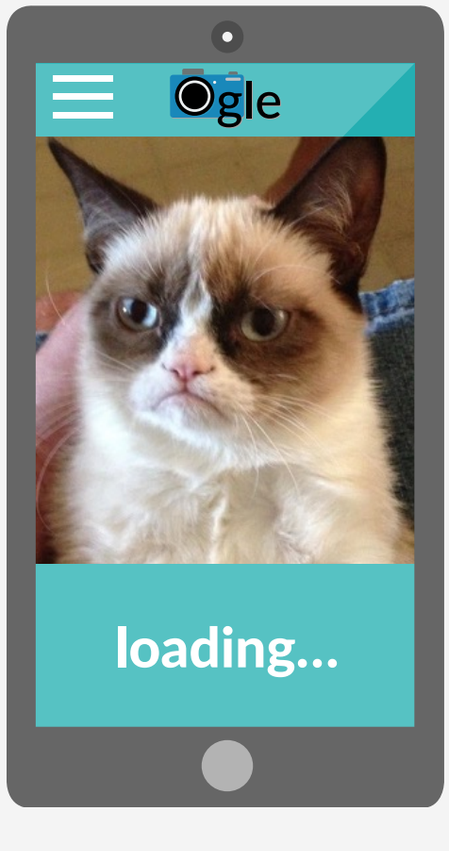
Summary of Optimizations

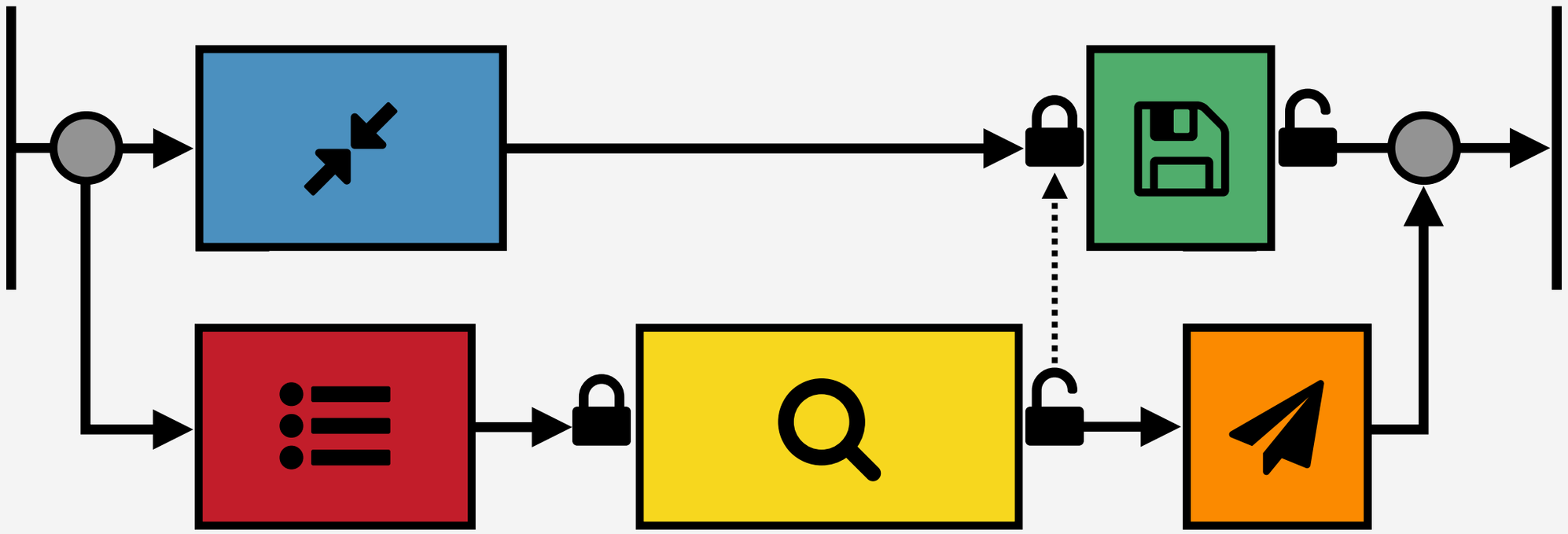
Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks
sqlite	25.60%	-3, +3	removed DIY vtable implementation
blackscholes	2.56%	-61, +4	manual common subexpression elimination
dedup	8.95%	-3, +3	fixed degenerate hash function
ferret	21.27%	-4, +4	rebalanced pipeline thread allocation
fluidanimate	37.50%	-1, +0	removed custom barrier with high contention
streamcluster	68.40%	-1, +0	removed custom barrier with high contention
swaptions	15.80%	-10, +16	reordered loop nests

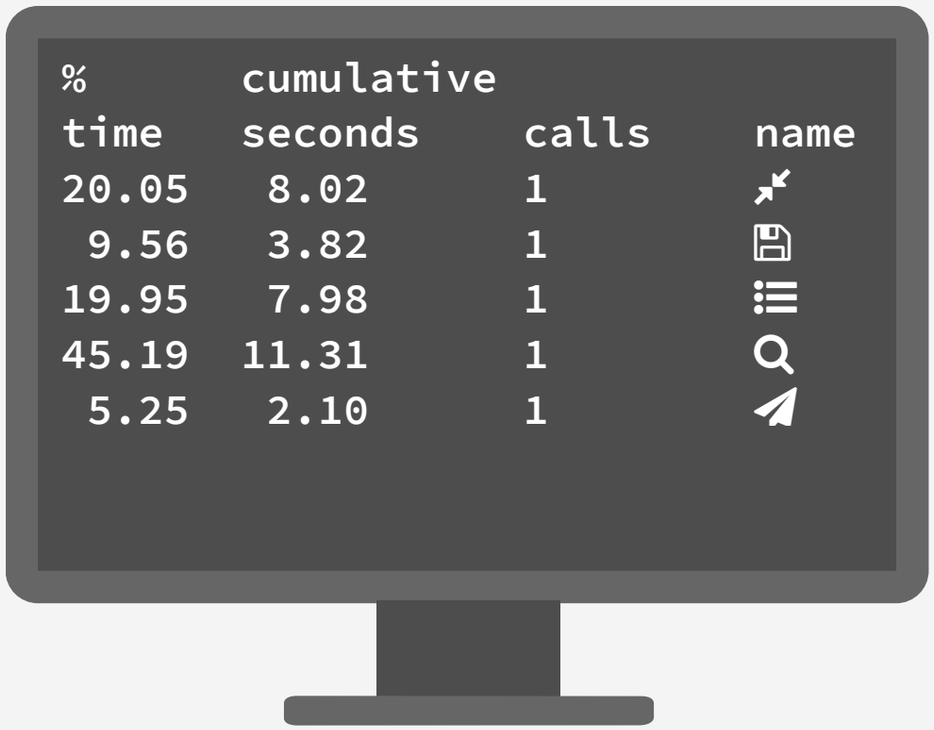

%
cumulative
time
seconds
calls
name
20.05
8.02
1
9.56
3.82
1
19.95
7.98
1
45.19
11.31
1
5.25
2.10
1

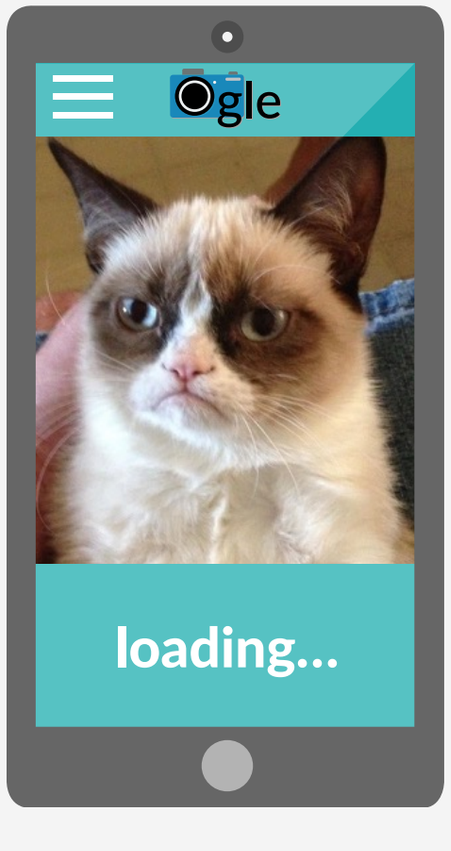

gle
loading...

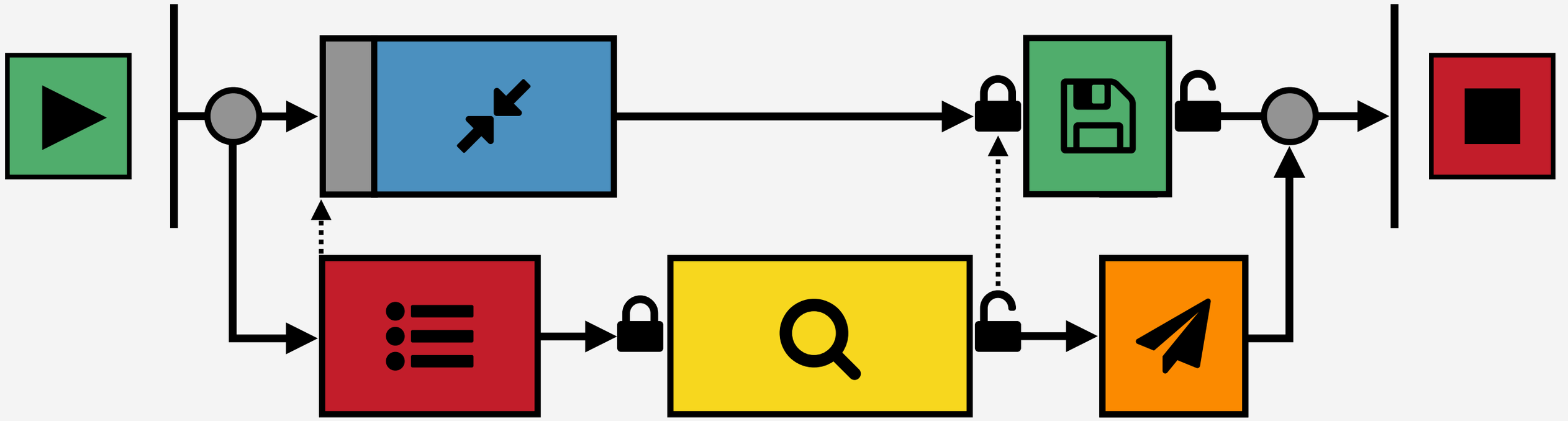

%
cumulative
time
seconds
calls
name
20.05
8.02
1
9.56
3.82
1
19.95
7.98
1
45.19
11.31
1
5.25
2.10
1


gle
loading...



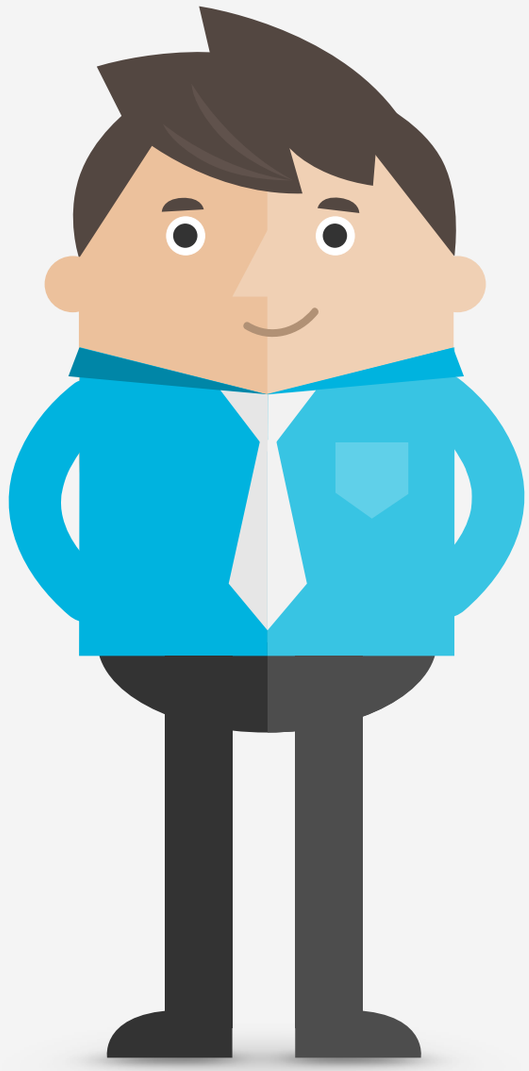

%
cumulative
time
seconds
calls
name
20.05
8.02
1
9.56
3.82
1
19.95
7.98
1
45.19
11.31
1
5.25
2.10
1


gle
loading...



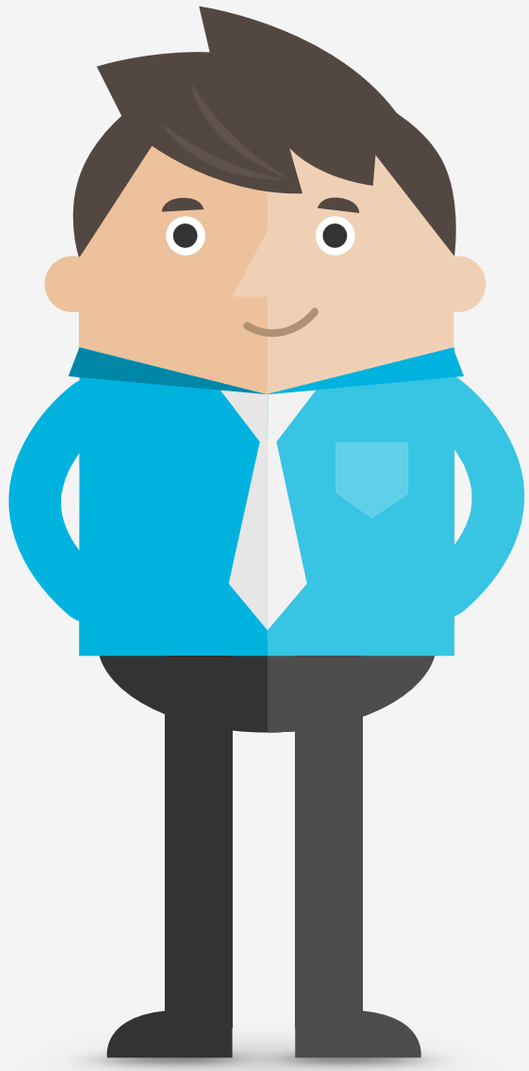
This *is* the profiler you are looking for.

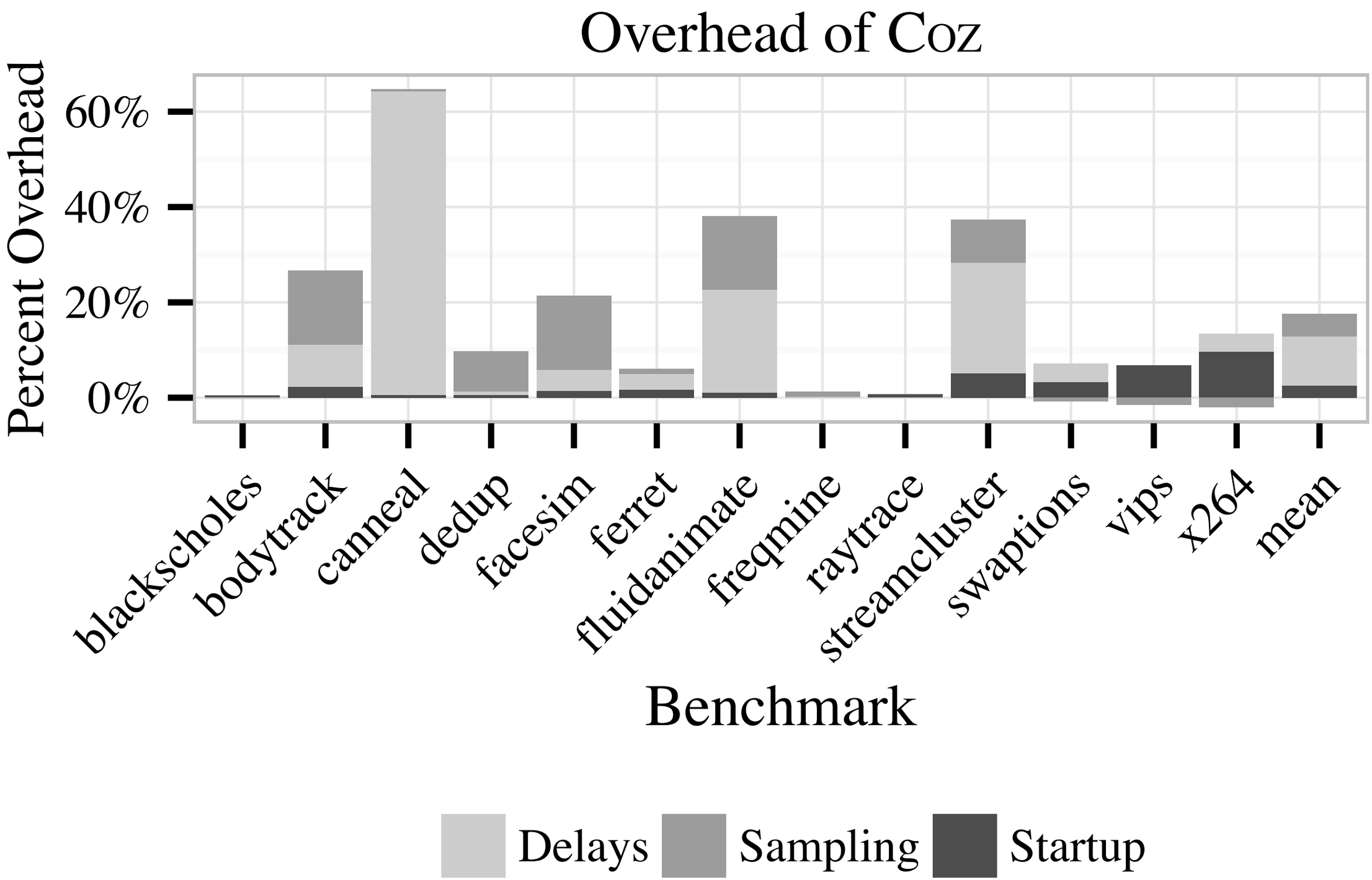
coz-profiler.org



This *is* the profiler you are looking for.

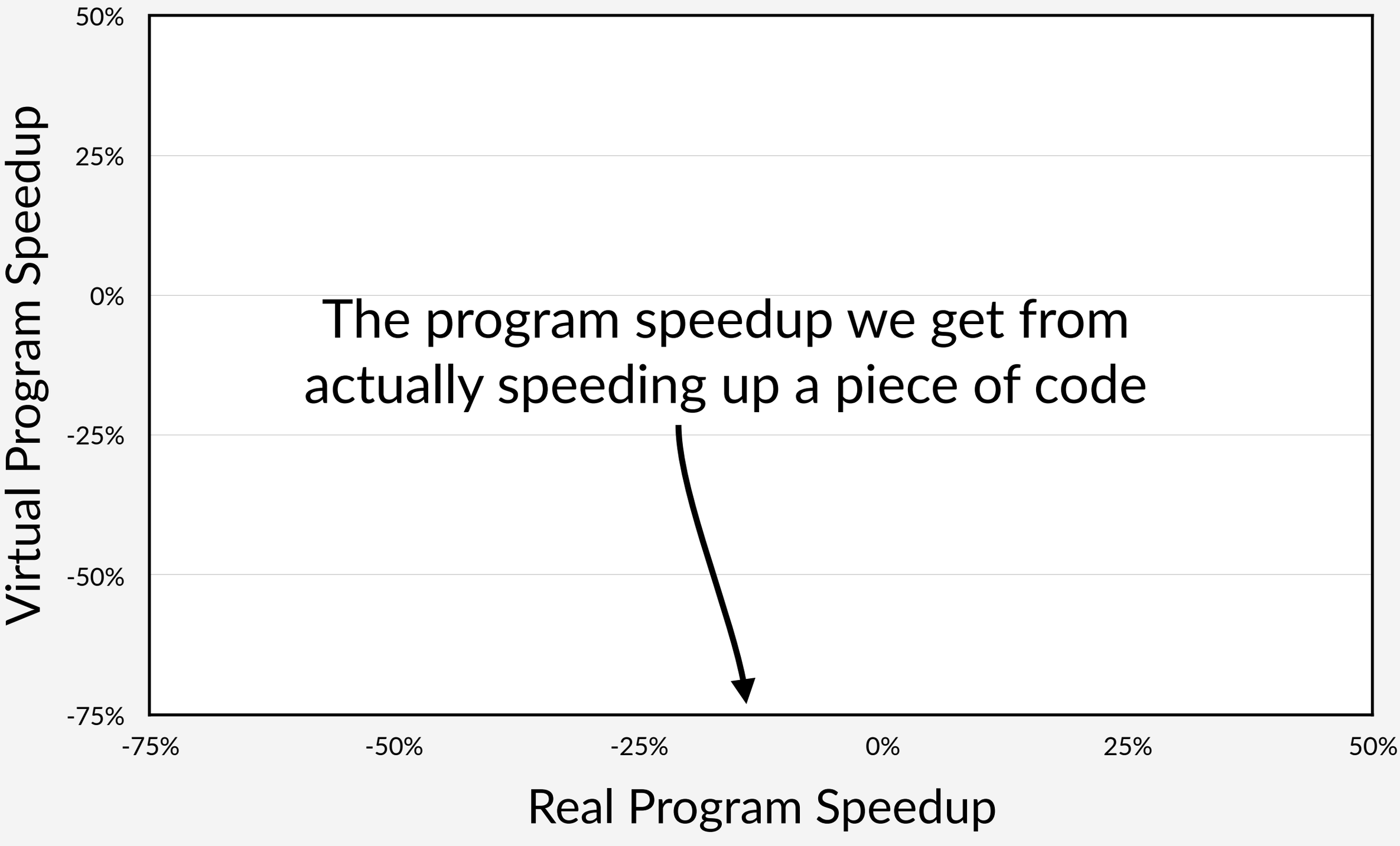
coz-profiler.org




Overhead of Coz
Percent Overhead
60%
40%
20%
0%
blackscholes
bodytrack
canneal
dedup
facesim
ferret
fluidanimate
freqmine
raytrace
streamcluster
swaptions
vips
x264
mean
Benchmark
Delays
Sampling
Startup

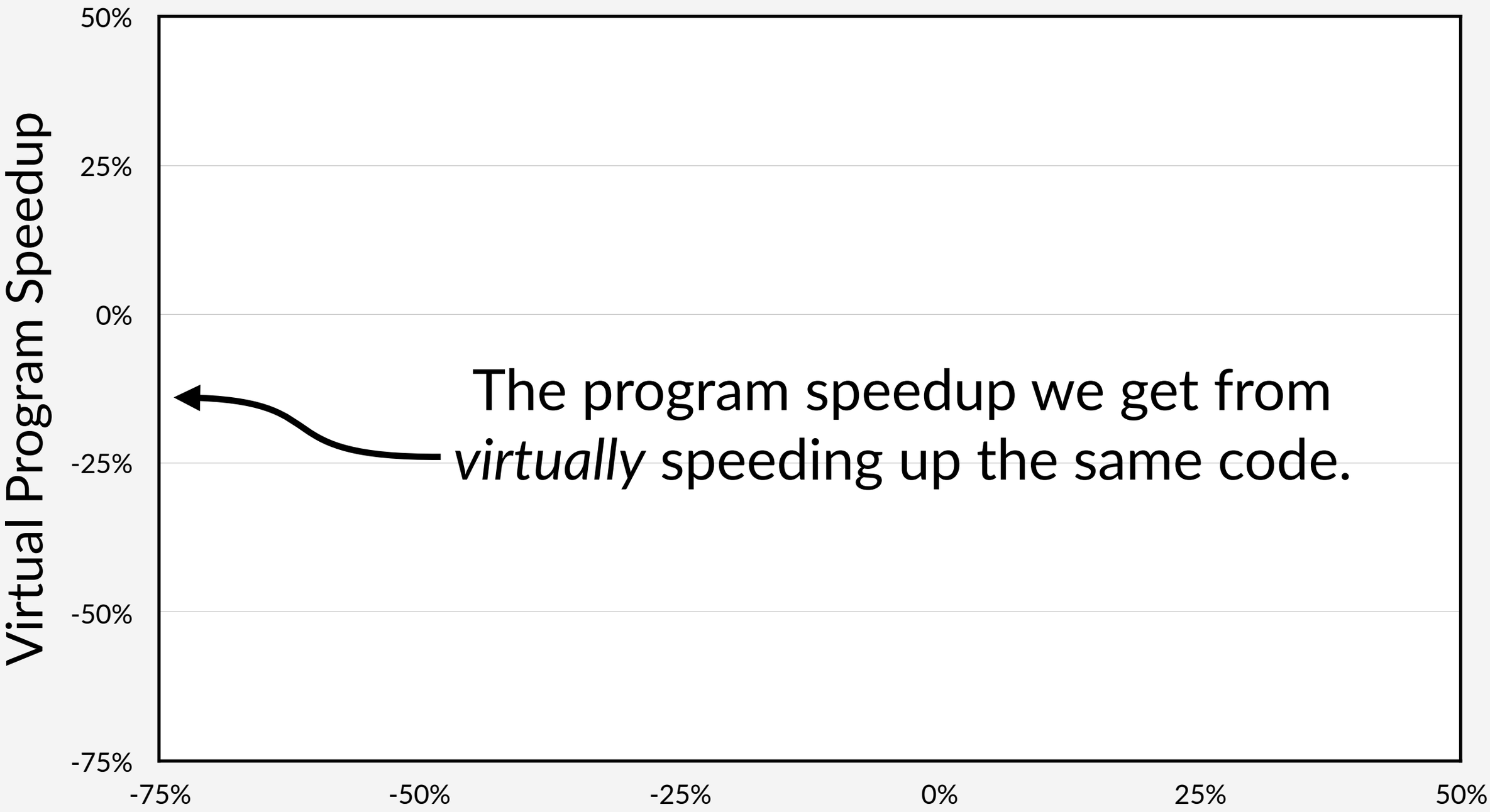
Virtual Speedup Accuracy

Parallel compression program `pbzip2`



Virtual Speedup Accuracy

Parallel compression program `pbzip2`



Virtual Speedup Accuracy

Parallel compression program `pbzip2`