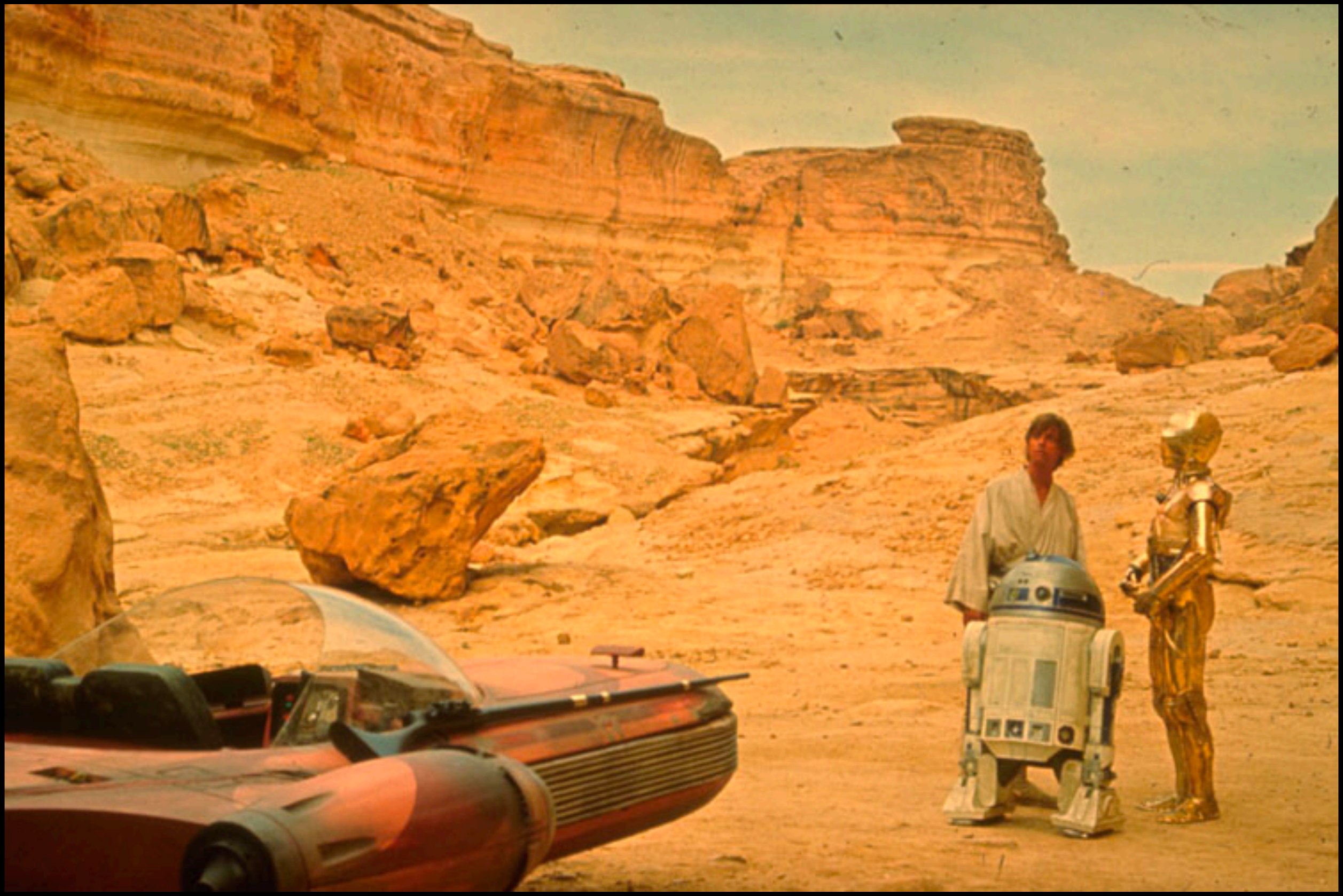
Performance Matters

Emery Berger
College of Information and Computer Sciences
UMass Amherst

(joint work with Charlie Curtsinger, Grinnell College)

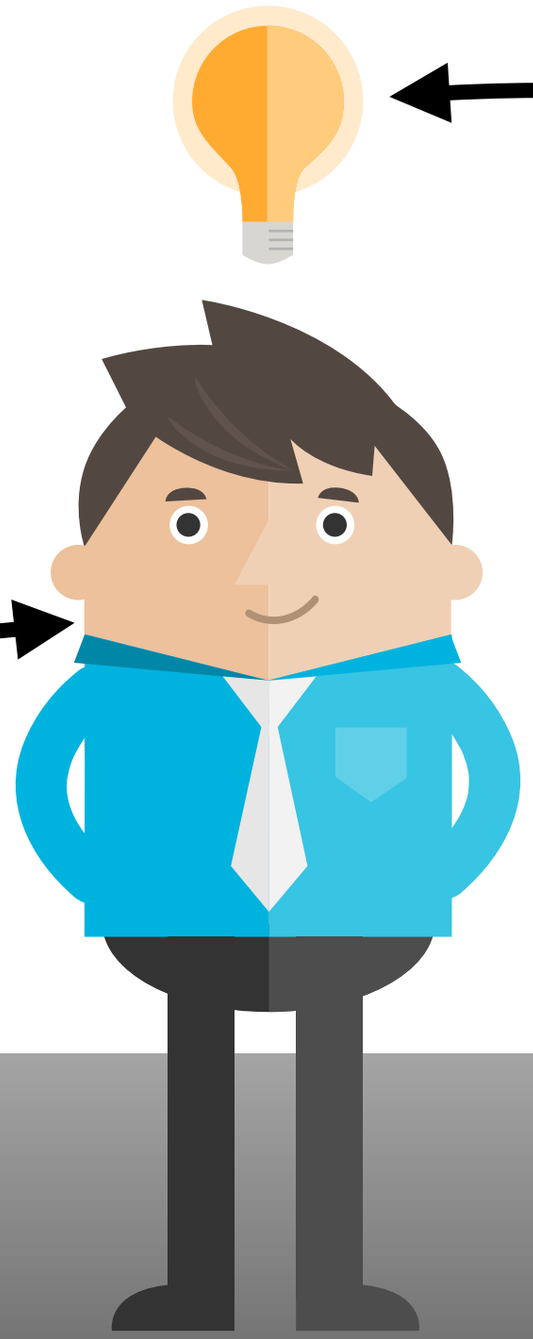
emeryberger.com, @emeryberger

A short time ago in a valley far, far away...

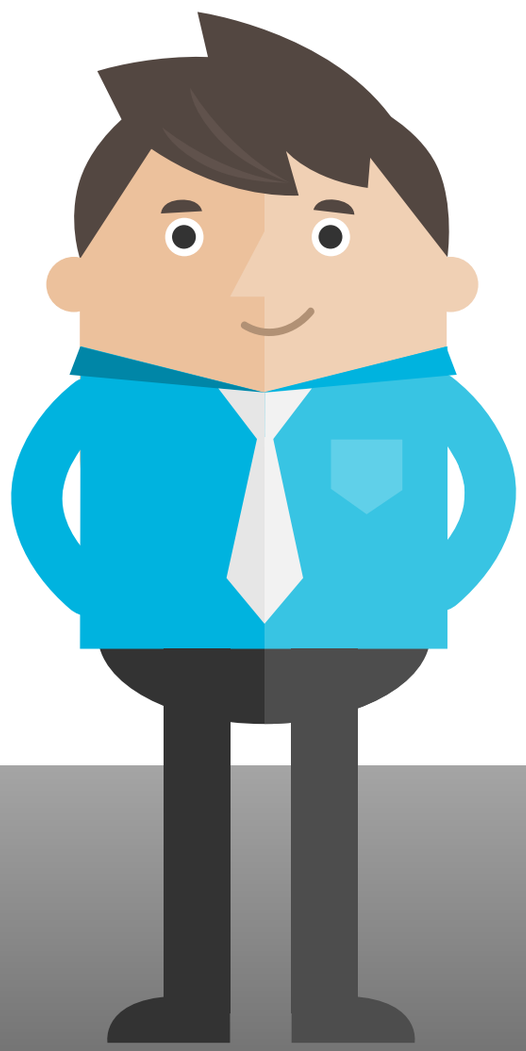


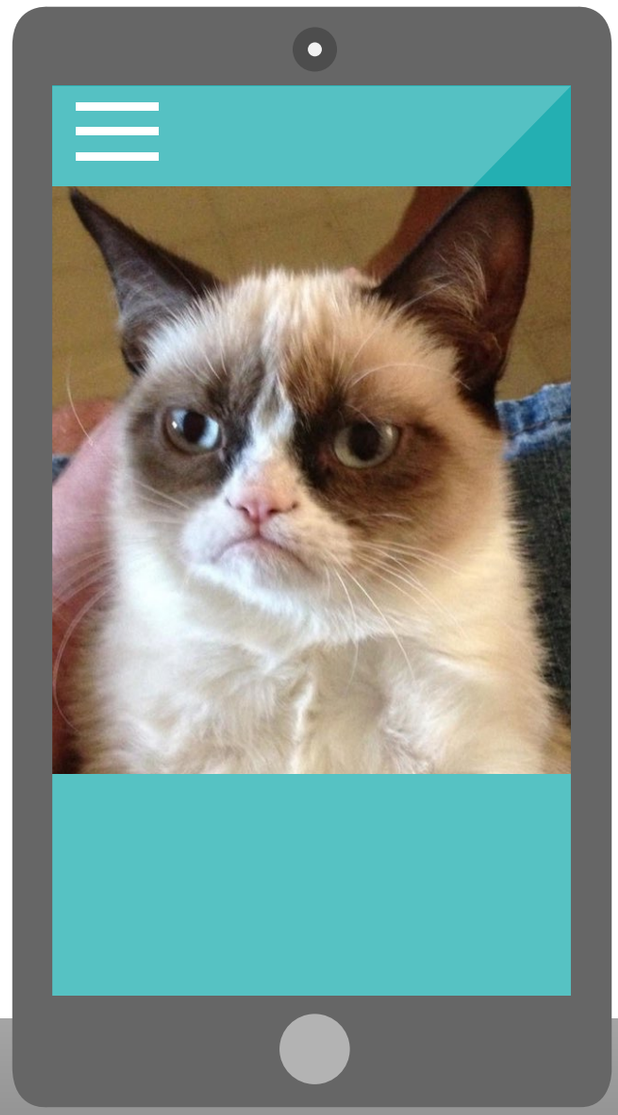

facebook
ORACLE
hooli
hp
SILICON
VALLEY
Google
YouTube

Luke
(our hero)

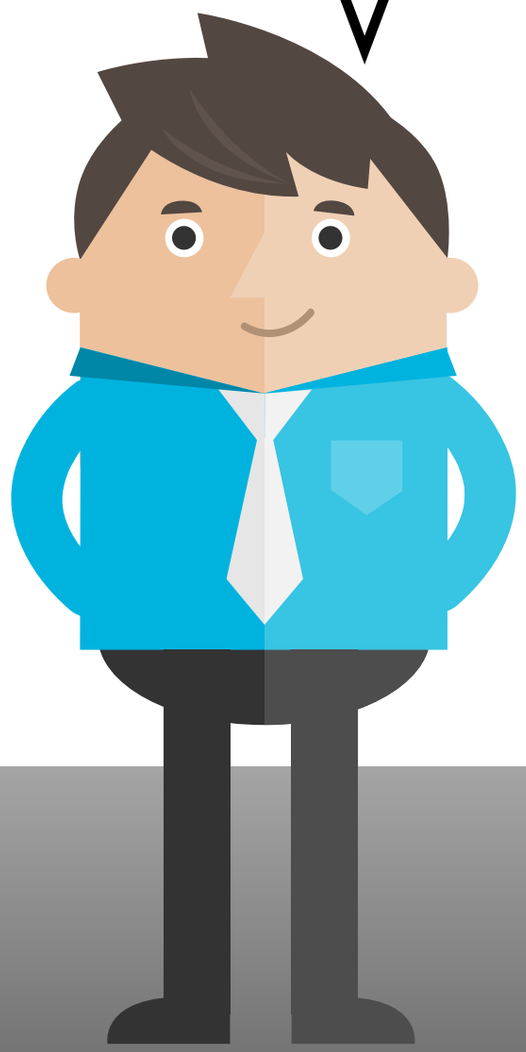


Luke has an idea
for an app.





It’s going to totally disrupt image search.

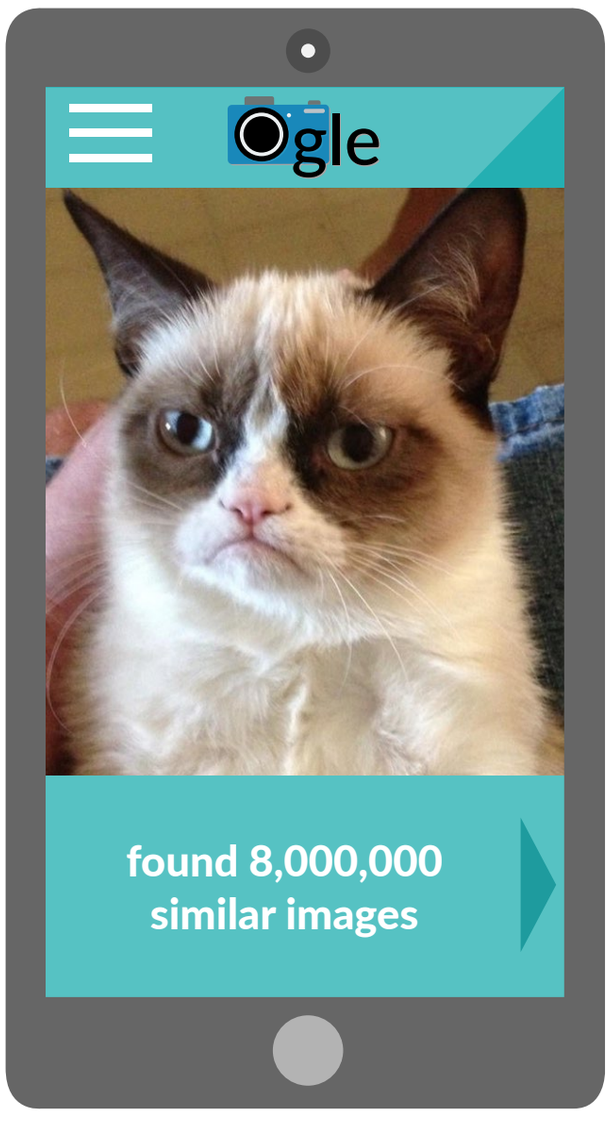


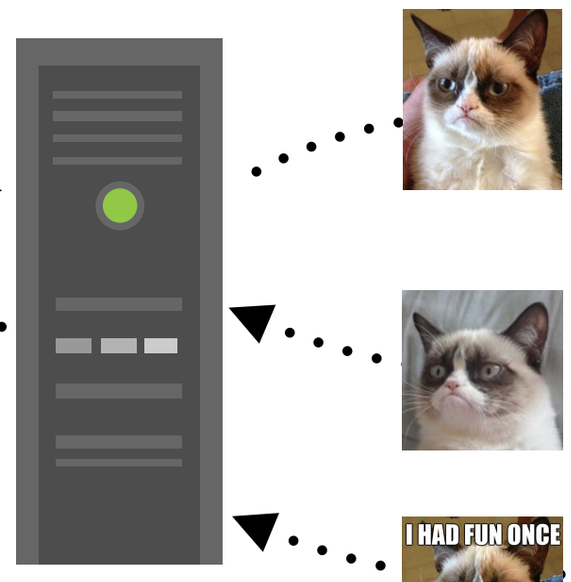
The Prototype Ogle

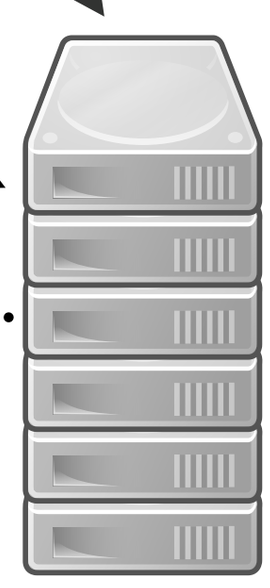
Take a picture

Send it to Ogle

Add it to the database

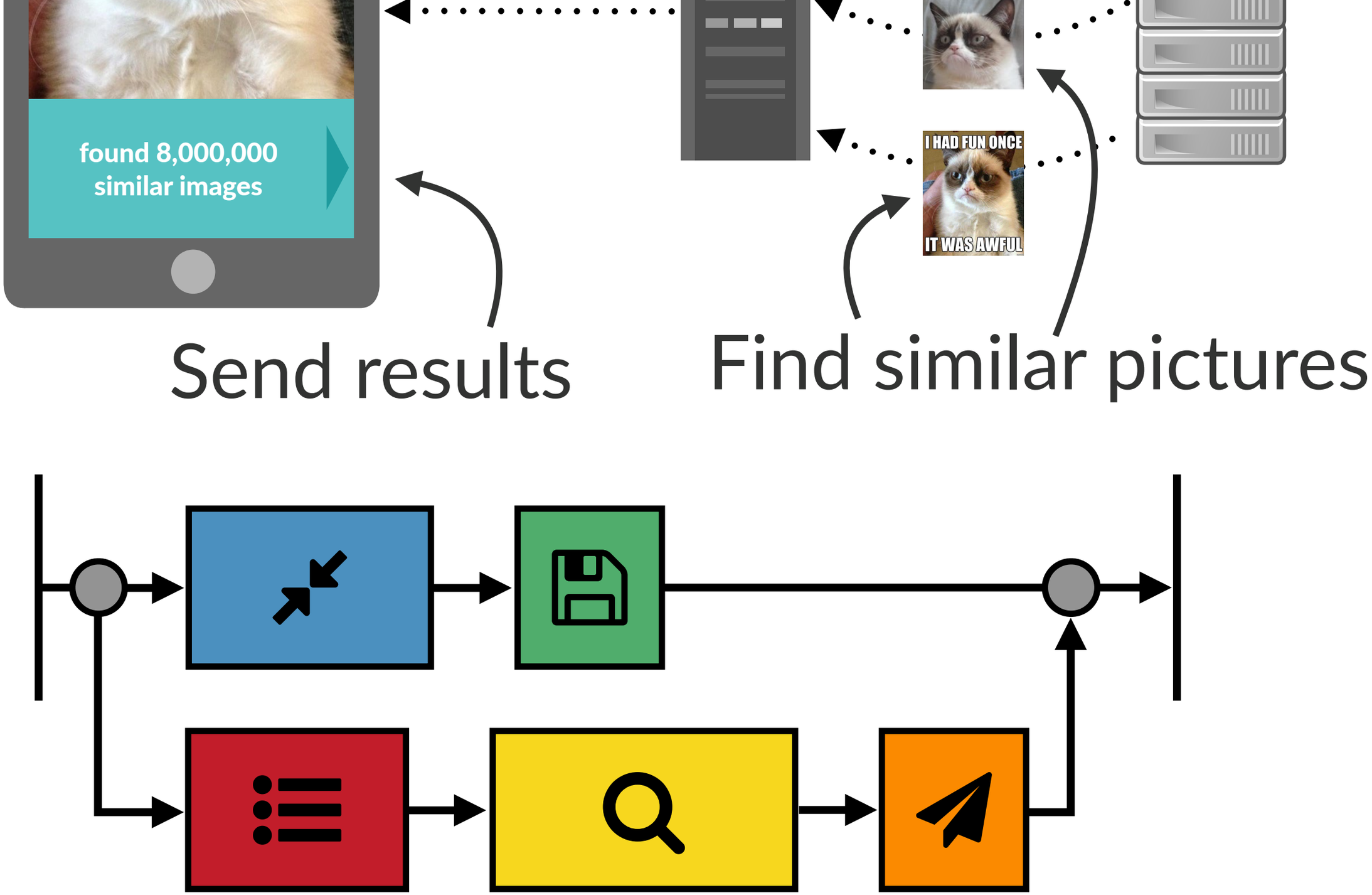


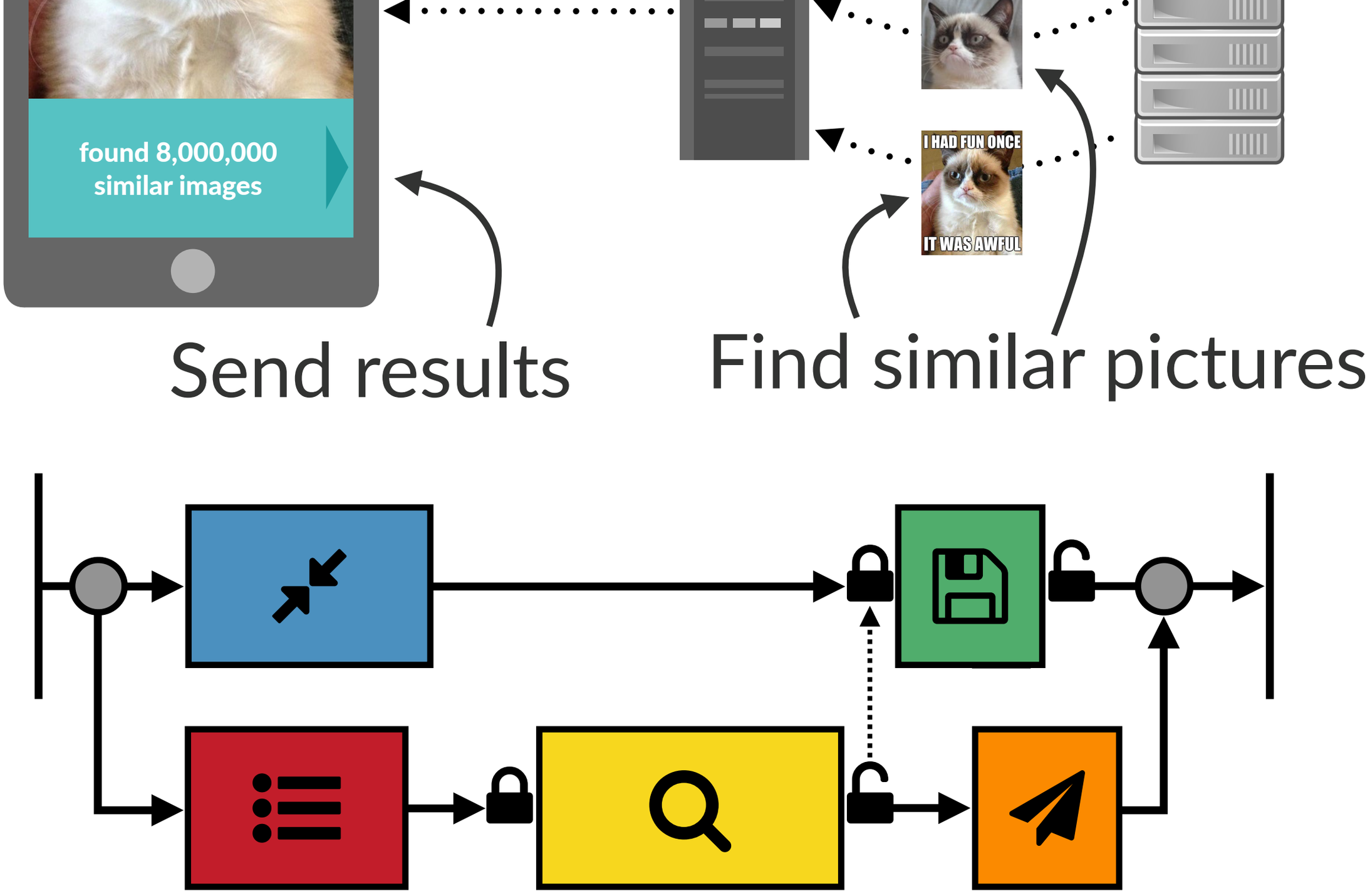




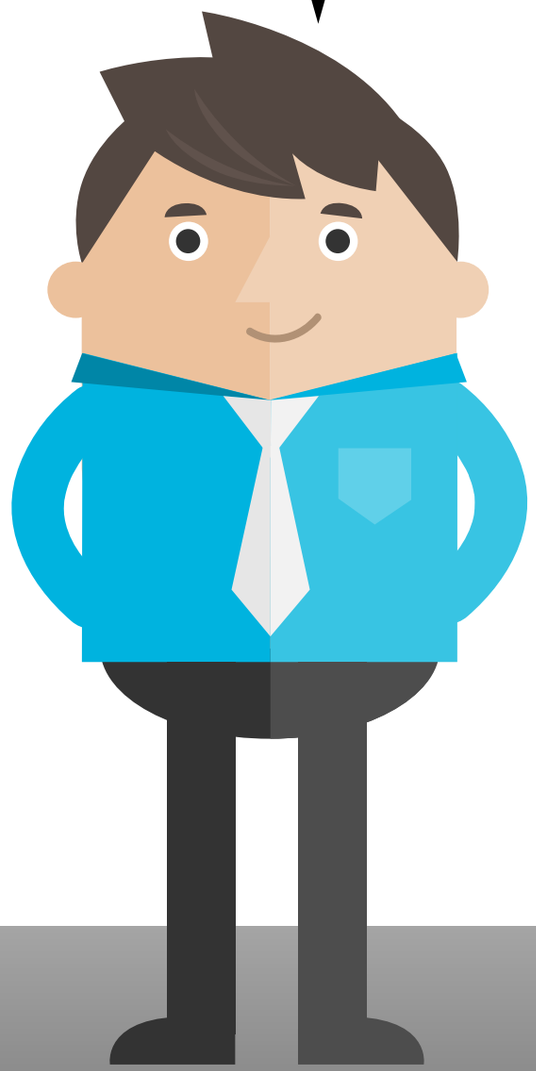
Send results

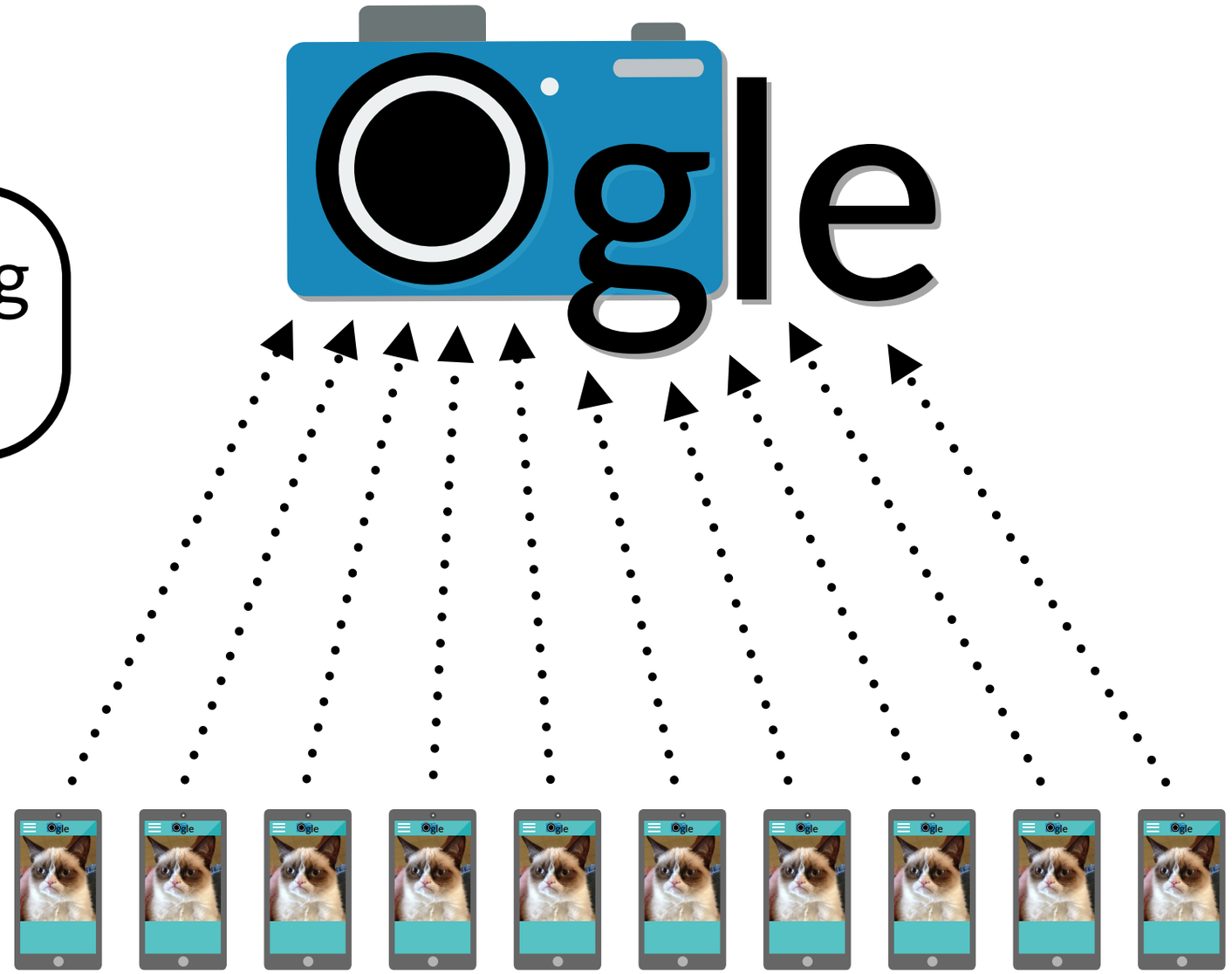
Find similar pictures


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

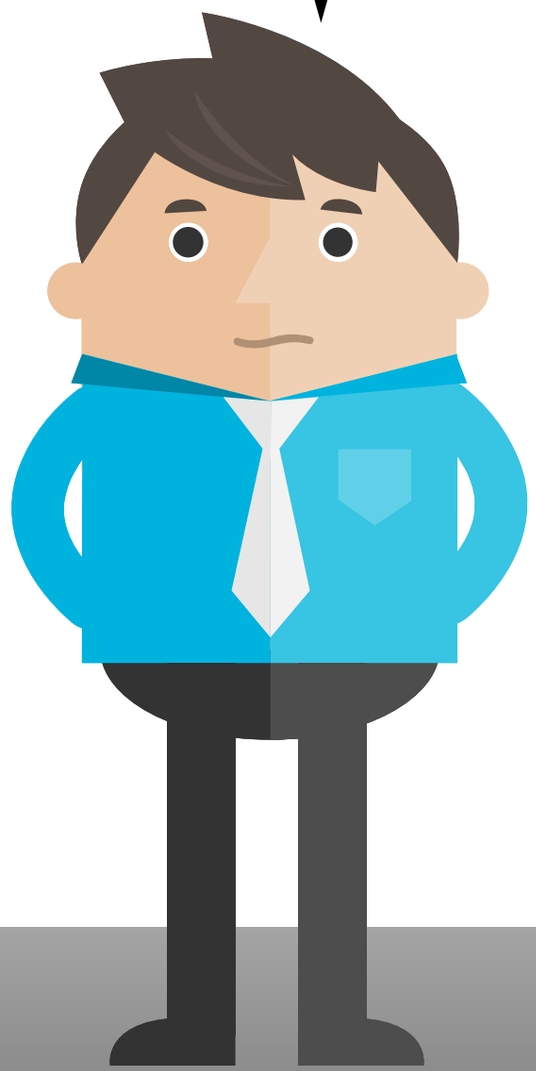

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

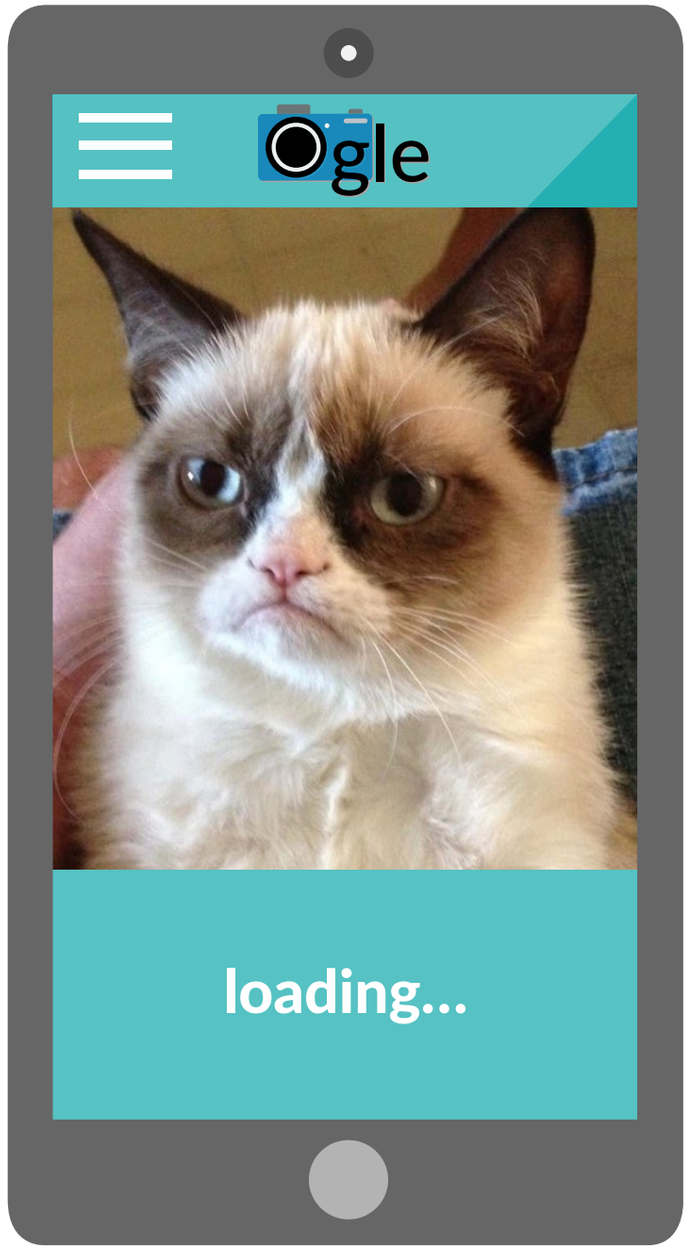
Ogle is totally disrupting image search.





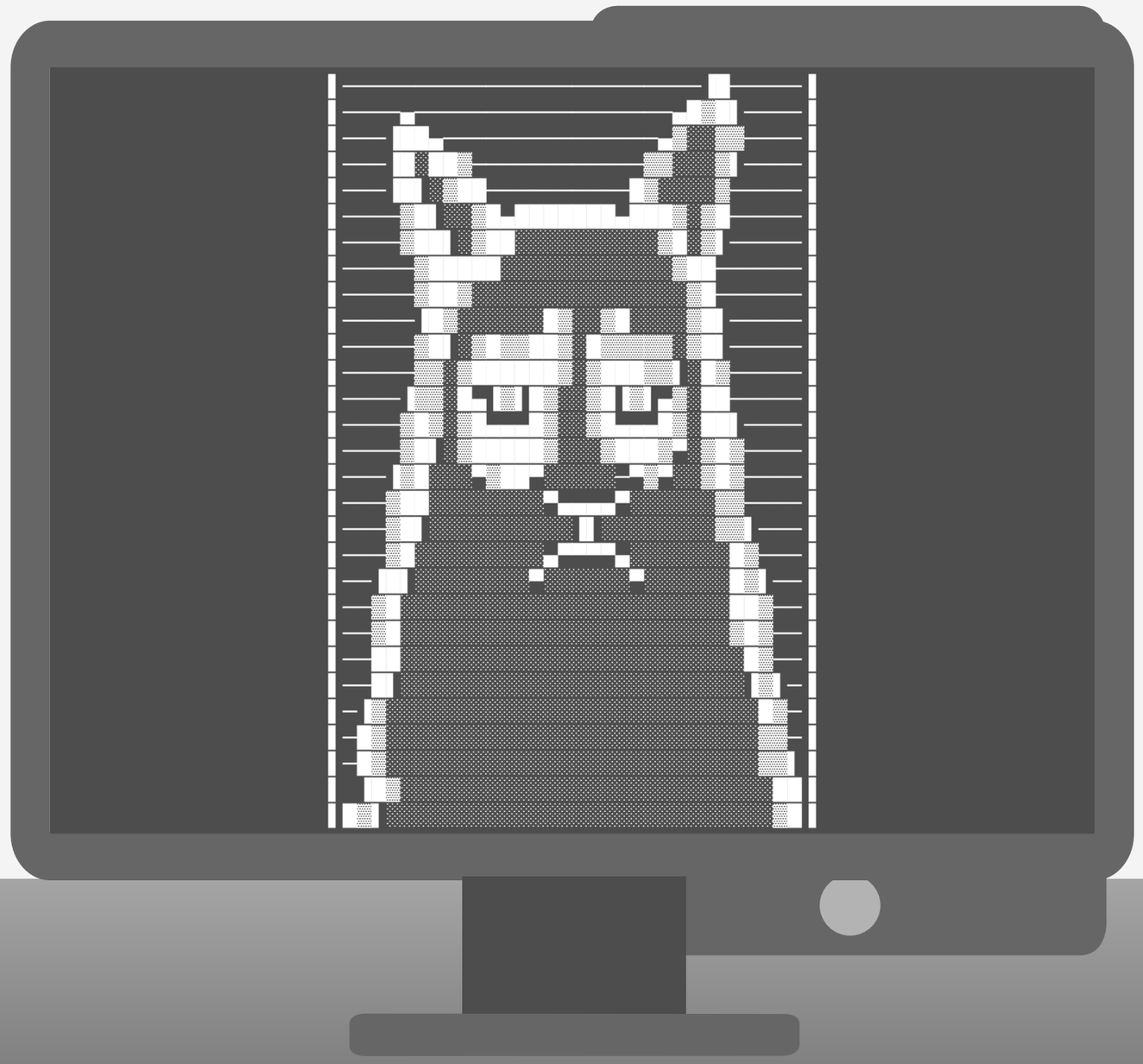
Ogle is too slow!

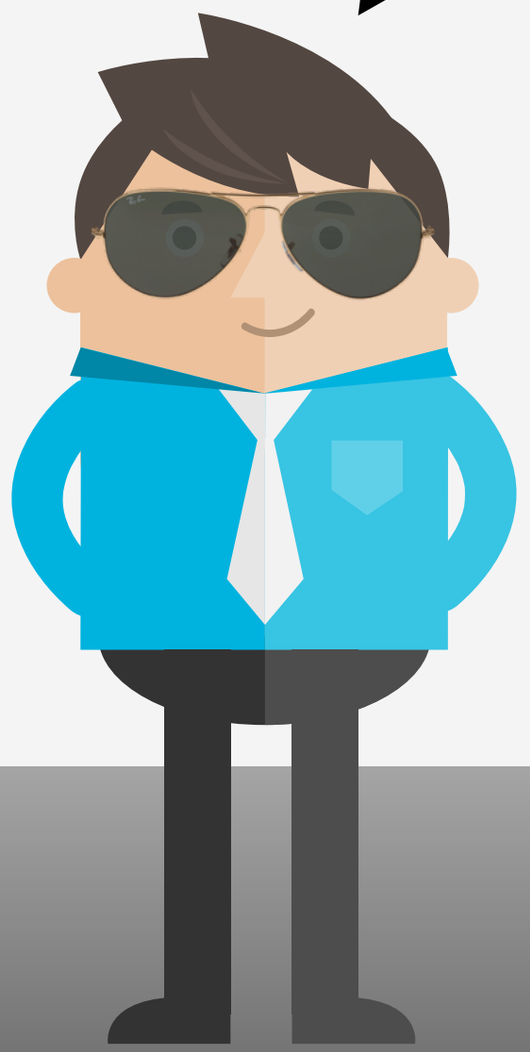




This is also Luke.

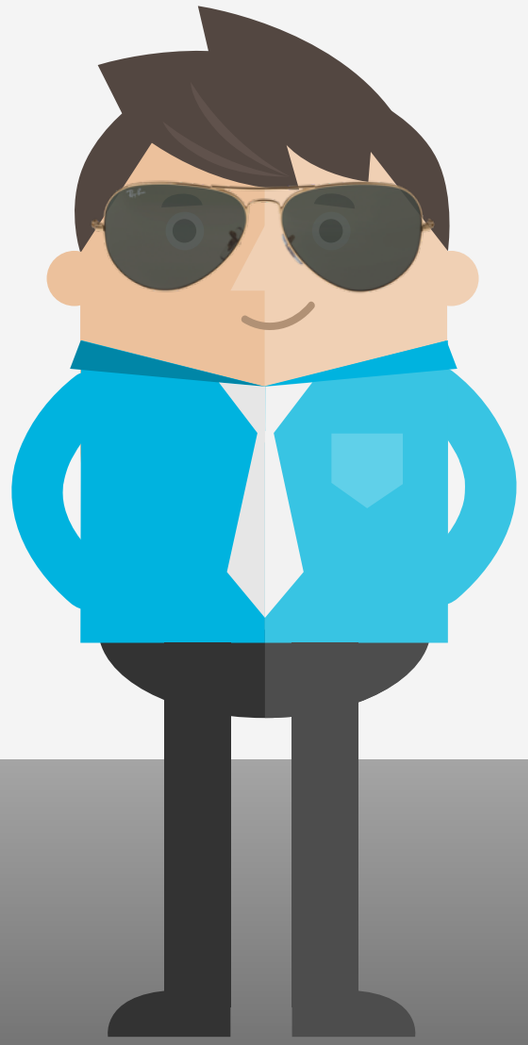
in the ’80s





OGLE '84 1.0

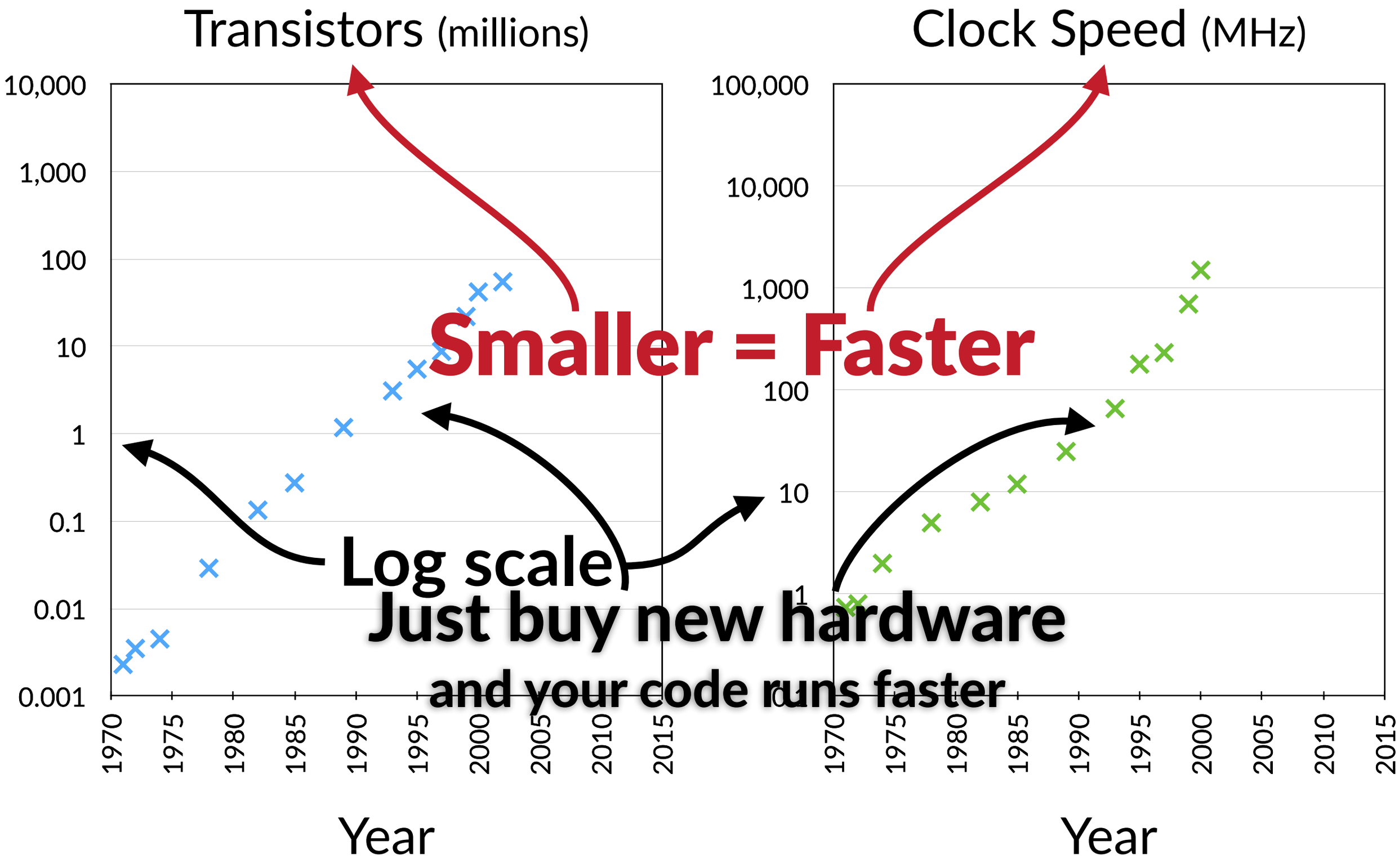
```
Searching database...

found 4 results:
catisgumpy.bmp
funisawful.bmp
catdoesnotlikefun.bmp
from_dad.bmp
```



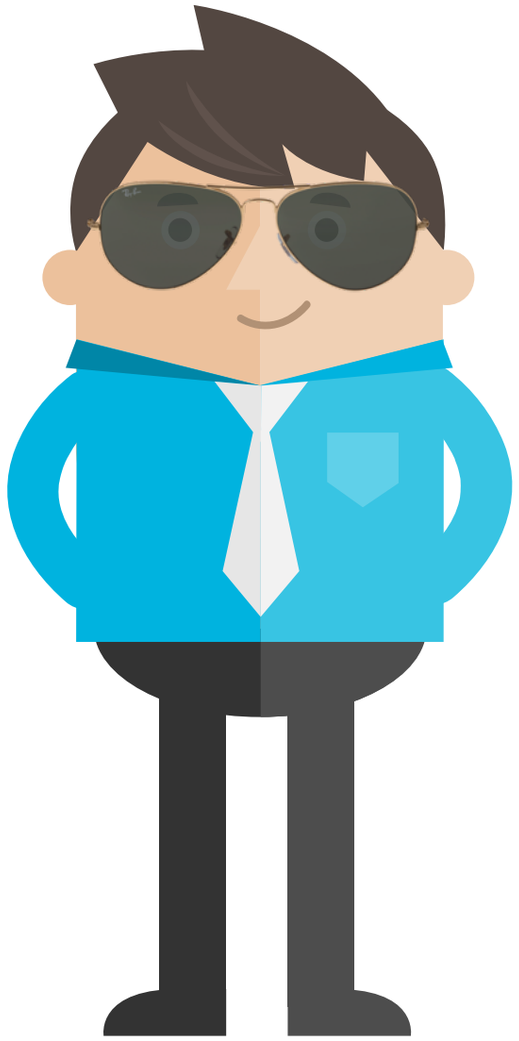
OGLE’84 is too slow!

Ogle is too slow!

Performance used to be easy



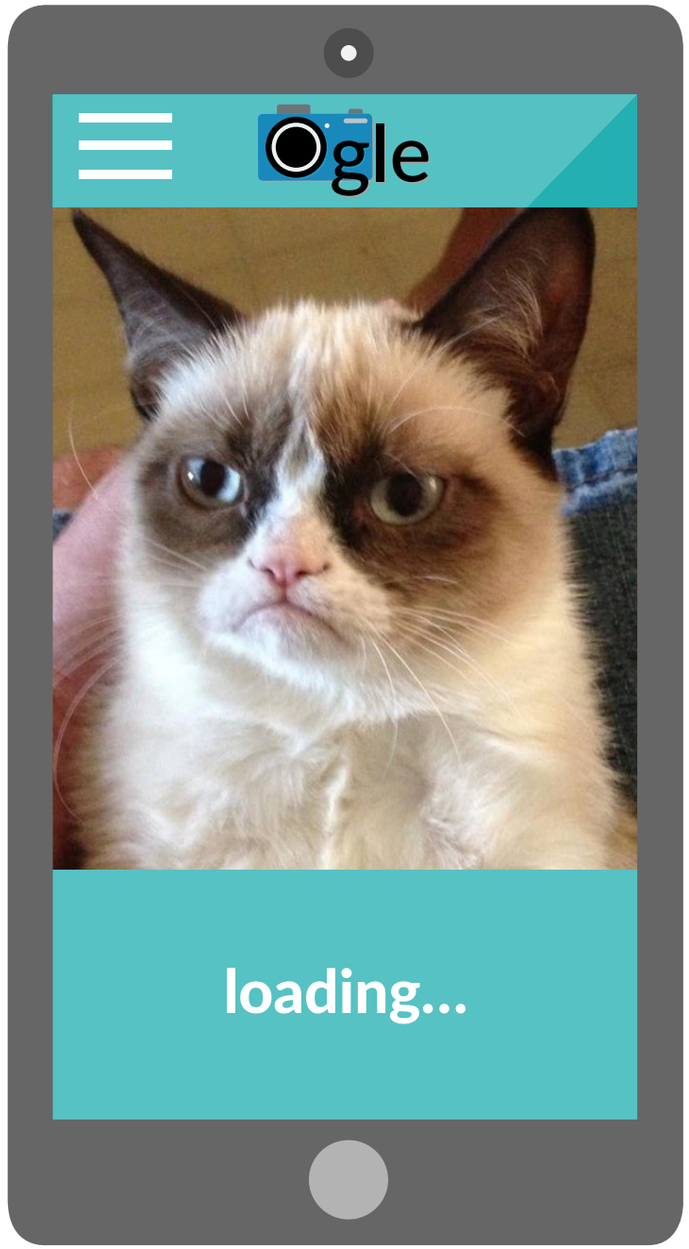
Performance improvement in the ‘80s



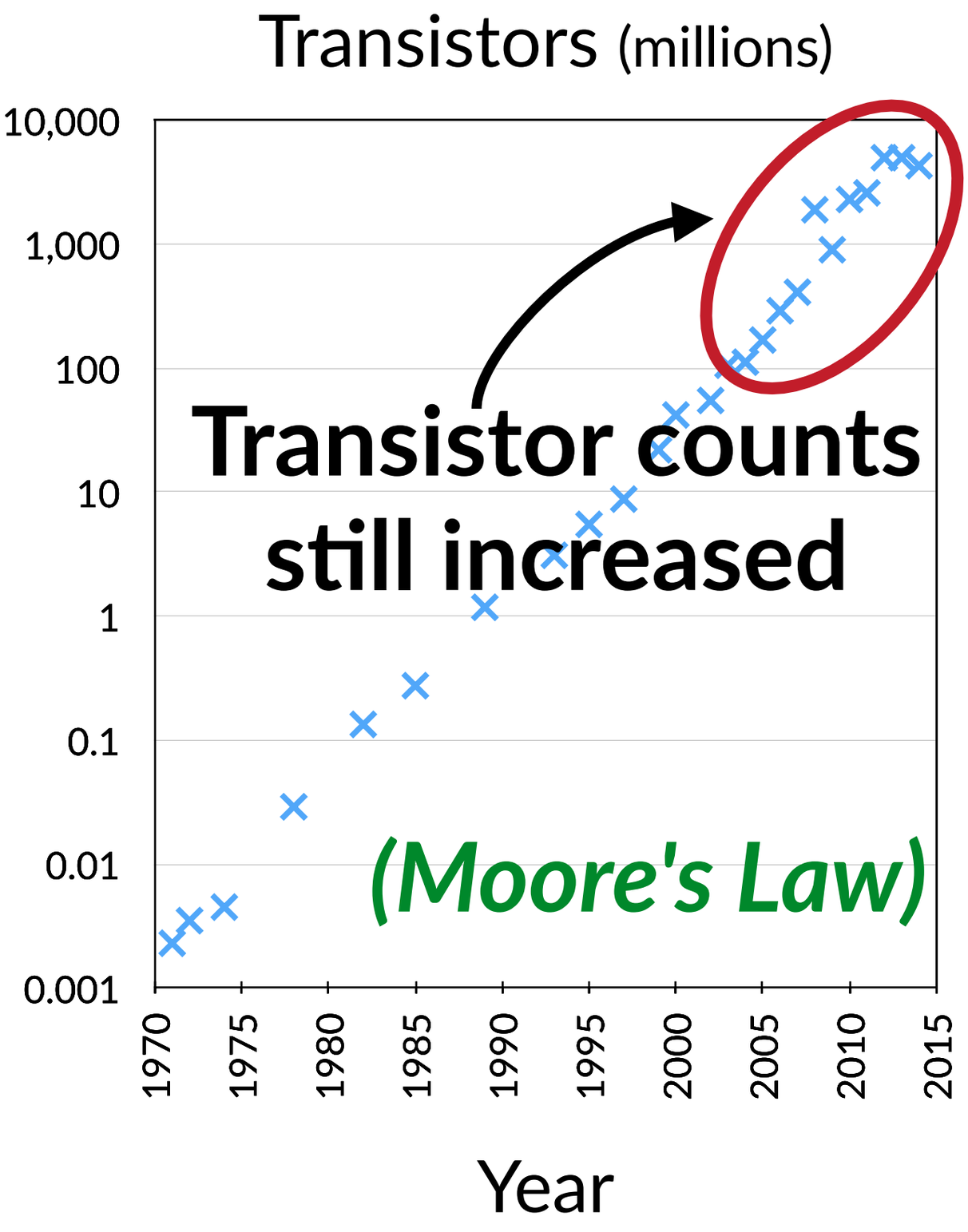
Performance improvement in the ‘80s

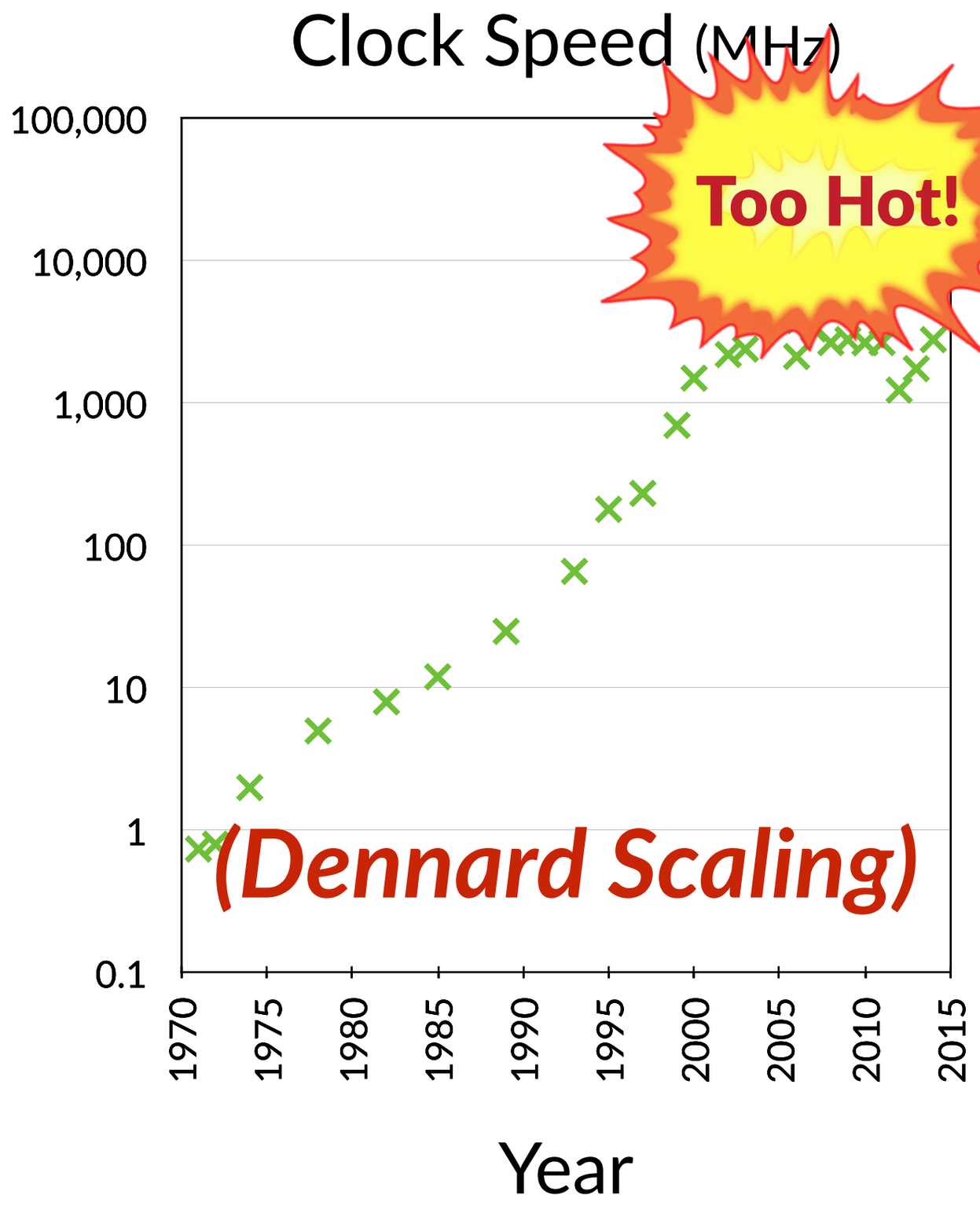
just chill and wait for faster hardware!

Back to the present...



Performance not easy anymore





Dropbox
Today

What's new:
• Under-the-hood updates for performance

We release updates regularly, a
looking for ways to make things
have any feedback or run into i
us in our forums. We're happy

MyFitnessPal
Today

Whether you want to log your l
years, your first run in weeks, c

Google Maps GPS Navigatic
Today

Thanks for using Google Maps!
brings bug fixes and awesome

Google Voice
Mar 17, 2018

- Support for Smart Invert Colors
- Bug fixes and performance imp

Note: This release no longer sup

Signal - Private Messenger
Mar 17, 2018

• You can enable an optional Reg
PIN that will be required in order

JetBlue
Mar 17, 2018

- Updated Flight Tracker
- Updated flight search

Inbox by Gmail

Inbox by Gma
Mar 17, 2018

• Bug fixes and performance i

Gmail - Email by Google
Mar 17, 2018

• You can now change your pa
your profile picture and more I
Account in settings.

Expedia: Hote Flights & Car
Mar 17, 2018

Paper is so 2017. Ditch the pri
paperless with the Expedia Ap
all your travel details in one cc

Bank of Amer Mobile Bankir

Yelp: Discover Local Favorites
Mar 15, 2018

OPEN

We've decided to start referring to "performance improvements" as "app engine calibrations" because it sounds cooler. Therefore: For this version, we've further calibrated the app engine. See what we mean? Also, we fixed a bunch of bugs.

Facebook
Mar 15, 2018

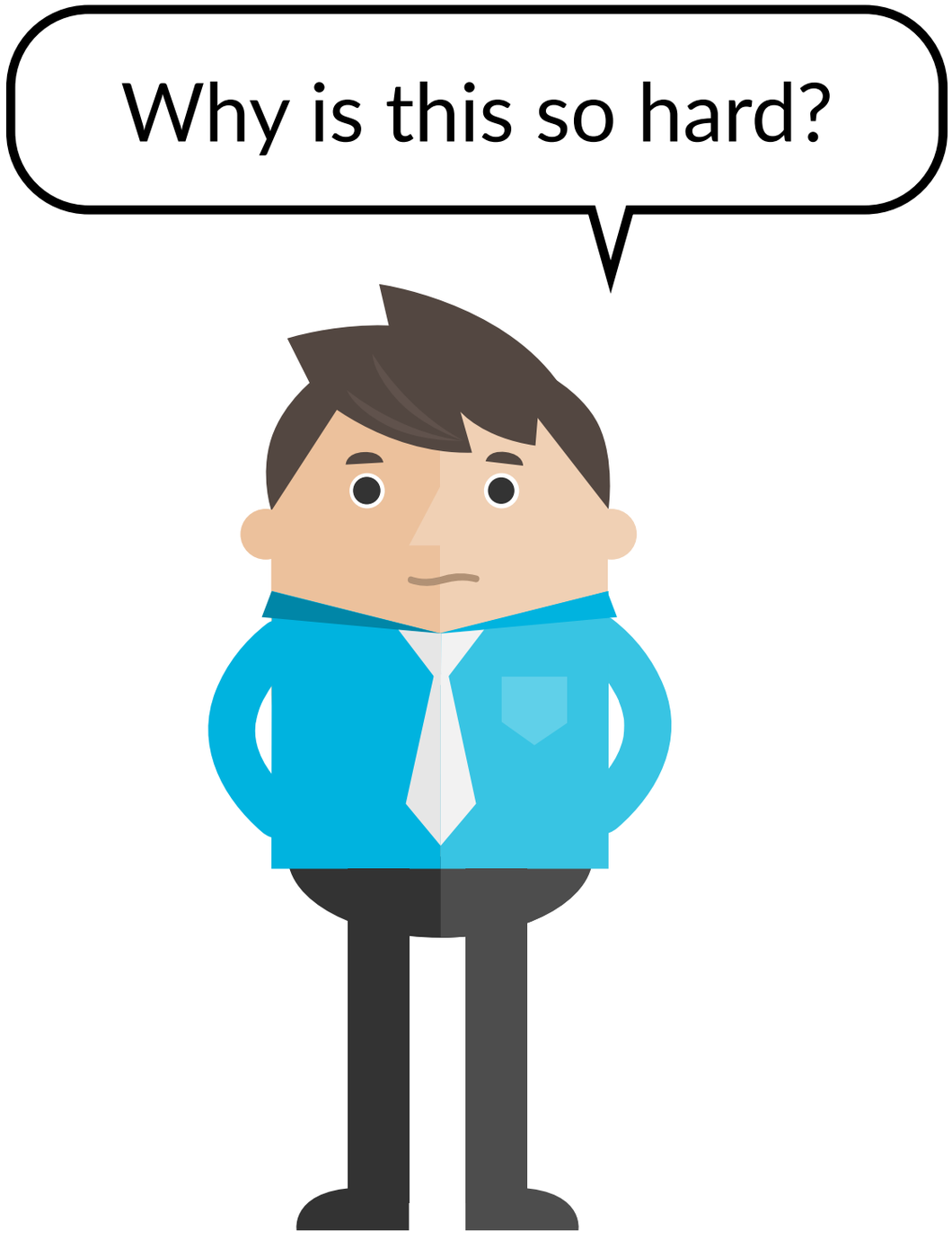
OPEN

Thanks for using Facebook! To make our app better for you, we bring updates to the App Store regularly.

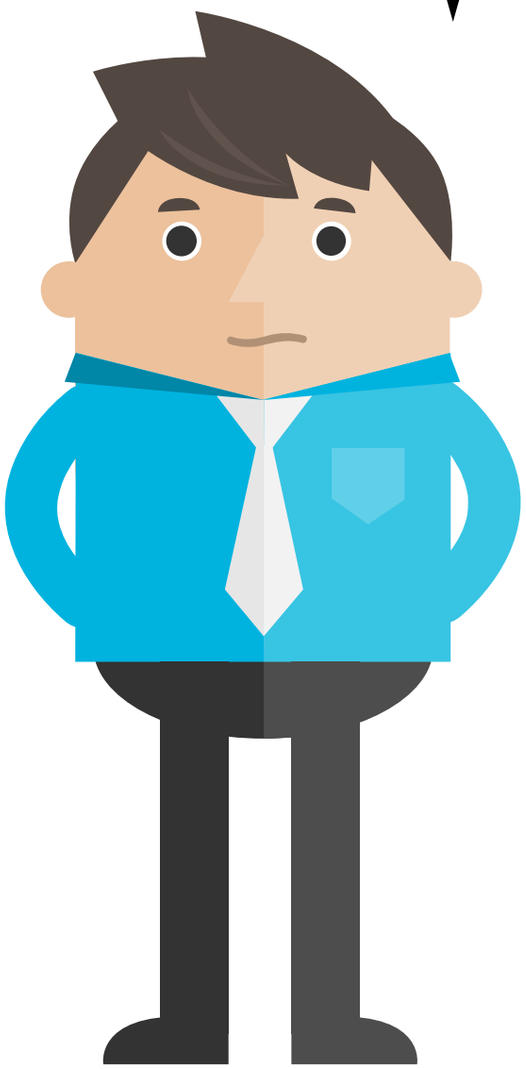
Every update of our Facebook app includes improvements for speed and reliability. As new features become available, we'll highlight those for you in the app.

WhatsApp Messenger

OPEN


Why is this so hard?

Why is this so hard?

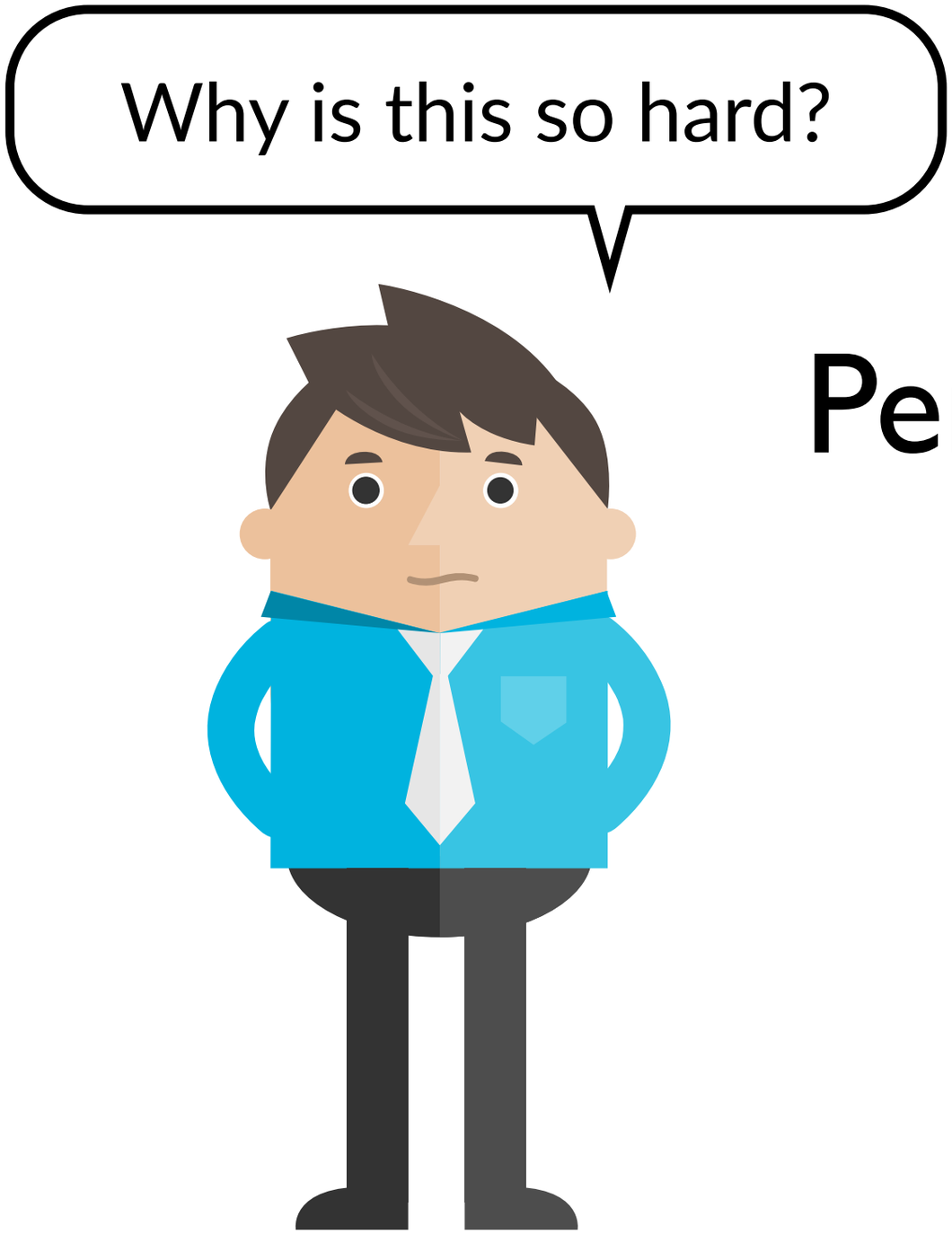


Performance Analysis

how to do it right

Performance Profiling

how to do it better



Performance Analysis

how to do it right

Typical performance evaluation

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Typical performance evaluation

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Typical performance evaluation

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Typical performance evaluation

A

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

Typical performance evaluation

A

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Typical performance evaluation

A

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Typical performance evaluation

A

A′

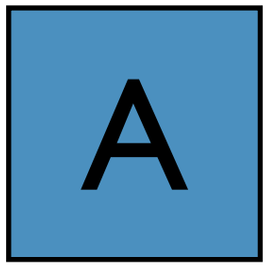
```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

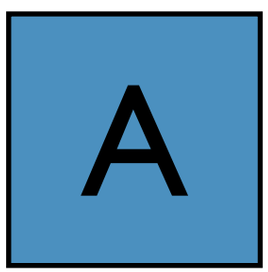
```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

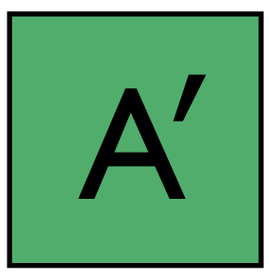
Which is faster?



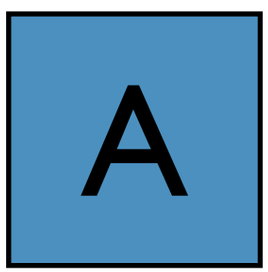
Which is faster?



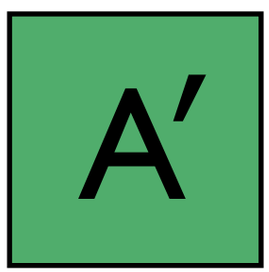
90s



Which is faster?



90s



87.5s

Which is faster?

A

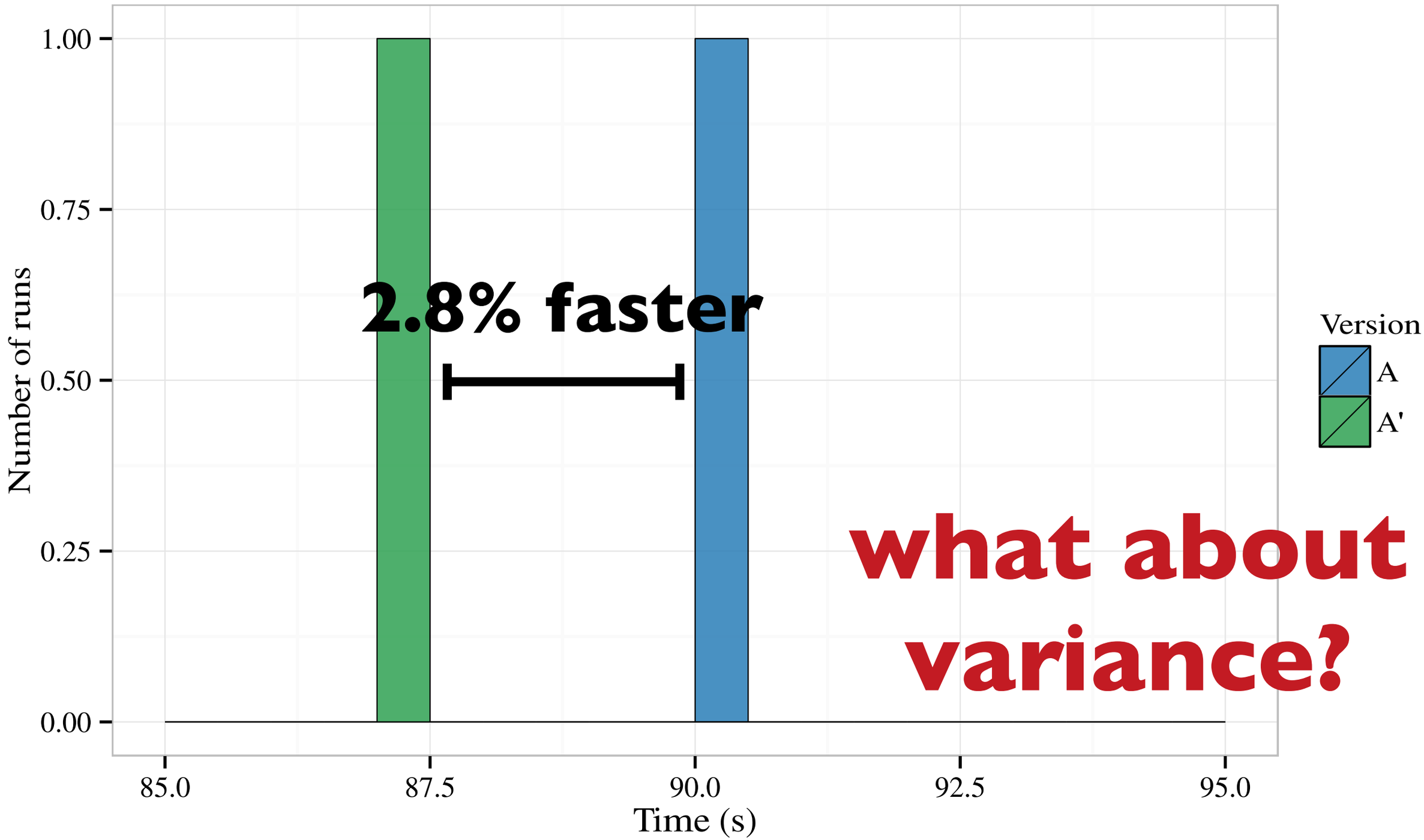
90s

A′

87.5s

A′ **2.8% faster!**

Is A′ faster than A ?

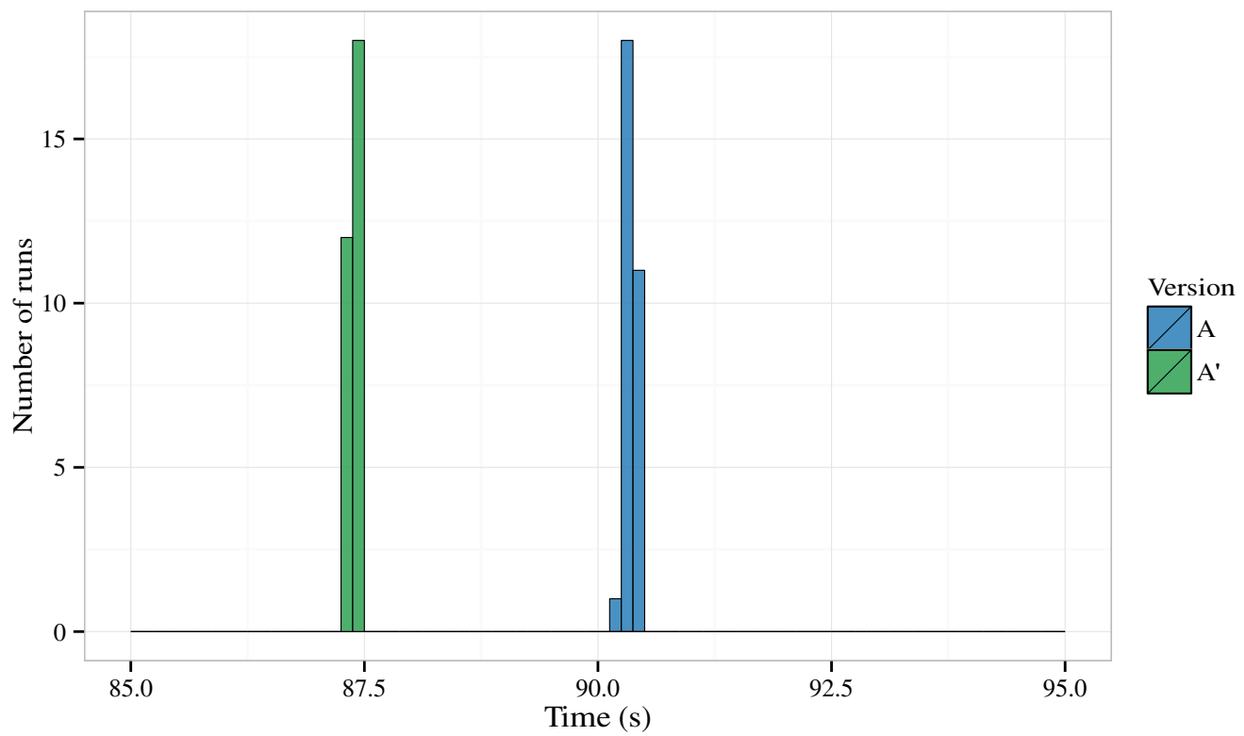


what about variance?

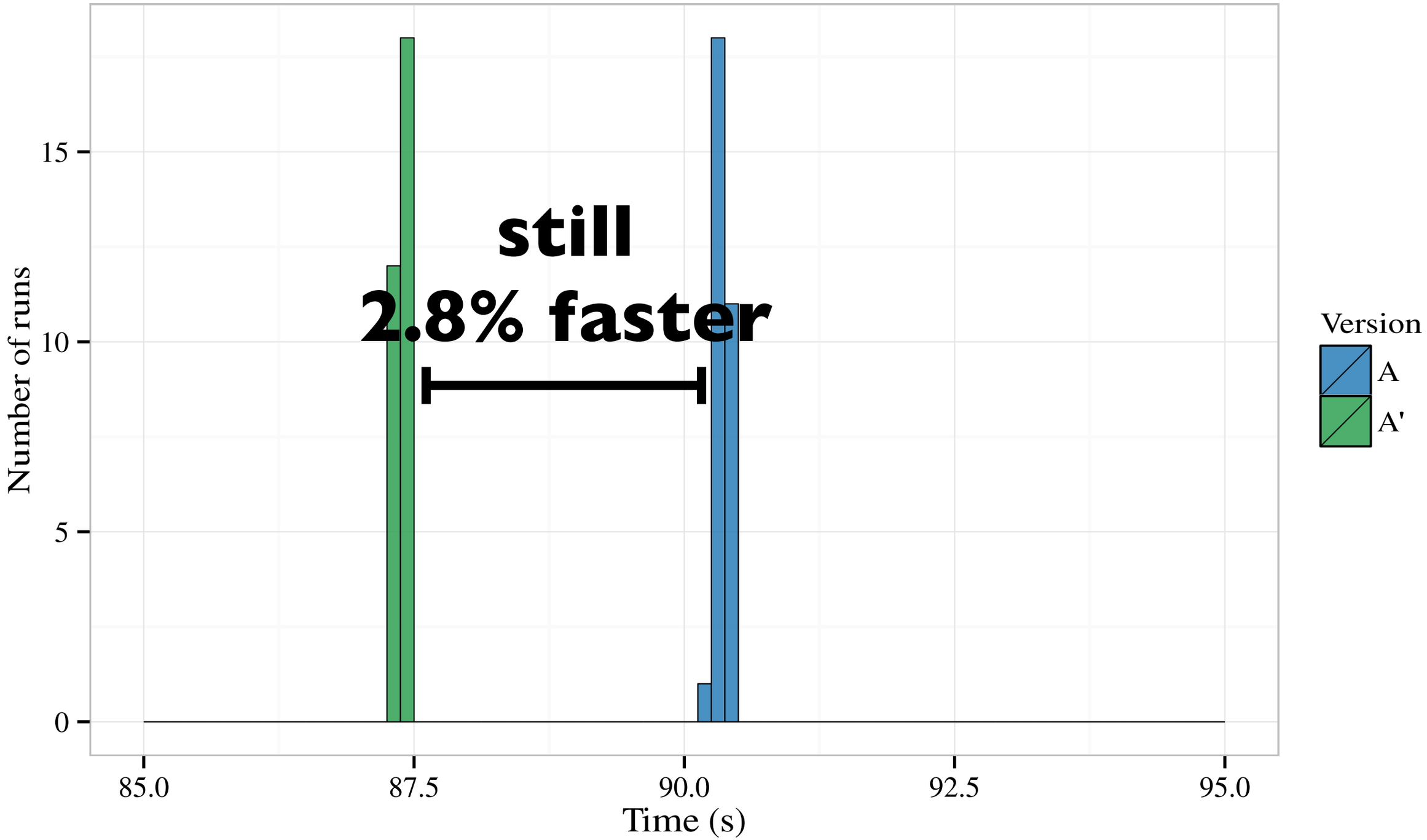
Which is faster?

A′ ×30

A ×30



Is A′ faster than A ?



Why is A′ faster than A ?

Why is A′ faster than A ?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A ?

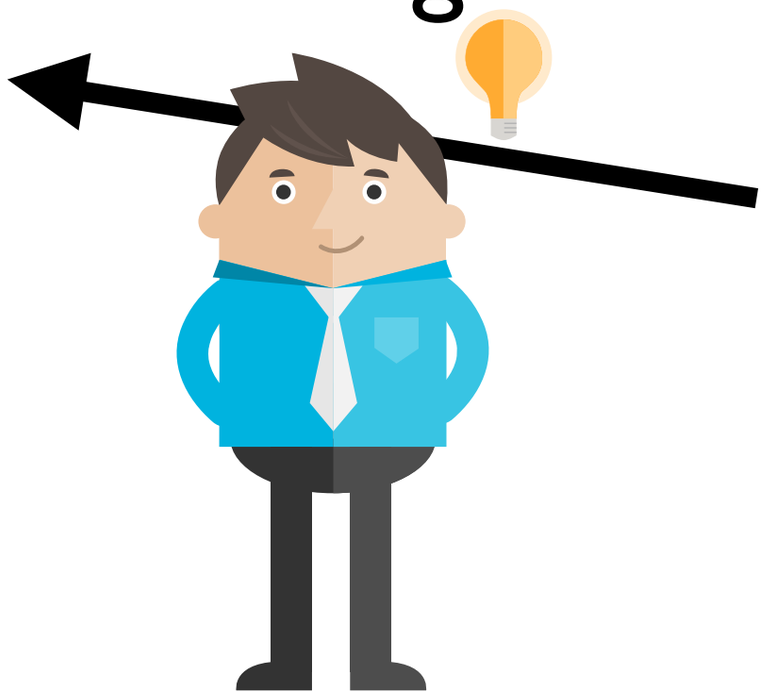
```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

The code change!



```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A ?

Or was it
an accident?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A ?

Layout biases measurement

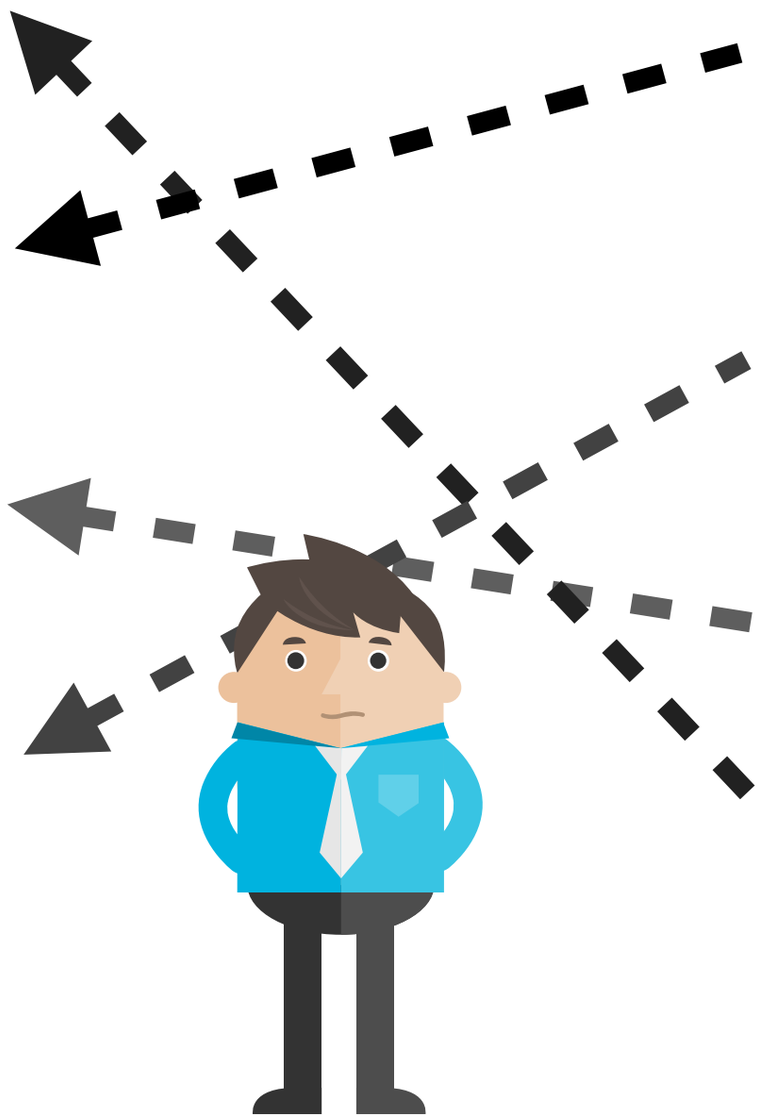
Mytkowicz et al. (ASPLOS'09)

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```



```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Producing Wrong Data Without Doing Anything Obviously Wrong!

Todd Mytkowicz Amer Diwan
Department of Computer Science
University of Colorado
Boulder, CO, USA
{mytkowit,diwan}@colorado.edu

Matthias Hauswirth
Faculty of Informatics
University of Lugano
Lugano, CH
Matthias.Hauswirth@unisi.ch

Peter F. Sweeney
IBM Research
Hawthorne, NY, USA
pfs@us.ibm.com

Link Order

Changes function addresses

Environment Variable Size

Moves the program stack

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Link Order

Changes function addresses

Larger than the impact of -O3! (±40%)

Environment Variable Size

Moves the program stack

Why is A′ faster than A ?

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A ?

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to same cache set **conflict**

Why is A′ faster than A ?

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Why is A′ faster than A ?

A

A′

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to same set

Nothing here

no conflict

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

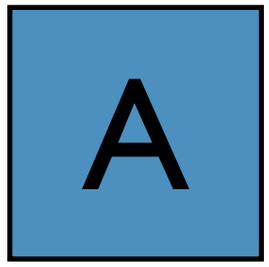
It's the cache

or TLB

or branch predictor

or branch target predictor

or prefetcher




```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Might be faster *today*…

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

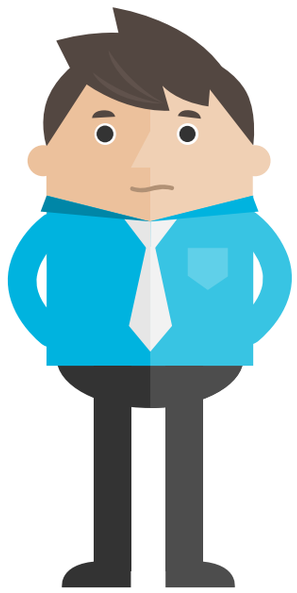
```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

One more new

Changes layout



Might be faster *today*…

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

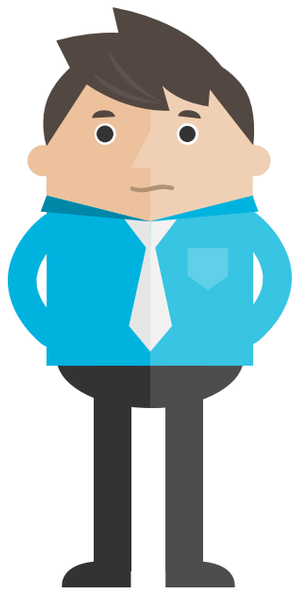
```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Change one `class`

Changes layout



Might be faster *today*…


```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

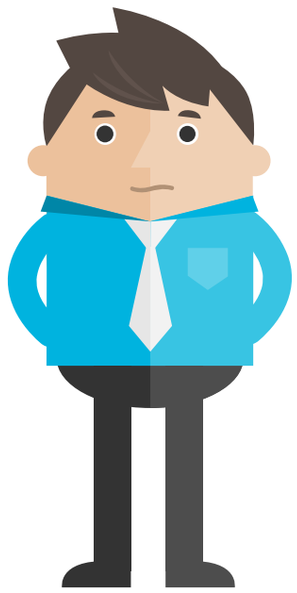
```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Upgrade `libc`

Changes layout



Might be faster *today*…

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

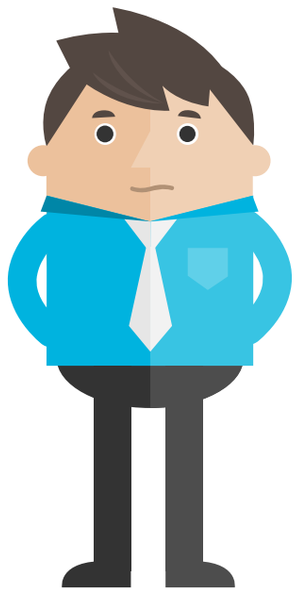
```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Run in a new directory

Changes layout



```
% pwd
/users/vader
% cd /users/luke
```

Might be faster *today*…

A′

same issues for perf regression!

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

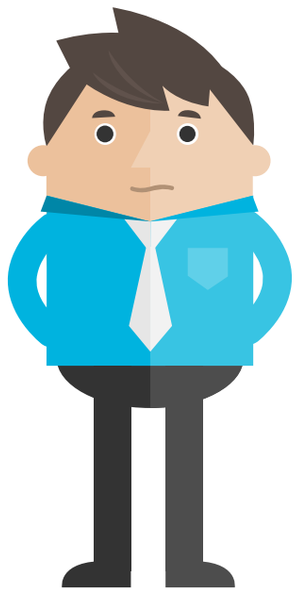
```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Change Username

Changes layout



```
% sudo -u luke
```

Layout is Brittle

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Can we eliminate the effect of layout?

Layout biases measurement

Can we eliminate the effect of layout?

YES

with *randomization*

Memory layout affects performance

makes performance evaluation difficult

Use randomization to eliminate the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

STABILIZER

Memory layout affects performance

makes performance evaluation difficult

Use randomization to eliminate the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

STABILIZER

randomizes layout

function addresses stack frame sizes

heap allocations

Layout biases measurement

STABILIZER

during execution

repeatedly randomizes layout

function addresses

stack frame sizes

heap allocations

Layout biases measurement

STABILIZER

repeatedly randomizes layout

function addresses stack frame sizes

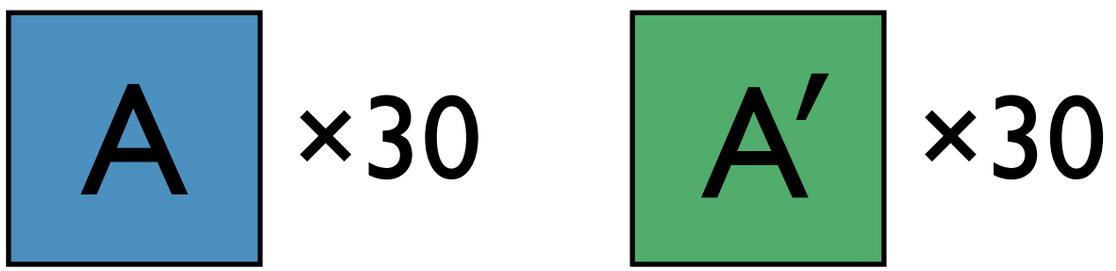
heap allocations

~~**Layout biases measurement**~~

completely random layout

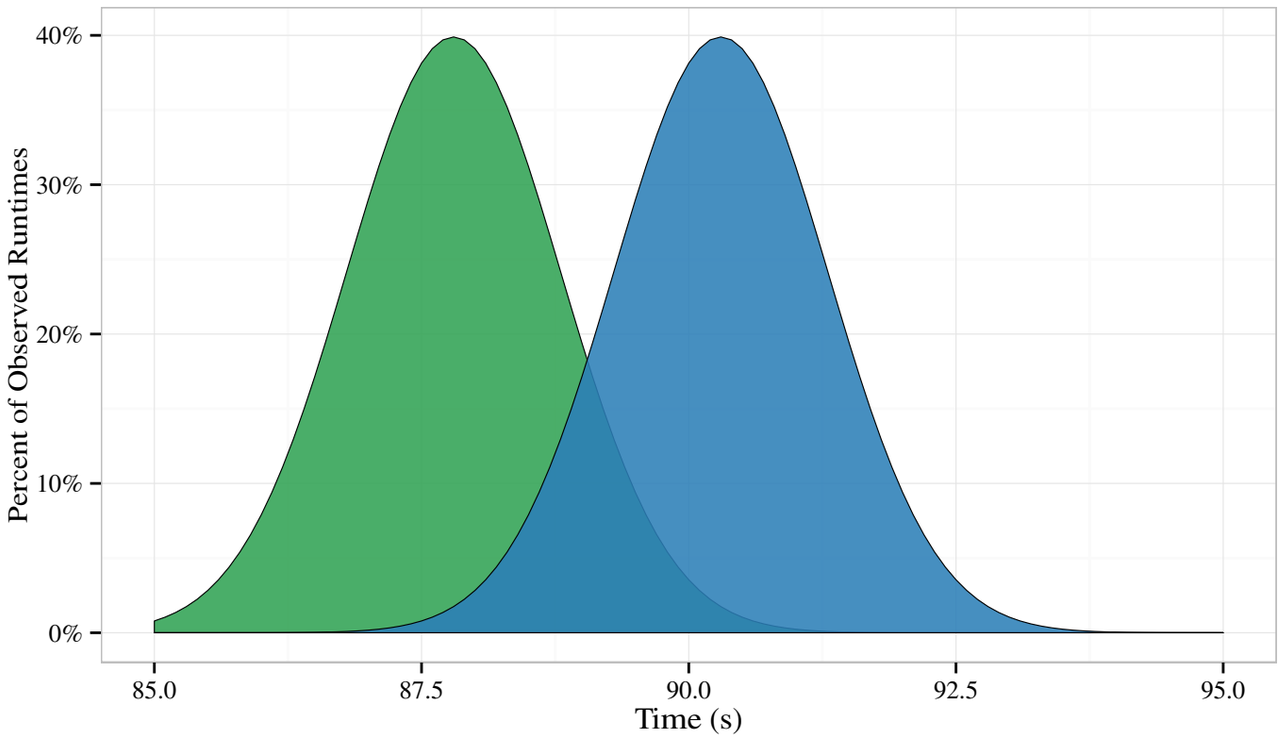
cannot bias results

Sound Performance Evaluation

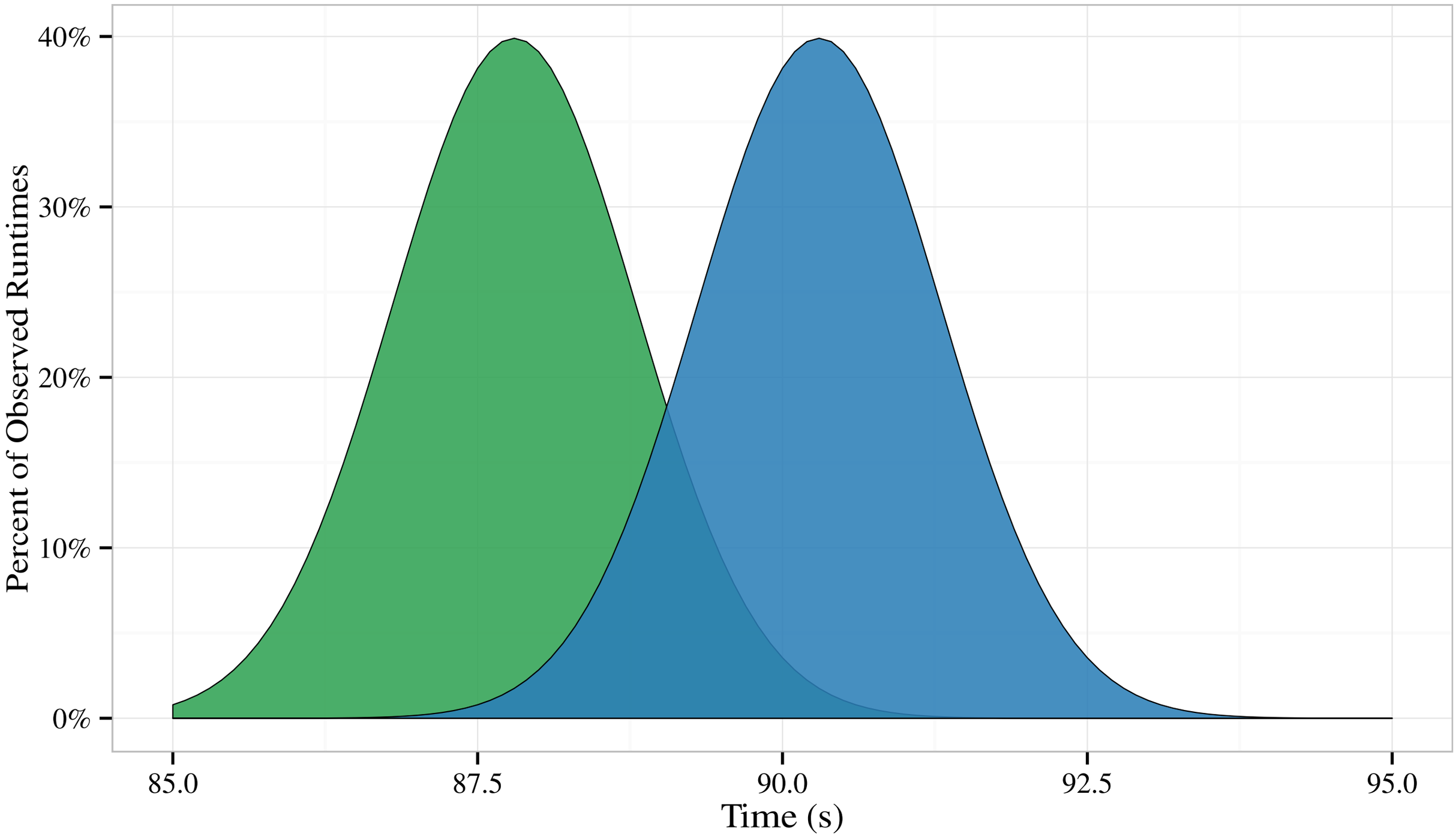


Sound Performance Evaluation

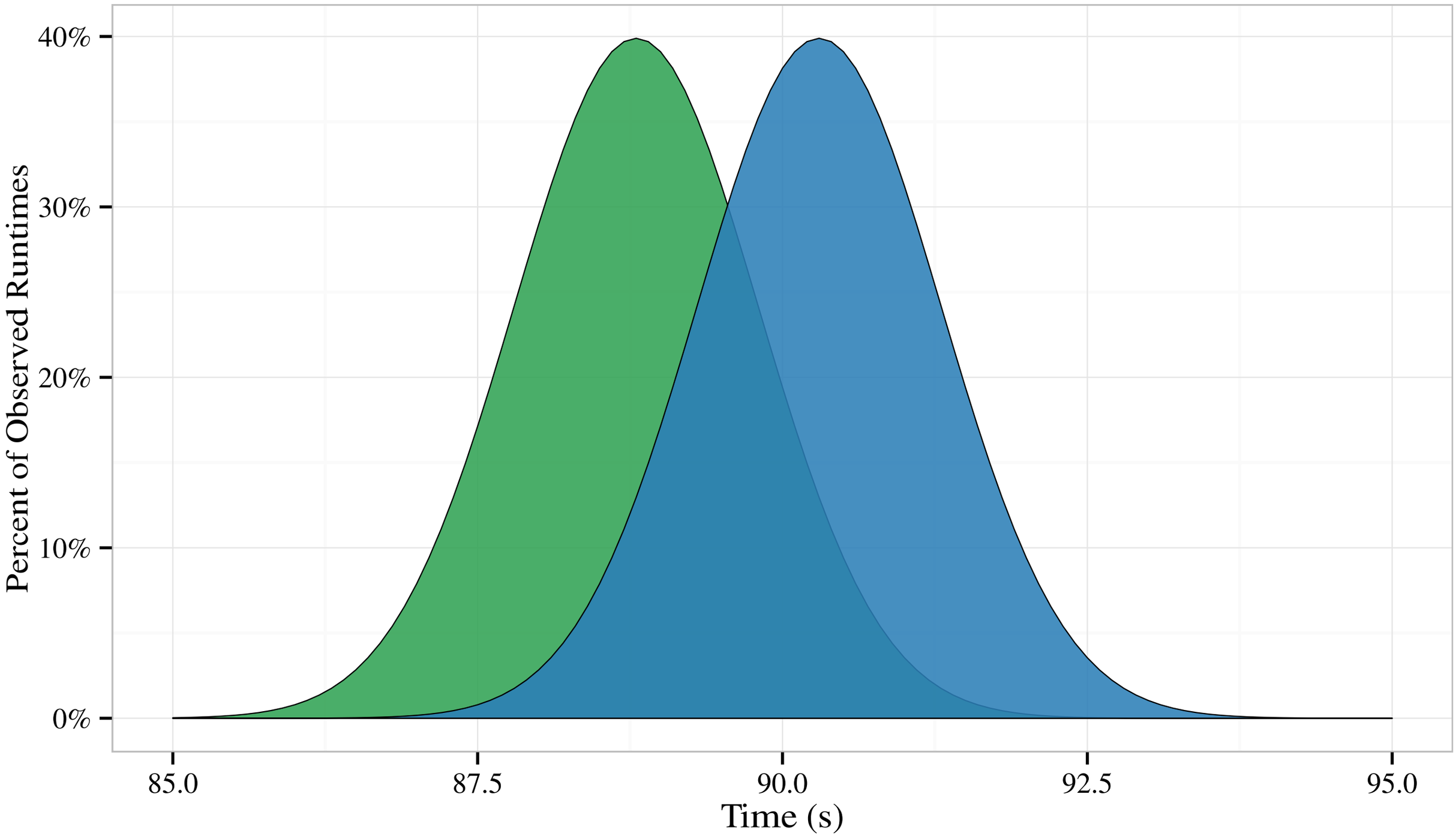
A ×30 A′ ×30



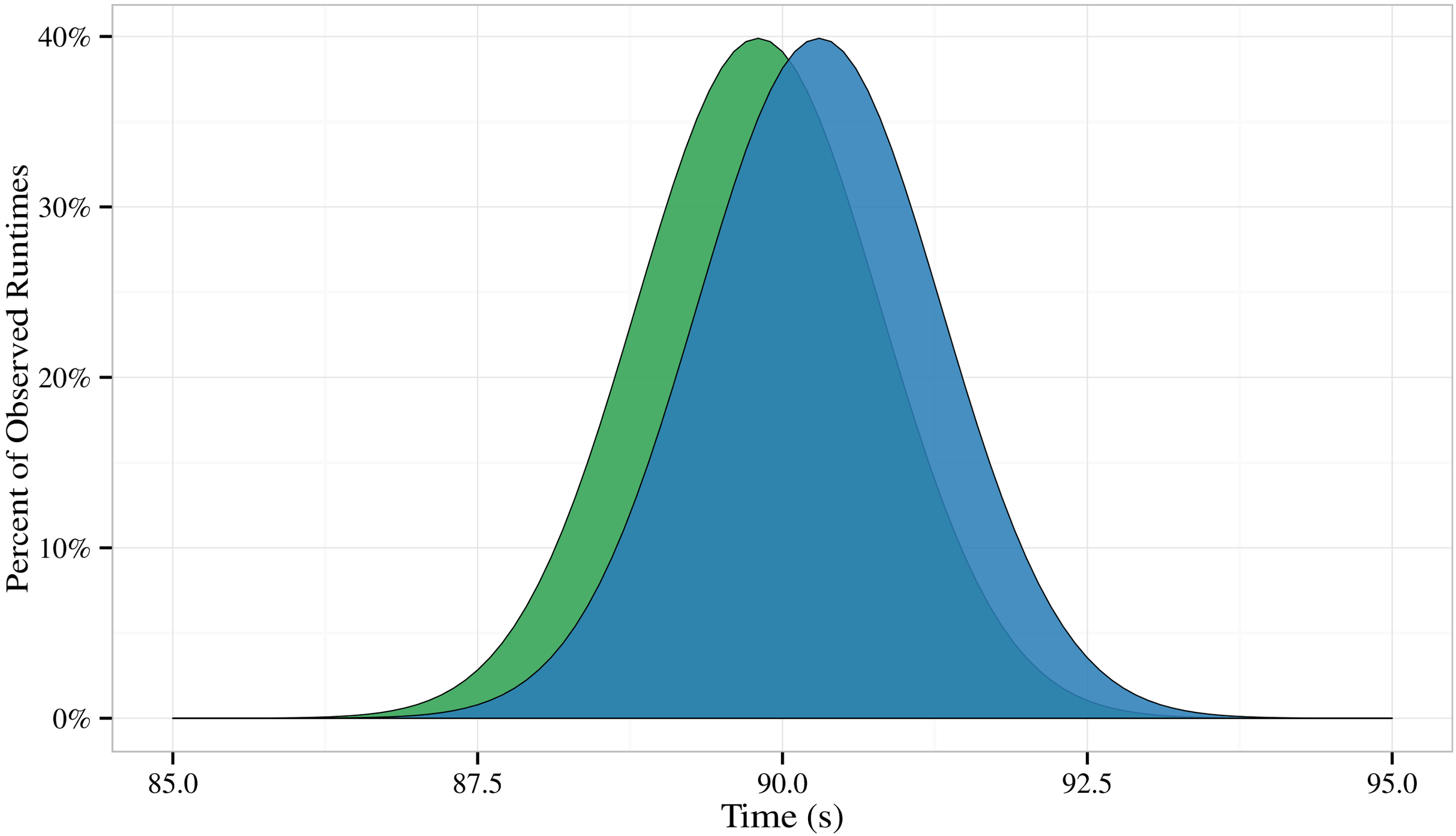
Is A′ faster than A?



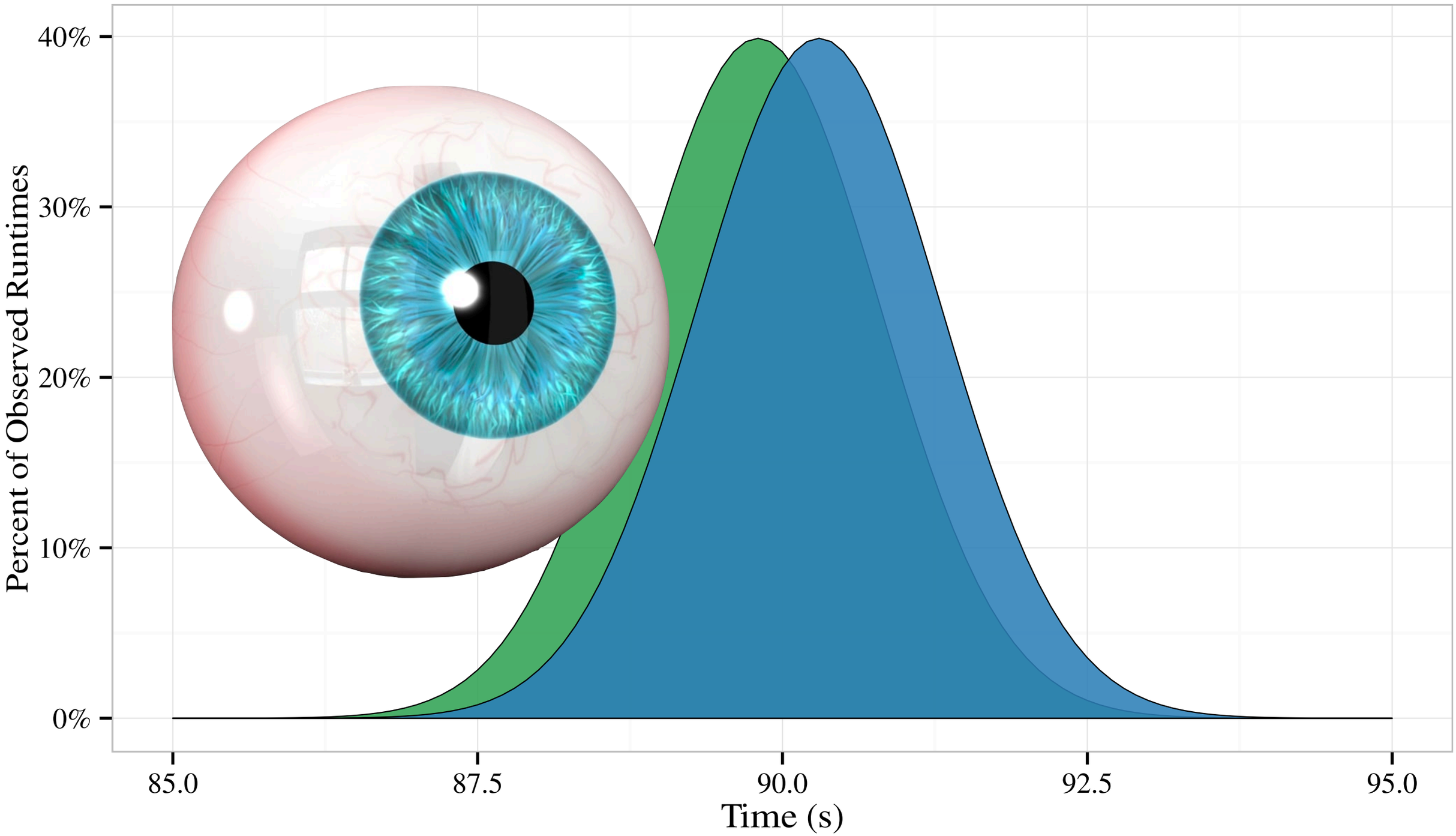
Is A′ faster than A?



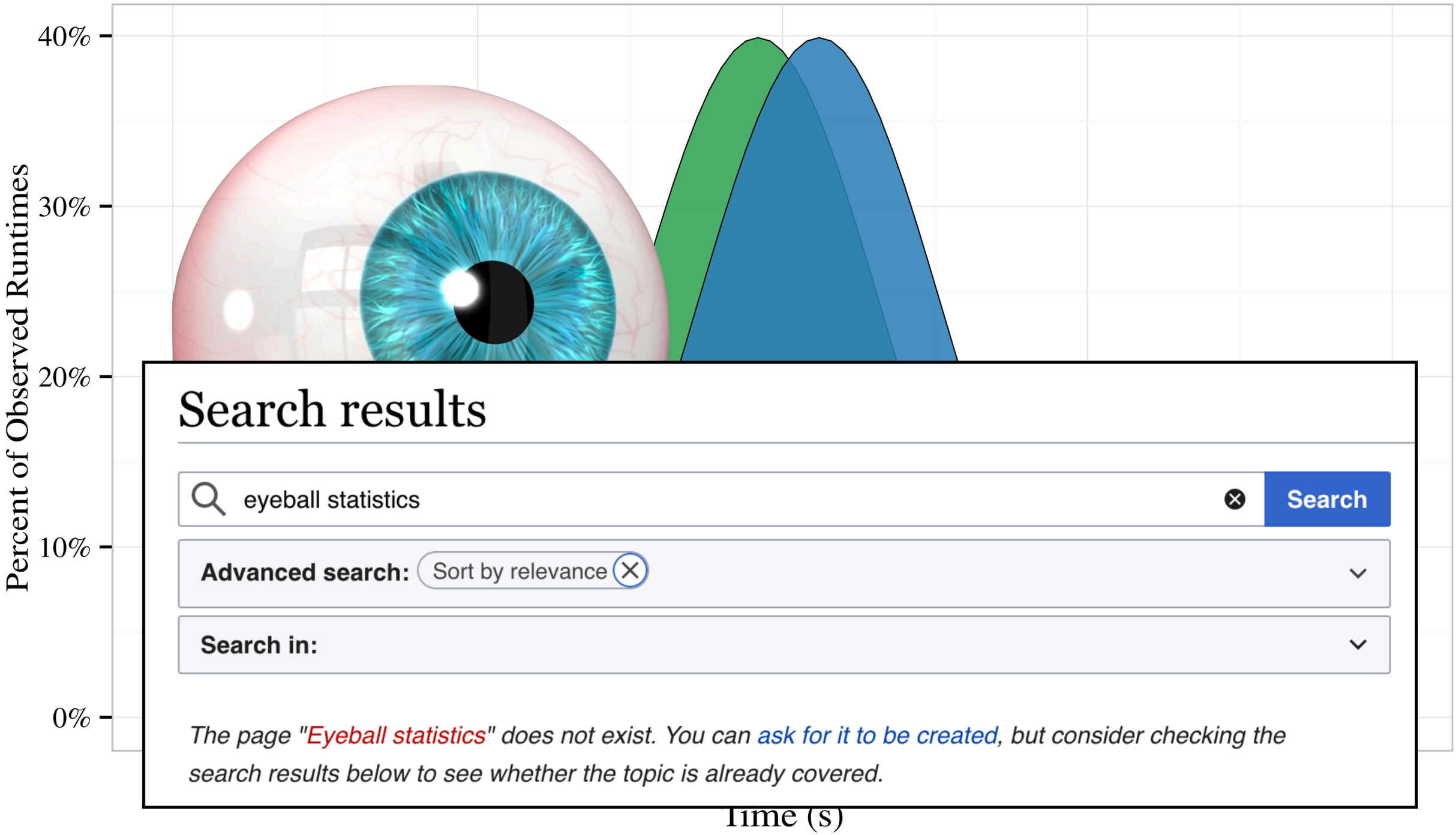
Is A′ faster than A?



Is A′ faster than A?



Is A′ faster than A?

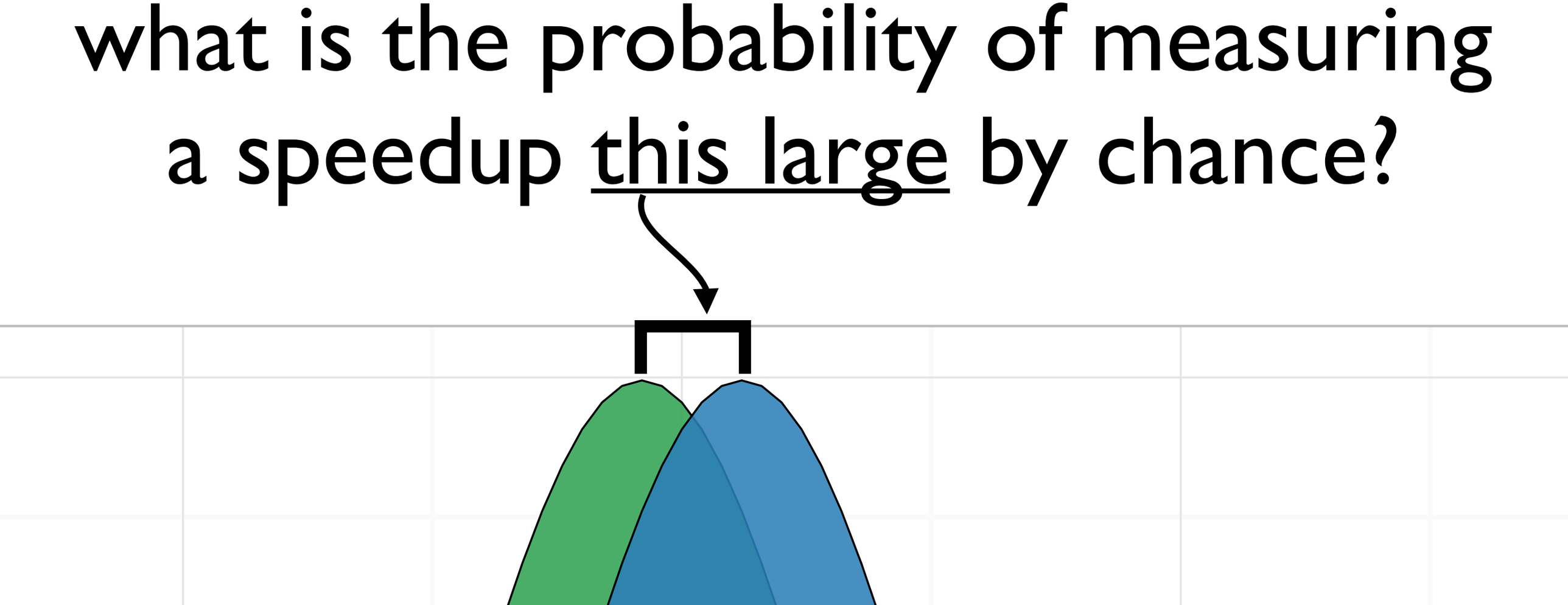


The Statistical Approach

null hypothesis significance testing

If A′ = A

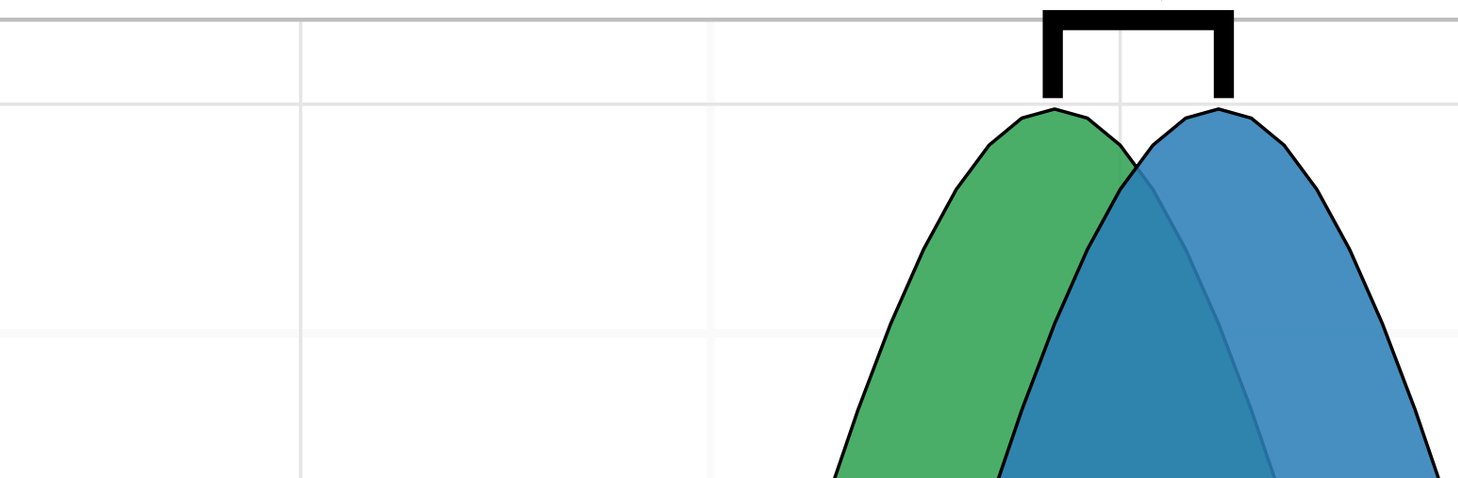
what is the probability of measuring a speedup this large by chance?



If A′ = A

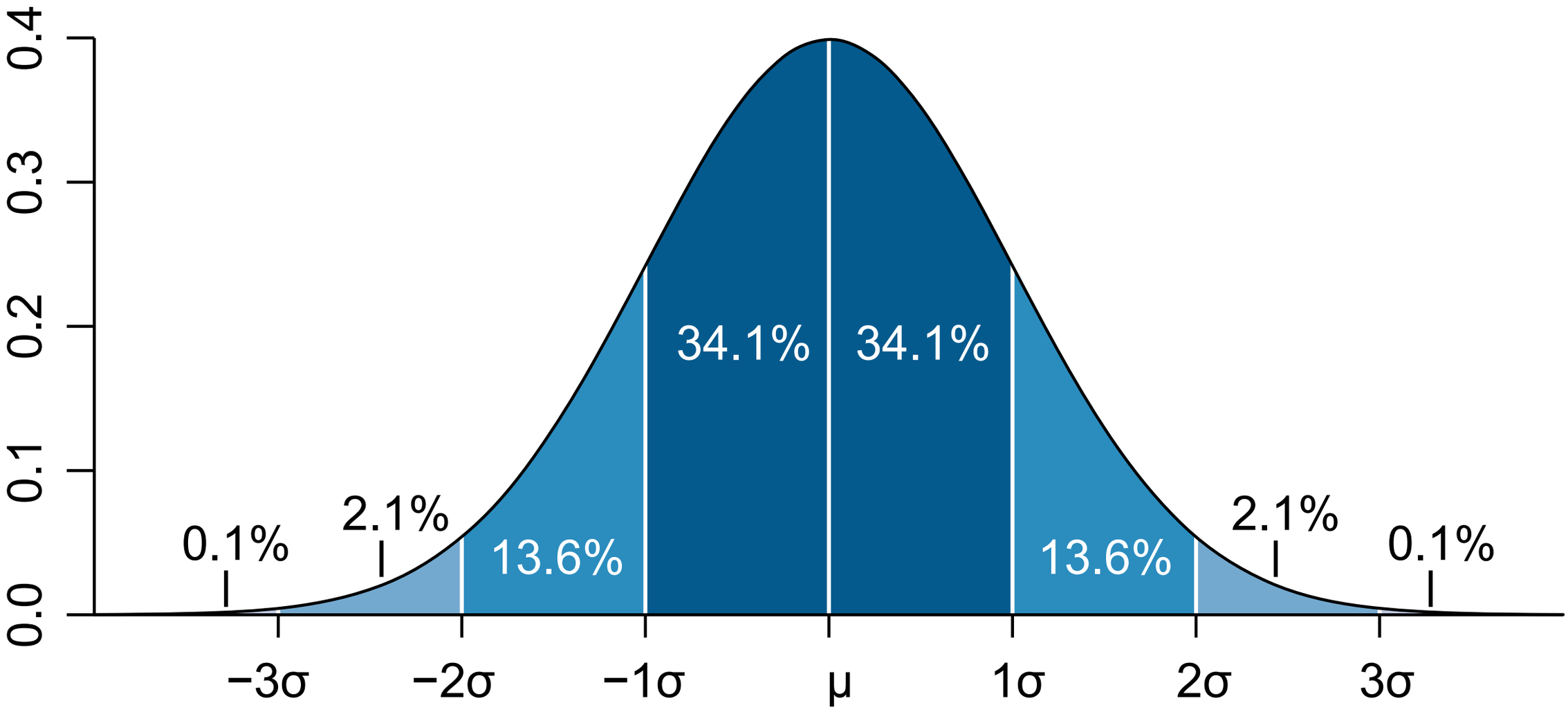
easy to compute for the normal distribution

what is the probability of measuring a speedup this large by chance?



If A' = A

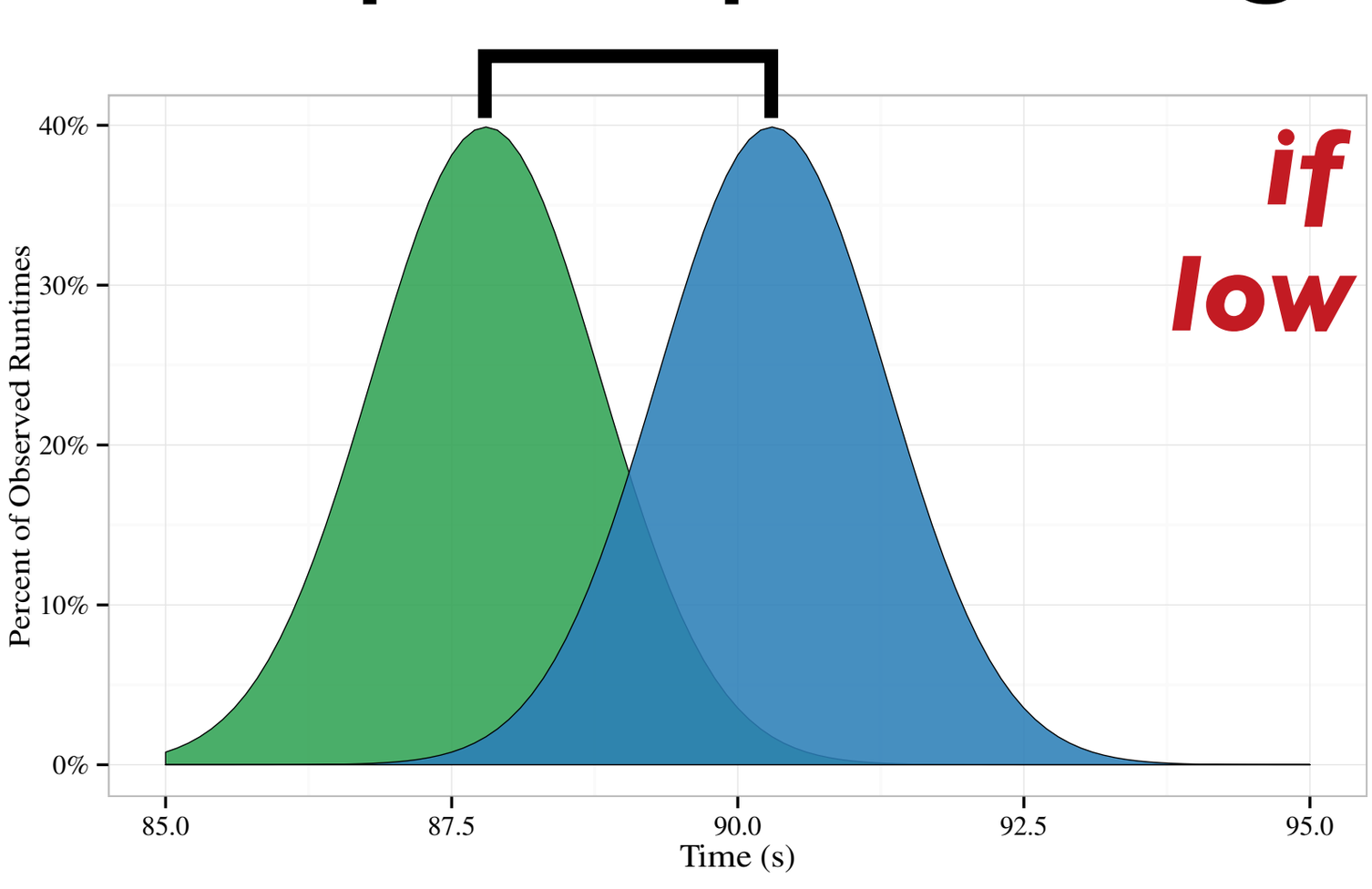
easy to compute for the normal distribution



STABILIZER

repeatedly randomizes layout

what is the probability of measuring a speedup this large by chance?



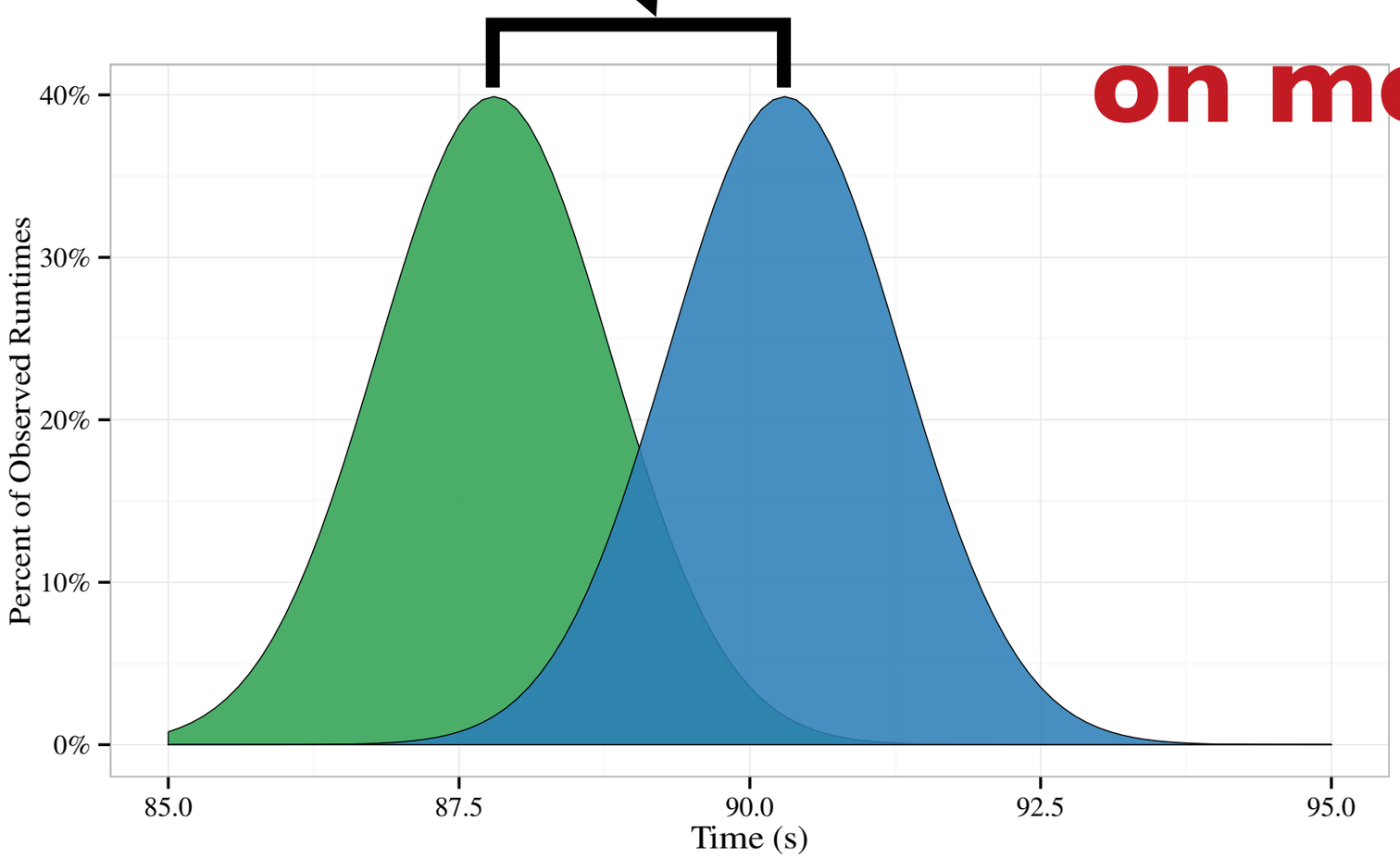
if there is a low probability

STABILIZER

repeatedly randomizes layout

this speedup is real

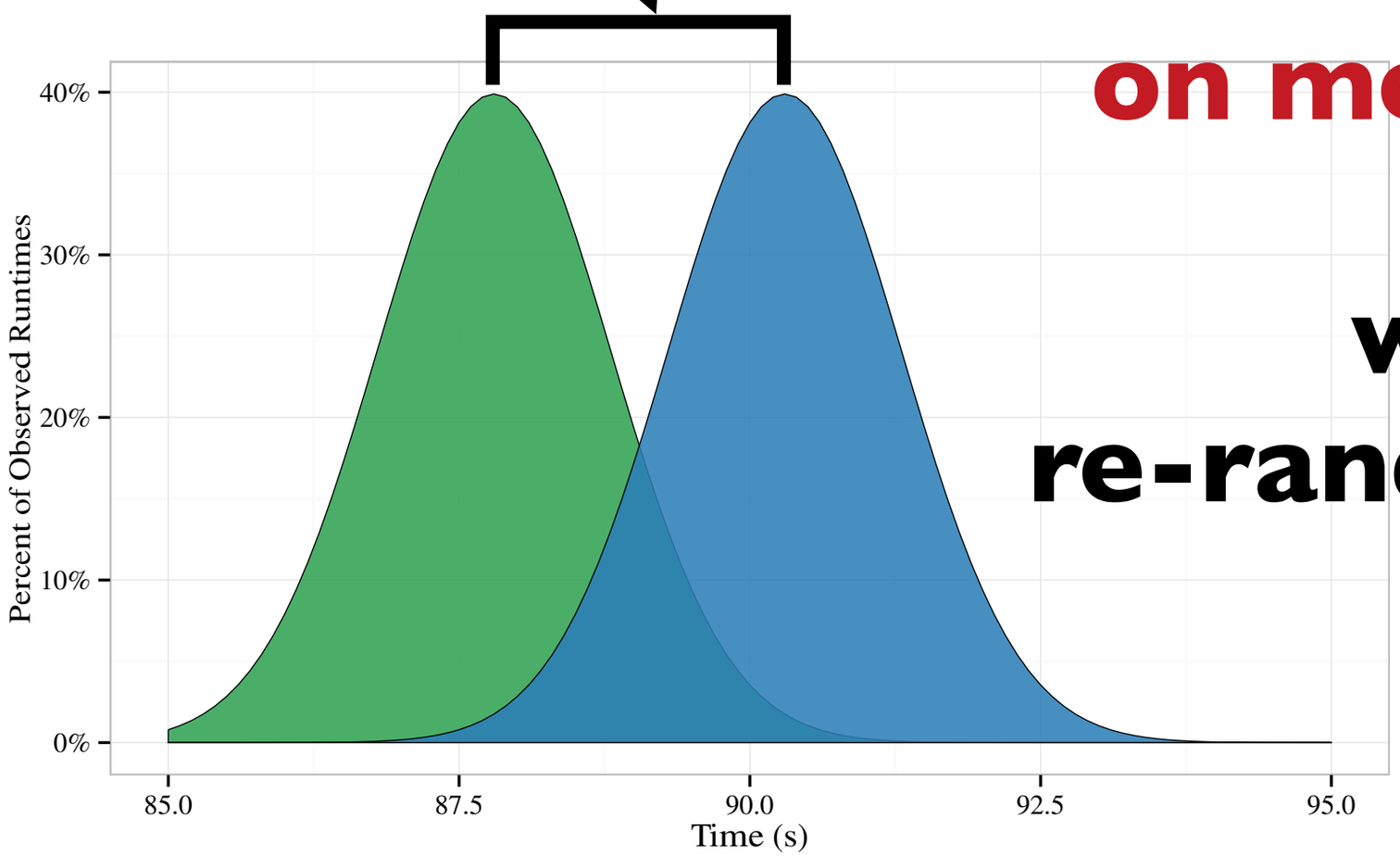
not due to the effect on memory layout



STABILIZER

repeatedly randomizes layout

this speedup is real

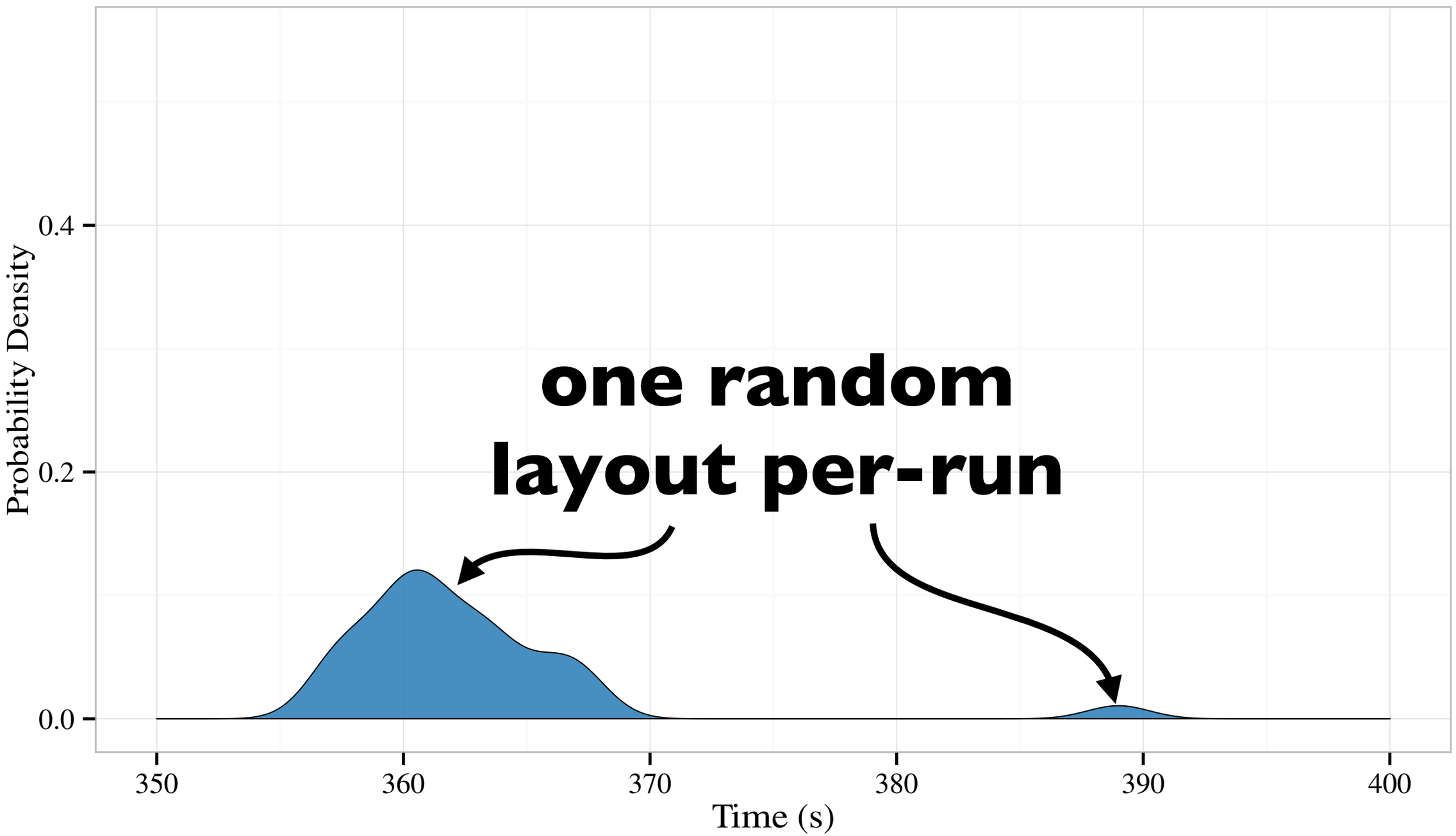


not due to the effect on memory layout

what does re-randomization do?

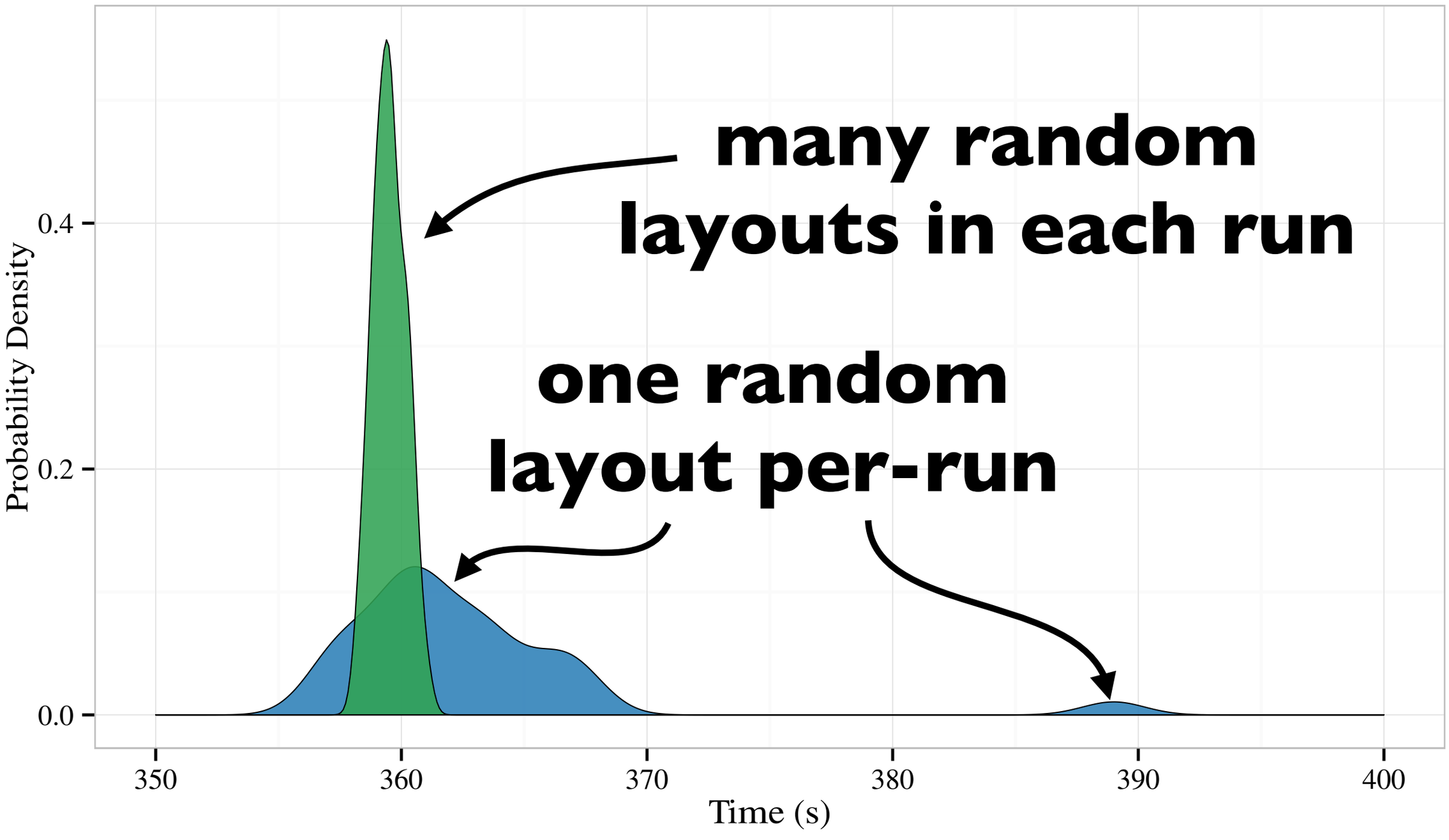
STABILIZER

repeatedly randomizes layout



STABILIZER

repeatedly randomizes layout



STABILIZER

repeatedly randomizes layout

STABILIZER generates a new random layout every ½ second

Total execution time is the sum of all periods

STABILIZER

repeatedly randomizes layout

STABILIZER generates a new random layout every ½ second

Total execution time is the sum of all periods

The sum

STABILIZER

repeatedly randomizes layout

STABILIZER generates a new random layout every ½ second

Total execution time is the sum of all periods

The sum of a sufficient number

STABILIZER

repeatedly randomizes layout

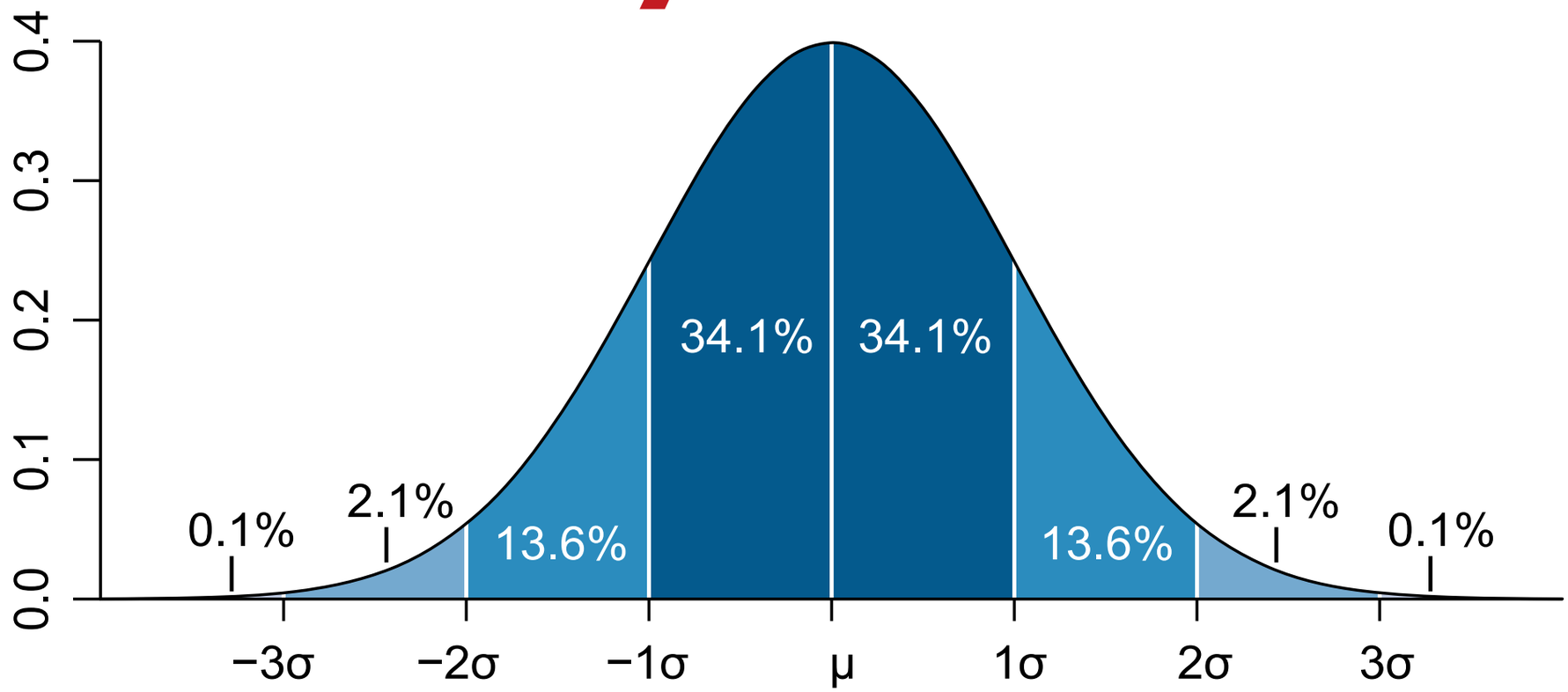
STABILIZER generates a new random layout every ½ second

Total execution time is the sum of all periods

The sum of a sufficient number of independent, identically distributed random variables

Central Limit Theorem

execution times are normally distributed



The sum of a sufficient number of independent, identically distributed random variables is approximately normally distributed.

STABILIZER

Memory layout affects performance

makes performance evaluation difficult

Use randomization to eliminate the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

STABILIZER

Memory layout affects performance

makes performance evaluation difficult

Use randomization to eliminate the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

Case Studies

evaluation of LLVM's optimizations with STABILIZER

on each benchmark

across the whole benchmark suite

first, build benchmarks with STABILIZER

Build programs with STABILIZER

```
> szc main.c
```

Build programs with STABILIZER

```
> szc main.c
```

Build programs with STABILIZER

```
> szc -Rcode main.c
```

Build programs with STABILIZER

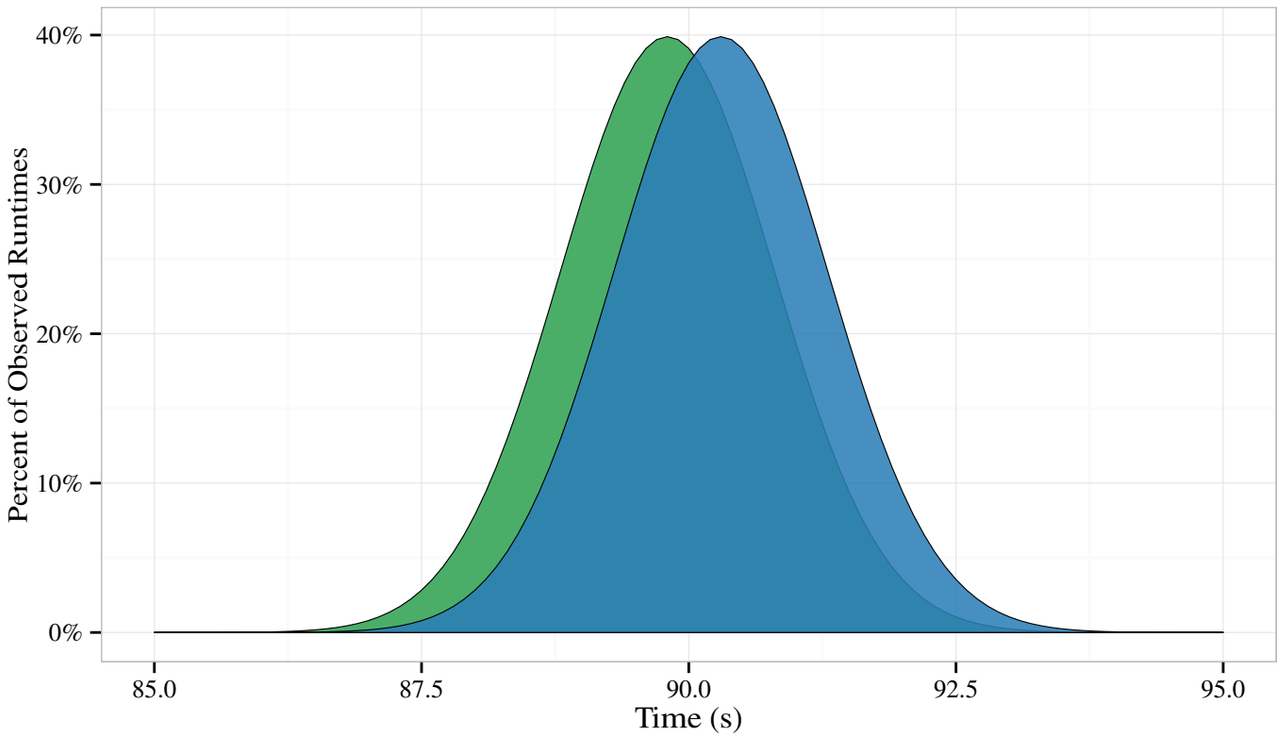
```
> szc -Rcode -Rheap -Rstack main.c
```

now run the benchmarks

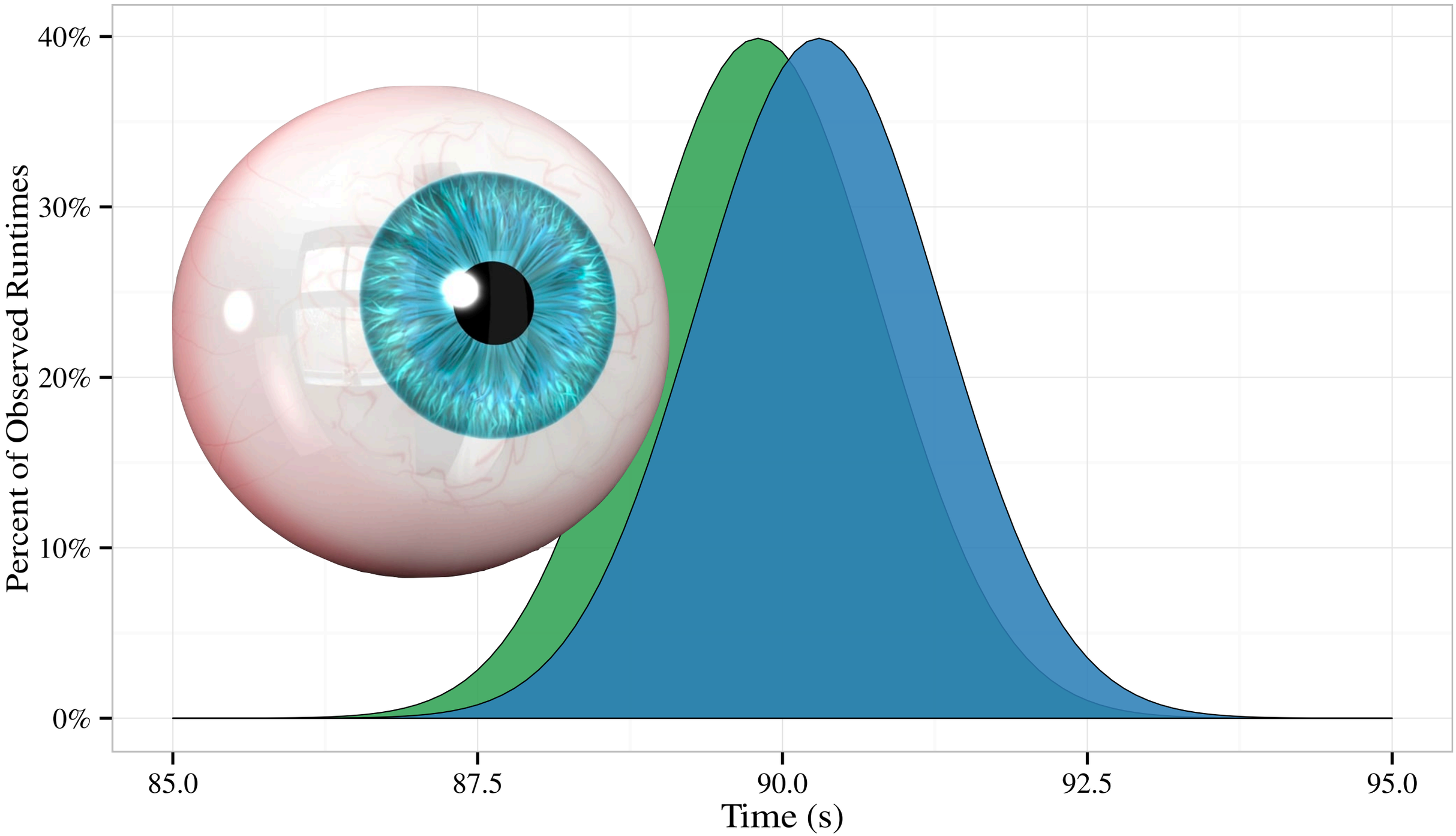
Run benchmarks as usual

A ×30 A′ ×30

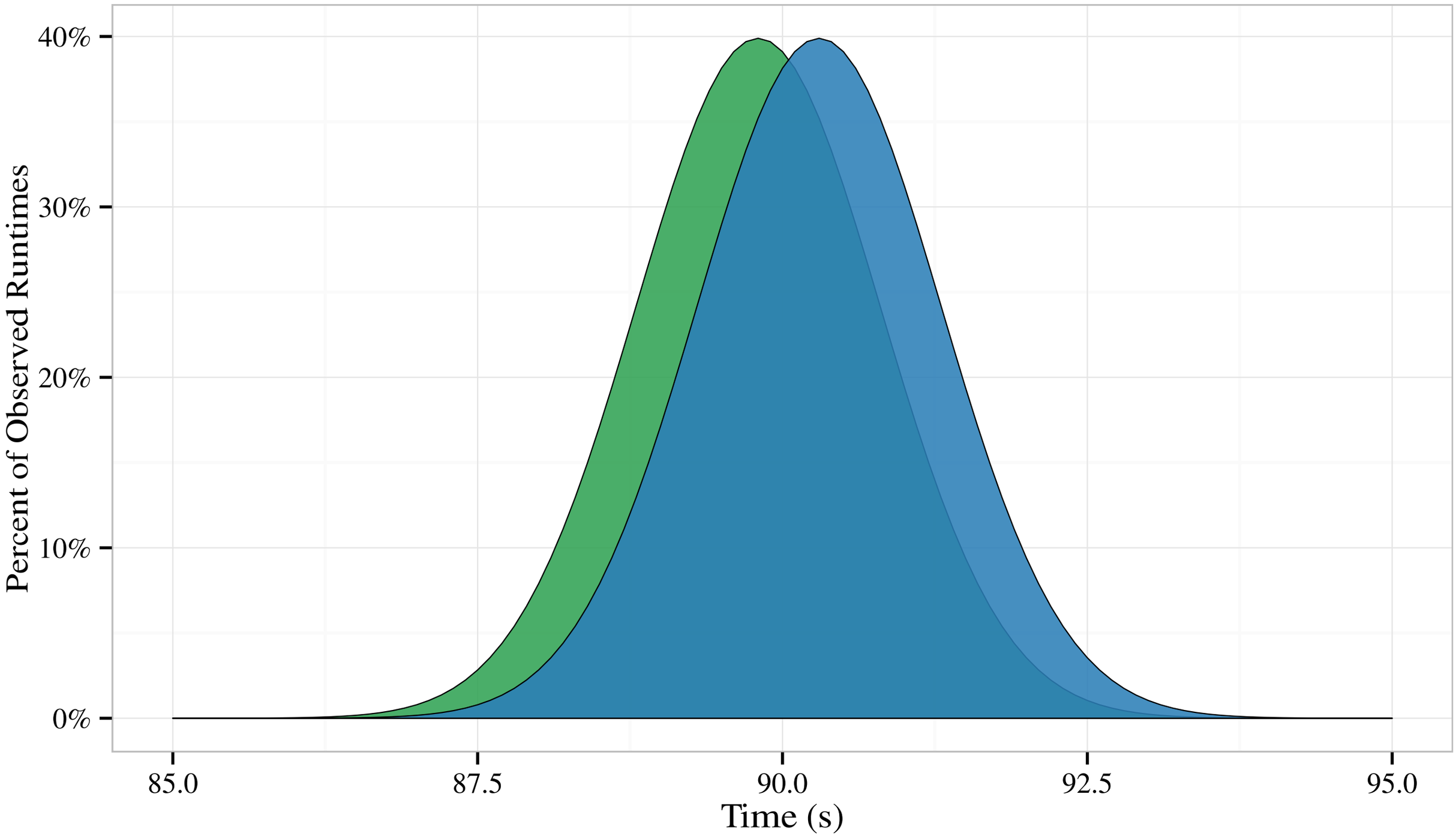
drop the results into R



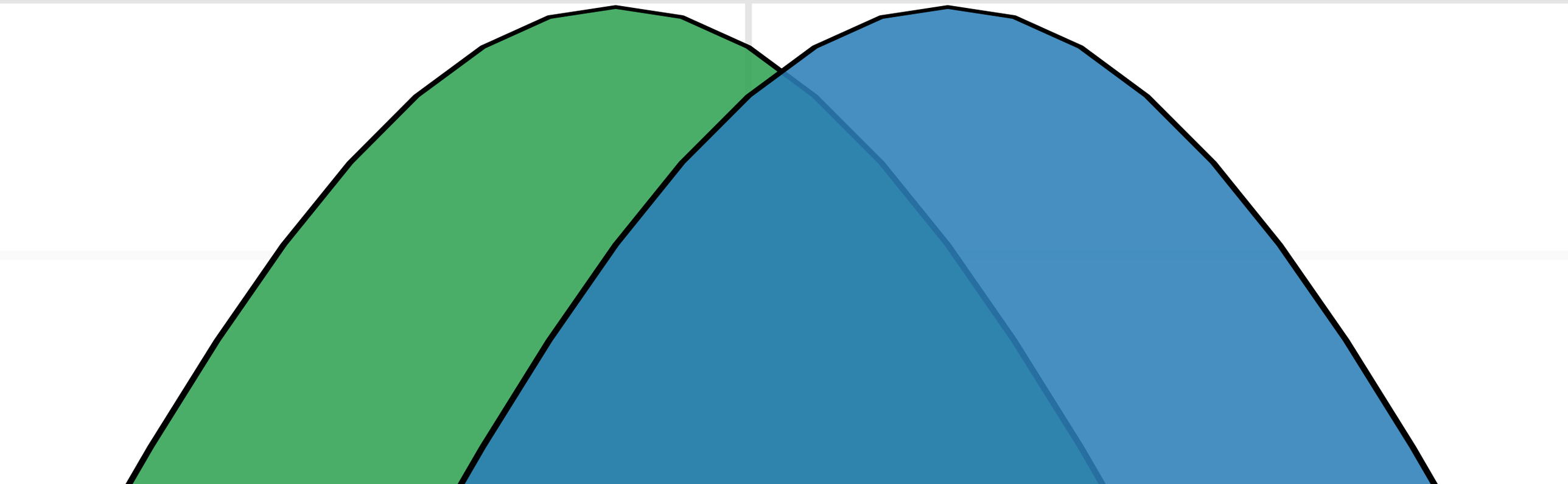
Is A′ faster than A?



If A′ = A

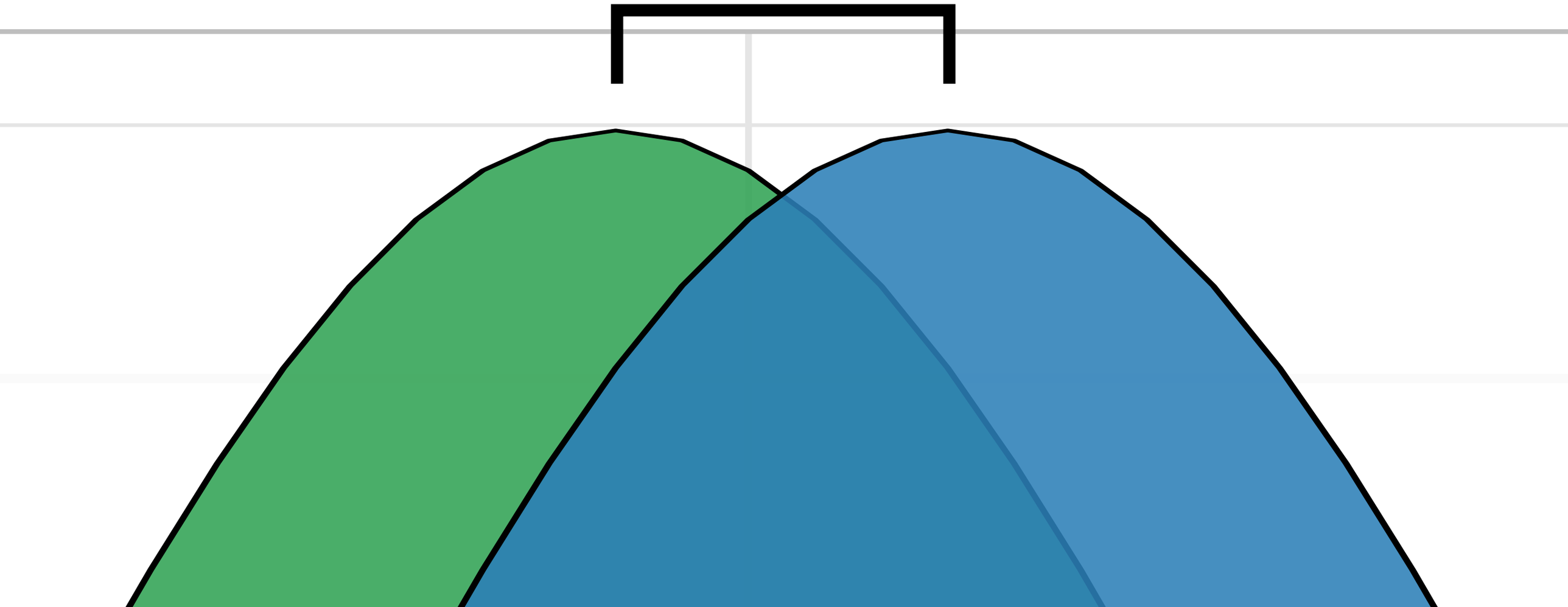

40%
30%
20%
10%
0%
Percent of Observed Runtimes
85.0
87.5
90.0
92.5
95.0
Time (s)

If A' = A



If A′ = A

what is the *probability* of measuring a difference at least this large?



The Student's t-test

`t.test(times.A', times.A)`

If A′ = A

what is the *probability* of measuring a difference at least this large?

The Student's t-test

If p-value

If A′ = A

what is the *probability* of measuring a difference at least this large?

The Student's t-test

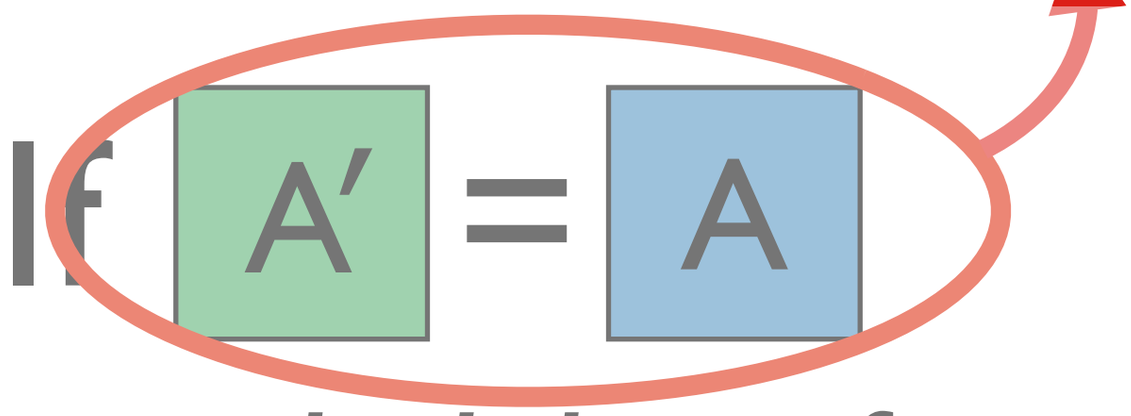
If p-value ≤ 5%

If A′ = A

what is the *probability* of measuring a difference at least this large?

The Student's t-test

If p-value ≤ 5%
we reject the null hypothesis



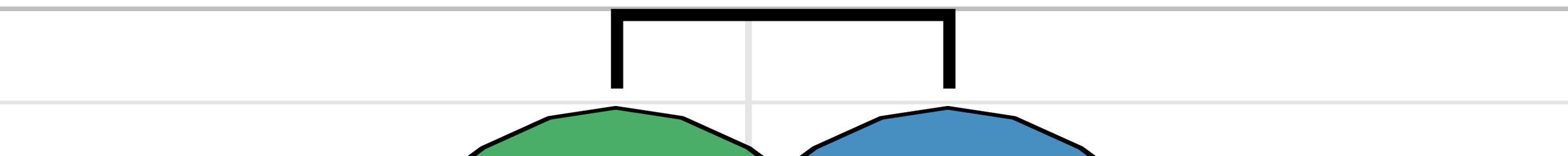
what is the *probability* of measuring a difference at least this large?

The Student's t-test

If p-value ≤ 5%
we reject the null hypothesis

$A' \neq A$

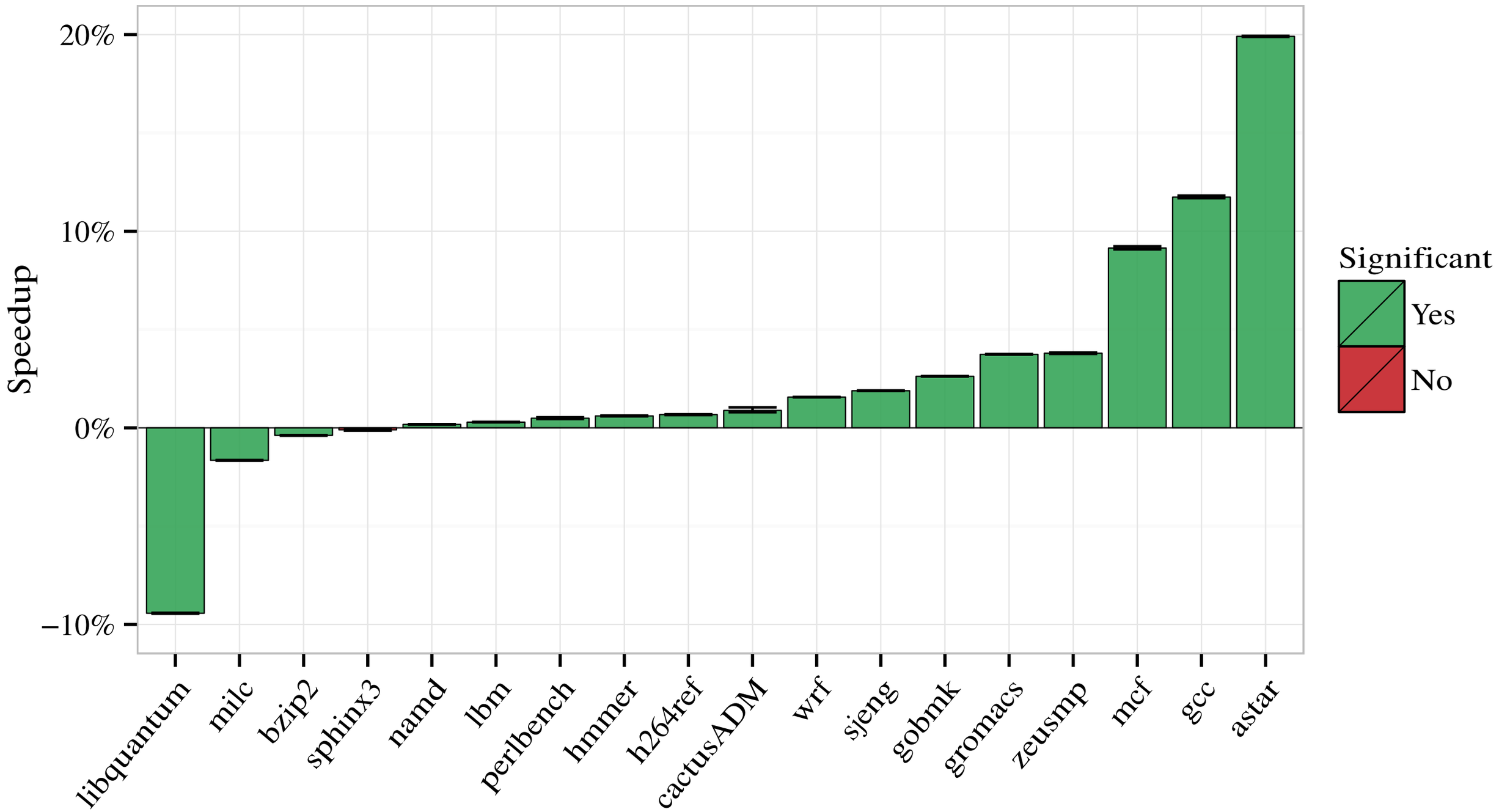
Random chance not responsible for the measured difference



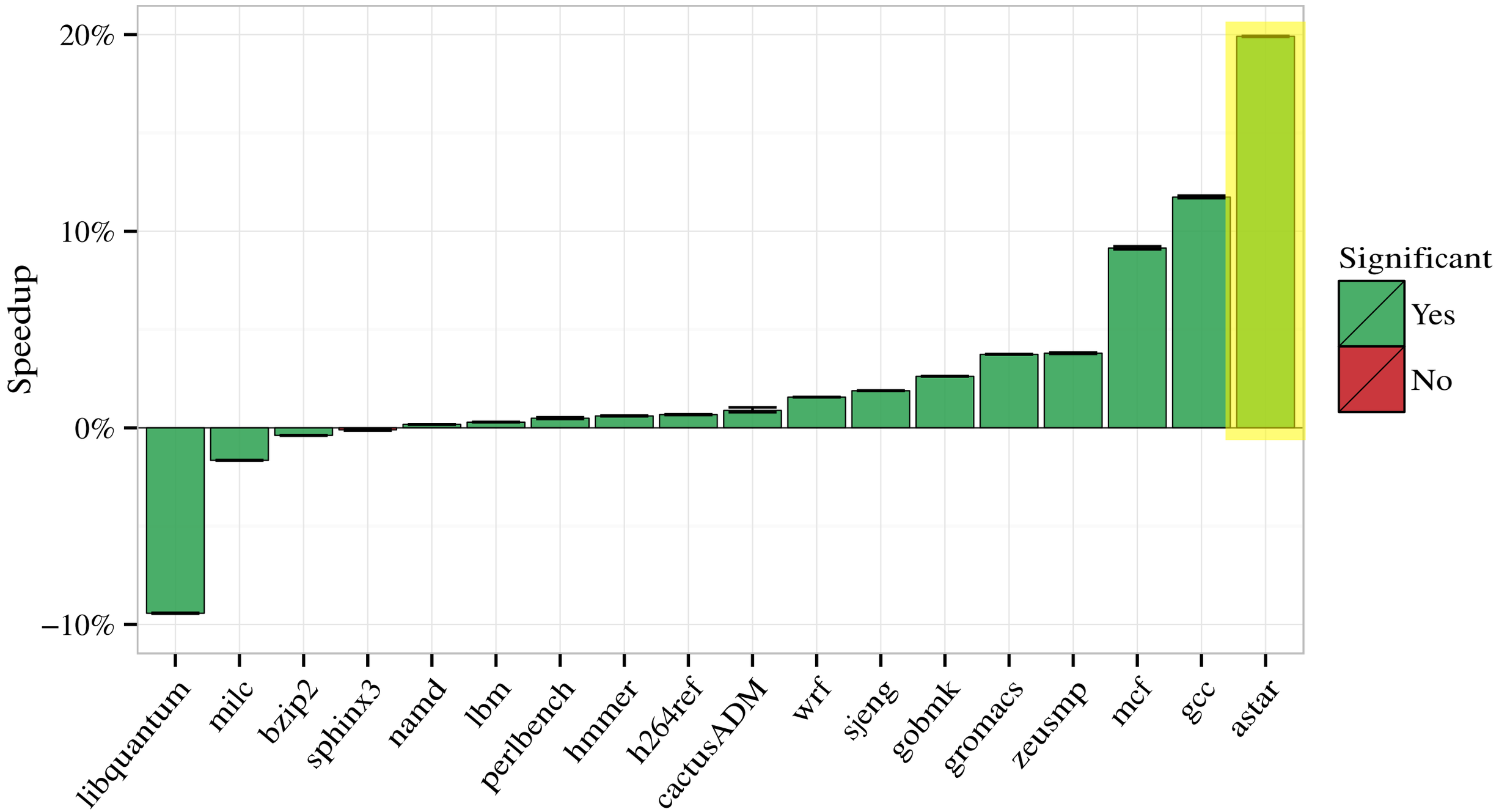
The difference is real

$$A' \neq A$$

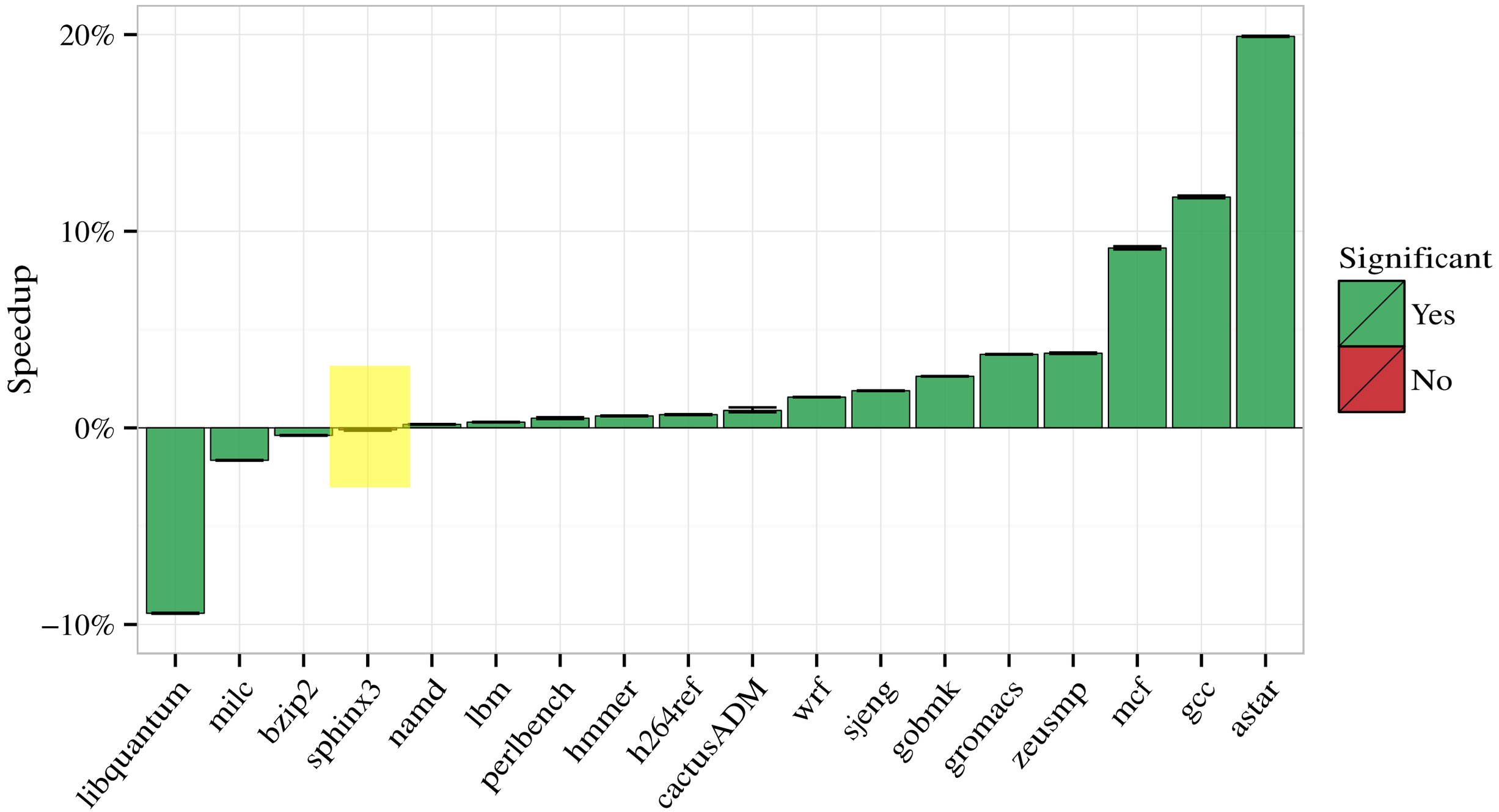
Speedup of -O2 over -O1



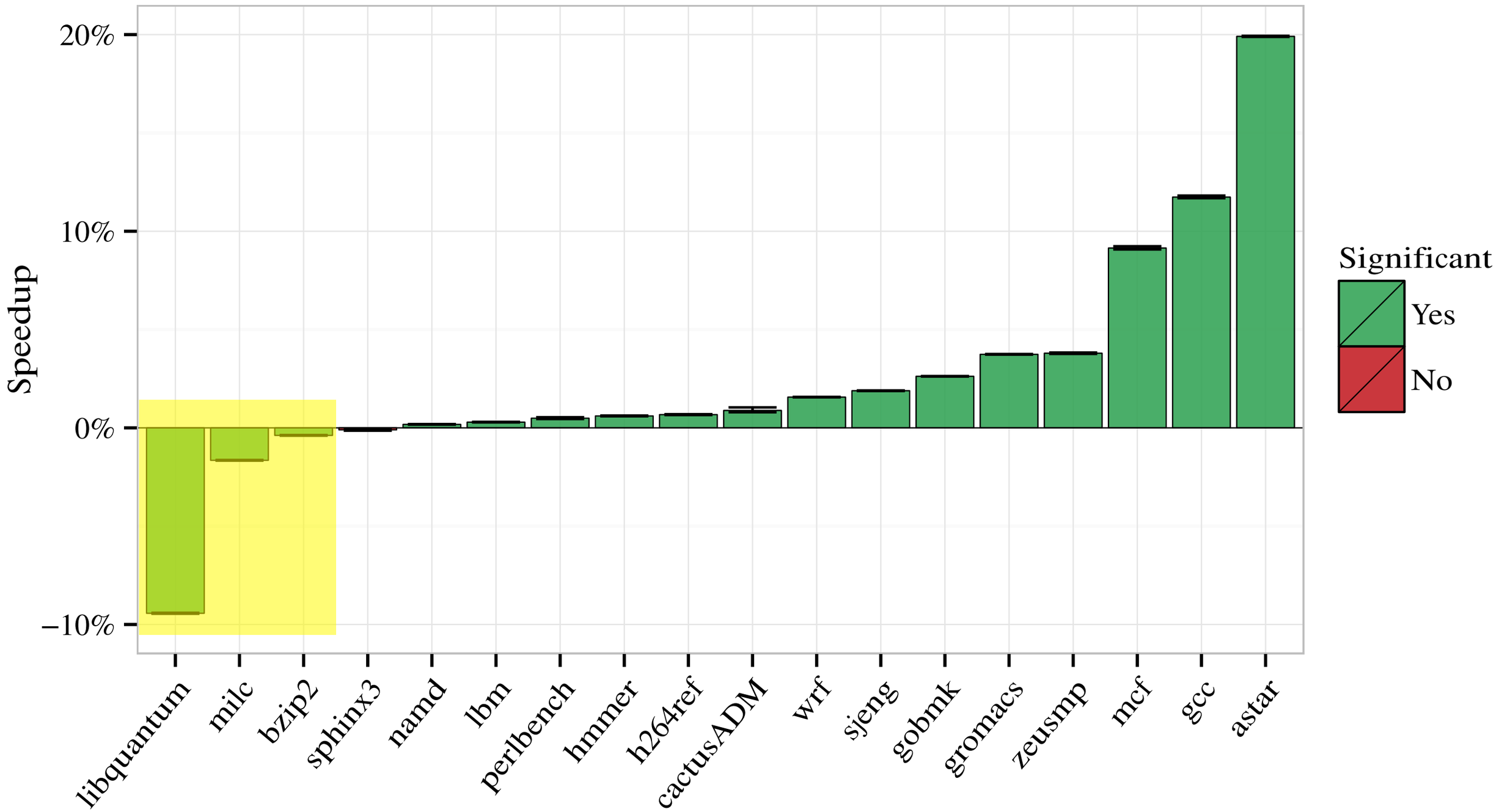
Speedup of -O2 over -O1



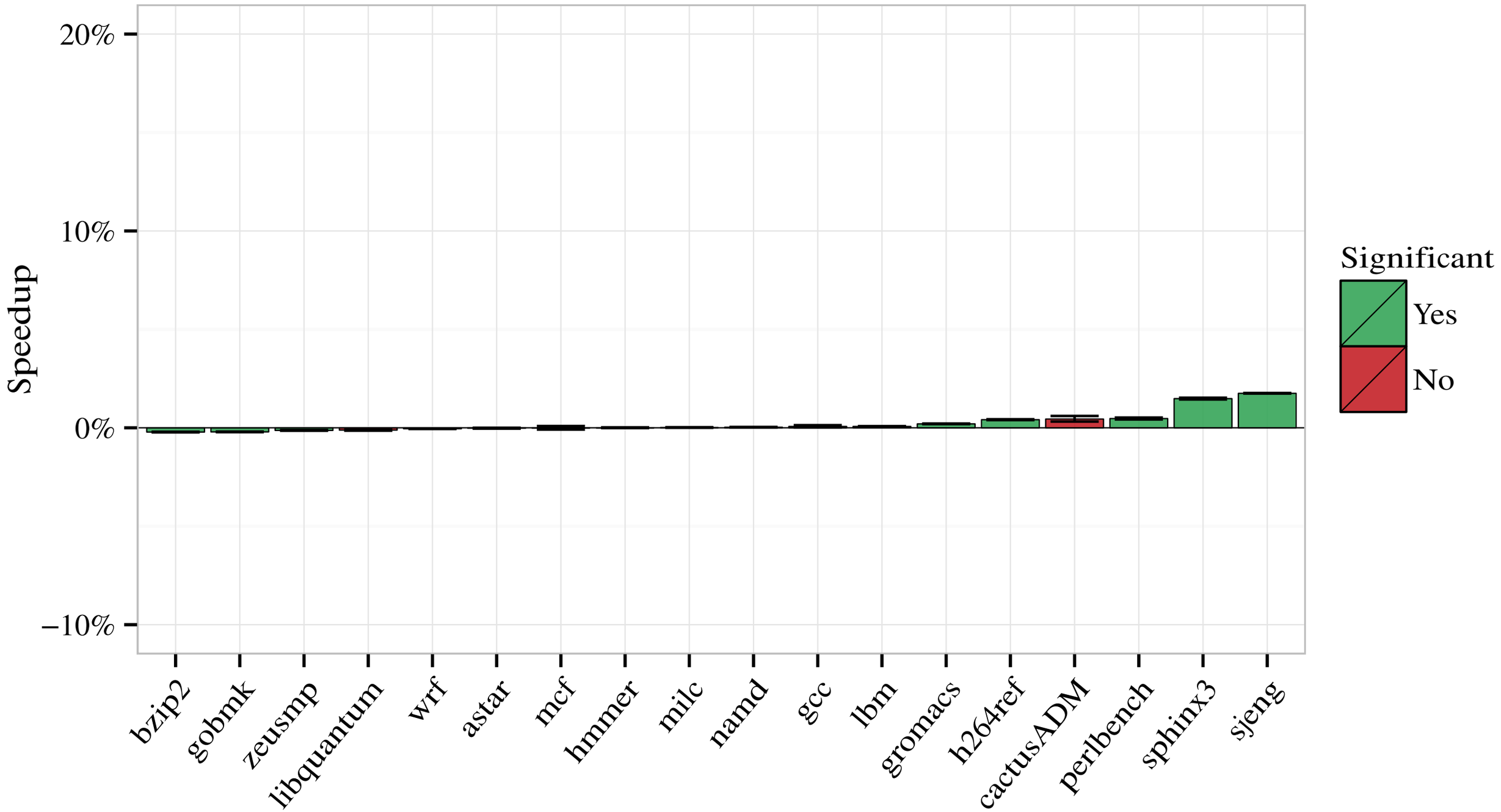
Speedup of -O2 over -O1



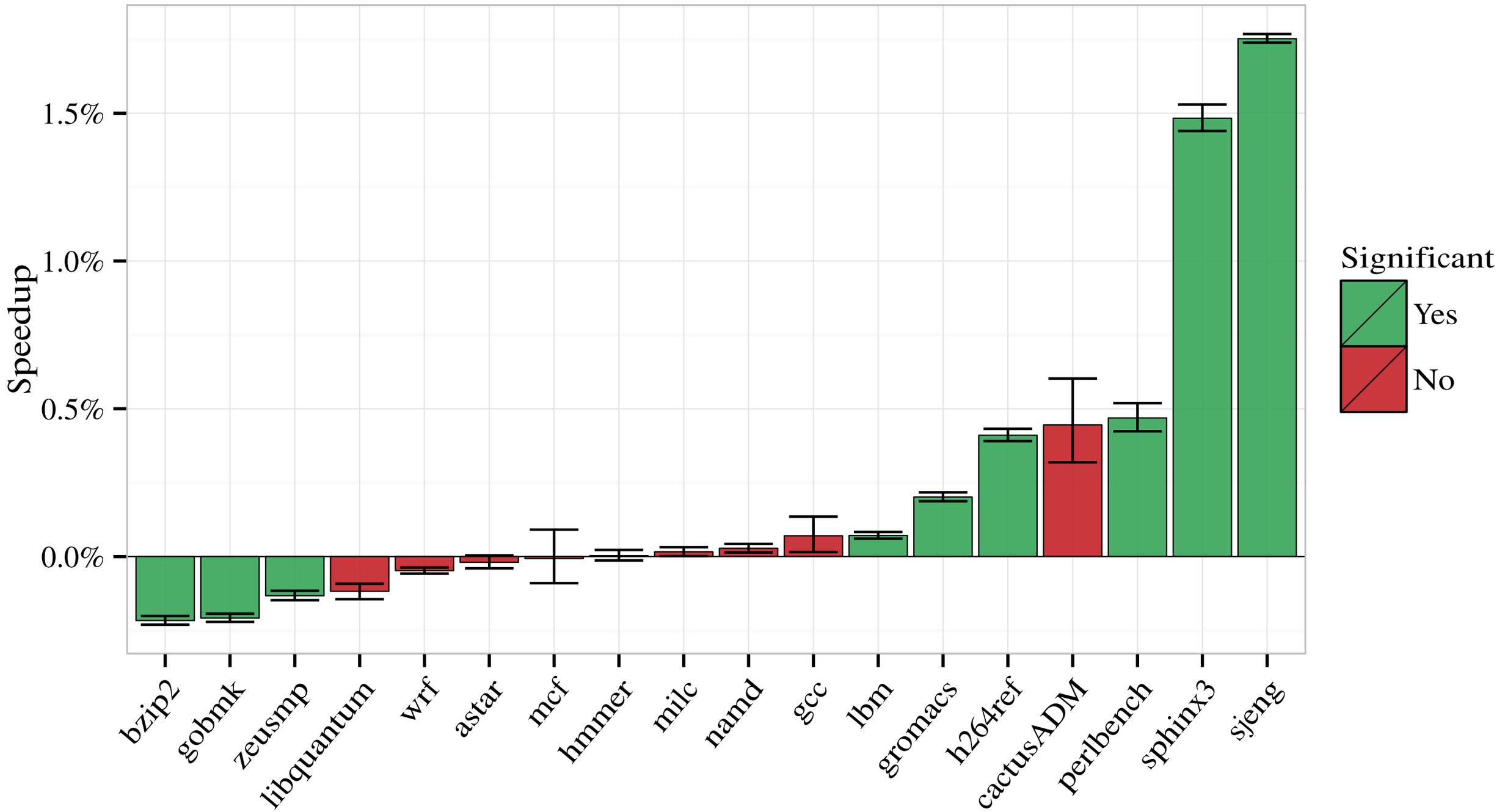
Speedup of -O2 over -O1



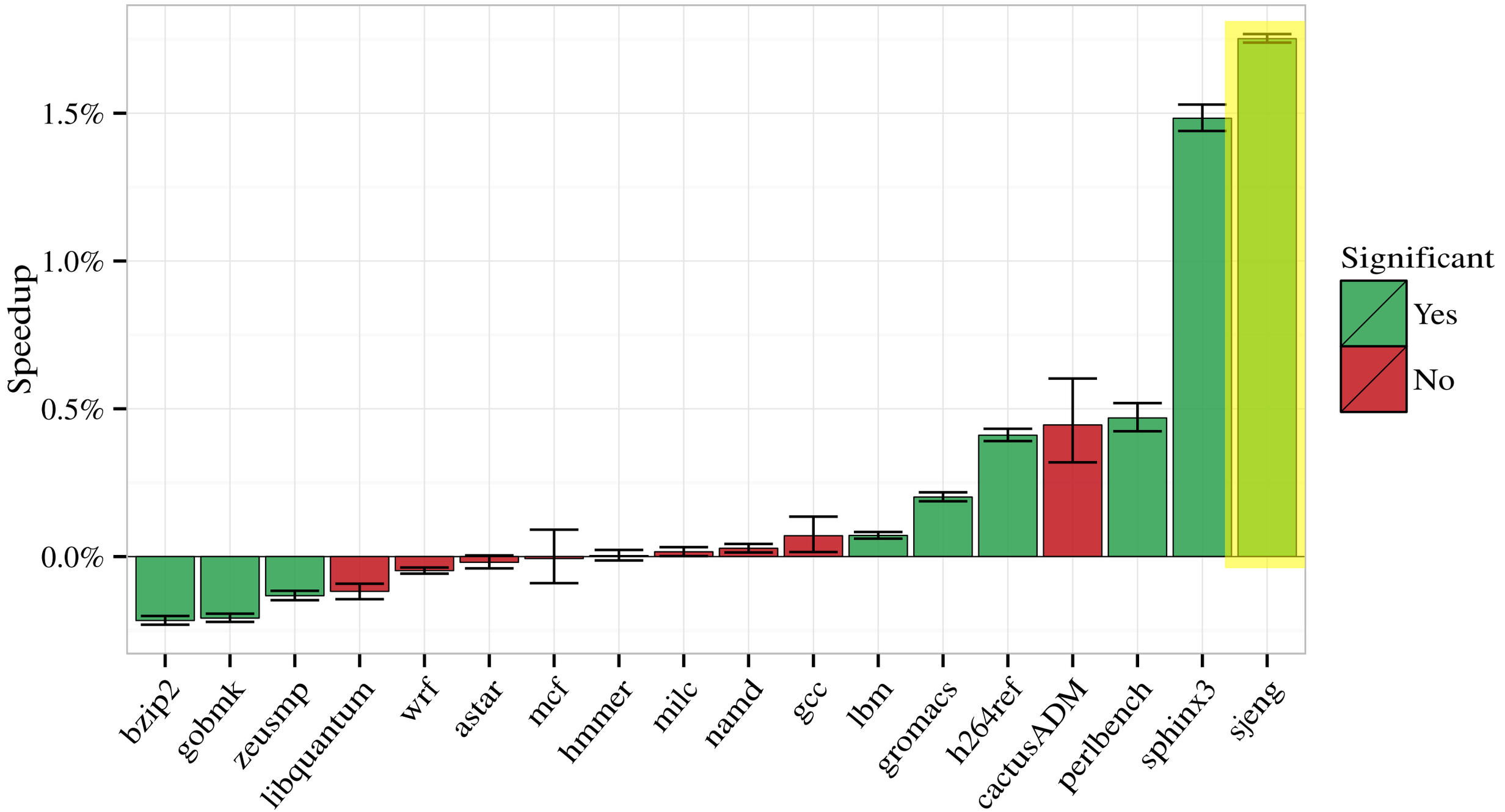
Speedup of -O3 over -O2



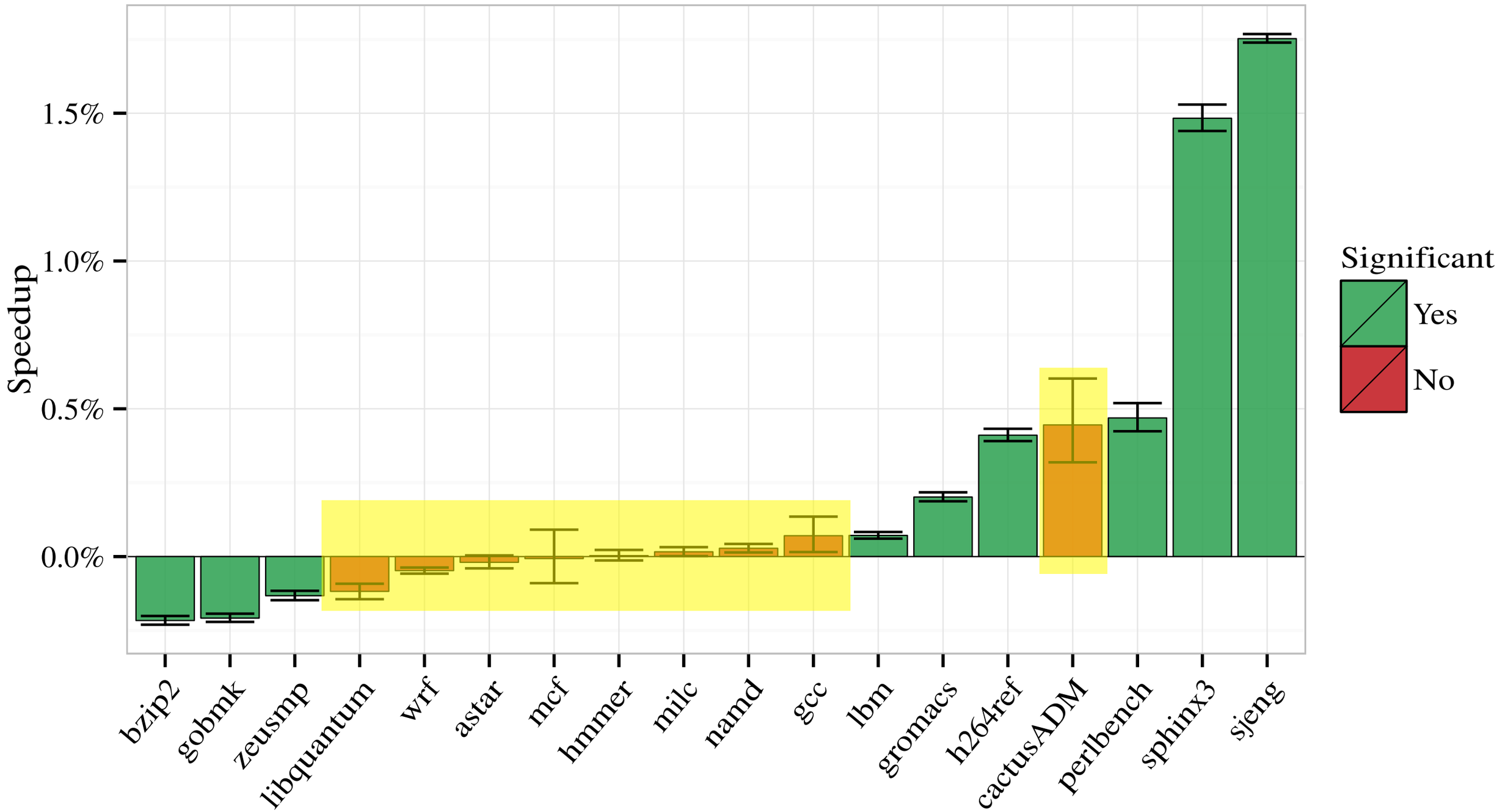
Speedup of -O3 over -O2



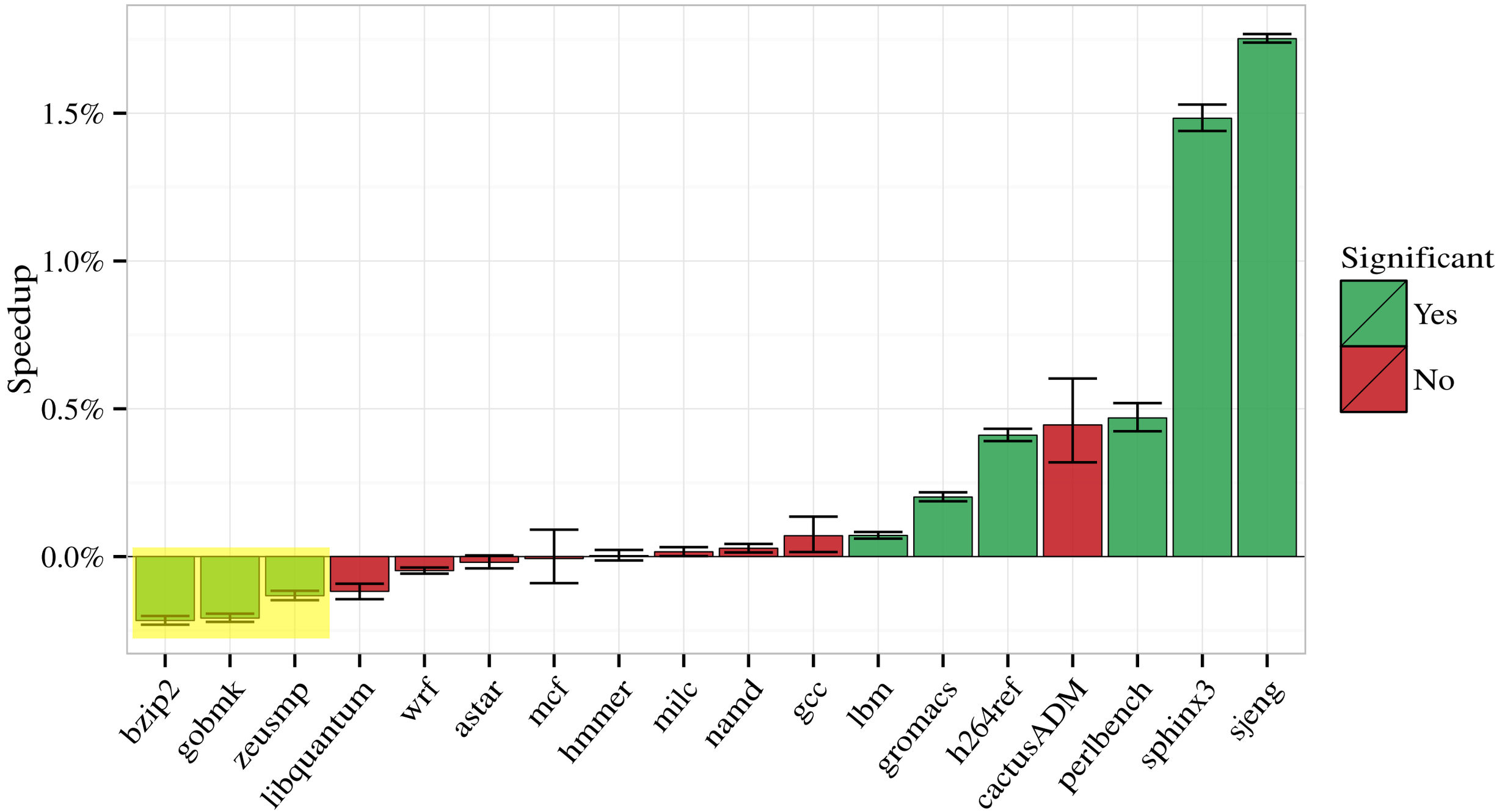
Speedup of -O3 over -O2



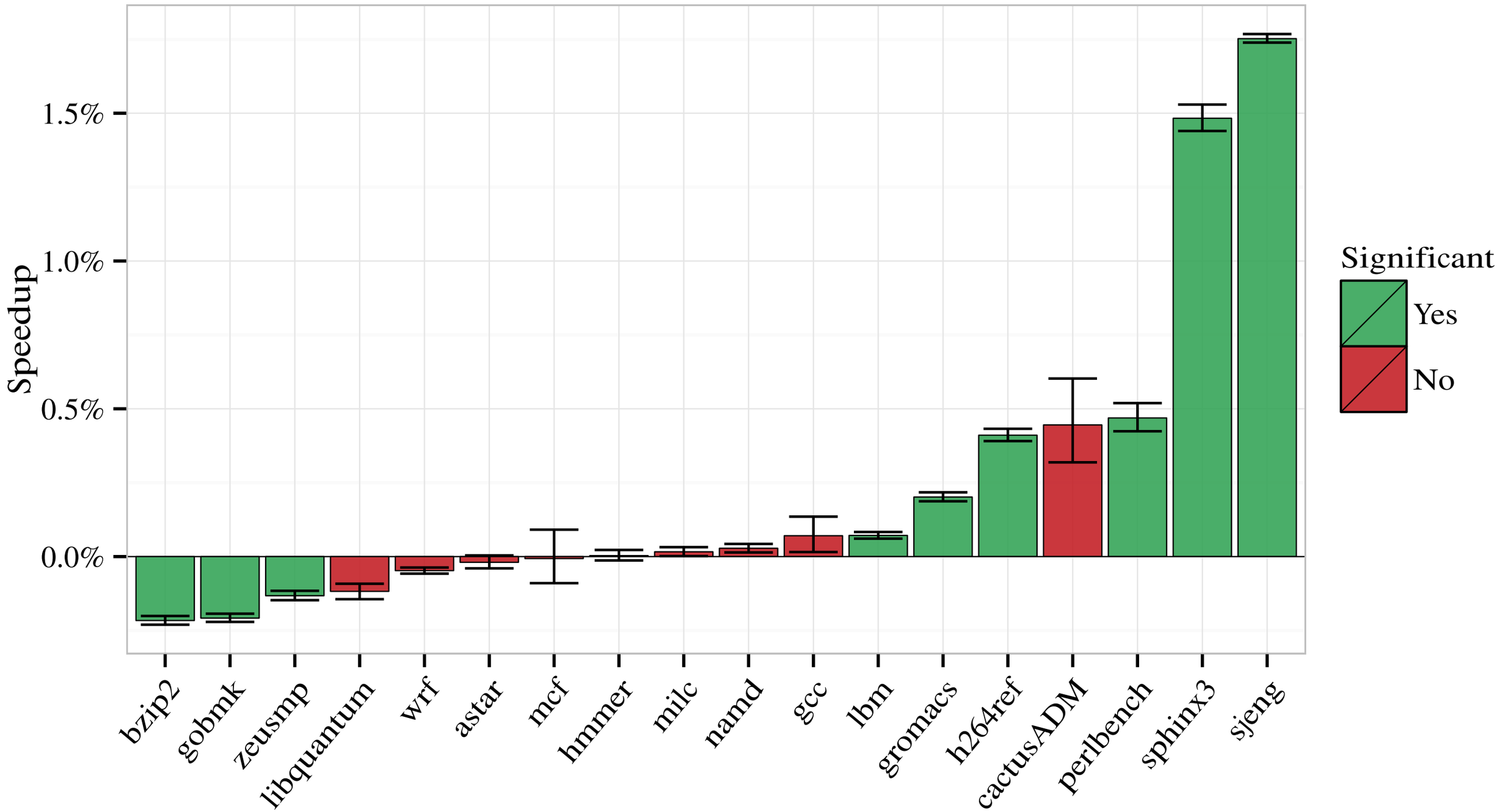
Speedup of -O3 over -O2



Speedup of -O3 over -O2



What do the results mean?



Comparing optimizations

-O2 ×30

-O3 ×30

Comparing optimizations

-O2 ×30

lbm ×30

-O3 ×30

lbm ×30

Comparing optimizations

-O2 ×30

lbm ×30

astar ×30

-O3 ×30

lbm ×30

astar ×30

Comparing optimizations

-O2 ×30

lbm ×30

astar ×30

-O3 ×30

lbm ×30

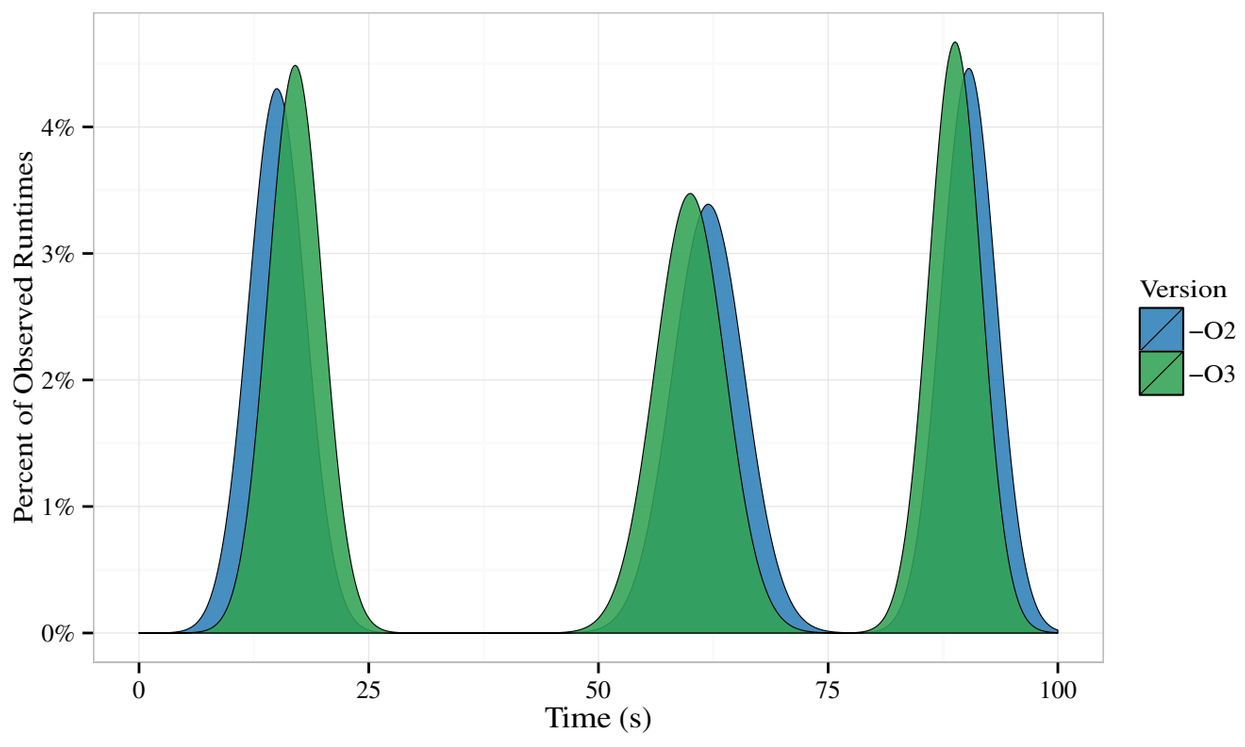
astar ×30

•••

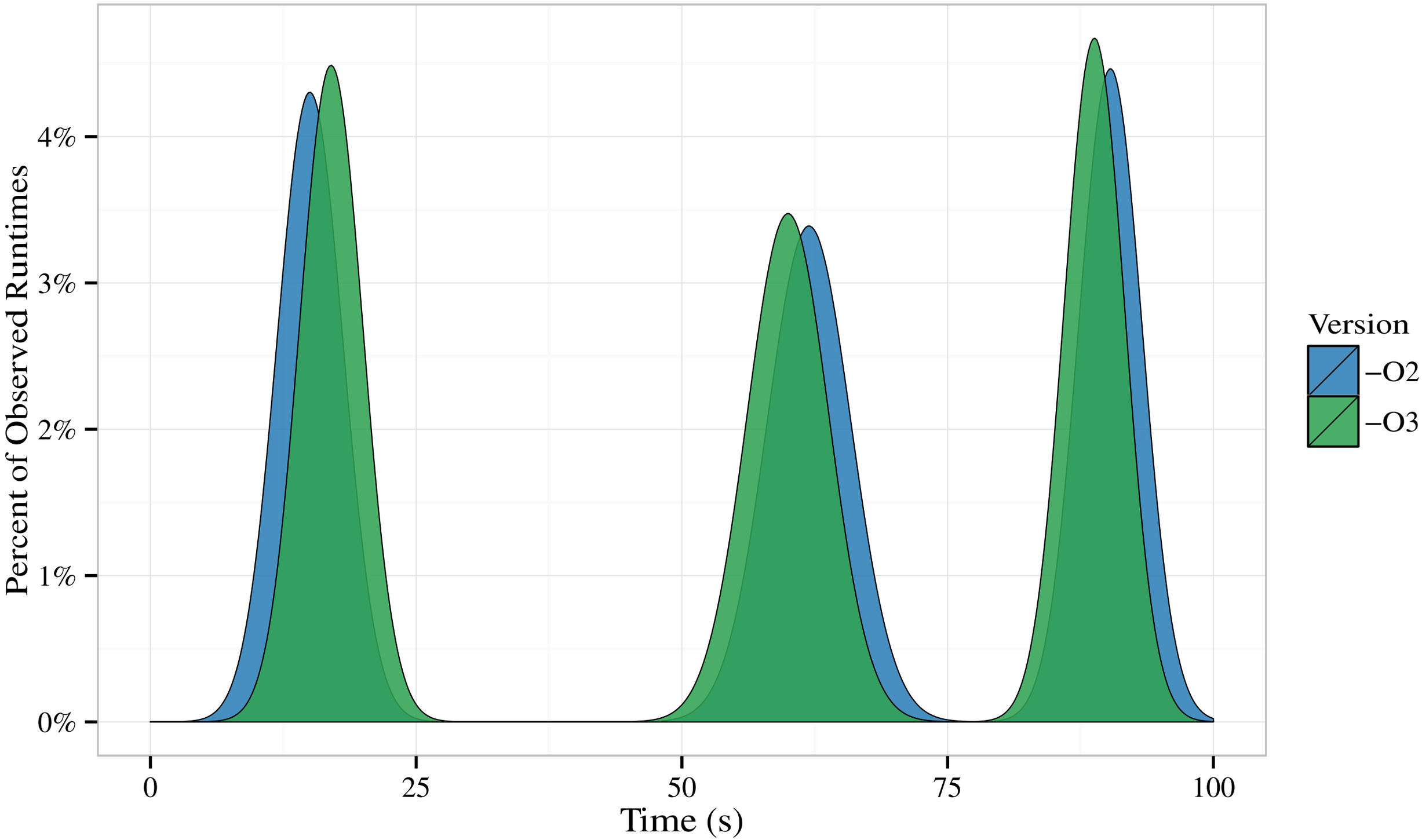
Comparing optimizations

-O2 ×30

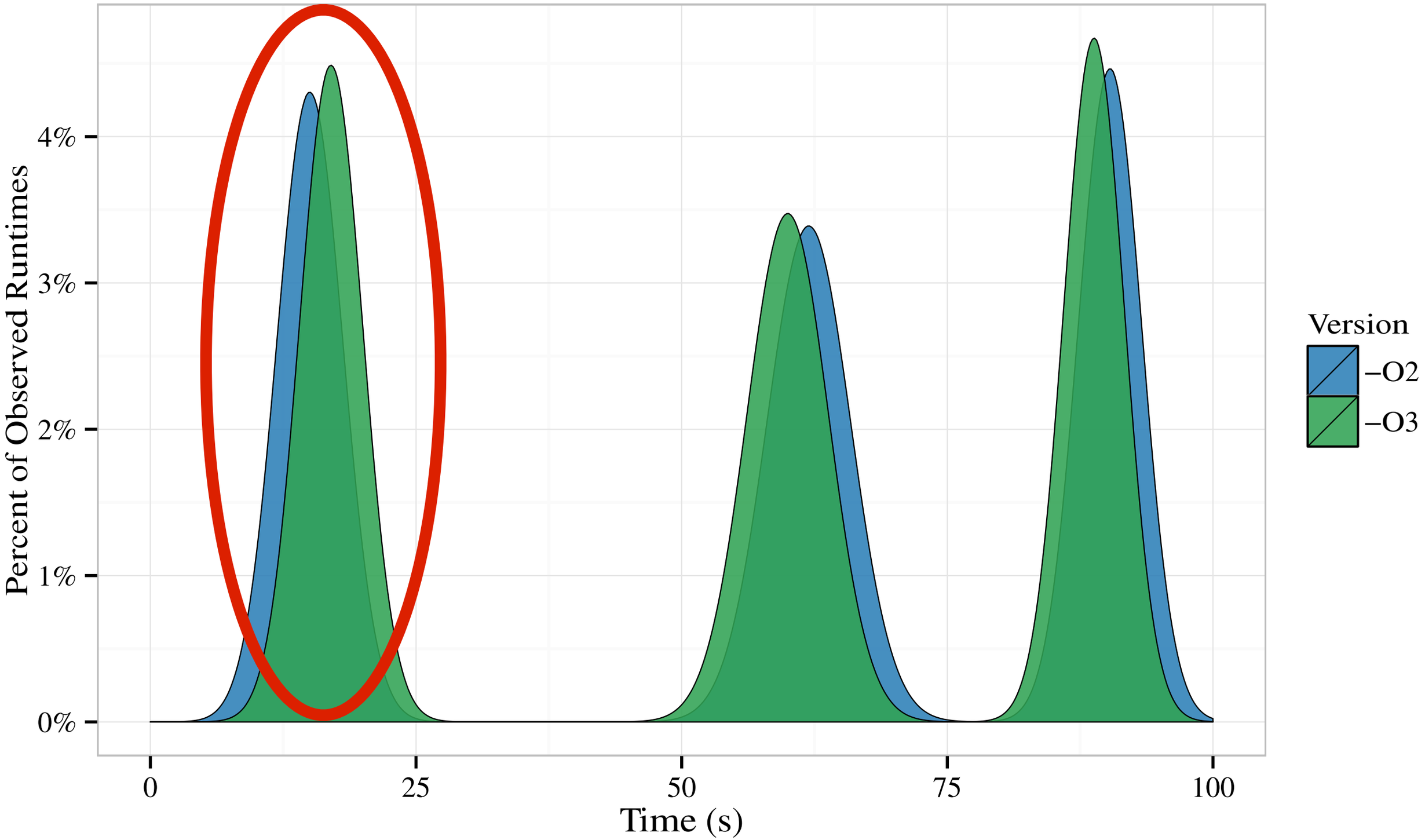
-O3 ×30



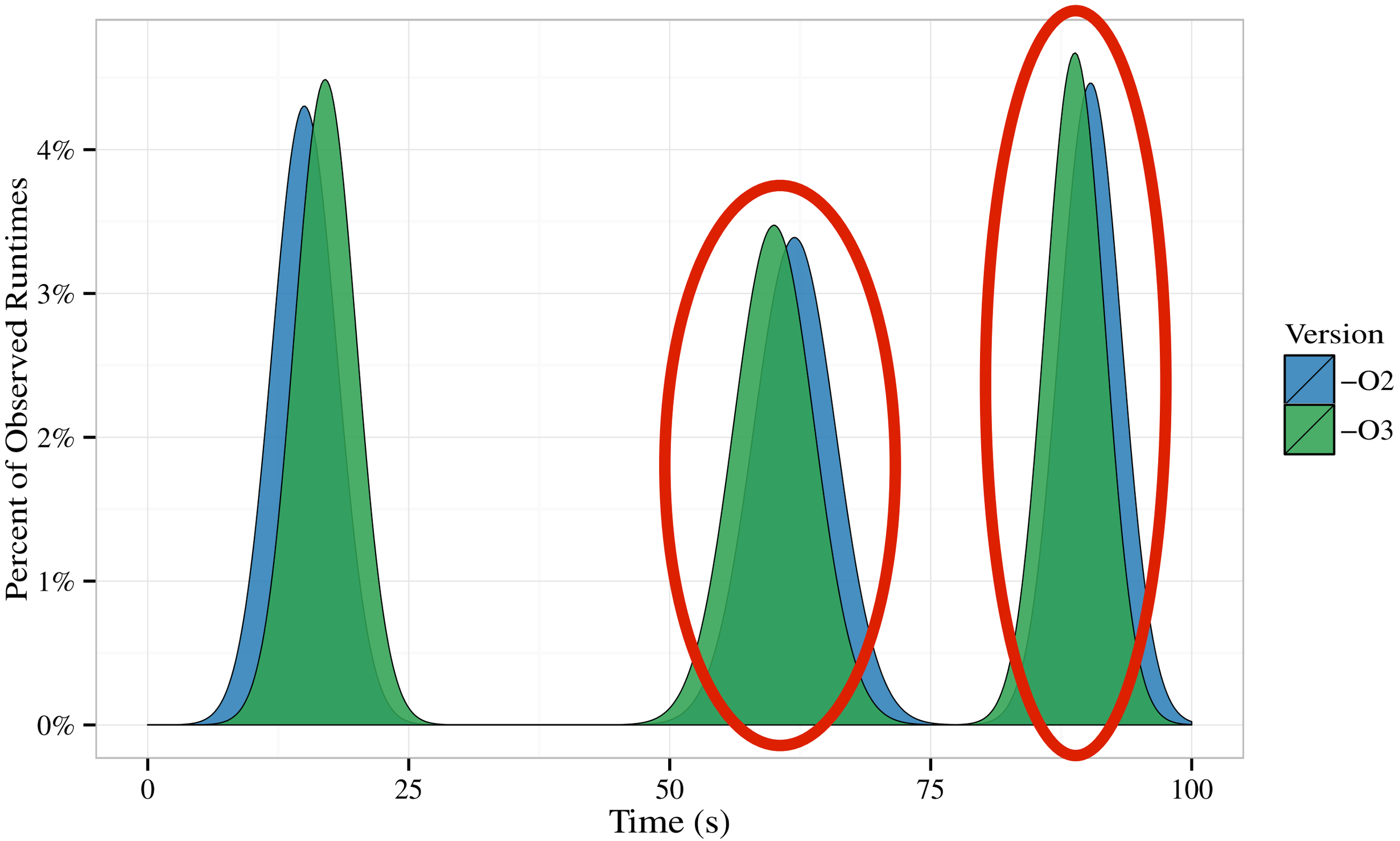
Is -O3 faster than -O2?



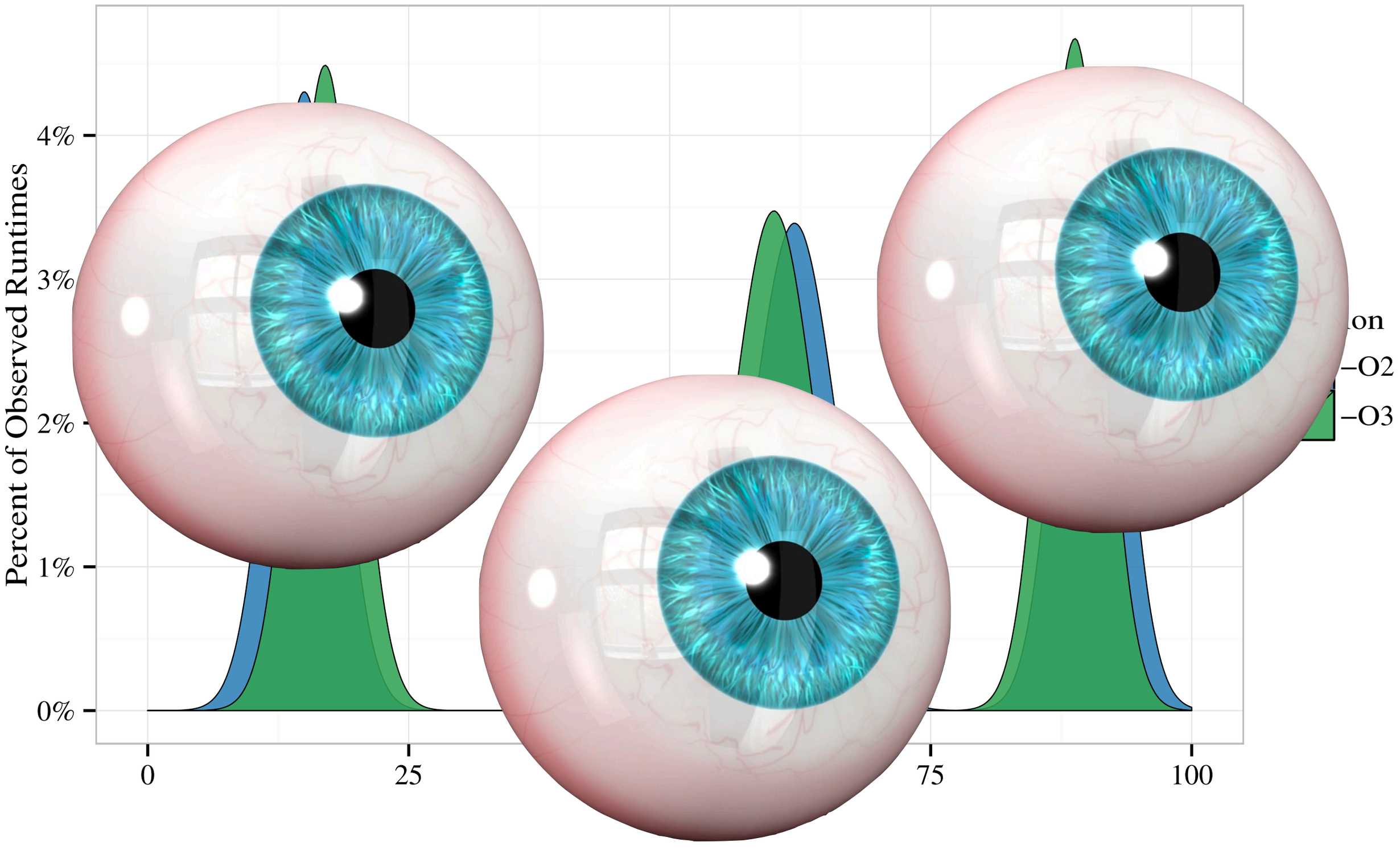
Is -O3 faster than -O2?



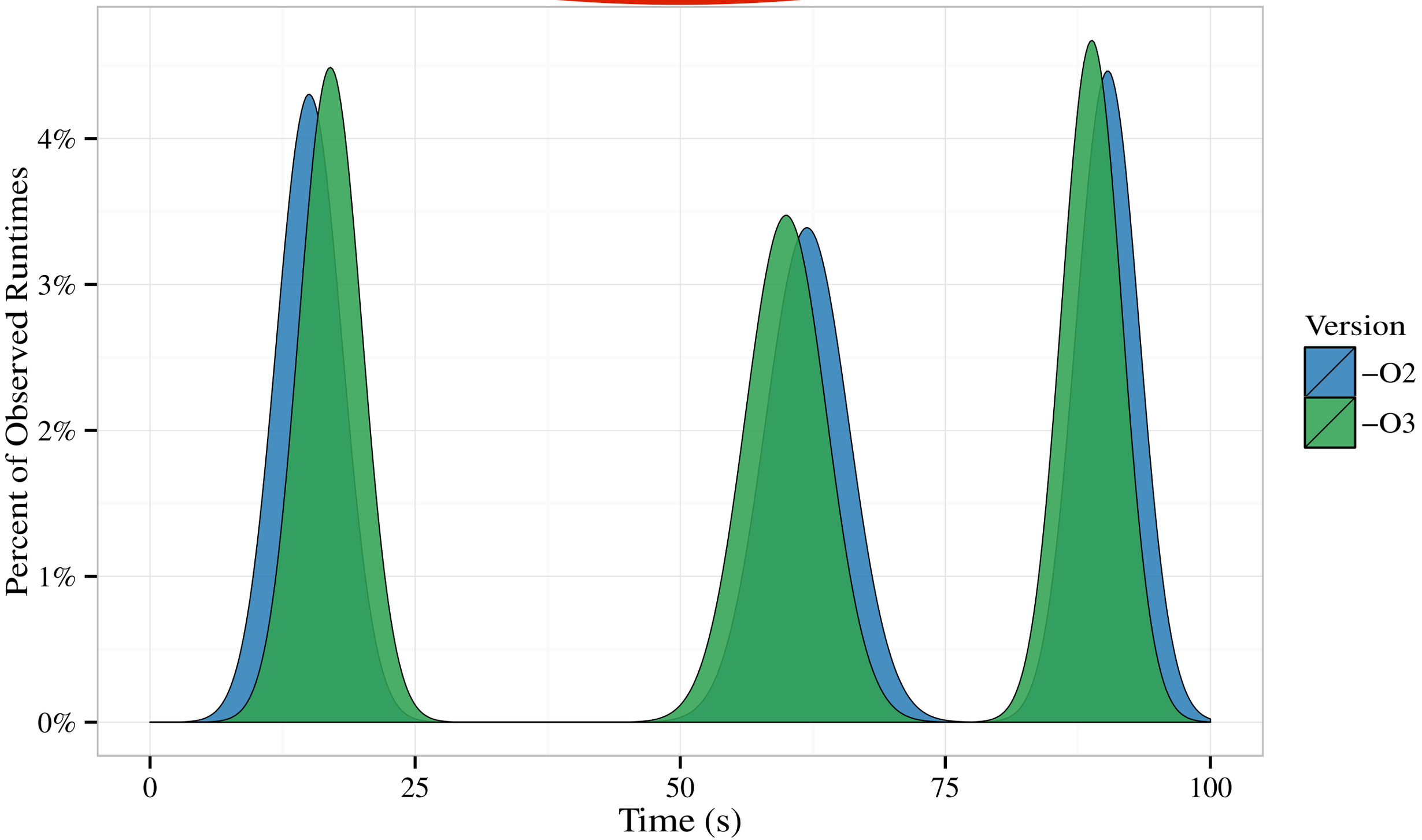
Is -O3 faster than -O2?



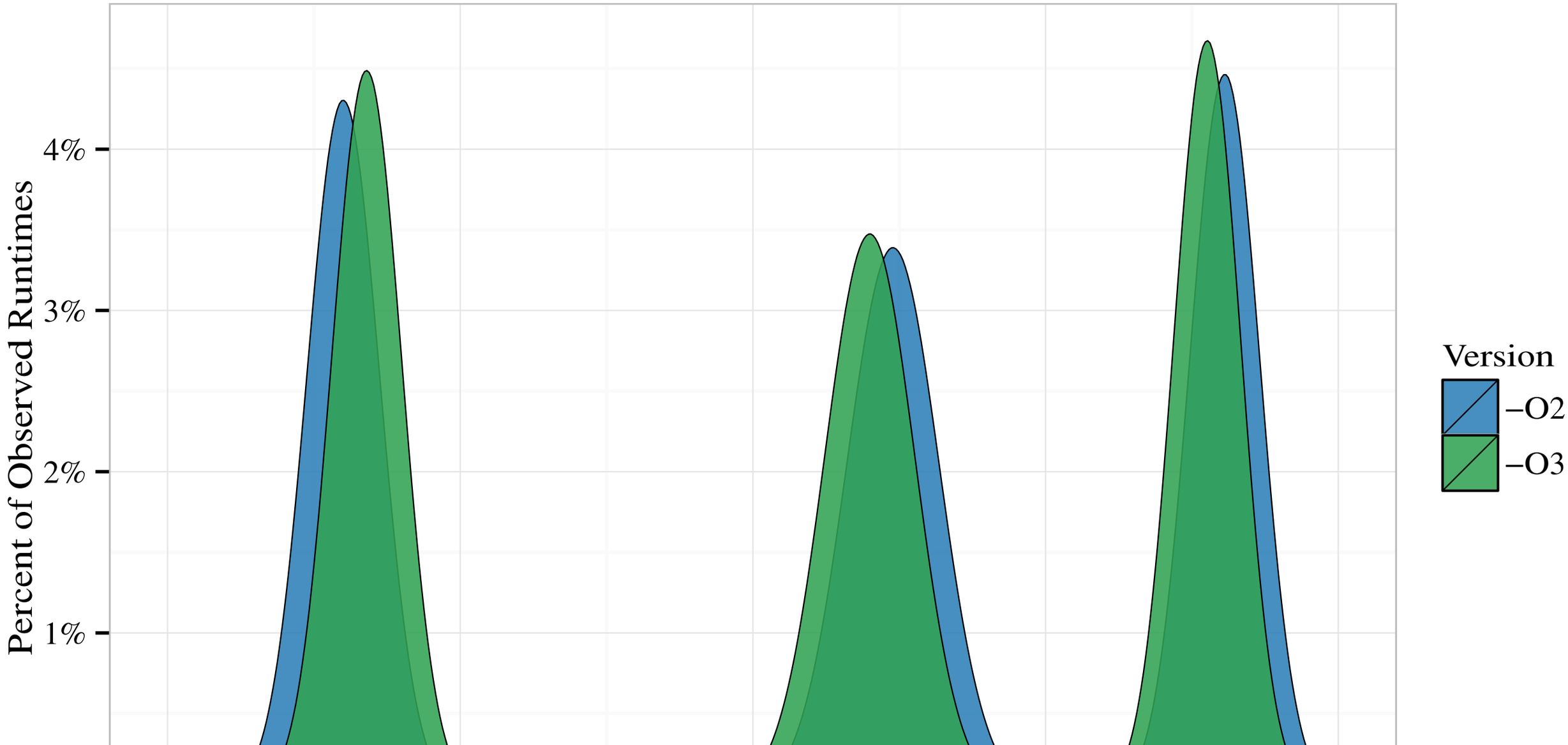
Is -O3 faster than -O2?



If -O3 = -O2

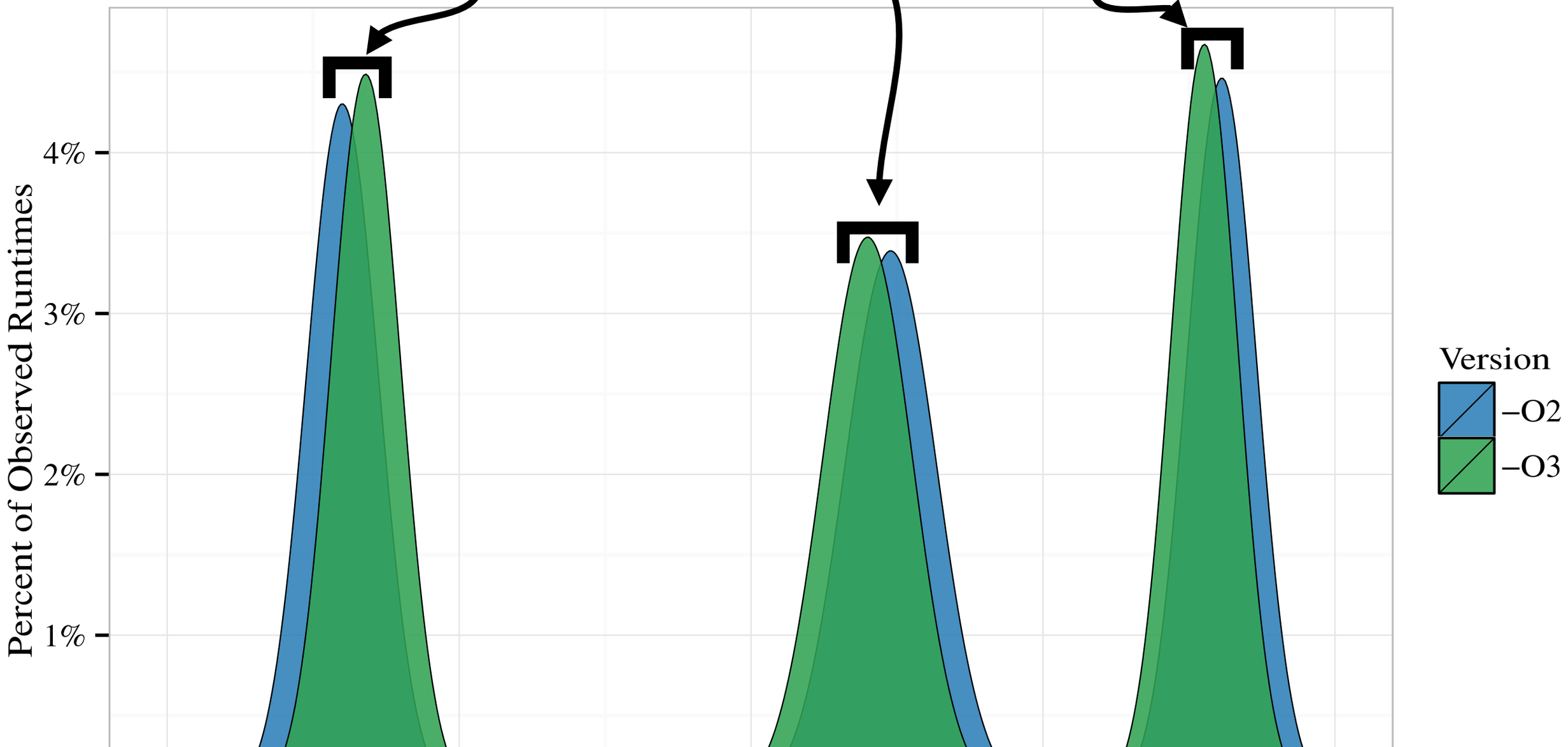

Percent of Observed Runtimes
4%
3%
2%
1%
0%
0
25
50
75
100
Time (s)
Version
–O2
–O3

If -O3 = -O2



If -O3 = -O2

what is the probability of measuring these differences?

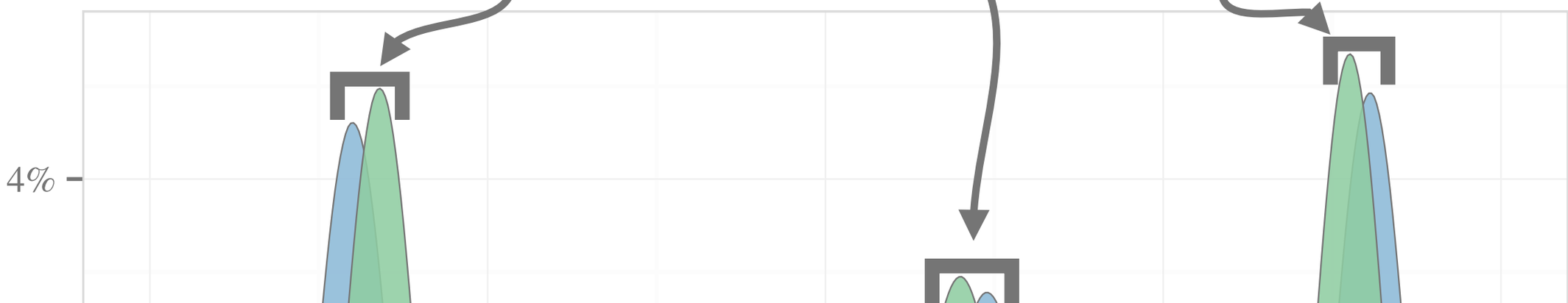


Analysis of Variance

`aov(time~opt+Error(benchmark/opt), times)`

If -O3 = -O2

what is the probability of measuring these differences?



Analysis of Variance

If p-value

If -O3 = -O2

what is the probability of measuring these differences?

Analysis of Variance

If p-value ≤ 5%

If -O3 = -O2

what is the probability of measuring these differences?

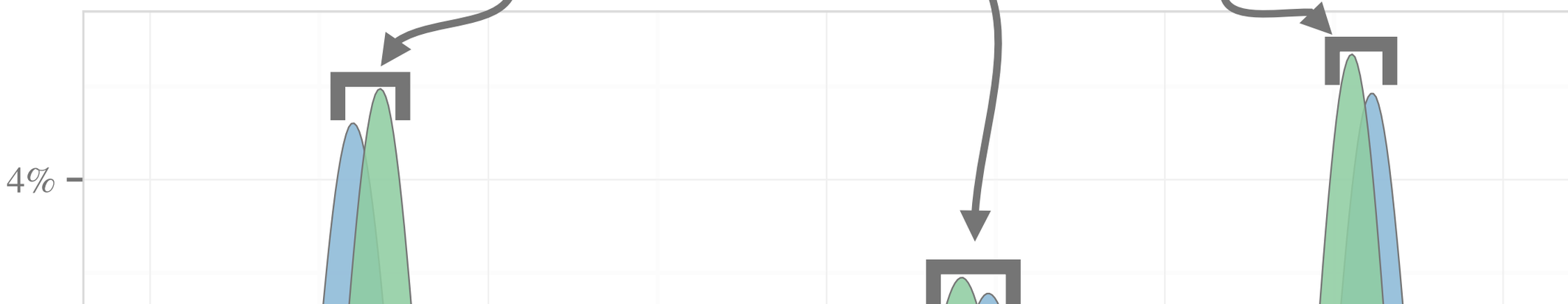
Analysis of Variance

If p-value ≤ 5%

we reject the null hypothesis

If -O3 = -O2

what is the probability of measuring these differences?



Analysis of Variance

If p-value ≤ 5%
we reject the null hypothesis

-O3 vs -O2

p-value = 26.4%

one in four experiments will show an effect that does not exist!

Analysis of Variance

If p-value ≤ 5%
we reject the null hypothesis

-O3 vs -O2

p-value = 26.4%

fail to reject the null hypothesis

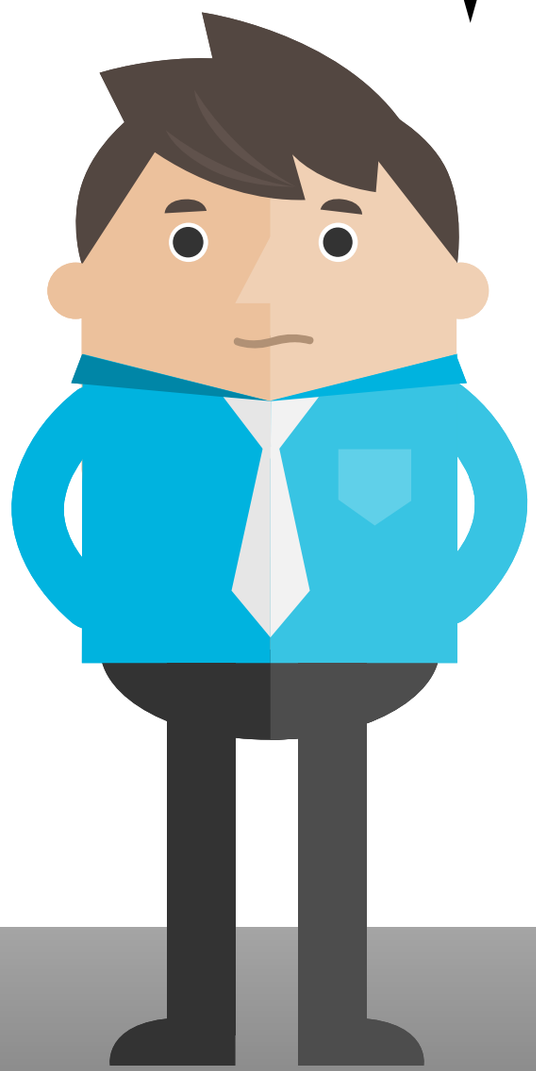
Analysis of Variance

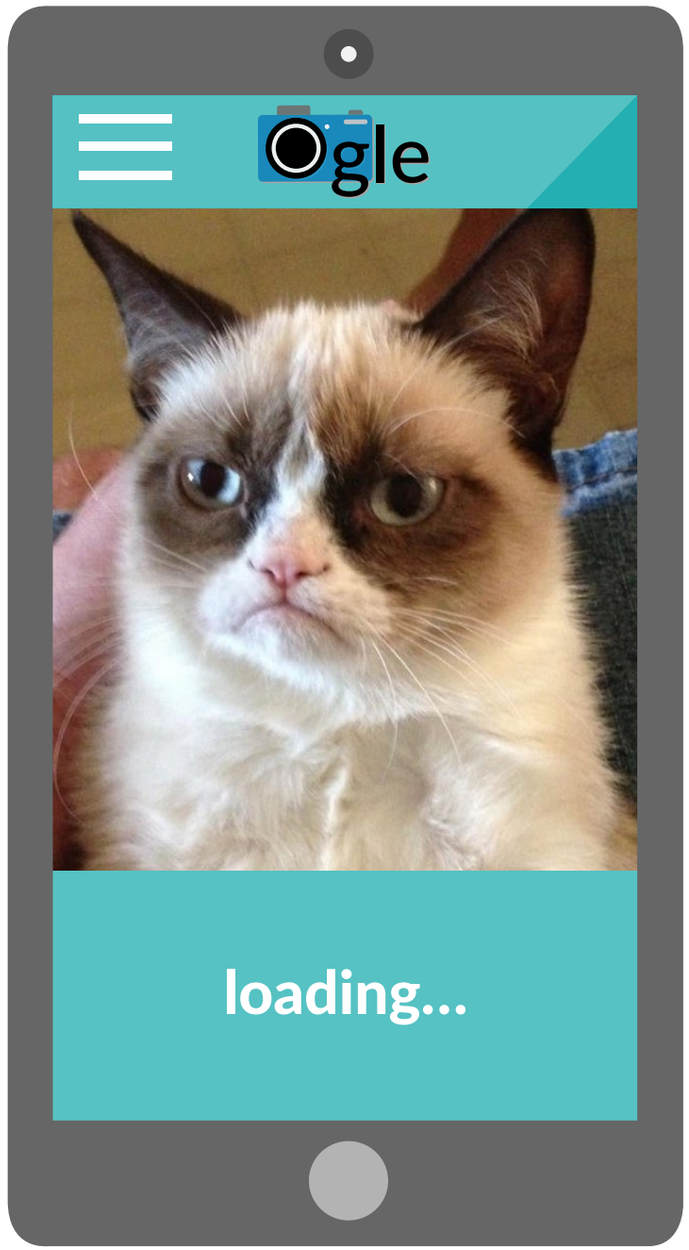
If p-value ≤ 5%

we reject the null hypothesis

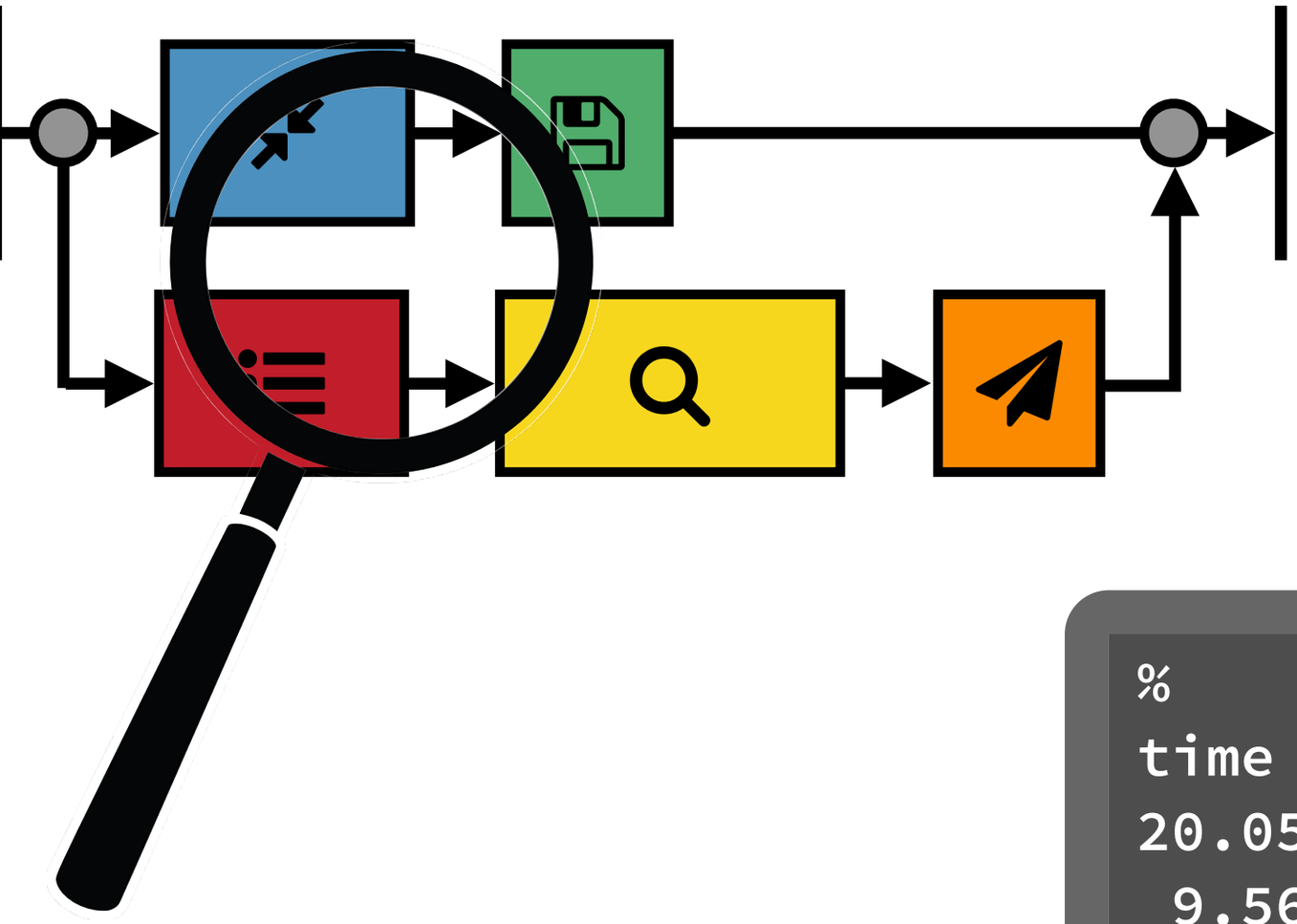
The effect of **-O3** over **-O2** is indistinguishable from noise

…did you try -O**9**?





Software Profilers

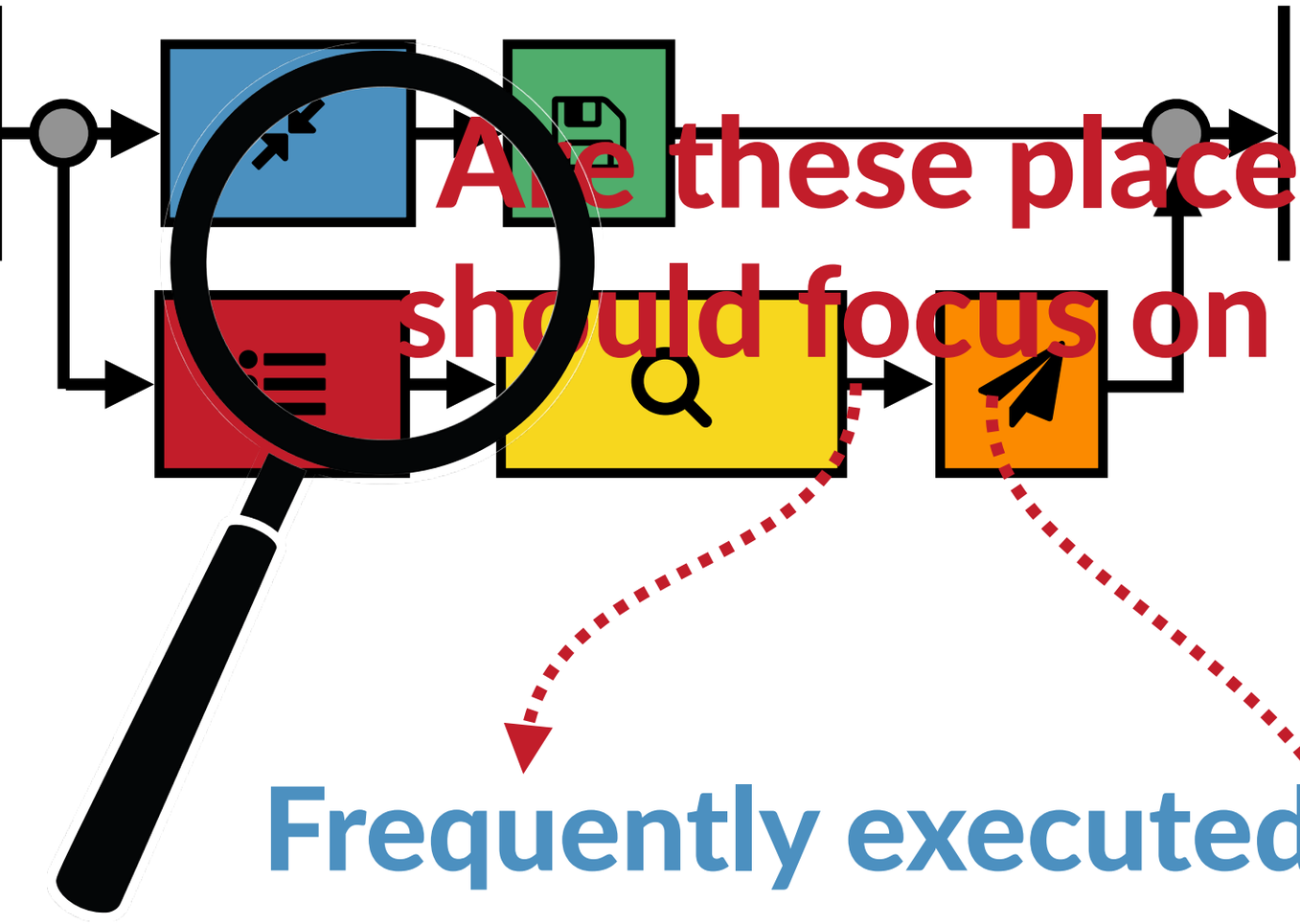


Number of calls to each function

Runtime for each function

```
%       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
```

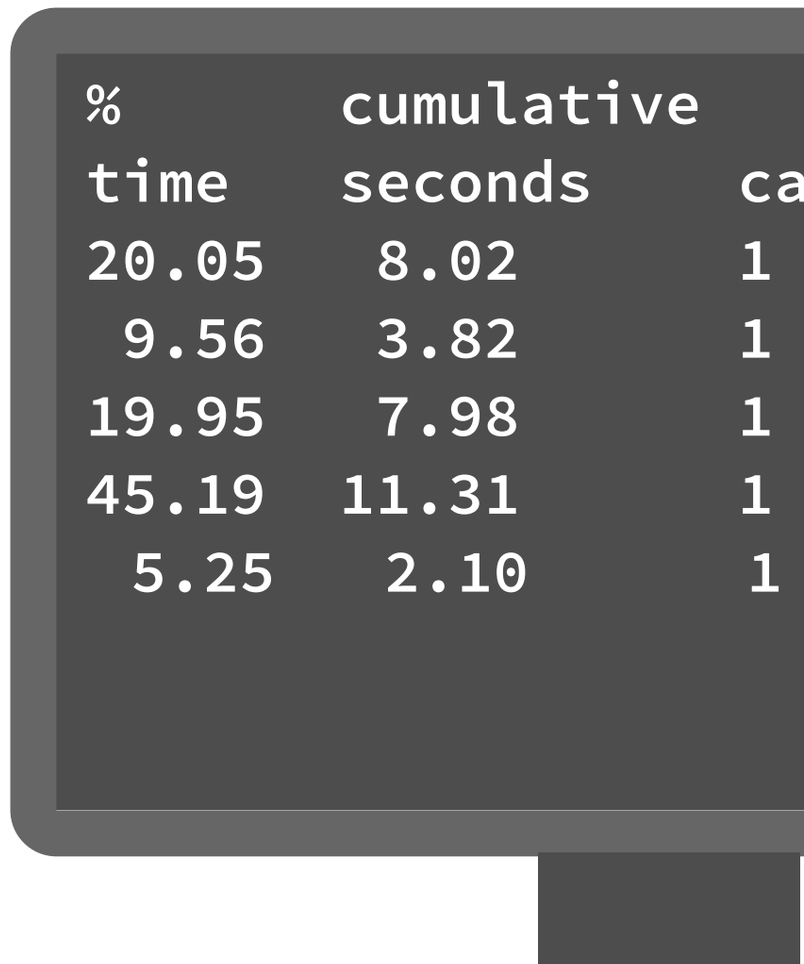
Software Profilers



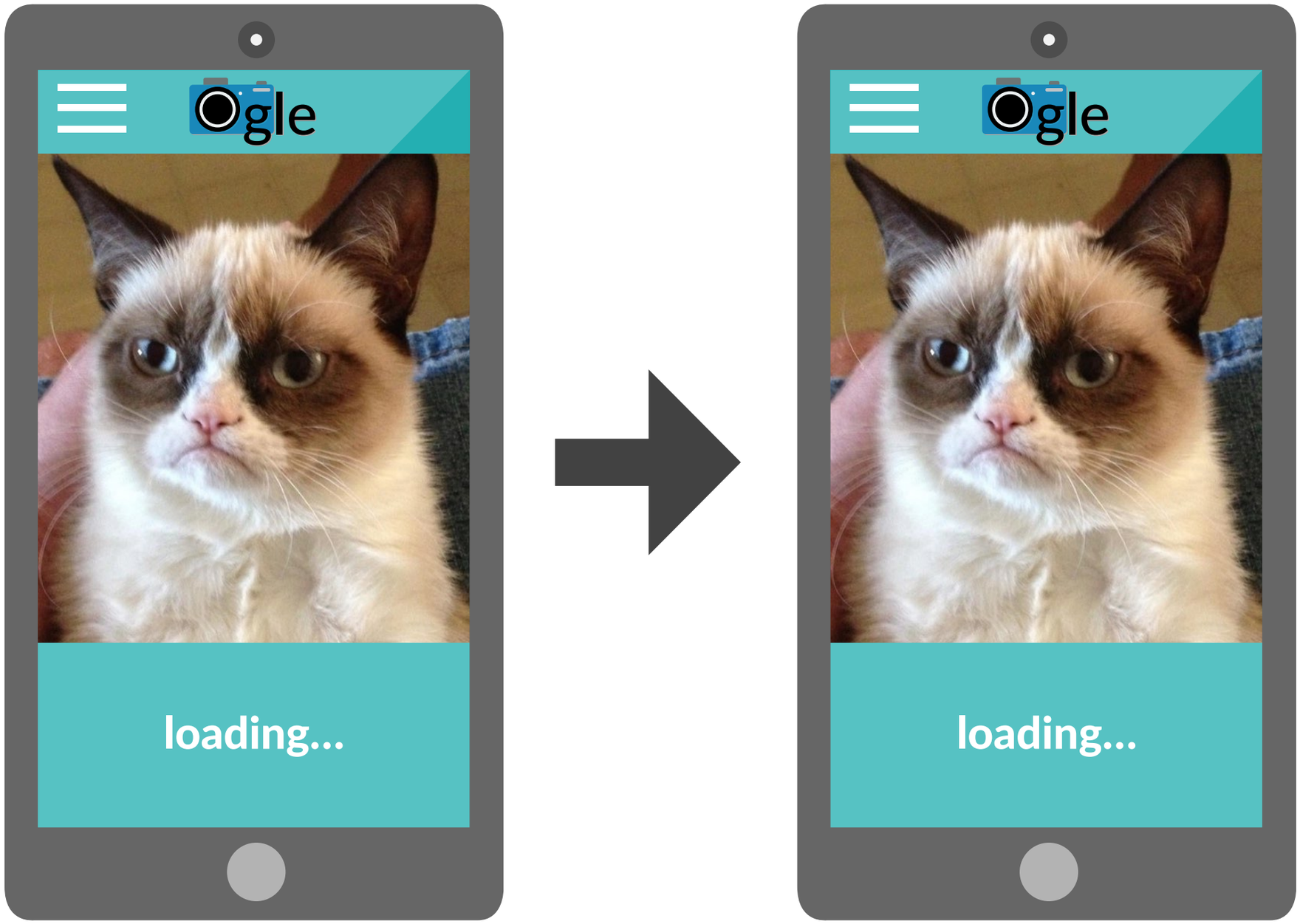
Are these places where Luke should focus on performance?

Frequently executed code

Code that runs for a long time



Would this speed up Ogle?

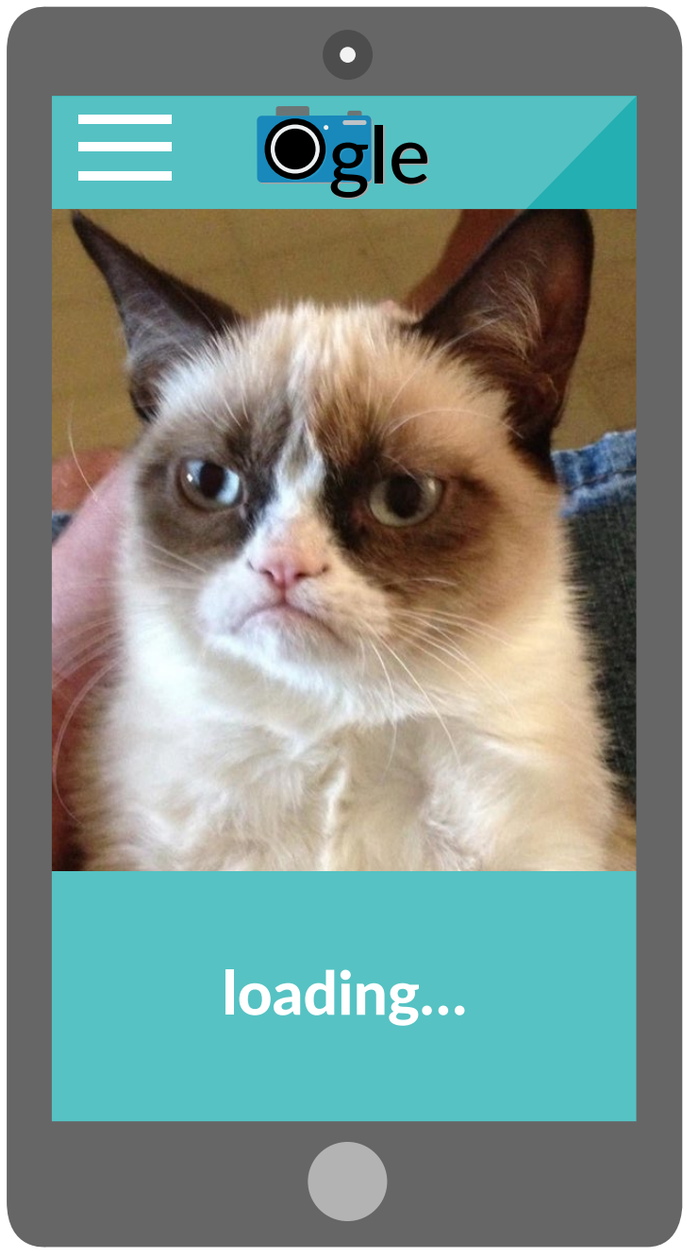


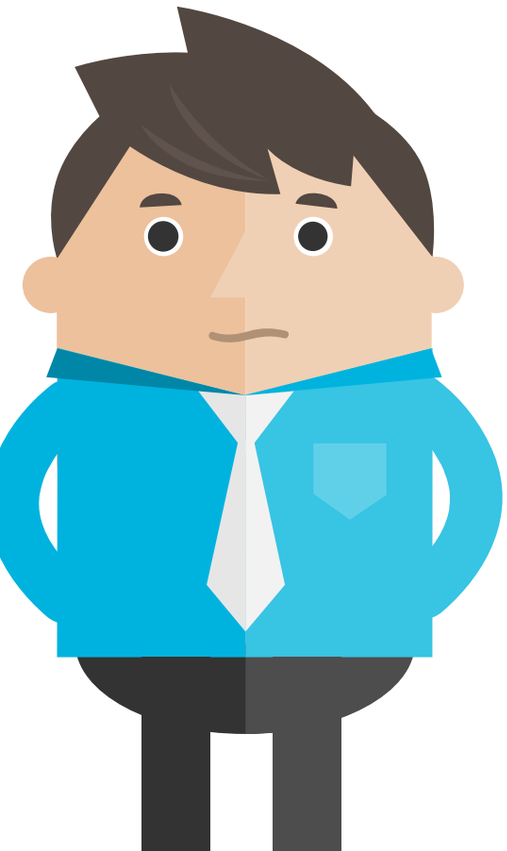
Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

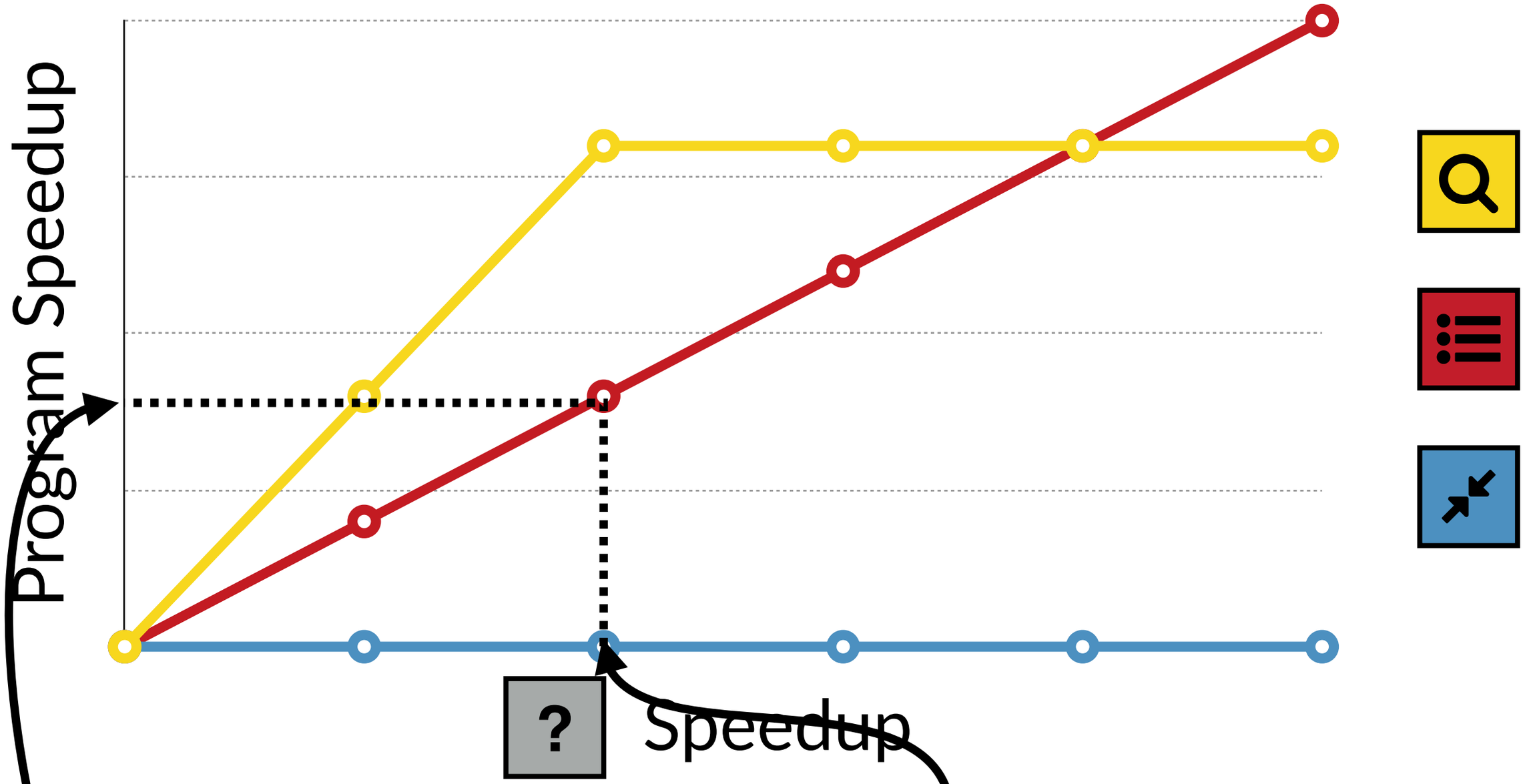
Profilers do a bad job finding important code in parallel / asynchronous / concurrent programs.





We need to do better.

Causal Profile

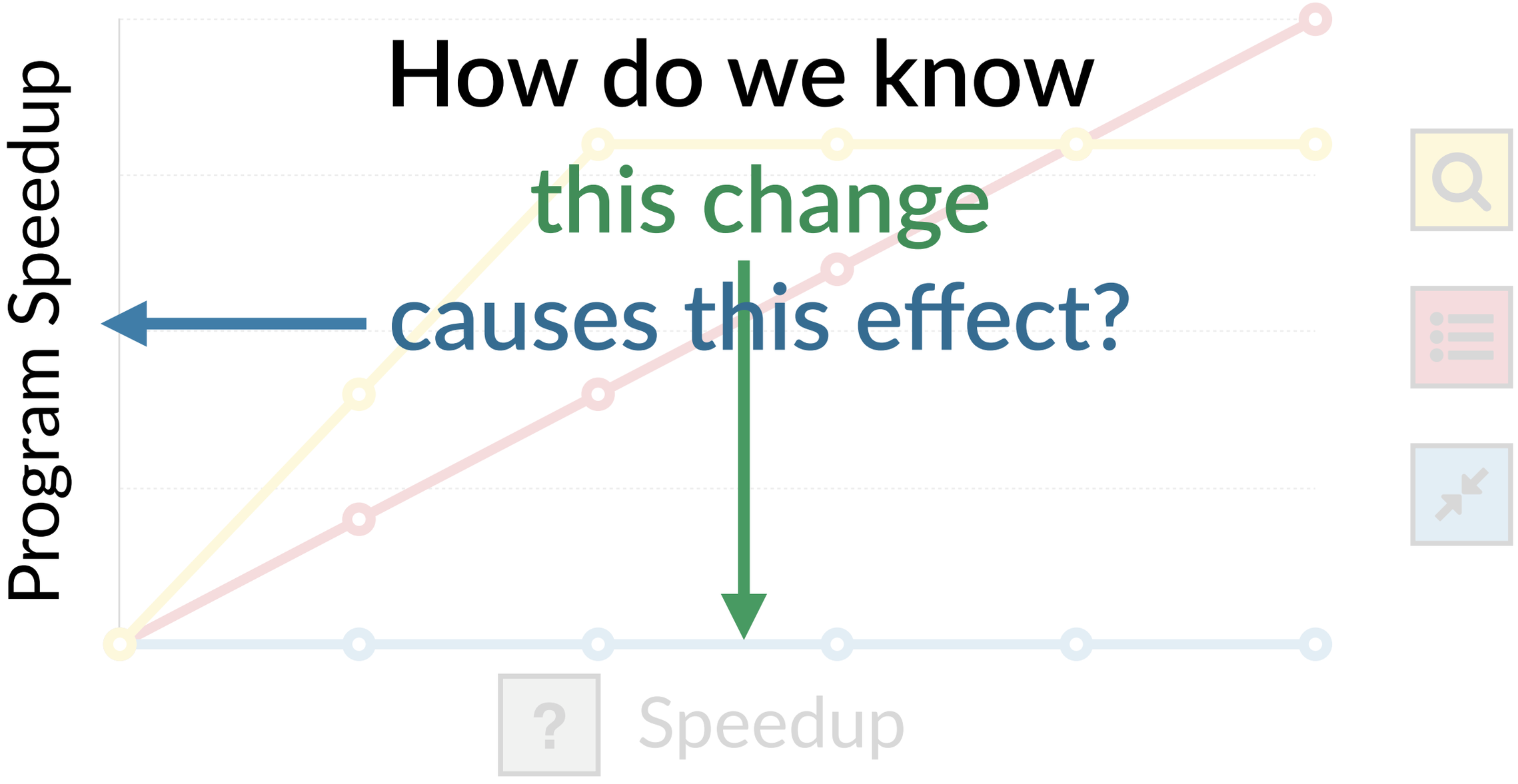


Causal Profile

How do we know

this change

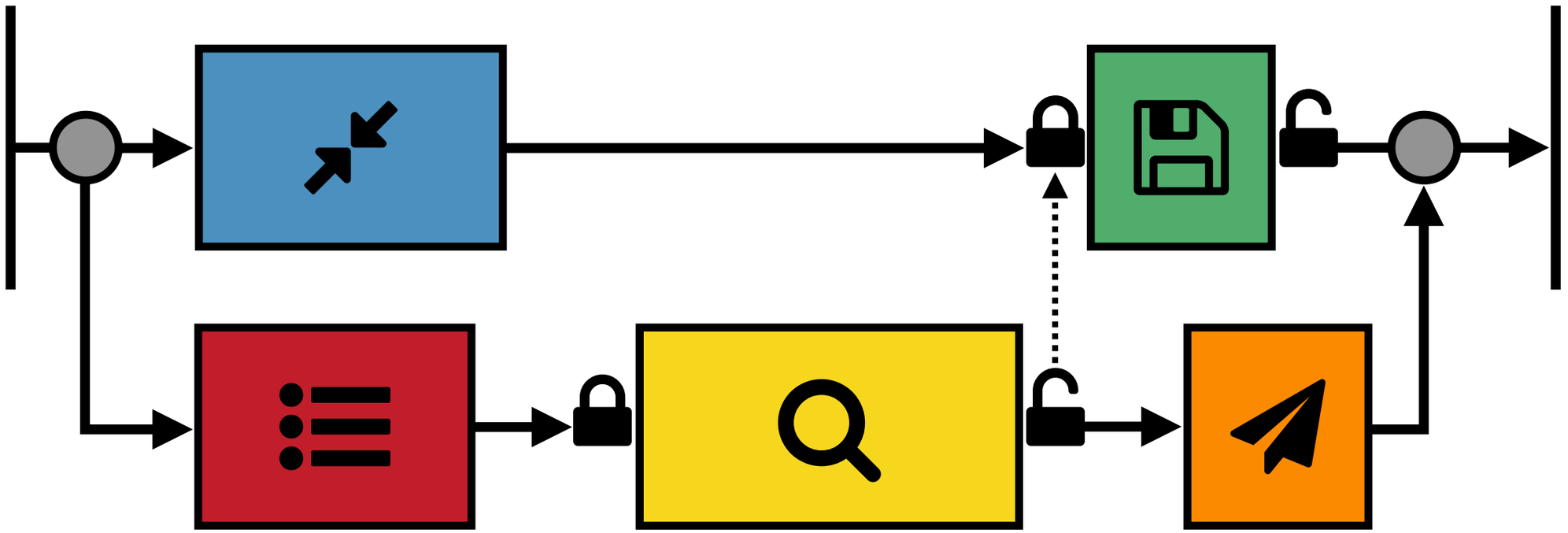
causes this effect?



Run an experiment

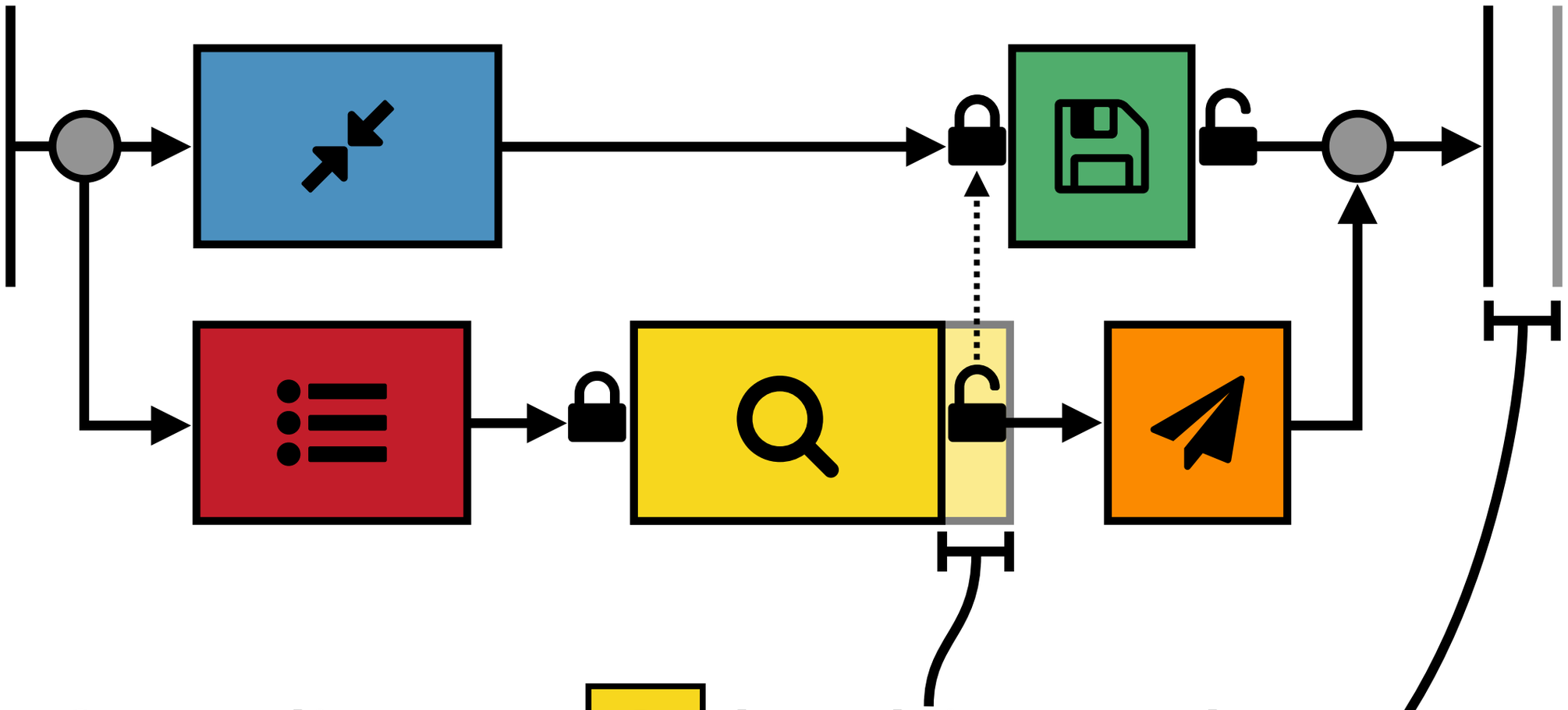
Performance Experiments

If we could magically speed up  ...



Performance Experiments

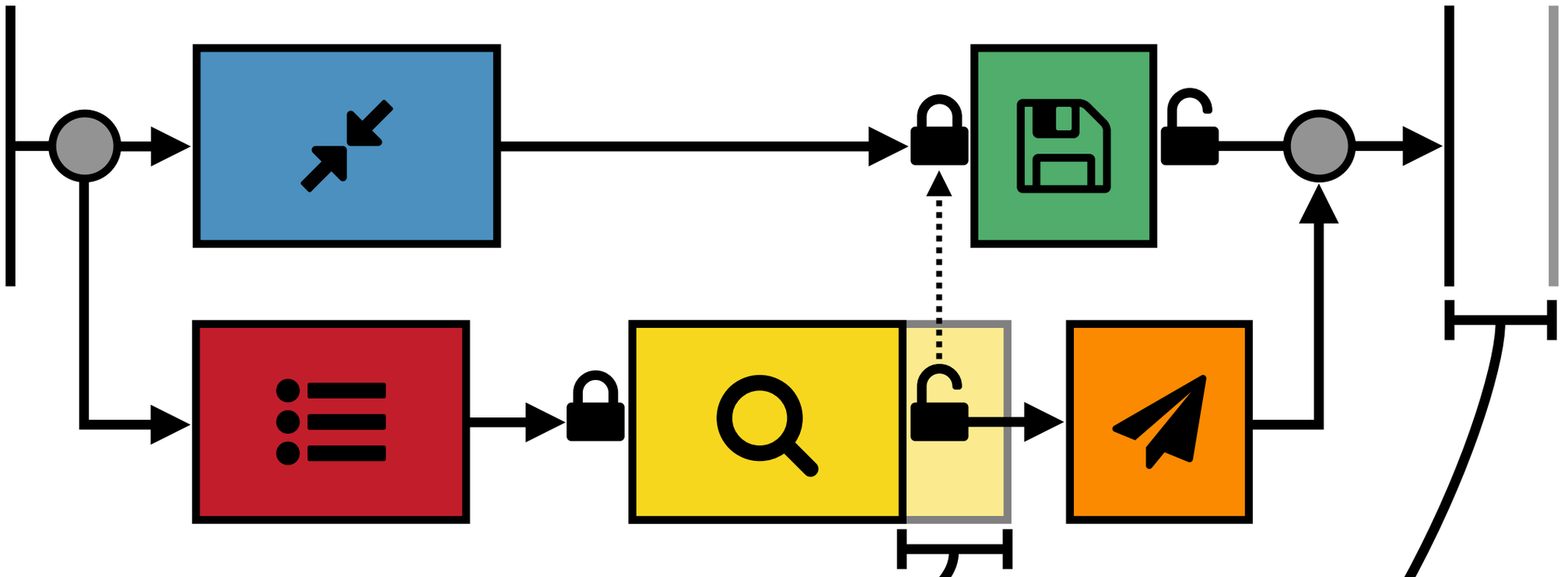
If we could magically speed up 🔍 ...



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

Performance Experiments

If we could magically speed up 🔍 …

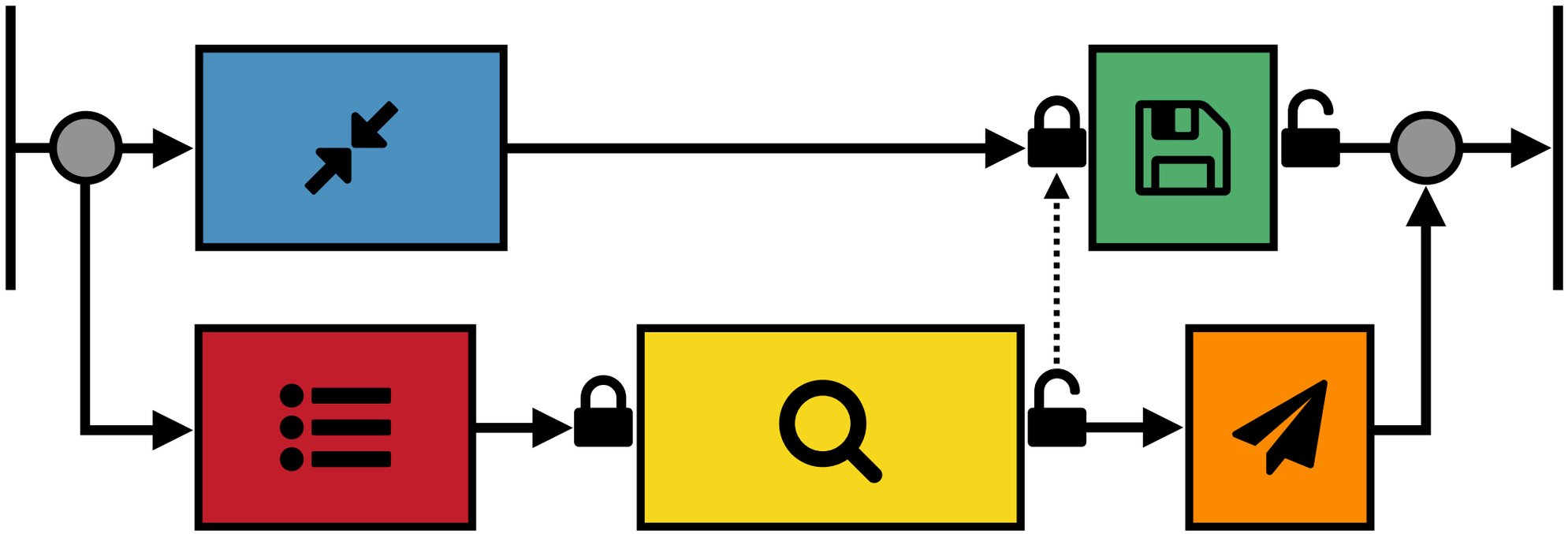


More speedup in 🔍 …

leads to a larger program speedup.

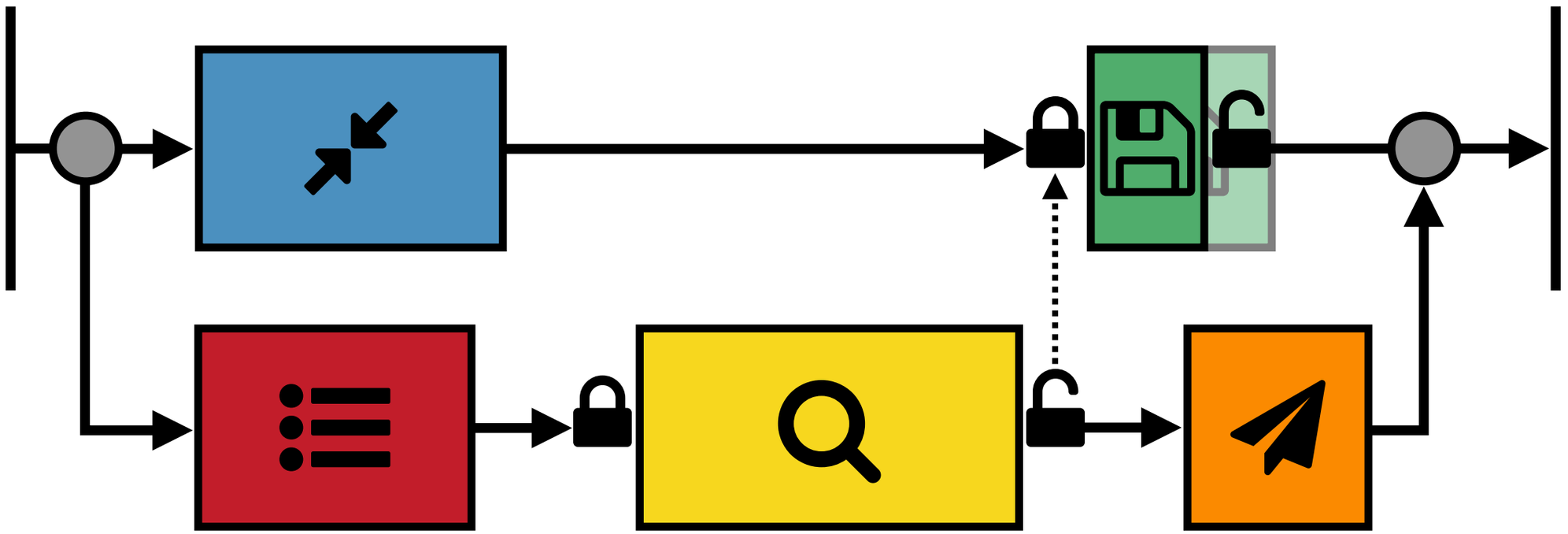
Performance Experiments

If we could magically speed up 💾 ...



Performance Experiments

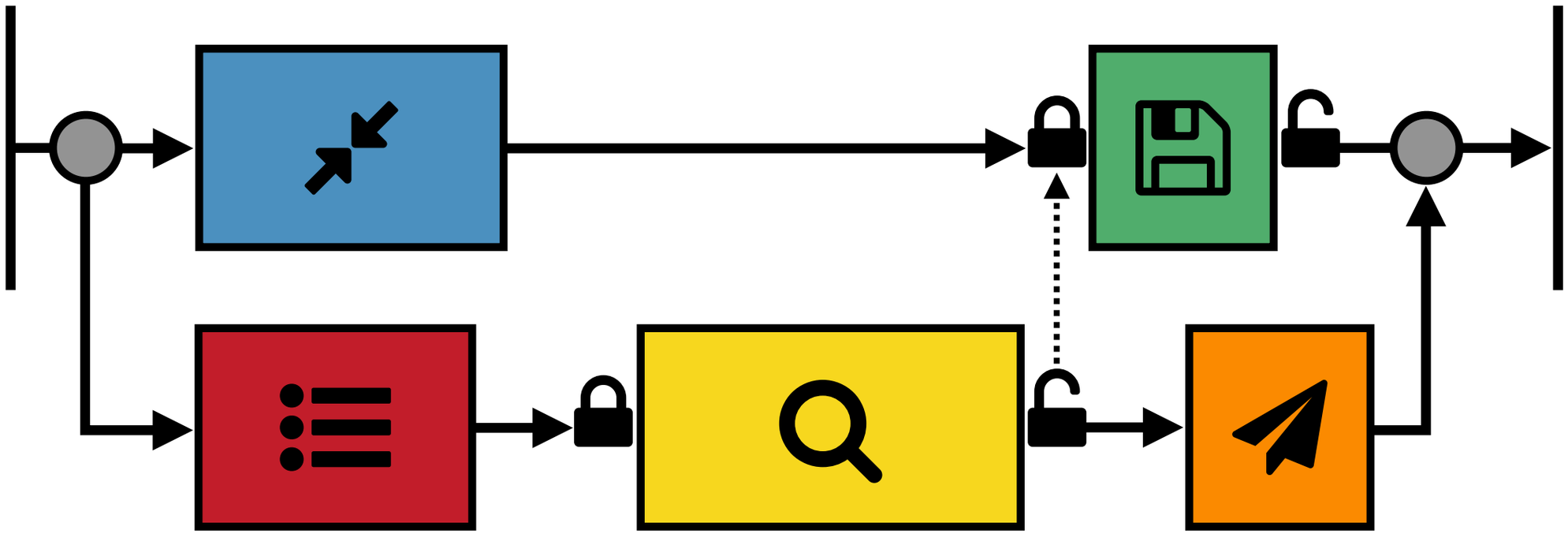
If we could magically speed up  ...



No program speedup

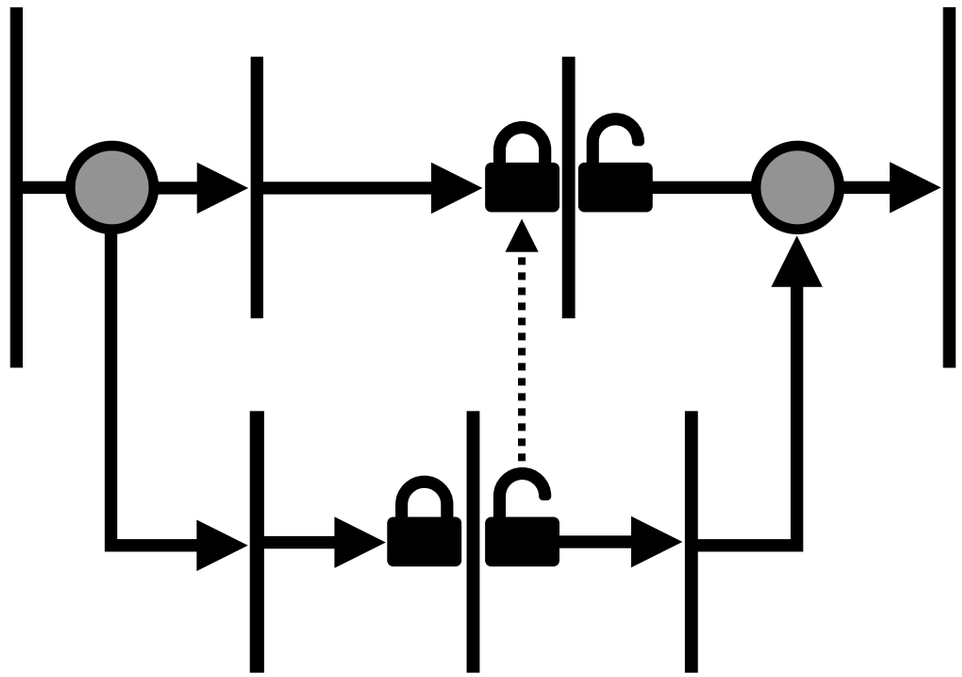
Performance Experiments

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



Performance Experiments

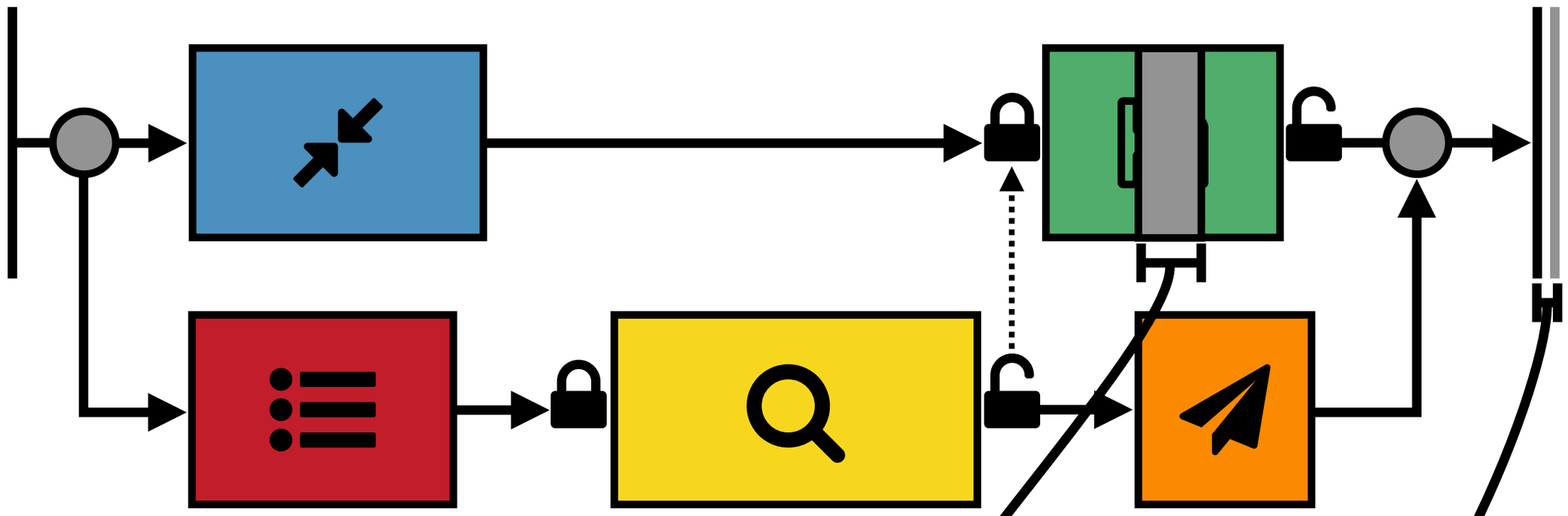
We’re going to have to do this *without magic*…



Otherwise, we’d just do this…

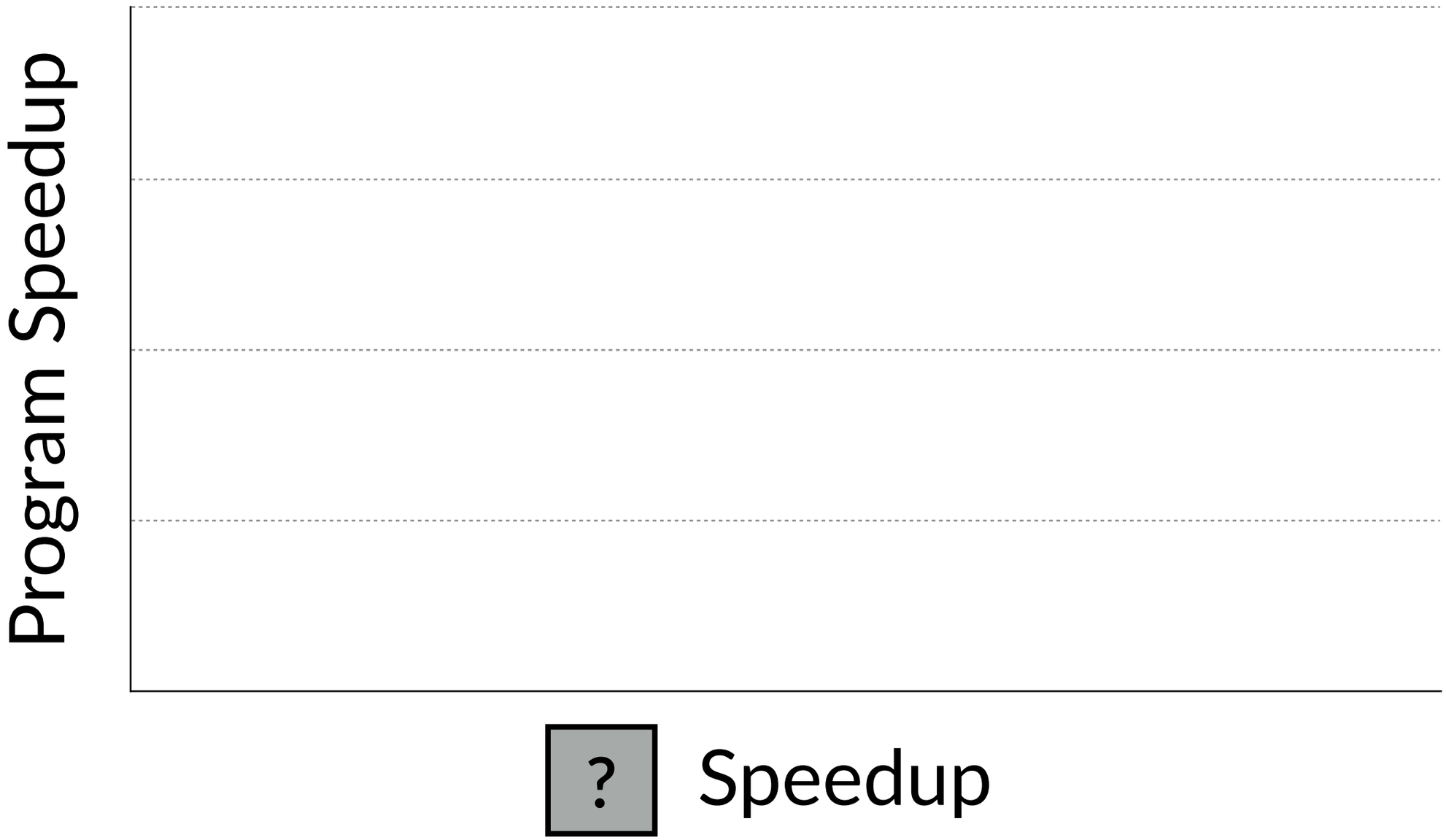
Virtual Speedup

“Speed up”  by slowing everything else down.

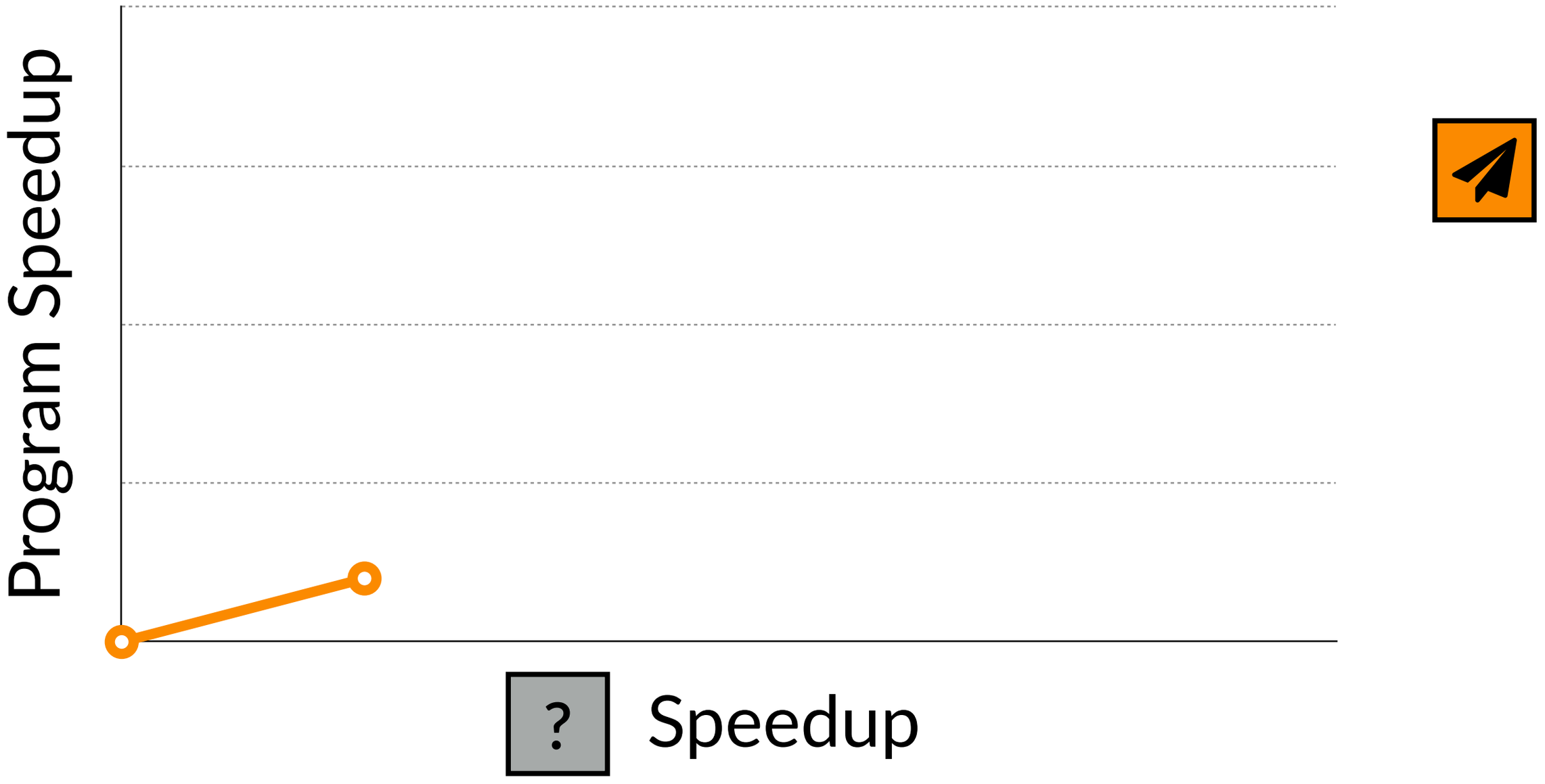


Speeding up  by this much…
speeds up the program by this much.

Speedup Results

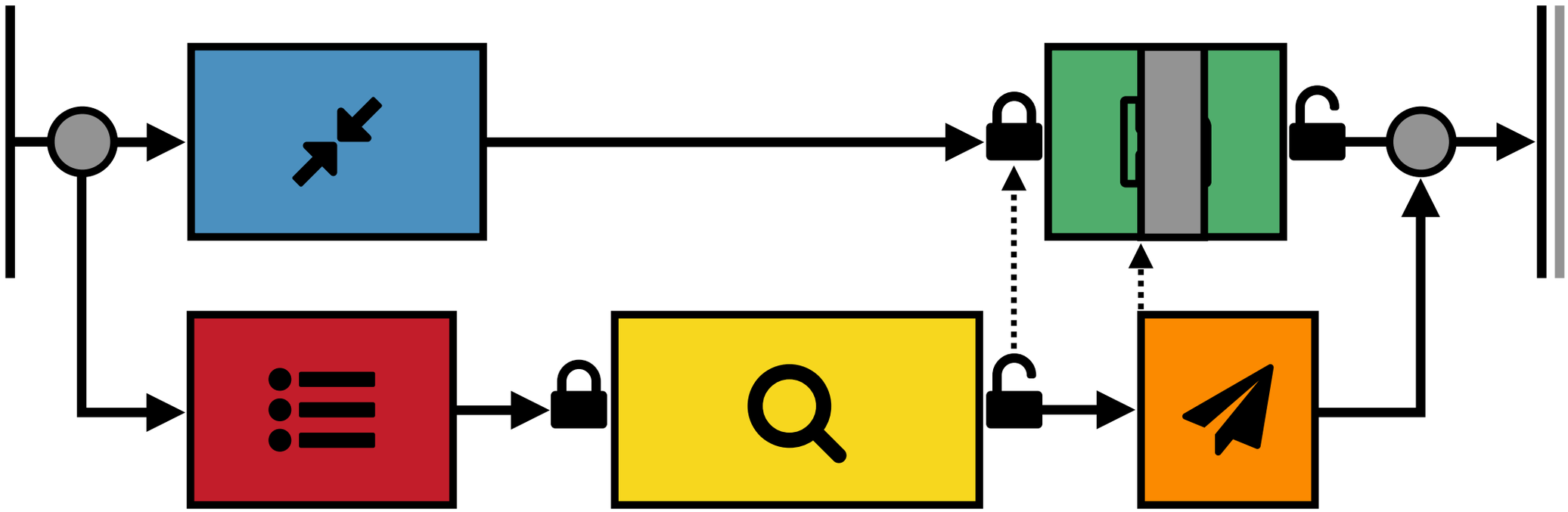


Speedup Results



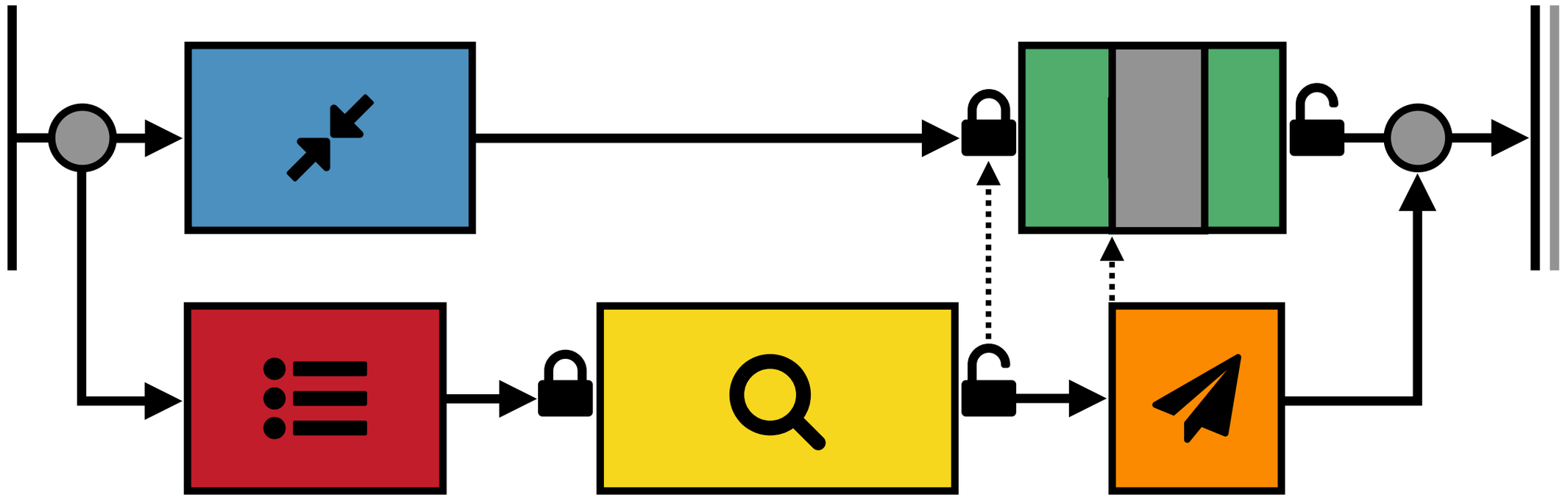
Virtual Speedup

“Speed up”  by slowing everything else down.



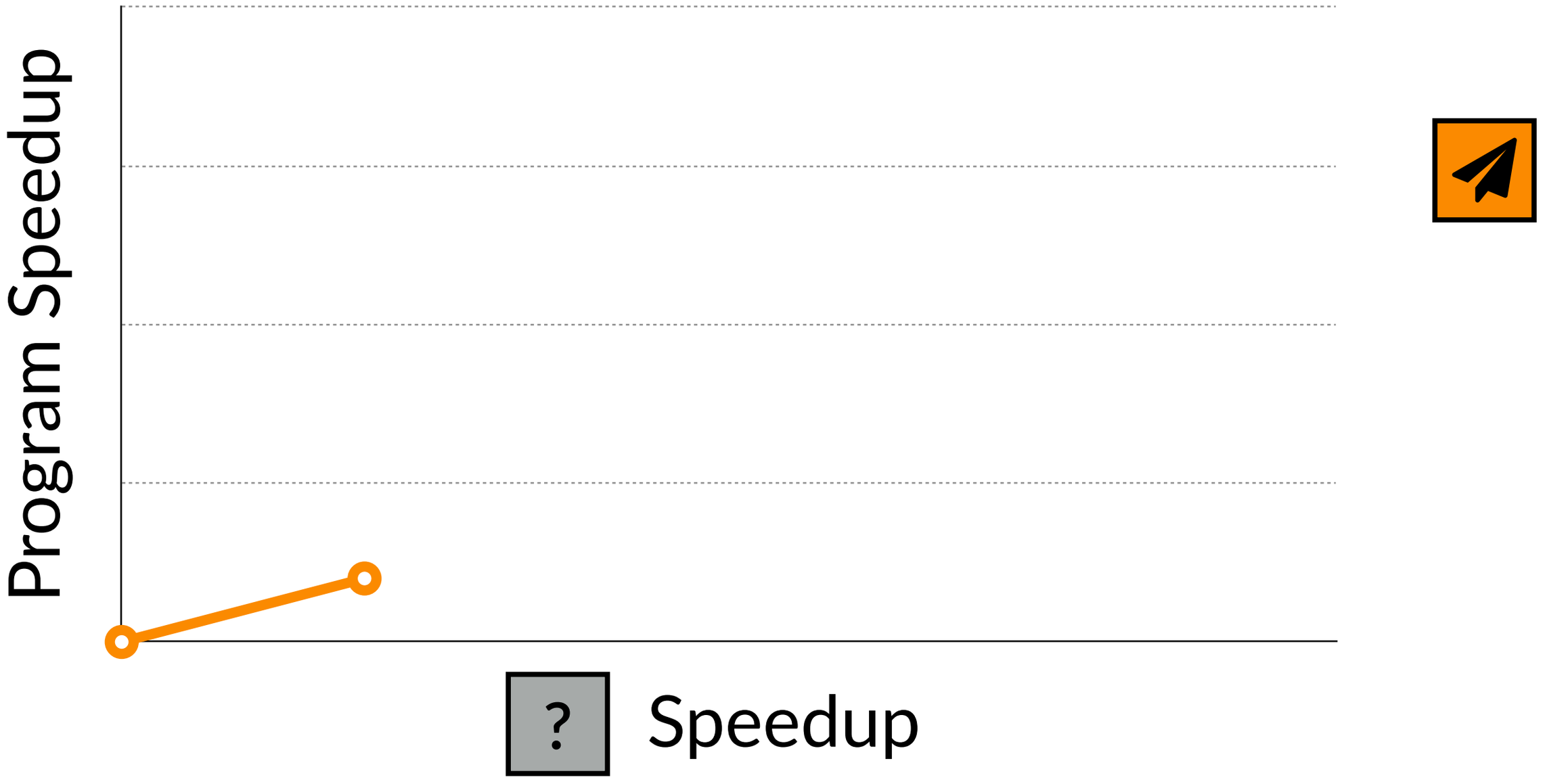
Virtual Speedup

“Speed up”  by slowing everything else down.

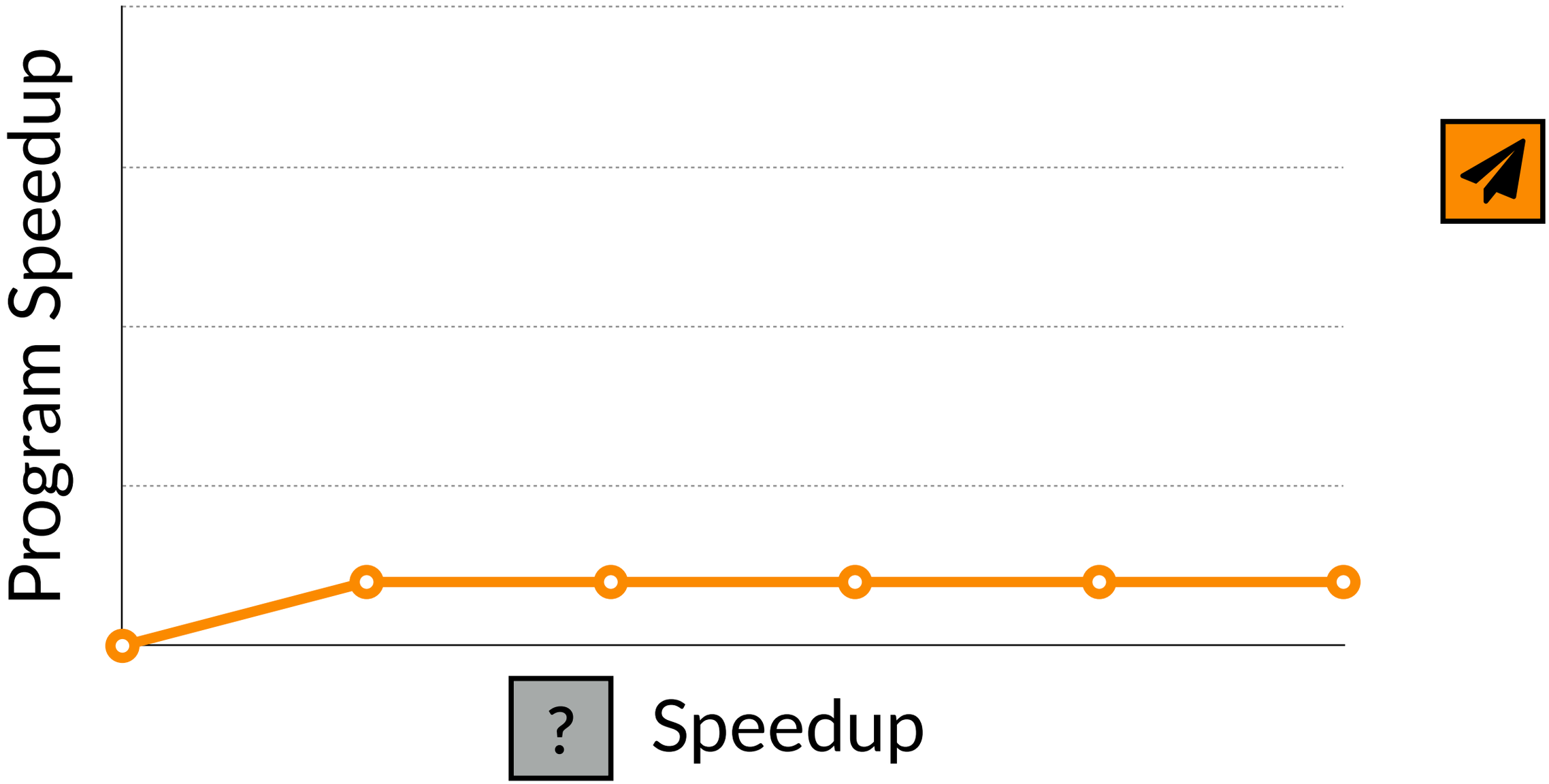


A larger speedup has no additional effect

Speedup Results

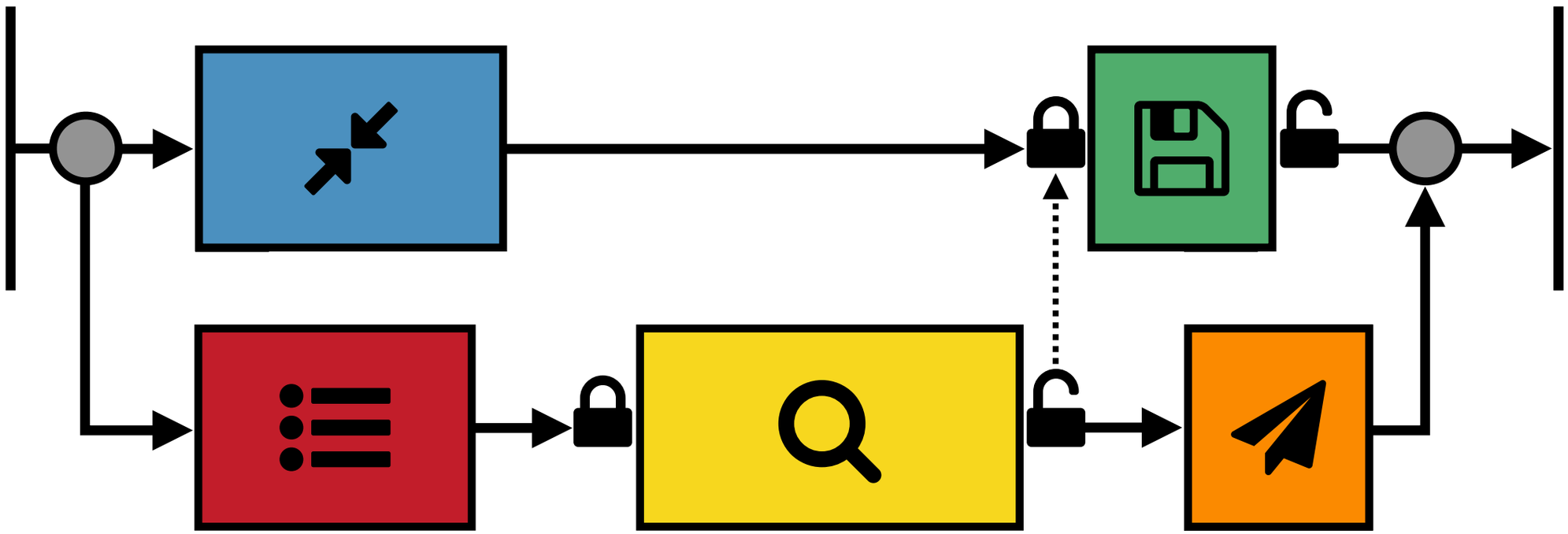


Speedup Results



Virtual Speedup

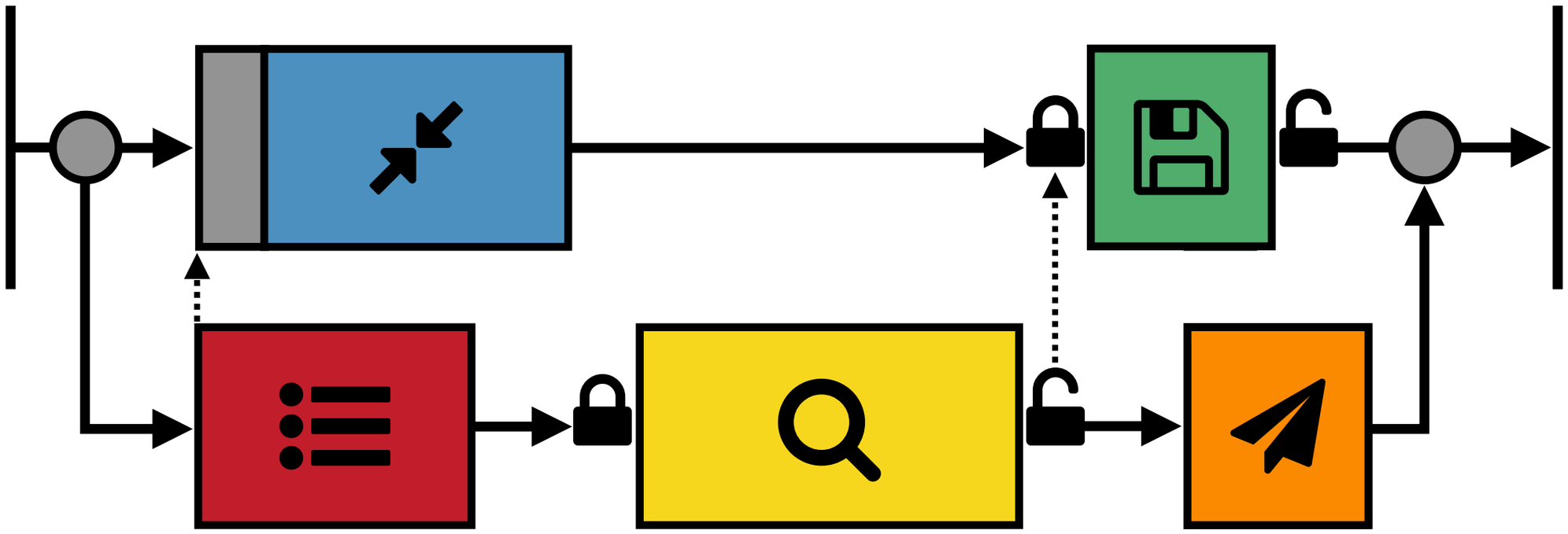
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

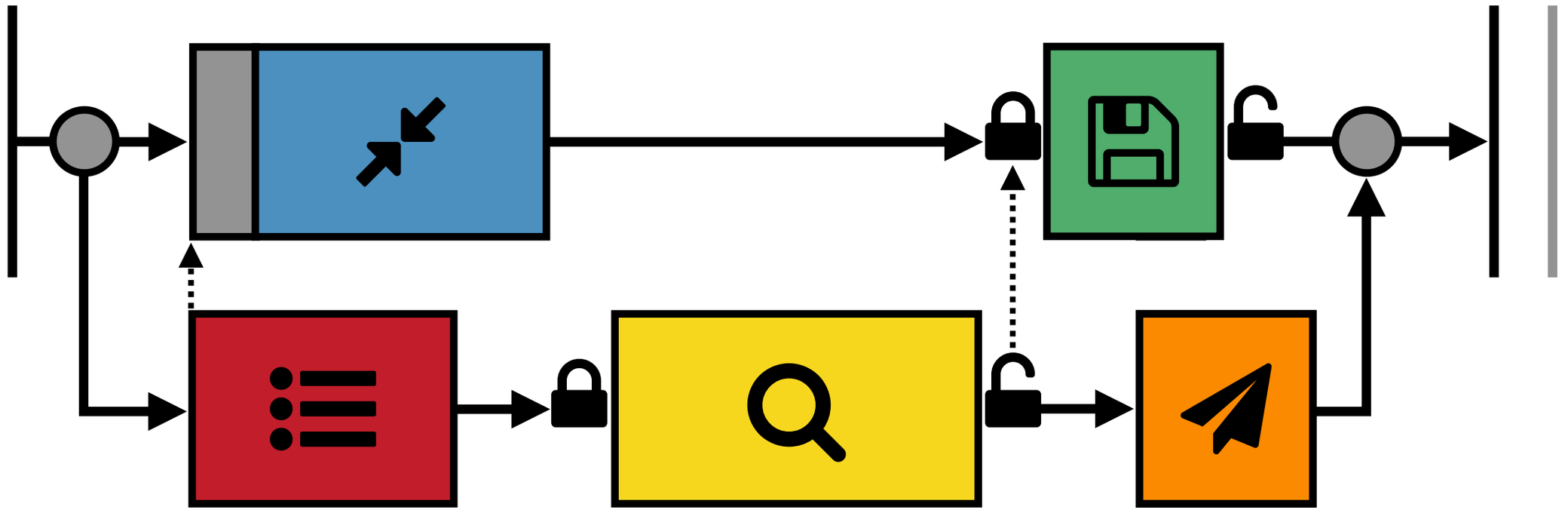
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

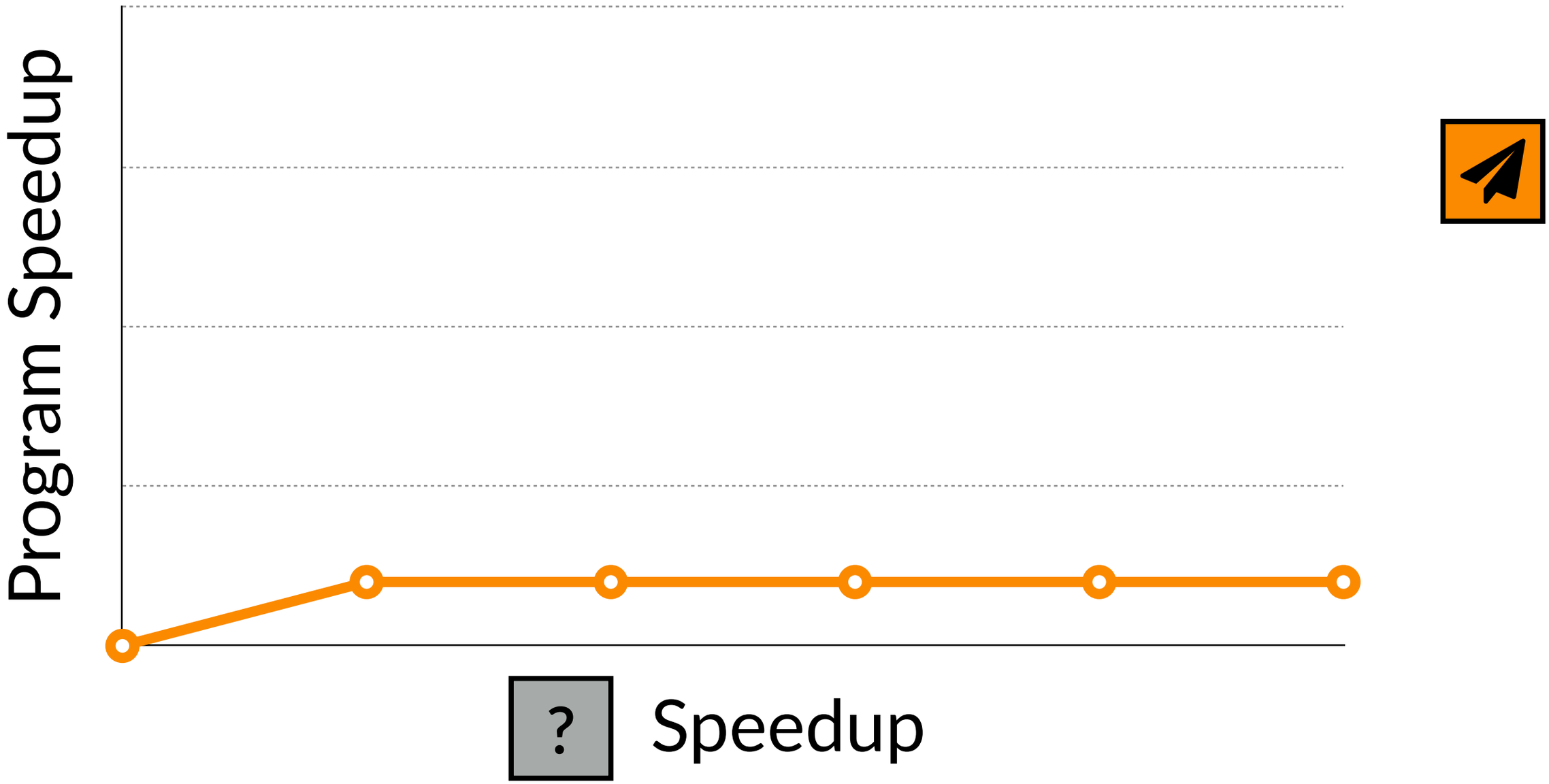
Virtual Speedup

“Speed up”  by slowing everything else down.

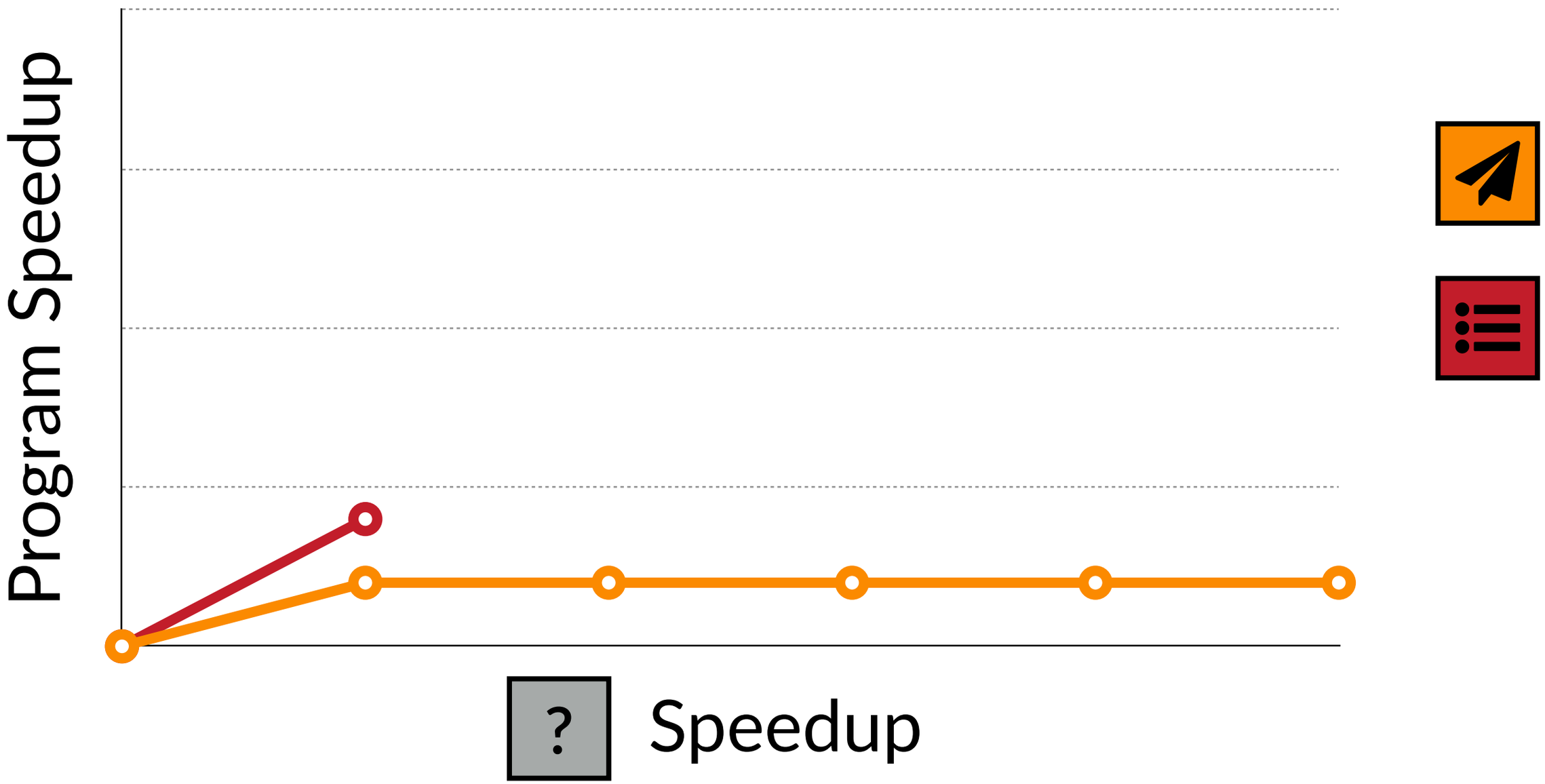


Each time  runs, pause all other threads.

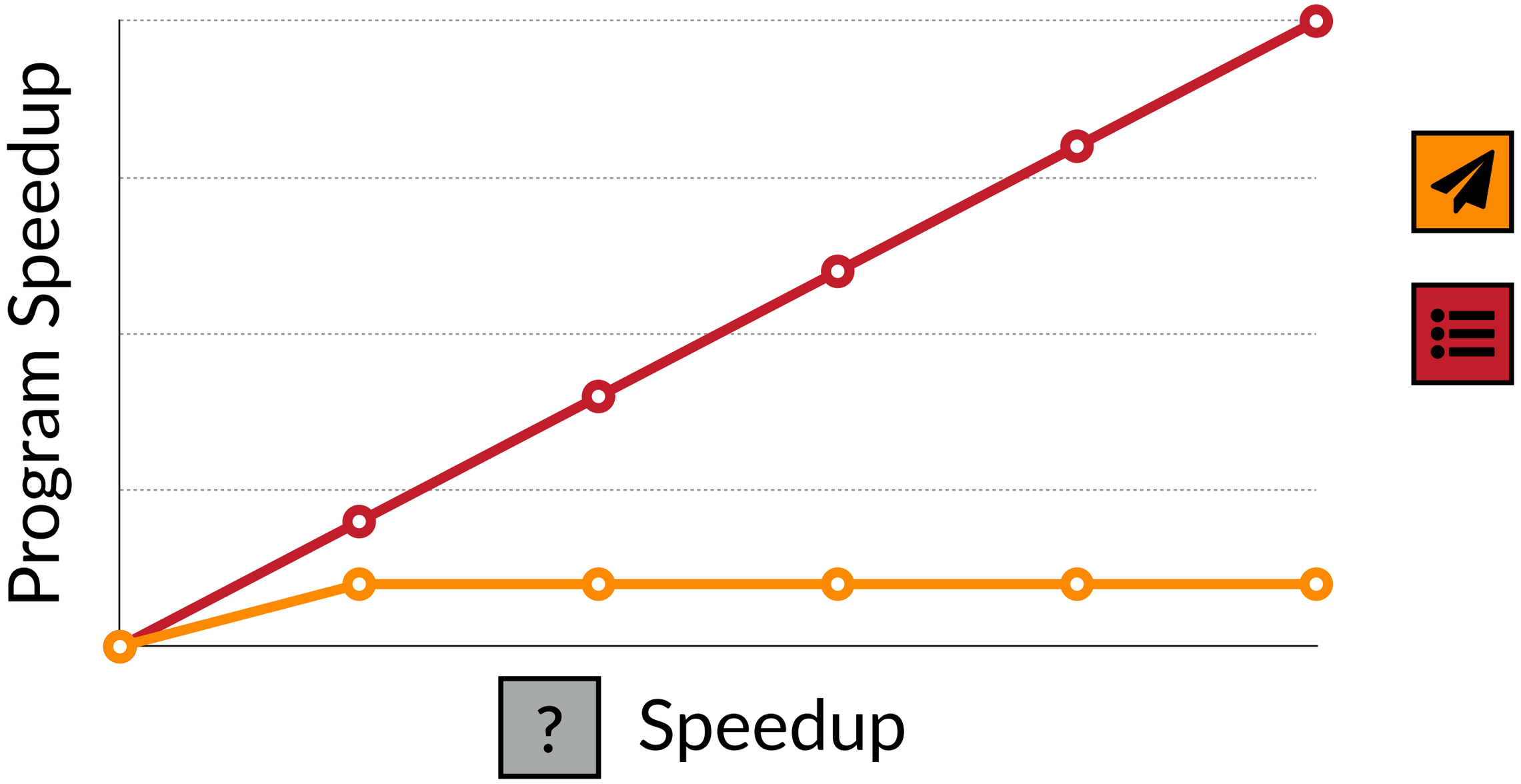
Speedup Results



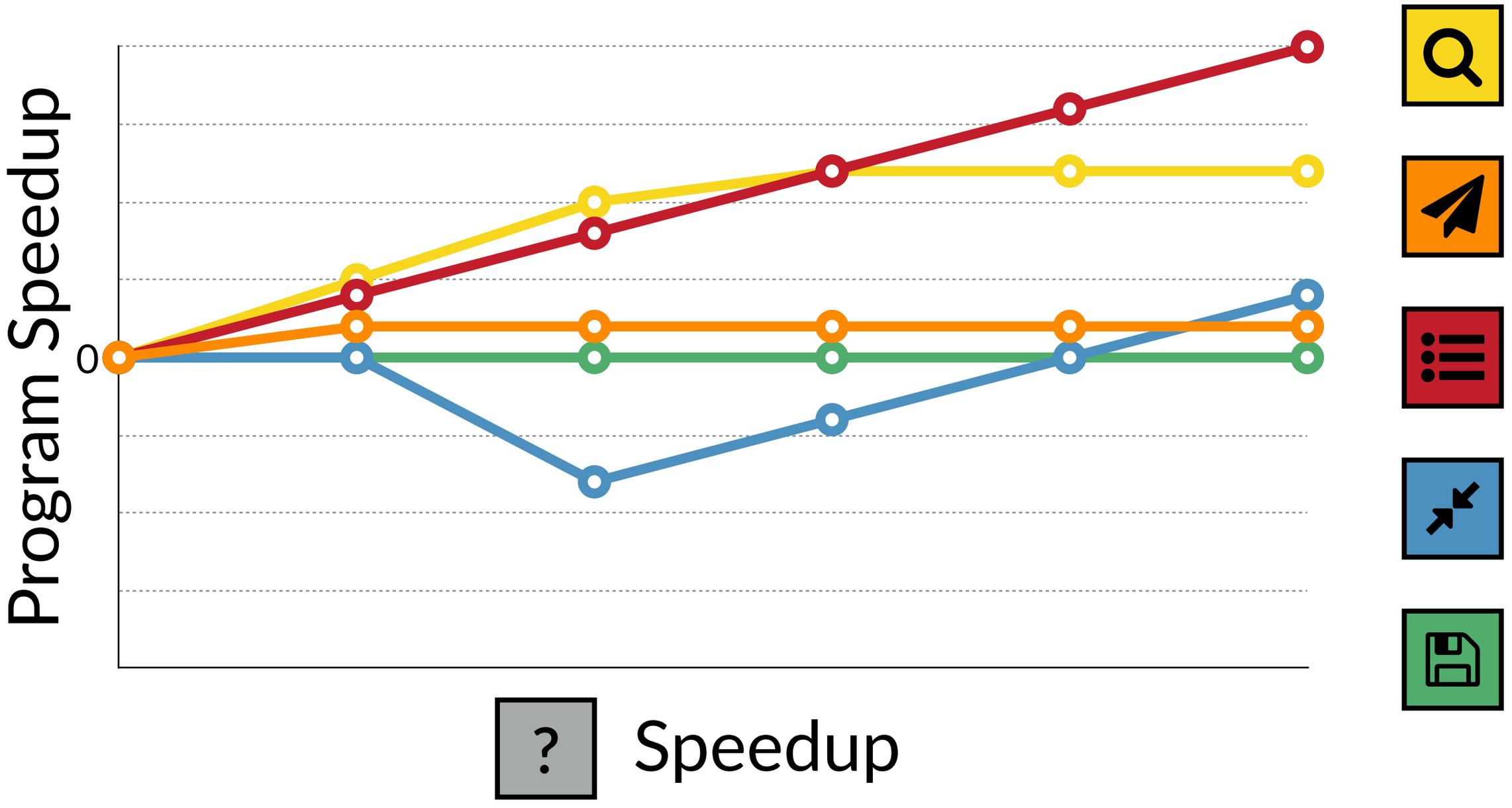
Speedup Results



Speedup Results



Speedup Results

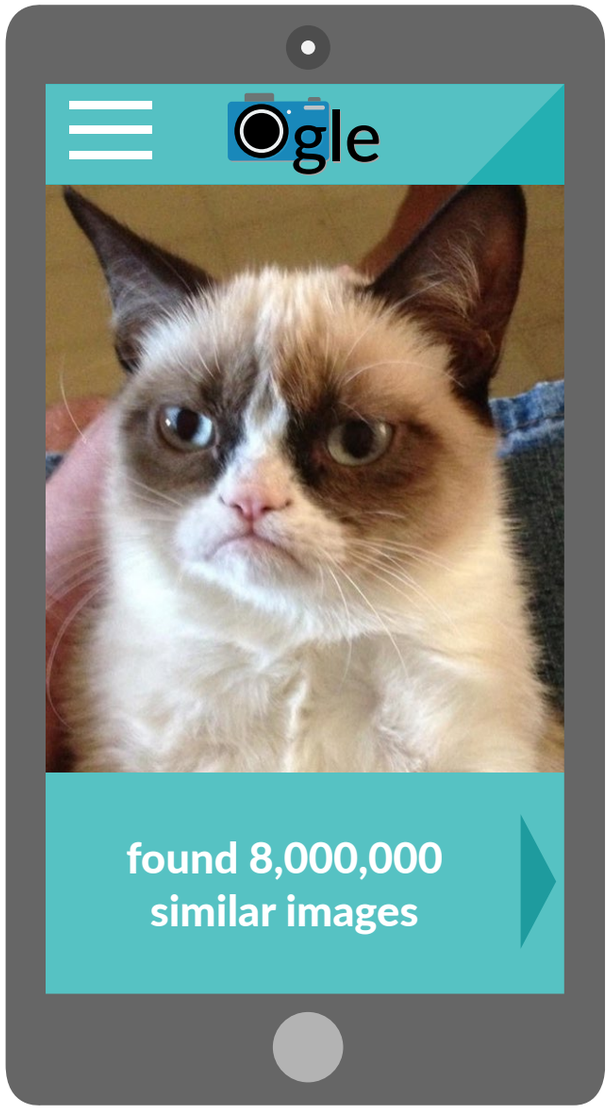


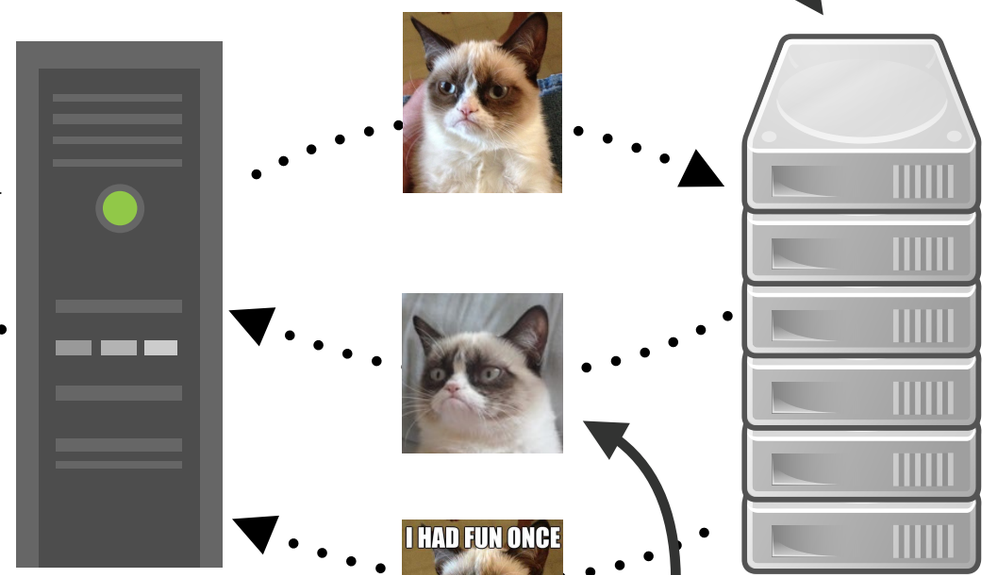
Ogle

Take a picture

Send it to Ogle

Add it to the database





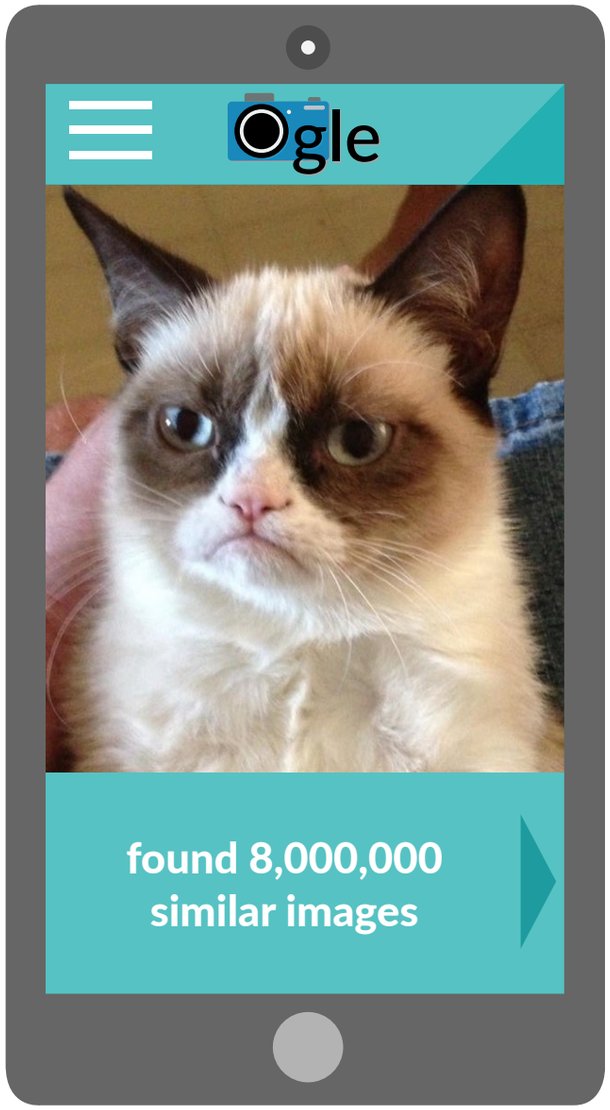
Send results

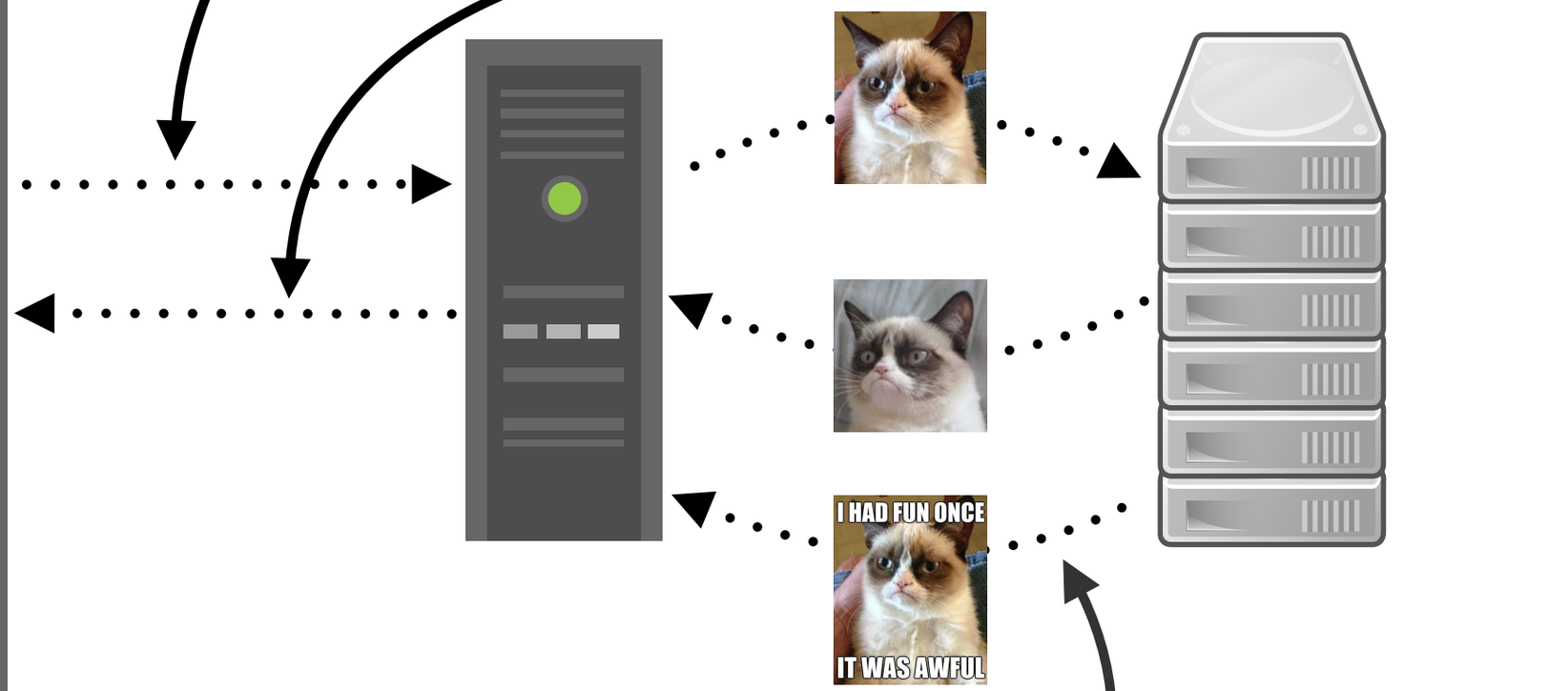
Find similar pictures

Ogle

How long between request and response?

(latency)

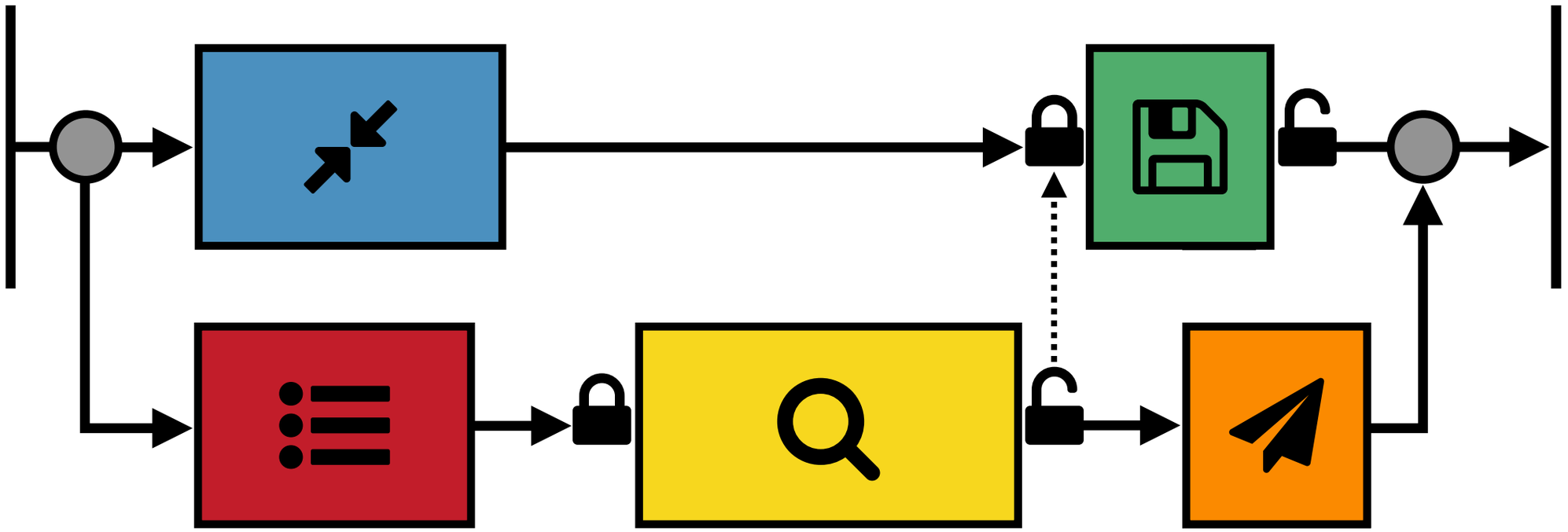




(thruput)

How fast do results come back?

Progress Points



Luke wants to send responses faster.

He marks  as a *progress point*.

Each time this code runs…

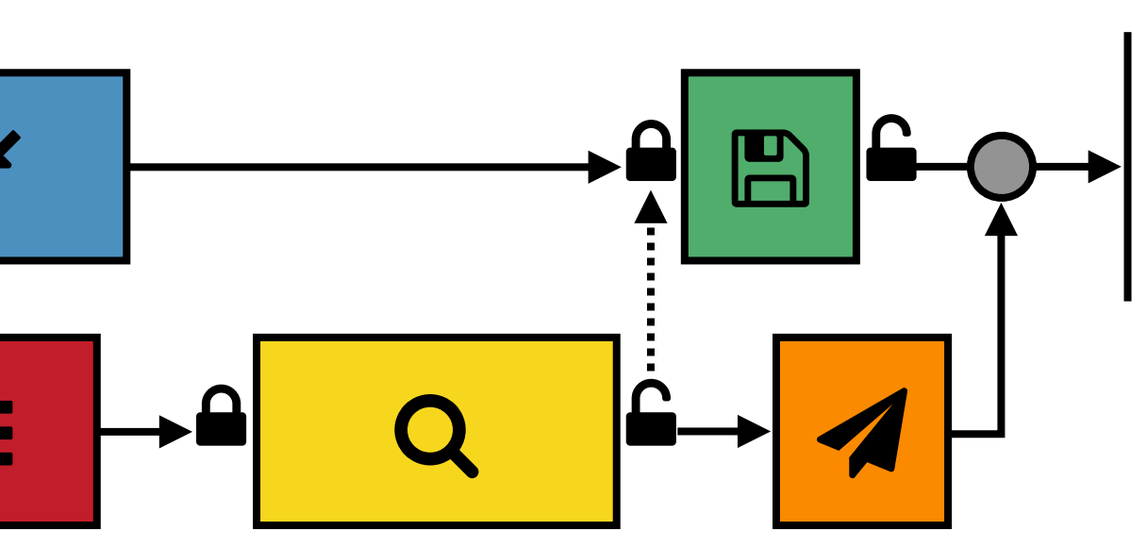
Progress Points

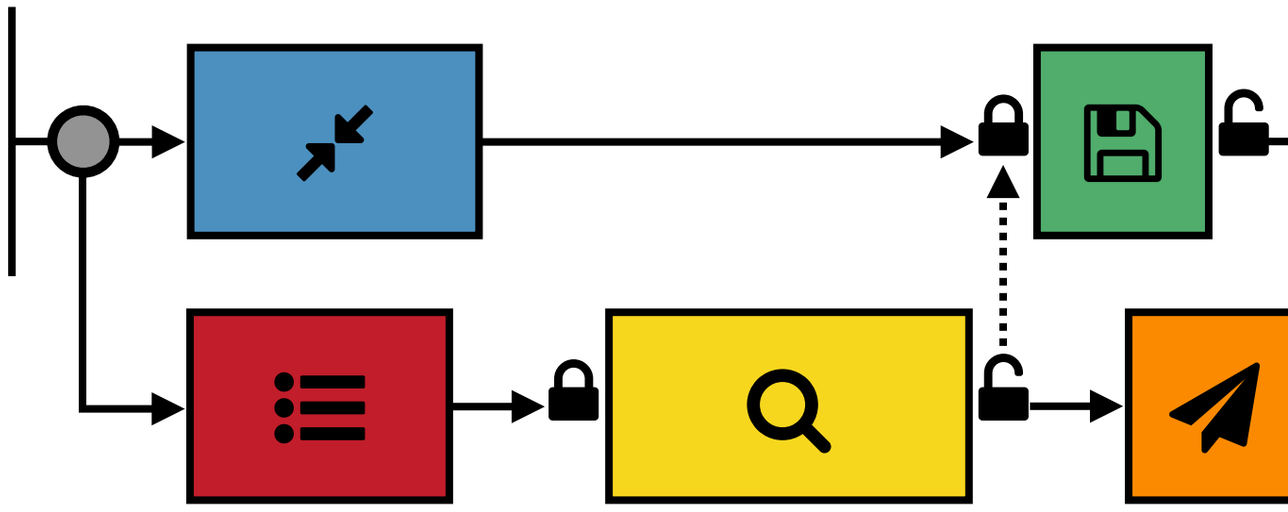
Many requests running for many users.

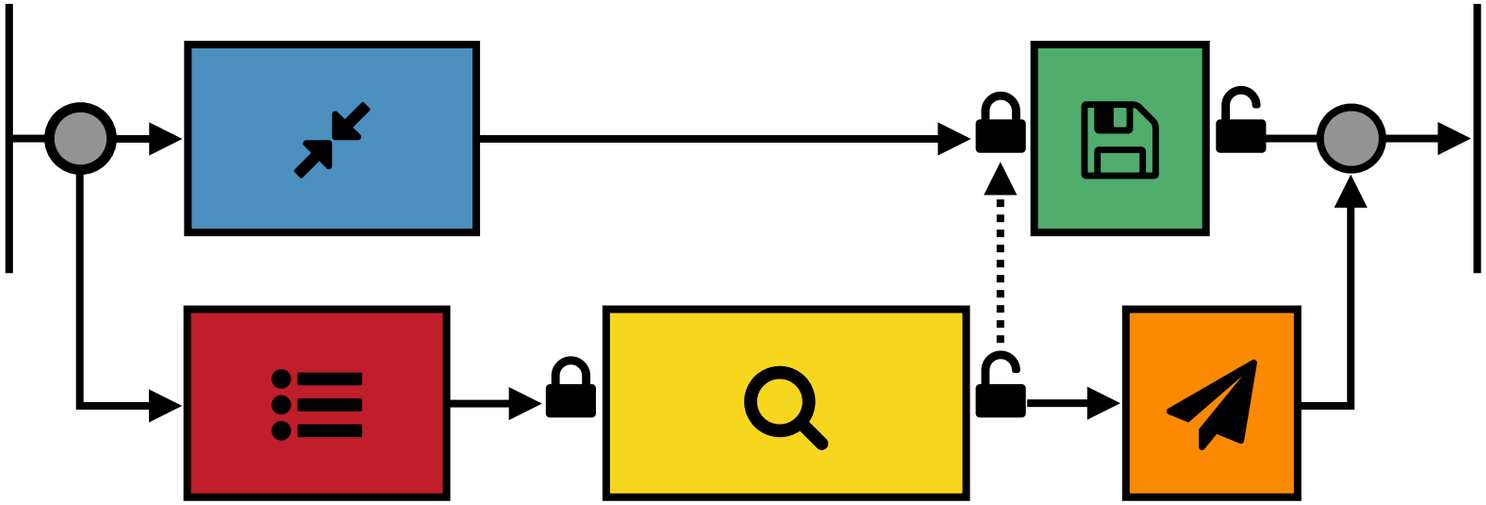
Progress Points

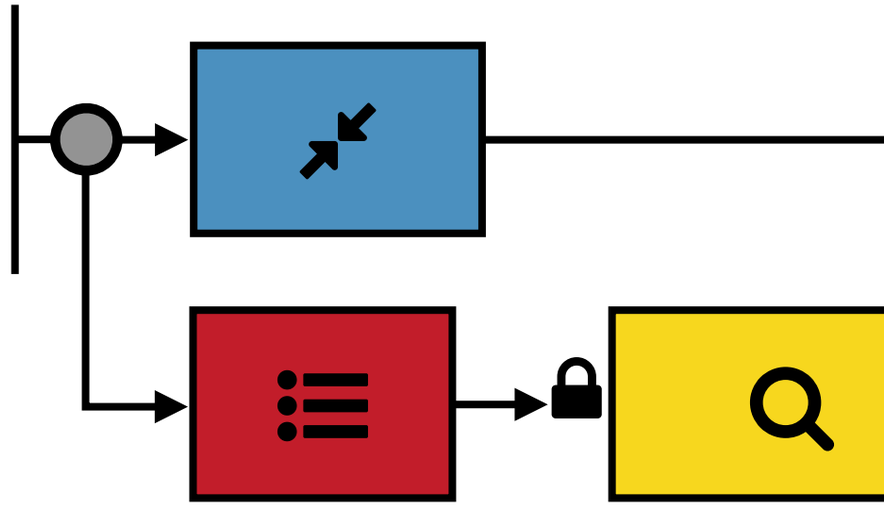
One progress point measures throughput.

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

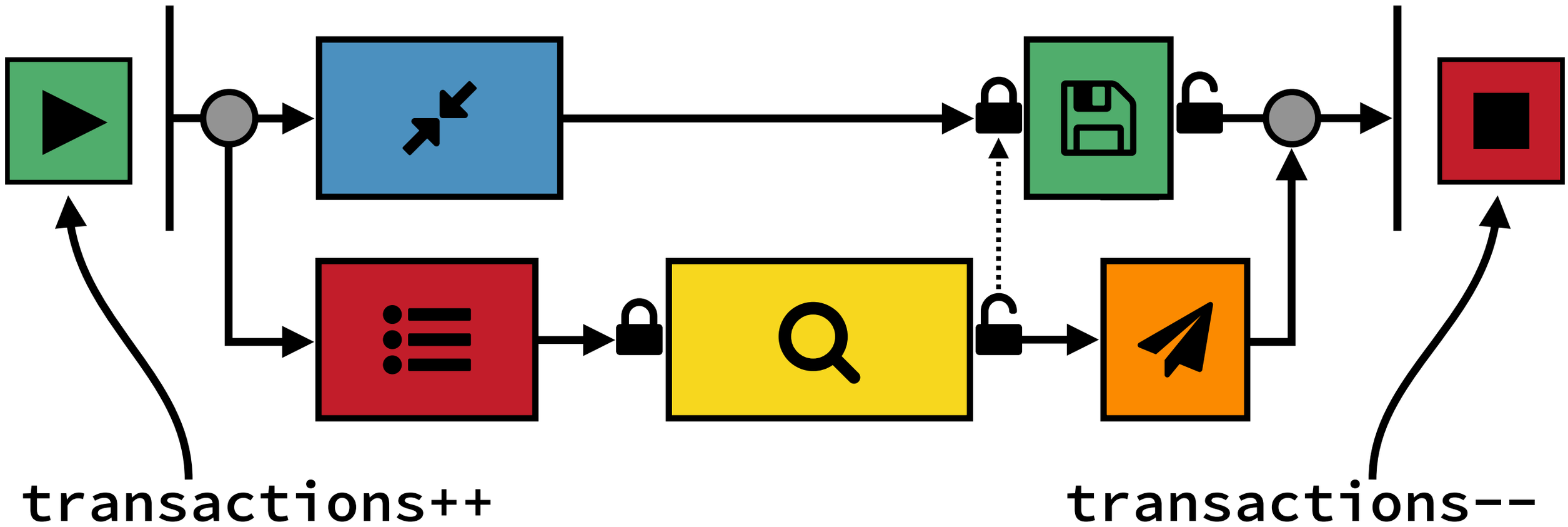








Progress Points



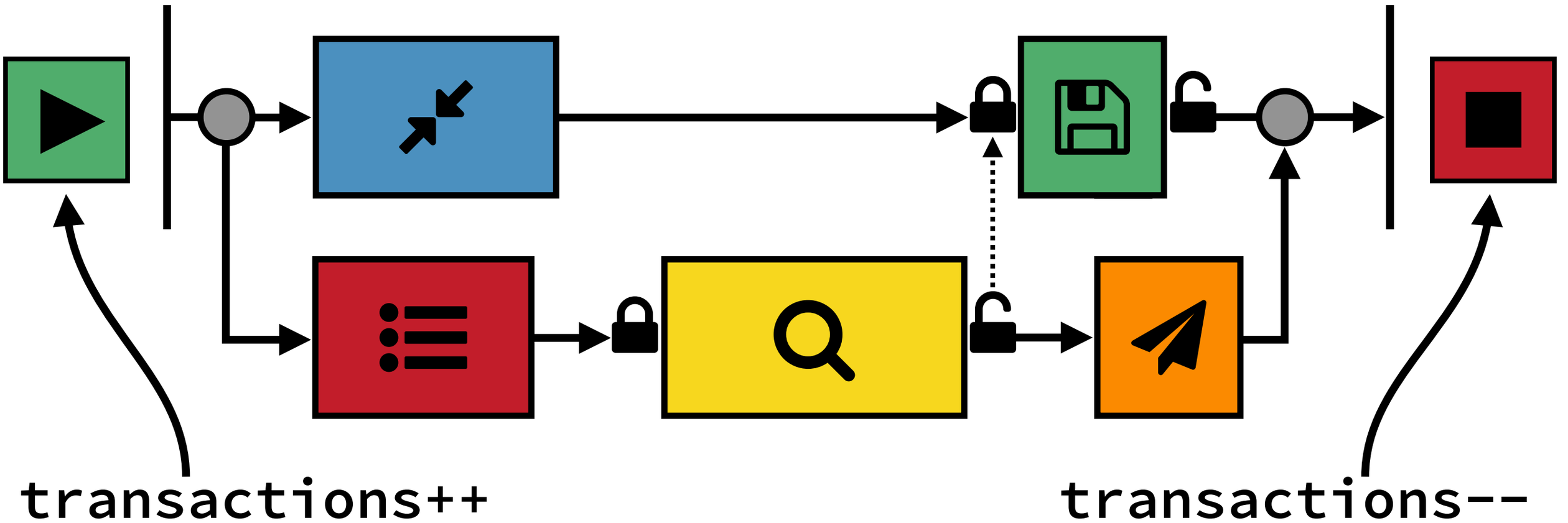
Luke wants to minimize response time.

He adds *latency progress points*.

Progress Points

Little's Law: W= L / λ

latency = `transactions` / throughput



Luke wants to minimize response time.

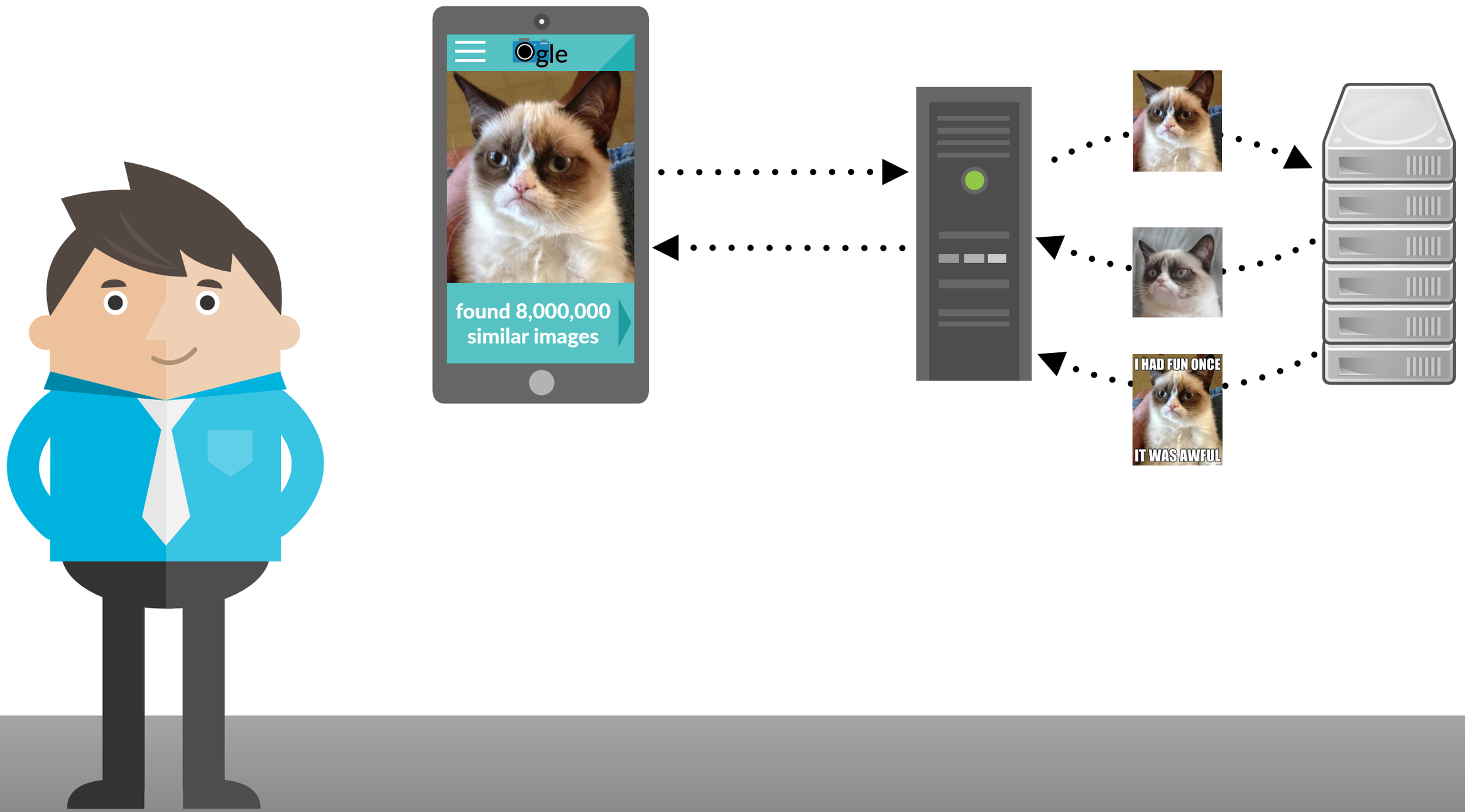
Coz: a Causal Profiler for Linux

(ships with Debian / Ubuntu)

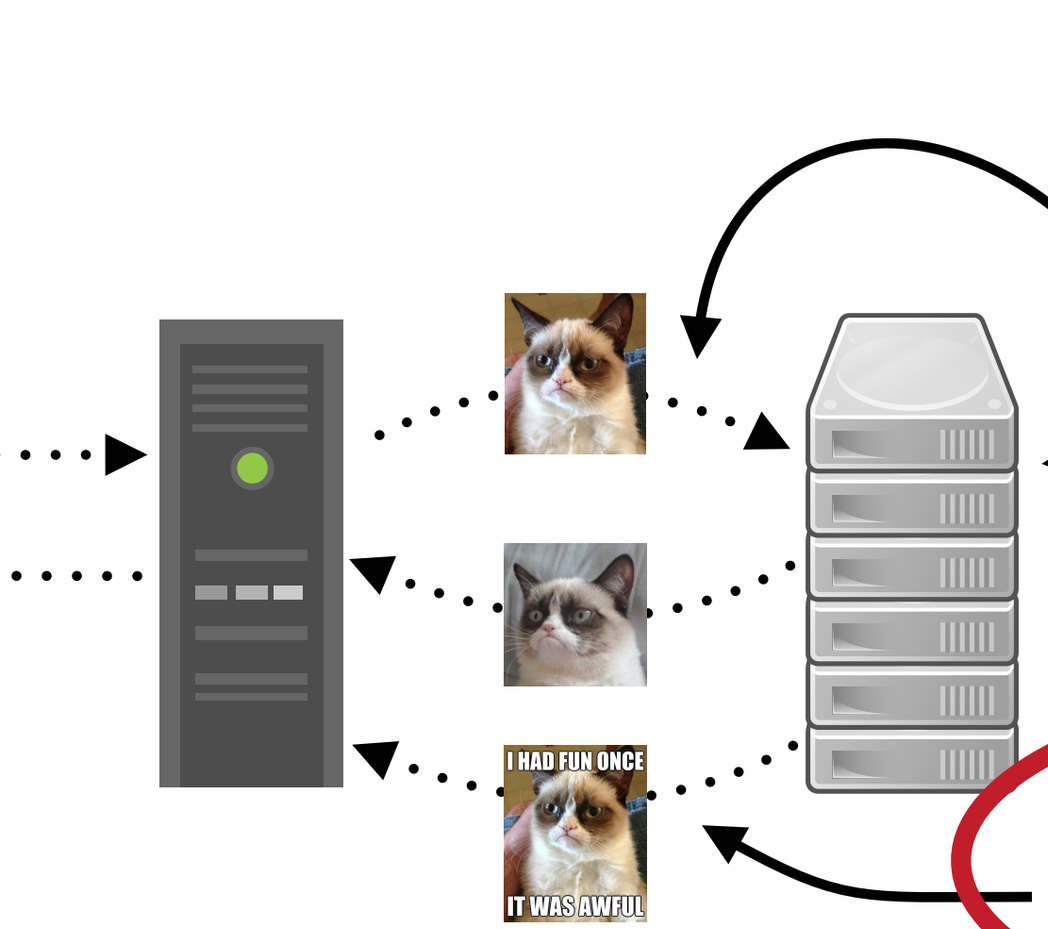
```
> sudo apt install coz-profiler
> coz run --- ./some_program args
```

Random *performance experiments*

Using Causal Profiling on Ogle



Using Causal Profiling on Ogle



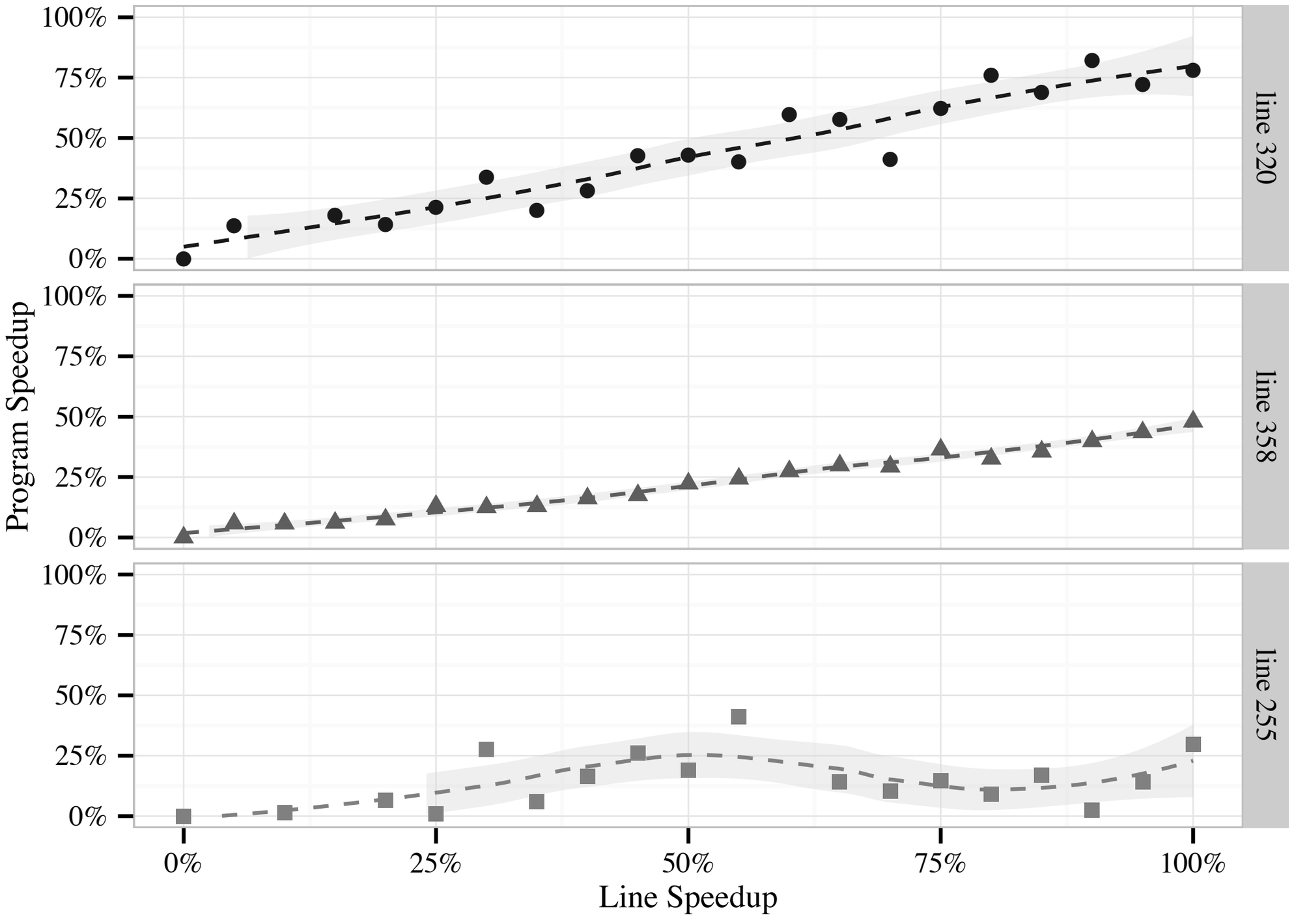
SQLite

dedup
compression

ferret
image comparison

PARSEC
['pär-''sek] A unit of measure

Ferret

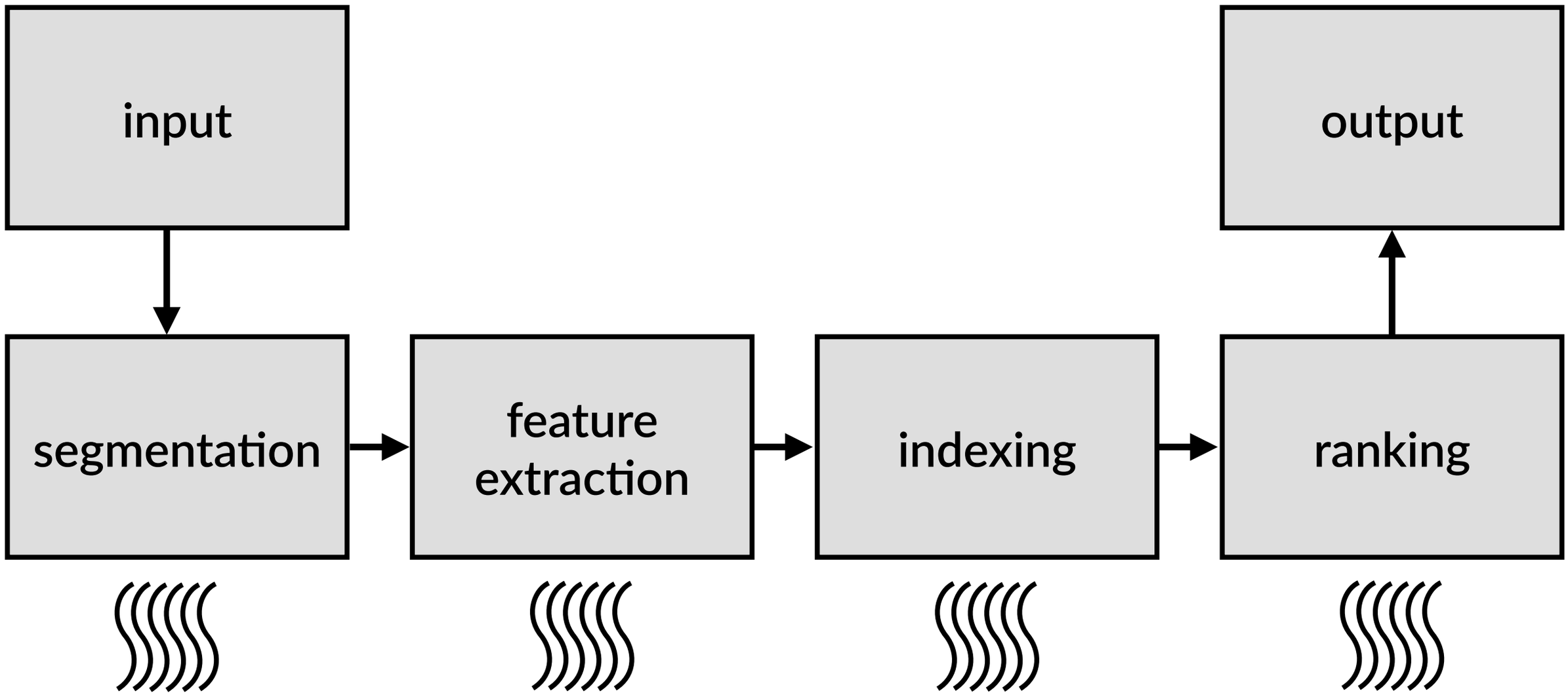


ranking

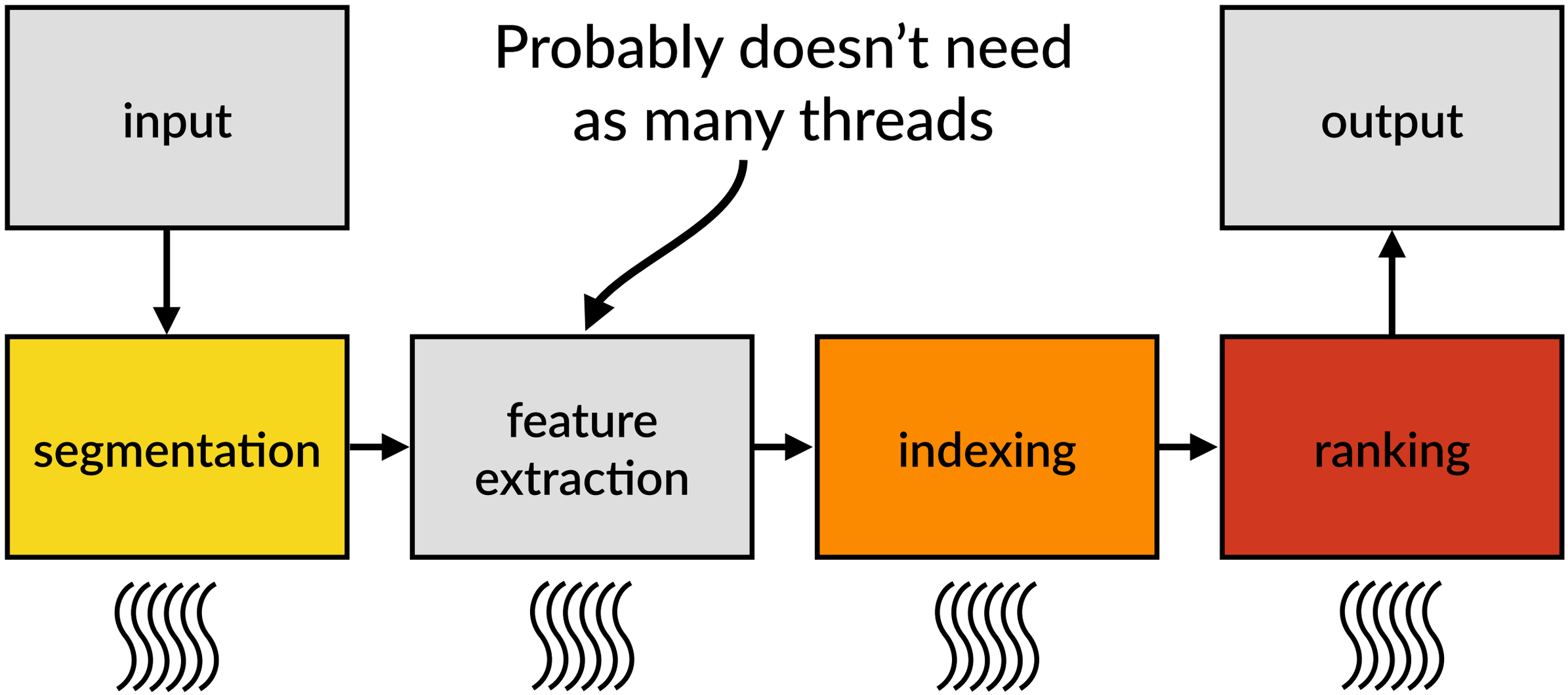
indexing

segmentation

Ferret

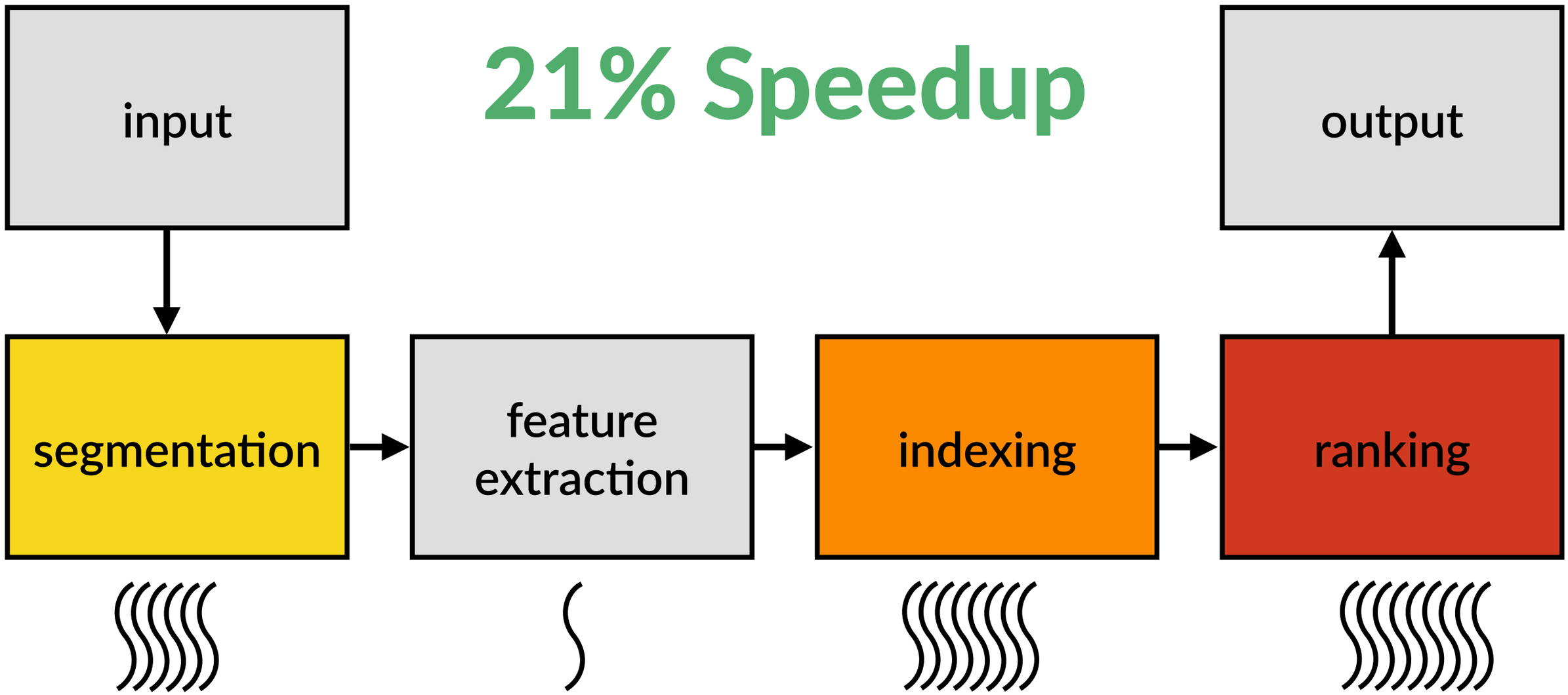

input
output
segmentation
feature extraction
indexing
ranking

Ferret


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

Ferret

21% Speedup

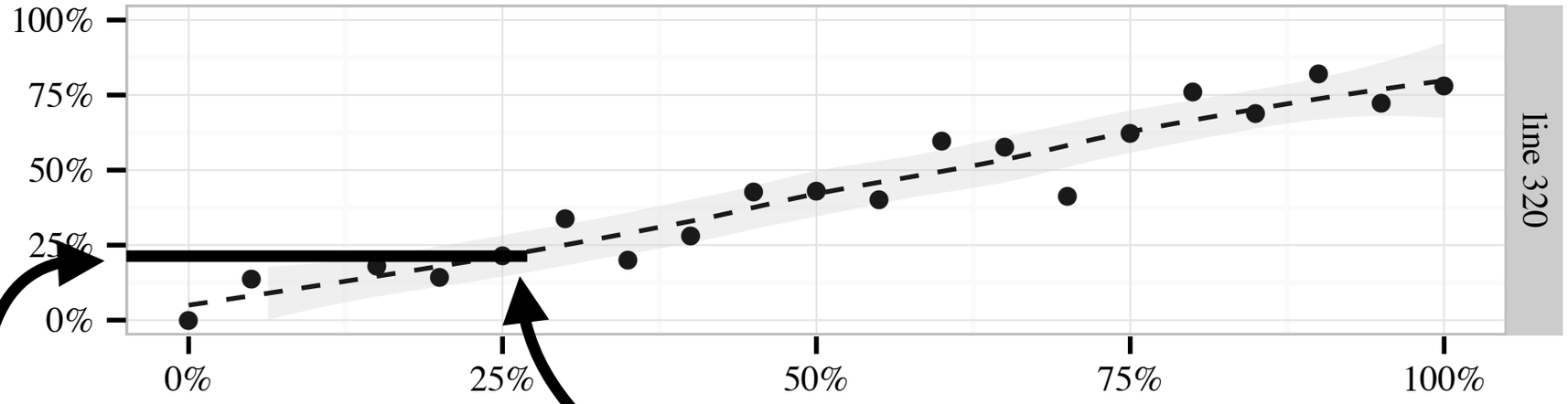


What did Coz predict?

Increased from 16 to 22 threads

27% increase in ranking throughput

What did Coz predict?



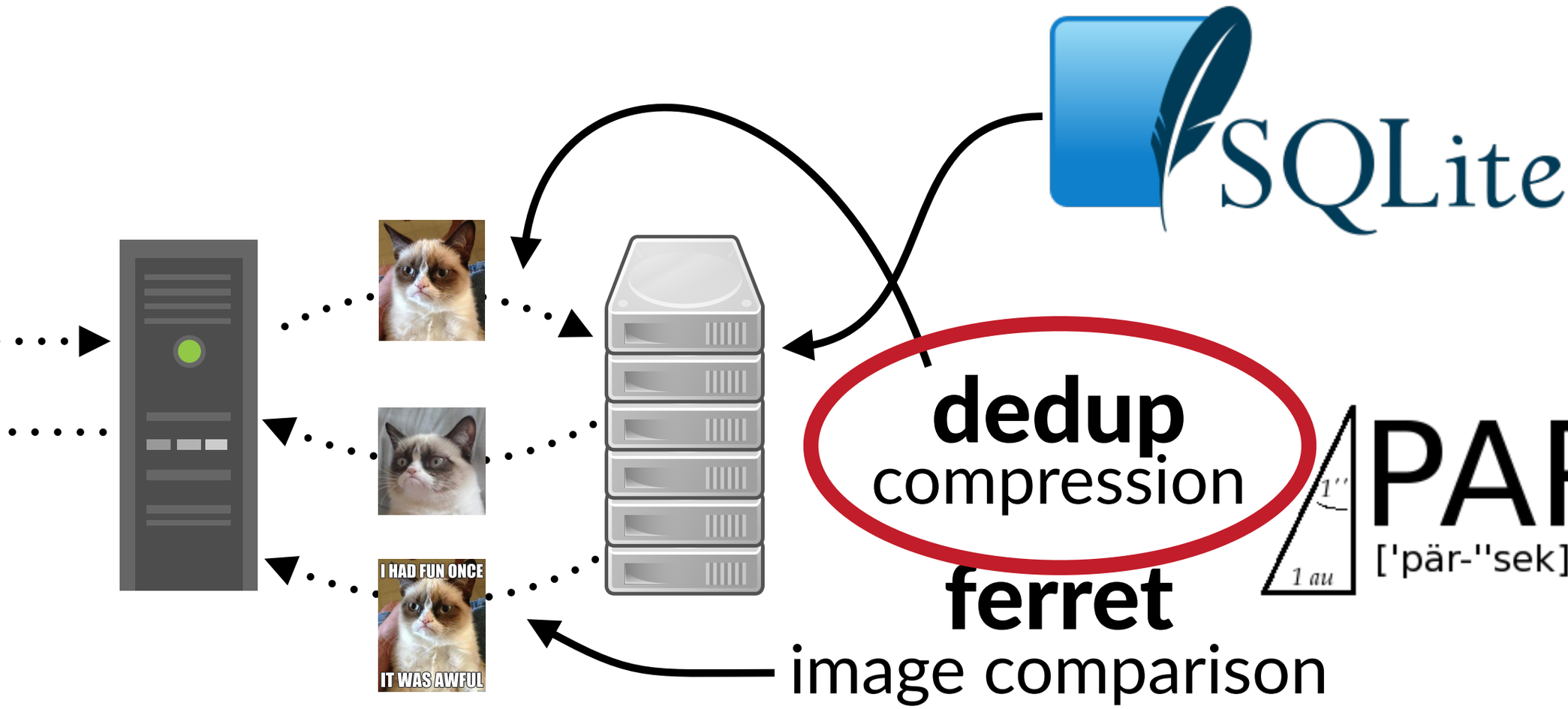
ranking

27% increase in ranking throughput

Coz predicted a 21% improvement

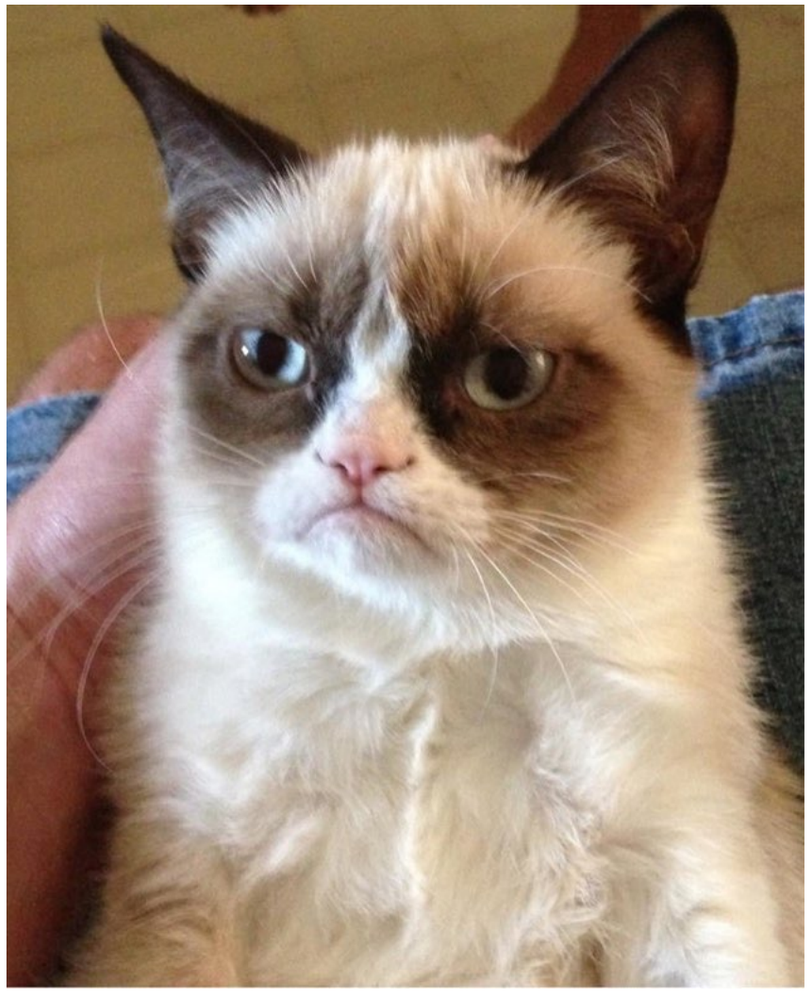
Exactly what we observed

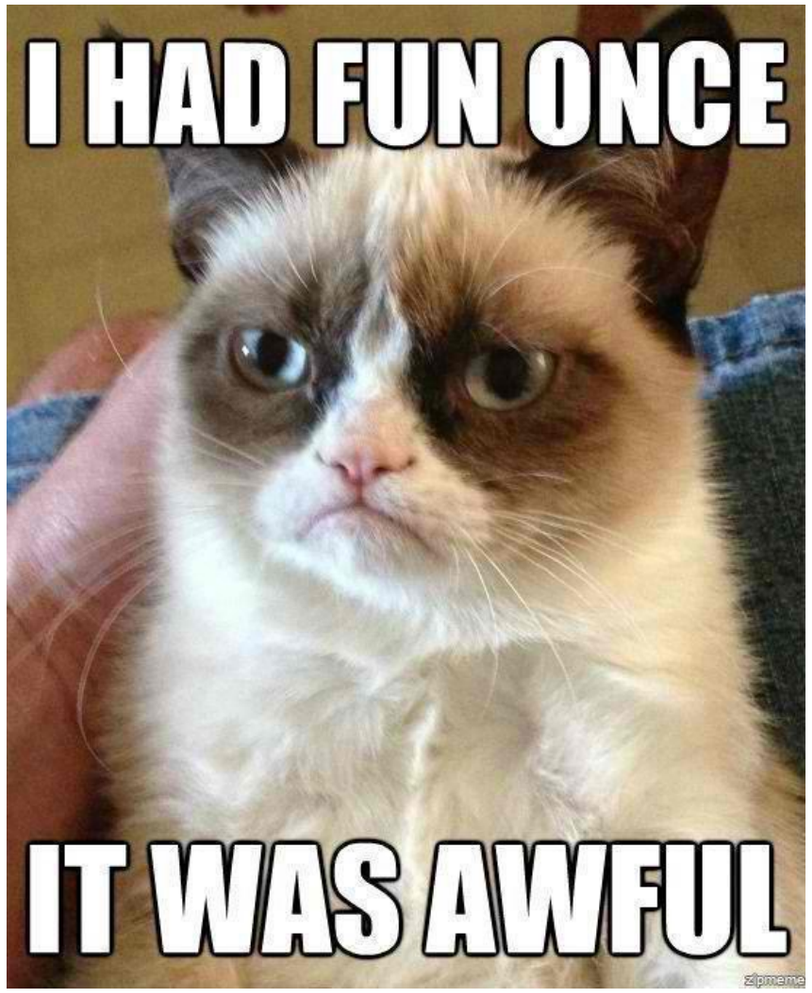
Using Causal Profiling on Ogle



Dedup

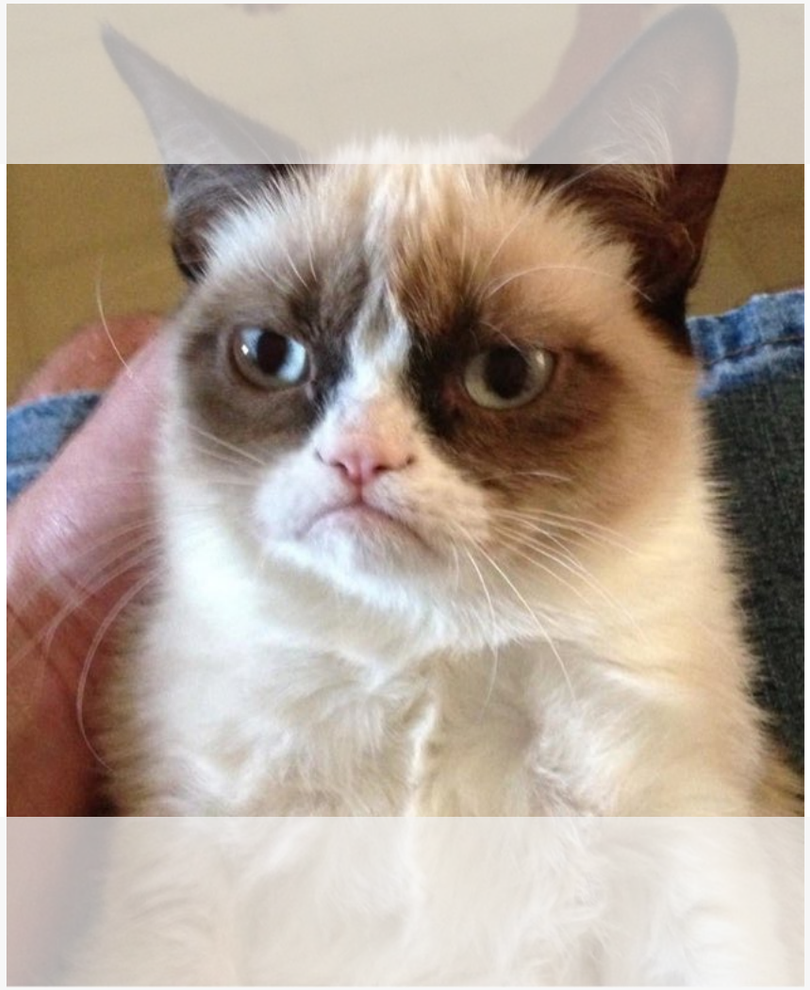
Compression via deduplication





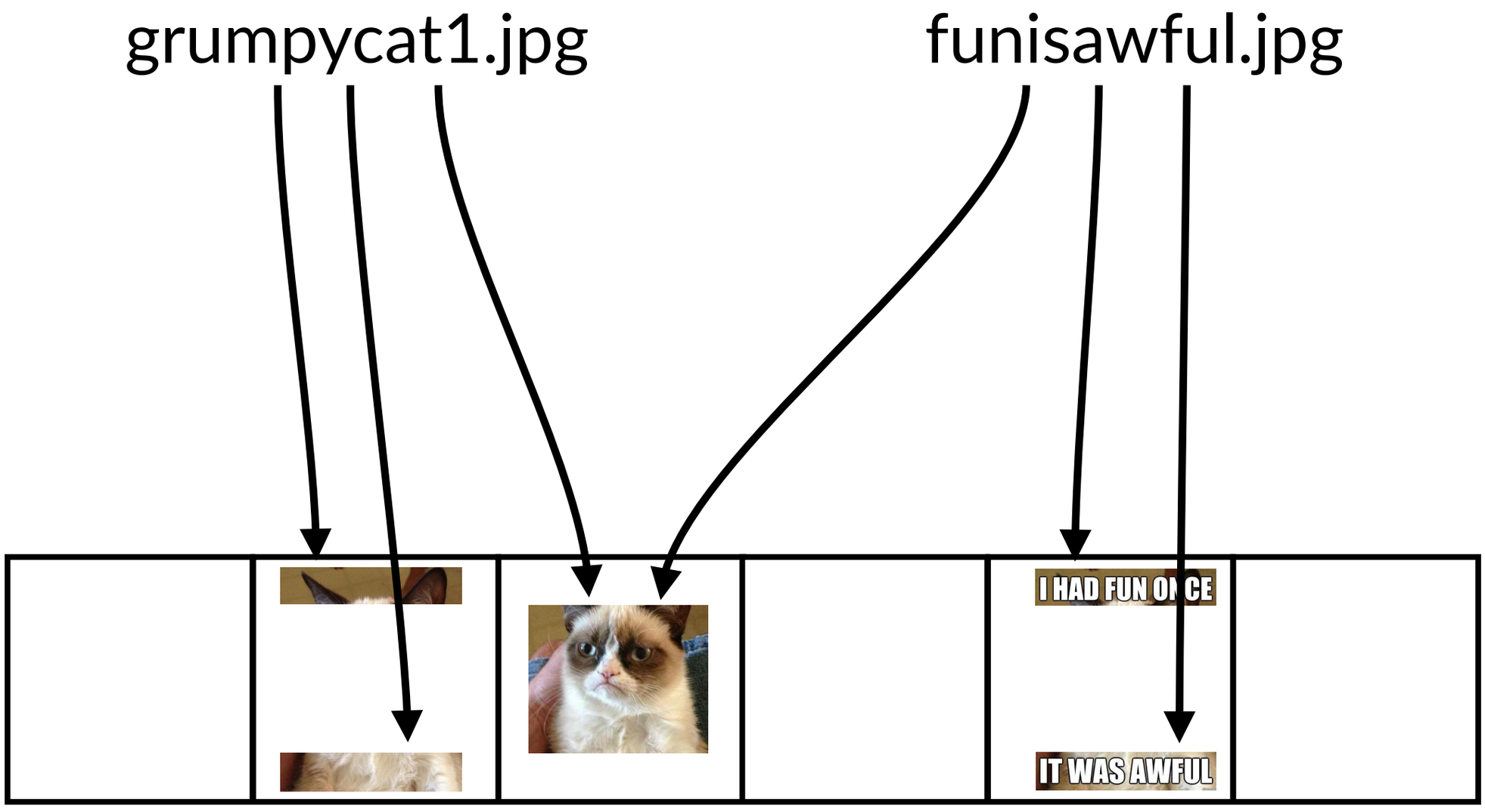
Dedup

Compression via deduplication



Dedup

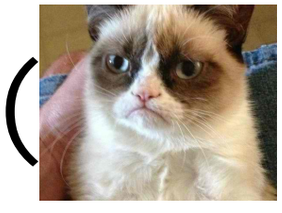
Compression via deduplication



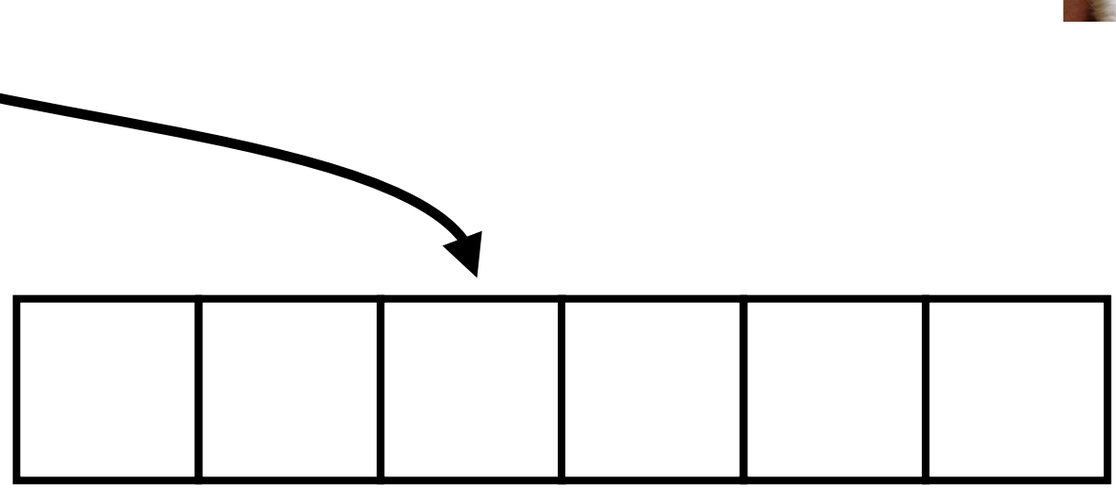
Dedup

Compression via deduplication

```
i = hash_function(
```



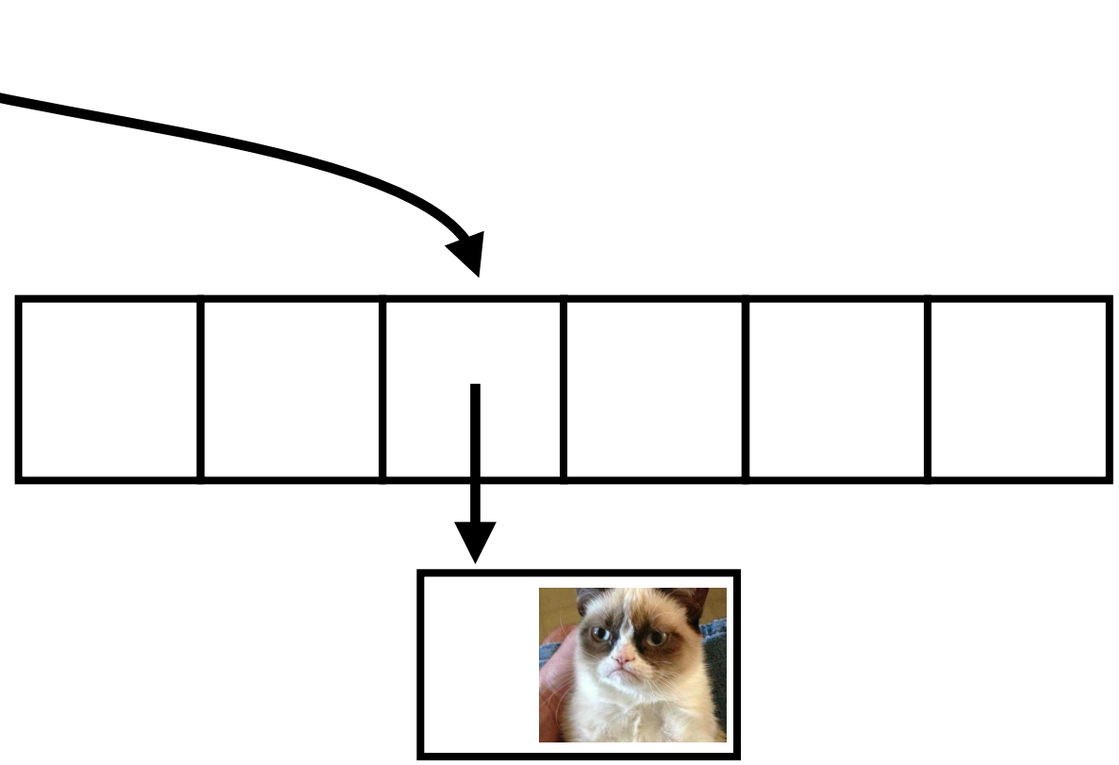
)



Dedup

Compression via deduplication

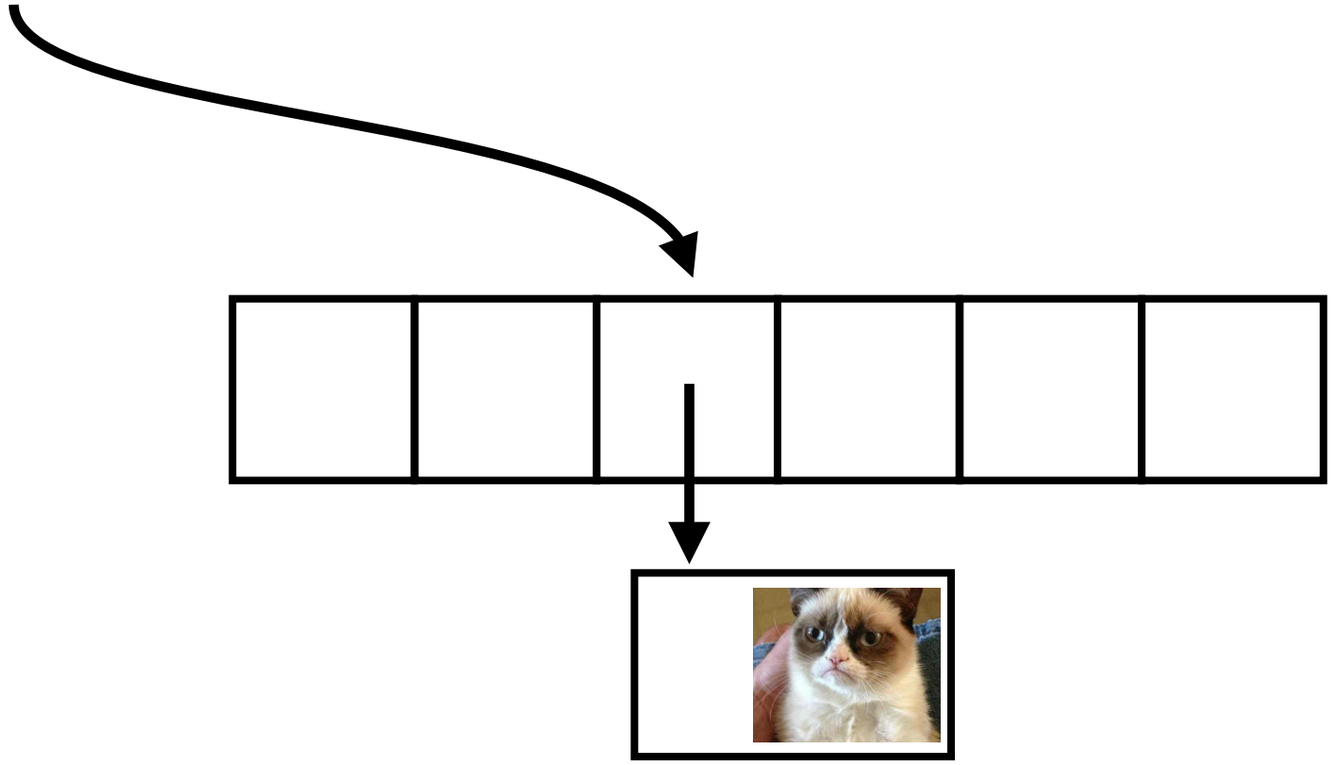
```
i = hash_function(    )
```



Dedup

Compression via deduplication

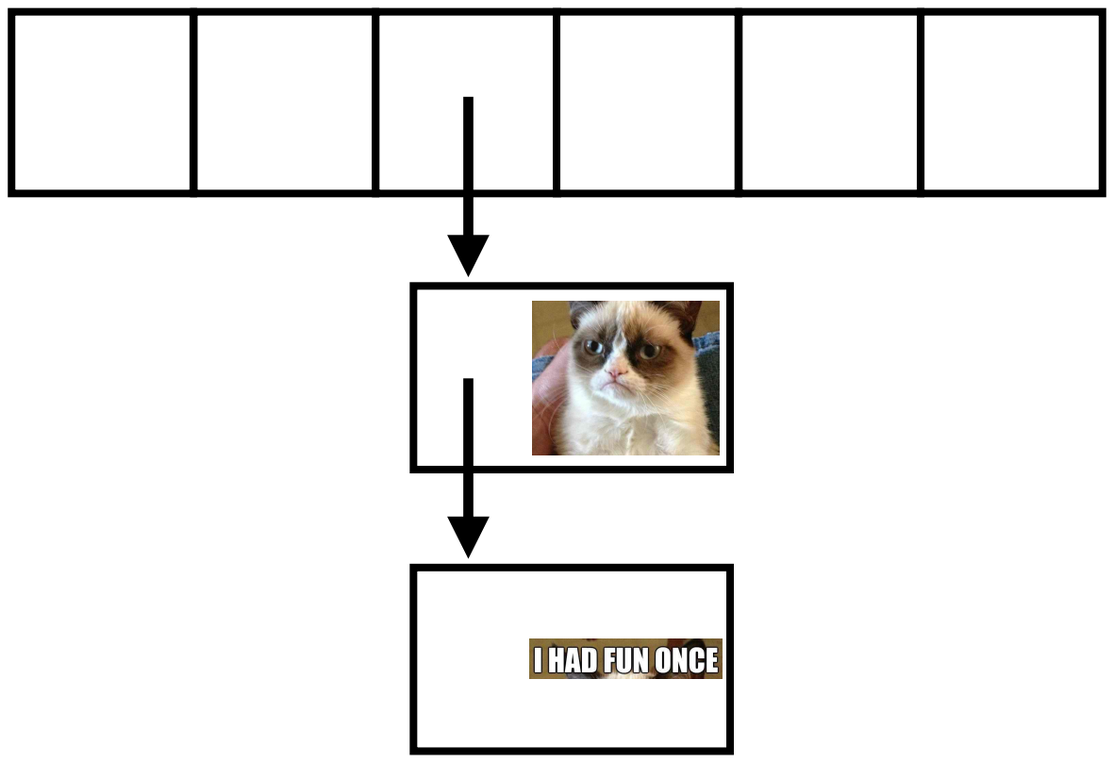
```
i = hash_function( I HAD FUN ONCE )
```



Dedup

Compression via deduplication

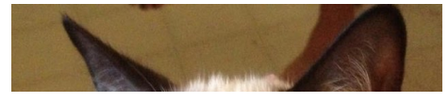
```
i = hash_function(          )
```



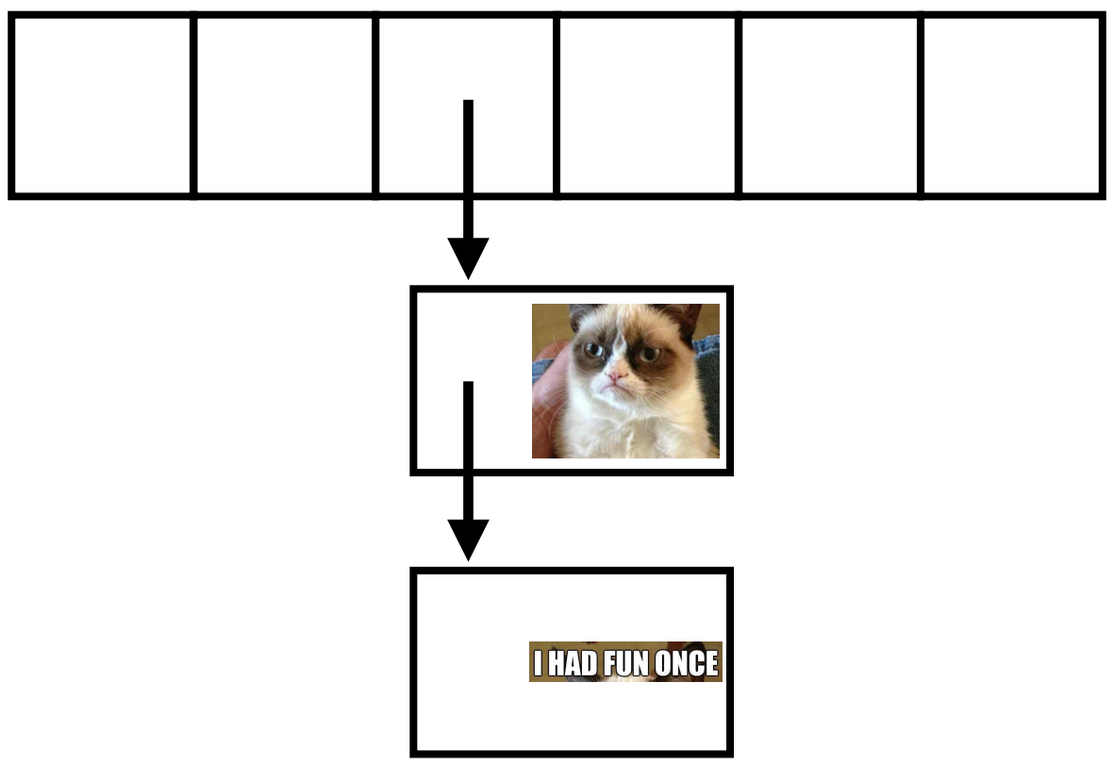
Dedup

Compression via deduplication

```
i = hash_function(
```



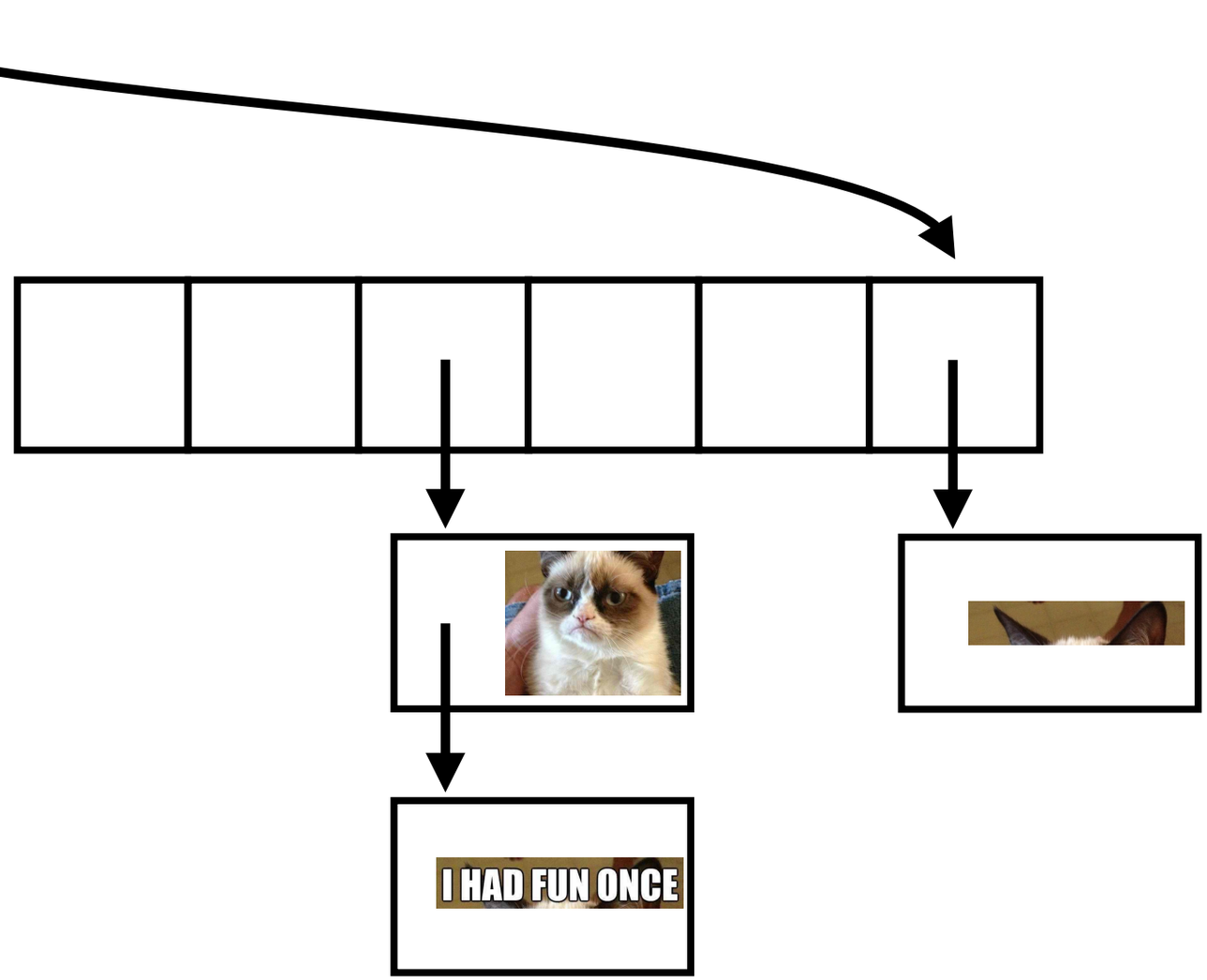
```
)
```



Dedup

Compression via deduplication

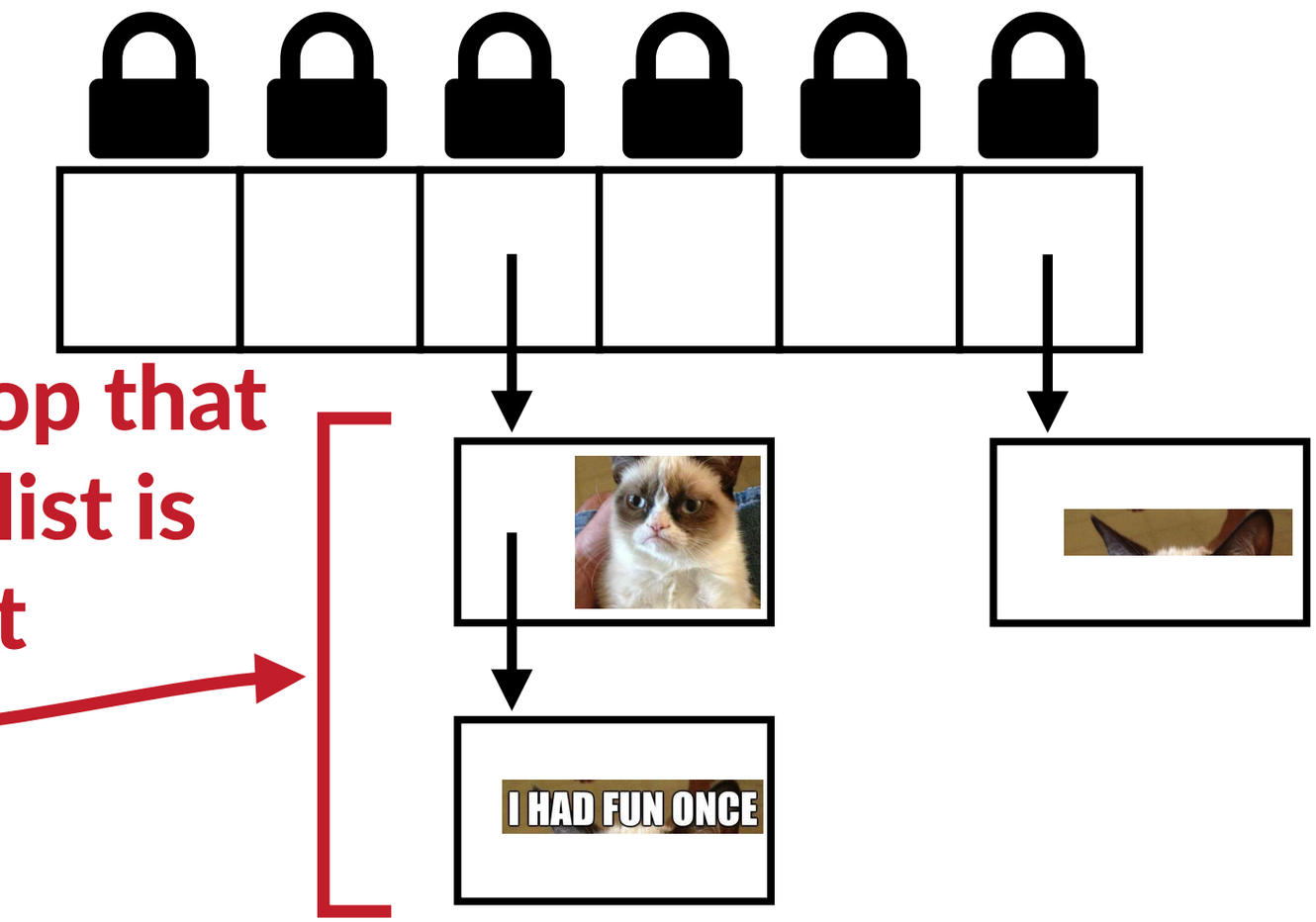
```
i = hash_function(          )
```



Dedup

Compression via deduplication

Hash table is accessed concurrently by many threads



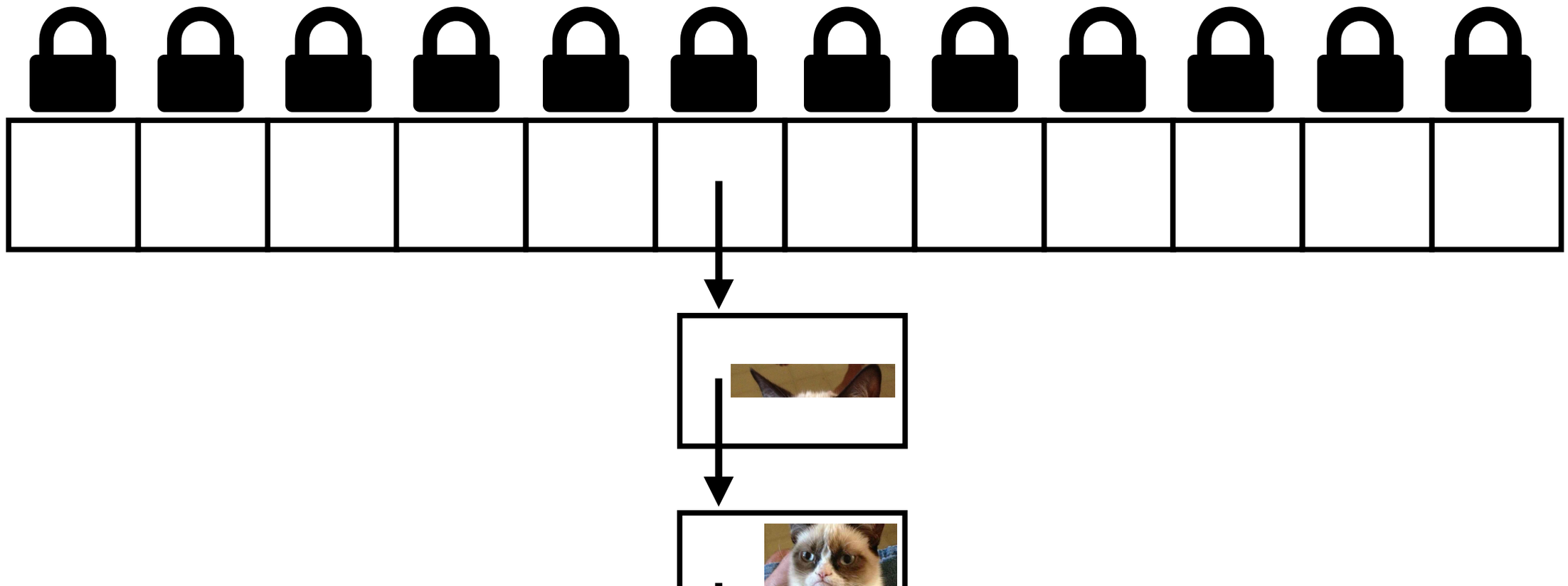
Coz says the loop that accesses this list is important

Dedup

Compression via deduplication

More hash buckets should lead to fewer collisions

No performance improvement

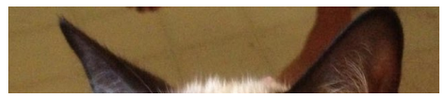


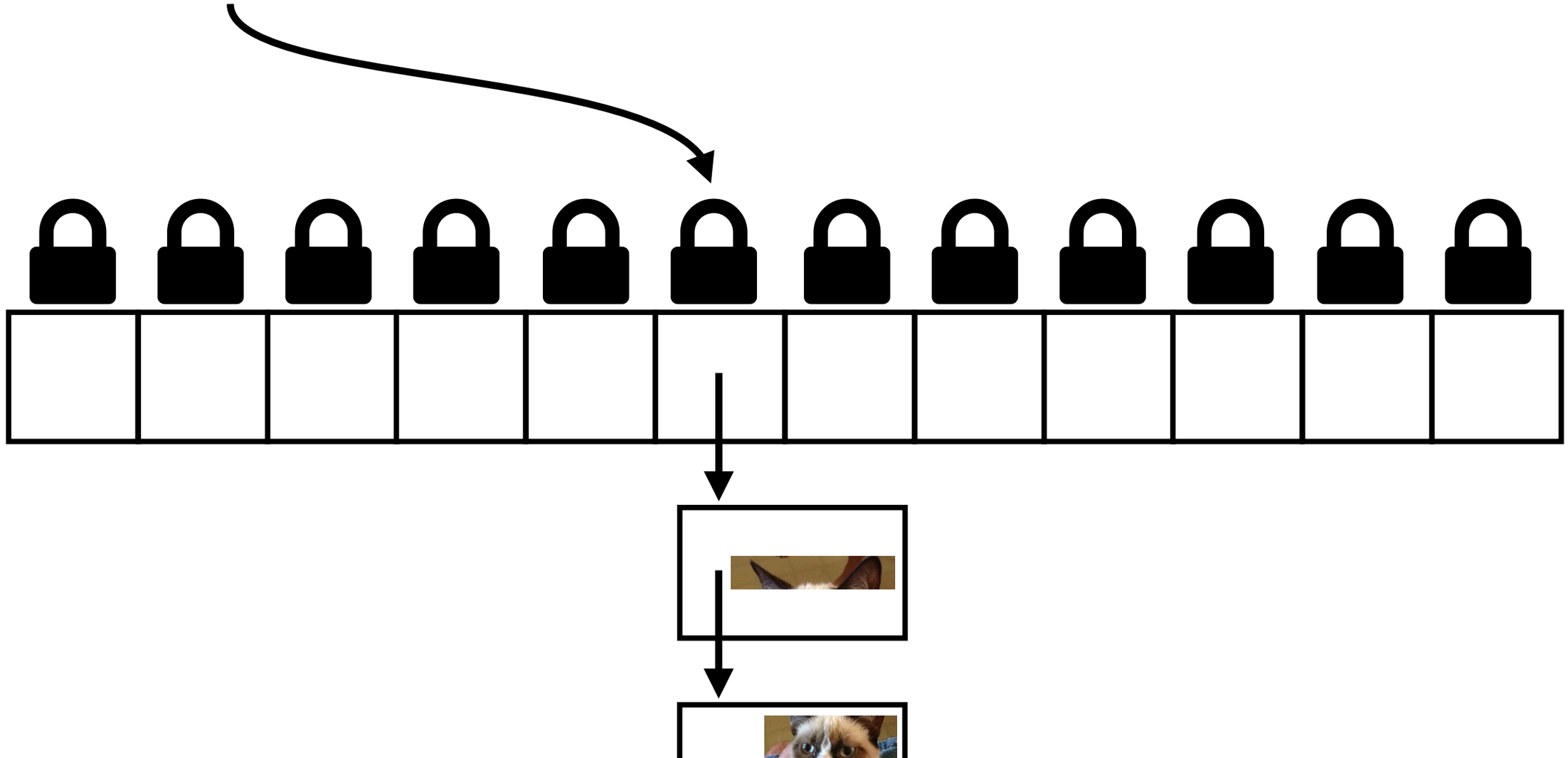
Dedup

Compression via deduplication

What else could be causing collisions?

```
i = hash_function(
```

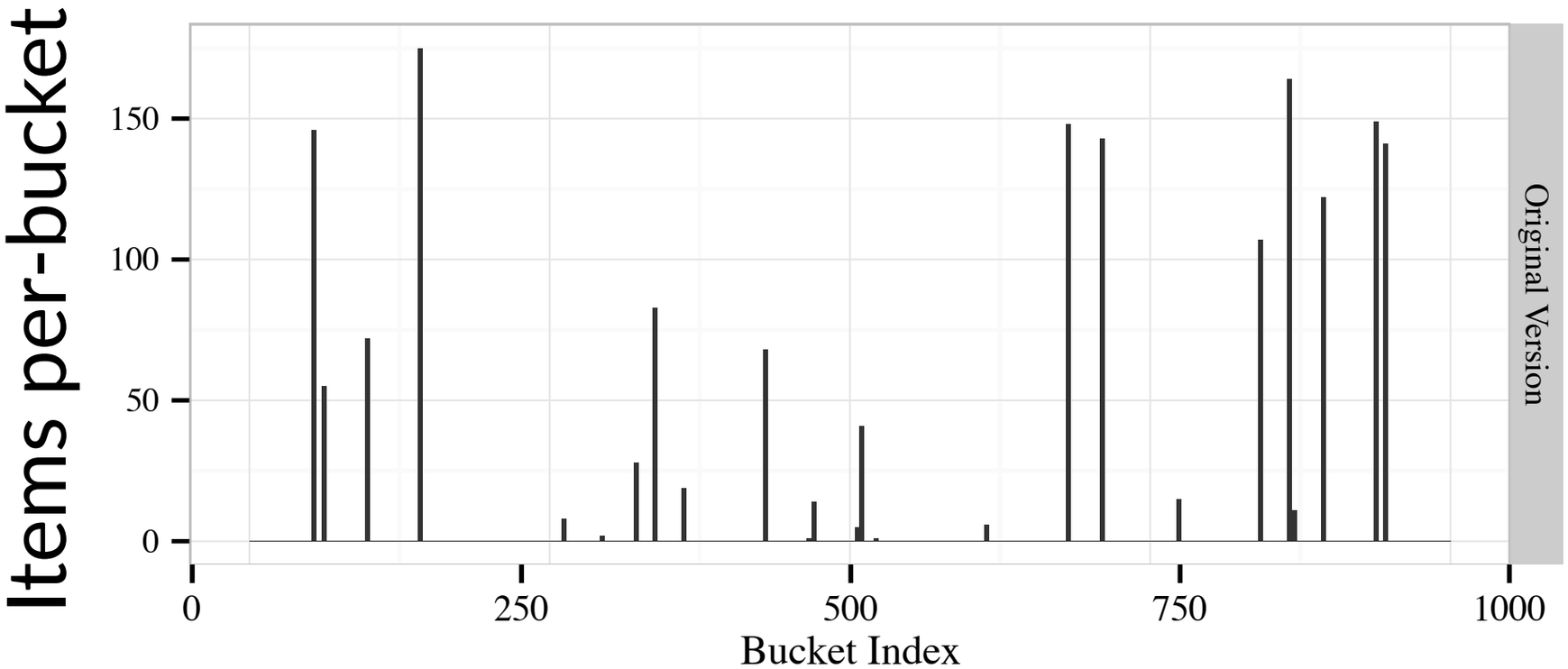




Dedup

Compression via deduplication

Horrible hash function!

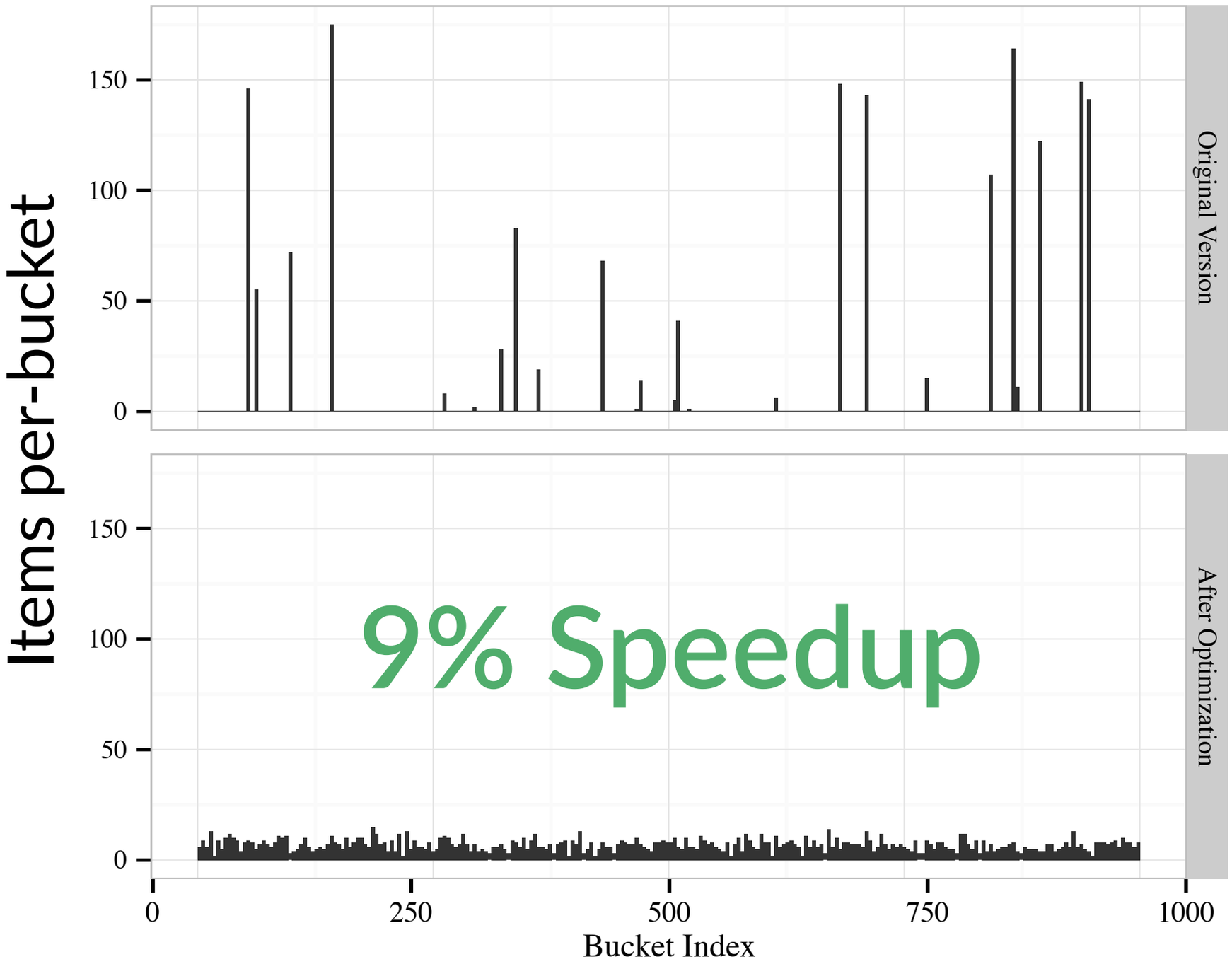


Bin Utilization

2.3%

Dedup

Compression via deduplication



Bin Utilization

2.3%

82%

Dedup

Compression via deduplication

What did Coz predict?

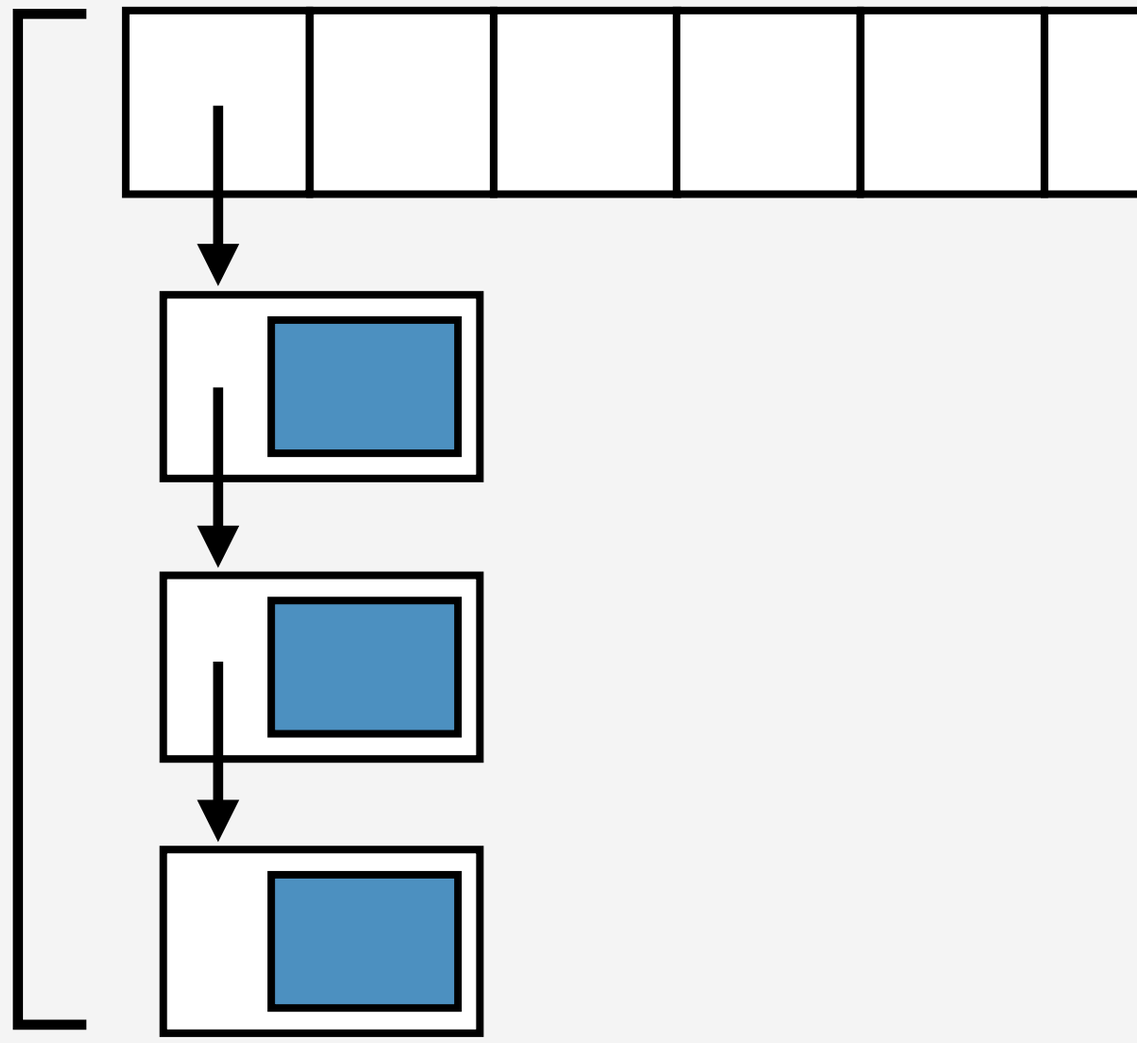
Blocks per-bucket

Before: 76.7

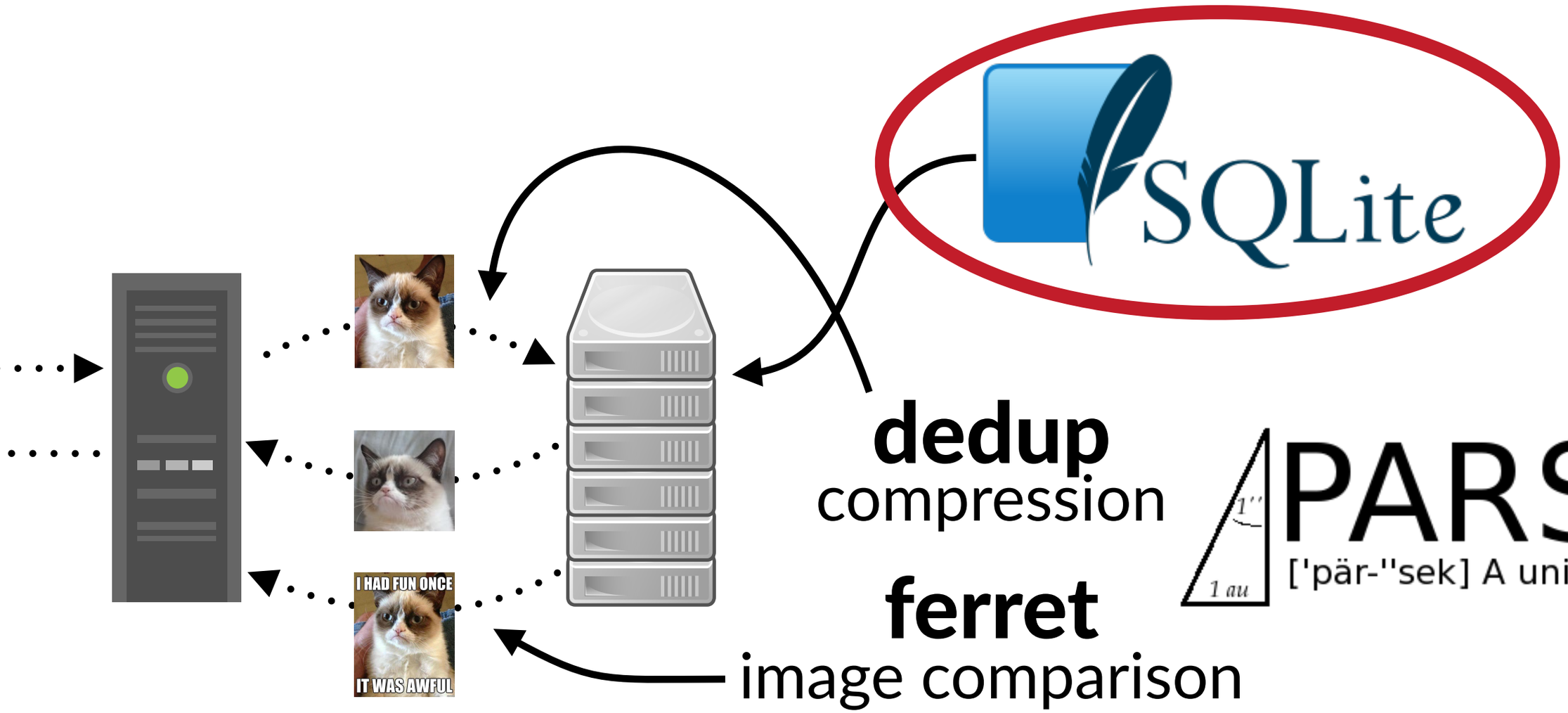
After: 2.09

96% traversal speedup

9% predicted speedup, exactly what we observed



Using Causal Profiling on Ogle

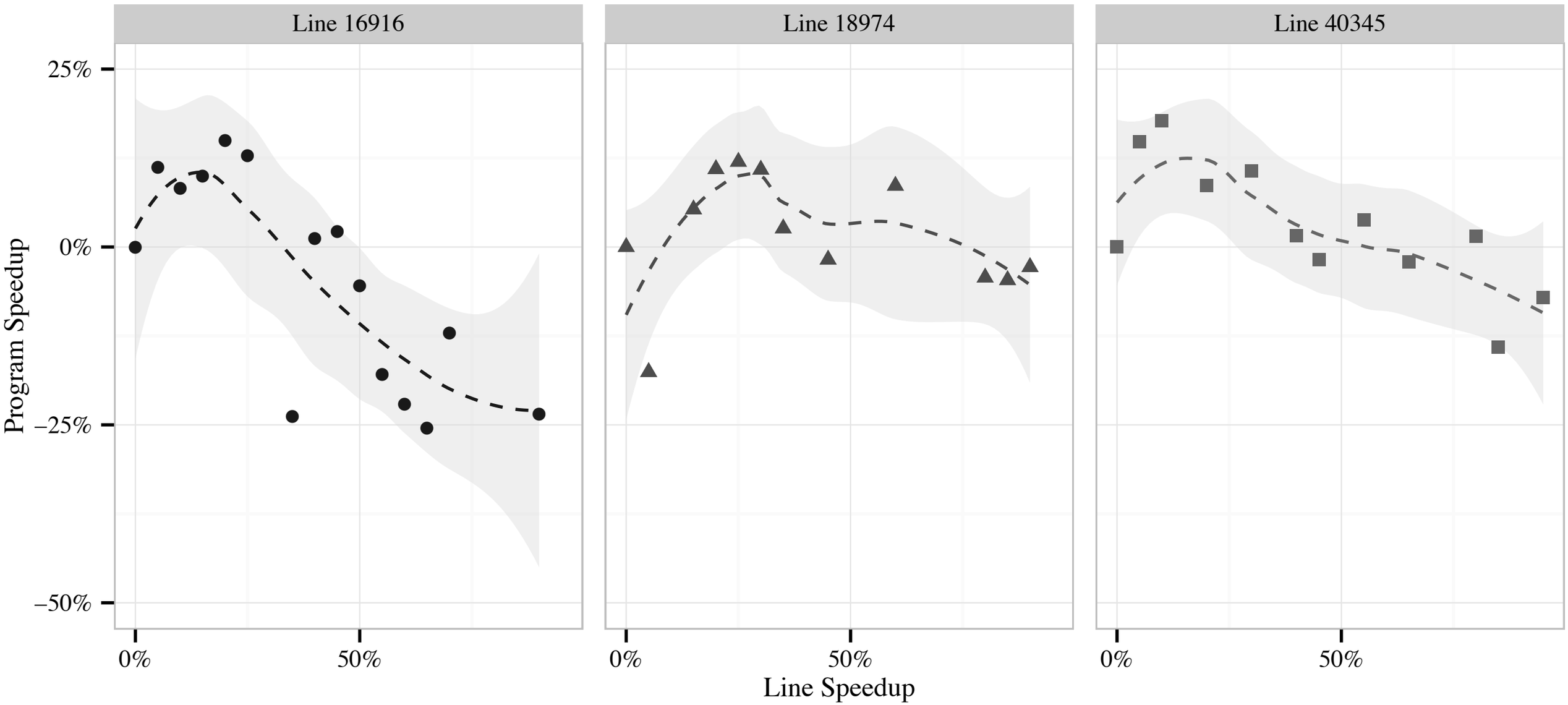


dedup
compression

ferret
image comparison

PARSEC
['pär-''sek] A unit of measure

Simple SQL Database



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

Indirect Call

Cheap, but almost the same cost as `pthread_mutex_unlock`

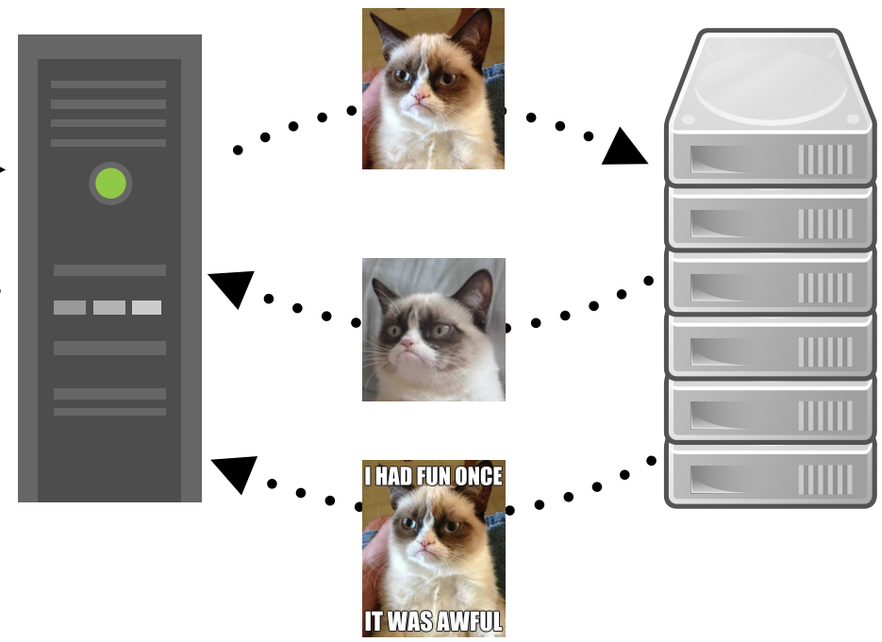
Simple SQL Database

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

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

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

Coz highlights these lines

Using Causal Profiling on Ogle

SQLite 25%



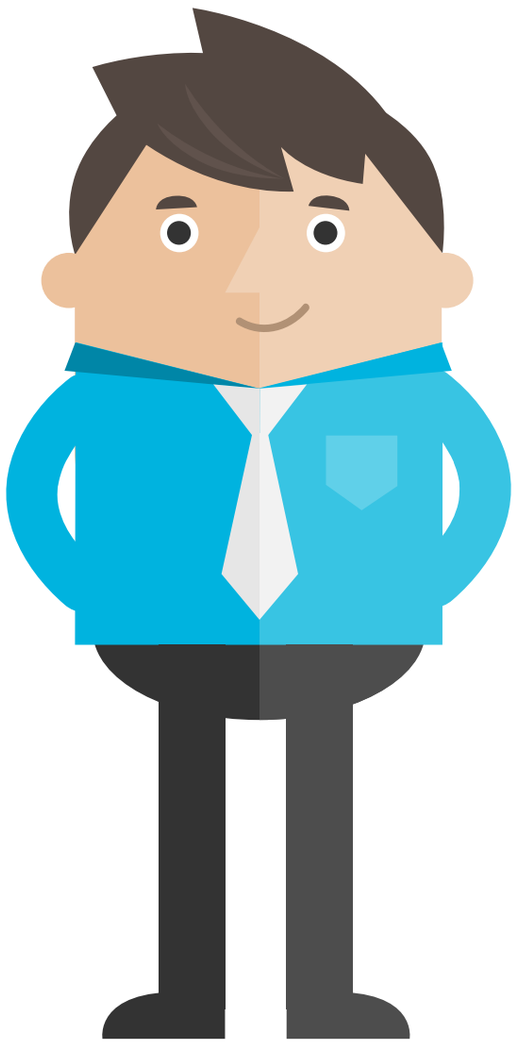
dedup
compression 9%

ferret
image comparison 21%

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

Sound Performance Analysis



Effective Performance Profiling

USE THE COZ

```
% sudo apt install coz-profiler
```