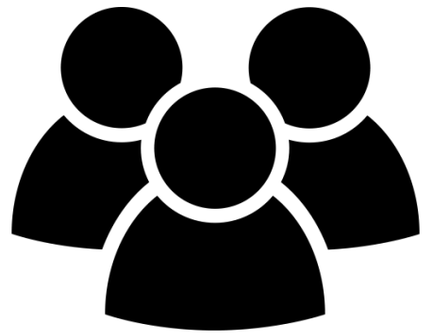
Failure Sketching: A Technique for Automated Root Cause Diagnosis of In-Production Failures

Baris Kasikci, Benjamin Schubert, Cristiano Pereira,
Gilles Pokam, George Candea

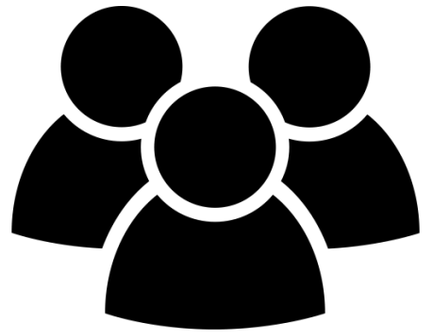
Debugging In-Production Software Failures Today

Debugging In-Production Software Failures Today



Debugging In-Production Software Failures Today

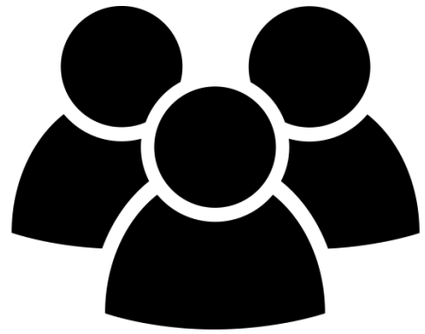
```
#0  0x00007f51abae820b in raise (sig=11) at ../nptl/
sysdeps/unix/sysv/linux/pt-raise.c:37
#1  0x0000000000042d289 in ap_buffered_log_writer
(r=0x7f51a40053d0, handle=0x20eeba0,
strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14,
len=82) at mod_log_config.c:1368
#2  0x0000000000042b10d in config_log_transaction
(r=0x7f51a40053d0, cls=0x20b9d50,
default_format=0x20ee370) at mod_log_config.c:930
#3  0x0000000000042aad6 in multi_log_transaction
(r=0x7f51a40053d0) at mod_log_config.c:950
#4  0x0000000000046cb2d in ap_run_log_transaction
(r=0x7f51a40053d0) at protocol.c:1563
#5  0x0000000000436e81 in ap_process_request
(r=0x7f51a40053d0) at http_request.c:312
#6  0x0000000000042e9da in ap_process_http_connection
(c=0x7f519c000b68) at http_core.c:293
#7  0x0000000000465cdd in ap_run_process_connection
(c=0x7f519c000b68) at connection.c:85
#8  0x00000000004661f5 in ap_process_connection
(c=0x7f519c000b68, csd=0x7f519c000a20) at
connection.c:211
#9  0x0000000000451ba0 in process_socket
(p=0x7f519c0009b8, sock=0x7f519c000a20,
my_child_num=0, my_thread_num=0,
bucket_alloc=0x7f51a4001348) at worker.c:632
#10 0x0000000000451221 in worker_thread
(thd=0x210fa90, dummy=0x7f51a40008c0) at worker.c:946
#11 0x00007f51ac87c555 in dummy_worker
(opaque=0x210fa90) at thread.c:127
#12 0x00007f51abae0182 in start_thread
(arg=0x7f51aa8ef700) at pthread_create.c:312
#13 0x00007f51ab80d47d in clone () at ../sysdeps/
unix/sysv/linux/x86_64/clone.S:111
```



Debugging In-Production Software Failures Today

Understand root cause

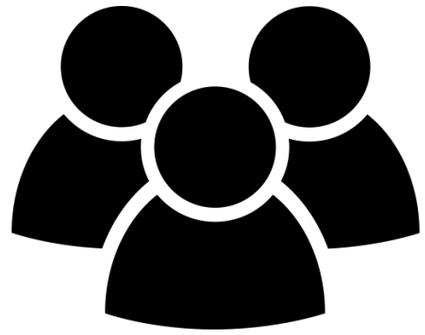
```
#0  0x00007f51abae820b in raise (sig=11) at ../nptl/
sysdeps/unix/sysv/linux/pt-raise.c:37
#1  0x000000000042d289 in ap_buffered_log_writer
(r=0x7f51a40053d0, handle=0x20eeba0,
strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14,
len=82) at mod_log_config.c:1368
#2  0x000000000042b10d in config_log_transaction
(r=0x7f51a40053d0, cls=0x20b9d50,
default_format=0x20ee370) at mod_log_config.c:930
#3  0x000000000042aad6 in multi_log_transaction
(r=0x7f51a40053d0) at mod_log_config.c:950
#4  0x000000000046cb2d in ap_run_log_transaction
(r=0x7f51a40053d0) at protocol.c:1563
#5  0x0000000000436e81 in ap_process_request
(r=0x7f51a40053d0) at http_request.c:312
#6  0x000000000042e9da in ap_process_http_connection
(c=0x7f519c000b68) at http_core.c:293
#7  0x0000000000465cdd in ap_run_process_connection
(c=0x7f519c000b68) at connection.c:85
#8  0x00000000004661f5 in ap_process_connection
(c=0x7f519c000b68, csd=0x7f519c000a20) at
connection.c:211
#9  0x0000000000451ba0 in process_socket
(p=0x7f519c0009b8, sock=0x7f519c000a20,
my_child_num=0, my_thread_num=0,
bucket_alloc=0x7f51a4001348) at worker.c:632
#10 0x0000000000451221 in worker_thread
(thd=0x210fa90, dummy=0x7f51a40008c0) at worker.c:946
#11 0x00007f51ac87c555 in dummy_worker
(opaque=0x210fa90) at thread.c:127
#12 0x00007f51abae0182 in start_thread
(arg=0x7f51aa8ef700) at pthread_create.c:312
#13 0x00007f51ab80d47d in clone () at ../sysdeps/
unix/sysv/linux/x86_64/clone.S:111
```



Debugging In-Production Software Failures Today

Understand root cause

```
#0  0x00007f51abae820b in raise (sig=11) at ../nptl/
sysdeps/unix/sysv/linux/pt-raise.c:37
#1  0x000000000042d289 in ap_buffered_log_writer
(r=0x7f51a40053d0, handle=0x20eeba0,
strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14,
len=82) at mod_log_config.c:1368
#2  0x000000000042b10d in config_log_transaction
(r=0x7f51a40053d0, cls=0x20b9d50,
default_format=0x20ee370) at mod_log_config.c:930
#3  0x000000000042aad6 in multi_log_transaction
(r=0x7f51a40053d0) at mod_log_config.c:950
#4  0x000000000046cb2d in ap_run_log_transaction
(r=0x7f51a40053d0) at protocol.c:1563
#5  0x0000000000436e81 in ap_process_request
(r=0x7f51a40053d0) at http_request.c:312
#6  0x000000000042e9da in ap_process_http_connection
(c=0x7f519c000b68) at http_core.c:293
#7  0x0000000000465cdd in ap_run_process_connection
(c=0x7f519c000b68) at connection.c:85
#8  0x00000000004661f5 in ap_process_connection
(c=0x7f519c000b68, csd=0x7f519c000a20) at
connection.c:211
#9  0x0000000000451ba0 in process_socket
(p=0x7f519c0009b8, sock=0x7f519c000a20,
my_child_num=0, my_thread_num=0,
bucket_alloc=0x7f51a4001348) at worker.c:632
#10 0x0000000000451221 in worker_thread
(thd=0x210fa90, dummy=0x7f51a40008c0) at worker.c:946
#11 0x00007f51ac87c555 in dummy_worker
(opaque=0x210fa90) at thread.c:127
#12 0x00007f51abae0182 in start_thread
(arg=0x7f51aa8ef700) at pthread_create.c:312
#13 0x00007f51ab80d47d in clone () at ../sysdeps/
unix/sysv/linux/x86_64/clone.S:111
```

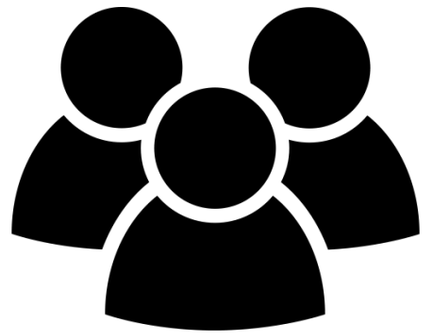


Reproduce the failure

Debugging In-Production Software Failures Today

Understand root cause

```
#0  0x00007f51abae820b in raise (sig=11) at ../nptl/
sysdeps/unix/sysv/linux/pt-raise.c:37
#1  0x000000000042d289 in ap_buffered_log_writer
(r=0x7f51a40053d0, handle=0x20eeba0,
strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14,
len=82) at mod_log_config.c:1368
#2  0x000000000042b10d in config_log_transaction
(r=0x7f51a40053d0, cls=0x20b9d50,
default_format=0x20ee370) at mod_log_config.c:930
#3  0x000000000042aad6 in multi_log_transaction
(r=0x7f51a40053d0) at mod_log_config.c:950
#4  0x000000000046cb2d in ap_run_log_transaction
(r=0x7f51a40053d0) at protocol.c:1563
#5  0x0000000000436e81 in ap_process_request
(r=0x7f51a40053d0) at http_request.c:312
#6  0x000000000042e9da in ap_process_http_connection
(c=0x7f519c000b68) at http_core.c:293
#7  0x0000000000465cdd in ap_run_process_connection
(c=0x7f519c000b68) at connection.c:85
#8  0x00000000004661f5 in ap_process_connection
(c=0x7f519c000b68, csd=0x7f519c000a20) at
connection.c:211
#9  0x0000000000451ba0 in process_socket
(p=0x7f519c0009b8, sock=0x7f519c000a20,
my_child_num=0, my_thread_num=0,
bucket_alloc=0x7f51a4001348) at worker.c:632
#10 0x0000000000451221 in worker_thread
(thd=0x210fa90, dummy=0x7f51a40008c0) at worker.c:946
#11 0x00007f51ac87c555 in dummy_worker
(opaque=0x210fa90) at thread.c:127
#12 0x00007f51abae0182 in start_thread
(arg=0x7f51aa8ef700) at pthread_create.c:312
#13 0x00007f51ab80d47d in clone () at ../sysdeps/
unix/sysv/linux/x86_64/clone.S:111
```



Reproduce the failure

Related Work

- Collaborative approaches
  - *WER [SOSP'09], CBI [PLDI'05], CCI [OOPSLA'10]*
- Identifying differences of failing and successful runs
  - *Delta debugging [TSE'02], Symbiosis [PLDI'15]*
- Record & replay, checkpointing
  - *ODR [SOSP'09], Triage [SOSP'07]*
- Hardware support
  - *PBI [ASPLOS'13], LBRA/LCRA [ASPLOS'14]*

Related Work

- Collaborative approaches
  - *WER [SOSP'09], CBI [PLDI'05], CCI [OOPSLA'10]*
- Identifying differences of failing and successful runs
  - *Delta debugging [TSE'02], Symbiosis [PLDI'15]*
- Record & replay, checkpointing
  - *ODR [SOSP'09], Triage [SOSP'07]*
- Hardware support
  - *PBI [ASPLOS'13], LBRA/LCRA [ASPLOS'14]*

Contributions

Contributions

Goal: automate the manual detective work of debugging

Contributions

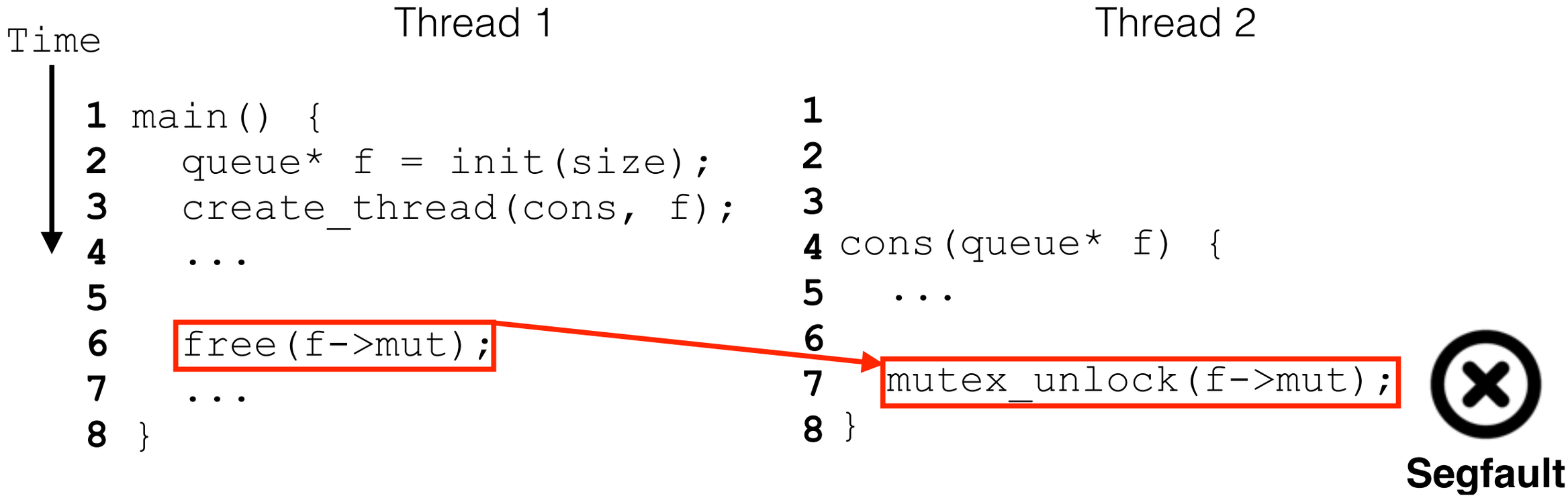
Goal: automate the manual detective work of debugging

Failure sketching

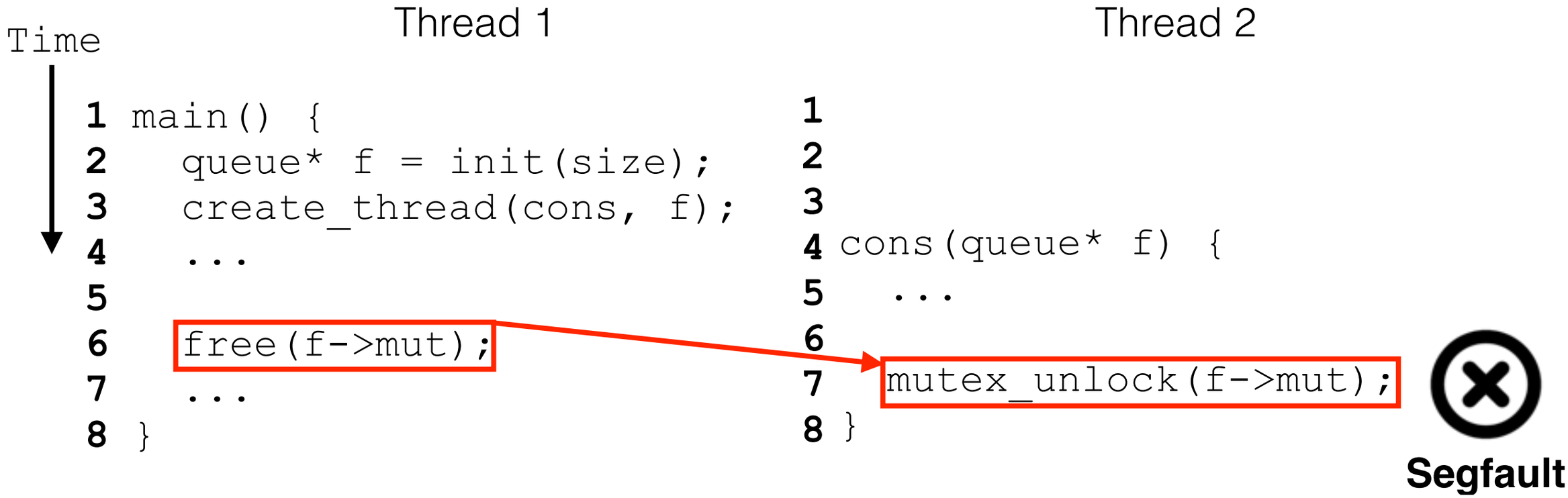
Complements in-house static analysis with in-production dynamic analysis

Automatically and efficiently builds accurate failure sketches that show root causes of failures

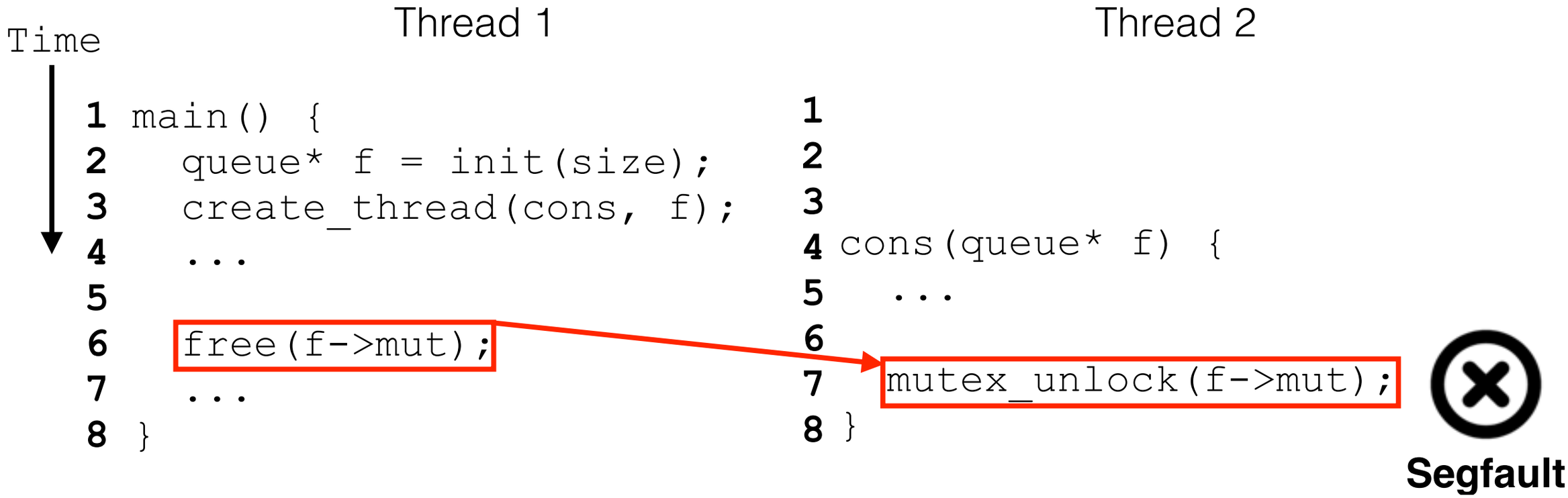
Failure Sketch



Failure Sketch



Failure Sketch



Failure Sketch

Time

Thread 1

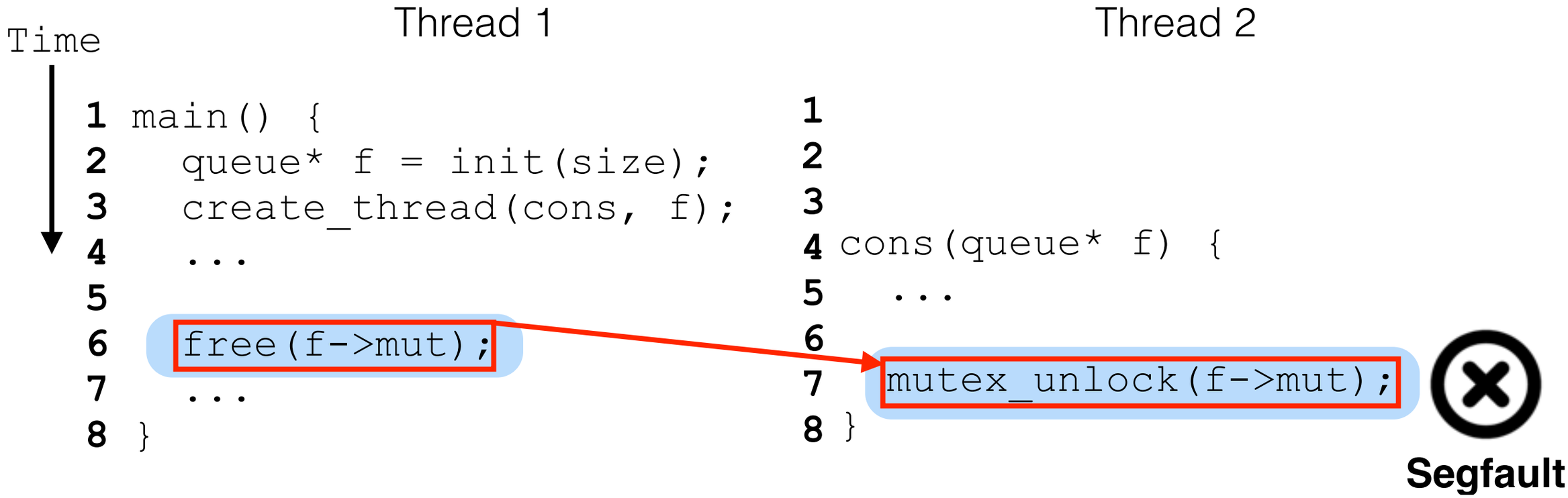
```
1 main() {
2   queue* f = init(size);
3   create_thread(cons, f);
4   ...
5
6   free(f->mut);
7   ...
8 }
```

Thread 2

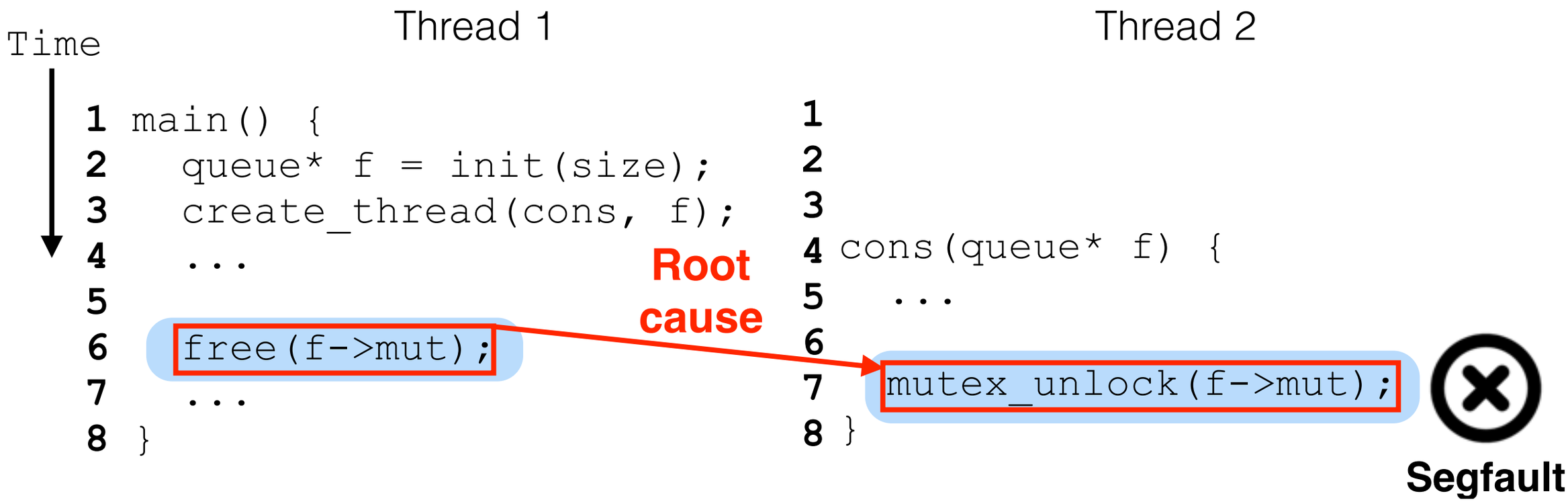
```
1
2
3
4 cons(queue* f) {
5   ...
6
7   mutex_unlock(f->mut);
8 }
```

Segfault

Failure Sketch



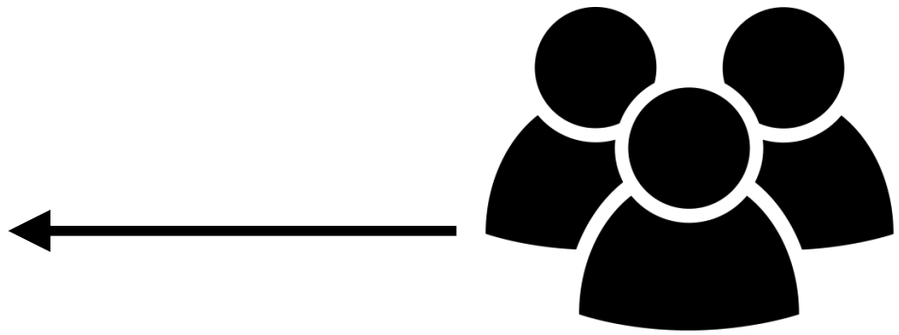
Failure Sketch

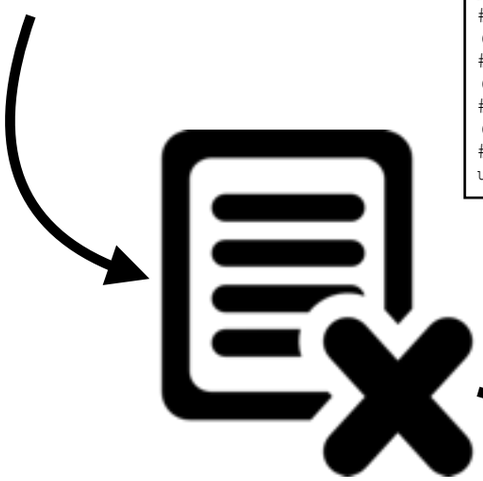


Failure Sketch Usage Model

Understand root cause

```
#0  0x00007f51abae820b in raise (sig=11) at ../nptl/
sysdeps/unix/sysv/linux/pt-raise.c:37
#1  0x000000000042d289 in ap_buffered_log_writer
(r=0x7f51a40053d0, handle=0x20eeba0,
strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14,
len=82) at mod_log_config.c:1368
#2  0x000000000042b10d in config_log_transaction
(r=0x7f51a40053d0, cls=0x20b9d50,
default_format=0x20ee370) at mod_log_config.c:930
#3  0x000000000042aad6 in multi_log_transaction
(r=0x7f51a40053d0) at mod_log_config.c:950
#4  0x000000000046cb2d in ap_run_log_transaction
(r=0x7f51a40053d0) at protocol.c:1563
#5  0x0000000000436e81 in ap_process_request
(r=0x7f51a40053d0) at http_request.c:312
#6  0x000000000042e9da in ap_process_http_connection
(c=0x7f519c000b68) at http_core.c:293
#7  0x0000000000465cdd in ap_run_process_connection
(c=0x7f519c000b68) at connection.c:85
#8  0x00000000004661f5 in ap_process_connection
(c=0x7f519c000b68, csd=0x7f519c000a20) at
connection.c:211
#9  0x0000000000451ba0 in process_socket
(p=0x7f519c0009b8, sock=0x7f519c000a20,
my_child_num=0, my_thread_num=0,
bucket_alloc=0x7f51a4001348) at worker.c:632
#10 0x0000000000451221 in worker_thread
(thd=0x210fa90, dummy=0x7f51a40008c0) at worker.c:946
#11 0x00007f51ac87c555 in dummy_worker
(opaque=0x210fa90) at thread.c:127
#12 0x00007f51abae0182 in start_thread
(arg=0x7f51aa8ef700) at pthread_create.c:312
#13 0x00007f51ab80d47d in clone () at ../sysdeps/
unix/sysv/linux/x86_64/clone.S:111
```



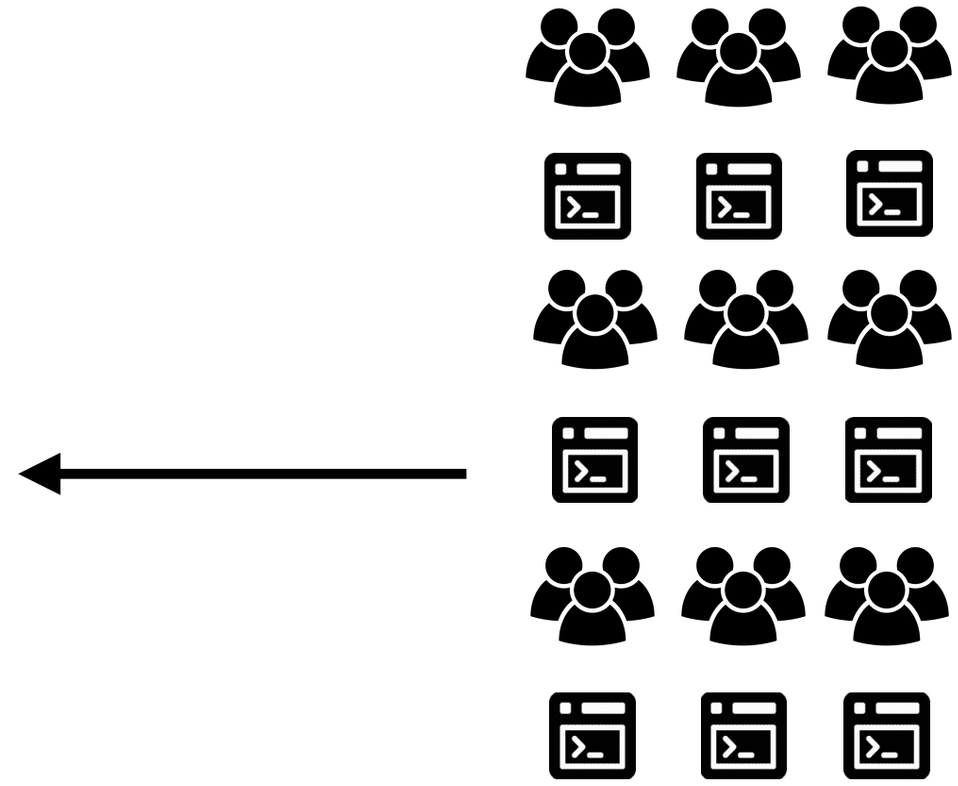


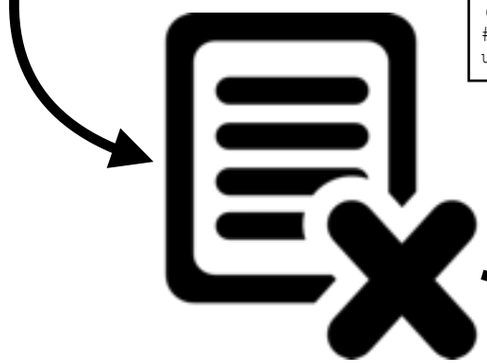
Reproduce the failure

Failure Sketch Usage Model

Understand root cause

```
#0  0x00007f51abae820b in raise (sig=11) at ../nptl/
sysdeps/unix/sysv/linux/pt-raise.c:37
#1  0x000000000042d289 in ap_buffered_log_writer
(r=0x7f51a40053d0, handle=0x20eeba0,
strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14,
len=82) at mod_log_config.c:1368
#2  0x000000000042b10d in config_log_transaction
(r=0x7f51a40053d0, cls=0x20b9d50,
default_format=0x20ee370) at mod_log_config.c:930
#3  0x000000000042aad6 in multi_log_transaction
(r=0x7f51a40053d0) at mod_log_config.c:950
#4  0x000000000046cb2d in ap_run_log_transaction
(r=0x7f51a40053d0) at protocol.c:1563
#5  0x0000000000436e81 in ap_process_request
(r=0x7f51a40053d0) at http_request.c:312
#6  0x000000000042e9da in ap_process_http_connection
(c=0x7f519c000b68) at http_core.c:293
#7  0x0000000000465cdd in ap_run_process_connection
(c=0x7f519c000b68) at connection.c:85
#8  0x00000000004661f5 in ap_process_connection
(c=0x7f519c000b68, csd=0x7f519c000a20) at
connection.c:211
#9  0x0000000000451ba0 in process_socket
(p=0x7f519c0009b8, sock=0x7f519c000a20,
my_child_num=0, my_thread_num=0,
bucket_alloc=0x7f51a4001348) at worker.c:632
#10 0x0000000000451221 in worker_thread
(thd=0x210fa90, dummy=0x7f51a40008c0) at worker.c:946
#11 0x00007f51ac87c555 in dummy_worker
(opaque=0x210fa90) at thread.c:127
#12 0x00007f51abae0182 in start_thread
(arg=0x7f51aa8ef700) at pthread_create.c:312
#13 0x00007f51ab80d47d in clone () at ../sysdeps/
unix/sysv/linux/x86_64/clone.S:111
```



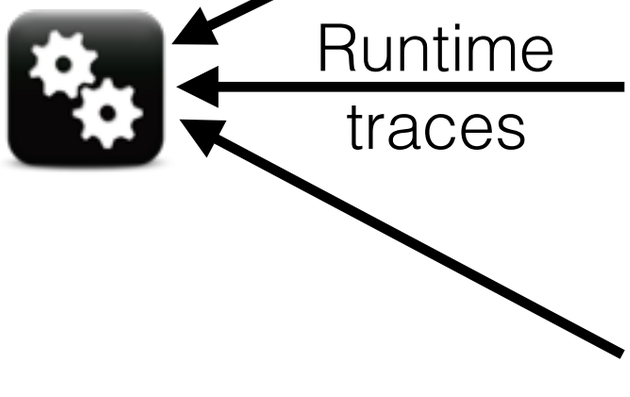


Reproduce the failure

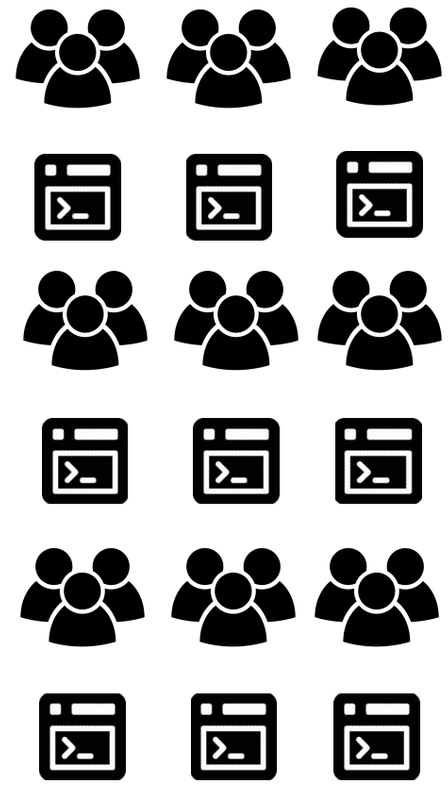
Failure Sketch Usage Model

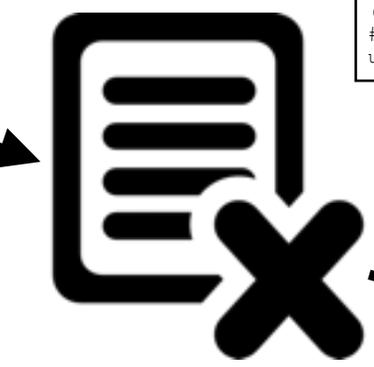
Understand root cause

```
#0  0x00007f51abae820b in raise (sig=11) at ../nptl/
sysdeps/unix/sysv/linux/pt-raise.c:37
#1  0x000000000042d289 in ap_buffered_log_writer
(r=0x7f51a40053d0, handle=0x20eeba0,
strs=0x7f51a4003578, strl=0x7f51a40035e8, nelts=14,
len=82) at mod_log_config.c:1368
#2  0x000000000042b10d in config_log_transaction
(r=0x7f51a40053d0, cls=0x20b9d50,
default_format=0x20ee370) at mod_log_config.c:930
#3  0x000000000042aad6 in multi_log_transaction
(r=0x7f51a40053d0) at mod_log_config.c:950
#4  0x000000000046cb2d in ap_run_log_transaction
(r=0x7f51a40053d0) at protocol.c:1563
#5  0x0000000000436e81 in ap_process_request
(r=0x7f51a40053d0) at http_request.c:312
#6  0x000000000042e9da in ap_process_http_connection
(c=0x7f519c000b68) at http_core.c:293
#7  0x00000000000465cdd in ap_run_process_connection
(c=0x7f519c000b68) at connection.c:85
#8  0x00000000004661f5 in ap_process_connection
(c=0x7f519c000b68, csd=0x7f519c000a20) at
connection.c:211
#9  0x0000000000451ba0 in process_socket
(p=0x7f519c0009b8, sock=0x7f519c000a20,
my_child_num=0, my_thread_num=0,
bucket_alloc=0x7f51a4001348) at worker.c:632
#10 0x0000000000451221 in worker_thread
(thd=0x210fa90, dummy=0x7f51a40008c0) at worker.c:946
#11 0x00007f51ac87c555 in dummy_worker
(opaque=0x210fa90) at thread.c:127
#12 0x00007f51abae0182 in start_thread
(arg=0x7f51aa8ef700) at pthread_create.c:312
#13 0x00007f51ab80d47d in clone () at ../sysdeps/
unix/sysv/linux/x86_64/clone.S:111
```



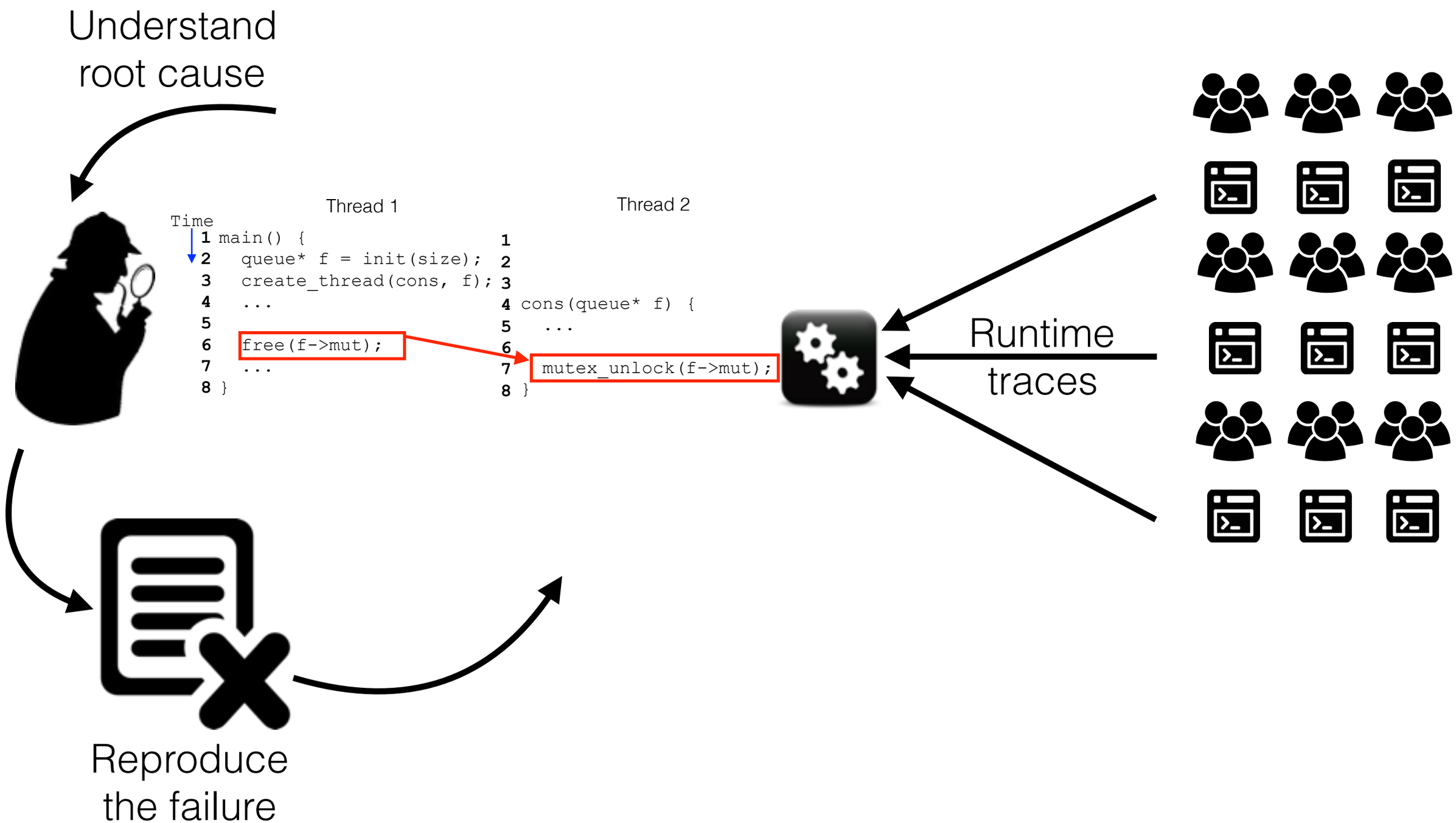
Runtime traces



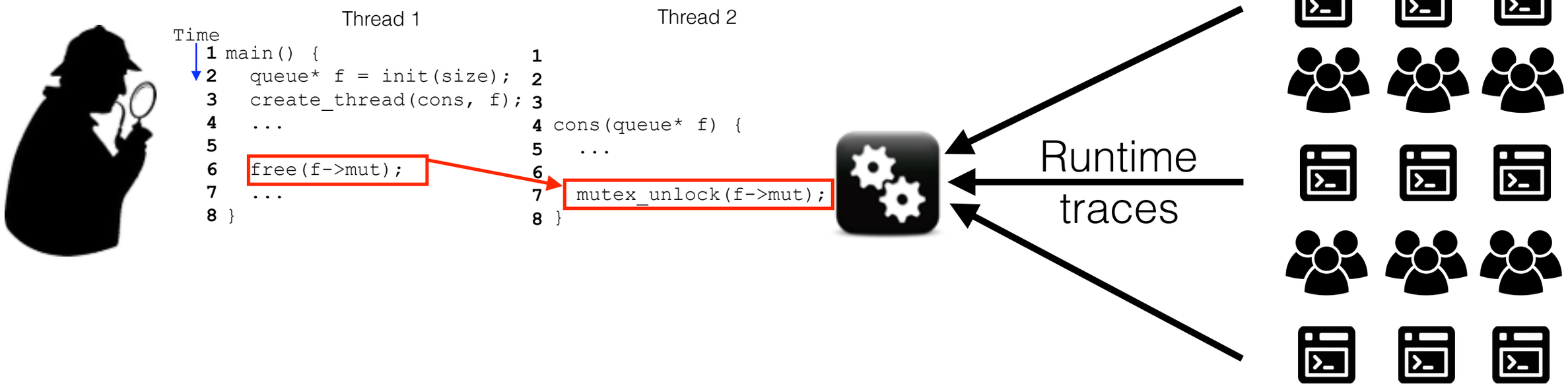


Reproduce the failure

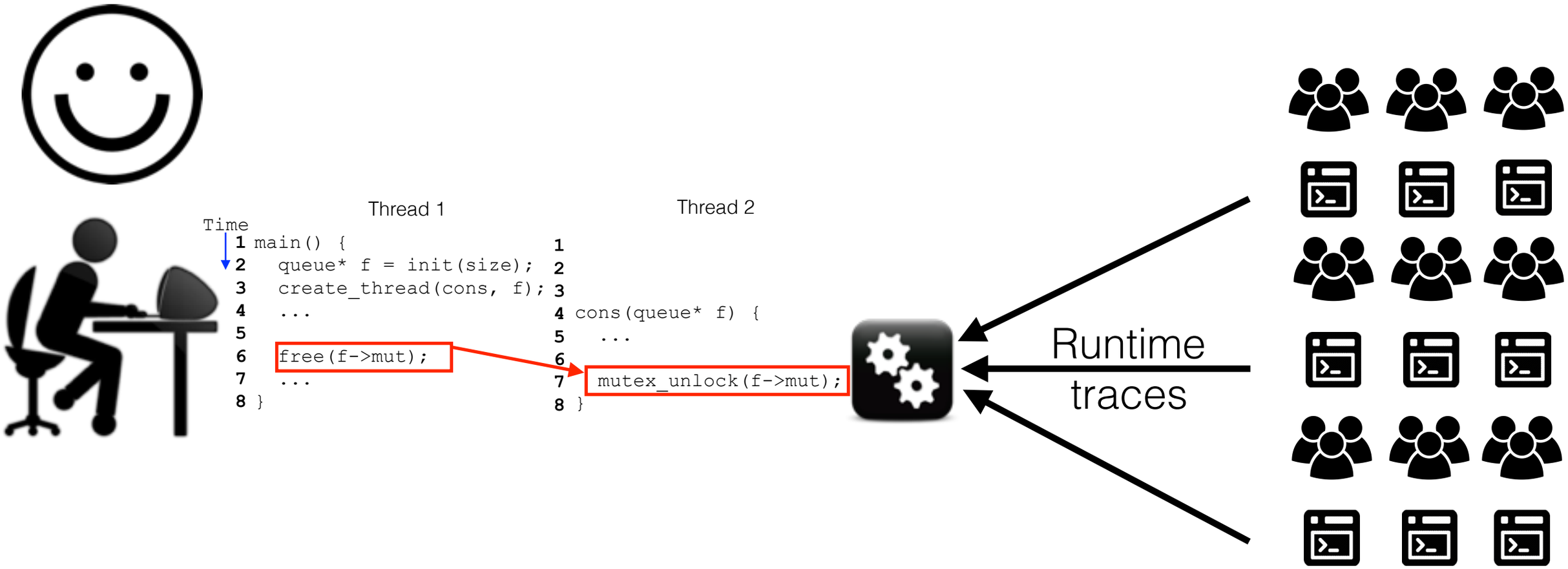
Failure Sketch Usage Model



Failure Sketch Usage Model



Failure Sketch Usage Model



Outline

- Challenges
- Design
- Evaluation

Outline

- Challenges
- Design
- Evaluation

Challenges of Building Failure Sketches

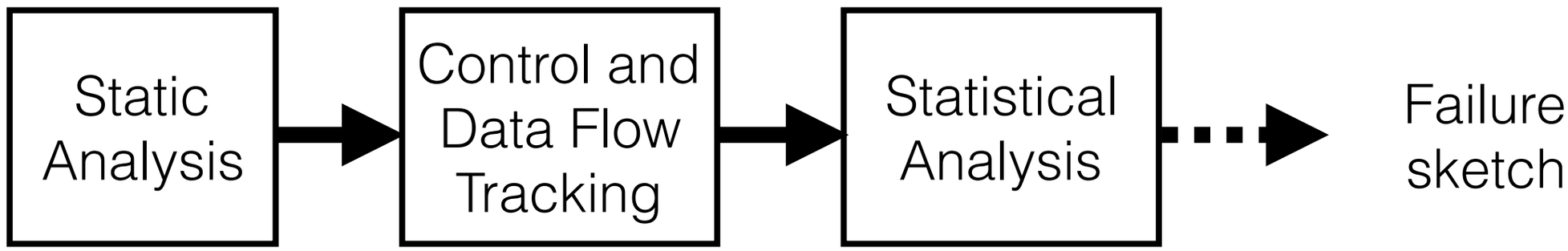
- Accuracy
  - *Exclude all irrelevant information, preserve all relevant one*
- Recurrence
  - *Gathering enough execution information from rare failures*
- Latency
  - *Achieve high accuracy after just a few recurrences*

Outline

- Challenges
- Design
- Evaluation

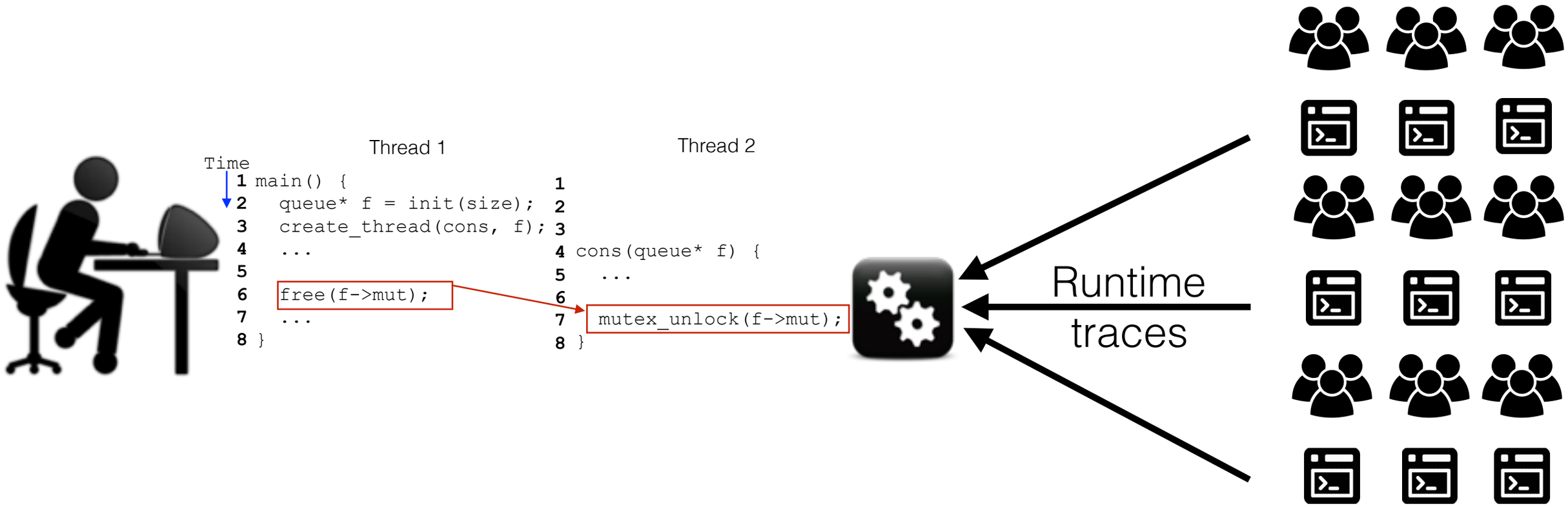
Outline

- Challenges
- Design



- Evaluation

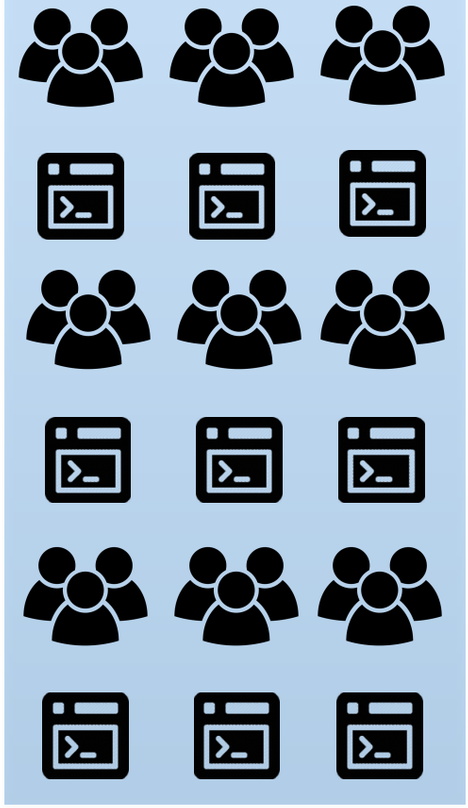
Gist System Architecture



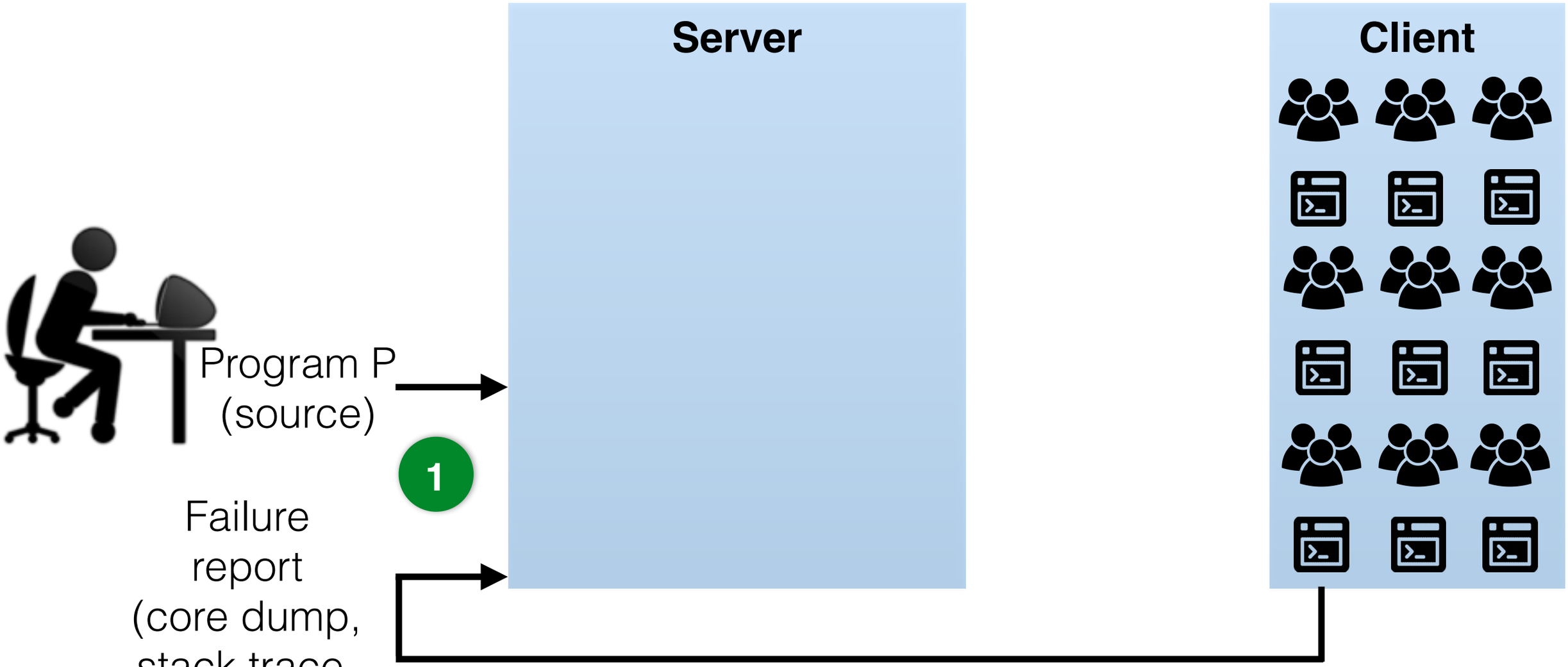
Gist System Architecture

Server

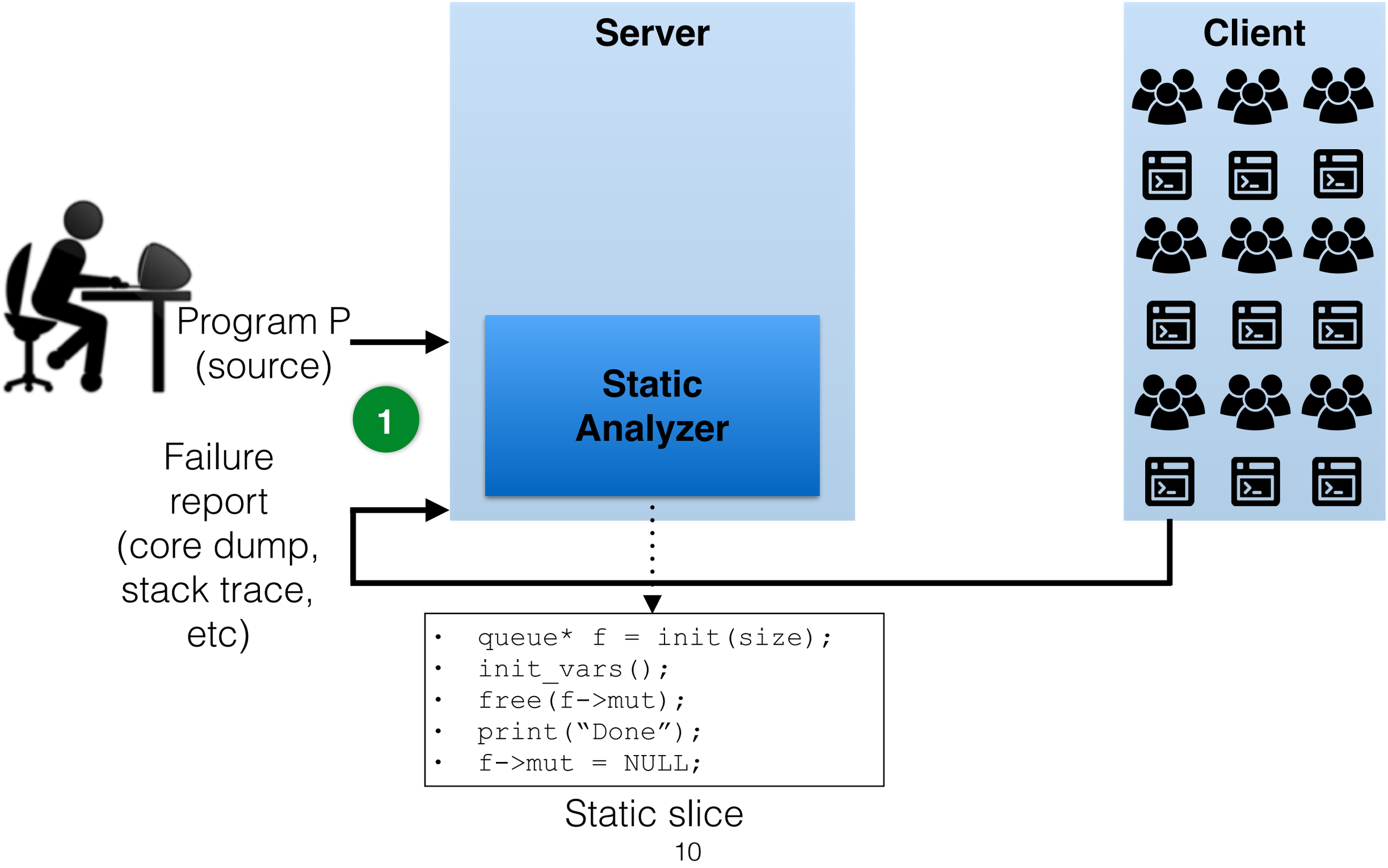
Client



Gist System Architecture



Gist System Architecture



Gist System Architecture

Program P
(source)

Failure
report
(core dump,
stack trace,
etc)

1

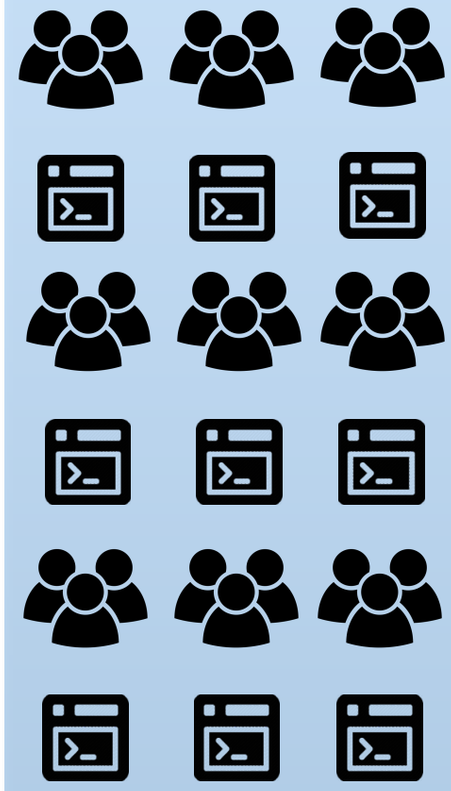
Server

Static Analyzer

2

Instrumentation

Client



Tracking
control and
data flow

```
•  queue* f = init(size);
•  init_vars();
•  free(f->mut);
•  print("Done");
•  f->mut = NULL;
```

Static slice

Gist System Architecture

Program P
(source)

Failure
report
(core dump,
stack trace,
etc)

1

Server

Static Analyzer

2

Instrumentation

3

Refinement with
runtime traces

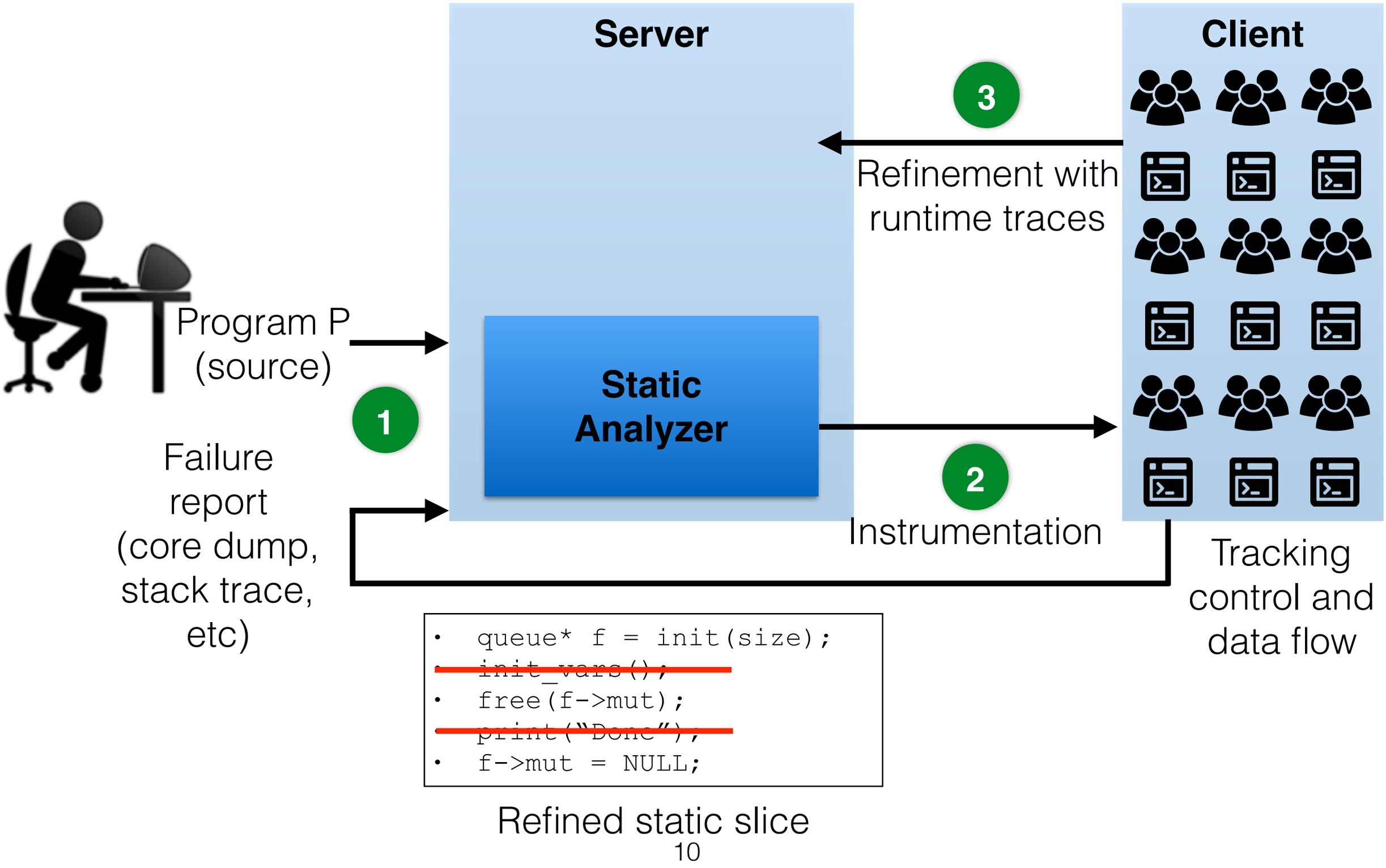
Client

Tracking
control and
data flow

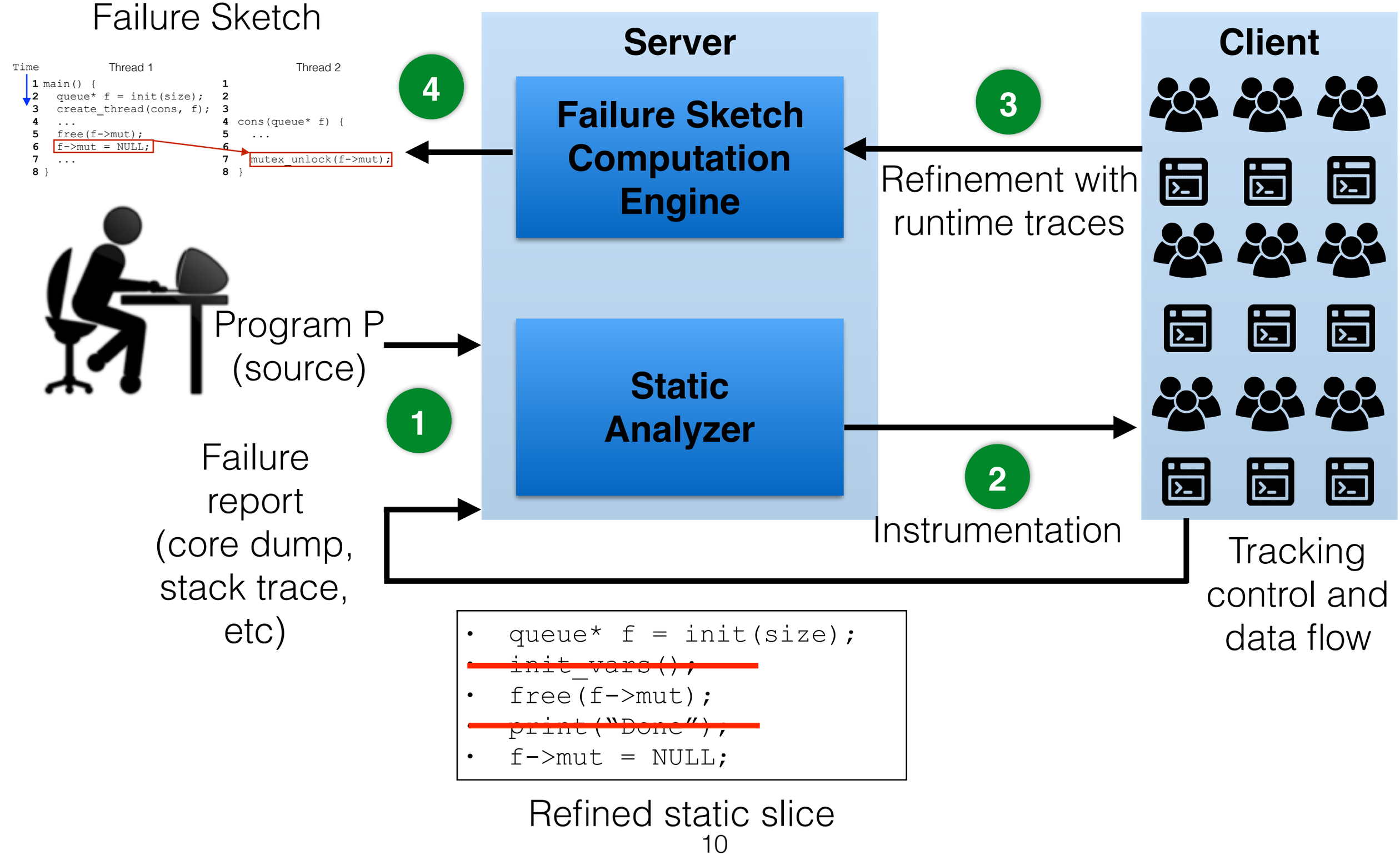
```
•  queue* f = init(size);
•  init_vars();
•  free(f->mut);
•  print("Done");
•  f->mut = NULL;
```

Static slice

Gist System Architecture

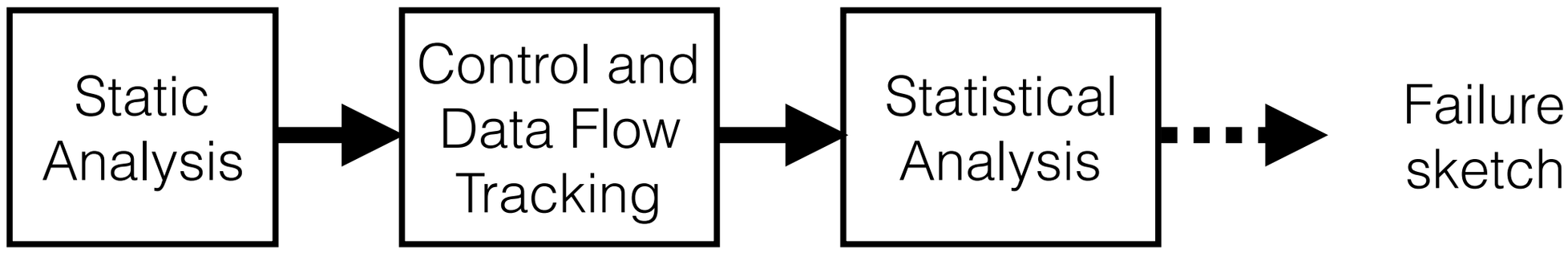


Gist System Architecture



Outline

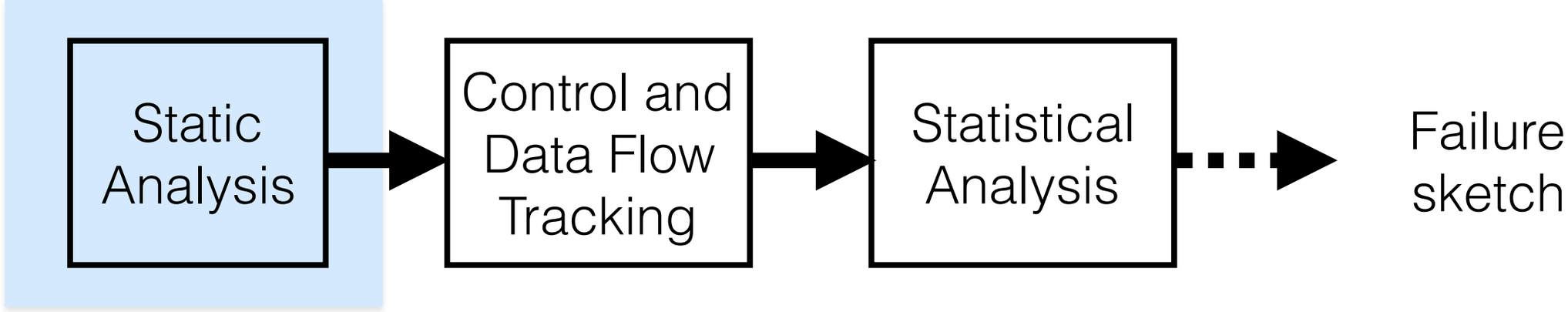
- Challenges
- Design



- Evaluation

Outline

- Challenges
- Design



- Evaluation

Static Analysis to Reduce the Overhead

- Computes backward slices
  - *Includes statements with dependencies to the failure*
  - *Excludes all other statements*
- Inter-procedural
  - *Identify dependencies across functions*

Static Analysis to Reduce the Overhead

- Computes backward slices
  - *Includes statements with dependencies to the failure*
  - *Excludes all other statements*
- Inter-procedural
  - *Identify dependencies across functions*

Static analysis reduces subsequent runtime tracking (20x)

Example

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Example

```
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}

void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
```

Example

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Example

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Example

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Example

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Segfault

Example

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display size");
  log("State: %u", s->size);
}
```

Segfault

Example: Static Backward Slicing

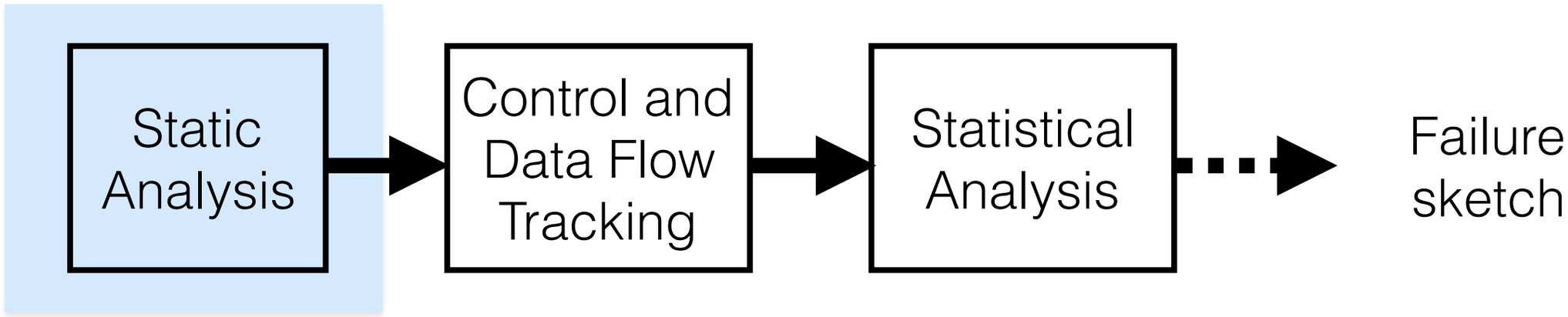
```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Segfault

Outline

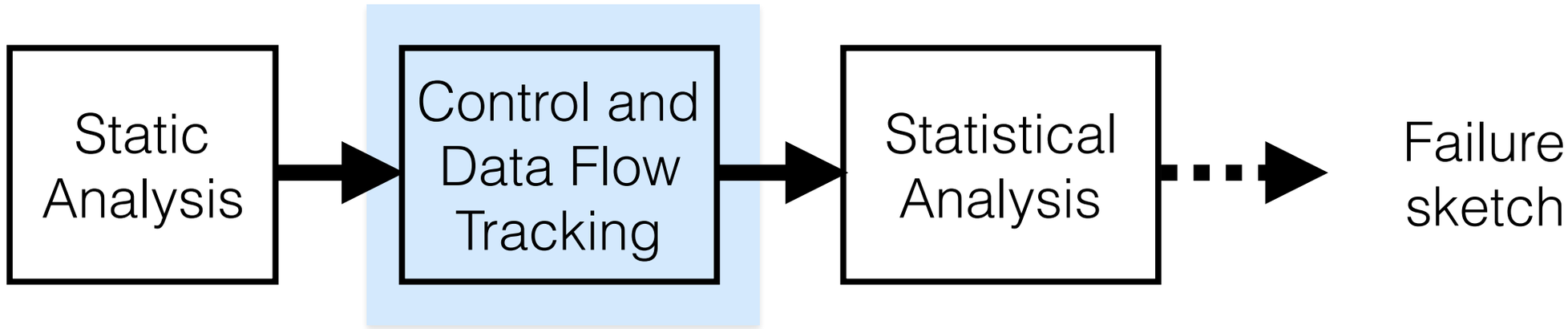
- Challenges
- Design



- Evaluation

Outline

- Challenges
- Design



- Evaluation

Low-Overhead Control Flow Tracking

- Software-based tracking is expensive (up to 15x)
- Hardware-based tracking is more efficient
  - *Intel PT: new feature in Intel CPUs (~40%)*
- Gist combines static analysis and hardware-based control flow tracking
  - *Low overhead (~2%)*

Low-Overhead Control Flow Tracking

- Software-based tracking is expensive (up to 15x)
- Hardware-based tracking is more efficient
  - *Intel PT: new feature in Intel CPUs (~40%)*
- Gist combines static analysis and hardware-based control flow tracking
  - *Low overhead (~2%)*

Static analysis → Low-overhead control flow tracking

Example: Control Flow Tracking (Step 1)

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Segfault

Example: Control Flow Tracking (Step 2)

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Segfault

Example: Control Flow Tracking (Step 2)

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Segfault

Static analysis + control flow tracking shorten the sketch

Data Flow Tracking to Increase Accuracy

Data Flow Tracking to Increase Accuracy

- Data flow information
  - *Variable values & total order of memory accesses*

Data Flow Tracking to Increase Accuracy

- Data flow information
  - *Variable values & total order of memory accesses*
- Hardware watchpoints
  - *Allow tracking reads and writes with low overhead*
  - *Allow tracking the total order of accesses*

Data Flow Tracking to Increase Accuracy

- Data flow information
  - *Variable values & total order of memory accesses*
- Hardware watchpoints
  - *Allow tracking reads and writes with low overhead*
  - *Allow tracking the total order of accesses*
- Monitor multiple clients when run out of watchpoints

Data Flow Tracking to Increase Accuracy

- Data flow information
  - *Variable values & total order of memory accesses*
- Hardware watchpoints
  - *Allow tracking reads and writes with low overhead*
  - *Allow tracking the total order of accesses*
- Monitor multiple clients when run out of watchpoints

Precise ordering information → High accuracy

Example: Data Flow Tracking

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display size");
  log("State: %u", s->size);
}
```

Segfault

Example: Data Flow Tracking

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Segfault

Watch &s

Example: Data Flow Tracking

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}

void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Segfault

Watch &s

Example: Data Flow Tracking

Thread 1

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Thread 2

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

Success

Example: Data Flow Tracking

Thread 1

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Thread 2

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

Success

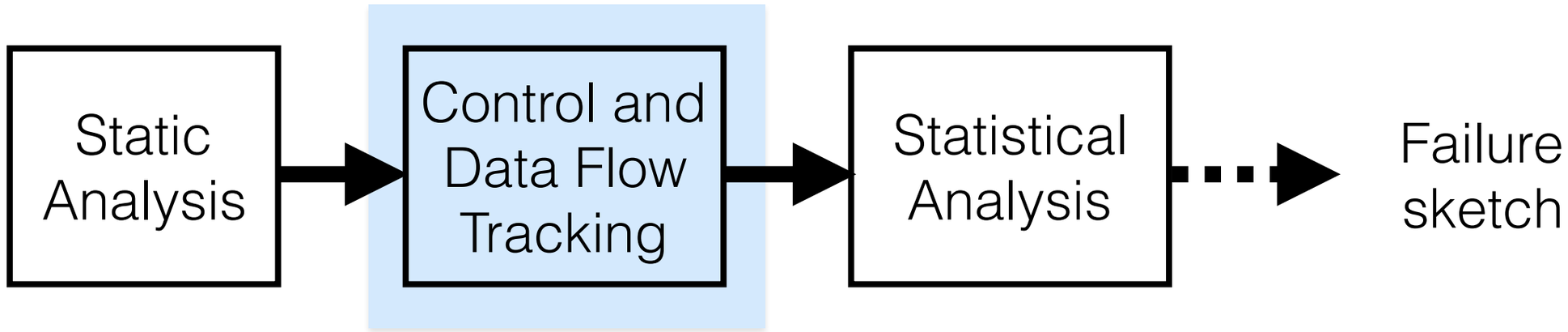
```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

```
void display_size(State* s)
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Failure

Outline

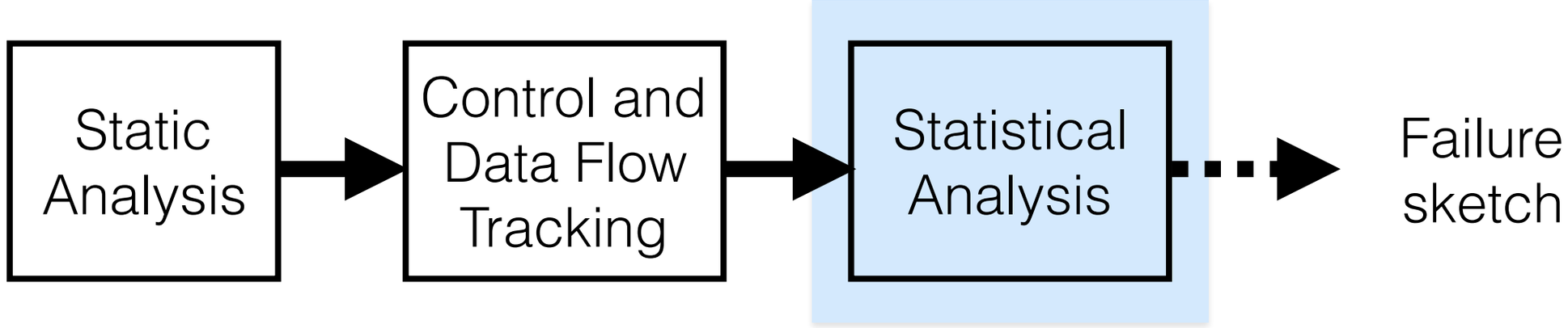
- Challenges
- Design



- Evaluation

Outline

- Challenges
- Design



- Evaluation

Statistical Analysis

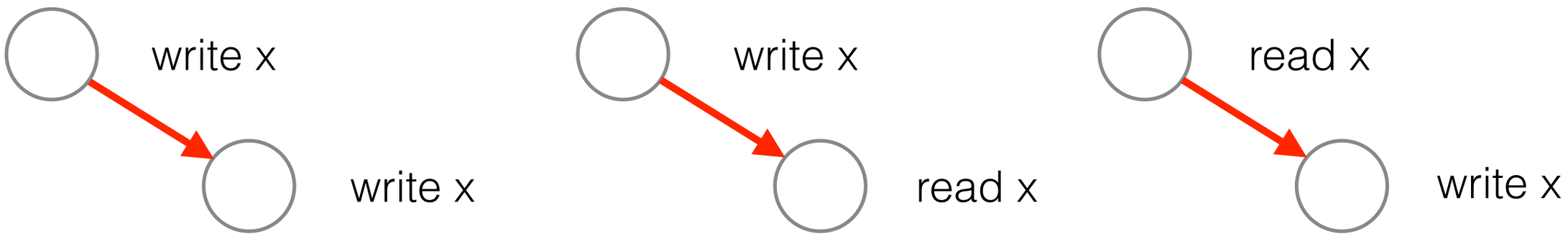
Statistical Analysis

- Identification of failure predictors[1]
  - *A good predictor portends a failure with high probability (e.g., data races, atomicity violations)*

[1] Liblit, B. et al. Scalable statistical bug isolation. PLDI 2005

Statistical Analysis

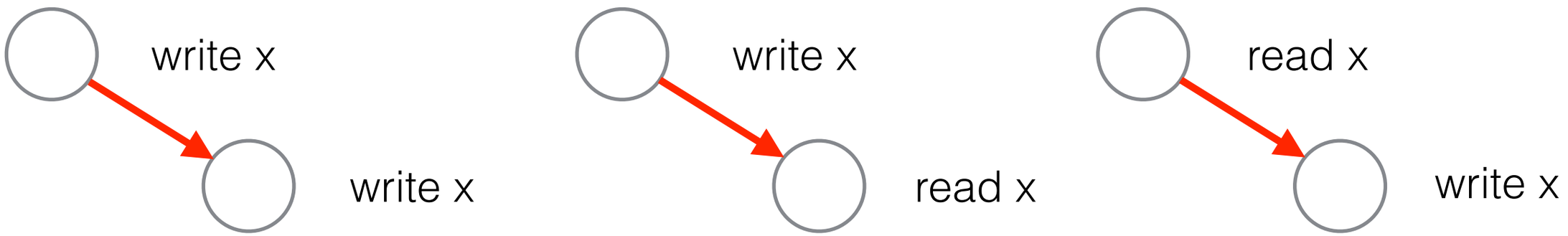
- Identification of failure predictors[1]
  - *A good predictor portends a failure with high probability (e.g., data races, atomicity violations)*
  - *Example: data races*



[1] Liblit, B. et al. Scalable statistical bug isolation. PLDI 2005

Statistical Analysis

- Identification of failure predictors[1]
  - *A good predictor portends a failure with high probability (e.g., data races, atomicity violations)*
  - *Example: data races*



Failure predictors across multiple executions

[1] Liblit, B. et al. Scalable statistical bug isolation. PLDI 2005

Example: Statistical Analysis

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

Success

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

```
void display_size(State* s)
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Failure

Example: Statistical Analysis

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

Success

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Failure

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

Failure

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```

Success

Example: Statistical Analysis

```
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
```

Success

```
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
```

Failure

```
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
```

Success

```
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
```

Failure

```
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
```

Success

```
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
```

Success

```
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
```

Success

```
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
```

Success

```
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
```

Success

```
void cleanup(State* s) {
  log(“Func:cleanup”);
  if(verbose)
    log(“Cleaning up %p”, s);
  delete s;
}
void display_size(State* s) {
  log(“Func:display_size”);
  log(“State: %u”, s->size);
}
```

Failure

Example: Statistical Analysis

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
                    void cleanup(State* s) {
                      log("Func:cleanup");
                      if(verbose)
                        log("Cleaning up %p", s);
                      delete s;
                    }
```

Success

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
                    void display_size(State* s) {
                      log("Func:display_size");
                      log("State: %u", s->size);
                    }
```

Failure

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
                    void cleanup(State* s) {
                      log("Func:cleanup");
                      if(verbose)
                        log("Cleaning up %p", s);
                      delete s;
                    }
```

Success

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
                    void display_size(State* s) {
                      log("Func:display_size");
                      log("State: %u", s->size);
                    }
```

Failure

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
                    void cleanup(State* s) {
                      log("Func:cleanup");
                      if(verbose)
                        log("Cleaning up %p", s);
                      delete s;
                    }
```

Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
                    void cleanup(State* s) {
                      log("Func:cleanup");
                      if(verbose)
                        log("Cleaning up %p", s);
                      delete s;
                    }
```

Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
                    void cleanup(State* s) {
                      log("Func:cleanup");
                      if(verbose)
                        log("Cleaning up %p", s);
                      delete s;
                    }
```

Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
                    void cleanup(State* s) {
                      log("Func:cleanup");
                      if(verbose)
                        log("Cleaning up %p", s);
                      delete s;
                    }
```

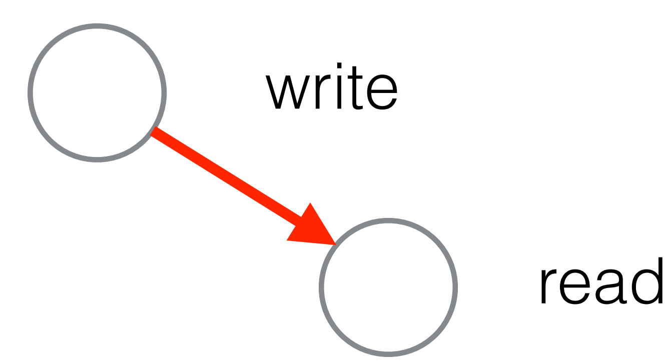
Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
                    void cleanup(State* s) {
                      log("Func:cleanup");
                      if(verbose)
                        log("Cleaning up %p", s);
                      delete s;
                    }
```

Success

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
                    void display_size(State* s) {
                      log("Func:display_size");
                      log("State: %u", s->size);
                    }
```

Failure



Example: Statistical Analysis

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```
Success

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```
Failure

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```
Success

```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```
Failure

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```
Success

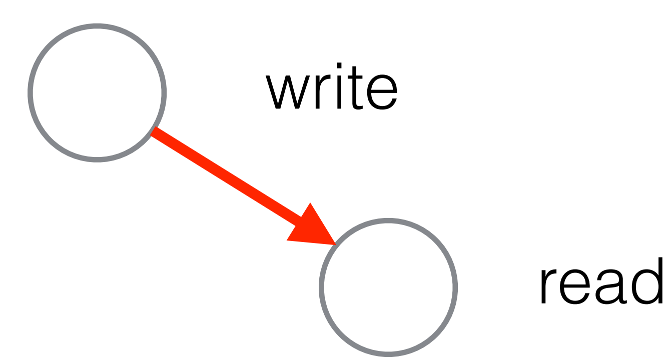
```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```
Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```
Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```
Success

```
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
```
Success

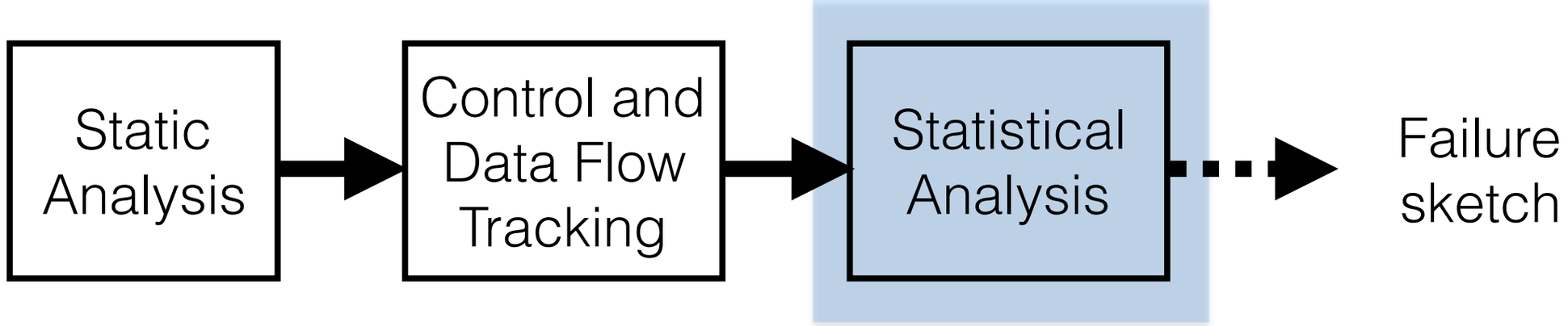
```
void cleanup(State* s) {
  log("Func:cleanup");
  if(verbose)
    log("Cleaning up %p", s);
  delete s;
}
void display_size(State* s) {
  log("Func:display_size");
  log("State: %u", s->size);
}
```
Failure



Static analysis + cooperative dynamic analysis

Outline

- Challenges
- Design



- Evaluation

Outline

- Challenges
- Design
- Evaluation
  - *Does Gist help developers do root cause diagnosis?*
  - *Is Gist efficient?*
  - *Is Gist accurate?*

Experimental Setup

- Client side executions are analyzed in the lab
- Real world server and desktop programs

Do Failure Sketches Help Developers?

- We manually analyzed the usefulness of Gist for 11 failures
- Gist-identified failure predictors point to root causes
- *Developers eliminated those root causes to fix the bugs*
- *Average number of statements to look at: 7*

Do Failure Sketches Help Developers?

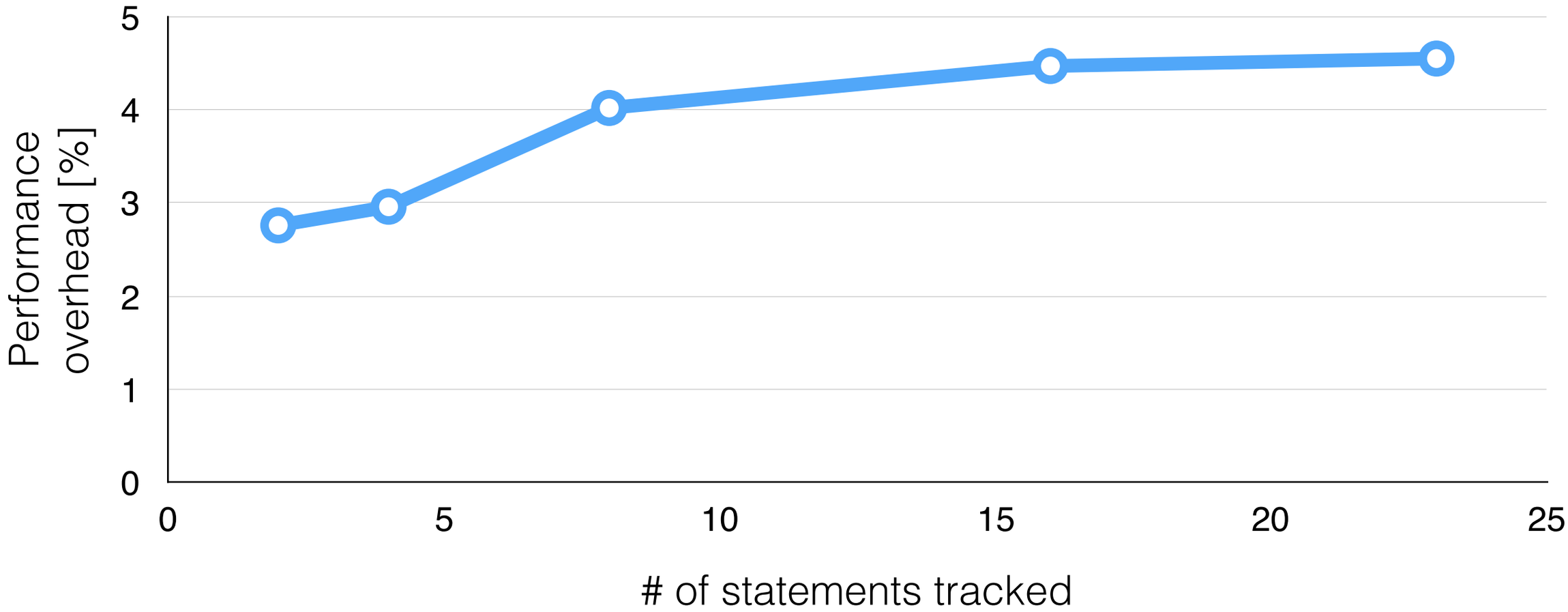
- We manually analyzed the usefulness of Gist for 11 failures

- Gist-identified failure predictors point to root causes
  - *Developers eliminated those root causes to fix the bugs*
  - *Average number of statements to look at: 7*

Gist points developers to the root causes of failures

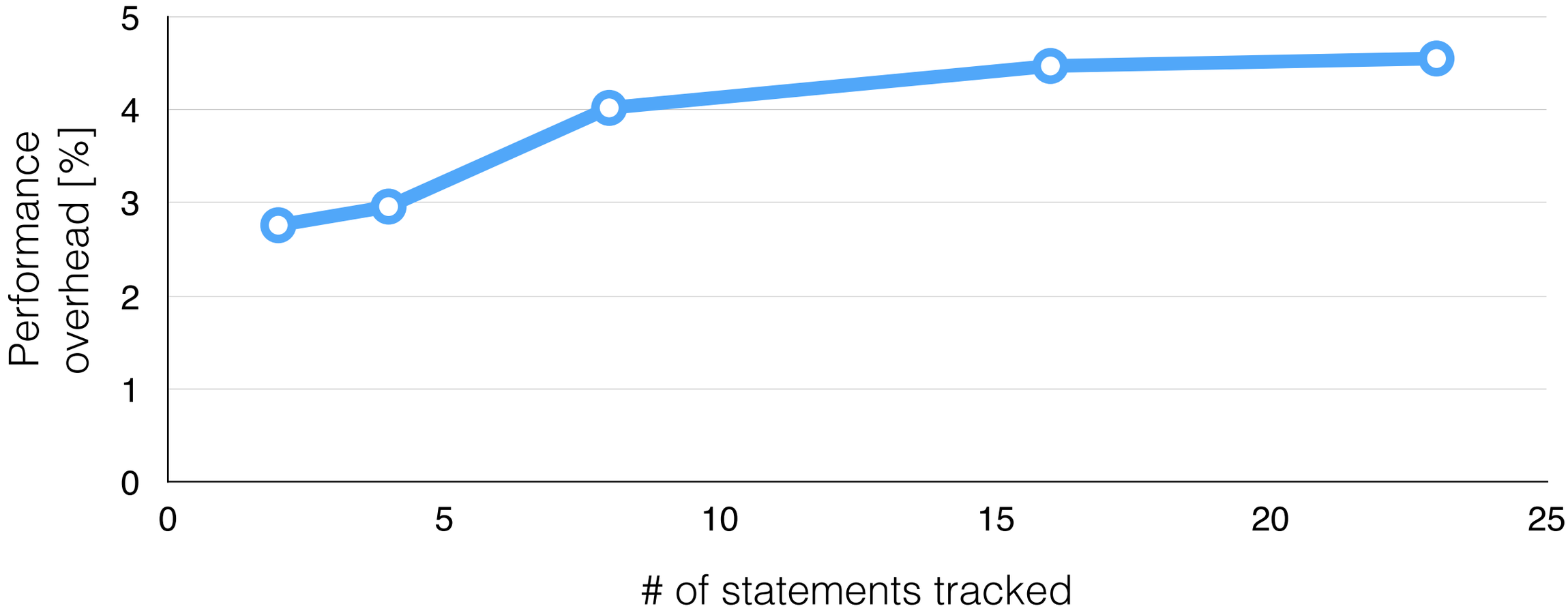
Efficiency

(Control & data flow tracking)



Efficiency

(Control & data flow tracking)



Gist has low average overhead (always below 5%)

Accuracy [%]

Apache-1

Apache-2

Apache-3

Apache-4

Cppcheck-1

Cppcheck-2

Curl

Transmission

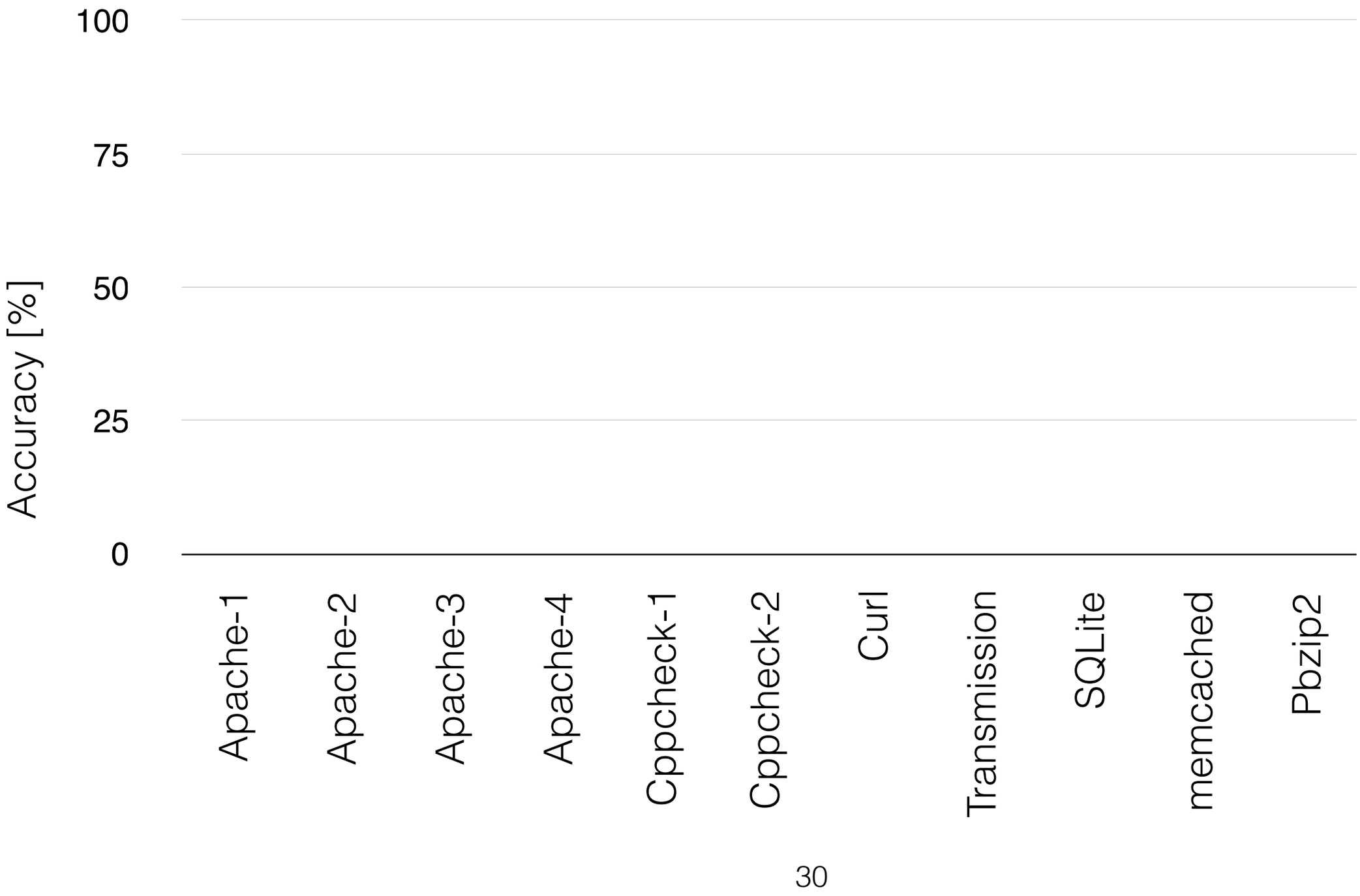
SQLite

memcached

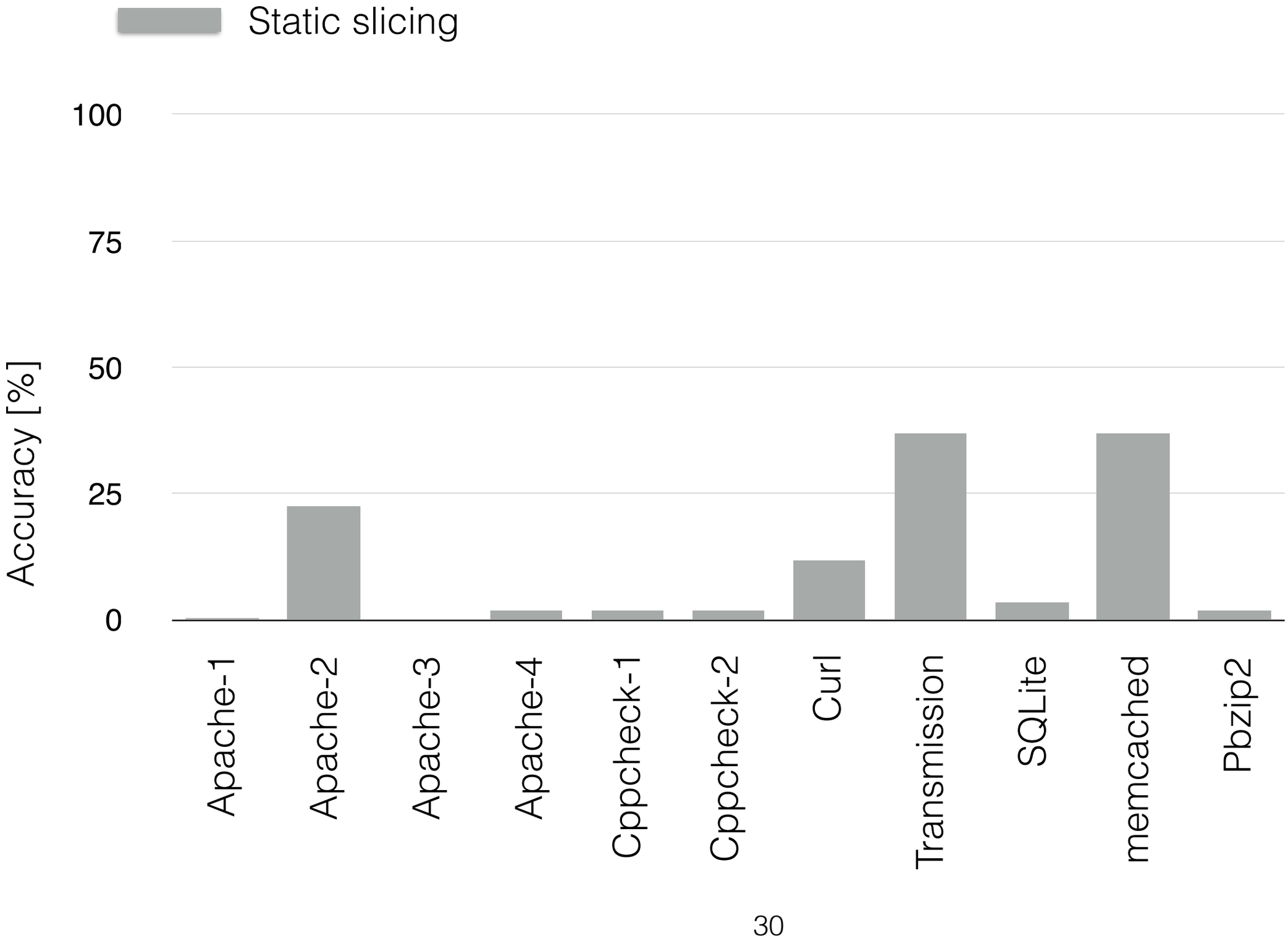
Pbzip2

Accuracy

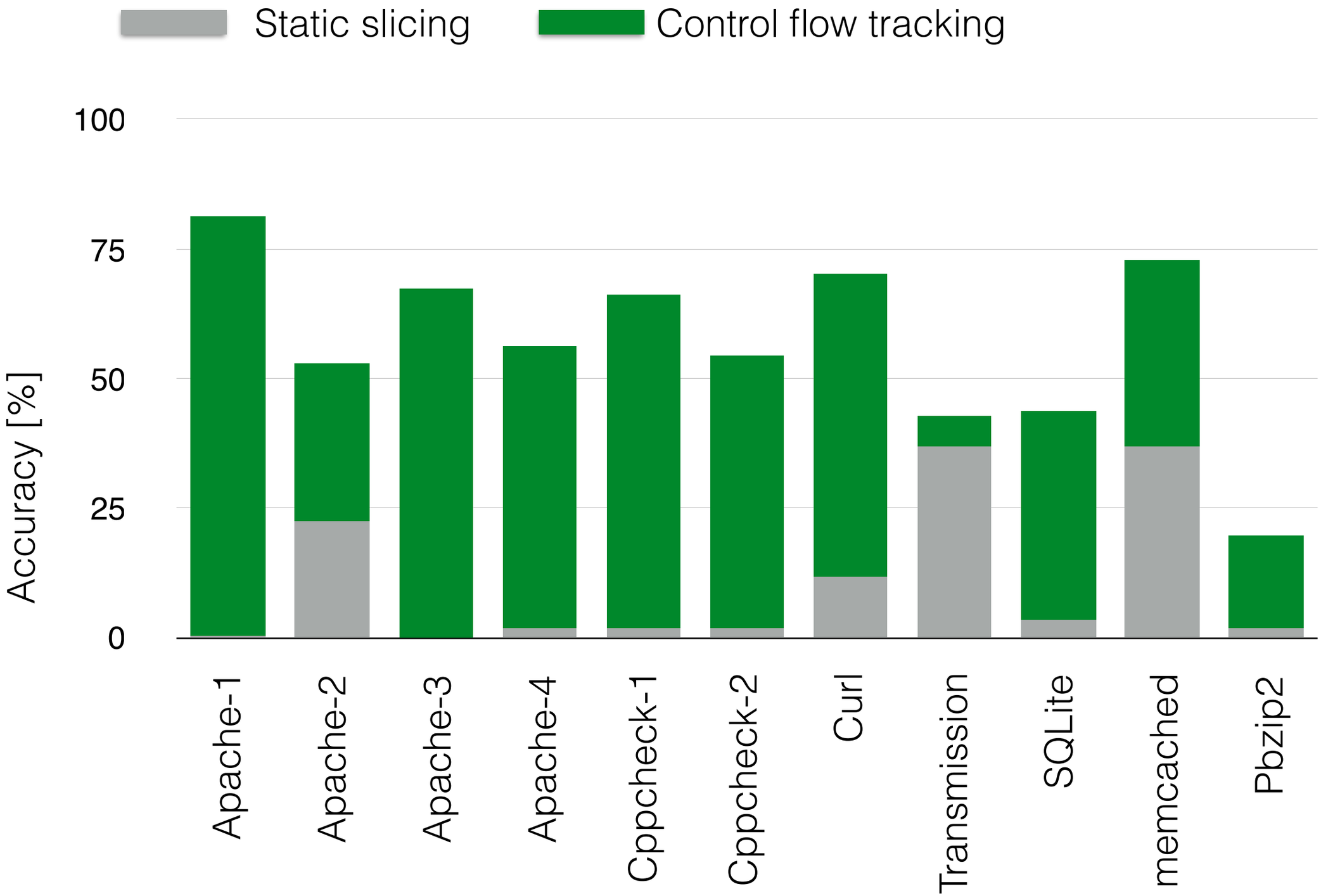
Accuracy


100
75
50
25
0
Accuracy [%]
Apache-1
Apache-2
Apache-3
Apache-4
Cppcheck-1
Cppcheck-2
Curl
Transmission
SQLite
memcached
Pbzip2

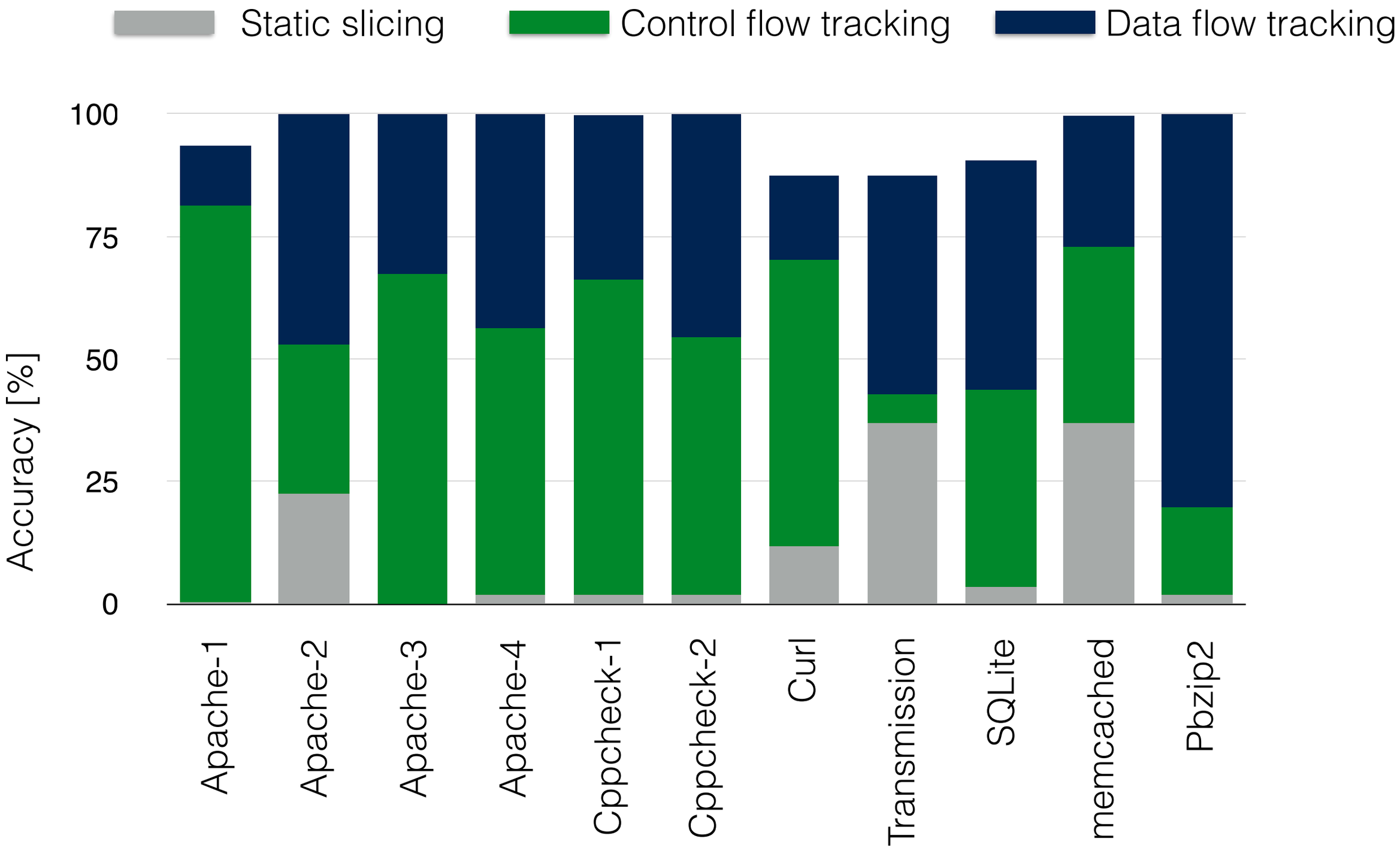
Accuracy


Static slicing
100
75
50
25
0
Accuracy [%]
Apache-1
Apache-2
Apache-3
Apache-4
Cppcheck-1
Cppcheck-2
Curl
Transmission
SQLite
memcached
Pbzip2

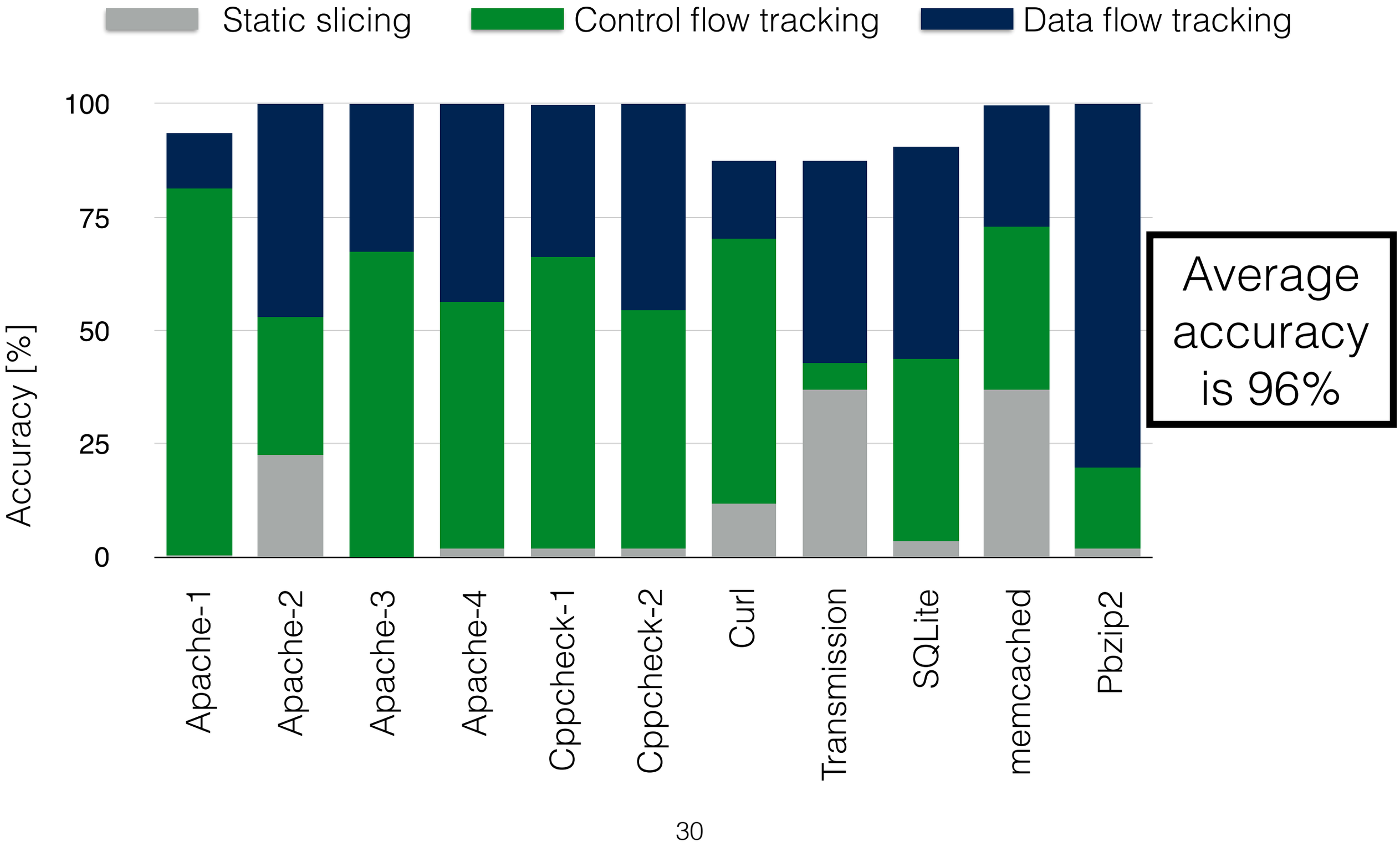
Accuracy


Static slicing
Control flow tracking
100
75
50
25
0
Accuracy [%]
Apache-1
Apache-2
Apache-3
Apache-4
Cppcheck-1
Cppcheck-2
Curl
Transmission
SQLite
memcached
Pbzip2

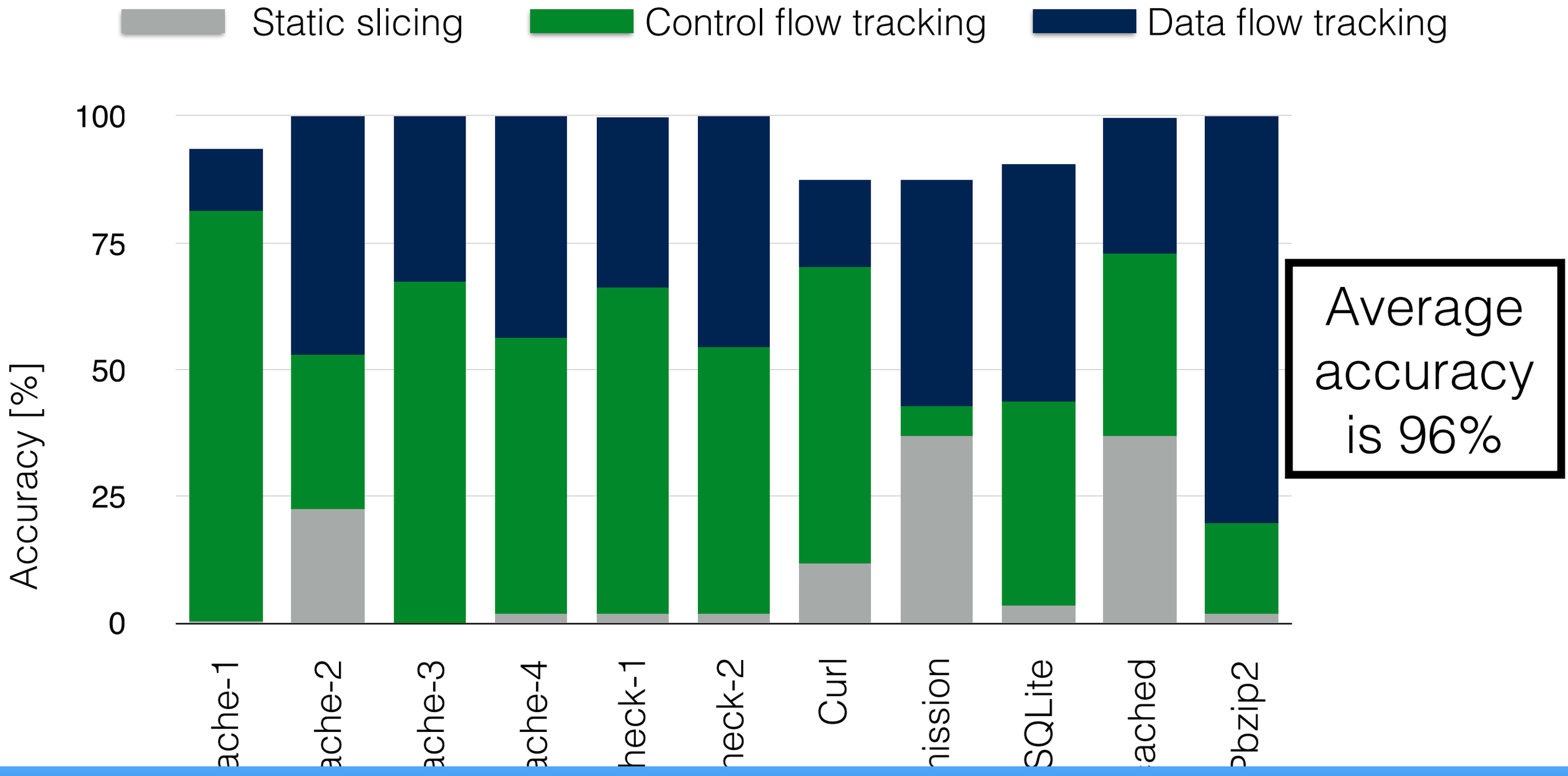
Accuracy


Static slicing
Control flow tracking
Data flow tracking
Accuracy [%]
100
75
50
25
0
Apache-1
Apache-2
Apache-3
Apache-4
Cppcheck-1
Cppcheck-2
Curl
Transmission
SQLite
memcached
Pbzip2

Accuracy

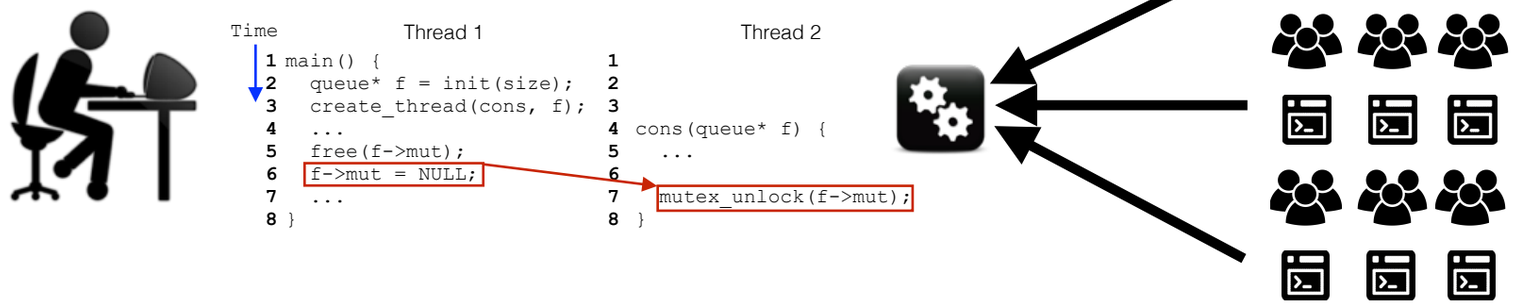

Static slicing
Control flow tracking
Data flow tracking
Accuracy [%]
100
75
50
25
0
Apache-1
Apache-2
Apache-3
Apache-4
Cppcheck-1
Cppcheck-2
Curl
Transmission
SQLite
memcached
Pbzip2
Average accuracy is 96%

Accuracy



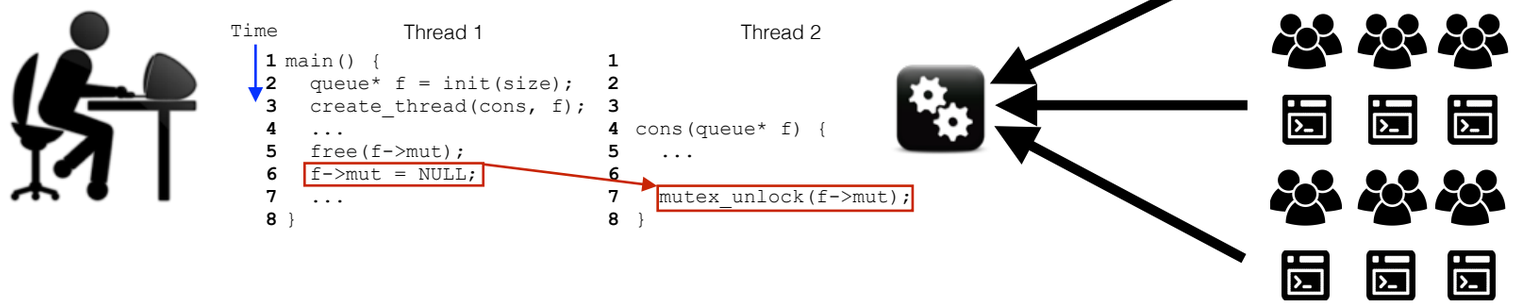
Each technique is needed for accuracy

Conclusion



- Failure sketching
  - *Combination of static and dynamic program analysis*
  - *Failure sketches are summaries explaining failure root causes*
  - *Accurate, efficient, improves developer productivity*

Conclusion



- Failure sketching
  - *Combination of static and dynamic program analysis*
  - *Failure sketches are summaries explaining failure root causes*
  - *Accurate, efficient, improves developer productivity*

http://dslab.epfl.ch/proj/gist

Baris Ben Cristiano Gilles George