SAFE SOFTWARE UPDATES VIA MULTI-VERSION EXECUTION

PETR HOSEK
CRISTIAN CADAR

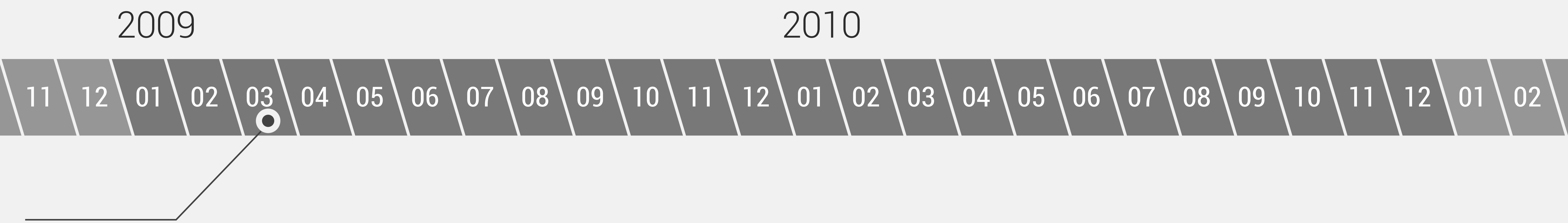
Imperial College London

Petr Hosek is a recipient of the **Google European Fellowship in Software Engineering** and this research is supported in part by this Google Fellowship

2009 2010

11 12 01 02 03 04 05 06 07 08 09 10 11 12 01 02 03 04 05 06 07 08 09 10 11 12 01 02

LIGHTTPD
fly light.




```
for (h = 0, i = 0; i < etag->used; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

LIGHTTPD fly light.


```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

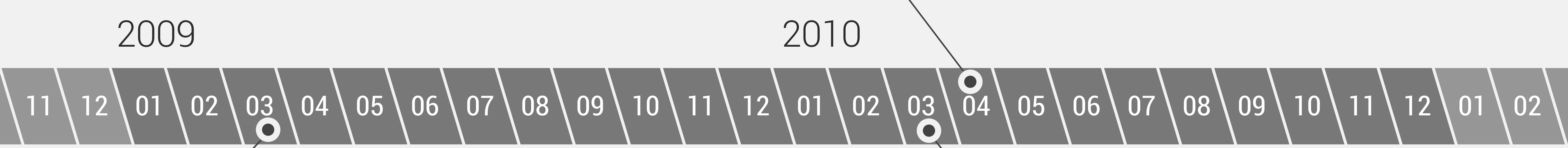

```c
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

Bug diagnosed in issue tracker

```
etag_mutate(con->physical.etag, srv->tmp_buf);
```

File (re)compression in `mod_compress_physical`



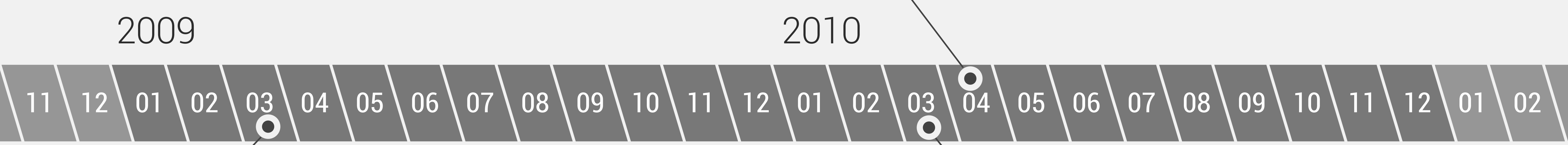

```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

Bug diagnosed in issue tracker

```c
if (use_etag) {
    etag_mutate(con->physical.etag, srv->tmp_buf);
}
```

File (re)compression in `mod_compress_physical`



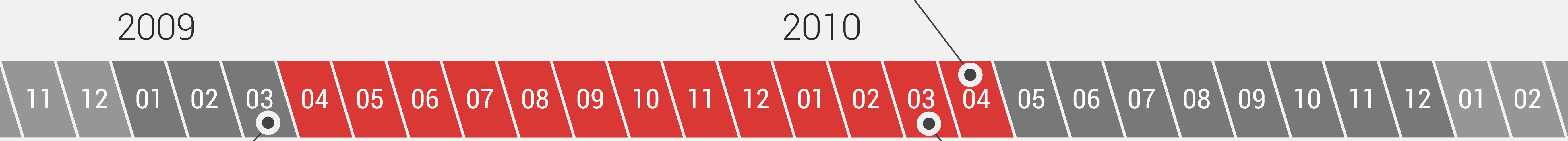

```c
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

Bug diagnosed in issue tracker

```
if (use_etag) {
    etag_mutate(con->physical.etag, srv->tmp_buf);
}
```

File (re)compression in `mod_compress_physical`




```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`

Bug diagnosed in issue tracker

A year ago in a city far far away...

Introducing novel approach for improving software updates:

- Multi-version execution based approach
- Relying on abundance of resources to improve reliability
- Run the new version in parallel with the existing one
- Synchronise the execution of the versions
- Use output of correctly executing version

LIGHTTPD **1.4.22**

LIGHTTPD **1.4.23**

Synchronisation and fail-recovery mechanism

LIGHTTPD **1.4.22**

LIGHTTPD **1.4.23**

Synchronisation

Compare individual **system calls** and their arguments

Synchronisation and fail-recovery mechanism

Synchronisation
Compare individual **system calls** and their arguments

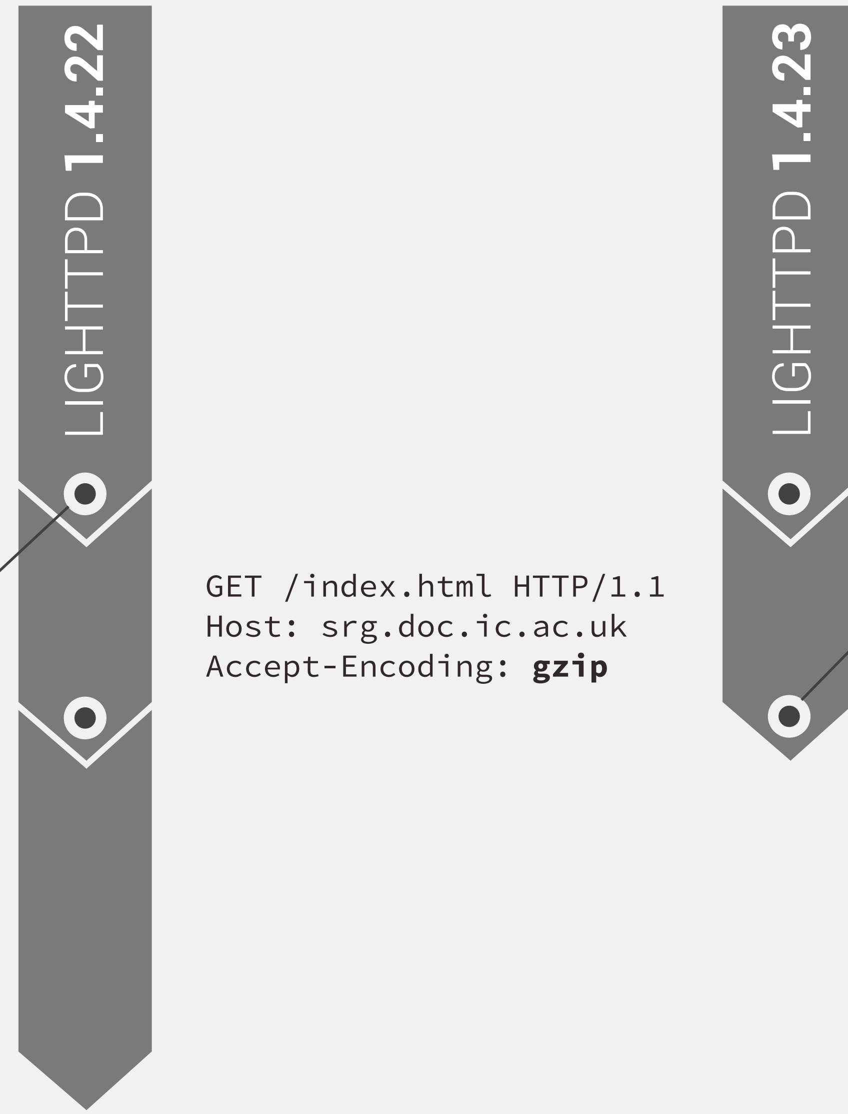
Synchronisation and fail-recovery mechanism

Synchronisation
Compare individual **system calls** and their arguments


```
GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip
```

Checkpointing
Use `clone` to take a snapshot of a process

Synchronisation and fail-recovery mechanism



Synchronisation

Compare individual **system calls** and their arguments

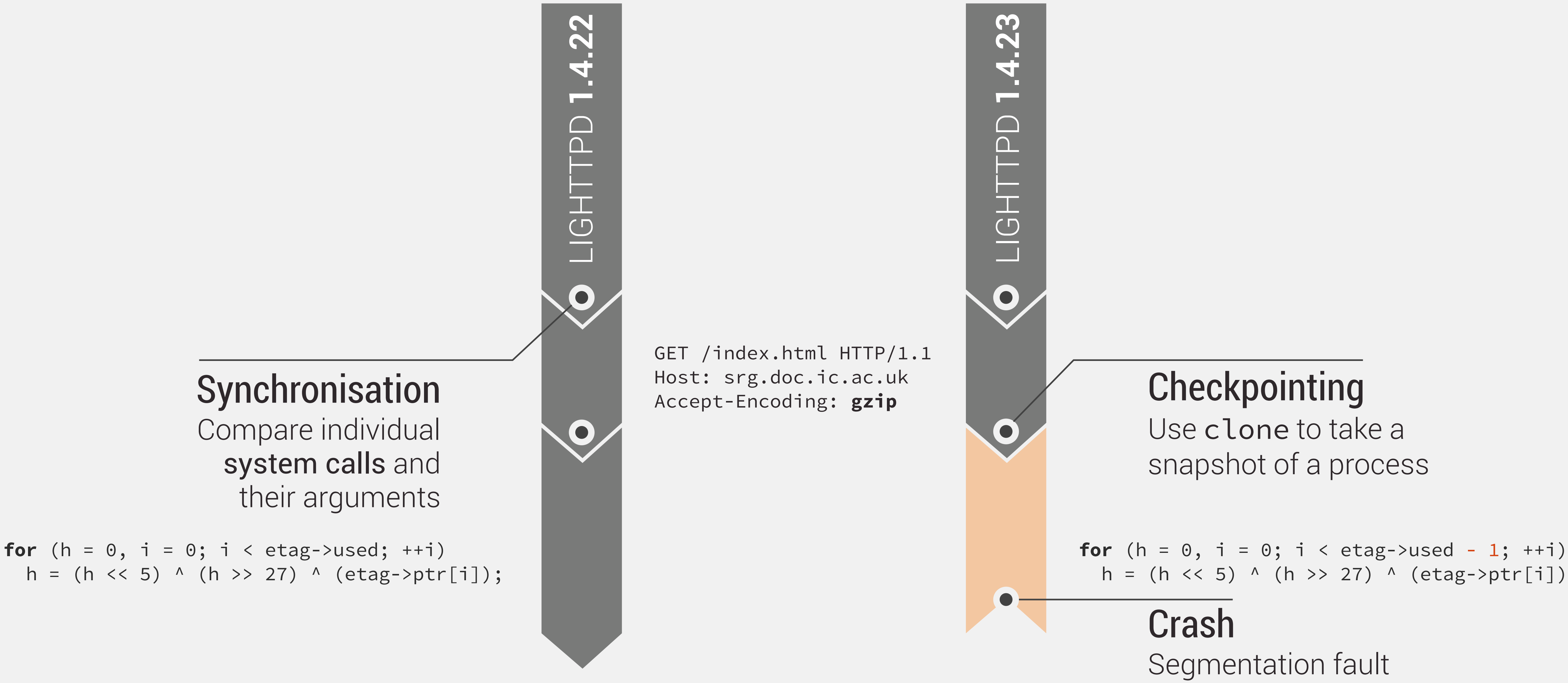
```
for (h = 0, i = 0; i < etag->used; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

```
GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip
```

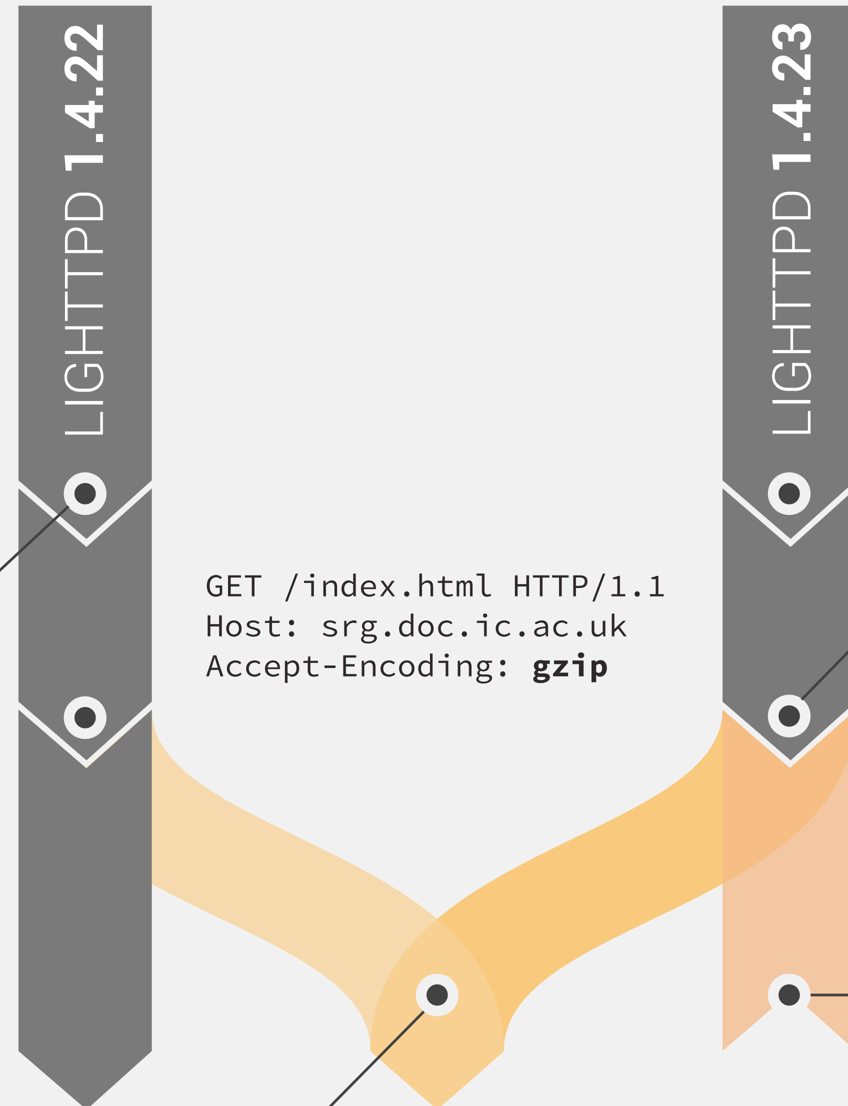
Checkpointing

Use `clone` to take a snapshot of a process

Synchronisation and fail-recovery mechanism



Synchronisation and fail-recovery mechanism




```
GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip
```

Synchronisation
Compare individual **system calls** and their arguments

```
for (h = 0, i = 0; i < etag->used; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

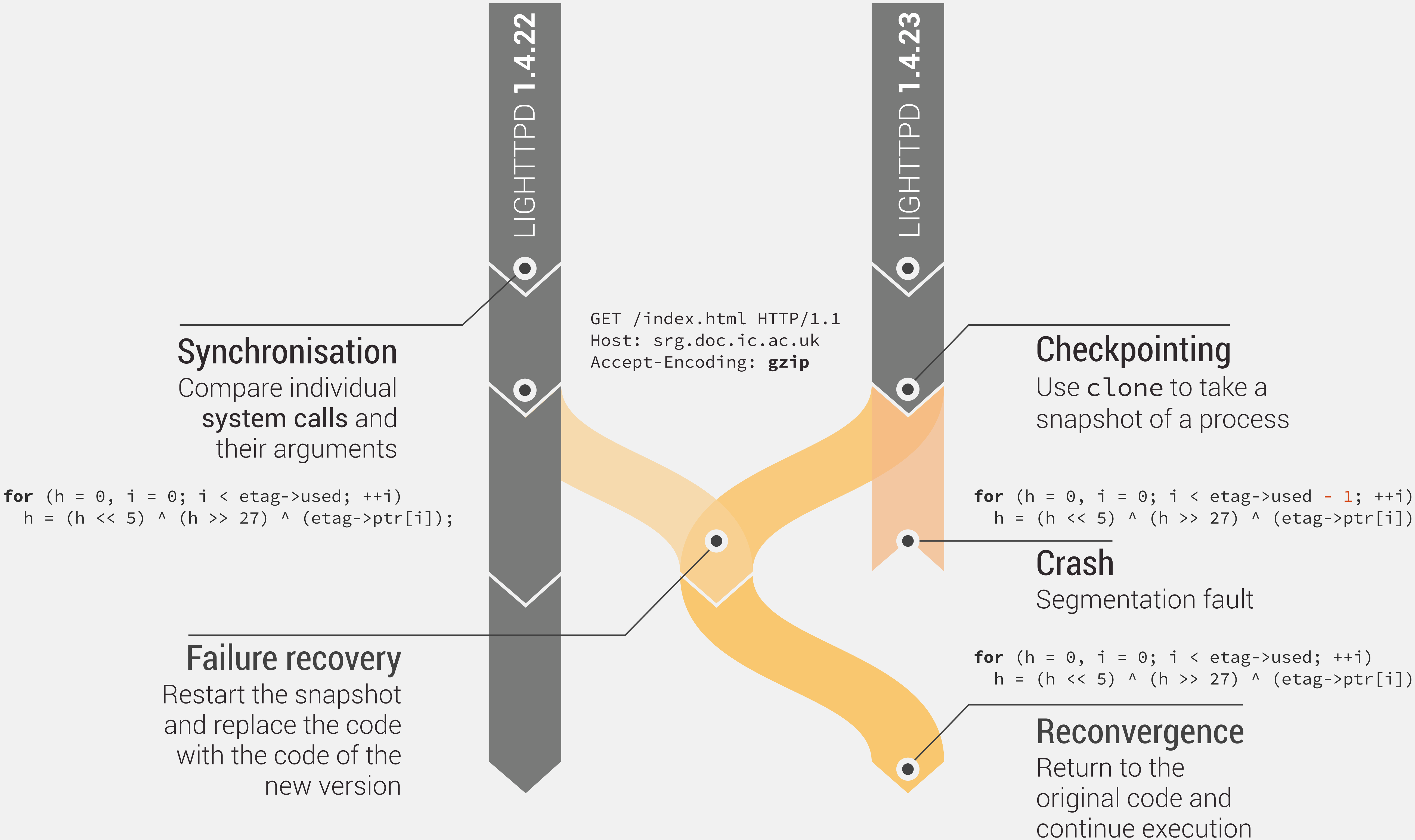
Checkpointing
Use `clone` to take a snapshot of a process

```
for (h = 0, i = 0; i < etag->used - 1; ++i)
  h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

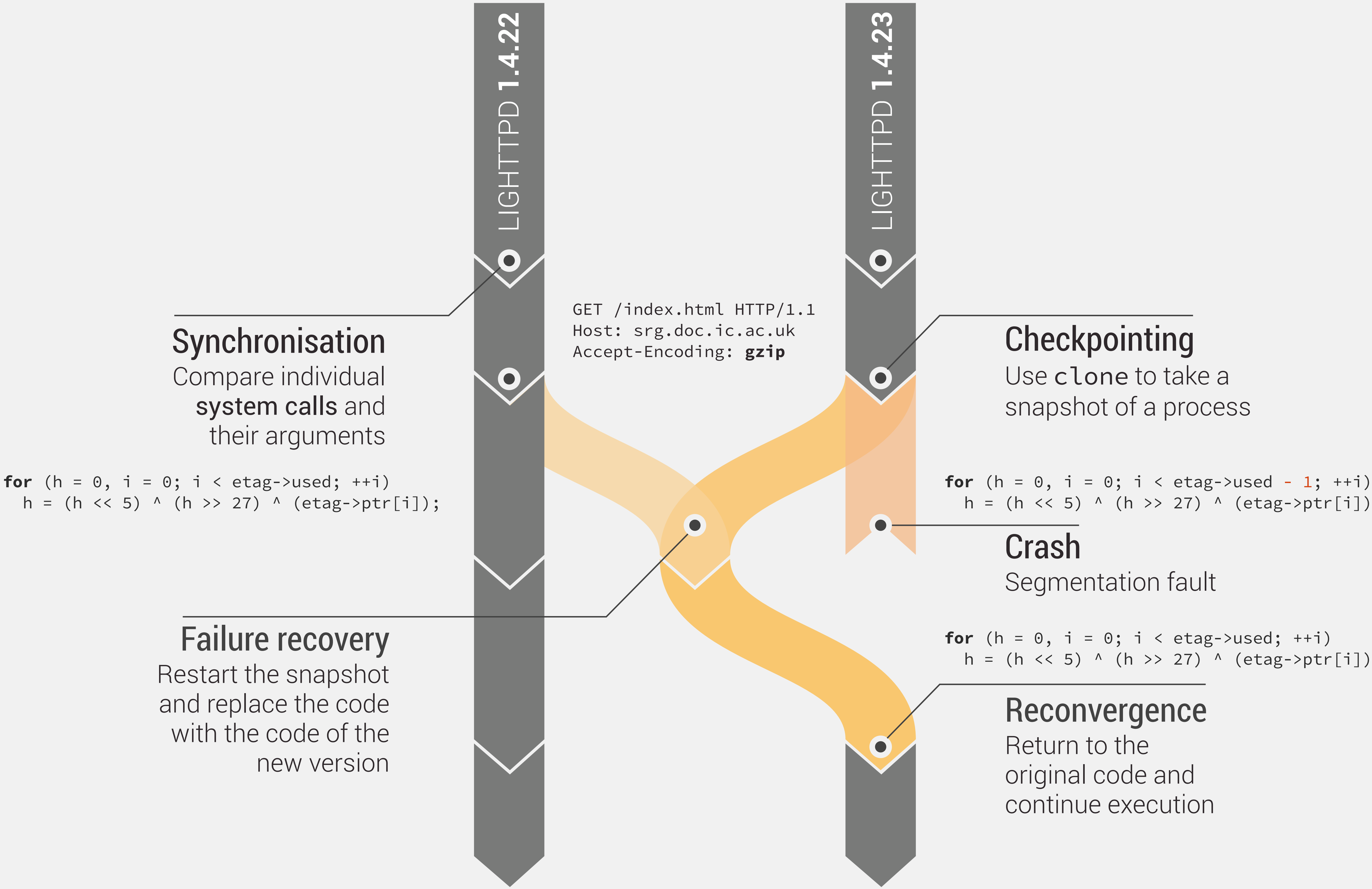
Crash
Segmentation fault

Failure recovery
Restart the snapshot and replace the code with the code of the new version

Synchronisation and fail-recovery mechanism



Synchronisation and fail-recovery mechanism



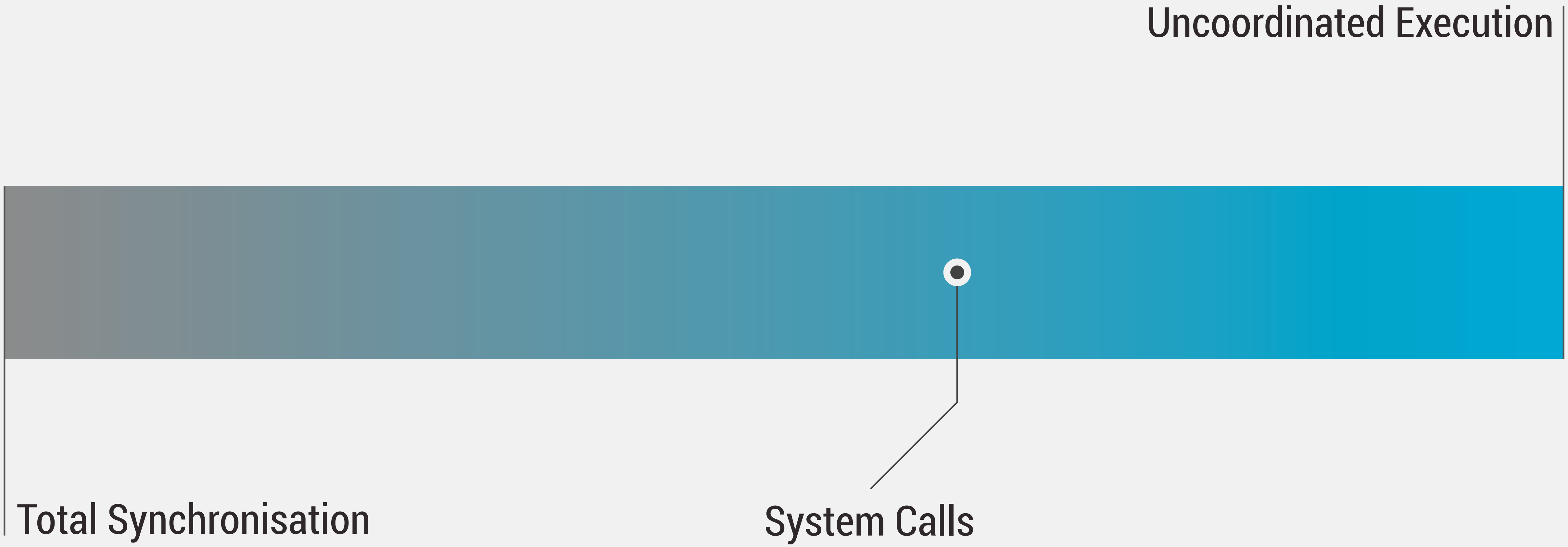
Synchronisation and fail-recovery mechanism

Assumptions

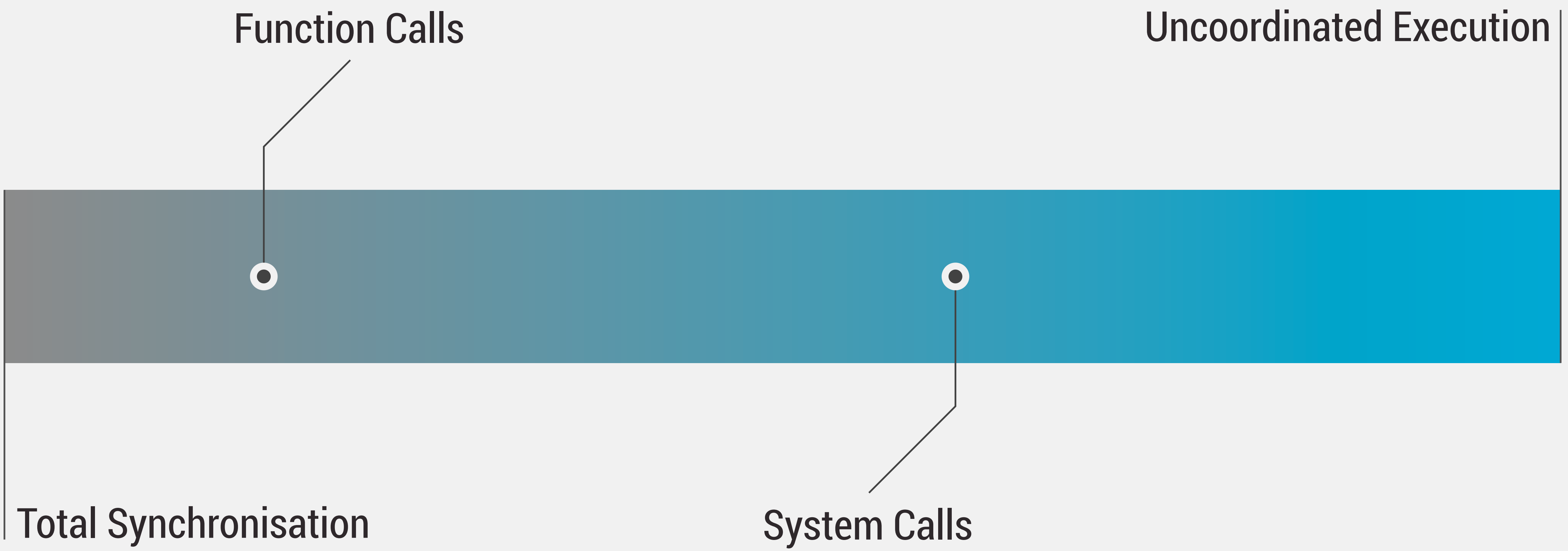
Recovery considered successful if versions exhibit the same externally observable behaviour after recovery:

- Assumes small bug *propagation distance*
- Crashes are the only type of observable divergences
- The non-crashing version used as an *oracle*
- If unrecoverable, continue with the non-crashing version

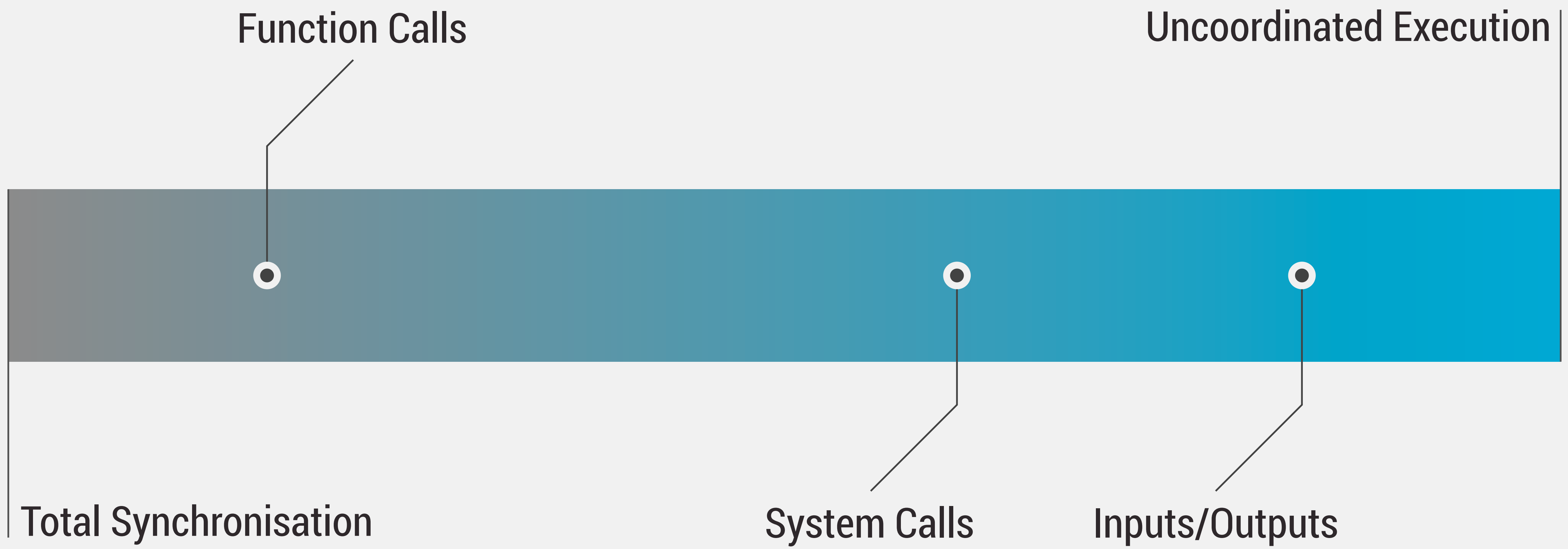
Synchronisation possible at multiple levels of abstraction



Synchronisation possible at multiple levels of abstraction



Synchronisation possible at multiple levels of abstraction



Synchronisation possible at multiple levels of abstraction

System calls define **external behaviour**

VERSION 1	VERSION 2

VERSION 1

```
void fib(int n)
{
    int f[n+1];
    f[1] = f[2] = 1;
    for (int i = 3; i <= n; ++i)
        f[i] = f[i-1] + f[i-2];

    printf("%d\n", f[n]);
}
```

VERSION 2

```
void fib(int n)
{
    int a = 1, b = 1;
    for (int i = 3; i <= n; ++i) {
        int c = a + b;
        a = b, b = c;
    }
    printf("%d\n", b);
}
```

```
int main(int argc, char **argv)
{
    fib(5);
    fib(6);
}
```

Example testing code
Tested with both implementations

System calls define **external behaviour**

VERSION 1	VERSION 2
`write(1, “5\n”, 2) = 2` `write(1, “8\n”, 2) = 2`	`write(1, “5\n”, 2) = 2` `write(1, “8\n”, 2) = 2`

Snippet of system call trace
Obtained using the *strace* tool

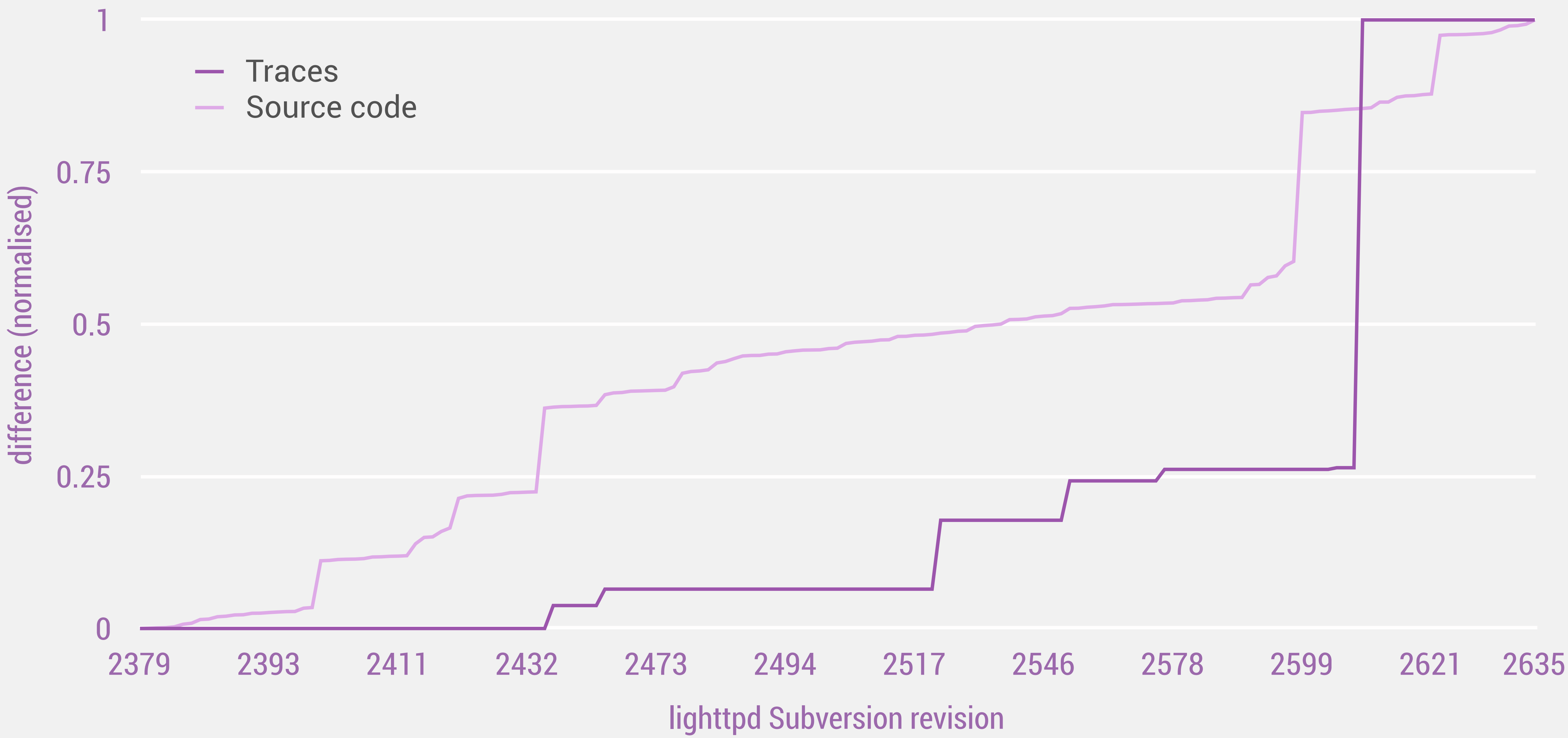
Snippet of system call trace
Obtained using the *strace* tool

```
int main(int argc, char **argv)
{
    fib(5);
    fib(6);
}
```

Example testing code
Tested with both implementations

External behaviour **evolves sporadically**

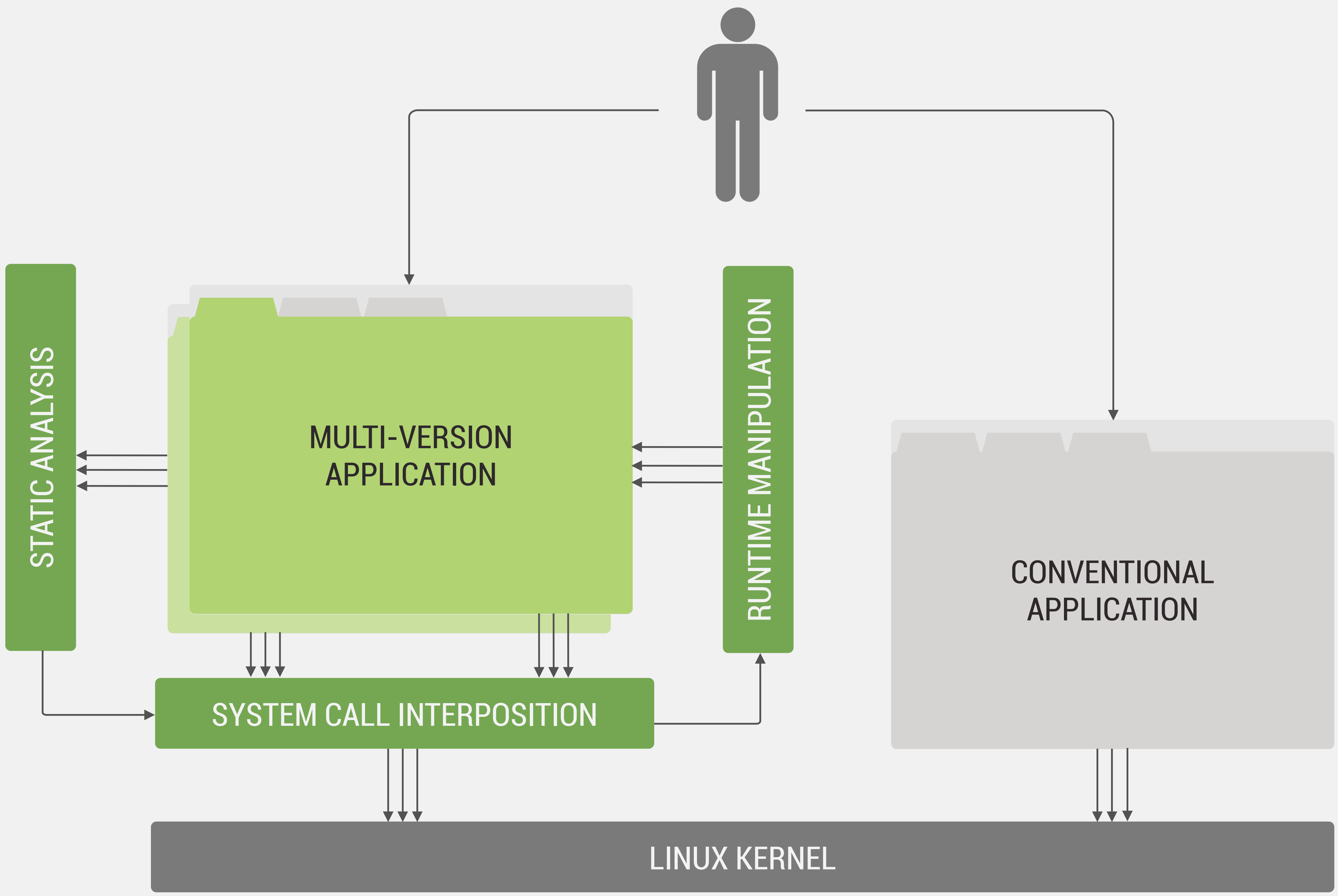
95% of revisions introduce *no change*



Measured using *lighttpd* regression suite on 164 revisions

Taken on Linux kernel 2.6.40 and glibc 2.14 using *strace* tool and custom post-processing (details in the paper)

Mx architecture



Implementation for x86 and x86-64 Linux

Combines binary static analysis, lightweight checkpointing and runtime code patching

Completely transparent, runs on unmodified binaries

Runs two versions with small differences in behaviour

Focus on application crashes and recovery

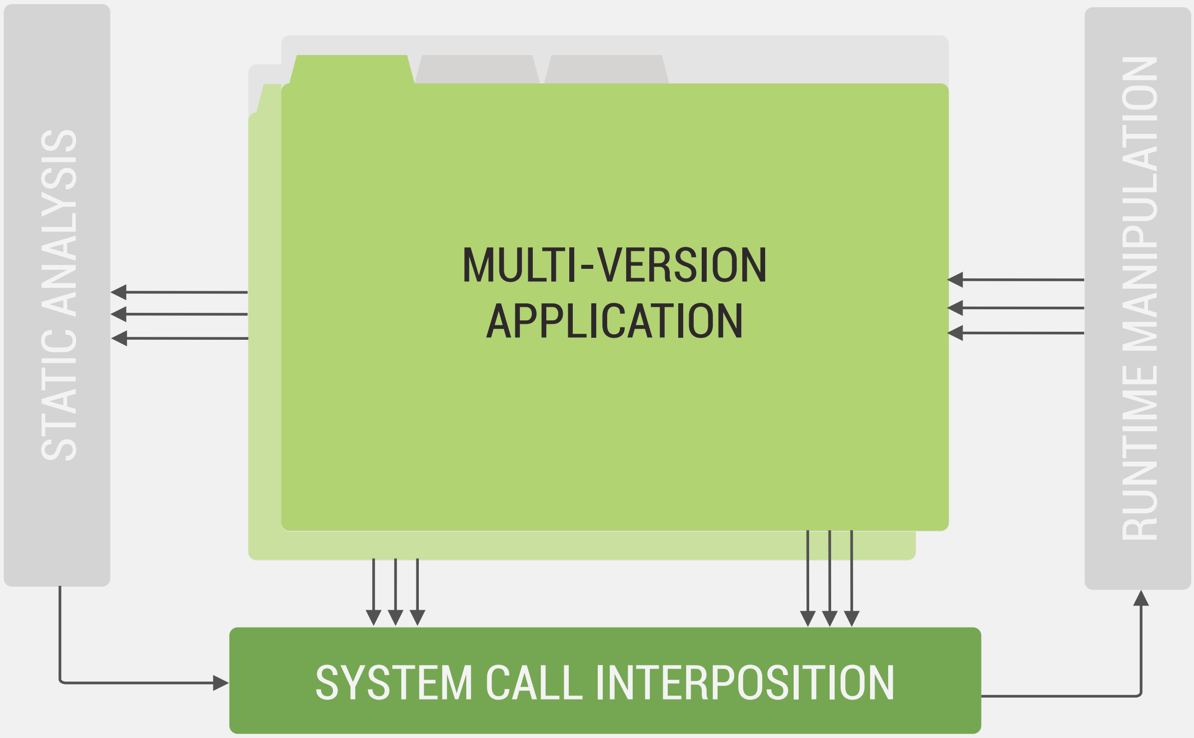
Multi-eXecution Monitor

Execute and monitor multi-version applications:

Intercepting system calls (via `ptrace` interface)

Semantically comparing system calls arguments

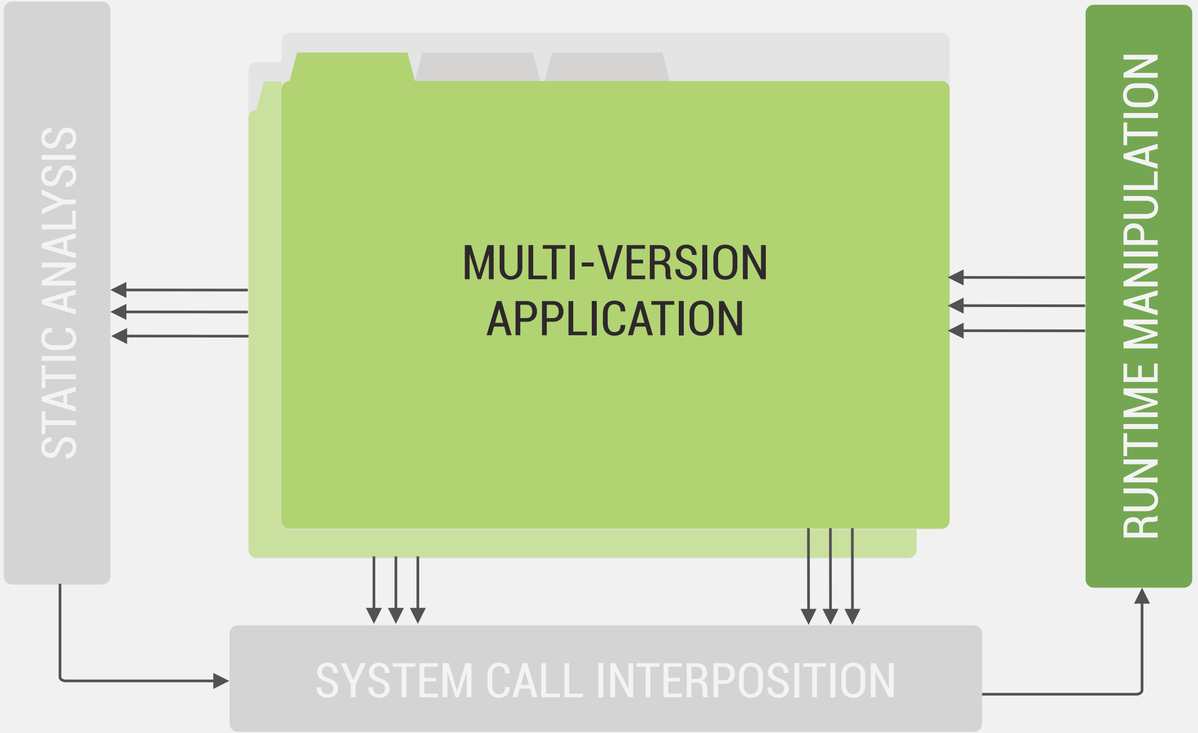
Environment virtualisation (e.g. files and sockets)



Runtime Execution Manipulator

Runtime code patching and fault recovery:

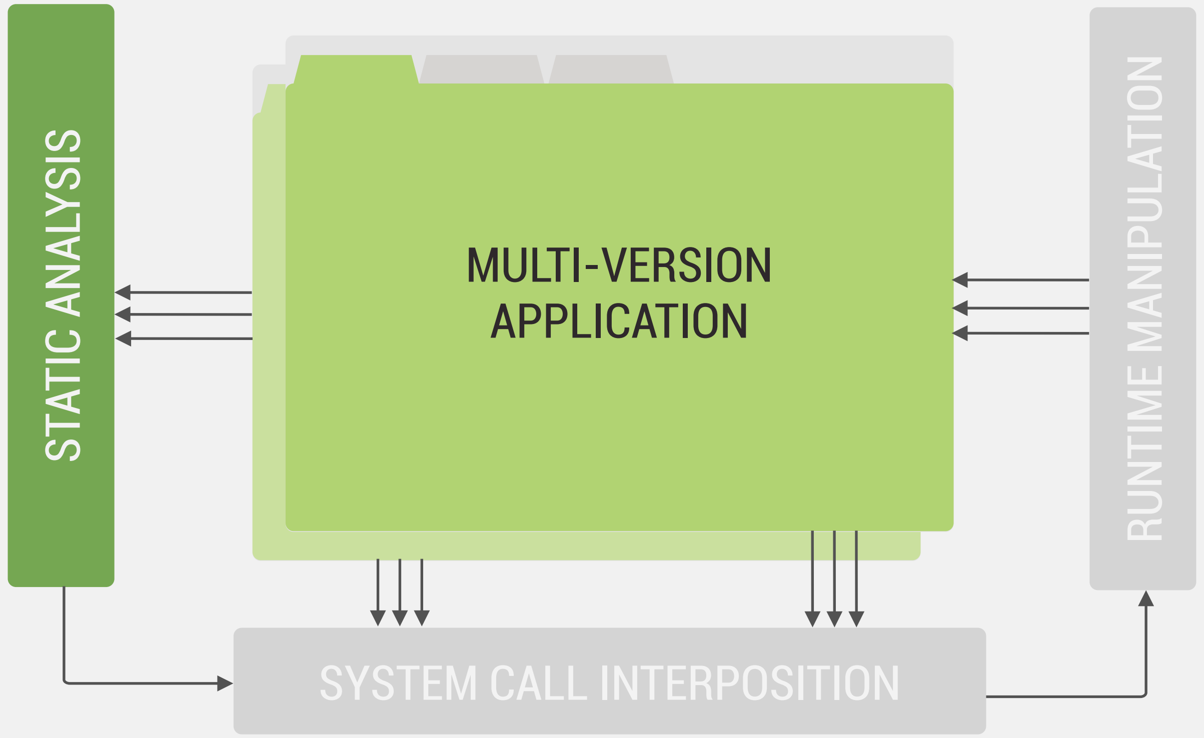
- OS-level checkpointing (using `clone` syscall)
- Runtime stack rewriting (`libunwind`)
- Breakpoint insertion and handling

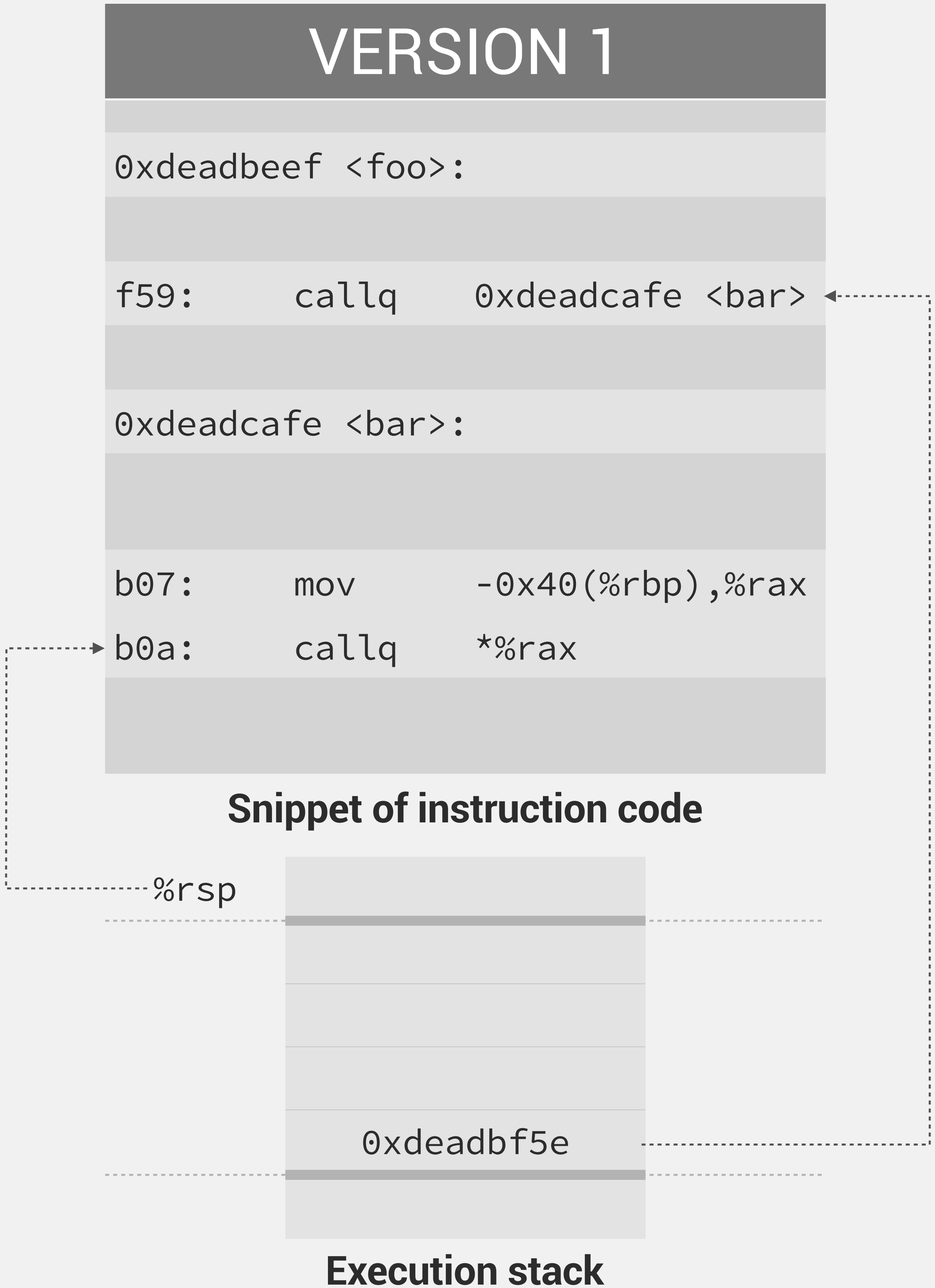


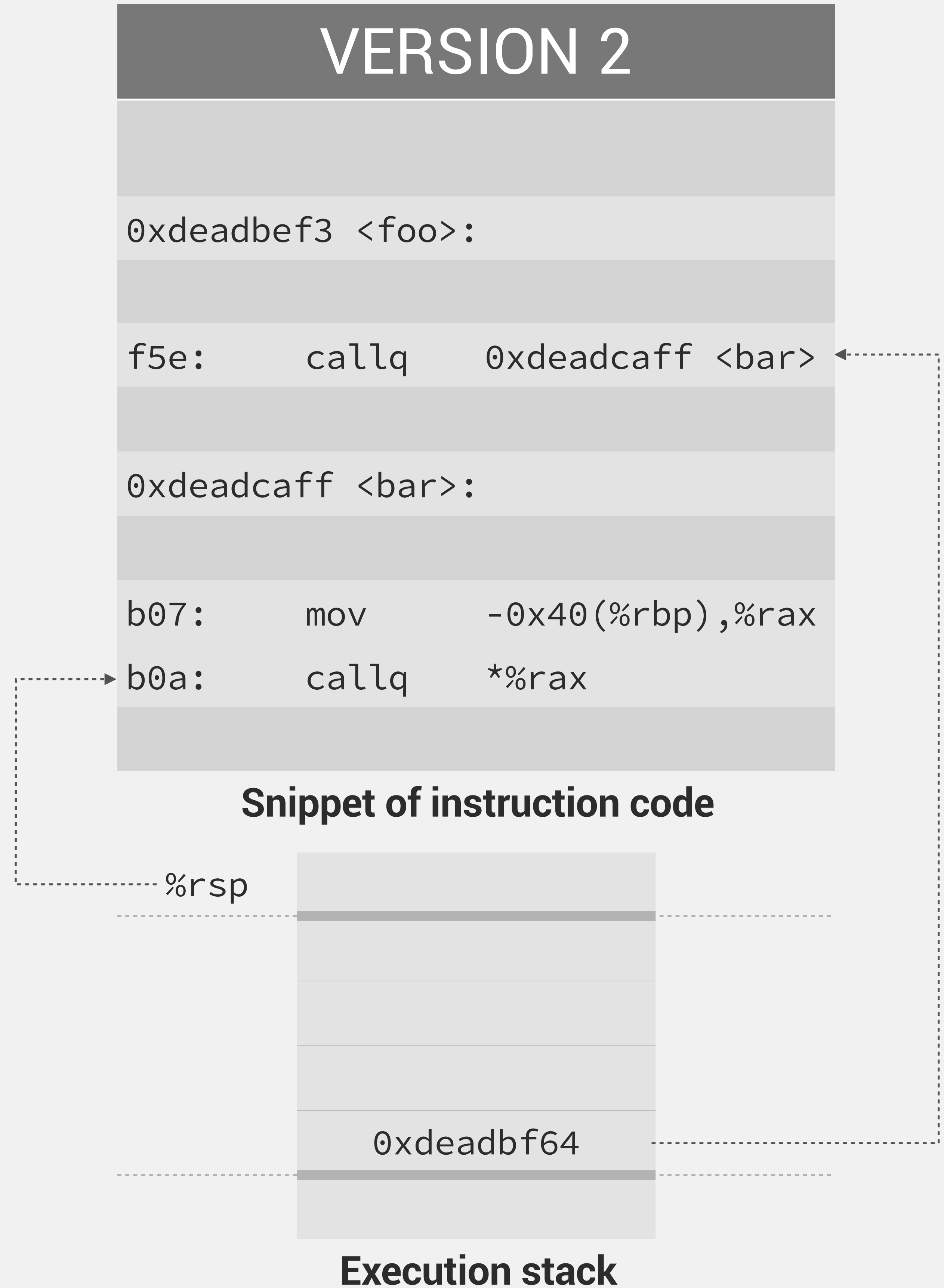
Static Executable Analyser

Create various mappings between the two version binaries:

- Extracting function symbols from binaries (`libbfd`)
- Machine code disassembling and analysis (`libopcodes`)
- Binary call graph reconstruction and matching




VERSION 1
0xdeadbeef <foo>:
f59: callq 0xdeadcafe <bar>
0xdeadcafe <bar>:
b07: mov -0x40(%rbp),%rax
b0a: callq *%rax
Snippet of instruction code
%rsp
0xdeadbf5e
Execution stack


VERSION 2
0xdeadbef3 <foo>:
f5e: callq 0xdeadcaff <bar>
0xdeadcaff <bar>:
b07: mov -0x40(%rbp),%rax
b0a: callq *%rax
Snippet of instruction code
%rsp
0xdeadbf64
Execution stack

VERSION 1

```
0xdeadbeef <foo>:

f59:      callq    0xdeadcafe <bar>

0xdeadcafe <bar>:

b07:      mov      -0x40(%rbp),%rax
b0a:      callq    *%rax
```

Snippet of instruction code

```
%rsp

0xdeadbf5e
```

Execution stack

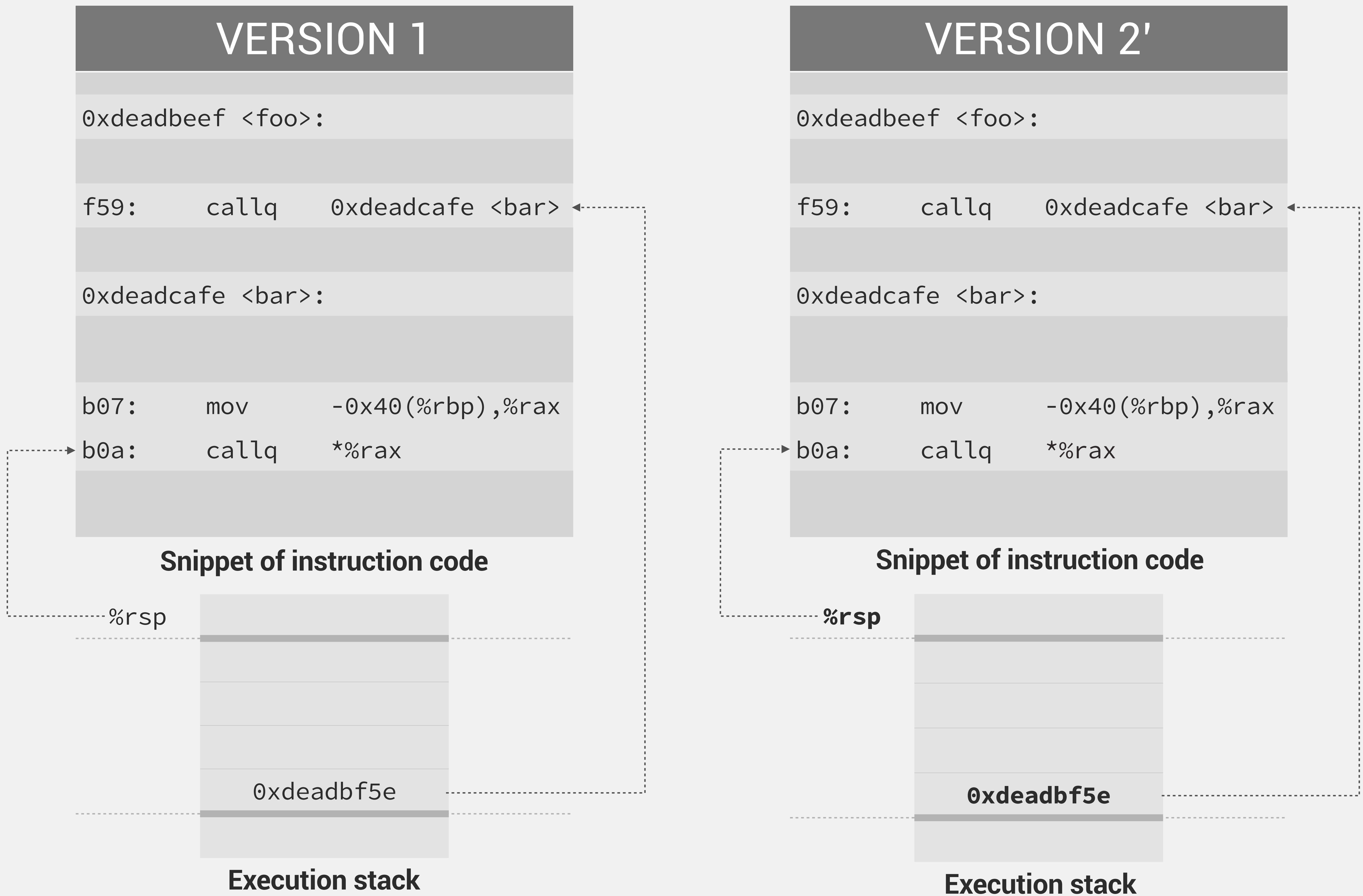
VERSION 2'

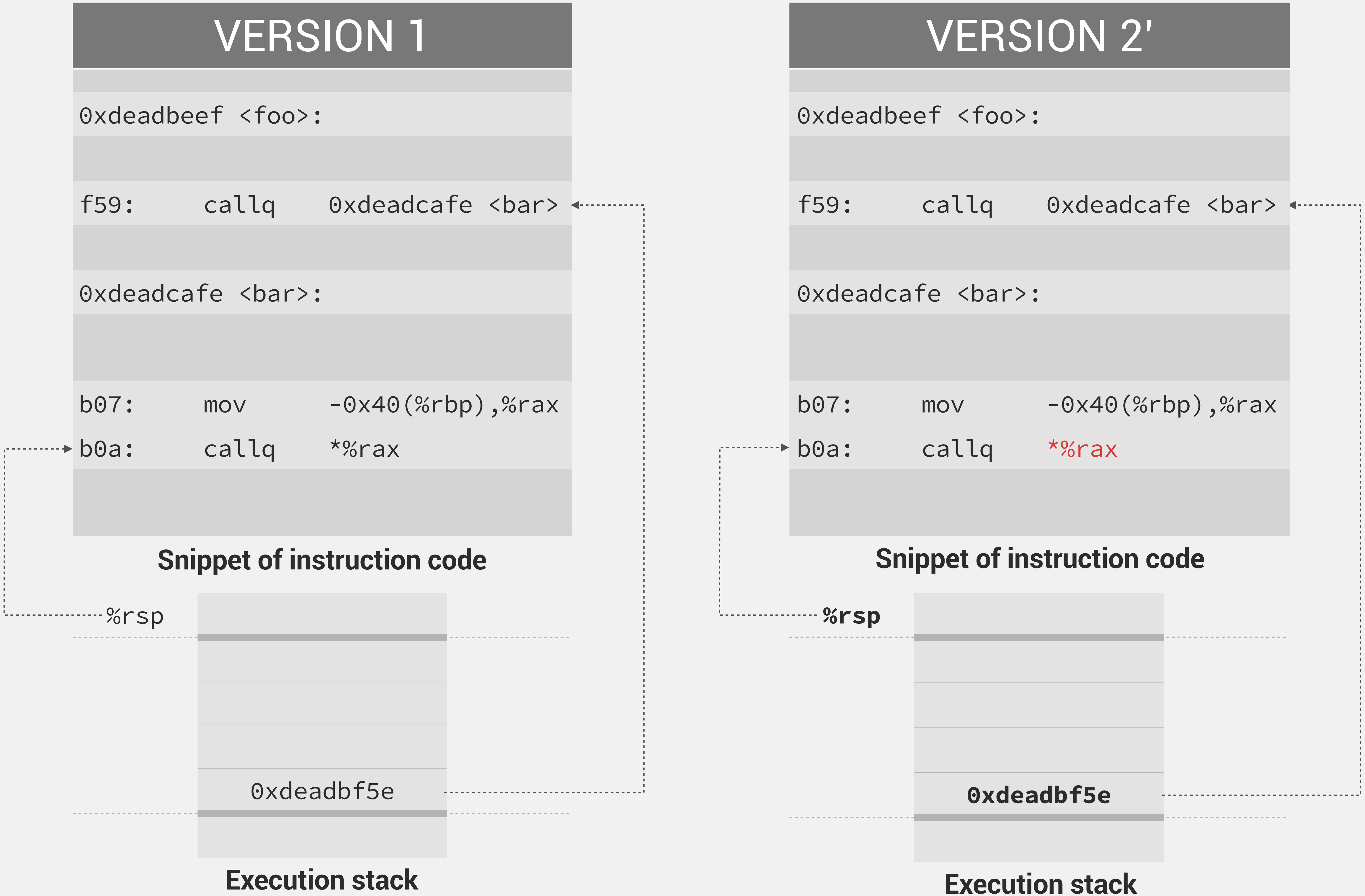
```
0xdeadbeef <foo>:

f59:      callq    0xdeadcafe <bar>

0xdeadcafe <bar>:

b07:      mov      -0x40(%rbp),%rax
b0a:      callq    *%rax
```

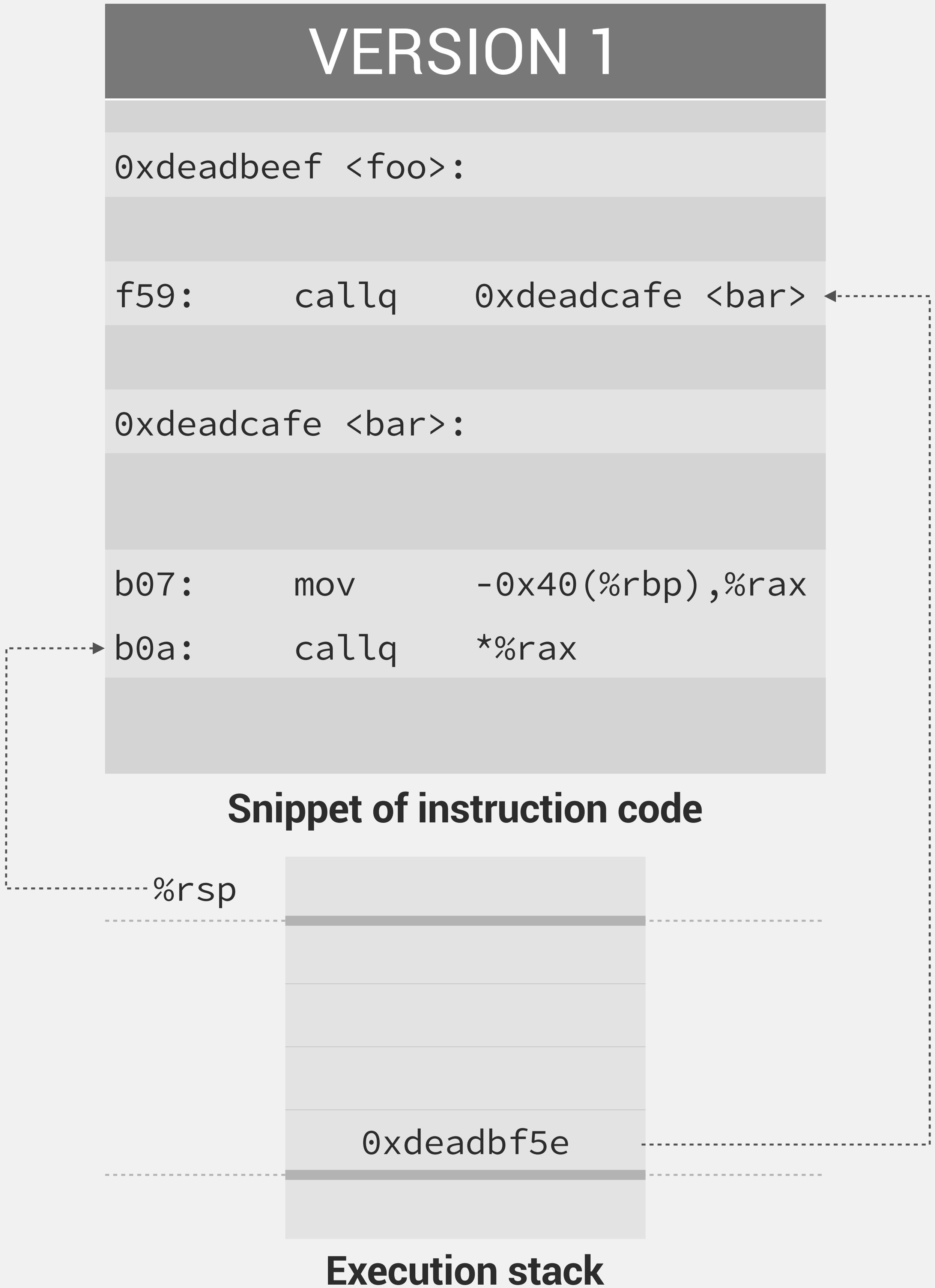
Snippet of instruction code

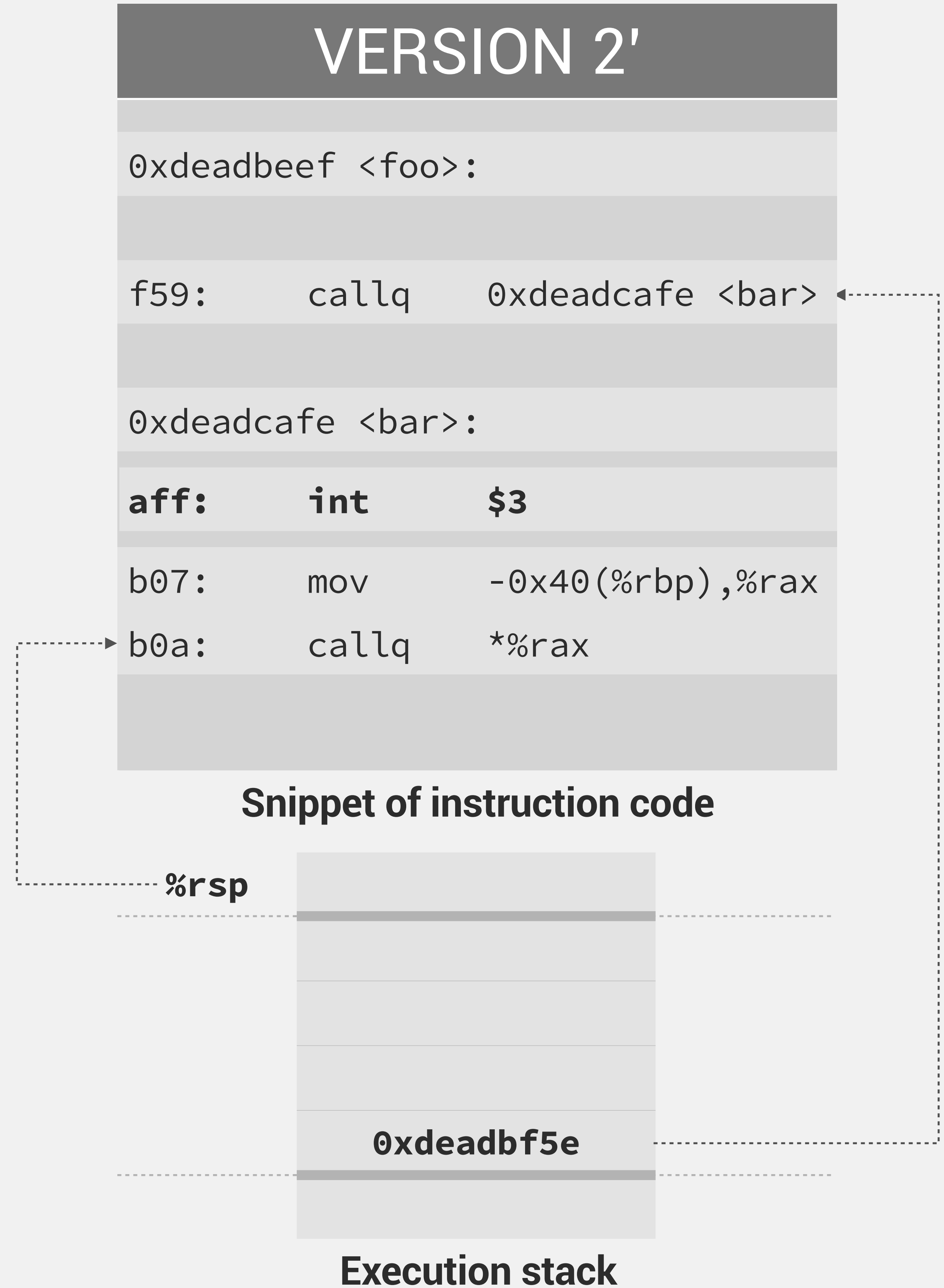
```
%rsp

0xdeadbf64
```

Execution stack


VERSION 1
0xdeadbeef <foo>:
f59: callq 0xdeadcafe <bar>
0xdeadcafe <bar>:
b07: mov -0x40(%rbp),%rax
b0a: callq *%rax
Snippet of instruction code
%rsp
0xdeadbf5e
Execution stack
VERSION 2'
0xdeadbeef <foo>:
f59: callq 0xdeadcafe <bar>
0xdeadcafe <bar>:
b07: mov -0x40(%rbp),%rax
b0a: callq *%rax
Snippet of instruction code
%rsp
0xdeadbf5e
Execution stack


VERSION 1
0xdeadbeef <foo>:
f59: callq 0xdeadcafe <bar>
0xdeadcafe <bar>:
b07: mov -0x40(%rbp),%rax
b0a: callq *%rax
Snippet of instruction code
%rsp
0xdeadbf5e
Execution stack
VERSION 2′
0xdeadbeef <foo>:
f59: callq 0xdeadcafe <bar>
0xdeadcafe <bar>:
b07: mov -0x40(%rbp),%rax
b0a: callq *%rax
Snippet of instruction code
%rsp
0xdeadbf5e
Execution stack


VERSION 1
0xdeadbeef <foo>:
f59: callq 0xdeadcafe <bar>
0xdeadcafe <bar>:
b07: mov -0x40(%rbp),%rax
b0a: callq *%rax
Snippet of instruction code
%rsp
0xdeadbf5e
Execution stack


VERSION 2'
0xdeadbeef <foo>:
f59: callq 0xdeadcafe <bar>
0xdeadcafe <bar>:
aff: int $3
b07: mov -0x40(%rbp),%rax
b0a: callq *%rax
Snippet of instruction code
%rsp
0xdeadbf5e
Execution stack

Suitable for type of changes and applications:

- Changes which do not affect memory layout
  - *e.g., refactorings, security patches*
- Applications which provide synchronisation points
  - *e.g., servers structured around the main dispatch loop*
- Where reliability is more important than performance
  - *e.g., interactive apps, some server scenarios*

Survived a number of crash bugs
in several popular server applications

redis

In-memory NoSQL database

```
robj *o = lookupKeyRead(c->db, c->argv[1]);
if (o == NULL) {
    addReplySds(c,sdscatprintf(sdsempty(),
      "*%d\r\n",c->argc-2));
    for (i = 2; i < c->argc; i++) {
        addReply(c,shared.nullbulk);
    }
    return;
} else {
    if (o->type != REDIS_HASH) {
        addReply(c,shared.wrongtypeerr);
        return;
    }
}
addReplySds(c,sdscatprintf(sdsempty(),
  "*%d\r\n",c->argc-2));
```

Redis regression bug #344 introduced during refactoring
HMGET command implementation in `hmgetCommand` function

Survived a number of crash bugs
in several popular server applications

redis

In-memory NoSQL database

```
robj *o, *value;
o = lookupKeyRead(c->db,c->argv[1]);
if (o != NULL && o->type != REDIS_HASH) {
    addReply(c,shared.wrongtypeerr);
    return;
}
addReplySds(c,sdscatprintf(sdsempty(),
  "*%d\r\n",c->argc-2));
for (i = 2; i < c->argc; i++) {
    if (o != NULL && (value =
          hashGet(o,c->argv[i])) != NULL) {
        addReplyBulk(c,value);
        decrRefCount(value);
    } else {
        addReply(c,shared.nullbulk);
    }
}
```

Missing `return` statement

Redis regression bug #344 introduced during refactoring
HMGET command implementation in `hmgetCommand` function

Interactive applications:

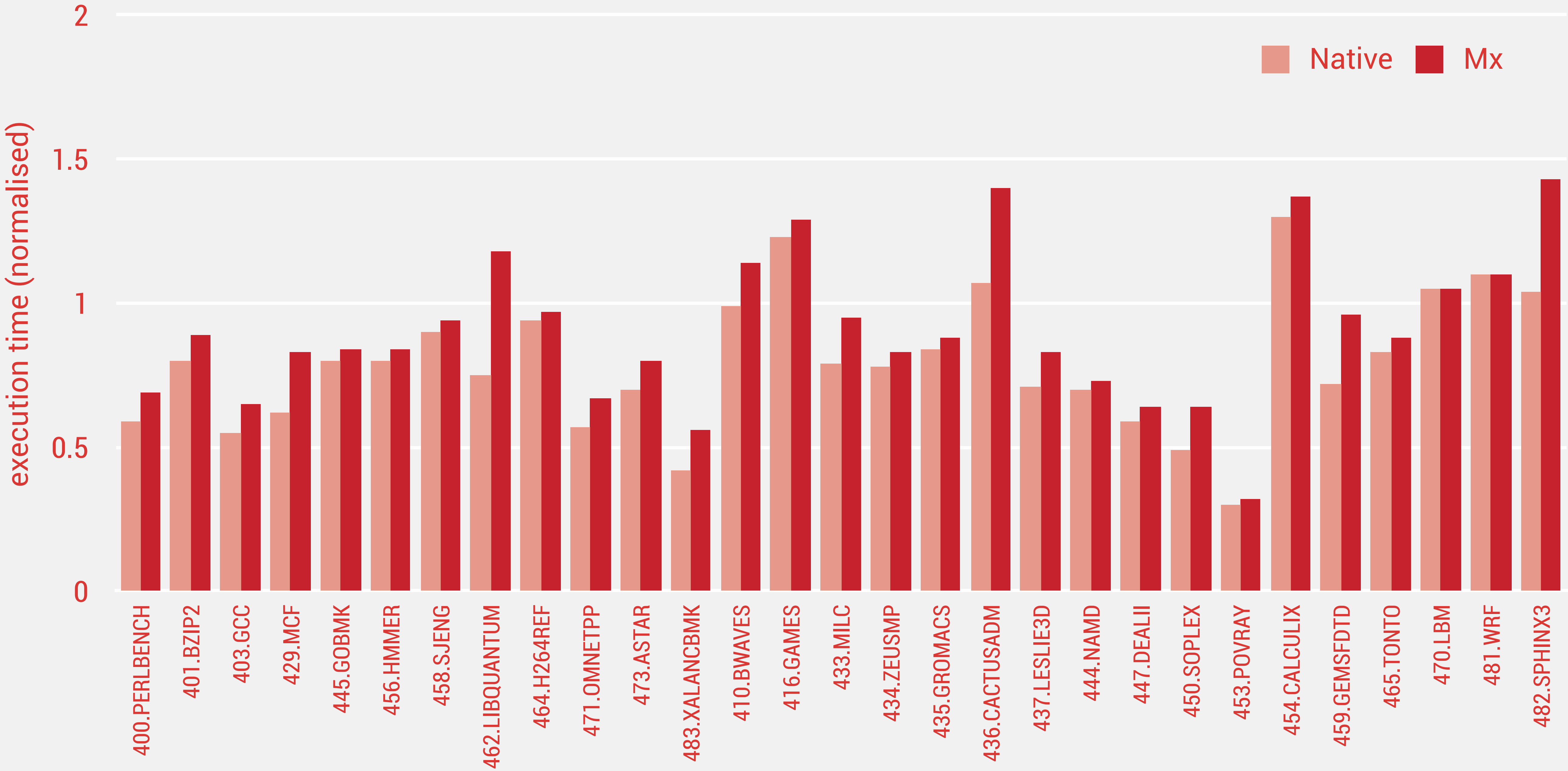
UTILITY	BUG	TIME SPAN
md5sum sha1sum	Buffer underflow	1,124 revs. (1 year 7 months)
mkdir mkfifo mknod	`NULL`-pointer dereference	2,937 revs. (over 4 years)
cut	Buffer overflow	1,201 revs. (2 years 3 months)

Server applications:

APPLICATION/ISSUE	BUG	TIME SPAN
lighttpd #2169	Loop index underflow	87 revs. (2 months 2 days)
lighttpd #2140	Off-by-one error	12 revs. (2 months 1 day)
redis #344	Missing `return` statement	27 revs. (6 days)

17.91% **overhead** on SPEC CPU2006

over single version despite **2x** utilisation cost



Measured using SPEC CPU2006 1.2

Taken on 3.50 GHz Intel Xeon E3 1280 with 16 GB of RAM, Linux kernel 3.1.9

Interactive applications:

UTILITY	INPUT SIZE	OVERHEAD
md5sum sha1sum	<1.25MB	<100ms (imperceptible)
mkdir mkfifo mknod	<115 nested directories	<100ms (imperceptible)
cut	<1.10MB	<100ms (imperceptible)

Measured using Coreutils 6.10
Taken on 3.50 GHz Intel Xeon E3 1280 with 16 GB of RAM, Linux kernel 3.1.9

Server applications:

APPLICATION	SCENARIO	OVERHEAD
lighttpd	localhost/network	2.60x – 3.49x
	distant networks	1.01x – 1.04x
redis	localhost/network	3.74x – 16.72x
	distant networks	1.00x – 1.05x

Measured using redis-benchmark and http_load
Taken on 3.50 GHz Intel Xeon E3 1280 with 16 GB of RAM, Linux kernel 3.1.9

“The New Mx”

Better performance overhead:

System call binary rewriting

Tolerance to system call divergences:

Event streaming

REDIS
`read(6, “PING\r\n”, 1024)`

Snippet of system call trace

Snippet of system call trace

REDIS

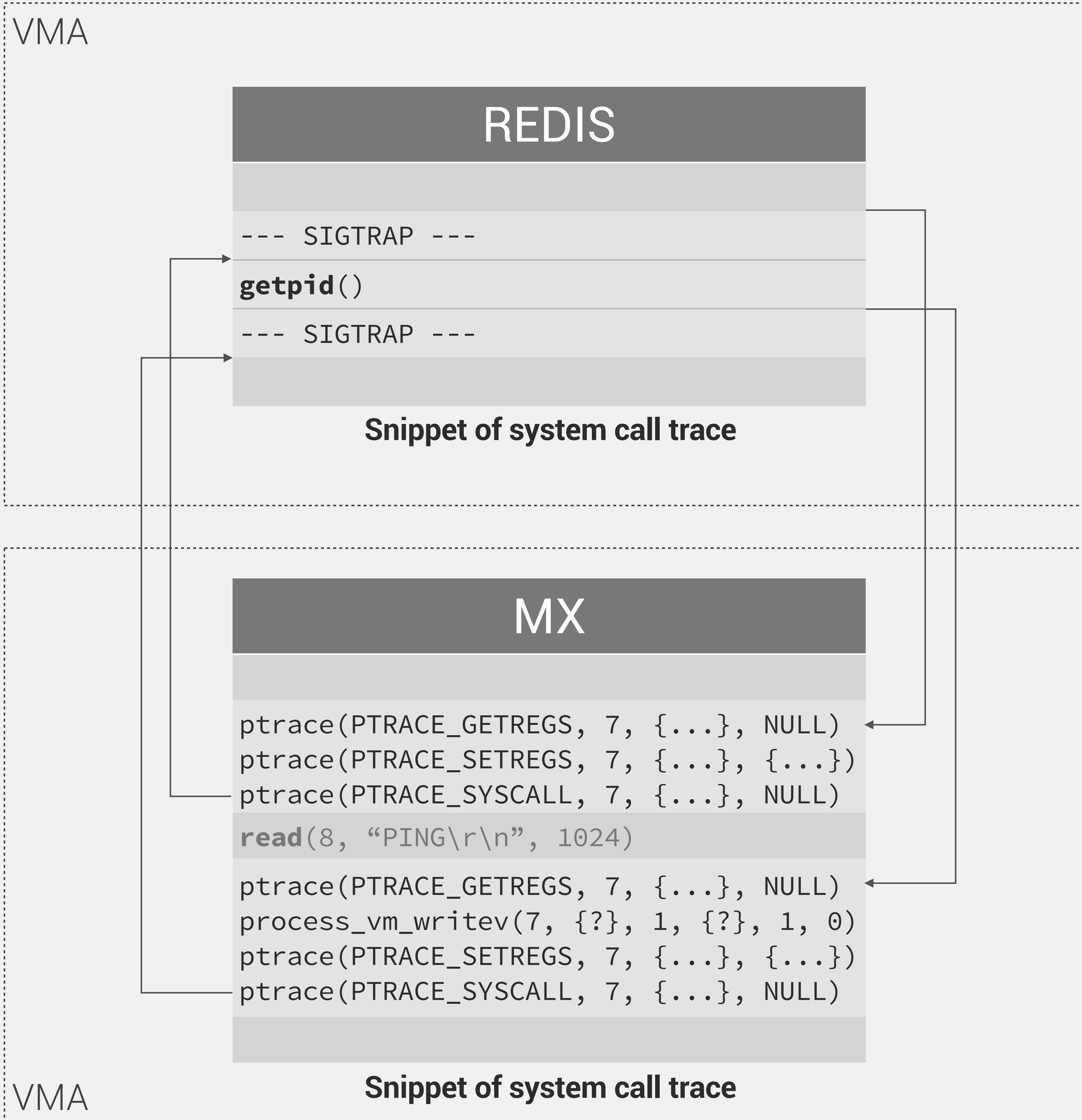
```
--- SIGTRAP ---
getpid()
--- SIGTRAP ---
```

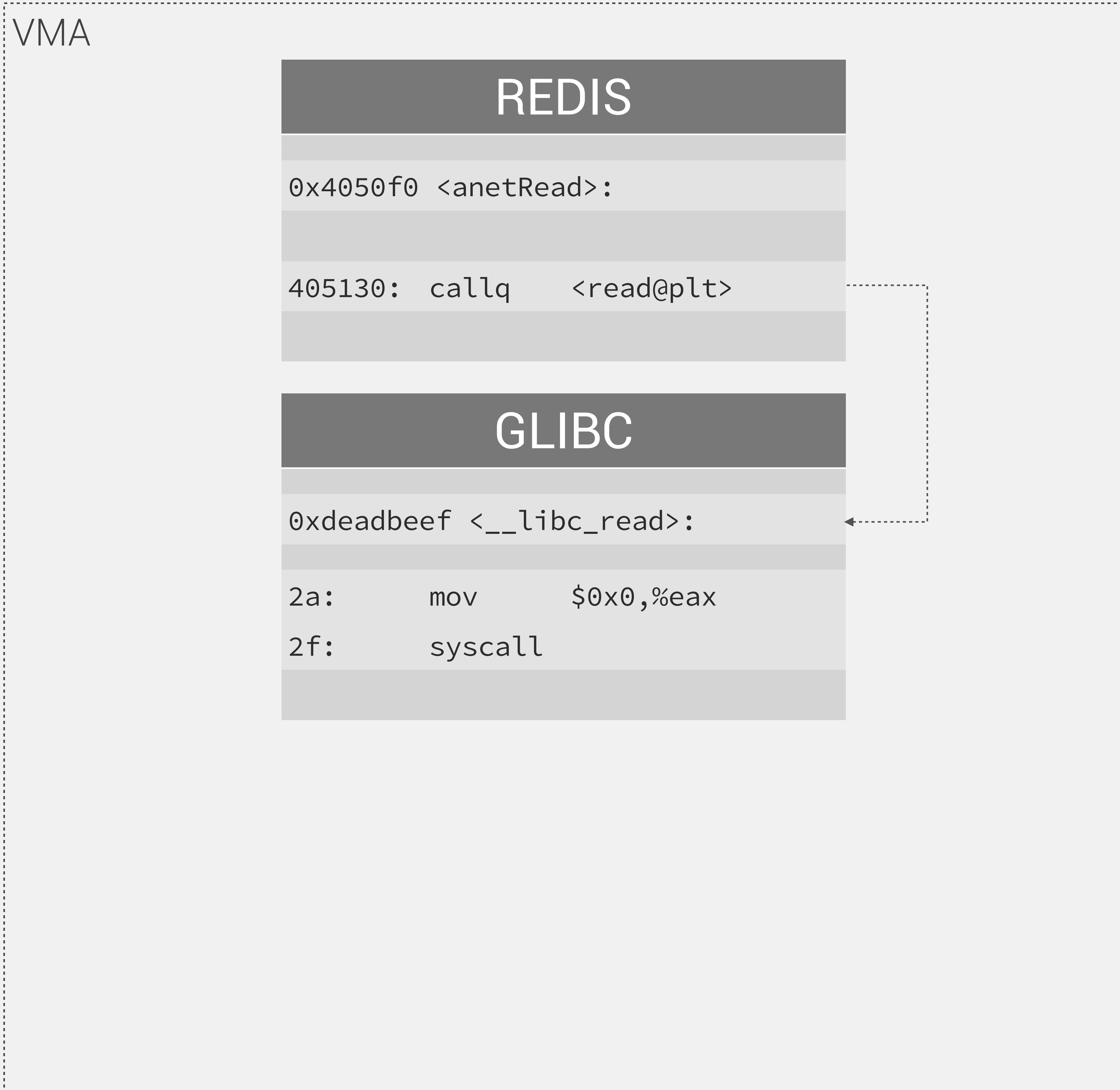
Snippet of system call trace

MX

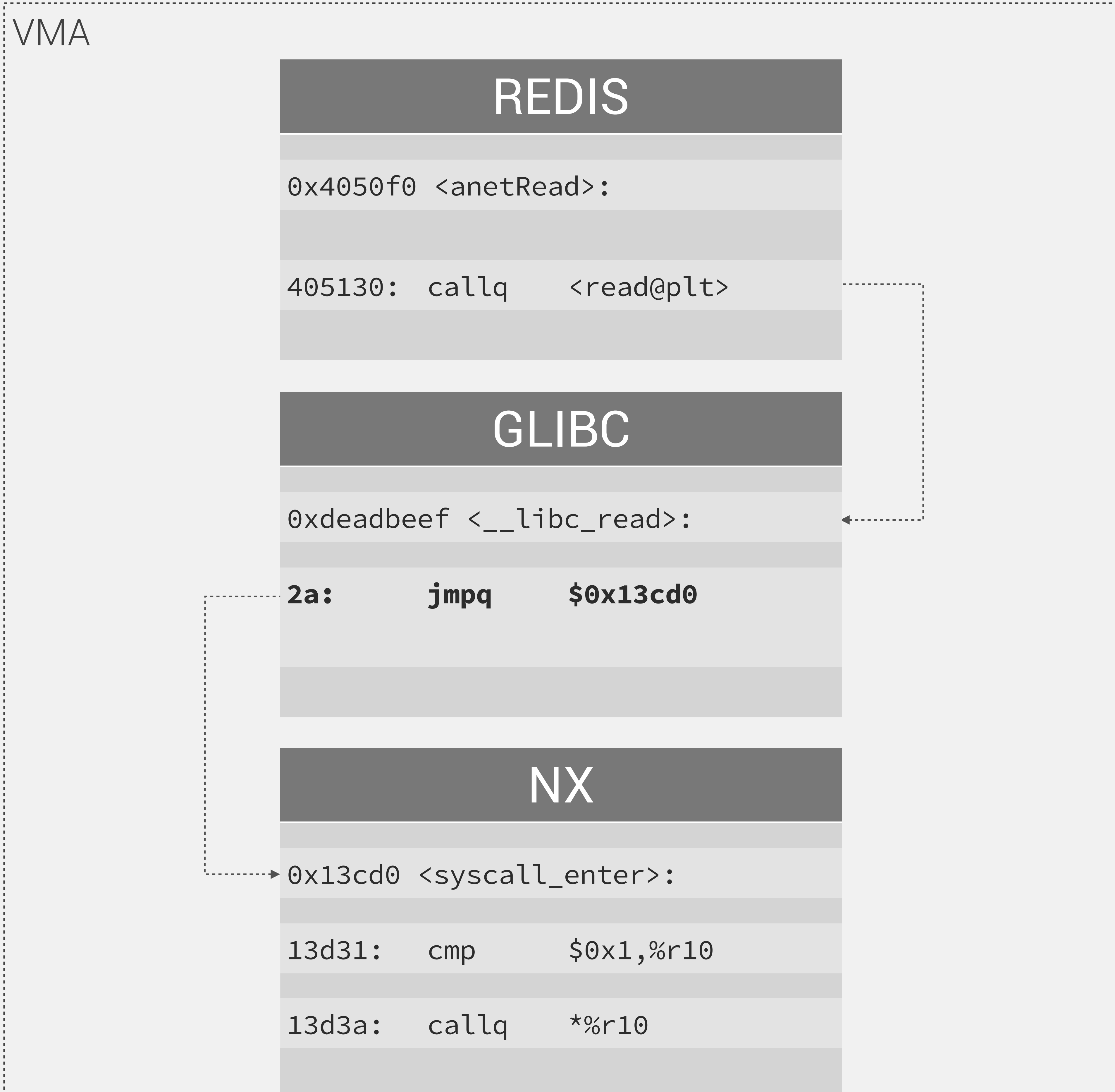
```
ptrace(PTRACE_GETREGS, 7, {...}, NULL)
ptrace(PTRACE_SETREGS, 7, {...}, {...})
ptrace(PTRACE_SYSCALL, 7, {...}, NULL)
read(8, “PING\r\n”, 1024)
ptrace(PTRACE_GETREGS, 7, {...}, NULL)
process_vm_writev(7, {?}, 1, {?}, 1, 0)
ptrace(PTRACE_SETREGS, 7, {...}, {...})
ptrace(PTRACE_SYSCALL, 7, {...}, NULL)
```

Snippet of system call trace


VMA
REDIS
--- SIGTRAP ---
getpid()
--- SIGTRAP ---
Snippet of system call trace
MX
ptrace(PTRACE_GETREGS, 7, {...}, NULL)
ptrace(PTRACE_SETREGS, 7, {...}, {...})
ptrace(PTRACE_SYSCALL, 7, {...}, NULL)
read(8, “PING\r\n”, 1024)
ptrace(PTRACE_GETREGS, 7, {...}, NULL)
process_vm_writev(7, {?}, 1, {?}, 1, 0)
ptrace(PTRACE_SETREGS, 7, {...}, {...})
ptrace(PTRACE_SYSCALL, 7, {...}, NULL)
Snippet of system call trace
VMA



Snippet of instruction code

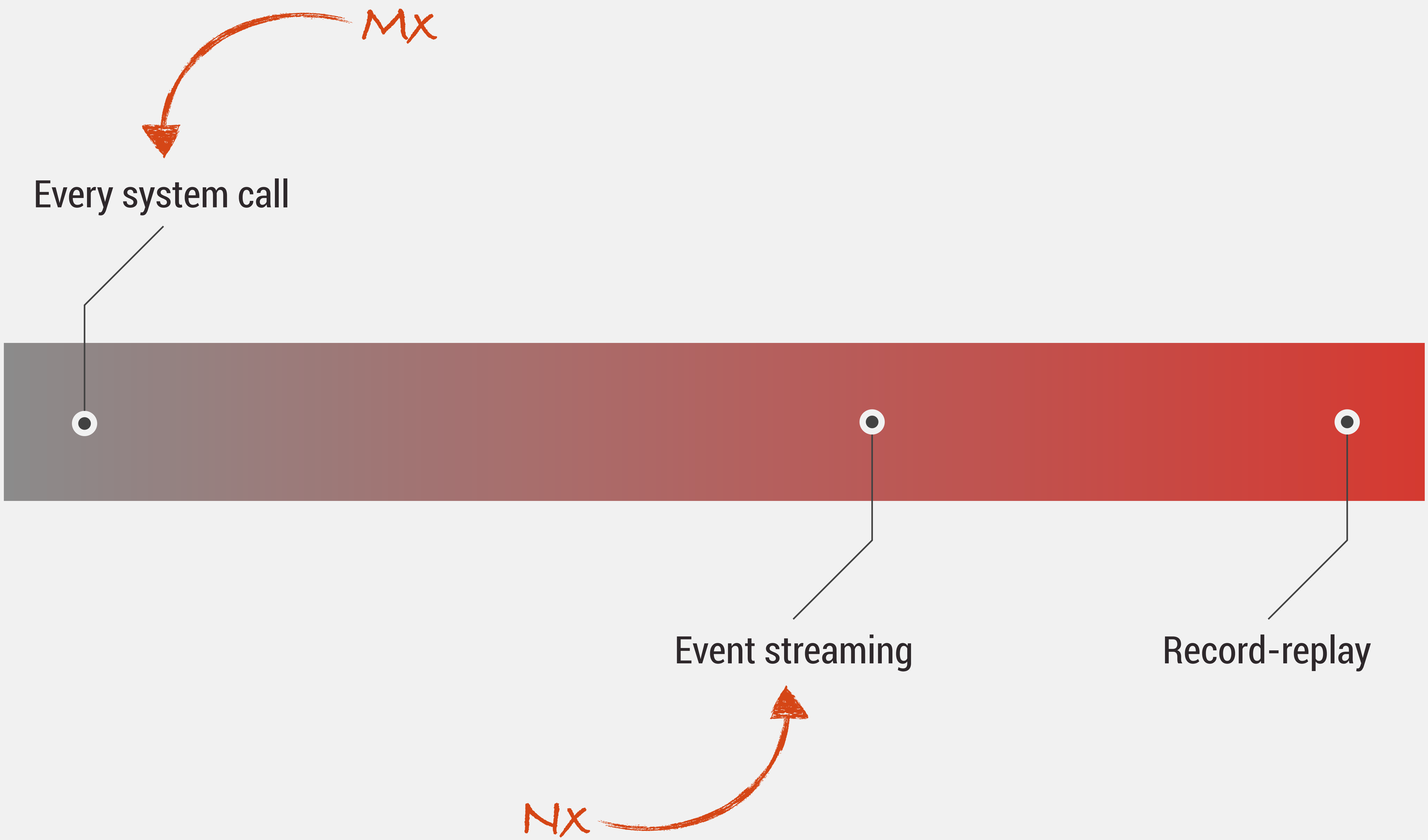


Snippet of instruction code

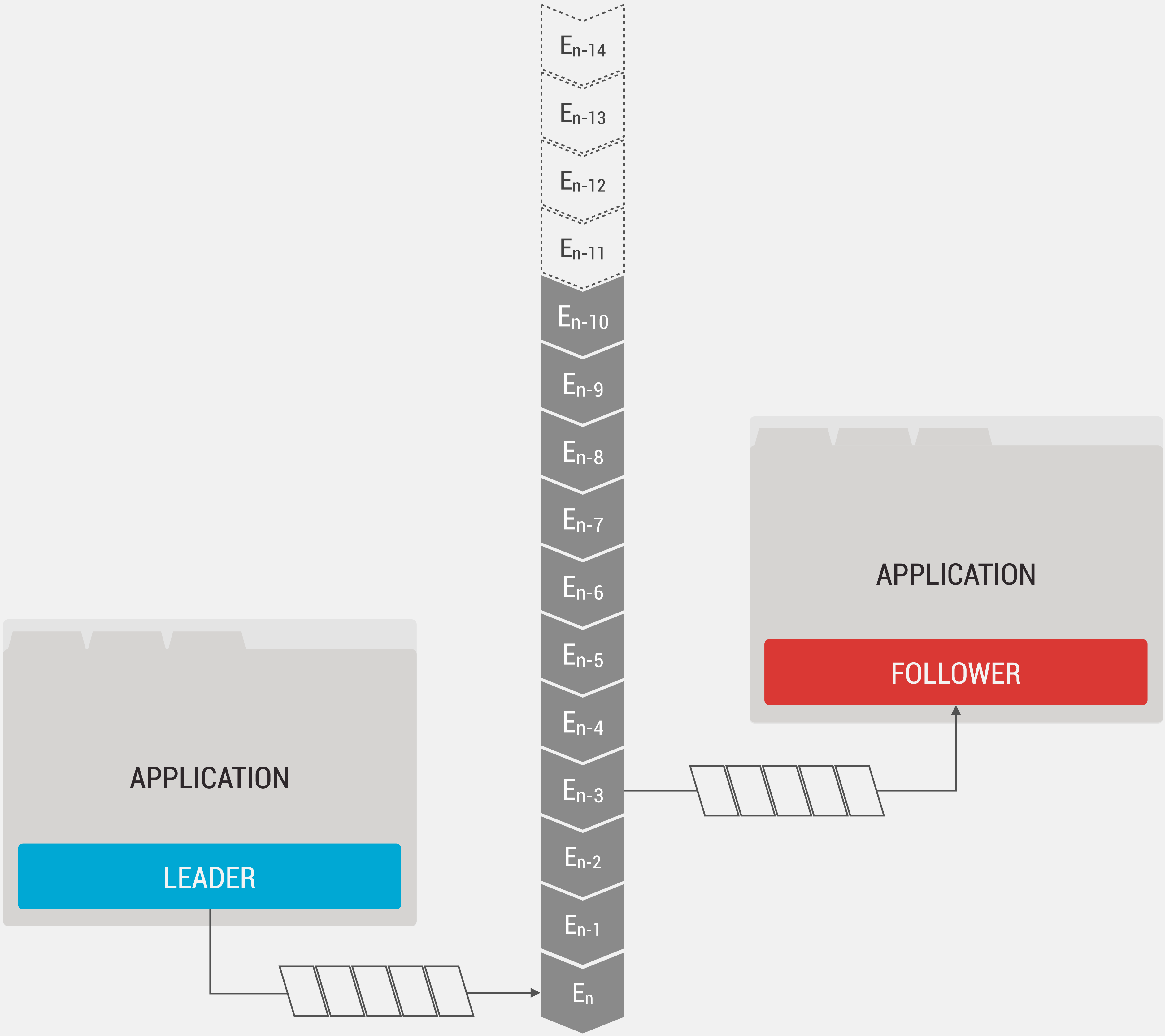
System call synchronisation possible at different phases

System call synchronisation possible at different phases

System call synchronisation possible at different phases

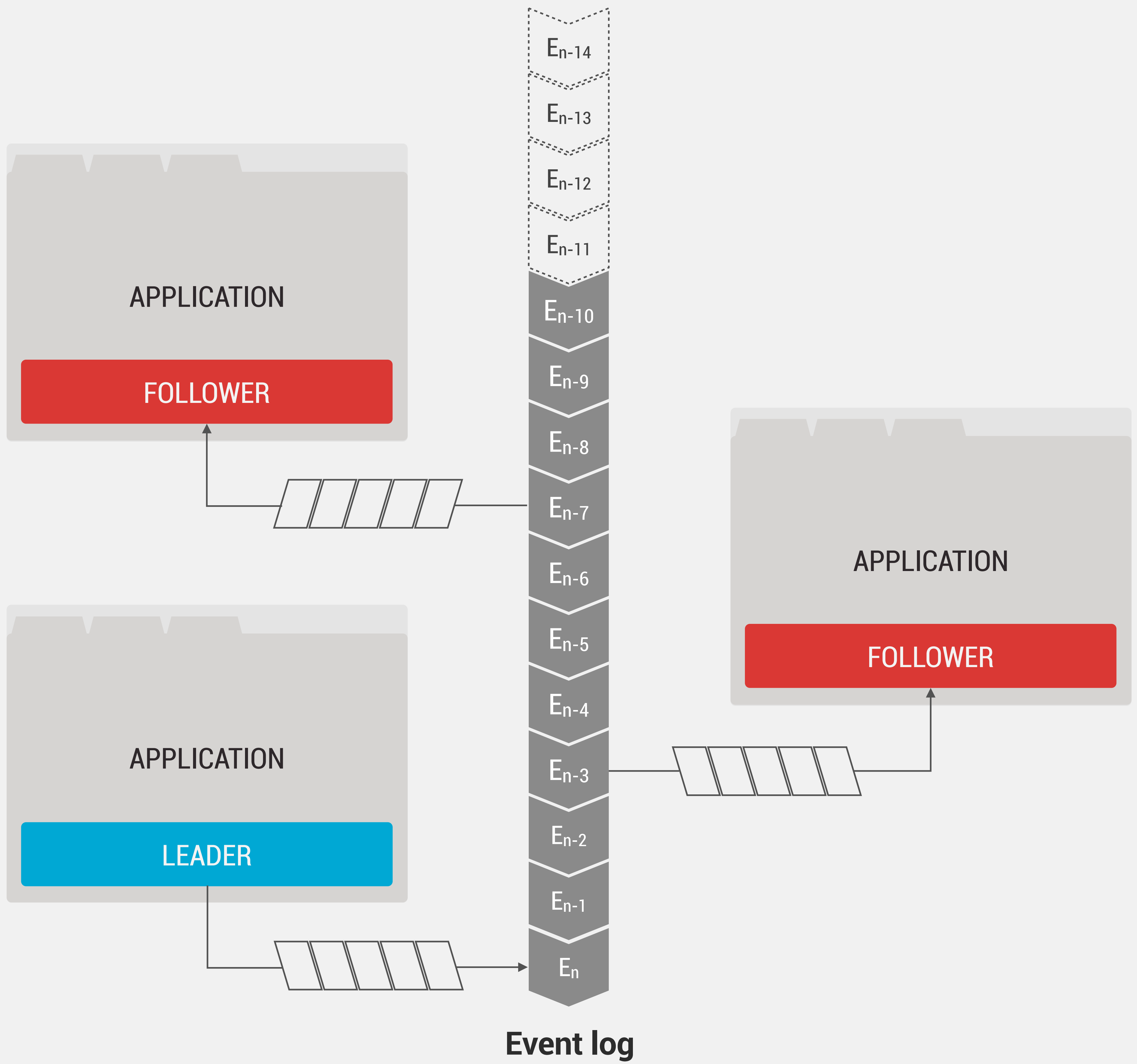


System call synchronisation possible at different phases

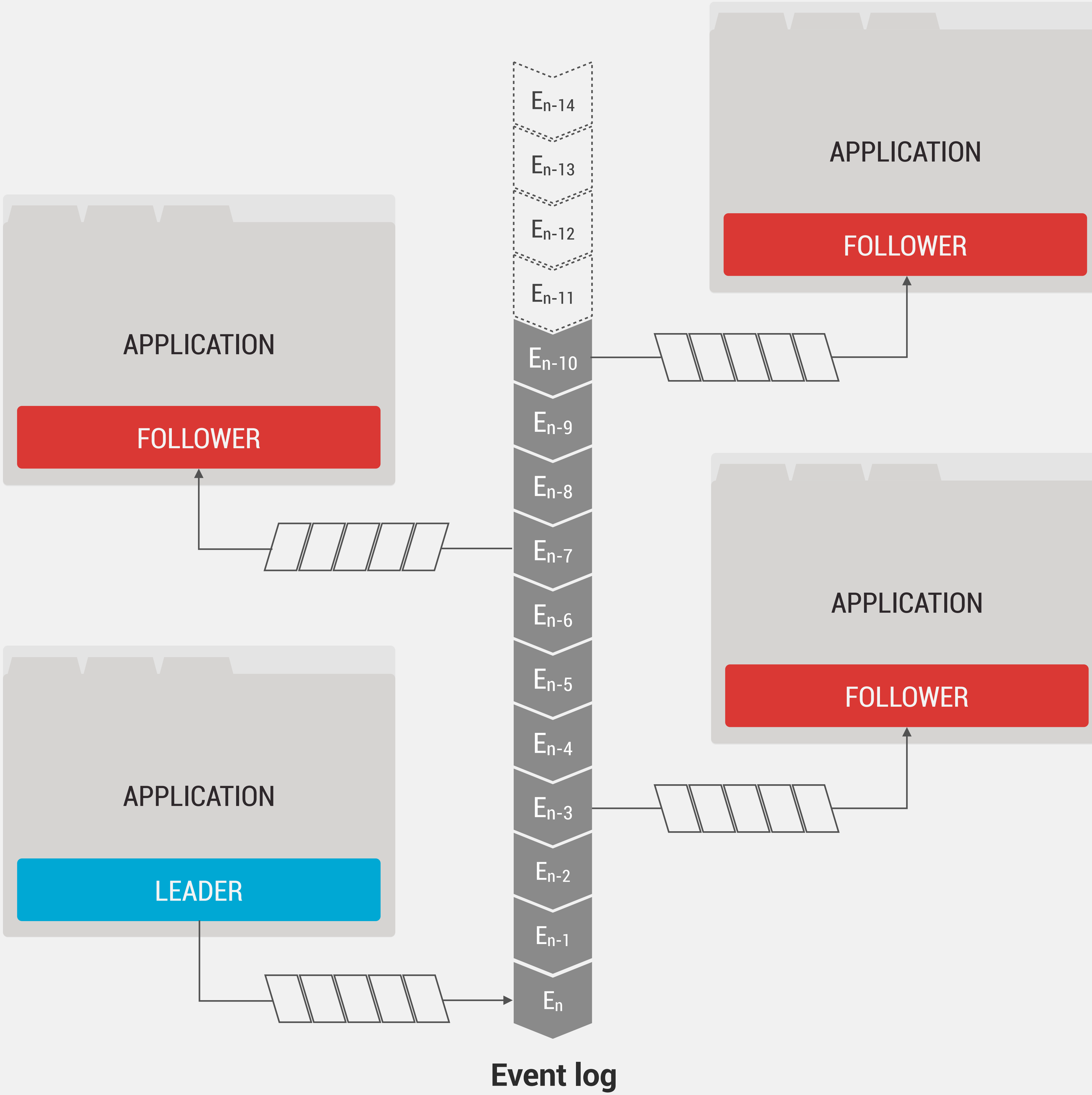


En-14
En-13
En-12
En-11
En-10
En-9
En-8
En-7
En-6
En-5
En-4
En-3
En-2
En-1
En
APPLICATION
FOLLOWER
APPLICATION
LEADER

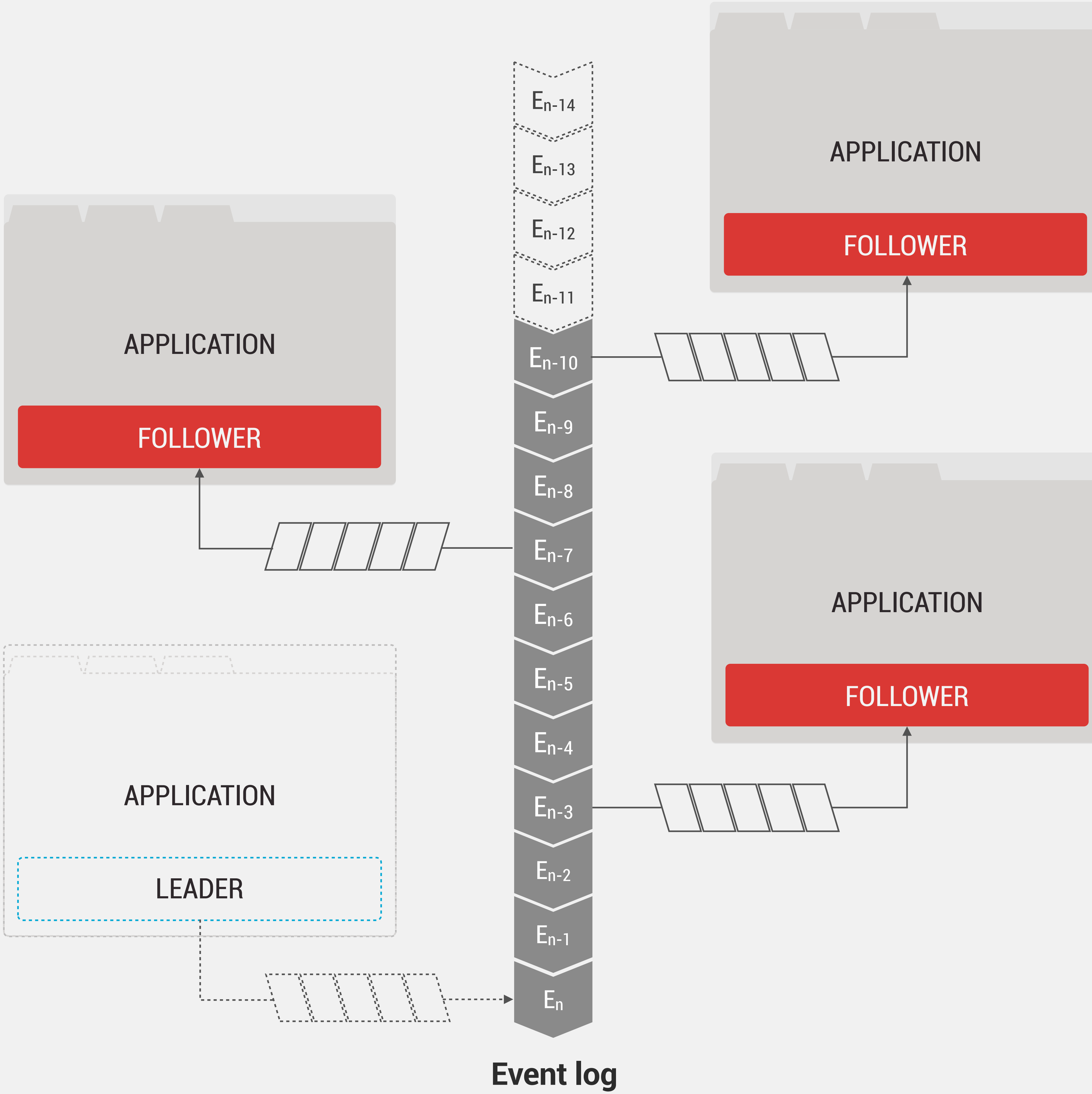
Event log

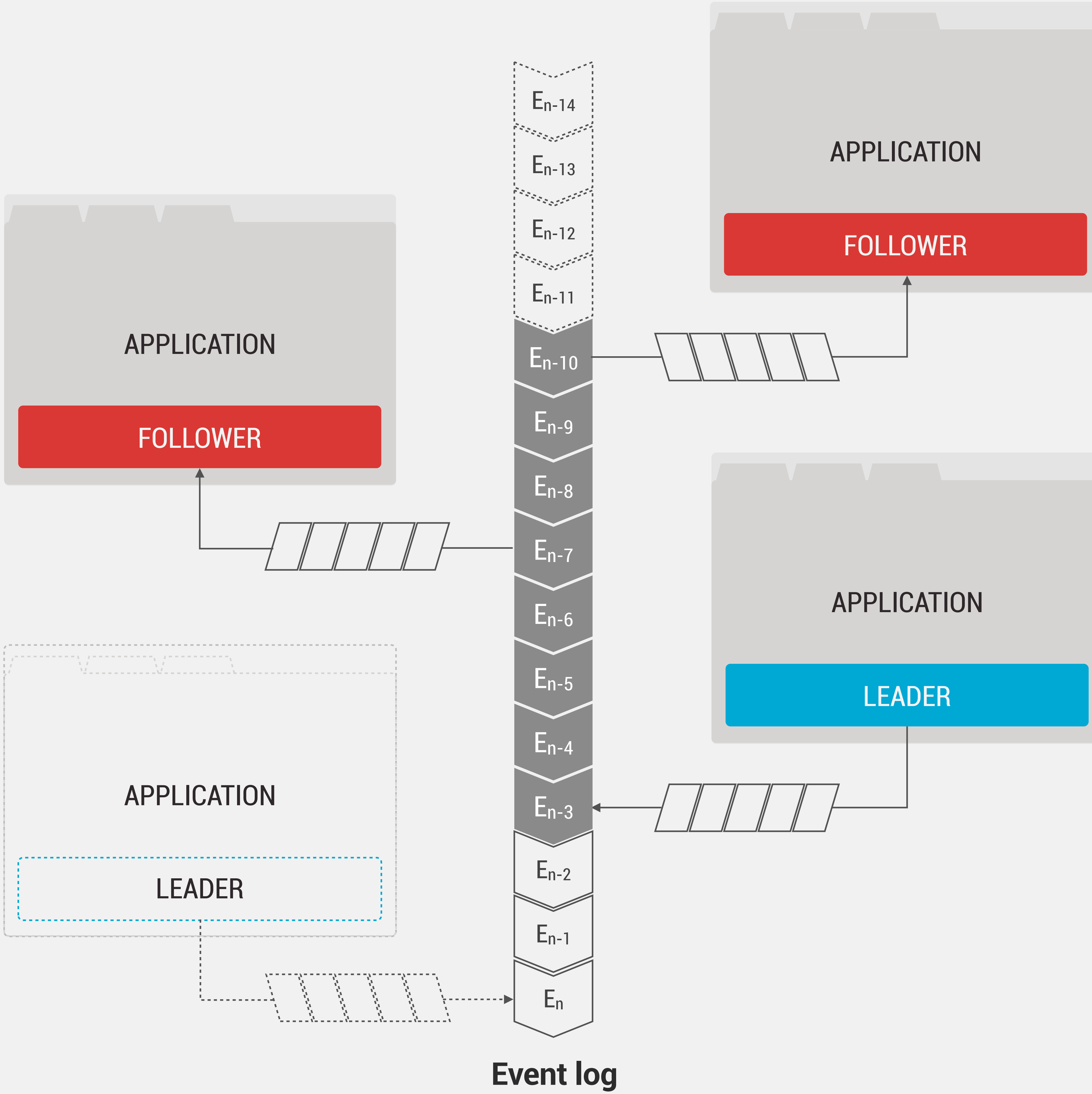


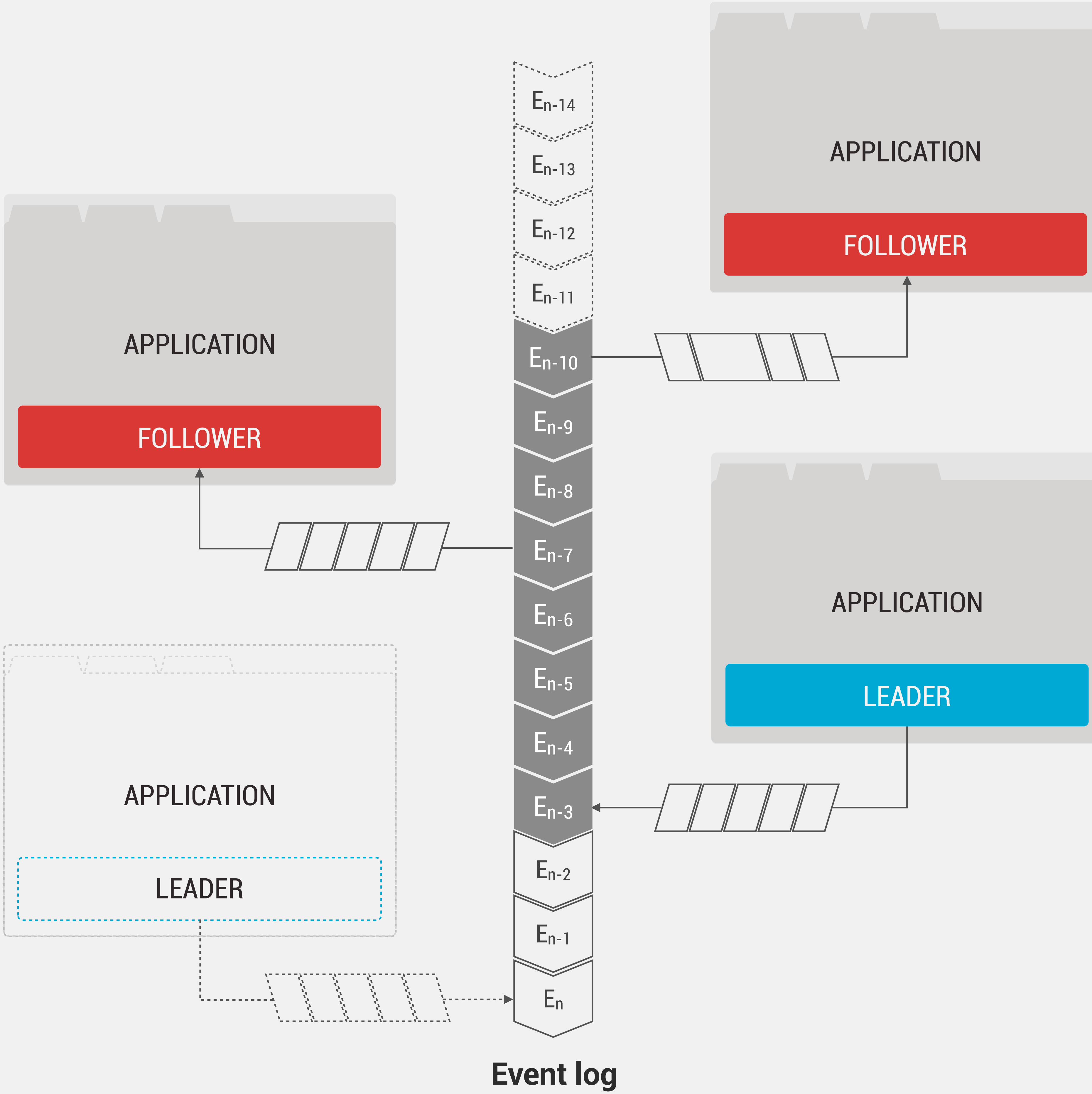
APPLICATION
FOLLOWER
APPLICATION
LEADER
APPLICATION
FOLLOWER
E_{n-14}
E_{n-13}
E_{n-12}
E_{n-11}
E_{n-10}
E_{n-9}
E_{n-8}
E_{n-7}
E_{n-6}
E_{n-5}
E_{n-4}
E_{n-3}
E_{n-2}
E_{n-1}
E_n
Event log



APPLICATION
FOLLOWER
APPLICATION
FOLLOWER
APPLICATION
FOLLOWER
APPLICATION
LEADER
En-14
En-13
En-12
En-11
En-10
En-9
En-8
En-7
En-6
En-5
En-4
En-3
En-2
En-1
En
Event log


APPLICATION
FOLLOWER
APPLICATION
FOLLOWER
APPLICATION
FOLLOWER
APPLICATION
LEADER
E_{n-14}
E_{n-13}
E_{n-12}
E_{n-11}
E_{n-10}
E_{n-9}
E_{n-8}
E_{n-7}
E_{n-6}
E_{n-5}
E_{n-4}
E_{n-3}
E_{n-2}
E_{n-1}
E_{n}
Event log


APPLICATION
FOLLOWER
APPLICATION
FOLLOWER
APPLICATION
LEADER
APPLICATION
LEADER
E_{n-14}
E_{n-13}
E_{n-12}
E_{n-11}
E_{n-10}
E_{n-9}
E_{n-8}
E_{n-7}
E_{n-6}
E_{n-5}
E_{n-4}
E_{n-3}
E_{n-2}
E_{n-1}
E_n
Event log


APPLICATION
FOLLOWER
APPLICATION
FOLLOWER
APPLICATION
LEADER
APPLICATION
LEADER
En-14
En-13
En-12
En-11
En-10
En-9
En-8
En-7
En-6
En-5
En-4
En-3
En-2
En-1
En
Event log

Future Work

Support for more complex code changes:

- Data structure inference & excavation
- Control flow graph isomorphisms
- Call stack reconstruction

Support for non-crashing type of divergences:

- Infinite loops and deadlocks

Summary

Novel approach for improving software updates:

- Based on multi-version execution
- Mx can survive crash bugs in real apps

Many opportunities for future work:

- Better performance overhead
- Tolerance to system call divergencies
- Support for more complex code changes
- Support for non-crashing type of divergences

Distinct code bases, manually-generated

N-version programming: A fault-tolerance approach to reliability of software operation

Chen, L., and Avizienis, A. FTCS'78

Using replicated execution for a more secure and reliable web browser

Xue, H., Dautenhahn, N., and King, S. T. NDSS'12

Variants of the same code, automatically generated

N-variant systems: a secretless framework for security through diversity

Cox, B., Evans, D., Filipi, A., Rowanhill, J., Hu, W., Davidson, J., Knight, J., Nguyen-Tuong, A., and Hiser, J. USENIX Security'06

Run-time defense against code injection attacks using replicated execution

Salamat, B., Jackson, T., Wagner, G., Wimmer, C., and Franz, M. IEEE Transactions 2011

Online validation of different manually-evolved versions

Efficient online validation with delta execution

Tucek, J., Xiong, W., Zhou, Y. ASPLOS'09

Tachyon: Tandem Execution for Efficient Live Patch Testing

Maurer, M., Brumley, D. USENIX Security'12