Mx: Safe Software Updates via Multi-version Execution

Petr Hosek Cristian Cadar

Software Reliability Group

Department of Computing

Imperial College London

18th CREST Workshop, London, UK

22nd March 2012

“The fundamental problem with program maintenance is that fixing a defect has a substantial (20*-50%) chance of introducing another. So the whole process is two steps forward and one step back.”

— Fred Brooks, 1975

*≥14.8~24.4% for major operating system patches

Yin, Z., Yuan, D., Zhou, Y., Pasupathy, S., and Bairavasundaram, L. How Do Fixes Become Bugs? ESEC/FSE’11

Motivation

Software evolves, with new versions and patches being released frequently

Software updates often present a high risk

Many users refuse to upgrade their software…

…relying instead on outdated versions flawed with vulnerabilities or missing useful features and bug fixes

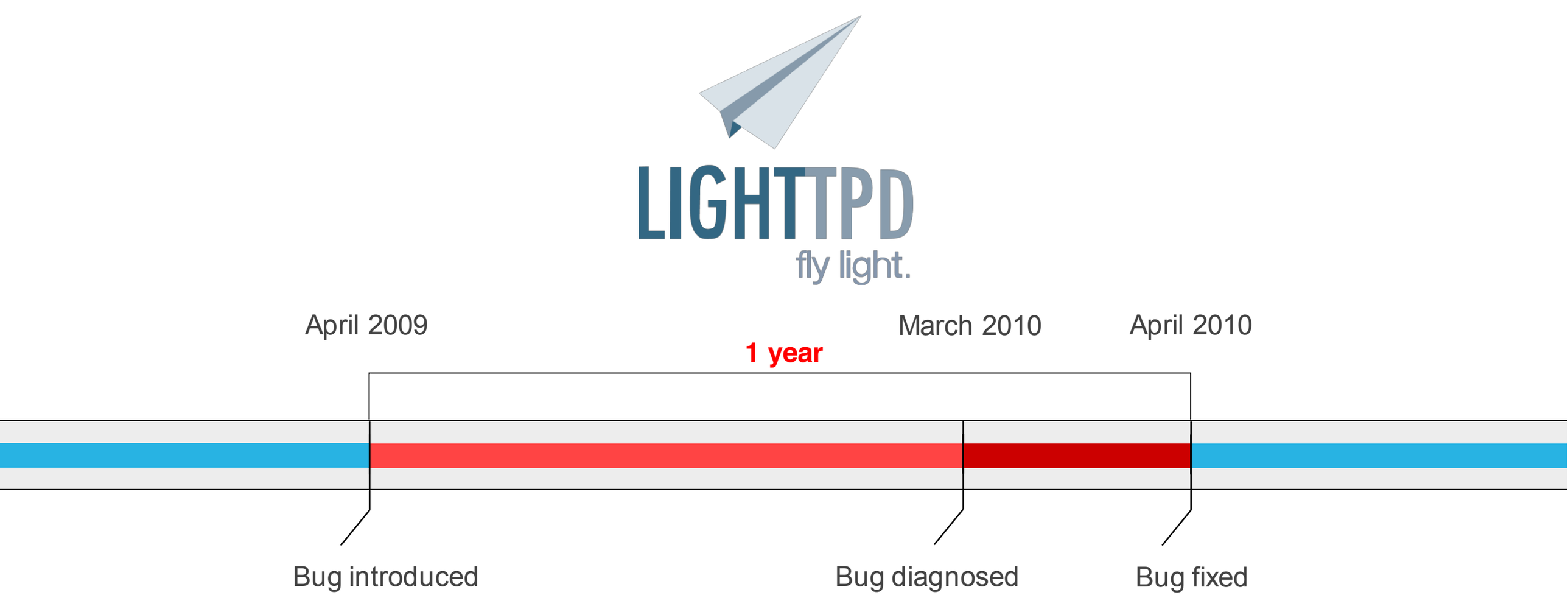
Powers several popular sites such as YouTube, Wikipedia, Meebo

File (re)compression in mod_compress_physical

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

HTTP ETag hash value computation in etag_mutate

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



File (re)compression in mod_compress_physical

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

HTTP ETag hash value computation in etag_mutate

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

Goals

Improve the software update process to provide

- Benefits of the newer version
- Stability of the older version

Solution

Multi-version execution based approach

- Run both versions in parallel
- Use output of correctly executing version at any given time

Challenges

1. **Allowing multiple versions to run side-by-side**
2. **Handling divergences and recovering from failures**

(in the context of multi-core CPUs)

Challenge 1: MV execution environment

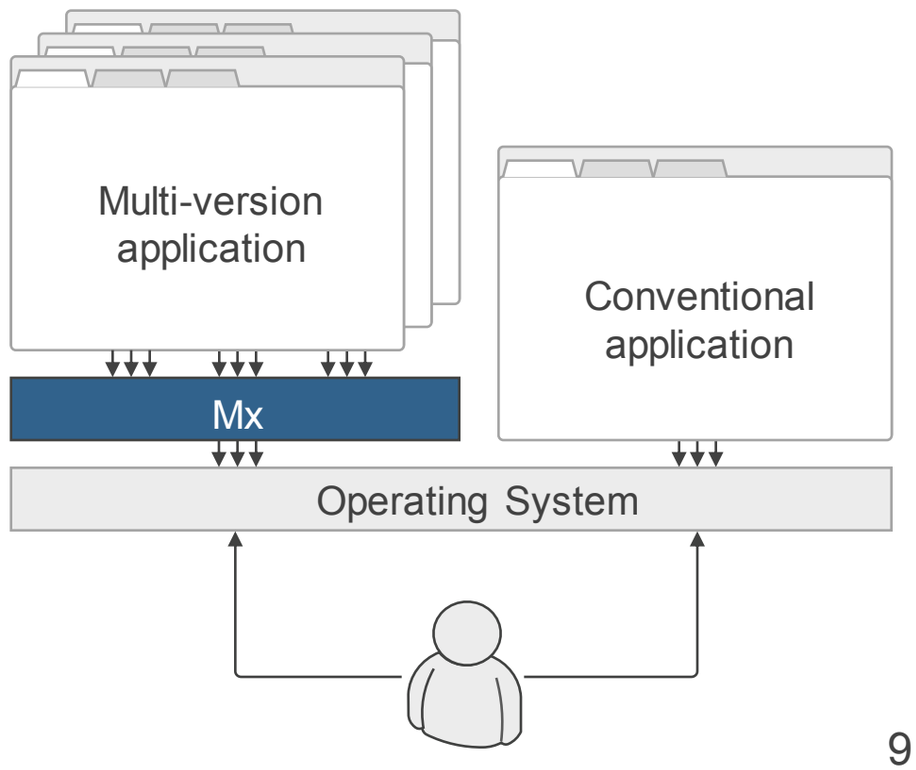
Multi-version execution environment

Synchronize execution of multiple versions

Multi-version app acts as one to the external world

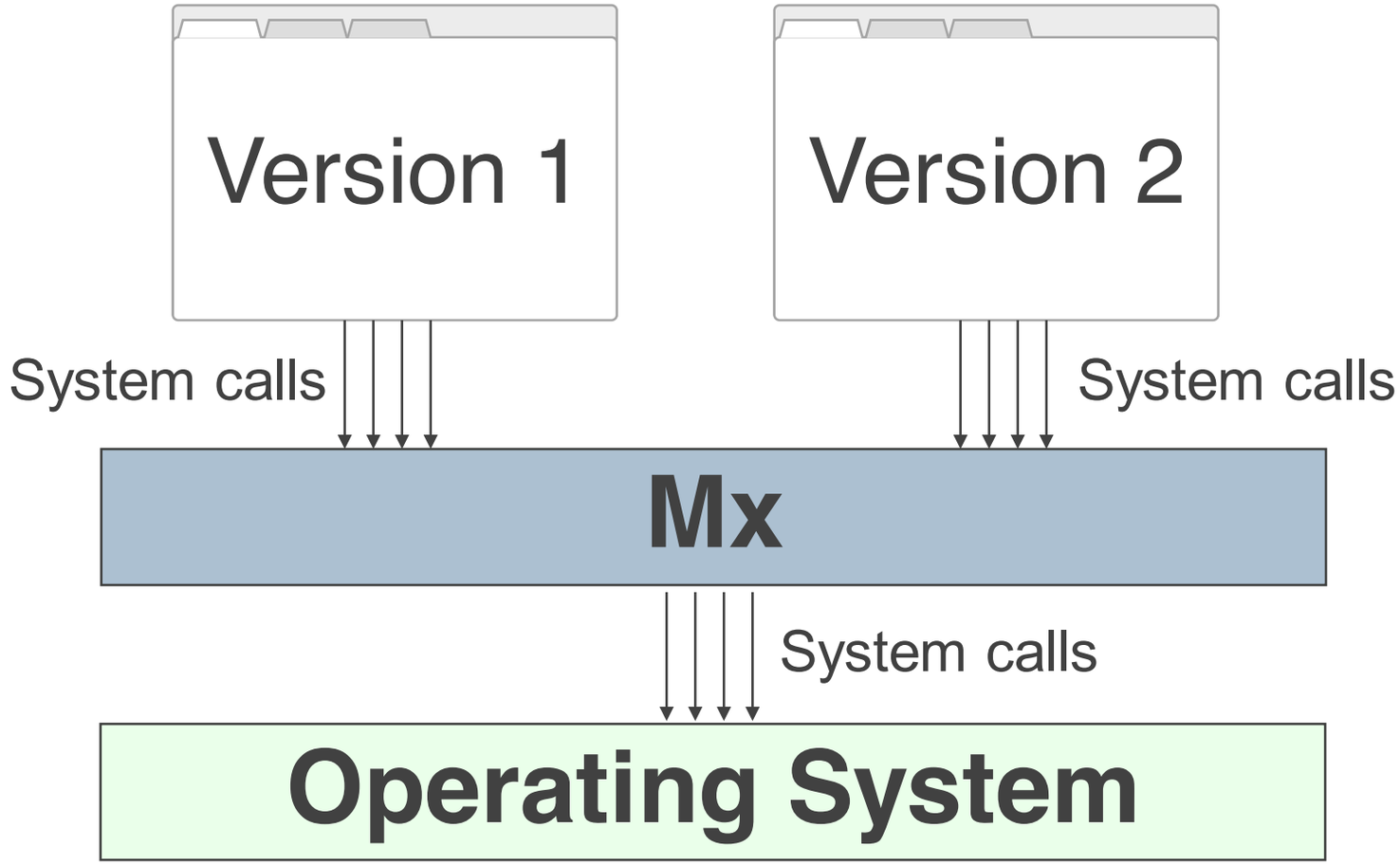
Reasonable performance overhead

Support for native applications



Synchronization

Synchronization (and virtualization) at the level of system calls



System calls define external behavior

Version 1

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr);

    bsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

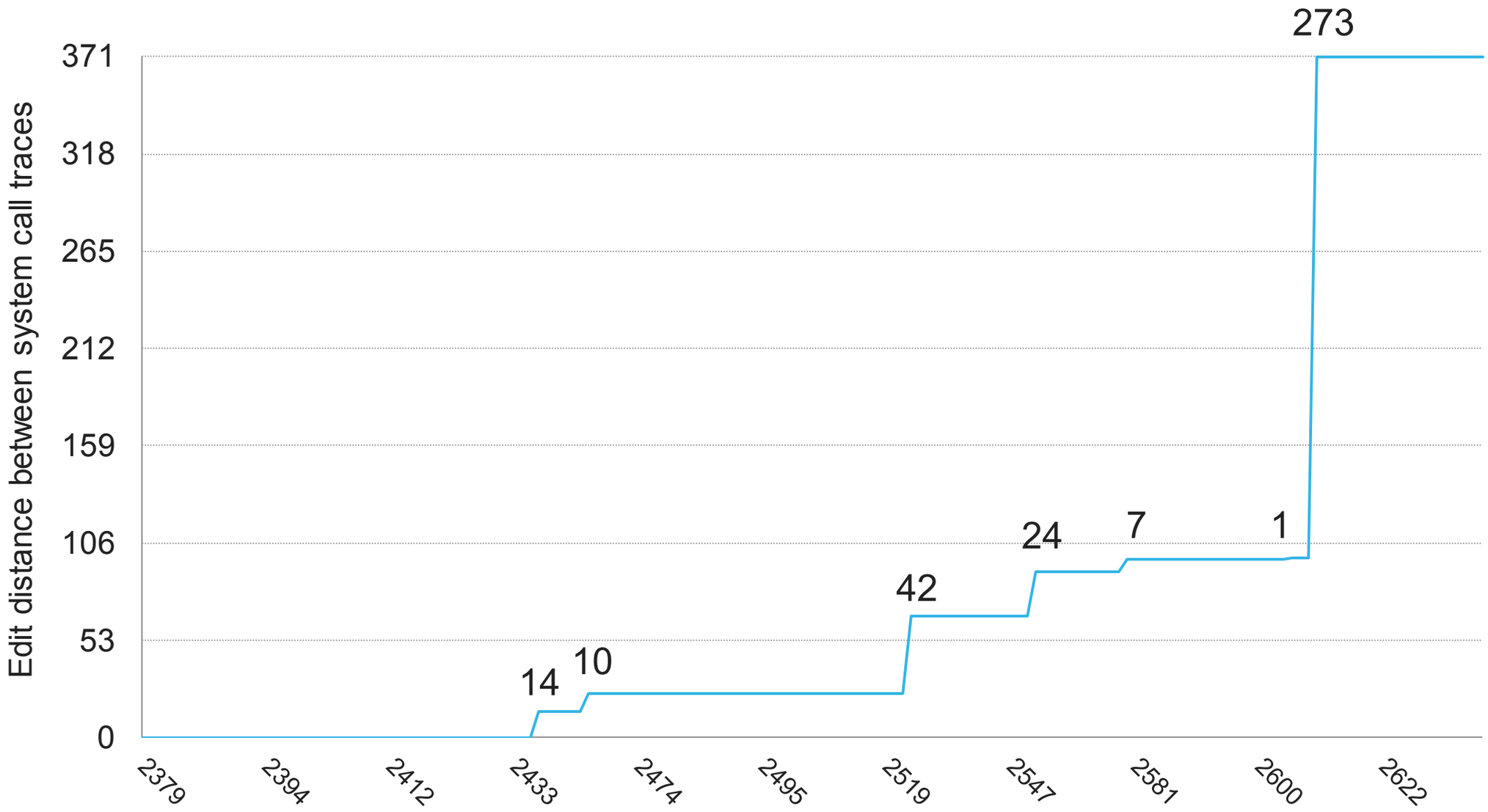
Version 2

```
void print_sorted(int *arr, size_t len)
{
    int sarr[len];
    memcpy(sarr, arr, sizeof(sarr);

    qsort(sarr, len, sizeof(int), cmp);
    for (int i = 0; i < len; ++i)
        printf("%d\n", sarr[i]);
}
```

```
int arr[] = { 6, 4, 3, 7 };
print_sorted(arr, 4);

...
write(1, "3\n", 2) = 2
write(1, "4\n", 2) = 2
write(1, "6\n", 2) = 2
write(1, "7\n", 2) = 2
...
```

External behavior evolves sporadically

95% of revisions introduce *no change**



Measured using lighttpd regression suite on 164 revisions (~10 months)

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

Challenge 2: Handling divergences

Handle divergences across versions

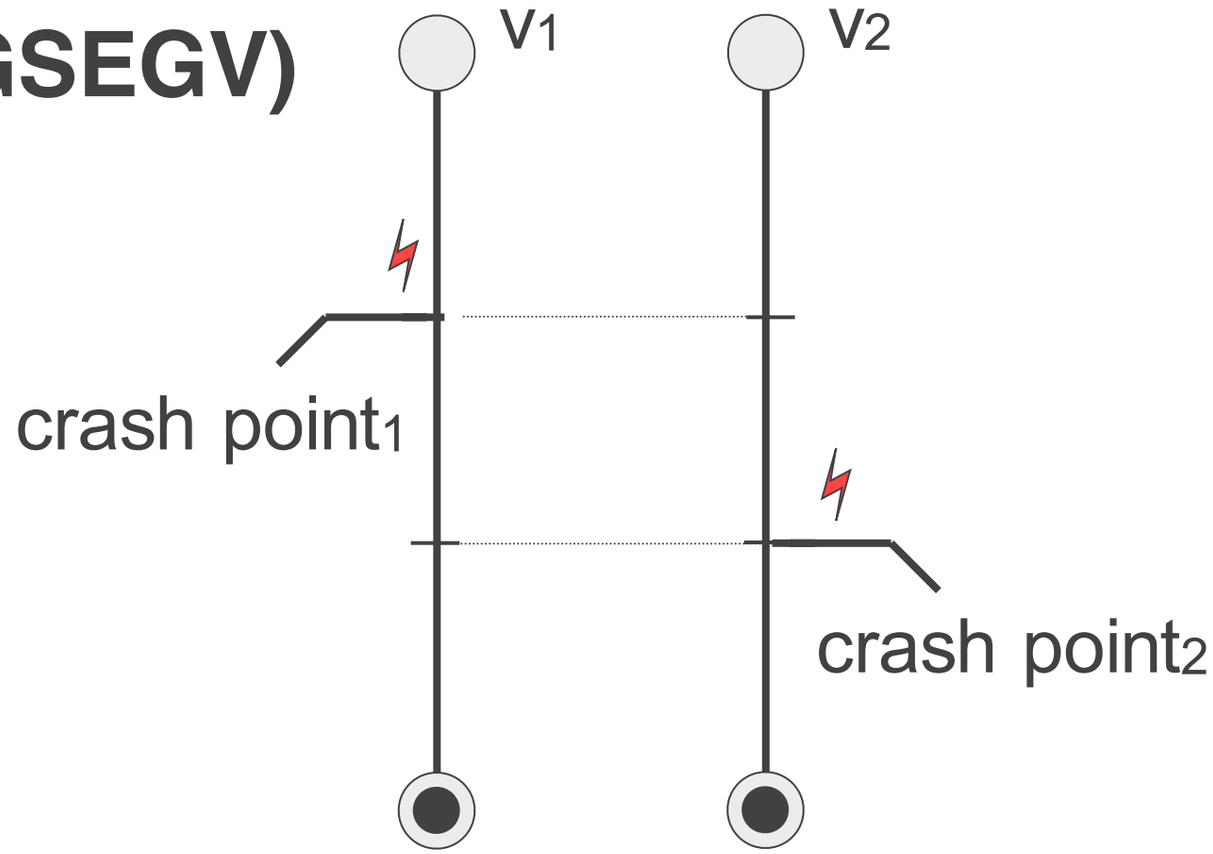
- Accurately detect divergences
- Recover from failures
- Re-synchronize executions

Failure Recovery: Scope

Small differences in external behavior

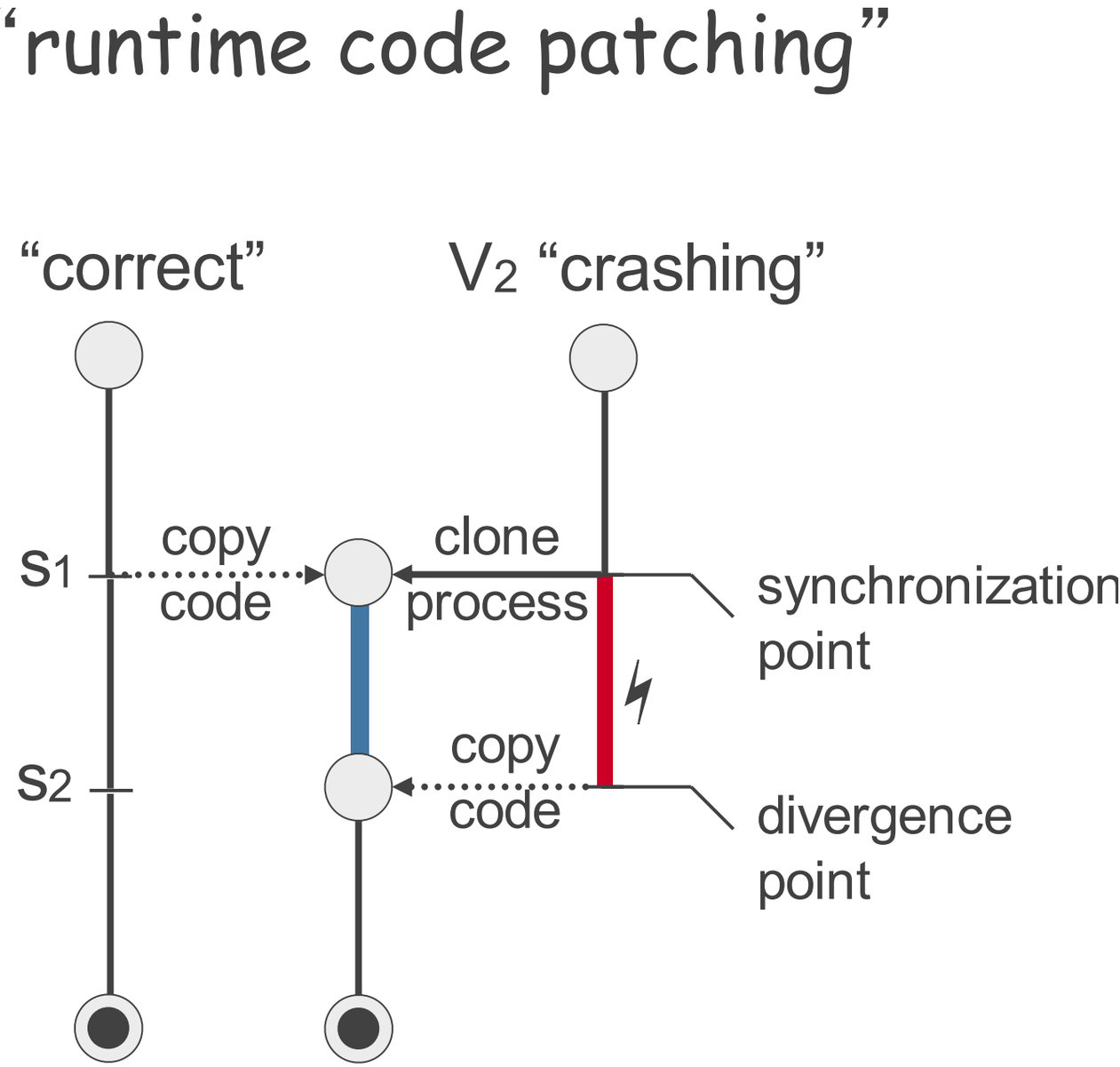
E.g., two minor versions

Divergences are crashes (SIGSEGV)



Failure Recovery Process

1. Revert to last successful synchronization point
2. Copy code from “correct” version
3. Run to divergence point
4. Revert back to original code
5. Restart multi-version execution

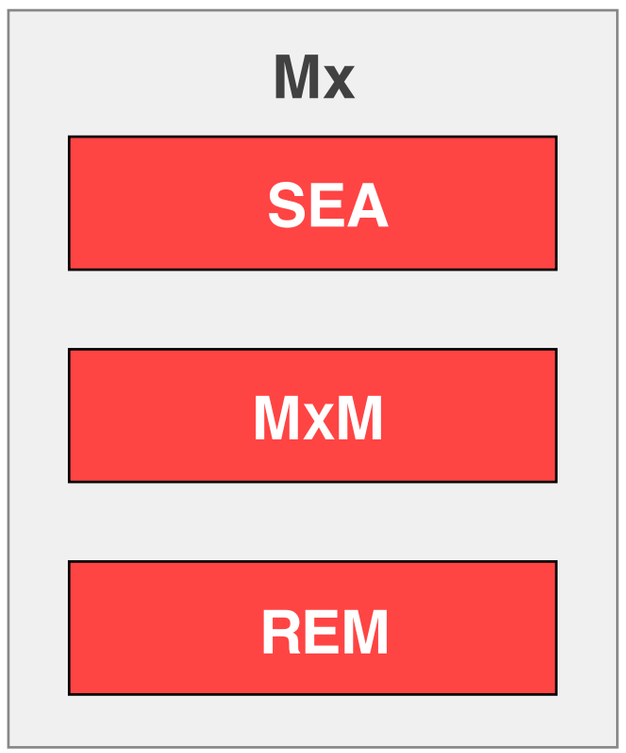


Mx Prototype

System targets multi-core processors

Support for x86 and x86-64 Linux systems

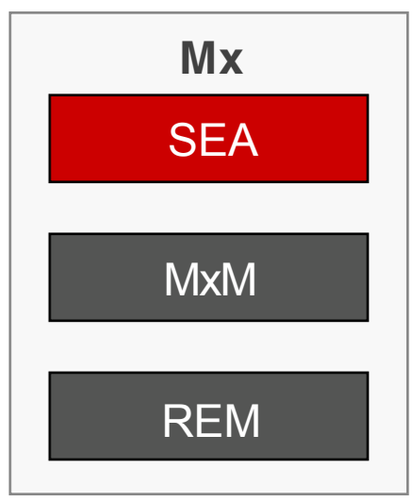
Combines system call interposition, OS-level checkpointing, binary static analysis, and runtime code patching



SEA: Static Binary Analyzer

Create various mappings between the two version binaries

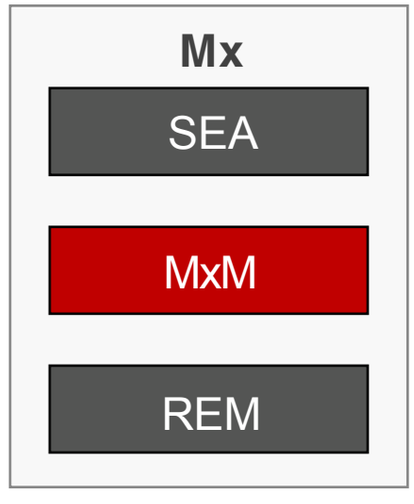
- Static analysis of binary executables
- Extracting function symbols from binaries
- Machine code disassembling and analysis
- Binary call graph reconstruction



MxM: Multi-eXecution Monitor

Execute and monitor multi-version applications

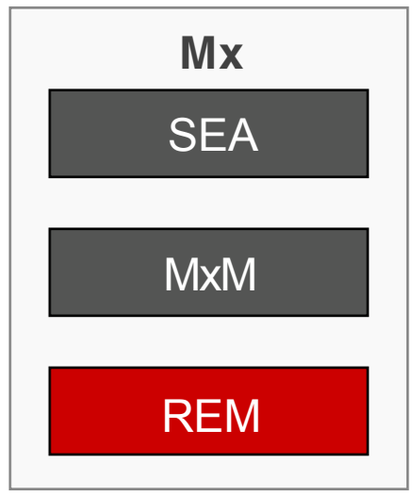
- Synchronization at the level of system calls
- System call interposition (via `ptrace` interface)
- Environment virtualization (i.e. files and sockets)
- Support for multi-process applications



REM: Runtime Execution Manipulator

Runtime code patching and fault recovery

- Runtime stack manipulation
- Breakpoint insertion and handling
- OS-level checkpointing (using clone syscall)



Preliminary Results

Survived a number of crash bugs in two popular servers

Web-server used by several popular sites such as YouTube, Wikipedia, Meebo

Key-value data structure server, used by popular services such as GitHub, Digg, Flickr

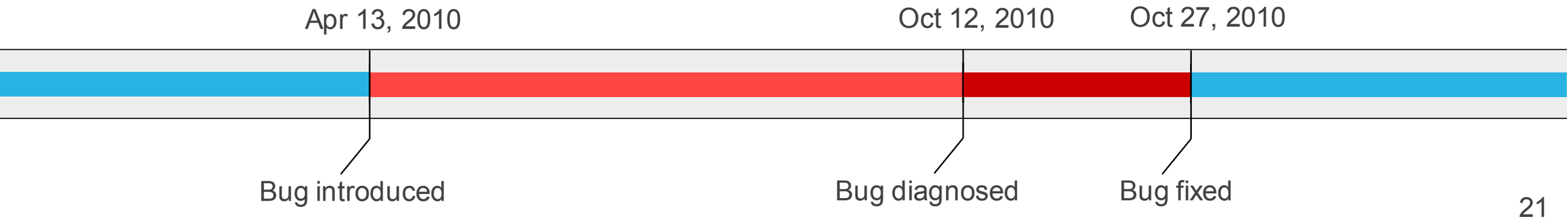
HMGET command hmgetCommand function

```
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));
```

```
robj *o, *value;
o = lookupKeyRead(c->db,c->argv[1]);
if (o != NULL && o->type != REDIS_HASH) {
    addReply(c,shared.wrongtypeerr);
    return; <- missing 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);
    }
}
```

Refactor

Bug may result in loosing some or even all of the stored data

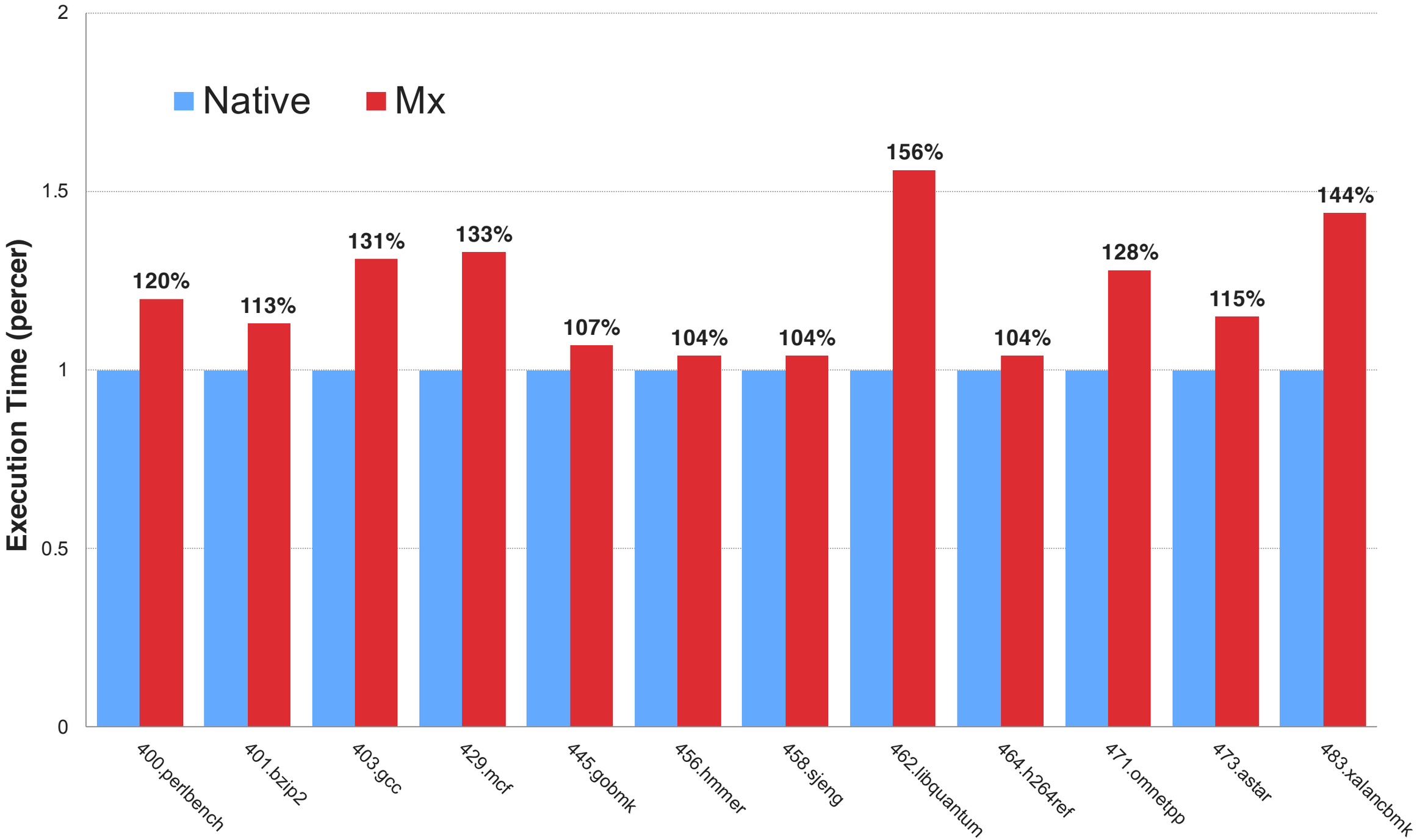


Maximum distance between versions

Application	Max distance	Time span
lighttpd #2169	87	2 months 2 days
lighttpd #2140	12	2 months 1 day
redis #344	27	6 days

21.48% overhead on SPEC CINT CPU2006

WiP: up to **17x** for some other benchmarks



SPEC CINT CPU2006 1.2

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

Selected Related Work

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

Diehard: Probabilistic memory safety for unsafe languages. Berger, E, and Zorn, B. *PLDI'06*

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 TDSC '11*

Different manually-evolved versions of the same code base

Multi-version Software Updates. Cadar, C., and Hosek, P. *HotSWUp'12 (position paper)*

Safe Software Updates via Multi-version Execution. Hosek, P., and Cadar, C. Tech Rep 2011

Summary

Novel approach for improving software updates

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

Many opportunities for future work

Better performance

- Kernel modules, paravirtualization API, skipping safe code, etc.

Support for more complex code changes

- Automatic stack reconstruction, inference of data structure changes, epoch-based system call record & replay