Mx: Safe Software Updates via Multi-version Execution

Petr Hosek Cristian Cadar

Software Reliability Group

Department of Computing

Imperial College London

Stanford University **20th May 2013**

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…relying instead on outdated versions flawed with vulnerabilities or missing useful features and bug fixes

Many admins (70% of those interviewed) refuse to upgrade

Crameri, O., Knezevic, N., Kostic, D., Bianchini, R., Zwaenepoel, W.
Staged deployment in Mirage, an integrated software upgrade testing and distribution system. SOSP'07

> "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 fixes

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

LIGHTTPD

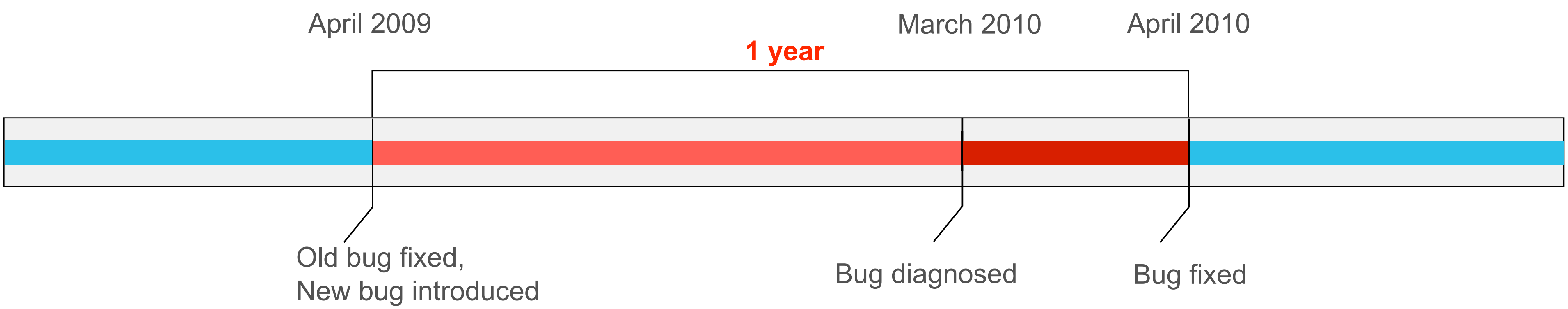
fly light.

Single-threaded event-driven web server

Powers several popular sites such as YouTube, Wikipedia, Meebo

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]);
```



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]);
```

File (re)compression in mod_compress_physical

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

Goals

Improve the software update process to provide

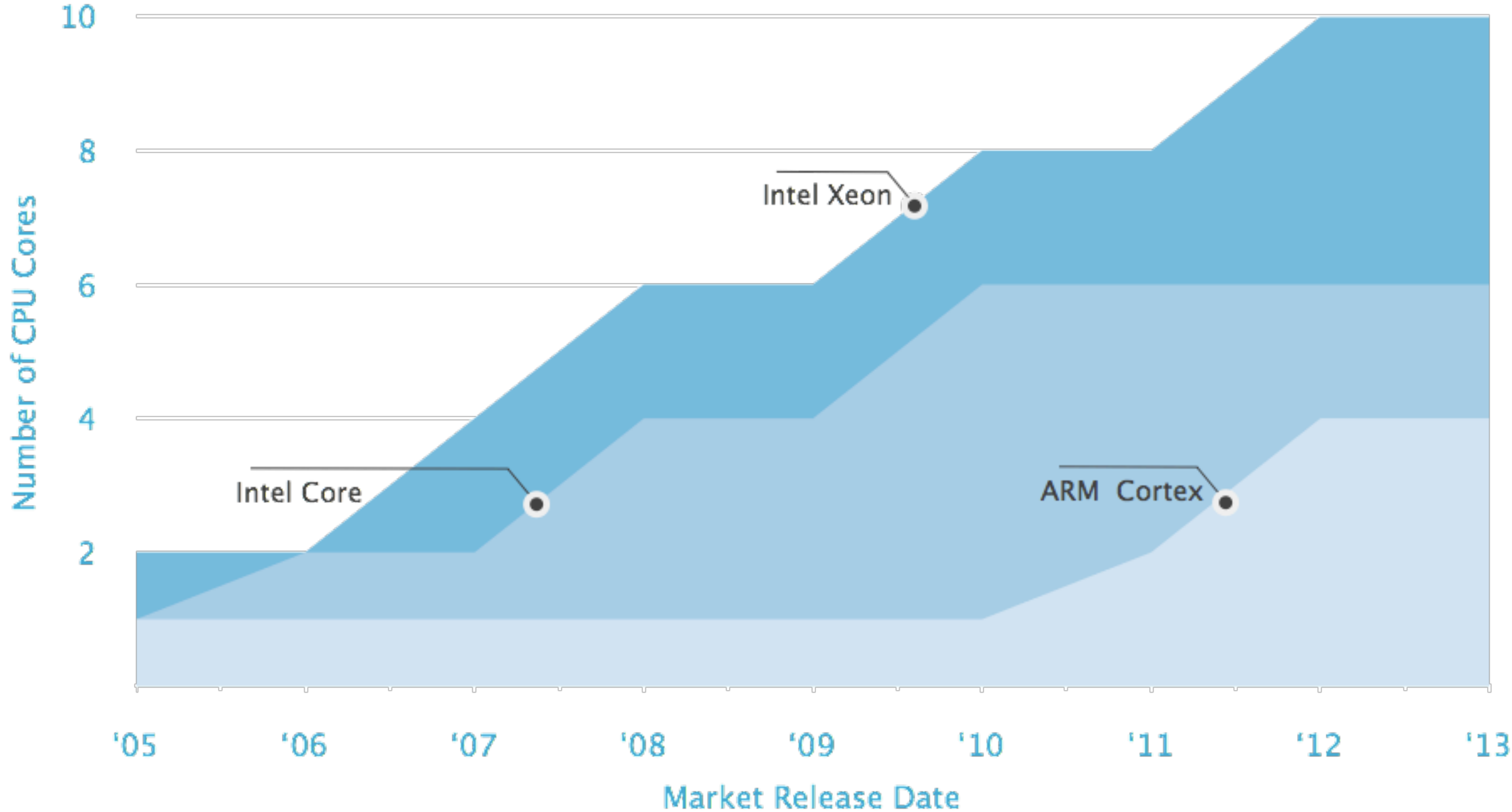
- Benefits of the newer version
- Stability of the older version

Idea

Multi-version execution based approach

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

MultiCore CPUs becoming standard



Idle parallel resources, with no benefit to inherently sequential applications

Cadar, C., Pietzuch, P., Wolf, A. *Multiplicity computing: A vision of software engineering for next-generation computing platform applications.* FoSER'10

Challenges of Multi-Version Execution

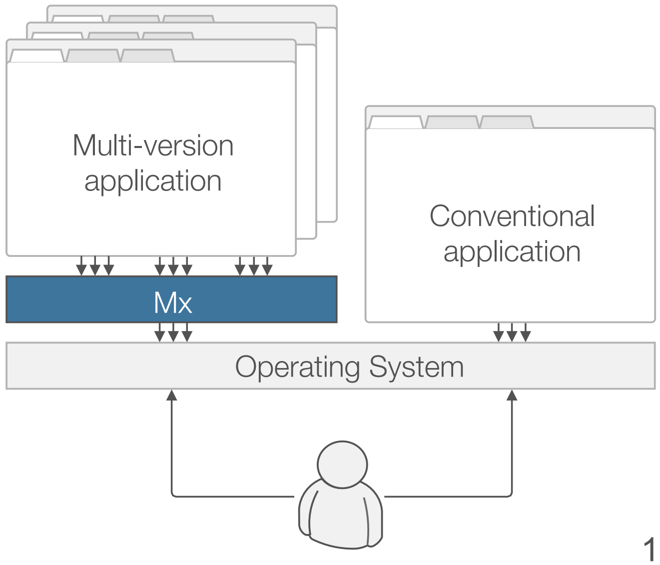
1. Allowing multiple versions to run side-by-side

2. Handling divergences and recovering from failures

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



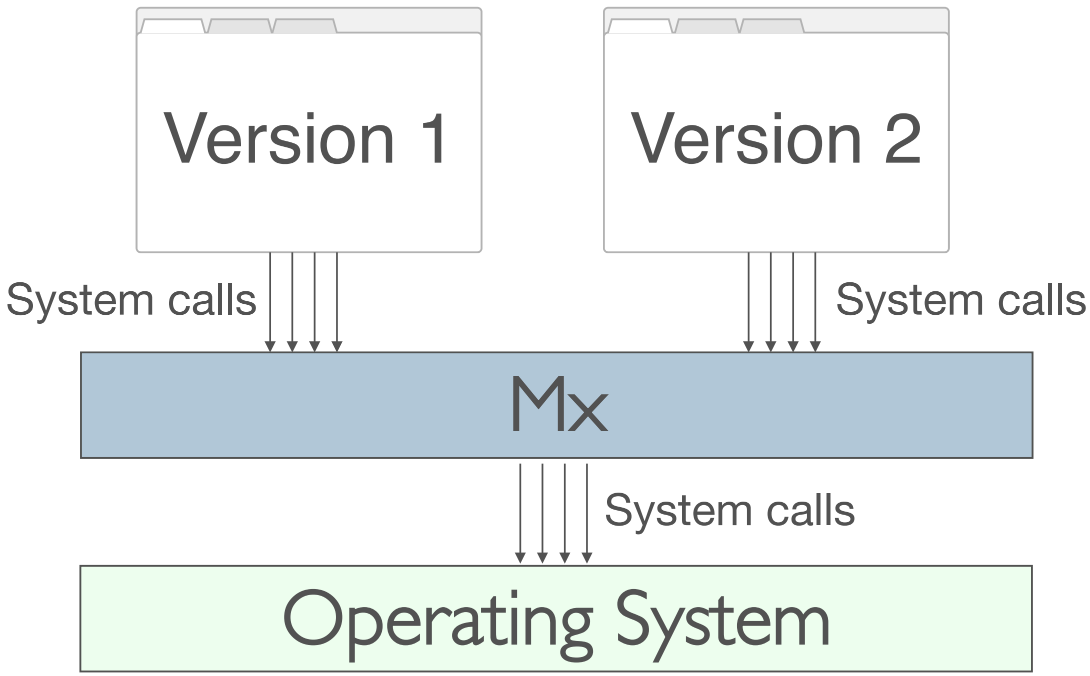
Synchronization

Synchronization possible at different levels of abstraction/granularity

- Application input/outputs
- Library calls
- System calls

Synchronization in Mx

Synchronization (and virtualization) at the level of system calls

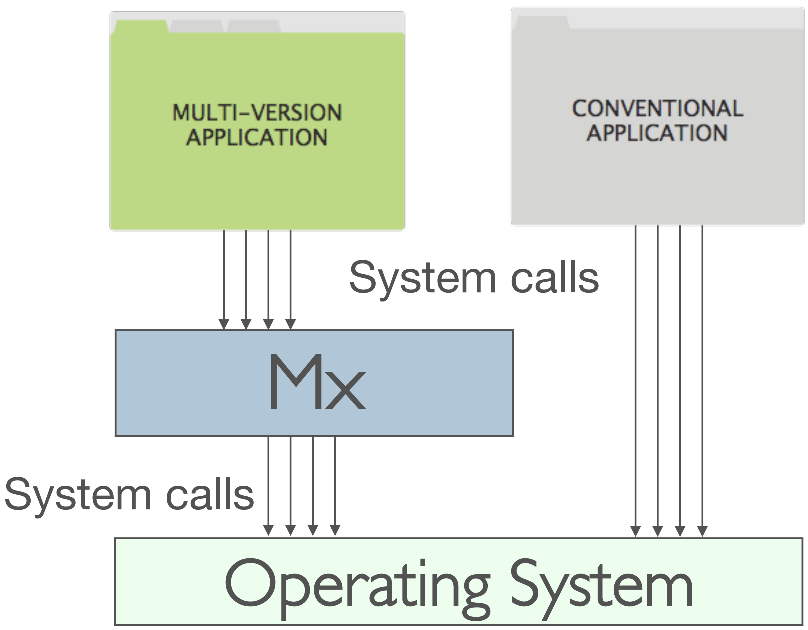


Advantages

General (not app specific)

Small number of system call types

Synchronization in Mx



Synchronization and virtualization at the level of system calls

System Calls Define External Behavior

Version 1	Version 2

Version 1

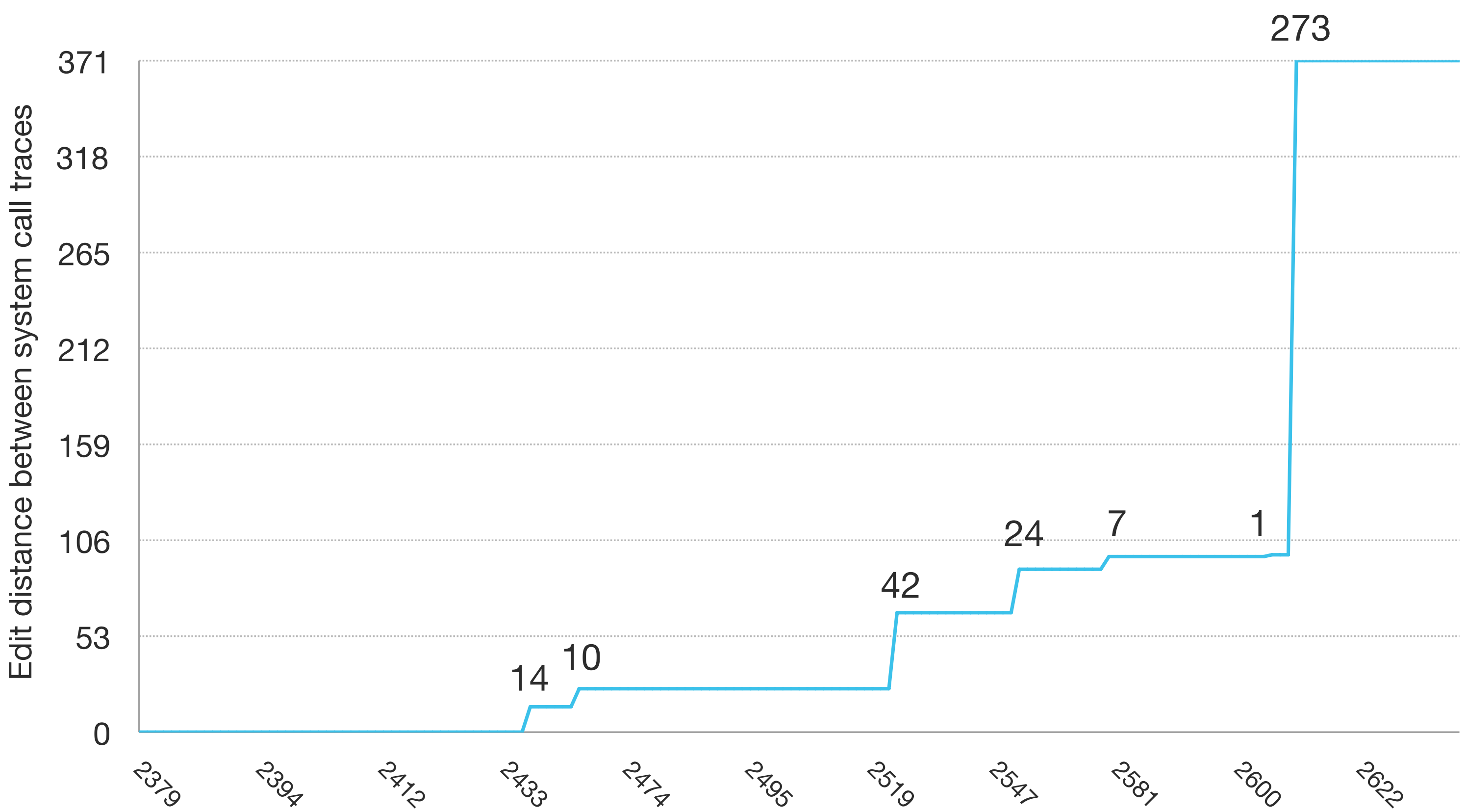
```
void pos_neg(int *a, size_t len) {
  int i, npos = 0;

  for (i=0; i<len; i++)
    if (a[i] >= 0)
      npos++;

  printf("%d\n", npos);
  printf("%d\n", len-npos);
}
```

Version 2

```
void pos_neg(int *a, size_t len) {
  int i, nneg = 0;

  for (i=len-1; i>=0; i--)
    if (a[i] < 0)
      nneg++;

  printf("%d\n", len - nneg);
  printf("%d\n", nneg);
}
```

```
int arr[] = { -3, -1, 2, -4 };
pos_neg(arr, 4);

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

External Behavior Evolves Sporadically



95% of lighttpd 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 [ICSE'13])

Challenge 2: Handling Divergences

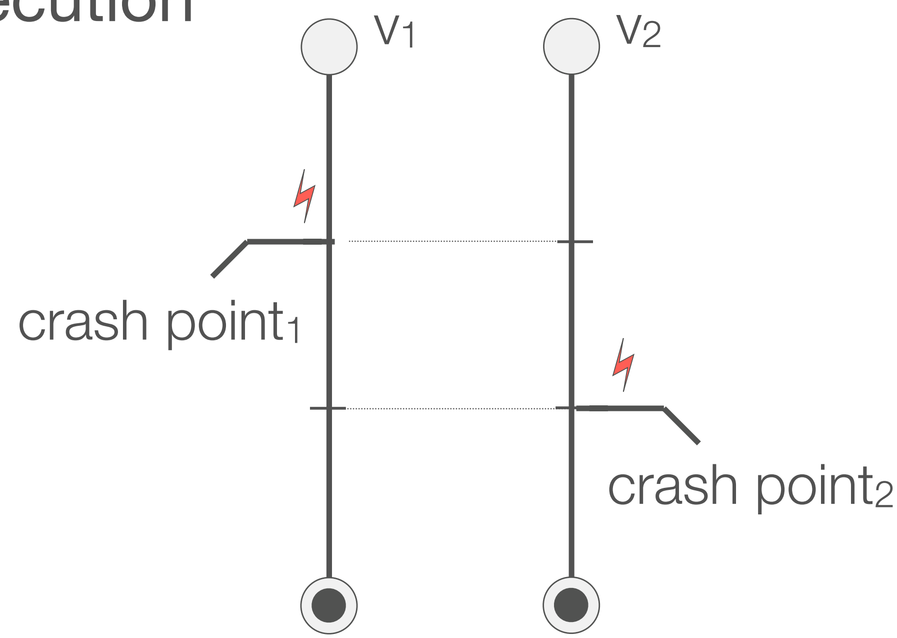
Handle divergences across versions

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

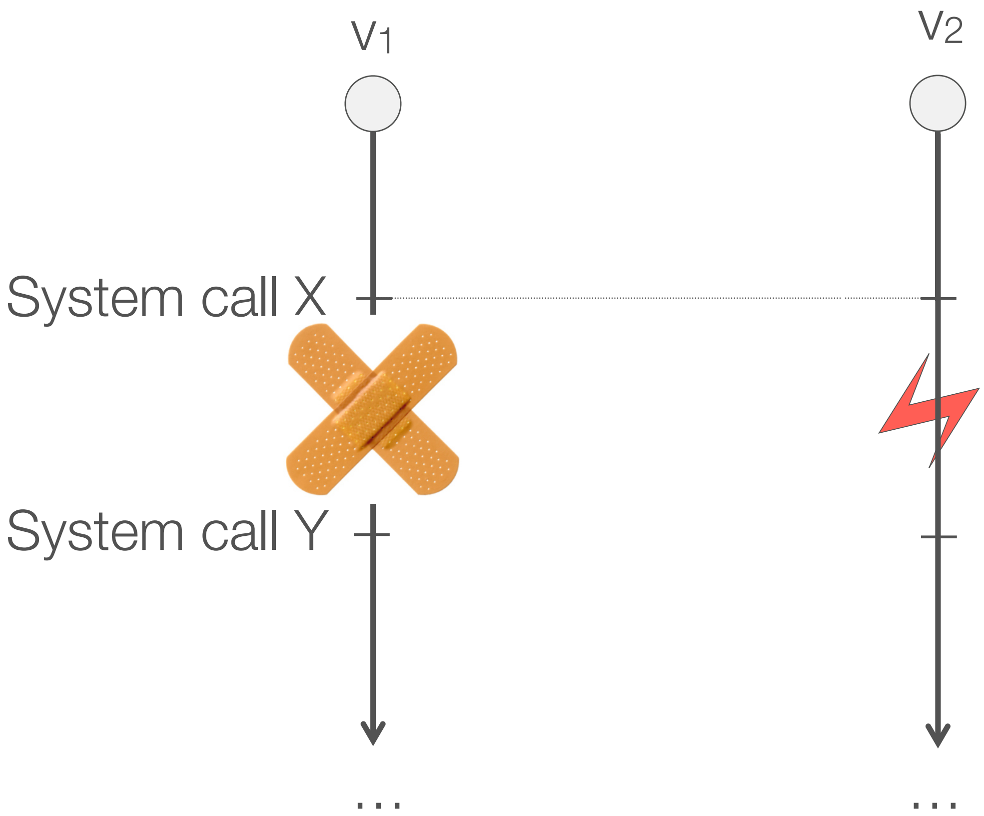
Failure Recovery: Scope

Focus exclusively on crashes

For other types of divergences, we switch to single-version execution



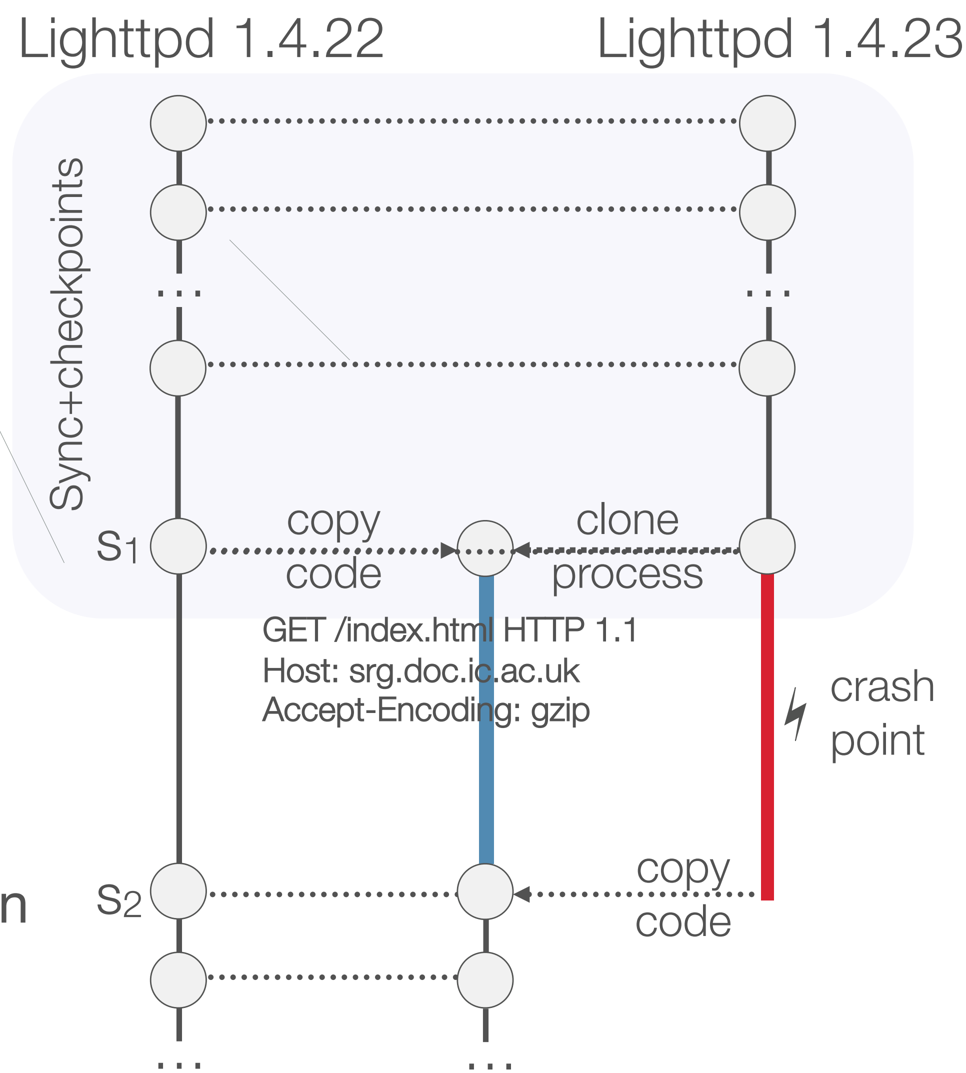
Failure Recovery: Runtime Code Patching



Failure Recovery Process

"runtime code patching"

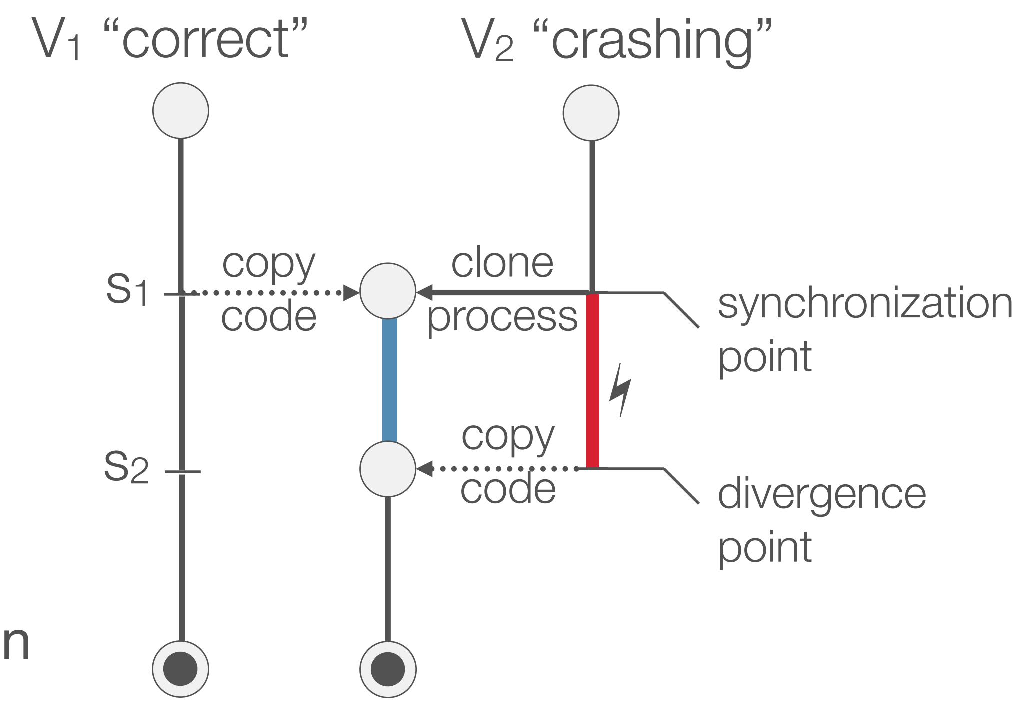
1. Revert to last successful synchronization point
2. Copy code from "correct" version
3. Run patched code to divergence point
4. Revert back to original code
5. Restart multi-version execution



Failure Recovery Process

"runtime code patching"

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



Failure Recovery: Suitable Scenarios

Errors with a small propagation distance

- “Localized” around a small portion of code

Applications which provide “natural” synchronization points

- E.g., servers structured around a main dispatch loop

Changes which do not affect memory layout

- E.g., refactorings, security patches

Where reliability is more important than performance

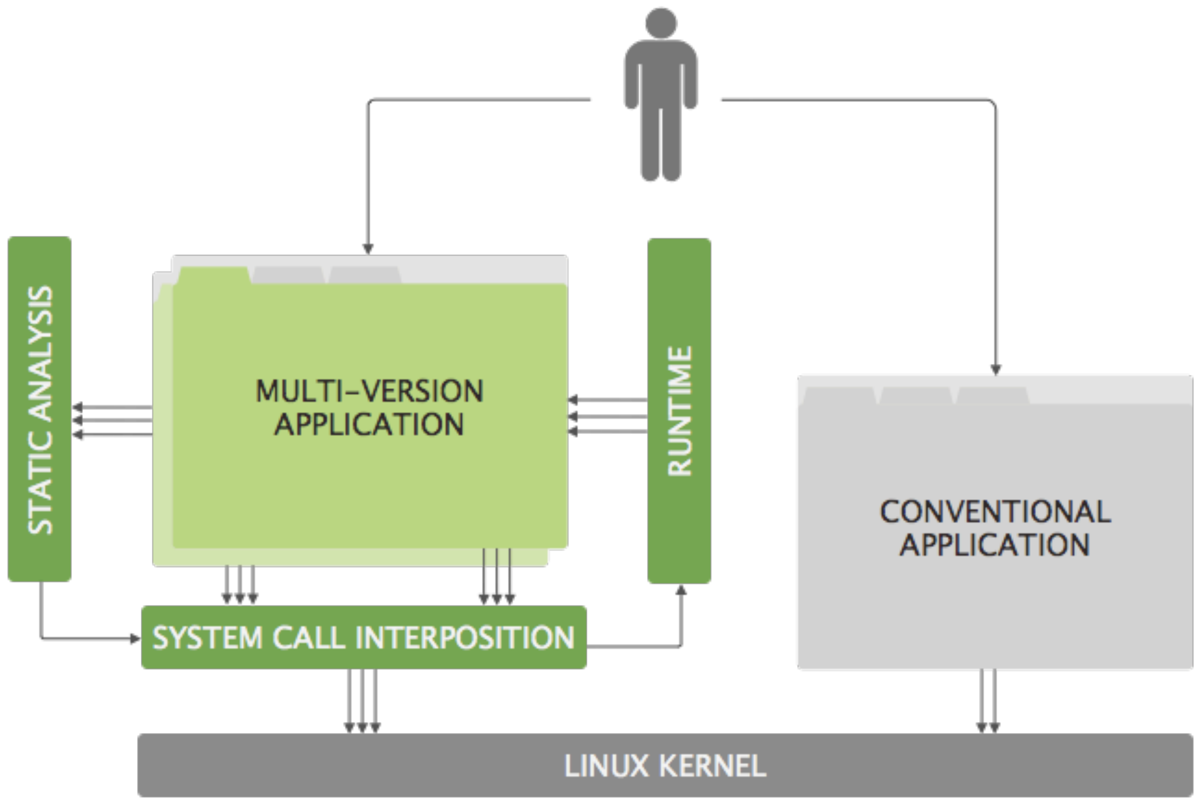
- E.g., interactive apps, some server scenarios

Failure Recovery: Guarantees?

Assumes that recovery is successful if versions exhibit the same external behavior after recovery

- If unrecoverable, Mx continues in single-version mode, using the non-crashed version
- (By design, Mx does not attempt to survive errors it cannot handle)

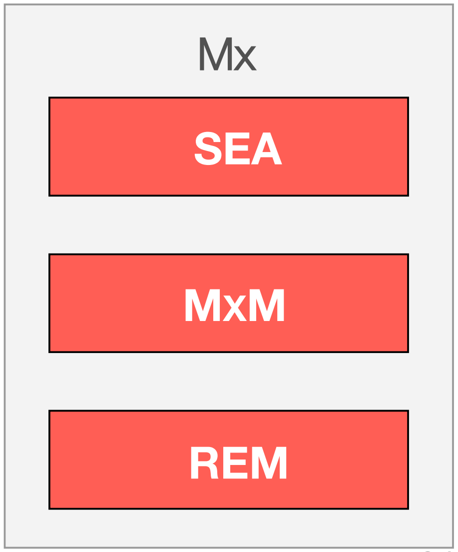
Mx Prototype



Mx Prototype

Targets multi-core processors

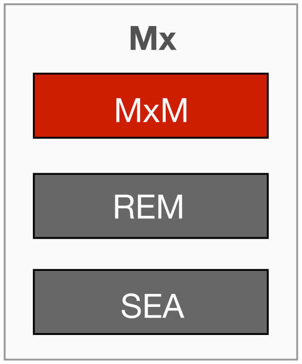
- Support for x86 and x86-64 Linux systems
- Combines binary static analysis, system call interposition, OS-level checkpointing, and runtime code patching
- Completely transparent, runs on unmodified binaries
- Currently limited to two versions



MxM: Multi-eXecution Monitor

Execute and monitor multi-version applications

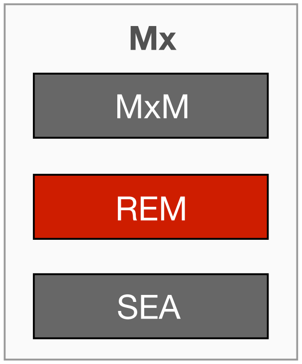
- Synchronization at the level of syscalls
  - System call interception (via `ptrace` interface)
  - Semantic comparison of syscall invocations (handles ASLR, etc.)
- Environment virtualization
  - E.g., files, sockets, pid’s
- Support for multi-threaded applications
  - One monitor instance per pair of threads



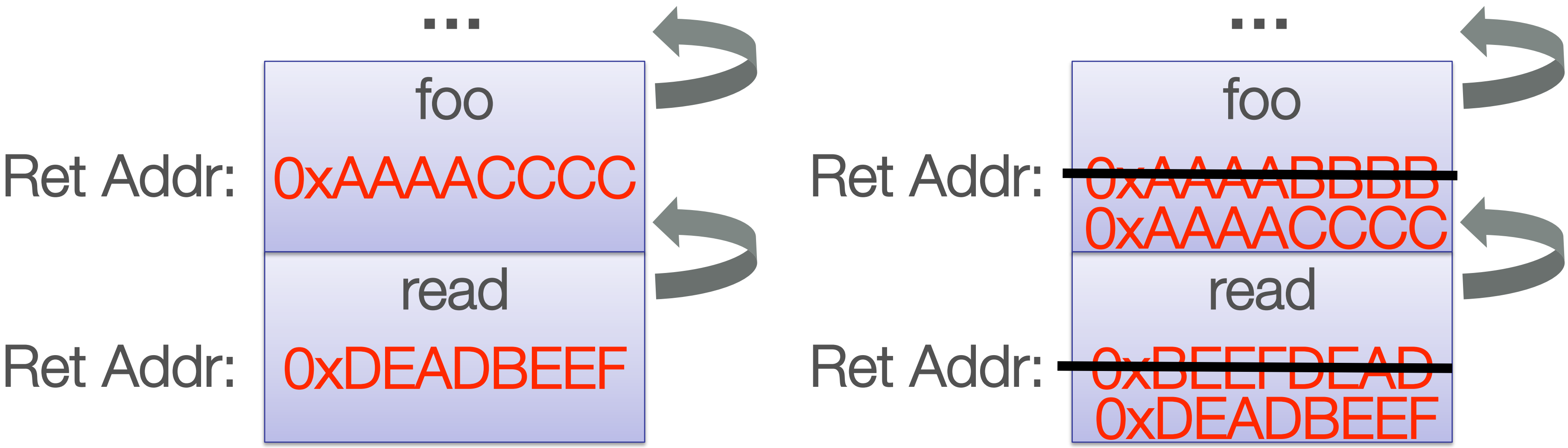
REM: Runtime Execution Manipulator

Runtime code patching and fault recovery

- OS-level checkpointing (using clone syscall)
- Code segment replacement
- Runtime stack manipulation
- Breakpoint insertion and handling (for indirect fun calls)



REM: Stack Patching



Version 1

0xDEADBEEF:

```
void foo() {
    ...
    write(1, buf, 3);
    ...
}
```

Version 2 (patched)

0xBEEFDEAD:

```
void foo() {
    ...
    write(1, buf, 3);
    ...
}
```

REM: Indirect Calls

Version 1

0x012345678:

```
fptr = bar;
...

void bar(int x) {
    ...
}

void foo() {
    ...
    fptr(1);
    ...
}
```

Memory

fptr: 0x12345678

Version 2 (patched)

0x87654321:

```
fptr = bar;
...

void bar(int x) {
    INT 3
    ...
}

void foo() {
    INT 3
    ...
    fptr(1);
    ...
}
```

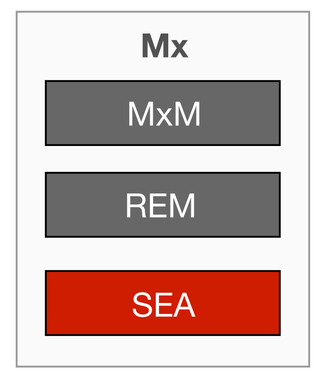
Memory

fptr: 0x876543210

SEA: Static Binary Analyzer

Create various mappings between the two version binaries

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



Evaluation

Survived a number of crash bugs in two popular servers

LIGHTTPD
fly light.

redis

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

Evaluation: survived several crash bugs

	Application	Bug
INTERACTIVE APPS	md5sum sha1sum	Buffer overflow
	mkdir mkfifo mknod	NULL-ptr dereference
	cut	Buffer overflow
SERVERS	lighttpd #1	Loop index underflow
	lighttpd #2	Off-by-one error
	redis	Missing return

redis

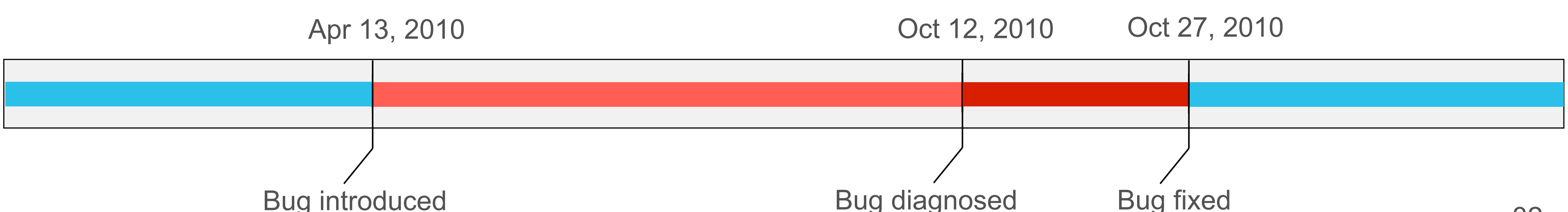
HMGET command hmgetCommand function

```
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));
```

Refactor

```
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);
    }
}
```

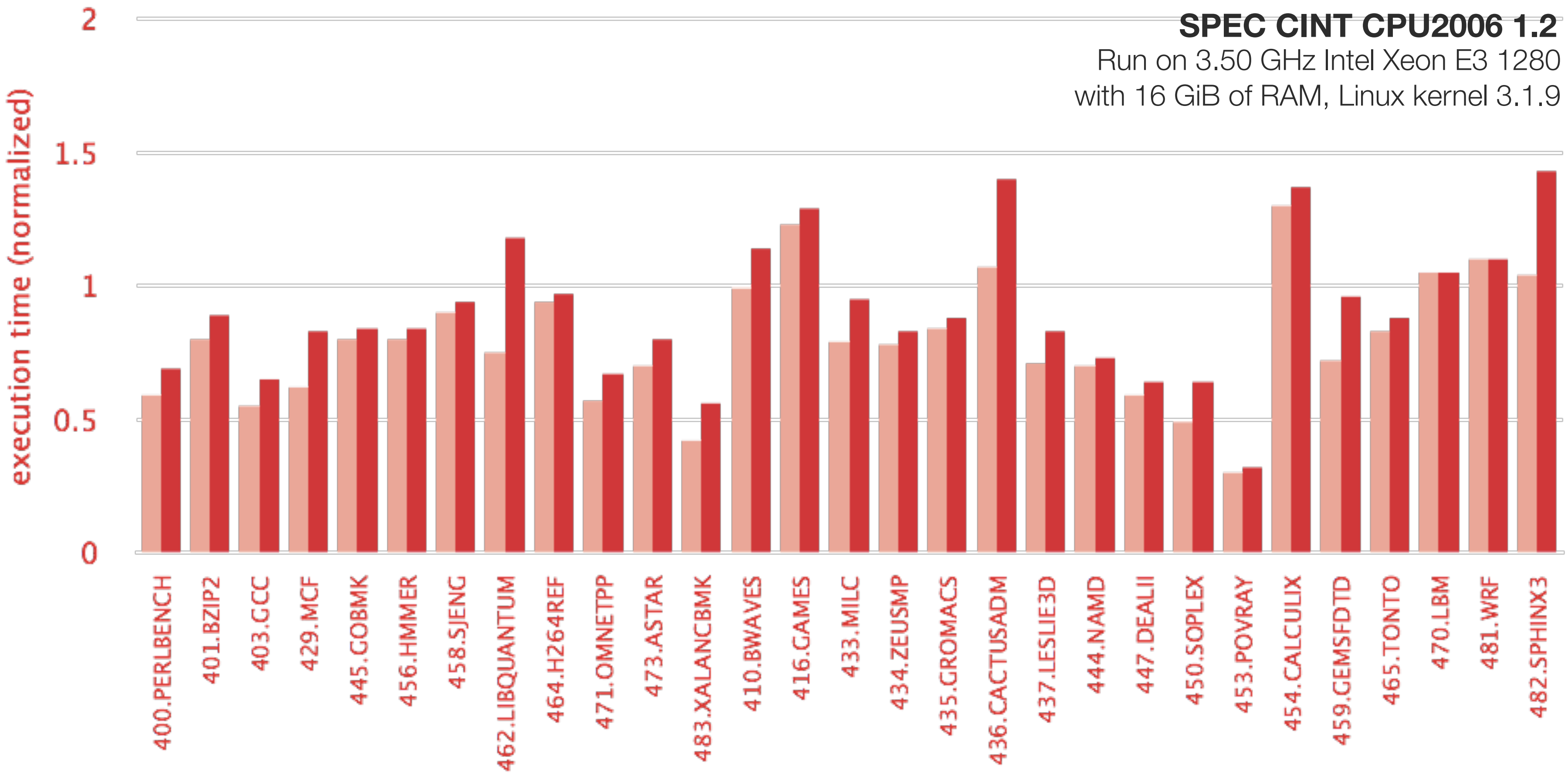
Bug may result in loosing some or even all of the stored data



Maximum distance between versions

Application	Version span	Time span
md5sum sha1sum	1,124 revs	1 year 7 months
mkdir mkfifo mknod	2,937 revs	> 4 years
cut	1,201 revs	2 years 3 months
lighttpd #1	87 revs	2 months 2 days
lighttpd #2	12 revs	2 months 1 day
redis #344	27 revs	6 days

17.81% overhead on SPEC INT CPU 2006



Describes one type of applications (CPU bound)

Allows comparison with other runtime techniques

Performance: Interactive and Server Apps

INTERACTIVE APPS

Utility	Max input size	Overhead
md5sum sha1sum	1.25 MB	< 100ms (imperceptible)
mkdir mkfifo mknod	115 nested directories	
cut	1.10 MB	

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

SERVERS

Application	Version span	Overhead
lighttpd	different continents same machine	1.01x – 1.04x 2.60x – 3.49x
redis	different continents same machine	1.00 – 1.05x 3.74 – 16.72x

Measured using http_load and redis_benchmark (default workload)
Run on 3.50 GHz Intel Xeon E3 1280 with 16 GB 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*

Online validation of 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

Mx: Parallel execution of manually-evolved versions, focus on surviving errors at runtime: HotSWUp'12, ICSE'13

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. *ICSE'13*

Mx: Safe Software Updates via MV Exec

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

- <u>Better performance</u>
  - Kernel modules, system call rewriting, skipping safe code, etc.
- <u>Support for more complex code changes & divergences</u>
  - Automatic stack reconstruction, inference of data structure changes, epoch-based system call record & replay

Can multiple software versions be effectively combined to increase software reliability and security?