JMVX

Fast Multi-threaded Multi-version Execution and Record Replay for Managed Languages

David Schwartz, Ankith Kowshik, and Luís Pina
University of Illinois Chicago

10/23/2024 - OOPSLA

JMVX

Fast Multi-threaded Multi-version Execution and Record Replay for Managed Languages

David Schwartz, Ankith Kowshik, and Luís Pina
University of Illinois Chicago

10/23/2024 - OOPSLA

Record Replay (RR)

```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```

Record Replay (RR)

```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```

Developer

User

Record Replay (RR)

```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```

Developer

User

User doesn't have a config file, leads to:

Record Replay (RR)

```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```

Developer

User

User doesn't have a config file, leads to:

```
Exception … java.io.FileNotFoundException: …
```

Record Replay (RR)

```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```

Developer

User

User doesn't have a config file, leads to:

```
Exception … java.io.FileNotFoundException: …
```

File [poorly documented] report to the developer

Record Replay (RR)

```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```

Developer

Developer has a config file.
Program runs fine!

User

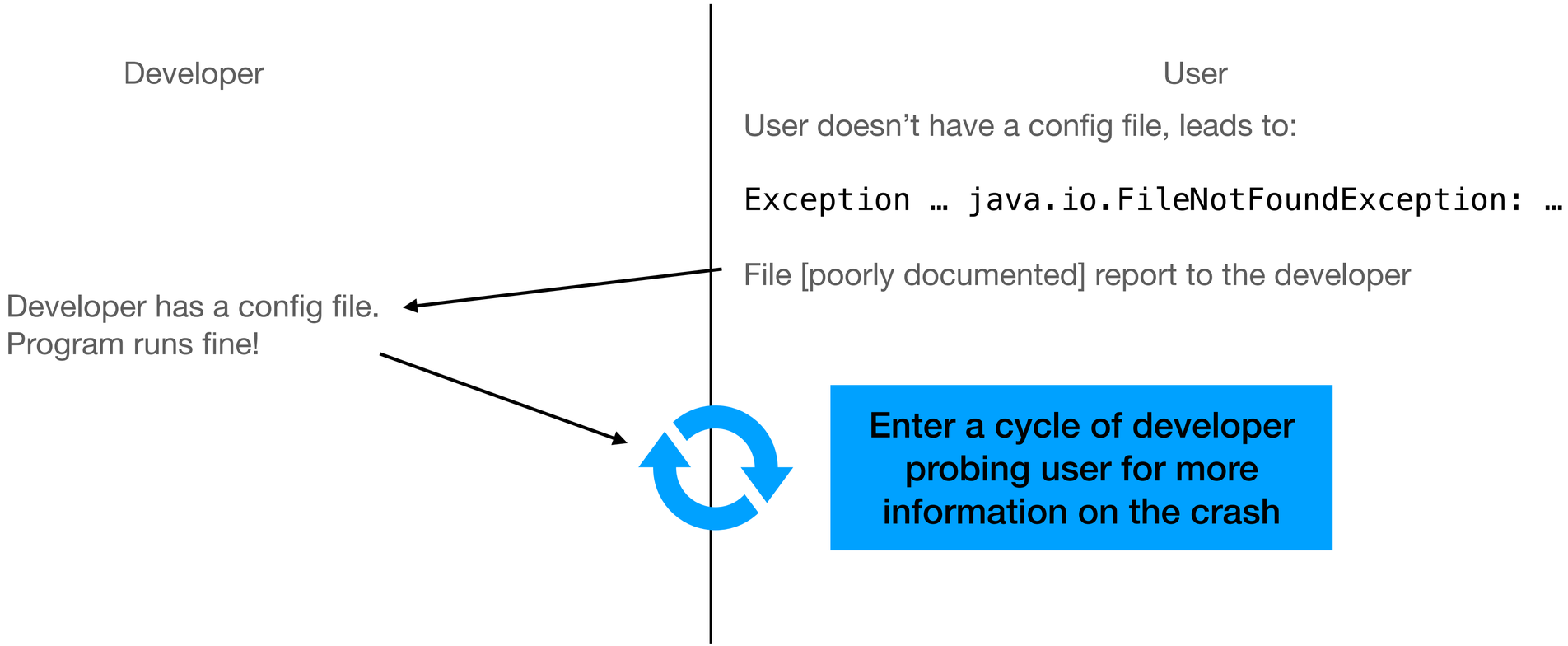
User doesn't have a config file, leads to:

`Exception … java.io.FileNotFoundException: …`

File [poorly documented] report to the developer

Record Replay (RR)

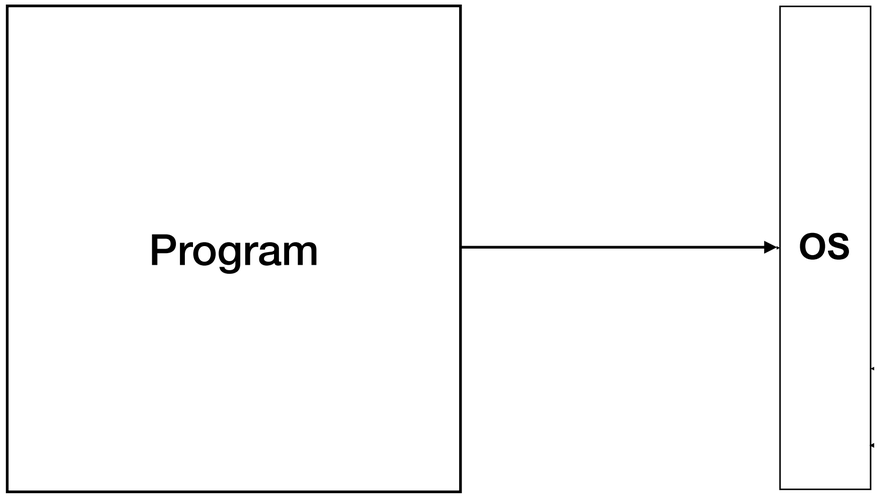
```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```



Record Replay (RR)

```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```

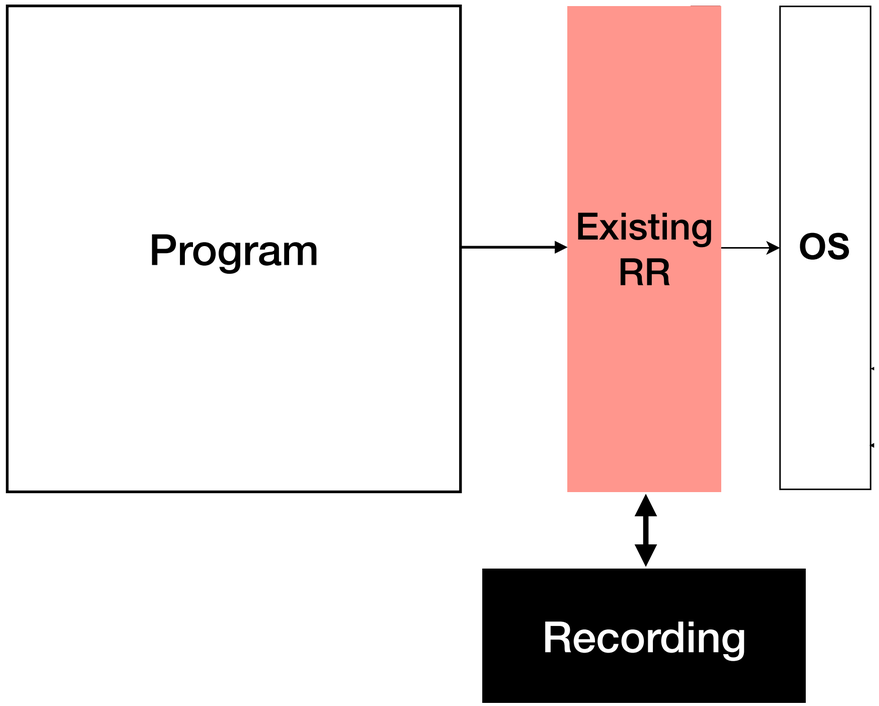
Normally, the program interacts directly with the OS.
Developer is unaware of the computing environment.



Record Replay (RR)

```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```

With Record Replay (RR), a recording of nondeterministic interactions with OS is made.



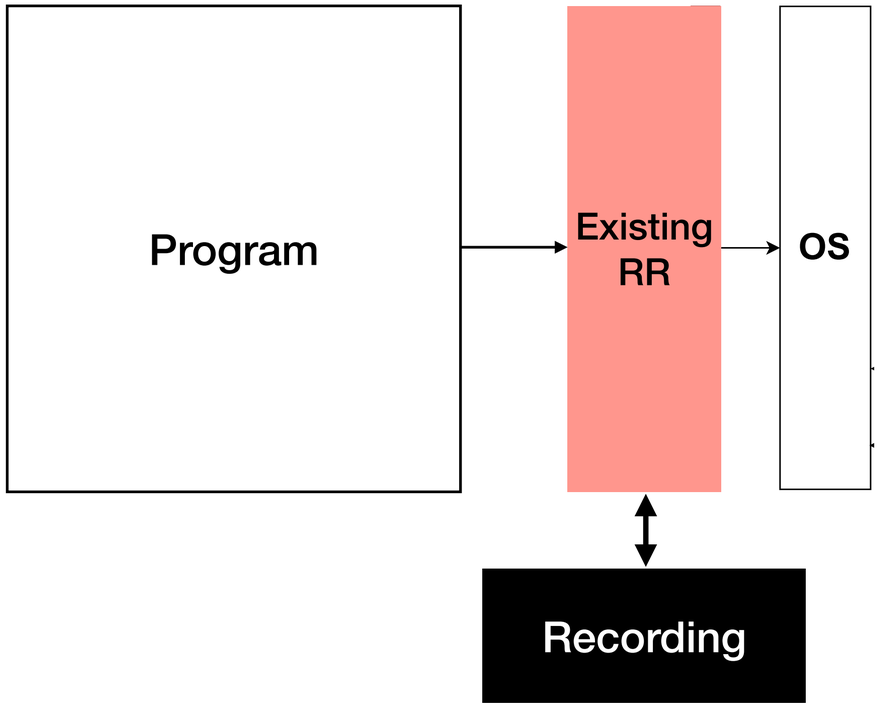
Record Replay (RR)

```
1:  public static void main(String[] args) throws Exception{
2:          FileInputStream configFile = new FileInputStream("config.conf");
3:          doWork(configFile);
4:  }
5:
```

With Record Replay (RR), a recording of nondeterministic interactions with OS is made.

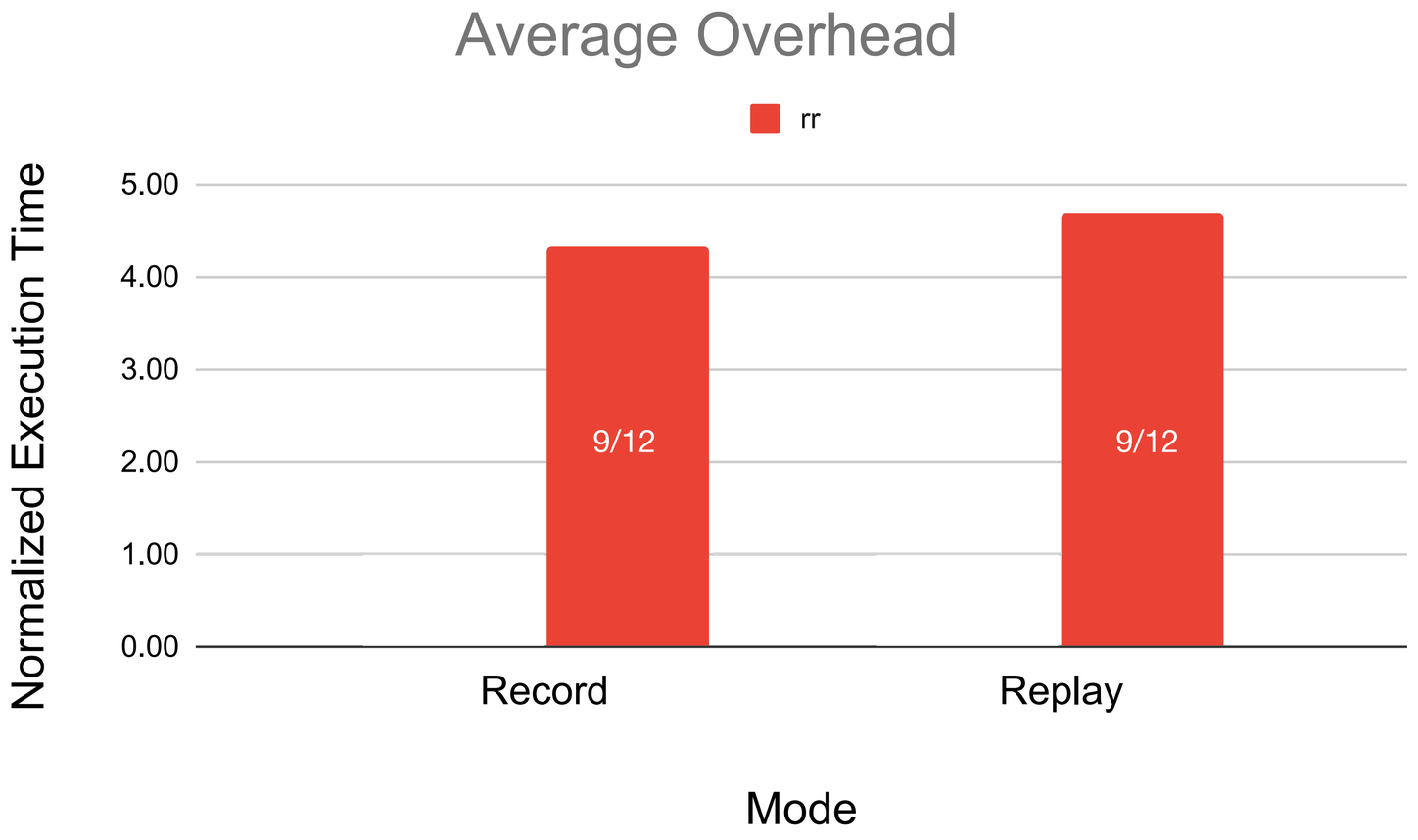
Recording can be given to a developer and replayed. This imposes the same interactions, allowing the bug to be easily reproduced.

No need to ask the user for more info



Motivation

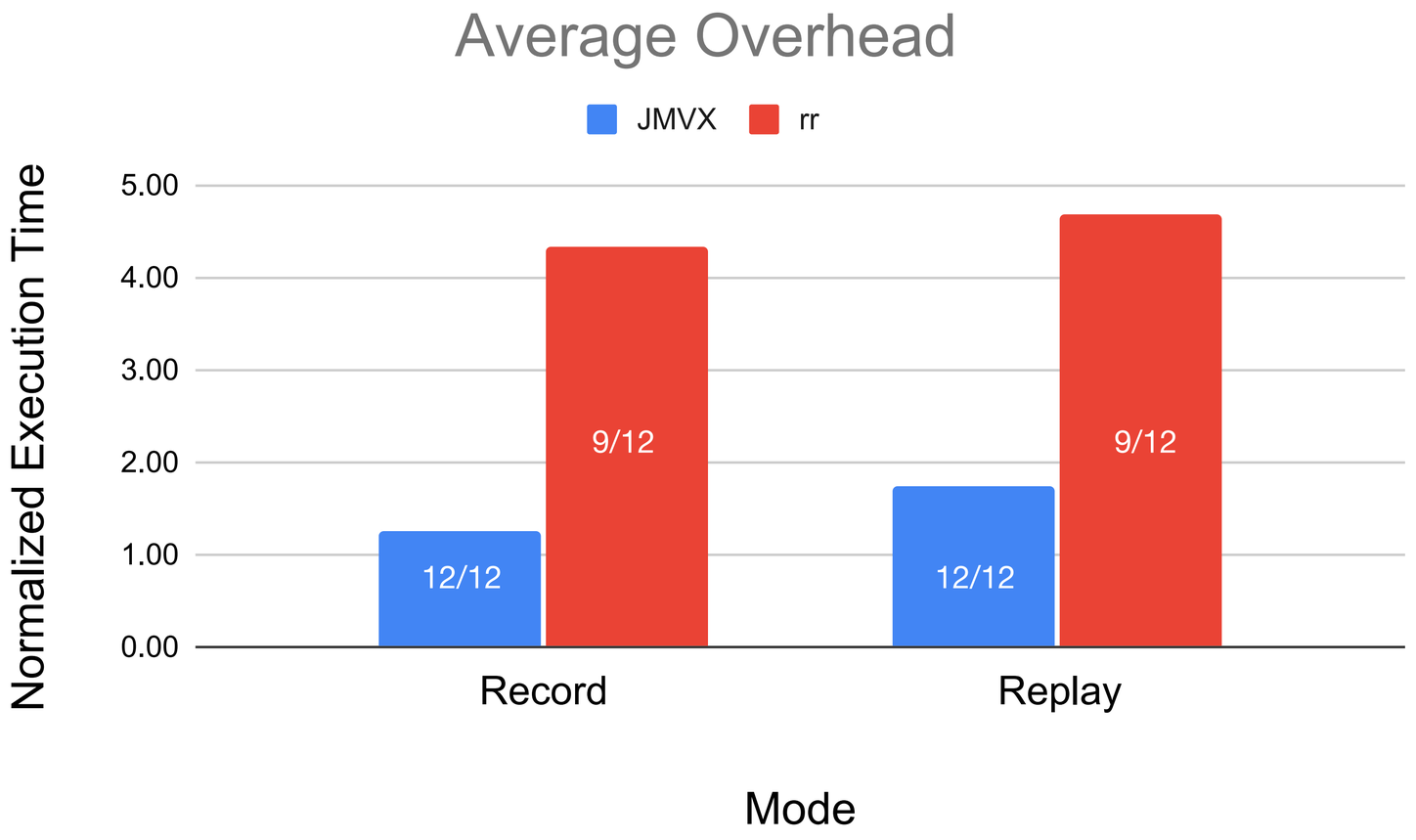
Text inside of bar:
Benchmarks passed / tested



Data is normalized to the vanilla (uninstrumented) benchmarks

Motivation

Text inside of bar:
Benchmarks passed / tested



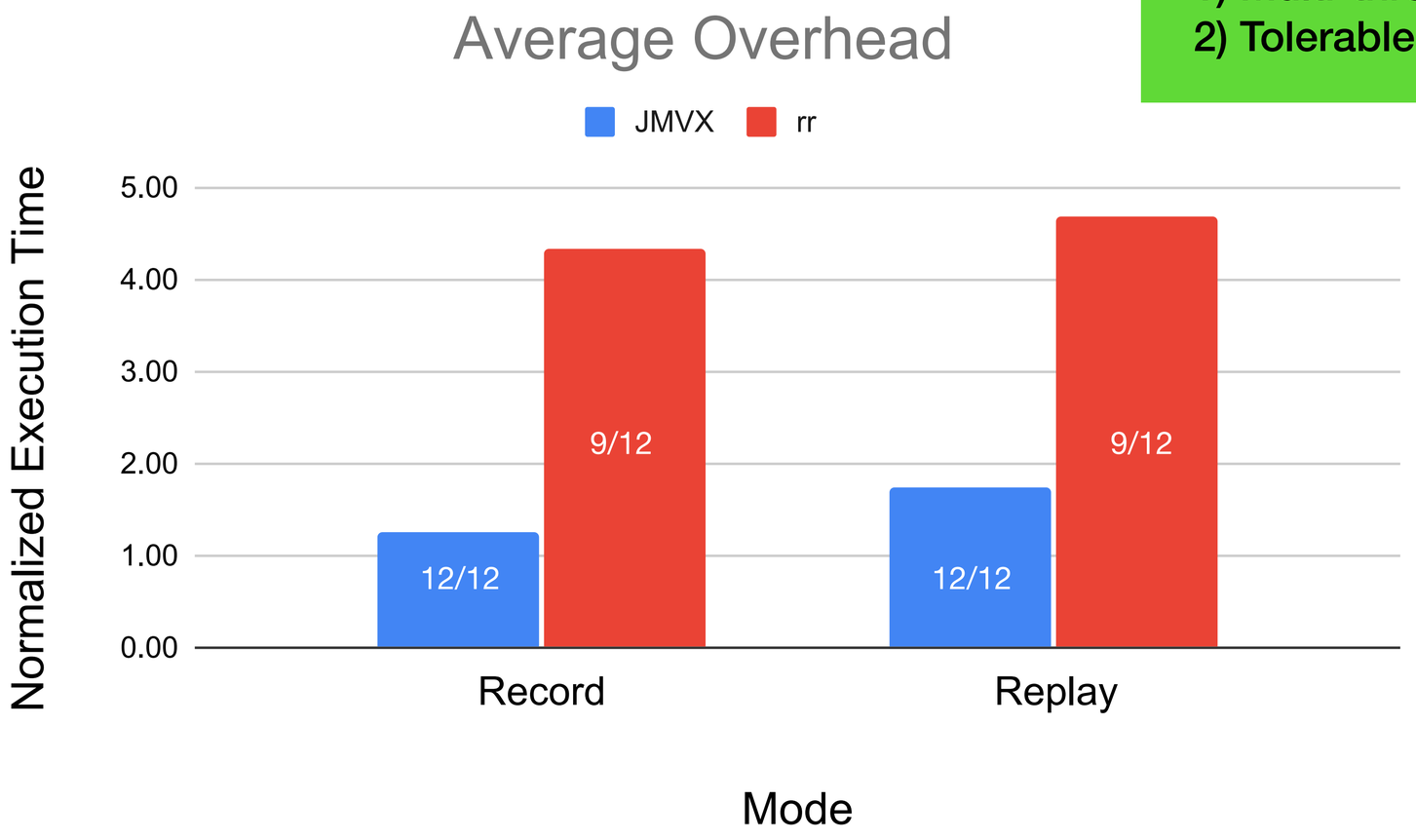
Data is normalized to the vanilla (uninstrumented) benchmarks

Authors: Schwartz, Kowshik, Pina

Motivation

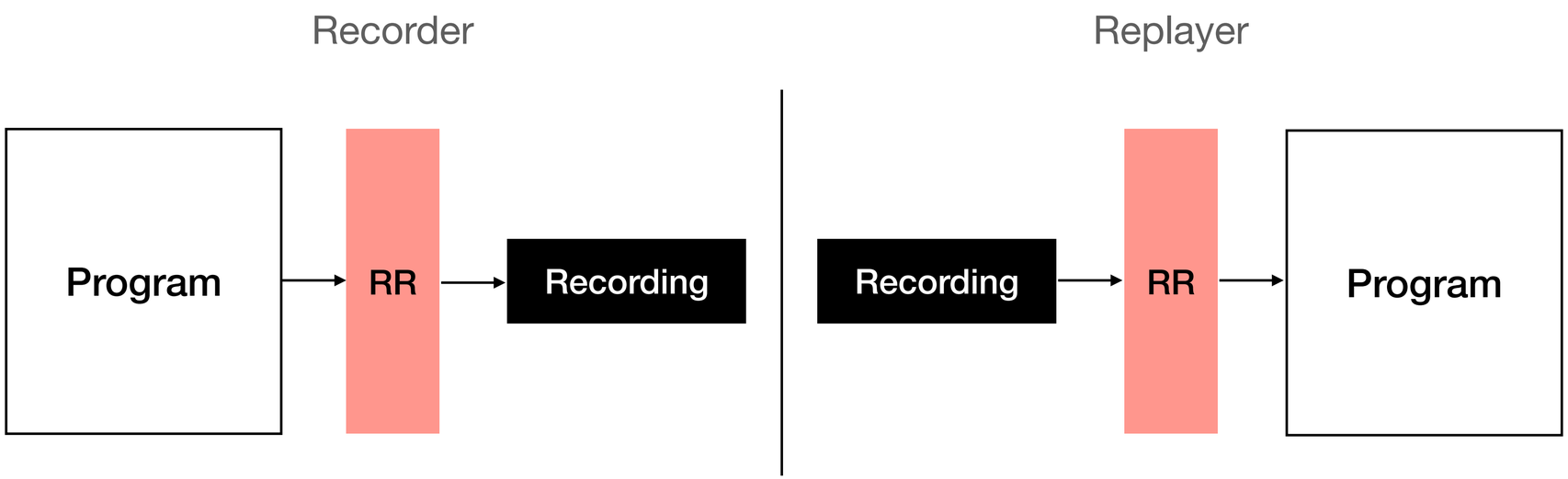
Achieved by:
1) Multi-threaded record replay
2) Tolerable benign divergences

Text inside of bar:
Benchmarks passed / tested

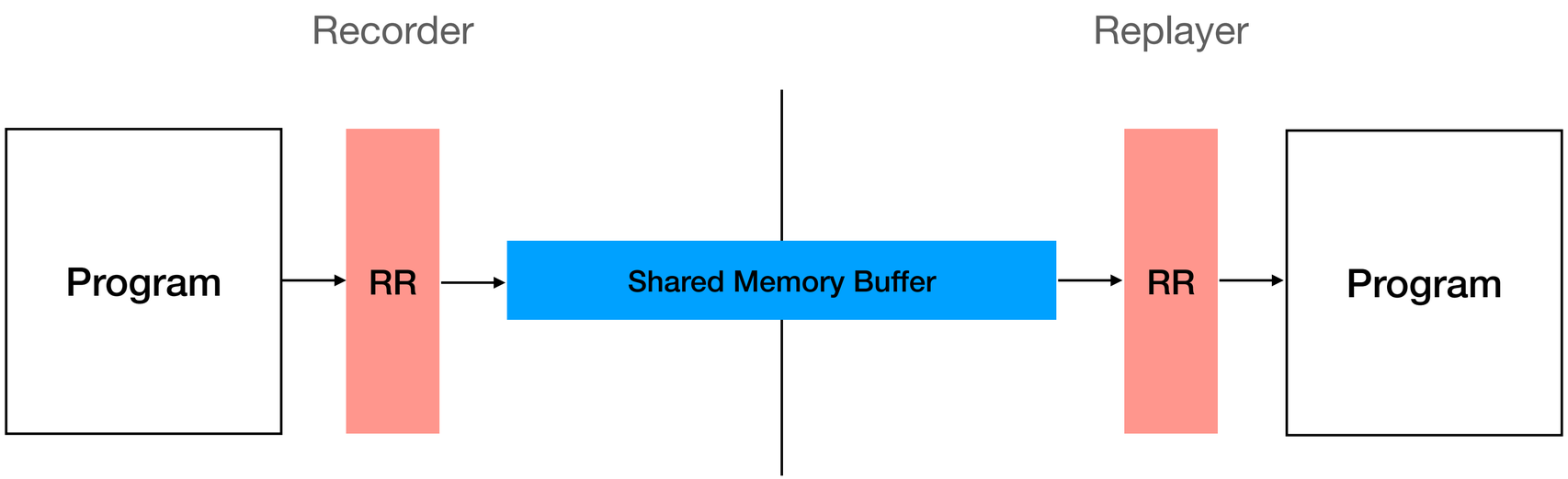


Data is normalized to the vanilla (uninstrumented) benchmarks

RR is Offline Multi-version Execution (MVX)

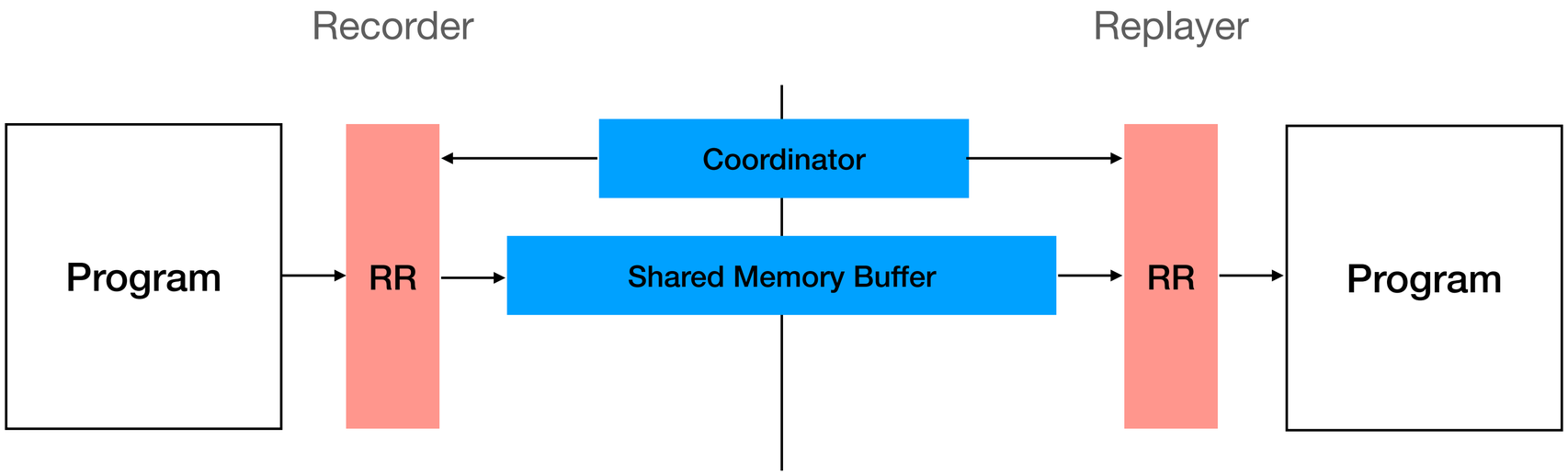


RR is Offline Multi-version Execution (MVX)



Log to a shared buffer rather than to disk

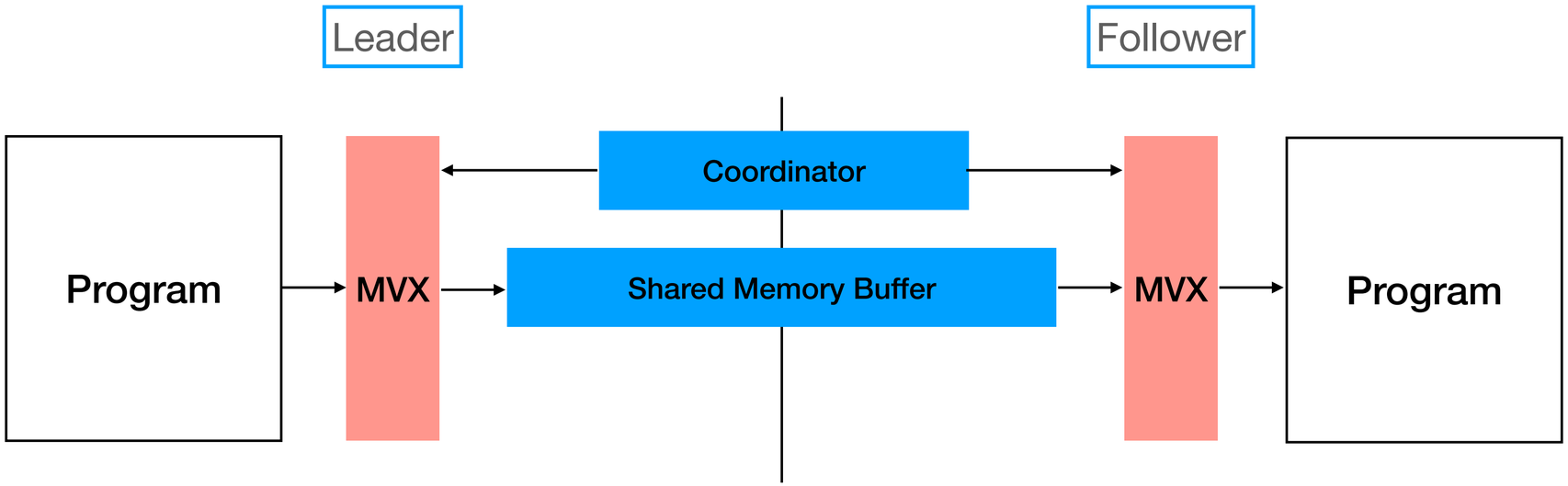
RR is Offline Multi-version Execution (MVX)



Log to a shared buffer rather than to disk

Use a coordinator process to establish communication channels and share resources

RR is Offline Multi-version Execution (MVX)

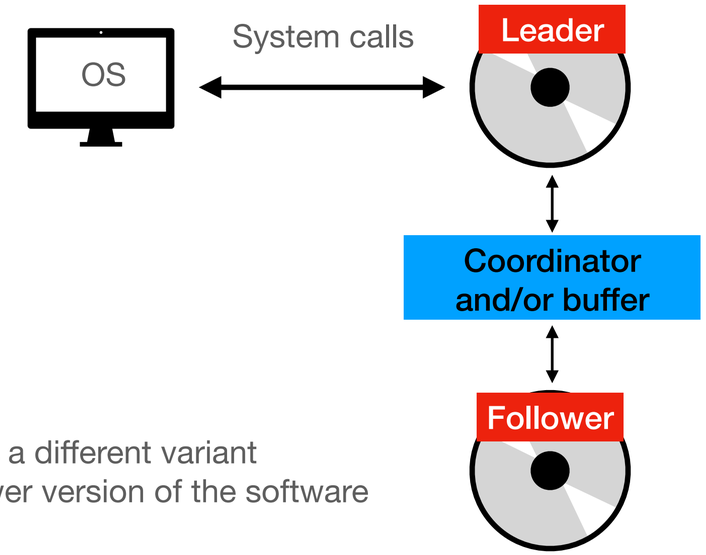


Log to a shared buffer rather than to disk

Use a coordinator process to establish communication channels and share resources

Multi-version Execution

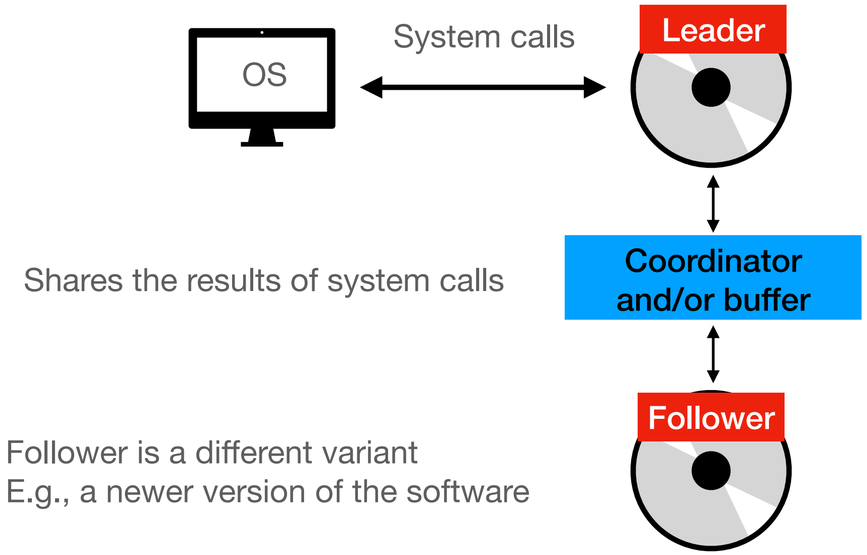
Run extra instances of our software



Follower is a different variant
E.g., a newer version of the software

Multi-version Execution

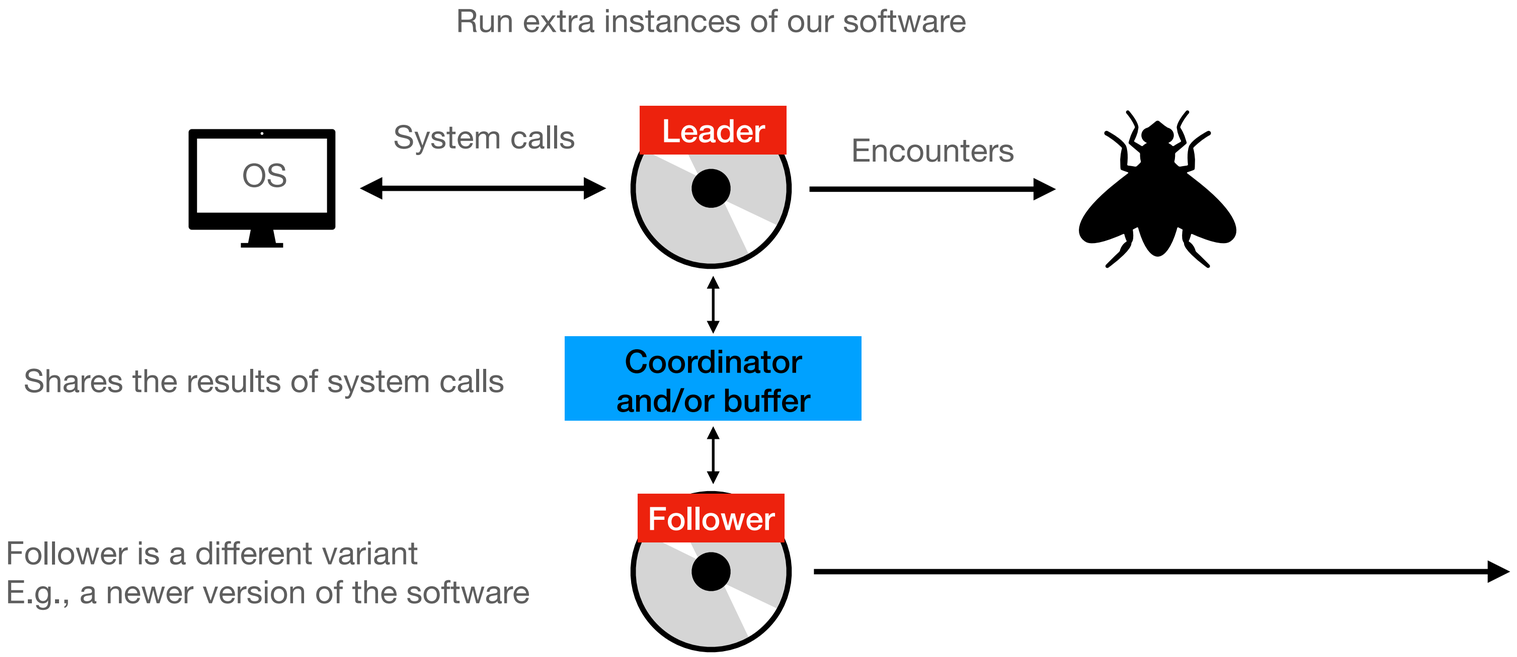
Run extra instances of our software



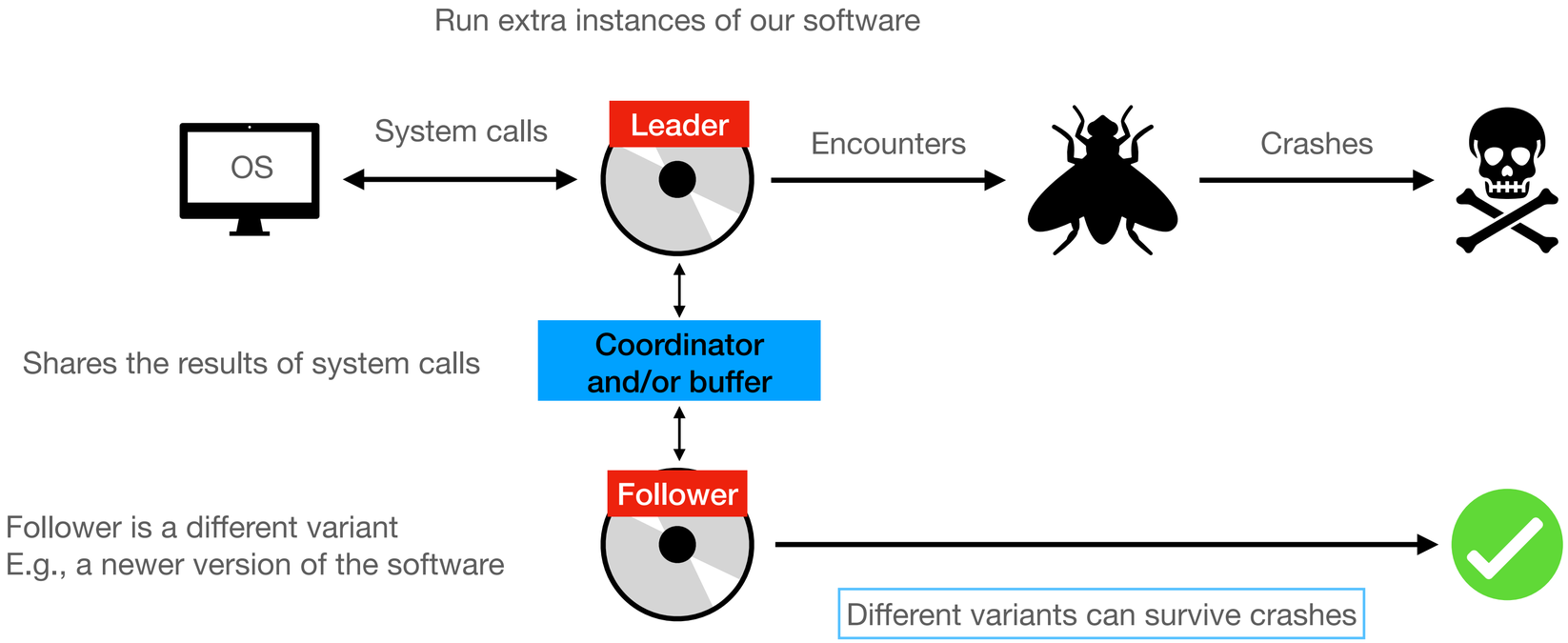
Shares the results of system calls

Follower is a different variant
E.g., a newer version of the software

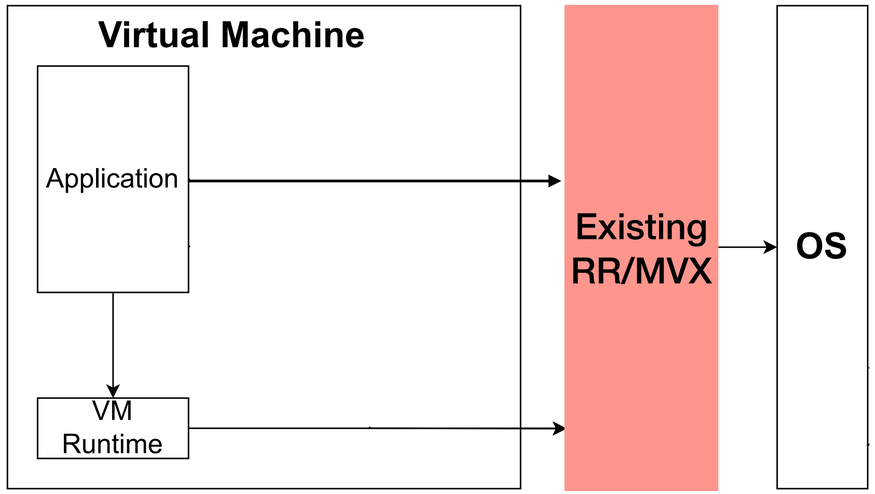
Multi-version Execution



Multi-version Execution

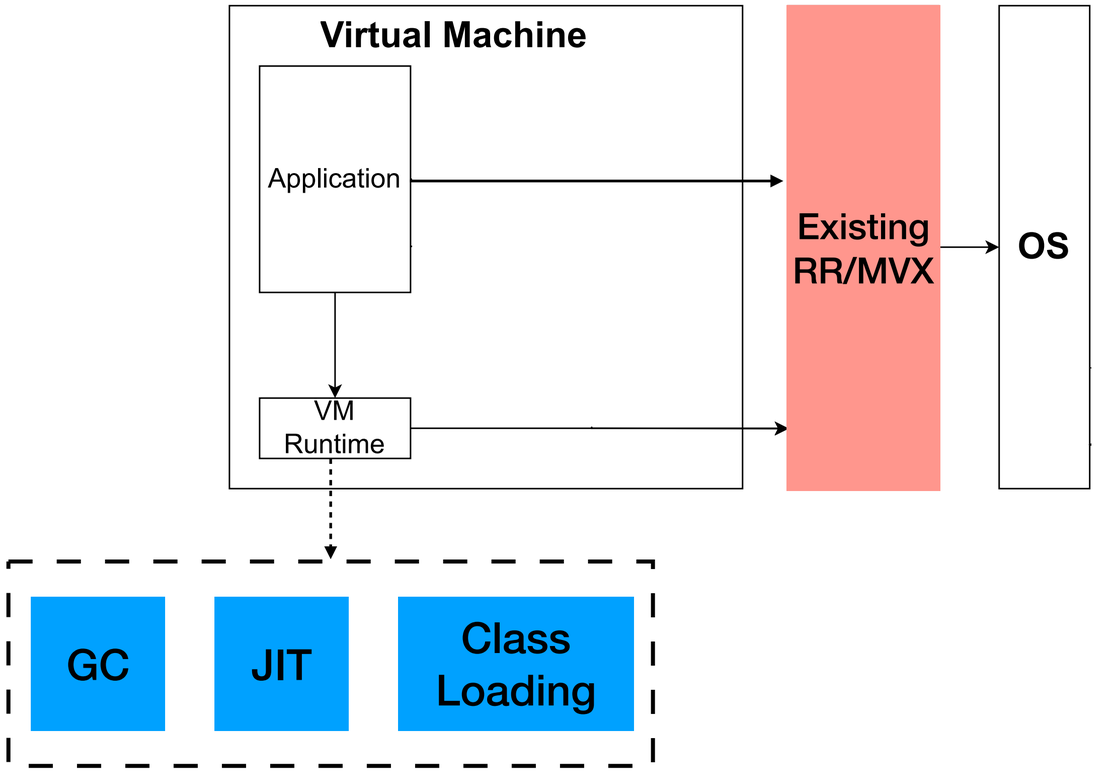


RR/MVX is Difficult for VM Based Languages



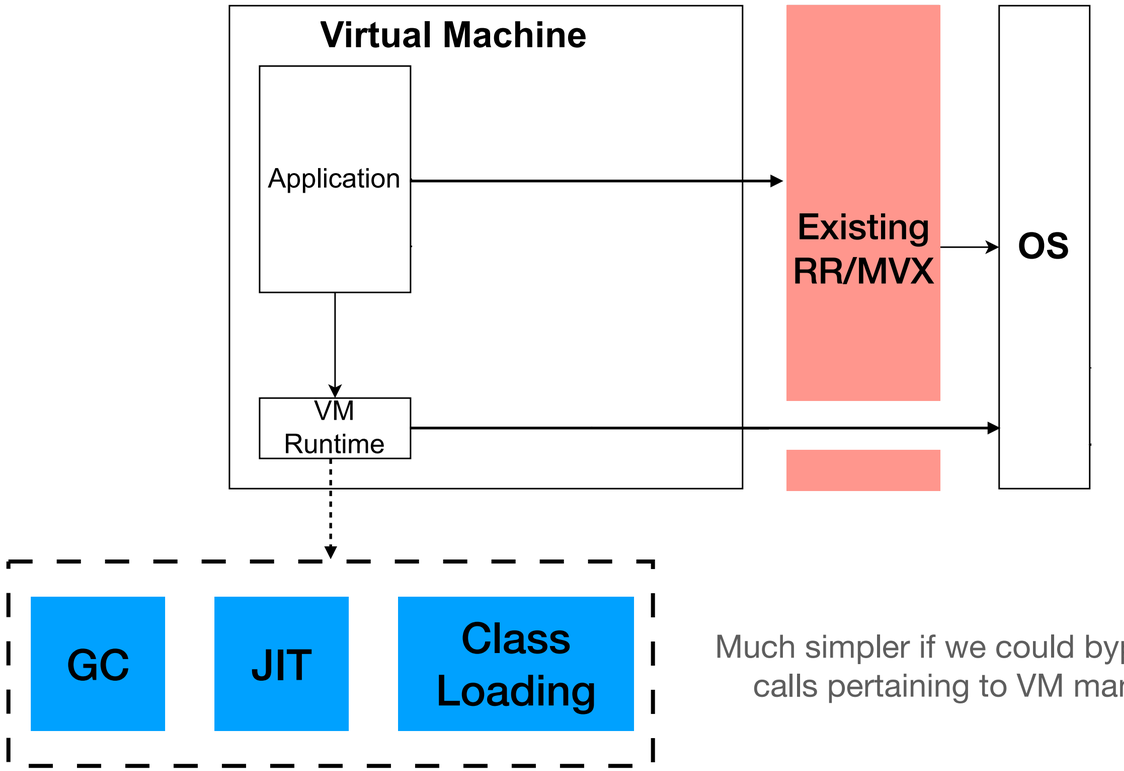
RR/MVX is Difficult for VM Based Languages

- VM management leads to divergent behavior



RR/MVX is Difficult for VM Based Languages

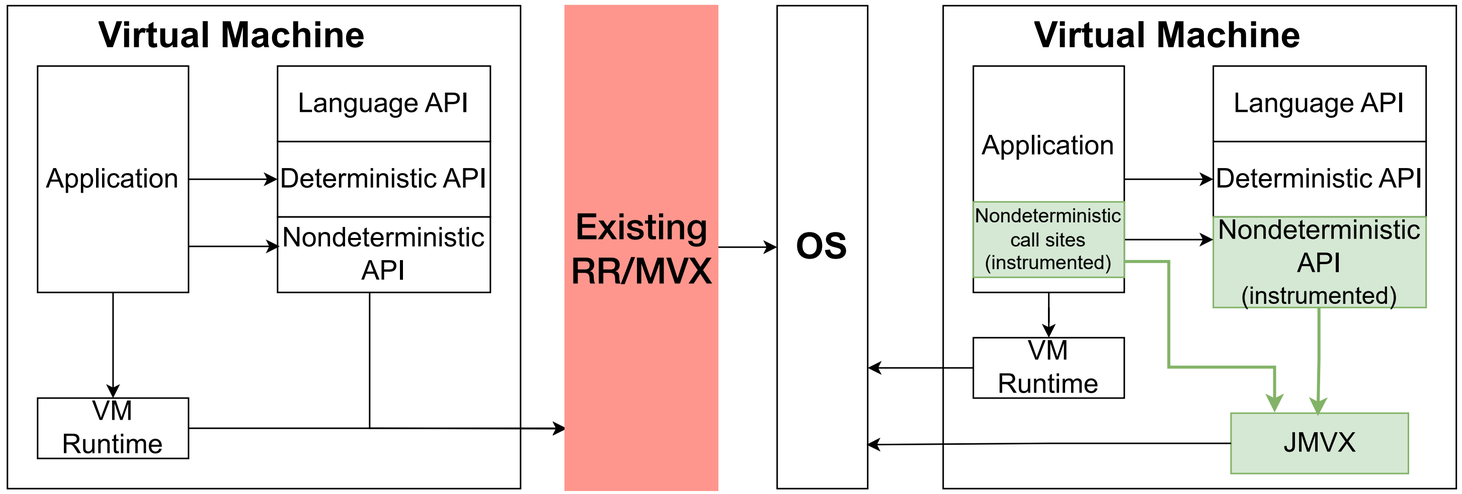
- VM management leads to divergent behavior



Much simpler if we could bypass system calls pertaining to VM management

RR/MVX is Difficult for VM Based Languages

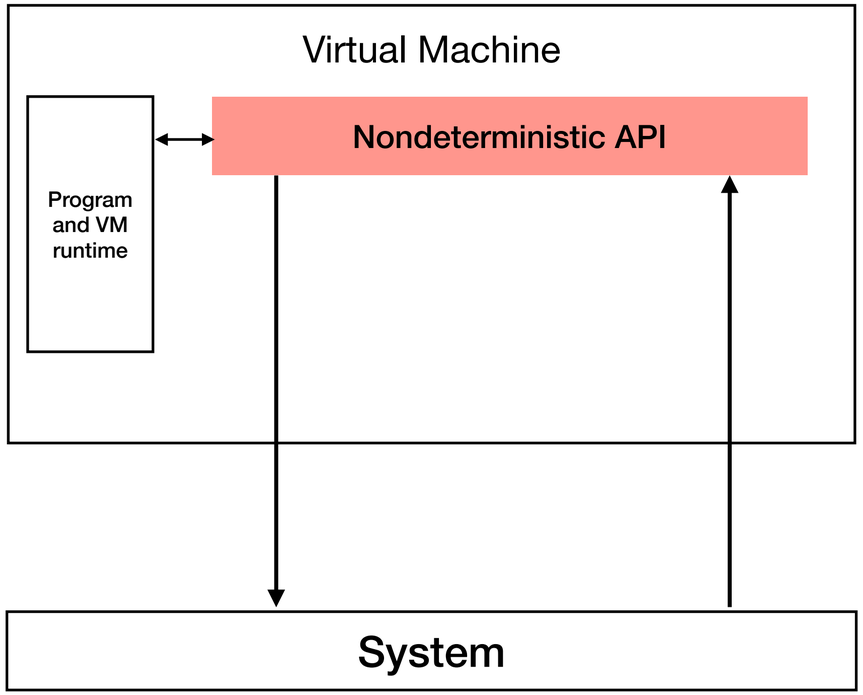
- Use a higher level of abstraction to tolerate* VM management divergences



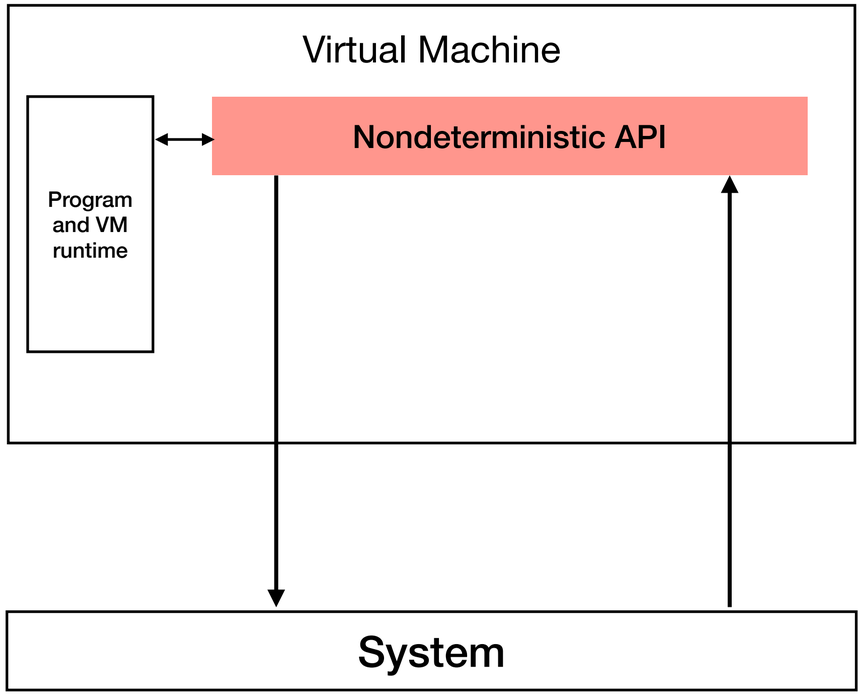
*Currently we can tolerate benign divergences from class loading and the JIT. GC can sometimes be a problem, see the paper for more details!

Much simpler if we could bypass system calls pertaining to VM management

Dynamic Tracing



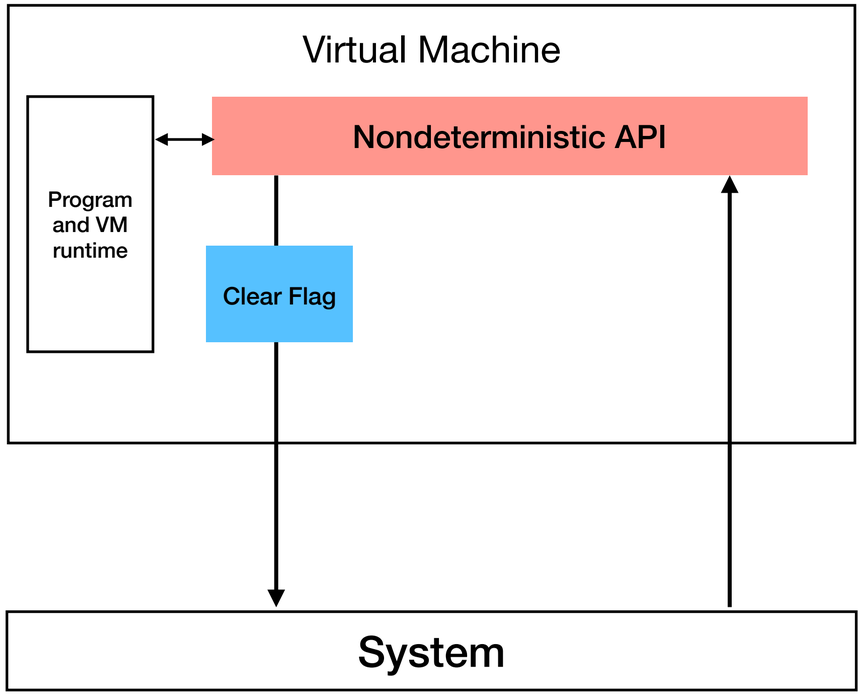
Dynamic Tracing



Instrumented bytecode to add hooks to natives

```
07: // Original code
08: nativeMethod();
09:
10: // Instrumented code
11: JMVX.beforeNative();
12: nativeMethod();
13: JMVX.afterNative();
```

Dynamic Tracing

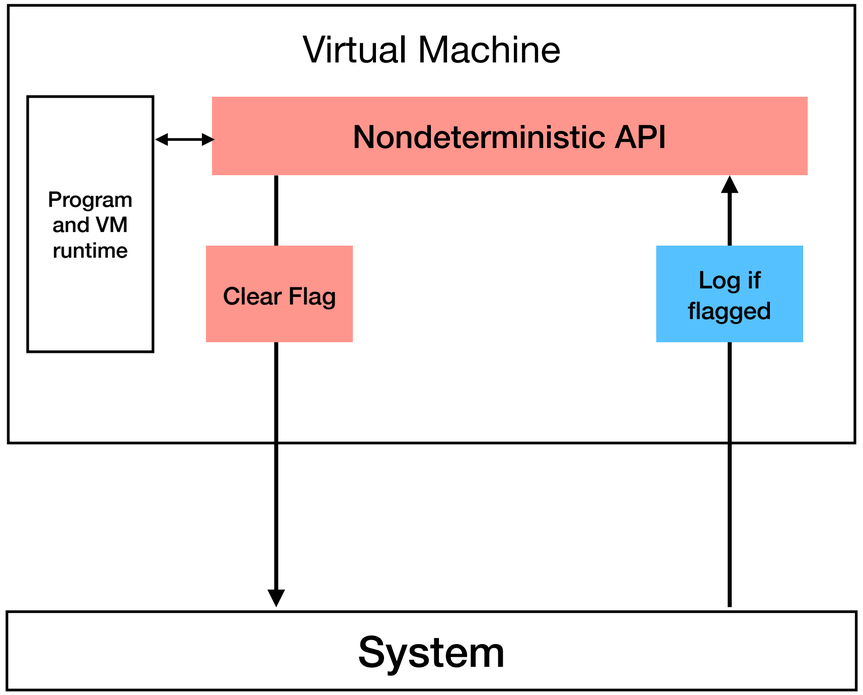


Instrumented bytecode to add hooks to natives

```
07: // Original code
08: nativeMethod();
09:
10: // Instrumented code
11: JMVX.beforeNative();
12: nativeMethod();
13: JMVX.afterNative();
```

```
17: static void beforeNative() {
18:   // Global Flag
19:   syscall = false;
20: }
```

Dynamic Tracing

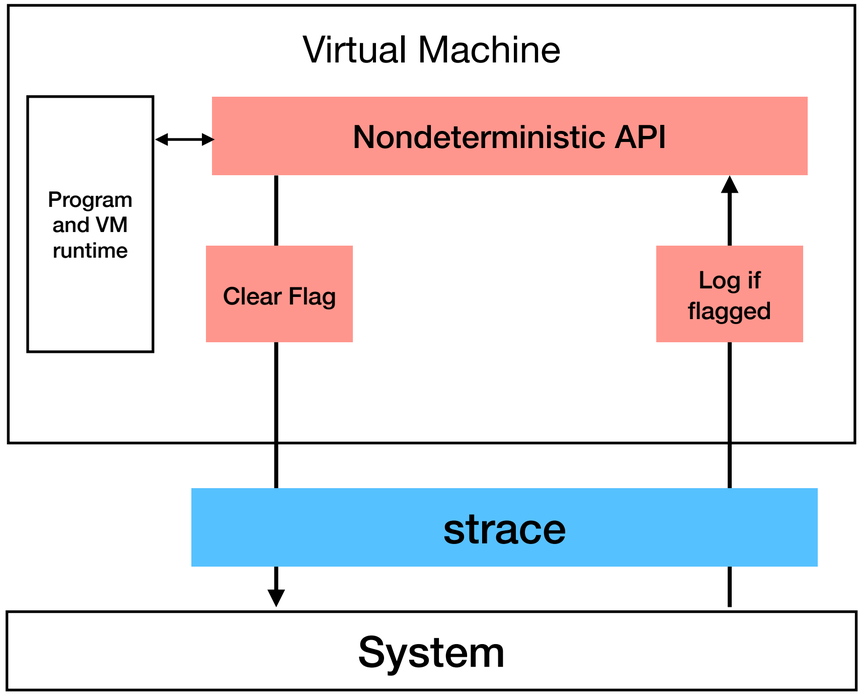


Instrumented bytecode to add hooks to natives

```
07: // Original code
08: nativeMethod();
09:
10: // Instrumented code
11: JMVX.beforeNative();
12: nativeMethod();
13: JMVX.afterNative();
```

```
17: static void beforeNative() {
18:   // Global Flag
19:   syscall = false;
20: }
21: static void afterNative() {
22:   if (syscall)
23:     logStackTrace();
24:   // else nop
25: }
```

Dynamic Tracing

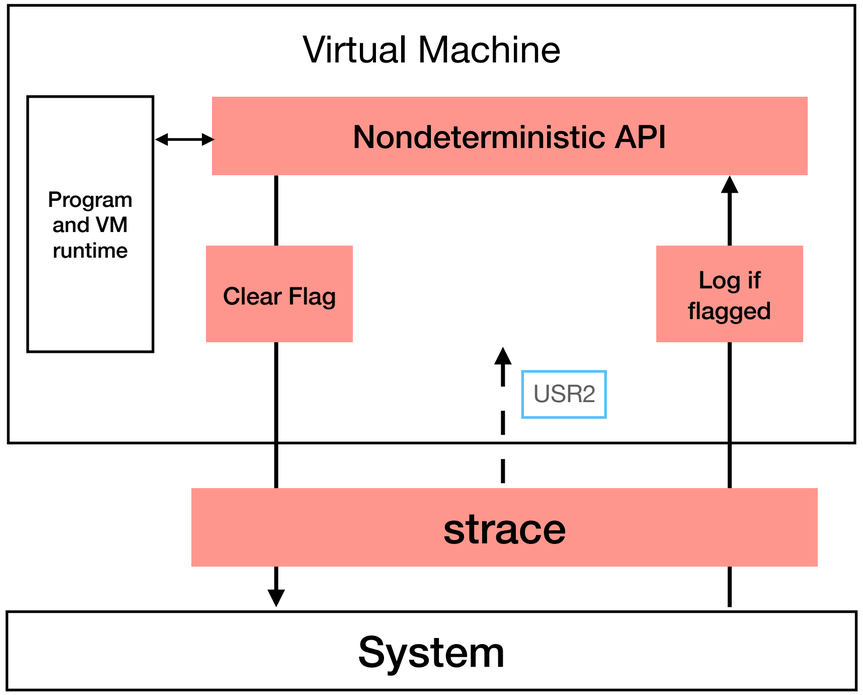


Instrumented bytecode to add hooks to natives

```
07: // Original code
08: nativeMethod();
09:
10: // Instrumented code
11: JMVX.beforeNative();
12: nativeMethod();
13: JMVX.afterNative();
```

```
17: static void beforeNative() {
18:  // Global Flag
19:  syscall = false;
20: }
21: static void afterNative() {
22:  if (syscall)
23:   logStackTrace();
24:  // else nop
25: }
```

Dynamic Tracing

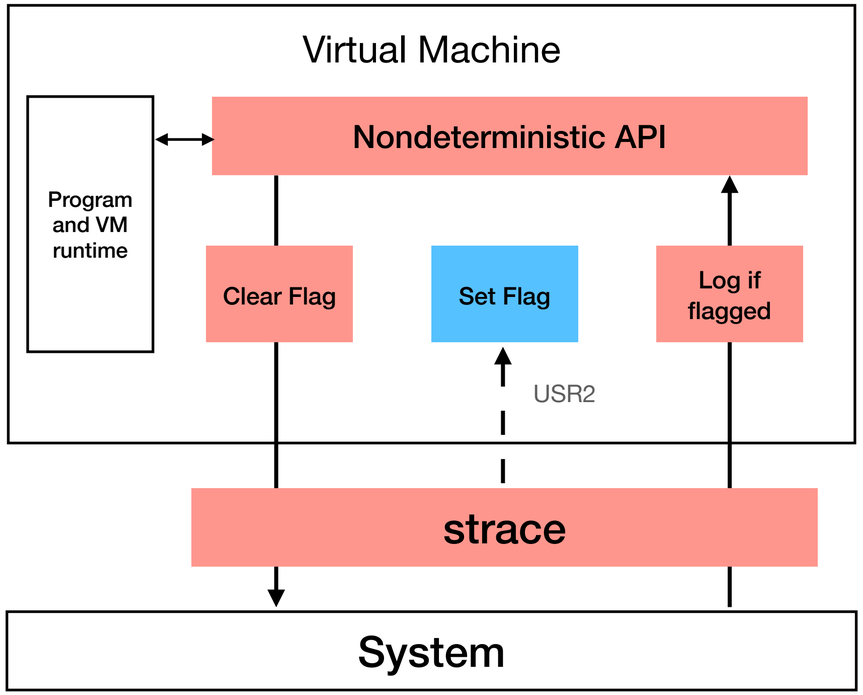


Instrumented bytecode to add hooks to natives

```
07: // Original code
08: nativeMethod();
09:
10: // Instrumented code
11: JMVX.beforeNative();
12: nativeMethod();
13: JMVX.afterNative();
```

```
17: static void beforeNative() {
18:  // Global Flag
19:  syscall = false;
20: }
21: static void afterNative() {
22:  if (syscall)
23:   logStackTrace();
24:  // else nop
25: }
```

Dynamic Tracing



Instrumented bytecode to add hooks to natives

```
07: // Original code
08: nativeMethod();
09:
10: // Instrumented code
11: JMVX.beforeNative();
12: nativeMethod();
13: JMVX.afterNative();
```

```
17: static void beforeNative() {
18:  // Global Flag
19:  syscall = false;
20: }
21: static void afterNative() {
22:  if (syscall)
23:   logStackTrace();
24:  // else nop
25: }
```

Signal handler sets flag

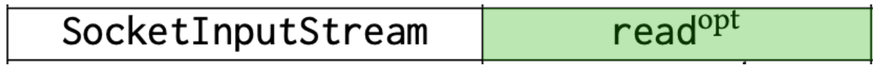
```
26: void signal(int sig) {
27:  if (sig == USR2)
28:   syscalls = true;
29: }
```

Dynamic Tracing Results

Package	Class	Name
java.util.zip	ZipFile	getEntryCSize getEntry getEntrySize getEntryBytes freeEntry getEntryMethod open getEntryFlag getEntryCrc getEntryTime close getTotal startsWithLOC getNextEntry
	ZipInputStream	read[opt]
	Adler32	updateBytes
	CRC32	updateBytes
	ZipOutputStream	write[opt]
java.util.jar	JarFile	getMetaInfEntry
sun.nio.ch	EPollArrayWrapper	epollCtl epollCreate sizeofEPollEvent
	FileDispatcherImpl	read write seek size close
java.util	TimeZone	getSystemTimeZone
java.lang	System	currentTimeMillis[opt] nanoTime[opt]
	ClassLoader	loadClass
	Object	wait[opt]
	Thread	run
java.util.concurrent	ThreadPoolExecutor	getTask
	QueueingFuture	done
	ConcurrentLinkedQueue	poll
synchronized[opt]		

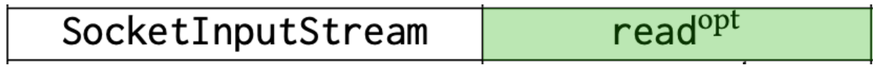
Package	Class	Name
java.io	RandomAccessFile	open close read[opt] write[opt] seek[opt]
	FileInputStream	open available[opt] read[opt] close
	FileOutputStream	open write[opt] close
	File	getCanonicalPath delete
java.nio	Files	createTempFile
sun.nio.fs.spi	FileSystemProvider	checkAccess copy
sun.nio.fs	UnixFileAttributes	get
	UnixNativeDispatcher	opendir fdopendir closedir readdir mkdir open dup stat lstat access realpath
java.net	ServerSocket	bind accept
	Socket	connect
	SocketInputStream	read[opt]
	SocketOutputStream	write[opt]

Bytecode Instrumentation




```
int read() { /* read a byte */ }
```

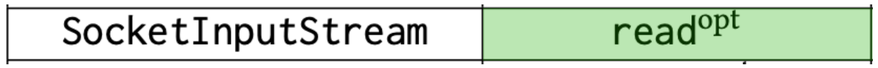
Bytecode Instrumentation




```
int $JMVX$read() { /* read a byte */ }
```

Rename the method

Bytecode Instrumentation

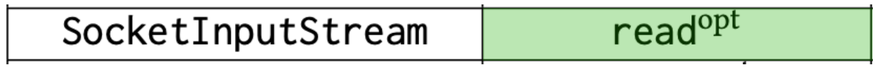



```
int $JMVX$read() { /* read a byte */ }
int read() { return JMVX.r.get().read(this); }
```

Rename the method

Add “new” method

Bytecode Instrumentation

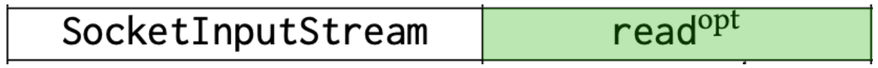



```
int $JMVX$read() { /* read a byte */ }
int read() { return JMVX.r.get().read(this); }
```

Rename the method

Add “new” method

Bytecode Instrumentation




```
int $JMVX$read() { /* read a byte */ }
int read() { return JMVX.r.get().read(this); }
```

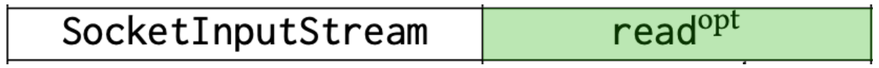
Rename the method

Add “new” method

Recorder Strategy

```
01:  Recorder.read(SocketInputStream s){
02:    int r = s.$JMVX$read();
03:    log.write(new SocketReadI(r));
04:    return r;
05:  }
```

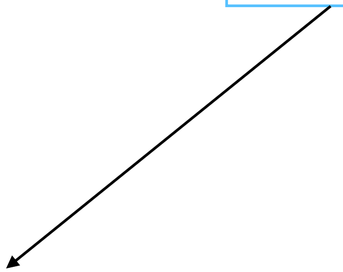
Bytecode Instrumentation




```
int $JMVX$read() { /* read a byte */ }
int read() { return JMVX.r.get().read(this); }
```

Rename the method

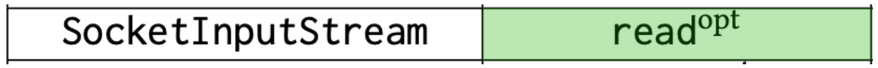
Add “new” method



Recorder Strategy

```
01:  Recorder.read(SocketInputStream s){
02:     int r = s.$JMVX$read();
03:     log.write(new SocketReadI(r));
04:     return r;
05:  }
```

Bytecode Instrumentation




```
int $JMVX$read() { /* read a byte */ }
int read() { return JMVX.r.get().read(this); }
```

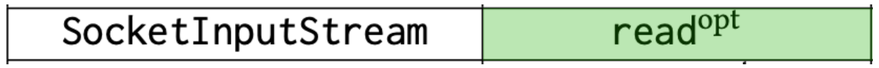
Rename the method

Add “new” method

Recorder Strategy

```
01:  Recorder.read(SocketInputStream s){
02:    int r = s.$JMVX$read();
03:    log.write(new SocketReadI(r));
04:    return r;
05:  }
```

Bytecode Instrumentation




```
int $JMVX$read() { /* read a byte */ }
int read() { return JMVX.r.get().read(this); }
```

Rename the method

Add “new” method

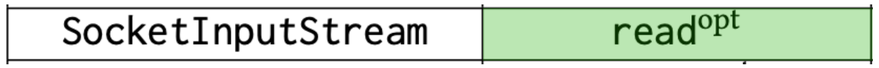
Recorder Strategy

```
01:  Recorder.read(SocketInputStream s){
02:    int r = s.$JMVX$read();
03:    log.write(new SocketReadI(r));
04:    return r;
05:  }
```

Replayer Strategy

```
07:  Replayer.read(SocketInputStream s){
08:    Object o = log.read();
09:    assert(o instanceof SocketReadI);
10:    return o.i;
11:  }
```

Bytecode Instrumentation




```
int $JMVX$read() { /* read a byte */ }
int read() { return JMVX.r.get().read(this); }
```

Rename the method

Add “new” method

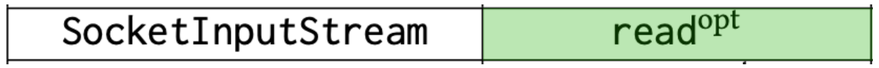
Recorder Strategy

```
01:  Recorder.read(SocketInputStream s){
02:    int r = s.$JMVX$read();
03:    log.write(new SocketReadI(r));
04:    return r;
05:  }
```

Replayer Strategy

```
07:  Replayer.read(SocketInputStream s){
08:    Object o = log.read();
09:    assert(o instanceof SocketReadI);
10:    return o.i;
11:  }
```

Bytecode Instrumentation




```
int $JMVX$read() { /* read a byte */ }
int read() { return JMVX.r.get().read(this); }
```

Rename the method

Add “new” method

Recorder Strategy

```
01:  Recorder.read(SocketInputStream s){
02:    int r = s.$JMVX$read();
03:    log.write(new SocketReadI(r));
04:    return r;
05:  }
```

Replayer Strategy

```
07:  Replayer.read(SocketInputStream s){
08:    Object o = log.read();
09:    assert(o instanceof SocketReadI);
10:    return o.i;
11:  }
```

Dynamic Tracing Results

Package	Class	Name
java.util.zip	ZipFile	getEntryCSize getEntry getEntrySize getEntryBytes freeEntry getEntryMethod open getEntryFlag getEntryCrc getEntryTime close getTotal startsWithLOC getNextEntry
	ZipInputStream	read^{opt}
	Adler32	updateBytes
	CRC32	updateBytes
	ZipOutputStream	write^{opt}
java.util.jar	JarFile	getMetaInfEntry
sun.nio.ch	EPollArrayWrapper	epollCtl epollCreate sizeofEPollEvent
	FileDispatcherImpl	read write seek size close
java.util	TimeZone	getSystemTimeZone
java.lang	System	currentTimeMillis^{opt} nanoTime^{opt}
	ClassLoader	loadClass
	Object	wait^{opt}
	Thread	run
java.util.concurrent	ThreadPoolExecutor	getTask
	QueueingFuture	done
	ConcurrentLinkedQueue	poll
synchronized^{opt}		

Package	Class	Name
java.io	RandomAccessFile	open close read^{opt} write^{opt} seek^{opt}
	FileInputStream	open available^{opt} read^{opt} close
	FileOutputStream	open write^{opt} close
	File	getCanonicalPath delete
java.nio	Files	createTempFile
sun.nio.fs.spi	FileSystemProvider	checkAccess copy
sun.nio.fs	UnixFileAttributes	get
	UnixNativeDispatcher	opendir fdopendir closedir readdir mkdir open dup stat lstat access realpath
java.net	ServerSocket	bind accept
	Socket	connect
	SocketInputStream	read^{opt}
	SocketOutputStream	write^{opt}

Dynamic Tracing Results

Package	Class	Name
java.util.zip	ZipFile	getEntryCSize getEntry getEntrySize getEntryBytes freeEntry getEntryMethod open getEntryFlag getEntryCrc getEntryTime close getTotal startsWithLOC getNextEntry
	ZipInputStream	read[opt]
	Adler32	updateBytes
	CRC32	updateBytes
	ZipOutputStream	write[opt]
java.util.jar	JarFile	getMetaInfEntry
sun.nio.ch	EPollArrayWrapper	epollCtl epollCreate sizeofEPollEvent
	FileDispatcherImpl	read write seek size close
java.util	TimeZone	getSystemTimeZone
	System	currentTimeMillis[opt] nanoTime[opt]
java.lang	ClassLoader	loadClass
	Object	wait[opt]
	Thread	run
java.util.concurrent	ThreadPoolExecutor	getTask
	QueueingFuture	done
	ConcurrentLinkedQueue	poll
synchronized[opt]		

Package	Class	Name
java.io	RandomAccessFile	open close read[opt] write[opt] seek[opt]
	FileInputStream	open available[opt] read[opt] close
	FileOutputStream	open write[opt] close
	File	getCanonicalPath delete
java.nio	Files	createTempFile
sun.nio.fs.spi	FileSystemProvider	checkAccess copy
sun.nio.fs	UnixFileAttributes	get
	UnixNativeDispatcher	opendir fdopendir closedir readdir mkdir open dup stat lstat access realpath
java.net	ServerSocket	bind accept
	Socket	connect
	SocketInputStream	read[opt]
	SocketOutputStream	write[opt]

Dynamic Tracing Results

java.lang	ClassLoader	loadClass

Dynamic Tracing Results

java.lang	ClassLoader	loadClass

Order classes are loaded in is nondeterministic

Dynamic Tracing Results

java.lang	Object Thread	wait[opt] run
java.util.concurrent	ThreadPoolExecutor QueueingFuture ConcurrentLinkedQueue	getTask done poll
synchronized[opt]		

Dynamic Tracing Results

Required for multi-threaded program support.
A major reason why we perform better than rr.

java.lang	Object Thread	wait[opt] run
java.util.concurrent	ThreadPoolExecutor QueueingFuture ConcurrentLinkedQueue	getTask done poll
synchronized[opt]		

Dynamic Tracing Results

Required for multi-threaded program support.
A major reason why we perform better than rr.

java.lang	Object Thread	wait[opt] run
java.util.concurrent	ThreadPoolExecutor QueueingFuture ConcurrentLinkedQueue	getTask done poll
synchronized[opt]		

Manually identified

Dynamic Tracing Results

Required for multi-threaded program support.
A major reason why we perform better than rr.

java.lang	Object Thread	wait[opt] run
java.util.concurrent	ThreadPoolExecutor QueueingFuture ConcurrentLinkedQueue	getTask done poll
synchronized[opt]		

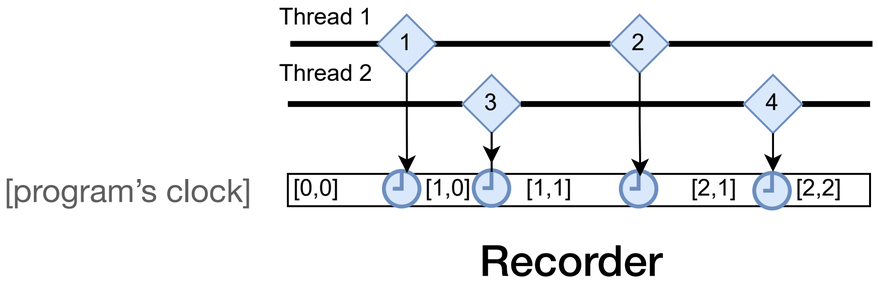
Manually identified

Automatically detected and instrumented

```
46: public synchronized void m() {
47:
48:     /* original body*/
49:
50: }
```

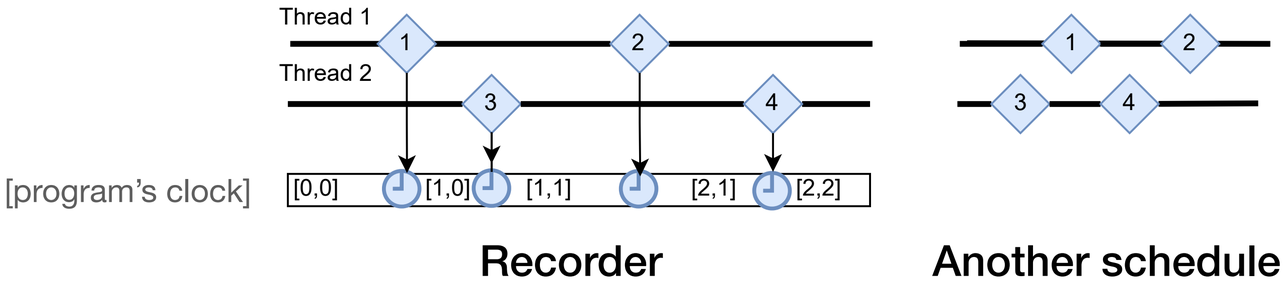
Multi-threading

- Lamport clock provides total order of monitor entry (locking an object)



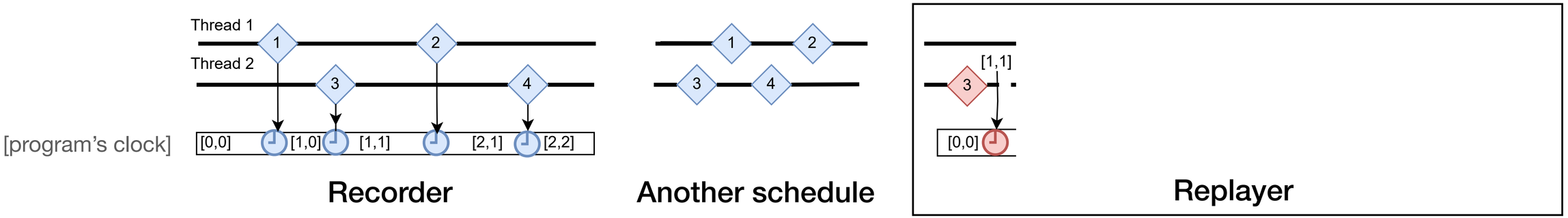
Multi-threading

- Lamport clock provides total order of monitor entry (locking an object)



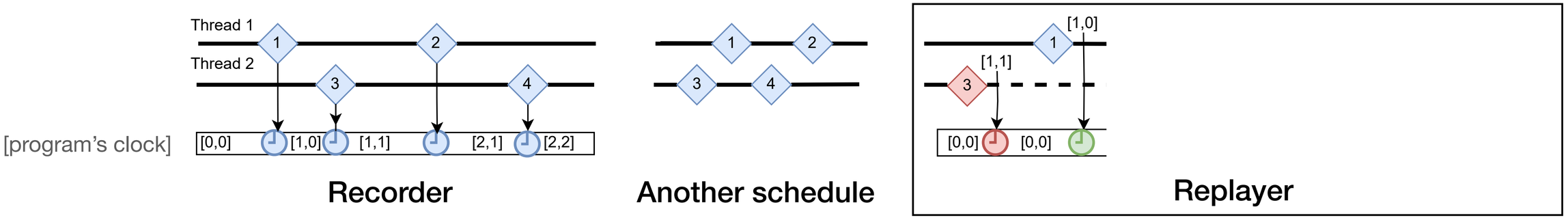
Multi-threading

- Lamport clock provides total order of monitor entry (locking an object)



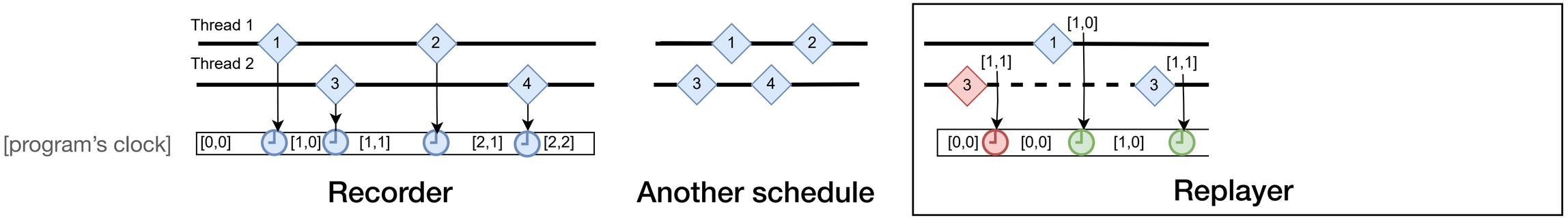
Multi-threading

- Lamport clock provides total order of monitor entry (locking an object)



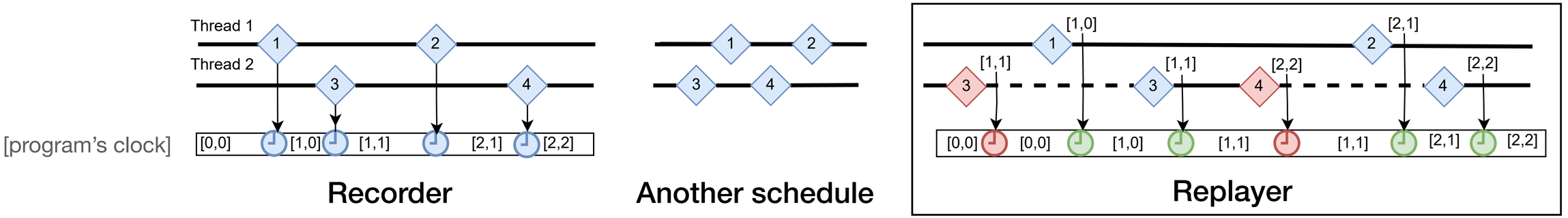
Multi-threading

- Lamport clock provides total order of monitor entry (locking an object)



Multi-threading

- Lamport clock provides total order of monitor entry (locking an object)



Multi-threading Instrumentation

```
46: public synchronized void m() {
47:
48:     /* original body*/
49:
50: }
```

Multi-threading Instrumentation

```
46: public synchronized void m() {
47:
48:     /* original body*/
49:
50: }
```

Multi-threading Instrumentation

```
46: public synchronized void $JMVX$m() {
47:
48:     /* original body*/
49:
50: }
```

Multi-threading Instrumentation

```
46: public synchronized void $JMVX$m() {
47:
48:     /* original body*/
49:
50: }
```

Add a new method

```
59: public void m() {
60:     JMVX.monitorEnter(this);
61:     try { $JMVX$m(); }
62:     finally { JMVX.monitorExit(this); }
63: }
```

Multi-threading Instrumentation

```
46: public synchronized void $JMVX$m() {
47:
48:     /* original body*/
49:
50: }
```

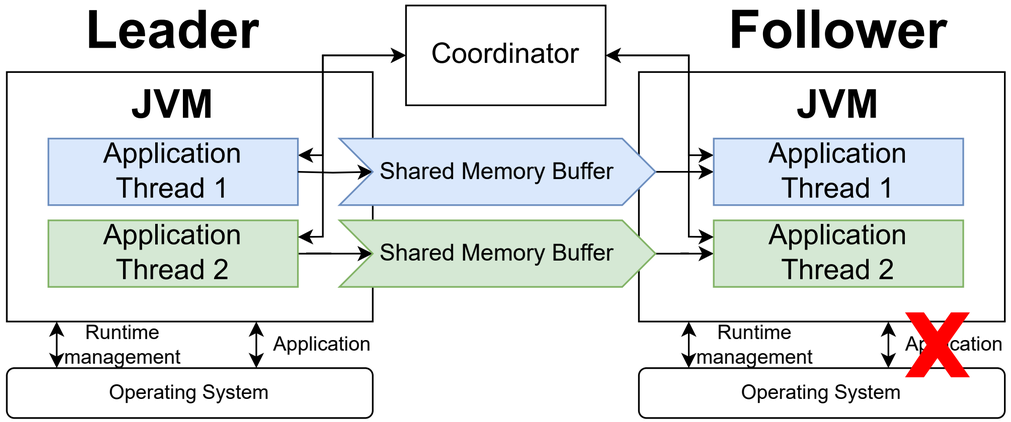
Add a new method

```
59: public void m() {
60:     JMVX.monitorEnter(this);
61:     try { $JMVX$m(); }
62:     finally { JMVX.monitorExit(this); }
63: }
```

Implements the *synchronized* logic through JMVX while logging or enforcing the ordering of the vector clock

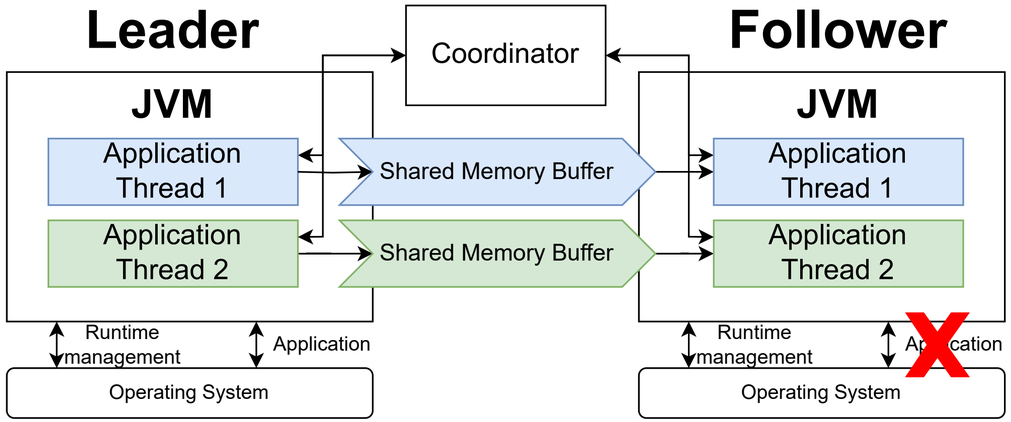
Ring Buffer

- Fast shared memory queue
- Shared off heap byte buffer
- Made with mmap
- Managed with Unsafe
  - Java class that allows direct memory access



Ring Buffer

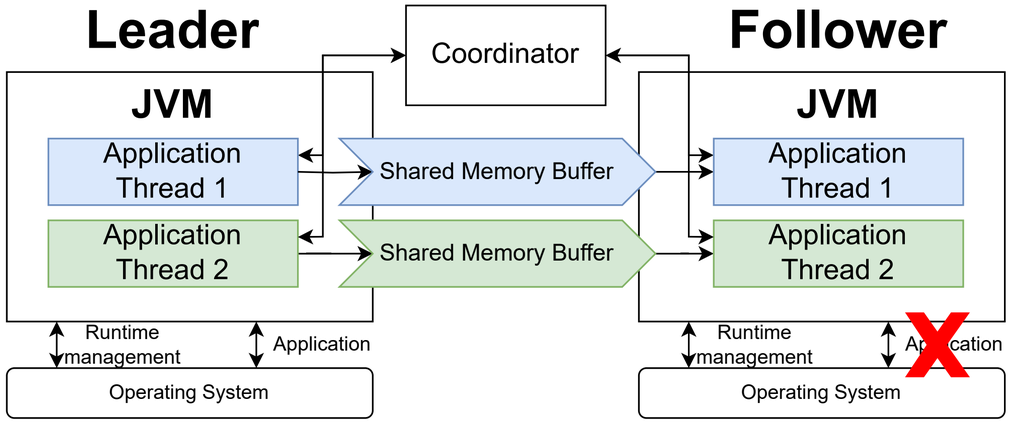
- Fast shared memory queue
- Shared off heap byte buffer
- Made with mmap
- Managed with Unsafe
  - Java class that allows direct memory access



No locks used in this design, so it's very fast!

Ring Buffer

- Fast shared memory queue
- Shared off heap byte buffer
- Made with mmap
- Managed with Unsafe
  - Java class that allows direct memory access



No locks used in this design, so it's very fast!

More details in the paper!

Divergence Handling

Leader Output

```
===== DaCapo unknown pmd starting =====
PMD checked 601 files.
===== DaCapo unknown pmd PASSED in 9518 msec =====
```

Follower Output

```
===== DaCapo unknown pmd starting =====
PMD checked 601 files.
===== DaCapo unknown pmd PASSED in 10558 msec =====
```

Divergence Handling

Leader Output

```
===== DaCapo unknown pmd starting =====
PMD checked 601 files.
===== DaCapo unknown pmd PASSED in 9518 msec =====
```

Follower Output

```
===== DaCapo unknown pmd starting =====
PMD checked 601 files.
===== DaCapo unknown pmd PASSED in 10558 msec =====
```

DIVERGENCE

Divergence Handling

Leader Output

```
===== DaCapo unknown pmd starting =====
PMD checked 601 files.
===== DaCapo unknown pmd PASSED in 9518 msec =====
```

Follower Output

```
===== DaCapo unknown pmd starting =====
PMD checked 601 files.
===== DaCapo unknown pmd PASSED in 10558 msec =====
```

DIVERGENCE

```
72: HandlerStatus<Integer> handleDivergence(List<StackTraceElement> trace, Metadata data)            {
73:   if (trace.size() < 18 || !(data.getLeader() instanceof WriteB))  return new HandlerStatus(FAIL);
74:   StackTraceElement frame = stack.get(17);
75:   if (!frame.getClassName().equals("org.dacapo.harness.Callback")) return new HandlerStatus(FAIL);
76:   if (!frame.getMethodName().equals("complete"))                   return new HandlerStatus(FAIL);
77:
78:   data.getFollower().bytes = new String(data.getFollower().bytes) + " and divergence handled ";
79:   return new HandlerStatus(OK, data.getFollower().length);                                        }
```

Divergence Handling

Leader Output

```
===== DaCapo unknown pmd starting =====
PMD checked 601 files.
===== DaCapo unknown pmd PASSED in 9518 msec =====
```

Follower Output

```
===== DaCapo unknown pmd starting =====
PMD checked 601 files.
===== DaCapo unknown pmd PASSED in 10558 msec and divergence handled =====
```

```
72: HandlerStatus<Integer> handleDivergence(List<StackTraceElement> trace, Metadata data)            {
73:   if (trace.size() < 18 || !(data.getLeader() instanceof WriteB))  return new HandlerStatus(FAIL);
74:   StackTraceElement frame = stack.get(17);
75:   if (!frame.getClassName().equals("org.dacapo.harness.Callback")) return new HandlerStatus(FAIL);
76:   if (!frame.getMethodName().equals("complete"))                   return new HandlerStatus(FAIL);
77:
78:   data.getFollower().bytes = new String(data.getFollower().bytes) + " and divergence handled ";
79:   return new HandlerStatus(OK, data.getFollower().length);                                        }
```

Evaluation: Research Questions

- What's the bytecode instrumentation overhead?
- What's the RR overhead?
  - Compare with rr and Chronicler
- What's the MVX overhead?
- How does JMVX scale?
  - With respect to number of threads and the size of the ring buffer
- (Mostly) Used programs from the DaCapo benchmark suite

Evaluation (Note): Sync Vs Nosync

- **Full/partial instrumentation respectively** (sync vs nosync)

Package	Class	Name
java.util.zip	ZipFile	getEntryCSize getEntry getEntrySize getEntryBytes freeEntry getEntryMethod open getEntryFlag getEntryCrc getEntryTime close getTotal startsWithLOC getNextEntry
	ZipInputStream	read^{opt}
	Adler32	updateBytes
	CRC32	updateBytes
	ZipOutputStream	write^{opt}
java.util.jar	JarFile	getMetaInfEntry
sun.nio.ch	EPollArrayWrapper	epollCtl epollCreate sizeofEPollEvent
	FileDispatcherImpl	read write seek size close
java.util	TimeZone	getSystemTimeZone
java.lang	System	currentTimeMillis^{opt} nanoTime^{opt}
	ClassLoader	loadClass
	Object	wait^{opt}
	Thread	run
java.util.concurrent	ThreadPoolExecutor	getTask
	QueueingFuture	done
	ConcurrentLinkedQueue	poll
synchronized^{opt}		

Package	Class	Name
java.io	RandomAccessFile	open close read^{opt} write^{opt} seek^{opt}
	FileInputStream	open available^{opt} read^{opt} close
	FileOutputStream	open write^{opt} close
	File	getCanonicalPath delete
java.nio	Files	createTempFile
sun.nio.fs.spi	FileSystemProvider	checkAccess copy
sun.nio.fs	UnixFileAttributes	get
	UnixNativeDispatcher	opendir fdopendir closedir readdir mkdir open dup stat lstat access realpath
java.net	ServerSocket	bind accept
	Socket	connect
	SocketInputStream	read^{opt}
	SocketOutputStream	write^{opt}

Evaluation (Note): Sync Vs Nosync

- **Full/partial instrumentation respectively** (sync vs nosync)

Package	Class	Name
java.util.zip	ZipFile	getEntryCSize getEntry getEntrySize getEntryBytes freeEntry getEntryMethod open getEntryFlag getEntryCrc getEntryTime close getTotal startsWithLOC getNextEntry
	ZipInputStream	read^{opt}
	Adler32	updateBytes
	CRC32	updateBytes
	ZipOutputStream	write^{opt}
java.util.jar	JarFile	getMetaInfEntry
sun.nio.ch	EPollArrayWrapper	epollCtl epollCreate sizeofEPollEvent
	FileDispatcherImpl	read write seek size close
java.util	TimeZone	getSystemTimeZone
java.lang	System	currentTimeMillis^{opt} nanoTime^{opt}
	ClassLoader	loadClass
	Object Thread	wait^{opt} run
java.util.concurrent	ThreadPoolExecutor QueueingFuture ConcurrentLinkedQueue	getTask done poll
synchronized^{opt}		

Package	Class	Name
java.io	RandomAccessFile	open close read^{opt} write^{opt} seek^{opt}
	FileInputStream	open available^{opt} read^{opt} close
	FileOutputStream	open write^{opt} close
	File	getCanonicalPath delete
java.nio	Files	createTempFile
sun.nio.fs.spi	FileSystemProvider	checkAccess copy
sun.nio.fs	UnixFileAttributes	get
	UnixNativeDispatcher	opendir fdopendir closedir readdir mkdir open dup stat lstat access realpath
java.net	ServerSocket	bind accept
	Socket	connect
	SocketInputStream	read^{opt}
	SocketOutputStream	write^{opt}

Nosync/partial instrumentation excludes classes that ordered multi-threaded events

Evaluation (Note): Sync Vs Nosync

- **Full/partial instrumentation respectively** (sync vs nosync)

Modes exist to have a fair comparison with other systems

Nosync/partial instrumentation excludes classes that ordered multi-threaded events

Package	Class	Name
java.util.zip	ZipFile	getEntryCSize getEntry getEntrySize getEntryBytes freeEntry getEntryMethod open getEntryFlag getEntryCrc getEntryTime close getTotal startsWithLOC getNextEntry
	ZipInputStream	read^{opt}
	Adler32	updateBytes
	CRC32	updateBytes
	ZipOutputStream	write^{opt}
java.util.jar	JarFile	getMetaInfEntry
sun.nio.ch	EPollArrayWrapper	epollCtl epollCreate sizeofEPollEvent
	FileDispatcherImpl	read write seek size close
java.util	TimeZone	getSystemTimeZone
java.lang	System	currentTimeMillis^{opt} nanoTime^{opt}
	ClassLoader	loadClass
	Object Thread	wait^{opt} run
java.util.concurrent	ThreadPoolExecutor QueueingFuture ConcurrentLinkedQueue	getTask done poll
synchronized^{opt}		

Package	Class	Name
java.io	RandomAccessFile	open close read^{opt} write^{opt} seek^{opt}
	FileInputStream	open available^{opt} read^{opt} close
	FileOutputStream	open write^{opt} close
	File	getCanonicalPath delete
java.nio	Files	createTempFile
sun.nio.fs.spi	FileSystemProvider	checkAccess copy
sun.nio.fs	UnixFileAttributes	get
	UnixNativeDispatcher	opendir fdopendir closedir readdir mkdir open dup stat lstat access realpath
java.net	ServerSocket	bind accept
	Socket	connect
	SocketInputStream	read^{opt}
	SocketOutputStream	write^{opt}

Evaluation: Instrumentation

- Pass through strategy used to measure overhead
- **Measures cost of diverting to JMVX**

```
int Passthrough.read(SocketInputStream s){
  //no logging
  return s.$JMVX$read();
}
```

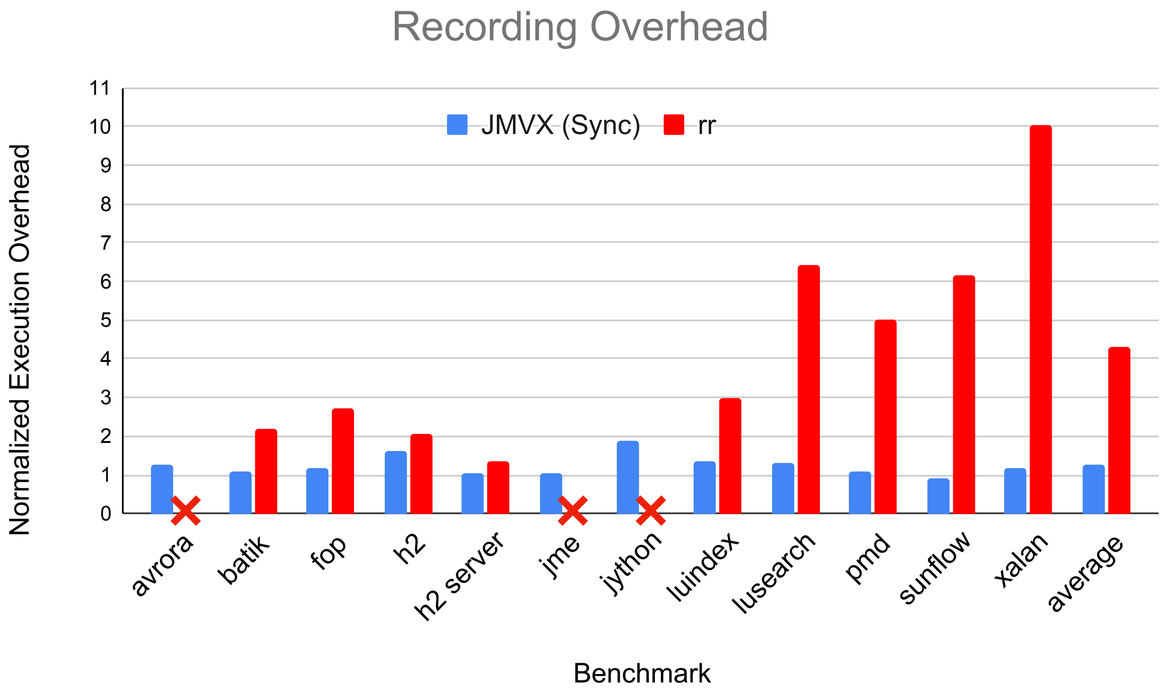
Evaluation: Instrumentation

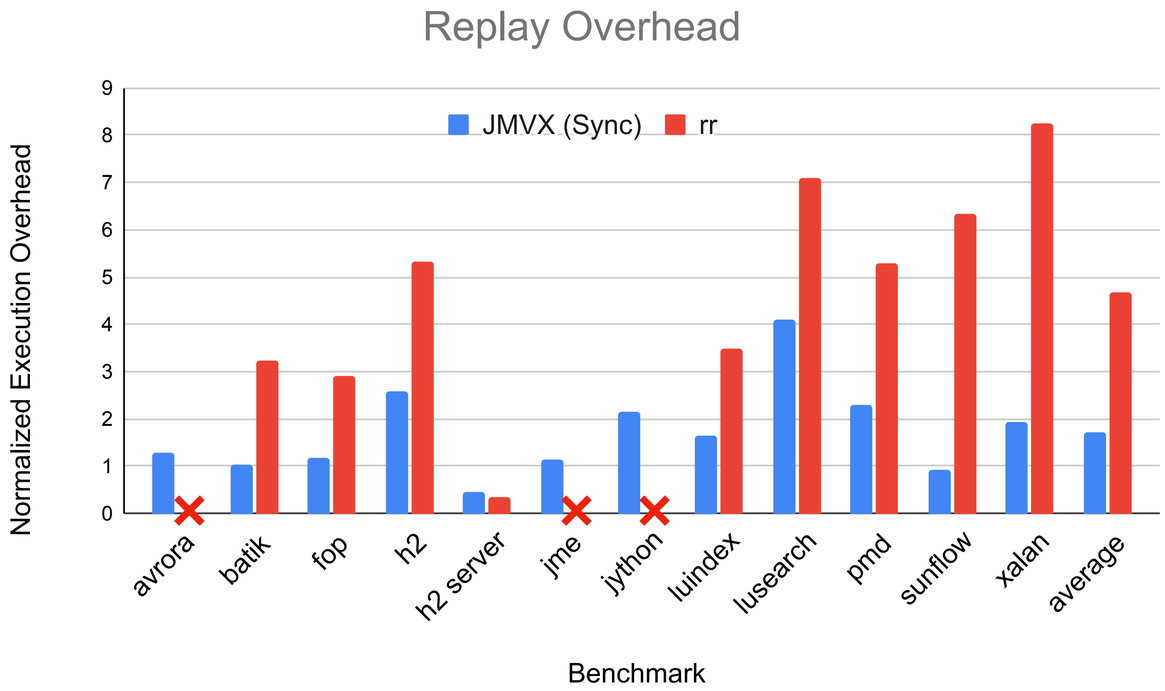
- Pass through strategy used to measure overhead
- **Measures cost of diverting to JMVX**
- Nosync instrumentation: **2% overhead**
- Sync instrumentation: **5% overhead**

```
int Passthrough.read(SocketInputStream s){
  //no logging
  return s.$JMVX$read();
}
```

Evaluation: RR (vs rr)

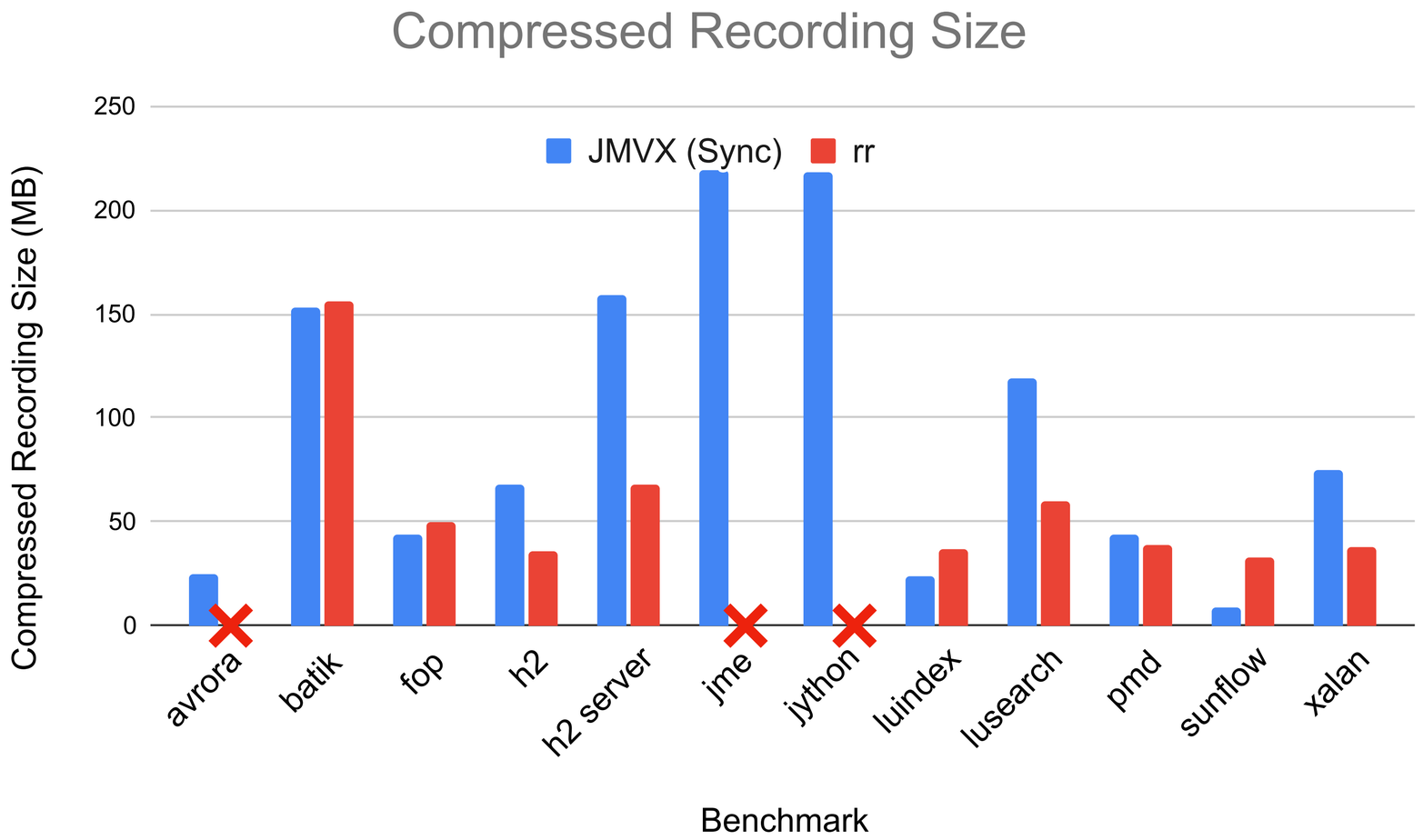
- Record: JMVX 1.25x overhead | rr 4.33x overhead
- Replay: JMVX 1.73x overhead | rr 4.70x overhead





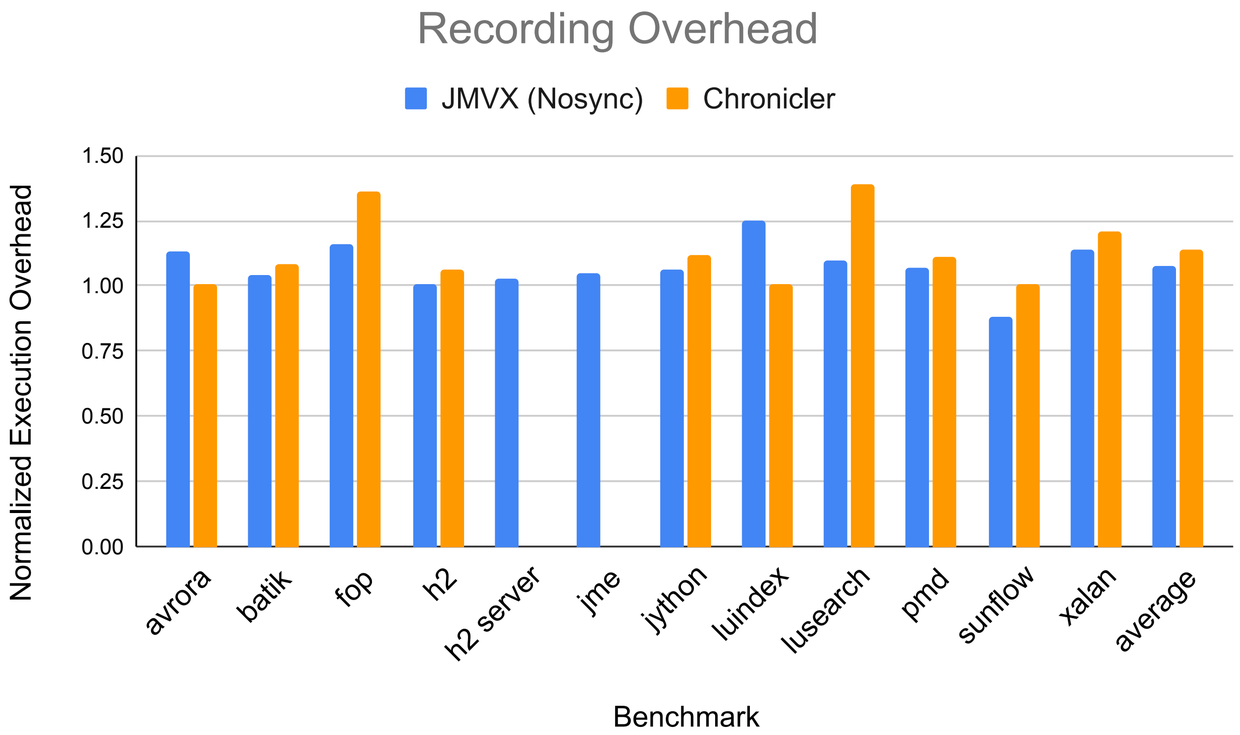
Data is normalized to the vanilla (uninstrumented) benchmark

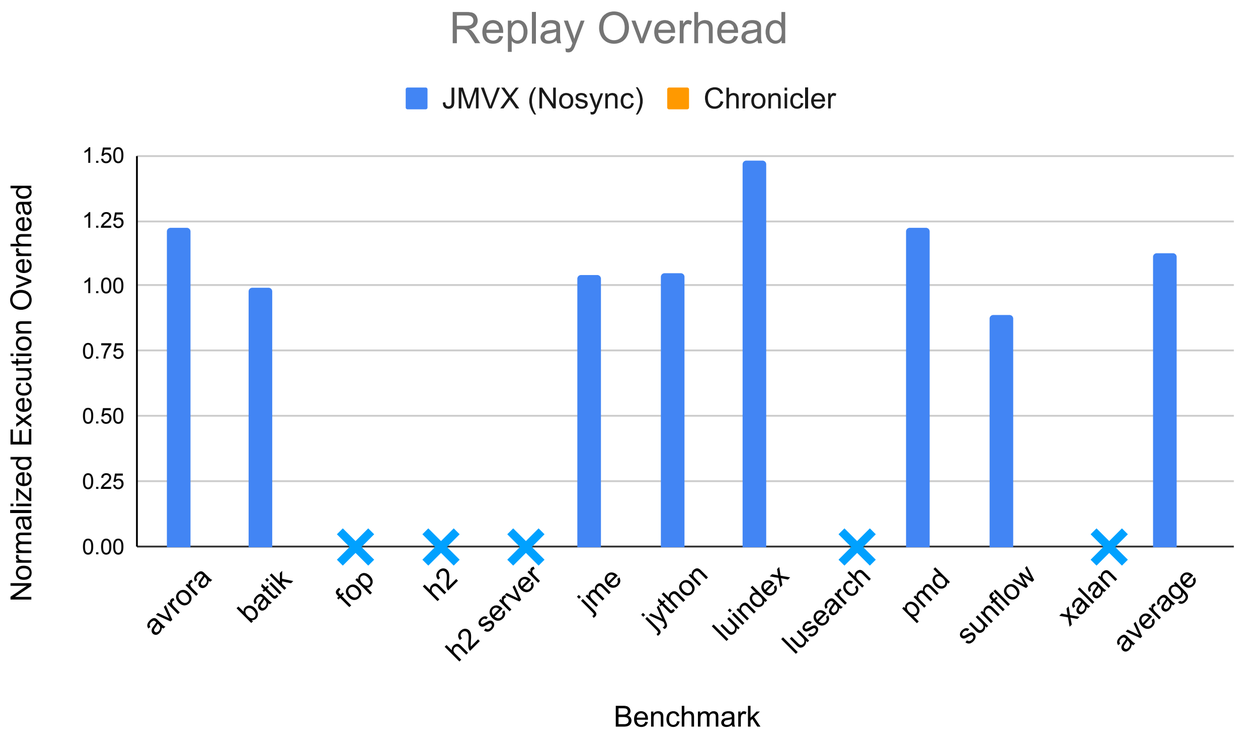
Evaluation: Recording Size



Evaluation: RR (vs Chronicler)

- Record: JMVX 1.08x overhead | Chronicler 1.14x overhead
- Replay: JMVX 1.13x overhead | Chronicler NA

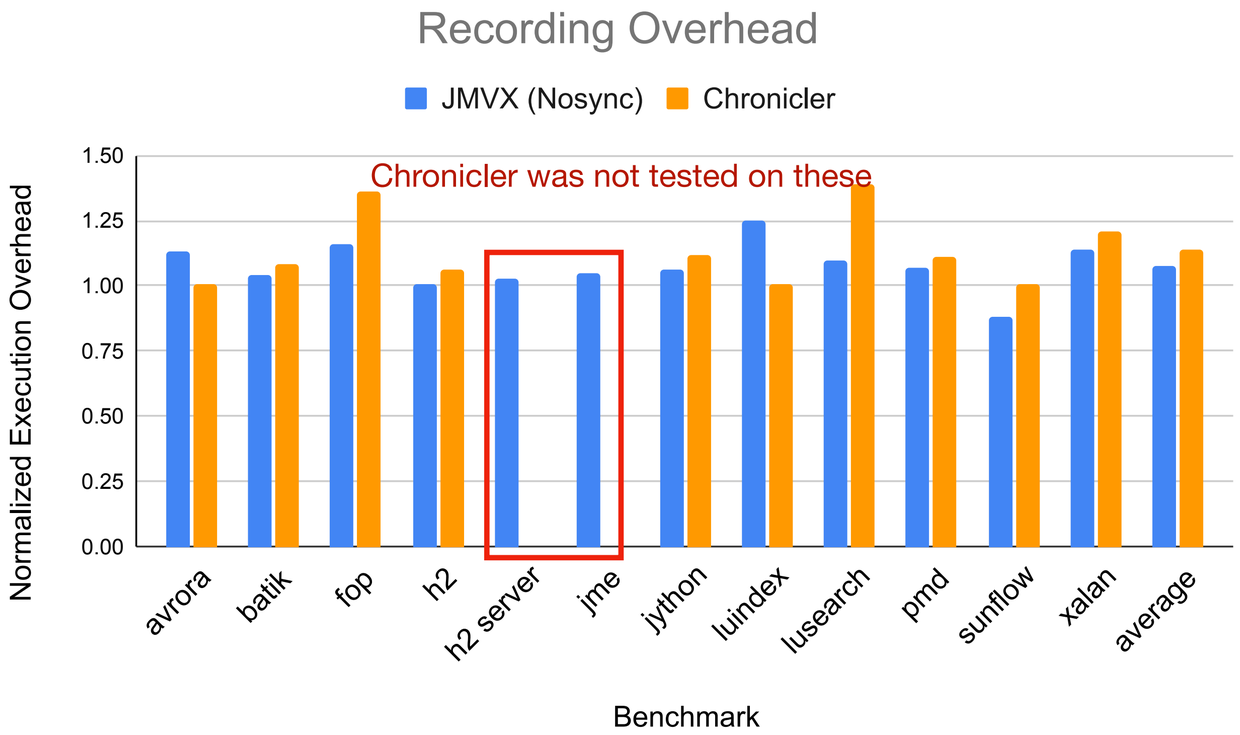


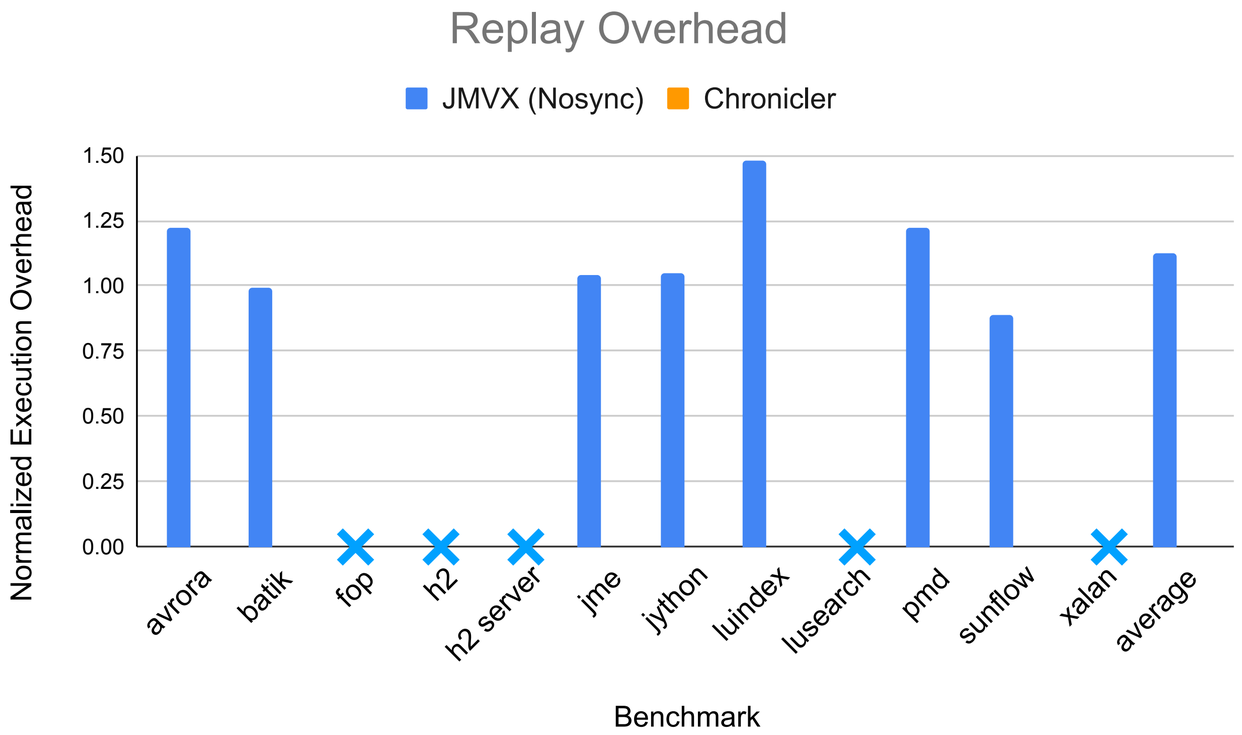


Data is normalized to the vanilla (uninstrumented) benchmark

Evaluation: RR (vs Chronicler)

- Record: JMVX 1.08x overhead | Chronicler 1.14x overhead
- Replay: JMVX 1.13x overhead | Chronicler NA



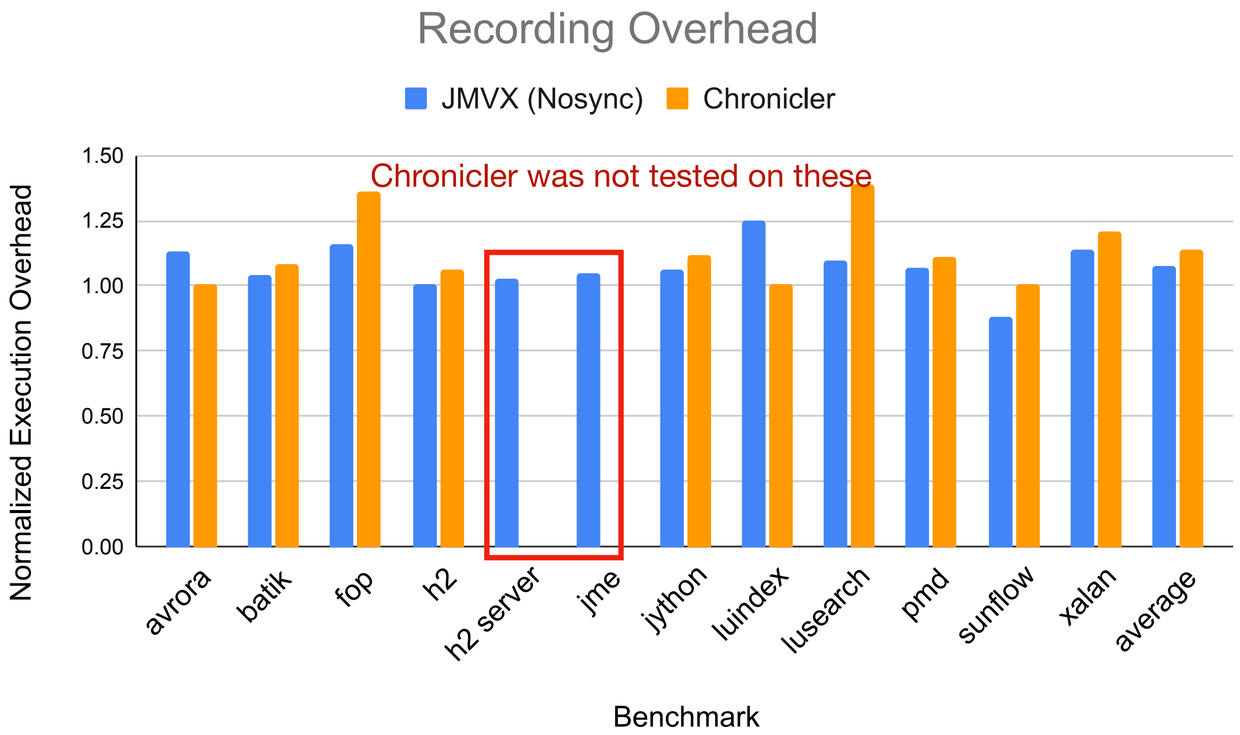


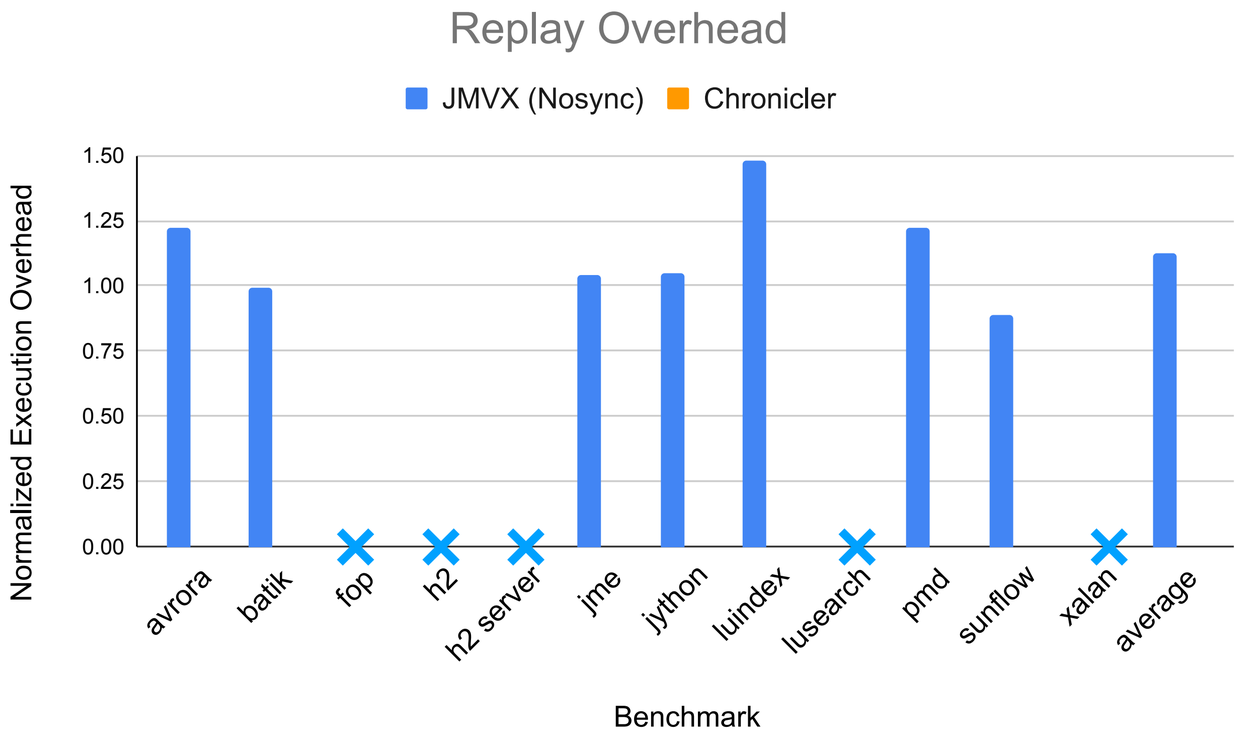
Data is normalized to the vanilla (uninstrumented) benchmark

Evaluation: RR (vs Chronicler)

- Record: JMVX 1.08x overhead | Chronicler 1.14x overhead
- Replay: JMVX 1.13x overhead | Chronicler NA

Chronicler does not provide replay runtime data in the paper

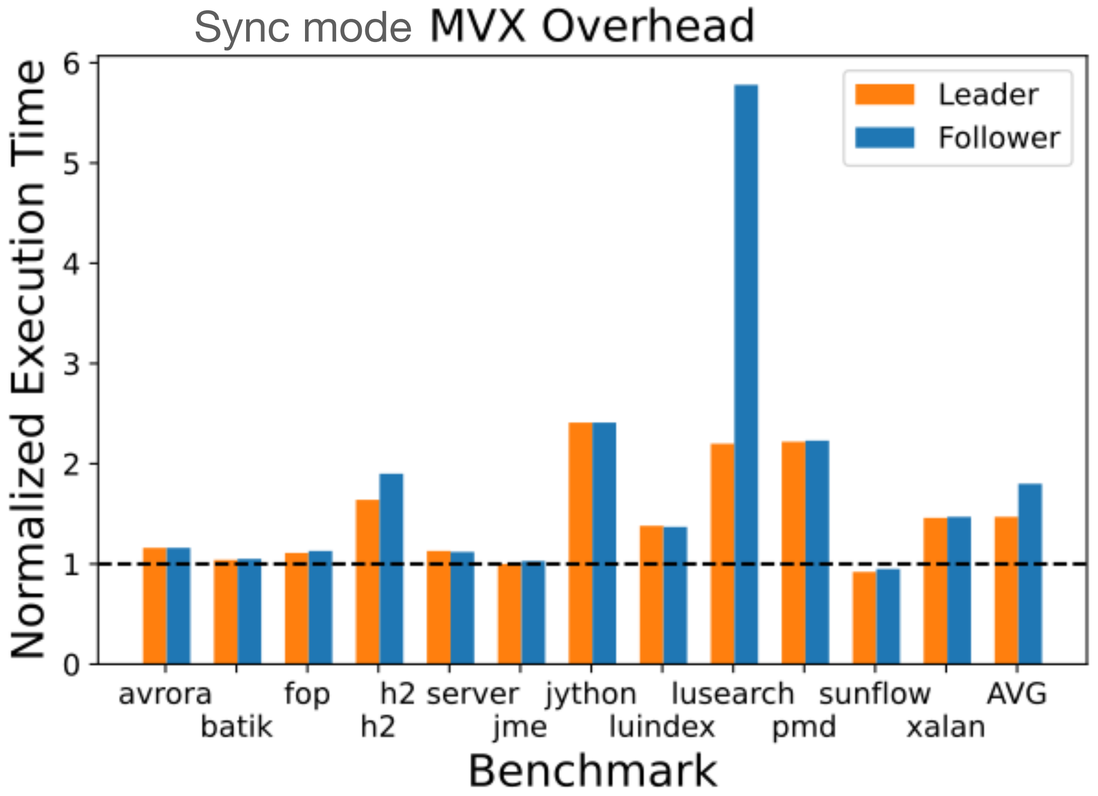




Data is normalized to the vanilla (uninstrumented) benchmark

Evaluation: MVX

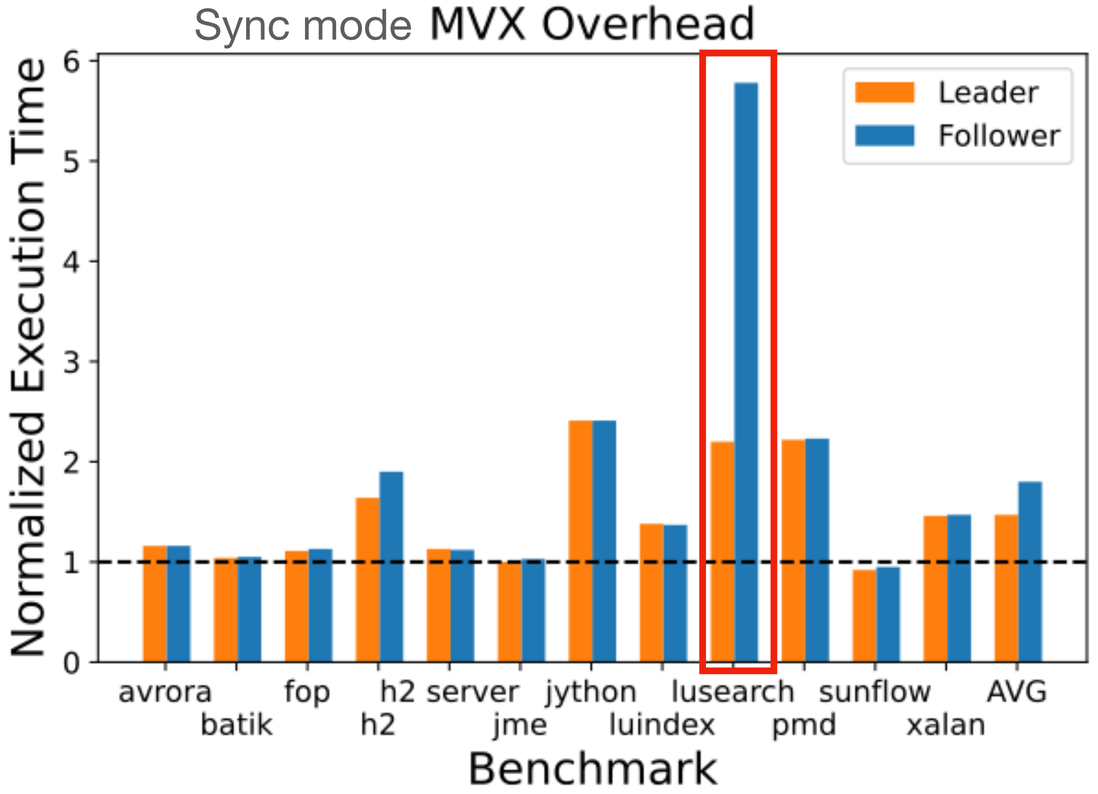
- Leader: 1.47x overhead
- Follower: 1.80x overhead



Data is normalized to the vanilla (uninstrumented) benchmark

Evaluation: MVX

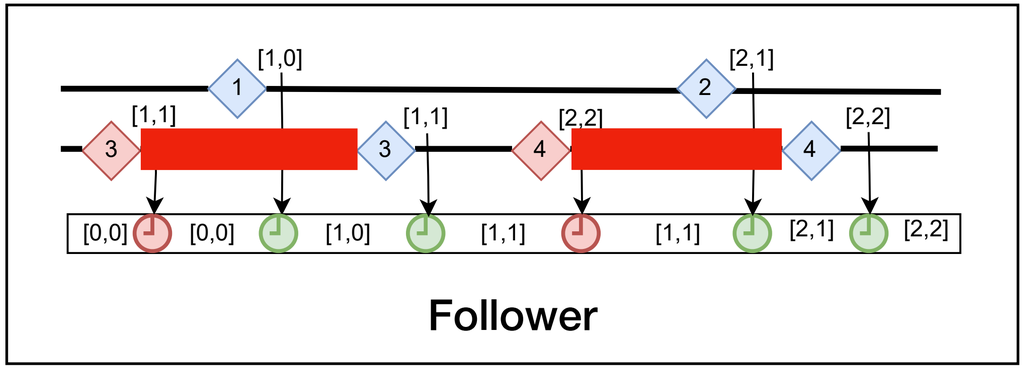
- Leader: 1.47x overhead
- Follower: 1.80x overhead

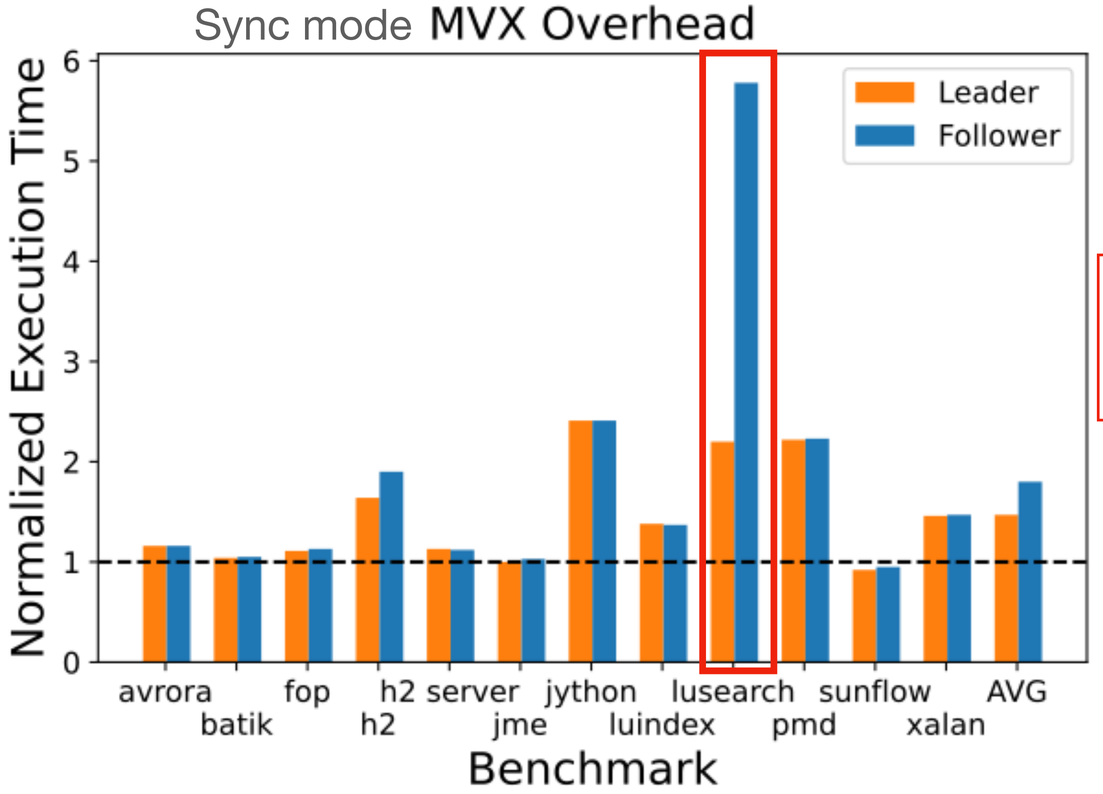


Data is normalized to the vanilla (uninstrumented) benchmark

Evaluation: MVX

- Leader: 1.47x overhead
- Follower: 1.80x overhead





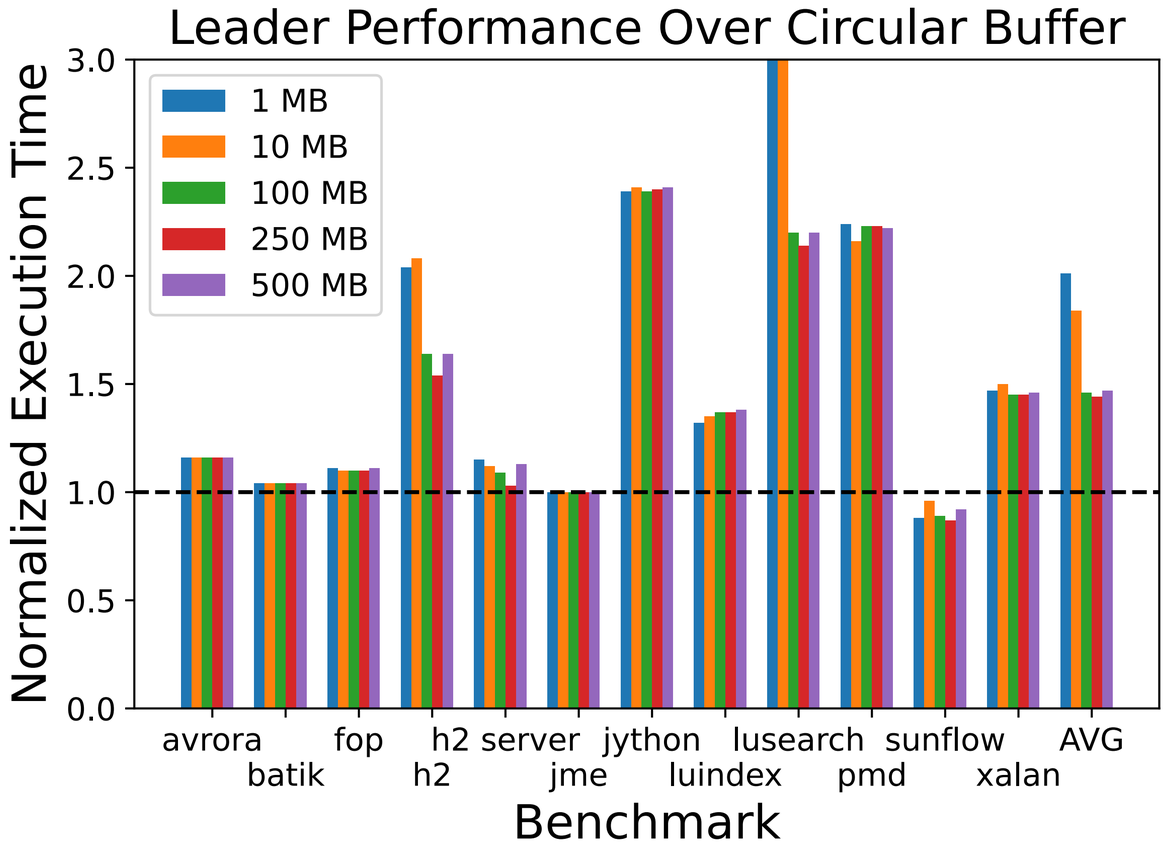
Many short critical sections.
Vector clock is biased for the Leader.
Delayed events add up.

Data is normalized to the vanilla (uninstrumented) benchmark

Evaluation: Circular Buffer Size

- Delay the user experiences is from the leader's execution

Each bar is a different size of the circular buffer

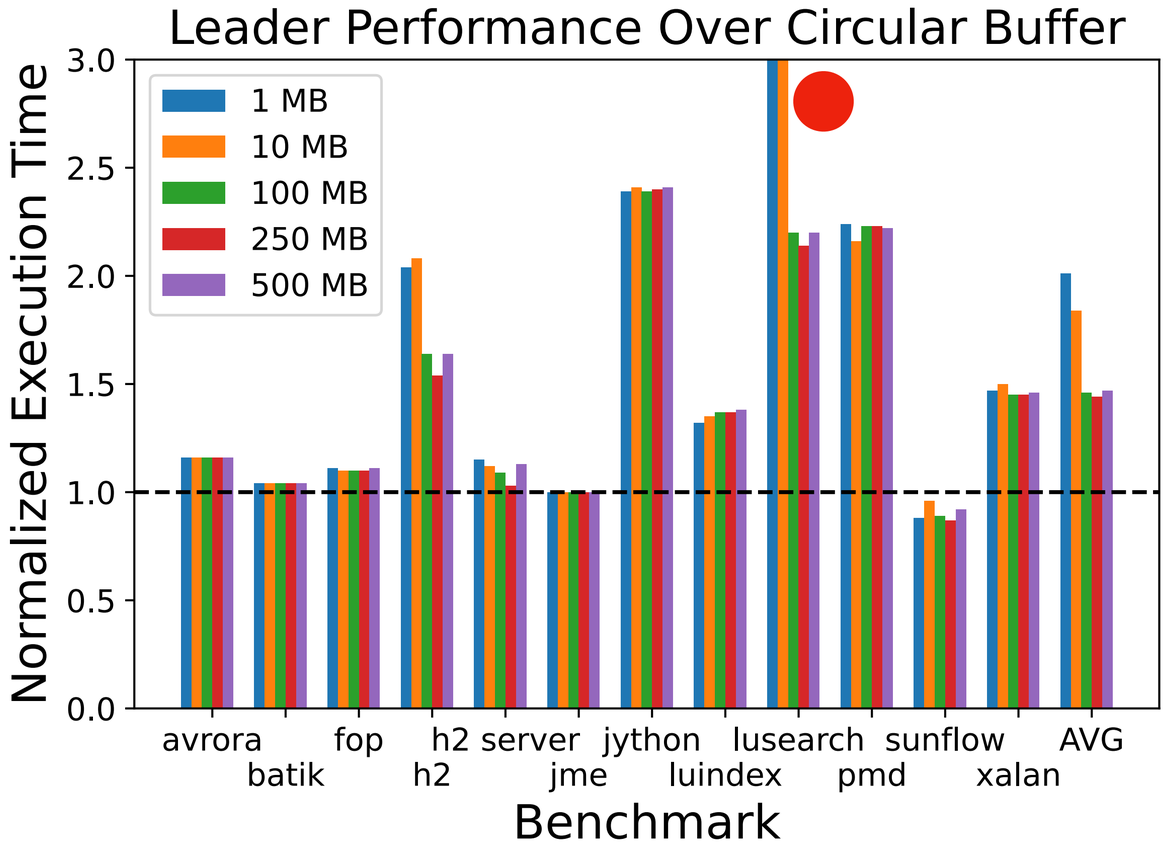


Data is normalized to the vanilla (uninstrumented) benchmark

Evaluation: Circular Buffer Size

- Delay the user experiences is from the leader's execution

Each bar is a different size of the circular buffer

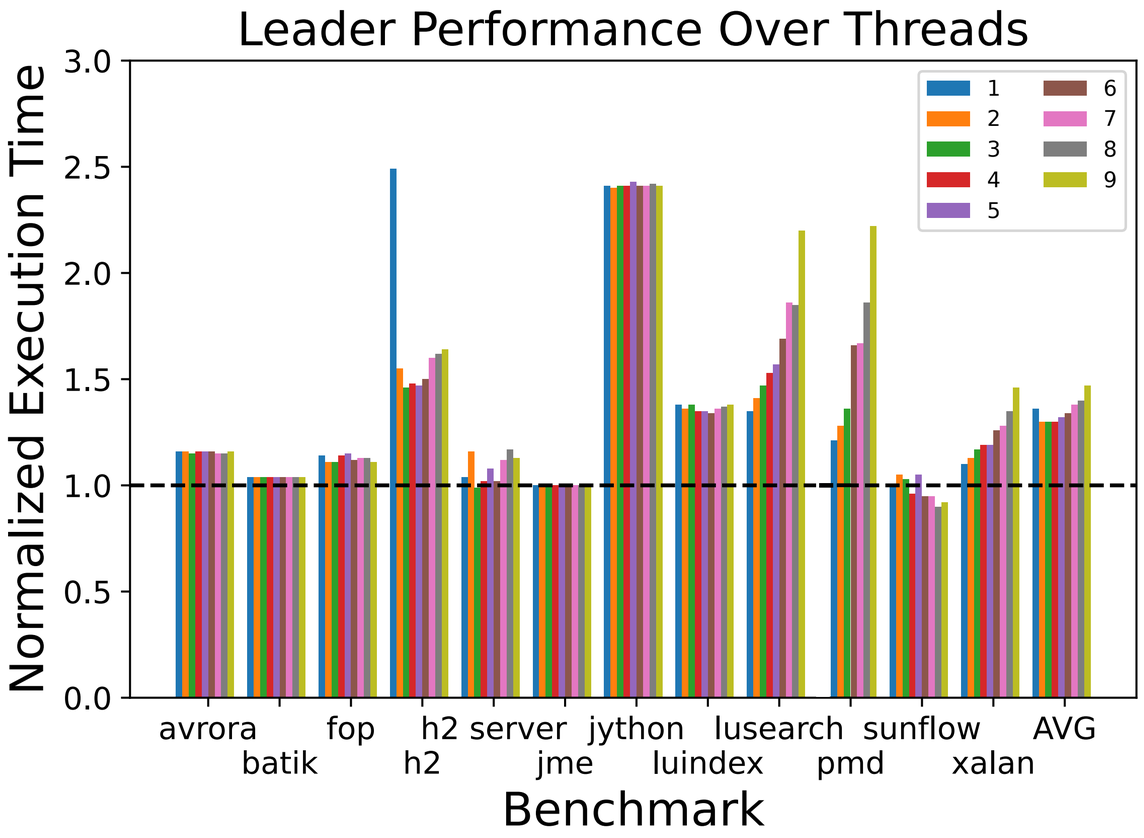


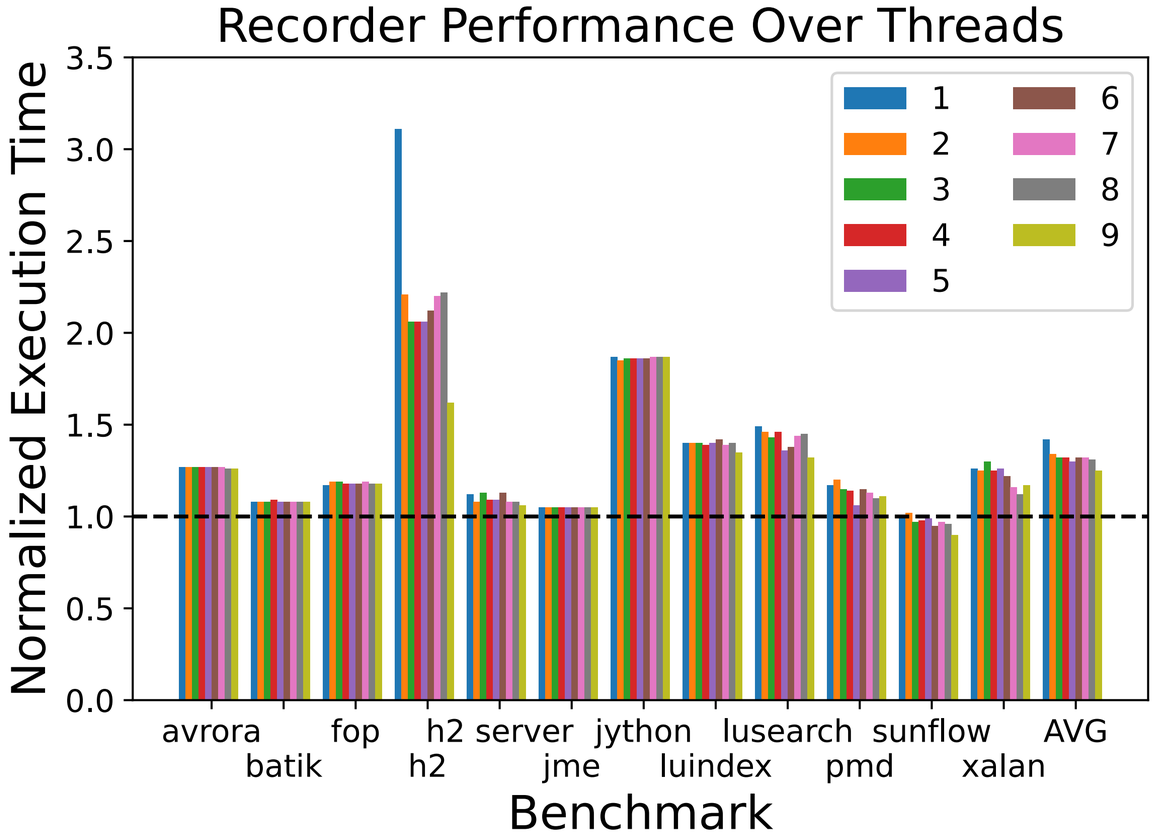
Follower does not process events in the ring buffer fast enough and slows down the leader

Data is normalized to the vanilla (uninstrumented) benchmark

Evaluation: Number of Threads

Bars are average results of running with a set thread count



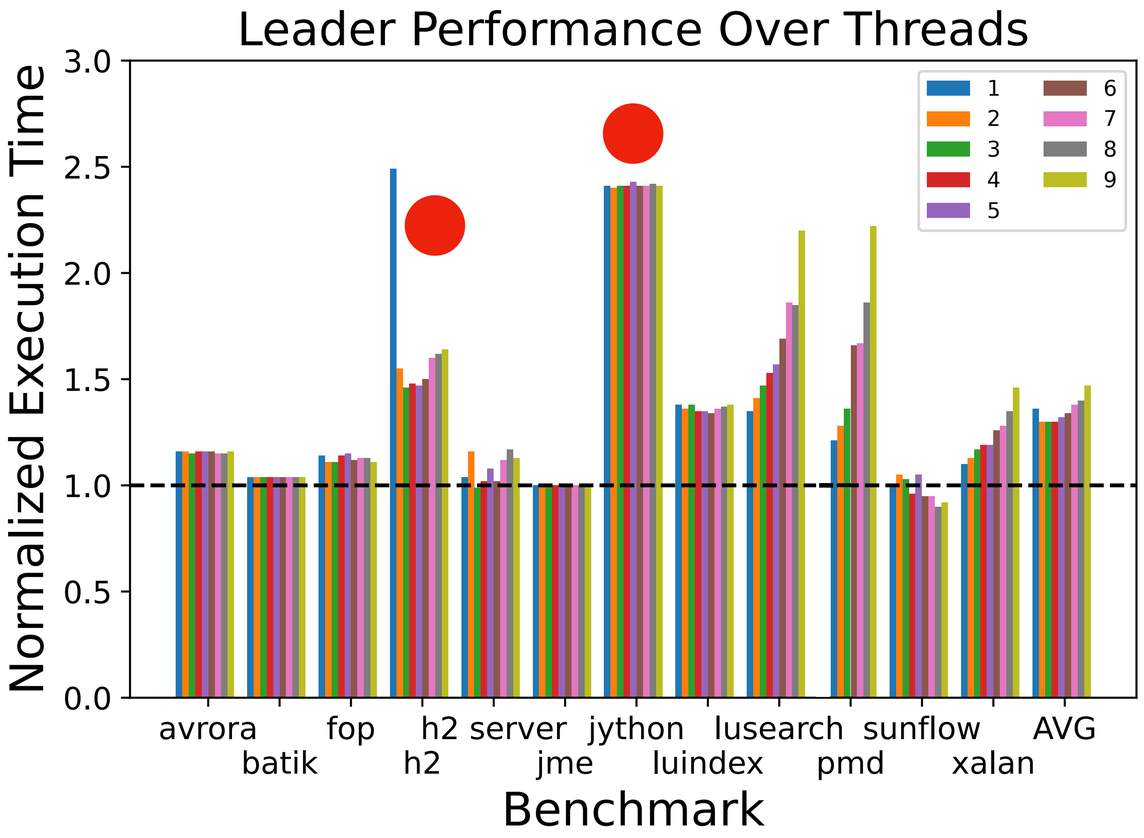


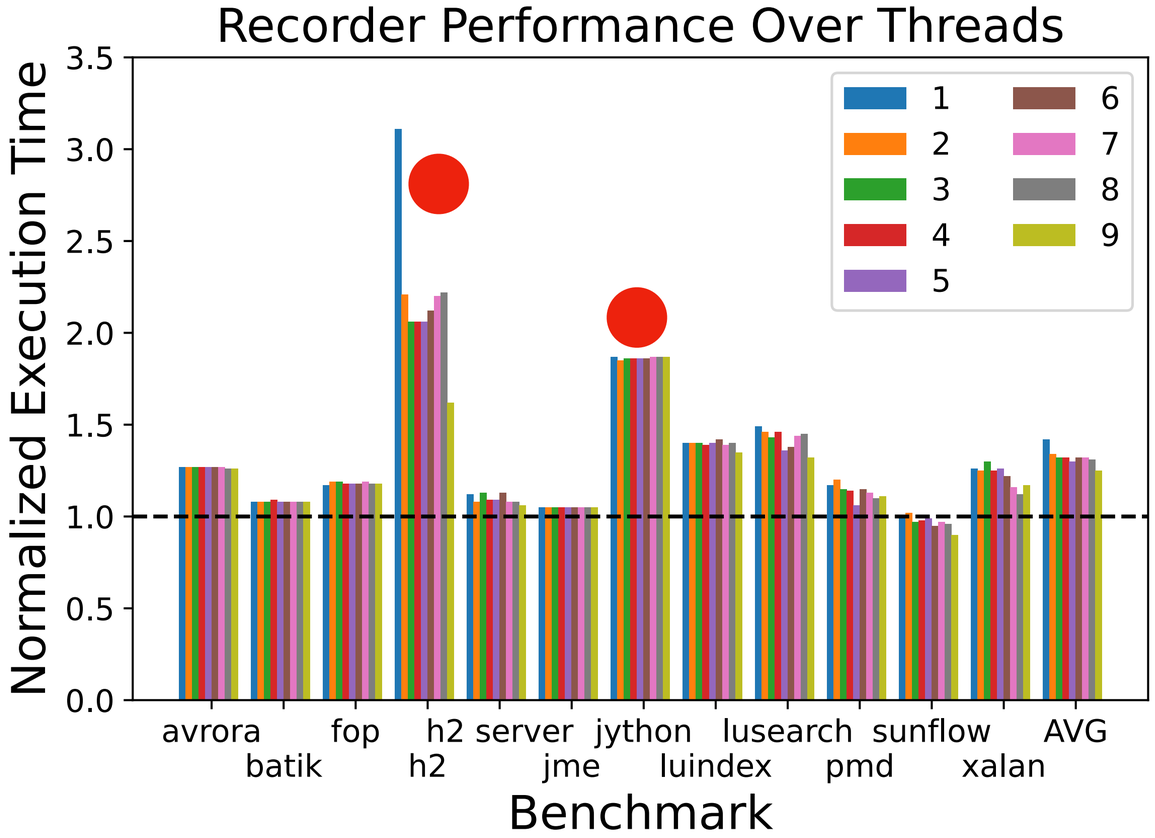
Data is normalized to the vanilla (uninstrumented) benchmark

Evaluation: Number of Threads

- Single threaded benchmarks which use optimized locks

Bars are average results of running with a set thread count



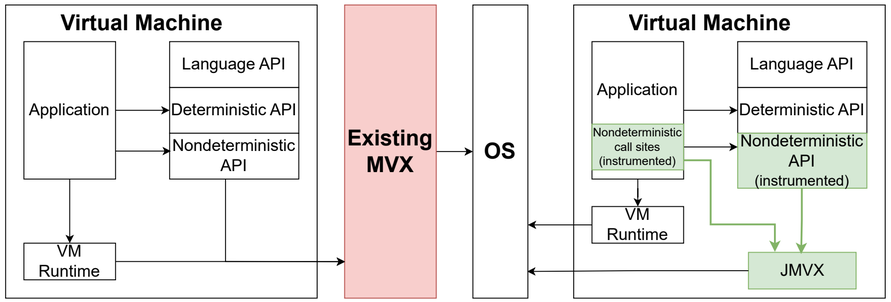


Data is normalized to the vanilla (uninstrumented) benchmark

Conclusion

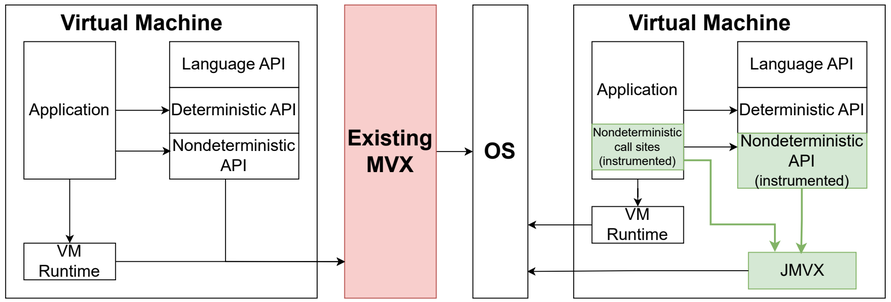
Conclusion

1. JMVX operates in bytecode rather than directly on system calls

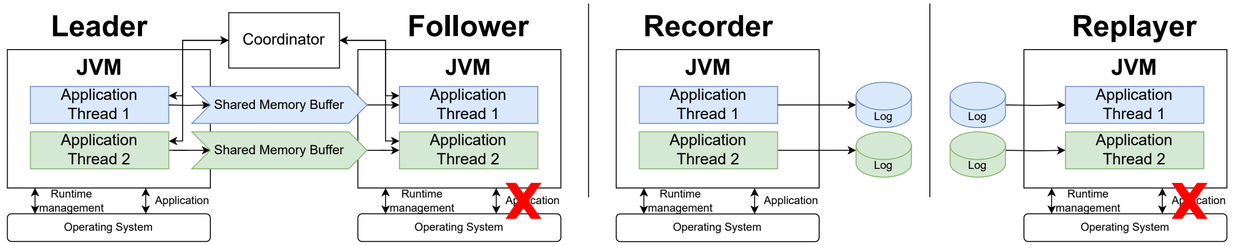


Conclusion

1 JMVX operates in bytecode rather than directly on system calls

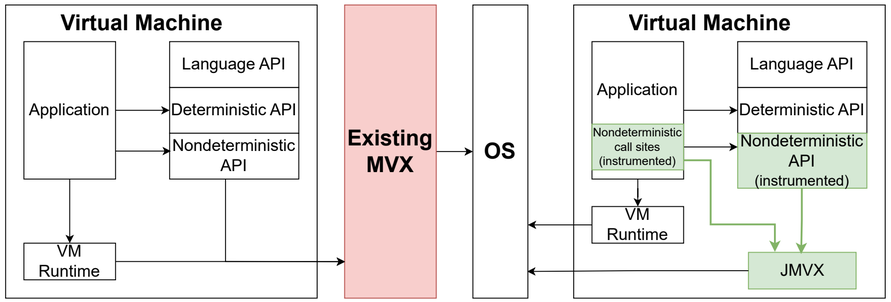


2 Supports both record replay and multi-version execution

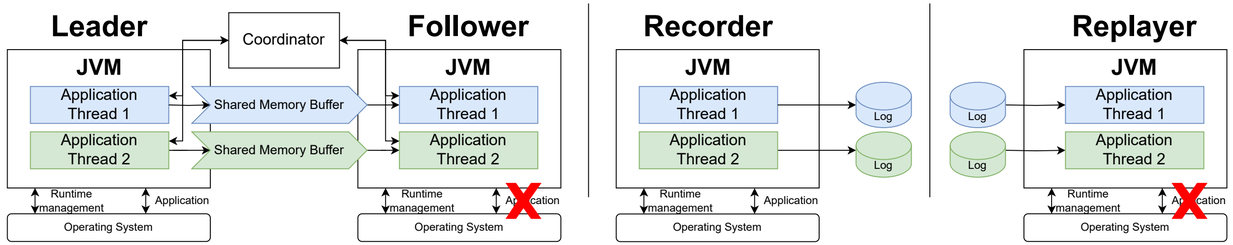


Conclusion

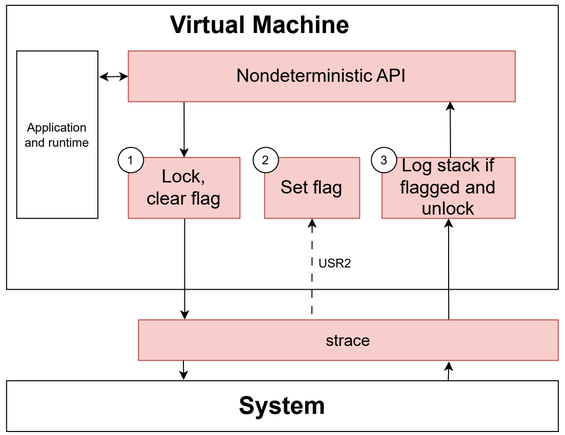
1 JMVX operates in bytecode rather than directly on system calls



2 Supports both record replay and multi-version execution



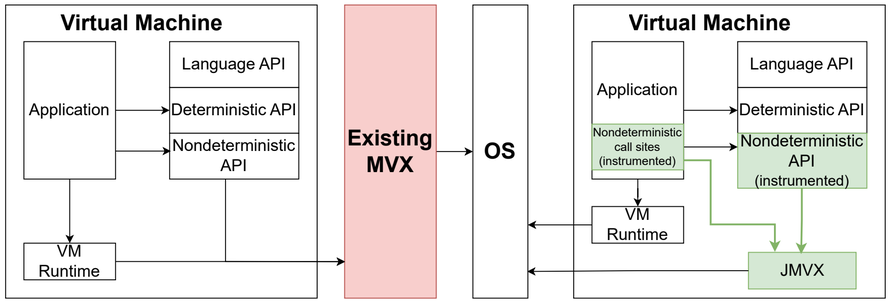
3 Identifies methods to instrument via dynamic tracing



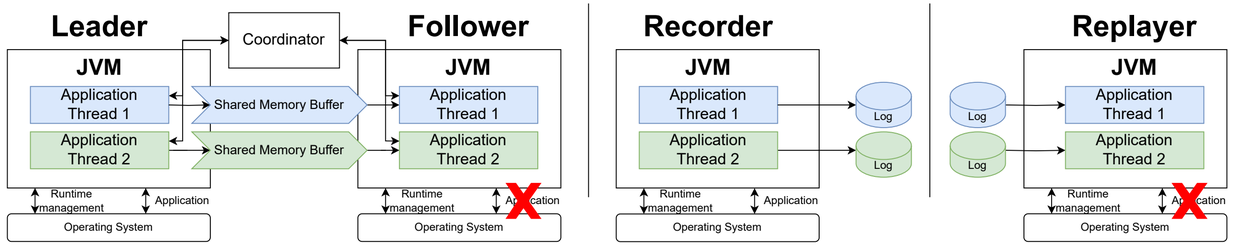
Authors: Schwartz, Kowshik, Pina

Conclusion

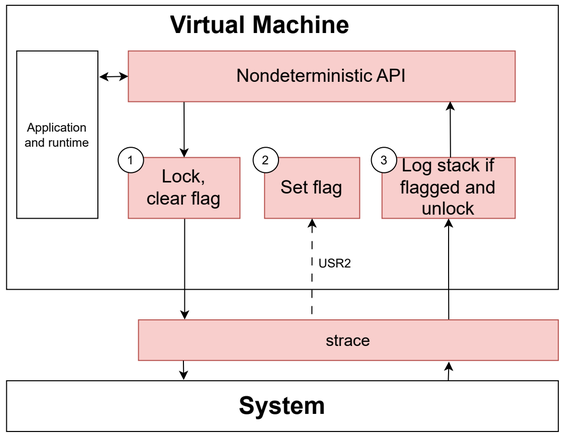
1 JMVX operates in bytecode rather than directly on system calls



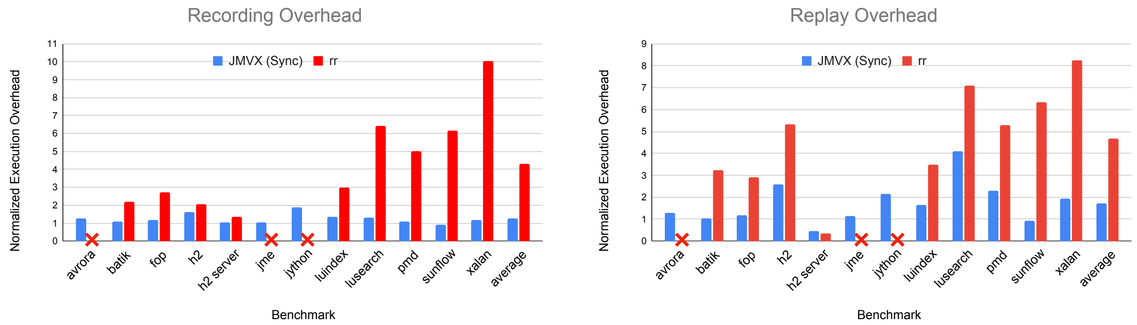
2 Supports both record replay and multi-version execution



3 Identifies methods to instrument via dynamic tracing



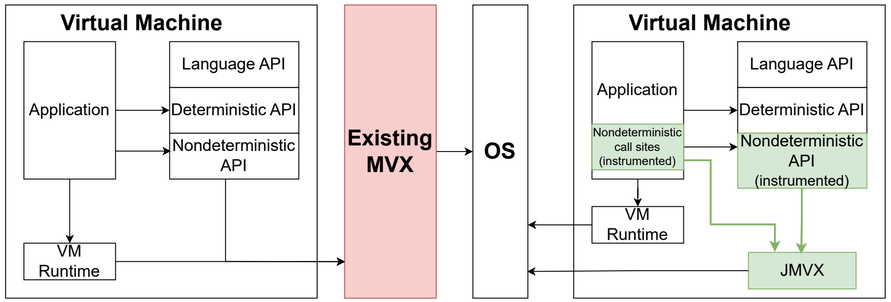
4 Outperforms rr, the most popular user space record/replay tool



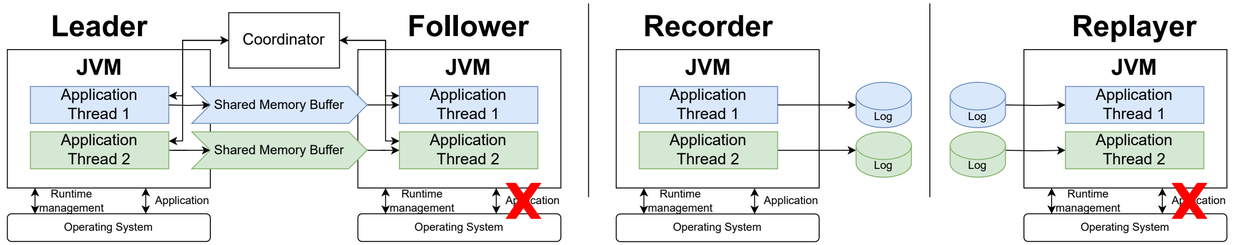
Authors: Schwartz, Kowshik, Pina

Conclusion

1 JMVX operates in bytecode rather than directly on system calls

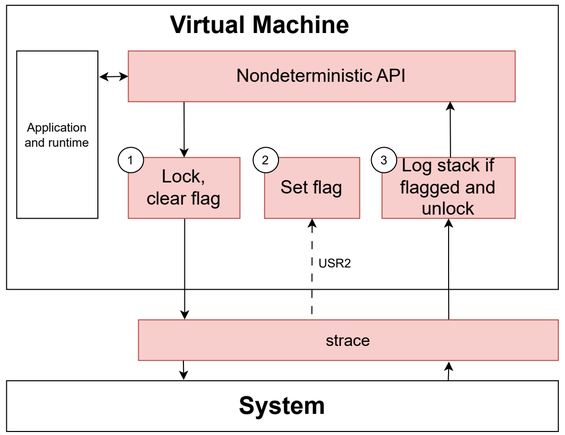


2 Supports both record replay and multi-version execution



Code available at: https://github.com/bitslab/jmvx

3 Identifies methods to instrument via dynamic tracing



4 Outperforms rr, the most popular user space record/replay tool