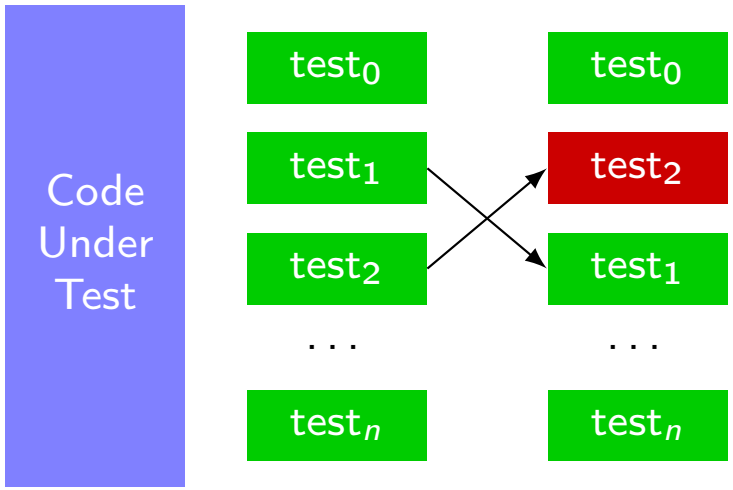
Debugging the Performance of Maven's Test Isolation: Experience Report

Pengyu Nie [1] Ahmet Celik [2] Matthew Coley [3]
Aleksandar Milicevic [4] Jonathan Bell [3] Milos Gligoric [1]

[1]The University of Texas at Austin [2]Facebook, Inc.
[3]George Mason University [4]Microsoft

ISSTA 2020

Need for Test Isolation



- Tests in industry are riddled with **flakiness**
  - tests may pass or fail nondeterministically without code changes
- A common practice to combat flaky tests is to run them in **isolation** from each other, to eliminate test-order dependencies

Test Isolation Introduces Substantial Overhead

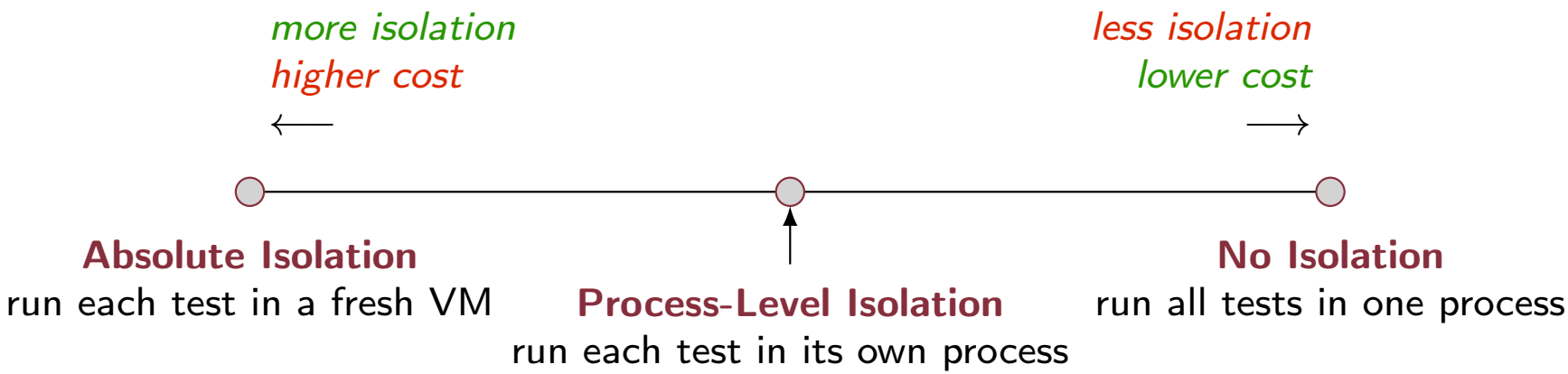
Absolute Isolation
run each test in a fresh VM

Test Isolation Introduces Substantial Overhead

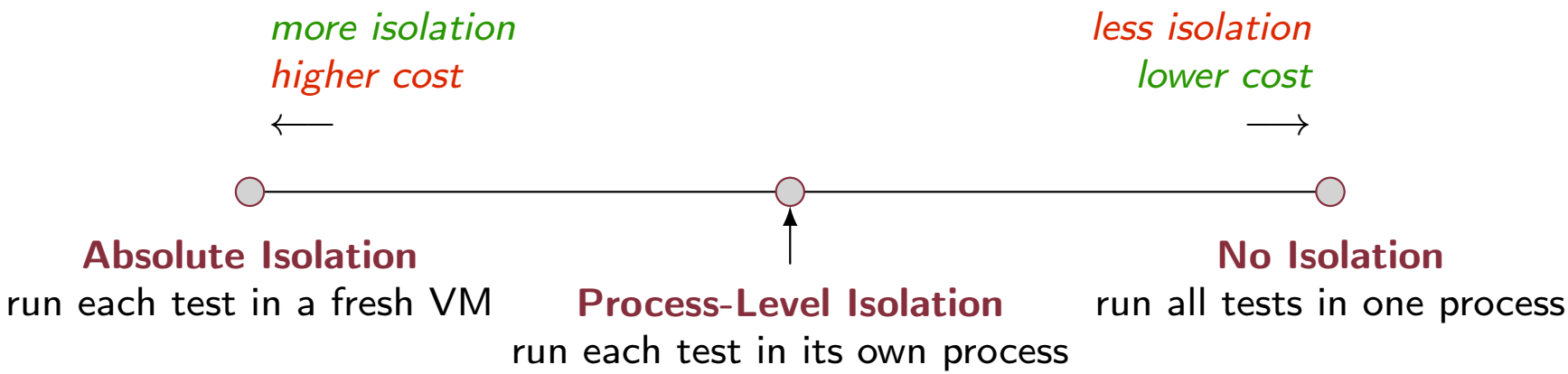
Absolute Isolation
run each test in a fresh VM

No Isolation
run all tests in one process

Test Isolation Introduces Substantial Overhead

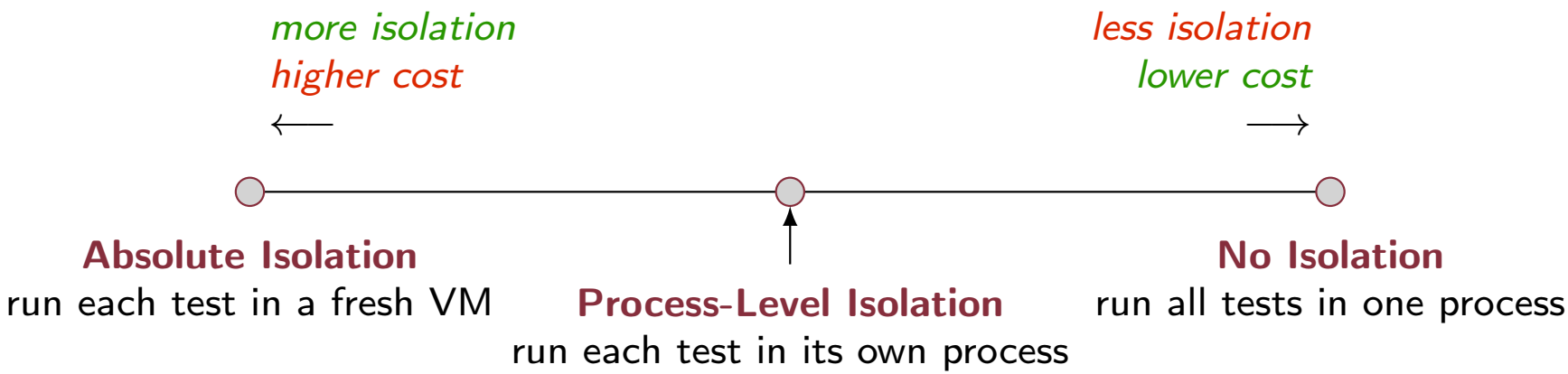


Test Isolation Introduces Substantial Overhead



- For Java: **forking** a separate JVM process for each test case

Test Isolation Introduces Substantial Overhead



- For Java: **forking** a separate JVM process for each test case
- Process-level test isolation still introduces **substantial overhead**
- Potential sources: startup cost, inter process communication

High Overhead of Test Isolation in Maven

- We performed an exploratory study to measure **per-test overhead introduced by the build systems**
- Execute test: `Thread.sleep(250)`
- Overhead = actual time − 250ms

Build System	Overhead (ms)
Ant 1.10.6	259
Gradle 5.6.1	412
Maven (Surefire 3.0.0-M3)	596

High Overhead of Test Isolation in Maven

- We performed an exploratory study to measure **per-test overhead introduced by the build systems**
- Execute test: `Thread.sleep(250)`
- Overhead = actual time − 250ms

Build System	Overhead (ms)
Ant 1.10.6	259
Gradle 5.6.1	412
Maven (Surefire 3.0.0-M3)	596

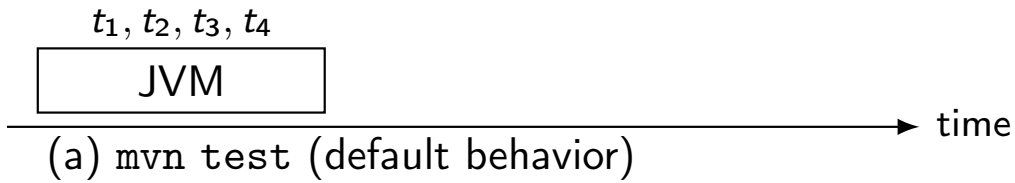
- Surprising findings:
  - Very different overhead among different build systems
  - **Maven** has huge overhead compared to others

Contributions

- **ForkScript**: a novel technique to minimize inter process communication overhead in test isolation, that saved test execution time by **50%**
- Guided by the development of ForkScript, we **found and fixed a performance bug** in Maven's test execution, and our patch has been accepted and merged in Maven
- **Evaluation** of ForkScript and the Maven with our patch on 29 open-source projects totaling 2M LOC
- **Implications and lessons learned**

Maven's Test Execution: Users' View

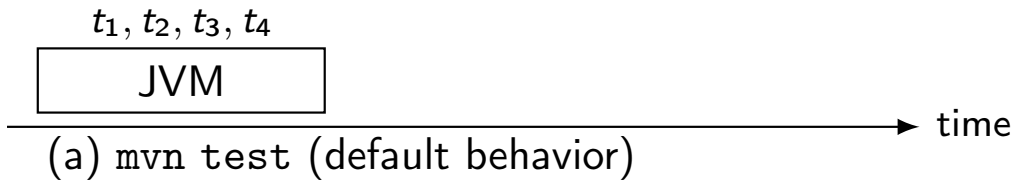
- Maven uses Surefire plugin to manage test execution
- `mvn test`: executes tests with no isolation



(a) `mvn test` (default behavior)

Maven's Test Execution: Users' View

- Maven uses Surefire plugin to manage test execution
- `mvn test`: executes tests with no isolation



(a) `mvn test` (default behavior)

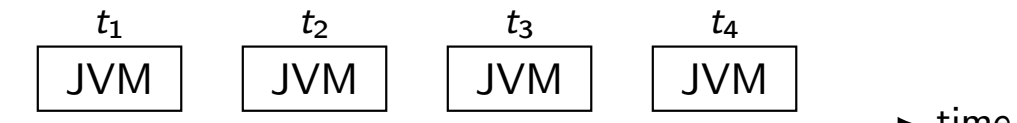
- Test isolation with `-DreuseForks` and `-DforkCount`

Maven's Test Execution: Users' View

- Maven uses Surefire plugin to manage test execution
- `mvn test`: executes tests with no isolation

(a) `mvn test` (default behavior)

- Test isolation with `-DreuseForks` and `-DforkCount`



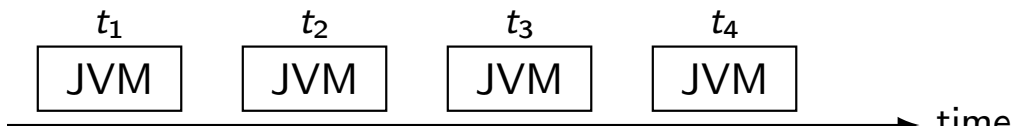
(b) `-DreuseForks=false -DforkCount=1`

Maven's Test Execution: Users' View

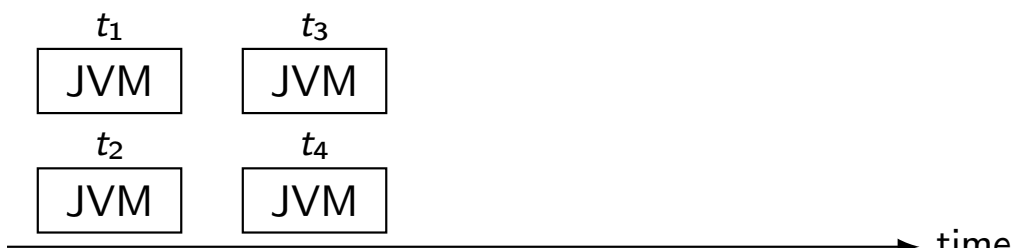
- Maven uses Surefire plugin to manage test execution
- `mvn test`: executes tests with no isolation

(a) `mvn test` (default behavior)

- Test isolation with `-DreuseForks` and `-DforkCount`

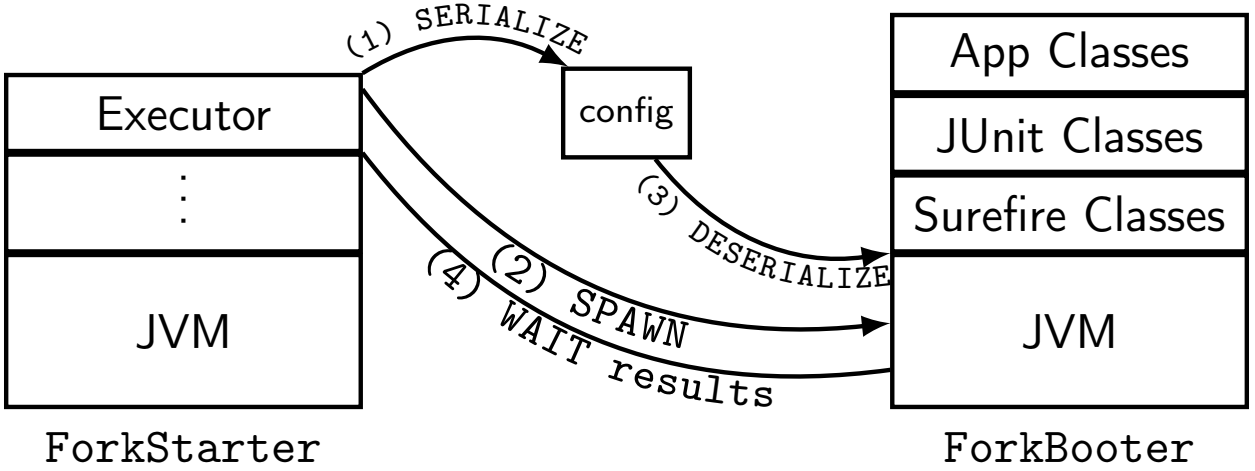


(b) `-DreuseForks=false -DforkCount=1`



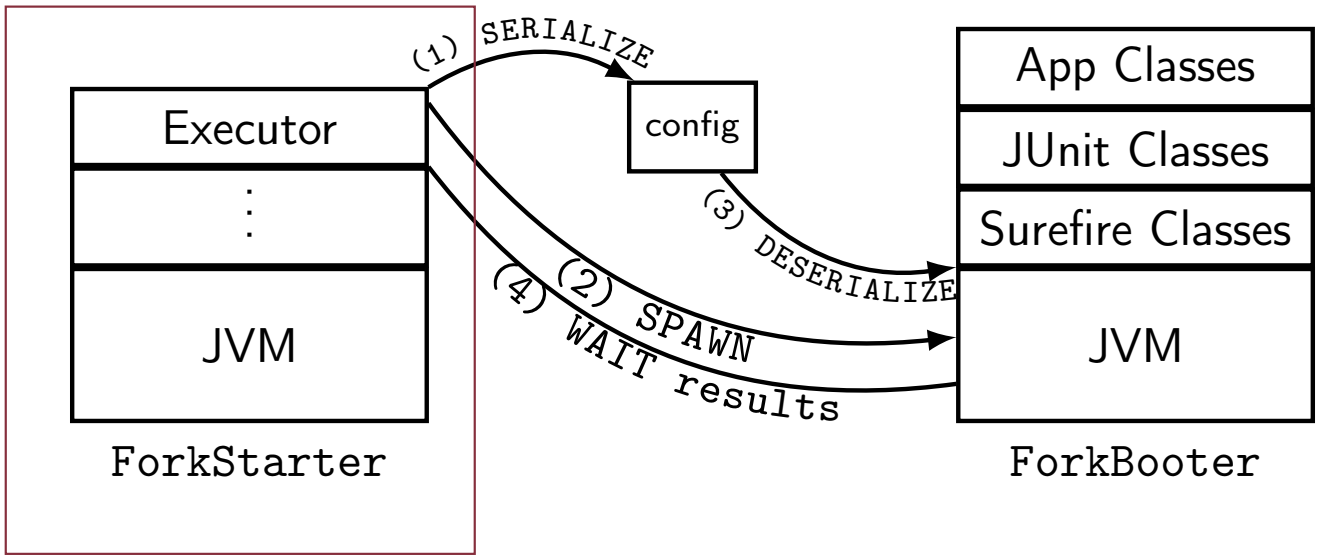
(c) `-DreuseForks=false -DforkCount=2`

Maven's Test Execution: Behind the Scenes



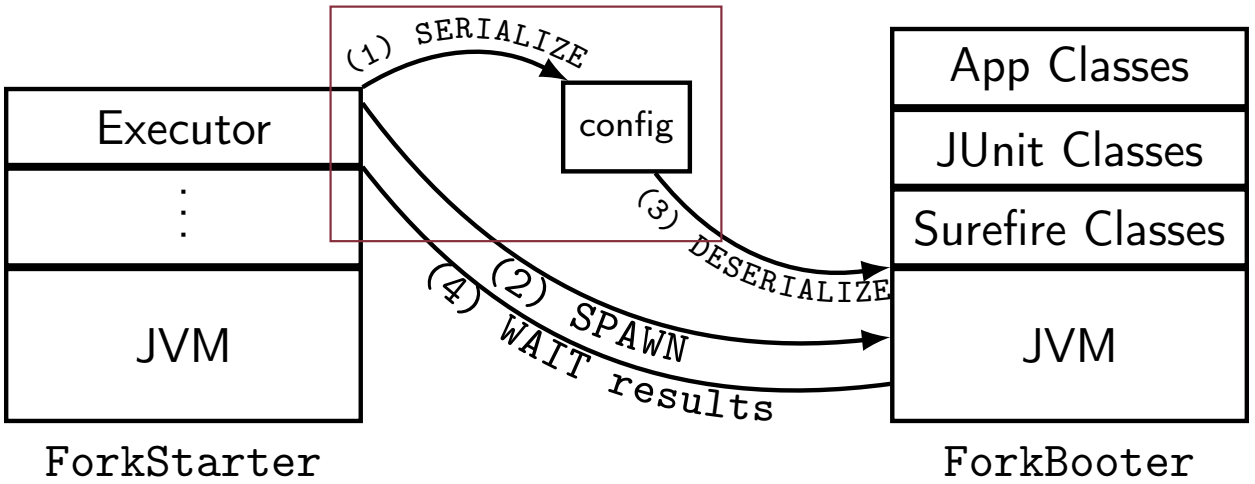
- Two key classes: `ForkStarter` and `ForkBooter`

Maven's Test Execution: Behind the Scenes



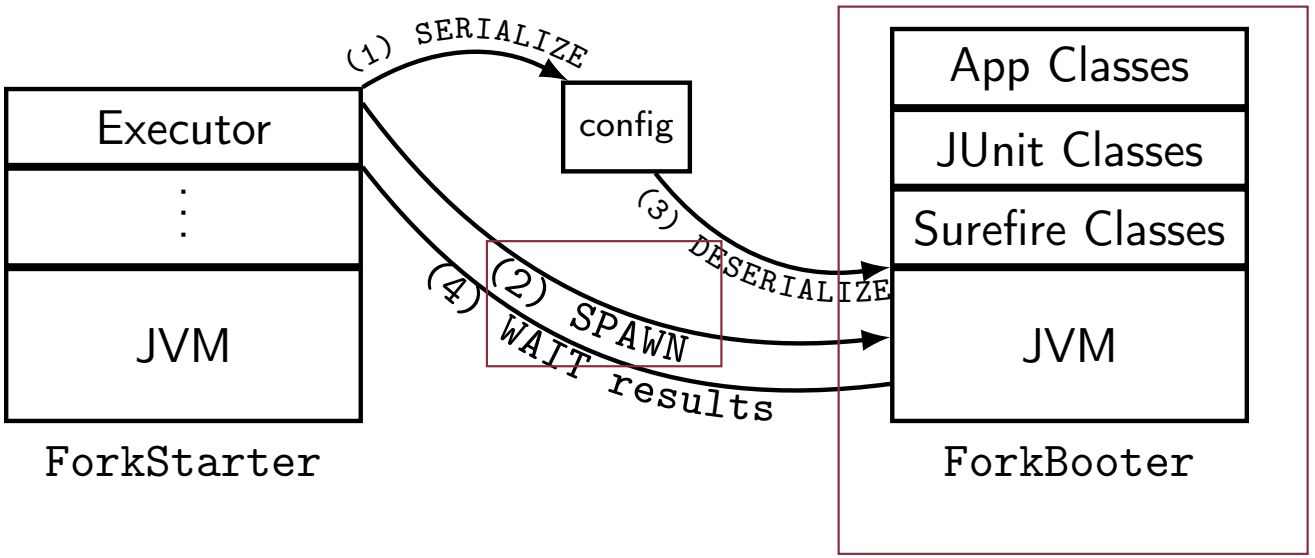
- Two key classes: `ForkStarter` and `ForkBooter`
- `ForkStarter` creates an `Executor` (thread pool)

Maven's Test Execution: Behind the Scenes



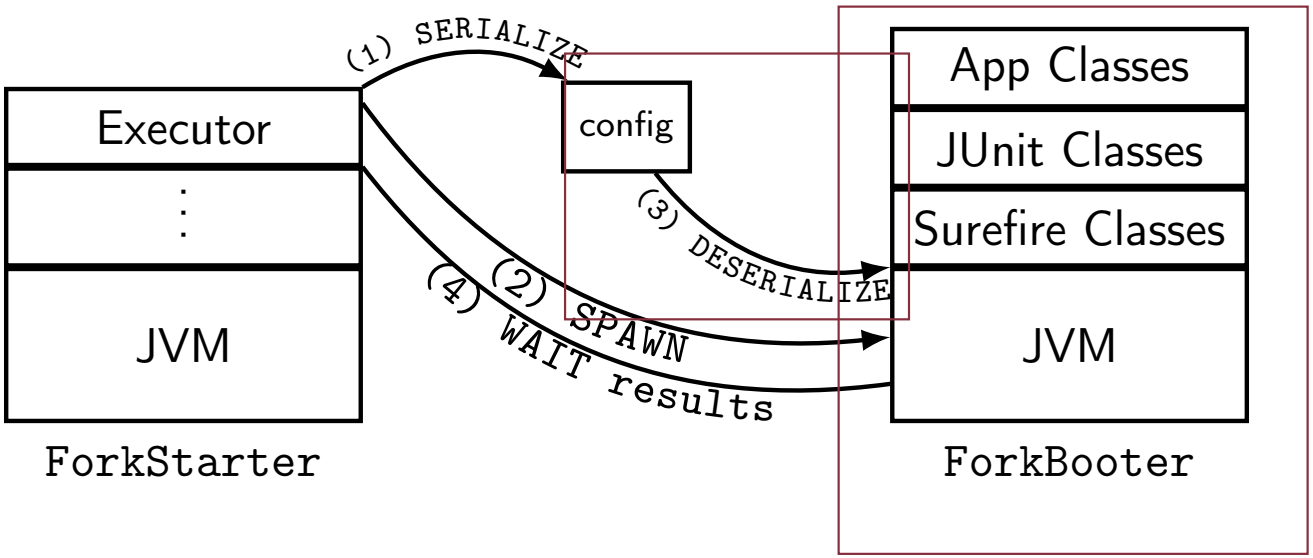
- Two key classes: `ForkStarter` and `ForkBooter`
- `ForkStarter` creates an `Executor` (thread pool)
- For each test:
  - `ForkStarter` serializes configurations to file

Maven's Test Execution: Behind the Scenes



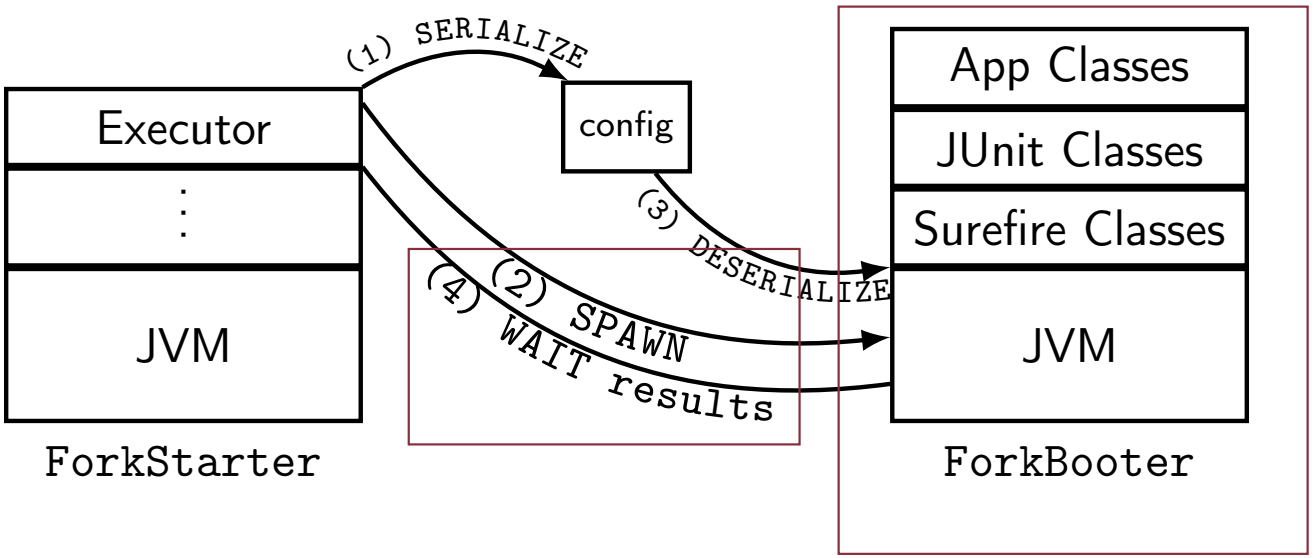
- Two key classes: `ForkStarter` and `ForkBooter`
- `ForkStarter` creates an `Executor` (thread pool)
- For each test:
  - `ForkStarter` serializes configurations to file
  - `ForkStarter` spawns a child JVM w/ main class `ForkBooter`

Maven's Test Execution: Behind the Scenes



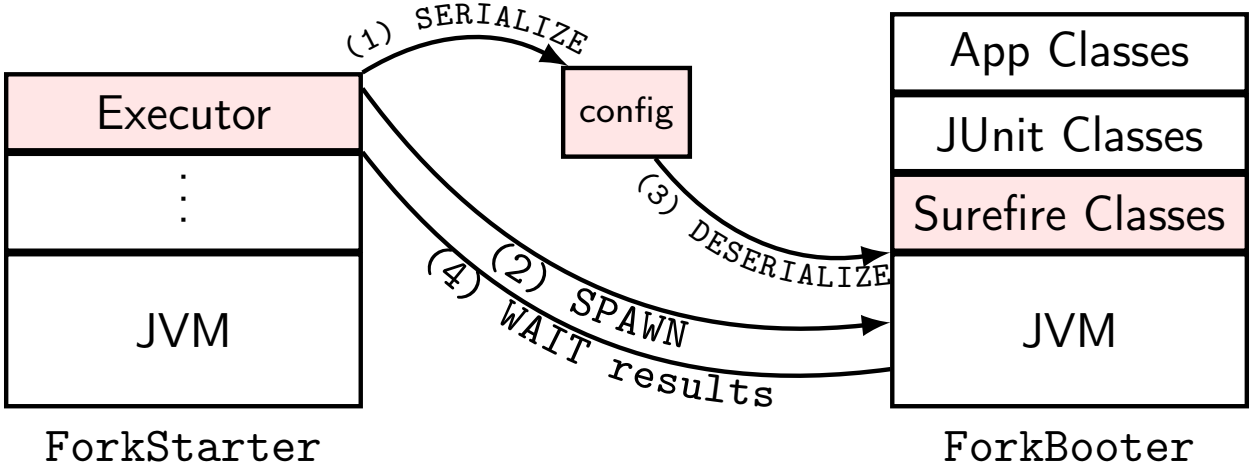
- Two key classes: `ForkStarter` and `ForkBooter`
- `ForkStarter` creates an `Executor` (thread pool)
- For each test:
  - `ForkStarter` serializes configurations to file
  - `ForkStarter` spawns a child JVM w/ main class `ForkBooter`
  - `ForkBooter` deserializes configurations from file

Maven's Test Execution: Behind the Scenes



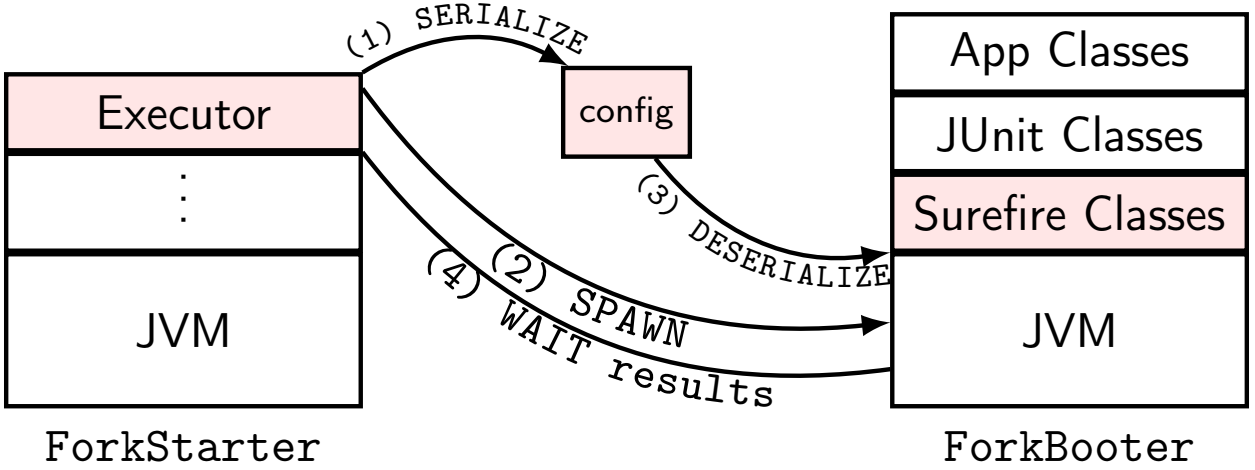
- Two key classes: `ForkStarter` and `ForkBooter`
- `ForkStarter` creates an `Executor` (thread pool)
- For each test:
  - `ForkStarter` serializes configurations to file
  - `ForkStarter` spawns a child JVM w/ main class `ForkBooter`
  - `ForkBooter` deserializes configurations from file
  - `ForkBooter` executes the test with JUnit
  - `ForkStarter` waits for `ForkBooter` to send a "goodbye" signal when the test finishes

Maven's Test Execution: Behind the Scenes



- Inter process communication (IPC) is costly
  - Using thread pool and executors to manage processes
  - Exchanging configuration with new JVMs via (de)serialization
  - Class loading of Surefire's classes
  - "Pumping" input/output between the JVMs

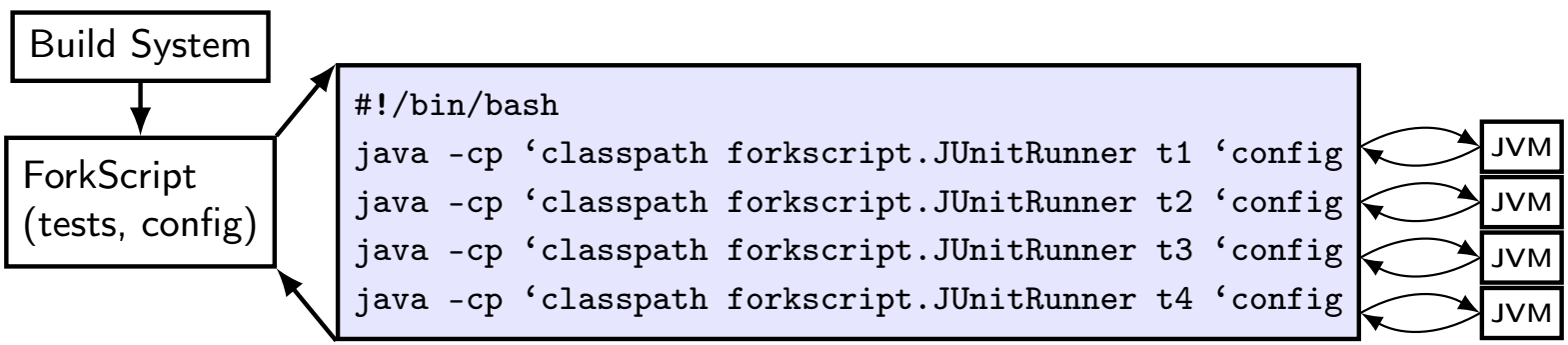
Maven's Test Execution: Behind the Scenes



- Inter process communication (IPC) is costly
  - ~~Using thread pool and executors to manage processes~~
  - ~~Exchanging configuration with new JVMs via (de)serialization~~
  - ~~Class loading of Surefire's classes~~
  - ~~"Pumping" input/output between the JVMs~~

ForkScript

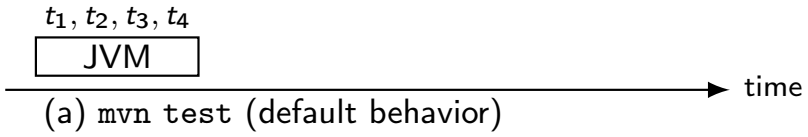
- ForkScript generates a single **on-the-fly specialized execution script** for running all configured tests and collecting test results
- **No IPC** between the build system and test processes
- Relies on operating system's process management



on-the-fly script (simplified)

ForkScript Scripts Examples

- ForkScript supports test isolation, sequential and parallel testing

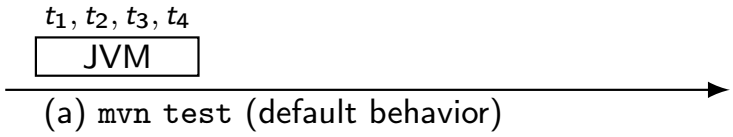


(a) `mvn test` (default behavior)

```
#!/bin/bash
java -cp ‘classpath forkscript.JUnitRunner t1 t2 t3 t4 ‘config
```

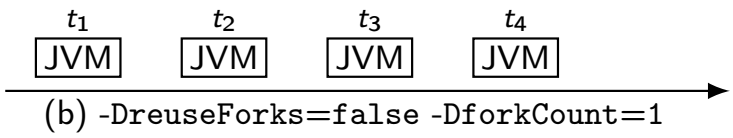
ForkScript Scripts Examples

- ForkScript supports test isolation, sequential and parallel testing



(a) `mvn test` (default behavior)

```
#!/bin/bash
java -cp ‘classpath forkscript.JUnitRunner t1 t2 t3 t4 ‘config
```

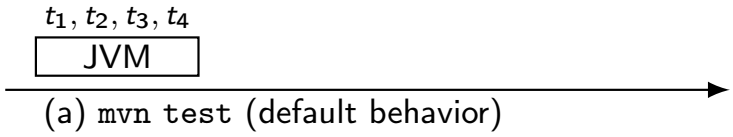


(b) `-DreuseForks=false -DforkCount=1`

```
#!/bin/bash
java -cp ‘classpath forkscript.JUnitRunner t1 ‘config
java -cp ‘classpath forkscript.JUnitRunner t2 ‘config
java -cp ‘classpath forkscript.JUnitRunner t3 ‘config
java -cp ‘classpath forkscript.JUnitRunner t4 ‘config
```

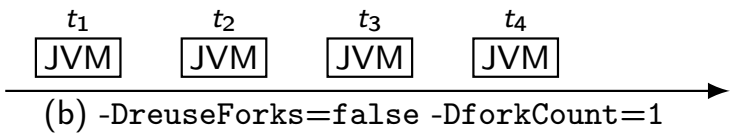
ForkScript Scripts Examples

- ForkScript supports test isolation, sequential and parallel testing



(a) `mvn test` (default behavior)

```
#!/bin/bash
java -cp ‘classpath forkscript.JUnitRunner t1 t2 t3 t4 ‘config
```



(b) `-DreuseForks=false -DforkCount=1`

```
#!/bin/bash
java -cp ‘classpath forkscript.JUnitRunner t1 ‘config
java -cp ‘classpath forkscript.JUnitRunner t2 ‘config
java -cp ‘classpath forkscript.JUnitRunner t3 ‘config
java -cp ‘classpath forkscript.JUnitRunner t4 ‘config
```

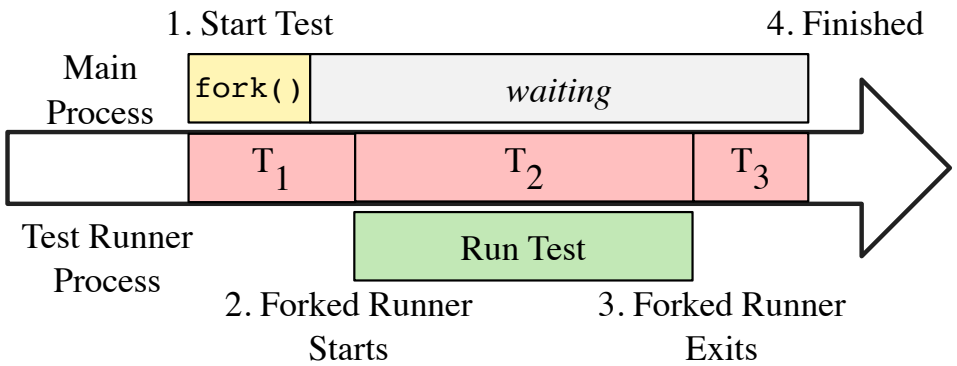
(c) `-DreuseForks=false -DforkCount=2`

```
#!/bin/bash
java -cp ‘classpath forkscript.JUnitRunner t1 ‘config &
java -cp ‘classpath forkscript.JUnitRunner t2 ‘config &
wait
java -cp ‘classpath forkscript.JUnitRunner t3 ‘config &
java -cp ‘classpath forkscript.JUnitRunner t4 ‘config &
wait
```

Performance Profiling Maven

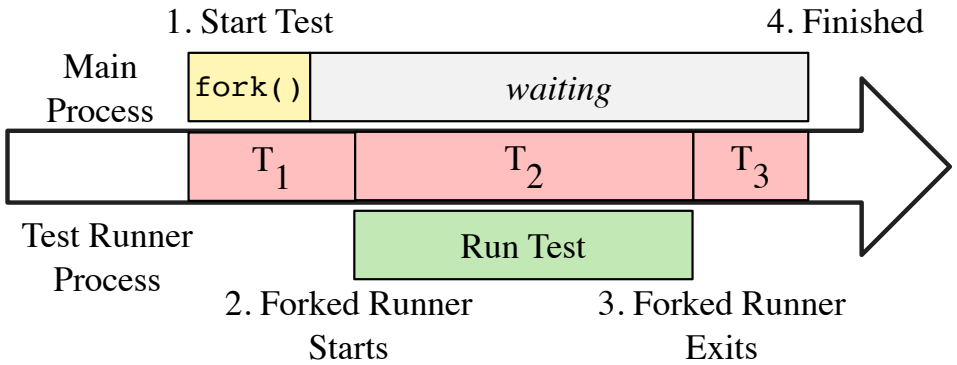
- ForkScript provides a barebones, stripped down mechanism for test isolation, but doesn't support all configuration options
- We also carefully **profiled** Maven to identify the source of the additional overhead

Performance Profiling Maven: Setup



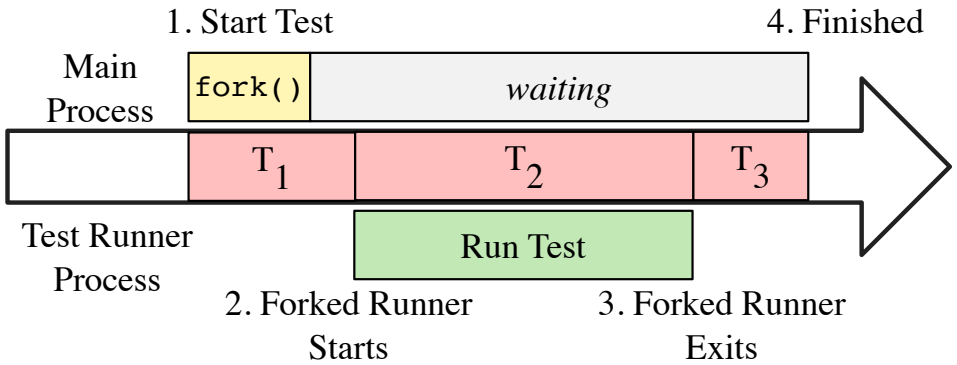
- T_1: **between** when the build system begins running a test **until** the child process starts
- T_2: **between** when the child process starts **until** the child process terminates
- T_3: **between** when the child process terminates **until** when the build system determines the test has completed

Performance Profiling Maven: Findings



Build System	T_1[ms]	T_2[ms]	T_3[ms]
Ant 1.10.6	250	253	9
Gradle 5.6.1	395	253	17
Maven (Surefire 3.0.0-M3)	244	253	**352**

Performance Profiling Maven: Findings



Build System	T_1[ms]	T_2[ms]	T_3[ms]
Ant 1.10.6	250	253	9
Gradle 5.6.1	395	253	17
Maven (Surefire 3.0.0-M3)	244	253	**352**

- Performance bug in Maven: child process keeps reading from ⟨stdin⟩, so it cannot be interrupted (terminated) immediately

Performance Profiling Maven: Patch

- To fix the performance bug, we went over many iterations with Maven developers for several months
- First, we prepared a **large patch** that removed all sources of the overhead, but it was hard for developers to review and integrate

Performance Profiling Maven: Patch

- To fix the performance bug, we went over many iterations with Maven developers for several months
- First, we prepared a **large patch** that removed all sources of the overhead, but it was hard for developers to review and integrate
- Then, we prepared another **small patch** that changed several lines, was easy to inspect, but didn't remove all sources of overhead

Performance Profiling Maven: Patch

- To fix the performance bug, we went over many iterations with Maven developers for several months
- First, we prepared a **large patch** that removed all sources of the overhead, but it was hard for developers to review and integrate
- Then, we prepared another **small patch** that changed several lines, was easy to inspect, but didn't remove all sources of overhead
- The small patch was **merged to Maven Surefire 3.0.0-M5**

Performance Profiling Maven: Patch

- To fix the performance bug, we went over many iterations with Maven developers for several months
- First, we prepared a **large patch** that removed all sources of the overhead, but it was hard for developers to review and integrate
- Then, we prepared another **small patch** that changed several lines, was easy to inspect, but didn't remove all sources of overhead
- The small patch was **merged to Maven Surefire 3.0.0-M5**

Build System	T_1[ms]	T_2[ms]	T_3[ms]
Maven (Surefire 3.0.0-M3)	244	253	**352**
Maven (With our patch)	217	252	**17**

Evaluation: Research Questions

RQ1 What are the performance improvements obtained by **ForkScript** compared to the **unpatched Maven**?

RQ2 How does the improvement scale as the **number of concurrent processes** increase?

RQ3 How does the **patched Maven** compare to ForkScript?

Evaluation: Subjects

- 29 projects used in recent testing literature, and:
  - use Maven build system
  - have non-trivial number of tests
  - have tests whose execution time is non-negligible
  - successfully build at its latest revision

Evaluation: Subjects

- 29 projects used in recent testing literature, and:
  - use Maven build system
  - have non-trivial number of tests
  - have tests whose execution time is non-negligible
  - successfully build at its latest revision
- LOC: total 2.12M, average 73.0K
- number of test classes: total 6.14K, average 211
- number of test methods: total 209K, average 7.22K

Evaluation: Setup

For each project:

- Clone the project
- Execute `mvn install` to download all necessary dependencies, then switch to offline mode
- Run tests using {ForkScript, unpatched Maven, patched Maven} and measure time

Evaluation Results: Sequential Runs

RQ1 What are the performance improvements obtained by **ForkScript** compared to the **unpatched Maven**?

- `mvn test -DreuseForks=false -DforkCount=1`
- T^{mvn}: Maven; T^{FS}: ForkScript; $RT = \frac{T^{mvn} - T^{FS}}{T^{mvn}} \times 100\%$

	T^{mvn} [s]	T^{FS} [s]	RT
Avg.	154.66	80.74	50%
Σ	4,485.16	2,341.60	

Evaluation Results: Sequential Runs

RQ1 What are the performance improvements obtained by **ForkScript** compared to the **unpatched Maven**?

- `mvn test -DreuseForks=false -DforkCount=1`
- T^{mvn}: Maven; T^{FS}: ForkScript; $RT = \frac{T^{mvn} - T^{FS}}{T^{mvn}} \times 100\%$

	T^{mvn} [s]	T^{FS} [s]	RT
Avg.	154.66	80.74	50%
Σ	4,485.16	2,341.60	

- ForkScript reduces testing time by **50%** on average and up to **75%**
- Projects with smaller tests (lower time per test) benefit more

Evaluation Results: Parallel Runs

RQ2 How does the improvement scale as the **number of concurrent processes** increase?

- `mvn test -DreuseForks=false -DforkCount=2`
- T^{mvn}: Maven; T^{FS}: ForkScript; $RT = \frac{T^{mvn} - T^{FS}}{T^{mvn}} \times 100\%$

	T^{mvn} [s]	T^{FS} [s]	RT
Avg.	72.02	49.88	32%
Σ	2,088.81	1,446.70	

Evaluation Results: Parallel Runs

RQ2 How does the improvement scale as the **number of concurrent processes** increase?

- `mvn test -DreuseForks=false -DforkCount=2`
- T^{mvn}: Maven; T^{FS}: ForkScript; $RT = \frac{T^{mvn} - T^{FS}}{T^{mvn}} \times 100\%$

	T^{mvn} [s]	T^{FS} [s]	RT
Avg.	72.02	49.88	32%
Σ	2,088.81	1,446.70	

- ForkScript reduces testing time by **32%** on average and up to **63%**
- The reduction in savings compared to sequential runs is due to total execution time approaches theoretical maximum (i.e., time to execute the longest test)

Evaluation Results: Comparison with Patched Maven

RQ3 How does the **patched Maven** compare to ForkScript?

- `mvn test -DreuseForks=false`
  Fork 1: `-DforkCount=1`; *Fork 2*: `-DforkCount=2`
- T^{mvn}: Maven; T^{FS}: ForkScript; T^{new}: patched Maven

	Fork 1			Fork 2		
	T^{mvn}[s]	T^{FS}[s]	T^{new}[s]	T^{mvn}[s]	T^{FS}[s]	T^{new}[s]
Avg.	154.66	80.74	95.88	72.02	49.88	55.45
Σ	4,485.16	2,341.60	2,780.54	2,088.81	1,446.70	1,608.06

Evaluation Results: Comparison with Patched Maven

RQ3 How does the **patched Maven** compare to ForkScript?

- `mvn test -DreuseForks=false`
  Fork 1: `-DforkCount=1`; *Fork 2*: `-DforkCount=2`
- T^{mvn}: Maven; T^{FS}: ForkScript; T^{new}: patched Maven

	Fork 1			Fork 2		
	T^{mvn}[s]	T^{FS}[s]	T^{new}[s]	T^{mvn}[s]	T^{FS}[s]	T^{new}[s]
Avg.	154.66	80.74	95.88	72.02	49.88	55.45
Σ	4,485.16	2,341.60	2,780.54	2,088.81	1,446.70	1,608.06

- Patched Maven substantially outperforms the non-patched version
- ForkScript slightly outperforms patched Maven

Implications and Lessons Learned

- Detect **performance bugs** through **differential testing**
  - Performance bugs are notoriously difficult to find, but when there are alternative systems that accomplish the same goal, differential testing can help to reveal them

Implications and Lessons Learned

- Detect **performance bugs** through **differential testing**
  - Performance bugs are notoriously difficult to find, but when there are alternative systems that accomplish the same goal, differential testing can help to reveal them
- Find simple fixes that can **be integrated today**
  - Our patch is already helping developers while the long-term fix (to completely remove ⟨stdin⟩) is still going on

Implications and Lessons Learned

- Detect **performance bugs** through **differential testing**
  - Performance bugs are notoriously difficult to find, but when there are alternative systems that accomplish the same goal, differential testing can help to reveal them
- Find simple fixes that can **be integrated today**
  - Our patch is already helping developers while the long-term fix (to completely remove ⟨stdin⟩) is still going on
- *Researchers*: **engage** in the open source community
  - Bug fixes, pull requests, Slack channel, etc.
  - Find a balance between research novelty and practical impact

Implications and Lessons Learned

- Detect **performance bugs** through **differential testing**
  - Performance bugs are notoriously difficult to find, but when there are alternative systems that accomplish the same goal, differential testing can help to reveal them
- Find simple fixes that can **be integrated today**
  - Our patch is already helping developers while the long-term fix (to completely remove ⟨stdin⟩) is still going on
- *Researchers*: **engage** in the open source community
  - Bug fixes, pull requests, Slack channel, etc.
  - Find a balance between research novelty and practical impact
- *Researchers*: continue **testing of build systems**

Conclusions

- Demystified why **test isolation** is costly
- Found a performance bug in Maven build system related to **IPC**
- **ForkScript**, a research prototype that minimizes IPC to speed up test isolation
- **Evaluation** on 29 open-source projects totaling 2M LOC
- **Our patch was accepted and merged to Maven**, which is already saving significant test execution time for many developers

Pengyu Nie
pynie@utexas.edu