# An Experimental Study of Methods for Executing Test Suites in Memory Constrained Environments

Suvarshi Bhadra[‡] and Alexander Conrad, Charles Hurkes, Brian Kirklin, Gregory M. Kapfhammer[†]

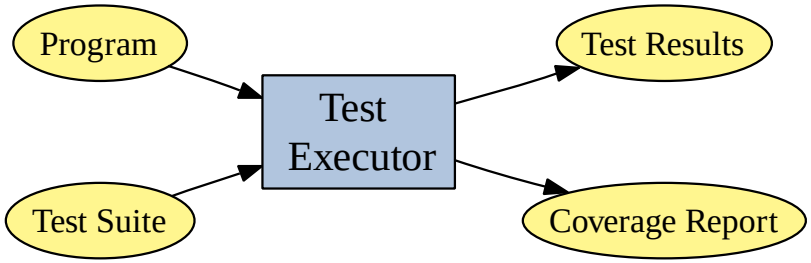
[‡] Milcord LLC

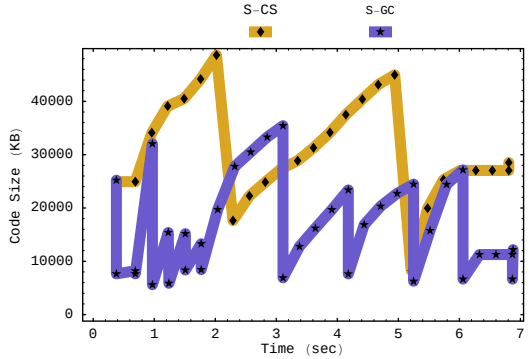
[†] Department of Computer Science
Allegheny College


4th Workshop on the Automation of Software Test (AST 2009)

May 18 - 19, 2009

# Important Contributions



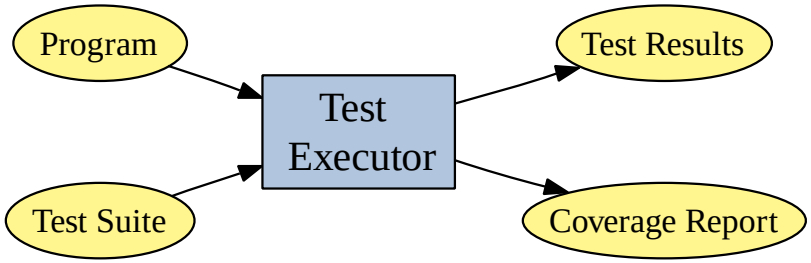



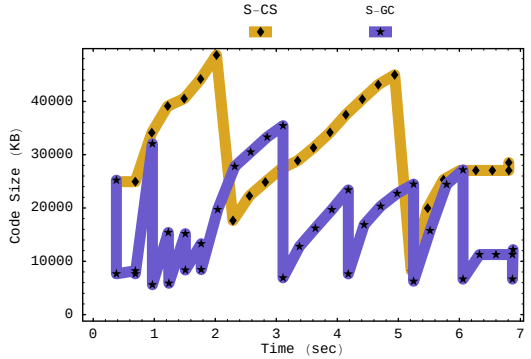

Automated Test Execution

Detailed Empirical Results

**Implement** and empirically **evaluate** the efficiency and effectiveness of techniques for automatically **running** test suites in **memory constrained** execution environments

# Important Contributions



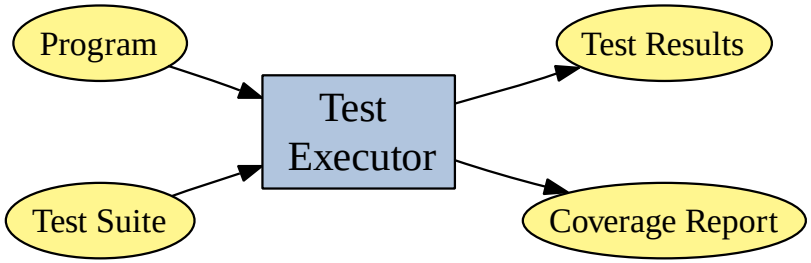



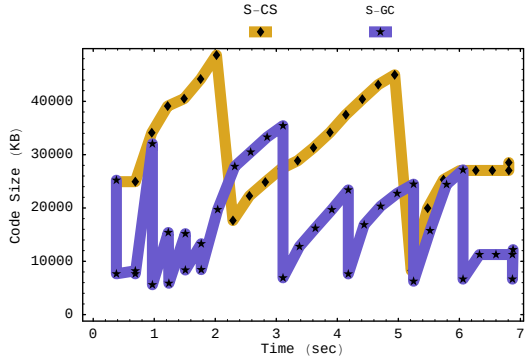

Automated Test Execution

Detailed Empirical Results

**Implement** and empirically **evaluate** the efficiency and effectiveness of techniques for automatically **running** test suites in **memory constrained** execution environments

# Important Contributions



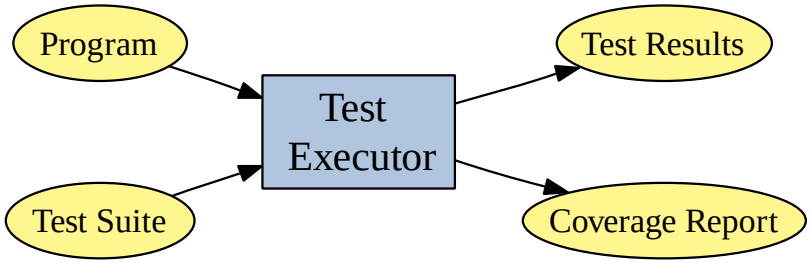



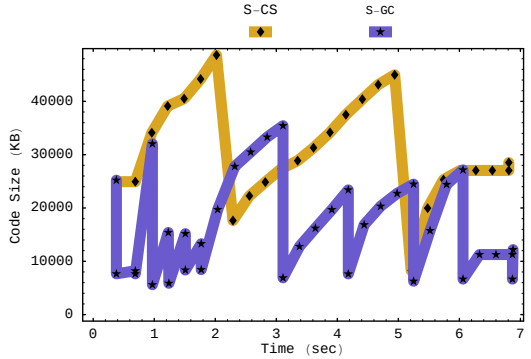

Automated Test Execution

Detailed Empirical Results

**Implement** and empirically **evaluate** the efficiency and effectiveness of techniques for automatically **running** test suites in **memory constrained** execution environments

# Important Contributions



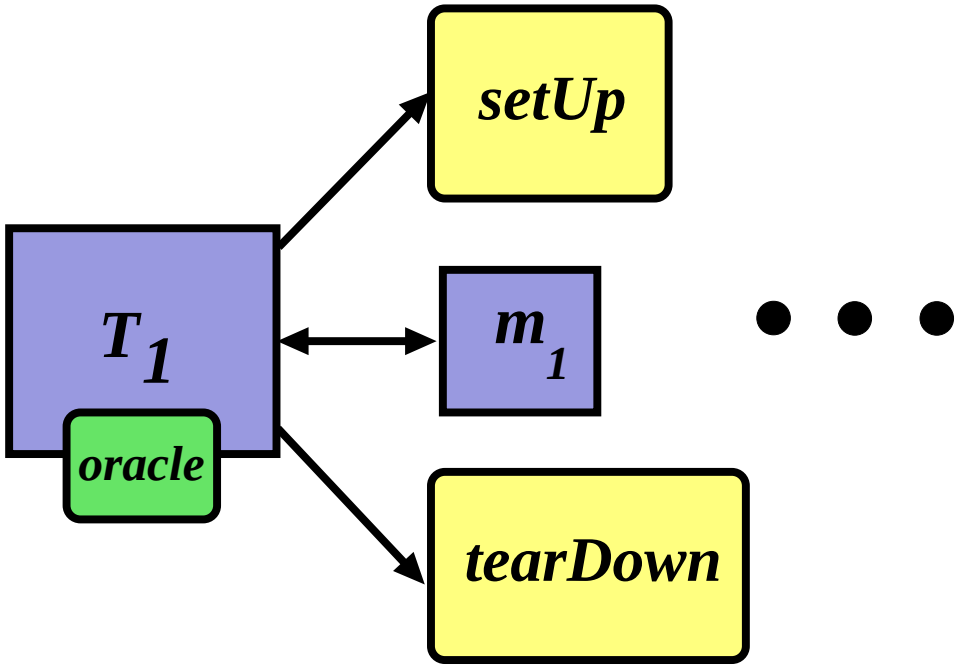




Automated Test Execution

Detailed Empirical Results

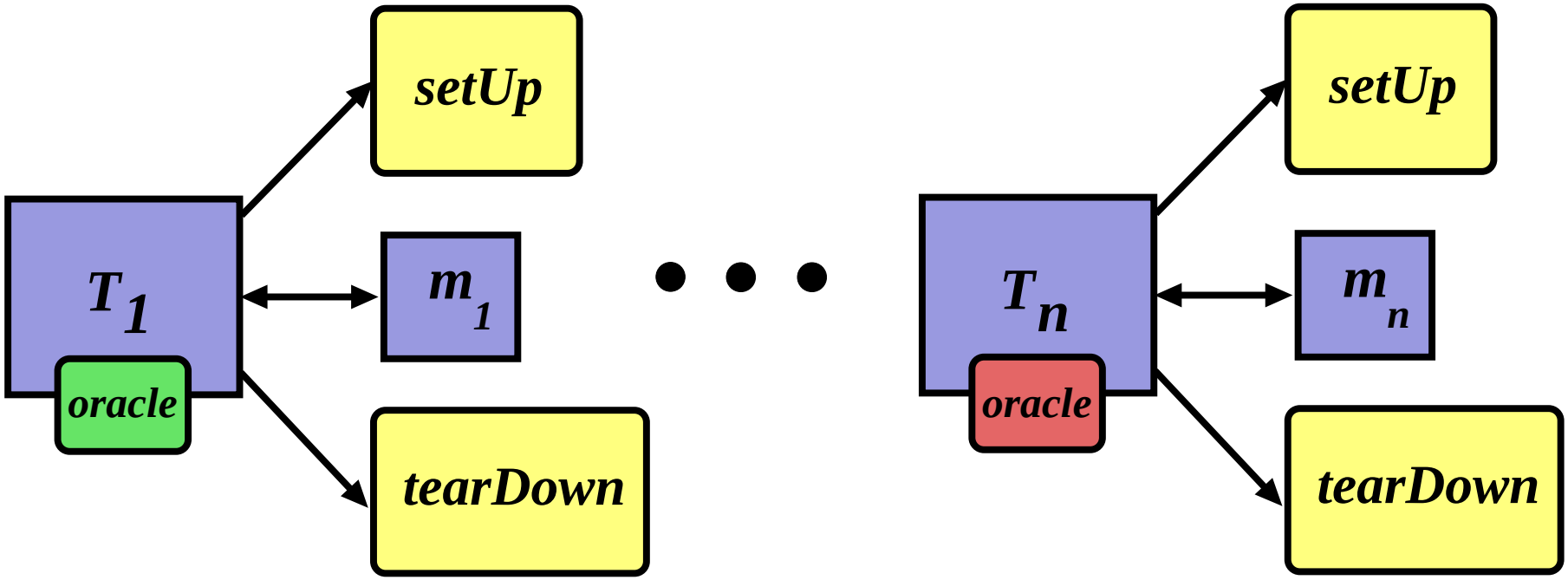
**Implement** and empirically **evaluate** the efficiency and effectiveness of techniques for automatically **running** test suites in **memory constrained** execution environments

# Automated Test Suite Execution



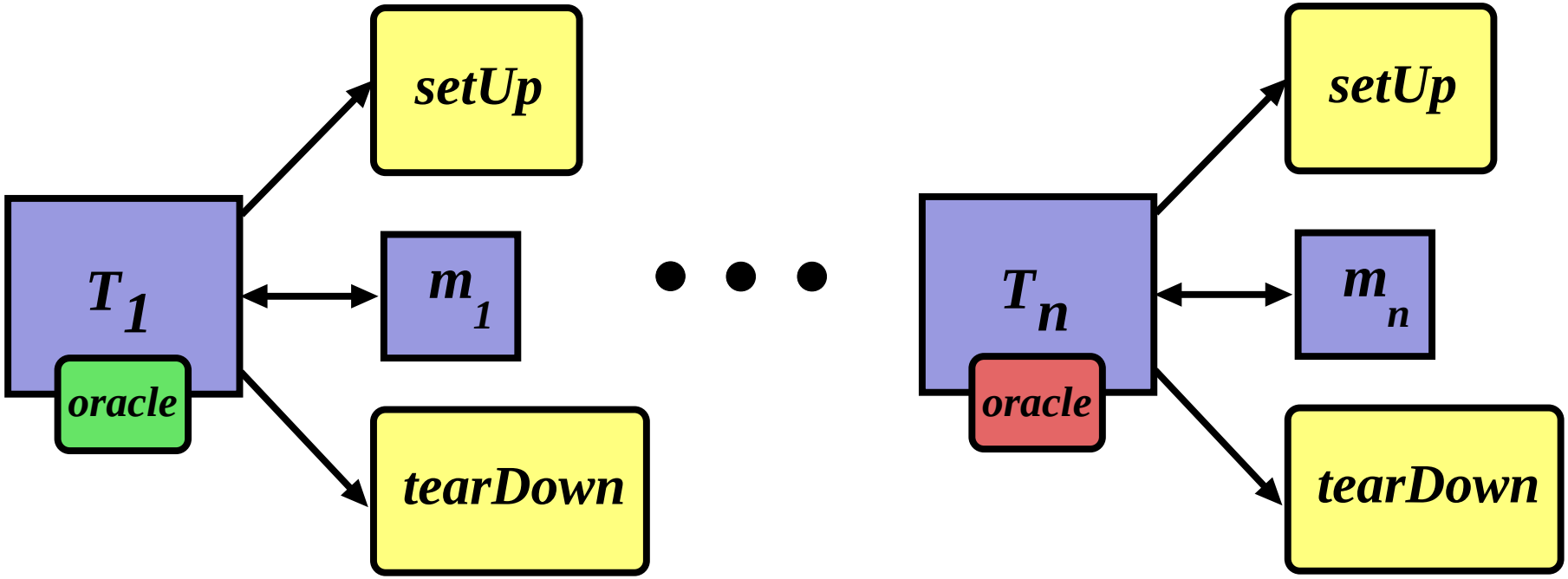

A **test** invokes one or more **methods** under test and uses an **oracle** to determine if the method's **output** is correct

# Automated Test Suite Execution



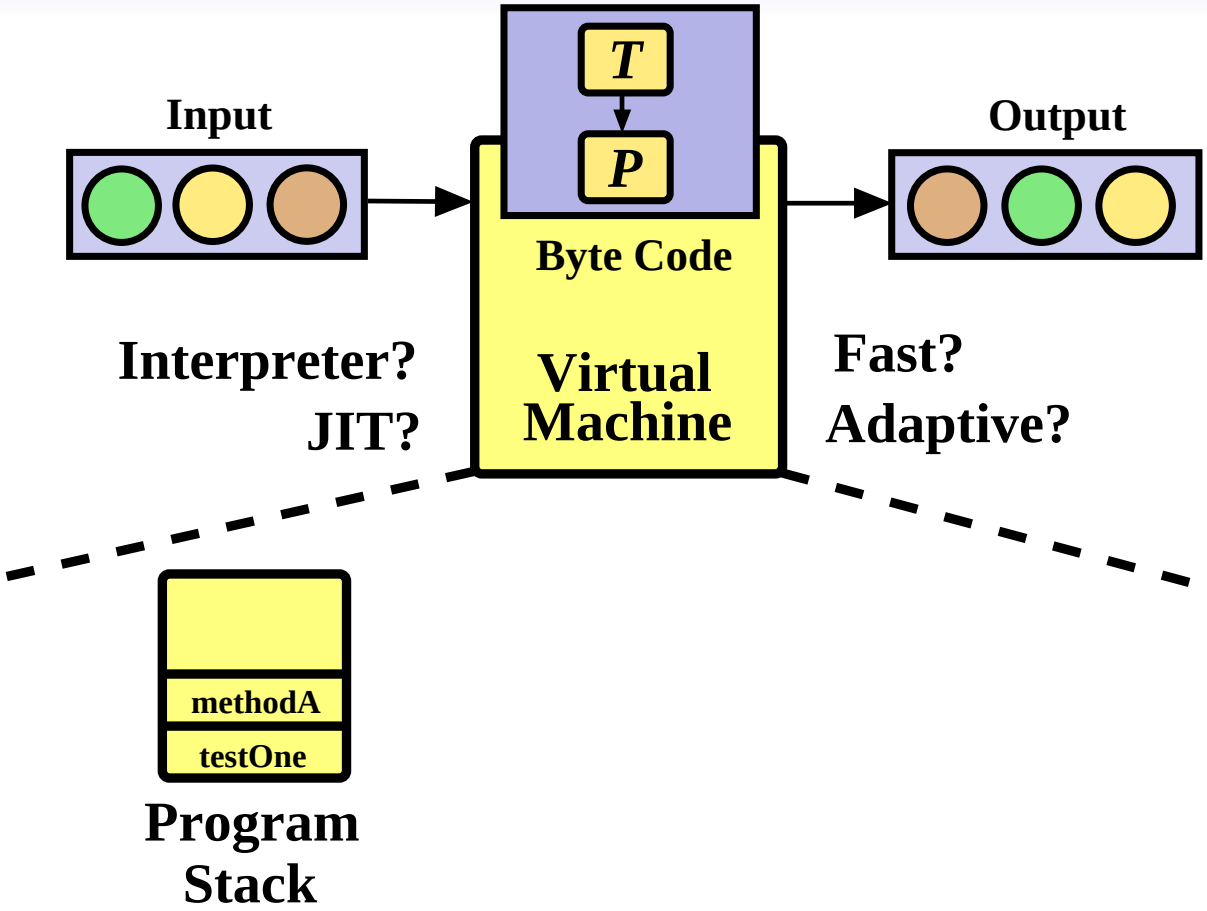

A **test** invokes one or more **methods** under test and uses an **oracle** to determine if the method's **output** is correct

# Automated Test Suite Execution



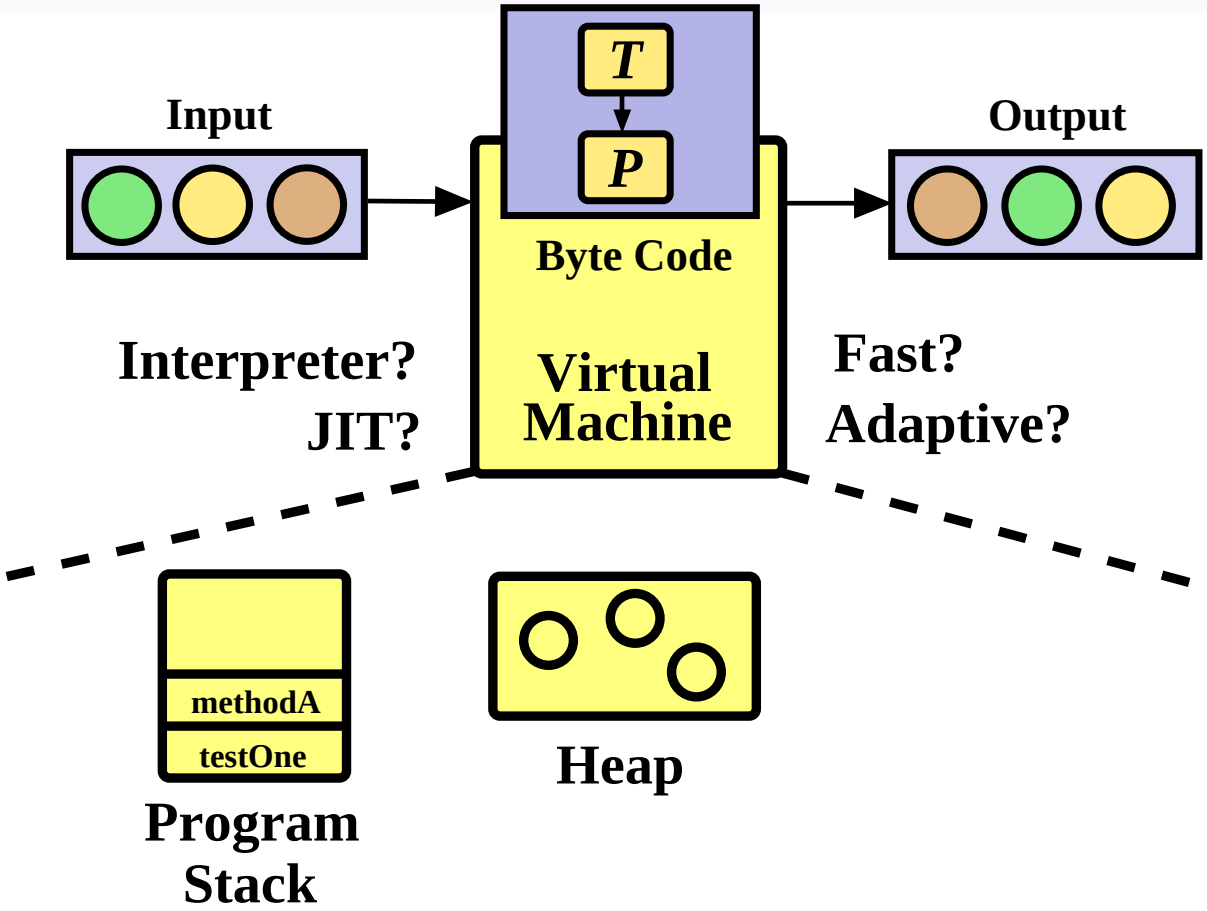

Test suite execution **frameworks** exist for many different programming languages (e.g., JUnit for Java)

# Test Suite Execution Challenges



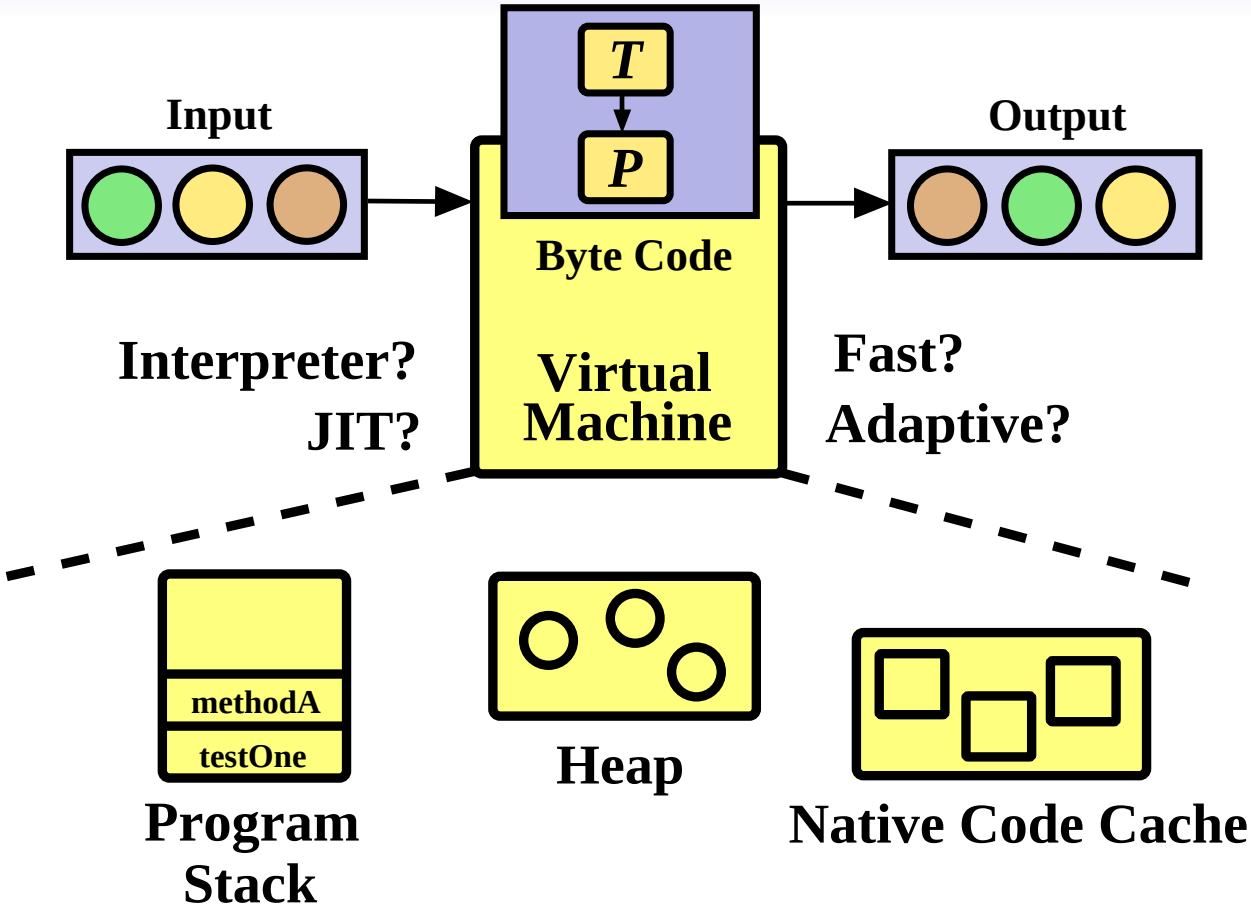

The **virtual machine** manages limited **memory** during testing

# Test Suite Execution Challenges



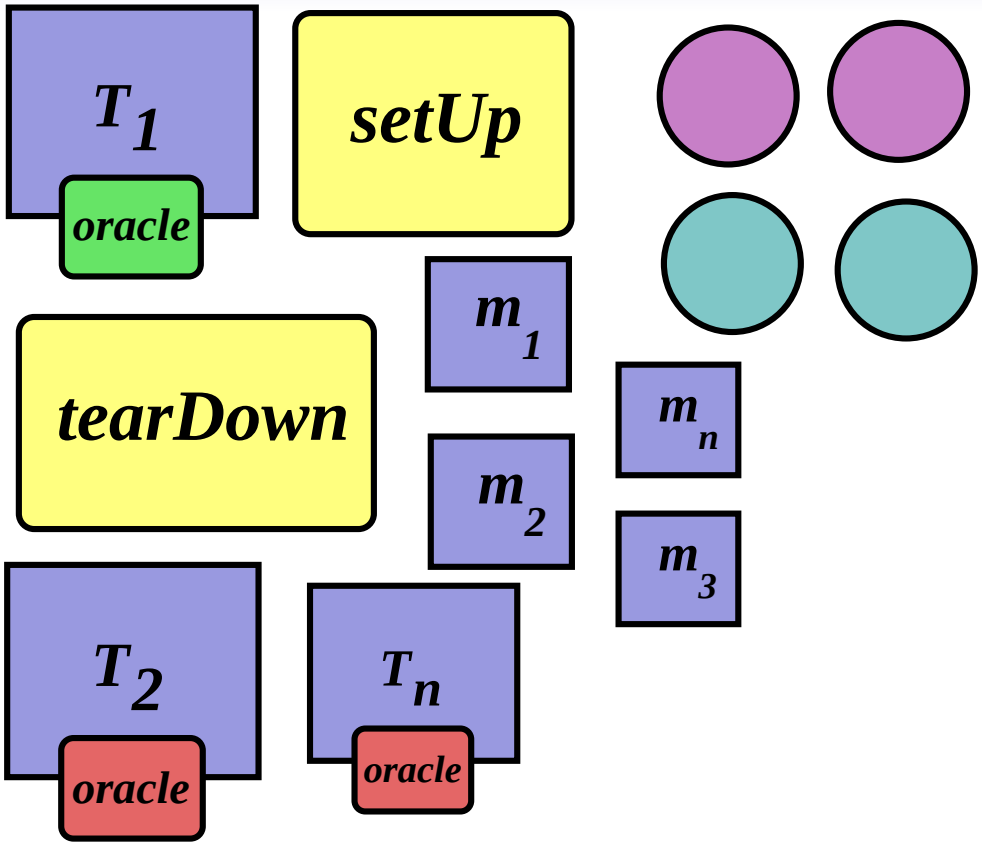

The **virtual machine** manages limited **memory** during testing

# Test Suite Execution Challenges



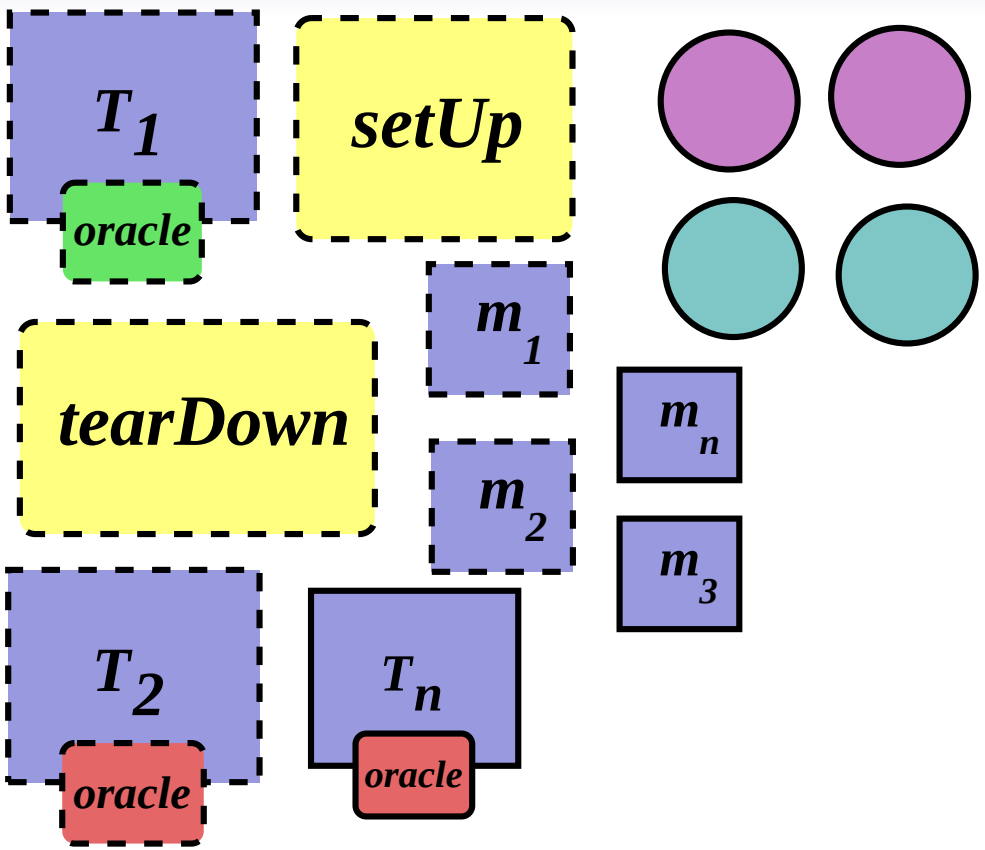

The **virtual machine** manages limited **memory** during testing

# Testing with Memory Constraints



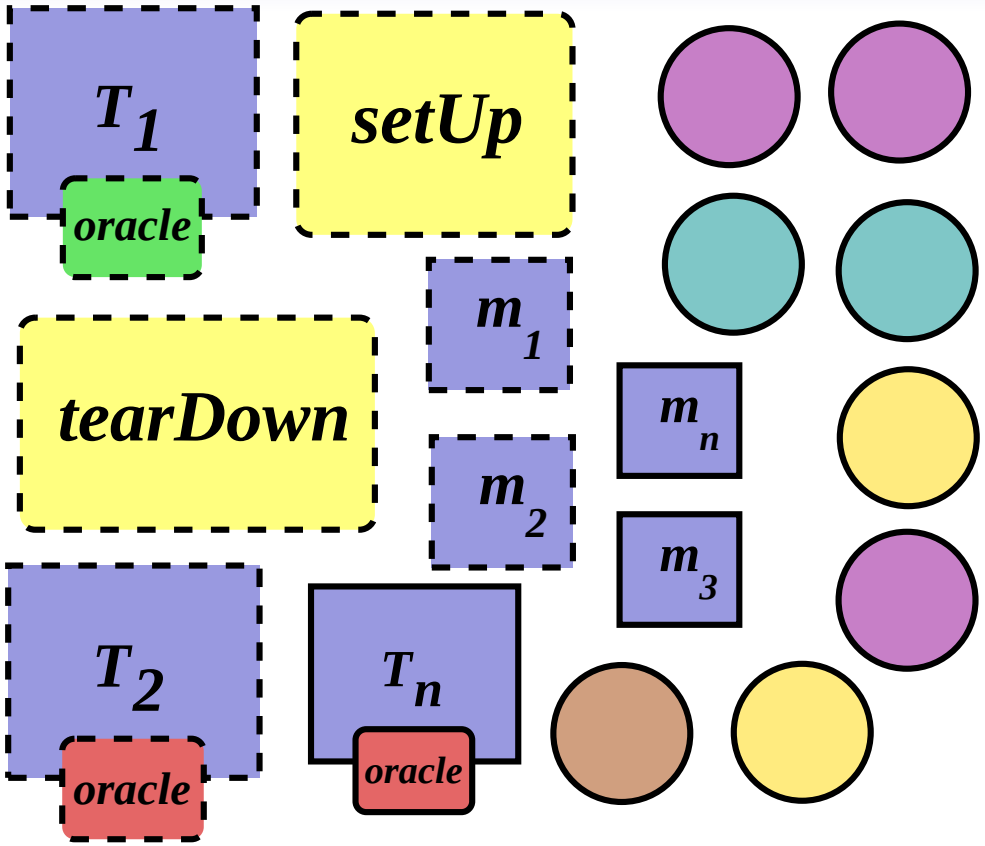

**Startup**: Store bytecodes and the initial objects

# Testing with Memory Constraints



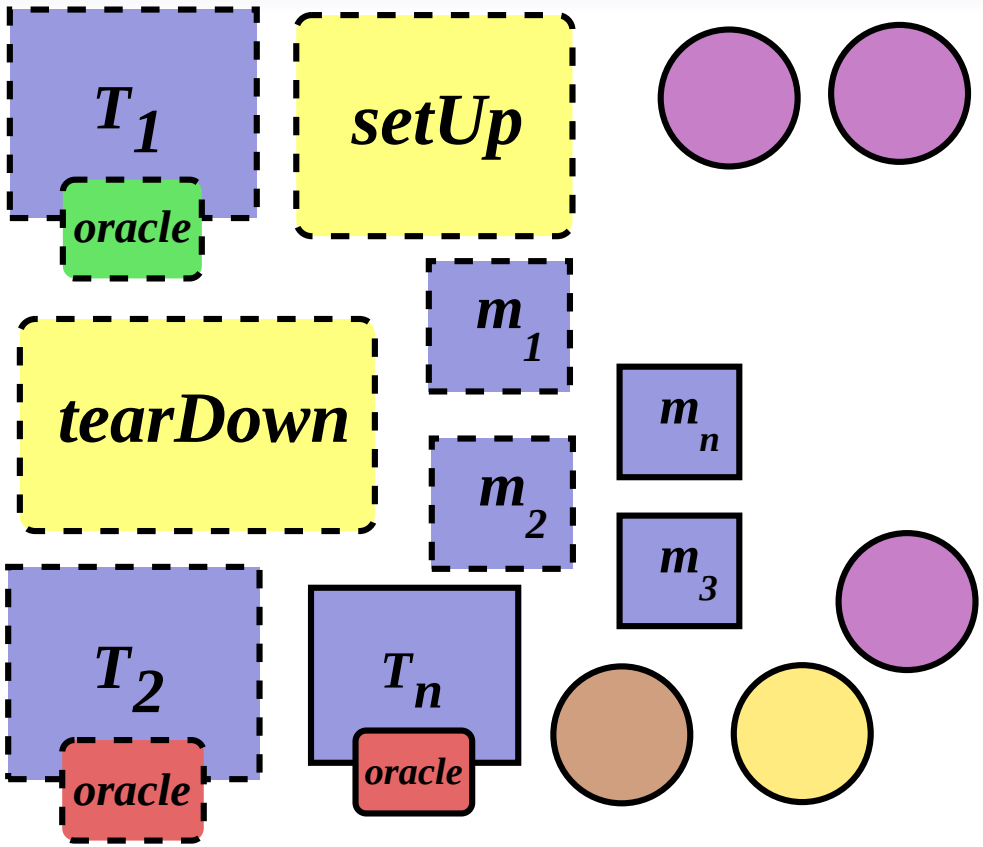

**Optimize**: Create native code from bytecodes

# Testing with Memory Constraints



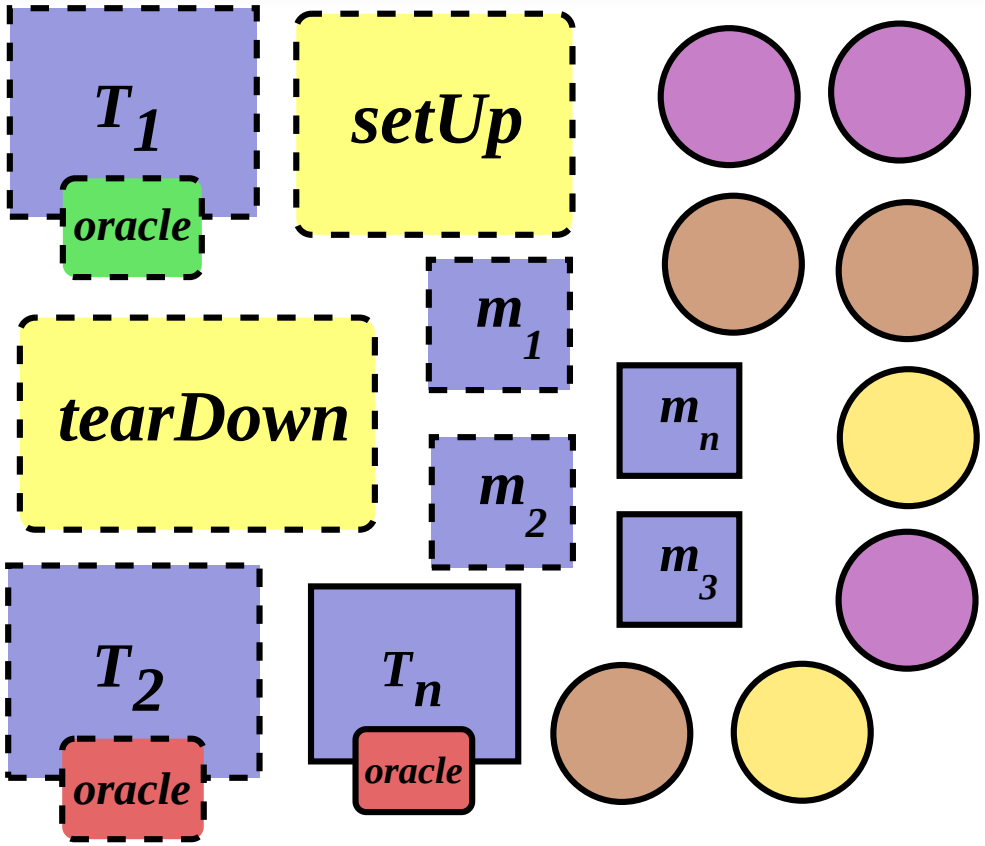

**Threshold**: Allocate too many additional objects

# Testing with Memory Constraints



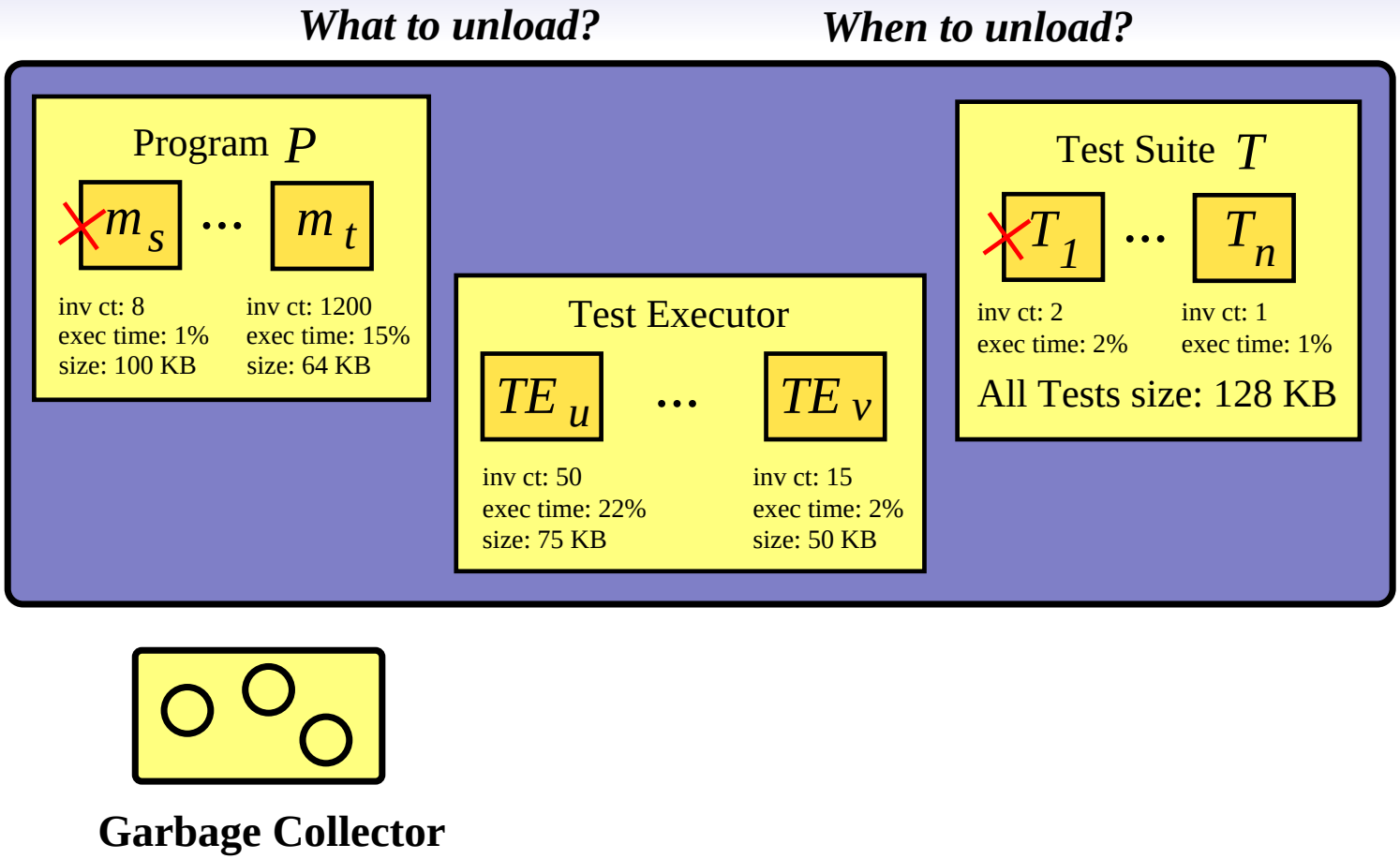

**Collection**: Remove dead objects from the heap

# Testing with Memory Constraints



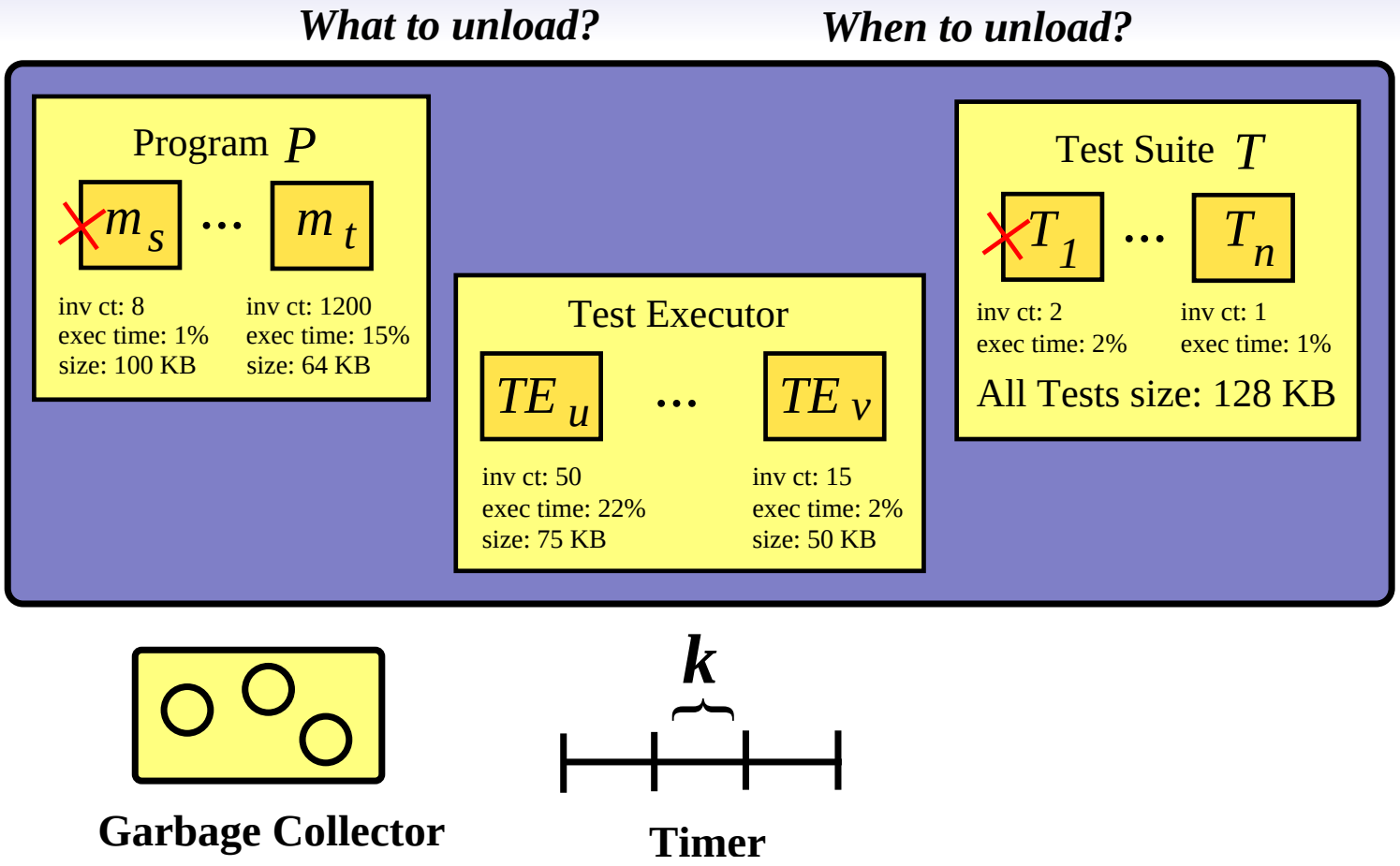

**Problem**: Collector does not remove native code!

# Testing with Native Code Unloading



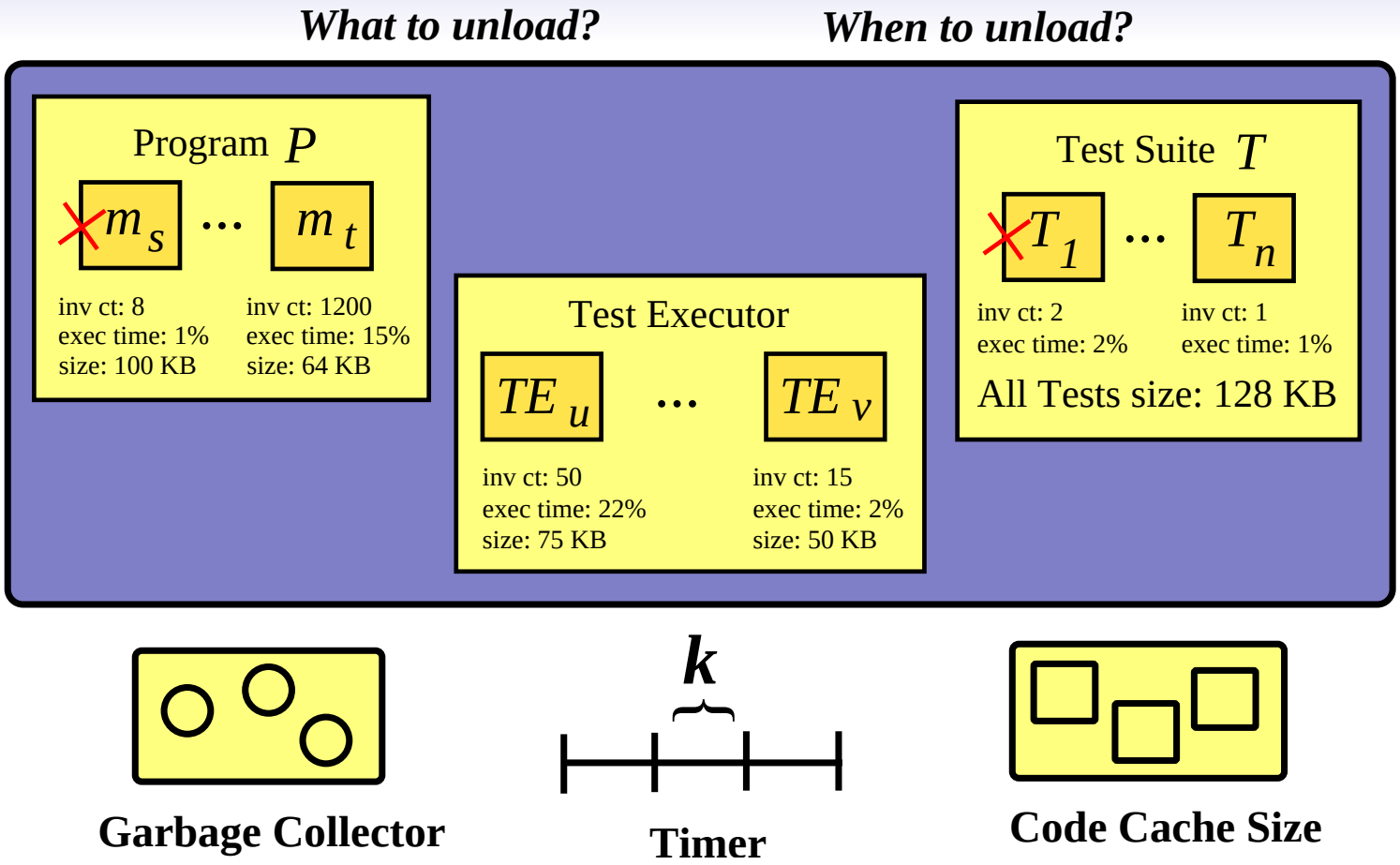

L. Zhang and C. Krintz. *ACM Trans. on Arch. and Code Optim.*

# Testing with Native Code Unloading




L. Zhang and C. Krintz. *ACM Trans. on Arch. and Code Optim.*

# Testing with Native Code Unloading




L. Zhang and C. Krintz. *ACM Trans. on Arch. and Code Optim.*

# Case Study Applications

| Name | *Min* Size (MB) | # Tests | NCSS |
|---|---|---|---|
| UniqueBoundedStack (UBS) | 8 | 24 | 362 |
| Library (L) | 8 | 53 | 551 |
| ShoppingCart (SC) | 8 | 20 | 229 |
| Stack (S) | 8 | 58 | 624 |
| JDepend (JD) | 10 | 53 | 2124 |
| IDTable (ID) | 11 | 24 | 315 |

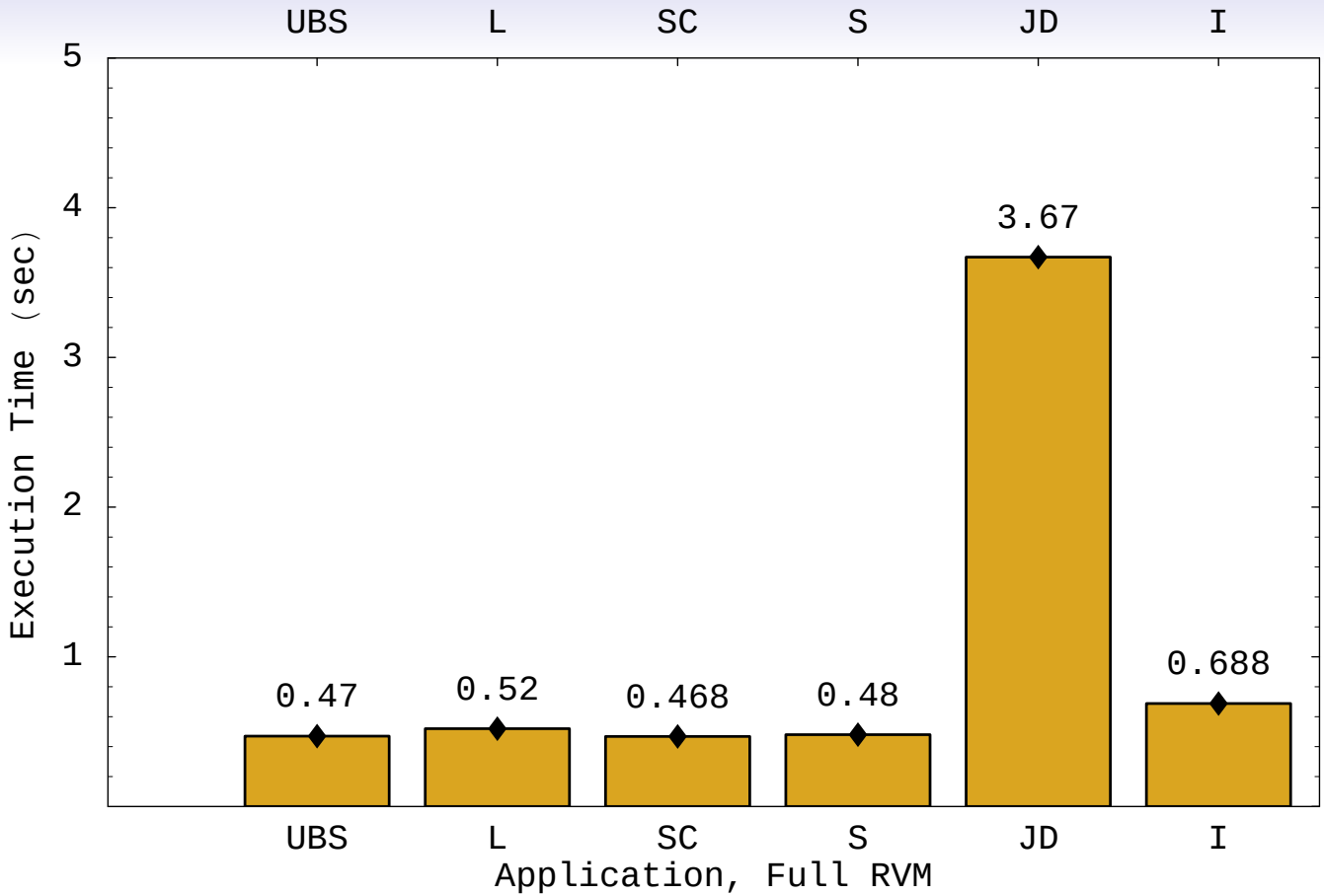
Empirically determined the *Min* Jikes RVM heap size

# Case Study Applications

| Name | *Min* Size (MB) | # Tests | NCSS |
|---|---|---|---|
| UniqueBoundedStack (UBS) | 8 | 24 | 362 |
| Library (L) | 8 | 53 | 551 |
| ShoppingCart (SC) | 8 | 20 | 229 |
| Stack (S) | 8 | 58 | 624 |
| JDepend (JD) | 10 | 53 | 2124 |
| IDTable (ID) | 11 | 24 | 315 |

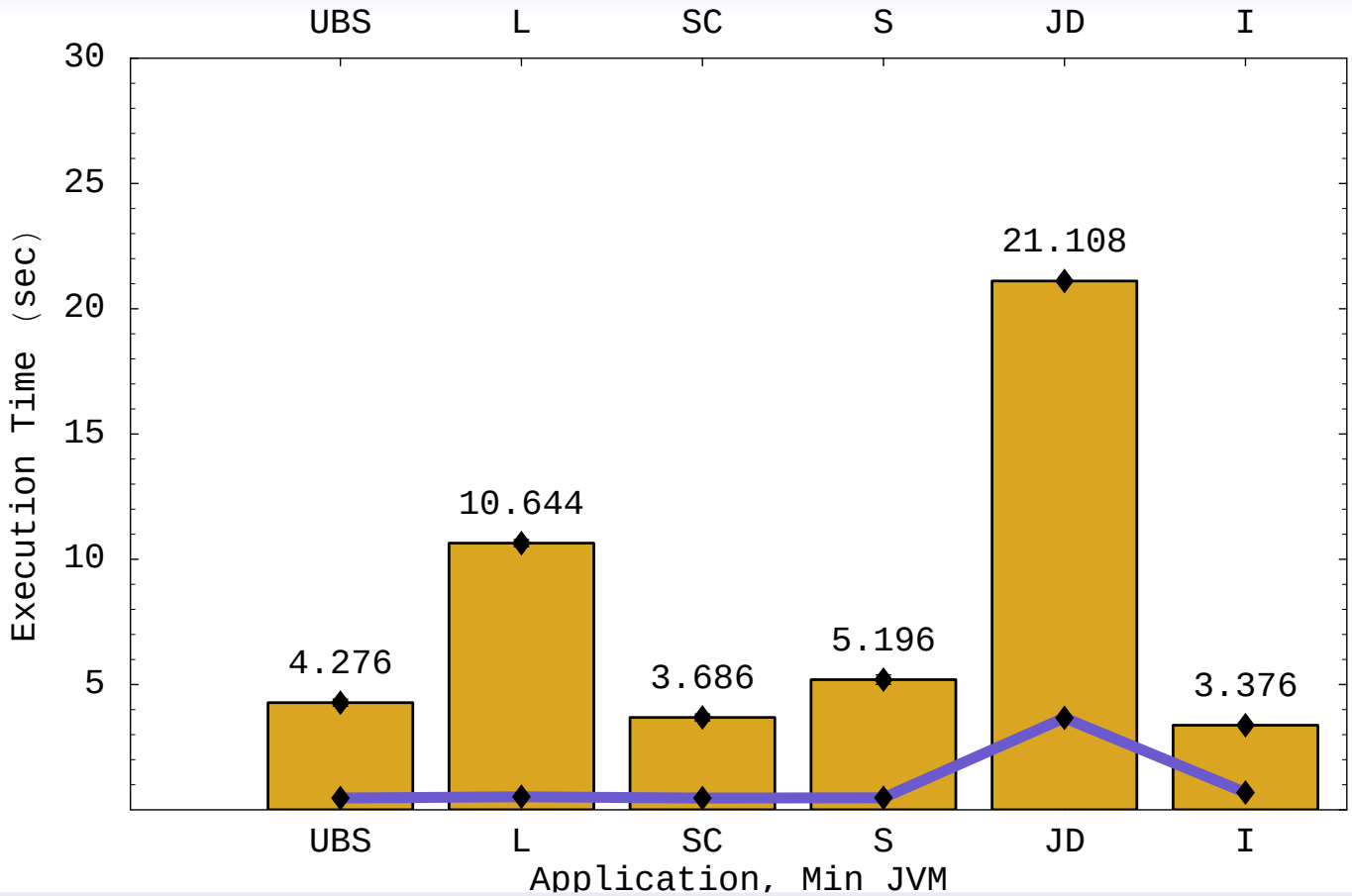
**Future Work**: Conduct experiments with larger applications

# Testing Time Overhead: *Full* RVM




When memory is **not constrained**, testing time is **acceptable**

# Testing Time Overhead: *Min* RVM




Testing time **increases** significantly when memory is *Min*

# Summary of Reductions for Library

| Technique | $\mathcal{T}_R^{\%}(P,T)$ | $\mathcal{S}_R^{\%}(P,T)$ |
|---|---|---|
| *S-GC* | 32.7 | 78.8 ✓ |
| *X-GC* | 32.1 | 65.0 |
| *S-TM* | 32.0 | 72.8 |
| *X-TM* | 31.5 | 62.3 |
| *S-CS* | 34.3 ✓ | 61.4 |
| *X-CS* | 33.4 | 59.8 |

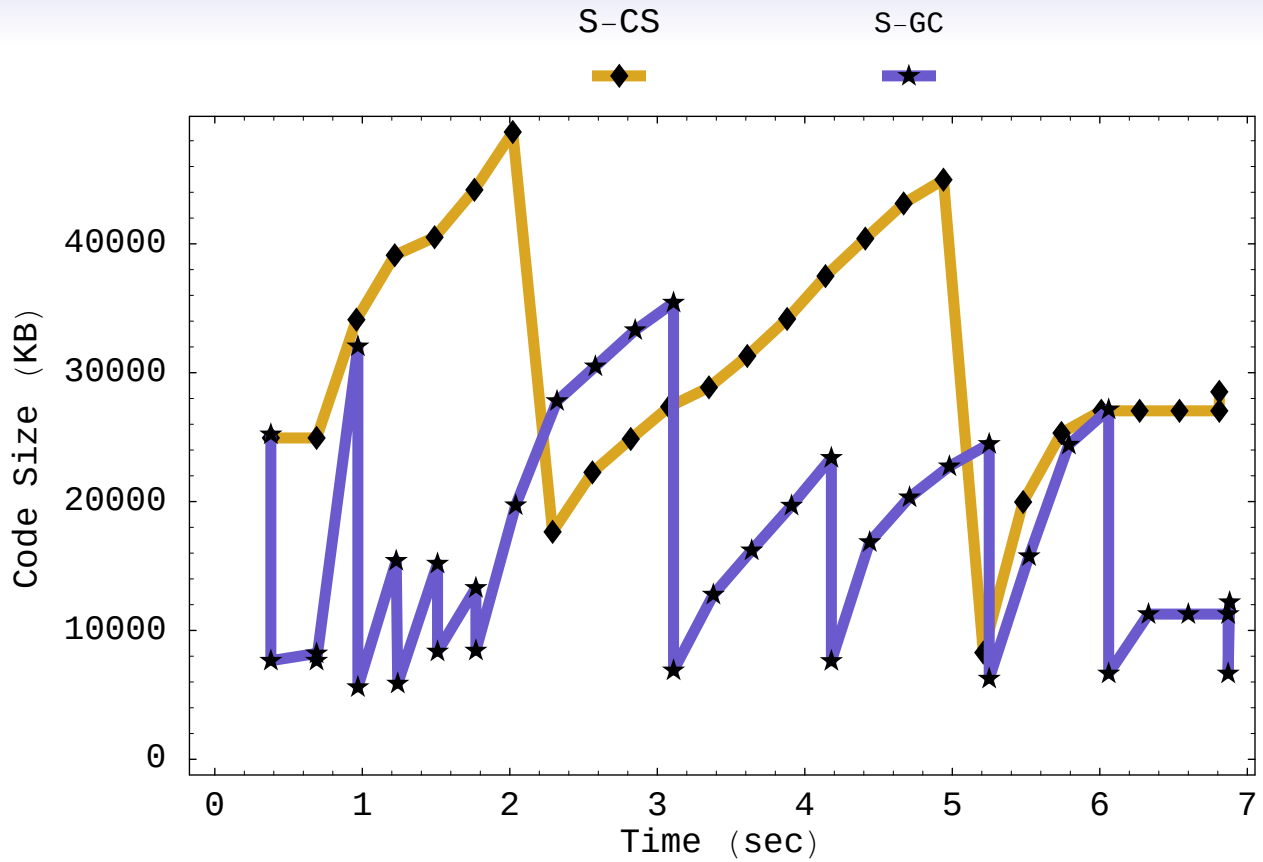
Significant **reductions** in **time** and **space** required for testing

# Summary of Reductions for Library

| Technique | $\mathcal{T}_R^{\%}(P,T)$ | $\mathcal{S}_R^{\%}(P,T)$ |
|---|---|---|
| *S-GC* | 32.7 | 78.8 ✓ |
| *X-GC* | 32.1 | 65.0 |
| *S-TM* | 32.0 | 72.8 |
| *X-TM* | 31.5 | 62.3 |
| *S-CS* | 34.3 ✓ | 61.4 |
| *X-CS* | 33.4 | 59.8 |

Significant **reductions** in **time** and **space** required for testing

# Code Size Fluctuations for Library




*S-GC* causes code size **fluctuations** that **increase** testing time

# Summary of Reductions for Identifier

| Technique | $\mathcal{T}_R^{\%}(P,T)$ | $\mathcal{S}_R^{\%}(P,T)$ |
|---|---|---|
| *S-GC* | -1.1 | 42.5 |
| *X-GC* | -1.1 | 26.7 |
| *S-TM* | -1.2 | 44.5 |
| *X-TM* | -.29 ✓ | 28.8 |
| *S-CS* | -.77 | 51.4 |
| *X-CS* | -1.4 | 61.4 ✓ |

A **decrease** in native code **size** leads to an **increase** in test execution **time**! *Why?* Identifier has a large working set.

# Summary of Reductions for Identifier

| Technique | $\mathcal{T}_R^{\%}(P,T)$ | $\mathcal{S}_R^{\%}(P,T)$ |
|---|---|---|
| *S-GC* | -1.1 | 42.5 |
| *X-GC* | -1.1 | 26.7 |
| *S-TM* | -1.2 | 44.5 |
| *X-TM* | -.29 ✓ | 28.8 |
| *S-CS* | -.77 | 51.4 |
| *X-CS* | -1.4 | 61.4 ✓ |

A **decrease** in native code **size** leads to an **increase** in test execution **time**! *Why?* Identifier has a large working set.

# Improvements to Automated Testing

| Technique | $\mathcal{T}_R^{\%}(P,T)$ | $\mathcal{S}_R^{\%}(P,T)$ |
|---|---|---|
| *S-GC* | 16.1 | 68.4 ✓ |
| *X-GC* | 16.4 | 52.8 |
| *S-TM* | 17.1 | 62.6 |
| *X-TM* | 16.4 | 45.9 |
| *S-CS* | 17.6 ✓ | 58.8 |
| *X-CS* | 15.3 | 54.8 |

Across all applications, adaptive code unloading techniques **reduce** both testing **time** and **space** overhead

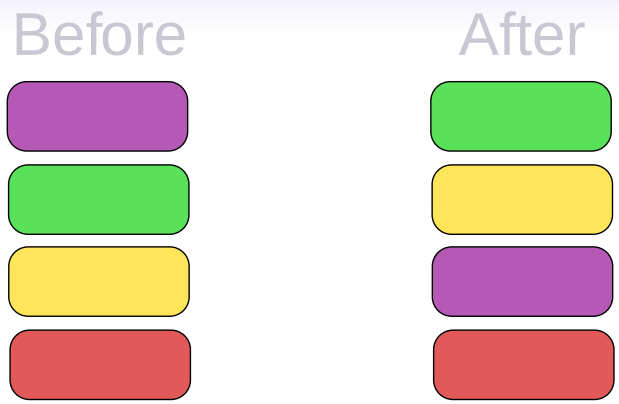
# Improvements to Automated Testing

| Technique | $\mathcal{T}_R^{\%}(P,T)$ | $\mathcal{S}_R^{\%}(P,T)$ |
|---|---|---|
| *S-GC* | 16.1 | 68.4 ✓ |
| *X-GC* | 16.4 | 52.8 |
| *S-TM* | 17.1 | 62.6 |
| *X-TM* | 16.4 | 45.9 |
| *S-CS* | 17.6 ✓ | 58.8 |
| *X-CS* | 15.3 | 54.8 |

Across all applications, adaptive code unloading techniques **reduce** both testing **time** and **space** overhead

# Future Work: Reduction and Prioritization


Reduction Prunes the Test Suite




Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Reduction** discards some of the $n$ tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection**.

# Future Work: Reduction and Prioritization



Reduction Prunes the Test Suite

Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Reduction** discards some of the $n$ tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection**.

# Future Work: Reduction and Prioritization



Reduction Prunes the Test Suite

Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Reduction** discards some of the $n$ tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection**.

# Future Work: Reduction and Prioritization


Reduction Prunes the Test Suite


Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Reduction** discards some of the $n$ tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection**.

# Future Work: Reduction and Prioritization

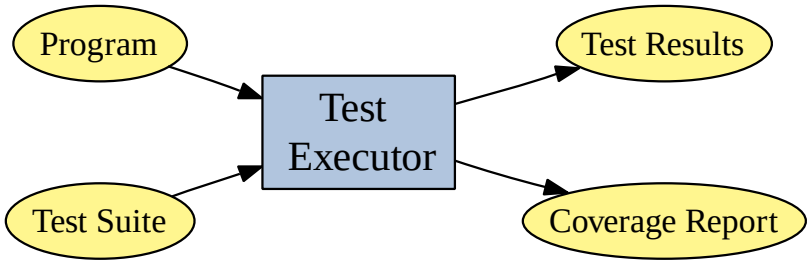


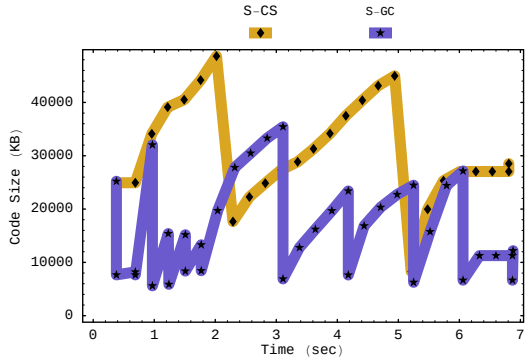
Reduction Prunes the Test Suite

Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Prioritization** searches through the $n! = n \times n - 1 \times \ldots \times 1$ orderings for those that **maximize** an objective function like memory **loads** and **unloads**.

# Concluding Remarks



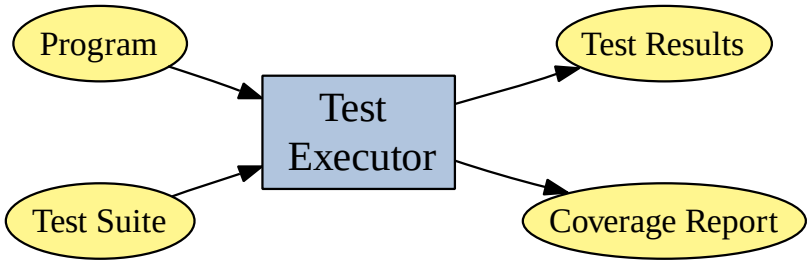

Automated Test Execution

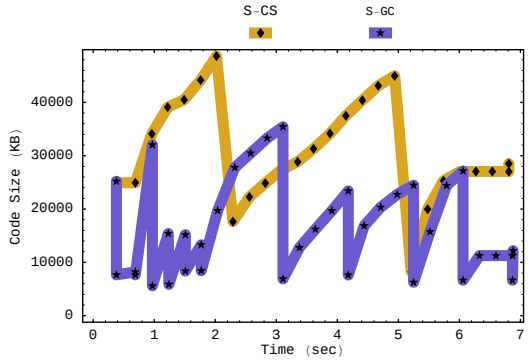
Detailed Empirical Results

- **Implementation** and empirical **evaluation** of methods for testing in memory constrained environments
- Aim to **apply** these methods to T-Mobile G1 with Google Android

**http://www.cs.allegheny.edu/~gkapfham/research/juggernaut/**

# Concluding Remarks



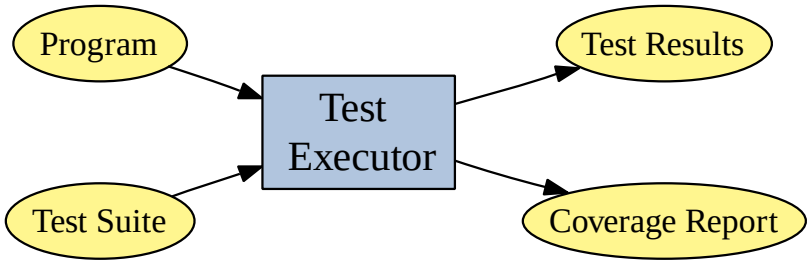

Automated Test Execution

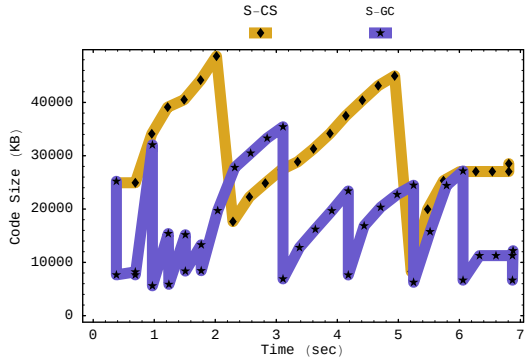
Detailed Empirical Results

- **Implementation** and empirical **evaluation** of methods for testing in memory constrained environments
- Aim to **apply** these methods to T-Mobile G1 with Google Android

**http://www.cs.allegheny.edu/~gkapfham/research/juggernaut/**

# Concluding Remarks



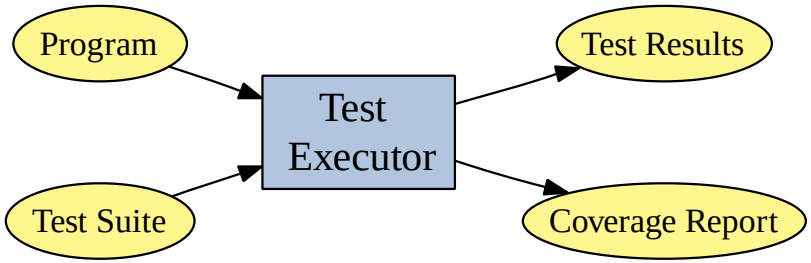

Automated Test Execution

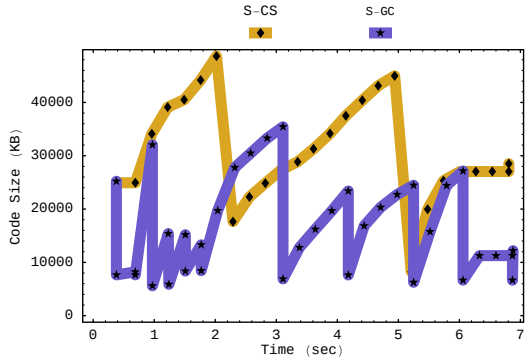
Detailed Empirical Results

- **Implementation** and empirical **evaluation** of methods for testing in memory constrained environments
- Aim to **apply** these methods to T-Mobile G1 with Google Android

**http://www.cs.allegheny.edu/~gkapfham/research/juggernaut/**

# Concluding Remarks




Automated Test Execution

Detailed Empirical Results

- **Implementation** and empirical **evaluation** of methods for testing in memory constrained environments
- Aim to **apply** these methods to T-Mobile G1 with Google Android

**http://www.cs.allegheny.edu/~gkapfham/research/juggernaut/**