



# ***Exploring Time-Aware Test Suite Prioritization***

---

**Mary Lou Sofa**  
**University of Virginia**

**Collaborators:**

**Kristen Walcott**

**Gregory M. Kapfhammer,**  
**Allegheny College**



# *Regression testing*

---

## ***Software constantly modified***

- ***Bug fixes***
- ***Addition of functionality***

## ***After changes, regression testing – run test case in test suite and provide more***

- ***Provides confidence modifications correct***
- ***Helps find new error***

## ***Large number of test cases – continues to grow***

- ***Weeks/months to run entire test suite***
- ***Costs high – 1/2 cost of maintenance***



# *Reducing cost regression testing*

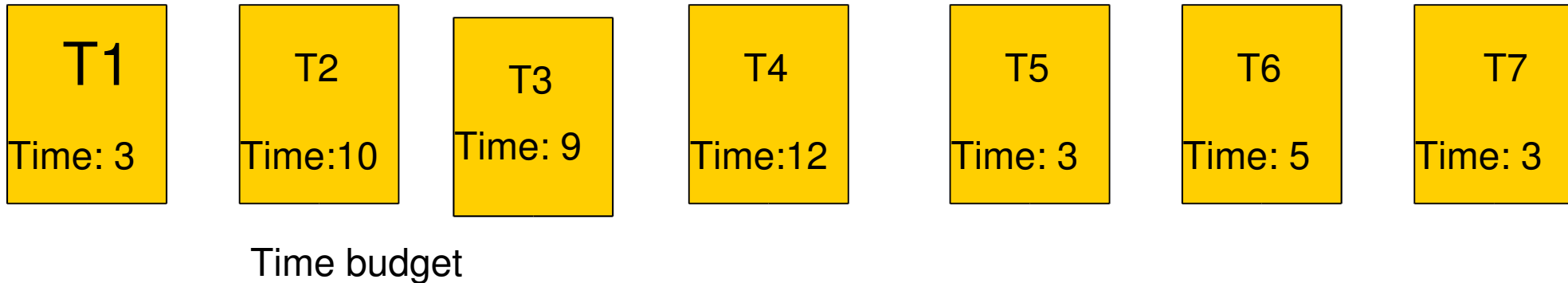
---

- *To reduce cost, do not run all test cases – prioritize tests i.e., reorder them*
- *Test Prioritization Techniques*
  - *Original order*
  - *Based on fault detection ability*
  - *Analysis to determine what test cases affected by change and order*
  - *Random selection – order tests randomly*
  - *Reverse – run tests in reverse order*



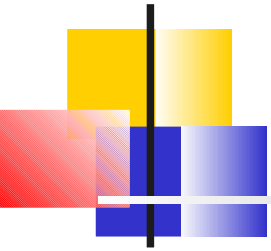
## *Example – after prioritization*

---



But, retesting usually has a time budget –  
based on time, was the above order the best order?

**Contribution: A test prioritization technique that intelligently incorporates the test time budget**



# Fault Matrix Example

Given modified program, have 6 test cases

Assume a priori knowledge of faults,  $f$

<b>FAULTS/ EST CASE</b>	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$	$f_8$
$\mathcal{T}_1$	$\mathcal{X}$		$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
$\mathcal{T}_2$	$\mathcal{X}$							
$\mathcal{T}_3$	$\mathcal{X}$				$\mathcal{X}$			
$\mathcal{T}_4$		$\mathcal{X}$	$\mathcal{X}$				$\mathcal{X}$	
$\mathcal{T}_5$				$\mathcal{X}$		$\mathcal{X}$		
$\mathcal{T}_6$		$\mathcal{X}$		$\mathcal{X}$		$\mathcal{X}$		



# Test Suite

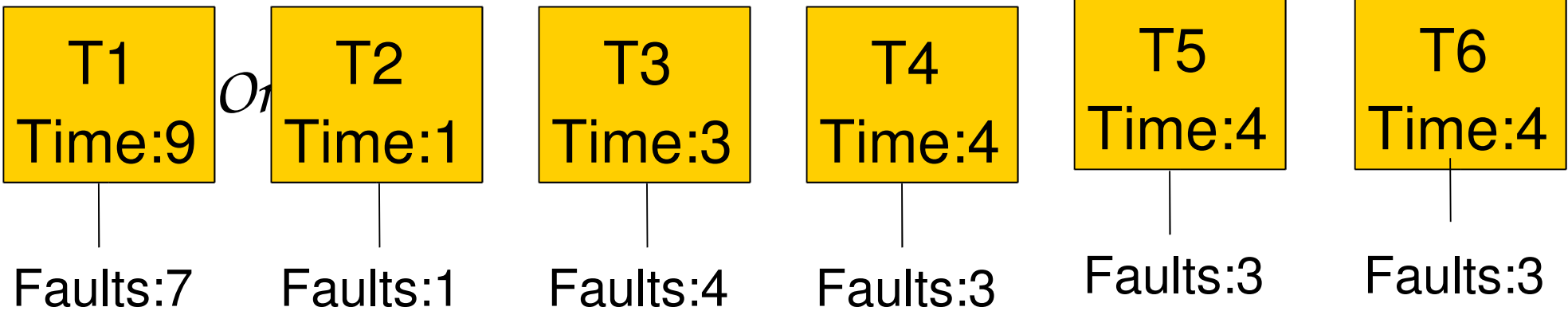
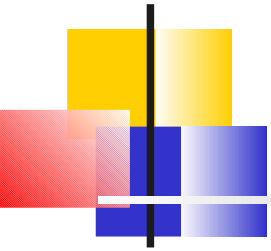
## Faults and Time

	<i>#faults</i>	<i>Time costs</i>	<i>avg faults/min</i>
$\mathcal{T}1$	7	9	0.778
$\mathcal{T}2$	1	1	1.0
$\mathcal{T}3$	2	3	0.667
$\mathcal{T}4$	3	4	0.75
$\mathcal{T}5$	3	4	0.75
$\mathcal{T}6$	3	4	0.75

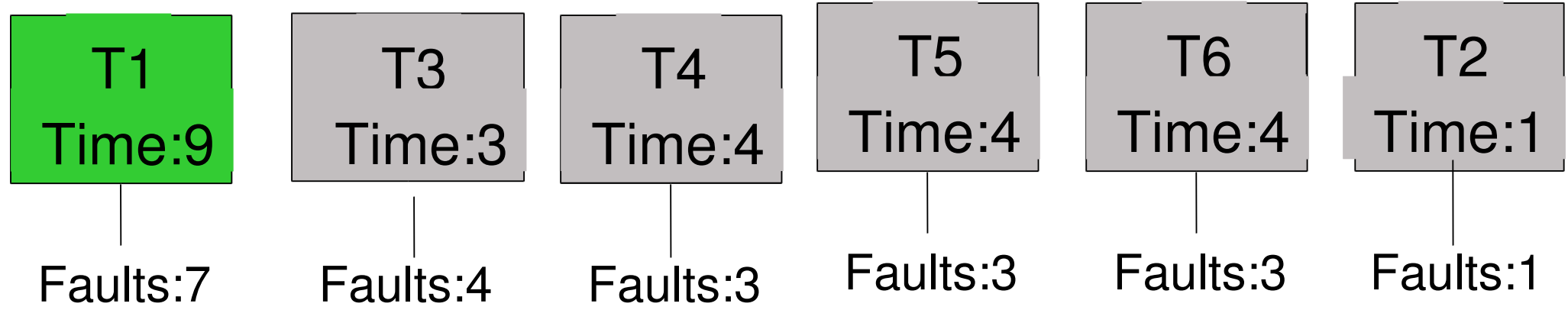
Tests vary according to the time overhead and their ability to reveal faults

**GOAL: When testing, find as many faults as soon as possible**

# Fault – aware Prioritization - Time limit 12 minutes



## Fault based order



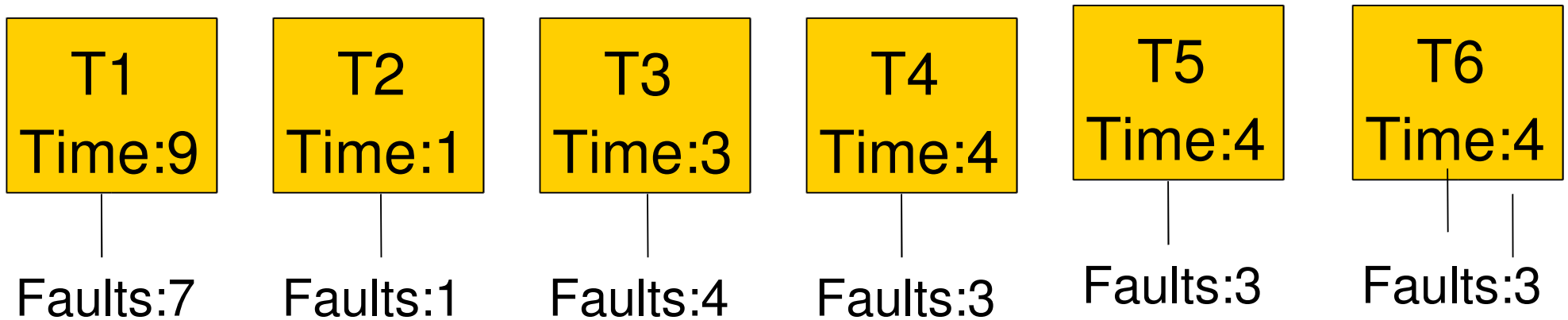
**7 faults found in 9 minutes**



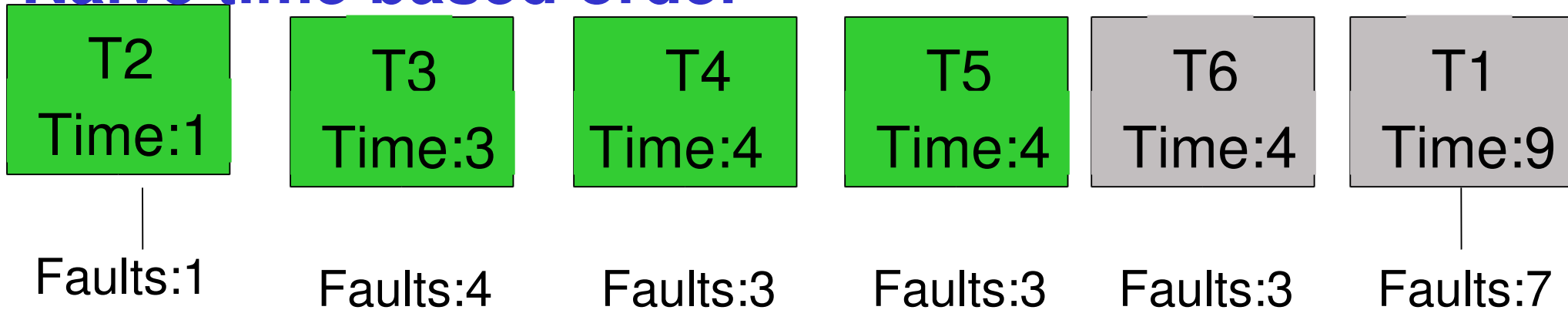
# *Naïve time-Based prioritization*

---

## ■ *Original Order*



## *Naïve time based order*

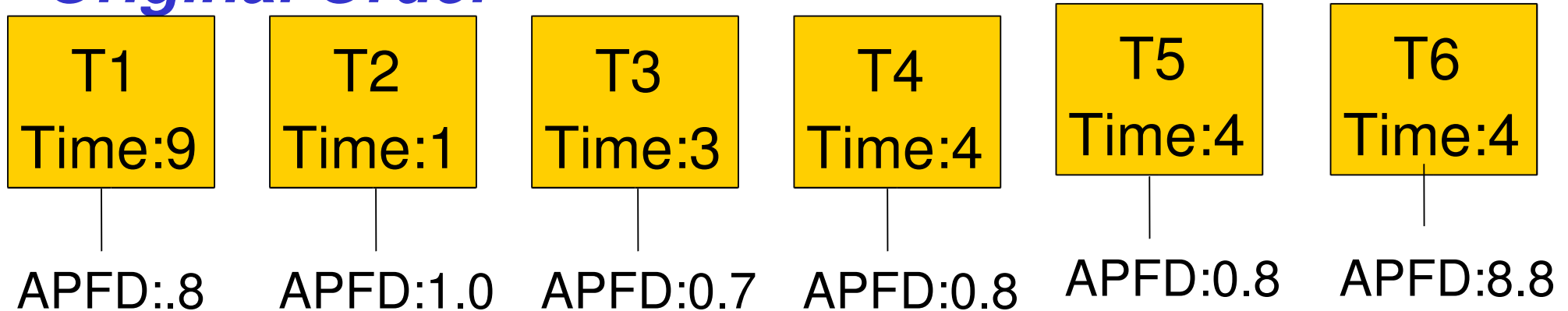


***8 faults in 12 minutes***

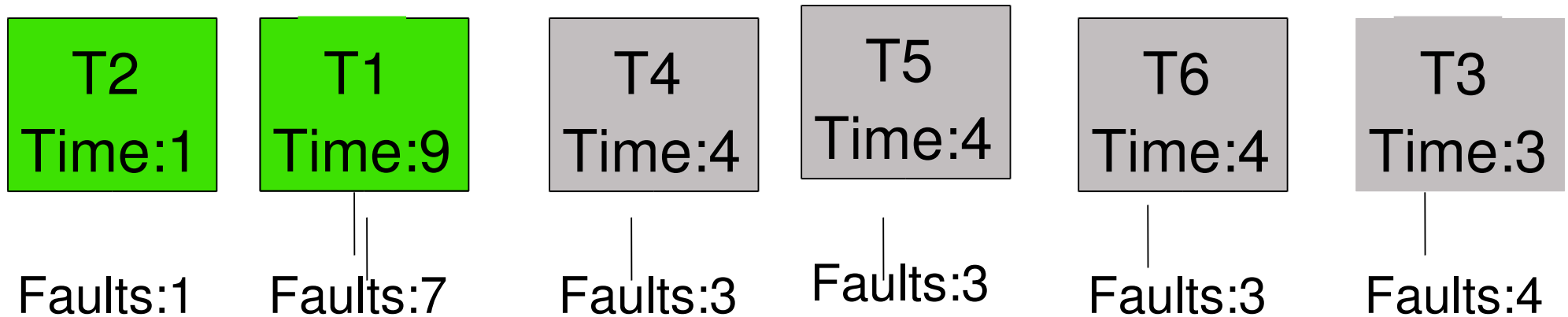


# Average Percent Fault Detection -Based Prioritization

## Original Order



## APFD



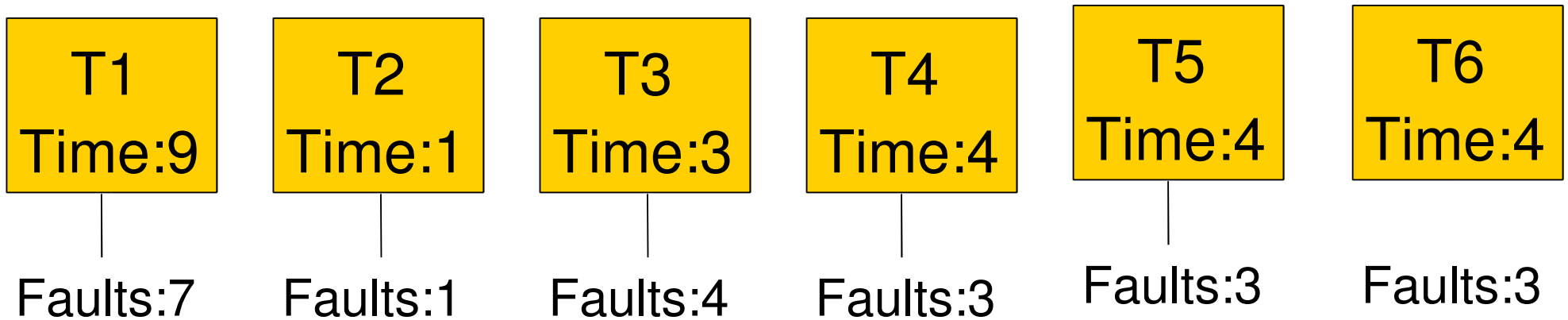
**7 faults in 10 minutes**



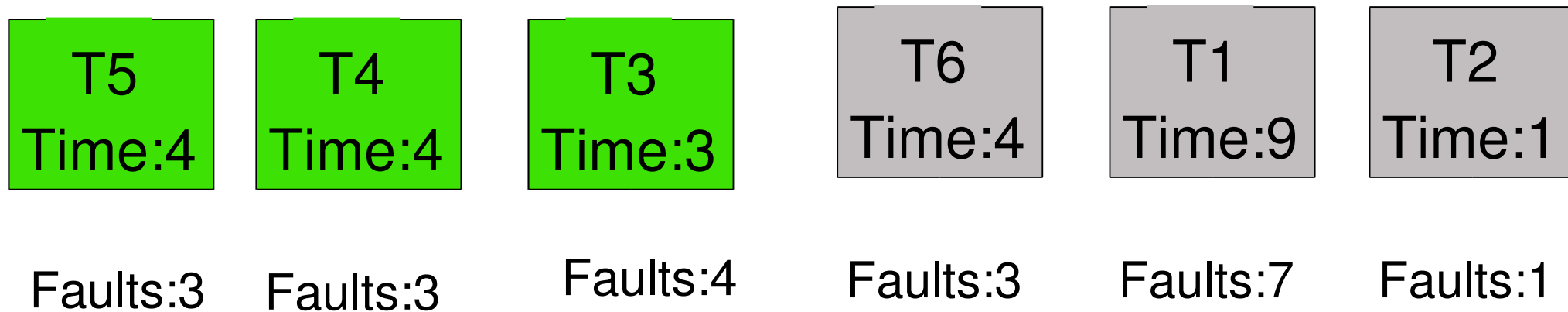
# *Intelligent Time-aware prioritization*

---

## ■ *Original order*



## • *Intelligent Time-aware prioritization*



**8 faults in 11 minutes**

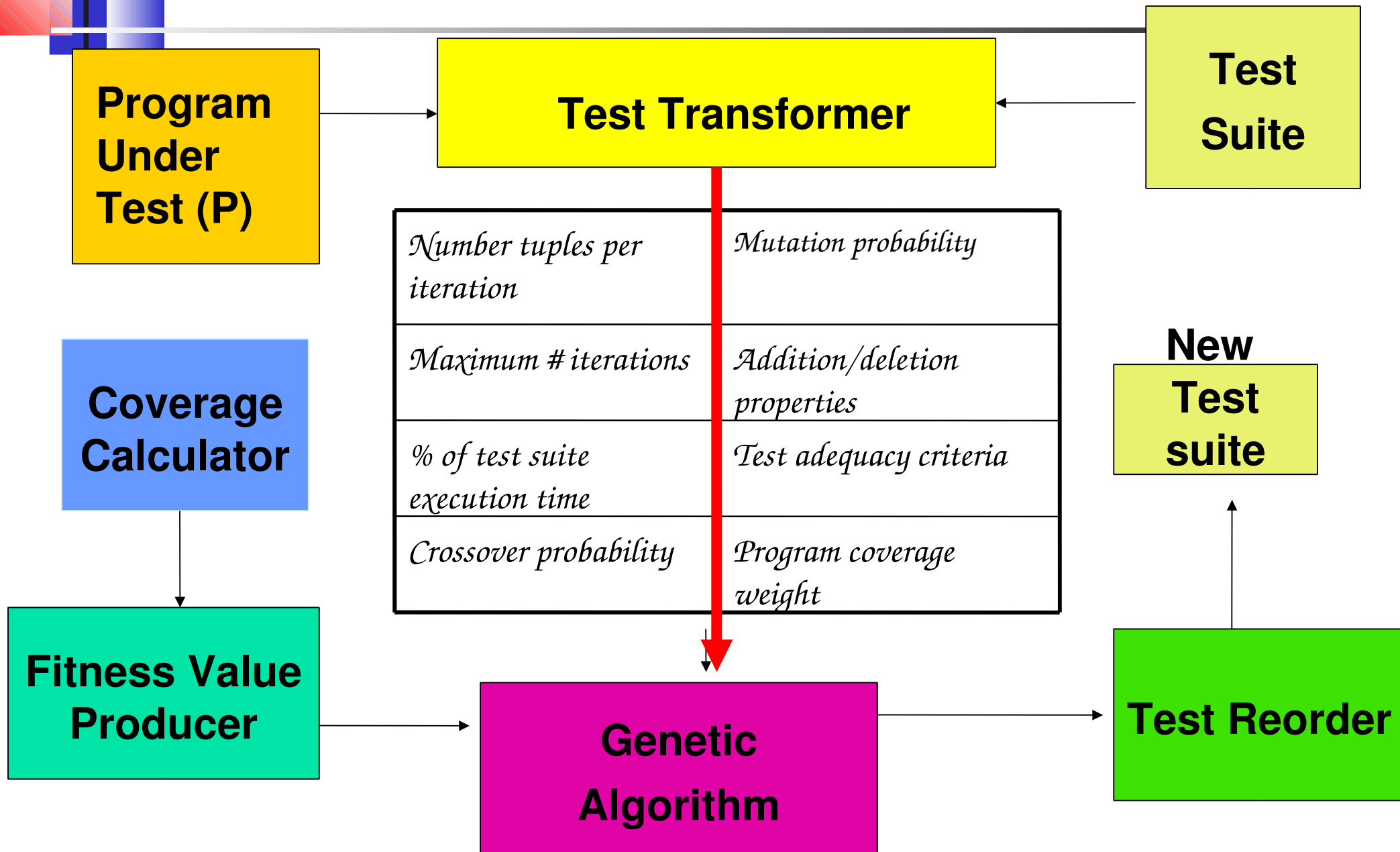


# Comparing Test Prioritization

---

- *Intelligent scheme performs better – finding most faults in shortest time*
- *Considers testing time budget and overlapping fault detection of test*
- *Time-aware prioritization requires heuristic solution to  $\mathcal{NP}$ -complete*
- *Use genetic algorithm*
  - *Fitness function based on code coverage for ability to find faults and time*

# Infrastructure





# *Fitness Function*

---

- *Since fault information unknown, use method and block coverage to measure test suite potential*
- *Coverage is aggregated for entire test suite*
- *Test prioritization fitness measures*
  - *The percentage of  $\mathcal{P}$ 's code that is covered by  $T_i$*
  - *The time at which each test case covers code within  $\mathcal{P}$  – can use percentages of code coverage*



## *Change the order of test cases*

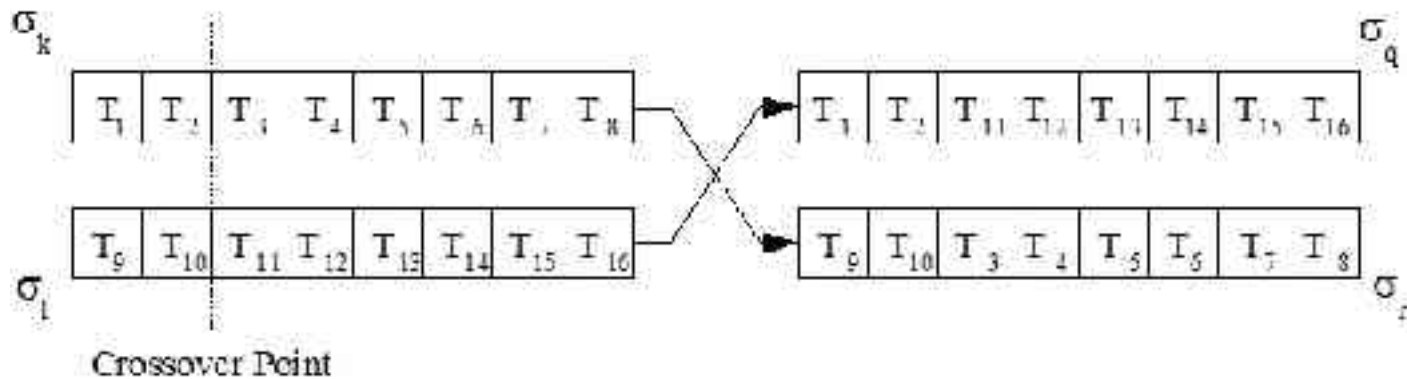
---

- *Develop smaller test suites based on operators that change*
  - *Order*
  - *Test cases included*

*Fitness evaluation determines goodness of the changed suite.*

# Crossover Operator

- *Vary test prioritizations by recombination at a randomly chosen crossover point*

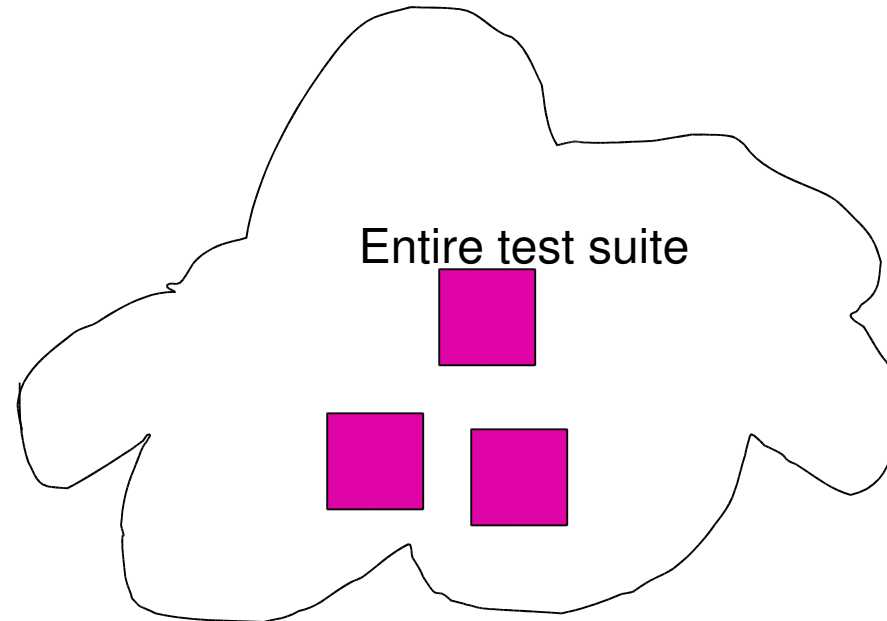
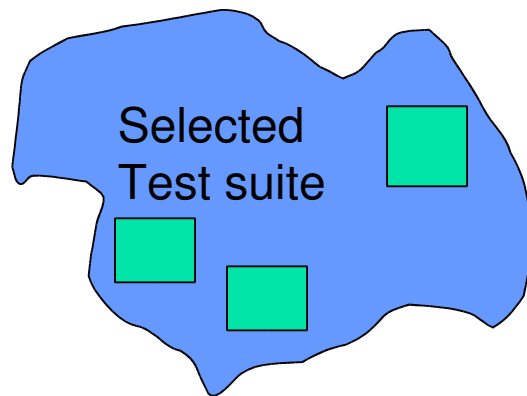




# *Addition and Deletion Operators*

---

- *Operators*



Add operator

Delete operator





# *Mutation Operators*

---

- *Another way to add variation to create new population*
- *Test cases are mutated –*
  - *replaced by an unused test case*
  - *Swap test cases if no unused test case*



# *Experiment Goals and Design*

---

- *Determine if the GA-produced prioritizations, on average, outperform a selected set of other prioritizations*
- *Identify overhead - time and space - associated with the creation of the prioritized test suite*



# *Experiments*

---

- *Block or method coverage*
- *Order*
  - *Initial order*
  - *Reverse order*
  - *Random order*
  - *Fault-aware prioritization*



# Experimental Design

---

- *GNU/Linux Workstation – 1.80 GHz Intel Pentium and 1GB of main memory*
- *Used JUnit to prioritize test cases*
- *Seeded faults: 25%, 50%, 75% of 40 faults*
- *Used Emma to compute coverage criteria*
- *2 Case studies*
  - **Gradebook**
  - **JDepend – traverse directories of Java class files**

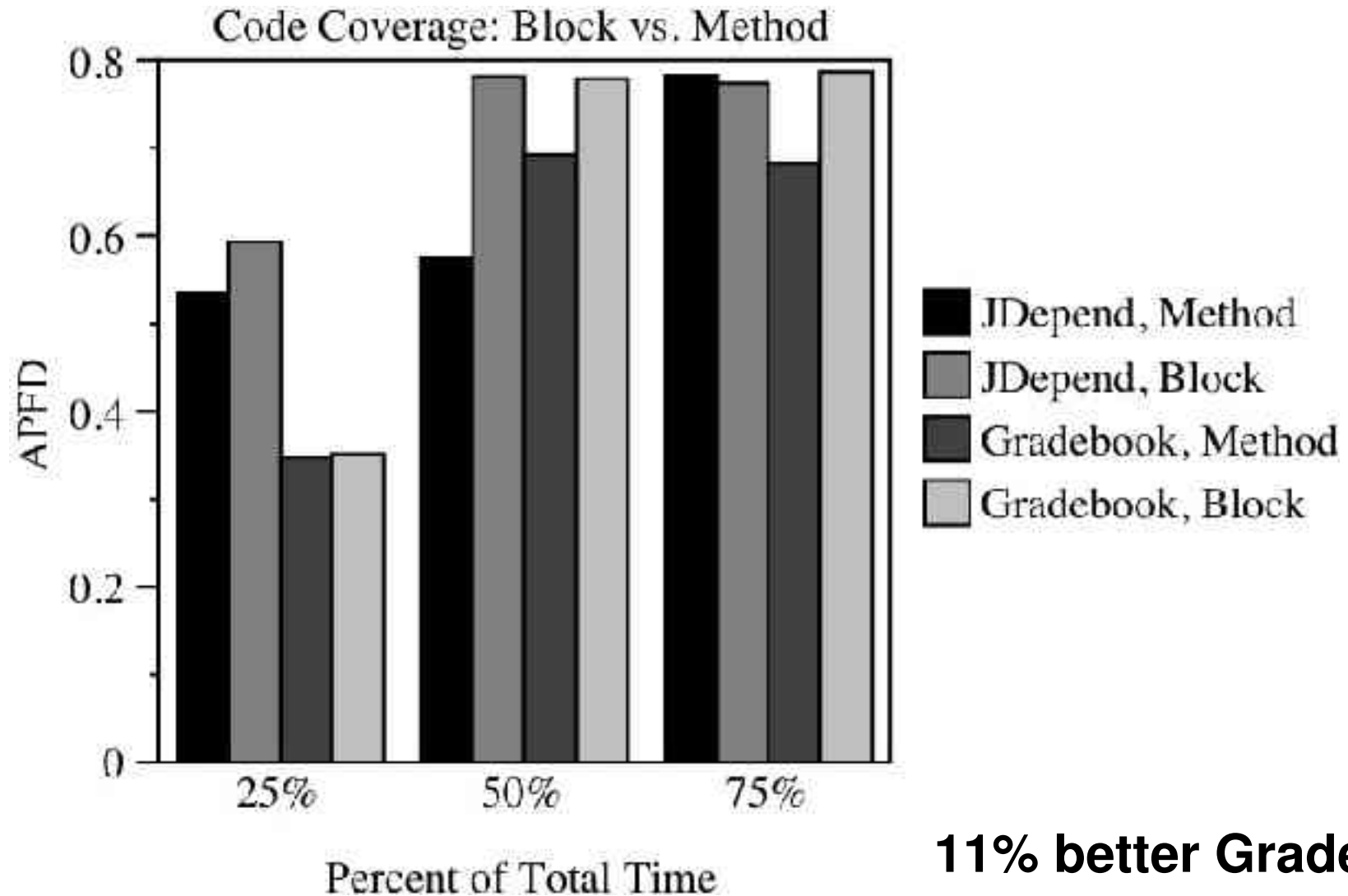


# *Test Adequacy Metrics*

---

- *Method coverage*
  - *Considered covered when entered*
- *Basic block coverage*
  - *A sequence of byte code instructions without any jumps or jump targets*
  - *Considered covered when entered*
- *How much of the code has been executed – used 100%*

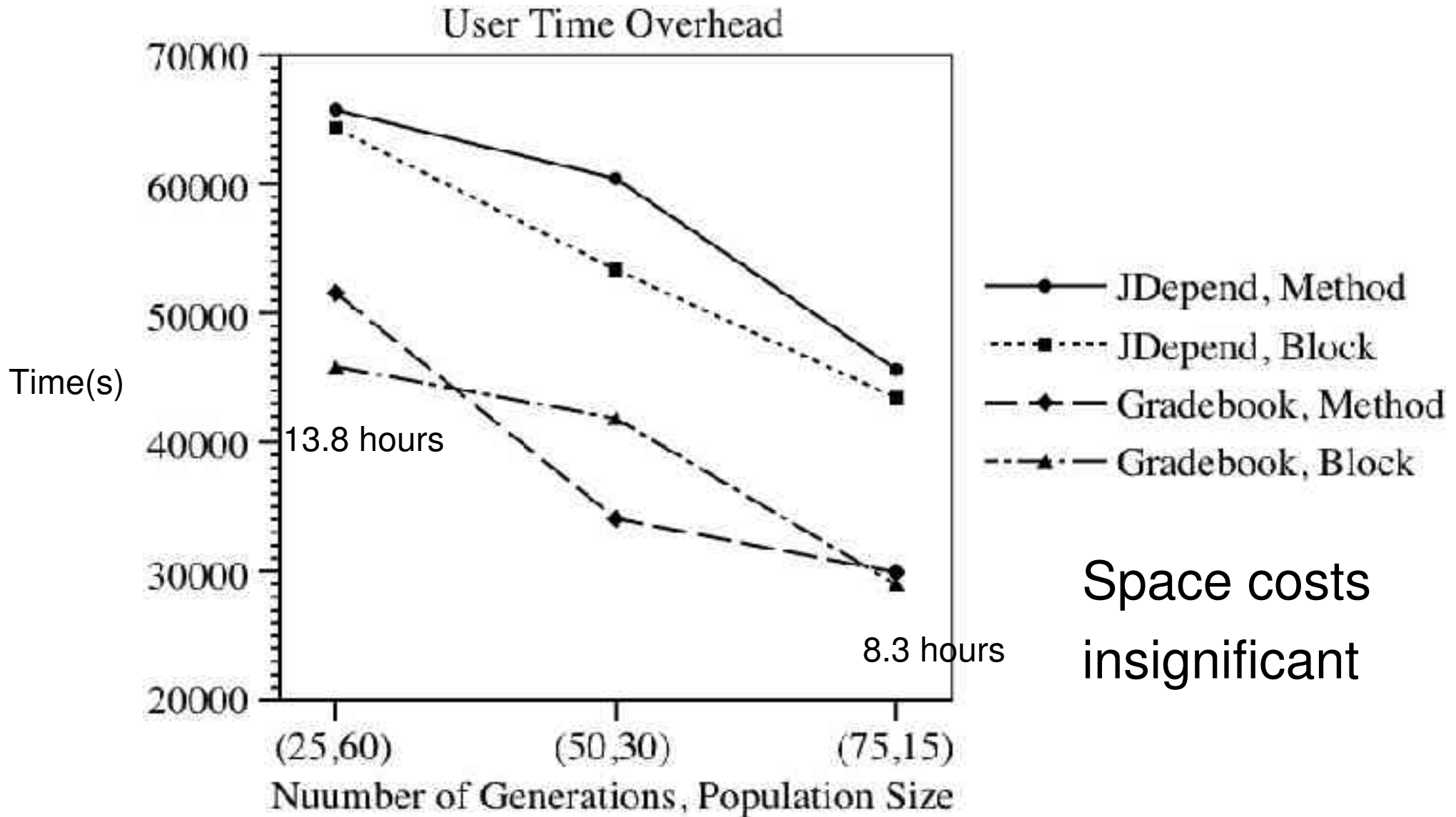
# APFD Results for Block and Method Coverage

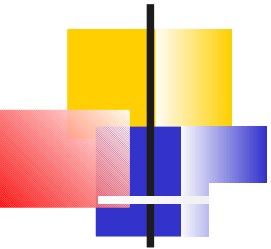


**11% better Gradebook**

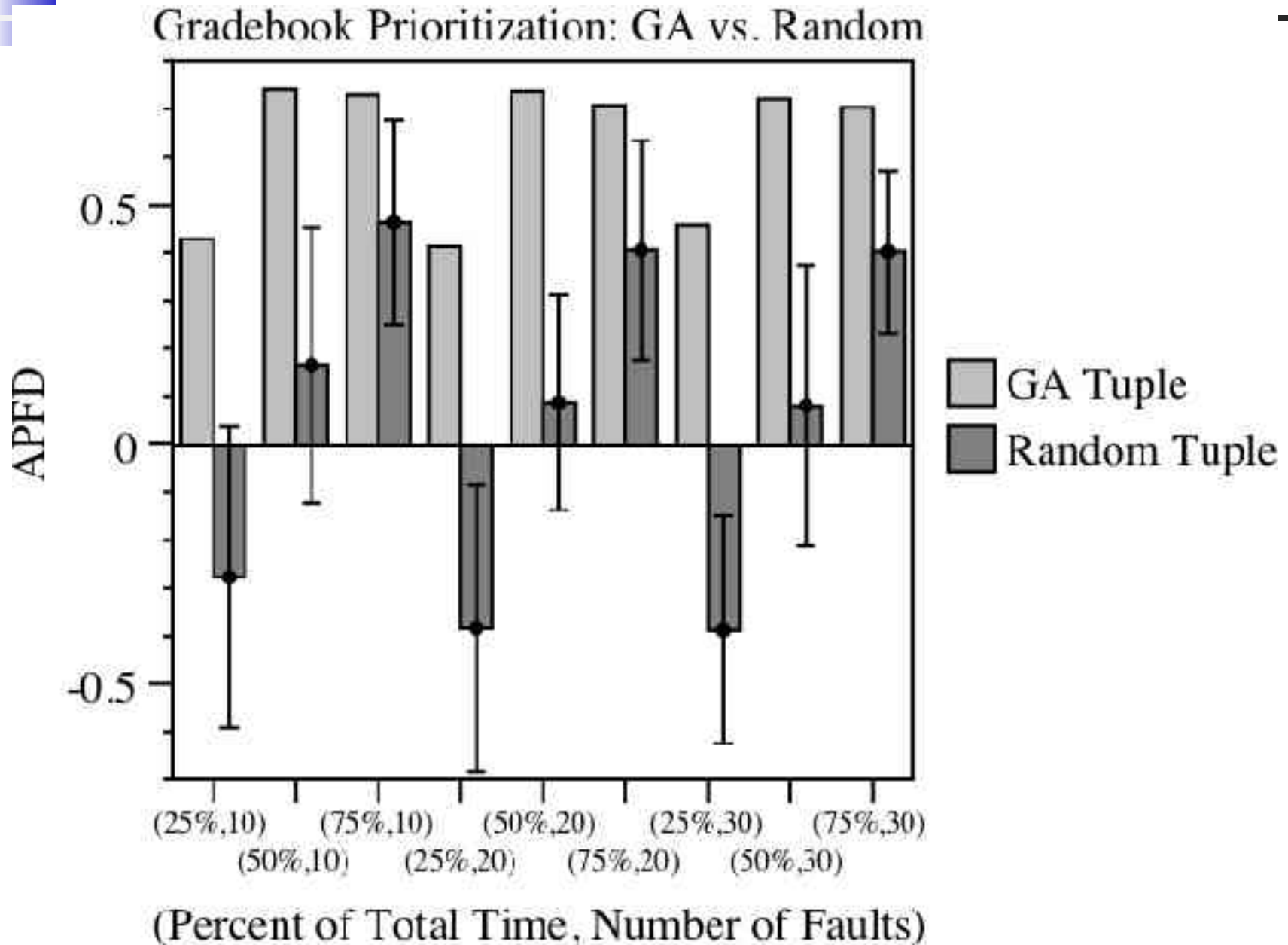
**13% better JDepend**

# Prioritization Efficiency



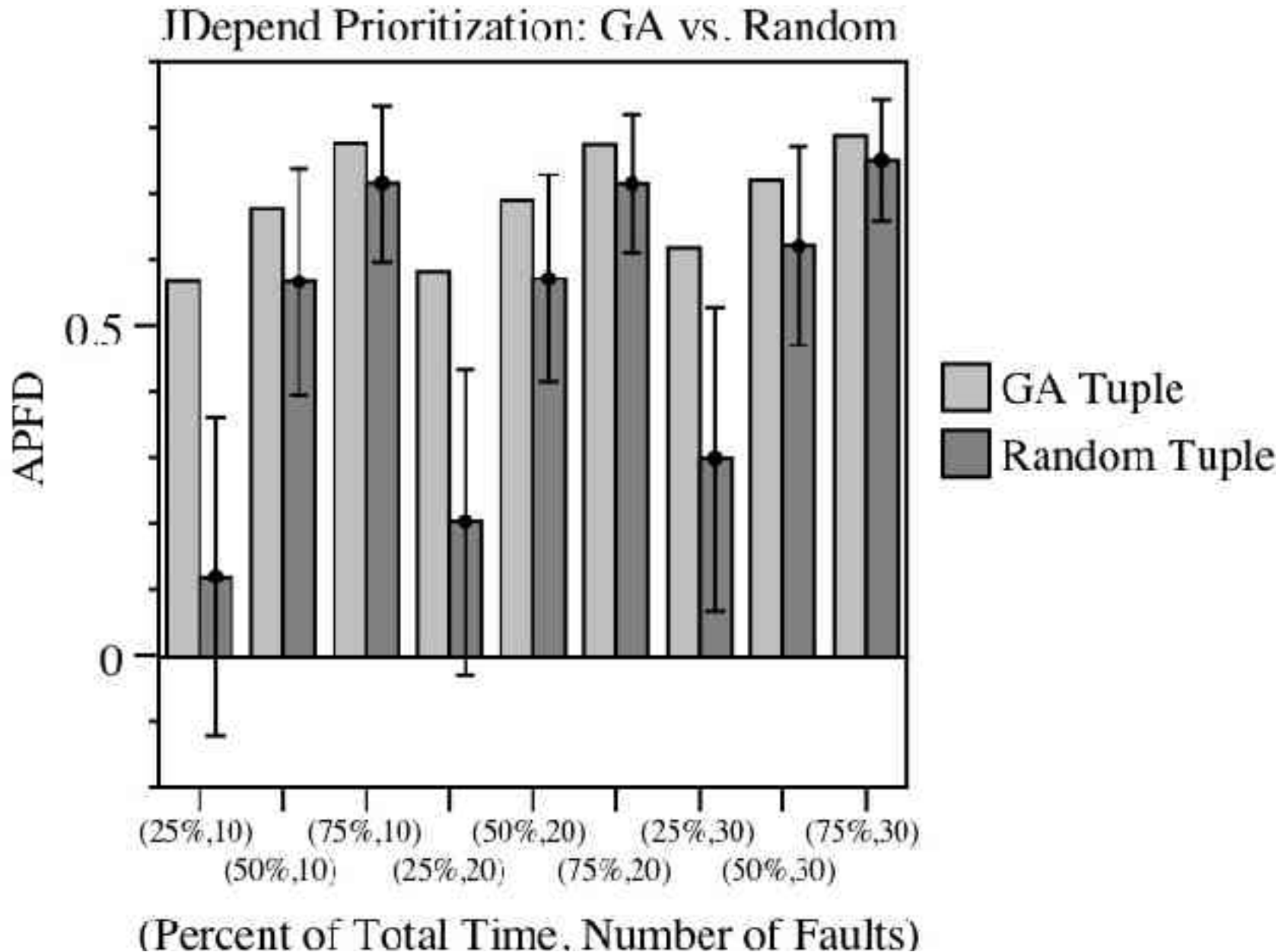


# Gradebook: *Intelligent vs Random*





# *IDepend: Intelligent vs Random*





## *Comparisons with other orders*

---

- *Experiments to compare with other types of prioritizations*
- *Original*
- *Reverse*
- *Fault aware (impossible to implement)*
- *Time aware*



## *APFD Metric*

---

*Let  $T$  be the test suite under evaluation,  $g$  the number of faults contained in the program under test  $P$ ,  $n$  the total number of test cases, and  $reveal(i, T)$  the position of the first test in  $T$  that exposes fault  $i$ .*

$$APFD(T, P) = 1 - \frac{\sum_{i=1}^g reveal(i, T)}{ng} + \frac{1}{2n}$$



## *Gradebook: Alternative Prioritizations*

$P_i$	$F_i$	<i>Initial</i>	<i>Reverse</i>	<i>Fault aware</i>	$GA$
<i>0.25</i>	<i>10</i>	<i>-0.6</i>	<i>-0.2</i>	<i>0.7</i>	<i>0.4</i>
<i>0.25</i>	<i>20</i>	<i>-0.9</i>	<i>-0.2</i>	<i>0.7</i>	<i>0.4</i>
<i>0.25</i>	<i>30</i>	<i>-0.9</i>	<i>-0.0</i>	<i>0.5</i>	<i>0.6</i>
<i>0.50</i>	<i>10</i>	<i>-0.04</i>	<i>0.1</i>	<i>0.9</i>	<i>0.7</i>
<i>0.50</i>	<i>20</i>	<i>-0.2</i>	<i>0.2</i>	<i>0.9</i>	<i>0.7</i>
<i>0.50</i>	<i>30</i>	<i>-0.3</i>	<i>0.3</i>	<i>0.8</i>	<i>0.7</i>
<i>0.75</i>	<i>10</i>	<i>0.3</i>	<i>0.5</i>	<i>0.9</i>	<i>0.9</i>
<i>0.75</i>	<i>20</i>	<i>0.1</i>	<i>0.4</i>	<i>0.9</i>	<i>0.7</i>
<i>0.75</i>	<i>30</i>	<i>0.04</i>	<i>0.5</i>	<i>0.9</i>	<i>0.7</i>



# Results

---

- *Comparison of*
  - *Original*
  - *Fault-aware (impossible to implement)*
  - *Reverse*
  - *Gradebook*
    - *120% better than original*
    - *Time aware better than original*
- *JDepend*
  - *Produced better results*



# *Technique Enhancements*

---

- *Make fitness calculation faster*
  - *Eliminate the majority of coverage cover overlap by reducing the test suite*
  - *Record coverage on a per-test basis*
- *Distribute execution of fitness function*
- *Exploit test execution histories and favor tests that have recently revealed faults*
- *Terminate the genetic algorithm when it achieves fitness equivalent to previous prioritizations*



## *Conclusions and Future Work*

---

- *Contribution: a test prioritization technique that includes the testing time budget*
- *Time-aware prioritization can yield a 120% improvement in APFD when compared to alternative prioritizations*
- *Different heuristics - analysis*



*Paper to appear*

---

- *International Symposium on Software Testing and Analysis (ISSTA)*
- *July, 2006*