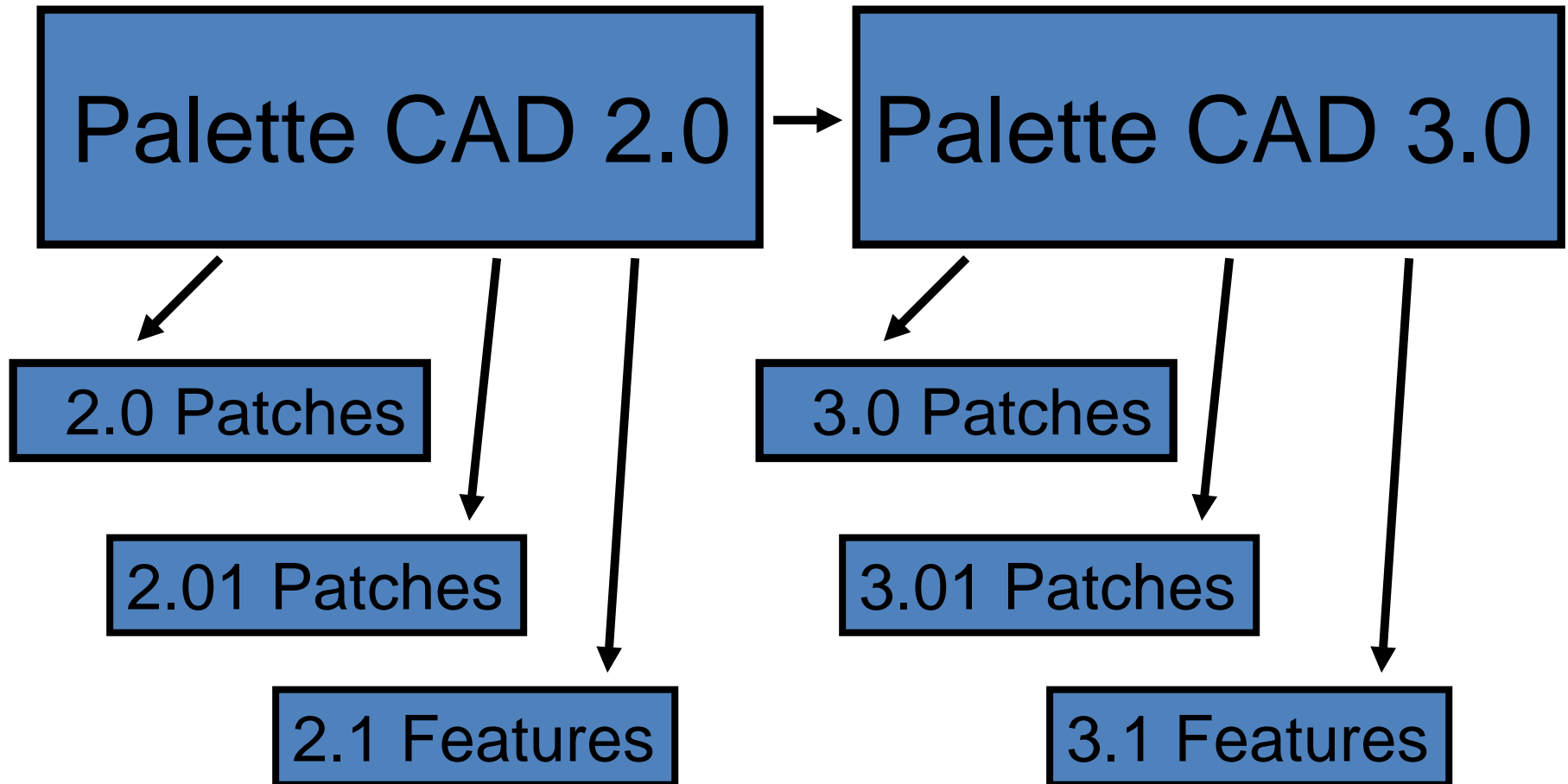


Improving Regression Testing in Continuous Integration Development Environments

Gregg Rothermel

Department of Computer Science and Engineering
University of Nebraska – Lincoln

Software Evolution



Software Evolution: A Traditional Process Model

Release P

Release P'



Preliminary period

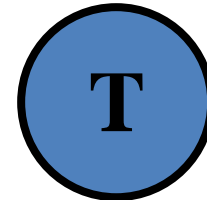
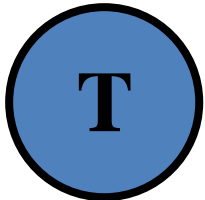
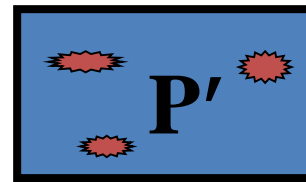


Critical period

Software Evolution with Regression Testing

Release P

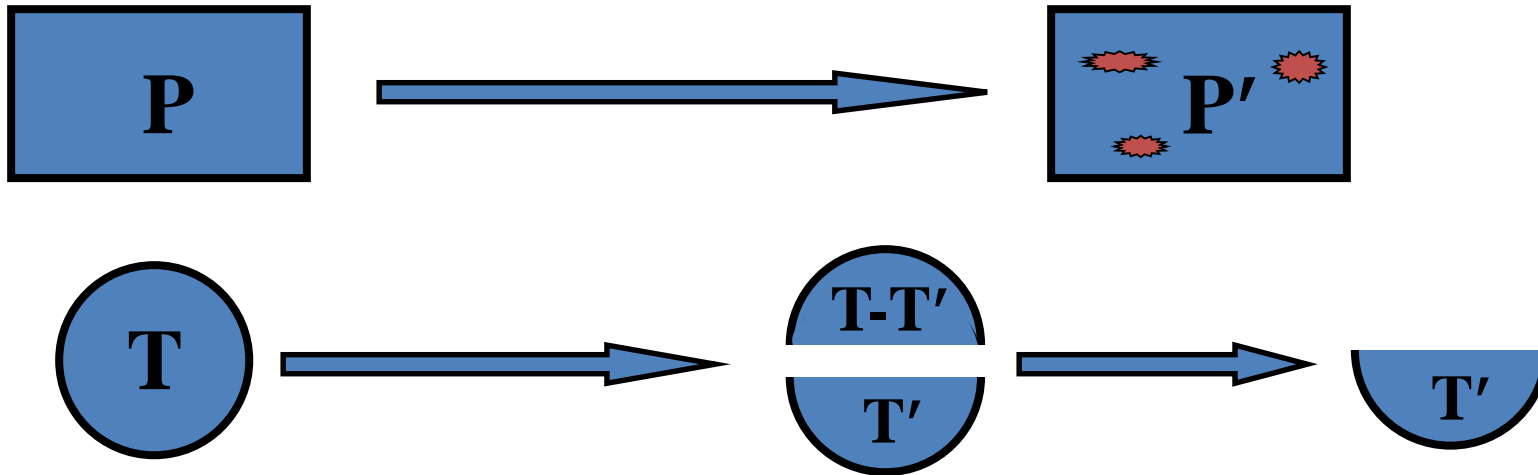
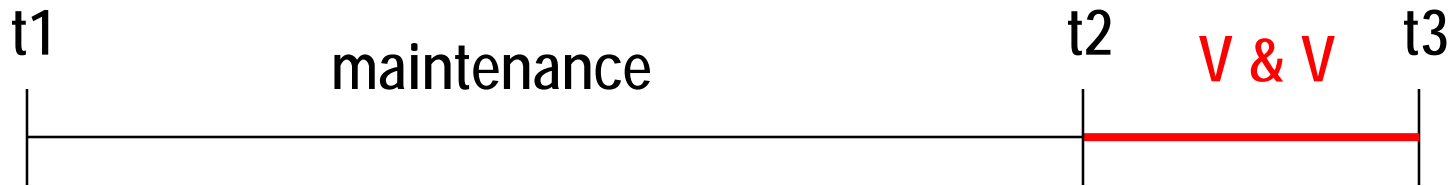
Release P'



Regression Test Selection (RTS)

Release P

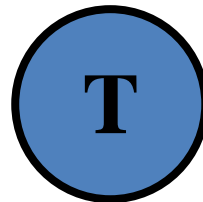
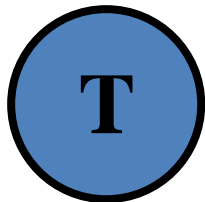
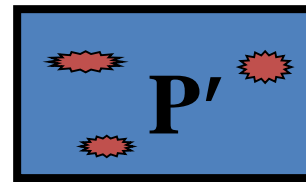
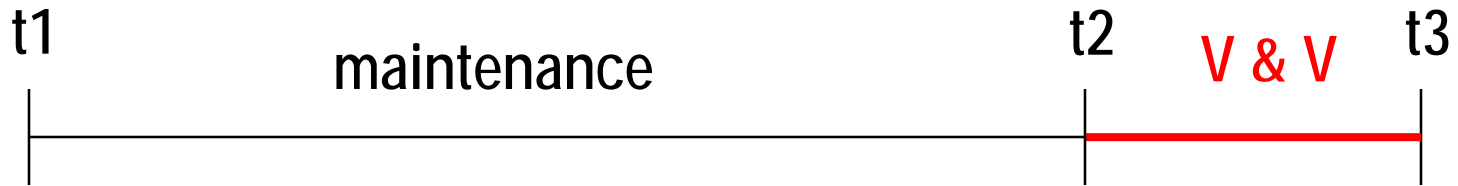
Release P'



Test Case Prioritization (TCP)

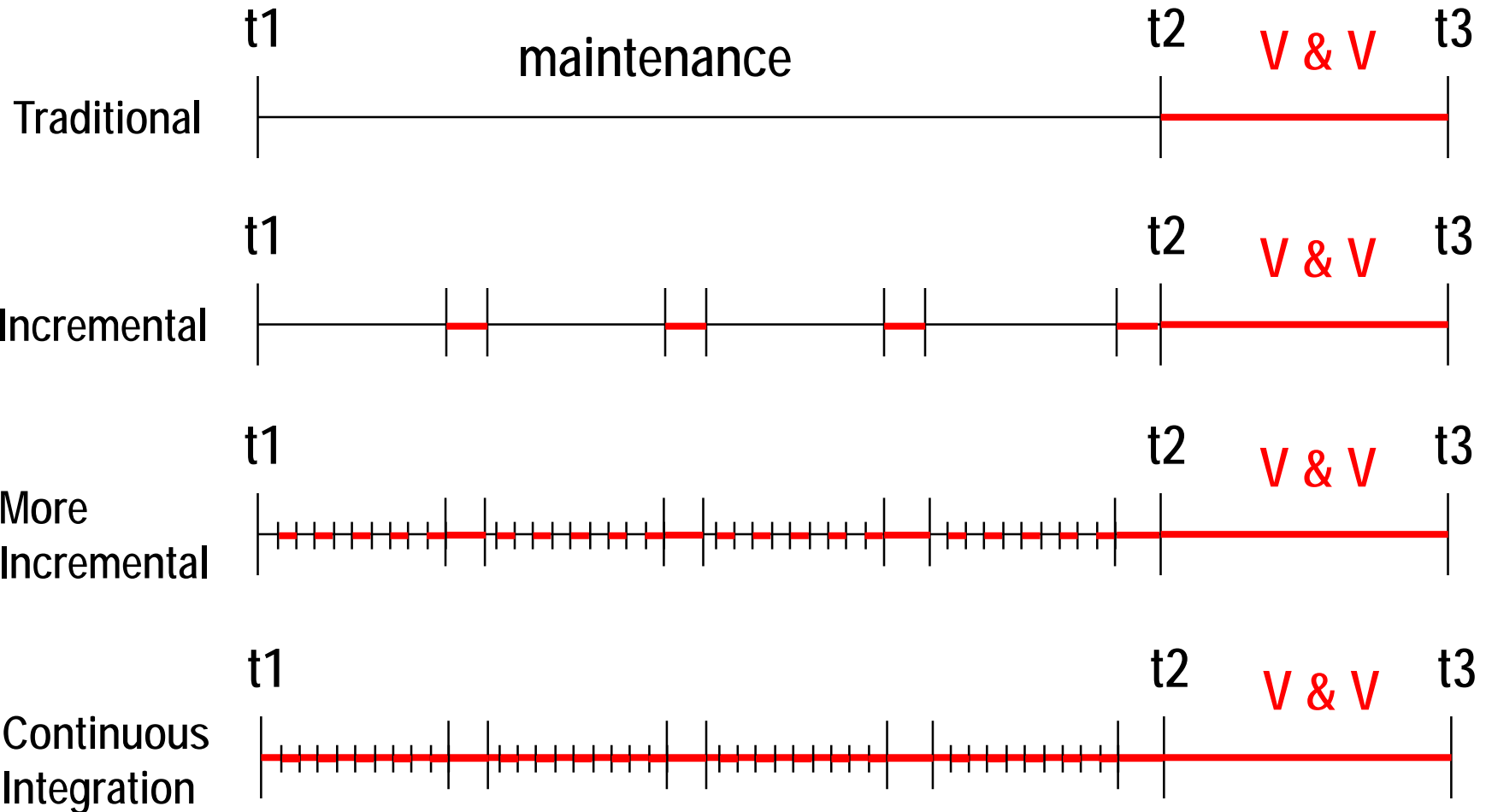
Release P

Release P'



test 1
test 2
test 3
.
.
.
test n

Software Evolution Process Models



Continuous Integration

Continuous Integration

- Engineers merge code with the code base frequently; with each merge, tests are executed on changed and affected code
- Being practiced by major software organizations including Google, Facebook, Microsoft, Amazon, Twitter, and open-source projects like Firefox
- Supported by various tools including Travis CI, Jenkins, Atlassian, ThoughtWorks, and others

Travis CI

The screenshot displays the Travis CI web interface. On the left, a sidebar titled 'My Repositories' lists several projects: 'green-egg/ham' (build #22, 30 sec), 'one-fish/two-fish' (build #2686, 33 min 46 sec), 'hop-on/pop' (build #7001, 22 min 54 sec), and 'horton-hears/awho' (build #209, 53 sec). The main area shows the 'Current' build for 'green-eggs/ham', which is 'passing'. It includes a commit message from Sven Fuchs, a list of build statistics (209 passed, commit d019f29, 53 sec), and a terminal log showing system information and build steps like 'git clone' and 'Starting PostgreSQL v9.3'.

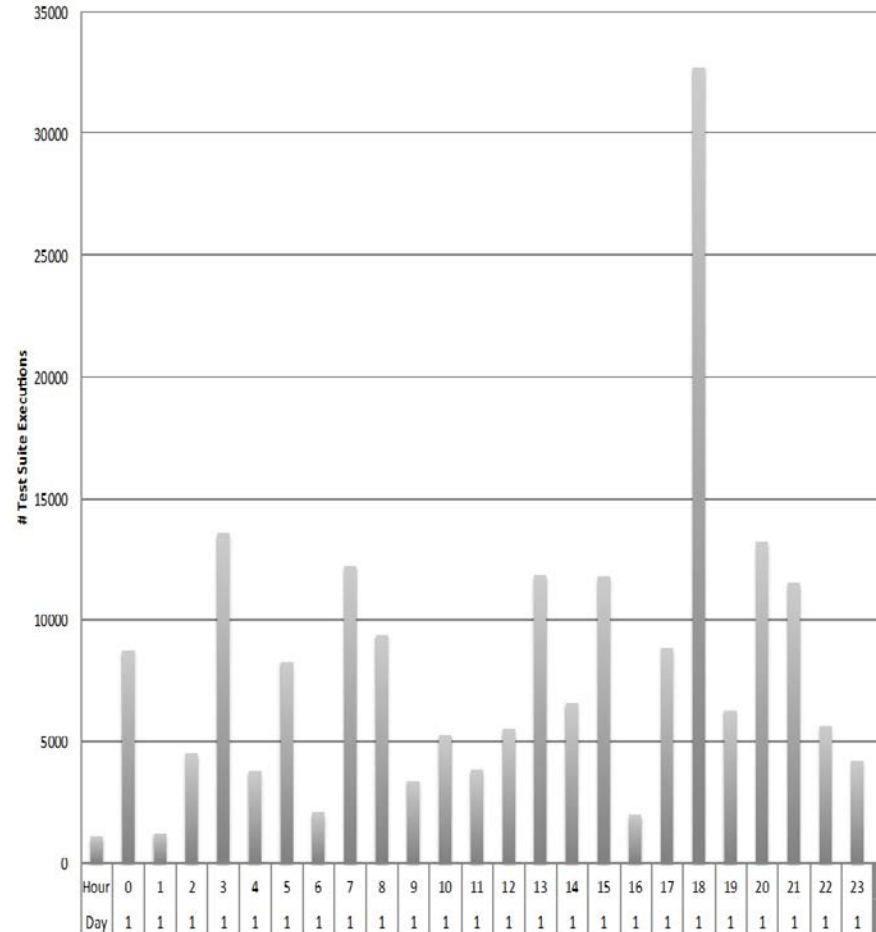
The home of

Over 900k open source projects and 600k users are testing on Travis CI.

932977

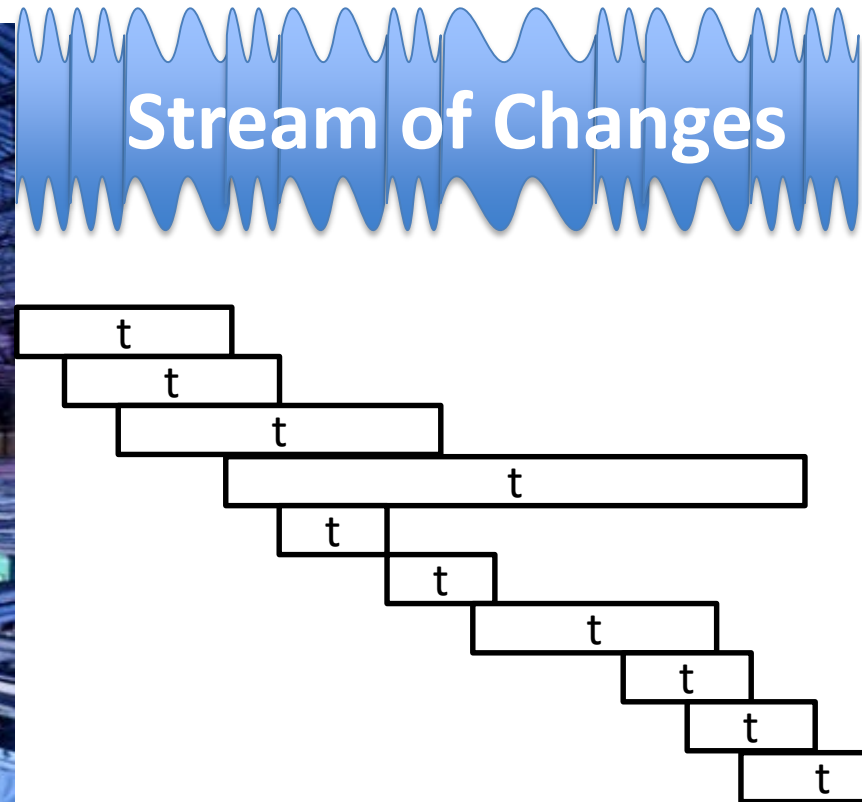
Continuous Integration at Google

- Automated builds, test execution, notification
- Big data centers
- Thousands of developers
- Tens of changes per minute



Continuous Integration at Google

- Automated builds, test execution, notification
- Big data centers
- Thousands of developers
- Tens of changes per minute



The Speed and Scale of Google

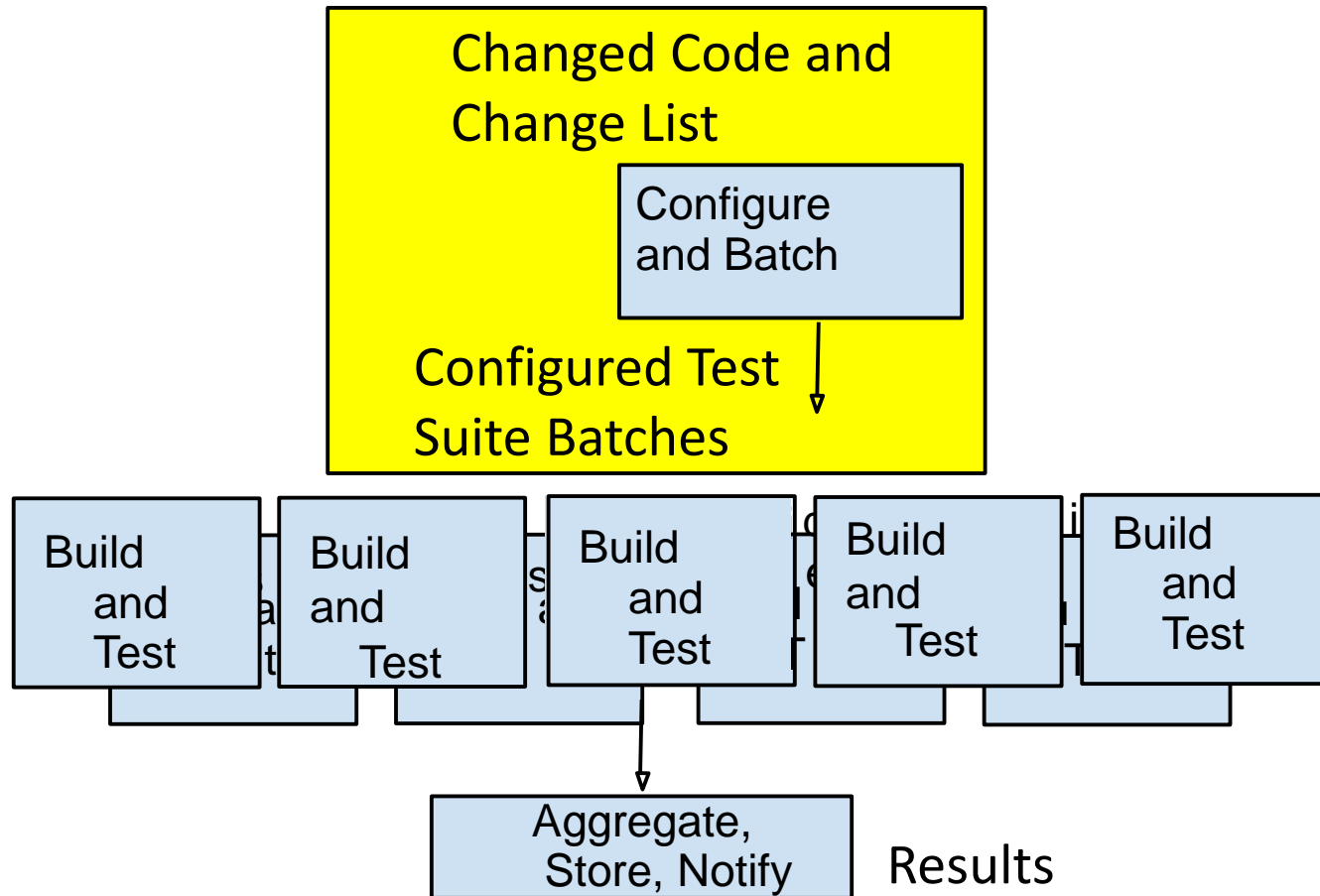
- Single monolithic code tree, mixed languages
- Development on head; all releases from source at head
- More than 10,000 developers in more than 40 offices
- More than 13,000 projects under active development
- 800,000 builds per day
- 20+ code changes per minute
- More than 150 million test cases run per day

From “Testing at the speed and scale of Google”, Gupta, Ivey, Penix,

http://googletesting.blogspot.com/2014/01/the-google-test-and-development_21.html

(Numbers updated from “Taming Google-Scale Continuous Testing”, A. Memon et al., ICSE 2017)

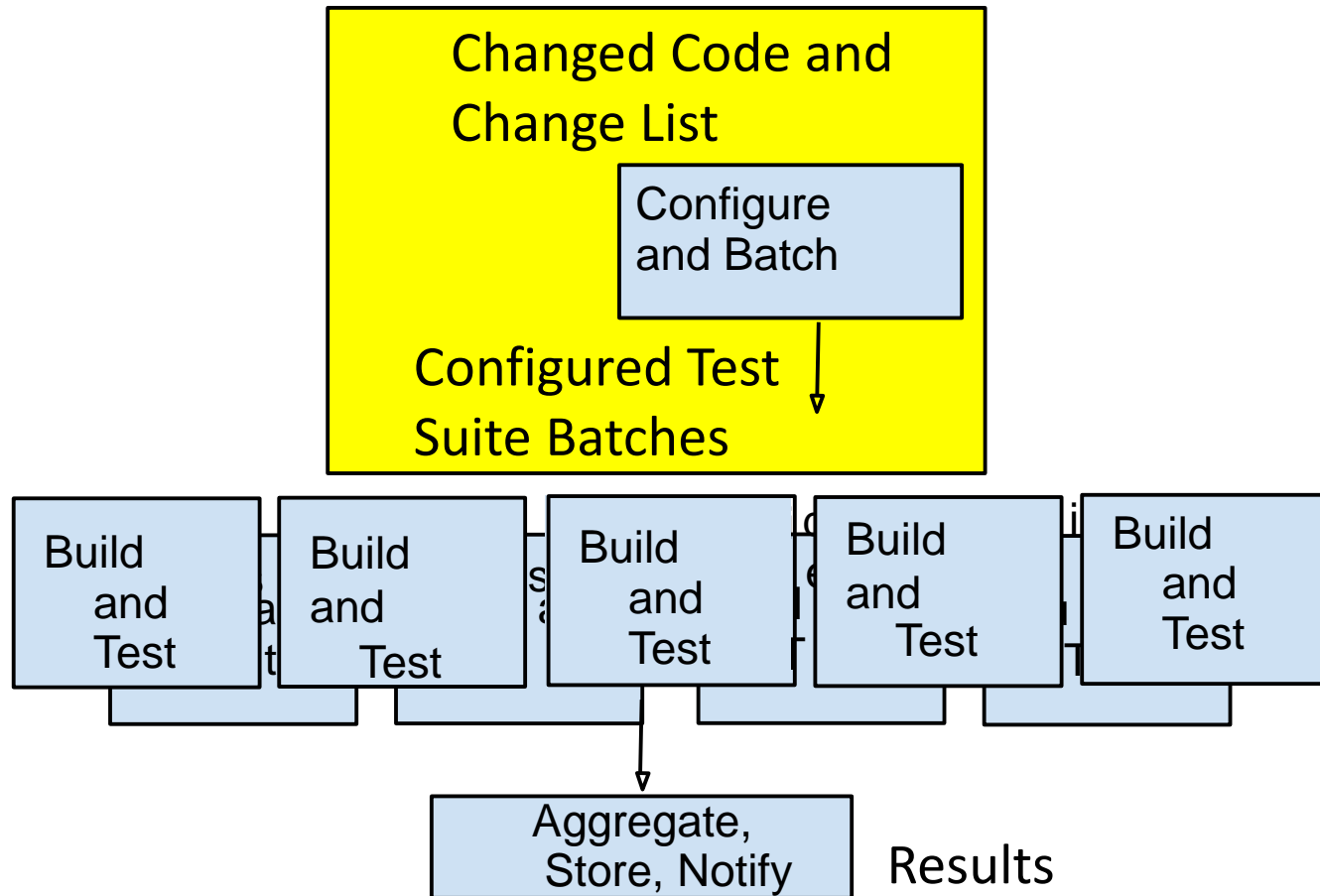
Continuous Testing Process



From "Testing at the speed and scale of Google", Gupta, Ivey, Penix,
http://googletesting.blogspot.com/2014/01/the-google-test-and-development_21.html

Continuous Testing Process

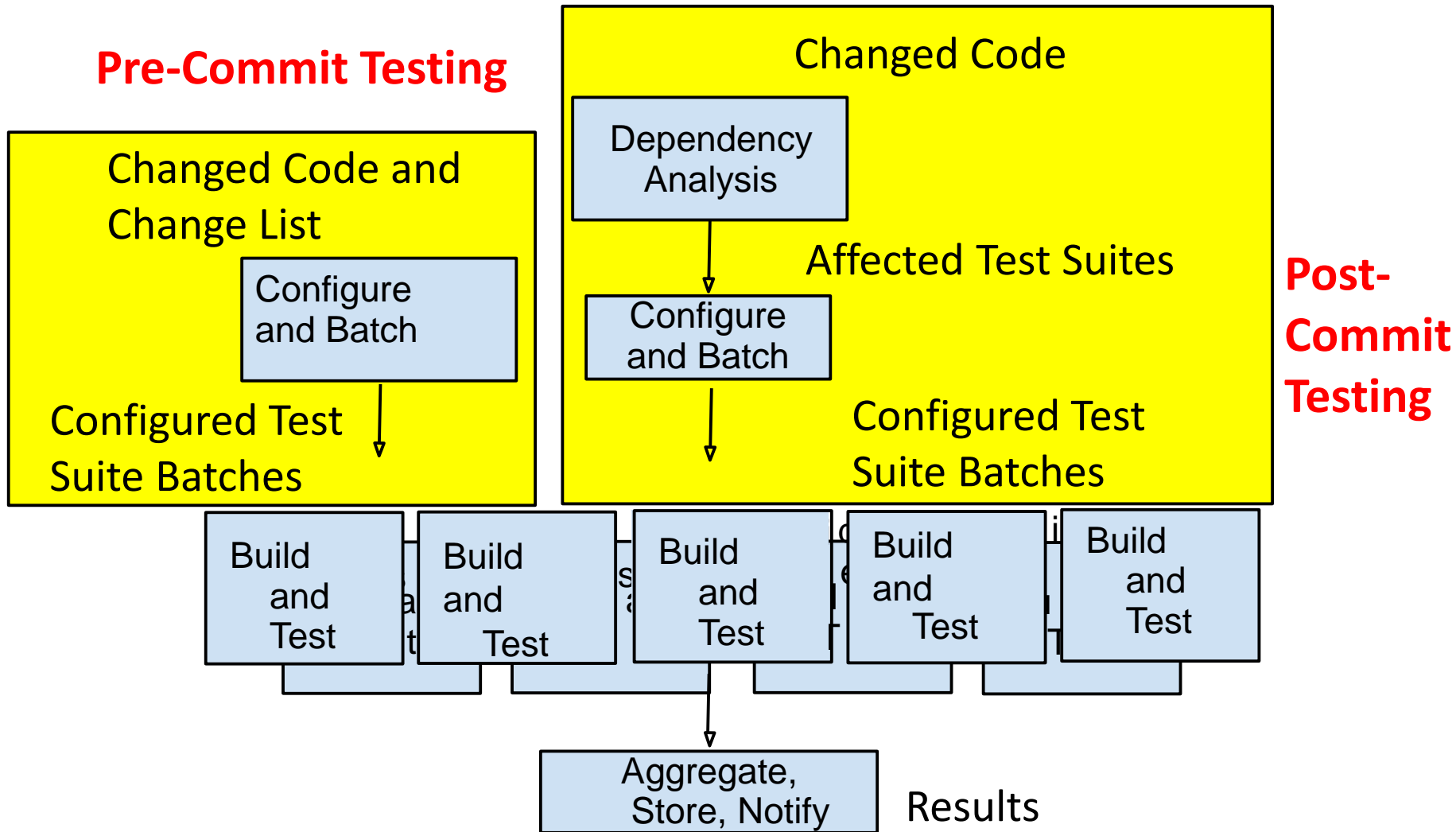
Pre-Commit Testing



From "Testing at the speed and scale of Google", Gupta, Ivey, Penix,

http://googletesting.blogspot.com/2014/01/the-google-test-and-development_21.html

Continuous Testing Process



From "Testing at the speed and scale of Google", Gupta, Ivey, Penix,
http://googletesting.blogspot.com/2014/01/the-google-test-and-development_21.html

Challenges for CI at Google

- Frequent builds (800,000 per day) and testing runs require non-trivial time and resources
- The volume of tests to be run (150 million per day) can delay the speed with which developers receive feedback
 - Developers must wait 45 minutes to 9 hours to receive testing results, despite the massive parallelism that is available.*
- Adding more servers won't solve the problem; testing keeps expanding until it swamps available resources*

*From "Taming Google-Scale Continuous Testing", A. Memon et al., ICSE 2017

Regression Testing Approaches for Continuous Integration Environments*

Part I: Pre-Commit

S. Elbaum, G. Rothermel, J. Penix, “Techniques for Improving Regression Testing In Continuous Integration Development Environments”, FSE 2014

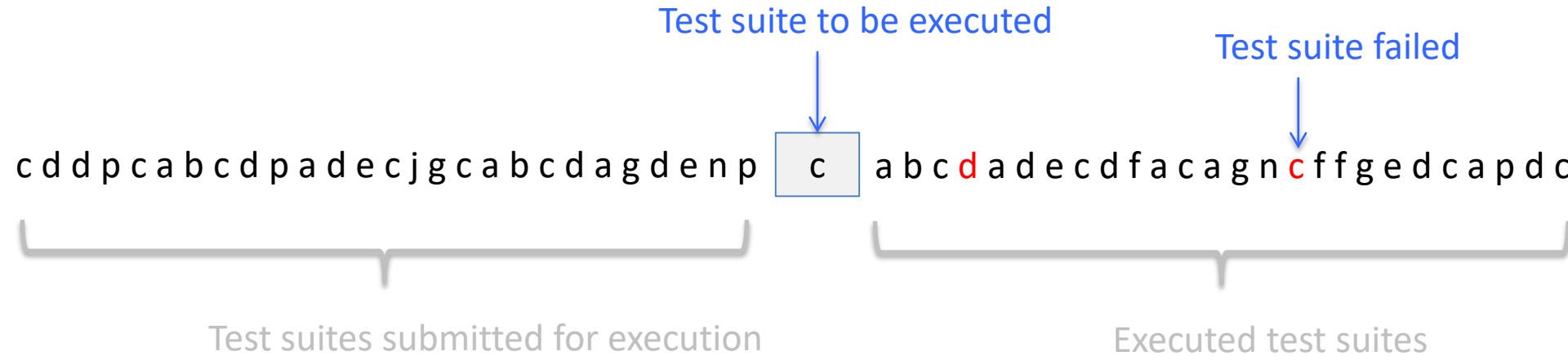
Pre-Commit: Continuous-RTS

- Regression test selection
- At the level of test suites, not test cases
- Except that we cannot assume that:
 - We have reliable coverage information
 - We have time to perform heavyweight analyses
- Key insight: value of lightweight test history data
 - Test suite executions
 - Test suite failures

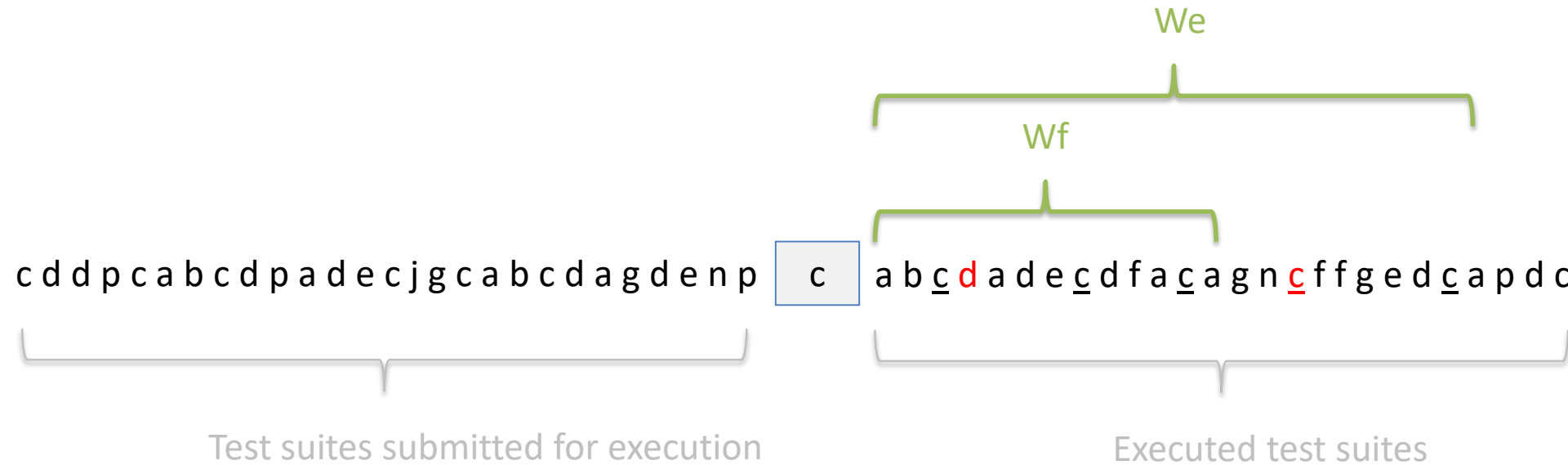
Continuous-RTS Intuition

- If a test suite is new
 - Execute it
- If a test suite failed recently
 - May be associated with “hot” area, worth re-execution
- If a test suite has not executed in a while
 - Needs a “refresher”
- Else skip it

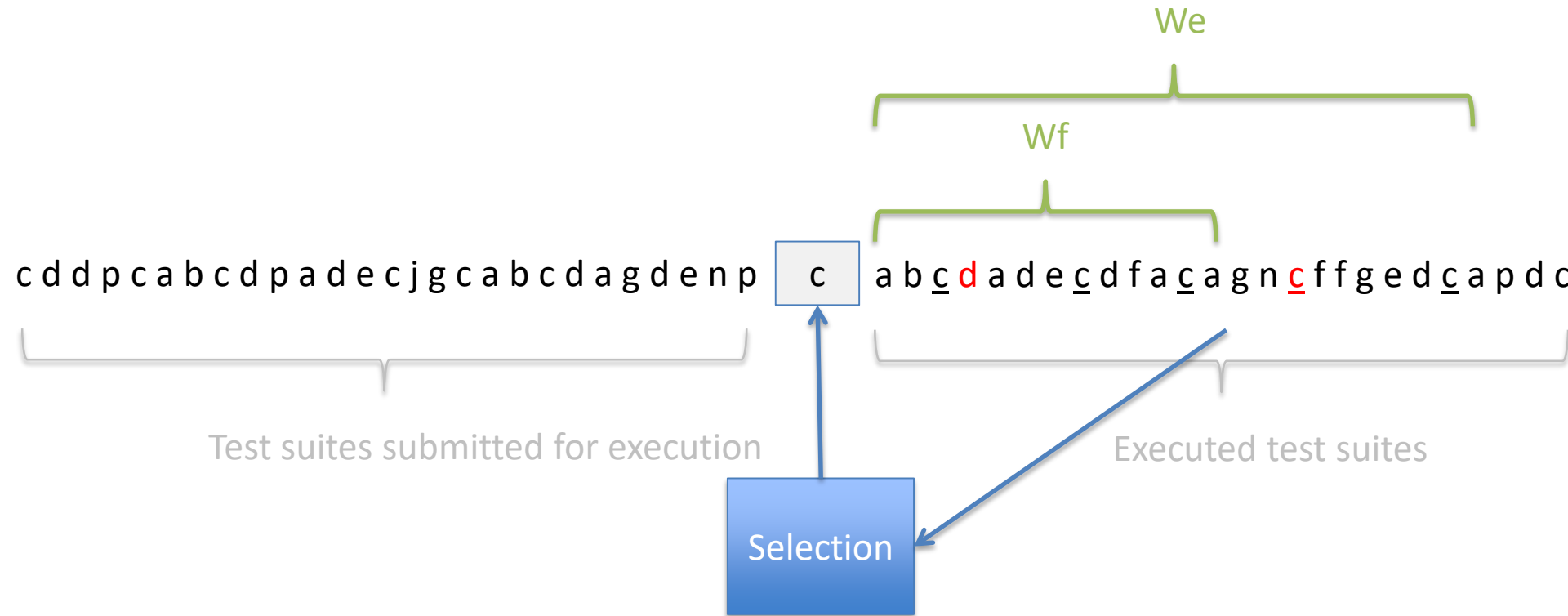
Continuous-RTS Example



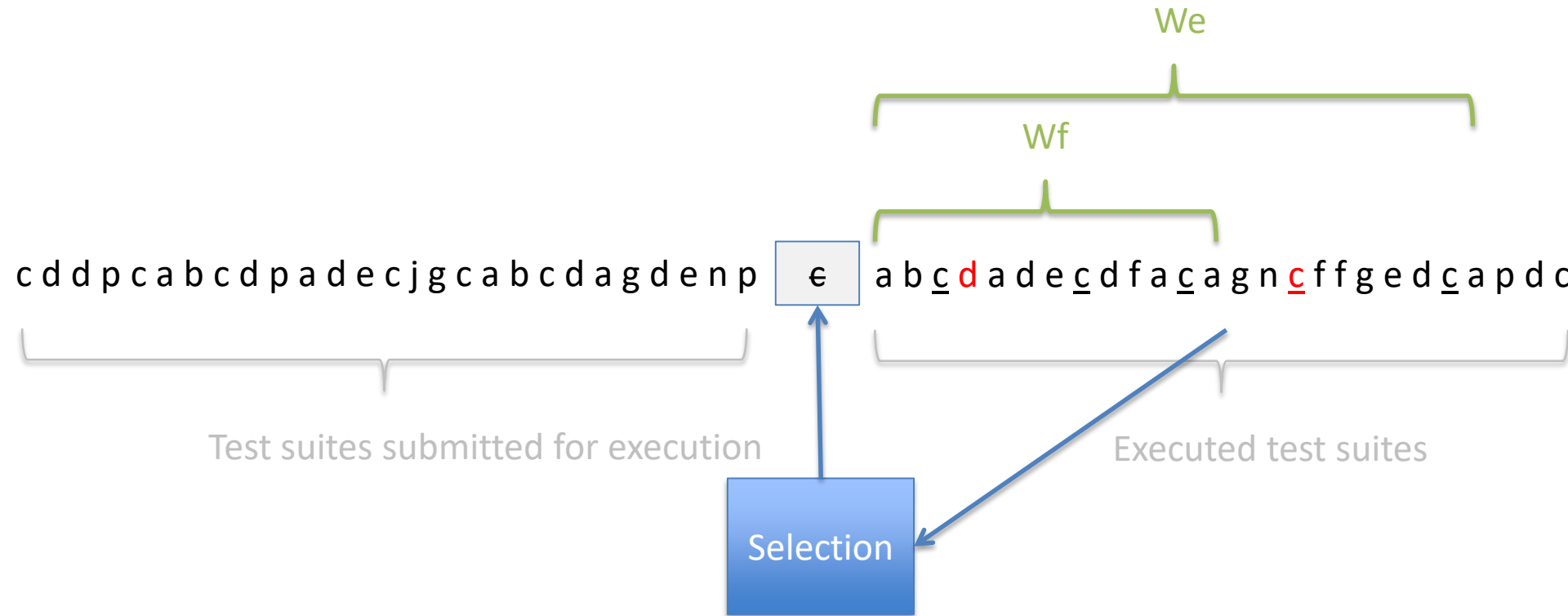
Continuous-RTS Example



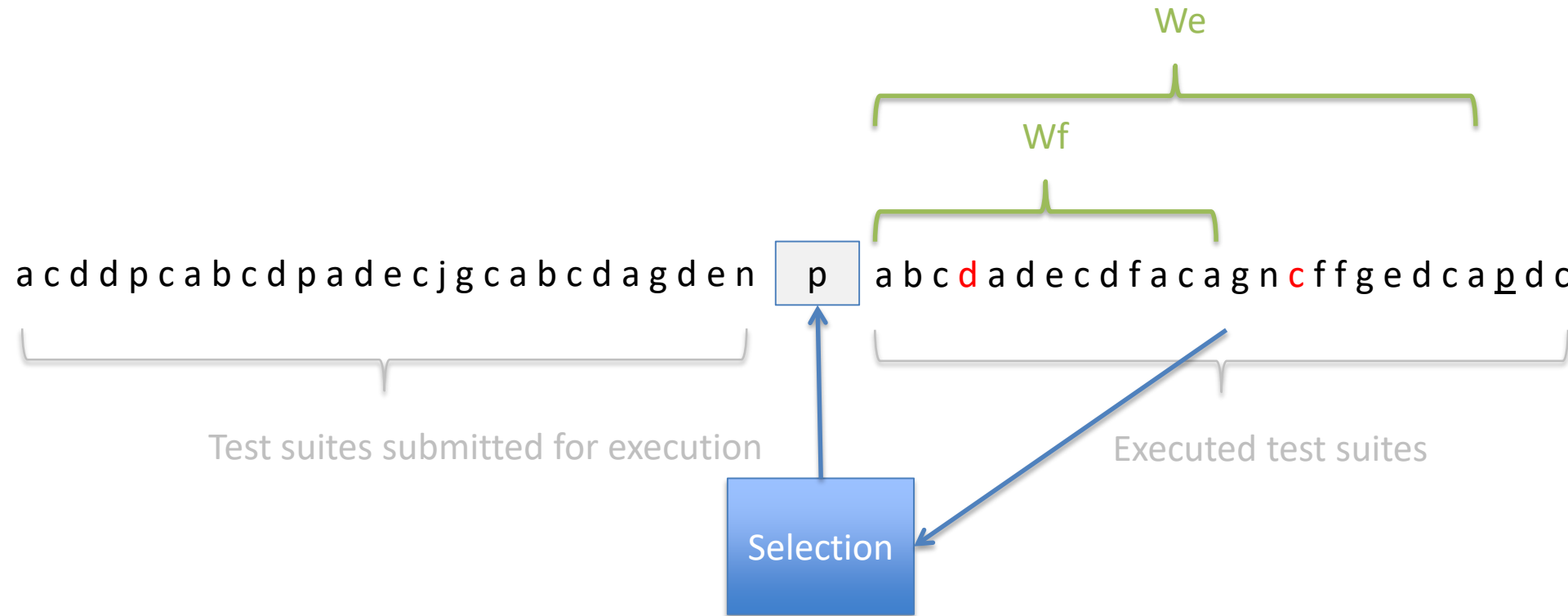
Continuous-RTS Example



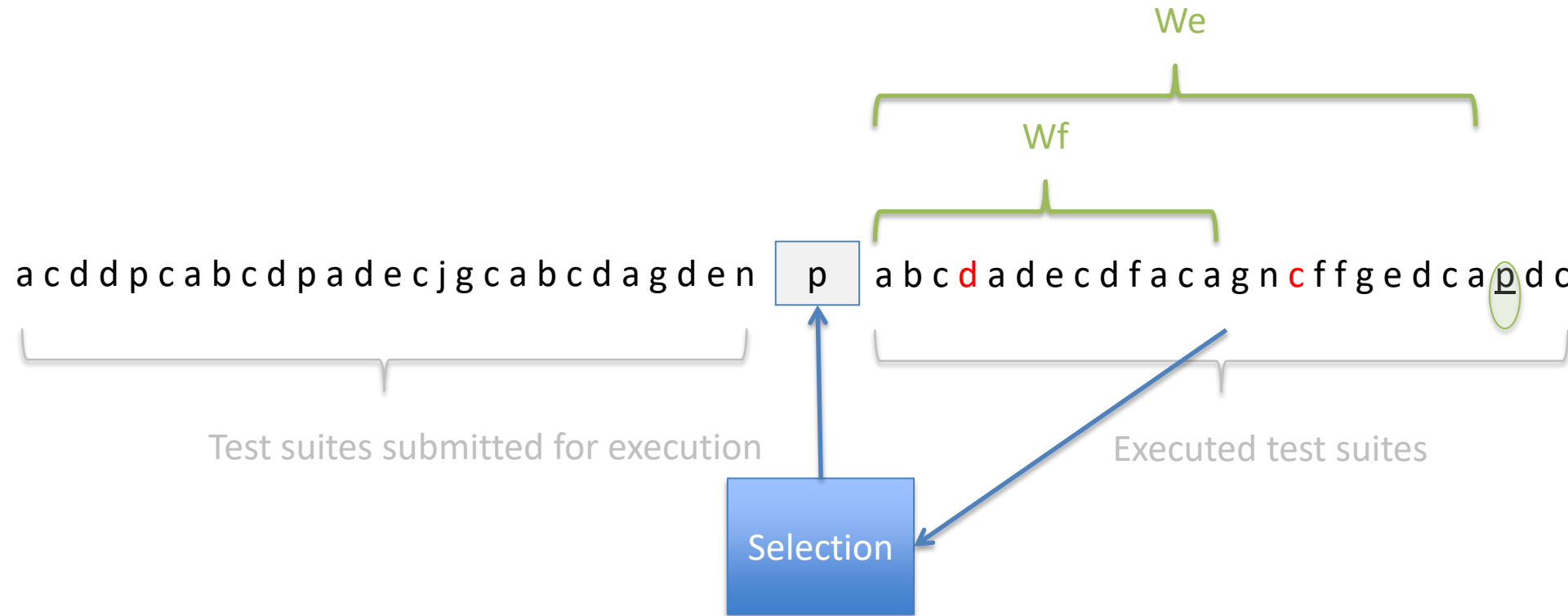
Continuous-RTS Example



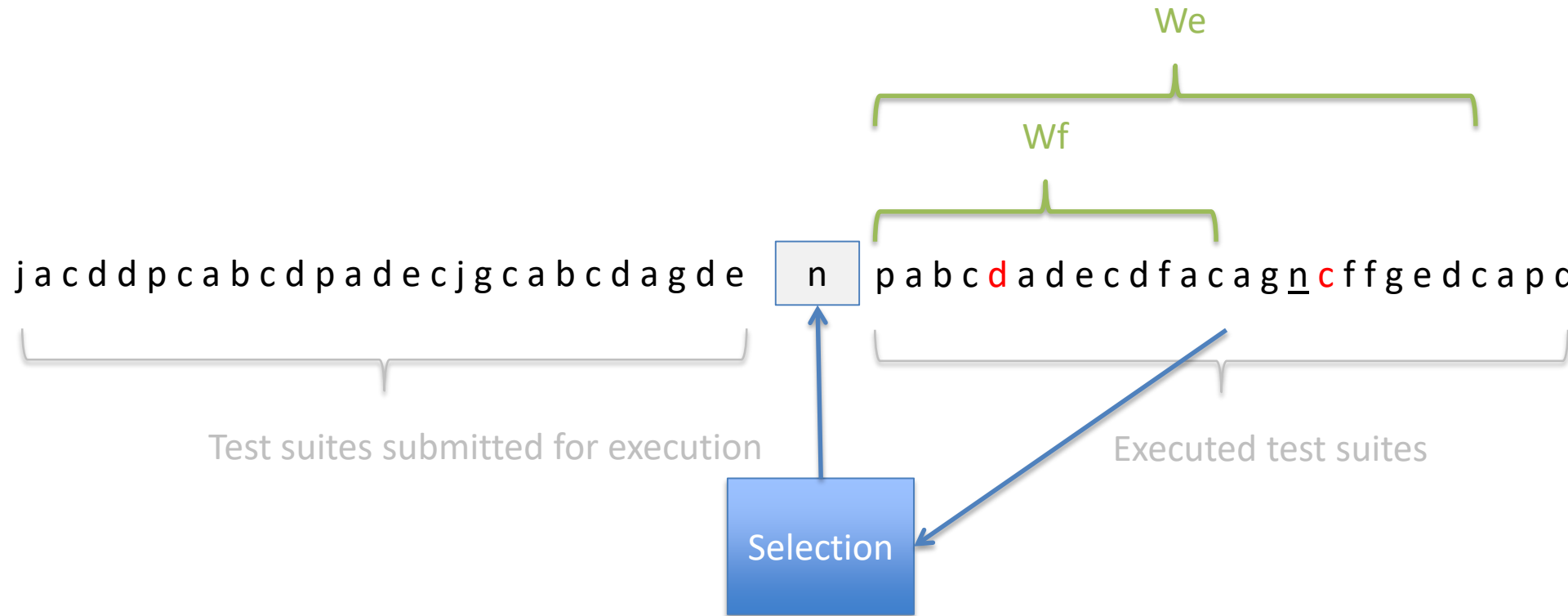
Continuous-RTS Example



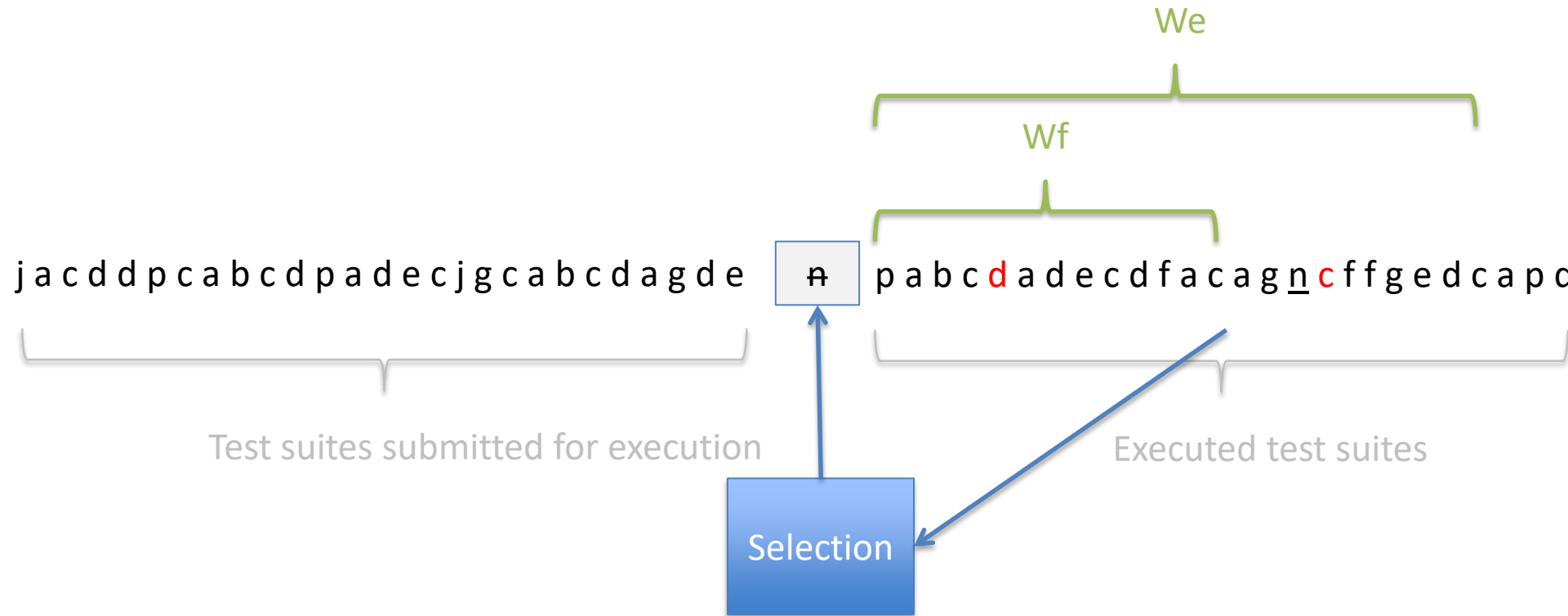
Continuous-RTS Example



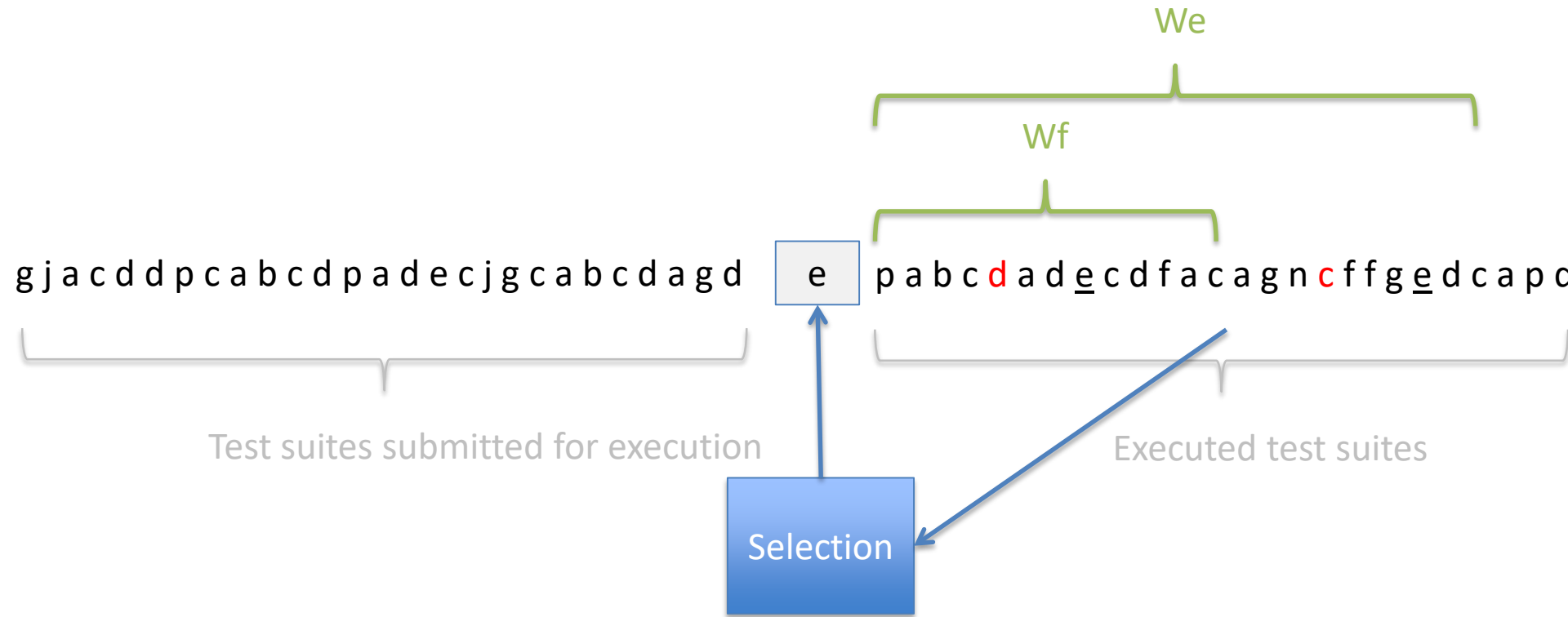
Continuous-RTS Example



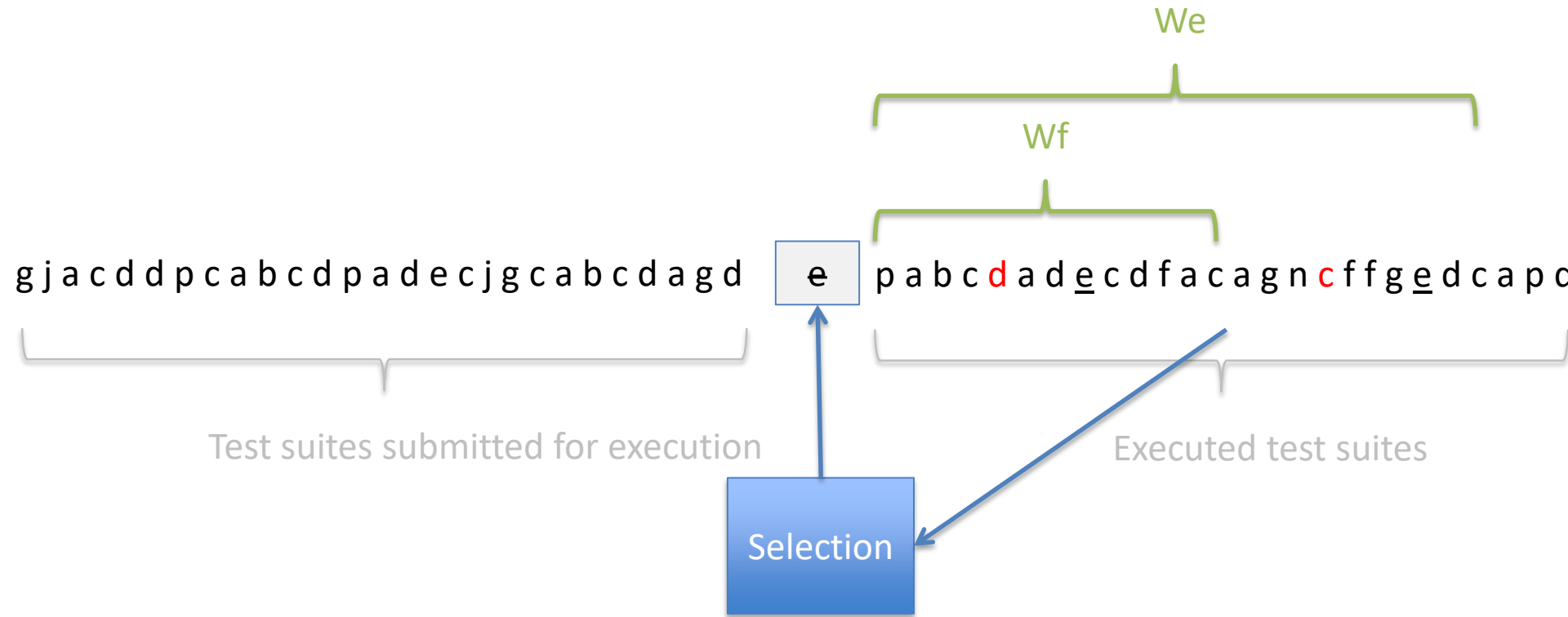
Continuous-RTS Example



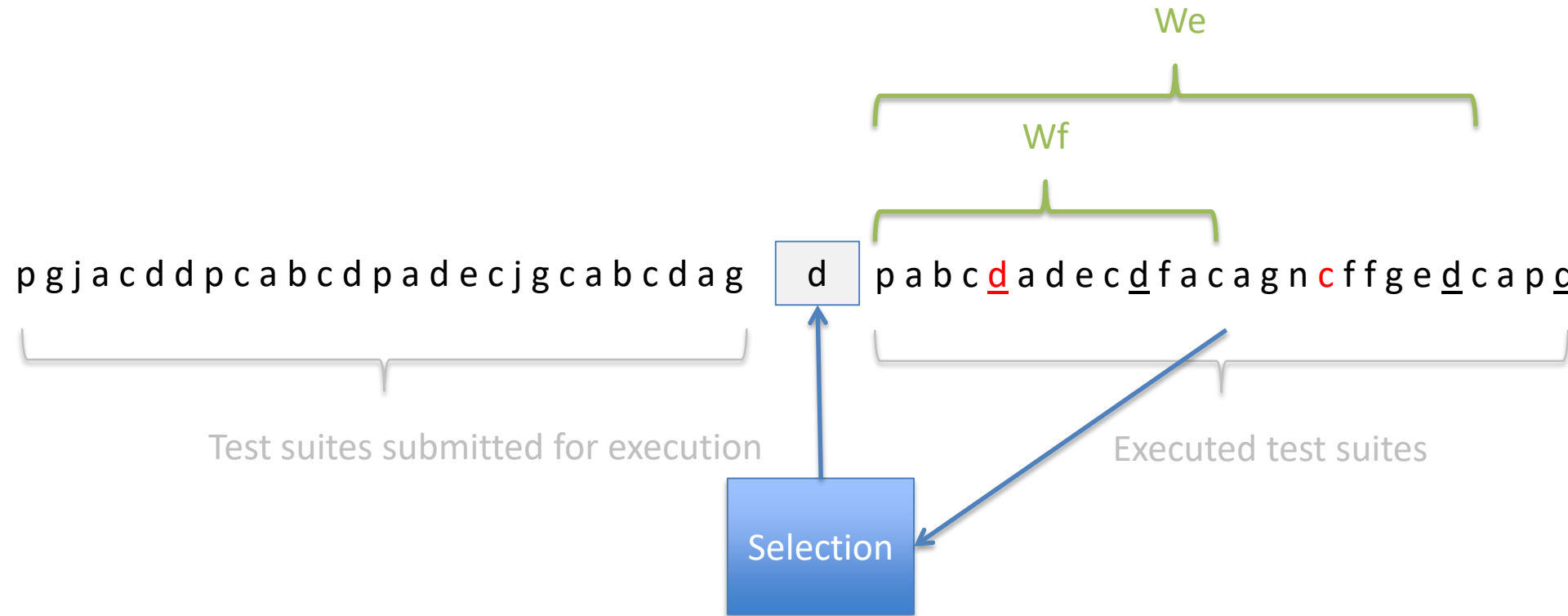
Continuous-RTS Example



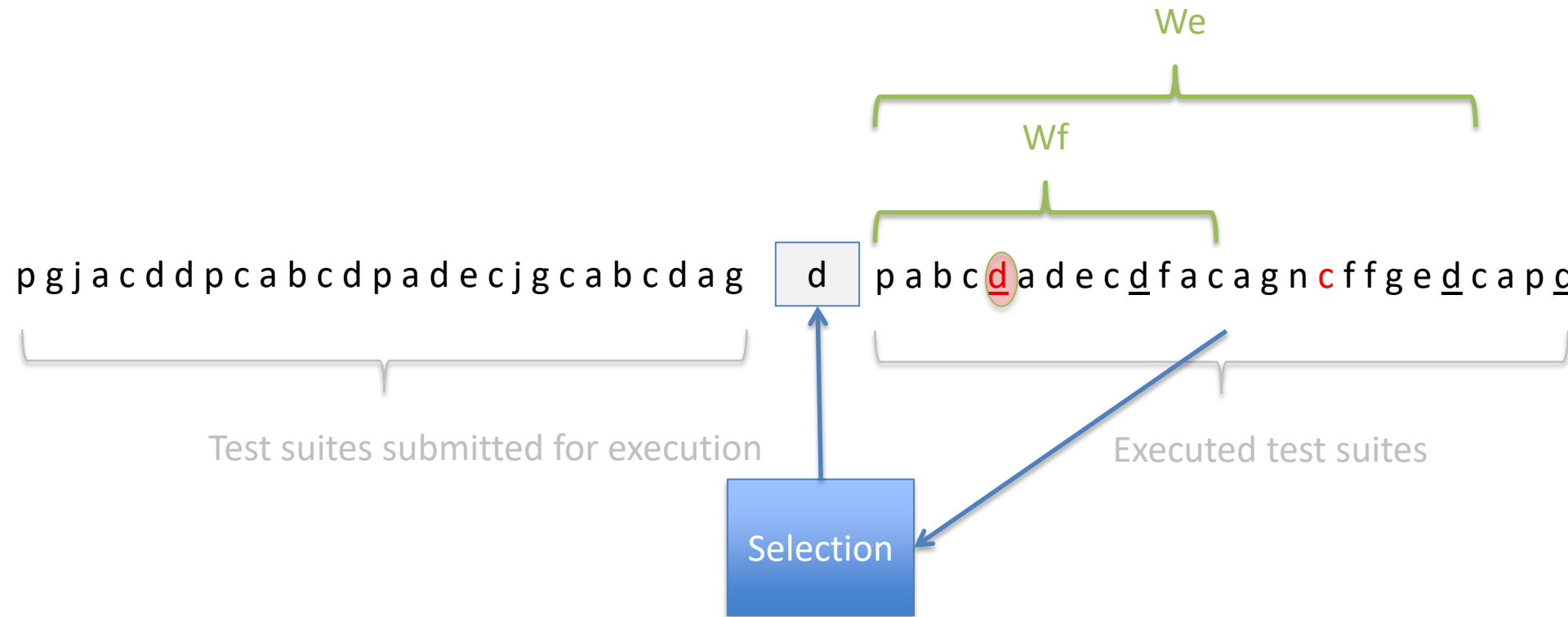
Continuous-RTS Example



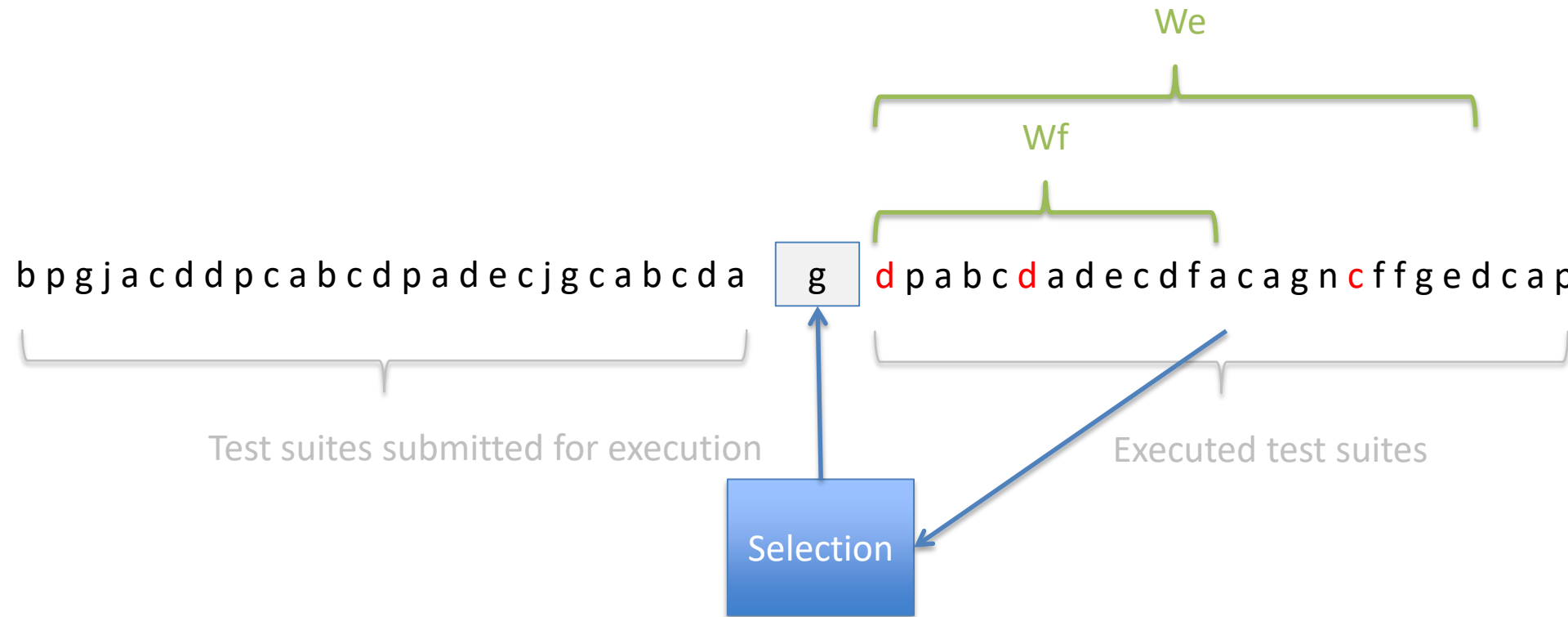
Continuous-RTS Example



Continuous-RTS Example



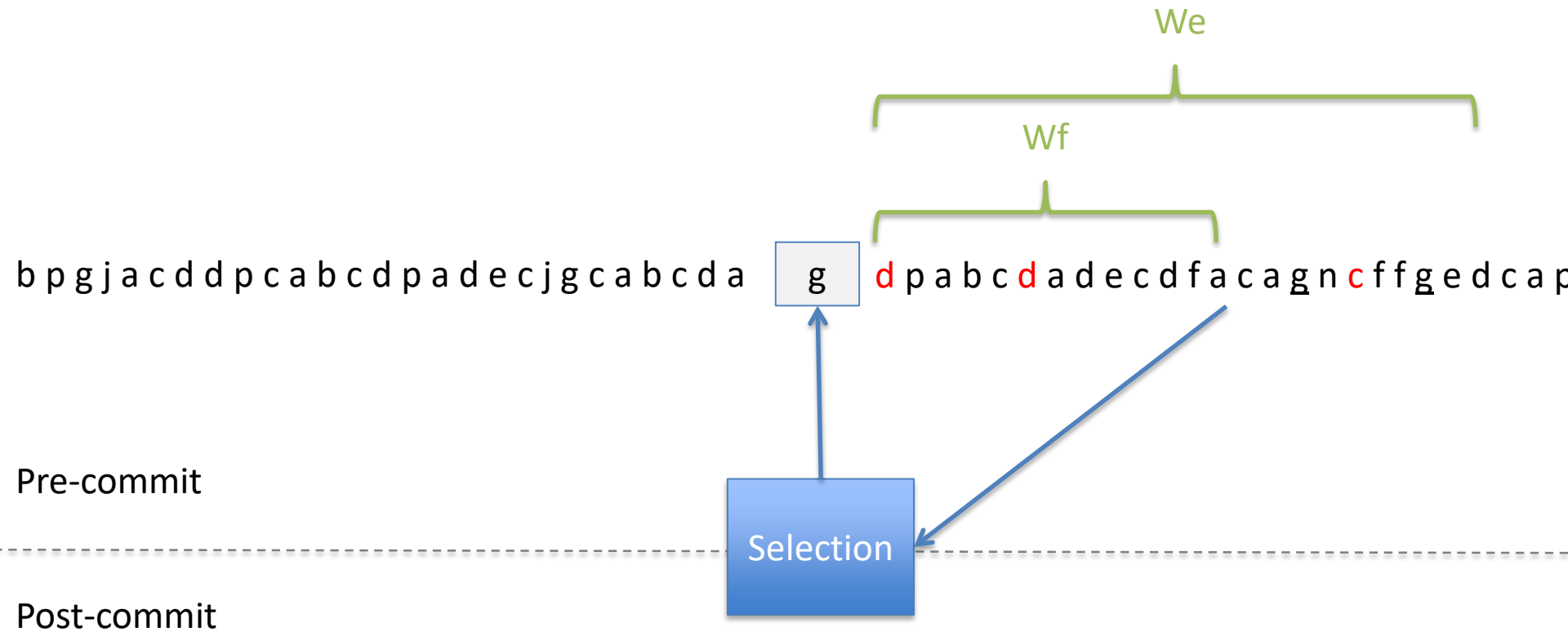
Continuous-RTS Example



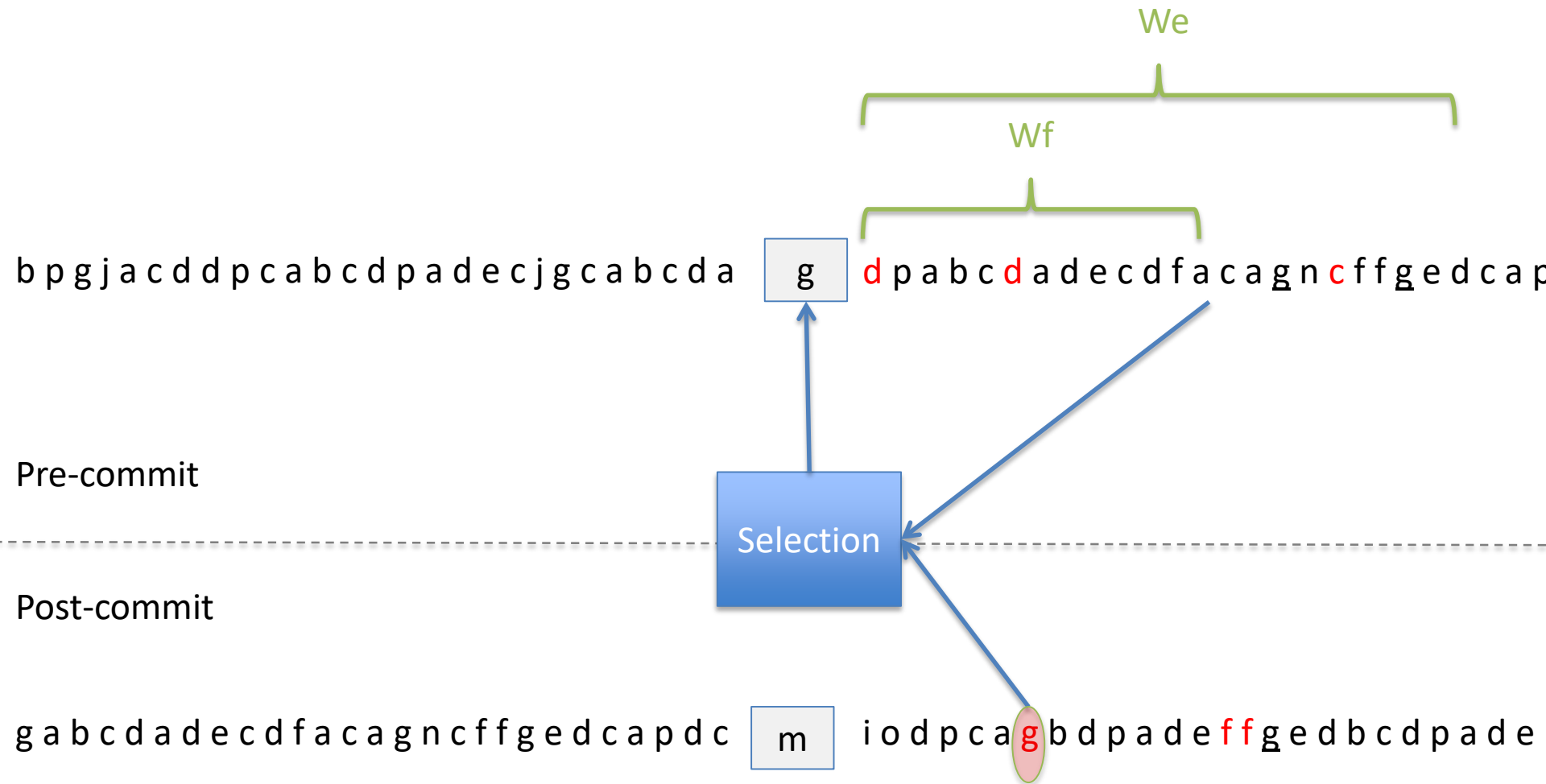
Continuous-RTS Intuition II

- If a test suite is new
 - Execute it
- If a test suite has failed recently
 - May be associated with “hot” area, worth re-execution
- If a test suite has not executed in a while
 - Needs a “refresher”
- Else skip
 - Use the post-commit testing phase as a safety net and a source of information

Continuous-RTS Example



Continuous-RTS Example



Continuous-RTS Algorithm

```
/*  
 * Called After Every Pre-Commit Request  
 */  
  
CI-RTS ( $T$ ,  $W_f$ ,  $W_e$ ) //  $T$ : set of test suites associated with a commit  
  
  forall  $T_i \in T$  do  
    if TimeSinceLastFailure( $T_i$ )  $\leq W_f$  or // hot test suite  
       TimeSinceLastExecution( $T_i$ )  $> W_e$  or // need refresher  
        $T_i$  is new // just added  
    then  $T' \leftarrow T' \cup T_i$   
  end if  
end for  
return  $T'$ 
```

Research Question

- RQ: How cost-effective is CI-RTS, and how does its cost-effectiveness vary with different settings of W_f and W_e ?

Google Testing Dataset

3.5 million test executions gathered over 30 days on change requests, info on test times and phases when executed, test outcomes, test execution times. About 0.05% of the test executions result in *failures*.

sebastianelbaum@gmail.com ▾ | [My favorites](#) ▾ | [Profile](#) | [Sign out](#)



google-shared-dataset-of-test-suite-results

Google shared dataset of test suite results

 Search projects

Project Home | [Wiki](#) | [Issues](#) | [Source](#) | [Administer](#)

Summary | [People](#)

Project Information

★ Starred by 4 users

[Project feeds](#)

Code license

[New BSD License](#)

Labels

[Research](#), [Dataset](#), [Testing](#)

Members

[sebastianelbaum](#),
selbaum@google.com,
jpenix@google.com,
amclaugh@google.com

Your role

[Owner](#)

Links

External links

[Other Dataset: SIR](#)
[Other Dataset: PROMISE](#)
[Issues](#)

The dataset provides the software testing and analysis community with a sample of **3.5 Million** test suite execution results from a fast and large scale continuous testing infrastructure.

Details on the rationale, context, and process to get the data can be found in [SummaryDataset](#), details on the data fields can be found in [DataFields](#), faults/suggestions/requests about the dataset can be seen and reported under Issues, a list of efforts using the dataset can be found under [UsaGe](#), and a list of frequently asked questions including potential "gotcha"s can be found at [FaQs](#).

If you use it please:

- Cite it as: Sebastian Elbaum, Andrew McLaughlin, and John Penix, "The Google Dataset of Testing Results", <https://code.google.com/p/google-shared-dataset-of-test-suite-results>, 2014.

- (Nov 2014) Let us know so that we can add your work to the [UsaGe](#) page.

Variables

- Independent
 - Technique: Continuous-RTS, Retest-all, Random
 - We: 24, 48, 96 (hours)
 - Wf: 0.25, 0.5, 1, 2, 4, 12, 24, 48, 96 (hours)
- Dependent
 - Percentage of failures detected
 - Percentage of test suite executed
 - Percentage of test suite execution time required

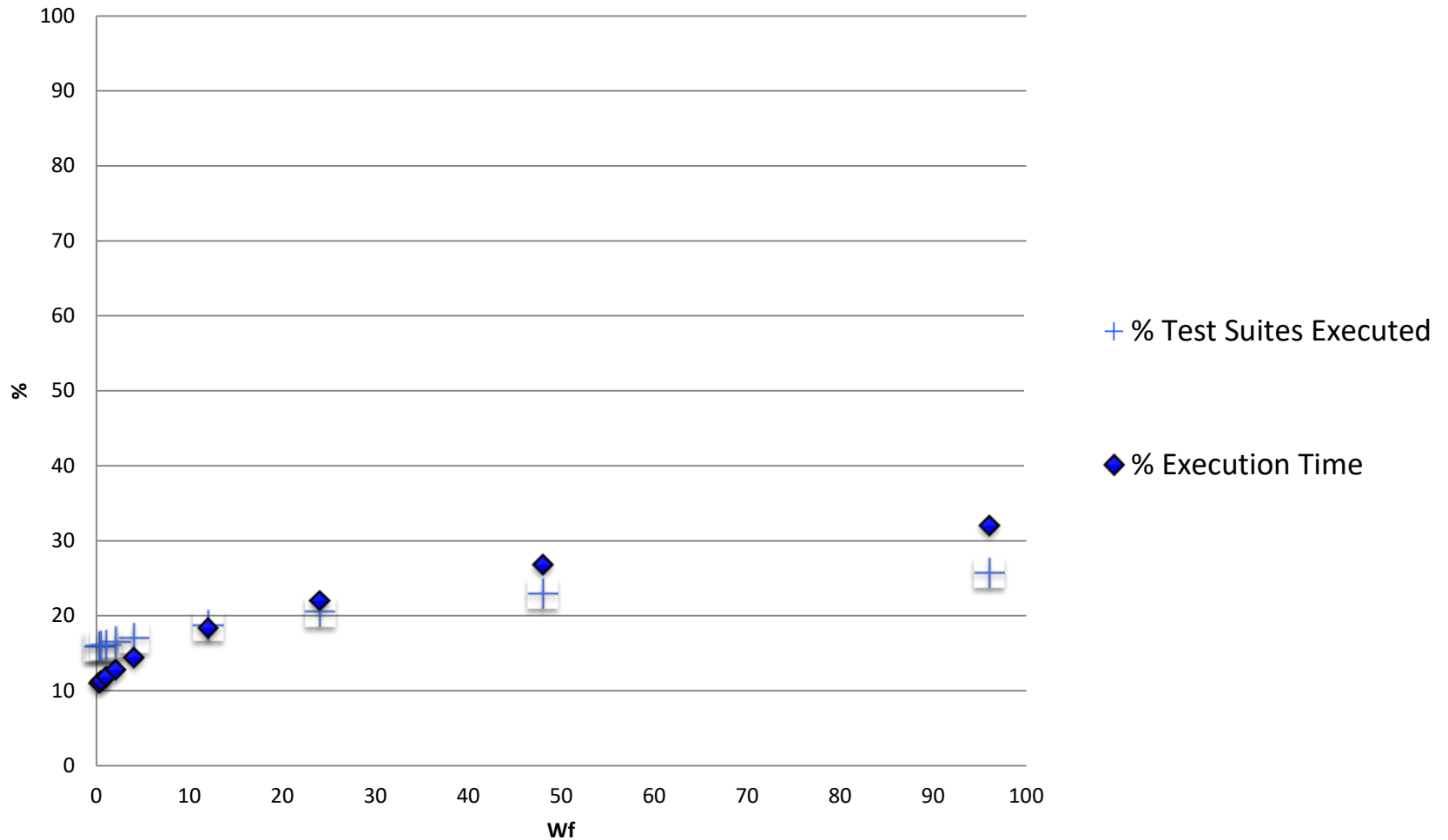
Study Operation

- Implemented Continuous-RTS algorithm in Python
- Simulated techniques by walking through the Google Testing Dataset

Threats to Validity

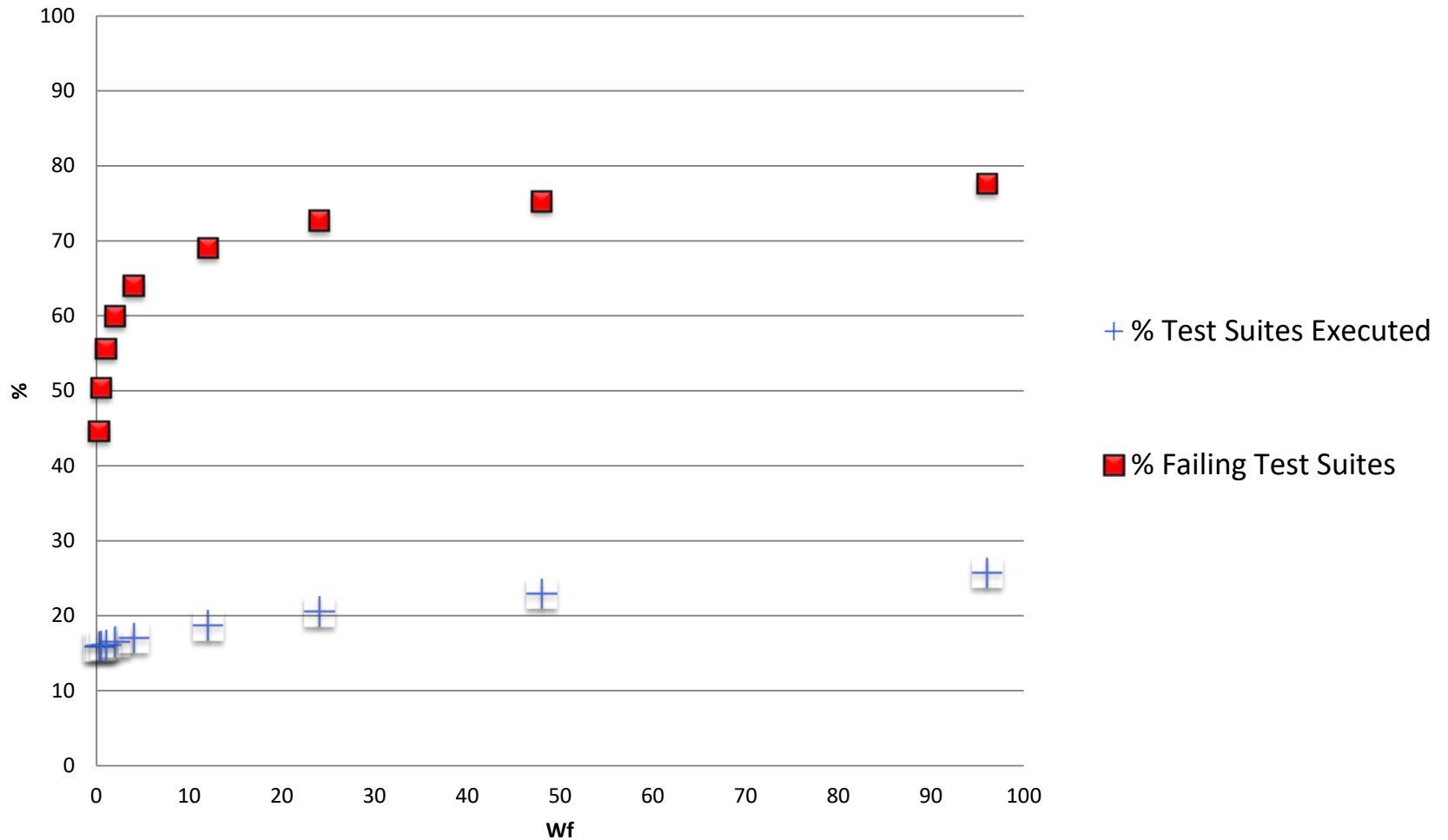
- External:
 - Only one dataset, only a few baselines, limited set of window sizes, no consideration of available computing infrastructure
- Internal:
 - Possible implementation errors
- Construct:
 - Measures such as costs in engineer time or delays in receiving feedback not considered

Results: CI-RTS in Pre-Commit



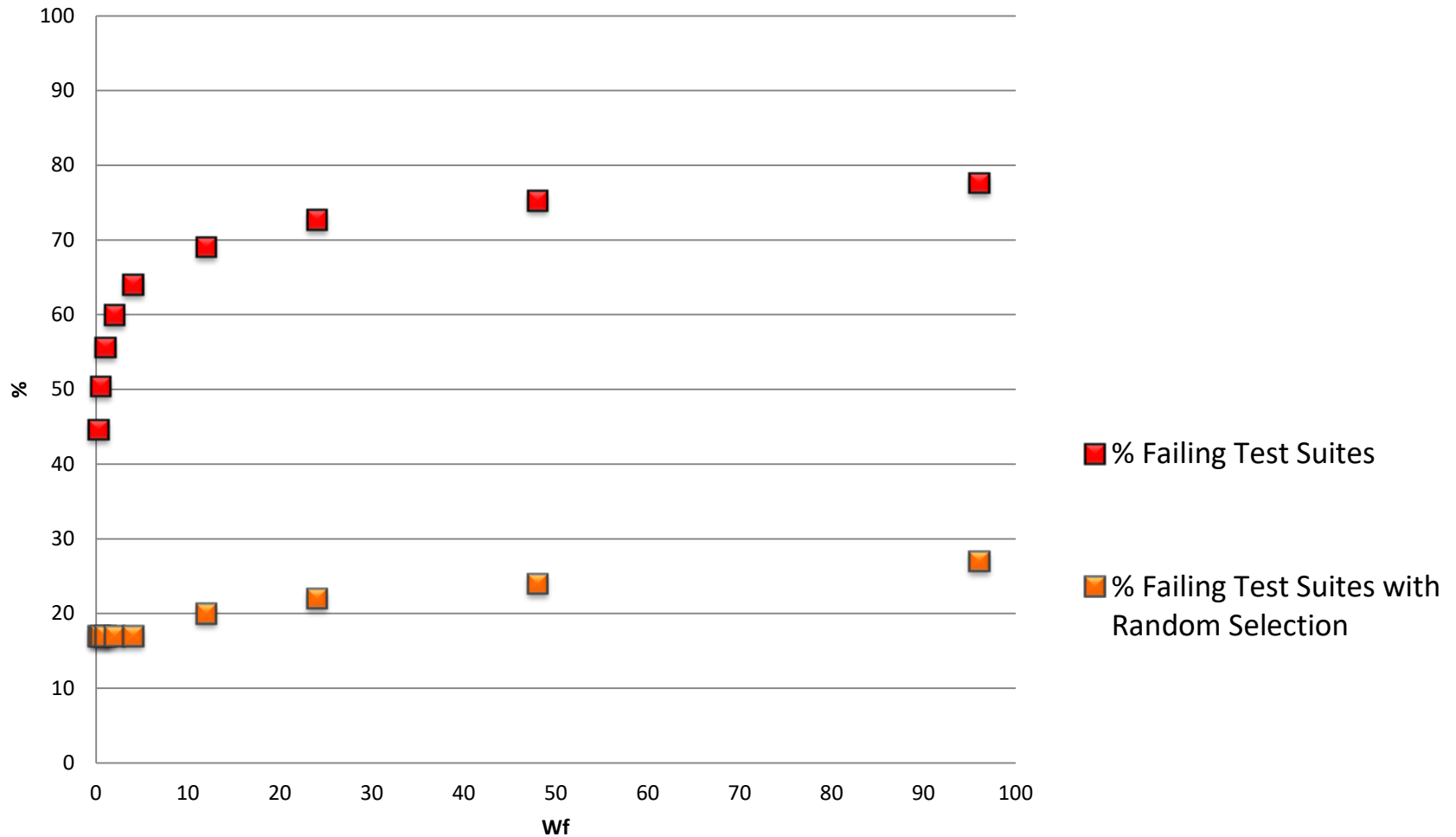
We = 24 hours

Results: CI-RTS in Pre-Commit



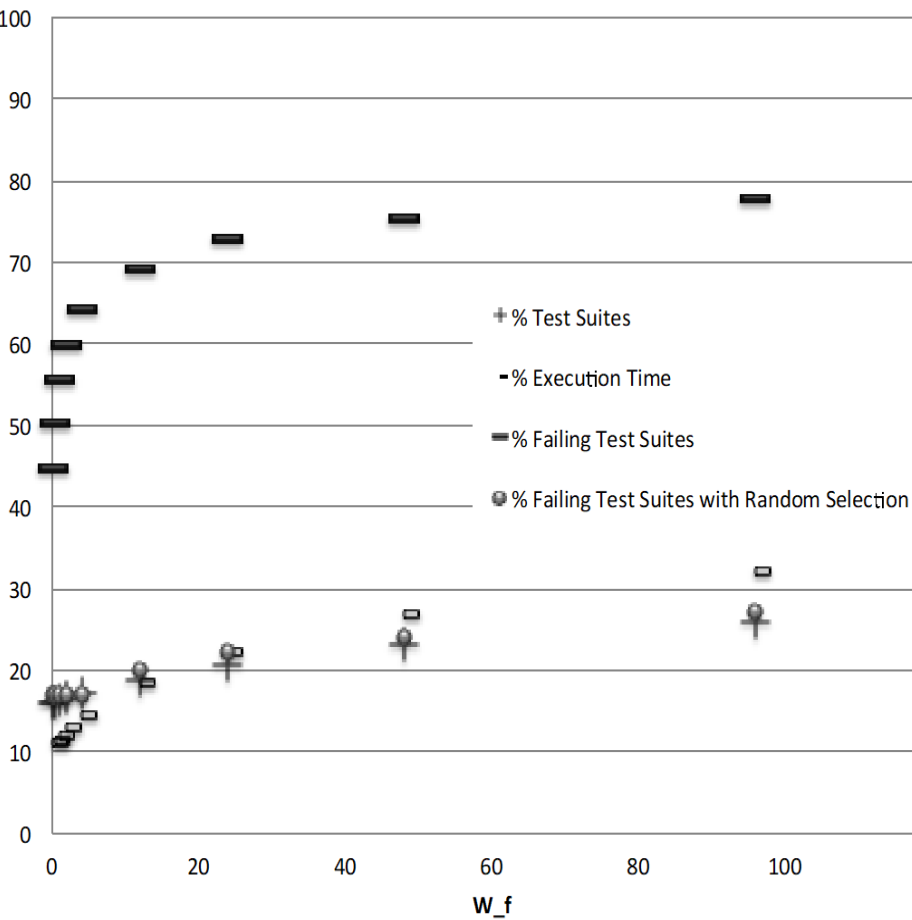
We = 24 hours

Results: CI-RTS in Pre-Commit

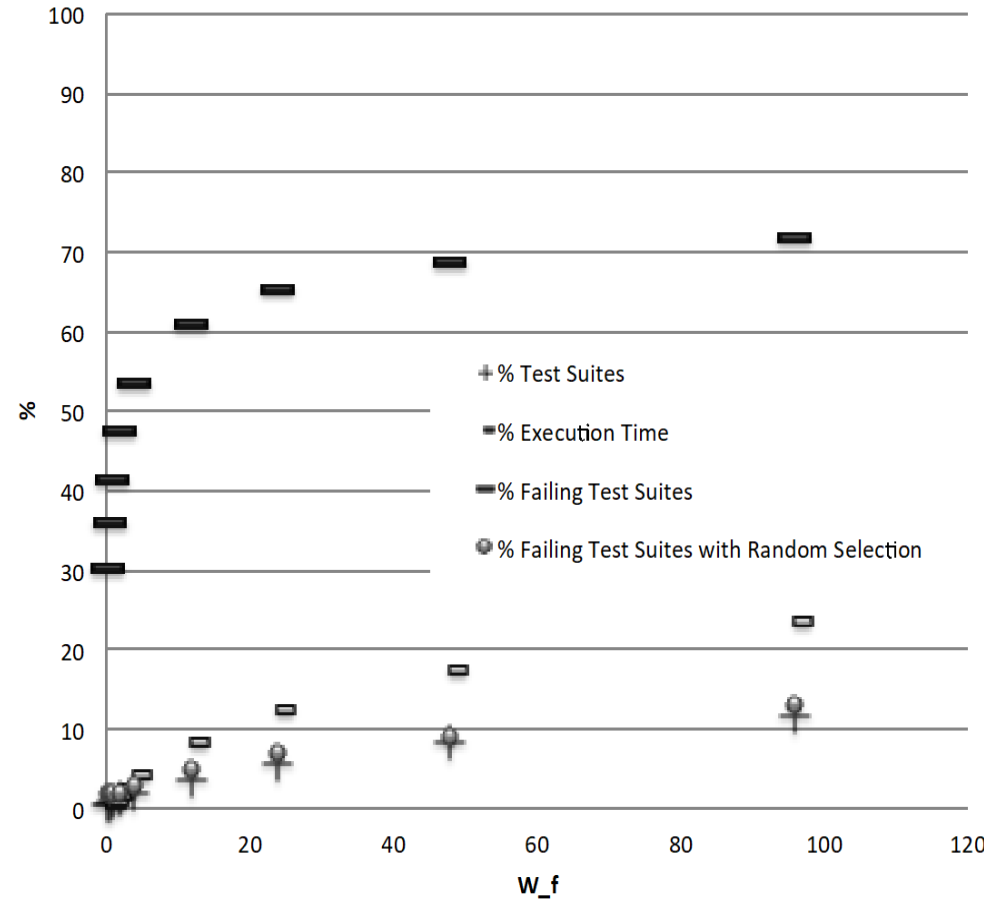


We = 24 hours

Results: CI-RTS in Pre-Commit



We = 24 hours



We = 48 hours

Regression Testing Approaches for Continuous Integration Environments

Part II: Post-Commit

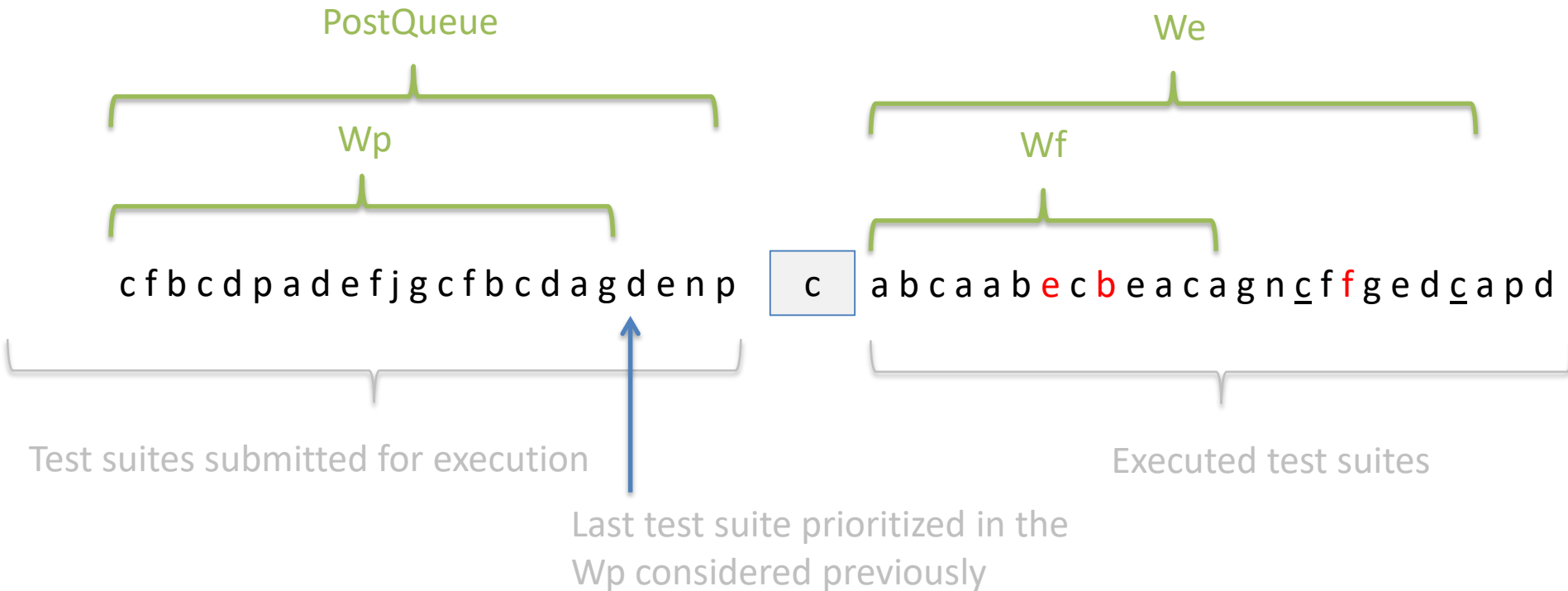
Post-Commit: Continuous-TSP

- Test Suite Prioritization (TSP)
- At the level of test suites, not test cases
- Except that we cannot assume that:
 - We have reliable coverage information
 - We have time to perform heavyweight analyses
- Key insight: value of lightweight test history data
 - Test executions
 - Test failures
- Key difference from RTS: We can't prioritize per test suite arrival, we need to prioritize over a "batch" or "window" **Wp** of arrivals, found in a **PostQueue**

Continuous-TSP Intuition

- If a test suite in W_p is new
 - It needs to be executed; rank “higher” than others
- If a test suite in W_p failed recently
 - May be associated with “hot” area; worth ranking “higher” than others
- If a test suite in W_p has not executed in a while
 - Needs a “refresher”; worth ranking “higher” than others
- Else rank it lower

Continuous-TSP Example



Continuous-TSP Example

PostQueue

Wp

c f c a d f j g c f c a g b d p e b d d e n p

Test suites submitted for execution

We

Wf

c

a b c a a b **e** **c** **b** e a c a g n c **f** g e d c a p d

Executed test suites

Research Question

- RQ: How cost-effective is CI-TSP, and how does its cost-effectiveness vary with different settings of W_p ?

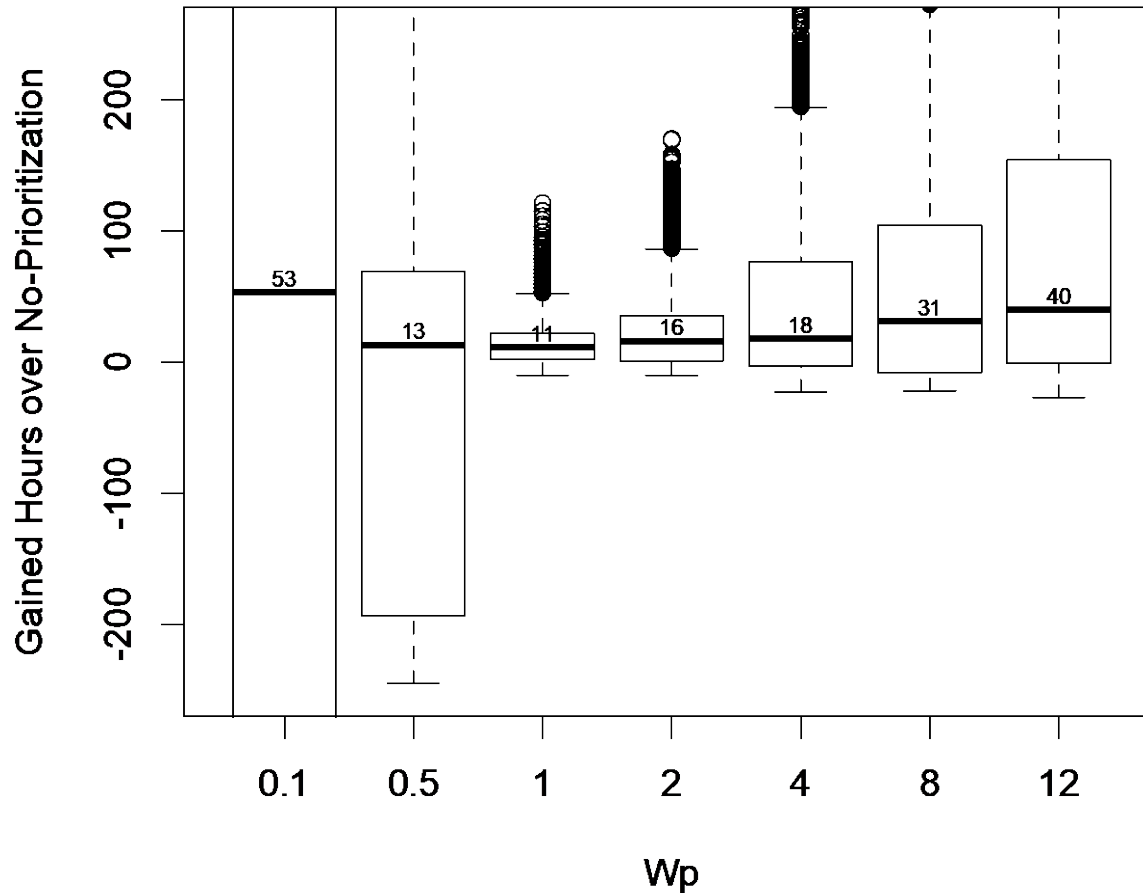
Variables

- Independent
 - Technique: CI-TSP, Unprioritized
 - W_p : 0.1, 0.5, 1, 2, 4, 8, 12 (hours)
 - W_f set to 12, W_e set to 24 (hours)
- Dependent
 - Hours gained in exposing failures

Study Operation

- Implemented C-TSP in Python
- Simulated techniques by walking through the Google Testing Dataset

Results: CI-TCP in Post-Submit



- All instantiations of prioritization outperform no prioritization (medians)
- Lower W_p values lead to higher variance
- Higher W_p values boost hours gained

Conclusion

- With CI, testing is no longer discrete and bounded
- Testing approaches that assume it is are not going to scale or keep up
- Need to perform “instantaneous” selection, prioritization, and analysis
- Our CI-RTS and CI-TSP approaches achieve this

Ongoing and Future Work

- Additional studies on additional data sets
 - More effective RTS/TCP techniques
 - Let windows represent numbers of tests, not time
 - Prioritize commits, not test suites*
 - Dynamically adjust windows (adapt) as throughput changes
 - Use additional sources of information
 - Test suites can fail too. Use data on test suite modification as part of RTS/TCP
 - Consider large-scale computing infrastructures
 - Consider other variants of CI processes
-
- J. Liang, S. Elbaum, G. Rothermel, “Redefining Prioritization: Continuous Prioritization for Continuous Integration”, ICSE 2018

Improving Regression Testing in Continuous Integration Development Environments

Gregg Rothermel

Department of Computer Science and Engineering
University of Nebraska – Lincoln