Regression Testing

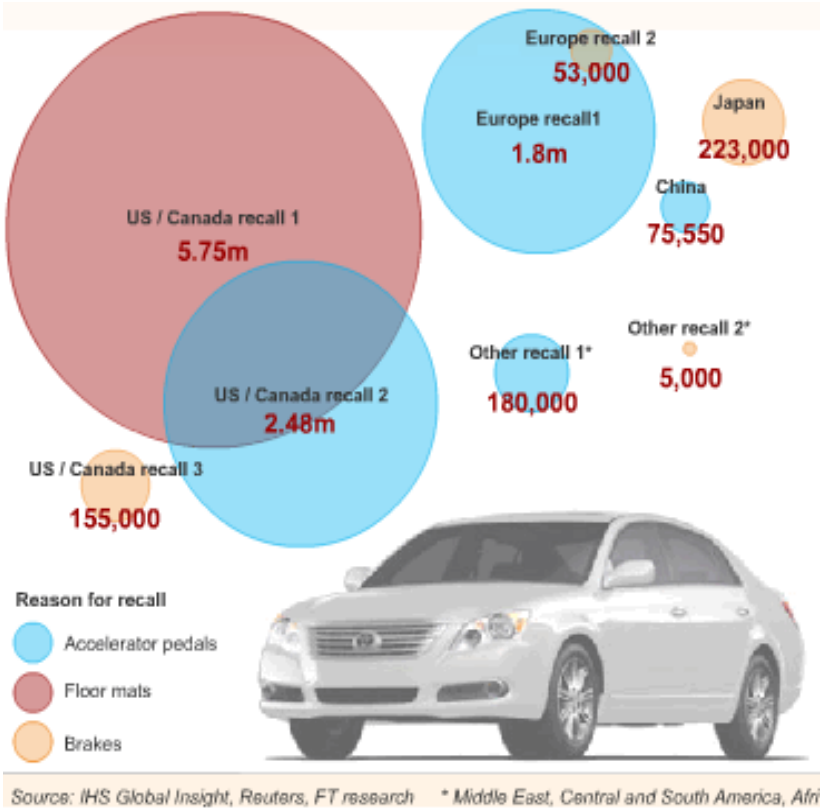
Milos Gligoric

gliga@Illinois.edu

cs427

11/04/2014

NSF CCF 10-12759

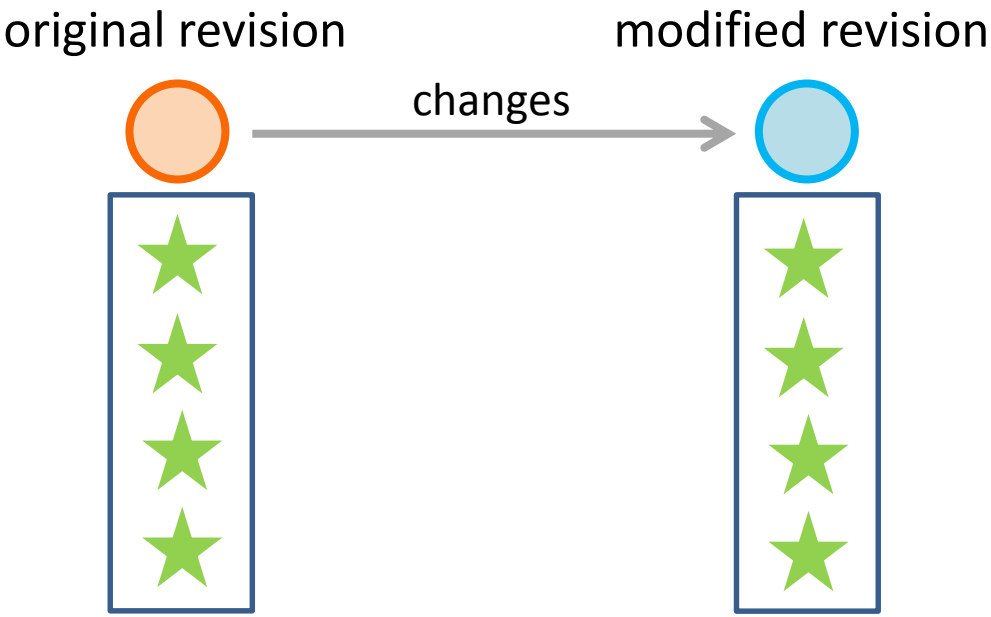


Toyota - total number of recalls = 8.6mio worldwide

FDA: Software Responsible for 24% Of All Medical Device Failures

Regression Testing

- Widely used in industry
- Executes tests for each new revision
- Checks if changes broke something



Regression Testing Cost

~5min

guava-libraries

Guava: Google Core Libraries for Java 1.6+

~15min

jetty://

~45min

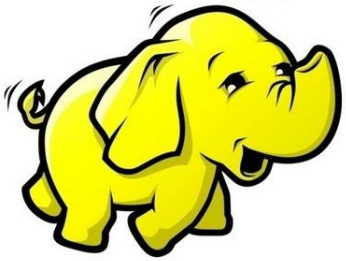
continuum™

~45min

~45min

~4h

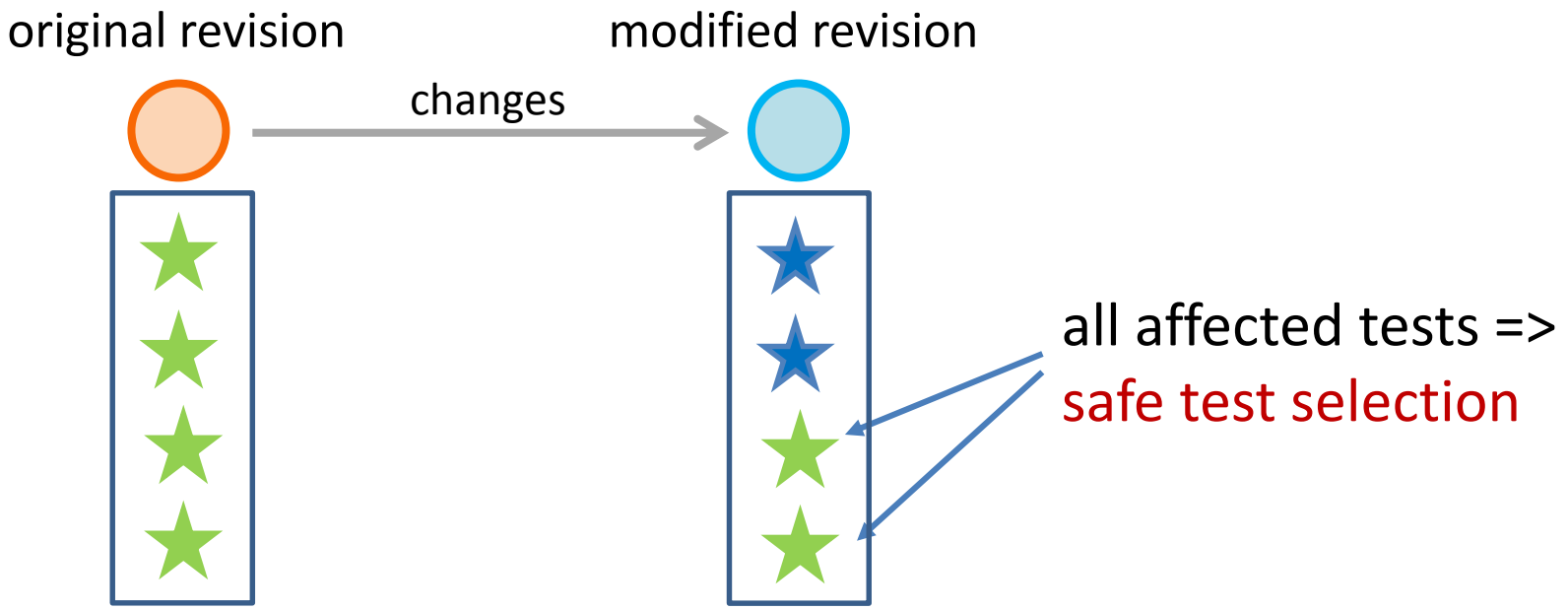
hadoop



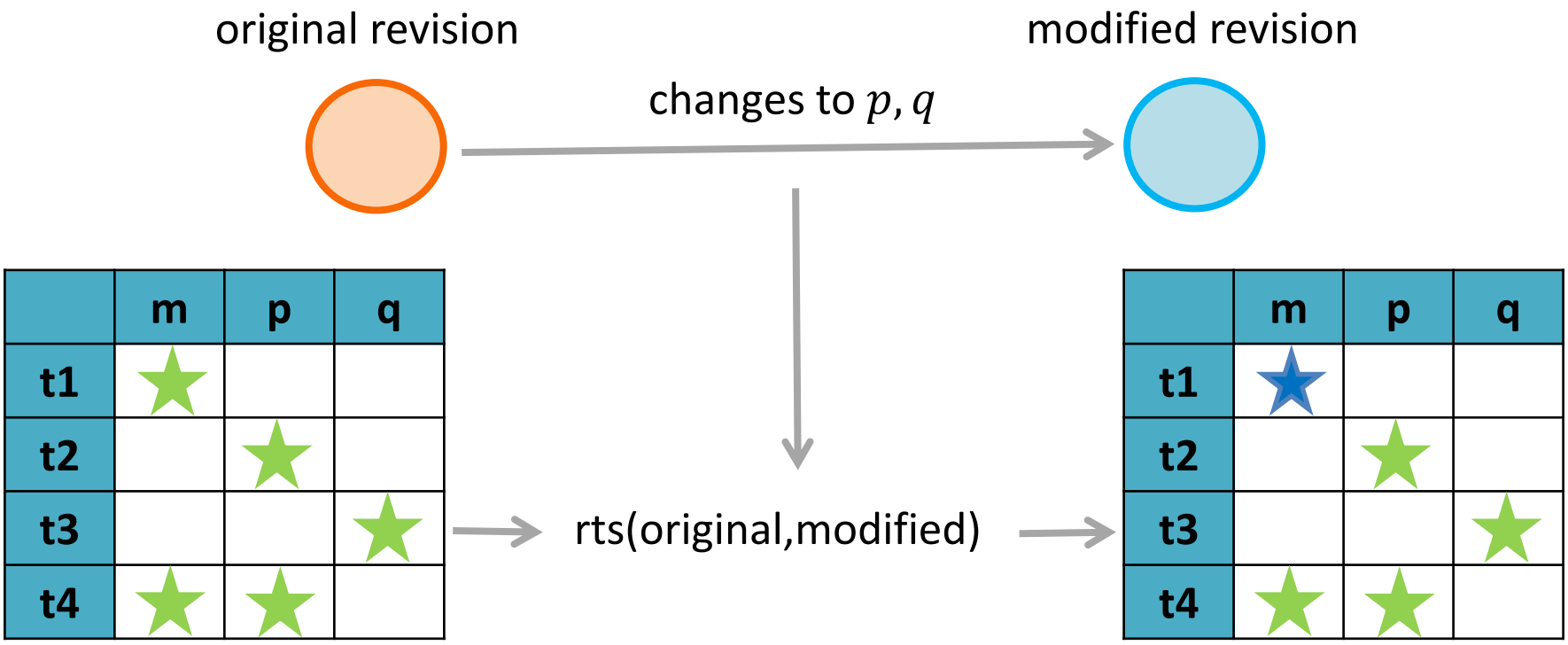
~17h

Regression Test Selection (RTS)

- Optimizes regression testing
- Analyzes changes to a codebase
- Runs only tests whose behavior may be affected



Regression Test Selection (Example)

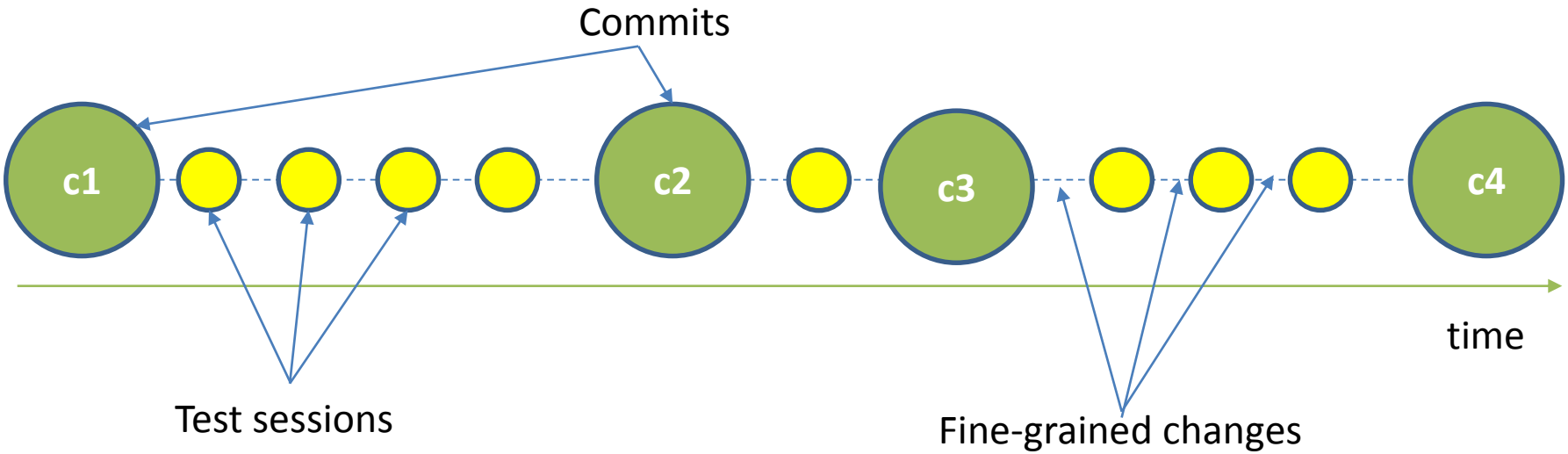


Current Practice

- Few systems used in practice: Google TAP
  - Mapping of tests based on dependencies across projects
  - Not applicable to day-to-day work within single project
- No widely adoptable automated RTS tool after ~30 years of research
- Developers' options:
  - RetestAll (expensive) or manual RTS (imprecise)

Real Time Data

- Data was captured using a record-and-replay tool, CodingTracker
- Data had info not just on commits, but also on test sessions (runs of 1 or more tests)
- Live data allowed us to study manual RTS



Study Setup

- 14 developers working on 17 projects
- 3 months study
- 918 hours of development, 5757 test sessions, 264,562 executed tests
- 5 professional programmers, 9 UIUC students

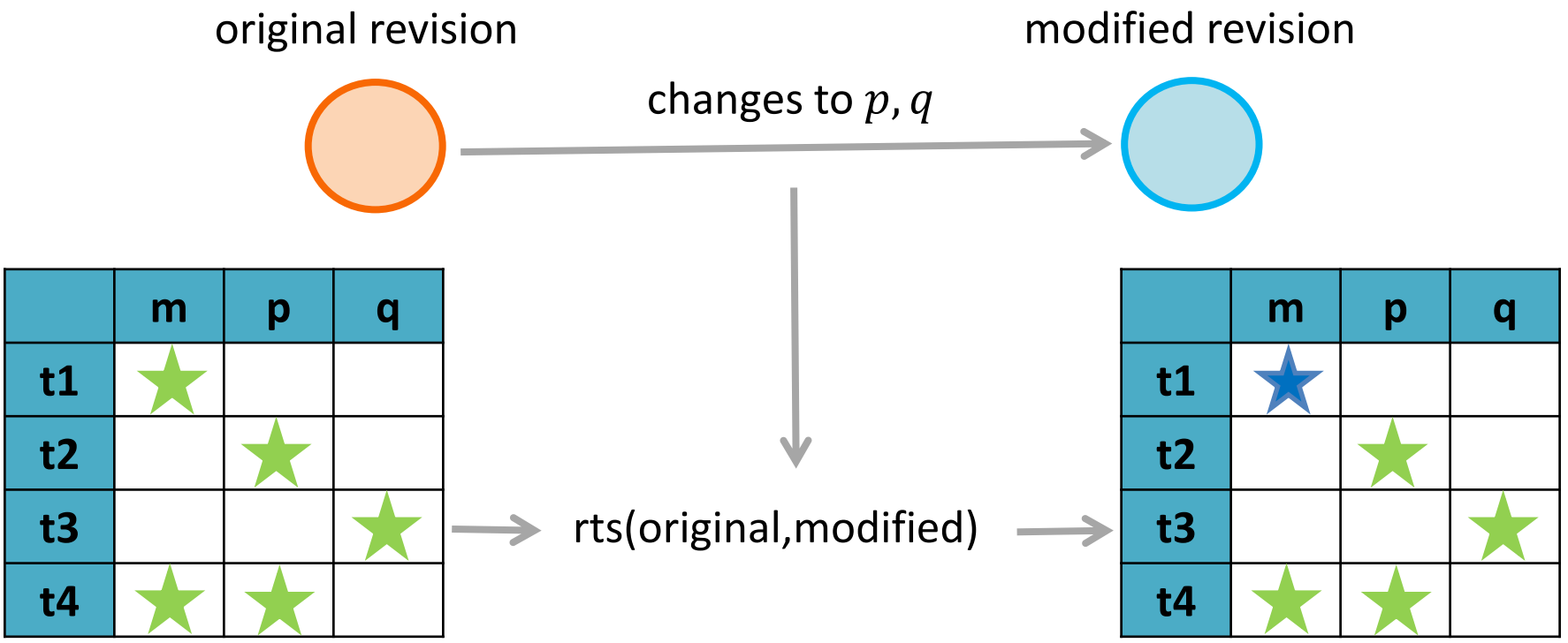
Manual vs. Automated RTS

- Precision and safety
- ~70% of the time, Manual RTS > Auto RTS
  - Potentially wasting time
- ~30% of the time, Manual RTS < Auto RTS
  - Potentially missing faults

THE WORLD NEEDS REGRESSION TEST SELECTION

- **Tests are taking a lot of time**
- **Developers are doing a poor job with Manual RTS**

RTS: Execution vs. Overhead

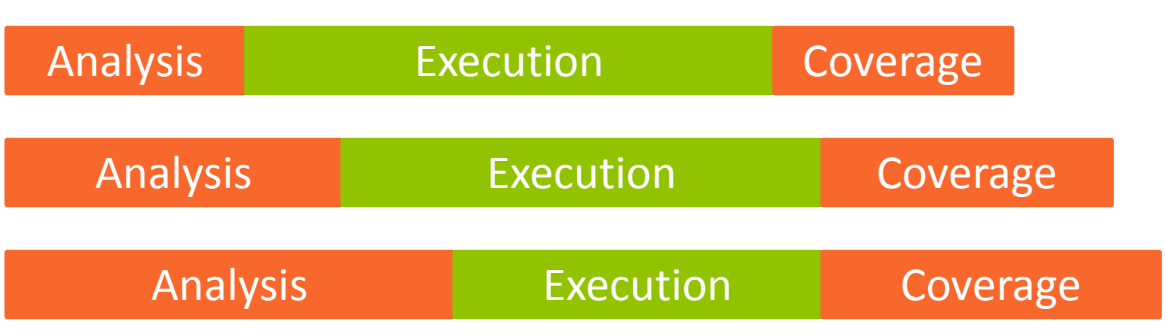


Analysis | Execution of Selected | Coverage

Analysis | Coverage >? Execution of non-Selected

Fine-grained Mappings

- Mapping from test to various code elements
  - edge in CFG, method, basic block, statement, ….

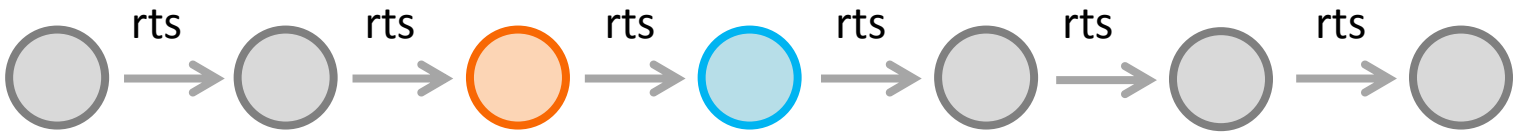


Implementation Complexity

Integration Complexity

Linear Software Histories

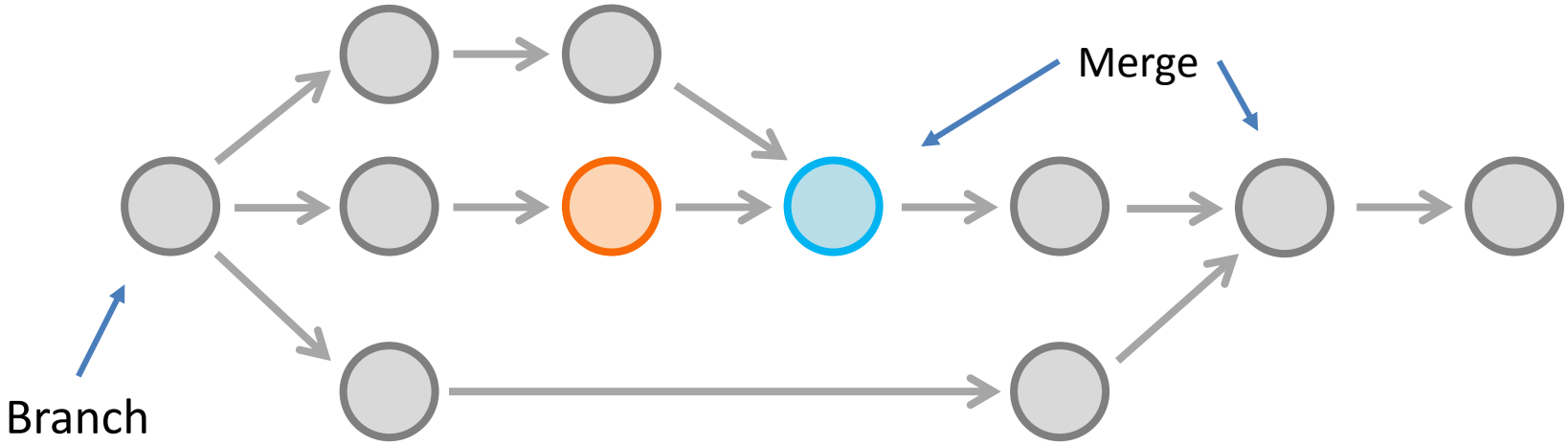
- Traditional test selection
  - Two revisions of code at a time
- Easy to extend to a linear sequence



  - Centralized version control systems (CVS, SVN, etc.)

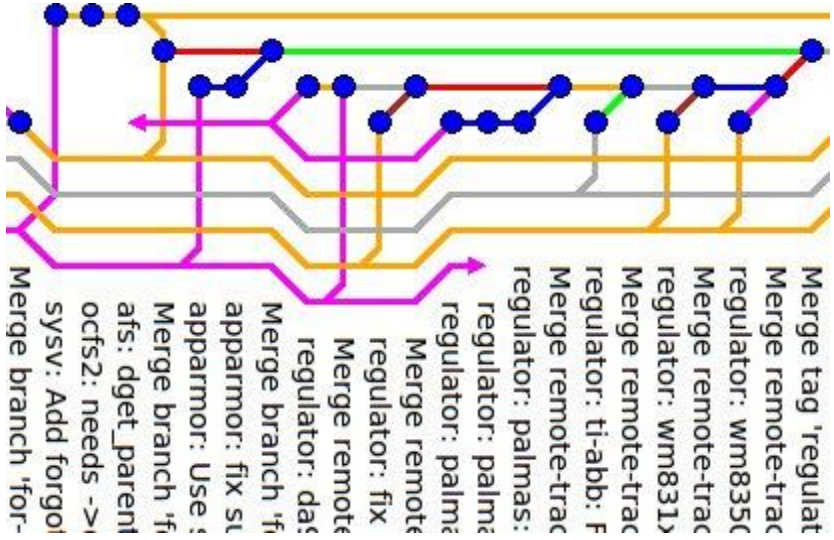
Distributed Software Histories

- Distributed version control systems (e.g., Git)
- Complex graphs created by branching, merging, etc.



How to extend rts for distributed software histories?

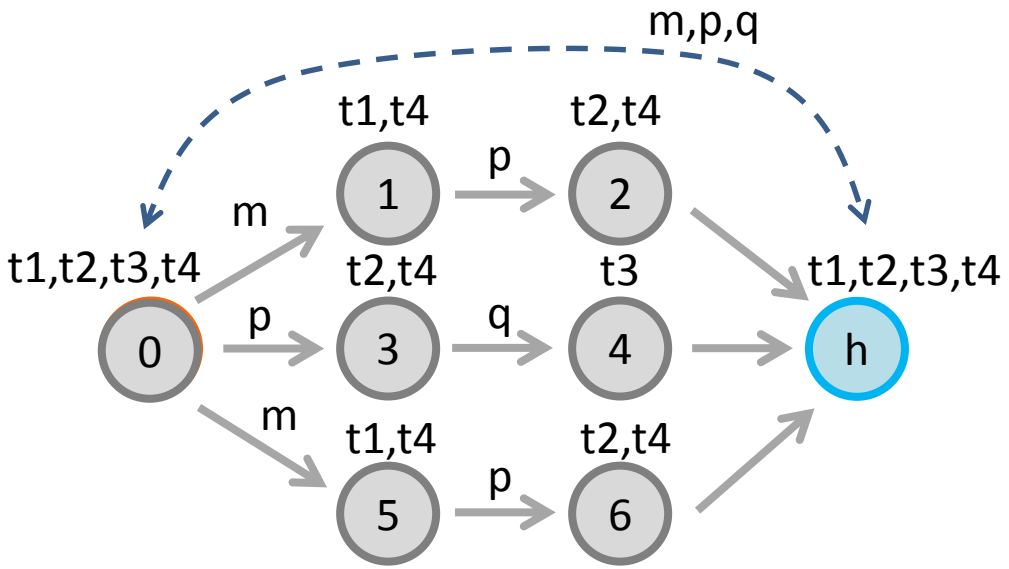
Frequent Branching and Merging



- Analyzed 27 open-source projects on GitHub
- 30% of revisions are merges

Test Selection for Merge Command

	m	p	q
t1	★		
t2		★	
t3			★
t4	★	★	



$$S^1_{merge}(h) = rts(imd(h), h)$$

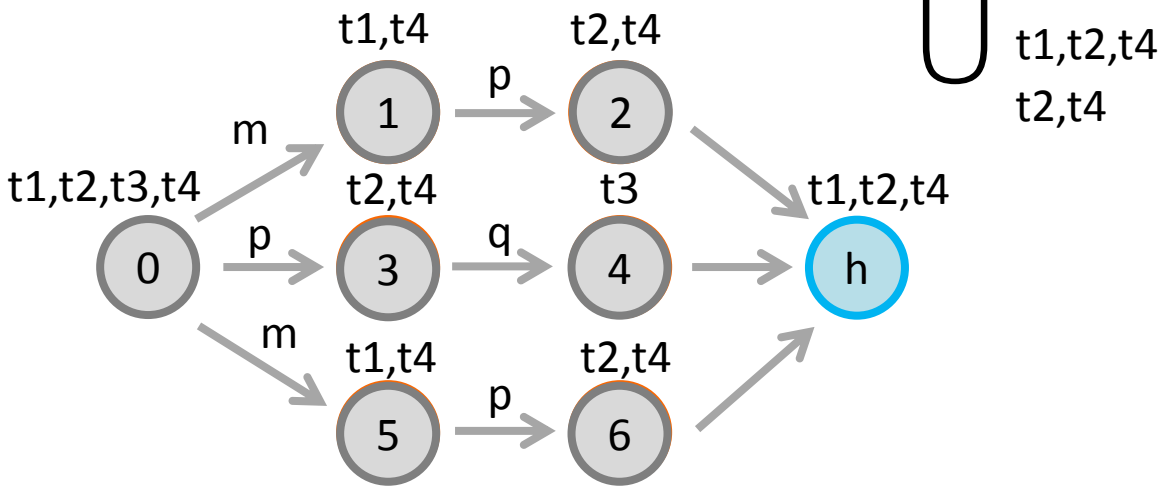
imd – immediate dominator

Pro: Runs traditional test selection only once (i.e., fast)

Con: There may be many changes between imd(h) and h =>
many tests selected to run (i.e., slow)

Merge Command (Option S^0)

	m	p	q
t1	★		
t2		★	
t3			★
t4	★	★	



If a test is affected on multiple branches, changes from different branches together may lead to diff result

Pro: Does not run traditional test selection, but uses history results

Con: Selects more tests than S^k (e.g., new tests in one of the branches)

$$S^0_{merge}(h) = S_{aff}(h) \cup \left(A(h) \setminus \bigcap_{p \in pred(h)} A(p) \right)$$

$$S_{aff}(h) = \bigcup_{p,p' \in pred(h), p \neq p', d = dom(p,p')} \left(\bigcup_{n \in d \leq * p \setminus \{d\}} S_{sel}(n) \right) \cap \left(\bigcup_{n \in d \leq * p' \setminus \{d\}} S_{sel}(n) \right)$$