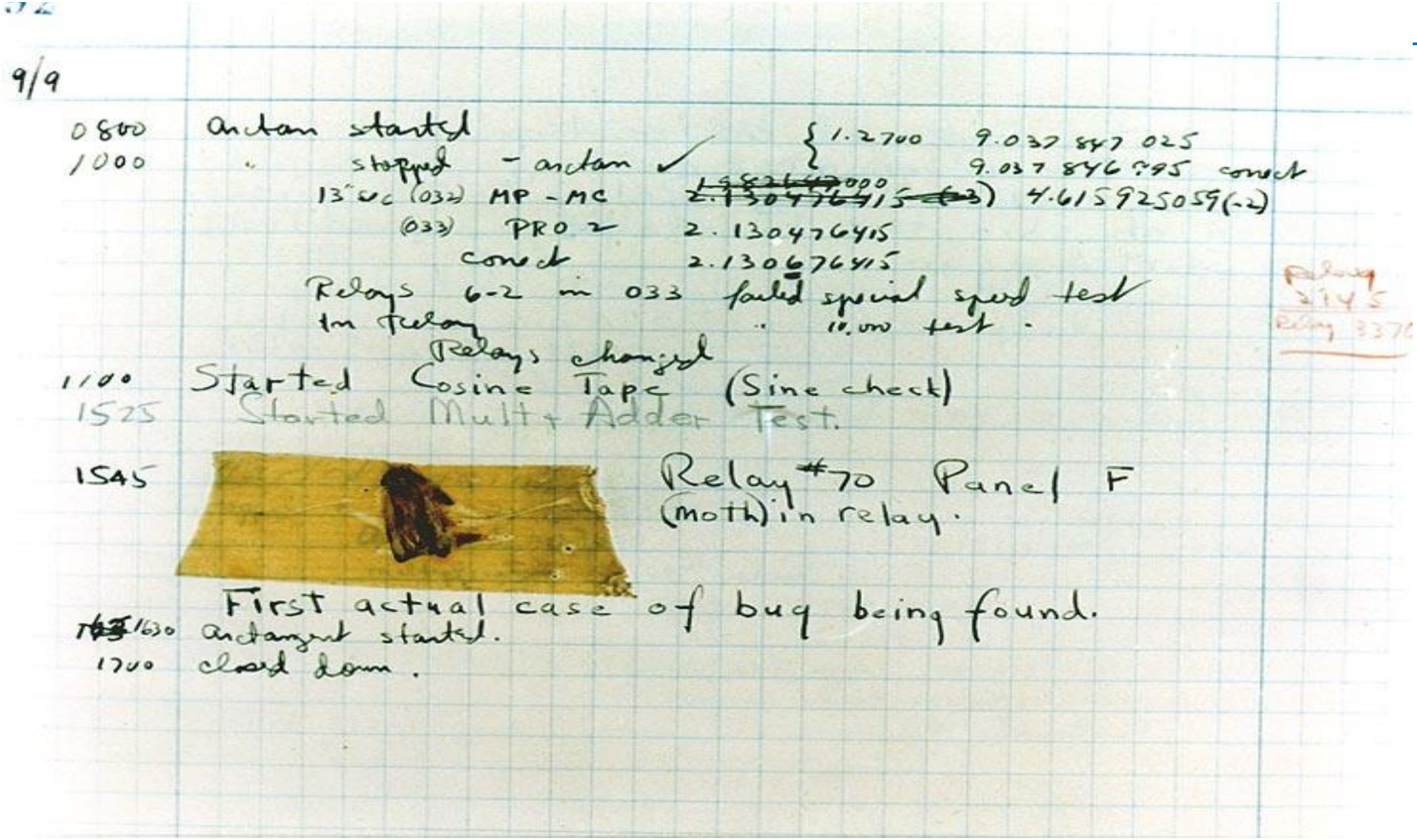
Mark II logbook, Sep 9, 1947



Debugging

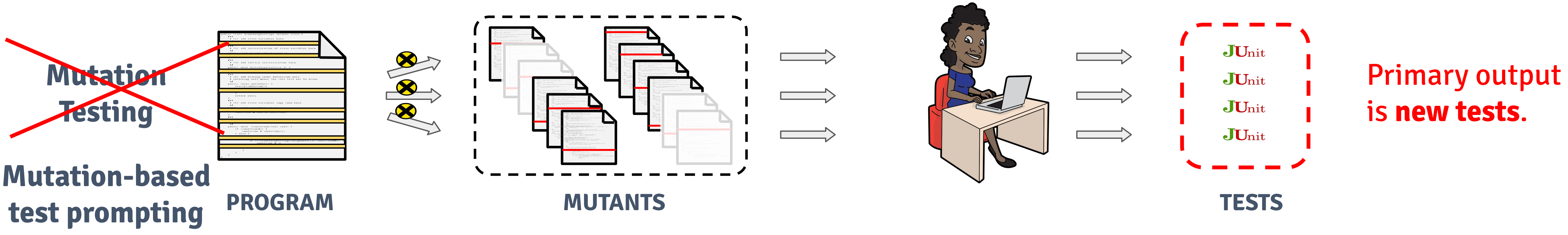
UW CSE P 504

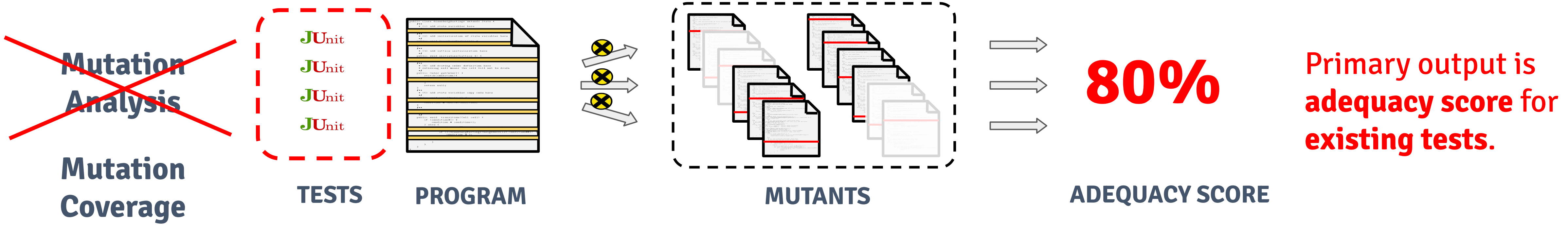
Today’s outline

- Recap of coverage- and mutation-based testing
- Debugging basics
- Delta debugging technique
- Demo
- In-class activity: delta debugging

Background Reading:
Simplifying and Isolating Failure-Inducing Input, Zeller and Hildebandt, 2002

Mutation test prompting vs. mutation coverage





How expensive is mutation testing?

Is the mutation score meaningful?

Discussion: Coverage- and mutation-based testing

- Coverage-based testing
  - Any questions?
  - What are the pros and cons?
  - Will you test differently in the future?
- Mutation-based testing: open discussion
  - Any questions?
  - What have you observed and learned?
  - What are the pros and cons?
    - Example challenge: mutant comprehension vs. test creation
  - Do you feel it is worth using?

A Bug's Life

Defect – mistake committed by a human

Error – incorrect computation

Failure – visible error: program violates its specification

Debugging starts when a failure is observed

- Unit testing
- Integration testing
- In the field

Goal of debugging: go *from failure back to defect*

Ways to get your code right

- Design & verification
  - Prevent defects from appearing in the first place
- Defensive programming
  - Program with debugging in mind: fail fast
- Testing & validation
  - Uncover problems (even in spec), increase confidence
- Debugging
  - Find out why a program is not functioning as intended
- Testing ≠ debugging
  - test: reveals existence of problem (failure)
  - debug: pinpoint location + cause of problem (defect)

Defense in depth

1. Make errors impossible
   A (memory-)managed language prevents type errors, memory corruption
2. Don't introduce defects
   Correctness: get things right the first time
3. Make errors immediately visible
   Example: assertions; **`checkRep()`**
   Converts an error to a failure; reduces distance from defect to failure
4. Debugging is the <u>last resort</u>
   Work from effect (failure) to cause (defect)
   Scientific method: Design experiments to gain information about the defect
   Easiest in a modular program with good specs and test suites

Debugging and the scientific method

Debugging must be systematic

- Carefully decide what to do (avoid fruitless avenues)
- Record everything that you do (actions and results)
  - Can replicate previous work
  - Or avoid the need to do so

Iterative scientific process:

Should have explanatory power
Not just “Maybe line 3 is wrong”

Formulate a hypothesis

Design an experiment

Should investigate cause

Perform the experiment

Interpret results

The typical debugging process

- **Identify** – it’s a bug, not a feature
- **Reproduce** – what are the inputs and conditions causing the error
- **Test** – create a (minimal) **regression test** to illustrate the issue
- **Localize** – locate the problematic code
- **Fix** the code
- **Validate** - run the regression test and the original failing scenario

A good bug report (issue)

A bug report should contain:

- How to **reproduce**, including context. (What is “context”?)
  - Preferably commands that can be cut-and-pasted into a shell
    - Starting with `git clone`
  - Don’t use vague English; provide an exact command
- All inputs
- Full output
  - No screenshots of textual output
- What you expected; why the behavior is wrong

A test case should be as simple as possible (“**minimized**”)

- Why?

A test case should be as simple as possible

Why?

A test case should be as simple as possible

Why?

- Helps to localize the defect
  - fewer lines and features are exercised
- Speeds up the edit-compile-test cycle
- You need a small regression test

Quotation from Zeller & Hildebrandt 2002

Often people who encounter a bug spend a lot of time investigating which changes to the input file will make the bug go away and which changes will not affect it.

— Richard Stallman, Using and Porting GNU CC

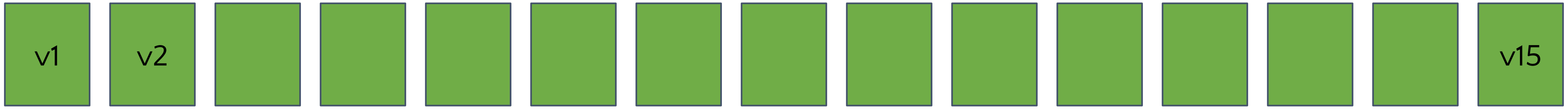
Quotation from Zeller & Hildebrandt 2002

Often people who encounter a bug spend a lot of time investigating which changes to the input file will make the bug go away and which changes will not affect it. This is often time consuming and not very useful, because the way we will find the bug is by running a single example under the debugger with breakpoints, not by pure deduction from a series of examples. We recommend that you save your time for something else.

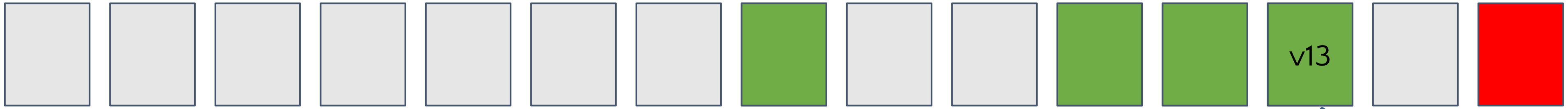
— Richard Stallman, Using and Porting GNU CC

Binary search (e.g., git bisect)

Continuous integration runs:



Add a new test case:



Binary search is not guaranteed to reproduce the original failure.

=> You might not fix the defect that caused that failure.

v13 might have failed in a different way,

for a different reason.

Did this use the scientific method?

Delta Debugging

A debugging technique
to create a minimal test case
that fails *in the same way*.

Input:

- Program
- Failing test case

Output:

- Failing test case that is as small as possible

This is a crashing test case

```
<td align=left valign=top>
<SELECT NAME="op sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All
<OPTION VALUE="Windows 3.1">Windows 3.1
<OPTION VALUE="Windows 95">Windows 95
<OPTION VALUE="Windows 98">Windows 98
<OPTION VALUE="Windows ME">Windows ME
<OPTION VALUE="Windows 2000">Windows 2000
<OPTION VALUE="Windows NT">Windows NT
<OPTION VALUE="Mac System 7">Mac System 7
<OPTION VALUE="Mac System 7.5">Mac System 7.5
<OPTION VALUE="Mac System 7.6.1">Mac System 7.6.1
<OPTION VALUE="Mac System 8.0">Mac System 8.0
<OPTION VALUE="Mac System 8.5">Mac System 8.5
<OPTION VALUE="Mac System 8.6">Mac System 8.6
<OPTION VALUE="Mac System 9.x">Mac System 9.x
<OPTION VALUE="MacOS X">MacOS X
<OPTION VALUE="Linux">Linux
<OPTION VALUE="BSDI">BSDI
<OPTION VALUE="FreeBSD">FreeBSD
<OPTION VALUE="NetBSD">NetBSD
<OPTION VALUE="OpenBSD">OpenBSD
<OPTION VALUE="AIX">AIX
<OPTION VALUE="BeOS">BeOS
<OPTION VALUE="HP-UX">HP-UX
<OPTION VALUE="IRIX">IRIX
<OPTION VALUE="Neutrino">Neutrino
<OPTION VALUE="OpenVMS">OpenVMS
<OPTION VALUE="OS/2">OS/2
<OPTION VALUE="OSF/1">OSF/1
<OPTION VALUE="Solaris">Solaris
<OPTION VALUE="SunOS">SunOS
<OPTION VALUE="other">other</SELECT></td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION
VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION
VALUE="P5">P5</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="bug severity" MULTIPLE SIZE=7>
<OPTION VALUE="blocker">blocker<OPTION VALUE="critical">critical<OPTION
VALUE="major">major<OPTION
VALUE="normal">normal<OPTION VALUE="minor">minor<OPTION
VALUE="trivial">trivial<OPTION VALUE="enhancement">enhancement</SELECT>
</tr>
</table>
```

- Crashed Mozilla
- Consider 370 of these being filed!
- How would you debug the problem?

This is a crashing test case

```
<td align=left valign=top>
<SELECT NAME="op sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All
<OPTION VALUE="Windows 3.1">Windows 3.1
<OPTION VALUE="Windows 95">Windows 95
<OPTION VALUE="Windows 98">Windows 98
<OPTION VALUE="Windows ME">Windows ME
<OPTION VALUE="Windows 2000">Windows 2000
<OPTION VALUE="Windows NT">Windows NT
<OPTION VALUE="Mac System 7">Mac System 7
<OPTION VALUE="Mac System 7.5">Mac System 7.5
<OPTION VALUE="Mac System 7.6.1">Mac System 7.6.1
<OPTION VALUE="Mac System 8.0">Mac System 8.0
<OPTION VALUE="Mac System 8.5">Mac System 8.5
<OPTION VALUE="Mac System 8.6">Mac System 8.6
<OPTION VALUE="Mac System 9.x">Mac System 9.x
<OPTION VALUE="MacOS X">MacOS X
<OPTION VALUE="Linux">Linux
<OPTION VALUE="BSDI">BSDI
<OPTION VALUE="FreeBSD">FreeBSD
<OPTION VALUE="NetBSD">NetBSD
<OPTION VALUE="OpenBSD">OpenBSD
<OPTION VALUE="AIX">AIX
<OPTION VALUE="BeOS">BeOS
<OPTION VALUE="HP-UX">HP-UX
<OPTION VALUE="IRIX">IRIX
<OPTION VALUE="Neutrino">Neutrino
<OPTION VALUE="OpenVMS">OpenVMS
<OPTION VALUE="OS/2">OS/2
<OPTION VALUE="OSF/1">OSF/1
<OPTION VALUE="Solaris">Solaris
<OPTION VALUE="SunOS">SunOS
<OPTION VALUE="other">other</SELECT></td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION
VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION
VALUE="P5">P5</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="bug severity" MULTIPLE SIZE=7>
<OPTION VALUE="blocker">blocker<OPTION VALUE="critical">critical<OPTION
VALUE="major">major<OPTION
VALUE="normal">normal<OPTION VALUE="minor">minor<OPTION
VALUE="trivial">trivial<OPTION VALUE="enhancement">enhancement</SELECT>
</tr>
</table>
```

- Crashed Mozilla
- Consider 370 of these being filed!
- How would you debug the problem?
- What content is sufficient to reproduce the failure?

This is a crashing test case

```
<td align=left valign=top>
<SELECT NAME="op sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All
<OPTION VALUE="Windows 3.1">Windows 3.1
<OPTION VALUE="Windows 95">Windows 95
<OPTION VALUE="Windows 98">Windows 98
<OPTION VALUE="Windows ME">Windows ME
<OPTION VALUE="Windows 2000">Windows 2000
<OPTION VALUE="Windows NT">Windows NT
<OPTION VALUE="Mac System 7">Mac System 7
<OPTION VALUE="Mac System 7.5">Mac System 7.5
<OPTION VALUE="Mac System 7.6.1">Mac System 7.6.1
<OPTION VALUE="Mac System 8.0">Mac System 8.0
<OPTION VALUE="Mac System 8.5">Mac System 8.5
<OPTION VALUE="Mac System 8.6">Mac System 8.6
<OPTION VALUE="Mac System 9.x">Mac System 9.x
<OPTION VALUE="MacOS X">MacOS X
<OPTION VALUE="Linux">Linux
<OPTION VALUE="BSDI">BSDI
<OPTION VALUE="FreeBSD">FreeBSD
<OPTION VALUE="NetBSD">NetBSD
<OPTION VALUE="OpenBSD">OpenBSD
<OPTION VALUE="AIX">AIX
<OPTION VALUE="BeOS">BeOS
<OPTION VALUE="HP-UX">HP-UX
<OPTION VALUE="IRIX">IRIX
<OPTION VALUE="Neutrino">Neutrino
<OPTION VALUE="OpenVMS">OpenVMS
<OPTION VALUE="OS/2">OS/2
<OPTION VALUE="OSF/1">OSF/1
<OPTION VALUE="Solaris">Solaris
<OPTION VALUE="SunOS">SunOS
<OPTION VALUE="other">other</SELECT></td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION
VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION
VALUE="P5">P5</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="bug severity" MULTIPLE SIZE=7>
<OPTION VALUE="blocker">blocker<OPTION VALUE="critical">critical<OPTION
VALUE="major">major<OPTION
VALUE="normal">normal<OPTION VALUE="minor">minor<OPTION
VALUE="trivial">trivial<OPTION VALUE="enhancement">enhancement</SELECT>
</tr>
</table>
```

- Crashed Mozilla
- A minimal test case is: **<SELECT>**

This is a crashing test case

```
<td align=left valign=top>
<SELECT NAME="op sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All
<OPTION VALUE="Windows 3.1">Windows 3.1
<OPTION VALUE="Windows 95">Windows 95
<OPTION VALUE="Windows 98">Windows 98
<OPTION VALUE="Windows ME">Windows ME
<OPTION VALUE="Windows 2000">Windows 2000
<OPTION VALUE="Windows NT">Windows NT
<OPTION VALUE="Mac System 7">Mac System 7
<OPTION VALUE="Mac System 7.5">Mac System 7.5
<OPTION VALUE="Mac System 7.6.1">Mac System 7.6.1
<OPTION VALUE="Mac System 8.0">Mac System 8.0
<OPTION VALUE="Mac System 8.5">Mac System 8.5
<OPTION VALUE="Mac System 8.6">Mac System 8.6
<OPTION VALUE="Mac System 9.x">Mac System 9.x
<OPTION VALUE="MacOS X">MacOS X
<OPTION VALUE="Linux">Linux
<OPTION VALUE="BSDI">BSDI
<OPTION VALUE="FreeBSD">FreeBSD
<OPTION VALUE="NetBSD">NetBSD
<OPTION VALUE="OpenBSD">OpenBSD
<OPTION VALUE="AIX">AIX
<OPTION VALUE="BeOS">BeOS
<OPTION VALUE="HP-UX">HP-UX
<OPTION VALUE="IRIX">IRIX
<OPTION VALUE="Neutrino">Neutrino
<OPTION VALUE="OpenVMS">OpenVMS
<OPTION VALUE="OS/2">OS/2
<OPTION VALUE="OSF/1">OSF/1
<OPTION VALUE="Solaris">Solaris
<OPTION VALUE="SunOS">SunOS
<OPTION VALUE="other">other</SELECT></td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION
VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION
VALUE="P5">P5</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="bug severity" MULTIPLE SIZE=7>
<OPTION VALUE="blocker">blocker<OPTION VALUE="critical">critical<OPTION
VALUE="major">major<OPTION
VALUE="normal">normal<OPTION VALUE="minor">minor<OPTION
VALUE="trivial">trivial<OPTION VALUE="enhancement">enhancement</SELECT>
</tr>
</table>
```

- Crashed Mozilla
- A minimal test case is: **<SELECT>**
- Can we automate the process of minimizing test cases?
- Idea: use binary search

Try the first half of the input

```
<td align=left valign=top>
<SELECT NAME="op sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All
<OPTION VALUE="Windows 3.1">Windows 3.1
<OPTION VALUE="Windows 95">Windows 95
<OPTION VALUE="Windows 98">Windows 98
<OPTION VALUE="Windows ME">Windows ME
<OPTION VALUE="Windows 2000">Windows 2000
<OPTION VALUE="Windows NT">Windows NT
<OPTION VALUE="Mac System 7">Mac System 7
<OPTION VALUE="Mac System 7.5">Mac System 7.5
<OPTION VALUE="Mac System 7.6.1">Mac System 7.6.1
<OPTION VALUE="Mac System 8.0">Mac System 8.0
<OPTION VALUE="Mac System 8.5">Mac System 8.5
<OPTION VALUE="Mac System 8.6">Mac System 8.6
<OPTION VALUE="Mac System 9.x">Mac System 9.x
<OPTION VALUE="MacOS X">MacOS X
<OPTION VALUE="Linux">Linux
<OPTION VALUE="BSDI">BSDI
<OPTION VALUE="FreeBSD">FreeBSD
<OPTION VALUE="NetB
```

- Crashed Mozilla
- A minimal test case is: **<SELECT>**
- Can we automate the process of minimizing test cases?
- Idea: use binary search
- What is the result of the test?

Delta Debugging

A debugging technique
to create a minimal test case
that fails *in the same way*.

Input:

- Program
- Failing test case
-

Output:

- Failing test case that is as small as possible

Delta Debugging

A debugging technique
to create a minimal test case
that fails *in the same way*.

Input:
- Program
- Failing test case
- Predicate on executions:
  did the execution fail in the same way?

Test passes => false
Test fails in the same way => true
Test fails in some other way => false

Output:
- Failing test case that is as small as possible

Minimizing test cases

Test case	Test case	Test case
Failing	**Passing**	**Passing**

Goal: minimize the failing test case (remove some lines)

The happy path: binary search

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

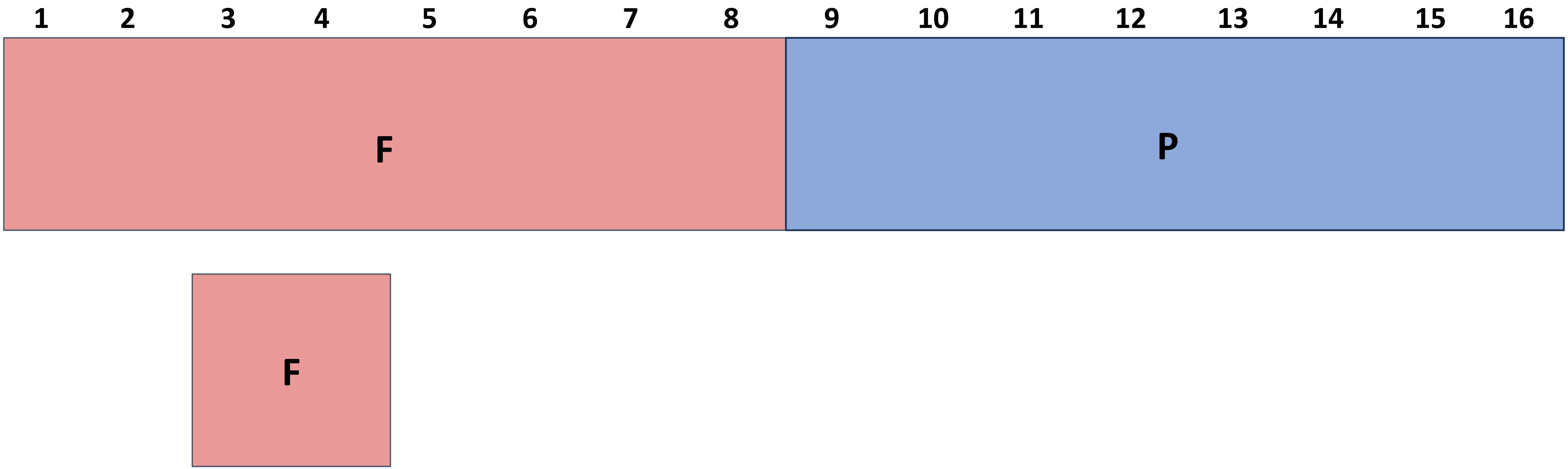
F

F

Failing test with 16 lines

The minimal failing test has 2 lines: 3 and 4

The happy path: binary search



The happy path: binary search

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

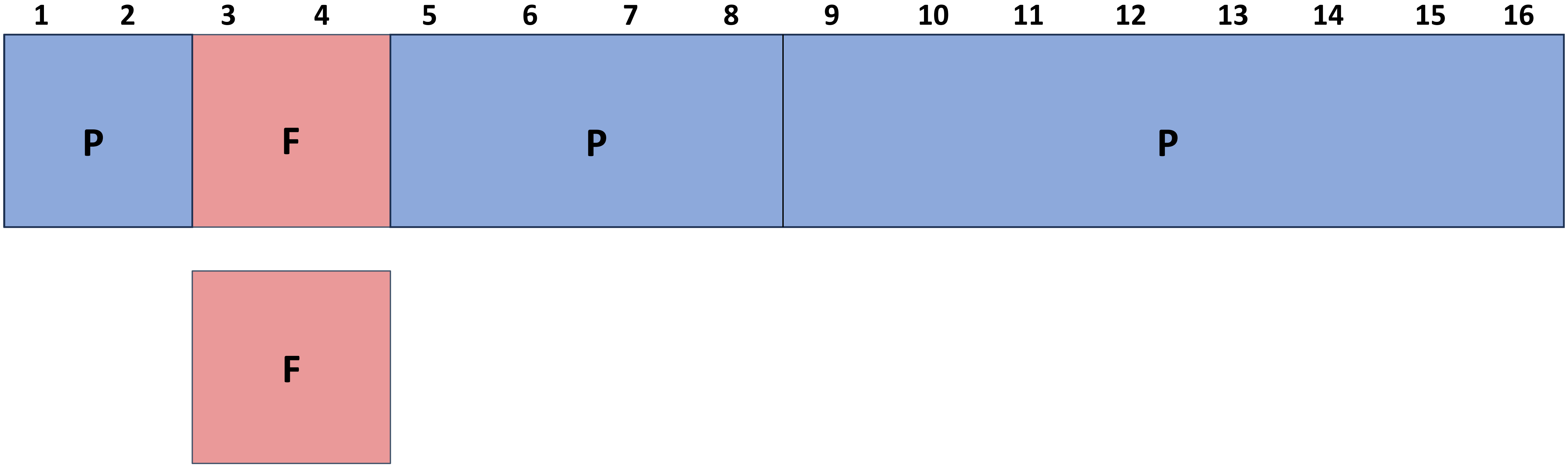
F

P

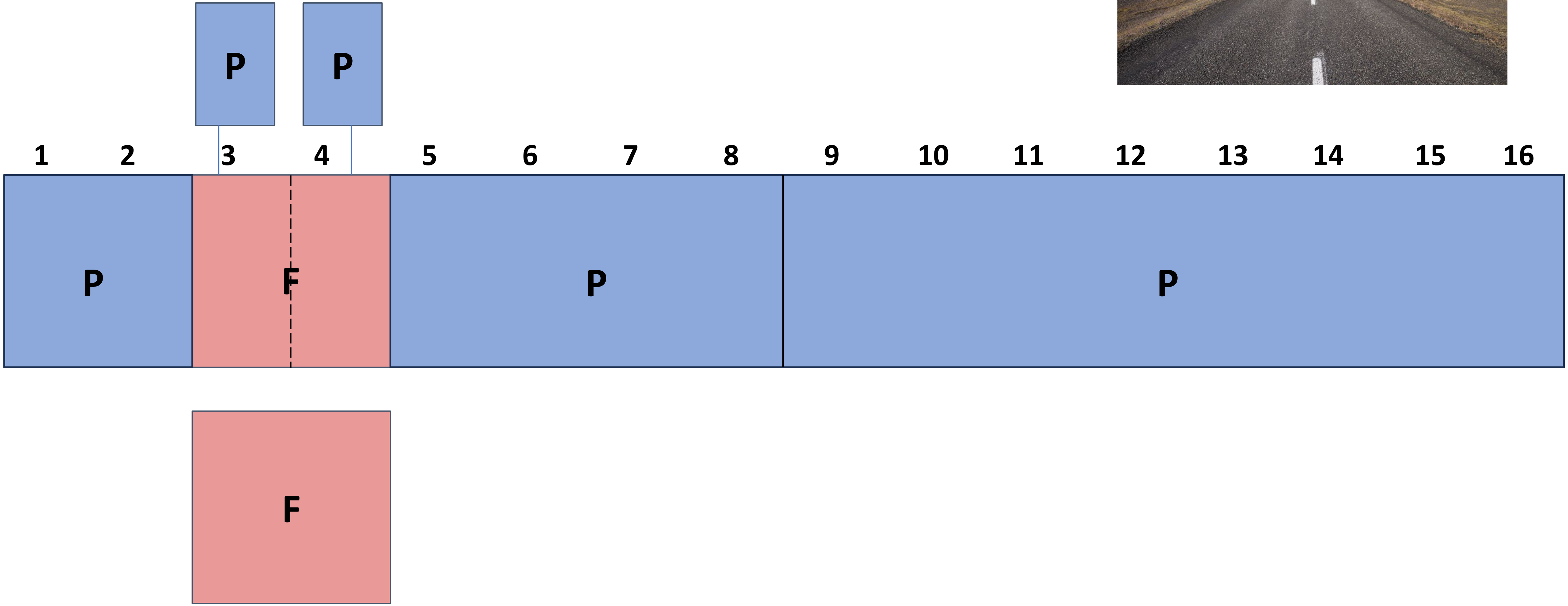
P

F

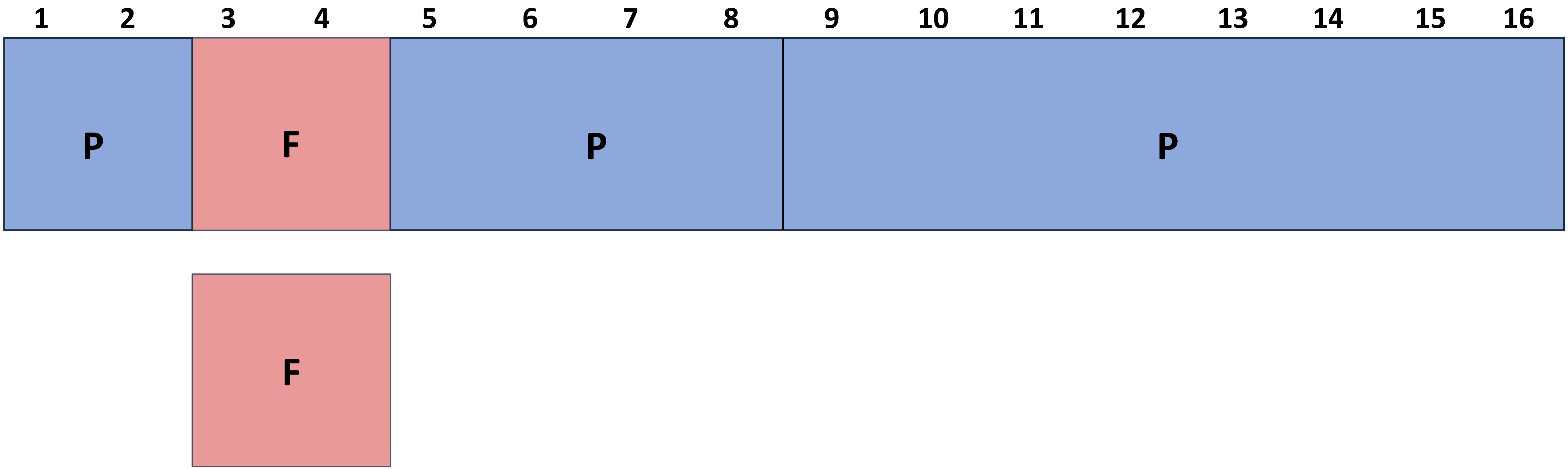
The happy path: binary search



The happy path: binary search

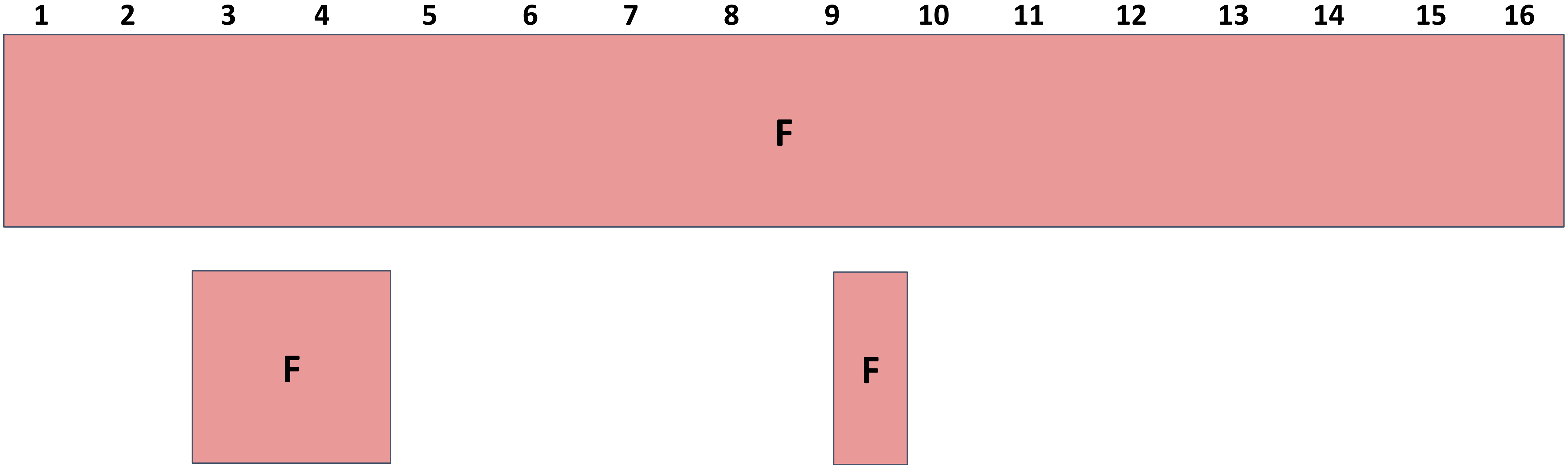


The happy path: binary search



Successfully minimized the failing test to 2 lines

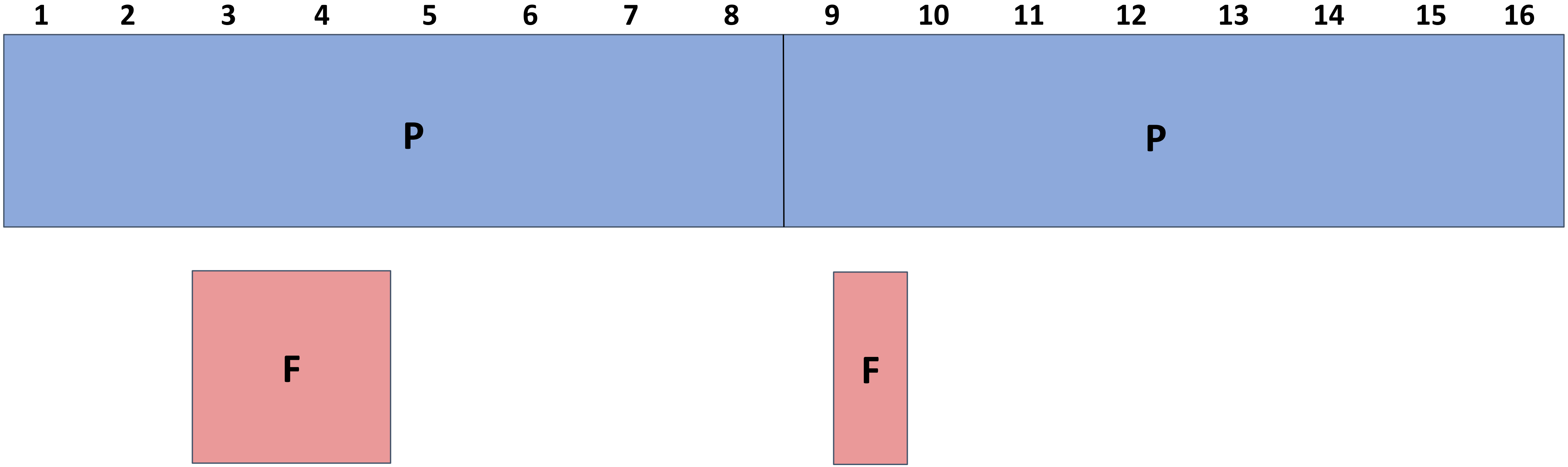
The not so happy path...



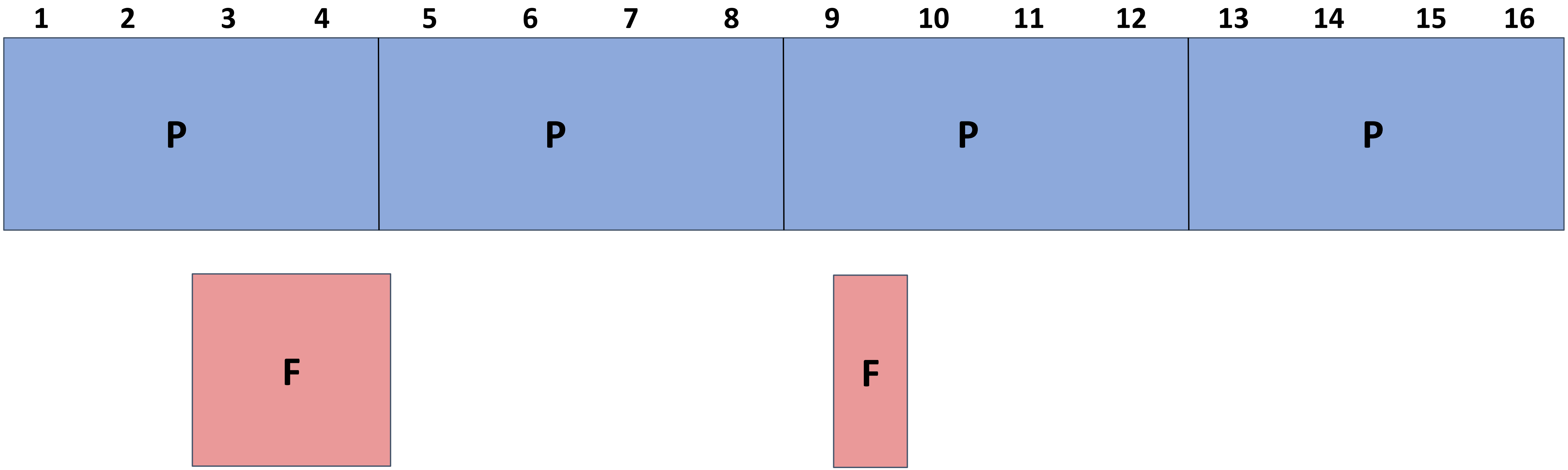
Failing test with 16 lines

The minimal failing test has 3 lines: 3, 4, and 9

The not so happy path...



The not so happy path...



Binary search is no help

Delta Debugging = binary search
+ handle subsets
+ account for multiple types of test outcomes

See paper:
Simplifying and Isolating Failure-Inducing Input
Zeller and Hildebandt, 2002

The Delta Debugging algorithm

Three basic phases (in a loop):

1. Test each subset*
   (= binary subdivision)
2. Test each complement*
3. Increase granularity
   (double the # of subsets)

*On success, reduce the # of subsets (don't continue testing)

Complement example:
Input = 1, 2, 3, 4
A subset is { 1 }
Its complement is { 2, 3, 4 }

Minimizing Delta Debugging Algorithm

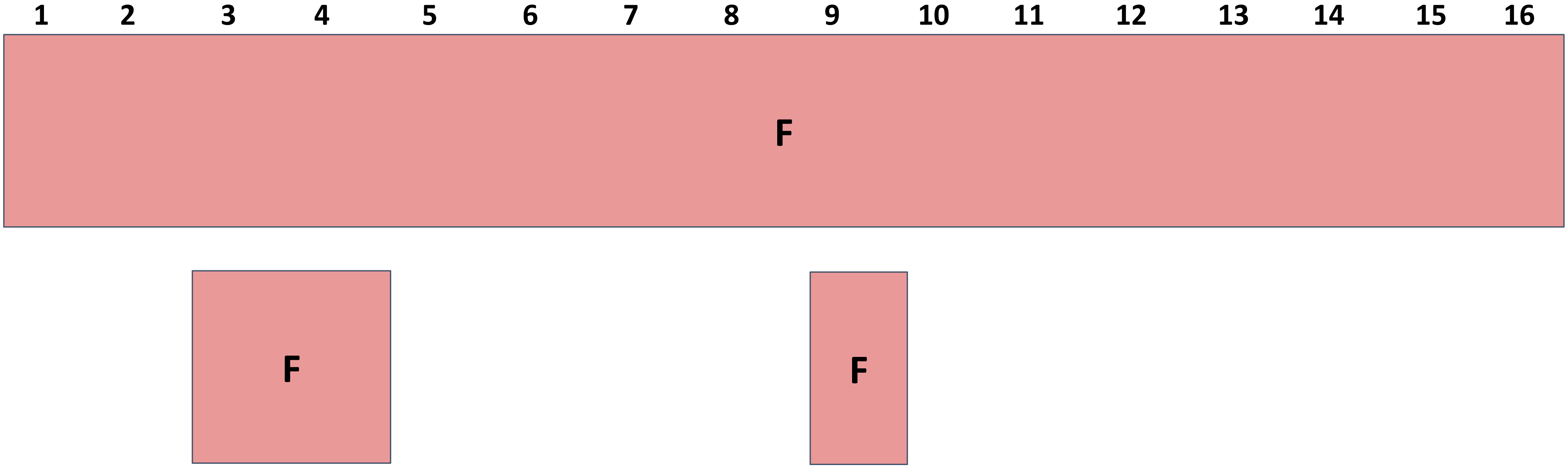
Let *test* and $c_{\text{✘}}$ be given such that $test(\emptyset) = \text{✔} \wedge test(c_{\text{✘}}) = \text{✘}$ hold.
The goal is to find $c'_{\text{✘}} = ddmin(c_{\text{✘}})$ such that $c'_{\text{✘}} \subseteq c_{\text{✘}}$, $test(c'_{\text{✘}}) = \text{✘}$, and $c'_{\text{✘}}$ is 1-minimal.
The *minimizing Delta Debugging algorithm* $ddmin(c)$ is

$$ddmin(c_{\text{✘}}) = ddmin_2(c_{\text{✘}}, 2) \quad \text{where}$$

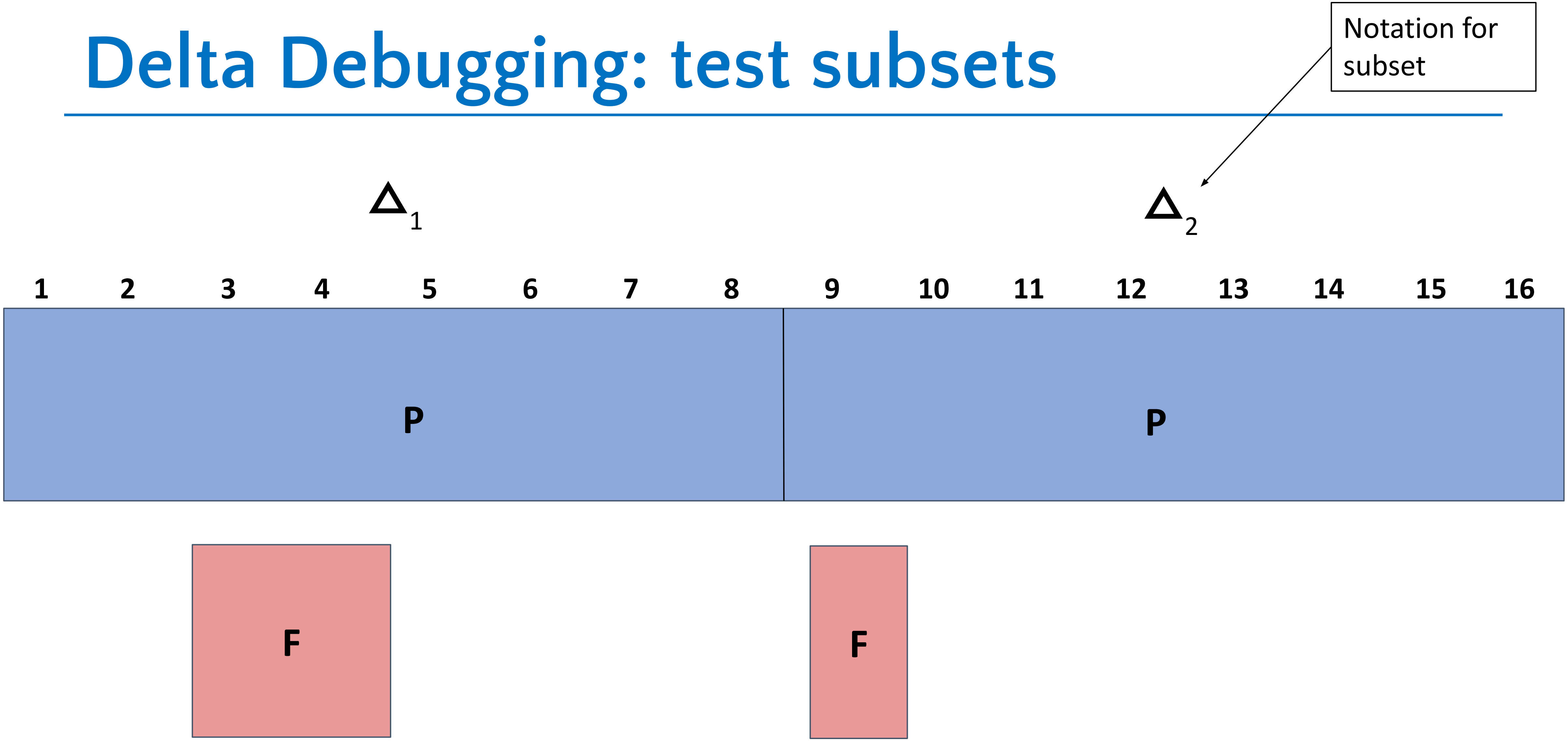
$$ddmin_2(c'_{\text{✘}}, n) = \begin{cases} ddmin_2(\Delta_i, 2) & \text{if } \exists i \in \{1, \ldots, n\} \cdot test(\Delta_i) = \text{✘ (“reduce to subset”)} \\ ddmin_2\big(\nabla_i, \max(n-1, 2)\big) & \text{else if } \exists i \in \{1, \ldots, n\} \cdot test(\nabla_i) = \text{✘ (“reduce to complement”)} \\ ddmin_2\big(c'_{\text{✘}}, \min(|c'_{\text{✘}}|, 2n)\big) & \text{else if } n < |c'_{\text{✘}}| \text{ (“increase granularity”)} \\ c'_{\text{✘}} & \text{otherwise (“done”).} \end{cases}$$

where $\nabla_i = c'_{\text{✘}} - \Delta_i$, $c'_{\text{✘}} = \Delta_1 \cup \Delta_2 \cup \cdots \cup \Delta_n$, all Δ_i are pairwise disjoint, and $\forall \Delta_i \cdot |\Delta_i| \approx |c'_{\text{✘}}|/n$ holds.
The recursion invariant (and thus precondition) for $ddmin_2$ is $test(c'_{\text{✘}}) = \text{✘} \wedge n \leq |c'_{\text{✘}}|$.

Delta Debugging is mostly binary search

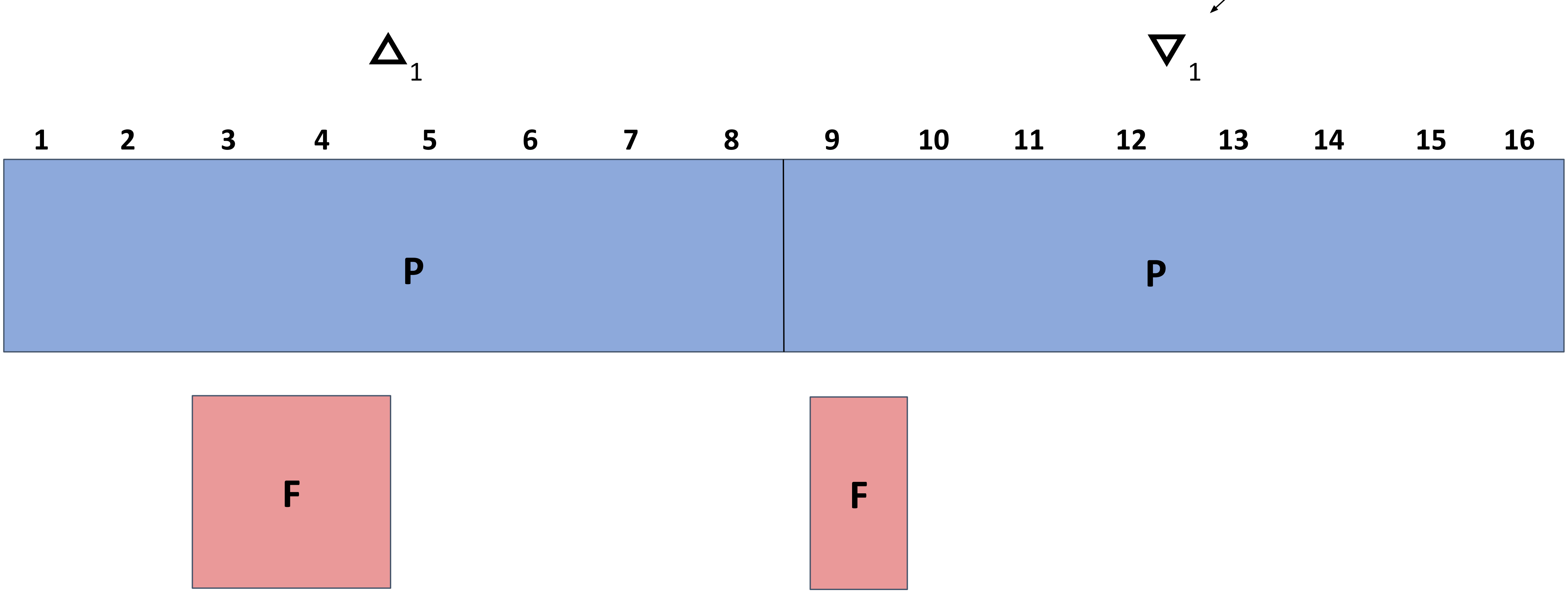


Delta Debugging: test subsets

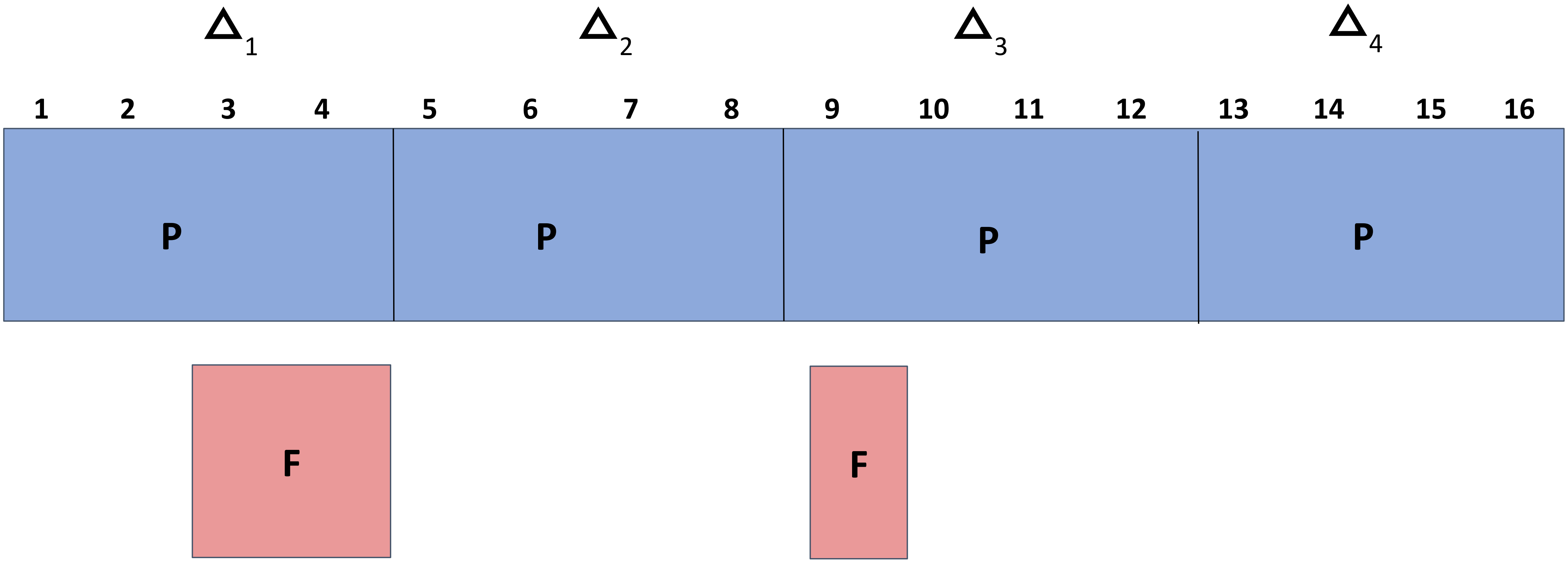


Delta Debugging: test complements

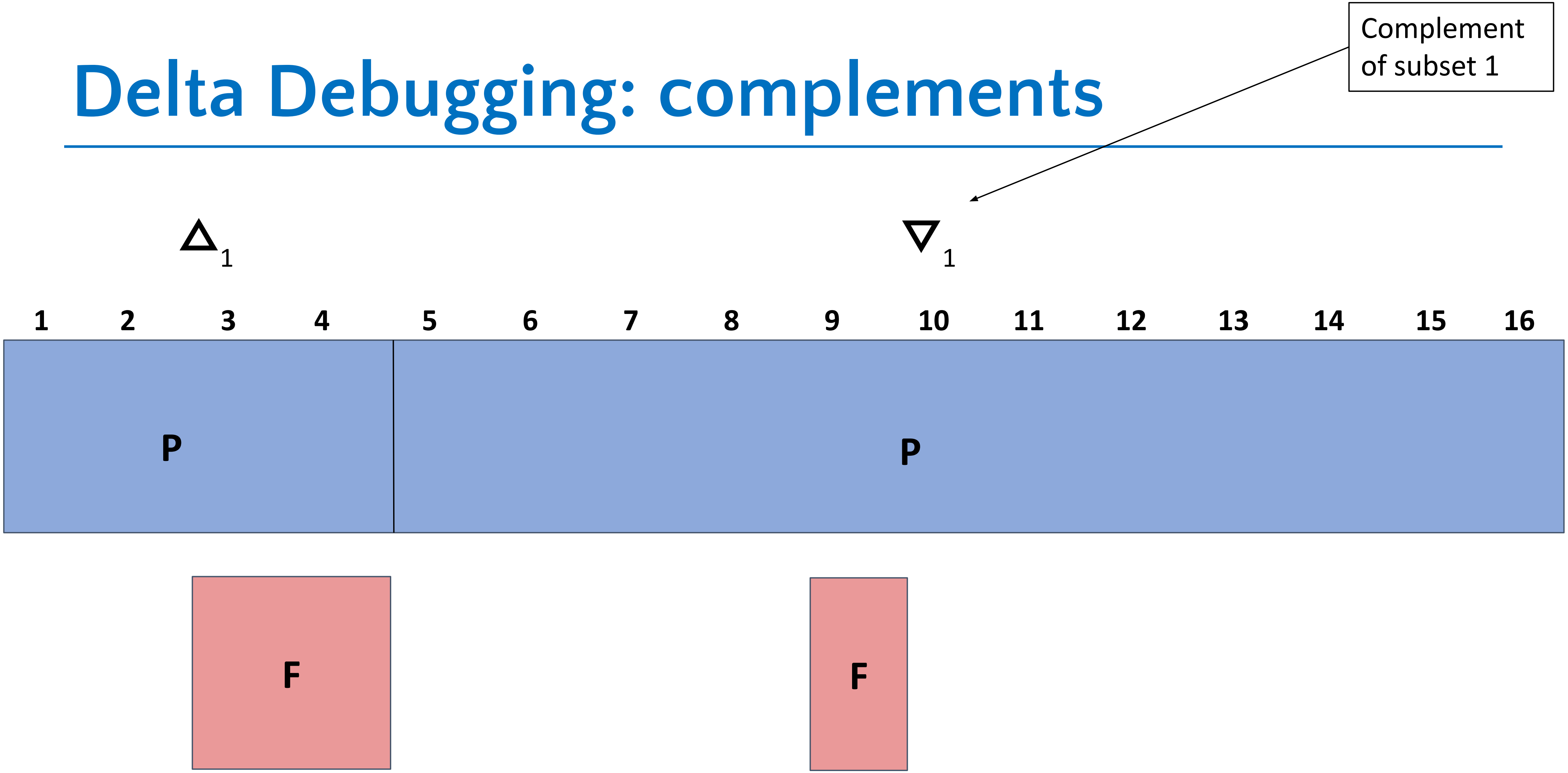
Notation for complement of subset 1



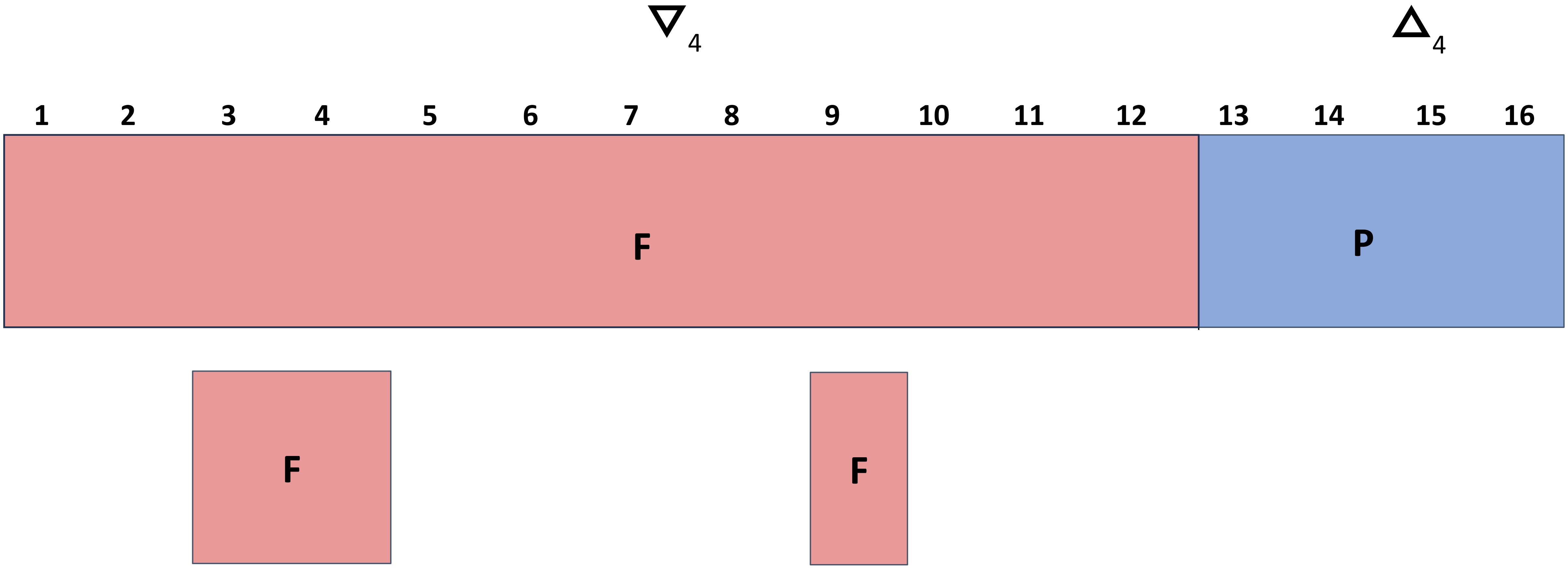
Delta Debugging: increase granularity



Delta Debugging: complements

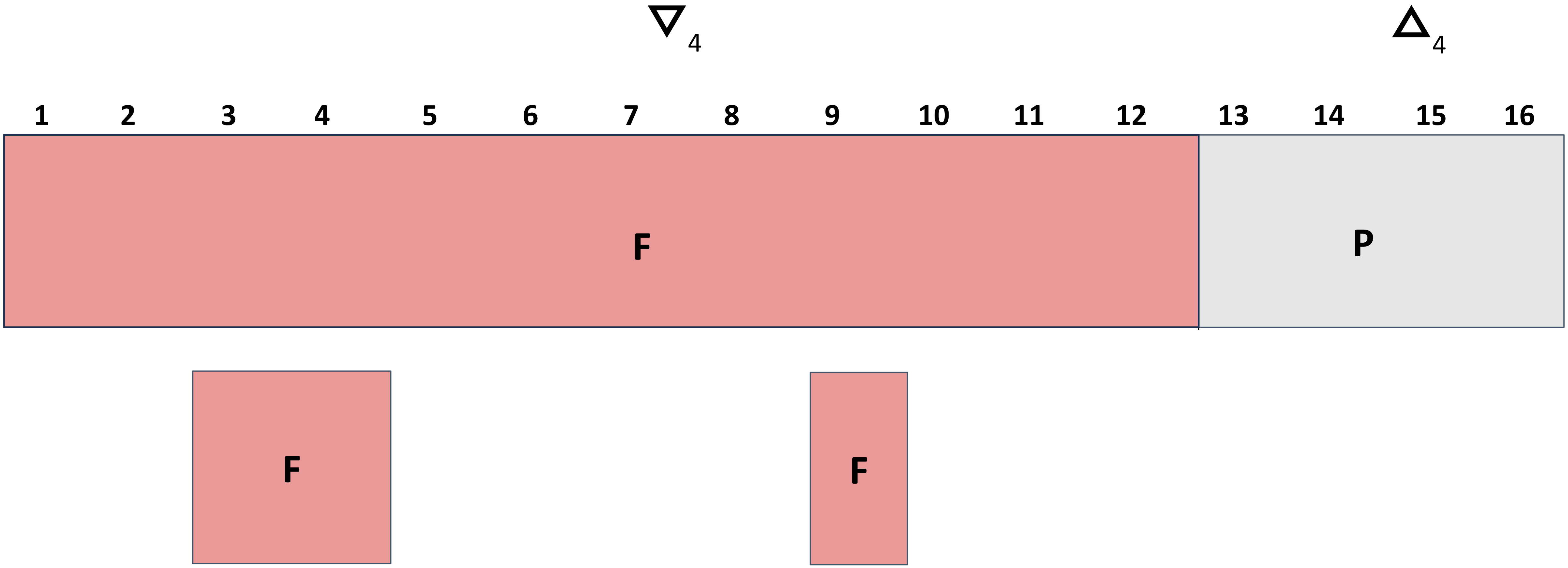


Delta Debugging: complements

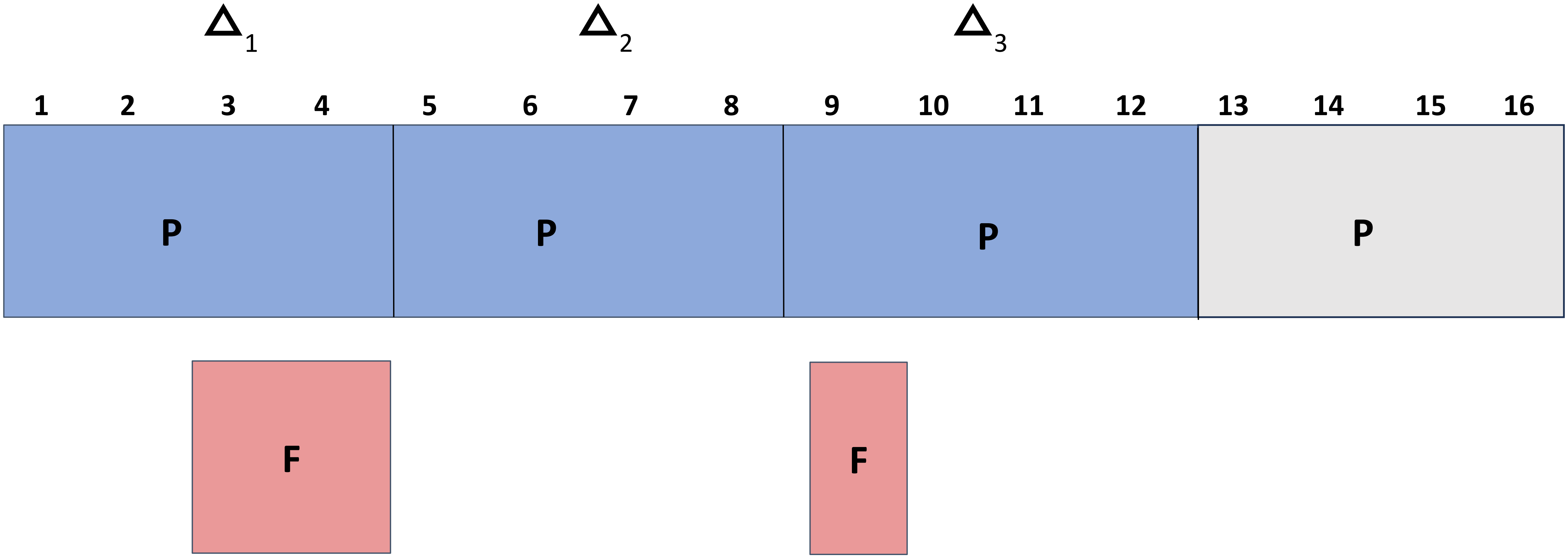


* ∇_2 also fails tests

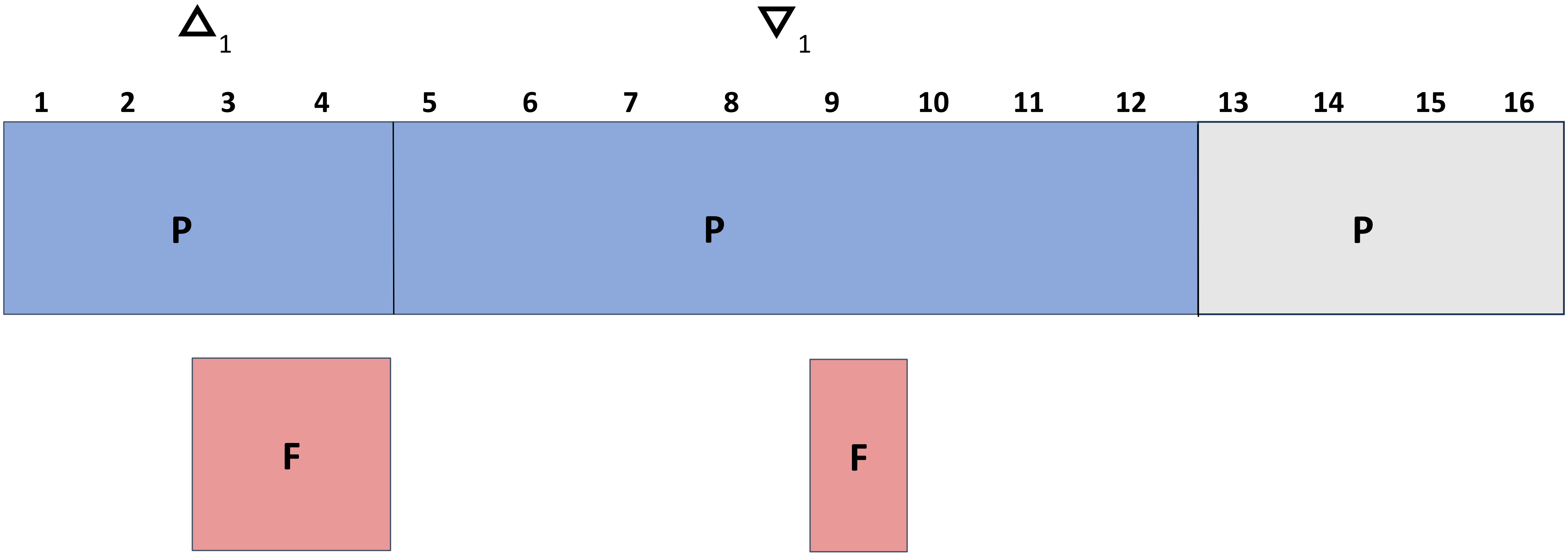
Delta Debugging: reduce



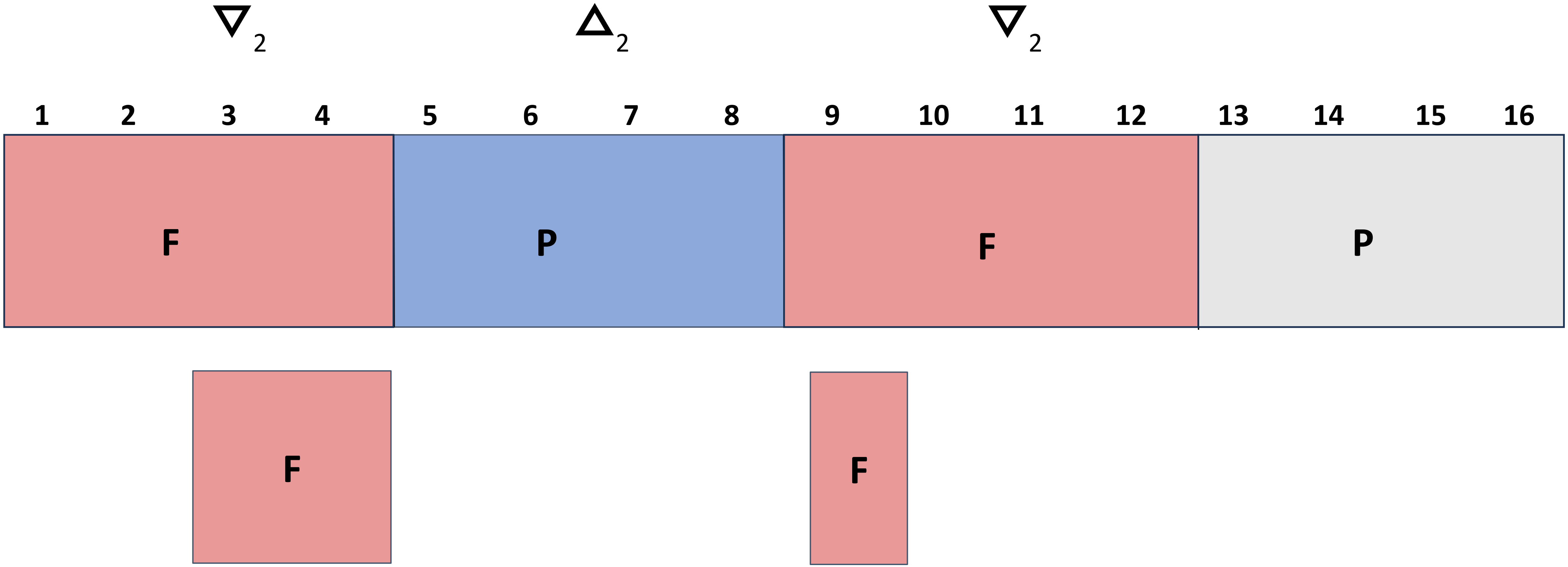
Delta Debugging: test subsets



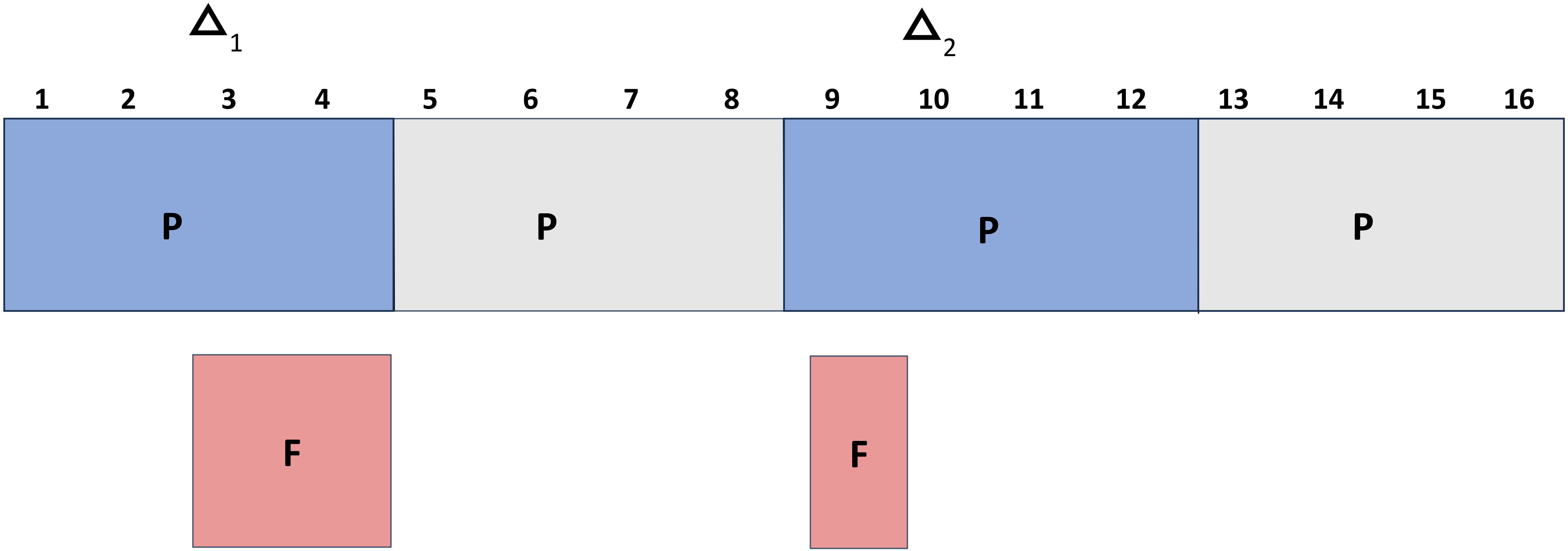
Delta Debugging: complements



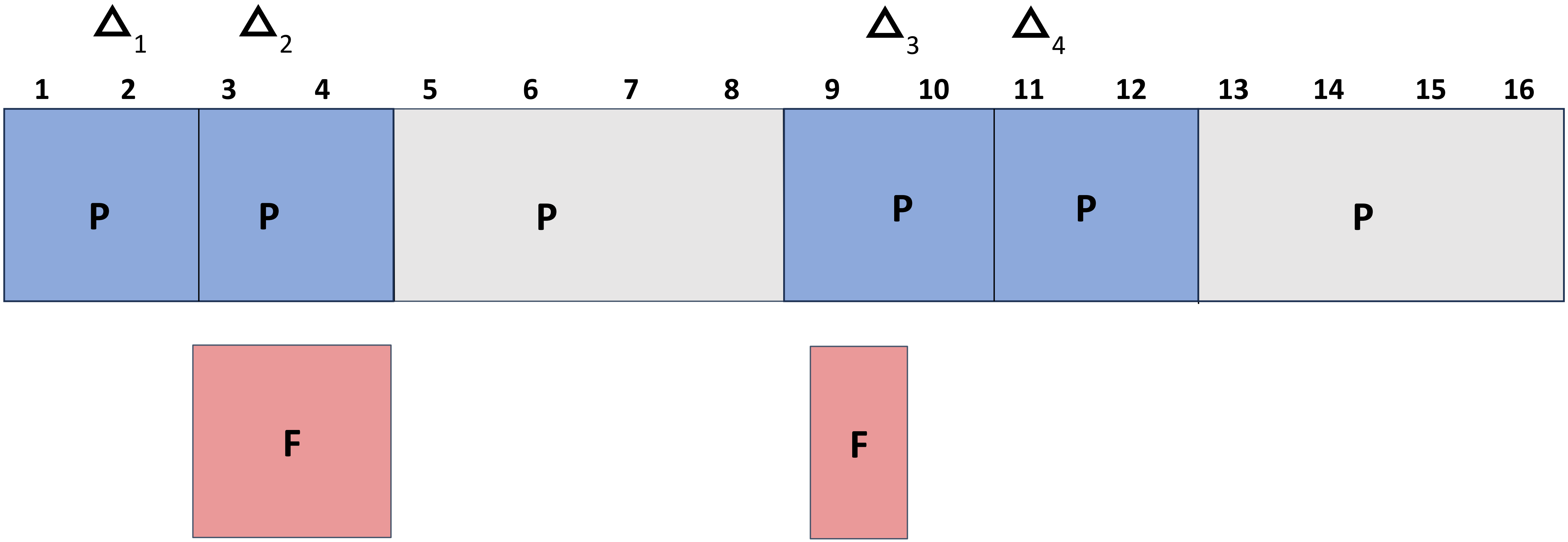
Delta Debugging: complements



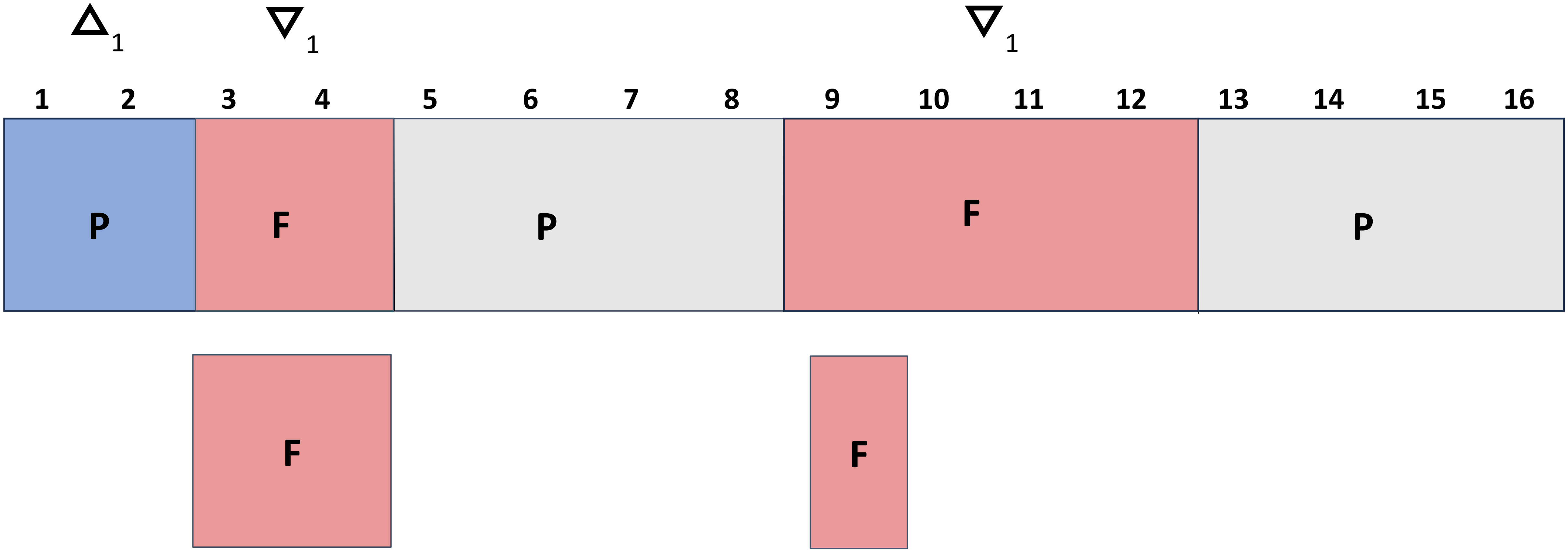
Delta Debugging: reduce



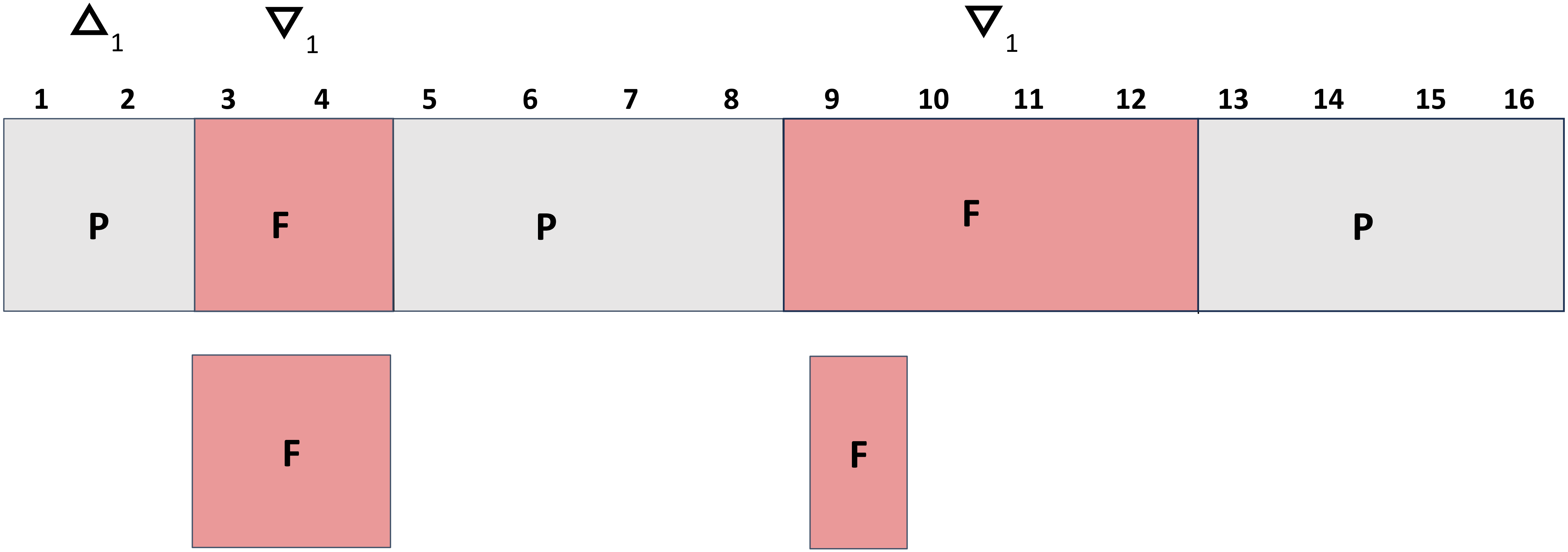
Delta Debugging: increase granularity



Delta Debugging: complements

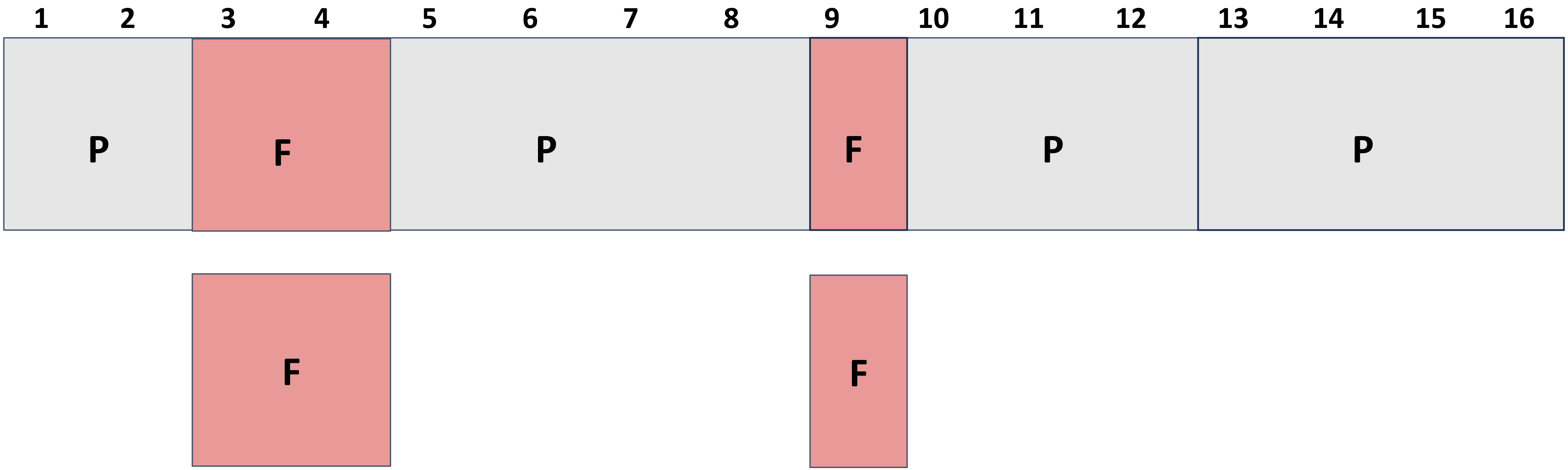


Delta Debugging: reduce



And so on...

Delta Debugging finds a “1-minimal” solution



Failing test cases must be deterministic and monotone

Delta debugging: live example

Delta Debugging: live example

Program and initial test case

- Program **P crashes** whenever the input contains **1 7 8**
- Initial crashing test case is: **1 2 3 4 5 6 7 8**

Syntax:

- % ./delta –test=./**test.sh** –cp_minimal=./min.txt < **failing.txt**
- test.sh returns 0 if input causes failure, 1 if input passes

Delta debugging approach:

- Test each subset*
- Test each complement*
- Increase granularity

Reduce on success

Example taken from the reading

Quiz

Program and initial test case

- Program *P* takes as **input a String of a**$_s$ **and b**$_s$.
- *P* crashes whenever the input String contains an even number of **a**$_s$ **and** an odd number of **b**$_s$.
- Assume **character-level** granularity.
- Example crashing test inputs: **babab**, **aaaabbb**.

Determine the following test cases

1. Smallest
2. 1-minimal but not smallest

Give an input such that DD outputs each of these.

Quiz

Program and initial test case

- Program *P* takes as **input a String of a$_s$ and b$_s$**.
- *P* crashes whenever the input String contains an even number of **a**$_s$ **and** an odd number of **b**$_s$.
- Assume **character-level** granularity.
- Example crashing test inputs: **babab**, **aaaabbb**.

Determine the following test cases

1. Smallest **b**
2. 1-minimal but not smallest **aab**

Give an input such that DD outputs each of these.

Let’s try one more

Program and initial test case

- Program ***P*** takes as **input a list of integers *lst***.
- ***P* crashes** whenever ***lst* contains 4,2**.
- Initial crashing test input is: **2,4,2,4**

Complete the following table (add a new row whenever the number of subsets changes)

Iteration	Number of subsets	input	$\Delta_1, \ldots, \Delta_n$ $\nabla_1, \ldots, \nabla_n$
1	2	2424	
2			
3			
4			

Let’s try one more

Program and initial test case

- Program ***P*** takes as **input a list of integers *lst***.
- ***P* crashes** whenever ***lst* contains 4,2**.
- Initial crashing test input is: **2,4,2,4**

Complete the following table (add a new row whenever the number of subsets changes)

Iteration	n	input	$\Delta_1, \ldots, \Delta_n$ $\nabla_1, \ldots, \nabla_n$
1	2	2424	24, (24)
2	4	2424	2, 4, (2), (4), **424**, ~~224~~, ~~244~~, ~~242~~
3	3	424	(4), (2), (4), (24), 44, **42**
4	2	42	(4), (2)

- Parenthesized values are cached.
- Boldfaced values are success.
- Struck out values were not tested.
- An implementation chooses the order of subsets/complements.

Delta debugging: summary

Discussion

- Non-deterministic programs
- Input structure and granularity
- Monotonicity
- Complexity

- What other structures can you delta debug over?

Delta debugging: in-class exercise

Examples from demo

Four basic phases:

1. Test subsets
2. Test complements
3. Increase granularity
4. Reduce

Oracle test (look for (1 and 2) or 8, 0=test fails, 1=test passes)

test.sh

```
#!/bin/sh
cat $1 | tr -d '\n' | grep -q -E “12|8”
```

review-files.sh

```
#!/bin/sh
for file in tmp0/*.c ; do cat $file | tr -d '\n' ; echo ; done
```

Sample oracle test if you don’t have one yet

test.sh

```
#!/bin/sh
# -*-sh-*-
if timeout 0.5s perl ../../mysort.pl $1 >/dev/null; then exit 1; fi
```