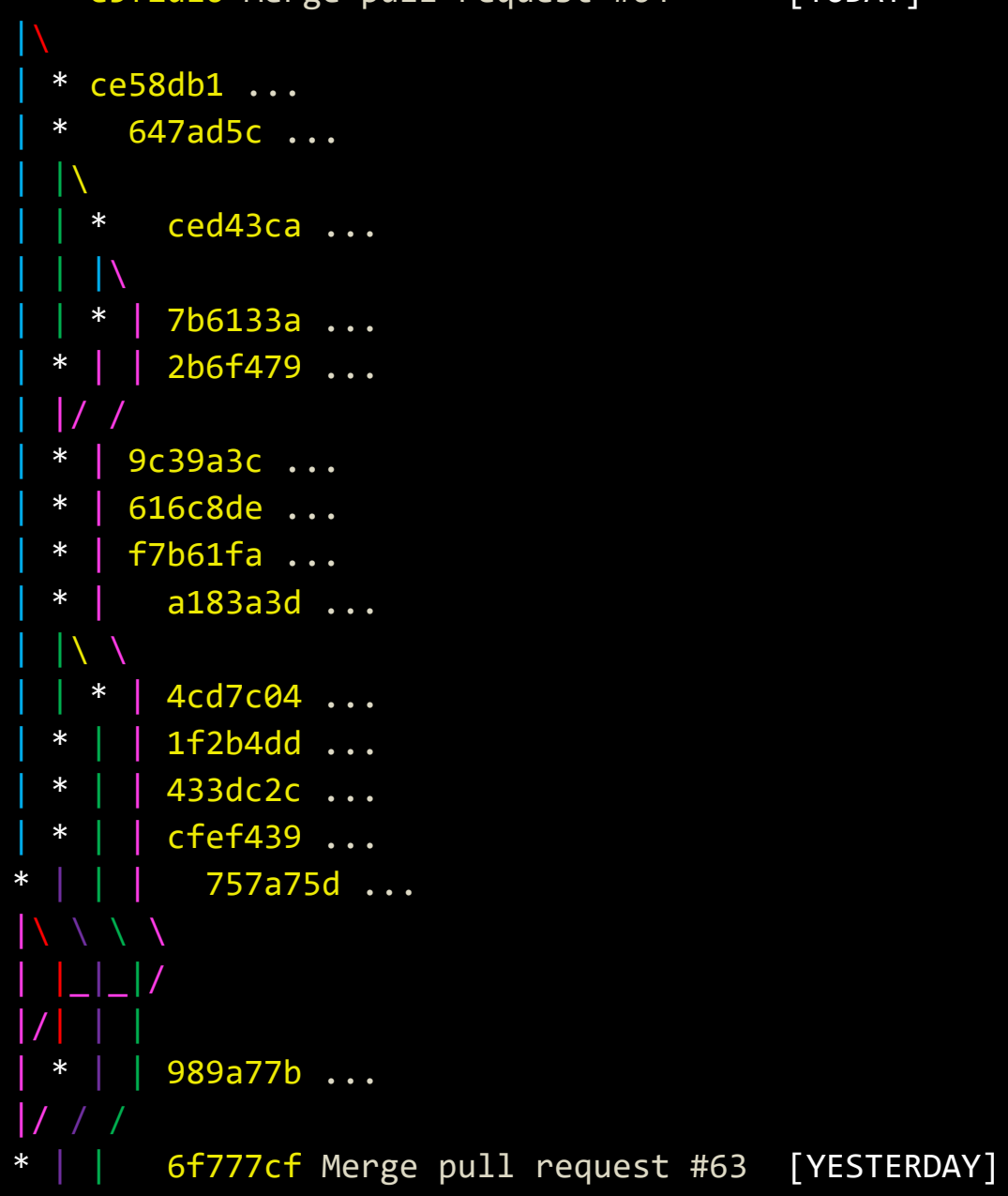
YESTERDAY, MY PROGRAM WORKED. TODAY, IT DOES NOT. WHY?

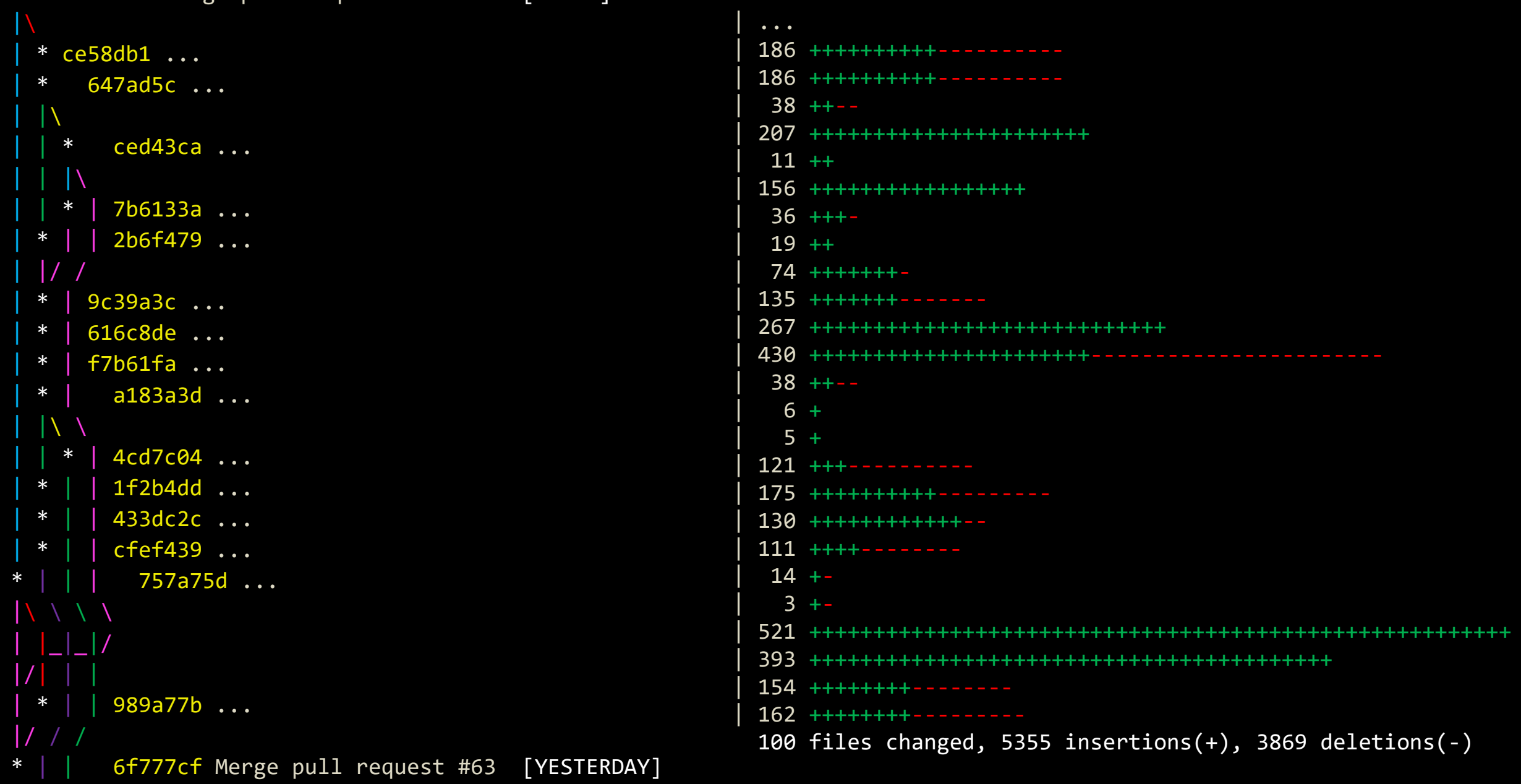
Andreas Zeller

Presented by: Ming-Ho Yee
November 22, 2016

Impact of Delta Debugging

- Introduced “delta debugging,” an *automated* debugging technique
  - First in a series of papers
- Won the ACM SIGSOFT Impact Paper Award (2009)
- Over 100 citations
  - Still being cited today!


* e5f2d16 Merge pull request #64 [TODAY]
* ce58db1 ...
* 647ad5c ...
* ced43ca ...
* 7b6133a ...
* 2b6f479 ...
* 9c39a3c ...
* 616c8de ...
* f7b61fa ...
* a183a3d ...
* 4cd7c04 ...
* 1f2b4dd ...
* 433dc2c ...
* cfef439 ...
* 757a75d ...
* 989a77b ...
* 6f777cf Merge pull request #63 [YESTERDAY]


* e5f2d16 Merge pull request #64 [TODAY]
* ce58db1 ...
* 647ad5c ...
* ced43ca ...
* 7b6133a ...
* 2b6f479 ...
* 9c39a3c ...
* 616c8de ...
* f7b61fa ...
* a183a3d ...
* 4cd7c04 ...
* 1f2b4dd ...
* 433dc2c ...
* cfef439 ...
* 757a75d ...
* 989a77b ...
* 6f777cf Merge pull request #63 [YESTERDAY]
... | 186 +++++++++++---------- | 186 +++++++++++---------- | 38 ++-- | 207 ++++++++++++++++++++++ | 11 ++ | 156 ++++++++++++++++ | 36 +++- | 19 ++ | 74 +++++++- | 135 +++++++------- | 267 ++++++++++++++++++++++++++++ | 430 ++++++++++++++++++++++++----------------------- | 38 ++-- | 6 + | 5 + | 121 +++++---------- | 175 ++++++++++---------- | 130 ++++++++++-- | 111 ++++-------- | 14 +- | 3 +- | 521 ++ | 393 ++ | 154 ++++++++-------- | 162 ++++++++---------- 100 files changed, 5355 insertions(+), 3869 deletions(-)

Prerequisites for Delta Debugging

- Set of all possible changes: $\mathcal{C} = \{ \Delta_1, \Delta_2, \ldots \Delta_n \}$
  - A change set $c \subseteq \mathcal{C}$ is called a *configuration*.
  - An empty configuration $c = \emptyset$ is called a *baseline*.
- Function $test : 2^{\mathcal{C}} \rightarrow \{✓, ✗, ?\}$
  - ✓ - Pass
  - ✗ - Fail
  - ? - Unresolved
- Assumption:

$$test(\emptyset) = ✓ \wedge test(\mathcal{C}) = ✗$$

Minimal Failure-Inducing Change Set

- **Goal**: Find the minimal failure-inducing change set

- A change set $c \subseteq \mathcal{C}$ is *failure-inducing* if the following holds:

$$\forall c'(c \subseteq c' \subseteq \mathcal{C} \rightarrow test(c') \neq \checkmark)$$

- A failure-inducing change set $B \subseteq \mathcal{C}$ is *minimal* if the following holds:

$$\forall c \subset B\ (test(c) \neq \text{✗})$$

Three Useful Properties

- **Monotone**

$$\forall c \subseteq \mathcal{C}(test(c) = \text{✗} \rightarrow \forall c' \supseteq c(test(c') \neq \text{✓}))$$
$$\forall c \subseteq \mathcal{C}(test(c) = \text{✓} \rightarrow \forall c' \subseteq c(test(c') \neq \text{✗}))$$

- **Unambiguous**

$$\forall c_1, c_2 \subseteq \mathcal{C}(test(c_1) = \text{✗} \wedge test(c_2) = \text{✗} \rightarrow test(c_1 \cap c_2) \neq \text{✓})$$

- **Consistent**

$$\forall c \subseteq \mathcal{C}(test(c) \neq ?)$$

Delta Debugging

$$dd(c) = dd_2(c, \emptyset)$$

$$dd_2(c, r) =$$

$$\text{let } c_1, c_2 \subseteq c \text{ s.t. } \left(c_1 \cup c_2 = c \wedge c_1 \cap c_2 = \emptyset \wedge |c_1| \approx |c_2| \approx \frac{|c|}{2}\right)$$

$$\text{in } \begin{cases} c & \text{if } |c| = 1 \\ dd_2(c_1, r) & \text{if } test(c_1 \cup r) = \text{✗} \\ dd_2(c_2, r) & \text{if } test(c_2 \cup r) = \text{✗} \\ dd_2(c_1, c_2 \cup r) \cup dd_2(c_2, c_1 \cup r) & \text{otherwise} \end{cases}$$

Invariant: $test(r) = \text{✓} \wedge test(c \cup r) = \text{✗}$

Delta Debugging Example

$dd_2(\{1,2,3,4,5,6,7,8\}, \emptyset)$

Step	Configuration	*test*

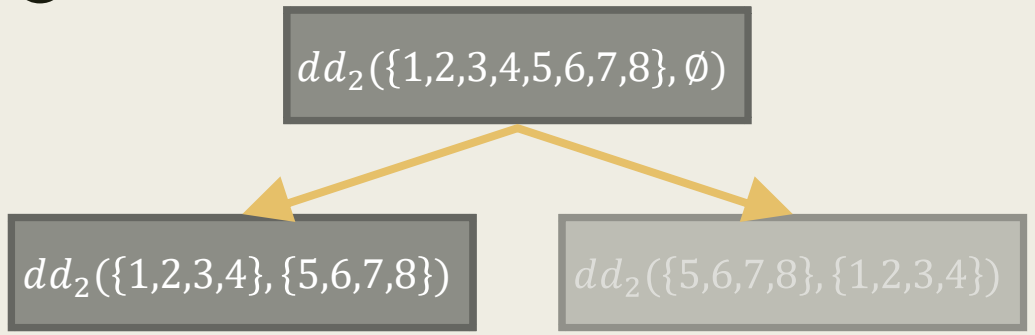
Delta Debugging Example

$dd_2(\{1,2,3,4,5,6,7,8\}, \emptyset)$

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓

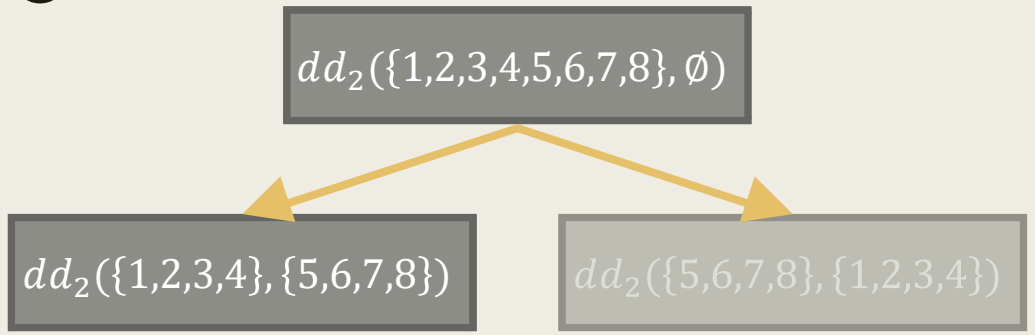
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓



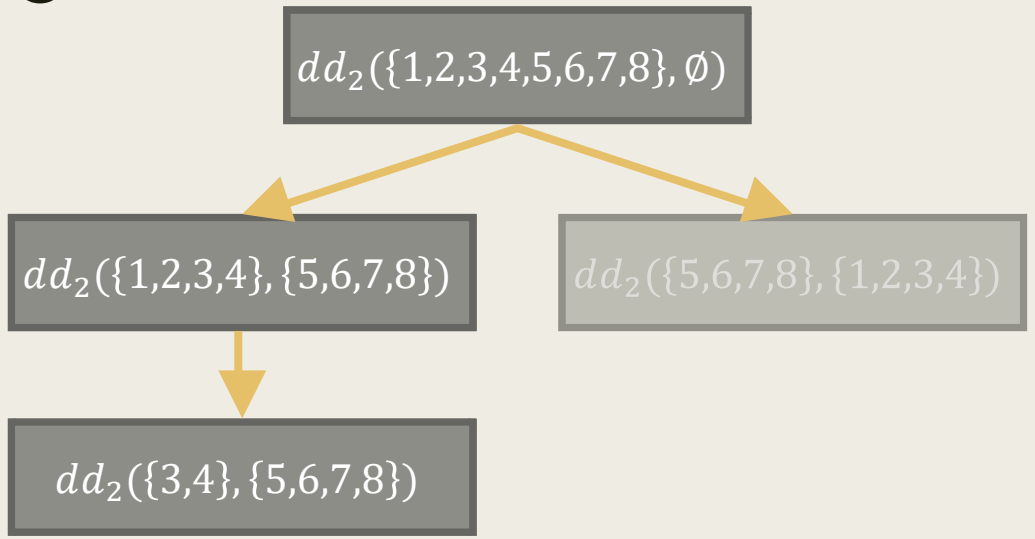
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗



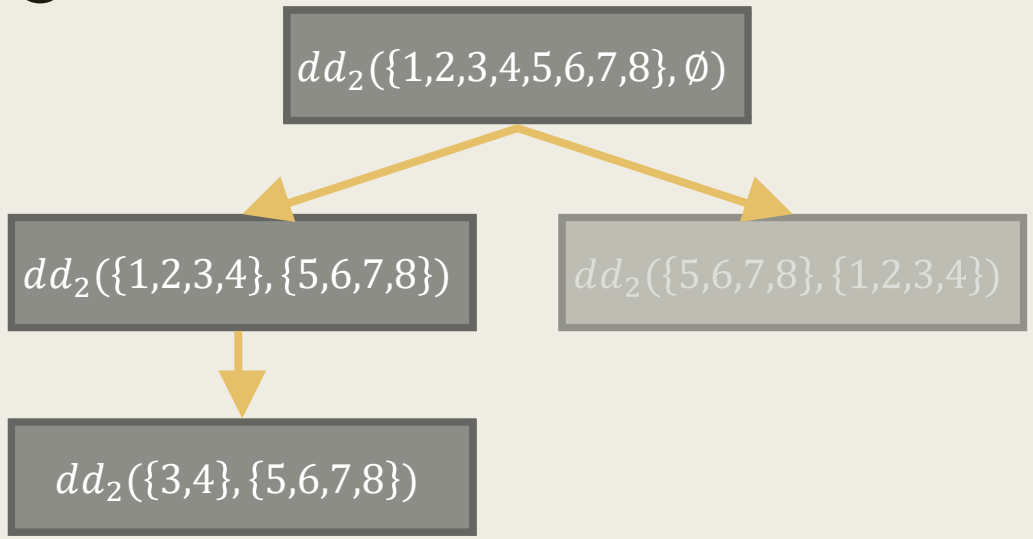
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗



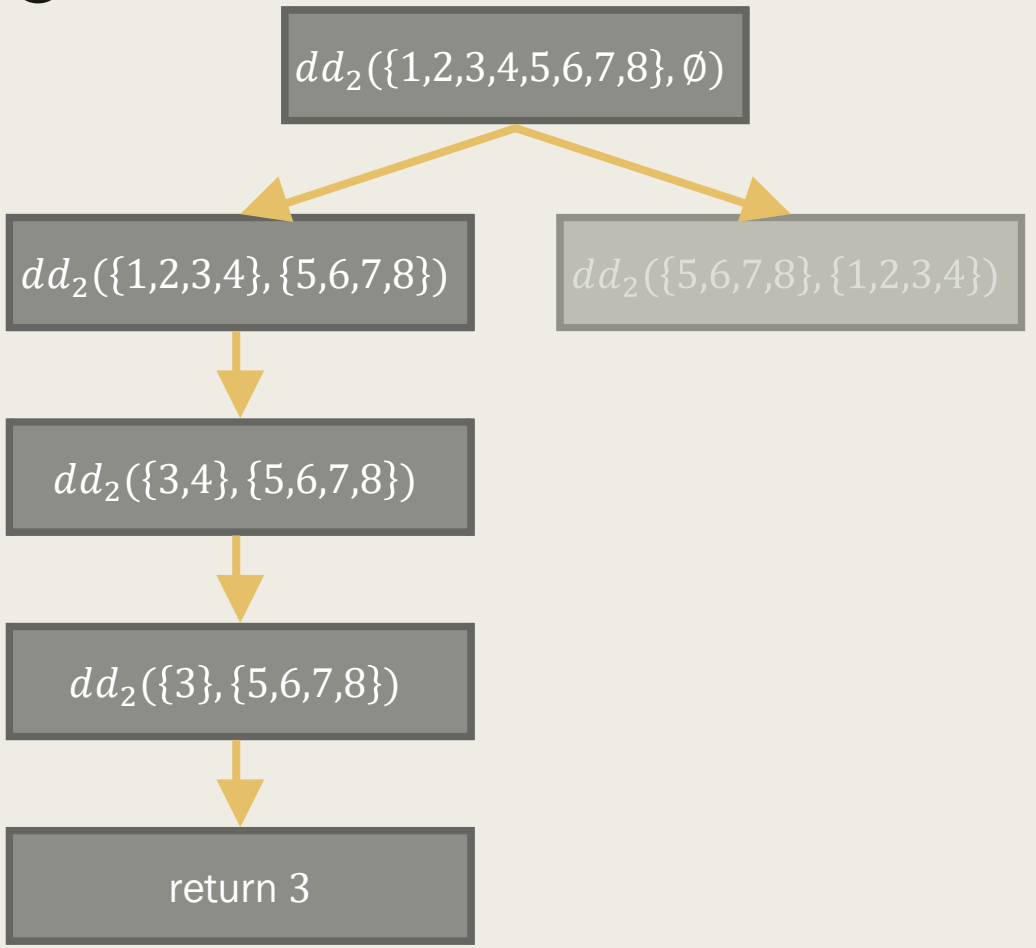
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗
5	.	.	3	.	5	6	7	8	✗



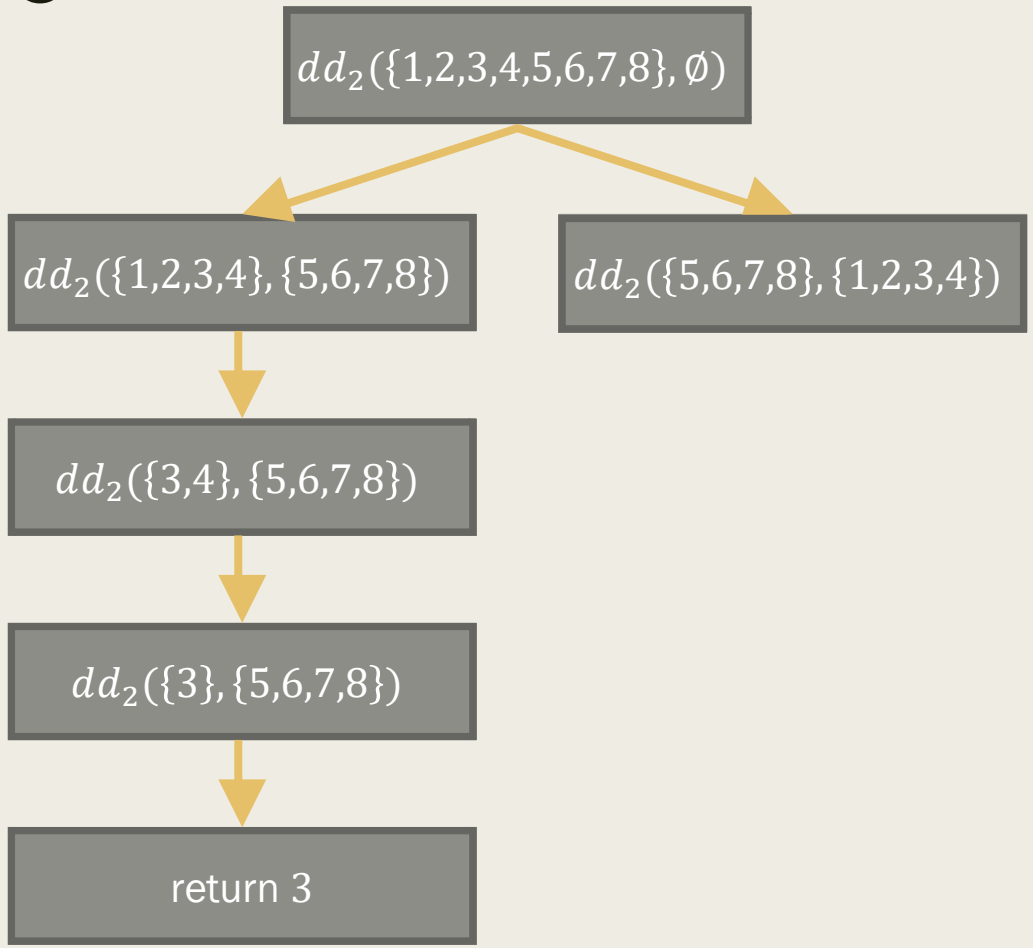
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗
5	.	.	3	.	5	6	7	8	✗



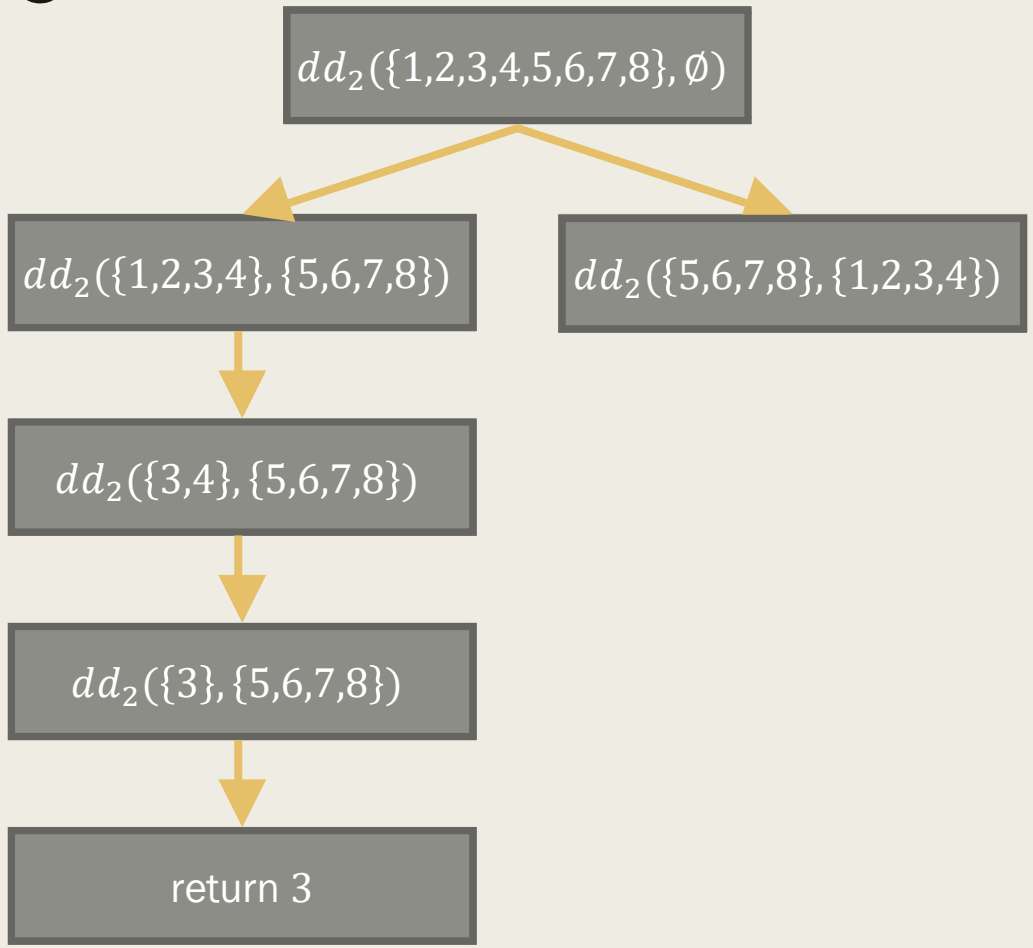
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗
5	.	.	3	.	5	6	7	8	✗



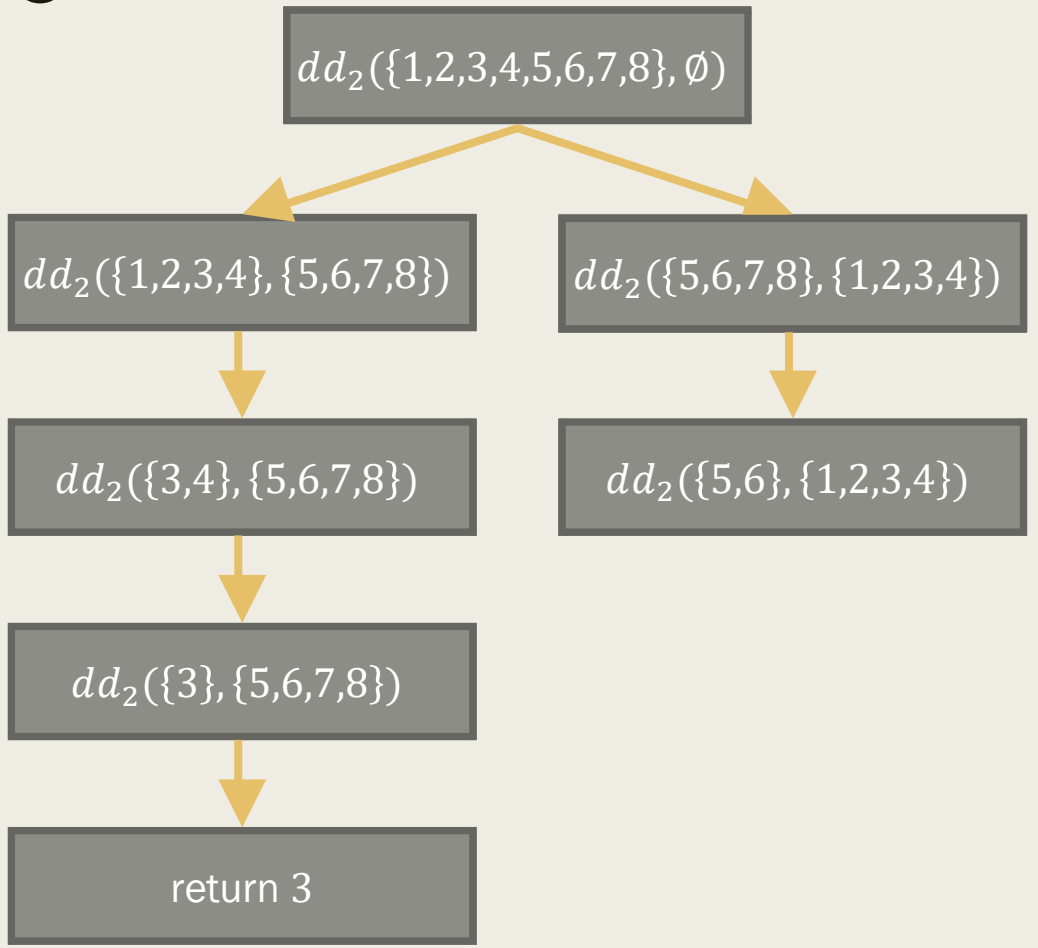
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗
5	.	.	3	.	5	6	7	8	✗
6	1	2	3	4	5	6	.	.	✗



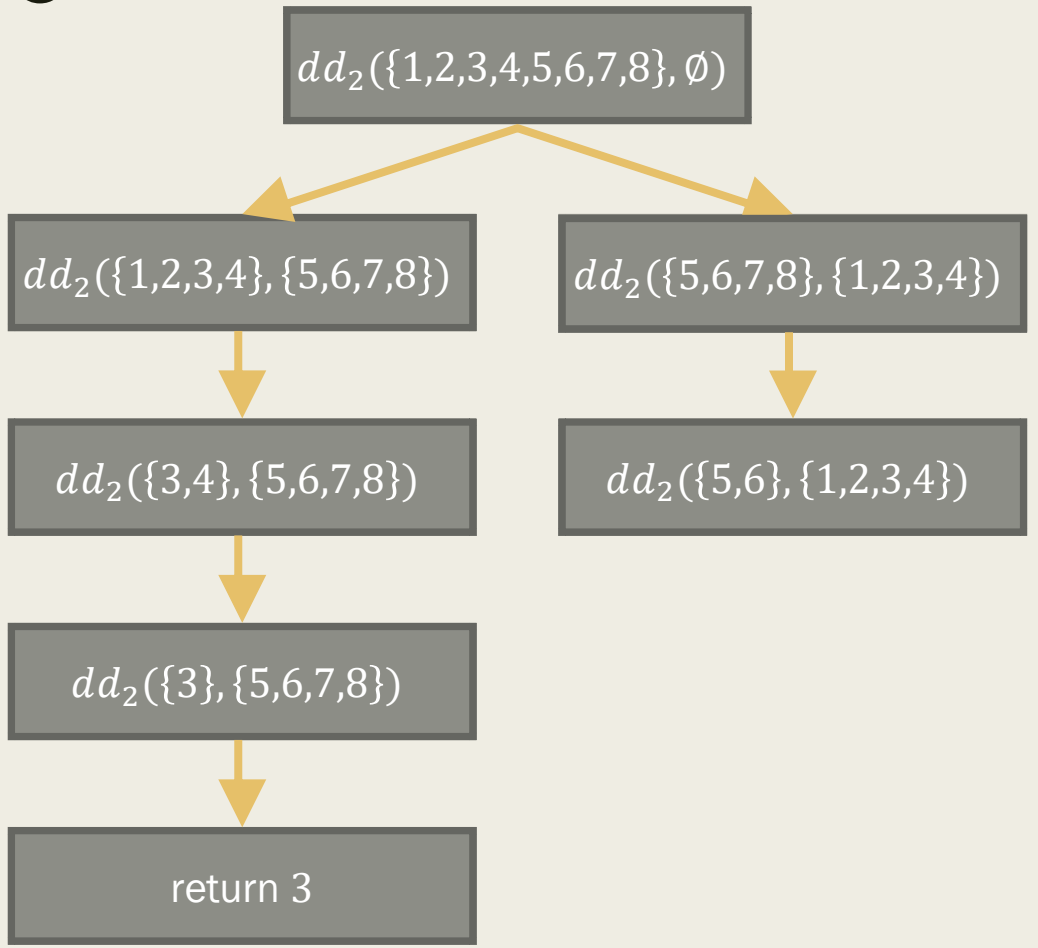
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗
5	.	.	3	.	5	6	7	8	✗
6	1	2	3	4	5	6	.	.	✗



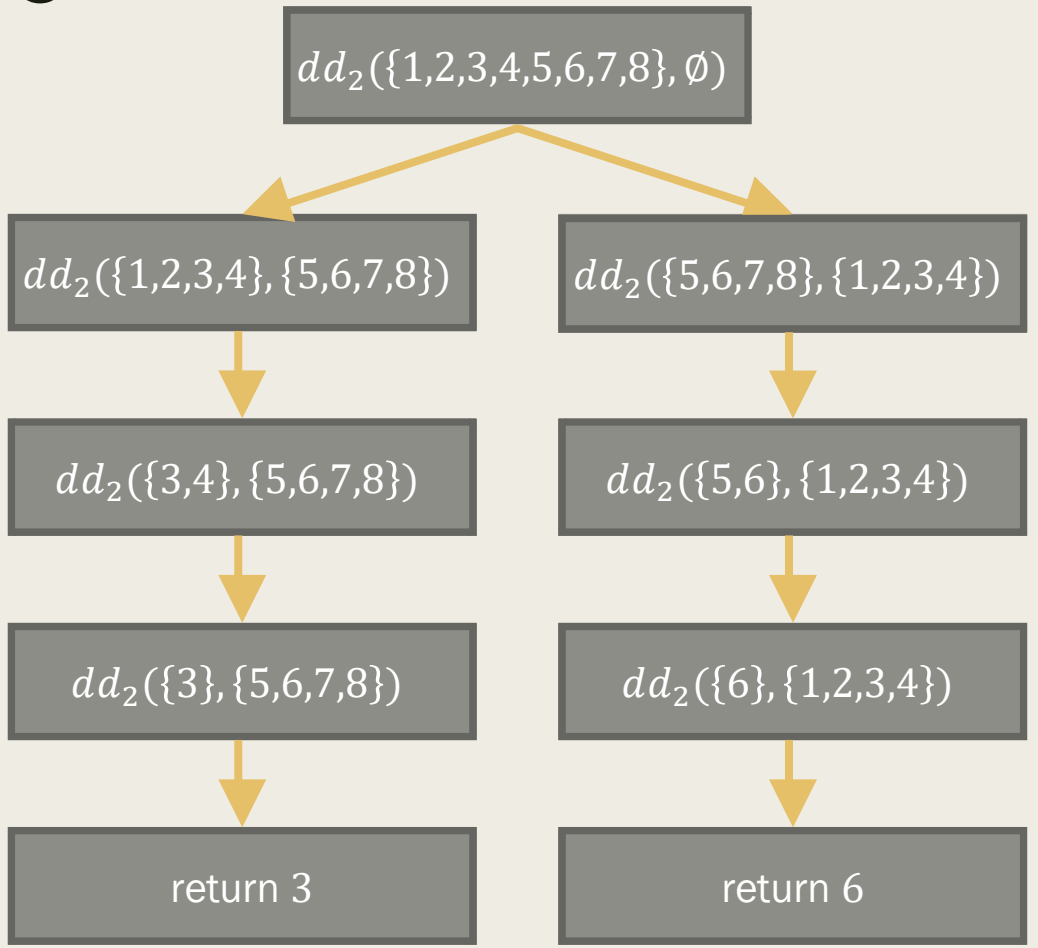
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗
5	.	.	3	.	5	6	7	8	✗
6	1	2	3	4	5	6	.	.	✗
7	1	2	3	4	5	.	.	.	✓



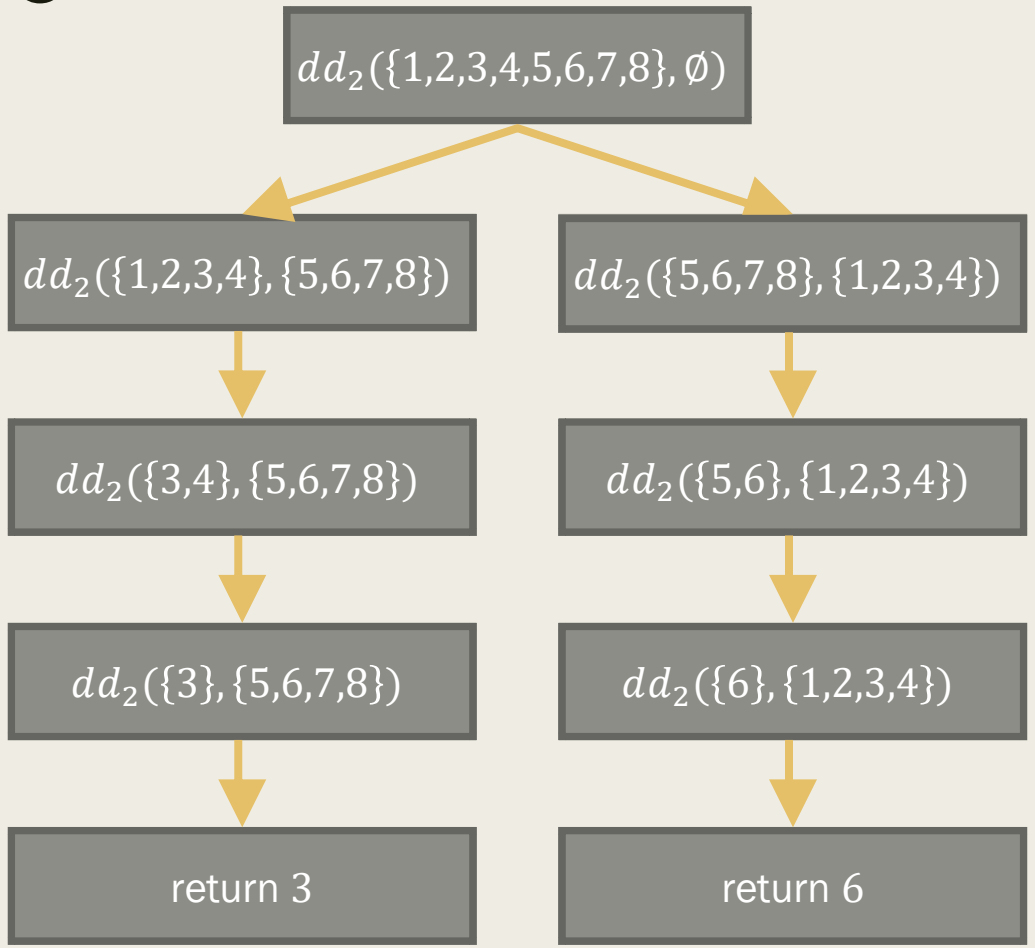
Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗
5	.	.	3	.	5	6	7	8	✗
6	1	2	3	4	5	6	.	.	✗
7	1	2	3	4	5	.	.	.	✓



Delta Debugging Example

Step	Configuration								*test*
1	1	2	3	4	.	.	.	.	✓
2	.	.	.	.	5	6	7	8	✓
3	1	2	.	.	5	6	7	8	✓
4	.	.	3	4	5	6	7	8	✗
5	.	.	3	.	5	6	7	8	✗
6	1	2	3	4	5	6	.	.	✗
7	1	2	3	4	5	.	.	.	✓
Result	.	.	3	.	.	6	.	.	



Non-Monotonicity and Ambiguity

- **Non-monotone** configuration: a later change might “undo” a failure
  - But “today” is broken, so there exists another failure-inducing change

- **Ambiguous** configuration: multiple failure-inducing change sets
  - Delta debugging will find one of them

Inconsistency

- Sometimes the outcome of a test cannot be determined
  - E.g. change cannot be applied, program will not build, program does not execute correctly

- Approach: split c into smaller subsets
  - $\emptyset$ (“yesterday”) and $\mathcal{C}$ (“today”) are consistent
  - Want a configuration closer to “yesterday” or “today”

Extending Delta Debugging

- Generalize dd to test n subsets instead of 2
  - Instead of testing c_1 and c_2, test c_i and $\overline{c_i}$, for all c_i

- Cases to consider:
  - **Found** ($test(c_i) = \text{✗}$)
  - **Interference** ($test(c_i) = \text{✓} \wedge test(\overline{c_i}) = \text{✓}$)
  - **Preference** ($test(c_i) = \text{?} \wedge test(\overline{c_i}) = \text{✓}$)
- Failure-inducing change in c_i, but apply all changes in $\overline{c_i}$ for consistency
  - **Try again** (otherwise)

- Resulting set may not be minimal

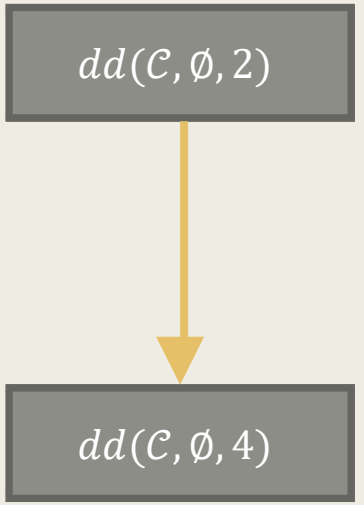
Step	c_i	Configuration	*test*
Result			

$dd(\mathcal{C}, \emptyset, 2)$

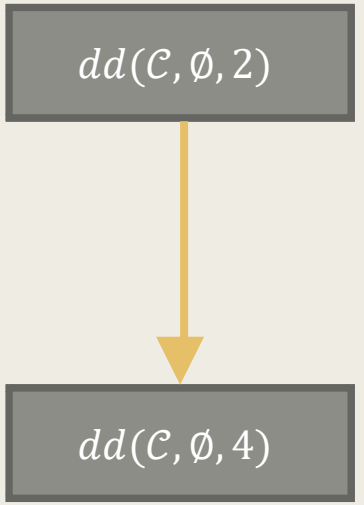
Step	c_i	Configuration								*test*
1	$c_{11} = \overline{c_{12}}$	1	2	3	4	.	.	.	.	?
2	$c_{12} = \overline{c_{11}}$	.	.	.	.	5	6	7	8	?
Result										

$dd(\mathcal{C}, \emptyset, 2)$

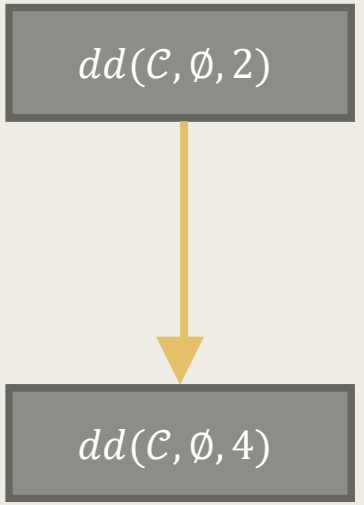
Step	c_i	Configuration								*test*
1	$c_{11} = \overline{c_{12}}$	1	2	3	4	.	.	.	.	?
2	$c_{12} = \overline{c_{11}}$	.	.	.	.	5	6	7	8	?
Result										



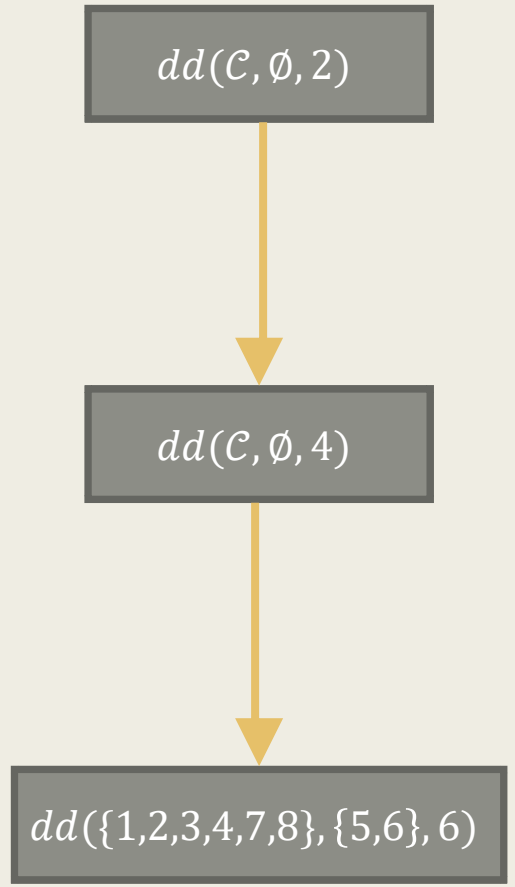
Step	c_i	Configuration								*test*
1	$c_{11} = \overline{c_{12}}$	1	2	3	4	.	.	.	.	?
2	$c_{12} = \overline{c_{11}}$	.	.	.	.	5	6	7	8	?
3	c_{21}	1	2	.	.	.	.	.	.	?
4	c_{22}	.	.	3	4	.	.	.	.	?
5	c_{23}	.	.	.	.	5	6	.	.	✓
6	c_{24}	.	.	.	.	.	.	7	8	?
Result										



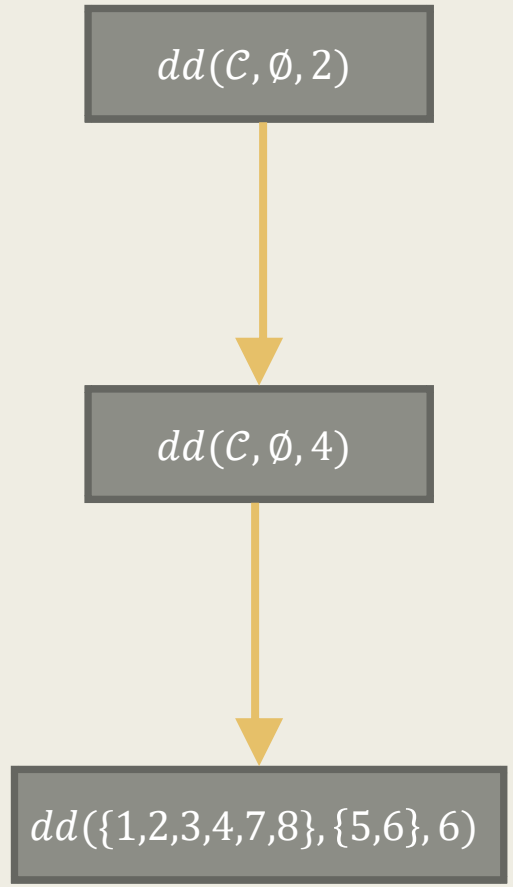
Step	c_i	Configuration								*test*
1	$c_{11} = \overline{c_{12}}$	1	2	3	4	.	.	.	.	?
2	$c_{12} = \overline{c_{11}}$	.	.	.	.	5	6	7	8	?
3	c_{21}	1	2	.	.	.	.	.	.	?
4	c_{22}	.	.	3	4	.	.	.	.	?
5	c_{23}	.	.	.	.	5	6	.	.	✓
6	c_{24}	.	.	.	.	.	.	7	8	?
7	$\overline{c_{21}}$	.	.	3	4	5	6	7	8	?
8	$\overline{c_{22}}$	1	2	.	.	5	6	7	8	?
9	$\overline{c_{23}}$	1	2	3	4	.	.	7	8	✗
10	$\overline{c_{24}}$	1	2	3	4	5	6	.	.	?
Result										



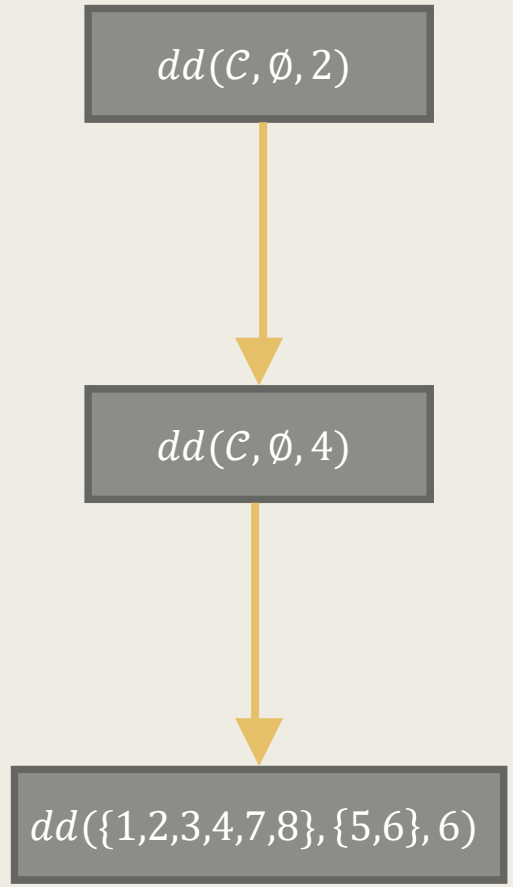
Step	c_i	Configuration								*test*
1	$c_{11} = \overline{c_{12}}$	1	2	3	4	.	.	.	.	?
2	$c_{12} = \overline{c_{11}}$	.	.	.	.	5	6	7	8	?
3	c_{21}	1	2	.	.	.	.	.	.	?
4	c_{22}	.	.	3	4	.	.	.	.	?
5	c_{23}	.	.	.	.	5	6	.	.	✓
6	c_{24}	.	.	.	.	.	.	7	8	?
7	$\overline{c_{21}}$	.	.	3	4	5	6	7	8	?
8	$\overline{c_{22}}$	1	2	.	.	5	6	7	8	?
9	$\overline{c_{23}}$	1	2	3	4	.	.	7	8	✗
10	$\overline{c_{24}}$	1	2	3	4	5	6	.	.	?
Result										



Step	c_i	Configuration								*test*
1	$c_{11} = \overline{c_{12}}$	1	2	3	4	.	.	.	.	?
2	$c_{12} = \overline{c_{11}}$	.	.	.	.	5	6	7	8	?
3	c_{21}	1	2	.	.	.	.	.	.	?
4	c_{22}	.	.	3	4	.	.	.	.	?
5	c_{23}	.	.	.	.	5	6	.	.	✓
6	c_{24}	.	.	.	.	.	.	7	8	?
7	$\overline{c_{21}}$	.	.	3	4	5	6	7	8	?
8	$\overline{c_{22}}$	1	2	.	.	5	6	7	8	?
9	$\overline{c_{23}}$	1	2	3	4	.	.	7	8	✗
10	$\overline{c_{24}}$	1	2	3	4	5	6	.	.	?
11	c_{31}	1	.	.	.	5	6	.	.	✓
12	c_{32}	.	2	.	.	5	6	.	.	?
13	c_{33}	.	.	3	.	5	6	.	.	?
14	c_{34}	.	.	.	4	5	6	.	.	✓
15	c_{35}	.	.	.	.	5	6	7	.	?
16	c_{36}	.	.	.	.	5	6	.	8	✗
Result										



Step	c_i	Configuration								*test*
1	$c_{11} = \overline{c_{12}}$	1	2	3	4	.	.	.	.	?
2	$c_{12} = \overline{c_{11}}$	.	.	.	.	5	6	7	8	?
3	c_{21}	1	2	.	.	.	.	.	.	?
4	c_{22}	.	.	3	4	.	.	.	.	?
5	c_{23}	.	.	.	.	5	6	.	.	✓
6	c_{24}	.	.	.	.	.	.	7	8	?
7	$\overline{c_{21}}$	.	.	3	4	5	6	7	8	?
8	$\overline{c_{22}}$	1	2	.	.	5	6	7	8	?
9	$\overline{c_{23}}$	1	2	3	4	.	.	7	8	✗
10	$\overline{c_{24}}$	1	2	3	4	5	6	.	.	?
11	c_{31}	1	.	.	.	5	6	.	.	✓
12	c_{32}	.	2	.	.	5	6	.	.	?
13	c_{33}	.	.	3	.	5	6	.	.	?
14	c_{34}	.	.	.	4	5	6	.	.	✓
15	c_{35}	.	.	.	.	5	6	7	.	?
16	c_{36}	.	.	.	.	5	6	.	8	✗
Result		.	.	.	.	.	.	.	8	



Avoiding Inconsistency

- **Grouping related changes**
  - E.g. group changes time, file or directory, identifiers referenced

- **Predicting test outcomes**
  - Try to predict if a test is unresolved, instead of running it
  - E.g. order changes, assume a change requires all previous changes

- GDB case study:
  - Original run: 470 tests in 48 hours
  - Reducing inconsistencies: 289 tests in 20 hours

Related Work

- Andreas Zeller has published several papers related to delta debugging
  - Reducing failure-inducing input
  - Finding a failure-inducing thread schedule
  - Isolating cause-effect chains

Implementations

- Eclipse plug-ins
- MyDD Python module
- Delta
- C-Reduce
- Lithium
- WALA's JS Delta
- Flix delta debugging

Conclusions

- Delta debugging can automatically find find failure-inducing changes
- Domain knowledge can help reduce inconsistencies
- The delta debugging technique can be used to minimize test input
  - Many implementations exist

Discussion

- Is delta debugging actually useful for finding changes?
  - Most implementations use delta debugging to reduce test input
  - Inconsistencies seem like they would be very, very common
- What are other applications of delta debugging?