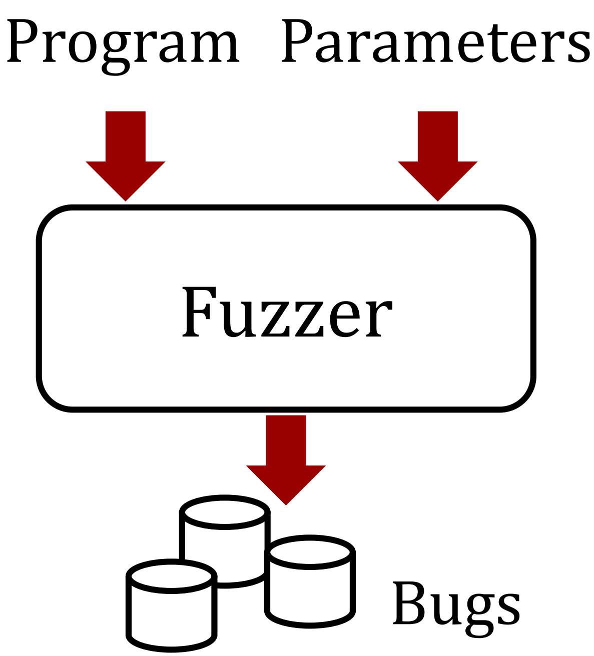
Optimizing Seed Selection for Fuzzing

USENIX Security 2014

Alexandre Rebert
Sang Kil Cha
Thanassis Avgerinos
Jonathan Foote
David Warren
Gustavo Grieco
David Brumley

Optimizing Seed Selection for Fuzzing

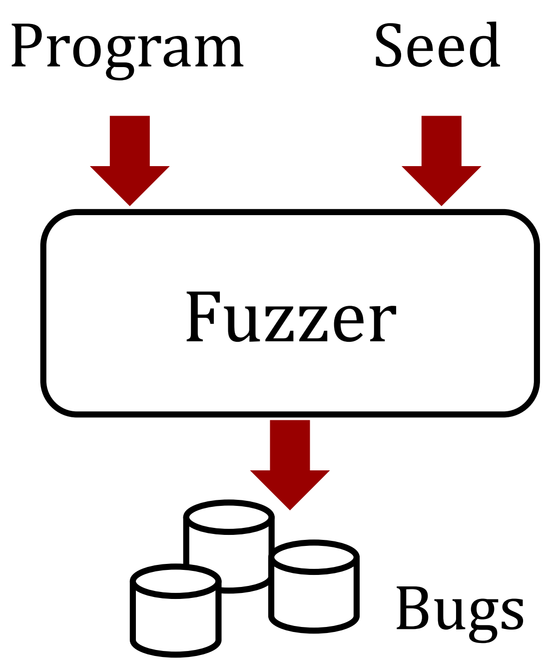
Fuzzing = Bug Finding



Optimizing Seed Selection for Fuzzing

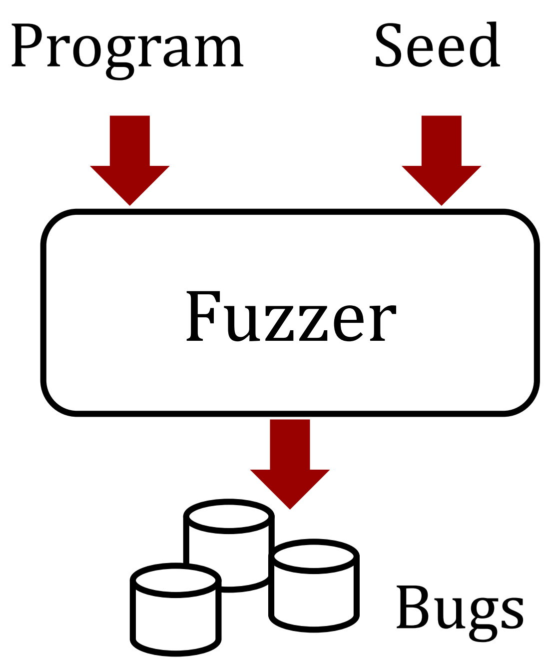
BFF, FileFuzz, jsfunfuzz, Peach, Sage, ZZUF

and many more …



Seed = Well-Structured Input

Seed Selection Challenge



Given:

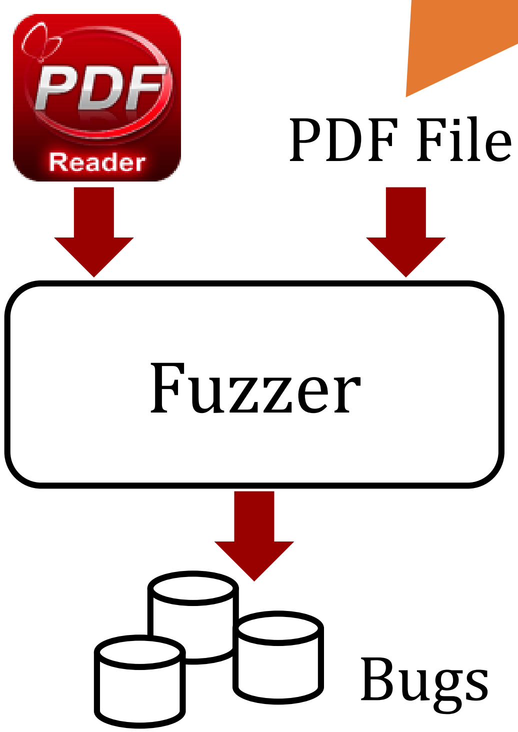
- Program
- Fuzzer
- Time limit T

Seed Selection Challenge

You can run the fuzzer with any seed for any arbitrary time period (total time ≤ T)

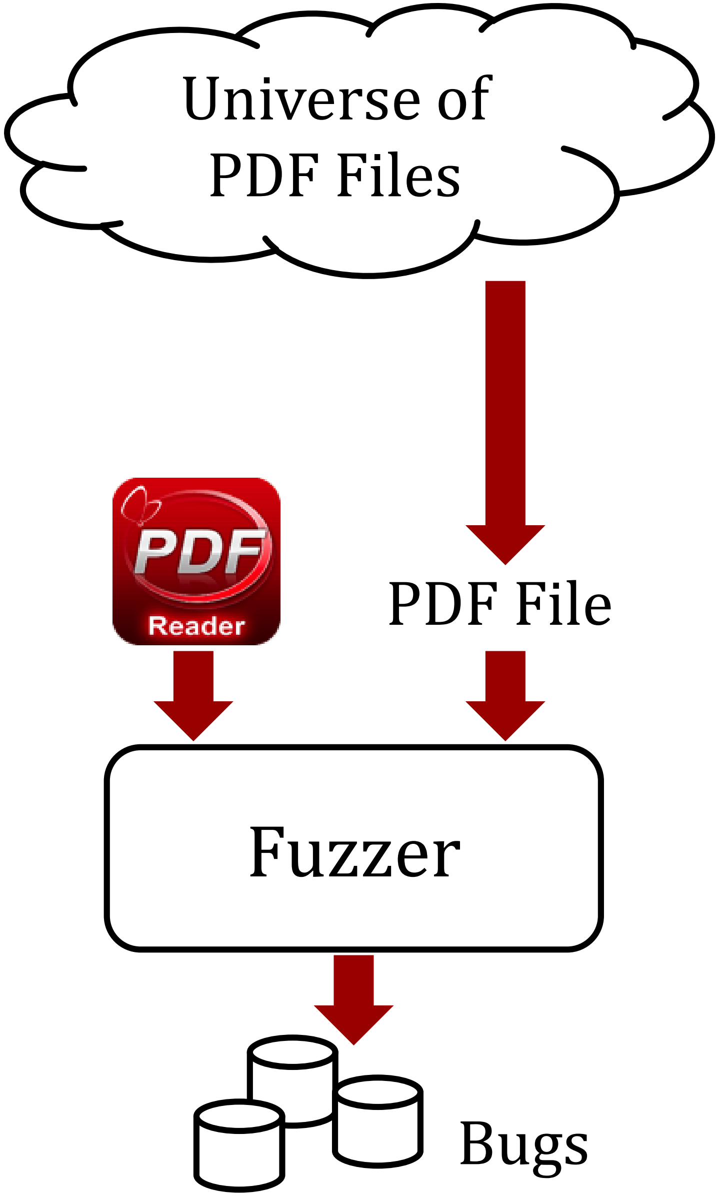
Given:

- Program
- Fuzzer
- Time limit T

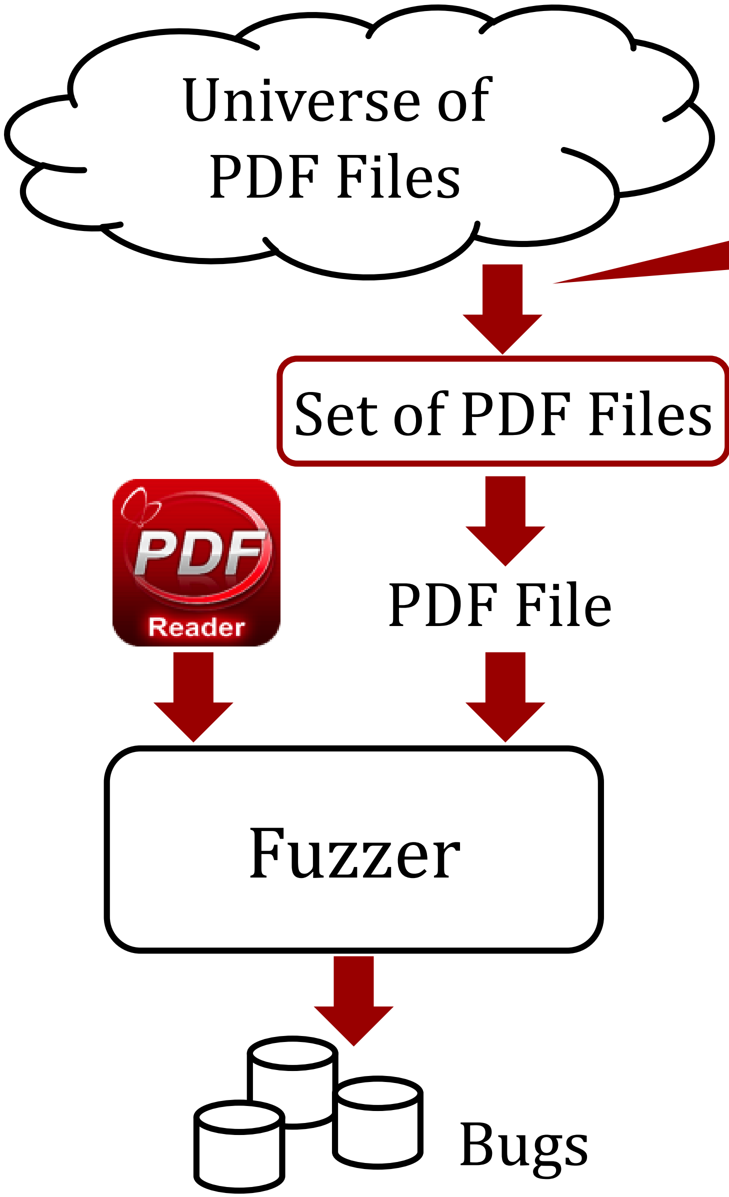


Goal: find as many bugs as possible

Research Questions

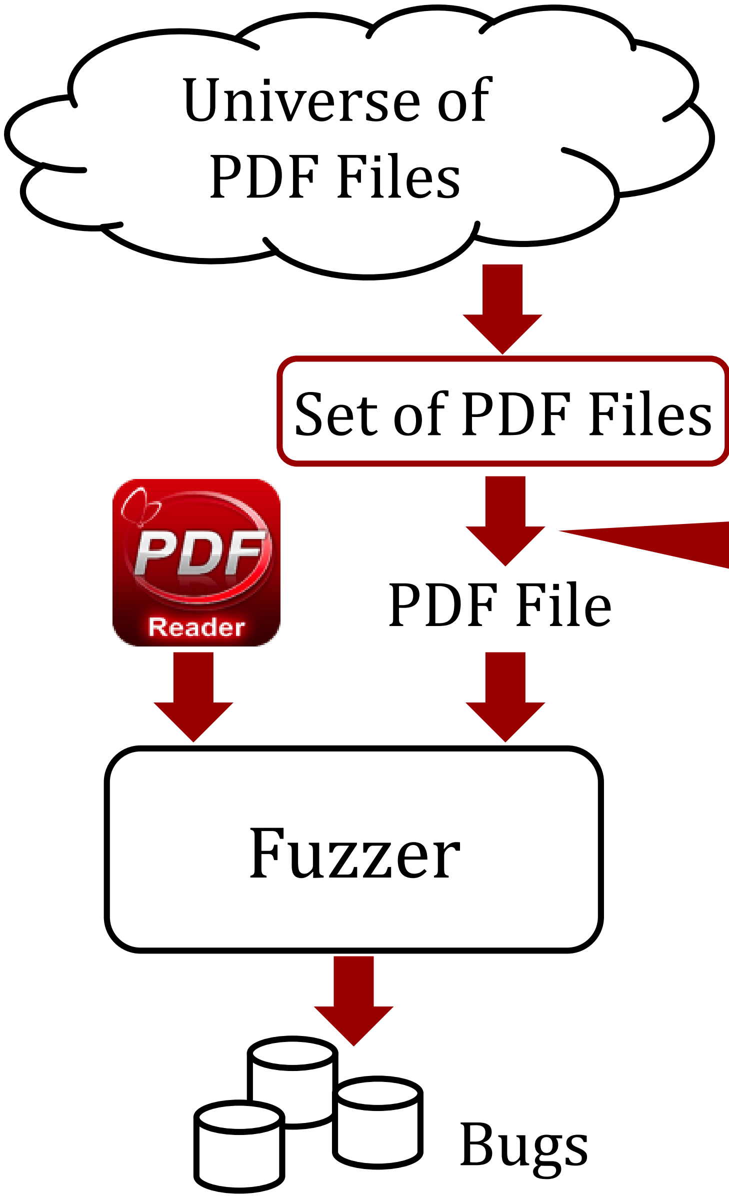


Research Questions



Question #1
How to select seeds to use?

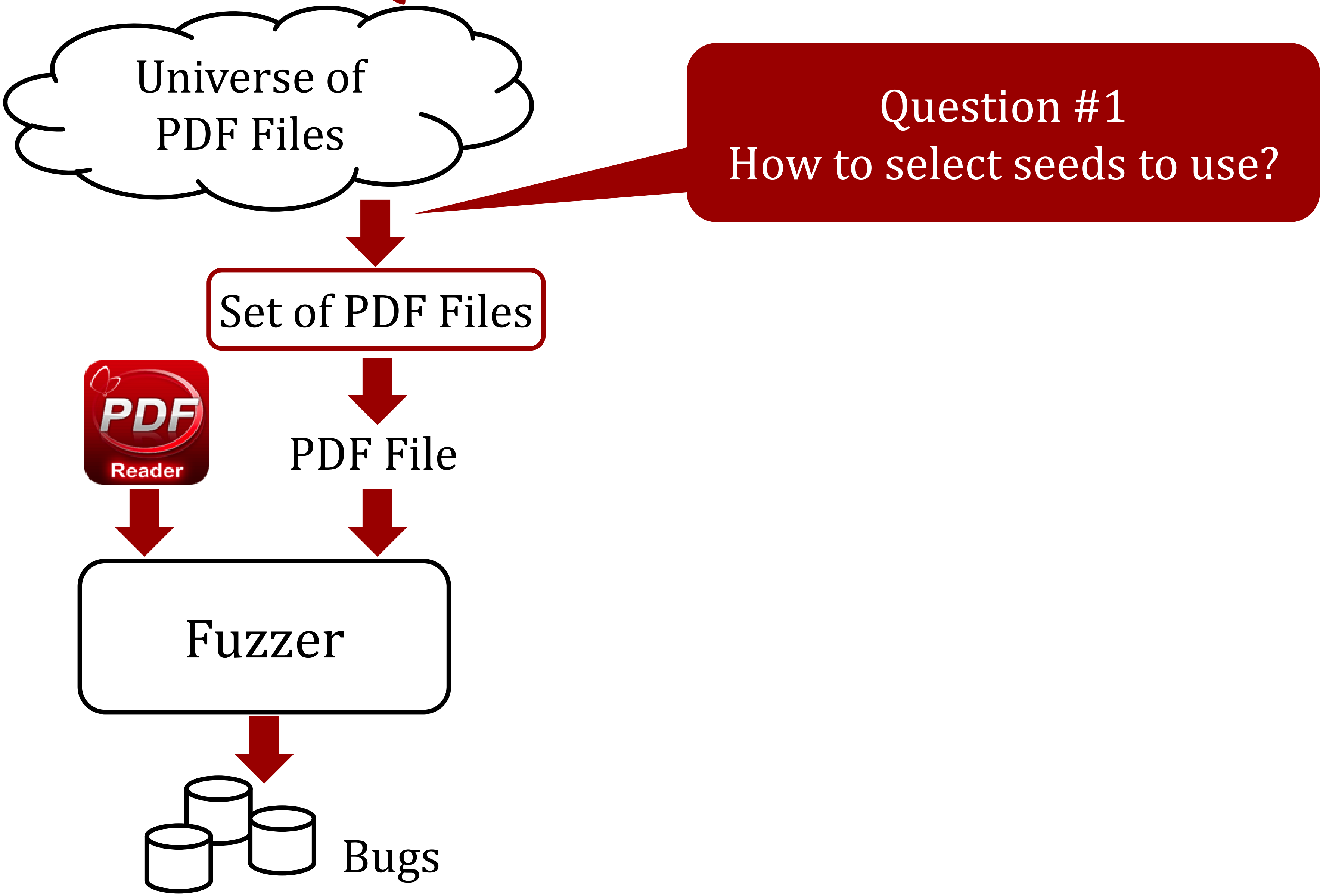
Research Questions



Question #2
How to schedule seeds?
Can we obtain the maximum # of bugs that can be found for a given set of seeds?

#bugs found = #unique crashes identified by stackhash

Q1: Seed Selection



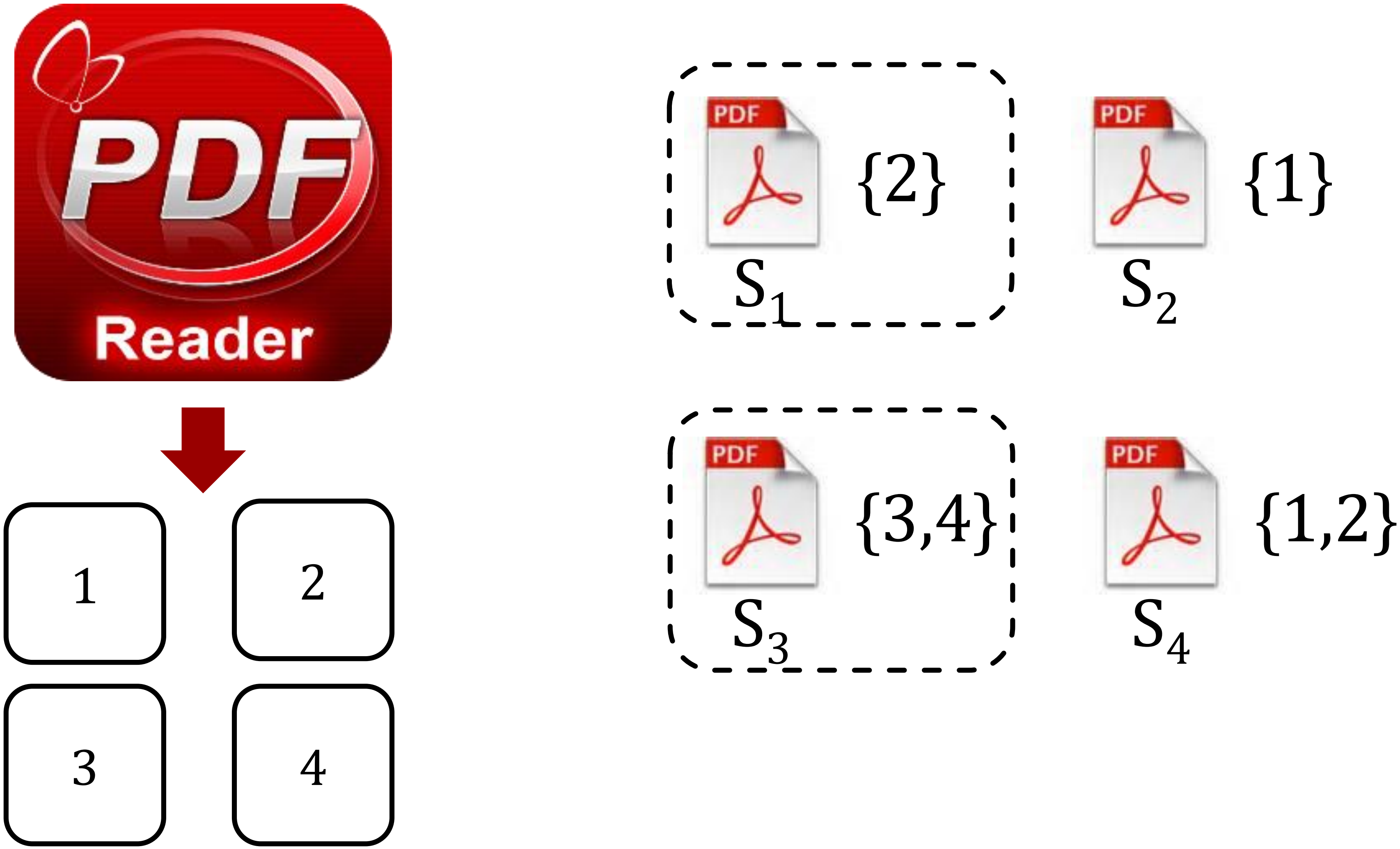
Find a Set of Seeds Maximizing Code Coverage

- Miller reports an 1% increase in code coverage increases the percentage of bugs found by 0.92%[1]
- Peach uses code coverage to select seeds[2]

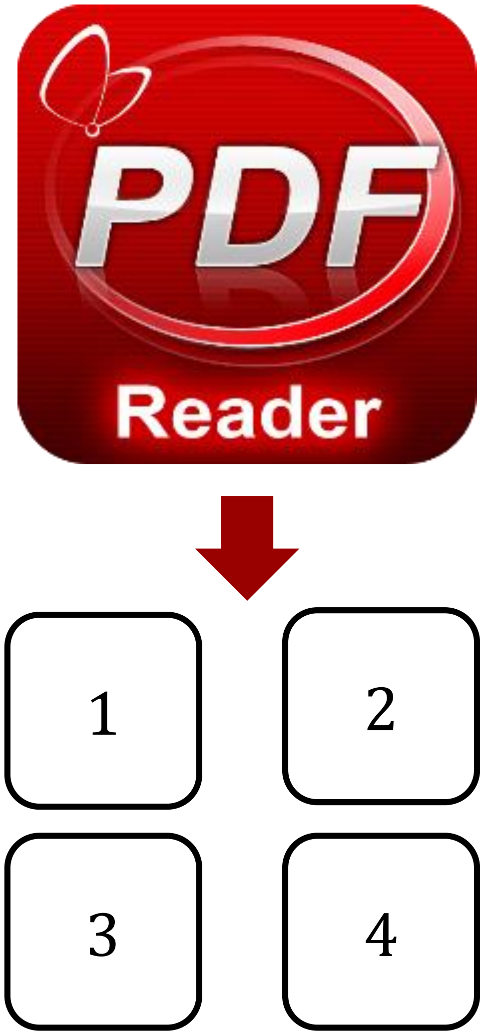
Minimal Set-Cover Problem

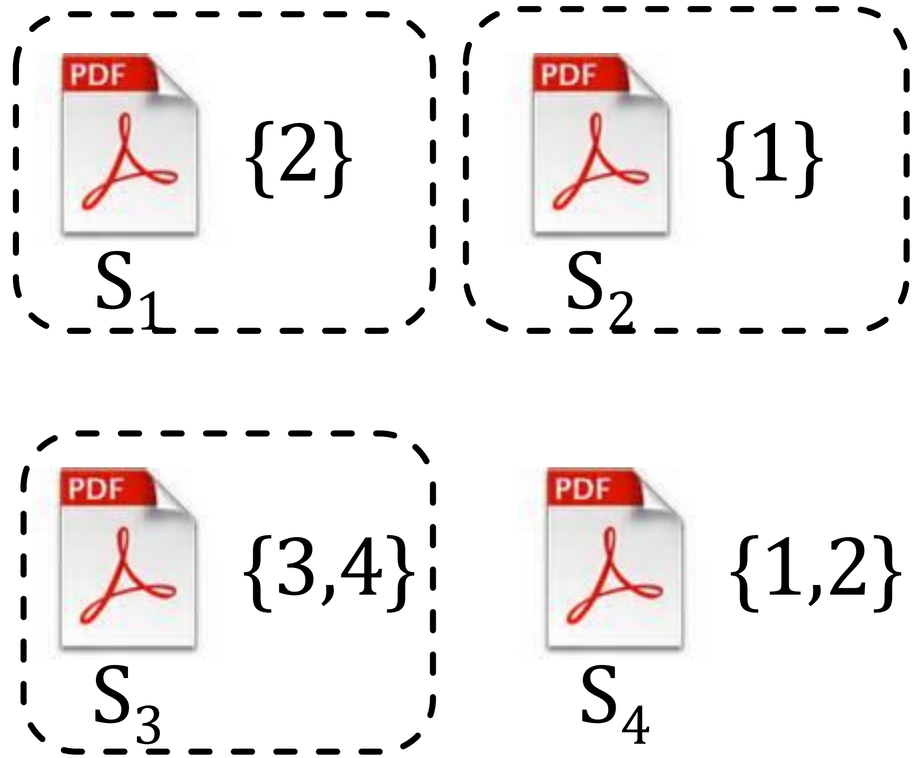
[1] Fuzz by Number, CanSecWest 2008
[2] http://peachfuzzer.com

Minimal Set-Cover Problem (MSCP)

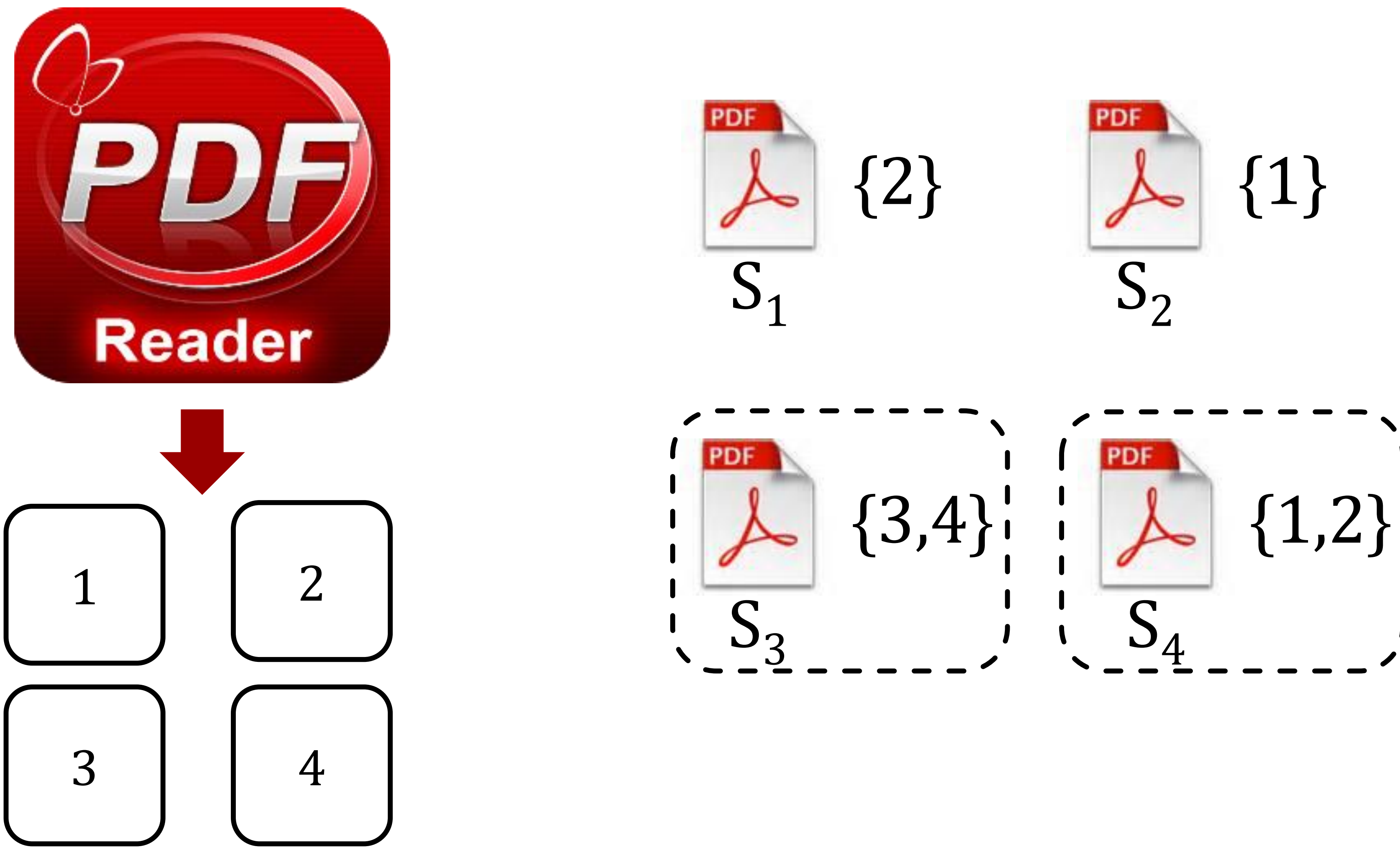


Minimal Set-Cover Problem (MSCP)





Minimal Set-Cover Problem (MSCP)



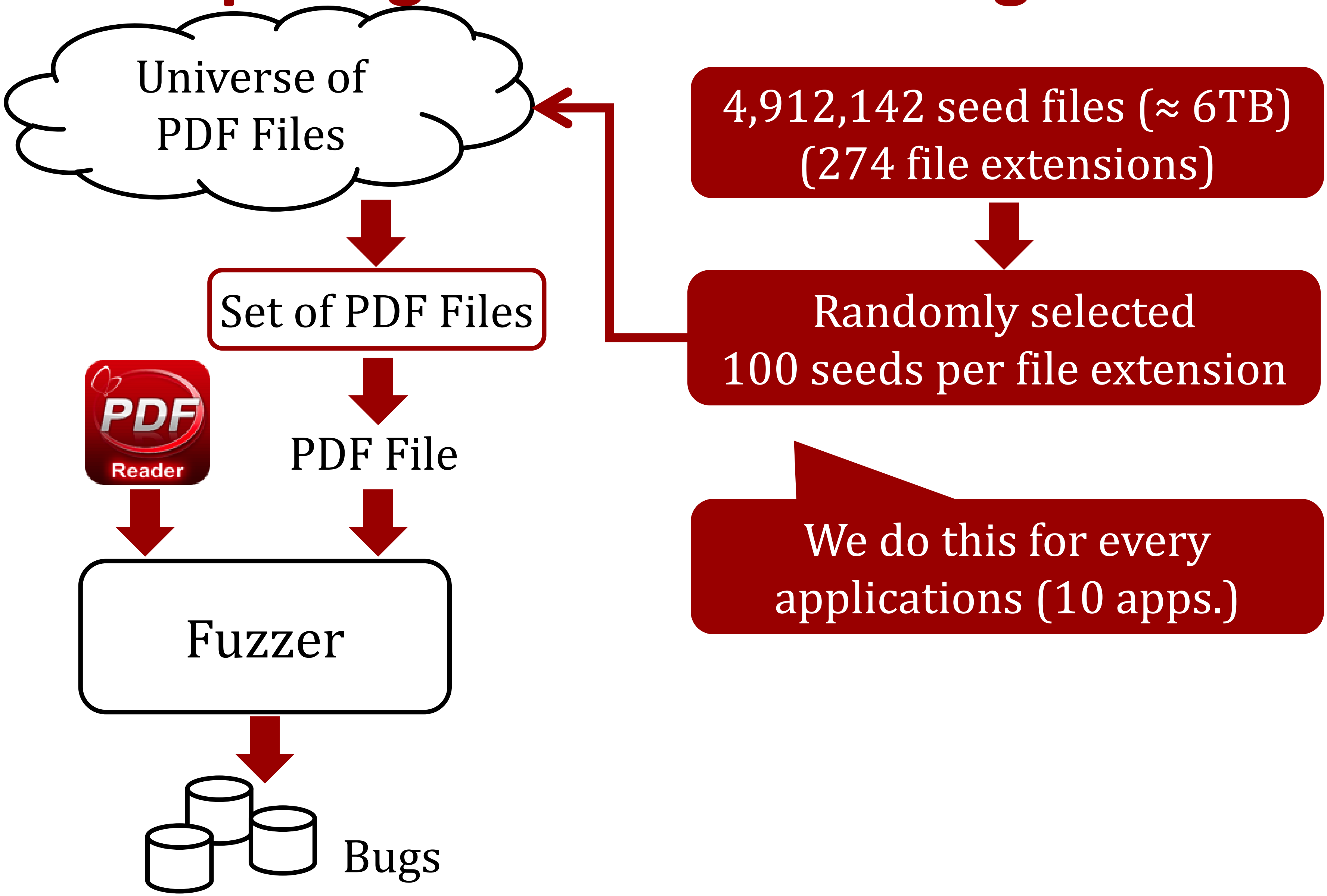
MSCP is NP-Hard, But

We use a greedy polynomial-time approximation algorithm

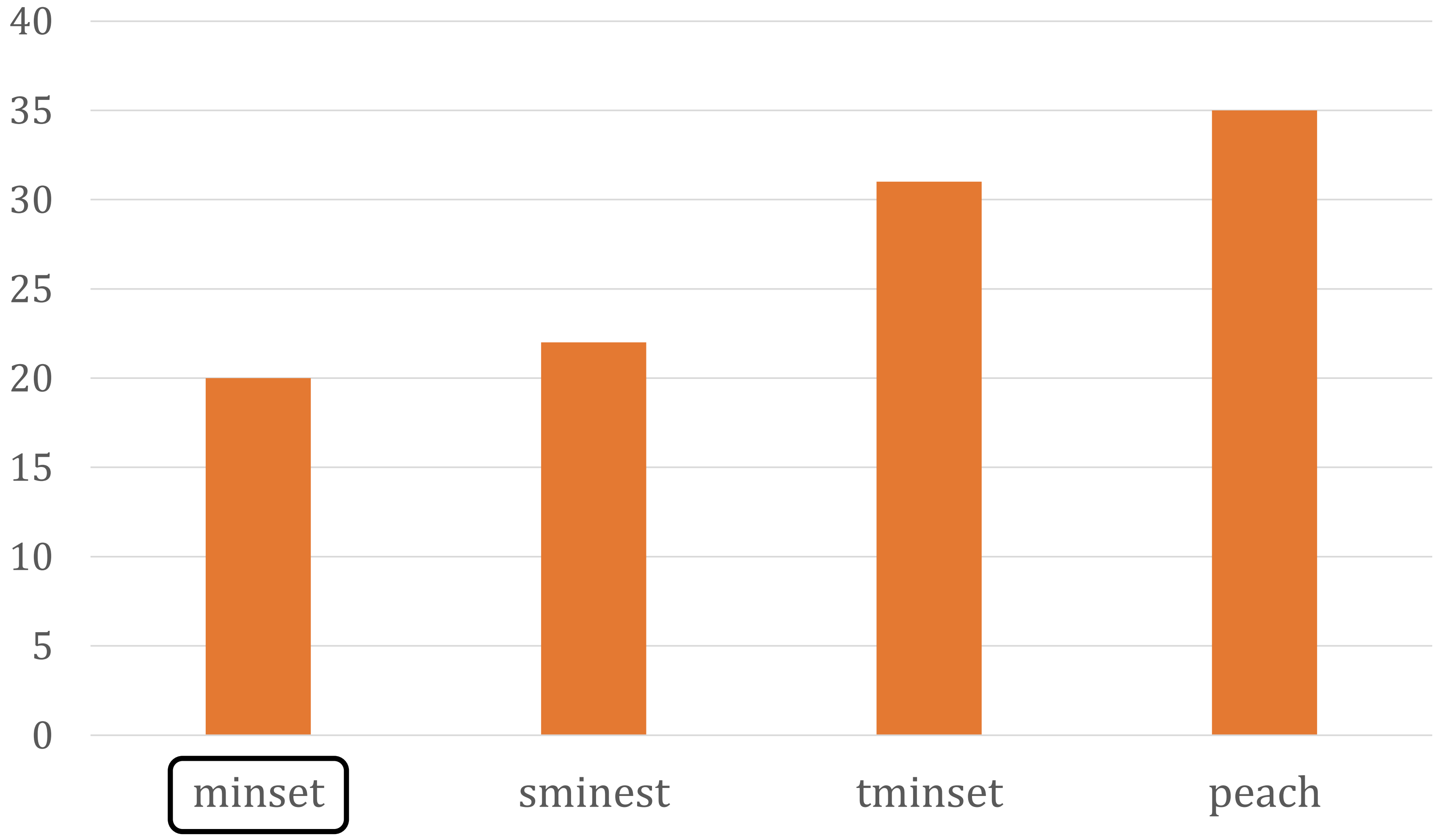
- **Unweighted MinSet**: MSCP
- **Time MinSet**: Weighted MSCP with exec. time
- **Size MinSet**: Weighted MSCP with seed file size
- **Peach Set**: derived from peach fuzzer

More details in the paper

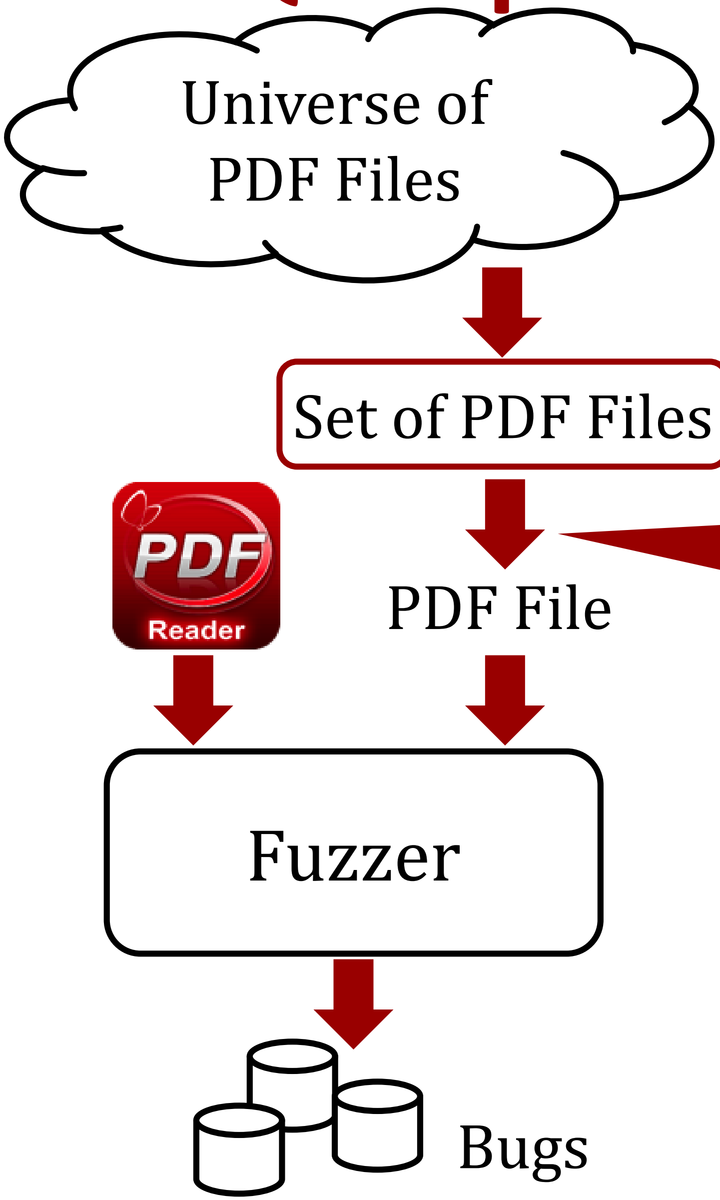
Comparing Seed Selection Algorithms



#Seeds after Seed Selection (From 100 Seeds)



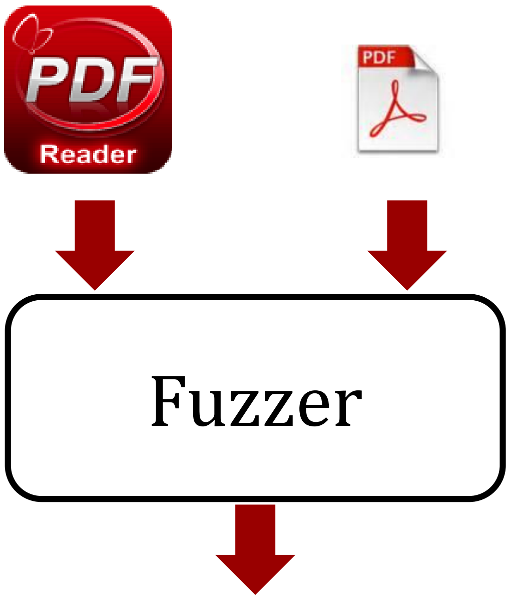
Q2: Optimal Seed Scheduling



Question #2
For a given set of seeds, what is the maximum # of bugs that can be found within a time limit?

We introduce a methodology of evaluating seed selection algorithms

Compute Optimal Scheduling from Collected Ground Truth Data



Per-Seed
Ground Truth Collection

Ground Truth = a sequence of (**bug ID**, **seed ID**, **time stamp**)

$(B_1, S_1, T_1), (B_2, S_1, T_2), \ldots$

Compute Optimal Scheduling from Collected Ground Truth Data

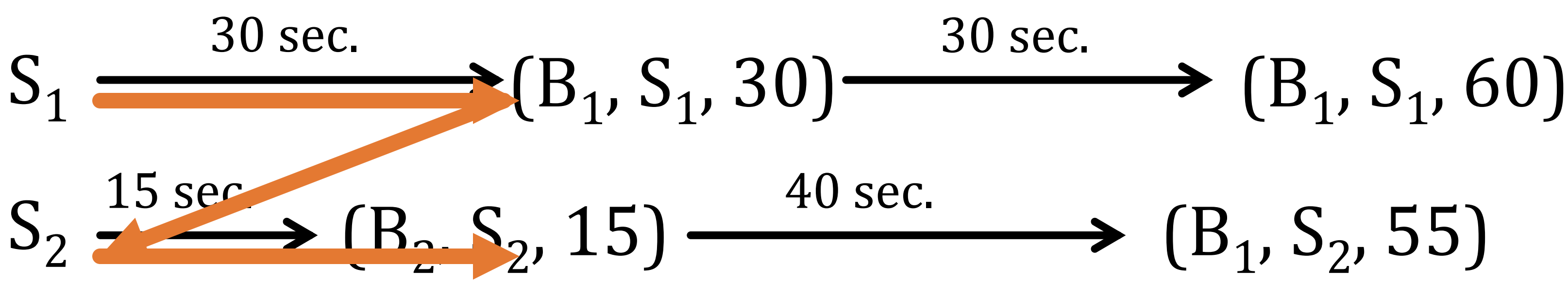
For all the seeds in the universe

$$\begin{cases} (B_1, S_1, T_1), (B_2, S_1, T_2), \ldots \\ (B_4, S_2, T_1), (B_2, S_3, T_2), \ldots \\ \ldots \\ (B_4, S_2, T_1), (B_2, S_3, T_2), \ldots \end{cases}$$

Finding an optimal scheduling is NP-hard

⇒ ***ILP*** (Integer Linear Programming)

ILP Formulation Example



- Fuzzing 1 program with 2 seed files (S_1 and S_2)
- 1 minute fuzzing run with each seed
- 2 bugs found in total (B_1 and B_2)

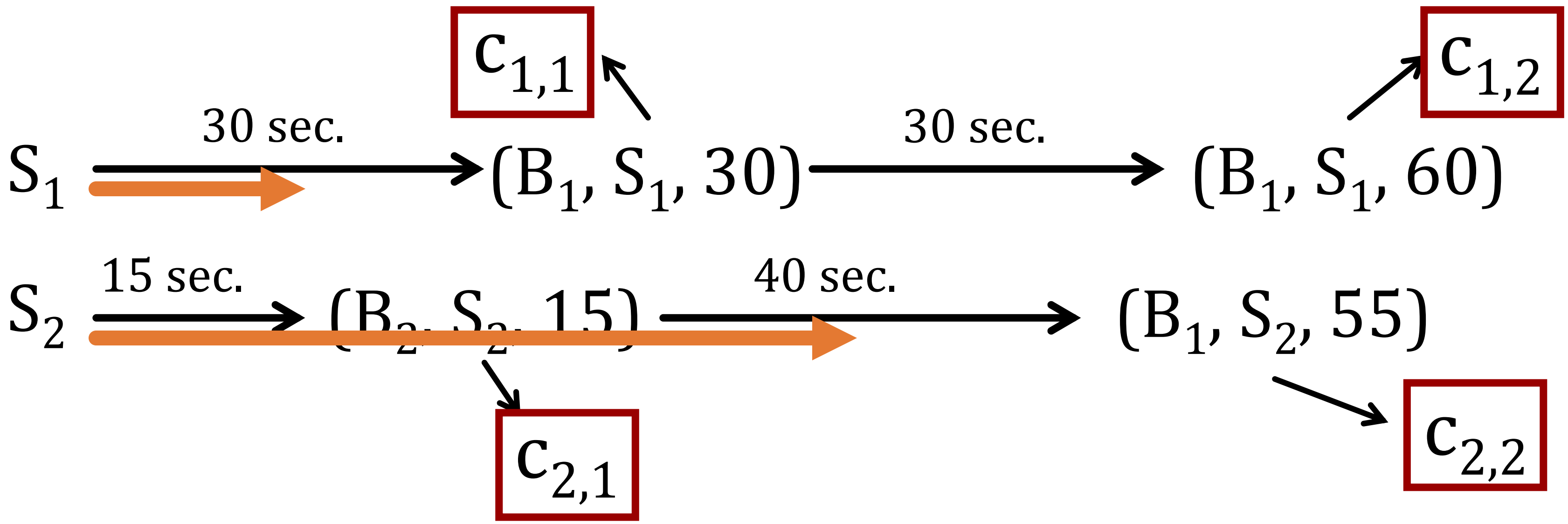
Steps in ILP Formulation

1. Define the goal

Maximize the # of Bugs

2. Define ILP variables
3. Define constraints over the variables

Introducing Crash Indicator Variable $c_{i,j}$



$c_{i,j}$ = j^{th} crash in the i^{th} seed

If we select S_1 for 15 sec., then $c_{1,1} = 0$, $c_{1,2} = 0$

If we select S_2 for 40 sec., then $c_{2,1} = 1$, $c_{2,2} = 0$

Introducing Time Variable $t_{i,j}$

$\boxed{t_{1,1}}$ $c_{1,1}$ $\boxed{t_{1,2}}$ $c_{1,2}$

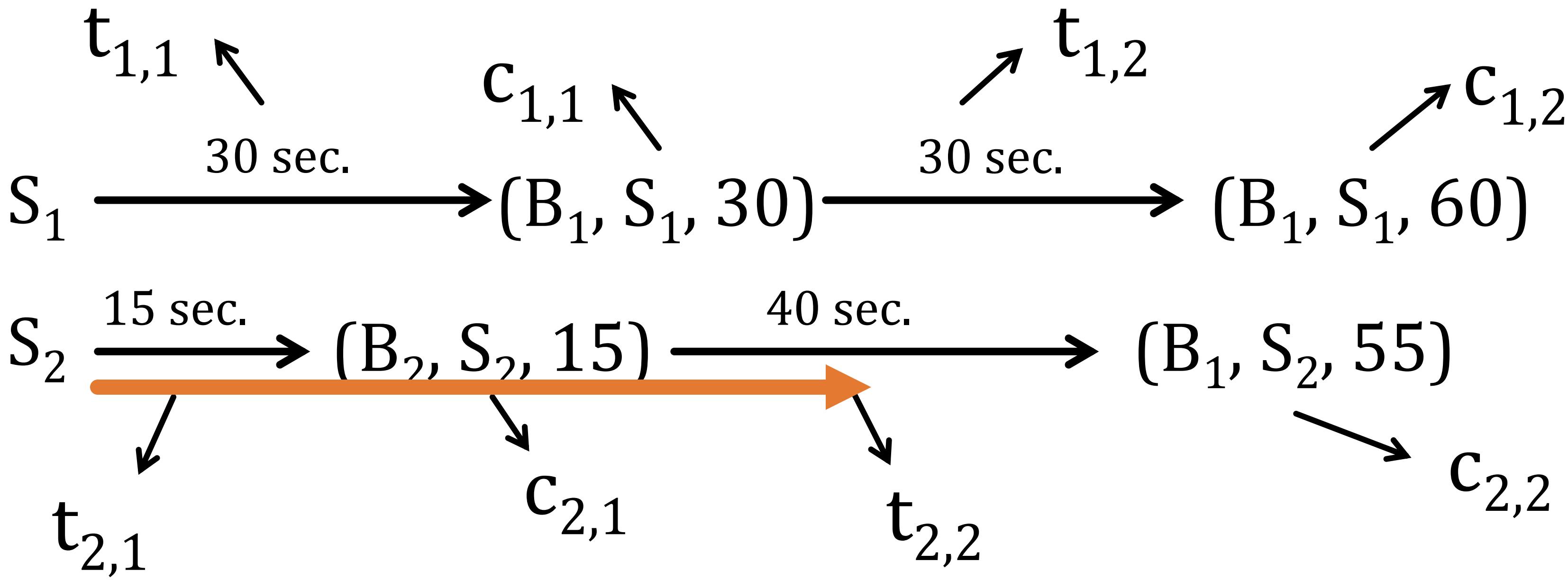
$$S_1 \xrightarrow{30\text{ sec.}} (B_1, S_1, 30) \xrightarrow{30\text{ sec.}} (B_1, S_1, 60)$$

$$S_2 \xrightarrow{15\text{ sec.}} (B_2, S_2, 15) \xrightarrow{40\text{ sec.}} (B_1, S_2, 55)$$

$\boxed{t_{2,1}}$ $c_{2,1}$ $\boxed{t_{2,2}}$ $c_{2,2}$

$t_{i,j}$ = j^{th} time interval of the i^{th} seed

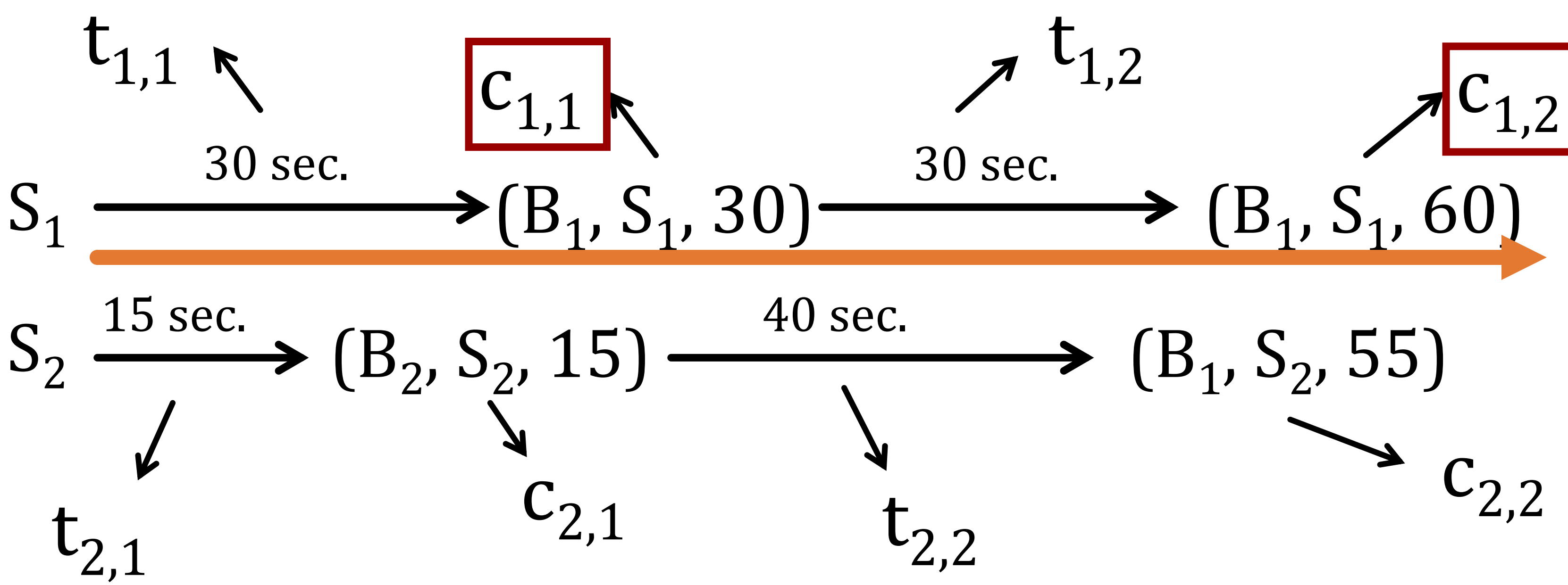
Introducing Bug Indicator Variable b_x



$$\boxed{b_x} = 1 \text{ iff } \exists\, i, j : \boxed{\mu(c_{i,j}) = x}$$

If we select S_2 for 40 sec., $b_2 = 1$

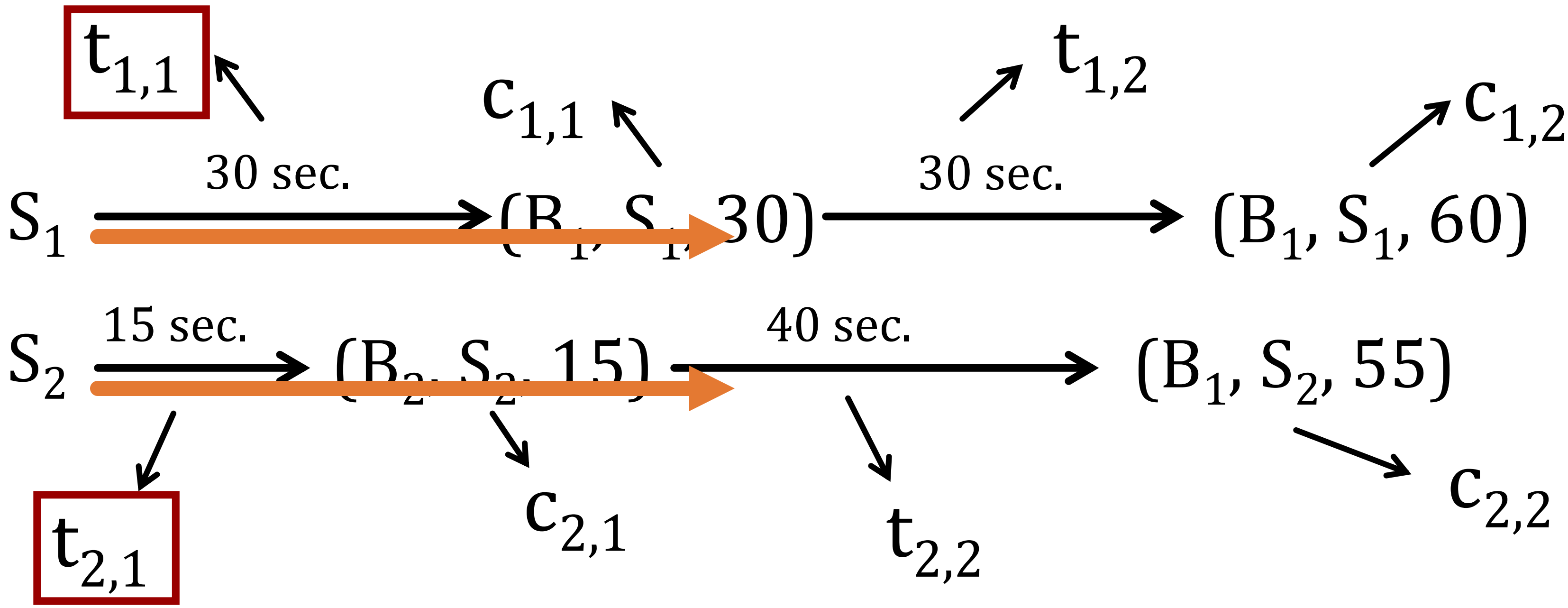
Constraint 1: Order of Crashes



$$\forall_{i,j} . c_{i,j+1} \leq c_{i,j}$$

Preserve
the order of crashes

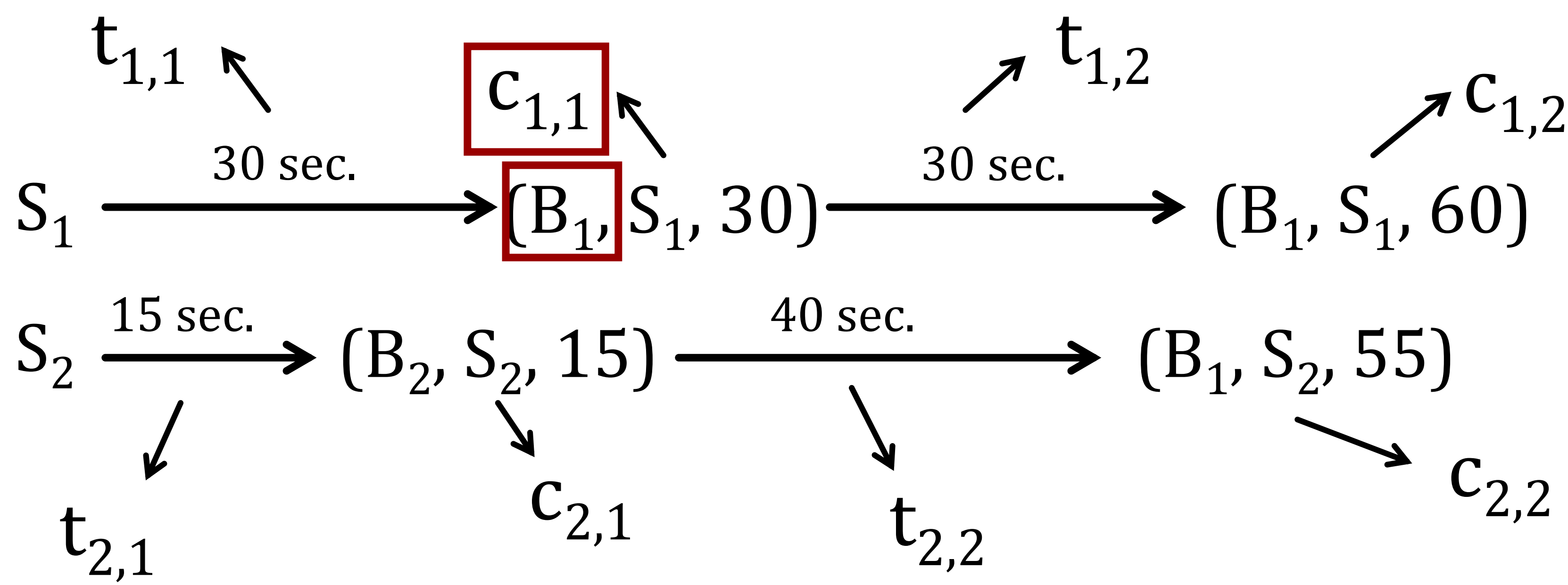
Constraint 2: Time Limit



$$\sum_{i,j} c_{i,j} \cdot t_{i,j} \leq t_{\text{thres}}$$

Do not exceed
the time limit

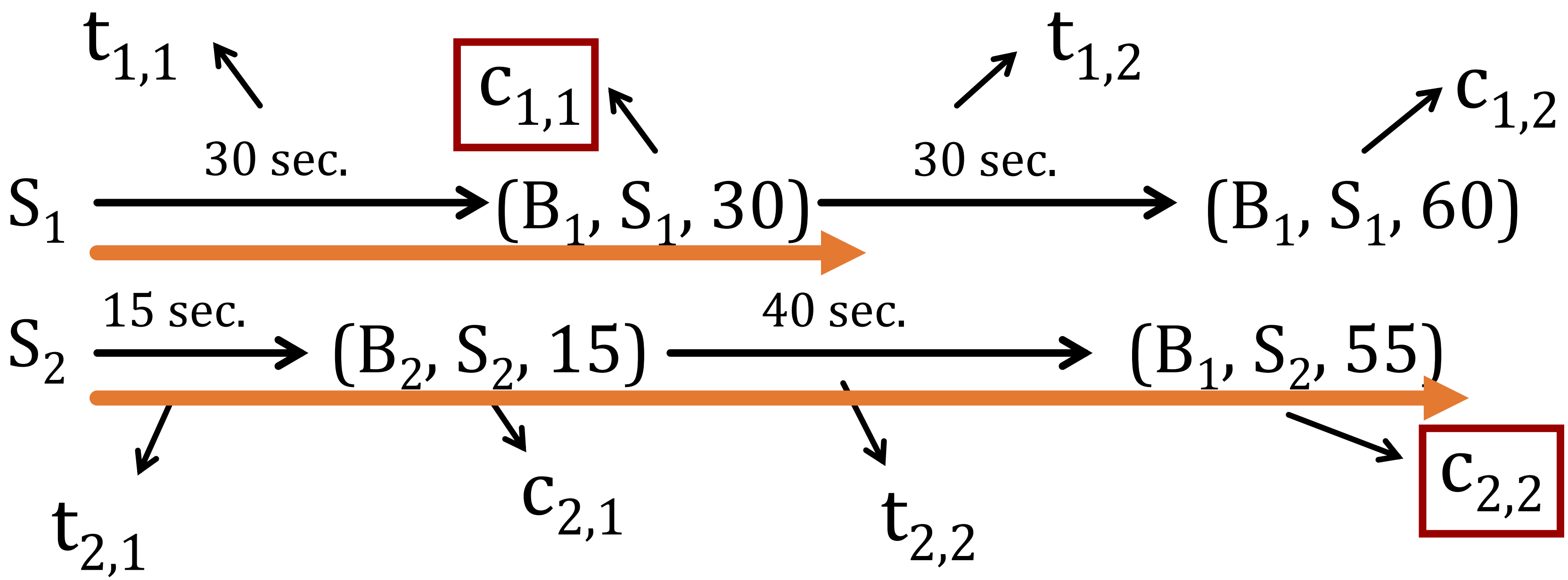
Constraint 3: Crash ⇒ Bug



$$\boxed{\forall_{i,j} .\, c_{i,j} \leq b_x \text{ where } \mu(c_{i,j}) = x}$$

If a crash is found, then the corresponding bug is found

Constraint 4: Bug ⇒ Crash



$$\boxed{\forall_{x} . b_x \leq \sum_{i,j} c_{i,j} \text{ where } \mu(c_{i,j}) = x}$$

If a bug is found, then
one of the corresponding crashes is found

Final ILP Formulation

maximize $\boxed{\sum_x b_x}$ maximize # of bugs found

subject to

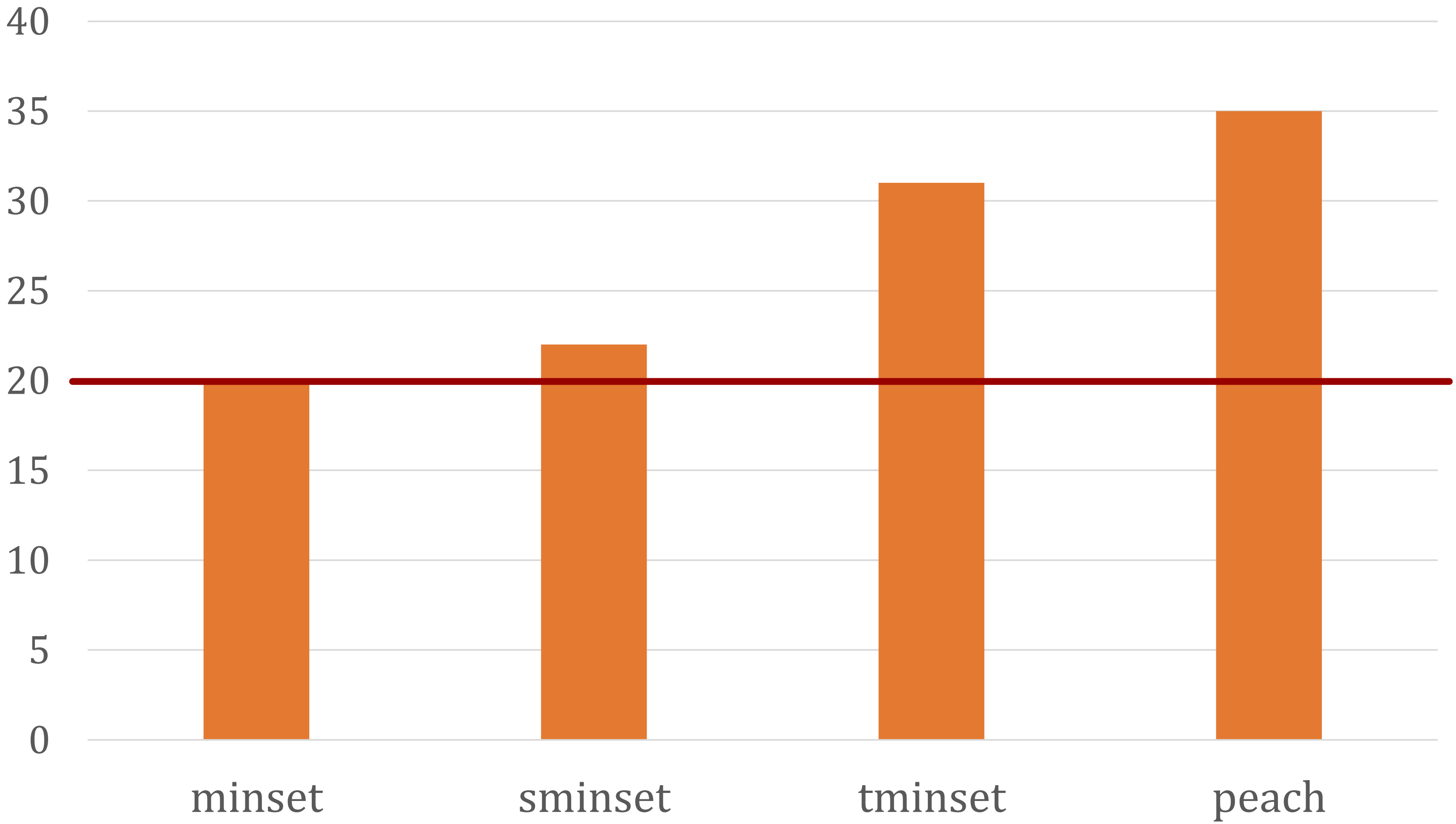
$$\boxed{\forall_{i,j} . c_{i,j+1} \leq c_{i,j}}$$

$$\boxed{\sum_{i,j} c_{i,j} \cdot t_{i,j} \leq t_{\text{thres}}}$$

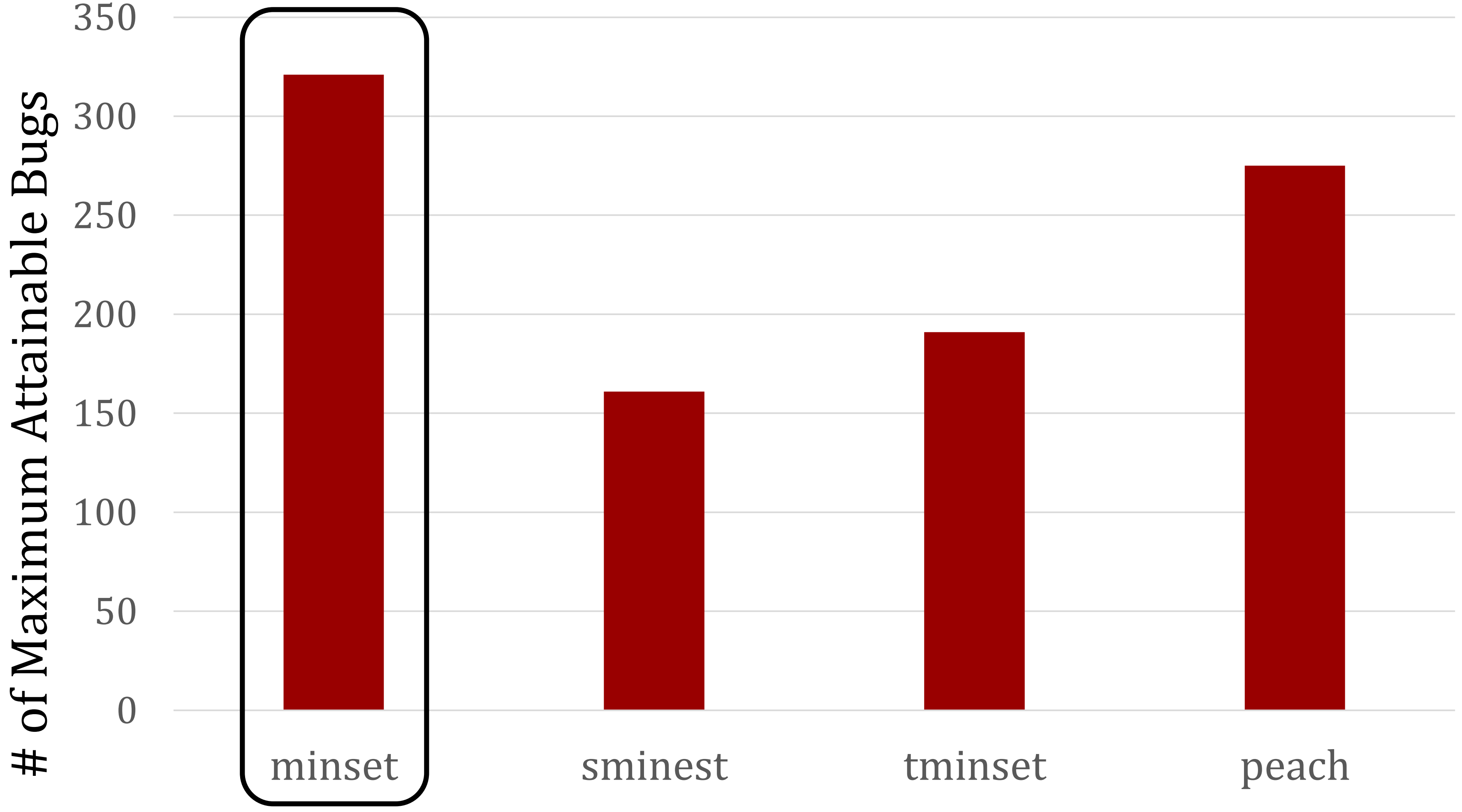
$$\boxed{\forall_{i,j} . c_{i,j} \leq b_x \text{ where } \mu(c_{i,j}) = x}$$

$$\boxed{\forall_x . b_x \leq \sum_{i,j} c_{i,j} \text{ where } \mu(c_{i,j}) = x}$$

#Seeds after Seed Selection (From 100 Seeds)



of Maximum Attainable Bugs using 20 Seeds over 10 Apps.



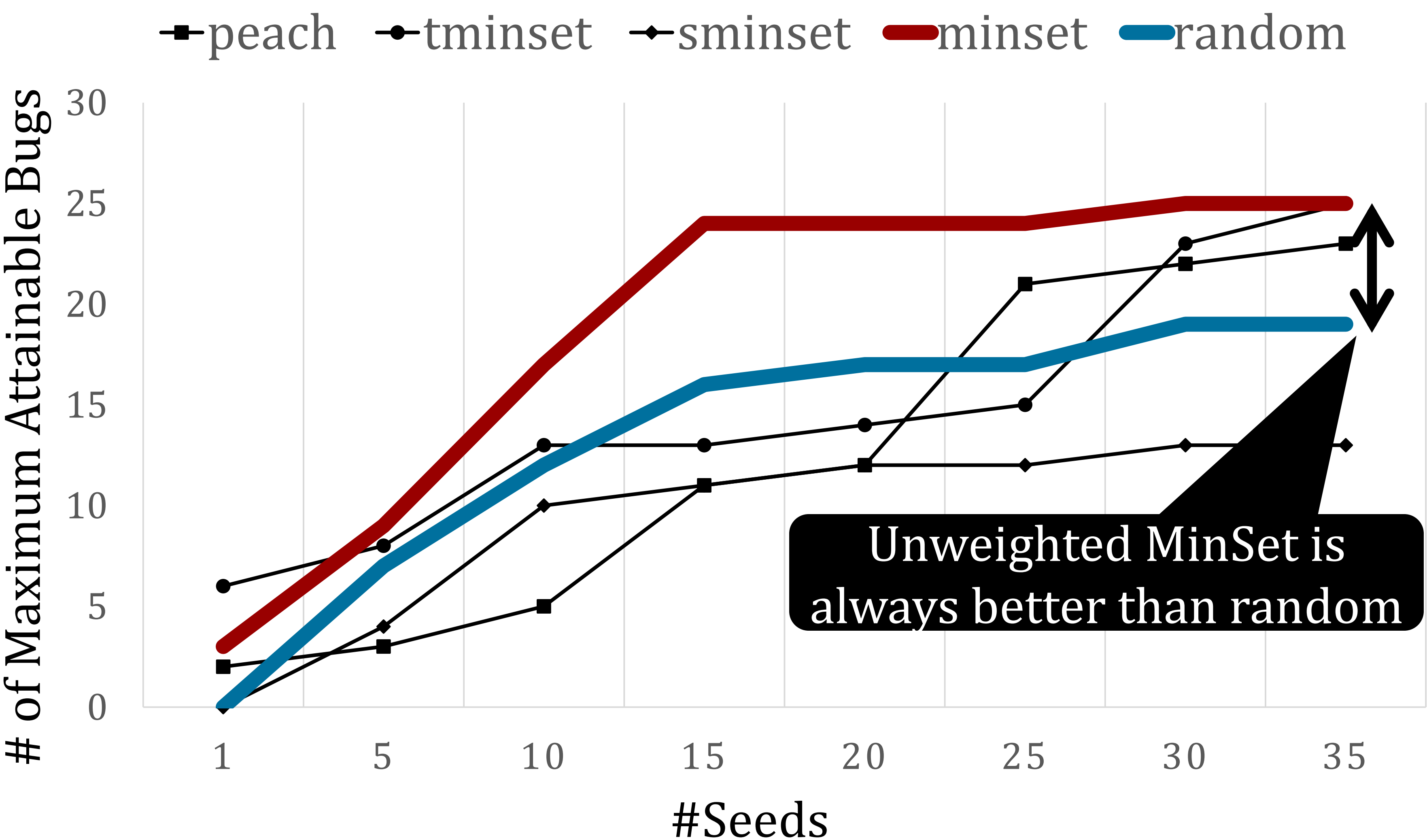
Comparing Seed Selection Algorithms Against Random Set

- **Random Set**: pick *k* seeds at random
- **Unweighted MinSet**: MSCP
- **Time MinSet**: WMSCP with execution time
- **Size MinSet**: WMSCP with seed file size
- **Peach Set**: derived from peach fuzzer

Simulated random set ***1000*** times per program

Compare # of bugs found per *k*

Unweighted MinSet Performs Best



More on the Paper

- Detailed seed selection algorithms
- Detailed ILP formulation
- More evaluation

Conclusion

- We formalized, implemented, and tested a number of seed selection algorithms for fuzzing
- We introduced a ***methodology*** for evaluating seed selection algorithms for fuzzing

Thank You

Sang Kil Cha

sangkilc@cmu.edu

Code & Data will be soon available:

http://security.ece.cmu.edu/coverset