Overview of Software Debugging

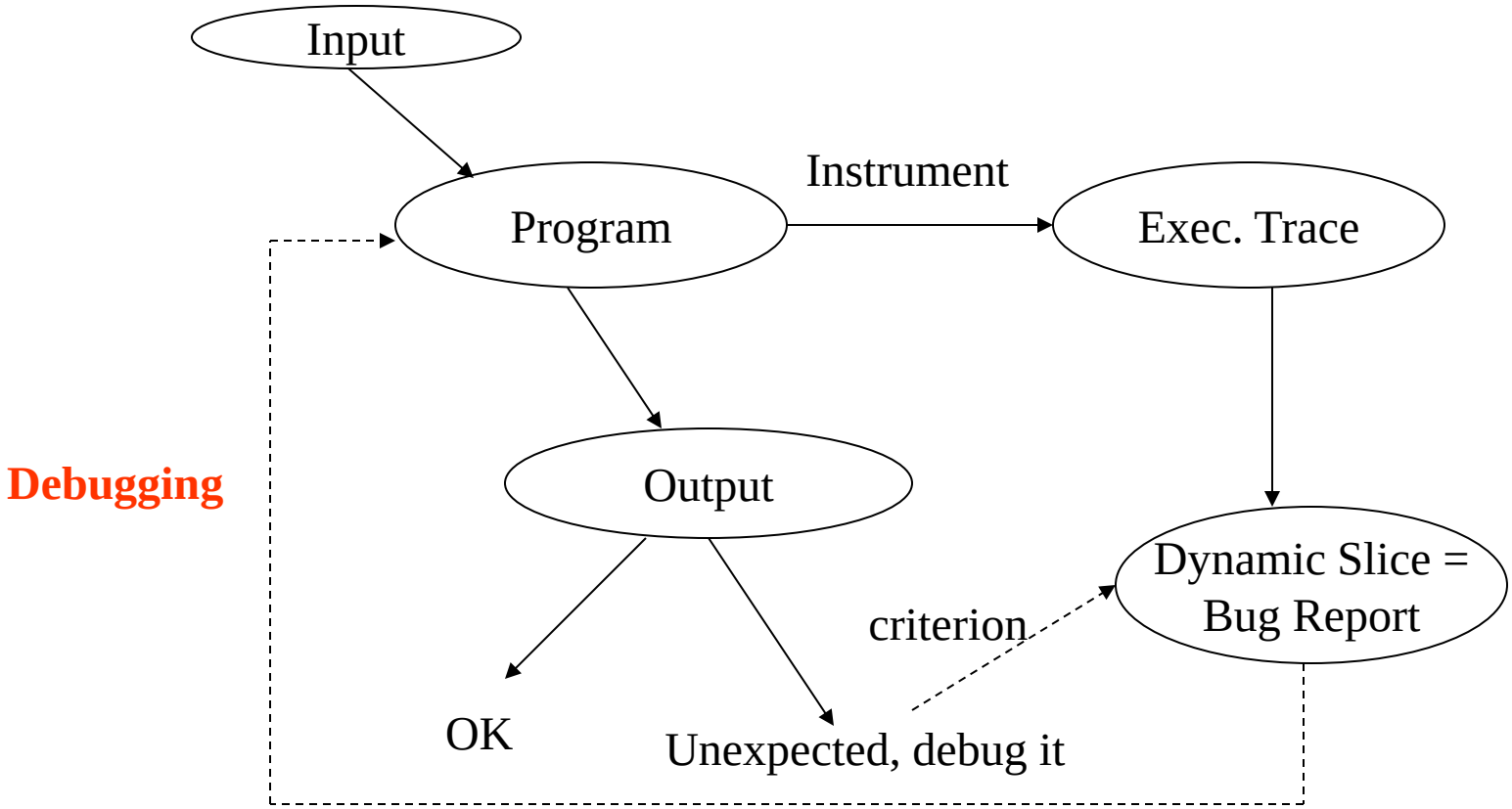
Abhik Roychoudhury
National University of Singapore
abhik@comp.nus.edu.sg

In the next 20 minutes

- Tutorial like talk on software debugging from Software Engineering perspective
  - Program Dependencies and Slicing
  - Delta debugging
    - Trace based Fault localization
  - Statistical Fault localization

- Some recent research results
  - Debugging software regressions
  - Large scale experiments – embedded Linux Busybox

Dynamic Slicing for Debugging



Dynamic Slicing

Consider input a == 2

```
1    b=2;
2    y=1;
3    If (a>1){
4        if (b>1){
5            x=2;
         }
     }
6    printf ("%d", x);
```

Control Dependence

Data Dependence

Slicing Criterion

Problem with dynamic slicing

- Huge overheads
  - Backwards slicing requires trace storage.
  - Jslice tool for Java
    - Online trace compression & traversal
    - http://jslice.sourceforge.net
- Dynamic Slice is still too large …
  - … for human comprehension
  - Interleave computation and comprehension
- Dynamic Slice can also be too small!
  - What do I mean here?

Relevant Slicing

input: a=2

Source of Failure

Dynamic Slice

Execution is omitted

```
1   b=1;
2   x=1;
3   If (a>1){
4       if (b>1){
5           x=2;
        }
    }
6   printf ("%d", x);
```

Relevant Slice

Captures statements which affect the output by *not* getting executed

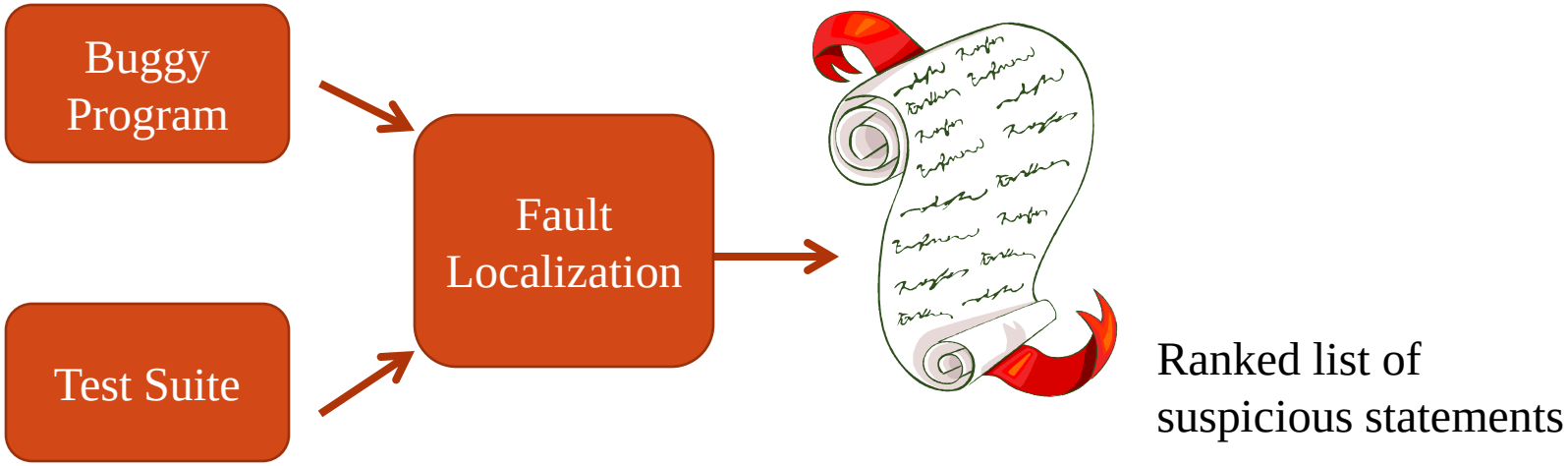
input: a=2

```
1   b=1;
2   x=1;
3   If (a>1){
4       if (b>1){
5           x=2;
        }
    }
6   printf ("%d", x);
```

Potential Dependence

Dynamic Data Dependence

Statistical Fault localization



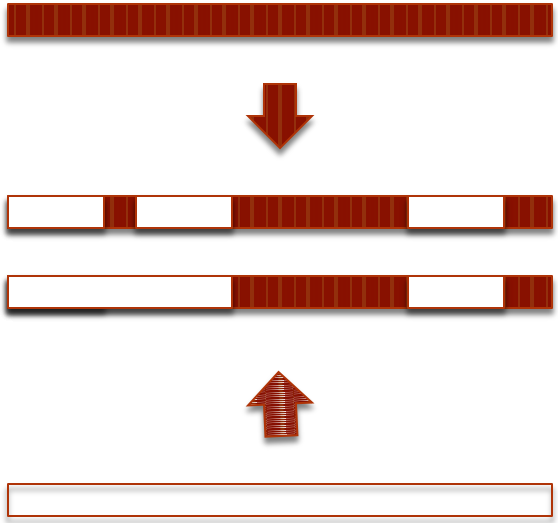
Assign scores to program statements based on their occurrence in passing / failing tests. ***Correlation equals causation!***

$$\text{Score}(s) = \frac{\frac{\text{fail}(s)}{\text{allfail}}}{\frac{\text{fail}(s)}{\text{allfail}} + \frac{\text{pass}(s)}{\text{allpass}}}$$

An example of scoring scheme [Tarantula]

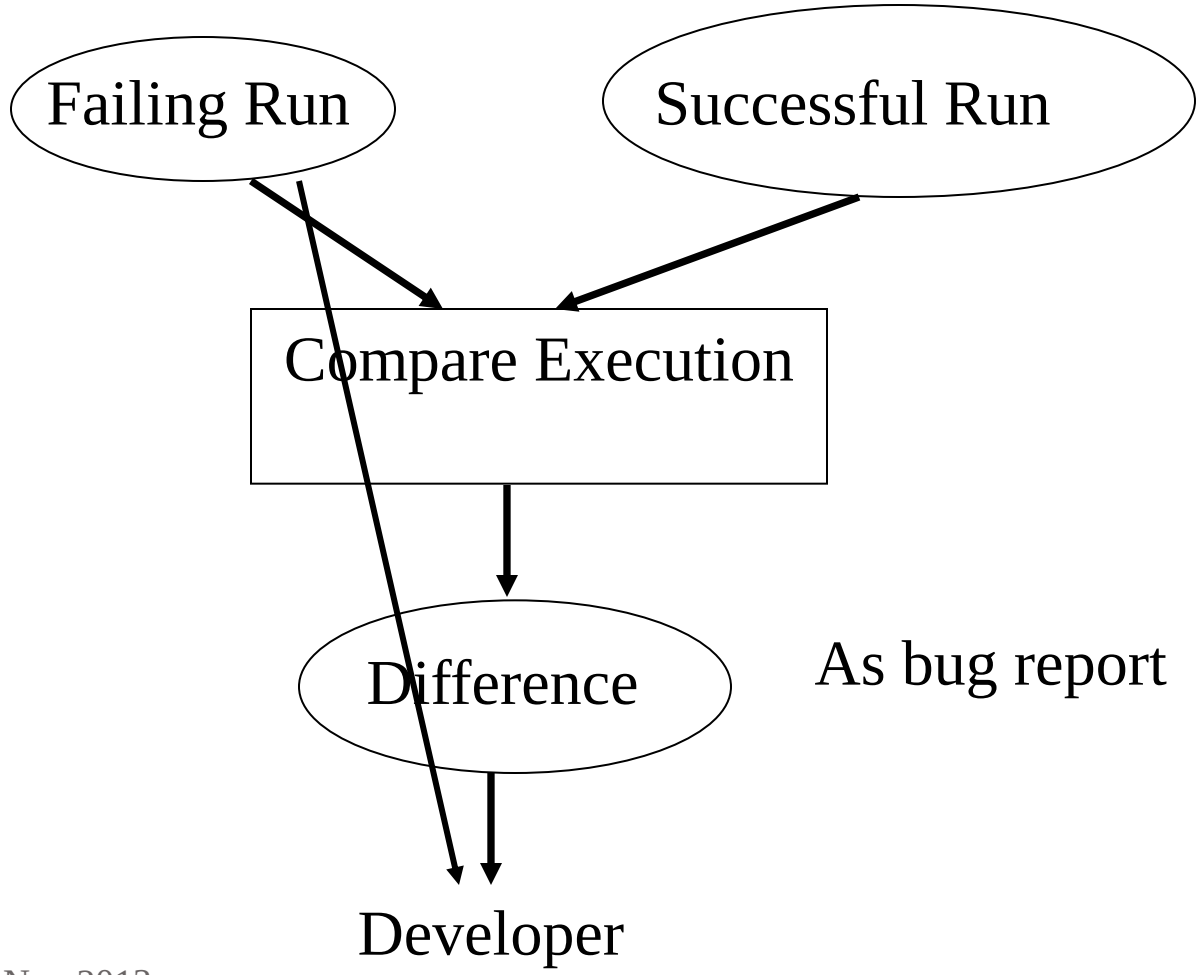
Isolating failure causes a la delta debugging

- How to figure out a minimal cause that ‘explains’ an error?
- Use a variation on binary search: narrow the difference between passing and failing inputs
  - Can do it on code (old version to new version)
  - On thread schedules



A. Zeller: Why Programs Fail, A Guide to Systematic Debugging

Fault Localization: overview



Comparing executions

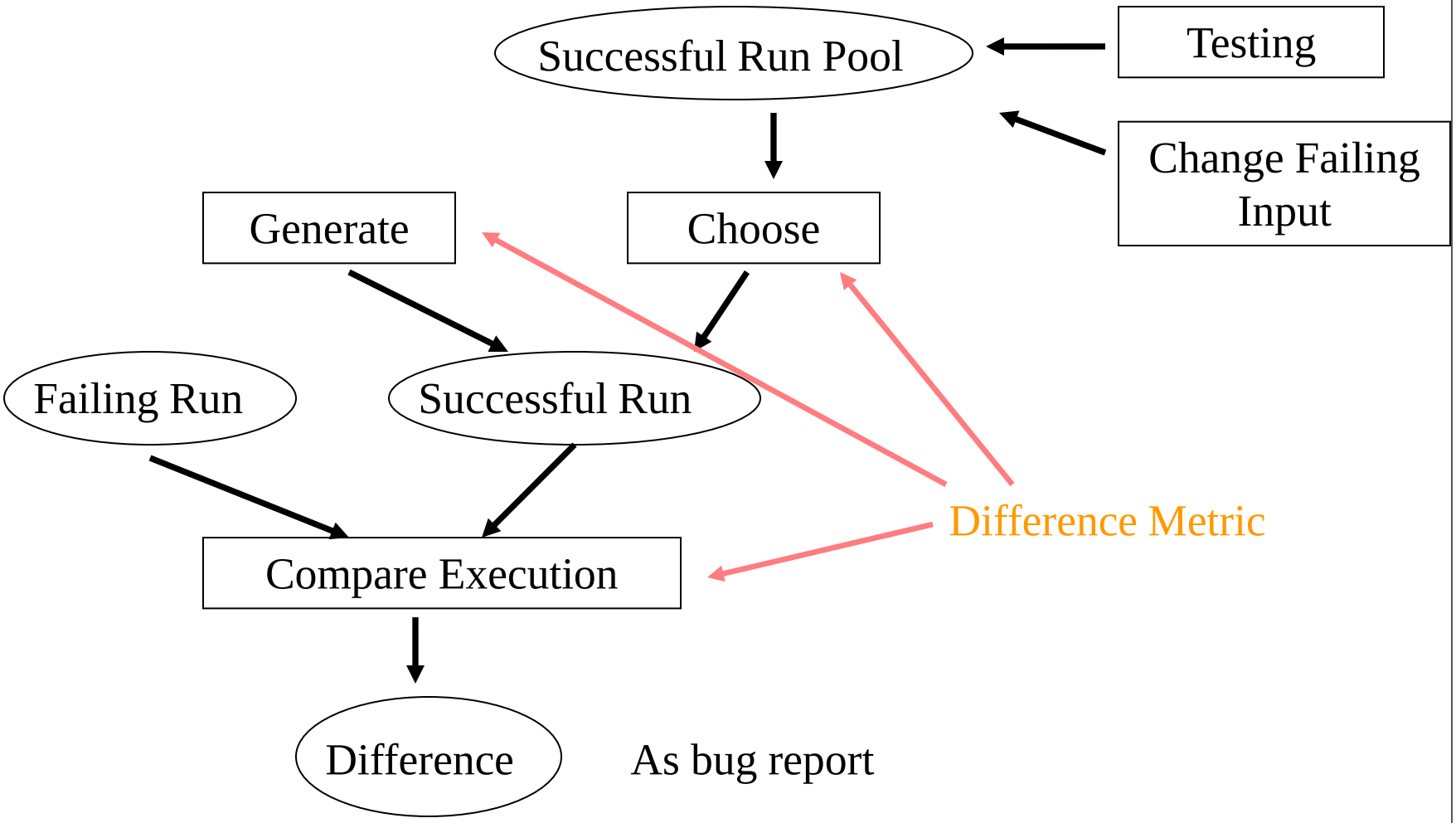
```
1 .   m=...
2.     if (m >= 0) {
3.         ...
4.         lastm = m;
5.     }
6.     …..
```

Failing run

```
1 .   m=...
2.     if (m >= 0) {
3.         ...
4.         lastm = m;
5.     }
6.     …..
```

Successful run

Fault localization



Comparing executions

```
1.  if (a)
2.      i = i + 1;
3.  if (b)
4.      j = j + 1;
5.  if (c)
6.      if (d)
7.          k = k + 1;
8.      else
9.          k = k + 2;
10. printf("%d", k);
```

Execution run π

```
1.  if (a)
2.      i = i + 1;
3.  if (b)
4.      j = j + 1;
5.  if (c)
6.      if (d)
7.          k = k + 1;
8.      else
9.          k = k + 2;
10. printf("%d", k);
```

Execution run π1

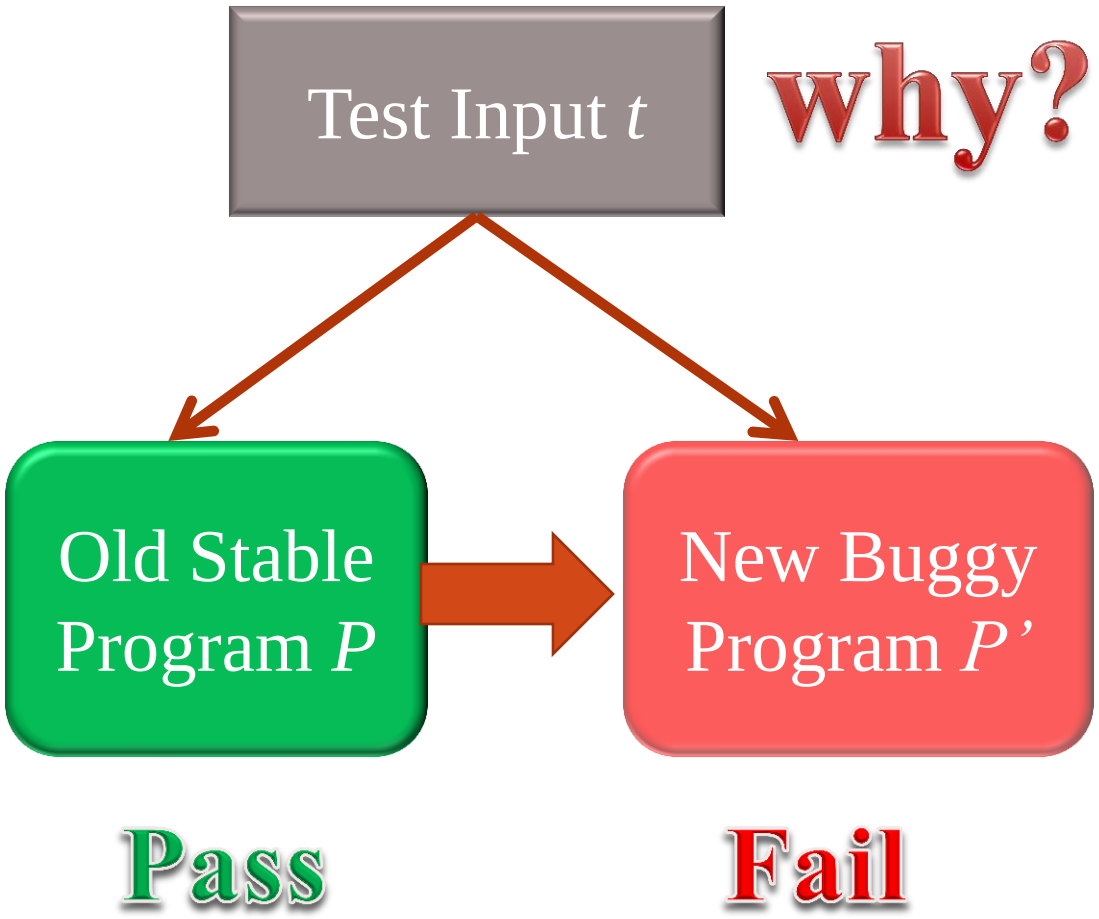
Set of statements

- S = Set of statements executed in π
  - {1,3,5,6,7,10}
- S1 = Set of statements executed in $\pi 1$
  - {1,3,4,5,6,9,10}
- If π is faulty and $\pi 1$ is OK
  - Bug report = S – S1 = {7}
- Choice of the execution run to compare with is very important.

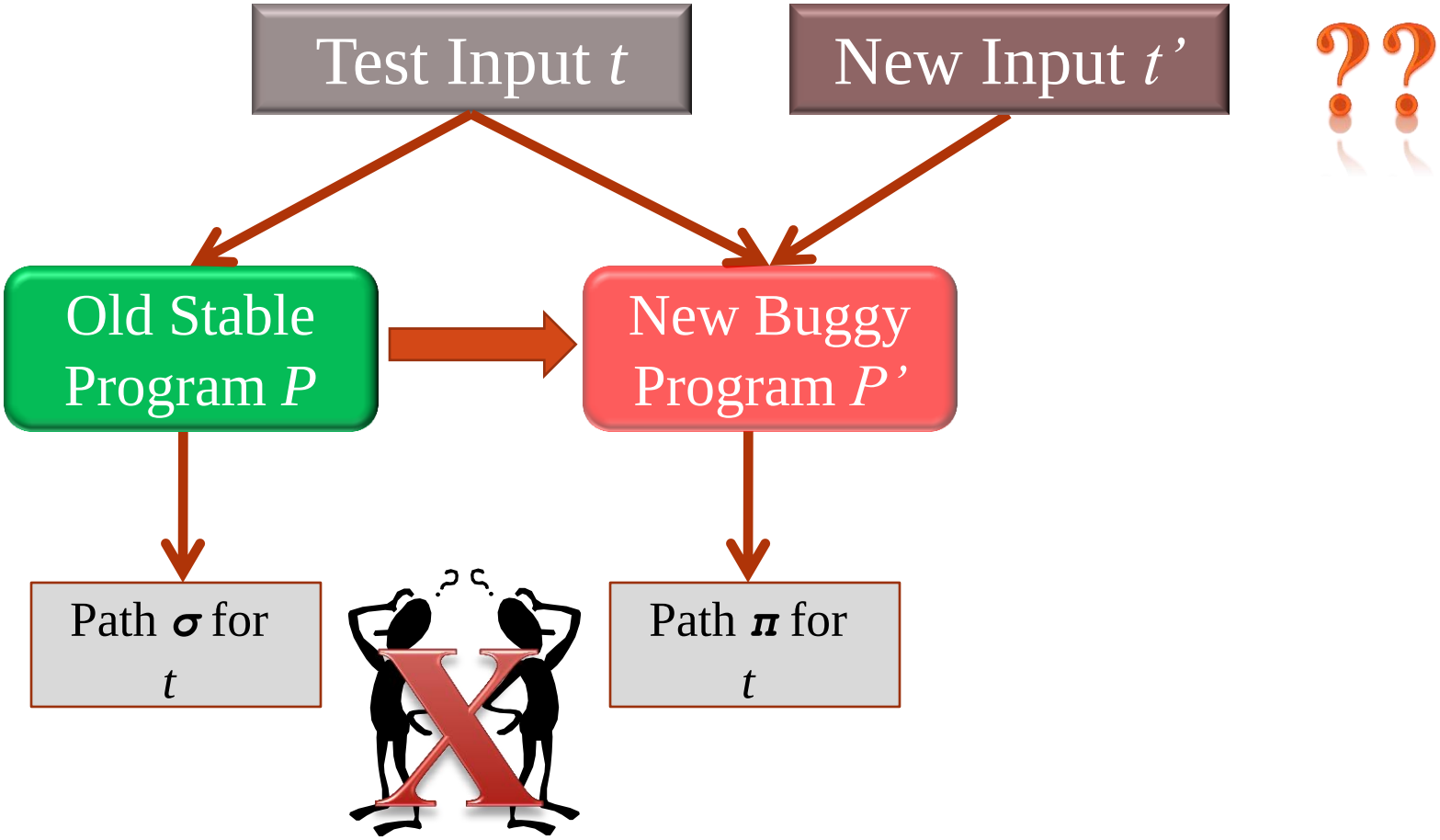
Trace alignment and differences

Execution Run			Alignment				Difference	
π	π'	π''	π	π'	π'	π''	$diff(\pi, \pi')$	$diff(\pi', \pi'')$
1^1	1^1	1^1						
2^2	2^2	2^2						
3^3	3^3	3^3					•	
4^4		4^4						
5^5		5^5						
7^6	7^4	7^6						•
8^7	8^5							
9^8	9^6							
		12^7						
1^9	1^7	1^8						
2^{10}	2^8	2^9						
3^{11}	3^9	3^{10}						
4^{12}	4^{10}	4^{11}						
5^{13}	5^{11}	5^{12}						
7^{14}	7^{12}	7^{13}					•	•
8^{15}								
9^{16}								
	12^{13}	12^{14}						
14^{17}	14^{14}	14^{15}						

Regression Debugging

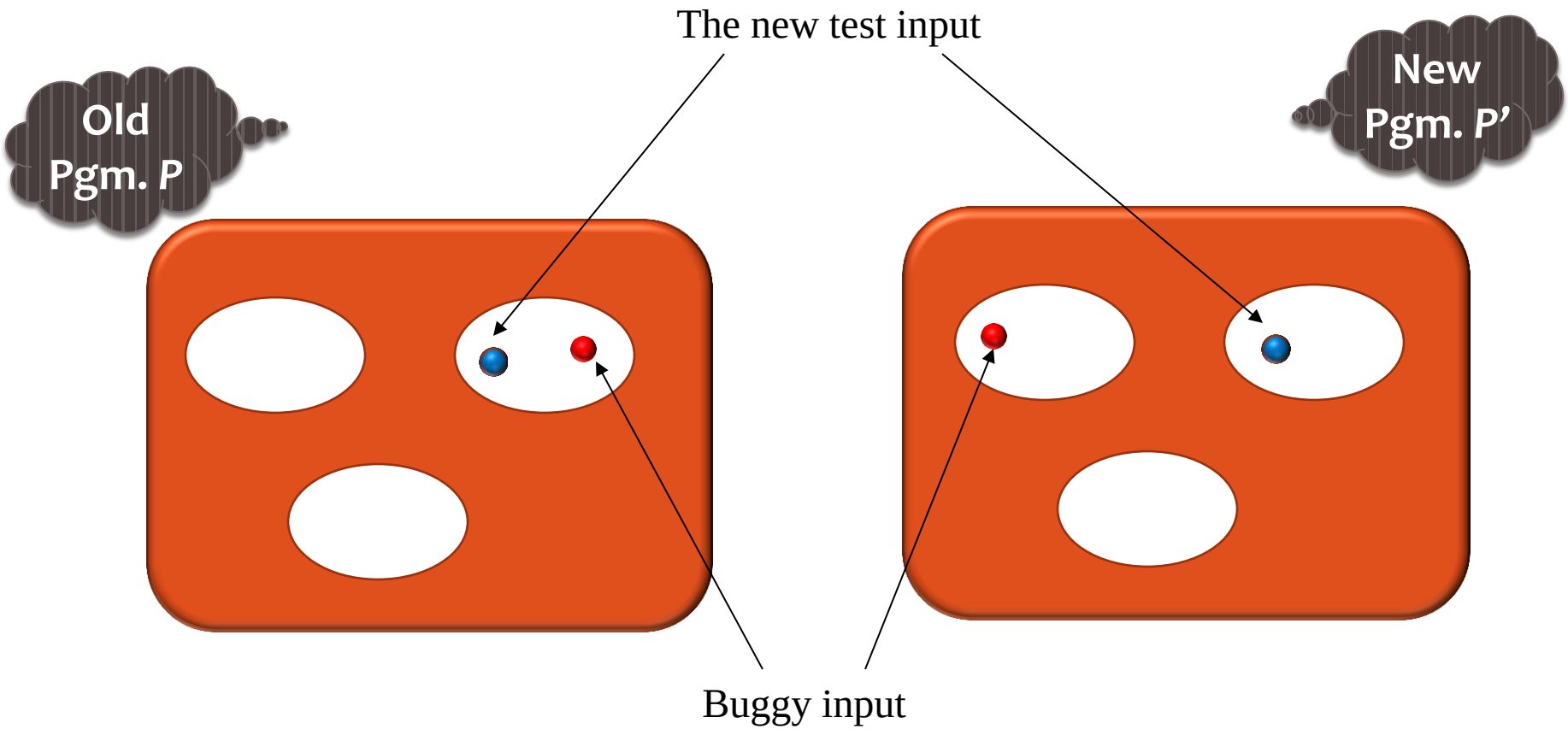


Adapting Trace Comparison



Directly Compare σ and π

How to obtain the new test?



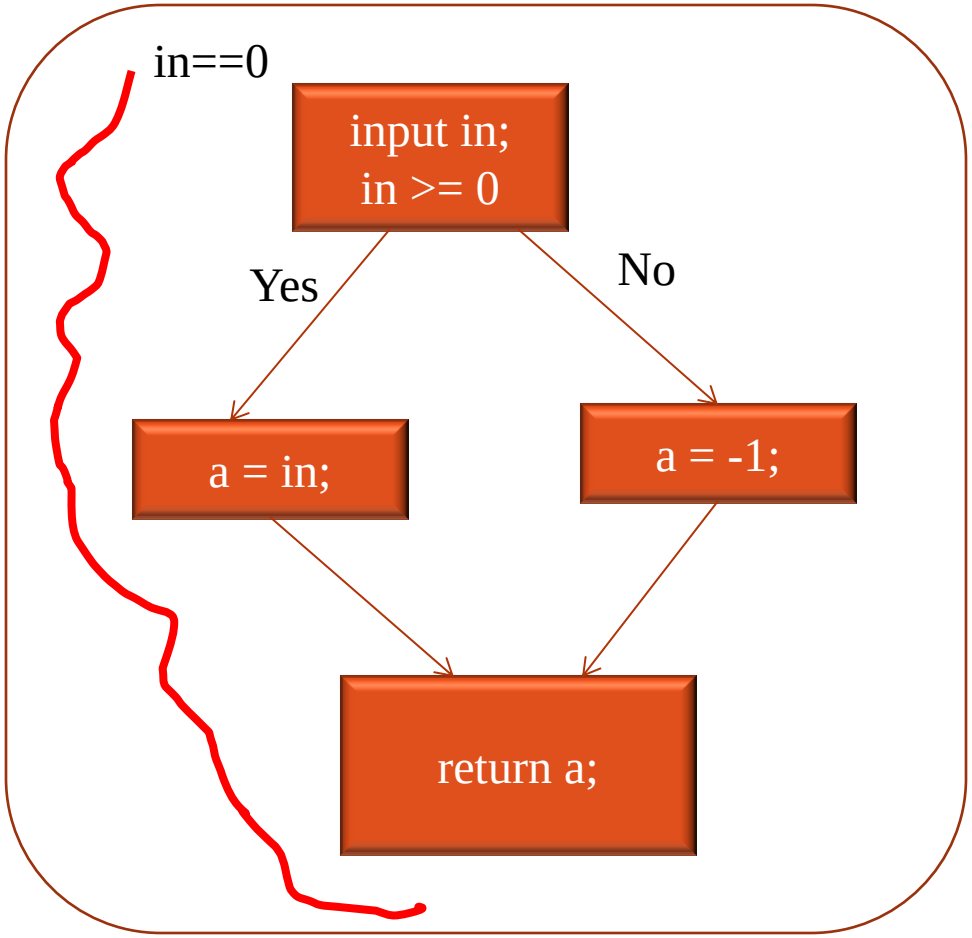
Path condition

Useful to find:

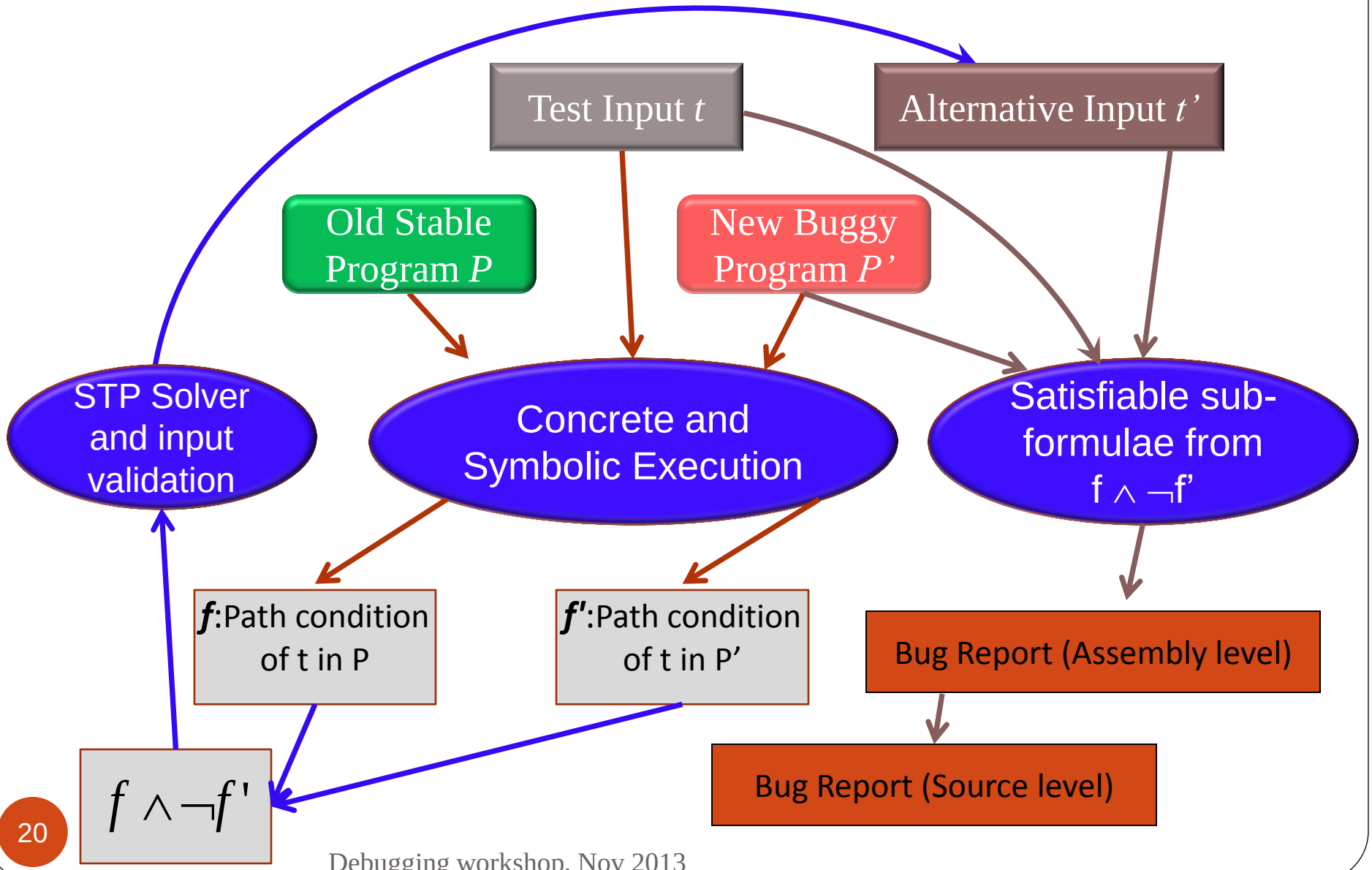
"the set of all inputs which trace a given path"

-> *Path condition*

in ≥ 0



DARWIN


Test Input *t*
Alternative Input *t'*
Old Stable Program *P*
New Buggy Program *P'*
STP Solver and input validation
Concrete and Symbolic Execution
Satisfiable sub-formulae from f ∧ ¬f'
f:Path condition of t in P
f':Path condition of t in P'
Bug Report (Assembly level)
Bug Report (Source level)
$f \wedge \neg f'$

Results

Buggy Program	Stable program	Time taken	Bug report size
LibPNG v1.0.7 (31164 loc)	LibPNG v1.2.21 (36776 loc)	13 m 34 s	9
TCPflow (patched)	TCPflow (unpatched)	31m	6
Miniweb (2838 loc)	Apache (358379 loc)	14s	5
Savant (8730 loc)	Apache httpd (358379 loc)	9m	46

If we require the alternative input to behave the same in buggy program and reference program (passing test) -
the **bug report size is 1 in all three cases.**

LibPNG v1.0.7 – v1.2.21

Bug we are debugging

Should be

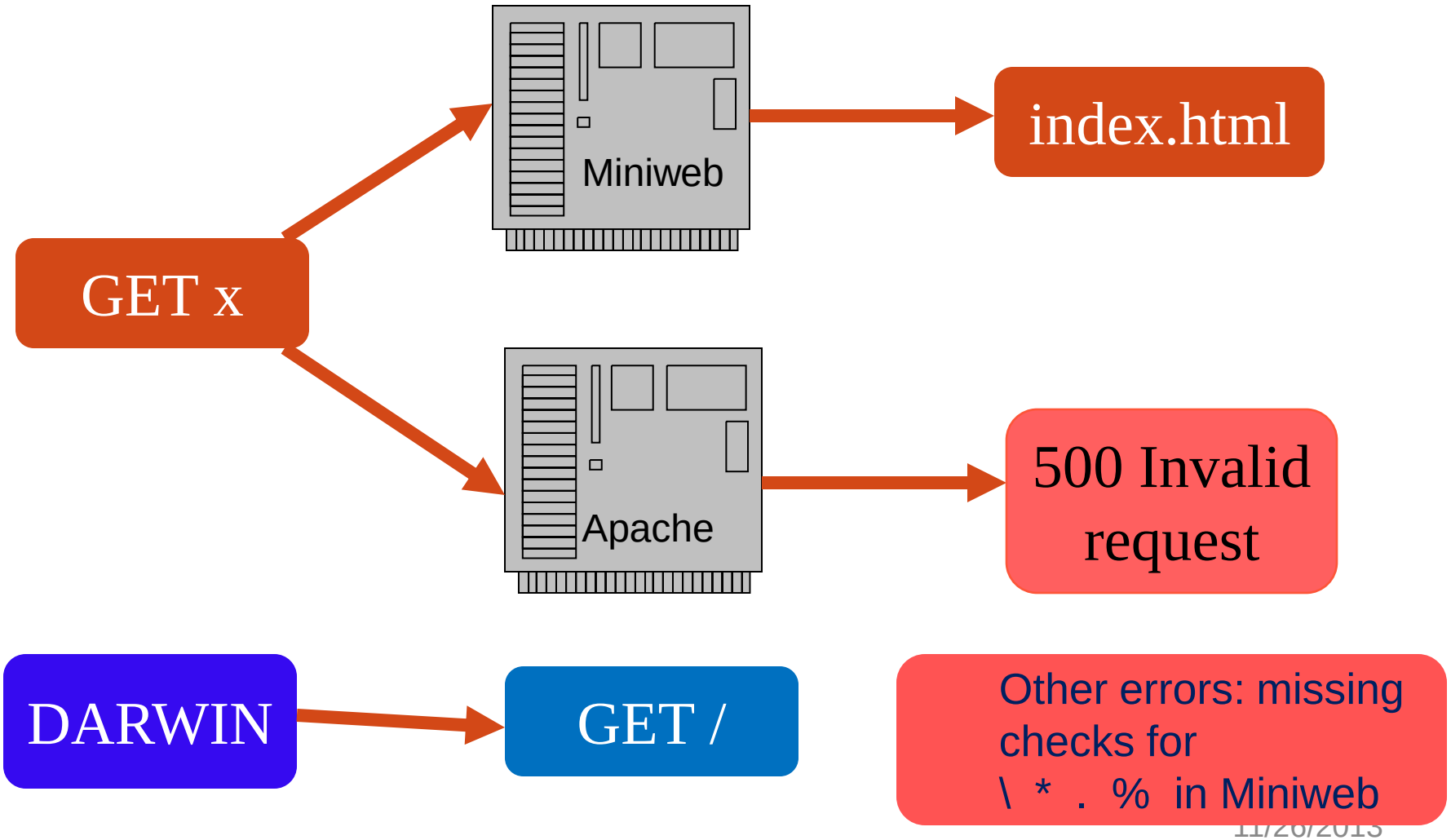
if **(length > …)**

```
if (!(png_ptr->mode
{
        png_warni g(png_ptr,  ...ssi          before tRNS");
}
else if (length > (png_uint_32)png_ptr->num_palette)
{
        png_warning(png_ptr, "Incorrect tRNS chunk length");
        png_crc_finish(png_ptr, length);
        return;
}
```

Buggy Input

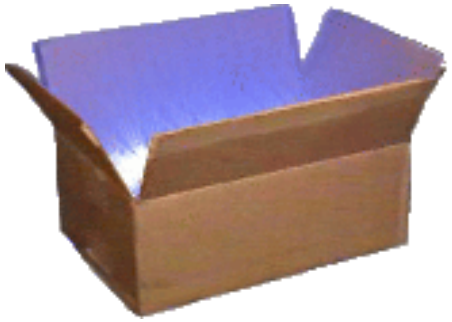
```
89 50 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44 52  |.PNG........IHDR|
00 00 00 5b 00 00 00 45  08 03 00 00 01 65 33 5a  |...[...E.....e3Z|
d6 00 00 02 00 74 52 4e  53 00 00 b1 8f 0b fc 61  |.....tRNS......a|
05 00 00 00 04 73 42 49  54 05 05 05 05 4d a5 2d  |.....sBIT....M.-|
```

Miniweb-Apache Httpd



Applications

Validated Embedded Linux
Busybox

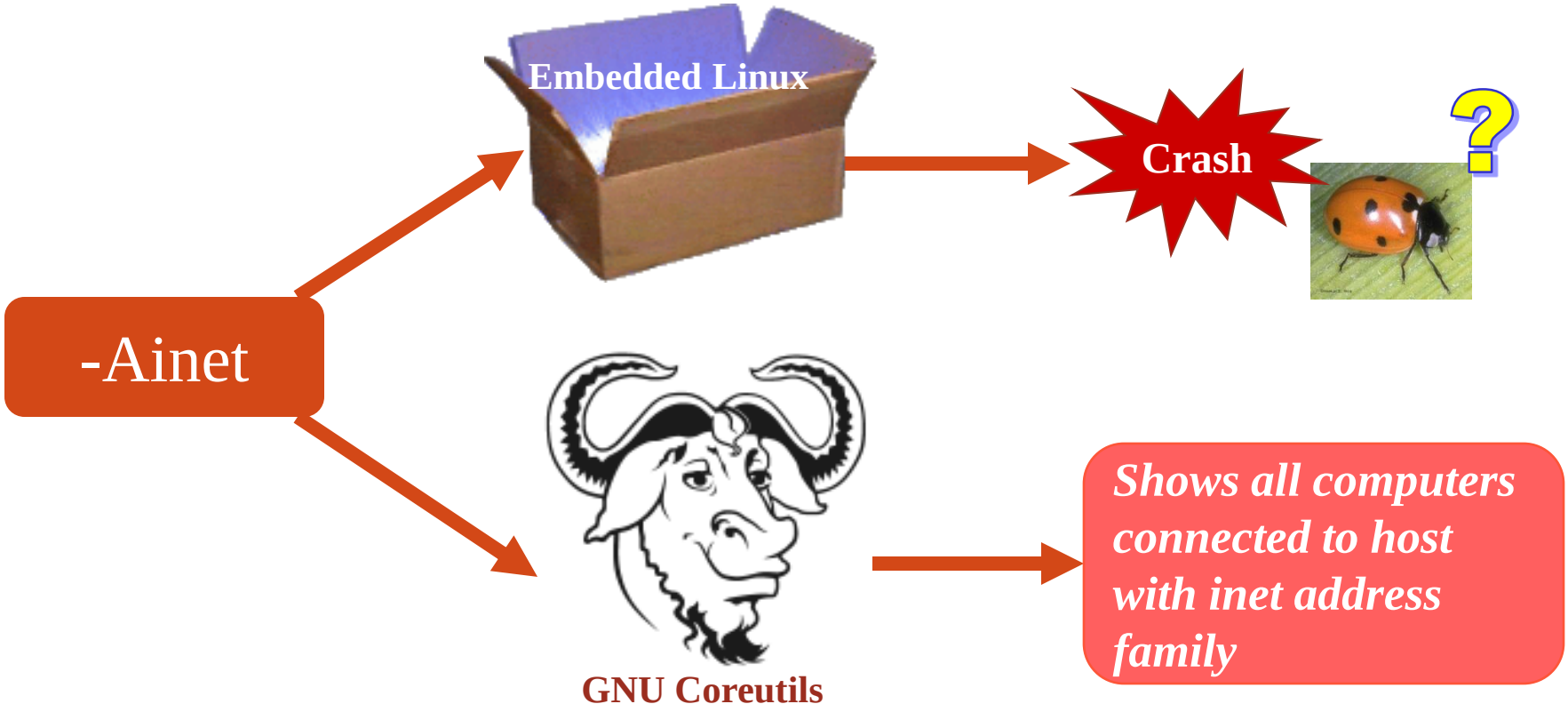


AGAINST

Linux (GNU Core-utils, net –tools)

Busybox distribution is 121 KLOC.
Various errors to be root-caused in `tr`, `arp`, `top`, `printf`.

ARP bug in busybox



Crash identified as NULL pointer access at crash site
hw_type unexpectedly set as NULL at crash site

Experiments on Embedded Linux

Utility	Trace Size	Slice Size	WP terms	WP terms (after elim.)	LOC in BugReport	Time taken
arp	5039 : 4764	56524 : 51448	722 : 434	27 : 34	1 : 3	1m30 s
top	1637 : 3921	34523 : 332281	566 : 2501	8 : 6	2 : 0	1m28 s
printf	3702 : 3633	27781 : 40403	241 : 414	21 : 35	1 : 3	1m20 s
tr	5474 : 138538	85047 : 29375	445 : 280	9 : 9	1 : 0	2m28 s

- Each : separated tuple in Columns 2-6 refers to data from embedded Linux and GNU Coreutils in that order
- Trace Size refers to no. of assembly / intermediate level instructions
- Tautology elimination reduces a significant WP analysis overhead
- Bug report size is quite small in each of the cases

Overall Perspective on debugging

- Breakpoints - Manual
- "Automated" Debugging
  - Trace comparison, …
  - Input mutation, …
  - …
  - Try successful artifacts of the buggy program.

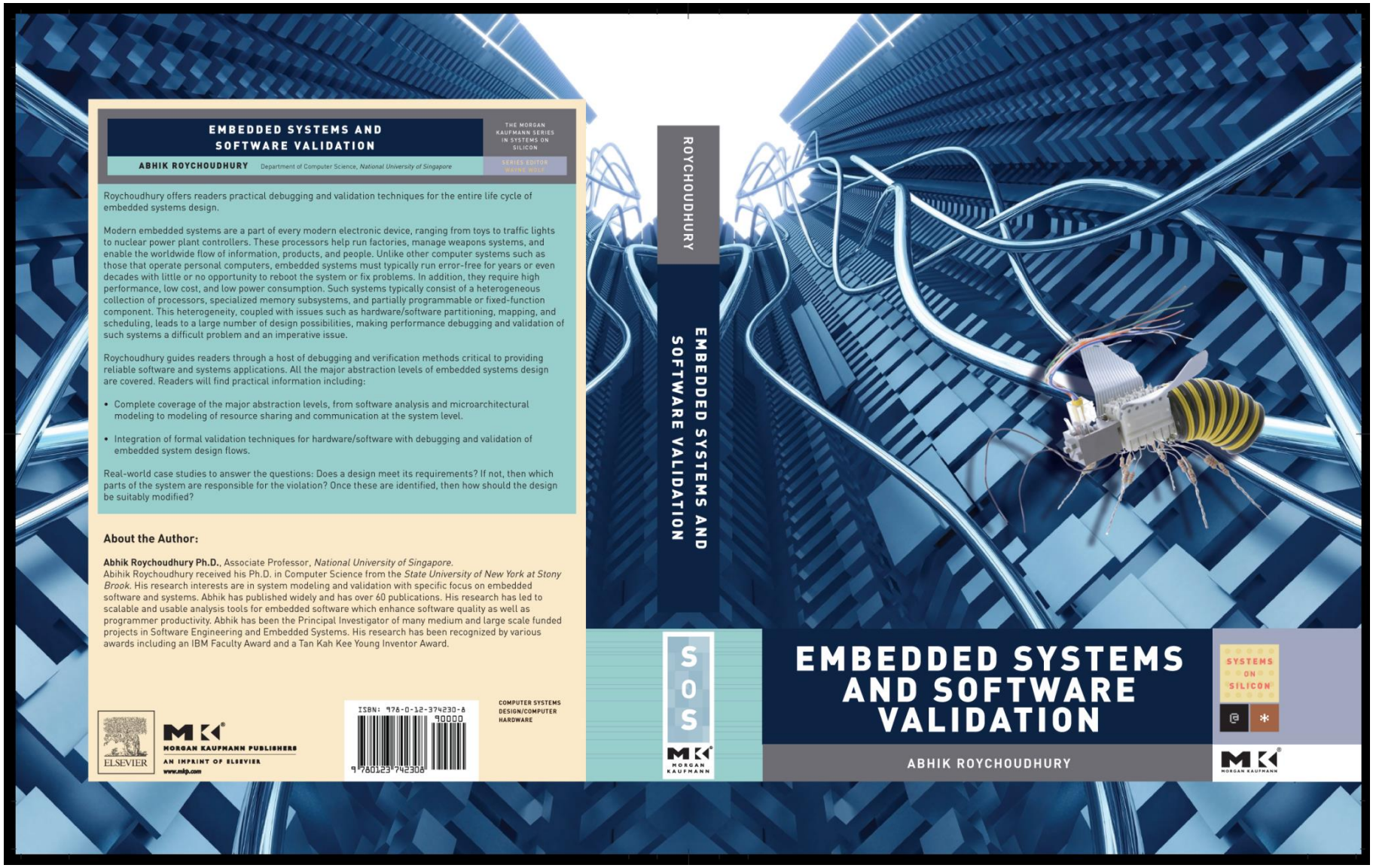
  - Symbolic Methods
    - Replace repeated experimentation with constraint solving.
    - Discover and (partially) infer intended semantics by symbolic analysis of failing trace.

Acknowledgements

- Funding
  - MoE, DRTech Singapore
- Co-authors
  - NUS: Zhenkai Liang, Dawei Qi, Ansuman Banerjee,…
  - MSRI: Kapil Vaswani
  - IBM: Satish Chandra.

- References

  - DARWIN: an approach for debugging evolving programs
    Dawei Qi, Abhik Roychoudhury, Zhenkai Liang, Kapil Vaswani, ESEC/FSE '09.
  - Golden implementation driven software debugging
    Ansuman Banerjee, Abhik Roychoudhury, Johannes A. Harlie, Zhenkai Liang, FSE '10.


EMBEDDED SYSTEMS AND SOFTWARE VALIDATION
THE MORGAN KAUFMANN SERIES IN SYSTEMS ON SILICON
ABHIK ROYCHOUDHURY
Department of Computer Science, National University of Singapore
SERIES EDITOR WAYNE WOLF
Roychoudhury offers readers practical debugging and validation techniques for the entire life cycle of embedded systems design.
Modern embedded systems are a part of every modern electronic device, ranging from toys to traffic lights to nuclear power plant controllers. These processors help run factories, manage weapons systems, and enable the worldwide flow of information, products, and people. Unlike other computer systems such as those that operate personal computers, embedded systems must typically run error-free for years or even decades with little or no opportunity to reboot the system or fix problems. In addition, they require high performance, low cost, and low power consumption. Such systems typically consist of a heterogeneous collection of processors, specialized memory subsystems, and partially programmable or fixed-function component. This heterogeneity, coupled with issues such as hardware/software partitioning, mapping, and scheduling, leads to a large number of design possibilities, making performance debugging and validation of such systems a difficult problem and an imperative issue.
Roychoudhury guides readers through a host of debugging and verification methods critical to providing reliable software and systems applications. All the major abstraction levels of embedded systems design are covered. Readers will find practical information including:
• Complete coverage of the major abstraction levels, from software analysis and microarchitectural modeling to modeling of resource sharing and communication at the system level.
• Integration of formal validation techniques for hardware/software with debugging and validation of embedded system design flows.
Real-world case studies to answer the questions: Does a design meet its requirements? If not, then which parts of the system are responsible for the violation? Once these are identified, then how should the design be suitably modified?
About the Author:
Abhik Roychoudhury Ph.D., Associate Professor, National University of Singapore.
Abhik Roychoudhury received his Ph.D. in Computer Science from the State University of New York at Stony Brook. His research interests are in system modeling and validation with specific focus on embedded software and systems. Abhik has published widely and has over 60 publications. His research has led to scalable and usable analysis tools for embedded software which enhance software quality as well as programmer productivity. Abhik has been the Principal Investigator of many medium and large scale funded projects in Software Engineering and Embedded Systems. His research has been recognized by various awards including an IBM Faculty Award and a Tan Kah Kee Young Inventor Award.
ELSEVIER
MORGAN KAUFMANN PUBLISHERS
AN IMPRINT OF ELSEVIER
www.mkp.com
ISBN: 978-0-12-374230-8
90000
9 780123 742308
COMPUTER SYSTEMS DESIGN/COMPUTER HARDWARE
ROYCHOUDHURY
EMBEDDED SYSTEMS AND SOFTWARE VALIDATION
SOS
MORGAN KAUFMANN
EMBEDDED SYSTEMS AND SOFTWARE VALIDATION
ABHIK ROYCHOUDHURY
SYSTEMS ON SILICON
MORGAN KAUFMANN