Automatically Detect Flaws in Cloud Platforms and System Software

Kang Li
Department of Computer Science
University of Georgia

About Me

- @ UGA
  - Faculty of Computer Science
  - Leading the Security Research Cluster

- Beyond UGA
  - Frequent BlackHat and ShmooCon Speaker
  - Founder of the *disekt* CTF Team

Motivation for Automation (analysis, offense/defense)

1. Demand from Software Practice

2. The (in)Balance of "Hacking" Power

Demand from SW practice

- Sample High Profile Victims in the News

Golden Age of Bugs!

- High Profile Vulnerabilities
  - Heartbleed (4/2014), ShellShock (9/2014), POODLE (12/2014), GHOST (4/2015)

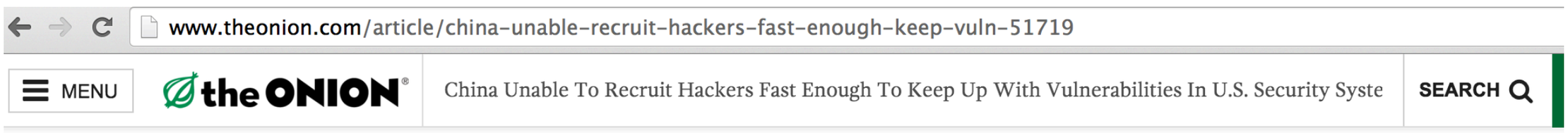


Demand from SW practice

- We (& the whole SW industry) are generating so many bugs, that the Onion made the following "news" when *China announce to abandon One-Child Policy in October 2015:*

Demand from SW practice

- We (& the whole SW industry) are generating so many bugs, that the Onion made the following "news" when *China announce to abandon One-Child Policy in October 2015:*



China Unable To Recruit Hackers Fast Enough To Keep Up With Vulnerabilities In U.S. Security Systems

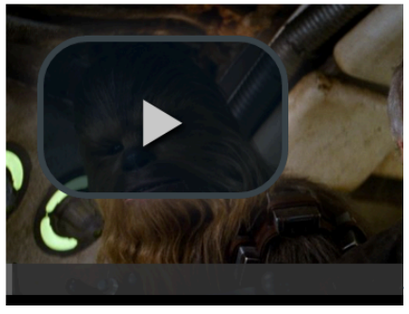
NEWS IN BRIEF

October 26, 2015

VOL 51 ISSUE 43

News · Technology · World · China

ONION VIDEO



Motivation for Automation (analysis, offense/defense)

1. Demand from Software Practice

2. The (in)Balance of "Hacking" Power

The (in)Balance of “Hacking” Power

Year	1st	From	2nd	From	3rd	From
2002	Digirev	**USA**	Immunix	**USA**	BrownTeam	**USA**
2003	Anomaly	**USA**	Digirev	**USA**	Immunix	**USA**
2004	Sk3wl0fr00	**USA**	IronGrep	**USA**	MOCYLIB	**USA**
2005	Shellphish	**USA**	PlanB	**USA**	Sk3wl0fr00t	**USA**
2006	1@stplace	**USA**	Shellphish	**USA**	Sk3wl0fr00t	**USA**
2007	1@stplace	**USA**	Sk3wl0fr00t	**USA**	SongofFreedon	**USA**
2008	Sk3wl0fr00	**USA**	Routards	**France**	1@stplace	**USA**
2009	VedaGodz	**USA**	Routards	**France**	PLUS@postech	**South Korea**
2010	ACME Phar	**USA**	Routards	**France**	GoN	**South Korea**
2011	Nopsled	**Denmark**	Routards	**France**	Hates Irony	**USA**
2012	Samurai	**USA**	PPP	**USA**	NopSled	**Denmark**
2013	PPP	**USA**	ManInBlackH	**USA**	RAON_ASRT	**South Korea**
2014	PPP	**USA**	HITCON	**TAIWAN**	Dragon Sector	**Poland**
2015	DEFKOR	**South Korea**	PPP	**USA**	0daysober	**France/SZ**

TOP 3 CTF Teams in DEFCON CTF Finals

CTF will be Played by Machines

DARPA Cyber Grand Challenge

http://cybergrandchallenge.com/

A tournament for fully automated network defense

DARPA Cyber Grand Challenge

www.cybergrandchallenge.com

TEAMS

Below are the 7 top-ranking teams from the First Scored Event that occurred on 12/02/14 in rank order. Please note that the teams in 3rd and 4th place are tied:

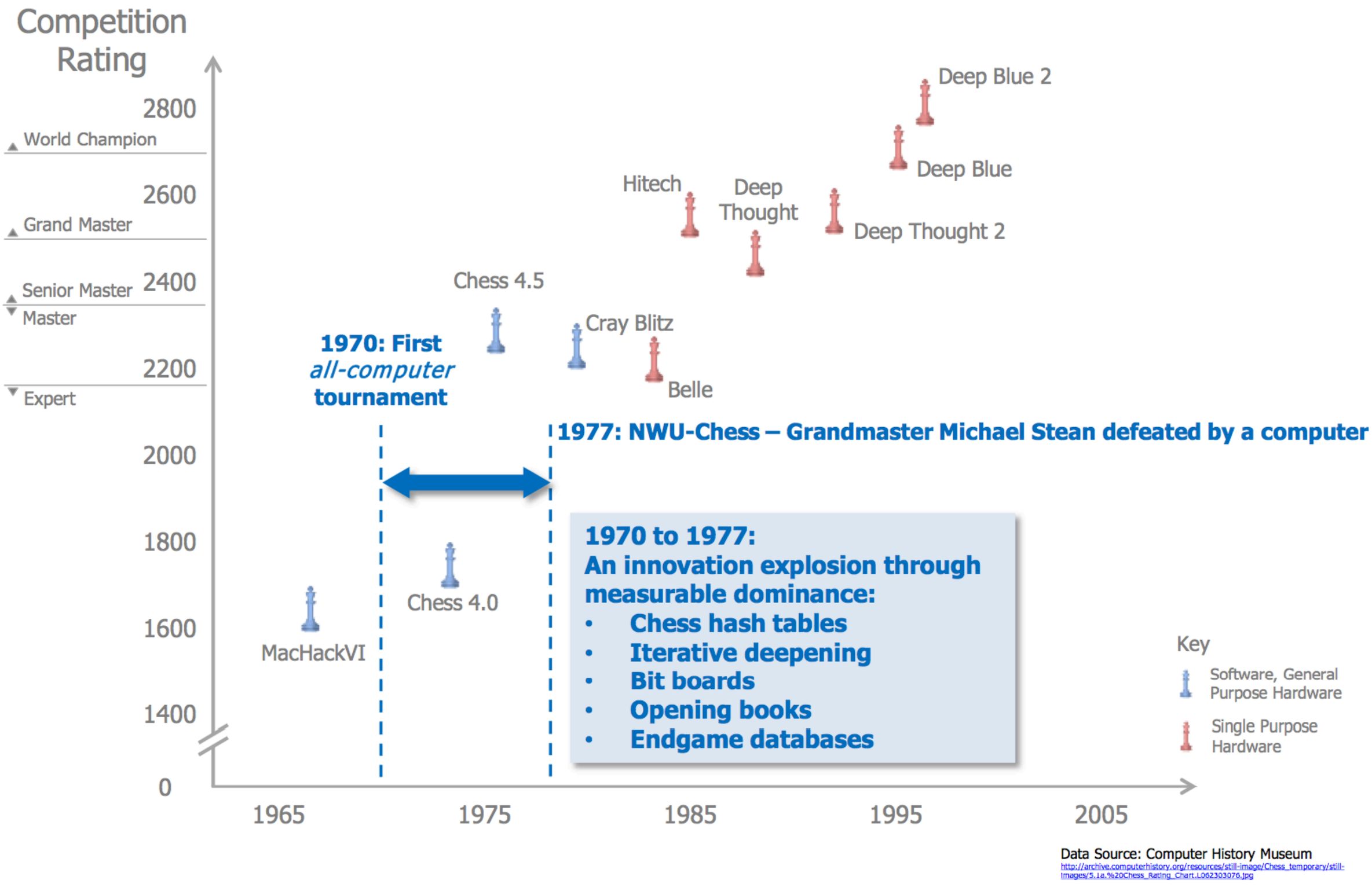
1. Deep Red [Open Track]
2. CSDS [Open Track]
3,4. Shellphish and disekt [Open Track]
5. ForAllSecure [Funded Track]
6. Codejitsu [Funded Track]
7. TechXicians [Funded Track]

ATHENS, GA
disekt - Athens, GA
Ranked 4 in the First Scored Event

CYBER GRAND_CHALLENGE

Contact | Legal | DARPA

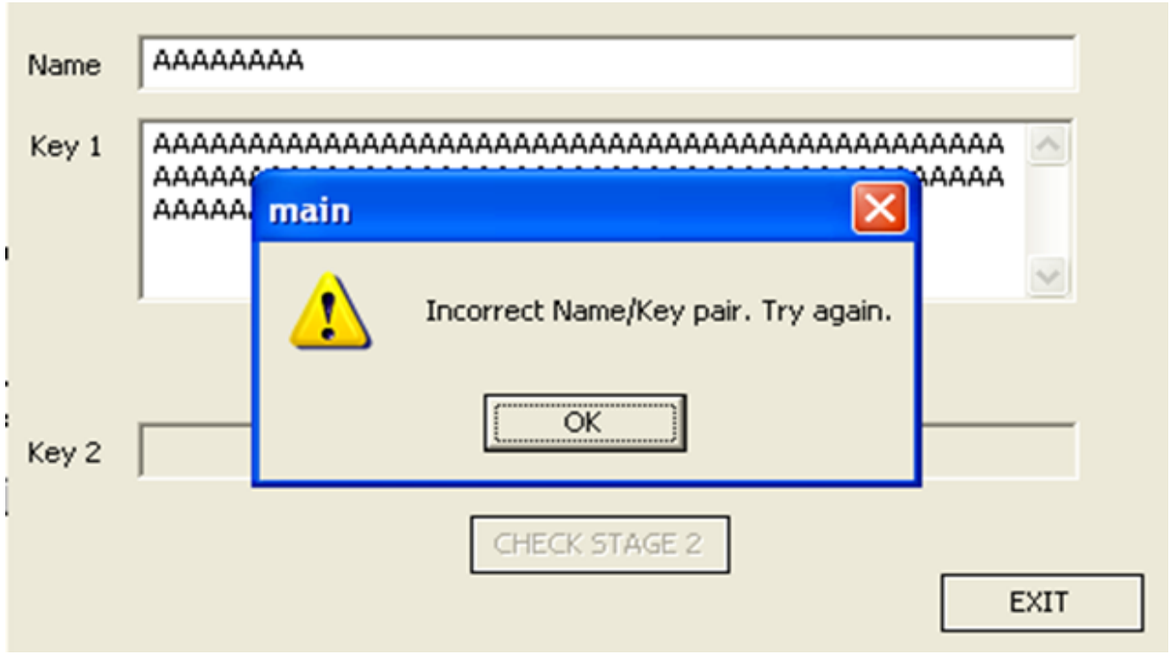
Other Success Example (Chess Master)



Source: Mike Walker's presentation at the ISSTA 2014 conference

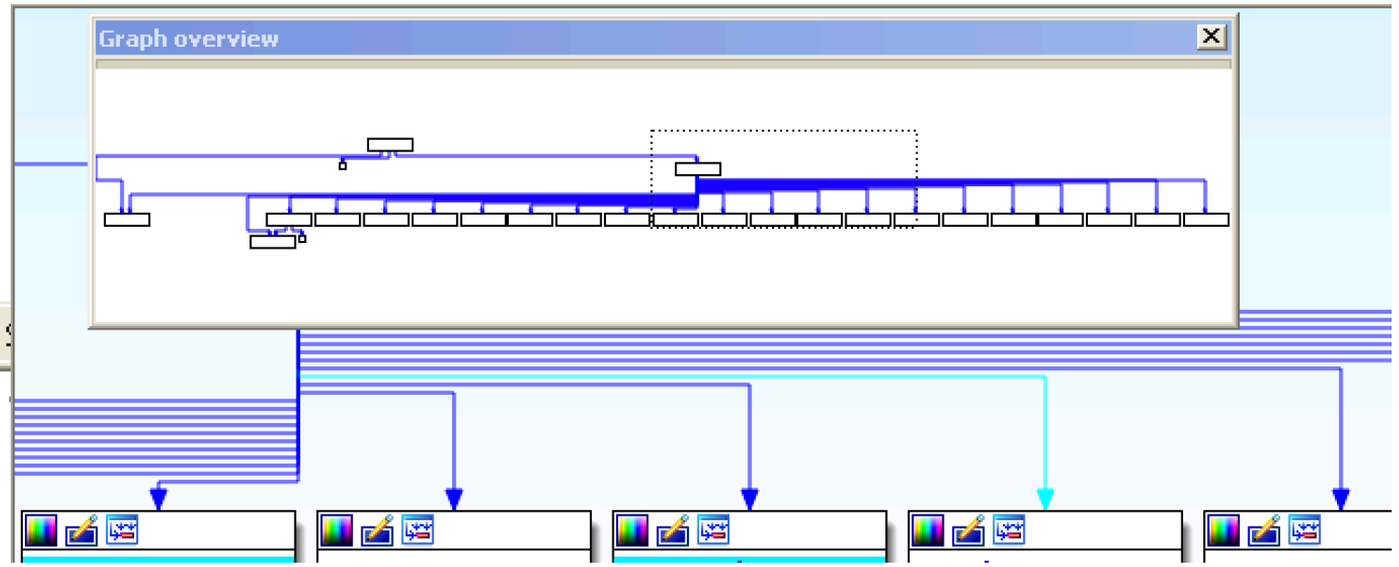
Can Machine Do It (auto analysis, offense/defense)?

Example: CrackMe Challenges



Can Machine Do It (auto analysis, offense/defense)?

Example: CrackMe Challenges



IDA View-A | Pseudocode-A | Hex View-1

```
1  INT_PTR __stdcall DialogFunc(HWND hDlg, UINT
2  {
3    LPVOID v4; // esi@2
4    unsigned int v5; // ecx@2
5
6    if
7    {
8      hI                                    )0xC8, (LPCSTR)0xA);
9      hI                                    );
10     dI                                    esInfo);
11     vI
12     dI                                    d_4050BB + 4);
13     vI
14     *
15     qI                                    ;
16     sI
17   }
18   else if ( a2 == 273 )
```

Warning

Decompilation failure:
40120E: call analysis failed

Please refer to the manual to find appropriate actions

OK

Don't display this message again (for this session only)

000004E4 DialogFunc:16

Symbolic Execution

Code Snippet

```
…
update_RA (int value)
{
      R_A = value & 0x0000FFFF;

      return R_A
}
```

- Execution with Concrete values
  - Input

    value = 0xDEADBEEF
  - Output

    R_A = 0x0000BEEF;

- Execution with Symbolic values
  - Input

    value = α
  - Output

    R_A = α & 0x0000FFFF;

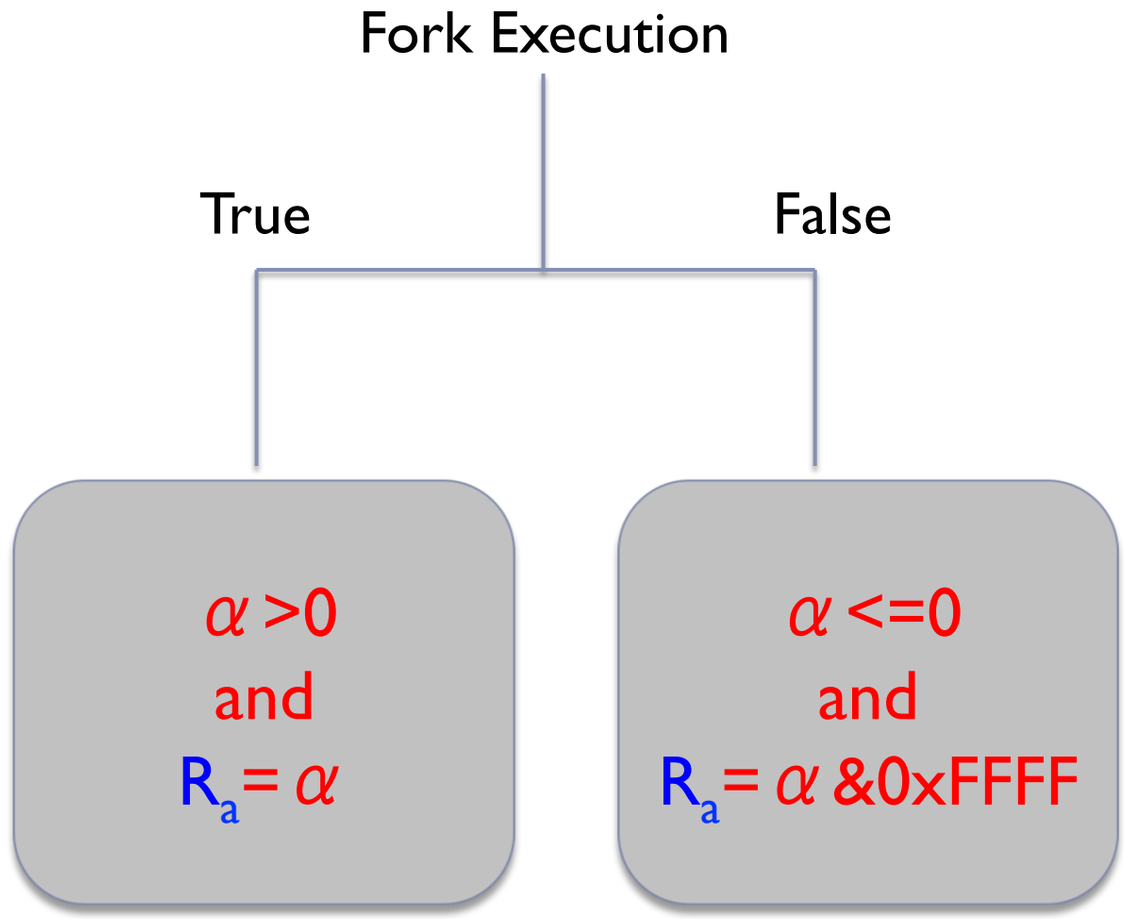
Symbolic Execution with Branches

- Possible Execution Paths

Code Snippet

```
…
update_RA (int value)
{
        if (value > 0)
                R_A = value;
        else
                R_A  = value & 0x0000FFFF;

        return R_A
}
```



Can We Do It (auto analysis, offense/defense)?

Solving CrakeMe with Symbolic Execution

Progress of Auto Program Analysis

- Detection of Well-defined Vulnerabilities
  - Static & Dynamic Checking for Properties
    - E.g. Memory Access Out of Bound
  - Rich Set of Prior Research Results/Tools
    - KLEE, BitBlaze, Mayhem, S2E, …

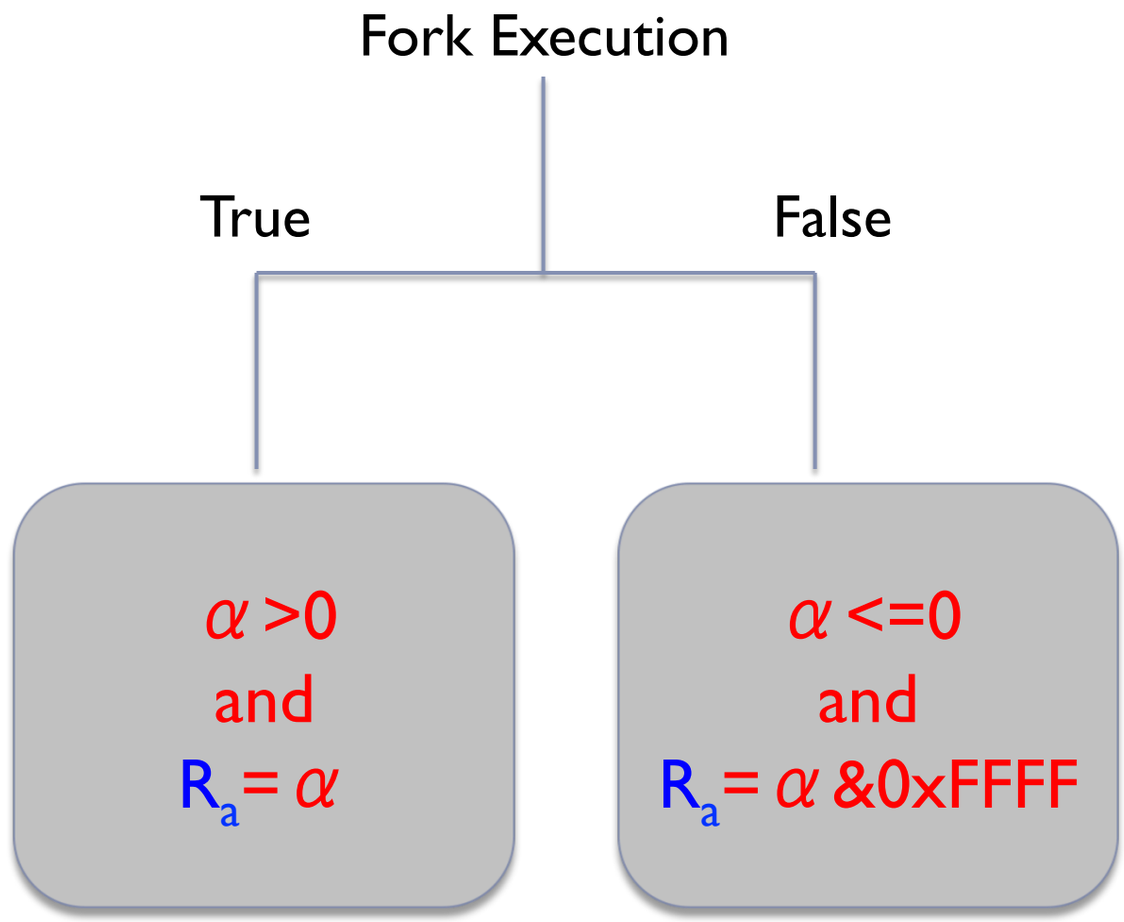
Property Checking

Code Snippet

```
…
update_RA (int value)
{
        if (value > 0)
                R_A = value;
        else
                R_A  = value & 0x0000FFFF;

        return R_A
}
```

- Possible Execution Paths



- Does the following condition hold for all possible input?

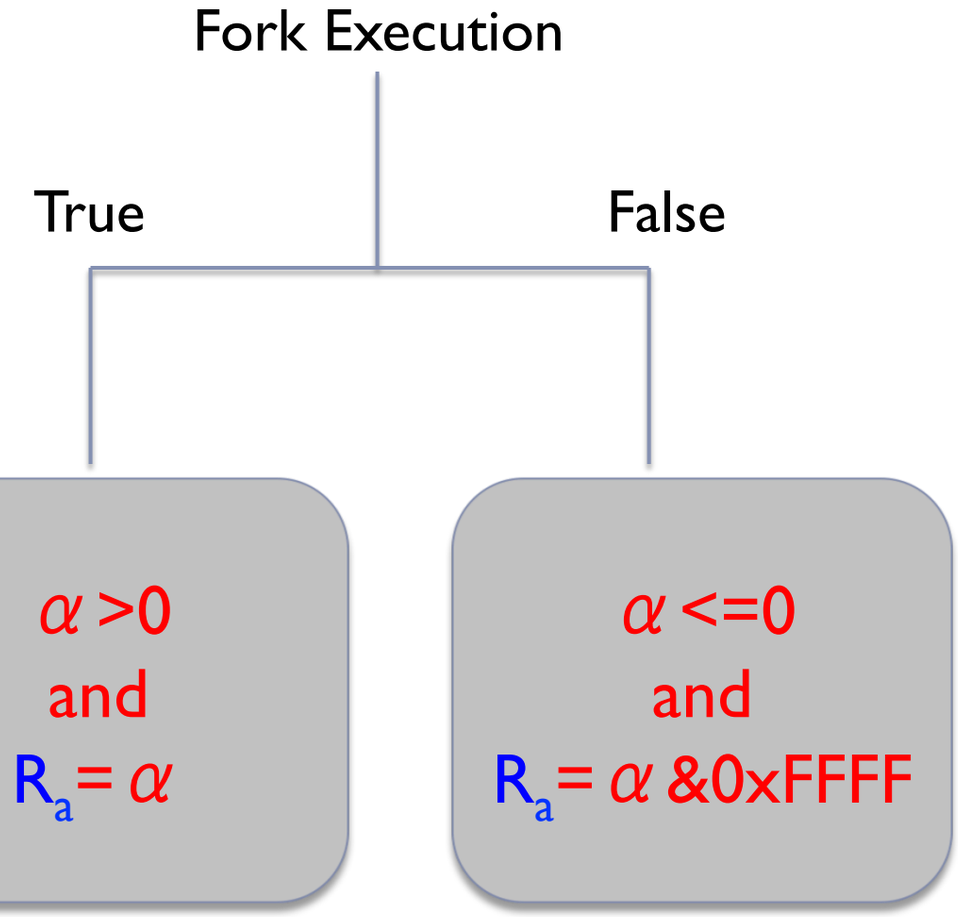
$R_A >= 0$

Property Checking

For each path,

solving the constrain

¬ (R_a>=0) && (Path Condition)



- Does the following condition hold for all possible input?

  R_A>=0

Property Checking

For the early example, the constraints to solve are:

$\neg (R_a >= 0) \;\&\&\; (\alpha > 0 \;\&\&\; R_a = \alpha)$

$\neg (R_a >= 0) \;\&\&\; (\alpha <= 0 \;\&\&\; R_a = \alpha \;\&0xFFFF)$

No solution means the following statement holds

$R_A >= 0$

Applying Symbolic Execution

- Detection of Well Defined Vulnerabilities
  - Manually define rules to check
    - E.g. memory access out of bound, double free on the same path
- Detection of Flaws in VMs and Embedded Firmware
  - Checking for specification violation
    - Cloud/VM Platform Implementations
    - Firmware (Bootloader) Implementations

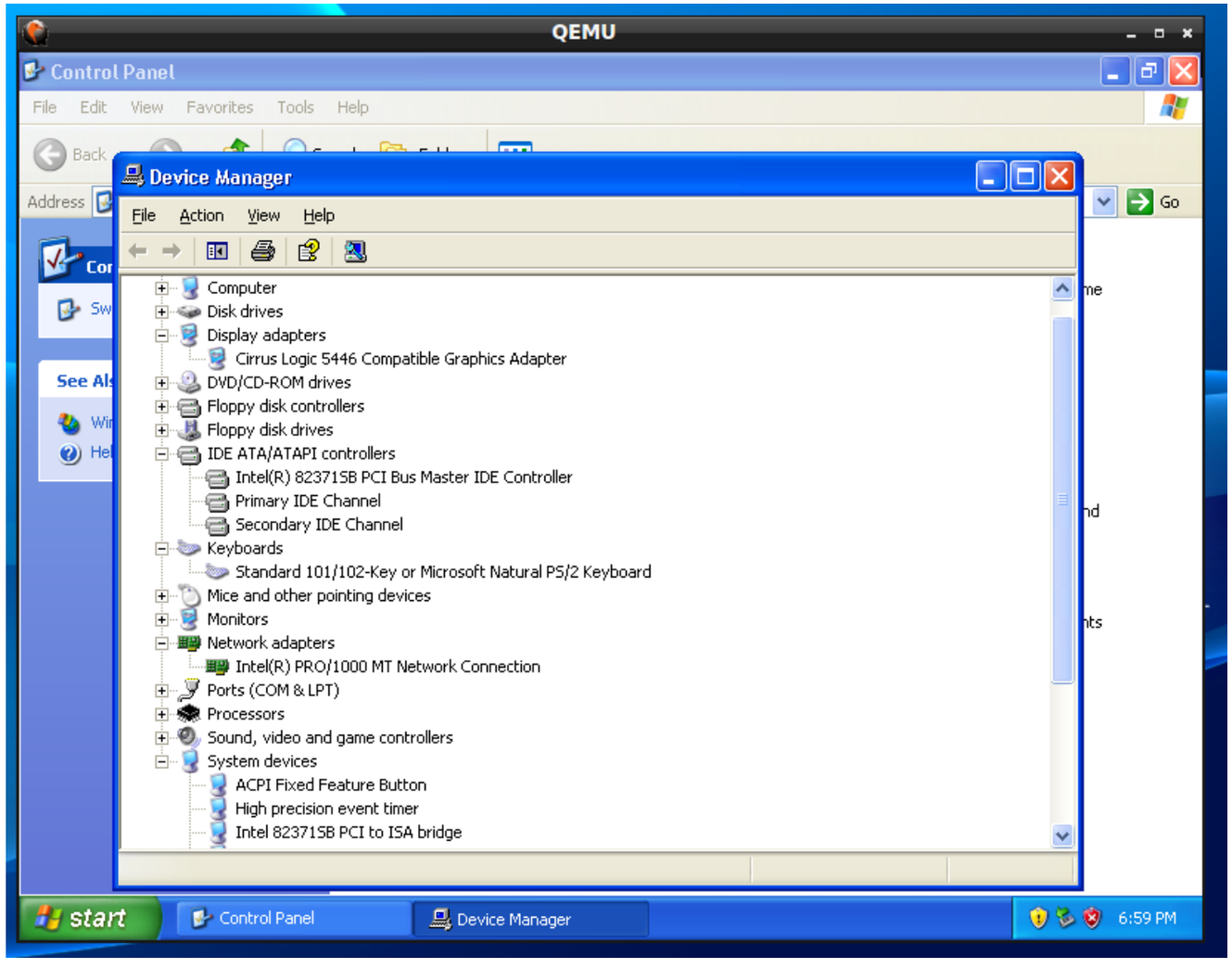
Applying Symbolic Execution

- Detection of Well Defined Vulnerabilities
  - Manually define rules to check
    - E.g. memory access out of bound, double free on the same path
- Detection of Flaws in VMs and Embedded Firmware
  - Checking for specification violation
    - Cloud/VM Platform Implementations
    - Firmware (Bootloader) Implementations

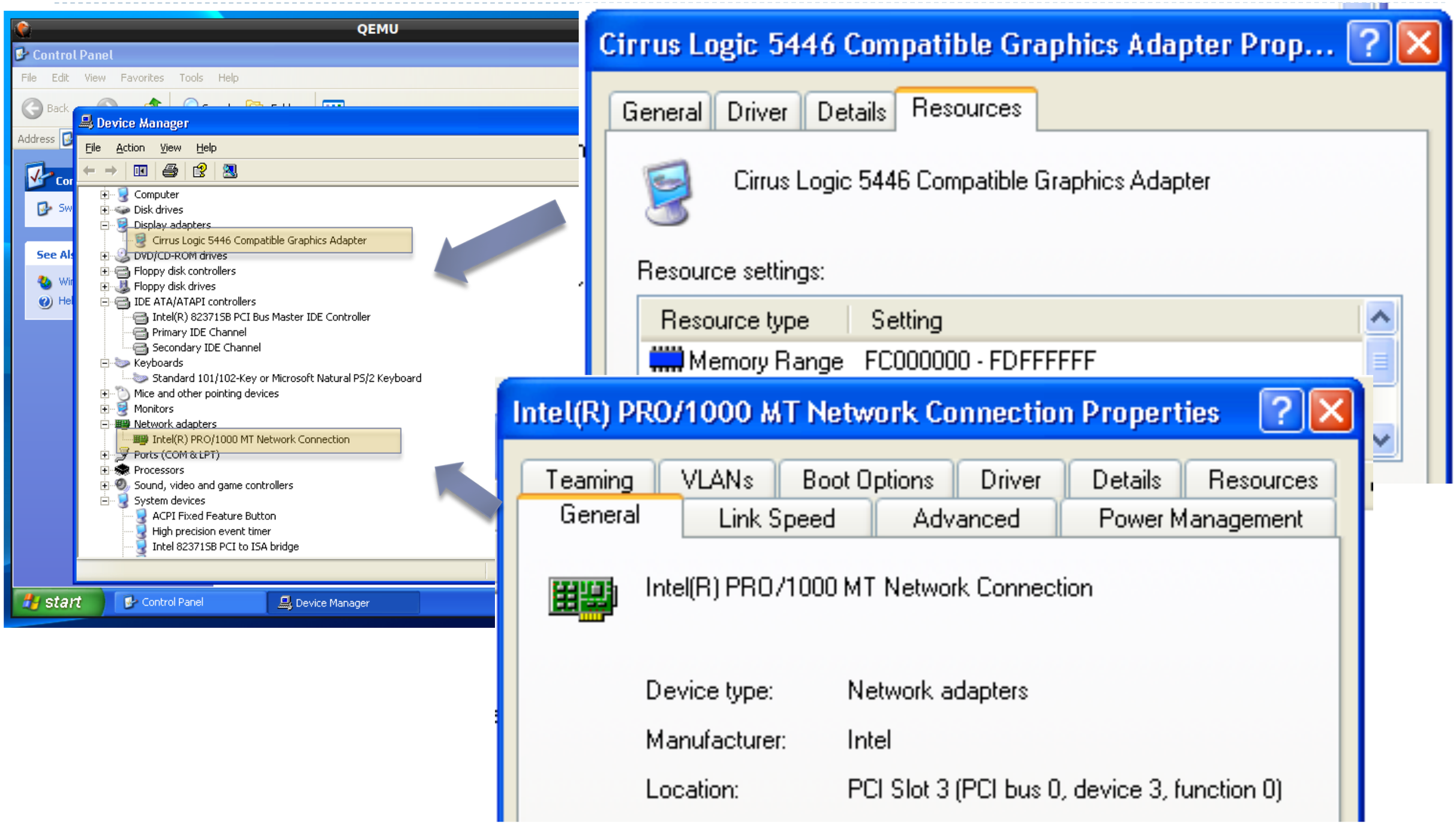
Applying Symbolic Execution

- Detection of Well Defined Vulnerabilities
  - Manually define rules to check
    - E.g. memory access out of bound, double free on the same path
- Detection of Flaws in VMs and Embedded Firmware
  - Checking for specification violation
    - Cloud/VM Platform Implementations
    - Firmware (Bootloader) Implementations
- Challenges to Automation
  - What property (predicates/invariants) to check?
  - How to handle incomplete programs?

Cloud and Virtual Machines



Examples of Virtual Hardware Devices



Recent VM Vulnerabilities

Oct, 2014

ars technica

Security bug in Xen may have exposed Amazon, other cloud services [Updated]

Flaw in hypervisor could let malicious VM read data from or crash other servers.

by **Sean Gallagher** - Oct 1, 2014 10:49am EDT

Share Tweet

March, 2015

New Xen vuln triggers Amazon, Rackspace reboot panic redux

Second hypervisor-related cloud meltdown in six months

Image & Text Source: http://www.theregister.co.uk/

Assumptions on Virtual Machine

- Software (drivers and OS) makes assumptions about hardware behavior.

- Virtual hardware does not behave exactly like Physical hardware.

- Such inconsistencies could lead to unexpected software failures, and some flaws could be fatal and exploitable by attackers.

Address the Challenge of "What to Check"

- The Idea:

  Check virtual HW device against its physical peer

  ➔ Behavior Comparison ("Model Checking")

- Actions:

1. Find the physical device (which the virtual device is based on)

2. Capture behavior of device under physical HW and virtual device, and compare them.

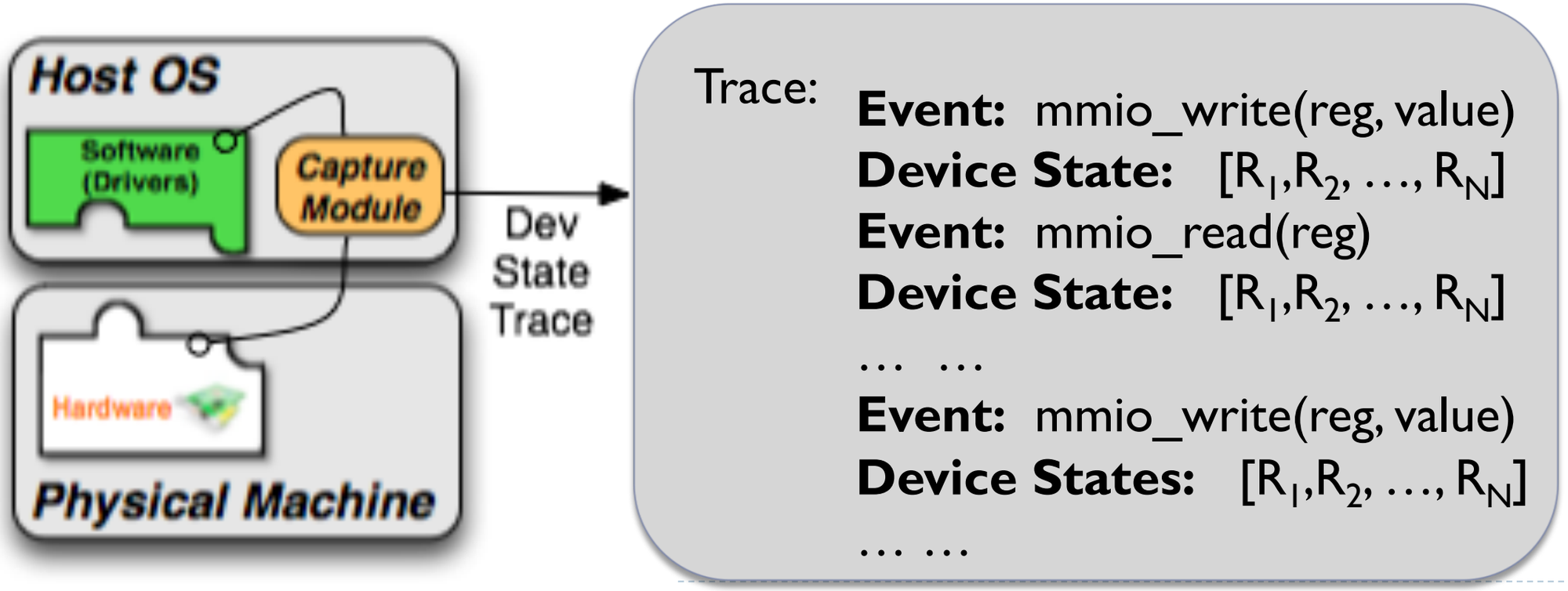
Address the Challenge of "What to Check"

- Detect virtual hardware behaviors that diverge from specification

- Focus on behaviors visible to Software
  - Do the hardware registers and memory contain the correct values during operation?

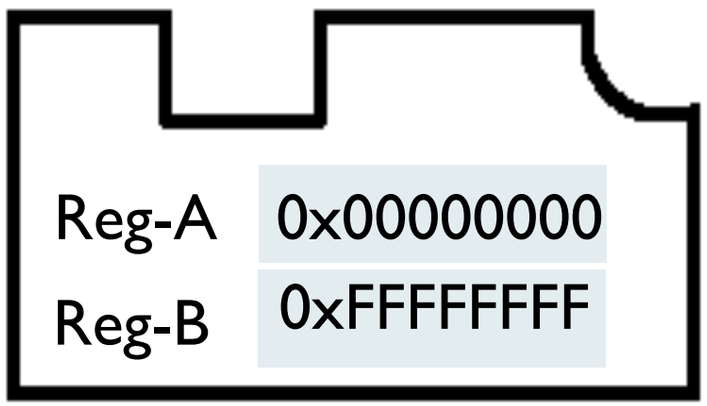
What can be observed

- The behavior of a HW device is defined by its registers and how registers respond to I/O events.
  - Full visibility at design time
  - But limited visibility on physical device (after manufacture)
- Observed by Capturing Traces (of events and dev states)



Example of Capturing HW Behavior

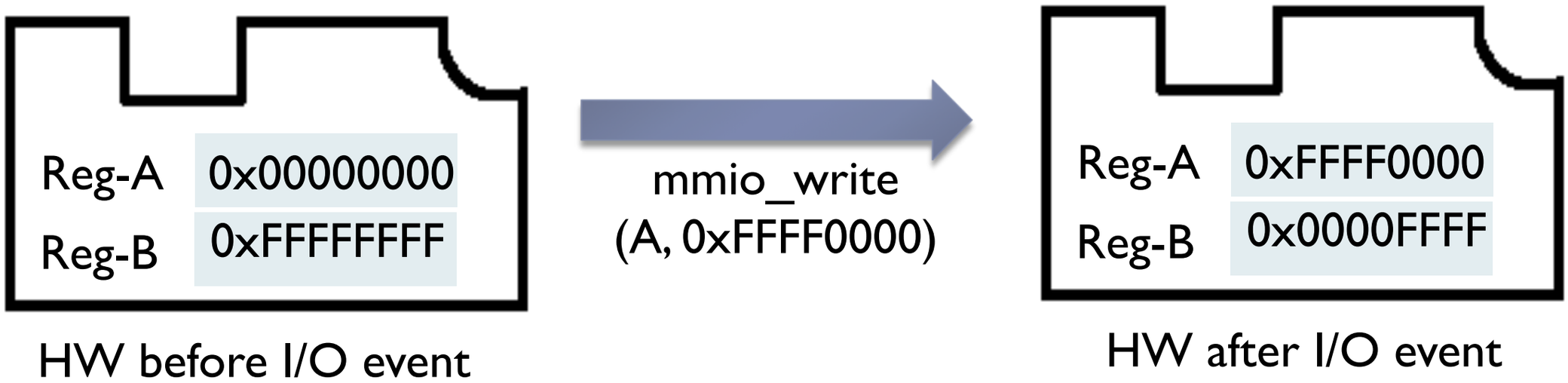
Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B \& \sim V_A$



HW before I/O event

Example of Capturing HW Behavior

Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B \& \sim V_A$

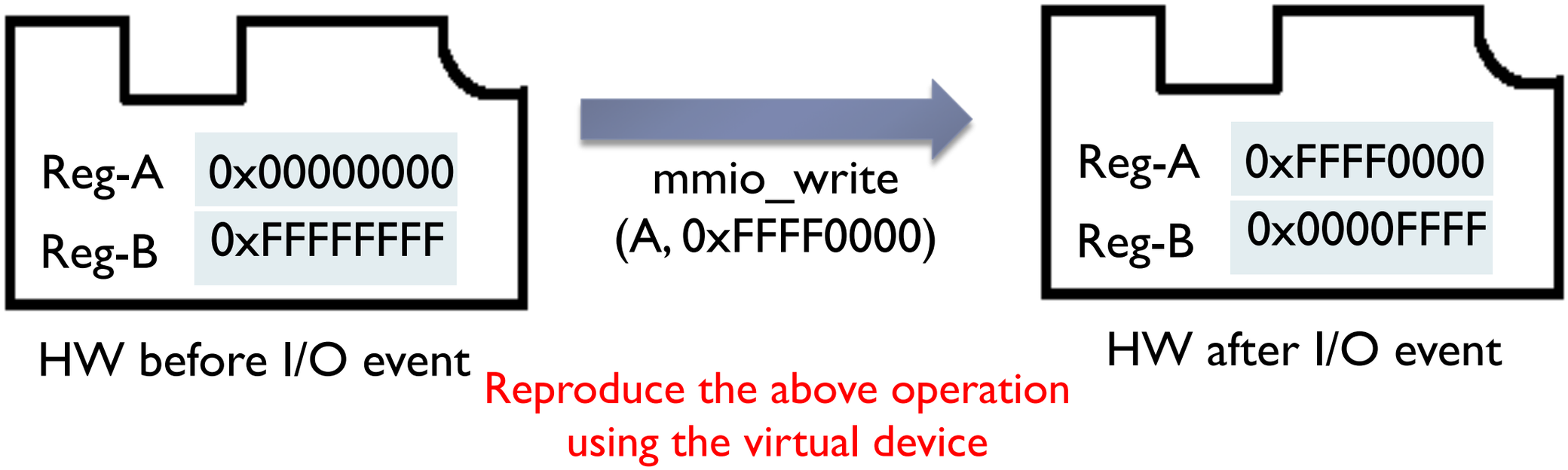


HW before I/O event

HW after I/O event

Example of Capturing HW Behavior

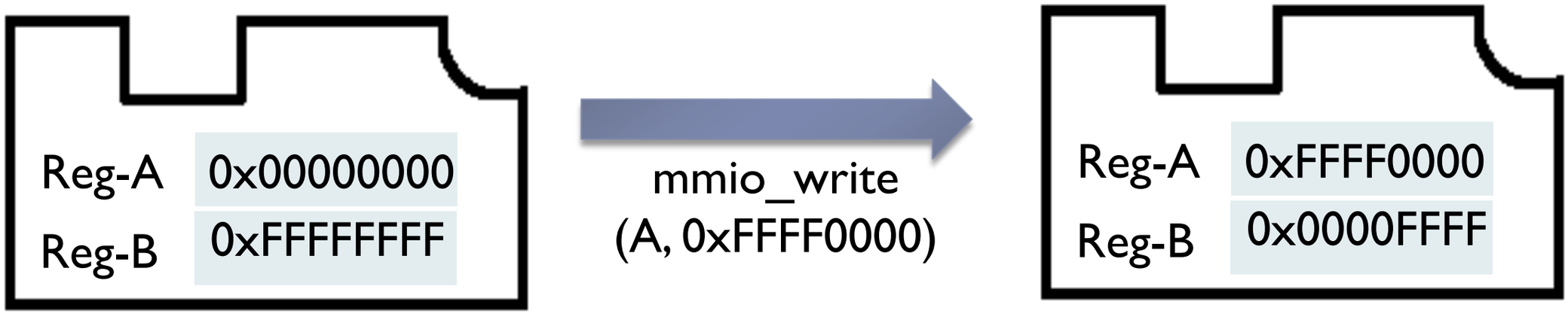
Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B \& \sim V_A$



Reproduce the above operation
using the virtual device

Example of Capturing HW Behavior

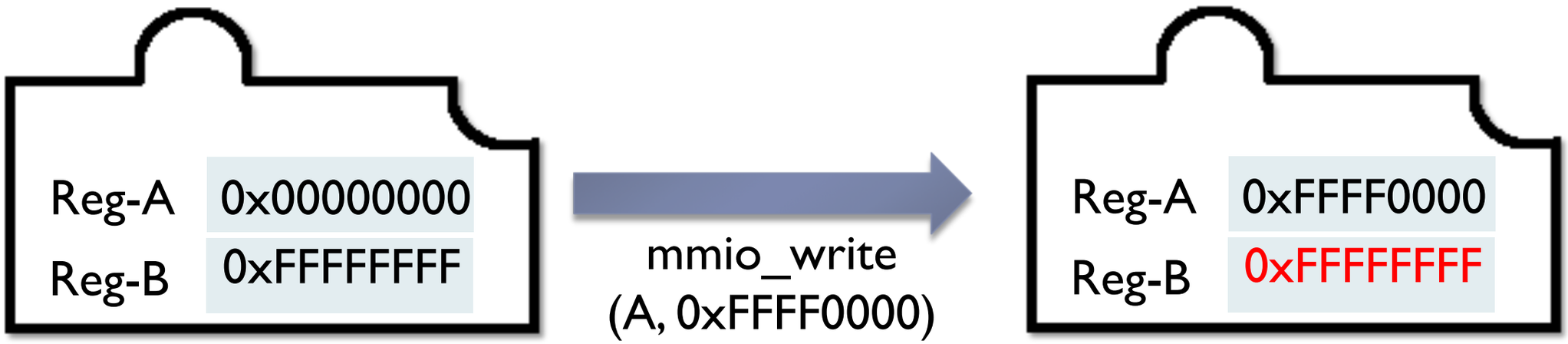
Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B \& \sim V_A$



HW before I/O event

HW after I/O event

Reproduce the above operation using the virtual device

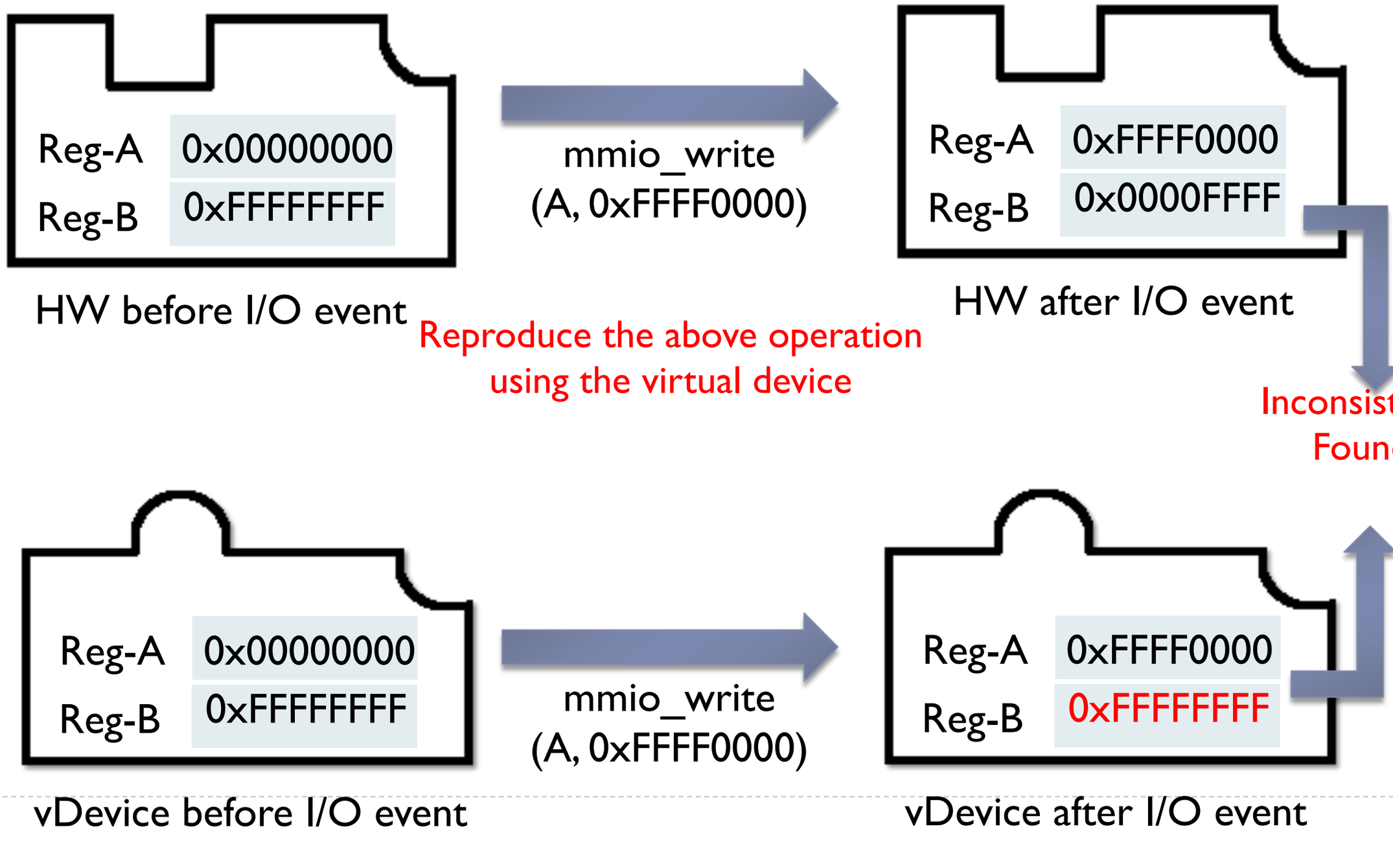


vDevice before I/O event

vDevice after I/O event

Example of Capturing HW Behavior

Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B \& \sim V_A$



HW Behavior Capturing (In Reality)

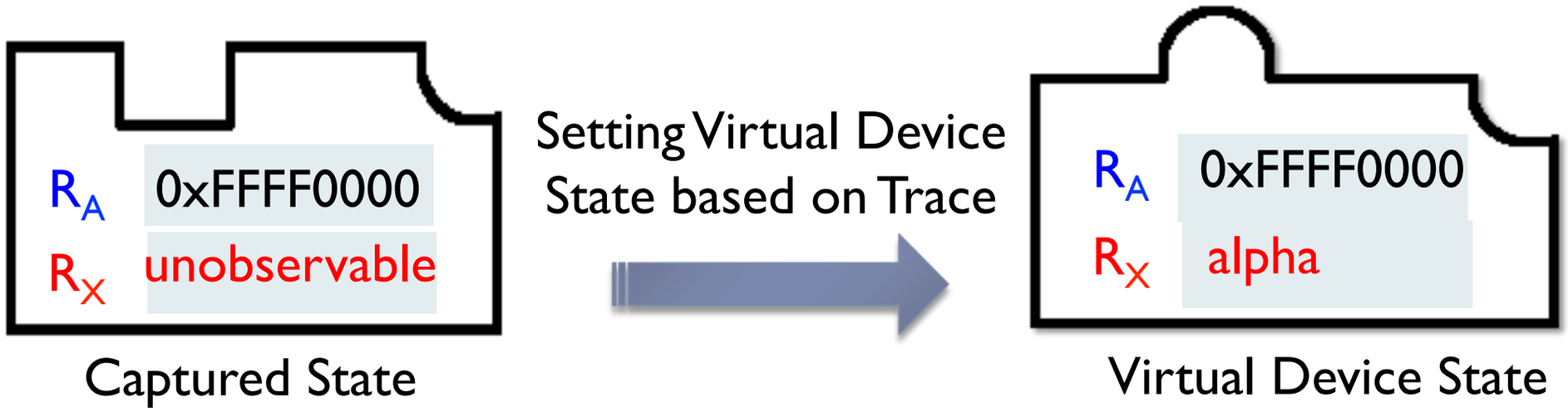
- Dump and replay only works in simple cases
  - Not all physical registers are observable (readable)
  - Some events are difficult or "expensive" to observe
  - Some registers are accessible, but have side effects

Symbolic Behavior Testing

- How to handle partially observable states?
- Our approach to deal with unobservable registers
  - Construct the virtual device state by setting
    - observable register values based on the trace
    - missing registers with symbolic values

Symbolic Register Values

- Example:
  - For a simple device with only 2 registers:
    - R_A (observable) and R_X (unobservable)
  - The device state in a trace looks like this: [R_A== 0xFFFF0000]



How to Run with Symbolic Values?

- Consider the following virtual device program:

Virtual Device Code Snippet

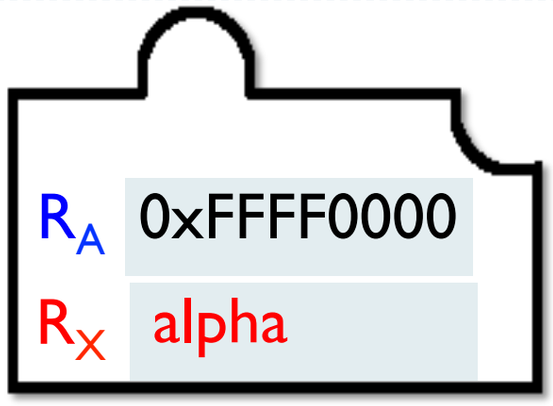
```
…
mmio_write_update_RA (value)
{
        if (R_X == 0)
                R_A = value;
        else
                R_A  = value & 0x0000FFFF;
}
```

How to Run with Symbolic Values?

- Consider the following virtual device program:

Virtual Device Code Snippet

```
…
mmio_write_update_RA (value)
{
        if (R_X == 0)
                R_A = value;
        else
                R_A  = value & 0x0000FFFF;
}
```



Virtual Device State

+ Event I: write (R_A, 0xC0FFEE)

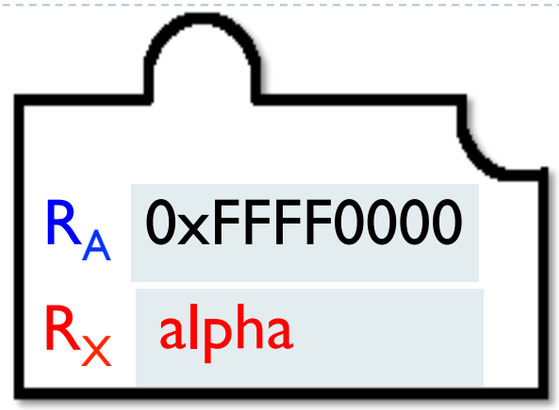
Suppose we have the above initial state and a given event …

How to Run with Symbolic Values?

- Consider the following virtual device program:

Virtual Device Code Snippet

```
…
mmio_write_update_RA (value)
{
        if (R_X == 0)
                R_A = value;
        else
                R_A  = value & 0x0000FFFF;
}
```



Virtual Device State

+ Event I: write (R_A, 0xC0FFEE)

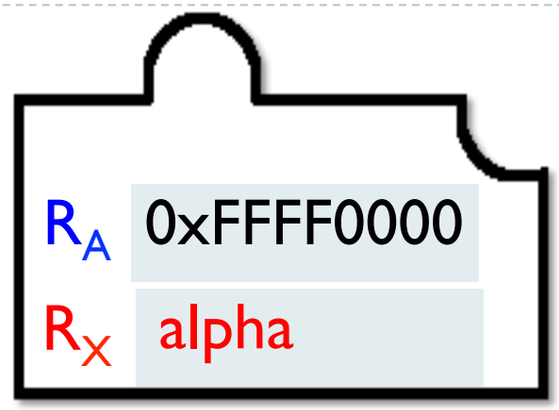
What will the virtual device state be after Event I?

Symbolic Execution

- Consider the following virtual device program:

Virtual Device Code Snippet

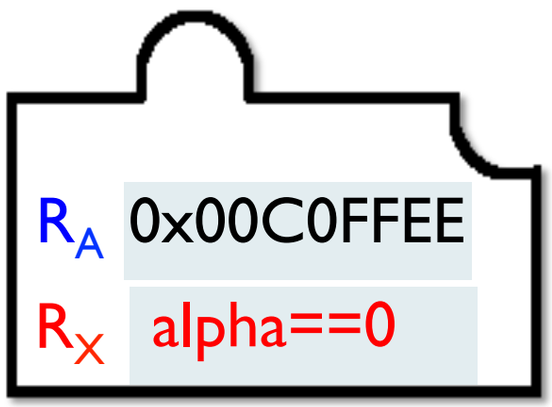
```
…
mmio_write_update_RA (value)
{
        if (RX == 0)
                RA = value;
        else
                RA  = value & 0x0000FFFF;
}
```



Virtual Device State

+ **Event I:** write (R_A, 0xC0FFEE)

If (alpha == 0)
Transaction #1



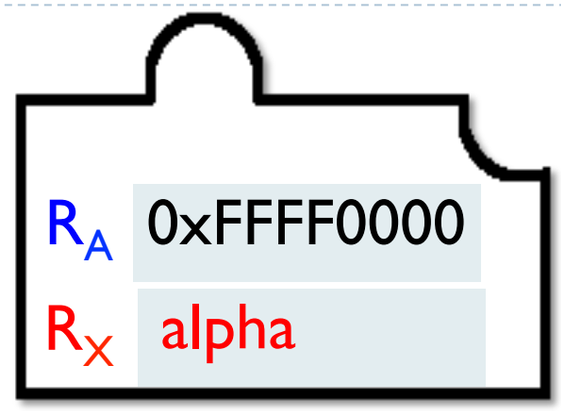
Virtual Device State

Symbolic Execution

- Consider the following virtual device program:

Virtual Device Code Snippet

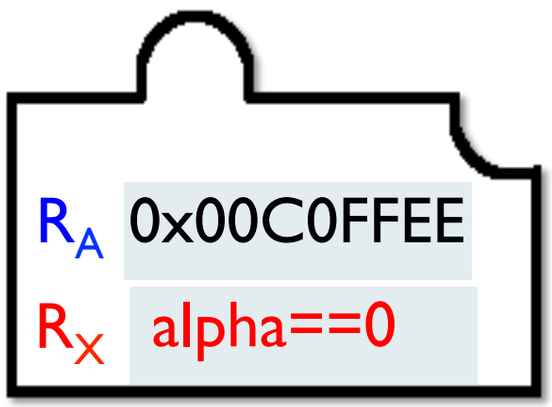
```
…
mmio_write_update_RA (value)
{
        if (RX == 0)
                RA = value;
        else
                RA  = value & 0x0000FFFF;
}
```



Virtual Device State

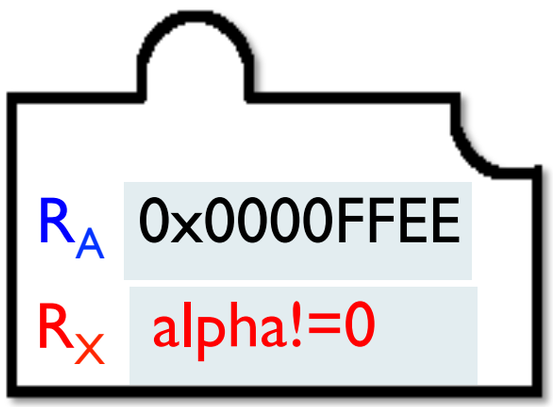
+ **Event I:** write (R_A, 0xC0FFEE)

If (alpha == 0)
Transaction #1



Virtual Device State

If (alpha != 0)
Transaction #2



Virtual Device State

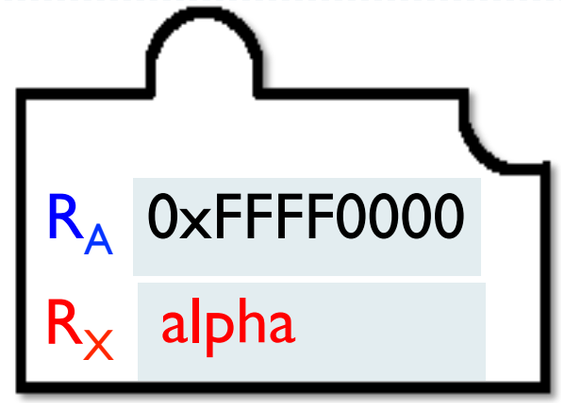
Searching for Inconsistencies

Given this Captured Trace:

…
Device State: [R_A== 0xFFFF0000]

Event 1: mmio_write (R_A, 0xC0FFEE)
Device State: [R_A== 0xFFEE]

…

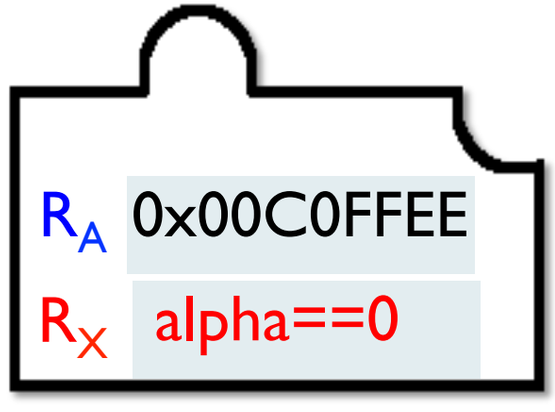


Virtual Device State

+ Event 1:
write (R_A, 0xC0FFEE)

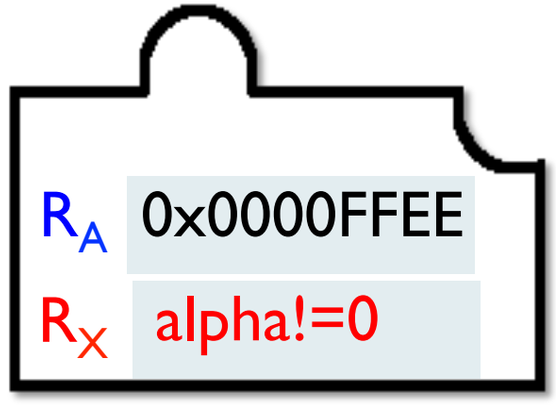
- Does one of the output virtual device states match the captured device state?

If (alpha == 0)
Transaction #1



Virtual Device State

If (alpha != 0)
Transaction #2



Virtual Device State

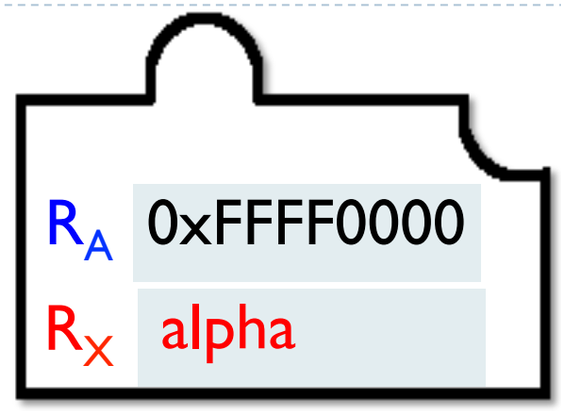
Searching for Inconsistencies

Given this Captured Trace:

…
Device State: [R_A== 0xFFFF0000]

Event 1: mmio_write (R_A, 0xC0FFEE)
Device State: [R_A== 0xFFEE]

…

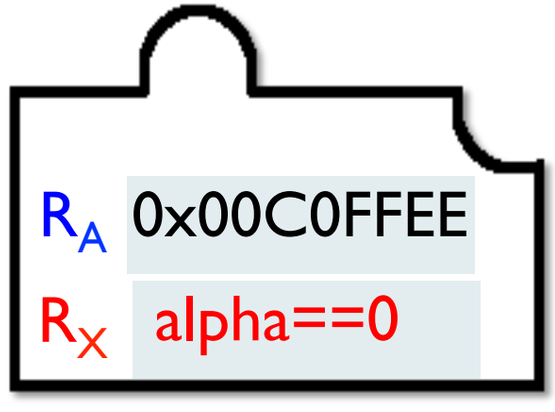


Virtual Device State

+ Event I:
write (R_A, 0xC0FFEE)

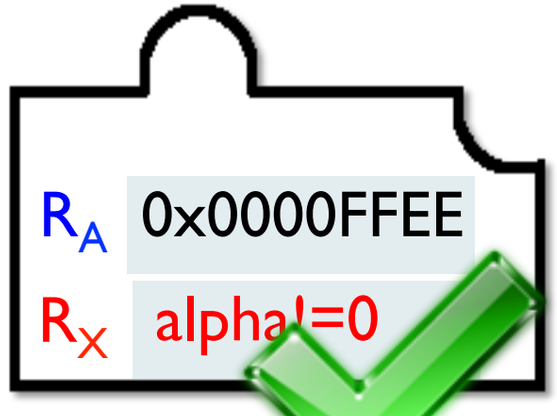
- Found a match, continue with the Transaction.
- If multiple matches found, follow each one.

If (alpha == 0)
Transaction #1



Virtual Device State

If (alpha != 0)
Transaction #2



Virtual Device State

Searching for Inconsistencies (cont.)

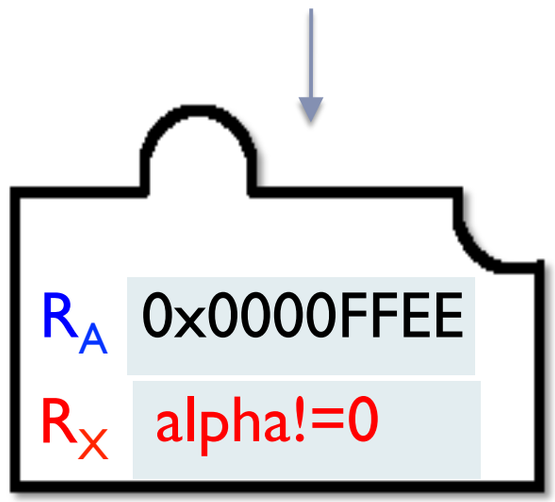
Given this Captured Trace:

…
Device State: [R_A== 0xFFFF0000]

Event 1: mmio_write (R_A, 0xC0FFEE)
Device State: [R_A== 0xFFEE]

Event II: mmio_write (R_A, 0x00BEEF00)
Device State: [R_A== 0xBEEF00]
…

Follow from previous transaction



Virtual Device State

+ **Event II:**
write (R_A, 0x00BEEF00

- Checking a trace with consecutive events

Searching for Inconsistencies (cont.)

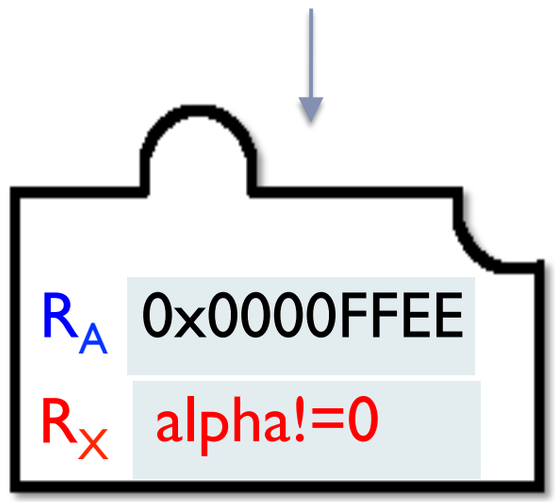
Given this Captured Trace:

…
Device State: [R_A== 0xFFFF0000]

Event 1: mmio_write (R_A, 0xC0FFEE)
Device State: [R_A== 0xFFEE]

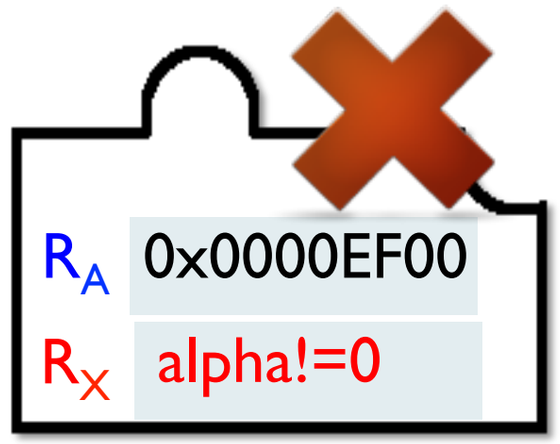
Event II: mmio_write (R_A, 0x00BEEF00)
Device State: [R_A== 0xBEEF00]
…

Follow from previous transaction



+ **Event II:**
write (R_A, 0x00BEEF00

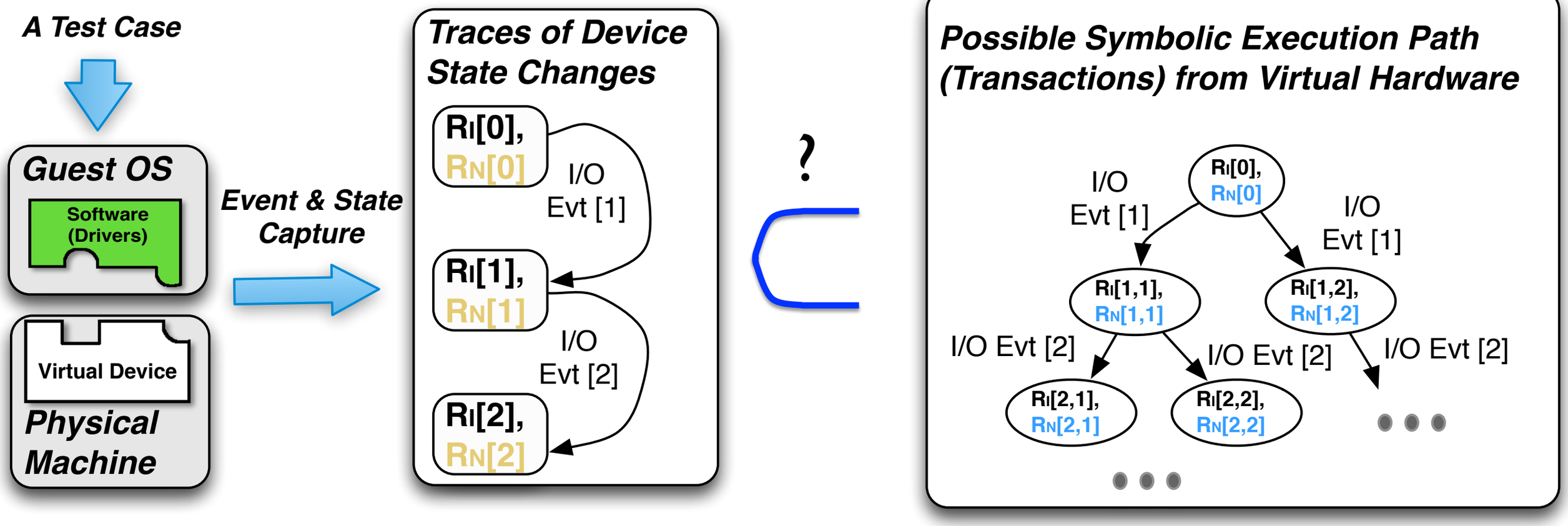
Virtual Device State



Virtual Device State

- Checking a trace with consecutive events
- No candidate match ➔ Inconsistency Found!

Detect Misbehaving Transactions



Results

- Evaluation
  - Use devices with well-tested virtual machines
    - QEMU/KVM virtual hardware devices
  - Focus on Network Interface Cards (NICs)
    - Intel EEPRO 100, E1000, X540
    - Broadcom BCM5751
- How to tell virtual vs. physical HW errors?
  - Specification
  - Hardware Errata

Example of Virtual HW Error (e1000)

- Test Event Sequence
  - MMIO writes to set the NIC MTU limit and receive queue tail,
  - Send a jumbo Ethernet frame to the NIC

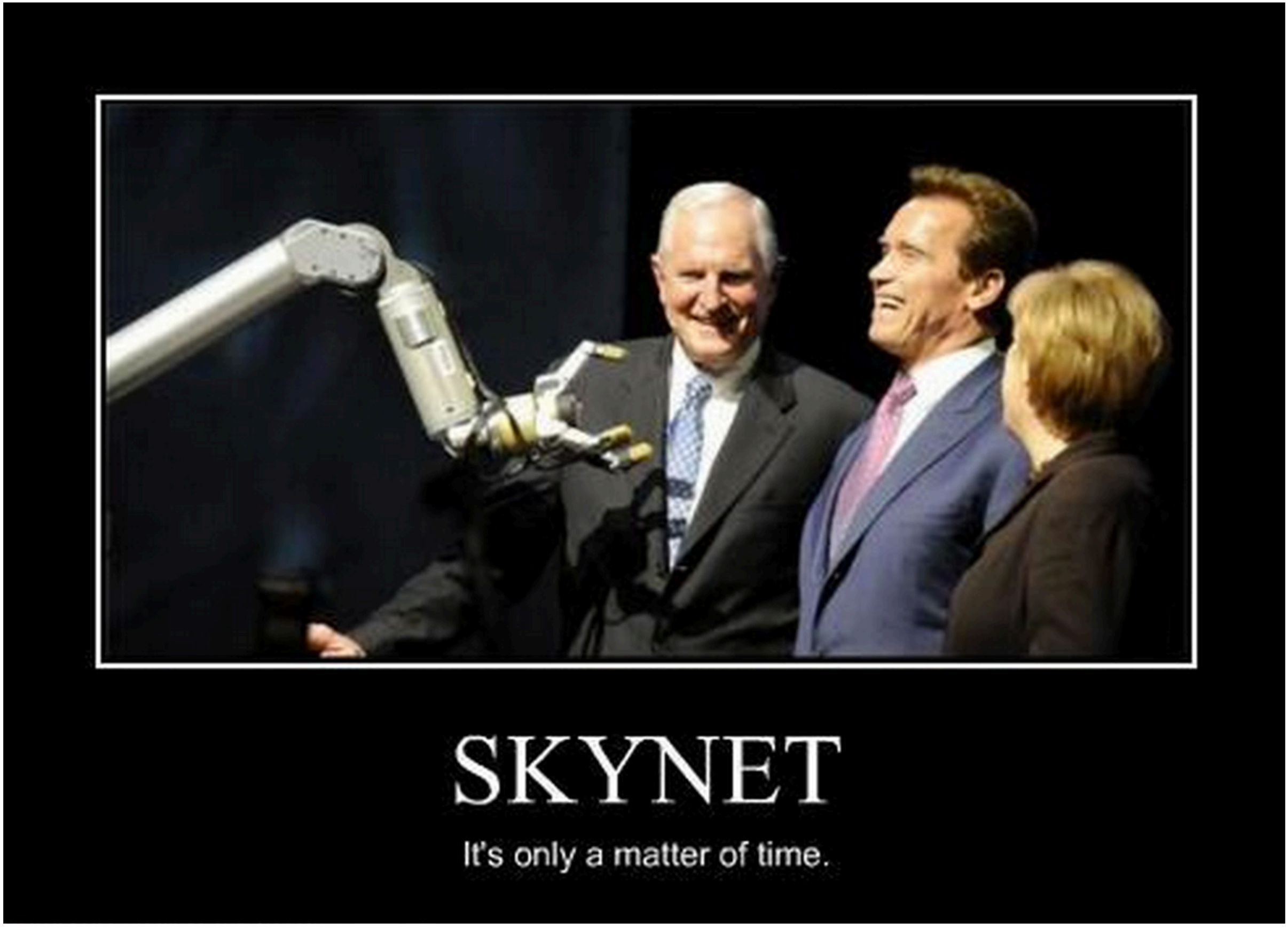
- Inconsistent values
  - **RLEC @ 0x04040 – Receive Length Error Count**
  - **PRC @ 0x0405C – Packets Received ([64-1522] Bytes) Count**
  - **BPRC @ 0x04078 – Broadcast Packets Received Count**
  - **MPRC @ 0x0407C – Multicast Packets Received Count**
  - **GPRC @ 0x04074 – Good Packet Received Count**

- **Inconsistencies resulted from a virtual hardware bug**

- **Reported to Redhat (QEMU) and confirmed as a severe bug.**

Summary

- Security of Virtual Machines and Cloud Platforms
  - Verify Virtual Machine Implementation
    - Compare virtual and physical hardware.
  - Verify Hardware Behavior after Manufacture
    - Dynamic Behavior Comparison
- Auto SW Vulnerability Scan and Flaw Finding
  - Critical Errors are not limited to traditional SW security bugs
    - Logical errors
    - Need more "Model" checking



http://oddnews.cosmobc.com/2010/05/18/skynet/

Thanks for your time!