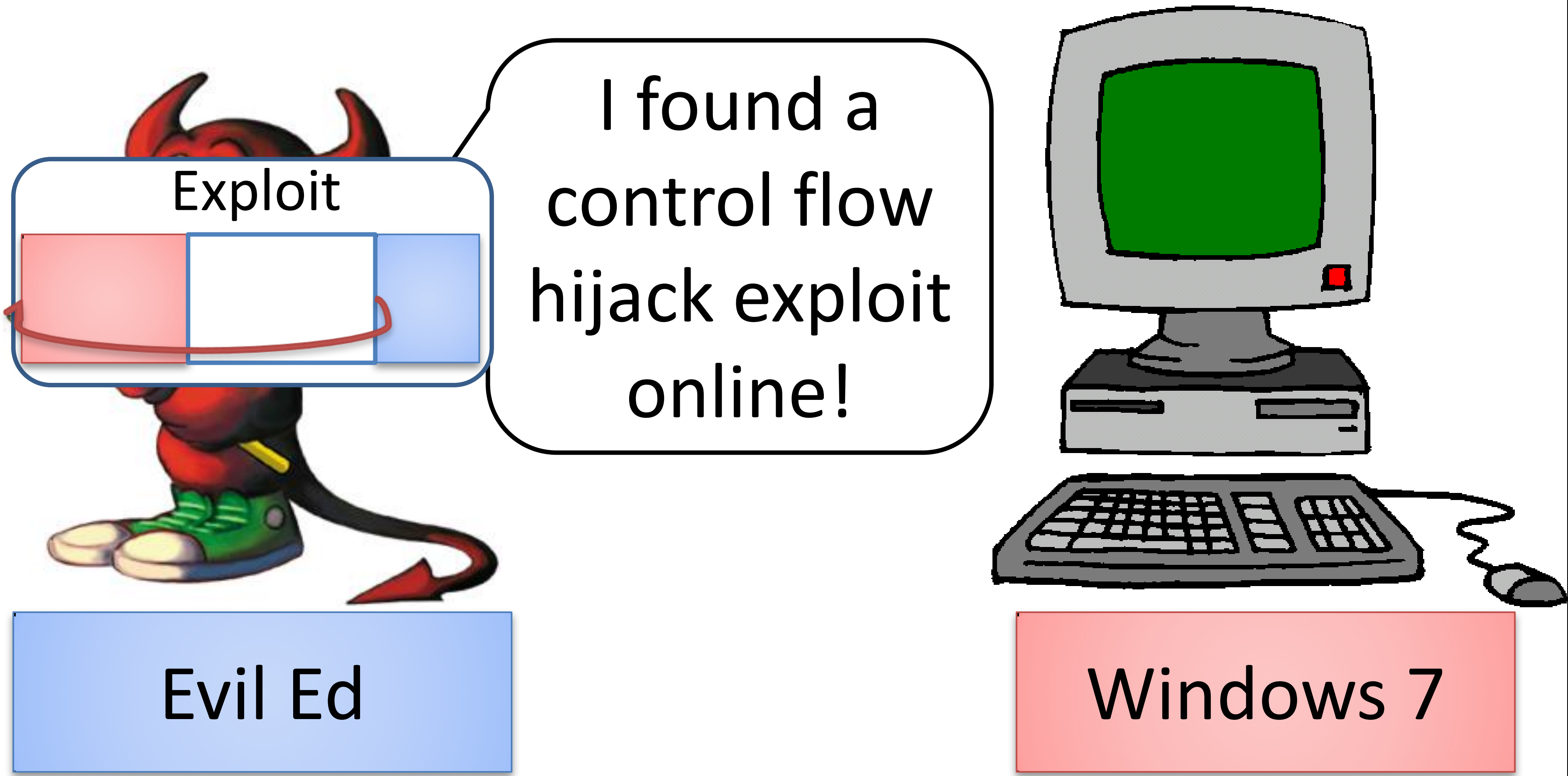
Q: Exploit Hardening Made Easy

Edward J. Schwartz, Thanassis Avgerinos, and David Brumley

Carnegie Mellon University

Downloading Exploits



A problem has been detected and windows has been shut down to prevent damage to your computer.

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to be sure you have adequate disk space. If a driver is identified in the Stop message, disable the driver or check with the manufacturer for driver updates. Try changing video adapters.

Check with your hardware vendor for any BIOS updates. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x0000007E (0xC0000005,0xF88FF190,0x0xF8975BA0,0xF89758A0)

*** EPUSBDSK.sys - Address F88FF190 base at FF88FE000, datestamp 3b9f3248

Beginning dump of physical memory

Downloading Exploits

Why didn't the exploit work?

Evil Ed

Windows 7

Causes of Broken Exploits

1. Exploit used OS/binary-specific tricks/features

2. OS Defenses

OS Defenses

- Modern OS defenses are designed to make exploiting difficult
  - **ASLR:** Address Space Layout Randomization
  - **DEP:** Data Execution Prevention
  - Do not guarantee control flow integrity

- **How difficult?**

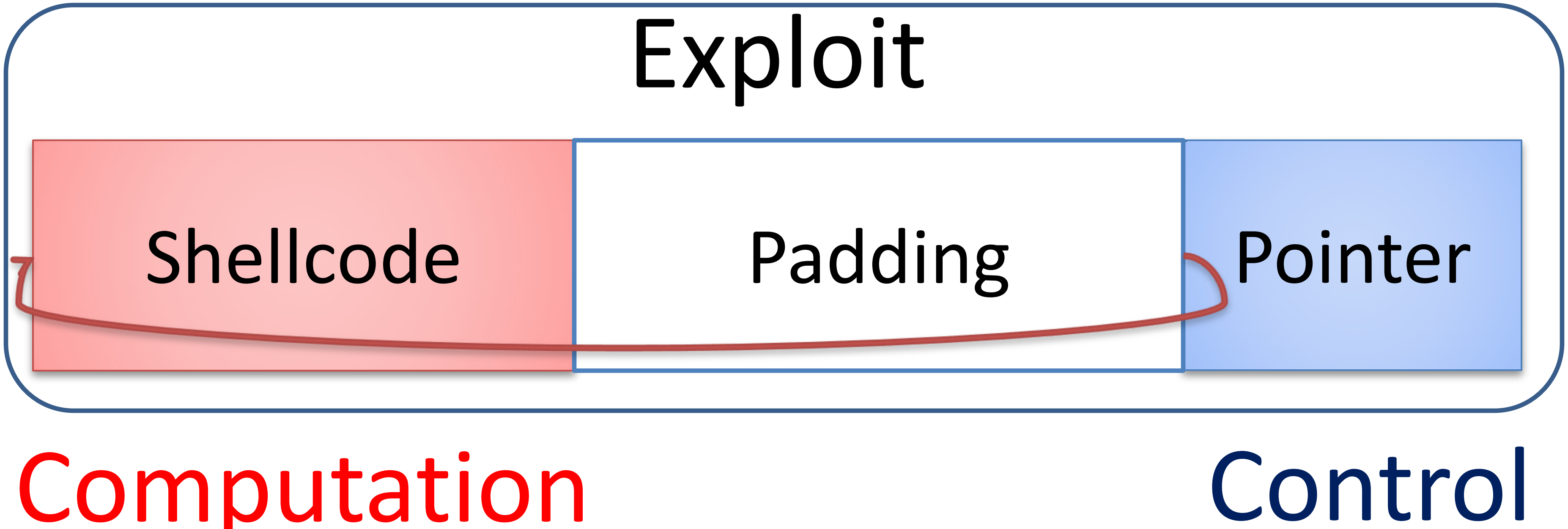
Exploit hardening:

Modifying exploits to bypass defenses

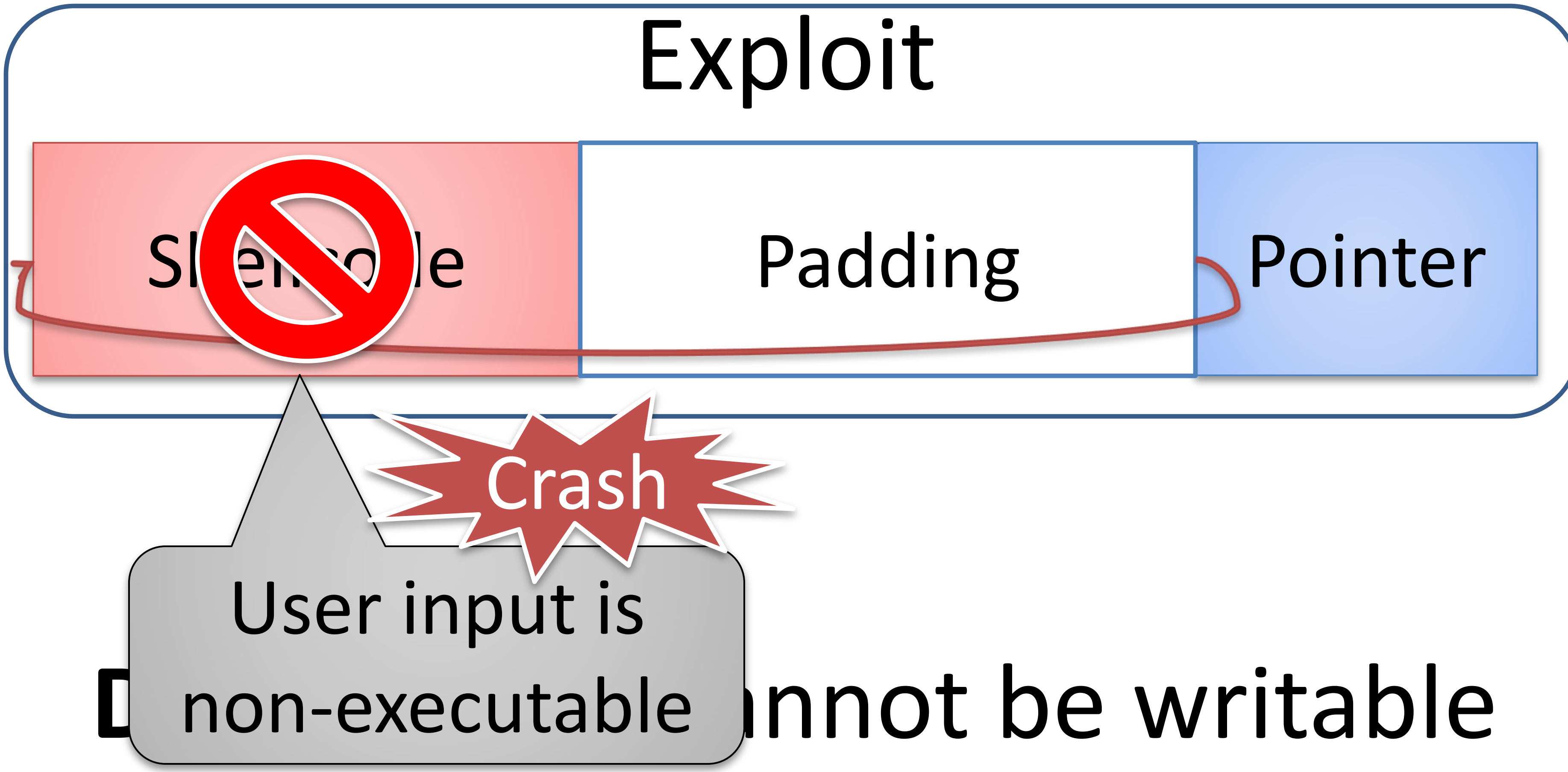
Overview

- **Background: Defenses and Return Oriented Programming (ROP)**
- Q: ROP + Hardening
  - Automatic ROP
  - Automatic Hardening
- Evaluation
- Limitations
- Conclusion

Simple Exploit



Data Execution Prevention (DEP)



D…annot be writable and executable

Bypassing DEP

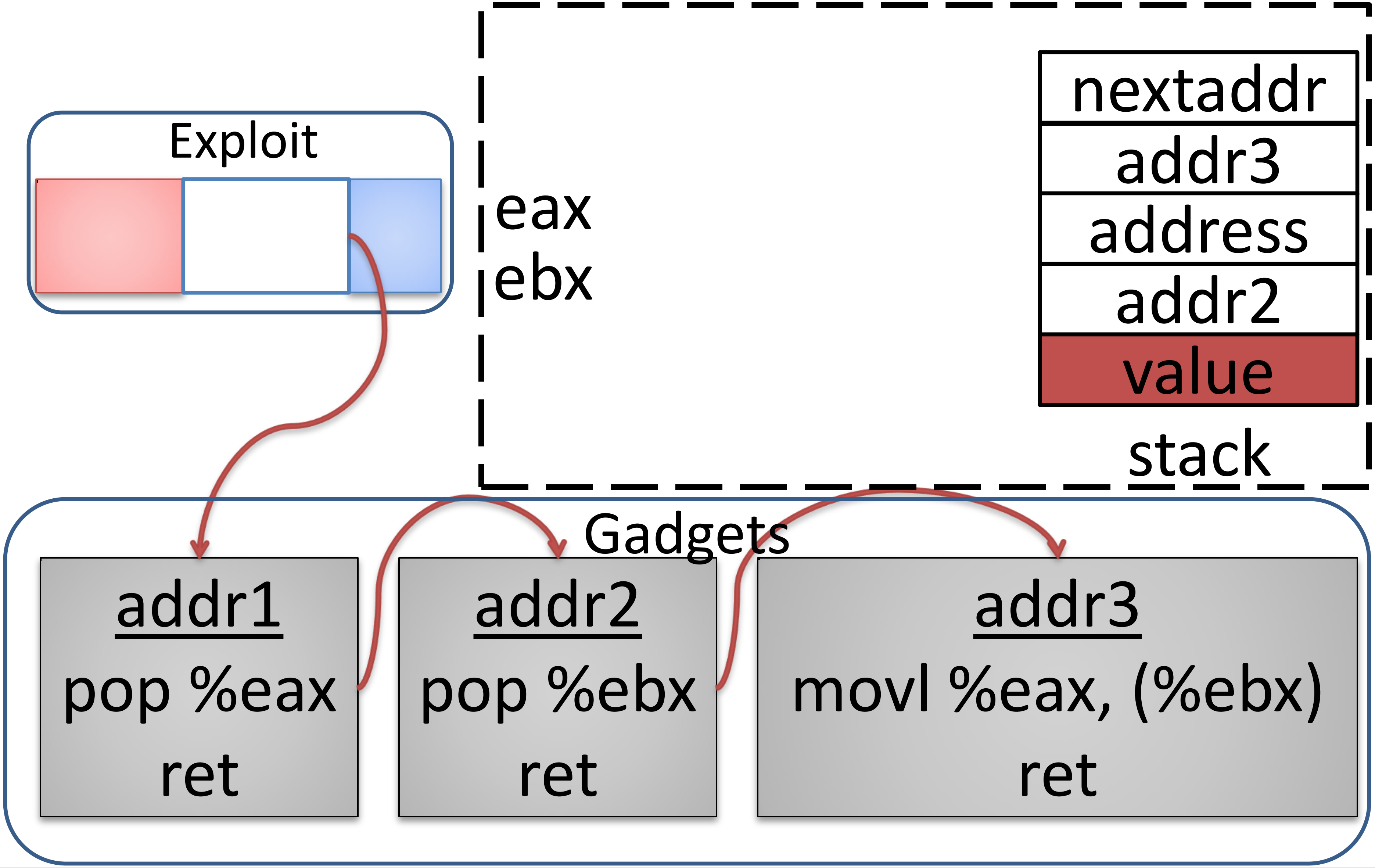
- **Goal:** Specify exploit computation even when DEP is enabled

- **Return Oriented Programming** [S07]
  – Use existing instructions from program in special order to encode computation

Return Oriented Programming

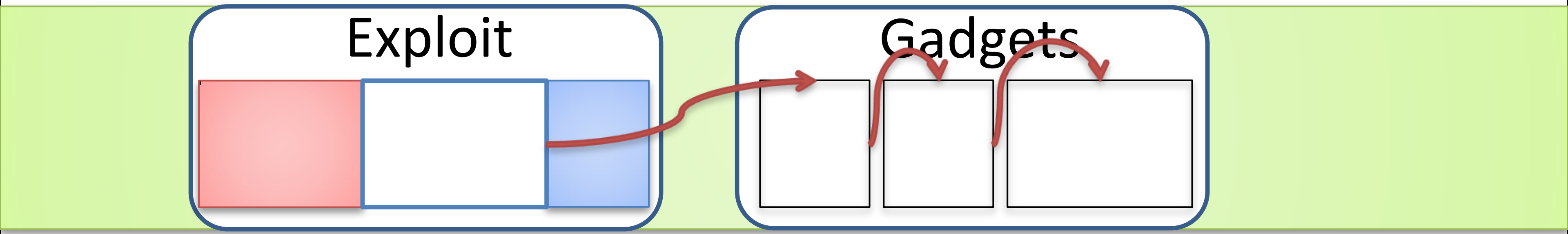
Example: How can we write to memory without shellcode?

Return Oriented Programming

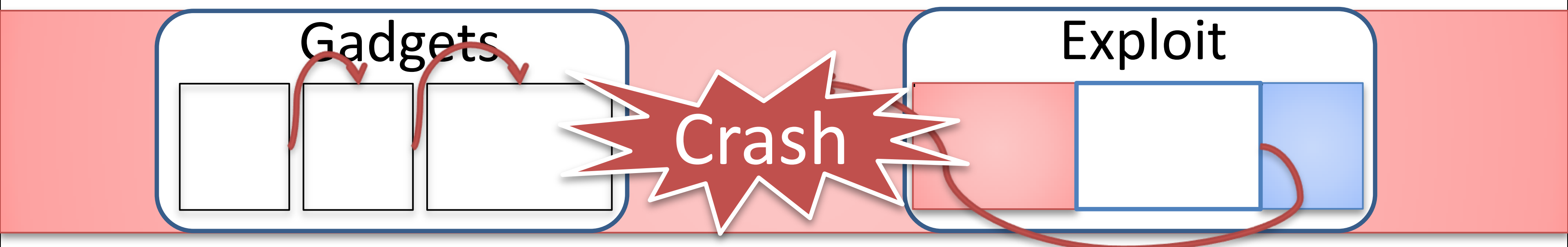


Address Space Layout Randomization (ASLR)

ASLR disabled



ASLR enabled



ASLR: Addresses are unpredictable

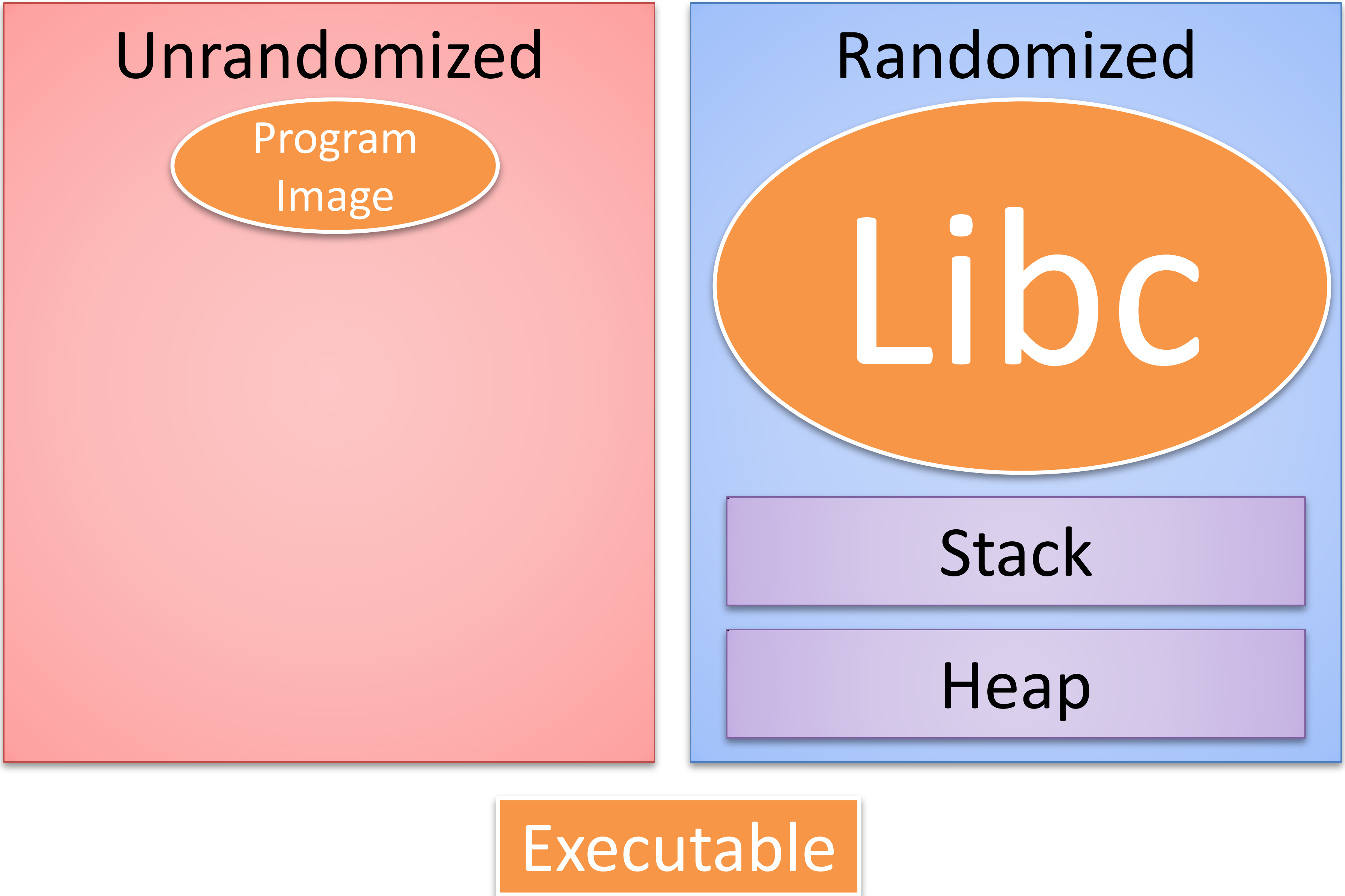
Return Oriented Programming + ASLR

- **Bad news:** Randomized code can't be used for ROP

- **Good news:** ASLR implementations leave small amounts of code unrandomized

Evil Ed

ASLR in Linux (Example)



Consequences

- **Challenge:** Program image is often the only unrandomized code
  - Small
  - Program-specific

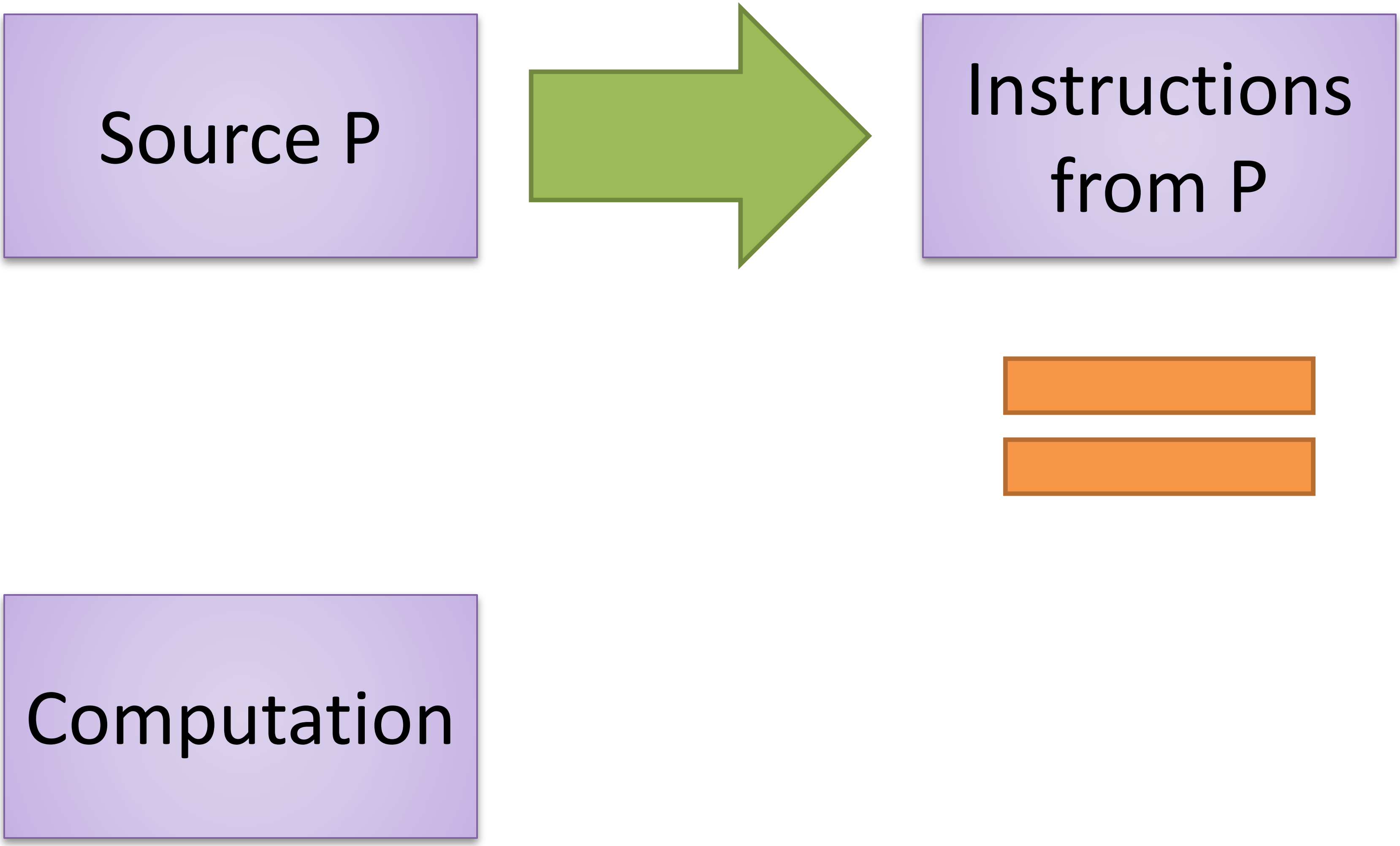
- Prior work on ROP assumes unrandomized large code bases; can't simply reuse

- We developed new automated ROP techniques for targeting the program image

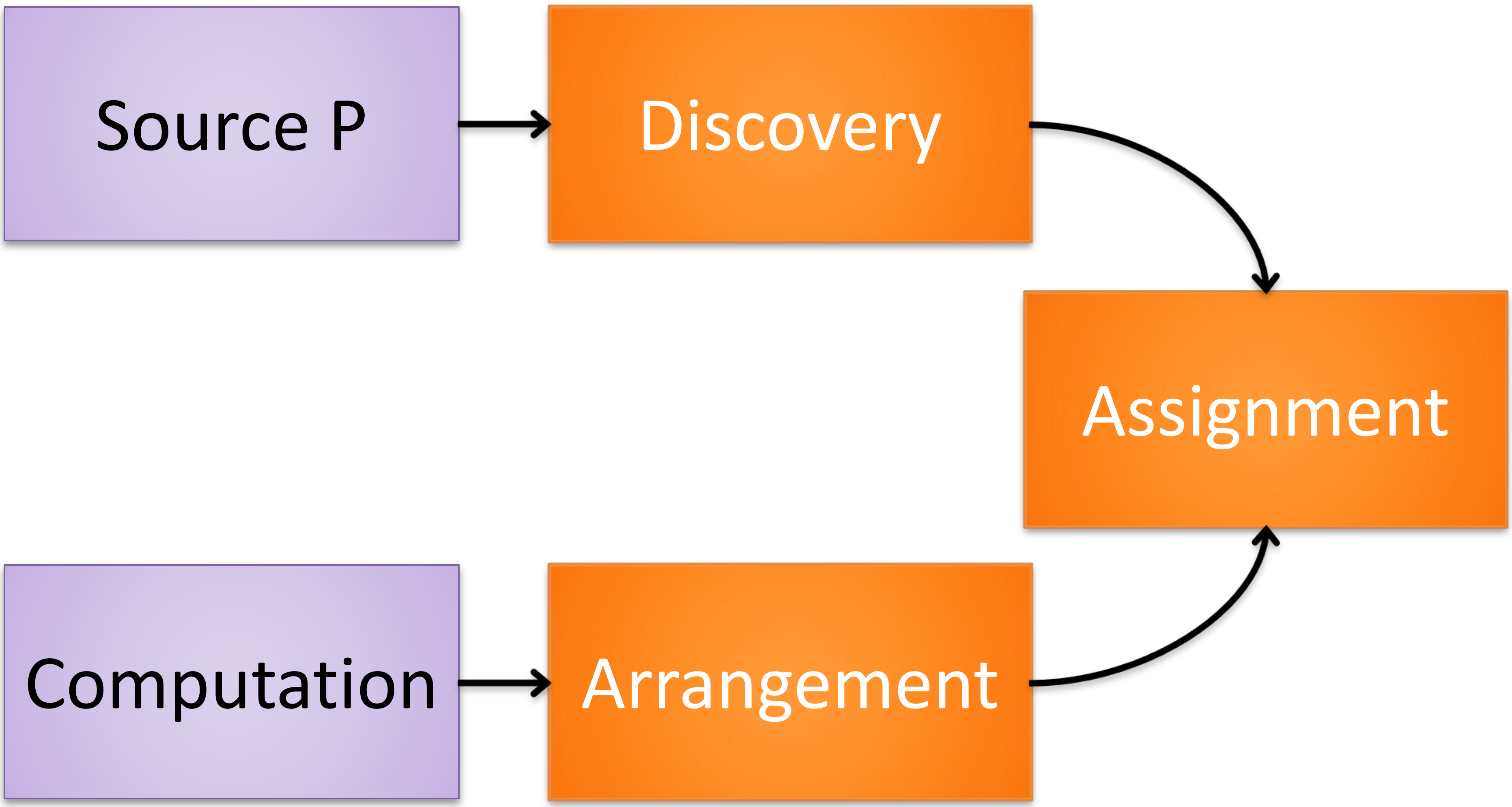
Overview

- Background: Defenses and Return Oriented Programming (ROP)
- Q: ROP + Hardening
  - **Automatic ROP**
  - Automatic Hardening
- Evaluation
- Limitations
- Conclusion

Automatic ROP Overview



ROP Overview



Gadget Discovery

- **Gadget Discovery:** Does instruction sequence do something we can use for our computation?
- Fast randomized test for **every program location** (thousands or millions)

```
sbb %eax, %eax;
neg %eax; ret
```

Randomized Testing

Before

EAX	0x0298a7bc
CF	0x1
ESP	0x81e4f104

```
sbb %eax, %eax;
neg %eax; ret
```

After

EAX	0x1
ESP	0x81e4f108
EBX	0x0298a7bc

OutReg <- InReg

Semantic Definition For Move

If 10 random runs satisfy a semantic definition, then Q **probably** found a gadget of that type

Q’s Gadget Types

Gadget Type	Semantic Definition	Real World Example
MoveRegG	Out <- In	xchg %eax, %ebp; ret
LoadConstG	Out <- Constant	pop %ebp; ret
ArithmeticG	Out <- In1 + In2	add %edx, %eax; ret
LoadMemG	Out <- M[Addr + Offset]	movl 0x60(%eax), %eax; ret
StoreMemG	M[Addr + Offset] <- In	mov %dl, 0x13(%eax); ret
ArithmeticLoadG	Out +<- M[Addr + Offset]	add 0x1376dbe4(%ebx), %ecx; (…); ret
ArithmeticStoreG	M[Addr + Offset] +<- In	add %al, 0x5de474c0(%ebp); ret

Q's Gadget Types

Gadget Type	Semantic Definition	Real World Example
MoveRegG	Out <- In	xchg %eax, %ebp; ret
LoadConstG	Out <- Constant	pop %ebp; ret
ArithmeticG	Out <- In1 + In2	add %edx, %eax; ret
LoadMemG	Out <- M[Addr + Offset]	movl 0x60(%eax), %eax; ret
StoreMemG	M[Addr + Offset] <- In	mov %dl, 0x13(%eax); ret
ArithmeticLoadG	Out +<- M[Addr + Offset]	add 0x1376dbe4(%ebx), %ecx; (…); ret
ArithmeticStoreG	M[Addr + Offset] +<- In	add %al, 0x5de474c0(%ebp); ret

Q's Gadget Types

Gadget Type	Semantic Definition	Real World Example
MoveRegG	Out <- In	xchg %eax, %ebp; ret
LoadConstG	Out <- Constant	pop %ebp; ret
ArithmeticG	Out <- In1 + In2	add %edx, %eax; ret
LoadMemG	Out <- M[Addr + Offset]	movl 0x60(%eax), %eax; ret
StoreMemG	M[Addr + Offset] <- In	mov %dl, 0x13(%eax); ret
ArithmeticLoadG	Out +<- M[Addr + Offset]	add 0x1376dbe4(%ebx), %ecx; (…); ret
ArithmeticStoreG	M[Addr + Offset] +<- In	add %al, 0x5de474c0(%ebp); ret

Randomized Testing

- Randomized testing tells us we **likely** found a gadget
  - Fast; filters out many candidates
  - Enables more expensive second stage

- **Second stage: SMT-based gadget discovery**
  - Gadget discovery is program verification

SMT-Based Gadget Discovery

sbb %eax, %eax
neg %eax; ret

EAX <- CF

[D76]

Weakest Precondition →

F

F

SMT Validity Check →

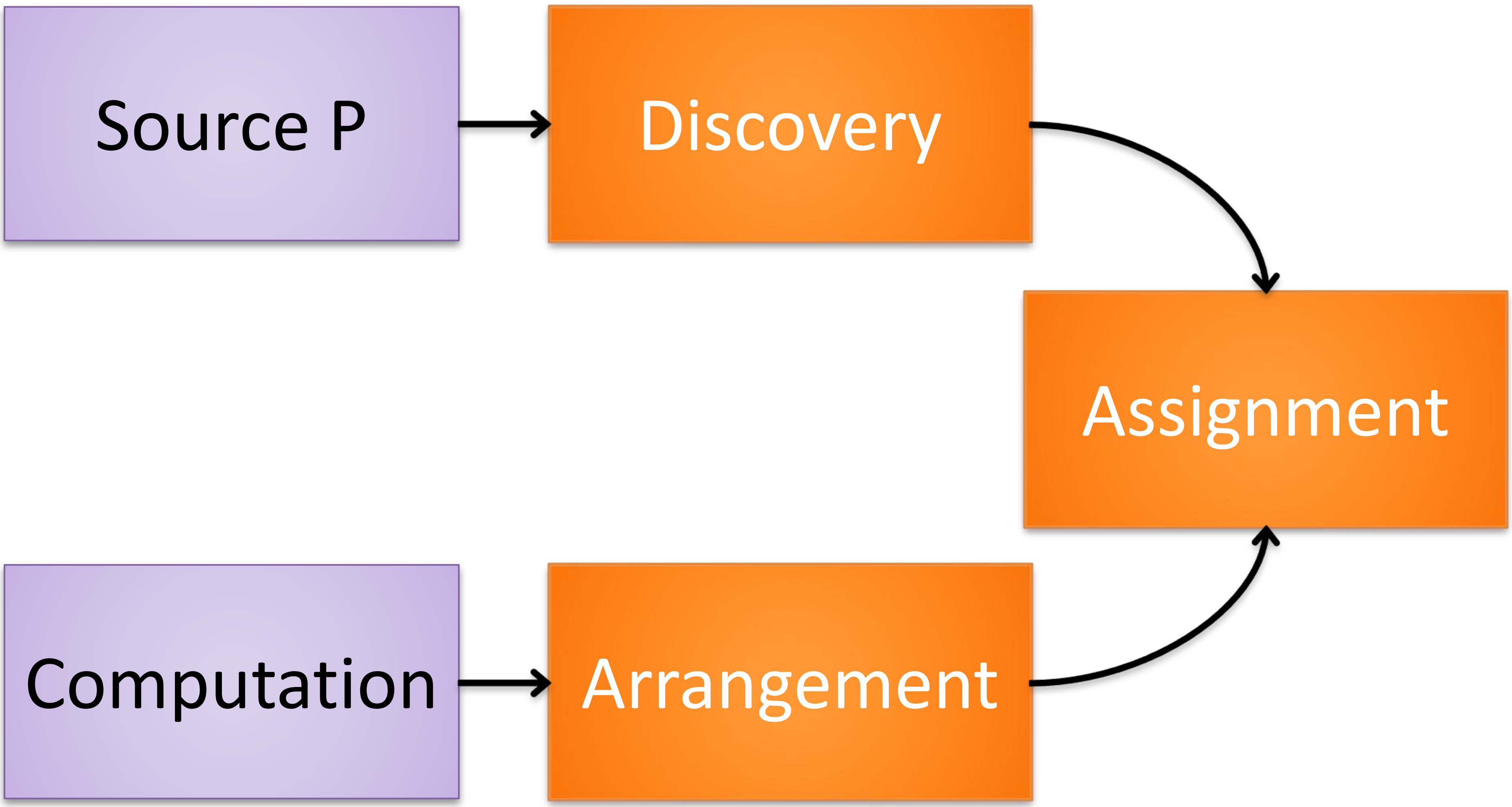
Valid (Gadget)
Invalid (not Gadget)

SMT-Based Gadget Discovery

- Q is better at finding gadgets than I am!

imul $1, %eax, %ebx ret	Move %eax to %ebx
lea (%ebx,%ecx,1), %eax ret	Store %ebx+%ecx in %eax
sbb %eax, %eax; neg %eax ret	Move carry flag to %eax

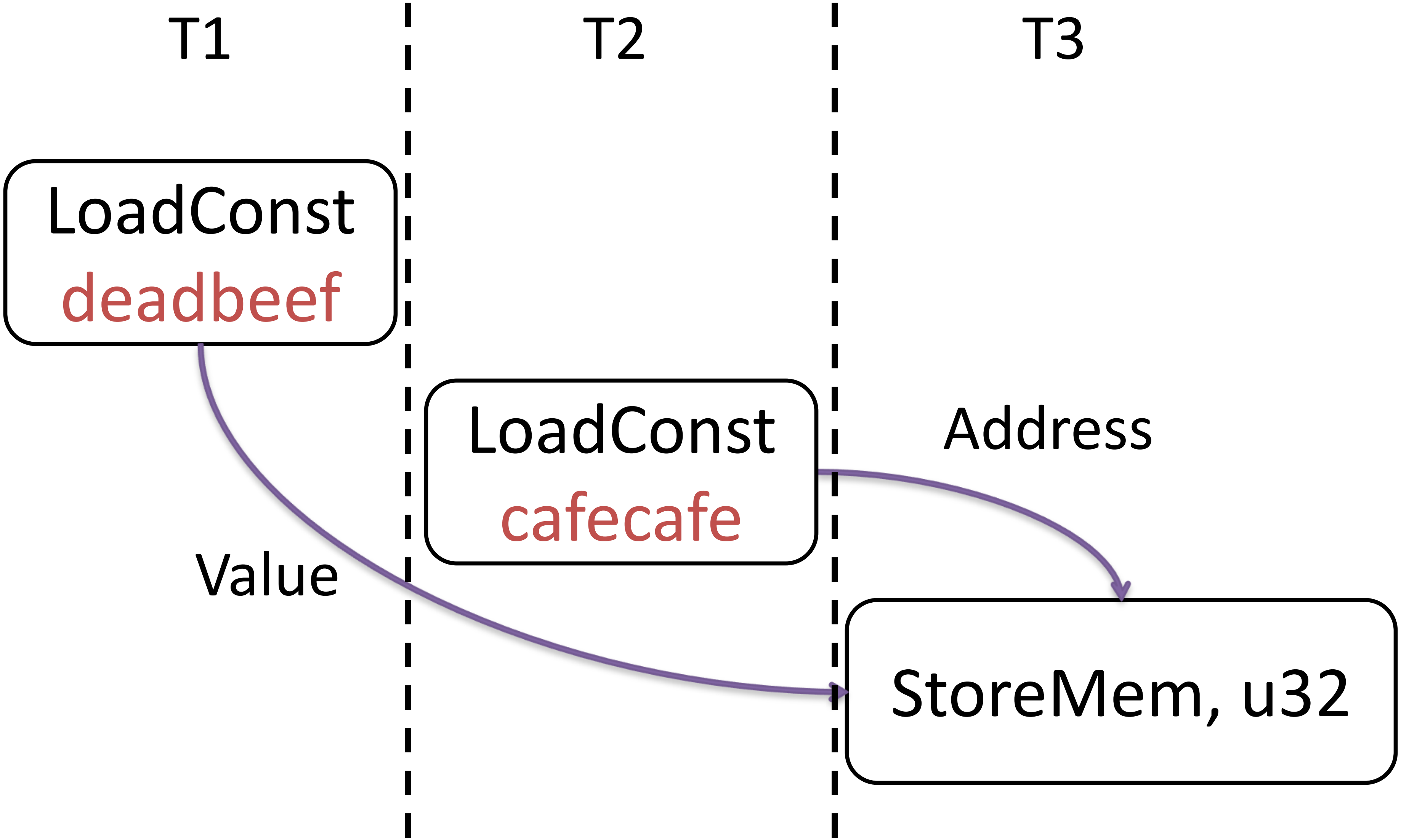
ROP Overview



Gadget Arrangement

- **Gadget Arrangement:** How can gadget types be combined to implement a computation?
- Alternate view: Compile user computation for gadget type architecture
- **Example**:
  M[0xcafecafe] := 0xdeadbeef

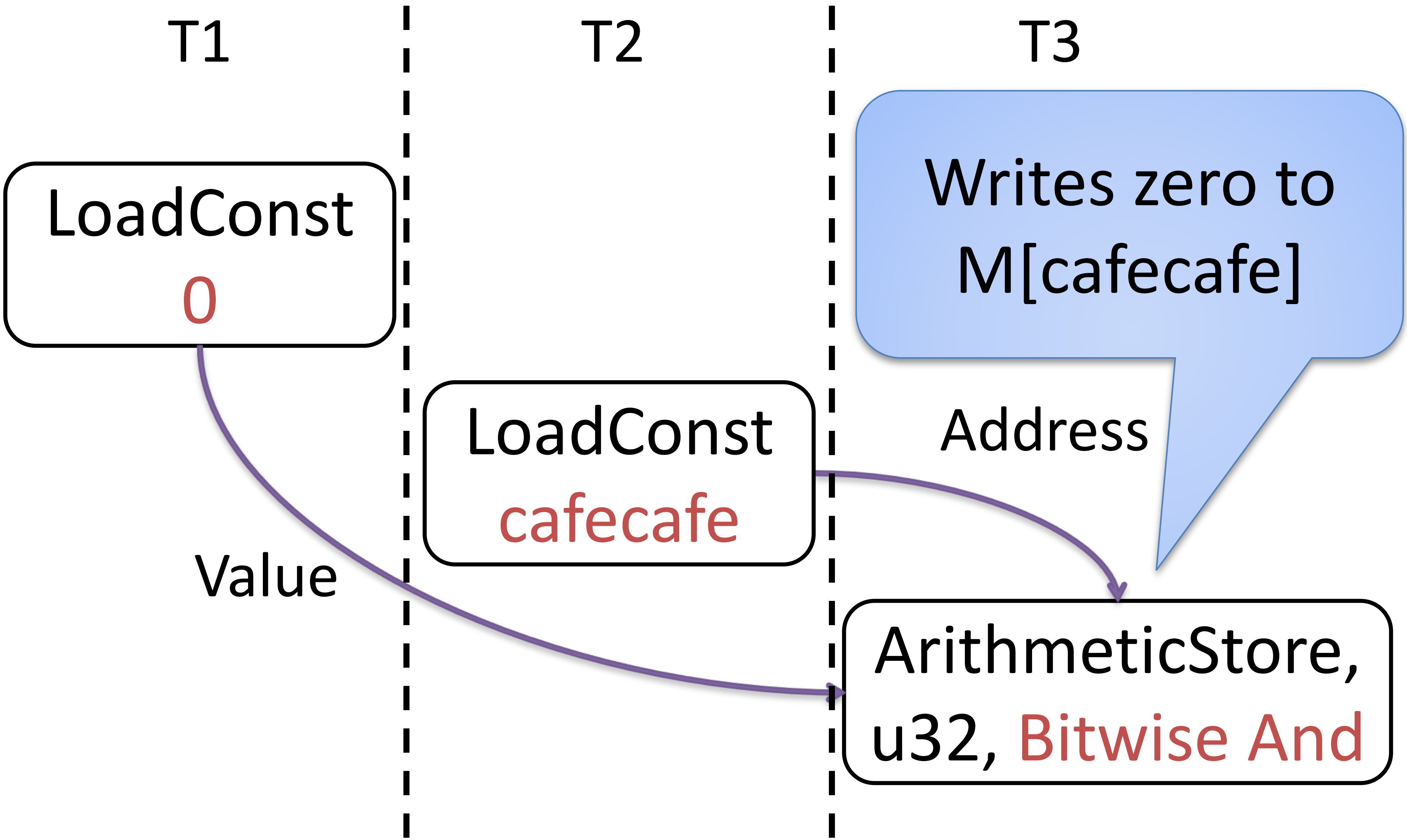
Arrangement: Storing to Memory



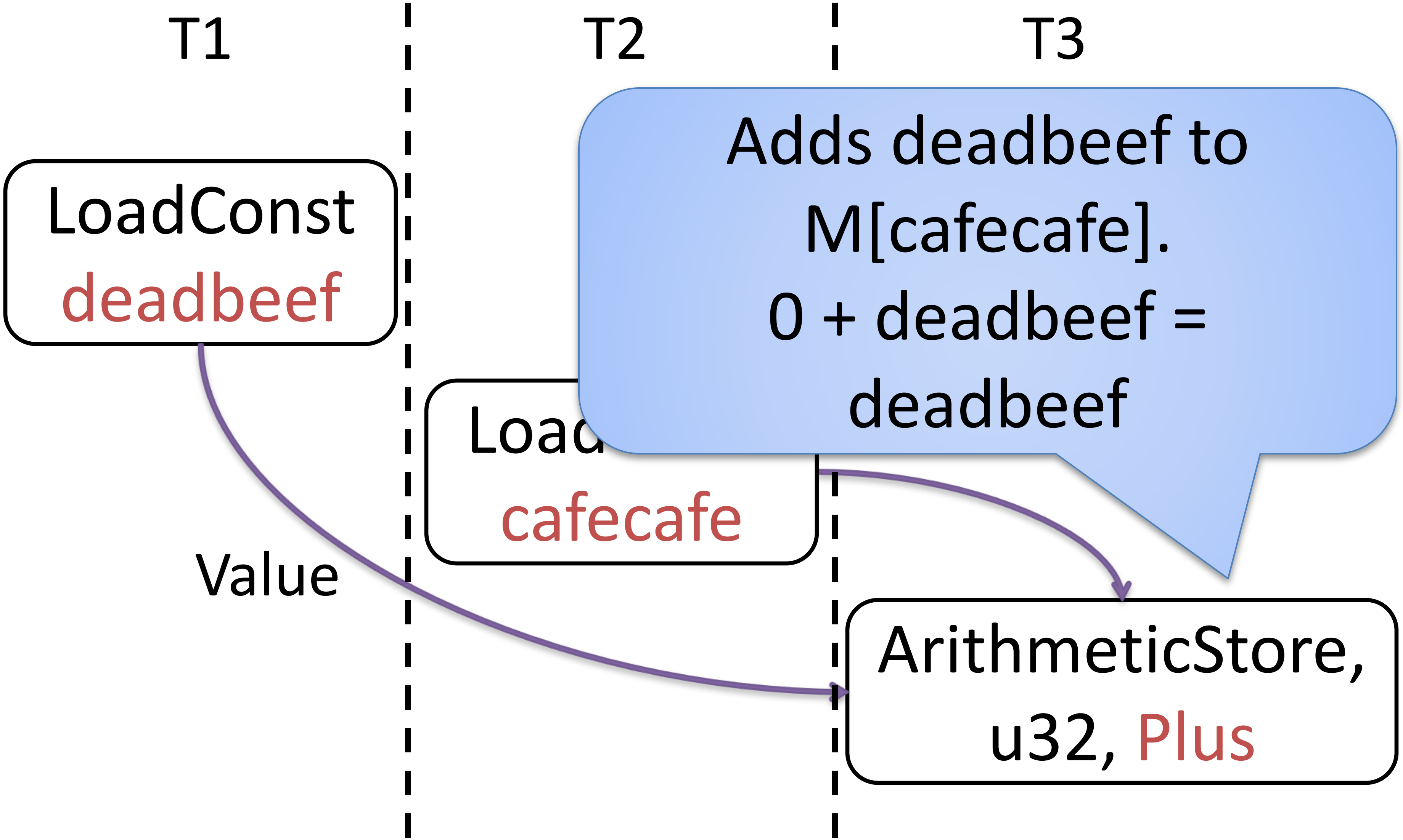
Gadget Arrangement

How can we write to memory without StoreMem?

Arrangement: Storing to Memory



Arrangement: Storing to Memory

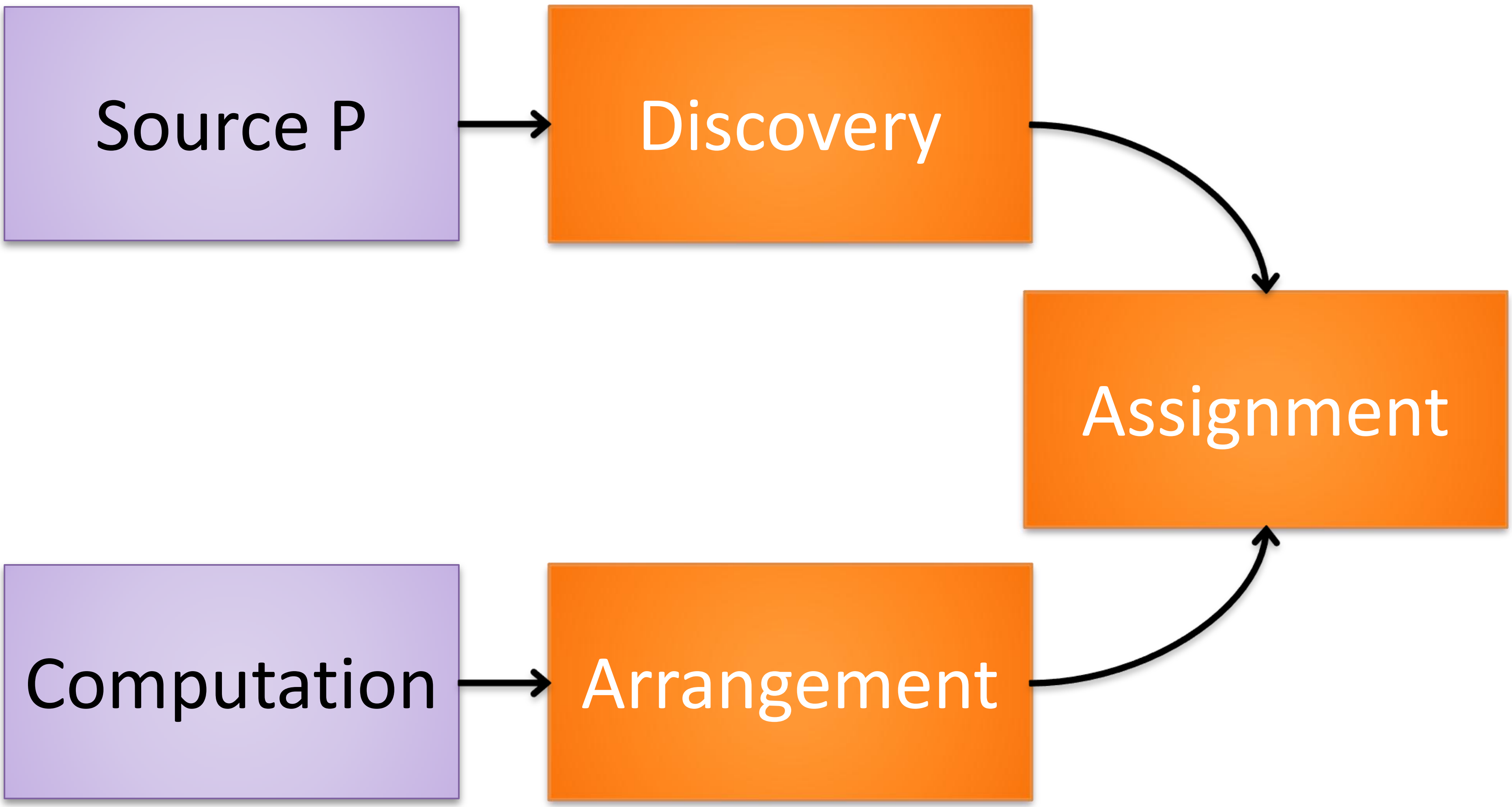


Gadget Arrangement

- Gadgets types are often unavailable
  - Synthesize alternatives on the fly

- Flexible arrangement rules are necessary for small code bases

ROP Overview

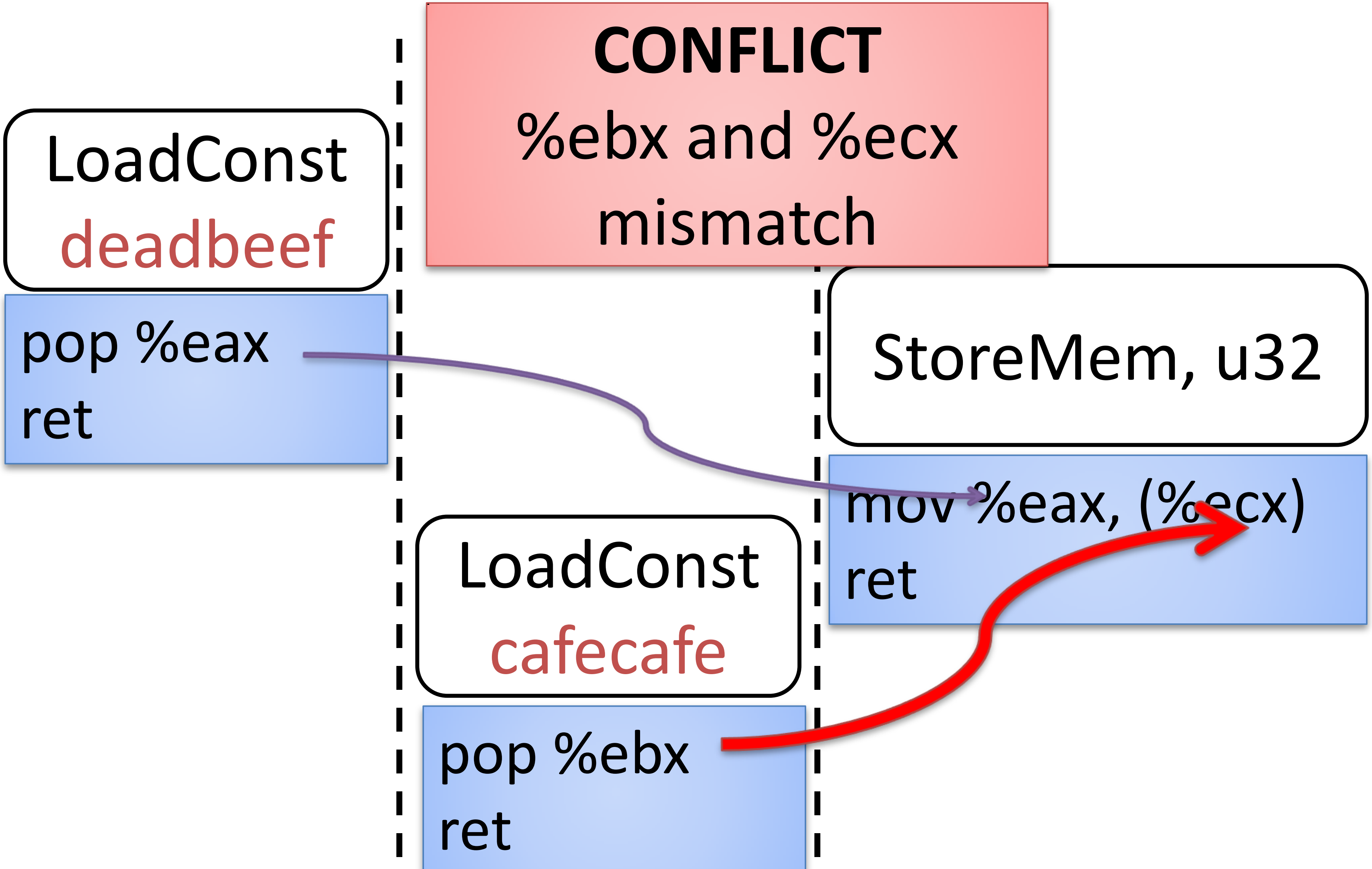


Assignment

- **Gadget Assignment:** Assign concrete gadgets found in source program to arrangements

- Assignments must be **compatible**

Assignment: Register Mismatch



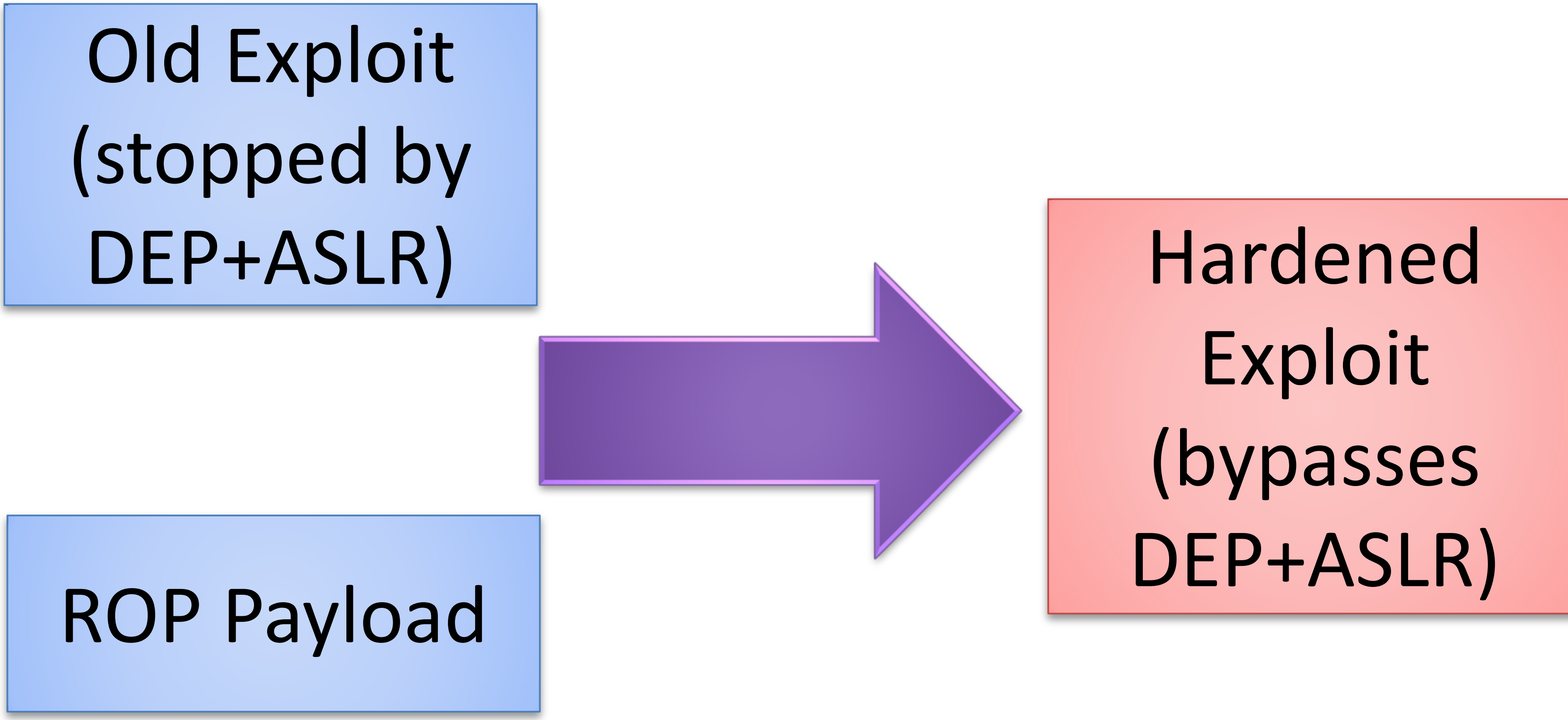
Gadget Assignment

- Need to search over
  - Gadgets
  - Schedules
- We developed dynamic programming approach to find assignment
- Easy to print payload bytes with assignment

Overview

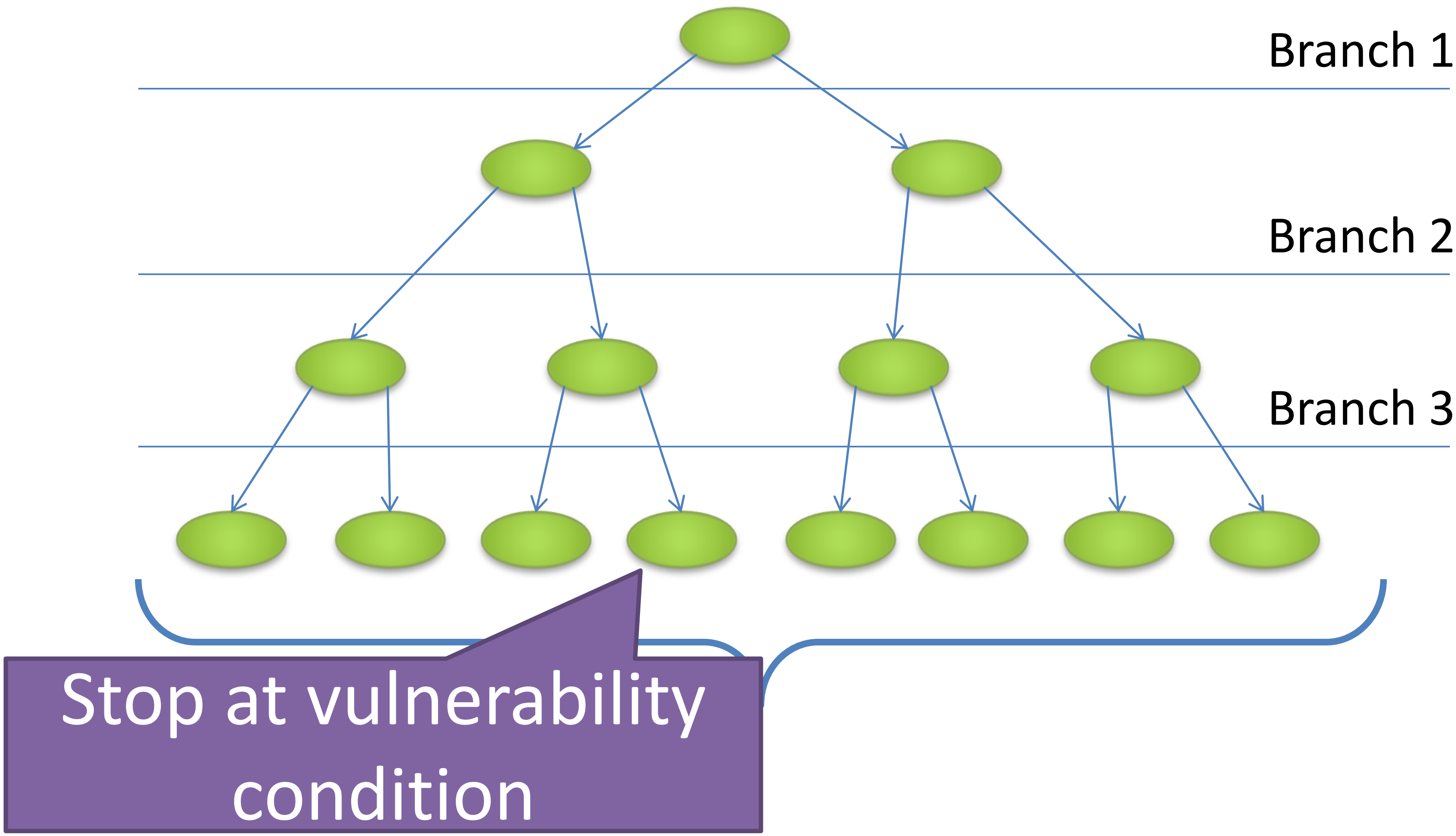
- Background: Defenses and Return Oriented Programming (ROP)
- Q: ROP + Hardening
  - Automatic ROP
  - **Automatic Hardening**
- Evaluation
- Limitations
- Conclusion

Exploit Hardening

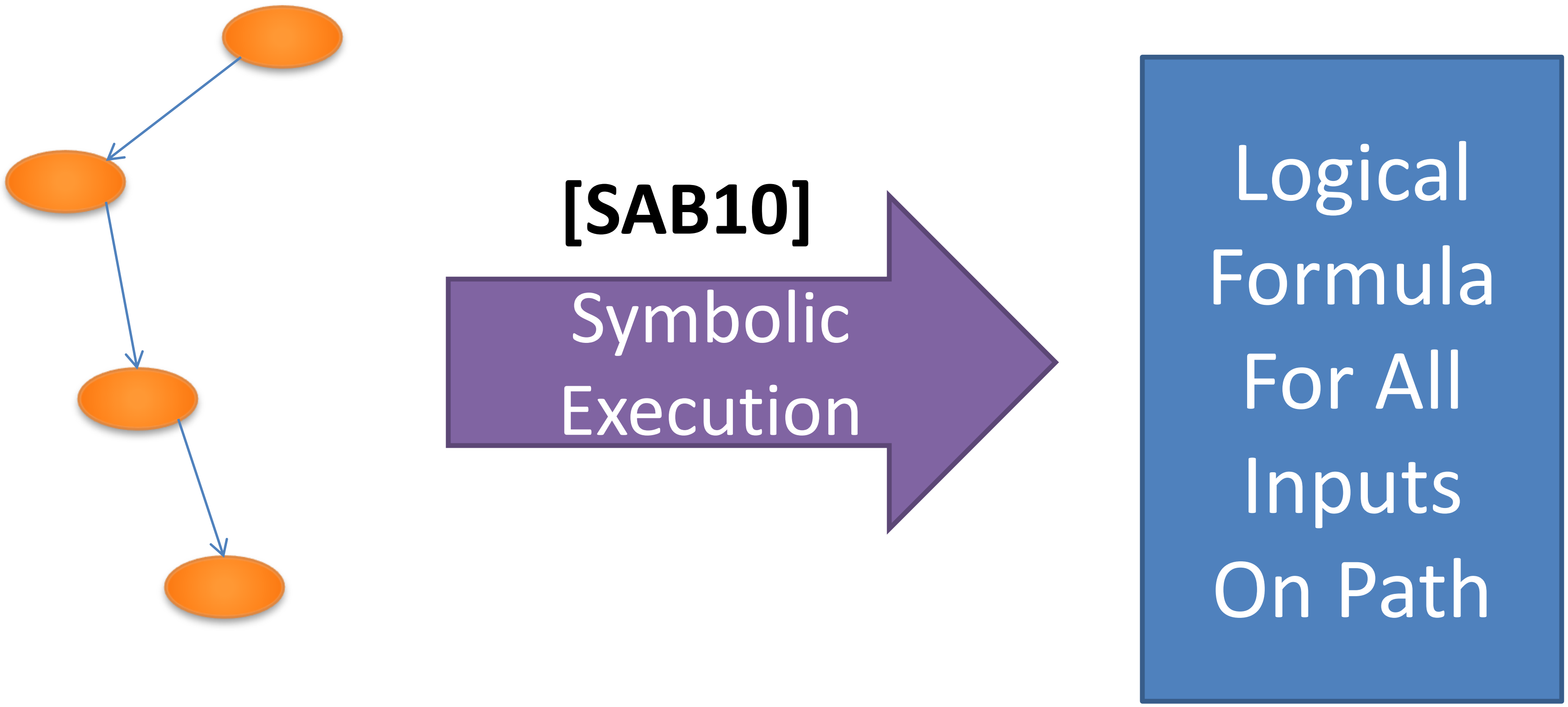


Trace-based Analysis

- Record P on the old exploit



Reasoning about Executions



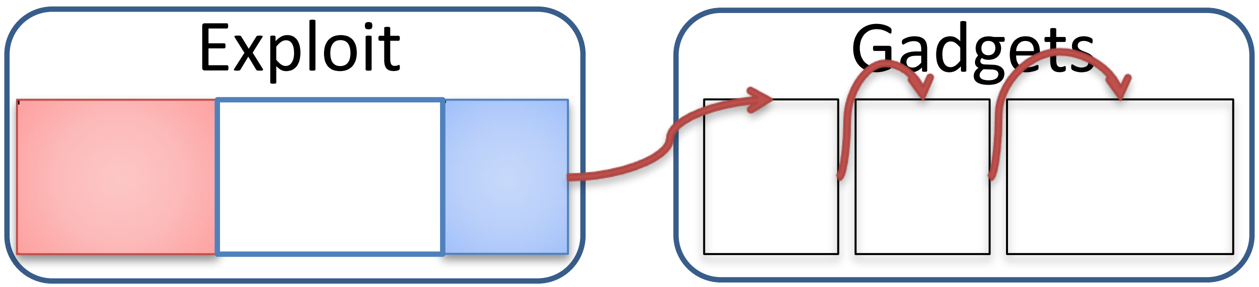
Exploit Constraints

Path

Exploit

Exploit Constraints

How do we ensure the ROP payload gets in the exploit?



M[ESP] = &gadget1

M[ESP+off1] = &gadget2

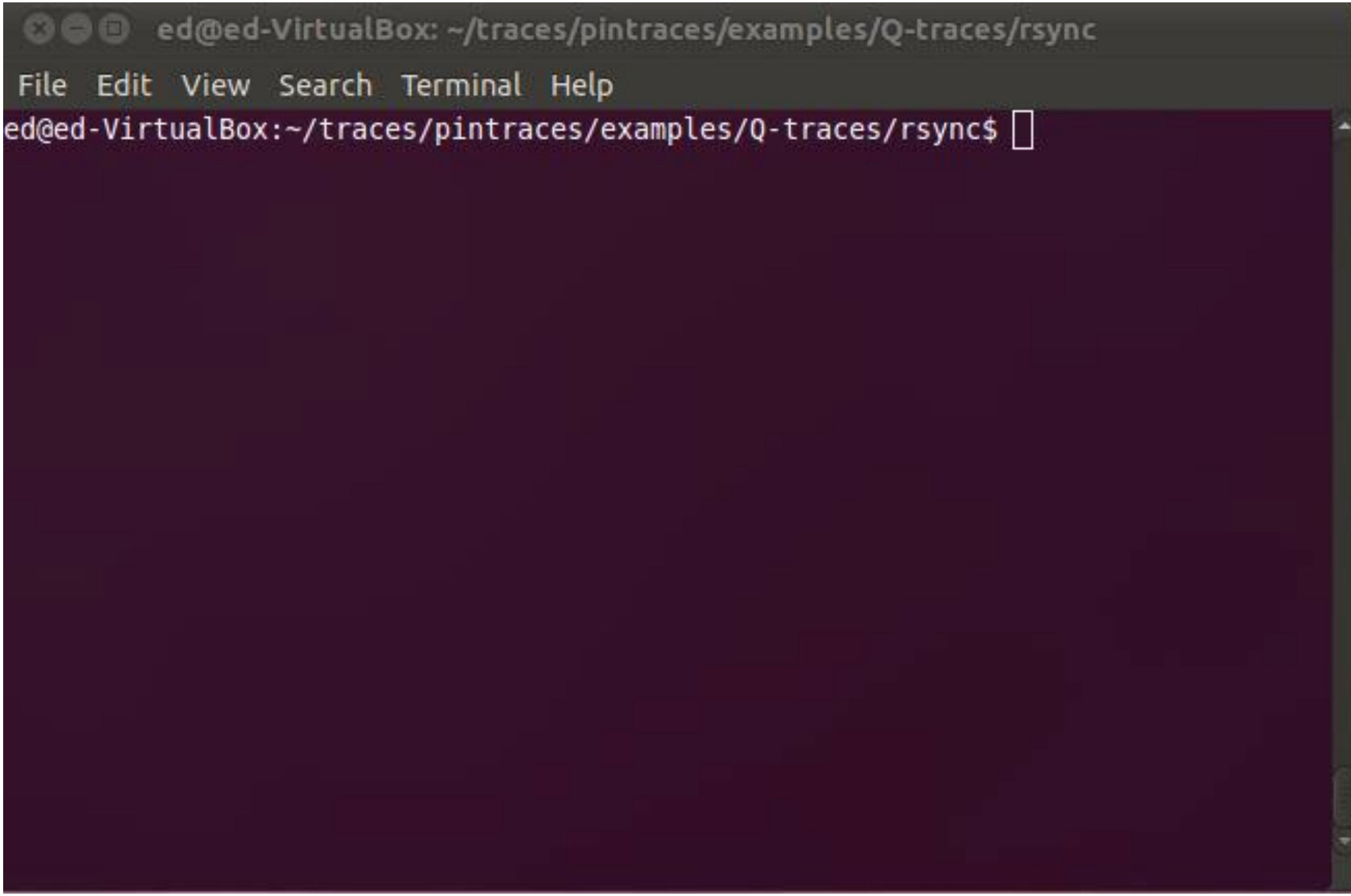
M[ESP+off2] = &gadget3

Exploit Constraints

Path Constraints

SMT → Exploit

Demo!


ed@ed-VirtualBox: ~/traces/pintraces/examples/Q-traces/rsync
File Edit View Search Terminal Help
ed@ed-VirtualBox:~/traces/pintraces/examples/Q-traces/rsync$

Overview

- Background: Defenses and Return Oriented Programming (ROP)
- Q: ROP + Hardening
  - Automatic ROP
  - Automatic Hardening
- **Evaluation**
- Limitations
- Conclusion

Evaluation Questions

1. Can Q harden exploits for real binary programs?

2. How much unrandomized code is sufficient to create ROP payloads?

Real Exploits

- Q was able to **automatically harden** nine exploits downloaded from exploit-db.com

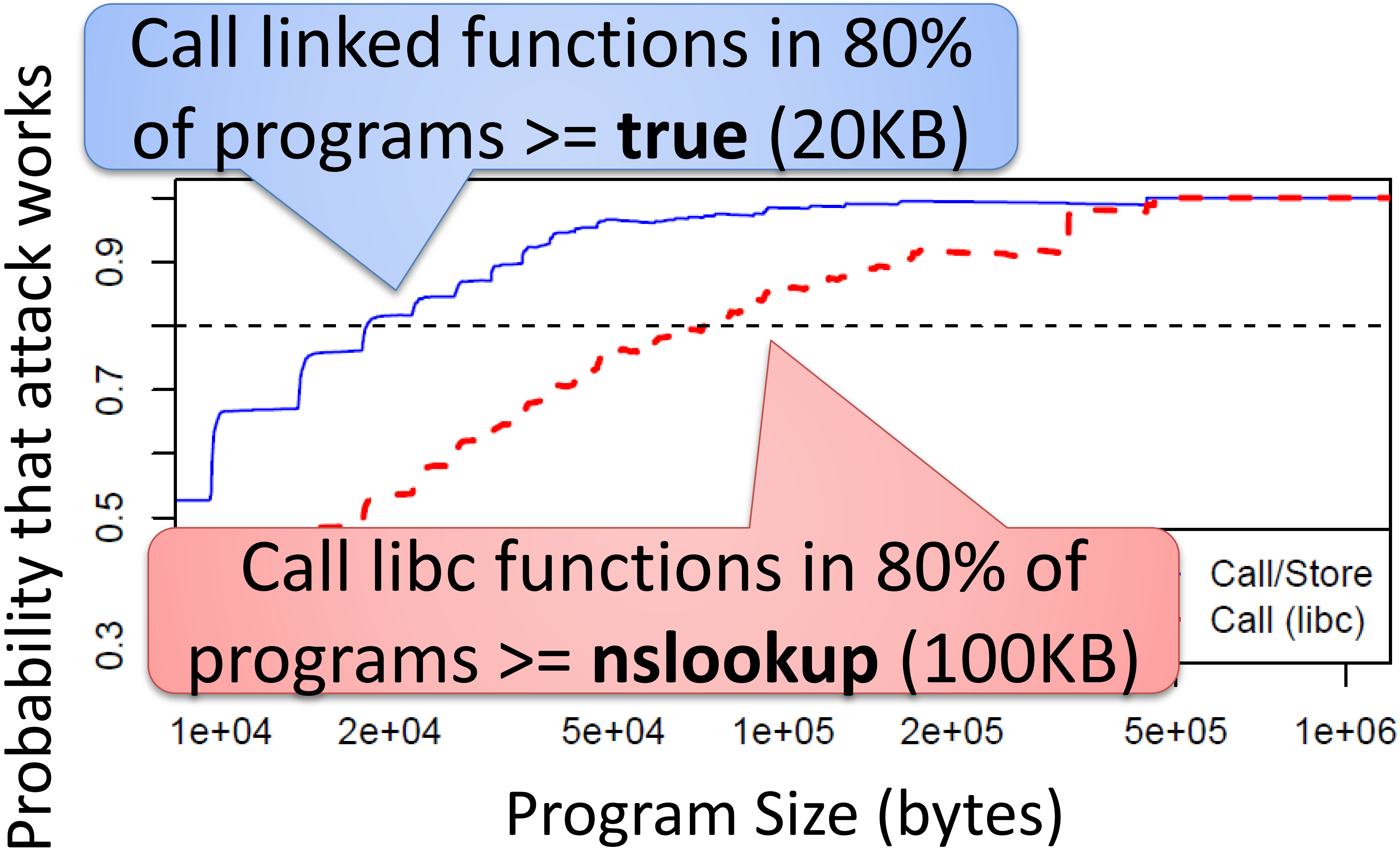
Name	Total Time	OS
Free CD to MP3 Converter	130s	Windows 7
Fatplayer	133s	Windows 7
A-PDF Converter	378s	Windows 7
A-PDF Converter (SEH exploit)	357s	Windows 7
MP3 CD Converter Pro	158s	Windows 7
rsync	65s	Linux
opendchub	225s	Linux
gv	237s	Linux
Proftpd	44s	Linux

ROP Probability

- Given program size, what is the probability Q can create a payload?
  - Measure over all programs in /usr/bin

- Depends on target computation
  - Call functions statically or dynamically linked by the program (blue on next slide)
  - Call any function in libc (red; harder)
    - system, execv, connect, mprotect, …

ROP Probability



Overview

- Background: Defenses and Return Oriented Programming (ROP)
- Q: ROP + Hardening
  - Automatic ROP
  - Automatic Hardening
- Evaluation
- **Limitations**
- Conclusion

Limitations

- Single path (trace-based) analysis
  - restrictive; prevents finding exploits
- Q's gadgets types are not Turing-complete
  - Calling system("/bin/sh") or mprotect() usually enough
  - Comparison with related work
- Q cannot find conditional gadgets
  - Potential automation of interesting work on ROP without Returns **[CDSSW10]**

Overview

- Background: Defenses and Return Oriented Programming (ROP)
- Q: ROP + Hardening
  - Automatic ROP
  - Automatic Hardening
- Evaluation
- Limitations
- **Conclusion**

Conclusion

- We built Q, a system that **automatically hardens exploits** to bypass defenses
  - **Challenge:** Reusing small amounts of code

- Q **automatically hardened nine** real exploits found in the wild against latest OS defenses

- **Takeaway:** Unrandomized code is dangerous
  - 20KB makes DEP+ASLR ineffective

Thanks! ☺

- Questions?

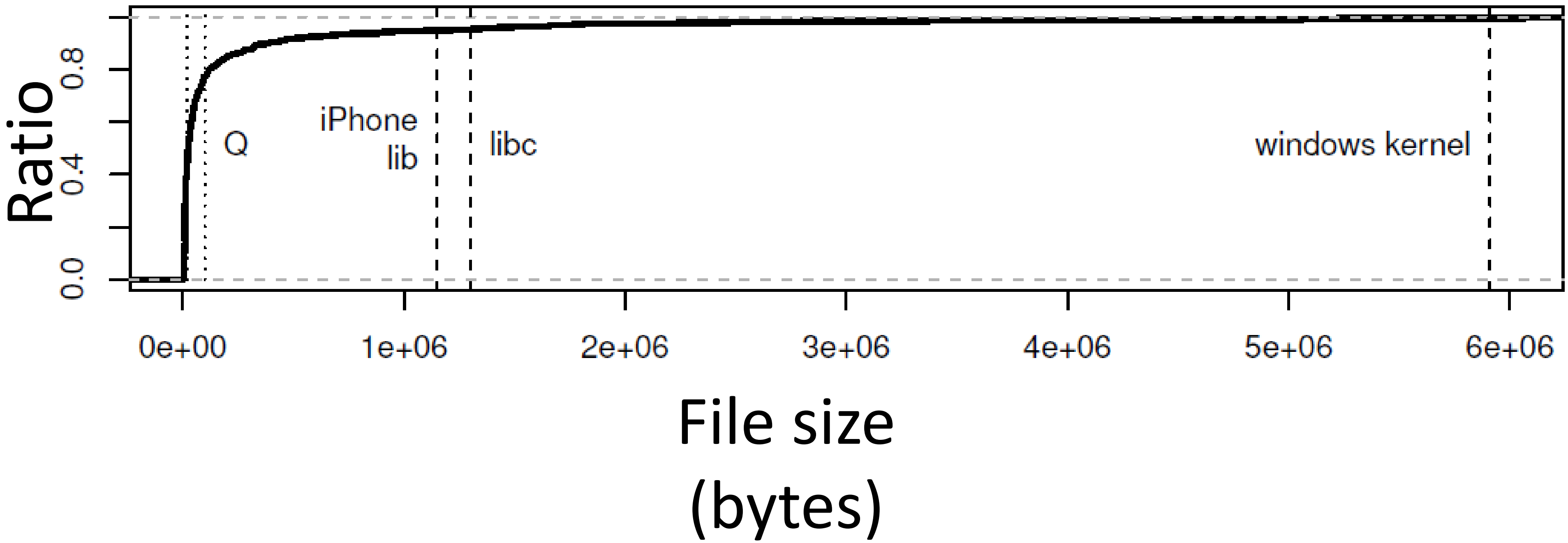
- Check out some of the gadgets Q can find at http://plaid.cylab.cmu.edu:8080/~ed/gadgets

Edward J. Schwartz

edmcman@cmu.edu

http://www.ece.cmu.edu/~ejschwar

Sizes of Gadget Sources



Types of Gadgets