Systems Security

PROFESSIONAL EDUCATION

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Hardware for Security

Howard Shrobe

Principal Research Scientist, Director Security@CSAIL

Computer Science and Artificial Intelligence Laboratory (CSAIL)

Massachusetts Institute of Technology

Overview

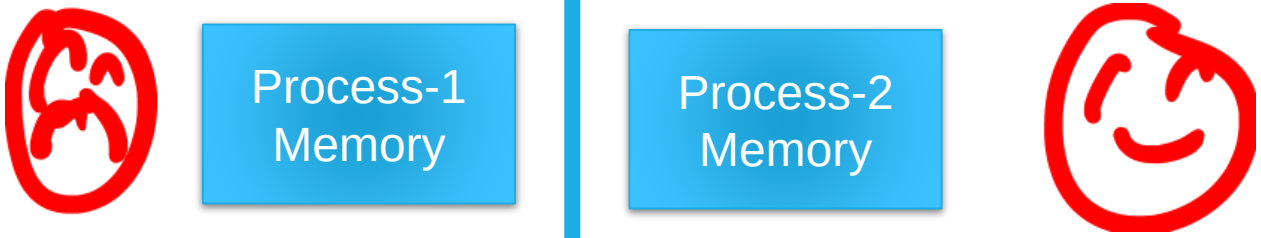
Why should we think about hardware-based protection?

- Current computers already have hardware-based protections
- New features are in the pipeline
- There are important properties that can be enforced more systematically in hardware
- There are important properties that can only be done efficiently in hardware
- Hardware can enforce the intended semantics of your software

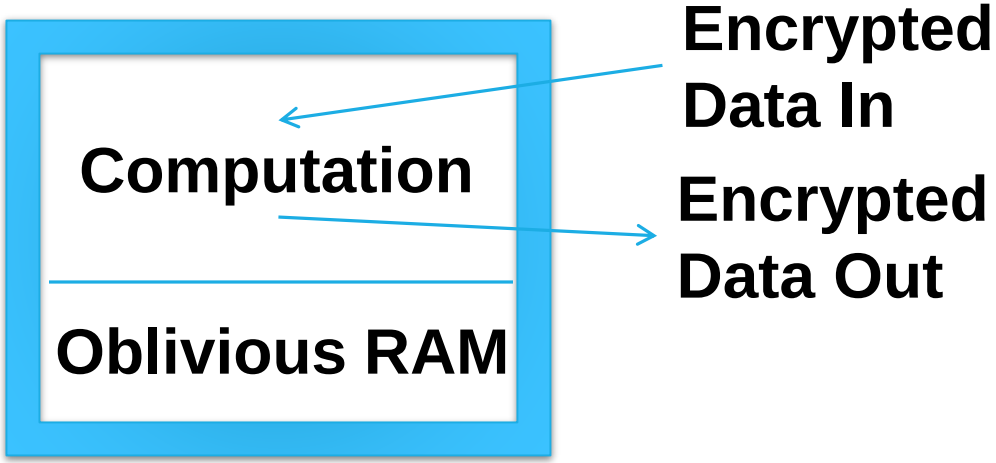
Examples of hardware-based protection

Process Isolation

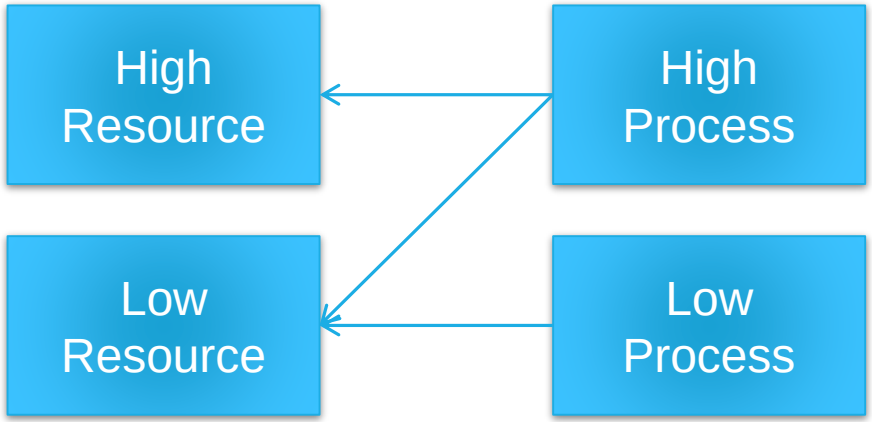
Any Processor with Virtual Memory



Data Protection - Ascend



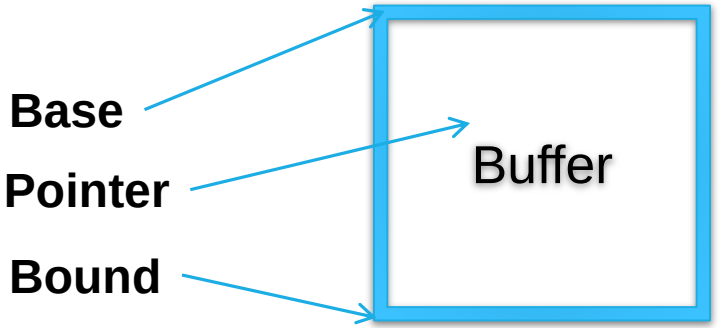
Enforcement of Access Policies



Policy:

- **High processes can access anything**
- **Low processes can only access low data**

Memory Safety Intel-MPX



Agenda

1. Learning from the past: Multics
2. Examples of what can go wrong
3. Tagged architectures
    1. Memory safety
    2. Type safety
    3. Information flow
    4. “Zero Kernel”
4. Capability architectures

Historically Interesting Machines

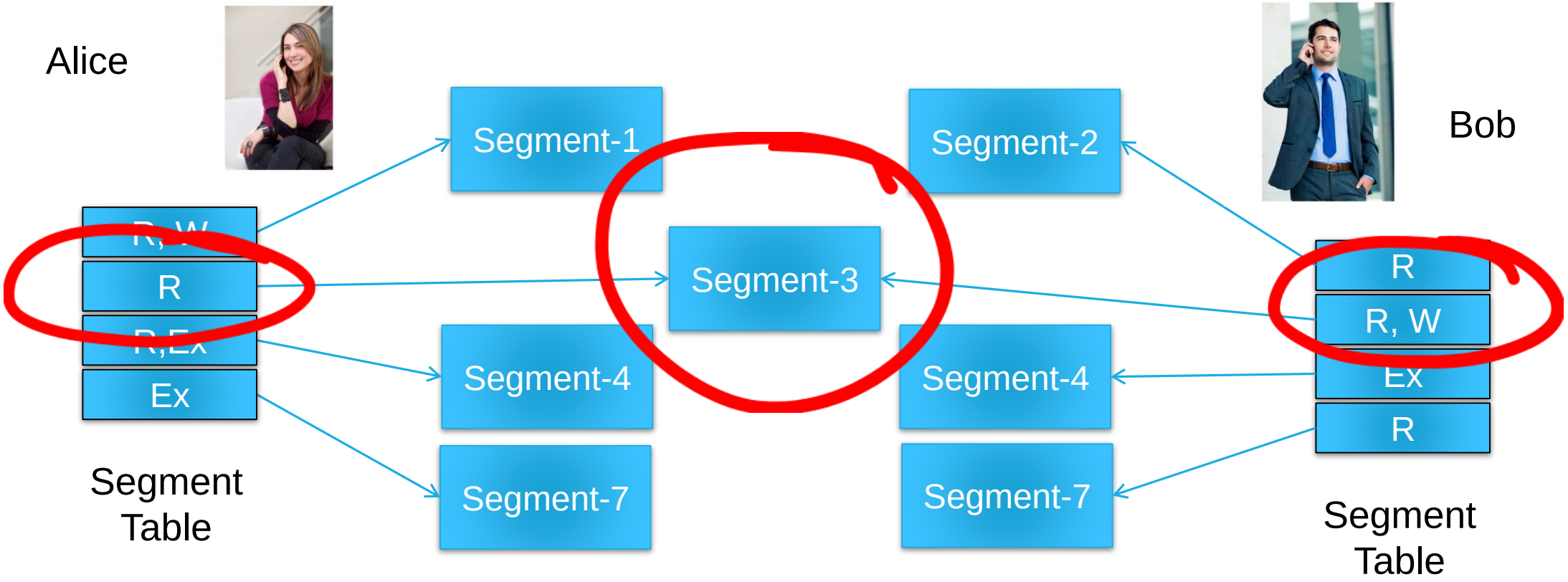
- Multics: 1964 (last machine decommissioned in 2000)
  - Segments, Rings
- Burroughs B5500/6500: 1967
  - Data type tags
- Cambridge CAP Machine: 1970 – 1977
  - Capability machine
- MIT Lisp Machine: 1973 (last machine still running)
  - Data type tags, bounds checking
- IBM System 38: 1978 (now IBM System I)
  - Capability Machine – current product

Multics: An early secure hardware base

The Multics Model: Controlled Sharing

- **Multics** was an early MIT computer architecture project (started in 1964)
- Multics was conceived as a *computer utility,* supporting many users
- Users needed to share resources to support collaboration
- Users also required protection of their private data

Multics: Segmented Memory & Permissions



- Memory addresses are two dimensional: Segment #, Address within segment
- Segments are composed of virtual memory pages
- Each user has a segment table, which points to a page table

Multics: Segments, Files, Access Lists

Alice

Alice makes a request to map File-1 into her Segment-1

File-1's access control list is checked, mapping succeeds

R, W
R
R,Ex
Ex

Segment Table

Segment-1
Segment-3
Segment-4
Segment-7

File-1
File-2
File-3
File-4
File-5
File-6

File System

Alice R,W
Bob R
Charley R

ACL

Alice R,W
Bob R
Charley R

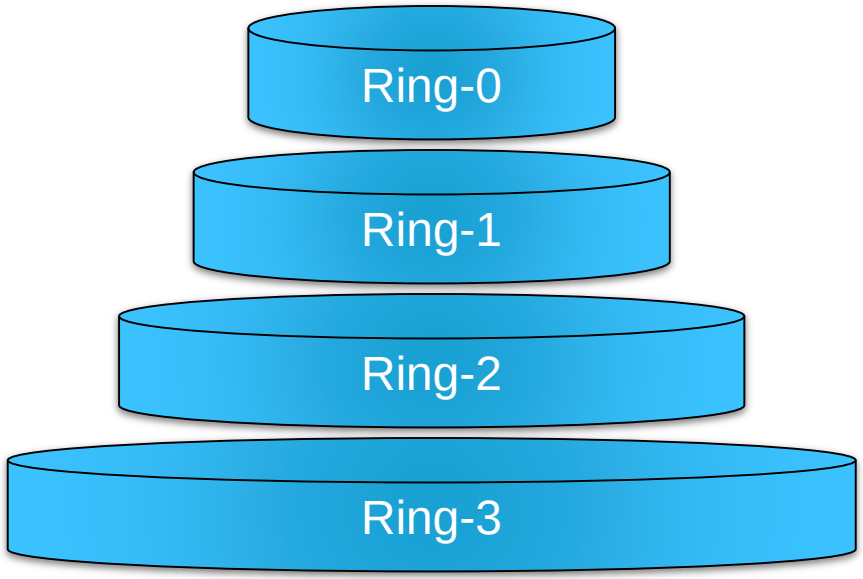
ACL

Alice makes a request to map File-6 into her Segment-7

File-6's access control list is checked, mapping fails

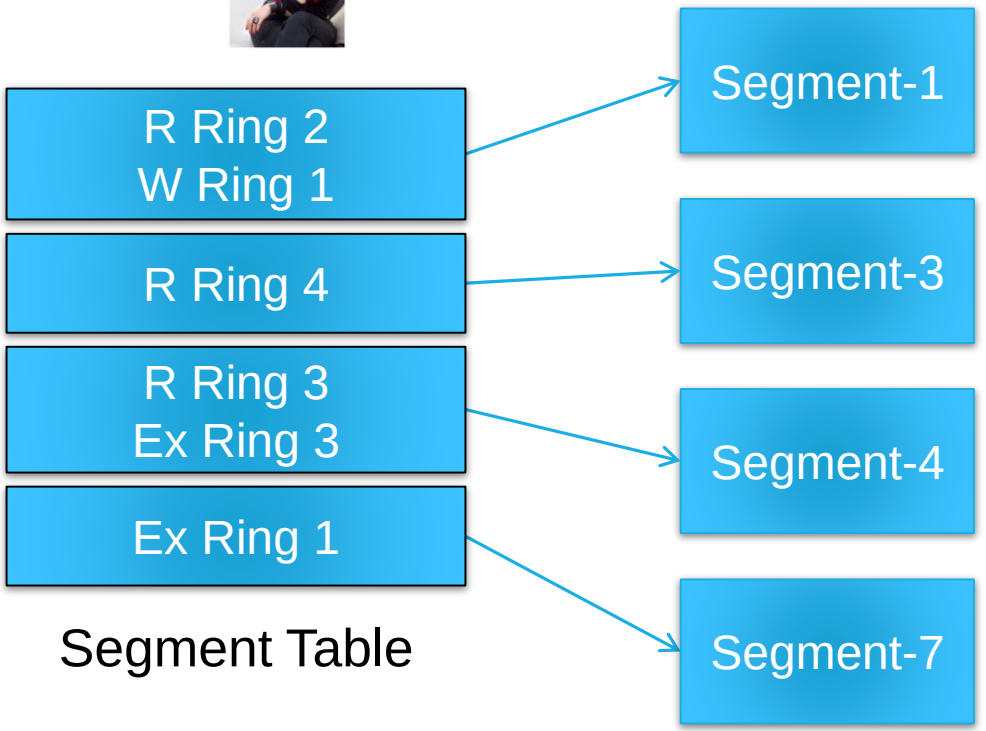
Multics: Rings of Protection

- Each ring is strictly more privileged than the next
- Ring-0 is the most privileged



- Alice can read Segment-1 if she's executing in Ring 0,1 or 2
- Alice can write Segment-1 but only if she's executing in Ring 0 or 1

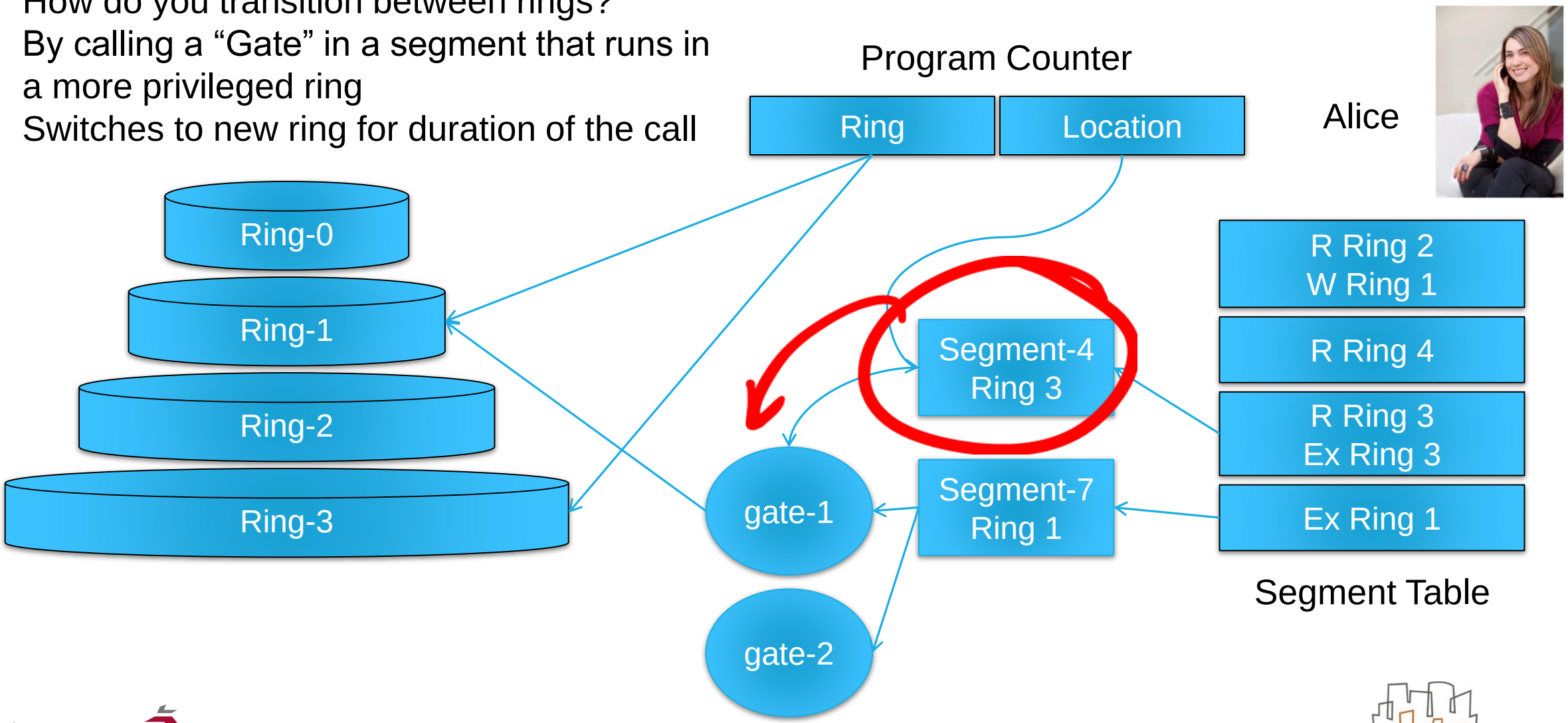
Alice



Segment Table

Multics: Gates for changing privilege levels

- How do you transition between rings?
- By calling a “Gate” in a segment that runs in a more privileged ring
- Switches to new ring for duration of the call



Key Principles

1. Complete Mediation
2. Separation of Privilege
3. Least Privilege
4. Economy of Mechanism
5. Acceptability & Productivity

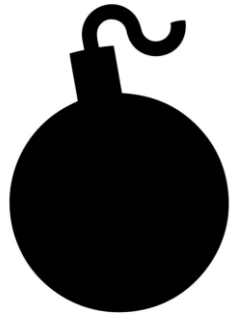
Saltzer & Schroeder: The Protection of Information in Computer Systems

Revised version in *Communications of the ACM* 17, 7 (July 1974)

The Multics Legacy

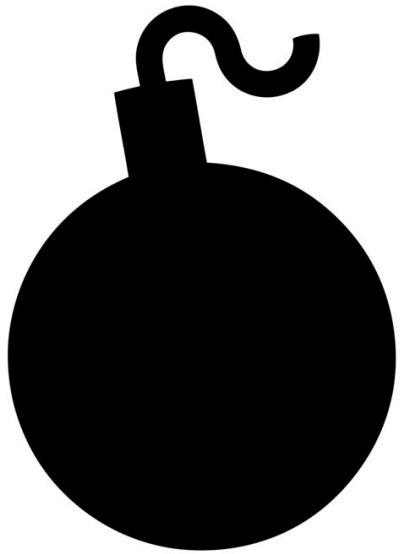
- For its day the Multics hardware was too expensive & used too many resources
- Unix was created as cheaper, simpler system, but with fewer guarantees
  - No segments, permission on pages
  - User & Kernel modes; mode switch is expensive
  - Kernel mode is totally privileged
  - C programming language for efficiency
- We've continued that trend ever since

What do attackers do?



What can go wrong. Examples of attacks:

- Memory corruption
- Data disclosures
- Code injection
- Control flow diversion
- Return-oriented programming



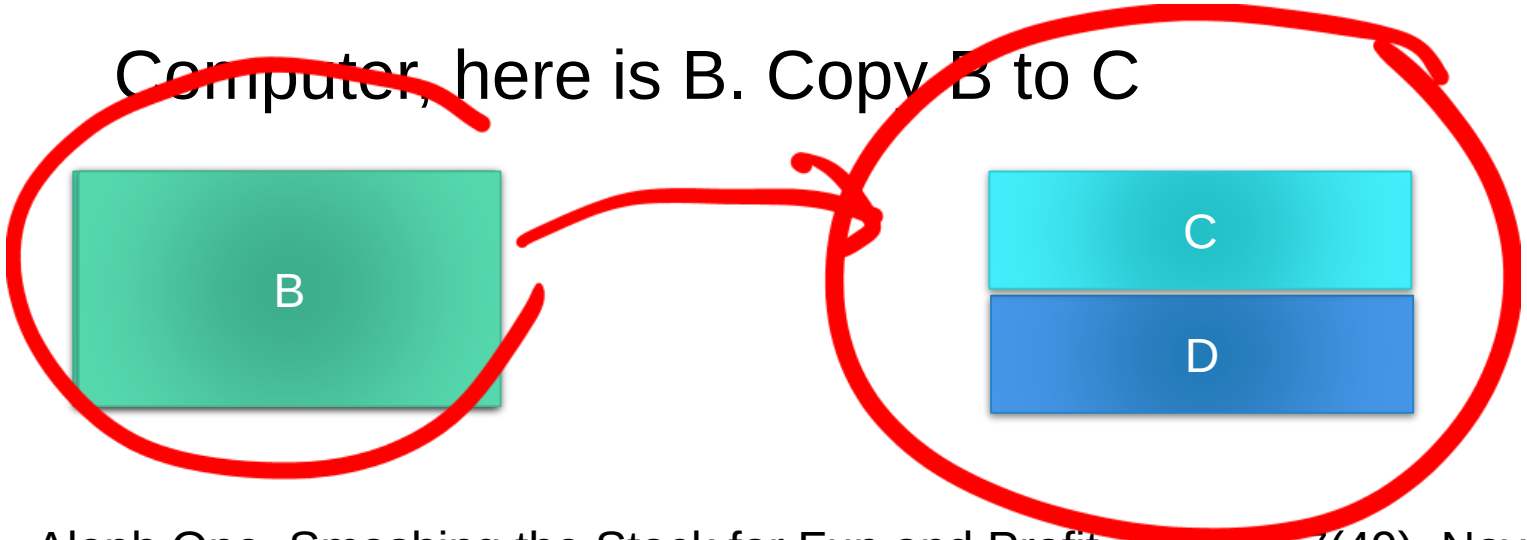
Memory corruption exploits

Suppose your program is copying data from the network to an array:

Computer, here is A. Copy A to C

No problem.

Computer, here is B. Copy B to C

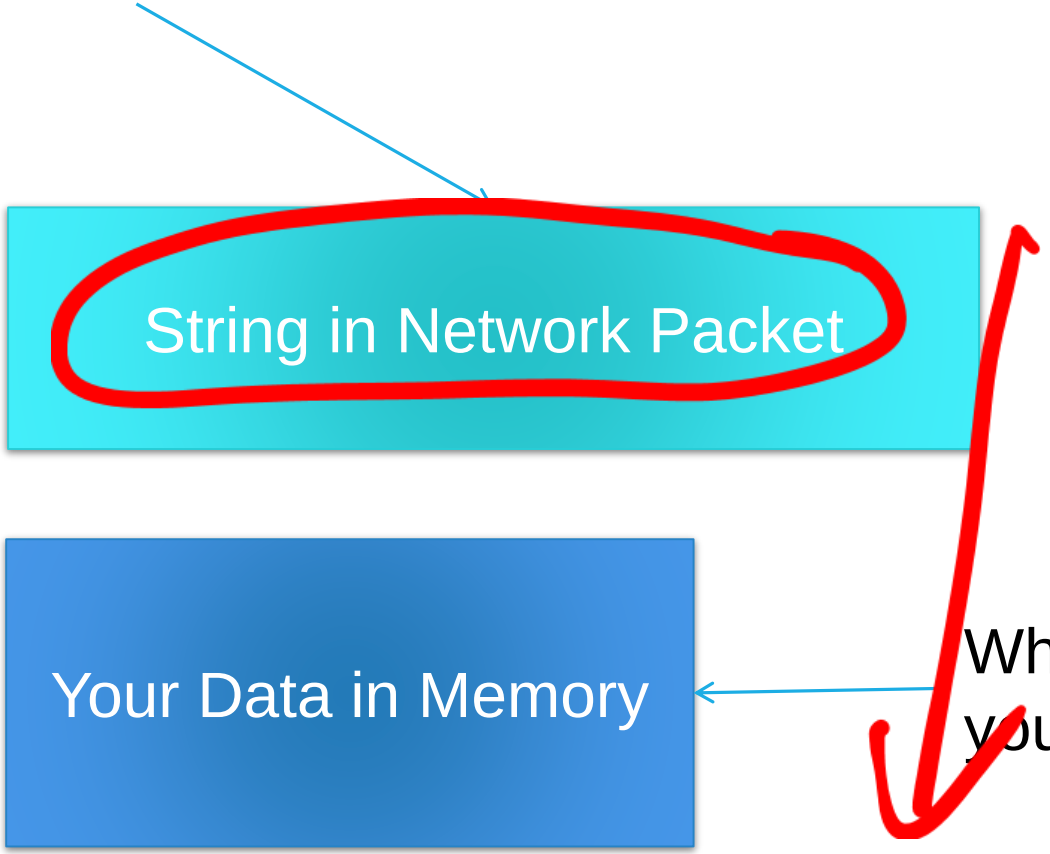


- If bounds aren't checked, there is a big problem, D is over-written
- Bounds that aren't represented can't be checked

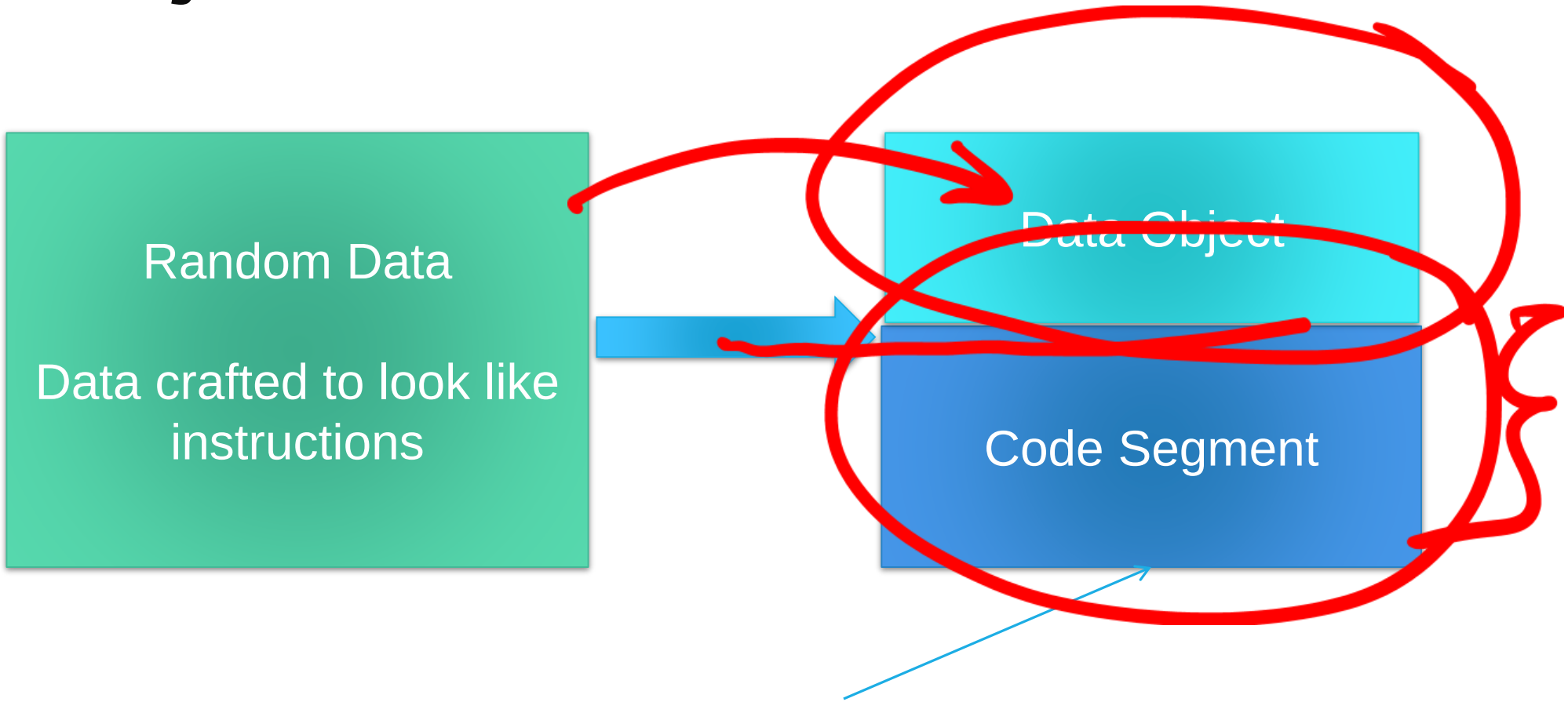
Aleph One. Smashing the Stack for Fun and Profit. Phrack, 7(49), November 1996.

Heart bleed memory disclosure attack

Network protocol request:
Echo 100,000 characters of this string

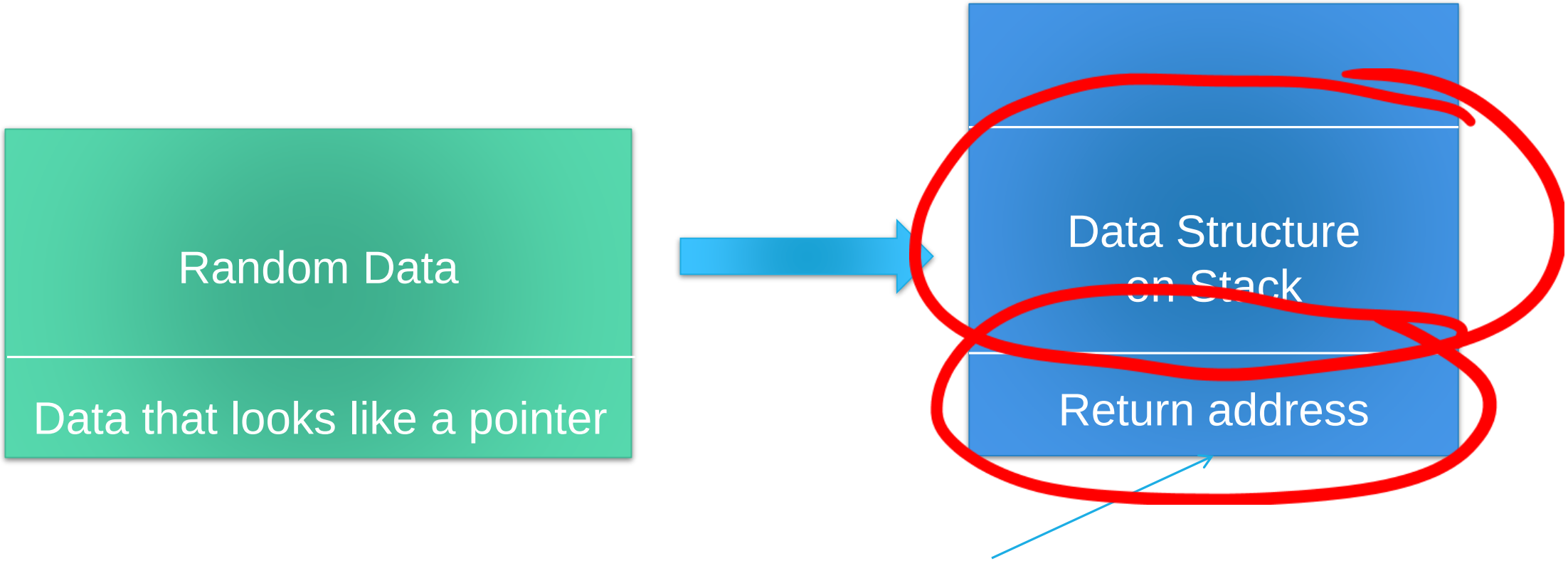


Code injection



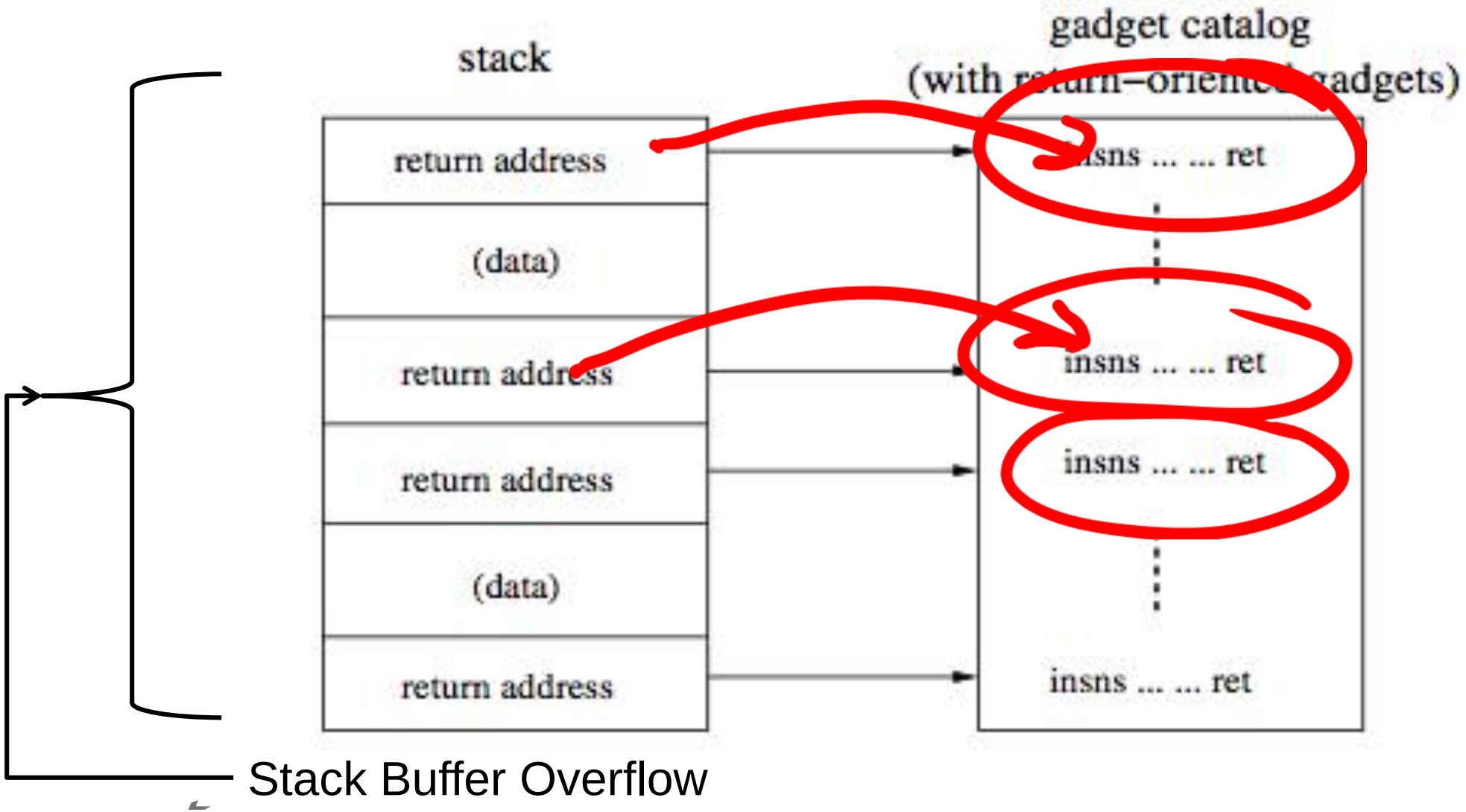
When someone calls this routine, the attacker's instructions get executed

Pointer over-write



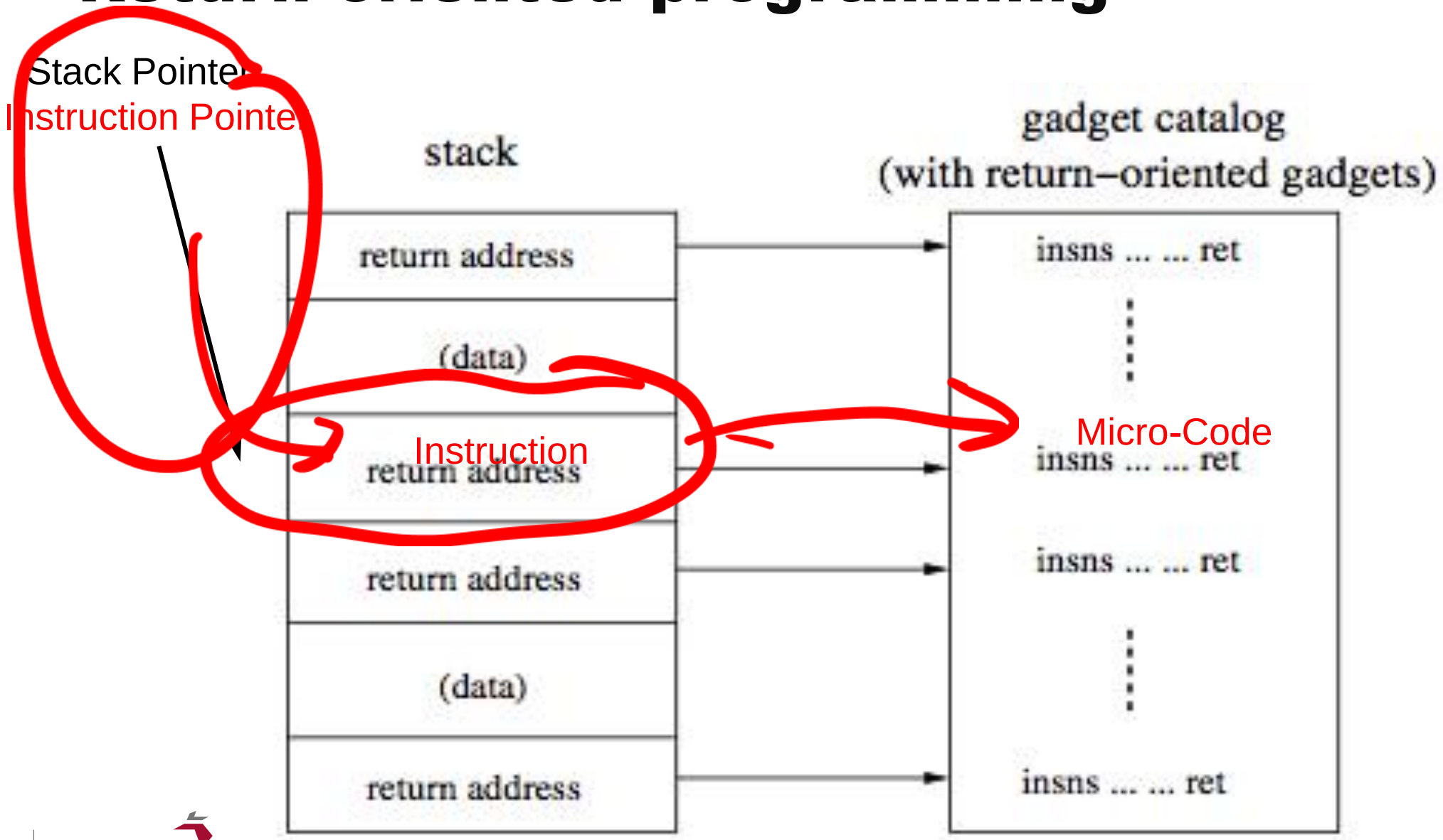
Over-writing the return address diverts execution to a place of the attacker's choosing

Return-oriented programming



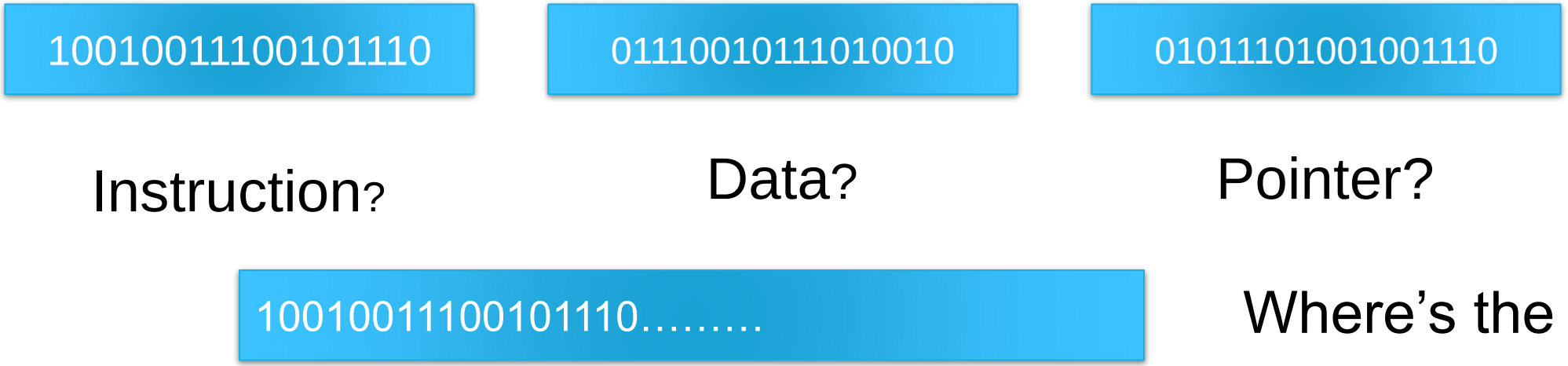
Stack Buffer Overflow

Return-oriented programming



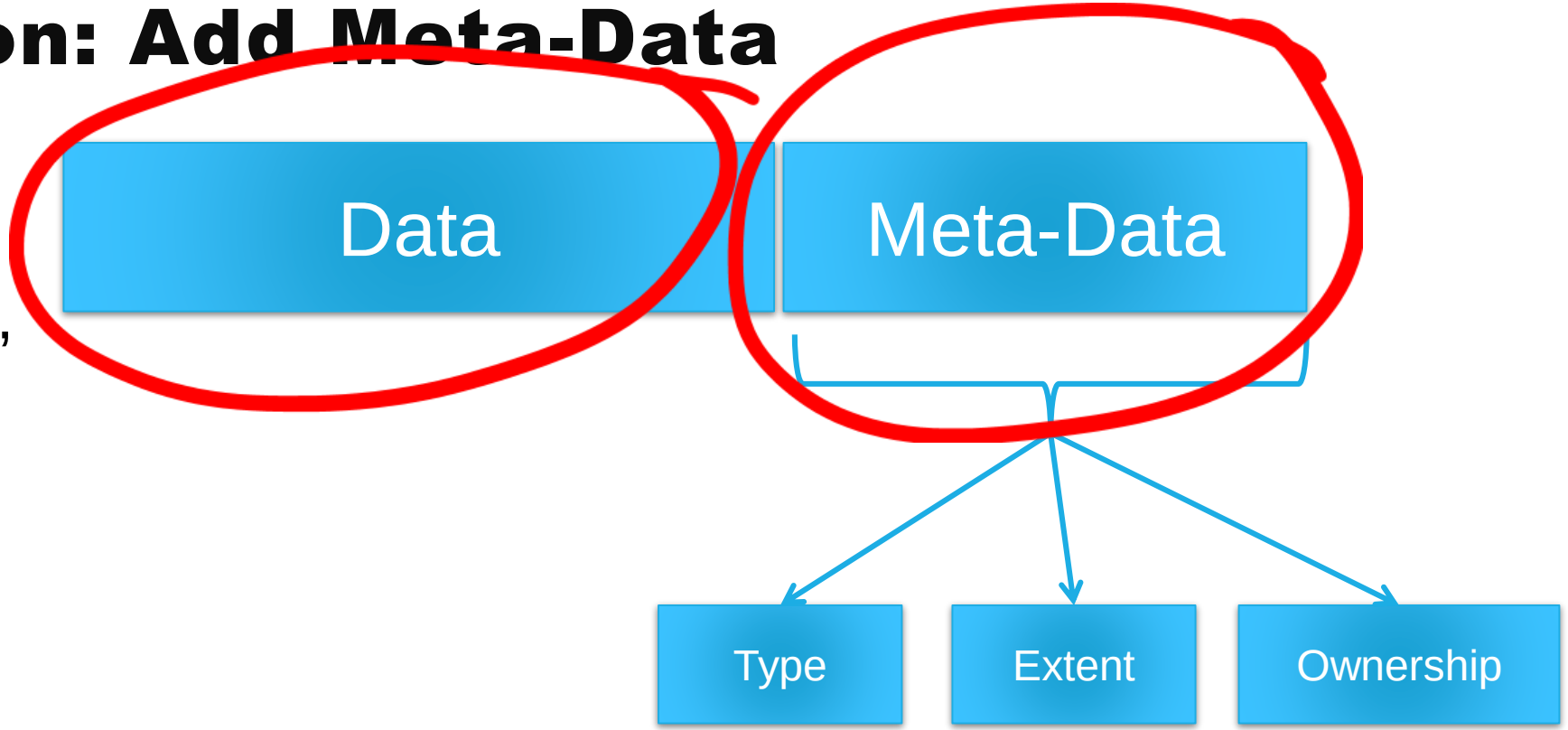
The Underlying Problem:

- Computers don't make important semantic distinctions
- At runtime, there's nothing but "Raw Seething Bits"



The Solution: Add Meta-Data

- Type
  - Instruction
  - Immediate Data,
  - Pointer
- Extent
  - Base
  - Bounds
- Ownership
  - Compartment
  - Access rights



Type Tagged Architectures

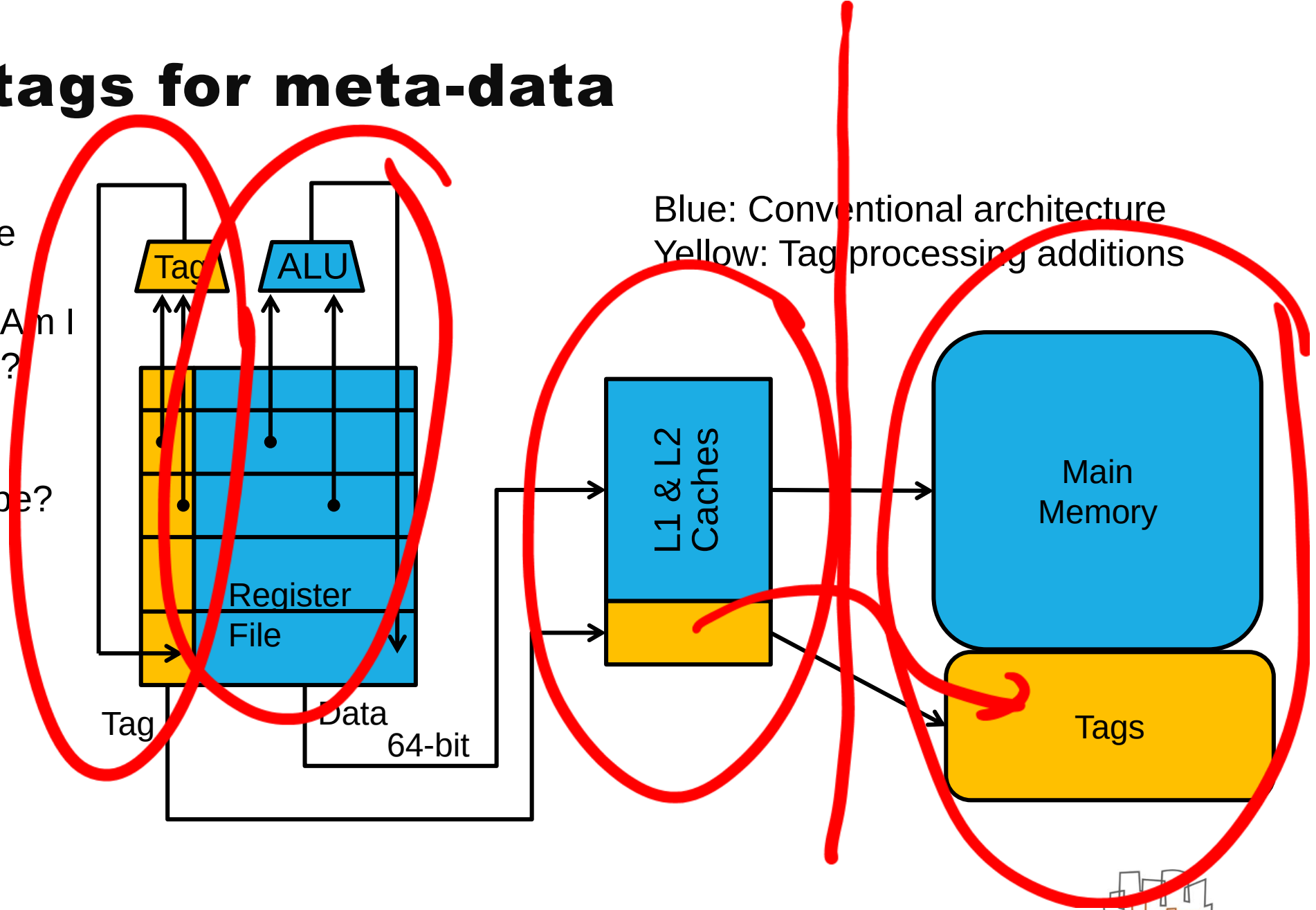
The Solution: Enforce policies

- Type
  - Only instructions can be executed
  - Only pointers can be followed
  - The argument to a call must be a function pointer
- Extent
  - You can’t increment a pointer outside its bounds
  - You can’t reference outside a pointer’s bounds
- Ownership
  - When you combine data the result maintains compartment
  - Only designated “principles” can access data

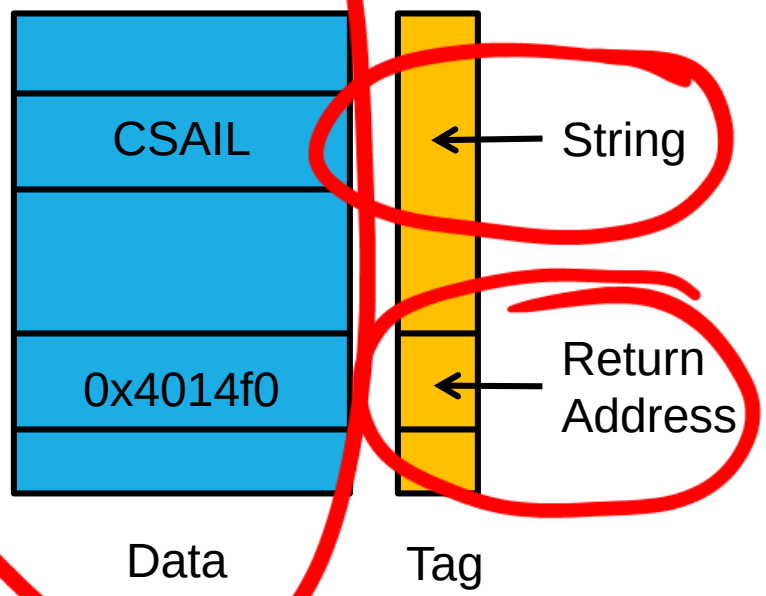
Adding tags for meta-data

Tag processor checks the meta-data tag:

- Does it make sense? Am I trying to add 2 strings?
- Is it allowed? Can I access data of this type?
- What's the tag of the result?



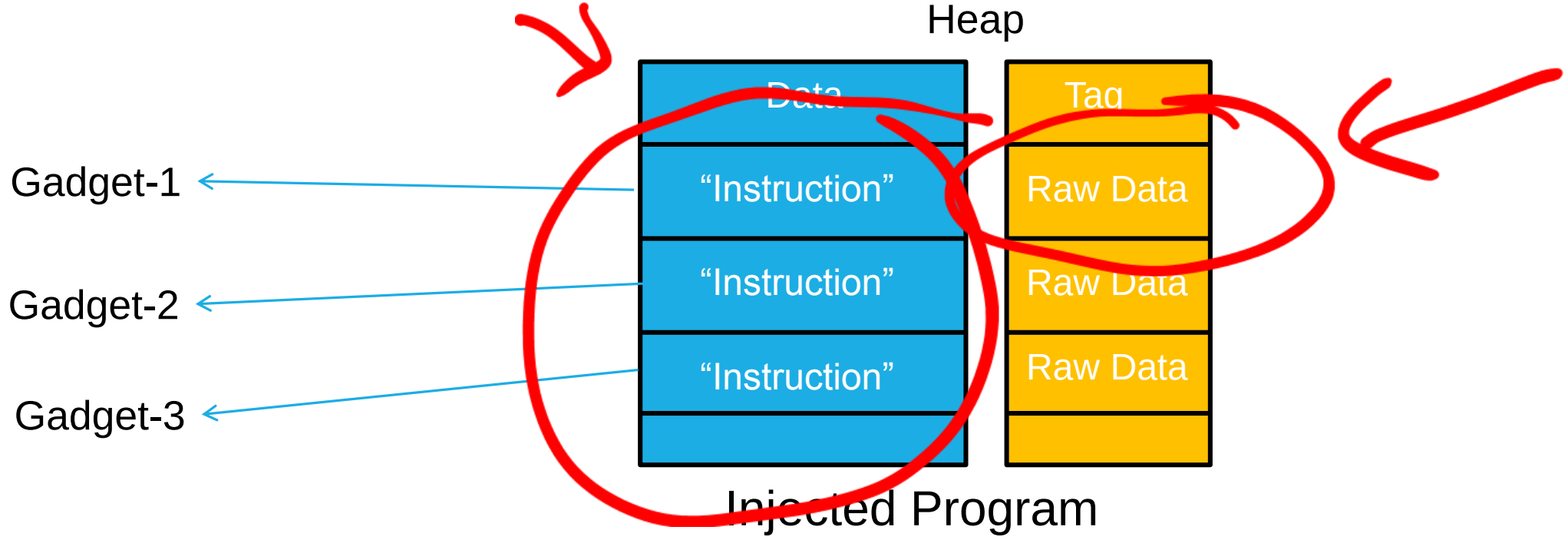
Data type tagging



Data Types	Pointer Types	Instruction Types
Raw data	Function Pointer	Subroutine Entry
Integer	Return Address	Return point
String	Data Pointer	Normal Instruction

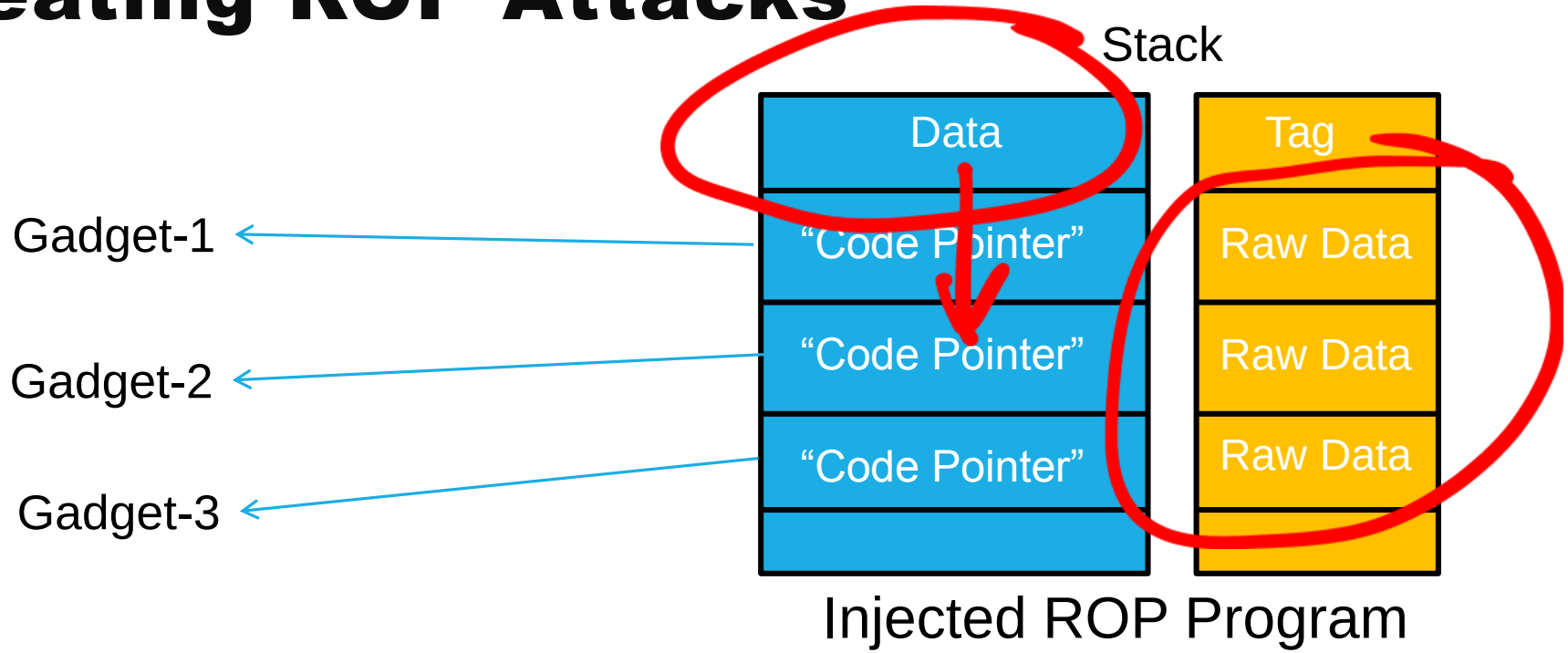
- **Store type information in protected metadata tag**
- **Enforce data-type oriented policies on each cycle**

Defeating Code Injection Attacks



- Although purported “instructions” can be pushed into the heap, they will have the tag of “Raw Data”
- **Attacker cannot forge instructions and hijack control flow**

Defeating ROP Attacks

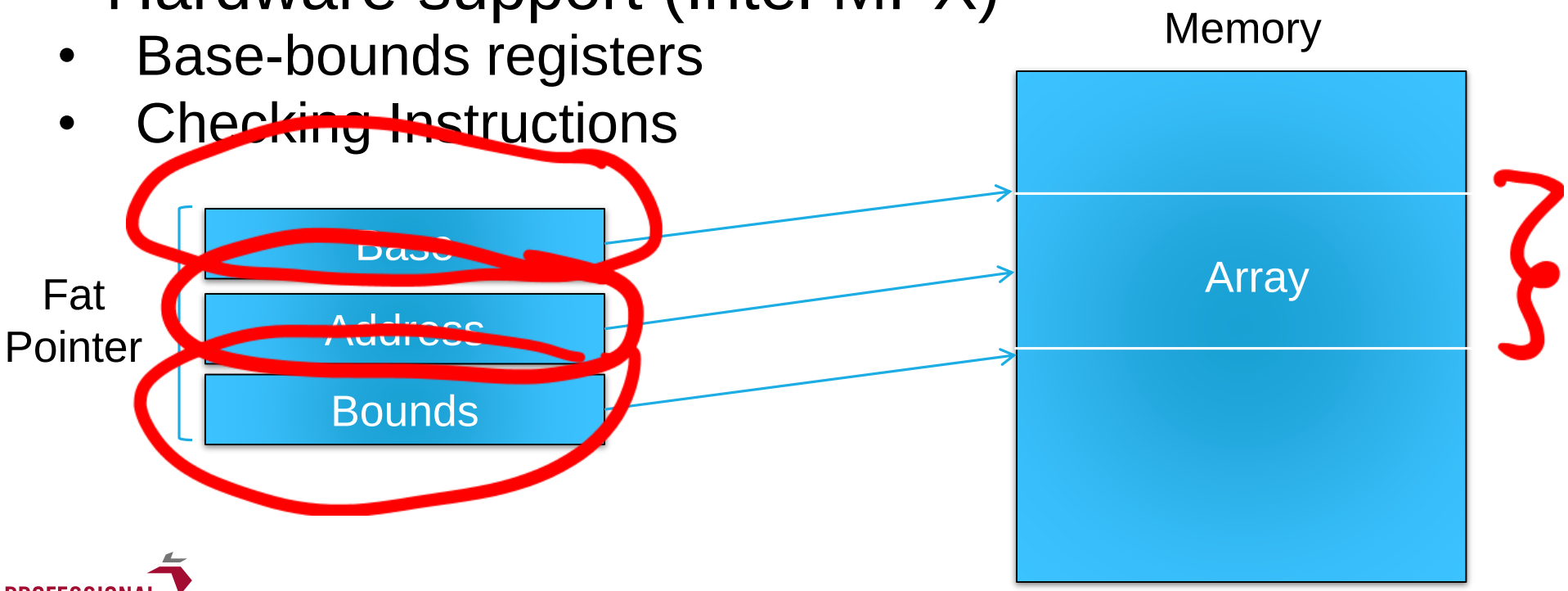


- Although purported "code pointers" can be pushed onto the stack, they will have the tag of "Raw Data"
- **Attacker cannot forge code pointers and hijack control flow**

Memory Safe Architectures

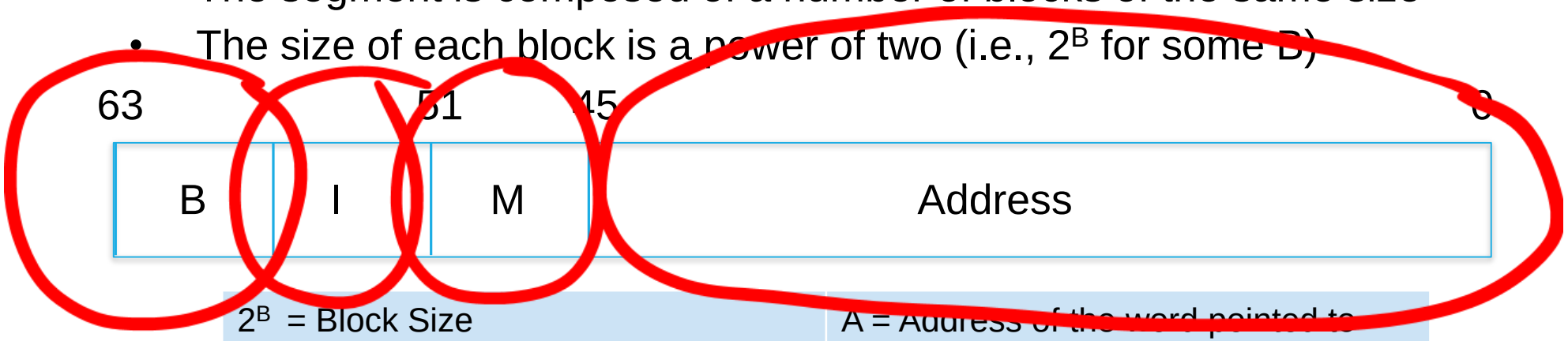
“Fat Pointers” for memory safety

- Every pointer has 2 other words associated with it for base and bounds
  - 100% runtime overhead when done in software
  - Lots of extra loads, stores and checks
- Hardware support (Intel MPX)
  - Base-bounds registers
  - Checking Instructions



“Low Fat Pointers”

- “SAFE” processor in DARPA CRASH program
- Trades allocation constraints for compactness
  - The segment is composed of a number of blocks of the same size
  - The size of each block is a power of two (i.e., 2^B for some B)

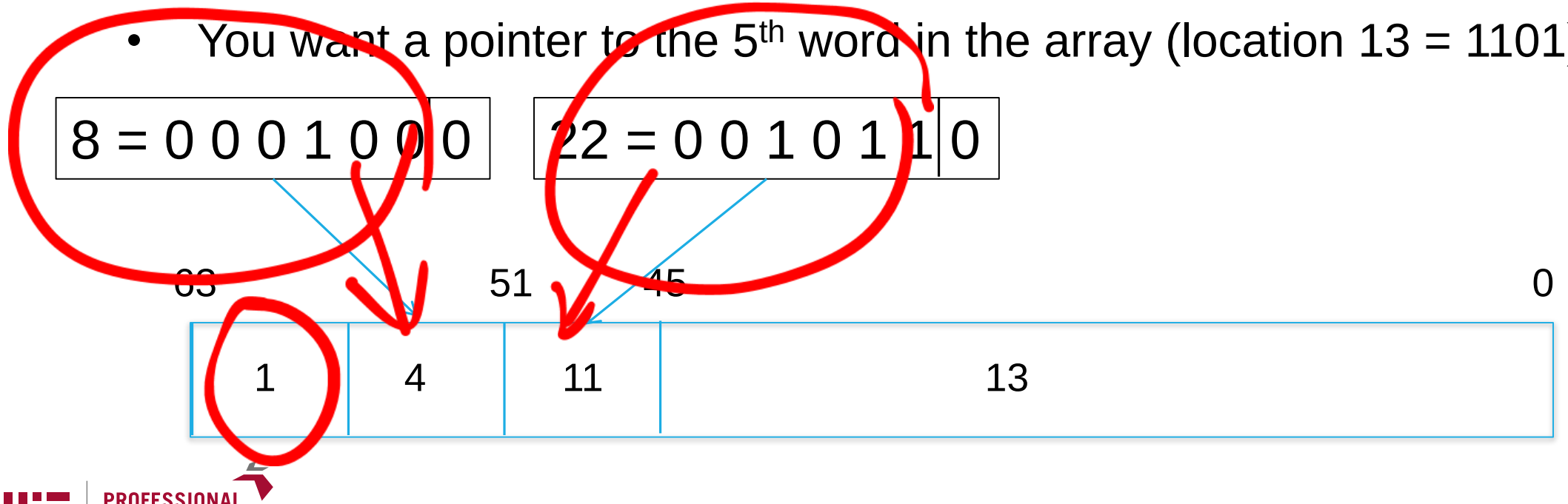


2^B = Block Size	A = Address of the word pointed to
I = The 6 bits above low order B bits of the Lower Bound	M =The 6 bits above the lower order B bits of the Upper Bound

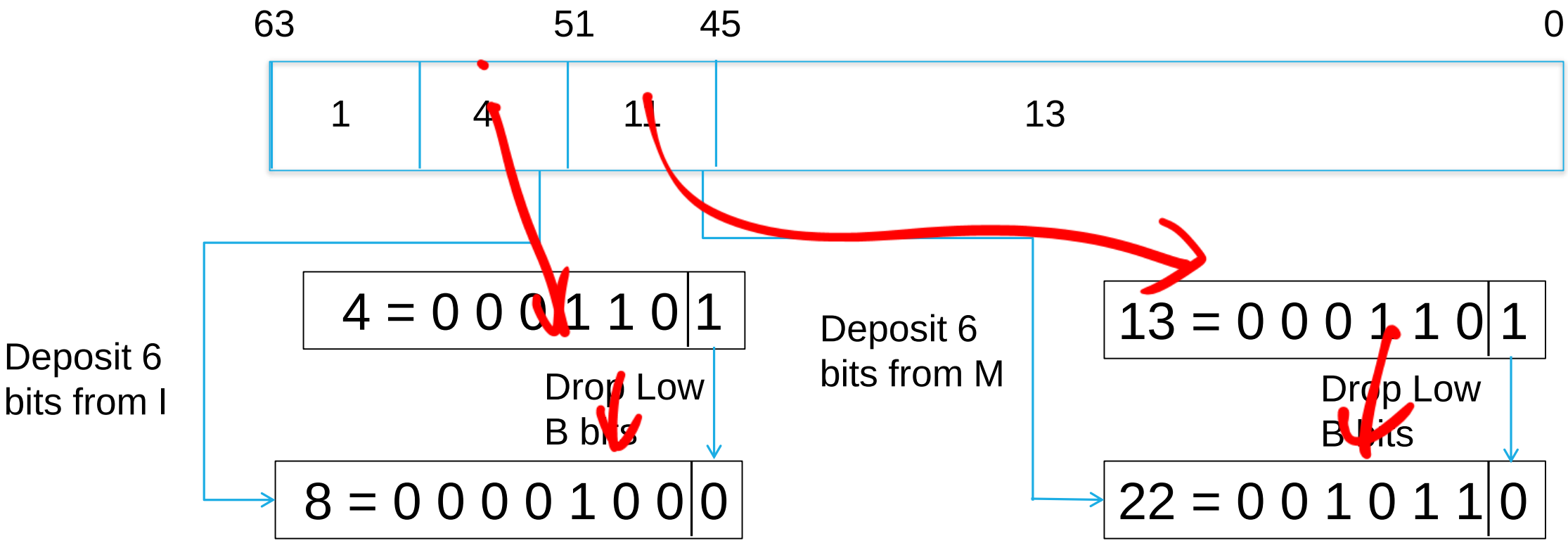
Low-Fat Pointers: Compact Encoding and Efficient Gate-Level Implementation of Fat Pointers …
20th ACM Conference on Computer and Communications Security, November 6, 2013, Berlin, Germany.

Low Fat Pointer Example

- Suppose you want an array of 13 words
- The next available space is at location 7
- Base = 8 (aligned to 2) 01000 in binary
- You need 14 words (7 blocks of 2 words),
- Bound = 22 10110 in binary
- You want a pointer to the 5th word in the array (location 13 = 1101)



Recovering base and bound



* It's actually a little more complicated in some cases, but the hardware can handle it

“Information Flow”

Information flow

- Keeping data from getting to where it shouldn't:
- Every word has a "label"
  - A set of "Principals", the owners of the data
  - The "compartment" the data is classified in
- As data is combined by the ALU, labels are combined by the Tag Management Unit
- The processor is always running on behalf of some specific principal
- Rules specify which principals can perform what operations on the data, given the labels

Example information flow policy

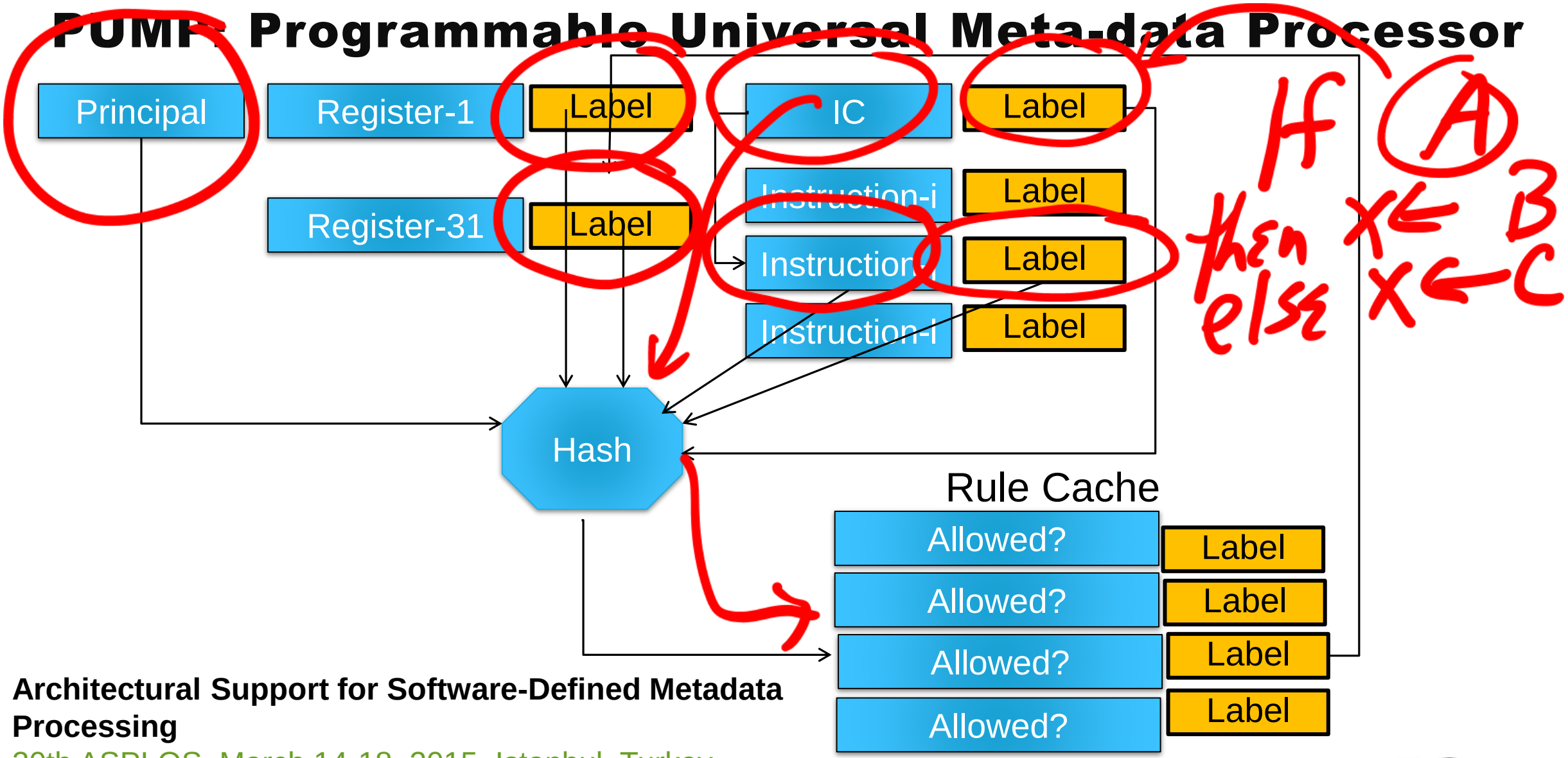
- Howie has a shared data compartment
  - Any friend of Howie can read this data
  - John is a friend of Howie
- John has a shared data compartment
  - John's friends can read data in that compartment
- John combines data from his shared data compartment with data from Howie's shared data compartment
  - This data can only be read by someone who is both a friend of Howie and a Friend of John

Information flow policy as rules

If the Principal is Howie
  the read-data is in Howie's private compartment
  the operation is Read
Then OK with Tag Howie's private compartment

If the Principal is John
  the first operand is in Howie's shared compartment
  the second operand is in John's shared compartment
  the operation is Add
Then OK with Tag John & Howie shared compartment

PUMP: Programmable Universal Meta-data Processor



Architectural Support for Software-Defined Metadata Processing

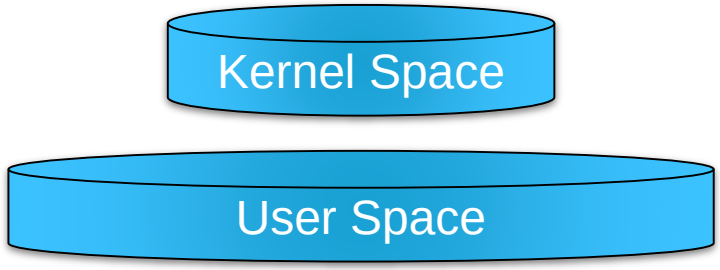
20th ASPLOS, March 14-18, 2015, Istanbul, Turkey.

Capability architectures

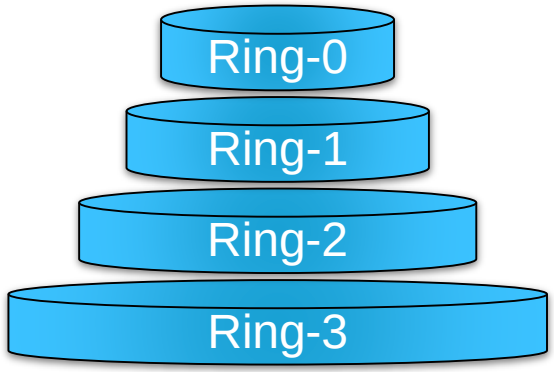
The problem with current architectures

- Address space and protection are identified
  - Context switching is very expensive
- We have, at best, a ring-like system in which the innermost ring (the kernel) is all privileged.
- This violates “least privilege”
  - Why should the scheduler be able to read my personal data?

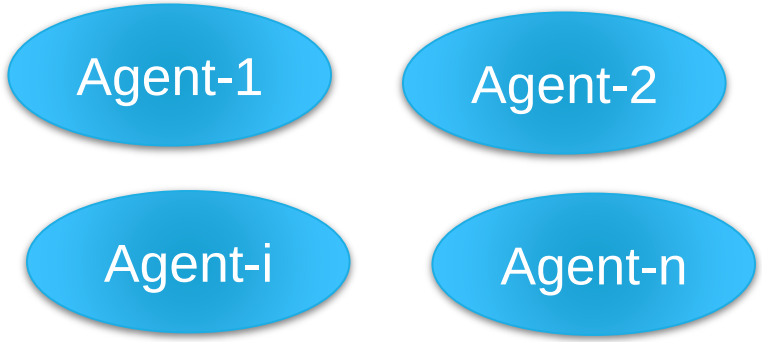
Unix/Linux
Older Hardware



Multics/Hypervisors
Current hardware



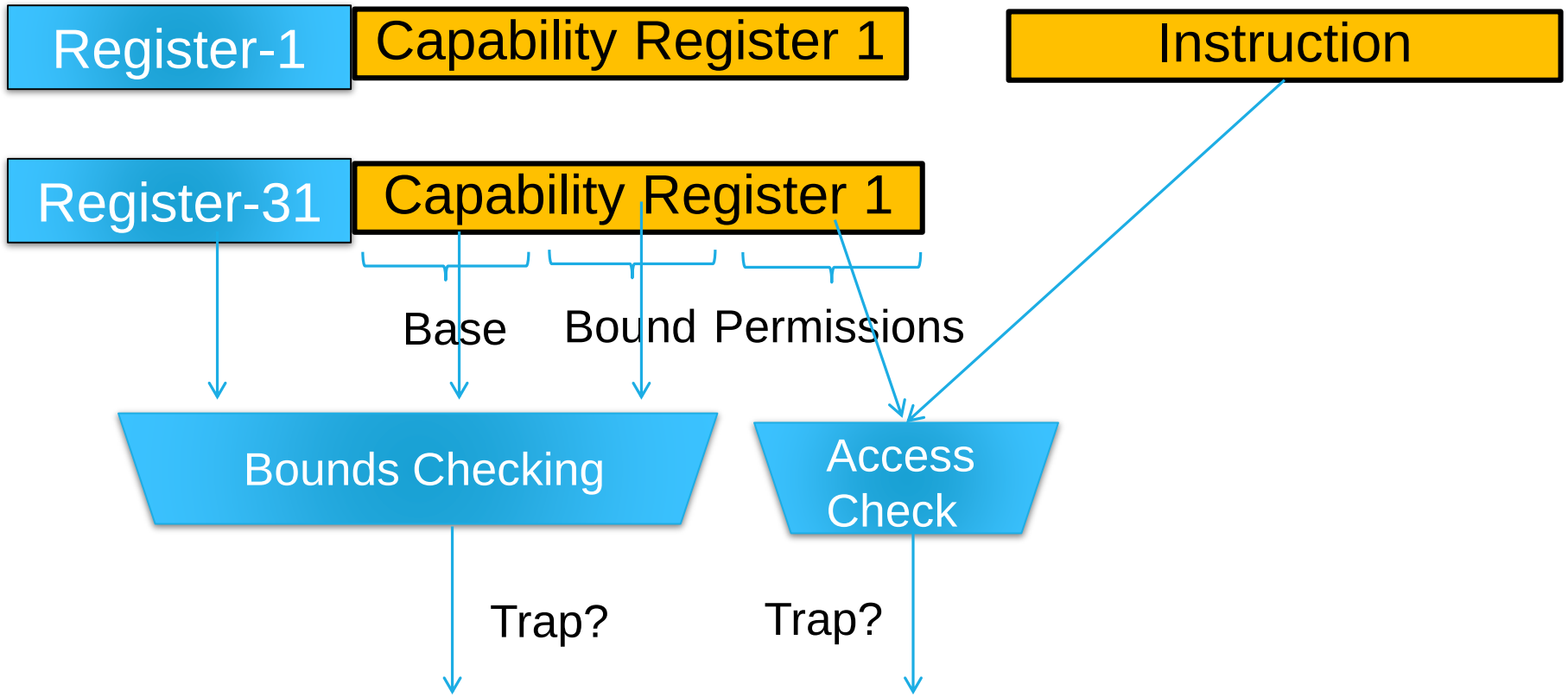
Communicating Peers
New Hardware



Capability Architectures

- A flat memory space
  - Process isolation through virtual memory isn't necessary
- No "raw pointers"
- A Capability is:
  - A pointer
  - Base and bounds
  - Access rights to the thing pointed at
  - Typing information
- You can only access things that you have a capability for

Capability hardware



Capability Architectures: Issues

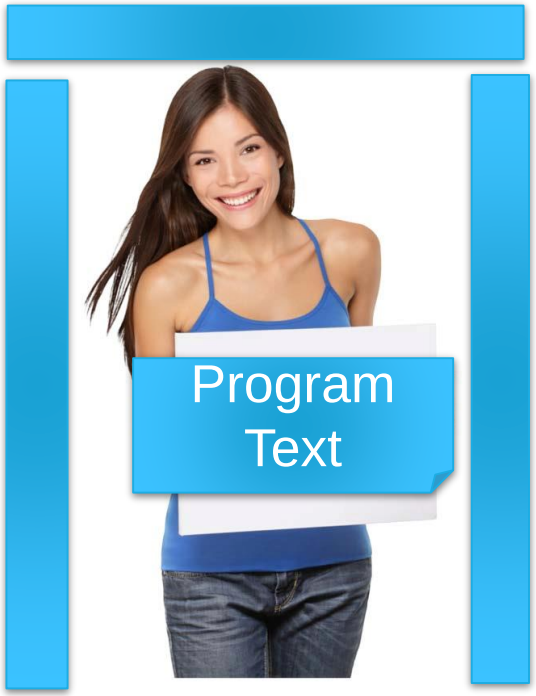
- Passing a capability is equivalent to granting privilege
  - Can lead to “confused deputy” problems
  - Revocation can be difficult
- What you want is to pass a “less capable” capability
  - Tighter bounds
  - More limited access rights
  - More specific type

A “Zero-Kernel” System

- Merge capability and information flow ideas
- Principals, Tagged Objects, Access Rules
- Gate calls:
  - Invoke a new routine with a changed principal
  - Principals are not strictly hierarchical
- The Kernel is broken up into many pieces
  - Each with its own principal and compartment
  - Controlled sharing within the kernel
  - Least privilege with respect to user data

Zero Kernel System: Building Blocks

Gate:
A Procedure Bound with a New Principal



Compartment:
A collection of Data with common access rights

Access Rule:
Which principal can do what operation on what data

P1	C1	C2	Add	OK	C1^C2
P2	C1	C2	Add	Trap	
P1	C3	C4	Add	Trap	
P2	C3	C4	Add	OK	C3

Zero Kernel System: Least Privilege

I'm the Principal accessing data in my Compartment

Howie

I make a Gate Call:
The new Principal is the Scheduler acting for me

Only the necessary data is copied into a shared compartment

Scheduler Acting For Howie

This Principal can only access data in the shared compartment.

Shared

This Principal can make Gate Calls to the core of the Scheduler, but it can't pass my data to it.

Core Scheduler

Scheduler Data

Summary

- Novel hardware can work with the OS and language runtime to enforce the intended semantics
- Memory & Type Safety & Information Flow
- Complete Mediation
- Least Privilege

THANK YOU

Howard Shrobe

Principal Research Scientist, Director Security@CSAIL