Locking Down Insecure Indirection with Hardware-Based Control-Data Isolation

William Arthur, Sahil Madeka,
Reetuparna Das, Todd Austin

MICRO, Waikiki, Hawaii, US
December 7th 2015

Goal of this work

MAKE **SOFTWARE** MORE **SECURE**

Reducing the software attack surface
by **subtracting** the **root cause**
leading to many software exploits today

Accomplished by **locking down**
insecure **indirection**

Locking Down Insecure Indirection (1)

- Every control transfer in executing application comes from the programmer:
  - Every PC address encoded in instructions, **OR**
  - Is derived from secure hardware structures

Executing application **always** adheres to the **programmer-defined** control-flow graph

Stopping control-flow **attacks**
which derail the CFG

Locking Down Insecure Indirection (2)

- Achieved by hardware-software co-design

Software:

Eliminate all indirect control-flow instructions – via **Control-Data Isolation (CDI)** [1]

Hardware:

Memoization of secure control transitions in secure hardware – via Indirect **Edge Cache**

[1] *Getting in Control of Your Control Flow with Control-Data Isolation*, Arthur et al., CGO 2015

Outline

- Software (in)security – Control-Flow Attack
- Hardware-Based Control-Data Isolation
- Measure performance and security
- Conclusions

Control-Flow Attack

Control-Flow Attacks

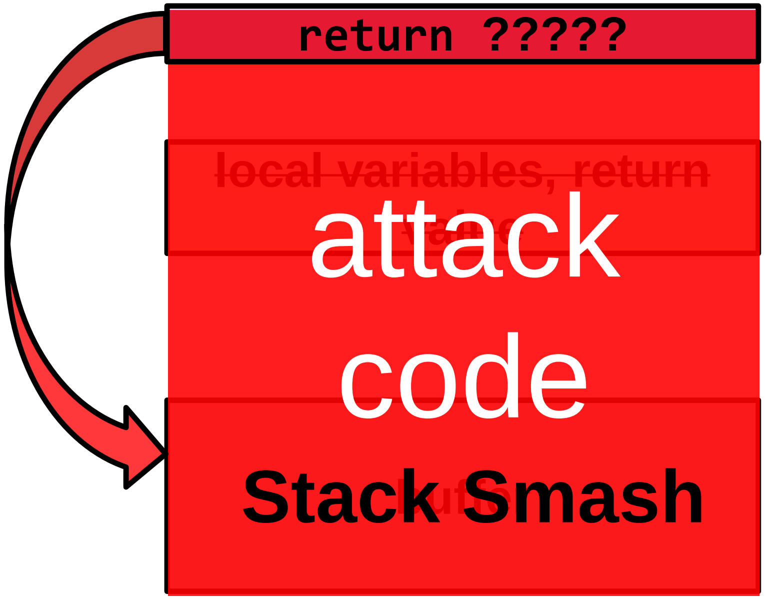
violate, at runtime, the CFG of an application by corrupting the PC with user-injected data

Buffer Overflow

Heap Spray

Return-to-libc

Code Gadgets



Outline

- Software (in)security
- Hardware-Based Control-Data Isolation
- Measure performance and security
- Conclusions

Control-Data Isolation

```
int bar() {
    // function code
    return;
}
```

Vulnerable Code

```
int bar() {
    // function code
    return;
}
```

ret

Vulnerable Code

White-list of valid CFG edges

```
int bar() {
// function code
if([%esp]==_ret1)
    jump _ret1;
else if([%esp]==_ret2)
    jump _ret1;
else
    call _abort;
}
```

“**Sled**” of conditional branches and **direct** jumps

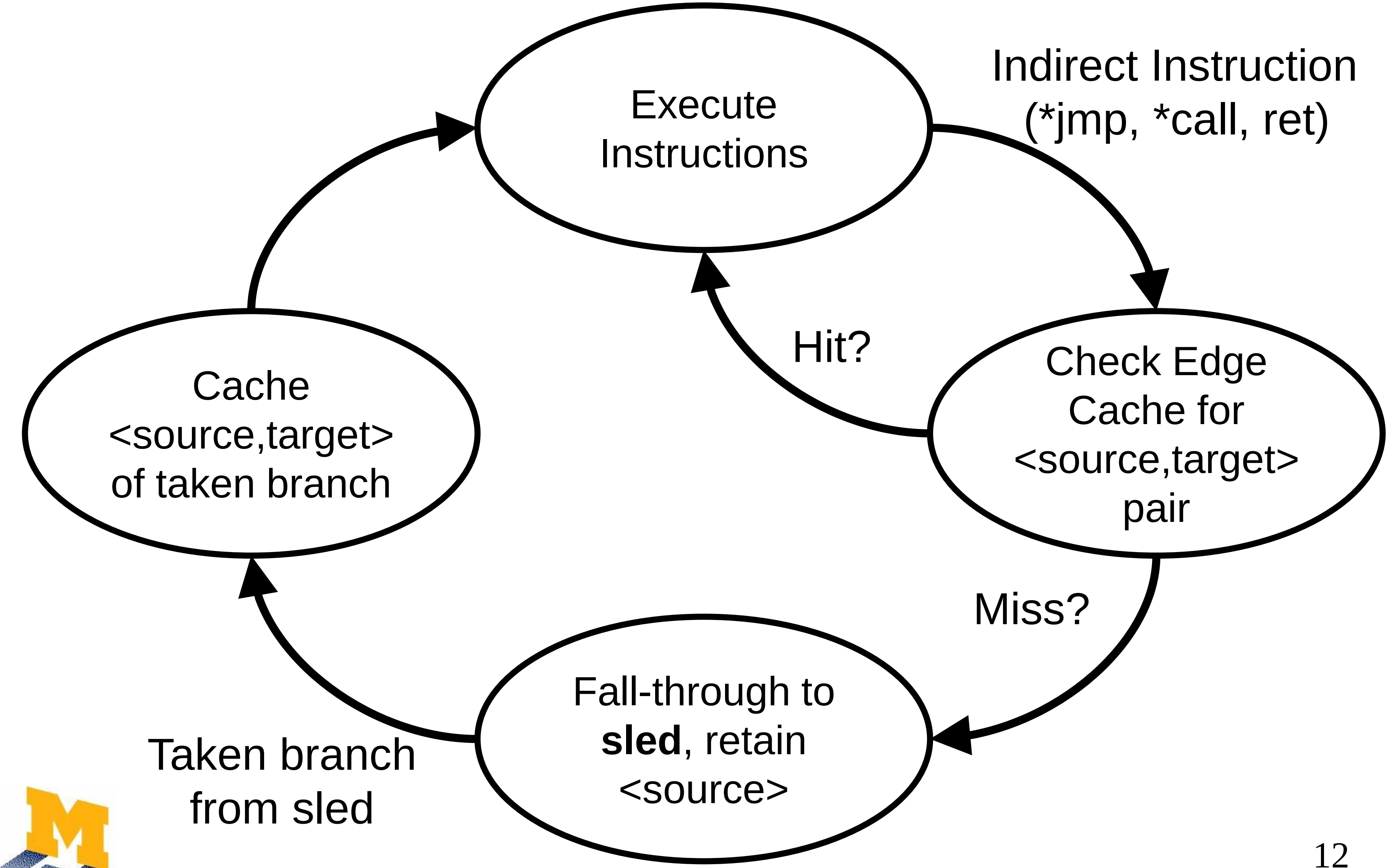
Hardware-Based CDI

- Software-only CDI (CGO ’15) retains higher than desired runtime overheads for some applications – 31% for gcc

Key insight: **Caching** previously executed sled **edges** obviates subsequent re-executions of the **sled**

Addition of hardware **edge cache**

Hardware-Based CDI Algorithm



Execute Instructions

Indirect Instruction (*jmp, *call, ret)

Check Edge Cache for <source,target> pair

Hit?

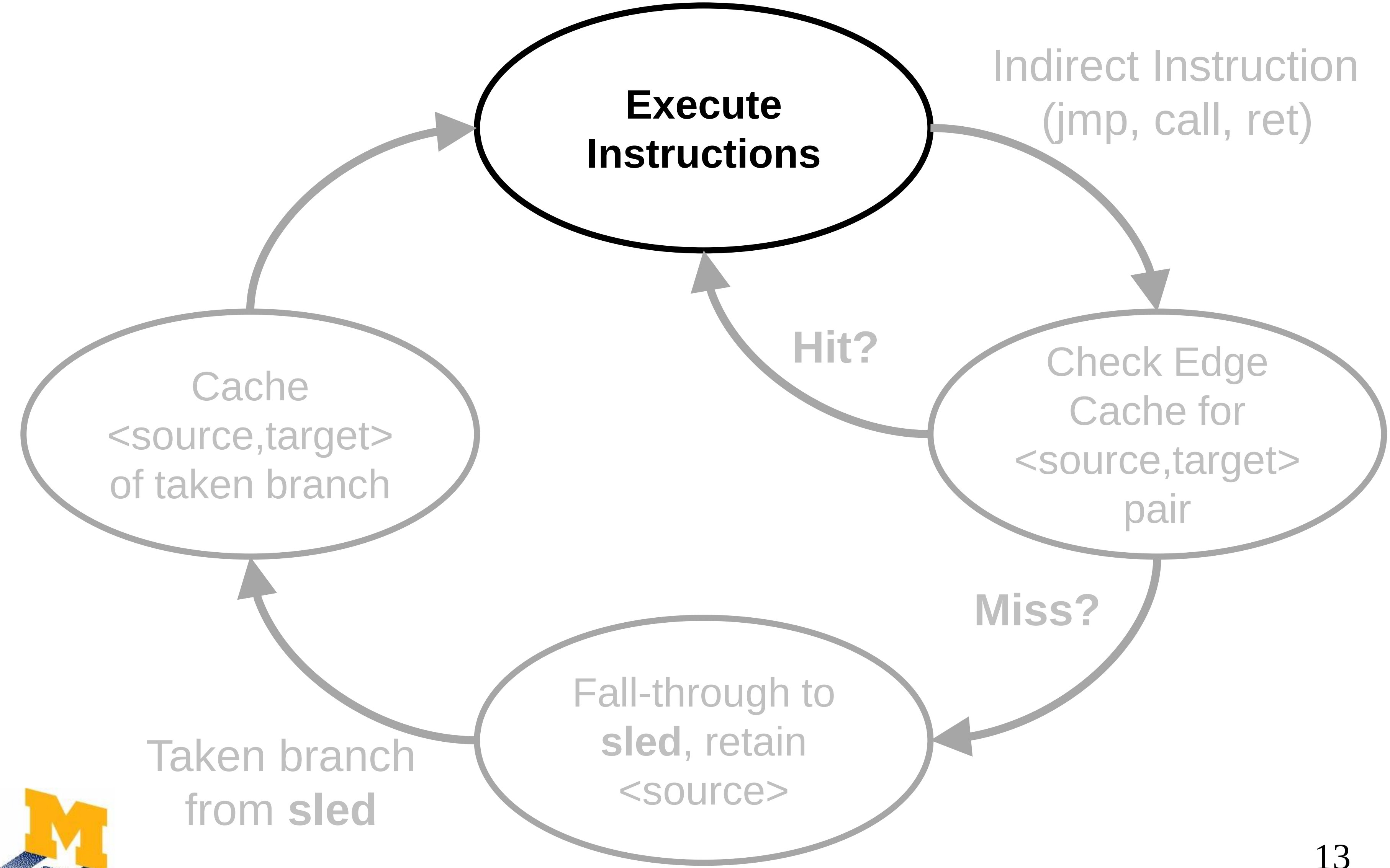
Miss?

Fall-through to **sled**, retain <source>

Taken branch from sled

Cache <source,target> of taken branch

Hardware-Based CDI Algorithm



Execute Instructions

Indirect Instruction (jmp, call, ret)

Check Edge Cache for <source,target> pair

Hit?

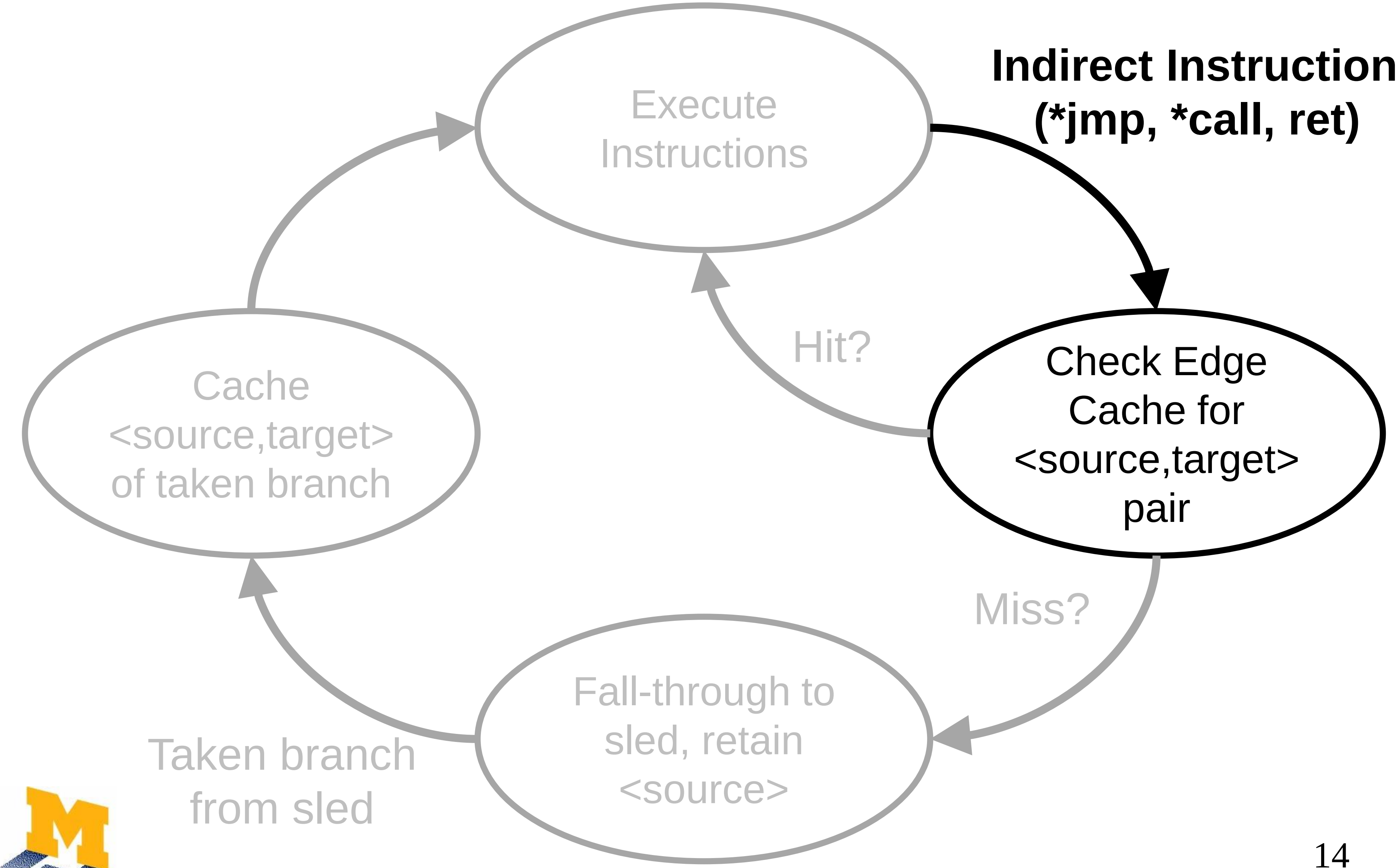
Miss?

Fall-through to **sled**, retain <source>

Taken branch from **sled**

Cache <source,target> of taken branch

Hardware-Based CDI Algorithm



Indirect Instruction (*jmp, *call, ret)

Execute Instructions

Check Edge Cache for <source,target> pair

Hit?

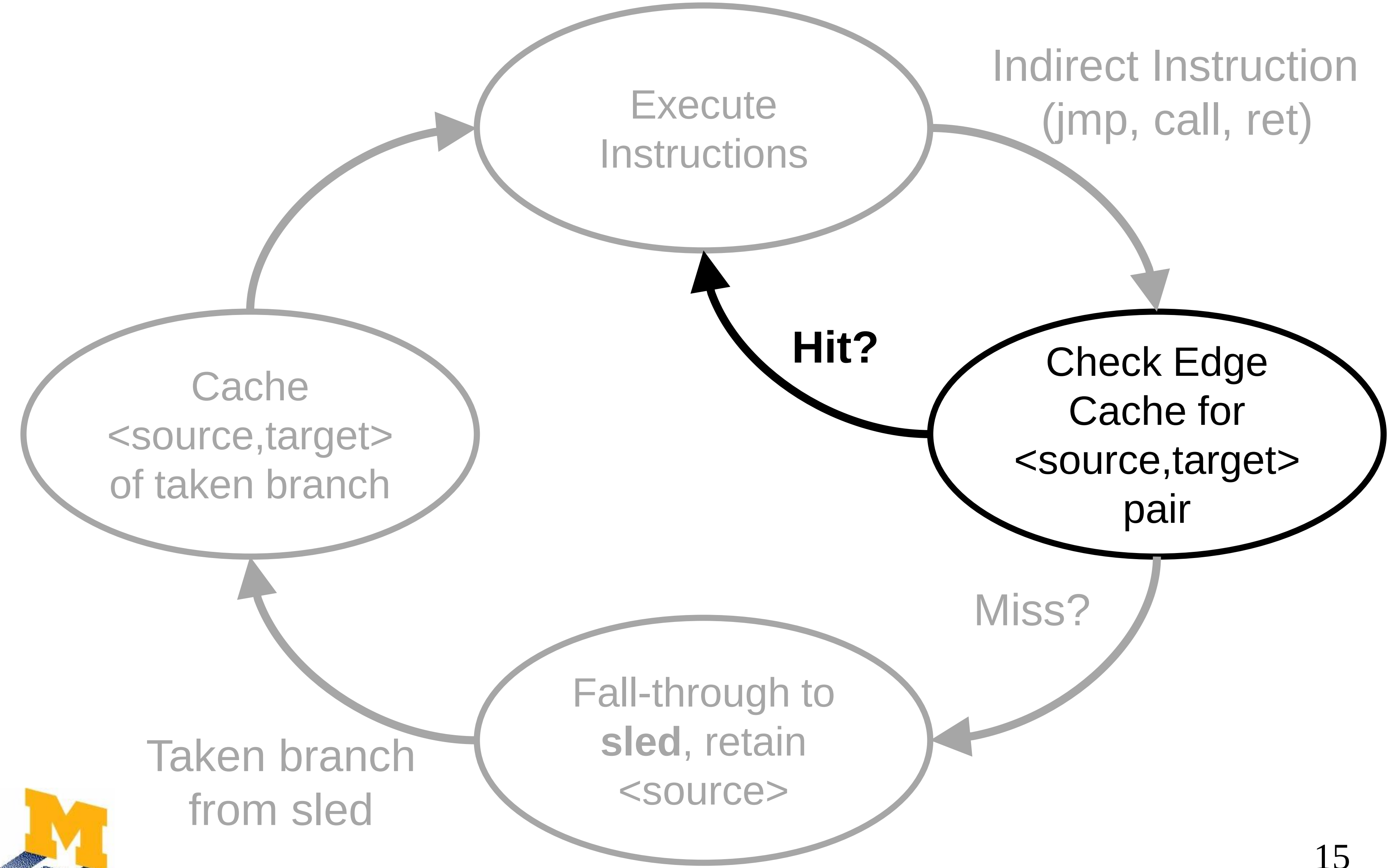
Miss?

Fall-through to sled, retain <source>

Taken branch from sled

Cache <source,target> of taken branch

Hardware-Based CDI Algorithm



Execute Instructions

Indirect Instruction (jmp, call, ret)

Check Edge Cache for <source,target> pair

Hit?

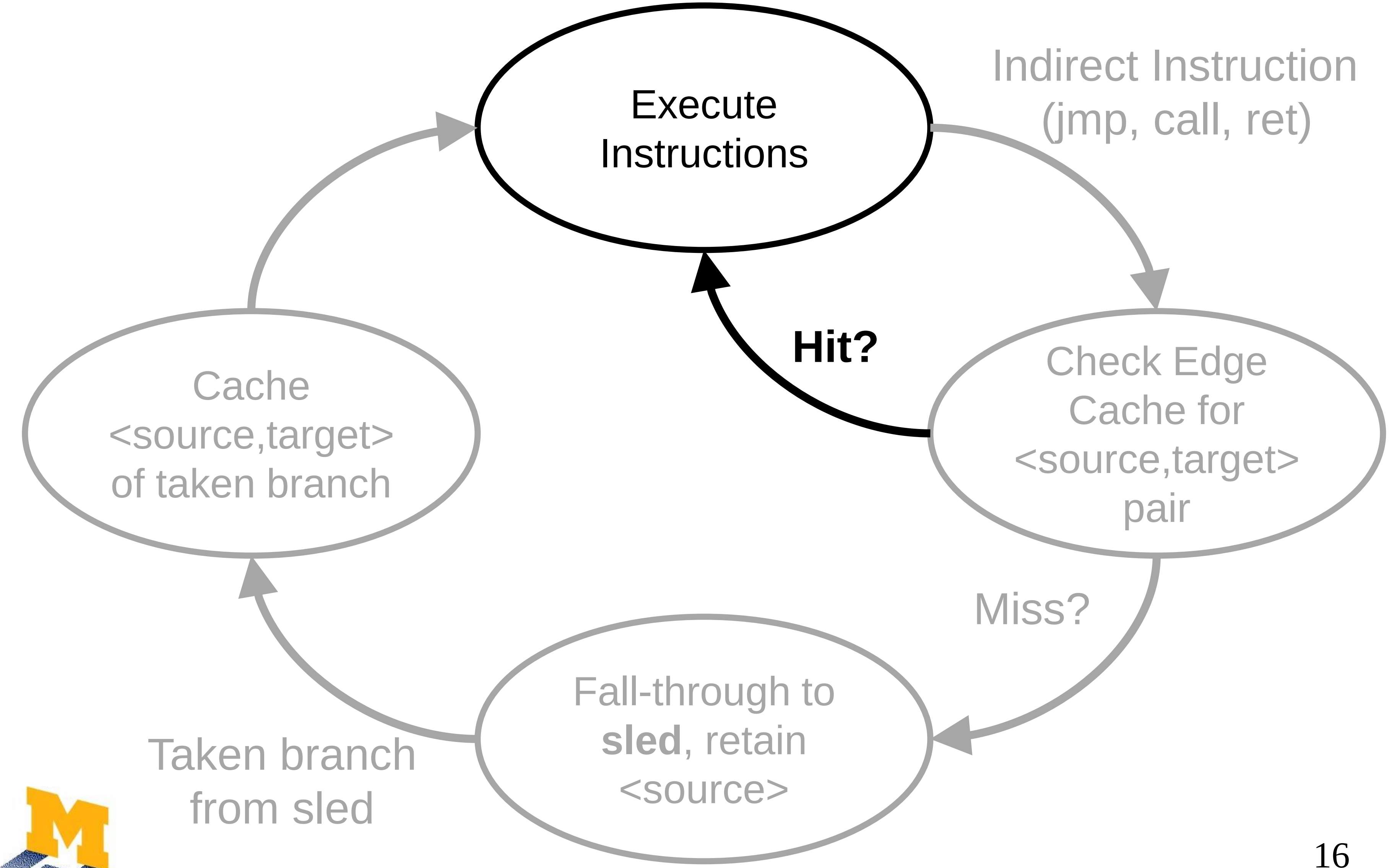
Miss?

Fall-through to **sled**, retain <source>

Taken branch from sled

Cache <source,target> of taken branch

Hardware-Based CDI Algorithm



Execute Instructions

Indirect Instruction (jmp, call, ret)

Check Edge Cache for <source,target> pair

Hit?

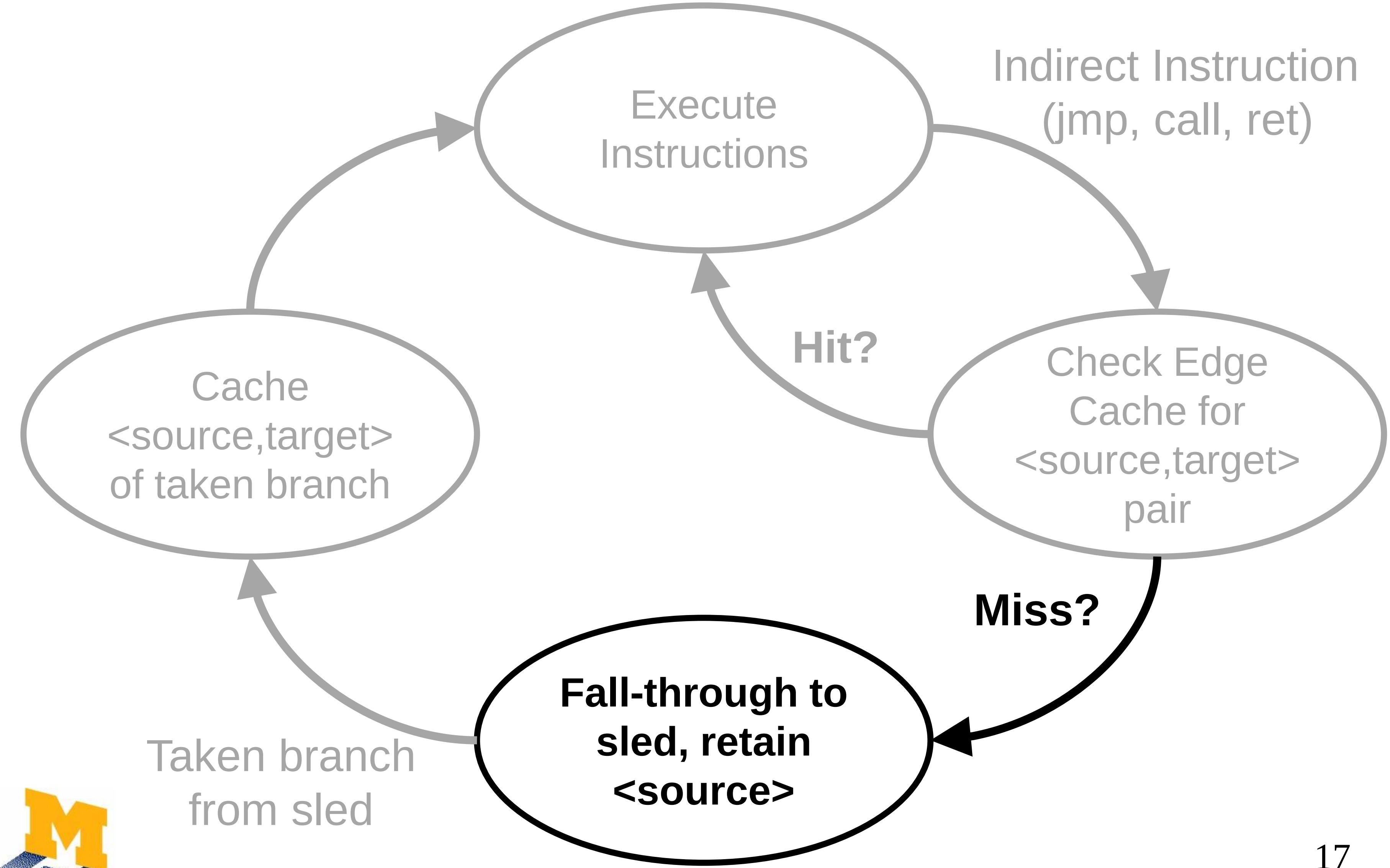
Miss?

Fall-through to **sled**, retain <source>

Taken branch from sled

Cache <source,target> of taken branch

Hardware-Based CDI Algorithm



Execute Instructions

Indirect Instruction (jmp, call, ret)

Check Edge Cache for <source,target> pair

Hit?

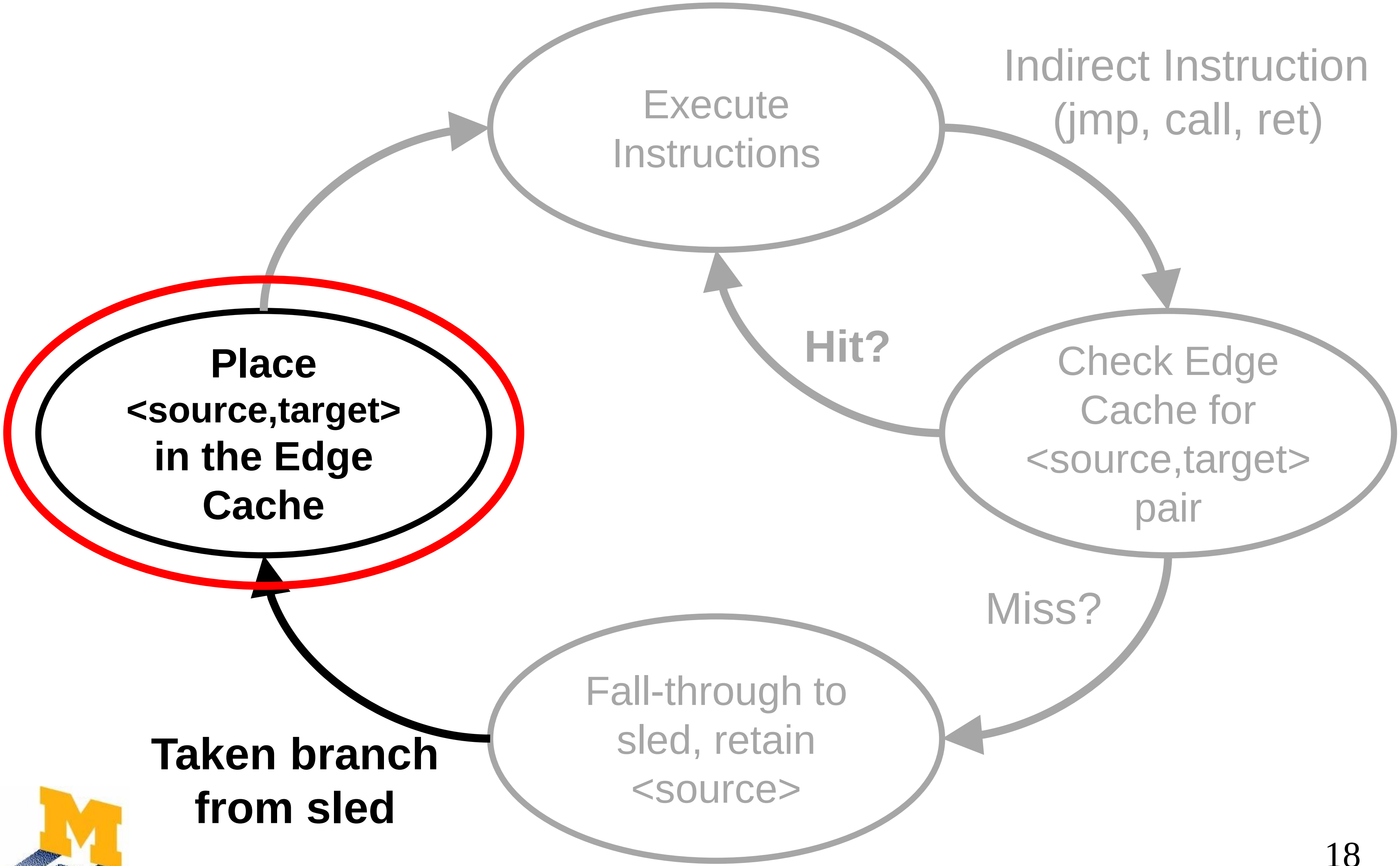
Miss?

Fall-through to sled, retain <source>

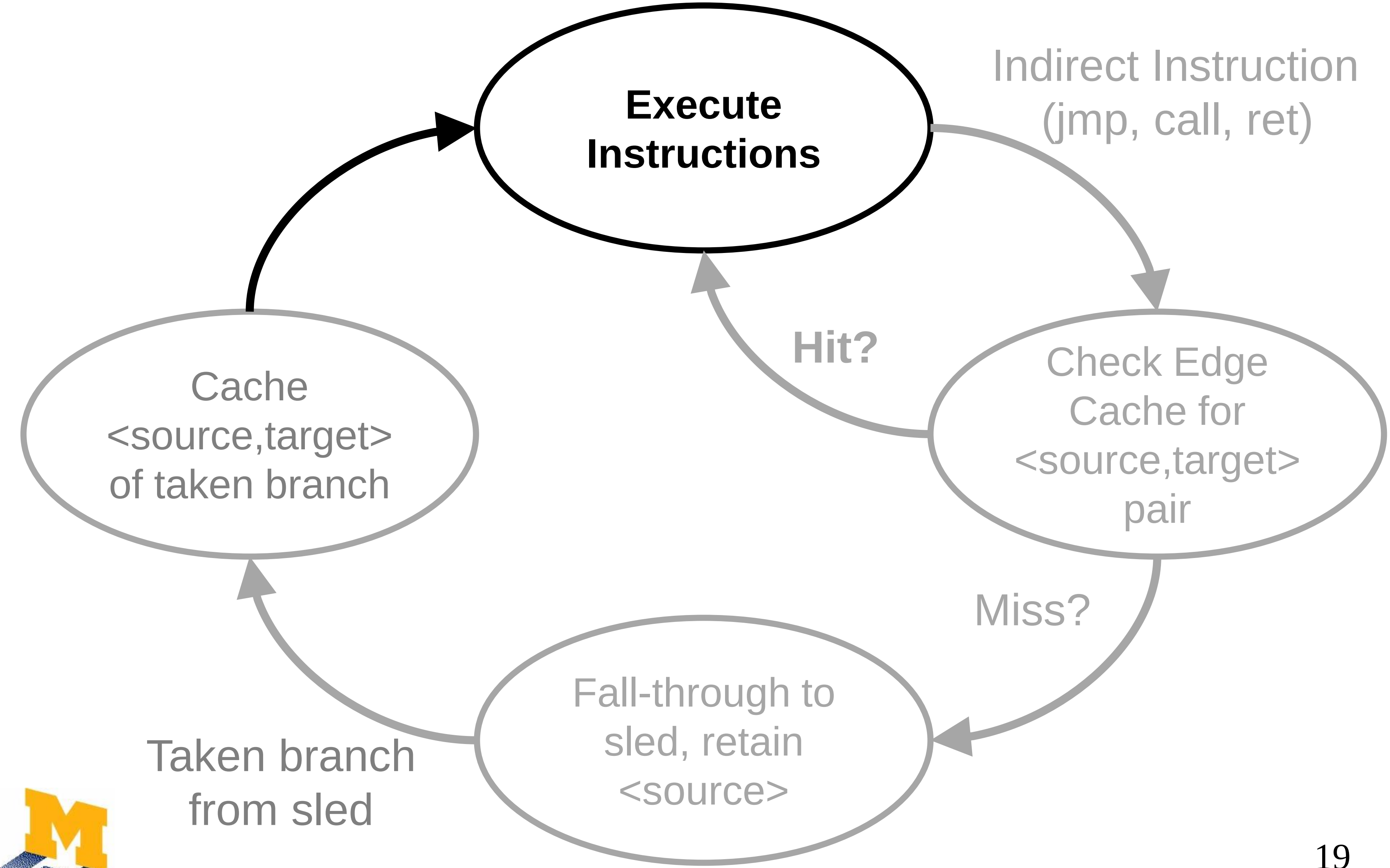
Taken branch from sled

Cache <source,target> of taken branch

Hardware-Based CDI Algorithm


Execute Instructions
Indirect Instruction (jmp, call, ret)
Check Edge Cache for <source,target> pair
Hit?
Miss?
Fall-through to sled, retain <source>
Taken branch from sled
Place <source,target> in the Edge Cache

Hardware-Based CDI Algorithm



Execute Instructions

Indirect Instruction (jmp, call, ret)

Check Edge Cache for <source,target> pair

Hit?

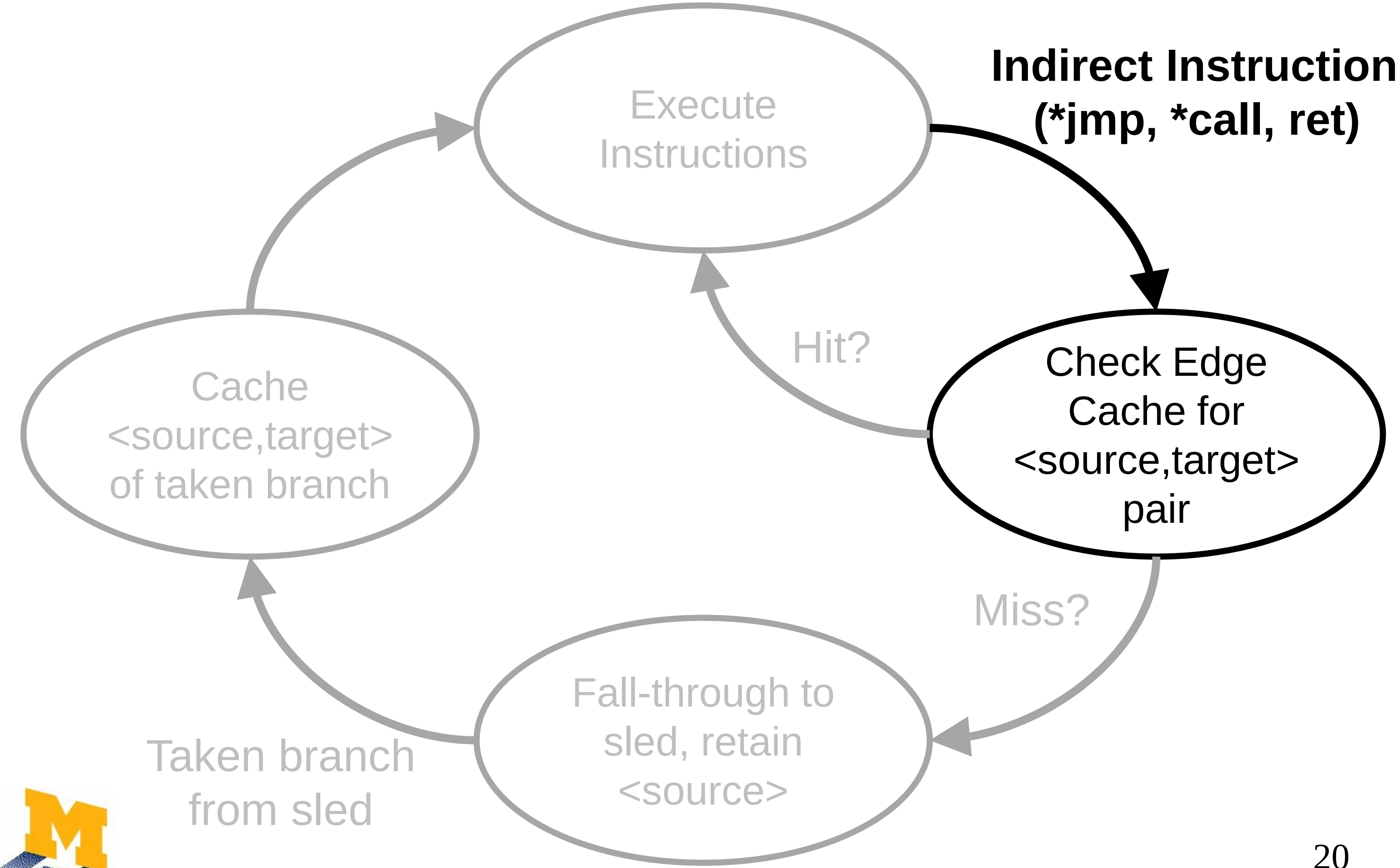
Miss?

Fall-through to sled, retain <source>

Taken branch from sled

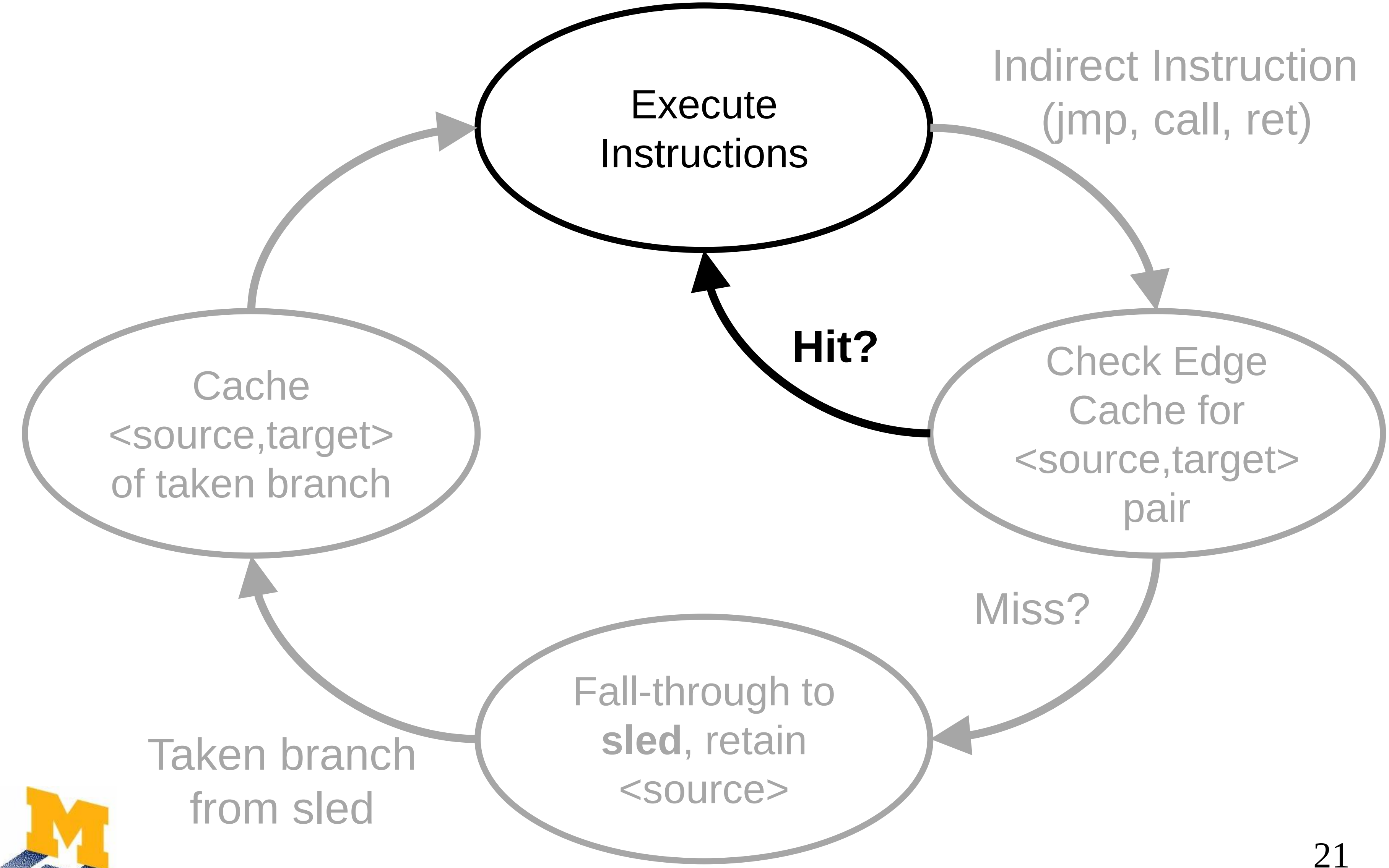
Cache <source,target> of taken branch

Hardware-Based CDI Algorithm



Indirect Instruction
(*jmp, *call, ret)
Execute Instructions
Check Edge Cache for <source,target> pair
Hit?
Miss?
Fall-through to sled, retain <source>
Taken branch from sled
Cache <source,target> of taken branch

Hardware-Based CDI Algorithm



Execute Instructions

Indirect Instruction (jmp, call, ret)

Check Edge Cache for <source,target> pair

Hit?

Miss?

Fall-through to **sled**, retain <source>

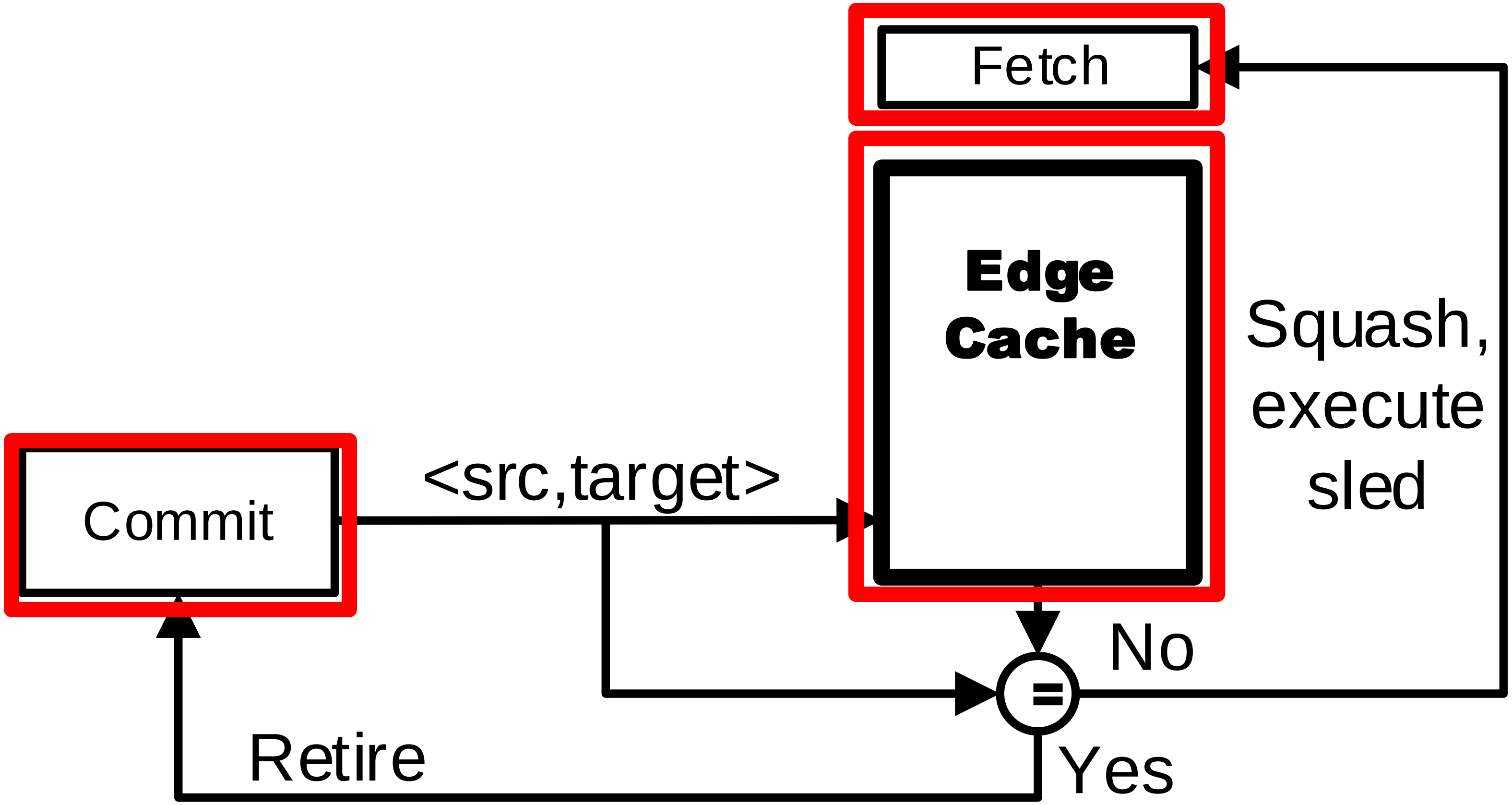
Taken branch from sled

Cache <source,target> of taken branch

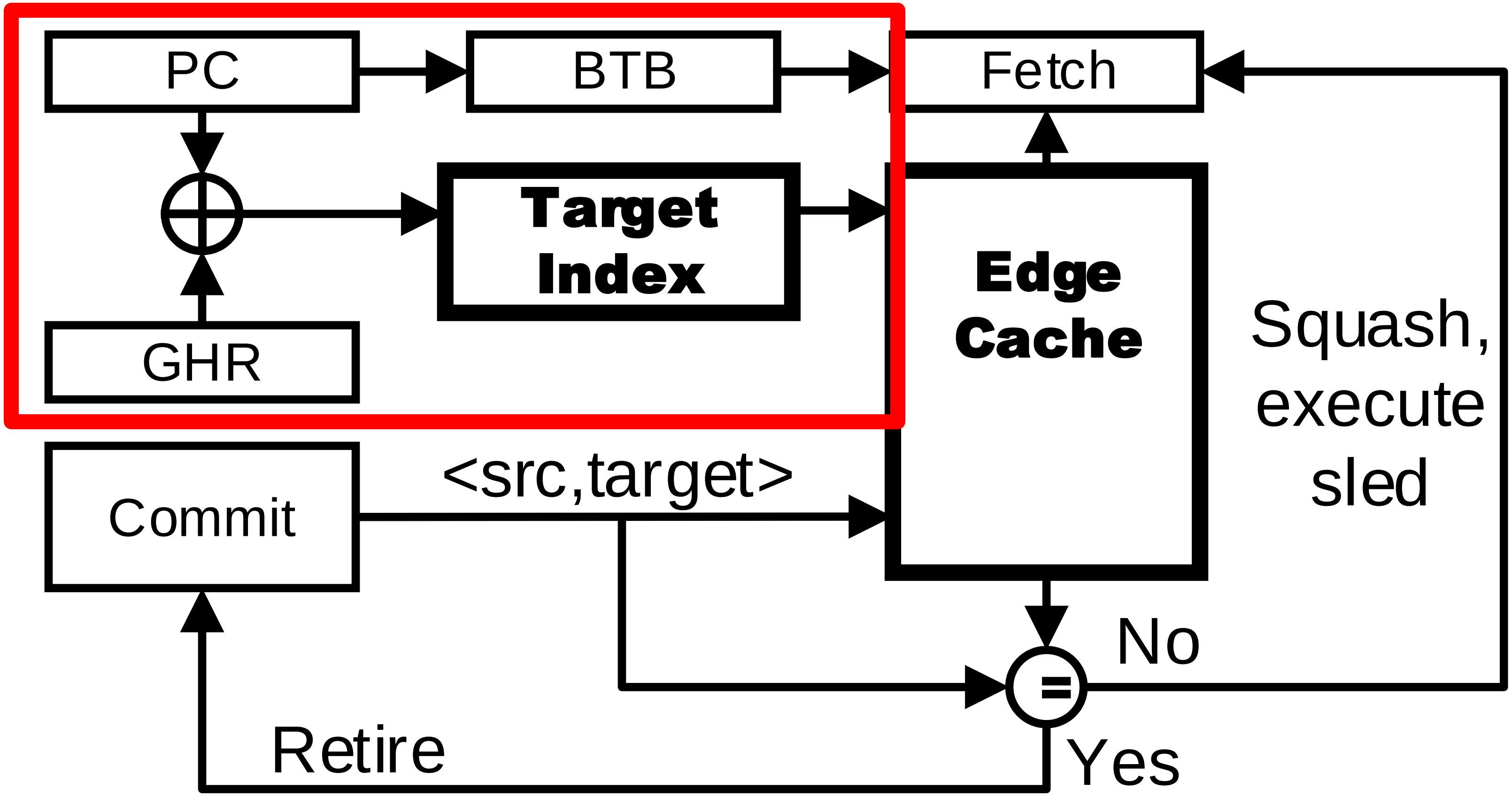
Edge Cache(1)

- New hardware structure – **edge cache**
  - **Memoization** of most recent indirect edges

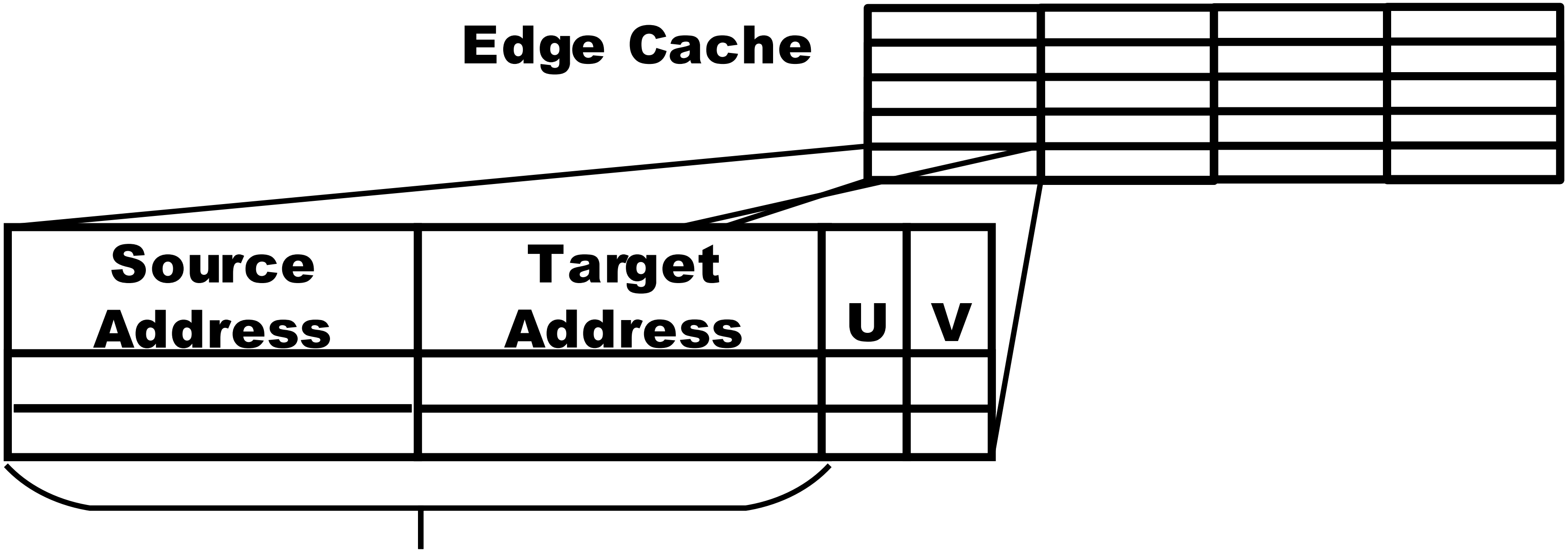
Edge Cache(2)



Edge Cache(2)


PC
BTB
Fetch
Target Index
Edge Cache
GHR
Squash, execute sled
Commit
<src,target>
No
=
Retire
Yes

Challenges

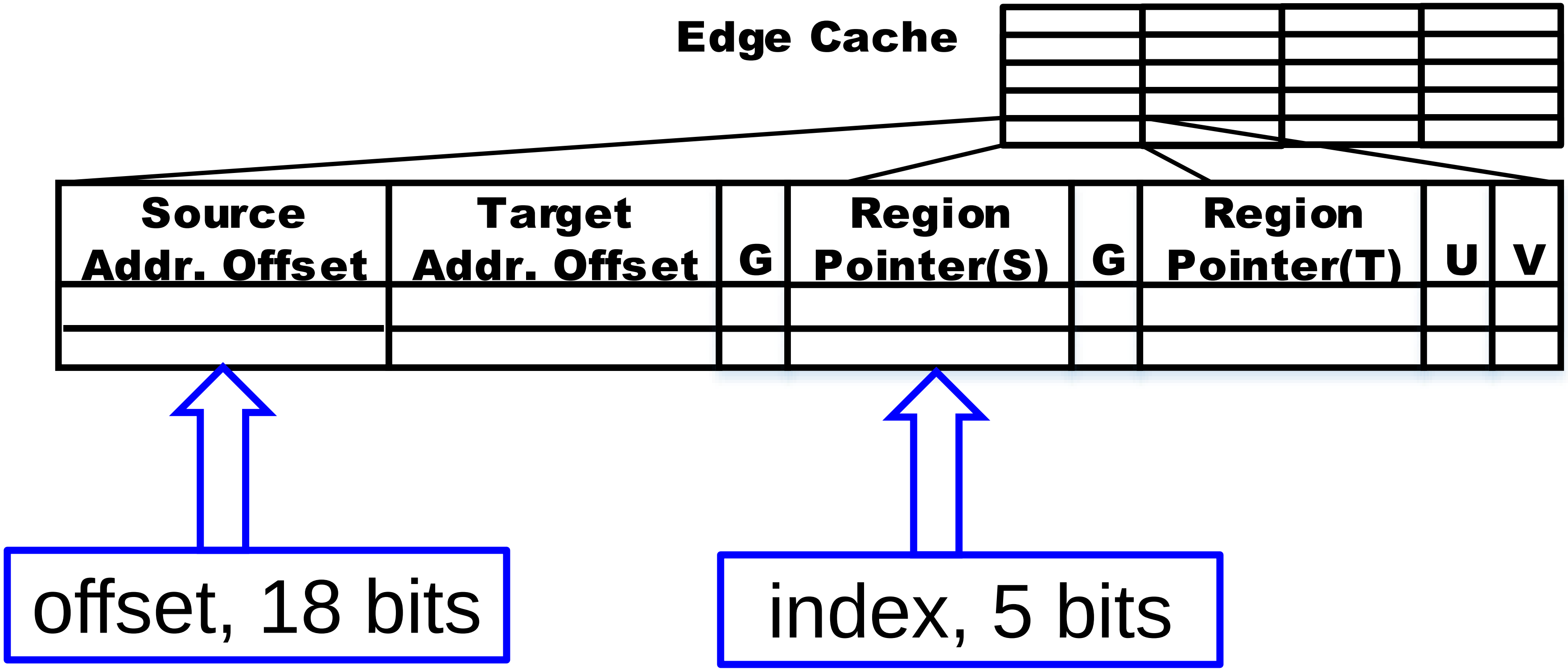


tag, full address

128 + 2 bits per entry!

1k entries = 16 kB

Region Table



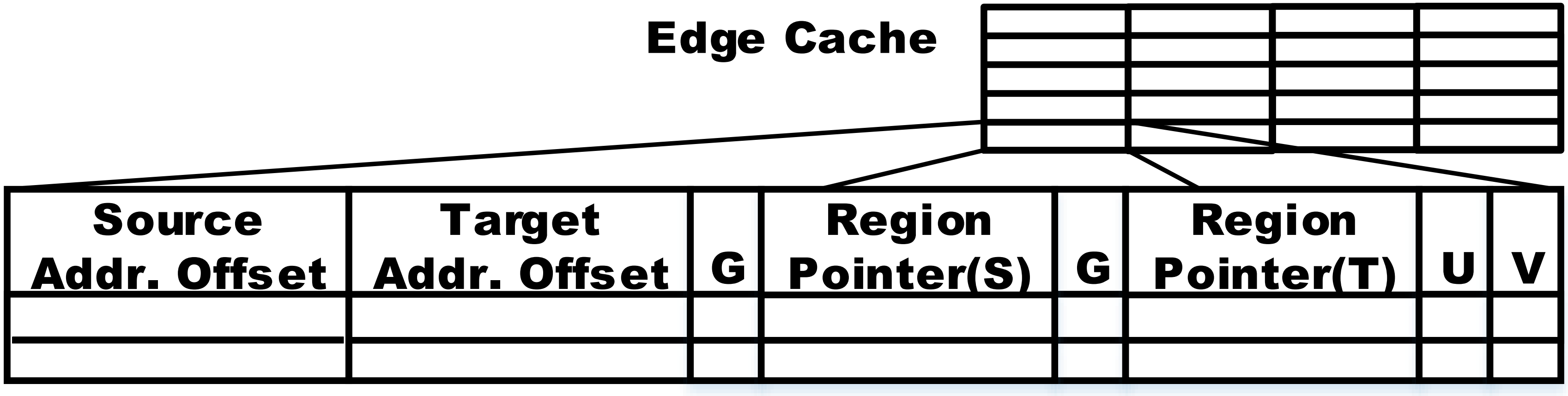
Edge Cache

Source Addr. Offset	Target Addr. Offset	G	Region Pointer(S)	G	Region Pointer(T)	U	V

offset, 18 bits

index, 5 bits

Region Table



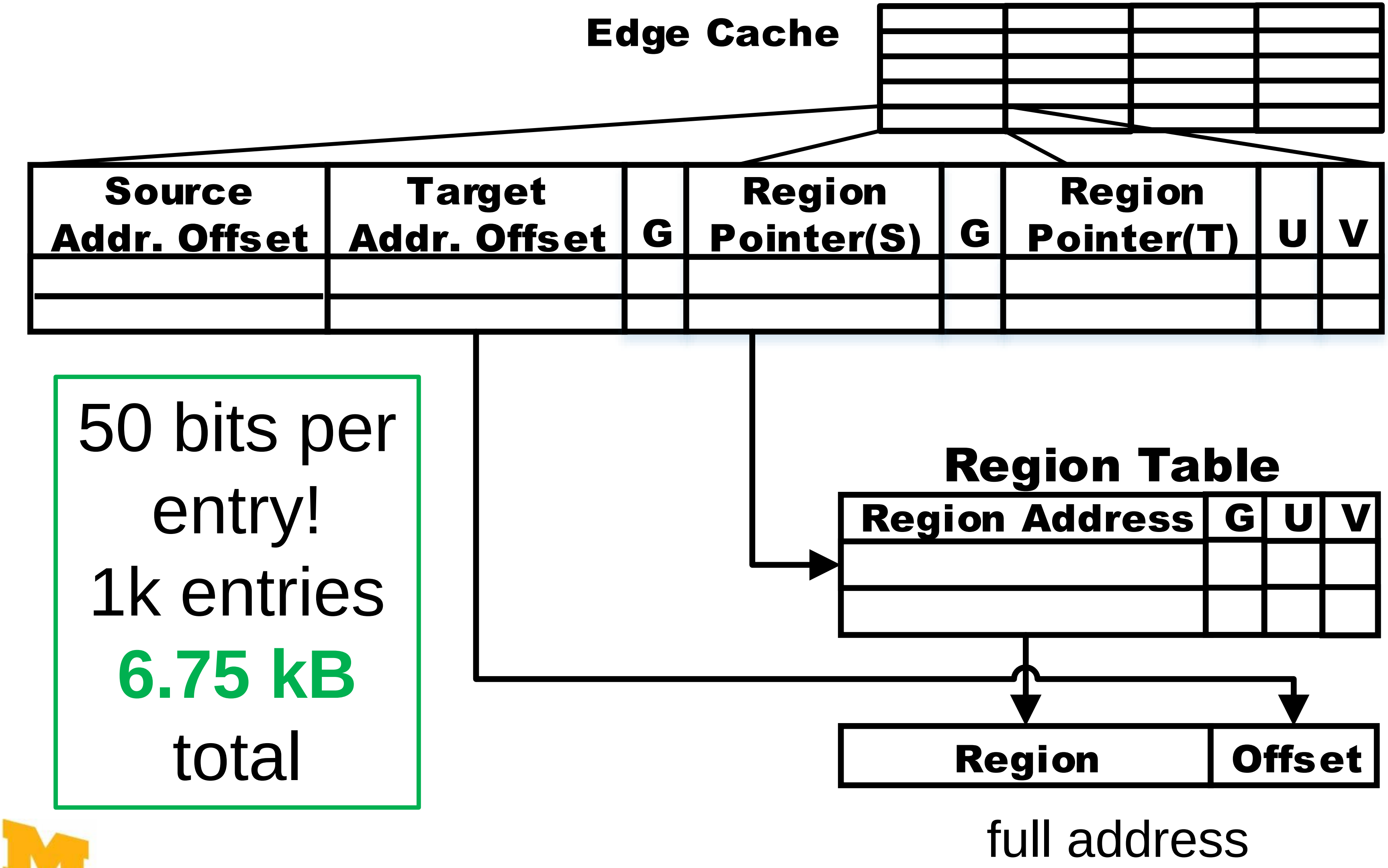
Edge Cache

Source Addr. Offset	Target Addr. Offset	G	Region Pointer(S)	G	Region Pointer(T)	U	V

Region Table

Region Address	G	U	V

Region Table



50 bits per entry!

1k entries

6.75 kB

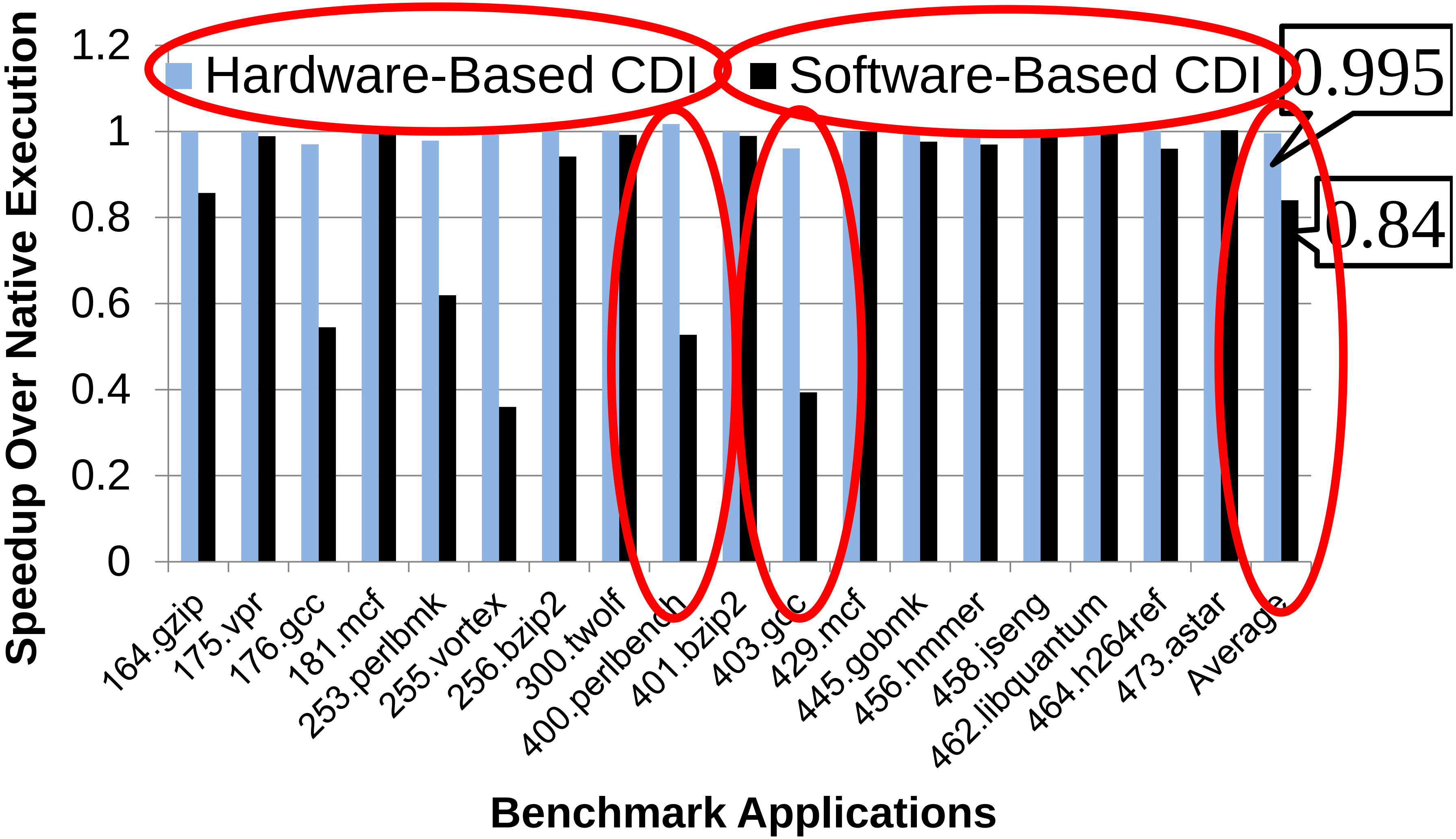
total

Outline

- Software (in)security
- Hardware-Based Control-Data Isolation
- Measure **performance** and **security**
- Conclusions

Experimental Setup

- ***gem5*** architectural simulator
  - Detailed **O3 cpu** model, configured similar to *Intel Haswell* processor, ***x86-64***
- SPECINT 2000 & 2006
- **1,024-entry edge cache**
  - 4-way set associative
- **32-entry region table**



Branch prediction – 6% speedup 400.perlbench vs BTB

Security

- Average Indirect target Reduction – AIR [2]
- Measure of the reduction in the software attack surface

99.999%+ reduction in indirect target set

Average of **tens of targets** per indirect

Previous works: average of **tens of thousands** of targets per indirect instruction

[2] *Control Flow Integrity for COTS Binaries*, Zhang and Sekar, USENIX Security 2013

Conclusions

- Locking down **insecure indirection** can **eliminate** contemporary control-flow attacks
- Hardware-based control-data isolation efficiently realizes this capability
  - **Minimal** runtime **overhead** – 0.5%

Thank You

Questions?