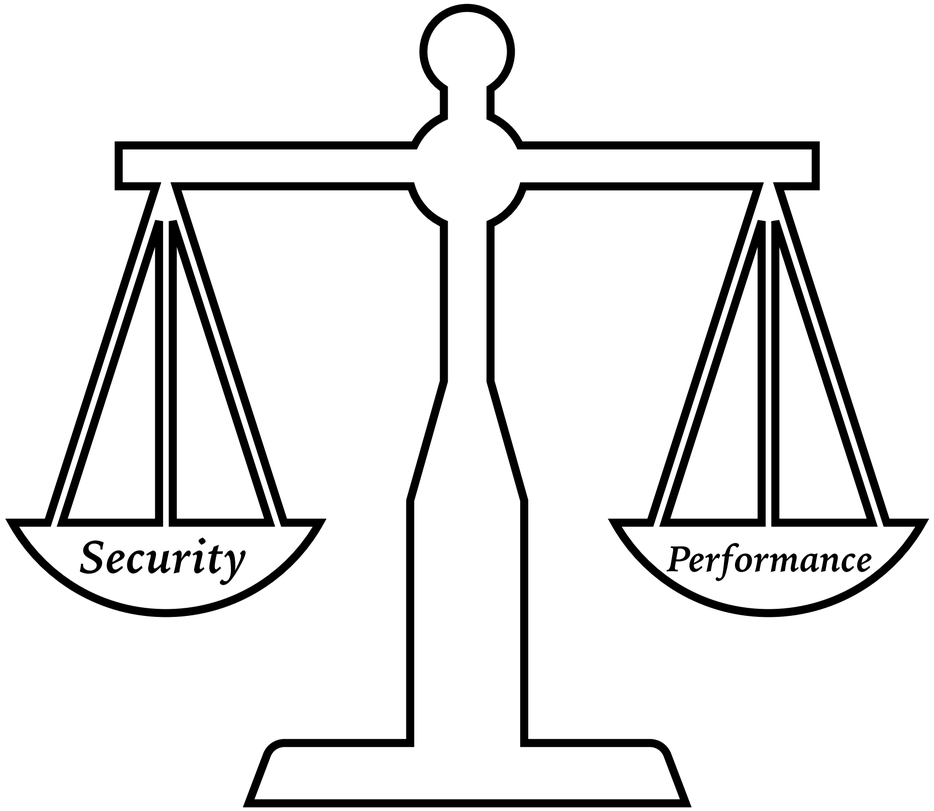
UC San Diego
JACOBS SCHOOL OF ENGINEERING
Computer Science and Engineering

University of Virginia

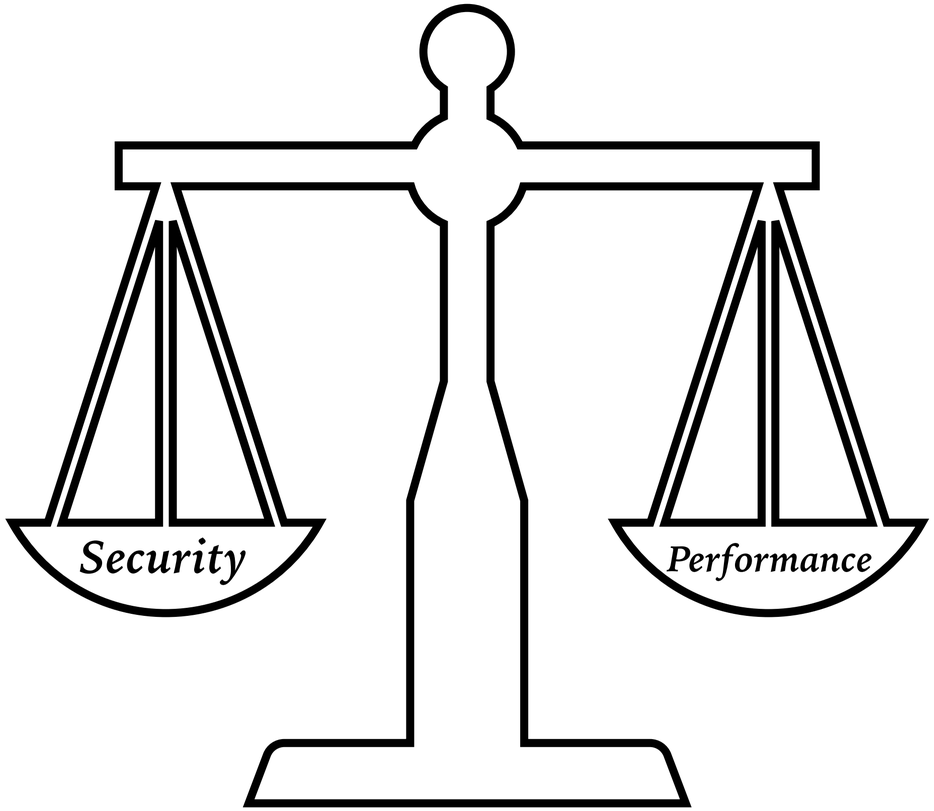
CONTEXT-SENSITIVE FENCING:
SECURING SPECULATIVE EXECUTION VIA MICROCODE CUSTOMIZATION

Mohammadkazem Taram, Ashish Venkat, Dean Tullsen
University of California San Diego, University of Virginia

PERFORMANCE V.S. SECURITY

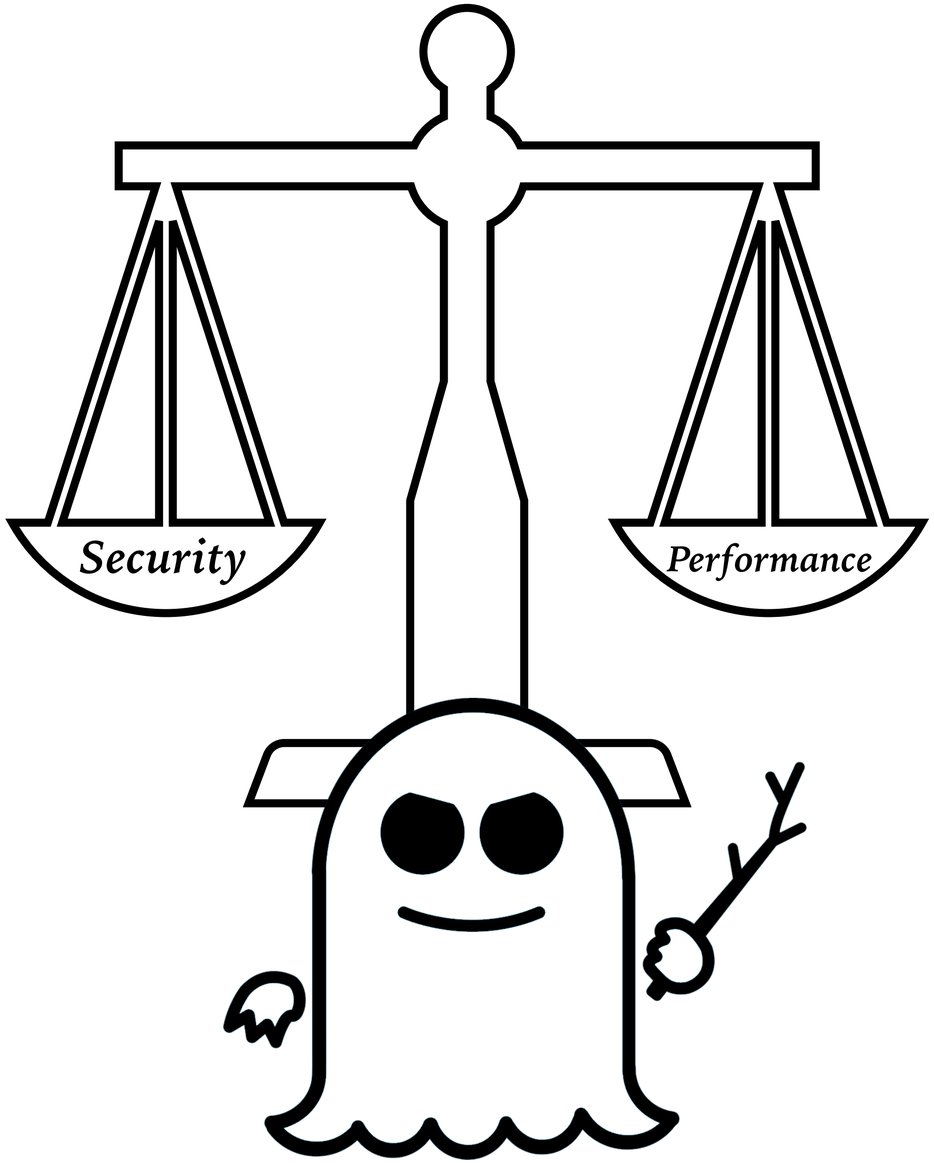


PERFORMANCE V.S. SECURITY



PERFORMANCE V.S. SECURITY

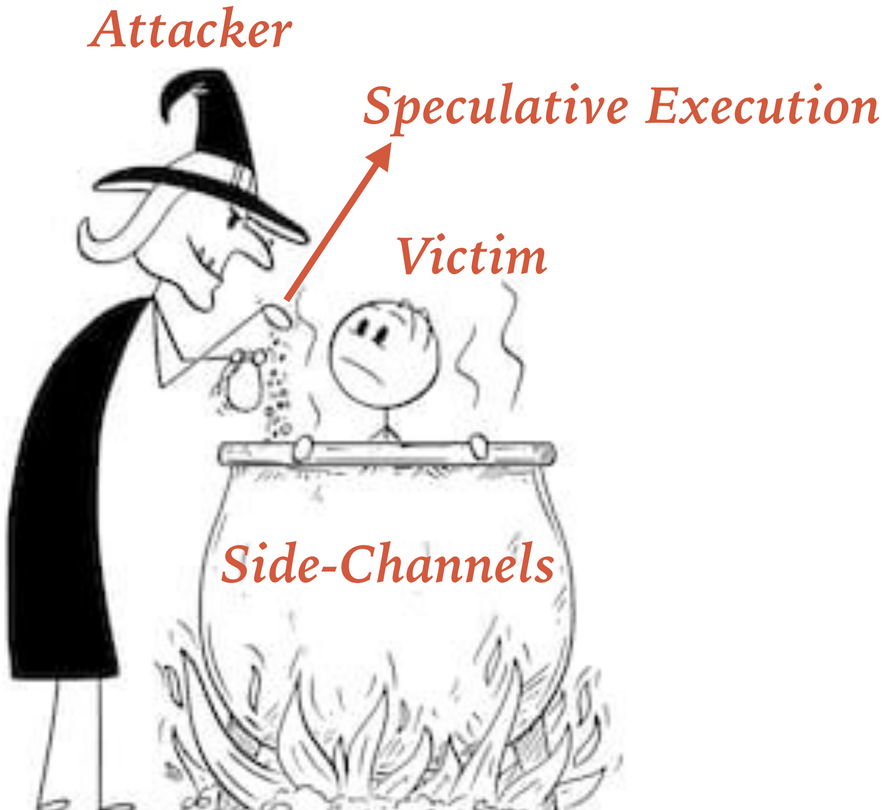
FORESHADOW



MELTDOWN

SPECTRE ATTACKS!

- Leak secrets via side-channels + speculative execution
- Any modern processor with a Branch Predictor is vulnerable



SPECTRE V1 — BOUNDS CHECK BYPASS

```
int Kernel_api_( int x ){

     y = array2[array1[x] * 64];
}
```

SPECTRE V1 — BOUNDS CHECK BYPASS

```
int Kernel_api_( int x ){
     if ( x < array1_size) //bounds check
        y = array2[array1[x] * 64];
}
```

SPECTRE V1 — BOUNDS CHECK BYPASS

```
int Kernel_api_( int x ){
Mispredicted  if ( x < array1_size) //bounds check
                 y = array2[array1[x] * 64]; //not taken/fallthrough code
}
```

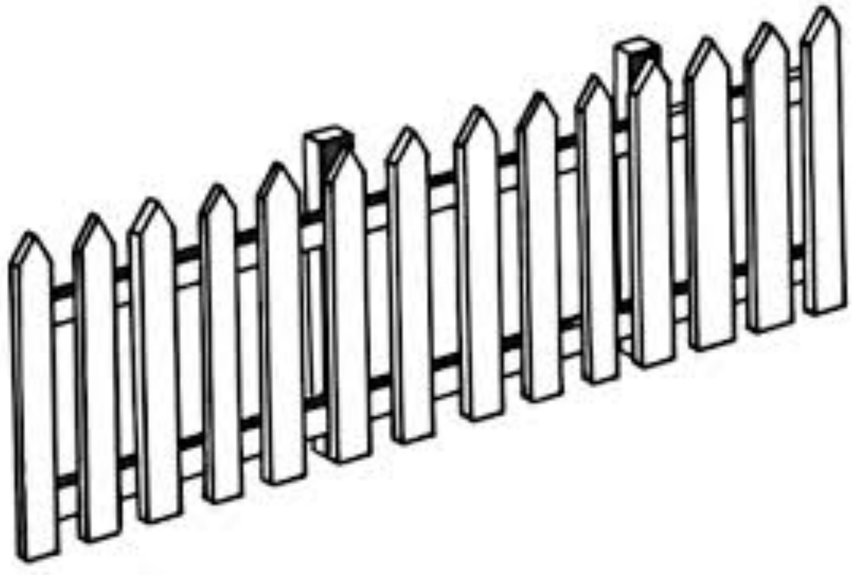
SPECTRE V1 — BOUNDS CHECK BYPASS

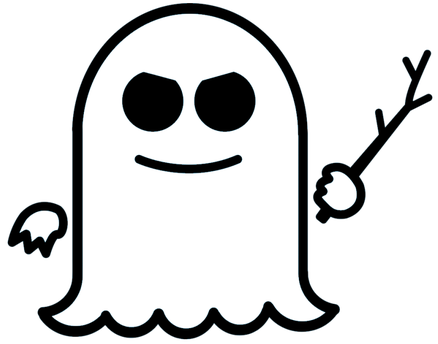
```
int Kernel_api_( int x ){
Mispredicted  if ( x < array1_size) //bounds check
                 y = array2[array1[x] * 64]; //not taken/fallthrough code
}
```

Too late to recover — data is exposed via side-channels

CURRENT SPECTRE V1 MITIGATIONS

- Restricting Speculation Using Fences and Barriers:

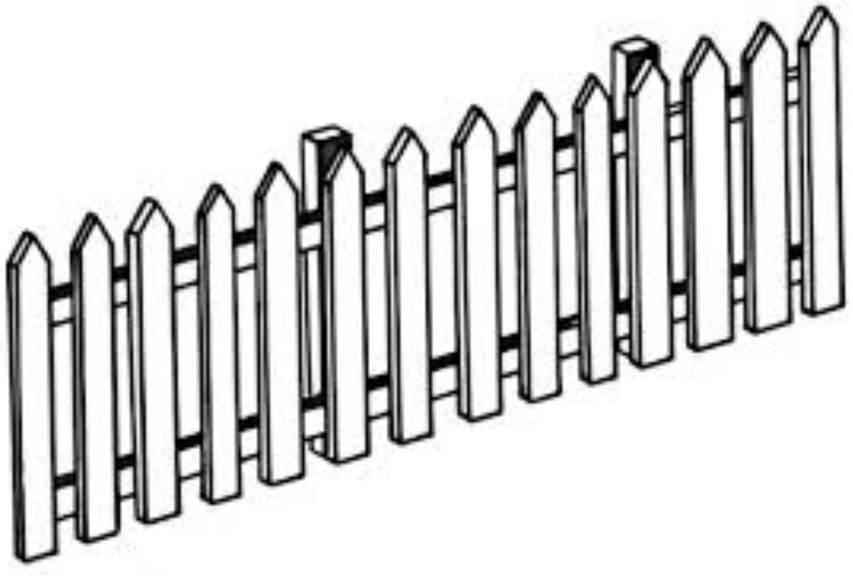


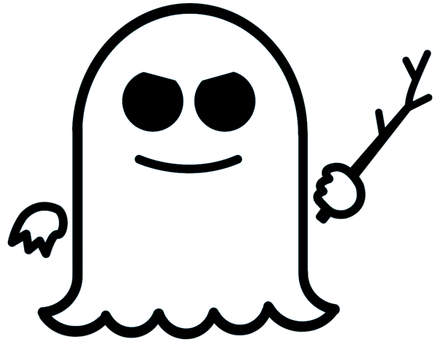


CURRENT SPECTRE V1 MITIGATIONS

- Restricting Speculation Using Fences and Barriers:

```
if ( x < array1_size)
  y = array2[array1[x] * 64];
```

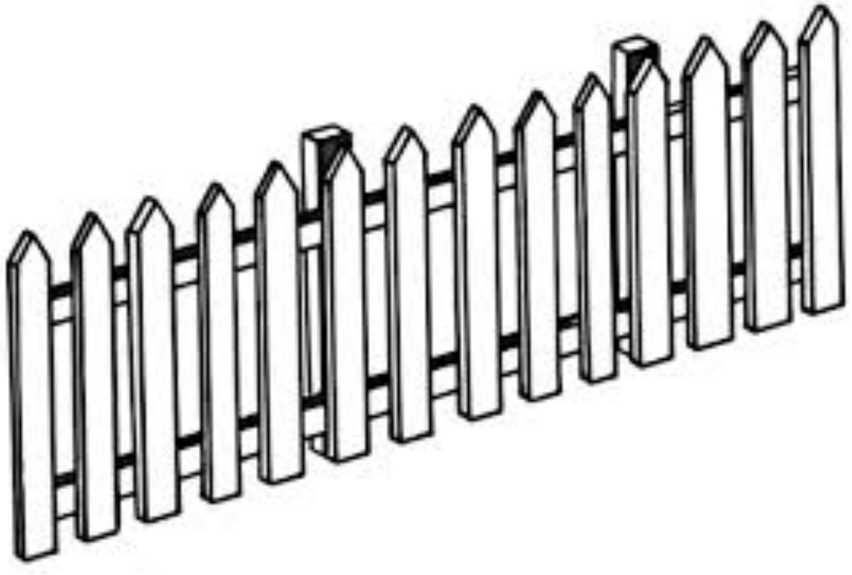


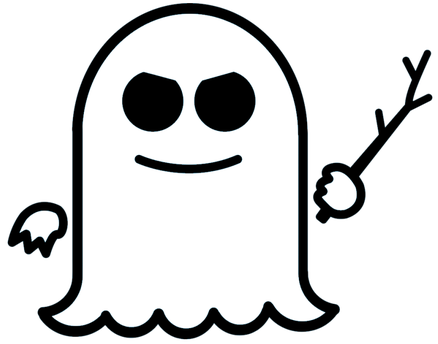


CURRENT SPECTRE V1 MITIGATIONS

- Restricting Speculation Using Fences and Barriers:

```
if ( x < array1_size)
  speculative_fence;
  y = array2[array1[x] * 64];
```





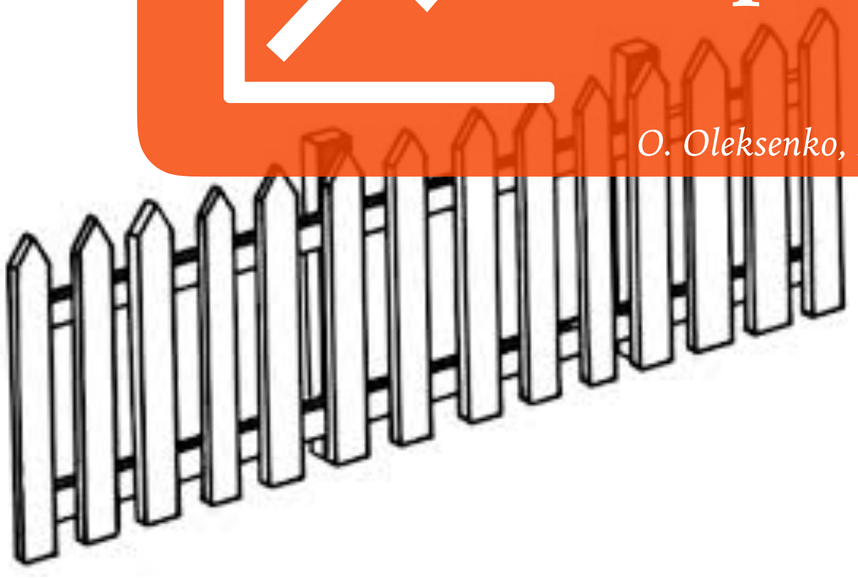
CURRENT SPECTRE V1 MITIGATIONS

- Restricting Speculation Using Fences and Barriers:

```
if ( x < array1_size)
    speculative_fence;
    y = array2[array1[x] * 64];
```

Up to 10x Performance Overhead!

O. Oleksenko, B. Trach, T. Reiher, M. Silberstein, and C.Fetzer. 2018. You Shall Not Bypass



THIS WORK : CONTEXT SENSITIVE FENCING

➤ **Surgically** injects fence micro-ops

THIS WORK : CONTEXT SENSITIVE FENCING

➤ **Surgically** injects fence micro-ops

Only When Necessary

THIS WORK : CONTEXT SENSITIVE FENCING

➤ **Surgically** injects fence micro-ops

`Only When Necessary`

`Right Type of Fence`

THIS WORK : CONTEXT SENSITIVE FENCING

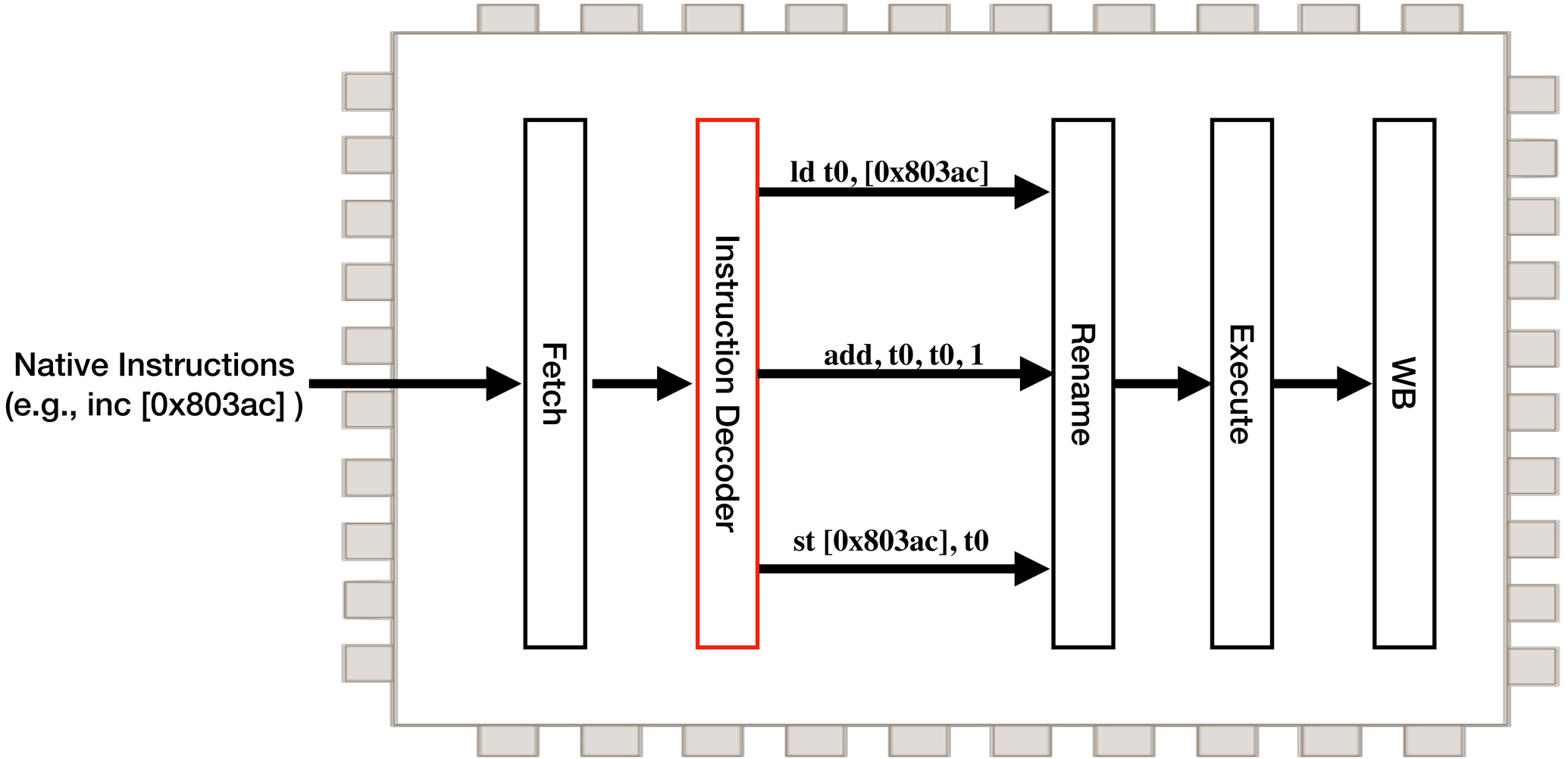
➤ **Surgically** injects fence micro-ops

Only When Necessary

Right Type of Fence

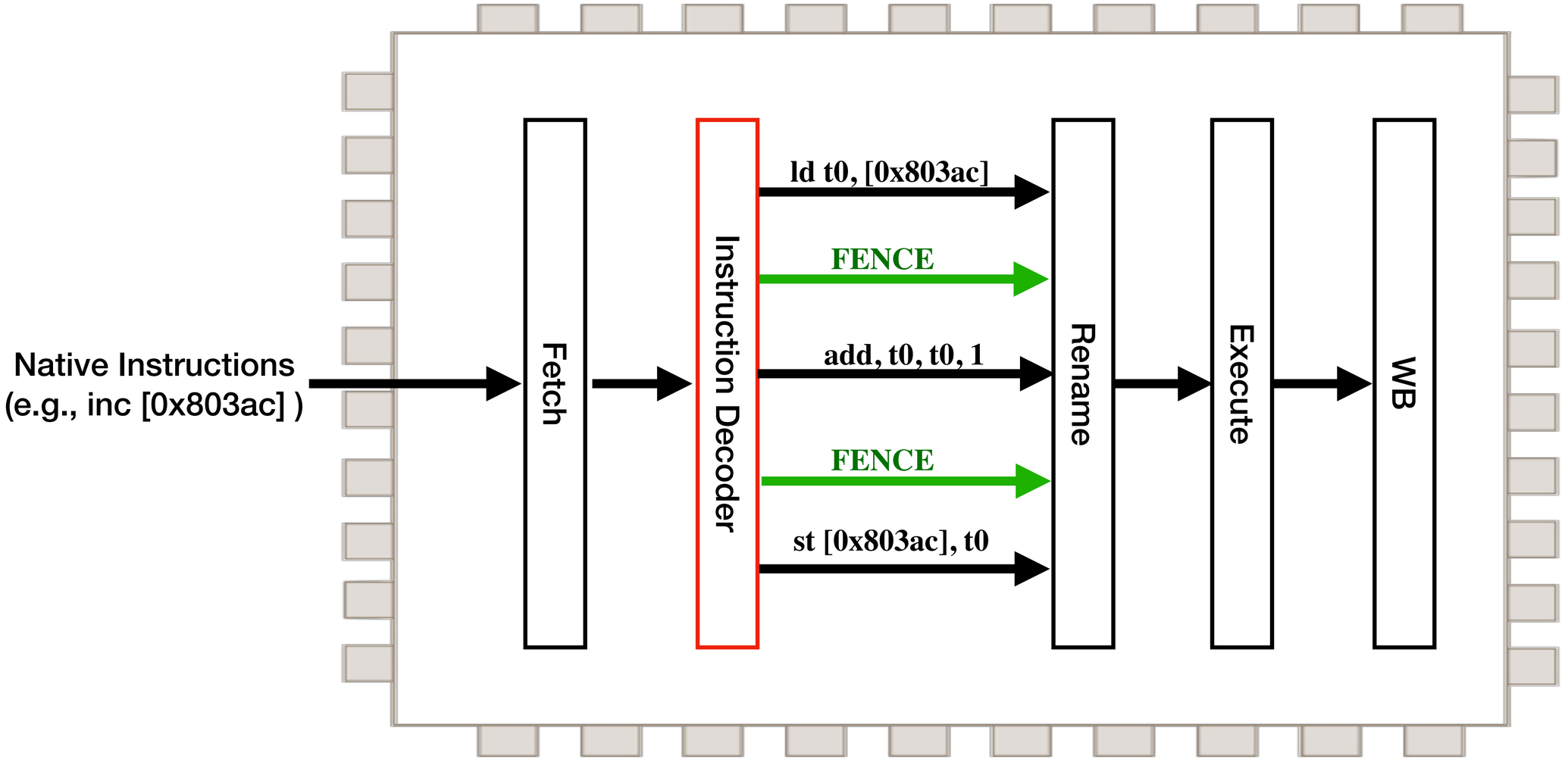
No Recompilation

MICRO-OP STREAM CUSTOMIZATION BY CONTEXT-SENSITIVE DECODING



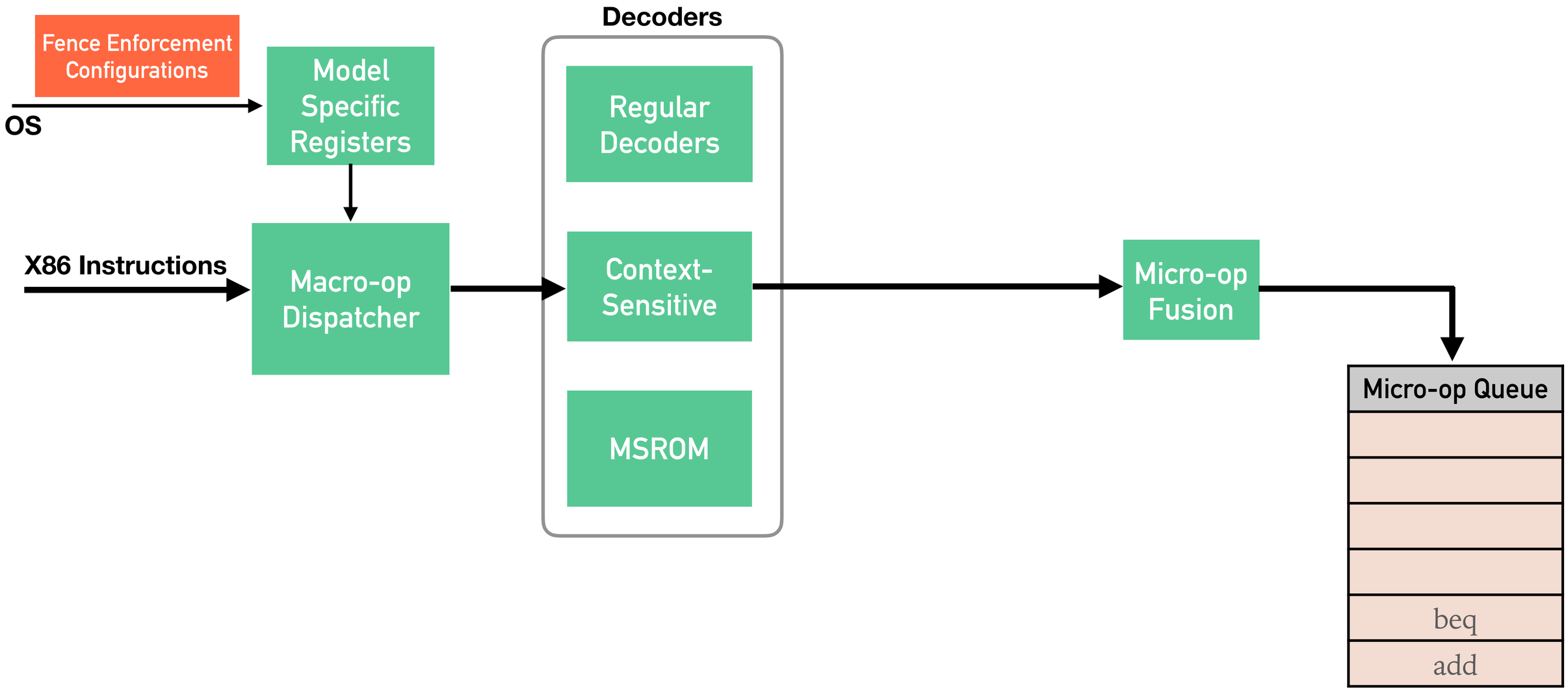
"Context-Sensitive Decoding: On-Demand Microcode Customization for Security and Energy Management"
ISCA 2018, IEEE Micro Top Picks 2019

MICRO-OP STREAM CUSTOMIZATION BY CONTEXT-SENSITIVE DECODING

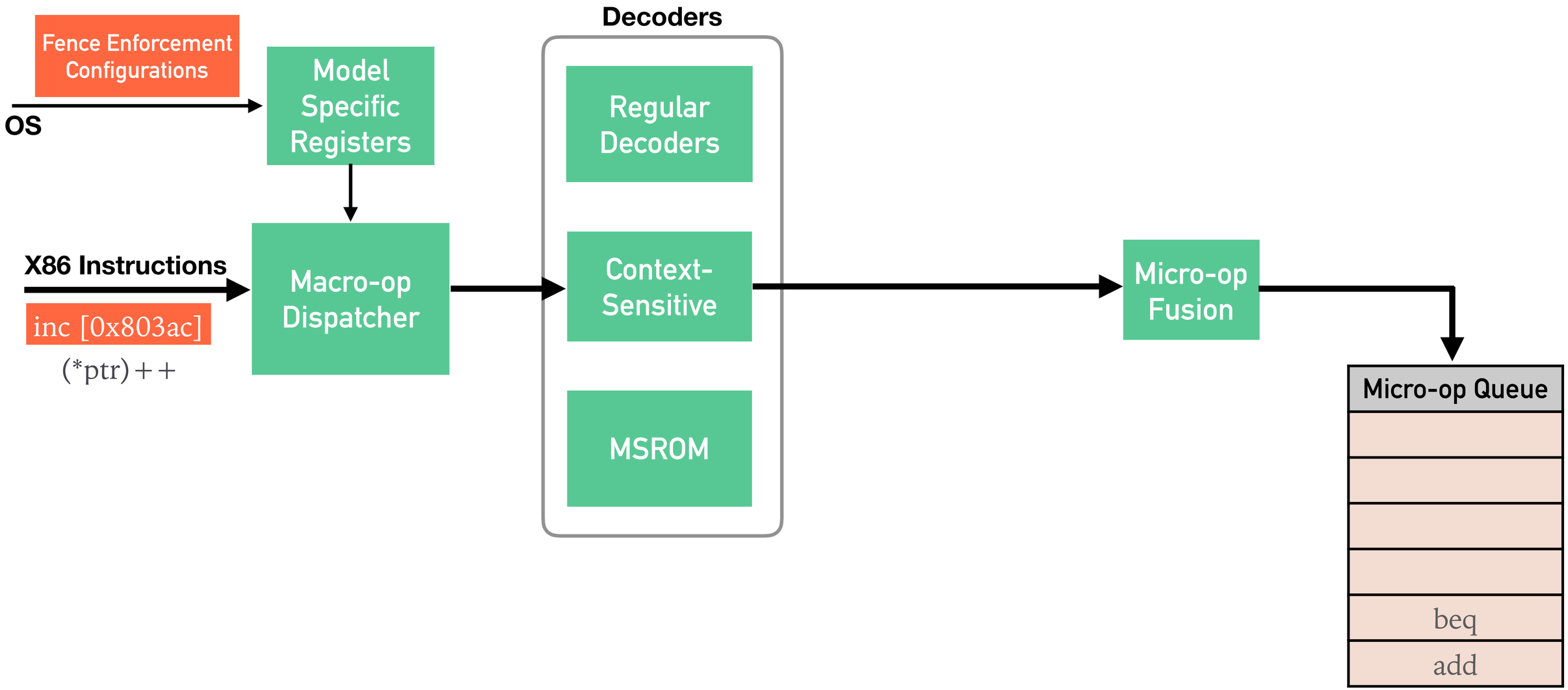


"Context-Sensitive Decoding: On-Demand Microcode Customization for Security and Energy Management"
ISCA 2018, IEEE Micro Top Picks 2019

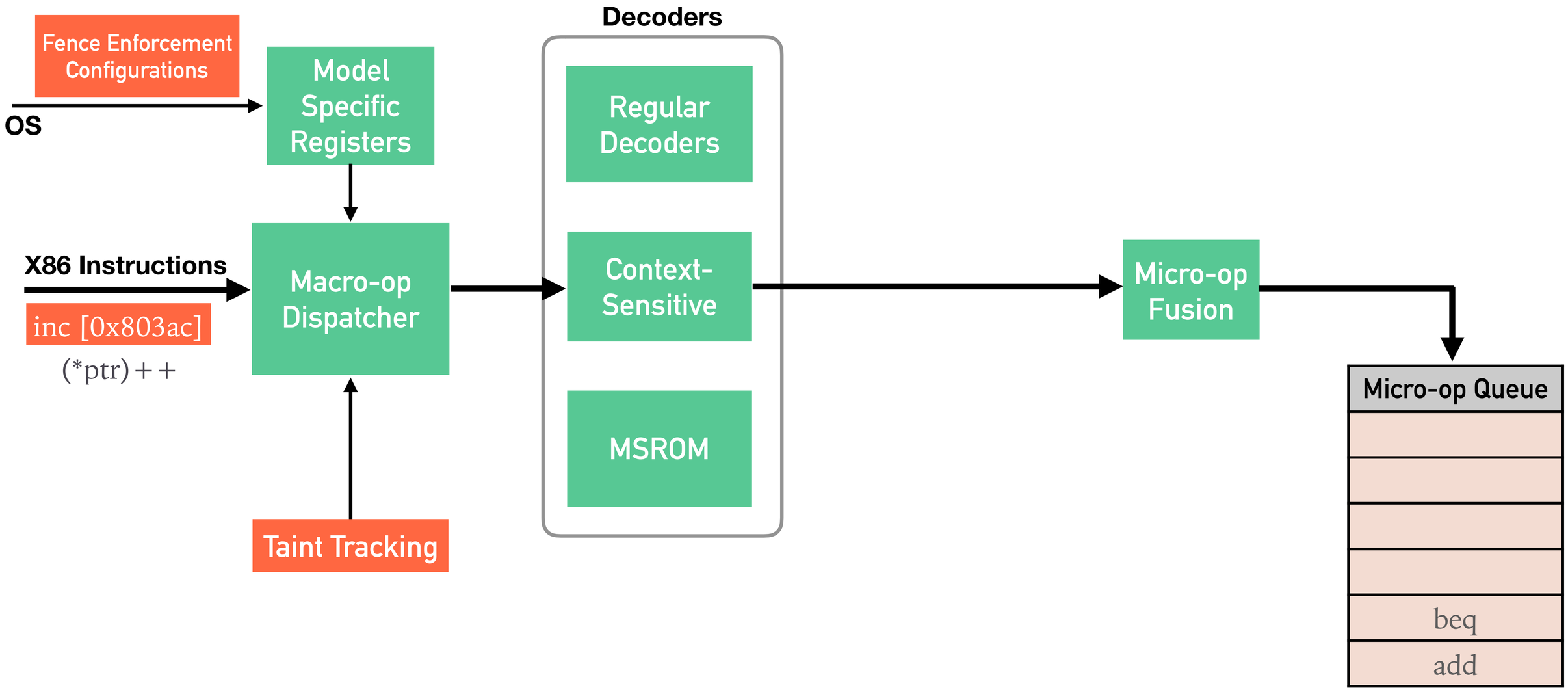
CONTEXT-SENSITIVE FENCING: AN EXAMPLE



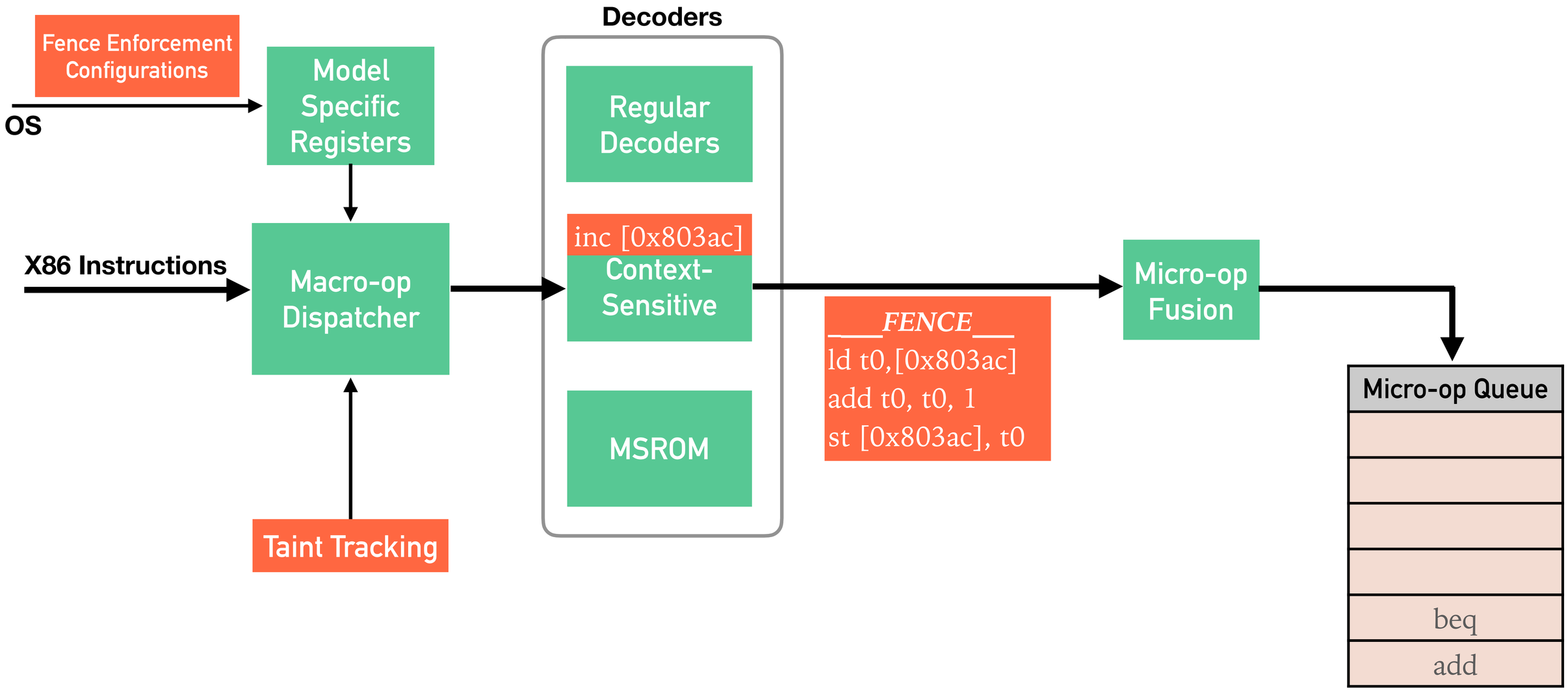
CONTEXT-SENSITIVE FENCING: AN EXAMPLE



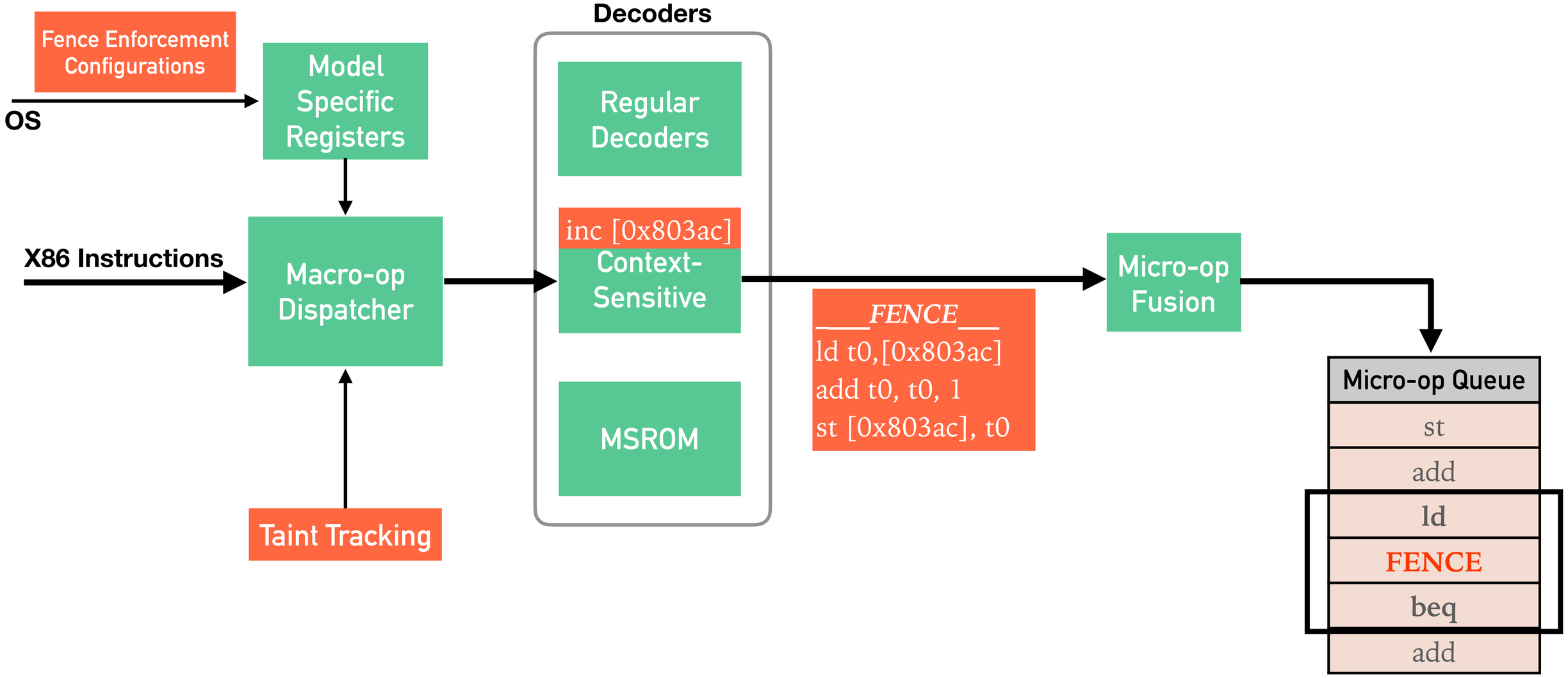
CONTEXT-SENSITIVE FENCING: AN EXAMPLE



CONTEXT-SENSITIVE FENCING: AN EXAMPLE



CONTEXT-SENSITIVE FENCING: AN EXAMPLE



CONTEXT SENSITIVE FENCING

➤ **Surgically** injects fence micro-ops

No Recompilation

Right Type of Fence

Only When Necessary

BUT WHAT FENCE SHOULD WE USE?

Existing Intel Fences

Type of Fence	Instruction Opcode	Description
Privileged Serializing Instructions	INVD	Invalidate Internal Caches
	INVEPT	Invalidate Translations from EPT
	INVLPG	Invalidate TLB Entries
	INVVPID	Invalidate Translations Based on VPID
	LIDT	Load Interrupt Descriptor Table Register
	LGDT	Load Global Descriptor Table Register
	LLDT	Load Local Descriptor Table Register
	LTR	Load Task Register
	MOV	Move to Control Register
	MOV	Move to Debug Register
	WBINVD	Write Back and Invalidate Cache
	WRMSR	Write to Model Specific Register
Non-Privileged Serializing Instructions	CPUID	CPU Identification
	IRET	Interrupt Return
	RSM	Resume from System Management Mode
Memory Ordering Instructions	SFENCE	Store Fence
	LFENCE	Load Fence
	MFENCE	Memory Fence

BUT WHAT FENCE SHOULD WE USE?

Existing Intel Fences

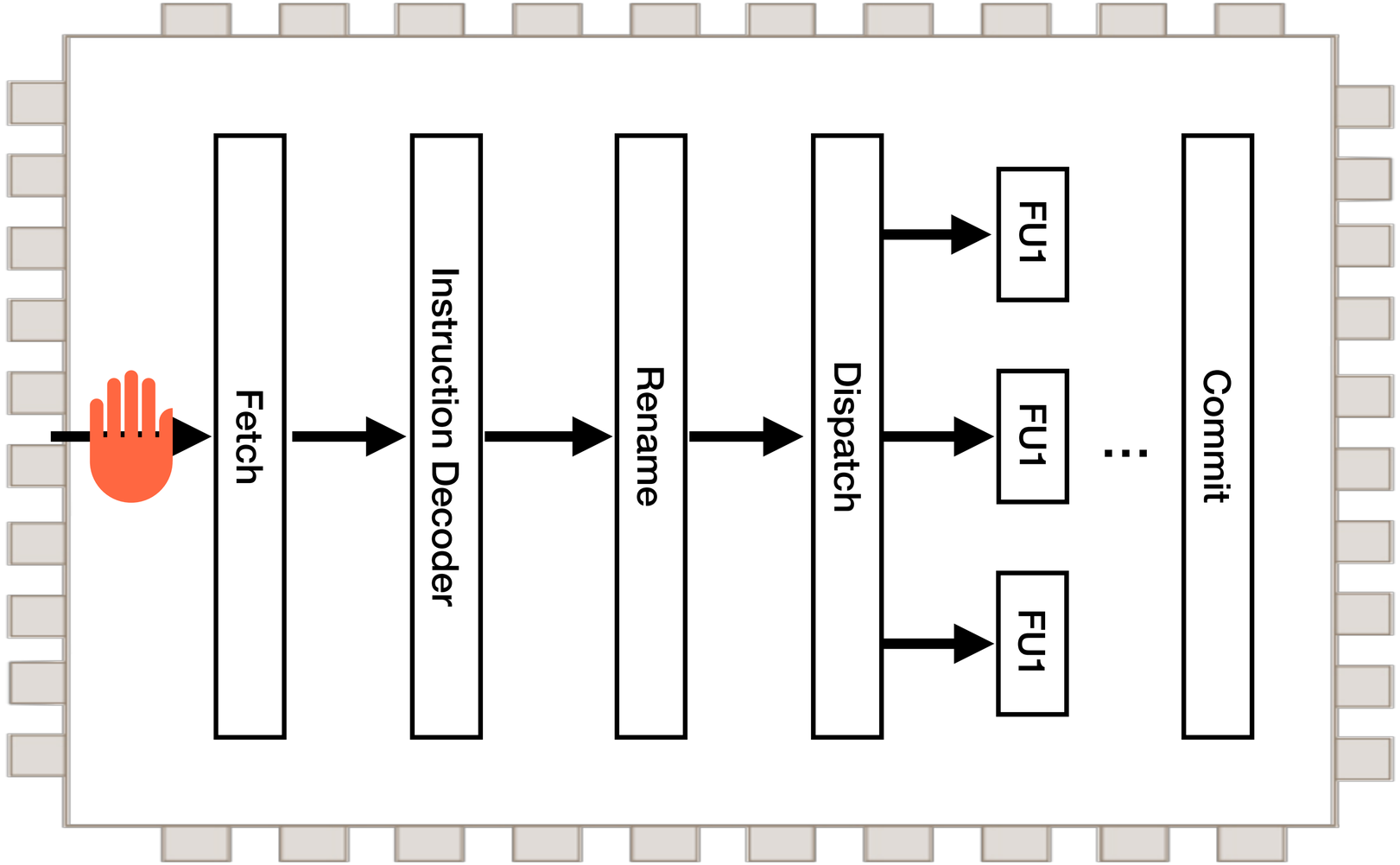
Type of Fence	Instruction Opcode	Description
Privileged Serializing Instructions	INVD	Invalidate Internal Caches
	INVEPT	Invalidate Translations from EPT
	INVLPG	Invalidate TLB Entries
	INVVPID	Invalidate Translations Based on VPID
	LIDT	Load Interrupt Descriptor Table Register
	LGDT	Load Global Descriptor Table Register
	LLDT	Load Local Descriptor Table Register
	LTR	Load Task Register
	MOV	Move to Debug Register
	WBINVD	Write Back and Invalidate Cache
	WRMSR	Write to Model Specific Register
Non-Privileged Serializing Instructions	CPUID	CPU Identification
	IRET	Interrupt Return
	RSM	Resume from System Management Mode
Memory Ordering Instructions	SFENCE	Store Fence
	LFENCE	Load Fence
	MFENCE	Memory Fence

Require Privileged Access

Clobber Architectural Registers

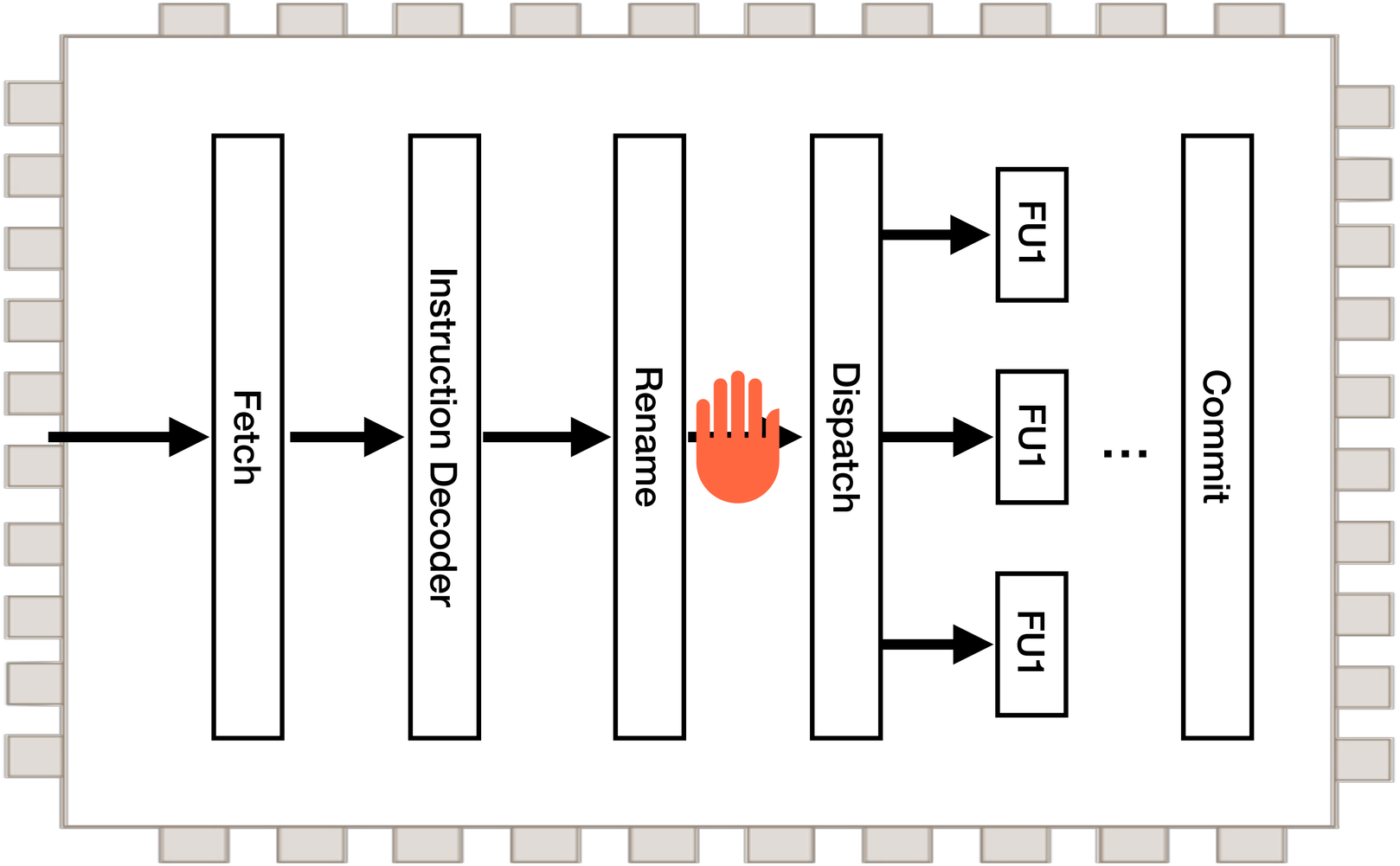
Enforced Early in the Pipeline

EXISTING FENCES: SERIALIZING INSTRUCTIONS (SI)



- Enforced early in the pipeline
- Examples:
  - All Serializing Instructions
  - Intel's MFENCE
  - Intel's SFENCE

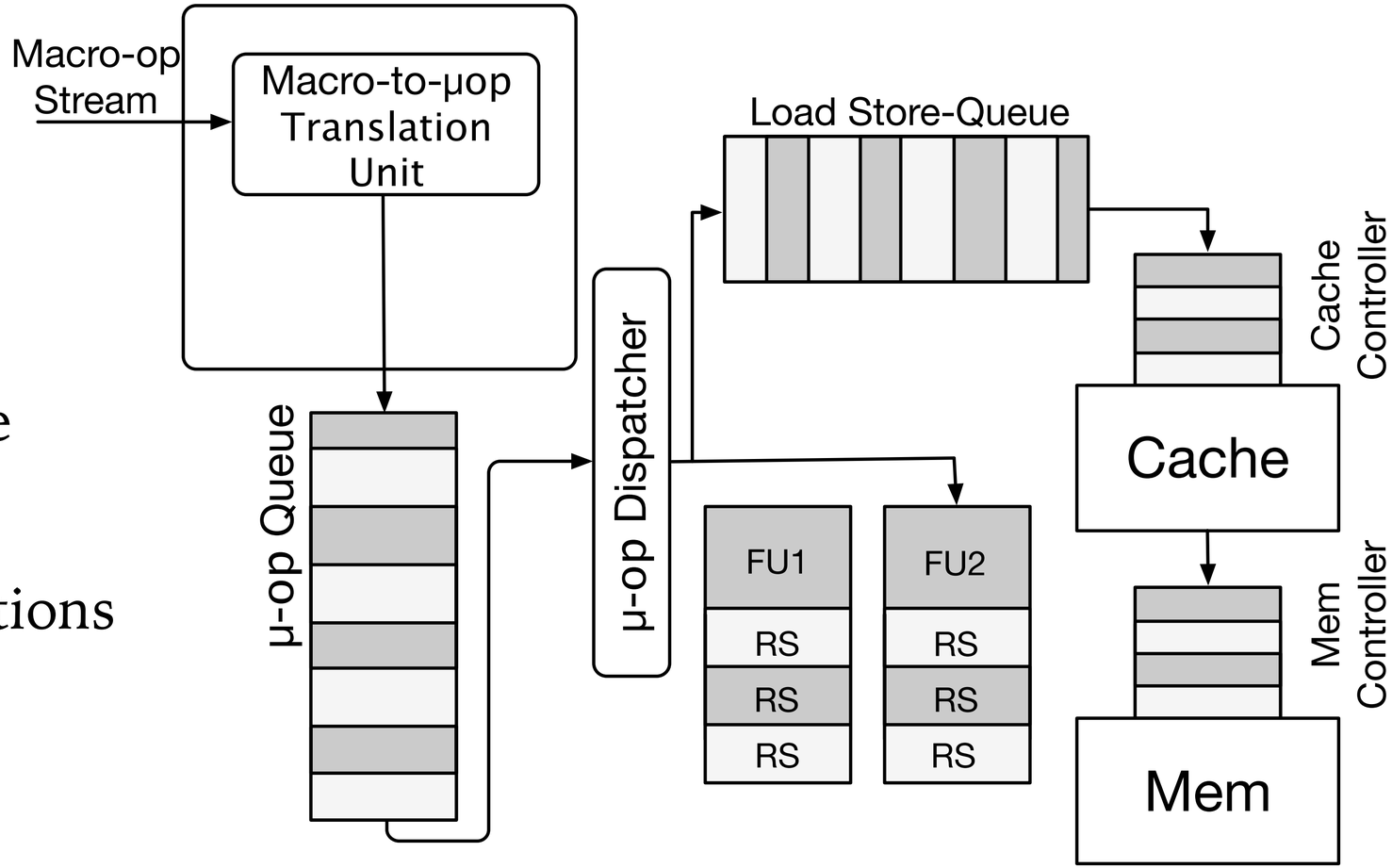
EXISTING FENCES: INTEL LFENCE



- Enforced early in the pipeline
- Example:
  - Intel's LFENCE

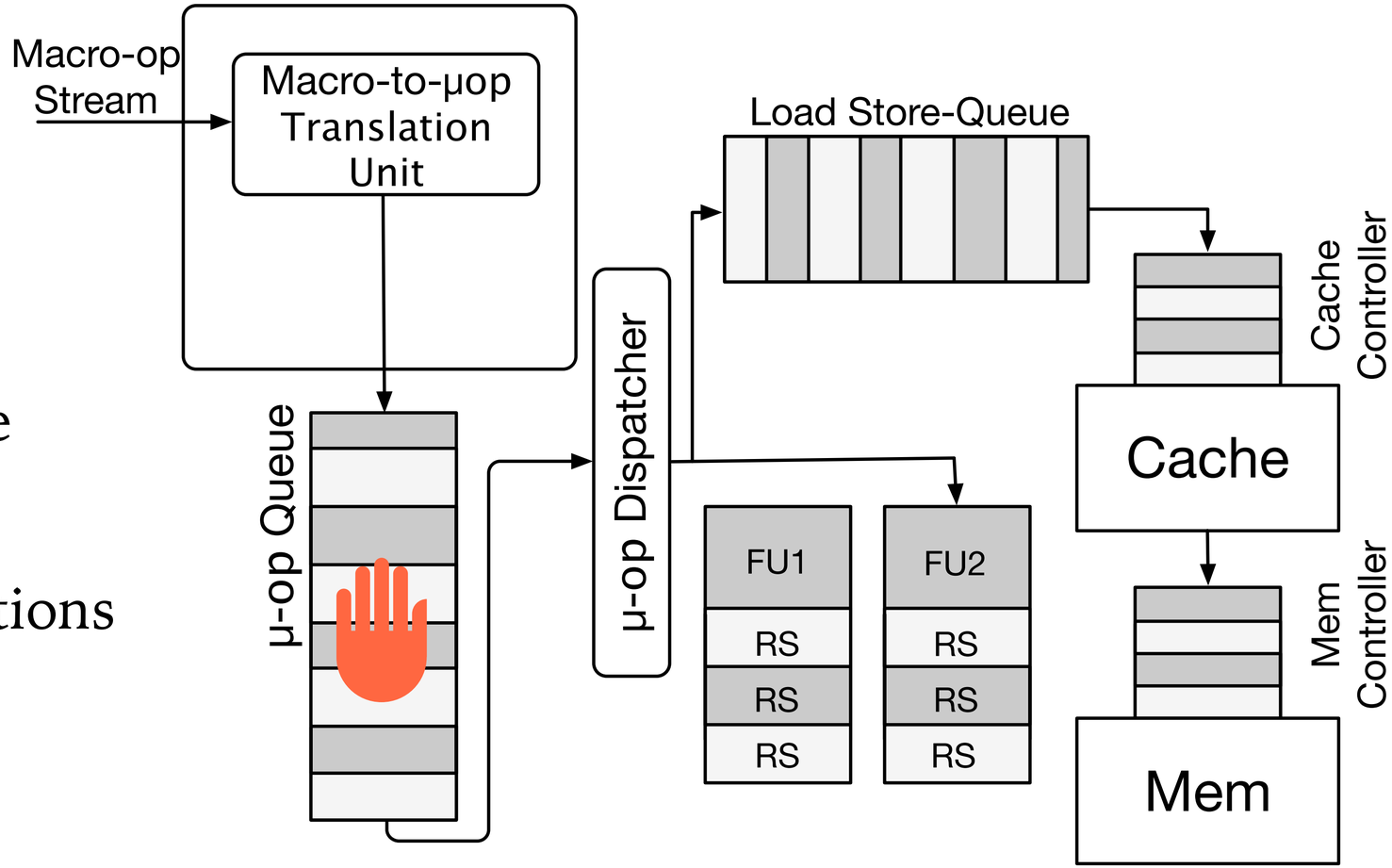
LATE ENFORCEMENT FENCES

- Shifts fence enforcement towards the leaking structure
- Reduces the impact on other instructions



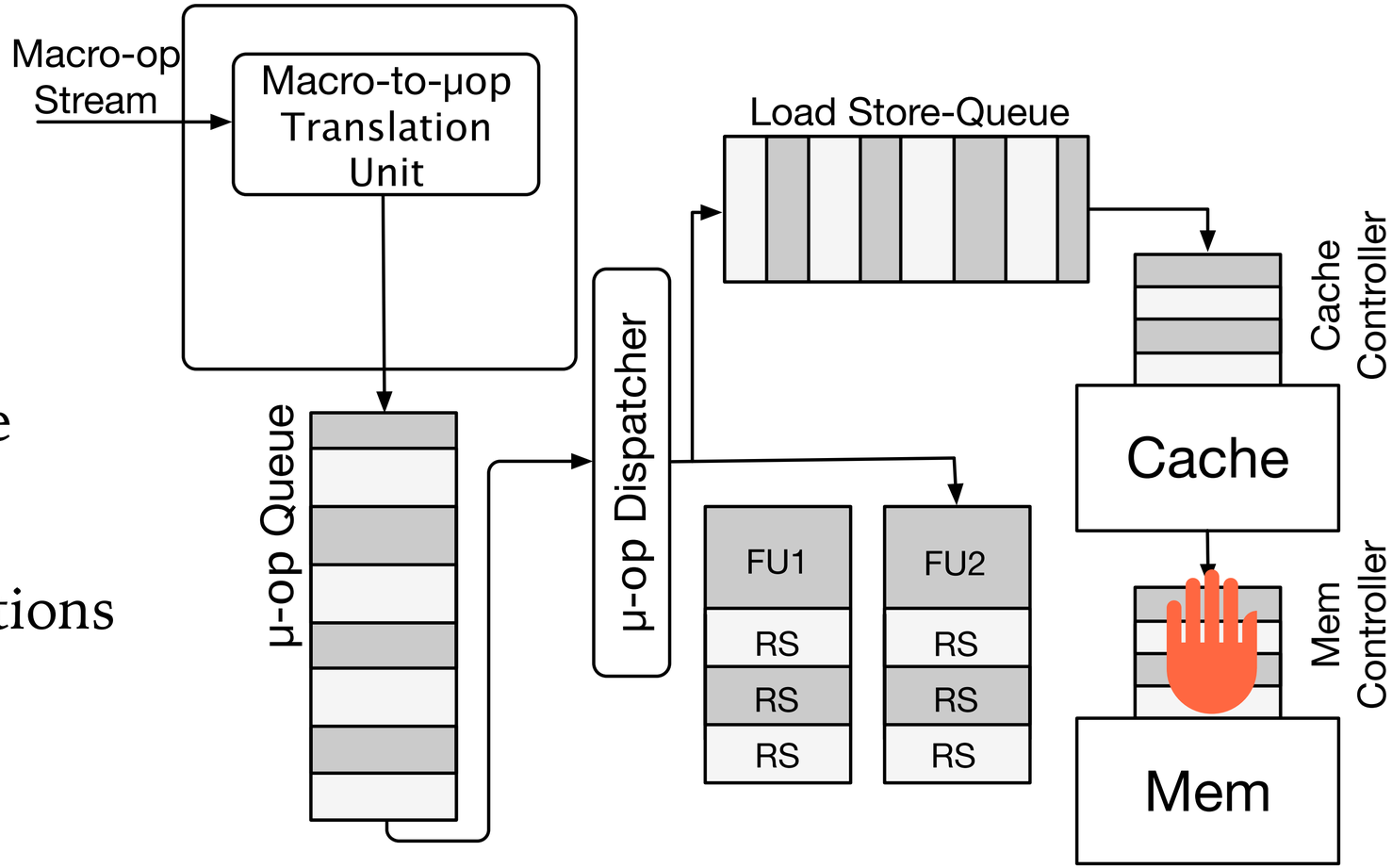
LATE ENFORCEMENT FENCES

- Shifts fence enforcement towards the leaking structure
- Reduces the impact on other instructions



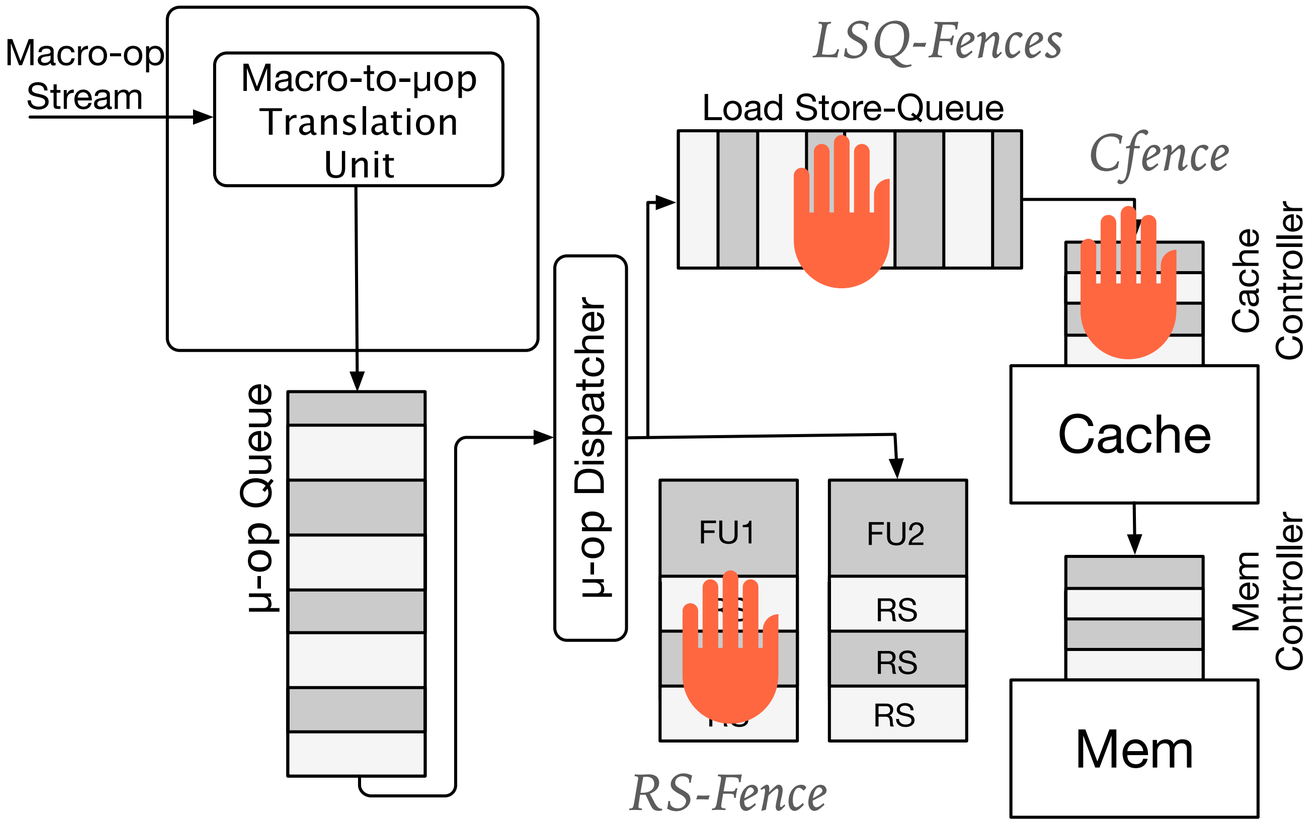
LATE ENFORCEMENT FENCES

- Shifts fence enforcement towards the leaking structure
- Reduces the impact on other instructions



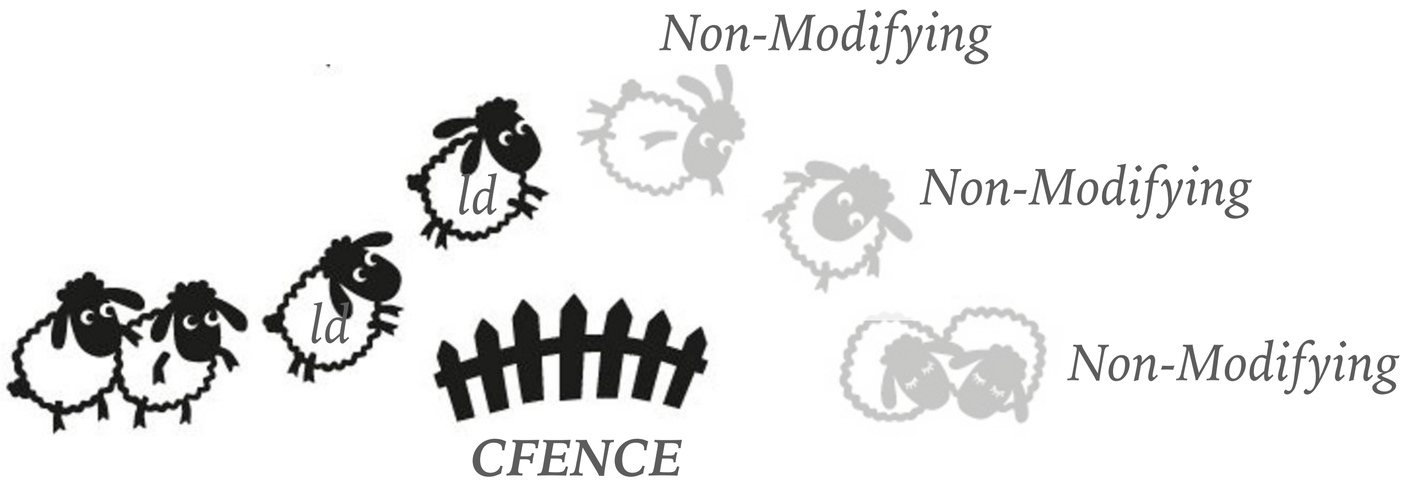
NEWLY PROPOSED FENCES

- Load-Store Queue LFENCE (LSQ-LFENCE)
- Load-Store Queue MFENCE (LSQ-MFENCE)
- Reservation Station Fence (RSFENCE)
- Cache Fence (CFENCE)

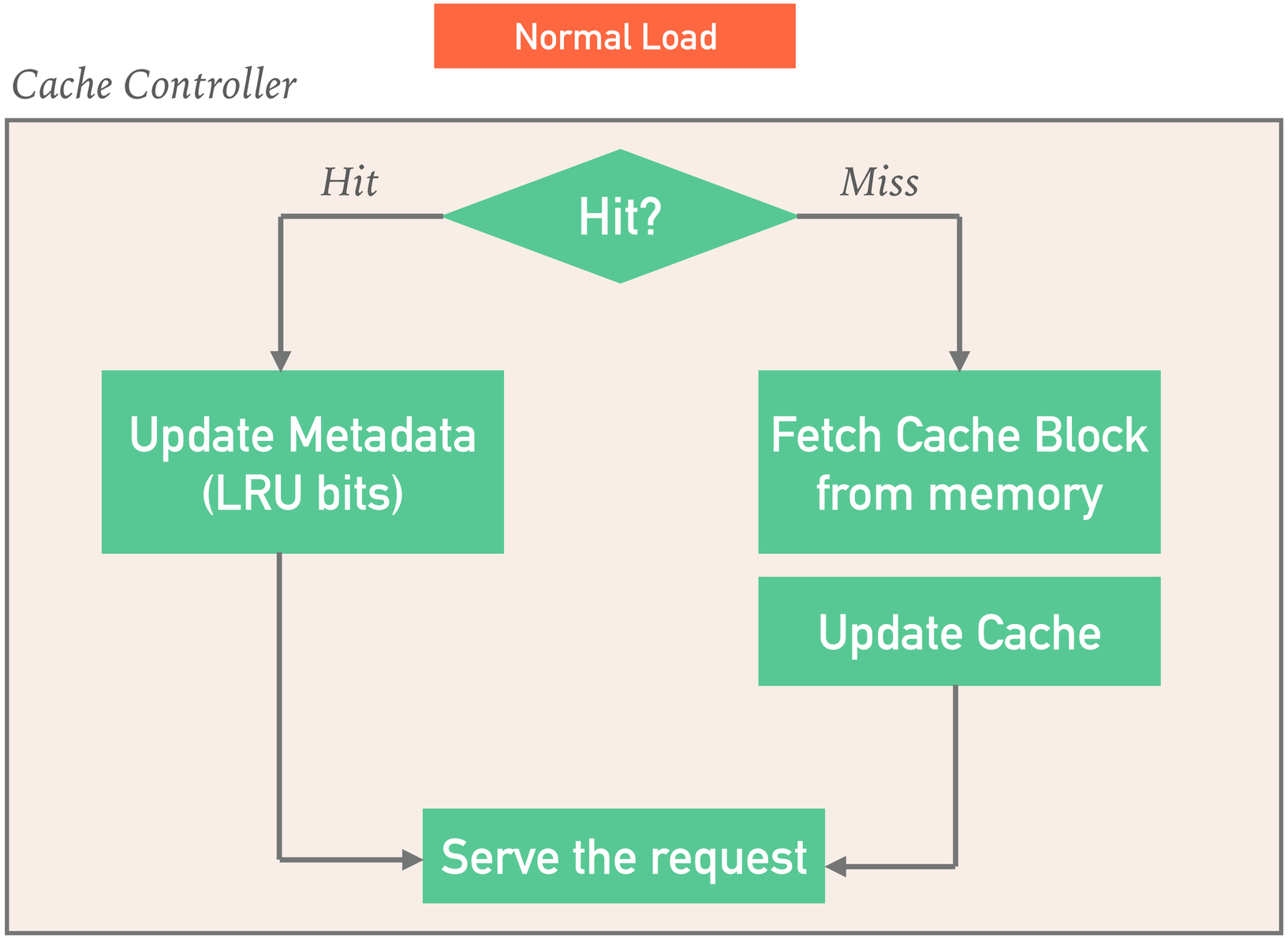


CACHE FENCE (CFENCE)

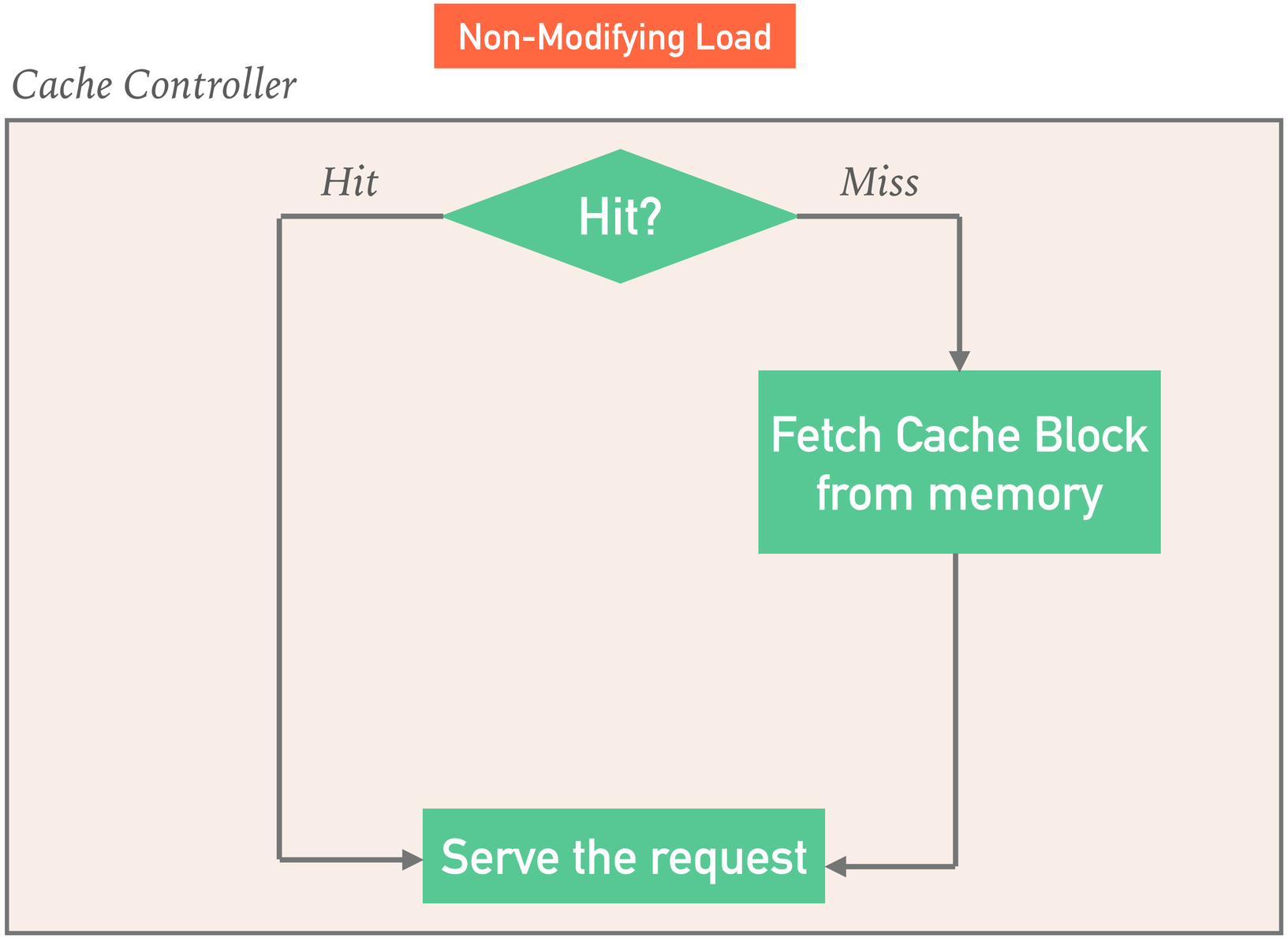
- Allows all the load and stores to pass
- *CFENCE* labels any subsequent load as a *non-modifying load*
- allows *non-modifying loads* to pass through the *CFENCE*
- Non-modifying loads are restricted from modifying the cache state.



CACHE FENCE (CFENCE)

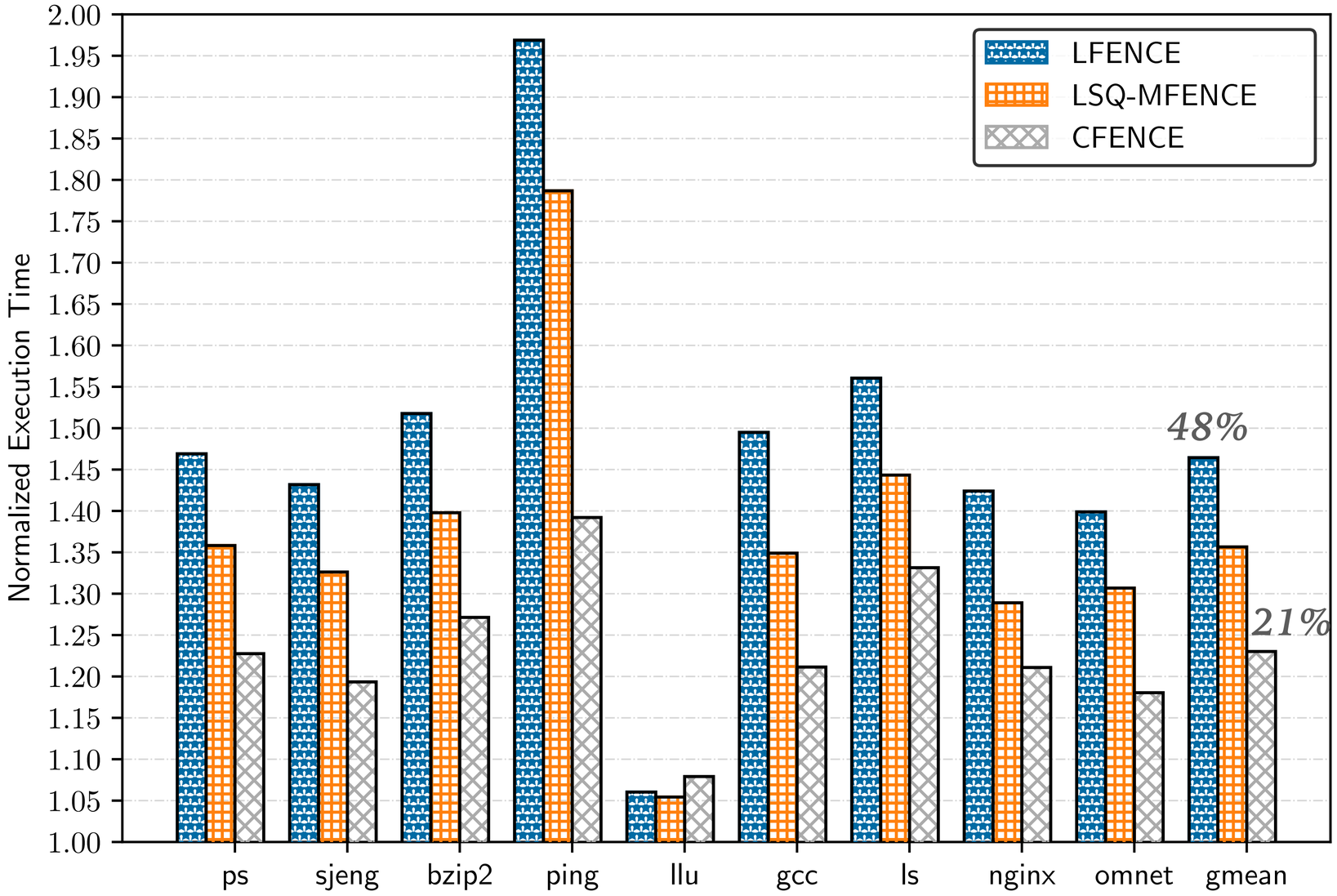


CACHE FENCE (CFENCE)



RESULTS — FENCE ENFORCEMENT POLICIES

- Our CFENCE reduces the incurred performance overhead by **2.3X**, bringing down the execution time overhead from 48% to 21%.



CONTEXT SENSITIVE FENCING

➤ **Surgically** injects fence micro-ops

No Recompilation

Right Type of Fence

Only When Necessary

FENCE FREQUENCY OPTIMIZATIONS

- Liberal Injection
  - Injects fences before all the loads of a program
  - completely stops speculation

```
jeq
ld
add
ld
ld
```

FENCE FREQUENCY OPTIMIZATIONS

- Liberal Injection
  - Injects fences before all the loads of a program
  - completely stops speculation

```
jeq
Fence
ld
add
Fence
ld
Fence
ld
```

FENCE FREQUENCY OPTIMIZATIONS

- Basic Block-Level Fence Insertion*
  - Speculation begins with a branch prediction
  - A fence between branch and subsequent loads

```
jeq
Fence
ld
add
Fence
ld
Fence
ld
```

** Targeted Optimization — Only protects against variant 1*

FENCE FREQUENCY OPTIMIZATIONS

- Basic Block-Level Fence Insertion
  - Speculation begins with a branch prediction
  - We want a fence between each branch and subsequent loads

```
jeq
Fence
ld
add
ld
ld
```

** Targeted Optimization — Only protects against variant 1*

FENCE FREQUENCY OPTIMIZATIONS

- Taint-Based Fence Insertion
  - Even one fence per basic block is too conservative
  - Attacker performs operations based on untrusted data
    (e.g., attacker controlled out of bound index)
  - Insert fences for only vulnerable loads that operate on untrusted data
  - Dynamic Information Flow Tracker (DIFT)

DLIFT- AN INFORMATION FLOW TRACKER FOR SPECTRE ERA

- Classic Information Flow Trackers
  - Maintain and Evaluate Taints at Late Stages of the Pipeline
  - Not so useful for Spectre!

DLIFT- AN INFORMATION FLOW TRACKER FOR SPECTRE ERA

- Classic Information Flow Trackers
  - Maintain and Evaluate Taints at Late Stages of the Pipeline
  - Not so useful for Spectre!

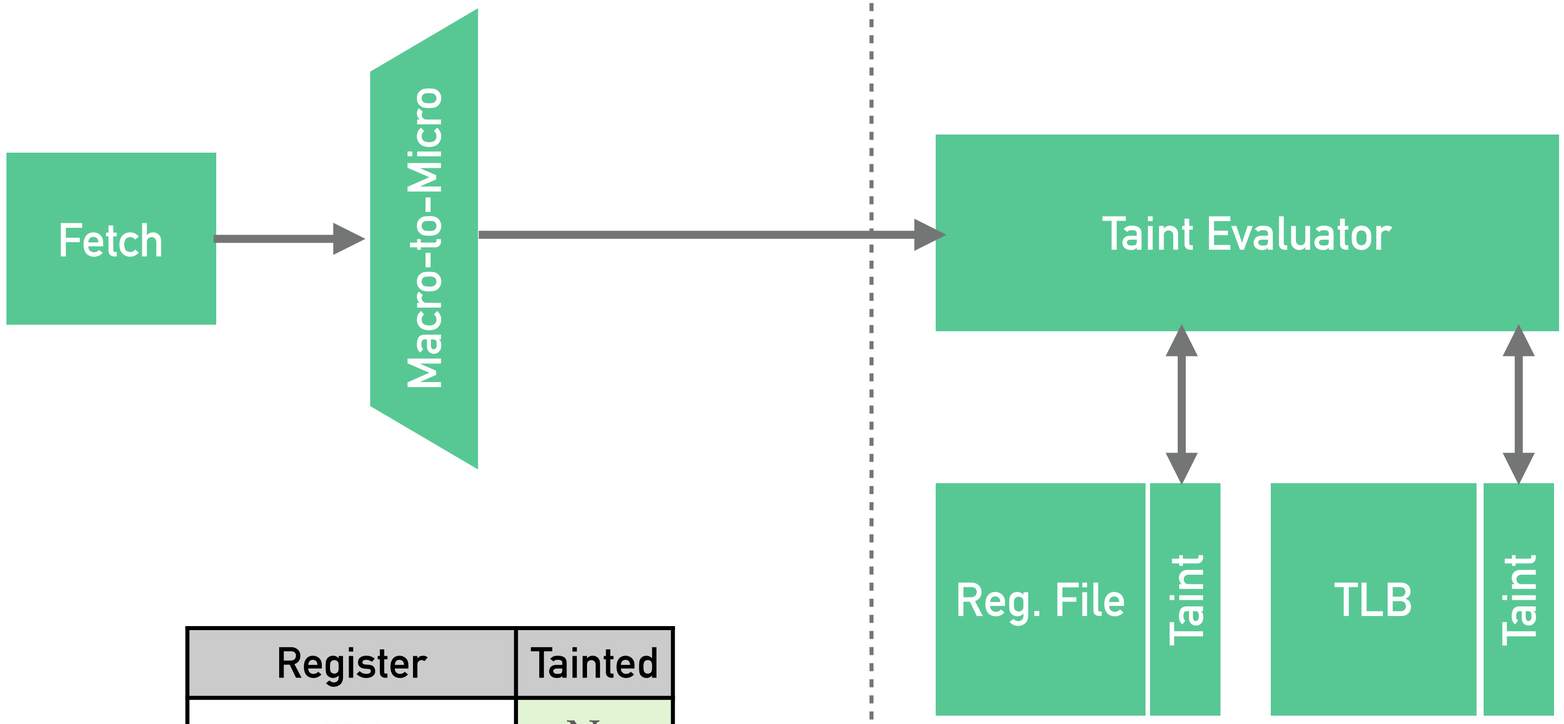
DLIFT- AN INFORMATION FLOW TRACKER FOR SPECTRE ERA

- Classic Information Flow Trackers
  - Maintain and Evaluate Taints at Late Stages of the Pipeline
  - Not so useful for Spectre!

Detect
The
Threat
before
it's too late.

DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

Fetch/Decode | *Execute* | *Commit*



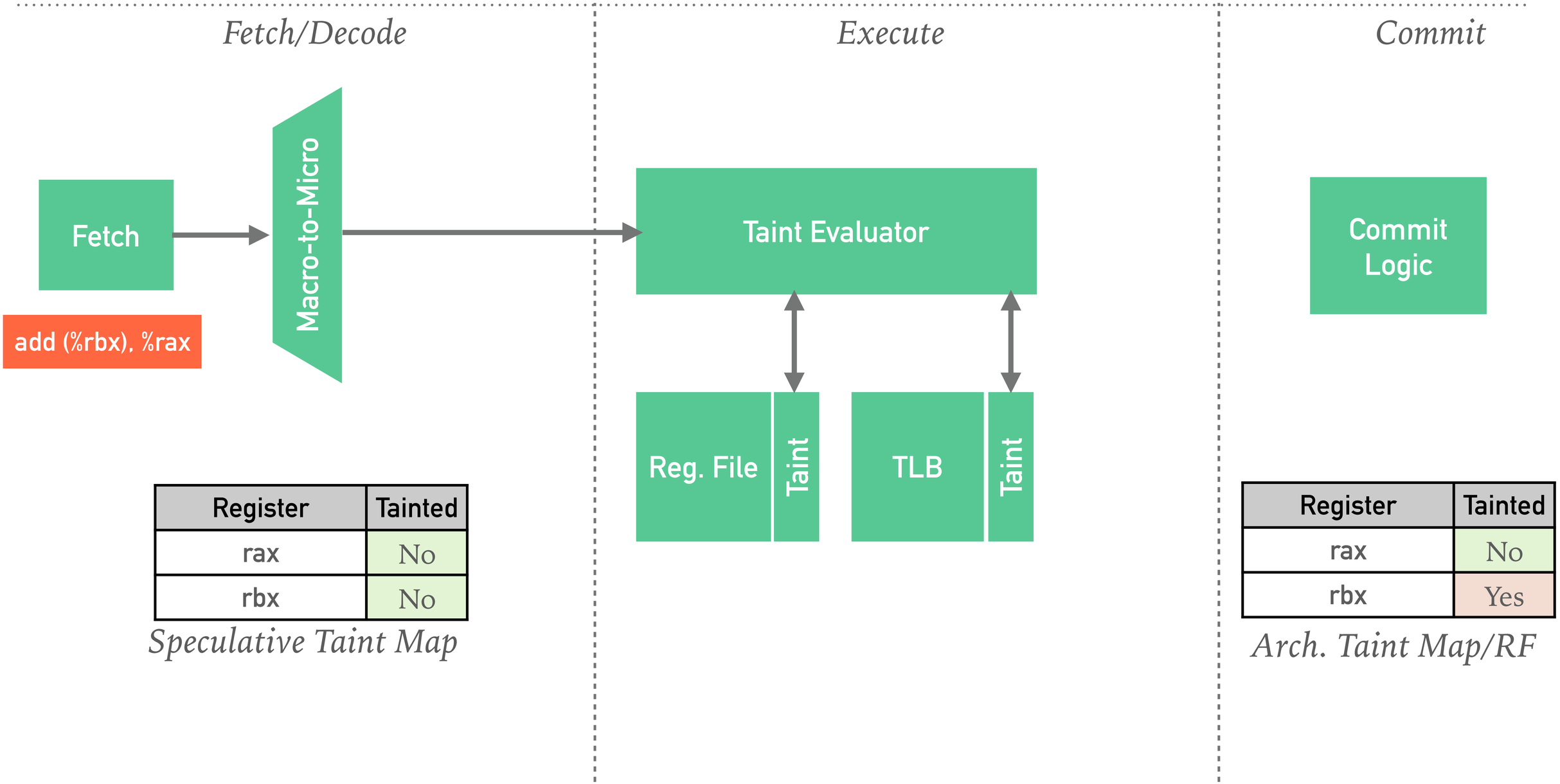
Register	Tainted
rax	No
rbx	No

Speculative Taint Map

Register	Tainted
rax	No
rbx	Yes

Arch. Taint Map/RF

DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER



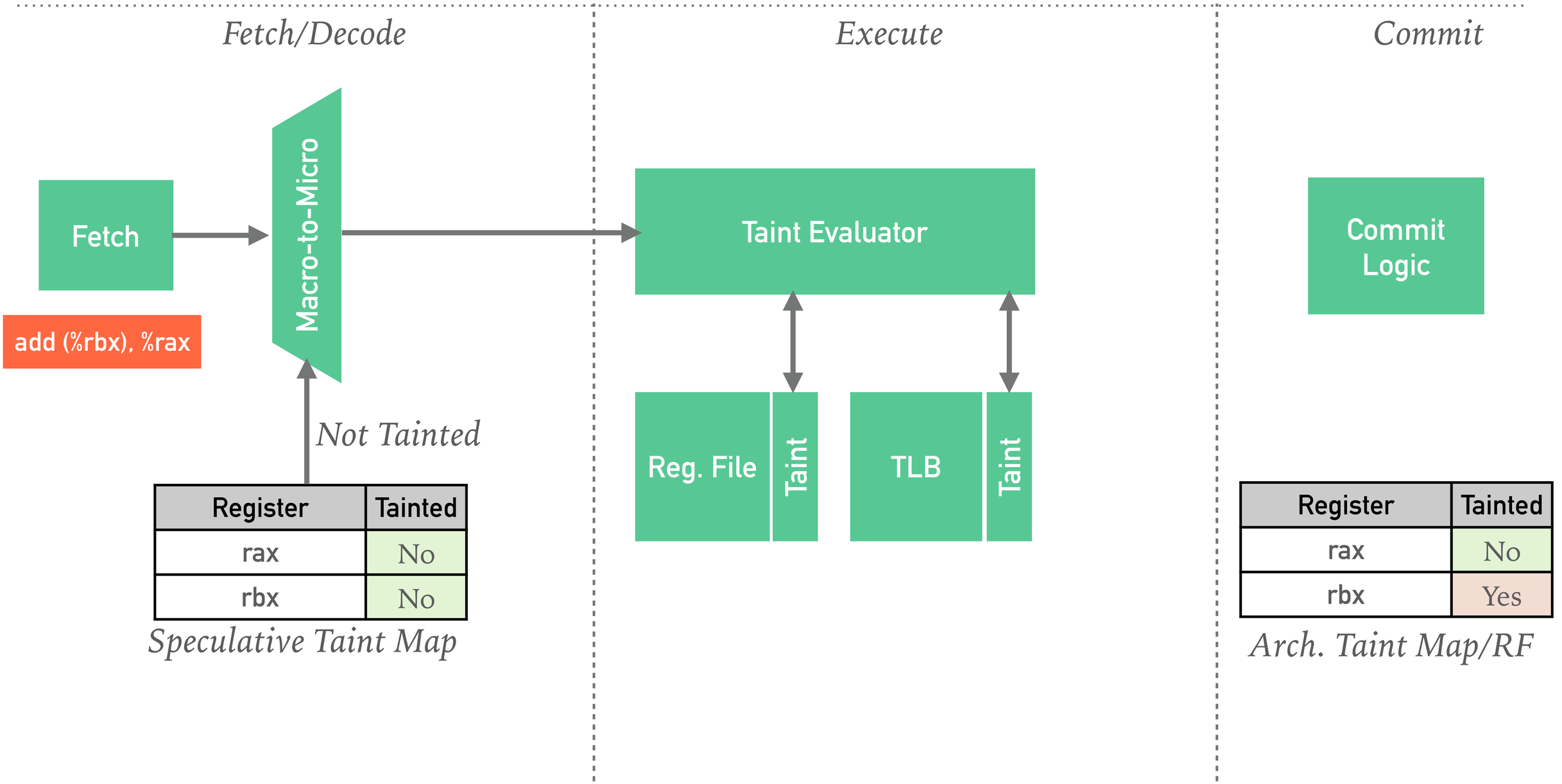
Register	Tainted
rax	No
rbx	No

Speculative Taint Map

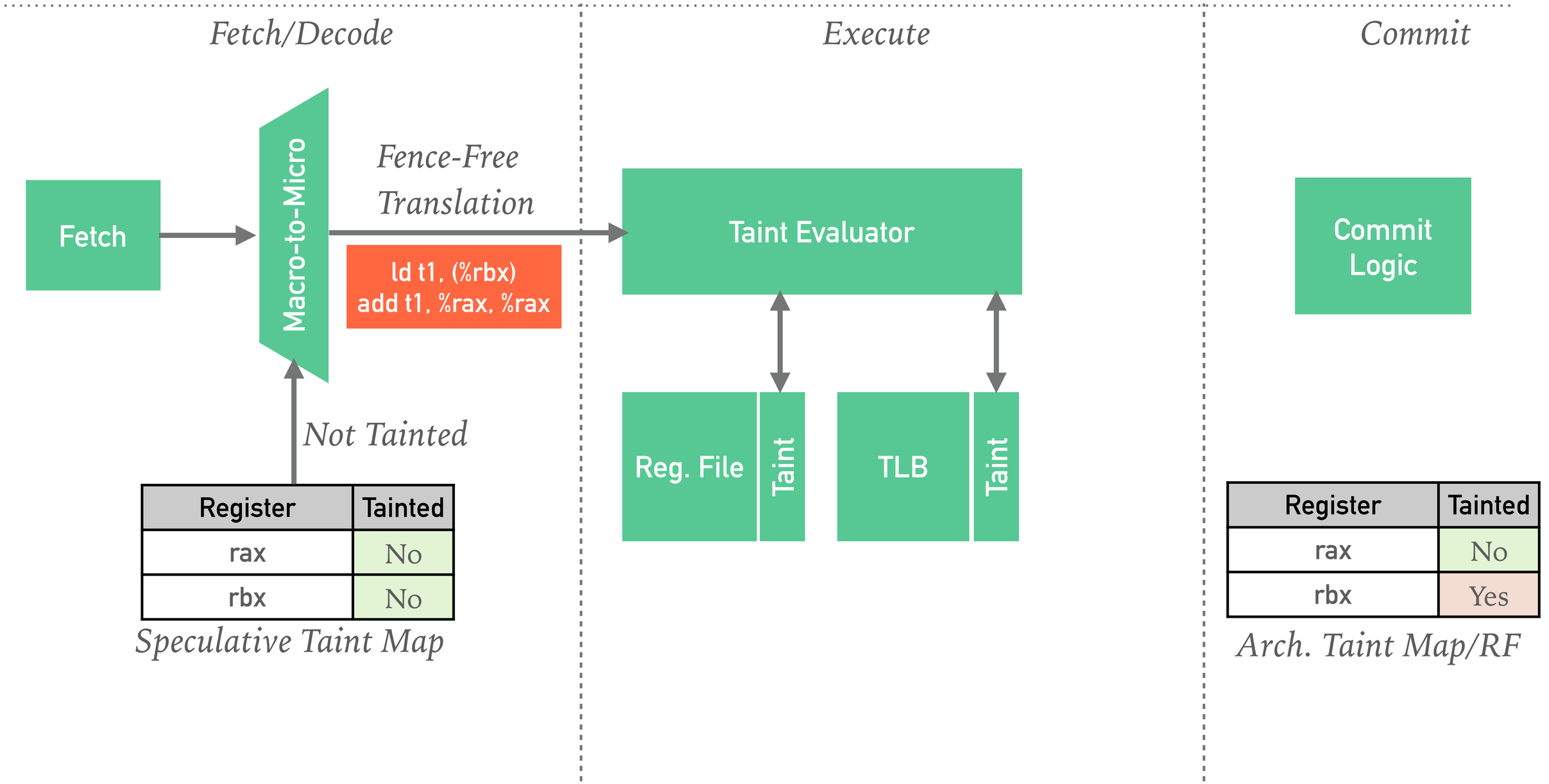
Register	Tainted
rax	No
rbx	Yes

Arch. Taint Map/RF

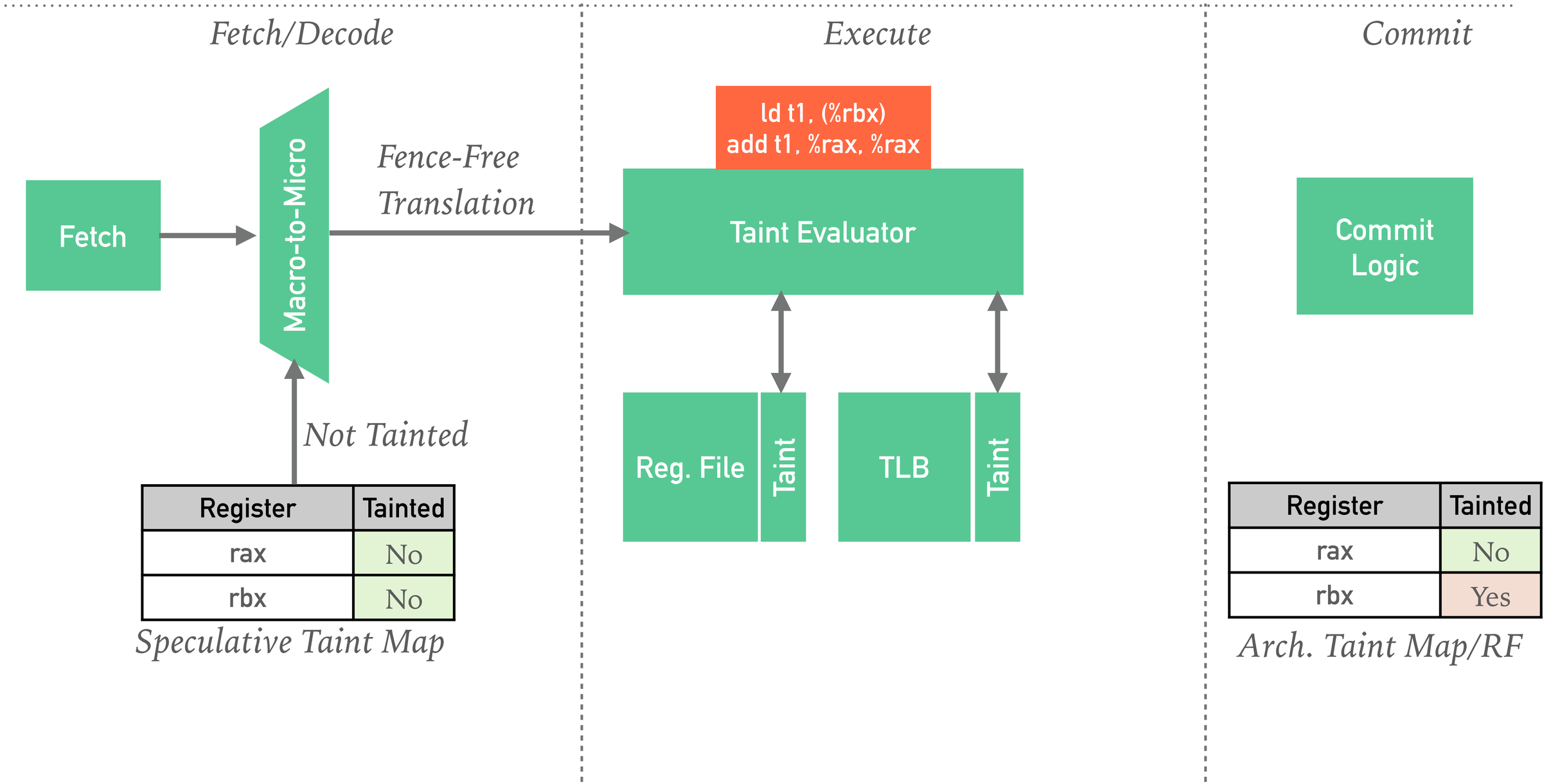
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER



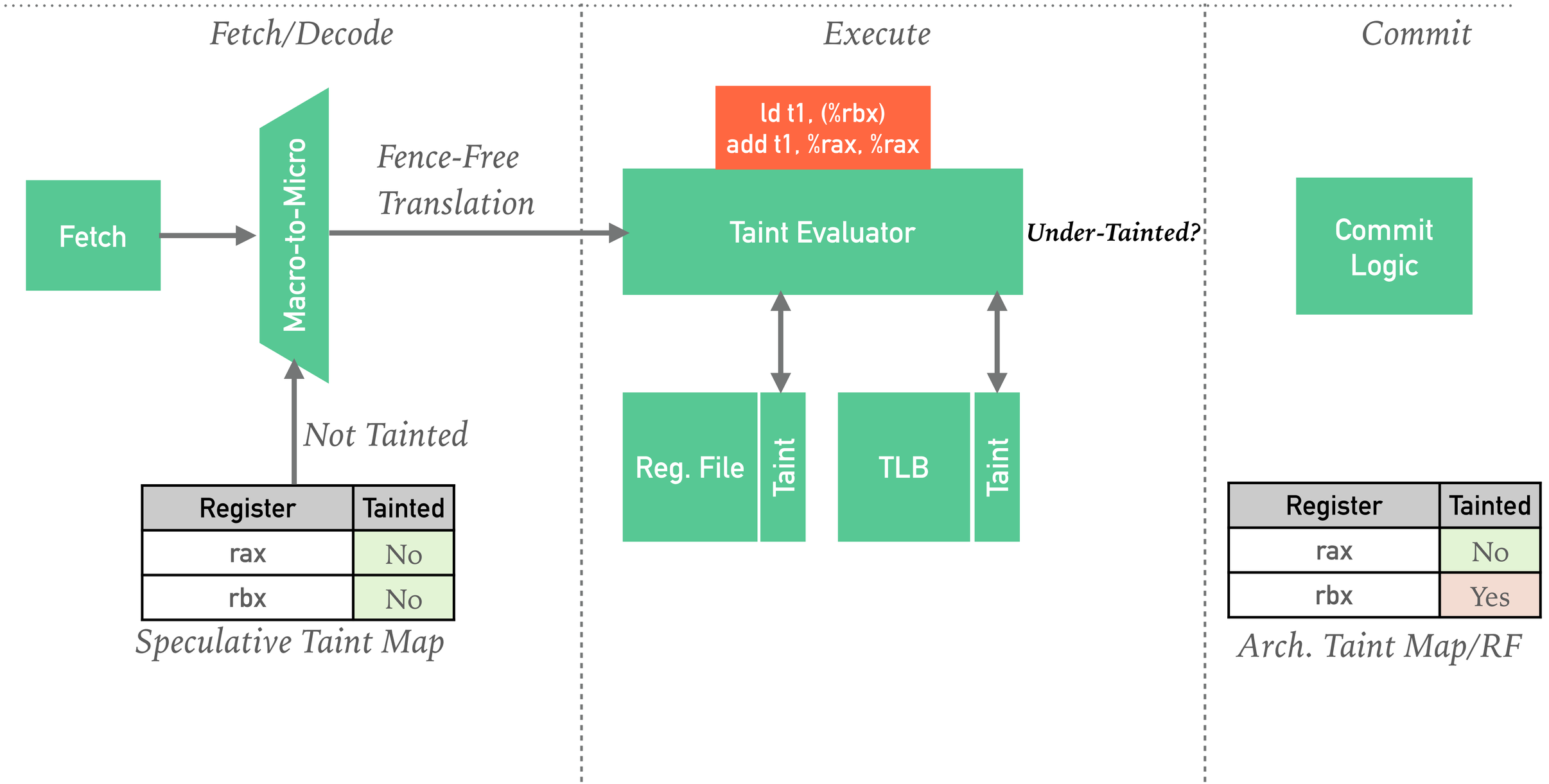
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER



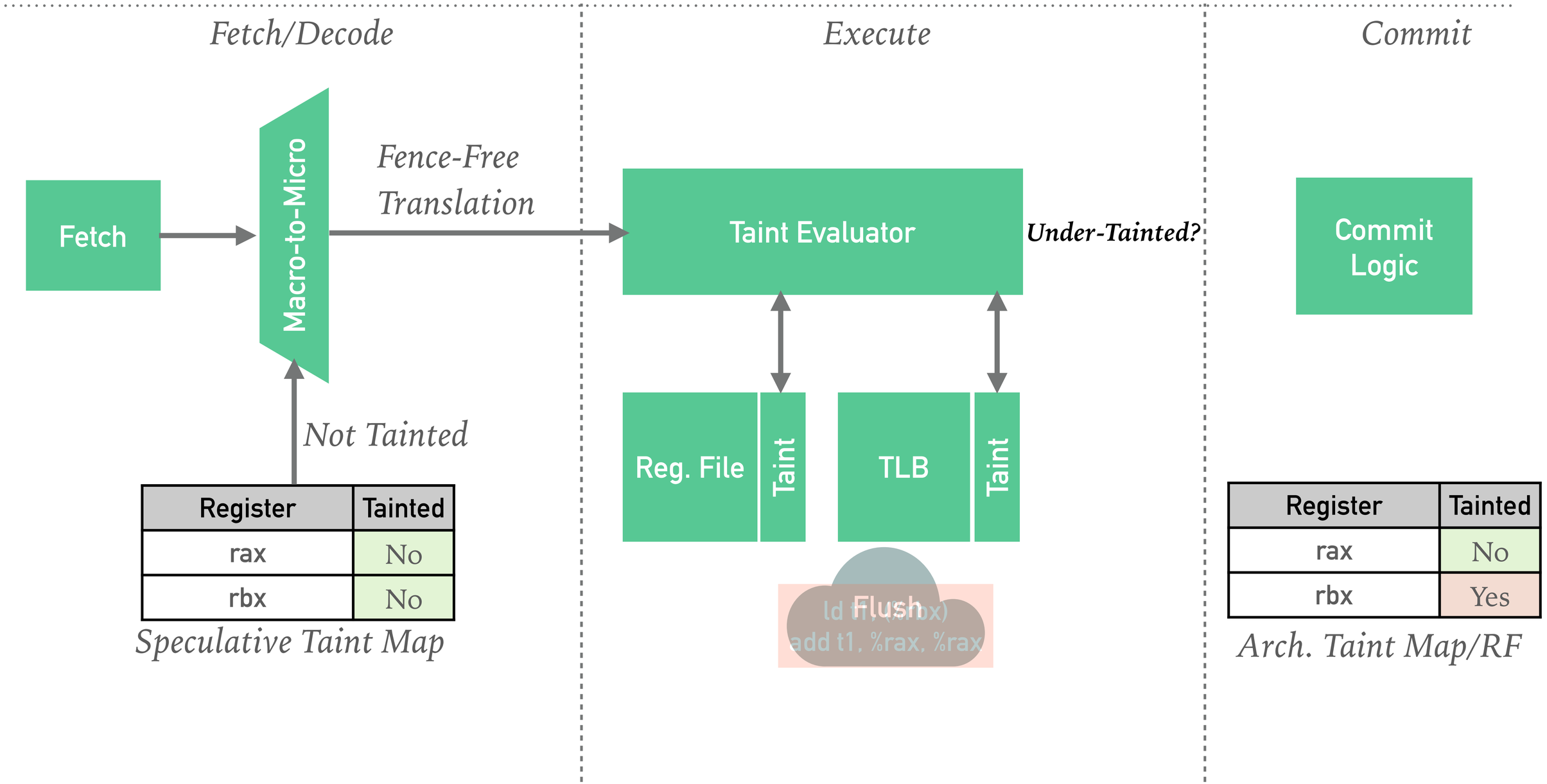
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER



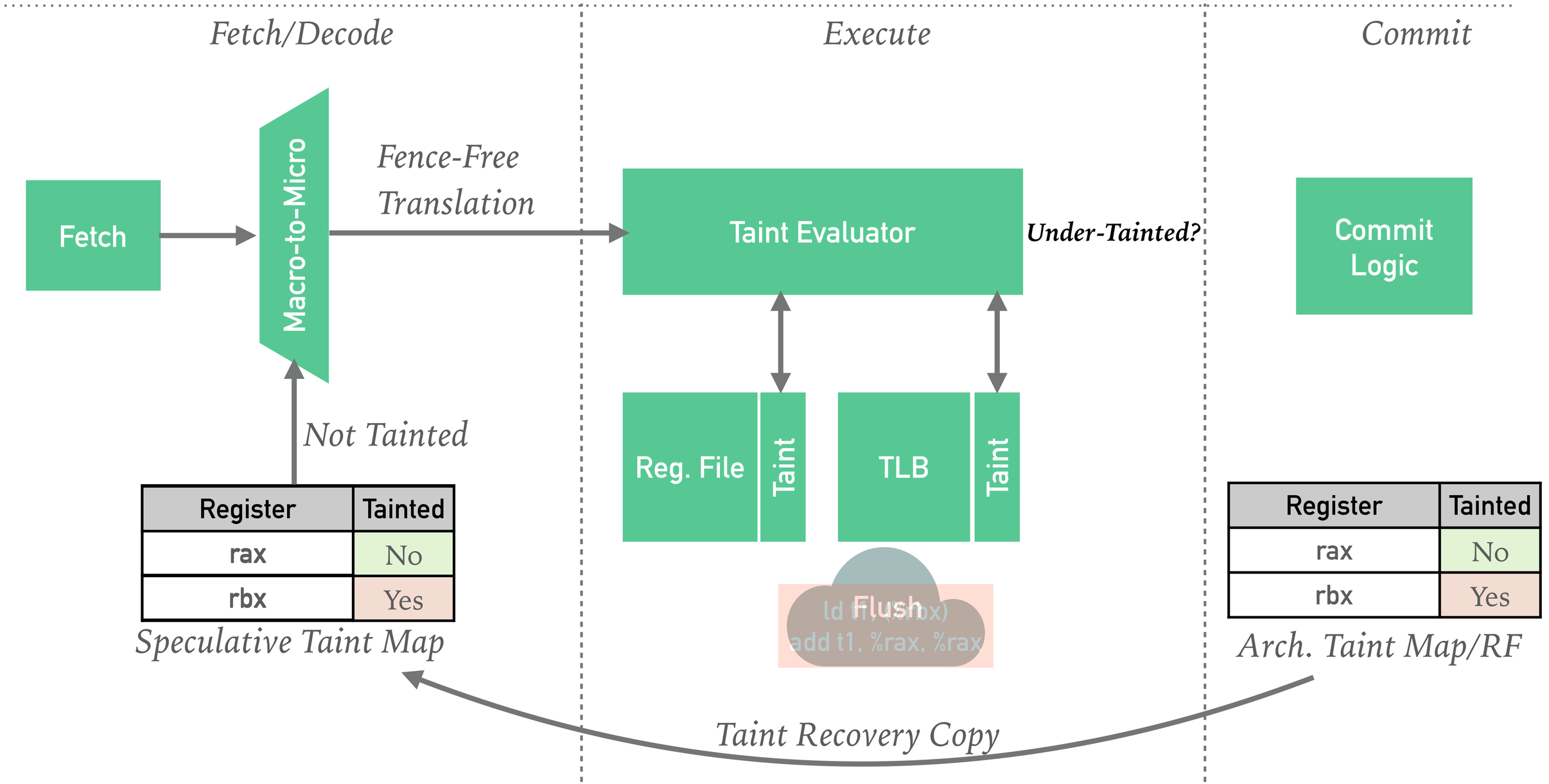
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER



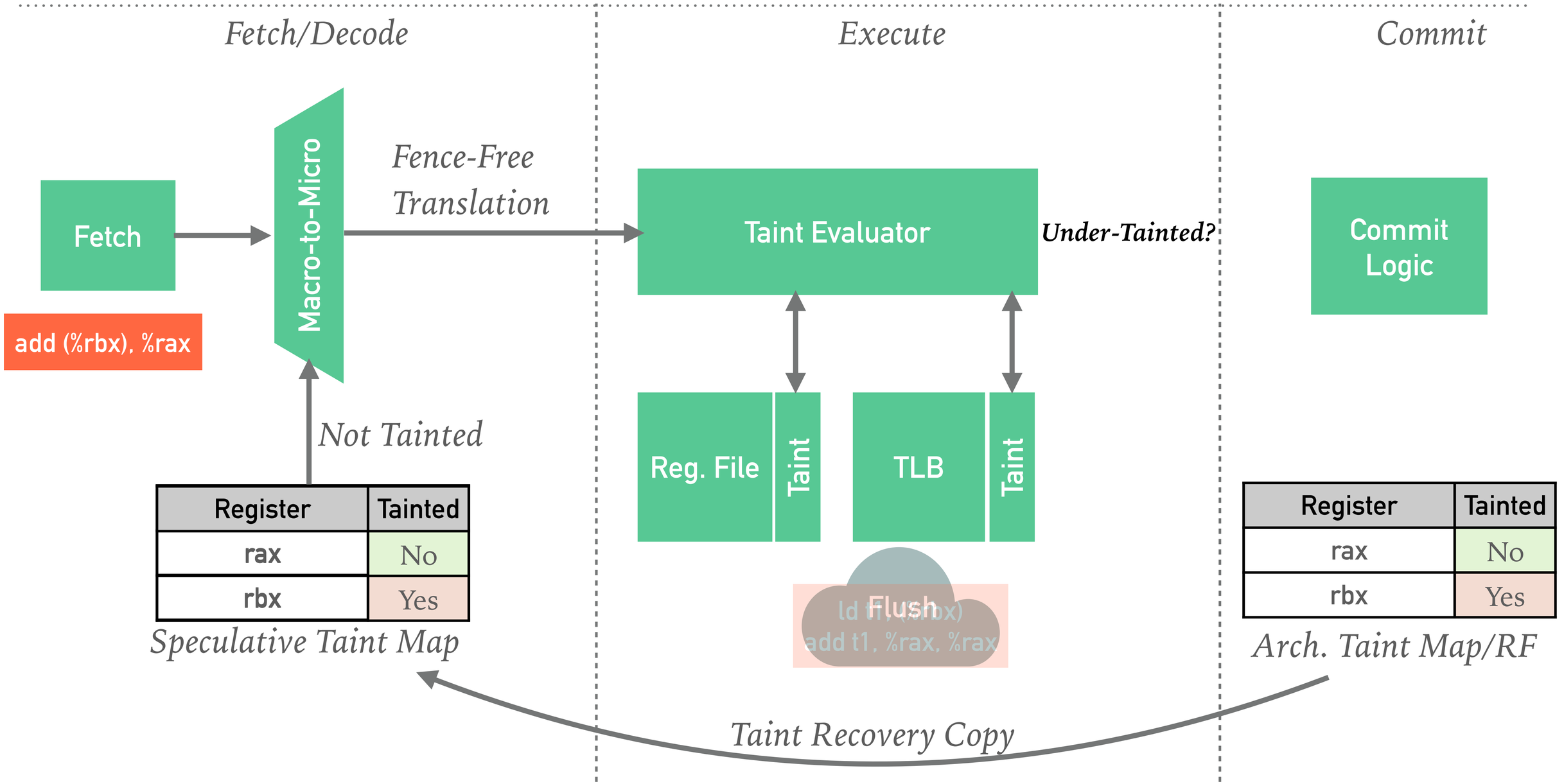
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER



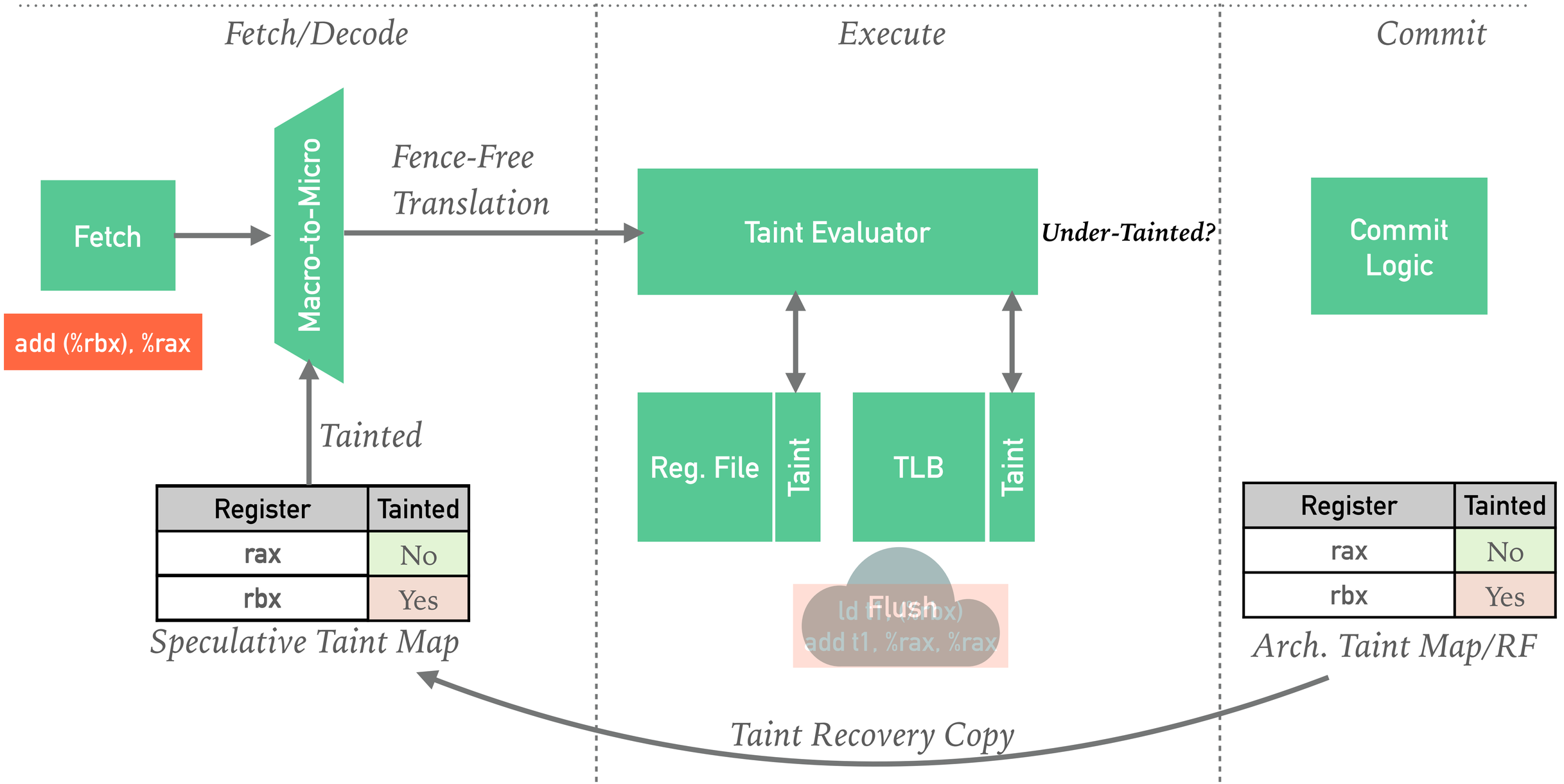
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER



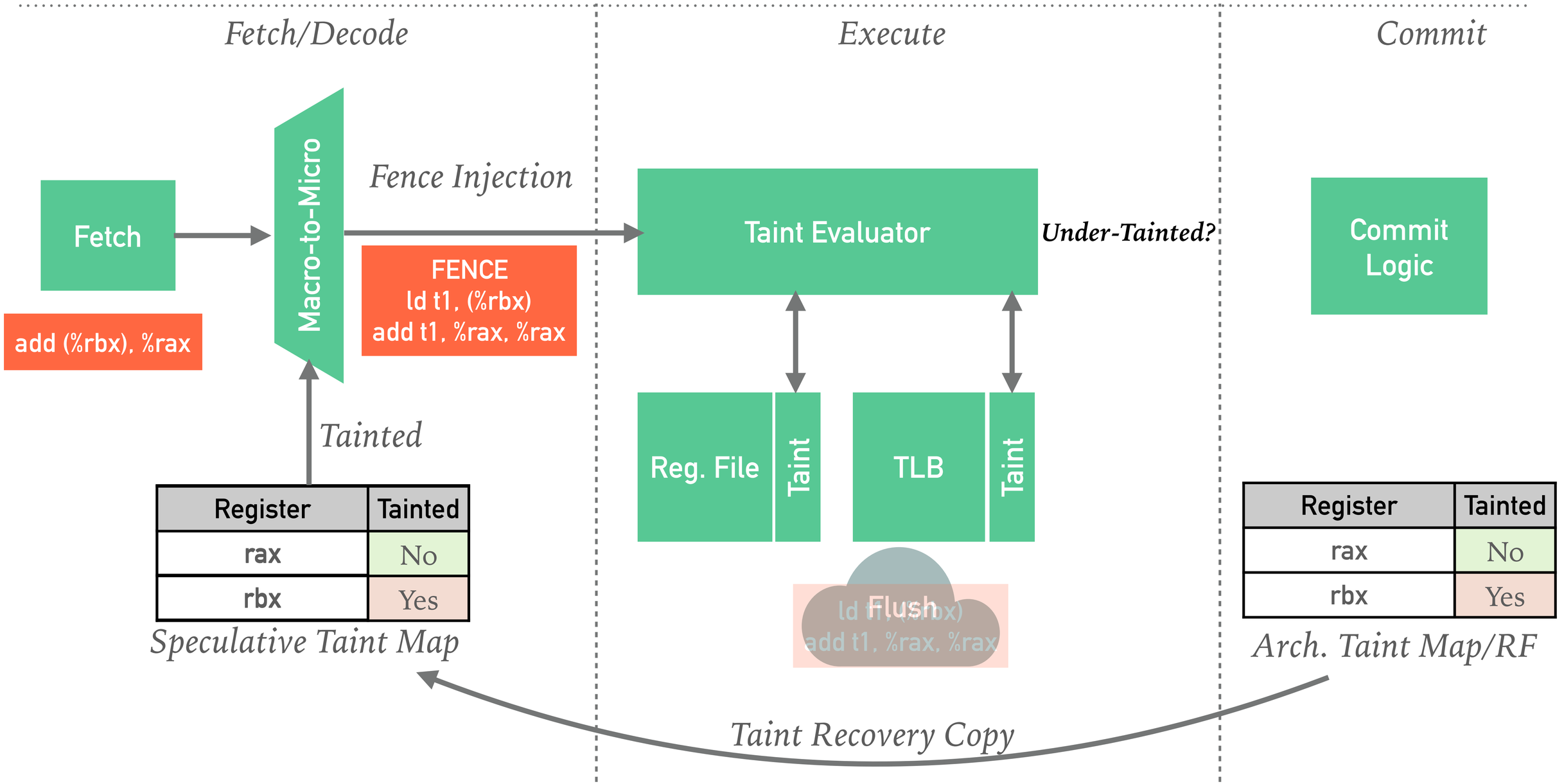
DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER



DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER

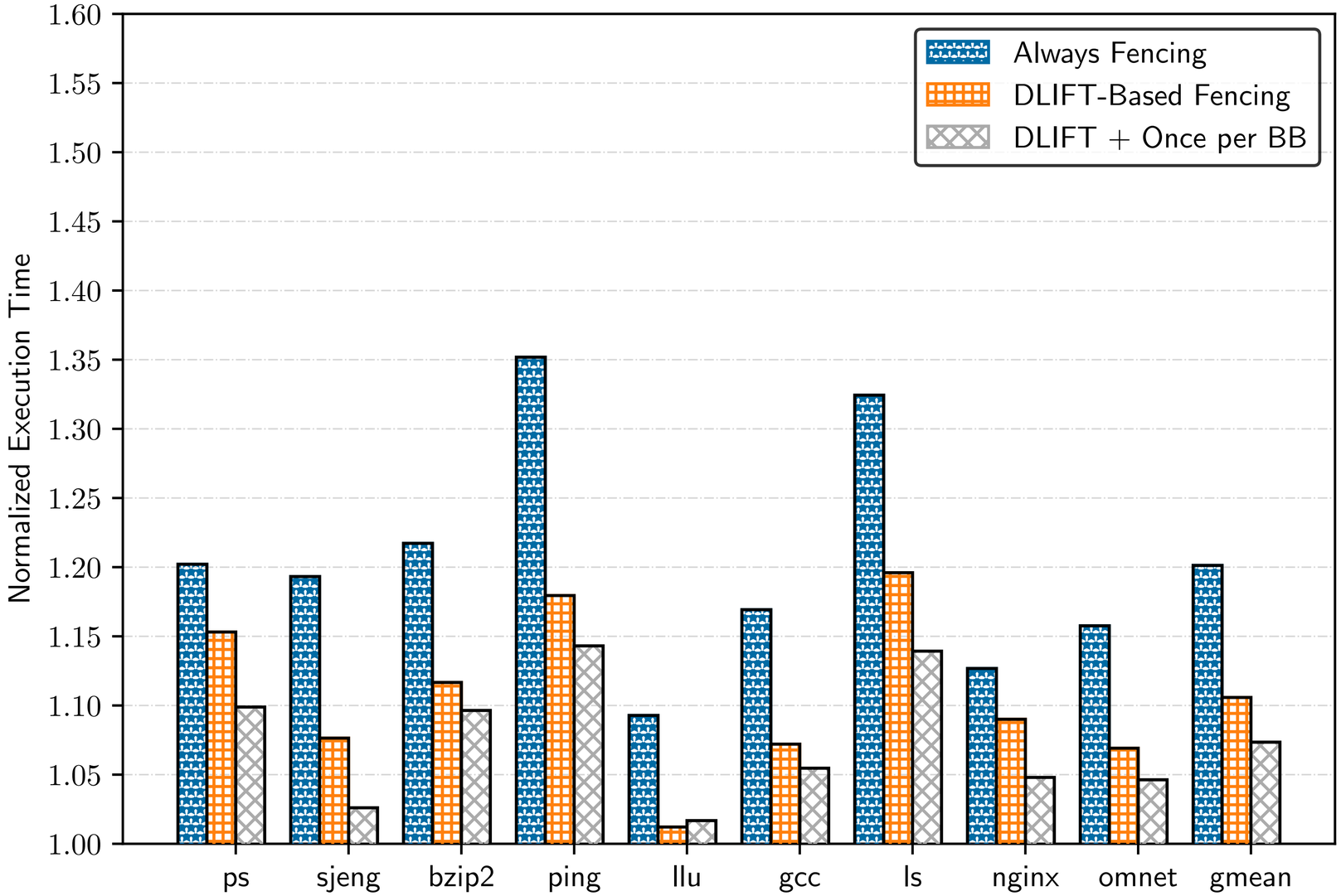


DECODER-LEVEL (SPECULATIVE) INFORMATION FLOW TRACKER



RESULTS — FENCE FREQUENCY OPTIMIZATION

- Taint-Based CFENCE injection reduces the performance overhead **to just 7.7%**

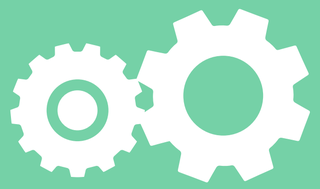


CONTEXT-SENSITIVE FENCING

Low Performance Overhead

CONTEXT-SENSITIVE FENCING

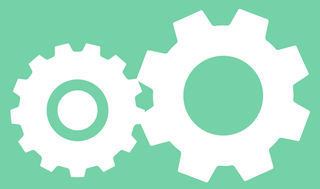
Low Performance Overhead



No Recompilation

CONTEXT-SENSITIVE FENCING

Low Performance Overhead



No Recompilation

Minimal Changes to Processor

THANKS!
QUESTIONS?