HPCA 2019 Keynote

# Towards Secure High-Performance Computer Architectures

**Srini Devadas**

**Massachusetts Institute of Technology**

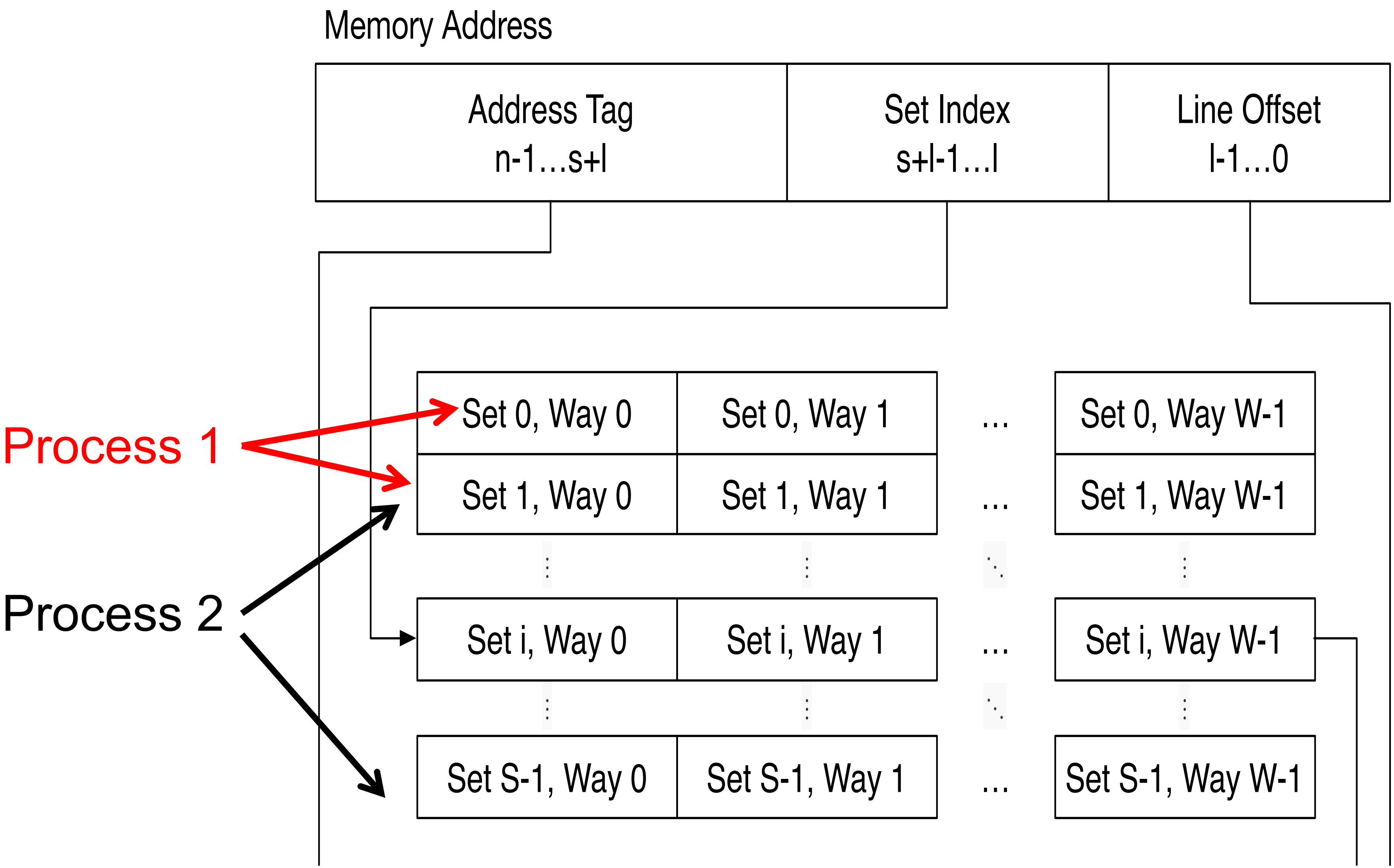
# Architectural Isolation of Processes

Fundamental to maintaining correctness and privacy!

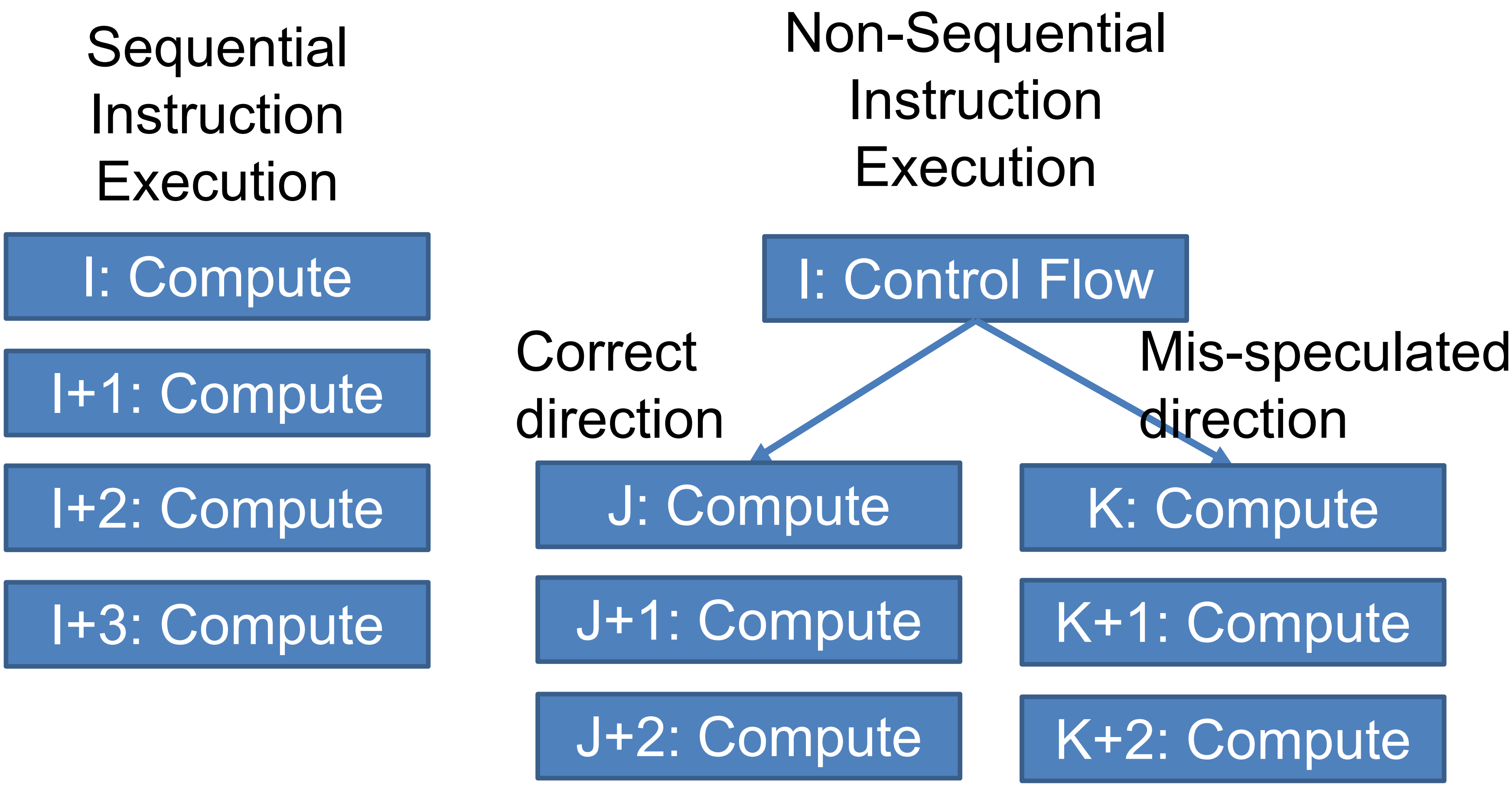
# Performance Dictates Microarchitectural Optimization

Isolation Breaks Because of Shared Microarchitectural State!

# Shared Last Level Cache



# Control Flow Speculation for Performance



# Control Flow Speculation is <u>insecure</u>

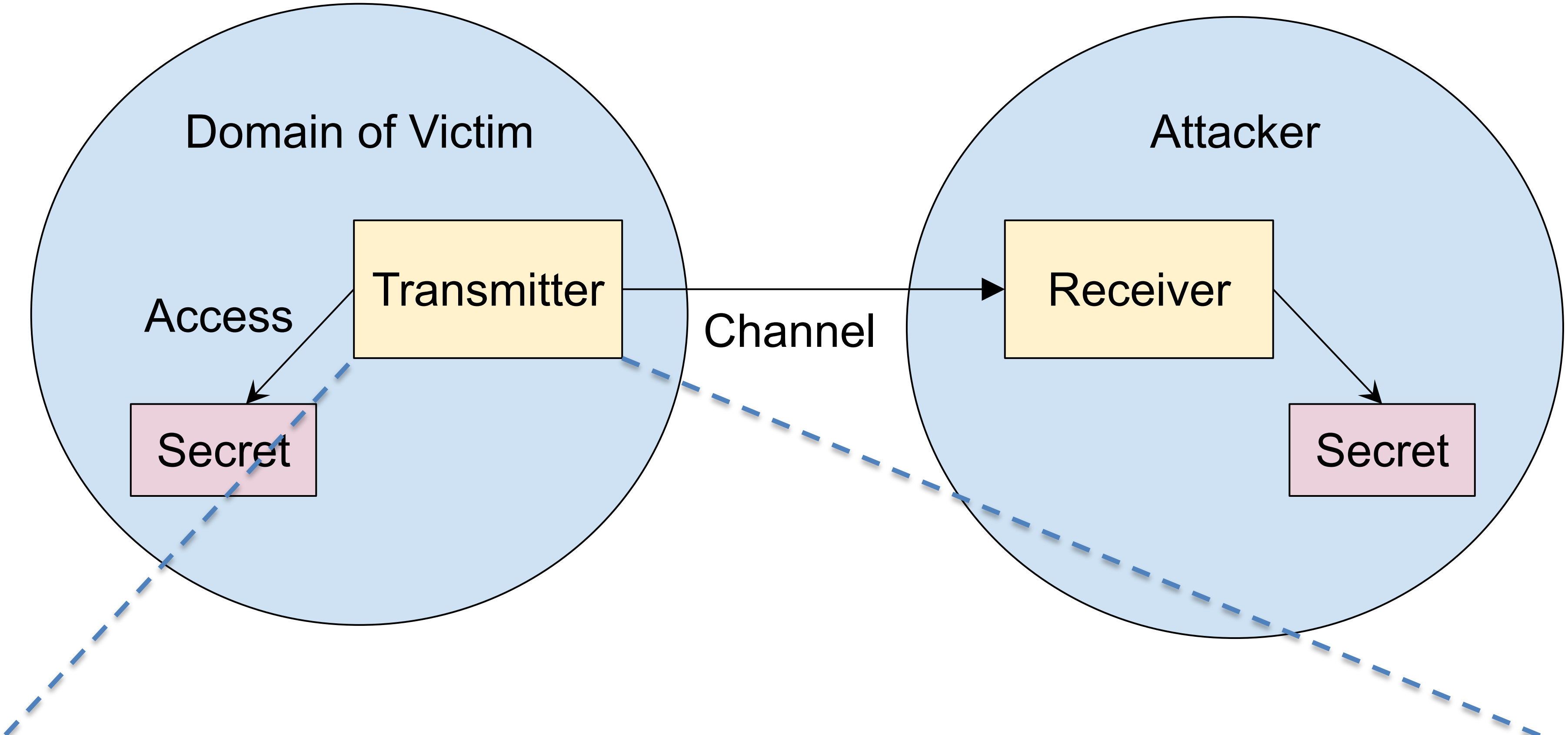
Speculative execution does not affect architectural state → "correct"

… but can be observed via some "side channels" (primarily cache tag state)

… and <u>attacker</u> can influence (mis)speculation (branch predictor inputs not authenticated)

A huge, complex attack surface!

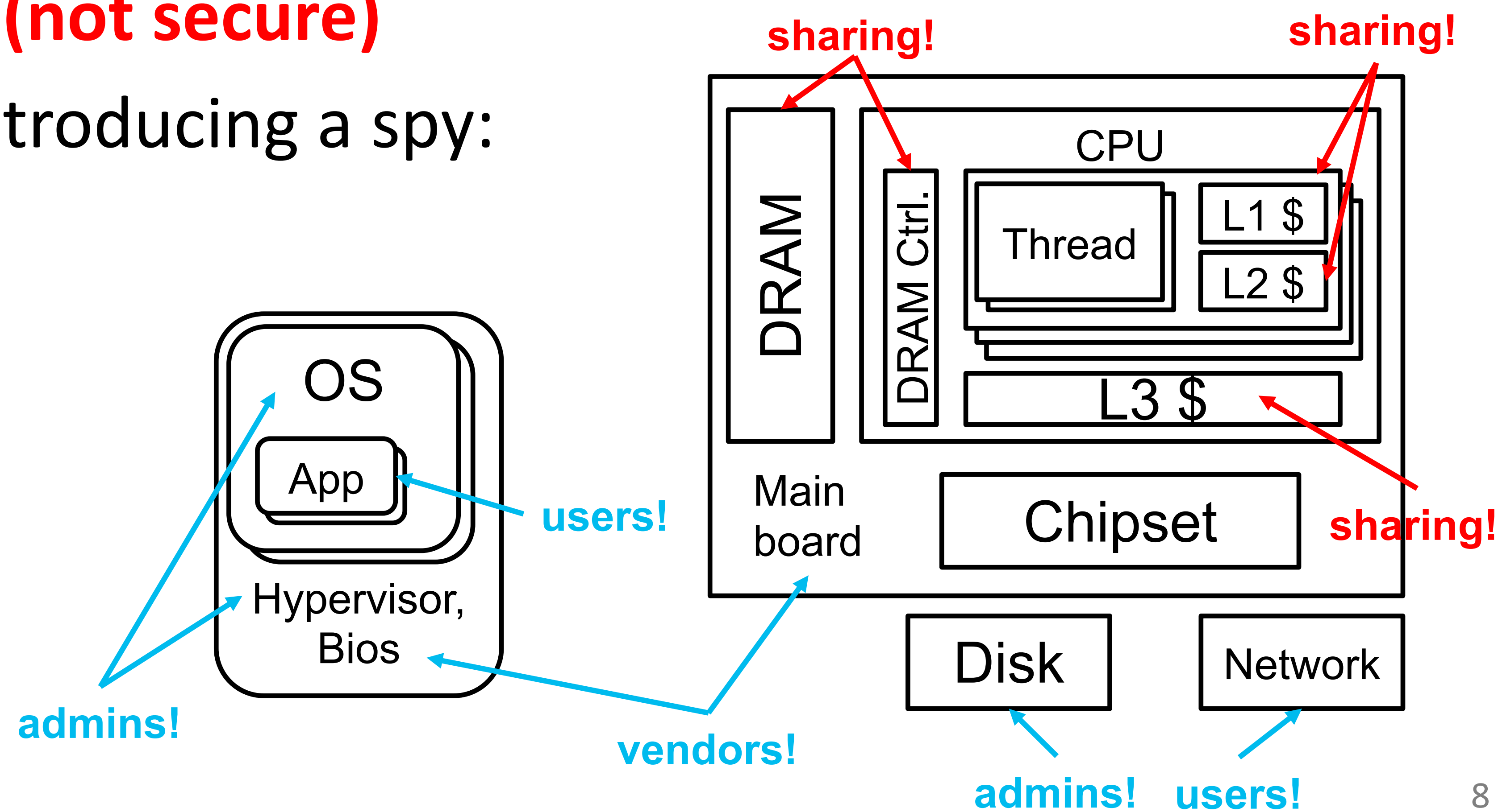
# Building a Transmitter




Pre-existing (RSA conditional execution example)
Written by attacker (Meltdown)
Synthesized out of existing victim code by attacker (Spectre)

# Side Channels Gone Wild!

- Real systems: large, complex, cyberphysical **(not secure)**
- Introducing a spy:



# Philosophy

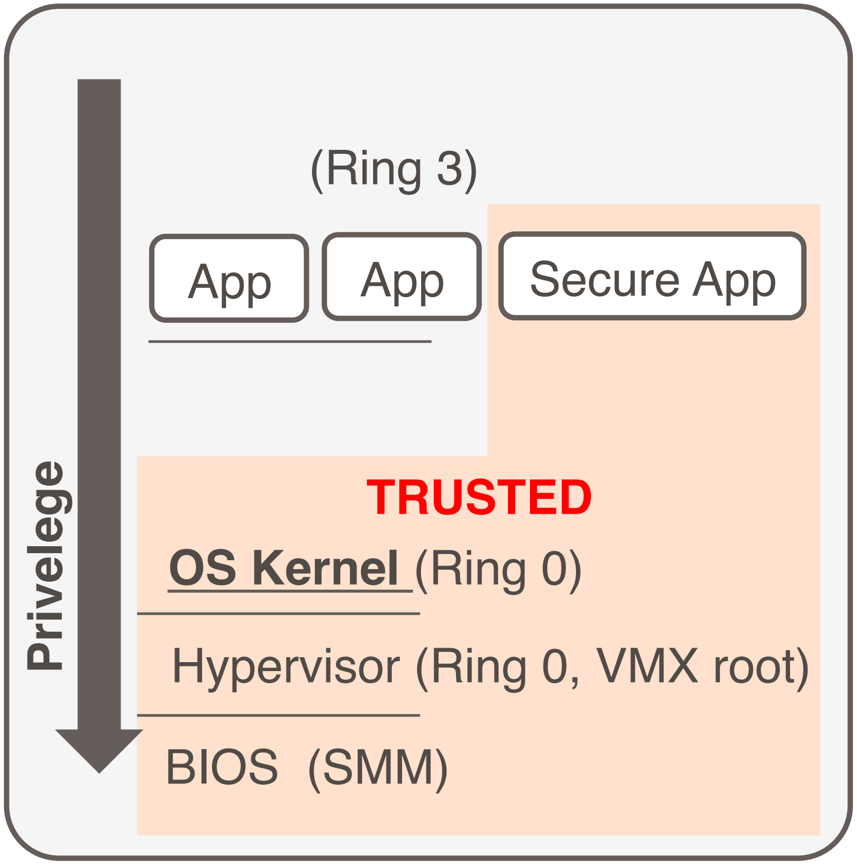
**Build enclaves on an enclave platform, not just processes**

# Enclaves strengthen the process abstraction

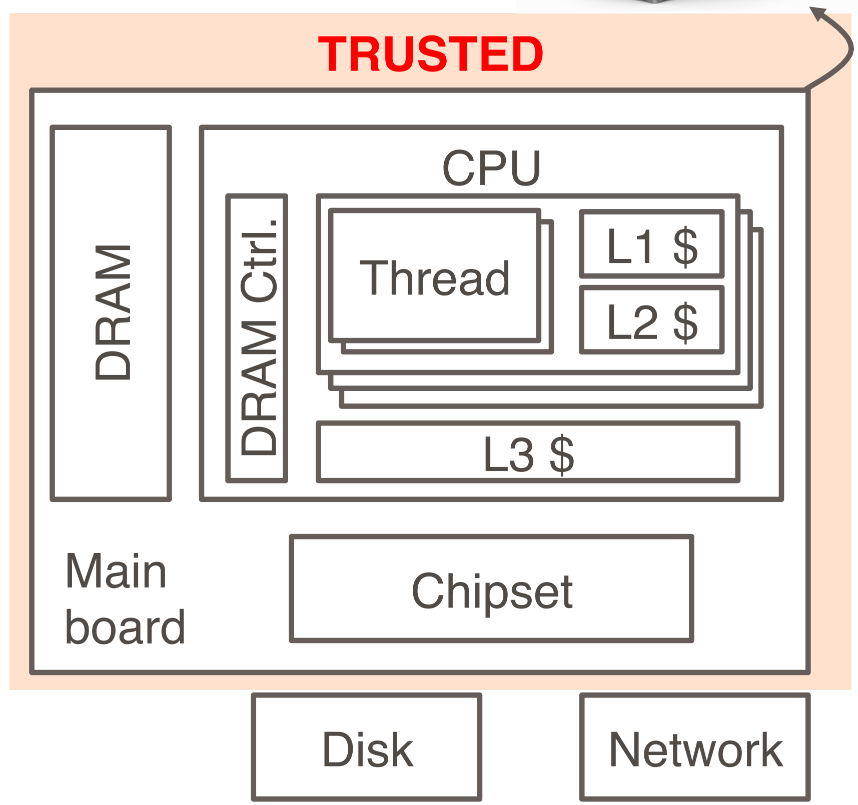
- Processes guarantee *isolation of memory*
- Enclaves provide a stronger guarantee
  - No other program can infer anything private from the enclave program through its use of shared resources or shared microarchitectural state
- Largely decouple performance considerations from security
- Minimally invasive hardware changes
- Provable security under chosen threat model

# A Typical Computer Trusted Computing Base

Software…

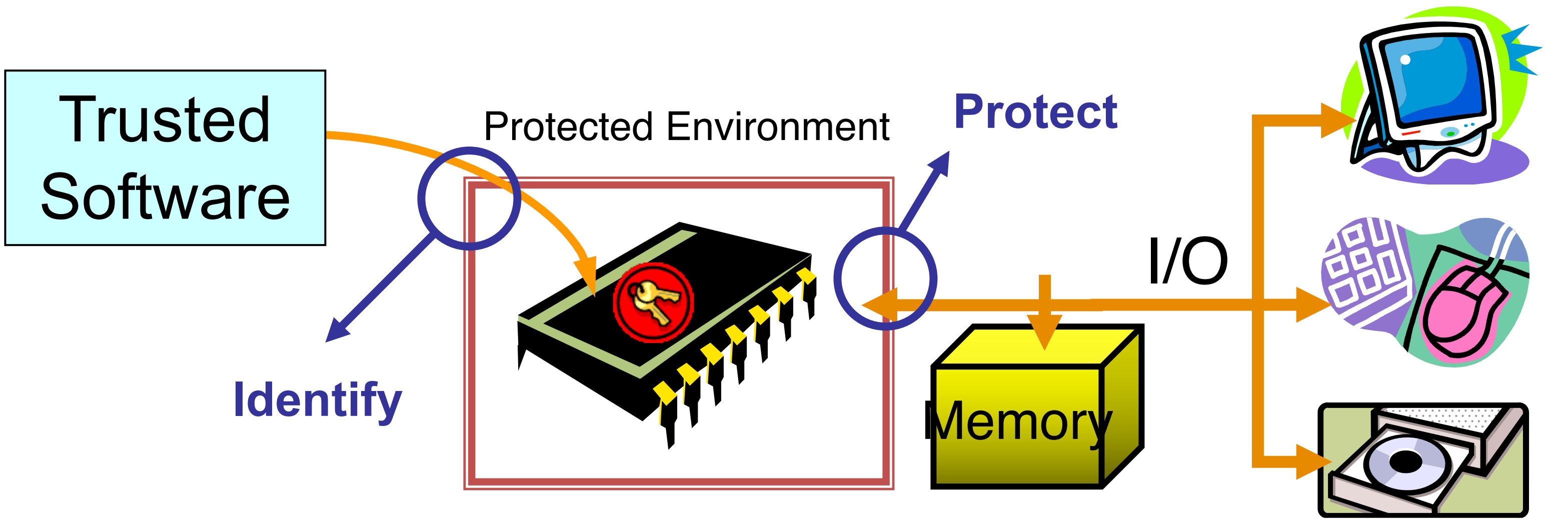



… Running on hardware



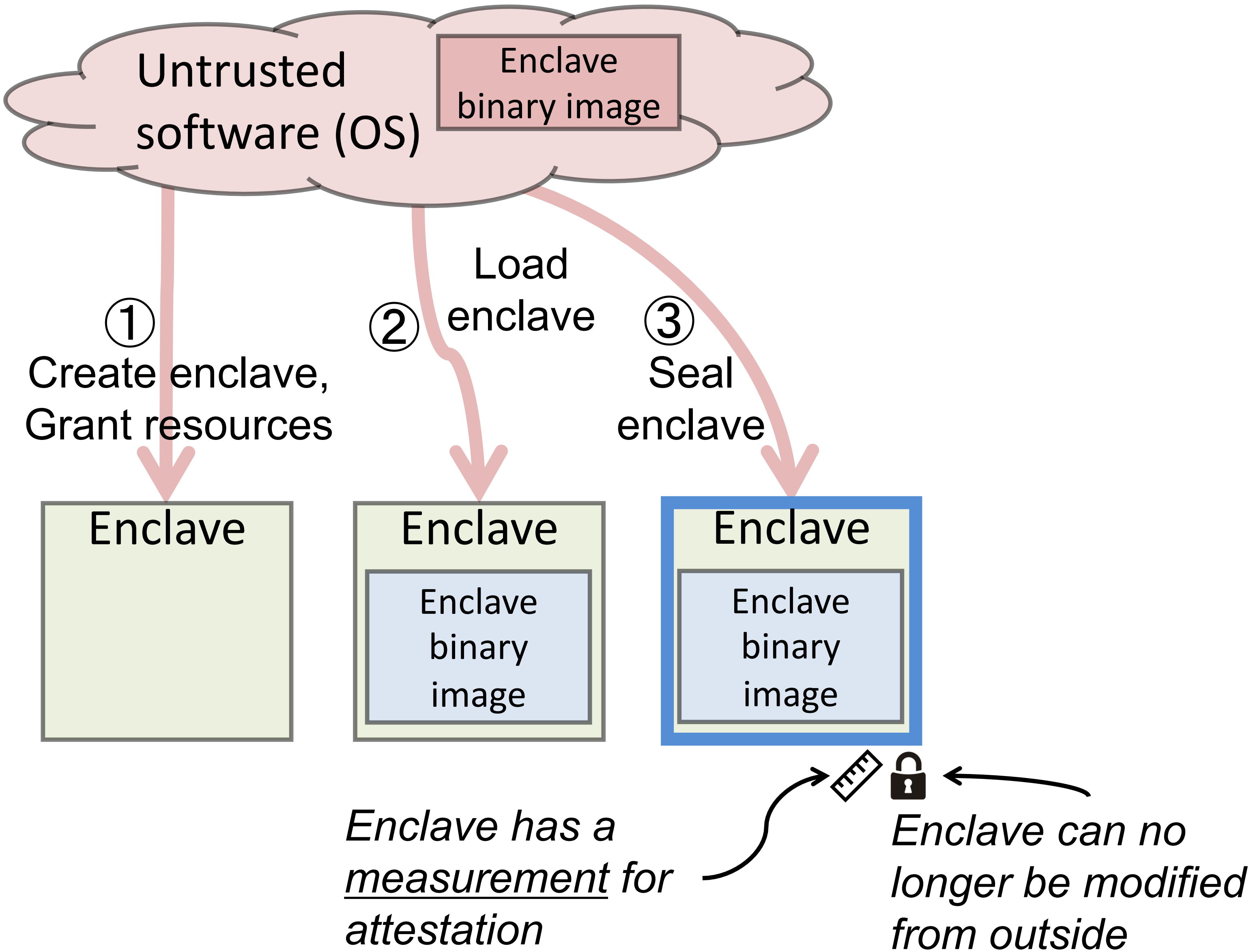
# Single-Chip Secure Processor: Shrink the TCB

***Edward Suh's ICS 2003 Talk on Aegis processor***

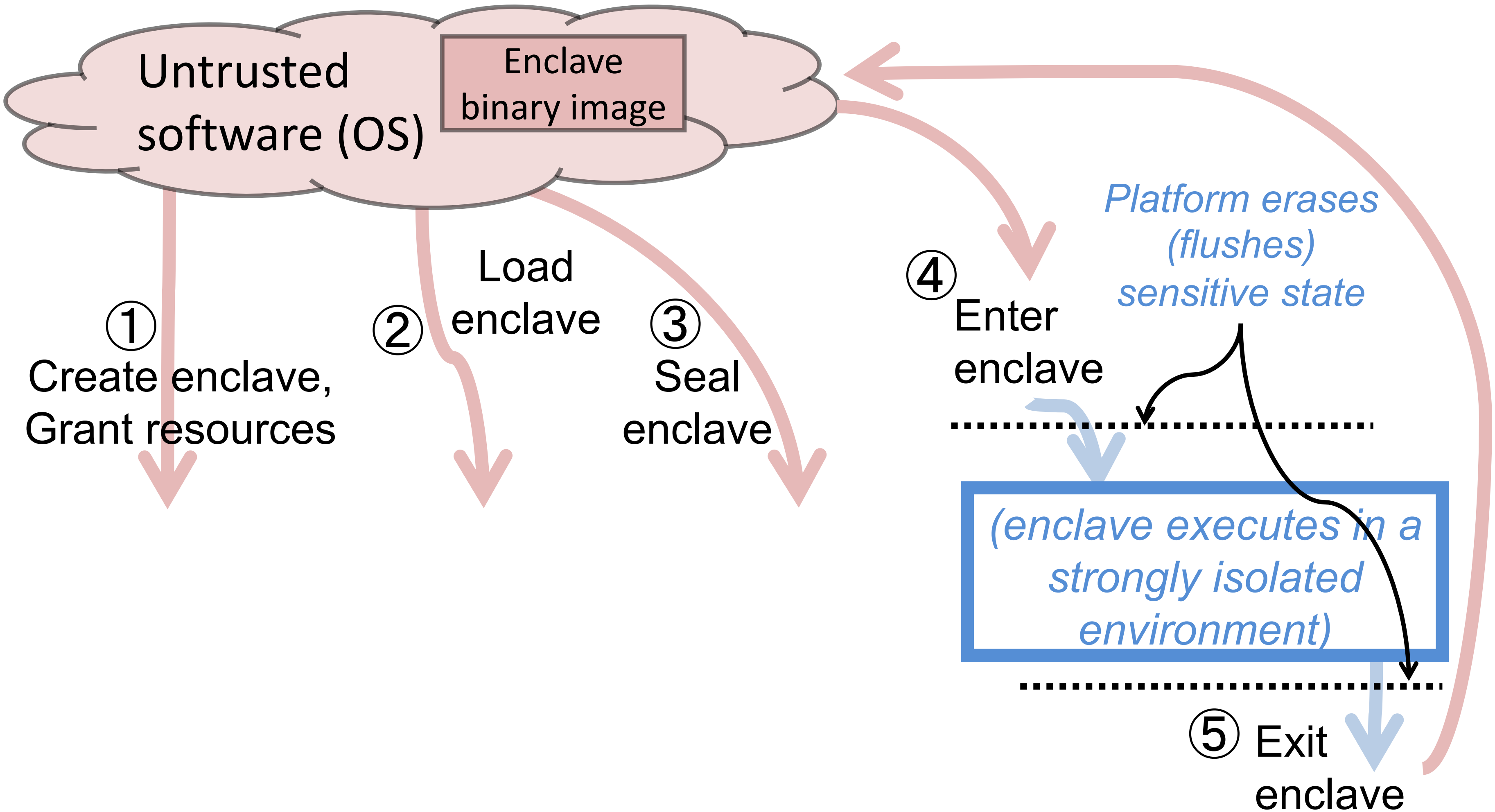



- Enclave assumes trusted hardware + trusted software "monitor"
- Operating system is untrusted

# Enclave Lifecycle (simplified)



# Enclave Lifecycle (simplified)



# Strong Isolation Goal

- Any attack by a privileged attacker on the same machine as the victim that can extract a secret inside the victim enclave, could also have been run successfully by an attacker on a different machine than the victim.
  - *No protection against an enclave leaking its own secrets through its public API.*

- Three strategies for isolation: Spatial isolation, temporal isolation and cryptography

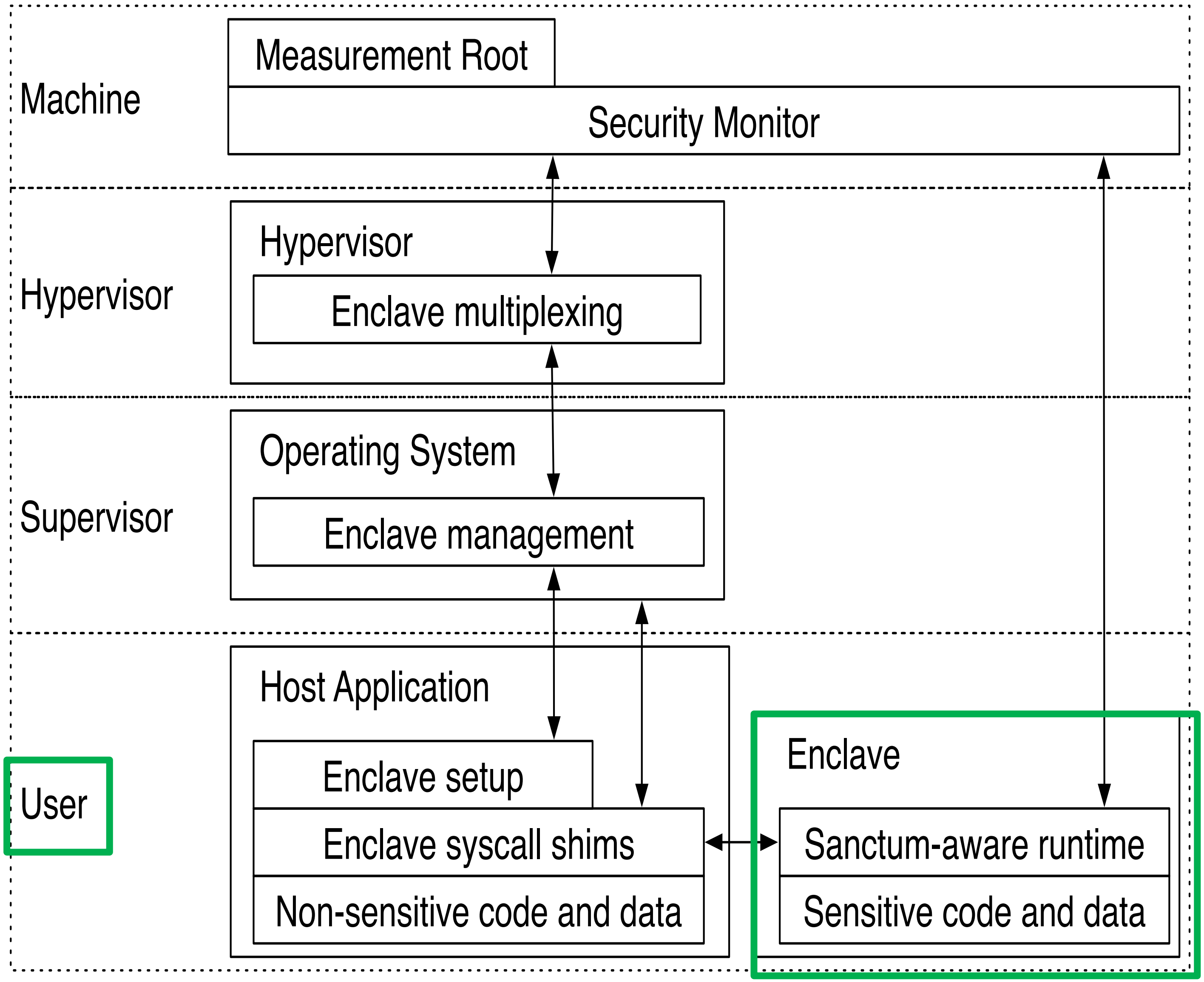
# Sanctum Design

Victor Costan, Ilia Lebedev

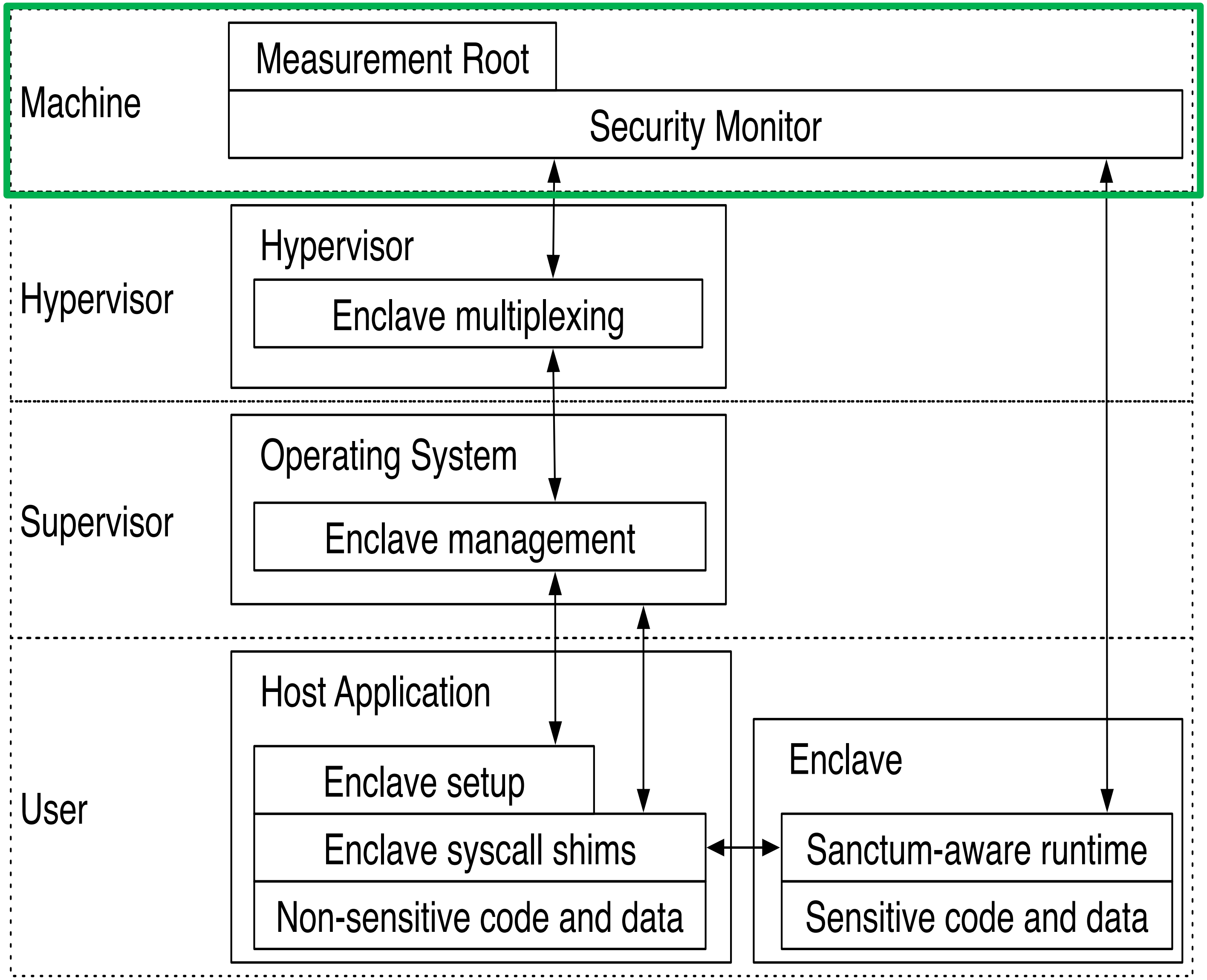
*Sanctum: Minimal Hardware Extensions for Strong Software Isolation*

CSAIL

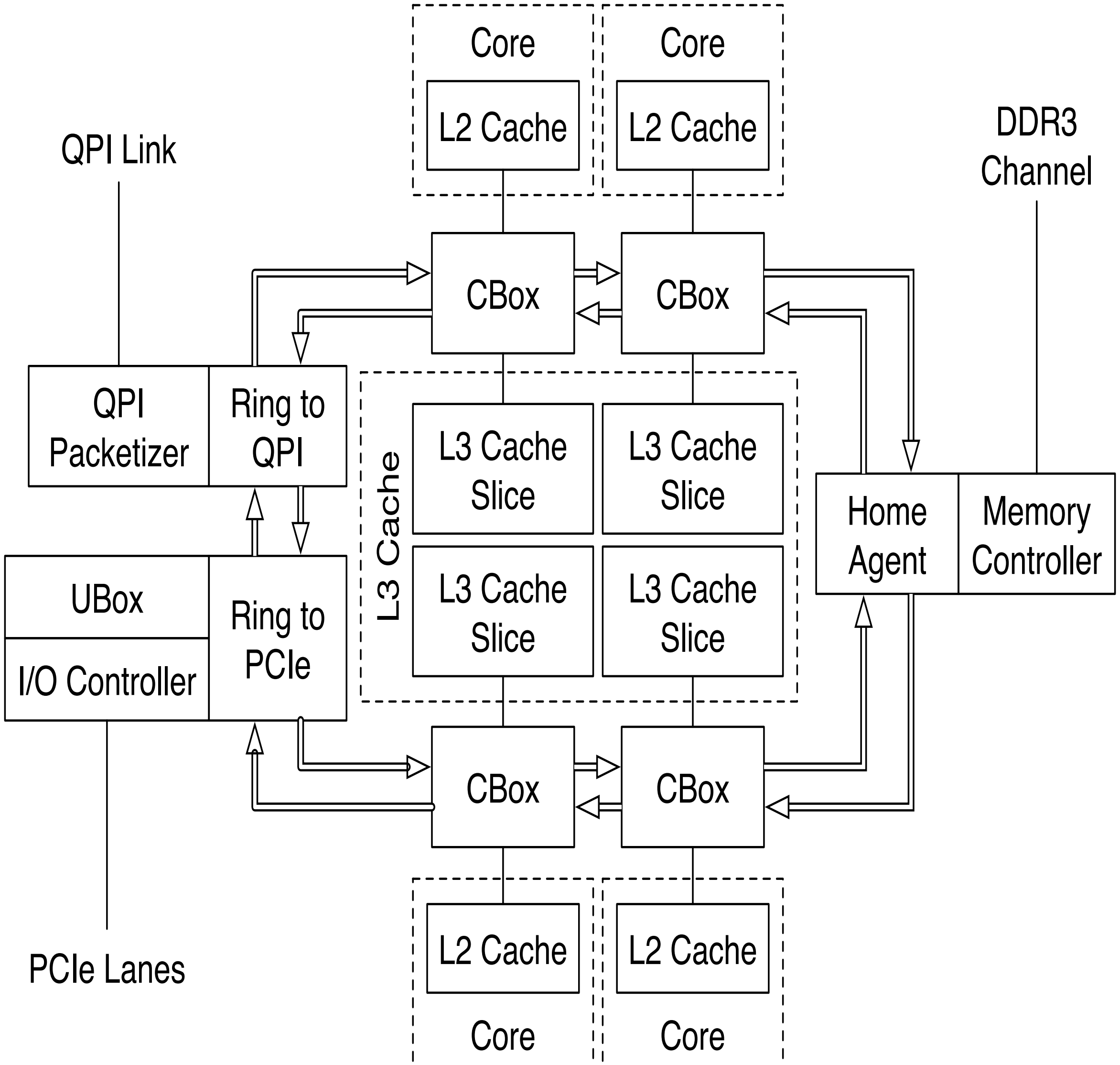
# Software Stack



# Software Stack



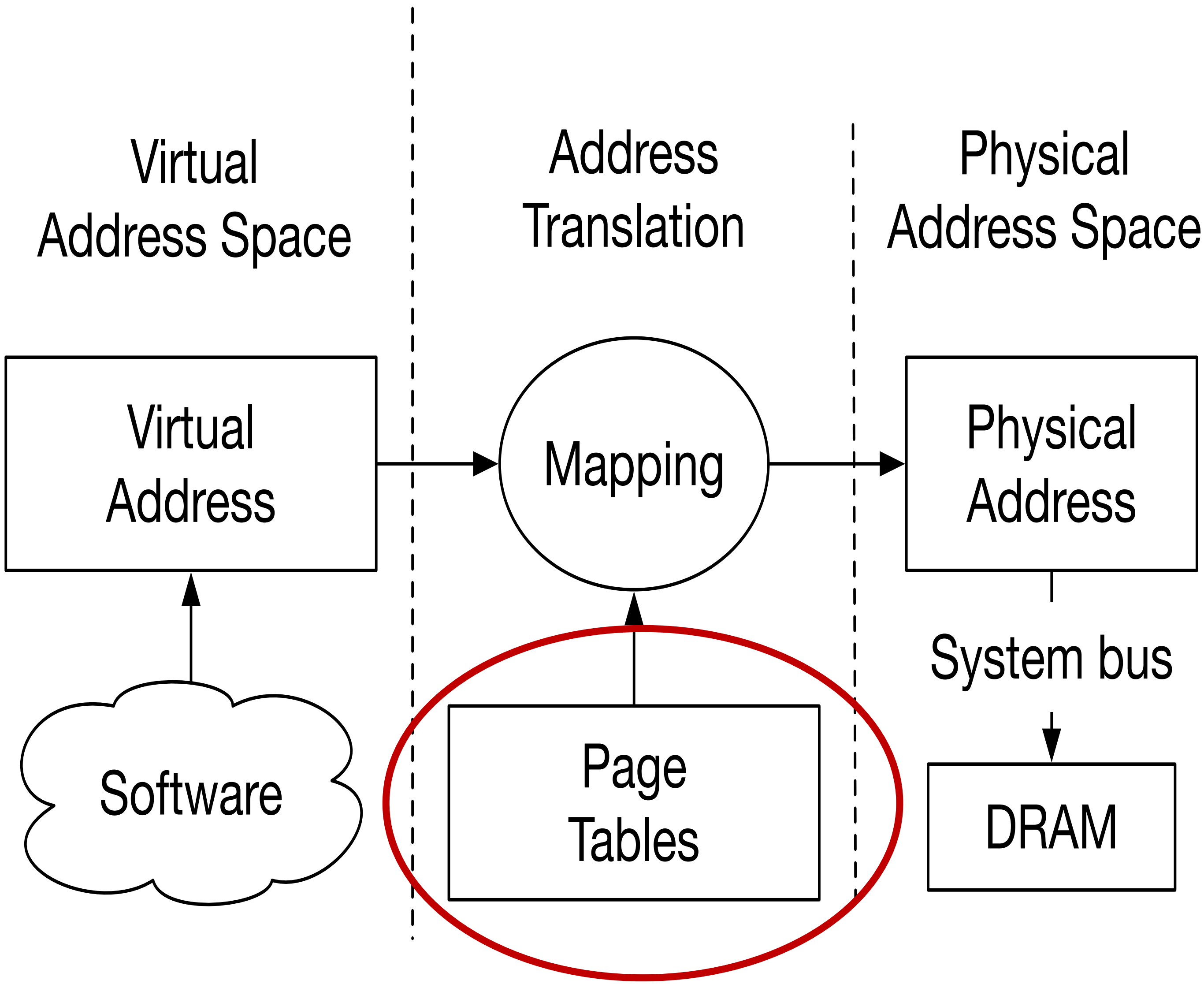
# Target: multi-core processor (no hyperthreading, no speculation)



# Microarchitectural State Isolation in Sanctum Enclaves

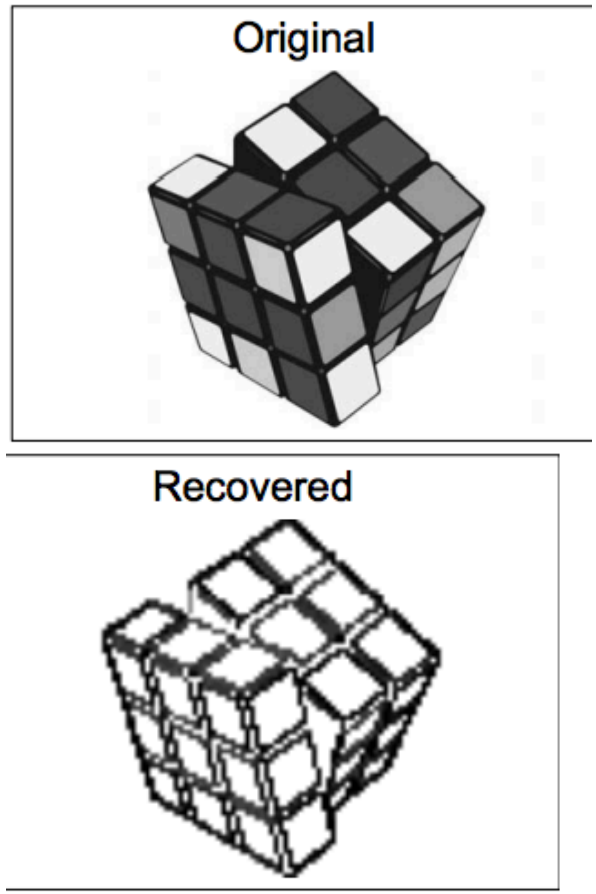
- Resources exclusively granted to an enclave, and scheduled at the granularity of process context switches are isolated temporally
  - Register files, branch predictors, private caches, and private TLBs
- Resources shared between processes on-demand, with arbitrarily small granularity are isolated spatially by partitioning
  - Shared caches and shared TLBs

# Operating System Manages Page Tables

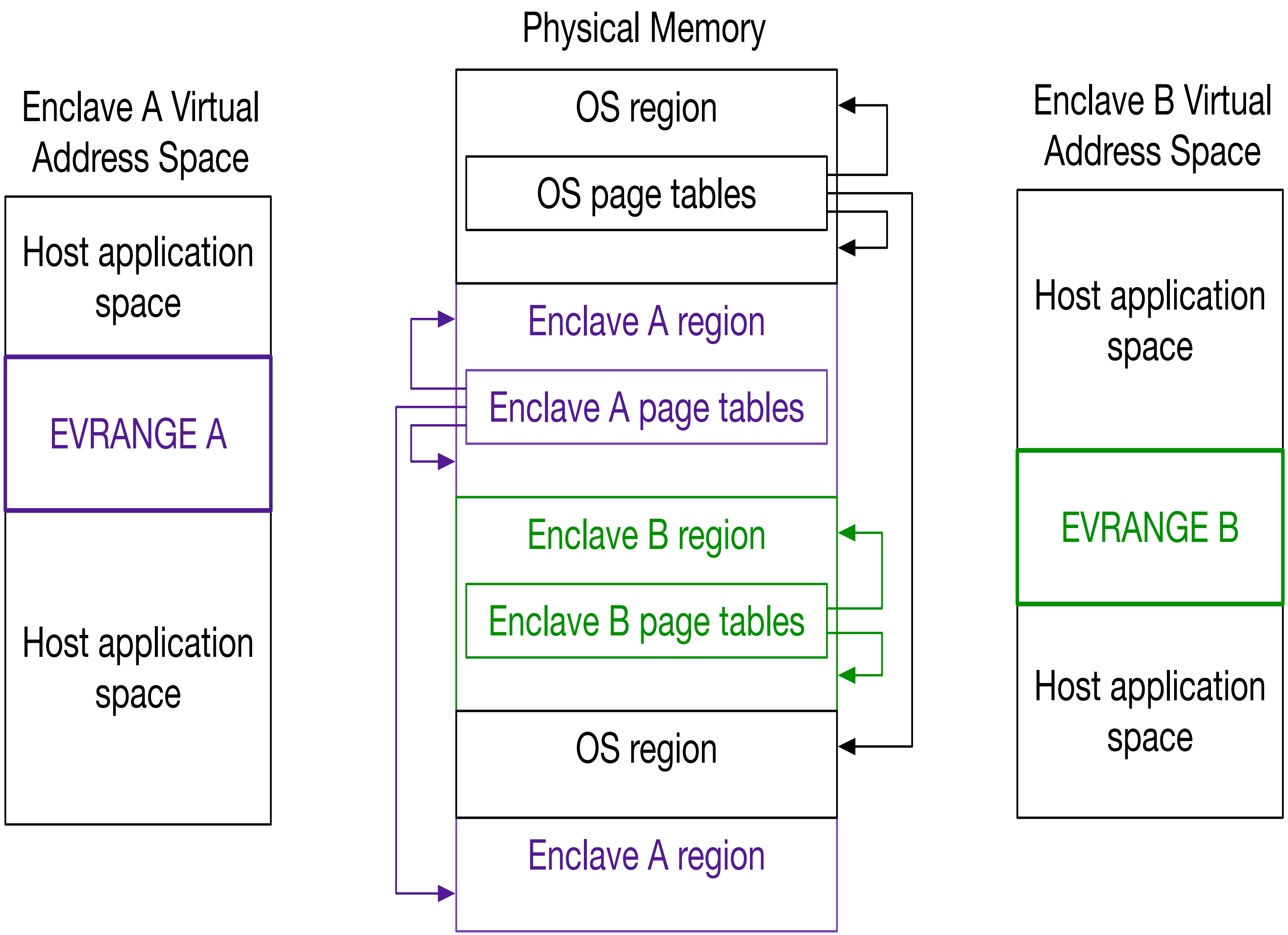


# Practical Software Attack on SGX "Simulators"

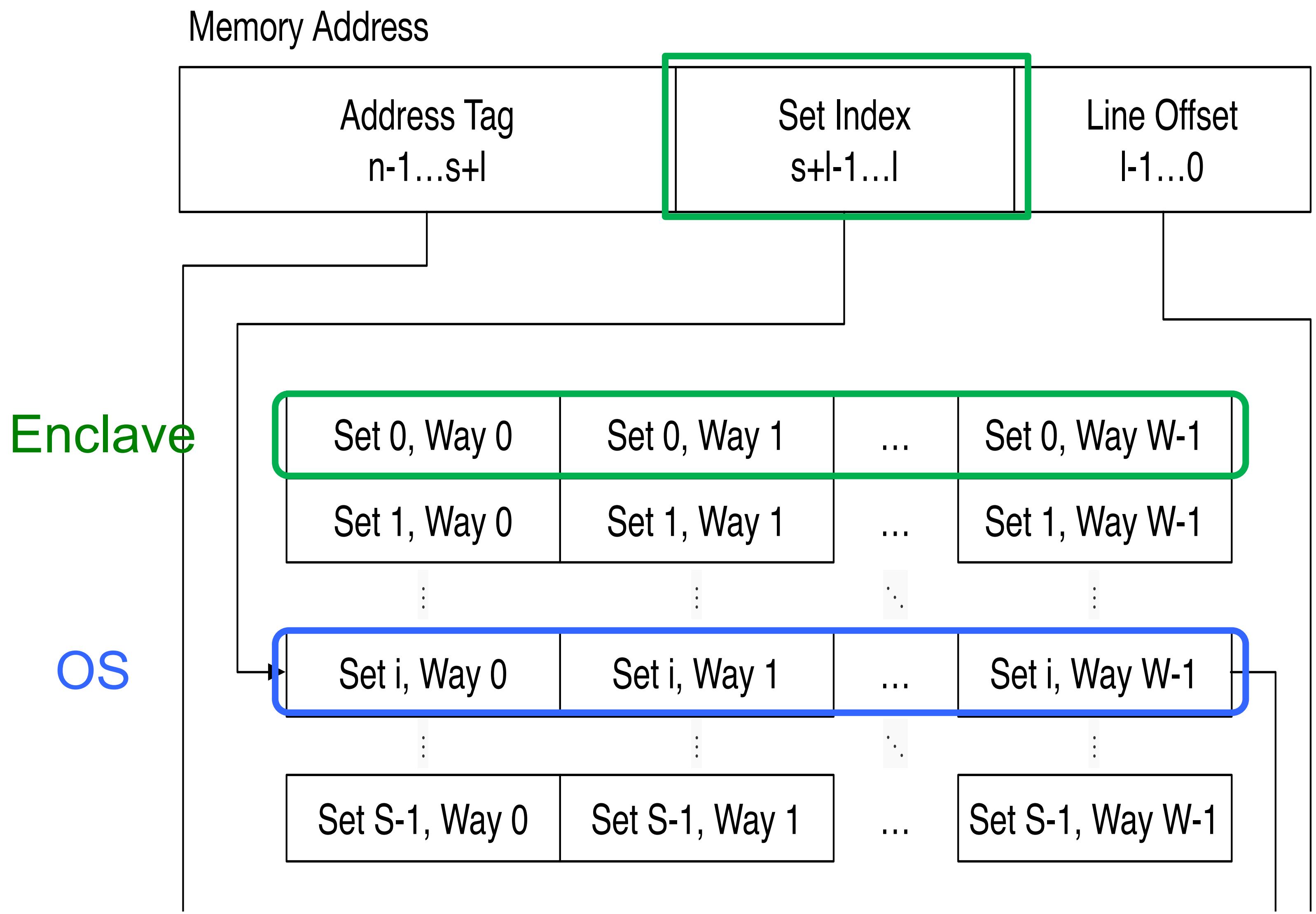
- Microsoft Research, IEEE S&P 2015: Exploit no-noise side channel due to page faults



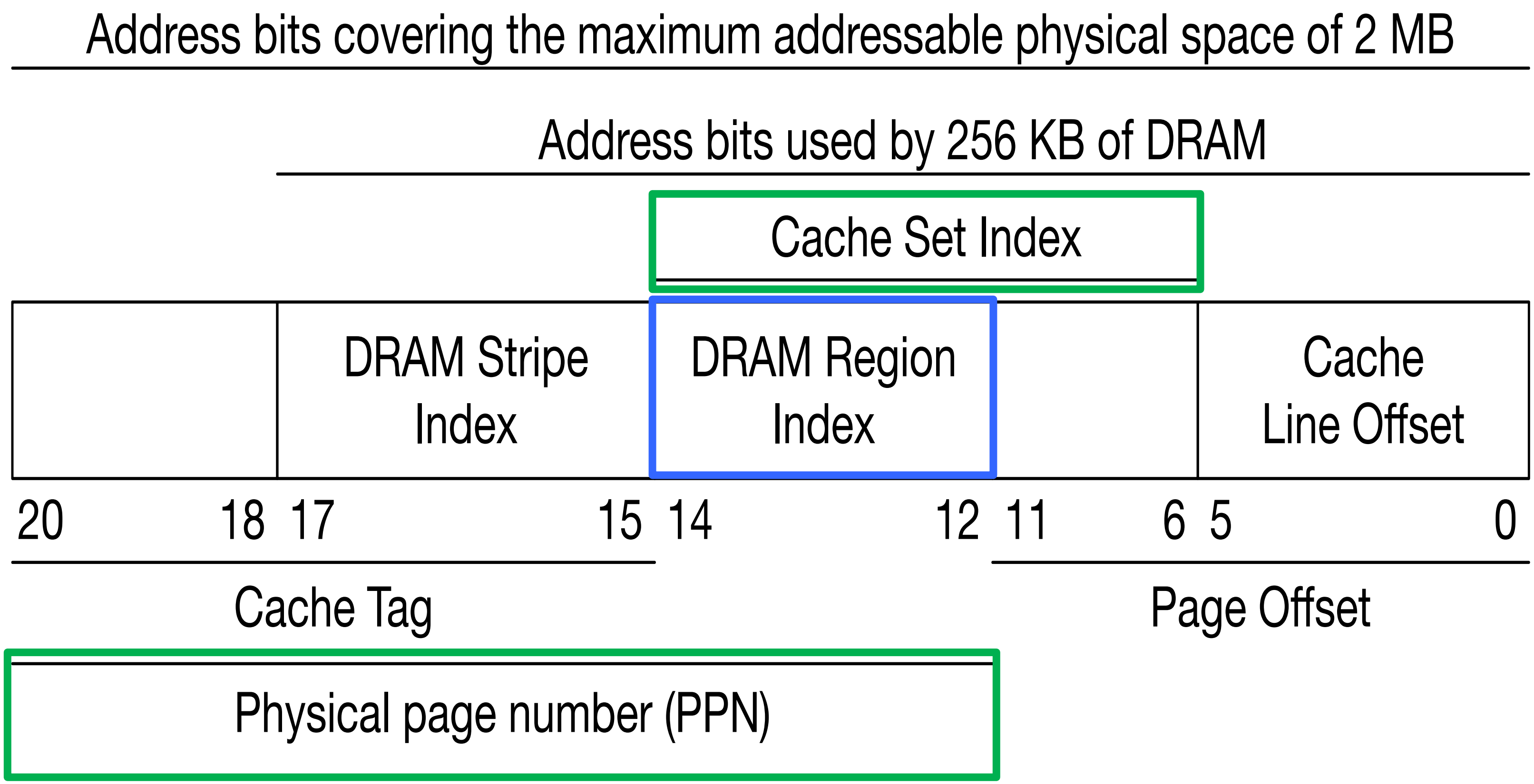
# Page Table Isolation



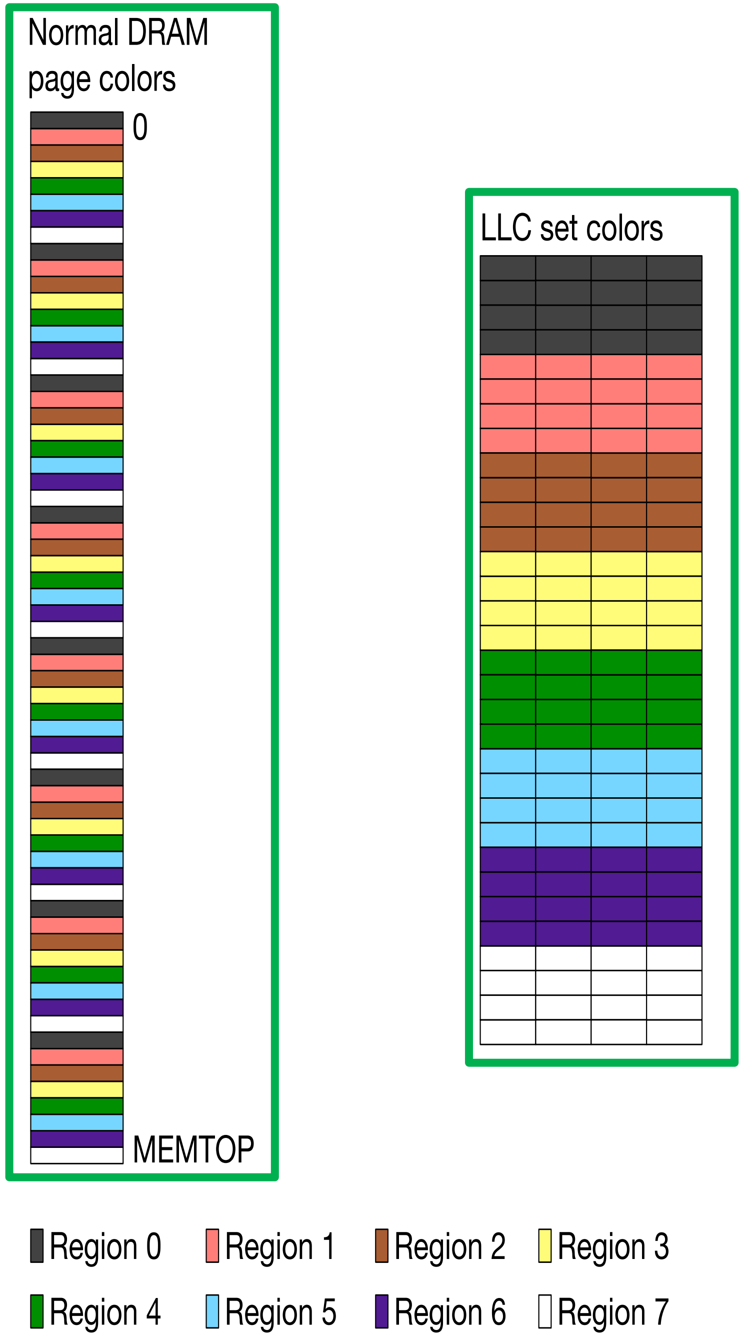
# Partitioning to Prevent Timing Attacks



# Page Coloring

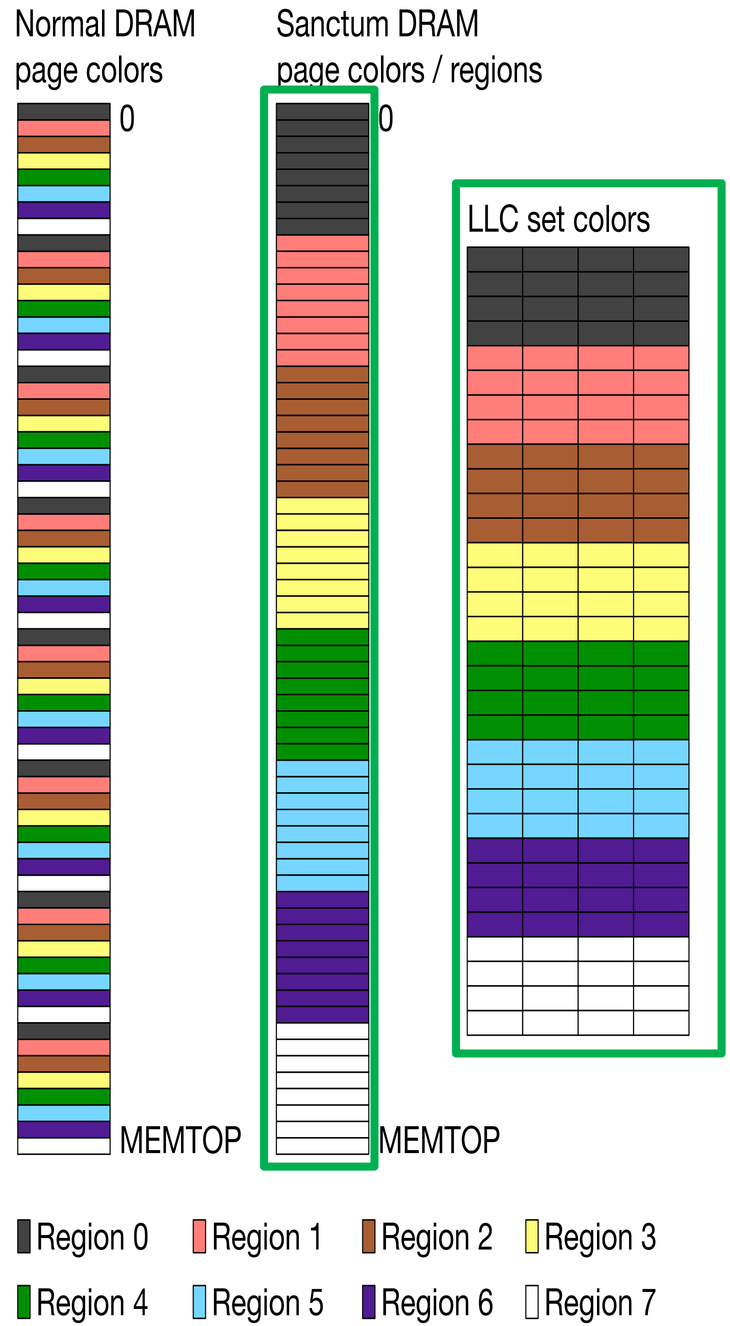


# Page Colors = DRAM Regions



# Page Colors = DRAM Regions

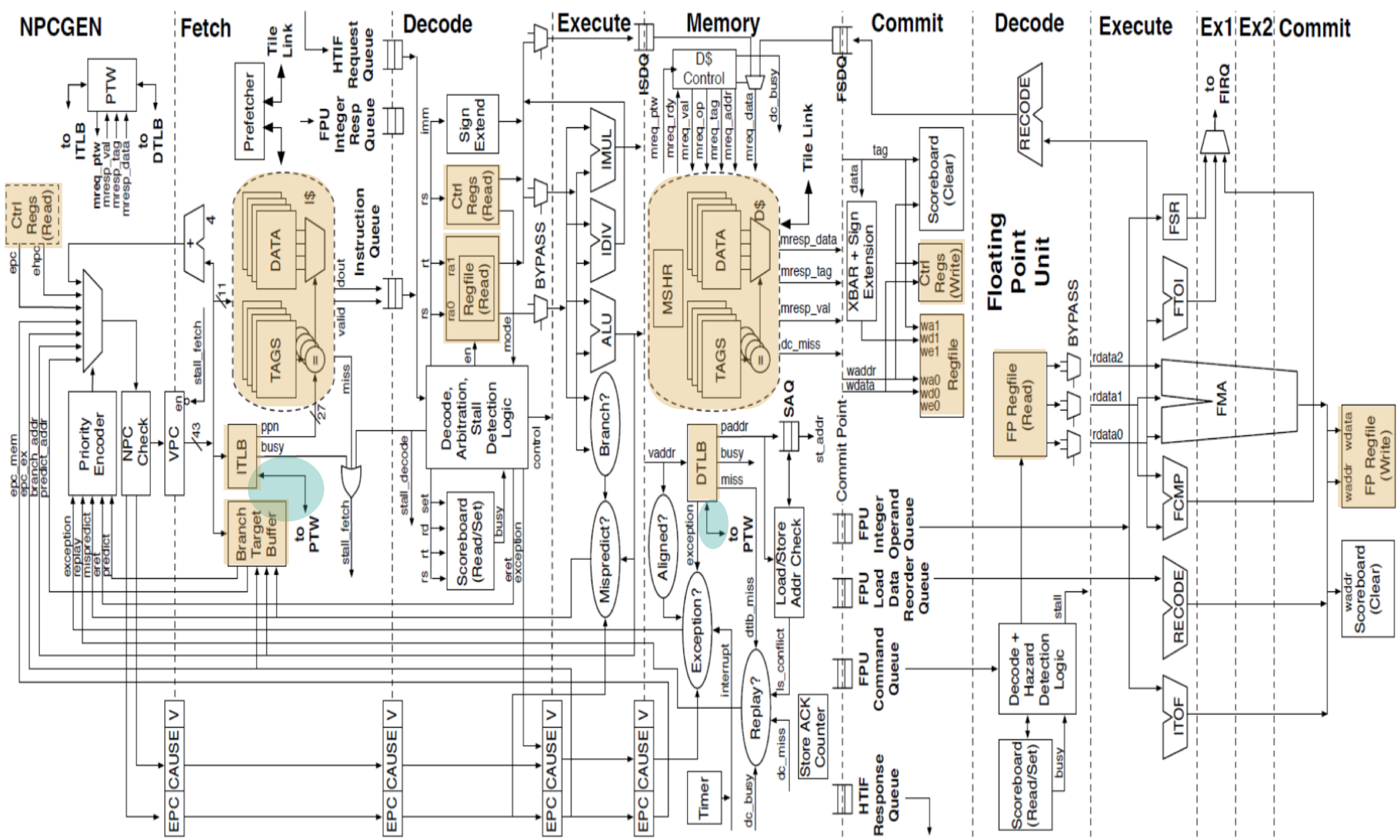
A little bit-shifting gets us a large contiguous DRAM region



# Sanctum Secure Processor

## No Speculation, No Hyperthreading

RISCV Rocket Core, Changes required by Sanctum (+ ~2% of core)




Also requires 9 new config registers

# Sanctum Status and Current Limitations

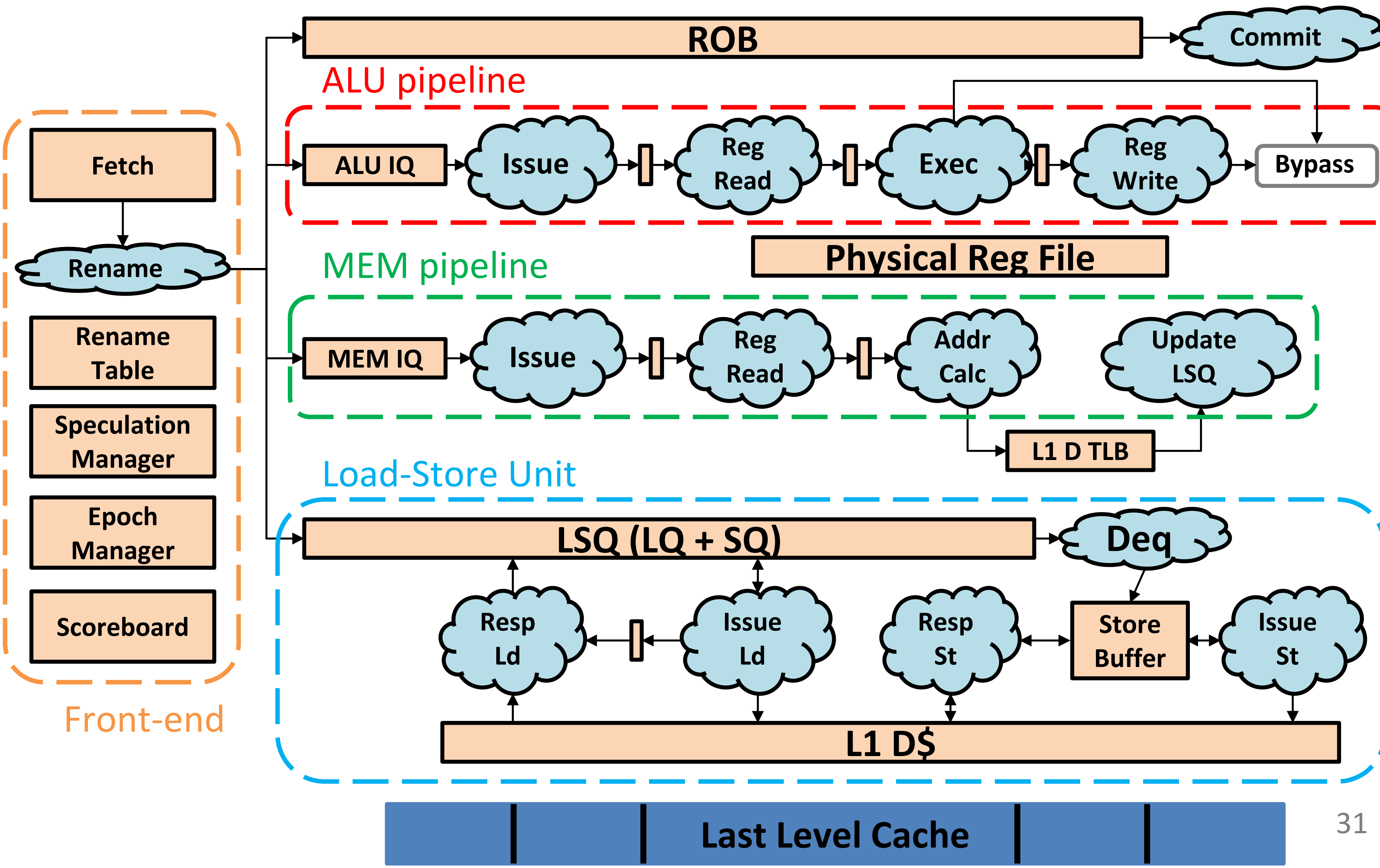
- We have built an open-source Sanctum based on the RISC-V ISA
  - Low performance and area overhead to support enclaves
  - Ongoing formal verification effort
- Sanctum is an academic, lightweight processor
- **Apply its design philosophy to speculative out-of-order (OOO) processors, which need to protect against Spectre-style attacks**

# MI6 Design

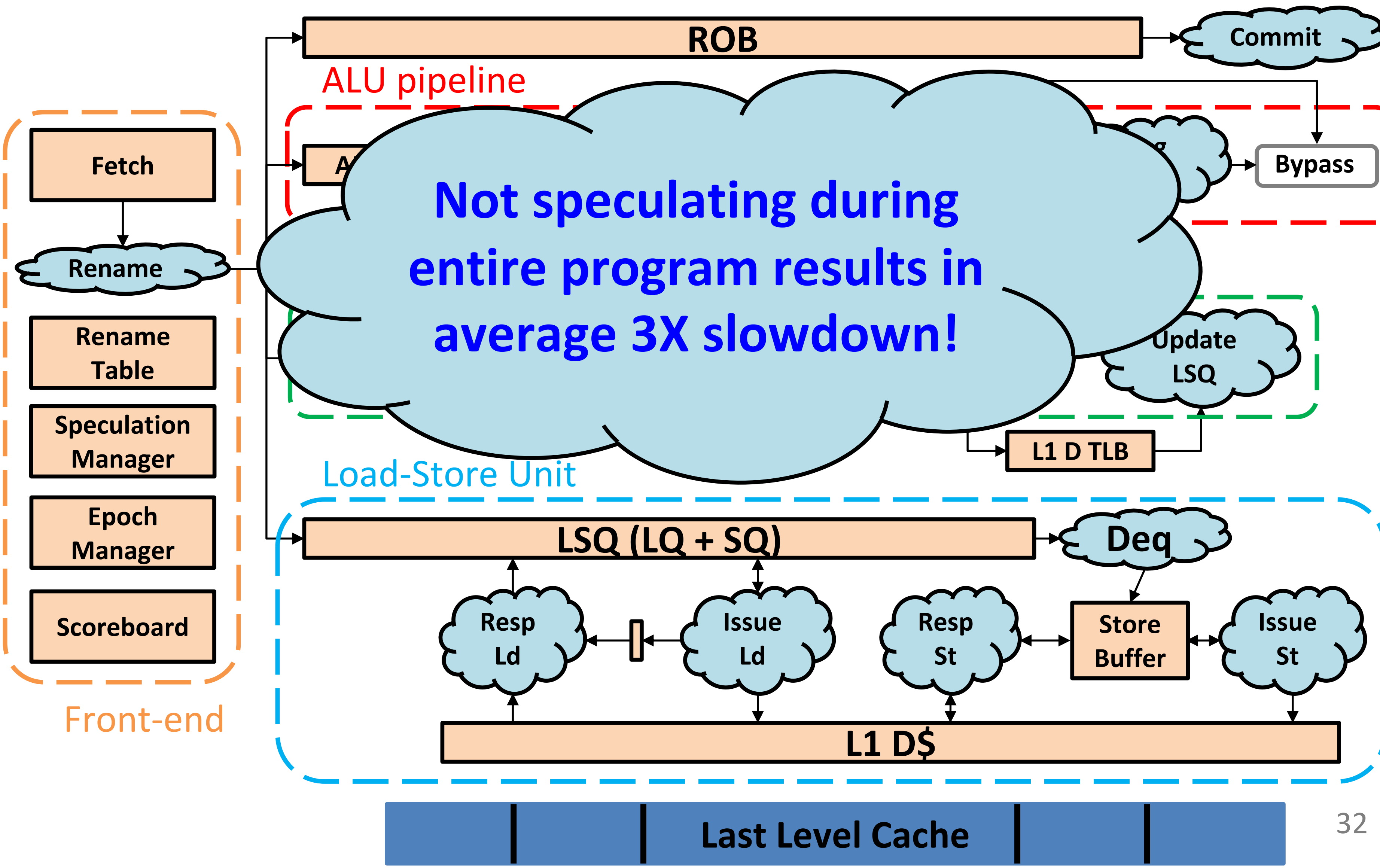
Thomas Bourgeat, Ilia Lebedev,
Andrew Wright, Sizhuo Zhang, Arvind

*MI6: Secure Enclaves in a Speculative Out-of-Order Processor*

# RiscyOO Processor

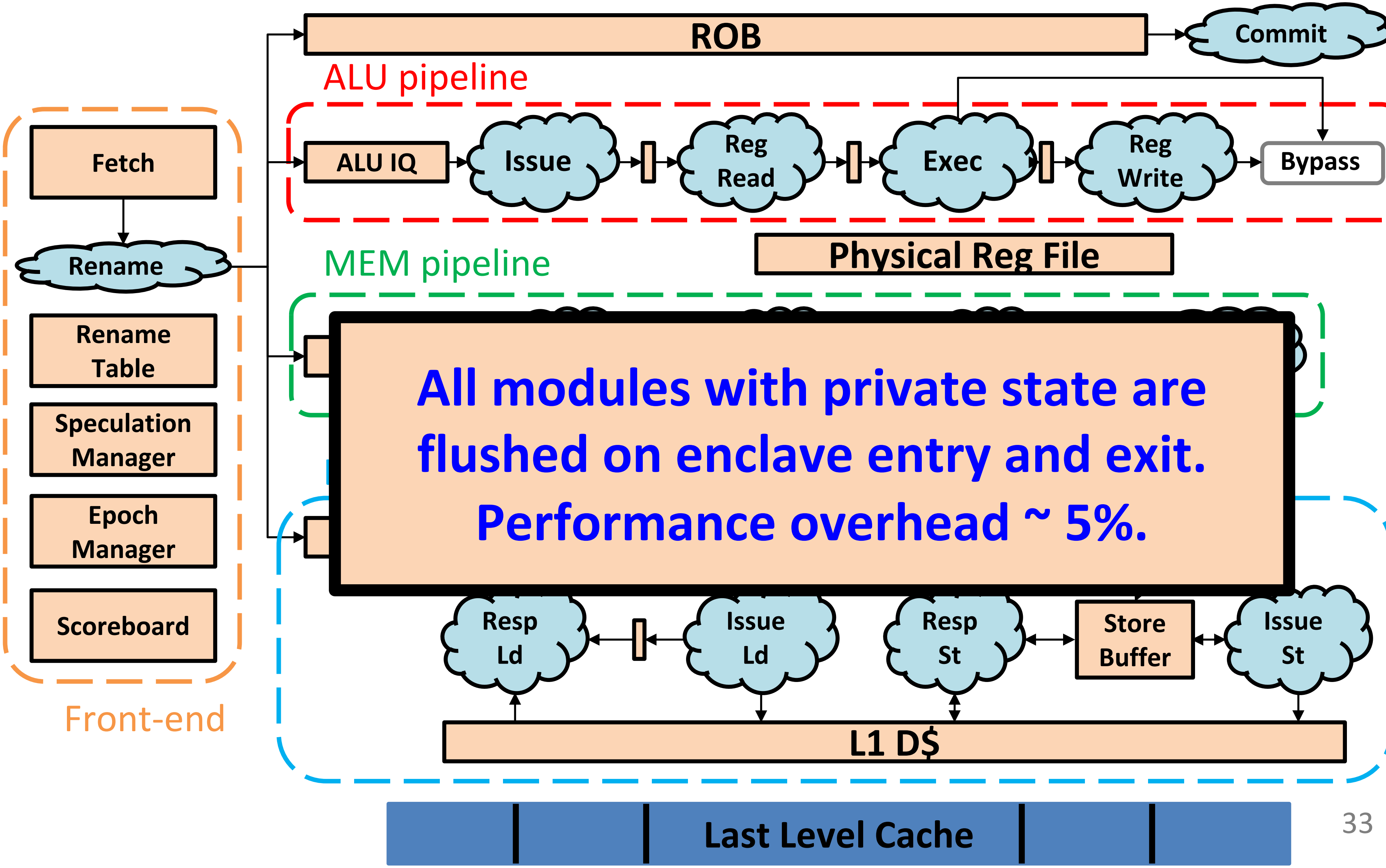


# RiscyOO Processor



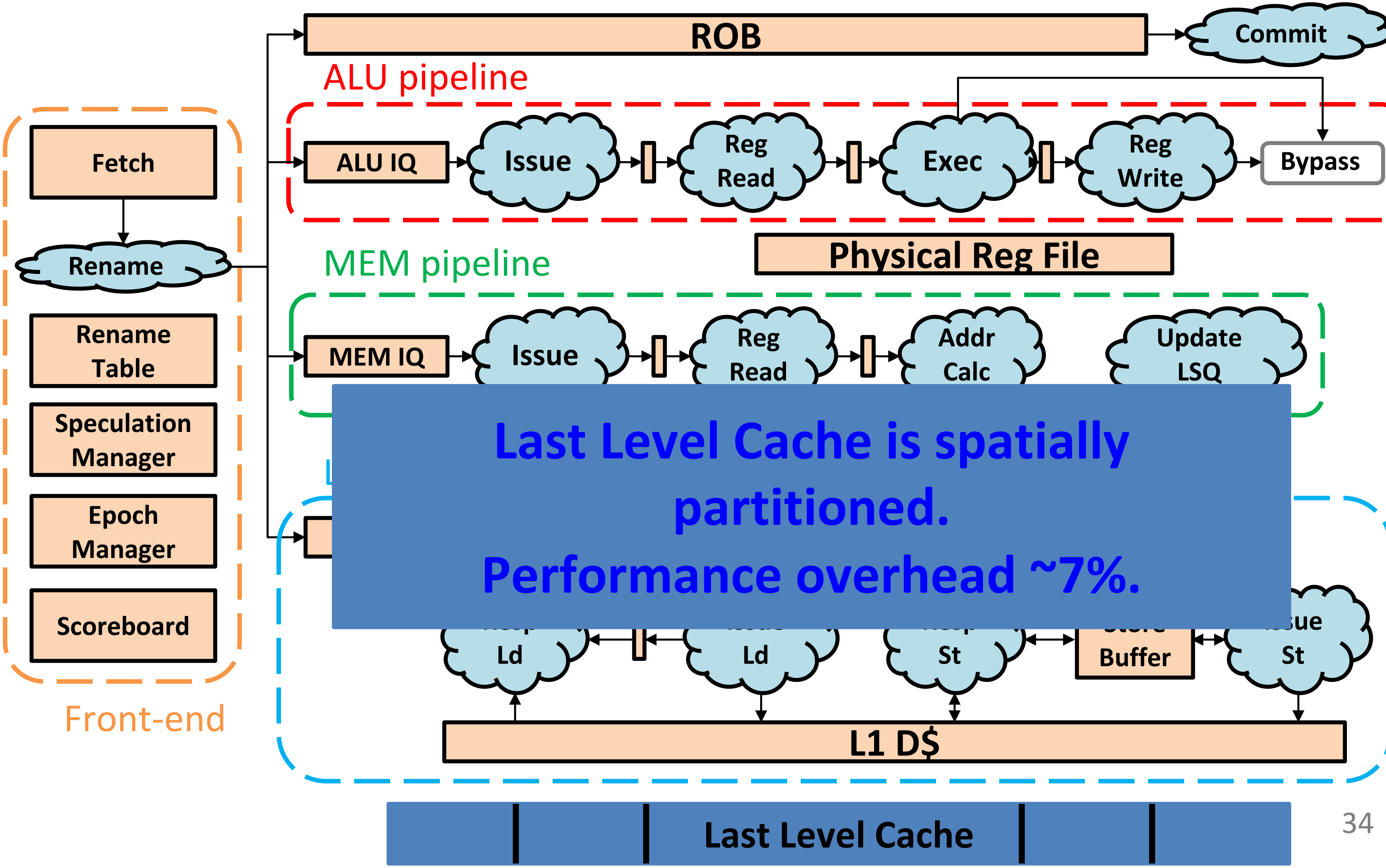

**Not speculating during entire program results in average 3X slowdown!**

# MI6 Processor




**All modules with private state are flushed on enclave entry and exit. Performance overhead ~ 5%.**

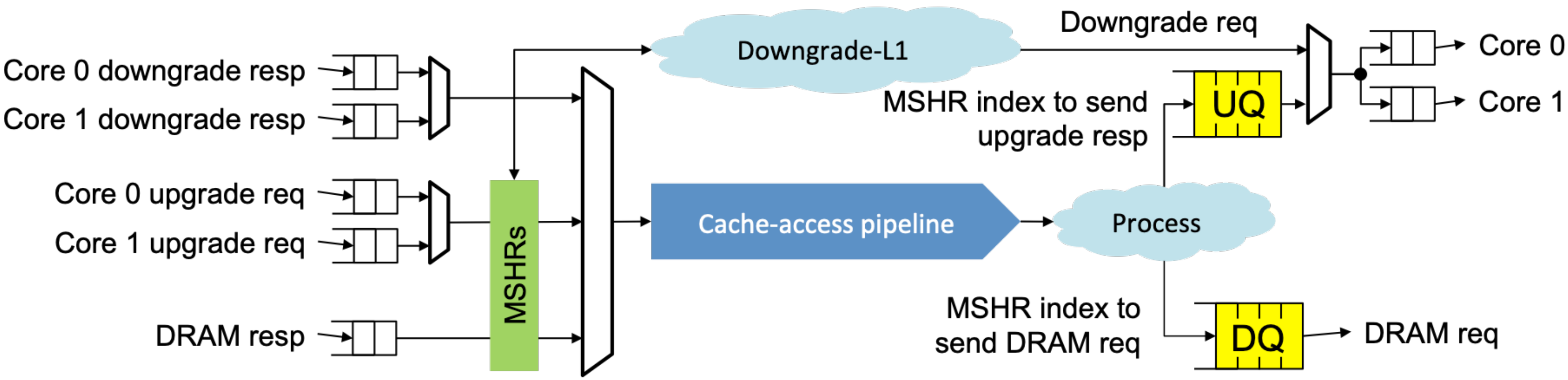
# MI6 Processor



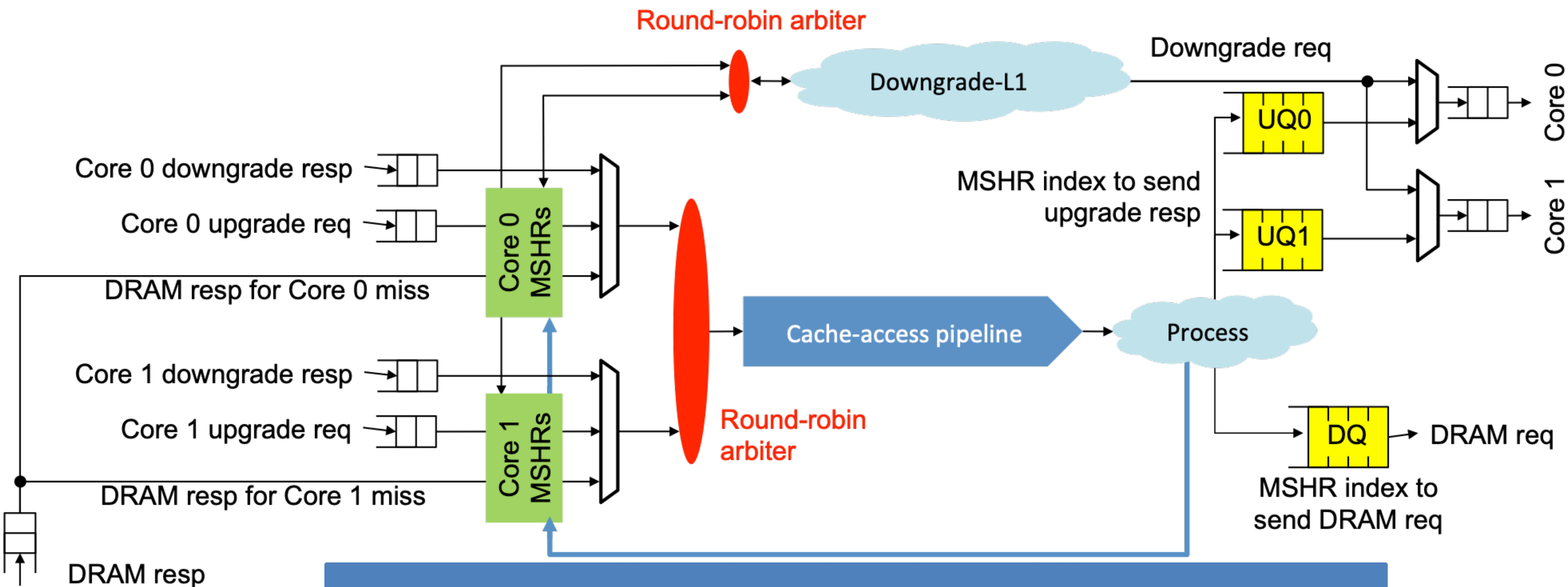

**Last Level Cache is spatially partitioned.**

**Performance overhead ~7%.**

# Leaky Cache Hierarchy

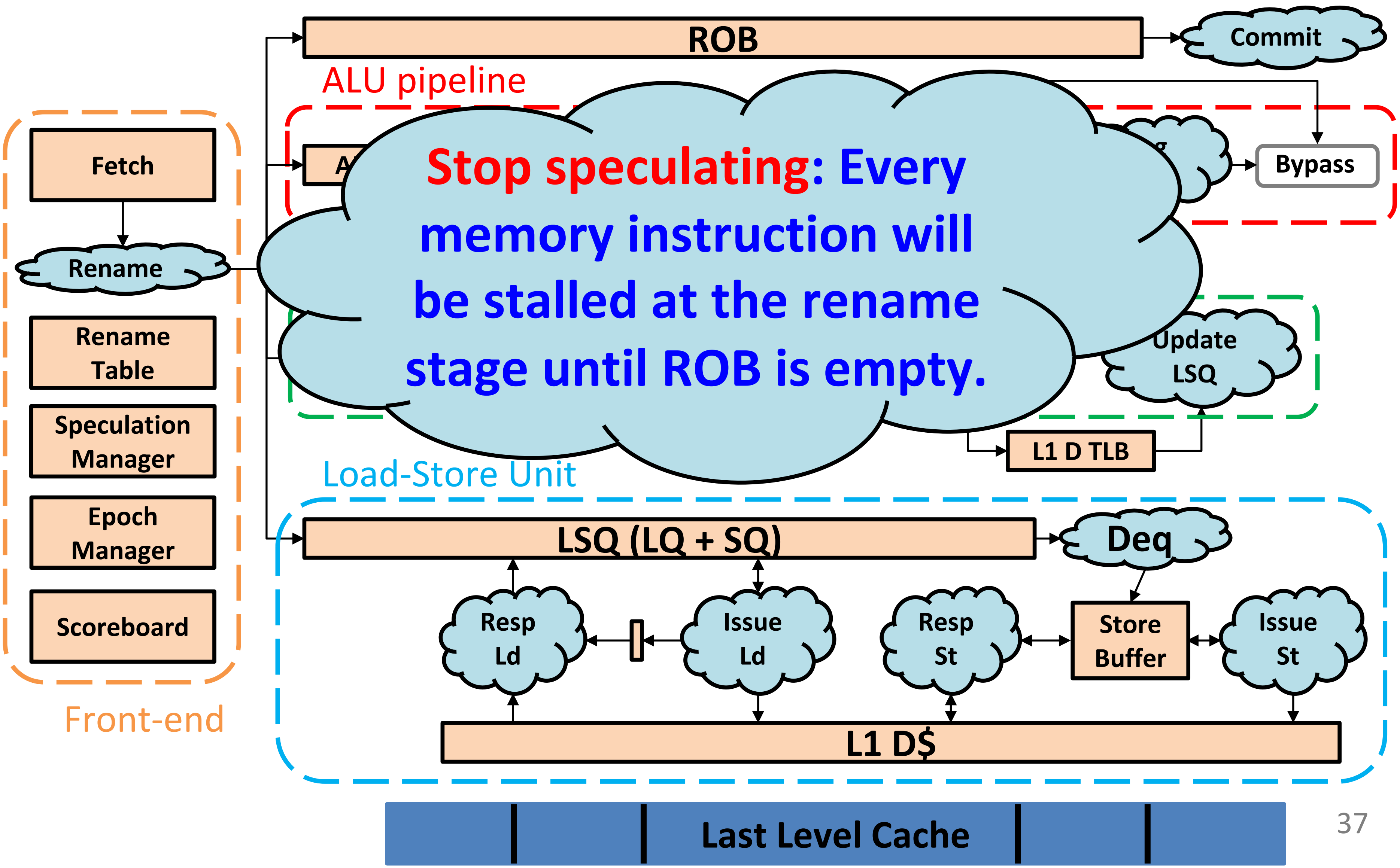


# Timing Independent Cache Hierarchy

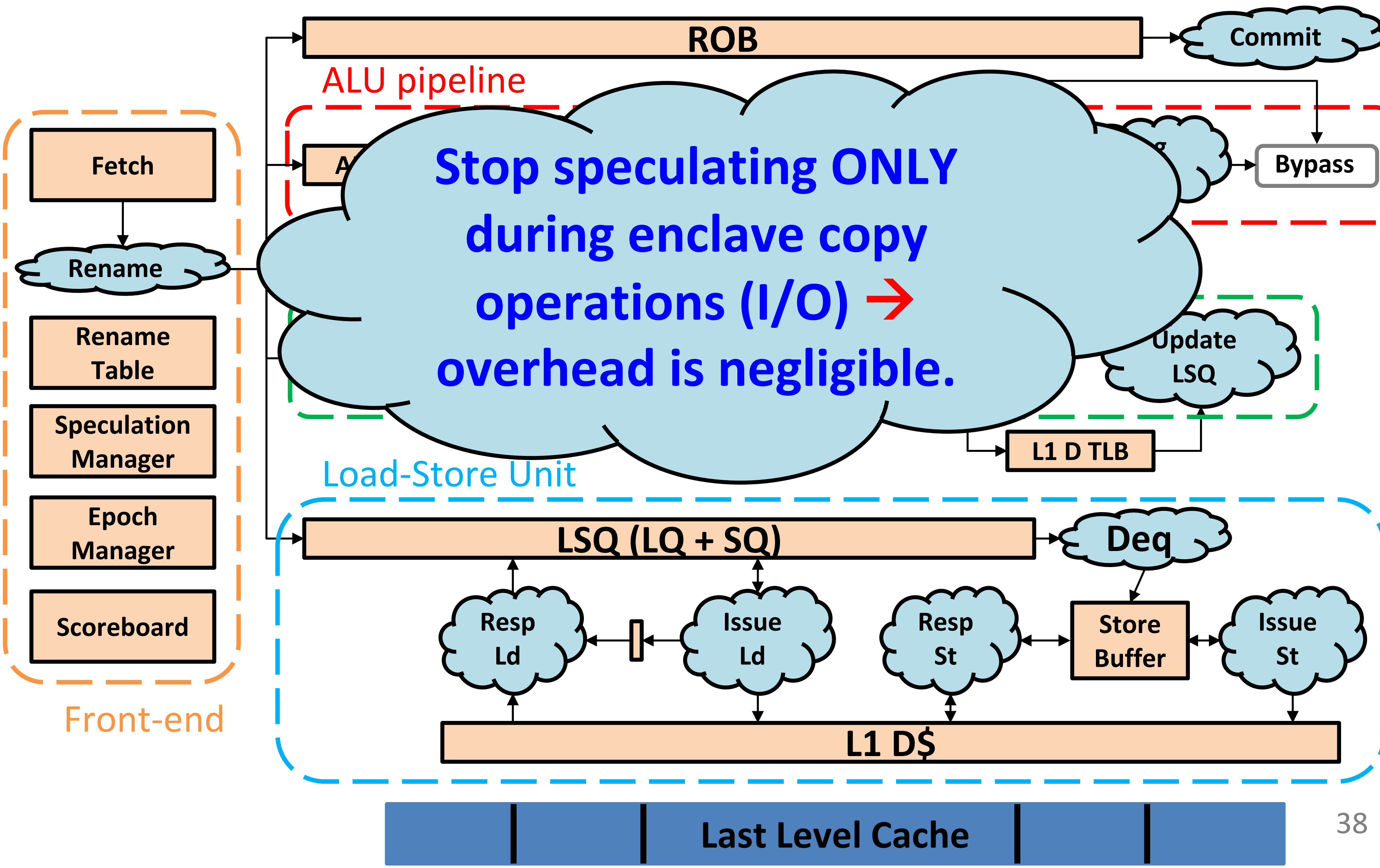



**Fair LLC Arbiter.**
**Performance overhead ~8%.**

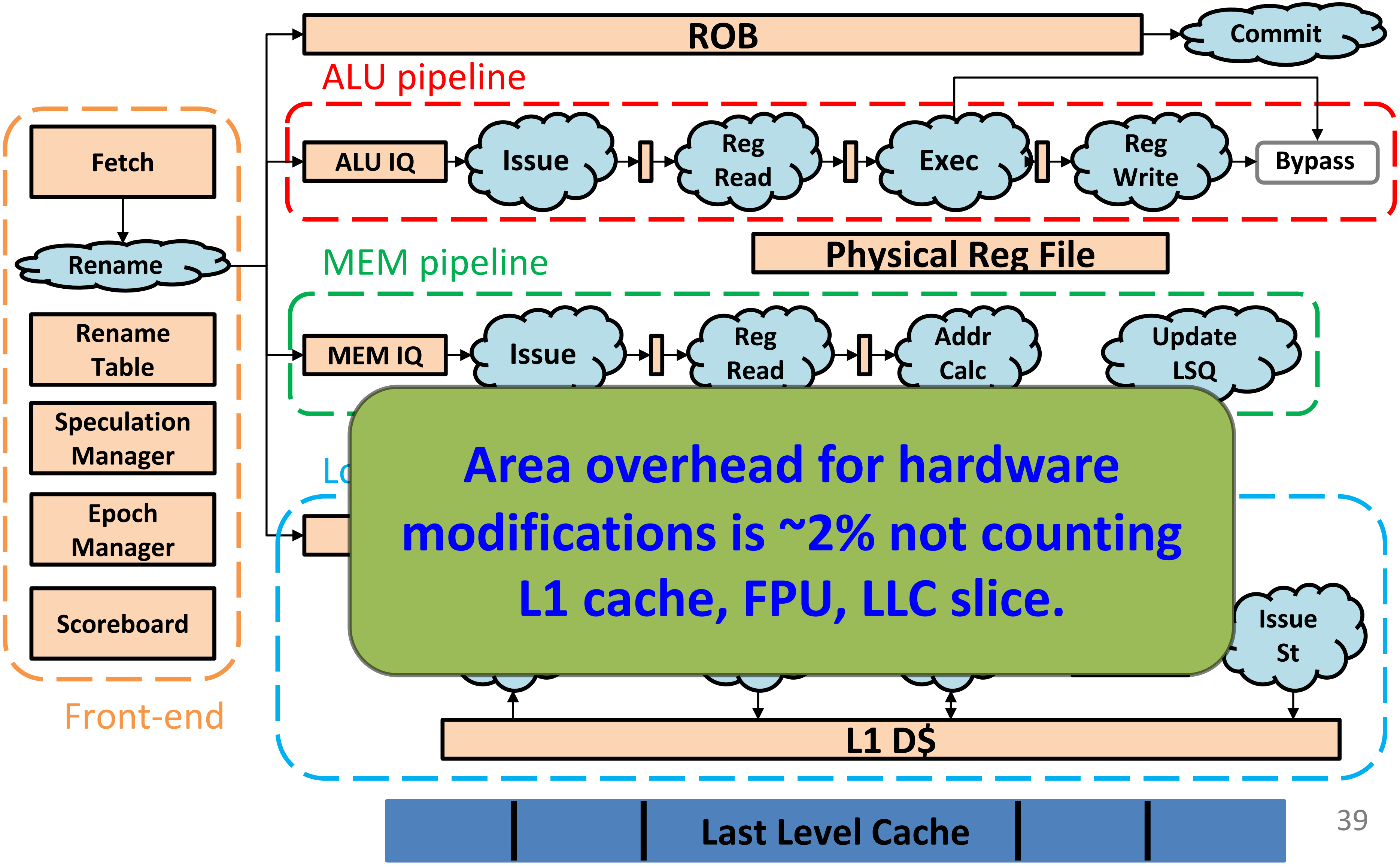
# MI6 Processor



# MI6 Processor

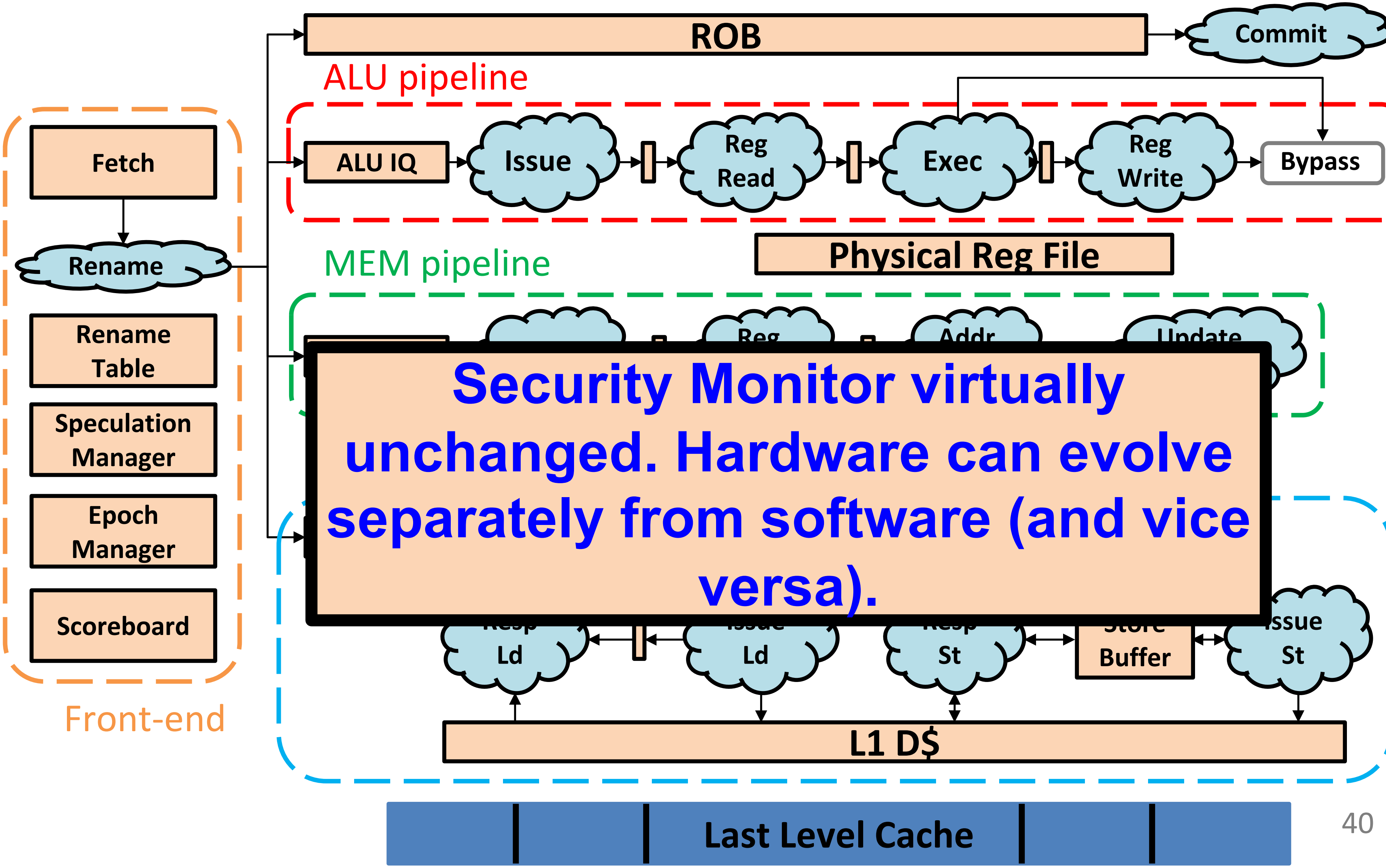


# MI6 Processor




**Area overhead for hardware modifications is ~2% not counting L1 cache, FPU, LLC slice.**

# MI6 Processor




**Security Monitor virtually unchanged. Hardware can evolve separately from software (and vice versa).**

# Challenge: Expressivity

- ~15% performance overhead for enclaves
- Enclaves trade expressivity for security
  - Cannot make system calls directly since OS can't be trusted to restore an enclave's execution state
  - Enclave's runtime must ask the host application to proxy file system and network I/O requests
  - **What syscall functionality should the enclave's runtime provide?**

# Challenge: Adaptivity

- Runtime decisions based on sensitive data leak information through timing: *completion time, resource usage*
- Crypto to the rescue?
  - Secure demand paging using page-level memory encryption, integrity verification and ORAM
  - **Secure and efficient dynamic memory allocation in enclaves an open problem**

# Challenge: Interaction

- Interaction with the outside may leak information
  - Public schedule for interaction does not leak


- **Can we bound leakage of adaptive interactions with users, other programs?**

# Challenge: (Formal) Verification

Open Source → Independent Verification

Properties of Enclaves:

Measurement := Different enclaves have different measurements (also inverse)

Integrity := Modelled attacker cannot affect enclave state

Confidentiality := Modelled attacker cannot observe enclave state

# Modeling the Adversary

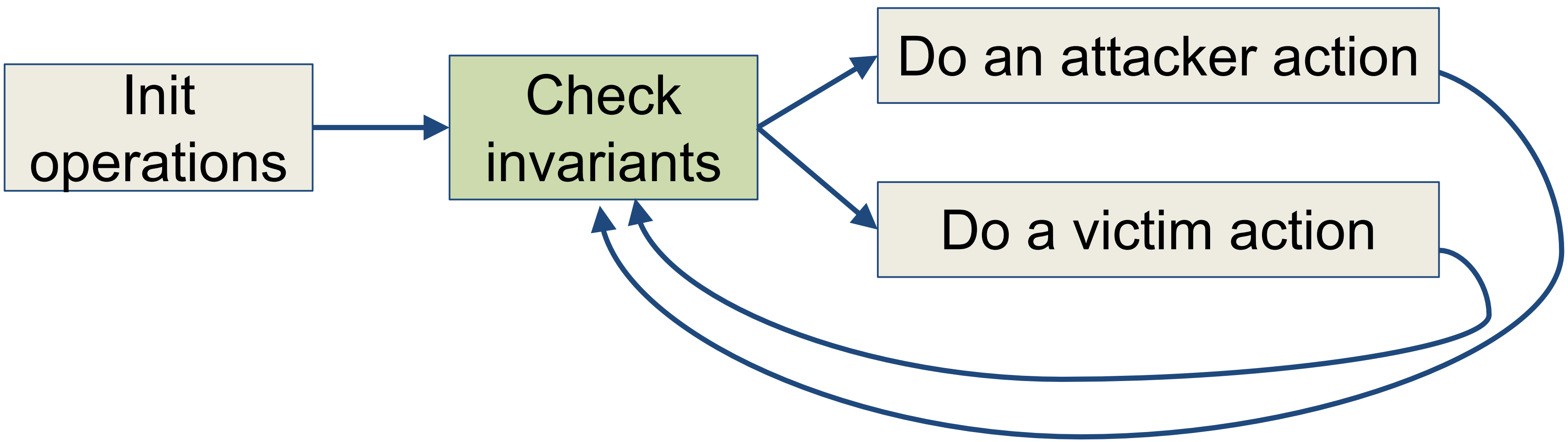
Adversary := set of ops an attacker can use to tamper with or observe enclave state. Any combination of these can be used at any time.

Threat model := U(observation function, tamper function, model initial state)

**Specify non-interference properties or invariants that execution should satisfy**

# Invariants and Non-Interference

The proof describes a CFG with "forks". Search this graph for a path that violates an invariant.



# Summary: Desiderata for Single-Chip Secure Processor

- Open source
- Formally verified (small) TCB
- Secure against all practica… …e attacks
- Secure against phys… …ks on memory
- Enhanced phy… …urity against invasive attacks
- Mi… …formance overhead

CLOSE(R) TO REALITY!

# Acknowledgements


- Edward Suh
- Victor Costan
- Ilia Lebedev
- Chris Fletcher
- Ling Ren
- Albert Kwon
- Sanjit Seshia
- Pramod Subramanyan
- Arvind
- Thomas Bourgeat
- Andrew Wright
- Sizhuo Zhang
- Kyle Hogan
- Jules Drean
- Rohit Sinha
- *NSF, DARPA, ADI, Delta*


Thank you for your attention!