Morpheus: A Vulnerability-Tolerant Secure Architecture Based on Ensembles of Moving Target Defenses with Churn

Mark Gallagher, Lauren Biernacki, et al.

ASPLOS'19

Mark Gallagher
University of Michigan

Lauren Biernacki
University of Michigan

Shibo Chen
University of Michigan

Zelalem Birhanu Aweke
University of Michigan

Salessawi Ferede Yitbarek
University of Michigan

Misiker Tadesse Aga
University of Michigan

Austin Harris
University of Texas at Austin

Zhixing Xu
Princeton University

Baris Kasikci
University of Michigan

Valeria Bertacco
University of Michigan

Sharad Malik
Princeton University

Mohit Tiwari
University of Texas at Austin

Todd Austin
University of Michigan

- Andrew S. Tanenbaum and Todd Austin, Structured Computer Organization, 6th Ed., Pearson, 2012.

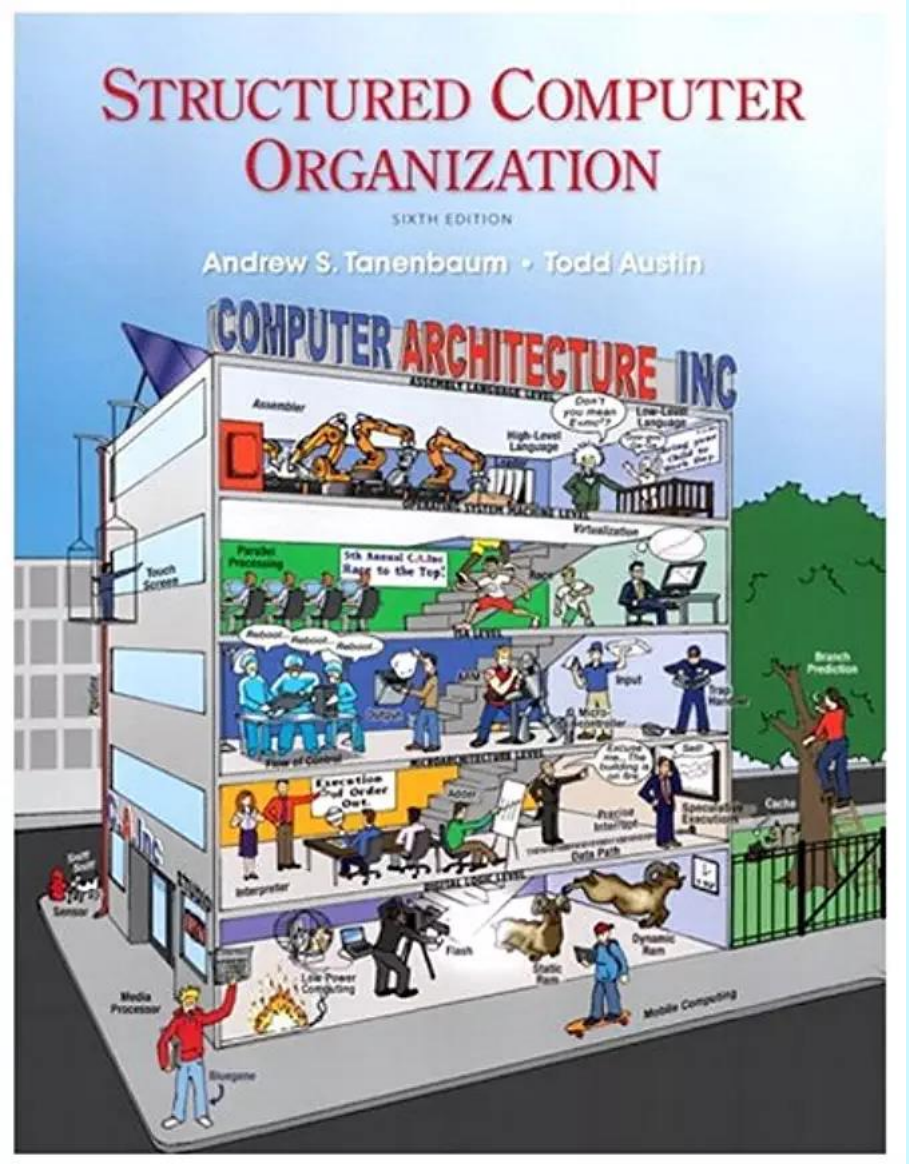


Image from: https://www.amazon.com/Structured-Computer-Organization-Andrew-Tanenbaum/dp/0132916525

Outline

- Introduction
- Threat Model
- The Morpheus Secure Architecture
- Evaluation: Security and Performance
- Related Work
- Conclusions
- Future Directions

An Observation

- “We observe that normal programs utilize defined program-level semantics, while malicious programs lean heavily on undefined semantics. ”
- “static undefined semantics”

An Example

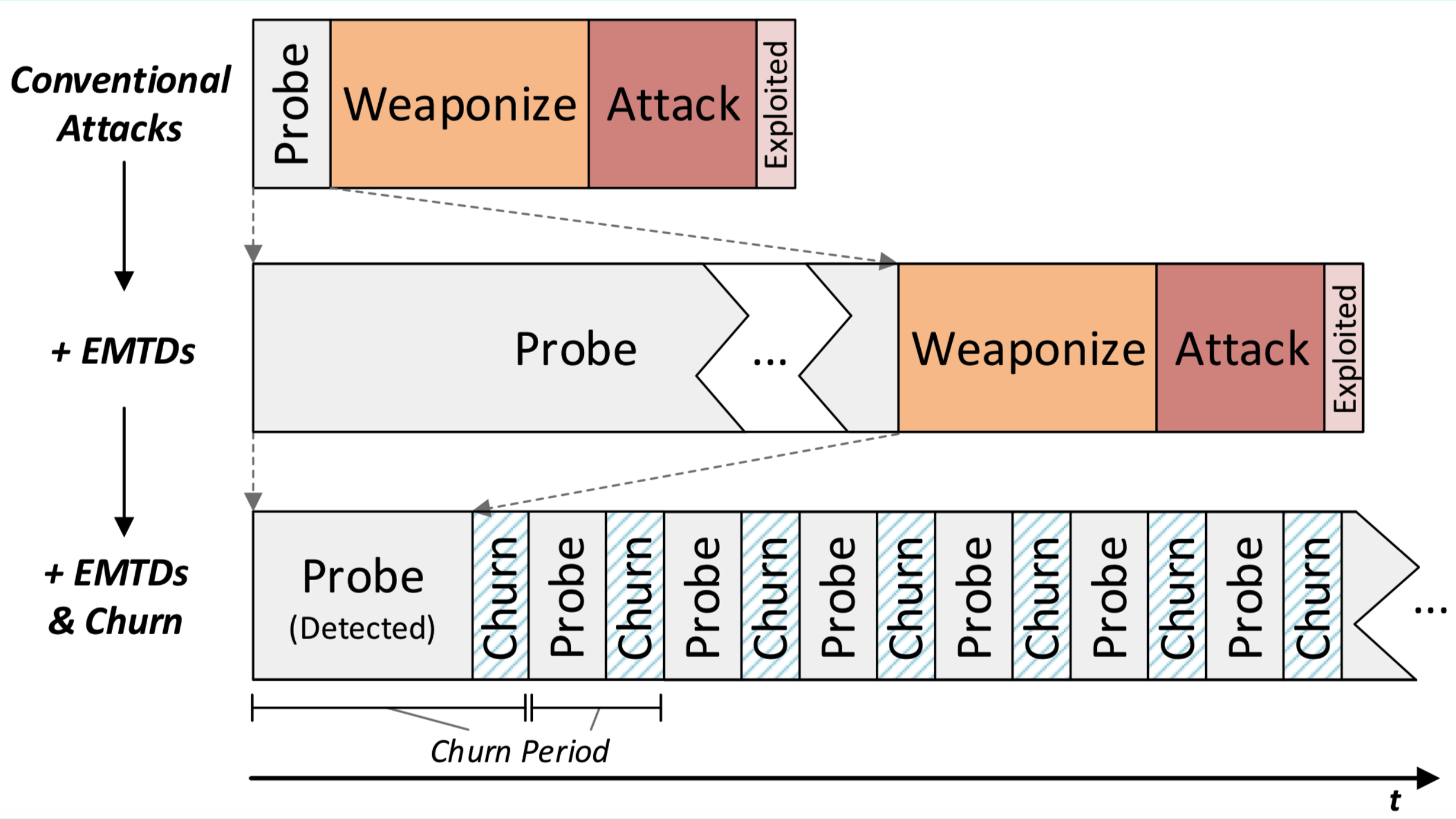
```
void target() {
  printf("You overflowed successfully, gg");
  exit(0);
}
void vulnerable(char* str1) {
  char buf[5];
  strcpy(buf, str1);
}
int main() {
  vulnerable("ffffffffffffffff\xf0\x03\x02\x01");
  printf("This only prints in normal control flow");
}
```

Review: Undefined Behaviors in C

- Dividing by 0 (`a / 0`)
- Uninitialized variables (`int a; printf("%d", a)`)
- Dereferencing NULL pointer (`*(int *)NULL`)
- Out of bound access (`"gg"[3]`)
- Signed integer overflow (`INT_MAX + 1`)
- Shift amount out of range [0, bit-width) (`1 << 64`)
- The evaluation order of sub-expressions (`++i + i++`)
- …

Introduction

- This paper introduces a secure system design approach called **ensembles of moving target defenses (EMTDs) with churn**,
- constructs a secure system that is transparent to normal programs but intentionally hostile to malicious programs,
- and present **Morpheus**, a RISC-V-based system that incorporates EMTDs with churn to thwart control-flow attacks.


Conventional Attacks
Probe
Weaponize
Attack
Exploited
+ EMTDs
Probe
...
Weaponize
Attack
Exploited
+ EMTDs & Churn
Probe
(Detected)
Churn
Probe
Churn
Probe
Churn
Probe
Churn
Probe
Churn
Probe
Churn
Probe
Churn
...
Churn Period
t

Threat Model

- Morpheus is designed to mitigate **control-flow attacks**, many of which utilize memory exploits.
- Assuming the following about the attacker:
  - i) cannot physically threaten the system;
  - ii) cannot manipulate the system's boot sequence;
    cannot anticipate the output of the random number generator;
  - iii) interact with the system via an interface (network or keyboard);
  - iv) cannot modify the original binary;
  - v) able to locate a memory corruption or disclosure vulnerability in the target program to exploit.

The Morpheus Secure Architecture

- Domain Tagging Mechanism and Attack Detector
- Two Moving Target Defenses:
  - Pointer Displacement
  - Domain Encryption
- Churn Unit

The Morpheus Secure Architecture

- Components hashed with diagonal lines augment the baseline RISC-V system to support Morpheus defenses.
- The dotted line is a bus used for churn control signals and transmitting keys.

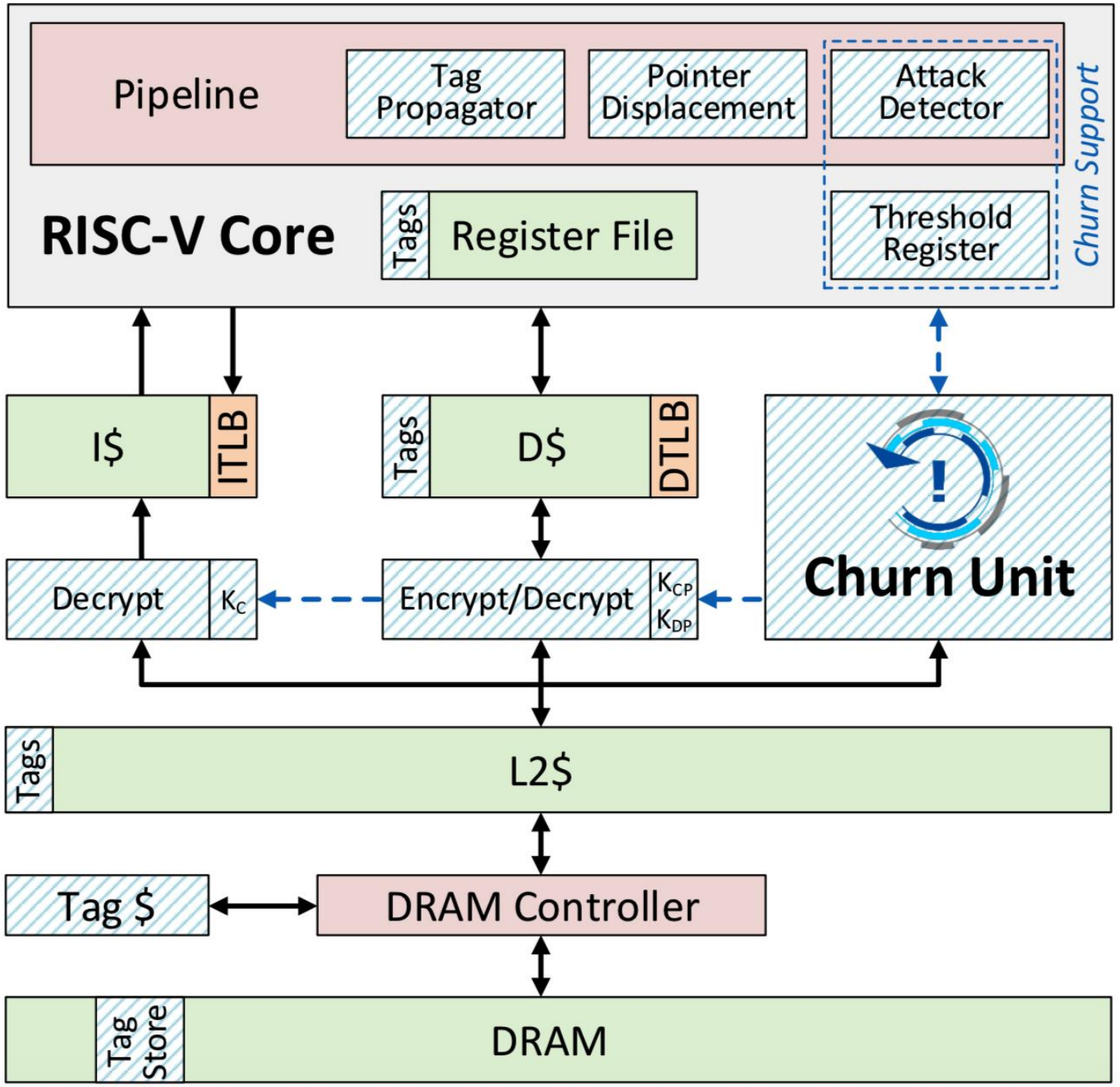
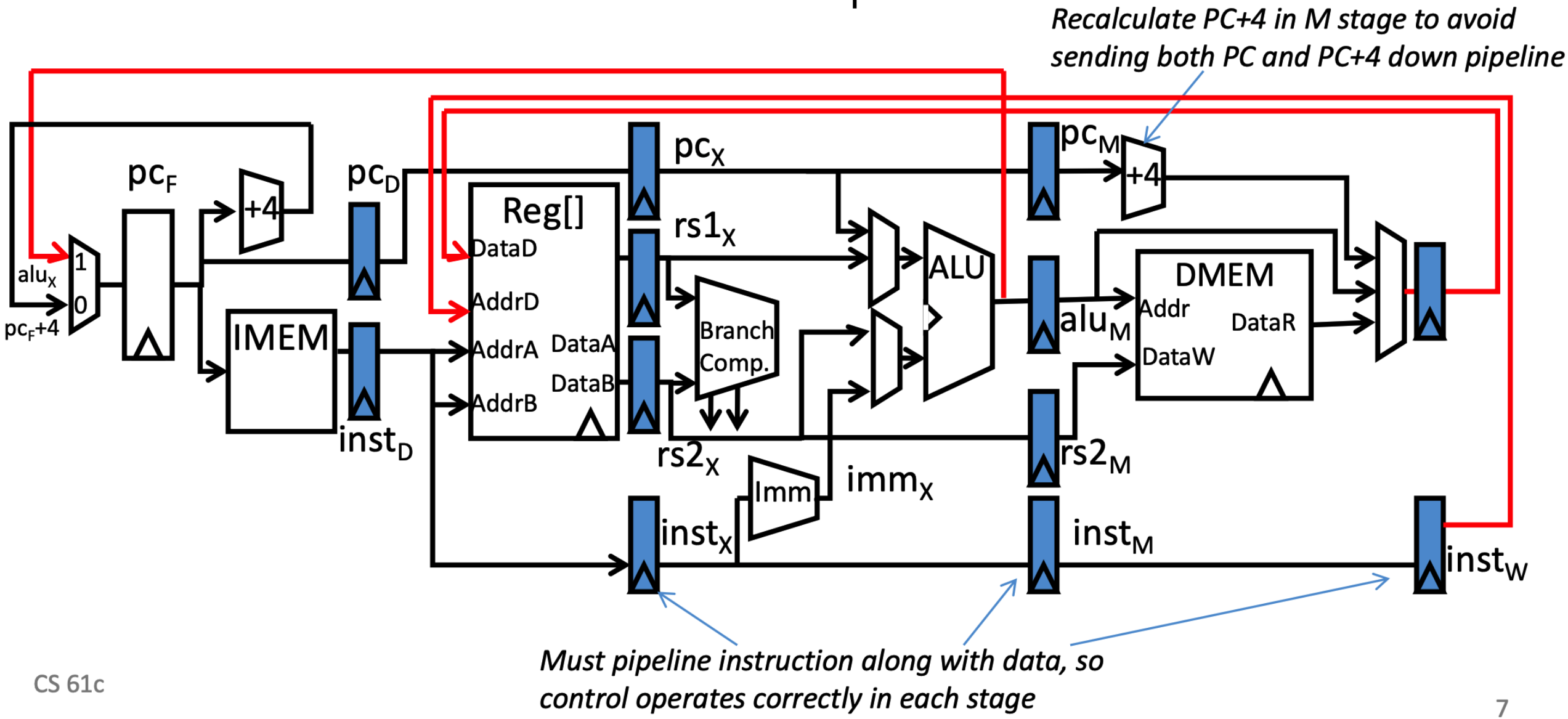


Image from: http://www-inst.eecs.berkeley.edu/~cs61c/fa17/lec/13/L13%20Pipelining%20(1up).pdf

Review: The RISC-V Pipeline



Domain Tagging Mechanism

- Morpheus tracks four distinct domains using 2-bit domain tags: code (C), code pointers (CP), data pointers (DP), and other data (D).
- The pipeline is responsible for propagating tags.
- Initial tag values come from the compiler.
- The microarchitecture is augmented to support tag storage.

LLVM Passes

- 2 LLVM passes.
- A global variable domain analysis labels each memory object in statically initialized data sections as data, a code pointer, or a data pointer.
- An instruction labeling pass identifies and labels instructions that initialize dynamically created memory objects (i.e. values on the stack, heap, and .bss segment).
- This produces a labeled binary and a domain tag file that contains the initial tags for memory objects.

Microarchitecture Modifications

- All registers are extended to include a 2-bit tag.
- One tag for each 64-bit aligned word, as pointers in the RISC-V RV64 ISA are 64 bits wide.
  - All data cache blocks are extended with 2-bits per 64-bit word, to store the additional domain tag bits with each cache block.
- Tags are cached in a **tag cache**.

Domain Tagging Mechanism (cont.)

- The domain tagging's propagation rules enforce closure for pointers under all computation; i.e. all computation with a pointer produces a pointer.
- twd2: what about well-defined pointer subtractions?

Attack Detector

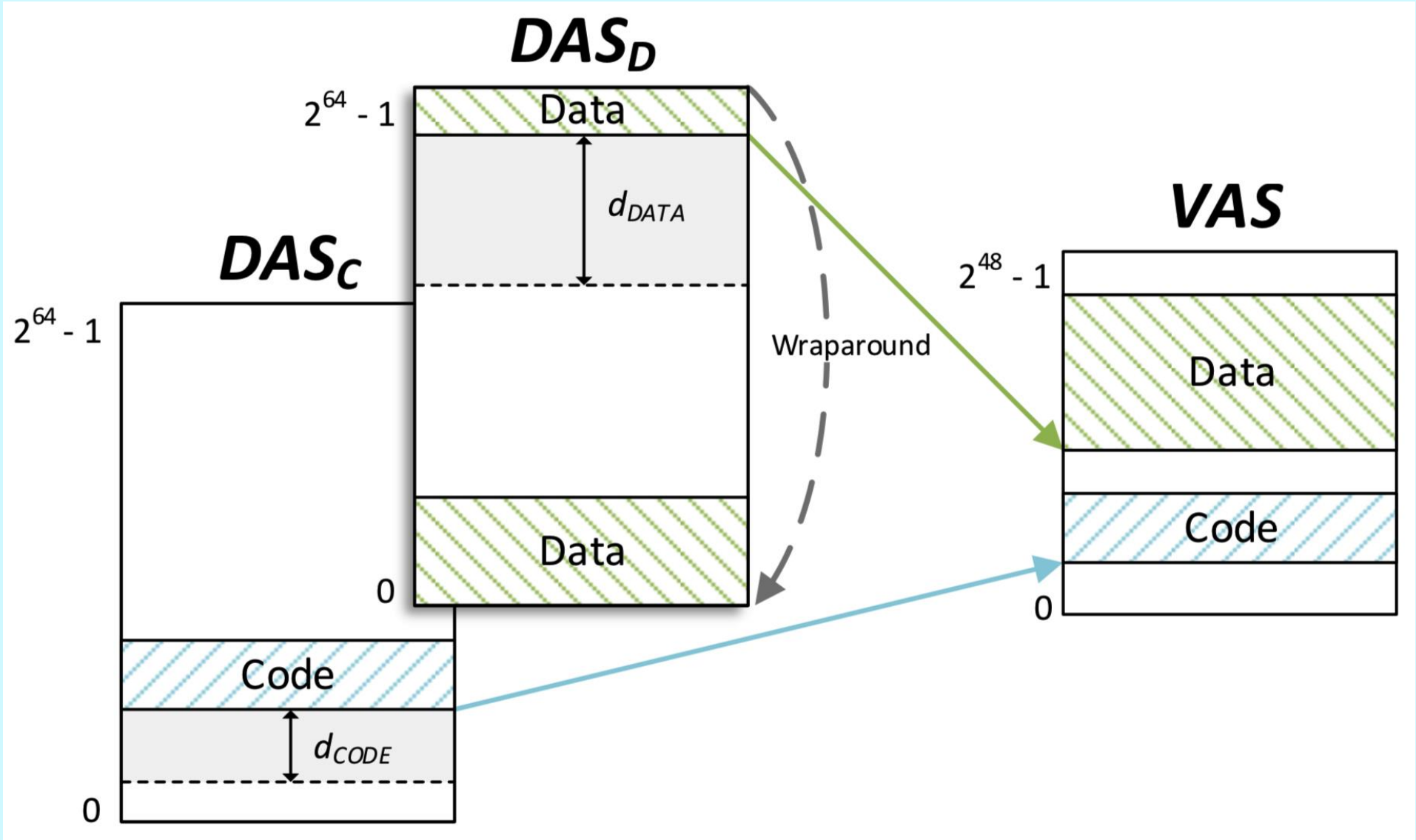
- Abort or Churn
- Policies:

	<OP>	Check Condition	Rule
ABORT	Execute	Insn.tag != C	Only execute C
	ANY	R1/R2.tag == C	No C in the pipeline
	JAL(R)	R1.tag != CP	Jump target must be CP
	LD/ST	R1.tag != DP	Address must be a DP
CHURN	COMPARE	R1.tag != R2.tag	No inter-domain compares
	ANY (not JAL(R))	R1.tag == CP	CP arithmetic suspicious
	ANY (not LD/ST)	R2.tag == DP	DP arithmetic suspicious, except add/sub D
	ANY	Overflow Occurs	Overflows are undefined
	SHIFT	Shift > RegWidth	Invalid shift is undefined

Pointer Displacement

- Virtually all control-flow attacks require knowledge of where memory objects reside.
- Morpheus utilizes pointer displacement to create two randomly displaced address spaces (DAS_C and DAS_D) above the virtual address space (VAS).
- This is implemented by incrementing all code pointers by d_{CODE}, and all data pointers by d_{DATA}.
- A program sees all pointers as displaced for its lifetime, including pointers in the registers, caches, and memory.

Pointer Displacement (cont.)



Microarchitecture Modifications

- In the decode stage, the displacement key is subtracted from the LOAD/STORE offset: offset - d_{DATA}.
  - Then in the execute stage, this delta is added to the base register to produce the effective address.
- A similar approach is used for JAL(R)/RET targets, except using the code pointer displacement (d_{CODE}).

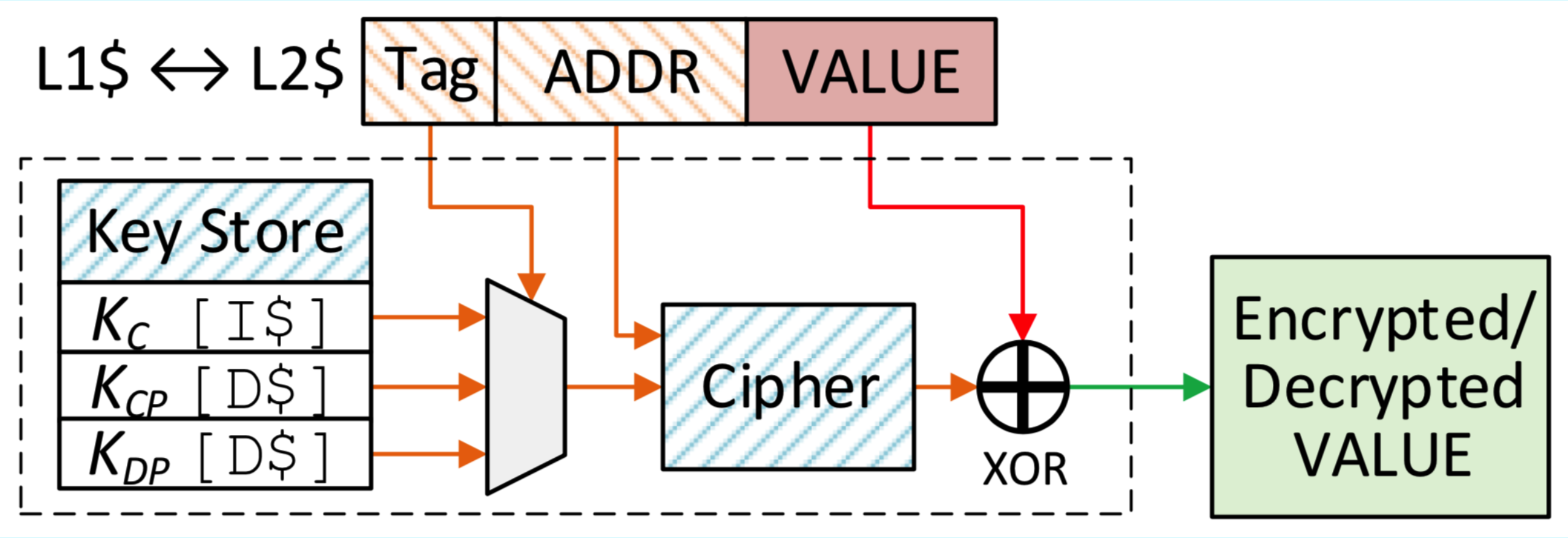
Pointer Displacement (cont.)

- This retains localities and incurs no performance penalties on the memory system.
- Shared memory?
- The defense permits a full 60-bits of entropy during DAS displacement.
- twd2: pointer additions? (???)
- twd2: PIC?

Domain Encryption

- Protected domains are decrypted when memory is read (load or instruction fetch) and encrypted when memory is written (store) between the L1-L2 boundary, keeping the L2 cache and DRAM encrypted.
- Code key K_C, code pointer key K_{CP}, or data pointer key K_{DP}.
- Encrypt the address, and then XOR it with the data.
- Cipher: $QARMA_7$-64-σ_1 (a low latency cipher, for Arm's Pointer Authentication technology)

Domain Encryption (cont.)



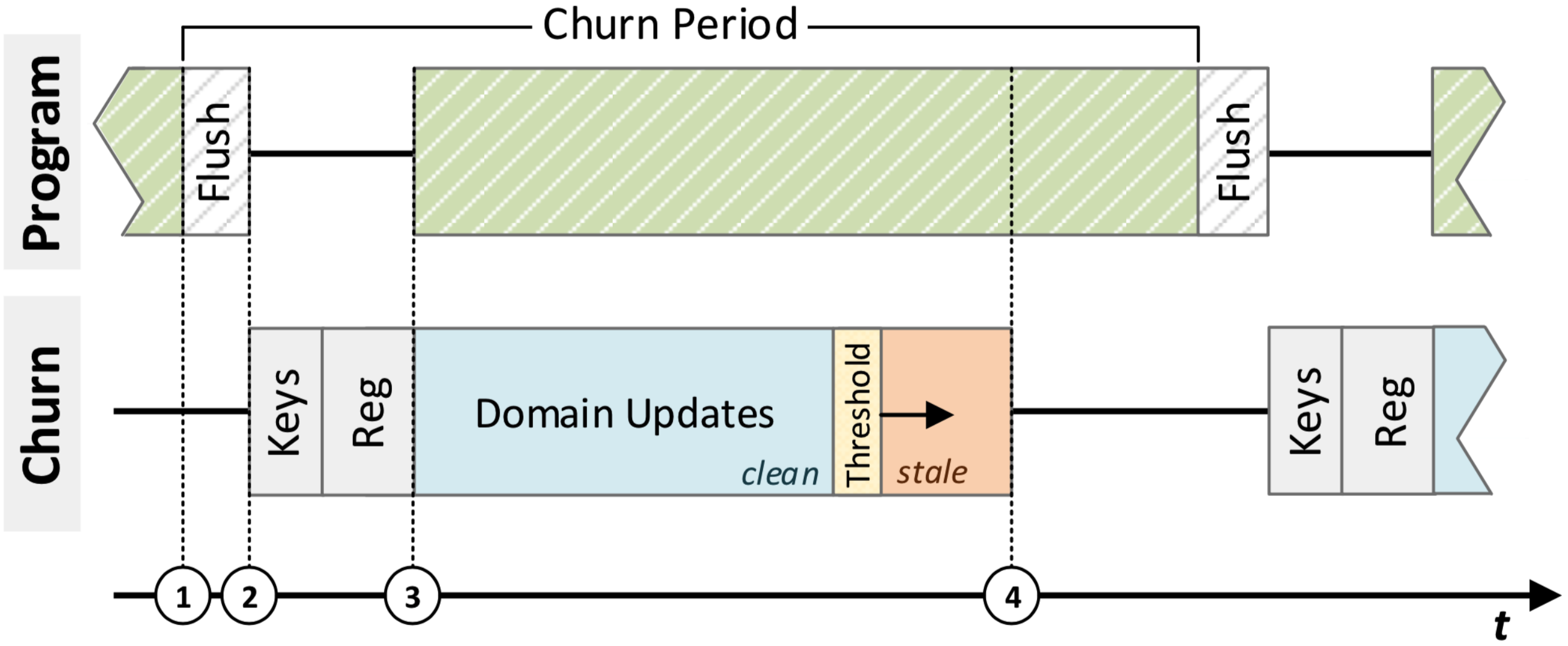
Domain Encryption (cont.)

- Since code and pointers are always **encrypted outside of the pipeline and L1 caches**, any attempt to exfiltrate them by writing them to an I/O location, or via DMA, RDMA, or cold-boot attack (all of which access DRAM or the L2 cache) will result in the capture of a useless encrypted instruction or pointer.
- twd2: shared libraries need to share encryption keys?

Churn Unit

- The churn unit implements re-randomization of code and pointers in coordination with the main core.
- State that has been processed to use the new keys is “**clean**”, while state using old keys and awaiting update is “**stale**”.
- **Re-encrypt** code and pointers, and **update** displacements.

Churn Unit (cont.)



Churn Unit (cont.)

- The churn unit maintains 4 invariants:
- i) all pipeline state (e.g., instructions and pointers in registers and latches) is clean,
- ii) all memory values **below** the threshold address are **clean**,
- iii) all memory values **above** the threshold are **stale**,
- iv) the memory value **at** the threshold address is currently **being processed** by the churn unit.

Support for Context Switching

- The OS can request the **current context** from the churn unit, which is encrypted with a boot-time private churn key.
- The encrypted context is passed to the kernel, which stores it in the kernel's **process control block**.
- The churn context contains the **threshold** register, the **old keys** for all defenses, the **new keys** for all defenses, and the **time** that the last churn cycle was initiated.

Evaluation: Experimental Framework

- Morpheus prototype is implemented on the RISC-V port of the **gem5 simulator**.
- Use **DRAMSim2** to model the memory system and assess the performance of tag scanning and churn operations.
- The churn unit is implemented as a simple FSM with access to the cache-coherent bus between the main core's L1-cache and the L2-cache.

Morpheus Microarchitecture Configuration

Core Type	MinorCPU (In-Order)
CPU Frequency	2.5GHz
Cache Line Size	64B
L1 Instruction Cache Size	32KB with 2-cycle latency
L1 Data Cache Size	32KB with 2-cycle latency
L2 Unified Cache Size	256KB with 20-cycle latency
Tag Cache Size	4KB

Evaluation: Security Analysis

- They ran tests from an ongoing port of the RIPE control-flow attack suite (stack overflow, heap overflow, and ROP attacks).
- Additional attacks: heap spray, format string, integer overflow, and back-call-site attacks.
- All attacks aimed to overwrite an existing return address or function pointer as a means to manipulate control flow.
- The Morpheus architecture stopped all of the attack classes from this paper's penetration testing suite.

Penetration Testing Results

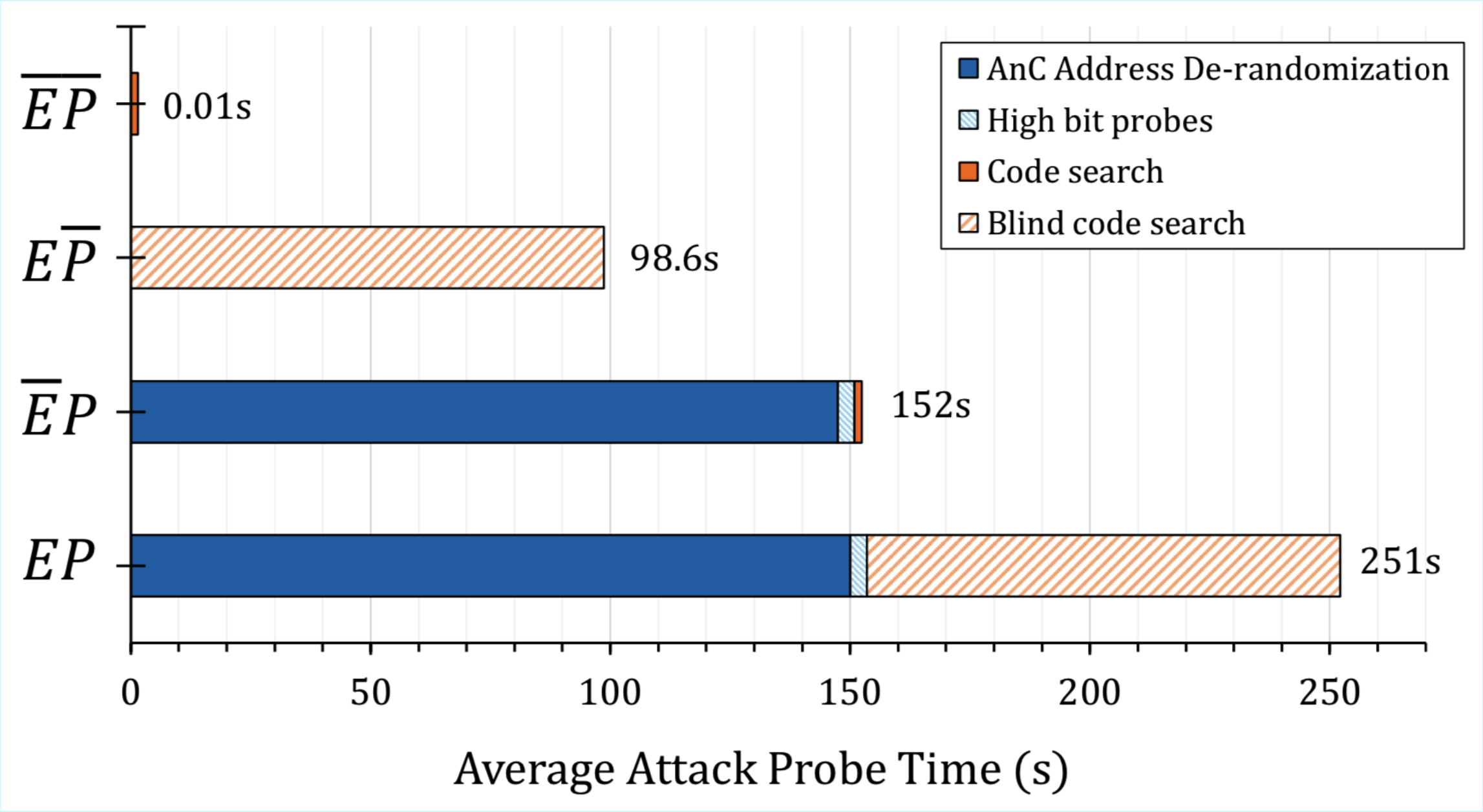
Attack	Defenses Engaged					Bits of Entropy
	E-C	E-CP	E-DP	P-CP	P-DP	
Stack Buffer Overflow [2]	X	✓	X	✓	X	188
Heap Overflow [22, 36]	X	✓	X	✓	X	188
Heap Spray [57]	X	✓	X	✓	X	188
Format String [52, 59]	X	✓	✓	✓	✓	376
Integer Overflow [12]	X	✓	✓	✓	✓	376
ROP [62]	✓	✓	✓	✓	✓	504
Back-Call-Site Attack [75]	✓	✓	✓	✓	✓	504

Attacking Morpheus

- The attack scenario involves a local program attacking a victim program (SPEC's gobmk) via an IPC interface, where the program under attack has exception reporting and crash recovery comparable to vanilla Linux.
- The goal of the attack is to call `system()`.

Attacking Morpheus (cont.)

- Churn and the attack detector are disabled.
- A single displacement, and a single key.
- $\overline{\text{EP}}$, with no defenses, E$\overline{\text{P}}$, with only encryption, $\overline{\text{E}}$P, with only pointer displacement, and EP, with all defenses engaged.


AnC Address De-randomization
High bit probes
Code search
Blind code search
$\overline{E}\,\overline{P}$
$E\,\overline{P}$
$\overline{E}\,P$
$E\,P$
0.01s
98.6s
152s
251s
0
50
100
150
200
250
Average Attack Probe Time (s)

Evaluation: Performance Impact

- **MiBench** and **SPEC'06**
- All programs were built using LLVM 5.0.0 for the RISC-V **RV64IMA** architecture with -O2. The benchmarks were linked against a Morpheus-built RISC-V Musl C library.

RISC-V Base Plus Standard Extensions

- A few base integer ISAs
  - RV32E, RV32I, RV64I
    - RV32E is 16-reg subset of RV32I
  - <50 hardware instructions in base (Similar to RISC-I!*)
- Standard extensions
  - M: Integer multiply/divide
  - A: Atomic memory operations
  - F/D: Single/Double-precision FI-point
  - C: Compressed Instructions (<x86)
  - V: Vector Extension for DLP (>SIMD**)

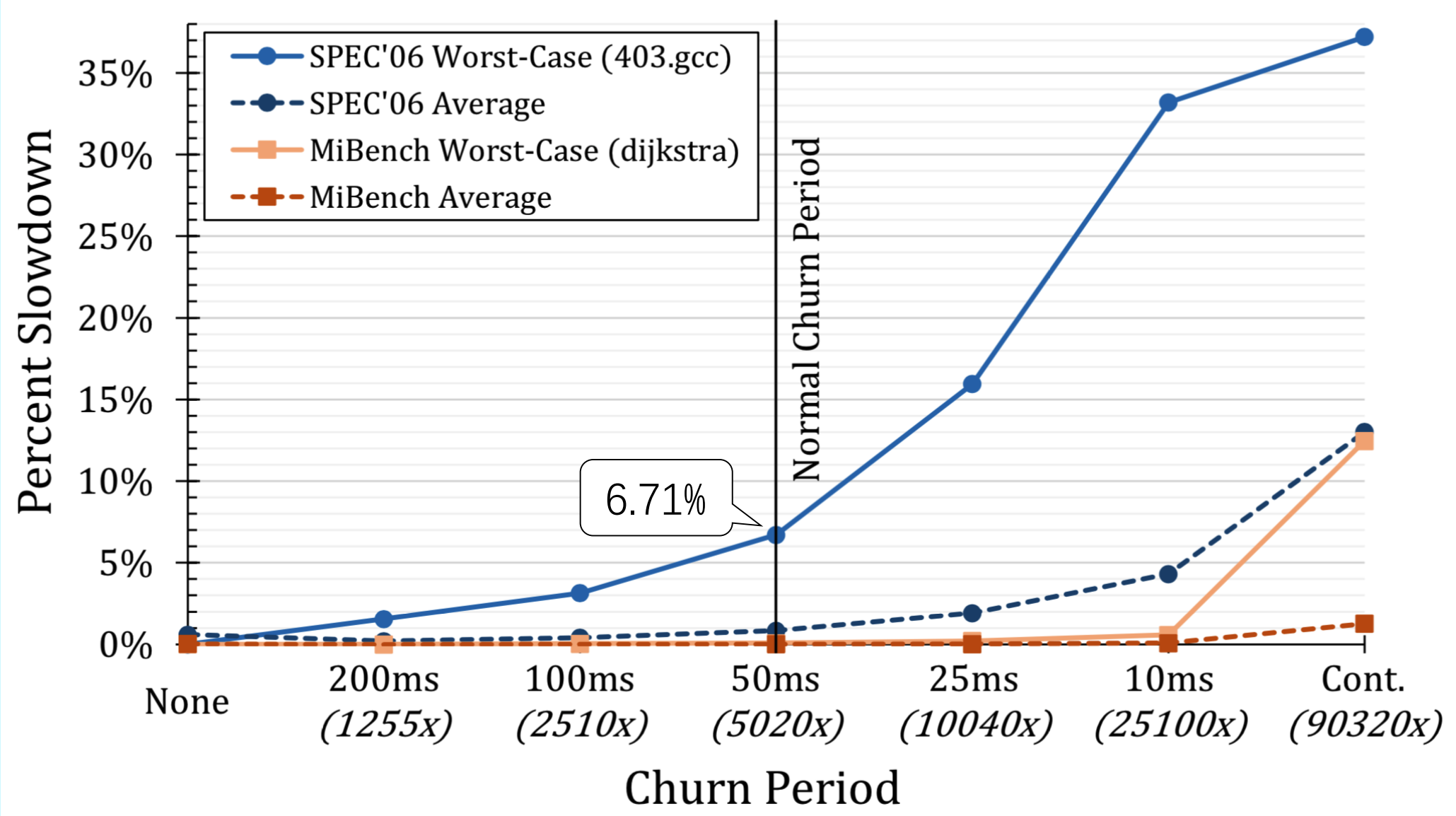
- Standard RISC encoding in fixed 32-bit instruction format
- Supported forever by RISC-V Foundation

G (general) = IMAFD

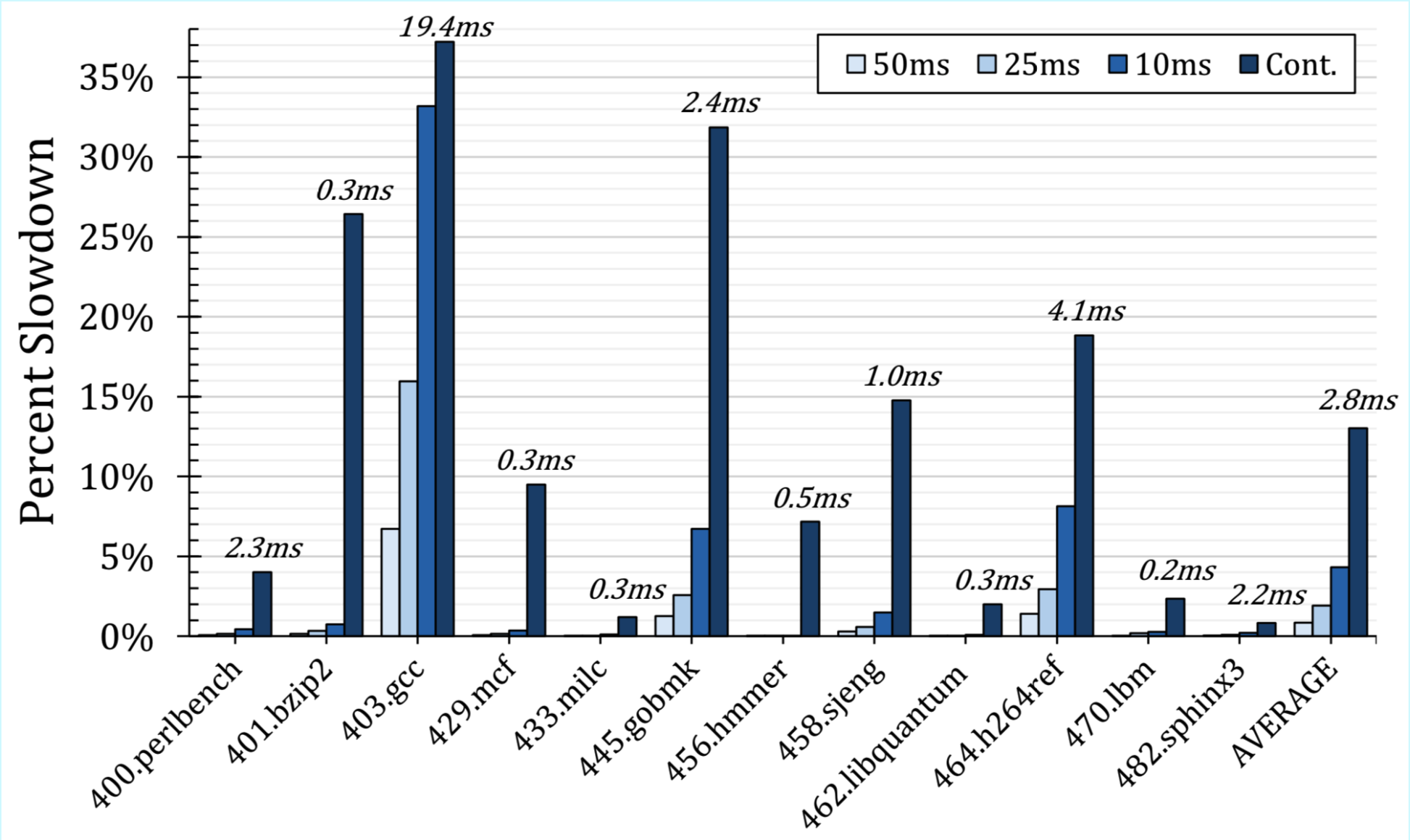
* "How close is RISC-V to RISC-I?" David Patterson, 9/19/17, ASPIRE Blog

** "SIMD Instructions Considered Harmful," David Patterson and Andrew Waterman, 9/18/17

Performance Overhead



SPEC'06 Performance Overheads



Related Work

- Previous encryptions are weak.
- Morpheus' hardware-based defenses have lower overhead while delivering more randomization.
- Previous tagging systems often face high false-positive rates, leading to the failure of benign program. Morpheus operates differently, as false-positive security violations from the attack detector only trigger a churn cycle.

	Name		Assets C	Assets CP	Assets DP	Entropy/Key Size	Runtime Churn	Avg. Overhead
Displacement	64-bit PaX ASLR [54, 55]		-	✓	✓	29-30 bits (48-bit vaddr)	No	3.6%
	TASR [9]		-	✓	✓	29-30 bits (48-bit vaddr)	At I/O Only	30-40%†
	Remix [17]		-	✓	X	ASLR + $\log_2$(basic blocks per func.) Apache on x86 (32-bit): 16+4 = 20 bits max	Random Interval	2.8% (one-time)
	RuntimeASLR [46]		-	✓	✓	28-48 bits (48-bit vaddr)	At `fork()` Only	0.5%
Encryption	PointGuard [18]		X	✓	✓	64 bits (weak XOR cipher, on 64-bit ISA)	No	10.0%
	CCFI [47]		X	✓	✓*	128 bits (strong cipher)	No	23.0%
	ASIST [53]		✓	✓*	X	32-128 bits (weak XOR cipher) or 32 bits (weak transposition)	No	1.0%
	Polyglot [68]		✓	X	X	163 bits (strong cipher)	No	4.6%
Enc. + Disp.	Shuffler [83]	Displacement	-	✓	X	27 bits (48-bit vaddr)	Fixed Interval	14.9%
		Encryption	X	✓*	X	64 bits (weak XOR cipher)		
	Morpheus	Displacement	-	✓	✓	60 bits per segment	Fixed Interval	0.9%
		Encryption	✓	✓	✓	128 bits per asset (strong cipher)		

Conclusions

- Traditional: find and fix vulnerabilities.
- EMTDs with churn (this paper): protects a system by randomizing the information assets that attackers need to craft successful attacks.
- These protections demonstrate a high level of protection against control-flow attacks with very low overheads.

Future Directions

- A similar approach could be adopted to protect against side-channel attacks, timing attacks, Rowhammer attacks, and even cache attacks.
- Explore what assets the attacker needs and then develop efficient mechanisms to boost uncertainty and stifle attacks.

Thanks.