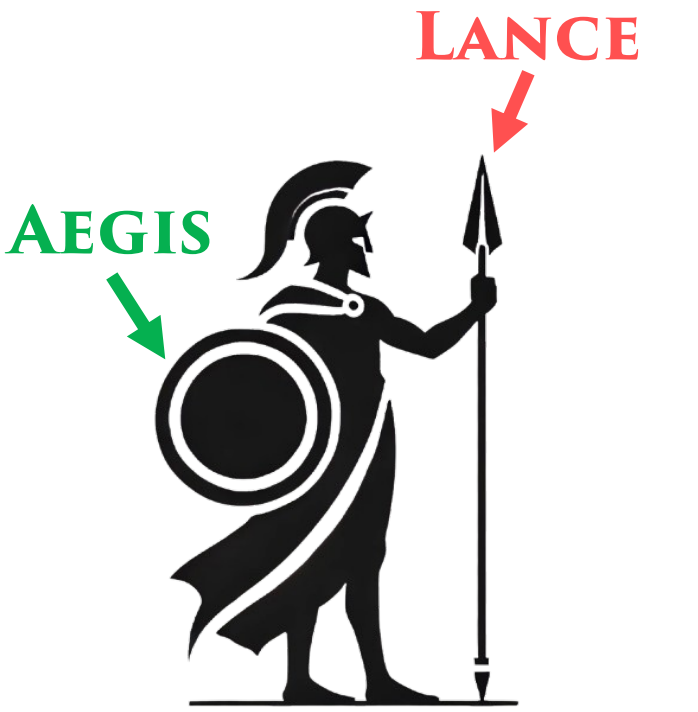
LÆGIS: Pinpointing and Addressing Performance Overheads of GPU-based Confidential Computing

Yang Yang, Adwait Jog
{yangyang, ajog} @virginia.edu

Insight Computer Architecture Lab

ISCA 2026

Summary



- We identify ***three inefficiencies*** in GPU-based CC when it comes to UVM.
- We propose **LÆGIS**, a design that ***opportunistically*** pre-encrypts pages in CC and uses secure **HBM** for IV management.
- **LÆGIS** introduces an **in-memory IV Bank** in 3D-stacked HBM, decoupling encryption from CPU–GPU synchronization and removing the need for integrity trees.

Average speedup of 2.22× (up to 3.13×) and 2.74× (up to 5.05×) under default and aggressive prefetching.

[PDF] https://adwaitjog.github.io/docs/pdf/yang_laegis_isca26.pdf
[GitHub] https://github.com/insight-cal-uva/laegis-isca26-artifact

Outline

- Introduction
- Background
- UVM Performance Dissection under CC
- LÆGIS: Overview & Implementation
- Evaluation
- Takeaway

Outline

- Introduction
- Background
- UVM Performance Dissection under CC
- LÆGIS: Overview & Implementation
- Evaluation
- Takeaway

Needs for Security and Privacy

- **Data is Private**
  - Client privacy (LLMs, …)
  - Regulated (GDPR, PII, …)

- **Data is Massive**
  - Outsource compute & data to public cloud Infra.

What techniques can be deployed to strengthen trust in cloud services?

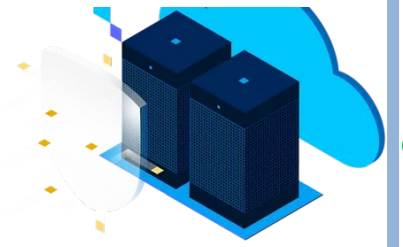
Protection Principles

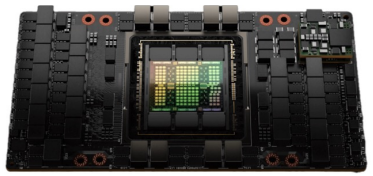
Hardware-based TEE

Isolation + Encryption

Trusted/Confidential Computing

Trusted Computing



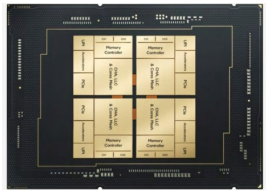


The first commercial GPU TEE

NVIDIA H100 CC **2023**

CPUs With TDX

Intel SPR **2023** Intel EMR **2023**



GPU-based CC

ARM TrustZone **2002**

Intel SGX 2015

AMD SEV **2016**

CC Consortium **2019**

Intel TDX **2020**

AMD SEV-SNP **2020**

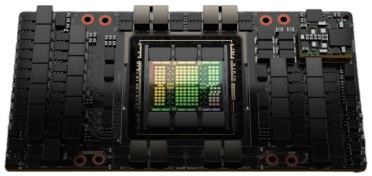
ARM CCA **2021**

RISC-V KeyStone **2019**

RISC-V Penglai **2020**

Credit/Source: ARM/INTEL/AMD/CCC/RISC-V/NVIDIA/Wiki

Trusted Computing



The first commercial GPU TEE

NVIDIA H100 CC **2023**

Intel TDX **2020**

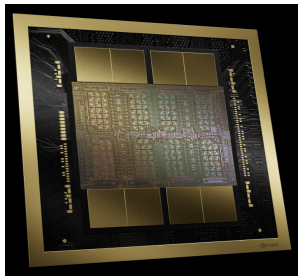
CPUs With TDX

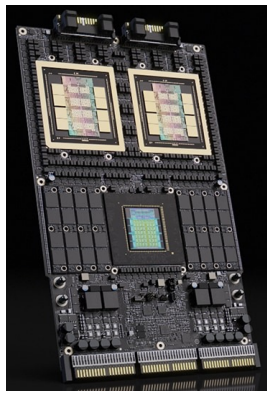
AMD SEV-SNP **2020**

ARM CCA **2021**

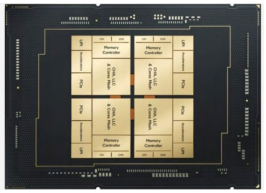
Intel SPR **2023**

Intel EMR **2023**





RISC-V Penglai **2020**



NVIDIA Blackwell **2024-2025**

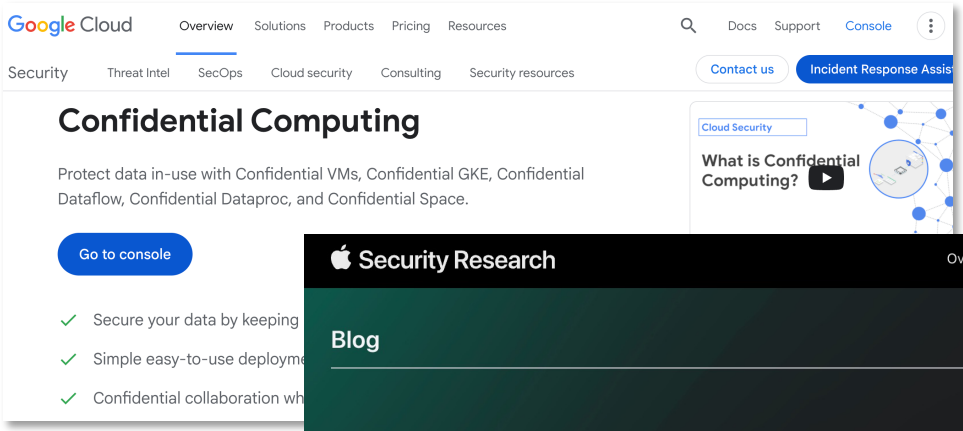
NVIDIA Vera-Rubin **2026**

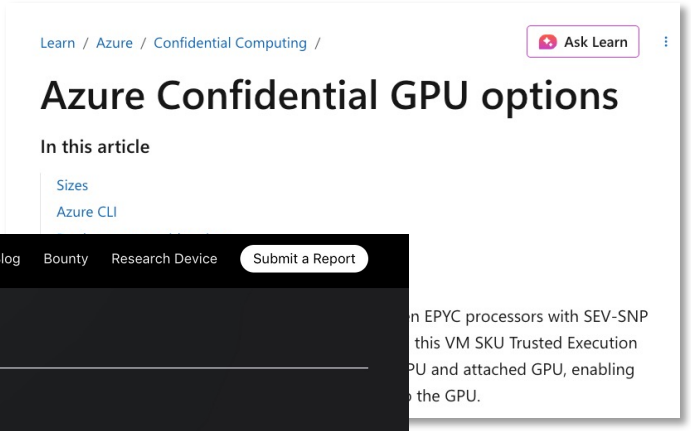
GPU-based CC

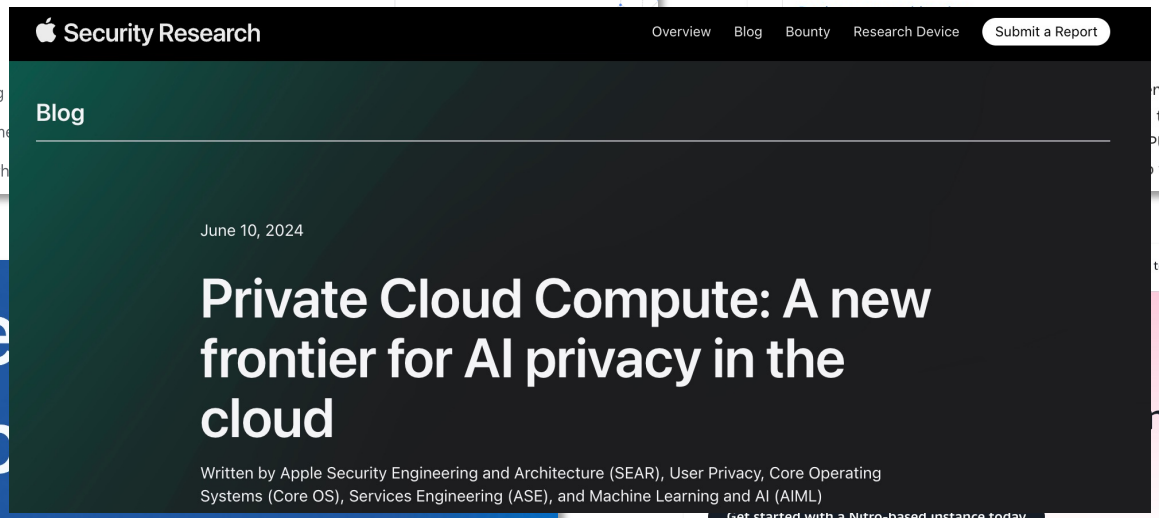
Rack-scale CC

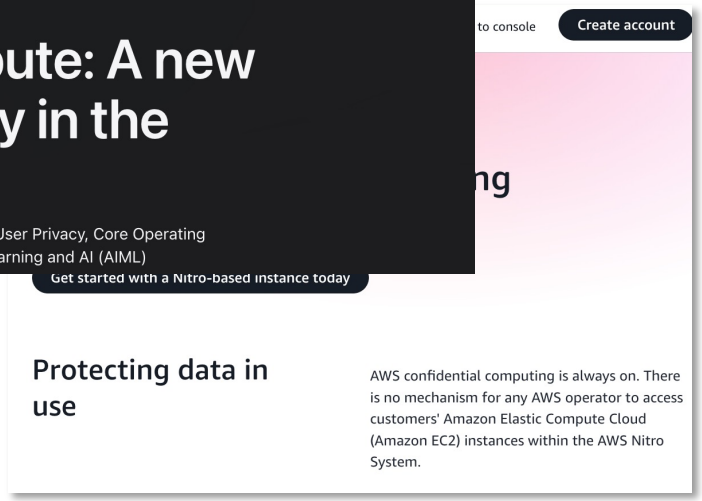
Credit/Source: ARM/INTEL/AMD/CCC/RISC-V/NVIDIA/WiKi

CC in Real World





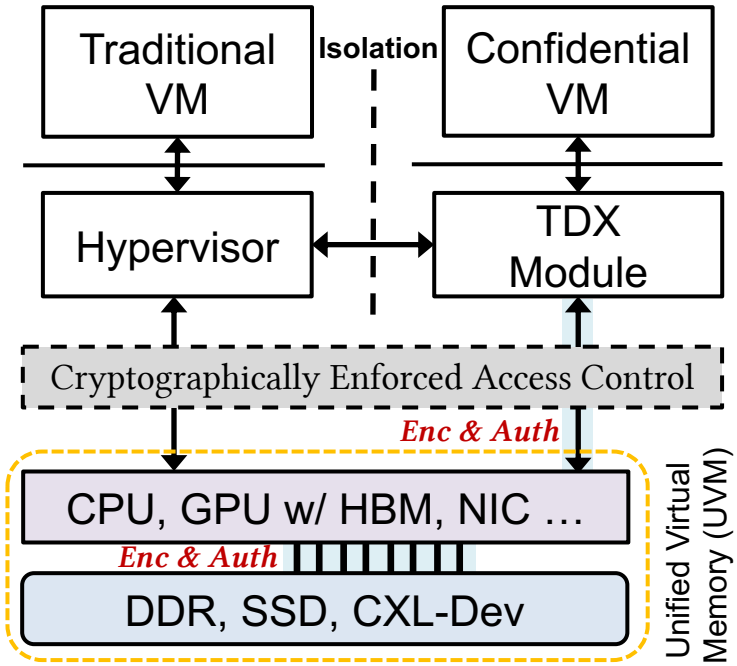




CC w/ GPU is practical

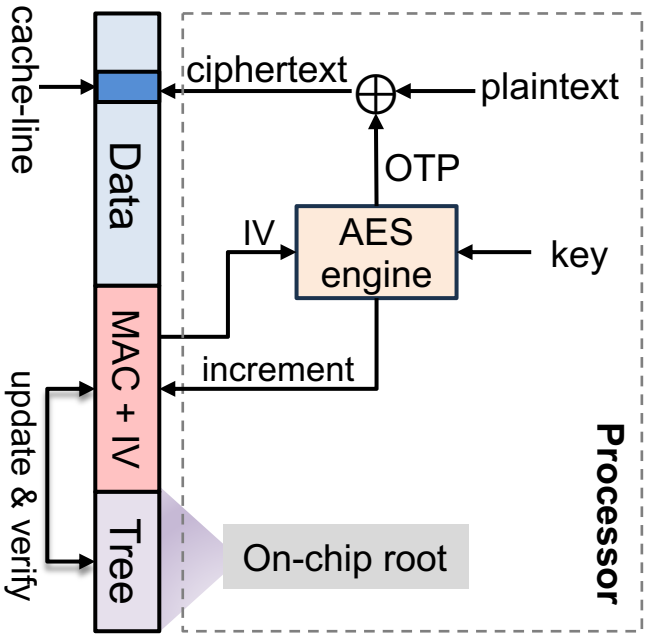
Closer Look

Isolation



(a) CC with UVM

Encryption



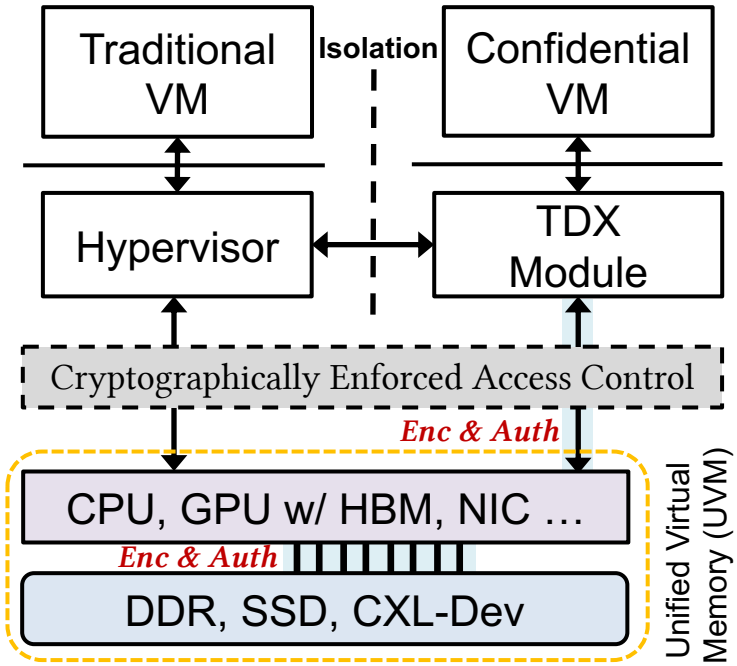
(b) Traditional memory encryption

IV: Initialization Vector

$$IV{+}{+};\ OTP = AES_K(IV||CTR);\ Ct = Pt \oplus OTP$$

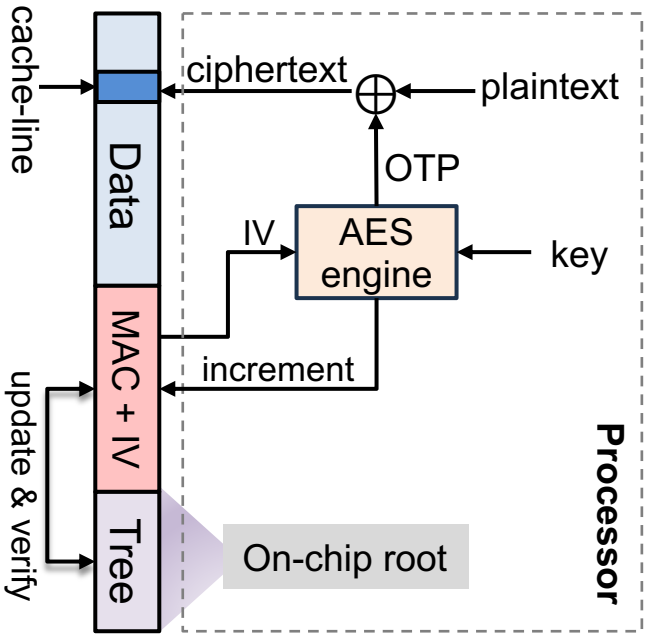
Closer Look

Isolation



(a) CC with UVM

Encryption

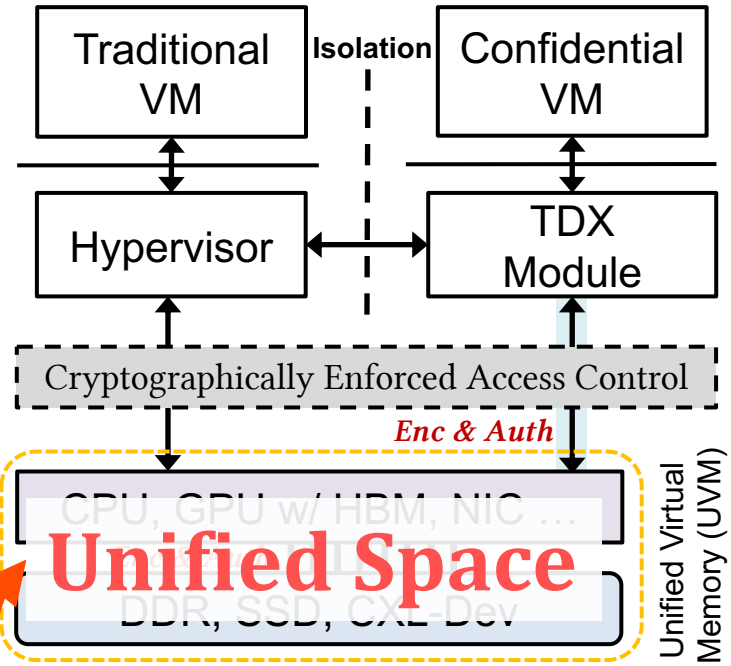


(b) Traditional memory encryption

The memory problem!

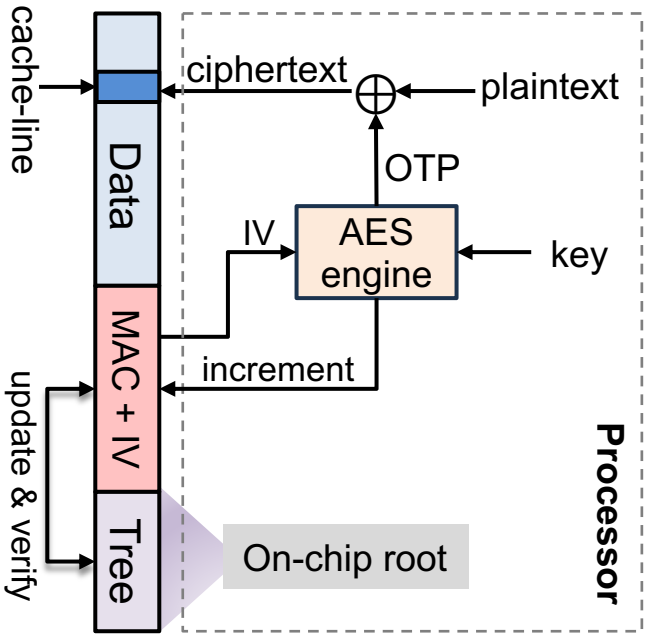
Closer Look

Isolation



(a) CC with UVM

Encryption

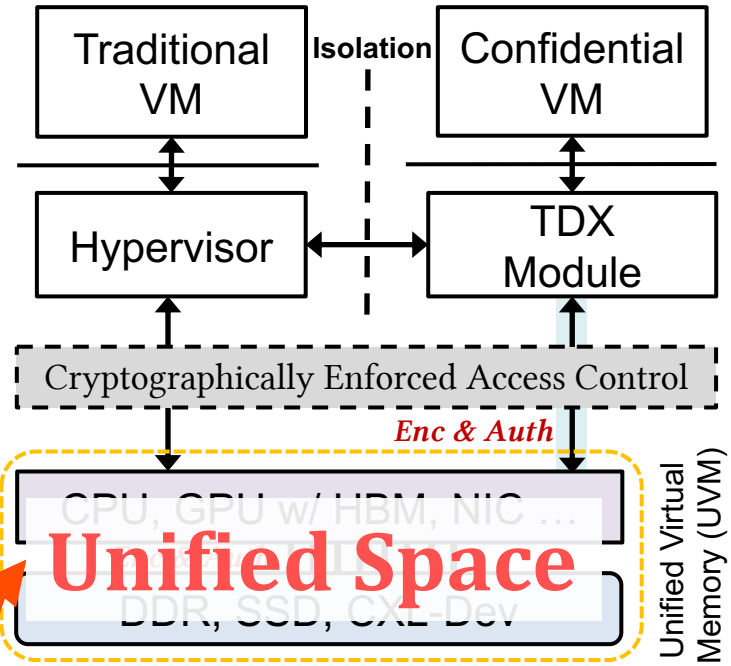


(b) Traditional memory encryption

The memory problem!

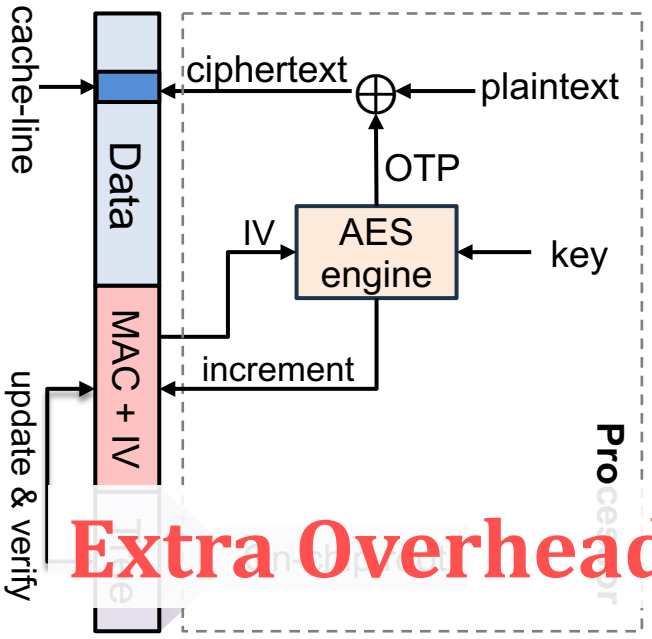
Closer Look

Isolation



(a) CC with UVM

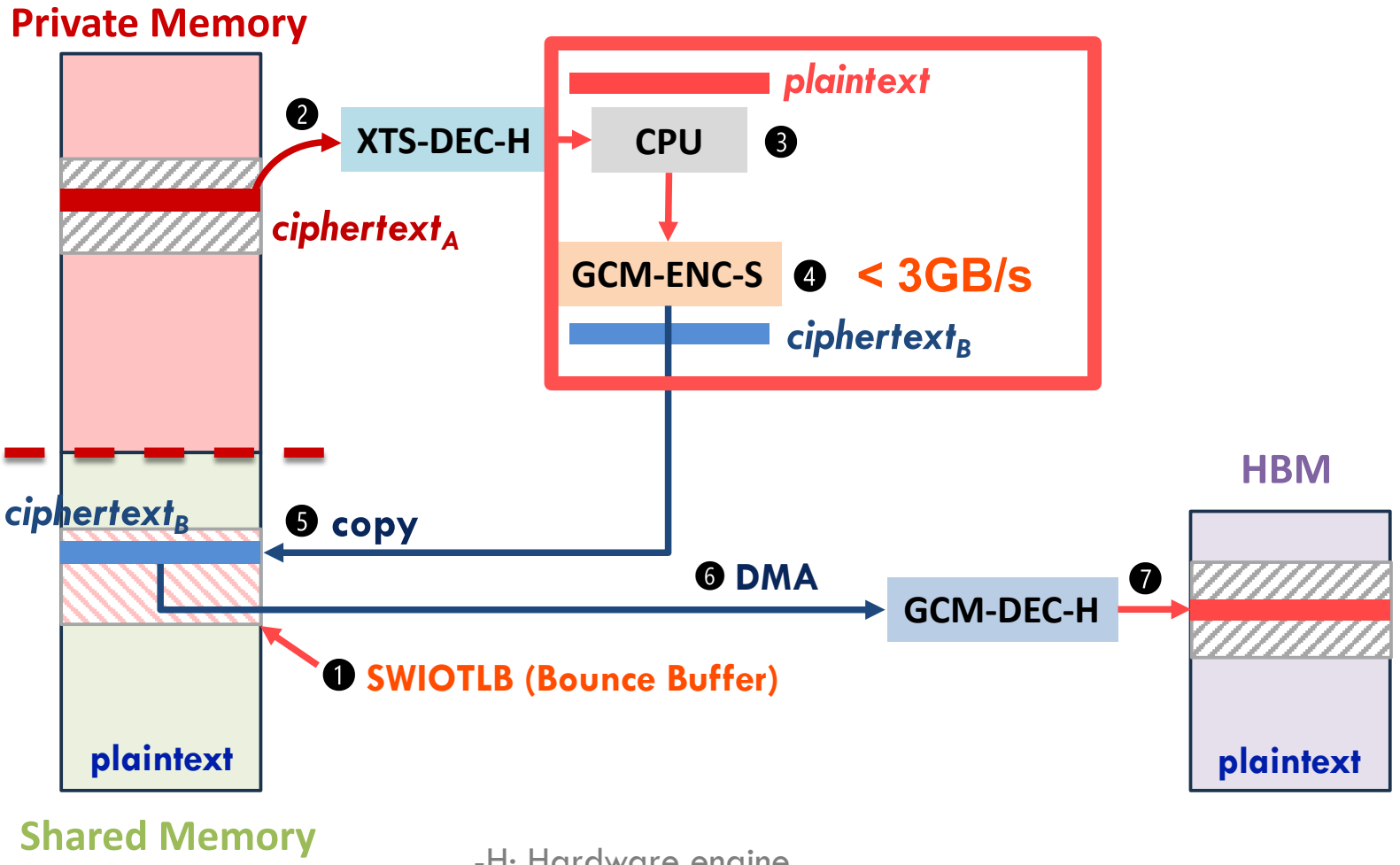
Encryption



(b) Traditional memory encryption

The memory problem!

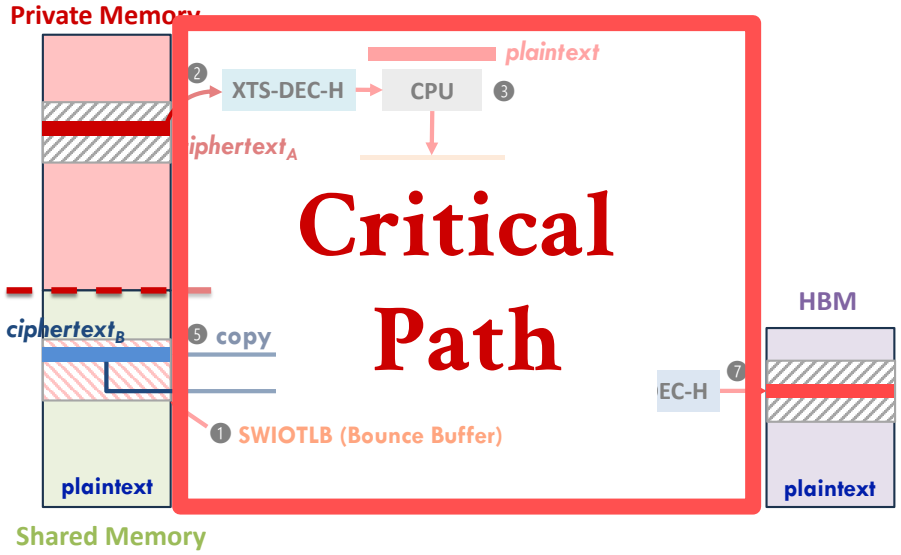
Problem: Bounce Buffer CC



-H: Hardware engine

-S: Software implementation

Problem



UVM brings frequent CPU-GPU interaction

↓

[ISPASS'25] Low performance under CC

↓

Lack of explicit IV (Encryption on the critical path)

The missing IV management makes encryption synchronized with runtime access.

This Work: LÆGIS

Bring flexible IV management back to GPU-based CC

Consideration	Prior Counter Mode	GPU-based CC (UVM)
IV / MAC	✓ Explicit per-CL	× Implicit, access-based
Integrity Tree	× Required	✓ Deprecated
OTP Timing	✓ Decoupled	× On-access only
Granularity	× Cache-line (64B/128B)	✓ Page-level (64KB, 2MB)
Memory	× Encrypted DRAM	✓ Plaintext HBM

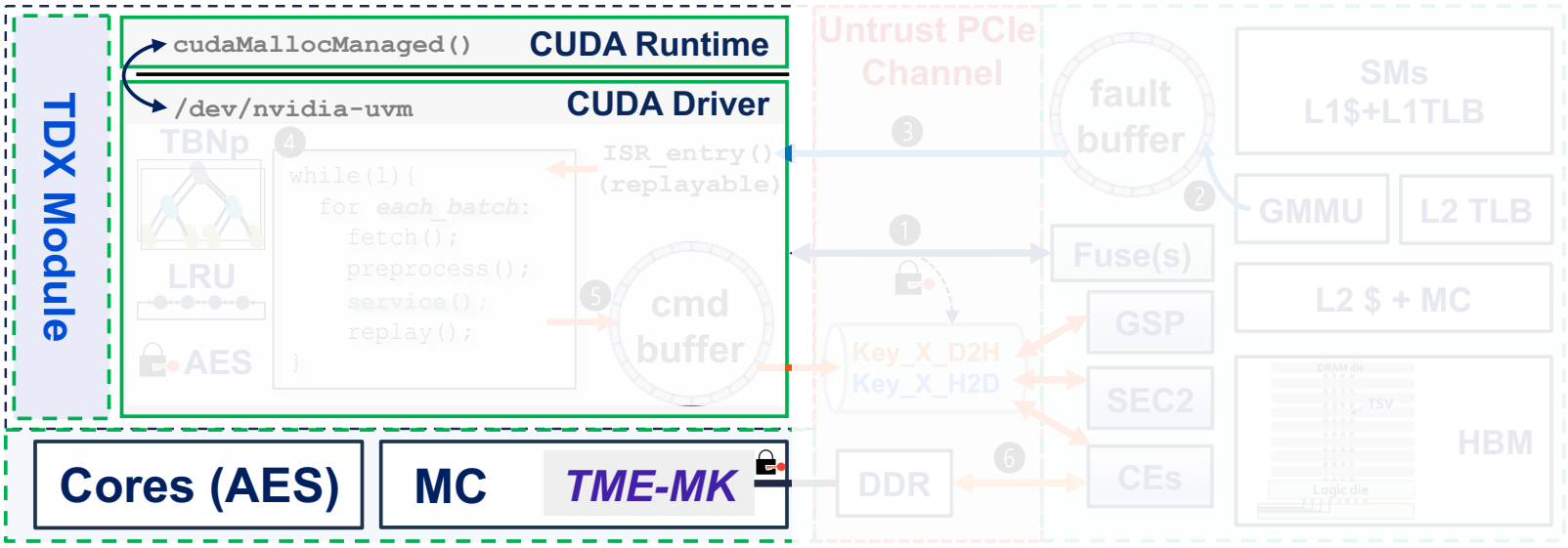
Encryption context management is more critical.

- ❖ Performance dissection of UVM under CC
- ❖ Observations + Opportunities
- ❖ LÆGIS: opportunistically pre-encryption + HBM for IV management

Outline

- Introduction
- **Background**
- UVM Performance Dissection under CC
- LÆGIS: Overview & Implementation
- Evaluation
- Takeaway

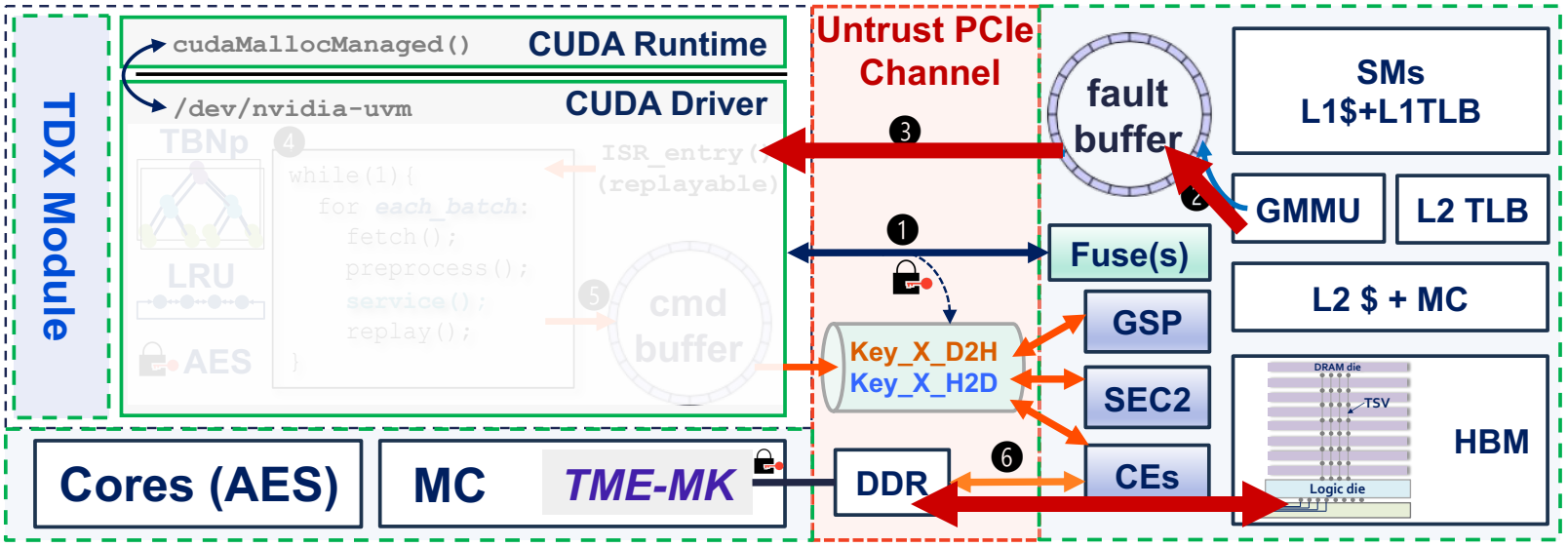
UVM in TDX-based System



(a) TDX-capable CPU

(b) CC Mode GPU

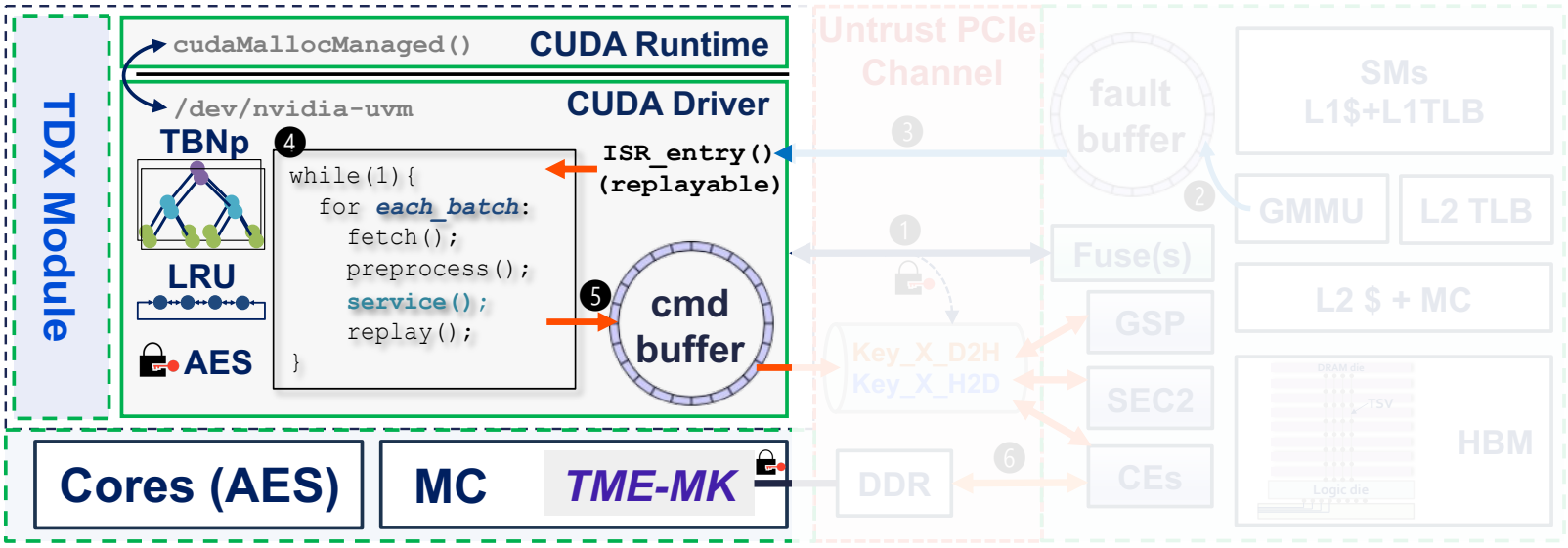
UVM in TDX-based System

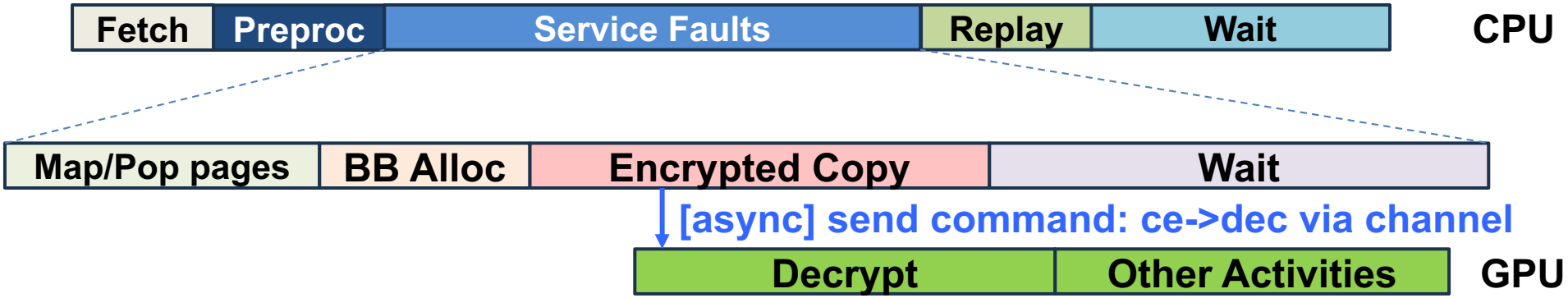


(a) TDX-capable CPU

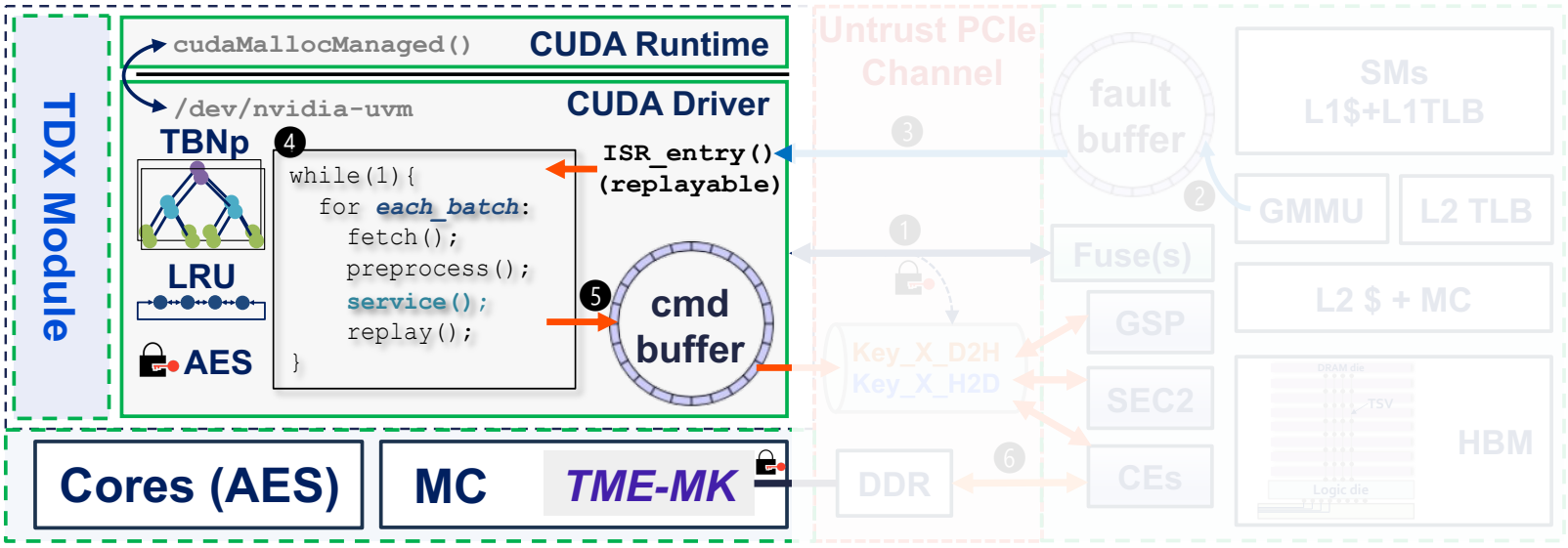
(b) CC Mode GPU

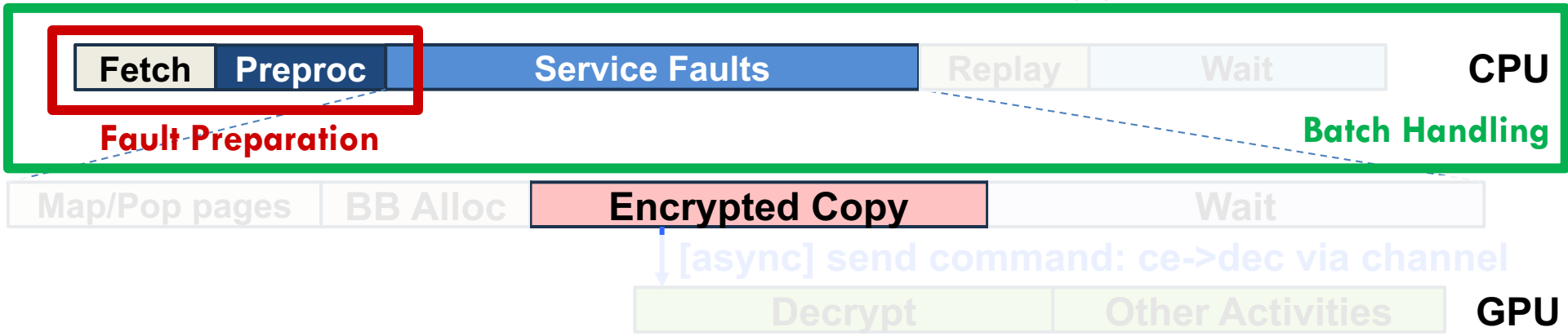
UVM in TDX-based System



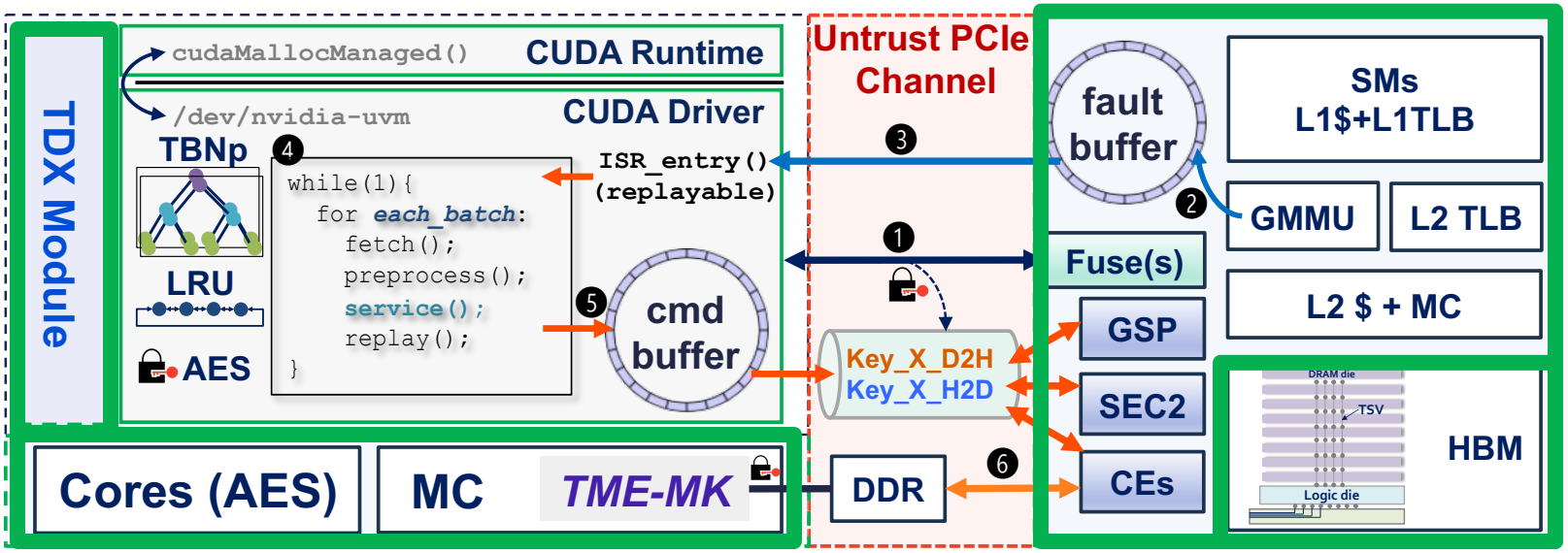


UVM in TDX-based System





Threat Model



(a) TDX-capable CPU

(b) CC Mode GPU

Signed software

Chip packages

Internal structures (HBM)

Check our paper: Section IX for more discussions on threat model

Outline

- Introduction
- Background
- **UVM Performance Dissection under CC**
- LAEGIS: Overview & Implementation
- Evaluation
- Takeaway

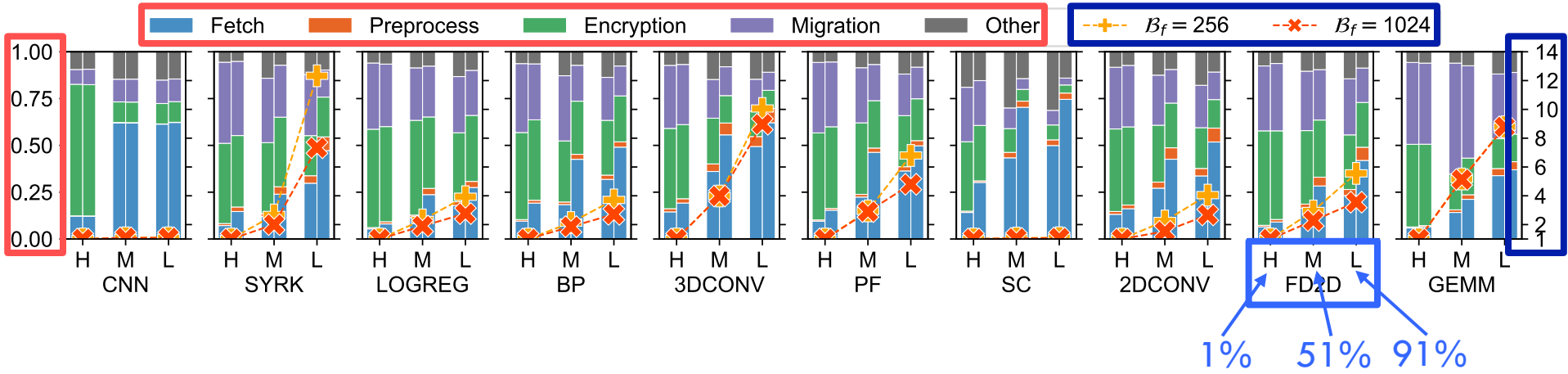
Setup

- 5th Gen Intel Xeon 6530 Gold
- NVIDIA H100 NVL
- CUDA 12.4
- OpenSSL v3.0.2
- Linux 6.2.0-mvp10v1+8-generic

UVM performance evaluated using an instrumented NVIDIA driver (version 550.163.01)

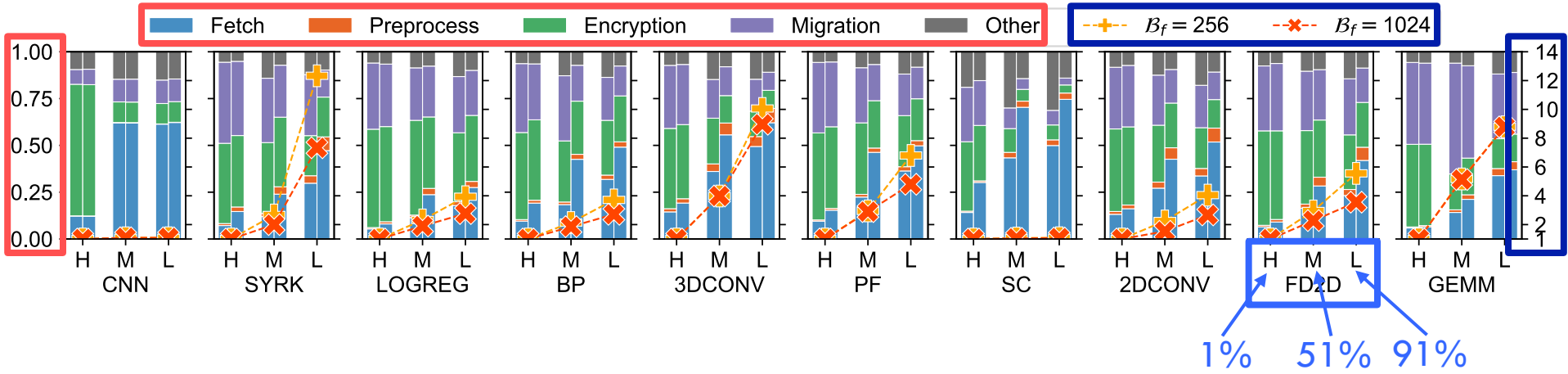
Batch Handling Performance

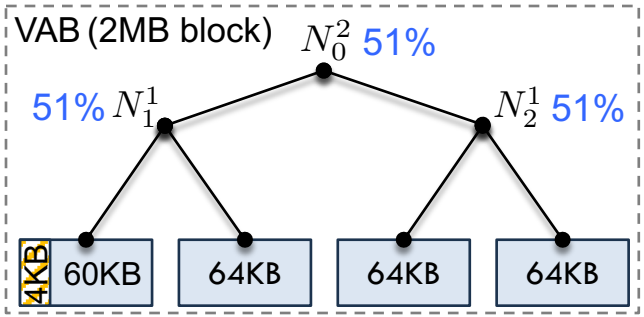
❖ **Evaluated using an instrumented NVIDIA driver (version 550.163.01)**



Batch Handling Performance

❖ Evaluated using an instrumented NVIDIA driver (version 550.163.01)



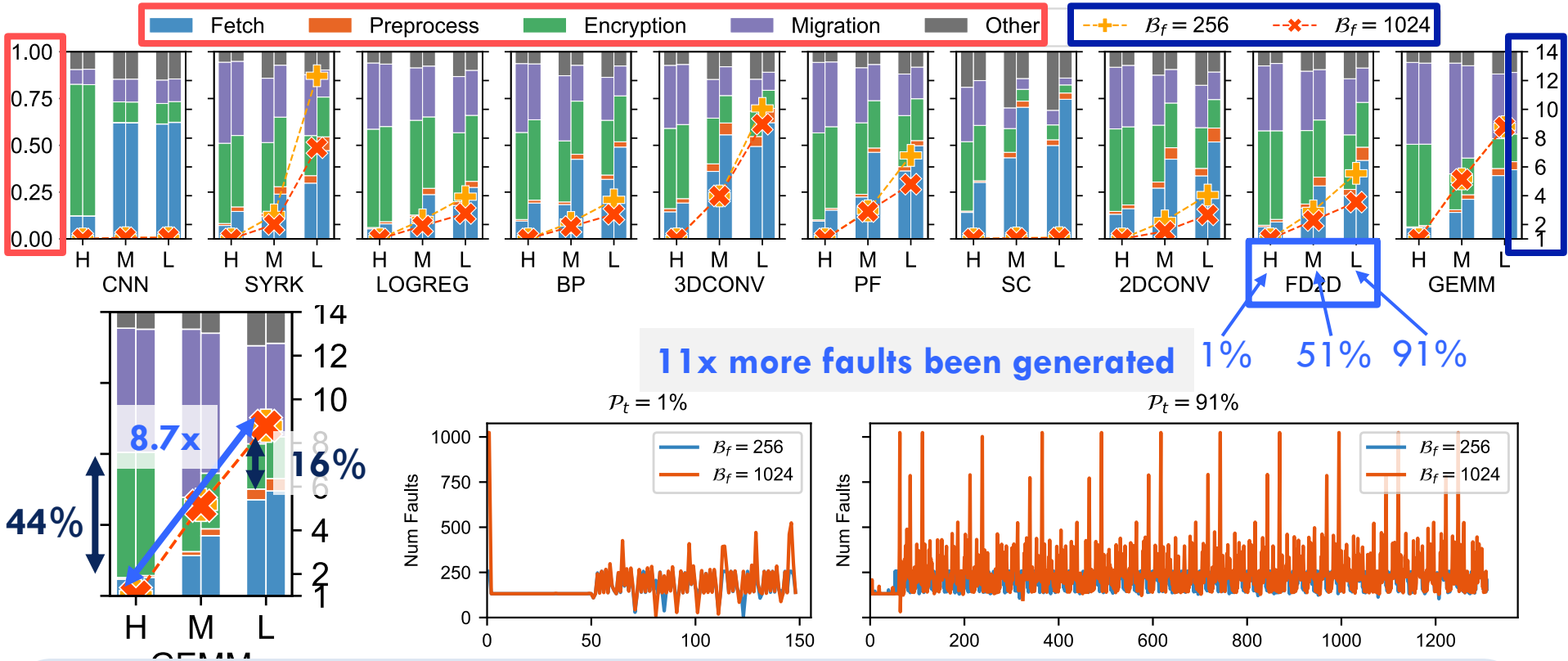


Tree-based Neighborhood Prefetcher (TBNp)

When more than 51% pages been faulted, the sub-tree will be prefetched.

Batch Handling Performance

❖ **Evaluated using an instrumented NVIDIA driver (version 550.163.01)**



Observation 1: **Aggressive prefetching (P_t = 1%) reduces costly GPU-CPU interactions. However, it shifts the burden to encryption, which dominates the overall handling time and can exceed 70%. Thus, encryption is the critical bottleneck.**

Encryption Mechanism

```
1 va_bb = dma_alloc_coherent();
2 for_each_va_block_page_in_region(page_index, region) {
3     void *va_src = kmap(src_page);
4     uvm_cc_cpu_encrypt(push->channel, va_bb, va_src, iv,
      ↪  PAGE_SIZE, va_tag);
5     kunmap(src_page);
6     gpu->parent->ce->decrypt(push, gpu_dst, va_bb, PAGE_SIZE,
      ↪  va_tag);
7 }
```

Encryption Mechanism

```
@[aead_enc,
    crypto_aead_encrypt+1
    crypto_aead_encrypt+57
    lkca_aead_internal+243
    libspdm_aead_prealloced+961
    libspdm_aead_aes_gcm_encrypt_prealloc+157
    ccslEncrypt_KERNEL+201
    nvGpuOpsCcslEncrypt+32
    nvUvmInterfaceCslEncrypt+64
    uvm_conf_computing_cpu_encrypt+112
    uvm_channel_end_push+3012
    uvm_push_end+14
    ......
    service_fault_batch+182
    uvm_gpu_service_replayable_faults+356
    ......
    replayable_faults_isr_bottom_half_entry+39
    _main_loop+137
    kthread+235
    ret_from_fork+41
]: 1
```

Linux Kernel Crypto APIs

libspdm/lkca

NVIDIA
Cryptography Services Library

UVM Fault Handling
/dev/nvidia-uvm

Kernel Threading

Encryption Mechanism

integrity/freshness/...?

(decoupled encryption)

iv_space [page_src]

uvm_cc_cpu_encrypt (page_src, iv++)

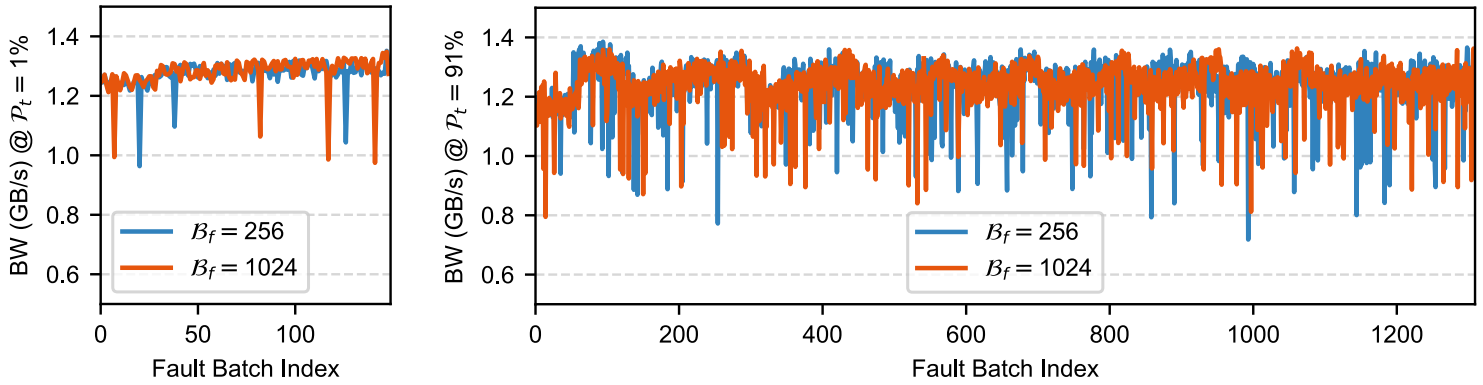
```
1 va_bb = dma_alloc_coherent();
2 for_each_va_block_page_in_region(page_index, region) {
3     void *va_src = kmap(src_page);
4     uvm_cc_cpu_encrypt(push->channel, va_bb, va_src, iv,
      ↪  PAGE_SIZE, va_tag);
5     kunmap(src_page);
6     gpu->parent->ce->decrypt(push, gpu_dst, va_bb, PAGE_SIZE,
      ↪  va_tag);
7 }
```

Observation 2: Under GPU-based CC, UVM paging encryption is a kernel-space, Linux Kernel Crypto API (LCA)-backed path that operates synchronously at 4KB page granularity with IV synchronization. This differs from user-space OpenSSL-based bulk encryption.

Encryption Performance

```
1 va_bb = dma_alloc_coherent();
2 for_each_va_block_page_in_region(page_index, region) {
3     void *va_src = kmap(src_page);
4     uvm_cc_cpu_encrypt(push->channel, va_bb, va_src, iv,
      ↪  PAGE_SIZE, va_tag);
5     kunmap(src_page);
6     gpu->parent->ce->decrypt(push, gpu_dst, va_bb, PAGE_SIZE,
      ↪  va_tag);
7 }
```

Encrypting a 4KB page takes 3 µs

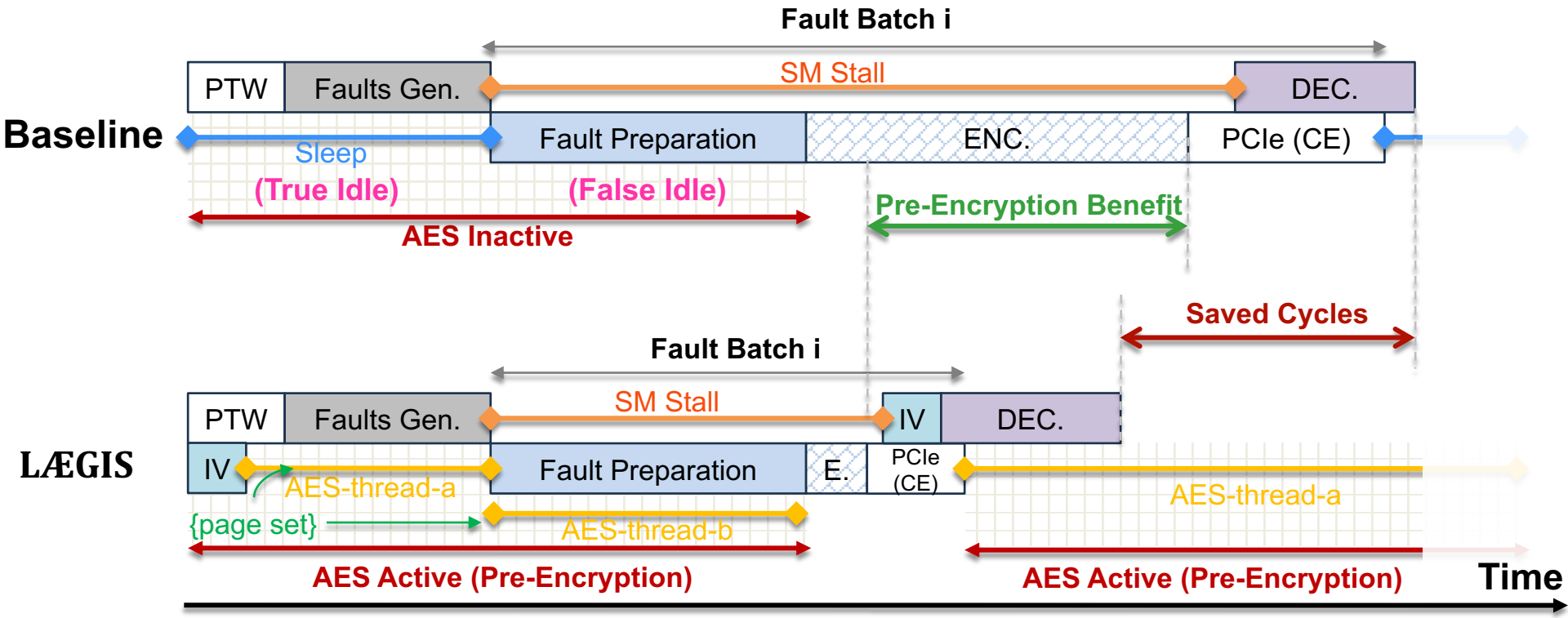


Observation 3: GPU CC presents slow (1.3 GB/s) and *synchronous* software encryption in UVM, placing it on the critical path. Such low throughput presents significant performance overhead.

Outline

- Introduction
- Background
- UVM Performance Dissection under CC
- **LÆGIS: Overview** & Implementation
- Evaluation
- Takeaway

Overview

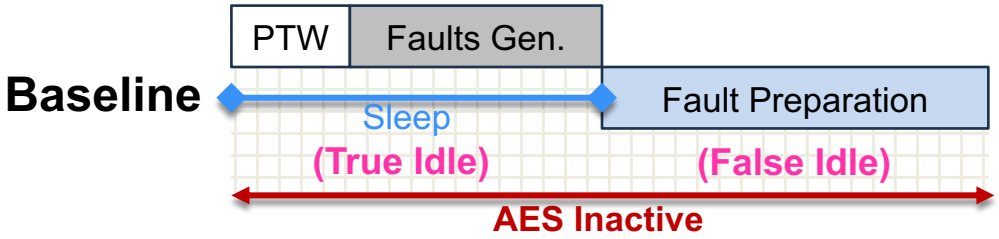


* Assume IV can be explicitly managed by LÆGIS

PRE-ENCRYPTION

When, How and What

Pre-Encryption Definition

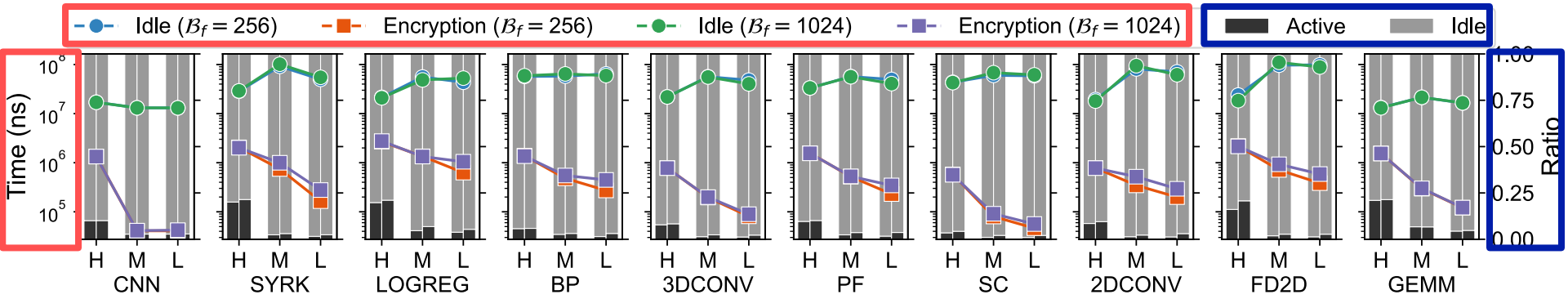


Invariant

In LÆGIS, all pre-encrypted pages will be *committed* in the end.

LÆGIS tolerates wrong prediction

Idle Service Routine: When to Encrypt



On average, the driver remains idle for 87% of its execution time, during which encryption threads are *underutilized*.

Opportunity 1: Between fault batches, the service thread enters *true idle*, and during fault preparation the thread experiences *false idle*. Both leave CPU encryption resources *underutilized*. These intervals can be used to hide (some) encryption latency if *pre-encryption* is applied.

LÆGIS Goal

RQ1 : How to design an encryption scheme tailored for **GPU-based CC** that can support **out-of-order encryption**;

RQ2 : With this scheme, how to achieve **efficient batch handling**.

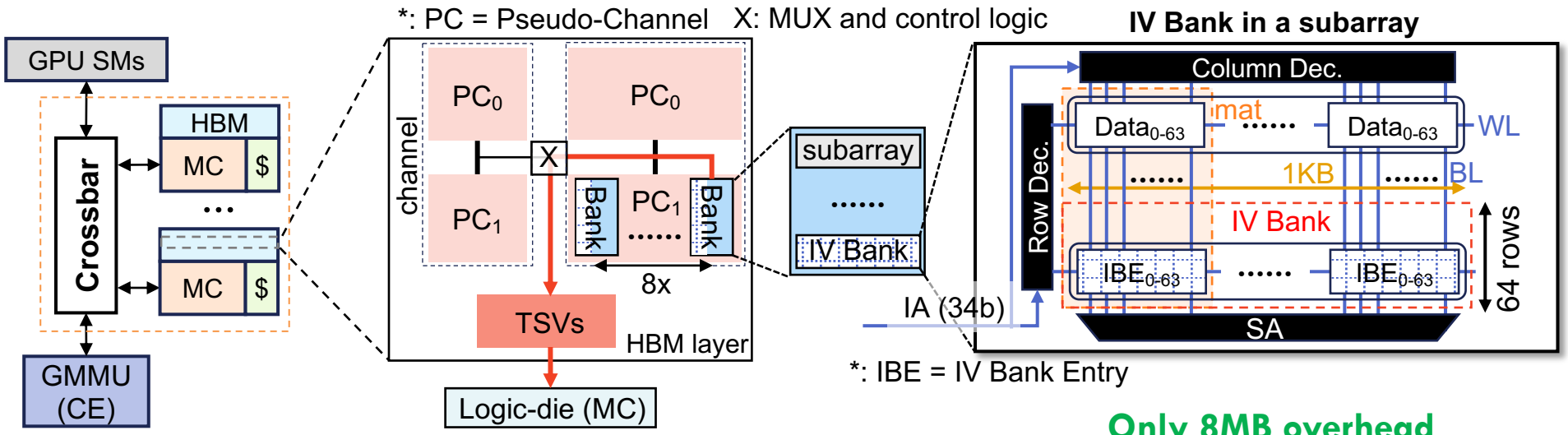
- ✓ **Leverages HBM for metadata (e.g., IV) storage.**
- ✓ **Decouples encryption from access ordering by introducing explicit metadata management.**
- ✓ **Proposes an out-of-order pre-encryption scheme exploiting idle intervals.**

Outline

- Introduction
- Background
- UVM Performance Dissection under CC
- **LÆGIS: Overview & Implementation**
- Evaluation
- Takeaway

IV Bank Concept

Reserved logical region, termed the IV Bank, in each HBM stack



Only 8MB overhead

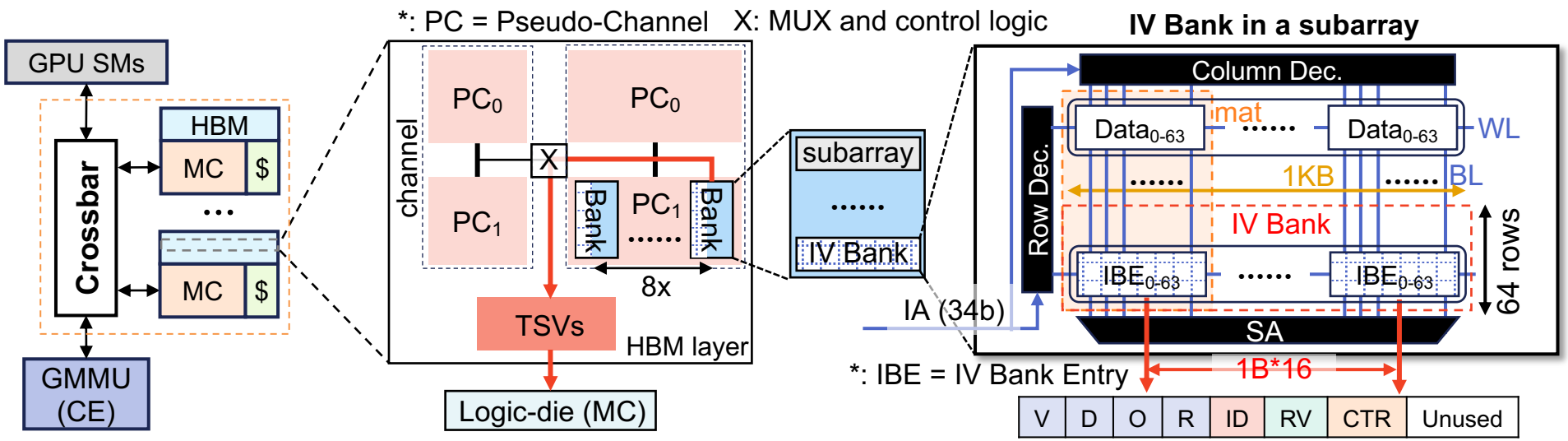
More details please check the paper.

The IV Bank stores per-VABlock IV Bank Entries (IBEs)

IBEs are interleaved to all physical banks

IV Bank Concept

Reserved logical region, termed the IV Bank, in each HBM stack

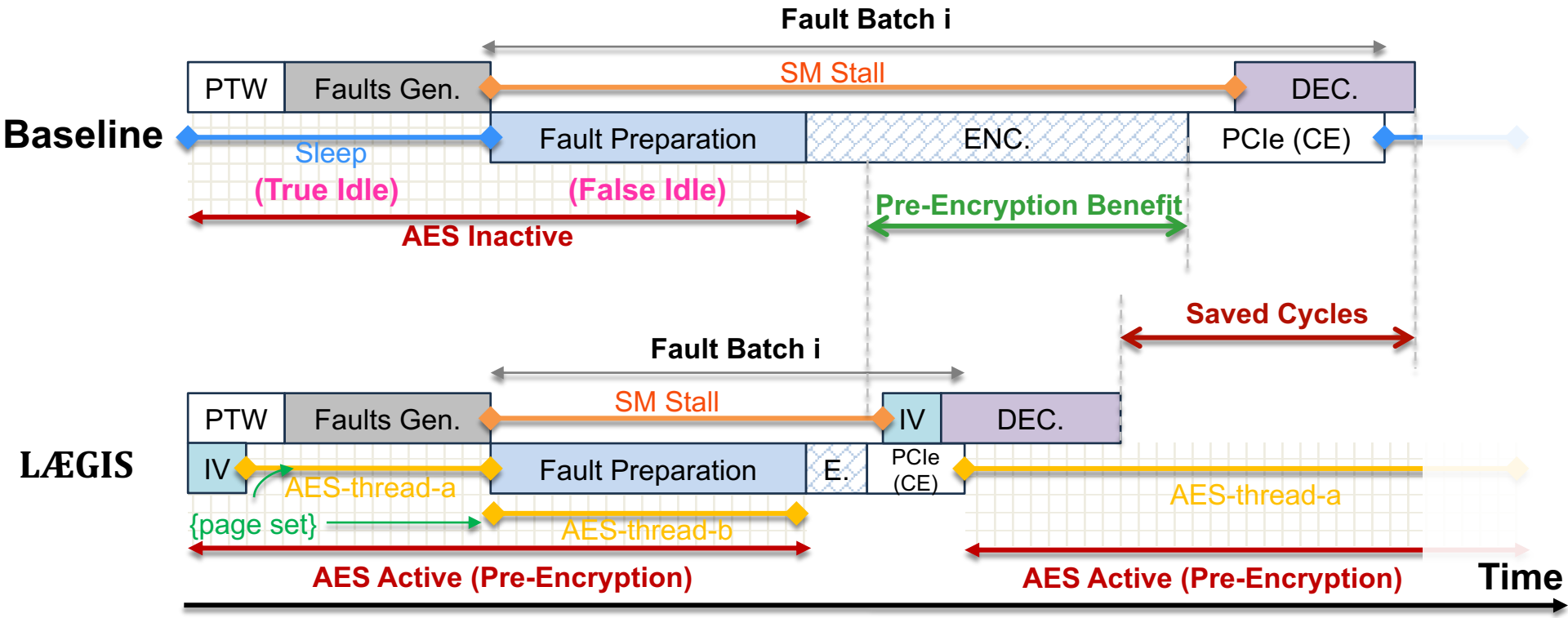


More details please read the paper.

To avoid IV collisions, the 96-bit IV into two parts:

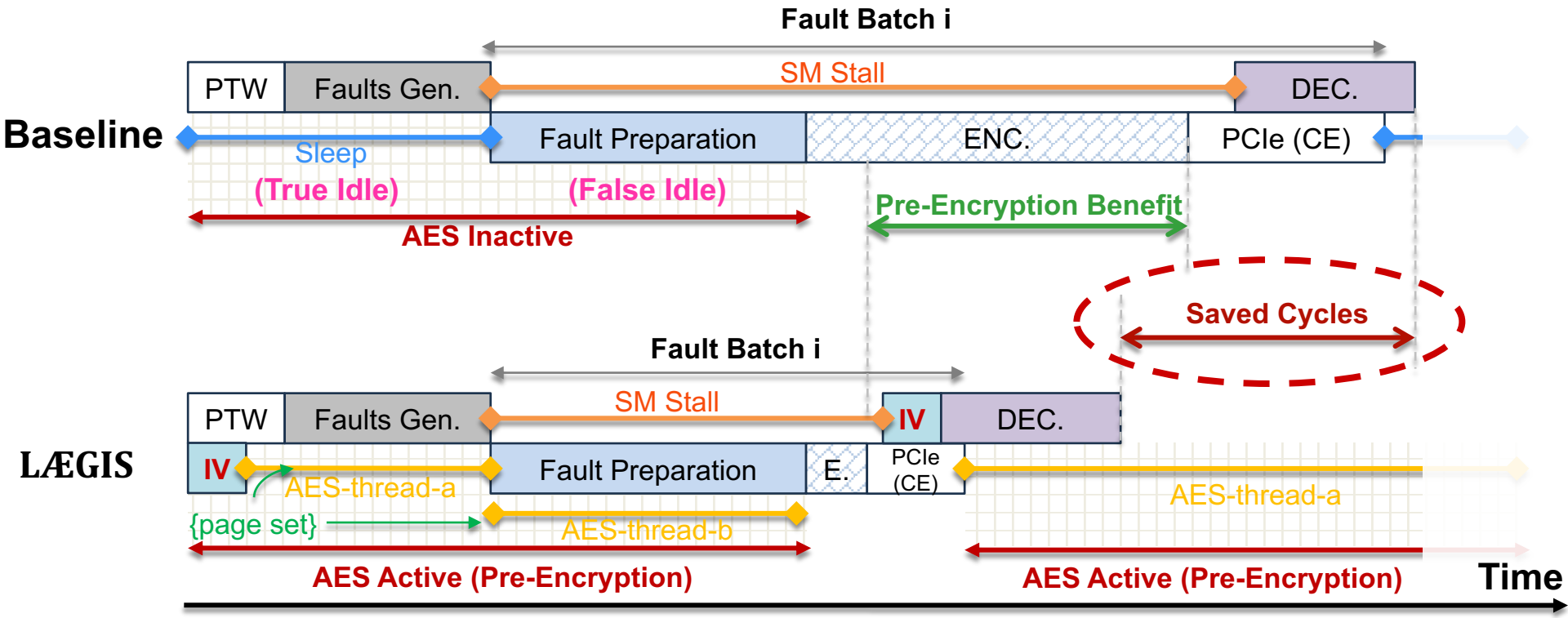
- ✓ **19-bit identifier (ID)**
- ✓ **77-bit random value (RV)**

HBM Assisted Encryption



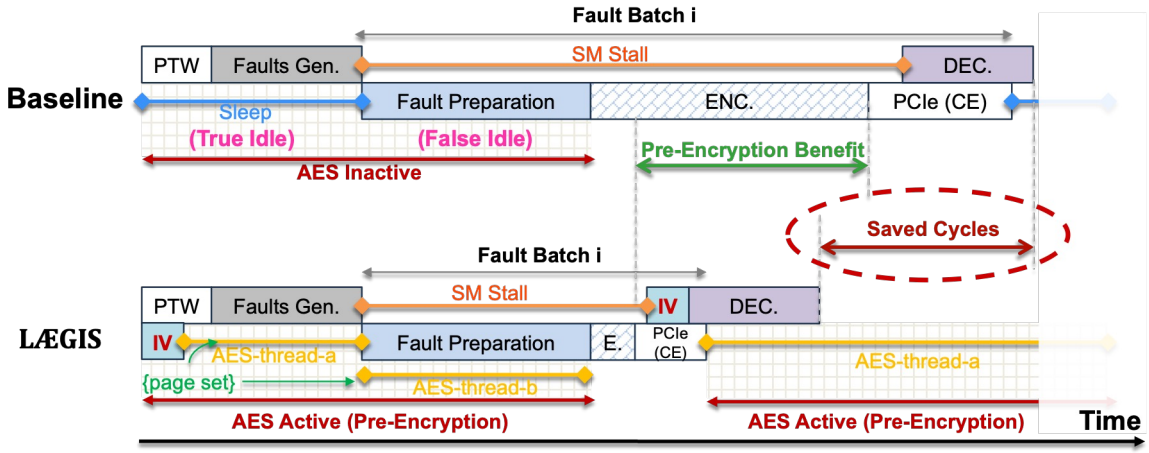
* Assume IV can be explicitly managed by LÆGIS

HBM Assisted Encryption



* Now IV can be explicitly managed by LÆGIS !

HBM Assisted Encryption



*** Now IV can be explicitly managed by LÆGIS !**

Full walkthrough example please read the paper.

Opportunistically pre-encrypt pages with the help of IV Bank during batch handling.

Even random order encryption will work:
IV Bank decouples encryption from access.

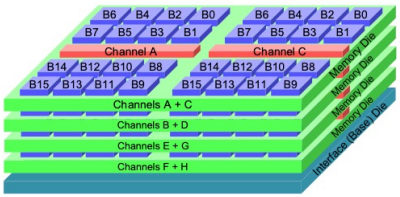
Outline

- Introduction
- Background
- UVM Performance Dissection under CC
- LÆGIS: Overview & Implementation
- **Evaluation**
- Takeaway

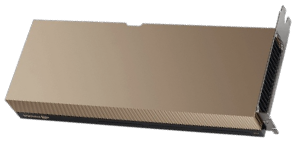
Simulation Setup

GPGPU-Sim v4.2 + UVMSmart

80 SMs, 64 warps/SM; shared GMMU; private TLB
8GB HBM2 (1 stack);
Page size: 4 KB base, 2 MB big;
PCIe: 64 GB/s per direction;
Pt: 51% (default), 1% (aggressive);
Thread switch: 2 µs
Fault preparation and idle time: profiled from hardware setup



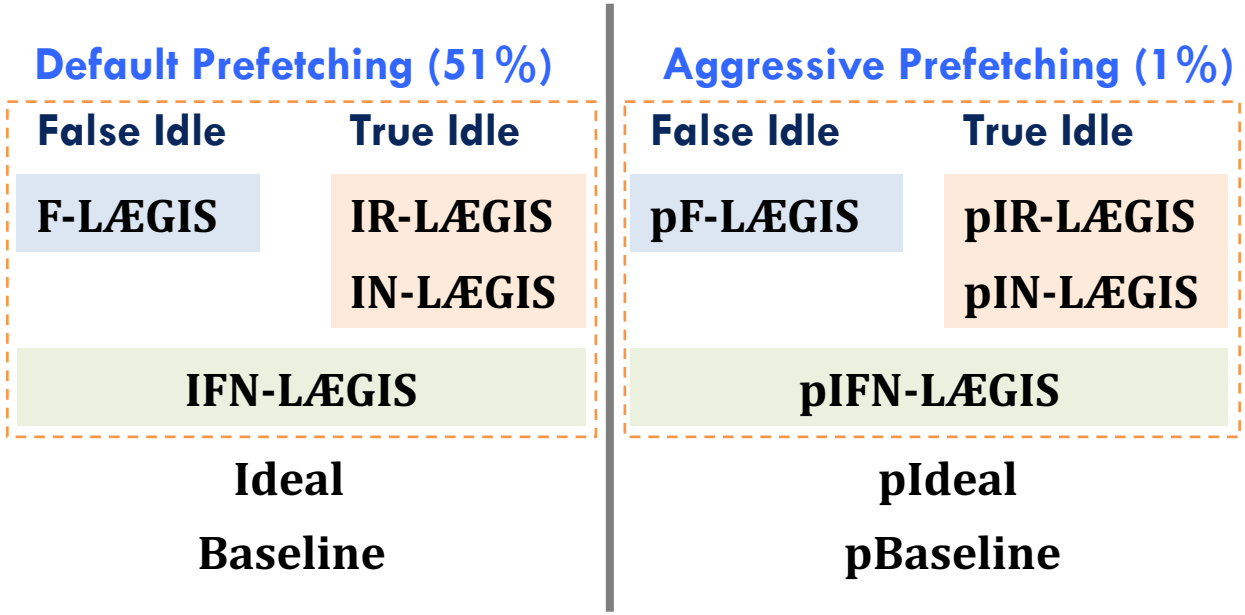
8GB 4-Hi stack, 8 channels, 16 pseudo channels
FR-FCFS; 16 banks, 4 bank groups/channel; 128-bit interface
256 GB/s via 8 channels @1GHz;
Static mapping



5th Gen Intel Xeon 6530 Gold
NVIDIA H100 NVL
CUDA 12.4
OpenSSL v3.0.2
Linux 6.2.0-mvp10v1+8-generic

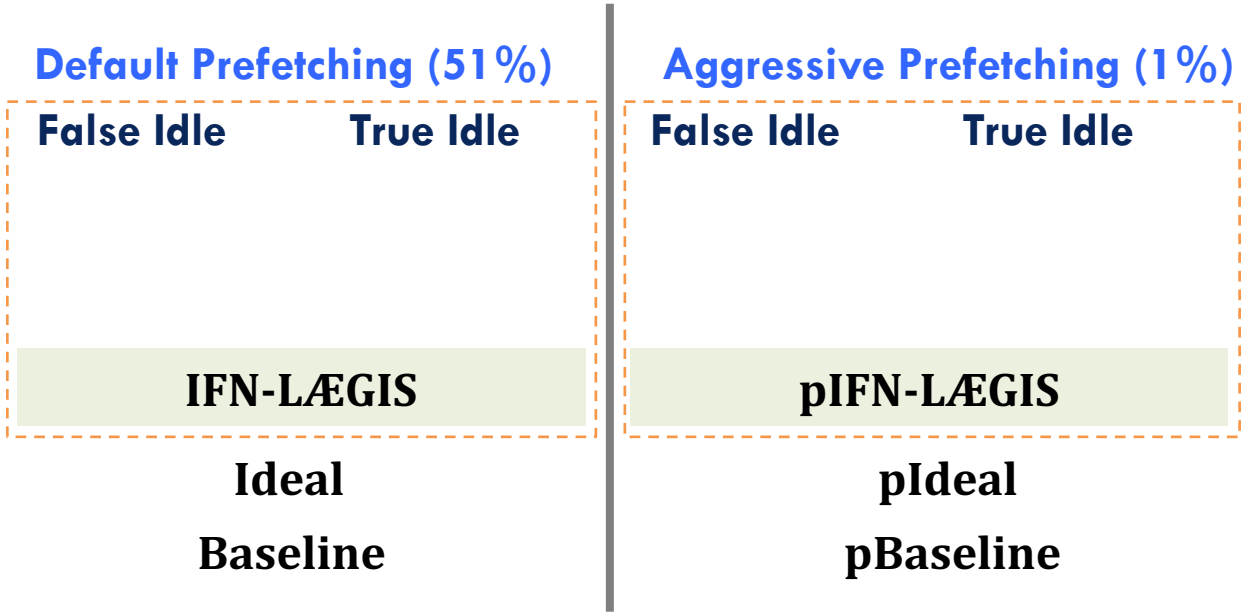
Method

16 applications from widely adopted benchmark suites



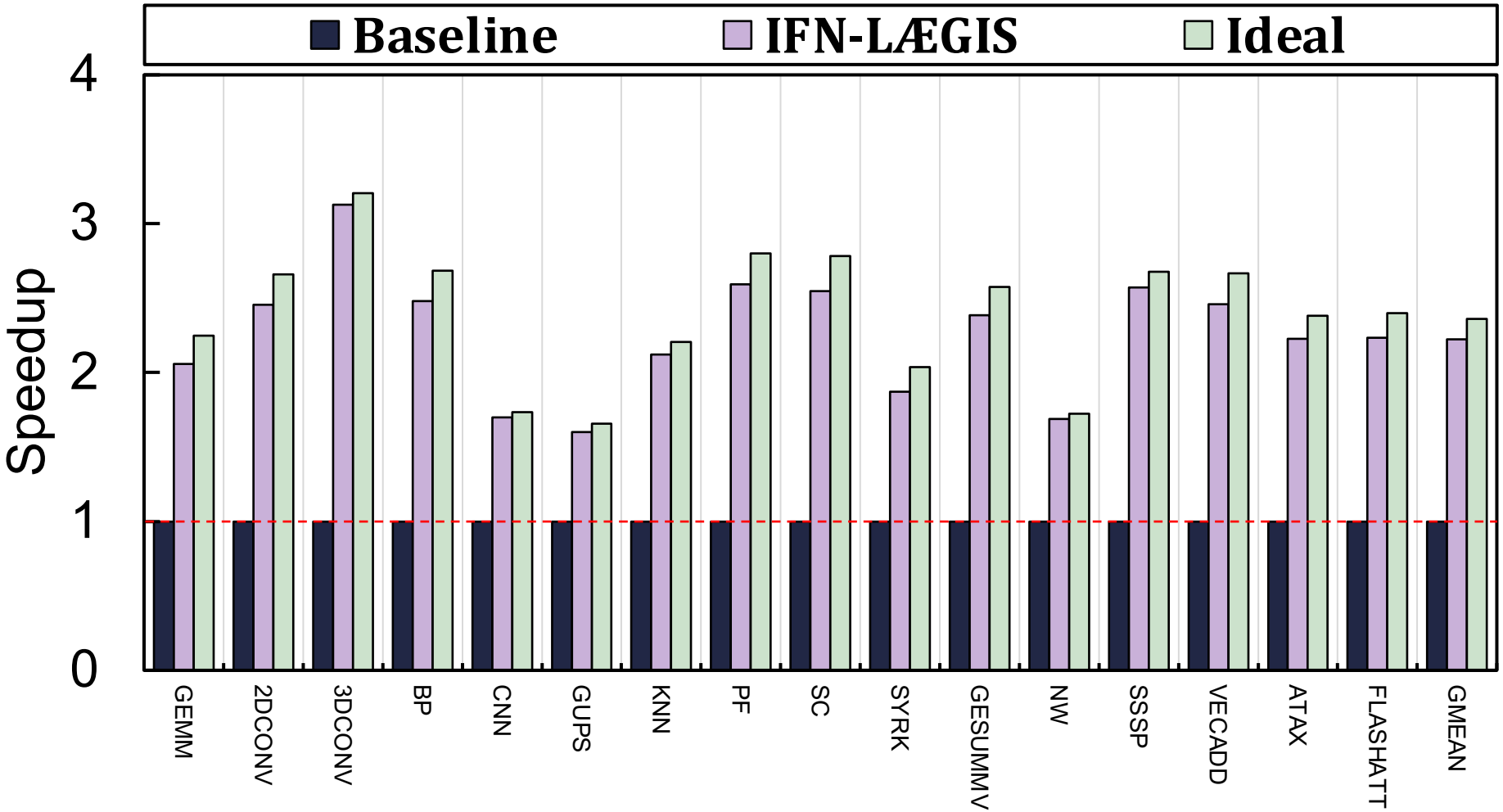
Method

16 applications from widely adopted benchmark suites

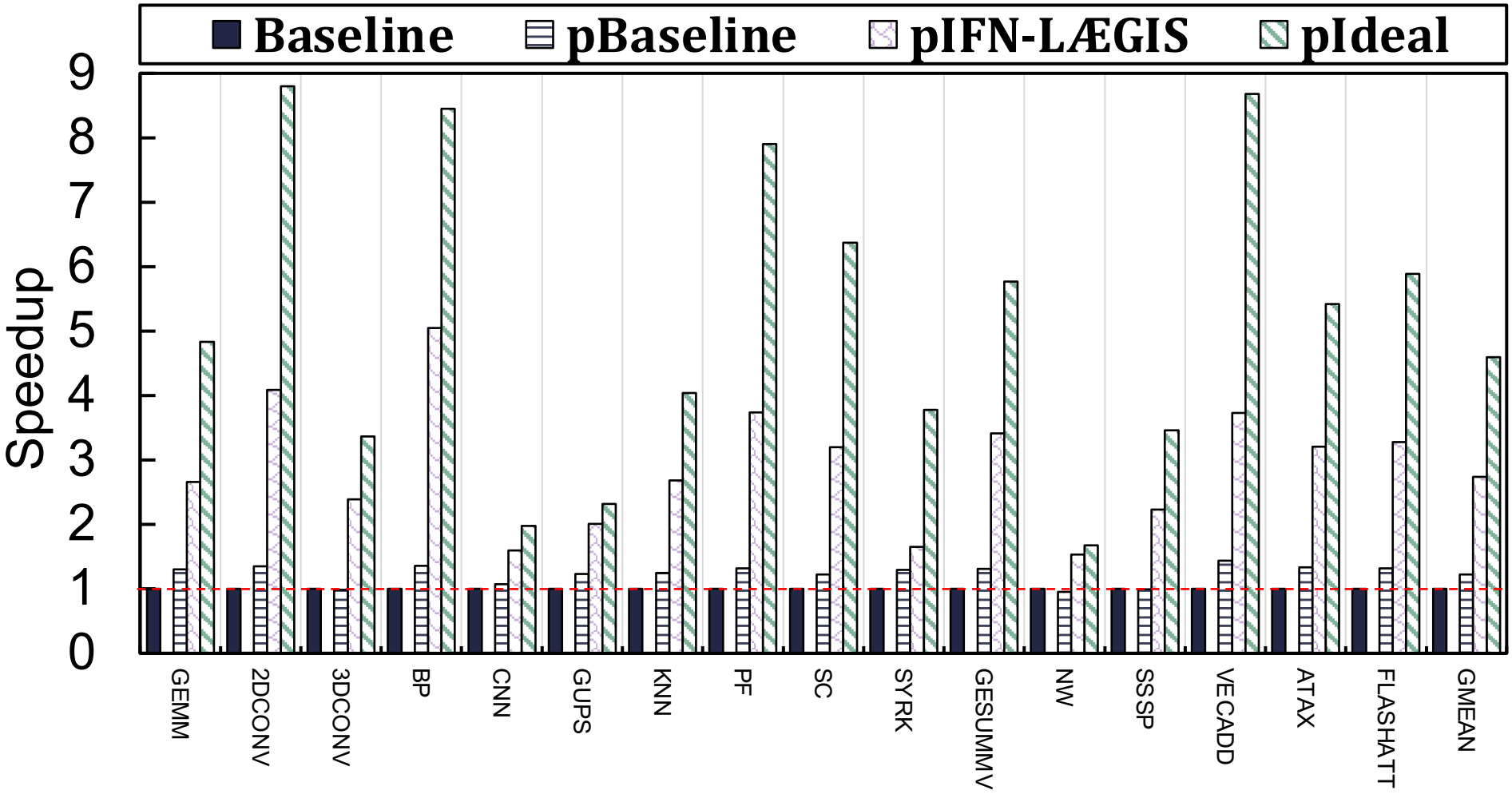


More results please check the paper.

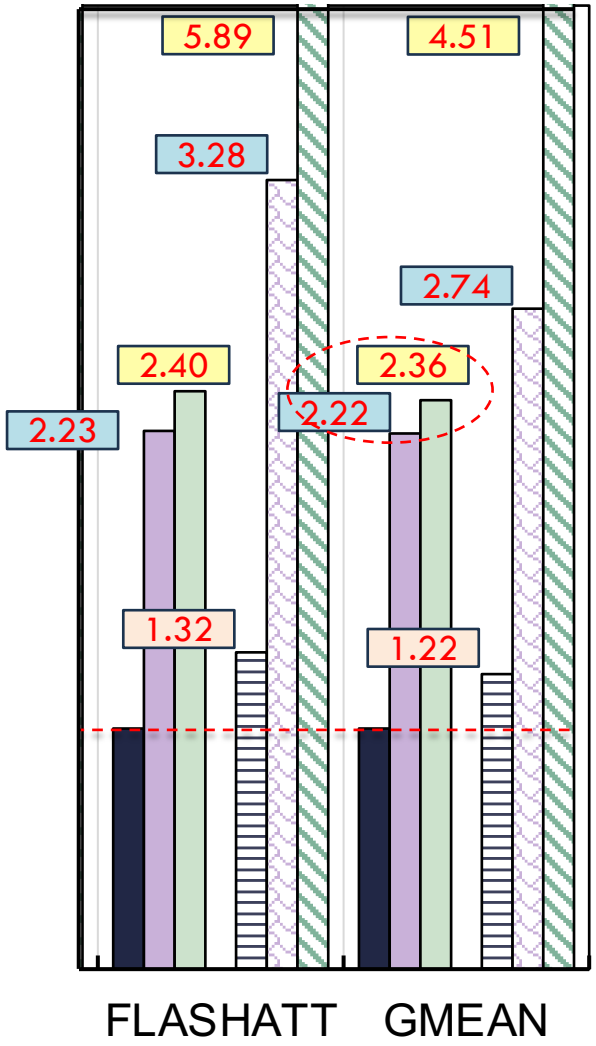
Main Results



Main Results



Main Results



On average, **IFN-LÆGIS** achieves a **2.22×** speedup, with a maximum of **3.13×**, reducing the gap to Ideal to only **5.8%**.

pIFN-LÆGIS further improves performance under aggressive prefetching. It achieves a maximum speedup of **5.05×** and an average of **2.74×**, narrowing the gap with **pIdeal** to 40.34%.

More Details

- Threat model and HBM security discussion
- Security analysis
- Pre-Encryption: how and what
- IV Bank details
- Pre-Encryption walkthrough example
- More LÆGIS variants results
- Future directions
- Scalability
- Related works
- More baselines for CPU encryption:
  - Multi-threading
  - Advanced hardware
  - OpenSSL versions

.....

Outline

- Introduction
- Background
- UVM Performance Dissection under CC
- LÆGIS: Overview & Implementation
- Evaluation
- **Takeaway**

Take Away Messages

- We identify **three inefficiencies** in GPU-based CC when it comes to UVM.
  - CPU–GPU IV synchronization
  - Driver idleness
  - Slow software encryption

- We propose **LÆGIS**, a design that **opportunistically** pre-encrypts pages in CC and uses secure **HBM** for IV management.

- **LÆGIS** introduces an **in-memory IV Bank** in 3D-stacked HBM, decoupling encryption from CPU–GPU synchronization and removing the need for integrity trees.

Average speedup of 2.22× (up to 3.13×) and 2.74× (up to 5.05×) under default and aggressive prefetching.

[PDF] https://adwaitjog.github.io/docs/pdf/yang_laegis_isca26.pdf
[GitHub] https://github.com/insight-cal-uva/laegis-isca26-artifact

Thank You!

Questions?

LÆGIS: Pinpointing and Addressing Performance Overheads of GPU-based Confidential Computing

Yang Yang, Adwait Jog
{*yangyang, ajog*} @ *virginia.edu*

UNIVERSITY of VIRGINIA

ISCA 2026