Common Counters: Compressed Encryption Counters for Secure GPU Memory

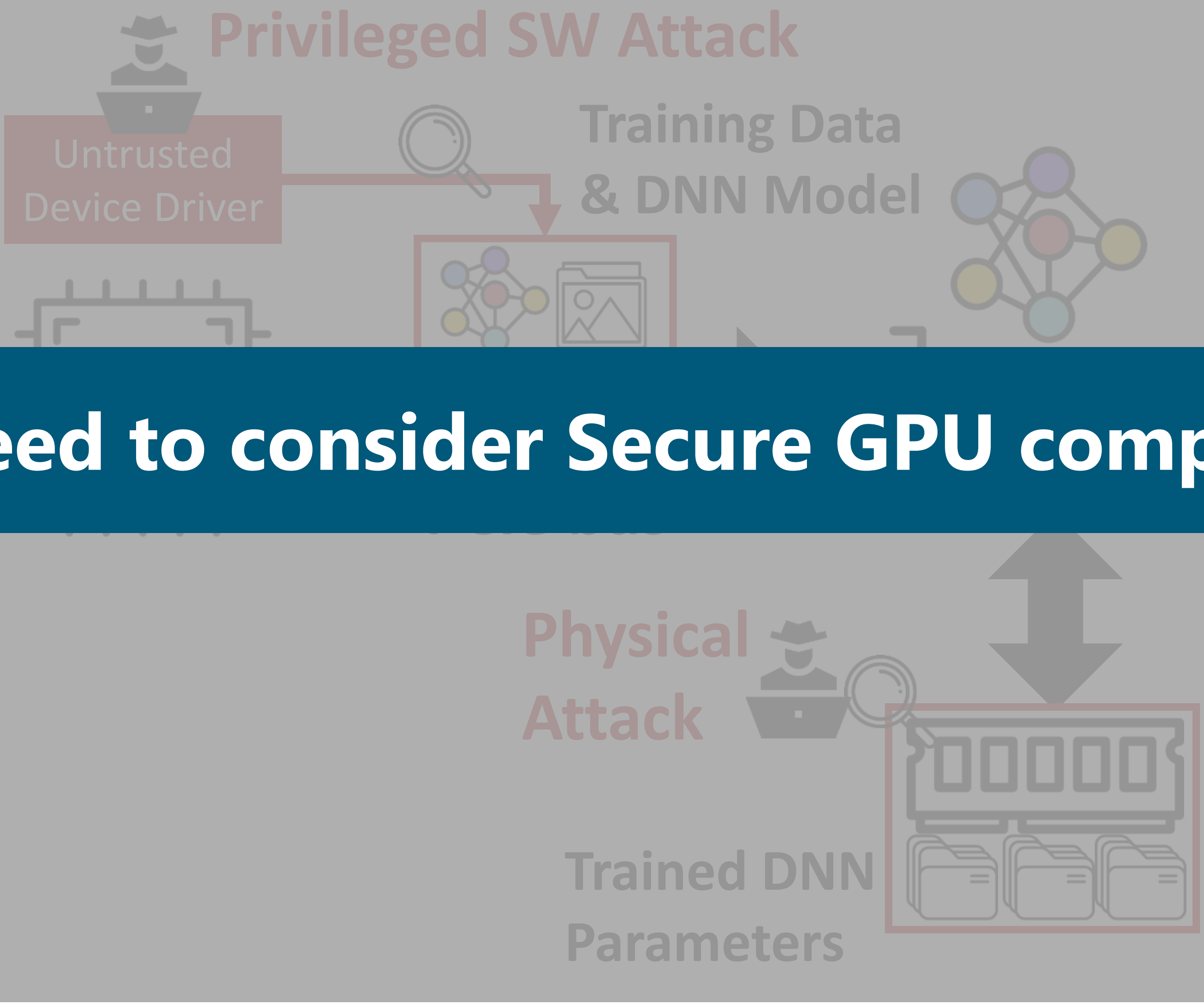
Seonjin Na, Sunho Lee, Yeonjae Kim, Jongse Park, Jaehyuk Huh

KAIST, School of Computing

CASYS
Computer Architecture and Systems Lab

KAIST

Need for Secure GPU Computing

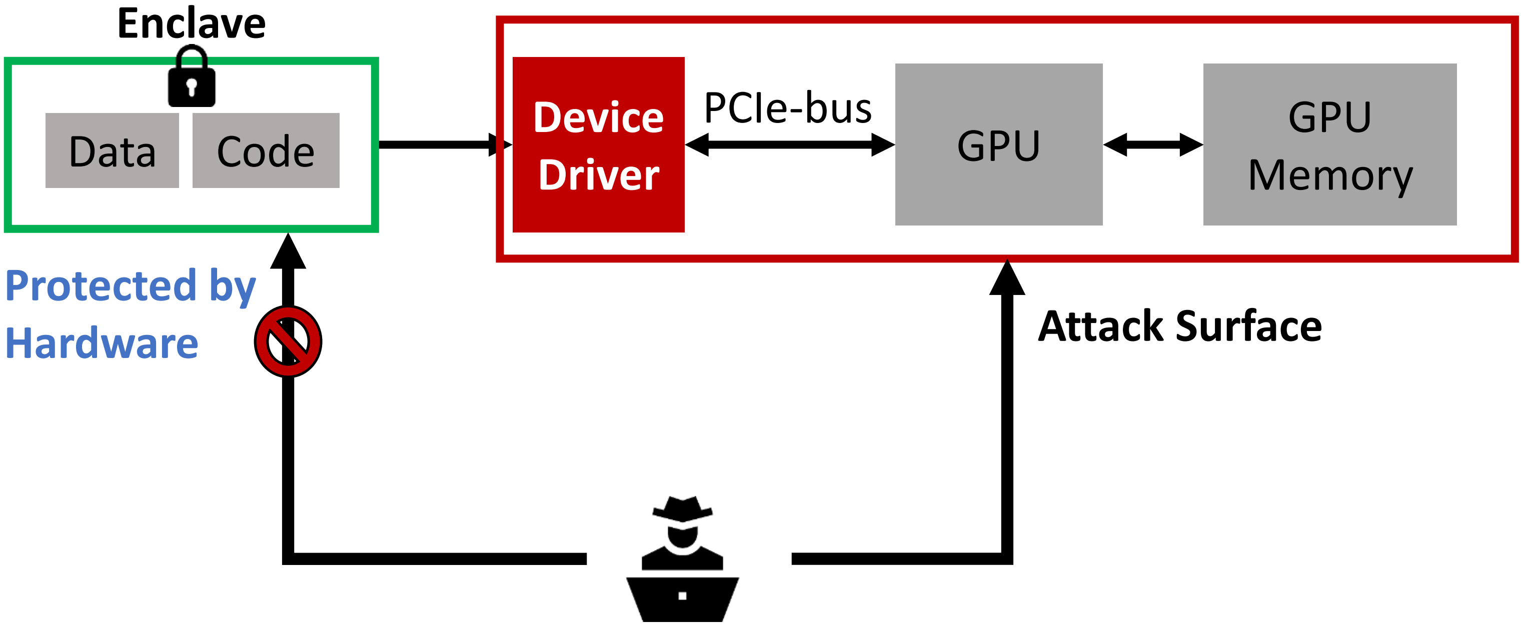


We need to consider Secure GPU computing !

Trusted GPU Computing

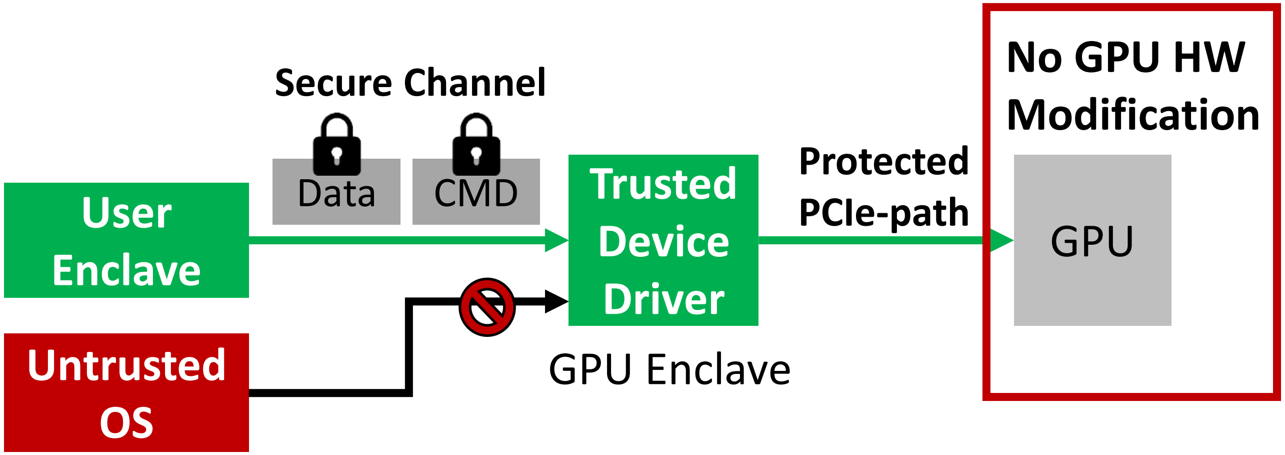
- Trusted Execution Environment (TEE)
    - Intel SGX, ARM TrustZone

- Existing TEEs **does not provide TEE on GPUs**



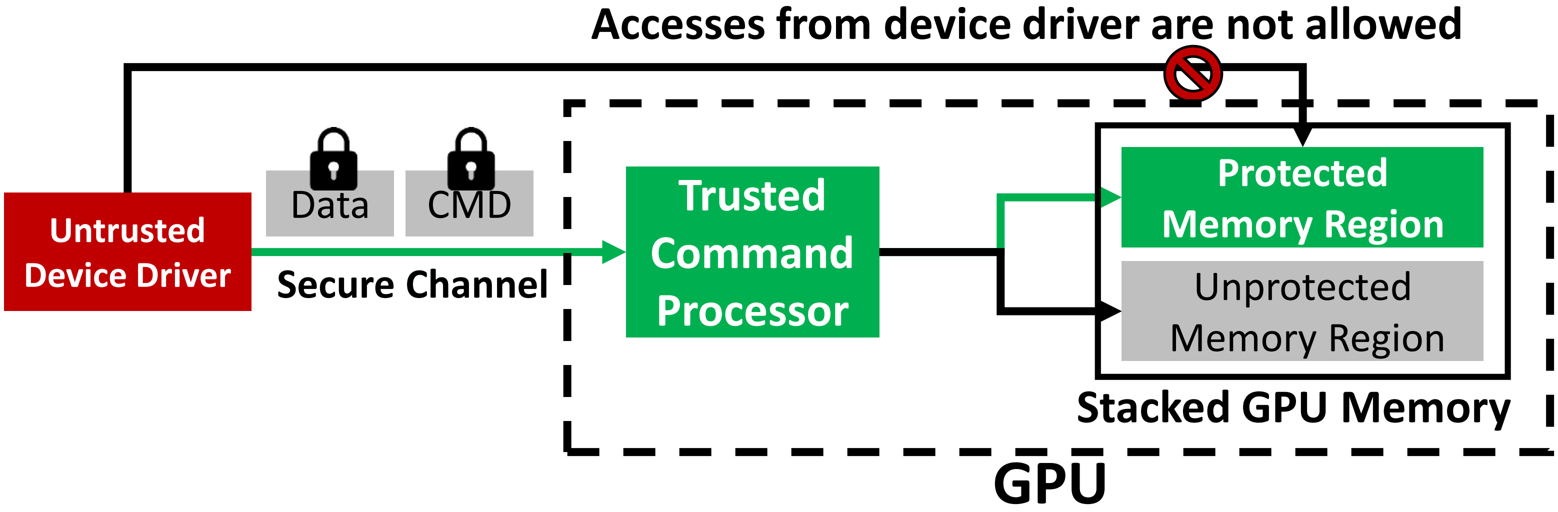
Prior Work : HIX

- **HIX [ASPLOS ‘19]: Securing I/O Path from CPU to GPU**
  - All device I/O accesses to GPU are controlled by **trusted device driver**



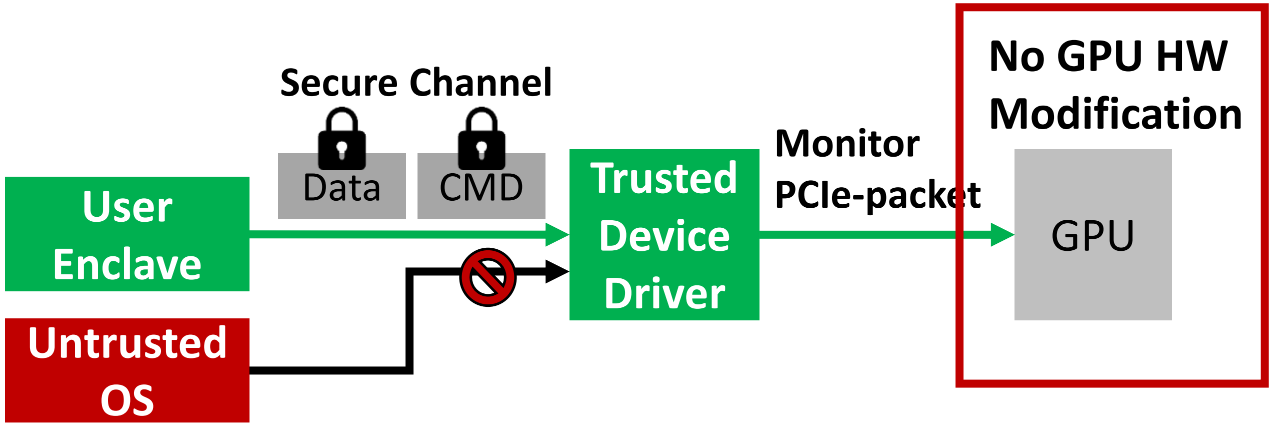
Prior Work : Graviton

- **Graviton [OSDI '18]: Trusted GPU by changing GPU HW**
  - **Trusted Command Processor** handles critical GPU operations instead of driver


Accesses from device driver are not allowed
Untrusted Device Driver
Data
CMD
Secure Channel
Trusted Command Processor
Protected Memory Region
Unprotected Memory Region
Stacked GPU Memory
GPU

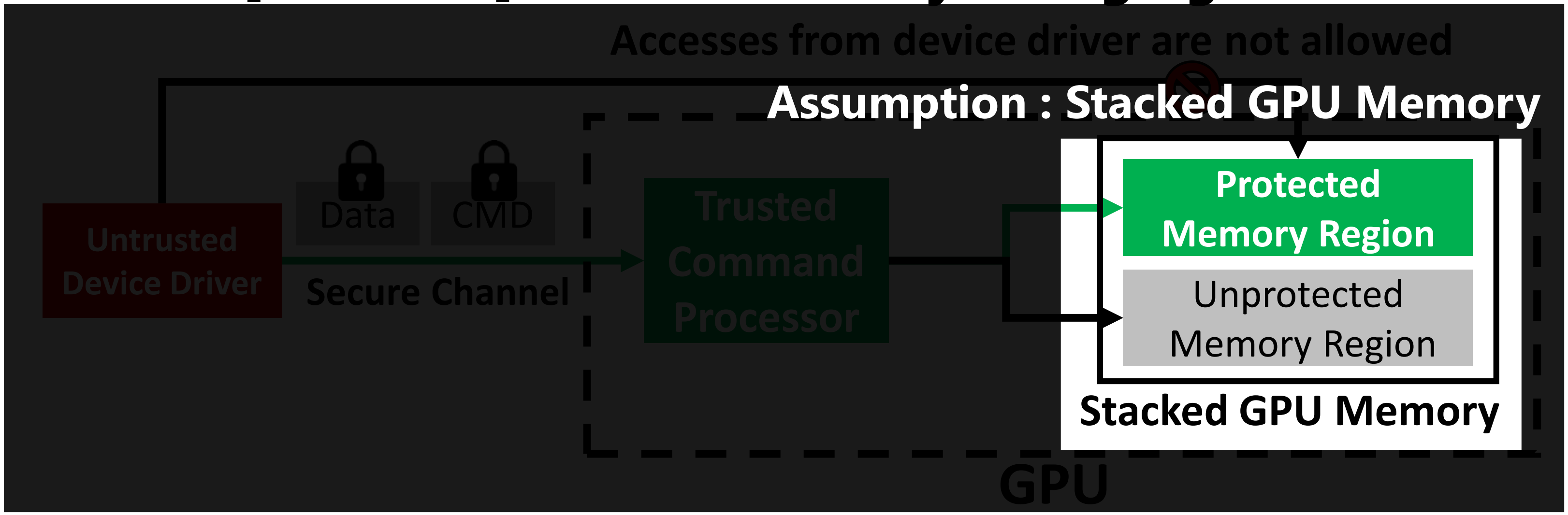
Limitations of Prior Work

- **HIX [ASPLOS '19]: Securing I/O Path from CPU to GPU**



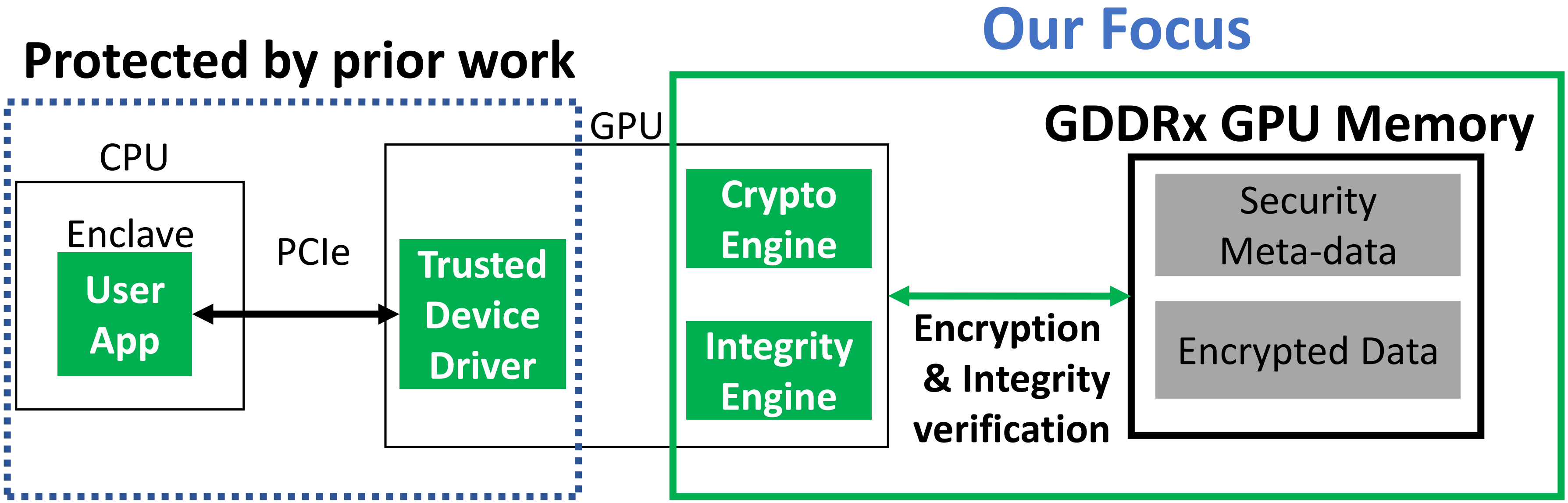
Out-of-scope : GPU Physical Attack

- **Graviton [OSDI '18]: Trusted GPU by changing GPU HW**


Accesses from device driver are not allowed
Untrusted Device Driver
Data
CMD
Secure Channel
Trusted Command Processor
Protected Memory Region
Unprotected Memory Region
Stacked GPU Memory
GPU

Assumption : Stacked GPU Memory

Goal: Secure GPU Memory



- **Main Contributions**
  - Provide secure GPU memory **with low performance overheads**
  - Exploit **unique memory update behavior** of common GPU applications
  - Reduce the average performance overhead to **2.9 %**

Threat Model & Assumptions

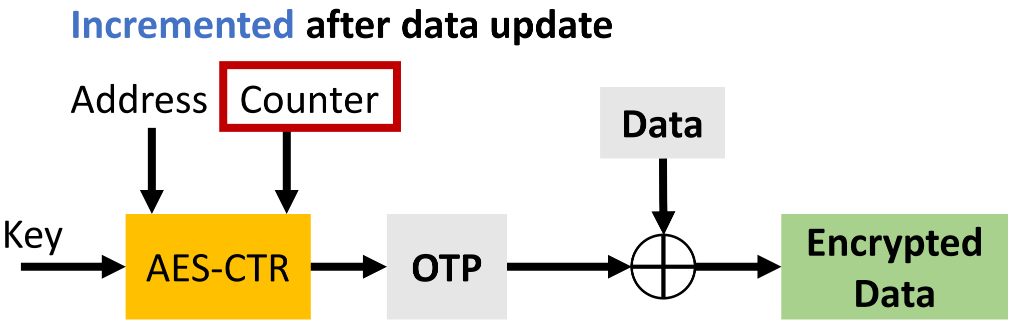
- Threat Model
  - Attackers can fully control **operating system/hypervisor**
  - Attackers can **physically access the whole system**
- Trusted Computing Base (TCB)
  - GPU processor & GPU software running on the GPU
  - CPU chip & user application in an CPU Enclave
- Out of Scope
  - Denial of Service(DoS) attacks
  - Side-channel attacks

Outline

- Introduction
- **Background & Motivation**
- Common Counter
  - Main Idea
  - Additional Metadata
  - Common Counter Mechanism
- Evaluation

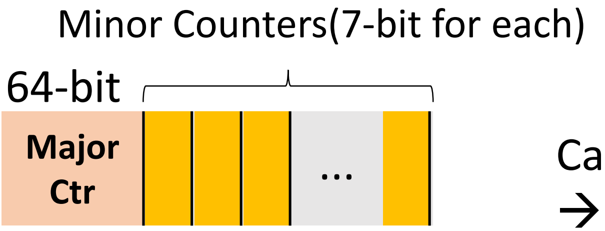
Background : Securing Memory

- Memory Encryption
  - Counter mode encryption



  - Split Counter scheme

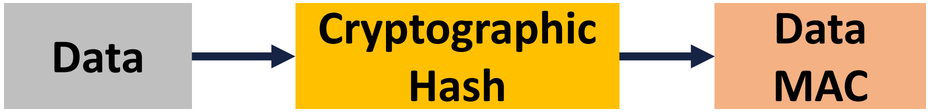
Counter = Major | Minor



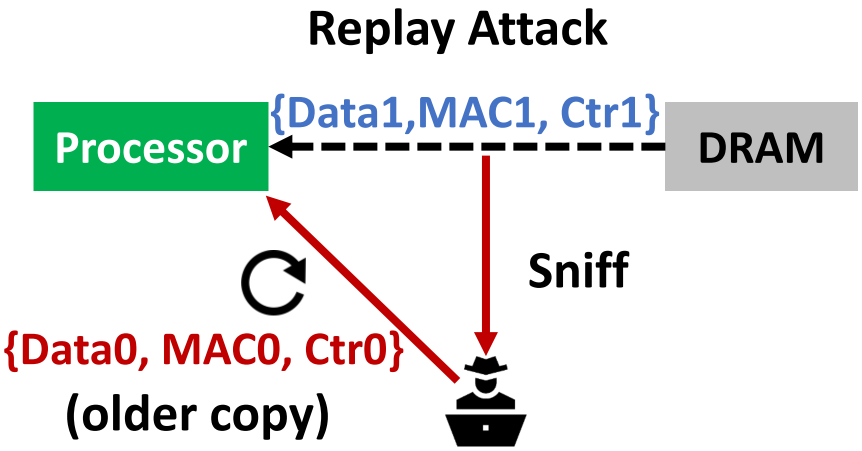
Cache block :128B
→ **128 minor counters**

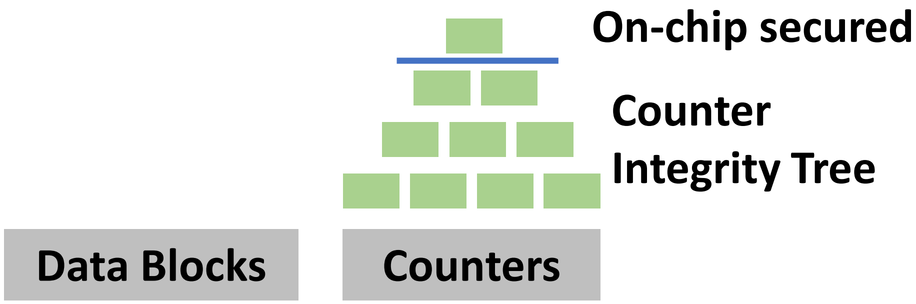
Background : Securing Memory

- Memory Integrity Verification
  - Message Authentication Code (MAC)



  - Counter Integrity Tree



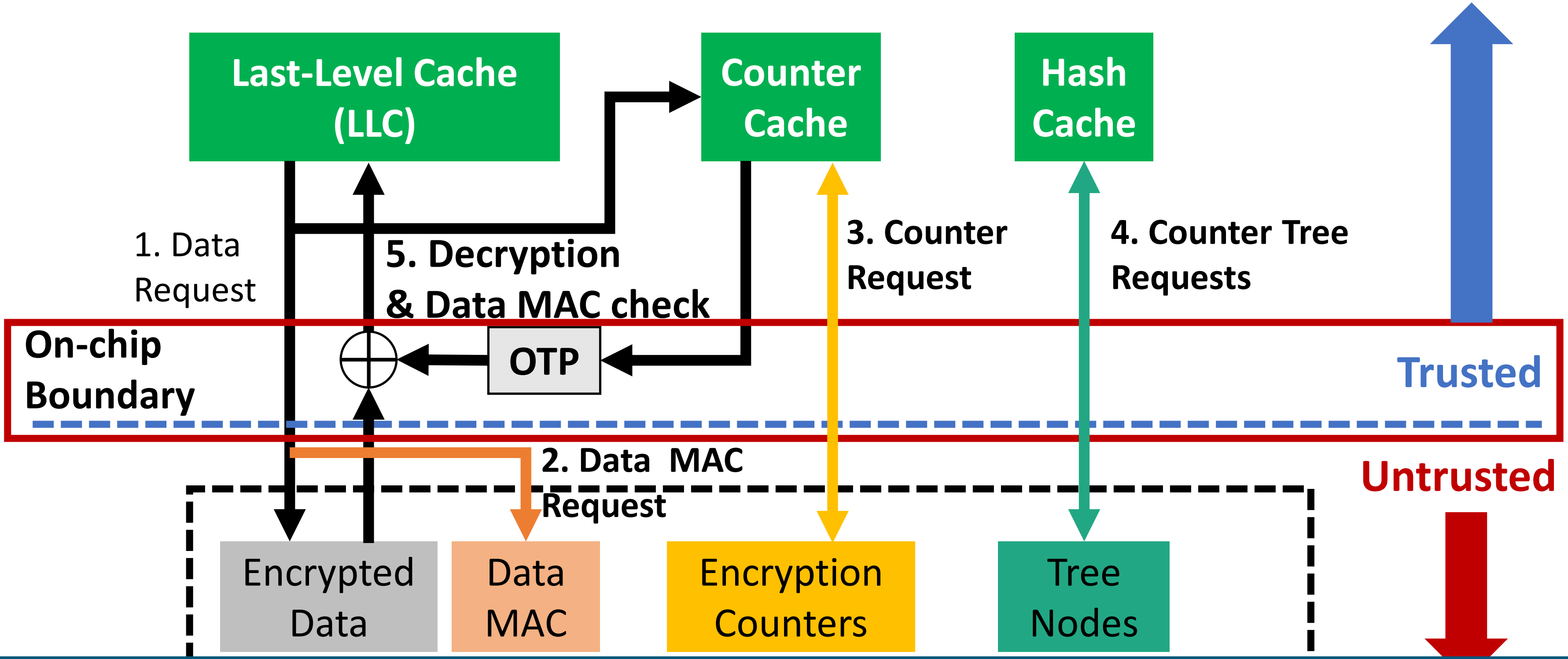


Our baseline : **SC-128**

128-ary (Split Counter + Counter integrity tree)

Problem : Performance Overhead

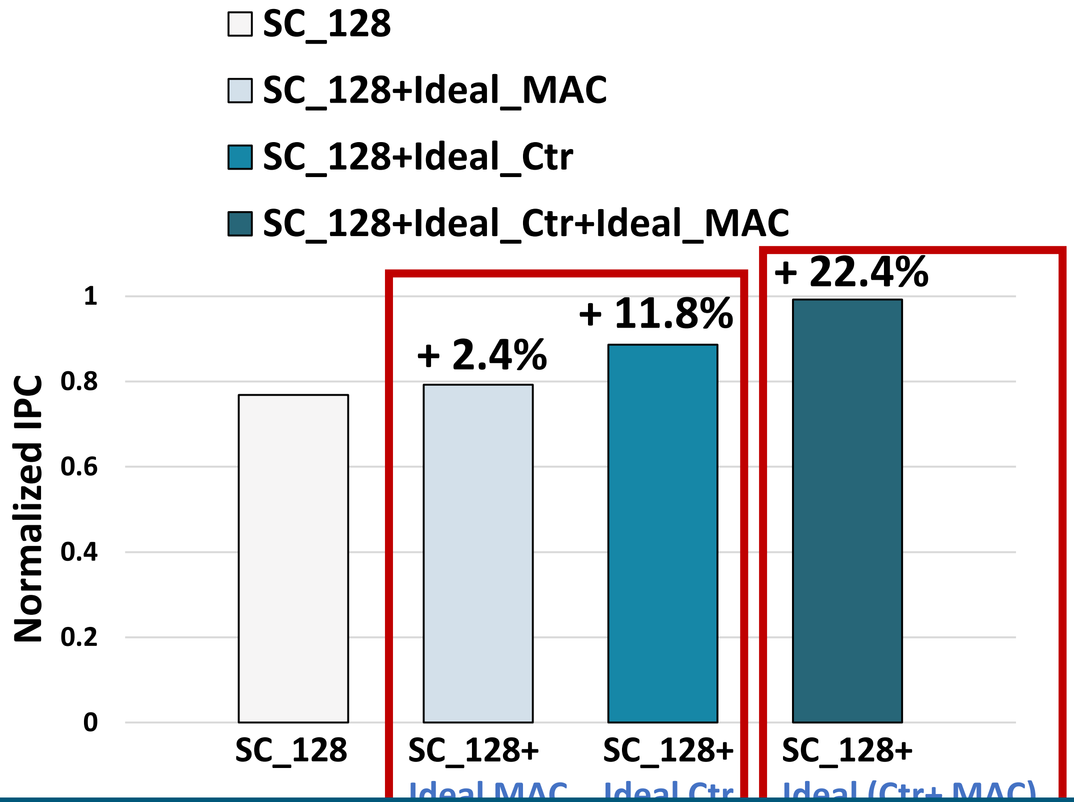
- Secure memory require **additional meta-data requests**



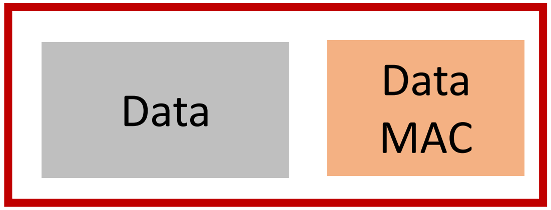
Secure memory **adds decryption latency** and **increases memory bandwidth**

Performance Breakdown Analysis

- GPU memory protection overhead result for GPU benchmark suites



For data MAC overhead



With ECC memory,
Data & MAC can be provided by
1 memory access using Synergy[1]

Counter mode encryption is one of the key bottlenecks

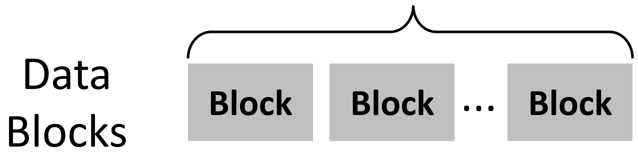
[1] :SYNERGY: Rethinking Secure-Memory Design for Error-Correcting Memories, HPCA'18

Uniformly Updated Segments

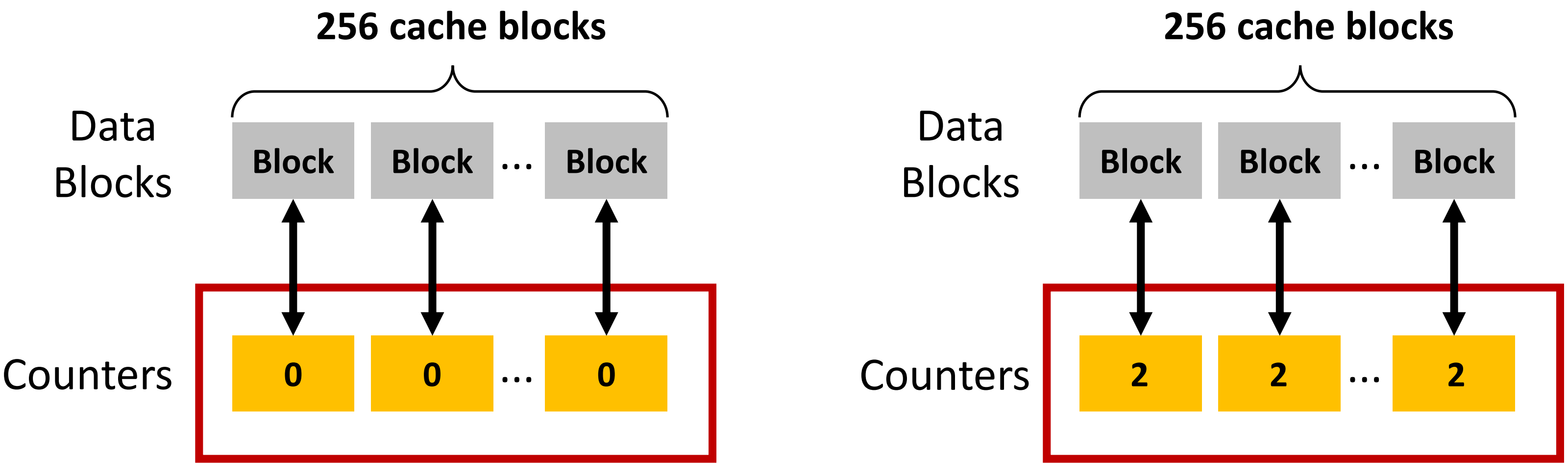
- Memory segment: Contiguous memory region

Example granularity: 32KB

32KB/ 128B = **256 Data cache blocks**



- Uniformly updated segment: **Read-only** **+** **uniformly written**

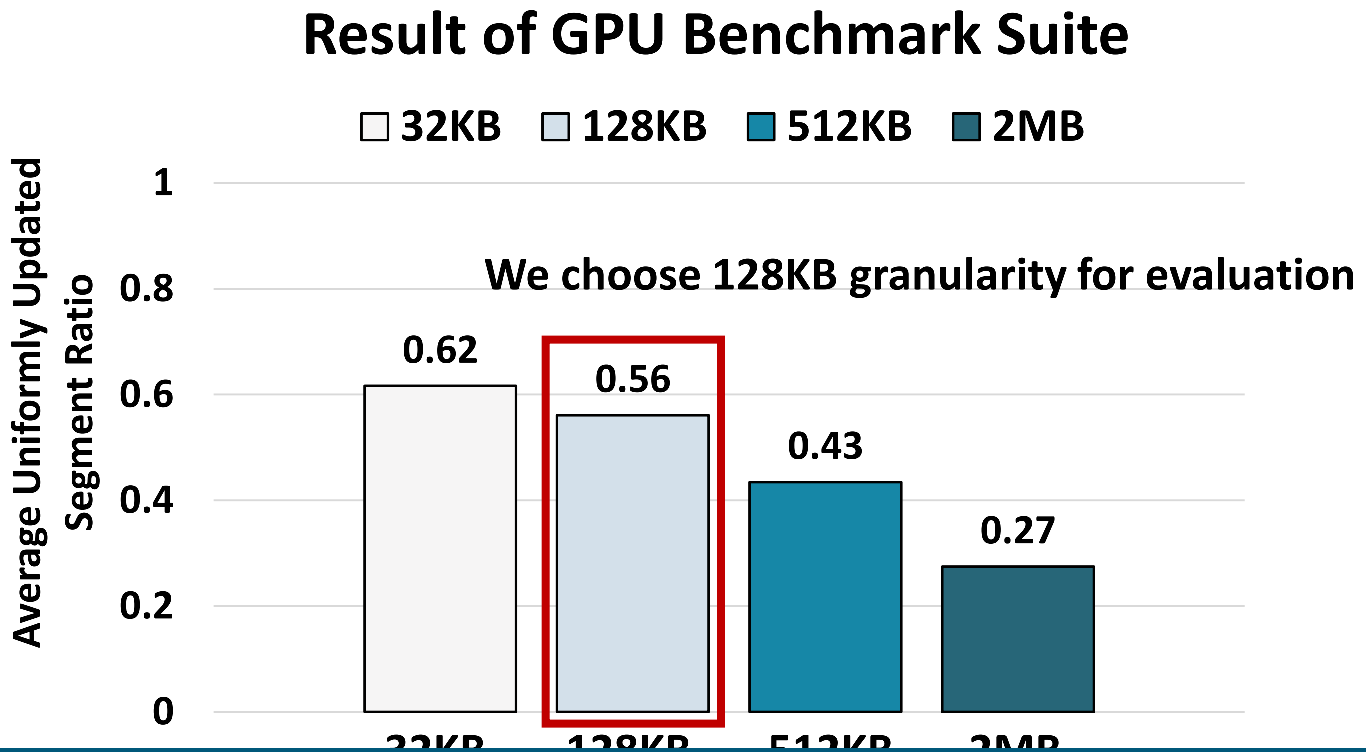


1. Read-only

2. Uniformly written

Observation : GPU SW Write Patterns

- Analyze memory read/write behavior by using **NVBit [MICRO '19]**



We choose 128KB granularity for evaluation

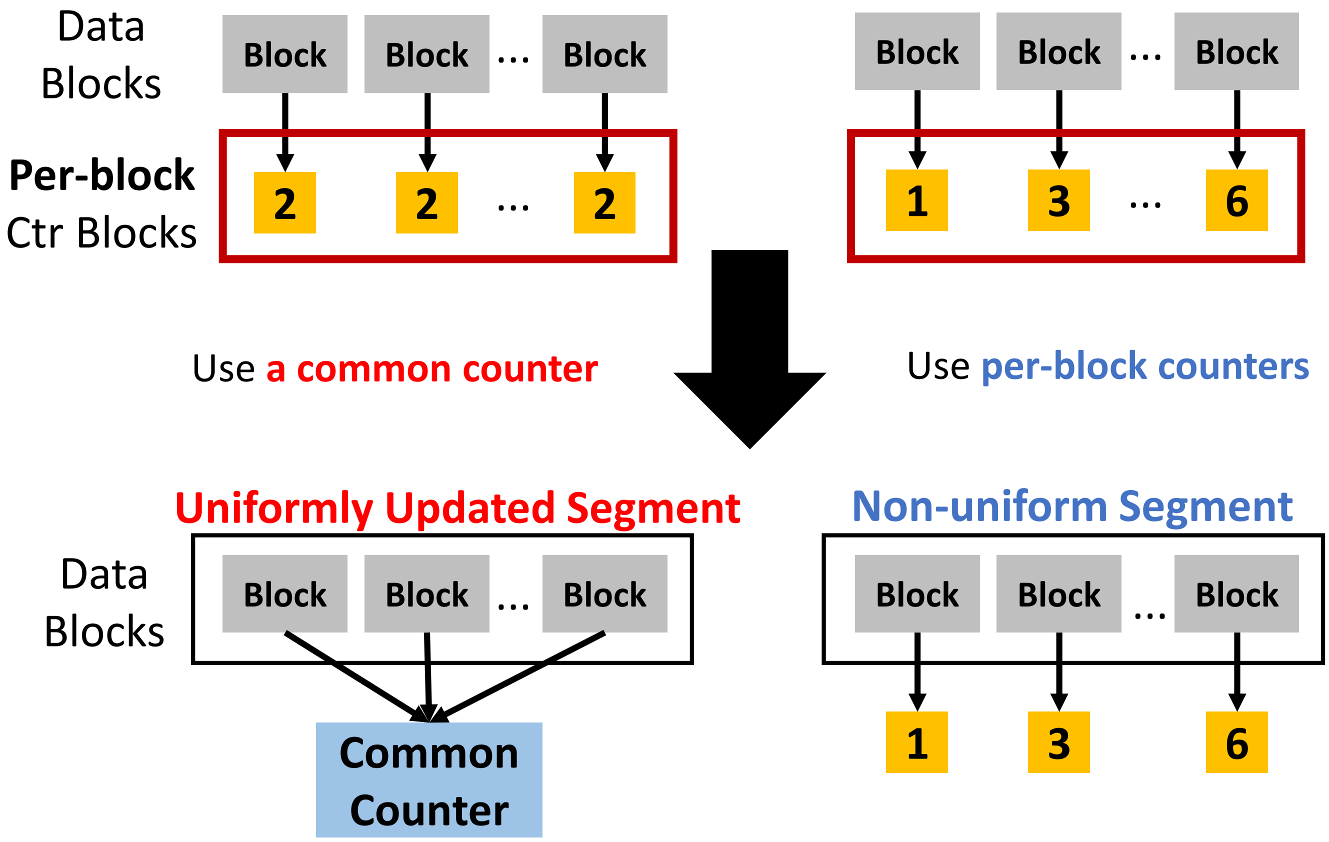
Observation 1**: GPU programs tend to uniformly update memory**
Observation 2**: The number of distinct counter values is small**

Outline

- Introduction
- Background & Motivation
- **Common Counter**
  - **Main Idea**
  - **Additional Metadata**
  - **Common Counter Mechanism**
- Evaluation

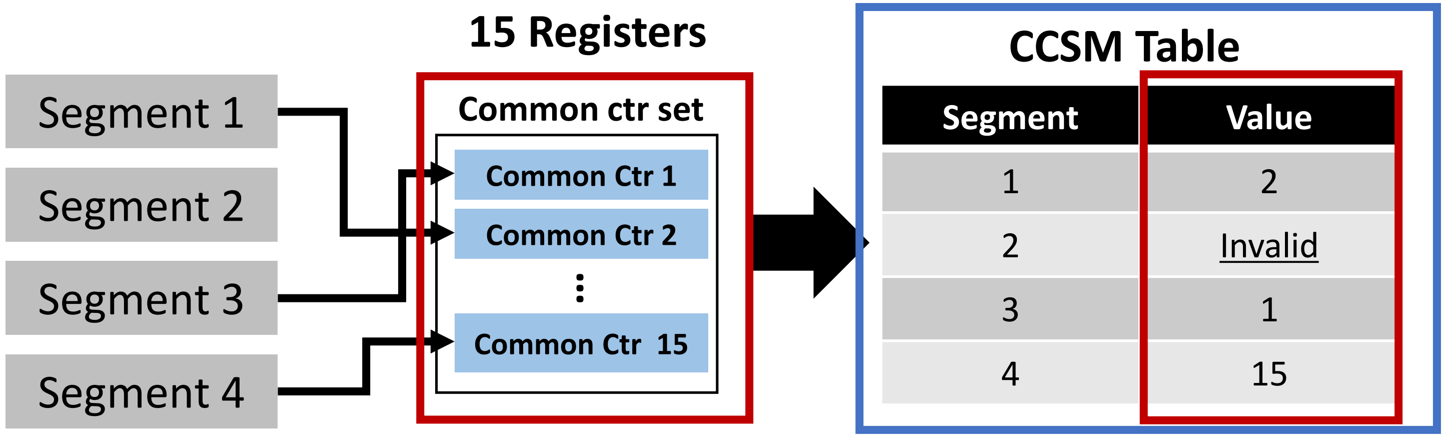
Common Counter : Main Idea

- Use **coarse-grained counters** for uniformly updated segments



Finding Uniformly Updated Segments

- Common Counter Status Map (CCSM)
  - Check whether a memory segment uses a common counter or not



Can be cached in **CCSM cache**

CCSM Table

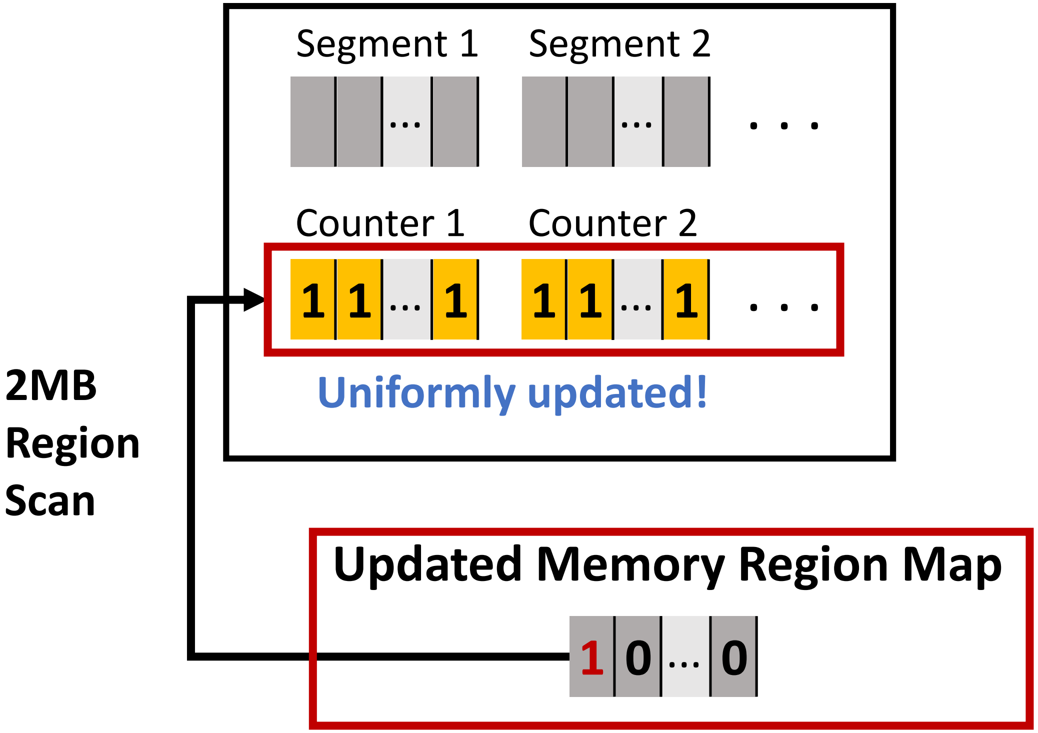
Segment	Value
1	2
2	<u>Invalid</u>
3	1
4	15

Index of common counter

<u>Invalid</u> : not uniformly updated segment

Updating CCSM Table

- Initialized at application launch
- Scanning Procedure
  - When? After a kernel is completed

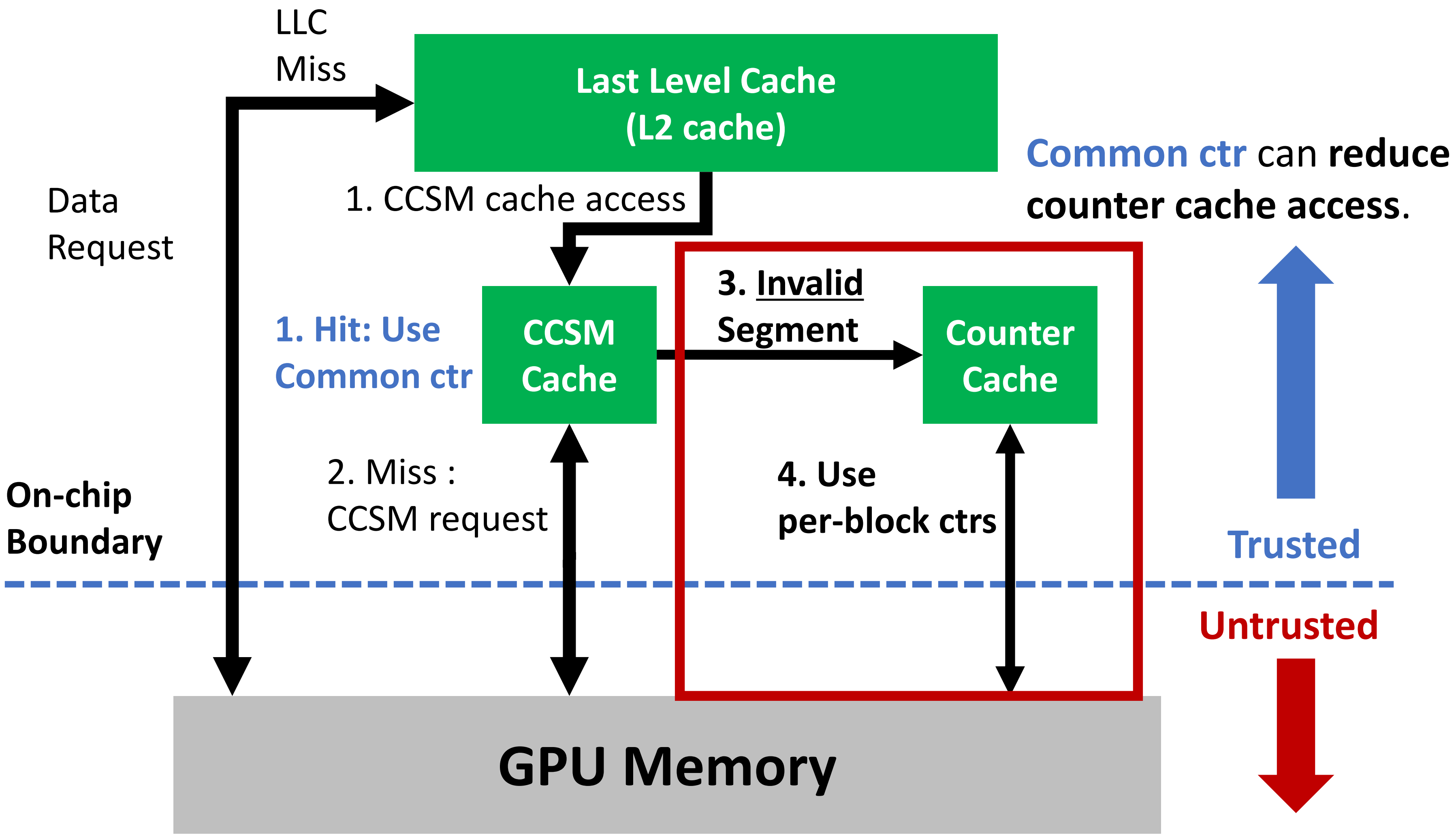


Bit vector for recording memory update status at 2MB granularity

CCSM Table

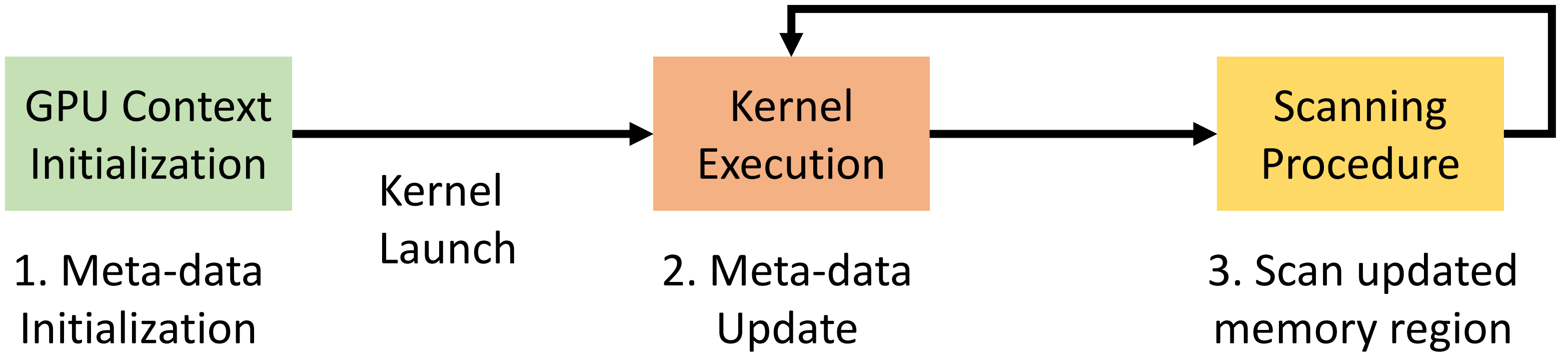
Segment	Value
1	Invalid 1
2	1
⋮	⋮

LLC Miss Handling with Common Counters



Common ctr can **reduce counter cache access.**

GPU Execution with Common Counter



GPU Context Initialization

Kernel Launch

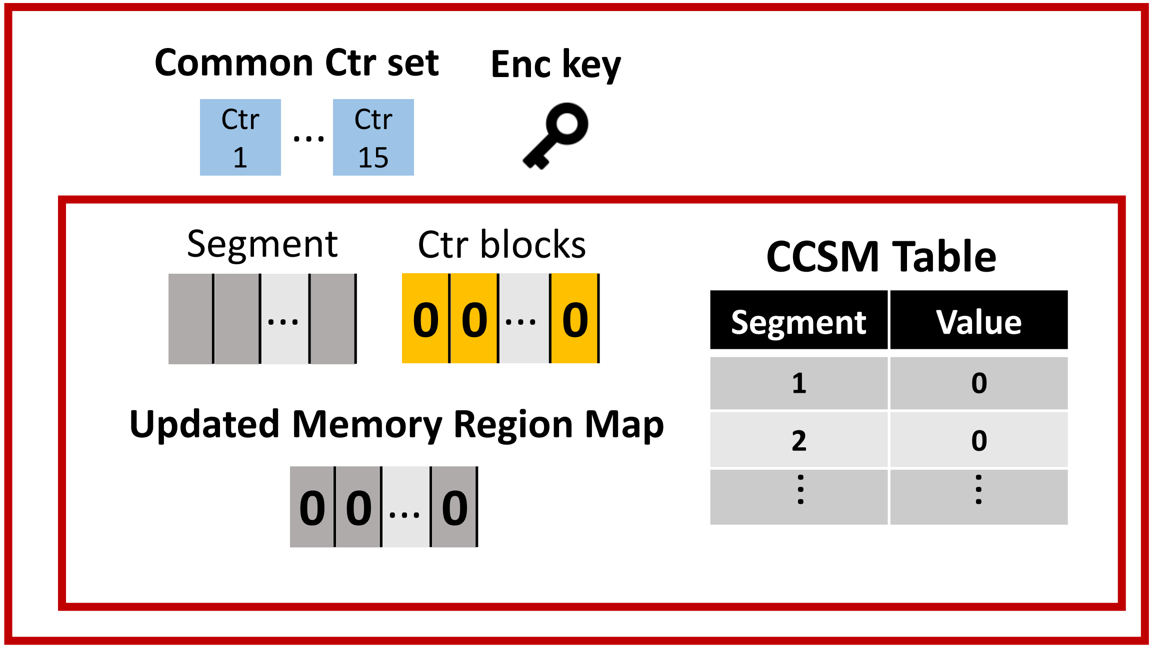
Kernel Execution

Scanning Procedure

1. Meta-data Initialization

2. Meta-data Update

3. Scan updated memory region



CCSM Table

Segment	Value
1	0
2	0
⋮	⋮

Outline

- Introduction
- Background & Motivation
- Common Counter
  - Main Idea
  - Additional Metadata
  - Common Counter Mechanism
- **Evaluation**

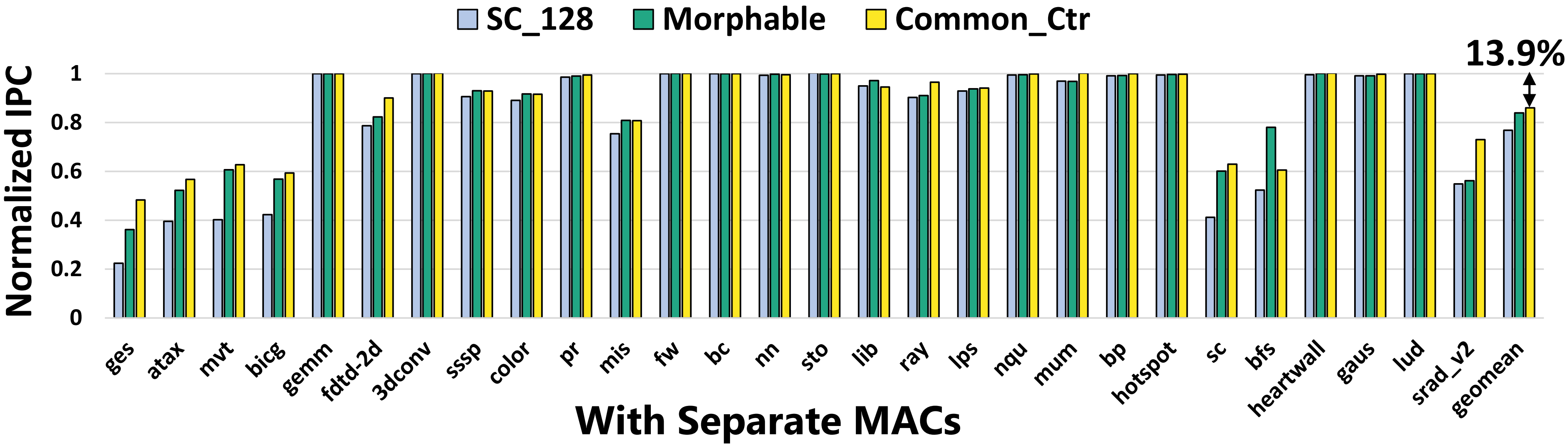
Methodology

- Simulator: GPGPU-Sim
- Workloads: ISPASS, Rodinia, Polybench, Pannotia
- System configuration: Models NVIDIA TITAN X Pascal GPU

GPU Core Configuration	
System overview	28 SMs, 64 warps per SM
Shader core	1,417 MHz, GTO Scheduler
Cache & Memory Configuration	
L1 cache	48 KB
L2 cache	3 MB
DRAM	GDDR5X 1,251 MHz, 12 GB
Counter cache, Hash cache	**16 KB**
CCSM cache	**1 KB**
Segment size	**128 KB**
Number of common ctrs	**15**

Performance: Separate MACs

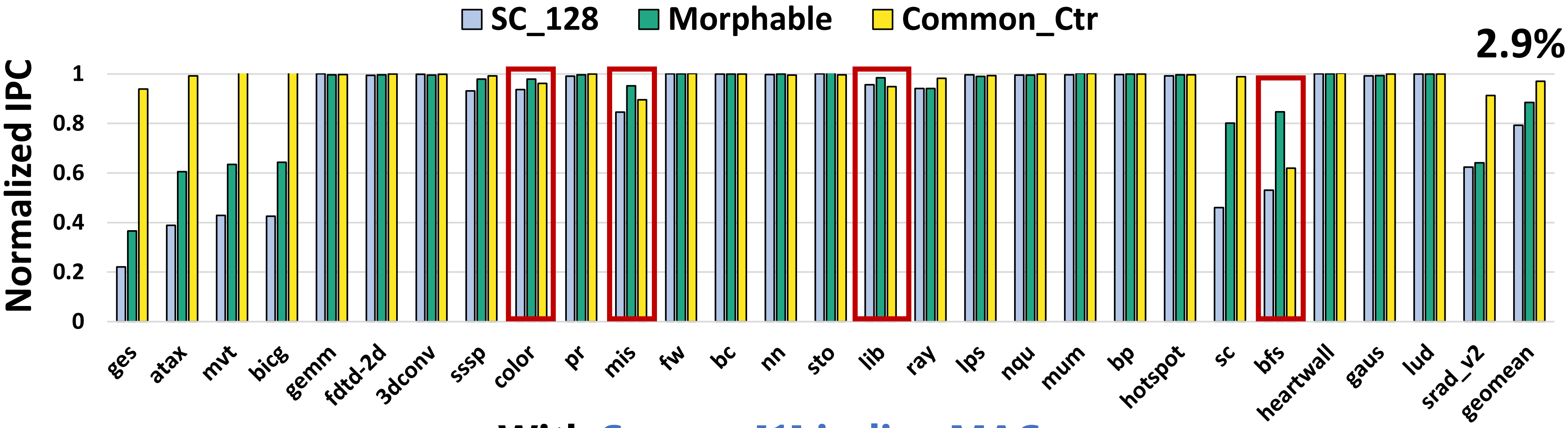
- Performance overhead analysis (Baseline: Non-secure GPU)
  - SC-128: **128-arity** split counter
  - Morphable Counter [MICRO '18]: **256-arity** split counter
  - Common_Ctr: Implemented on top of SC-128 (**128-arity**)



Performance: Synergy In-line MACs

- **Common counter** reduces the performance degradation to 2.9%

color, mis,lib,bfs : As kernel runs, # of requests served by **common counters** decreases.



With Synergy[1] in-line MACs

[1] :SYNERGY: Rethinking Secure-Memory Design for Error-Correcting Memories, HPCA'18

More Results in the Paper

- Uniformly updated ratios of real-world GPU Applications
- Hardware area/energy cost for common counter mechanism
- Ratios of LLC misses served by common counters
- Scanning Overheads
- Counter cache sensitivity experiments

Please Refer to our paper for more details!

Conclusion

- **Result**
  - **Common Counter** reduces the performance degradation to **2.9%**

- **Problem**
  - **Memory encryption** is one of the critical bottlenecks for secure GPU memory

- **Key Observation**
  - GPU programs tend to uniformly update memory
  - The number of distinct common counters is small

- **Our Approach**
  - **Common Counter** provides compressed representation of per-block counters

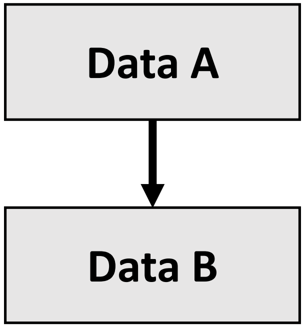
Backup Slides

Type of Physical Attacks

Unauthorized Data Leak

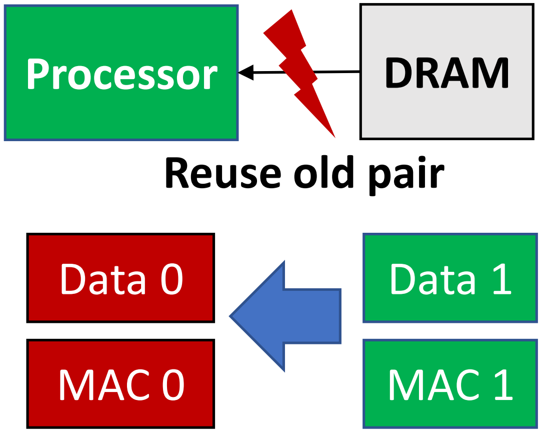
Cold Boot Attack

Unauthorized Data Modification



DMA Attack

Replay Attack



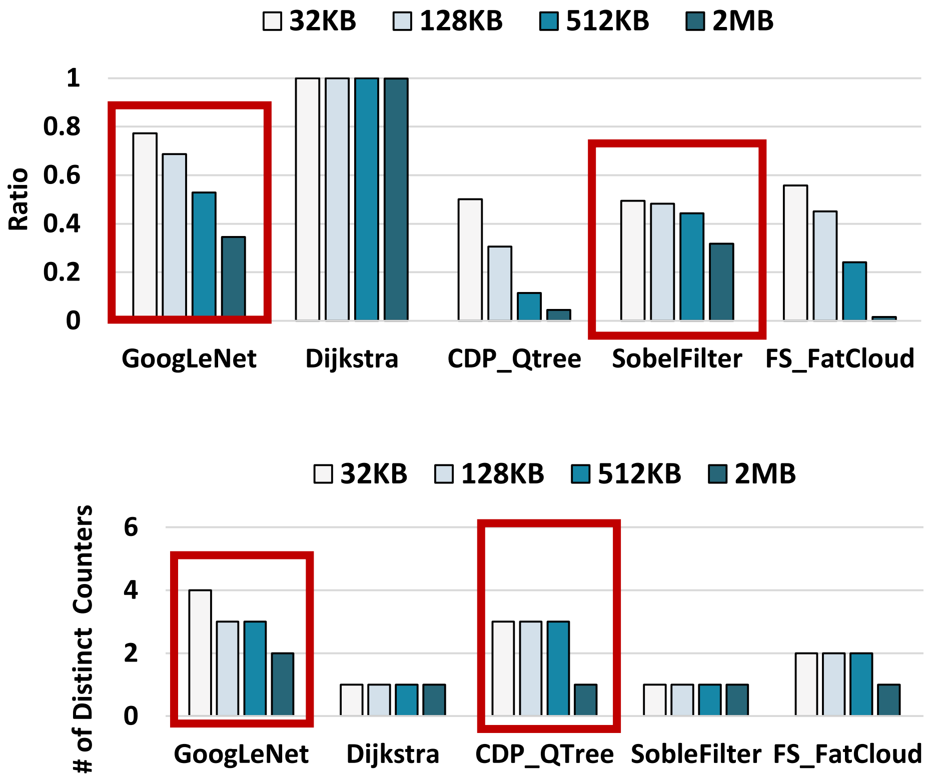
Man in the middle attack

Secure memory needs to protect against these attacks !

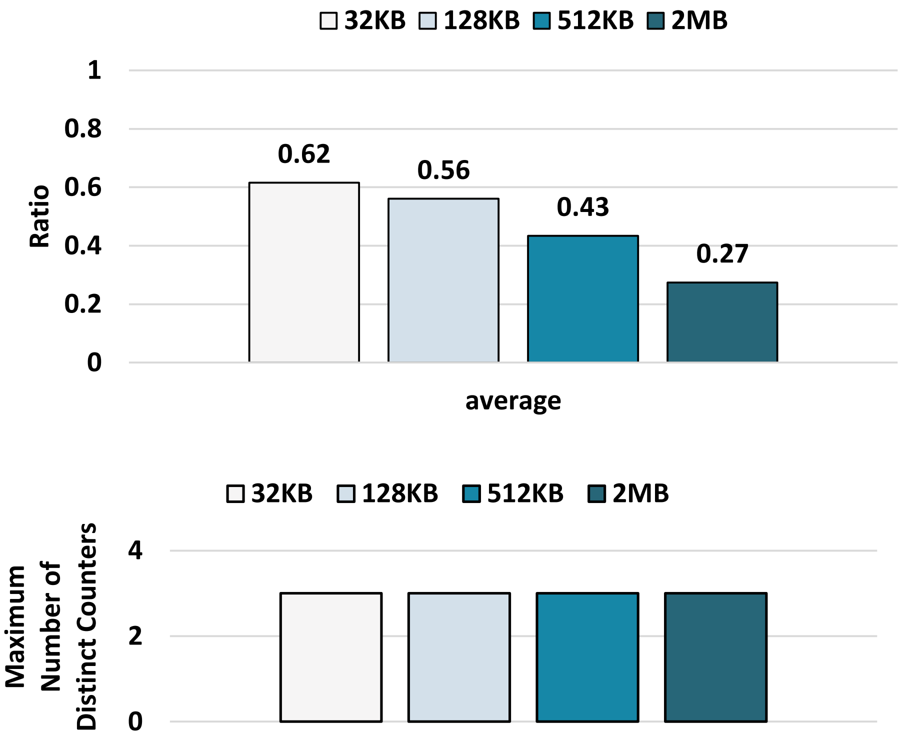
GPU SW Memory Write Patterns

- Analyze memory read/write behavior by using **NVBit [1]**
  - Collect traces for load/store instructions

Result of Real-World GPU Applications



Result of GPU Benchmark Suite

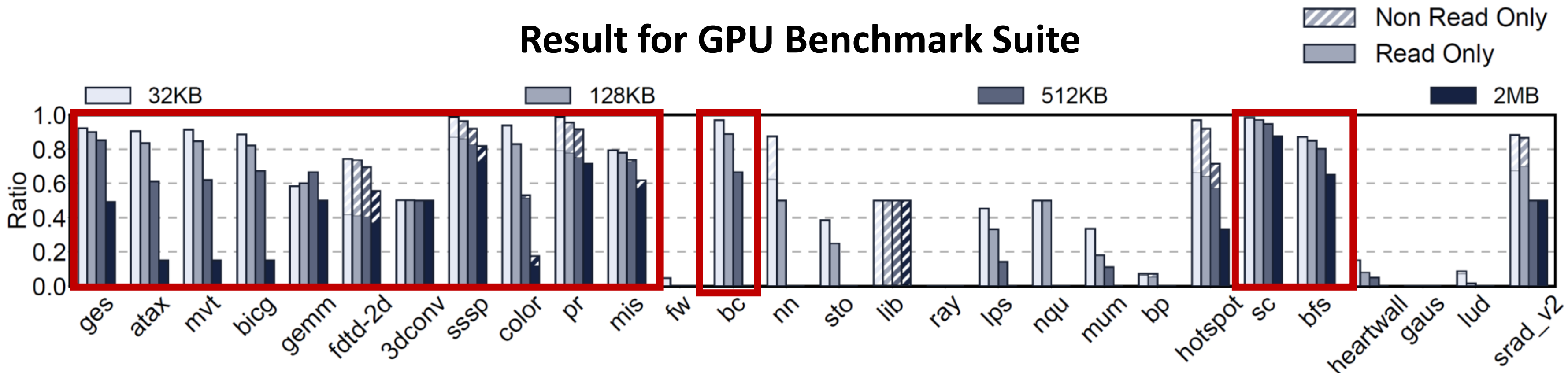


[1] : NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs, MICRO'19

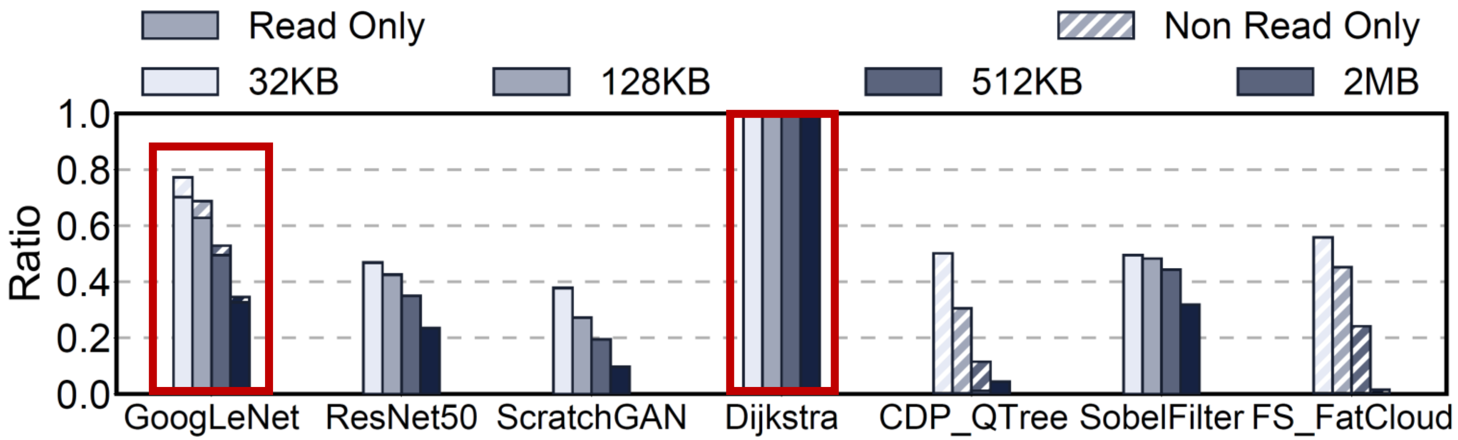
GPU SW Memory Write Patterns

- Analyze

Result for GPU Benchmark Suite

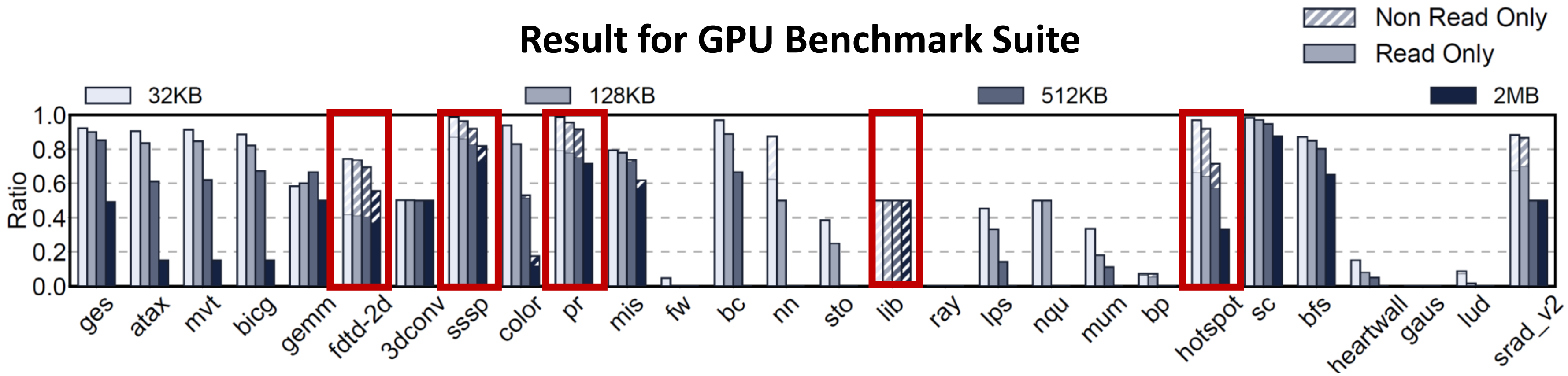


Result for Real-World GPU Applications

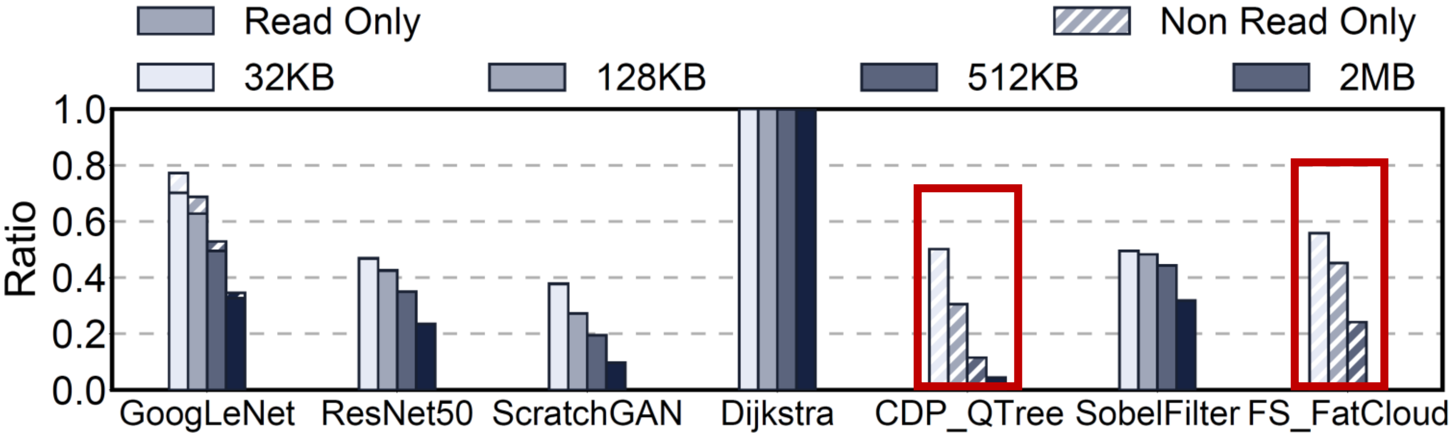


GPU SW Memory Write Analysis

Result for GPU Benchmark Suite



Result for Real-World GPU Applications



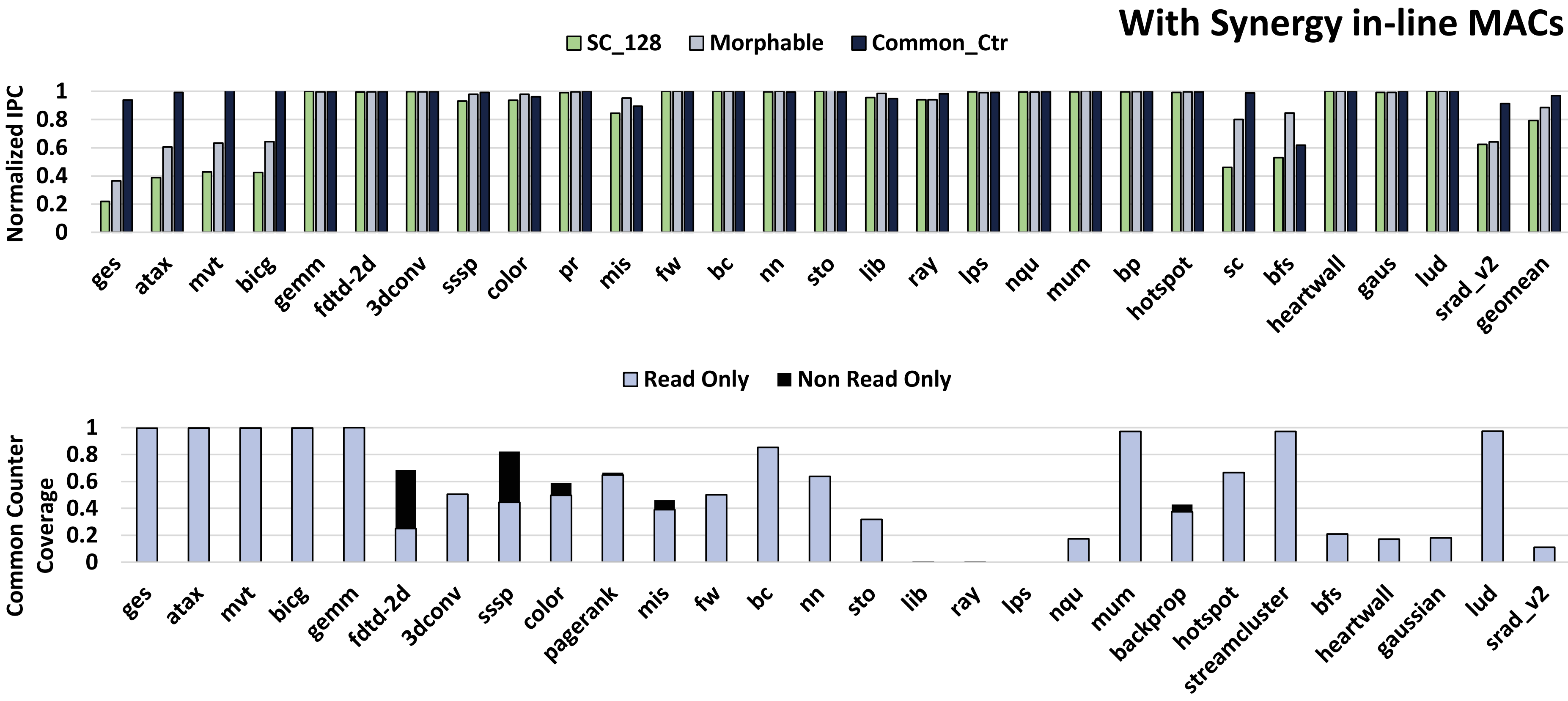
Coverage

- Coverage comparison

Scheme	Granularity	Per-block Coverage
split counter	128B data block	128 * 16KB = **16KB data**
Common counter	128 KB data block	256 * 128KB = **32MB data**

2048x efficient coverage

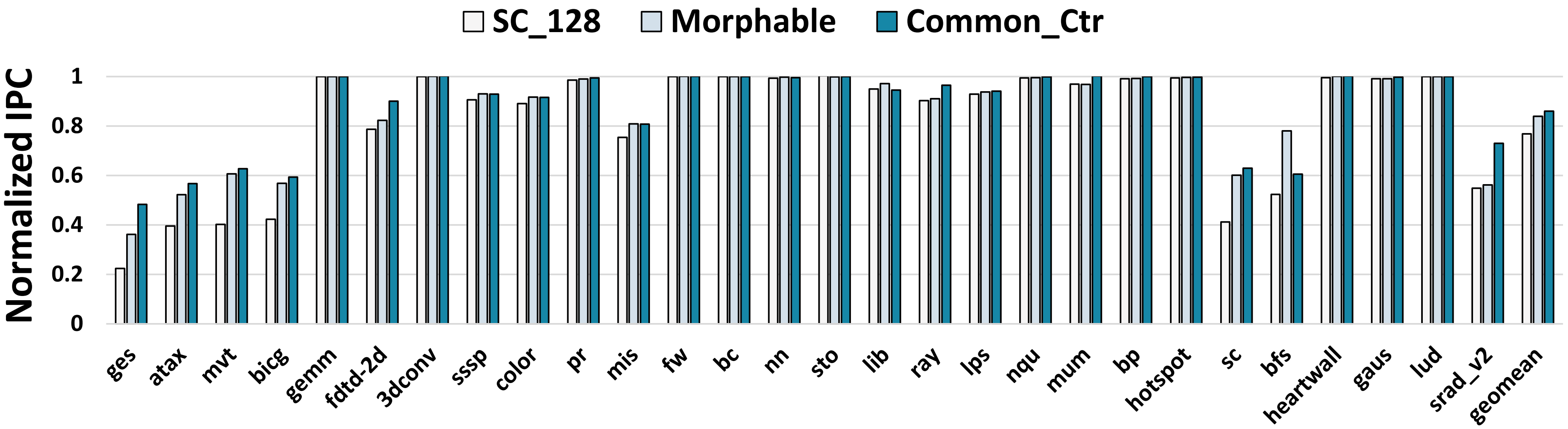
Performance Result & Counter Coverage

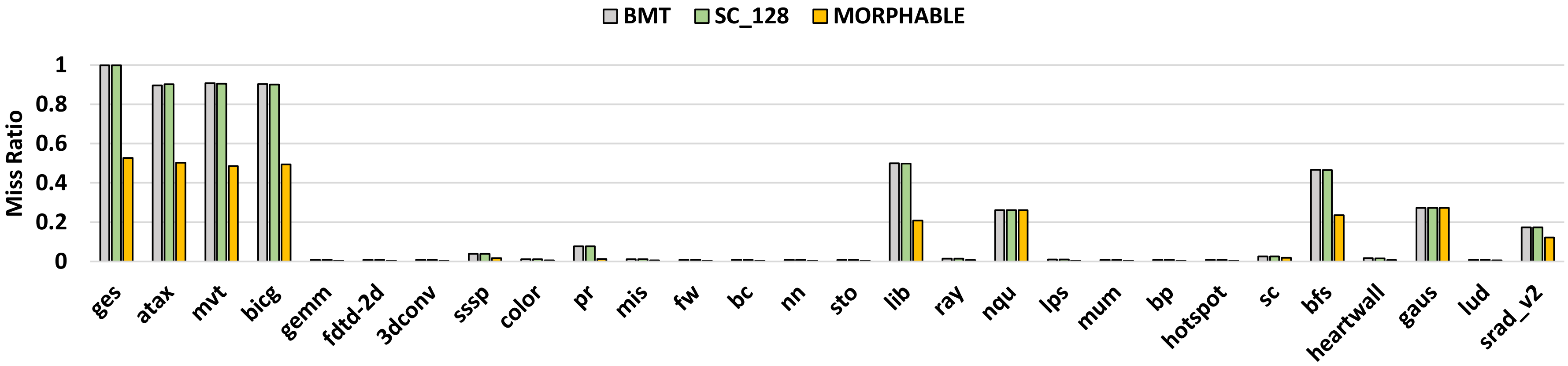


CASYS Computer Architecture and Systems Lab

KAIST

Counter Cache Miss ratio

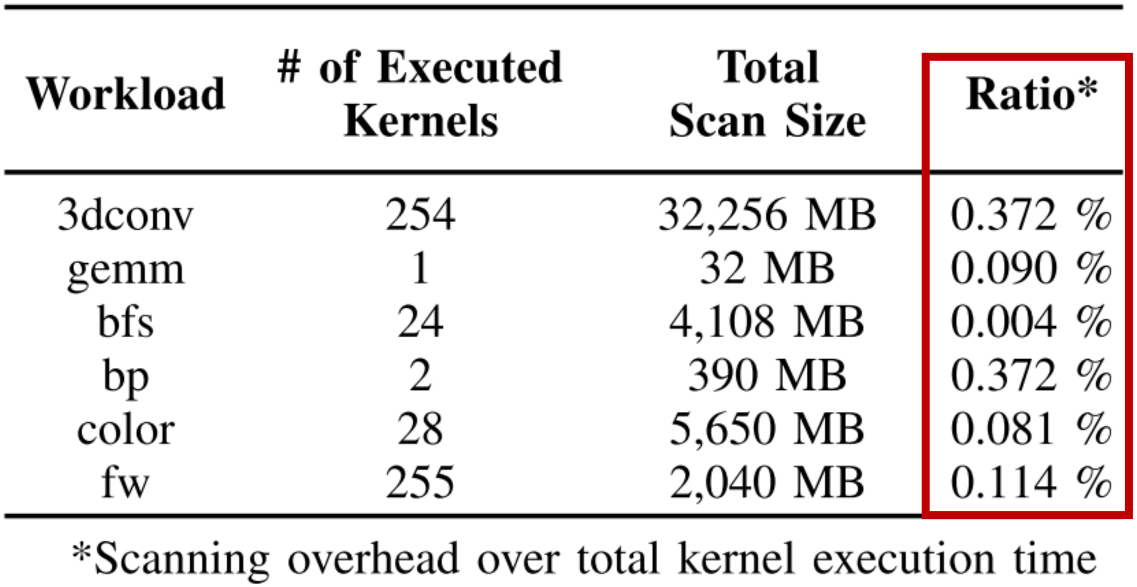




Scanning Overhead

- Evaluate scanning procedure

Neglig



Workload	# of Executed Kernels	Total Scan Size	Ratio*
3dconv	254	32,256 MB	0.372 %
gemm	1	32 MB	0.090 %
bfs	24	4,108 MB	0.004 %
bp	2	390 MB	0.372 %
color	28	5,650 MB	0.081 %
fw	255	2,040 MB	0.114 %

*Scanning overhead over total kernel execution time