Improving the Performance and Endurance of Encrypted Non-volatile Main Memory through Deduplicating Writes

Pengfei Zuo, Yu Hua, Ming Zhao*, Wen Zhou, Yuncheng Guo
Huazhong University of Science and Technology (HUST), China
**Arizona State University (ASU), USA*

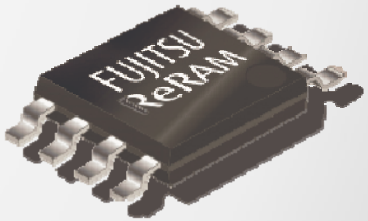
The 51st IEEE/ACM International Symposium on Microarchitecture (**MICRO**), 2018

Non-volatile Memory (NVM)

- Non-volatile memory is expected to replace or complement DRAM in memory hierarchy
  - ✓ Non-volatility, low power, high density, large capacity

	PCM	ReRAM	DRAM
Read (*ns*)	20-70	20-50	10
Write (*ns*)	150-220	70-140	10
Non-volatility	√	√	×
Standby Power	~0	~0	High
Endurance	10^7~10^9	10^8~10^{12}	10^{15}
Density (*Gb/cm²*)	13.5	24.5	9.1

PCM



ReRAM

C. Xu et al. "Overcoming the Challenges of Crossbar Resistive Memory Architectures", HPCA, 2015.
K. Suzuki and S. Swanson. "A Survey of Trends in Non-Volatile Memory Technologies: 2000-2014", IMW 2015.

Endurance and Security in Non-volatile Memory

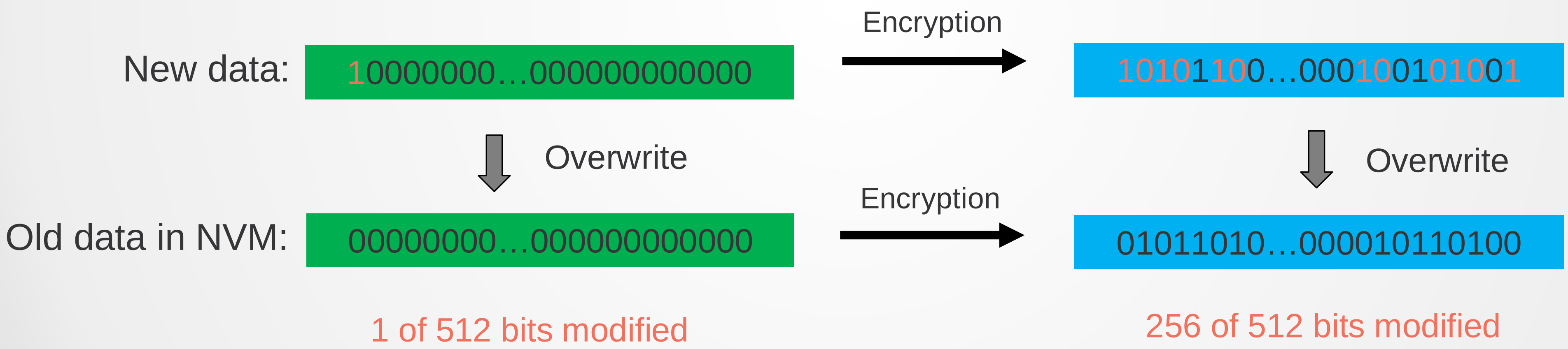
- NVM typically has limited endurance
  - 10^7~10^9 for PCM, 10^8~10^{12} for ReRAM
  - Writes have much higher latency than reads
  - Write reduction matters for NVM

However, memory encryption increases writes to NVM

  - NVM still retains data after systems power down
  - An attacker can directly read data from the stolen NVM
  - Memory encryption matters for NVM

Encryption Increases Bit Flips to NVM

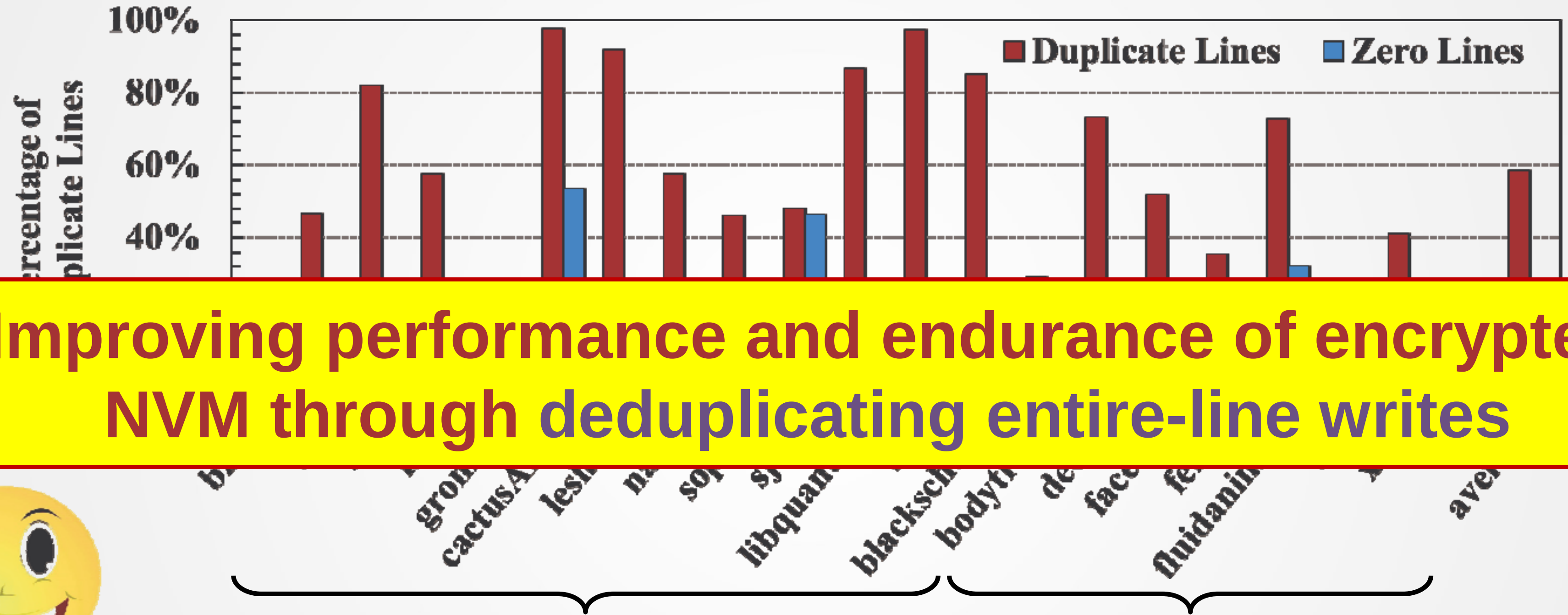
- Diffusion property of encryption
  - The change of one bit in the original data has to modify half of bits in the encrypted data



Encryption Increases Bit Flips to NVM

Encryption renders existing bit-level write reduction techniques ineffective

Observation and Motivation



Improving performance and endurance of encrypted NVM through deduplicating entire-line writes

SPEC CPU2006

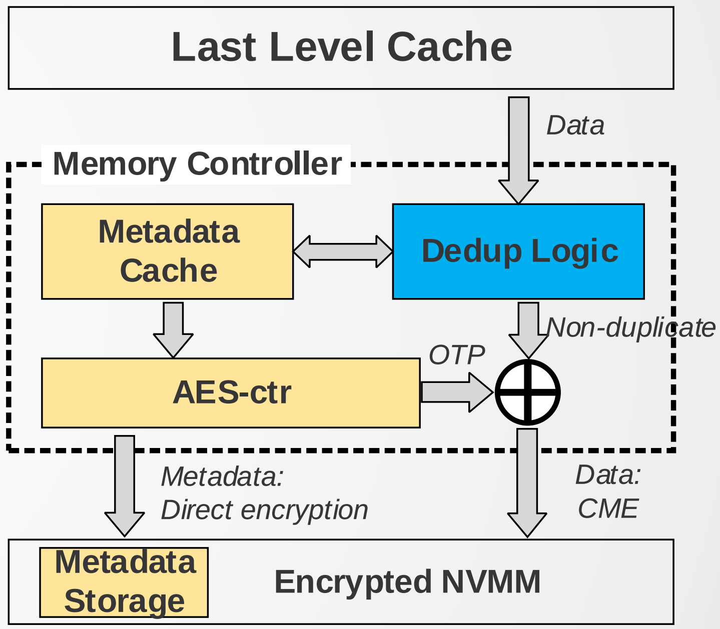
PARSEC 2.1

- A large number of **entire-line** duplicates exist in real-world applications

DeWrite

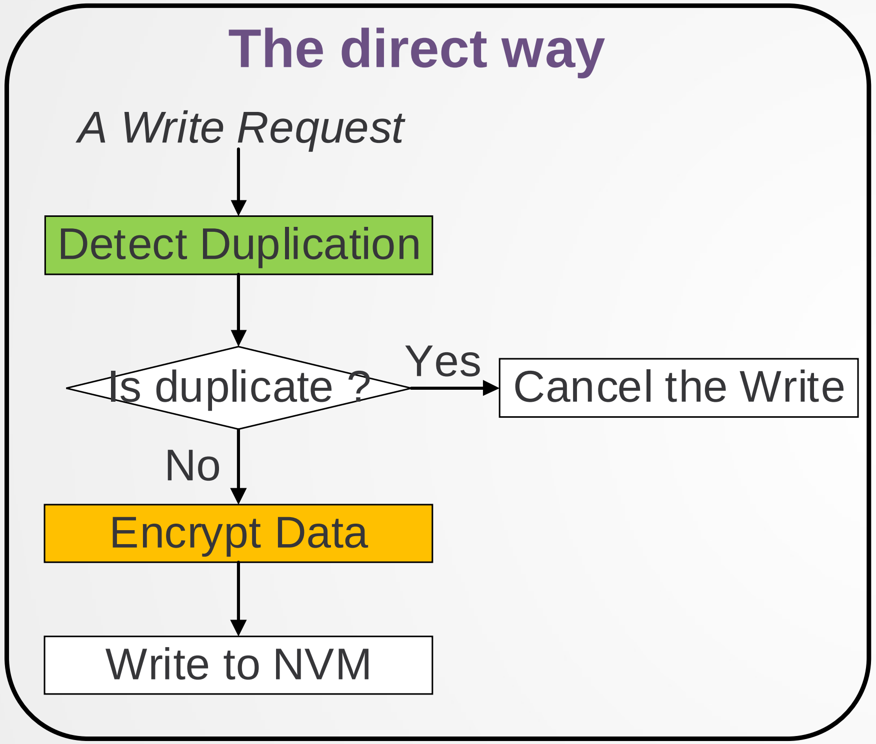
- **Lightweight cache-line-level deduplication for NVMM**
  - Employ lightweight hashing
  - Leverage NVM read/write asymmetry
  - Eliminate a write at the cost of a read

- **Efficient synergization between deduplication and encryption**
  - Opportunistic parallelism
  - Metadata storage co-location



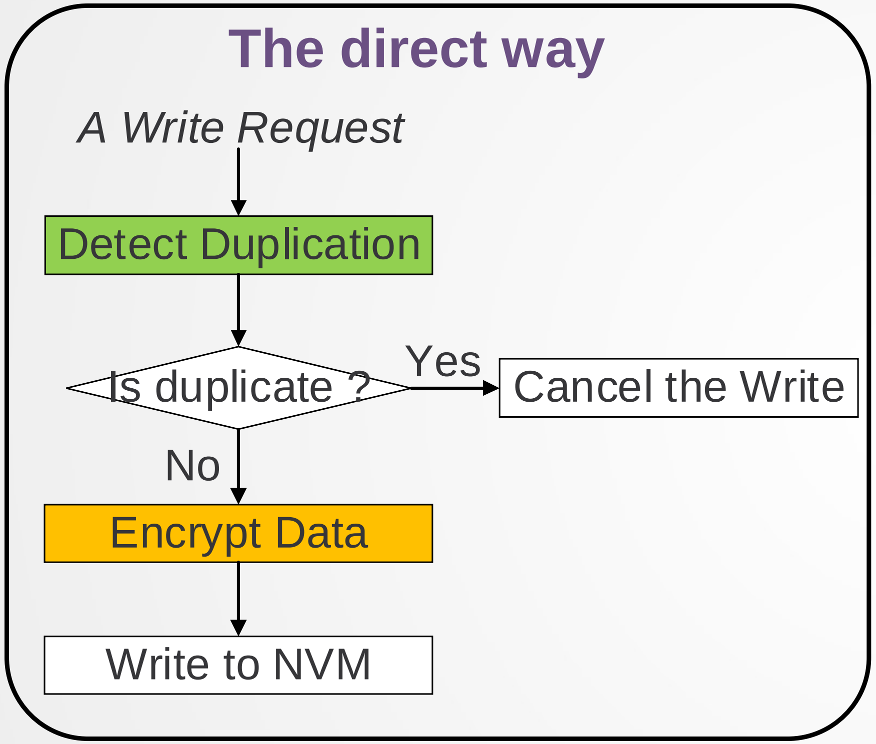
Hardware Architecture

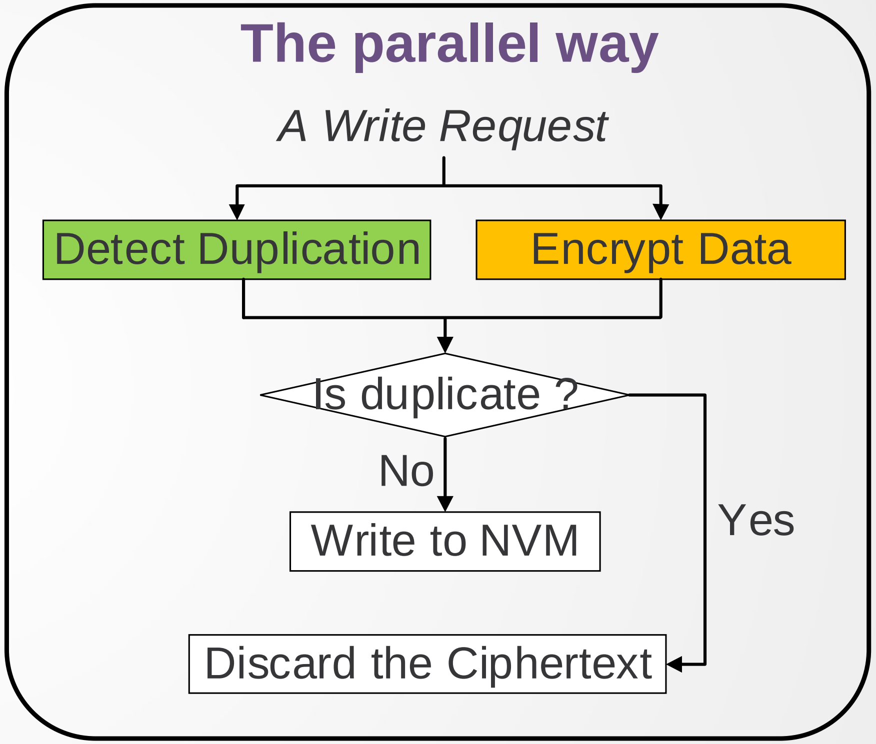
Prediction-based Parallelism



- Be inefficient for non-duplicate writes
    - Serial execution latency

Prediction-based Parallelism

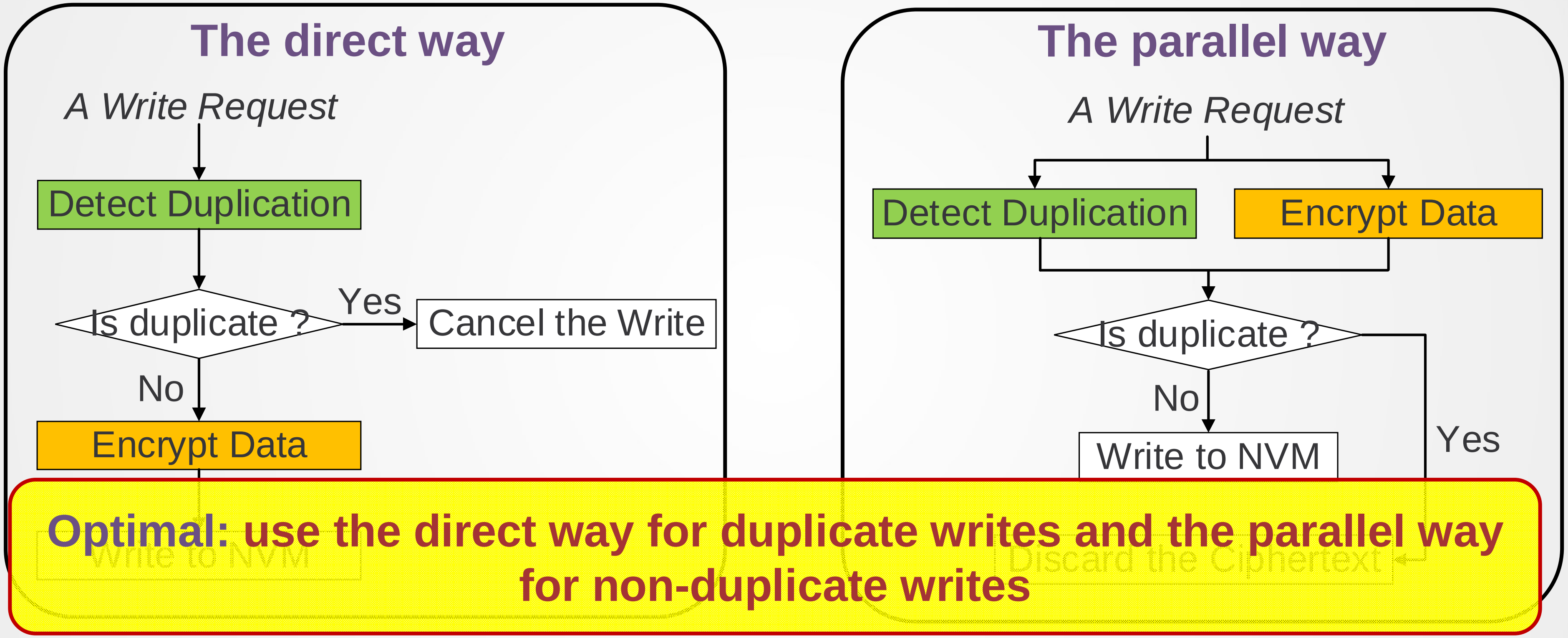




- Be inefficient for non-duplicate writes
  - Serial execution latency

- Be inefficient for duplicate writes
  - Unnecessary encryption

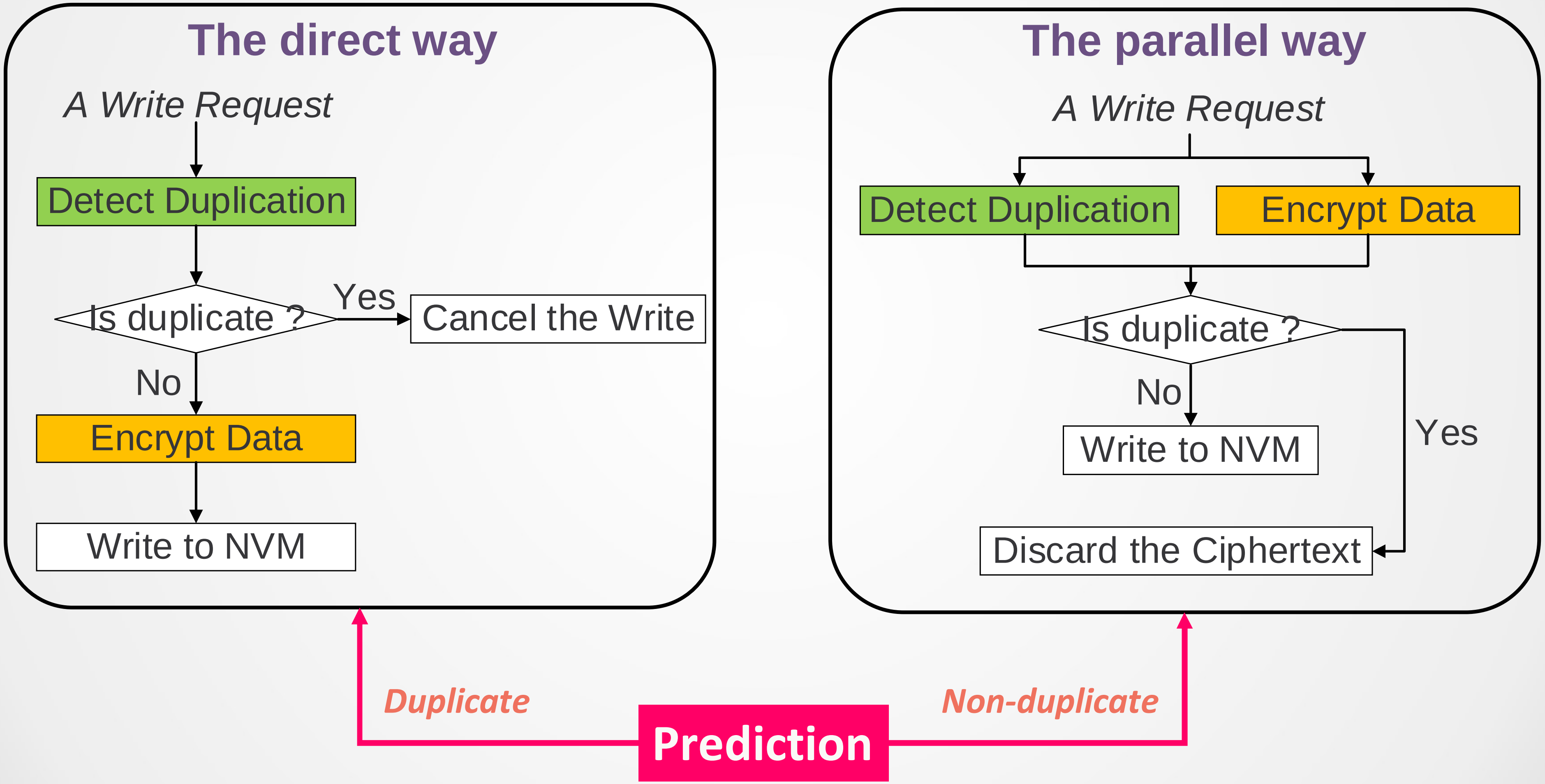
Prediction-based Parallelism



Optimal: use the direct way for duplicate writes and the parallel way for non-duplicate writes

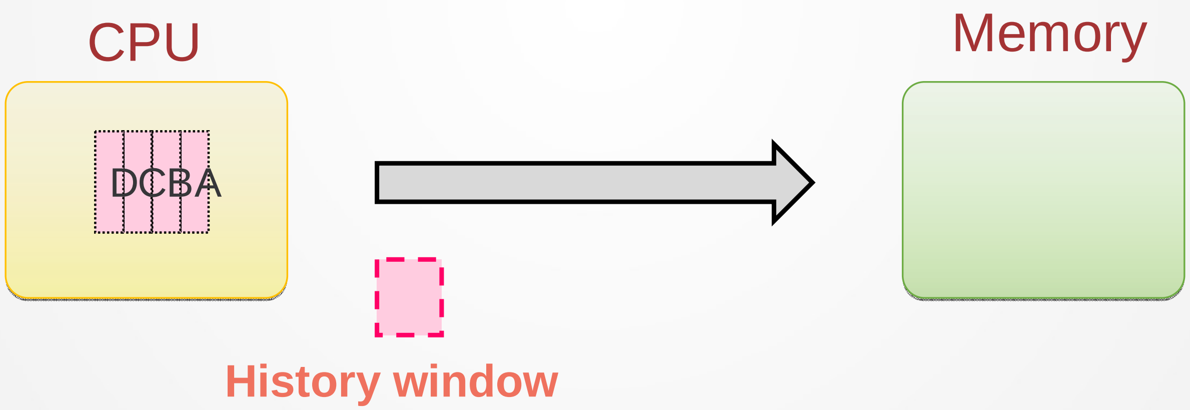
- Be inefficient for non-duplicate writes
  - Serial execution latency
- Be inefficient for duplicate writes
  - Unnecessary encryption

Prediction-based Parallelism



Prediction-based Parallelism

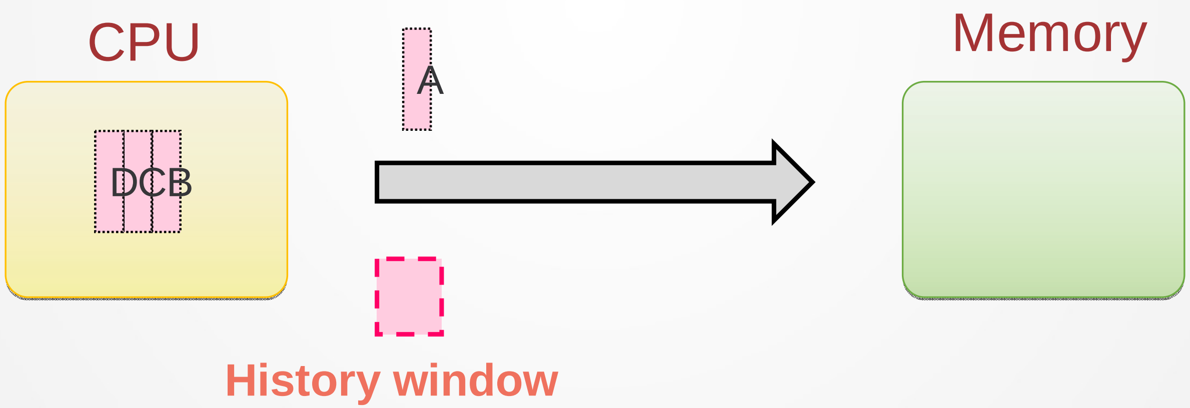
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate 0: non-duplicate

Prediction-based Parallelism

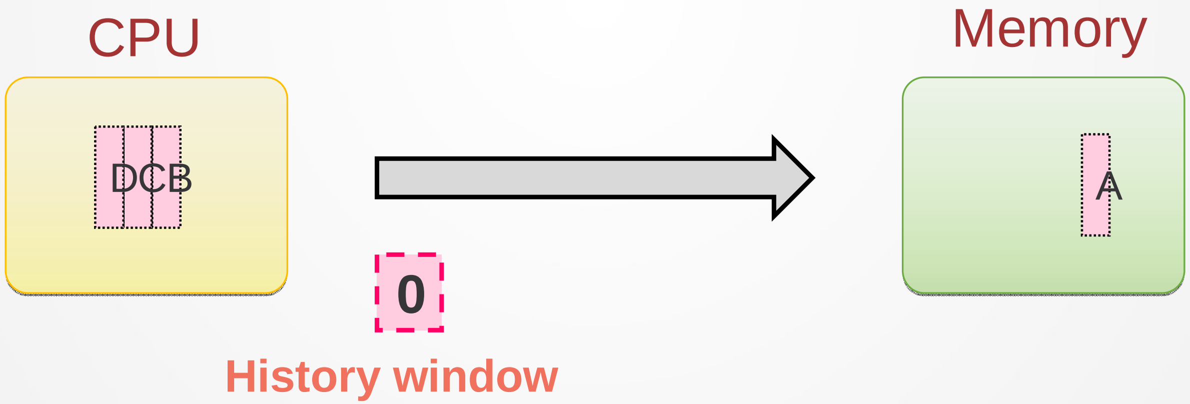
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate   0: non-duplicate

Prediction-based Parallelism

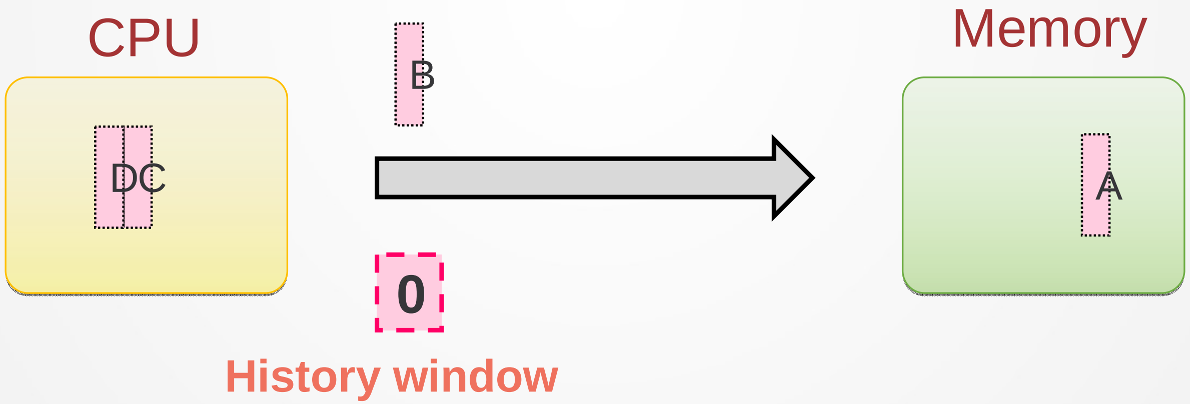
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate   0: non-duplicate

Prediction-based Parallelism

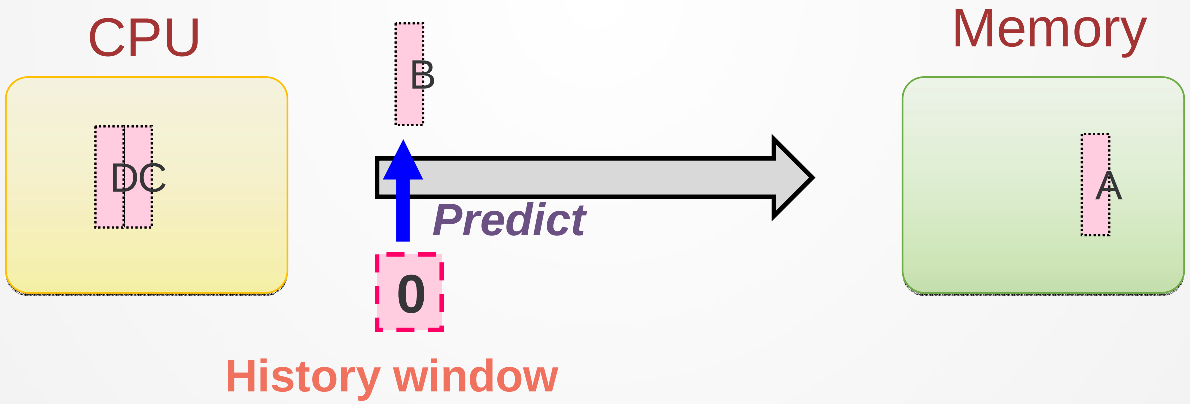
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate 0: non-duplicate

Prediction-based Parallelism

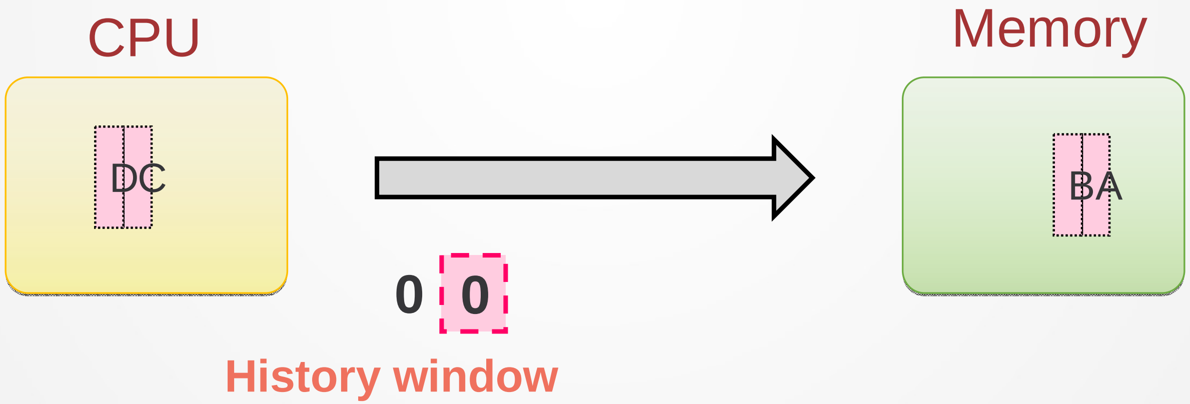
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate 0: non-duplicate

Prediction-based Parallelism

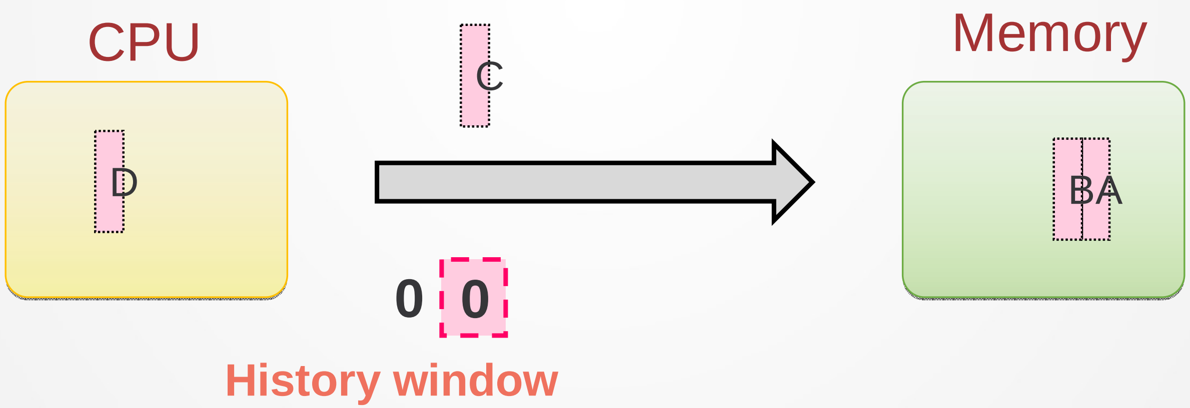
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate 0: non-duplicate

Prediction-based Parallelism

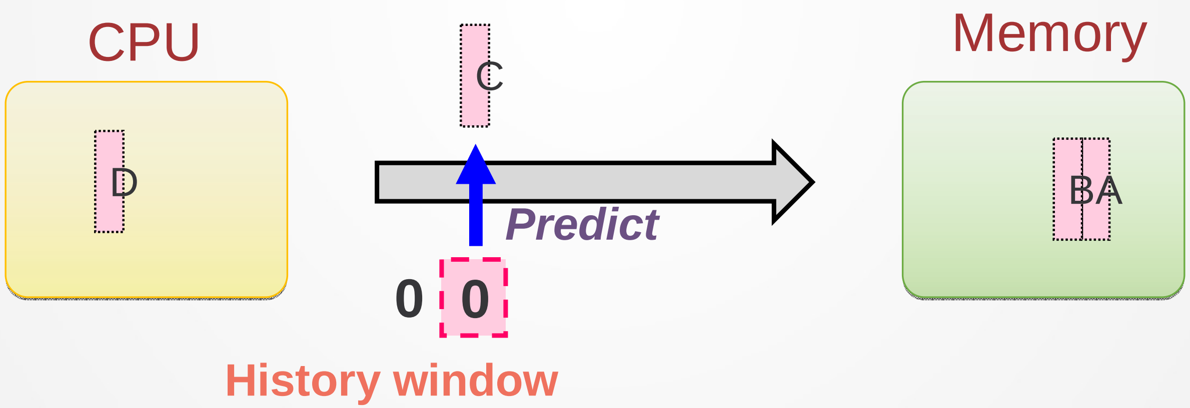
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate 0: non-duplicate

Prediction-based Parallelism

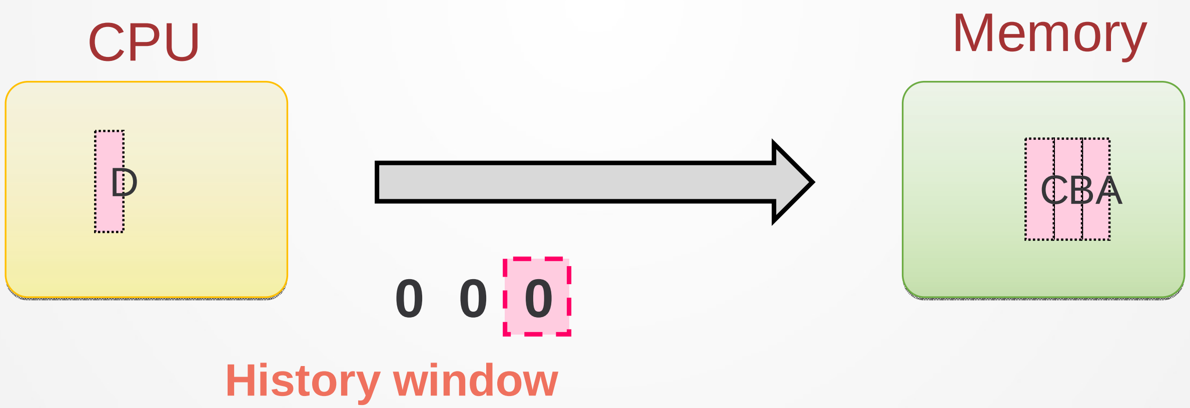
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate   0: non-duplicate

Prediction-based Parallelism

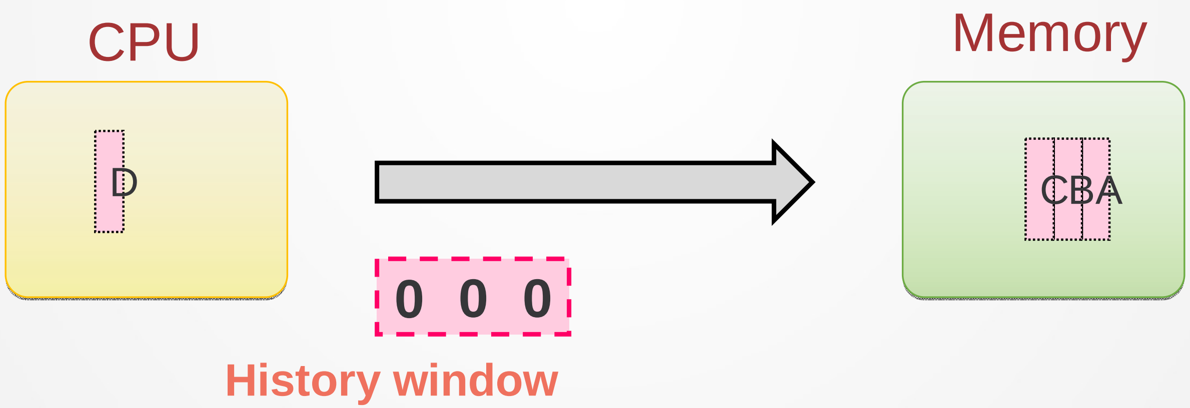
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:** **92.1% accuracy**



1: duplicate   0: non-duplicate

Prediction-based Parallelism

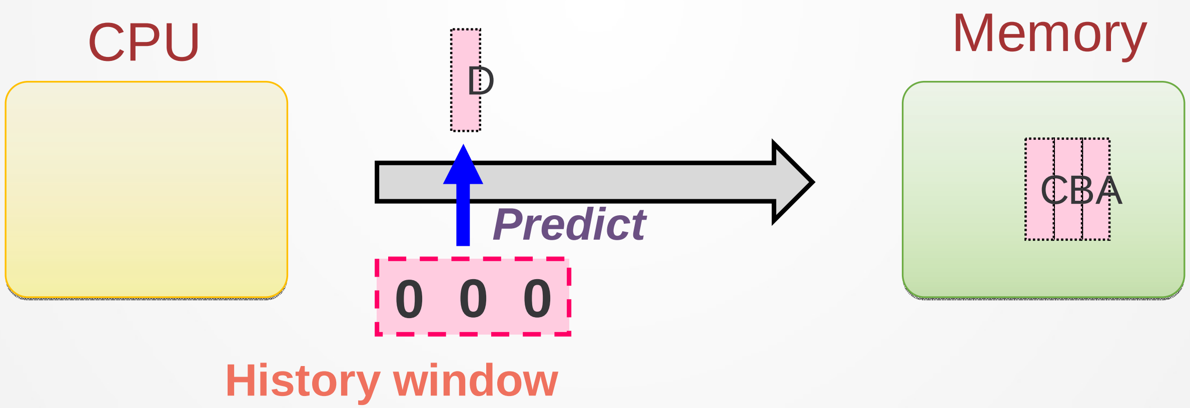
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate 0: non-duplicate

Prediction-based Parallelism

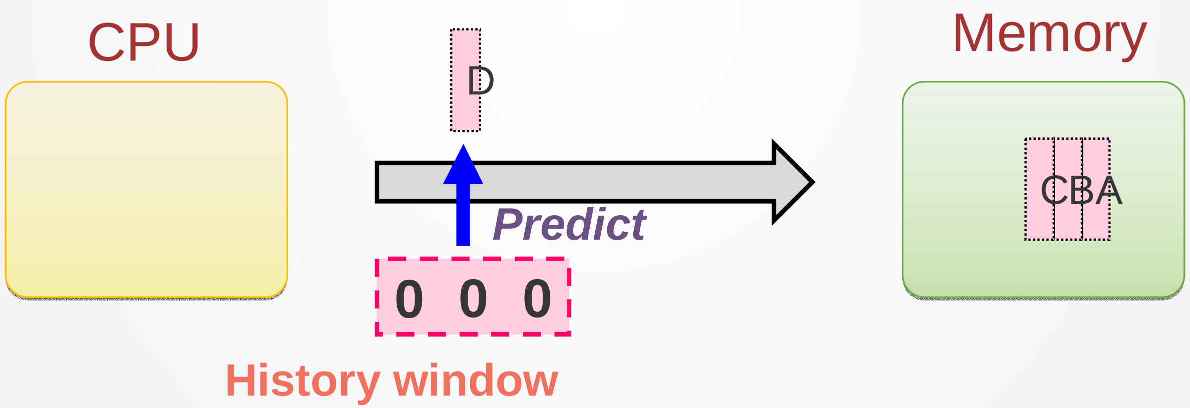
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:**



1: duplicate 0: non-duplicate

Prediction-based Parallelism

- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:** **92.1%** → **93.6%**

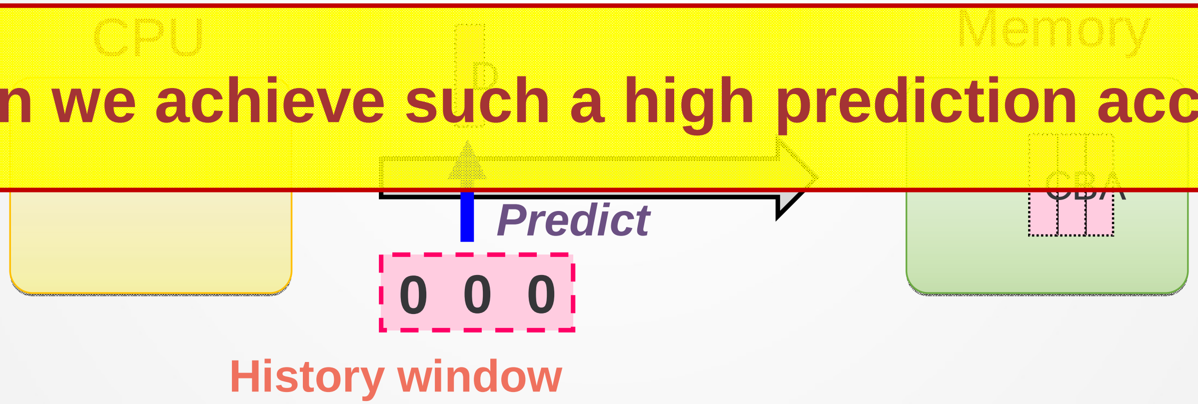


1: duplicate 0: non-duplicate

Prediction-based Parallelism

- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:** **92.1% → 93.6%**

Why can we achieve such a high prediction accuracy?



1: duplicate 0: non-duplicate

Prediction-based Parallelism

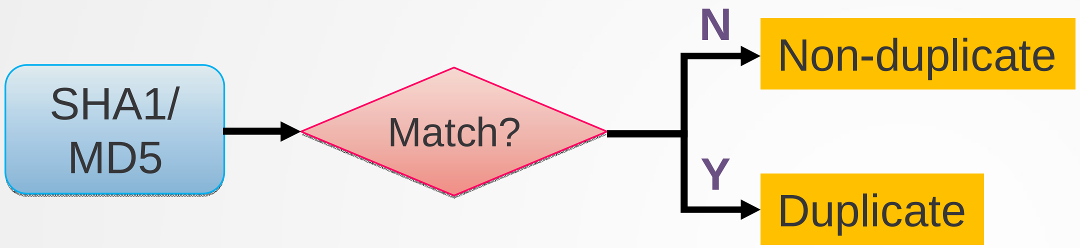
- How to know whether a cache line is duplicate beforehand?
- **Observation:** duplication states of most memory writes are the same as those of their previous ones
- **A prediction scheme:** **92.1% → 93.6%**

Why can we achieve such a high prediction accuracy?

- **Rationale:** the size of duplicate (non-duplicate) data is usually much larger than a cache line
  - E.g., a page (4KB) is duplicate or non-duplicate: 100% accuracy

Lightweight Deduplication for NVMM

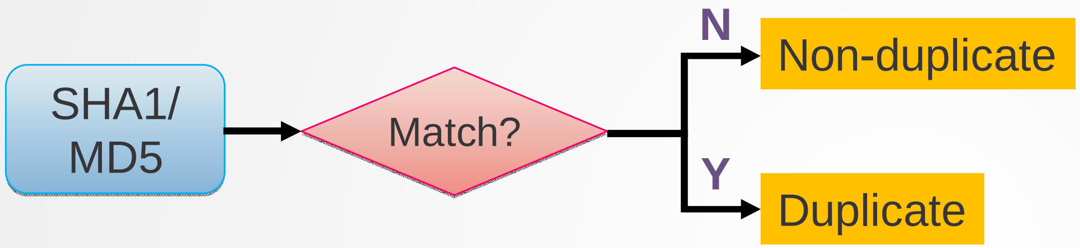
- Traditional deduplication



Hash computation latency: >300ns
≈ NVM write latency

Lightweight Deduplication for NVMM

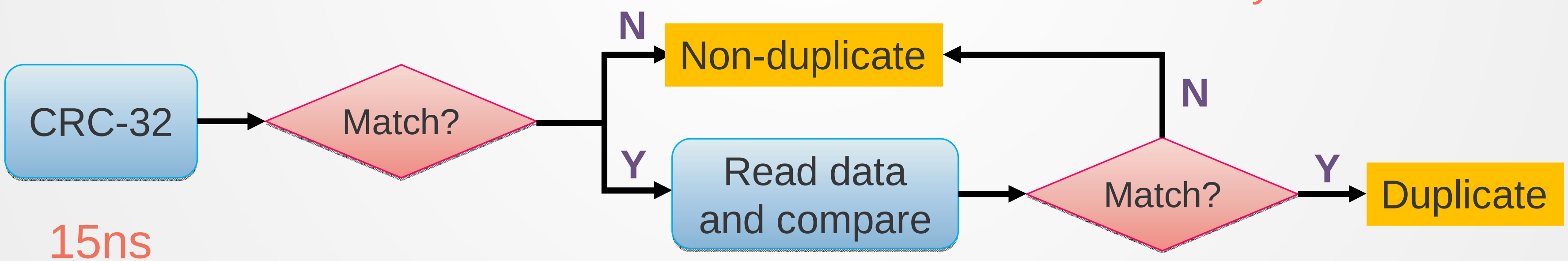
- Traditional deduplication



Hash computation latency: >300ns
≈ NVM write latency

- DeWrite

The latency is 91ns at most

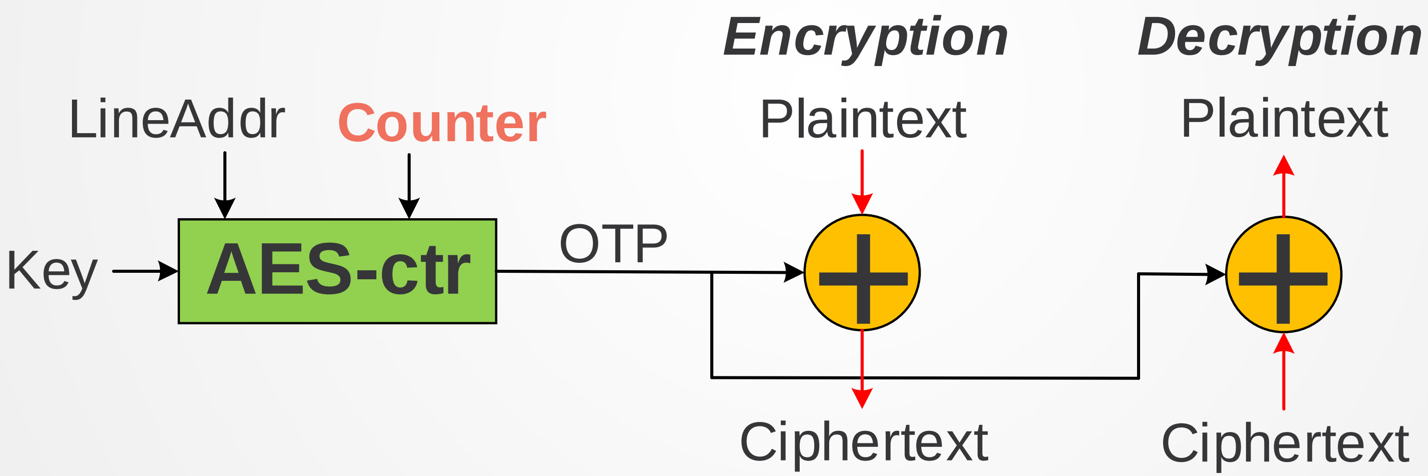


15ns

75ns+1ns

Metadata Colocation

- Encryption metadata: per-line counter



Metadata Colocation

- Encryption metadata: per-line counter
- Deduplication metadata: address mapping, reverted hash

Metadata Colocation

- Encryption metadata: per-line counter
- Deduplication metadata: address mapping, reverted hash

Deduplicated

initAddr:	0	1	2	3	4	5		n
hash:	h_0	h_1	-	h_3	-	-	…	h_n

(a) The inverted hash table

'-': empty

initAddr:	0	1	2	3	4	5		n
realAddr:	-	-	a_2	-	a_4	a_5	…	a_n

(b) The address mapping table

Metadata Colocation

- Encryption metadata: per-line counter
- Deduplication metadata: address mapping, reverted hash

initAddr:	0	1	2	3	4	5		n
hash:	h_0	h_1	c_2	h_3	c_4	c_5	…	h_n

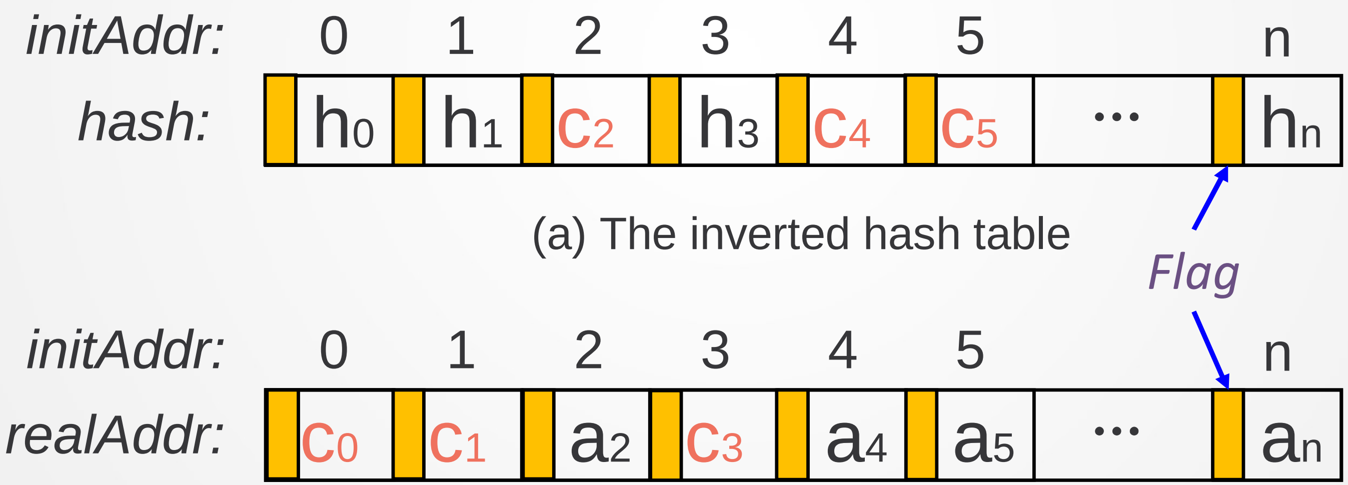
(a) The inverted hash table

initAddr:	0	1	2	3	4	5		n
realAddr:	c_0	c_1	a_2	c_3	a_4	a_5	…	a_n

(b) The address mapping table

Metadata Colocation

- Encryption metadata: per-line counter
- Deduplication metadata: address mapping, reverted hash



(a) The inverted hash table

(b) The address mapping table

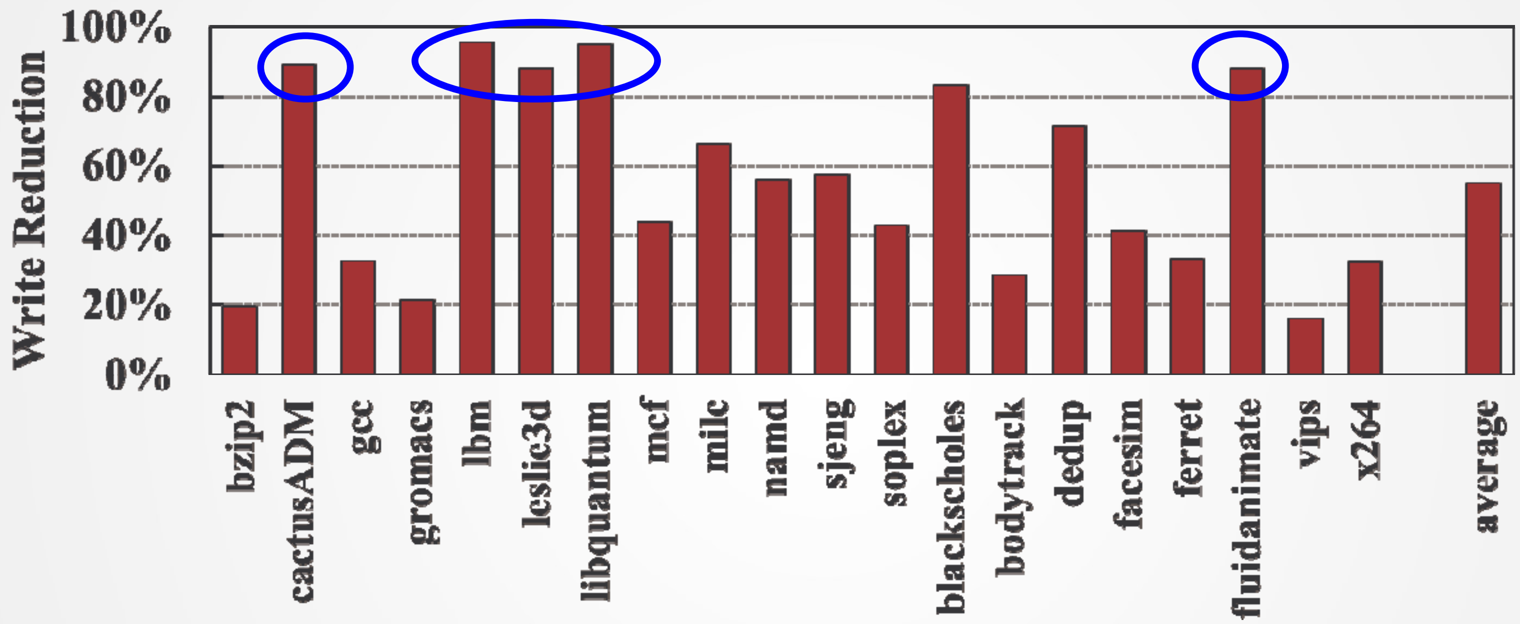
Evaluation

- Simulation: gem5 + NVMain

Processor	
CPU	4 cores, X86-64 processor, 2GHz
Private L1 cache	32KB, 8-way, LRU, 2-CPU-cycle latency
Private L2 cache	128KB, 8-way, LRU, 8-CPU-cycle latency
Shared L3 cache	2MB, 8-way, LRU, 25-CPU-cycle latency
Shared L4 cache	32MB, 8-way, LRU, 50-CPU-cycle latency
Main Memory Using PCM	
Capacity	16GB, (16 banks, distributed in 2 ranks)
Read/write latency	75ns/300ns
Metadata cache	2MB, LRU, 25-CPU-cycle latency

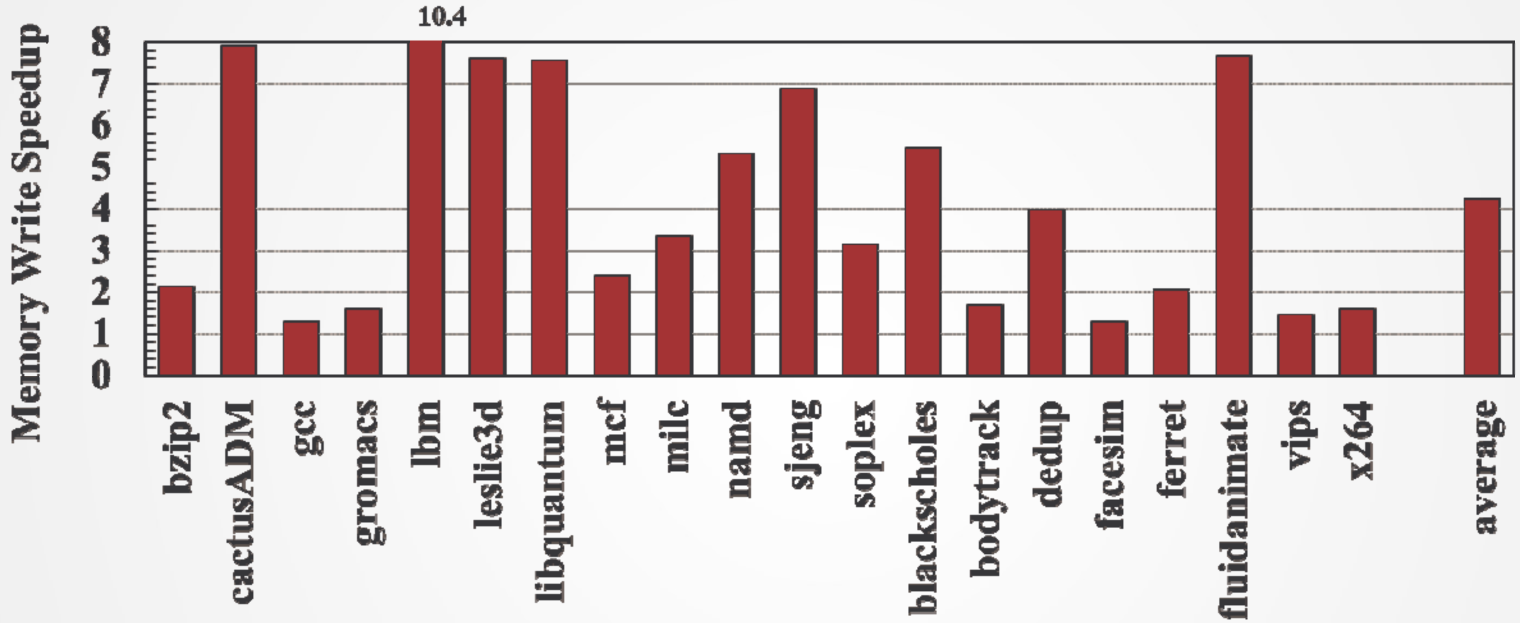
- Benchmarks
  - 12 Benchmarks from SPEC CPU2006: single-threaded
  - 8 benchmarks from m PARSEC 2.1: multiple-threaded

NVM Endurance



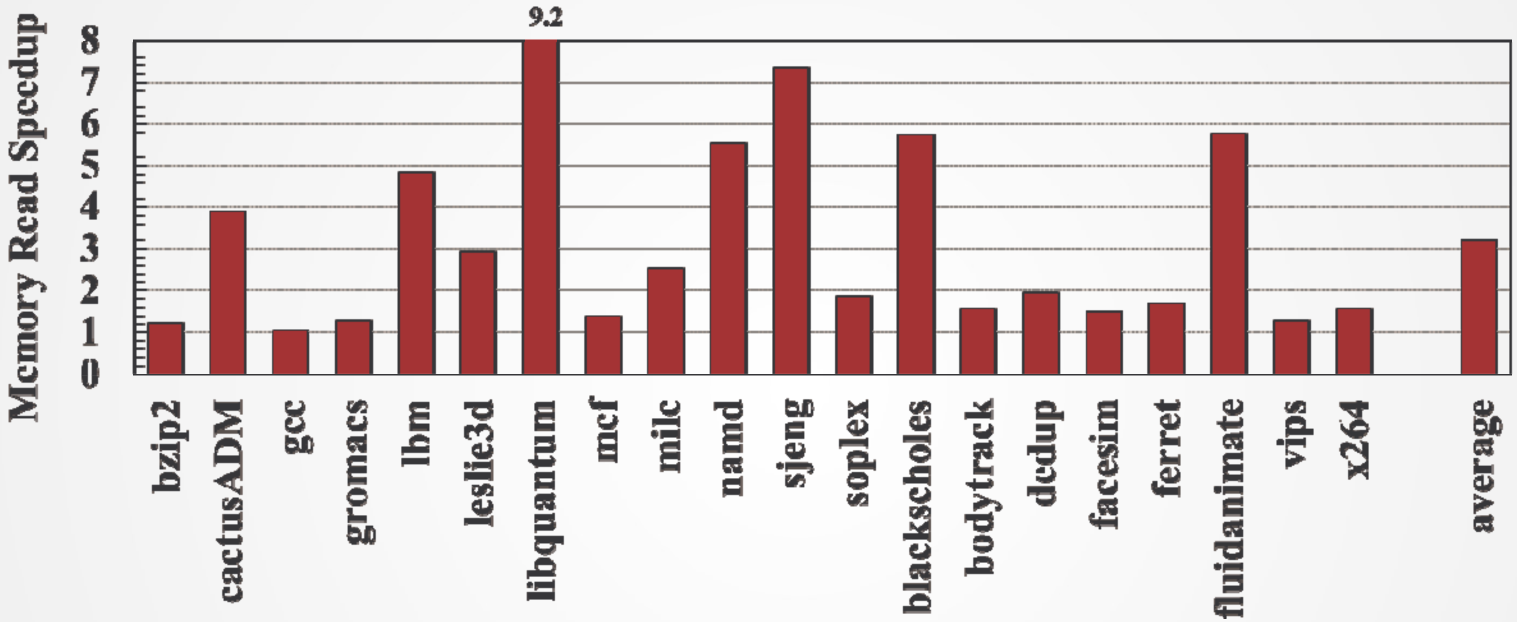
➢ DeWrite reduces 54% writes to secure NVM on average

Write Speedup



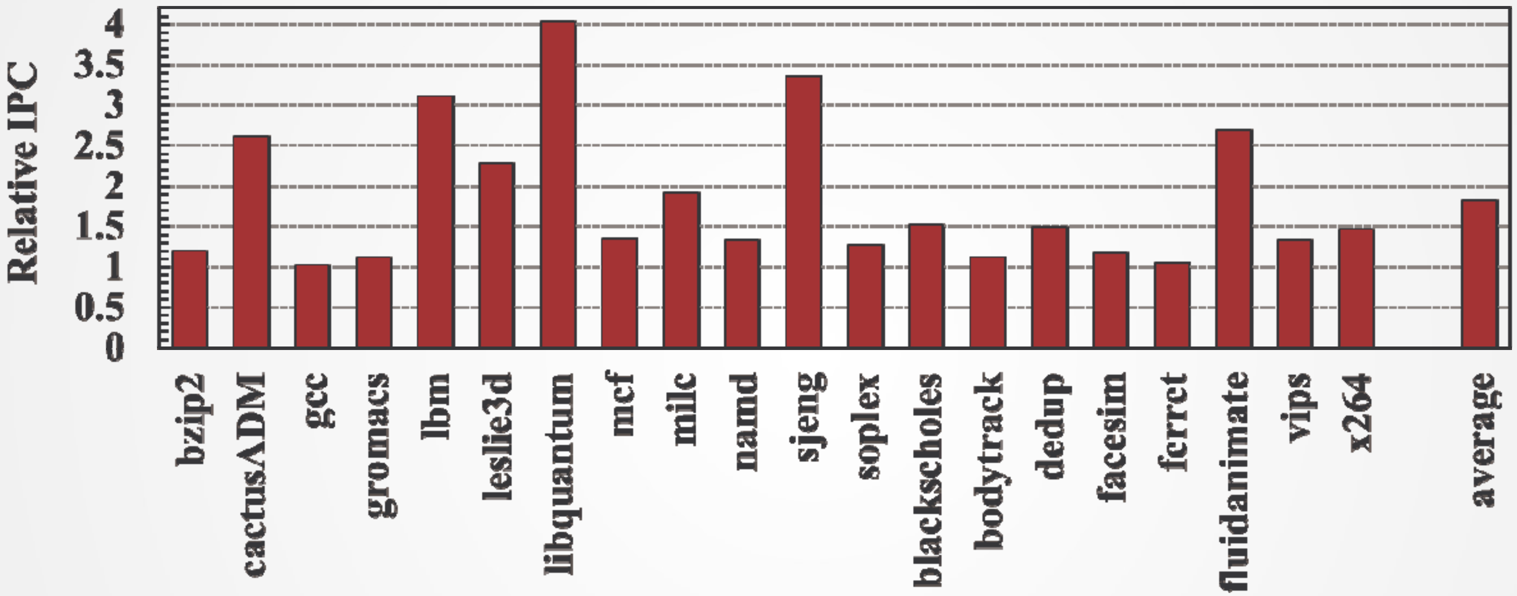
- DeWrite speeds up NVM writes by 4.2X on average

Read Speedup



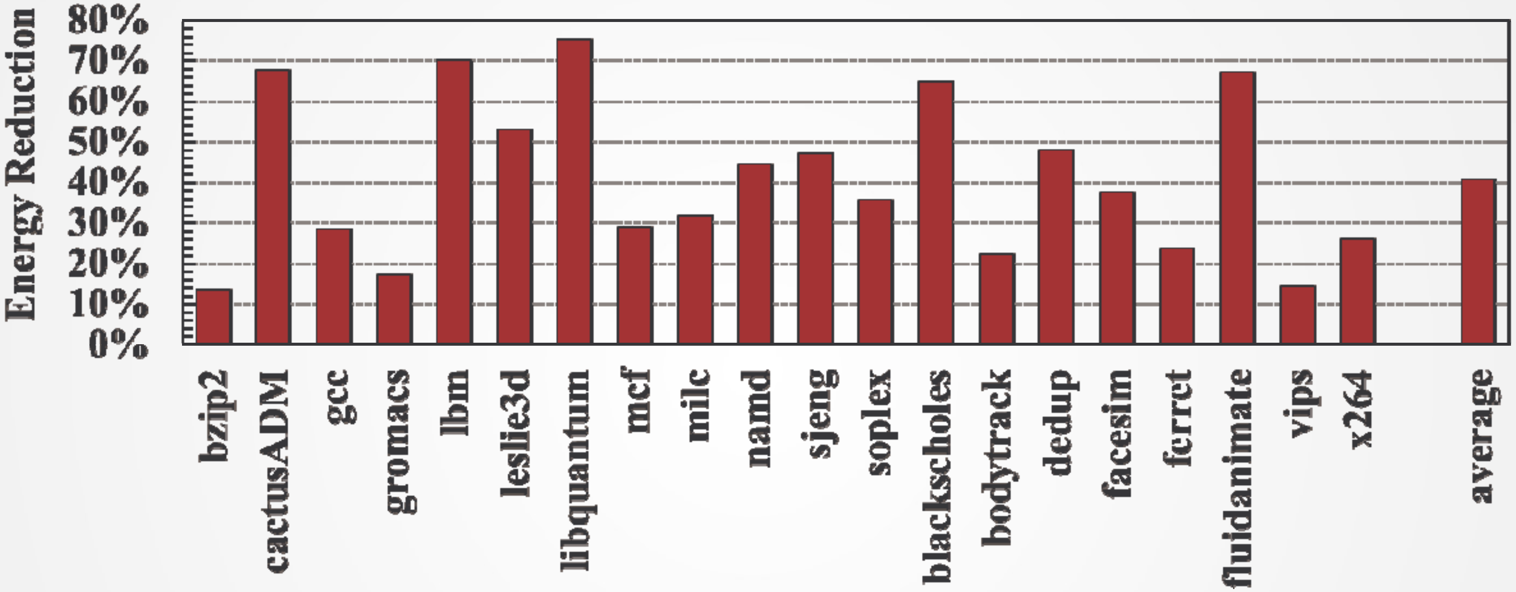
- DeWrite speeds up NVM reads by 3.1X on average

Instructions per Cycle



- DeWrite improves the IPC by 80% on average

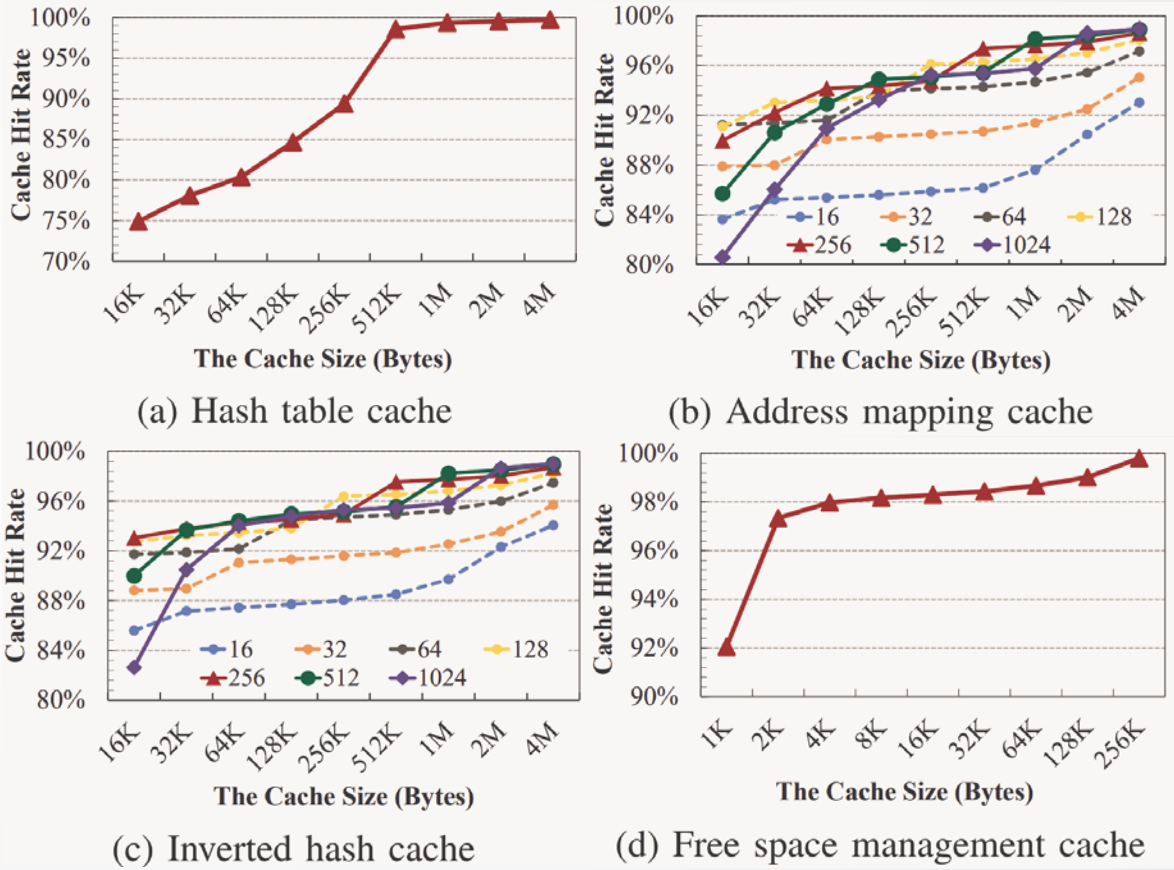
Energy Consumption



- DeWrite reduces energy consumption by 40% on average

Space Overheads of Metadata Storage & Cache

- Metadata storage
  - 6.25%
- Metadata cache
  - (a) 512KB
  - (b) 512KB
  - (c) 512KB
  - (d) 128KB
  - Total <2MB



Conclusion

- Memory encryption renders the bit-level write reduction techniques ineffective for secure NVMM
- This paper proposes **DeWrite**, a line-level write reduction technique to enhance the endurance and performance
  - Lightweight cahe-line-level deduplication
  - Efficient synergization of deduplication and encryption
- Reduce 54% writes, speed up memory writes and reads of secure NVMM by 4.2× and 3.1×, on average

Thanks! Q&A