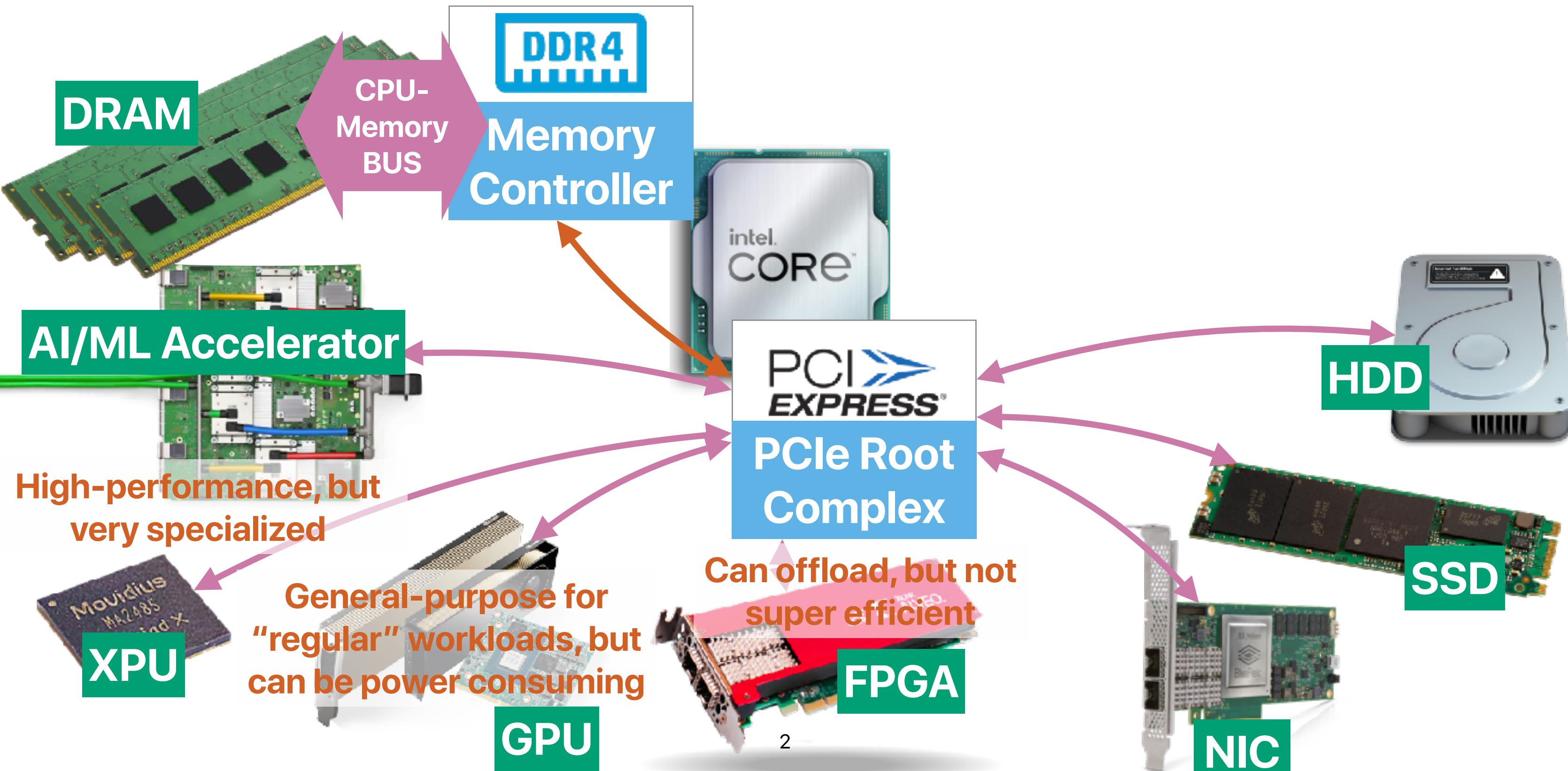


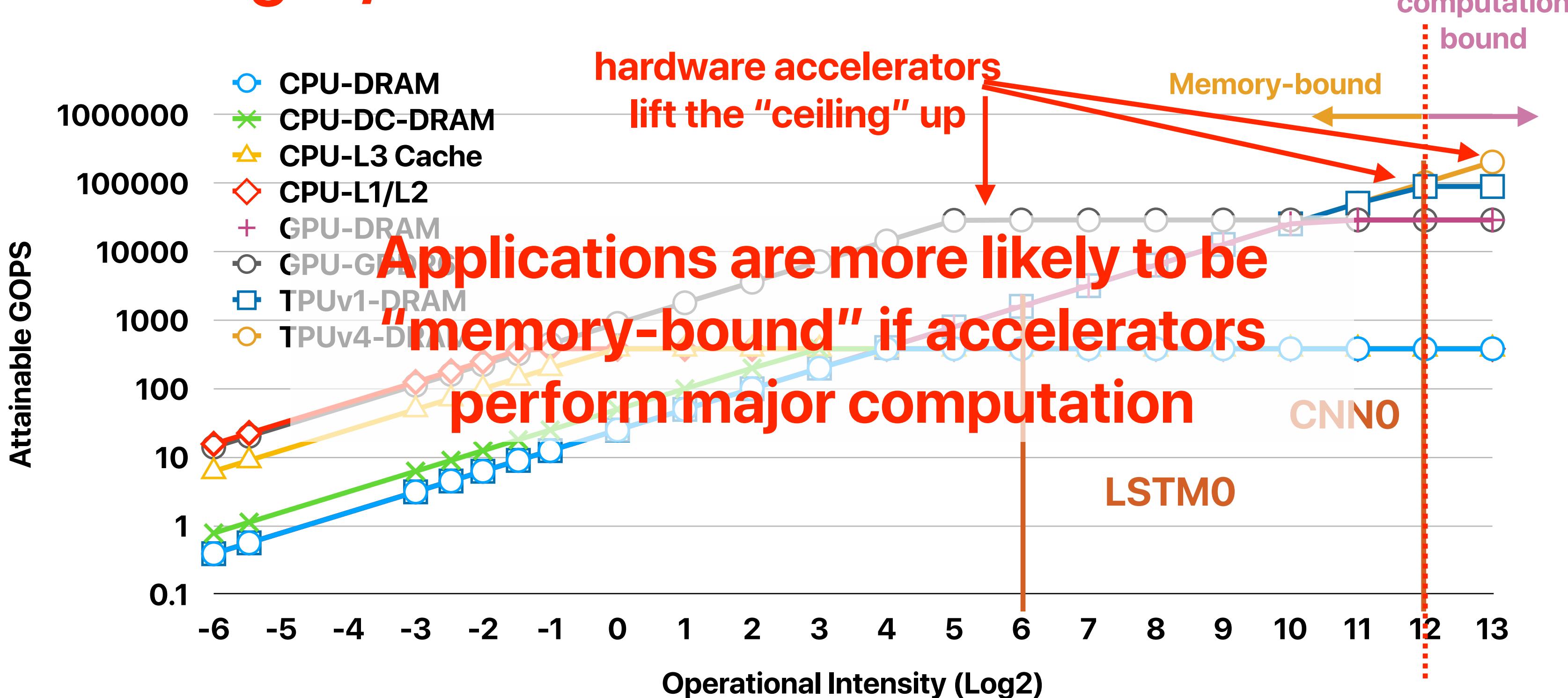
# In/Near Memory Processing (1)

Hung-Wei Tseng

# Recap: The landscape of modern computers

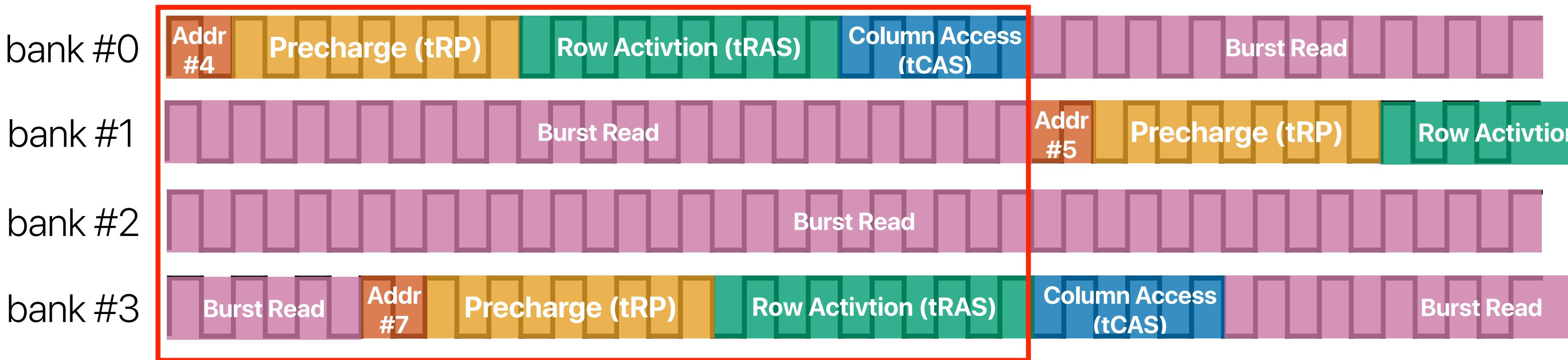


# Placing AI/ML Accelerators in the roofline model



# Recap: Multi-bank access

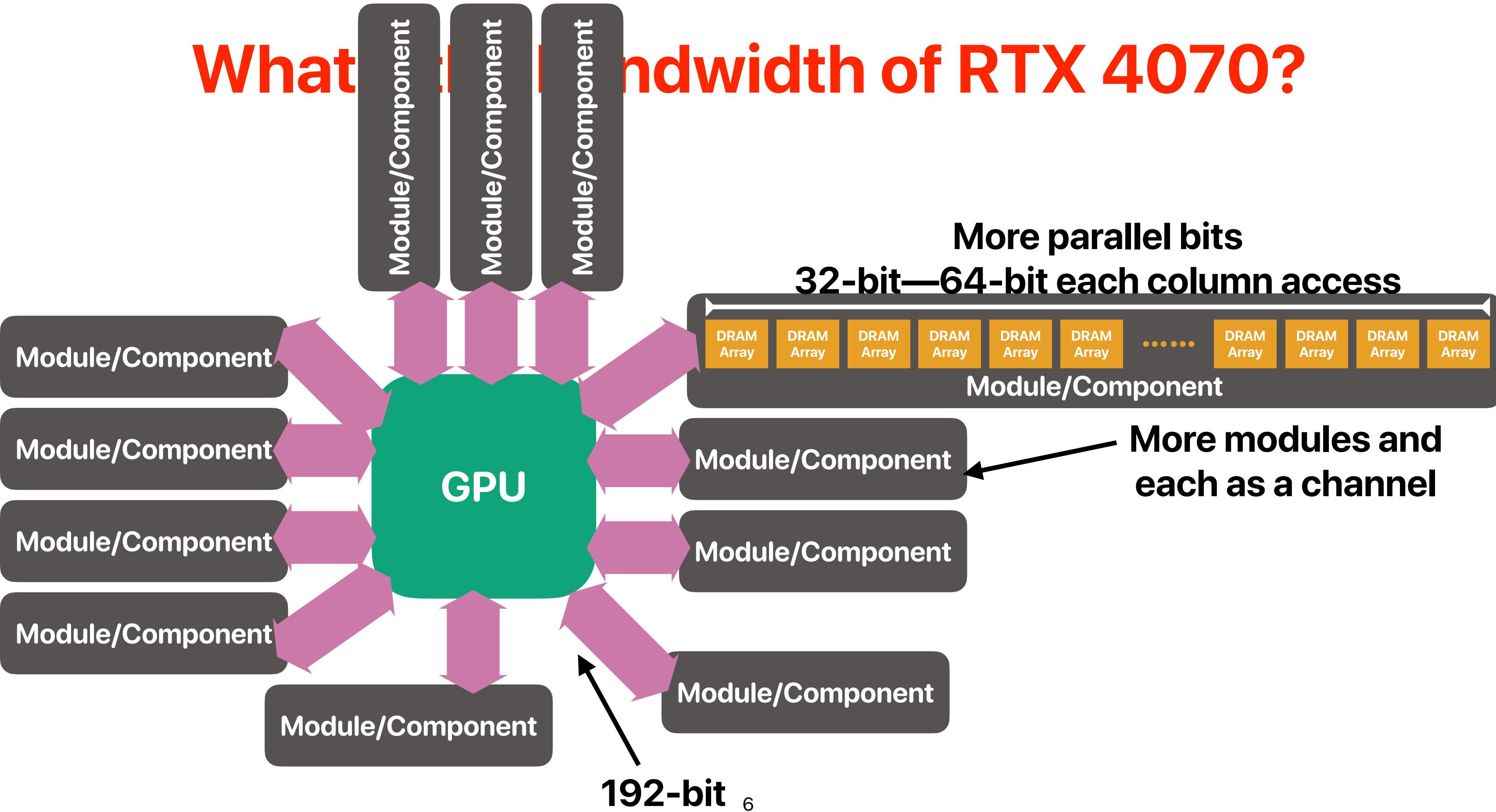
The latency of pre-charge, row/column accesses is fully covered!



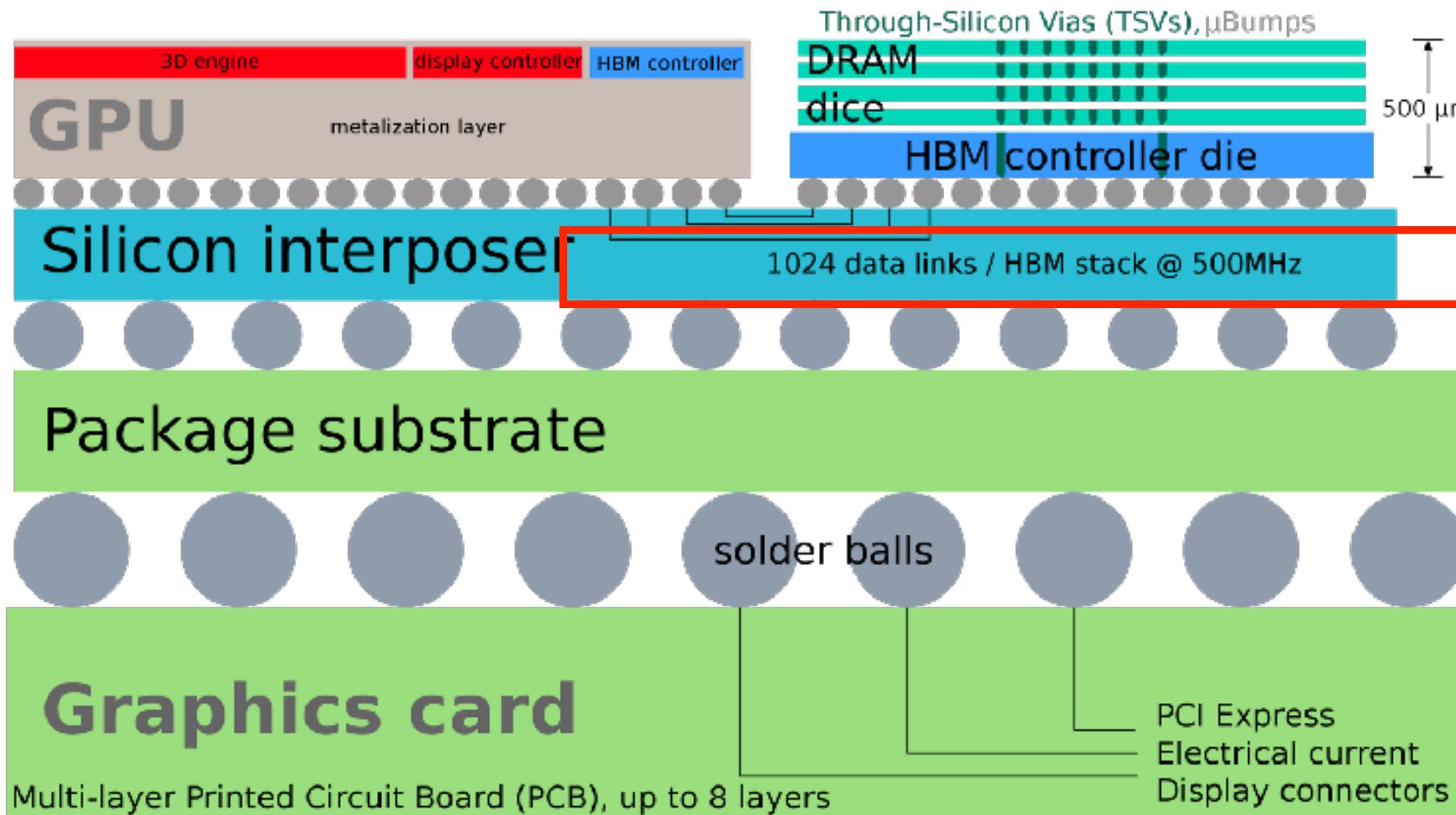
# Recap: Improving memory-bounded applications

- Faster memory technologies
- Higher memory bandwidth
- Lower data volume

# What is the bandwidth of RTX 4070?



# HBM (High Bandwidth Memory)



$$0.5G \times \frac{1024\text{ bits}}{8\text{ bits}} = 64\text{ GB/sec}$$

If you have 4x modules —  
 $4 * 64 = 256 \text{ GB/sec}$

HBM2 increase the clock  
rate to 2GHz — 1TB/sec

# Outline

- Performance of main memory subsystem (cont.)
- Near/In-memory processing

# The program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <time.h> /* for clock_gettime */
#include <malloc.h>

void write_memory_loop(void* array, size_t size) {
    size_t* carray = (size_t*) array;
    size_t i;
    for (i = 0; i < size / sizeof(size_t); i++) {
        carray[i] = 1;
    }
}

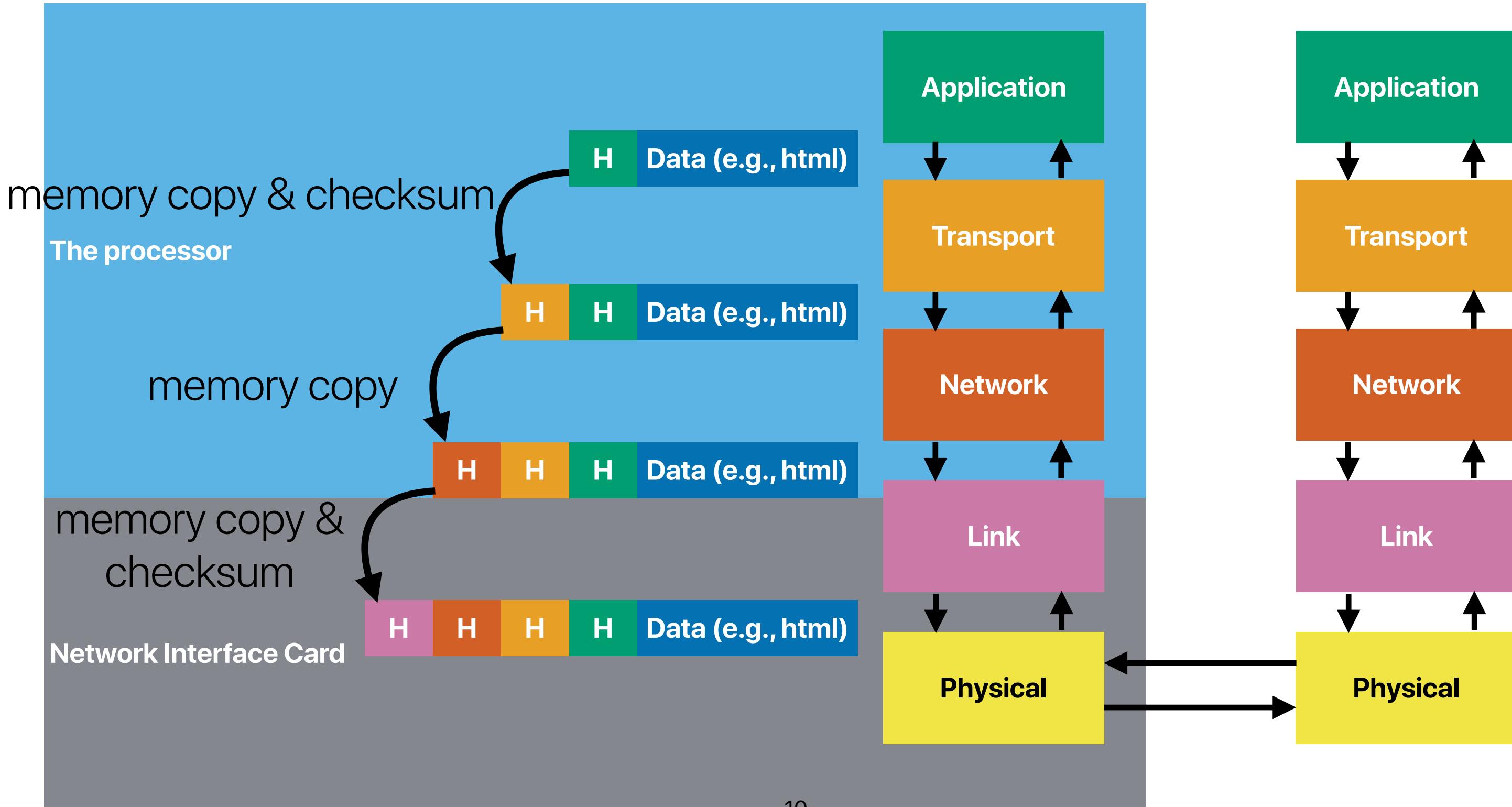
int main(int argc, char **argv)
{
    size_t *array;
    size_t *dest;
    size_t size;
    double total_time;
    struct timespec start, end;
    struct timeval time_start, time_end;
    size = atoi(argv[1])/sizeof(size_t);
    array = (size_t *)malloc(sizeof(size_t)*size);
```

```
    dest = (size_t *)malloc(sizeof(size_t)*size);
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start
time */
    #ifdef SIMD
        write_memory_avx(array, size);
    #else
        write_memory_loop(array, size);
    #endif
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start
time */
    memcpy(dest, array, size*sizeof(size_t));
    clock_gettime(CLOCK_MONOTONIC, &end); /* mark start
time */
    total_time = ((end.tv_sec * 1000000000.0 +
end.tv_nsec) - (start.tv_sec * 1000000000.0 +
start.tv_nsec));
    fprintf(stderr,"Latency: %.0lf ns, GBps: %lf,
%llu\n",total_time, (double)((double)size*sizeof(size_t)/
(total_time)), dest[rand()%size]);
    return 0;
}
```

## memmove & memcpy: 5% cycles in Google's datacenter

Svilen Kanev, Juan Pablo Darago, Kim Hazelwood,  
Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and  
David Brooks. ISCA '15

# Network protocol stack



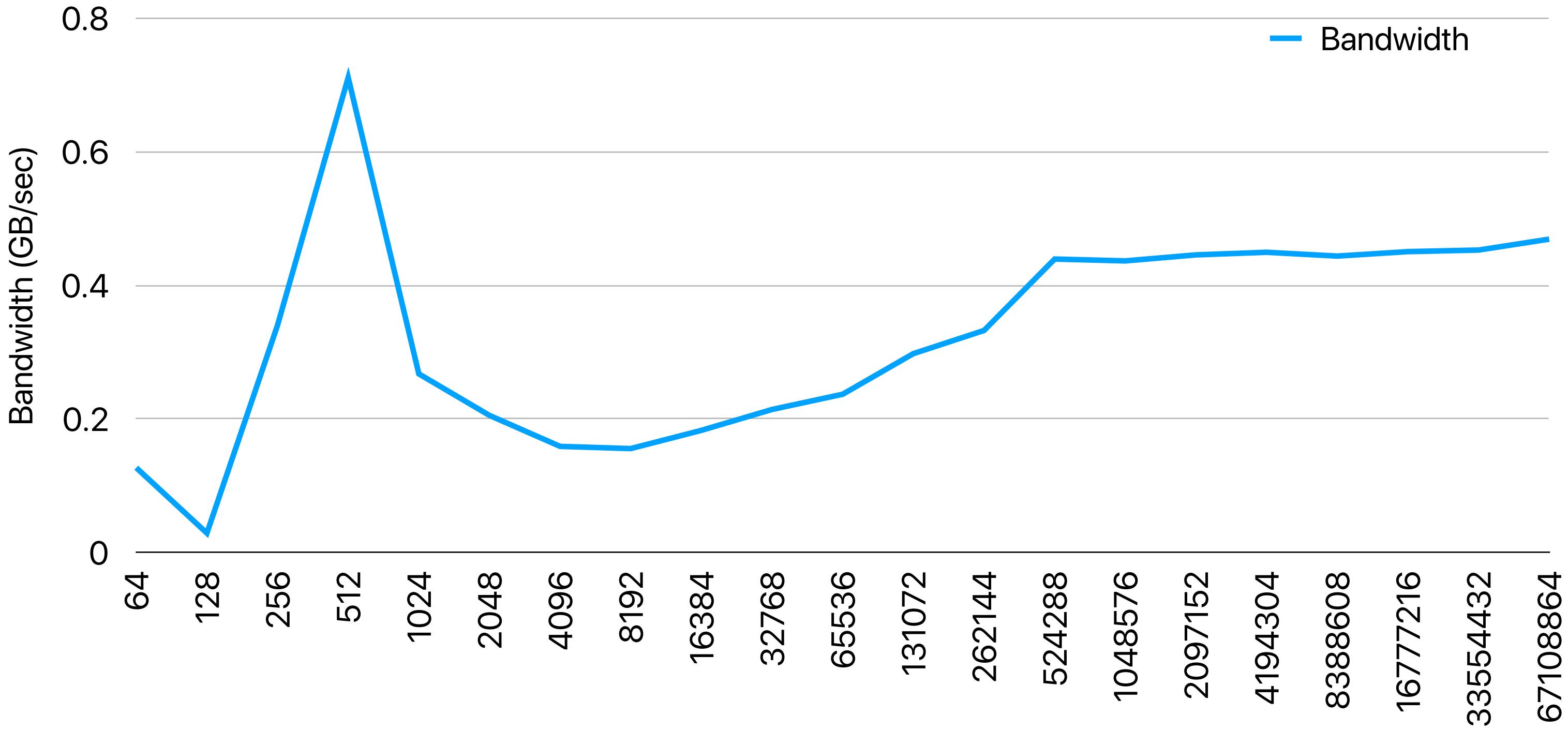
# Use cases of memcpy

- Network packets (each layer needs to adjust the header and payload locations)
- Copy-on-write (spawning new process)



Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://PollEv.com/app)

# Memory copy bandwidth



# Why is memcpy so slow?

# Why is memcpy slow?



What's the x86 code for memcpy

# memcpy in x86 by Bard

Regenerate draft

- Here's a breakdown of the x86 code for memcpy, incorporating insights from various sources and addressing potential concerns:

## Key Considerations:

- Architecture and Optimization: The exact code can vary depending on the specific x86 architecture (32-bit vs. 64-bit) and compiler optimizations.
- Overlapping Memory: `memcpy` assumes non-overlapping memory regions. For overlapping regions, use `memmove`.

## Common Implementation (32-bit x86):

### Code snippet

```
1      memcpy:  
         push ebp  
         mov  ebp, esp  
  
         push esi  
         push edi  
         push ecx  
  
         mov  edi, [ebp + 8] ; Destination address  
         mov  esi, [ebp + 12] ; Source address  
         mov  ecx, [ebp + 16] ; Number of bytes to copy  
  
         rep  movsb    ; Copy bytes using movsb instruction  
  
         pop  ecx  
         pop  edi  
         pop  esi  
  
         pop  ebp  
         ret
```

### Explanation:

#### 1. Prologue:

- Saves registers `ebp`, `esi`, `edi`, and `ecx` on the stack.

#### 2. Parameter Access:

- Loads function arguments (destination, source, and size) into registers.

#### 3. Main Loop:

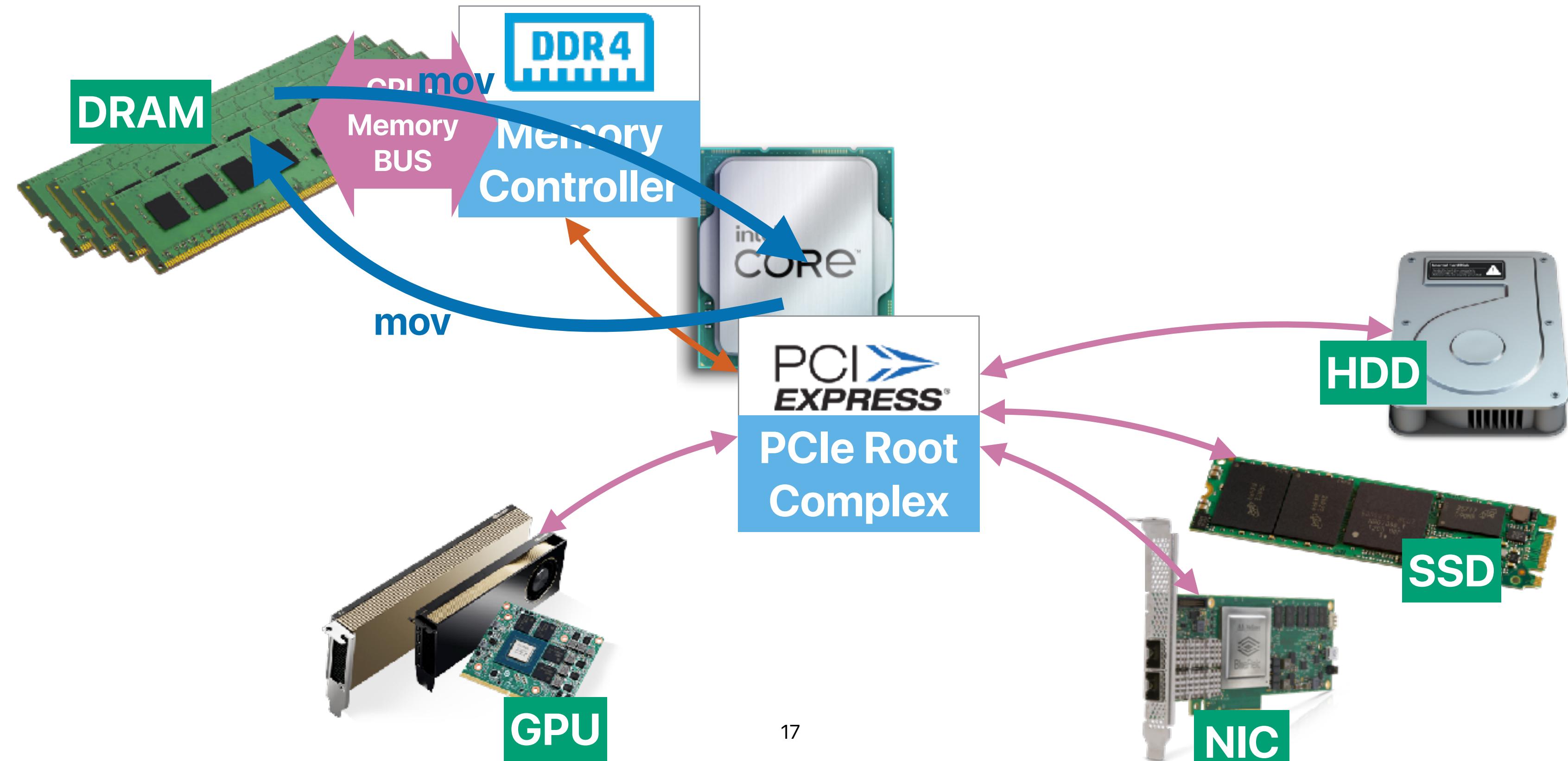
- `rep movsb`:
  - `rep`: Repeat until `ecx` becomes zero.
  - `movsb`: Move a byte from `esi` to `edi`.

#### 4. Epilogue:

- Restores saved registers.
- Returns from the function.



# Recap: The landscape of modern computers



# Why is memcpy slow?

- Each instruction can only move limited amount of data
- Lots of instructions to move data — processor can only handle limited outstanding memory requests
- They must temporarily store data in CPU registers
- Data have to flow through CPU-memory bus twice
- Other overhead
  - Page fault

How can we make memcpy  
faster?

## 11.8 4K ALIASING

4-KByte memory aliasing occurs when the code stores to one memory location and shortly after that it loads from a different memory location with a 4-KByte offset between them. For example, a load to linear address 0x400020 follows a store to linear address 0x401020.

The load and store have the same value for bits 5 - 11 of their addresses and the accessed byte offsets should have partial or complete overlap.

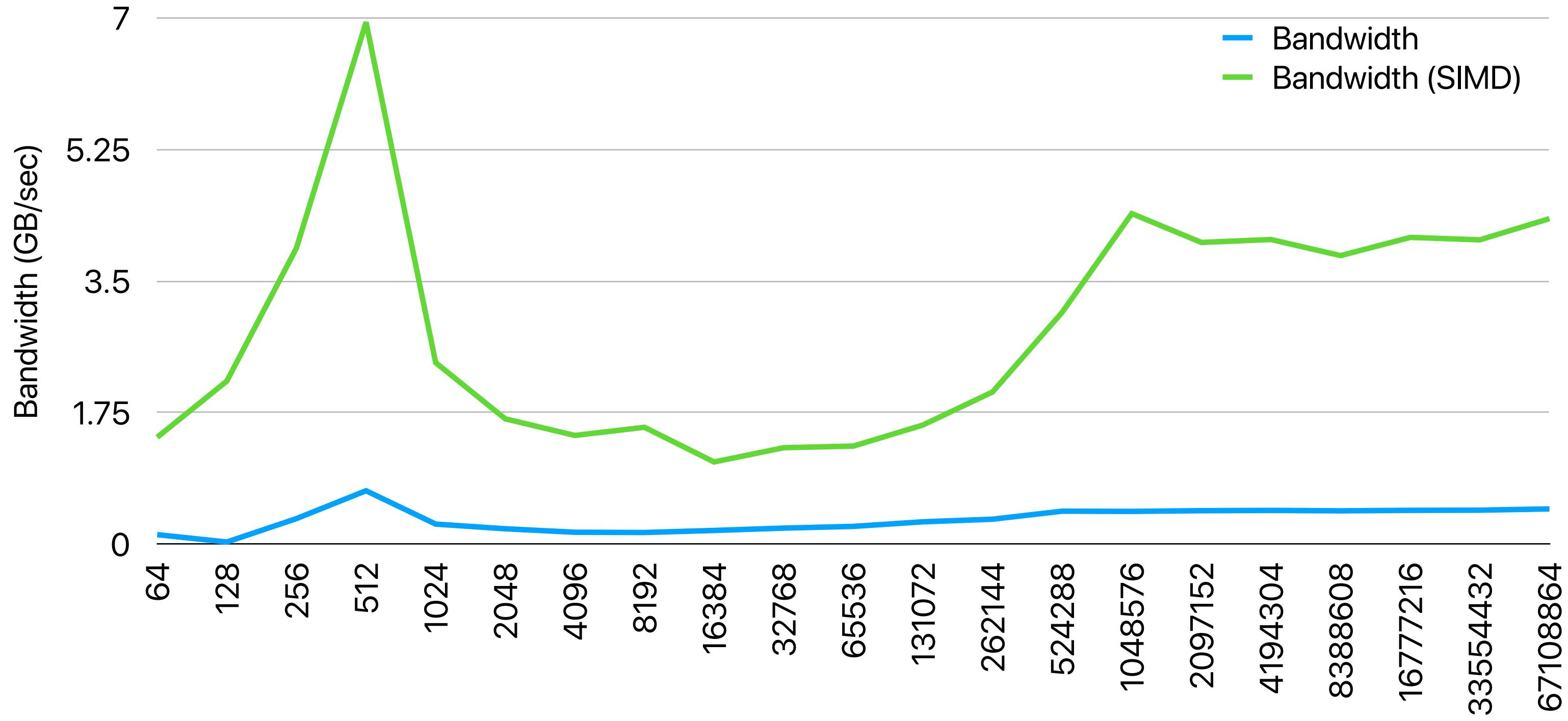
4K aliasing may have a five-cycle penalty on the load latency. This penalty may be significant when 4K aliasing happens repeatedly and the loads are on the critical path. If the load spans two cache lines it might be delayed until the conflicting store is committed to the cache. Therefore 4K aliasing that happens on repeated unaligned Intel AVX loads incurs a higher performance penalty.

To detect 4K aliasing, use the LD\_BLOCKS\_PARTIAL.ADDRESS\_ALIAS event that counts the number of times Intel AVX loads were blocked due to 4K aliasing.

To resolve 4K aliasing, try the following methods in the following order:

- Align data to 32 Bytes.
- Change offsets between input and output buffers if possible.
- Use 16-Byte memory accesses on memory which is not 32-Byte aligned.

# Memory copy bandwidth using SIMD

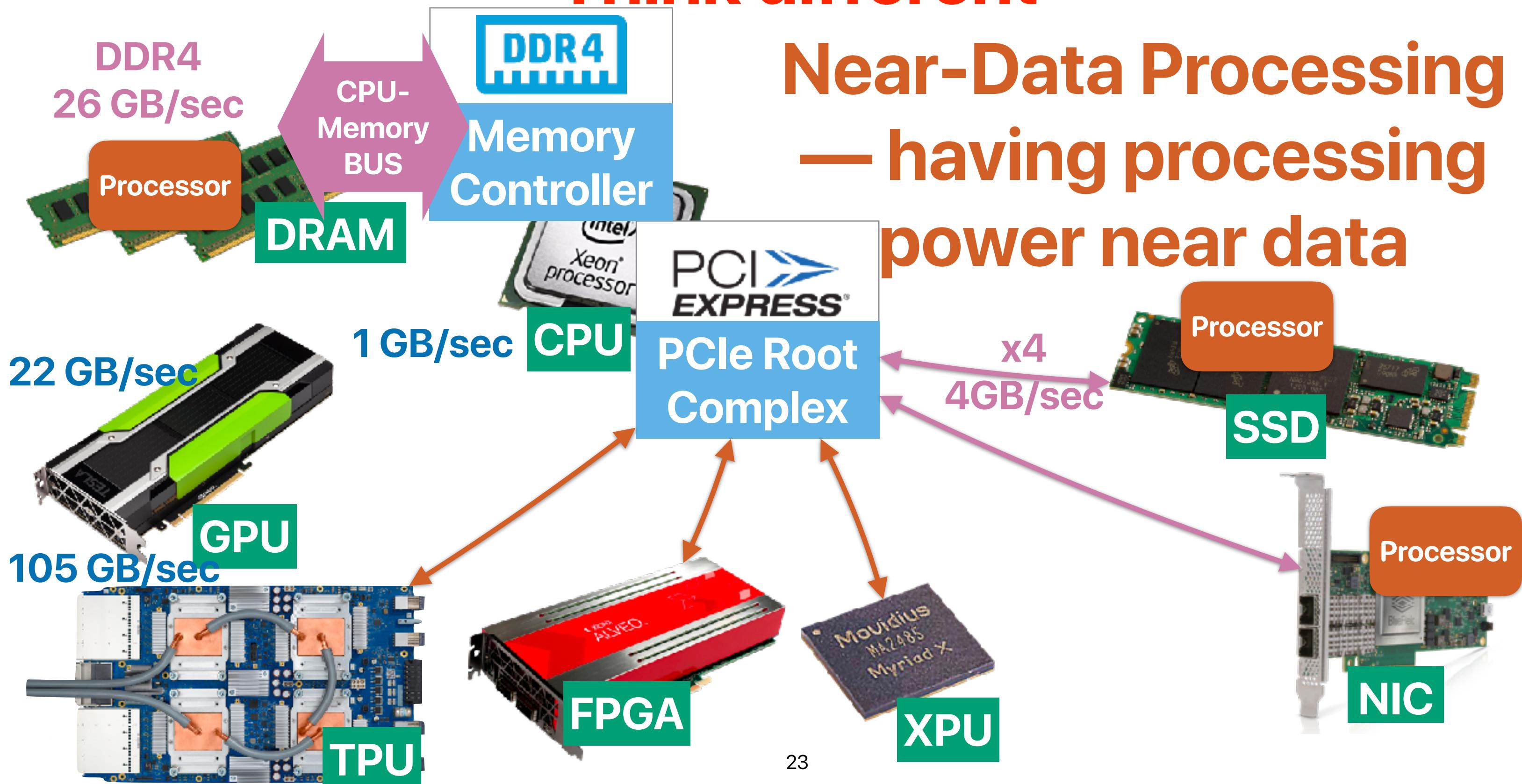


# Implications

- The cost of load a word is a lot
  - More than just a load — you need to calculate the effective address
  - That's why we want AVX to load 256-bit (32B or 4 64-bit words) to load in one instruction
- The cost of a page fault is significant
  - That's why we see the first write/read is a lot longer
  - Huge page can be helpful
- Performance optimization in software is hard!!!

# Think different

## Near-Data Processing — having processing power near data



# Near-memory processing

# Active Pages

The chip

upper bits of address

Row Decoder

0

1

2

$n-1$

Sense Amplifiers

lower bits of address

Row Buffer

Operation

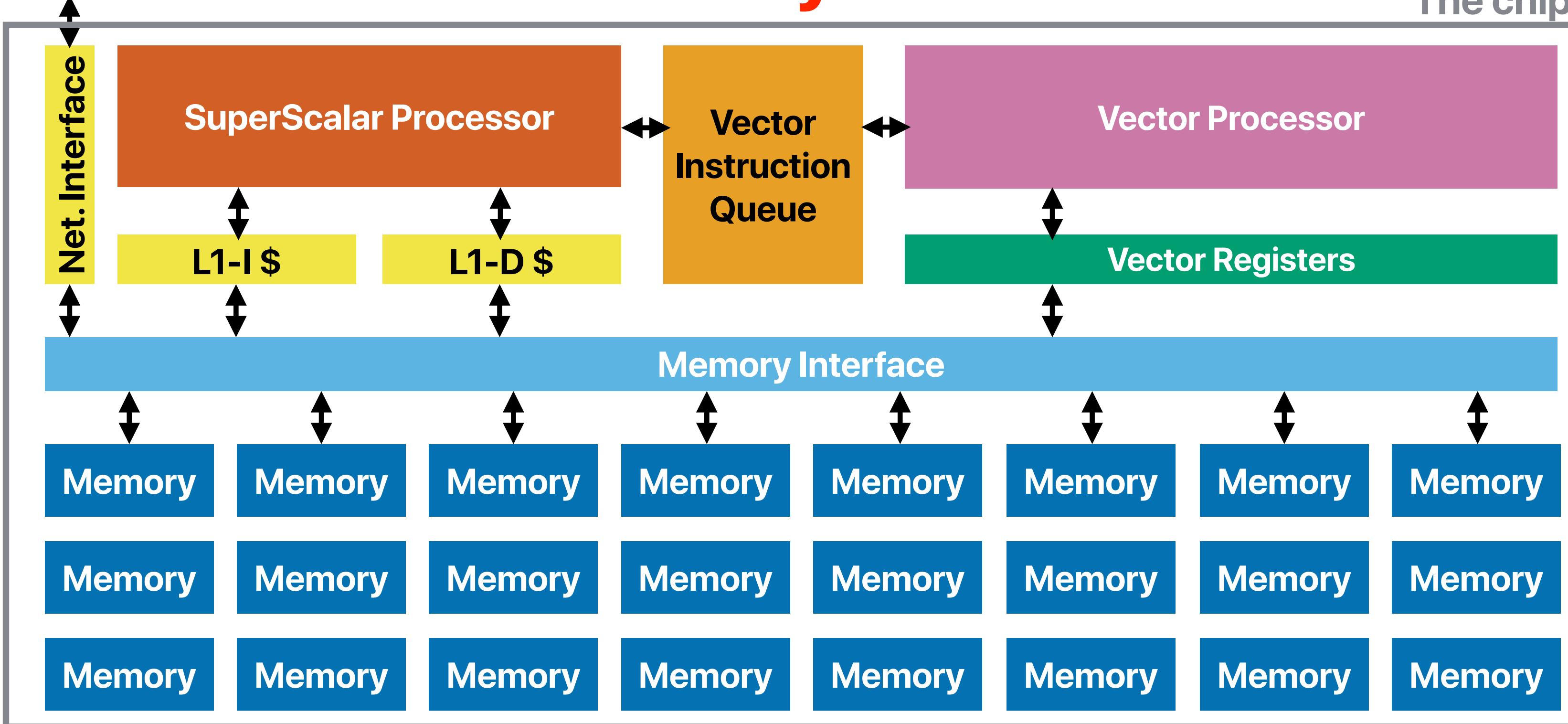
Reconfigurable Logic

Output

M. Oskin, F. Chong and T. Sherwood, "Active Pages: A Computation Model for Intelligent Memory," in Computer Architecture, International Symposium on, Barcelona, Spain, 1998

# Berkeley IRAM

The chip



David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 1997. A Case for Intelligent RAM. IEEE Micro 17, 2 (March 1997), 34–44.

**What kinds of and what applications  
would fit well in IRAM or Active  
Pages?**

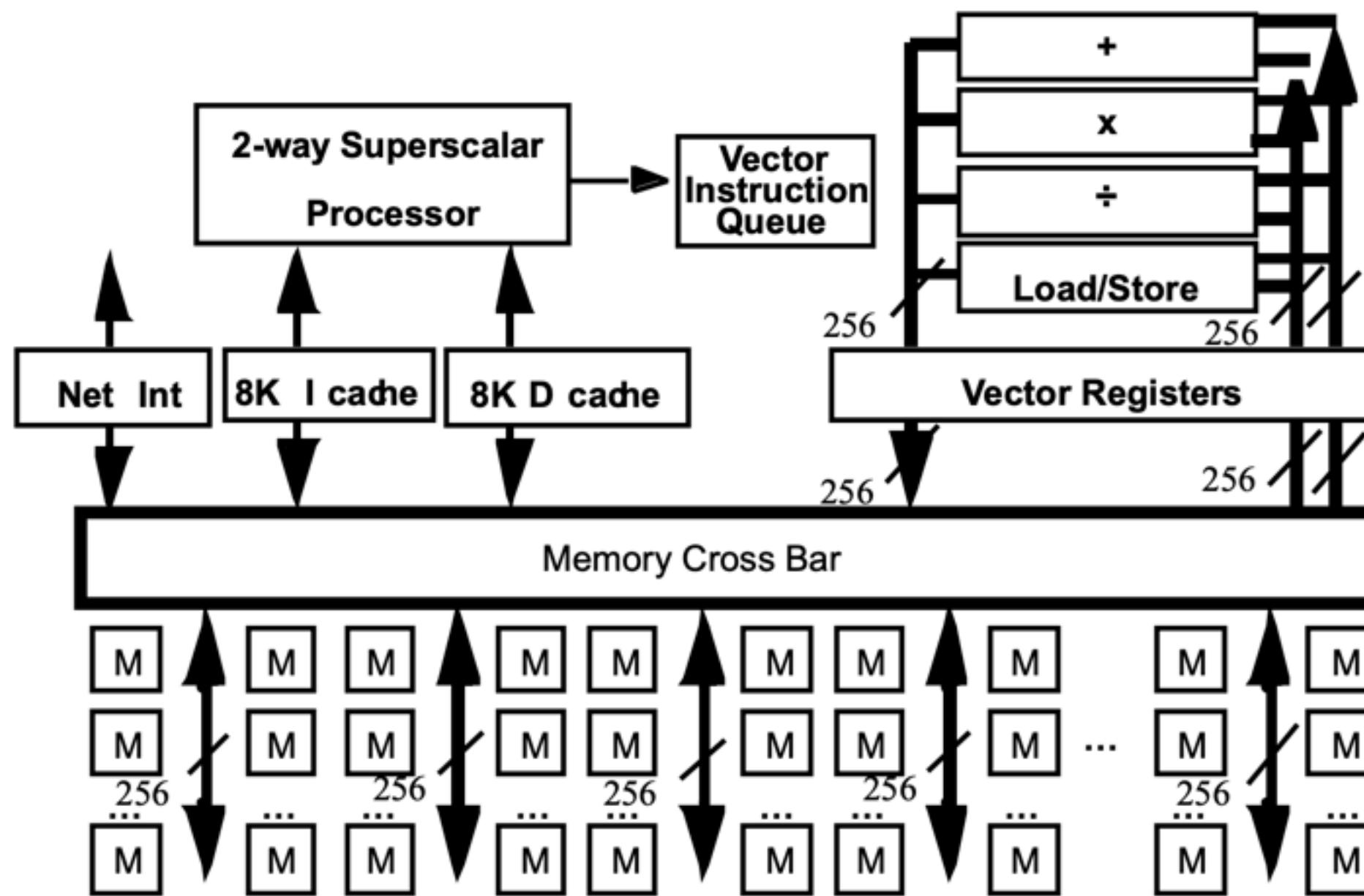
# What kind of applications?

- Both Active Pages and IRAM operates on a “page” or “pages” of data simultaneously
- Throughput oriented rather than latency sensitive
- Cannot be cached well
  - DRAM-intensive
  - Low locality (hard-to-predict patterns)
  - Large data footprint

**Table 2: CPI, cache misses, and time spent in Alpha 21164 for four programs**

Category	SPECint92	SPECfp92	Database	Sparse
Clocks Per Instruction (CPI)	1.2	1.2	3.6	3.0
I cache misses per 1000 instructions	7	2	97	0
D cache misses per 1000 instructions	25	47	82	38
L2 cache misses per 1000 instructions	11	12	119	36
L3 cache misses per 1000 instructions	0	0	13	23
Fraction of time in processor	0.78	0.68	0.23	0.27
Fraction of time in I cache misses	0.03	0.01	0.16	0.00
Fraction of time in D cache misses	0.13	0.23	0.14	0.08
Fraction of time in L2 cache misses	0.05	0.06	0.20	0.07
Fraction of time in L3 cache misses	0.00	0.02	0.27	0.58

# $0.25 \mu\text{m}$ , Fast Logic IRAM: $\approx 500\text{MHz}$ 5 GFLOPS(64b)/40 GOPS(8b)/24MB



benchmark	Small			Large		
	Conven-	IRAM		Conven-	IRAM	
		tional	(.75 X)	(1.0 X)		
hsfsys	138	112 (0.81)	150 (1.08)	149	114 (0.77)	152 (1.02)
noway	111	99 (0.89)	132 (1.19)	127	104 (0.82)	139 (1.09)
nowsort	109	104 (0.95)	138 (1.27)	136	110 (0.81)	147 (1.08)
gs	119	107 (0.90)	142 (1.20)	141	109 (0.78)	146 (1.04)
ispell	145	113 (0.78)	151 (1.04)	149	115 (0.77)	153 (1.03)
compress	91	102 (1.13)	137 (1.50)	127	104 (0.82)	139 (1.09)
go	97	96 (0.99)	128 (1.31)	128	98 (0.76)	130 (1.02)
perl	136	106 (0.78)	141 (1.04)	140	107 (0.76)	142 (1.01)

**Table 6: Performance (in MIPS) of IRAM versus conventional processors, as a function of processor slowdown in a DRAM process.**  
 Only the models with the 32:1 DRAM to SRAM-cache area density ratio are shown. The values in parentheses are the ratios of performances of the IRAM models compared to the CONVENTIONAL implementations. Ratios greater than 1.0 indicate that IRAM has higher performance.

# Active Pages: Applications

Memory-Centric Applications			
Name	Application	Processor Computation	Active Page Computation
Array	C++ standard template library array class	C++ code using array class Cross-page moves	Array insert, delete, and find
Database	Address Database	Initiates queries Summarizes results	Searches unindexed data
Median	Median filter for images	Image I/O	Median of neighboring pixels
Dynamic Prog	Protein sequence matching	Backtracking	Compute MINs and fills table

Processor-Centric Applications			
Name	Application	Processor Computation	Active Page Computation
Matrix	Matrix multiply for Simplex and finite element	Floating point multiplies	Index comparison and gather/scatter of data
MPEG-MMX	MPEG decoder using MMX instructions	MMX dispatch Discrete cosine transform	MMX instructions

Table 2: Summary of partitioning of applications between processor and active pages

# Active Pages

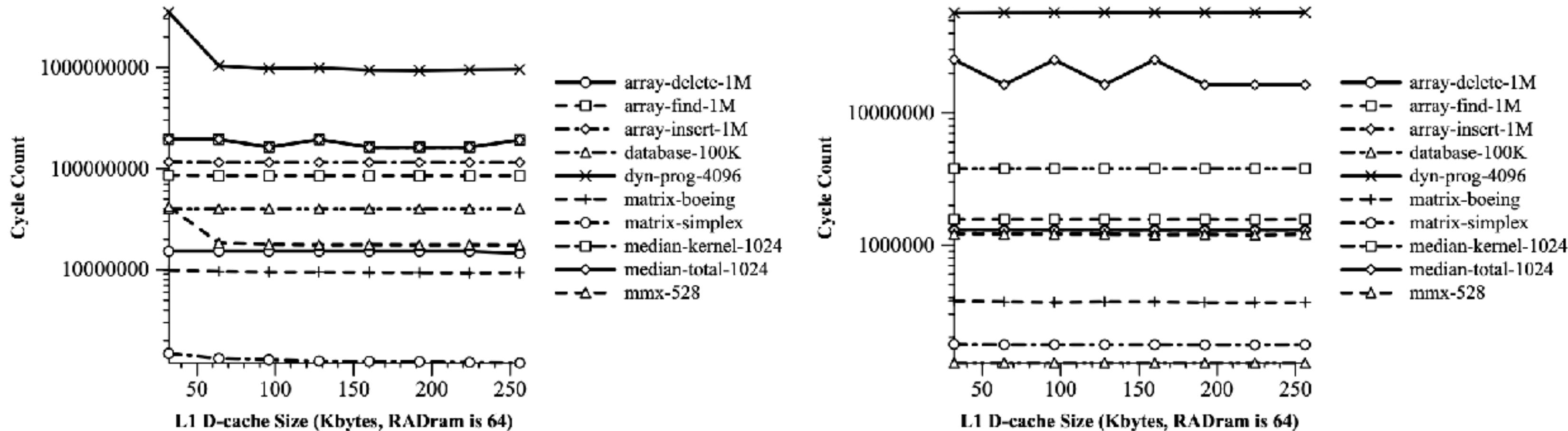


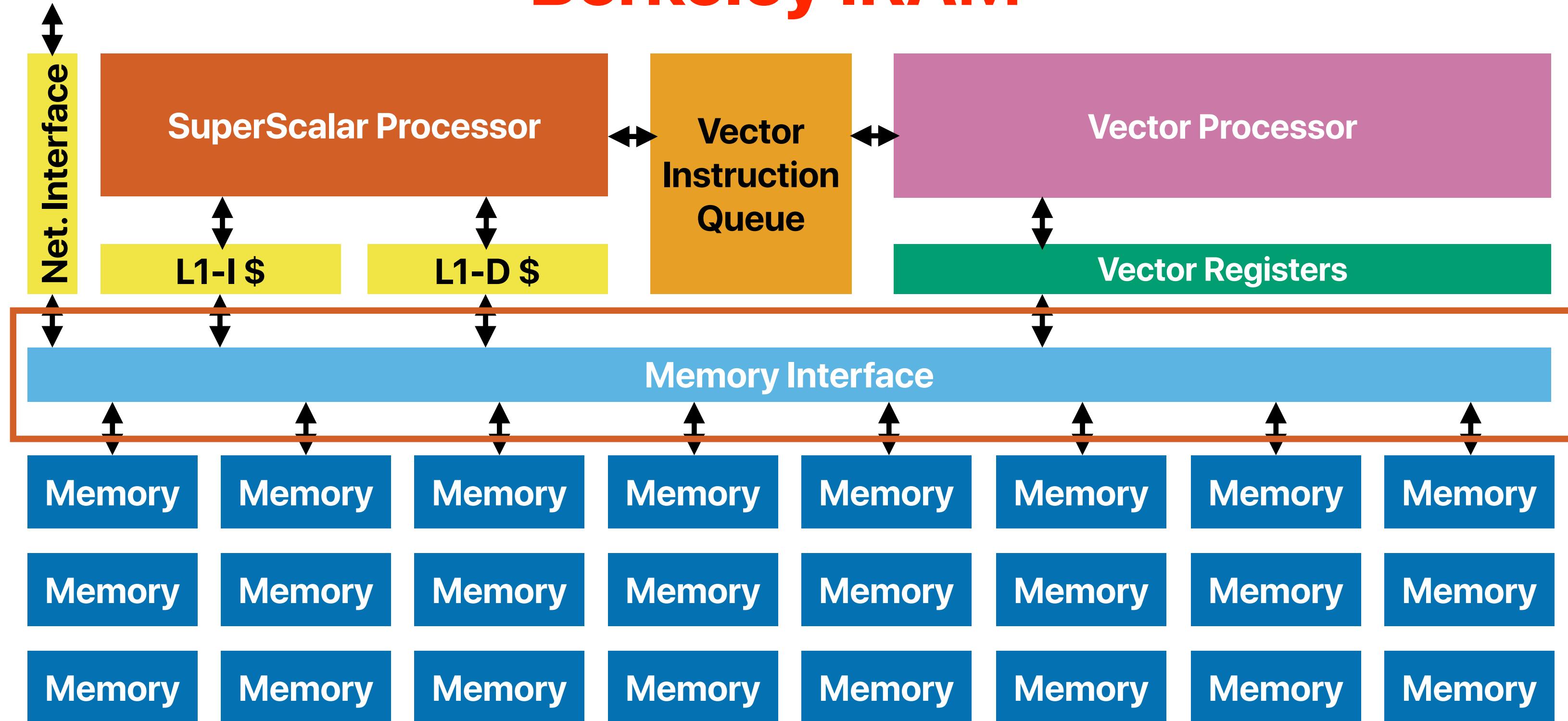
Figure 5: Conventional (**left**) and RADram (**right**) Execution Time vs. L1 Data Cache Size

**Why “active pages” and “iRAM”  
do not work out during that era?**

# Why IRAM or Active Pages does not work out?

- Programming
  - Why GPU succeed later?
- Applications
- Lack of industrial implementation, support
- Hardware design — Processor v.s. DRAM process technologies
- Cost
  - For Active Pages — small processor each module
  - For IRAM
    - High bandwidth on-chip interconnect was expensive
    - Other hardware design issues
      - Significant differences in clock rates among units

# Berkeley IRAM



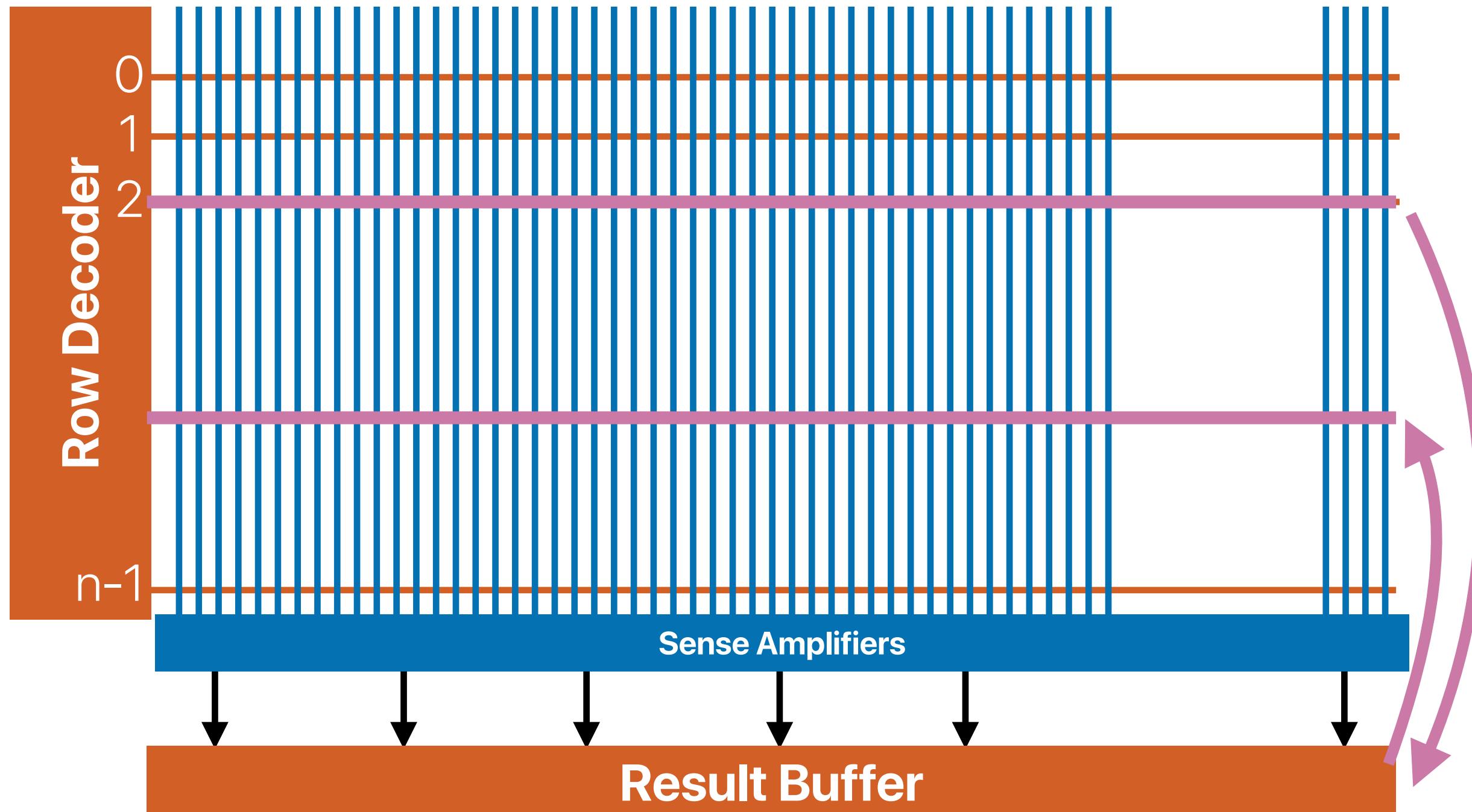
David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 1997. A Case for Intelligent RAM. IEEE Micro 17, 2 (March 1997), 34–44.

# 3D-die stacking

- Providing huge bandwidth between memory chips and processors (e.g., HBM)
- The renaissance of near-memory processing!
  - Zhu, Qiuling, Berkin Akin, H. Ekin Sumbul, Fazle Sadi, James C. Hoe, Larry Pileggi, and Franz Franchetti. A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing. In 3DIC. 2013.
  - Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L. Greathouse, Lifan Xu, and Michael Ignatowski. TOP-PIM: throughput-oriented programmable processing in memory. HPDC '14. 2014
  - Farmahini-Farahani, Amin, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In HPCA. 2015.
  - Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. ISCA '15. 2015.
  - Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In MICRO '52. 2019

# **“In”-DRAM Processing**

# Or we just clone/move?

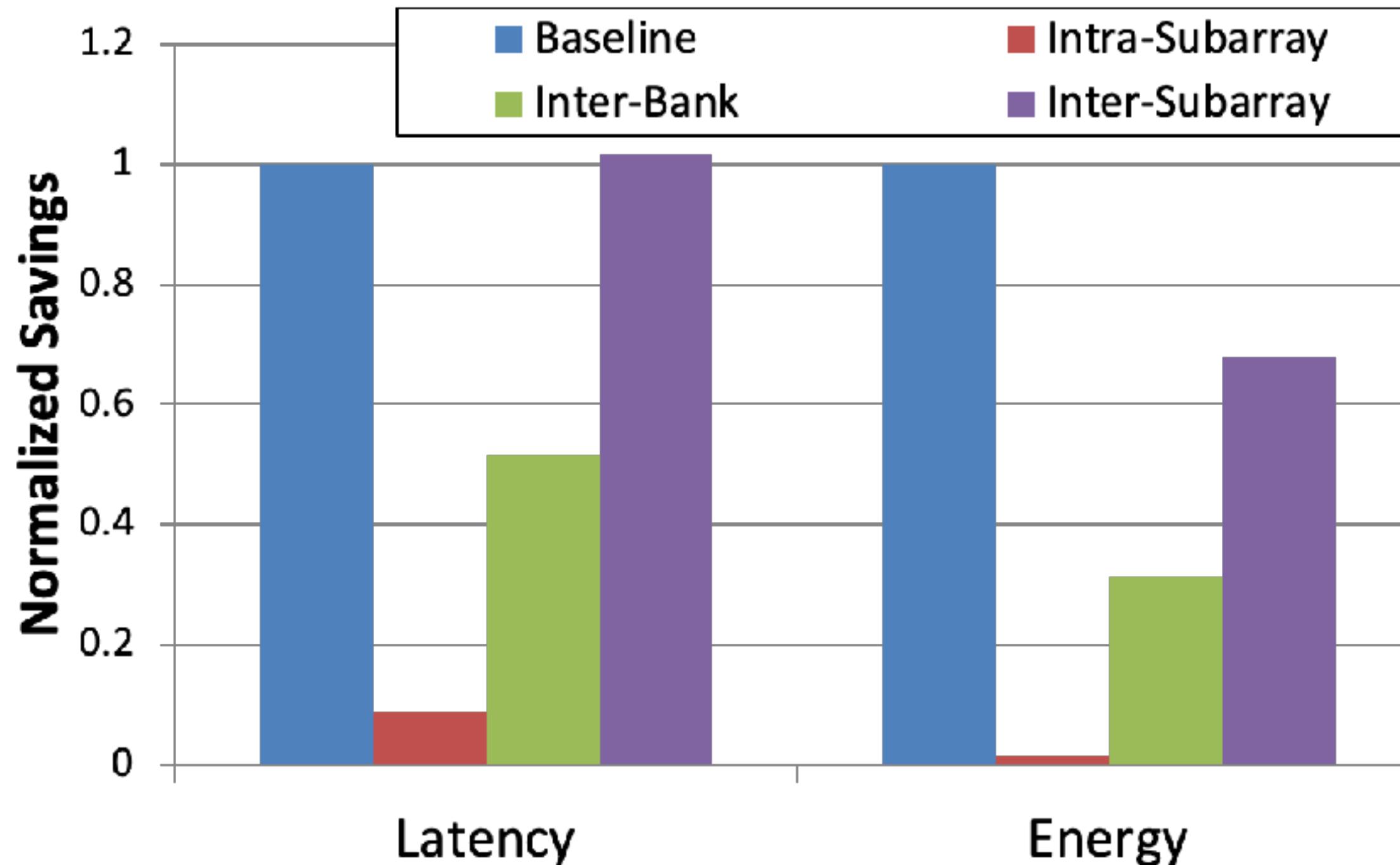


**memmove & memcpy: 5% cycles in Google's datacenter**

Svilen Kanev, Juan Pablo Darago, Kim Hazelwood,  
Partha Sarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and  
David Brooks. Profiling a warehouse-scale computer ISCA '15

Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry.  
RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization. In MICRO-46.

# The effect of RowClone

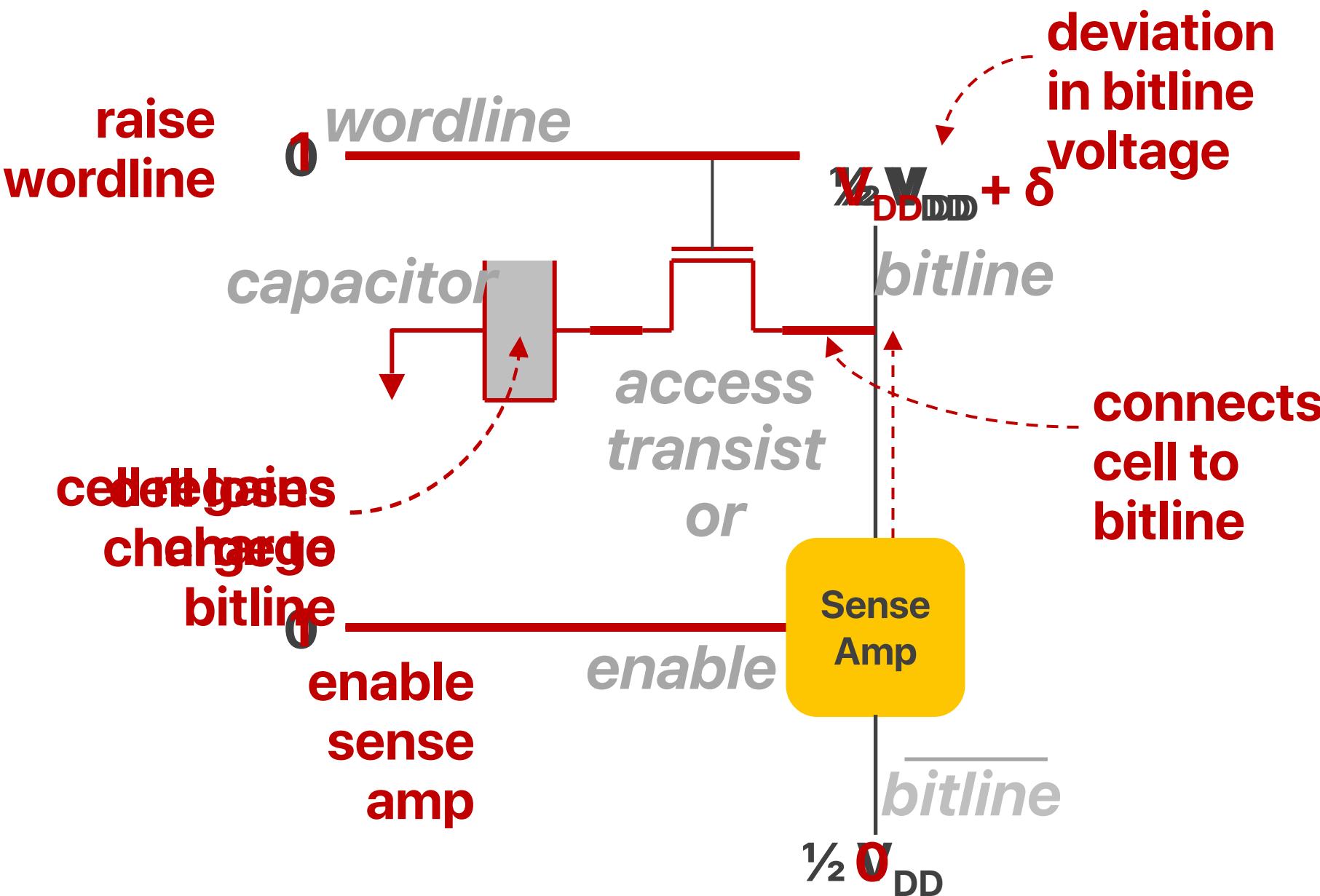


Do you think RowClone is a good idea? Why or why not?

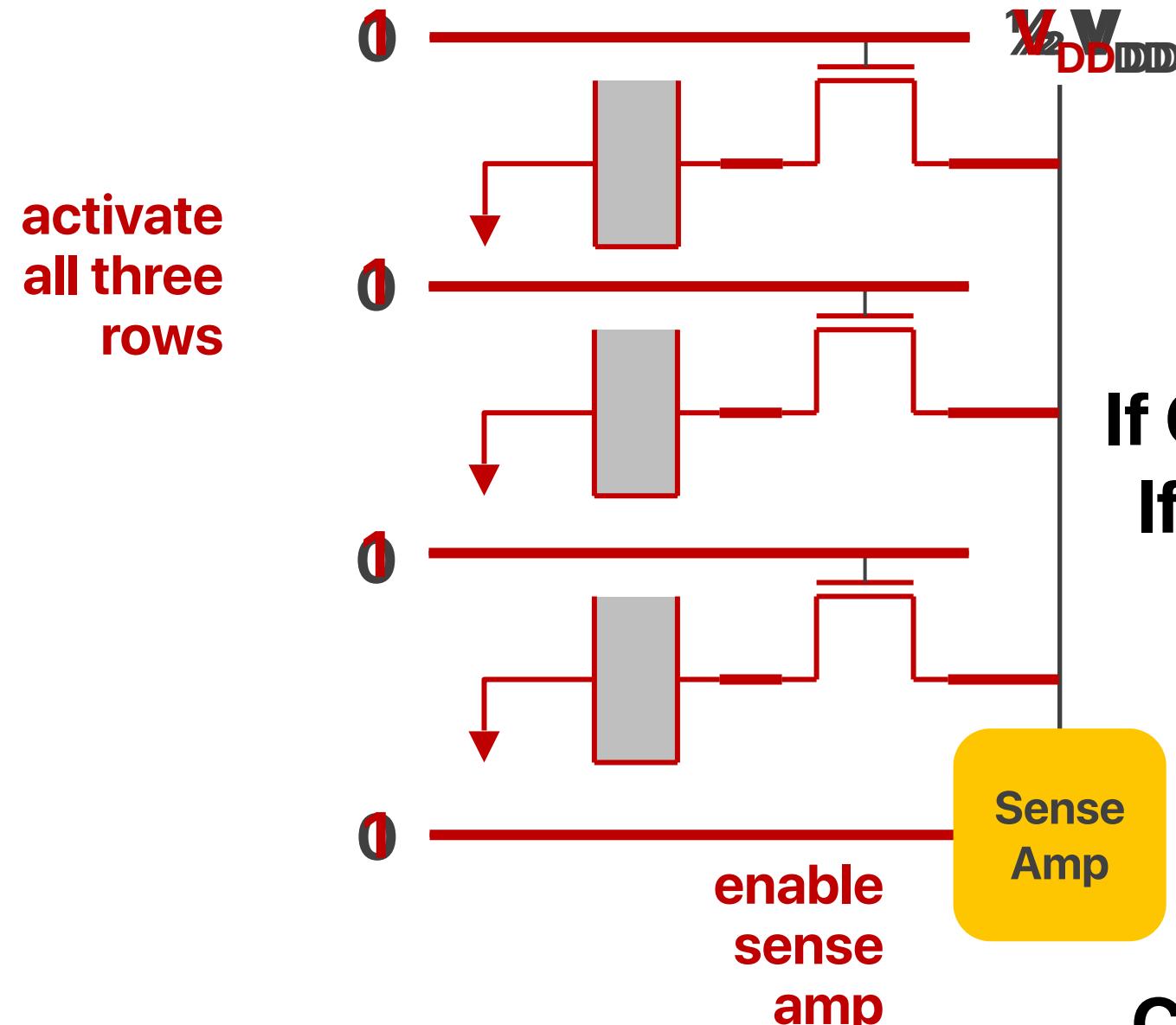
# Limitations of RowClone

- Limited functionality — can only perform RowClone
- Data mapping — what if pages are not within the same chip?
- Hardware design — additional cost

# Basic DRAM Cell Operation



# Activate Three Rows



$$AB + BC + AC = C(A+B) + C'AB$$

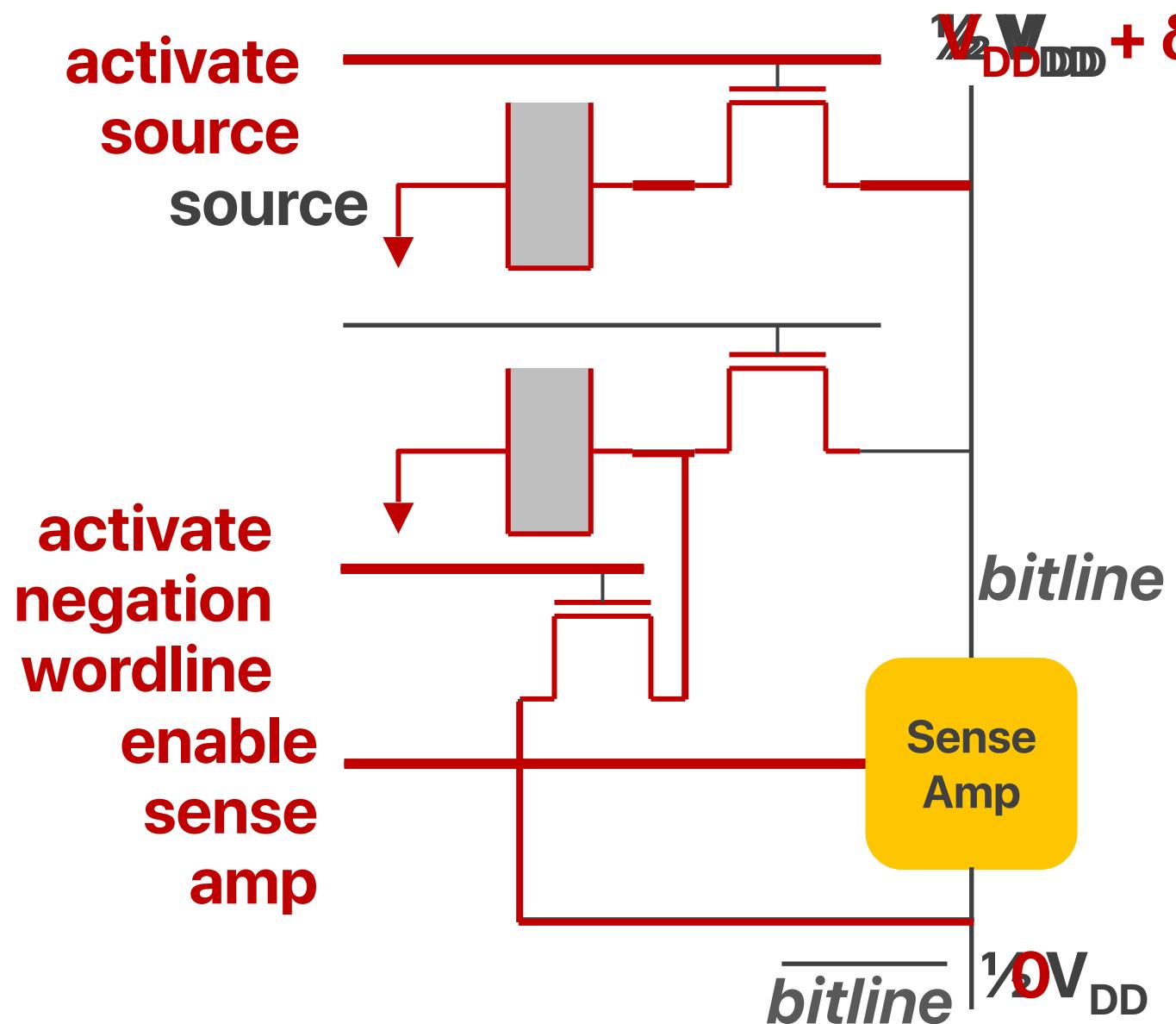
If C is 0, we can compute AND  
If C is 1, we can compute OR

Input			Output
A	B	C	
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

	A'B'	A'B	AB	AB'	
A'B	Out(A, B)	0,0	0,1	1,1	1,0
C'	0	0	0	1	0
C	1	0	1	1	1

BC      AB      AC

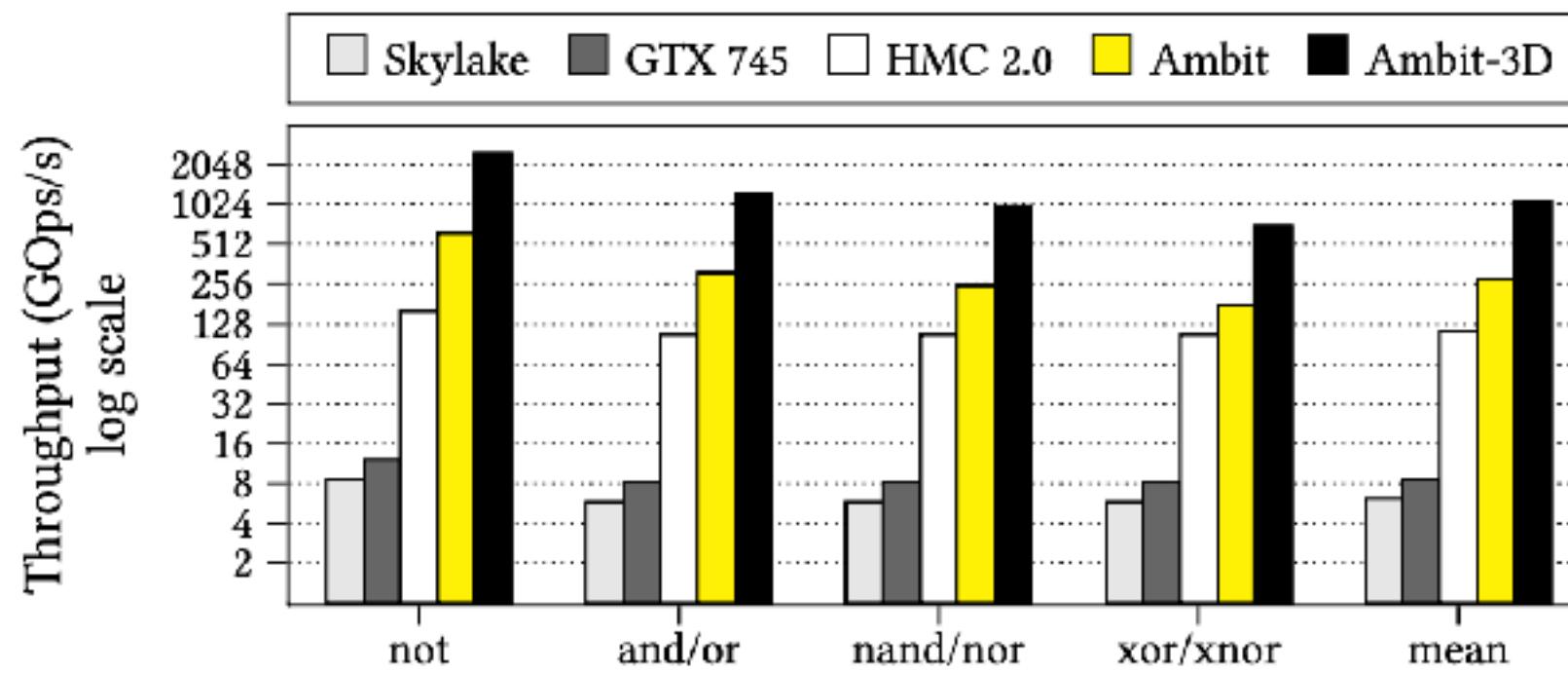
# We can also do NOT



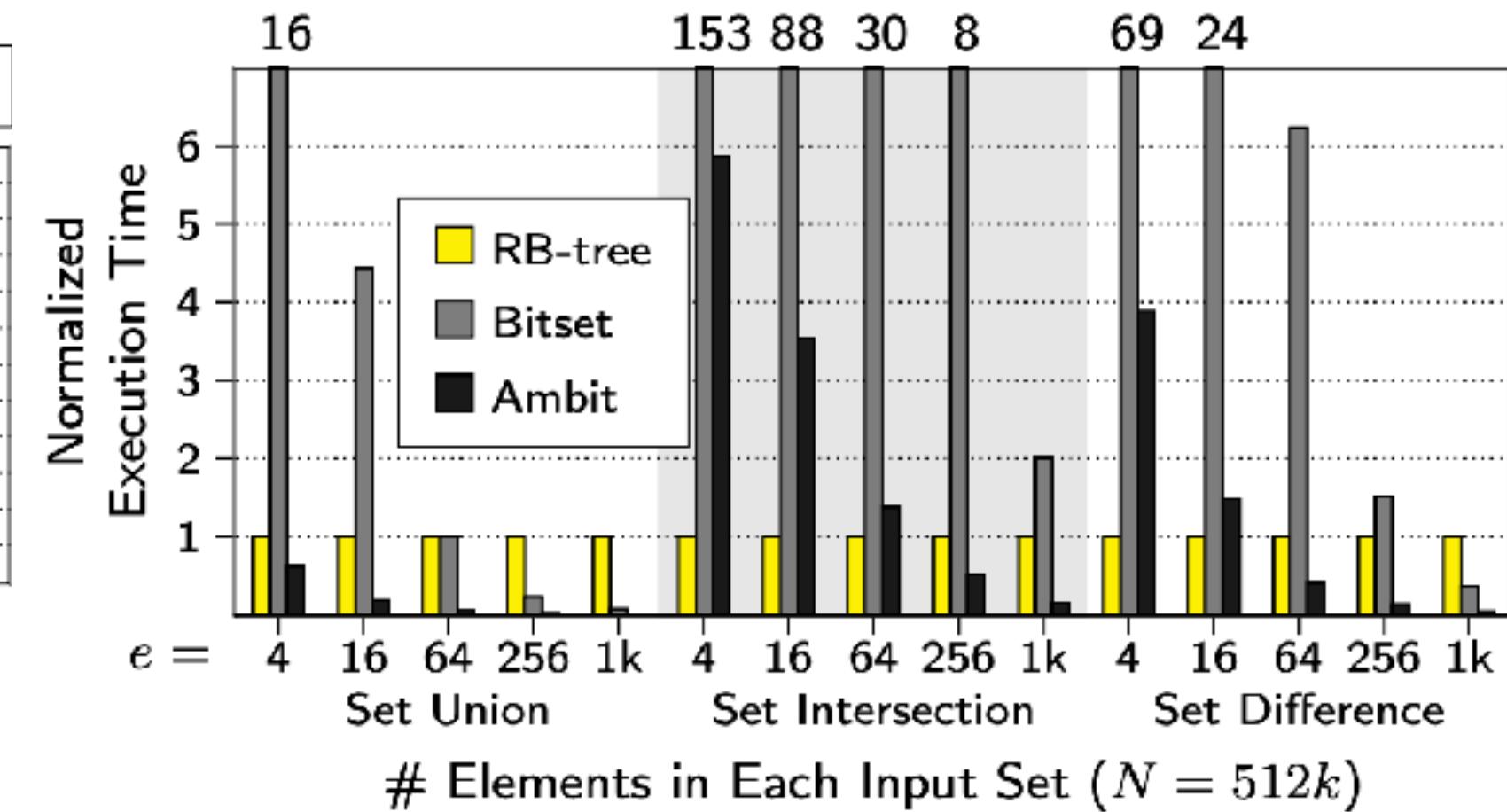
What's the meaning of the ability to perform NAND and NOR?

NAND and NOR are “universal gates” that you can achieve all logical functions with them!

# Ambit



**Figure 9: Throughput of bulk bitwise operations.**



**Figure 12: Performance of set operations**

# Limitations of in-DRAM processing

- Programming
- Data mapping — it's effective only if data happen to be within the same array/module
- Hardware design — not that many operations are available

# **Processing Using Memory**

# Charge-based v.s. resistive memory

- Charge-based memory (e.g., SRAM, DRAM, Flash)
  - Write data by capturing the charge
    - DRAM: capacitor — leakage
    - Flash: floating-gate — wear-out quickly
  - Read data by measuring the voltage level
- Resistive memory (e.g., PCM, MRAM, RRAM)
  - Write data by change the “material” properties
    - PCM: change material phase
    - STT: change magnet polarity
    - RRAM: change atomic structure/atom distance
  - Read data by measuring the resistance

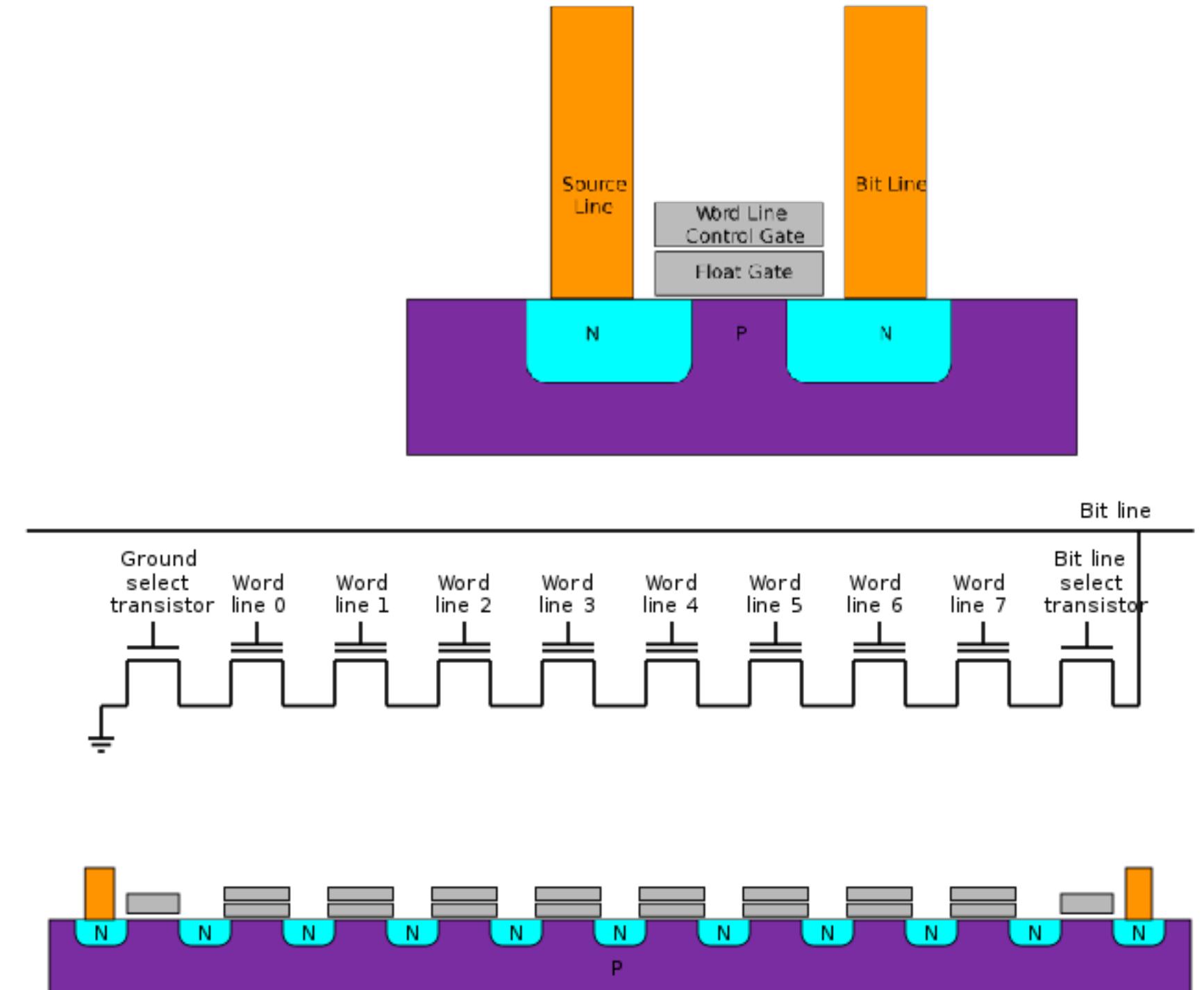
$$V_C = V_s \times e^{\frac{-t}{RC}}$$

Ohm's law

$$V = I \times R$$

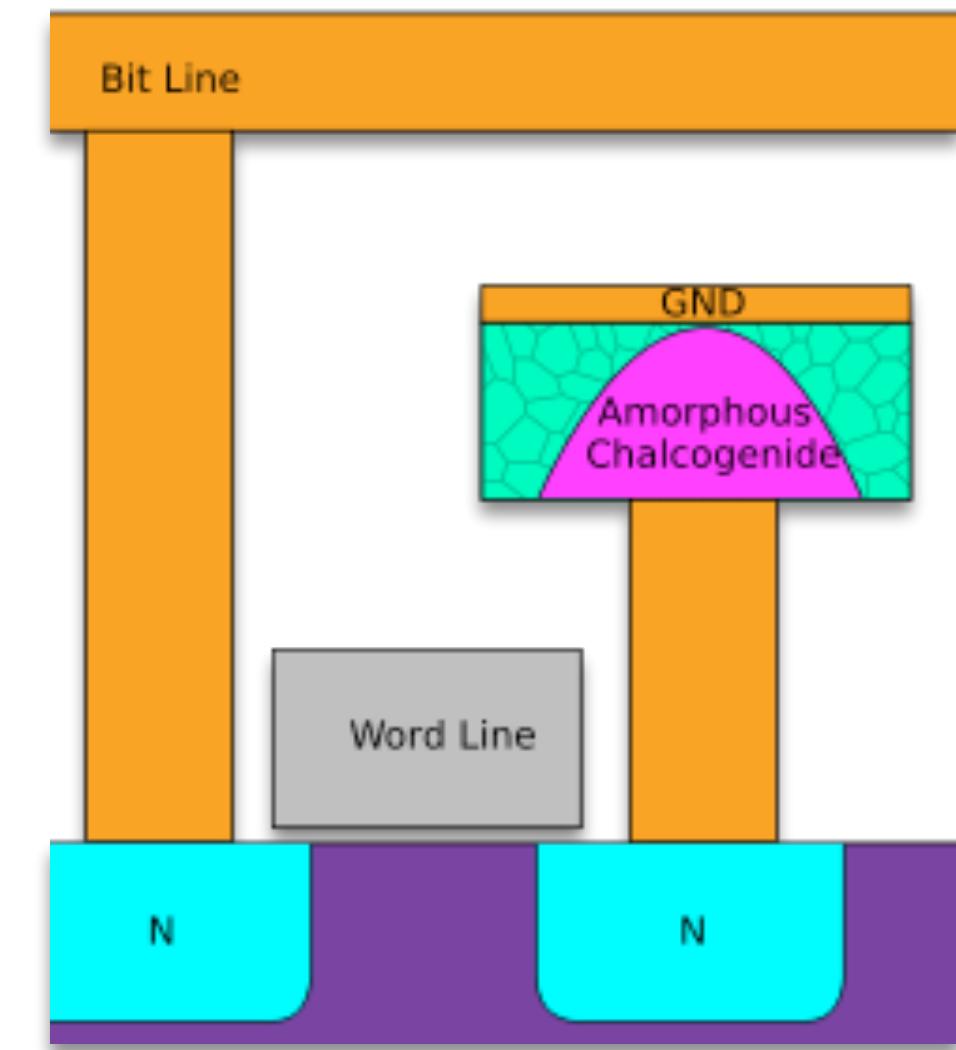
# Flash memory

- Floating gate made by polycrystalline silicon trap electrons
- The voltage level within the floating gate determines the value of the cell
- The floating gates will wear out eventually



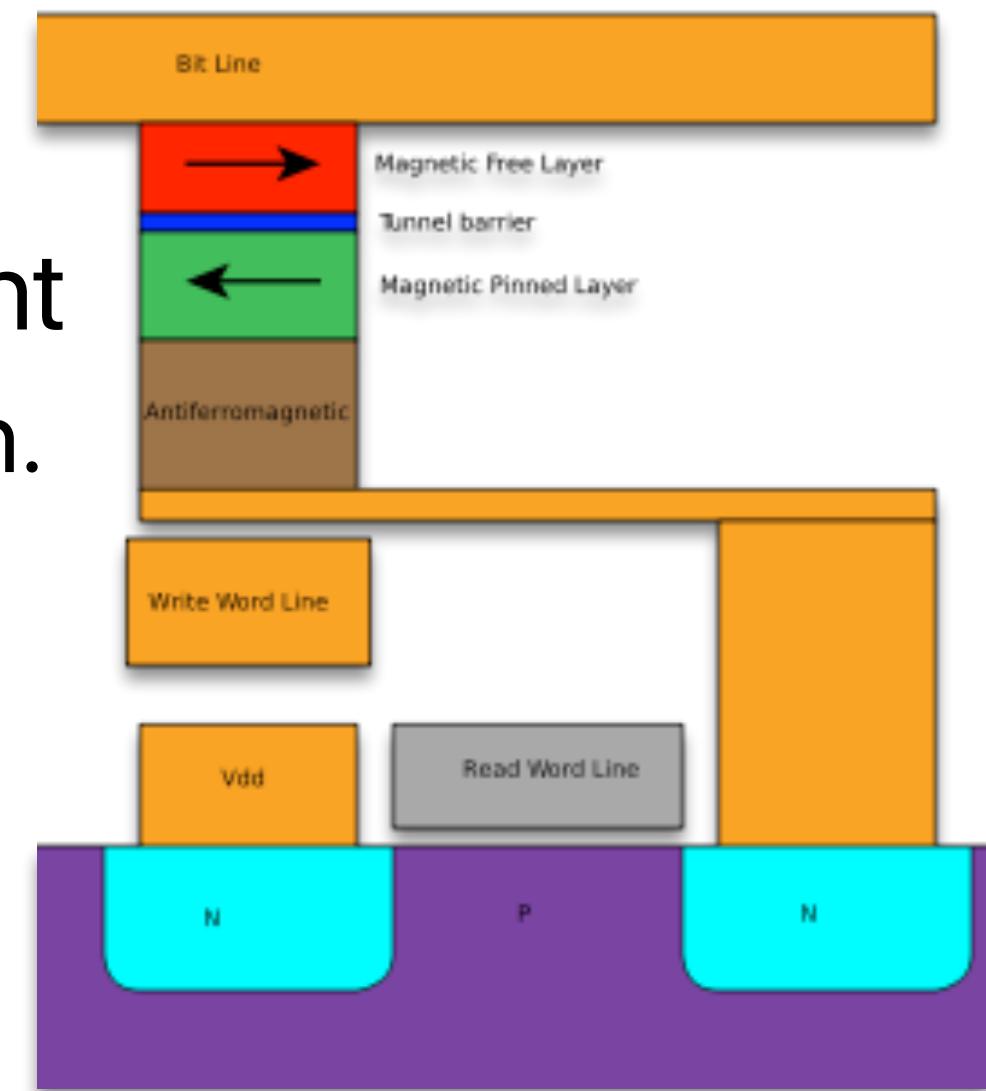
# Phase change memory

- The bit is stored in the crystal structure of a tiny spec of metal.
- To write, it melts the metal (650C)
  - let it cool quickly or slowly to set the value
  - Crystaline and amorphous states have different resistance
    - Amorphous: Low optical reflexivity and high electrical resistivity
    - Crystalline: High optical reflexivity and low electrical resistivity



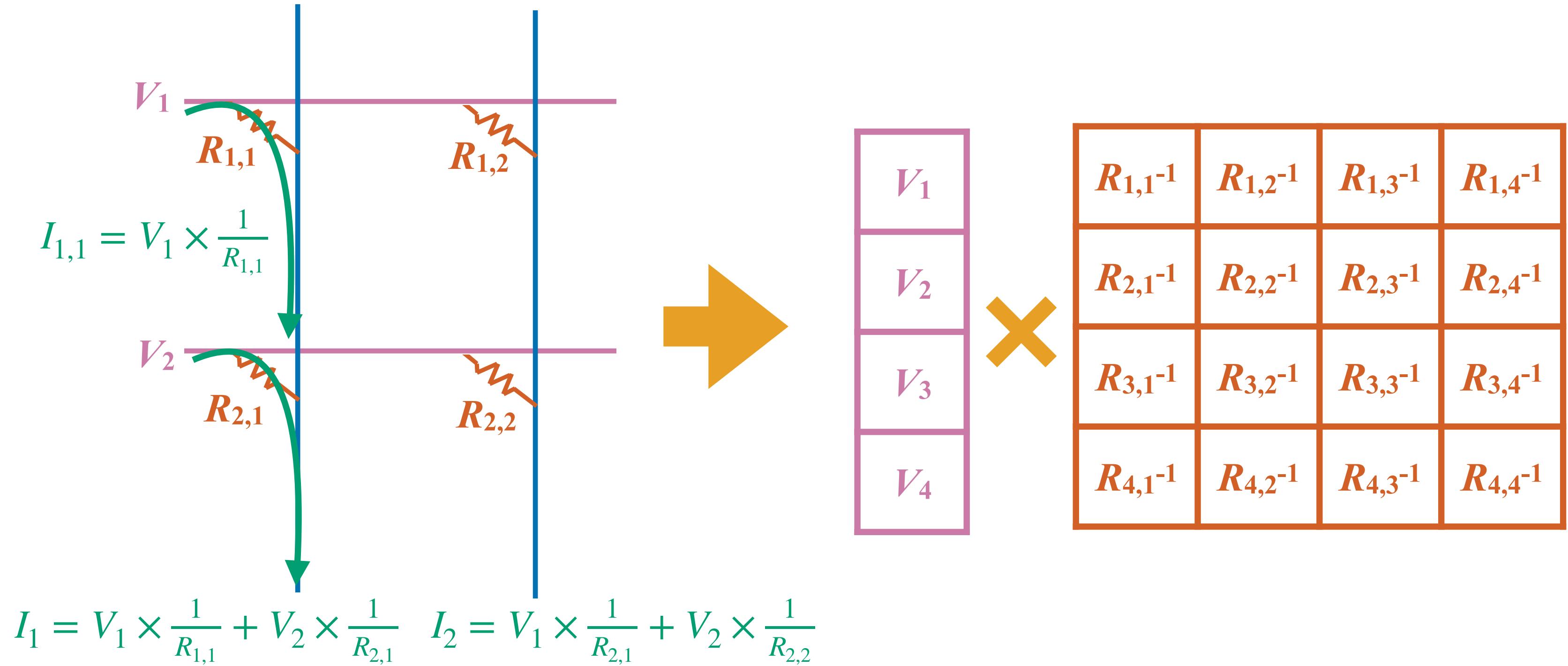
# Spin-torque transfer

- Bits stored as magnetic orientation of a thin film
- Change the state using polarized electrons (!)
- Depending on polarization, resistance differs
- More complex cell structure
- Great promise — potential DRAM replacement
  - Roughly the same speed, power, and bandwidth.
  - But it's durable!



# **“In”-NVM processing**

# ReRAM



**What's the limitation of ReRAM-based  
matrix multiplier? What applications  
can tolerate these limitations?**

# Limitations of ReRAM-based accelerator

# Limitations of ReRAM-based accelerator

- Limited Precision
- A/D and D/A Conversion
  - Area and power increases exponentially with ADC resolution and frequency
  - Large area, Power hungry e.g., 98% of the total area and 89% of the total power
- Array Size and Routing
  - Wire dominates energy for array size of  $1k \times 1k$
  - IR drop along wire can degrade read accuracy
- Write/programming energy
  - Multiple pulses can be costly
- Variations & Yield
  - Device-to-device, cycle-to-cycle
  - Non-linear conductance across range

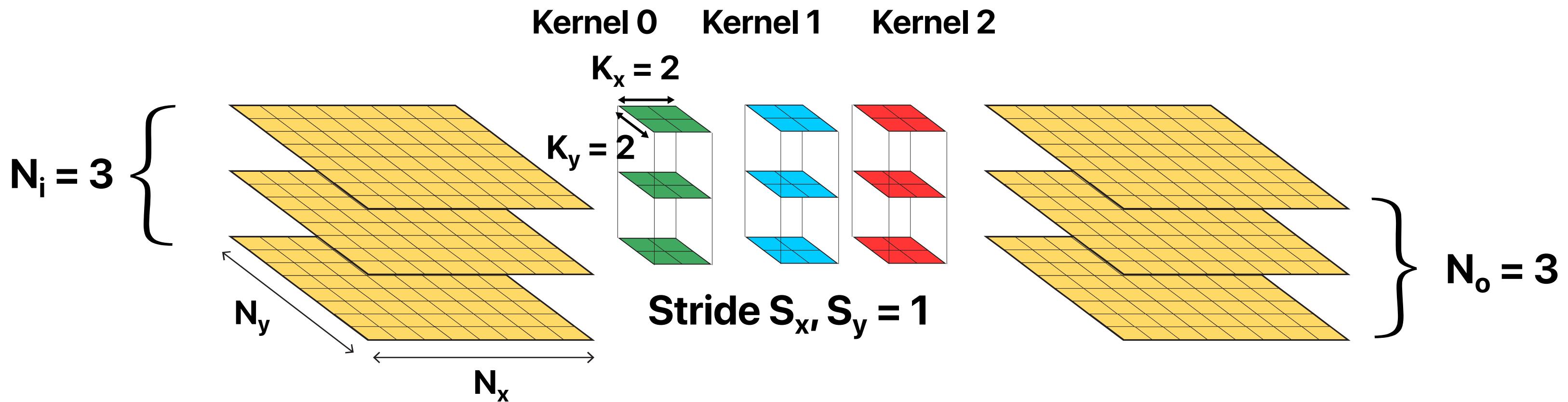
# References

- ADC/DAC optimizations
  - H. Yun, H. Shin, M. Kang and L. -S. Kim, "Optimizing ADC Utilization through Value-Aware Bypass in ReRAM-based DNN Accelerator," 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021, pp. 1087-1092, doi: 10.1109/DAC18074.2021.9586140.
  - Qilin Zheng, Zongwei Wang, Zishun Feng, Bonan Yan, Yimao Cai, Ru Huang, Yiran Chen, Chia-Lin Yang, and Hai (Helen) Li. 2020. Lattice: an ADC/DAC-less ReRAM-based processing-in-memory architecture for accelerating deep convolution neural networks. In Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference (DAC '20). IEEE Press, Article 190, 1–6.
- Digital PIM
  - Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. 2019. FloatPIM: in-memory acceleration of deep neural network training with high precision. In Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19). Association for Computing Machinery, New York, NY, USA, 802–815. <https://doi.org/10.1145/3307650.3322237>

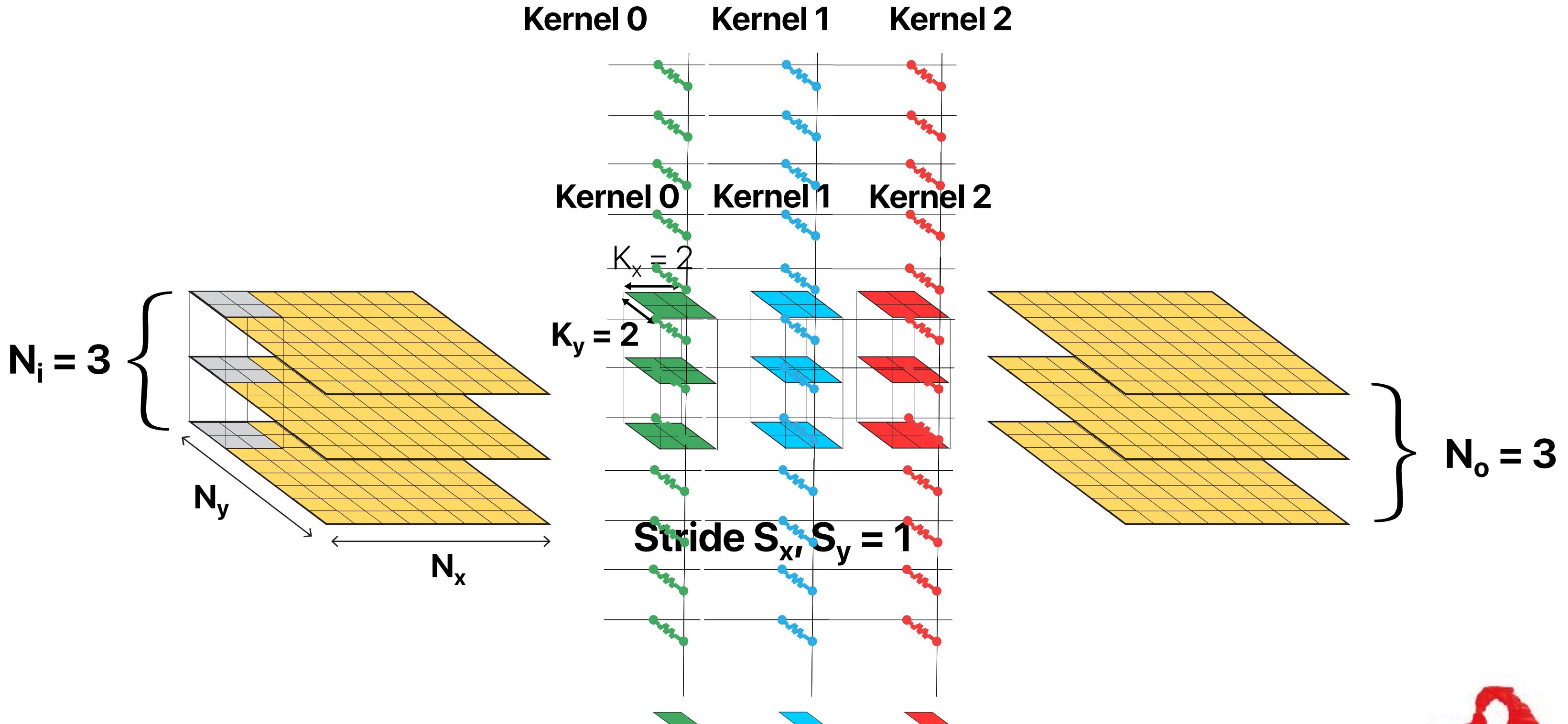
# Applications of ReRAM-based accelerator

- NN
  - Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In ISCA '16. 2016
  - Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In ISCA '16. 2016
  - Song, Linghao, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In HPCA 2017.
  - Mohsen Imani, Saransh Gupta, Yesoeng Kim, and Tajana Rosing. 2019. FloatPIM: in-memory acceleration of deep neural network training with high precision. ISCA. 2019.

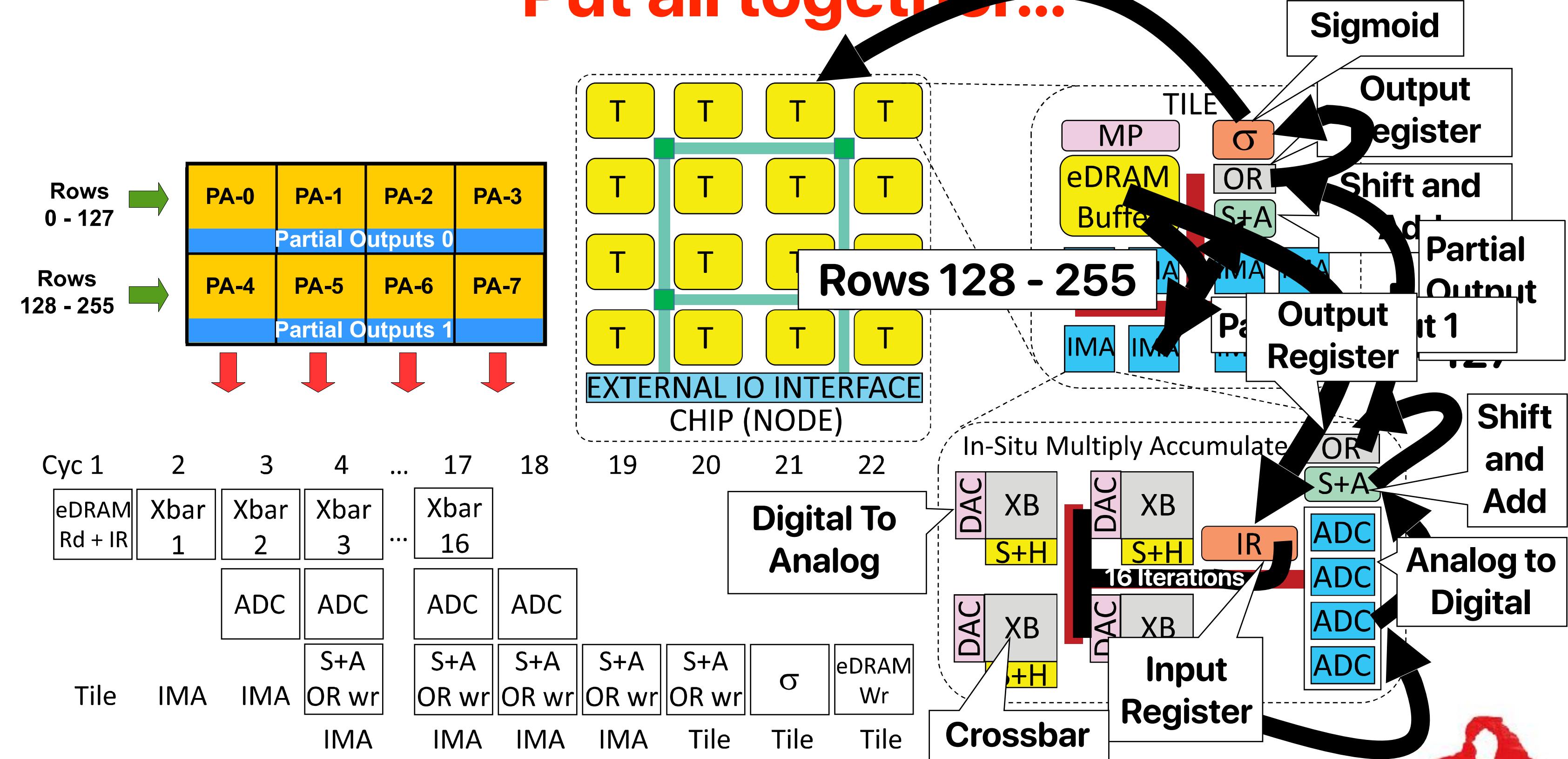
# Convolution



# Convolution in ReRAM



# Put all together...

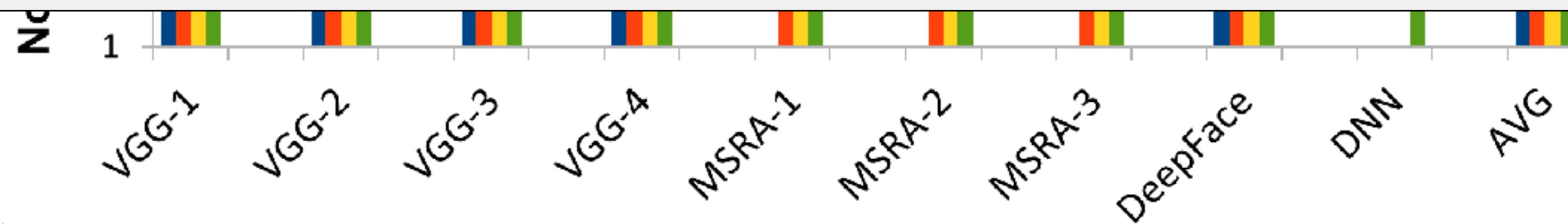




**Throughput: 14.8x** better because:

1. Memristor crossbar have high computational parallelism
2. DaDianNao fetches both inputs and weights from eDRAM, ISAAC fetches just inputs
3. DaDianNao suffers due to bandwidth limitation in fully connected layers.

ISAAC requires more power but is **5.5x** better in terms of **energy** due to above reasons.



## Deep Neural Net Benchmarks

TABLE I  
COMPARISON TABLE OF RECENT ANALOG IMC-BASED MVM/MULTIPLY-ACCUMULATE (MAC)-OPERATION ACCELERATORS

Metric	This work	ISSCC'21 [45]	ISSCC'21 [44]	ISSCC'20 [46]
CMOS technology	14 nm	22 nm	16 nm	7 nm
Memory technology	PCM	ReRAM	SRAM	SRAM
Non-volatile	Yes	Yes	No	No
Operating Voltage in V	0.8	0.8	0.8	0.8
Operation Frequency	1 GHz	-	200 MHz	-
ADC design	CCO-based ADC	Sense amplifier	8bit SAR ADC	4bit Flash-ADC
Memory size	65.5 K	4 M	4.5 MB	4 KB
Unit-cell	8T4R	1T1R	10T1C	8T
Number of input/weight/output-bits	8b/Analog/8b	8b/8b/14b	4b/4b/8b	4b/4b/4b
Peak Throughput (TOPS)	1.008	0.035	11.8   5.90 <sup>1</sup>	0.372   0.186 <sup>1</sup>
Energy Efficiency (TOPS/W)	10.5	11.91	121   60.5 <sup>1</sup>	351   175.5 <sup>1</sup>
Area Efficiency (TOPS/mm <sup>2</sup> )	1.59	0.013	2.67   1.34 <sup>1</sup>	116.3   58.13 <sup>1</sup>

and functionality of the digital ADC calibration procedure is described in detail and the MVM accuracy is quantified. Finally, the measured classification accuracies of deep learning (DL) inference applications on the MNIST and CIFAR-10 datasets, when two IMC cores are employed, are presented. For a performance density of 1.59 TOPS/mm<sup>2</sup>, a measured energy efficiency of 10.5 TOPS/W, at a main clock frequency of 1 GHz, is achieved.

**How can we lower the data volume?**

# **Lower data volume**

**How can we lower the data volume?**

# How can we lower data volume?

# How can we lower data volume

- Reduce precision
- Compression

# Demo

# Demo

- Does reduce the overhead of data movement
- Computation overhead is a lot!
- <https://github.com/lz4/lz4>
- <https://github.com/NVIDIA/nvcomp>



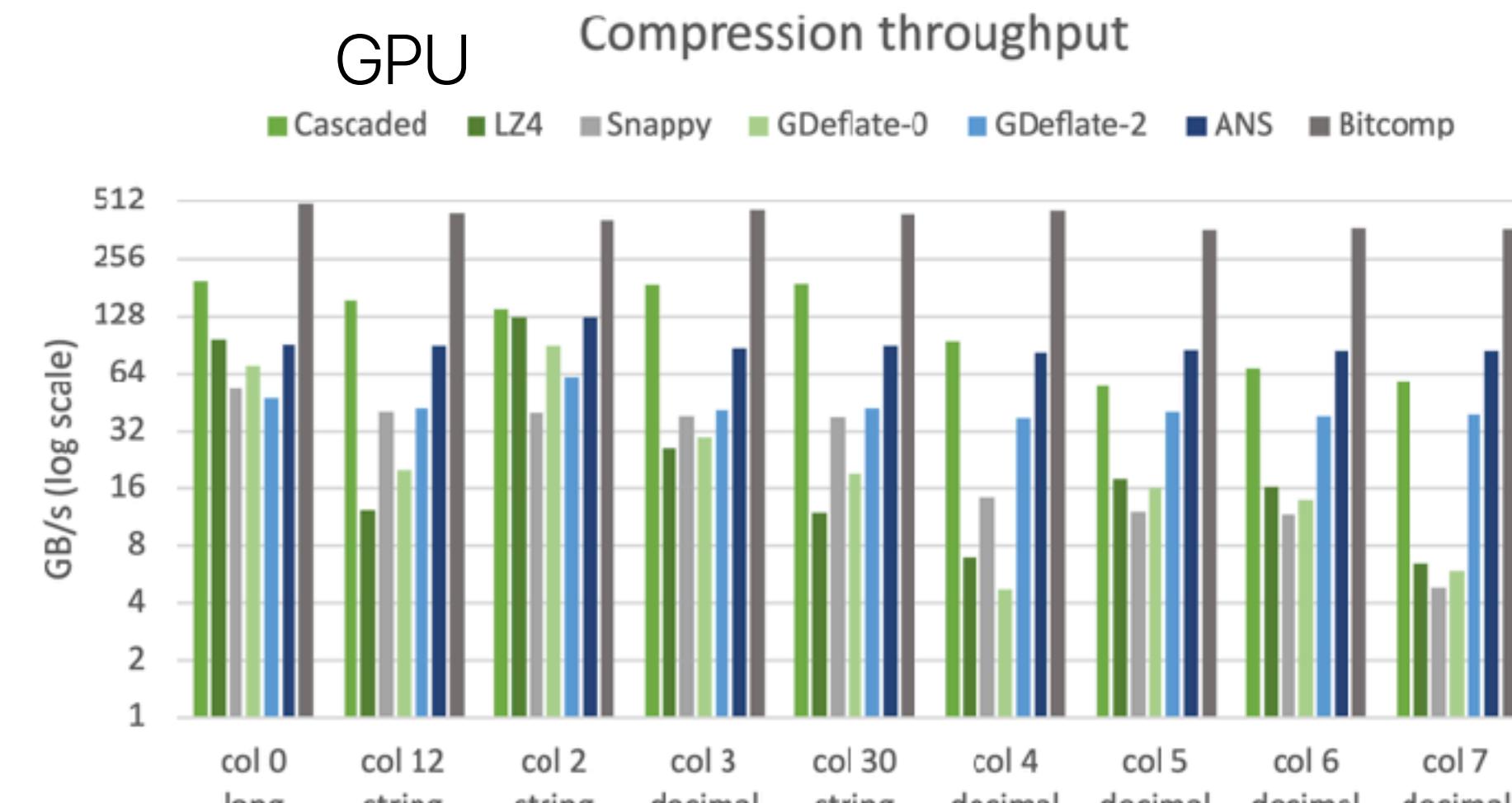
Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://PollEv.com/app)

## Benchmarks

### CPU

The benchmark uses [lzbench](#), from @inikep compiled with GCC v8.2.0 on Linux 64-bits (Ubuntu 4.18.0-17). The reference system uses a Core i7-9700K CPU @ 4.9GHz (w/ turbo boost). Benchmark evaluates the compression of reference [Silesia Corpus](#) in single-thread mode.

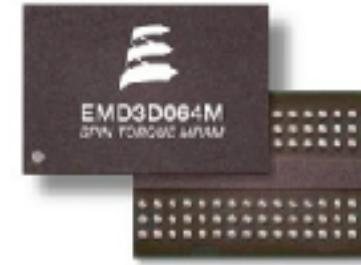
Compressor	Ratio	Compression	Decompression
memcpy	1.000	13700 MB/s	13700 MB/s
LZ4 default (v1.9.0)	2.101	780 MB/s	4970 MB/s
LZO 2.09	2.108	670 MB/s	860 MB/s
QuickLZ 1.5.0	2.238	575 MB/s	780 MB/s
Snappy 1.1.4	2.091	565 MB/s	1950 MB/s
Zstandard 1.4.0 -1	2.883	515 MB/s	1380 MB/s
LZF v3.6	2.073	415 MB/s	910 MB/s
<a href="#">zlib</a> deflate 1.2.11 -1	2.730	100 MB/s	415 MB/s
LZ4 HC -9 (v1.9.0)	2.721	41 MB/s	4900 MB/s
<a href="#">zlib</a> deflate 1.2.11 -6	3.099	36 MB/s	445 MB/s



LZ4 is also compatible and optimized for x32 mode, for which it provides additional speed performance.

# **Non-volatile memory**

# Non-volatile memory technologies



	H.D.D	Flash	Optane	STT-MRAM
Latency	~ 10-15 ms	~ 100 us (read) ~ 1 ms (write)	7 us (read) 18 us (write)	35 ns
Bandwidth	~200 MB/Sec	3.5 GB/sec (read) 2.1 GB/sec (write)	1.35 GB/sec (read) 290 MB/sec (write)	
Dollar/GB	0.0295	0.583	2.18	

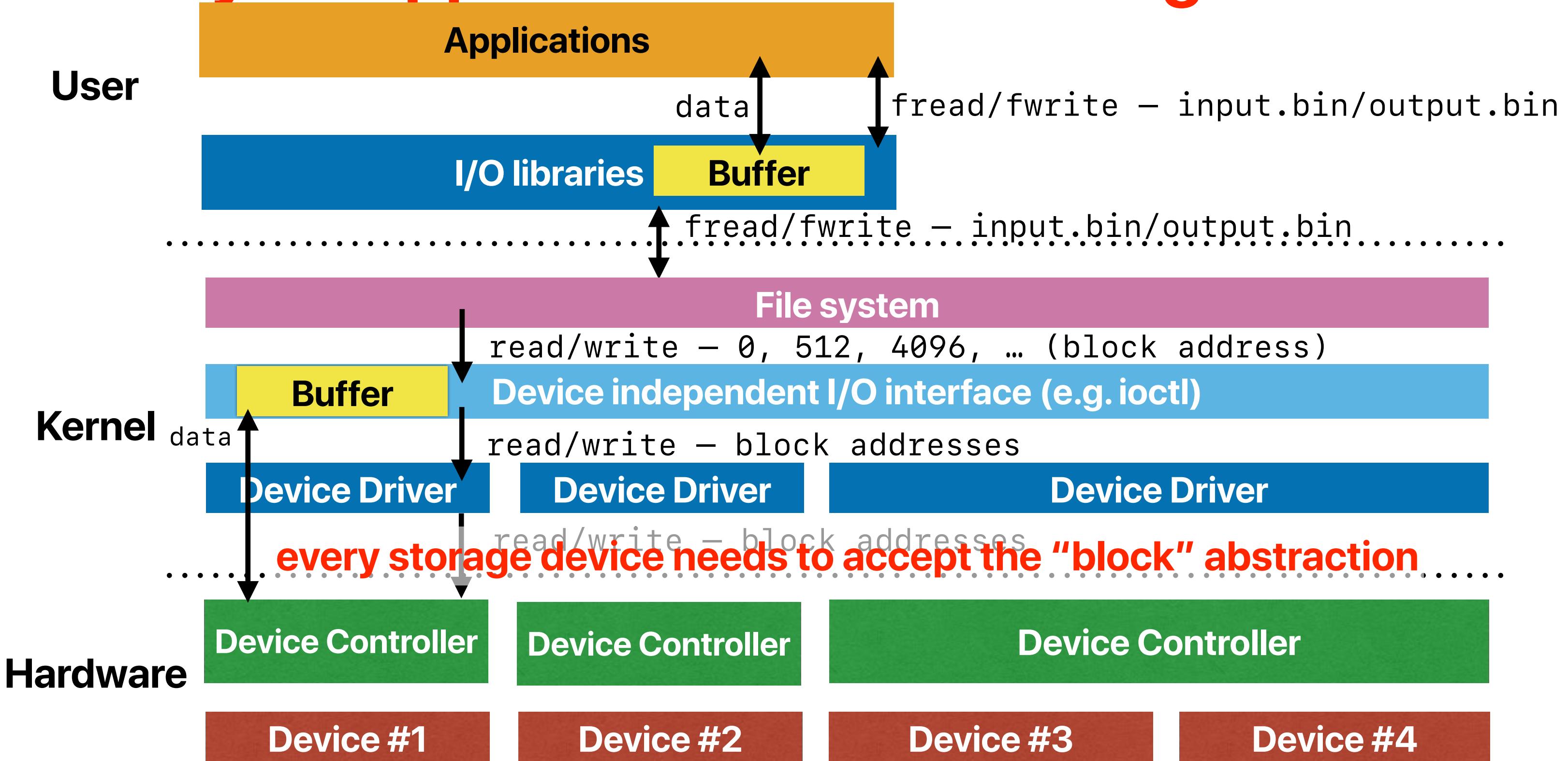
**Flash is still the most convincing technology for now**

Why do we need a controller  
inside an SSD?

# Why do we need controllers in SSDs?

- Maintaining the block device abstraction
- Dealing with the “weird” device characteristics
- Interfacing with the interconnect

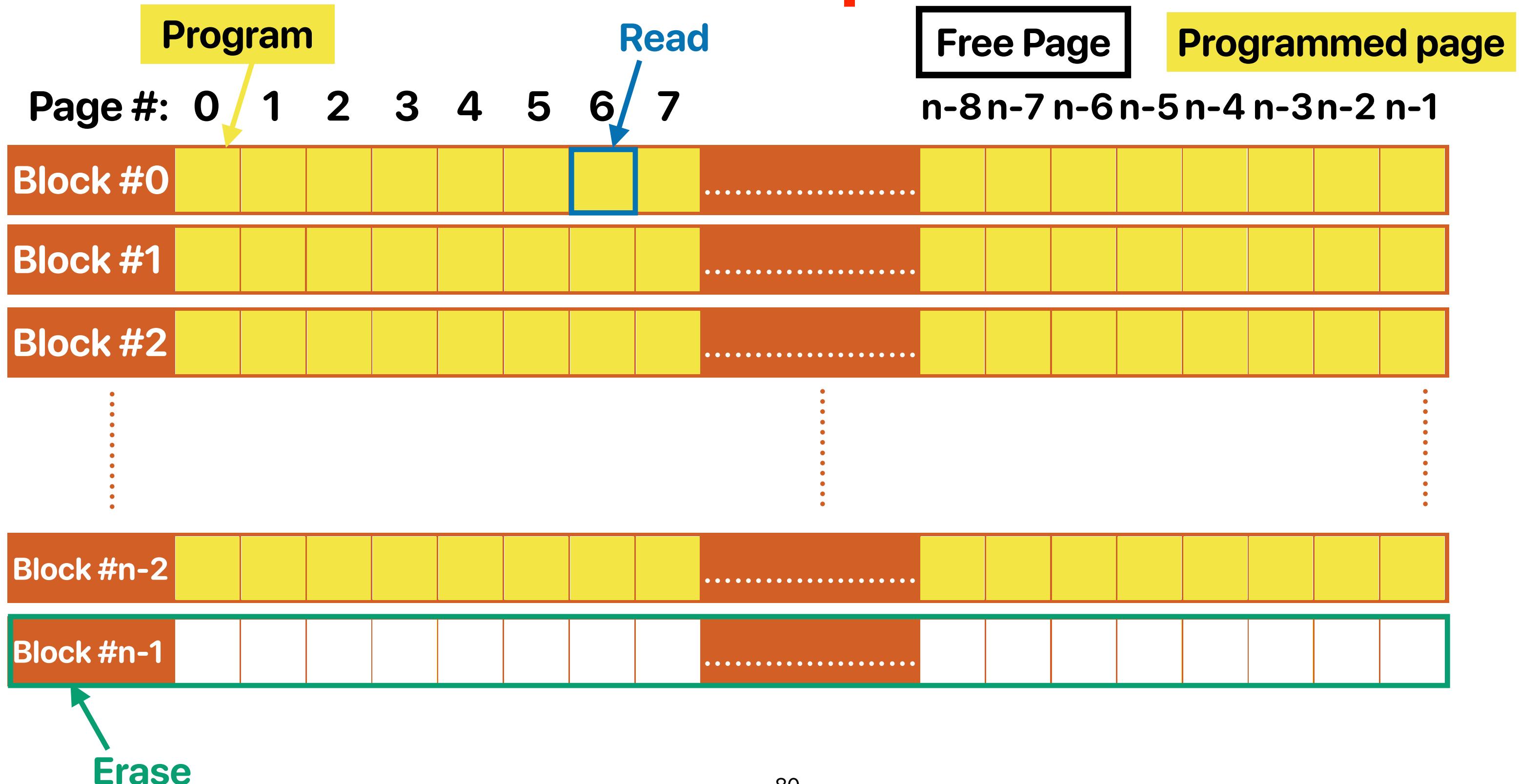
# How your application reaches storage devices



# NAND flash is just odd!

- Modern SSDs are based on NAND flash memory
- Different operation granularities
  - Read/Program in pages
  - Erase in blocks (64-384 pages)
- Performance of operation varies
  - Read — tens of us
  - Program — hundreds of us
  - Erase — ms
- Limited erase cycles
  - Only 1000 times for “QLC”

# Basic flash operations

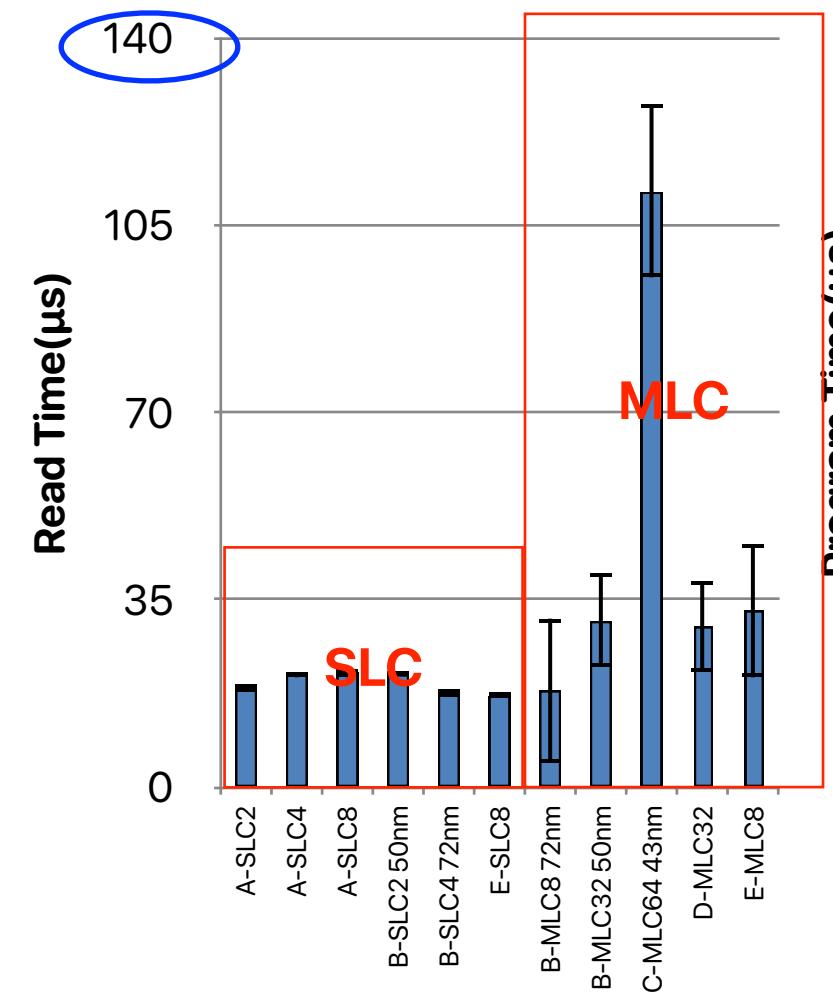


# Why we need a processor in SSDs?

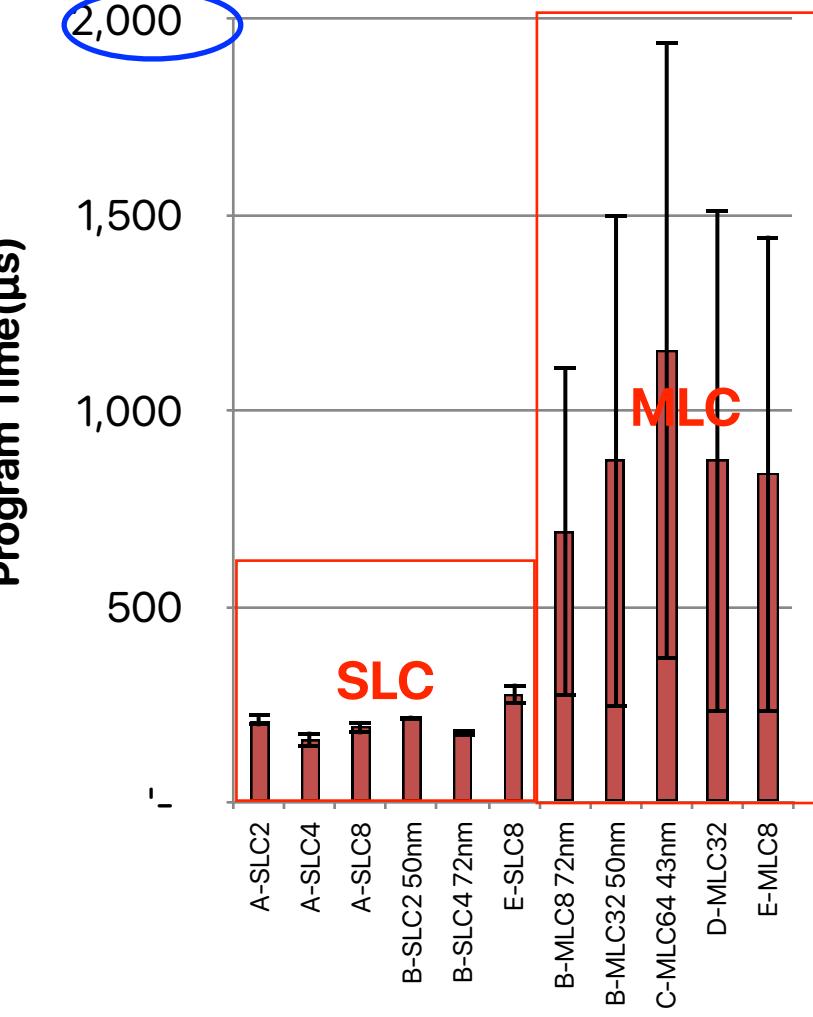
- Modern SSDs are based on NAND flash memory
- Different operation granularities
  - Read/Program in pages
  - Erase in blocks (64-384 pages)
- Performance of operation varies
  - Read — tens of us
  - Program — hundreds of us
  - Erase — ms
- Limited erase cycles
  - Only 1000 times for “QLC”

Not a good practice

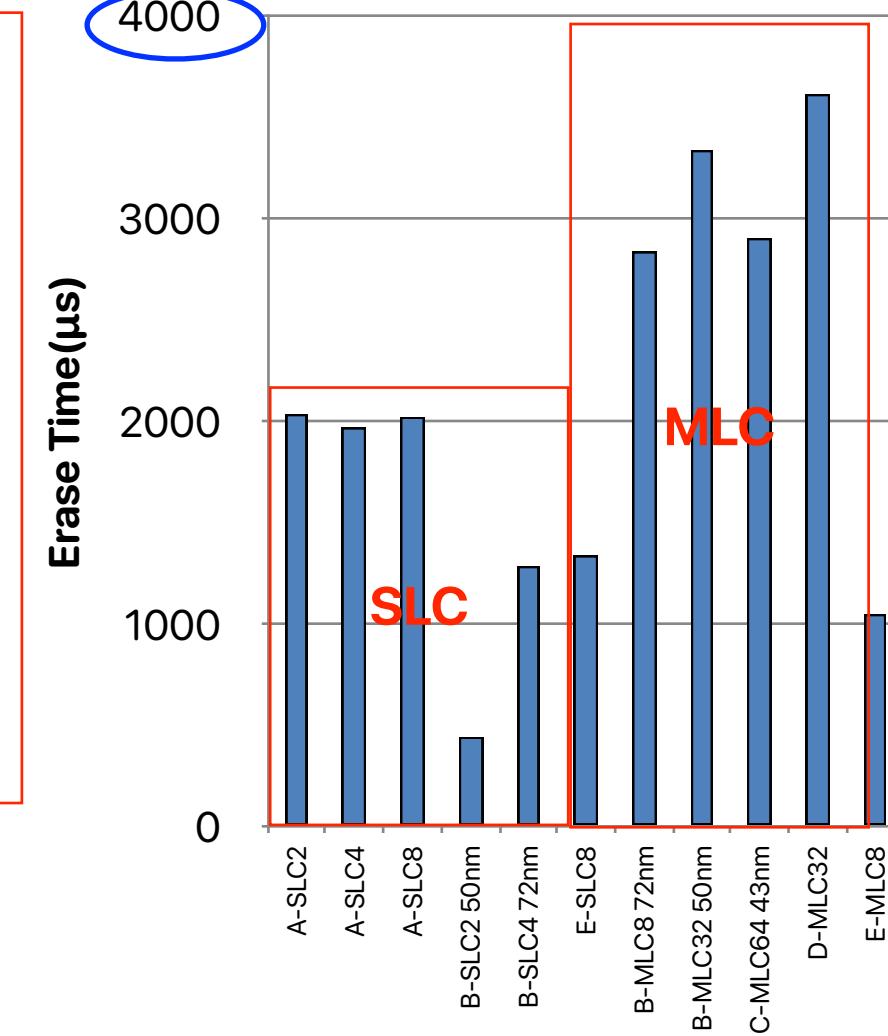
# Flash performance



Reads:  
less than 150us



Program/write:  
less than 2ms



Erase:  
less than 3.6ms

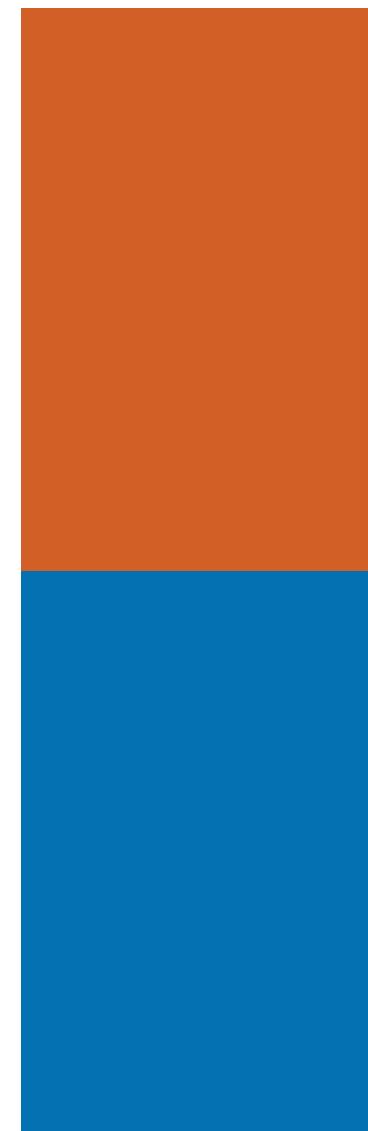
Similar relative performance for reads, writes and erases

# Why we need a processor in SSDs?

- Modern SSDs are based on NAND flash memory
- Different operation granularities
  - Read/Program in pages
  - Erase in blocks (64-384 pages)
- Performance of operation varies
  - Read — tens of us
  - Program — hundreds of us
  - Erase — ms
- Limited erase cycles
  - Only 1000 times for “QLC”

# Types of Flash Chips

2 voltage levels,  
1-bit



**Single-Level Cell  
(SLC)**

4 voltage levels,  
2-bit



**Multi-Level Cell  
(MLC)**

8 voltage levels,  
3-bit



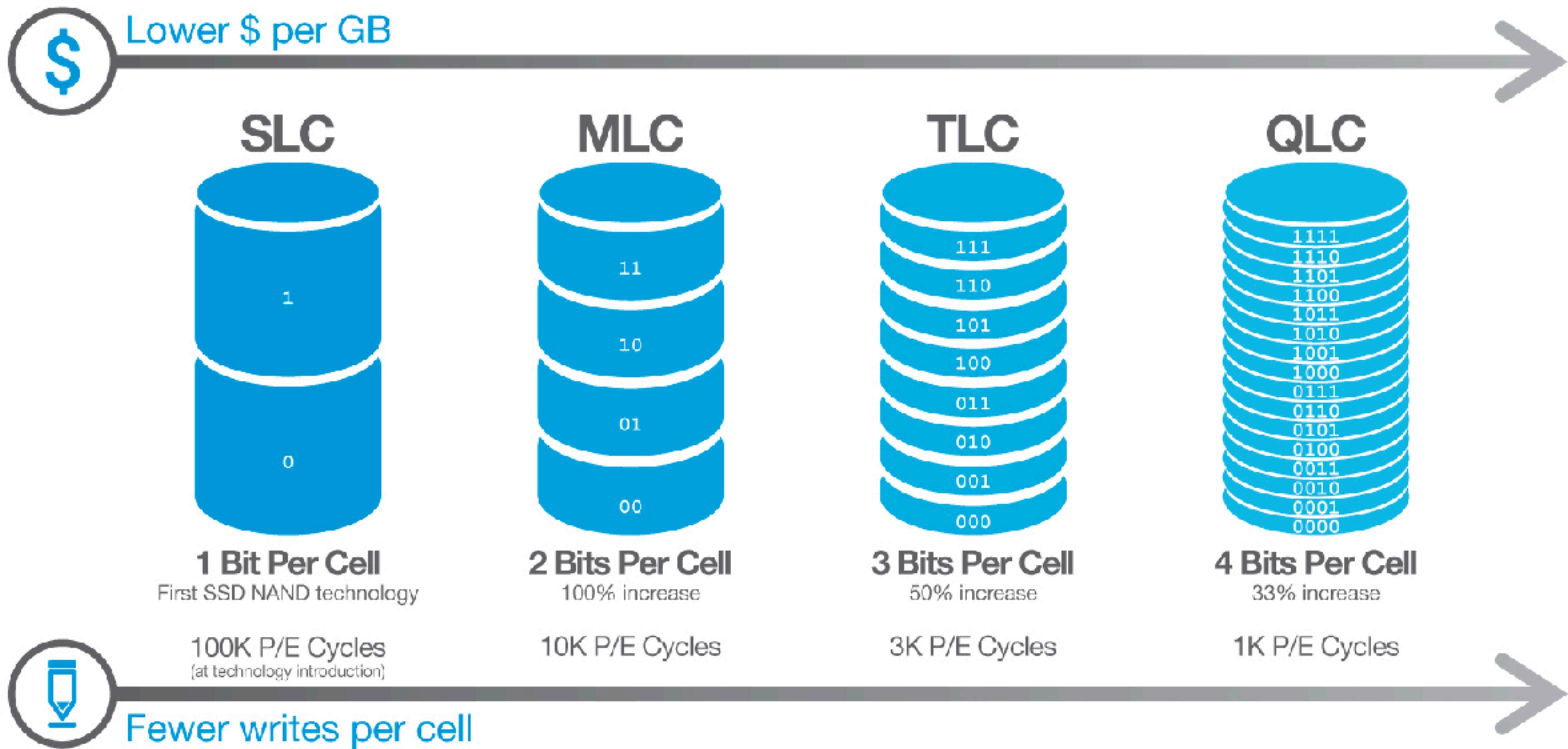
**Triple-Level Cell  
(TLC)**

16 voltage levels,  
4-bit

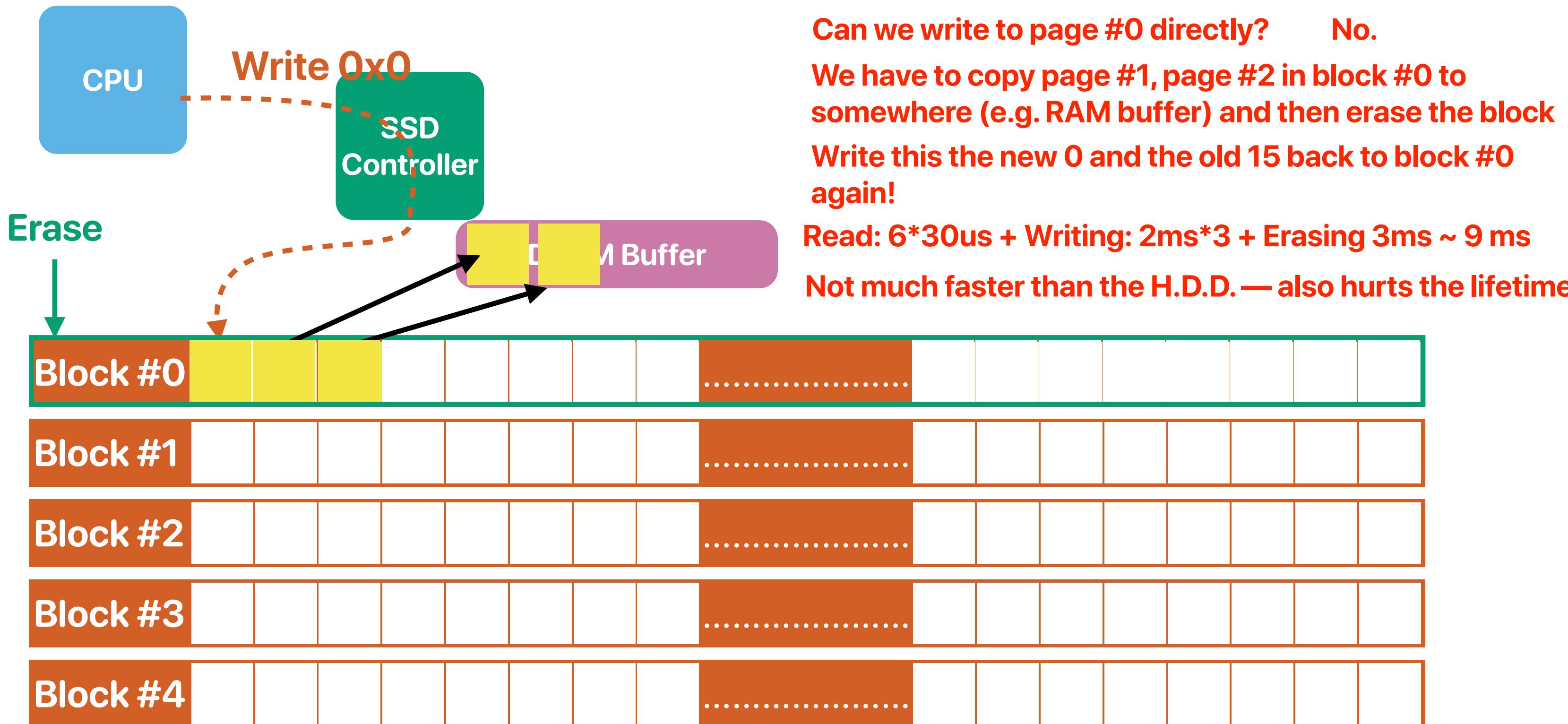


**Quad-Level Cell  
(QLC)**

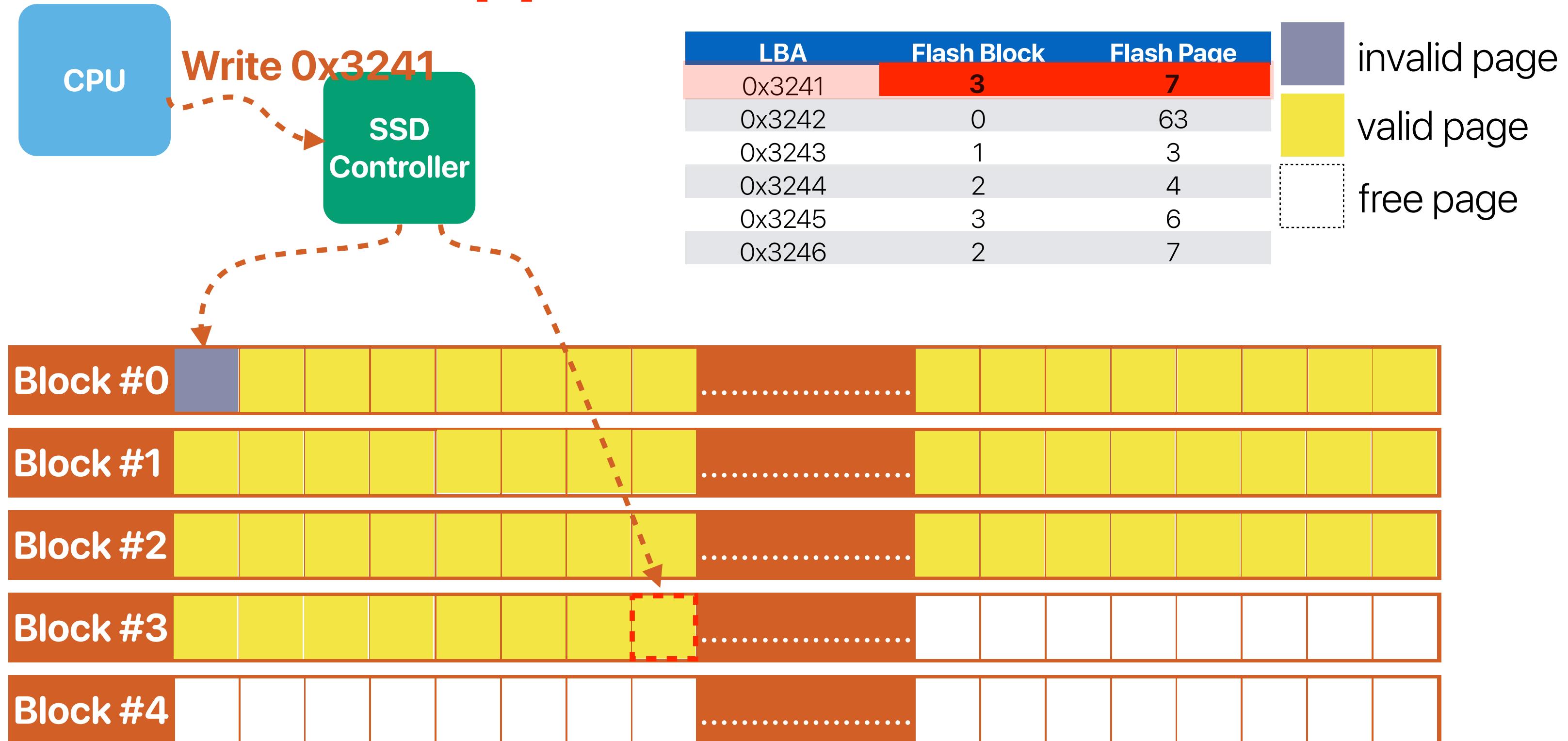
# QLC = More Density Per NAND Cell



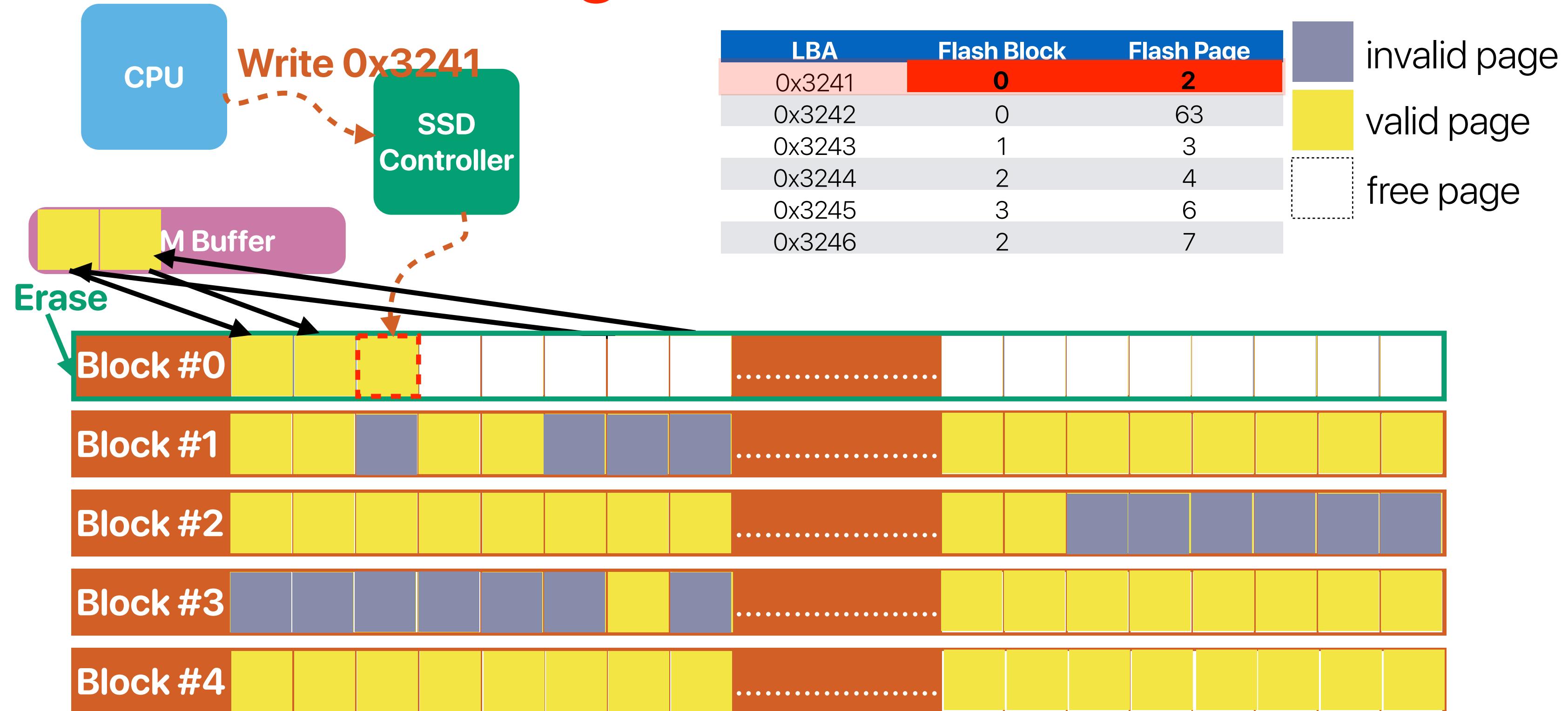
# What happens on a write if we use the same abstractions as H.D.D.



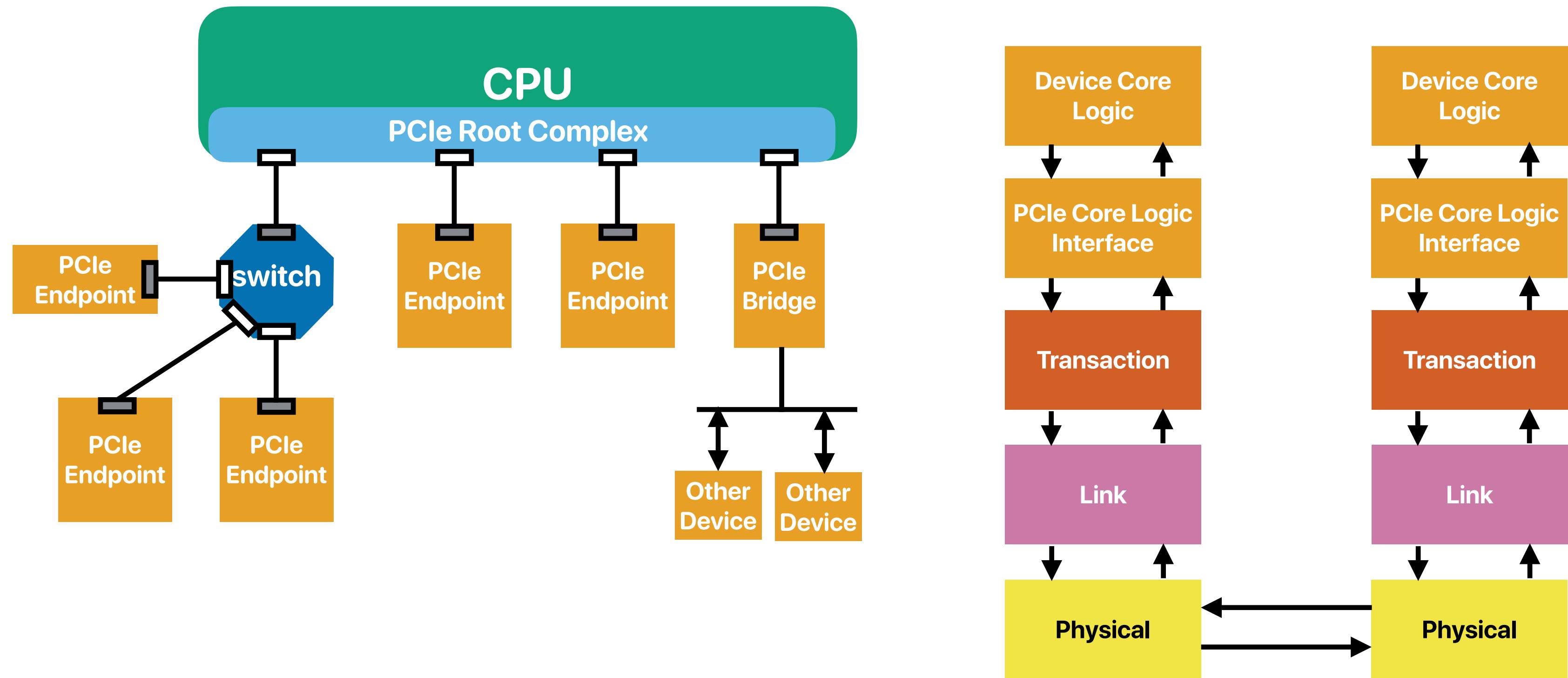
# What happens on a write with FTL



# Garbage Collection in FTL

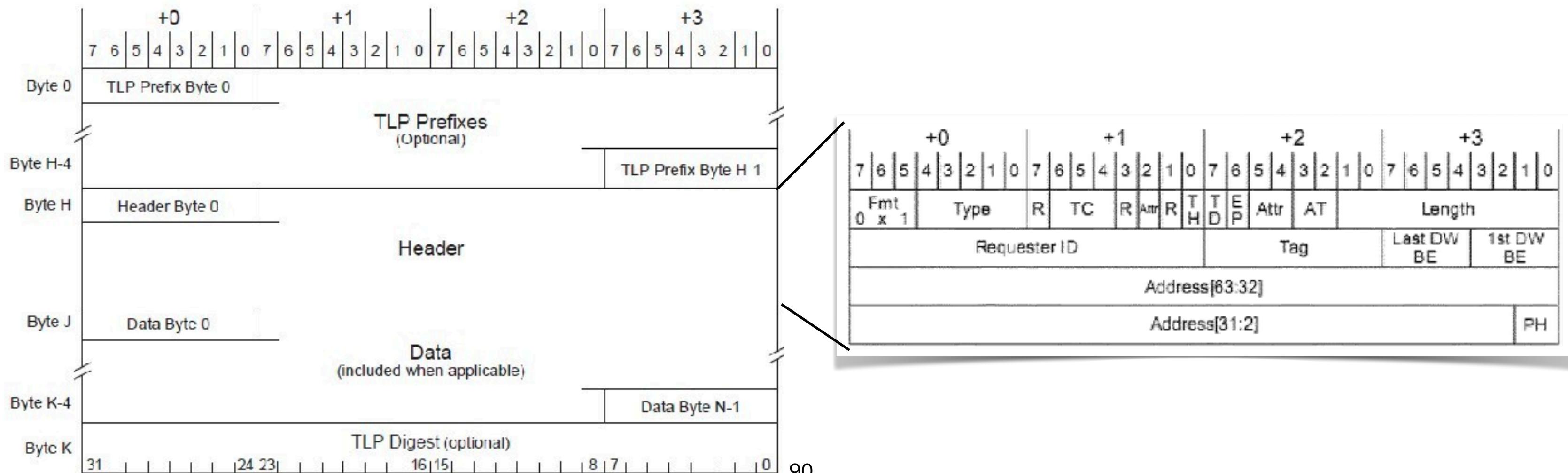


# PCIe “Interconnect”



# PCIe interconnect

- Very similar to computer networks
  - Use “memory addresses” as the identifier for routing
  - Peer-to-peer communication is possible



# NVMe

- The standard of PCIe SSD devices now
  - Provides multiple command queues to better support multithreading hardware
  - Allows more parallelism inside the SSD

0. The “payload” of a PCIe packet

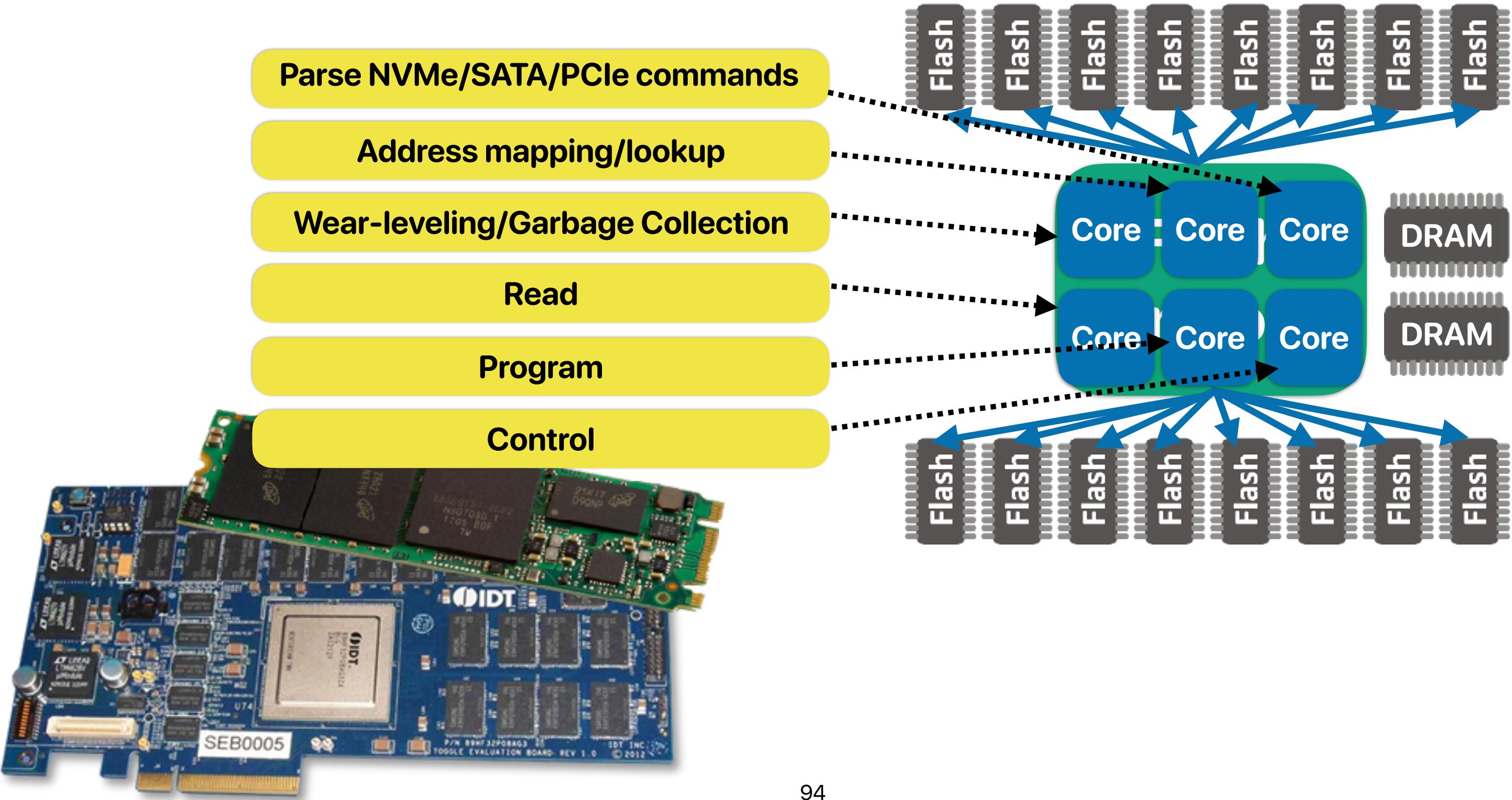
OPCODE	FLAGS	command id	Namespace ID	48	63
reserved					
metadata					
PRP1					
PRP2					
Start LBA					
length	control	Dataset management			
Reference tag		App tag		App mask	

# What is Firmware?

# Firmware is ...

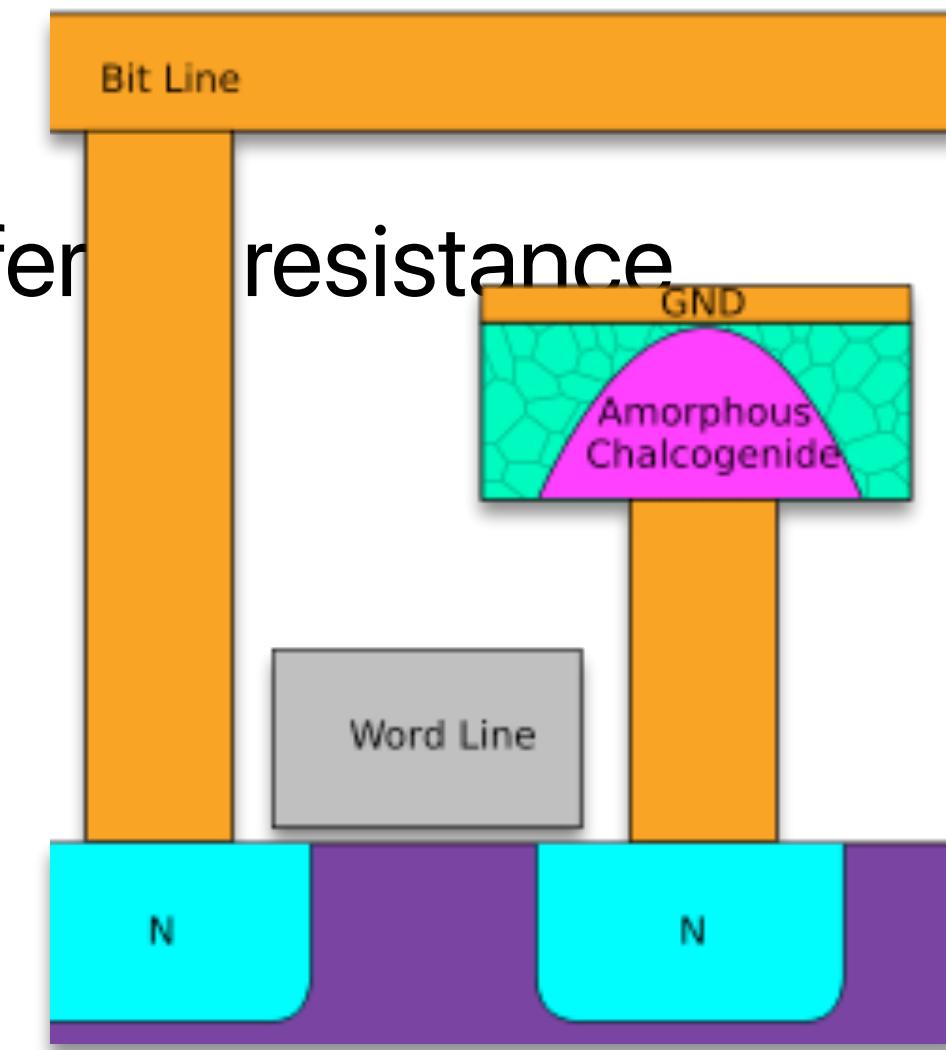
- A software program directly interacts with hardware without an operating system
- Typically a program works in between hardware and application “software”

# SSDs already have processors inside



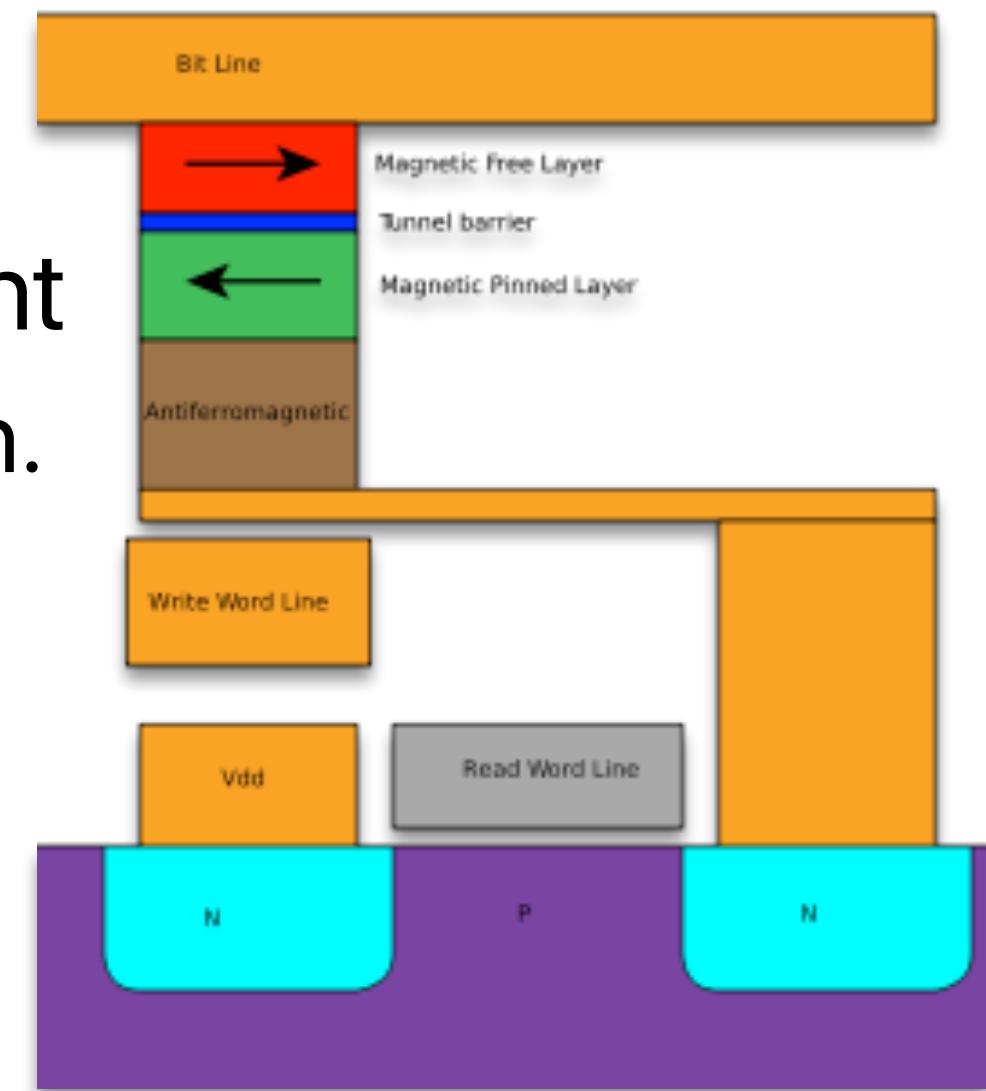
# Phase change memory

- The bit is stored in the crystal structure of a tiny spec of metal.
- To write, it melts the metal (650C)
  - let it cool quickly or slowly to set the value
  - Crystaline and amorphous states have different resistance



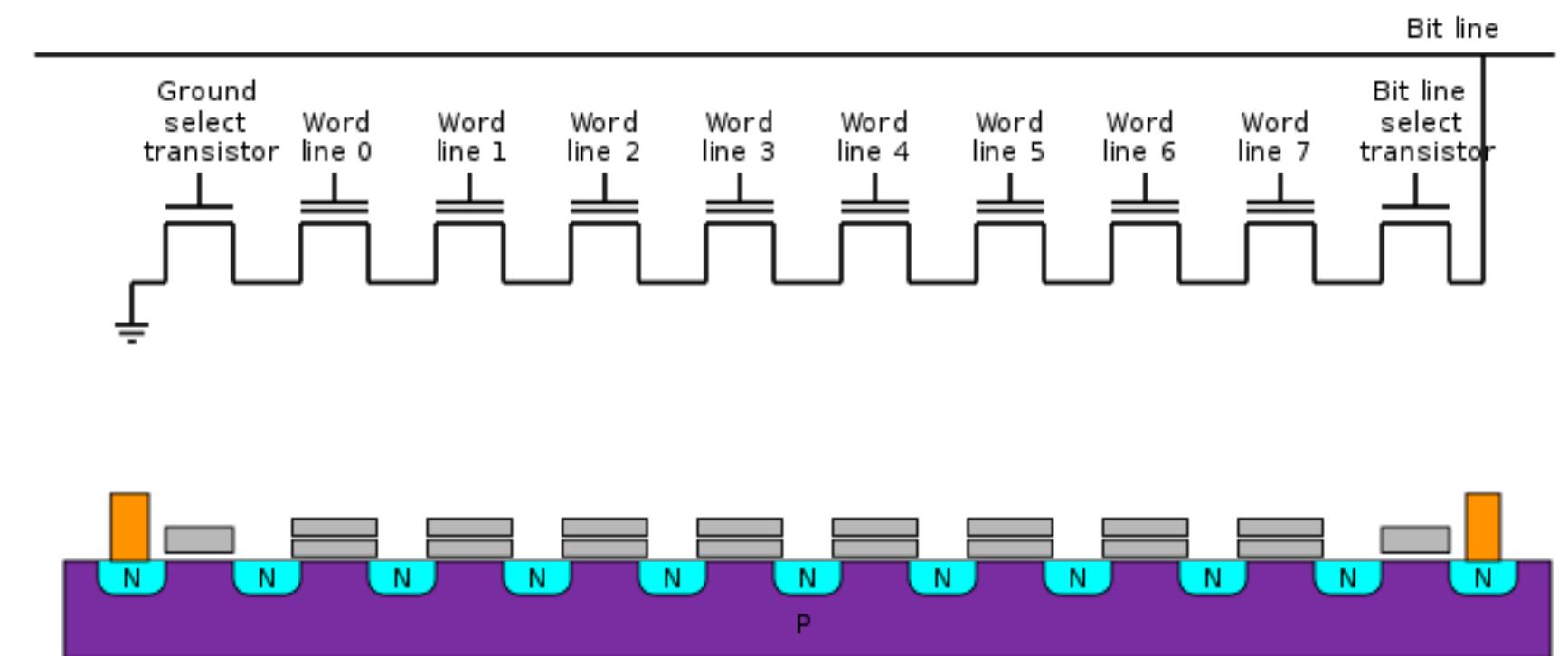
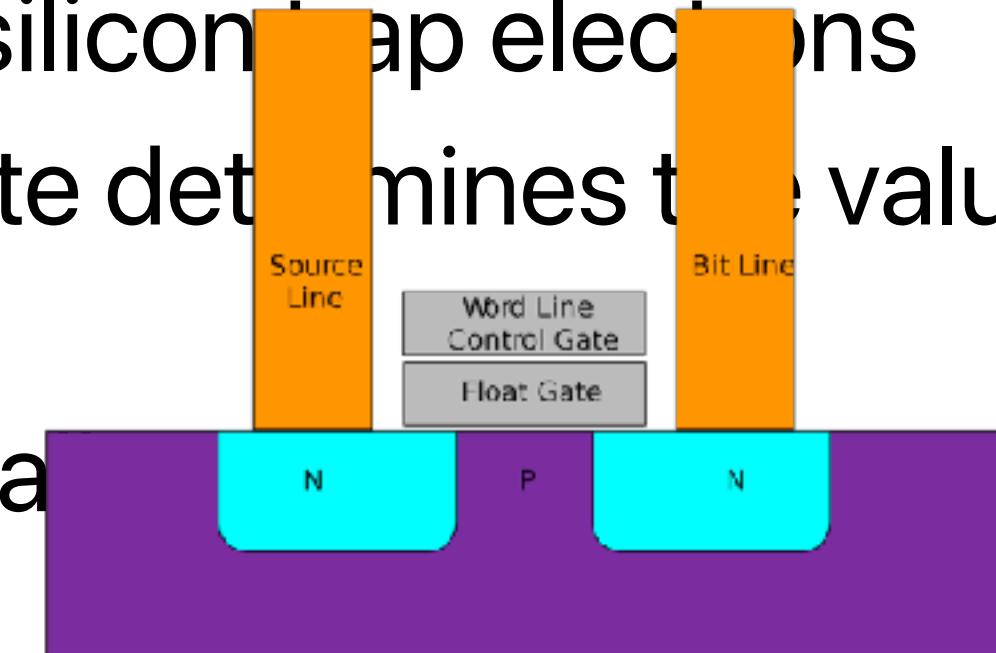
# Spin-torque transfer

- Bits stored as magnetic orientation of a thin film
- Change the state using polarized electrons (!)
- Depending on polarization, resistance differs
- More complex cell structure
- Great promise — potential DRAM replacement
  - Roughly the same speed, power, and bandwidth.
  - But it's durable!

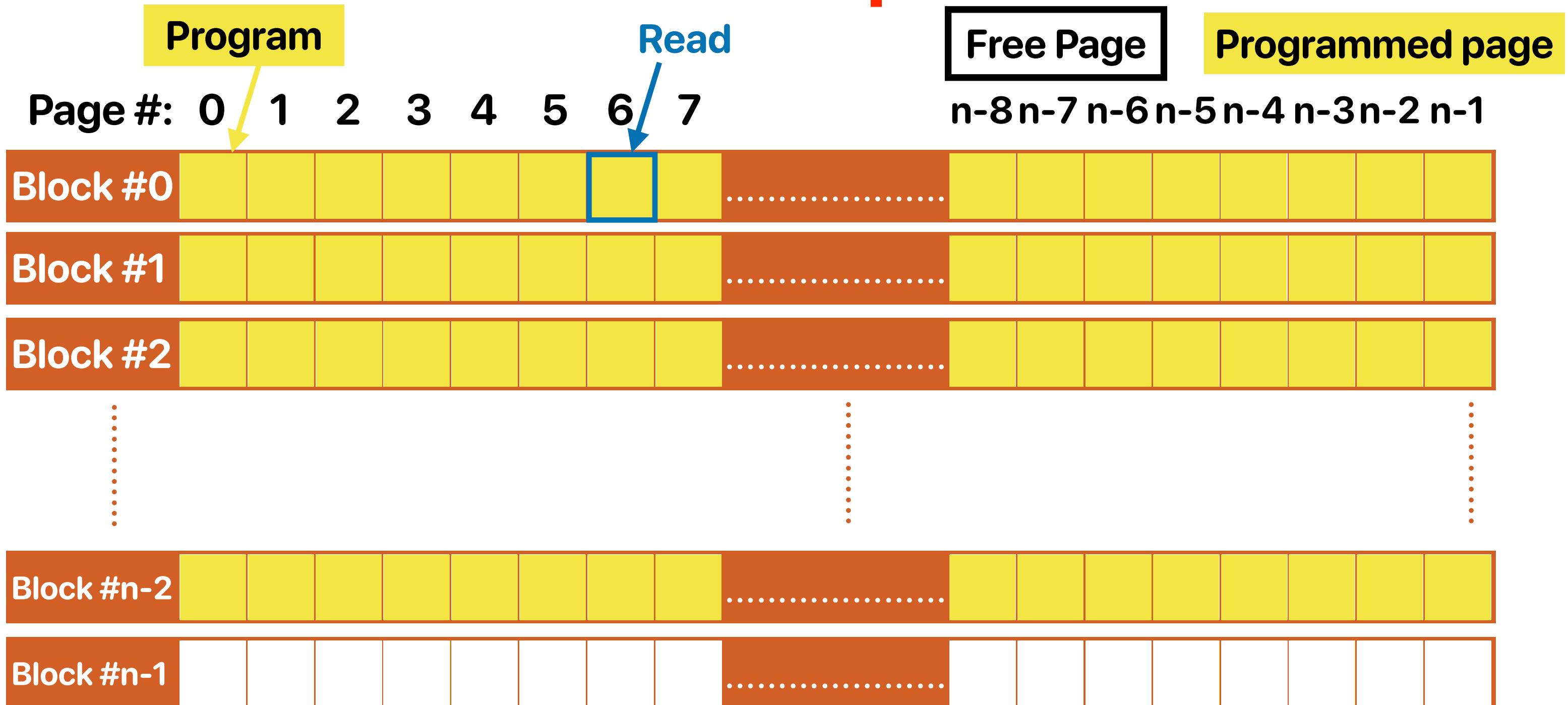


# Flash memory

- Floating gate made by polycrystalline silicon trap electrons
- The voltage level within the floating gate determines the value of the cell
- The floating gates will wear out eventually



# Basic flash operations



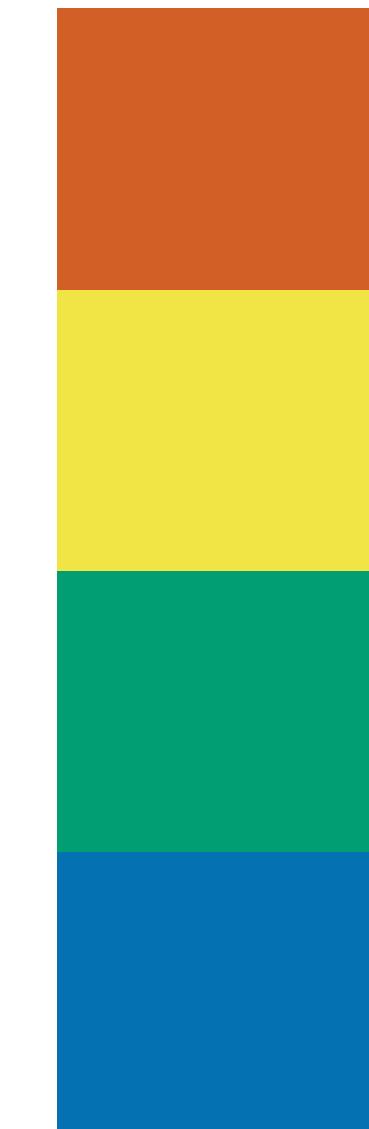
# Types of Flash Chips

2 voltage levels,  
1-bit



**Single-Level Cell  
(SLC)**

4 voltage levels,  
2-bit



**Multi-Level Cell  
(MLC)**

8 voltage levels,  
3-bit



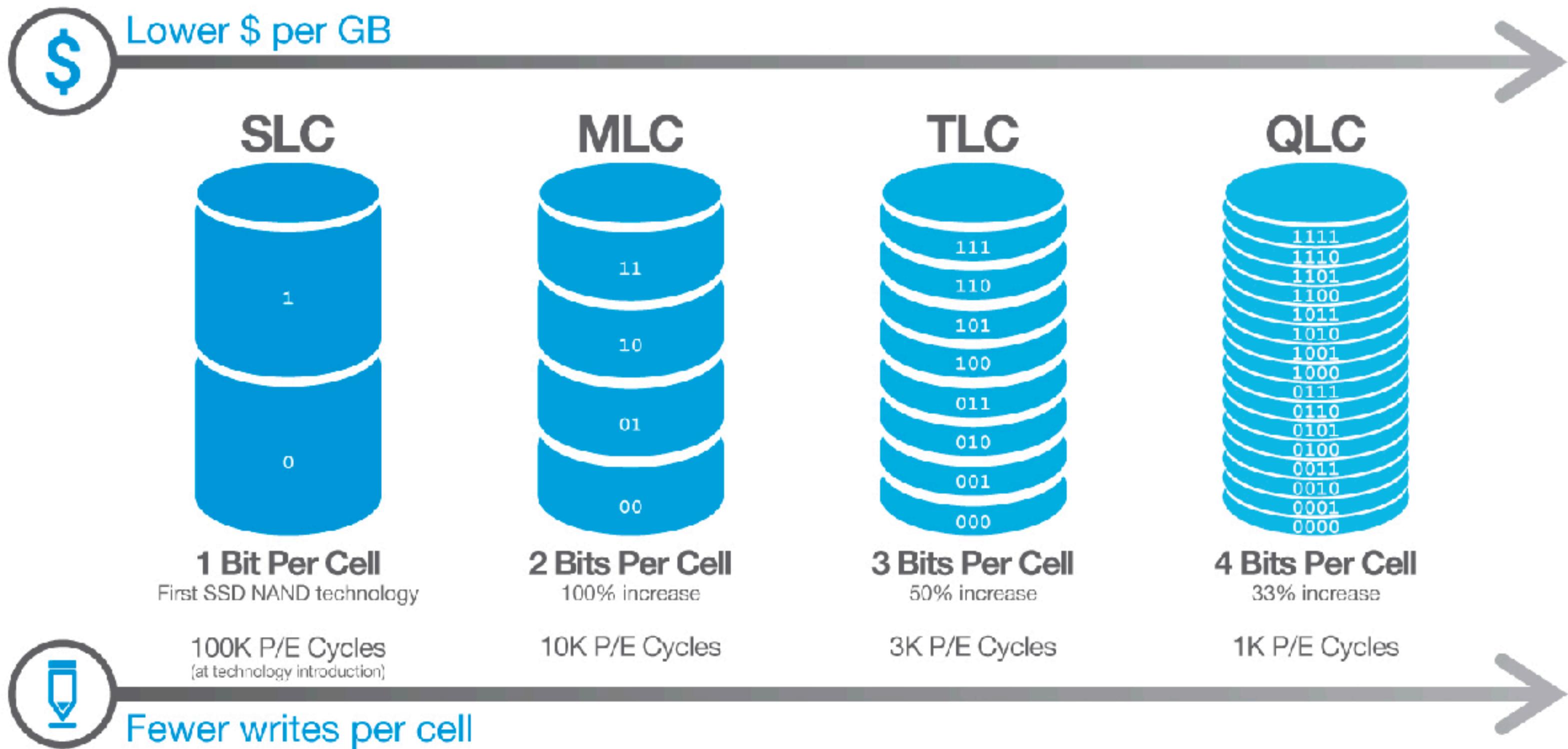
**Triple-Level Cell  
(TLC)**

16 voltage levels,  
4-bit



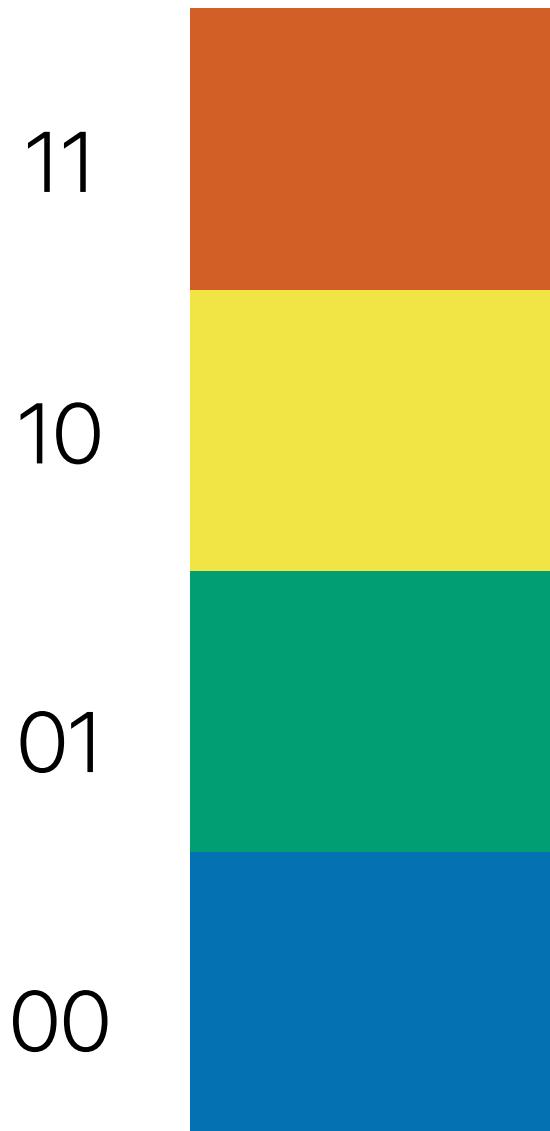
**Quad-Level Cell  
(QLC)**

# QLC = More Density Per NAND Cell



# Programming in MLC

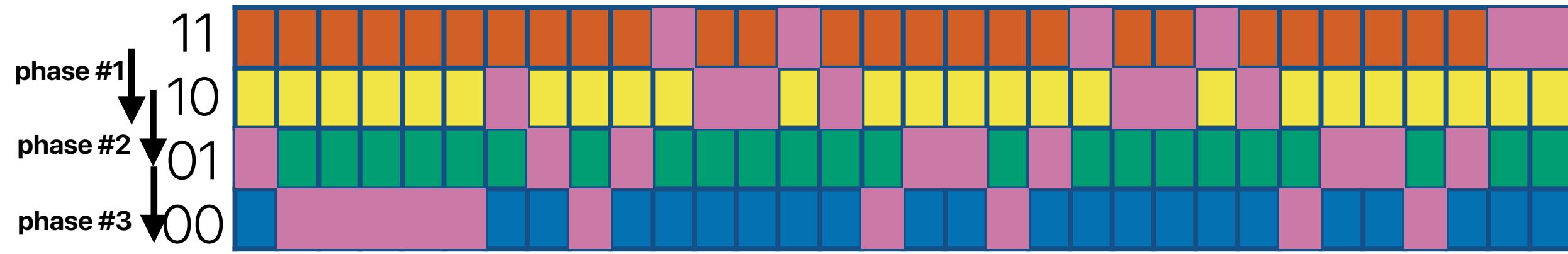
4 voltage levels,  
2-bit



**3.1400000000000001243449787580**

= **0x40091EB851EB851F**

= **01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111**

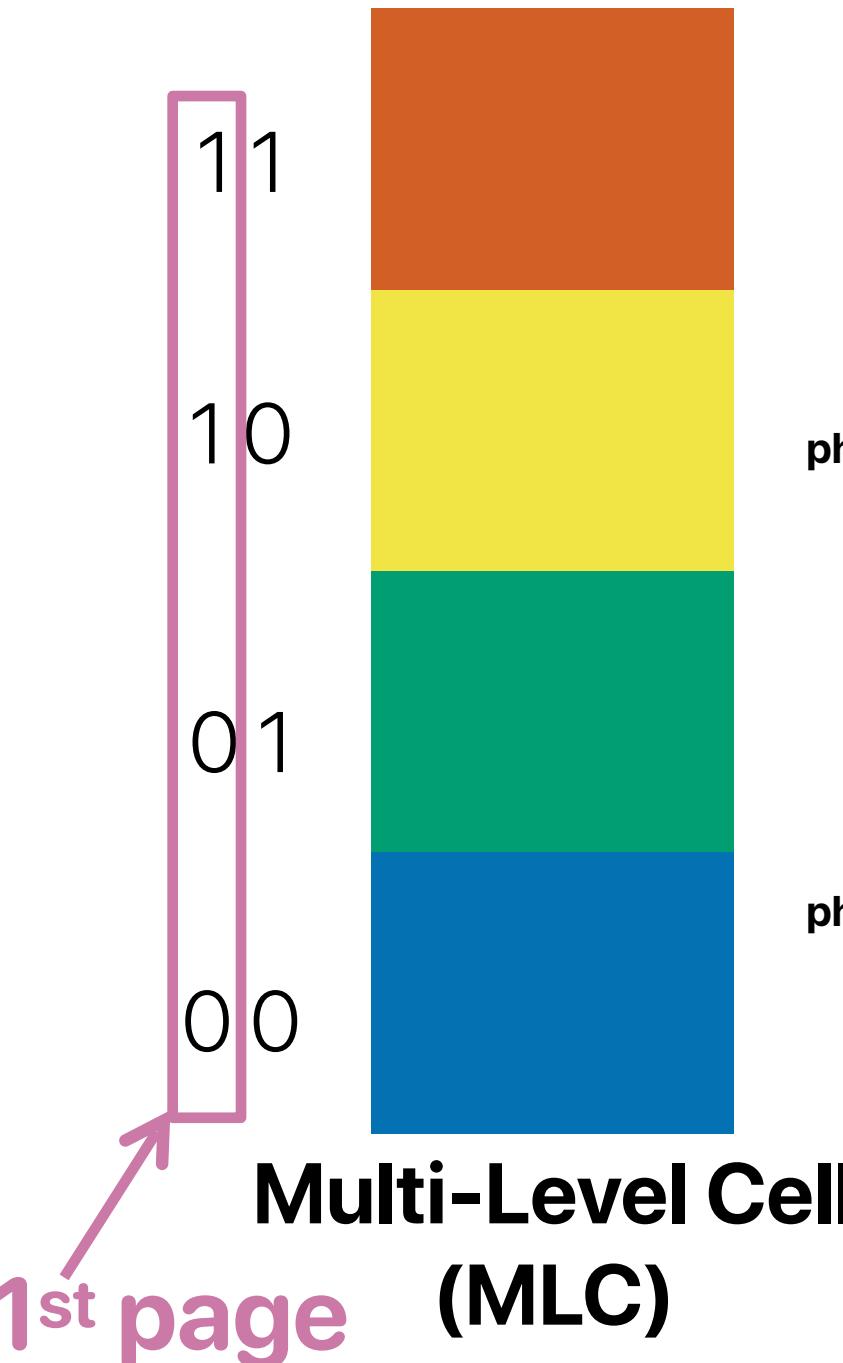


**3 Cycles/Phases to finish programming**

**Multi-Level Cell  
(MLC)**

# Programming in MLC

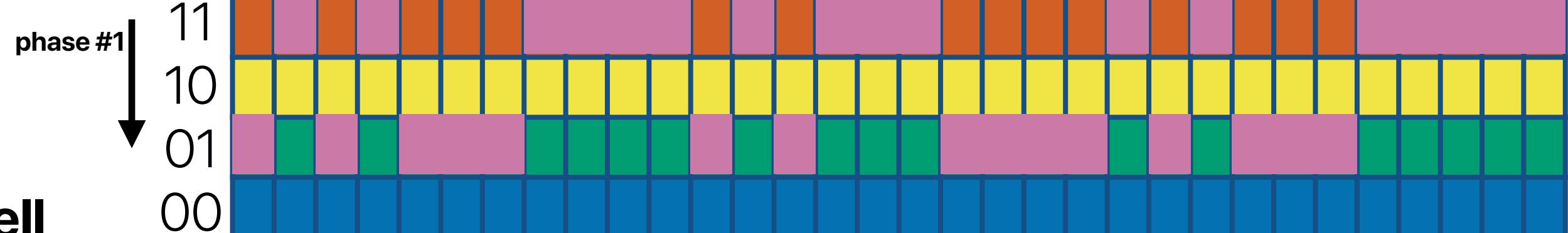
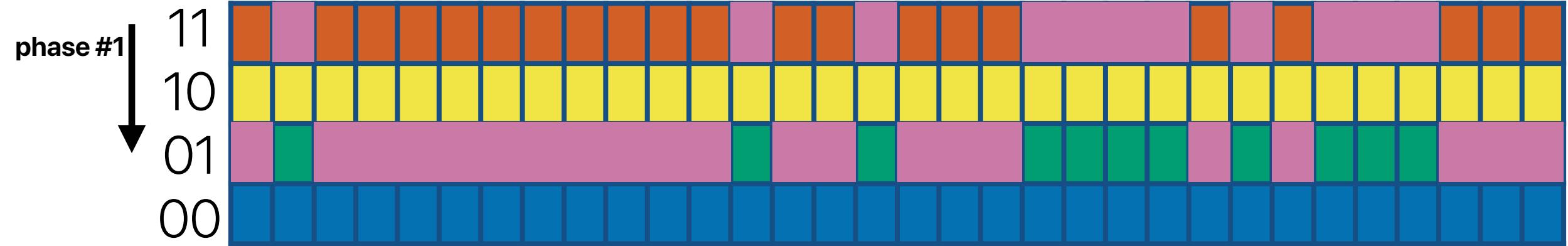
4 voltage levels,  
2-bit



**3.1400000000000001243449787580**

= **0x40091EB851EB851F**

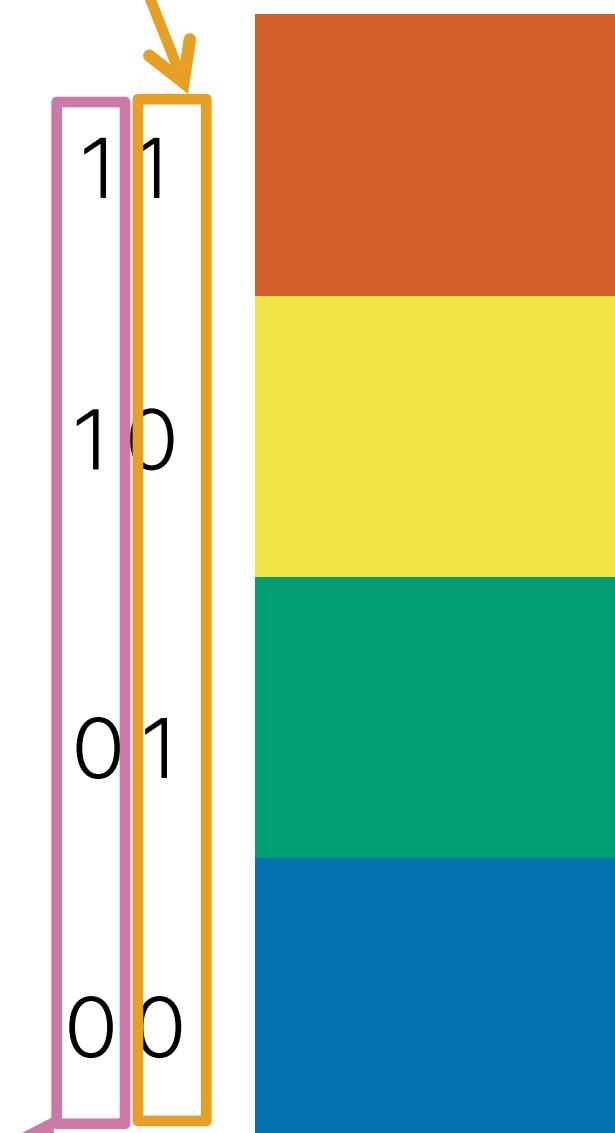
= **01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111**



**1 Phase to finish programming the first page!**

# Programming the 2nd page in MLC

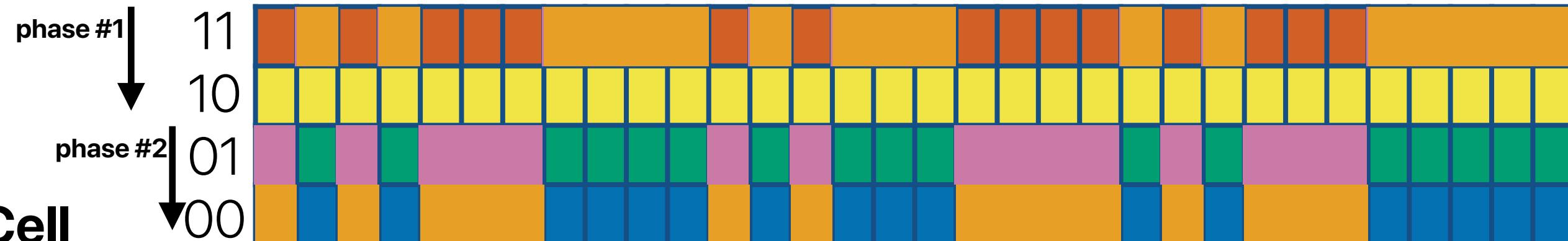
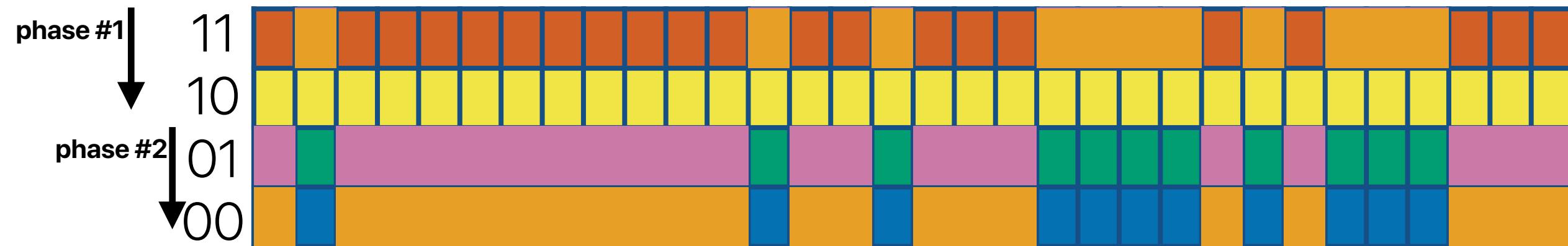
2<sup>nd</sup> page  
4 voltage levels,  
2-bit



3.1400000000000001243449787580

= 0x40091EB851EB851F

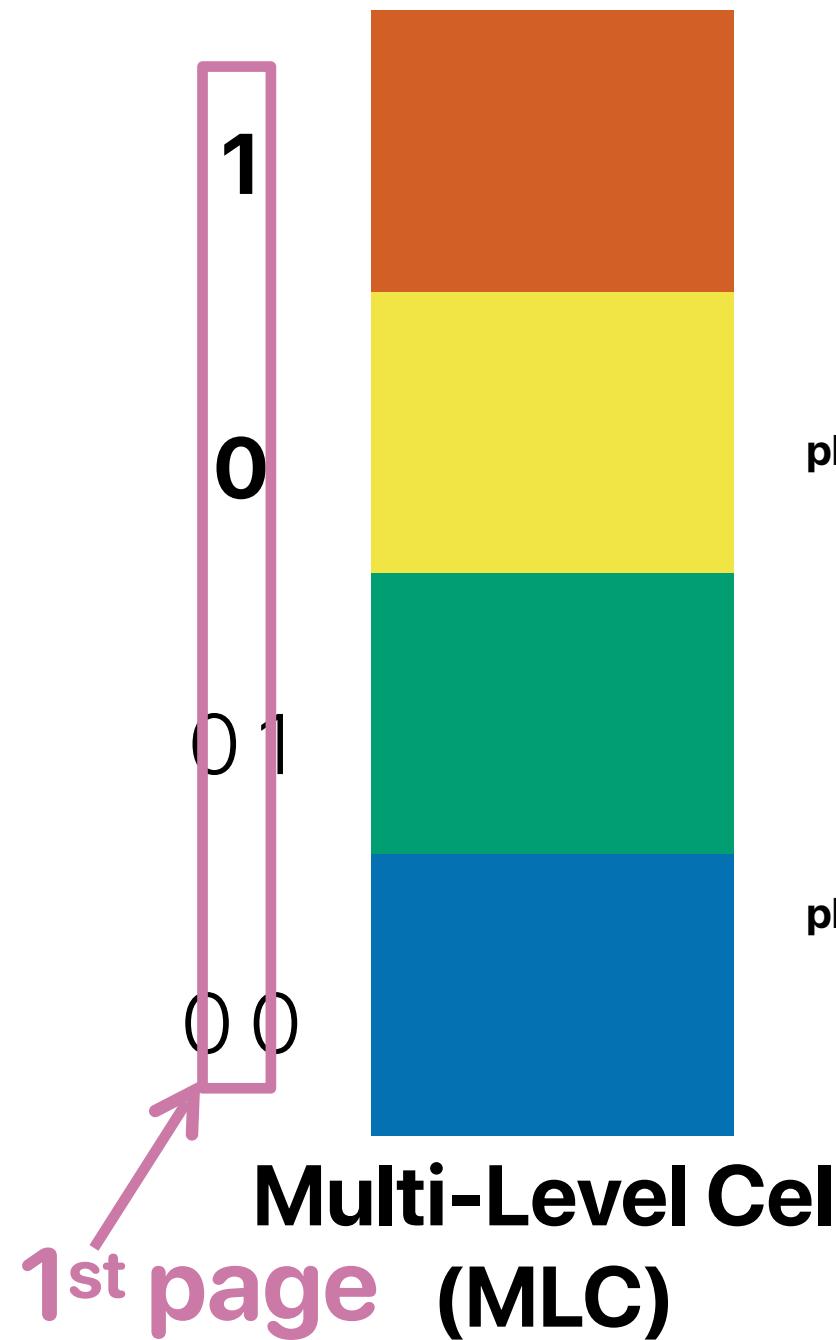
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111  
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



2 Phase to finish programming the second page!

# Optimizing 1st Page Programming in MLC

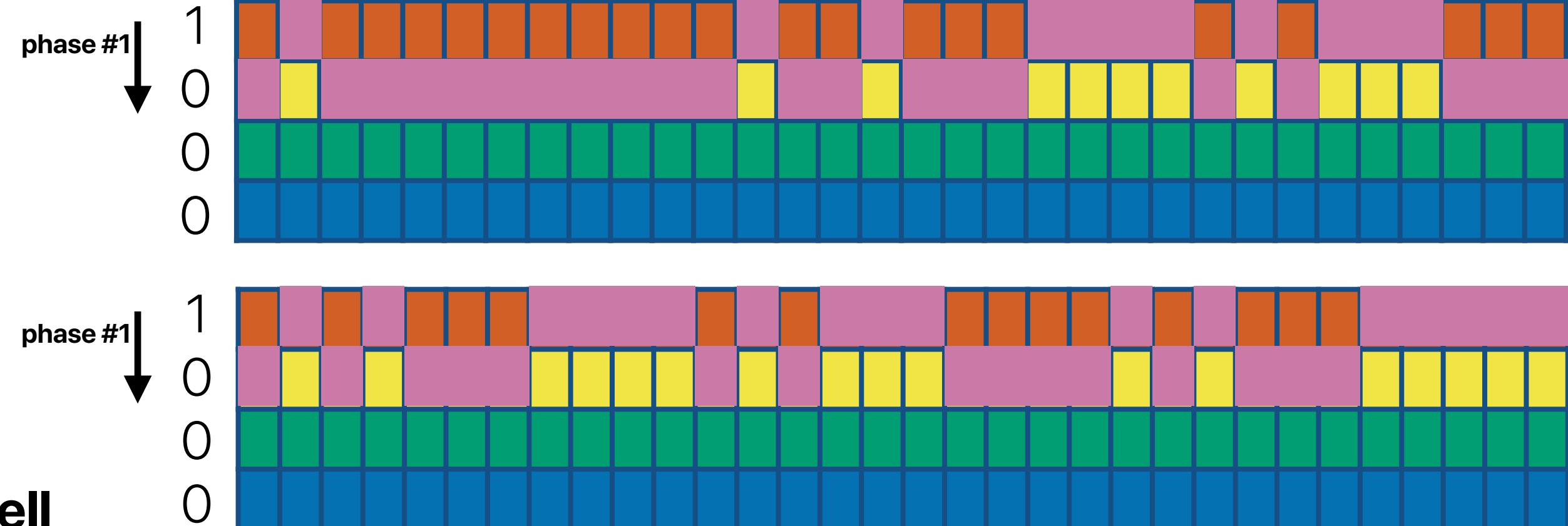
4 voltage levels,  
2-bit



3.1400000000000001243449787580

= 0x40091EB851EB851F

= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111

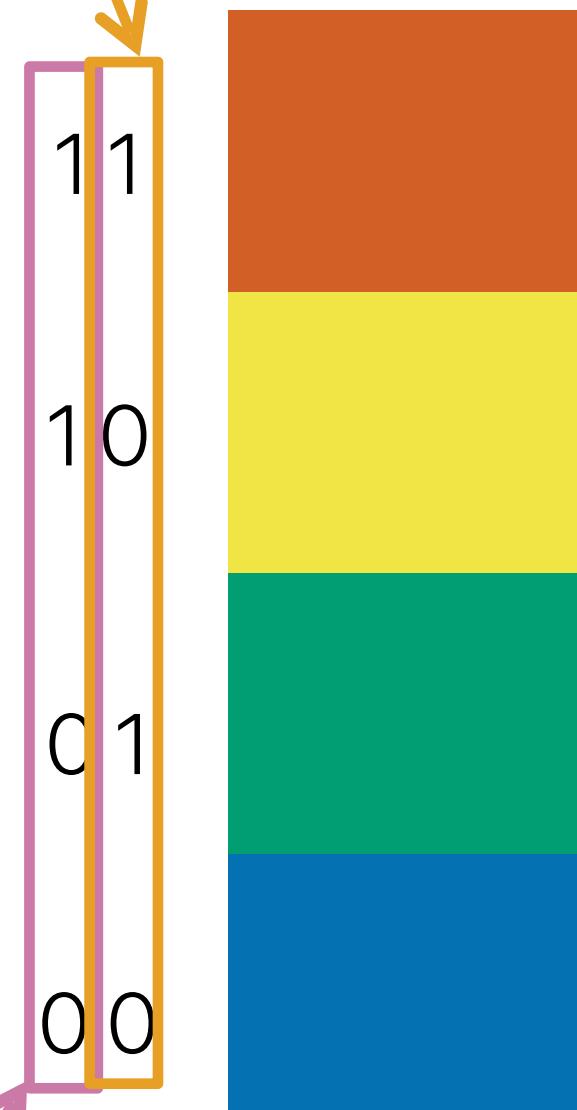


1 Phase to finish programming the first page!

10<sup>4</sup> — the phase is shorter now

# 2nd Page Programming in MLC

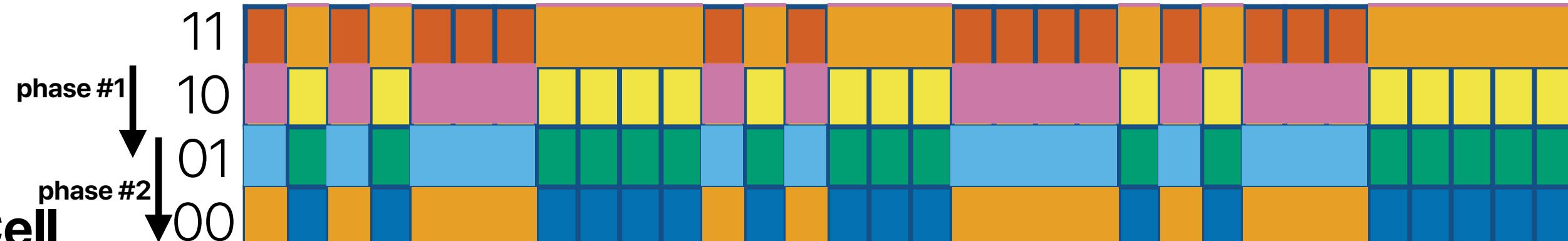
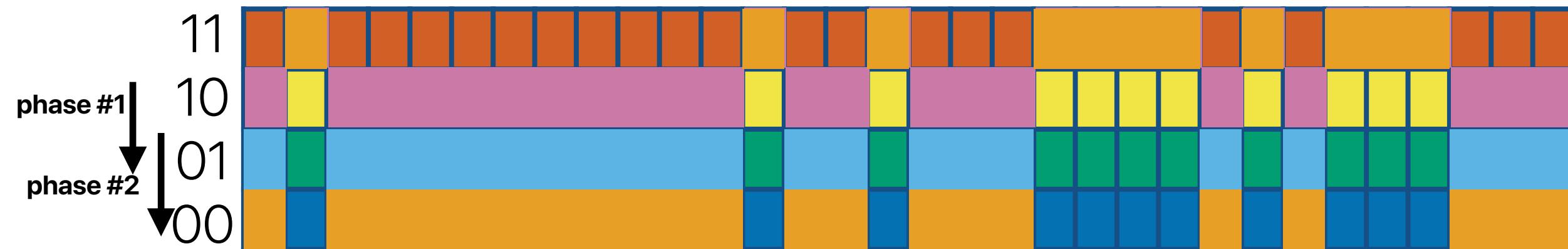
2nd page  
4 voltage levels,  
2-bit



3.1400000000000001243449787580

= 0x40091EB851EB851F

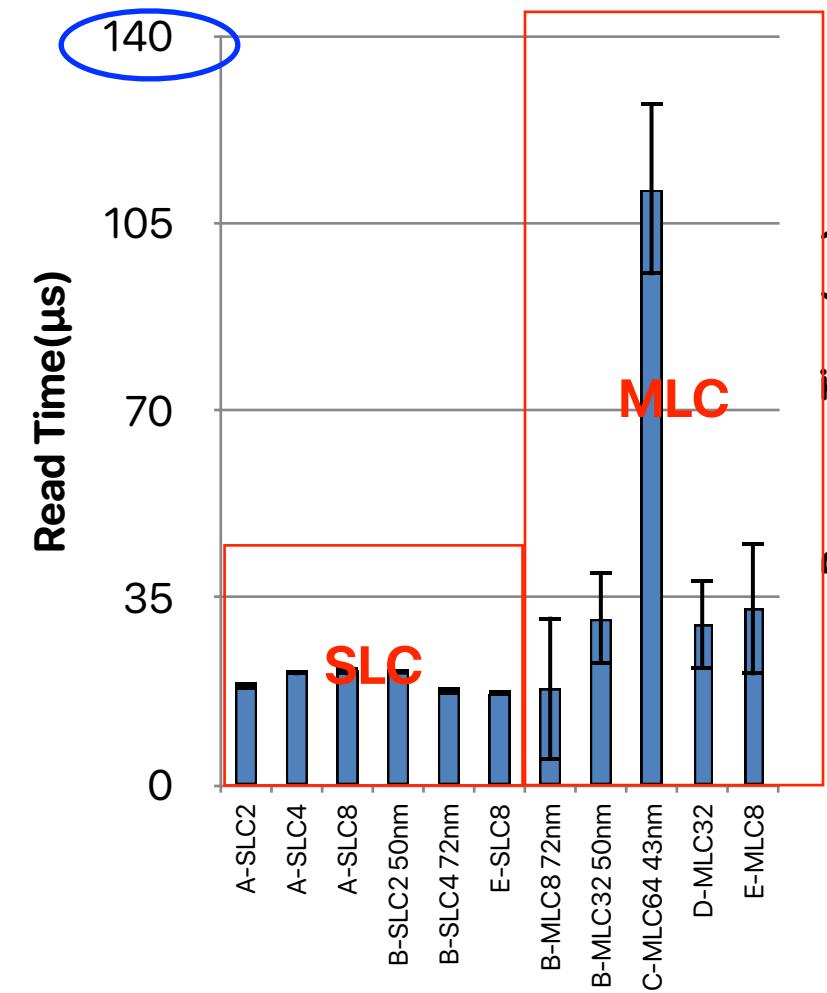
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111  
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



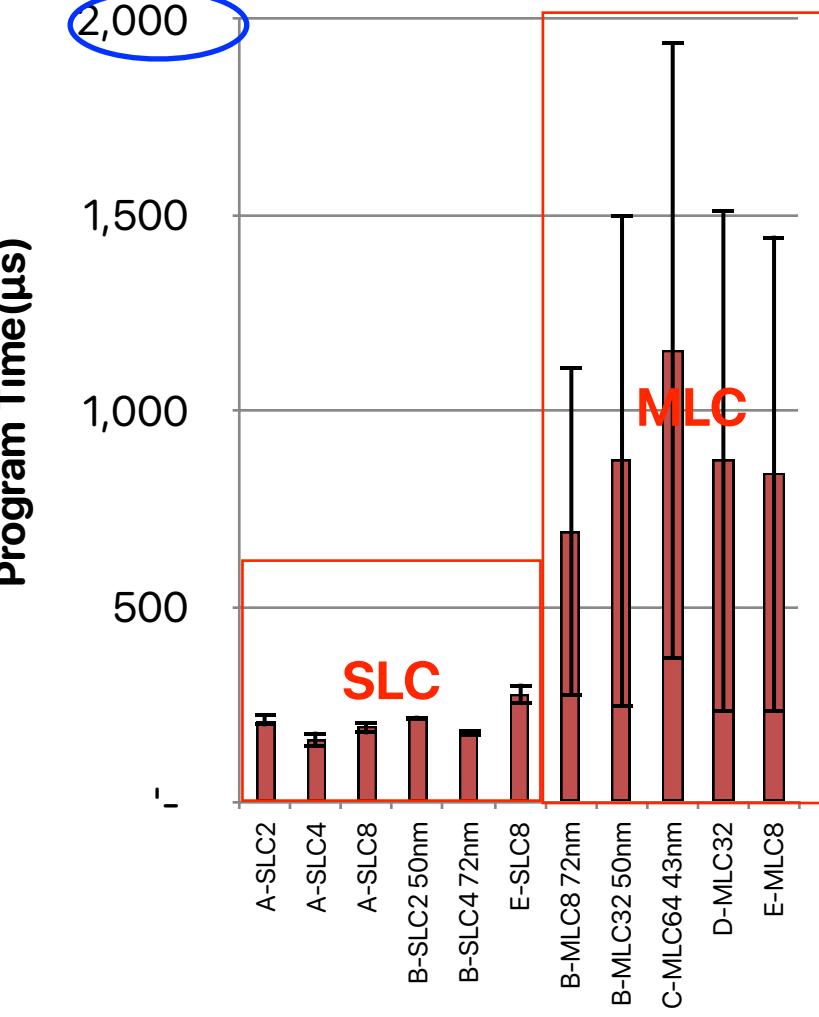
2 Phase to finish programming the second page!

Not a good practice

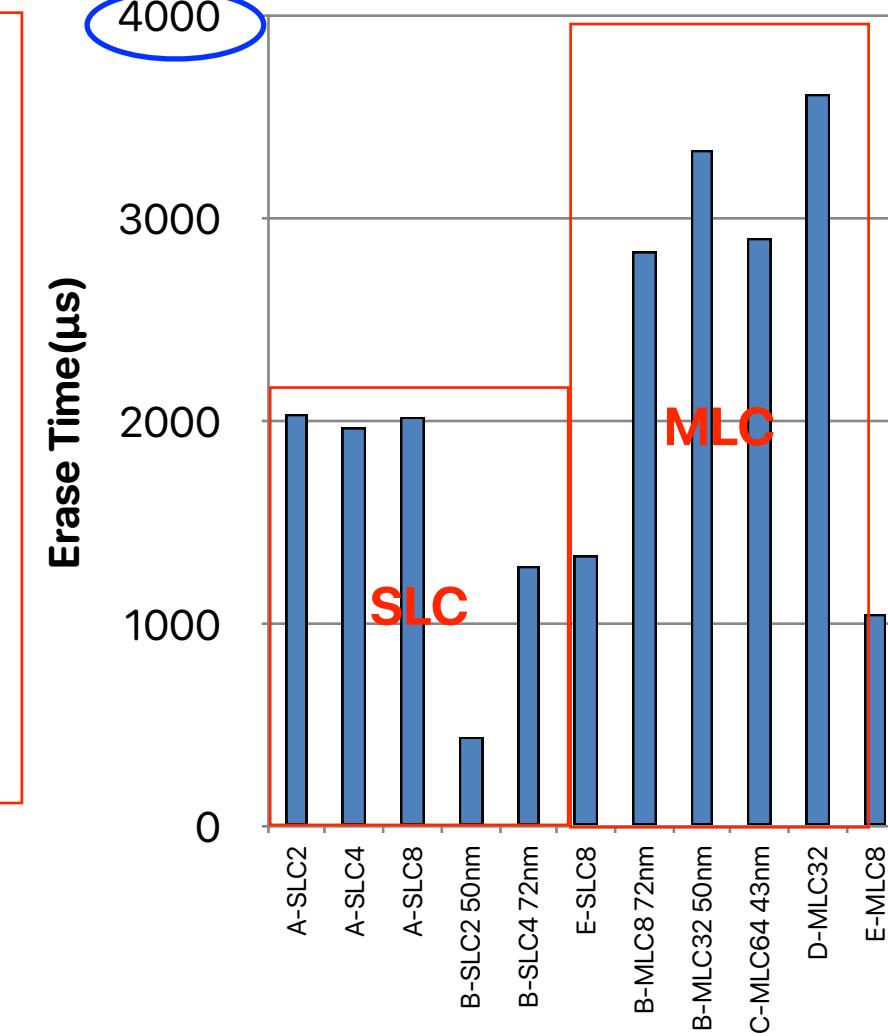
# Flash performance



**Reads:  
less than 150us**



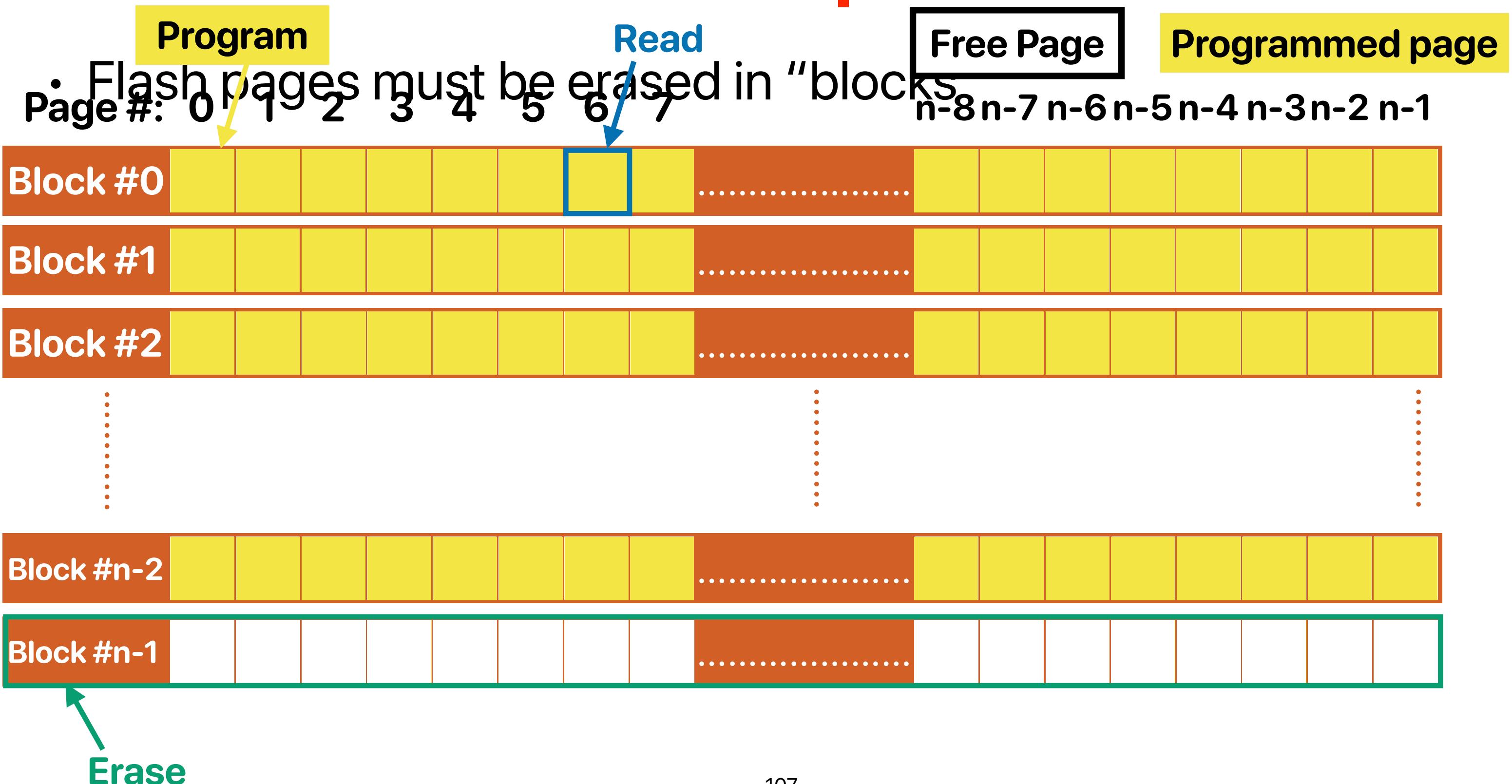
**Program/write:  
less than 2ms**



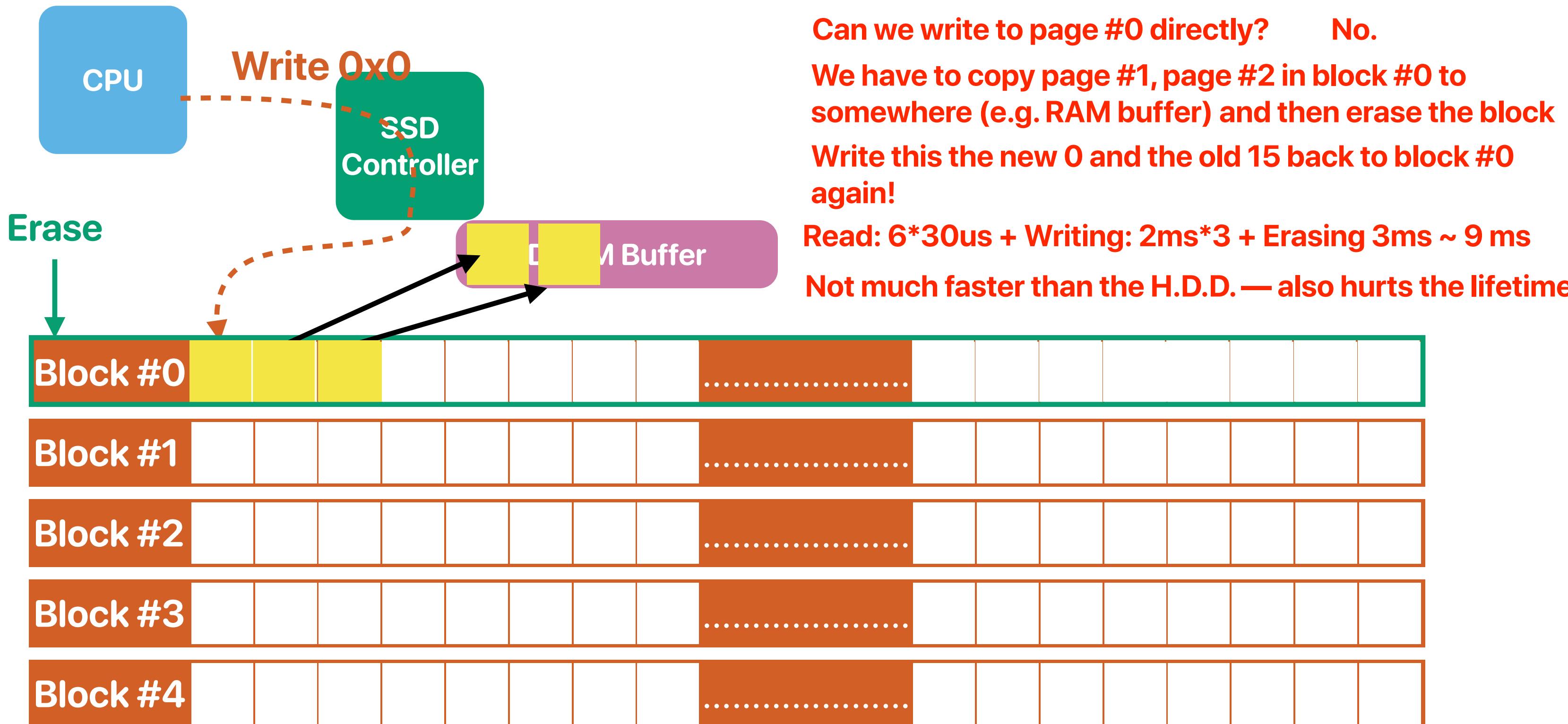
**Erase:  
less than 3.6ms**

**Similar relative performance for reads, writes and erases**

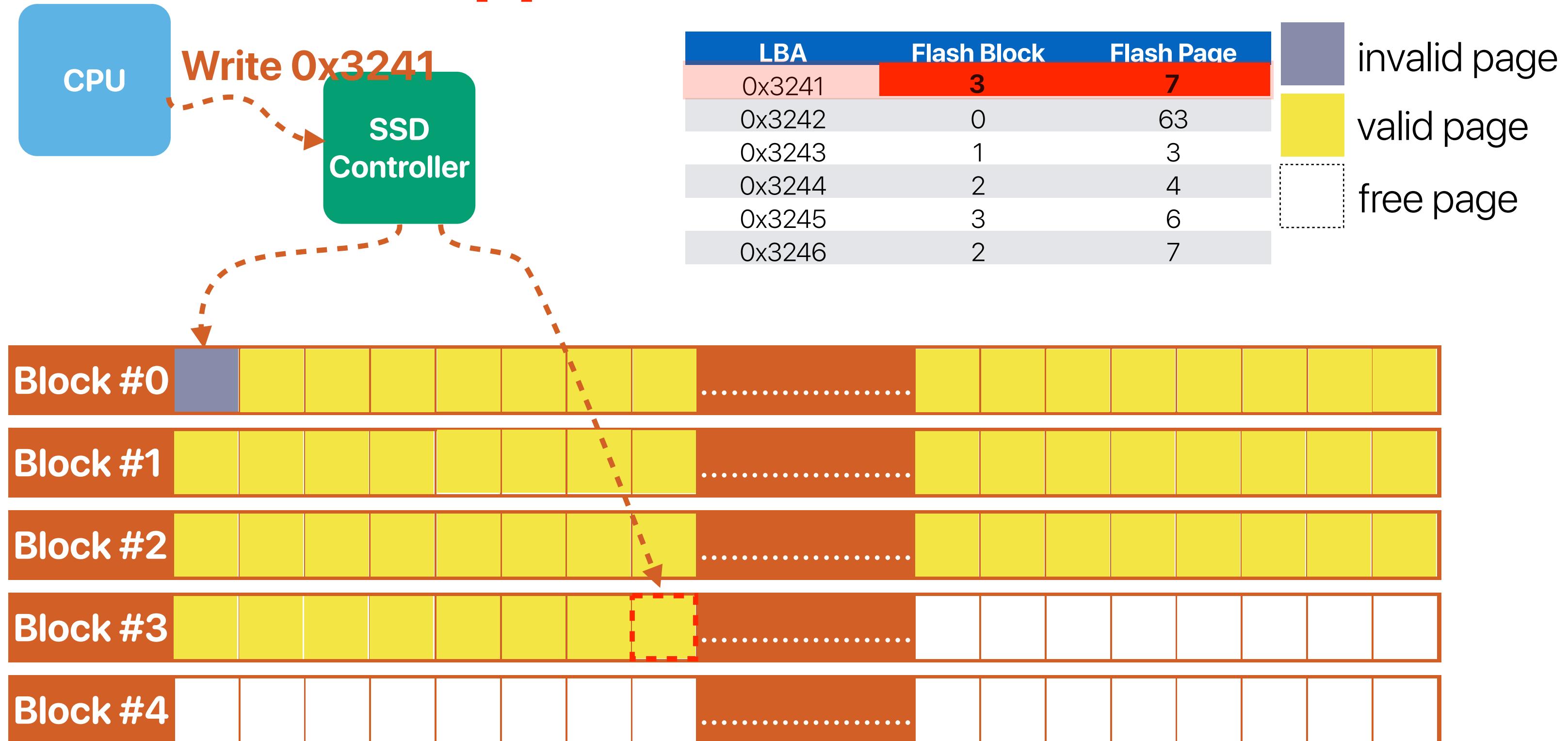
# Basic flash operations



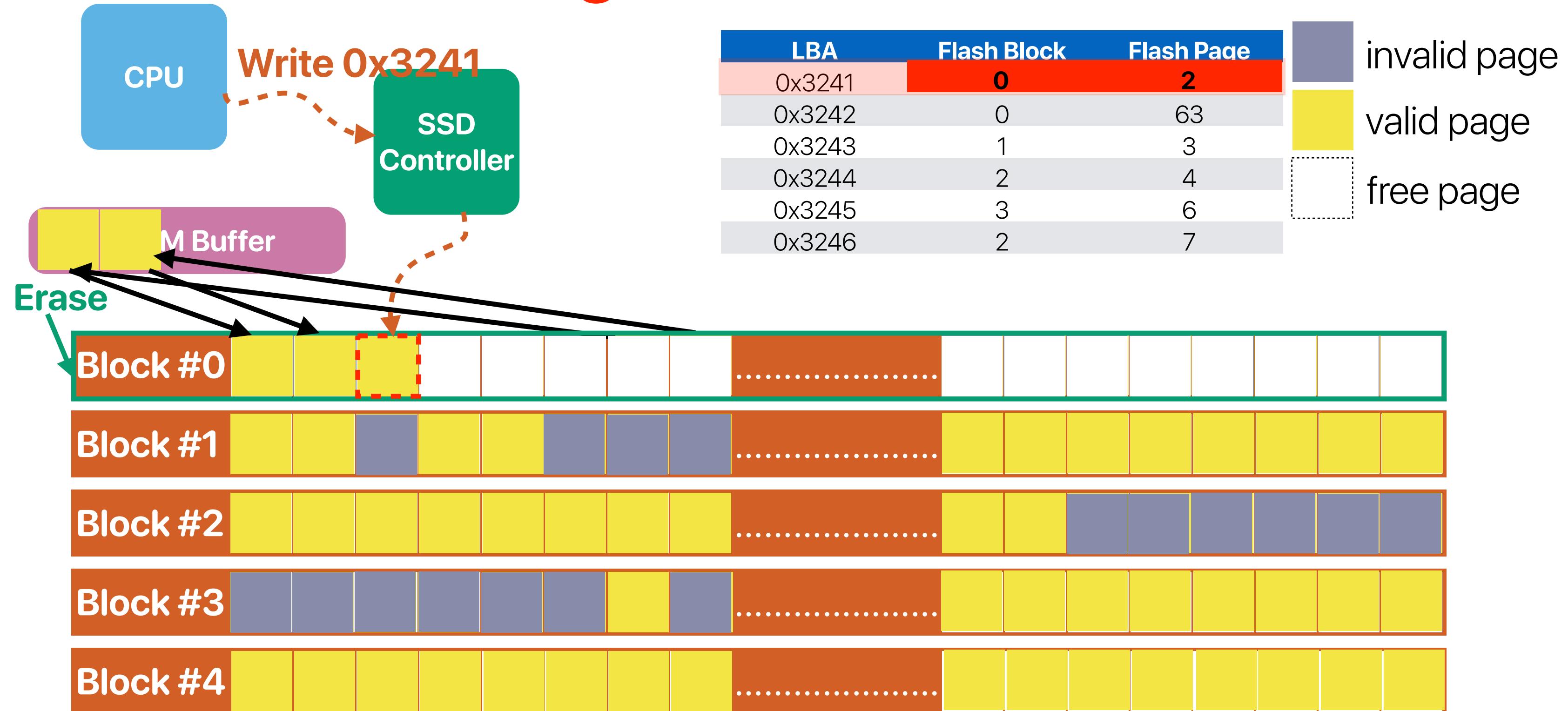
# What happens on a write if we use the same abstractions as H.D.D.



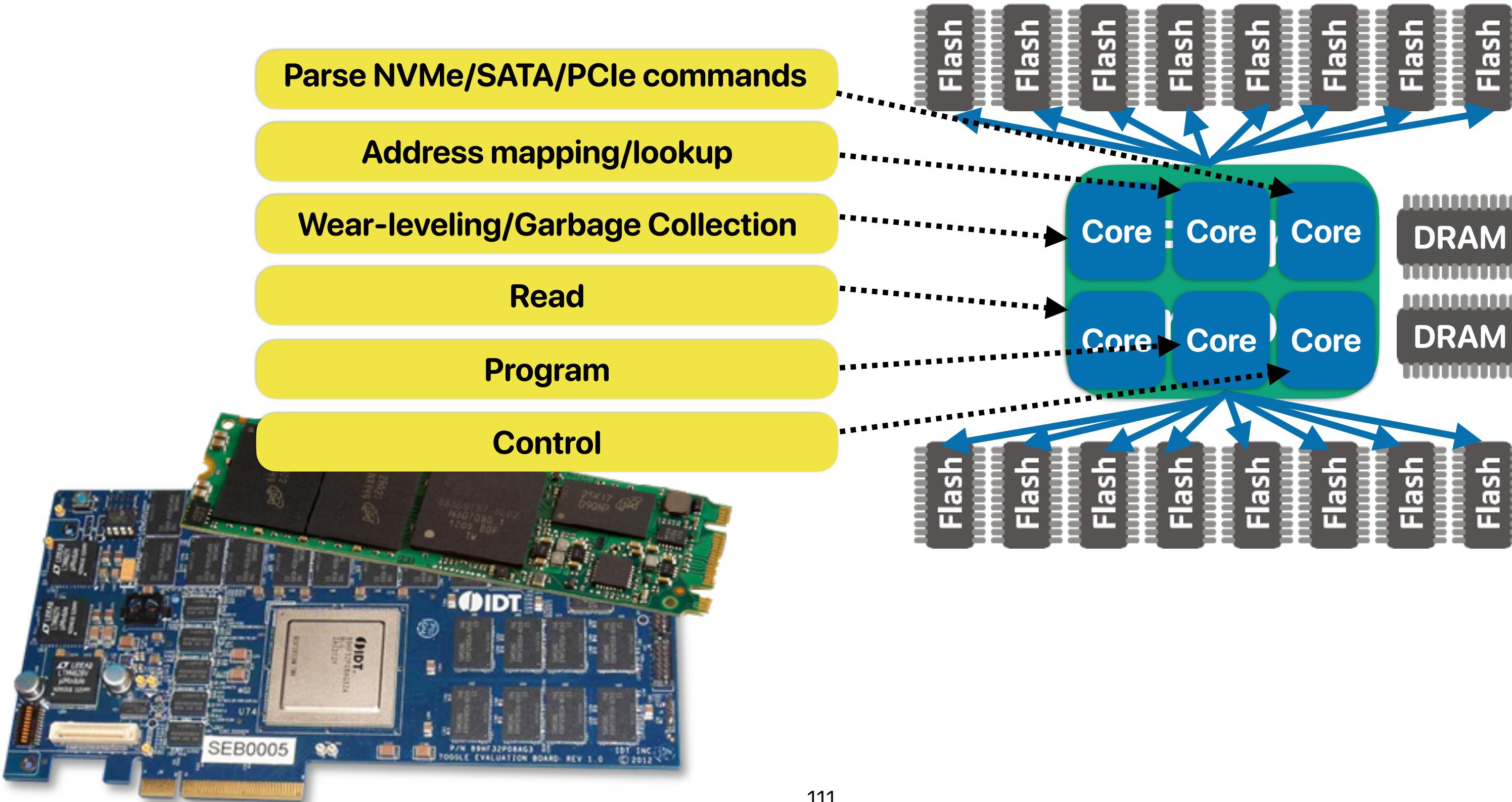
# What happens on a write with FTL



# Garbage Collection in FTL



# SSDs already have processors inside



# Roogle Project Presentations

- Make an appointment on 2/6 and 2/8 through the Google Calendar **ASAP**
- 12 minute presentation with 3 minute Q & A
- Why & what & how!!! — considering you're giving a presentation at Apple's keynote
  - 6-minute why — why should everyone care about this problem? Why is this still a problem?
  - 4-minute what — what are you proposing in this project to address the problem?
  - 2-minute how — expected platforms/engineering efforts, milestones and workload distribution among members
  - Please reference this article to make a good presentation <https://cseweb.ucsd.edu/~swanson/GivingTalks.html>

# Electrical Computer Science Engineering

277

つづく



# NVIDIA Grace Hopper

## NVIDIA GH200 Grace Hopper Superchip

