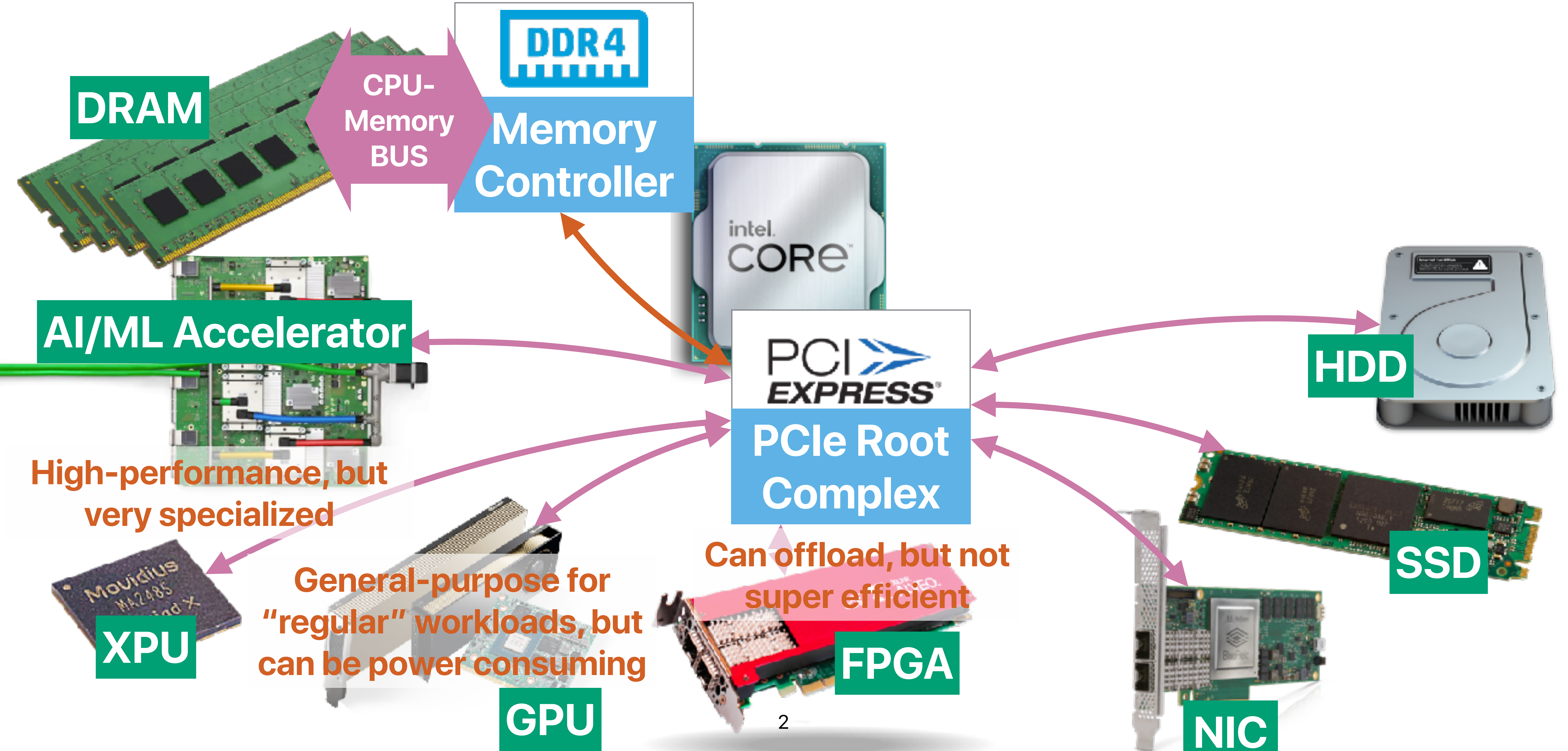
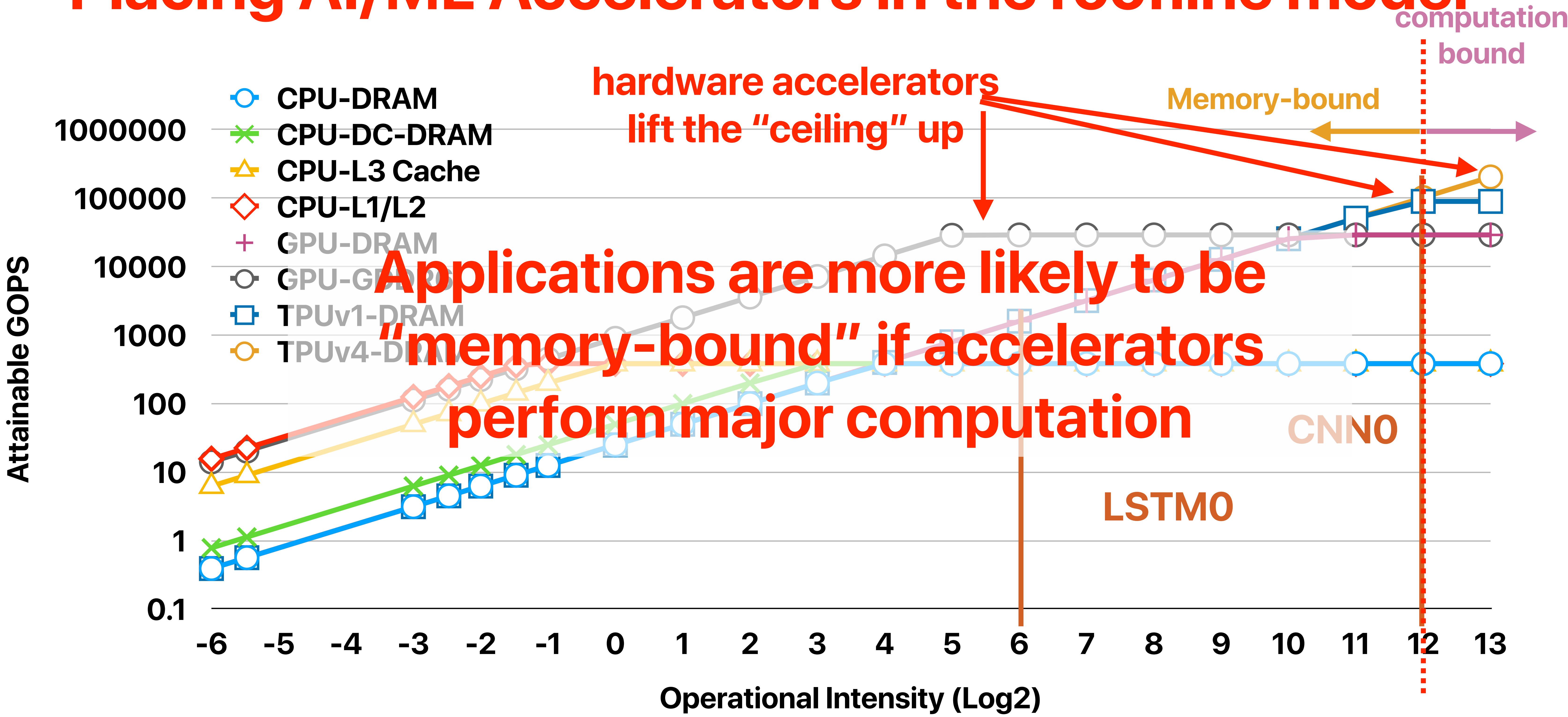
In/Near Memory Processing (1)

Hung-Wei Tseng

Recap: The landscape of modern computers


DRAM
CPU-Memory BUS
DDR4
Memory Controller
intel CORE
AI/ML Accelerator
PCI EXPRESS
PCIe Root Complex
HDD
SSD
NIC
FPGA
GPU
XPU
Movidius MA2485
High-performance, but very specialized
General-purpose for "regular" workloads, but can be power consuming
Can offload, but not super efficient

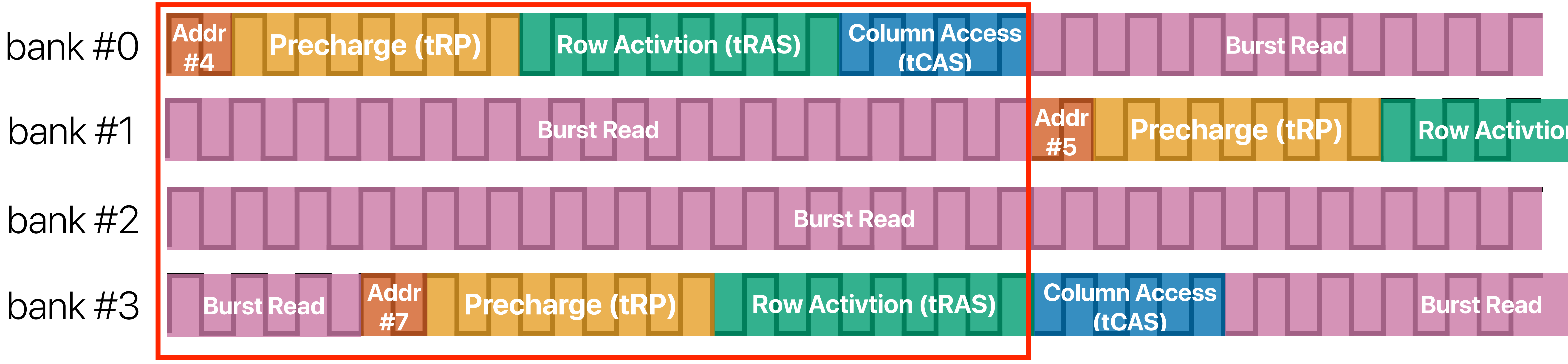
Placing AI/ML Accelerators in the roofline model



Applications are more likely to be "memory-bound" if accelerators perform major computation

Recap: Multi-bank access

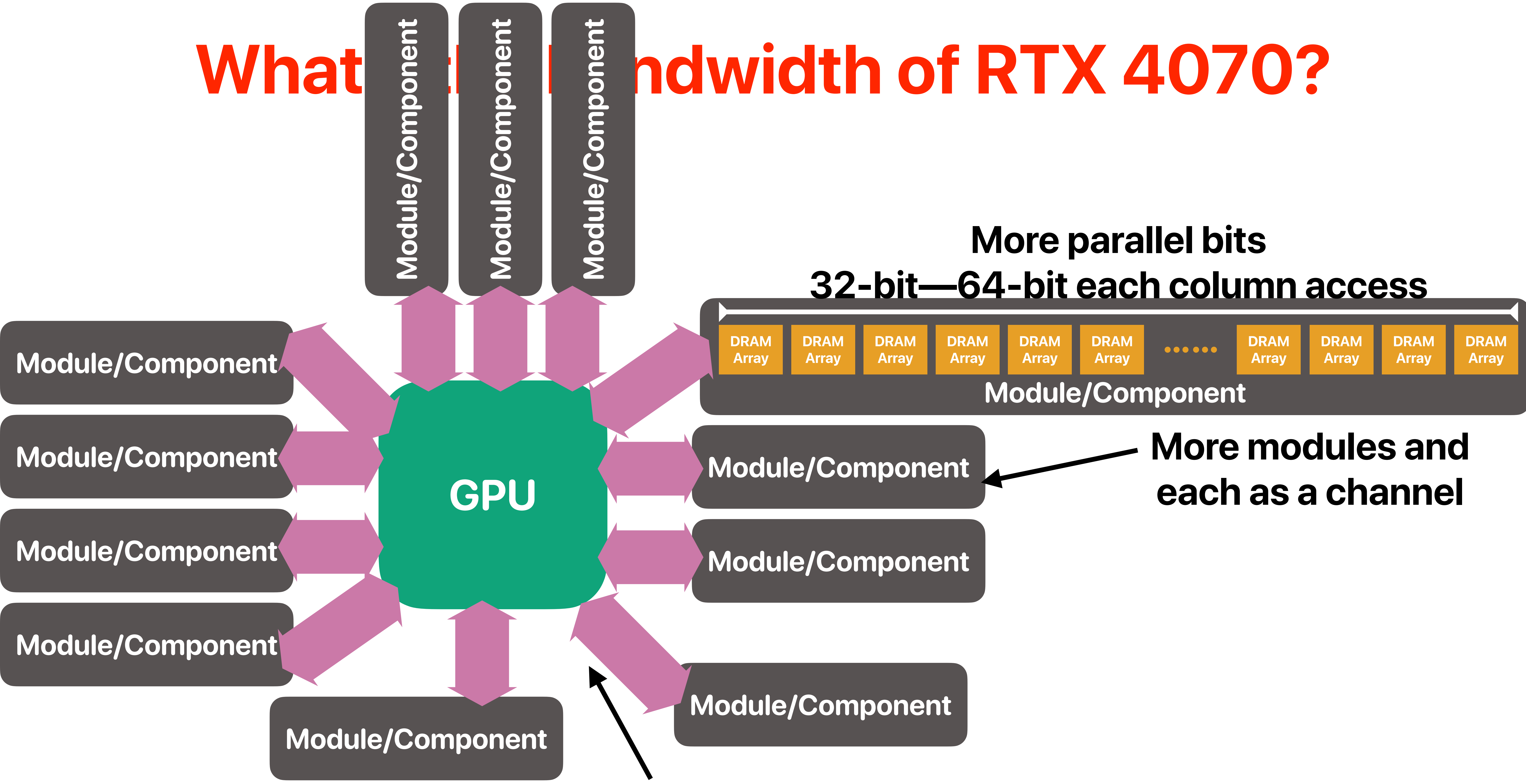
The latency of pre-charge, row/column accesses is fully covered!



Recap: Improving memory-bounded applications

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

What ... ndwidth of RTX 4070?



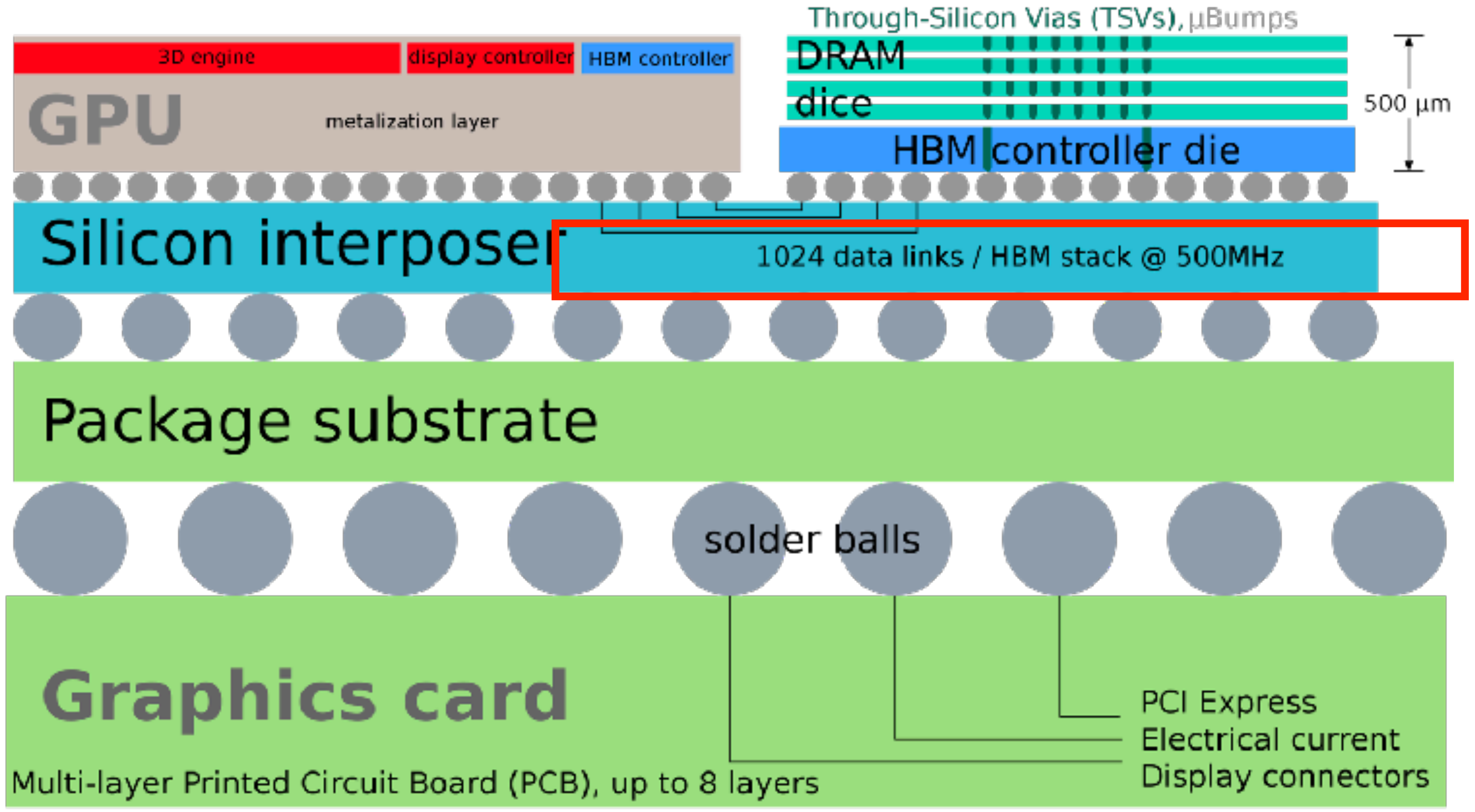
More parallel bits

32-bit—64-bit each column access

More modules and each as a channel

192-bit

HBM (High Bandwidth Memory)

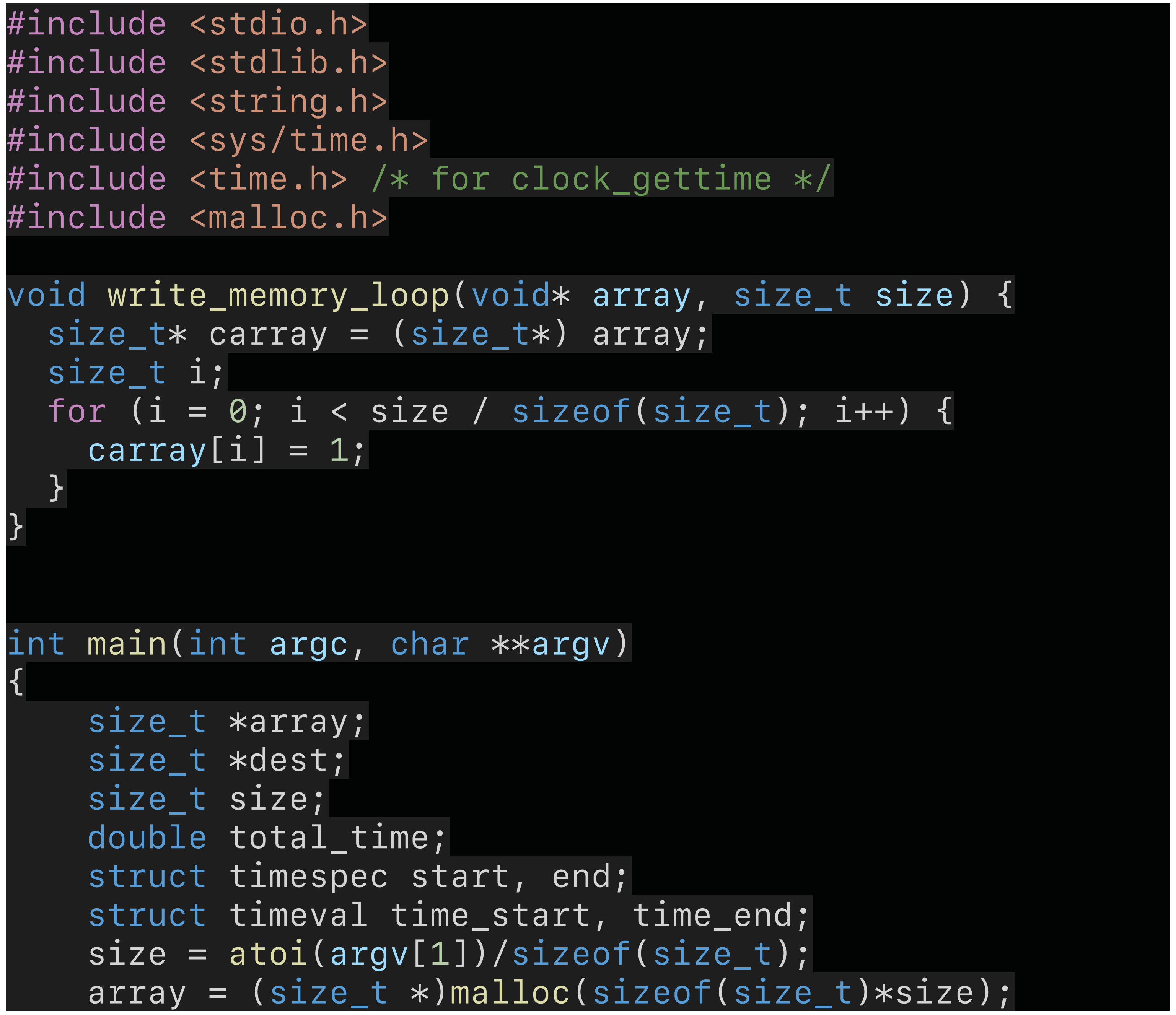


$$0.5G \times \frac{1024bits}{8bits} = 64GB/sec$$

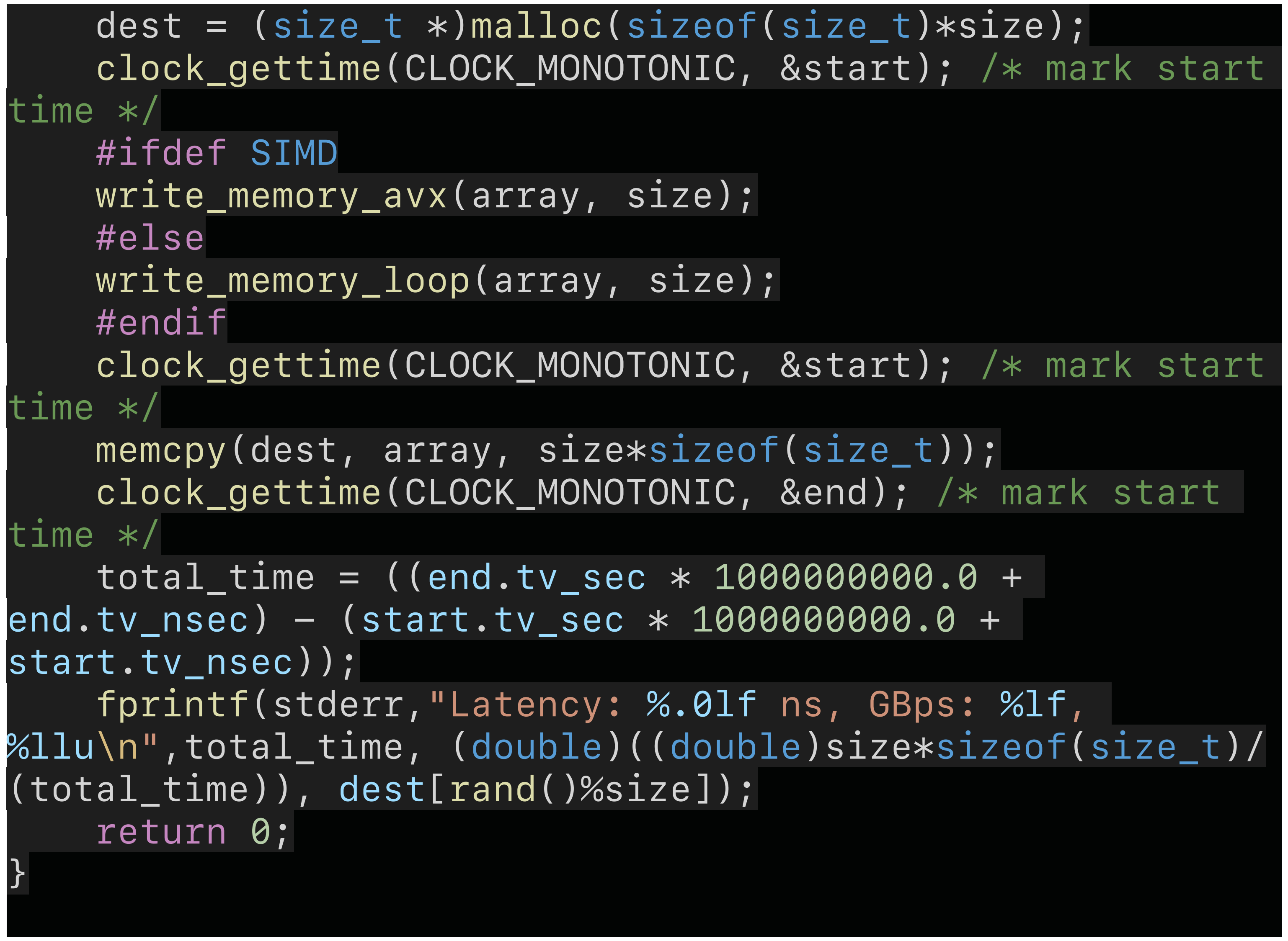
If you have 4x modules — 4*64 = 256 GB/sec

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

Recap: Memcpy



```
#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);
```

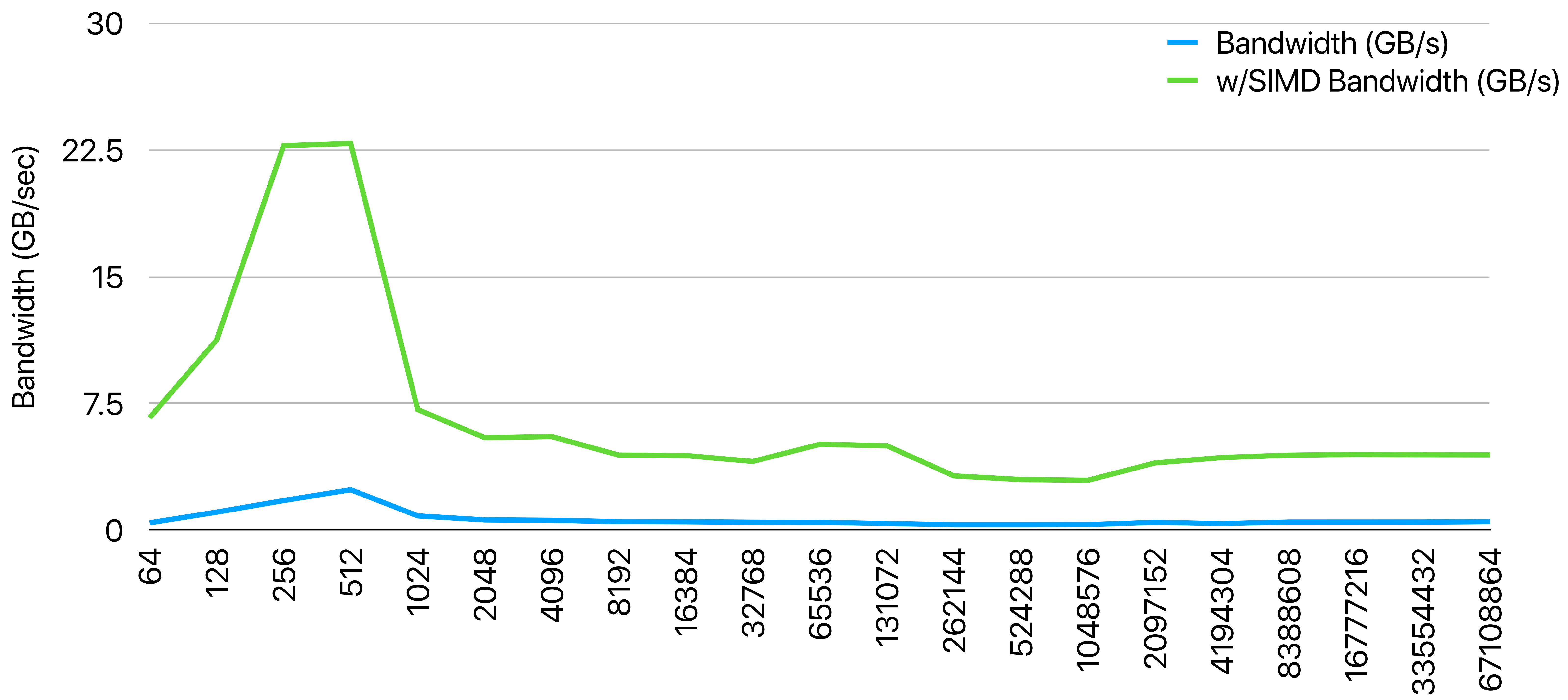


```
    dest = (size_t *)malloc(sizeof(size_t)*size);
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start 
time */
    #ifdef SIMD
    write_memory_avx(array, size);
    #else
    write_memory_loop(array, size);
    #endif
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start 
time */
    memcpy(dest, 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

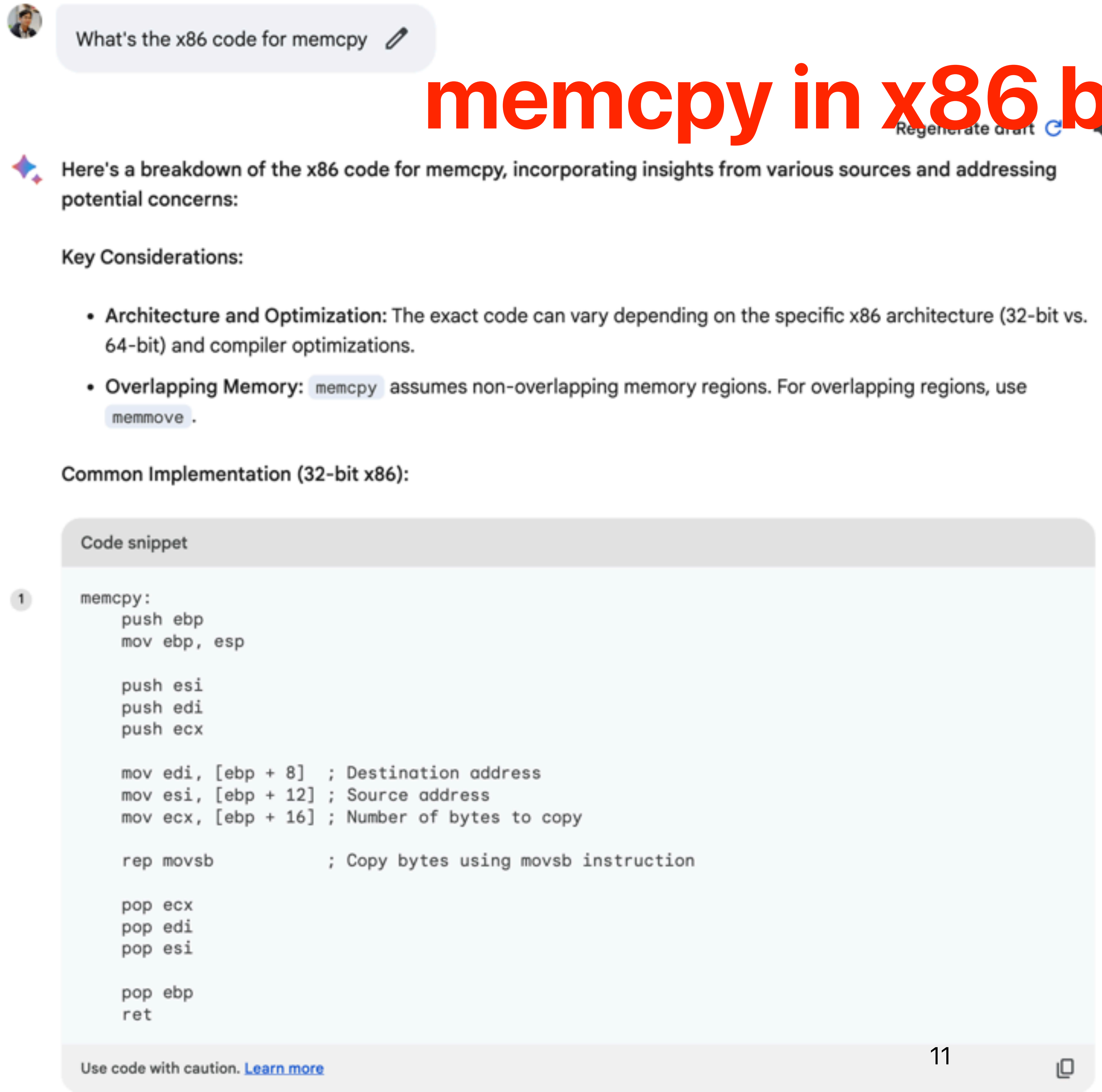
Memory copy bandwidth



Outline

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

memcpy in x86 by Gemini



What's the x86 code for memcpy

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

Key Considerations:

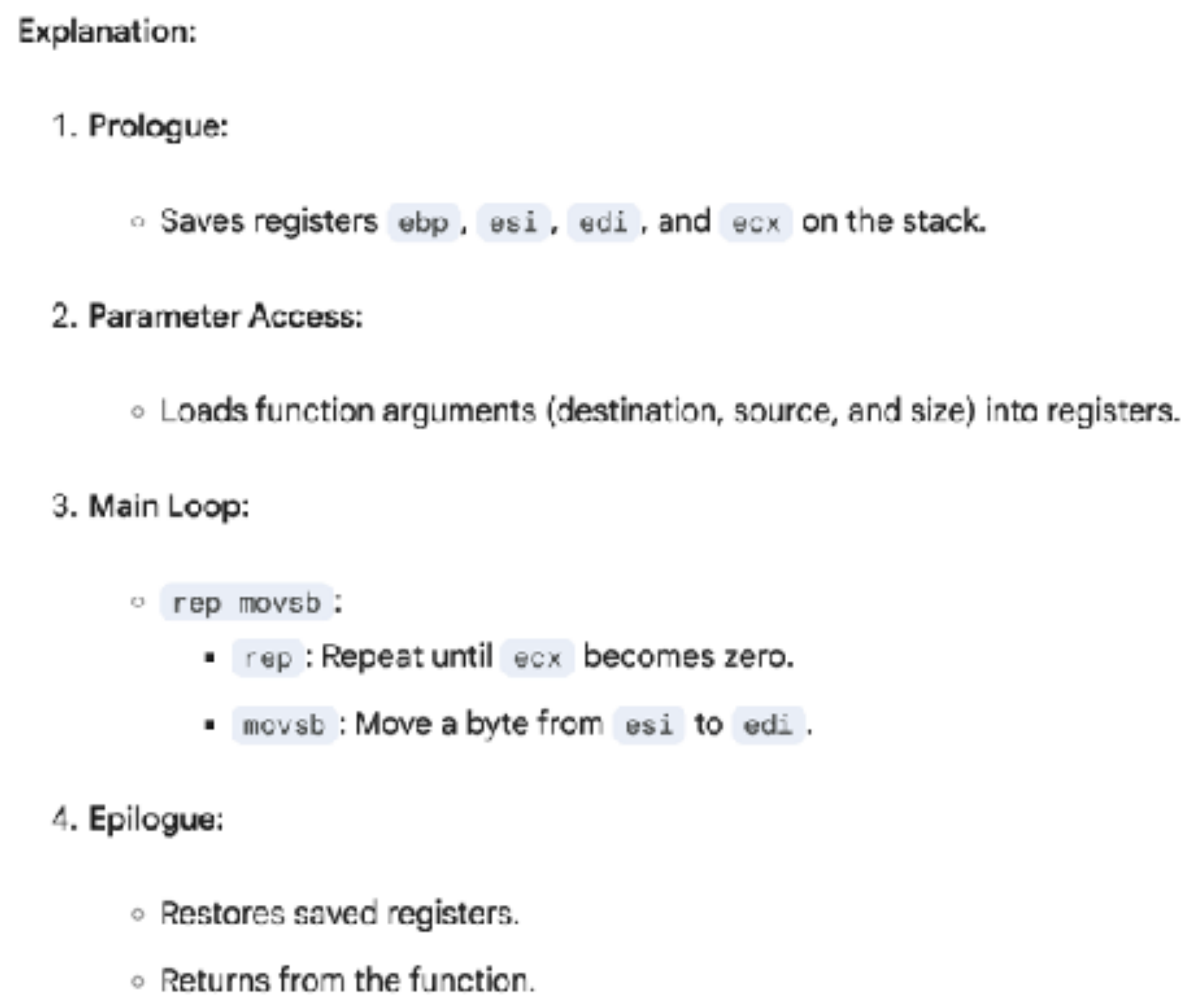
- **Architecture and Optimization:** The exact code can vary depending on the specific x86 architecture (32-bit vs. 64-bit) and compiler optimizations.
- **Overlapping Memory:** `memcpy` assumes non-overlapping memory regions. For overlapping regions, use `memmove`.

Common Implementation (32-bit x86):

Code snippet

```
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
```

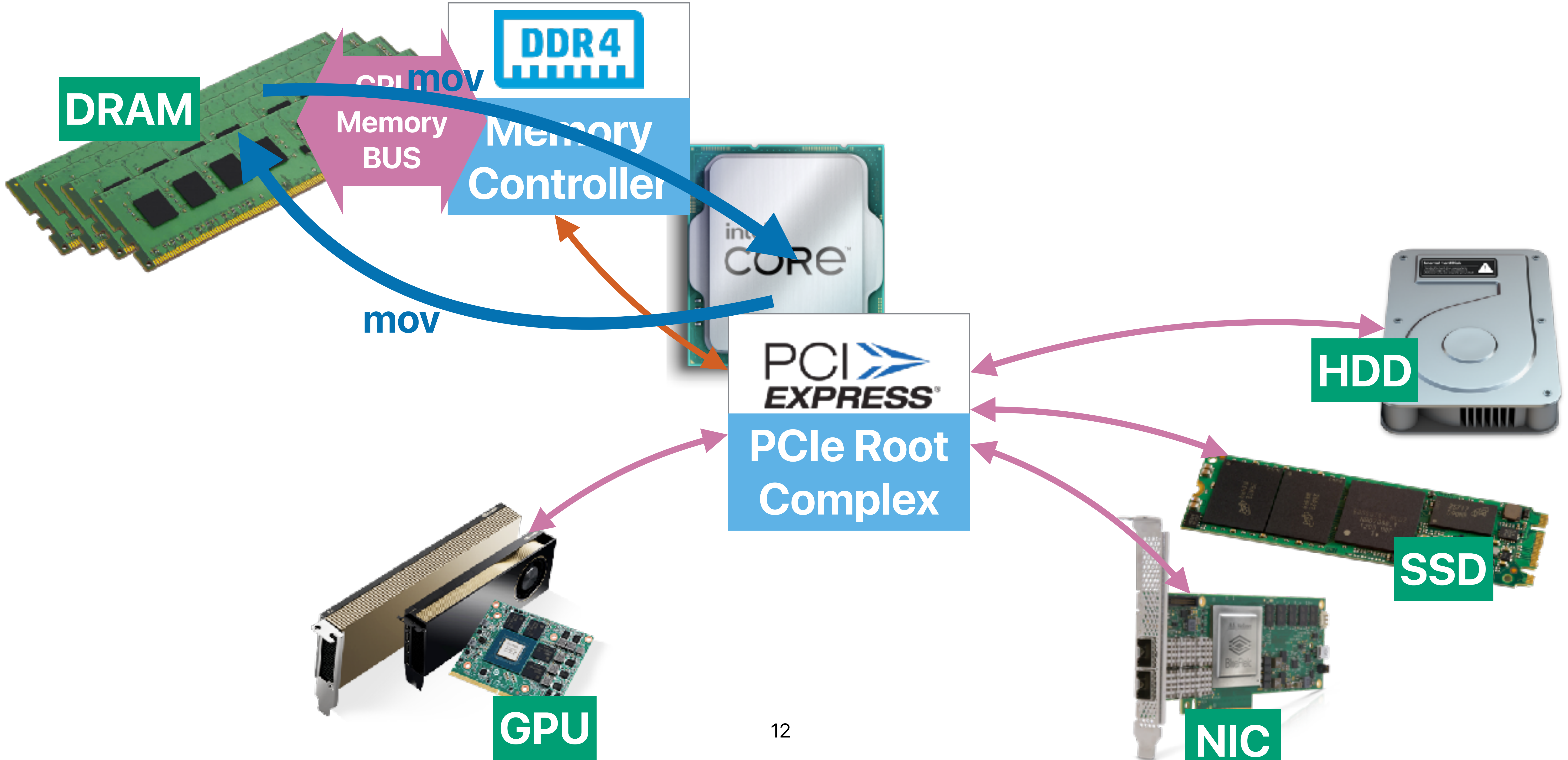
Use code with caution. Learn more



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



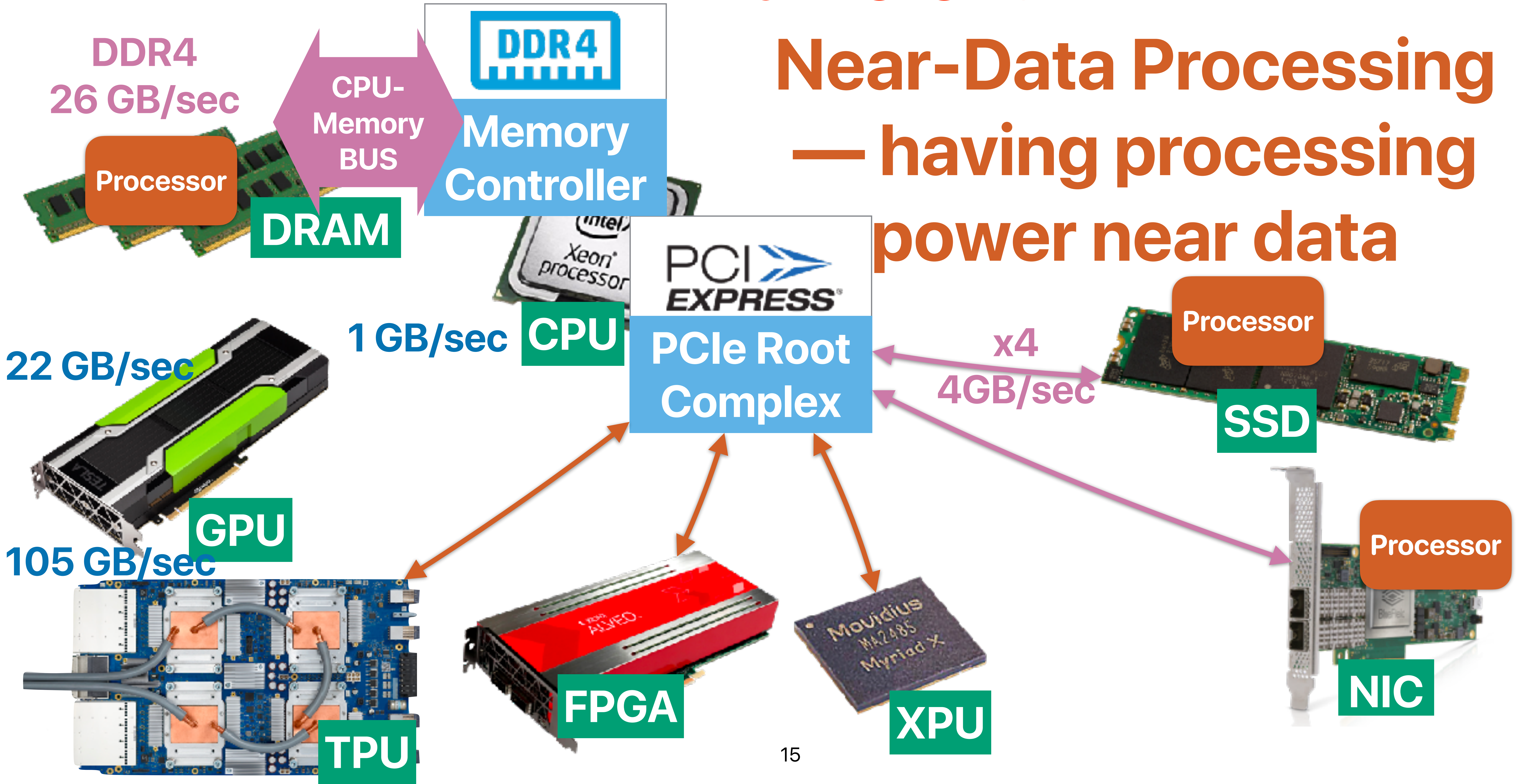
Why is memcpy slow?

- 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?

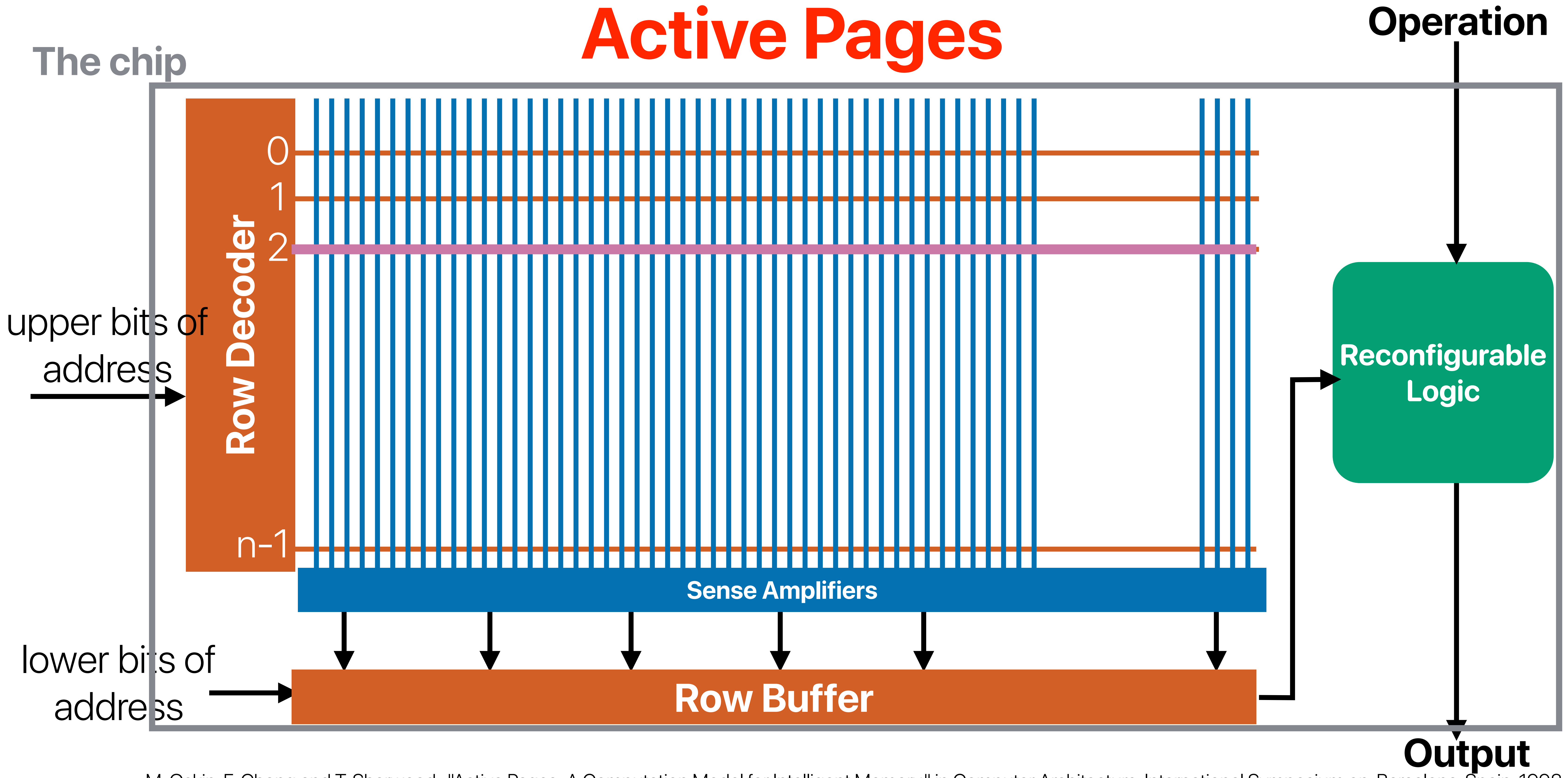
Think different

Near-Data Processing — having processing power near data



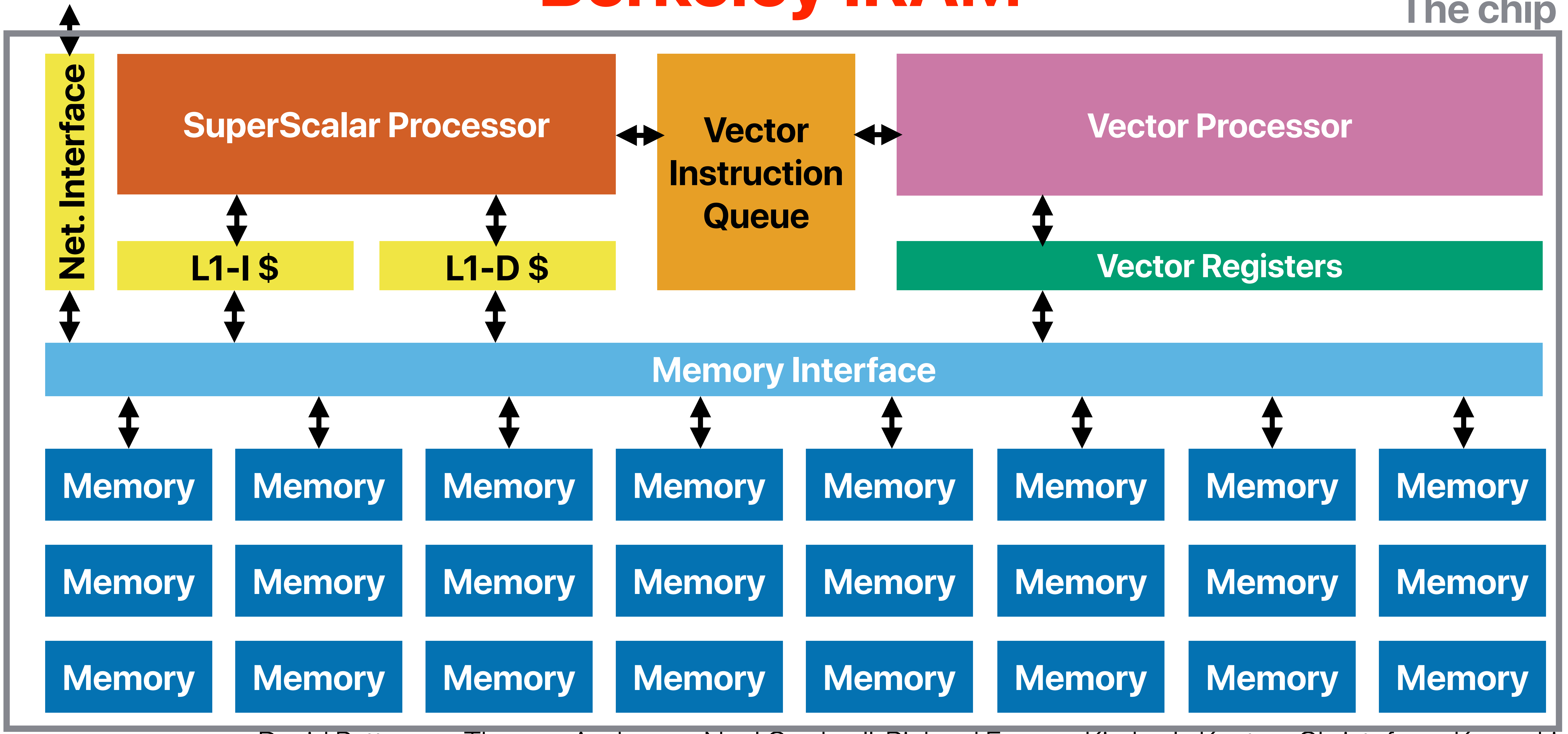
Near-memory processing

Active Pages



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



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.

3:00

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

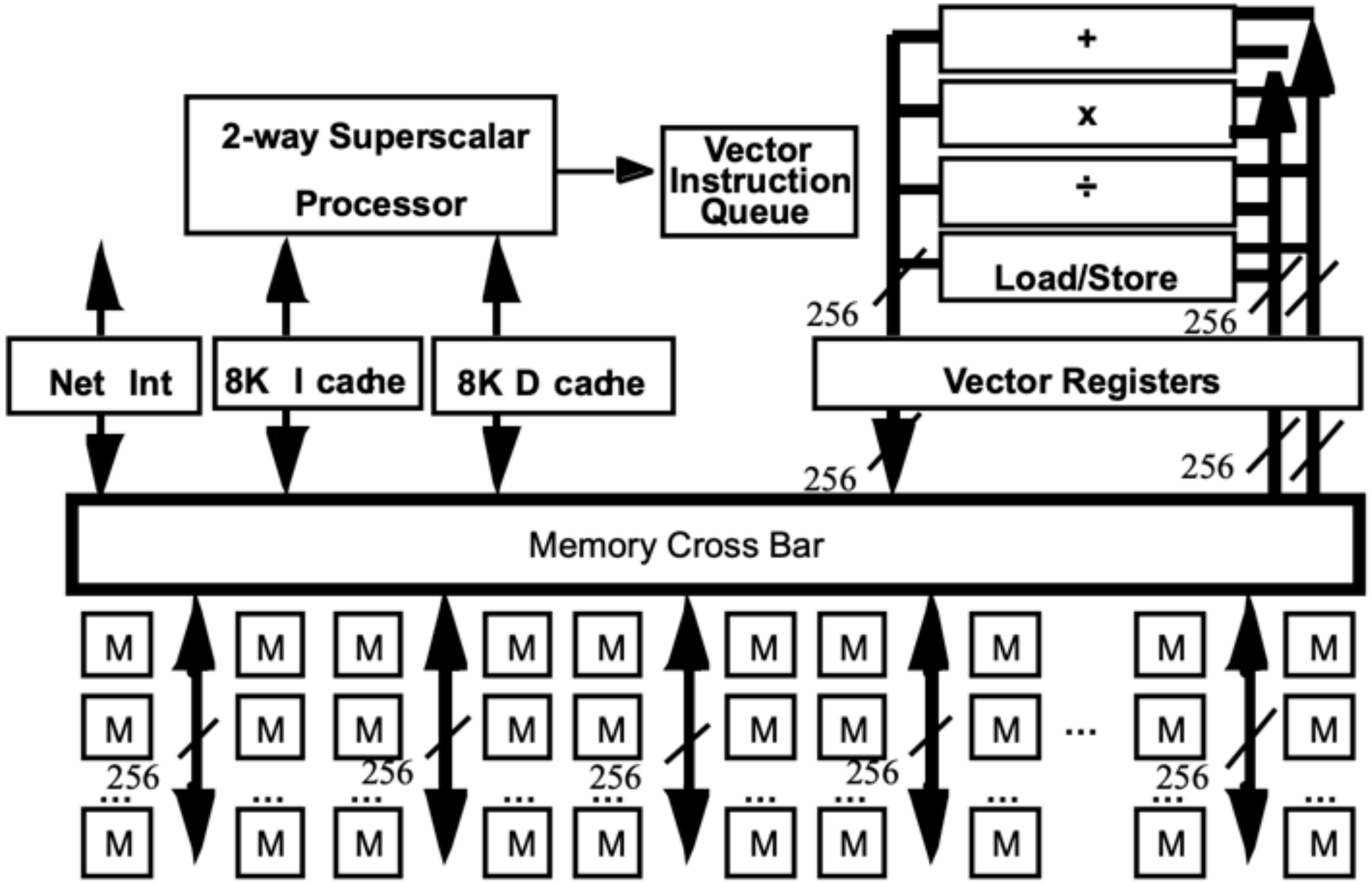
What kind of applications?

- 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 µm, Fast Logic IRAM: ≈500MHz 5 GFLOPS(64b)/40 GOPS(8b)/24MB

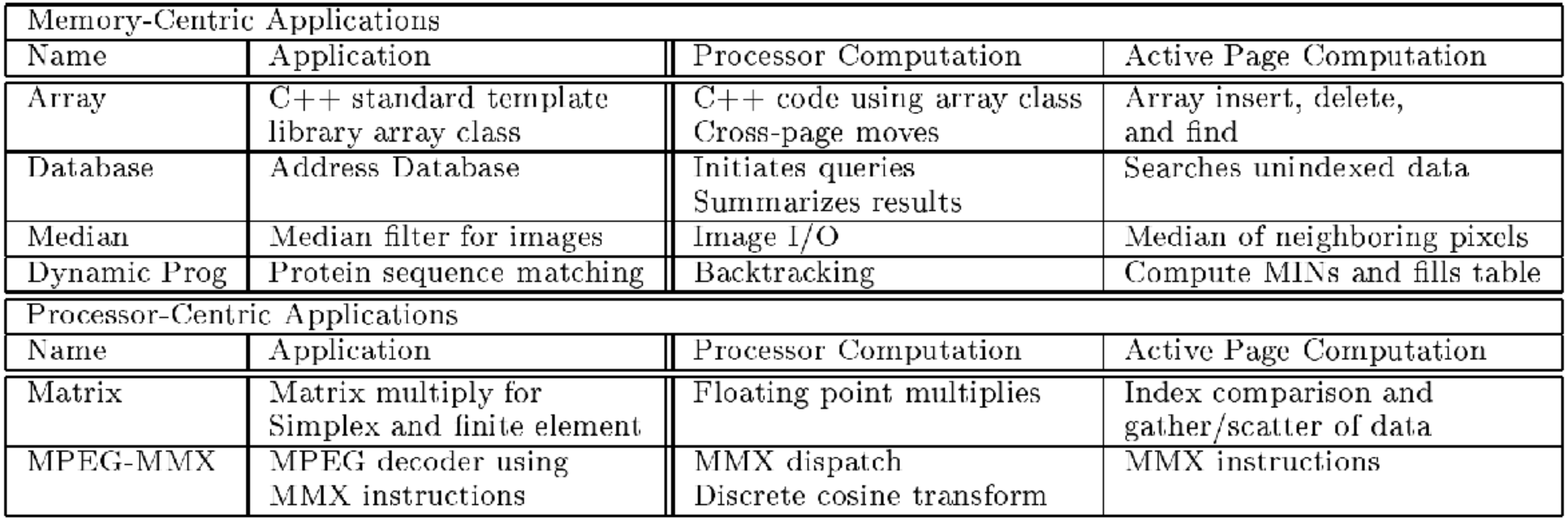


19

	Small			Large		
	Conven-	IRAM		Conven-	IRAM	
benchmark	tional	(.75 **X**)	(1.0 **X**)	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.

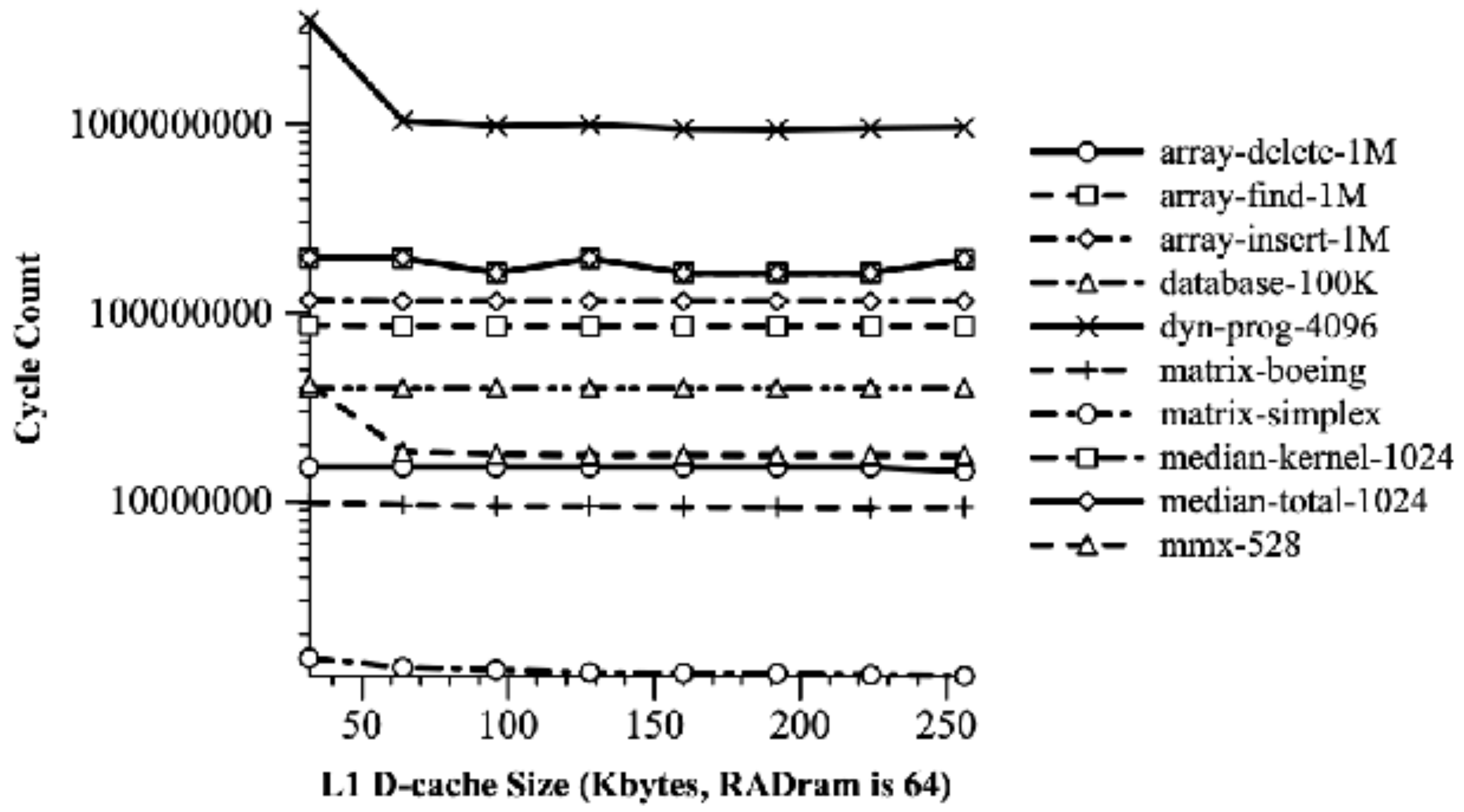
Active Pages: Applications



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



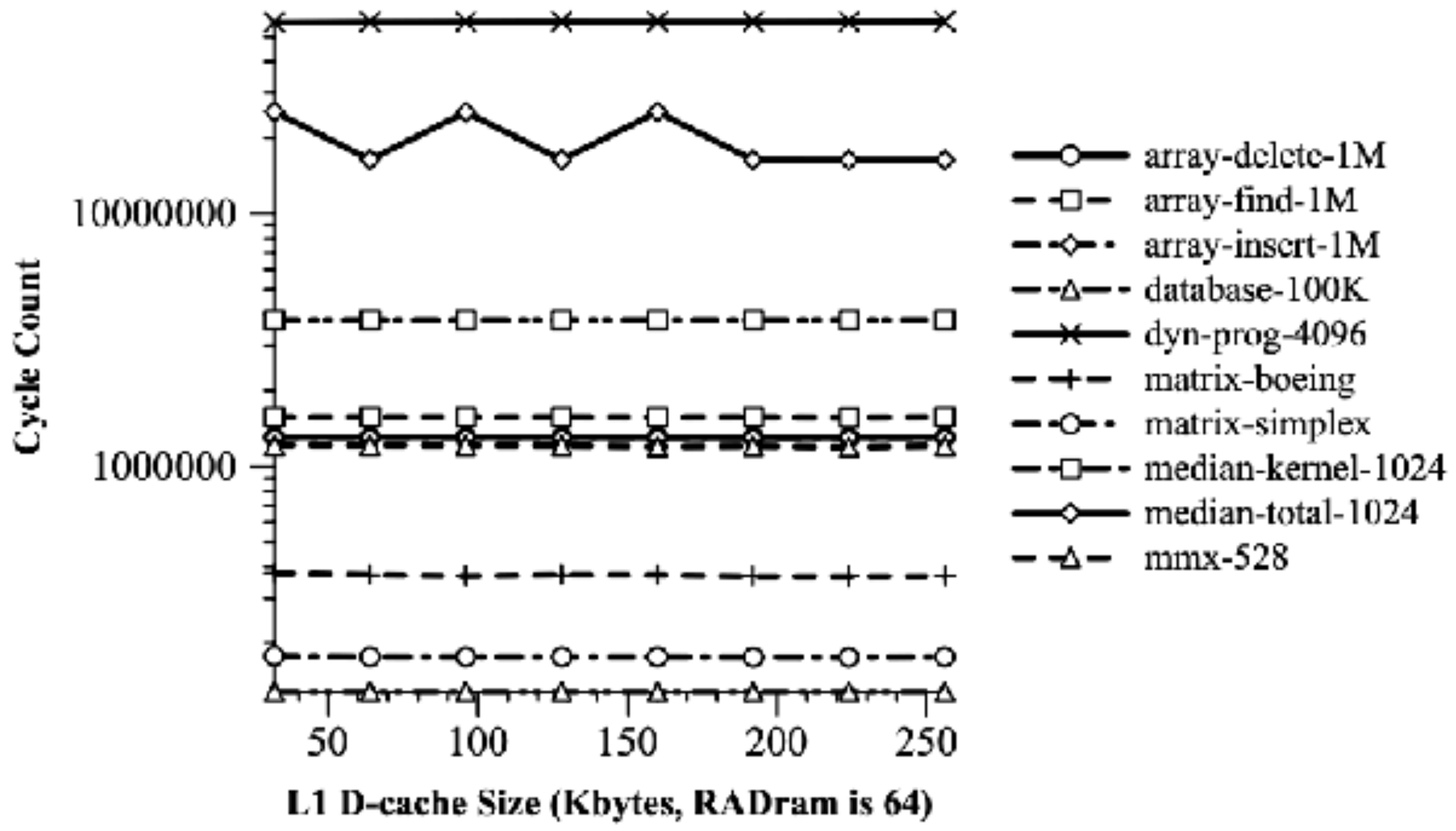


Figure 5: Conventional **(left)** and RADram **(right)** Execution Time vs. L1 Data Cache Size

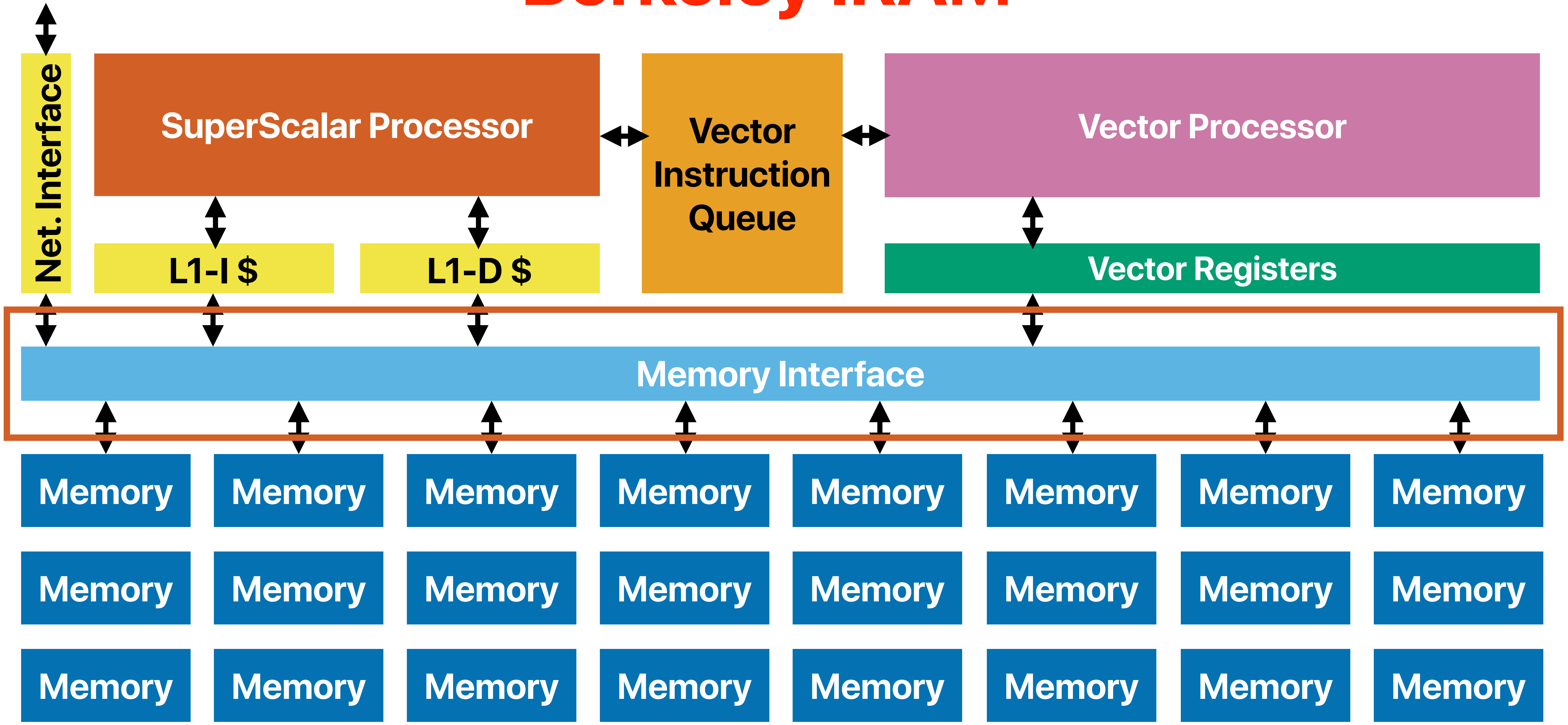
3:00

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

Why IRAM or Active Pages does not work out?

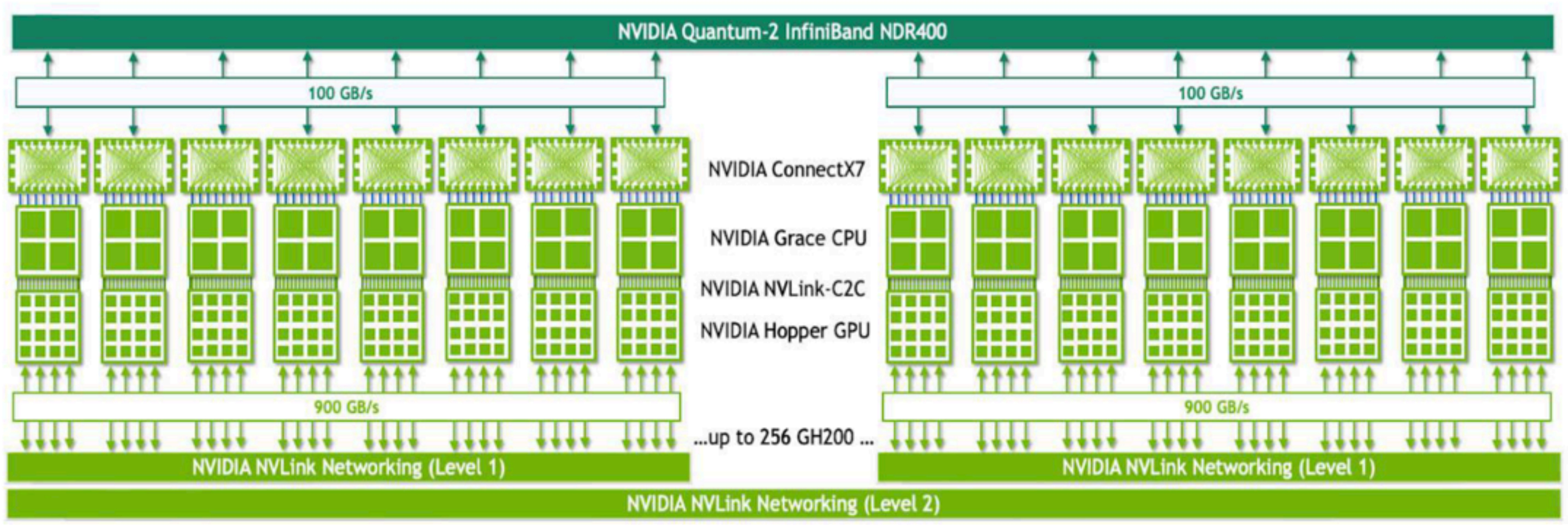
- Programming
  - Why GPU succeed later?
- Applications
- Lack of industrial implementation, support
- Hardware design — Processor v.s. DRAM process technologies
- Cost
  - For Active Pages — small processor each module
  - For IRAM
    - High bandwidth on-chip interconnect was expensive
    - Other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

NVIDIA Grace Hopper GH200


NVIDIA Quantum-2 InfiniBand NDR400
100 GB/s
100 GB/s
NVIDIA ConnectX7
NVIDIA Grace CPU
NVIDIA NVLink-C2C
NVIDIA Hopper GPU
900 GB/s
900 GB/s
...up to 256 GH200 ...
NVIDIA NVLink Networking (Level 1)
NVIDIA NVLink Networking (Level 1)
NVIDIA NVLink Networking (Level 2)

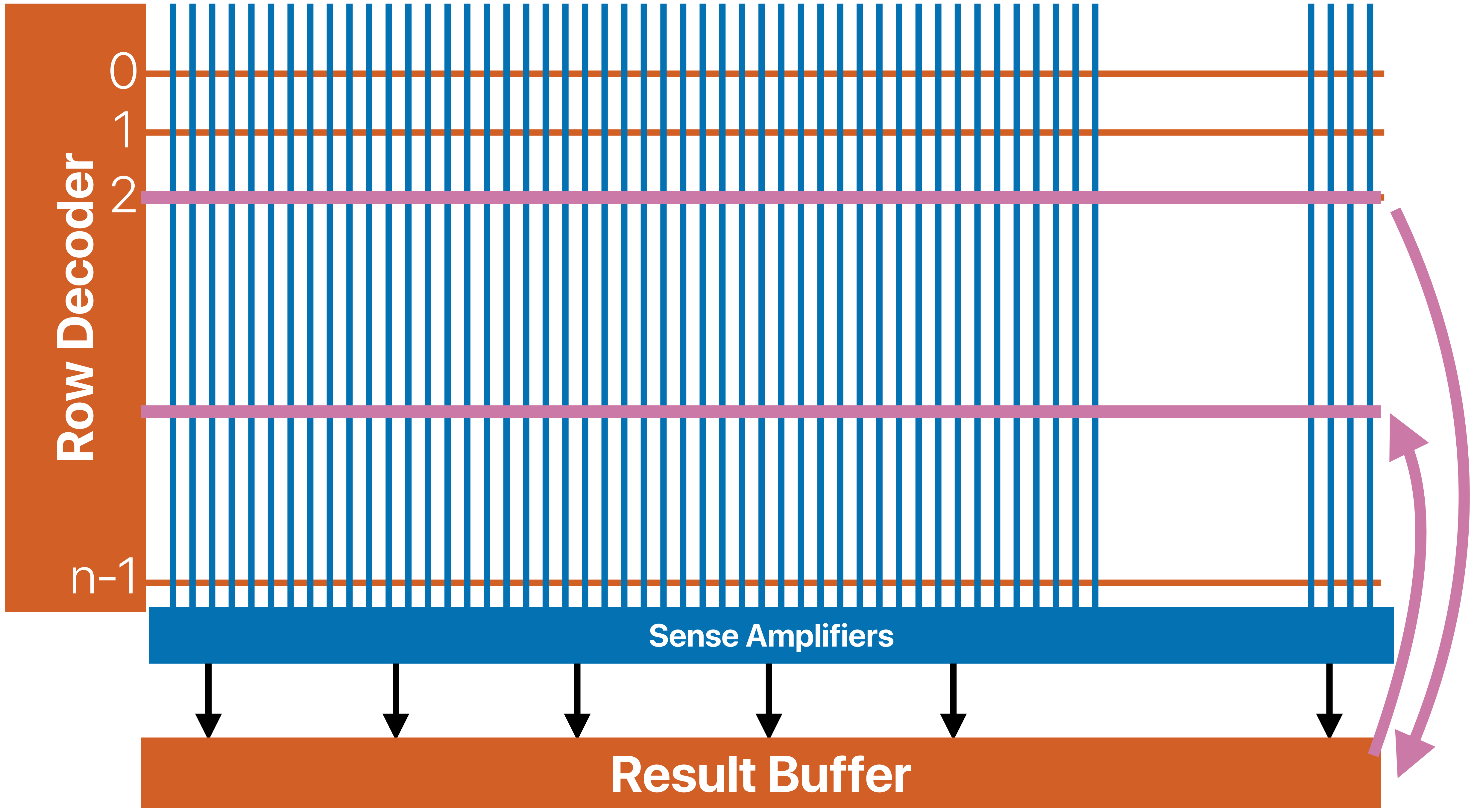
igure 4. NVIDIA GH200 NVL32 with NVLink Switch System for strong-scaling giant ML workloads

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?

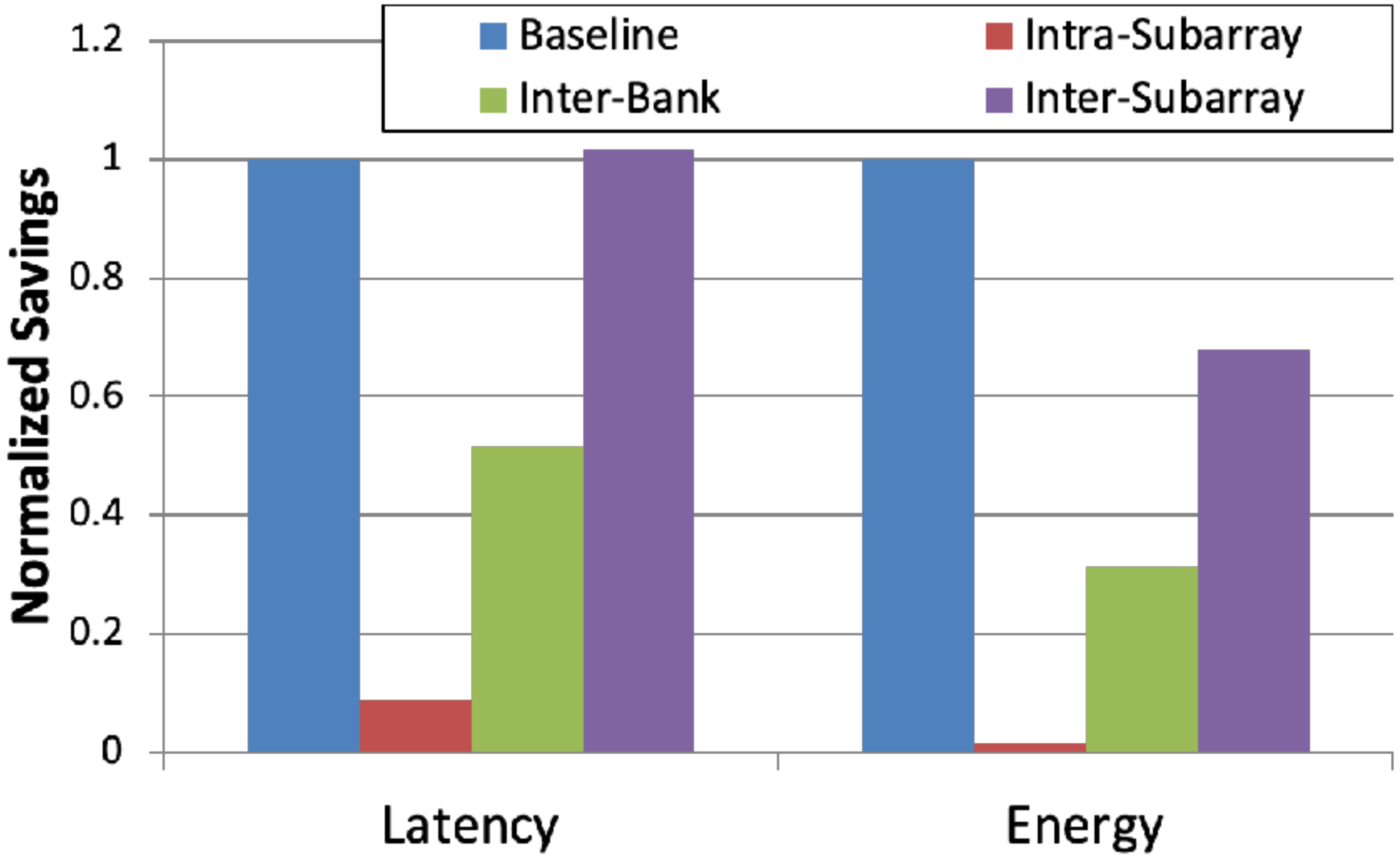


memmove & memcpy: 5% cycles in Google's datacenter

Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy 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



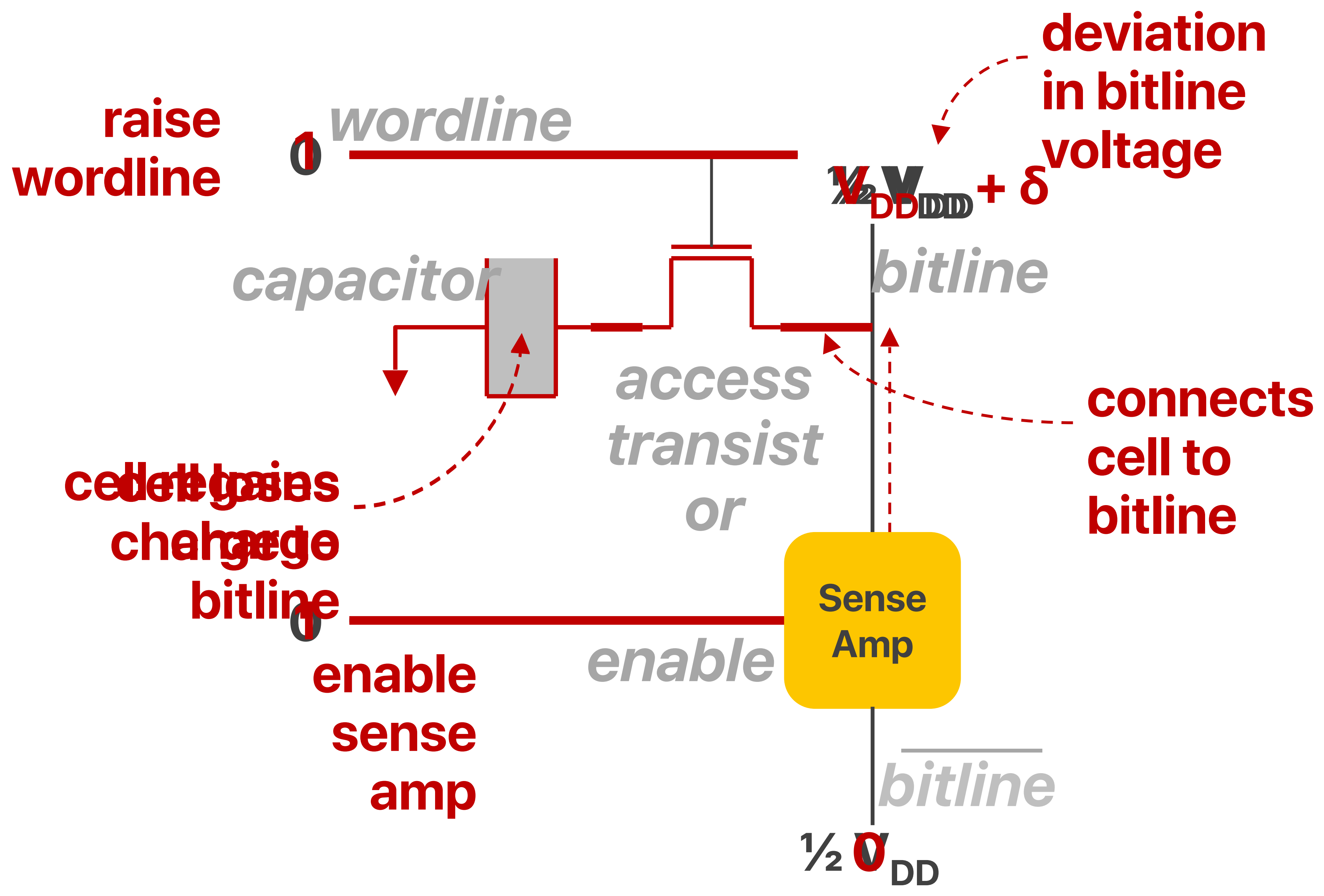
3:00

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

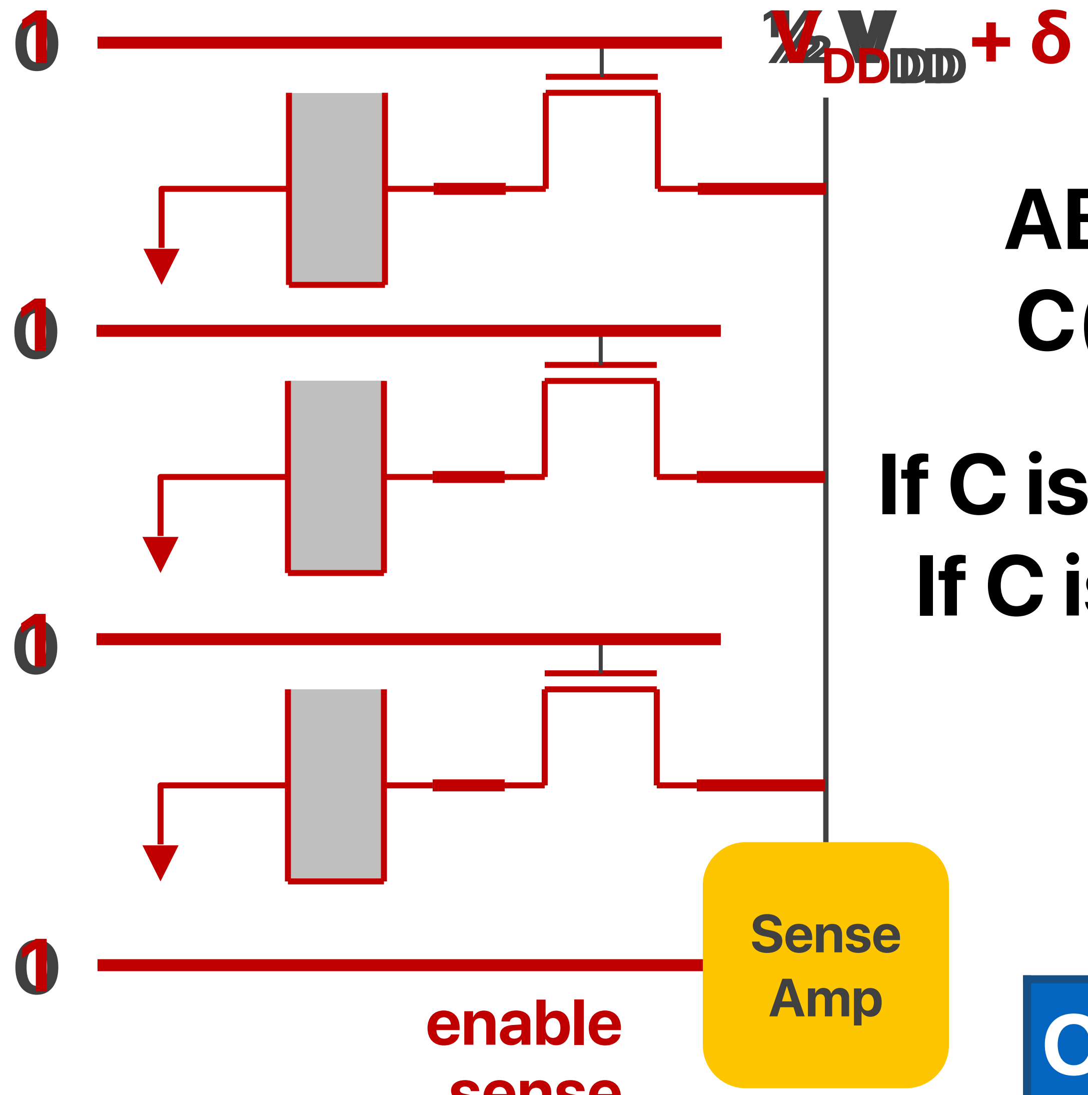
Limitations of RowClone

- 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



activate all three rows

enable sense amp

AB + BC + AC =
C(A+B) + C′AB

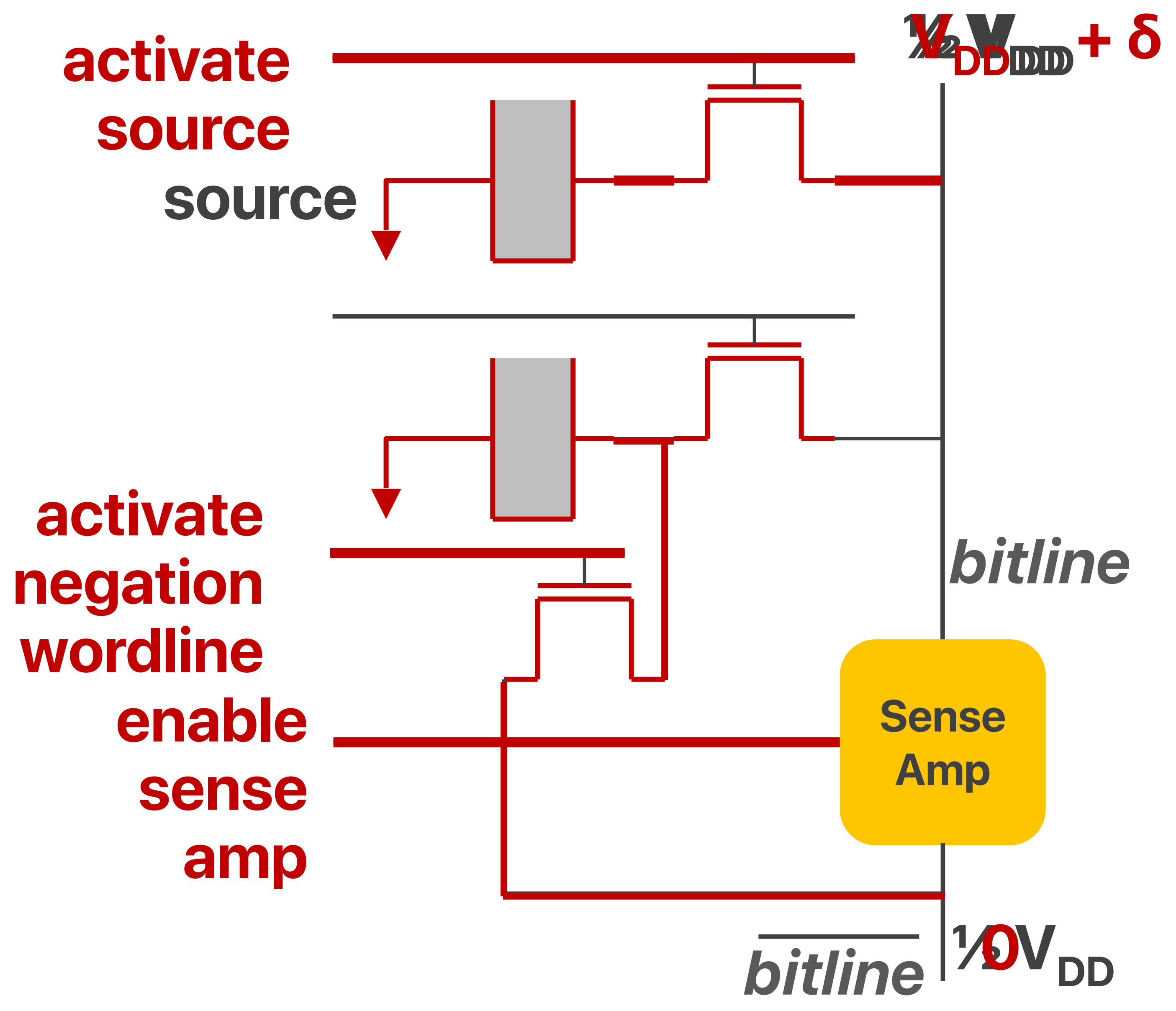
If C is 0, we can compute AND
If C is 1, we can compute OR

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′
	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

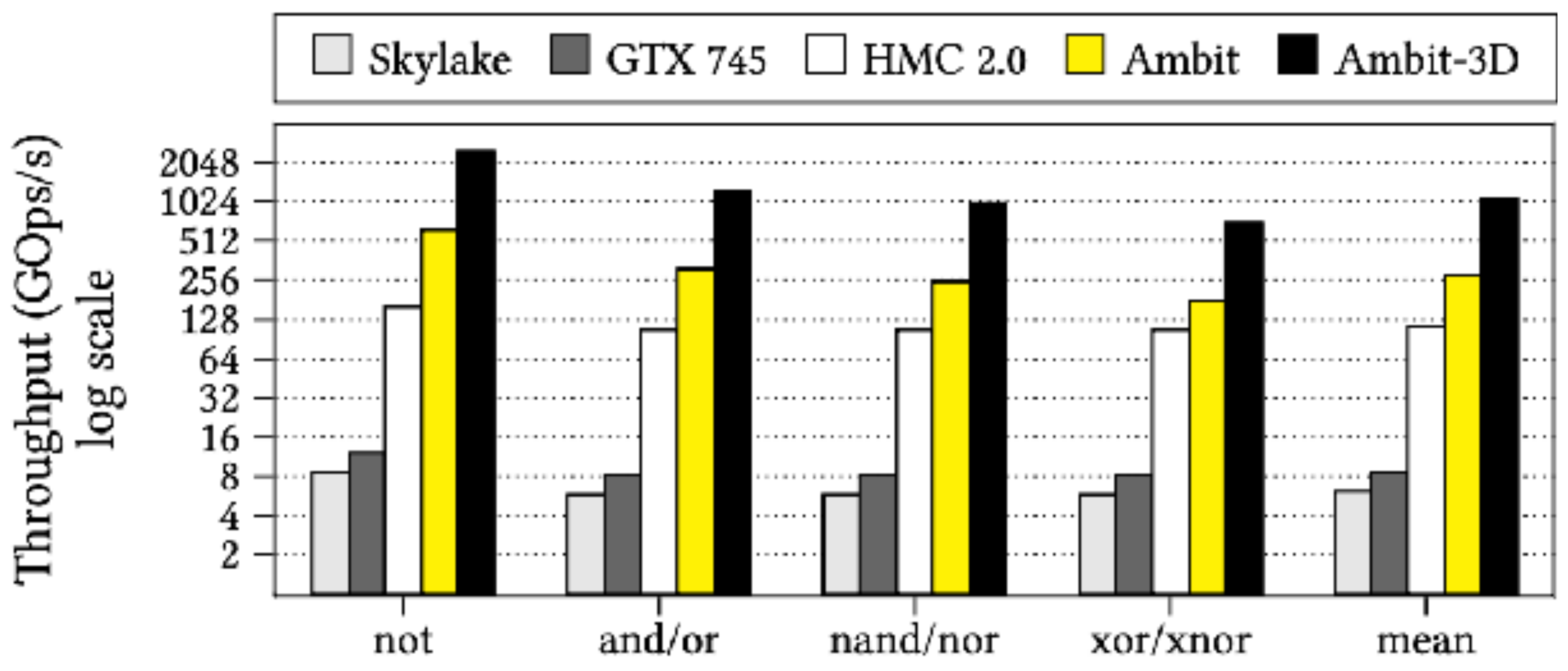


Figure 9: Throughput of bulk bitwise operations.

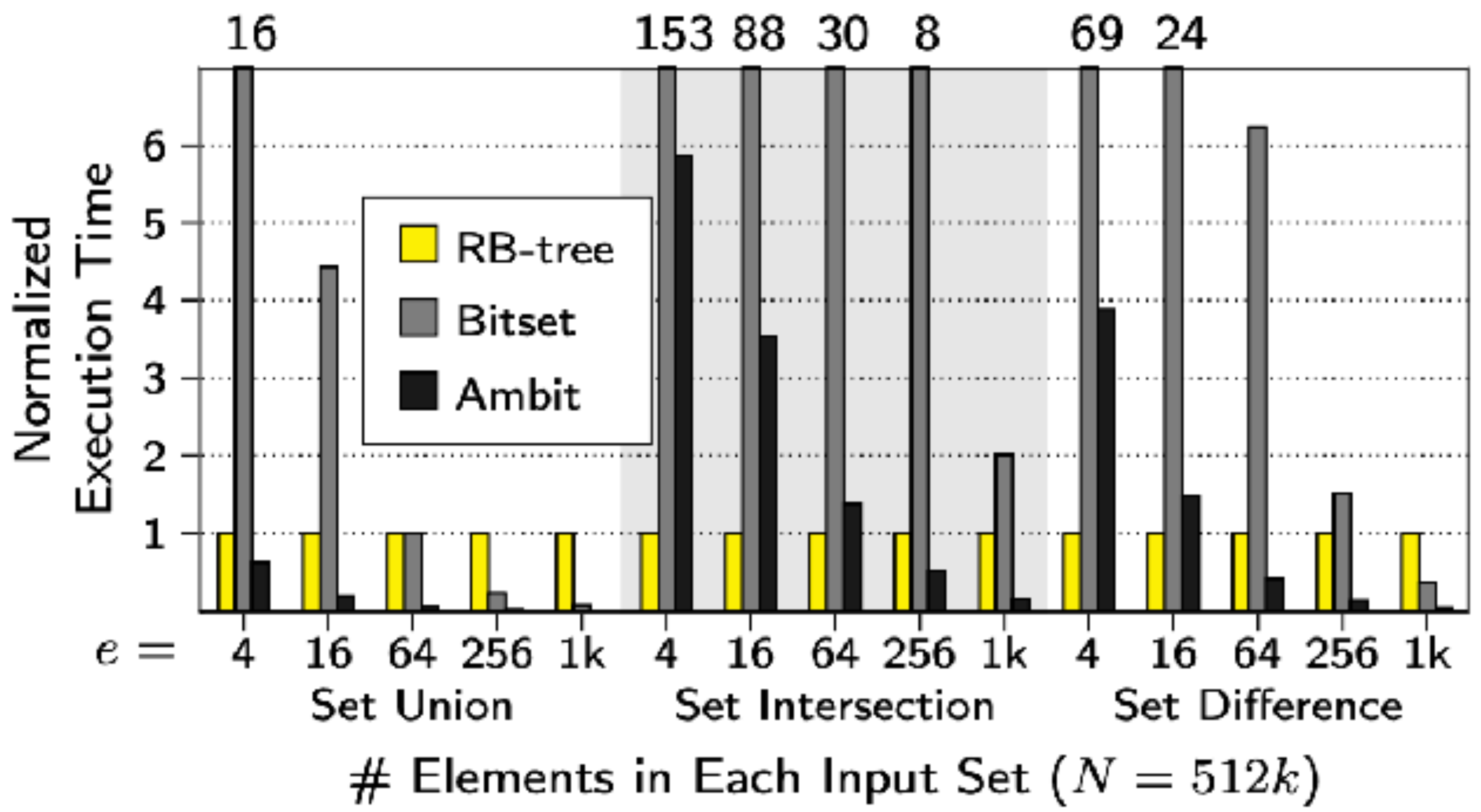


Figure 12: Performance of set operations

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

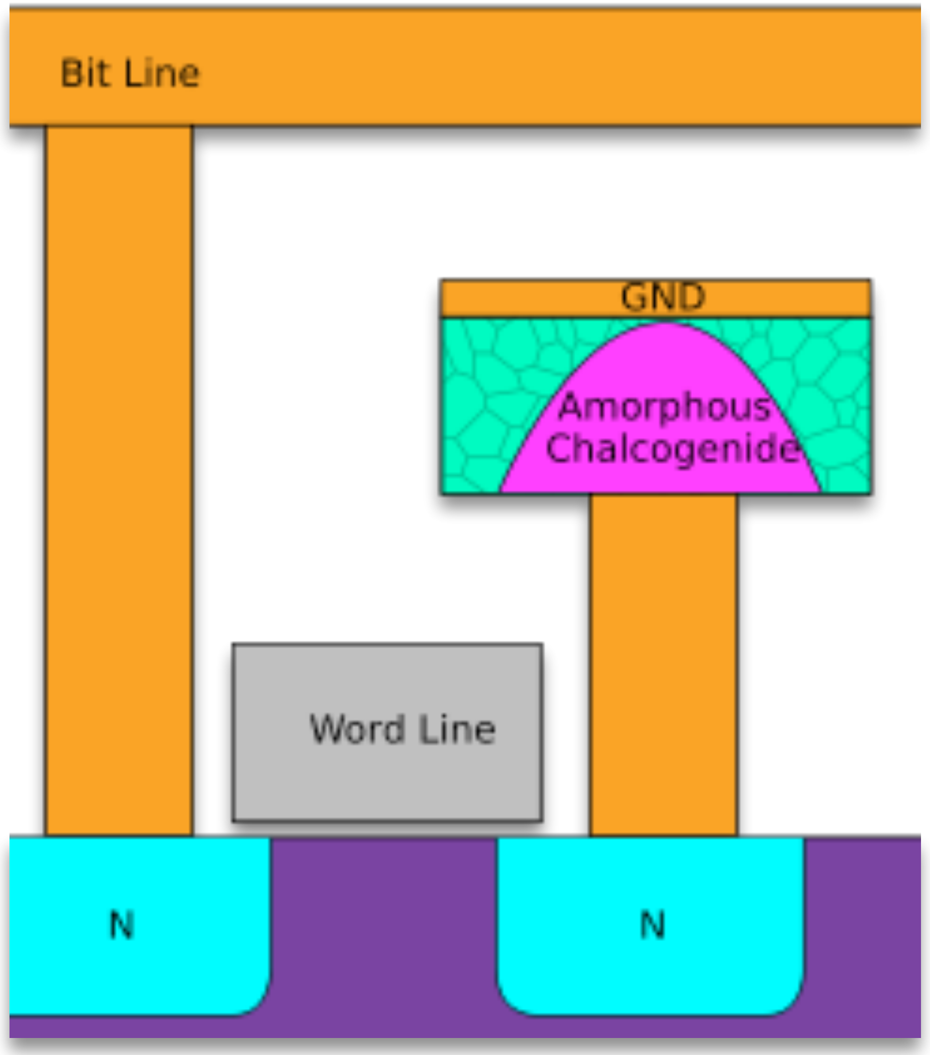
$$\boxed{V_c} = V_s \times e^{\frac{-t}{RC}}$$

Ohm's law

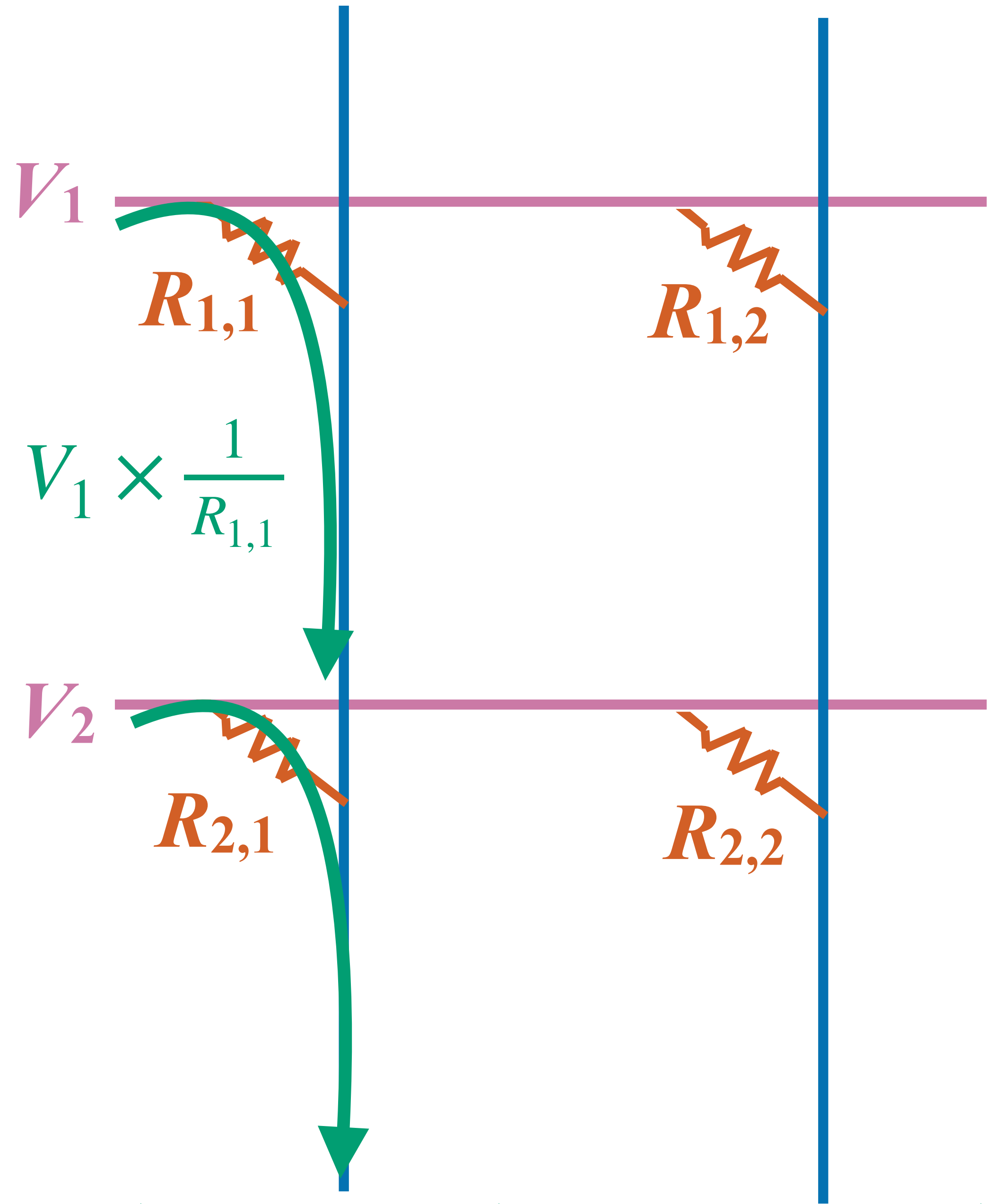
$$V = \boxed{I} \times R$$

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



ReRAM



$$I_{1,1} = V_1 \times \frac{1}{R_{1,1}}$$

$$I_1 = V_1 \times \frac{1}{R_{1,1}} + V_2 \times \frac{1}{R_{2,1}} \quad I_2 = V_1 \times \frac{1}{R_{2,1}} + V_2 \times \frac{1}{R_{2,2}}$$

$$\begin{array}{|c|}\hline V_1 \\ \hline V_2 \\ \hline V_3 \\ \hline V_4 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|}\hline R_{1,1}^{-1} & R_{1,2}^{-1} & R_{1,3}^{-1} & R_{1,4}^{-1} \\ \hline R_{2,1}^{-1} & R_{2,2}^{-1} & R_{2,3}^{-1} & R_{2,4}^{-1} \\ \hline R_{3,1}^{-1} & R_{3,2}^{-1} & R_{3,3}^{-1} & R_{3,4}^{-1} \\ \hline R_{4,1}^{-1} & R_{4,2}^{-1} & R_{4,3}^{-1} & R_{4,4}^{-1} \\ \hline \end{array}$$

3:00

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

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 × 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, Yeseong Kim, and Tajana Rosing. 2019. FloatPIM: in-memory acceleration of deep neural network training with high precision. ISCA. 2019.

Electrical
Computer Science
Engineering

277

つづく