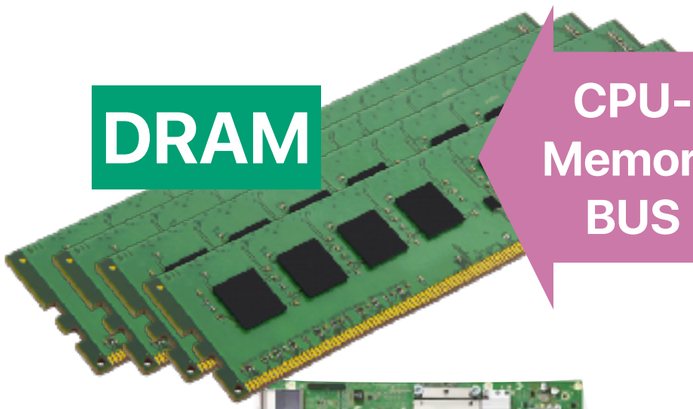
Memory subsystem (2) & In/Near Memory Processing

Hung-Wei Tseng

Recap: The landscape of modern computers



CPU-Memory BUS

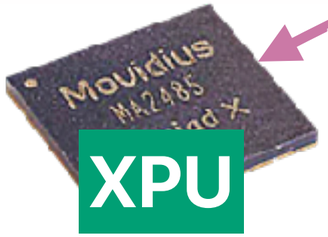
DDR4

Memory Controller

High-performance, but very specialized

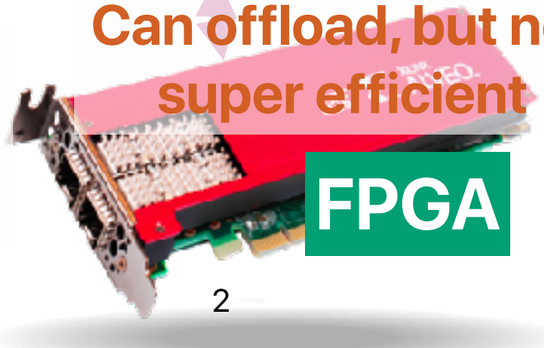
PCI EXPRESS

PCIe Root Complex

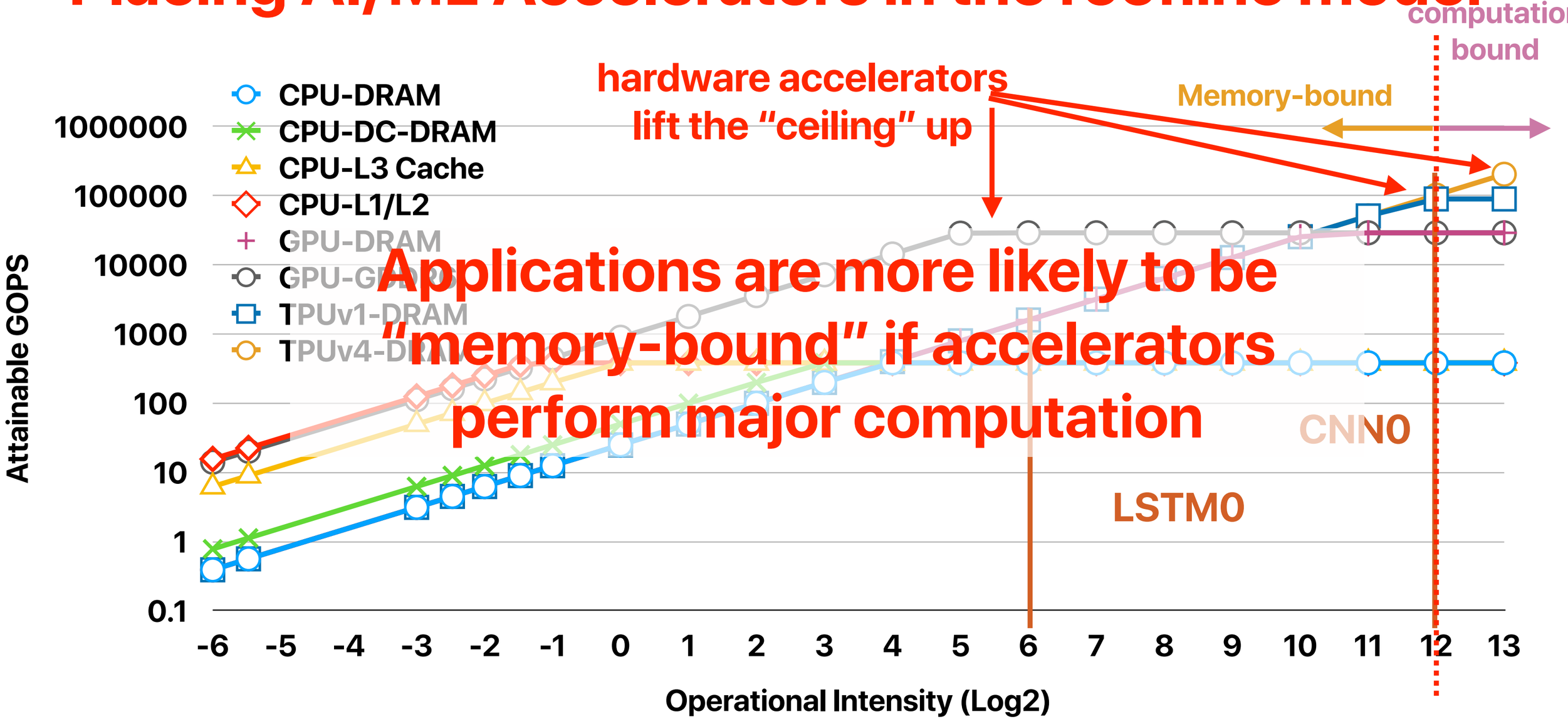


General-purpose for "regular" workloads, but can be power consuming

Can offload, but not super efficient



Placing AI/ML Accelerators in the roofline model

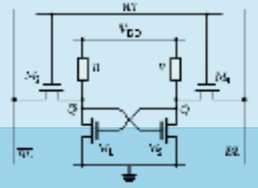


Recap: Memory technologies we have today

Volatile Memory

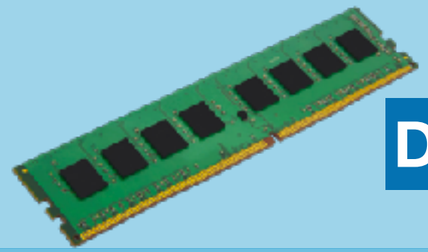
Non-Volatile Memory

100ps

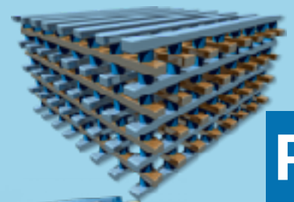


SRAM

ns



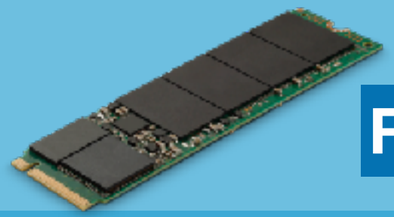
DRAM



RRAM

PCM 3DXPoint

us



Flash memory

ms

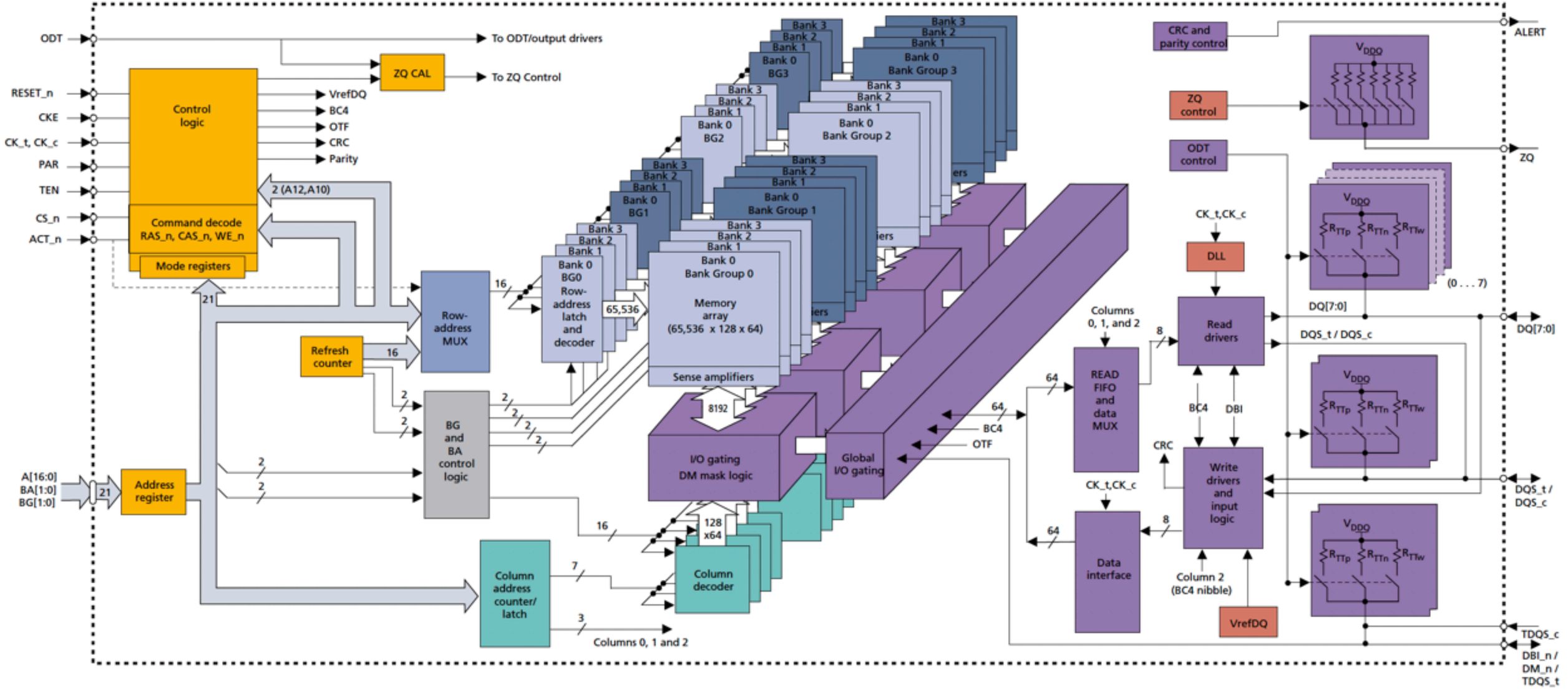
Hard Disk Drives

Recap: Improving memory-bounded applications

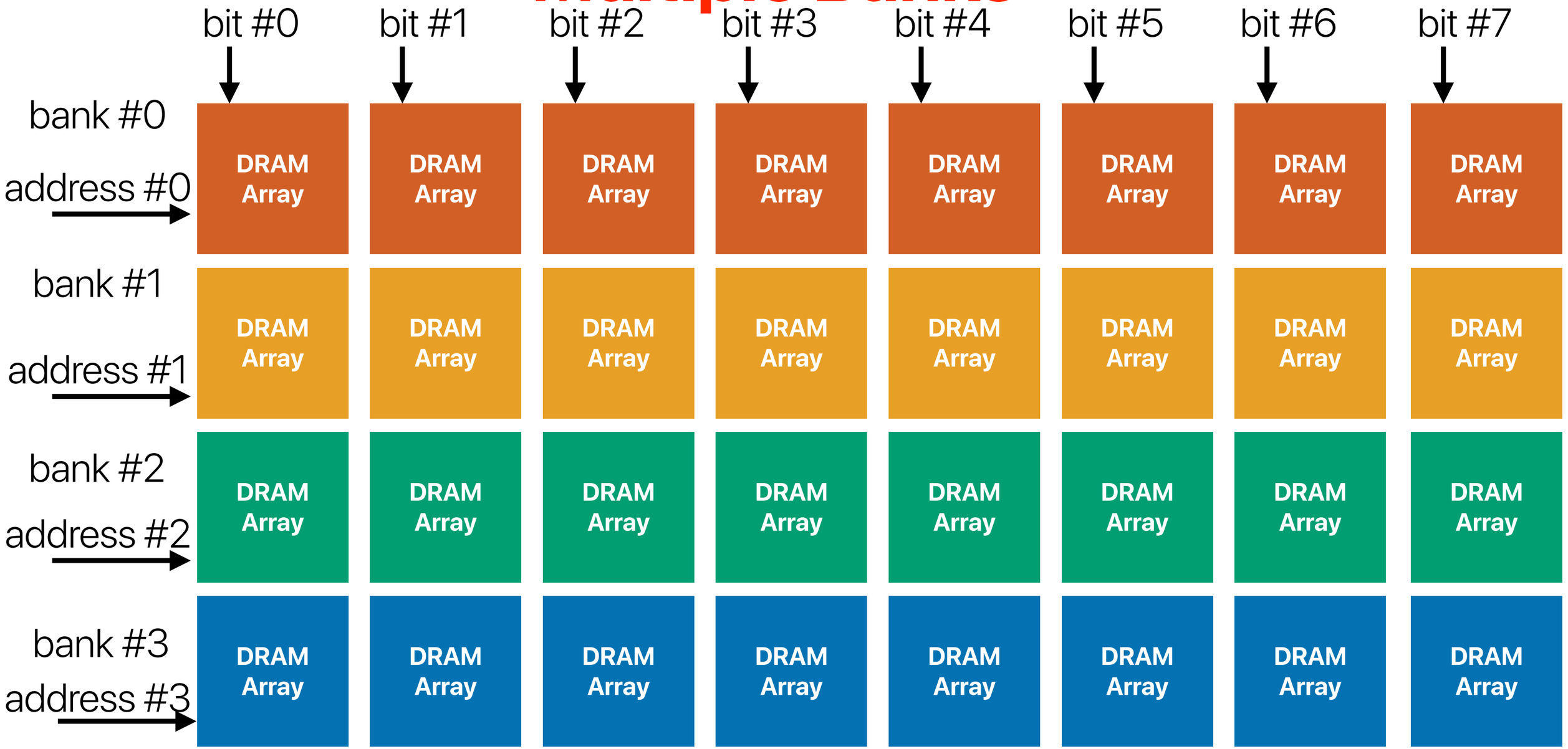
- Faster memory technologies
- Higher memory bandwidth
- Lower data volume

Outline

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

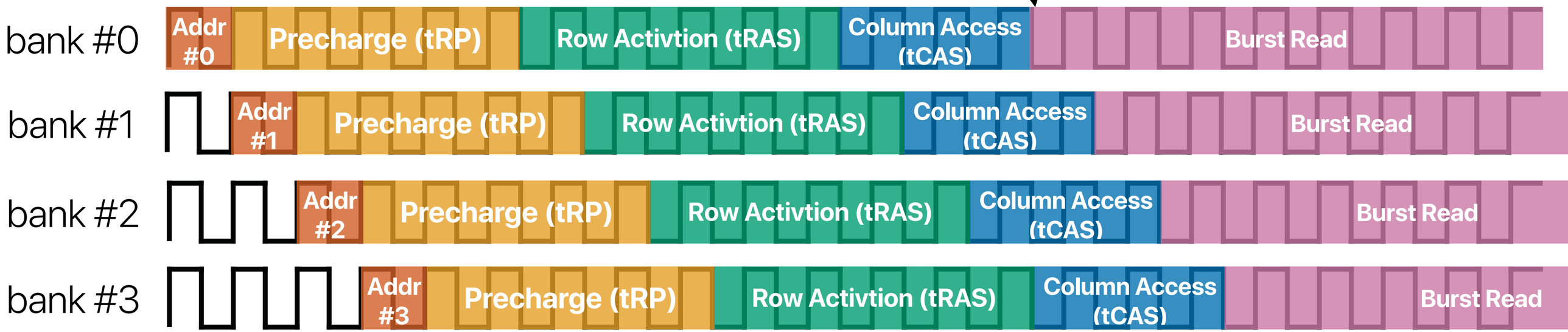

ODT
To ODT/output drivers
ZQ CAL
To ZQ Control
RESET_n
CKE
CK_t, CK_c
PAR
TEN
CS_n
ACT_n
Control logic
VrefDQ
BC4
OTF
CRC
Parity
2 (A12,A10)
Command decode RAS_n, CAS_n, WE_n
Mode registers
21
Row-address MUX
16
Refresh counter
16
2
2
BG and BA control logic
2
2
A[16:0]
BA[1:0]
BG[1:0]
21
Address register
Column address counter/ latch
7
3
Columns 0, 1 and 2
16
128 x64
Column decoder
Bank 0 BG0 Row-address latch and decoder
65,536
Bank 0 Bank Group 0
Memory array (65,536 x 128 x 64)
Sense amplifiers
8192
I/O gating DM mask logic
Global I/O gating
Bank 0 BG1
Bank 0 Bank Group 1
Bank 0 BG2
Bank 0 Bank Group 2
Bank 0 BG3
Bank 0 Bank Group 3
Bank 3
Bank 2
Bank 1
Bank 0
64
BC4
OTF
Columns 0, 1, and 2
64
READ FIFO and data MUX
8
CK_t,CK_c
64
Data interface
8
CRC and parity control
ZQ control
ODT control
ALERT
ZQ
CK_t,CK_c
DLL
Read drivers
BC4
DBI
CRC
Write drivers and input logic
Column 2 (BC4 nibble)
VrefDQ
VDDQ
RTTp
RTTn
RTTw
(0 . . . 7)
DQ[7:0]
DQS_t / DQS_c
DQS_t / DQS_c
TDQS_c
DBI_n / DM_n / TDQS_t

Multiple Banks



Multi-bank access

we can start output a “byte” from every 8 chips each cycle after this

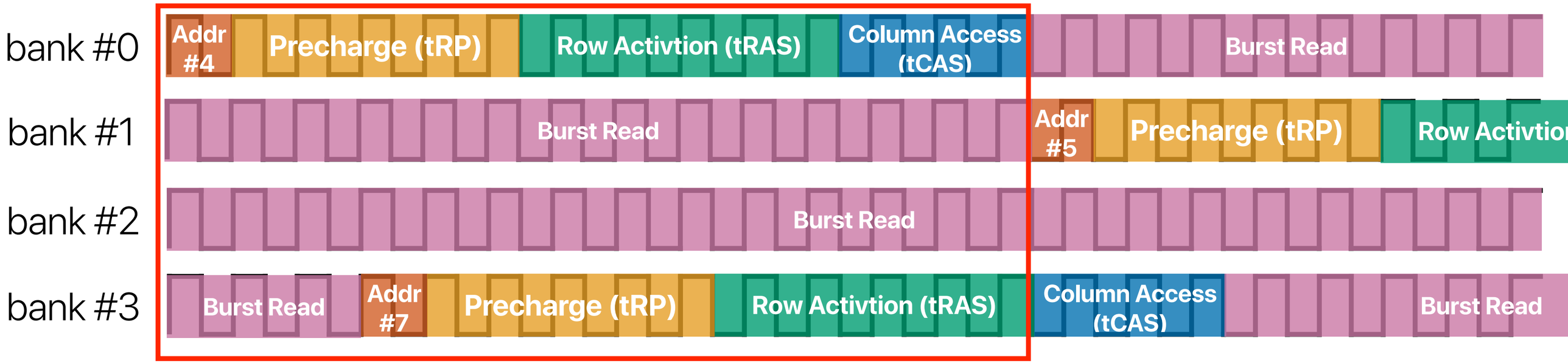


only one bank can accept request each cycle

the memory bandwidth can be fully utilized after this

Multi-bank access

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



DRAM Performance

- Latency per "8-bit" — 0.75 ns (if it's row-buffered)

- Bandwidth per die = $\frac{1}{0.75ns} = 1.33GB/sec$
- 16 chips = $16 \times \frac{1}{0.75ns} = 21.33GB/sec$

2. Key Features

[Table 2] 8Gb DDR4 C-die Speed bins

Speed	DDR4-1600	DDR4-1866	DDR4-2133	DDR4-2400	DDR4-2666		Unit
	11-11-11	13-13-13	15-15-15	17-17-17	19-19-19		
tCK(min)	1.25	1.071	0.937	0.833	0.75		ns
CAS Latency	11	13	15	17	19		nCK
tRCD(min)	13.75	13.92	14.06	14.16	14.25		ns
tRP(min)	13.75	13.92	14.06	14.16	14.25		ns
tRAS(min)	35	34	33	32	32		ns
tRC(min)	48.75	47.92	47.06	46.16	46.25		ns

- JEDEC standard 1.2V (1.14V~1.26V)
- V_{DDQ} = 1.2V (1.14V~1.26V)
- V_{PP} = 2.5V (2.375V~2.75V)
- 800 MHz f_{CK} for 1600Mb/sec/pin, 933 MHz f_{CK} for 1866Mb/sec/pin, 1067MHz f_{CK} for 2133Mb/sec/pin, 1200MHz f_{CK} for 2400Mb/sec/pin, 1333MHz f_{CK} for 2666Mb/sec/pin
- 8 Banks (2 Bank Groups)
- Programmable CAS Latency (posted CAS): 10,11,12,13,14,15,16,17,18,19,20
- Programmable CAS Write Latency (CWL) = 9,11 (DDR4-1600), 10,12 (DDR4-1866),11,14 (DDR4-2133),12,16 (DDR4-2400) and 14,18 (DDR4-2666)
- 8-bit pre-fetch
- Burst Length: 8, 4 with tCCD = 4 which does not allow seamless read or write [either On the fly using A12 or MRS]
- Bi-directional Differential Data-Strobe
- Internal (self) calibration: Internal self calibration through ZQ pin (RZQ: 240 ohm ± 1%)
- On Die Termination using ODT pin
- Average Refresh Period 7.8us at lower than T_{CASE} 85°C, 3.9us at 85°C < T_{CASE} ≤ 95 °C
- Connectivity Test Mode (TEN) is Supported

The 8Gb DDR4 SDRAM C-die is organized as a 64Mbit x 16 I/Os x 8banks device. This synchronous device achieves high speed double-data-rate transfer rates of up to 2666Mb/sec/pin (DDR4-2666) for general applications.

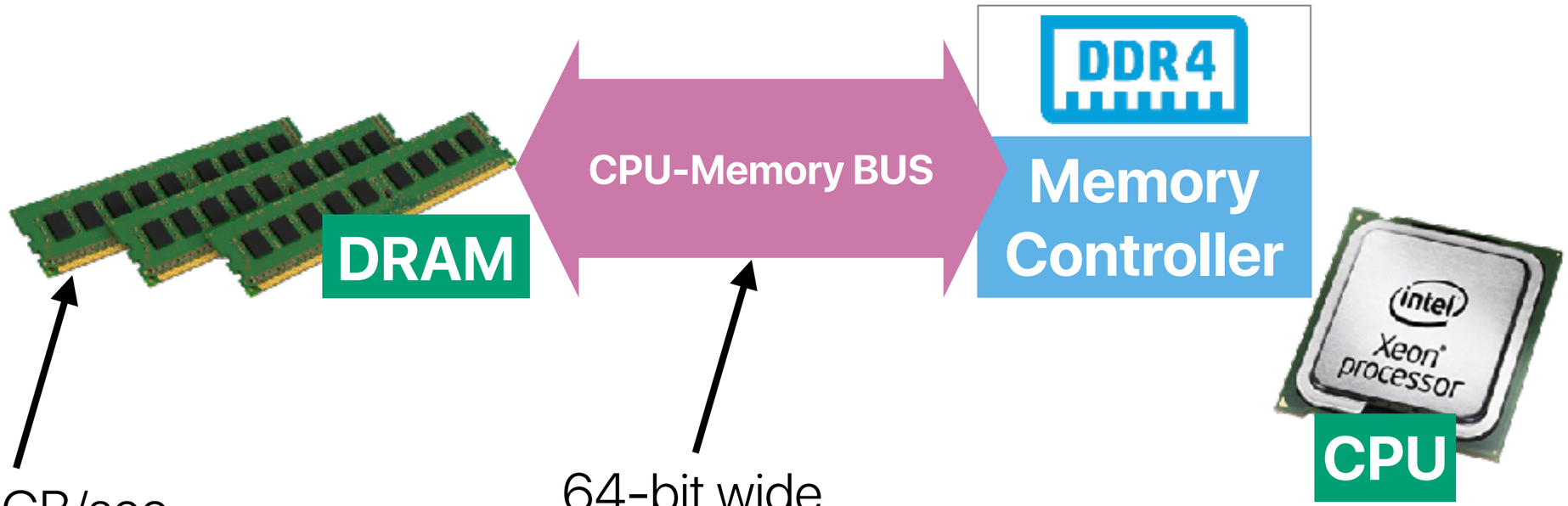
The chip is designed to comply with the following key DDR4 SDRAM features such as posted CAS, Programmable CWL, Internal (Self) Calibration, On Die Termination using ODT pin and Asynchronous Reset.

All of the control and address inputs are synchronized with a pair of externally supplied differential clocks. Inputs are latched at the crosspoint of differential clocks (CK rising and $\overline{CK}$ falling). All I/Os are synchronized with a pair of bidirectional strobes (DQS and $\overline{DQS}$) in a source synchronous fashion. The address bus is used to convey row, column, and bank address information in a $\overline{RAS}$/$\overline{CAS}$ multiplexing style. The DDR4 device operates with a single 1.2V (1.14V~1.26V) power supply, 1.2V(1.14V~1.26V) V_{DDQ} and 2.5V (2.375V~2.75V) V_{PP}.

The 8Gb DDR4 C-die device is available in 96ball FBGAs(x16).

https://semiconductor.samsung.com/resources/user-manual/x16%20only_8G_C_DDR4_Samsung_Spec_Rev1.5_Apr.17.pdf

How fast is the memory controller frequency?



To support such a data rate, the controller needs to operates at

$$\frac{21.33GB/sec}{64-bit} = \frac{21.33GB}{8B} = 2.667GHz$$

3:00

What’s the bandwidth of DDR5-5600?

DDR5-5600

To support such a data rate, the controller needs to operates at

$$\frac{x\ GB/sec}{64-bit} = \frac{x\ GB/sec}{8B} = 5.6GHz$$

$$x = 8 \times 5.6GHz = 44.8GB/sec$$

Memory bandwidth

3:00

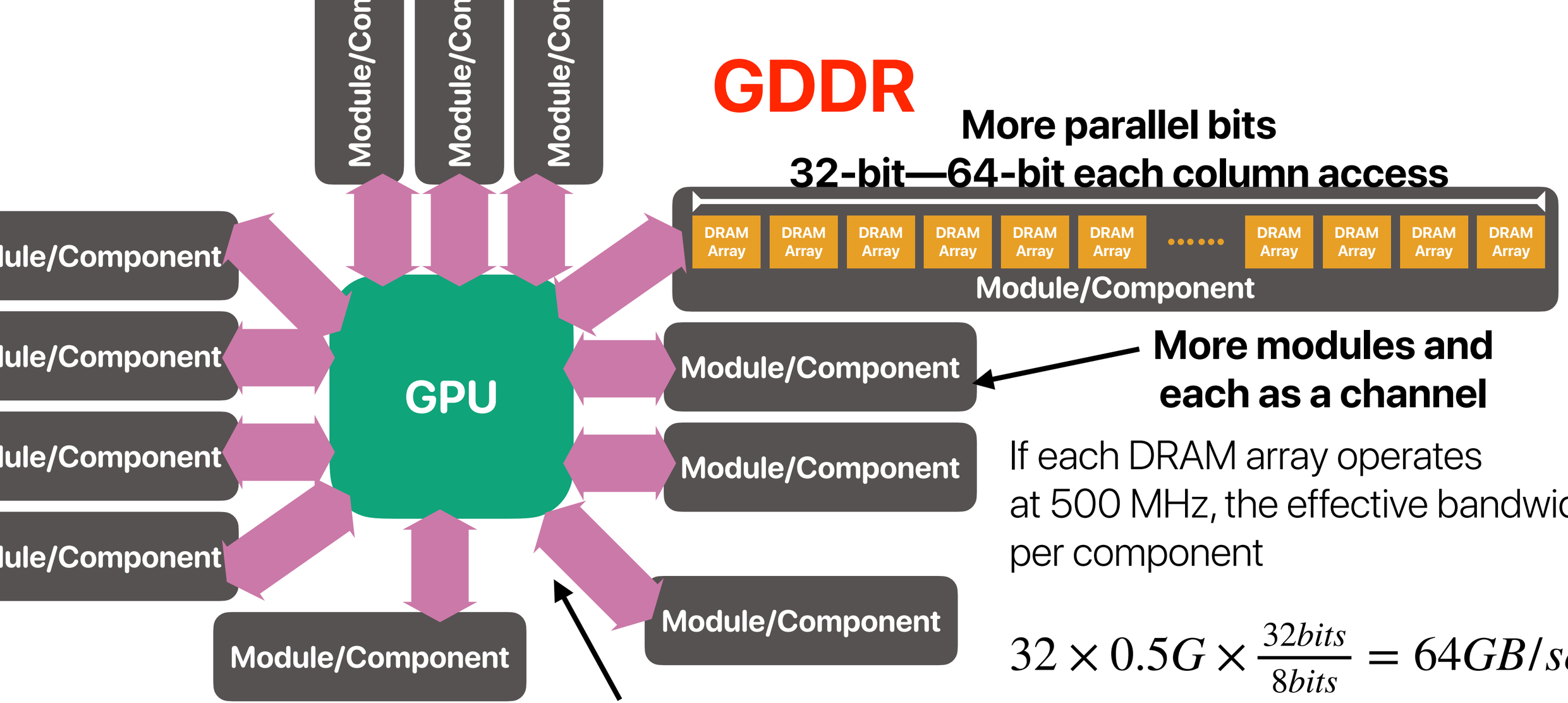
How can you increase the DRAM bandwidth?

Ideas of increasing bandwidth

Ideas of increasing bandwidth

- More parallel bits
- More banks
- More channels
- Widen the memory-processor bus

GDDR



More parallel bits
32-bit—64-bit each column access

More modules and each as a channel

If each DRAM array operates at 500 MHz, the effective bandwidth per component

$$32 \times 0.5G \times \frac{32bits}{8bits} = 64GB/sec$$

Wider bus: 192-bit — 384-bit bus

If the bus is 256-bit (32B) wide, the memory controller needs to be at

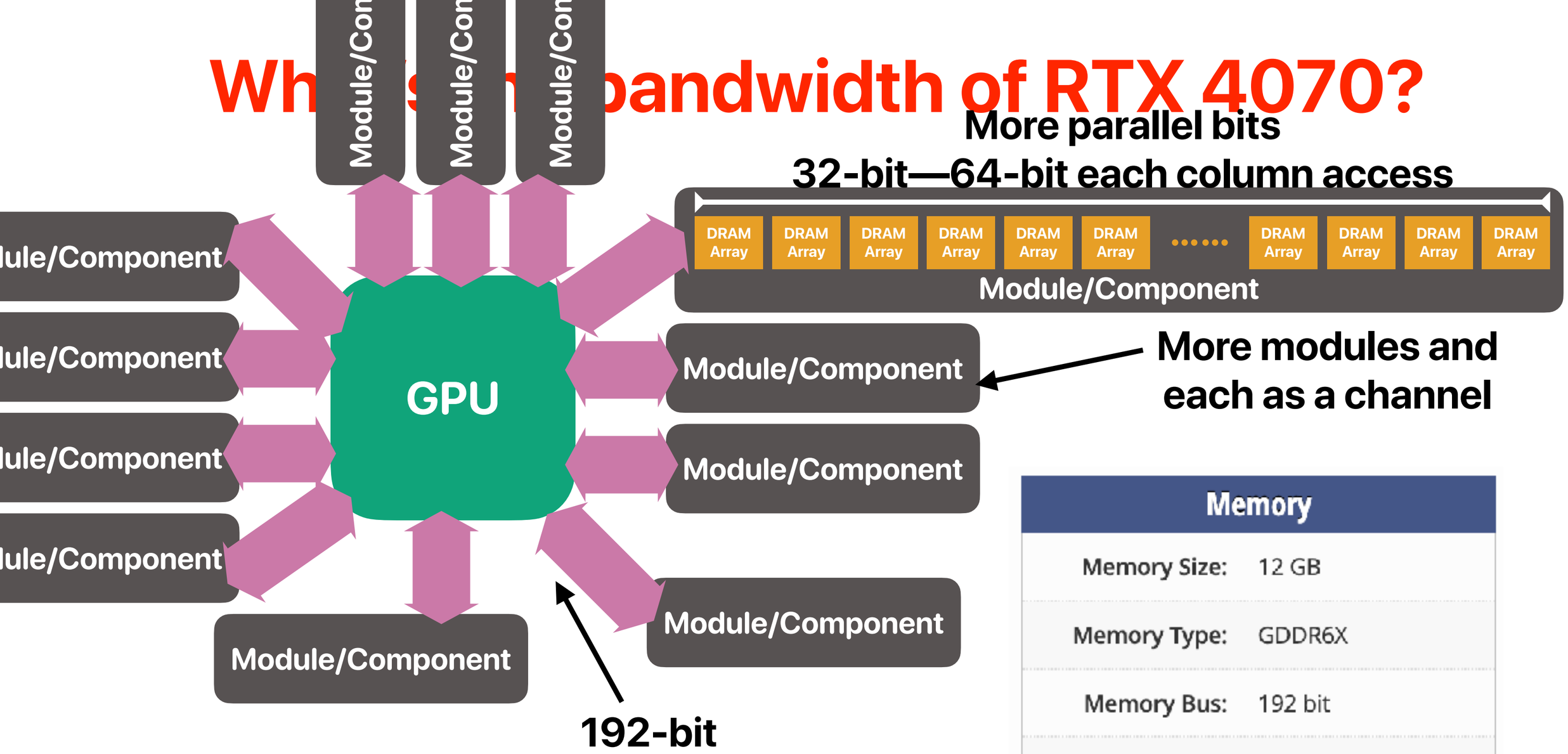
$$\frac{64GB/sec}{32B} = 2GHz$$

If you have 12x modules — 12*64 = 768 GB/sec

What's the bandwidth of RTX 4070?

	GeForce RTX 4070 Ti SUPER	GeForce RTX 4070 Ti	GeForce RTX 4070 SUPER	GeForce RTX 4070
GPU Engine Specs:				
NVIDIA CUDA® Cores	8448	7680	7168	5888
Boost Clock (GHz)	2.61	2.61	2.48	2.48
Base Clock (GHz)	2.34	2.31	1.98	1.92
Memory Specs:				
Standard Memory Config	16 GB GDDR6X	12 GB GDDR6X	12 GB GDDR6X	12 GB GDDR6X
Memory Interface Width	256-bit	192-bit	192-bit	192-bit

Wh[illegible] bandwidth of RTX 4070?



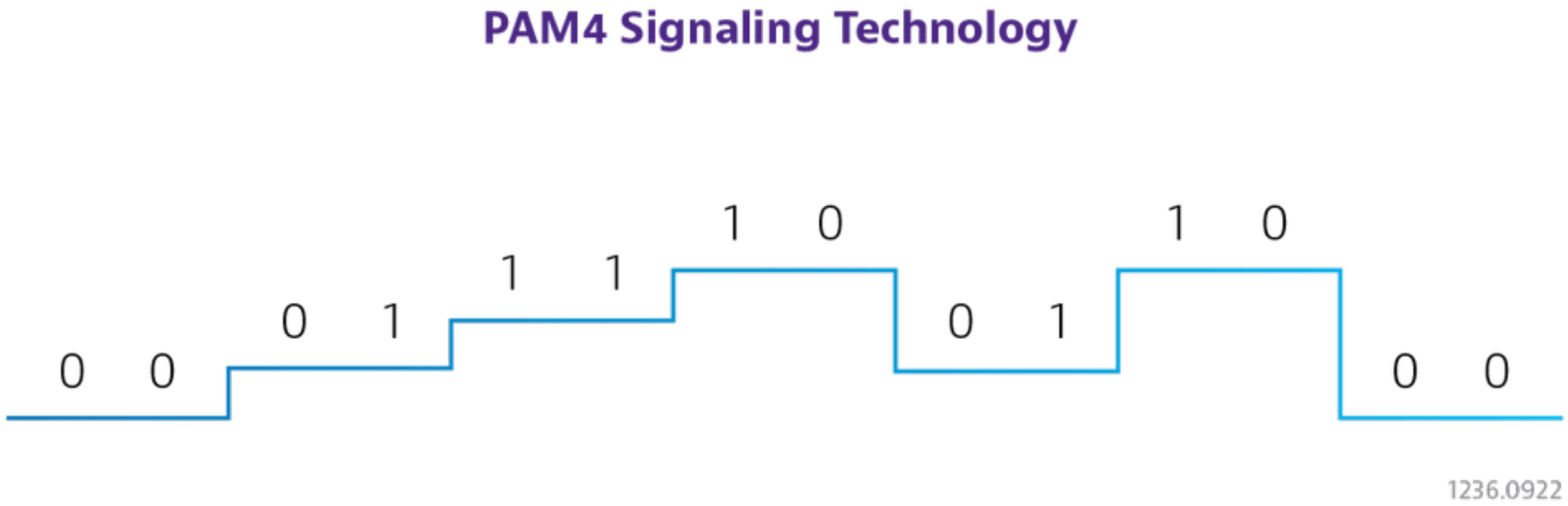
Memory	
Memory Size:	12 GB
Memory Type:	GDDR6X
Memory Bus:	192 bit
Bandwidth:	504.2 GB/s

If the bus is **192**-bit (24B) wide, bandwidth is

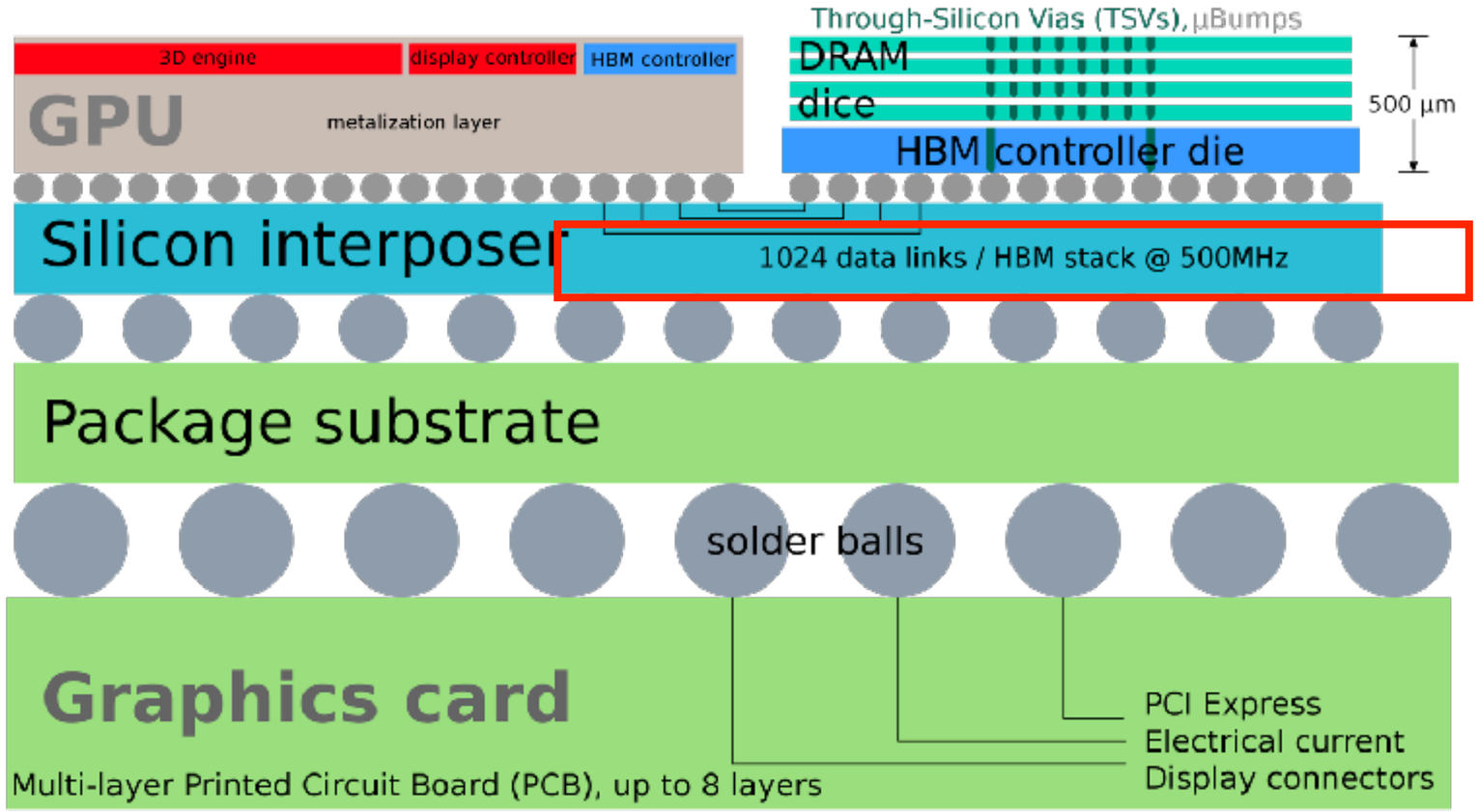
$$\frac{192}{8} \times 10501 \times 10^6 = 252024MB/sec = 252GB/sec$$

https://www.techpowerup.com/gpu-specs/geforce-rtx-4070.c3924

Signaling in DDR6X



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

3:00

What’s the pros & cons of GDDR v.s. HBM

GDDR v.s. HBM

GDDR v.s. HBM

- HBM’s bandwidth is wider
- HBM is more expensive — pin counts are more expensive
- HBM is limited by the per-chip heat dissipation — less powerful

The program

3:00

What're the differences between the 1st and the rest runs?

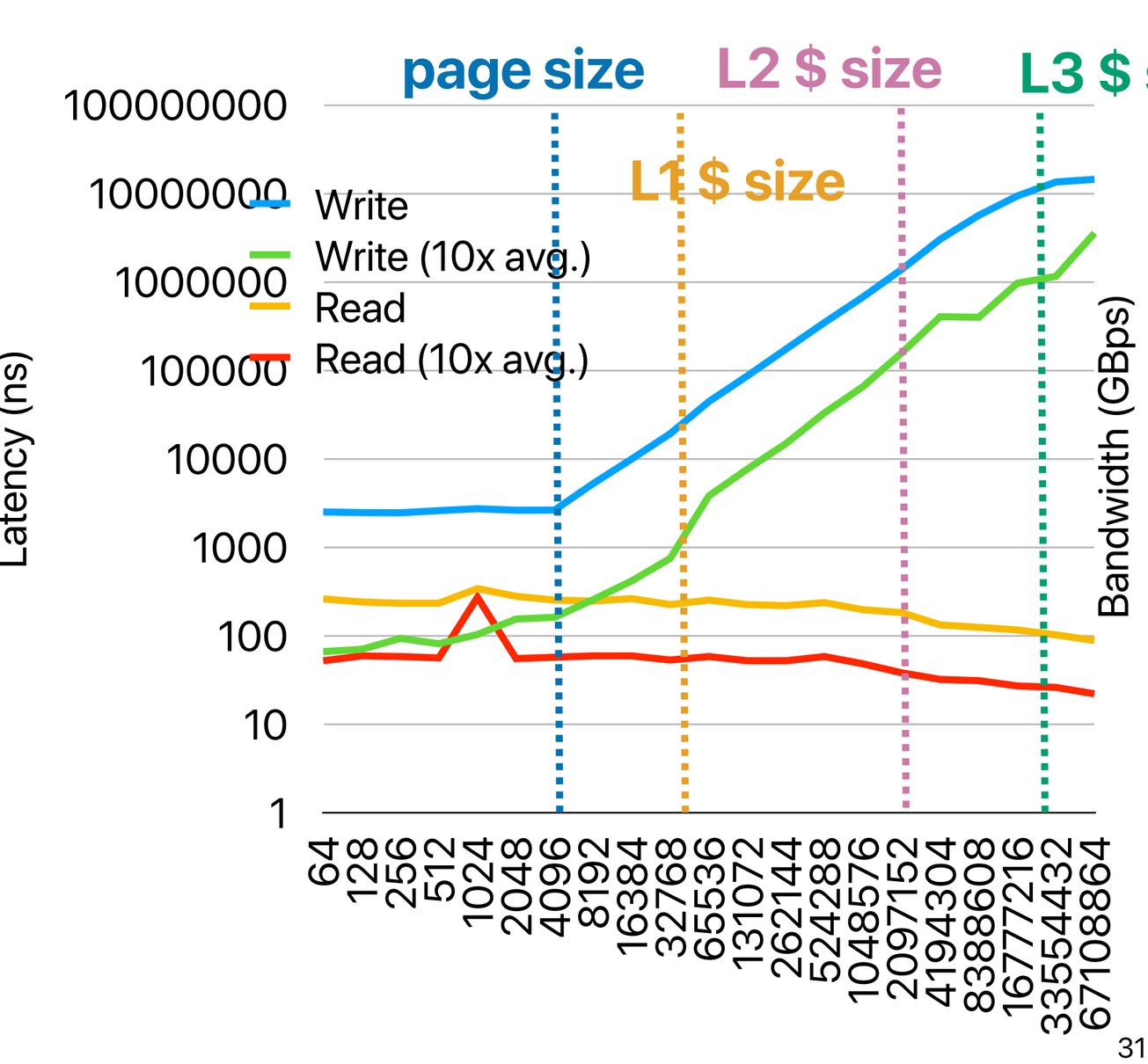
What's the purpose of using AVX/Vector Instructions here?

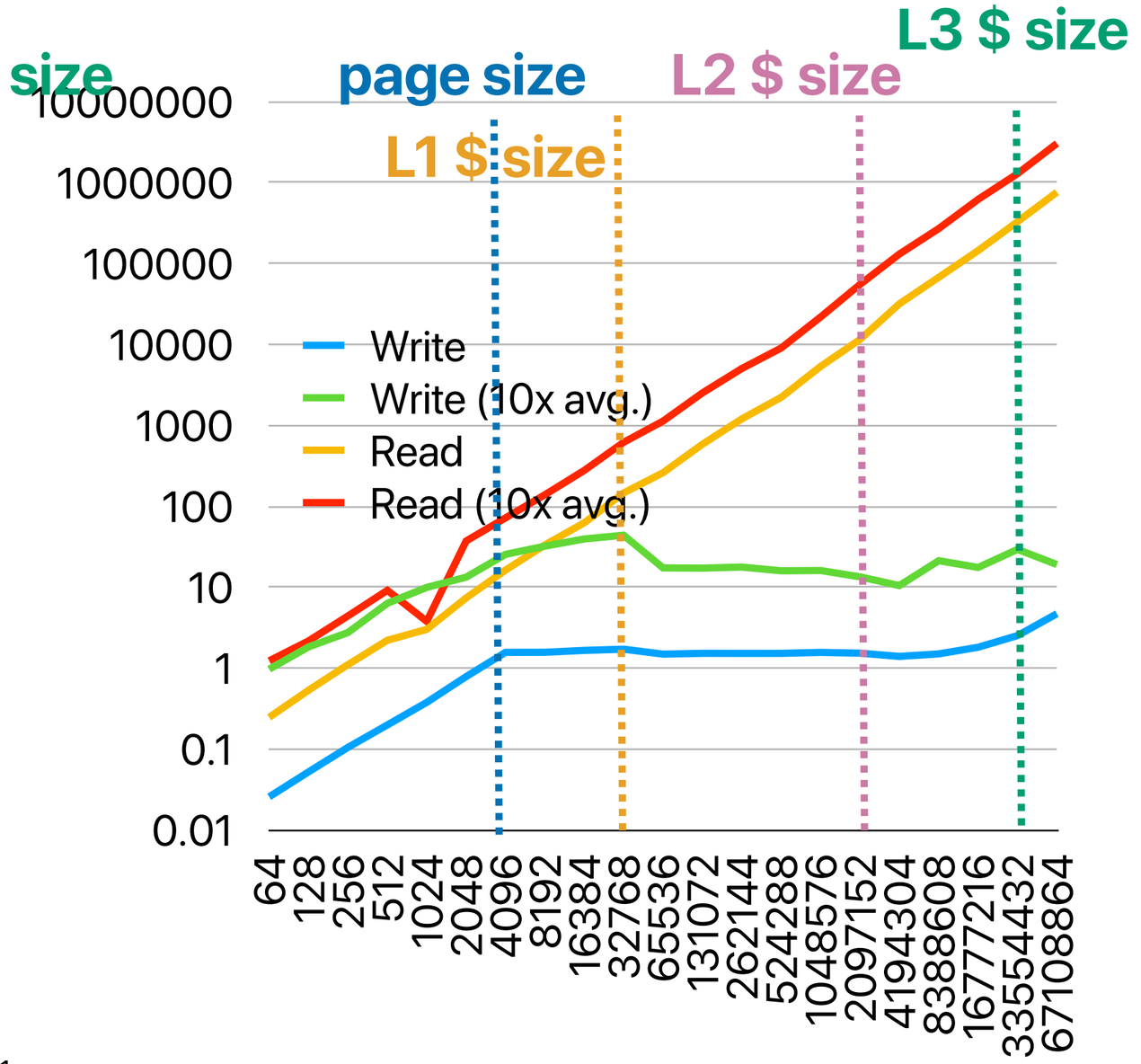
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>    /* for clock_gettime */
#include <malloc.h>
#include <immintrin.h>

void write_memory_avx(void* array, size_t size) {
  __m256i* varray = (__m256i*) array;

  __m256i vals = _mm256_set1_epi32(1);
  size_t i;
  for (i = 0; i < size / sizeof(__m256i); i++) {
    _mm256_store_si256(&varray[i], vals);
// This will generate the vmovaps instruction.
  }
}
int main(int argc, char **argv)
{
    size_t *array;
    size_t size;
    double total_time;
    struct timespec start, end;
    int i;
    size = atoi(argv[1])/sizeof(size_t);
    array = (size_t *)memalign(32,sizeof(size_t)*size);

    clock_gettime(CLOCK_MONOTONIC, &start);

    write_memory_avx(array, size);

    clock_gettime(CLOCK_MONOTONIC, &end);

    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\n",total_time, (double)((double)size*sizeof(size_t)/
(total_time)));
    clock_gettime(CLOCK_MONOTONIC, &start);

for(i = 0 ; i< 10; i++)
    write_memory_avx(array, size);

    clock_gettime(CLOCK_MONOTONIC, &end);
    total_time = ((end.tv_sec * 1000000000.0 +
end.tv_nsec) - (start.tv_sec * 1000000000.0 +
start.tv_nsec));
    fprintf(stderr,"Latency (10x average): %.0lf ns, GBps
(10x average): %lf\n",total_time/10, (double)
((double)size*10*sizeof(size_t)/(total_time)));
    return 0;
}
```

The program

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


void write_memory_avx(void* array, size_t size) {
  __m256i* varray = (__m256i*) array;

  __m256i vals = _mm256_set1_epi32(1);
  size_t i;
  for (i = 0; i < size / sizeof(__m256i); i++) {
    _mm256_store_si256(&varray[i], vals);  // This will
generate the vmovaps instruction.
  }
}

int main(int argc, char **argv)
{
    size_t *array;
    size_t size;
    double total_time;
    struct timespec start, end;
    int i;
    size = atoi(argv[1])/sizeof(size_t);
    array = (size_t *)memalign(32,sizeof(size_t)*size);
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start
time */
    write_memory_avx(array, size);

    clock_gettime(CLOCK_MONOTONIC, &end); /* mark the end
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\n",total_time, (double)((double)size*sizeof(size_t)/
(total_time)));
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start
time */
    for(i = 0 ; i< 10; i++)
        write_memory_avx(array, size);

    clock_gettime(CLOCK_MONOTONIC, &end); /* mark the end
time */
    total_time = ((end.tv_sec * 1000000000.0 +
end.tv_nsec) - (start.tv_sec * 1000000000.0 +
start.tv_nsec));
    fprintf(stderr,"Latency (10x average): %.0lf ns, GBps
(10x average): %lf\n",total_time/10, (double)
((double)size*10*sizeof(size_t)/(total_time)));
    return 0;
}
```

Performance Chart (on Intel Core i7 13700)





The program

```c
#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

3:00

Why is memcpy so often?

Network protocol stack

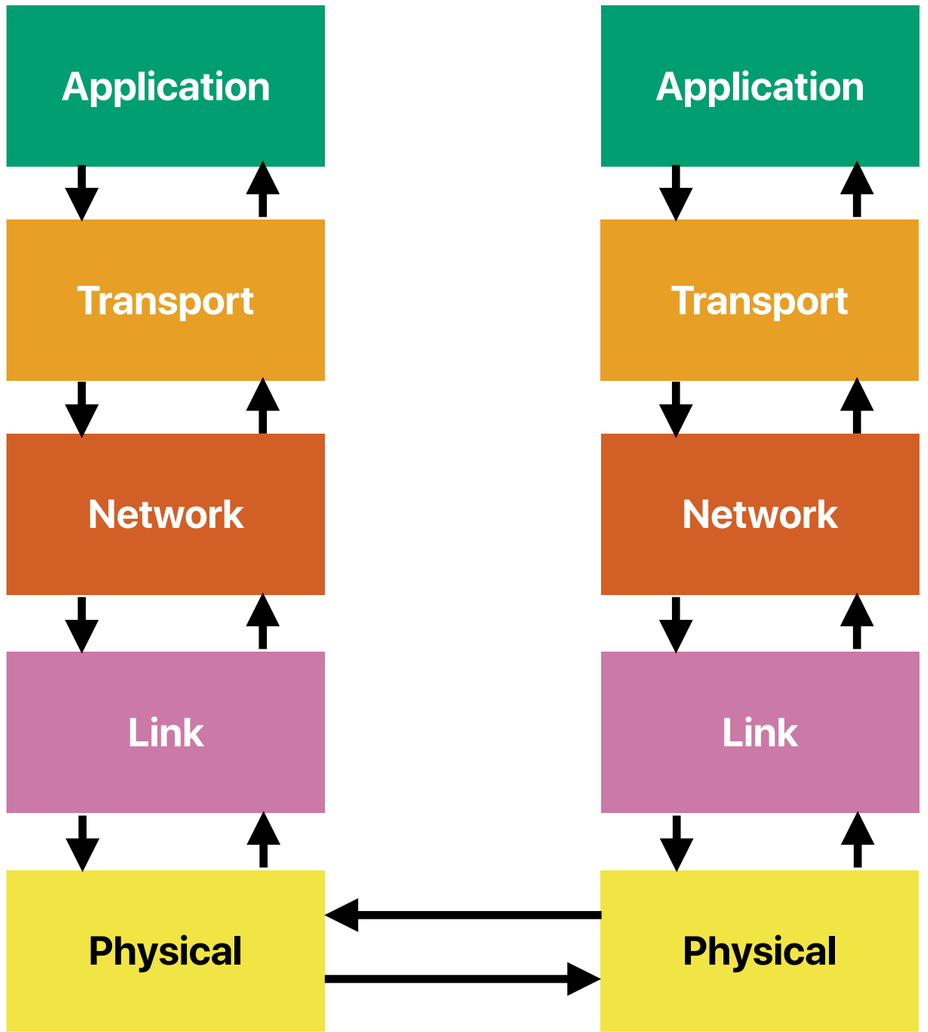
How do applications(e.g., server/client) interpret data: HTTPS, FTP, SSH, RTSP …

How do applications connect to each other (end-to-end reliability): TCP, UDP, RTP

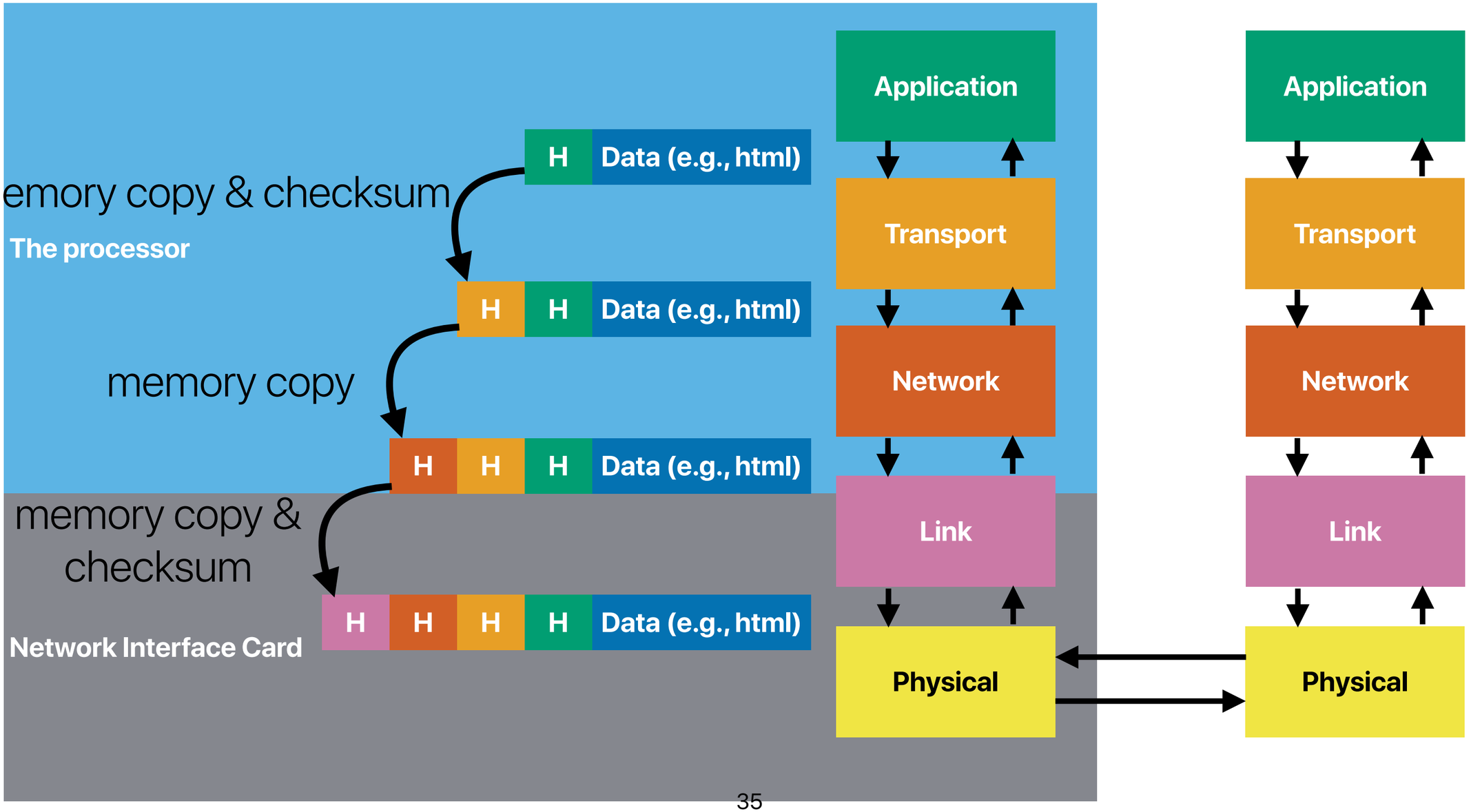
How do computers find each other: IP, ARP, ICMP

How do a computer use the media:
IEEE 802.3 (ethernet), IEEE 802.11 (WiFi)

How do the computer encode data on the media



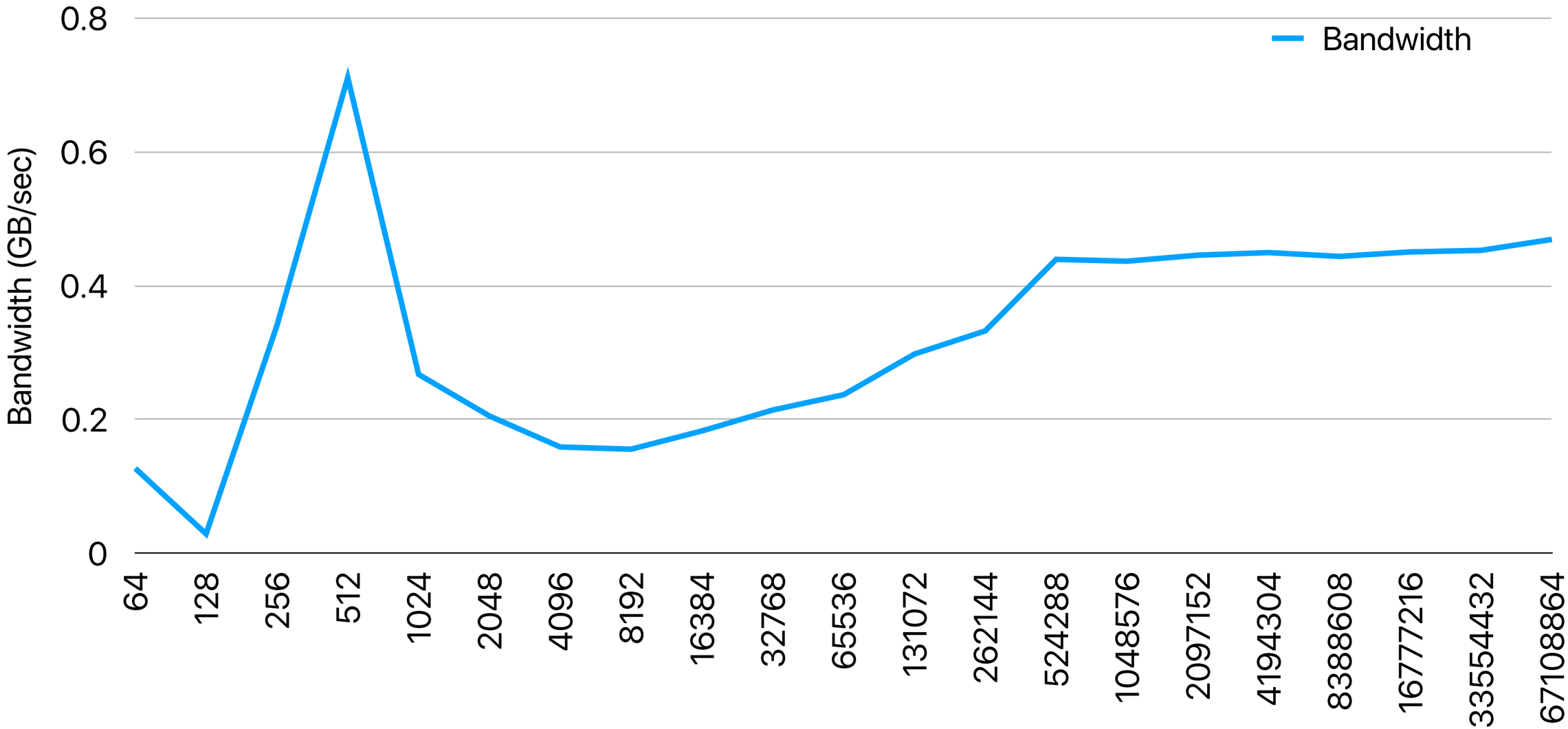
Network protocol stack



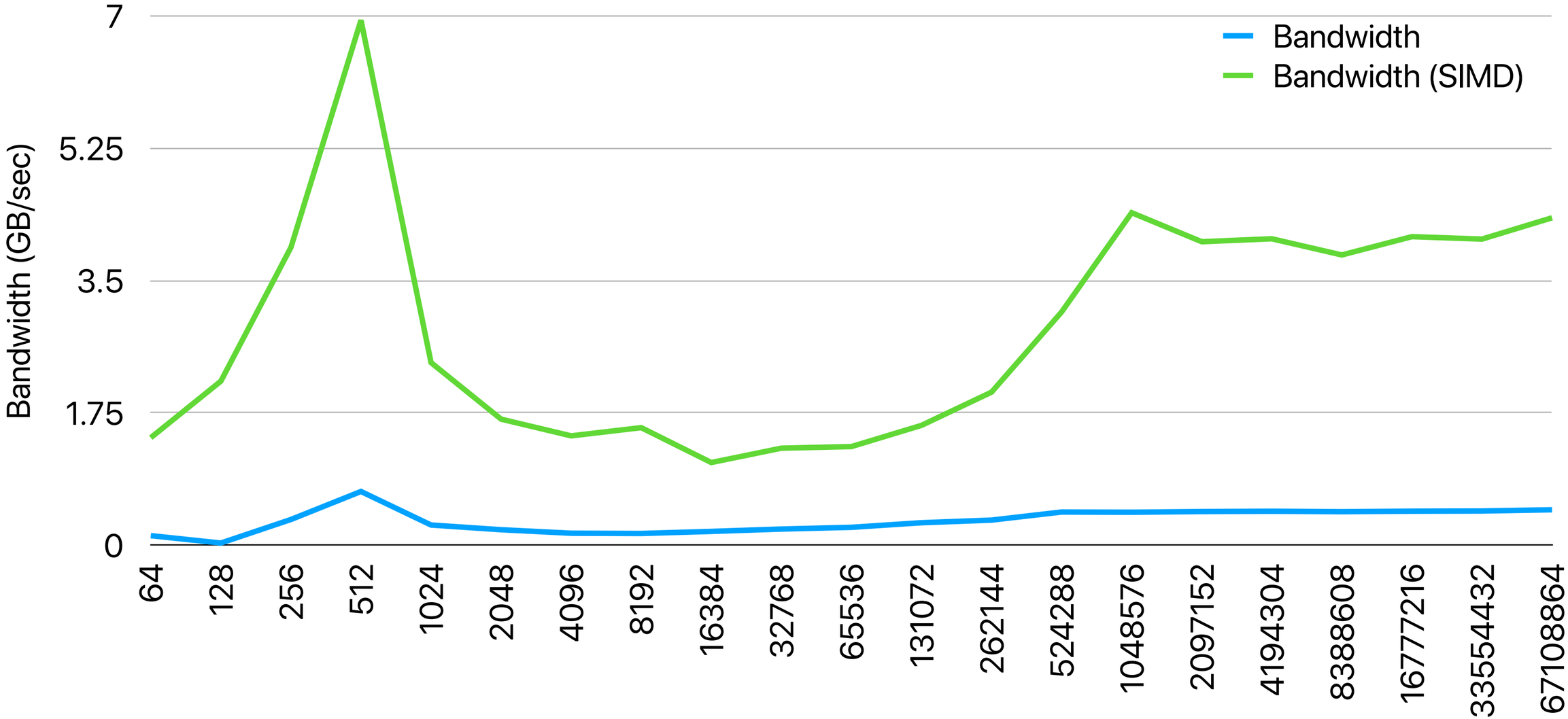
Use cases of memcpy

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

Memory copy bandwidth



Memory copy bandwidth using SIMD



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