

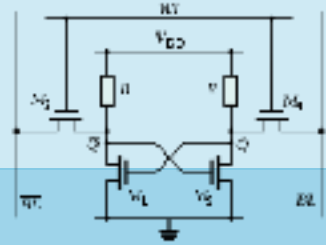
# Memory subsystem (4) & In/near-memory processing

Hung-Wei Tseng

# Recap: Memory technologies we have today

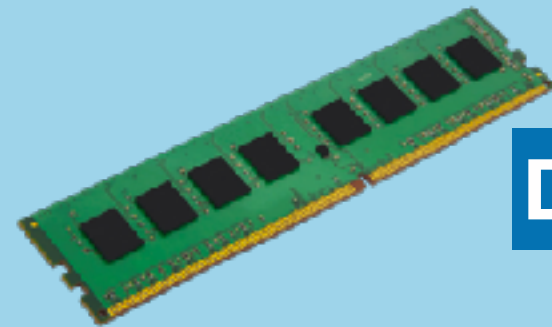
## Volatile Memory

100ps



SRAM

ns



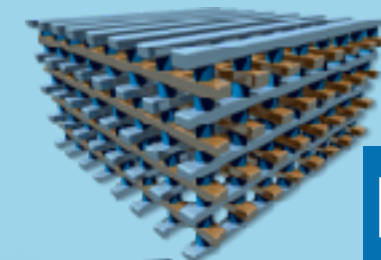
DRAM

us

ms

## Non-Volatile Memory

RRAM



PCM

3DXPoint



Flash memory



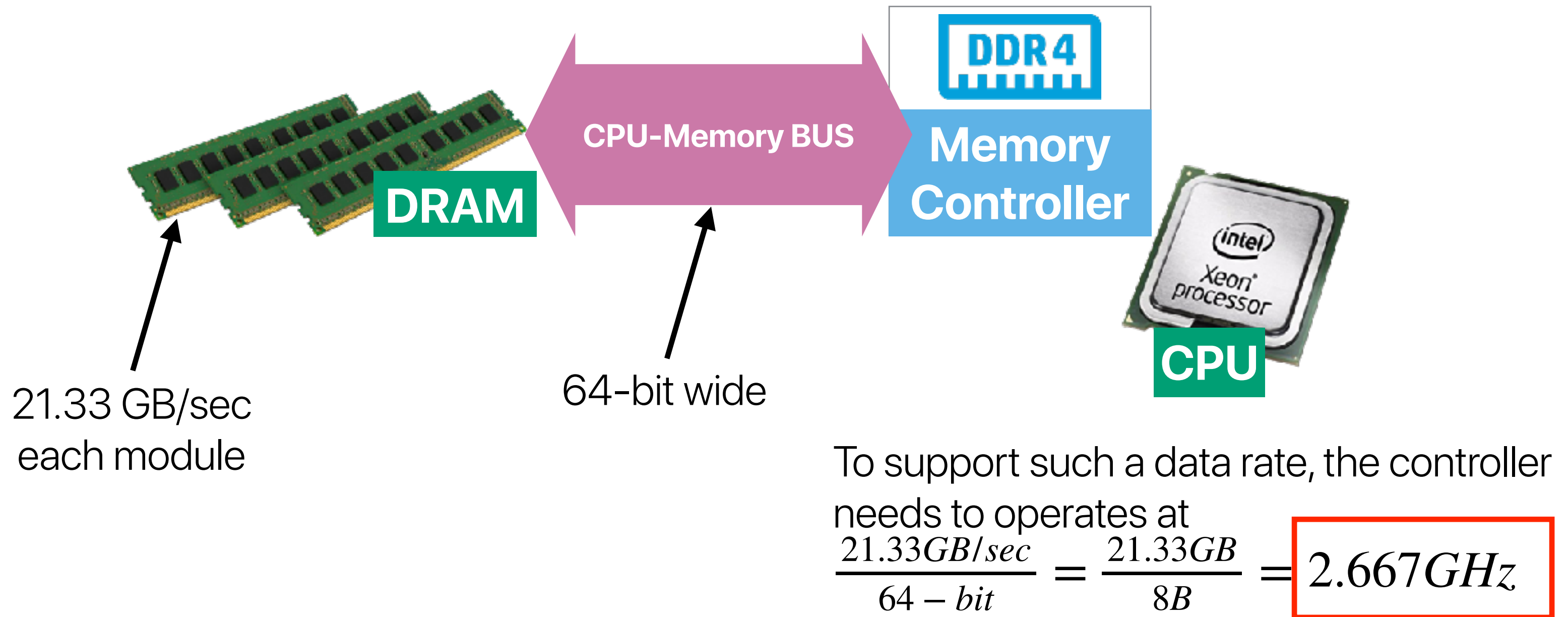
Hard Disk Drives





# Recap: Improving memory-bounded applications

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

# How fast is the memory controller frequency?



# CPU matters

Export comparison  			Intel® Xeon® Gold 6140 Pro...	Intel® Xeon® Silver 4108 Pr...
Total Threads	36	16		
Max Turbo Frequency	3.70 GHz	3.00 GHz		
Processor Base Frequency	2.30 GHz	1.80 GHz		
Cache	24.75 MB L3 Cache	11 MB L3 Cache		
Max # of UPI Links	3	2		
TDP	140 W	85 W		
Max Memory Size (dependent on memory type)	768 GB	768 GB		
Memory Types	DDR4-2666	DDR4-2400		
Maximum Memory Speed	2666 MHz	2400 MHz		
Max # of Memory Channels	6	6		
ECC Memory Supported <sup>†</sup>	Yes	Yes		

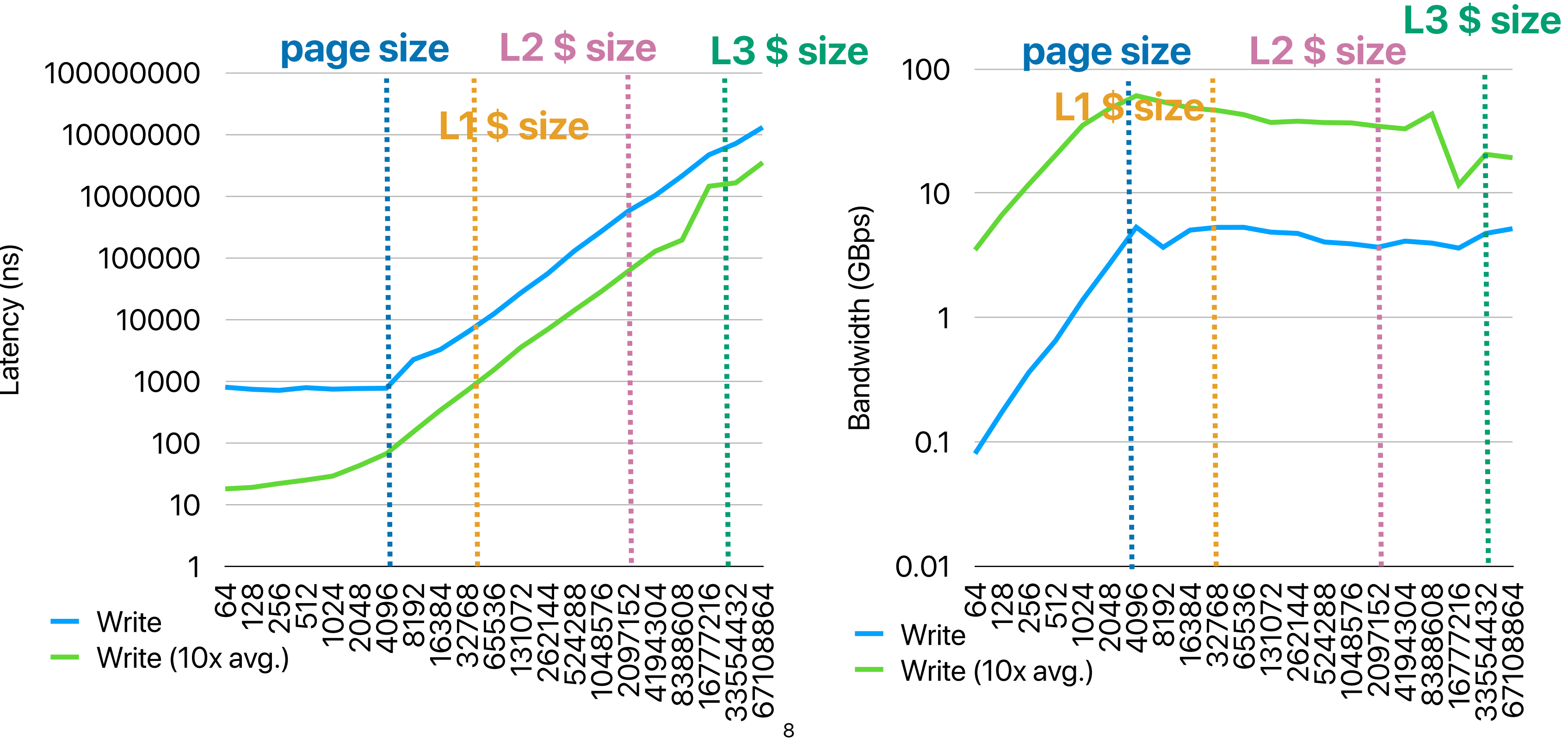
# If we ramp up the controller speed — DDR5-5600

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

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

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

# Recap: Performance Chart (on Intel Core i7 13700)



# Compare these two versions, what are the differences and implications to performance?

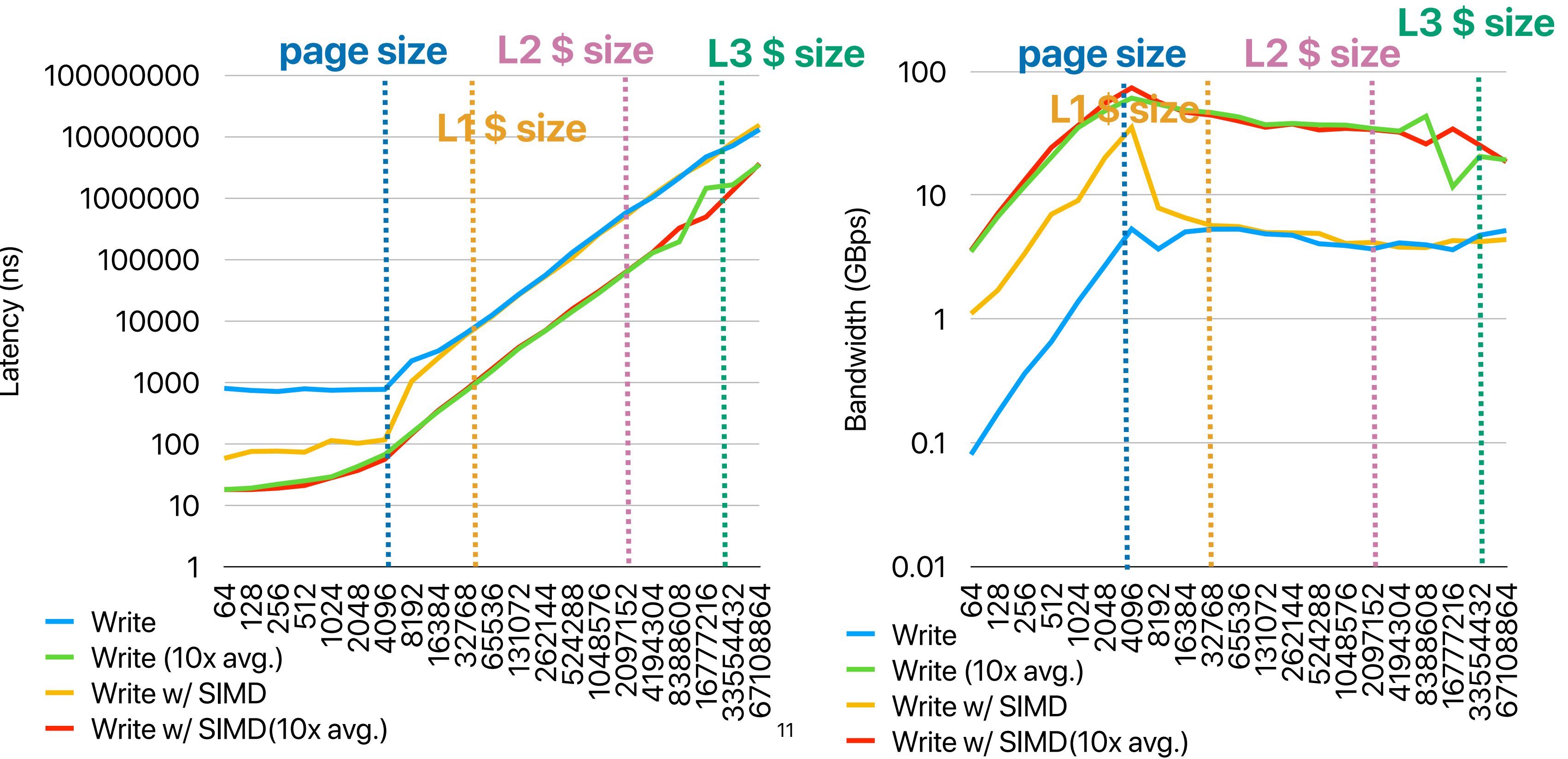
```
void write_memory_avx(void* array, size_t size) {
    __m256i* varray = (__m256i*) array;

    __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.
    }
}
```

```
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;
    }
}
```



# Performance Chart (on Intel Core i7 13700)



# If we ramp up the controller speed — DDR5

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

$$\frac{x \text{ GB/sec}}{64 - \text{bit}} = \frac{x \text{ GB/sec}}{8B} = 3.6\text{GHz}$$

$$x = 8 \times 3.6\text{GHz} = 28.8\text{GB/sec}$$



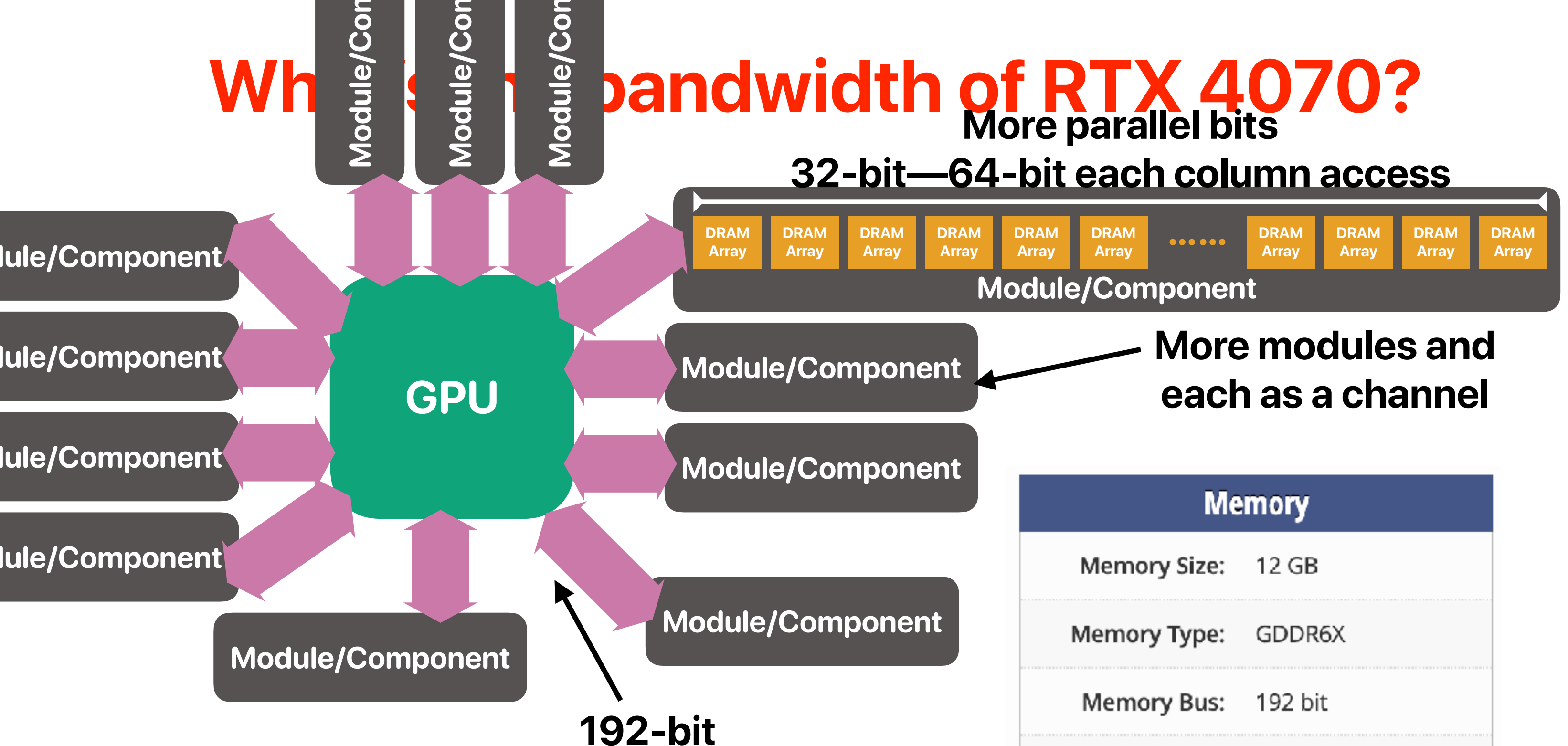
# What's the bandwidth of RTX 4070?

	GeForce RTX 4070 Ti SUPER	GeForce RTX 4070 Ti	GeForce RTX 4070 SUPER	GeForce RTX 4070
<b>GPU Engine Specs:</b>				
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
<b>Memory Specs:</b>				
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

# Why is the bandwidth of RTX 4070?

More parallel bits

32-bit—64-bit each column access



More modules and each as a channel

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

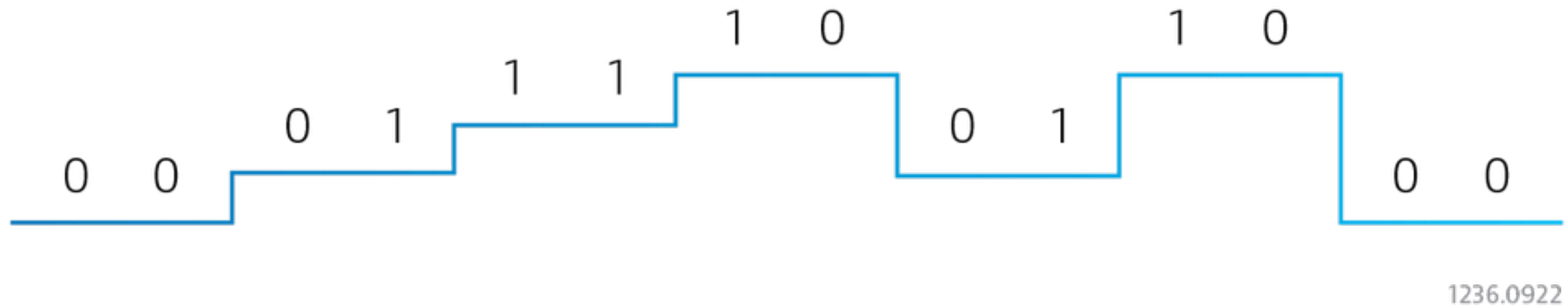
If the bus is **192**-bit (24B) wide, bandwidth is

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

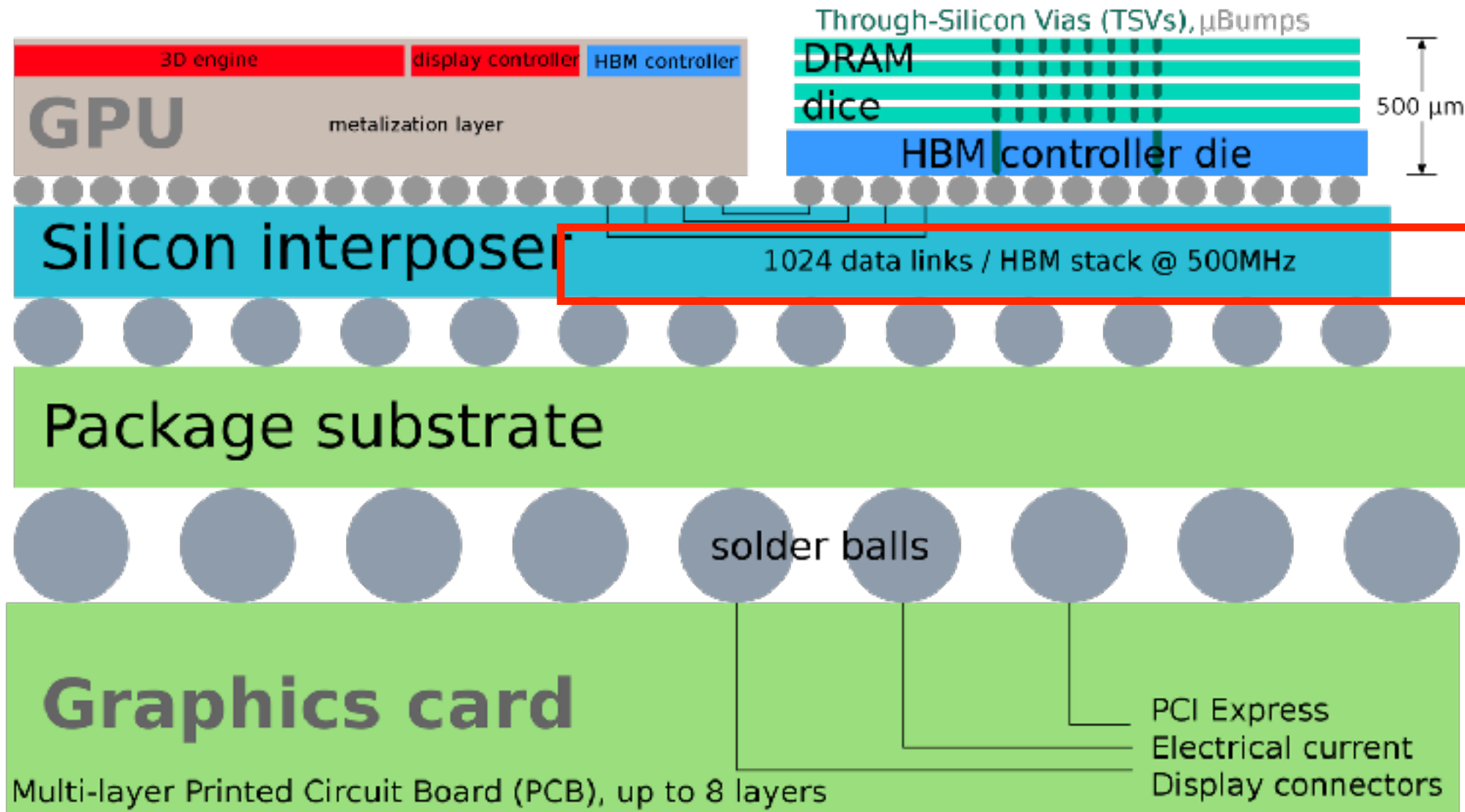
<https://www.techpowerup.com/gpu-specs/geforce-rtx-4070.c3924>

# Signaling in DDR6X

## PAM4 Signaling Technology



# 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**

# **What's the pros & cons of 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

# Case: what's the goal of the following program and what do you expect the performance would change when size varies?

```
#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_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 *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 */

    write_memory_avx(array, size);

    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;
}
```

# 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_avx(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 */

    write_memory_avx(array, size);

    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

**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



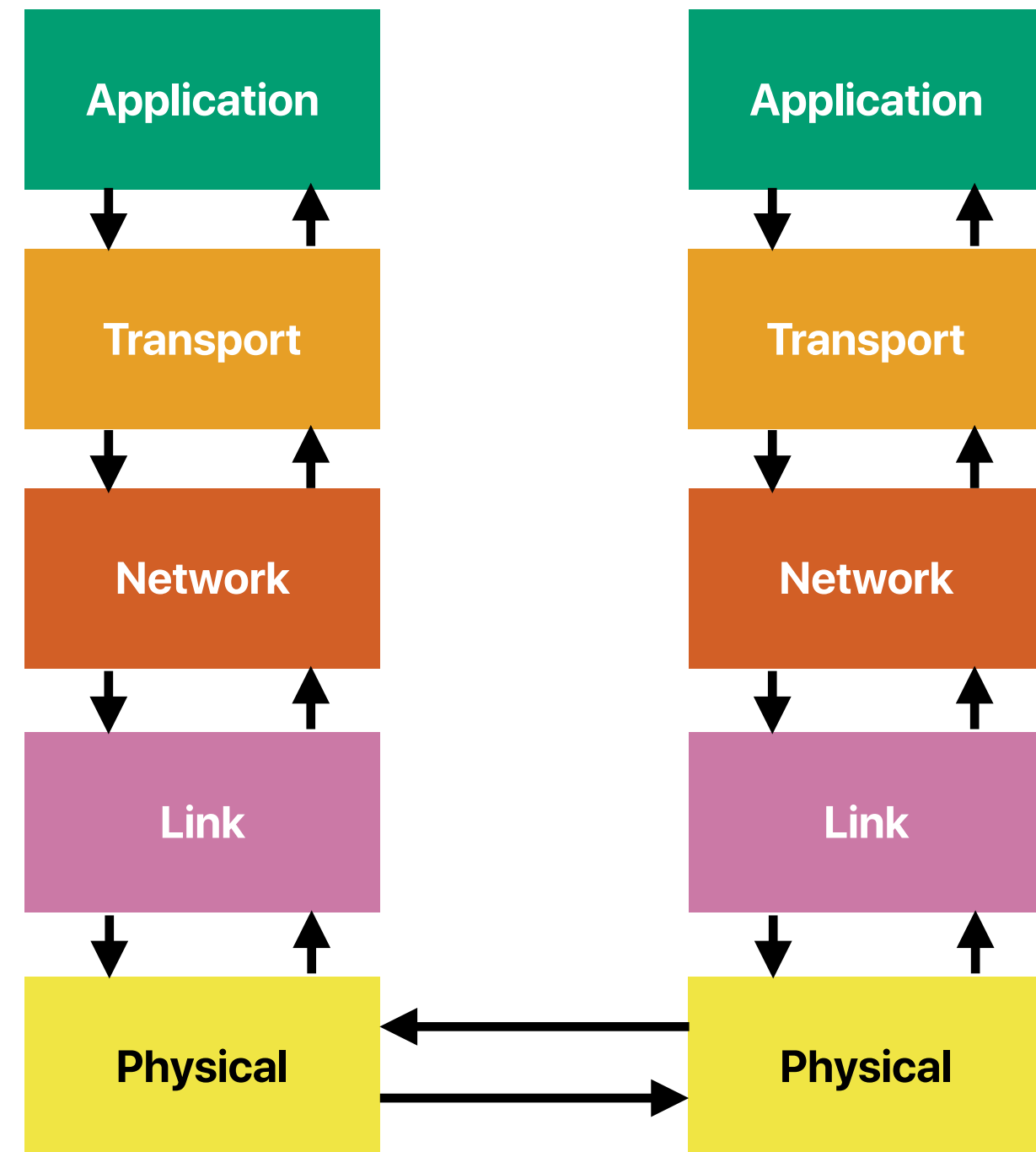
How do computers find each other: IP, ARP, ICMP



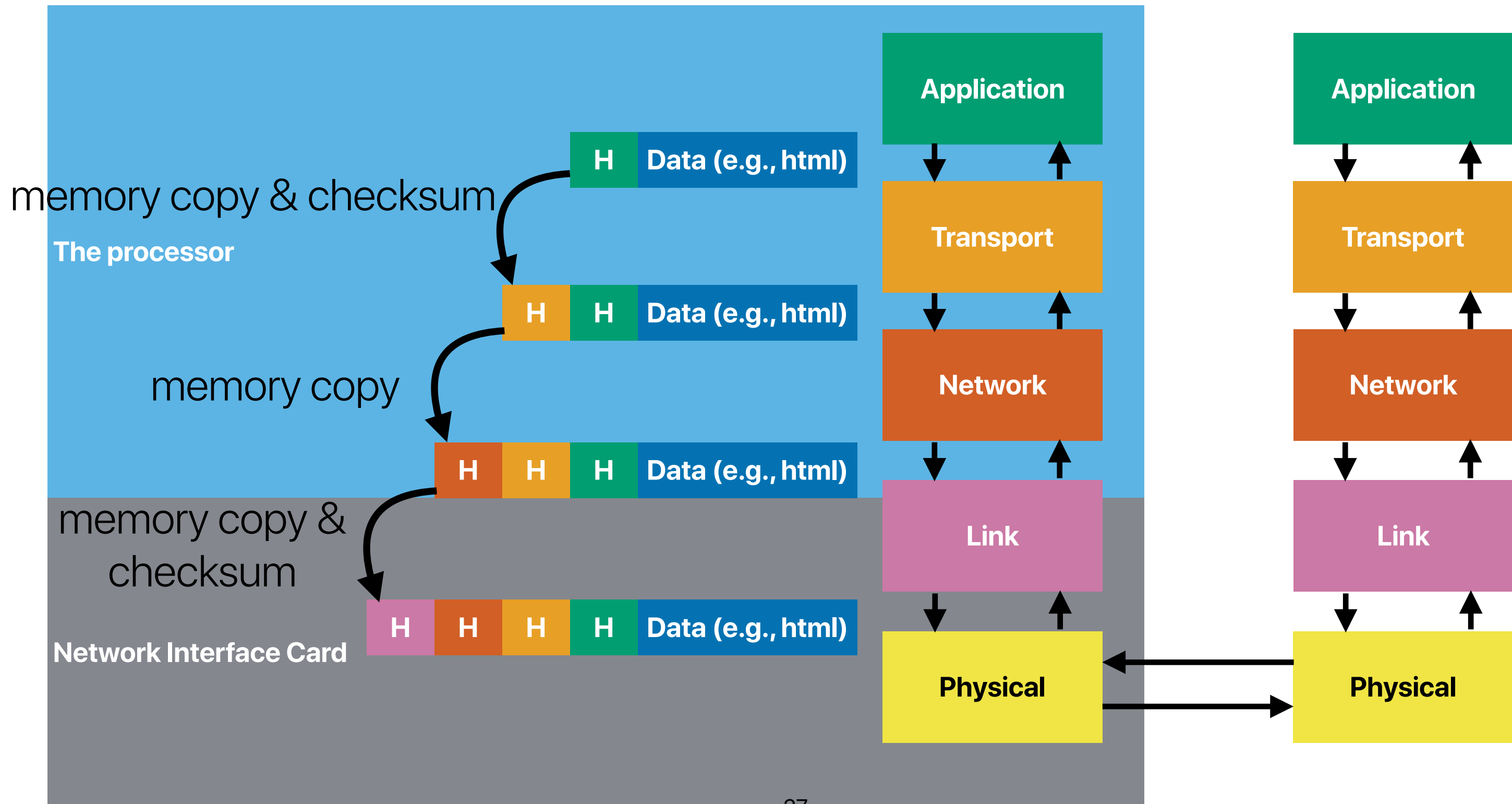
How do a computer use the media:  
IEEE 802.3 (ethernet), IEEE 802.11 (WiFi)



How do the computer encode data on the media



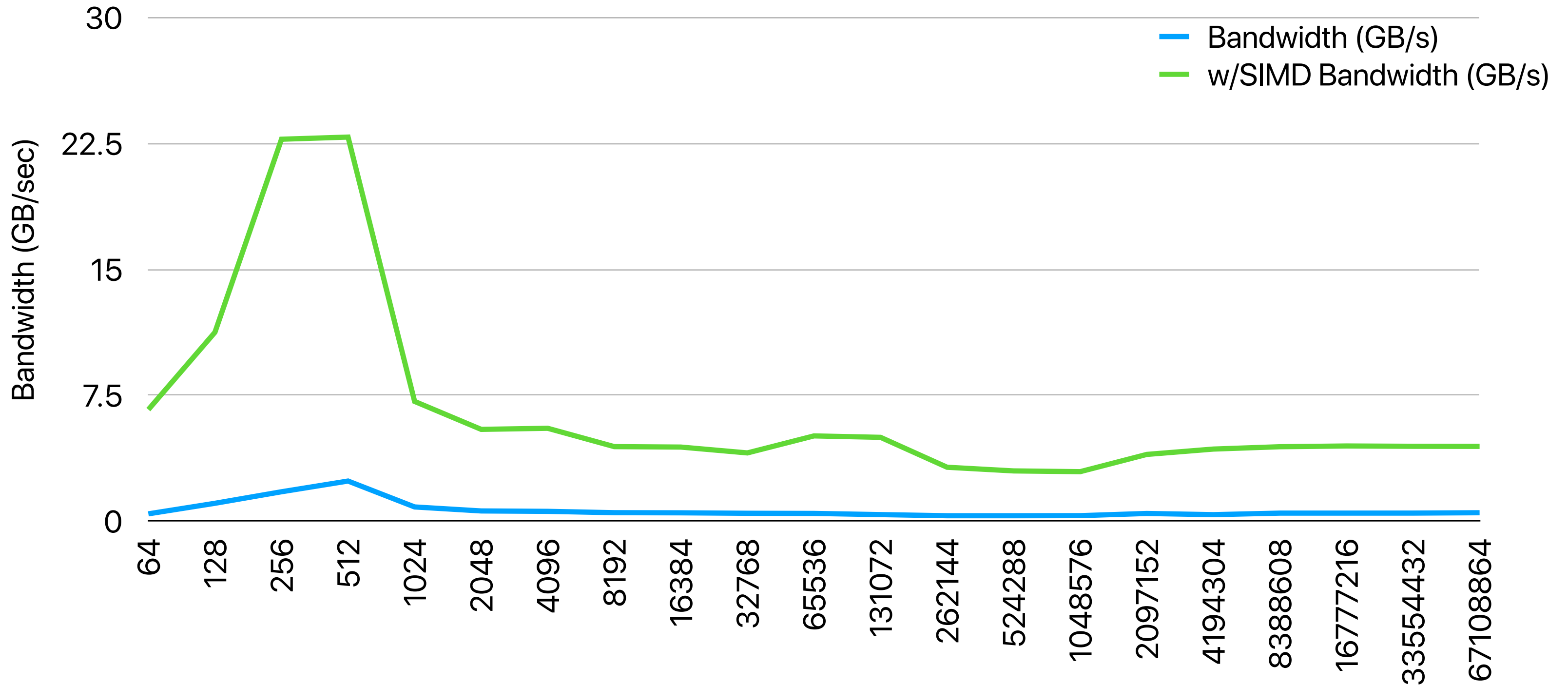
# Network protocol stack



# 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





# Electrical Computer Science Engineering

# 277

# つくづく

