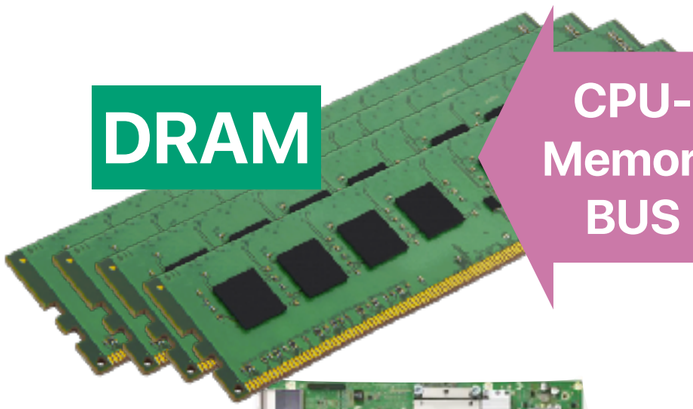
Memory subsystem (3)

Hung-Wei Tseng

Recap: The landscape of modern computers



CPU-Memory BUS

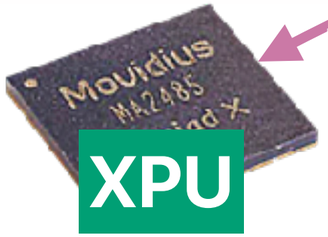
DDR4

Memory Controller

High-performance, but very specialized

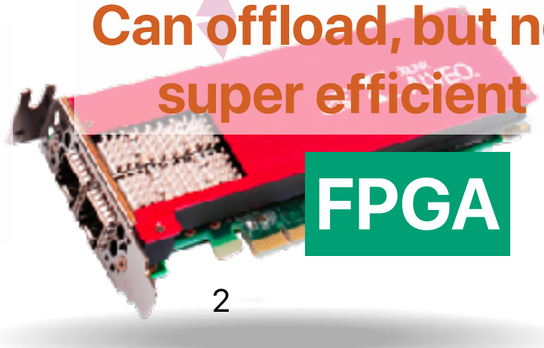
PCI EXPRESS

PCIe Root Complex

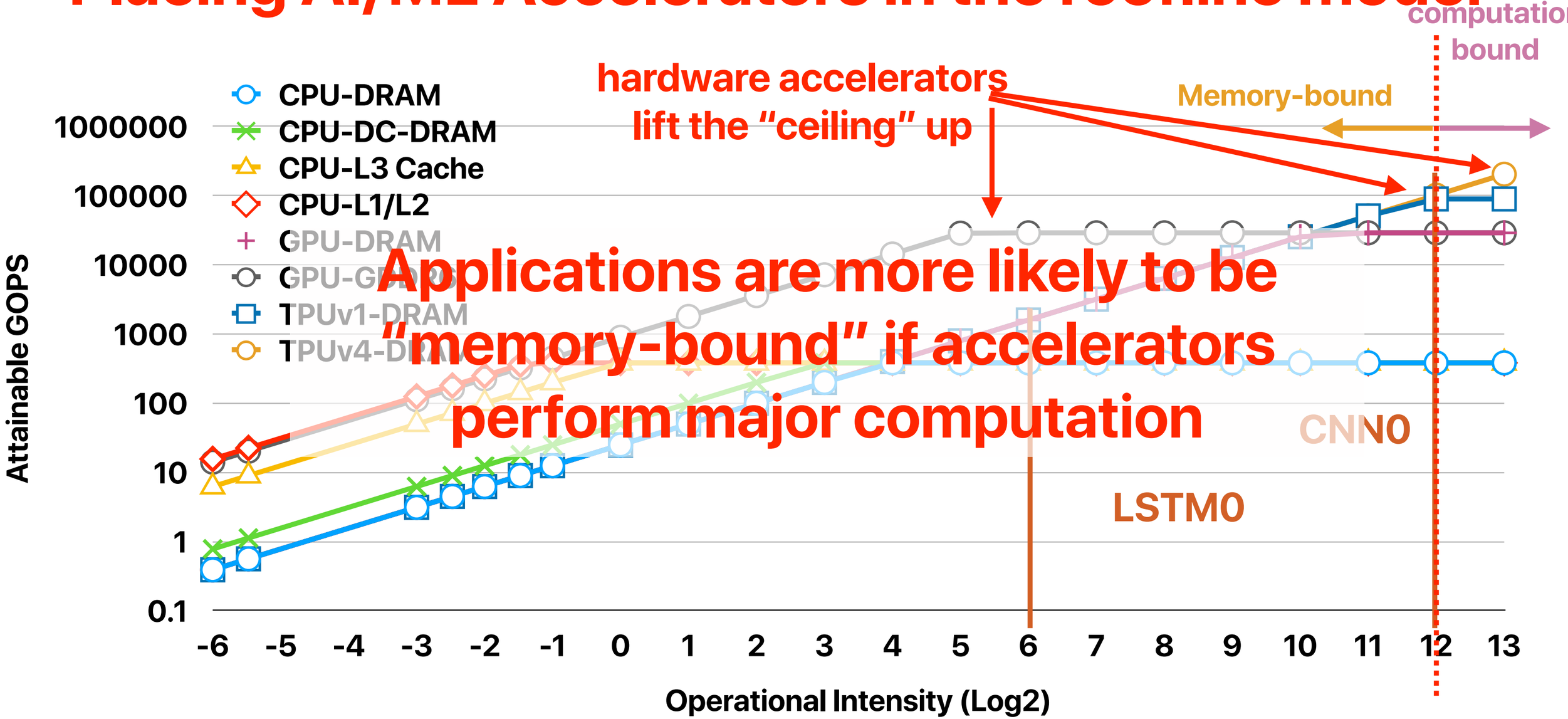


General-purpose for "regular" workloads, but can be power consuming

Can offload, but not super efficient



Placing AI/ML Accelerators in the roofline model

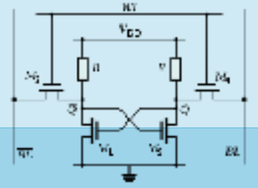


Recap: Memory technologies we have today

Volatile Memory

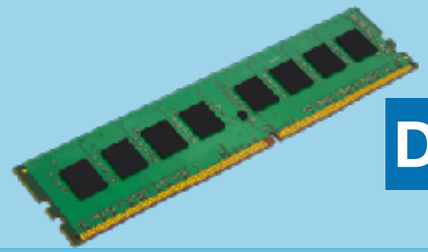
Non-Volatile Memory

100ps

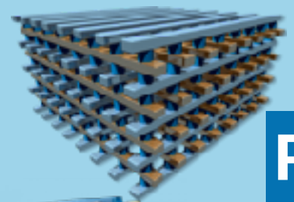


SRAM

ns



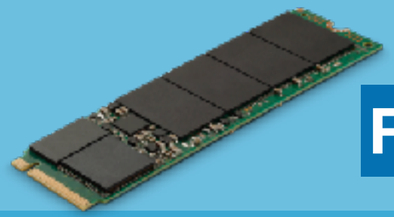
DRAM



RRAM

PCM 3DXPoint

us



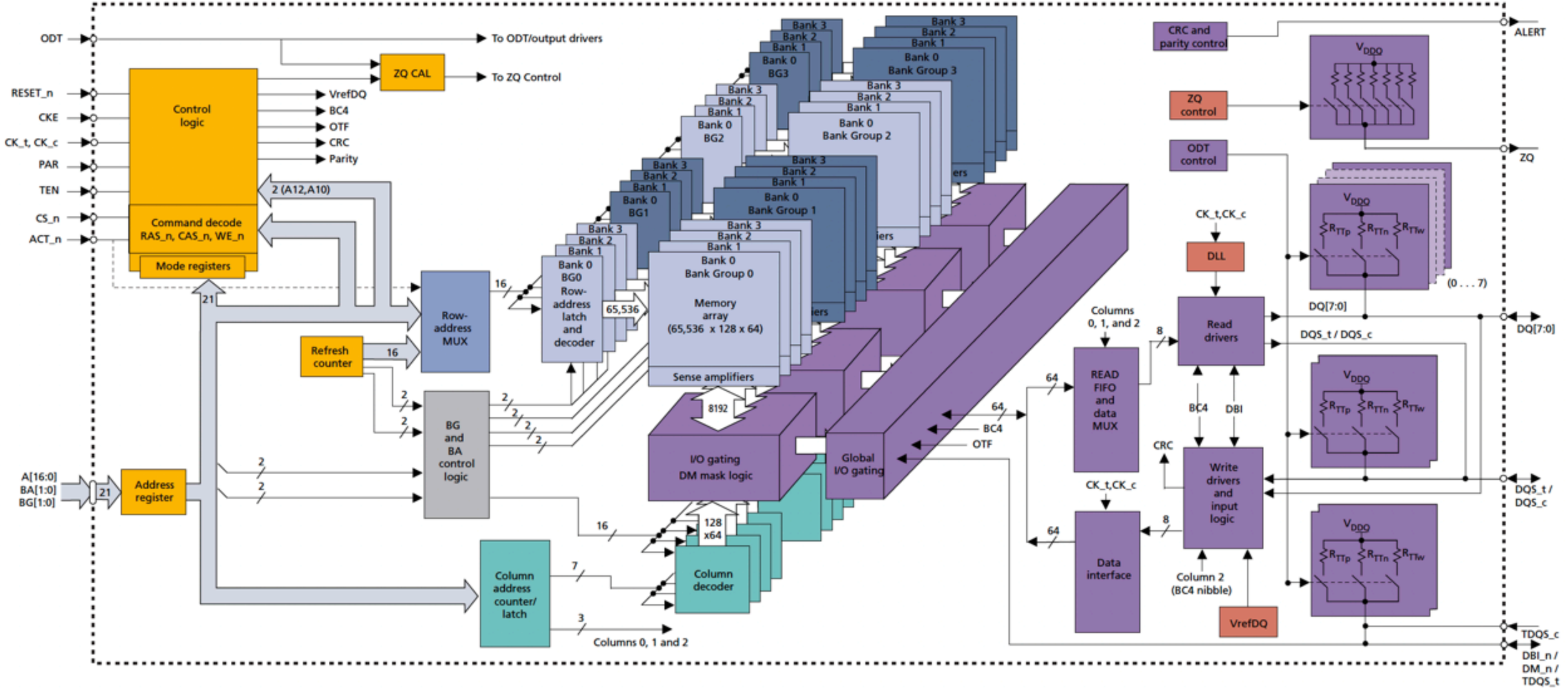
Flash memory

ms

Hard Disk Drives

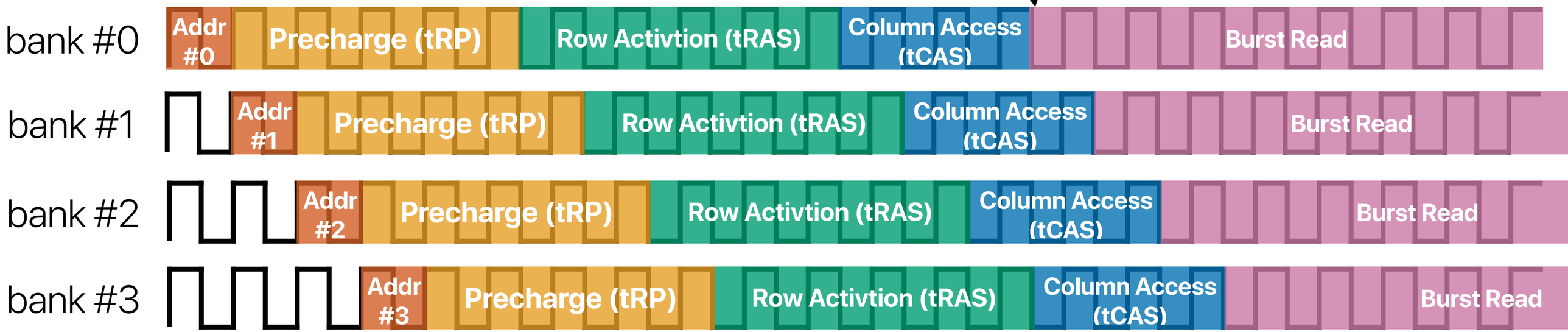
Recap: Improving memory-bounded applications

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


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
Read drivers
BC4
DBI
CRC
Write drivers and input logic
Column 2 (BC4 nibble)
VrefDQ
CRC and parity control
ZQ control
ODT control
CK_t,CK_c
DLL
DQ[7:0]
DQS_t / DQS_c
(0 . . . 7)
ALERT
ZQ
DQ[7:0]
DQS_t / DQS_c
TDQS_c
DBI_n / DM_n / TDQS_t
V_{DDQ}
R_{TTp}
R_{TTn}
R_{TTw}

Recap: Multi-bank access

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

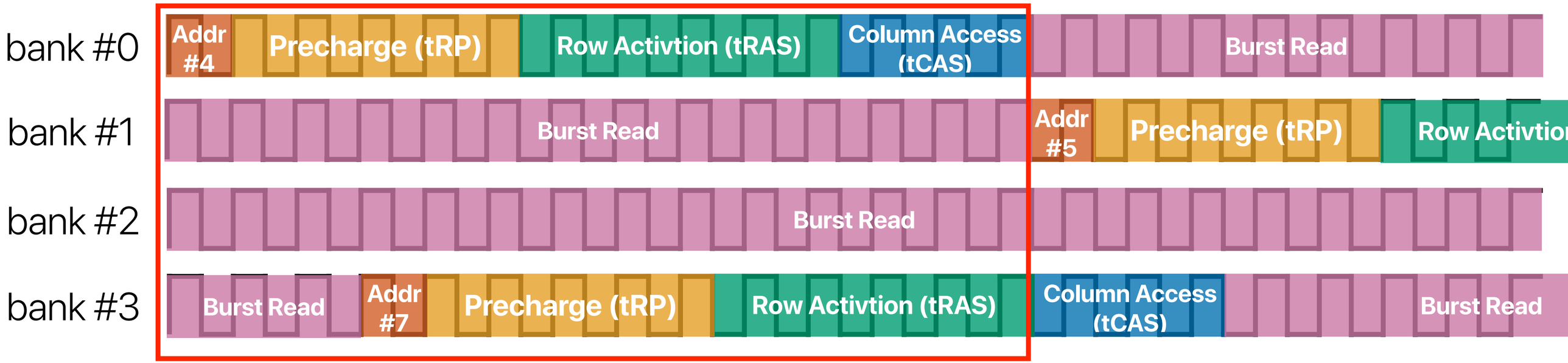


only one bank can accept request each cycle

the memory bandwidth can be fully utilized after this

Multi-bank access

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



Outline

- Improve the performance of main memory subsystem
- Near/In-memory processing

Memory bandwidth

3:00

How can you increase the DRAM bandwidth?

Ideas of increasing bandwidth

Ideas of increasing bandwidth

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

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.

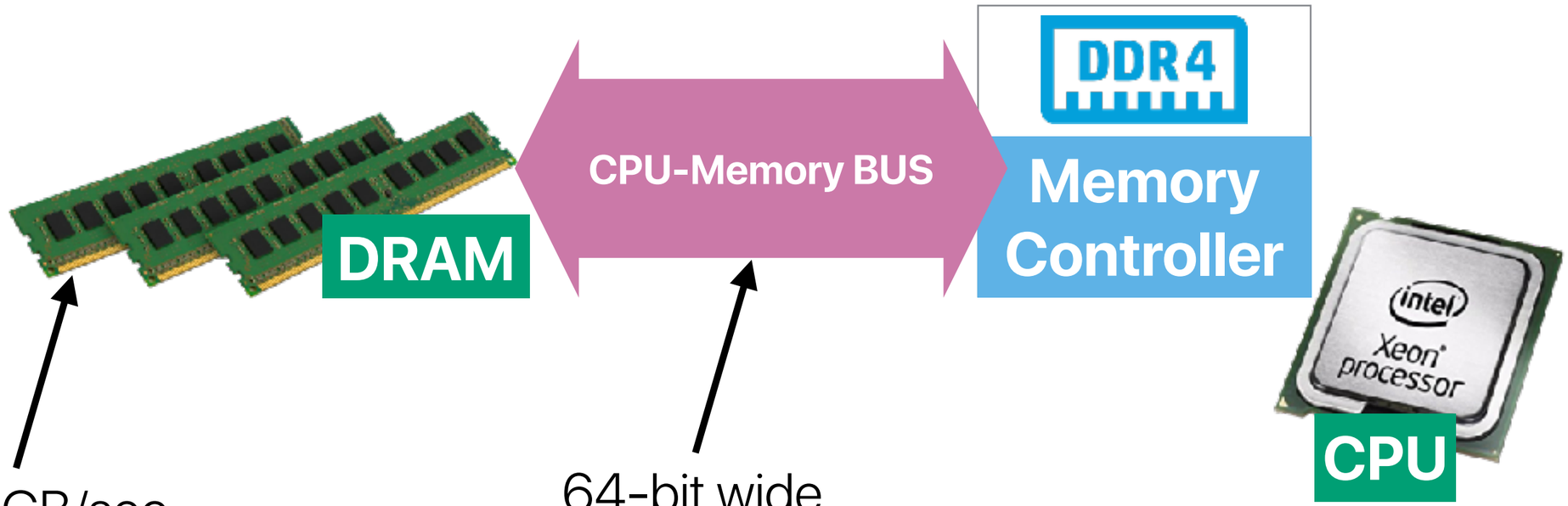
The chip is designed to comply with the following key DDR4 SDRAM features such as posted CAS, Programmable CWL, Internal (Self) Calibration, On Die Termination using ODT pin and Asynchronous Reset.

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

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

DDR5-5600

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

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

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

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

3:00

```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 */

    write_memory_loop(array, size);

    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start
time */
    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;
}
```

https://github.com/hungweitseng/EE277/tree/main/demo/memory

Electrical
Computer Science
Engineering

277

つづく