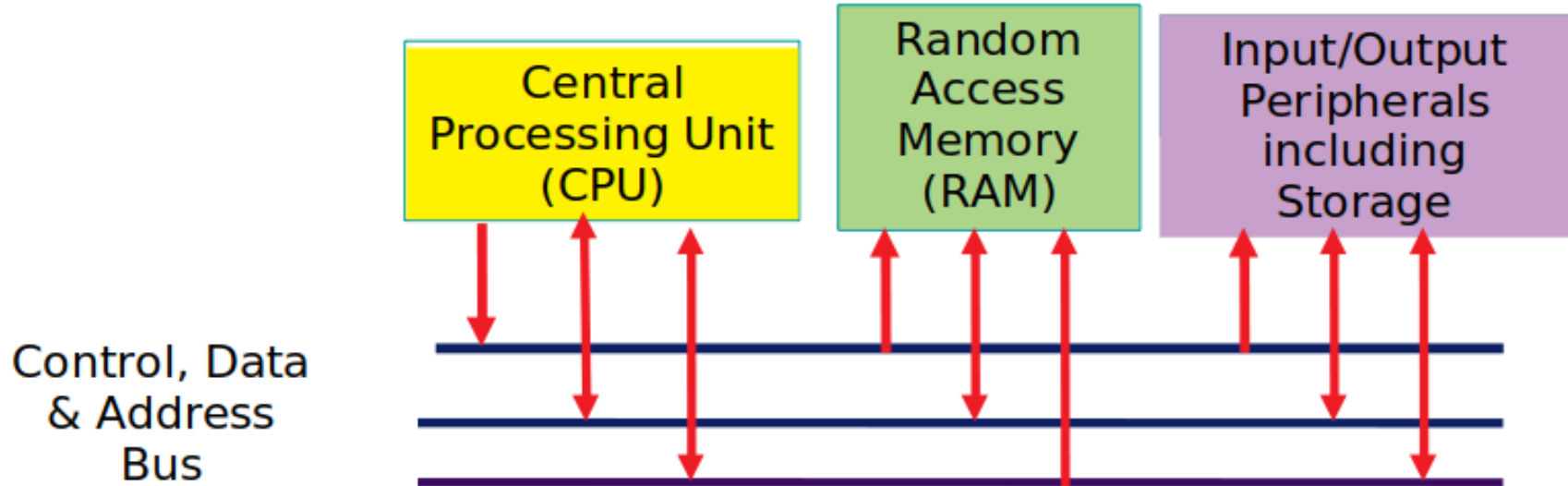


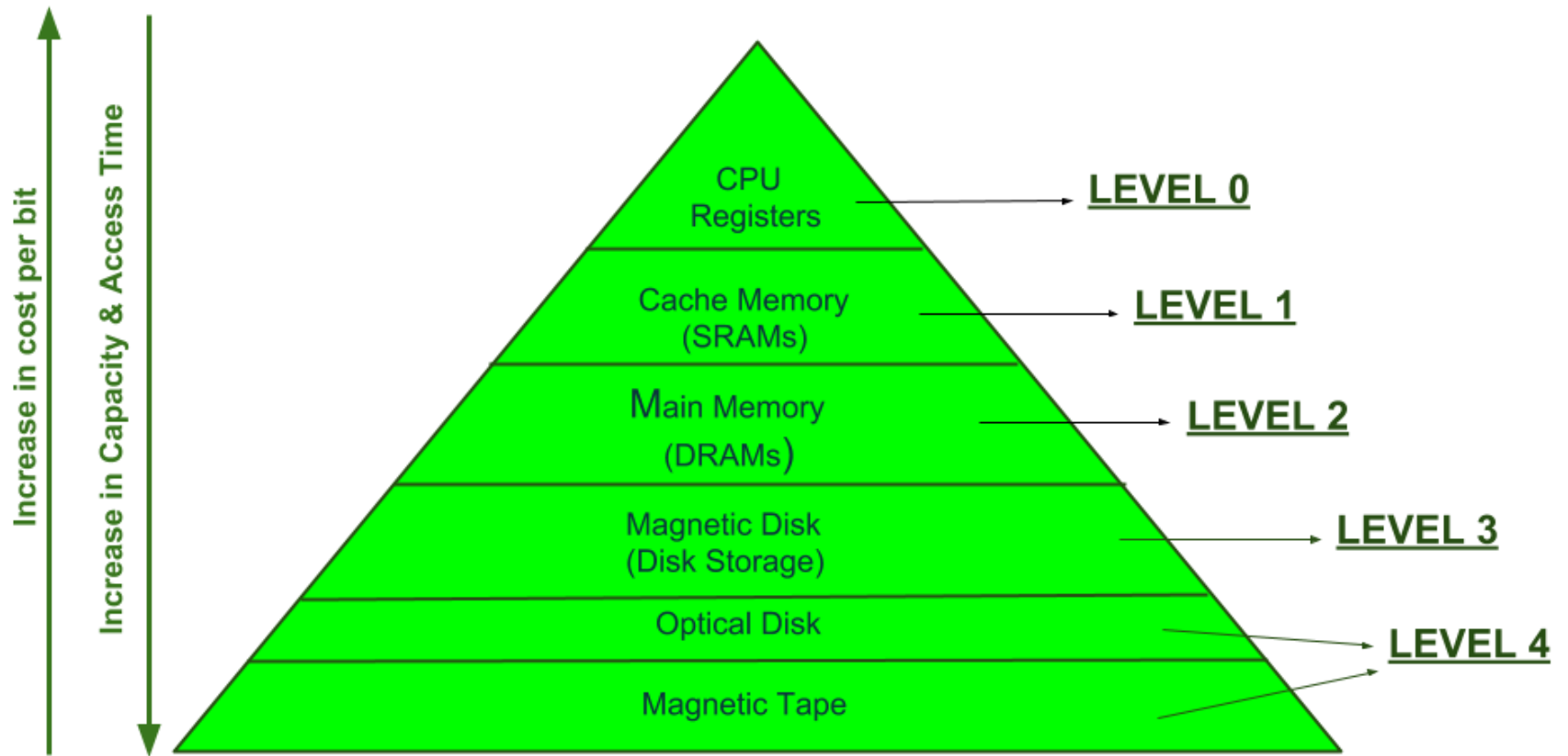
Lecture 7. Computer memory organization. Special registers.

Dmitry Zaitsev

<http://daze.ho.ua>

Basic X86 architecture





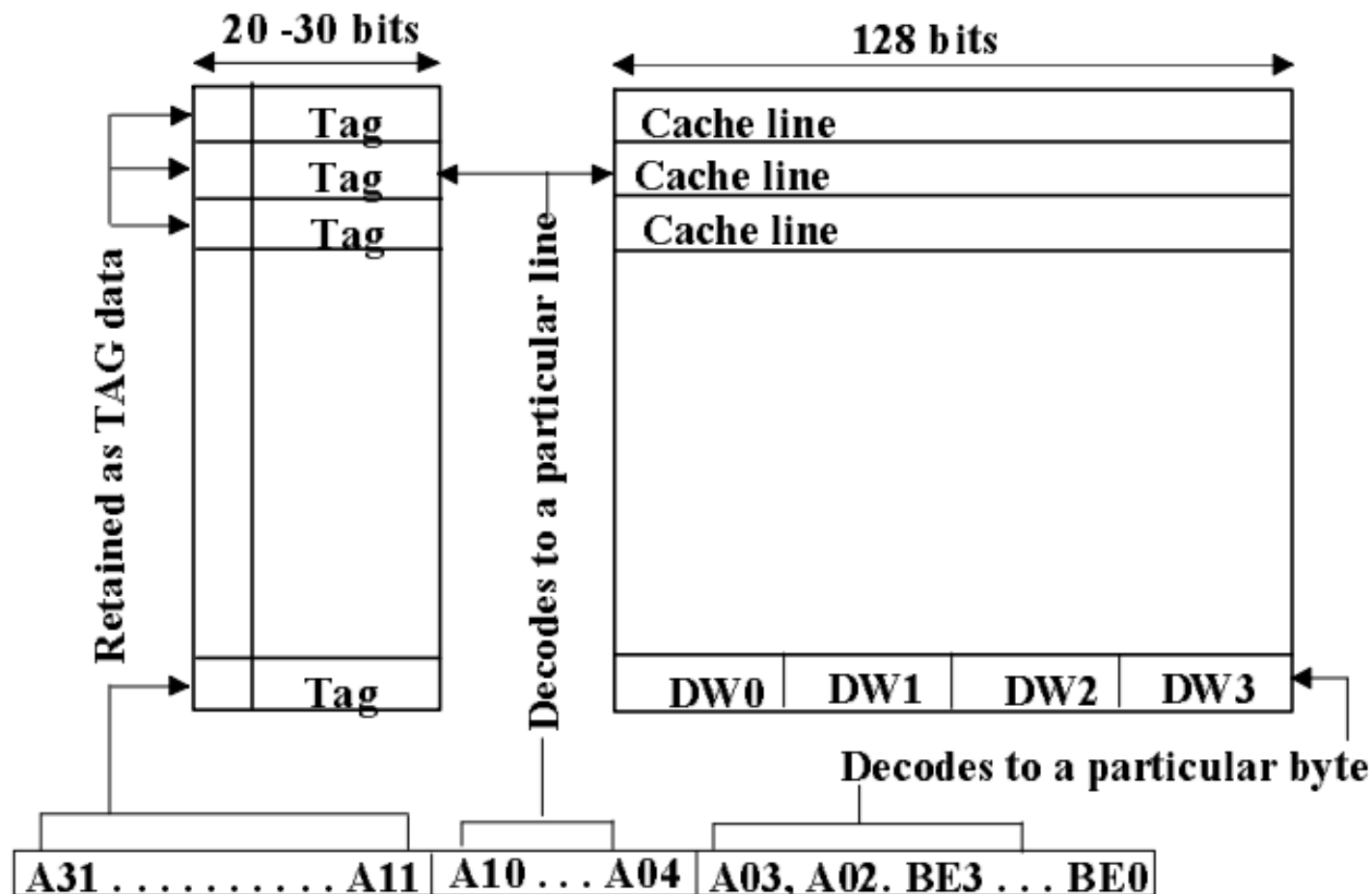
MEMORY HIERARCHY DESIGN

Caching Data

- a process that stores multiple copies of data or files in a temporary storage location—or cache—so they can be accessed faster
- caching is a good solution for the von Neumann bottleneck, which looks at ways to better serve faster memory access

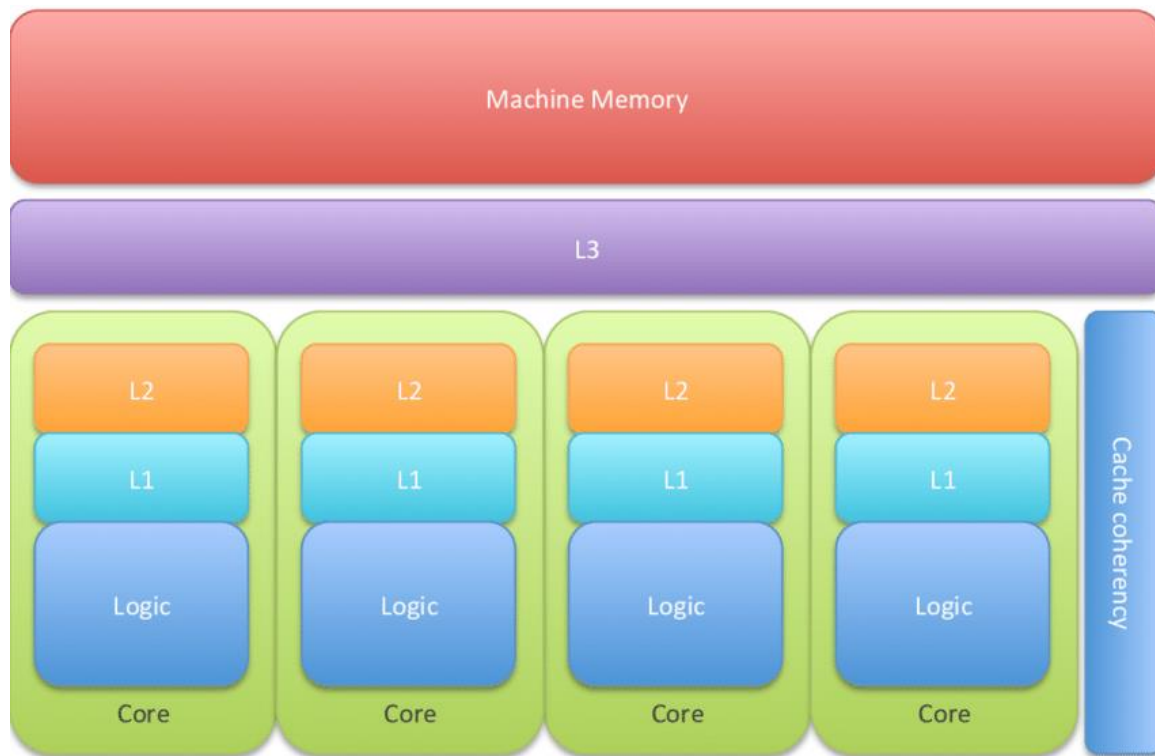
How caching works

- data is stored with a tag indicating its mapping to the memory
- data is stored until its time to live (TTL), which indicates how long content needs to be cached
- first look in cache than in memory



Cache work scheme

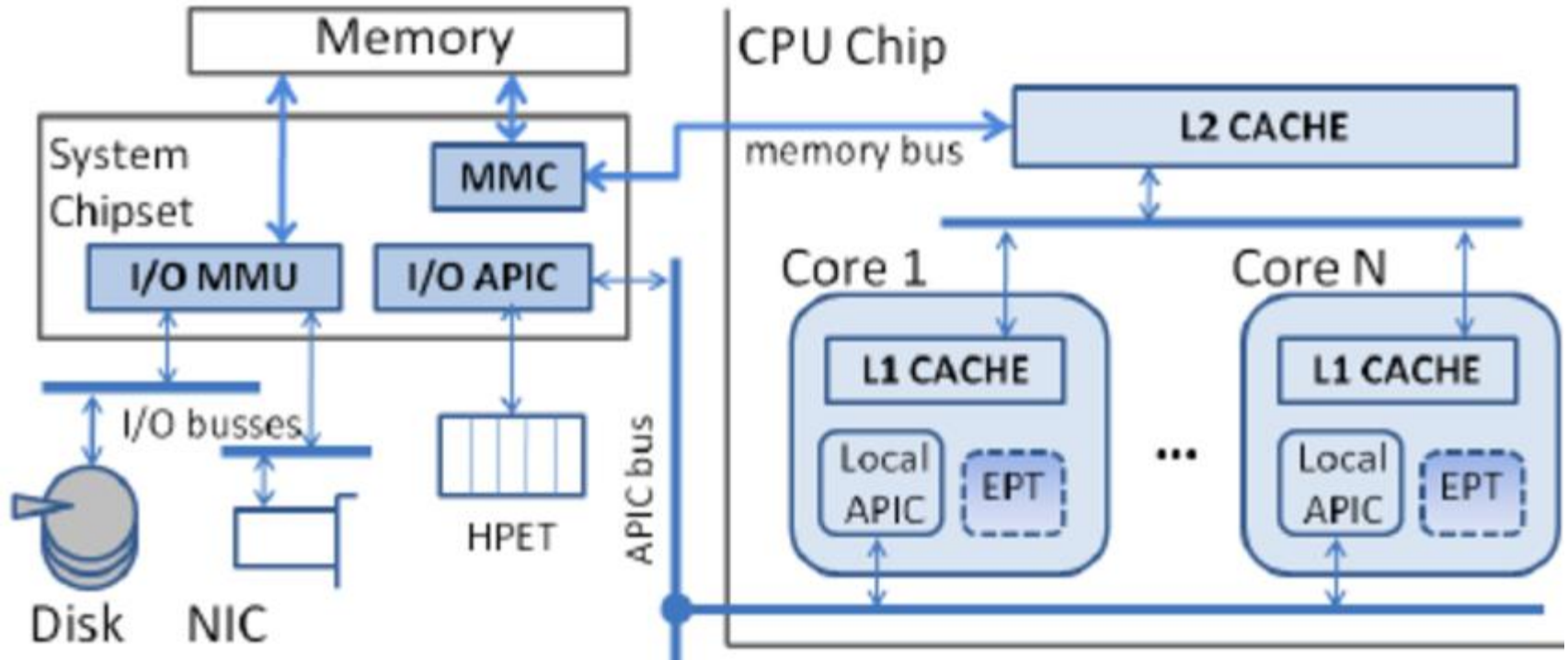
Multicore architecture and cache



Layers of cache

- L1 cache, or primary cache, is extremely fast but relatively small, and is usually embedded in the processor chip as CPU cache.
- L2 cache, or secondary cache, is often more capacious than L1.
- Level 3 (L3) cache is specialized memory developed to improve the performance of L1 and L2 which are faster than L3 (about double speed of DRAM)

Interaction of caches



Benchmarks

- Specially organized tests of performance for hardware and software
- Measure time of execution on a model task
- Examples of wide-known benchmarks for computers: LINPACK, HPL, HPCG
- Consequent fine tuning with profilers

Task

- implement comparative benchmark for personal implementation of math function in assembler and the corresponding libc function
- generate big array of source data
- measure time of running on the array for:
 - your implementation of math function
 - libc implementation of math function

Benchmark example for OpenMP

- OpenMP is recently a standard part of compilers (gcc) to employ multicore CPU
- test task – sum up two vectors
- what to compare: using 1, 2, ... threads (cores)
- remark: pick performance of modern processors is achieved with matrix multiplication

Sum of vectors, example $z=x+y$

Vector\Index	0	1	2	3	4	5
x	-2	1	-7	0	-1	4
y	5	1	-2	1	1	-6
z	3	2	-9	1	0	-2

We can sum up all elements in parallel having the number of threads equal to the length of array!

Parallel loop – the best case – independent passages

```
omp_set_num_threads(thn);  
#pragma omp parallel for  
for(i=0;i<n;i++)  
{  
    c[i] = a[i] + b[i];  
}
```

An example for $n=6$, $th=3$

- Thread 1:

$c[0]=a[0]+b[0]$

$c[3]=a[3]+b[3]$

- Thread 2:

$c[1]=a[1]+b[0]$

$c[4]=a[4]+b[4]$

- Thread 3:

$c[2]=a[2]+b[2]$

$c[5]=a[5]+b[5]$

Benchmark implementation

```
#include <time.h>
#include <bits/time.h>
#include <sys/time.h>
double magma_wtime( void )
{
    struct timeval t;
    gettimeofday( &t, NULL );
    return t.tv_sec + t.tv_usec*1e-6;
}
```

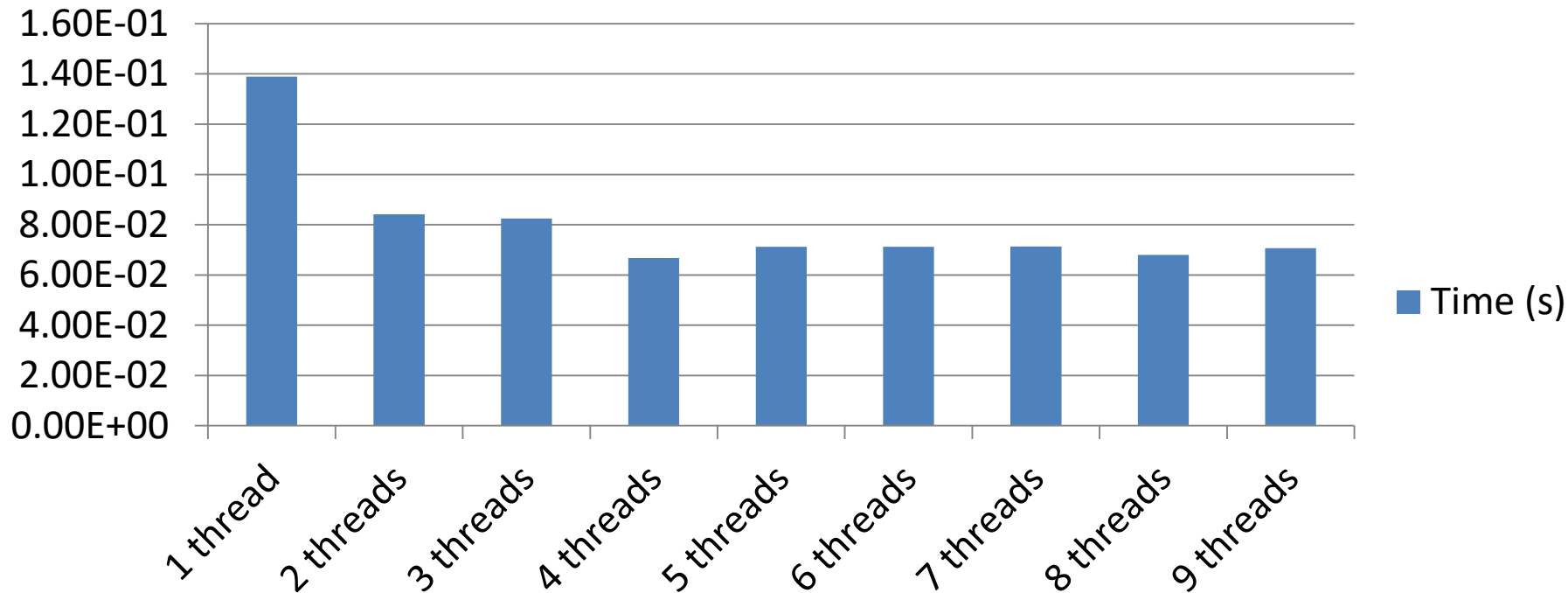
```
int main()
{
    double t1,t2;
    t1=magma_wtime();
    ...
    t2=magma_wtime();
    printf( "time=%#fs\n", t2-t1 );
}
```


Actual benchmarks

lengths=67108864, threads=1, time = 1.389780e-01 s.
lengths=67108864, threads=2, time = 8.413601e-02 s.
lengths=67108864, threads=3, time = 8.251595e-02 s.
lengths=67108864, threads=4, time = 6.673717e-02 s.
lengths=67108864, threads=5, time = 7.124591e-02 s.
lengths=67108864, threads=6, time = 7.118201e-02 s.
lengths=67108864, threads=7, time = 7.130098e-02 s.
lengths=67108864, threads=8, time = 6.796098e-02 s.
lengths=67108864, threads=9, time = 7.065511e-02 s.

Benchmark diagram

Time (s)



Special registers

- Segment registers
- EFLAGS Register
- EIP instruction pointer
- Control registers

Segment Registers

- Stack Segment (SS). Pointer to the stack ('S' stands for 'Stack').
- Code Segment (CS). Pointer to the code ('C' stands for 'Code').
- Data Segment (DS). Pointer to the data ('D' stands for 'Data').
- Extra Segment (ES). Pointer to extra data ('E' stands for 'Extra').
- F Segment (FS). Pointer to more extra data ('F' comes after 'E').
- G Segment (GS). Pointer to still more extra data ('G' comes after 'F').

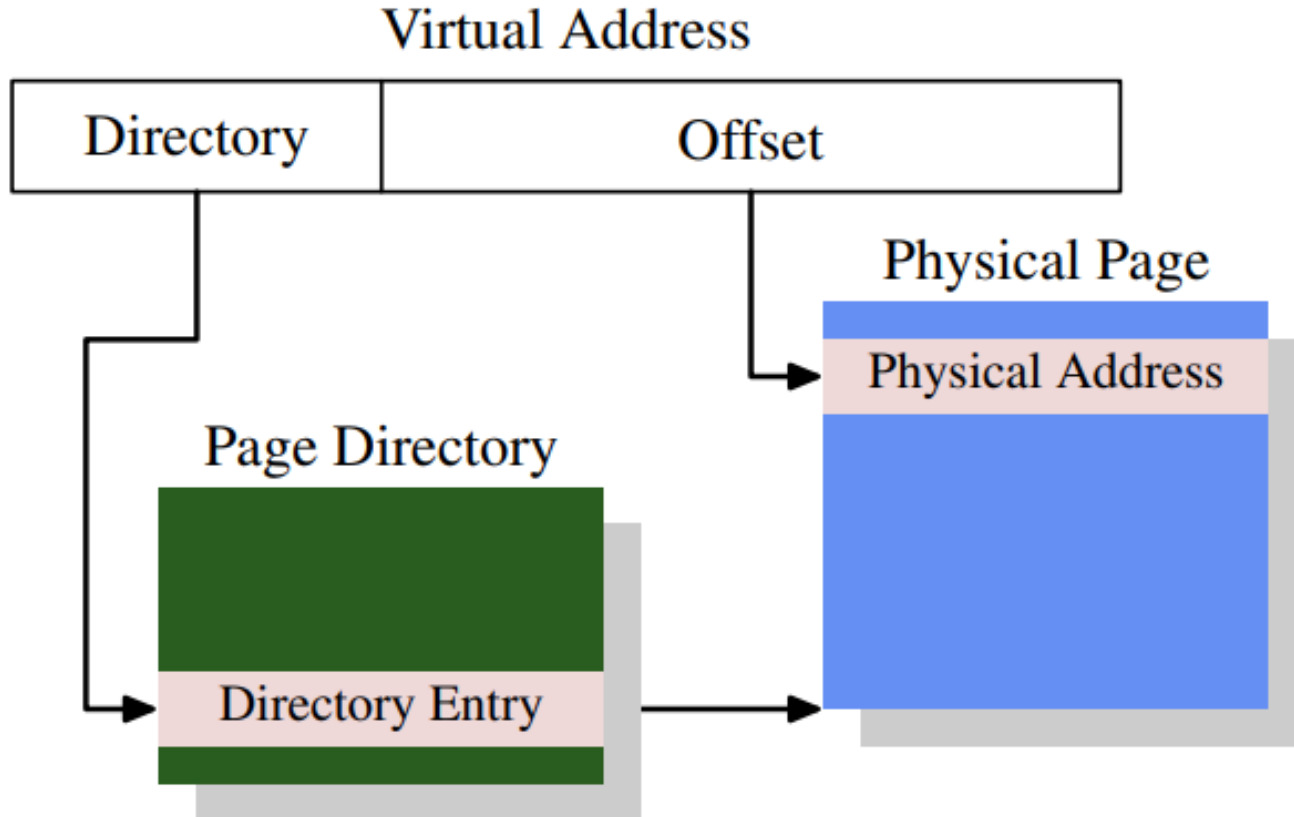
Control registers

- CR0 has various control flags that modify the basic operation of the processor.
- CR2 Page Fault Linear Address
- CR3 enables the processor to translate linear addresses into physical addresses.
- CR4 enables I/O breakpoints
- Extended Feature Enable Register
- CR8 is used to prioritize external interrupts and is referred to as the task-priority register (TPR)
- XCR0 Extended Control Register – enable or disable the processor's ability to execute specific instructions

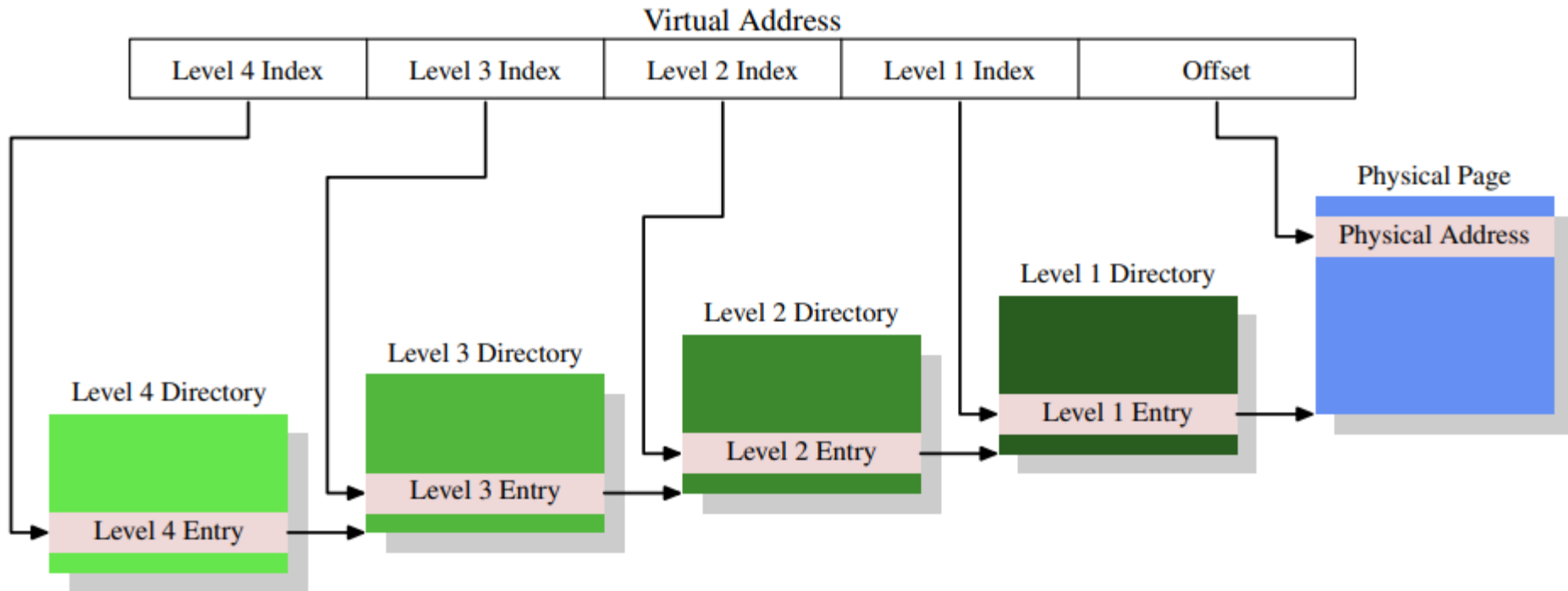
Virtual memory

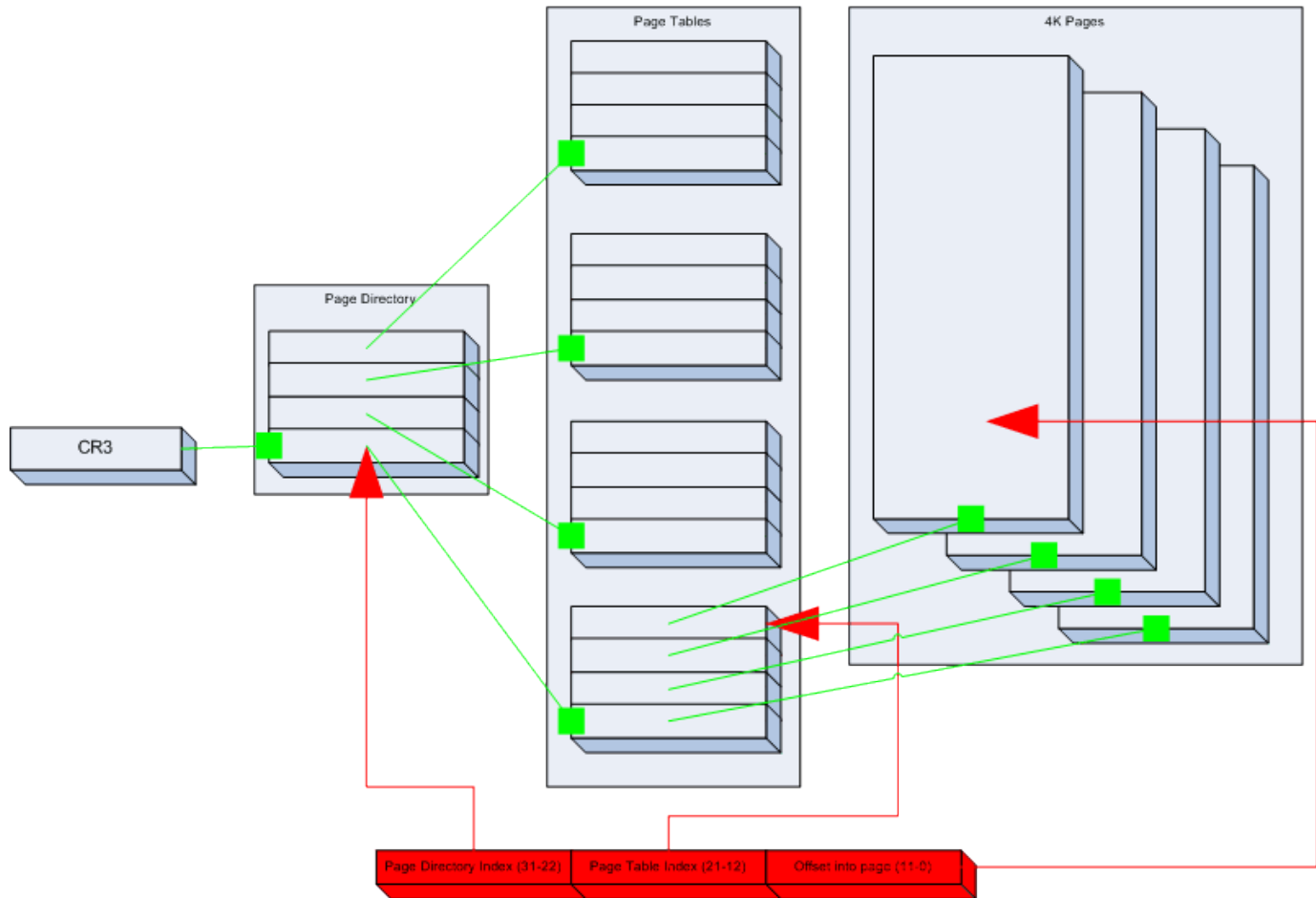
- The virtual memory (VM) subsystem of a processor implements the virtual address spaces provided to each process.
- A virtual address space is implemented by the Memory Management Unit (MMU) of the CPU.
- OS fills out the page table data structures

One-level translation of addresses



Multi-level address translation





Paging structure

Basic data structures

- CR3 register points to a page directory
- pages 4Mb-4Kb
- 1024 page directory entries (PDEs) that point to page tables
- page table contains 1024 page table entries (PTEs)

Formats of tables

Page Directory Entry (4 MB)

31	...	22	21	20	...	13	12	11	...	9	8	7	6	5	4	3	2	1	0
Bits 31-22 of address				R S V D (0)	Bits 39-32 of address				P A T	AVL	G	P S (1)	D	A	P C D	P W T	U / S	R / W	P

Page Directory Entry

31	...	12	11	...	8	7	6	5	4	3	2	1	0	
Bits 31-12 of address						AVL	P S (0)	A V L	A	P C D	P W T	U / S	R / W	P

P: Present	D: Dirty
R/W: Read/Write	PS: Page Size
U/S: User/Supervisor	G: Global
PWT: Write-Through	AVL: Available
PCD: Cache Disable	PAT: Page Attribute Table
A: Accessed	

Page Table Entry

31	...	12	11... 9	8	7	6	5	4	3	2	1	0
Bits 31-12 of address			AVL	G	P A T	D	A	P C D	P W T	U / S	R / W	P

P: Present	D: Dirty
R/W: Read/Write	G: Global
U/S: User/Supervisor	AVL: Available
PWT: Write-Through	PAT: Page Attribute Table
PCD: Cache Disable	
A: Accessed	

Memory mapping example

Physical Memory	
00x	H E L L
01x	R L D !
02x	O W O
03x	H A V E
04x	F U N
05x	L O T
06x	S O F
07x	; -)

Process A			
Page Table		Virtual Memory	
00x	00	00x	H E L L
01x	02	01x	O W O
02x	01	02x	R L D !
03x	n.a.	03x	#####
04x	n.a.	04x	#####
05x	07	05x	; -)

Process B			
Page Table		Virtual Memory	
00x	03	00x	H A V E
01x	05	01x	L O T
02x	06	02x	S O F
03x	04	03x	F U N
04x	n.a.	04x	#####
05x	07	05x	; -)

Page swapping

- Interrupt if page is not present in physical memory
- OS handles interrupt and starts swapping
- If there is free page, required page is loaded
- Otherwise swap out some page
- Last Recently Used (LRU); bit of access and bit of changes; periodical reset

Page swapping scheme

