АКОС №6

Виртуальная память

Загадки человечества

на которые мы ответим

- Почему у всех программ одни и те же адреса? Как они не мешают друг другу?
- Почему если выйти за границы массива то иногда происходит SEGFAULT, а иногда ничего не происходит (UB)?
- Как компьютер вообще понимает, какая память ”наша”, а какая - нет?
- Как работает malloc/new/free/delete? Откуда они берут память?
- Почему можно исполнять код программы, но не получится исполнять данные? Например, записать в данные машинный код и исполнить его.
- Почему не выйдет в ассемблере написать самомодифицирующийся код?

Организация памяти процесса

Как посмотреть?

$ vim /proc/*pid*/maps

z.B. $ vim /proc/self/maps

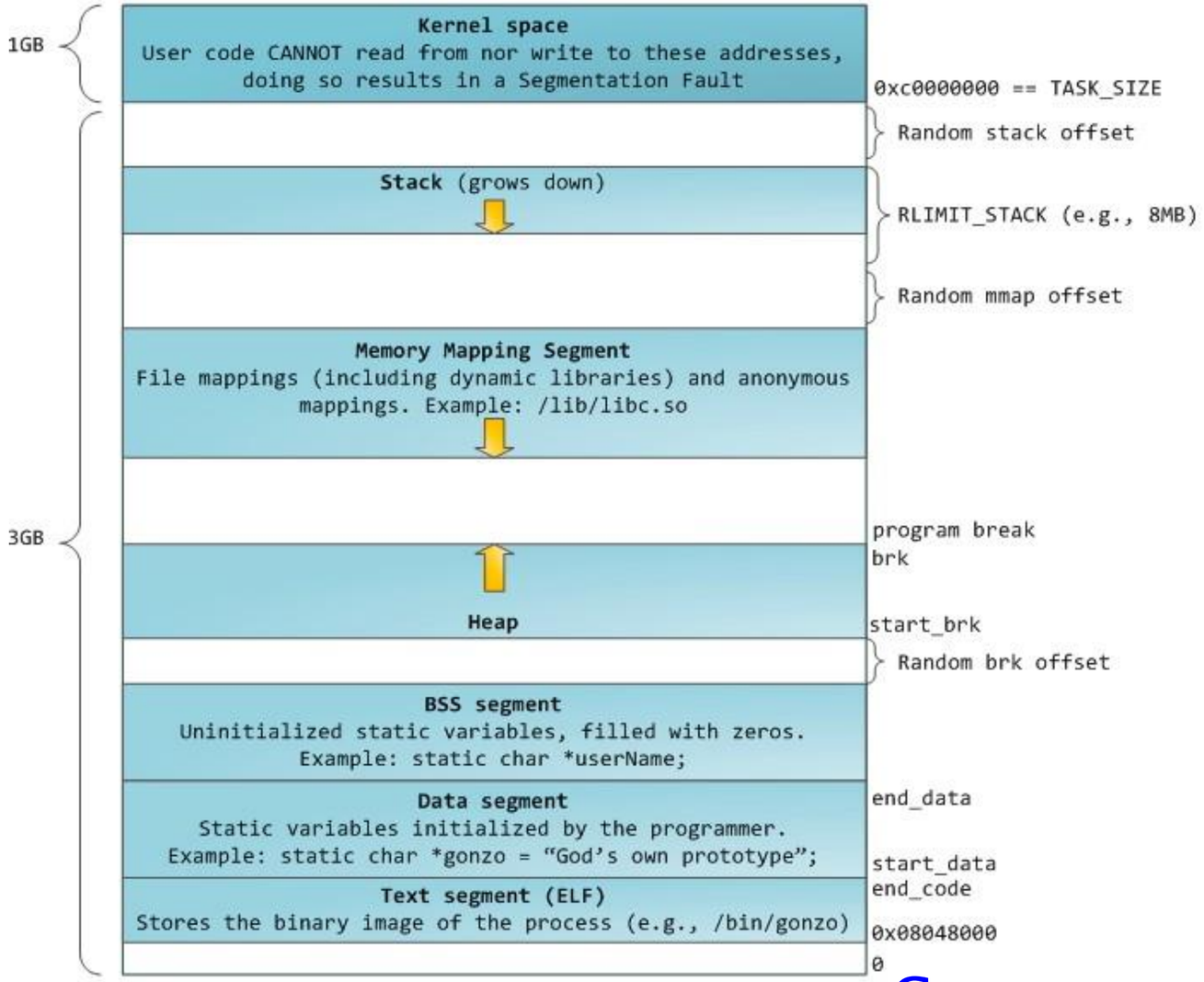
(self = текущий процесс)

(Не только maps, но и status, mem, map_files и т.д.)

$ objdump -t a.out

Вопрос.

Как процессы не конфликтуют друг с другом за адреса?



Статья на хабре

Исторический экскурс

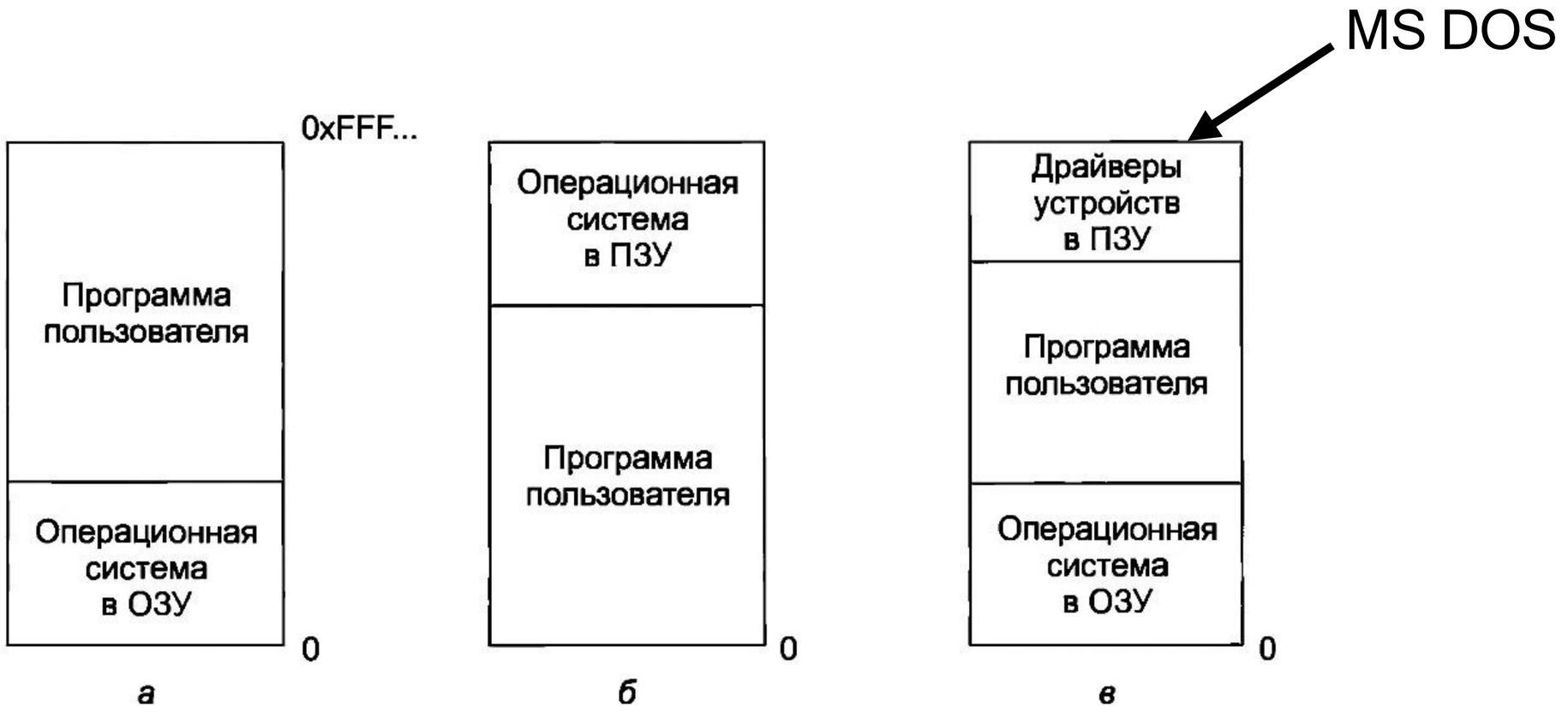


Рис. 3.1. Три простых способа организации памяти при наличии операционной системы и одного пользовательского процесса (существуют также и другие варианты)

Проблемы такого подхода

- Безопасность: можно портить чужую (в том числе ОС) память
- Разграничение: процессы находятся рядом друг с другом и могут мешать
- Абстракция: каждый процесс хочет жить в мире с доступной памятью от 0x0 до 0xFFF... F, а не на выделенном клочке

Сегментная модель

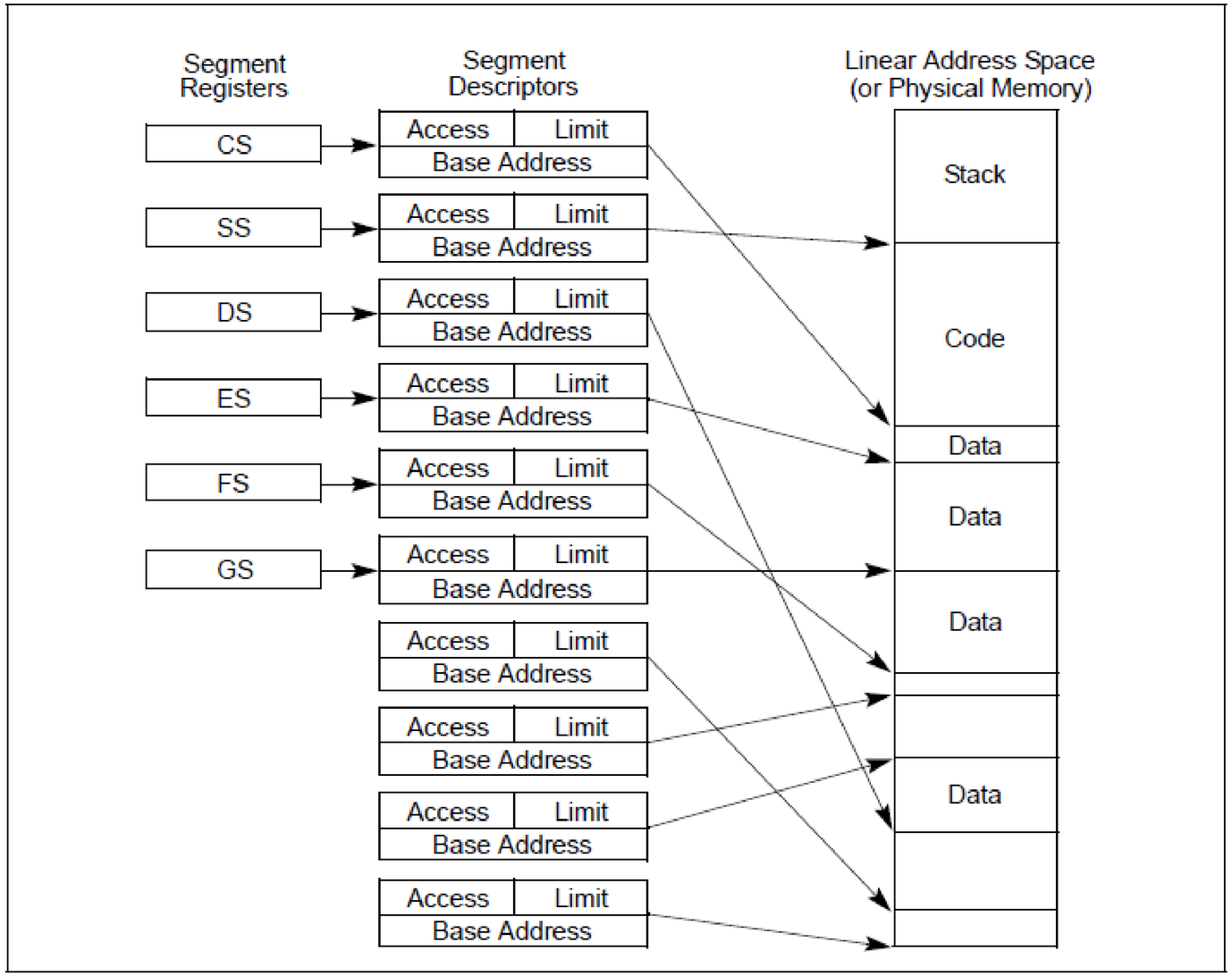


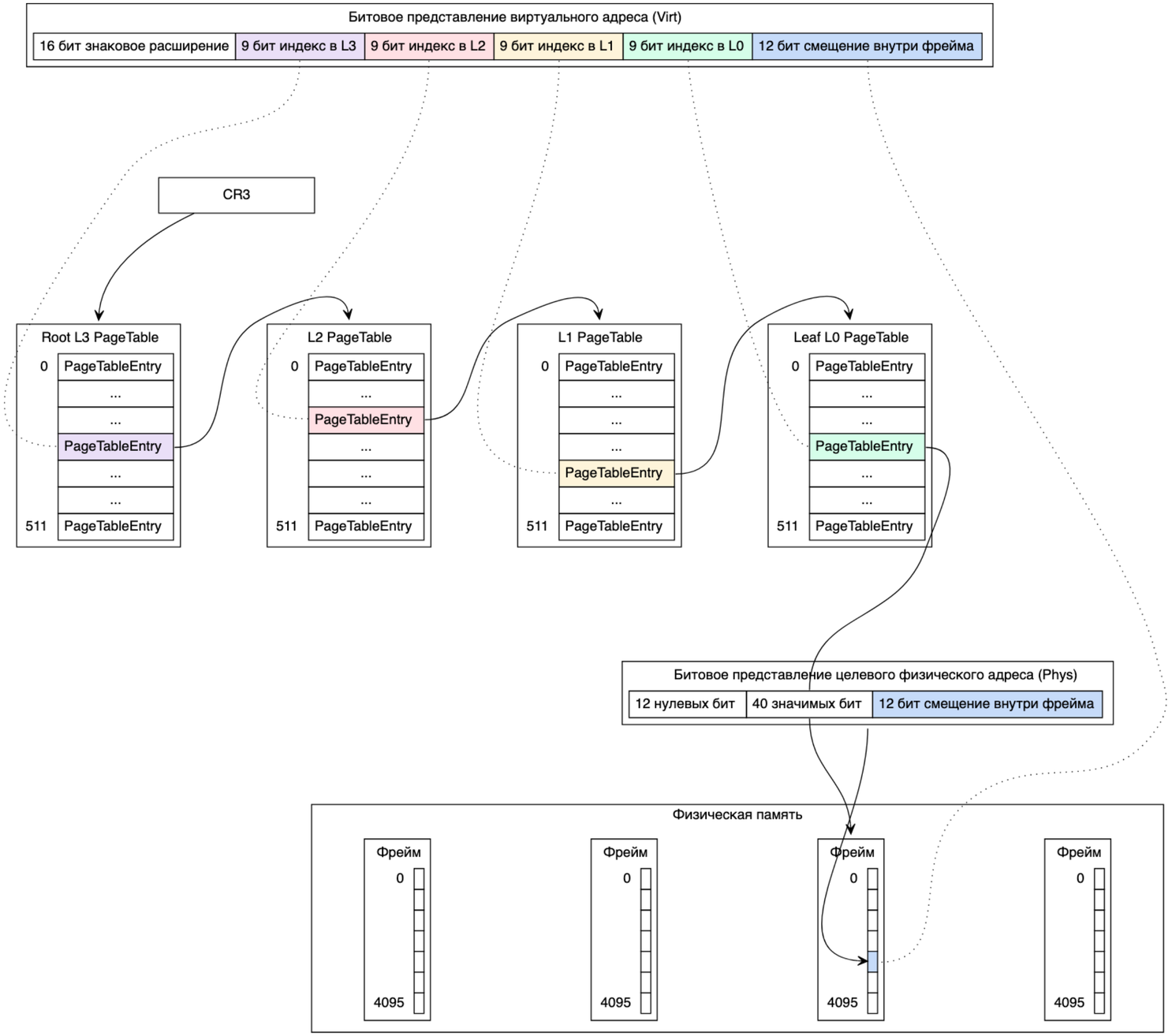
Figure 3-4. Multi-Segment Model

Виртуальная память

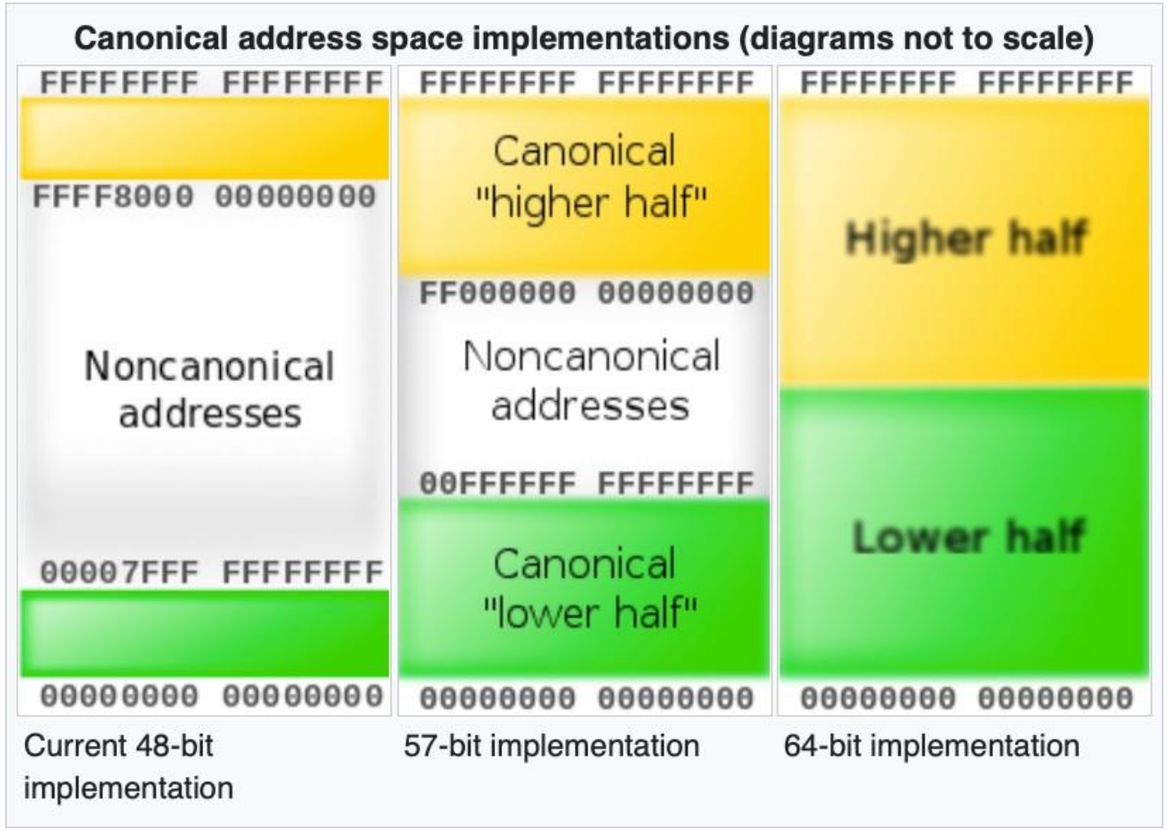
Хотим добиться следующей абстракции:

1) Доступны все адреса от 0 до 2^64 - как непрерывный блок
2) Память процессов может быть больше, чем вообще доступно памяти
3) Память процессов независима друг от друга

Page Walk


Битовое представление виртуального адреса (Virt)
16 бит знаковое расширение
9 бит индекс в L3
9 бит индекс в L2
9 бит индекс в L1
9 бит индекс в L0
12 бит смещение внутри фрейма
CR3
Root L3 PageTable
L2 PageTable
L1 PageTable
Leaf L0 PageTable
0
511
PageTableEntry
...
Битовое представление целевого физического адреса (Phys)
12 нулевых бит
40 значимых бит
12 бит смещение внутри фрейма
Физическая память
Фрейм
0
4095

Две половины адресов



Start addr	Offset	End addr	Size	VM area description
0000000000000000	0	00007fffffffffff	128 TB	user-space virtual memory, different per mm
0000800000000000	+128 TB	ffff7fffffffffff	~16M TB	... huge, almost 64 bits wide hole of non-canonical virtual memory addresses up to the -128 TB starting offset of kernel mappings.
				Kernel-space virtual memory, shared between all processes:
ffff800000000000	-128 TB	ffff87ffffffffff	8 TB	... guard hole, also reserved for hypervisor
ffff880000000000	-120 TB	ffff887fffffffff	0.5 TB	LDT remap for PTI
ffff888000000000	-119.5 TB	ffffc87fffffffff	64 TB	direct mapping of all physical memory (page_offset_base)
ffffc88000000000	-55.5 TB	ffffc8ffffffffff	0.5 TB	... unused hole
ffffc90000000000	-55 TB	ffffe8ffffffffff	32 TB	vmalloc/ioremap space (vmalloc_base)
ffffe90000000000	-23 TB	ffffe9ffffffffff	1 TB	... unused hole
ffffea0000000000	-22 TB	ffffeaffffffffff	1 TB	virtual memory map (vmemmap_base)
ffffeb0000000000	-21 TB	ffffebffffffffff	1 TB	... unused hole
ffffec0000000000	-20 TB	fffffbffffffffff	16 TB	KASAN shadow memory
				Identical layout to the 56-bit one from here on:
fffffc0000000000	-4 TB	fffffdffffffffff	2 TB	... unused hole vaddr_end for KASLR
fffffe0000000000	-2 TB	fffffe7fffffffff	0.5 TB	cpu_entry_area mapping
fffffe8000000000	-1.5 TB	fffffeffffffffff	0.5 TB	... unused hole
ffffff0000000000	-1 TB	ffffff7fffffffff	0.5 TB	%esp fixup stacks
ffffff8000000000	-512 GB	ffffffeeffffffff	444 GB	... unused hole
ffffffef00000000	-68 GB	fffffffeffffffff	64 GB	EFI region mapping space
ffffffff00000000	-4 GB	ffffffff7fffffff	2 GB	... unused hole
ffffffff80000000	-2 GB	ffffffff9fffffff	512 MB	kernel text mapping, mapped to physical address 0
ffffffff80000000	-2048 MB			
ffffffffa0000000	-1536 MB	fffffffffeffffff	1520 MB	module mapping space
ffffffffff000000	-16 MB			
FIXADDR_START	~-11 MB	ffffffffff5fffff	~0.5 MB	kernel-internal fixmap range, variable size and offset
ffffffffff600000	-10 MB	ffffffffff600fff	4 kB	legacy vsyscall ABI
ffffffffffe00000	-2 MB	ffffffffffffffff	2 MB	... unused hole

[Адреса ядра]

Как выглядят Page Table Entry

Бит(ы)	Название	Значение
0	present	страница в памяти
1	writable	разрешена запись
2	user accessible	если бит не установлен, то доступ к странице только у ядра
3	write through caching	запись напрямую в память
4	disable cache	отключить кэш для этой страницы
5	accessed	CPU устанавливает этот бит, когда страница используется
6	dirty	CPU устанавливает этот бит, когда происходит запись на страницу
7	huge page/null	нулевой бит в P1 и P4 создаёт страницы 1 КБ в P3, страницу 2 МБ в P2
8	global	страница не заполняется из кэша при переключении адресного пространства (должен быть установлен бит PGE регистра CR4)
9-11	available	ОС может их свободно использовать
12-51	physical address	выровненный по странице 52-битный физический адрес фрейма или следующей таблицы страниц
52-62	available	ОС может их свободно использовать
63	no execute	запрещает выполнение кода на этой странице (должен быть установлен бит NXE в регистре EFER)

Почитать тут

MMU

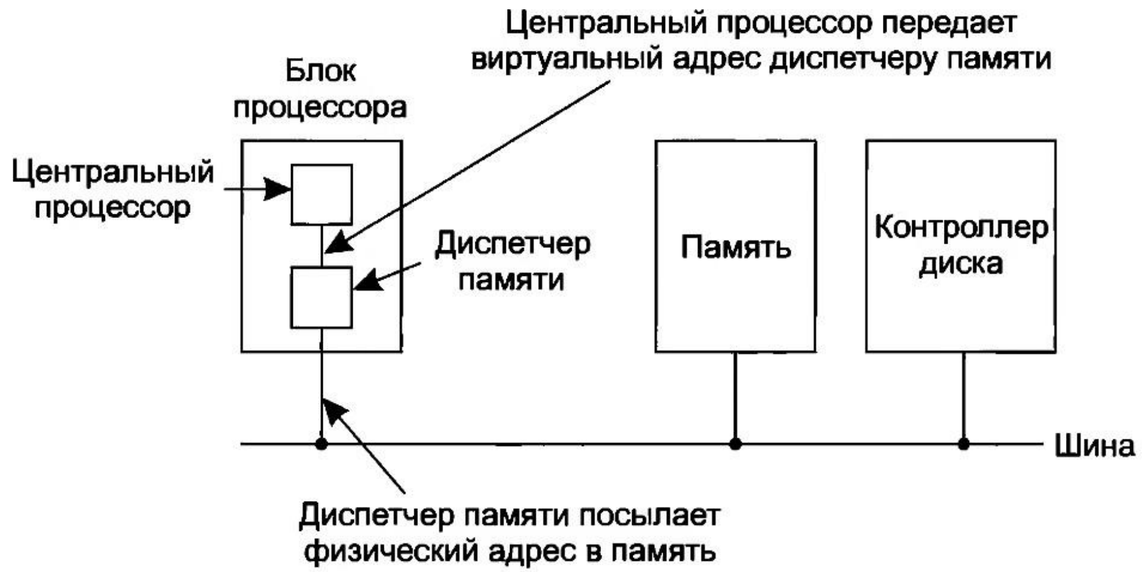


Рис. 3.8. Расположение и предназначение диспетчера памяти (MMU). Здесь диспетчер памяти показан в составе микросхемы центрального процессора, как это чаще всего и бывает в наши дни. Но логически он может размещаться и в отдельной микросхеме, как это было в прошлом

MMU (Memory Management Unit) - диспетчер памяти. Выполняет трансляцию адресов из виртуальных в физические и наоборот.

TLB (Translation Lookaside Buffer)

Таблица 3.1. Буфер быстрого преобразования адреса, используемый для ускорения страничного доступа к памяти

Задействована	Виртуальная страница	Изменена	Защищена	Страничный блок
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

(Всё в режиме ядра)

invlpg [addr] - сбросить вхождение в TLB

mov rax, cr3 ; mov cr3, rax - сбросить **все** вхождения в TLB

Page Fault

- Если виртуальному адресу не поставлен в соответствие физический фрейм (P(present) ==0), то процессор выставляет исключение PageFault.

- Случается при
  - Swap-файлах
  - Copy-on-write
  - Не было чтения ещё

Виды Page Fault

- **Минорный**
- Смысл: в RAM есть нужный фрейм, но нет отображения или фрейм нужно подготовить
- Причины:
  - Есть нужный фрейм, но нет записи в PTE (например, shared library)
  - Запрошенный фрейм не очищен
- Цена: **дёшево**

- **Мажорный**
- Смысл: в RAM нет нужного фрейма
- Причины:
  - Файловый маппинг
- Цена: **дорого (нужно ходить на диск)**

Лимиты

- Бывают жёсткие и мягкие
- $ ulimit -a
- setrlimit / getrlimit
- Размер стека по умолчанию = 8Мб
- Размер кучи = 128Кб + постоянно растёт

mmap

#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

int munmap(void *addr, size_t length);

- **addr** - подсказка, откуда аллоцировать память
- **length** - сколько байт
- **prot** - флаги доступа PROT_NONE/PROT_WRITE/PROT_READ/PROT_EXEC
- **flags** - обновлять ли фреймы у других процессов/файлов
  - **MAP_SHARED** - обнволять у всех владельцев
  - **MAP_PRIVATE** - приватная версия, Copy-on-Write
  - **MAP_ANONYMOUS** - не мапить в файл
  - **MAP_UNINITIALIZED** - неинициализировать страницы
- **MAP_HUGE_2MB**, **MAP_HUGE_1GB** - Huge Pages
- **fd** - файловый дескриптор файла, который мы отображаем
- **offset** - смещение в файле, относительно которого отображаем

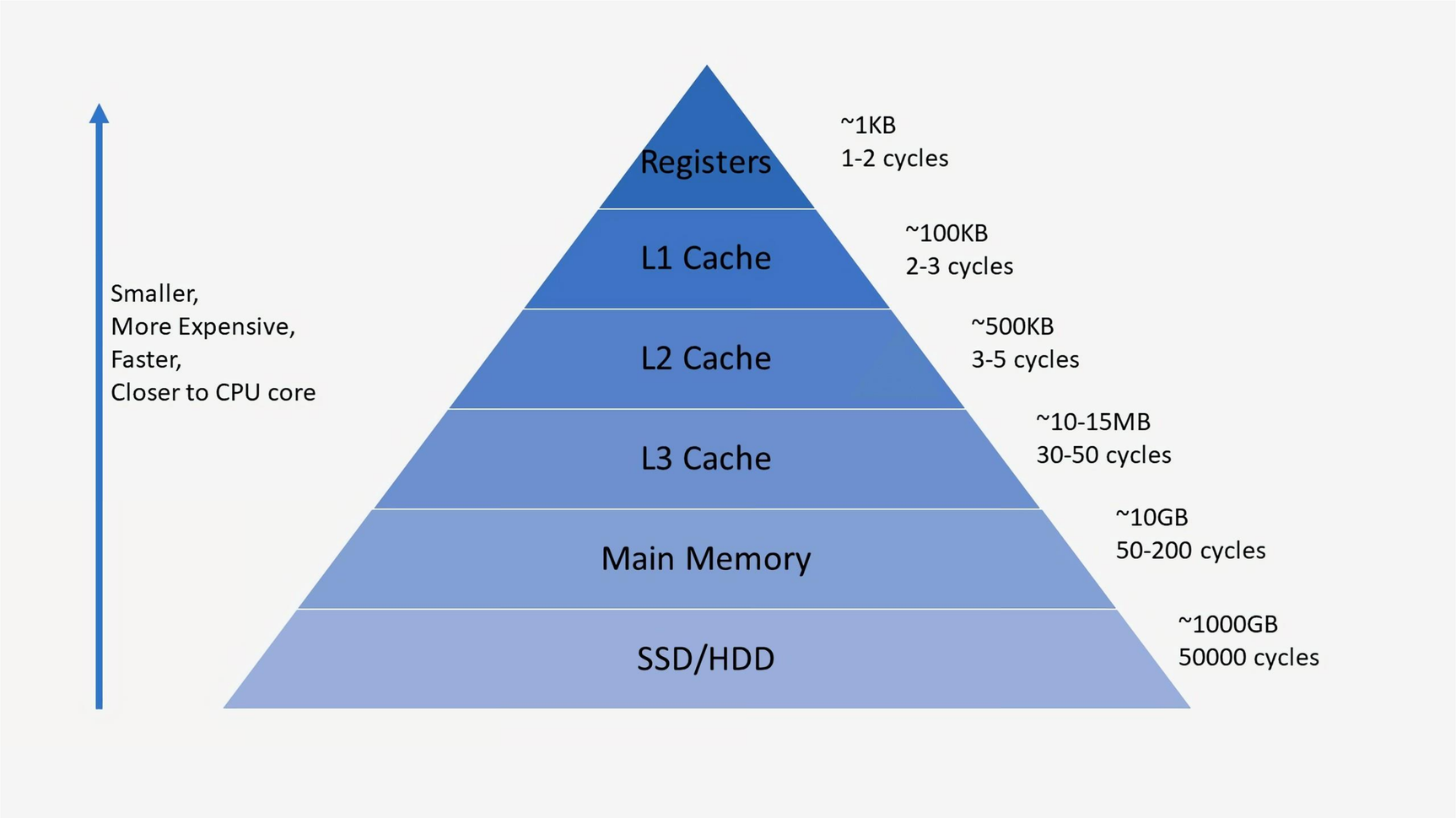
Подсказки

madvice – устанавливает политику работы со страницей
mlock / munlock – запрещает вытеснять страницы из памяти
msync – запрос на синхранизацию страниц (flush страницы)

malloc

- 1) Сначала сдвигает brk, пока размер кучи не составит MMAP_THRESHOLD (128Кб по умолчанию, можно выставить с помощью mallopt)
- 2) Затем, использует mmap, выделяет анонимные страницы
- 3) При обращении по адресу происходит page fault (исключение в процессоре), затем ОС его обрабатывает (мапит фрейм под нужные страницы), возвращает управление процессу, снова пытается выполнить обращение к памяти, успешный успех

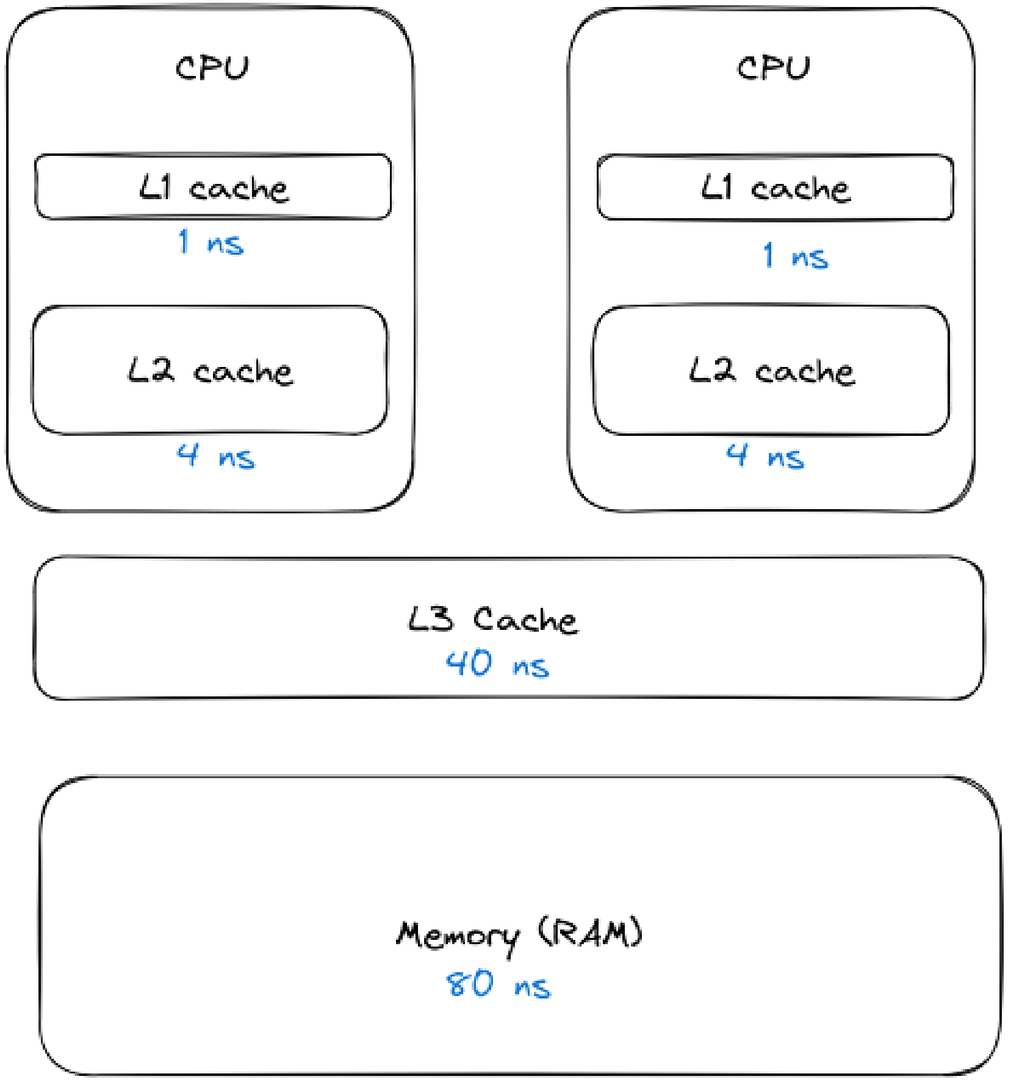
Caches



$lscpu -C

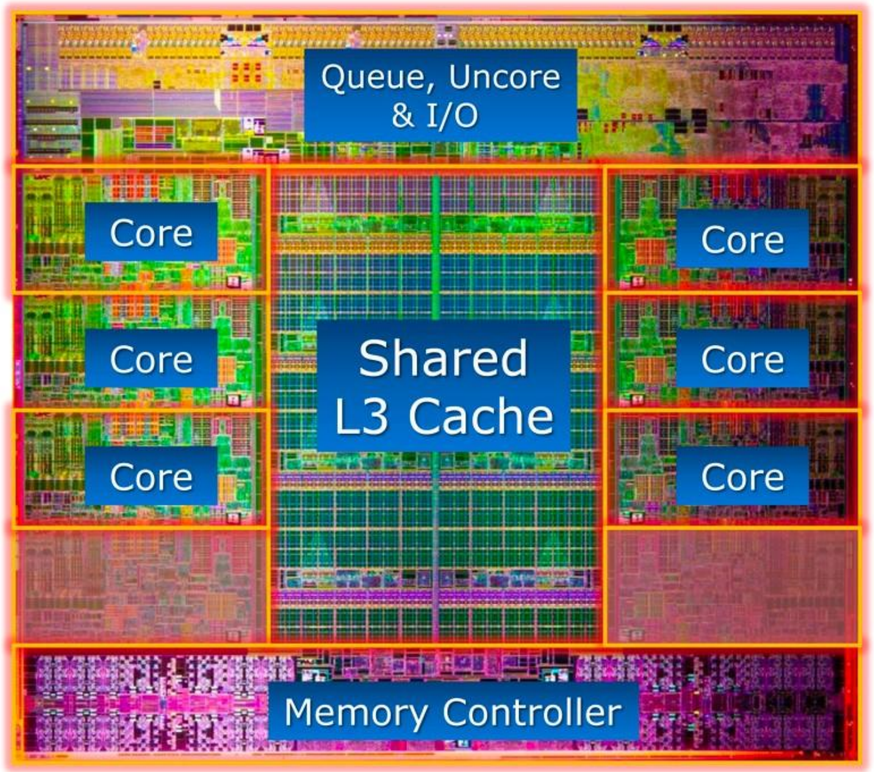
$ls /sys/devices/system/cpu/cpu*/cache

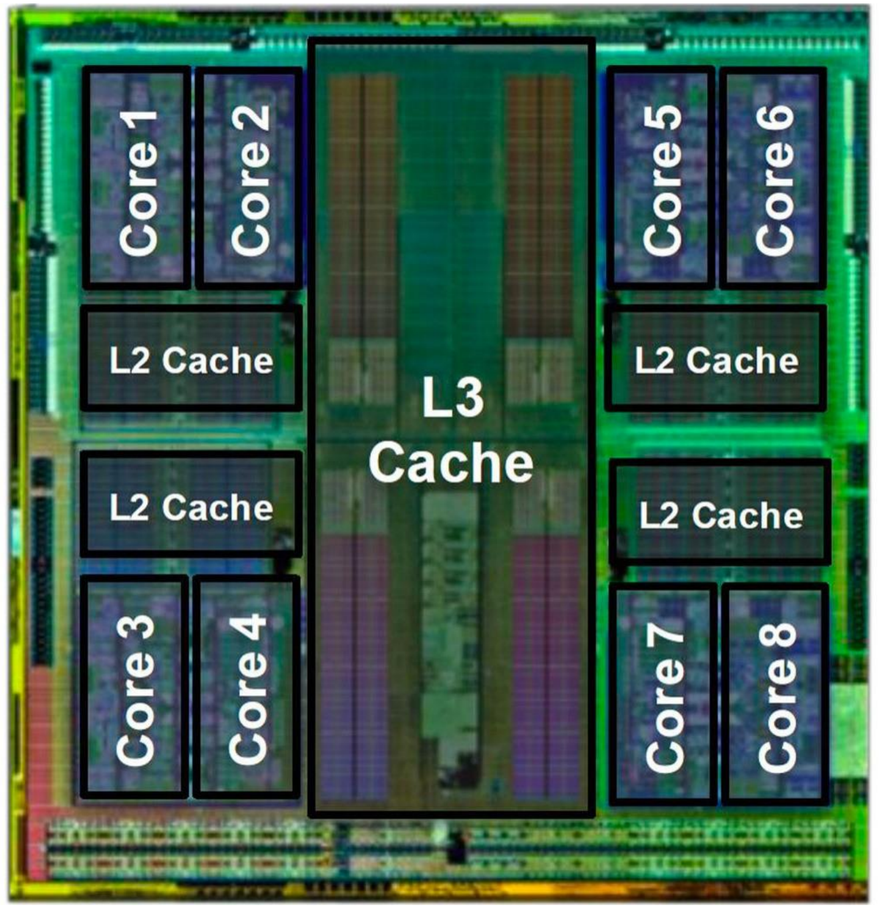
Caches


CPU
L1 cache
1 ns
L2 cache
4 ns
CPU
L1 cache
1 ns
L2 cache
4 ns
L3 Cache
40 ns
Memory (RAM)
80 ns

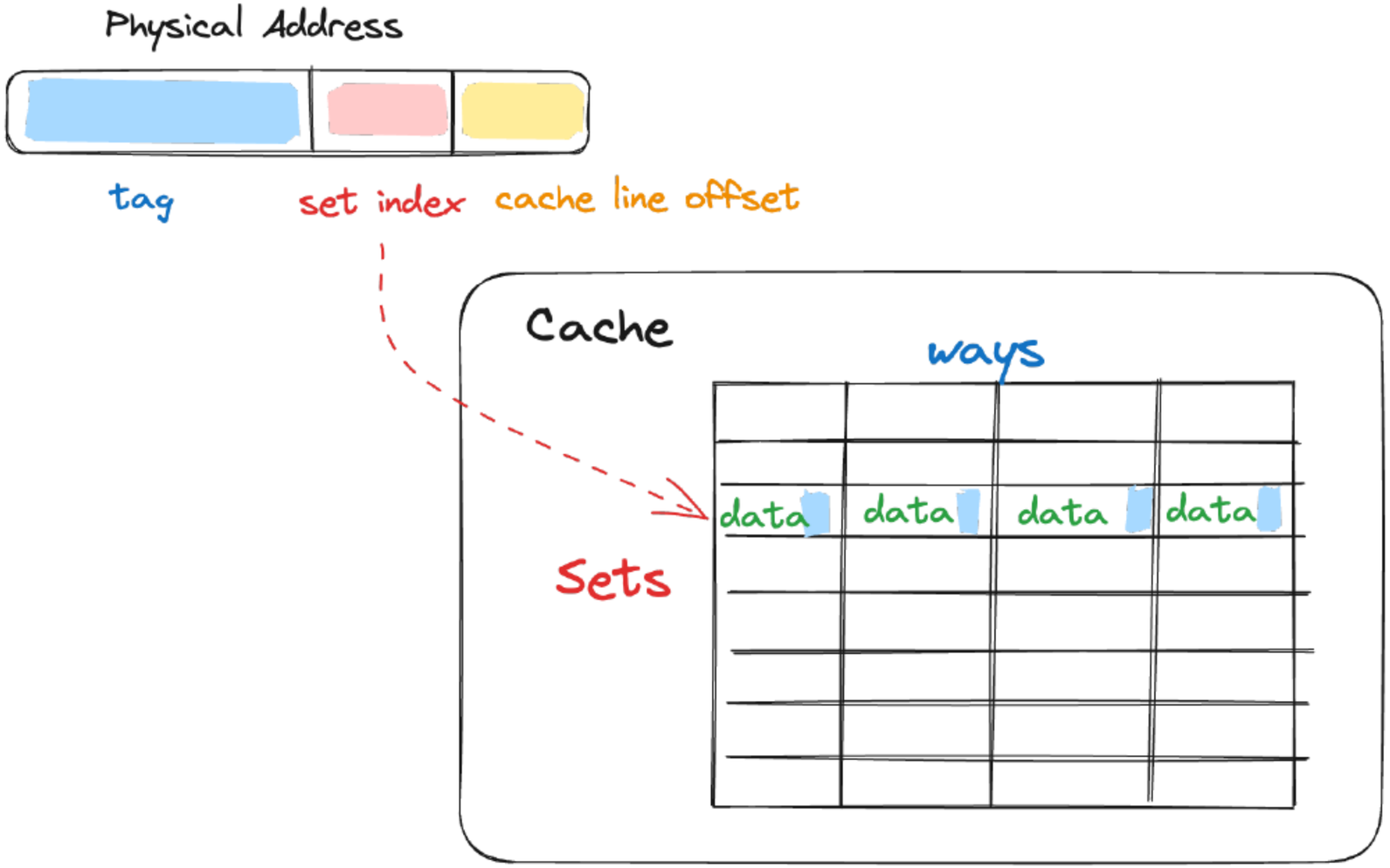
Расположение кэшей на кристалле

Intel® Core™ i7-3960X Processor Die Detail

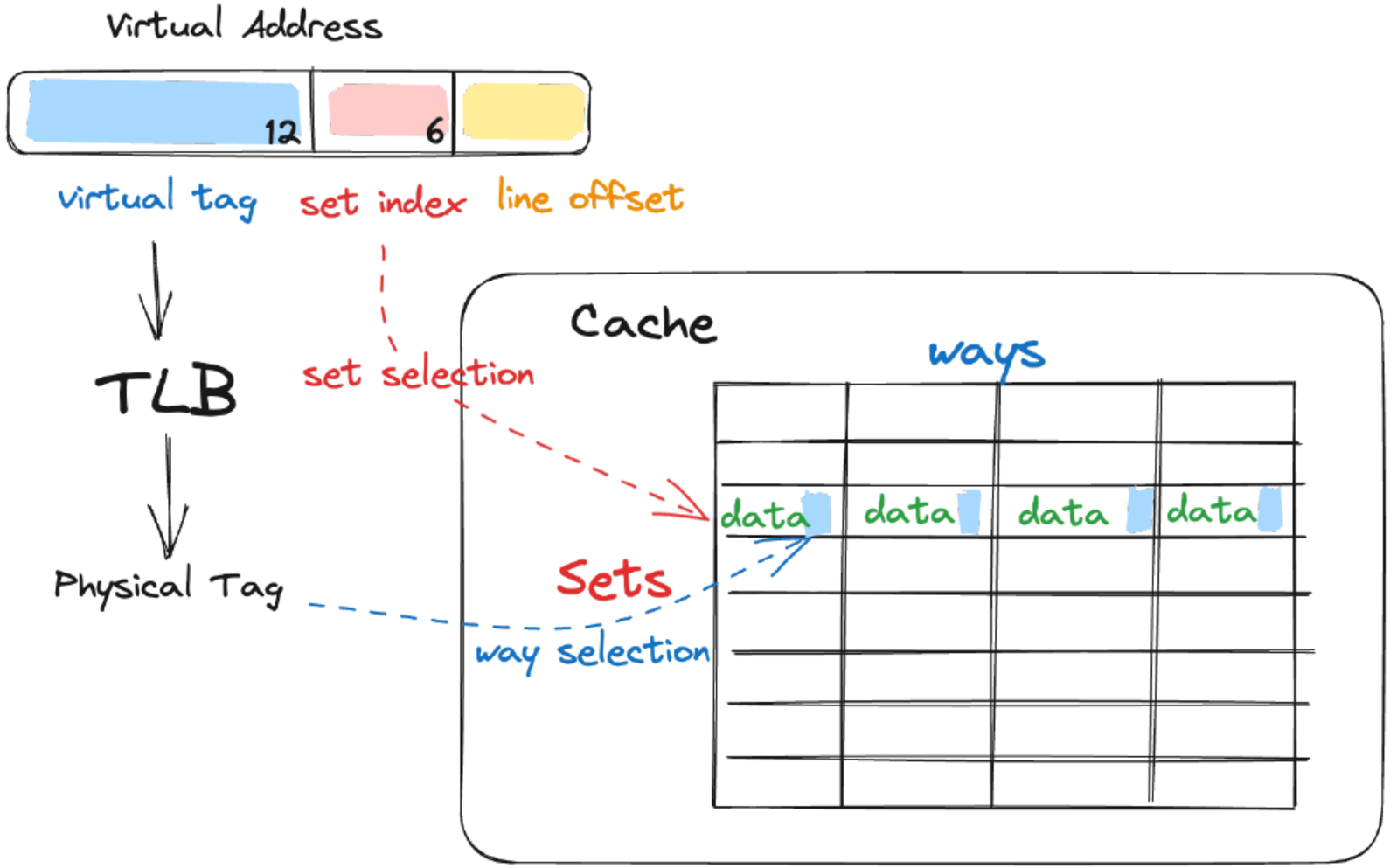




Устройство кэша



Устройство кэша



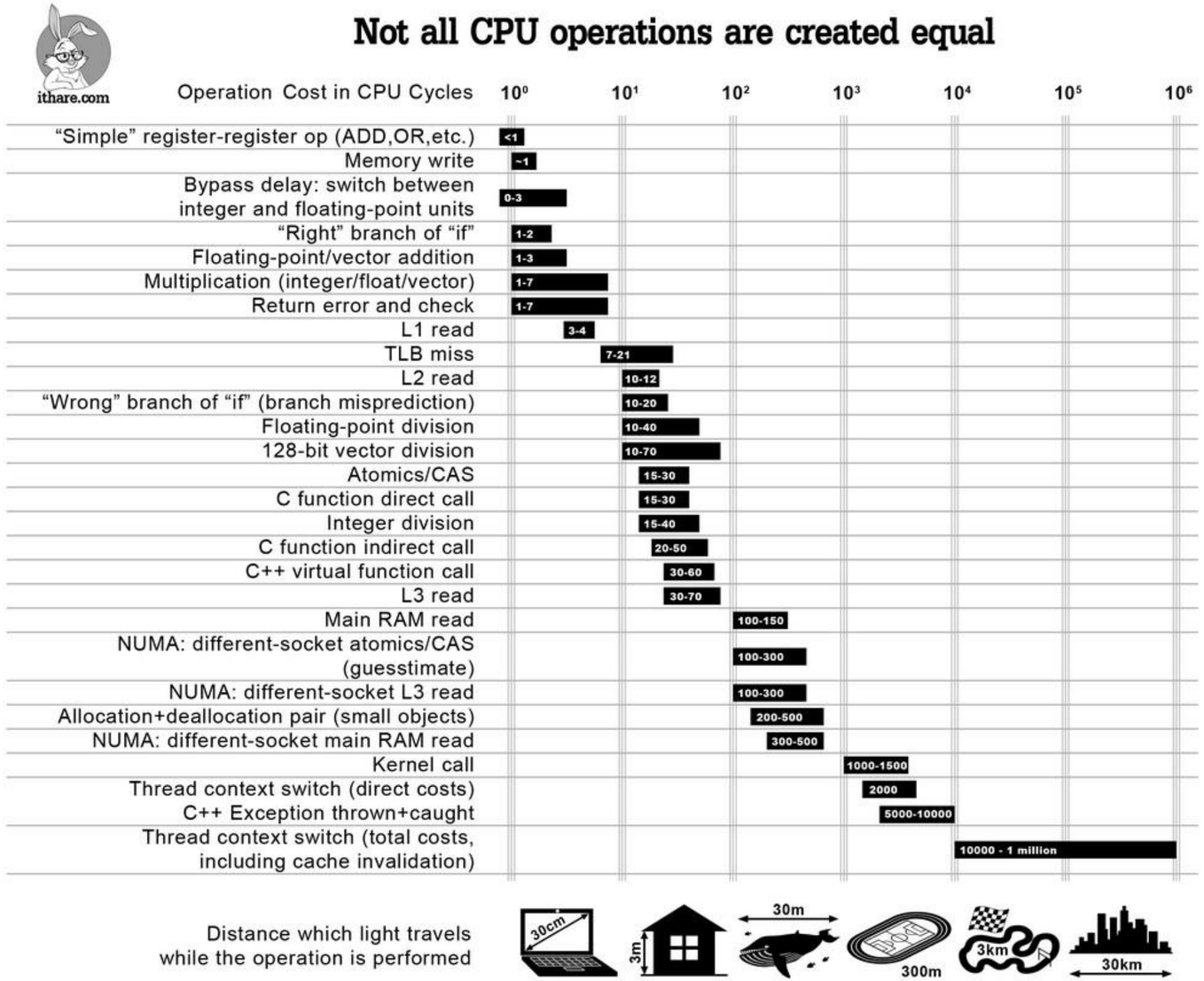
Cache misses

- Типовые значения:
  - 3-10% для L1
  - Может быть меньше 1% для L2

Что почитать и посмотреть

- Статья на хабре про кэши
- What every programmer should know about memory
- Лекции Д.С.Северова
- Статья на хабре про умножение матриц

Сравнение скорости операций



ASLR

- **ALSR - address space layout randomization**
- Сегменты располагаются по рандомным адресам
- Усложняет взлом программы
- Замедляет загрузку программы
- Усложняет и увеличивает код

Memory overcommitment

- А если запросить много памяти у malloc?
- OOM-Killer
- /proc/self/oom_adj (oom_score)

- Почитать здесь