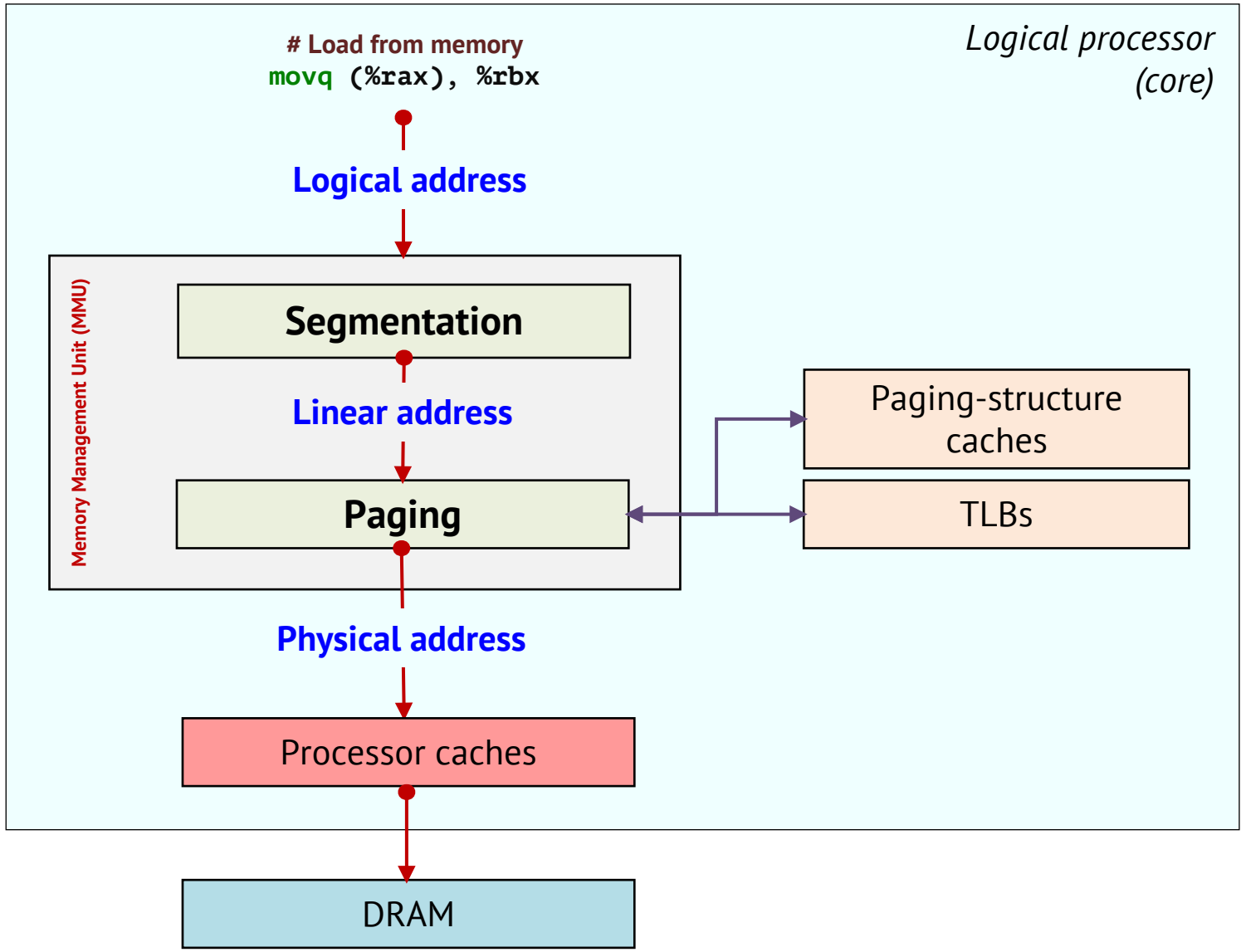
Курс «Архитектурно-ориентированная оптимизация кода»

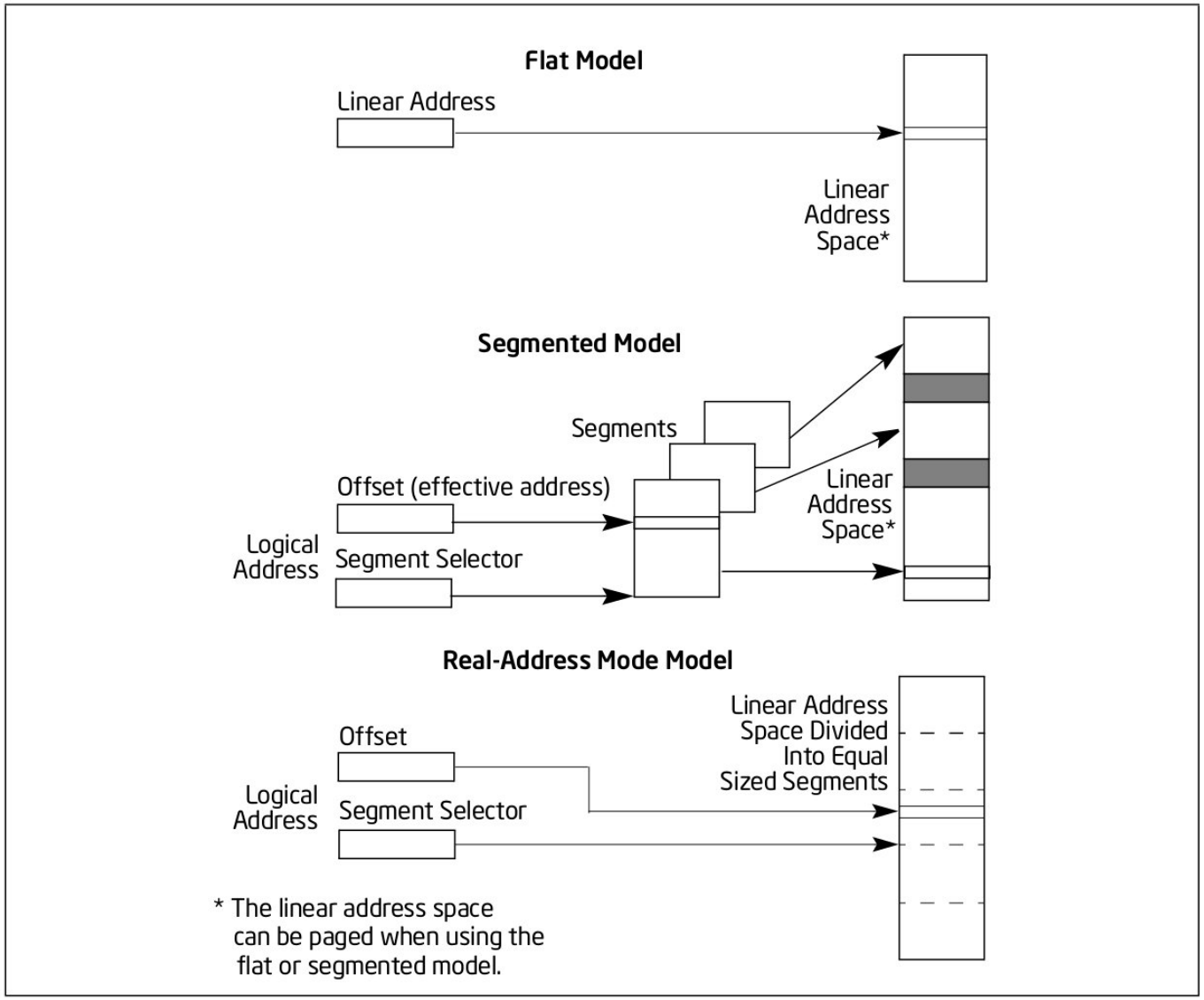
Оптимизация трансляции адресов, большие страницы памяти

Михаил Курносов

Страничная организация памяти



Три модели управления памятью Intel 64

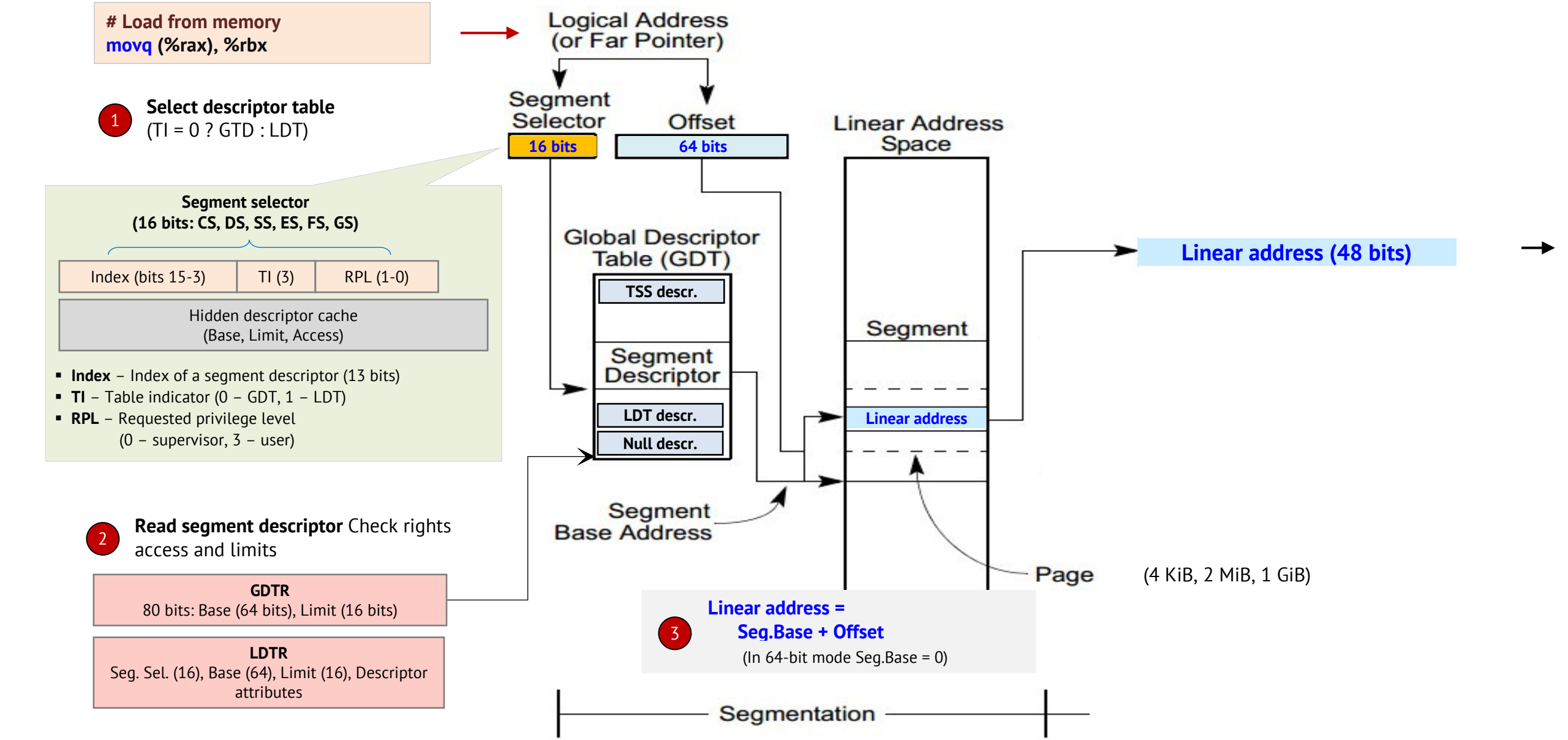


Intel 64 and IA-32 Architectures Software Developer's Manual

Translation: Logical address (SegSel : Offset) --> Linear address (48 bits)

(Vol. 3A Intel 64 and IA-32 Architectures Software Developer's Manual, Chapter 3 Protected Mode Memory Management)

```
# Load from memory
movq (%rax), %rbx
```



1 **Select descriptor table**
(TI = 0 ? GTD : LDT)

Segment selector
(16 bits: CS, DS, SS, ES, FS, GS)

Index (bits 15-3)	TI (3)	RPL (1-0)

Hidden descriptor cache
(Base, Limit, Access)

- **Index** – Index of a segment descriptor (13 bits)
- **TI** – Table indicator (0 – GDT, 1 – LDT)
- **RPL** – Requested privilege level
  (0 – supervisor, 3 – user)

2 **Read segment descriptor** Check rights access and limits

GDTR
80 bits: Base (64 bits), Limit (16 bits)

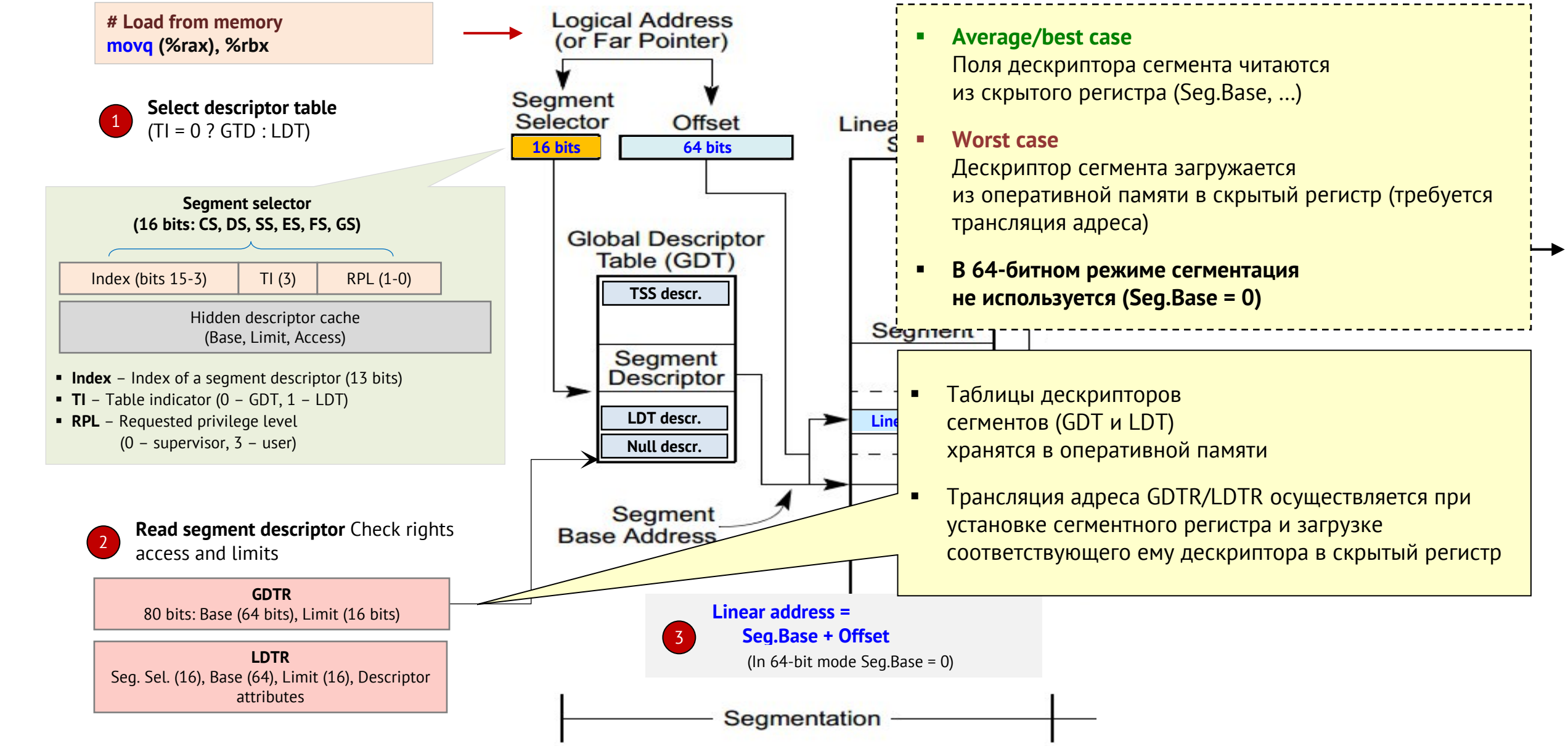
LDTR
Seg. Sel. (16), Base (64), Limit (16), Descriptor attributes

3 **Linear address =**
Seg.Base + Offset
(In 64-bit mode Seg.Base = 0)

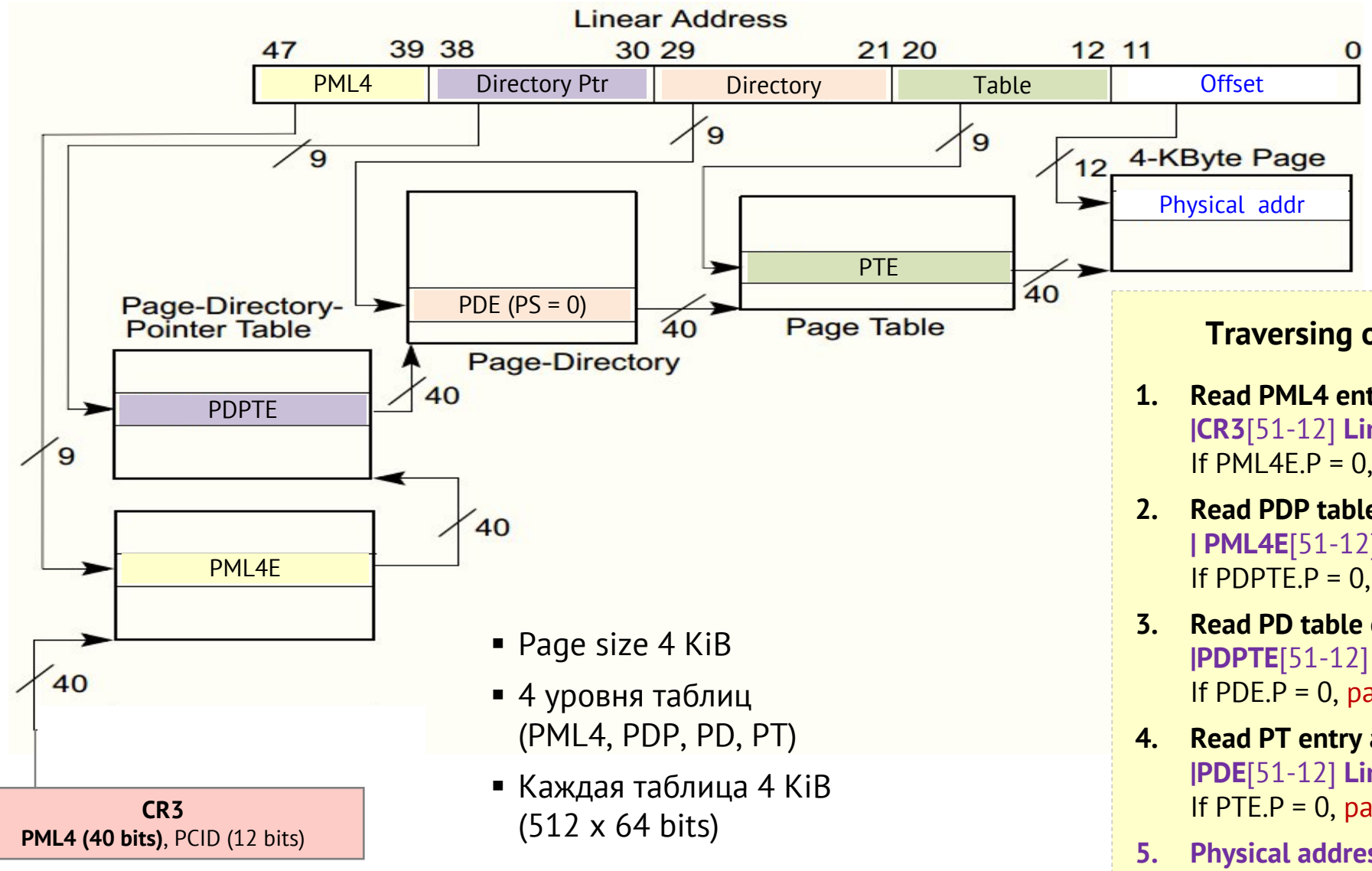
Translation: Logical address (SegSel : Offset) --> Linear address (48 bits)

(Vol. 3A Intel 64 and IA-32 Architectures Software Developer's Manual, Chapter 3 Protected Mode Memory Management)

```
# Load from memory
movq (%rax), %rbx
```



Translation: Linear address (48 bits) --> Physical address (52 bits)



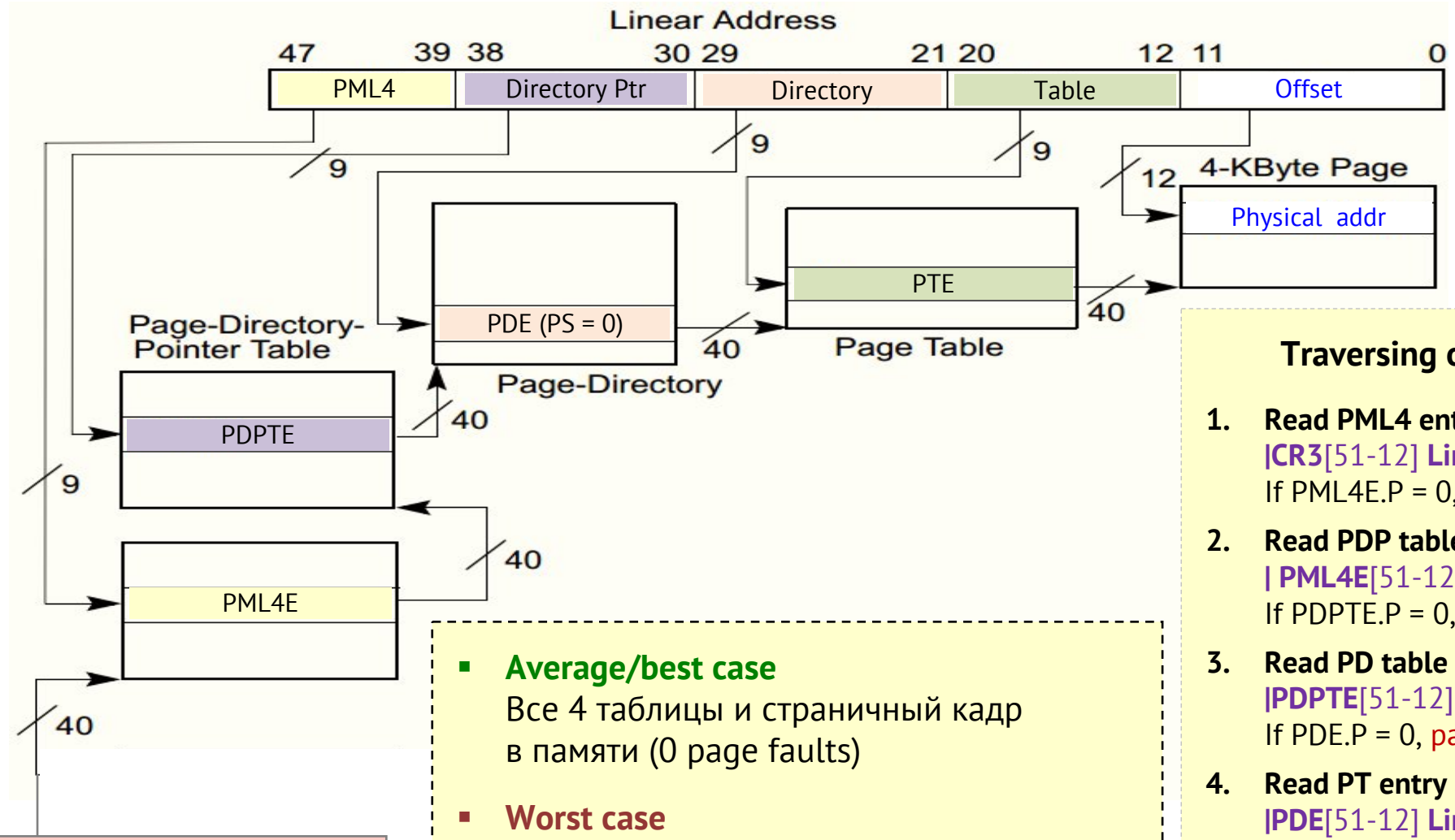
Physical addr (52 bits) =
PTE.Address + Offset

- Page size 4 KiB
- 4 уровня таблиц (PML4, PDP, PD, PT)
- Каждая таблица 4 KiB (512 x 64 bits)

Traversing of paging-structure hierarchy

1. **Read PML4 entry at physical address (52 bits)**
   |CR3[51-12] **LinearAddr**[47-39] **000|**
   If PML4E.P = 0, page fault, load PDP table
2. **Read PDP table entry at physical address**
   | PML4E[51-12] **LinearAddr**[38-30] **000|**
   If PDPTE.P = 0, page fault, load PD table
3. **Read PD table entry at physical address**
   |PDPTE[51-12] **LinearAddr**[29-21] **000|**
   If PDE.P = 0, page fault, load Page table
4. **Read PT entry at physical address**
   |PDE[51-12] **LinearAddr**[20-12] **000|**
   If PTE.P = 0, page fault, load Page frame
5. **Physical address =**
   |PTE.Addr[51-12] Offset[11-0]|

Translation: Linear address (48 bits) --> Physical address (52 bits)



Physical addr (52 bits) =
PTE.Address + Offset

- **Average/best case**
  Все 4 таблицы и страничный кадр в памяти (0 page faults)
- **Worst case**
  В памяти нет таблиц PDP, PD, PT и страничного кадра (4 page faults)

Traversing of paging-structure hierarchy

1. **Read PML4 entry at physical address (52 bits)**
   |CR3[51-12] **LinearAddr**[47-39] **000|**
   If PML4E.P = 0, page fault, load PDP table
2. **Read PDP table entry at physical address**
   | PML4E[51-12] **LinearAddr**[38-30] **000|**
   If PDPTE.P = 0, page fault, load PD table
3. **Read PD table entry at physical address**
   |PDPTE[51-12] **LinearAddr**[29-21] **000|**
   If PDE.P = 0, page fault, load Page table
4. **Read PT entry at physical address**
   |PDE[51-12] **LinearAddr**[20-12] **000|**
   If PTE.P = 0, page fault, load Page frame
5. **Physical address =**
   |PTE.Addr[51-12] Offset[11-0]|

TLB (Translation Lookaside Buffer)

- Кеш TLB хранит часто используемые элементы page-directory (PDE) и page-table (PTE)
- Кеш-память TLB разделена на 4 группы:
  - TLB для 4K страницы инструкций
  - TLB для 4K страницы данных
  - TLB для больших странниц инструкций (2 MB, 4 MB, 1 GB)
  - TLB для больших странниц данных (2 MB, 4 MB, 1 GB)

- Процессоры с микроархитектурой Core реализуют один уровень кеш-паямти TLB для инструкций (iTLB) и два уровня для TLB данных (dTLB)
- Процессоры Core i7 включают TLB второго уровня для инструкций и данных (unified)

TLB and paging-structure caches

TLB (ITLB/DTLB, levels 1, 2, …)

Page number (bits 47-12)	**PCID** (12 bits)	**Physical address** (40 bits)	**Access rights** (R/W, U/S, …)	**Attributes** (Dirty, memory type)

PML4 cache

PML4 index (bits 47-39)	**Physical address** (PML4E.Addr, 40 bits)	**Flags** (R/W, U/S, …)

PDPTE cache

|PML4 ind.|PDPT ind.| (bits 47-30)	**Physical address** (PDPTE.Addr, 40 bits)	**Flags** (R/W, U/S, …)

PDE cache

|PML4 ind.|PDPT ind.|PDT ind.| (bits 47-21)	**Physical address** (PDE.Addr, 40 bits)	**Flags** (R/W, U/S, …)

Linear address (48 bits)

1. **Search in TLB by page number (bits 47-12)**
   If found, return address & rights, attrs
2. **Search in PDE cache by bits [47-21]**
   If found, return PDE.Addr & flags
3. **Search in PDPTE cache by bits [47-30]**
   If found, return PDPTE.Addr & flags
4. **Search in PML4 cache by bits [47-39]**
   If found, return PML4E.Addr & flags
5. **Traverse paging-structure hierarchy**
   CR3 -> PML4E -> PDPTE -> PDE -> PTE

Physical address (52 bits)

TLB and paging-structure caches

TLB (ITLB/DTLB, levels 1, 2, …)

Page number (bits 47-12)	**PCID** (12 bits)	**Physical address** (40 bits)	**Access rights** (R/W, U/S, …)	**Attributes** (Dirty, memory type)

PML4 cache

PML4 index (bits 47-39)	**Physical address** (PML4E.Addr, 40 bits)	**Flags** (R/W, U/S, …)

PDPTE cache

|PML4 ind.|PDPT ind.| (bits 47-30)	**Physical address** (PDPTE.Addr, 40 bits)	**Flags** (R/W, U/S, …)

PDE cache

|PML4 ind.|PDPT ind.|PDT ind.| (bits 47-21)	**Physical address** (PDE.Addr, 40 bits)	**Flags** (R/W, U/S, …)

Linear address (48 bits)

Best case

1. **Search in TLB by page number (bits 47-12)**
   If found, return address & rights, attrs
2. **Search in PDE cache by bits [47-21]**
   If found, return PDE.Addr & flags
3. **Search in PDPTE cache by bits [47-30]**
   If found, return PDPTE.Addr & flags
4. **Search in PML4 cache by bits [47-39]**
   If found, return PML4E.Addr & flags

Worst case

5. **Traverse paging-structure hierarchy**
   CR3 -> PML4E -> PDPTE -> PDE -> PTE

Physical address (52 bits)

Invalidation of TLBs and Paging-Structure Caches

- Процессор создает записи в TLB и paging-structure caches (PSC) при трансляции линейных адресов
- Записи из TLB и PSC могут использоваться даже, если таблицы PML4T/PDPT/PDT/PT изменились в памяти
- Операционная систем должна делать недействительными (invalidate) записи в TLB и PSC, информация о которых изменилась в таблицах PML4T/PDPT/PDT/PT
- Инструкции аннулирования записей в TLB и PCS:
  - INVLPG LinearAddress
  - INVPCID
  - MOV to CR0 – invalidates all TLB & PCS entries
  - MOV to CR3 – invalidates all TLB & PCS entries (if CR4.PCIDE = 0)
  - MOV to CR4
  - Task switch
  - VMX transitions

Параметры TLB

```
$ cpuid
   Intel Tiger Lake-U B0 [Willow Cove] {Sunny Cove}, 10nm++
   Deterministic Address Translation Parameters
      4KB page size entries supported = true
      ways of associativity           = 0x8 (8)
      number of sets = 0x00000010 (16)
      translation cache type            = instruction TLB
   Deterministic Address Translation Parameters
      2MB page size entries supported = true
      4MB page size entries supported = true
      ways of associativity           = 0x8 (8)
      number of sets = 0x00000002 (2)
      translation cache type            = instruction TLB
   Deterministic Address Translation Parameters (0x
      4KB page size entries supported = true
      ways of associativity           = 0x10 (16)
      number of sets = 0x00000001 (1)
      translation cache type            = store-only TLB
      translation cache level           = 0x2 (2)
      fully associative                 = true
   Deterministic Address Translation Parameters (0x
      4KB page size entries supported = true
      ways of associativity           = 0x4 (4)
      number of sets = 0x00000010 (16)
      translation cache type            = load-only TLB
      translation cache level           = 0x2 (2)
   Deterministic Address Translation Parameters
      2MB page size entries supported = true
      4MB page size entries supported = true
      ways of associativity           = 0x4 (4)
      number of sets = 0x00000008 (8)
      translation cache type            = load-only TLB
      translation cache level           = 0x2 (2)
   Deterministic Address Translation Parameters
      1GB page size entries supported = true
      ways of associativity           = 0x8 (8)
      number of sets = 0x00000001 (1)
      translation cache type            = load-only TLB
      translation cache level           = 0x2 (2)
   Deterministic Address Translation Parameters
      4KB page size entries supported = true
      2MB page size entries supported = true
      4MB page size entries supported = true
      ways of associativity           = 0x8 (8)
      number of sets = 0x00000080 (128)
      translation cache type            = unified TLB
      translation cache level           = 0x3 (3)
   Deterministic Address Translation Parameters
      4KB page size entries supported = true
      1GB page size entries supported = true
      ways of associativity           = 0x8 (8)
      number of sets = 0x00000080 (128)
      translation cache type            = unified TLB
      translation cache level           = 0x3 (3)
```

iTLB 4KB pages
128 entries

iTLB 2/4MB pages
16 entries

Store dTLB
4KB pages
16 entries

Load dTLB
4KB pages
64 entries

Load dTLB
2/4MB pages
32 entries

Load dTLB
1GB pages
8 entries

Unified STLB
4KB, 2/4MB
pages
1024 entries

Unified STLB
4KB, 1GB
pages
1024 entries

- DTLB разделено на два — по типу операции load и store
- iTLB, store & load TLBs, STLB (unified TLB L2)

Transparent Huge Pages (x86-64)
(оптимизация работы с TLB)

Transparent Huge Pages (x86-64)

- Поддержка ядром Linux больших страниц памяти

```
$ grep HUGETLB /boot/config-$(uname -r)
CONFIG_CGROUP_HUGETLB=y
CONFIG_ARCH_WANT_GENERAL_HUGETLB=y
CONFIG_HUGETLBFS=y
CONFIG_HUGETLB_PAGE=y
CONFIG_HUGETLB_PAGE_FREE_VMEMMAP=y
# CONFIG_HUGETLB_PAGE_FREE_VMEMMAP_DEFAULT_ON is not set

$ hugeadm --page-sizes-all
2097152
1073741824

$ grep pse /proc/cpuinfo                          # 2MB
pages
flags           : [...] pse [...]

$ grep pdpe1gb /proc/cpuinfo                      # 1GB
pages
flags           : [...] pdpe1gb [...]
```

Architecture	Huge Page Size
arm64	4K, 2M, 1G (64K, 512M: CONFIG_ARM64_64K_PAGES
i386	4K, 4M (2M PAE mode)
ia64	4K, 8K, 64K, 256K, 1M, 4M, 16M, 256M
ppc64	4K, 16M

Transparent Huge Pages (x86-64)

- Поддержка ядром Linux больших страниц памяти

```
$ hugeadm --pool-list
      Size  Minimum  Current  Maximum  Default
   2097152        0        0        0        *
1073741824        0        0        0

$ grep Huge /proc/meminfo
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB

$ hugeadm --list-all-mounts
Mount Point          Options
/dev/hugepages       rw,relatime,pagesize=2M
```

Transparent Huge Pages (x86-64)

- Создание глобального пула больших страниц (/var/lib/hugetlbfs)

```
$ sudo hugeadm –create-global-mounts

$ sudo hugeadm –pool-pages-min 2M:128

$ grep  -i huge /proc/meminfo
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:     128
HugePages_Free:      128
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:          262144 k

$ cat /proc/sys/vm/nr_hugepages
128
```

Псевдослучайный обход массива
(обход связного списка, сортировка)

```c
#define N 10000000

void shuffle(double *data, int *idx, int n)
{
    srand(0);
    for (int i = 0; i < n; i++) {
        idx[i] = i;
        data[i] = i;
    }
    for (int i = 0; i < n; i++) {
        int left = n - i;
        int swap = rand() % left;
        int temp = idx[i];
        idx[i] = idx[swap];
        idx[swap] = temp;
    }
}

void sum(double *data, int *idx, int n)
{   /* Iterate in a random order */
    double s = 0;
    for (int i = 0; i < n; i++) {
        s += data[idx[i]];
    }
    data[0] = s;
}
```

```c
void run()
{
    // Pages = N * sizeof(double) / 4KB = 19532
    double *data = malloc(sizeof(*data) * N);
    // Pages = N * sizeof(int) / 4KB = 9767
    int *idx = malloc(sizeof(*idx) * N);

    shuffle(data, idx, N);
    sum(data, idx, N);

    free(data);
    free(idx);
}
```

```
$ perf stat -e dTLB-loads,dTLB-load-misses, \
               minor-faults,major-faults \
               taskset --cpu-list 0 ./loop

# N=10000000, time (sec) 0.278662, ticks 102
Performance counter stats:
2 234 578 403      dTLB-loads
  49 346 260      dTLB-load-misses # 2,21% of all dTLB cache accesses
     293 113       minor-faults
           0       major-faults
```

Выделение больших страниц памяти: madvise

```
void run_thp()
{
    /* Try to use transparent huge pages (THP) for data and idx arrays */
    void *data = NULL, *idx = NULL;
    size_t data_size = sizeof(double) * N;
    size_t idx_size = sizeof(int) * N;
    size_t huge_page_size = 1 << 21;      /* 2 MiB */

    int rc = posix_memalign(&data, huge_page_size, data_size);
    madvise(data, data_size, MADV_HUGEPAGE);

    rc = posix_memalign(&idx, huge_page_size, idx_size);
    madvise(idx, idx_size, MADV_HUGEPAGE);

    shuffle((double *)data, (int *)idx, N);

    sum((double *)data, (int *)idx, N);

    free(data);
    free(idx);
}
```

- Адрес должен быть выровнен на границу кратную размеру страницы памяти
- madvise(HUGEPAGE)
- В какой момент выделяются страницы памяти?

```
$ perf stat -e dTLB-loads,dTLB-load-misses, \
               minor-faults,major-faults \
               taskset --cpu-list 0 ./loop

# N=10000000, time (sec) 0.278662, ticks 102
Performance counter stats:

1 912 095 744      dTLB-loads
       4 813       dTLB-load-misses # 0,00% of all dTLB cache accesses
       1 882       minor-faults
           0       major-faults
```

Выделение больших страниц памяти: mmap

```
void run_thp_mmap()
{
    /* Try to use transparent huge pages (THP) for data and idx arrays */
    void *data = NULL, *idx = NULL;
    size_t data_size = sizeof(double) * N;
    size_t idx_size = sizeof(int) * N;
    size_t huge_page_size = 1 << 21;  /* 2 MiB */

    /* Round up to page size */
    size_t data_size_rup = ((data_size + huge_page_size - 1) / huge_page_size) * huge_page_size;
    size_t idx_size_rup = ((idx_size + huge_page_size - 1) / huge_page_size) * huge_page_size;

    data = mmap(NULL, data_size_rup,
                PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS | MAP_HUGETLB, 0, 0);
    idx = mmap(NULL, idx_size_rup,
               PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS | MAP_HUGETLB, 0, 0);

    shuffle((double *)data, (int *)idx, N);
    sum((double *)data, (int *)idx, N);

    munmap(data, data_size_rup);
    munmap(idx, idx_size_rup);
    return t;
}
```

Transparent Huge Pages (x86-64) для секций кода (iTLB)

- Запуск процесса с ипользованием THP для сегментов .text, .data
  - Выравниваем сегмент .text на границу 2MB (повторная компоновка программы)
  - Запускаем программу с размещением сегмента .text на строницах 2MB

```
$ gcc -o prog ./prog.c -Wl,-zcommon-page-size=2097152 -Wl,-zmax-page-size=2097152

$ hugectl --text=2097152 ./prog
```