Виртуальная память

Основы информатики

Компьютерные основы программирования

u.to/DbCmFA

На основе **CMU** 15-213/18-243:
Introduction to Computer Systems

u.to/XoKmFA

Лекция 13, 05 мая 2023

Лектор:

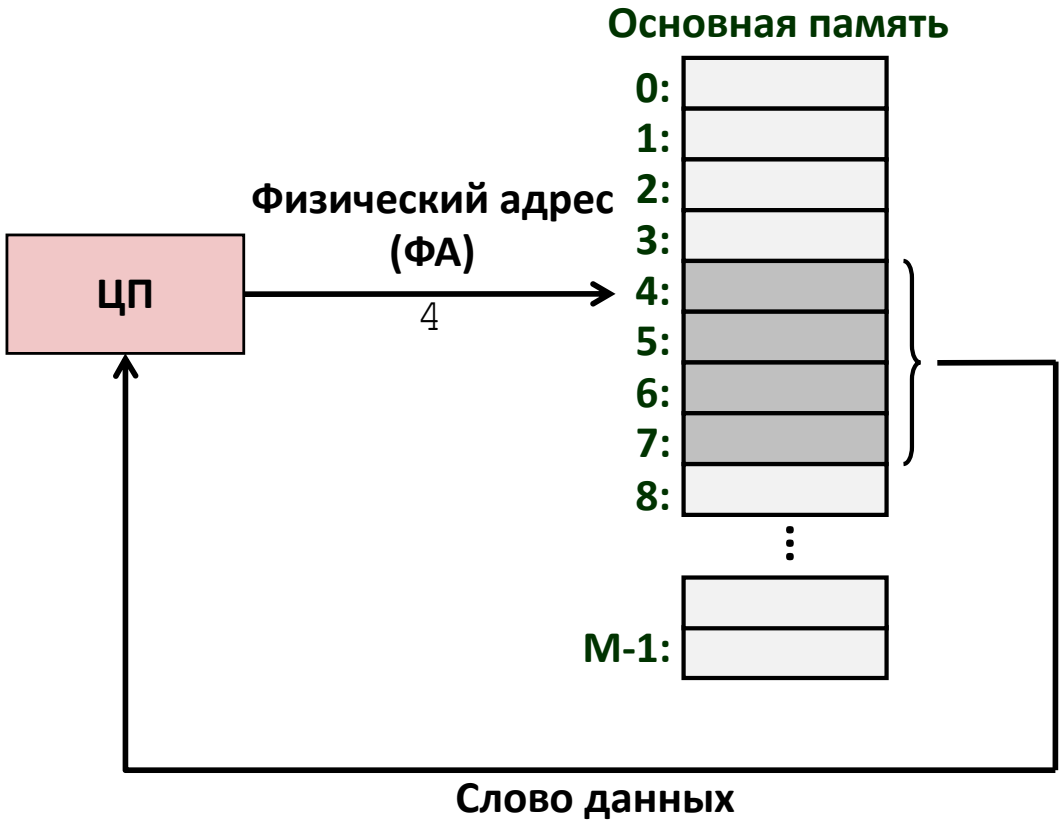
Дмитрий Северов, кафедра информатики 608 КПМ

cs.mipt.ru/wp/?page_id=346

Виртуальная память

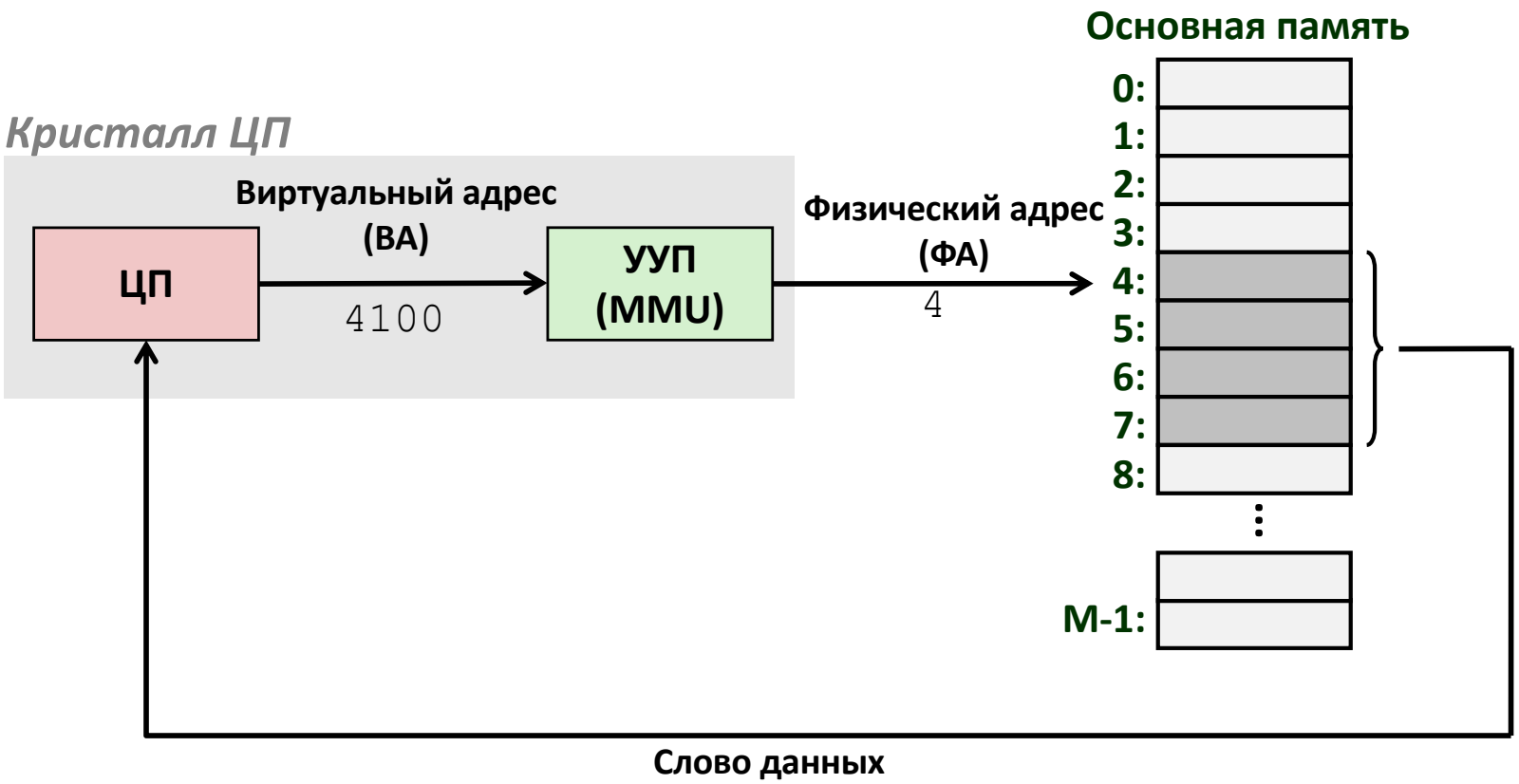
- Пространства адресов
- ВП как средство кеширования
- ВП как средство управления памятью
- ВП как средство защиты памяти
- Трансляция адресов
- Пример простой подсистемы памяти
- Пример подсистемы памяти Core i7/Linux
- Отображение памяти

Система с физической адресацией



- Используется в “простых” системах – микроконтроллерах, встроенных в датчики, регуляторы света, часы

Системы с виртуальной адресацией



- Используется во всех современных серверах, ПК, смартфонах
- Одна из замечательных идей в информатике

Адресные пространства

- **Линейное адресное пространство:** Упорядоченное множество смежных неотрицательных целых адресов: {0, 1, 2, 3 … }

- **Виртуальное адресное пространство:** Множество $N = 2^n$ виртуальных адресов: {0, 1, 2, 3, …, N-1}

- **Физическое адресное пространство:** Множество $M = 2^m$ физических адресов {0, 1, 2, 3, …, M-1}

- **Четкое различием между данными (байтами) и их атрибутами (адресами)**
- **Каждый объект может иметь несколько адресов**
- **Каждый байт в основной памяти имеет:**
  один физический адрес, один (или более) виртуальных адресов

Виртуальная память нужна, чтобы…

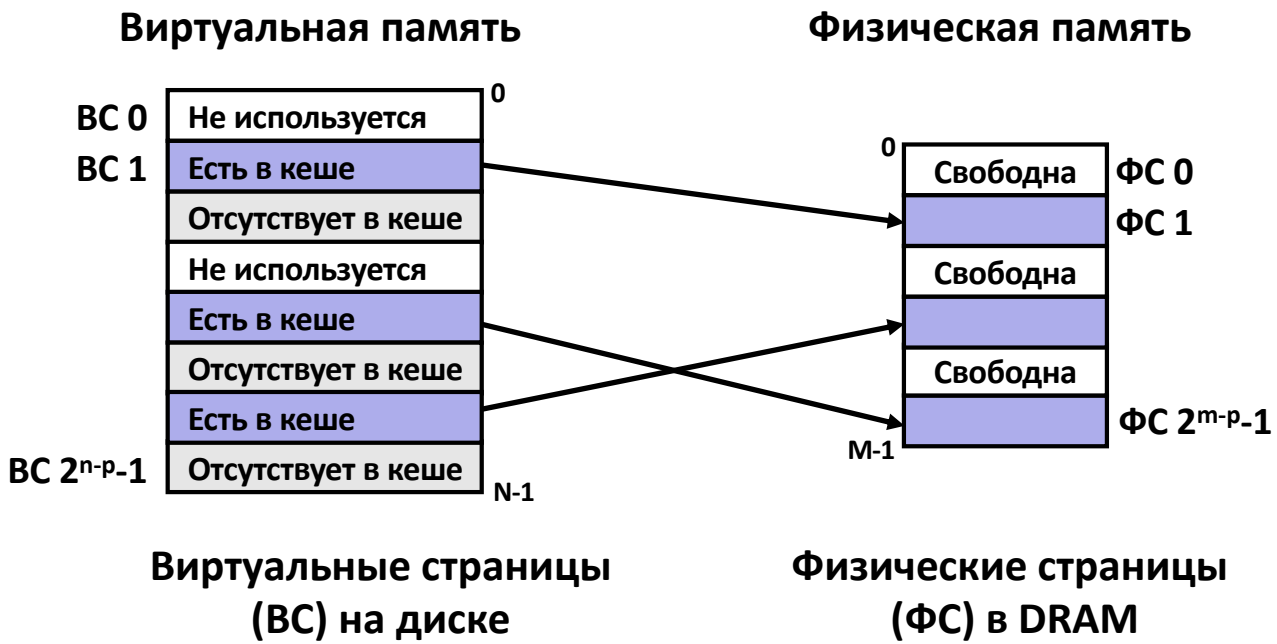
- **использовать физическую память эффективно**
  - DRAM – кеш к частям виртуального пространства
- **упростить управление памятью**
  - Каждый процесс получает типовой экземпляр линейного адресного пространства
- **изолировать адресные пространства**
  - Ни один процесс не может обращаться к памяти другого
  - Программы пользователей не имеют доступа привилегированной информации ядра ОС

Виртуальная память

- Пространства адресов
- ВП как средство кеширования
- ВП как средство управления памятью
- ВП как средство защиты памяти
- Трансляция адресов
- Пример простой подсистемы памяти
- Пример подсистемы памяти Core i7/Linux
- Отображение памяти

ВП как средство кеширования

- ***Виртуальная память -*** **массив N смежных байт на диске**
- **Содержимое массива на диске кешируется в *физической памяти* (*DRAM cache*)**
  - Блоки этого кеша – *страницы* размером $P = 2^p$ байт



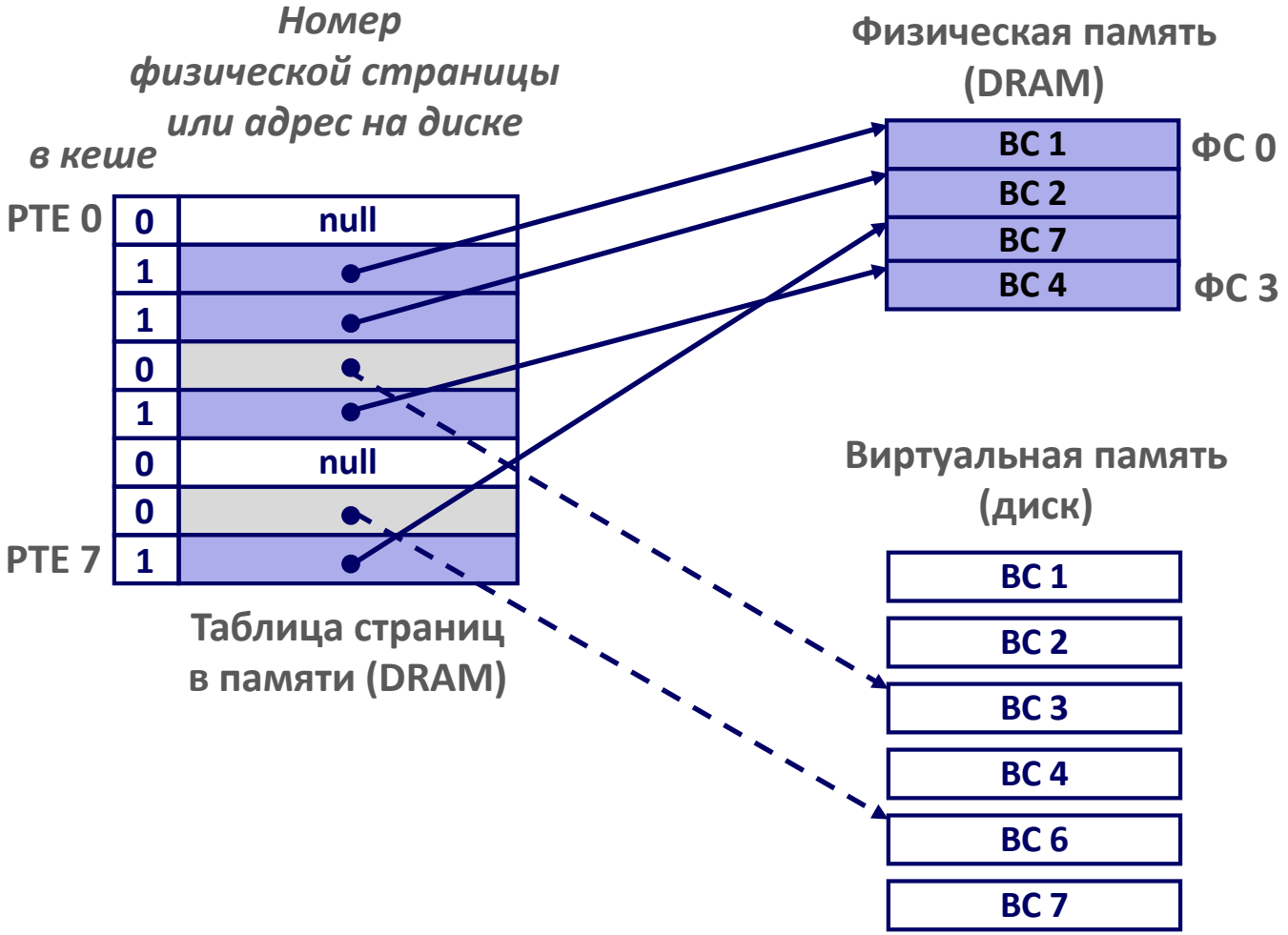
Организация DRAM-кеша

- **Определяется огромной ценой промаха**
  - DRAM на 2 порядка (***100 раз***) медленнее SRAM
  - Диск на 4 порядка (***10 тысяч раз***) медленнее DRAM

- **Следствия**
  - Большой размер страницы (блока): обычно 4-8 KB, иногда 4 MB
  - Полностью ассоциативен
    - любая ВС может быть размещена в любой ФС
    - требует “большую” функцию отображения, в отличие от кешей ЦП
  - Высокоизощрённые, дорогостоящие алгоритмы замены
    - Слишком сложные и неустоявшиеся для аппаратной реализации
  - Write-back, но не write-through

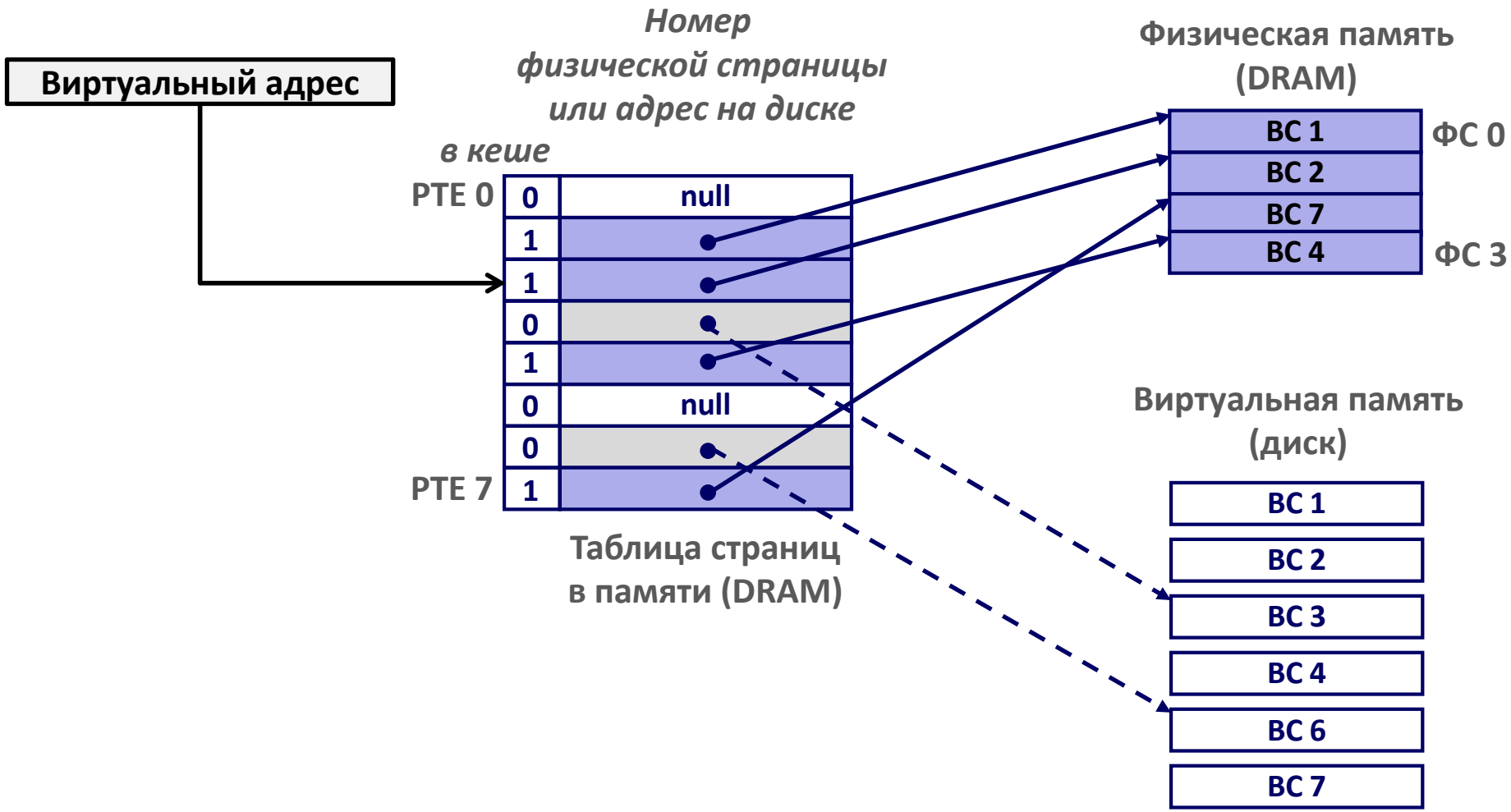
Таблица страниц

- ***Страничная таблица –*** **массив табличных записей (PTE) отображений виртуальных страниц на физические.**
  - Отдельная для каждого процесса структура данных ядра ОС в DRAM



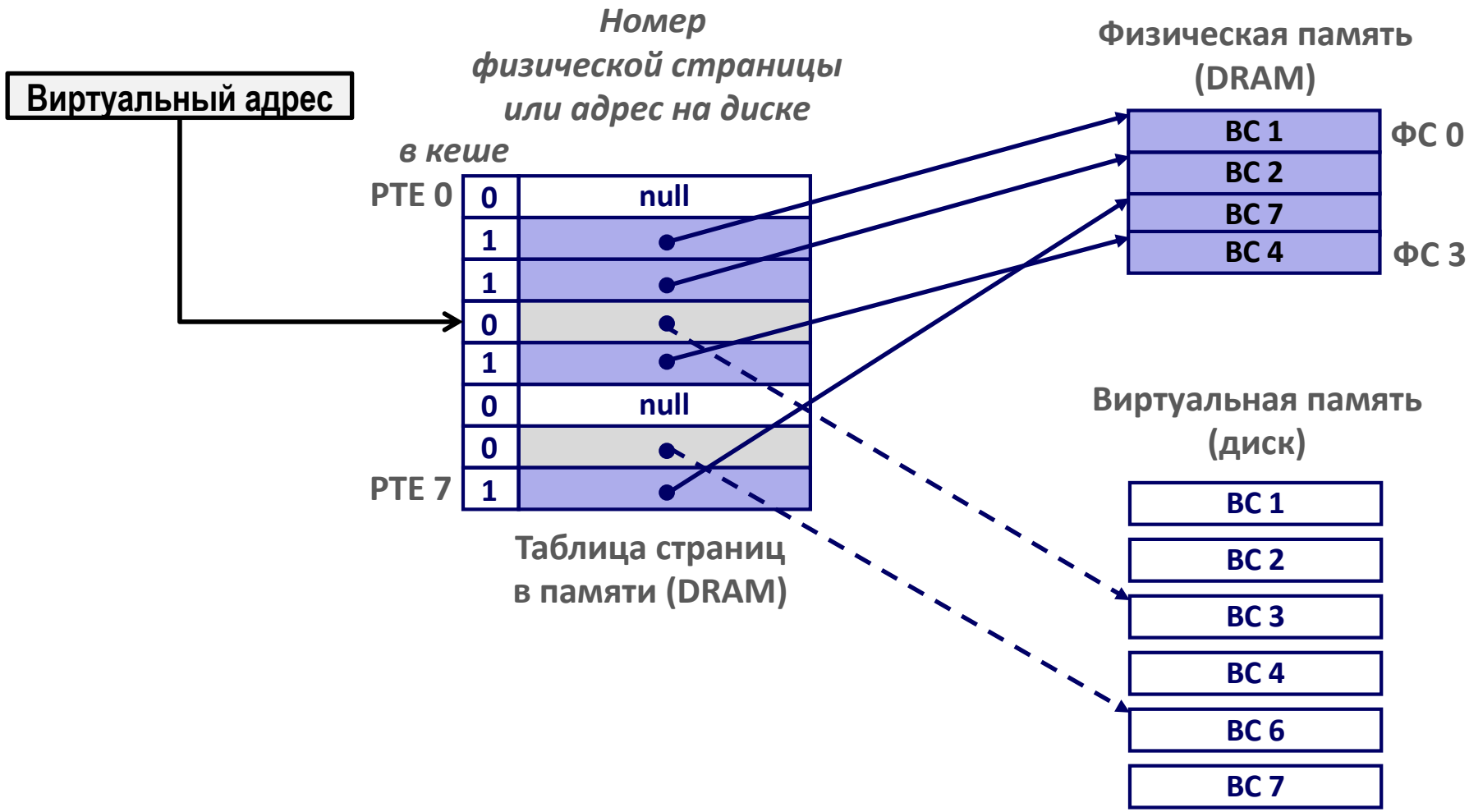
Страничное попадание

- ***Страничное попадание:*** **ссылка на страницу ВП говорит, что страница находится в DRAM (попадание DRAM-кеша)**



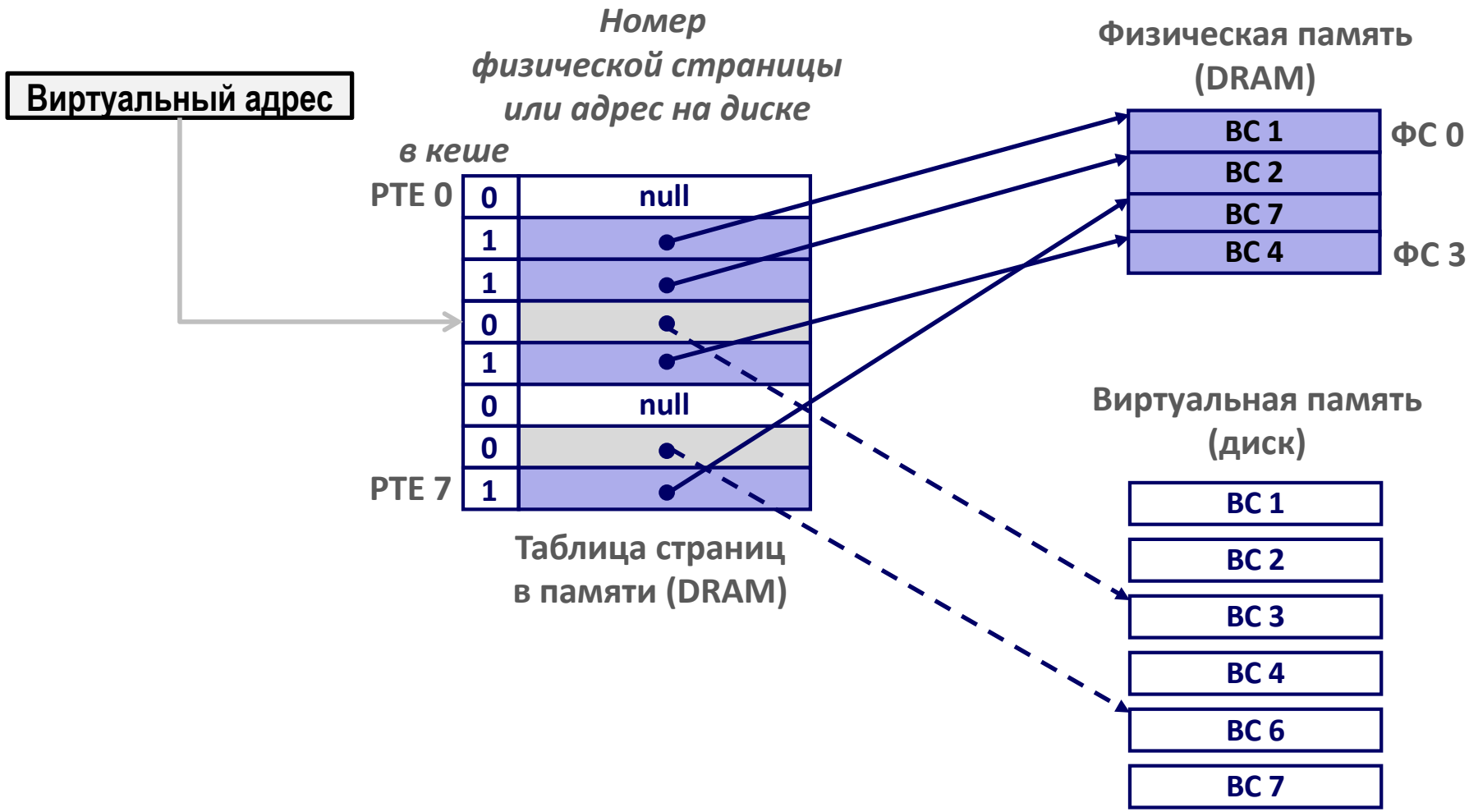
Страничный сбой

- ***Страничный сбой:*** **ссылка на страницу ВП говорит, что страница отсутствует в DRAM (промах DRAM-кеша)**



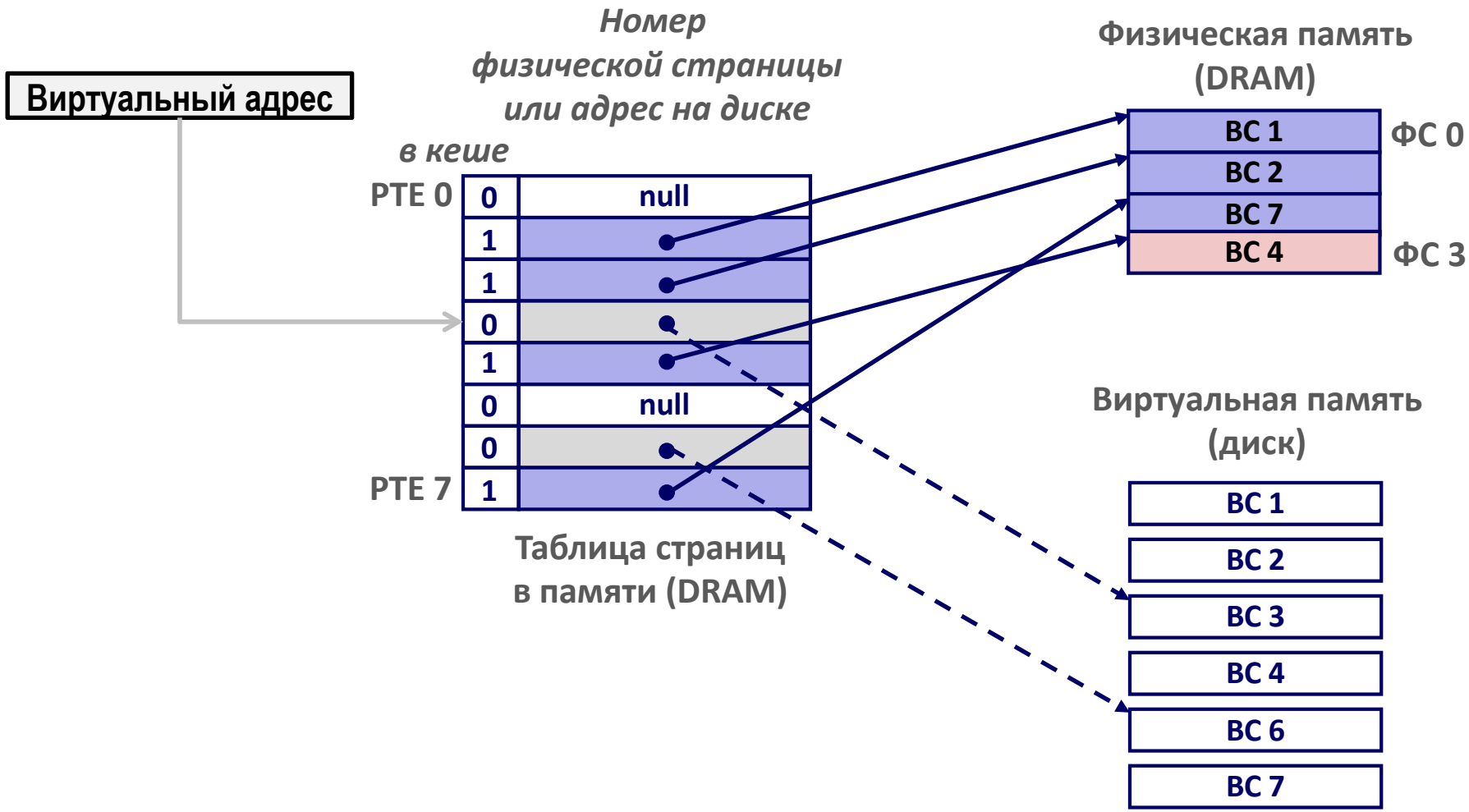
Обработка страничного сбоя – 1

- Страничный промах вызывает страничный сбой (исключение)



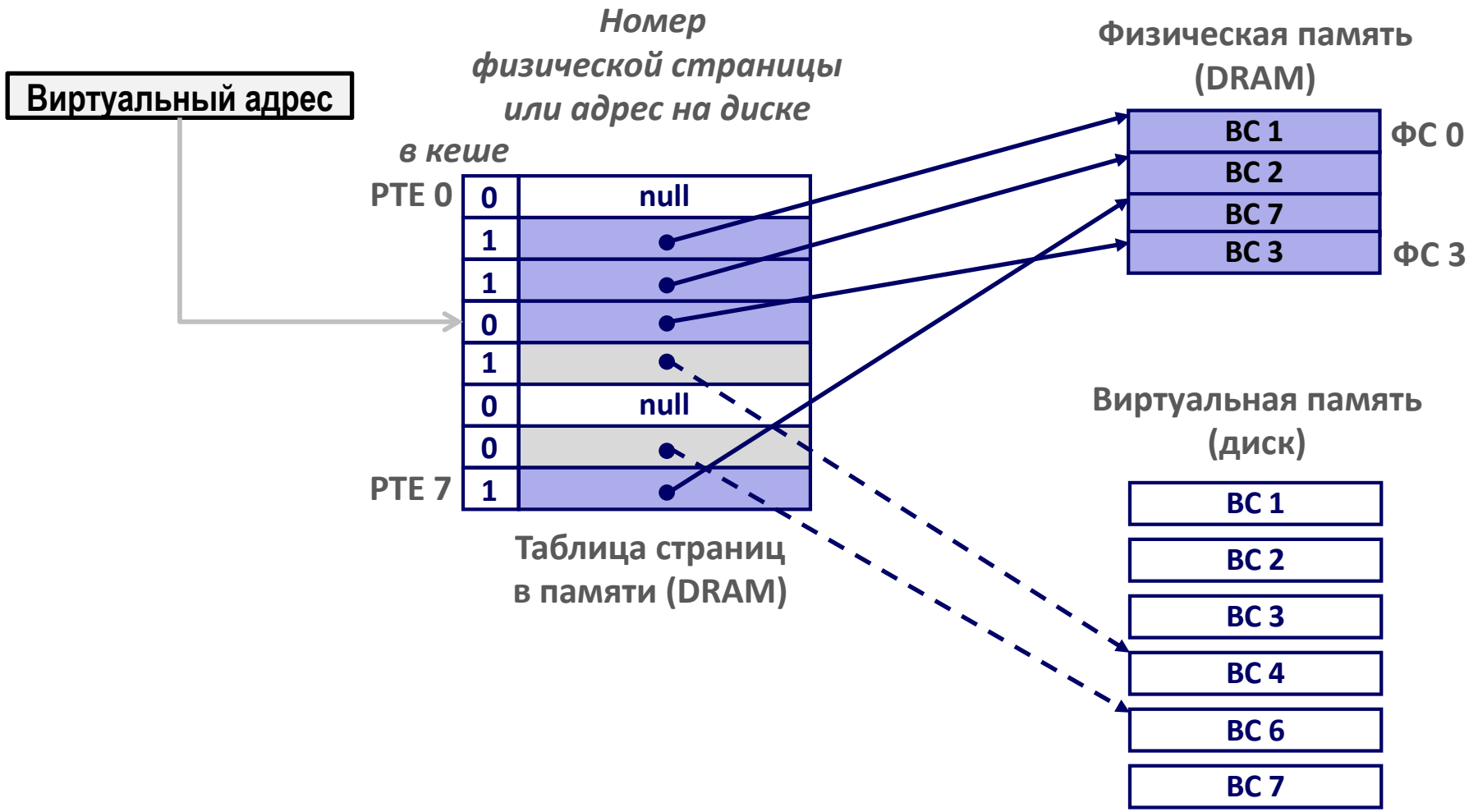
Обработка страничного сбоя – 2

- Страничный промах вызывает страничный сбой (исключение)
- Обработчик страничного сбоя выбирает жертву откачки (здесь ВС 4)



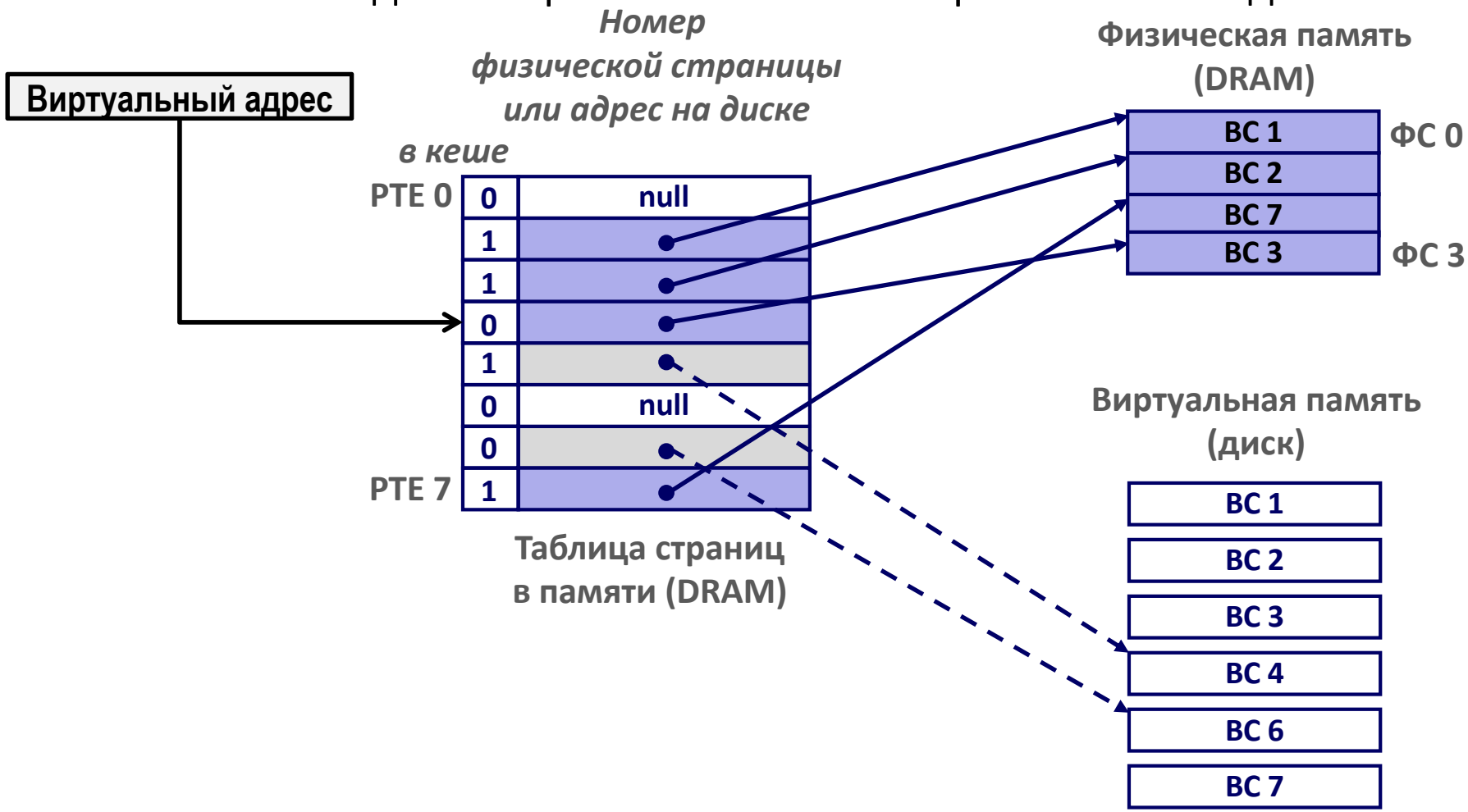
Обработка страничного сбоя - 3

- Страничный промах вызывает страничный сбой (исключение)
- Обработчик страничного сбоя выбирает жертву откачки (здесь ВС 4)
- Обработчик подкачивает с диска в память нужную страницу (здесь ВС 3)



Обработка страничного сбоя - 4

- Страничный промах вызывает страничный сбой (исключение)
- Обработчик страничного сбоя выбирает жертву откачки (здесь ВС 4)
- Обработчик подкачивает с диска в память нужную страницу (здесь ВС 3)
- Сбойная команда повторно выполняется : страничное попадание!



Локальность снова выручает!

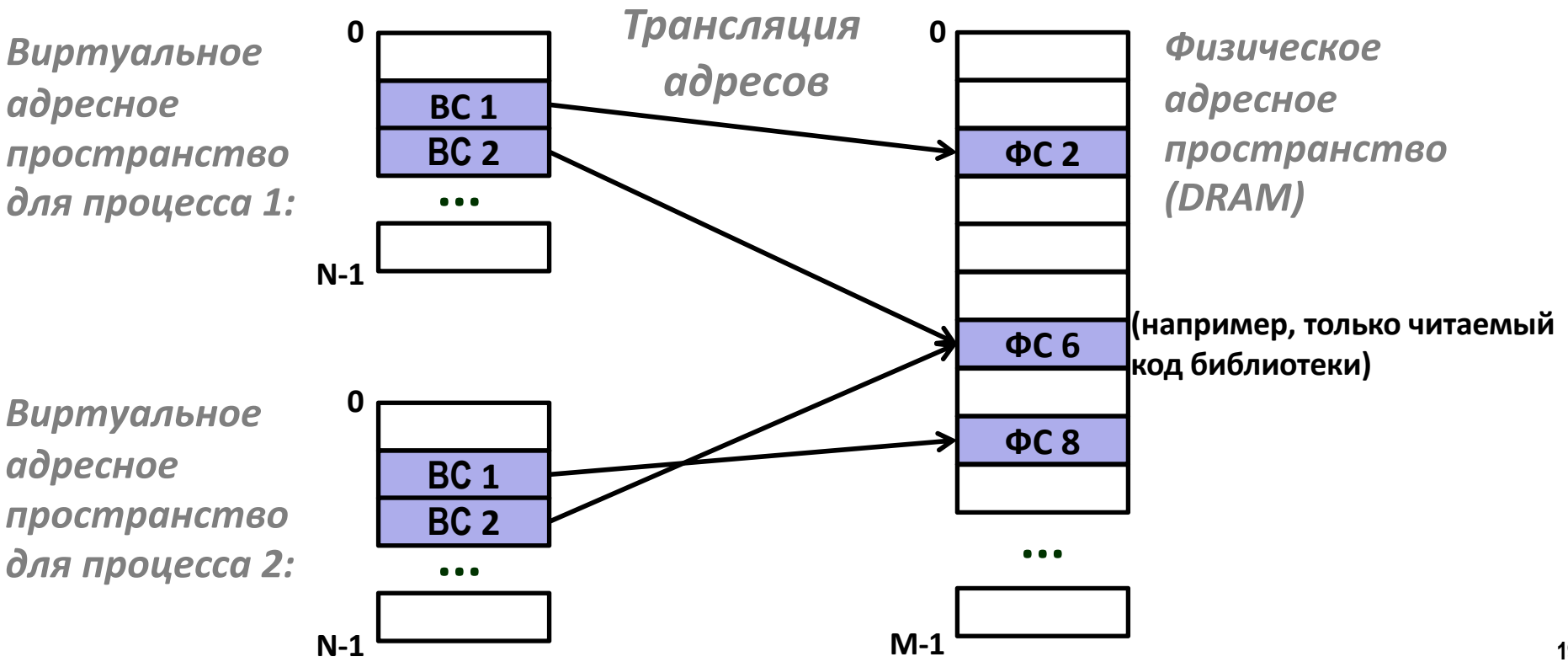
- **Виртуальная память работает благодаря локальности**
- **В каждый момент времени, программы стремятся доступаться к набору активных виртуальных страниц – *рабочему набору***
  - Программы с лучшей временной локальностью будут иметь рабочий набор меньшего размера
- **Если размер рабочего набора < размера основной памяти**
  - Хорошая производительность процесса после неизбежных промахов
- **Если сумма размеров рабочих наборов > размера основной памяти**
  - *Пробуксовка (Thrashing):* деградация производительности при непрерывной откачке/подкачке страниц

Виртуальная память

- Пространства адресов
- ВП как средство кеширования
- ВП как средство управления памятью
- ВП как средство защиты памяти
- Трансляция адресов
- Пример простой подсистемы памяти
- Пример подсистемы памяти Core i7/Linux
- Отображение памяти

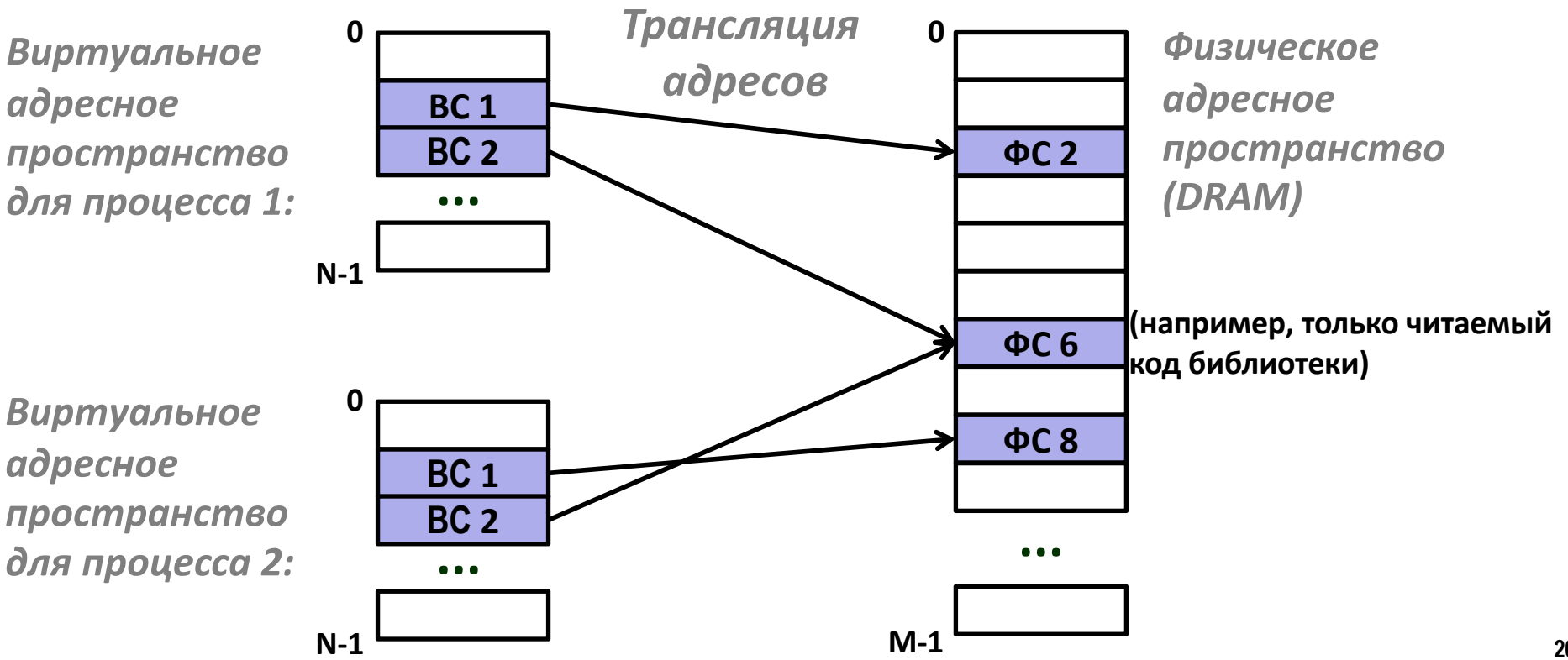
ВП как средство управления памятью - 1

- **Ключевая идея: каждому процессу – собственное виртуальное адресное пространство**
  - память представляется простым линейным массивом
  - отображение разбрасывает адреса по физической памяти
    - удачные отображения упрощают распределение и управление



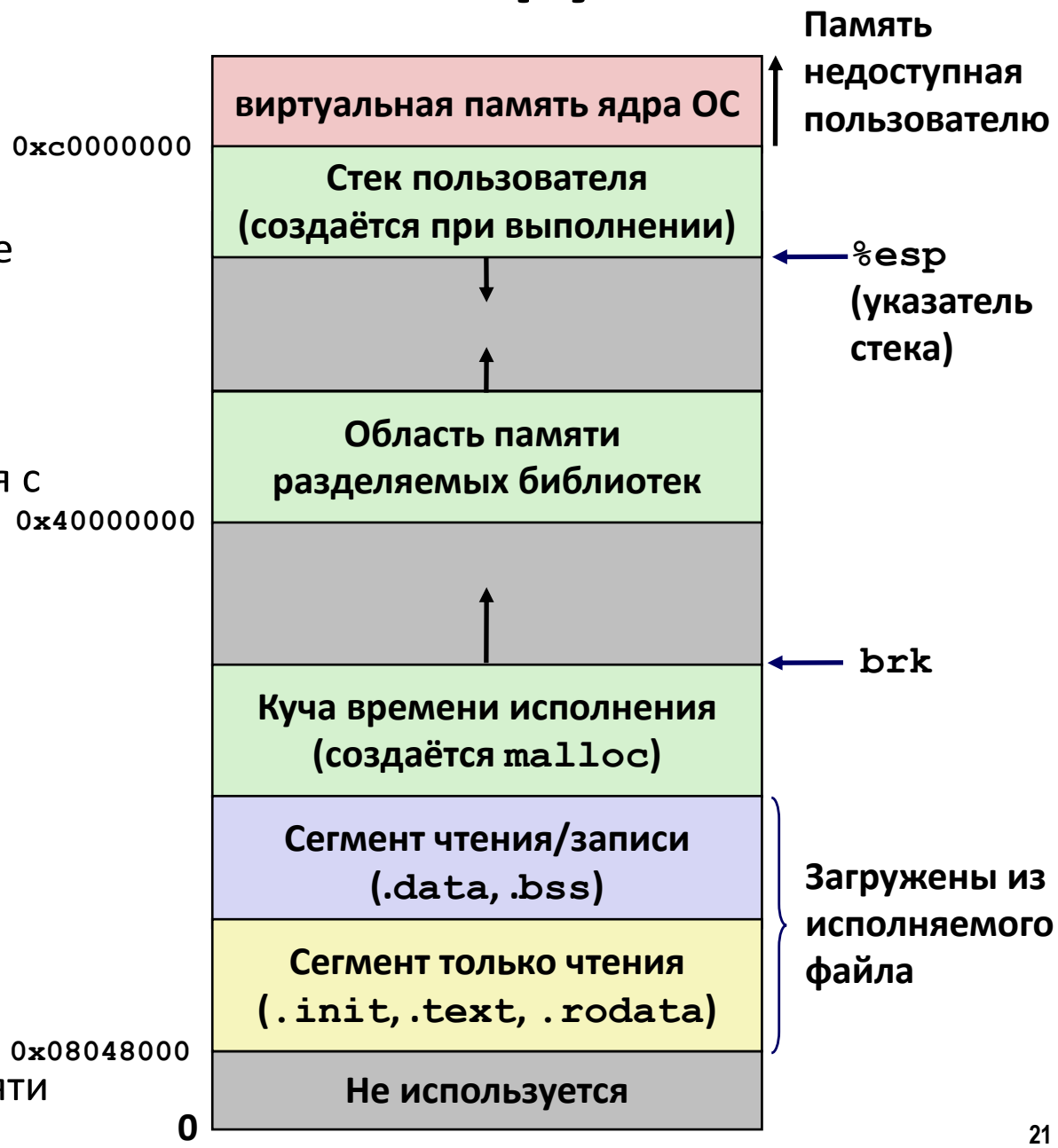
ВП как средство управления памятью - 2

- **Распределение памяти**
  - Любая виртуальная страница может отображаться на любую физическую
  - Виртуальная отображается в различные физические в разное время
- **Разделение кода и данных между процессами**
  - Отображение виртуальных страниц на одну физическую (здесь: ФС 6)



Упрощение связывания и загрузки

- **Связывание**
  - Все программы имеют сходное виртуальное адресное пространство
  - Код, стек и разделяемые библиотеки всегда начинаются с одинаковых адресов
- **Загрузка**
  - **`execve()`** лишь резервирует виртуальные страницы и для разделов .text and .data, т.е. помечает PTE как «не в кеше»
  - Разделы **`.text`** и **`.data`** подгружаются, страница за страницей по требованиям подсистемы виртуальной памяти

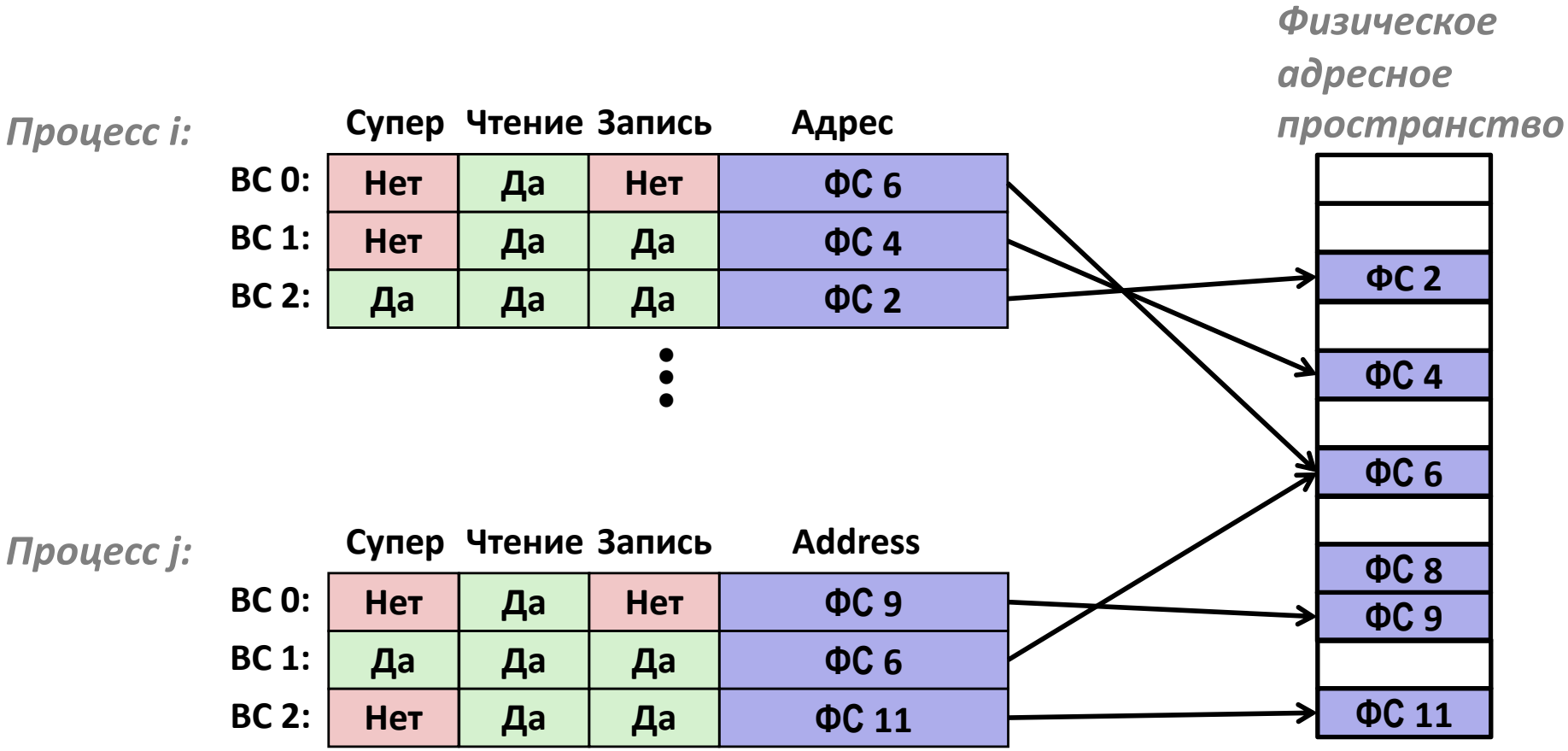


Виртуальная память

- Пространства адресов
- ВП как средство кеширования
- ВП как средство управления памятью
- ВП как средство защиты памяти
- Трансляция адресов
- Пример простой подсистемы памяти
- Пример подсистемы памяти Core i7/Linux
- Отображение памяти

ВП как средство защиты памяти

- **PTE дополняются битами-признаками разрешений**
- **Обработчик сбоев страниц проверяет перед отображением**
  - При нарушении, отправляет процессу SIGSEGV (segmentation fault)



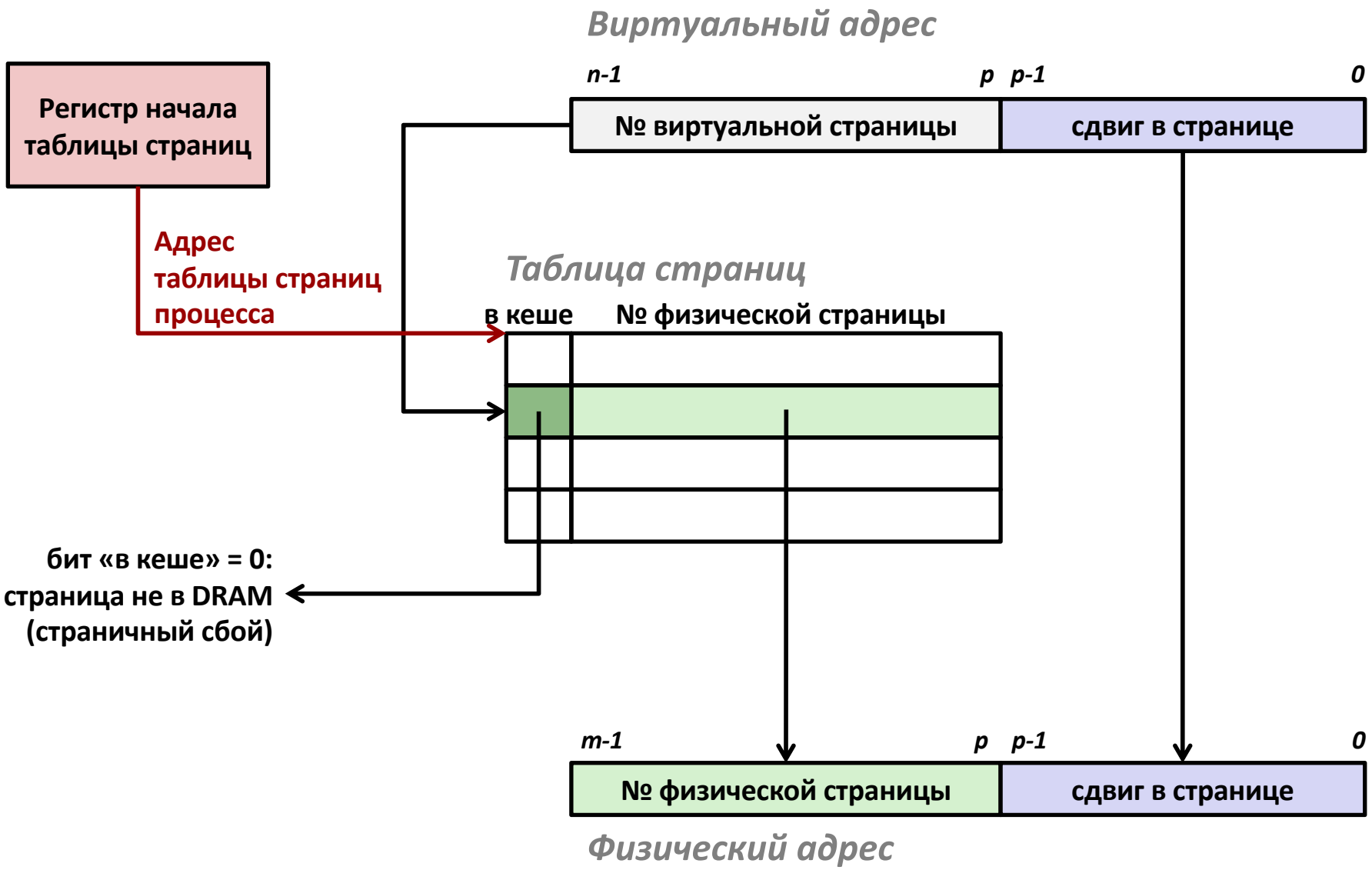
Виртуальная память

- Пространства адресов
- ВП как средство кеширования
- ВП как средство управления памятью
- ВП как средство защиты памяти
- Трансляция адресов
- Пример простой подсистемы памяти
- Пример подсистемы памяти Core i7/Linux
- Отображение памяти

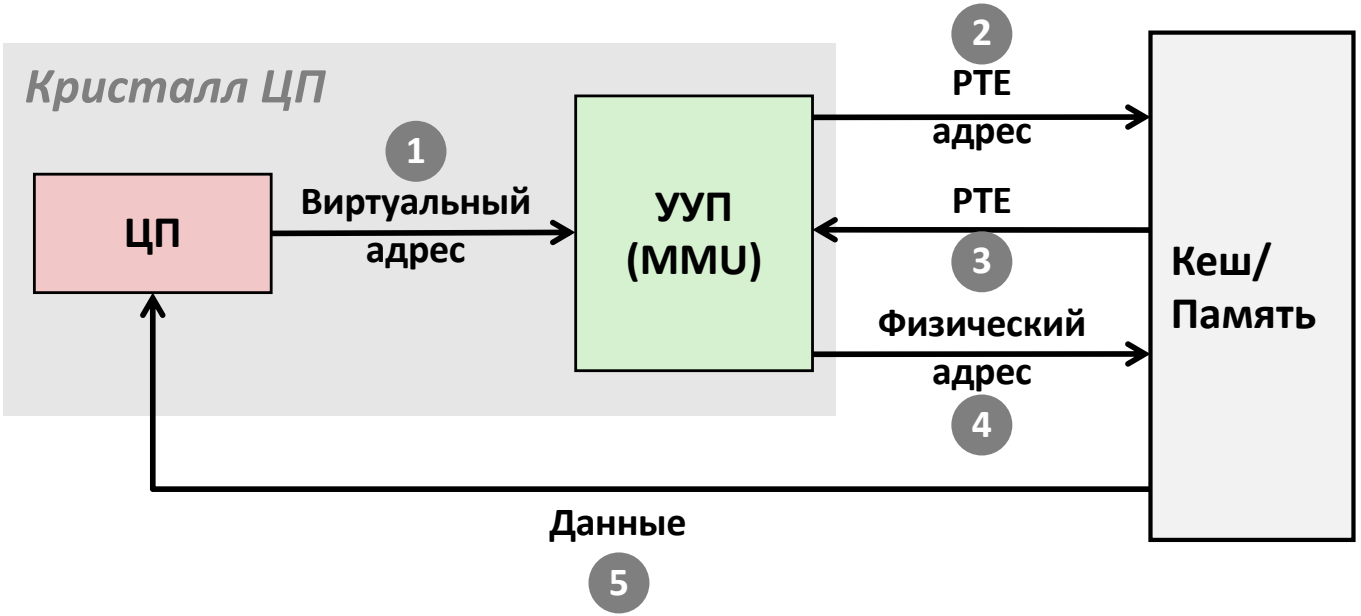
Трансляция адресов ВП

- **Виртуальное адресное пространство**
  - $V = \{0, 1, \ldots, N-1\}$
- **Физическое адресное пространство**
  - $P = \{0, 1, \ldots, M-1\}$
- **Трансляция адресов**
  - $MAP: V \rightarrow P \cup \{\emptyset\}$
  - Для виртуального адреса ***a***:
    - $MAP(a) = a'$ если данные по виртуальному адресу ***a*** находятся по физическому адресу ***a'*** в ***P***
    - $MAP(a) = \emptyset$ если данные по виртуальному адресу ***a*** отсутствуют в физической памяти – или их нет совсем, или они на диске

Трансляция адресов с таблицей страниц



Трансляция адресов: страничное попадание



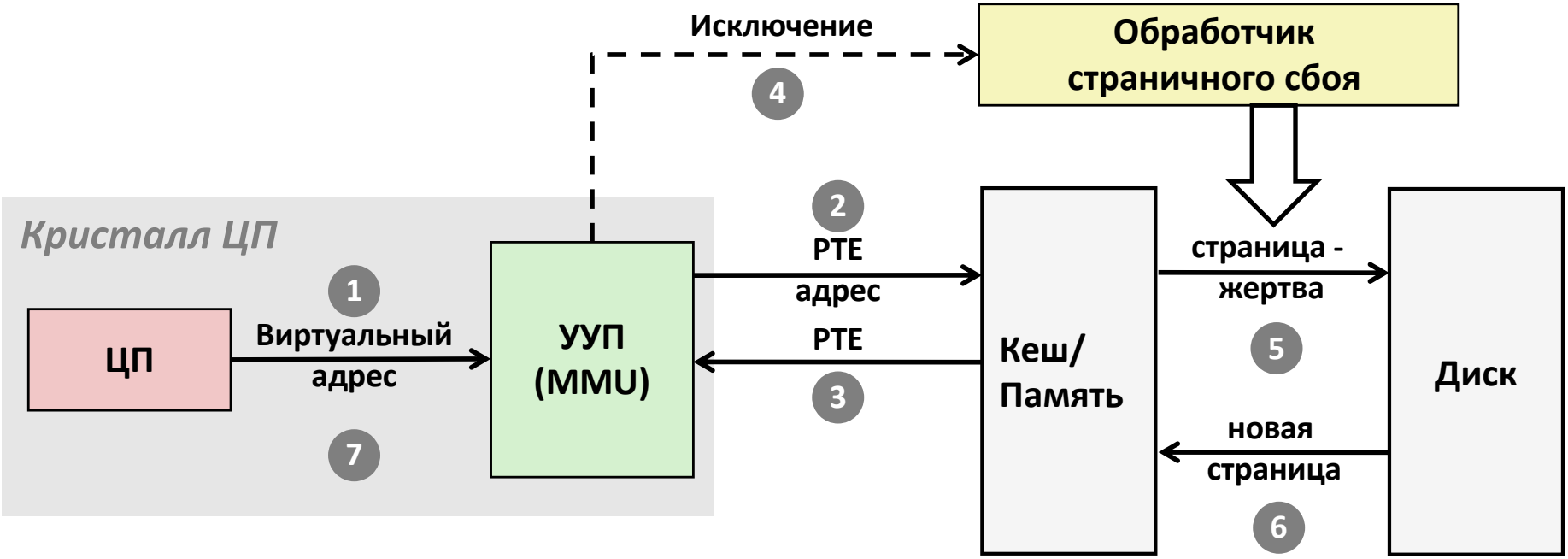
1) Процессор направляет виртуальный адрес УУП

2-3) УУП считывает PTE из таблицы страниц в памяти

4) УУП направляет физический адрес кешу/памяти

5) Кеш/память направляют данные процессору

Трансляция адресов: страничный сбой



1) Процессор направляет виртуальный адрес УУП

2-3) УУП считывает PTE из таблицы страниц в памяти

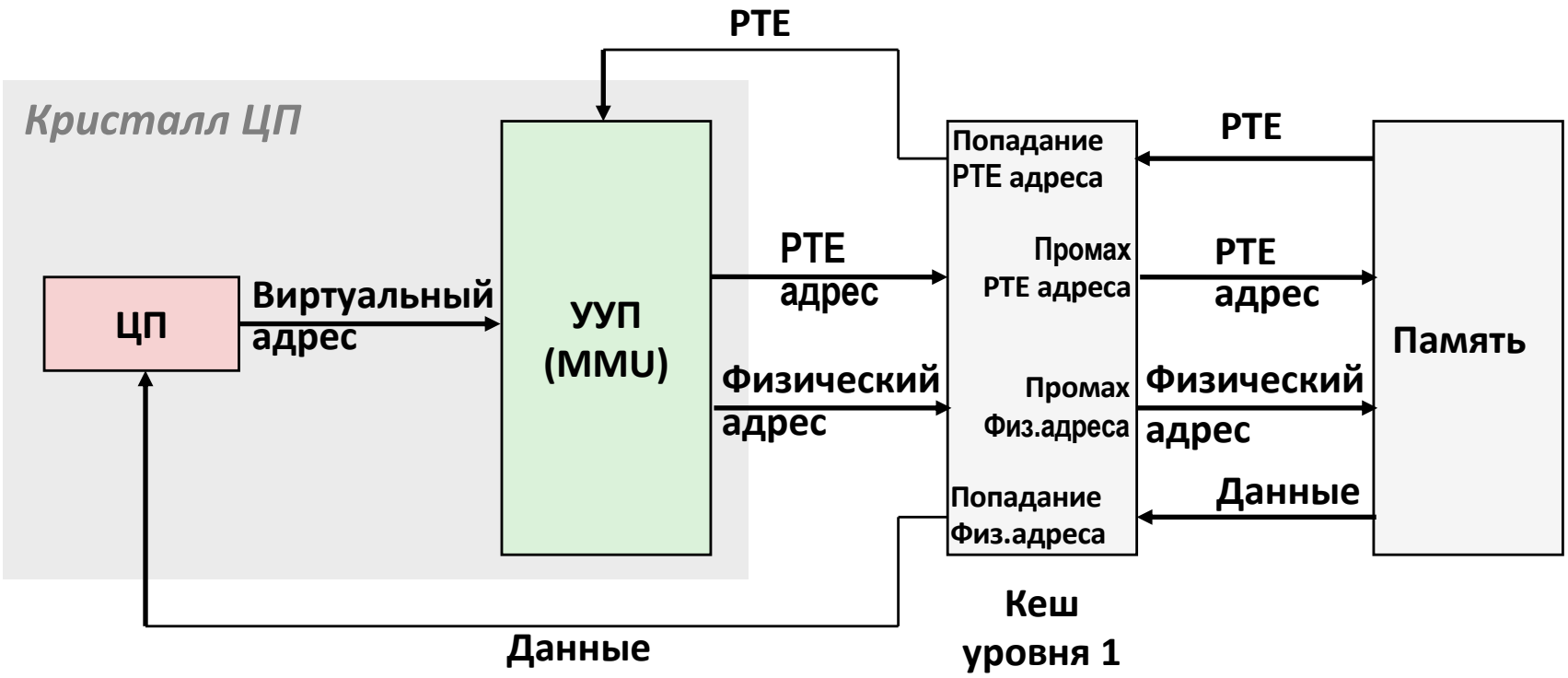
4) Бит «в кеше» - нулевой, УУП запускает исключение «страничный сбой»

5) Обработчик выбирает жертву (и, если менялась, откачивает на диск)

6) Обработчик подкачивает новую страницу и меняет PTE в памяти

7) Обработчик возвращается в процесс для повтора сбойной команды

Объединение ВП и кеша

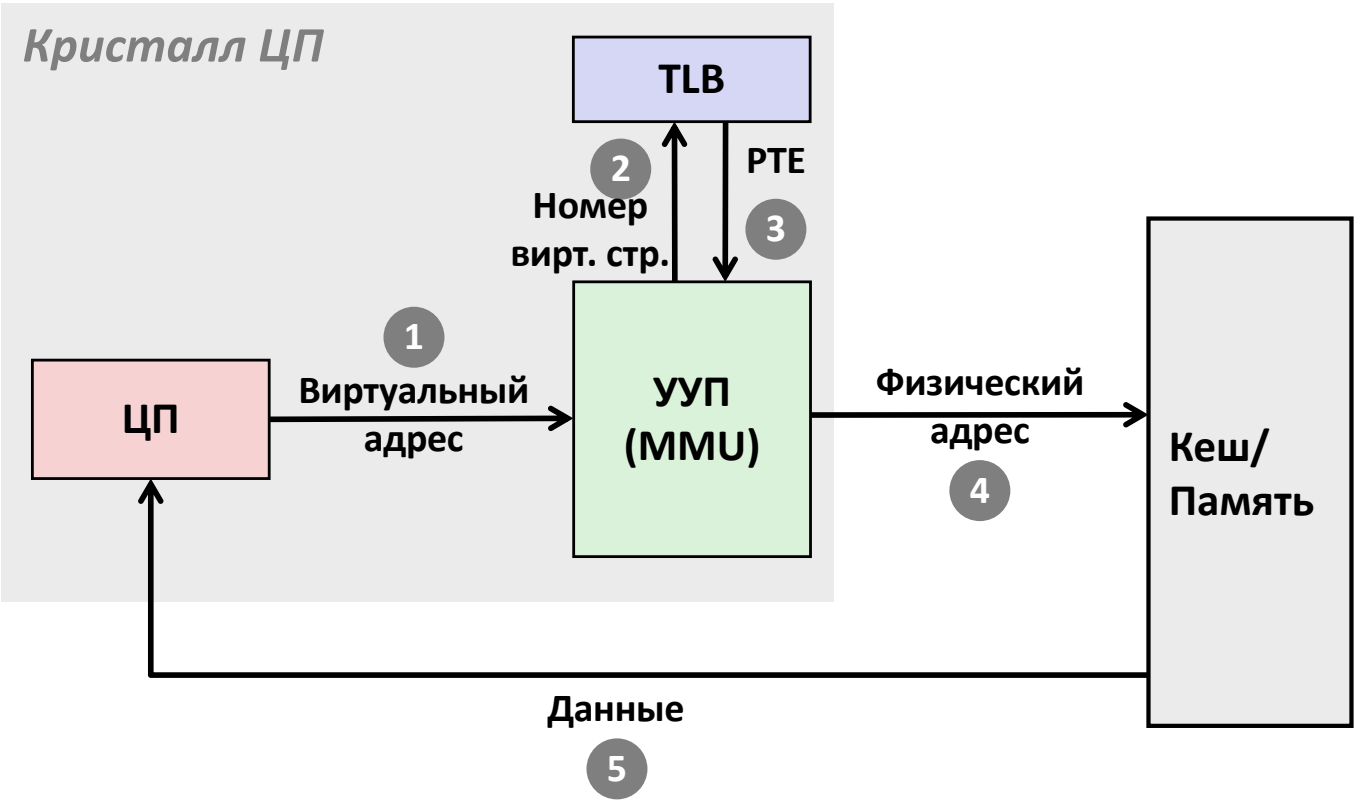


PTE: запись в таблице страниц

Ускорение трансляции с помощью TLB

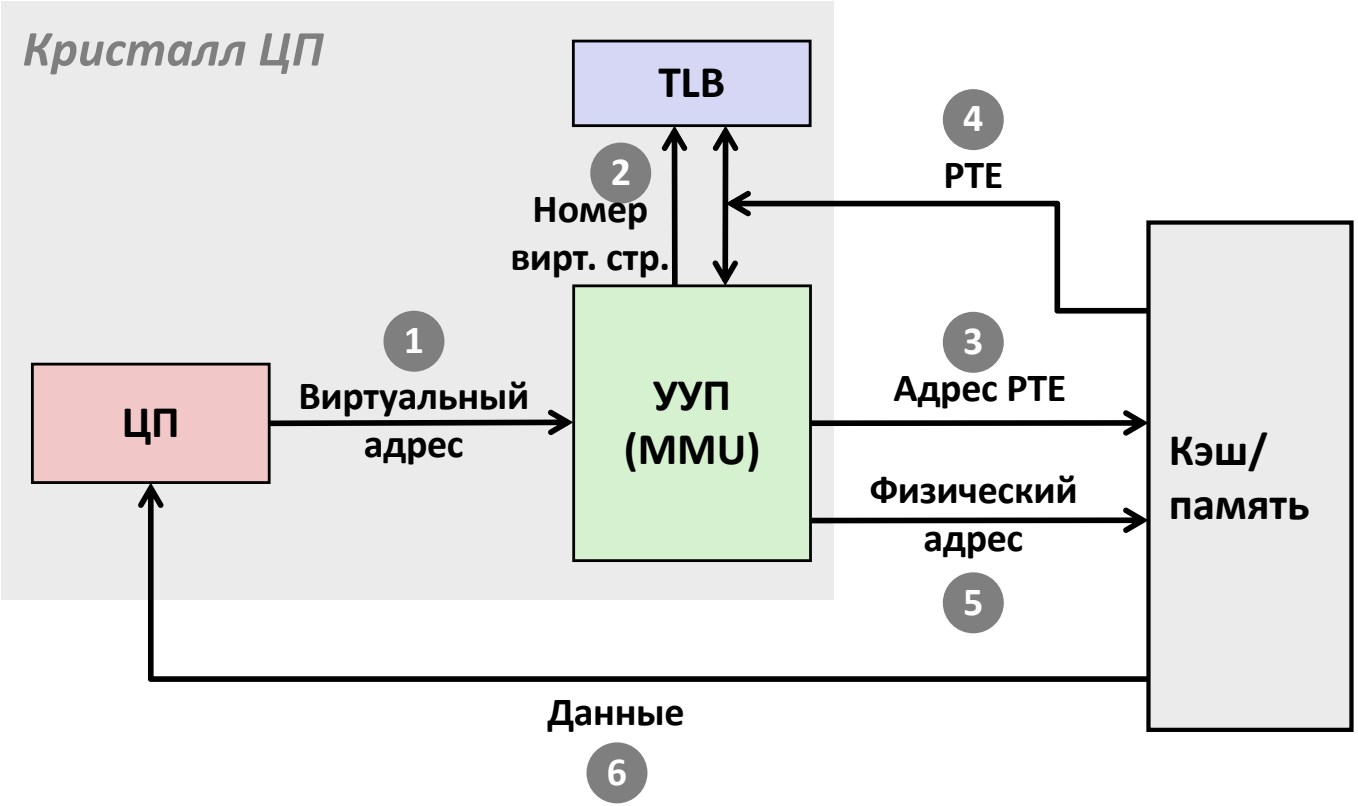
- **Записи в таблице страниц (PTE) кешируются в уровне 1 как любое другое слово памяти**
  - PTE могут выталкиваться обращениями к другим данным
  - попадание PTE всё ещё требует небольшой задержки уровня 1
- **Решение: *Translation Lookaside Buffer* (TLB)**
  - небольшой аппаратный кеш в УУП (MMU)
  - отображает номера виртуальных страниц в номера физических
  - содержит целиком записи для небольшого количества страниц

Попадание TLB



Попадание TLB - нет дополнительного обращения в память

Промах TLB

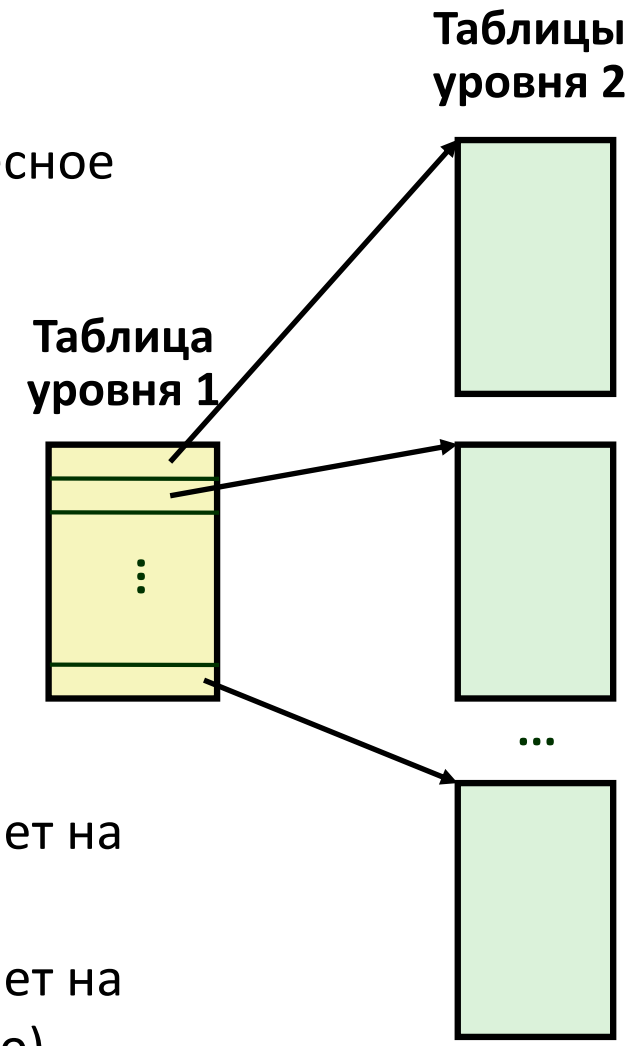


Промах TLB влечёт дополнительное обращение в память за PTE

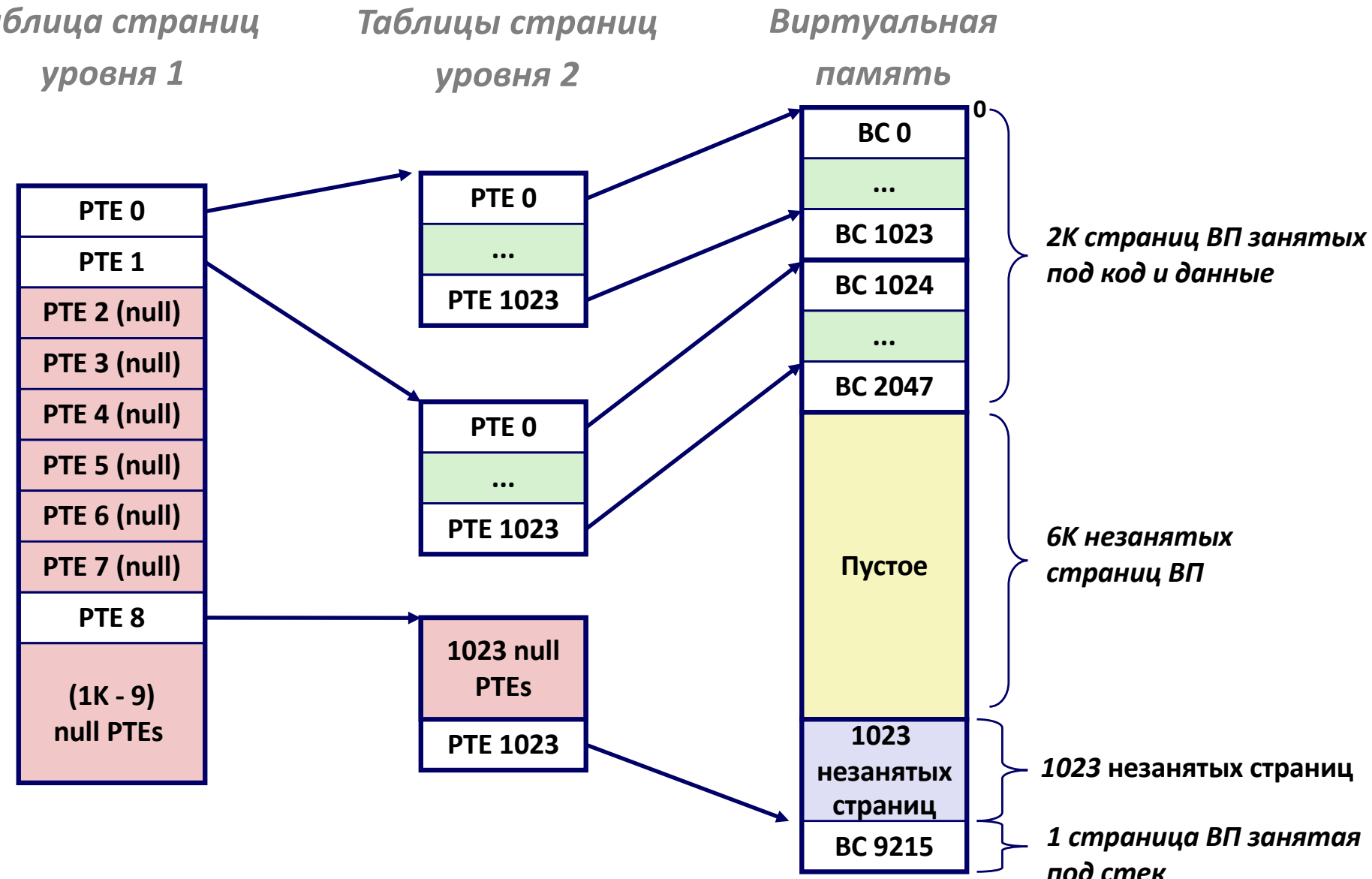
К счастью, промахи TLB редки. Почему?

Многоуровневые таблицы страниц

- **Предположим:**
  - 4KB (2^{12}) размер страницы, 48-битное адресное пространство, 8-байтные PTE
- **Проблема:**
  - Необходима таблица размером 512 ГБ!
    - $2^{48} * 2^{-12} * 2^{3} = 2^{39}$ байт
- **Типовое решение:**
  - Многоуровневые таблицы страниц
  - Пример: 2-уровневая таблица страниц
    - Таблица уровня 1: каждый PTE указывает на таблицу страниц (всегда в памяти)
    - Таблица уровня 2: каждый PTE указывает на страницу (подкачанную или откачанную)



Двухуровневая иерархия таблиц страниц



32-битные адреса, 4KB страницы, 4- байтные PTEs

Сводка понятий

- **С точки зрения программиста, виртуальная память…**
  - даёт каждому процессу собственное линейное адресное пространство
  - не может быть повреждена другим процессом

- **С точки зрения системы, виртуальная память…**
  - эффективно использует DRAM для кеширования виртуальных страниц
    - только благодаря локальности
  - упрощает управление памятью и программирование
  - упрощает защиту памяти обеспечивая удобную контрольную точку проверки разрешений

Термины и обозначения

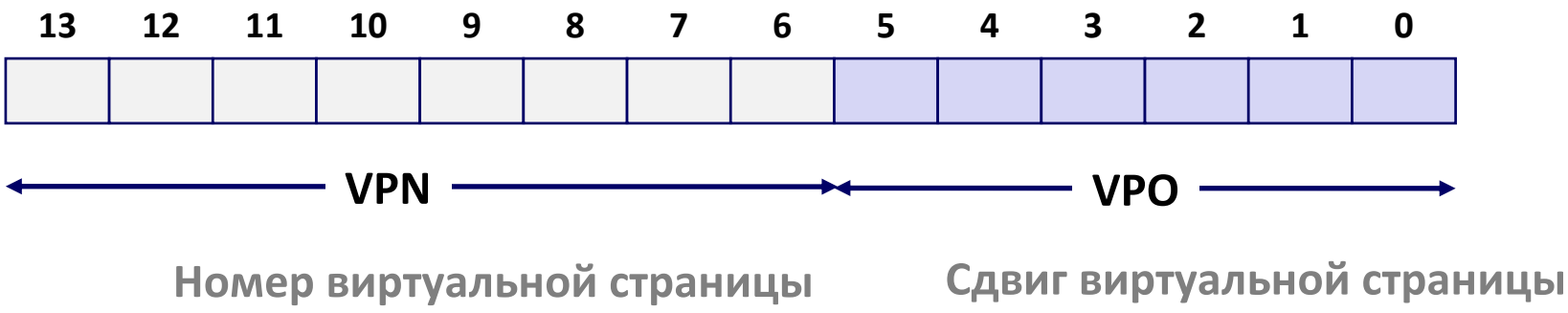
- **Основные параметры**
  - **$N = 2^n$** : количество адресов в виртуальном адресном пространстве
  - **$M = 2^m$** : количество адресов в физическом адресном пространстве
  - **$P = 2^p$** : размер страницы (в байтах)
- **Части виртуального адреса (VA)**
  - **TLBI**: индекс TLB
  - **TLBT**: метка TLB
  - **VPO**: сдвиг (от начала) виртуальной страницы
  - **VPN**: номер виртуальной страницы
- **Части физического адреса (PA)**
  - **PPO**: сдвиг (от начала) физической страницы (тот же, что VPO)
  - **PPN:** номер виртуальной страницы
  - **CO**: сдвиг байта (от начала) линии кеша
  - **CI:** индекс кеша
  - **CT**: метка кеша

Виртуальная память

- Пространства адресов
- ВП как средство кеширования
- ВП как средство управления памятью
- ВП как средство защиты памяти
- Трансляция адресов
- Пример простой подсистемы памяти
- Пример подсистемы памяти Core i7/Linux
- Отображение памяти

Пример простой подсистемы памяти

- **Адресация**
  - 14-битный виртуальный адрес
  - 12-битный физический адрес
  - Размер страницы = 64 байта



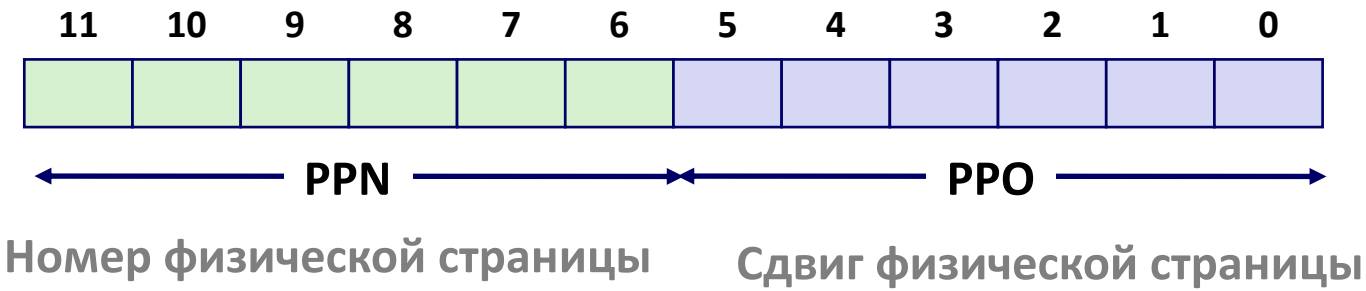


Таблица страниц простой подсистемы

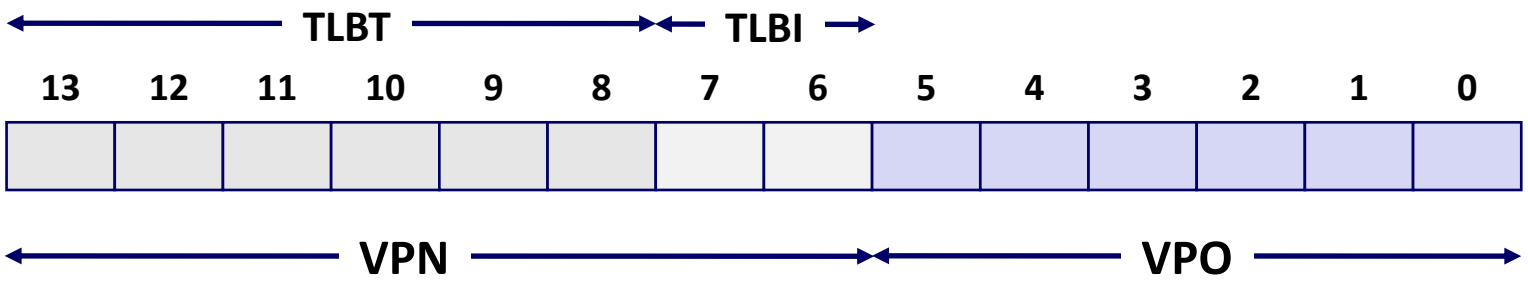
Показаны только первые 16 записей (из 256)

VPN	PPN	в кеше
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

VPN	PPN	в кеше
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1

TLB простой подсистемы

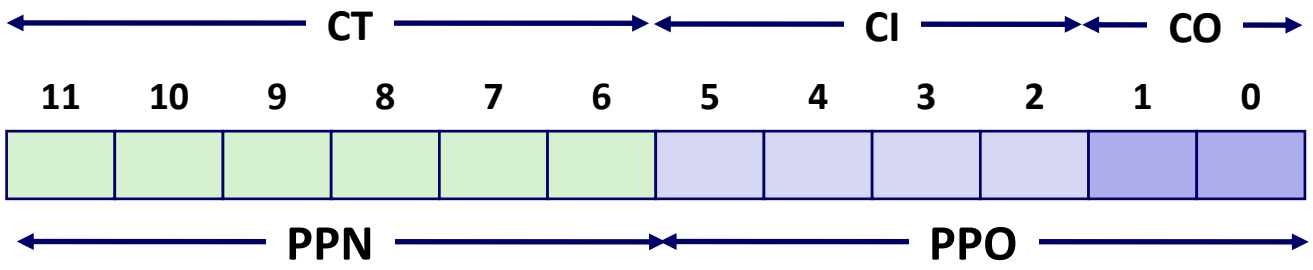
- 16 записей
- 4-вариантный



набор	метка	PPN	в кеше	метка	PPN	в кеше	метка	PPN	в кеше	метка	PPN	в кеше
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Кеш простой подсистемы

- 16 линий, размер блока 4 байта (B0, B1, B2, B3)
- Физическая адресация
- Прямое отображение (без вариантов)



индекс	*метка*	*в кеше*	*B0*	*B1*	*B2*	*B3*
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

индекс	*метка*	*в кеше*	*B0*	*B1*	*B2*	*B3*
8	24	1	3A	00	51	89
9	2D	0	–	–	–	–
A	2D	1	93	15	DA	3B
B	0B	0	–	–	–	–
C	12	0	–	–	–	–
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	–	–	–	–

Пример трансляции адресов №1

Виртуальный адрес: 0x03D4

TLBT						TLBI							
13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	1	0	1	0	0
VPN								VPO					

VPN 0x0F TLBI 0x3 TLBT 0x03 Попадание TLB? ДА Page Fault? НЕТ PPN: 0x0D

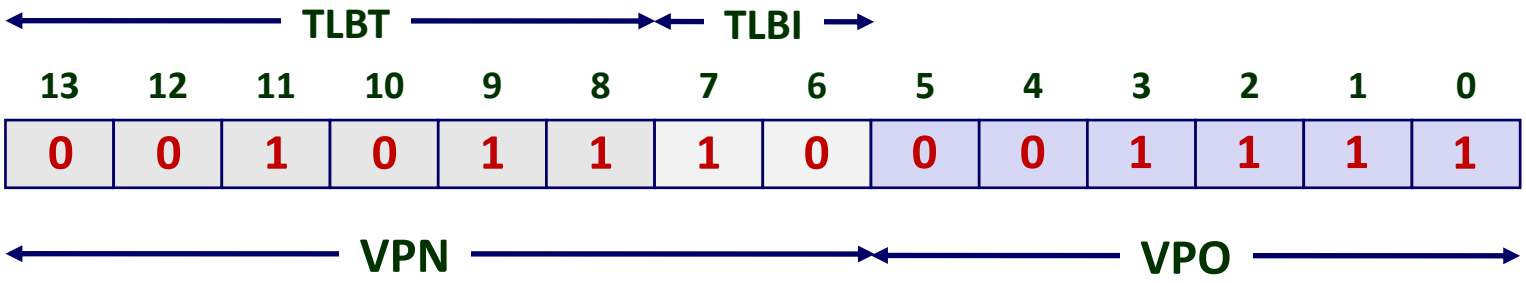
Физический адрес

CT						CI				CO	
11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	0	0
PPN						PPO					

CO 0 CI 0x5 CT 0x0D Попадание? ДА Байт: 0x36

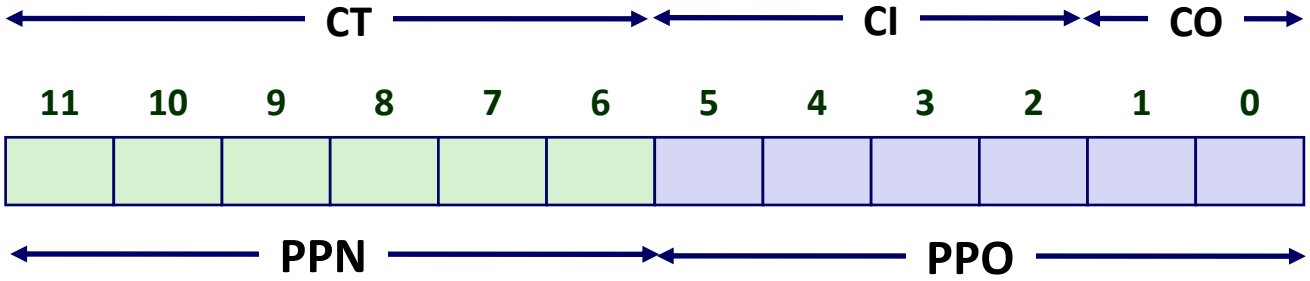
Пример трансляции адресов №2

Виртуальный адрес: `0x0B8F`



VPN 0x2E TLBI 2 TLBT 0x0B Попадание TLB? НЕТ Page Fault? ДА PPN: ???

Физический адрес



CO ___ CI___ CT ____ Попадание? ___ Байт: ____

Пример трансляции адресов №3

Виртуальный адрес: 0x0020

TLBT						TLBI		VPO					
13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0

VPN

VPN 0x00 TLBI 0 TLBT 0x00 Попадание TLB? НЕТ Page Fault? НЕТ PPN: 0x28

Физический адрес

CT						CI				CO	
11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	1	0	0	0	0	0

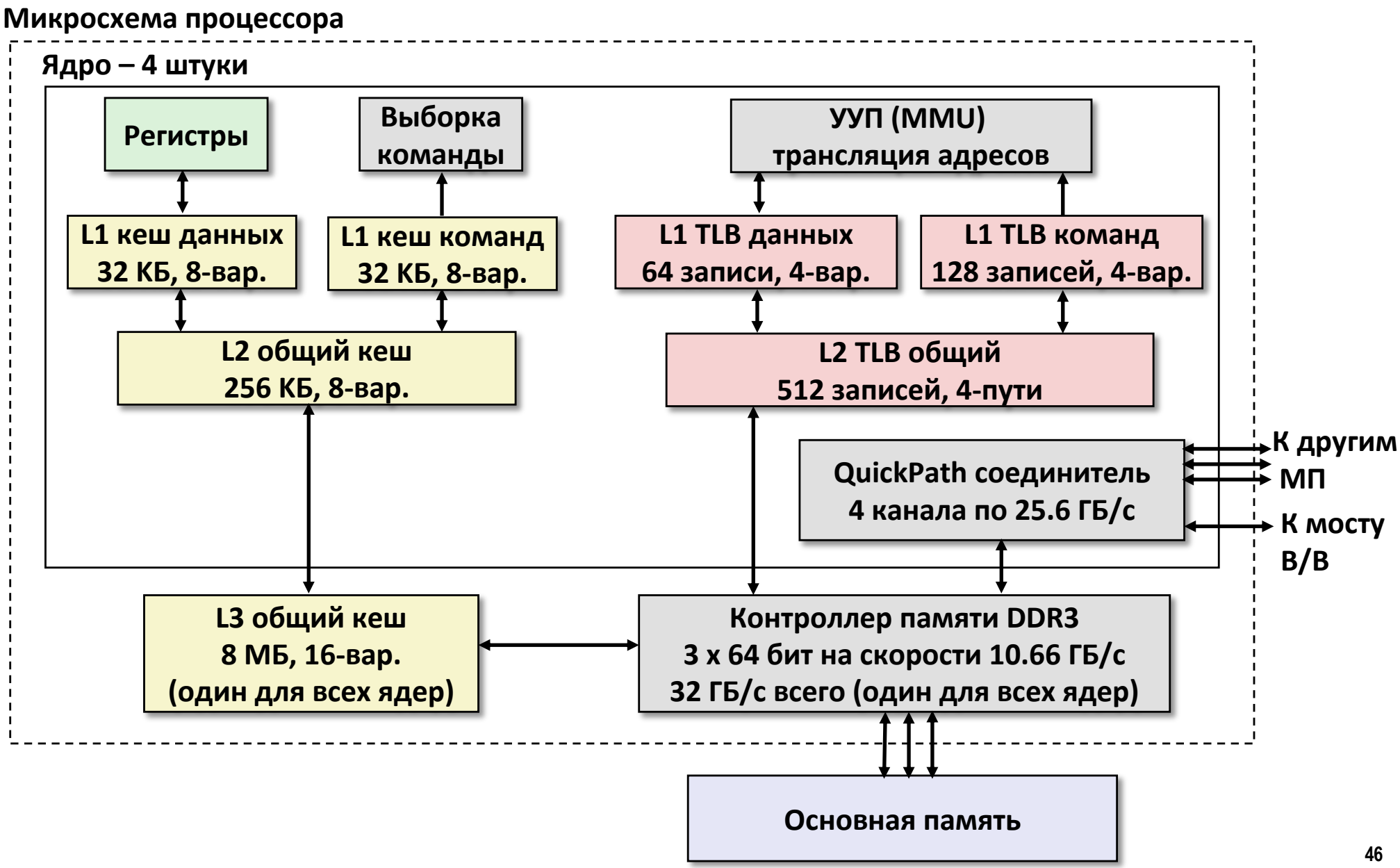
PPN PPO

CO 0 CI 0x8 CT 0x28 Попадание? НЕТ Байт: в памяти

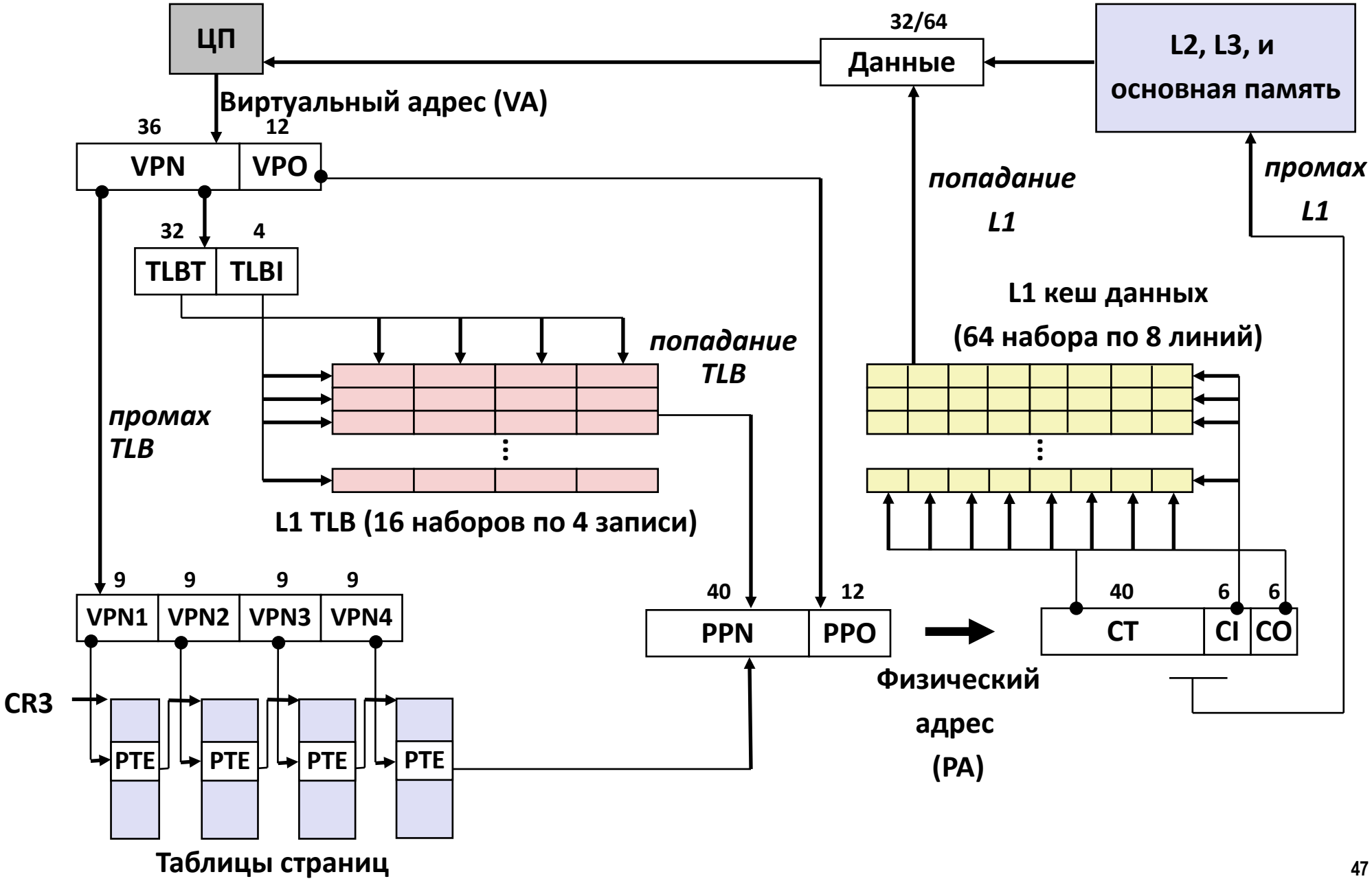
Виртуальная память

- Пространства адресов
- ВП как средство кеширования
- ВП как средство управления памятью
- ВП как средство защиты памяти
- Трансляция адресов
- Пример простой подсистемы памяти
- Пример подсистемы памяти Core i7/Linux
- Отображение памяти

Подсистема памяти Intel Core i7



Трансляция адресов Core i7 полностью



Табличные записи Core i7 уровней 1-3

63	62 – 52	51 – 12	11 – 9	8	7	6	5	4	3	2	1	0
XD	Свободны	Основа физического адреса таблицы страниц	Свободны	G	PS		A	CD	WT	U/S	R/W	P=1

Доступно для ОС (место таблицы страниц на диске)	P=0

Каждая запись обозначает дочернюю таблицу страниц размером 4КБ

P: присутствие в физической памяти дочерней таблицы страниц

R/W: разрешение на чтение-запись или только чтение для всех достижимых станиц

U/S: разрешение доступа пользователю или ядру для всех достижимых станицам

WT: политика записи в кеш дочерней таблицы страниц (отложенная или сквозная)

CD: разрешение или запрет кеширования дочерней таблицы страниц

A: признак обращения (устанавливается УУП при чтении и записи, сбрасывается ПО).

PS: размер страницы 4 КБ или 4 МБ (только для записей уровня 1).

G: признак глобальности таблицы страниц (не откачивается из TLB при переключении задач)

Основа физического адреса таблицы страниц: 40 старших бит физического адреса таблицы страниц (принудительное выравнивание на границу 4КБ)

Табличные записи Core i7 уровня 4

63	62 – 52	51 – 12	11 – 9	8	7	6	5	4	3	2	1	0
XD	Свободны	Основа физического адреса страницы	Свободны	G		D	A	CD	WT	U/S	R/W	P=1

Доступно для ОС (место таблицы страниц на диске)	P=0

Каждая запись обозначает дочернюю страницу размером 4КБ

P: присутствие в физической памяти дочерней страницы

R/W: разрешение на чтение-запись или только чтение для дочерней страницы

U/S: разрешение доступа пользователю или ядру для дочерней страницы

WT: политика записи в кеш дочерней страницы (отложенная или сквозная)

CD: разрешение или запрет кеширования дочерней страницы

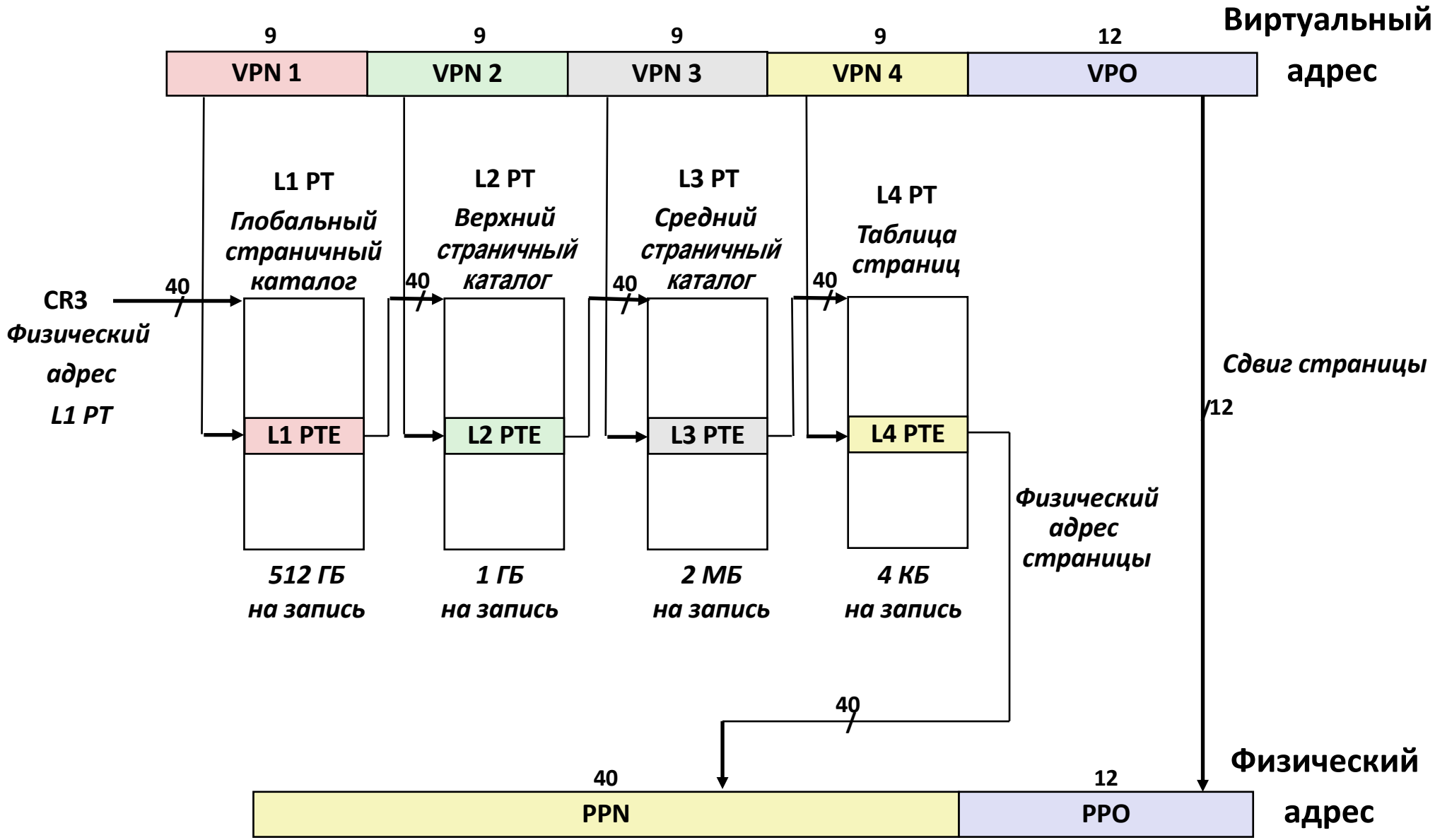
A: признак обращения (устанавливается УУП при чтении и записи, сбрасывается ПО).

D: признак записи (устанавливается УУП при записи, сбрасывается ПО)

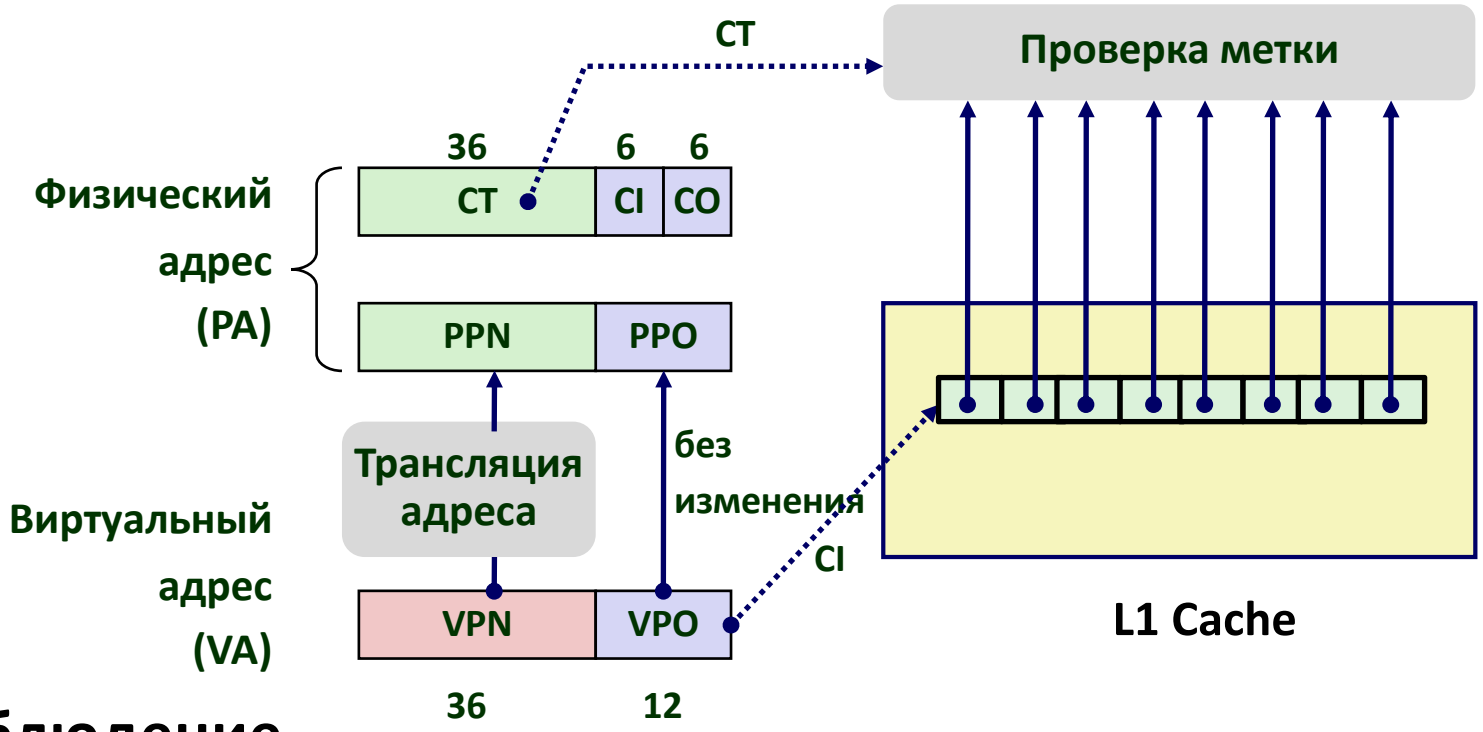
G: признак глобальности страницы (не откачивается из TLB при переключении задач)

Основа физического адреса страницы: 40 старших бит физического адреса страницы (принудительное выравнивание на границу 4КБ)

Трансляция страничных таблиц Core i7

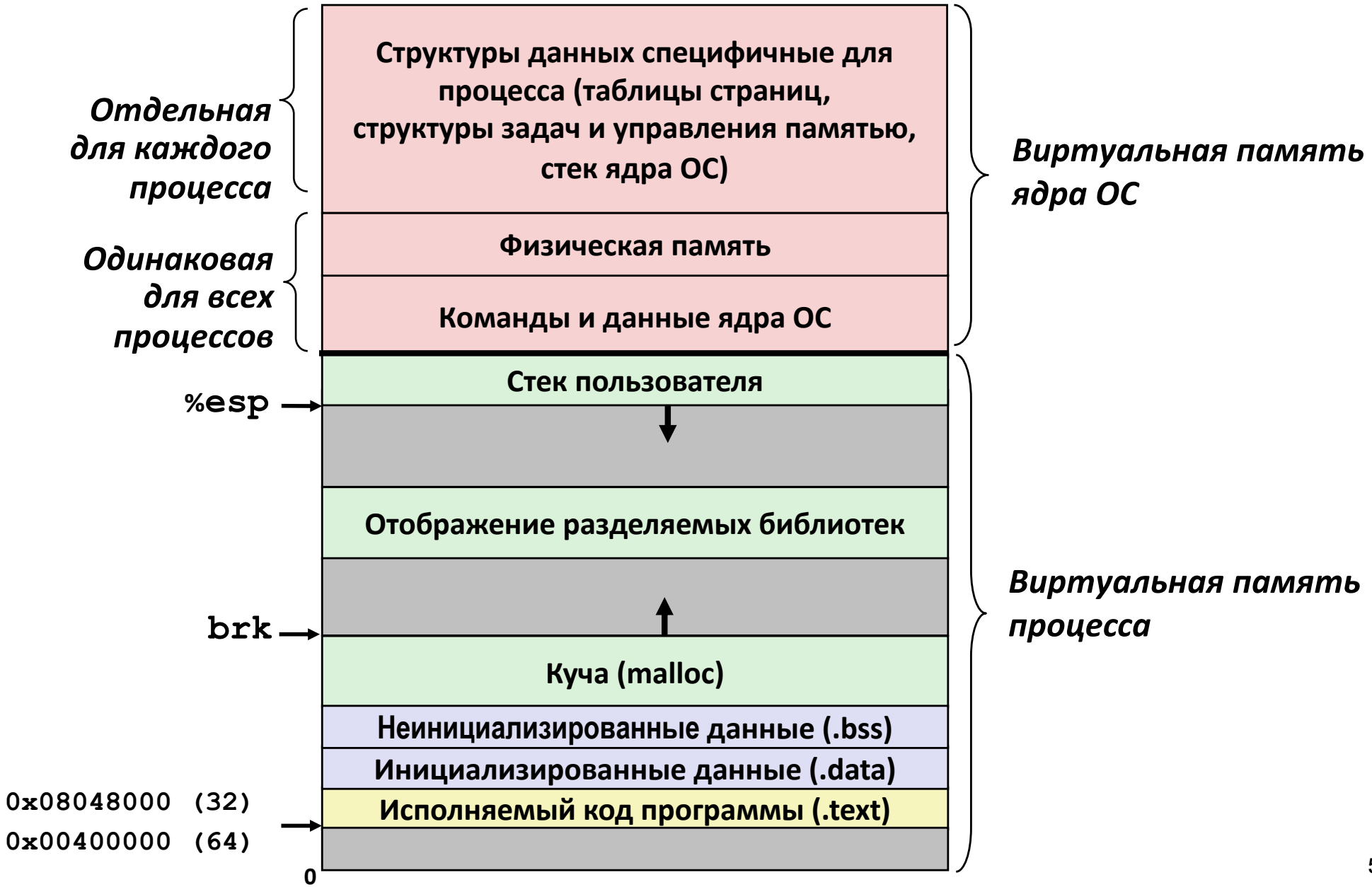


Ускорение доступа к L1

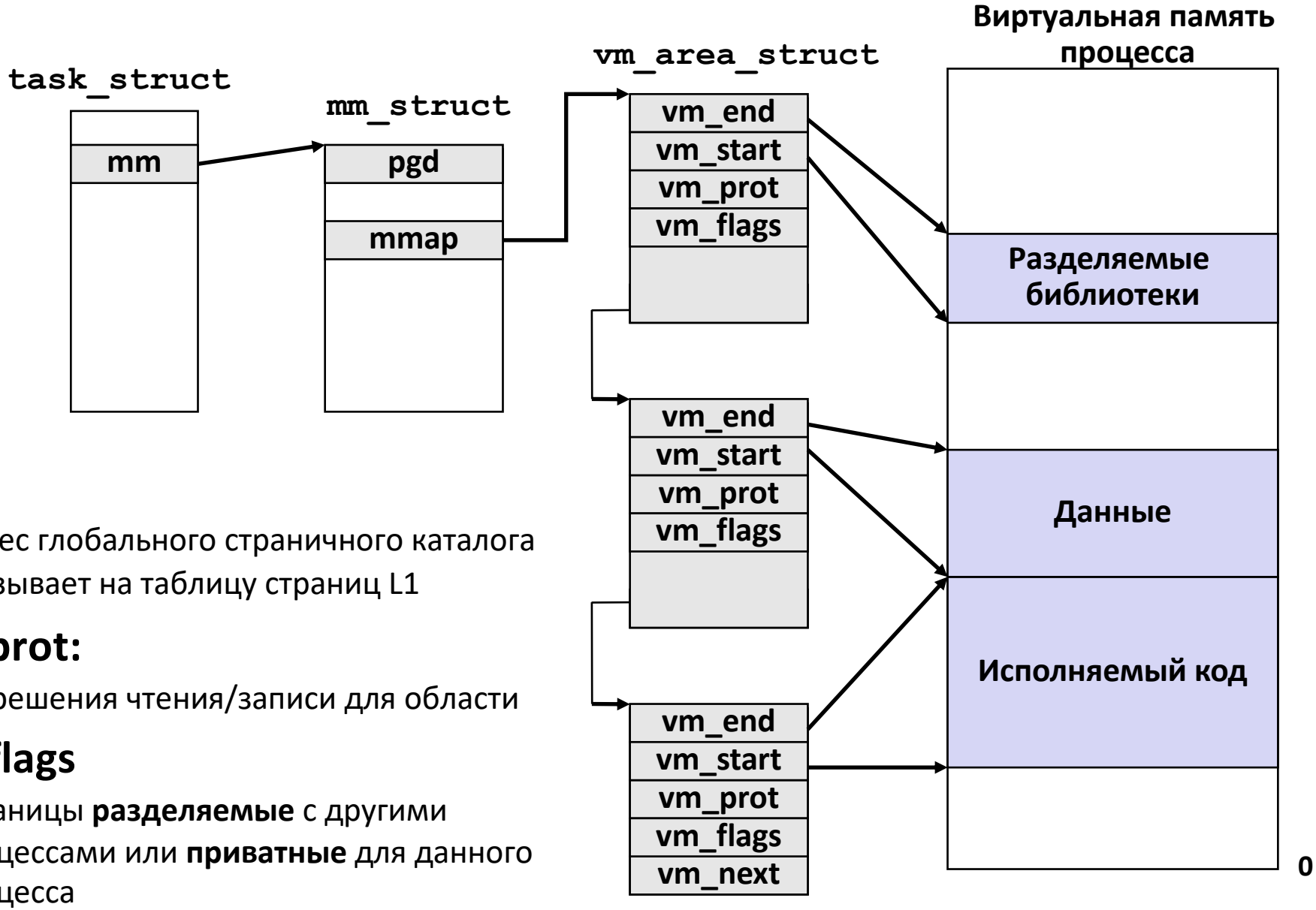


- **Наблюдение**
  - Биты CI идентичны в физическом и виртуальном адресах
  - Индексирование в кеше возможно при трансляции адресов
  - Обычно попадание в TLB, и биты PPN (CT) доступны немедленно
  - “Виртуально индексирован, физически размечен”
  - Чтобы обеспечить, размеры кеша аккуратно подобраны

Виртуальная память в процессах Linux

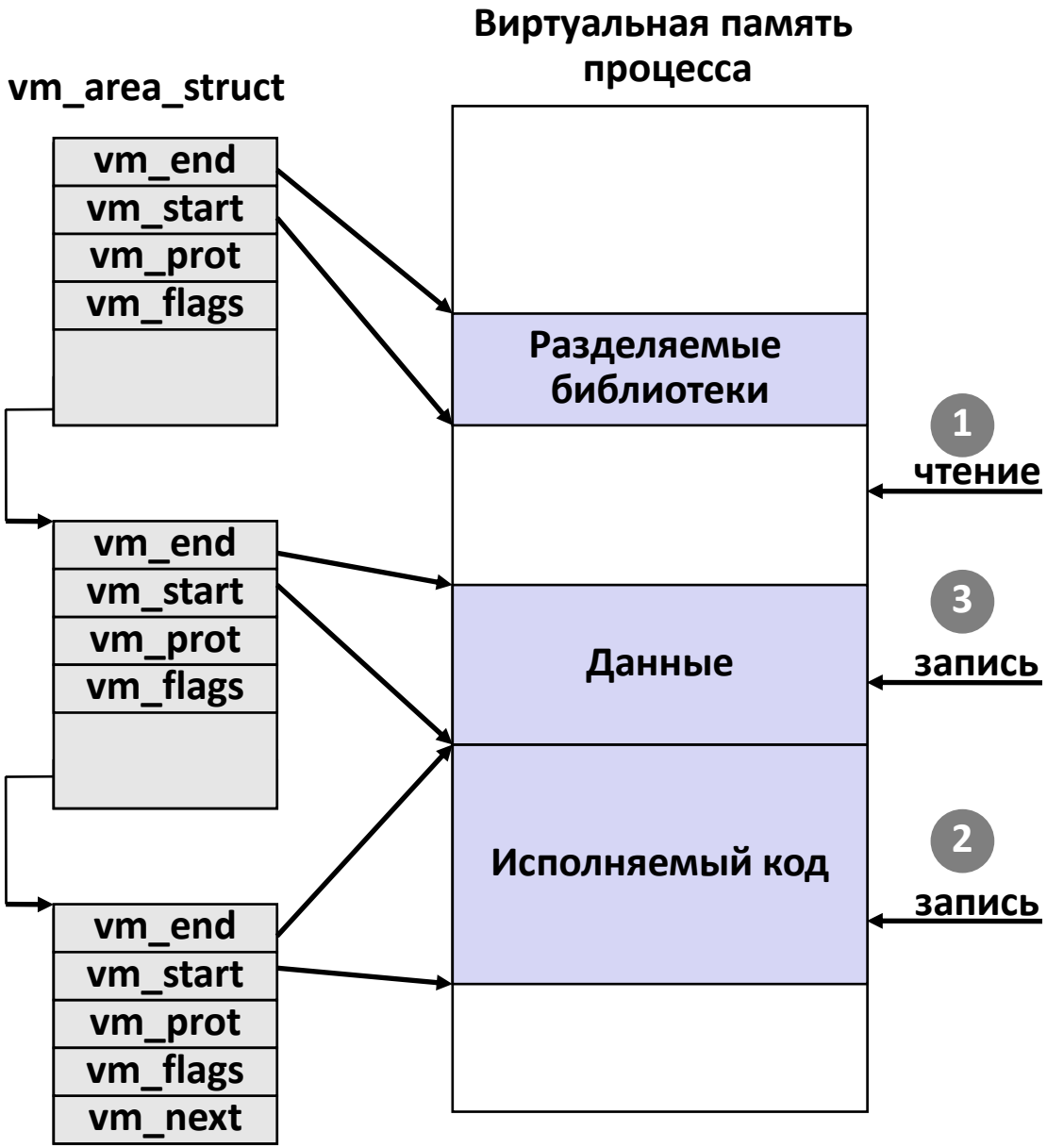


В Linux ВП – коллекция “областей”



- **pgd:**
  - Адрес глобального страничного каталога
  - Указывает на таблицу страниц L1
- **vm_prot:**
  - Разрешения чтения/записи для области
- **vm_flags**
  - Страницы **разделяемые** с другими процессами или **приватные** для данного процесса

Обработка страничного сбоя в Linux



Segmentation fault:
обращение к несуществующей странице

Обычный страничный сбой

Защитное исключение:
например, попытка записи в область только для чтения (Linux диагностирует как Segmentation fault)

Виртуальная память

- Пространства адресов
- ВП как средство кеширования
- ВП как средство управления памятью
- ВП как средство защиты памяти
- Трансляция адресов
- Пример простой подсистемы памяти
- Пример подсистемы памяти Core i7/Linux
- Отображение памяти

Отображение памяти

- **Области ВП инициализируются соотнесением их с объектами на диске**
  - Процесс называется ***отображение памяти (memory mapping)***

- **Области могут быть инициализированы содержимым…**
  - ***…обычных файлов на диске Regular*** (например файла исполняемого объекта)
    - Байты, инициализирующие страницы, поступают из разделов файла
  - ***…анонимных файлов*** (то есть из ниоткуда)
    - первый сбой занимает обнулённую страницу (***demand-zero page***)
    - после записи в страницу (***dirtied***), она обрабатывается как и прочие

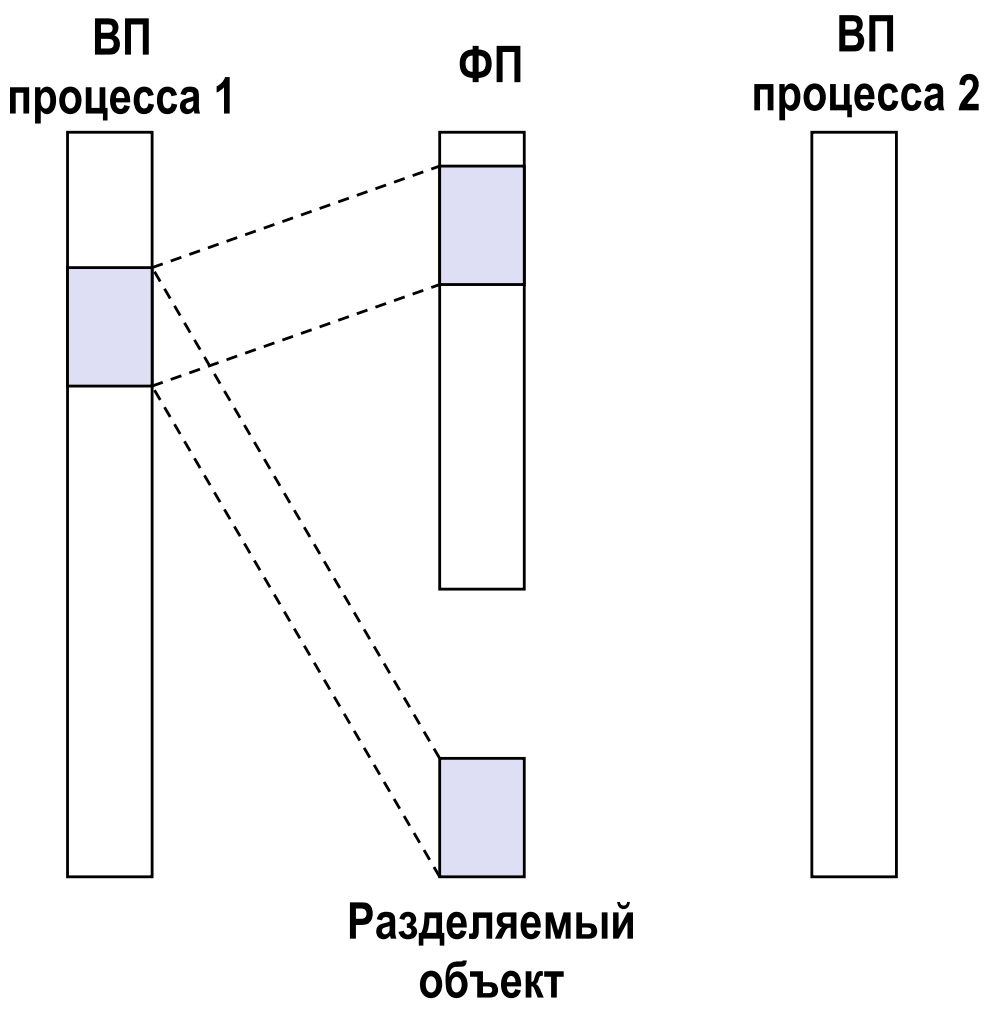
- **Страницами после записи обмениваются память и специальный *файл подкачки (swap file)***

Подкачка по требованию

- ***Ключевая идея:*** **виртуальные страницы не копируются в физическую память пока к ним не обращаются!**
  - Называется ***подкачка по требованию (demand paging)***

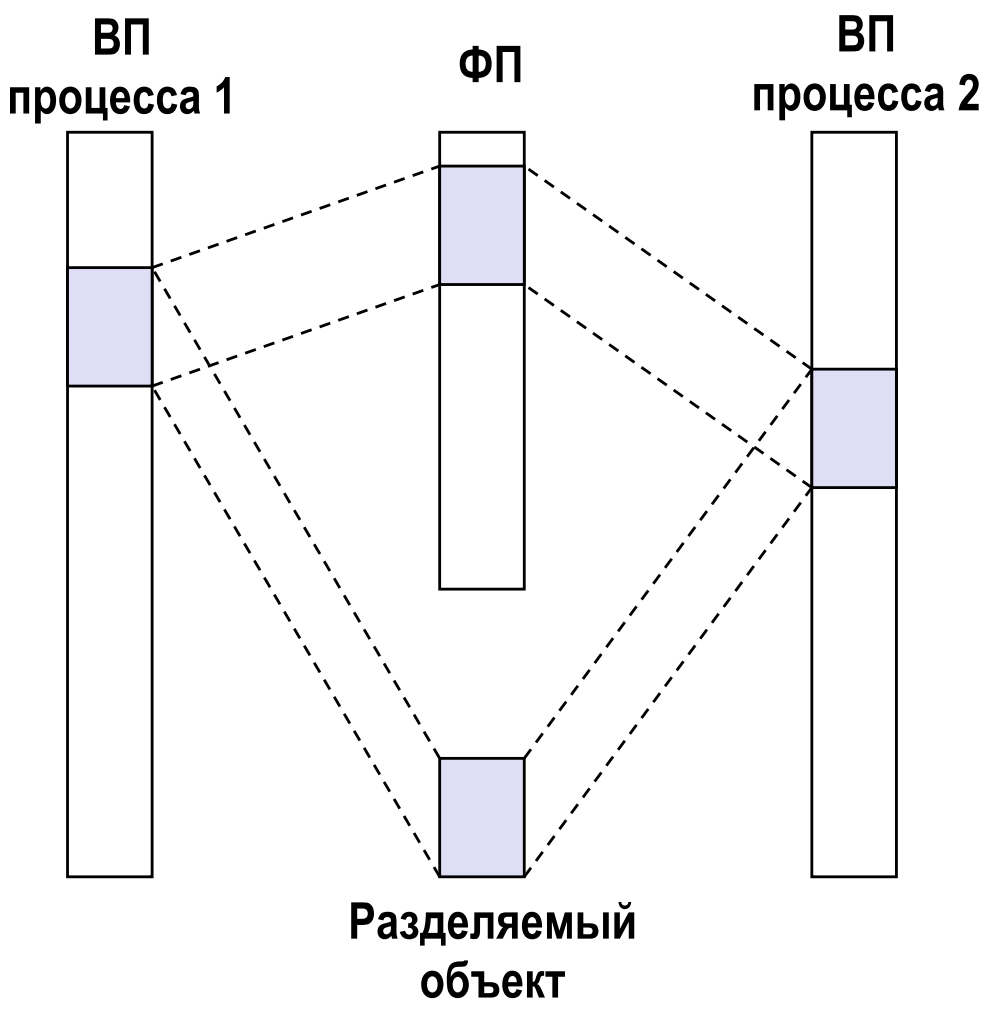
- **Критически важно для временной и пространственной эффективности**

Вновь о разделении: разделяемые объекты



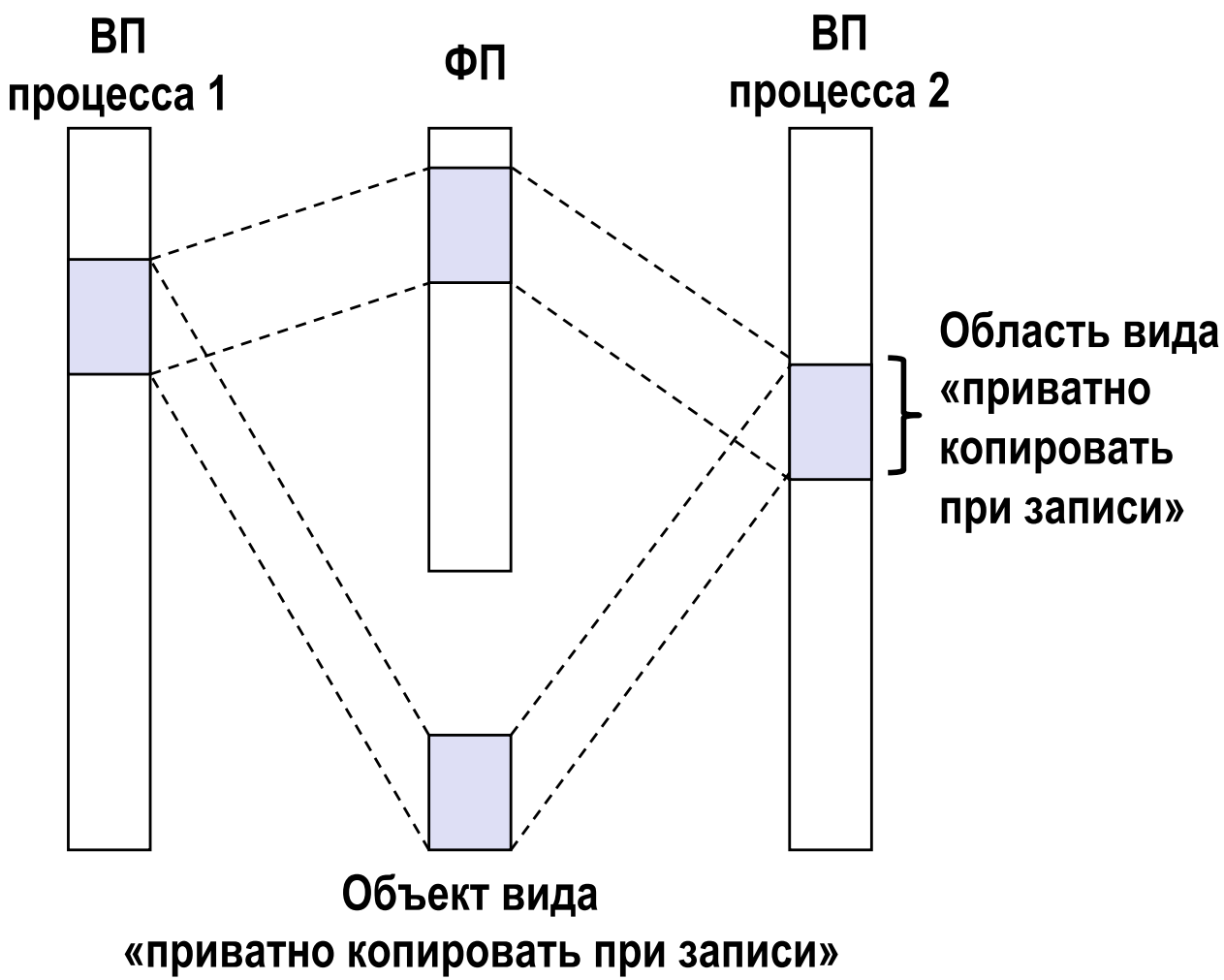
- Процесс 1 отображает разделяемый объект

Вновь о разделении: разделяемые объекты



- Процесс 2 отображает разделяемый объект
- Заметьте, что виртуальные адреса могут быть разными!

Вновь о разделении: приватная копия объекта при записи



- Два процесса отображают объект вида *приватно копировать при записи (COW)*
- Область помечена соответственно
- Страничные записи в приватных областях помечены «только чтение»

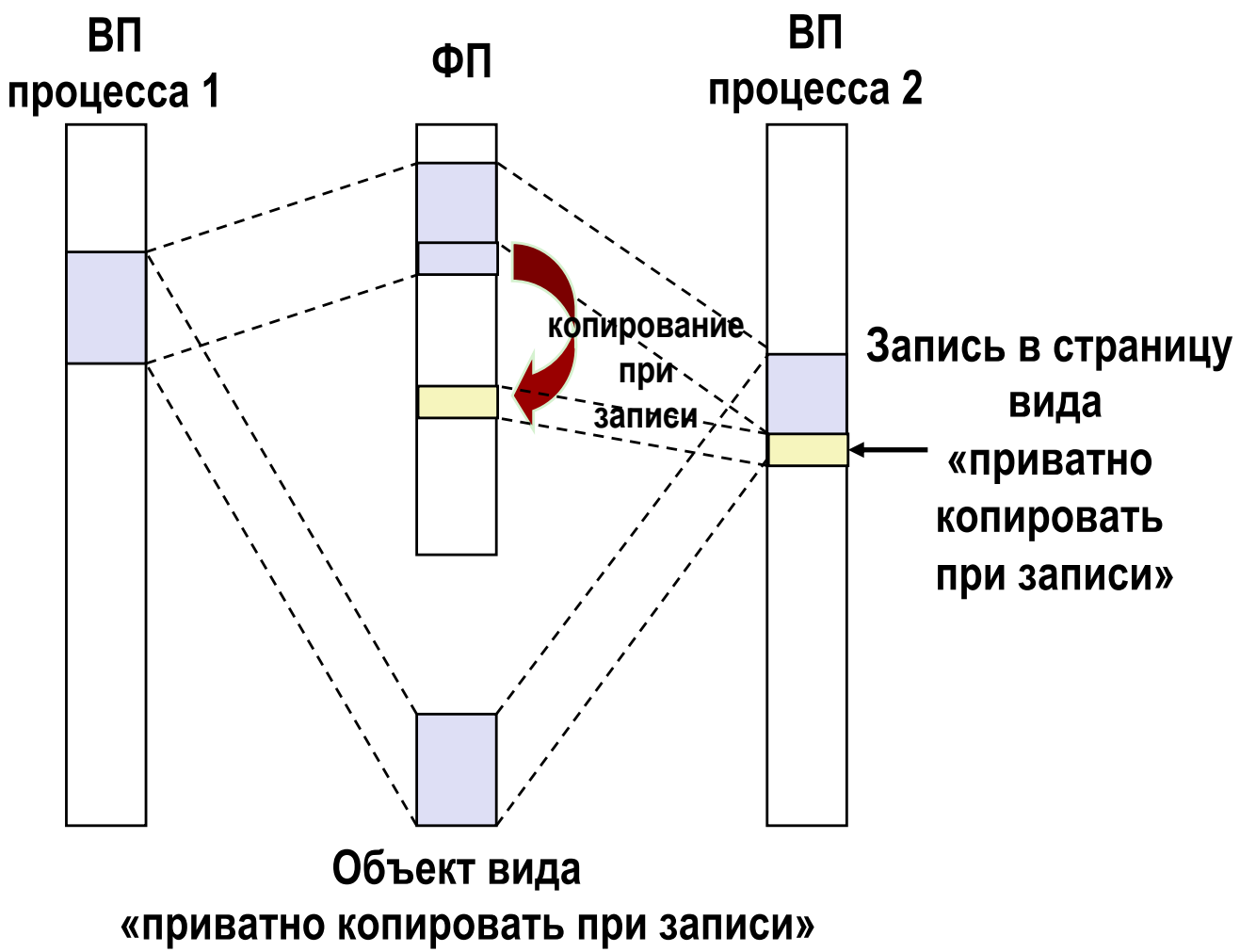
Использование mmap для копирования файлов

- Копирование без передачи данных в пространство пользователя

```
#include "csapp.h"

/*
 * mmapcopy - использует mmap
 *            для копирования
 *            файла fd в stdout
 */
void mmapcopy(int fd, int size)
{

   /* Указатель на область ВП
      отображённую в ФП */
    char *bufp;

    bufp = mmap(NULL, size,
                PROT_READ,
                MAP_PRIVATE, fd, 0);
    write(1, bufp, size);
    return;
}
```

```
/* Пускатель mmapcopy */
int main(int argc, char **argv)
{
    struct stat stat;
    int fd;

    /* Проверка необходимых параметров*/
    if (argc != 2) {
        printf("usage: %s <filename>\n”,
                argv[0]);
        exit(0);
    }

    /* Копируем параметр в stdout */
    fd = Open(argv[1], O_RDONLY, 0);
    fstat(fd, &stat);
    mmapcopy(fd, stat.st_size);
    exit(0);
}
```

Вновь о разделении: приватная копия объекта при записи



- Команда записи в приватную страницу даёт сбой защиты
- Обработчик создаёт новую страницу вида чтение/запись
- Команда перезапускается по выходе из обработчика
- Копирование откладывается насколько возможно!

Вновь о функции `fork`

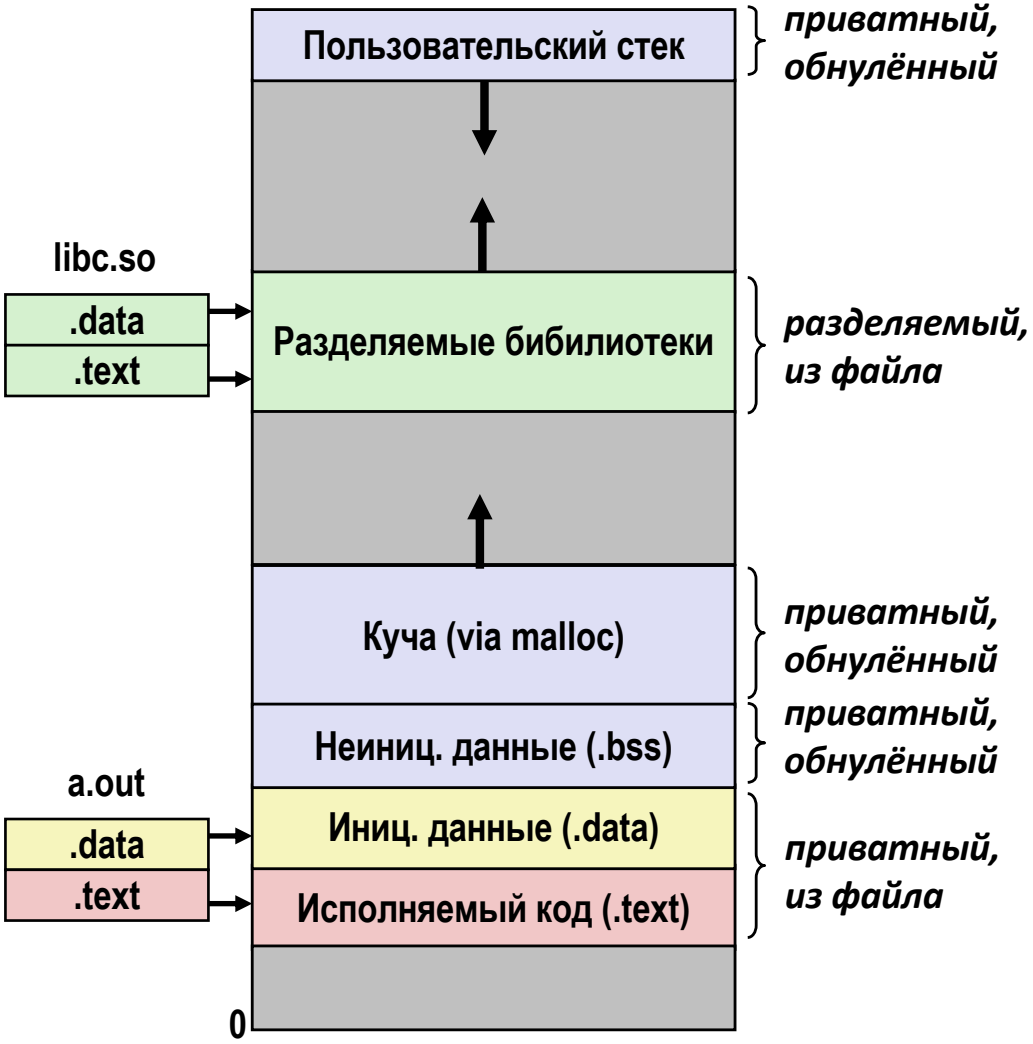
- **ВП и отображение памяти проясняют как `fork` обеспечивает приватное адресное пространство для каждого процесса**

- **При создании нового процесса ядро операционной системы**
  - В точности копируеет текущие `mm_struct`, `vm_area_struct`, и таблицы страниц.
  - Помечает «только чтение» каждую страницу в обоих процессах
  - Помечает «приватное копирование при записи» каждую `vm_area_struct` в обоих процессах

- **При возврате, у каждого процесса – точная копия ВП**

- **Очередные записи создают новые страницы с помощью механизма копирования при записи (COW)**

Вновь о функции execve



- **Для загрузки и исполнения новой программы `a.out` в текущем процессе используем `execve`:**
- **Освобождаем старые `vm_area_struct` и таблицы страниц**
- **Создаём новые `vm_area_struct`'s и таблицы страниц**
  - Код и инициализированные данные из объектных файлов
  - `.bss` и стек – из безымянных файлов
- **Устанавливаем PC на точку входа в `.text`**
  - Linux при необходимости будет обслуживать сбои страниц кода и данных.

Отображение памяти пользователем

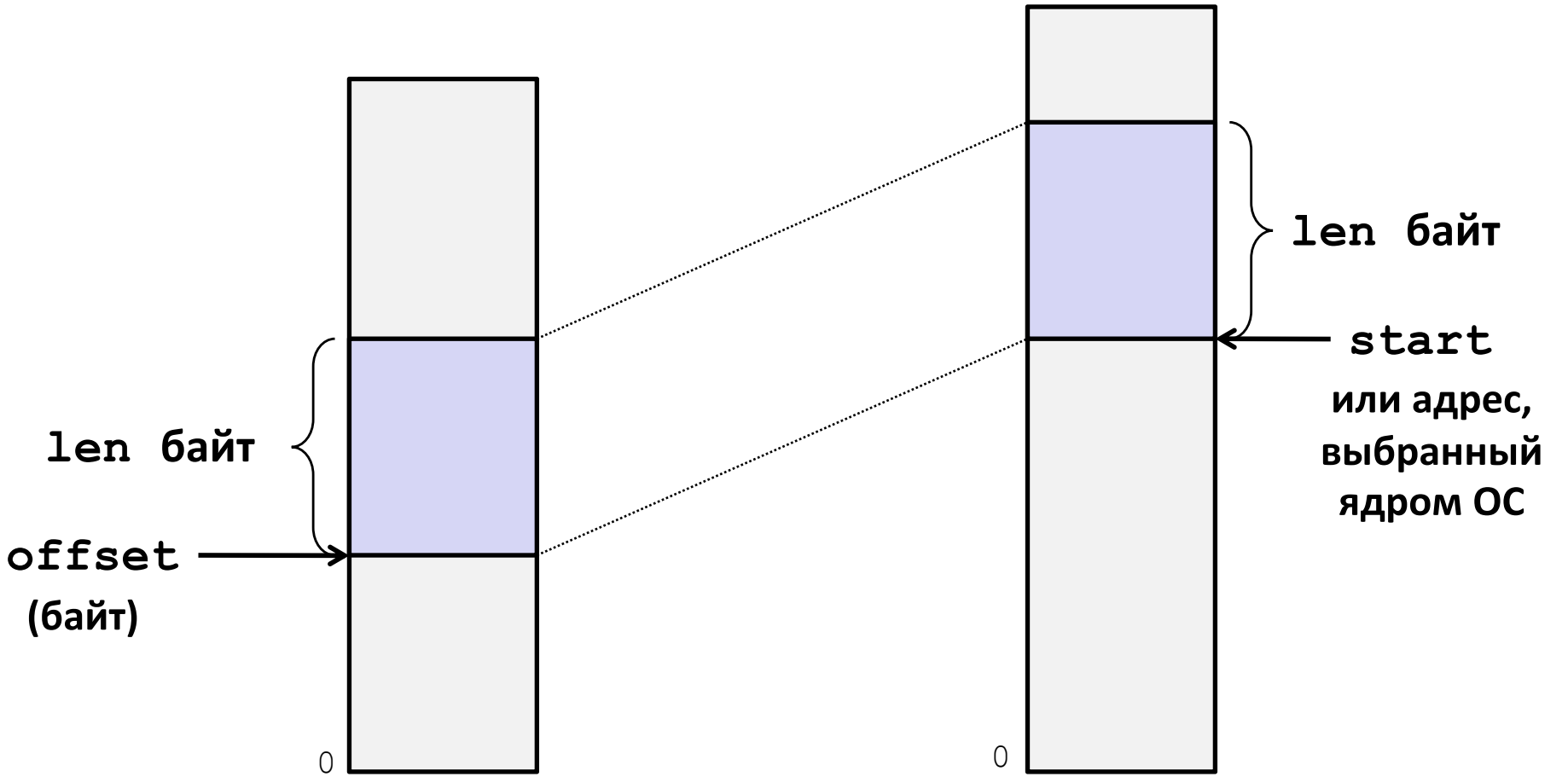
```
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset)
```

- **Отображает `len` байт начиная со смещения `offset` в файле, указанном файловым дескриптором `fd`, предпочтительно с адреса `start`**
  - **`start`:** может быть 0 для “выбери адрес”
  - **`prot`**: PROT_READ, PROT_WRITE, ...
  - **`flags`**: MAP_ANON, MAP_PRIVATE, MAP_SHARED, ...

- **Возвращает указатель на начало отображённой области (возможно не `start`)**

Отображение памяти пользователем

```
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset)
```



Дисковый файл, указанный файловым дескриптором **fd**

Виртуальная память процесса