Лекция 8. Управление памятью

CS221. Архитектура компьютера и операционные системы

23 ноября 2017 г.

Задачи управления памятью

Задачи

- Перемещаемость кода и данных.
- Защита памяти.
- Совместное использование памяти.
- Логическая организация в виде сегментов, модулей.
- Многоуровневая память.

Однородное распределение памяти

Рис. 1: организация памяти в ZX Spectrum 48

Организация памяти в виде банков

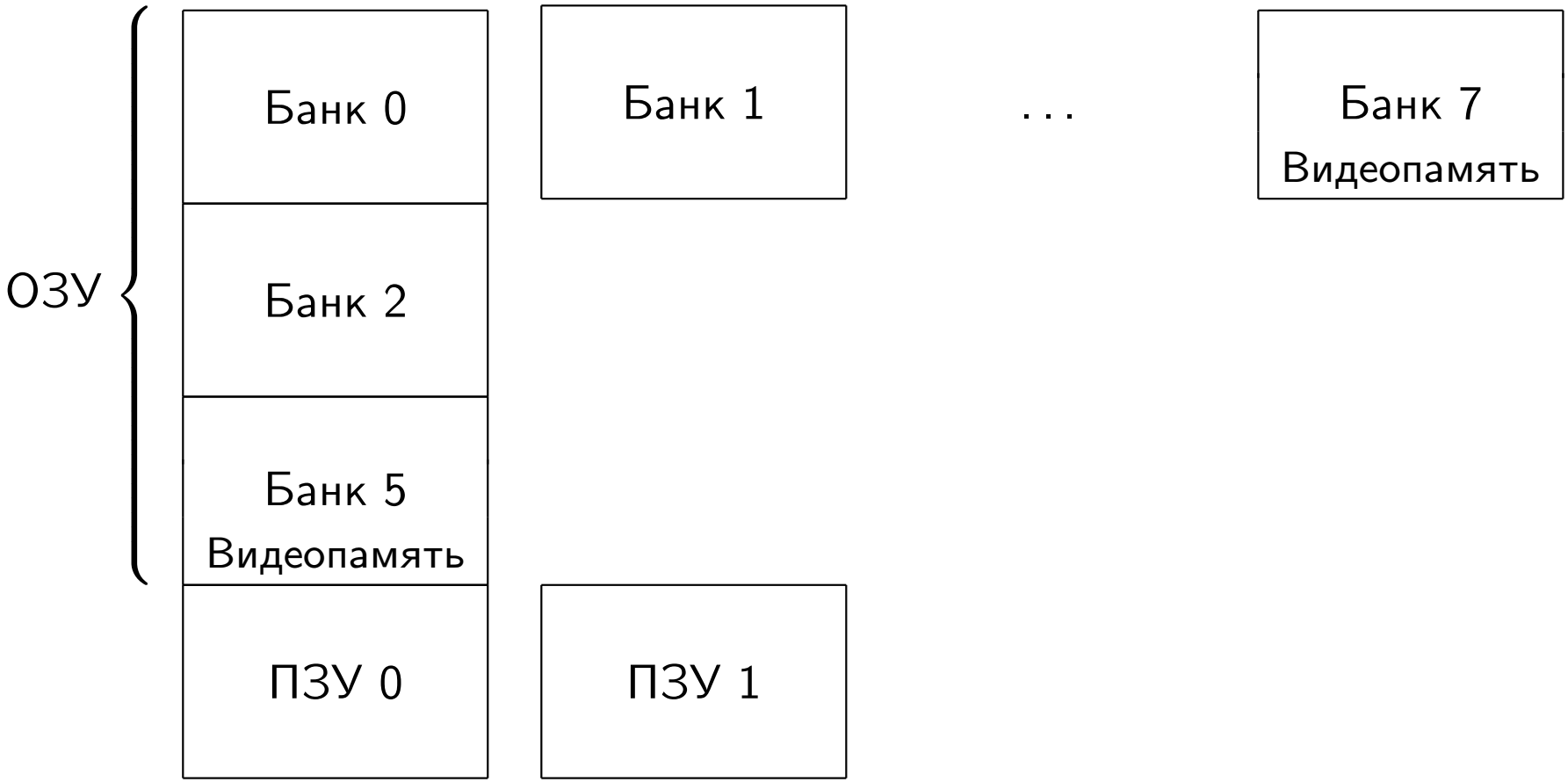

Рис. 2: организация памяти в ZX Spectrum 128

Блок управления памятью

Определение

Блок управления памятью: (memory management unit, MMU) — аппаратное устройство вычислительной системы, управляющее обращениями к памяти. Функции:

- организация виртуальной памяти;
- защита памяти;
- управление кэшами;
- арбитраж шины;
- переключение банков памяти.

Пример (Altera Nios II)

- Без MMU: μClinux, FreeRTOS, ChibiOS/RT, eCos, ...
- С MMU: Linux, ...

Виды управления памятью

Техники

- Однородное непрерывное размещение (single contiguous).
- Секционное размещение (partitioned).
- Сегментированное размещение (segmented).
- Страничное размещение (paged).

Виды фрагментации

Внешняя: (external) — неиспользованные блоки малого размера между занятыми.

Внутренняя: (internal) — неиспользованная память внутри блоков.

База и граница

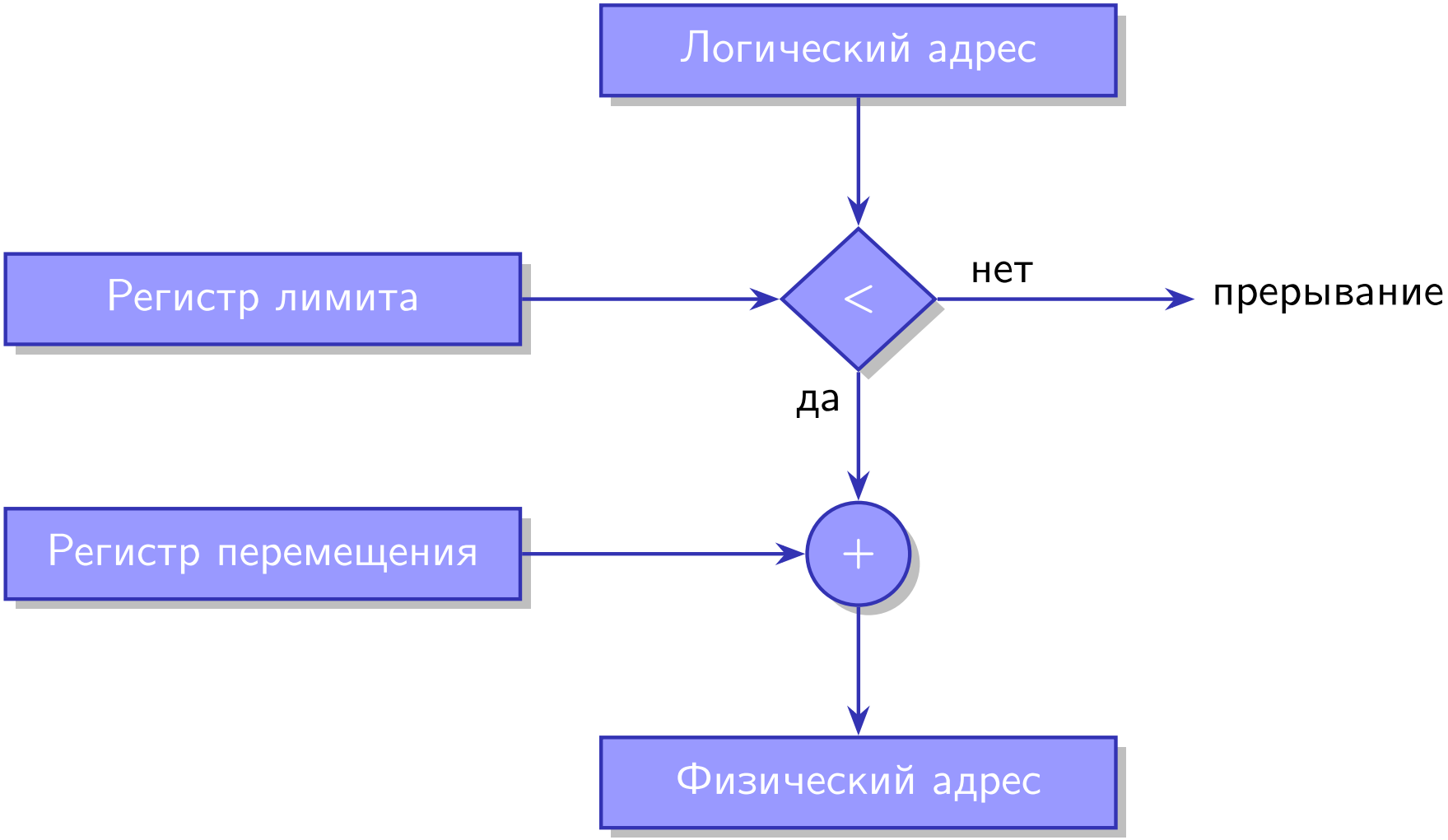

Рис. 3: техника управления памятью «база и граница» (base and limit)

Фиксированное распределение памяти

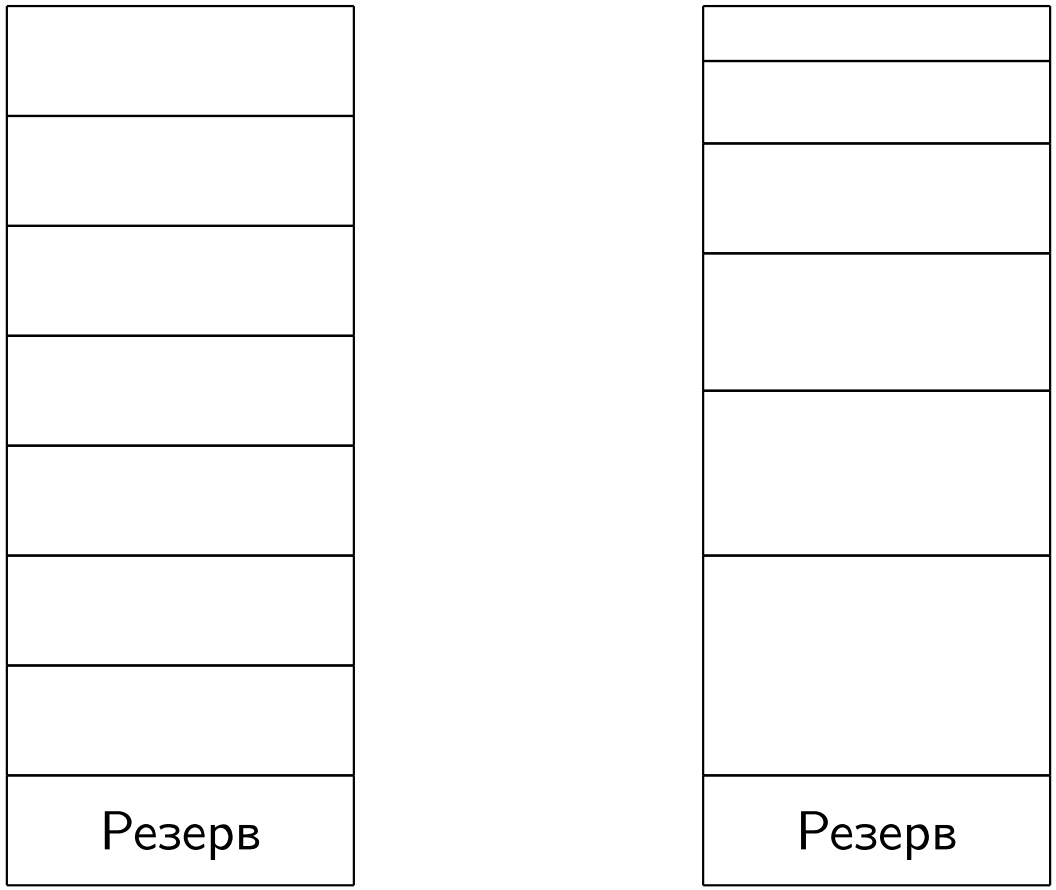

Рис. 4: концепция фиксированного распределения паяти

Распределение памяти в OS/360

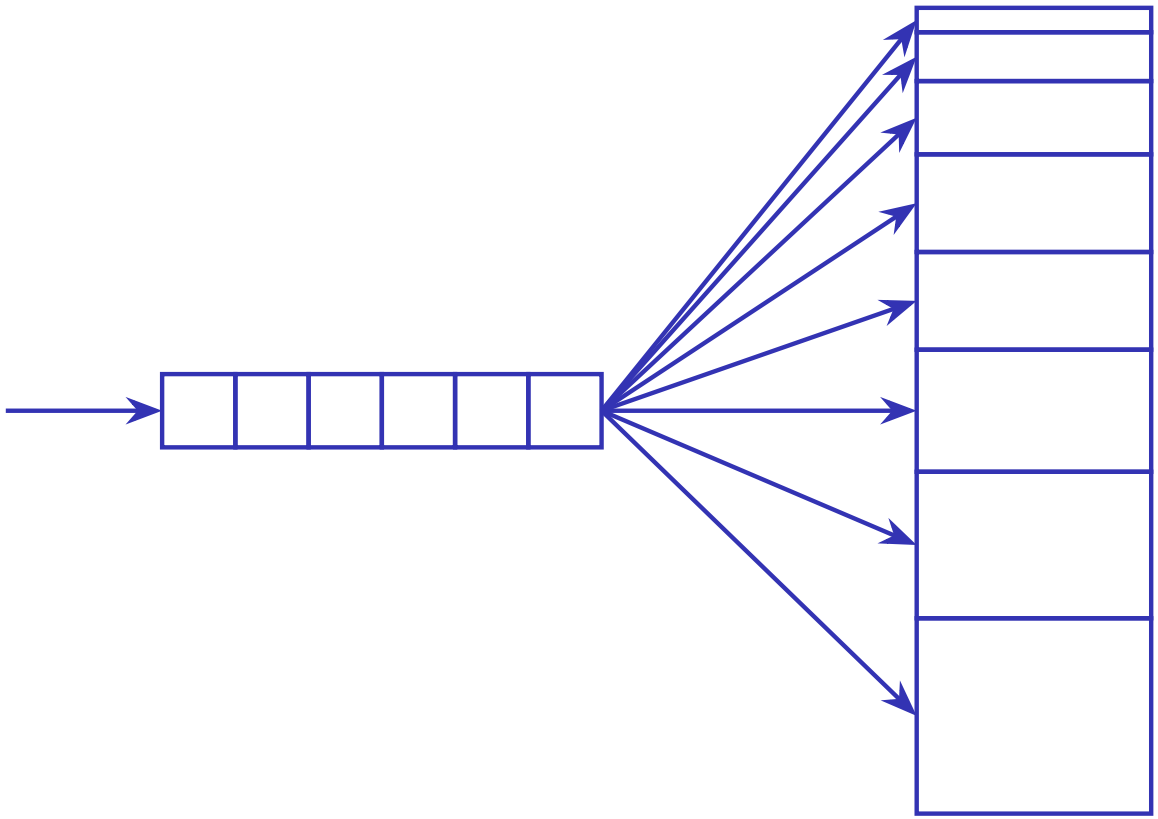

Рис. 5: организация памяти в IBM System/360 (OS/360) — фиксированное количество задач

Варианты OS/360

Версии ядра

- Однозадачный последовательный планировщик (Single Sequential Scheduler, SSS).
- Многозадачность с фиксированным количеством задач (Multiprogramming with a Fixed number of Tasks, MFT).
- Многозадачность с переменным количеством задач (Multiprogramming with a Variable number of Tasks, MVT).

Определение

Механизм ключей защиты: (Memory Protection Keys) — сравнивает ключи защиты, связанные с процессом и областью памяти: System/360, System z, HP/PA, Itanium, x86-64.

Способы разбиения памяти

Определения

Сегмент: (segment) — область памяти заданного размера, соответствующая логическому назначению информации:

- основная программа;
- библиотека подпрограмм;
- динамически загружаемая библиотека;
- глобальные данные;
- стек;
- куча;
- ...

Примеры

1961	Burroughs B5000	1975	Intel iAPX 432
1964	GE-645 (Multics)	1988	IBM AS/400

Сегменты 8086 (1978)

Рис. 6: адресация паяти в реальном режиме процессора 80x86

Сегменты 8086 (окончание)

Вычисление физического адреса

$$\text{адрес} = \texttt{сегмент} \times 16 + \text{смещение}$$

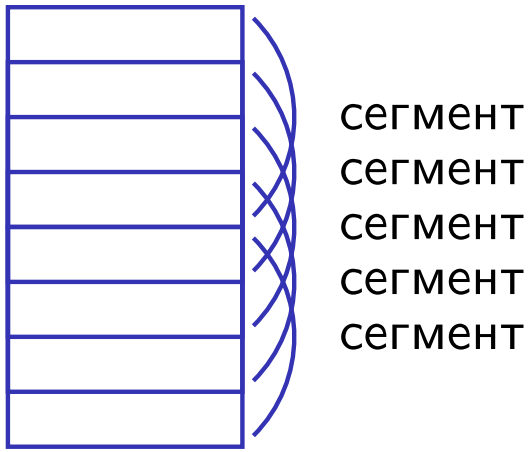

Рис. 7: адресация памяти в реальном режиме процессора 80x86

Короткие и длинные адреса

Пример (Assembler 8086)

```
mov     BX, 01h
mov     AX, word ptr [BX]     ; короткая адресация: DS:[BX]
mov     ES, AX
mov     CX, ES:[BX]           ; длинная адресация
```

Пример (Borland C++ 3.1 и т. д.)

```
int n;
int *far lpn = &n;
int *near npn = &n;
```

CS	кода
DS	данных
SS	стека
ES	результаты строковых инструкций (MOVS, ...)

Таблица 1: сегментные регистры 8086

Использование сегментов в MS-DOS

Элемент таблицы перемещений

```
struct EXE_RELOC
{
  unsigned short offset;
  unsigned short segment;
};
```

Вычисление физического адреса

```
адрес = (base_seg + segment):offset
*((word *) адрес) += base_seg
CS = CS_init + base_seg
SS = SS_init + base_seg
```

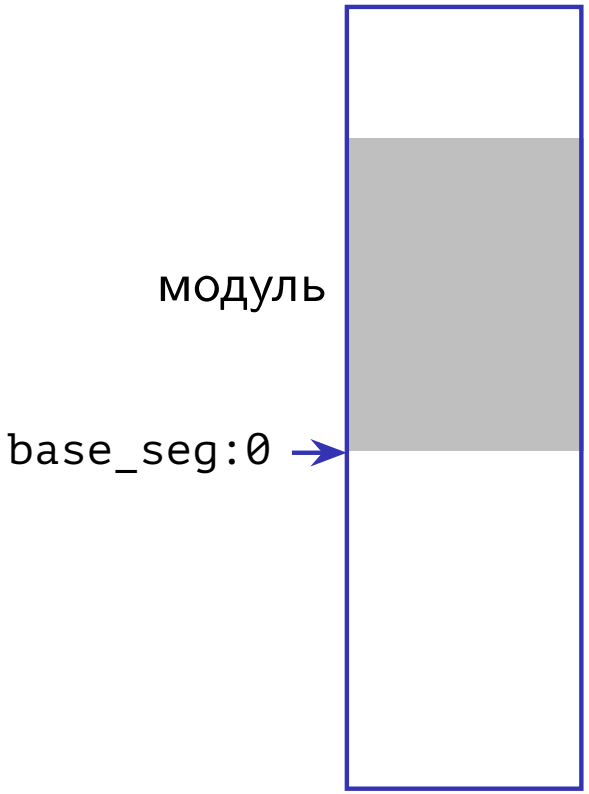

Рис. 8: загруженный в память модуль EXE.

Режимы адресации, начиная с 80286 (1982)

Режимы

- реальный (2^{20} байт = 1 Мбайт);
- защищённый (2^{24} байт = 16 Мбайт).

Назначение реального режима

- Совместимость с предыдущим ПО;
- Начальная загрузка системы.

Селектор

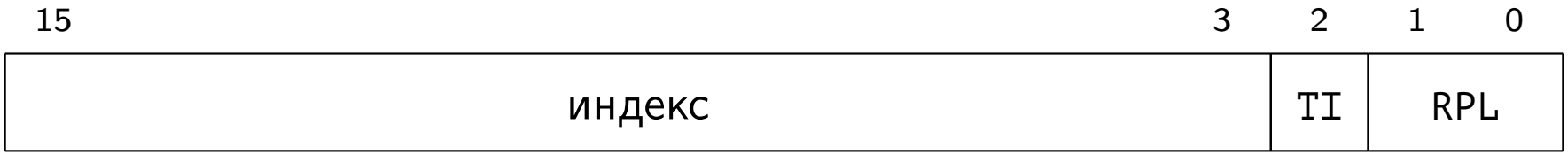

Рис. 9: формат селектора в защищённом режиме процессора 80286

Поле	Значения	Название	Примечание
RPL	0 . . . 3	Requested Privilege Level	0 ∼ режим ядра
TI	0 . . . 1	Table Indicator	0 ⇒ GDT, 1 ⇒ LDT
индекс	0 . . . 8 191	—	

Таблица 2: значения полей селектора

Сегментные регистры

Обозн.	Название	Примечание
CS	Code Segment	∋ CPL — Current Privilege Level
DS	Data Segment	
SS	Stack Segment	
ES	Extra Segment	
GDTR	Global Descriptor Table Register	адрес (32 бит) и размер (16 бит)
LDTR	Local Descriptor Table Register	адрес (32 бит) и размер (16 бит)

Таблица 3: сегментные регистры процессоров архитектуры X86

Трансляция адресов (80286)

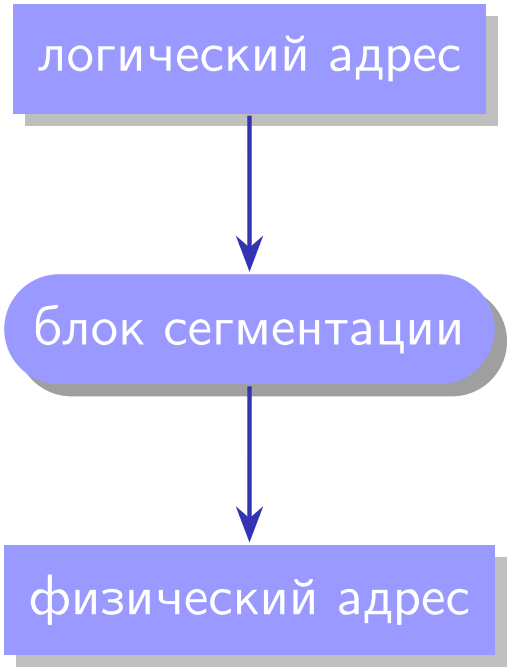

Рис. 10: трансляция адресов в защищённом режиме процессора 80286

Трансляция адресов (продолжение)

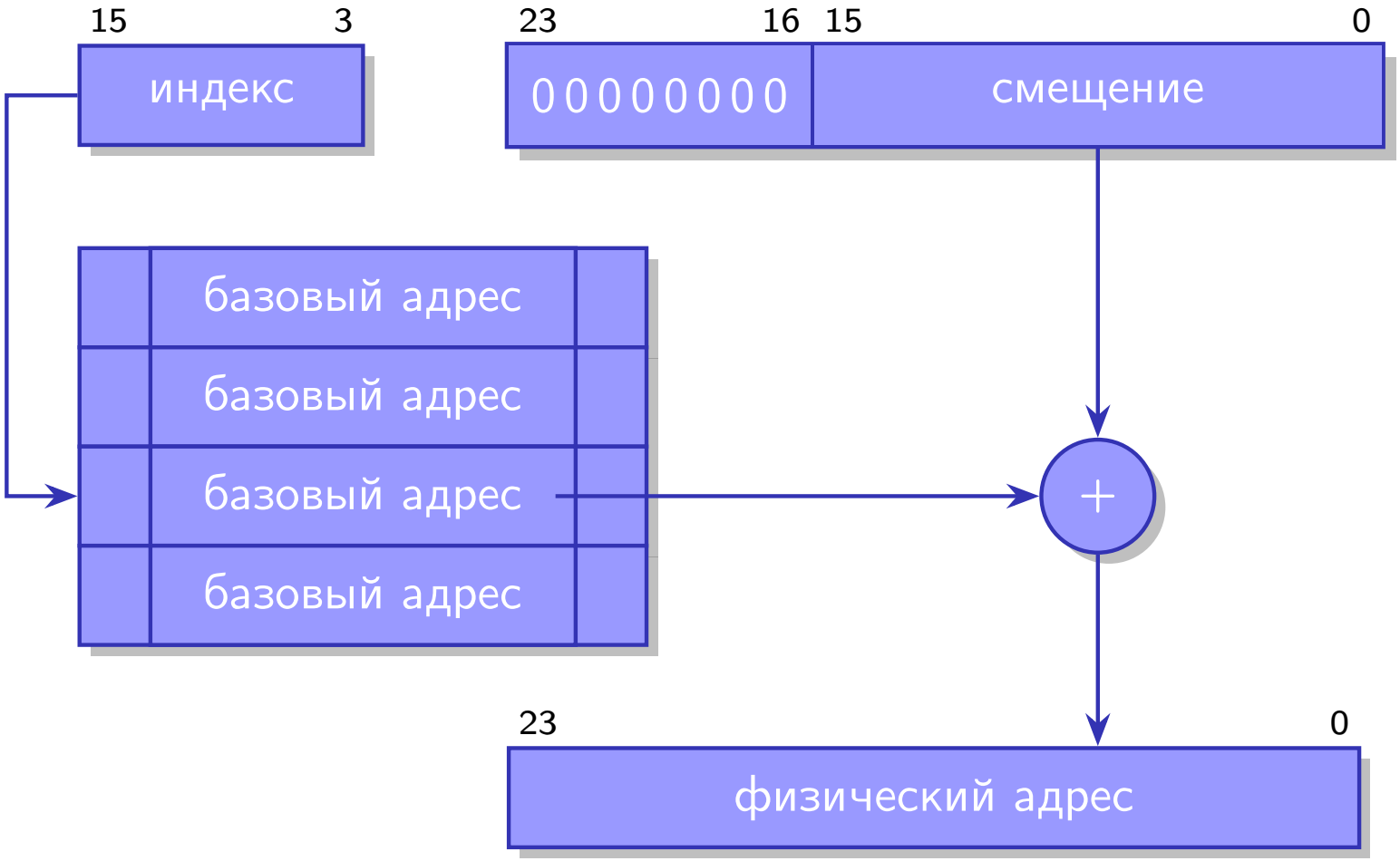

Рис. 11: трансляция адресов в защищённом режиме процессора 80286

Поля дескриптора сегмента (8 байт)

Поле	Бит	Название	Примечание
`Base`	32		линейный адрес начала
`G`	1	Granularity	0 ⇒ размер в байтах, иначе в 4 096-байтных блоках.
`Limit`	20		размер (1 байт ... 1 Мбайт или 4 Кбайт ... 4 Гбайт)
`S`	1	System	0 ⇒ хранит системные данные (LDT, ...) иначе обычный сегмент кода/данных.
`DPL`	2	Descriptor Privilege Level	Уровень привилегий.
`P`	1	Присутствие	1 ⇒ сегмент присутствует (в Linux всегда).
`Type`	4		тип сегмента

Таблица 4: некоторые поля дескриптора сегмента процессора 80286

Типы сегментов

Тип	Таблица	S	Хранение
кода	GDT, LDT	1	сегмент кода
данных	GDT, LDT	1	сегмент данных или стека
состояния задачи (TSS)	GDT	0	сегмент с процессорными регистрами.
локальной таблицы дескрипторов	GDT	0	сегмент с LDT.

Таблица 5: некоторые виды сегментов процессора 80286

Работа блока сегментации

Алгоритм

1. `TI = 0` $\Rightarrow$ базовый линейный адрес из GDTR, иначе — из LDTR.
2. адрес дескриптора сегмента = базовый линейный адрес + 8 $\times$ индекс;
3. $\max\{\texttt{CPL}, \texttt{RPL}\} > \texttt{DPL} \Rightarrow$ General Protection Fault (GP);
4. Из дескриптора сегмента выбирается поле «базовый адрес» (`Base`);
5. линейный адрес = базовый адрес + смещение.

Сегментация в поздних архитектурах

80386 (1985)

- Смещения увеличены с 16 до 32 бит.
- Дескрипторы сегмента увеличены с 24 до 32 бит.
- Добавлены регистры `FS` и `GS`.
- Добавлена страничная адресация.

x86-64 (2000, AMD)

- В длинном режиме регистры `CS`, `DS`, `SS` и `ES` принудительно устанавливаются в 0, предел $\sim$ сегментов: 2^{64} байт.

Использование сегментации в ОС

Linux

- Для каждого процессора существует GDT, содержащая 4 дескриптора сегментов (`Base = 0`, `G = 1`, `Limit = 0xFFFFF` $= 2^{20} - 1 \Rightarrow 4$ Гбайт):
  - Для всех процессов ядра (`DPL = 0`):
    - код ядра;
    - данные ядра;
  - Для всех процессов пользователя (`DPL = 3`):
    - код пользователя;
    - данные пользователя.
  - несколько сегментов TLS (используется `GS`).
- Для каждого процесса создаётся LDT по умолчанию ядром. Некоторые приложения обращаются к нему (Wine, . . .)

Использование сегментации в ОС (окончание)

Windows 3.1

- 1 GDT, 1 LDT для системной виртуальной машины и по 1 LDT для каждой ВМ для DOS-приложения (режим V86).
- Сегменты кода, данных приложений могут помечаться как фиксированные, перемещаемые, перемещаемые + удаляемые, ...

Windows NT

Модель flat: 1 сегмент для кода и данных для приложения, сегментные регистры не должны меняться им.

Особенности сегментной памяти

Достоинства

- Лучшая защита доступа по типам содержимого.

Недостатки

- Внешняя фрагментация;
- Необходимость в уплотнении (compaction);
- Прерывистость адресного пространства;
- Ограничение размера адресного пространства объёмом физической памяти.

Способы разбиения памяти

Определения

Страница: (page) — диапазон линейных адресов (размер зависит от архитектуры, ~ 4 Кбайт) и данные по ним.

Страничный кадр: (page frame, физическая страница) — единица разбиения физической памяти, содержит 1 страницу.

Трансляция адресов (80386)

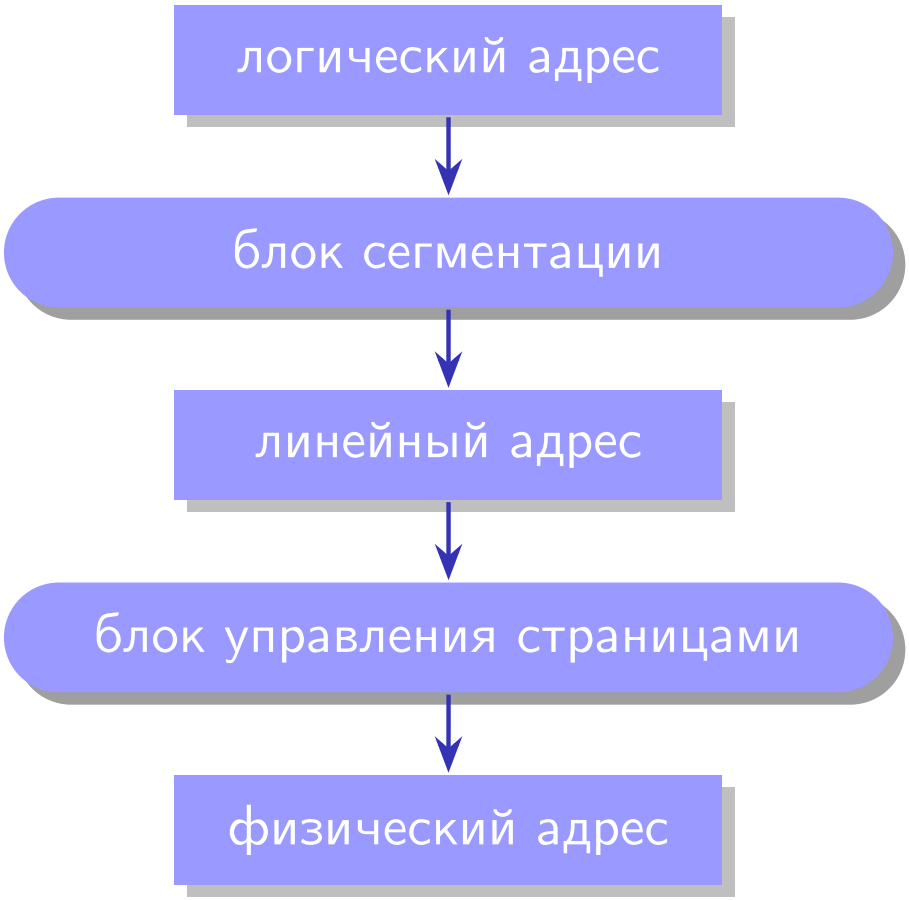

Рис. 12: трансляция адресов в защищённом режиме процессора 80386

Трансляция адресов (продолжение)

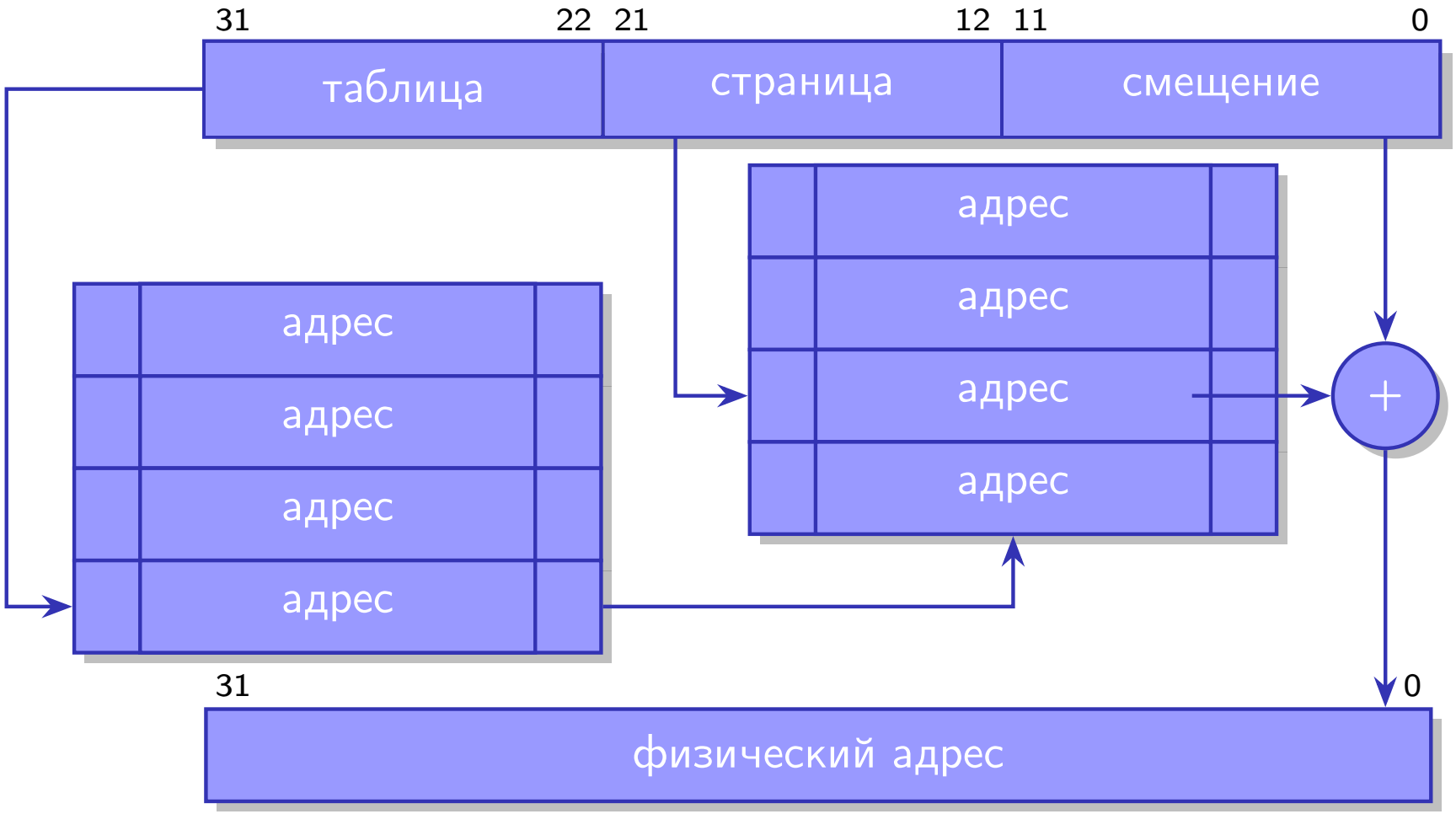

Рис. 13: трансляция адресов в защищённом режиме процессора 80386

Поля таблицы страниц (4 байт)

Поле	Бит	Примечание
Present	1	флаг присутствия
адрес	20	20 старших бит.
Accessed	1	устанавливается, когда блок управления адресует соответствующий кадр
Dirty	1	(для таблицы страниц) устанавливается, когда выполняется запись в кадр
Read/Write	1	права доступа на запись.
User/Supervisor	1	уровень привилегий ($= 3$ или < 3).
Page Size	1	(для каталога страниц) $= 1 \Rightarrow$ размер страницы = 4 Мбайт

Таблица 6: некоторые поля элемента таблицы страниц процессора 80386

Работа блока управления страницами

Алгоритм

1. Адрес элемента Каталога Таблицы Страниц = *CR3 + 4 × таблица;
2. Адрес Таблицы Страниц — из поля «адрес» элемента Каталога Таблиц;
3. Адрес элемента Таблицы Страниц = адрес Таблицы Страниц × 4 096 + 4 × страница;
4. Адрес страницы — из поля «адрес» дескриптора Таблицы Страниц;
5. Физический адрес = адрес страницы × 4 096 + смещение.

Вместимость Каталога Таблиц/Таблицы Страниц

До 1 024 элементов.

Буфер ассоциативной трансляции

Определение

Буфер ассоциативной трансляции: (*translation lookaside buffer, TLB*) — кэш, отображающий (недавно вычисленные) линейные адреса в физические.

Расширение размера страниц

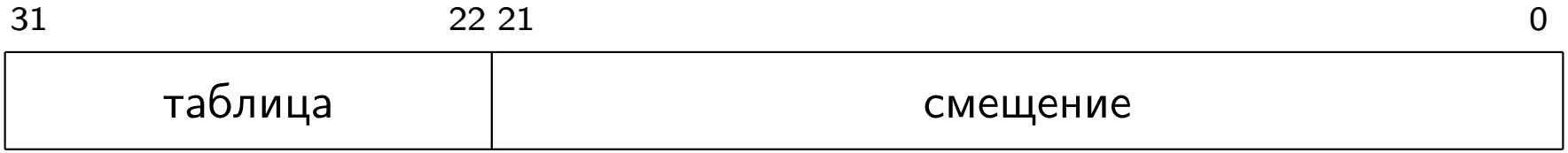

Рис. 14: формат смещения в режиме адресации расширенных страниц процессора Pentium

Определение

Режим расширения размера страниц: (*Page size extension, PSE*) — режим, включаемый полем PSE регистра CR4 (поле «Page size» элемента таблицы = 1), сосуществует с обычным режимом.

Механизм расширения физических адресов

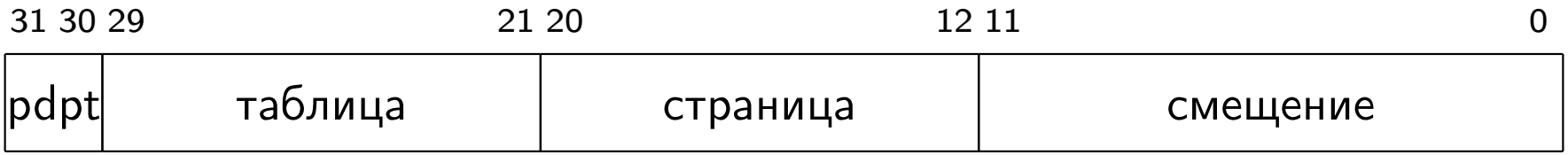

Рис. 15: формат смещения в режиме расширения физических адресов процессора Pentium Pro

Определение

Механизм расширения физических адресов: (*Physical Address Extension, PAE*) — режим, включаемый полем PAE регистра CR4. Отличия:

- поле «адрес» элемента таблицы увеличивается с 32 до 36 бит $\Rightarrow$ элемент таблицы увеличивается с 4 до 8 байт;
- количество элементов таблиц уменьшается с 1 024 до 512.

Логические адреса в x86-64

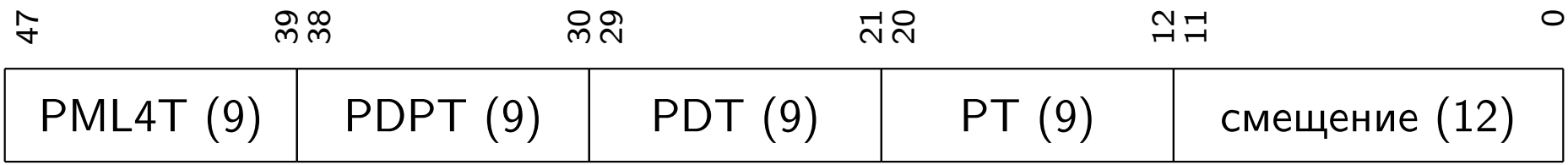

Рис. 16: формат смещения в режиме long mode архитектуры AMD64 (x86-64)

PML4T	Page-map level-4 table
PDPT	Page-directory pointer table
PDT	Page-directory table
PT	Page table

Таблица 7: уровни каталогов страниц x86-64

Использование страничной адресации в Windows 3.1

Windows 3.1

Страничный (32-битный) режим адресации используется только виртуальными драйверами, работающими в 0-м кольце защиты.

Состояния страниц

Состояния страниц в диспетчере памяти

- Свободная (free).
- Зарезервированная (reserved).
- Переданная (committed, private).
- Разделяемая (shareable).

Определение

Копирование при записи: (Copy-on-Write) — копирование при попытке изменения разделяемых данных.

Переходы состояний страниц

Последовательность работы со страницами

1. Резервирование (reserving).
2. Передача (committing).
3. Первое использование $\rightarrow$ создание, обнуление.
4. Возможная запись в файл подкачки.
5. Возврат (decommitting).
6. Освобождение (freeing).

Адресное пространство процесса

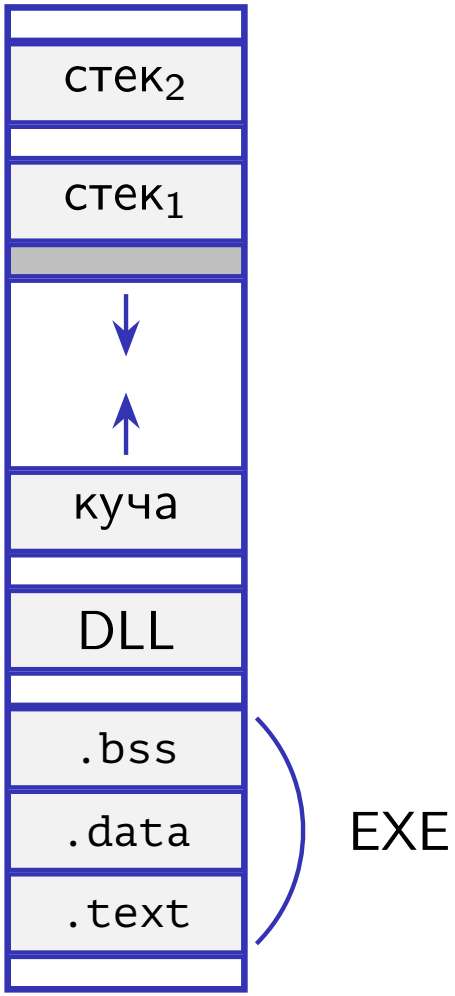

Рис. 17: расположение секций в памяти

Секции

Название	Назначение
`.text`	Секция кода.
`.data`	Секция данных чтения/записи. Глобальные переменные.
`.rdata`	Секция данных только на чтение. Строковые литералы, таблицы виртуальных функций C++/COM.
`.idata`	Таблица импорта.
`.edata`	Таблица экспорта.
`.rsrc`	Ресурсы, только для чтения.
`.bss`	Неинициализированные данные.
`.crt`	Данные библиотеки поддержки времени выполнения C++.
`.tls`	Данные локального хранилища потока.
`.reloc`	Базовые смещения в исполняемом файле.
`...`	Пользовательские секции.

Таблица 8: секции исполняемых модулей

Секции модуля

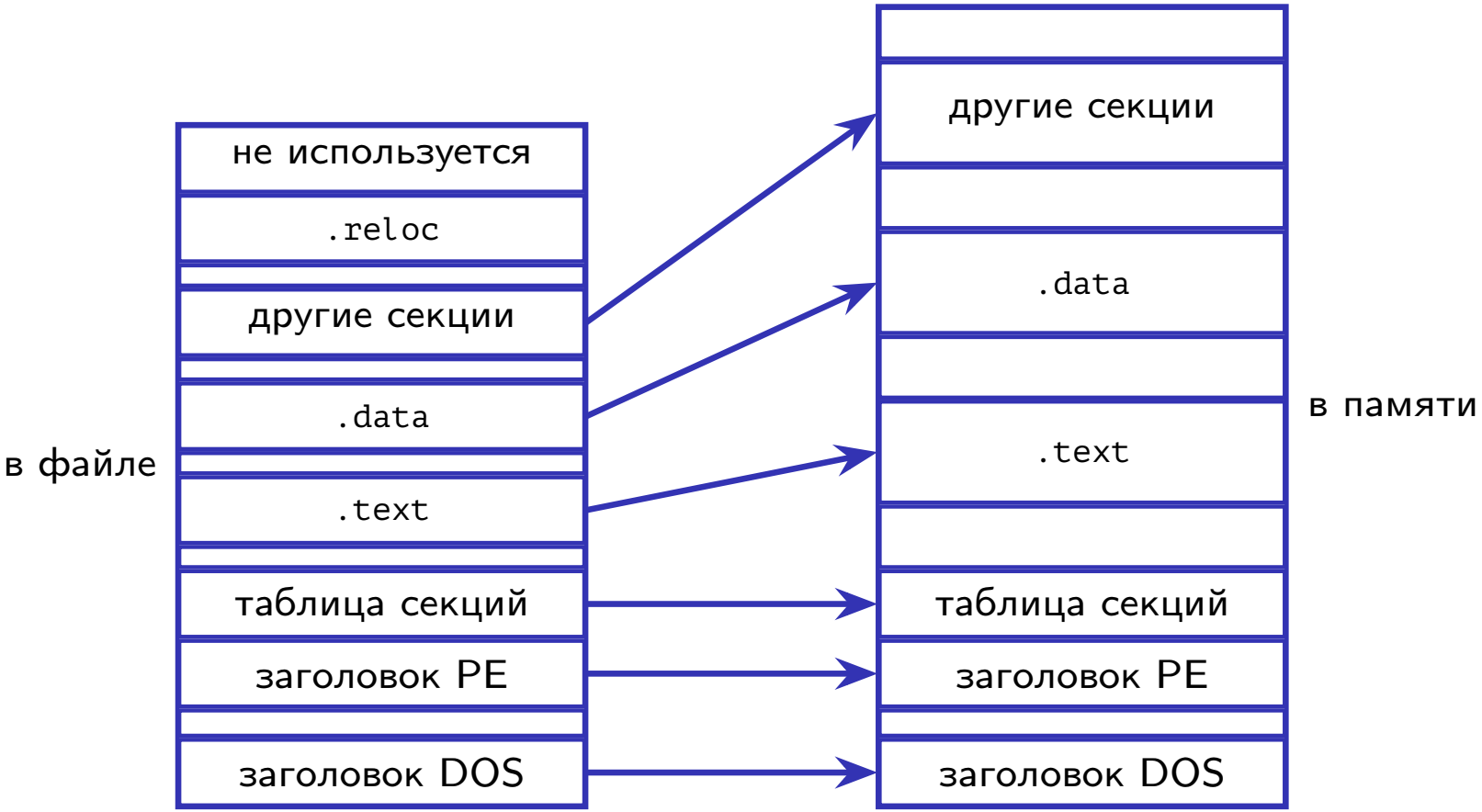

Рис. 18: секции модуля

Адресное пространство процесса в 32-битном режиме

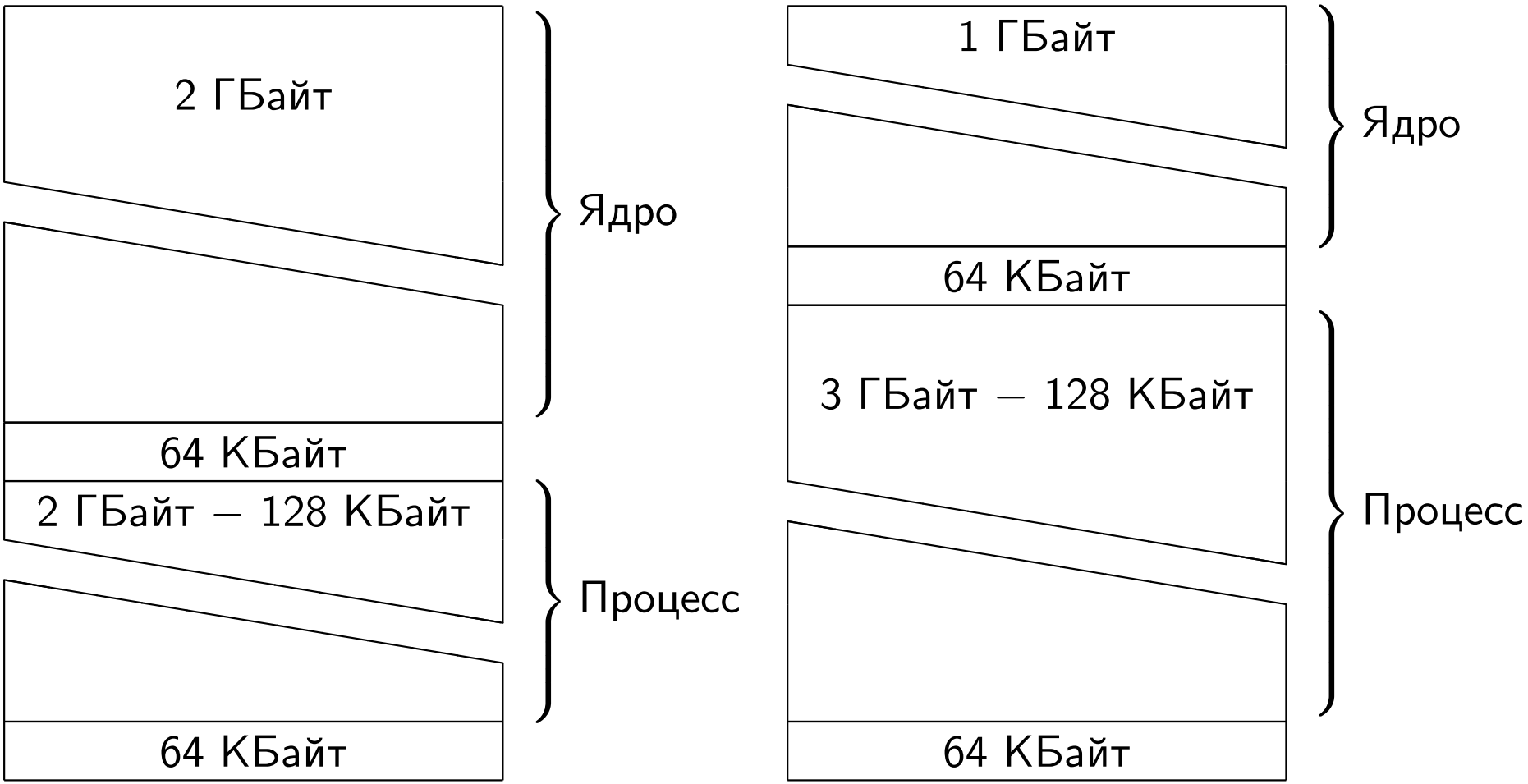

Рис. 19: адресное пространство процесса в Windows

Адресное пространство процесса в 64-битном режиме

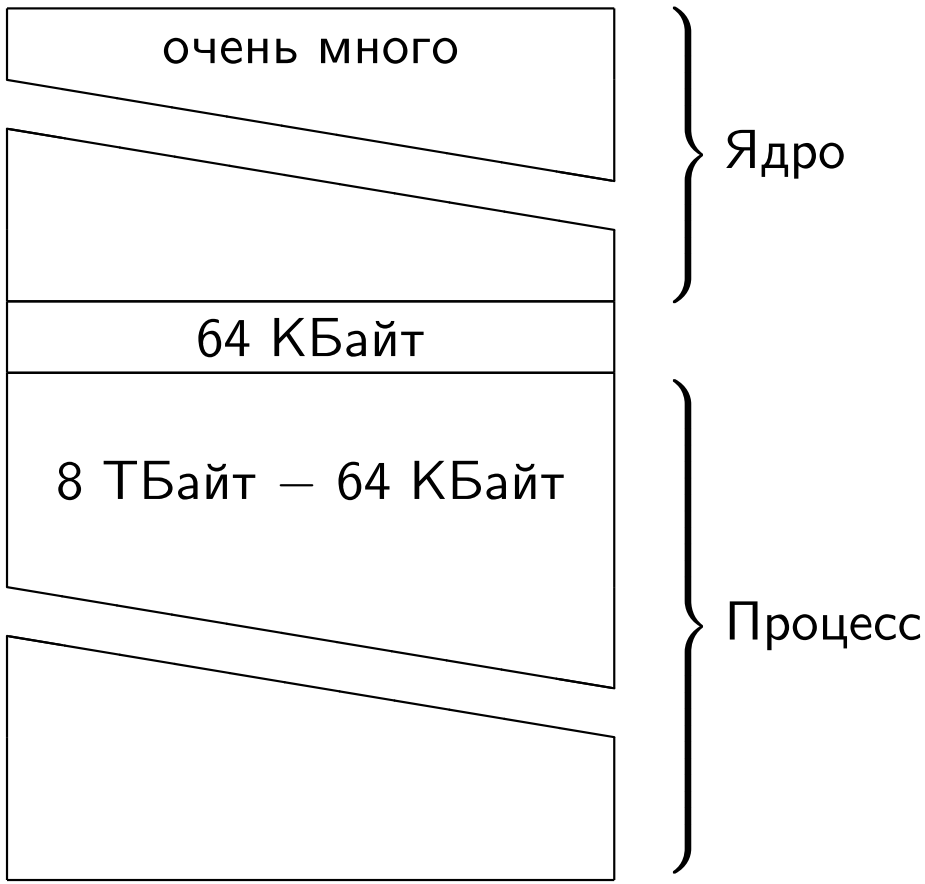

Рис. 20: адресное пространство процесса в Windows-64

Структура виртуального адресного пространства

Виды данных в адресном пространстве

- Принадлежащие процессу (код/данные);
- Принадлежащие сеансу (код/данные);
- Принадлежащие системе (код/данные);
  - Код ядра;
  - Драйверы;
  - Таблицы страниц процесса;
  - Области динамической системной подкачиваемой/неподкачиваемой памяти;
  - Системный кэш;
  - ...

Использование PSE

Возможности использования больших страниц

- Базовые образы ОС (`ntoskrnl.exe` и `hal.dll`);
- Базовые системные данные (структуры, описывающие состояния страничных кадров, ...);
- Запросы большого ввода/вывода;
- Для приложений закрытые области памяти, выделенные `VirtualAlloc()` с использованием флага `MEM_LARGE_PAGE`;
- Другие заданные драйверы при настройке в реестре.

Адресное пространство процесса в 32-битном режиме

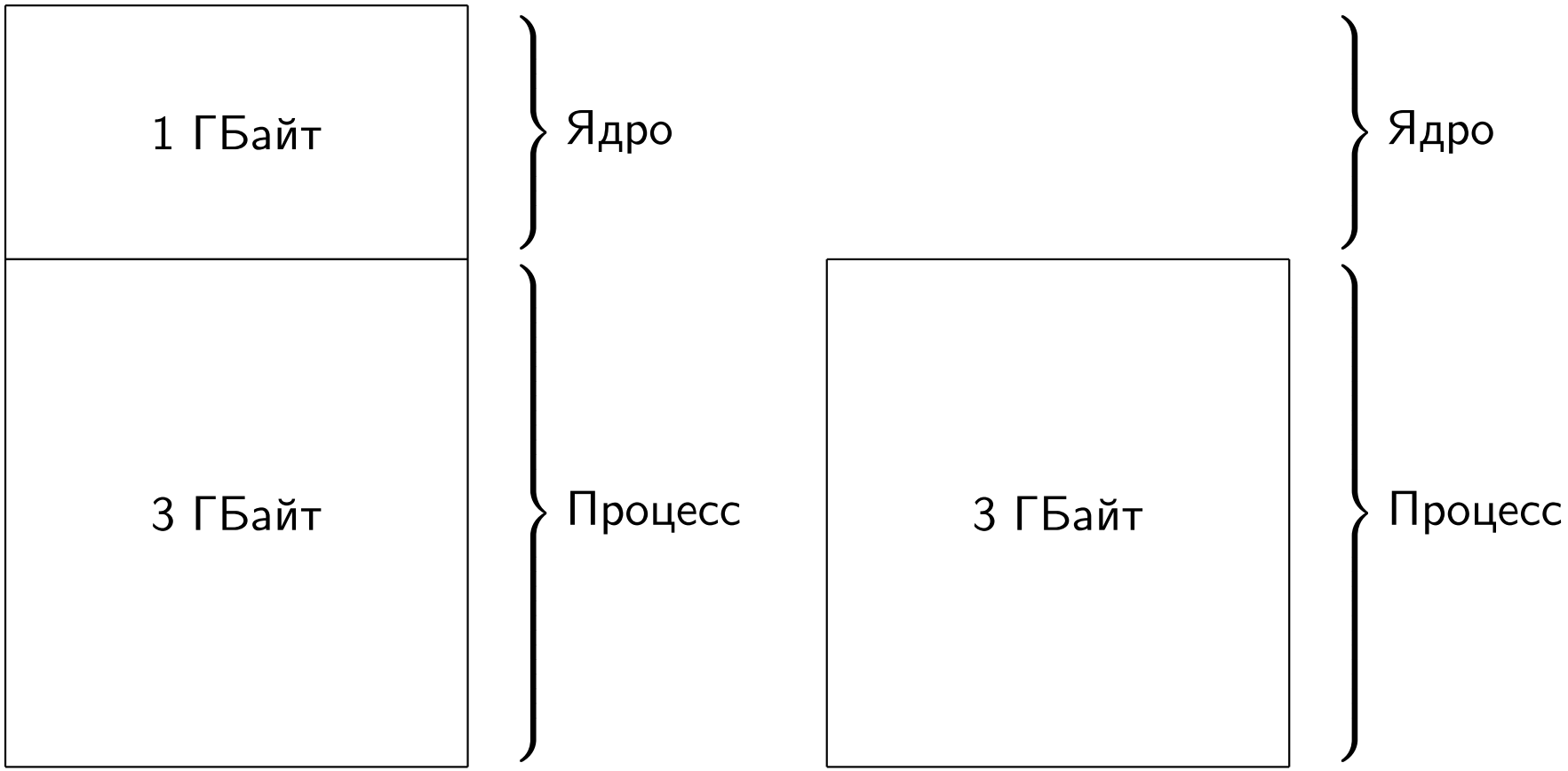

Рис. 21: адресное пространство процесса в Linux

Особенности распределения адресного пространства

Организация виртуального адресного пространства процессов

- Ядро сохраняет регистр CR3 в дескрипторе процесса $\Rightarrow$ каждый процесс имеет собственный глобальный каталог таблиц.
- Первые 3 ГБайт адресного пространства доступны для процессов, различны.
- Последний 1 ГБайт доступен в режиме ядра, отображается в одну область памяти.
- Потоки ядра работают в 3–4 Гбайтном диапазоне верхних адресов, не ссылаясь ниже $\Rightarrow$ неважно, какой таблицей страниц они пользуются. Для избежания лишних сбросов TLB-буферов используется таблица дескрипторов последнего процесса.

Управление динамической памятью

Виды оперативной памяти

- постоянно выделенная ядру;
- динамическая.

Запросы на выделение памяти со стороны ядра

Удовлетворяются немедленно:

- ядро является приоритетным компонентом;
- ядро доверяет самому себе.

Запросы на выделение памяти со стороны пользовательского процесса

- процессу сразу выделяется диапазон ячеек, само выделение откладывается до момента обращения (лениво);
- код процесса может содержать ошибки.

Исключения при обращении к памяти

Причины

- Адреса, указанные вследствие программных ошибок;
- Адреса принадлежат отсутствующей странице, хотя принадлежат адресному пространству процесса $\Rightarrow$ соответствующий кадр должен быть выделен.

Обработка исключения обращения к странице

если *адрес* $\in$ *адресному пространству процесса,* **то**
- **если** *вид доступа* $\sim$ *правам доступа к области,* **то**
  - допустимое обращение, выделить страничный кадр;
- **иначе**
  - недопустимое обращение, послать `SIGSEGV`;

иначе
- **если** *исключение в режиме пользователя,* **то**
  - недопустимое обращение, послать `SIGSEGV`;
- **иначе**
  - ошибка ядра, уничтожить процесс;

Рис. 22: упрощённый алгоритм обработки исключения обращения к странице

«Жадное» выделение памяти

Старая реализация функции `fork()`

1. Выделение страничных кадров под таблицы страниц потомка.
2. Выделение страничных кадров под страницы потомка.
3. Инициализация таблиц страниц потомка
4. Копирование страниц родителя в страницы потомка.

«Ленивое» выделение памяти

Особенности реализации функции `fork()`

- Страничные кадры не копируются, а используются совместно родителем и потомком.
- Страничные кадры помечаются недоступными для записи.
- Если родитель или потомок попытается записать в общую страницу, возникнет исключение, ядро создаст копию, оригинал останется доступным только для чтения.
- Если потом произойдёт попытка записи в исходную страницу, ядро проверит, является ли процесс единственным владельцем её (поле дескриптора страницы `_count == 0`). Да $\Rightarrow$ пометит её как доступную на запись.
- `_count == -1` $\Rightarrow$ страница считается свободной.

Выделение страниц по требованию

Пример (проверка выделения памяти)

```
int main()
{
  try
  {
    char *pchData = new char[100000000];
    // ...
  }
  catch (const std::bad_alloc &)
  {
    // ...
  }
}
```

Утилизация страничных кадров

Особенности работы с памятью

ОС Linux не делает проверок размеров доступной памяти $\Rightarrow$ кеши диска и т. д. растут неограниченно.

Определение

Алгоритм утилизации страничных кадров: (*Page Frame Reclaiming Algorithm, PFRA*) — выбирает занятые страничные кадры для выгрузки на диск.

Выгрузка страниц

Виды страниц

- Неутилизируемые (динамически выделенные ядру, временно заблокированные, . . .)
- Выгружаемые (анонимные в АП режима пользователя)
- Синхронизируемые (отображение файлов в режиме пользователя, дисковые кэши, . . .)
- На выброс (неиспользуемые, в кэшах памяти).

Случаи утилизации

- Дефицит памяти в системе;
- Гибернация;
- Периодическая утилизация.