# УПРАВЛЕНИЕ ОПЕРАТИВНОЙ ПАМЯТЬЮ

Лектор: доц., к.т.н.,
доцент кафедры ИСТАС
Иванов Н.А.

2021 г.

# Понятие «Оперативная память»

**Оперативная память** (*Random Access Memory, RAM*, память с произвольным доступом) или **оперативное запоминающее устройство** (**ОЗУ**) — энергозависимая часть системы компьютерной памяти, в которой во время работы компьютера хранится выполняемый машинный код (программы), а также входные, выходные и промежуточные данные, обрабатываемые процессором.

В отличие от внешней памяти (жесткий диск, USB флеш-накопитель, CD и DVD диски) оперативной памятью для сохранения информации требуется постоянное электропитание.

# ОЗУ современных компьютеров

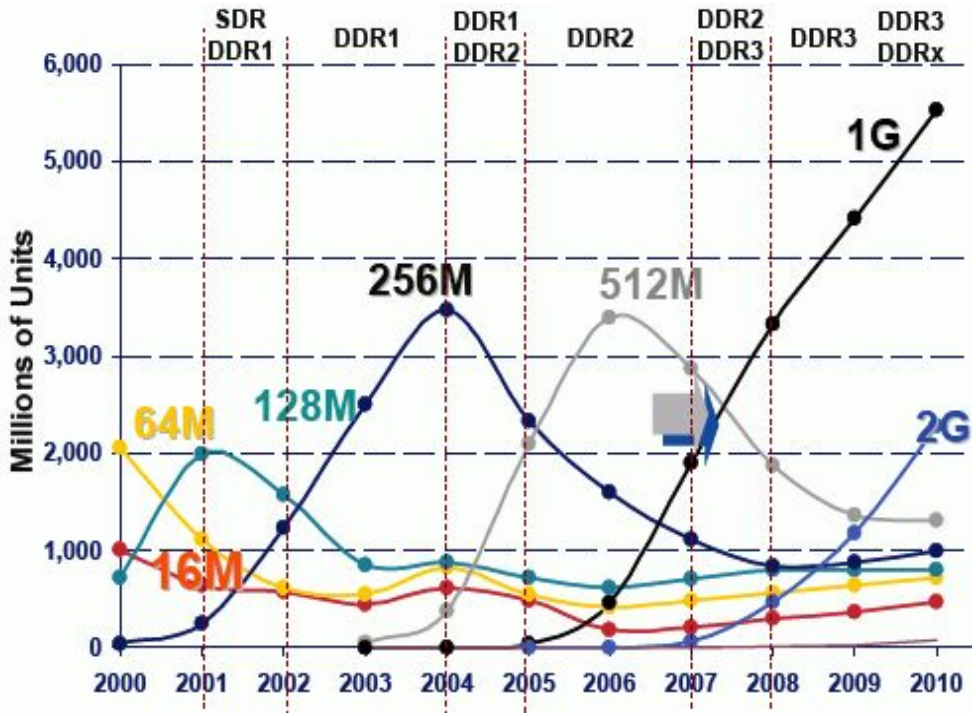
Наибольшее распространение получили два вида ОЗУ:

- динамическая память (***DRAM***), в виде массива конденсаторов;
- статическая память (***SRAM***), в виде массива триггеров;

***DRAM*** хранит бит данных в виде заряда конденсатора.
Однобитовая ячейка памяти содержит конденсатор и транзистор. Конденсатор заряжается до более высокого или низкого напряжения (логические 1 или 0). Транзистор выполняет функцию ключа, подключающего конденсатор к схеме управления, расположенного на том же чипе. Схема управления позволяет считывать состояние заряда конденсатора или изменять его.

***SRAM*** хранит бит данных в виде состояния триггера.
Этот вид памяти является более дорогим в расчёте на хранение 1 бита, но имеет наименьшее время доступа и меньшее энергопотребление, чем *DRAM*.

# SRAM и DRAM: использование, достоинства и недостатки




В современных компьютерах статическая память SRAM используют как кэш второго уровня (L2) и имеет относительно небольшой объем (обычно 1-6 Мб).
В кэше SRAM используется потому, что к ней предъявляется очень жесткие требования в плане производительности и надежности.

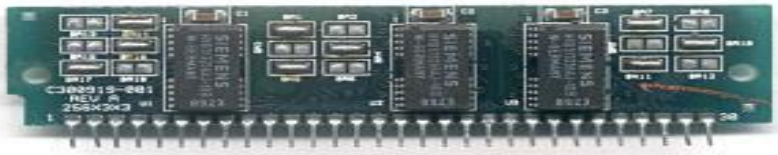
Основную память компьютера (ОЗУ) составляют микросхемы *динамической* памяти. Память типа DRAM широко применяется в компьютерной технике благодаря двум основным достоинствам перед SRAM - *дешевизне* и *плотности хранения данных*.

В настоящее время существует множество видов памяти DRAM, появившихся как результат попыток производителей и разработчиков памяти угнаться за интенсивным прогрессом в области центральных процессоров.
Широко применяются следующие типы DRAM: Video RAM, DDR SDRAM, DDR2 SDRAM, DDR3 SDRAM и DDR4 SDRAM.

# ОЗУ современных компьютеров: исполнение

В зависимости от форм-фактора выводы могут: реализовываться *в виде штырьков* (**DIP, SIPP**), располагаться *в виде дорожек*, подходящих к краю ножевого разъёма с одной стороны платы (**SIMM**), либо с двух сторон — **DIMM**.

DIP



SIPP

SIMM 30 pin

72 pin

DIMM

DDR DIMM

Эволюционное развитие конструкции модулей памяти, используемых в качестве ОЗУ компьютера.

Как отличить типы памяти SIMM, DIMM, DDR, DDR2, DDR3
http://abramov-online.ru/blog/2010/03/16/simm-dimm-ddr-ddr2-ddr3/

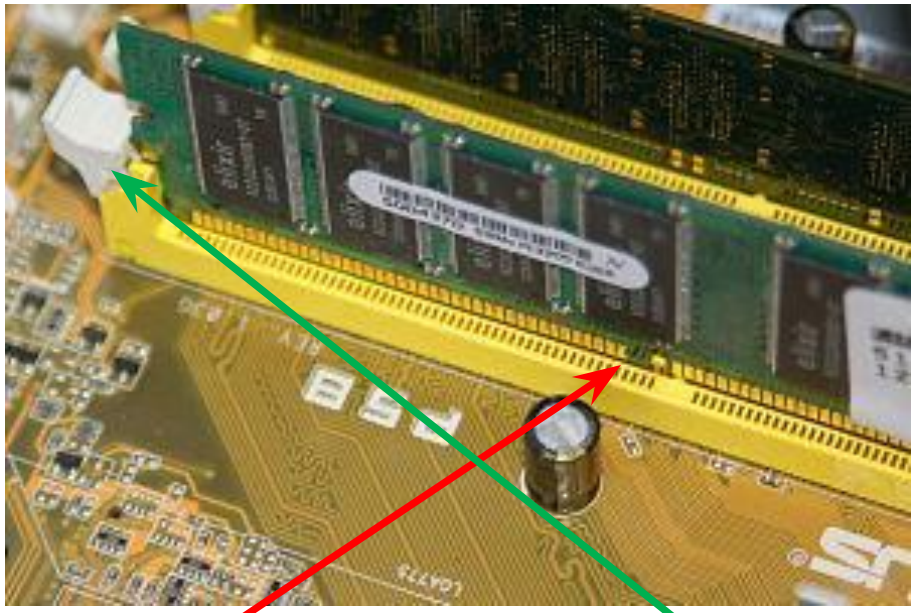
# ОЗУ современных компьютеров: подключение

Для корректной установки модуля памяти необходимо:
1) выбрать модуль подходящего типа 2) правильным способом его ориентировать

Средства защиты от неправильной ориентации модуля в разъёме

**«ключ»** — особая выемка в группе контактов, механически препятствующая установке модуля неподходящего поколения в конкретно взятый разъём

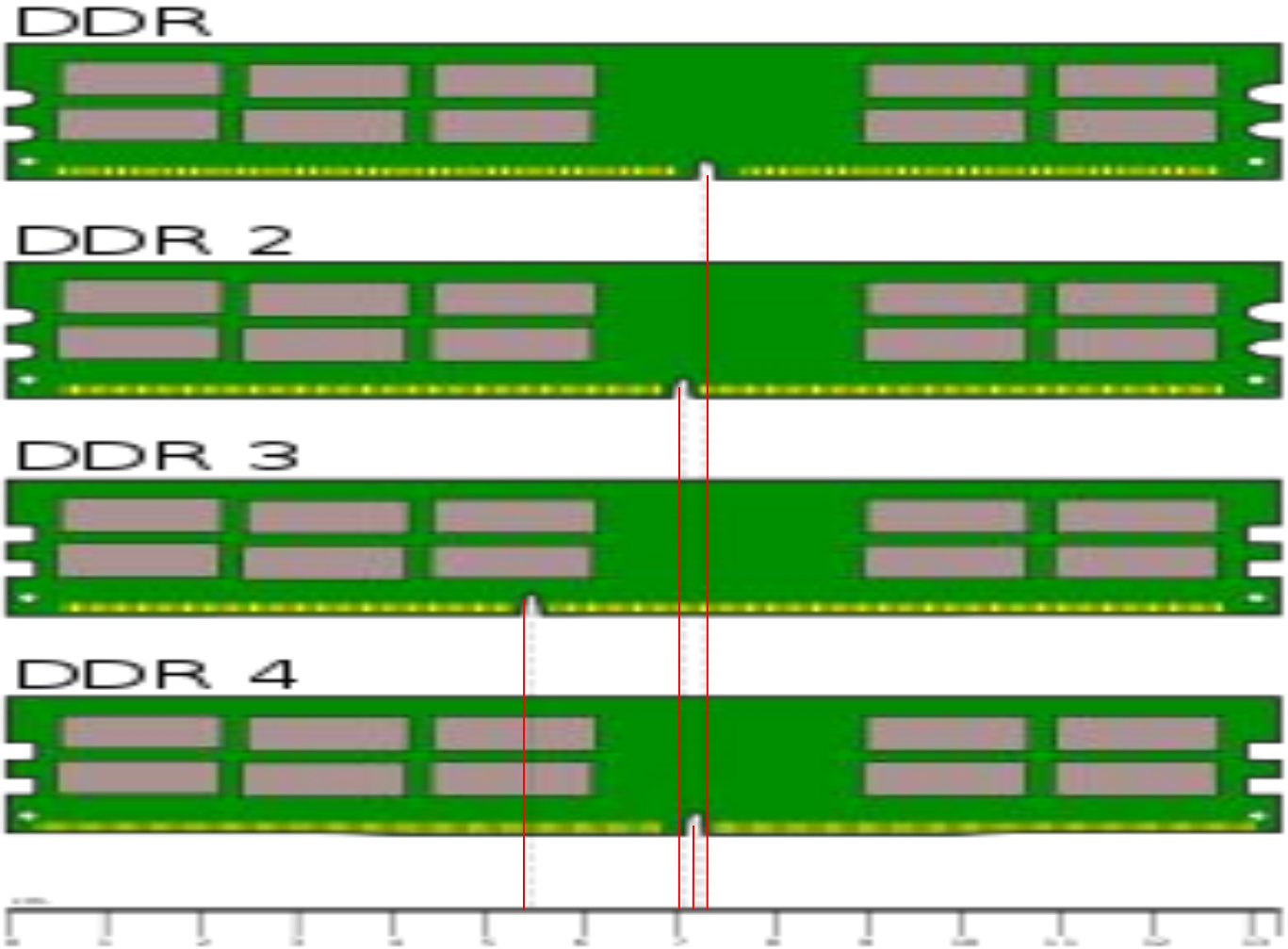
**защёлки** на разъеме, плотно фиксирующие модуль в разъёме — при правильной установке модуля в разъем с последующей фиксацией защёлок должен раздаться характерный щелчок



«Ключ»-выемка на модуле и защёлки на разъёме препятствуют некорректной установке модуля памяти.

# ОЗУ современных компьютеров: подключение

В различных „поколениях“ и видах DDR, «ключ»-выемка находится в разных местах разъёма.




Кроме того, размер и положение «ключа»-выемки могут кодировать дополнительные особенности модуля: отличающиеся от основной массы модулей напряжение питания, наличие схем ECC.

# Особая роль оперативной памяти

Оперативная память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы.

Особая роль оперативной памяти объясняется тем, что *процессор может выполнять инструкции программ только в том случае, если они находятся в оперативной памяти*.

Память распределяется как между модулями прикладных программ, так и между модулями самой операционной системы.

# Управление памятью в ОС

## Ранние однозадачные ОС

---

Управление памятью сводилось к загрузке программы и ее данных с некоторого внешнего накопителя (перфоленты, магнитной ленты или [illegible]) в память

## Многозадачные ОС

---

Важнейшими стали задачи, связанные с распределением имеющейся памяти между несколькими одновременно выполняющимися программами.

# Функции многозадачной ОС по управлению памятью

- ***отслеживание свободной и занятой памяти***;
- ***выделение памяти*** процессам и ***освобождение* памяти** по завершении процессов;
- ***динамическим распределением памяти;***
- ***вытеснение кодов и данных процессов*** из оперативной памяти на диск (полное или частичное), когда размеры основной памяти не достаточны для размещения в ней всех процессов, и ***возвращение их в оперативную память***, когда в ней освобождается место;
- ***настройка адресов программы на конкретную область физической памяти***;
- ***защита памяти***.

# Отслеживание свободной и занятой памяти

## Методы учета памяти.

| Метод | Пример применения |
|---|---|
| Параметрический:<br>- количественно<br>- адресами | Свободная память:<br>100 Мб<br>С 3-го по 4 Гб |
| Двоичных шкал:<br>100111 | Описатель участка памяти имеет бит занятости |
| Метод связных списков | Описатель участка памяти содержит ссылки на следующий и предыдущий участки |
| Табличный метод | НЕ ИСПОЛЬЗУЕТСЯ!!!<br>Предназначен ТОЛЬКО для учета внешних устройств! |

# Задачи защиты памяти в операционных системах

Защита адресного пространства одного процесса от попыток воспользоваться им другим процессом

Защита адресного пространства операционной системы от попыток воспользоваться им со стороны прикладного процесса

Защита адресного процесса от самого себя

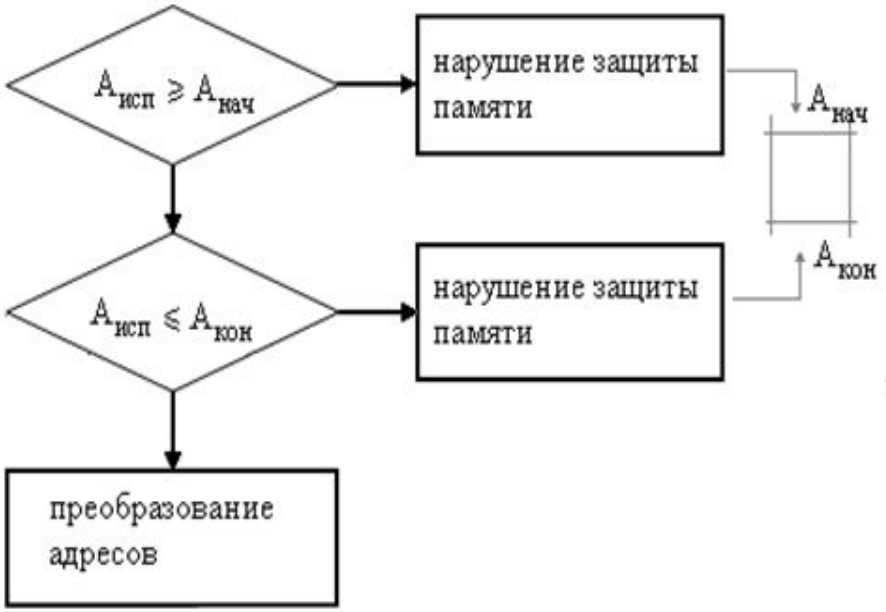
# Методы защиты памяти в операционных системах

| | | |
|---|---|---|
| Защита памяти по граничным адресам | Защита памяти по ключам защиты | Защита памяти по битам управления |

# Защита памяти по граничным адресам

Реализация этого способа защиты предусматривает выделение для каждой программы определенной области ОП, состоящей из ячеек с последовательными адресами.

Границы области отмечаются фиксированием адресов её начальной (Анач) и конечной ячеек (Акон).




Граничные адреса вводятся в регистры блока защиты памяти (БЗП) операционной системой перед началом выполнения прикладной программы. При выполнении прикладной программы каждый исполнительный адрес с помощью узлов сравнения кодов сравнивается с граничными адресами. По результатам сравнения устанавливается возможность обращения к ОП по поступившему адресу: если он находится в пределах граничных адресов, то разрешается доступ к соответствующей ячейке памяти, в противном случае вырабатывается сигнал «*нарушение защиты памяти*» и происходит *прерывание* выполняемой программы.

# Защита памяти по ключам защиты

Каждому процессу ОС назначает ключ защиты задачи (КЗЗ).

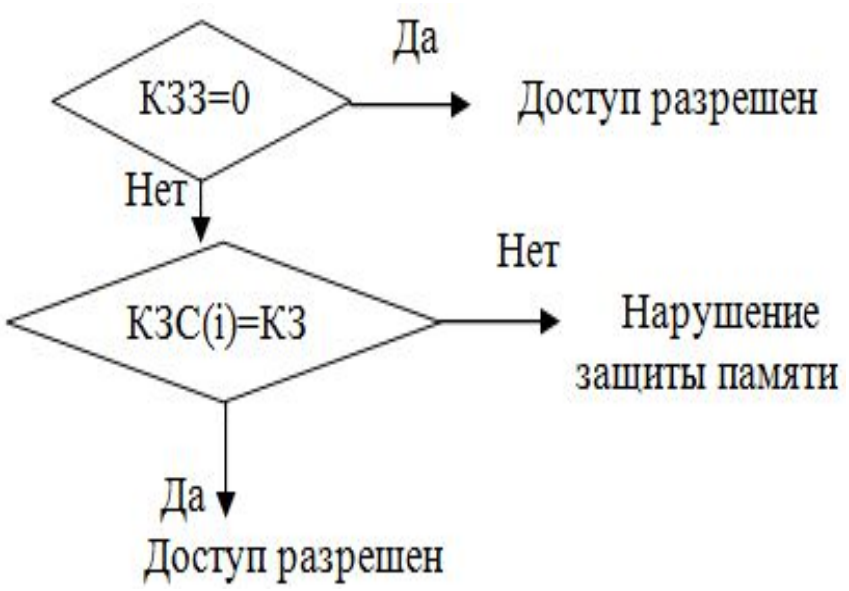
Все выделенные для данного процесса страницы памяти отмечаются одним и тем же ключом защиты страницы (КЗС).

В качестве ключа защиты обычно указывается двоичный номер процесса.

В процессе обращения к ОП производится сравнение ключа выполняемого процесса с ключами защиты соответствующих страниц памяти.

Обращение разрешается только при совпадении сравниваемых ключей.

Исключение: ядро системы (программа Supervisor) имеет ключ защиты равный 0 , получает доступ к *любой* странице оперативной памяти.



# Методы защиты памяти и задачи защиты памяти

| | Защита адресного пространства одного процесса от попыток воспользоваться им другим процессом | Защита адресного пространства операционной системы от попыток воспользоваться им со стороны прикладного процесса | Защита адресного процесса от самого себя |
|---|---|---|---|
| По граничным адресам | + | + | - |
| По ключам защиты | + | + | - |
| По битам управления | - | - | + |

# Защита памяти по битам управления

Каждой ячейке либо другой единице учета оперативной памяти ставится в соответствие ряд управляющих признаков – *битов управления*.

Значение бита, равное 0 соответствует невозможности выполнения операции, значение, равное 1 – разрешению операции.

| Биты управления | Код | Данные |
|---|---|---|
| Бит чтения | 0 / 1 | 0 / 1 |
| Бит модификации | НЗП | 0 / 1 |
| Бит выполнения | 0 / 1 | НЗП |

При попытке выполнить ту или иную операцию с участком оперативной памяти проверяются значения этих битов.

В случае отсутствия разрешения на операцию формируется сигнал «*нарушение защиты памяти*» (НЗП) и происходит *прерывание* выполняемой программы.

# Типы адресов

Для идентификации переменных и команд на разных этапах жизненного цикла программы используются *символьные* имена (метки), *виртуальные* адреса и *физические* адреса


***Символьные*** имена присваивает пользователь при написании программы на алгоритмическом языке или ассемблере

***Виртуальные*** адреса, называемые иногда ***математическими***, вырабатывает транслятор, переводящий программу на машинный язык.
Поскольку во время трансляции в общем случае неизвестно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, обычно считая по умолчанию, что ***начальным адресом***

***Физические*** адреса соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды

# Транслятор


***Транслятор*** — это программа-переводчик.
Она преобразует программу, написанную на одном из языков высокого уровня, в программу, состоящую из машинных команд.

Транслятор обычно выполняет также *диагностику ошибок*, *формирует словари идентификаторов*, выдаёт для *печати тексты программы* и т. д.

Язык, на котором представлена входная программа, называется ***исходным языком***, а сама программа — исходным кодом. Выходной язык называется ***целевым языком*** или объектным кодом.

Трансляторы реализуются в виде ***компиляторов*** или ***интерпретаторов***.

# Редактор связей и Загрузчик


*Редактор связей* – системная программа, выполняющая *связывание* двух или более отдельных оттранслированных программных модулей в один объектный модуль.

*Загрузчик* - это системная программа, выполняющая *загрузку* объектного модуля в физическую оперативную память.

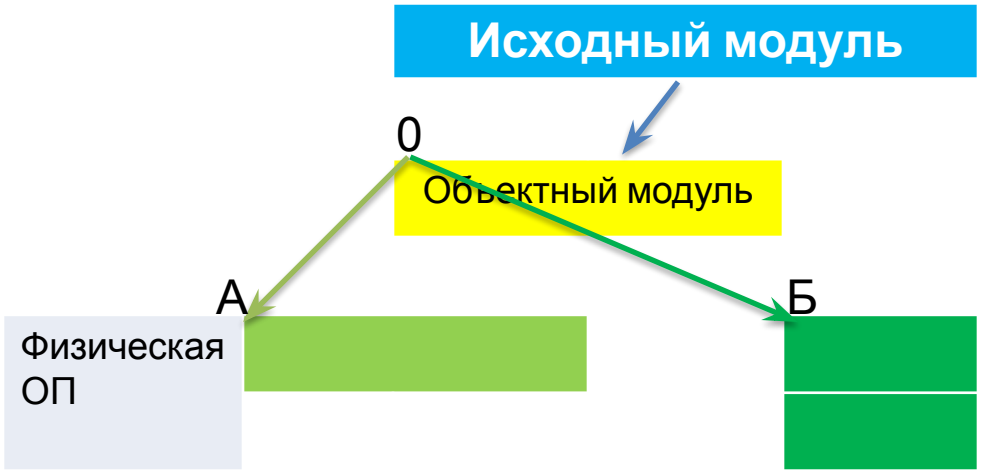
# Виды загрузчиков

# Абсолютный загрузчик

**Абсолютный загрузчик** гарантирует, что загружаемый модуль всегда будет располагаться *в одном и том же месте* физической оперативной памяти.
Следовательно, в модуле, передаваемом для **загрузки**, все обращения должны быть к конкретным или **абсолютным** адресам основной памяти.

# Перемещающий загрузчик

Специальная системная программа, которая на основании имеющихся у нее исходных данных о *начальном адресе физической памяти, в которую предстоит загружать программу*, а также информации, предоставленной транслятором *об адресно-зависимых элементах программы*, выполняет загрузку программы, совмещая ее с заменой виртуальных адресов вычисляемыми физическими адресами.




При этом происходит однократная замена всех виртуальных адресов на физические.

# Динамический загрузчик

Программа загружается в память в *неизмененном виде в виртуальных адресах*, то есть операнды инструкций и адреса переходов имеют те значения, которые выработал транслятор.

В наиболее простом случае, когда виртуальная и физическая память процесса представляют собой единые непрерывные области адресов, операционная система выполняет преобразование виртуальных адресов в физические по следующей схеме:

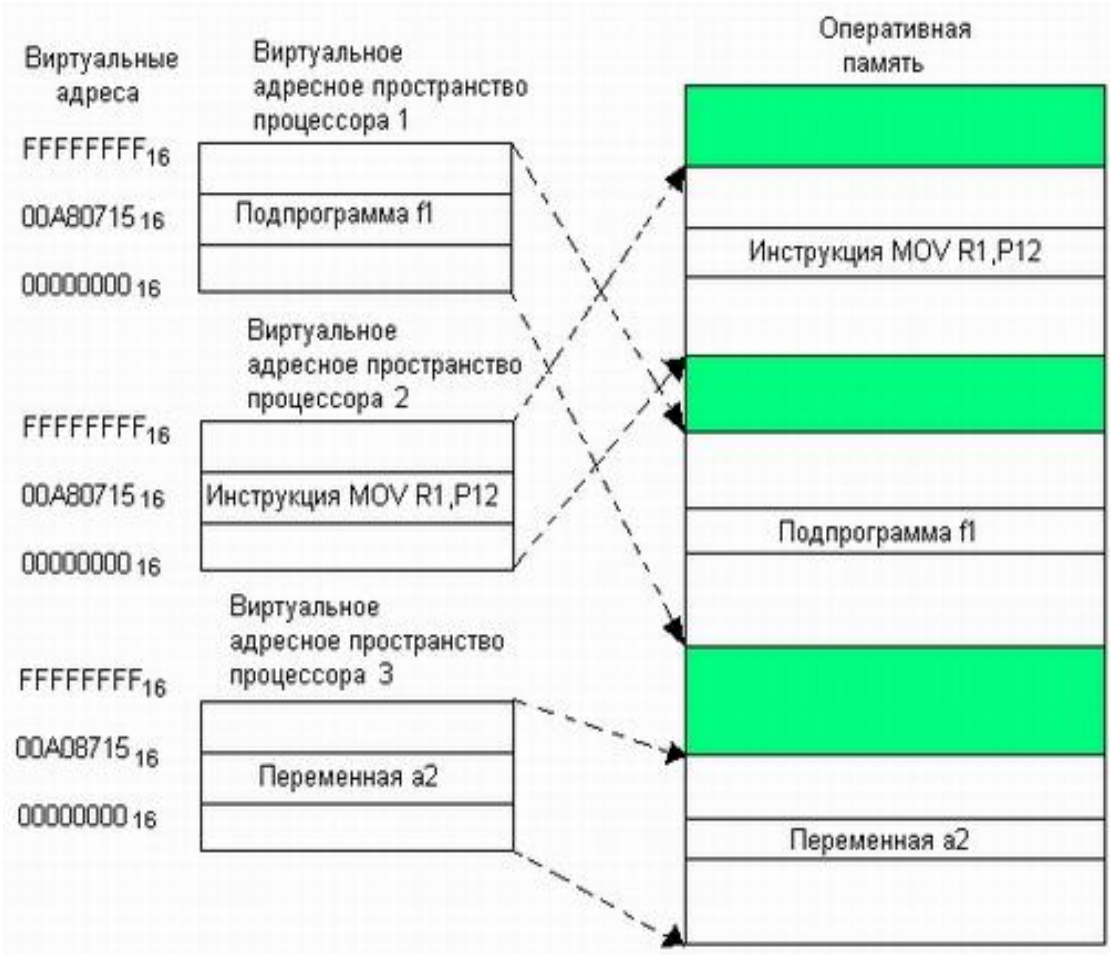
- при загрузке операционная система *фиксирует смещение действительного расположения программного кода* ***относительно начала*** *виртуального адресного пространства*;
- во время выполнения программы *при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический*.

# Подходы к преобразованию виртуальных адресов в физические

# Диапазон возможных адресов виртуального пространства и размер виртуального адресного пространства процесса

**Диапазон возможных адресов виртуального пространства у всех процессов является одним и тем же.**

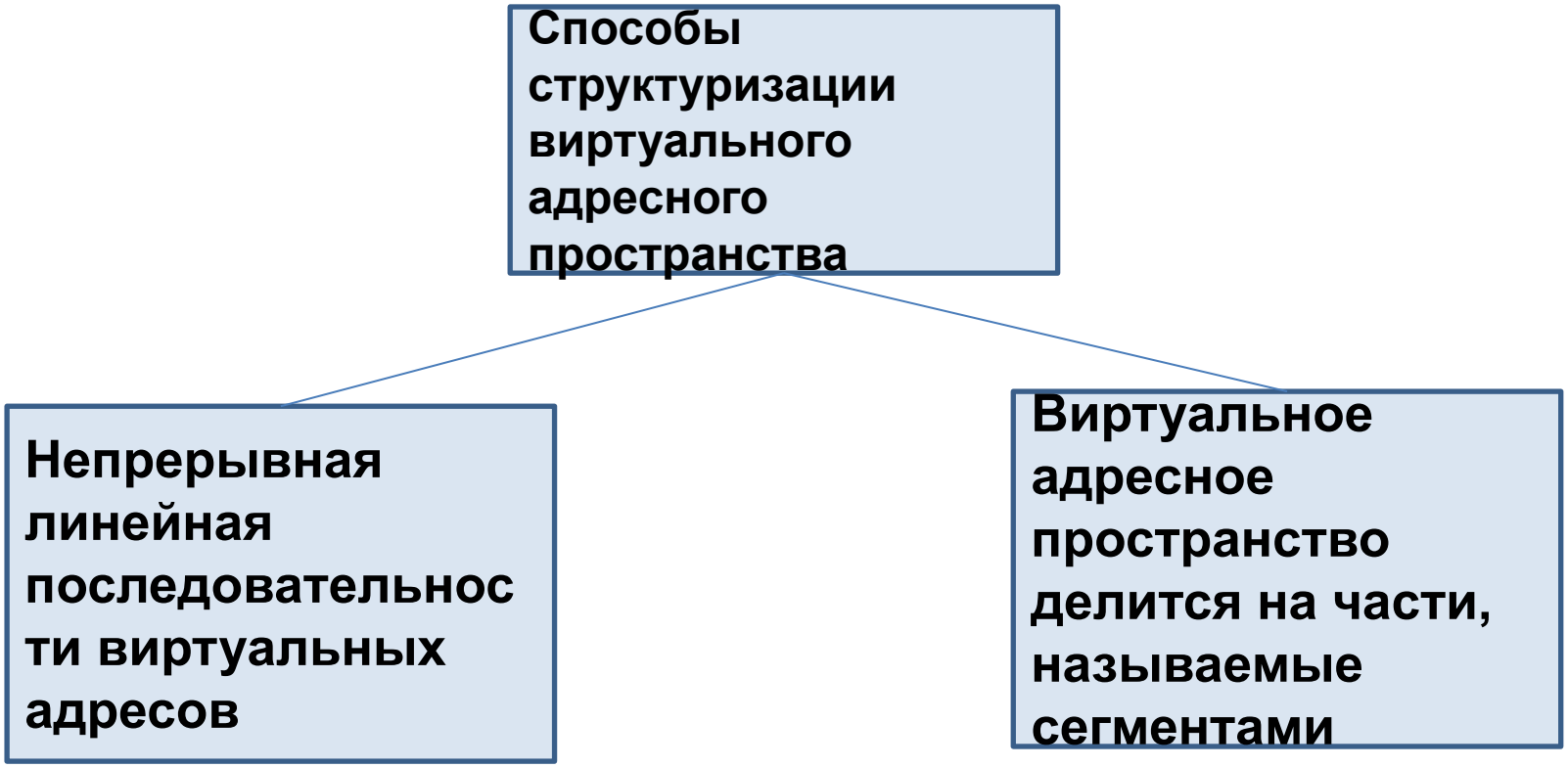
Например, при использовании 32-разрядных виртуальных адресов этот диапазон задается границами $00000000_{16}$ и $FFFFFFFF_{16}$.




**Размер виртуального адресного пространства каждого процесса определяется совокупностью виртуальных адресов процесса.** При этом транслятор присваивает виртуальные адреса переменным и кодам каждой программы независимо от других программ, начиная с адреса $00000000_{16}$.

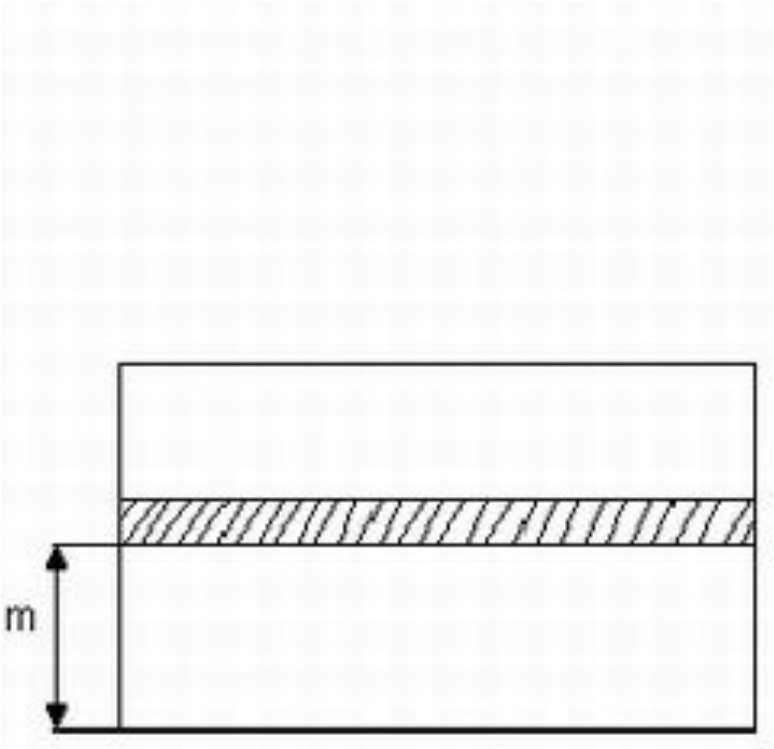
*Совпадение виртуальных адресов* переменных и команд различных процессов *не приводит к конфликтам*, так как в том случае, когда эти переменные одновременно присутствуют в памяти, *операционная система отображает их на разные физические адреса*.

# Способы структуризации виртуального адресного пространства



# Непрерывная линейная последовательности виртуальных адресов

Виртуальное адресное пространство процесса подобно физической памяти представлено в виде *непрерывной линейной последовательности виртуальных адресов*.




Такую структуру адресного пространства называют также *плоской* (flat).

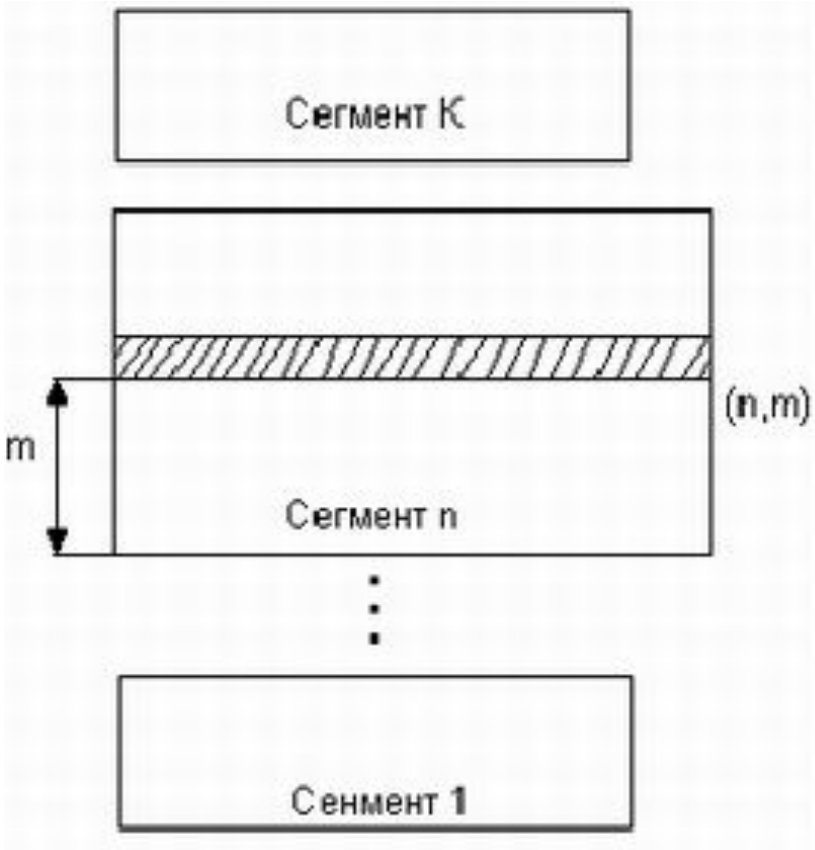
При этом *виртуальным адресом* является единственное число, представляющее собой *смещение* относительно начала виртуального адресного пространства.
Адрес такого типа называют *линейным виртуальным адресом*.

# Сегментированное виртуальное адресное пространство

Виртуальное адресное пространство делится на части, называемые *сегментами*.

Виртуальный адрес представляет собой пару чисел (n, m), где

*n* определяет сегмент,
*m* — смещение внутри сегмента.



# Отображение индивидуальных виртуальных адресных пространств на общую физическую память

Отображение *индивидуальных виртуальных адресных пространств* всех одновременно выполняющихся процессов на общую физическую память является важнейшей задачей многозадачной операционной системы.

При этом ОС отображает на физическую память либо всё виртуальное адресное пространство процесса, либо только определенную его часть.

Процедура преобразования виртуальных адресов в адреса физической ОП должна быть максимально прозрачна для пользователя и программиста.

# Структура виртуального адресного пространства процесса

Обычно виртуальное адресное пространство процесса делится на две непрерывные части: *системную* и *пользовательскую*.

Часть виртуального адресного пространства каждого процесса, отводимая под сегменты ОС, является идентичной для всех процессов.

Поэтому при смене активного процесса заменяется только вторая часть виртуального адресного пространства, содержащая его индивидуальные сегменты, как правило, — коды и данные прикладной программы.

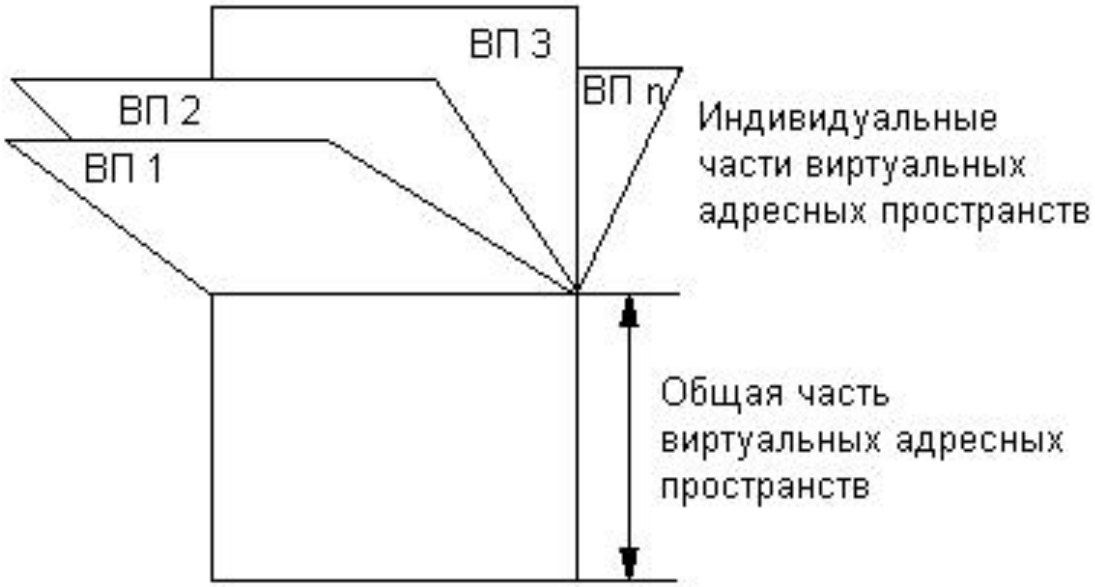



Рисунок взят из книги Гордеева «Операционные системы»

# Еще один взгляд на структуру виртуального адресного пространства процесса

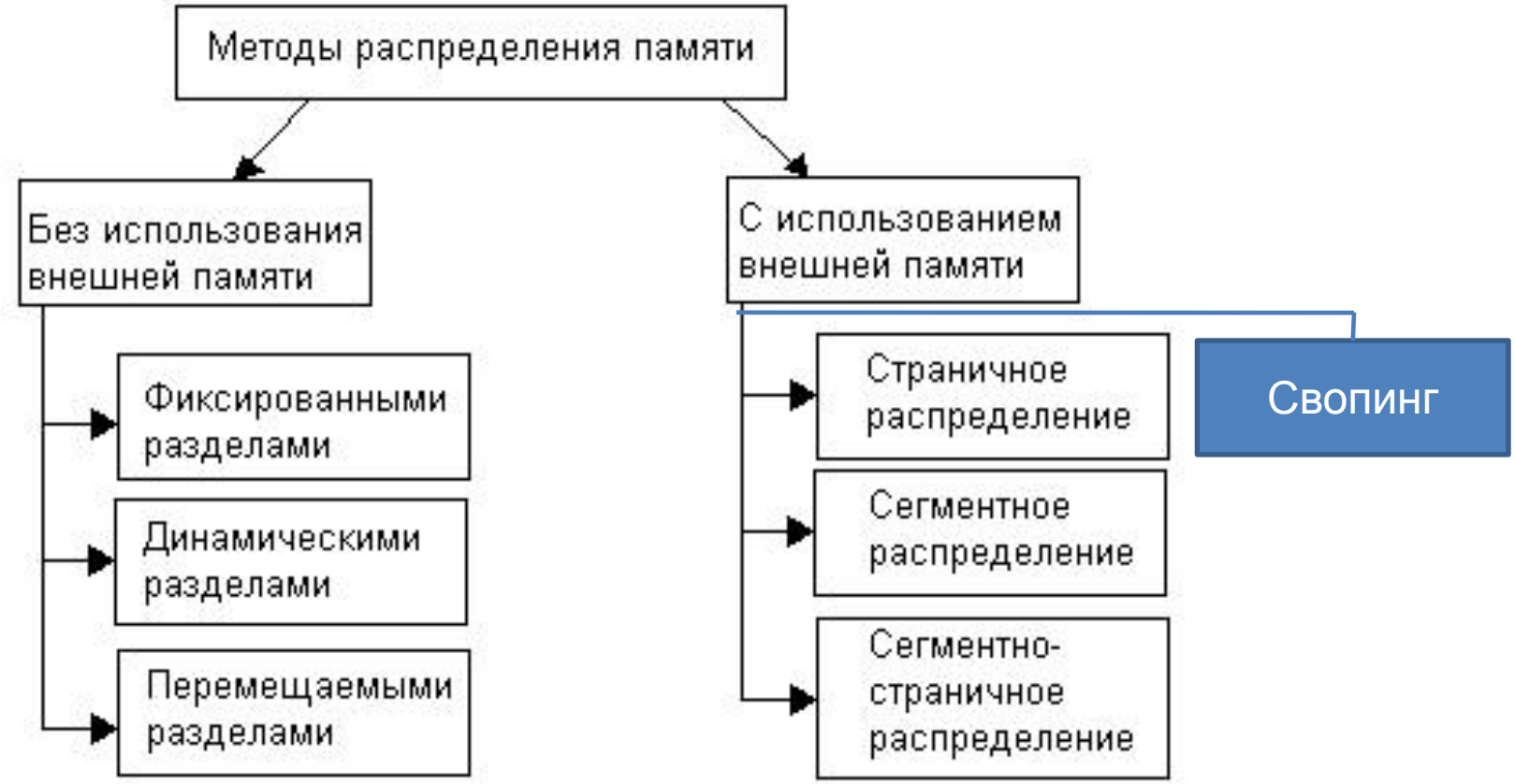
Обычно виртуальное адресное пространство процесса делится на две непрерывные части: *системную* и *пользовательскую*.

Второй способ представления: *код* и *данные.*

| | | |
|---|---|---|
| КОД | Системный | Прикладной |
| ДАННЫЕ | Постоянные | Переменные |

# Алгоритмы распределения памяти

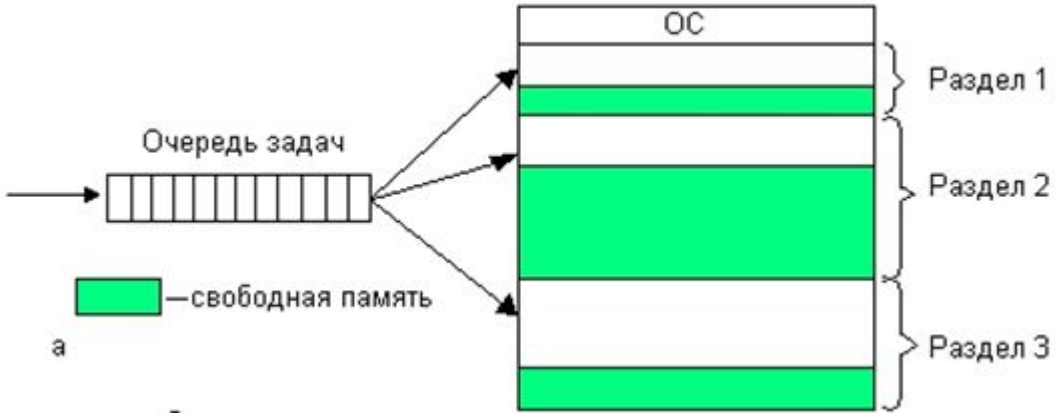
Все алгоритмы распределения памяти разделены на два класса:
- алгоритмы, в которых используется перемещение сегментов процессов между оперативной памятью и диском;
- алгоритмы, в которых внешняя память не привлекается.



# Распределение памяти фиксированными разделами

Простейший способ управления оперативной памятью состоит в том, что память разбивается на несколько областей фиксированной величины, называемых *разделами*.

Такое разбиение может быть выполнено вручную оператором во время старта системы или во время ее установки.
После этого границы разделов не изменяются.



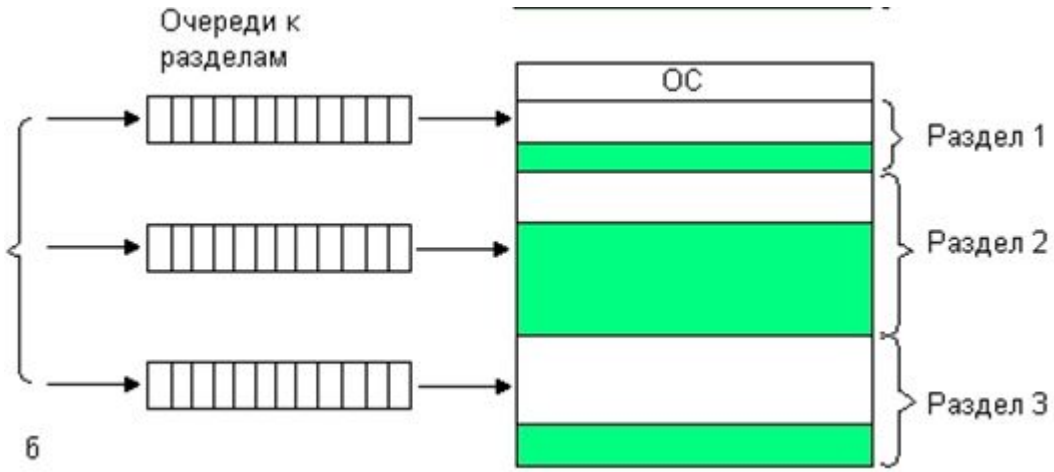




Рисунок взят из книги Гордеева «Операционные системы»

# Достоинства и недостатки распределения памяти фиксированными разделами

Преимущество — простота реализации.

Недостатки:

Первое — жесткость. Так как в каждом разделе может выполняться только один процесс, то уровень мультипрограммирования заранее ограничен числом разделов.

Второе, независимо от размера программы она будет занимать весь раздел.

Третье, разбиение памяти на разделы не позволяет выполнять процессы, программы которых не помещаются ни в один из разделов, но для которых было бы достаточно памяти нескольких разделов.

# Распределение памяти динамическими разделами

Память машины заранее на разделы не делится.
Сначала вся память, отводимая для приложений, свободна.

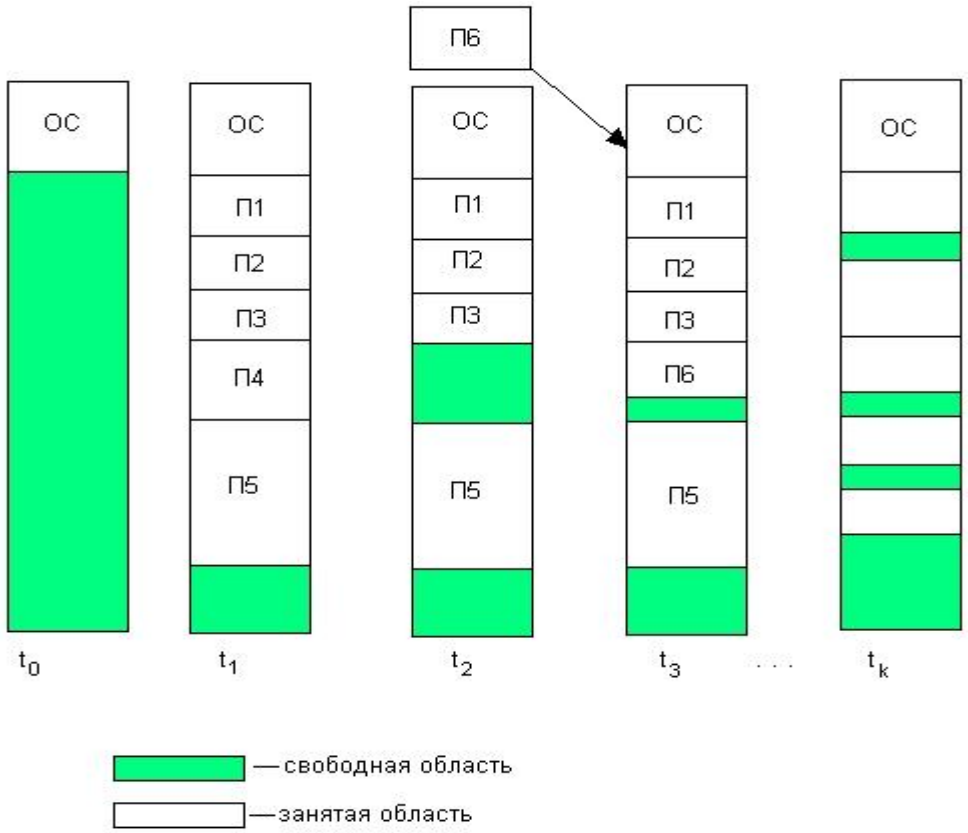
Каждому вновь поступающему на выполнение приложению на этапе создания процесса выделяется необходимая ему память (если достаточный объем памяти отсутствует, то приложение не принимается на выполнение и процесс для него не создается).

По мере развития процессы могут выдавать запросы на дополнительные участки памяти. Если нет ни одного раздела необходимого размера, процесс выдавший запрос, блокируется.

После завершения процесса память освобождается, и на это место может быть загружен другой процесс.

Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.

# Основная проблема распределение памяти динамическими разделами




Методу присущ очень серьезный недостаток — *фрагментация свободной памяти*, то есть наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов).

Размер этих фрагментов настолько мал, что они не могут быть использованы для удовлетворения какого-либо запроса на выделение памяти, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

Рисунок взят из книги Гордеева «Операционные системы»

# Перемещаемые разделы. Сжатие.

Решение проблемы фрагментации свободной памяти – ***сжатие***.

Процедура сжатия состоит в том, что ОС перемещает содержимое разделов из одного места памяти в другое, объединяя все свободные участки памяти в единое целое и корректируя таблицы свободных и занятых областей.

Сжатие может выполняться:

- либо при каждом завершении процесса;
- либо только тогда, когда для удовлетворения запроса на память от процесса при наличие достаточного объема свободной памяти нет свободного раздела достаточного размера.

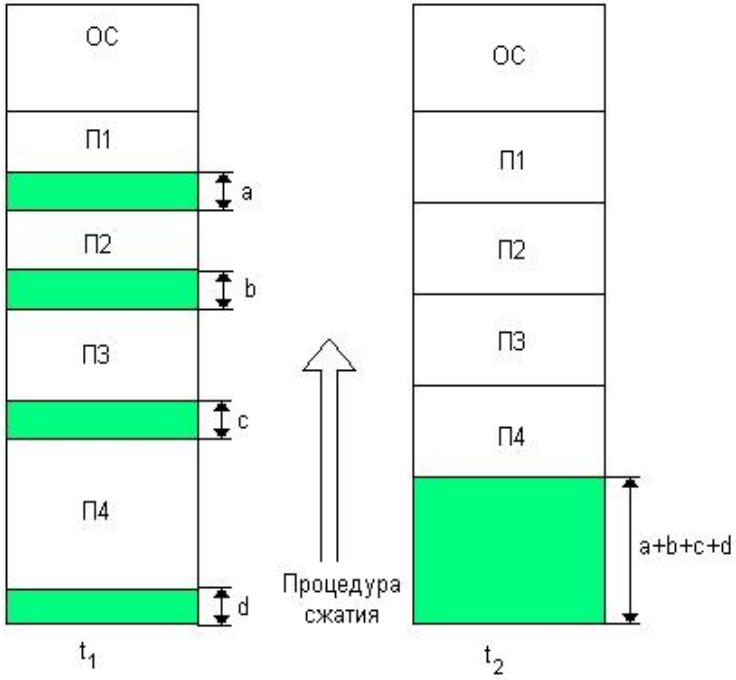
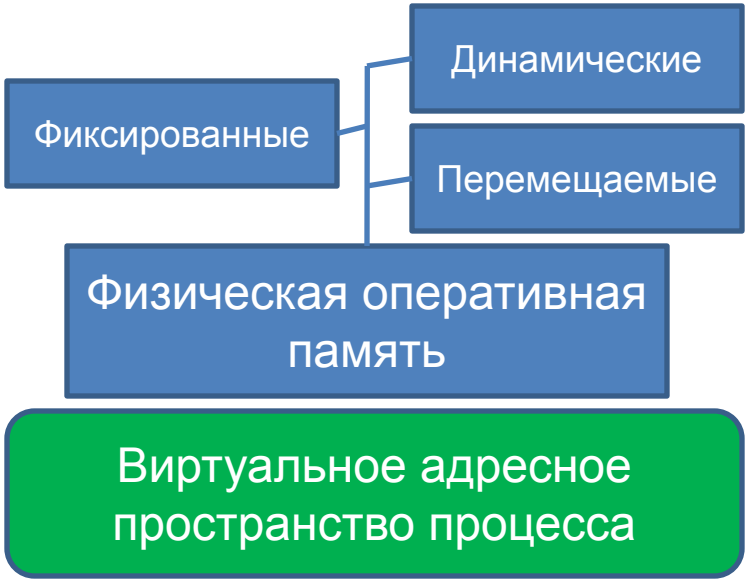



Рисунок взят из книги Гордеева «Операционные системы»

В первом случае требуется меньше вычислительной работы при корректировке таблиц свободных и занятых областей,
во втором — реже выполняется процедура сжатия.

# Программы с оверлейной структурой



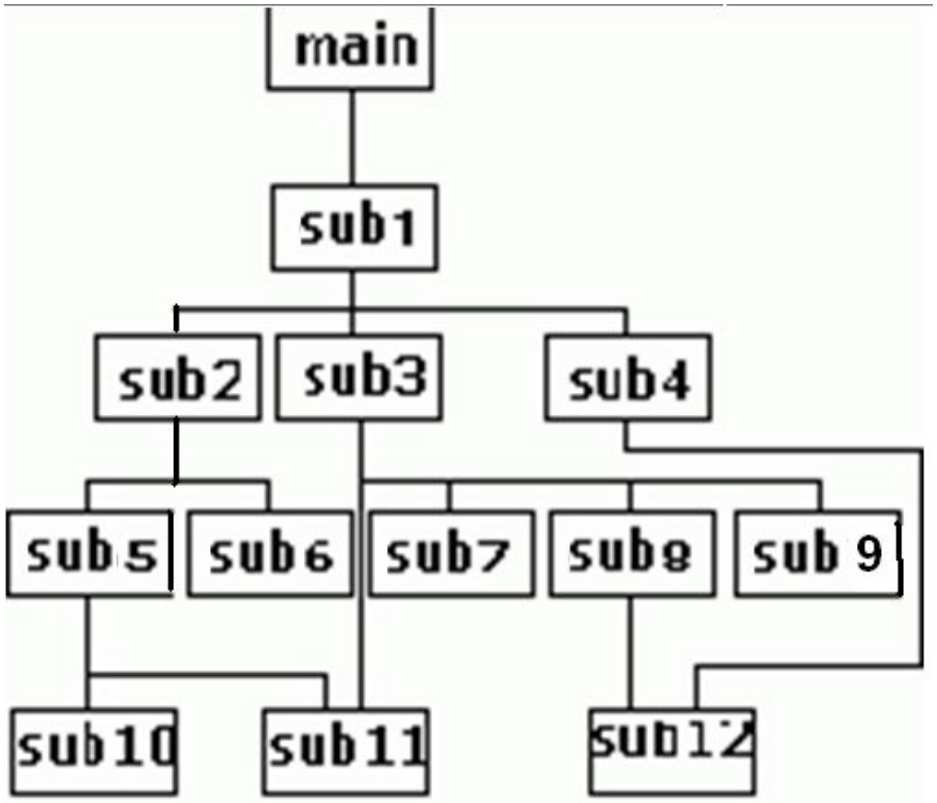

Размер виртуального адресного пространства процесса может быть больше, чем размер самого большого раздела или больше всей доступной пользовательским приложениям памяти.

Новая технология программирования - организация структуры программы с перекрытием или *оверлей* (overlay).

Основная идея – все оверлейные модули в готовом к работе виде хранятся на диске, а в оперативной памяти в каждый момент находится лишь один активный модуль и, возможно, небольшое число неактивных.

Этот подход позволяет *уменьшить объем требуемой памяти*, но увеличивает общее время работы приложения, так как приходится периодически загружать с диска оверлейные фрагменты.

# Программы с оверлейной структурой




За загрузку необходимых оверлейных модулей отвечает программист!

Тщательное проектирование *оверлейной структуры* отнимает много времени и требует знания устройства программы, её кода, данных и языка описания *оверлейной структуры*.

По этой причине применение технологии оверлеев было ограничено.

Проблема оверлейных *сегментов*, контролируемых программистом, отпадает благодаря появлению технологий виртуализации памяти.

# Мультипрограммирование и проблема нехватки оперативной памяти

Одним из критериев эффективности вычислительной системы является её производительность, то есть число задач, решаемых в единицу времени.

Чем больше производительность, тем эффективнее система.

Большое количество задач требует больших объемов оперативной памяти.

При этом в мультипрограммном режиме помимо активного процесса, имеются также приостановленные процессы, ожидающие завершения ввода-вывода или освобождения ресурсов, а также процессы в состоянии готовности, стоящие в очереди к процессору.

Но эти процессы занимают оперативную память, не позволяя загрузить новые процессы.

В условиях, когда для обеспечения приемлемого уровня мультипрограммирования имеющейся памяти оказывается недостаточно, был предложен метод организации вычислительного процесса, при котором образы некоторых процессов целиком или частично временно выгружаются на диск.

# Виртуализация памяти – способ решения проблемы нехватки оперативной памяти

Образы неактивных процессов могут быть временно, до следующего цикла активности, выгружены на диск.

Операционная система «знает» о существовании таких процессов и учитывает их при распределении процессорного времени и других ресурсов.

К моменту, когда подходит очередь выполнения выгруженного процесса, его образ возвращается с диска в оперативную память. Если при этом оказывается, что свободного места в оперативной памяти не хватает, то на диск выгружается другой процесс.

Такая подмена оперативной памяти дисковой памятью называется ***виртуализацией оперативной памяти***.

Она позволяет повысить уровень мультипрограммирования.

Виртуализация оперативной памяти осуществляется совокупностью программных модулей ОС и аппаратных схем процессора.

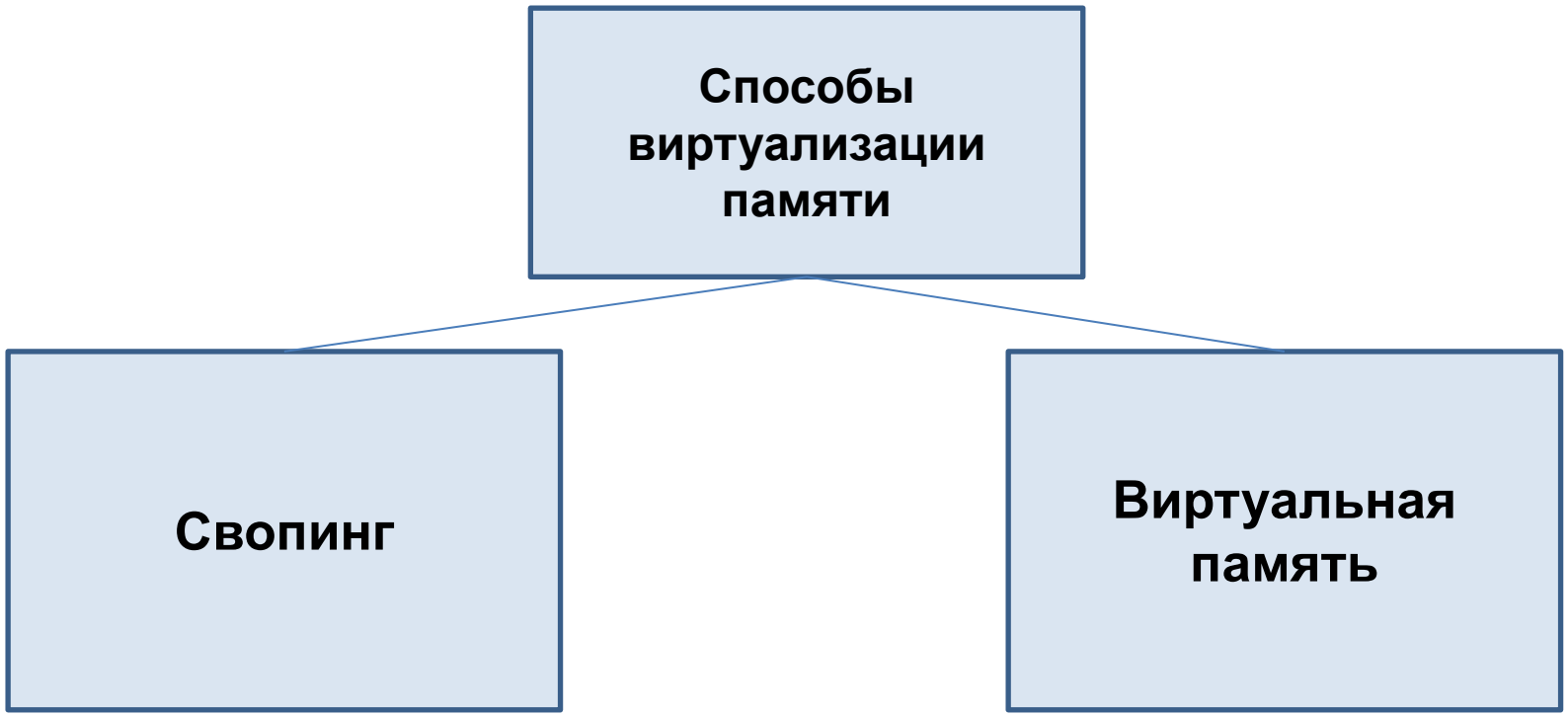
**ВСПОМНИМ:**

***Виртуальным*** *называется ресурс, который пользователю или пользовательской программе представляется обладающим свойствами, которыми он, в действительности, не обладает.*

# Задачи виртуализация памяти

- размещение данных в запоминающих устройствах различного типа;
- выбор образов процессов или их частей для перемещения из оперативной памяти на диск и обратно;
- перемещение по мере необходимости данных между памятью и диском;
- преобразование виртуальных адресов в физические.

# Способы виртуализации памяти




*свопинг* (swapping) — образы процессов выгружаются на диск и возвращаются в оперативную память **целиком**

*виртуальная память* (virtual memory) — между оперативной памятью и диском перемещаются **части** (сегменты, страницы и т. п.) **образов процессов**.

# Свопинг и его недостатки

*Проблема избыточности*:
когда ОС решает активизировать процесс, для его выполнения, как правило, не требуется загружать в оперативную память все его сегменты полностью — достаточно загрузить небольшую часть кодового сегмента с подлежащей выполнению инструкцией и частью сегментов данных, с которыми работает эта инструкция, а также отвести место под сегмент стека. Аналогично при освобождении памяти для загрузки нового процесса очень часто вовсе не требуется выгружать другой процесс на диск целиком, достаточно вытеснить на диск только часть его образа.
Перемещение *избыточной* информации *замедляет работу операционной системы*, а также приводит к неэффективному использованию памяти.

Кроме того, системы, поддерживающие свопинг, имеют еще один очень существенный недостаток: они *не способны загрузить для выполнения процесс, виртуальное адресное пространство которого превышает имеющуюся в наличии свободную память*.

# Виртуальная память - совершенный механизм виртуализации

В настоящее время все множество реализаций *виртуальной памяти* может быть представлено тремя классами:

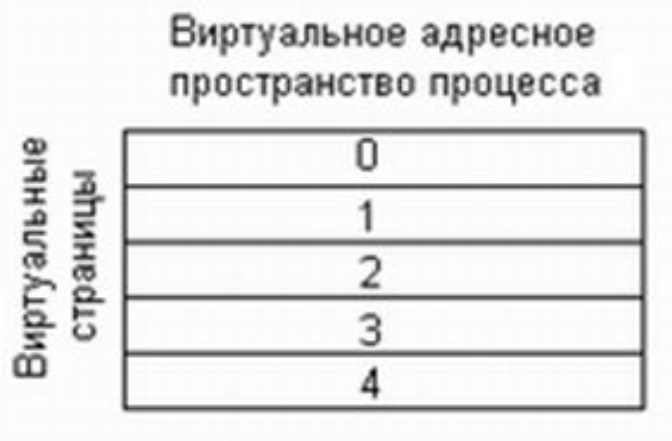
- *Страничная виртуальная память* организует перемещение данных между памятью и диском страницами — частями виртуального адресного пространства, фиксированного и сравнительно небольшого размера.
- *Сегментная* виртуальная память предусматривает перемещение данных сегментами — частями виртуального адресного пространства произвольного размера, полученными с учетом смыслового значения данных.
- *Сегментно-страничная виртуальная память* использует двухуровневое деление: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы. Единицей перемещения данных здесь является страница. Этот способ управления памятью объединяет в себе элементы обоих предыдущих подходов.

# Ключевая проблема механизма виртуальной памяти

Ключевой проблемой механизма виртуальной памяти, возникающей в результате многократного изменения местоположения в оперативной памяти образов процессов или их частей, является *преобразование виртуальных адресов в физические*.

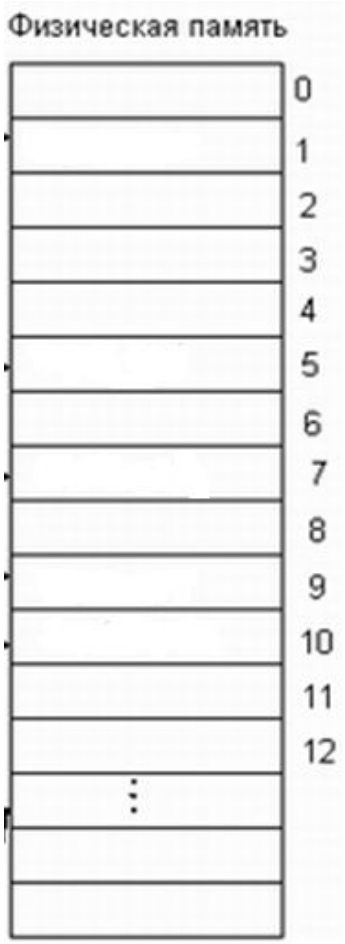
# Страничная организация виртуальной памяти.

Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые *виртуальными страницами*.
Размер страницы выбирается кратным степени числа 2 ( 512, 1024, 4096 байт и т. д).




В общем случае размер виртуального адресного пространства процесса не кратен размеру страницы, поэтому *последняя страница каждого процесса дополняется фиктивной областью*.

# Страничная организация виртуальной памяти



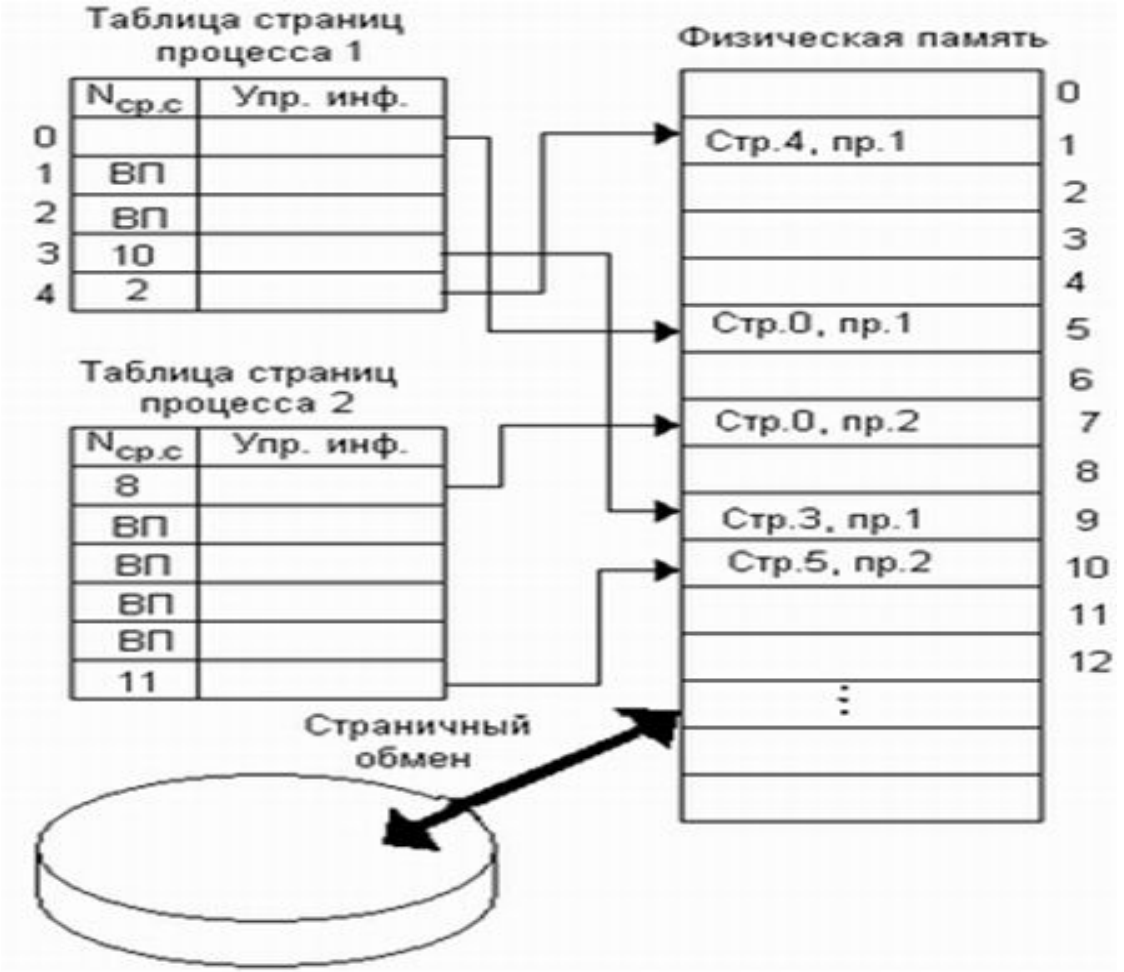

Вся физическая оперативная память машины также делится на части такого же размера, называемые *физическими страницами*.

# Страничная организация виртуальной памяти

При создании процесса ОС загружает в оперативную память несколько его виртуальных страниц (начальные страницы кодового сегмента и сегмента данных).

Копия всего виртуального адресного пространства процесса находится на диске.

Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах.




Для каждого процесса операционная система создает таблицу страниц — информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

# Дескриптор страницы

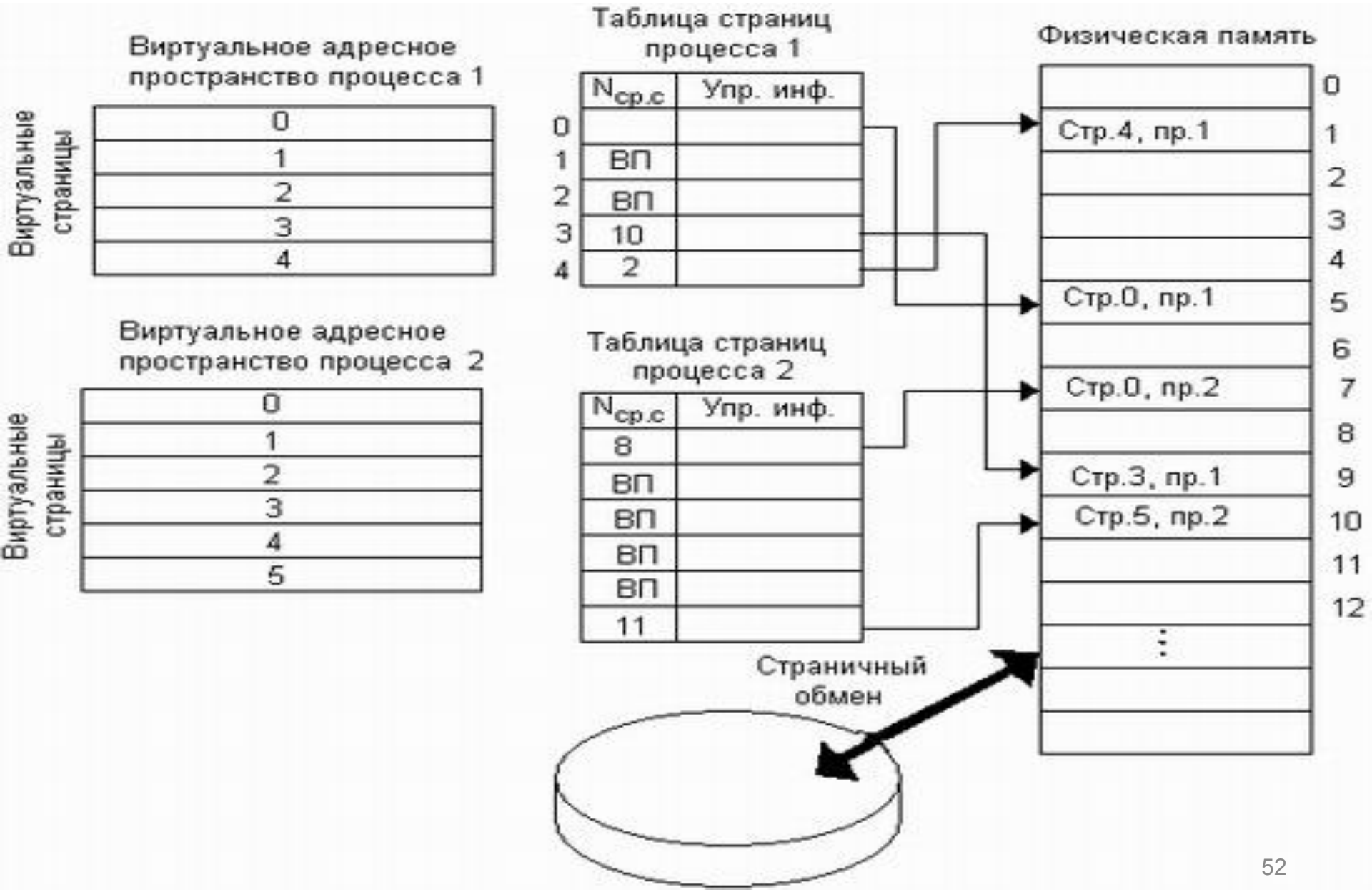
Таблица страниц

| | $N_{ср.с}$ | Упр. инф. |
|---|---|---|
| 0 | | |
| 1 | ВП | |
| 2 | ВП | |
| 3 | 10 | |
| 4 | 2 | |

Строка таблицы, называемая *дескриптором страницы*, включает следующую информацию:

- *номер физической страницы*, в которую загружена данная виртуальная страница;
- *признак присутствия*, устанавливаемый в единицу, если виртуальная страница находится в оперативной памяти;
- *признак модификации* страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;
- *признак обращения* к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

# Страничная организация виртуальной памяти. Общая схема



# Первое базовое свойство страничной организации памяти

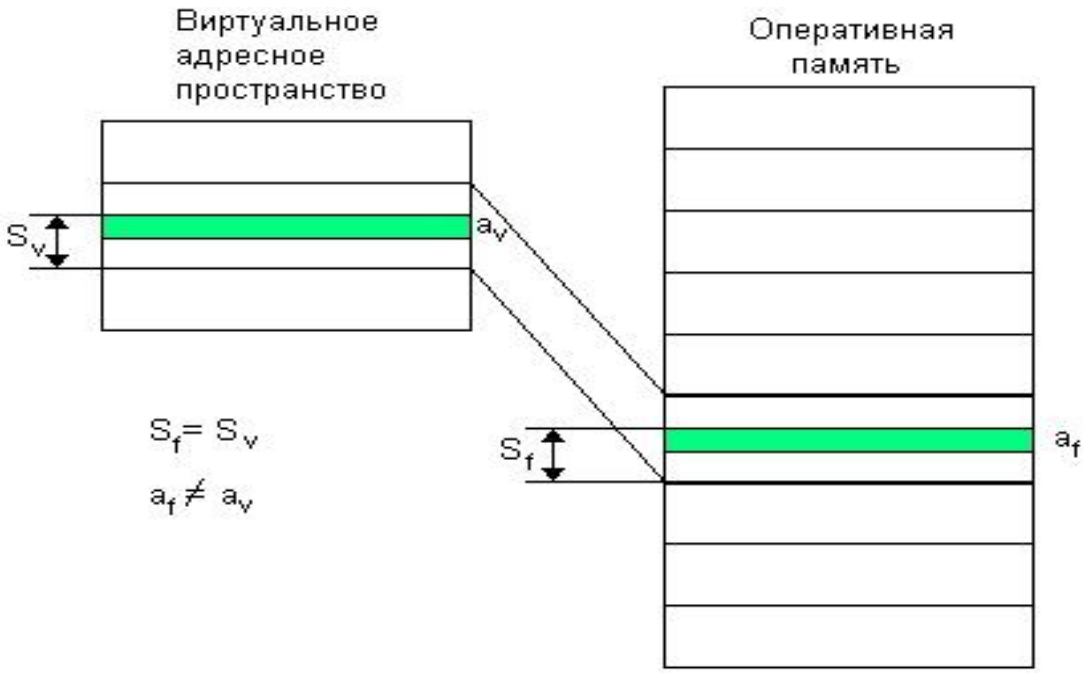
Так как размер страницы выбирается равным степени числа 2 ($2^k$), то *смещение s может быть получено простым отделением k младших разрядов в двоичной записи адреса*, а оставшиеся старшие разряды адреса представляют собой двоичную запись номера страницы (при этом неважно, является страница виртуальной или физической).

Например, если размер страницы 1 Кбайт ($2^{10}$), то из двоичной записи адреса 101 000 111 $001_2$ можно определить, что он принадлежит странице, номер которой в двоичном выражении равен $10_2$ и смещен относительно ее начала на 1 000 111 $001_2$ байт.

# Второе базовое свойство страничной организации памяти

В пределах страницы непрерывная последовательность виртуальных адресов однозначно отображается в непрерывную последовательность физических адресов.

Следовательно, *смещения в виртуальном и физическом адресах $s_v$ и $s_f$ равны между собой*.



# Схема преобразования виртуального адреса в физический при страничной организации памяти

Младшие разряды физического адреса, соответствующие смещению, получаются *переносом* такого же количества младших разрядов из виртуального адреса.

Старшие разряды физического адреса, соответствующие *номеру физической страницы*, определяются из таблицы страниц, в которой указывается соответствие виртуальных и физических страниц.

L - длина записи таблицы страниц.

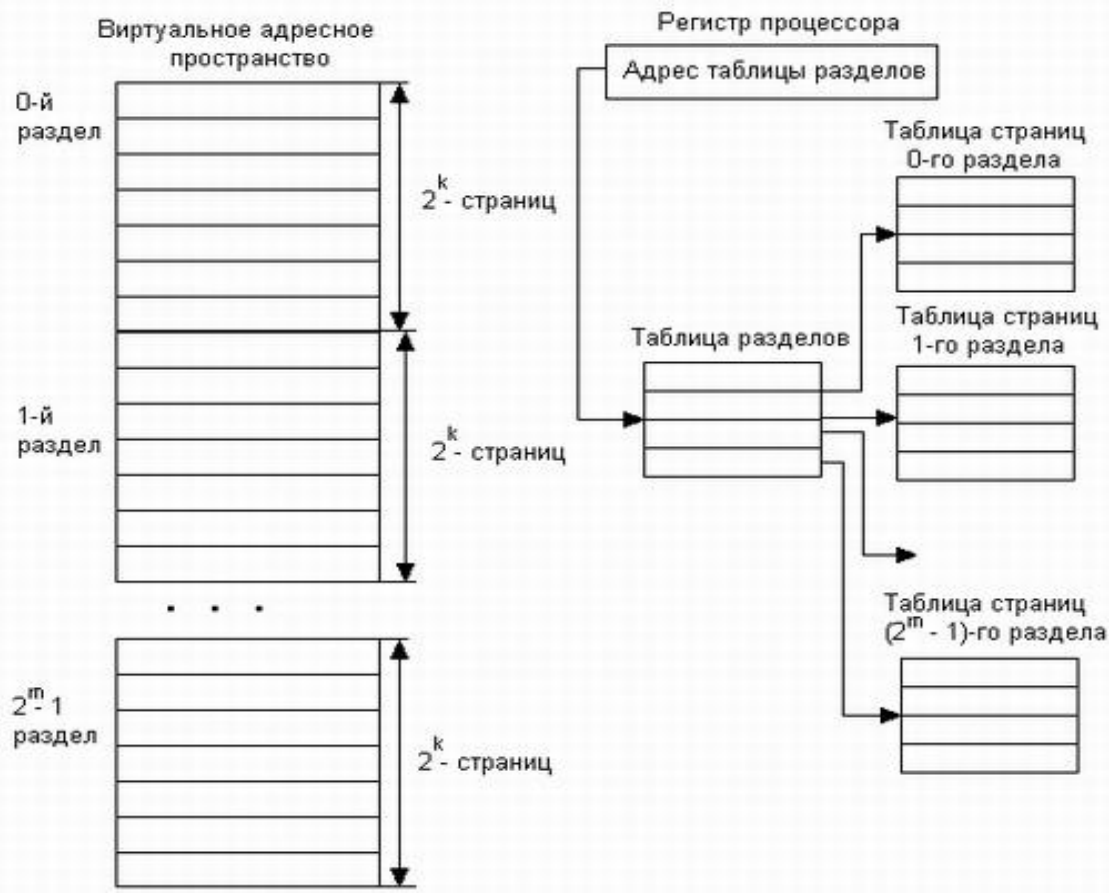
# Страничная организация памяти при больших размерах виртуального адресного пространства

Размер страницы влияет на количество записей в таблицах страниц: чем меньше страница, тем более объемными являются таблицы страниц процессов и тем больше места они занимают в памяти.

Учитывая, что в современных процессорах максимальный объем виртуального адресного пространства процесса, как правило, не меньше 4 Гбайт ($2^{32}$), то при размере страницы 4 Кбайт ($2^{12}$) и длине записи 4 байта для хранения таблицы страниц может потребоваться 4 Мбайт памяти или 1024 страницы!

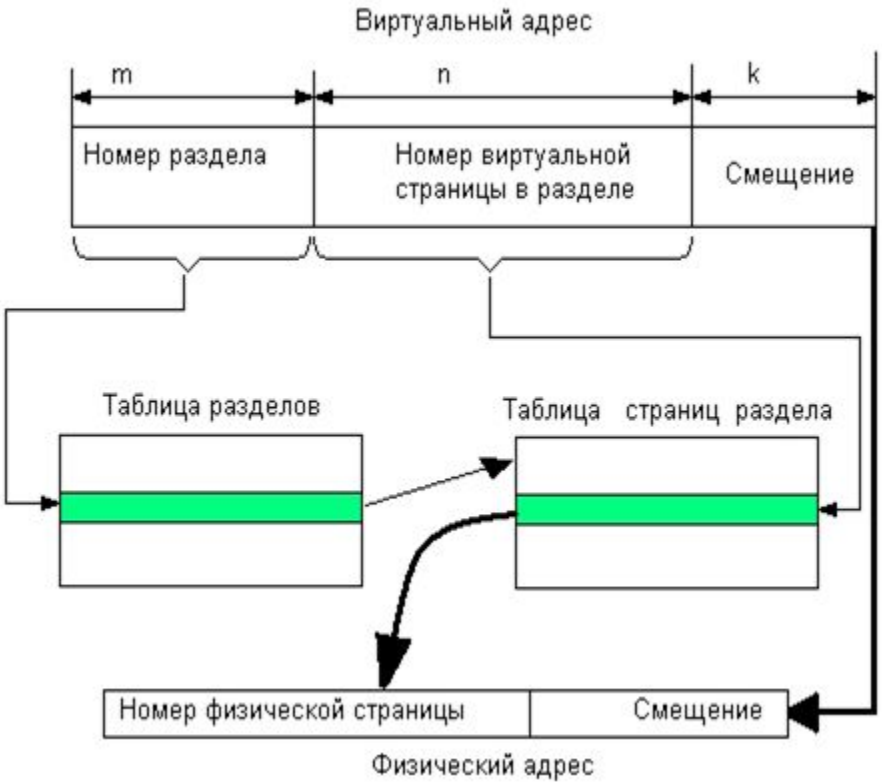
# Страничная организация памяти с использованием разделов

Выходом в такой ситуации является хранение в памяти только той *части таблицы страниц, которая активно используется в данный период времени* — так как сама таблица страниц хранится в таких же страницах физической памяти, что и описываемые ею страницы, то принципиально возможно временно вытеснять часть таблицы страниц из оперативной памяти.




Для *каждого раздела* строится собственная *таблица страниц*.
Количество дескрипторов в таблице и их размер подбираются так, чтобы *объем таблицы оказался равным объему страницы*.
Каждая таблица страниц описывается дескриптором, структура которого полностью совпадает со структурой дескриптора обычной страницы.
Эти дескрипторы сведены в *таблицу разделов*, называемую также *каталогом страниц*.

# Схема преобразования виртуального адреса в физический при страничной организации памяти с использованием разделов




1. Путем отбрасывания *k+n* младших разрядов в виртуальном адресе определяется *номер раздела*, к которому принадлежит данный виртуальный адрес.
2. По этому номеру из таблицы разделов извлекается *дескриптор соответствующей таблицы страниц*. Проверяется, находится ли данная таблица страниц в памяти. Если нет, то система загружает нужную страницу с диска.
3. Далее из этой таблицы страниц извлекается *дескриптор виртуальной страницы*, номер которой содержится в средних *n* разрядах преобразуемого виртуального адреса. Снова выполняется проверка наличия данной страницы в памяти и при необходимости ее загрузка.
4. Из дескриптора определяется *номер (базовый адрес) физической страницы*, в которую загружена данная виртуальная страница. К номеру физической страницы *пристыковывается смещение*, взятое из k младших разрядов виртуального адреса.

В результате получается искомый физический адрес.

# Недостатки страничной организации виртуальной памяти

При страничной организации виртуальное адресное пространство процесса делится на равные части *механически*, *без учета смыслового значения данных*.

В одной странице могут оказаться и коды команд, и инициализируемые переменные, и массив исходных данных программы.

# Сегментная организация памяти

Виртуальное адресное пространство процесса делится на части произвольного размера — ***сегменты***.

**Сегмент – часть виртуального адресного пространства, размер которой определяется с учетом смыслового значения содержащейся в ней информации.**

Отдельный сегмент может представлять собой подпрограмму, массив данных и т. п.

Деление виртуального адресного пространства на сегменты осуществляется компилятором на основе указаний программиста или по умолчанию, в соответствии с принятыми в системе соглашениями

Сегменты не упорядочиваются друг относительно друга,
так что общего для сегментов линейного виртуального адреса не существует,
виртуальный адрес задается парой чисел:
**номером сегмента** и **смещением внутри сегмента**.

# О размерах сегмента и виртуального адресного пространства

*Максимальный размер сегмента* определяется *разрядностью виртуального адреса*

Для 32-разрядных процессоров максимальный размер сегмента равен $2^{32}$ байтов или 4 Гбайт.

Максимально возможное *виртуальное адресное пространство процесса* представляет собой набор из N виртуальных сегментов, каждый размером по 4 Гбайт.

Для 32-разрядных процессоров в каждом сегменте виртуальные адреса находятся в диапазоне от $00000000_{16}$ до $FFFFFFFF_{16}$

# Таблица сегментов

На этапе создания процесса во время загрузки его образа в оперативную память система создает таблицу сегментов процесса, в которой для каждого сегмента указывается:

- базовый физический адрес сегмента в оперативной памяти;
- размер сегмента;
- права доступа к сегменту;
- признаки модификации, присутствия и обращения к данному сегменту, а также некоторая другая информация.

# Механизм сегментной виртуальной памяти

При загрузке процесса в оперативную память помещается только *часть его сегментов*, **полная копия** виртуального адресного пространства **находится на дисковой памяти**.

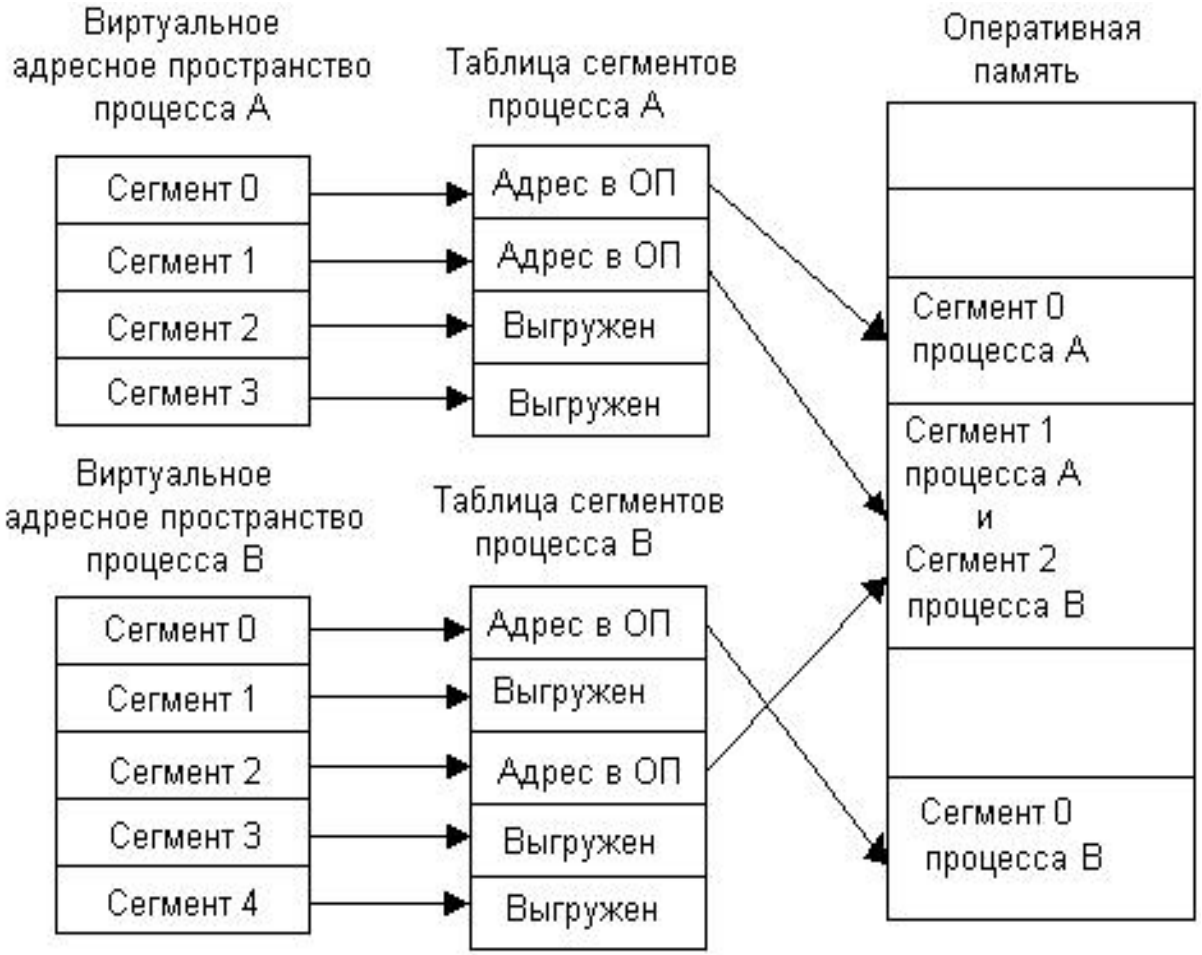
Для каждого загружаемого сегмента операционная система подыскивает *непрерывный участок свободной памяти достаточного размера*.

Смежные в виртуальной памяти сегменты одного процесса могут занимать в оперативной памяти несмежные участки.

Если во время выполнения процесса происходит обращение по виртуальному адресу, относящемуся к сегменту, который в данный момент отсутствует в памяти, то происходит прерывание. ОС приостанавливает активный процесс, запускает на выполнение следующий процесс из очереди, а параллельно организует загрузку нужного сегмента с диска. При отсутствии в памяти места, необходимого для загрузки сегмента, операционная система выбирает сегмент на выгрузку, при этом она использует критерии, основанные на признаках модификации, присутствия и обращения.

# Механизм сегментной виртуальной памяти

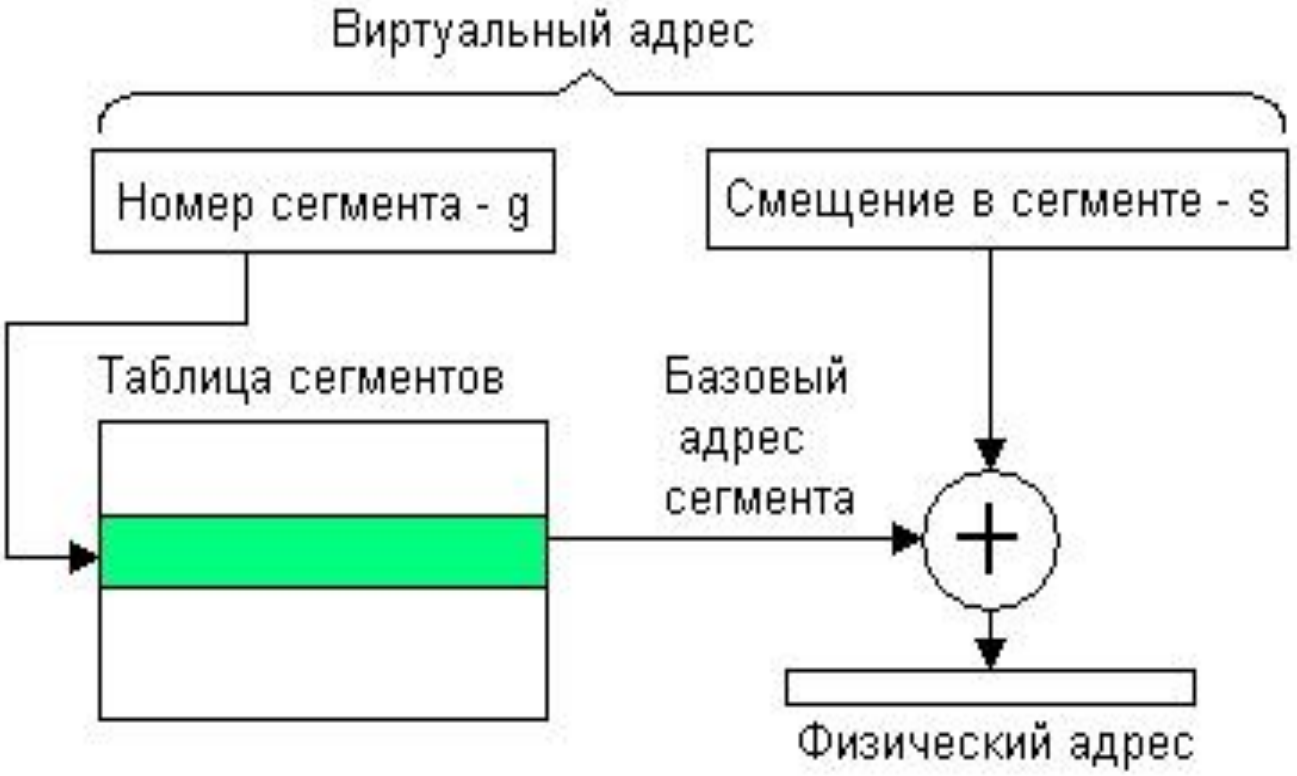
Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который данный сегмент загружается в единственном экземпляре.



# Механизмы преобразования адресов

Виртуальный адрес при сегментной организации памяти может быть представлен парой (g, s), где g — номер сегмента, а s — смещение в сегменте.

Физический адрес получается путем **сложения** базового адреса сегмента, который определяется по номеру сегмента g из таблицы сегментов и смещения s



# Недостатки сегментной организации виртуальной памяти

1. Использование операции сложения вместо конкатенации замедляет процедуру преобразования виртуального адреса в физический по сравнению со страничной организацией.
2. Избыточность. При сегментной организации единицей перемещения между памятью и диском является сегмент, имеющий в общем случае объем больший, чем страница. Однако во многих случаях для работы программы вовсе не требуется загружать весь сегмент целиком, достаточно было бы одной или двух страниц.
3. Главный недостаток — фрагментация.

# Достоинство сегментной организации виртуальной памяти

Возможность задания *дифференцированных прав доступа* к сегментам процесса.
Например, один сегмент данных, содержащий исходную информацию для приложения, может иметь права доступа «только чтение», а сегмент данных, представляющий результаты, — «чтение и запись».
Это свойство дает принципиальное преимущество сегментной модели памяти над страничной.

# Сегментно-страничная организации виртуальной памяти

- Так же как и при сегментной организации памяти, *виртуальное адресное пространство процесса разделено на сегменты*. Это позволяет определять разные права доступа к разным частям кодов и данных программы.
- *Перемещение данных между памятью и диском осуществляется не сегментами, а страницами*. Для этого каждый виртуальный сегмент и физическая память делятся на страницы равного размера, что позволяет более эффективно использовать память, сократив до минимума фрагментацию.

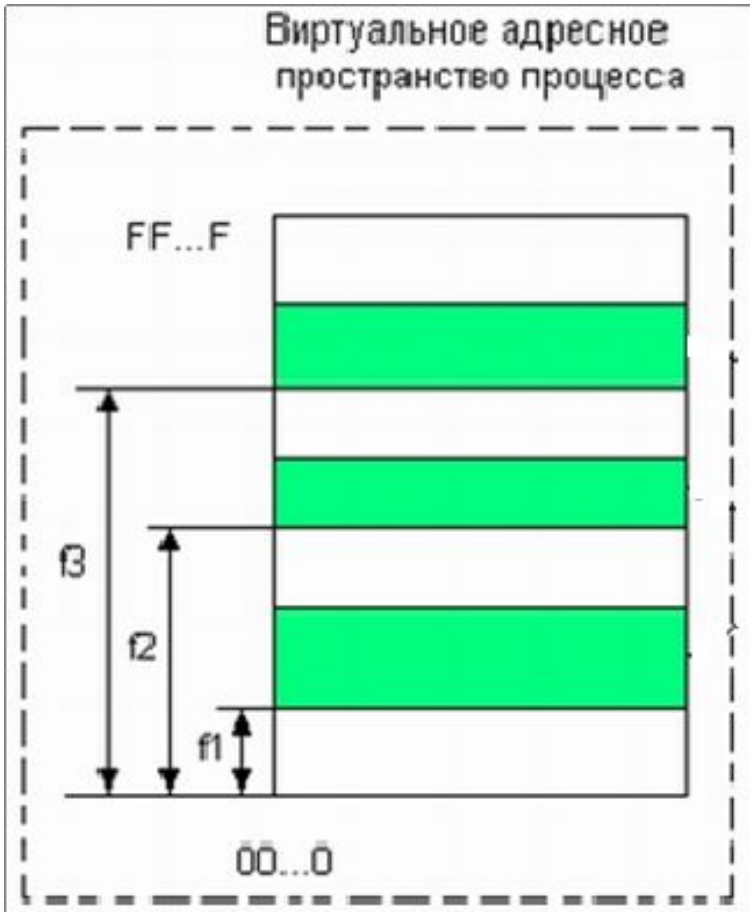
# Важное отличие от сегментной организации виртуальной памяти

*Все виртуальные сегменты образуют одно непрерывное линейное виртуальное адресное пространство*.

Координаты байта в виртуальном адресном пространстве при сегментно-страничной организации можно задать двумя способами.

Во-первых, *линейным виртуальным адресом*, который равен сдвигу данного байта относительно границы общего линейного виртуального пространства,

во-вторых, *парой чисел*, одно из которых является *номером сегмента*, а другое — *смещением* относительно начала сегмента.

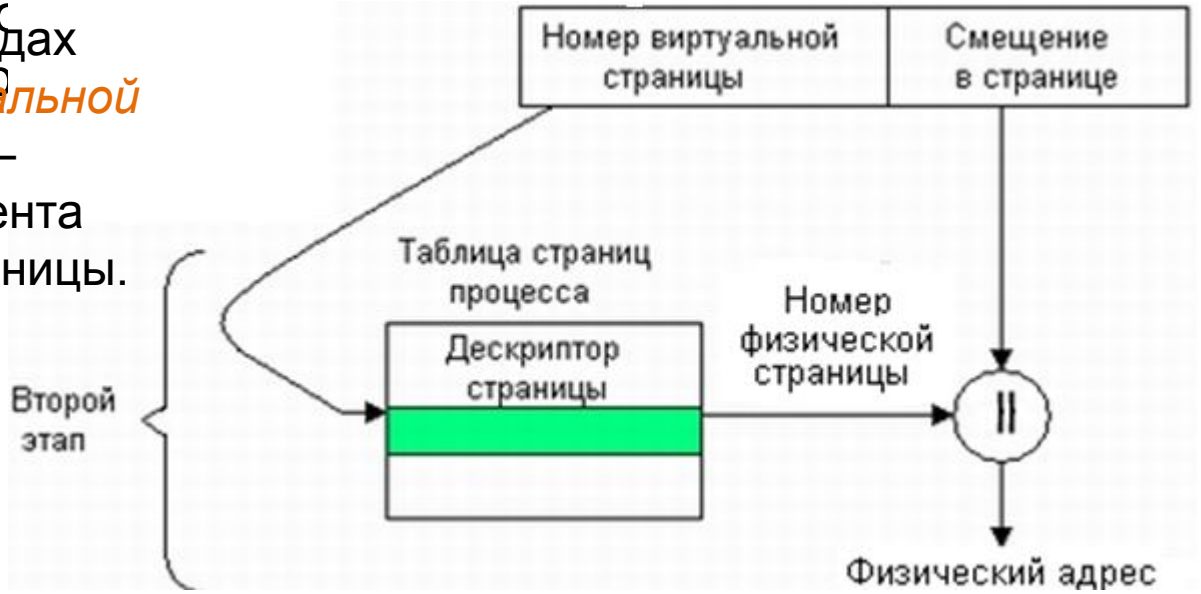


# Преобразование виртуального адреса в физический

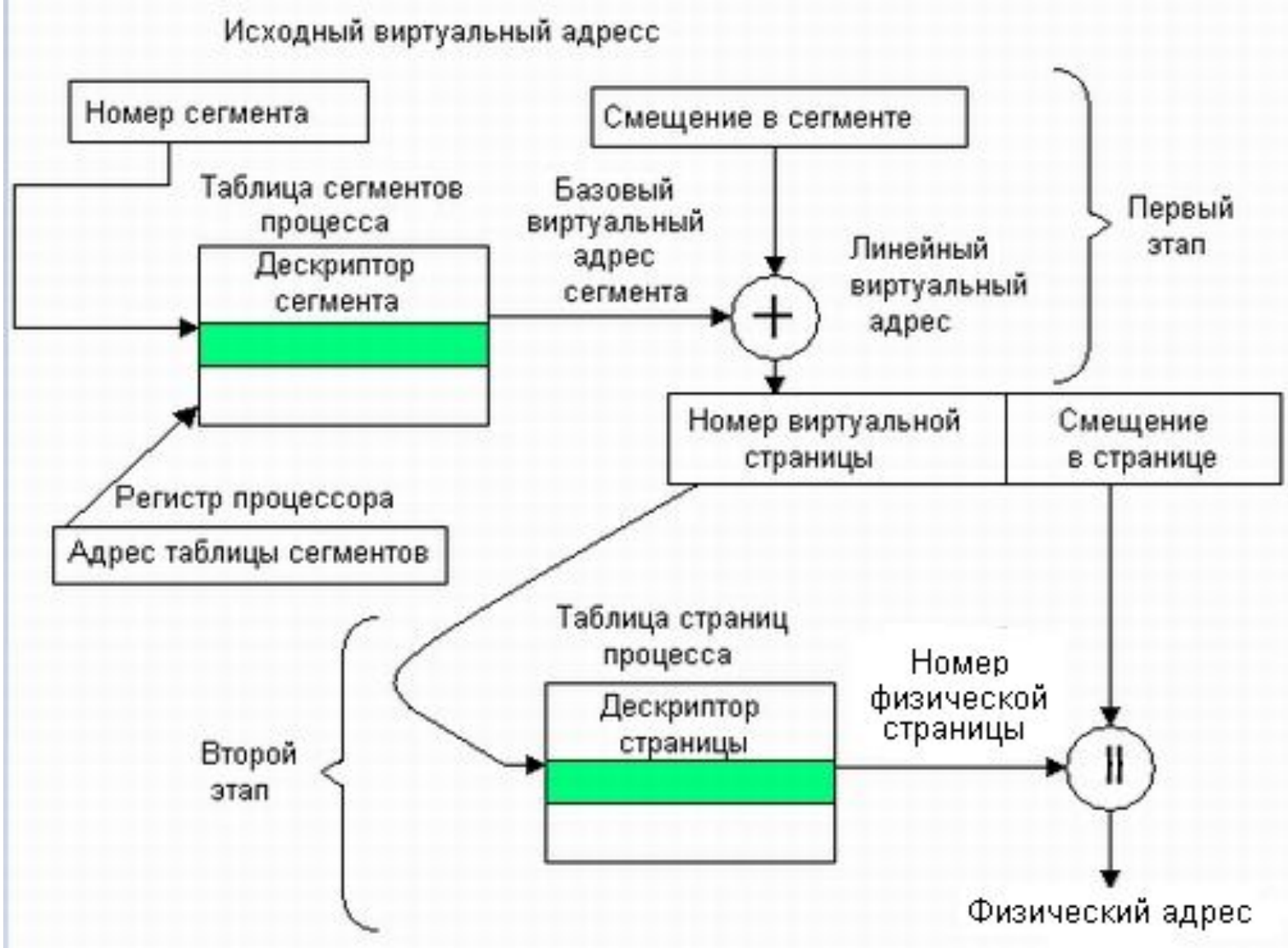
На ***первом этапе*** работает механизм сегментации.
Исходный виртуальный адрес, заданный в виде пары (номер сегмента, смещение), преобразуется в *линейный виртуальный адрес*.
Для этого на основании *базового адреса таблицы сегментов* и *номера сегмента* вычисляется *адрес дескриптора сегмента*.
Анализируются поля дескриптора и выполняется проверка возможности выполнения заданной операции.
Если доступ к сегменту разрешен, то вычисляется *линейный виртуальный адрес* путем *сложения* *базового адреса сегмента*, извлеченного из дескриптора, и *смещения*, заданного в исходном виртуальном адресе.

# Преобразование виртуального адреса в физический

На **втором этапе** работает страничный механизм.
Полученный *линейный виртуальный адрес* преобразуется в искомый *физический адрес*.
В результате преобразования *линейный виртуальный адрес* представляется
в том виде, в котором он используется при *страничной* организации памяти,
а именно в виде пары (*номер страницы, смещение в странице*).
Благодаря тому, что размер страницы выбран равным степени двойки, эта задача решается про
При этом в старших разрядах младших двоичных разр
содержится *номер виртуальной страницы*, а в младших — *смещение* искомого элемента относительно начала страницы.



# Преобразование виртуального адреса в физический



# Преобразование виртуальных адресов на платформе x64

64-разрядная Windows на платформе x64 применяет четырехуровневую схему таблиц страниц.

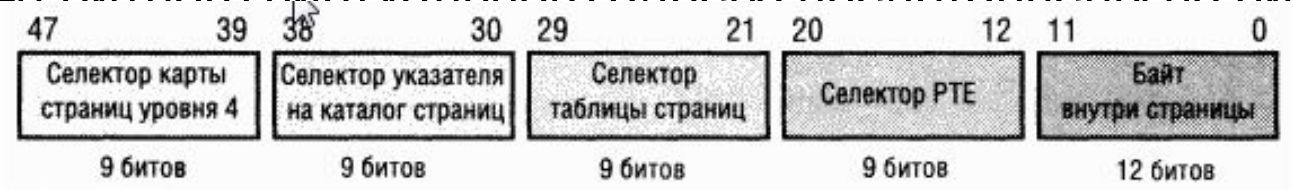
У каждого процесса имеется расширенный каталог страниц верхнего уровня (называемый *картой страниц уровня 4*), содержащий 512 указателей на структуру третьего уровня – *родительский каталог страниц*.

Каждый родительский каталог страниц хранит 512 указателей на *каталоги страниц второго уровня*, а те содержат по 512 указателей на *индивидуальные таблицы страниц*.

Наконец, таблицы страниц, в каждой из которых хранится 512 дескрипторов страниц (PTE), описывающих страницы виртуальной памяти.

**В текущих реализациях архитектуры x64 размер виртуальных адресов ограничен 48 битами.**

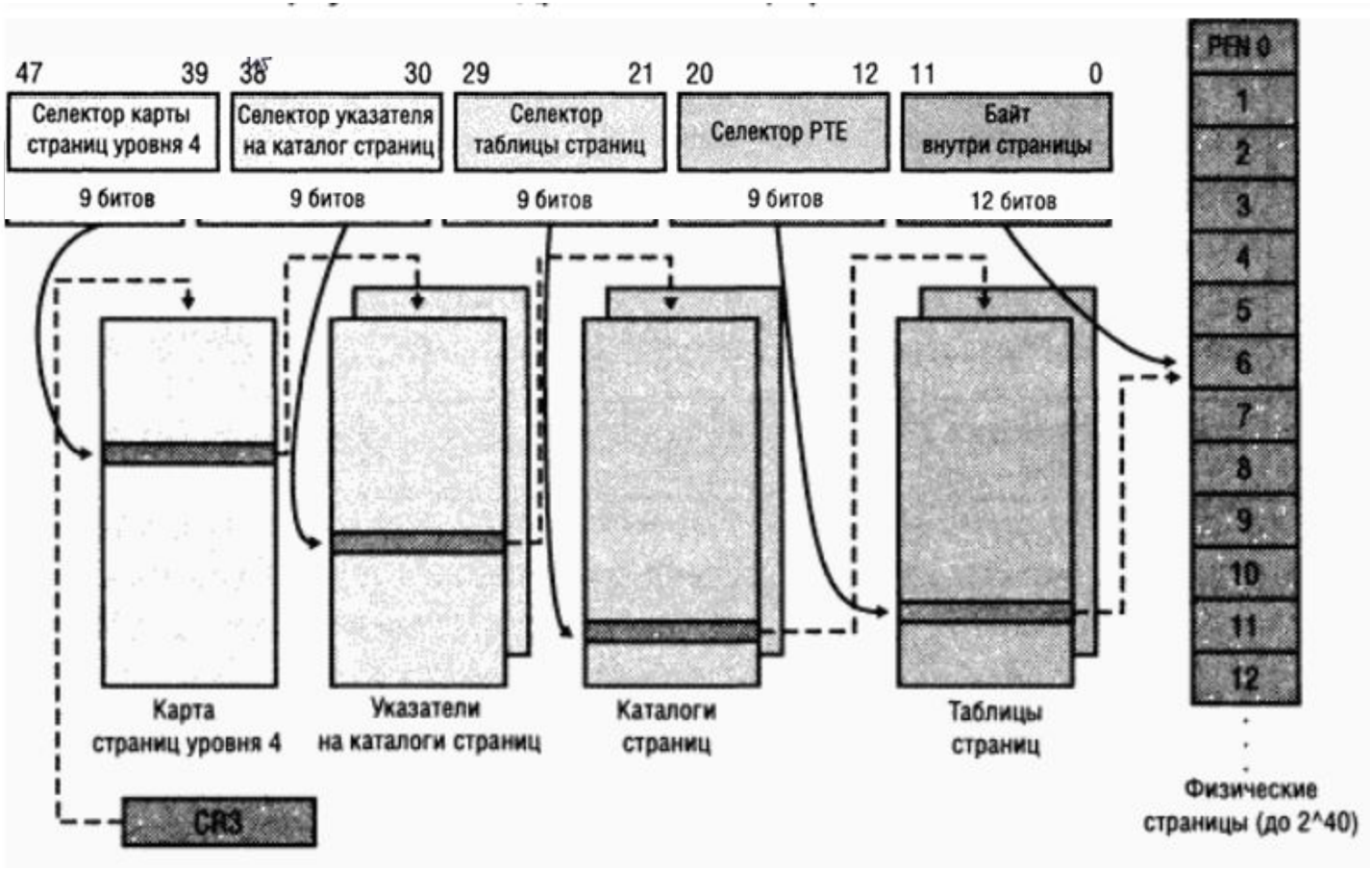
Элементы 48-битного виртуального адреса представлены на рисунке ниже:



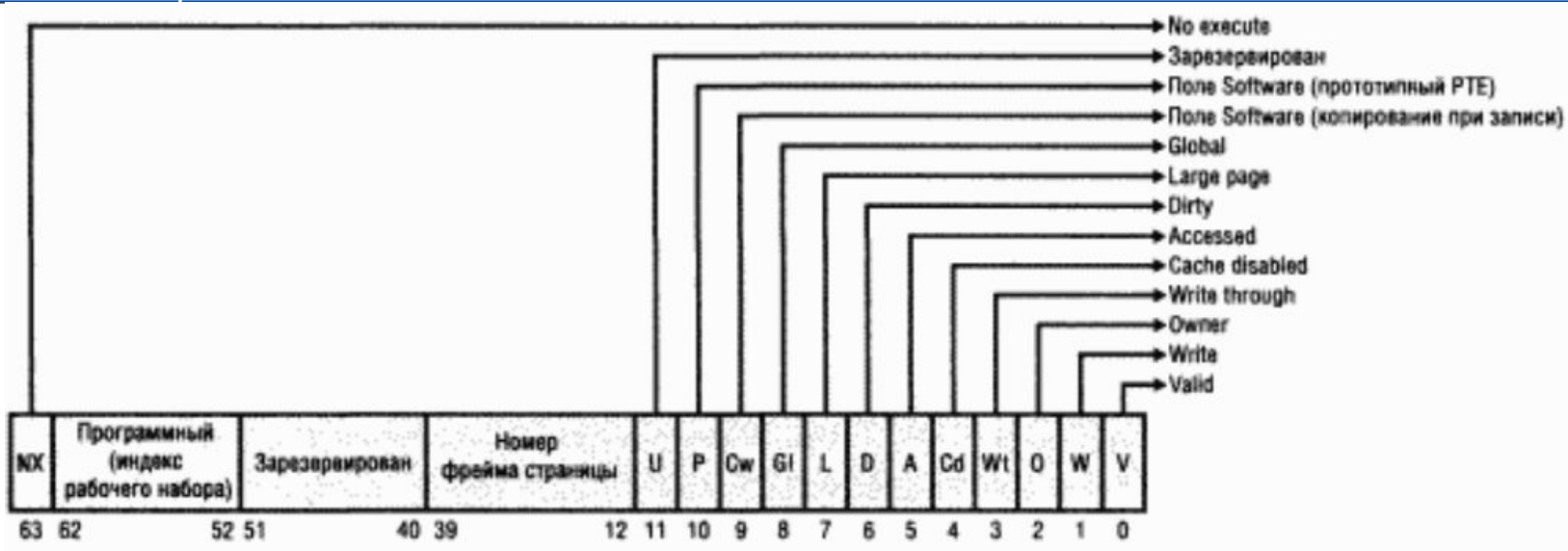

*Виртуальный адрес на платформе x64*

# Схема преобразования виртуальных адресов на платформе x64

CR3 – командный регистр микропроцессора

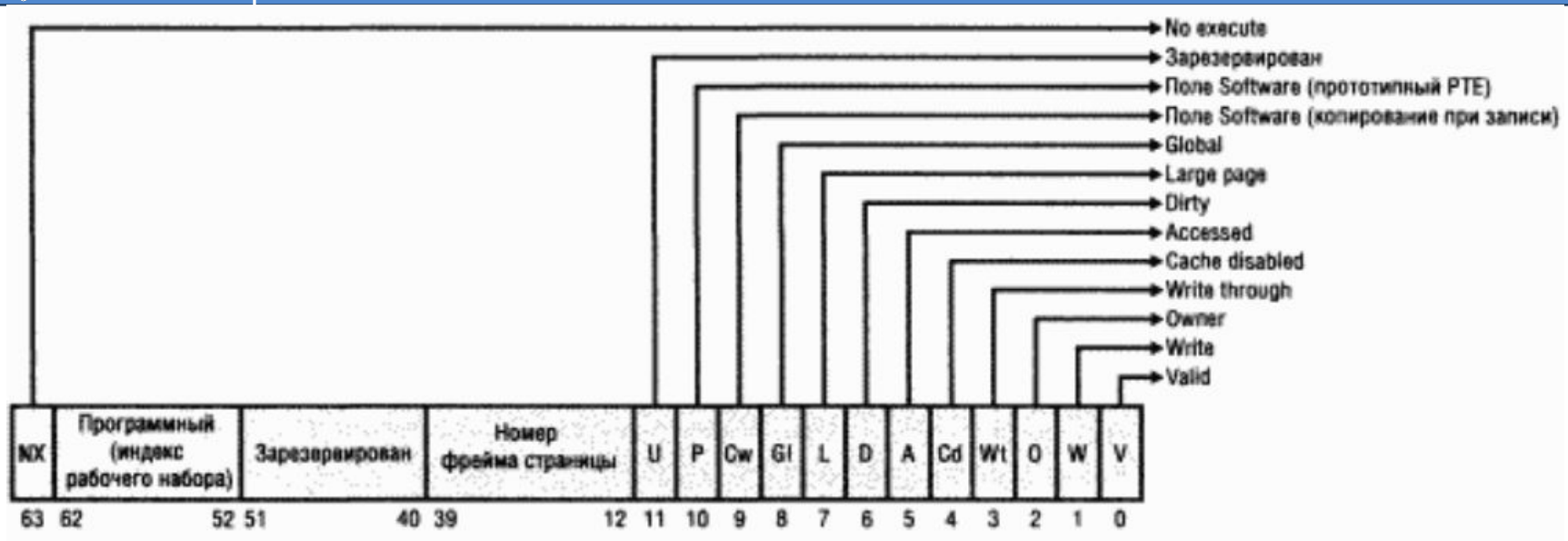


# Структура дескриптора страницы x64 PTE




| Битовые флаги | Описание |
|---|---|
| Accessed | Была операция чтения с данной страницы |
| Cache disabled | Кэширование данной страницы отключено |
| Dirty | Страница модифицирована |
| Global | Трансляция относится ко всем процессам (например, сброс буфера трансляции не повлияет на этот PTE) |
| Large page | Указывает, что PDE относится к 4-мегабайтной странице (или к 2-мегабайтной в PAE) |
| Owner | Указывает, доступна ли страница из кода пользовательского режима |
| Valid | Указывает, соответствует ли PTE странице в физической памяти |
| Write through | Отключает кэширование записи на данную страницу, в результате чего все измененные данные сразу же сбрасываются на диск |
| Write | В однопроцессорных системах указывает тип доступа (для чтения и записи или только для чтения), а в многопроцессорных системах определяет, доступна ли эта страница для записи (битовый флаг Write хранится в зарезервированной области PTE) |

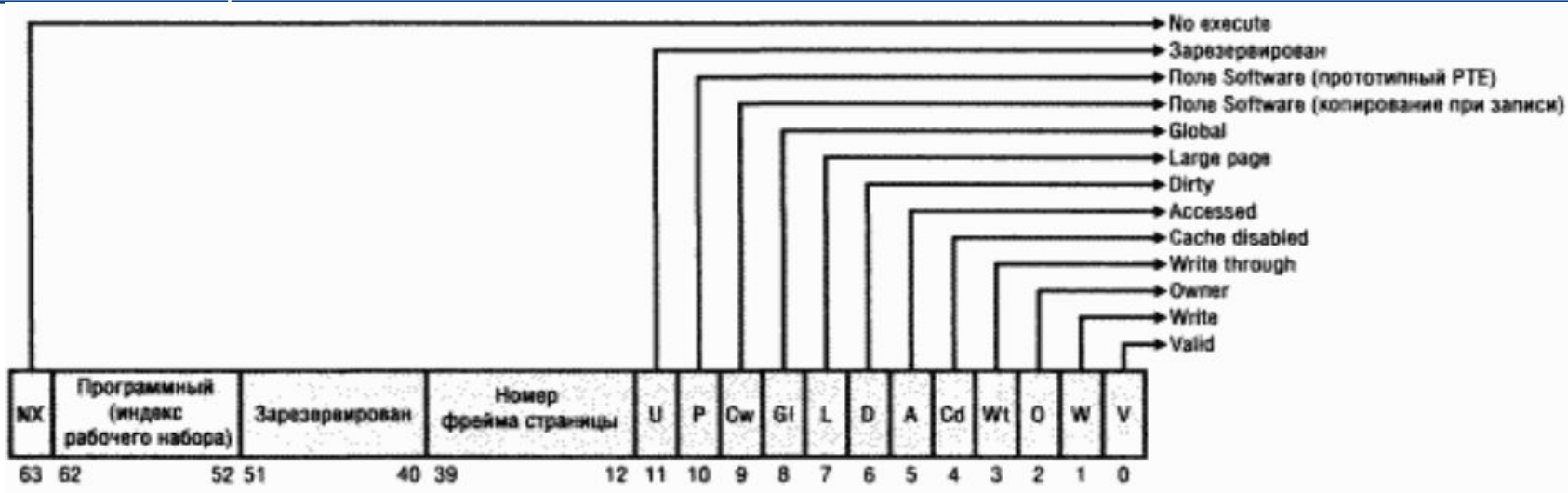
# Бит No execute/Запрет на выполнение




Предотвращение выполнения данных (data execution prevention, DEP), означает, что попытка передачи управления какой-либо инструкции на странице, помеченной атрибутом «запрет на выполнение», приведет к нарушению доступа к памяти.

Благодаря этому блокируются попытки определенных типов вирусов воспользоваться ошибками в ОС, которые иначе позволили бы выполнить код, размещенный на странице данных.

# Бит No execute/Запрет на выполнение




Попытка выполнить код на странице, помеченной атрибутом «запрет на выполнение», в режиме ядра вызывает крах системы с кодом ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY.

Если такая же попытка предпринимается в пользовательском режиме, то генерируется исключение STATUS_ACCESS_VIOLATION (0xc0000005); оно доставляется потоку, в котором была эта недопустимая ссылка.

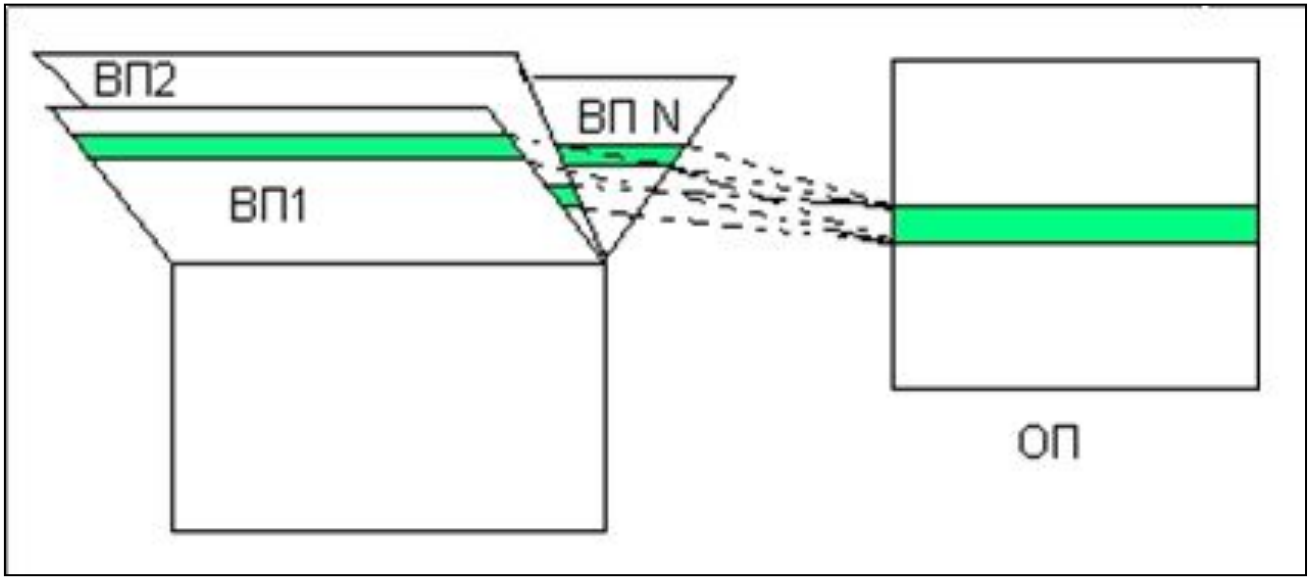
В 64-разрядных версиях Windows атрибут защиты «запрет на выполнение» всегда применяется ко всем 64-разрядным программам и драйверам устройств, и его нельзя отключить.

# Разделяемые сегменты

Подсистема виртуальной памяти представляет собой удобный механизм для решения *задачи совместного доступа* нескольких процессов к одному и тому же сегменту памяти, который в этом случае называется *разделяемой памятью*.

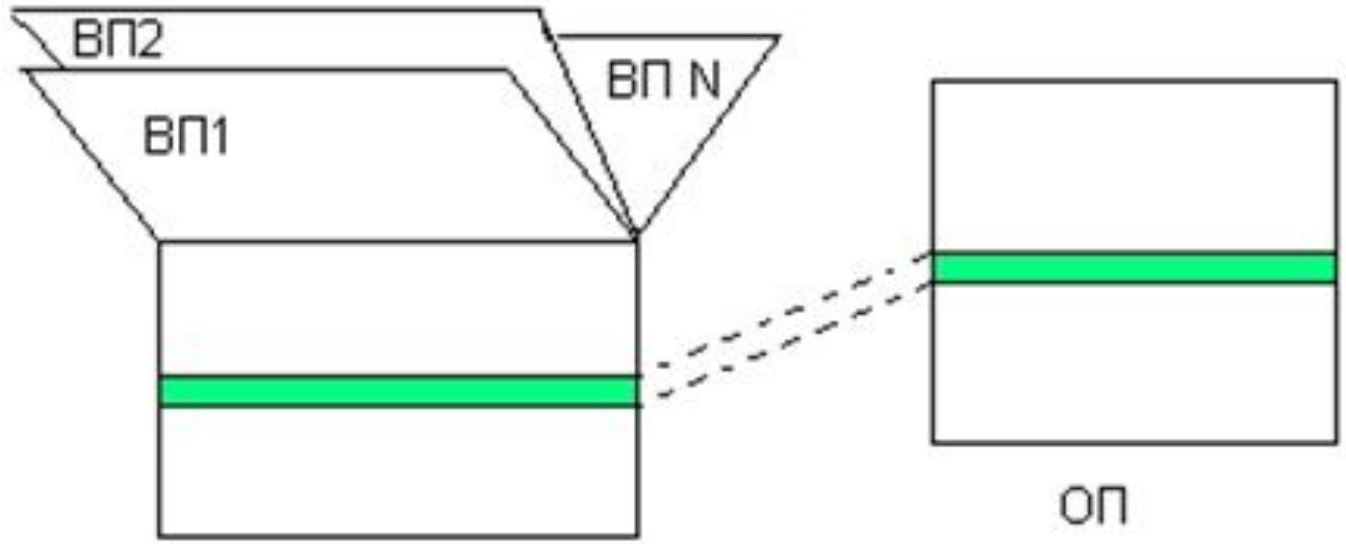
# Разделяемые сегменты. 1-й способ реализации.

Для организации разделяемого сегмента достаточно *поместить его в виртуальное адресное пространство каждого процесса*, которому нужен доступ к данному сегменту, а затем *настроить параметры отображения* этих виртуальных сегментов *так, чтобы они соответствовали одной и той же области оперативной памяти*.



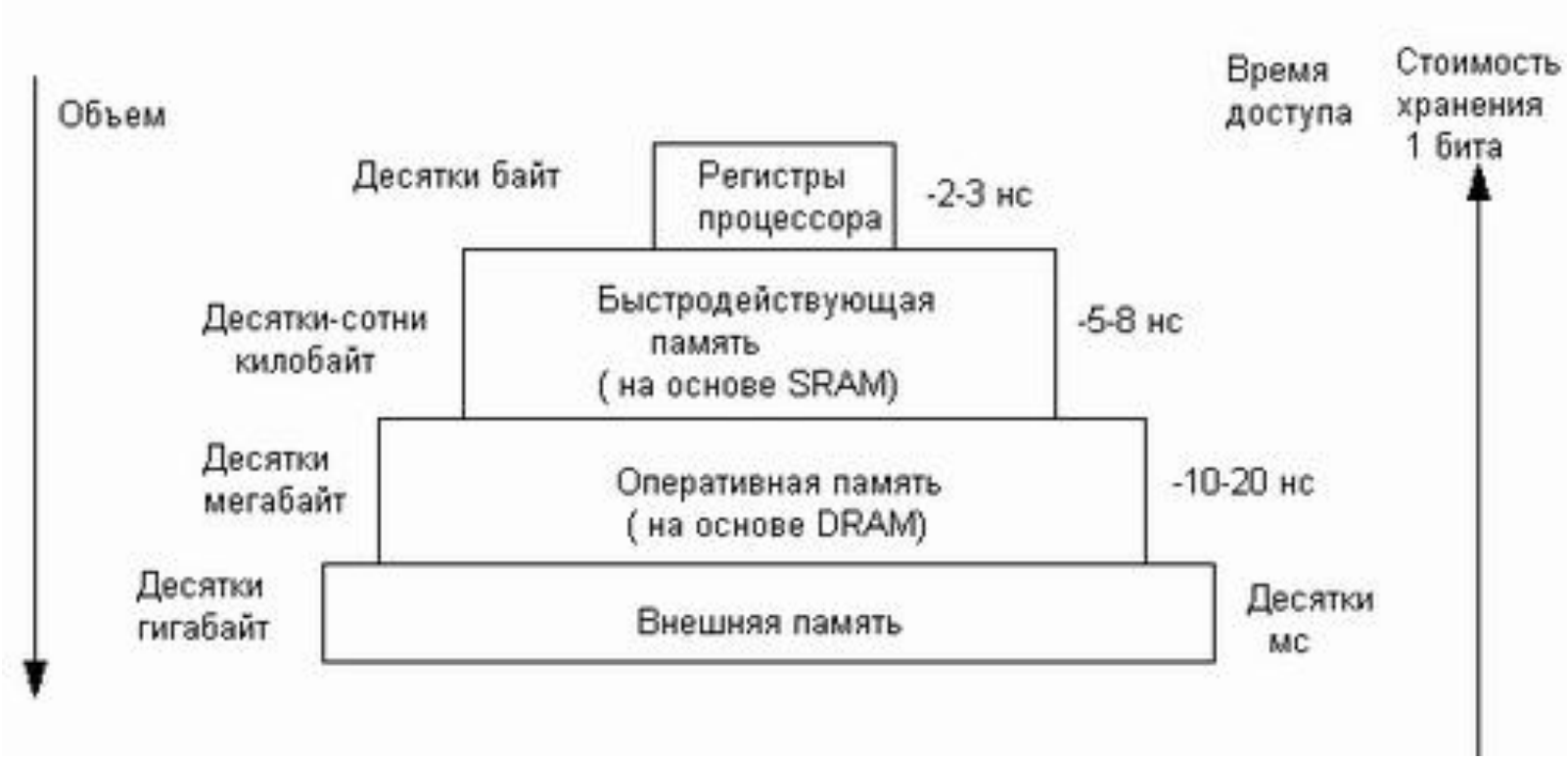
# Разделяемые сегменты. 2-й способ реализации.

Более экономичное для ОС решение — помещение **единственного разделяемого виртуального сегмента** в общую часть виртуального адресного пространства процессов, то есть **в ту часть, которая обычно используется для модулей ОС**.



# Иерархия запоминающих устройств

Память вычислительной машины представляет собой иерархию запоминающих устройств (ЗУ), отличающихся средним временем доступа к данным, объемом и стоимостью хранения одного бита.



# Кэширование данных

**Кэш-память**, или просто **кэш** (cache), — способ совместного функционирования двух типов запоминающих устройств, *отличающихся временем доступа и стоимостью хранения данных*, который за счет динамического копирования в «быстрое» ЗУ наиболее часто используемой информации из «медленного» ЗУ позволяет, с одной стороны, уменьшить среднее время доступа к данным, а с другой стороны, экономить более дорогую быстродействующую память.

При работе с оперативной памятью в качестве основной памяти выступает динамическая память DRAM (конденсаторы, «медленное» ЗУ), в качестве кэш-памяти выступает статическая память SRAM (триггеры, «быстрое» ЗУ).

# Состав записи кэш-памяти

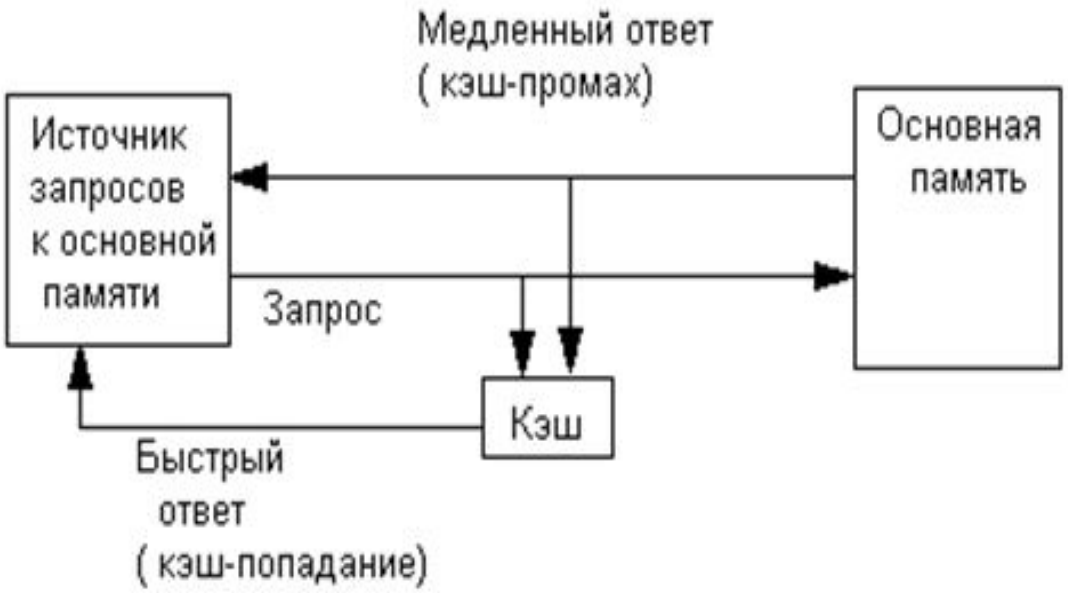
- значение элемента данных;
- адрес, который этот элемент данных имеет в основной памяти;
- дополнительная информация, которая используется для реализации алгоритма замещения данных в кэше и обычно включает признак модификации и признак обращения к данным.

Структура кэш-памяти

| Адрес данных в основной памяти | Данные | Управляющая информация |
|---|---|---|
| | | |
| | | |
| | | |

# Принцип действия кэш-памяти

При каждом обращении к основной памяти по физическому адресу просматривается содержимое кэш-памяти с целью определения, не находятся ли там нужные данные.
Кэш-память не является адресуемой, поэтому поиск нужных данных осуществляется по содержимому — *по взятому из запроса значению поля адреса в оперативной памяти*.




Далее возможен один из двух вариантов развития событий:
если данные обнаруживаются в кэш-памяти, то есть произошло ***кэш-попадание*** (cache-hit), они считываются из нее и результат передается источнику запроса;
если нужные данные отсутствуют в кэш-памяти, то есть произошел ***кэш-промах*** (cache-miss), они считываются из основной памяти, передаются источнику запроса и одновременно с этим копируются в кэш-память.

# Отчего зависит эффективность кэширования?

Пусть имеется основное запоминающее устройство со средним временем доступа к данным $t_1$ и кэш-память, имеющая время доступа $t_2$.

Из определения кэш-памяти следует, что что $t_2 < t_1$.

Пусть **t** — среднее время доступа к данным в системе с кэш-памятью,
**p** — вероятность кэш-попадания.

По формуле полной вероятности имеем:

$$t = t_1(1 - p) + t_2p = (t_2 - t_1)p + t_1$$

Среднее время доступа к данным в системе с кэш-памятью ***линейно зависит от вероятности попадания в кэш*** и изменяется от среднего времени доступа в основное запоминающее устройство $t_1$ при $p=0$ до среднего времени доступа непосредственно в кэш-память $t_2$ при $p=1$.

**Использование кэш-памяти имеет смысл только при высокой вероятности кэш-попадания.**

# Отчего зависит эффективность кэширования?

Вероятность обнаружения данных в кэше зависит:

- от объема кэша,
- от объема кэшируемой памяти,
- от алгоритма замещения данных в кэше,
- от особенностей выполняемой программы,
- от времени ее работы,
- от уровня мультипрограммирования.

# Объективные свойства данных

В большинстве реализаций кэш-памяти процент кэш-попаданий оказывается весьма высоким — свыше 90 %.

Такое высокое значение вероятности нахождения данных в кэш-памяти объясняется наличием у данных объективных свойств:
*пространственной и временной локальности*.

***Временная локальность***. *Если произошло обращение по некоторому адресу, то следующее обращение по тому же адресу с большой вероятностью произойдет в ближайшее время*.

***Пространственная локальность***. *Если произошло обращение по некоторому адресу, то с высокой степенью вероятности в ближайшее время произойдет обращение к соседним адресам*.

# Проблема согласования данных

- **Проблема согласования данных** возникает из-за наличия в компьютере *двух копий одних и тех же данных* — в основной памяти и в кэше.
- Если происходит запись в основную память по некоторому адресу, а содержимое этой ячейки находится в кэше, то в результате соответствующая запись в кэше становится недостоверной.

# Варианты согласования данных. Сквозная запись.

**Сквозная запись**.

- При каждом запросе к основной памяти, в том числе и при записи, просматривается кэш.
- Если данные, к которым выполняется обращение, находятся в кэше, то запись выполняется одновременно в кэш и основную память.
- Если же данные по запрашиваемому адресу отсутствуют, то запись выполняется только в основную память.

# Варианты согласования данных. Обратная запись.

**Отложенная (обратная) запись**.
При возникновении запроса к памяти выполняется просмотр кэша, и если запрашиваемых данных там нет, то запись выполняется только в основную память.
В противном случае запись производится только в кэш-память, при этом в описателе данных устанавливается признак модификации, который указывает на то, что при вытеснении этих данных из кэша необходимо переписать их в основную память, чтобы актуализировать устаревшее содержимое основной памяти.

# Стратегии вытеснения данных из кэш

Если список свободных ячеек кэш пуст, то выполняется алгоритм *вытеснения* одной из ячеек.

Стратегия вытеснения существенно влияет на *производительность* кэша.

Существуют следующие алгоритмы:

- *LRU (Least Recently Used) — вытесняются данные ячейке кэш, неиспользованные дольше всего;*
- *MRU (Most Recently Used) — вытесняются данные ячейки кэш, использованной последней;*
- *LFU (Least Frequently Used) — вытесняются данные ячейке кэш, использованной реже всех;*
- *ARC (Adaptive Replacement Cache) — алгоритм вытеснения, запатентованный IBM, комбинирующий LRU и LFU.*

# Отражение основной памяти на кэш

Алгоритм поиска и алгоритм замещения данных в кэше непосредственно зависят от того, каким образом основная память *отображается* на кэш-память.

При кэшировании данных из оперативной памяти используются две основные схемы отображения: **случайное отображение** и **детерминированное отображение**.

# Случайное отображение

- Элемент оперативной памяти в общем случае может *быть размещен в произвольном месте* кэш-памяти.
- Для того чтобы в дальнейшем можно было найти нужные данные в кэше, они помещаются туда *вместе со своим адресом*, то есть тем адресом, который данные имеют в оперативной памяти.
- При каждом запросе к оперативной памяти выполняется поиск в кэше, причем *критерием поиска* выступает *адрес оперативной памяти из запроса*.

# Ассоциативный поиск

Для кэшей со случайным отображением используется так называемый *ассоциативный поиск*, при котором *сравнение выполняется не последовательно с каждой записью кэша, а параллельно со всеми его записями*.
Признак, по которому выполняется сравнение, называется *тегом* (tag).
В данном случае *тегом является адрес данных в оперативной памяти*.

# Ассоциативные регистры

# Недостаток и область применения

- Недостаток: электронная реализация такой схемы приводит к удорожанию памяти, причем стоимость существенно возрастает с увеличением объема запоминающего устройства.

- Область применения: ассоциативная кэш-память используется в тех случаях, когда для обеспечения высокого процента попадания достаточно *небольшого объема памяти*.

# Детерминированное отображение

- *Любой элемент основной памяти всегда отображается в одно и то же место кэш-памяти.*
  В этом случае кэш-память разделена на строки, каждая из которых имеет свой номер и предназначена для хранения записи об одном элементе данных.
  Однако, между номерами строк кэш-памяти и адресами оперативной памяти устанавливается соответствие «один ко многим»: *одному номеру строки соответствует несколько (обычно достаточно много) адресов оперативной памяти.*

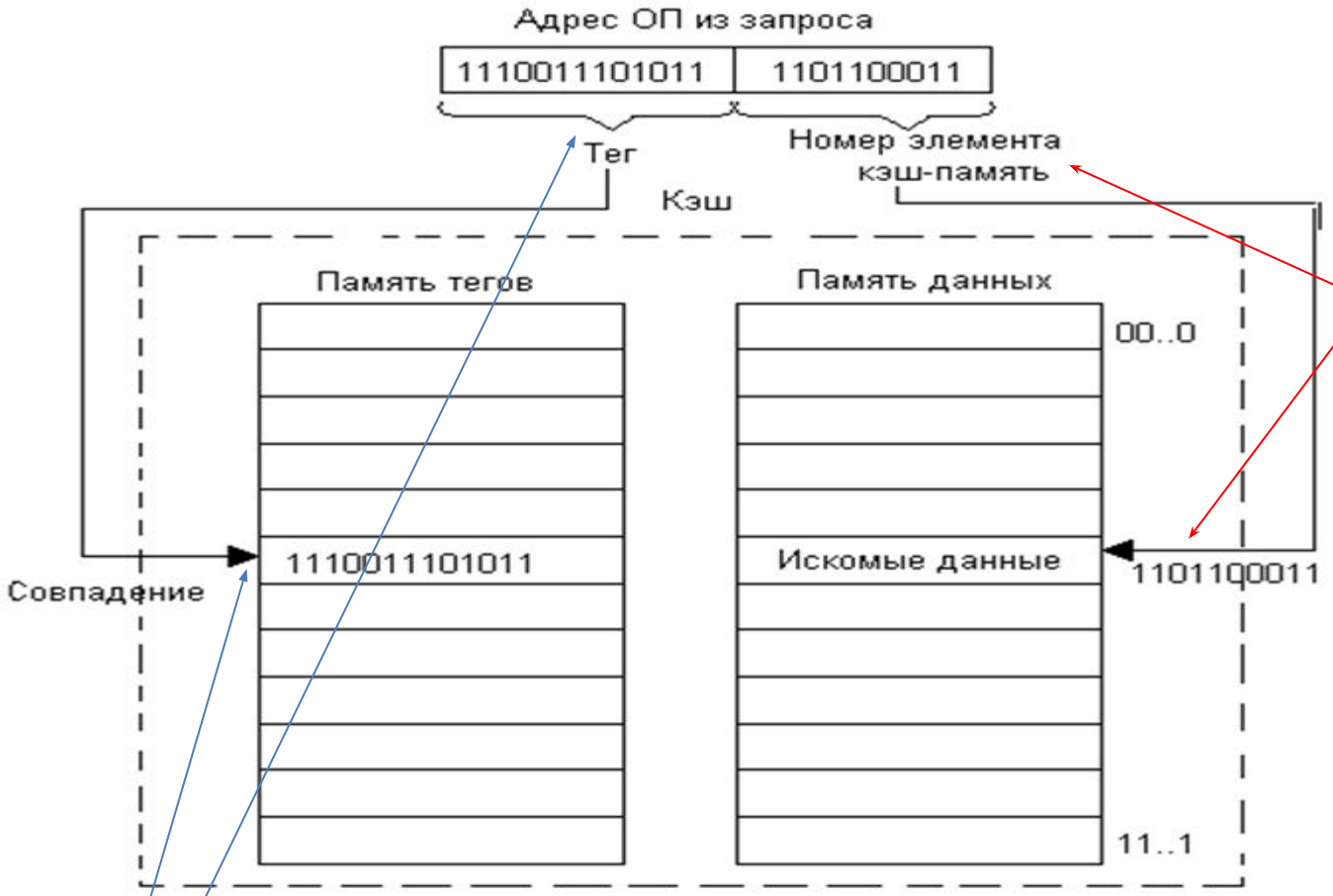
# Прямое отображение

Частным случаем детерминированного отображения является *прямое отображение*.

В качестве отображающей функции может использоваться простое *выделение нескольких разрядов* из адреса оперативной памяти, которые интерпретируются как номер строки кэш-памяти.


Пусть в кэш-памяти может храниться 1024 записи, то есть кэш имеет 1024 строки, пронумерованные от 0 до 1023. Тогда любой адрес оперативной памяти может быть отображен на адрес кэш-памяти простым отделением 10 двоичных разрядов

# Прямое отображение

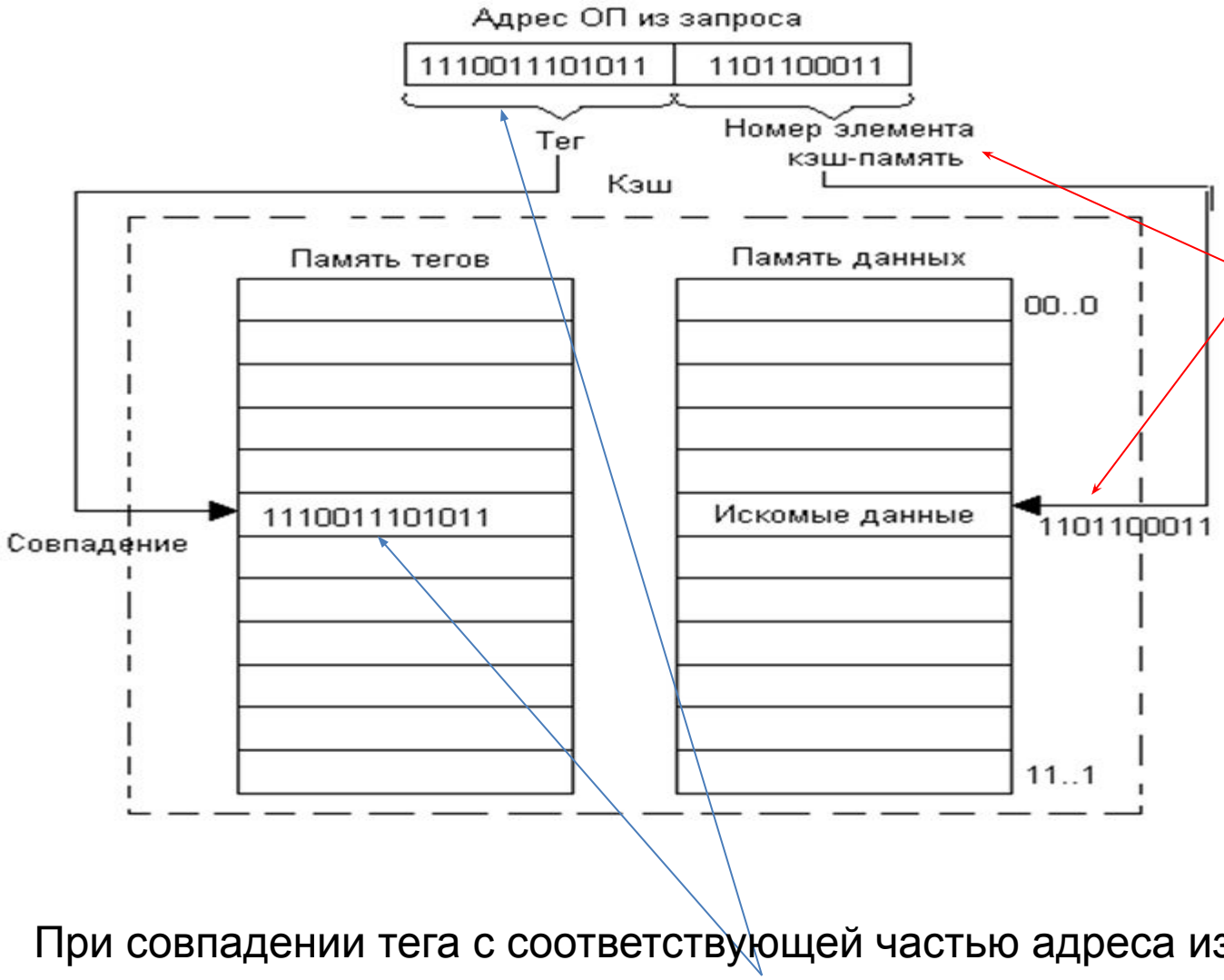



При поиске данных в кэше используется быстрый прямой доступ к записи по номеру строки, полученному путем обработки адреса оперативной памяти из запроса.

Однако поскольку в найденной строке могут находиться данные из любой ячейки оперативной памяти, младшие разряды адреса которой совпадают с номером строки, необходимо выполнить дополнительную проверку.

Для этих целей каждая строка кэш-памяти дополняется тегом, содержащим старшие биты адреса данных в оперативной памяти.

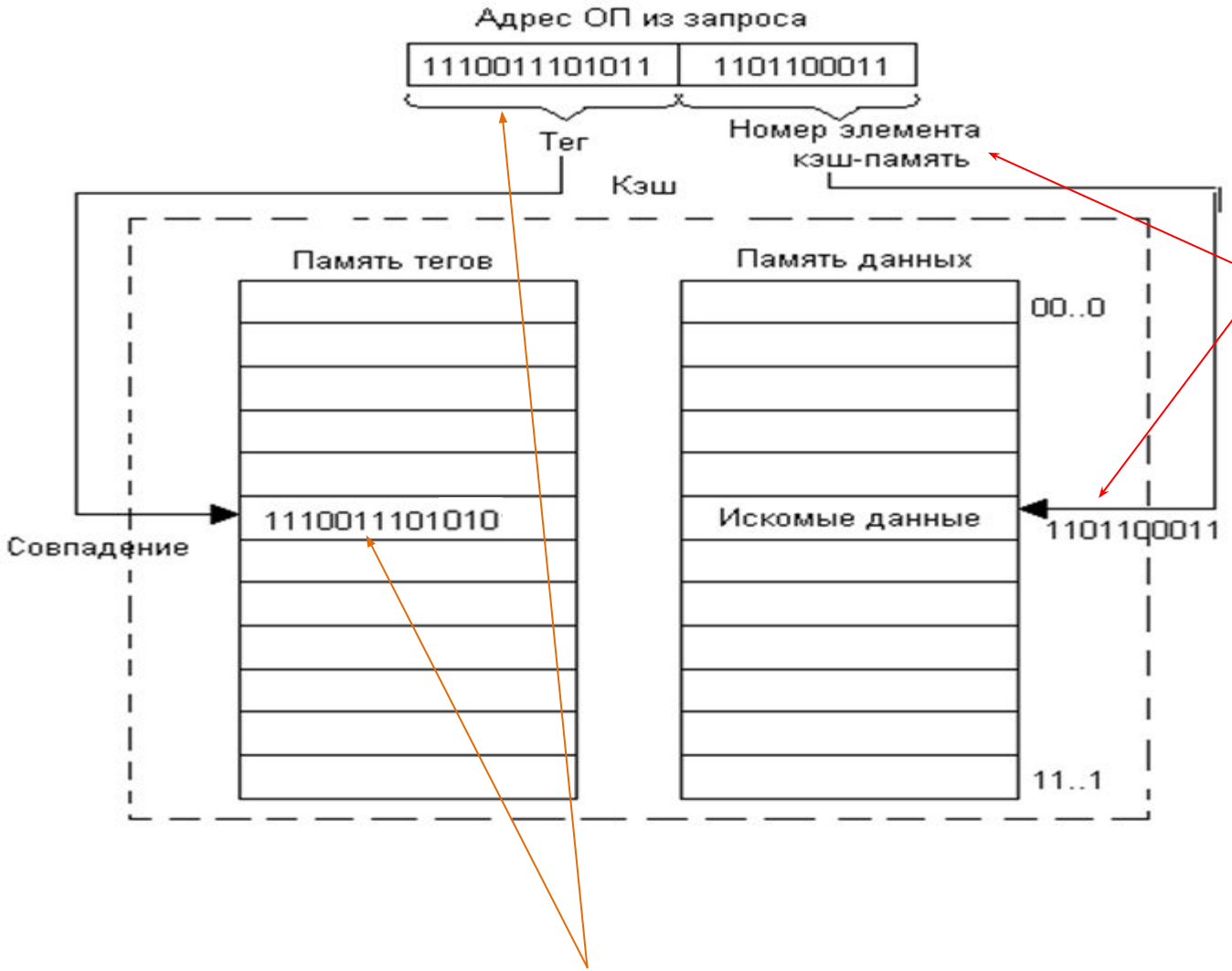
# Прямое отображение




При поиске данных в кэше используется быстрый прямой доступ к записи по номеру строки, полученному путем обработки адреса оперативной памяти из запроса.

При совпадении тега с соответствующей частью адреса из запроса констатируется кэш-попадание.

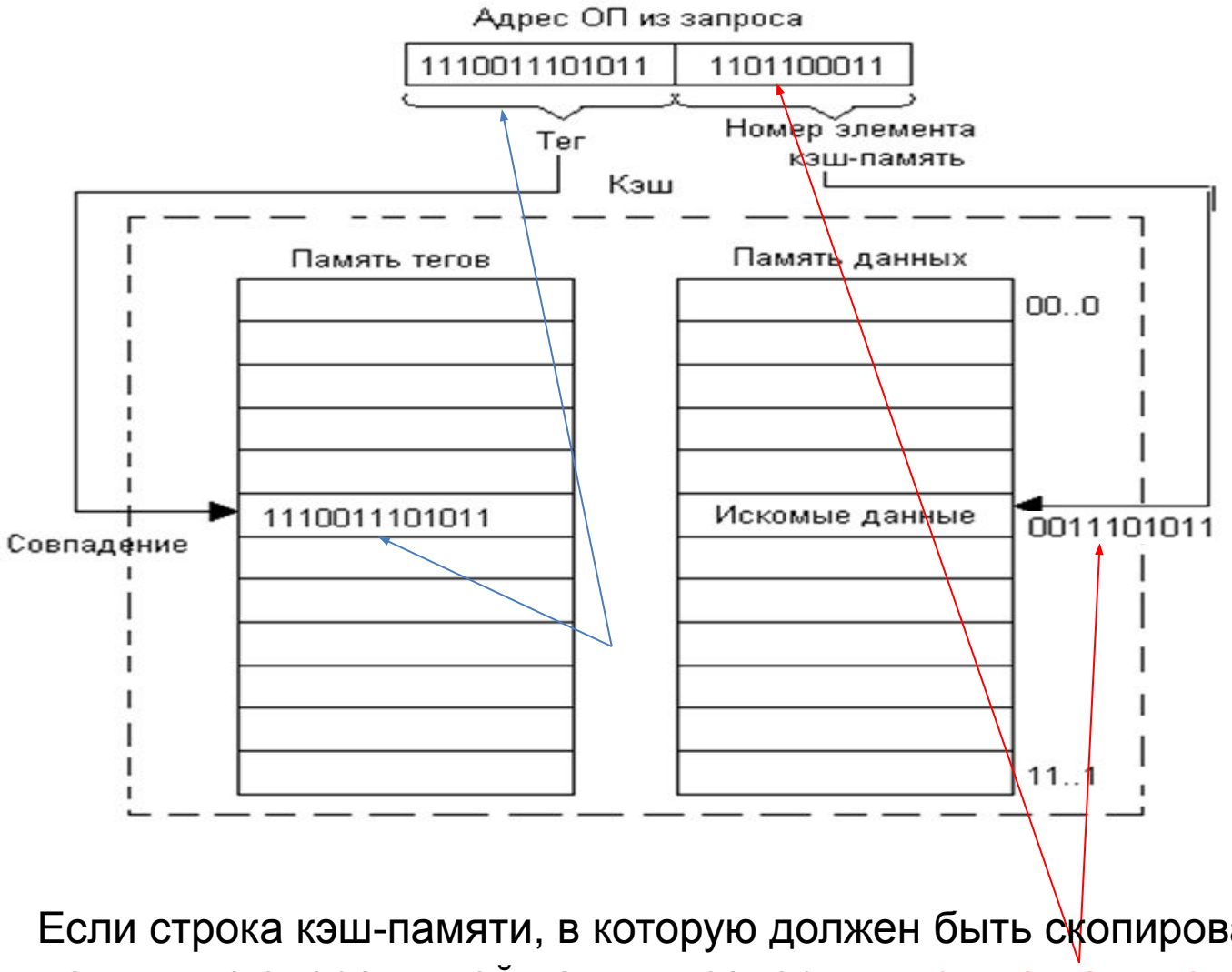
# Прямое отображение




При поиске данных в кэше используется быстрый прямой доступ к записи по номеру строки, полученному путем обработки адреса оперативной памяти из запроса.

Если же произошел кэш-промах, то данные считываются из оперативной памяти и копируются в кэш.

# Прямое отображение




Если строка кэш-памяти, в которую должен быть скопирован элемент данных из оперативной памяти, содержит другие данные, то последние **вытесняются из кэша** копированием элемента данных из оперативной памяти в кэш.

# Прямое отображение


Если строка кэш-памяти, в которую должен быть скопирован элемент данных из оперативной памяти, содержит другие данные, то последние **вытесняются из кэша** копированием элемента данных из оперативной памяти в кэш.

# Схема выполнения запросов к кэш-памяти