Тема 3. Управление памятью в операционных системах

3.1. Организация памяти современного компьютера

3.2. Функции операционной системы по управлению памятью

3.3. Алгоритмы распределение памяти

3.3.1. Классификация методов распределения памяти

3.3.2. Распределение памяти фиксированными разделами

3.3.3. Распределение памяти динамическими разделами

3.3.4. Распределение памяти перемещаемыми разделами

3.4. Виртуальная память

3.4.1. Методы структуризации виртуального адресного пространства

3.4.2. Страничная организация виртуальной памяти

3.4.3. Оптимизация функционирования страничной виртуальной памяти

3.4.4. Сегментная организация виртуальной памяти

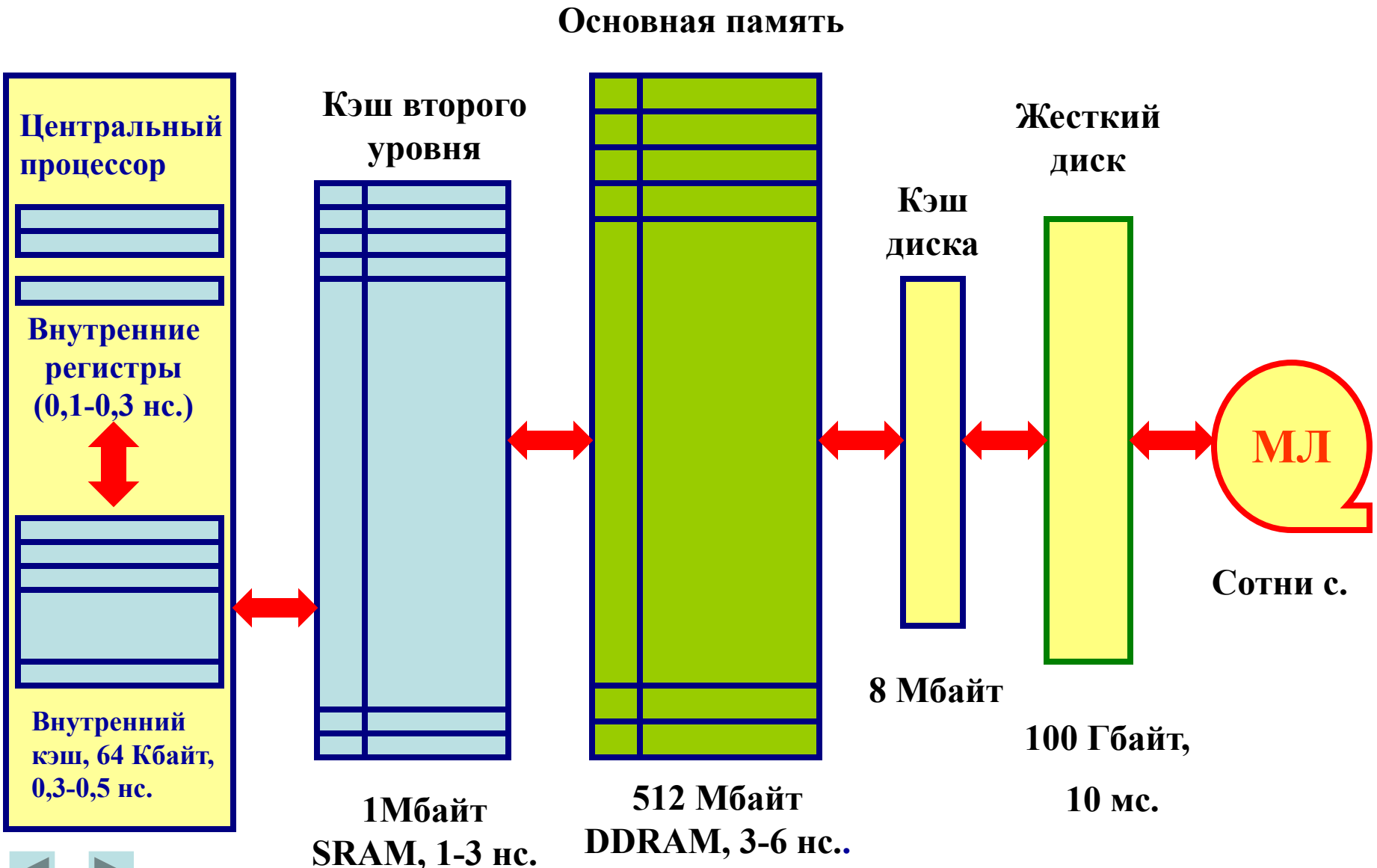
3.1. Организация памяти современного компьютера

3.1.1. Логическая организация памяти: **Линейное (одномерное) адресное пространство, отражающее особенности аппаратного обеспечения, но не соответствующее современной технологии создания программного обеспечения.**

Большинство программ организовано в виде модулей. Для эффективной работы с пользовательскими программами необходимо чтобы:

- **Модули могли быть созданы и скомпилированы независимо друг от друга, при этом все ссылки из одного модуля в другой разрешаются системой во время работы программы.**
- **Разные модули могли получать разные степени защиты (только чтение, только исполнение и т. п.) за счет весьма умеренных накладных расходов.**
- **Возможно применение механизма, обеспечивающего совместное использование модулей разными процессами (для случая сотрудничества разных процессов в работе над одной задачей).**

3.1.2. Физическая организация памяти

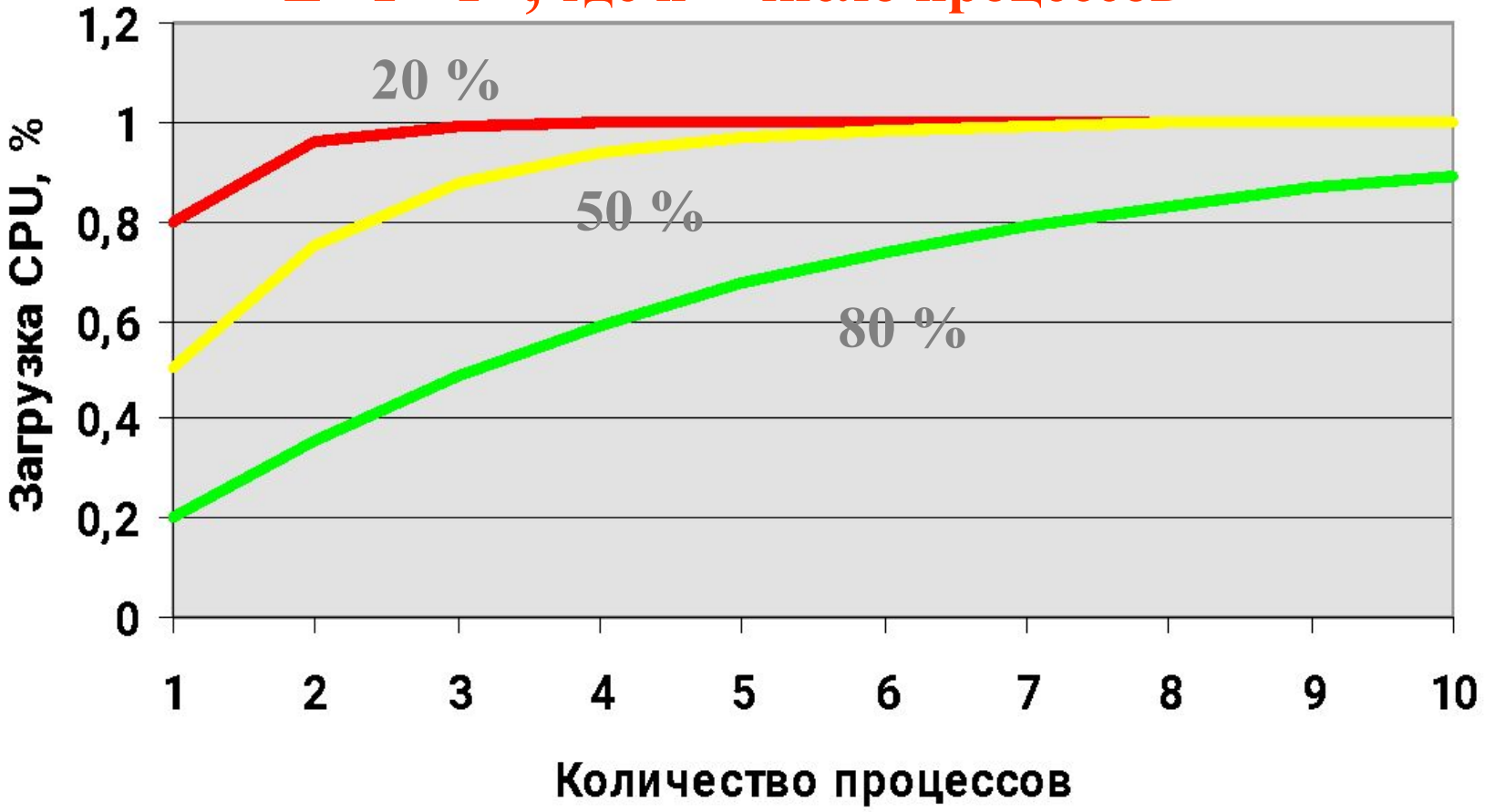


3.1.2. Физическая организация памяти

Предположим, процессор имеет доступ к памяти двух уровней. На первом уровне содержится E_1 слов, и он характеризуется временем доступа T_1 = 1 нс. К этому уровню процессор может обращаться непосредственно. Однако если требуется получить слово, находящееся на втором уровне, то его сначала нужно передать на первый уровень. При этом передается не только требуемое слово, а блок данных, содержащий это слово. Поскольку адреса, к которым обращается процессор, имеют тенденцию собираться в группы (циклы, подпрограммы), процессор обращается к небольшому повторяющему набору команд. Таким образом, работа процессора с вновь полученным блоком памяти будет проходить достаточно длительное время.

Зависимость загрузки процессора от количества процессов в памяти

$Z = 1 - P^n$, где n – число процессов



3.1.3. Виртуальная память

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

Очевидно, что имеет смысл *временно выгружать неактивные процессы*, находящиеся в ожидании каких-либо ресурсов, в том числе очередного кванта времени центрального процессора.

К моменту, когда *пройдет очередь выполнения выгруженного процесса, его образ возвращается с диска в оперативную память*.

Если при этом обнаруживается, что *свободного места в оперативной памяти не хватает*, то *на диск выгружается другой процесс*.

3.1.3. Виртуальная память

Такая подмена (виртуализация) оперативной памяти дисковой памятью позволяет повысить уровень мультипрограммирования, поскольку объем оперативной памяти теперь не столь жестко ограничивает число одновременно выполняемых процессов.

При этом суммарный объем оперативной памяти, занимаемой образами процессов, может существенно превосходить имеющийся объем оперативной памяти.

В данном случае в *распоряжение прикладного программиста предоставляется виртуальная оперативная память, размер которой намного превосходит реальную память системы* и ограничивается только возможностями адресации используемого процесса (в ПК на базе Pentium 2^{32} = 4 Гбайт).

3.1.3. Виртуальная память

Вообще *виртуальным* (кажущимся) называется ресурс, обладающий свойствами (в данном случае большой объем ОП), которых в действительности у него нет.

3.1.3. Виртуальная память

Виртуализация оперативной памяти осуществляется совокупностью аппаратных и программных (ОС) средств вычислительной системы автоматически без участия программиста и не сказывается на работе приложения.

Методы виртуализации памяти:

- свопинг (swapping) – образы процессов выгружаются на диск и возвращаются в ОЗУ целиком,
- виртуальная память (virtual memory) – между ОЗУ и диском перемещаются части образов процессов.

Достоинства свопинга: малые затраты времени на преобразование адресов в кодах программ. **Недостатки**: избыточность перемещаемых данных, замедление работы системы, неэффективное использование памяти, невозможность загрузить процесс, адресное пространство которого превышает объем свободной оперативной памяти.

3.1.3. Виртуальная память

Недостатки виртуальной памяти: необходимость преобразования виртуальных адресов в физические, сложность аппаратной и программной (ОС) поддержки.

3.2. Функции операционной системы по управлению памятью

В *однопрограммных операционных системах основная память разделяется на две* части.

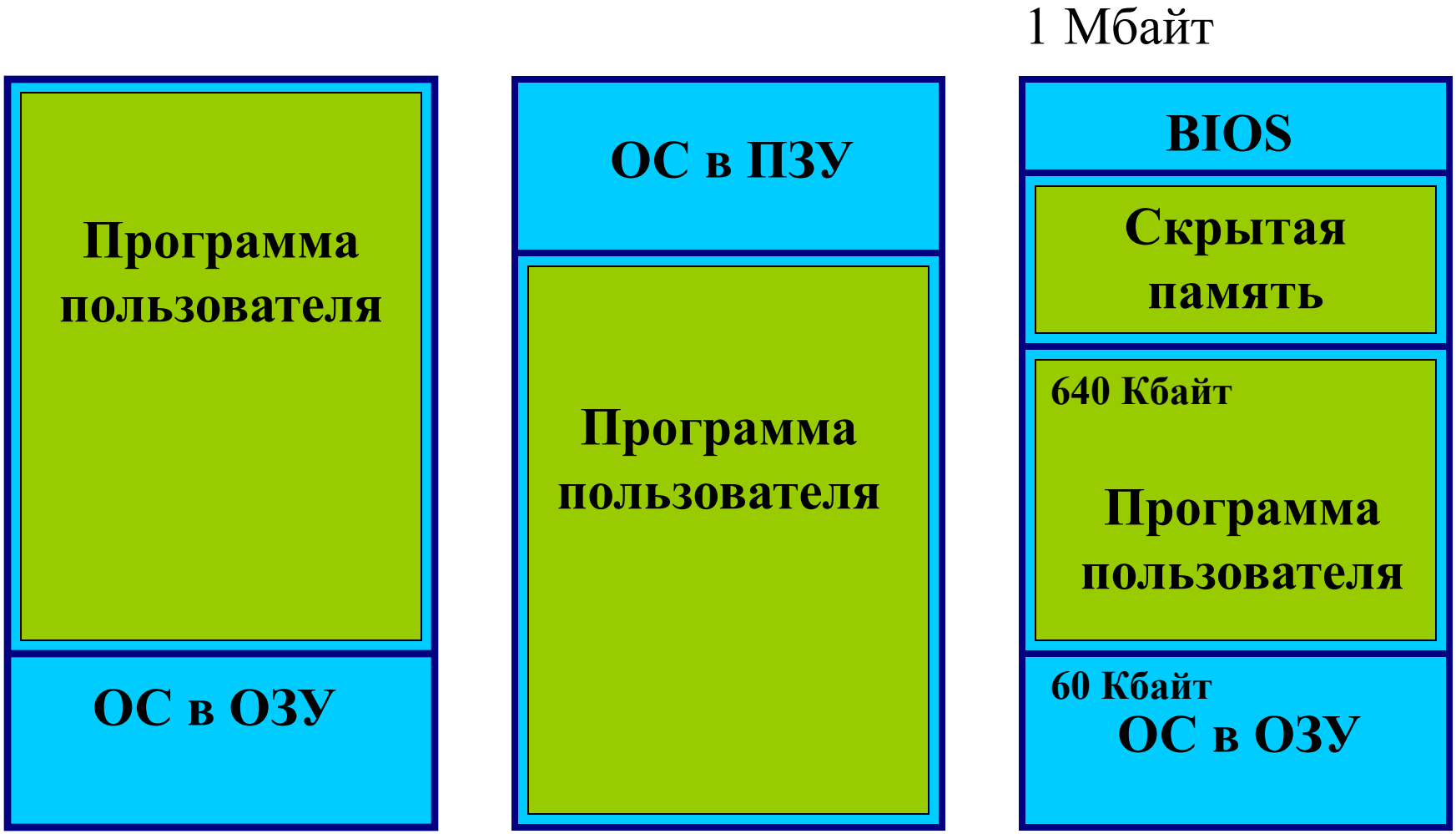
Одна часть - для операционной системы (резидентный монитор, ядро), а вторая - для выполняющейся в текущий момент времени программы.

В многопрограммных ОС «пользовательская» часть памяти - важнейший ресурс вычислительной системы - должна быть распределена для размещения нескольких процессов, в том числе процессов ОС.

3.2. Функции операционной системы по управлению памятью

Эта задача распределения выполняется операционной системой динамически специальной *подсистемой управления памятью* (memory management).

3.2. Функции операционной системы по управлению памятью



Распределение памяти в однопрограммных ОС

3.2. Функции операционной системы по управлению памятью

Распределение памяти в однопрограммных ОС

В ранних ОС управление памятью сводилось просто к загрузке программы и ее данных из некоторого внешнего накопителя (перфоленты, магнитной ленты или магнитного диска) в ОЗУ.

При этом память разделялась между программой и ОС. На рисунке показаны три варианта такой схемы.

Первая модель **раньше применялась на мэйнфреймах и мини-компьютерах.**

Вторая схема **сейчас используется на некоторых карманных компьютерах и встроенных системах.**

3.2. Функции операционной системы по управлению памятью

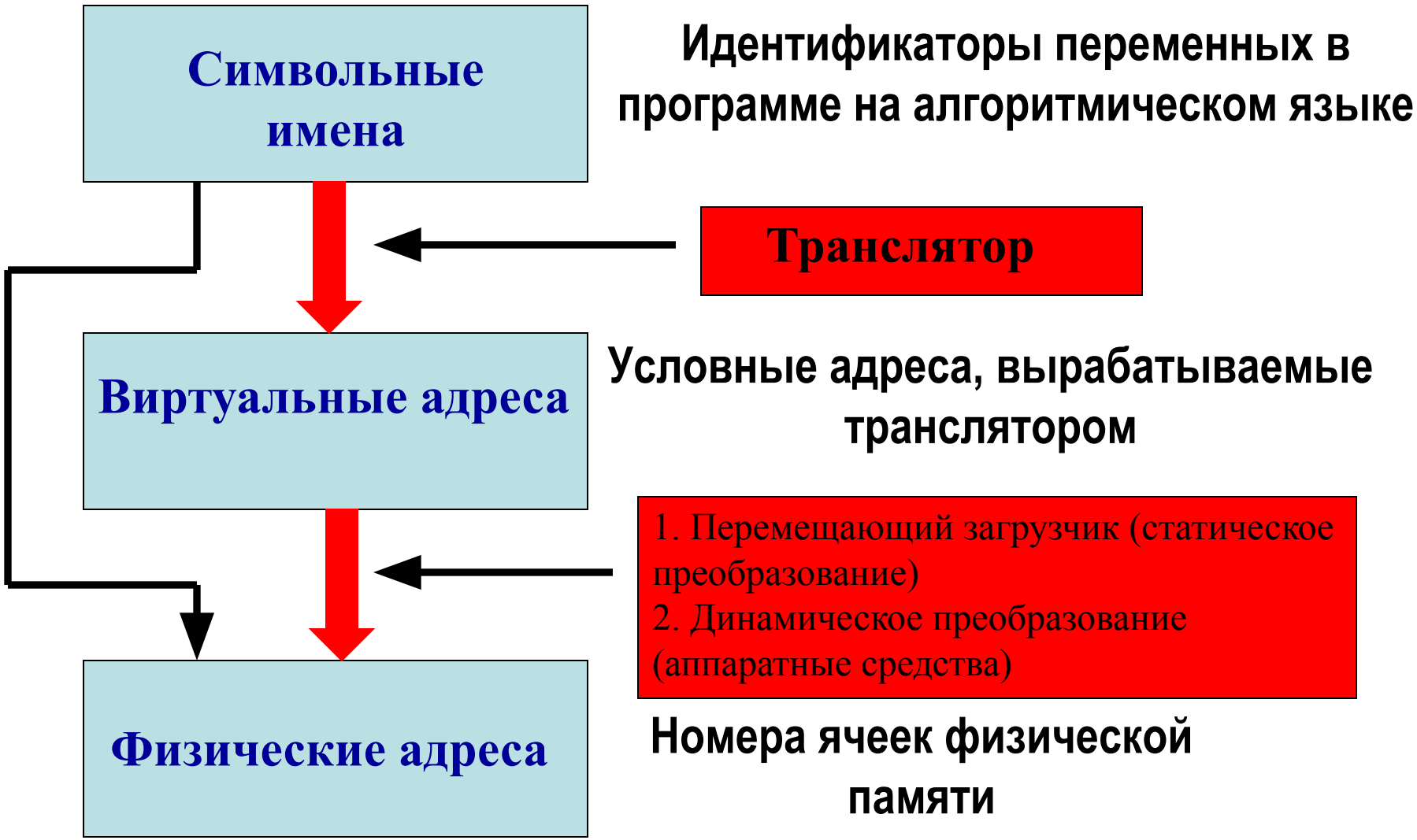
Распределение памяти в однопрограммных ОС

Третья модель **была характерна для ранних персональных компьютеров с MS DOS.**

Функции операционной системы по управлению памятью в мультипрограммных системах

- отслеживание (учет) свободной и занятой памяти;
- первоначальное и динамическое распределение памяти процессам приложений и самой ОС;
- освобождение памяти при завершении процессов;
- настройка адресов программы на конкретную область физической памяти;
- полное или частичное вытеснение кодов и данных процессов из ОП на диск, когда размеры ОП недостаточны для размещения всех процессов и возвращение их в ОП;
- защита памяти, выделенной процессу, от возможных вмешательств со стороны других процессов;
- дефрагментация памяти.

Типы адресов



Типы адресов

Для идентификации переменных и команд на разных этапах жизненного цикла программы используются *символьные* имена, *виртуальные* (математические, условные, логические - все это синонимы) и *физические* адреса.

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

Типы адресов

Виртуальные адреса **вырабатывают транслятор, переводящий программу на машинный язык. Поскольку во время трансляции не известно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, считая по умолчанию, что начальным адресом программы будет нулевой адрес.**

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

Совокупность виртуальных адресов процесса называется *виртуальным адресным пространством*.

Типы адресов

Диапазон адресов виртуального пространства у всех процессов один и тот же и определяется разрядностью адреса процессора (для Pentium адресное пространство составляет объем, равный 2^{32} байт).

Существует два принципиально отличающихся подхода к преобразованию виртуальных адресов в физические.

В первом случае *такое преобразование выполняется один раз для каждого процесса во время начальной загрузки программы в память*.

Типы адресов

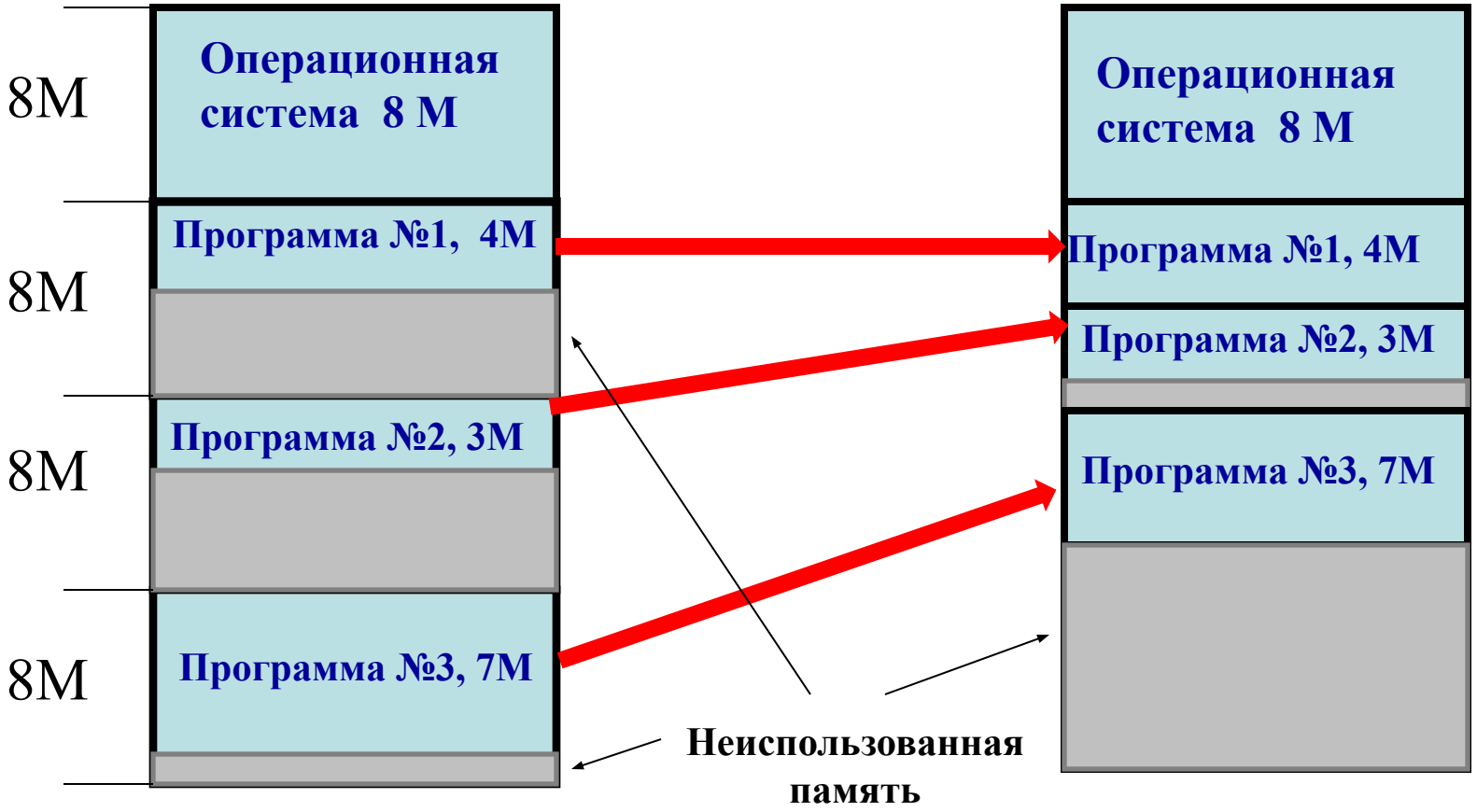
Преобразование осуществляет перемещающий загрузчик на основании имеющихся у него данных о начальном адресе физической памяти, в которую предстоит загружать программу, а также информации, предоставляемой транслятором об адресно-зависимых элементах программы.

Второй способ заключается в том, что *программа загружается в память в виртуальных адресах*. Во время выполнения программы при каждом обращении к памяти операционная система преобразует виртуальные адреса в физические.

3.3. Алгоритмы распределение памяти

3.3.1. Классификация методов распределения памяти

3.3.2. Распределение памяти фиксированными разделами (MFT в OS/360)



Разделы одинакового размера

Разделы разного размера

3.3.2. Распределение памяти фиксированными разделами

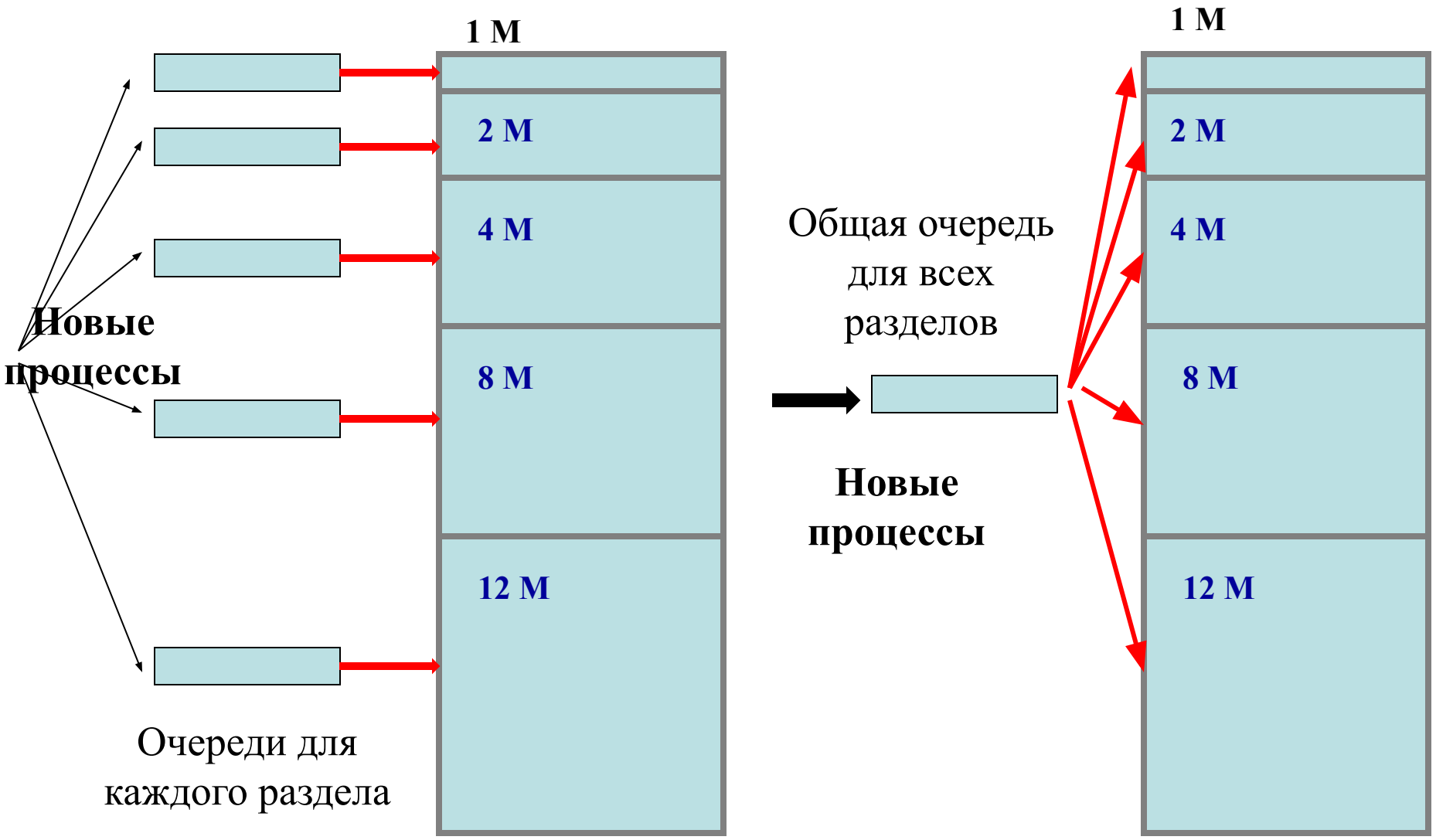
1. Разделы одинакового размера.

- Процесс загружается в любой доступный раздел.
- Если все разделы заняты процессами, то любой процесс не готовый к немедленной работе может быть выгружен для освобождения памяти новому процессу.

Недостатки:

- Необходимость разработки оверлеев при больших размерах программ. Когда требуется модуль, отсутствующий в данный момент в ОП, пользовательская программа должна сама его загрузить в раздел памяти программы. В данном случае управление памятью во многом возлагается на программиста.
- Неэффективное использование памяти. Любая программа, независимо от ее размера, занимает раздел целиком. При этом могут оставаться неиспользованные участки памяти большого размера (внутренняя фрагментация).

Разделы разного размера



Распределение памяти фиксированными разделами

2. Разделы разного размера. Очередь к каждому разделу.

Процесс размещается в наименьшем свободном разделе, способном вместить этот процесс.

Достоинство: возможность распределения процессов между разделами с минимизацией внутренней фрагментации.

Недостаток: возможно неэффективное использование памяти за счет «простоя» больших разделов при наличии только небольших процессов.

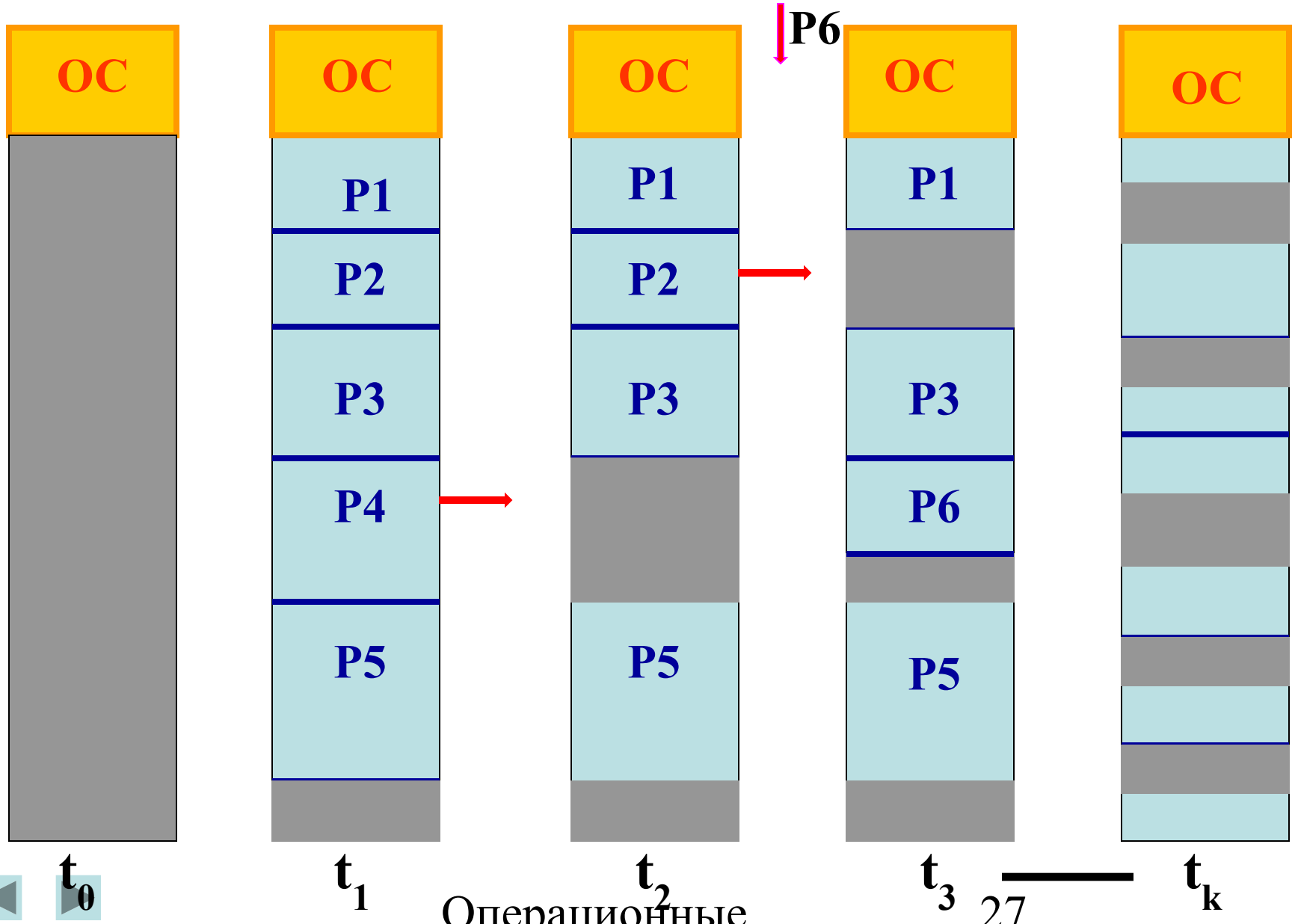
3. Разделы разного размера. Общая очередь к разделам.

Когда требуется загрузить процесс в ОП, выбирается наименьший доступный раздел, способный вместить данный процесс.

Достоинство: улучшается использование памяти.

В целом для распределения памяти фиксированными разделами
Достоинства: простота, минимальные требования к операционной системе. Недостатки: 1) количество разделов, определенных во время генерации ОС (режим MFT OS/360), ограничивает число активных процессов; 2) неэффективное использование памяти.

3.3.3. Распределение памяти динамическими разделами



Распределение памяти динамическими разделами

Функции ОС для реализации метода MVT OS/360 (ЕС ЭВМ):

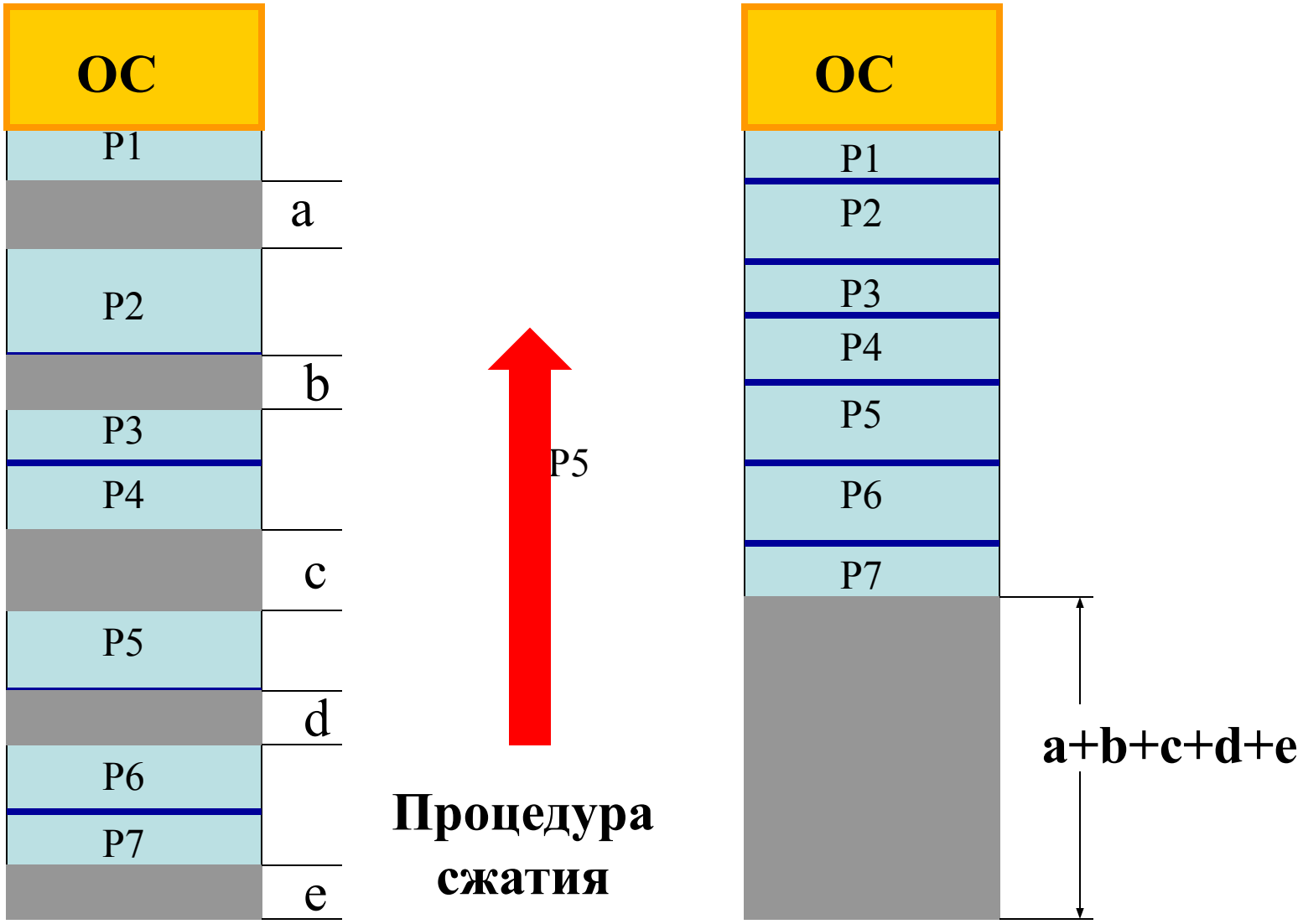
- ведение таблиц свободных и занятых областей ОП с указанием начального адреса и размера (поддерживается переменное число разделов переменной длины);
- при создании нового процесса просмотр таблиц и выбор раздела, достаточного для размещения процесса (наименьший и достаточный из свободных);
- загрузка процесса в выделенный раздел и корректировка таблиц свободных и занятых областей основной памяти;
- после завершения процесса корректировка таблиц свободных и занятых областей.

Распределение памяти динамическими разделами

Достоинства: большая гибкость по сравнению с фиксированными разделами.

Недостаток: внешняя фрагментация – после многократной загрузки/выгрузки процессов образуется множество мелких свободных участков памяти, в которых нельзя разместить новый процесс

3.3.4. Распределение памяти перемещаемыми разделами



Распределение памяти перемещаемыми разделами

1. **Перемещение всех занятых участков в сторону старших или младших адресов при каждом завершении процесса или для вновь создаваемого процесса в случае отсутствия раздела достаточного размера.**
2. **Коррекция таблиц свободных и занятых областей.**
3. **Изменение адресов команд и данных, к которым обращаются процессы при их перемещении в памяти за счет использования относительной адресации.**
4. **Аппаратная поддержка процесса динамического преобразования относительных адресов в абсолютные адреса основной памяти.**
5. **Защита памяти, выделяемой процессу, от взаимного влияния других процессов.**

Достоинства распределения памяти перемещаемыми разделами: эффективное использование оперативной памяти, исключение внутренней и внешней фрагментации. Недостаток: дополнительные накладные расходы ОС.

Распределение памяти перемещаемыми разделами

Уплотнение **может выполняться либо при** ***каждом завершении процесса*****, либо только тогда, когда для** ***вновь создаваемого процесса нет свободного раздела достаточного размера*****.**

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

Так как ***программа перемещается по оперативной памяти в ходе своего выполнения*****, то в данном случае** ***невозможно выполнить настройку адресов с помощью перемещающего загрузчика*****.**

Здесь более подходящим оказывается ***динамическое преобразование адресов*****.**

Распределение памяти перемещаемыми разделами

При использовании *фиксированной схемы* распределения *процесс всегда будет назначаться одному и тому же разделу памяти* после его выгрузки и последующей загрузки в память.

Это позволяет использовать *простейший загрузчик*, *замещающий при загрузке процесса все относительные ссылки абсолютными адресами памяти*, определенными на основе базового адреса загруженного процесса.

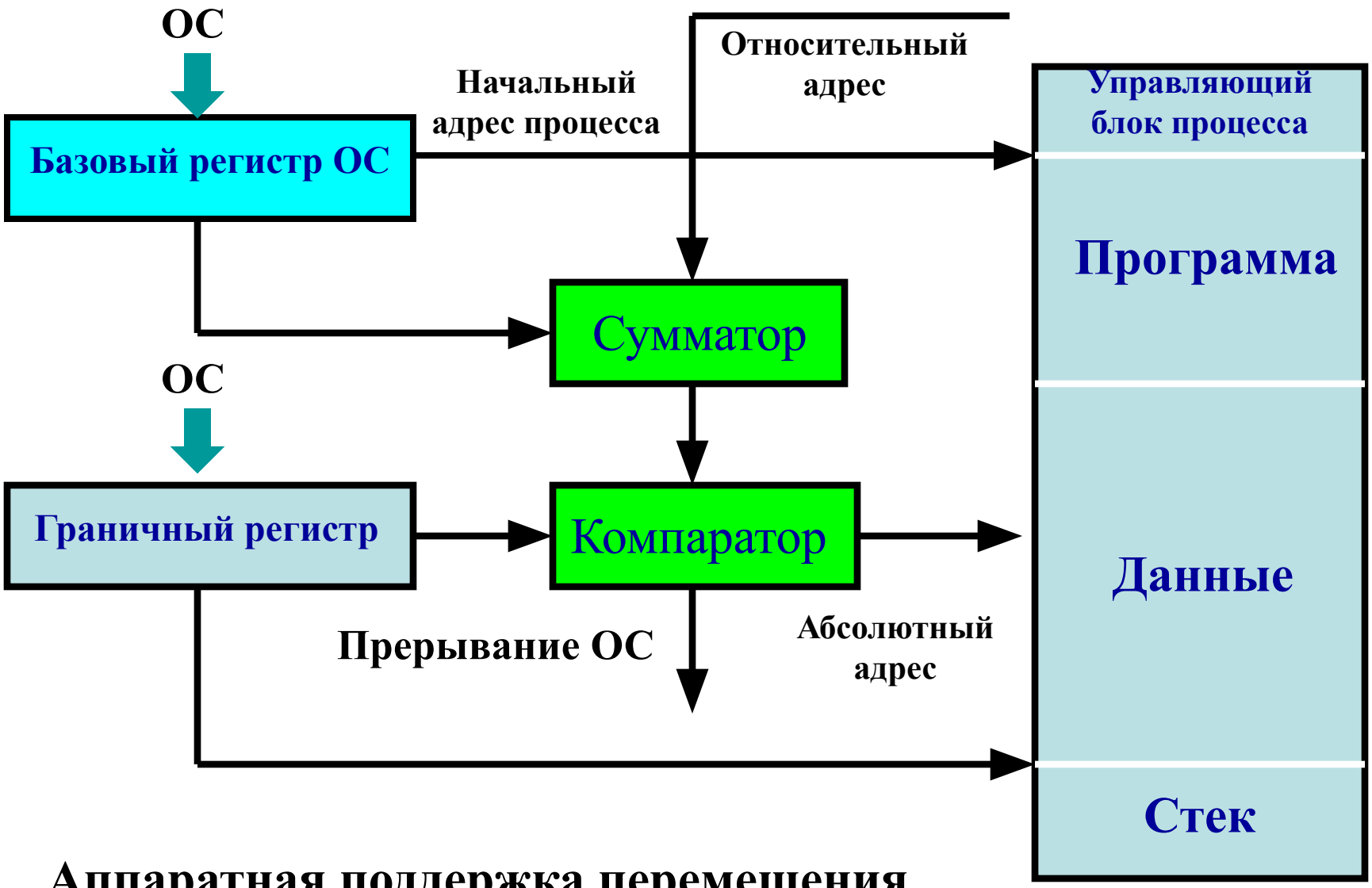
***Ситуация усложняется*, если размеры разделов равны (или неравны) и существует единая очередь процессов, тогда процесс по ходу работы может занимать разные разделы. Такая же ситуация возможна и при динамическом распределении.**

Распределение памяти перемещаемыми разделами

В этих случаях *расположение команд и данных*, к которым обращается процесс, *не является фиксированным и изменяется всякий* раз при выгрузке, загрузке или перемещении процесса.

Для *решения этой проблемы в программах используются относительные адреса*. Это означает, что *все ссылки на память в загружаемом процессе даются относительно начала этой программы*.

Таким образом, для *корректной работы программы требуется аппаратный механизм*, который бы транслировал относительные адреса в физические в процессе выполнения команды, которая обращается к памяти.



Аппаратная поддержка перемещения

Распределение памяти перемещаемыми разделами

Когда процесс переходит в состояние выполнения, в специальный регистр процесса, называемый базовым, загружается начальный адрес процесса в основной памяти.

Кроме того, используется «граничный» (bounds) регистр, в котором содержится адрес последней ячейки программы.

Эти значения заносятся в регистры при загрузке программы в основную память.

При выполнении процесса относительные адреса в командах обрабатываются процессором в два этапа.

Сначала к относительному адресу прибавляется значение базового регистра для получения абсолютного адреса.

Распределение памяти перемещаемыми разделами

Затем полученный абсолютный адрес сравнивается со значением в граничном регистре.

Если полученный абсолютный адрес принадлежит данному процессу, команда может быть выполнена.

В противном случае генерируется соответствующее данной ошибке прерывание.

3.4. Виртуальная память

3.4.1. Методы структуризации виртуального адресного пространства

1962 г. – Kilburn T. и др. “One –Level Storage System”

Методы реализации виртуальной памяти:

1. **Страничная виртуальная память – организует перемещение данных между ОП и диском страницами – частями виртуального адресного пространства *фиксированного и сравнительно небольшого размера*.**
2. **Сегментная виртуальная память предусматривает перемещение данных сегментами – *частями виртуального адресного пространства произвольного размера, полученными с учетом смыслового значения данных*.**
3. ***Сегментно-страничная виртуальная память использует двухуровневое деление*: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы. Единицей перемещения данных является страница.**
4. **Для временного хранения сегментов и страниц на диске отводится специальная область – *страничный файл или файл подкачки* (paging file).**

3.4. Виртуальная память

3.4.1. Методы структуризации виртуального адресного пространства

Когда используется виртуальная память, виртуальные адреса не передаются напрямую шиной памяти.

Вместо этого они передаются *диспетчеру памяти* (MMU - Memory Management Unit), который отображает виртуальные адреса на физические адреса памяти. Диспетчер памяти может быть частью микросхемы процессора, как обычно и бывает чаще всего.

Но логически он может быть и отдельной микросхемой, как было в недавнем прошлом.

3.4. Виртуальная память

3.4.1. Методы структуризации виртуального адресного пространства

Размер страничного файла в современных ОС является настраиваемым параметром, который выбирается администратором системы для достижения компромисса между уровнем программирования и быстродействием системы.

3.4.2. Страничная организация виртуальной памяти

При страничной организации виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые *виртуальными страницами* (Virtual pages).

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

***Вся оперативная память машины также делится на части такого же размера*, *называемые физическими страницами* (или блоками, или кадрами).**

3.4.2. Страничная организация виртуальной памяти

Размер страницы выбирается равным степени двойки: 1024, 2408, 4096 байт и т. д. Это позволяет упростить механизм преобразования адресов.

При создании процесса ОС загружает в операционную 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 находятся в смежных физических страницах.

3.4.2. Страничная организация виртуальной памяти

Для каждого процесса ОС создает таблицу страниц - информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

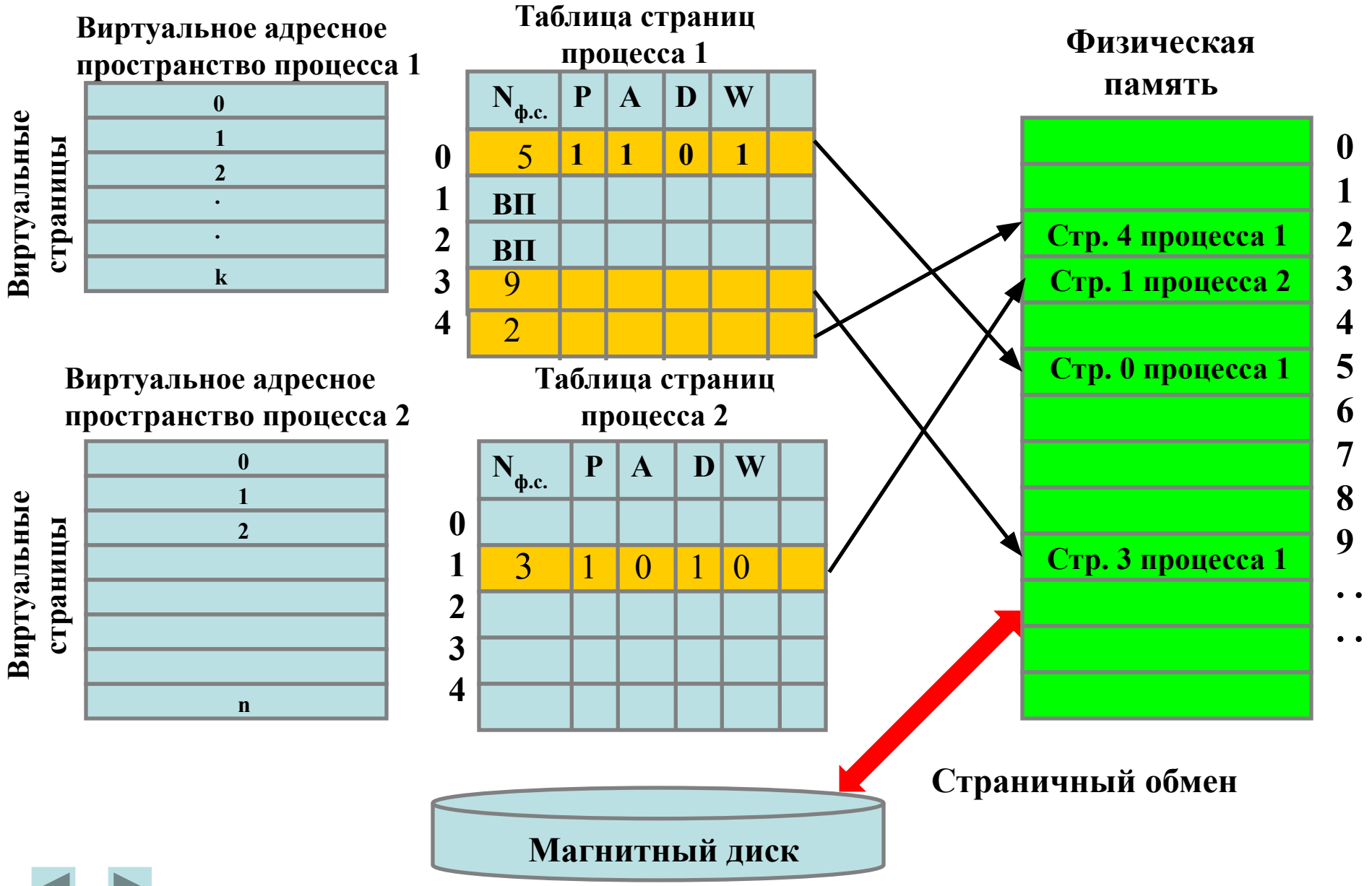
Запись таблицы (дескриптор страницы) включает следующую информацию:

- **номер физической страницы (N ф.с.), в которую загружена данная виртуальная страница;**
- **признак присутствия P, устанавливаемый в единицу, если данная страница находится в оперативной памяти;**

3.4.2. Страничная организация виртуальной памяти

- **признак модификации страницы D, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной страницы;**
- **признак обращения A к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице;**
- **другие управляющие биты, служащие, например для целей защиты или совместного использования памяти на уровне страниц.**

3.4.2. Страничная организация виртуальной памяти



3.4.2. Страничная организация виртуальной памяти

Перечисленные признаки в большинстве моделей процессов устанавливаются аппаратно схемами процессора при выполнении операций с памятью. Информация из таблицы страниц используется для решения вопроса о необходимости перемещения той или иной страницы между памятью и диском, а также для преобразования виртуального адреса в физический.

Сами таблицы страниц, так же как и описываемые ими страницы размещаются в оперативной памяти.

Поскольку процесс может использовать большой объем виртуальной памяти (например, в Windows 2000 он равен 2^{32} = 4 Гбайт), при использовании страницы объемом 4 Кбайт (2^{12}) потребуется 2^{20} записей в таблице страниц для каждого процесса.

3.4.2. Страничная организация виртуальной памяти

Выделять такое количество оперативной памяти под таблицы страниц нецелесообразно.

Для преодоления этой проблемы большинство схем виртуальной памяти хранит таблицы страниц не в реальной, а в виртуальной памяти.

Это означает, что сами таблицы страниц становятся объектами страничной организации.

При работе процесса как минимум часть его таблицы страниц должна располагаться в основной памяти, в том числе запись о странице, выполняющейся в настоящий момент.

Адрес таблицы страниц включается в контекст процесса.

3.4.2. Страничная организация виртуальной памяти

При активизации очередного процесса ОС загружает адрес его таблицы страниц в специальный регистр.

При каждом обращении к памяти выполняется поиск номера виртуальной страницы, содержащей требуемый адрес, затем по этому номеру определяется нужный элемент таблицы страниц, и из него извлекается описывающая страницу информация.

Далее анализируется признак присутствия, и если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический.

Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит страничное прерывание.

3.4.2. Страничная организация виртуальной памяти

Выполняющийся процесс переводится в состояние ожидания, и активизируется процесс из очереди процессов, находящихся в состояние готовности.

Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается ее загрузить в оперативную память.

Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно.

Если же свободных страниц нет, то на основании принятой в данной системе стратегии замещения страниц решается вопрос о том, какую страницу следует выгрузить из оперативной памяти.

После того как выбрана страница, которая должна покинуть оперативную память, обнуляется ее бит присутствия и анализируется ее признак модификации.

3.4.2. Страничная организация виртуальной памяти

Если удаляемая страница за время последнего требования в оперативной памяти была модифицирована, то ее новая версия должна быть переписана на диск.

Если нет, то принимается во внимание, что на диске уже имеется предыдущая копия этой виртуальной страницы, и никакой записи на диск не производится.

Физическая страница объявляется свободной.

Из соображений безопасности в некоторых системах освобождаемая страница обнуляется, чтобы невозможно было использовать содержимое выгруженной страницы.

Для хранения информации о положении вытесненной страницы в страничном файле ОС может использовать специальные поля таблицы страниц.

3.4.2. Страничная организация виртуальной памяти

Виртуальный адрес при страничном распределении может быть представлен в виде пары (P, S_v), где P - номер виртуальной страницы процесса (нумерация страниц начинается с 0), а S_v - смещение в пределах виртуальной страницы.

Физический адрес также может быть представлен в виде пары (N, S_f), где N - номер физической страницы, $a.S_f$ - смещение в пределах физической страницы.

Задача подсистемы виртуальной памяти состоит в отображении пары значений (P, S_v) в пару (N, S_f).

Объем страницы как виртуальной, так и физической выбирается равным степени двойки – 2^k (k = 8 и более).

Отсюда следует, что смещение S_v и S_f может быть получено отделением k младших разделов в двоичной записи виртуального и соответственно физического адреса страниц.

При этом оставшиеся старшие разделы адреса представляют собой двоичную запись номеров виртуальной и соответственно физической страницы. Дополнив эти номера к нулям, можно получить начальный адрес виртуальной и физической страниц.

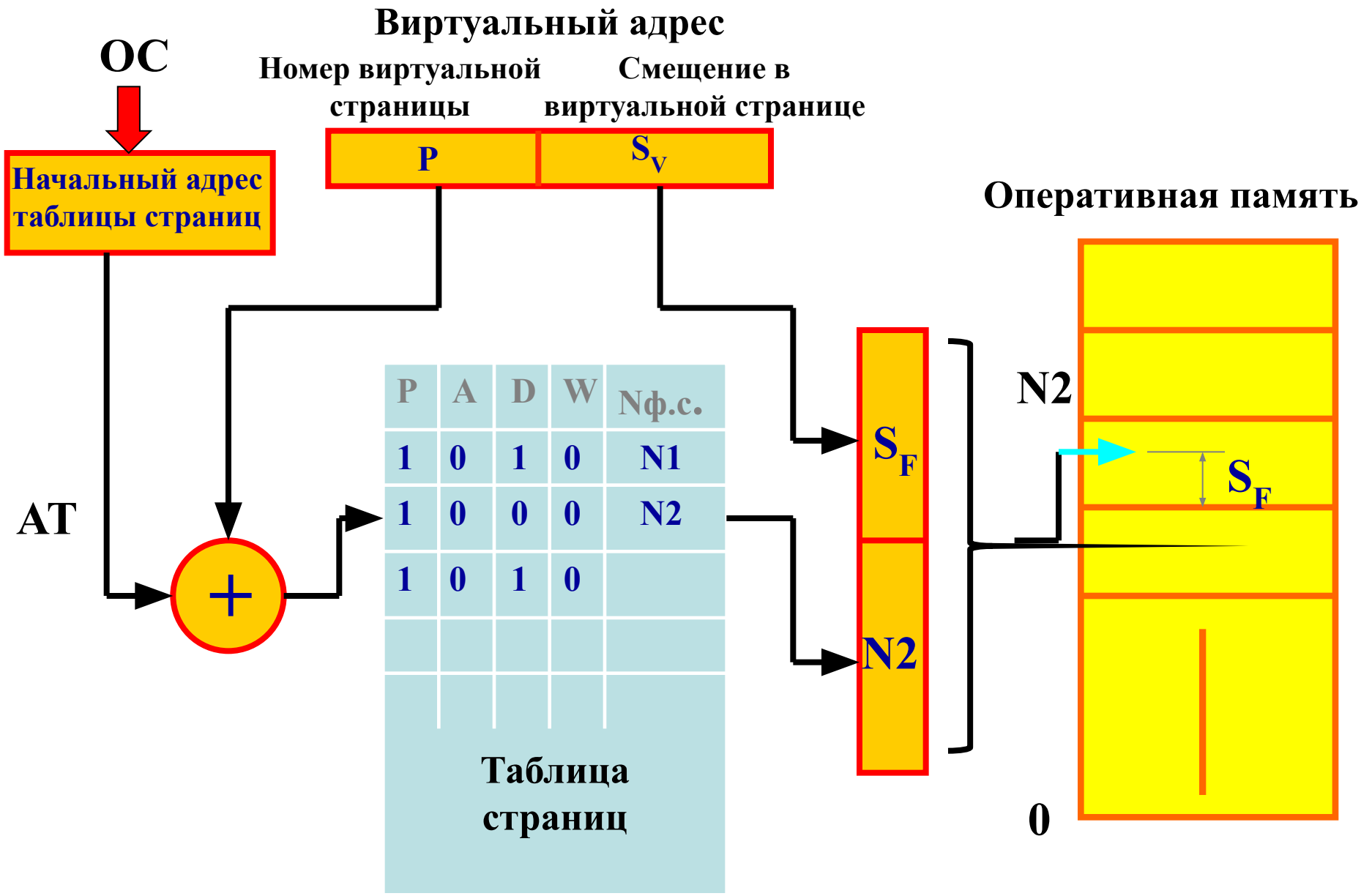
Второе свойство - линейность адресного пространства виртуальной и физической страницы - приводит к тому, что S_f =Sv. Отсюда следует простая схема преобразования виртуального адреса в физический.

3.4.2. Страничная организация виртуальной памяти

При обращении к памяти по некоторому виртуальному адресу (P, S_v) аппаратные схемы процессора выполняют следующие действия:

1. **Из специального регистра процессора извлекается начальный адрес АТ таблицы страниц активного процесса. С помощью сумматора S определяется адрес нужной записи в таблице страниц.**
2. **Считывается номер соответствующей физической страницы - N.**
3. **К номеру физической страницы присоединяется смещение S_v.**

В итоге полученный физический адрес оперативной памяти представляется парой значений (N, S_f).

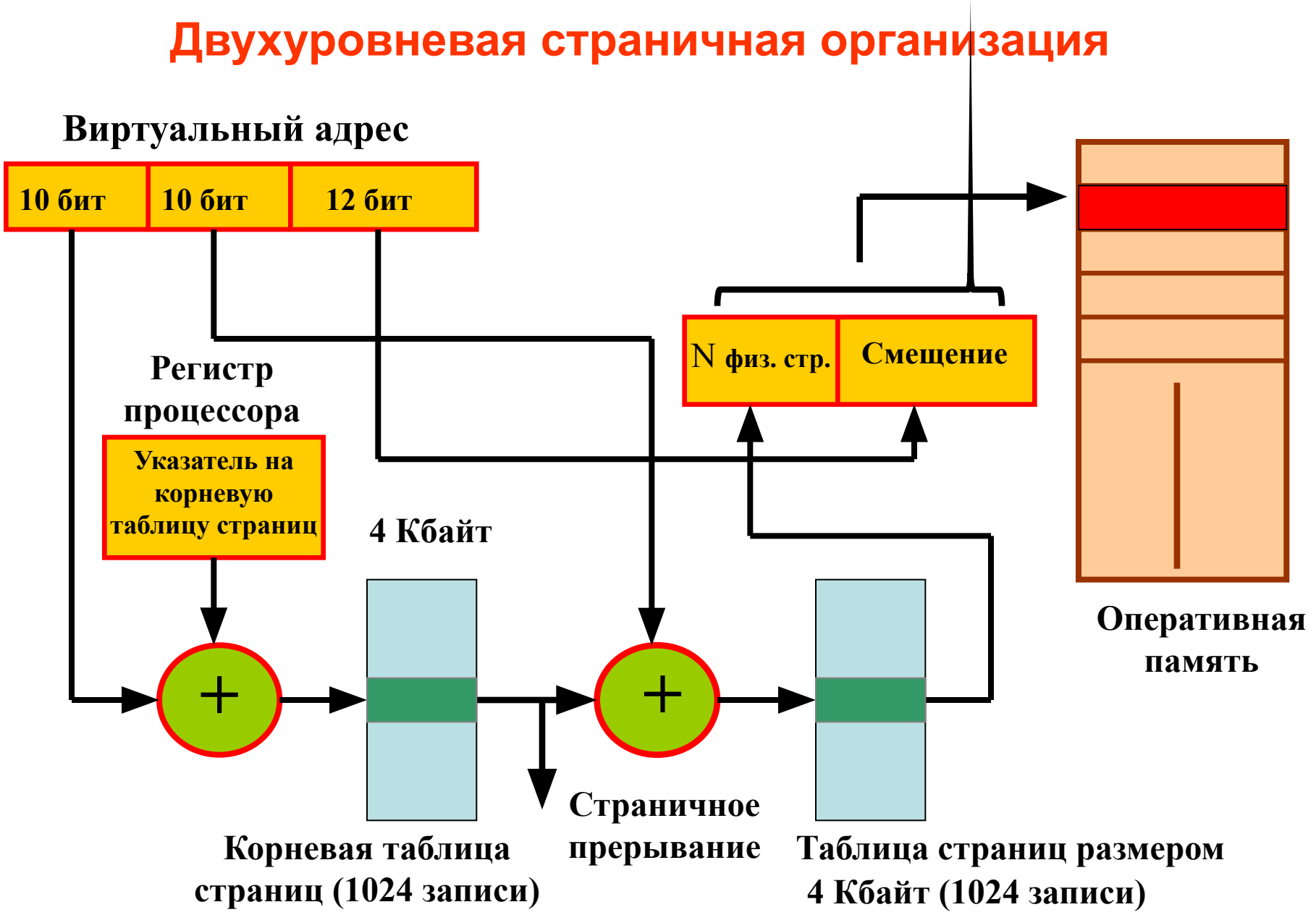

Виртуальный адрес
ОС
Номер виртуальной страницы
Смещение в виртуальной странице
Начальный адрес таблицы страниц
P
S_V
Оперативная память
AT
P A D W Nф.с.
1 0 1 0 N1
1 0 0 0 N2
1 0 1 0
Таблица страниц
S_F
N2
N2
S_F
0

3.4.3. Оптимизация функционирования страничной виртуальной памяти

Методы повышения эффективности функционирования страничной виртуальной памяти:

1. **Структуризация виртуального адресного пространства, например, двухуровневая (типичная для 32-битовой адресации).**
2. **Хранение активной части записей таблицы страниц в высокоскоростном КЭШе или буфере быстрого преобразования адреса (translation lookaside buffer – TLB).**
3. **Выбор оптимального размера страниц.**
4. **Эффективное управление страничным обменом, использование оптимальных алгоритмов замены страниц.**

Двухуровневая страничная организация



Двухуровневая страничная организация

При такой схеме имеется каталог таблиц страниц, в котором каждая запись указывает на таблицу страниц.

Обычно максимальный размер таблицы страниц определяется условием ее размещения в одной странице (такой подход используется в процессоре Pentium).

При 32-битовой адресации виртуальное адресное пространство пользовательского процесса может составлять 2^{32} = 4 Гбайт.

При объеме страницы 2^{12} = 4 Кбайт в этом пространстве размещается $2^{32}/2^{12} = 2^{20}$ страниц.

Таким образом, пользовательская таблица страниц будет иметь 2^{20} 4-байтных записей общим объемом 4 Мбайт.

Двухуровневая страничная организация

Разместить такие таблицы для нескольких процессов в ОП нереально. В двухуровневой схеме это и не требуется. В *основной памяти постоянно находится корневая таблица*, содержащая 1024 записи, указывающие на начальный адрес пользовательской таблицы страниц (ее объем, как указано выше 4 Мбайт).

Указание на начальный адрес корневой таблицы (активного процесса) заносится в регистр процессора.

Первые 10 бит виртуального адреса используются для индексации в корневой таблице для поиска записей о странице пользовательской таблицы.

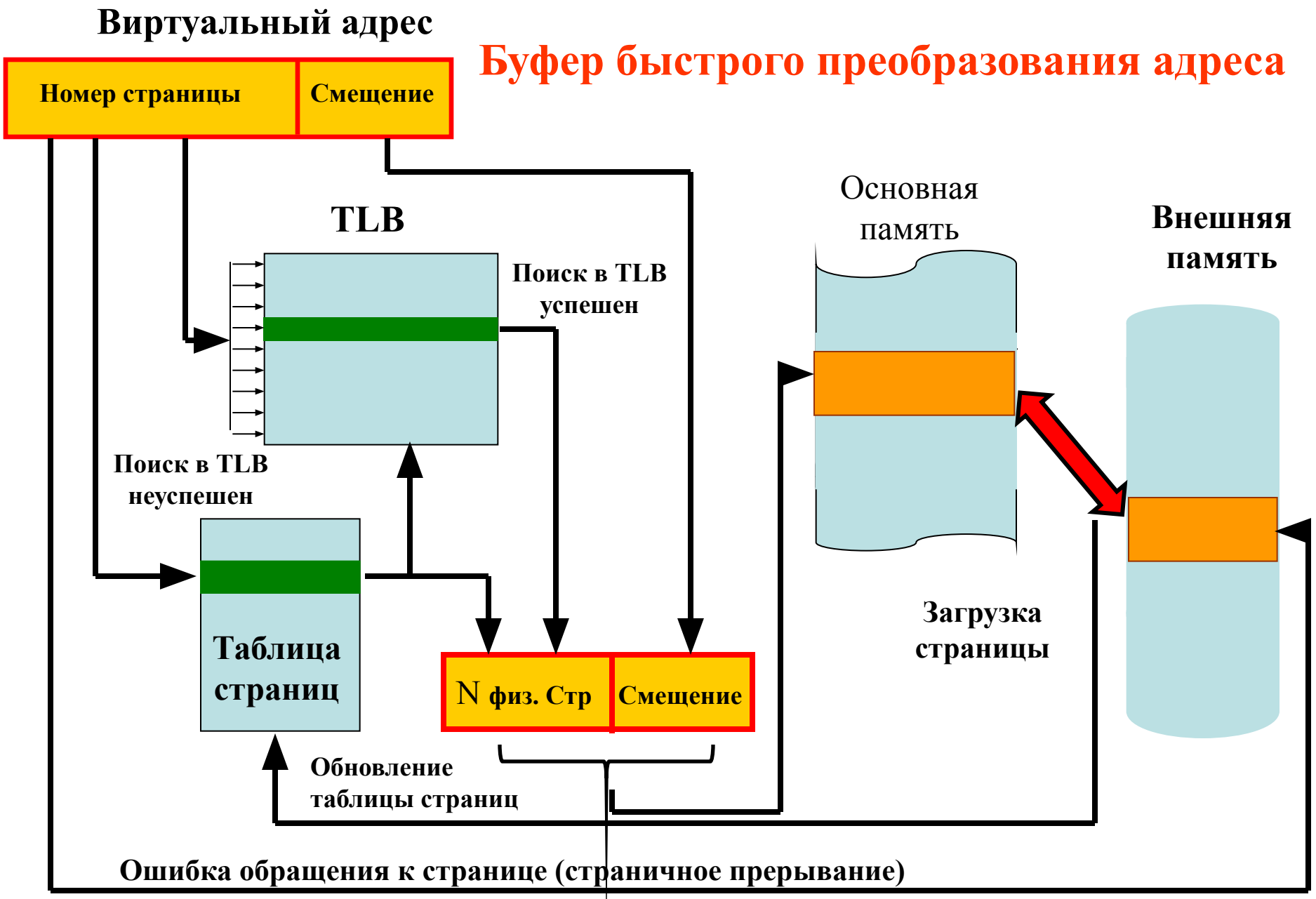
Если страница находится в ОП, то следующие 12 бит виртуального адреса используются для задания смещения в физической странице ОП.

Двухуровневая страничная организация

В противном случае генерируется страничное прерывание, но уже из-за отсутствия нужной страницы процесса в ОП.

Двухуровневая схема, сокращая объем памяти для хранения таблицы страниц, в общем случае замедляет преобразование виртуального адреса за счет большего числа возможных страничных прерываний.

Буфер быстрого преобразования адреса



Буфер быстрого преобразования адреса

Большинство реально применяющихся схем виртуальной памяти используют специальный высокоскоростной кэш для записей таблицы страниц, который обычно называют буфером быстрой трансляции адресов - TLB.

Этот кэш функционирует так же, как и обычный кэш памяти, и содержит те записи таблицы страниц, которые использовались последними.

Получив виртуальный адрес, процессор просматривает TLB. Если требуемая запись найдена, процессор получает адрес физической страницы и формирует реальный адрес.

Буфер быстрого преобразования адреса

Если запись в TLB не найдена, то процессор использует номер виртуальной страницы в качестве индекса для таблицы страниц процесса и просматривает соответствующую запись.

Если бит присутствия в ней установлен, значит, искомая страница находится в основной памяти, и процессор получает номер физической страницы из записи таблицы страниц, а затем формирует реальный физический адрес.

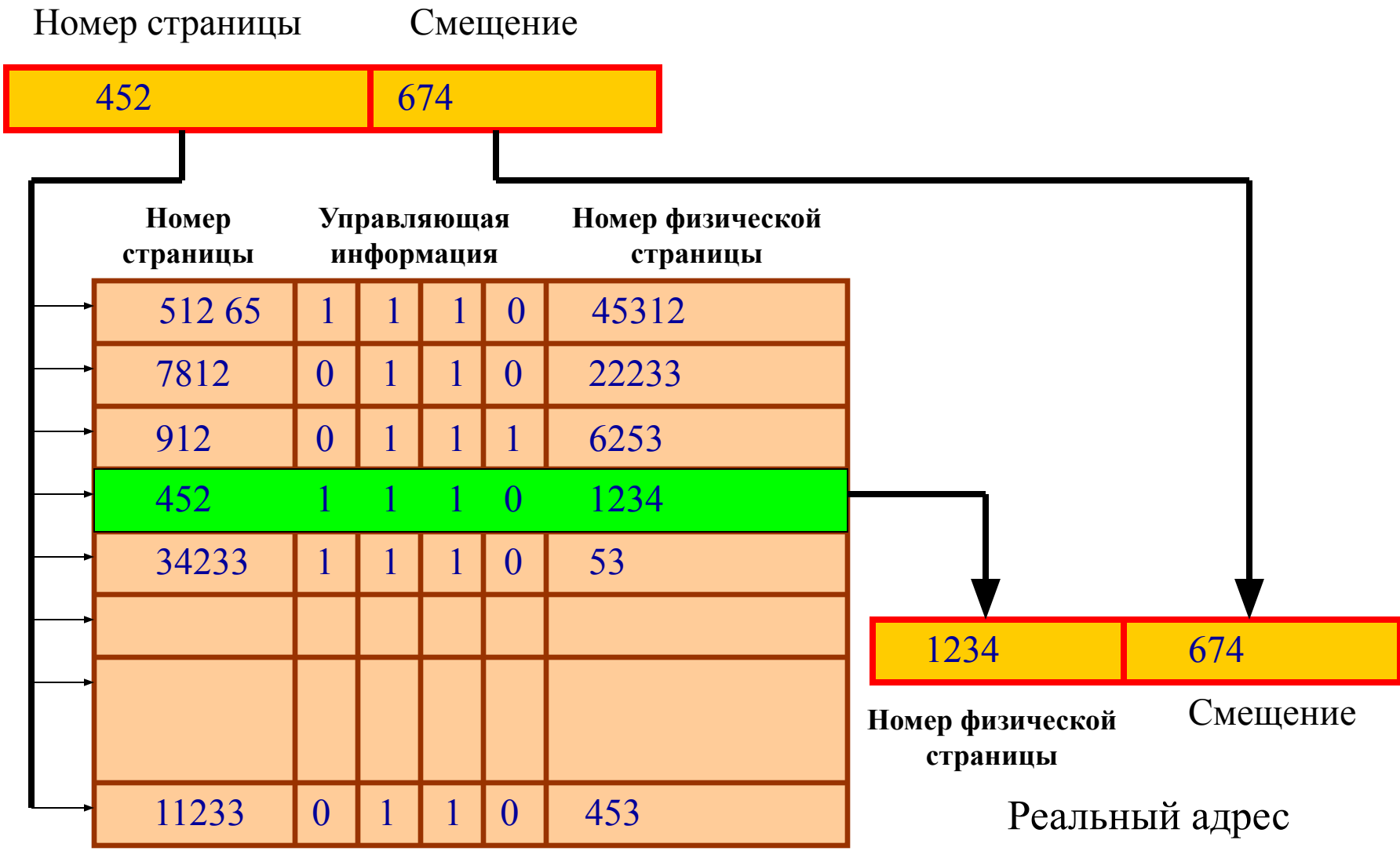
Одновременно вносится использованная запись таблицы страниц в TLB.

Буфер быстрого преобразования адреса

Если бит присутствия данной виртуальной страницы не установлен, это означает, что искомой страницы в оперативной памяти нет.

В этом случае процессор генерирует сигнал страничного прерывания, активизирующий операционную систему, которая загружает требуемую страницу в оперативную память и обновляет таблицу страниц.

Ассоциативное отображение



TLB

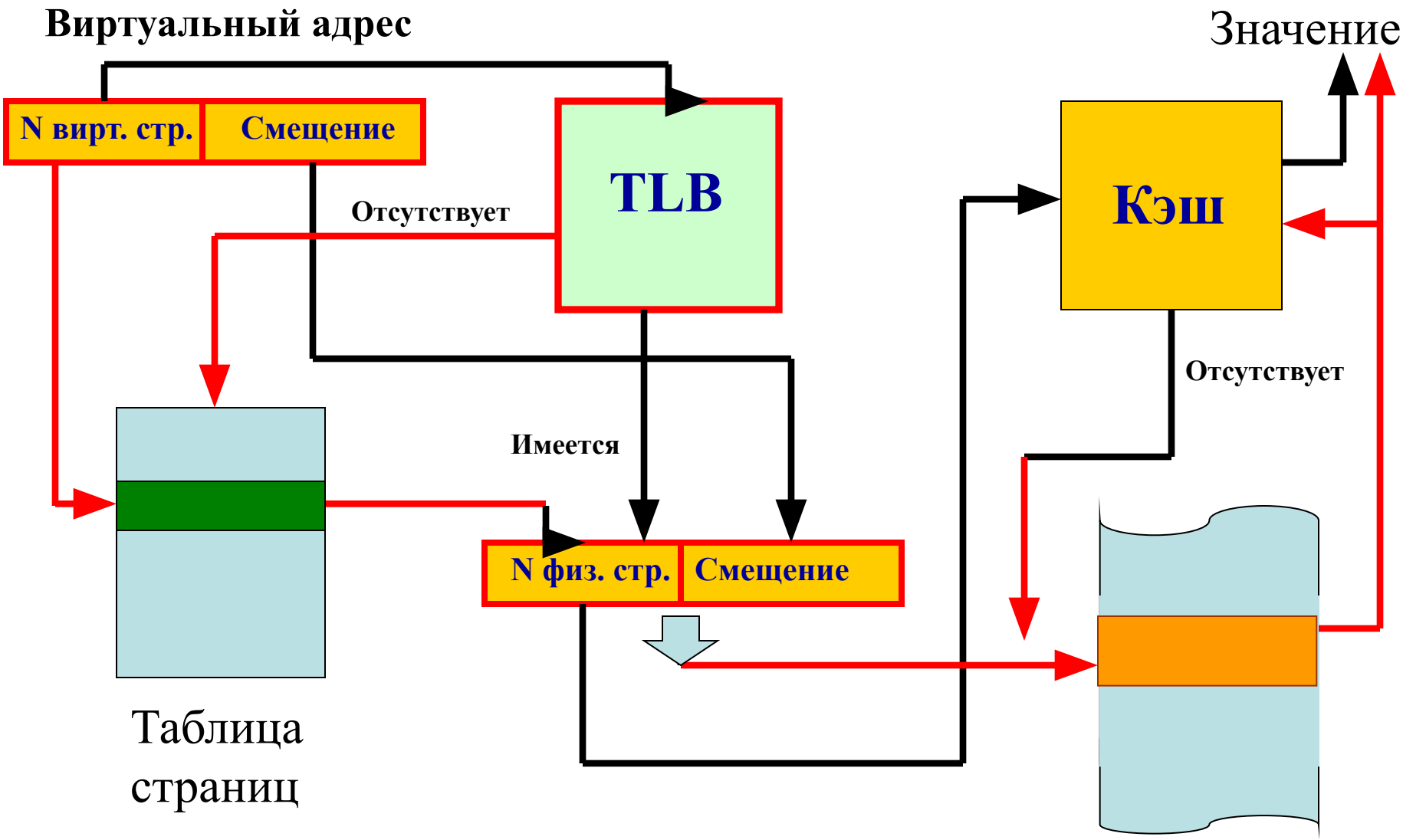
Ассоциативное отображение

Каждая запись TLB должна наряду с полной информацией из записи таблицы страниц включать также номер виртуальной страницы.

Процессор аппаратно способен одновременно опрашивать все записи TLB для определения того, какая из них соответствует заданному номеру страницы.

Такой подход известен как *ассоциативное отображение* (associative mapping), в отличие от прямого отображения, или индексирования, применяемого для поиска в таблице страниц.

Конструкция TLB должна также предусматривать способ организации записей в кэш и принятия решения о том, какая из старых записей должна быть удалена при внесении в кэш новой записи.



Взаимодействие кэша основной памяти и TLB

Взаимодействие кэша основной памяти и TLB

Механизм виртуальной памяти должен взаимодействовать с кэшем оперативной памяти (кроме TLB).

Сначала происходит обращение к TLB для выяснения, имеется ли в нем соответствующая запись таблицы страниц.

При положительном результате путем объединения номера физической страницы, получаемого из TLB, и смещения генерируется физический адрес.

Если требуемой записи в TLB нет, она выбирается из таблицы страниц.

Взаимодействие кэша основной памяти и TLB

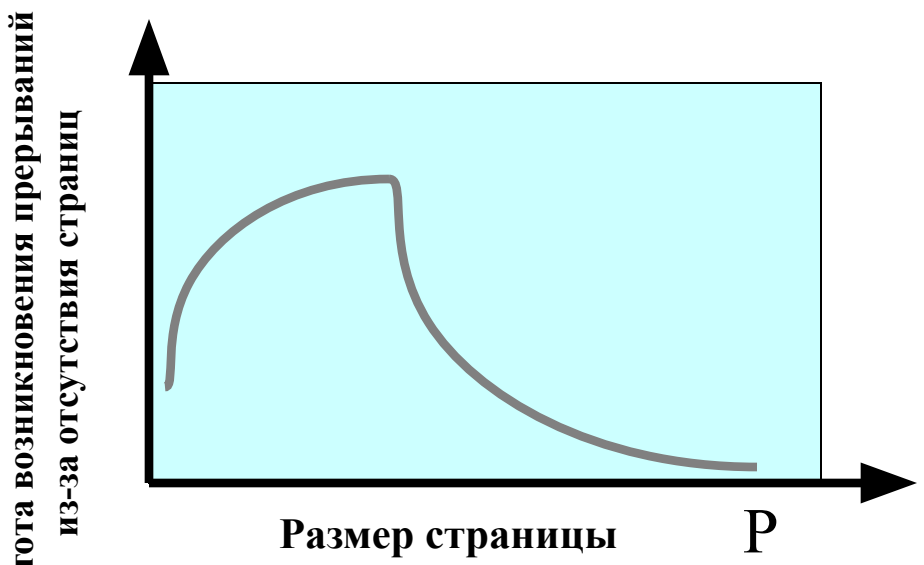
После получения физического адреса в обеих ситуациях выполняется обращение к кэшу для выяснения, не содержится ли в нем блок с требуемым физическим адресом.

Если ответ положительный, то требуемое значение (код или данные) передается процессору.

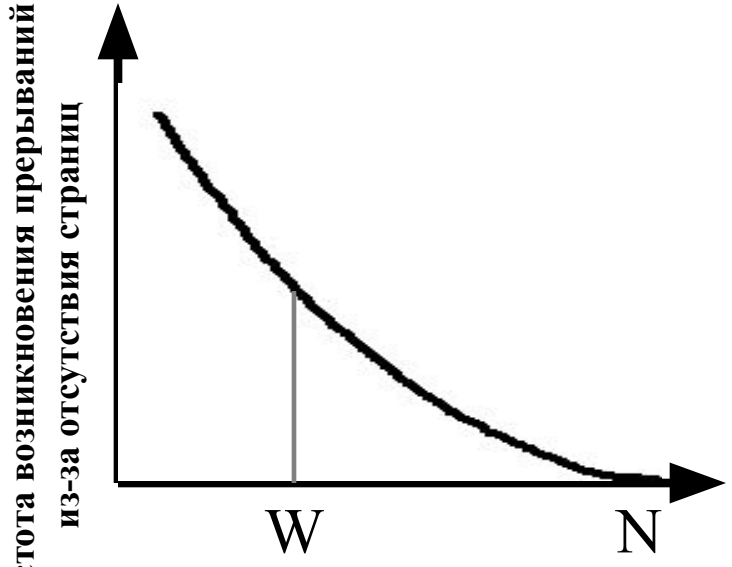
В противном случае производится выборка слова из основной памяти и обновляется содержимое кэша основной памяти.

Оптимальный размер страниц

1. С уменьшением размера страницы уменьшается внутренняя фрагментация.
2. С уменьшением размера страницы увеличивается объем страничных таблиц и следовательно накладные расходы на работу виртуальной памяти.
3. С увеличением размера страниц повышается скорость работы диска.
4. Частота страничных прерываний нелинейно зависит от размера страниц



P – размер процесса в страницах



Количество выделенных физических страниц

N – общее количество страниц процесса

W – размер рабочего множества

Оптимальный размер страниц

Внутренняя фрагментация уменьшается с уменьшением размера страницы.

Однако, чем меньше страницы, тем больше их требуется для процесса, что означает увеличение размера таблицы страниц.

Для больших программ в загруженной многозадачной среде это приведет к тому, что часть страничных таблиц активных процессов будет находиться в виртуальной памяти, и при отсутствии страницы будет возникать двойное прерывание: первое - для получения требуемой записи из таблицы страниц, второе -для получения доступа к требуемой странице процесса.

Оптимальный размер страниц

Такое двойное прерывание существенно снизит производительность виртуальной памяти. Кроме того, следует учитывать и факт повышения скорости работы диска при передаче больших блоков данных.

Таким образом, страницы небольшого размера нецелесообразны, поскольку уменьшение внутренней фрагментации в этом случае не столь значительно по сравнению со снижением производительности виртуальной памяти.

Проблема выбора размера страницы усложняется еще и тем, что размер страницы влияет и на частоту возникновения прерывания из-за отсутствия страницы в основной памяти.

Управление страничным обменом

Задачи управления страничным обменом:

- **- когда передавать страницу в основную память;**
- **- где размещать страницу в физической памяти;**
- **- какую страницу основной памяти выбирать для замещения, если в основной памяти нет свободных страниц;**
- **- сколько страниц процесса следует загрузить в основную память;**
- **- когда измененная страница должна быть записана во вторичную память;**
- **- сколько процессов размещать в основной памяти.**

НАИМЕНОВАНИЕ	ВОЗМОЖНЫЕ АЛГОРИТМЫ
Стратегия выборки (когда?)	По требованию, предварительная выборка
Стратегия размещения (где?)	Первый подходящий раздел для сегментной виртуальной памяти. Любая страница физической памяти для сегментно-страничной и страничной организации памяти.
Стратегия замещения (какие?)	Оптимальный выбор, дольше всех не использовавшиеся, первым вошел – первым вышел (FIFO), часовой, буферизация страниц.
Управление резидентным множеством (сколько?)	Фиксированный размер, переменный размер, локальная и глобальная области видимости.
Стратегия очистки (когда?)	По требованию, предварительная очистка
Управление загрузкой (сколько?) и приостановкой процессов	Рабочее множество, критерии L = S (среднее время между прерываниями = среднему времени обработки прерывания) и 50%

Управление страничным обменом

Стратегия выборки определяется, когда страница должна быть передана в основную память. Два основных варианта - по *требованию* и предварительно.

В первом случае страница передается в основную память только тогда, когда выполняется обращение к ячейке памяти, расположенной на этой странице.

В случае предварительной выборки загружается несколько страниц.

Управление страничным обменом

Такая выборка использует особенности работы дисковых устройств, заключающиеся в том, что несколько последовательно расположенных страниц загрузятся значительно быстрее, чем загрузка этих же страниц по одной в течение некоторого промежутка времени.

Стратегия размещения **определяет, где именно в физической памяти будут располагаться части процесса. Для систем, использующих сегментно-страничную или чисто страничную организацию виртуальной памяти, стратегия размещения не актуальна.**

Управление страничным обменом

В многопроцессорных системах с неоднородным доступом к памяти (различные расстояния между процессорами и модулями памяти) стратегия размещения становится очень важной и требует всестороннего исследования.

Стратегия замещения **определяет выбор страниц в основной памяти для замещения их загружаемыми из вторичной памяти страницами.**

Управление страничным обменом

Эта стратегия связана с решением следующих вопросов:

- **какое количество страниц в основной памяти должно быть выделено каждому активному процессу;**
- **должны ли замещаемые страницы относиться к одному процессу или в качестве кандидатов на замещение должны рассматриваться все страницы оперативной памяти;**
- **какие именно страницы из рассматриваемого множества следует выбрать для замещения.**

Управление страничным обменом

Все используемые *стратегии замещения* направлены на то, чтобы выгрузить страницы, обращений к которым в ближайшем будущем не последует.

Имеется ряд основных алгоритмов, используемых для выбора замещаемой страницы:

- **оптимальный алгоритм;**
- **алгоритм дольше всех неиспользующейся страницы;**
- **алгоритм «первым вошел - первым вышел» (FIFO);**
- **часовой алгоритм и др.**

Управление страничным обменом

Оптимальный алгоритм **состоит в выборе замещения той страницы, обращение к которой будет через наибольший промежуток времени по сравнению со всеми остальными страницами (реализовать такой алгоритм невозможно, поскольку для этого системе требуется знать все будущие события).**

Алгоритм FIFO рассматривает физические страницы процесса как циклический буфер с циклическим удалением страниц из него. Замещается страница, находящаяся в памяти дольше других.

Управление страничным обменом

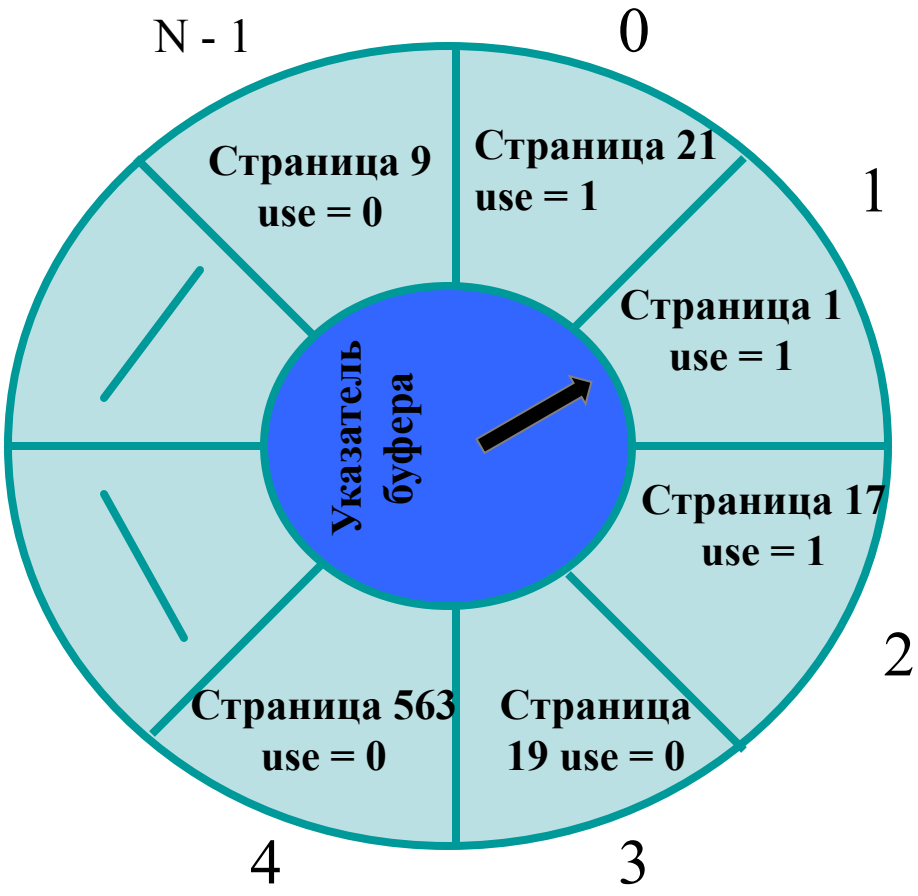
В простейшей схеме *часовой стратегии* с каждой физической страницей связан один бит, который называется битом использования.

Когда виртуальная страница загружается впервые в физическую страницу, бит использования переводится в 1.

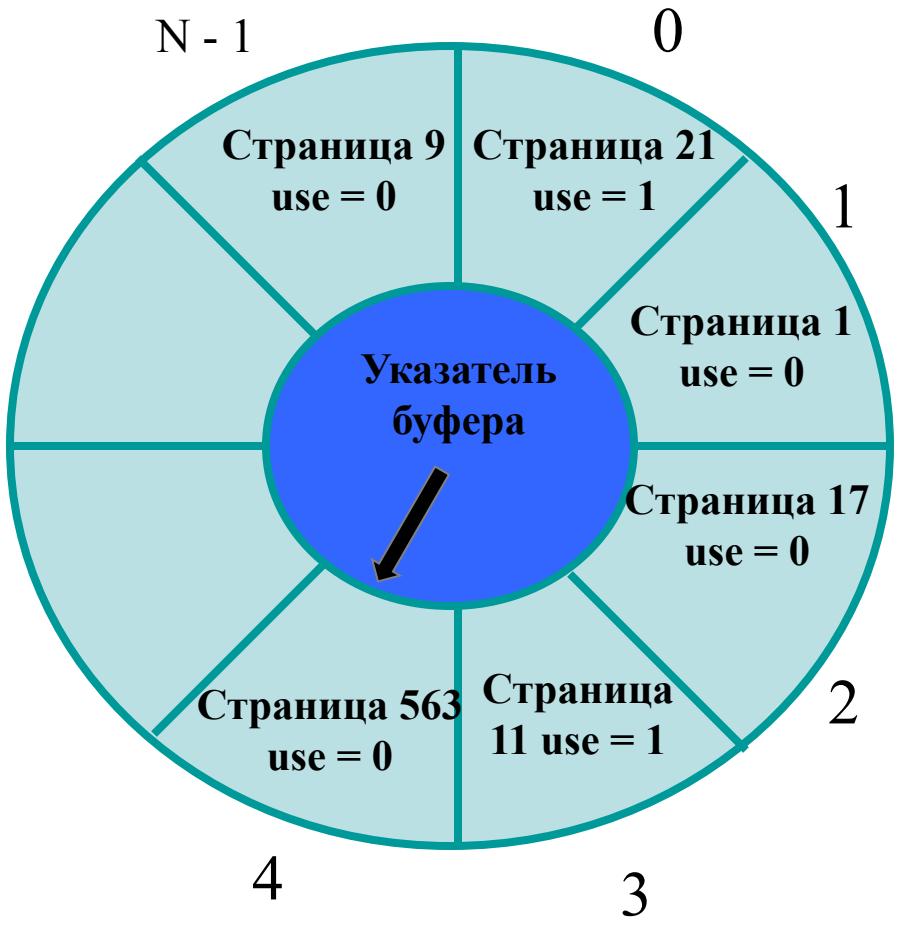
При последующих обращениях к странице, вызвавших прерывание из-за отсутствия страницы, этот бит устанавливается равным 1.

При работе алгоритма замещения множество страниц, являющихся кандидатами на замещение, рассматривается как циклический буфер, с которым связан указатель.

Часовая стратегия замещения



Состояние буфера перед замещением страниц



Состояние буфера после замещения страниц

Часовая стратегия замещения

При замещении страницы указатель перемещается к следующему кадру в буфере.

Когда наступает время замещения страницы, ОС сканирует буфер для поиска кадра, бит использования которого равен 0.

При этом когда в процессе поиска встречается кадр с битом использования, равным 1, он сбрасывается в 0.

Первый же встреченный кадр с нулевым битом использования выбирается для замещения.

Если все кадры имеют бит использования, равный 1, указатель совершает полный круг и возвращается к начальному положению, заменяя страницу в этом кадре.

Управление резидентным множеством

Эта задача включает решение следующих вопросов:

- **выбор размера резидентного множества;**
- **определение области видимости замещения.**

При использовании виртуальной памяти для подготовки процесса к выполнению нет необходимости размещать в оперативной памяти все его страницы.

Следовательно, ОС должна принять решение, какое количество страниц следует загрузить, т. е. сколько памяти выделить конкретному процессу.

Управление резидентным множеством

1. Чем меньше памяти выделяется процессу, тем большее количество процессов может одновременно находиться в памяти, тем больше степень мультипрограммирования.

2. При относительно небольшом количестве страниц процесса, размещенных в оперативной памяти, частота страничных прерываний будет достаточно велика.

В современных ОС используется два типа стратегий: *фиксированного* и *переменного распределения*.

В первом случае процессу выделяется фиксированное количество страниц основной памяти, в пределах которого он выполняется.

Управление резидентным множеством

Стратегия переменного распределения позволяет изменять во время работы процесса количество выделенных ему страниц оперативной памяти.

Использование стратегий переменного распределения связано с концепцией области видимости замещения.

Области видимости стратегии замещения можно классифицировать как локальную и глобальную.

Локальная стратегия замещения выбирает страницу только среди резидентных страниц того процесса, который стал причиной страничного прерывания.

Управление резидентным множеством

Глобальная стратегия замещения рассматривает в качестве кандидатов на замещение все незаблокированные страницы в основной памяти, независимо от принадлежности конкретной страницы тому или иному процессу.

Стратегия очистки является противоположностью стратегии выборки.

Ее задача состоит в определении момента, когда измененная страница должна быть записана во вторичною память.

Два ее основных метода - очистка по требованию и предварительная очистка.

Управление резидентным множеством

При очистке по требованию страница записывается во вторичную память только тогда, когда она выбирается для замещения. Предварительная очистка записывает модифицированные страницы до того, как потребуется занимаемая ими оперативная память, так что эти страницы могут записываться пакетами.

Улучшенный подход включает буферизацию страниц, что позволяет принять следующую стратегию: очищать только замещаемые страницы, но при этом разделить операции очистки и замещения.

Управление резидентным множеством

При использовании буферизации замещаемые страницы могут находиться в двух списках: модифицированных и немодифицированных страниц.

Страницы из списка модифицированных могут периодически записываться пакетами и переноситься в список немодифицированных.

Страница из списка не модифицированных страниц может быть удалена из него при обращении к ней, либо потеряна при загрузке в нее новой виртуальной страницы.

Управление загрузкой - это определение количества процессов, которые будут резидентными в основной памяти.

Управление резидентным множеством

Подход, известный под названием критерий L-S, был предложен Деннингом (Denning).

При этом подходе уровень многозадачности настраивается таким образом, чтобы среднее время между прерываниями равнялось среднему времени, требующемуся для обработки прерывания.

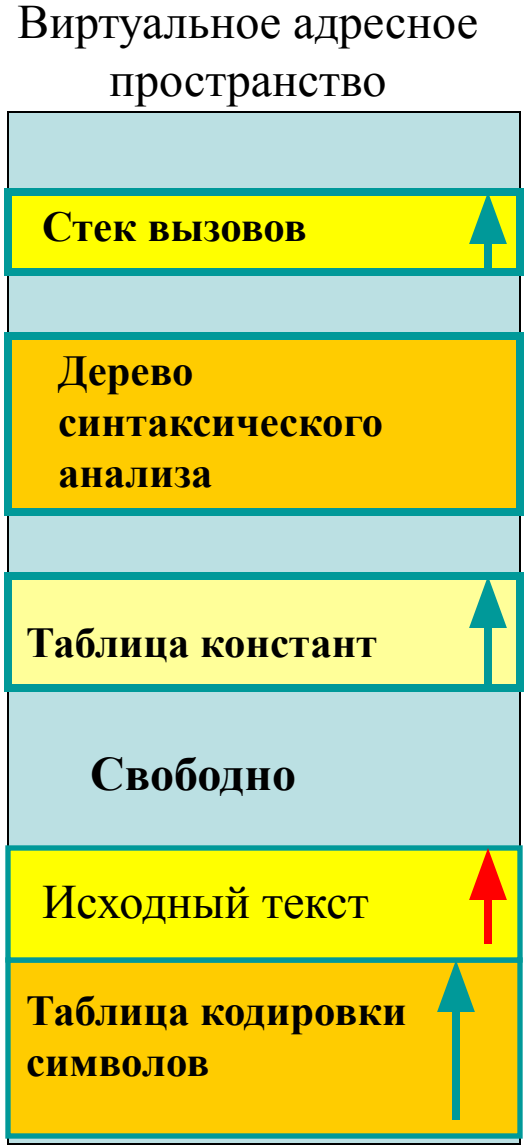
В результате исследований сделан вывод, что этот уровень многозадачности обеспечивает максимальную производительность процессора.

Управление резидентным множеством

При снижении степени многозадачности один или несколько резидентных процессов должны быть приостановлены и выгружены во вторичную память. Возможными вариантами выбора процесса для перемещения во внешнюю память могут быть:

- **процесс с наименьшим приоритетом;**
- **процесс, вызывающий прерывания;**
- **последний активированный процесс;**
- **процесс с минимальным резидентным множеством;**
- **наибольший процесс по числу занимаемых физических страниц памяти;**
- **процесс с максимальным остаточным временем исполнения.**

3.4.4. Сегментная организация виртуальной памяти



При компиляции возможно создание следующих сегментов:

1. Исходный текст, сохраненный для печати листинга программы.
2. Символьная таблица, содержащая имена и атрибуты переменных.
3. Таблица констант.
4. Дерево грамматического разбора, содержащее синтаксический анализ программы.
5. Стек, используемый для процедурных вызовов внутри компилятора.

Таблица кодировки символов достигла таблицы с исходным текстом

Сравнение страничной и сегментной организации памяти

Вопрос	Страничная	Сегментация
Нужно ли программисту знать о том, что используется эта техника?	Нет	Да
Сколько в системе линейных адресных пространств?	1	Много
Может ли суммарное адресное пространство превышать размеры физической памяти?	Да	Да
Возможно ли разделение процедур и данных, а также раздельная защита для них?	Нет	Да
Легко ли размещаются таблицы с непостоянными размерами?	Нет	Да
Облегчен ли совместный доступ пользователей к процедурам?	Нет	Да
Зачем была придумана эта техника?	Чтобы получить большое линейное адресное пространство без затрат на физическую память	Для разбиения программ и данных на независимые адресные пространства, облегчения защиты и совместного доступа

Операцион

Сегментная организация виртуальной памяти

Сегменты **- это независимые адресные пространства.**

Каждый сегмент содержит линейную последовательность адресов от 0 до некоторого максимума.

Различные сегменты могут быть различной длины.

Длины сегментов могут изменяться во время выполнения.

Поскольку каждый сегмент составляет отдельное адресное пространство, разные сегменты могут расти и сокращаться независимо друг от друга.

Сегментная организация виртуальной памяти

Чтобы определить адрес в такой сегментированной или двумерной памяти, программа должна указать адрес, состоящий из двух частей: *номер сегмента* и *адрес внутри сегмента*.

Максимальный размер сегмента определяется разрядностью виртуального адреса, например при 32-разрядном микропроцессоре он равен 2^{32} = 4 Гбайт.

При этом максимально возможное виртуальное адресное пространство представляет набор из N виртуальных сегментов (заметим, что общего для сегментов линейного виртуального адреса не существует).

Сегментная организация виртуальной памяти

Сегмент **- это *логический* объект, о чем программист знает и поэтому использует его как логический объект.**

Преимущества сегментной модели:

- **простота компоновки отдельно скомпилированных процедур (обращение к начальной точке процедуры осуществляется адресом вида (n, 0), где n - номер сегмента);**
- **легкость обеспечения дифференцируемого доступа к различным частям программы (например, запретить обращаться для записи в сегмент программы);**

Сегментная организация виртуальной памяти

- **простота организации совместного использования фрагментов программ различными процессами, например, библиотеки совместного доступа могут быть оформлены в виде отдельного сегмента, который может быть включен в виртуальное адресное пространство нескольких процессов.**

Сегментная организация виртуальной памяти

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

Для каждого загружаемого сегмента ОС подыскивает непрерывный участок свободной памяти достаточного размера.

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

Если во время выполнения процесса происходит обращение к отсутствующему в основной памяти сегменту, происходит прерывание.

Сегментная организация виртуальной памяти

Операционная система в данном случае работает аналогично подобному процессу в страничной виртуальной памяти.

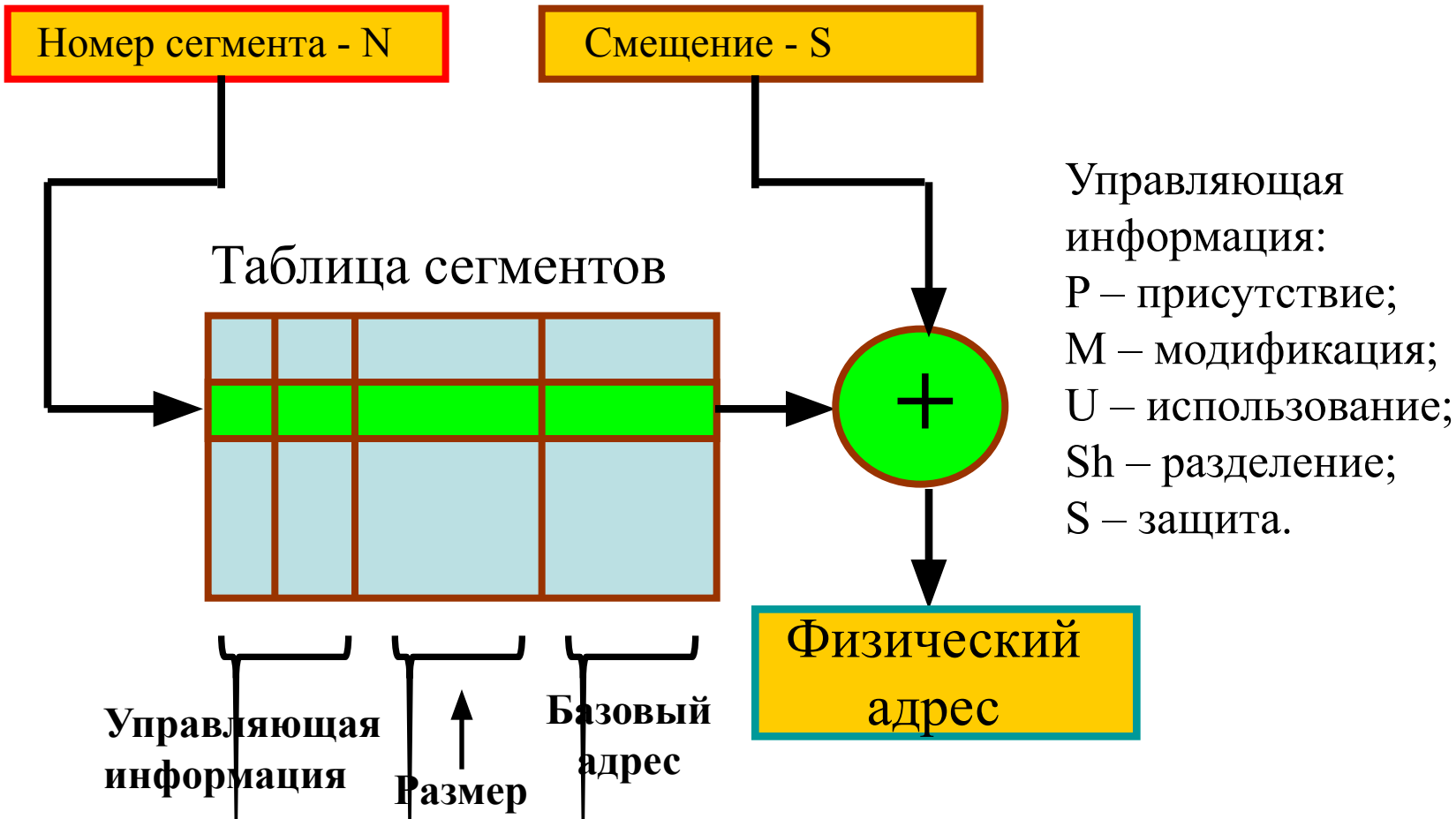
На этапе создания процесса во время загрузки его образа в оперативную память ОС создает таблицу сегментов процесса, аналогичную таблице страниц, в которой для каждого сегмента указывается:

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

Сегментная организация 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.

Виртуальный адрес



Недостатки сегментной организации: 1. Увеличение времени преобразования виртуального адреса в физический. 2. Избыточность перемещаемых данных (между диском и ОП). 3. Внешняя фрагментация памяти.

системы

Адрес

Физический адрес **получается сложением базового адреса сегмента, который определяется по номеру сегмента n из таблицы сегментов, и смещения S.**

Использование операции сложения вместо конкатенации замедляет процедуру преобразования виртуального адреса в физический.

Сегментно-страничная организация виртуальной памяти

Данный метод организации виртуальной памяти направлен на сочетание достоинств страничного и сегментного методов управления памятью.

В такой комбинированной системе адресное пространство пользователя разбивается на ряд сегментов по усмотрению программиста.

Каждый сегмент в свою очередь разбивается на страницы фиксированного размера, равные странице физической памяти. С точки зрения программиста, логический адрес в этом случае состоит из номера сегмента и смещения в нем. С позиции операционной системы смещение в сегменте следует рассматривать как номер страницы определенного сегмента и смещение в ней.

Сегментно-страничная организация виртуальной памяти

С каждым процессом связана одна таблица сегментов и несколько (по одной) на сегмент таблиц страниц.

При работе определенного процесса в регистре процессора хранится начальный адрес соответствующей таблицы сегментов.

Получив виртуальный адрес, процессор использует его часть, представляющую номер сегмента, в качестве индекса в таблице сегментов для поиска таблицы страниц данного сегмента.

После этого часть адреса, представляющая собой номер страницы, используется для поиска номера физической страницы в таблице страниц.

Сегментно-страничная организация виртуальной памяти

Затем часть адреса, представляющая смещение, используется для получения искомого физического адреса путем добавления к начальному адресу физической страницы.

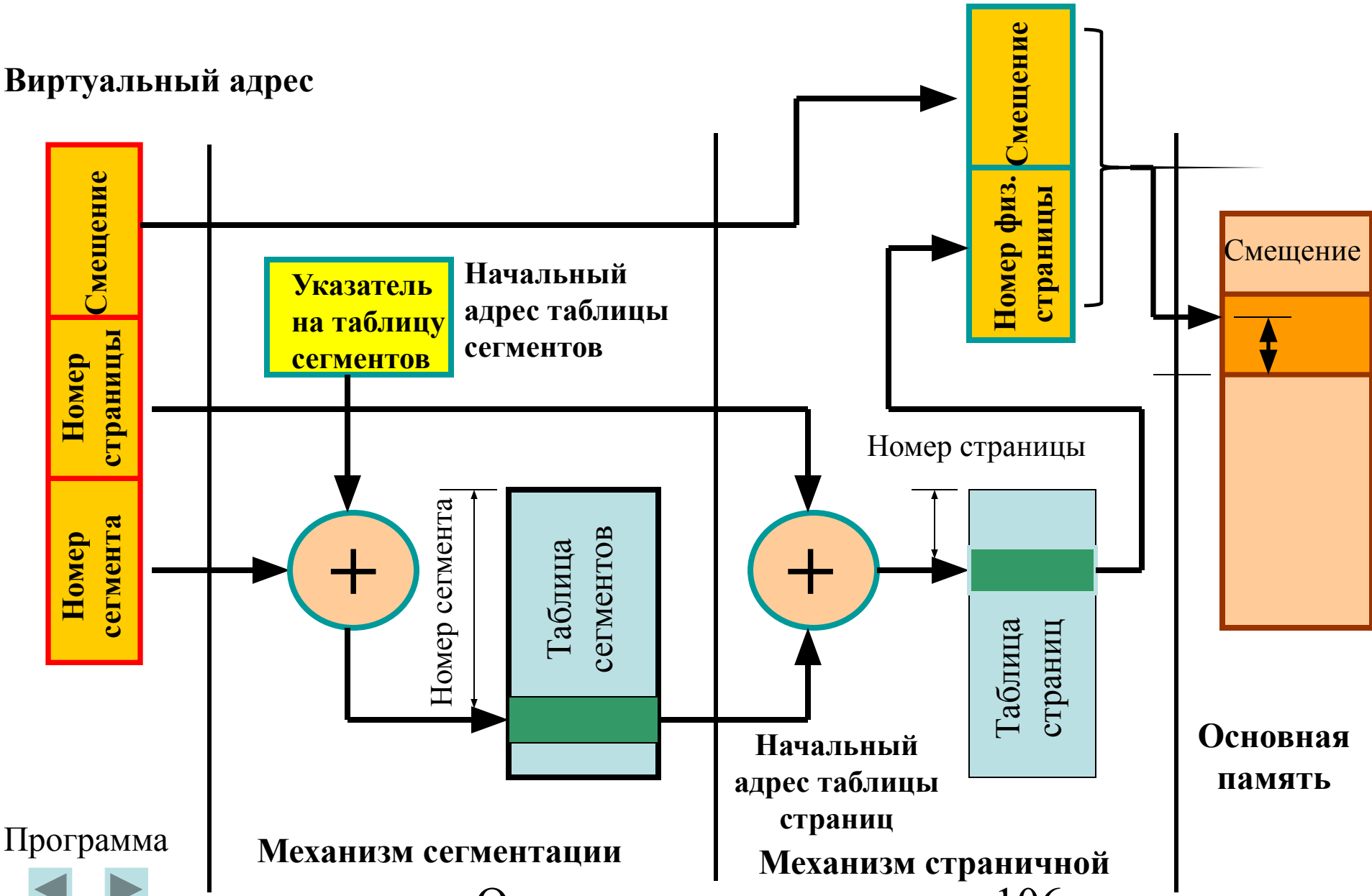
Сегментация удобна для реализации защиты и совместного использования сегментов разными процессами.

Поскольку каждая запись таблицы сегментов включает начальный адрес и значение длины, программа не в состоянии непреднамеренно обратиться к основной памяти за границами сегмента.

Сегментно-страничная организация виртуальной памяти

Для того чтобы отличить разделяемые сегменты от индивидуальных, записи таблицы сегментов содержат 1-битовое поле, имеющее два значения: shared (разделяемый) или private (индивидуальный).

Сегментно-страничная организация виртуальной памяти



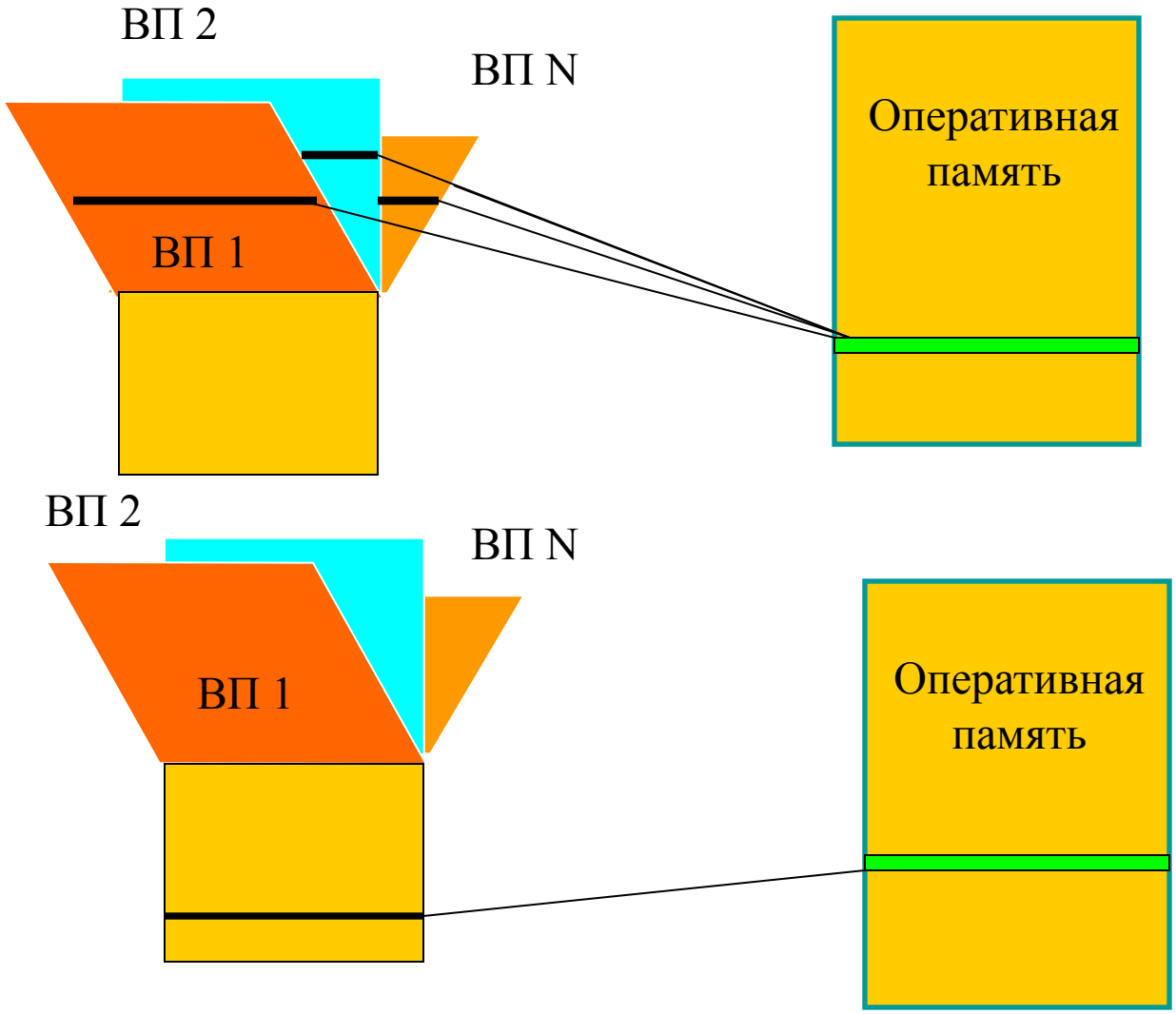
Способы создания разделяемого сегмента памяти

Для осуществления совместного использования сегмента он помещается в виртуальное адресное пространство нескольких процессов, при этом *параметры отображения этого сегмента настраиваются так, чтобы они соответствовали одной и той же области оперативной памяти* (делается это указанием одного и того же базового физического адреса сегмента).

Возможен и более экономичный для ОС метод создания разделяемого виртуального сегмента - *помещение его в общую часть виртуального адресного пространства, т. е. в ту часть, которая обычно используется для модулей ОС*.

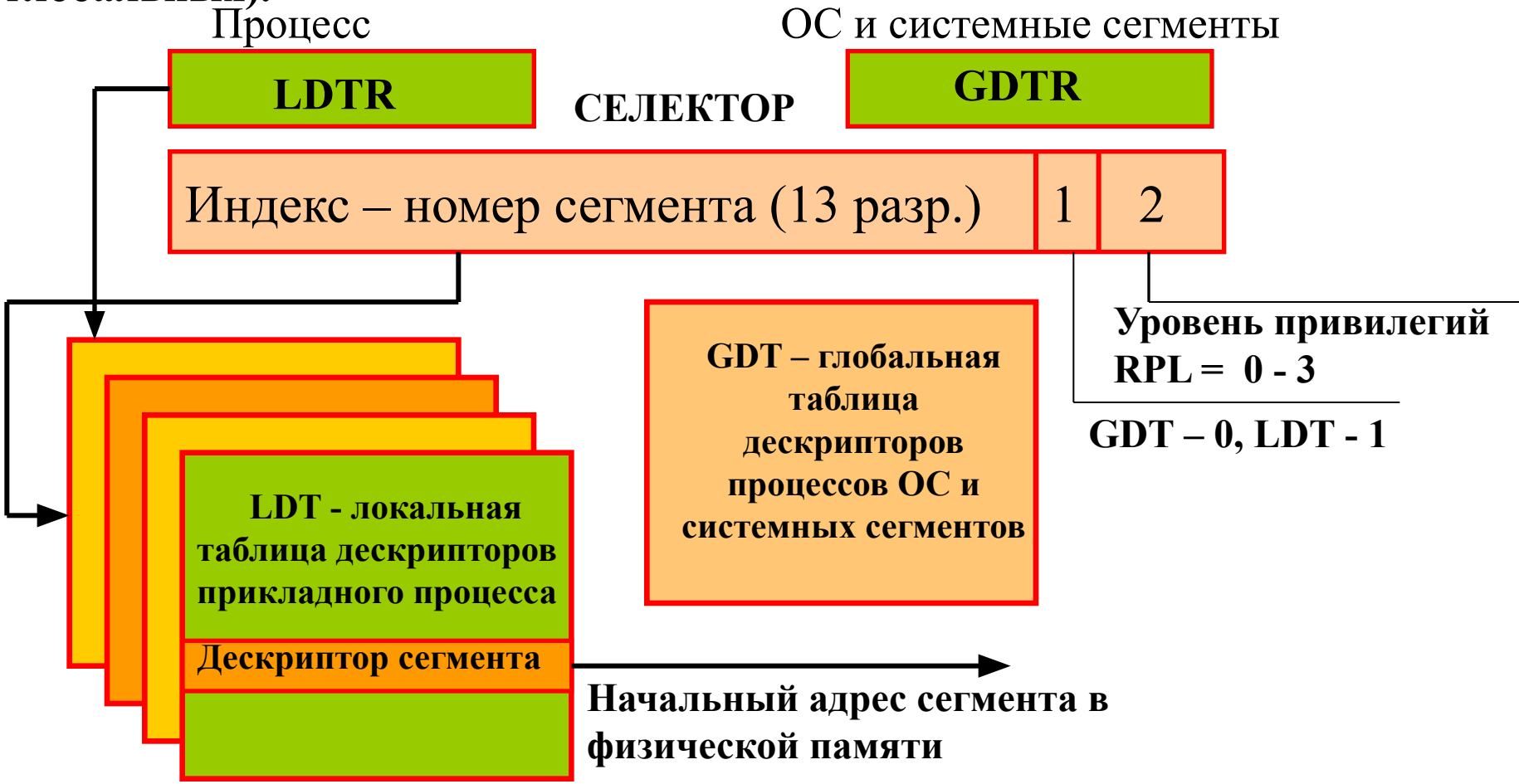
Способы создания разделяемого сегмента памяти

В этом случае настройка соответствующей записи для разделяемого сегмента выполняется только один раз, а все процессы пользуются такой настройкой и совместно используют часть оперативной памяти.



Способы создания разделяемого сегмента памяти

Виртуальная память Windows обеспечивает каждому процессу: 4 Гбайт виртуального адресного пространства (2 Гбайт – ОС, 2 Гбайт – пользовательская программа). 16 К независимых сегментов (8К локальных и 8К глобальных).



Организация памяти в Windows

Основа виртуальной памяти Windows 2000 представляется двумя таблицами: *локальной таблицей дескрипторов* LDT (Local Descriptor Table) и *глобальной таблицей дескрипторов* GDT (Global Descriptor Table).

У каждого процесса есть своя собственная таблица LDT, но *глобальная таблица дескрипторов одна*, ее совместно используют все процессы.

Таблица LDT описывает сегменты, локальные для каждой программы, включая ее код, данные, стек и т. д.; таблица GDT несет информацию о системных сегментах, включая саму операционную систему.

Организация памяти в Windows

В каждый момент времени в специальных регистрах GDTR и LDTR хранится информация о местоположении и размерах глобальной таблицы GDT и активной таблицы LDT.

Регистр LDTR указывает на расположение сегмента LDT в оперативной памяти косвенно - он содержит индекс дескриптора в таблице GTD, в котором содержится адрес таблицы LDT и ее размер.

Процесс обращается к физической памяти по виртуальному адресу, представляющему собой пару - селектор и смещение.

Селектор определяет номер сегмента, а смещение - положение искомого адреса относительно начала сегмента.

Организация памяти в Windows

***Селектор состоит из трех полей*.**

Индекс **задает пользовательский номер дескриптора в таблице GDT или LDT (всего 2^{13} = 8 К сегментов).**

Таким образом, виртуальное адресное пространство процесса состоит из 8К локальных и 8К глобальных сегментов, всего из 16К сегментов.

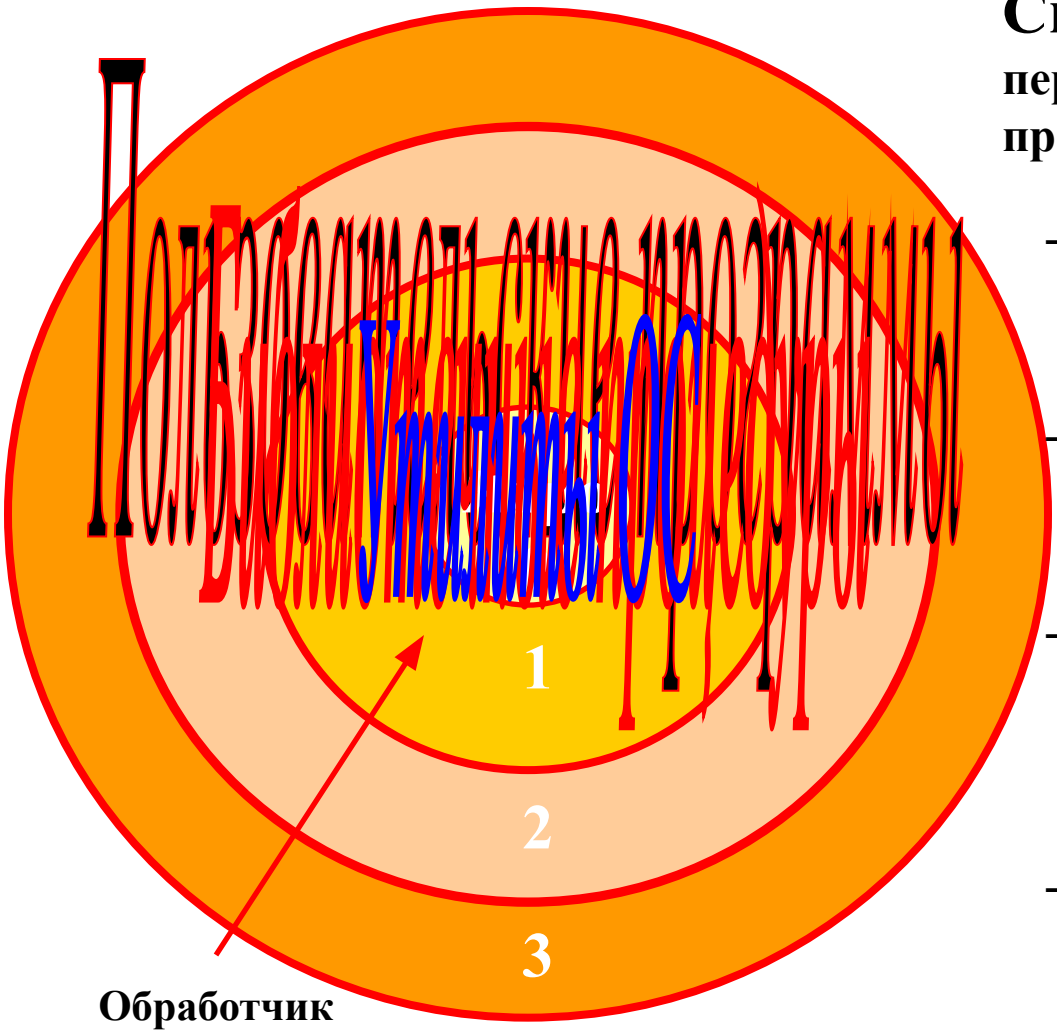
Каждый сегмент имеет максимальный размер 4 Гбайт при чисто сегментной организации виртуальной памяти (без включения страничного механизма), поэтому процесс может работать в виртуальном адресном пространстве в 64 Тбайт.

Организация памяти в Windows

Поле из двух битов селектора задает требуемый *уровень привилегий* и используется механизм защиты.

В системах на базе микропроцесса Pentium поддерживается 4 уровня защиты, где уровень 0 является наиболее привилегированным, а уровень 3 - наименее привилегированным.

Эти уровни образуют так называемые *кольца защиты*.



Система защиты использует переменные, характеризующие уровень привилегий:

-DPL (Descriptor Privilege Level) – задается полем DPL в дескрипторе сегмента;

-RPL (Requested Privilege Level) – запрашиваемый уровень привилегий, задается полем RPL селектора сегмента;

-CPL (Current Privilege Level) – текущий уровень привилегий выполняемого кода задается полем RPL селектора кодового сегмента (фиксируется в PSW);

-EPR (Effective Privilege Level) – эффективный уровень привилегий запроса.

Контроль доступа к сегменту данных осуществляется , если EPL <= DPL, где EPL = max { CPL, RPL }. Значение RPL – уровня запрашиваемых привилегий – определяется полем RPL селектора, указывающего на запрашиваемый сегмент.

Система защиты использует в Windows

Под *запросом* понимается любое обращение к памяти.

Уровни привилегий DPL и RPL назначаются операционной системой при создании новых процессов и во время их загрузки в память.

Уровень привилегий определяет не только возможности доступа к сегментам и дескрипторам, но и разрешенный набор инструкций.

В каждый момент времени работающая программа находится на определенном уровне, что отмечается 2-битовым полем в регистре слова состояние программы (PSW).

Уровень привилегий кодового сегмента DPL определяет текущий уровень привилегий CPL, фиксируемый в PSW.