Проектирование операционных систем
Лекция 6
Управление памятью

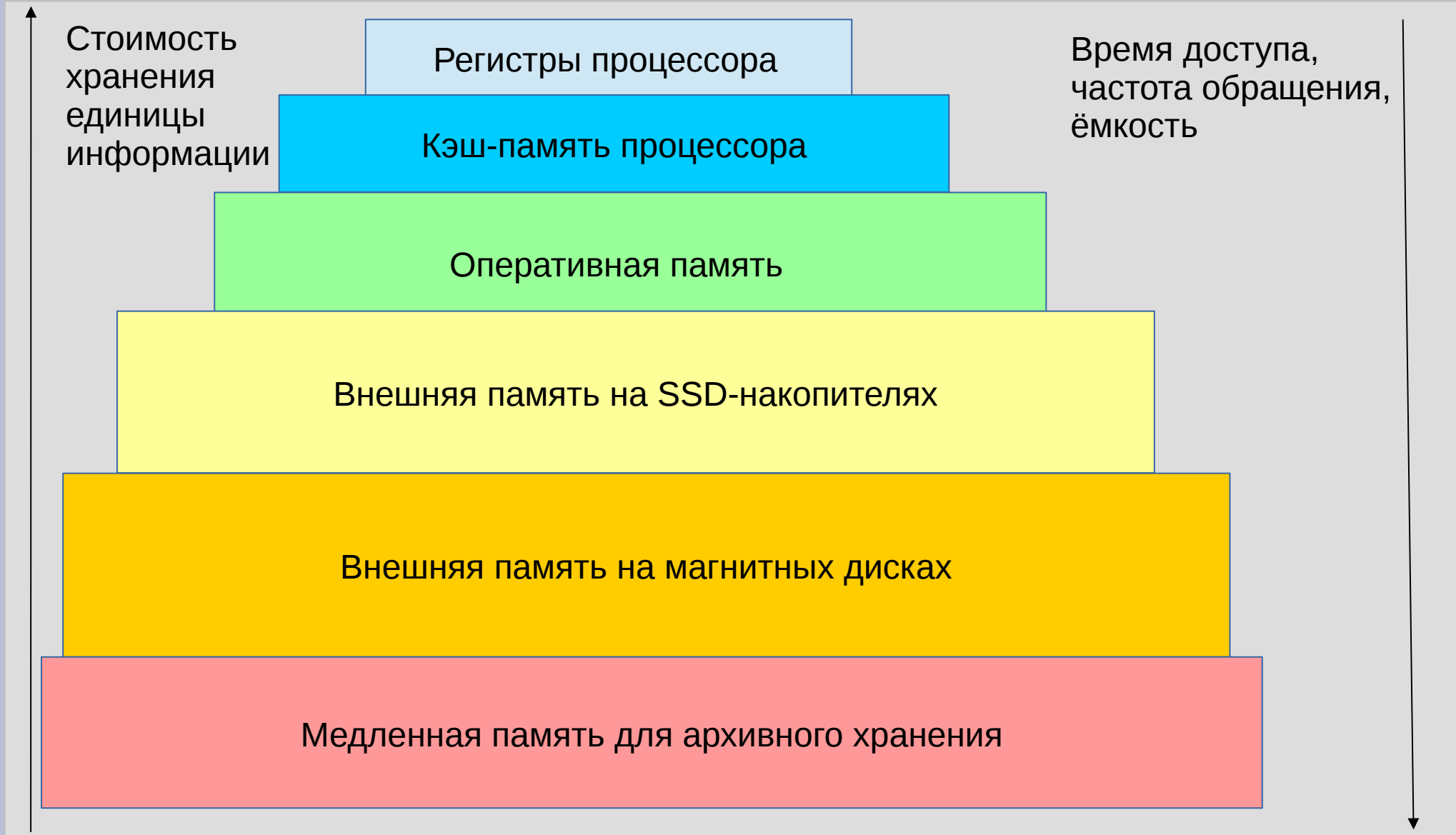
Боровцов Е.Г., 2016 г.

Необходимость эффективного управления памятью

Необходимость эффективного управления памятью вытекает из следующего:

- выполняемая программа (или ее фрагмент) должна находиться в памяти;
- память — один из самых дорогих ресурсов системы;
- памяти всегда мало — требования программ к памяти растут быстрее, чем объёмы памяти вычислительных систем;
- без эффективного управления памятью невозможна организация мультипрограммного режима работы операционной системы.

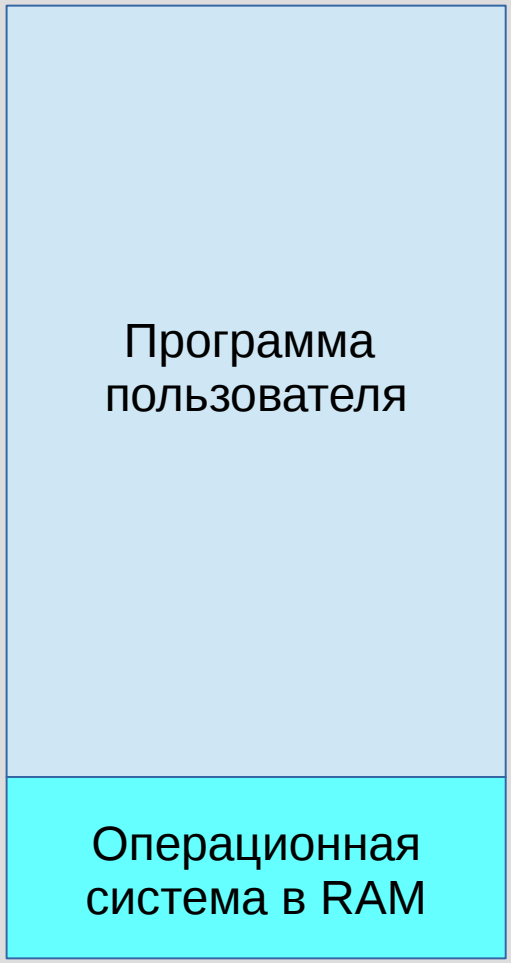
Иерархия памяти вычислительной системы



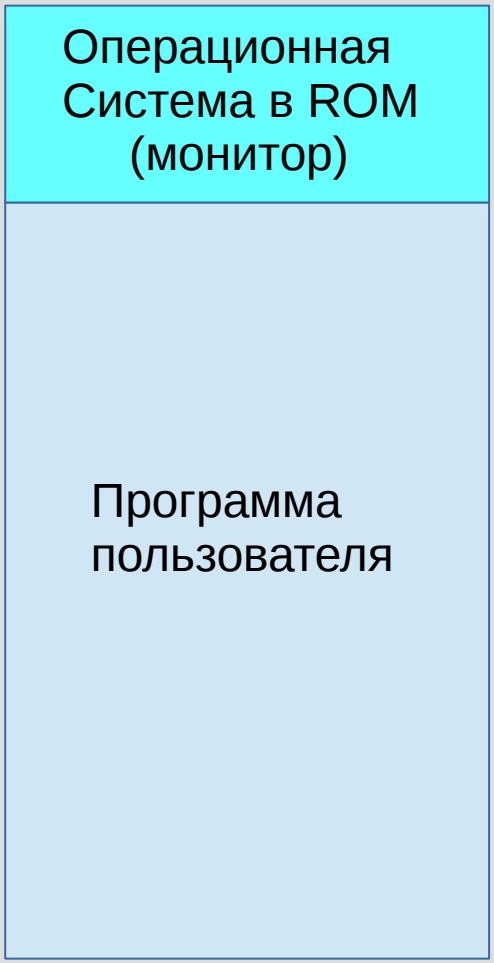
Отсутствие логических абстракций — работа с физической памятью

Ранние универсальные машины (до 1960 года), ранние мини-компьютеры (до 1970 года) и ранние персональные компьютеры (до 1980 года) не использовали абстракции памяти. Каждая программа просто видела всю физическую память. Программы использовали, в основном, абсолютную адресную схему и были неперемещаемыми. Использовалось фиксированное разбиение адресного пространства системы. Реализация мультипрограммного режима работы представлялась проблематичной вследствие неперемещаемости программ и отстуствия механизмов управления и защиты памяти

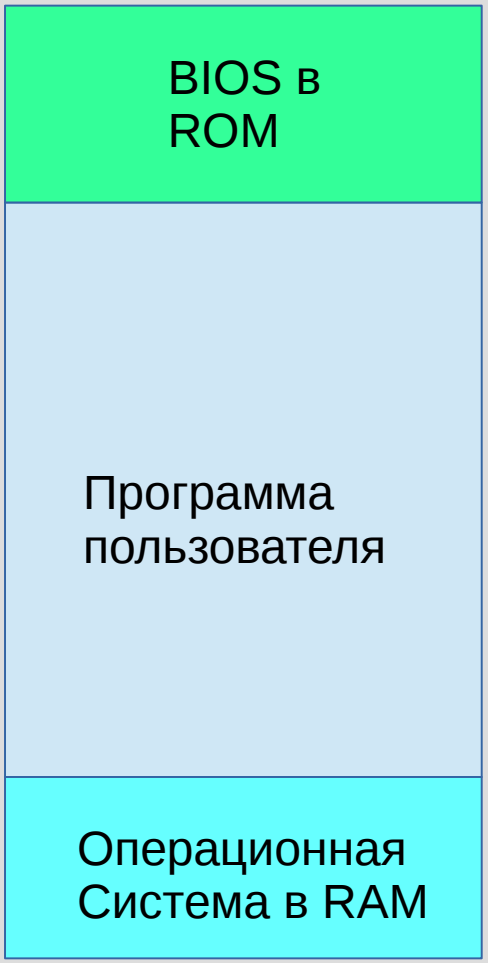
Фиксированное разбиение памяти



Ранние универсальные системы и мини-ЭВМ

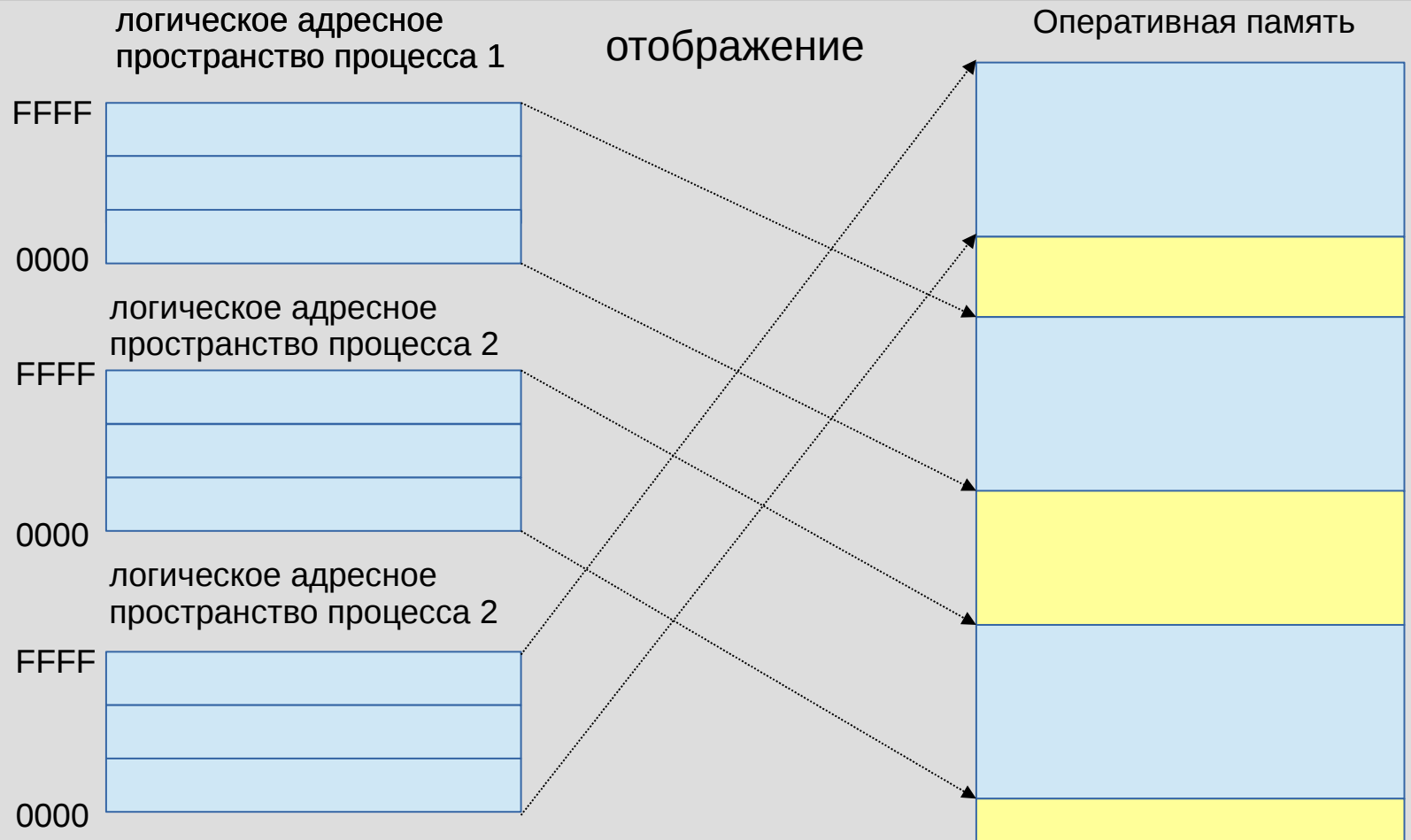


Первые ПК, КПК, Встраиваемые системы



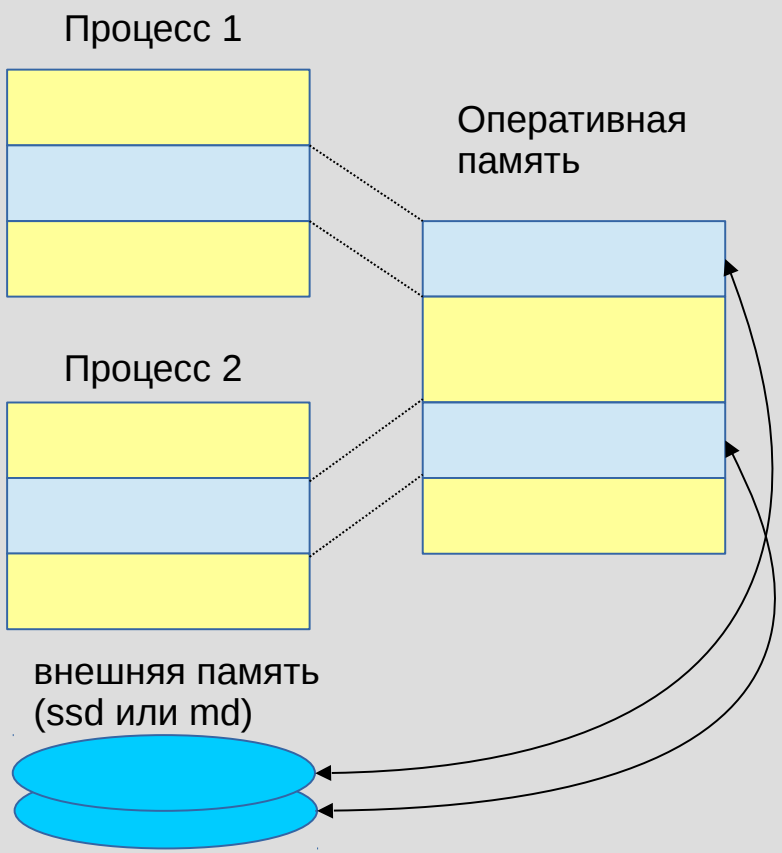
ПК под управлением DOS (CP/M,MS-DOS)

Логическое адресное пространство процесса и память системы

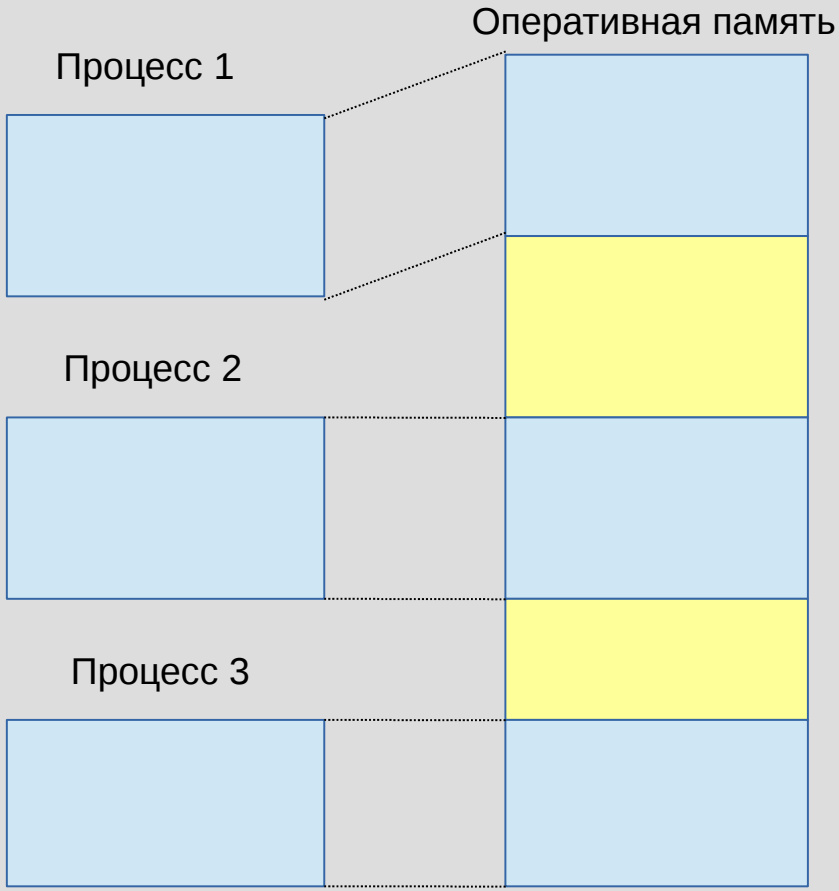


Каждый процесс работает в собственном логическом адресном пространстве, но все они отображаются на одно физическое адресное пространство

Соотношение логического адресного пространства процесса и физического простанства памяти



Оперативная память меньше логического пространства процессов. Используется либо механизм свопинга (swapping), либо виртуальная память



Оперативная память больше логического пространства процессов. Внешние ЗУ для управления памятью не используются

Способы отображения логического адресного пространства на физическую память

Отображение логического(виртуального) адресного пространства процесса на физическую память системы может осуществляться следующими способами:

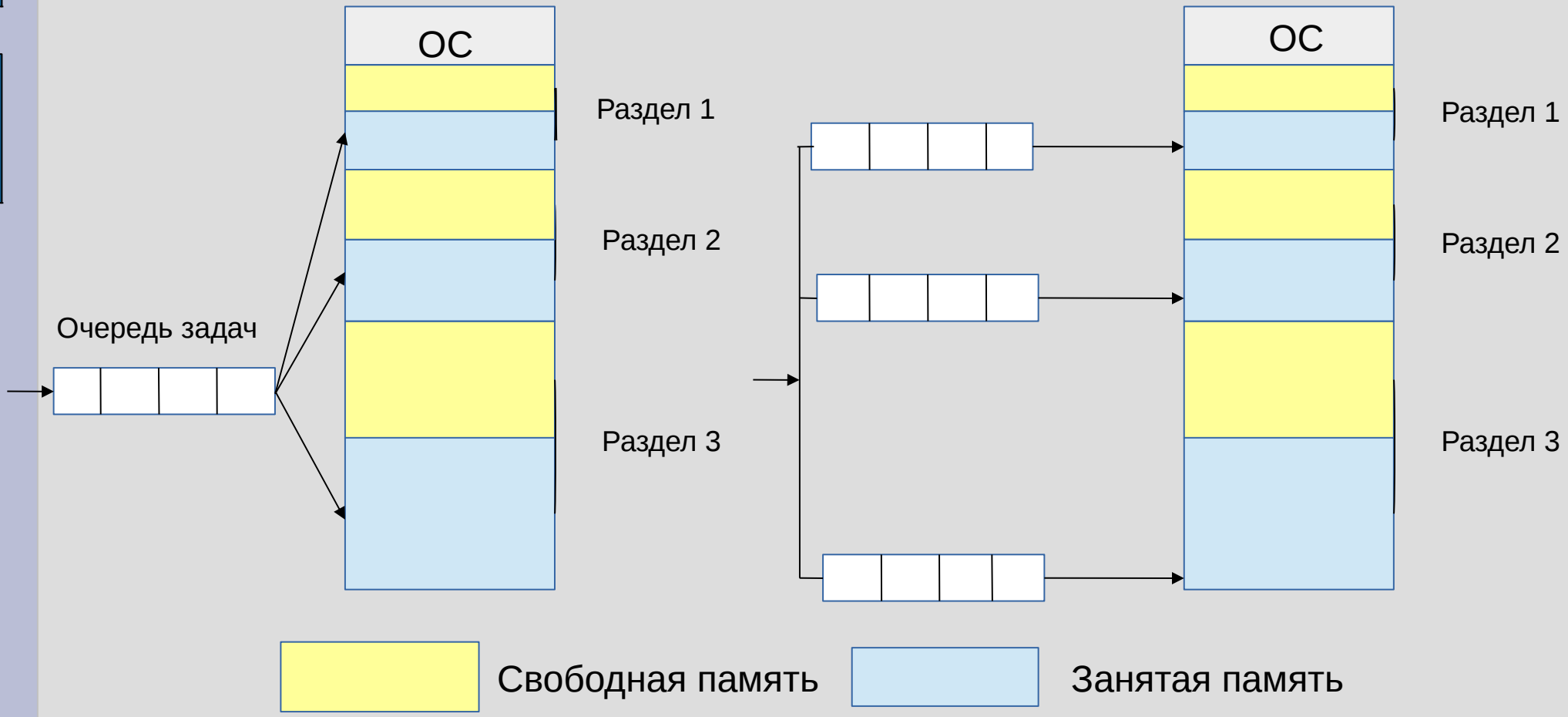
- статическое отображение - трансляция программы осуществляется на конкретные физические адреса с учётом ее последующего месторасположения в памяти вычислительной системы;
- однократное динамическое отображение — на этапе загрузки программы в память ***перемещающий загрузчик*** обеспечивает связывание логического пространства процесса с определённой областью памяти;
- Динамическая трансляция адресов в процессе выполнения — логический адрес каждой инструкции и операнда динамически транслируется в физический адрес памяти.

Функции подсистемы управления памятью

Чтобы обеспечить эффективный контроль использования памяти, ОС должна выполнять следующие функции:

- отображение адресного пространства процесса на конкретные области физической памяти;
- распределение памяти между конкурирующими процессами;
- контроль доступа к адресным пространствам процессов;
- выгрузка процессов (целиком или частично) во внешнюю память, когда в оперативной памяти недостаточно места;
- учет свободной и занятой памяти.

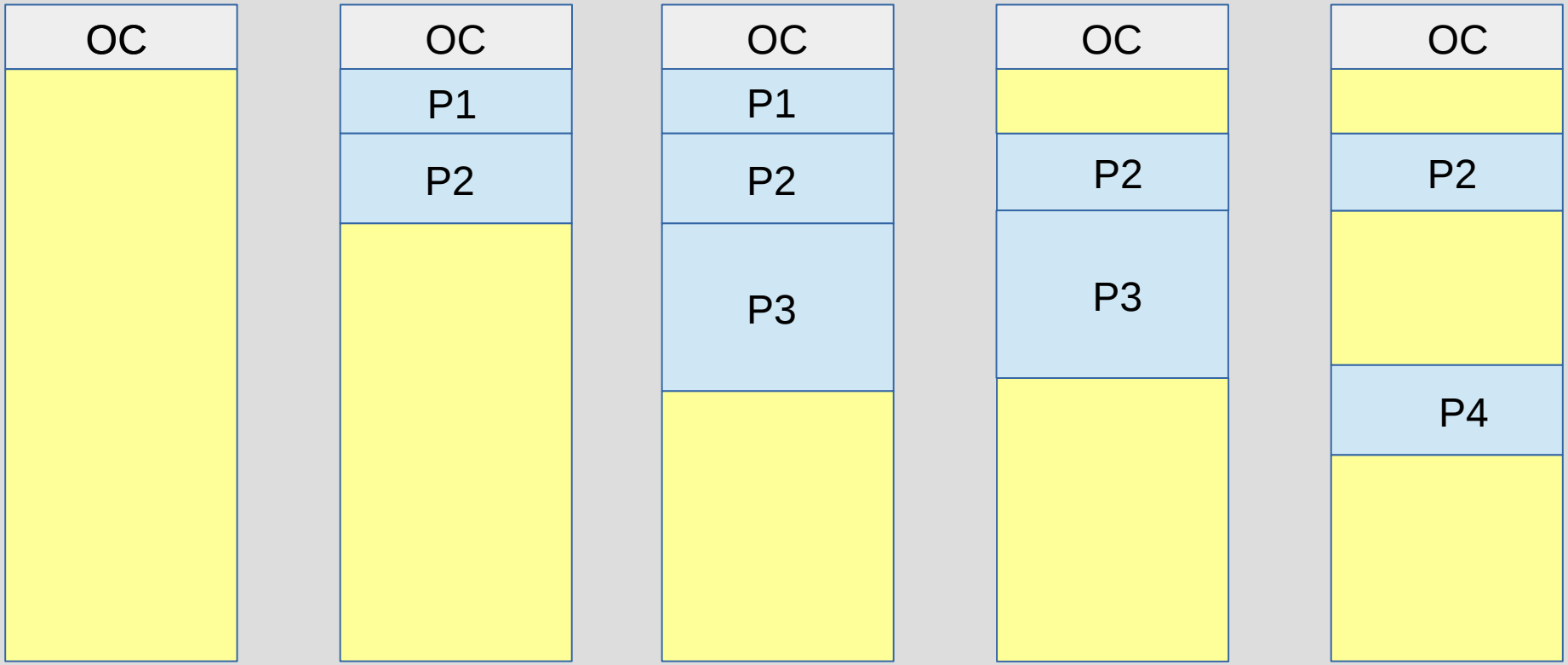
Распределение памяти фиксированными разделами



Достоинство-простота;

Недостатки: возможное неэффективное распределение памяти; фиксированное число задач ;

Распределение памяти разделами переменного размера



Свободная память

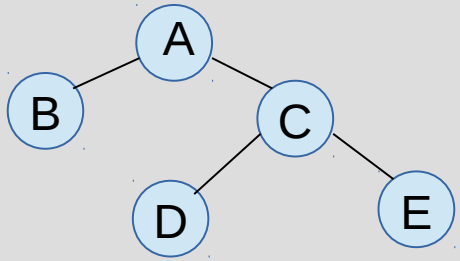
Занятая память

Основной недостаток — фрагментация памяти, для исключения которой применяют схемы с перемещаемыми разделами, фактически реализуя процедуру дефрагментации, требующую дополнительных ресурсов времени

Разрешение проблем нехватки оперативной памяти: оверлеи,свопинг, виртуальная память

Одно из первых решений проблемы нехватки оперативной памяти — использование структур с перекрытием - **оверлеев(overlay)**

```
Program A          Procedure C
  . . .              . . .
Call B             Call D
  . . .              . . .
Call C             Call E
```



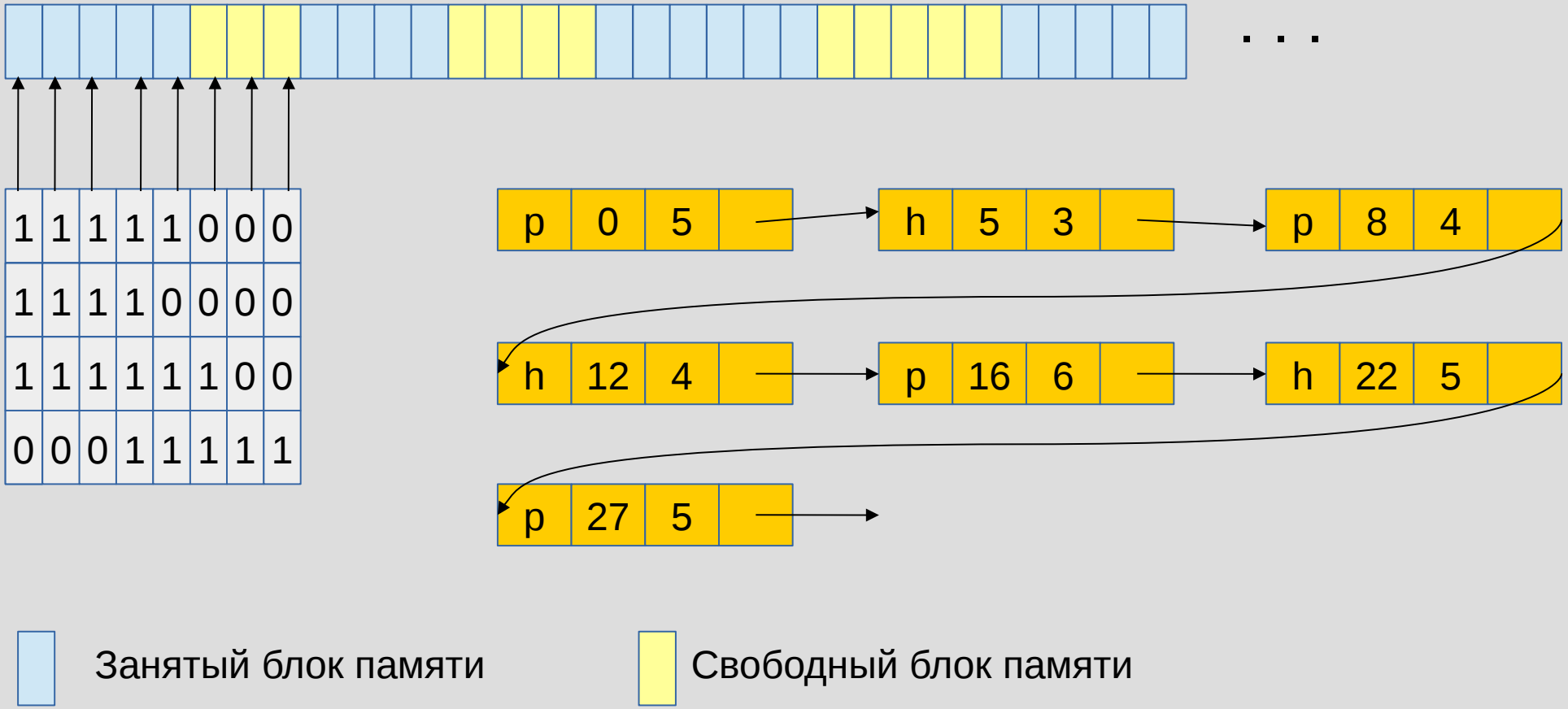
Планирование структуры программы (ее разбиение на сегменты) осуществляется программистом «вручную» на этапе её написания. В процессе функционирования программы необходимые сегменты подгружаются с диска

Следующий вариант —**свопинг** (swapping) при нехватке оперативной памяти реализуется выгрузка ожидающего процесса(***целиком***) на диск при загрузке другого процесса, получившего время процессора и готового к выполнению.

Виртуальная память — использует принцип обмена между оперативной памятью и внешним хранилищем не образами процессов, а их фрагментами — сегментами или страницами.

Управление свободной памятью

При реализации схем с динамическим распределением памяти возникает проблема управления блоками свободной памяти. Проблема в общем случае решается двумя способами - использованием битовых матриц и связных списков.



Виртуальная память и способы её организации

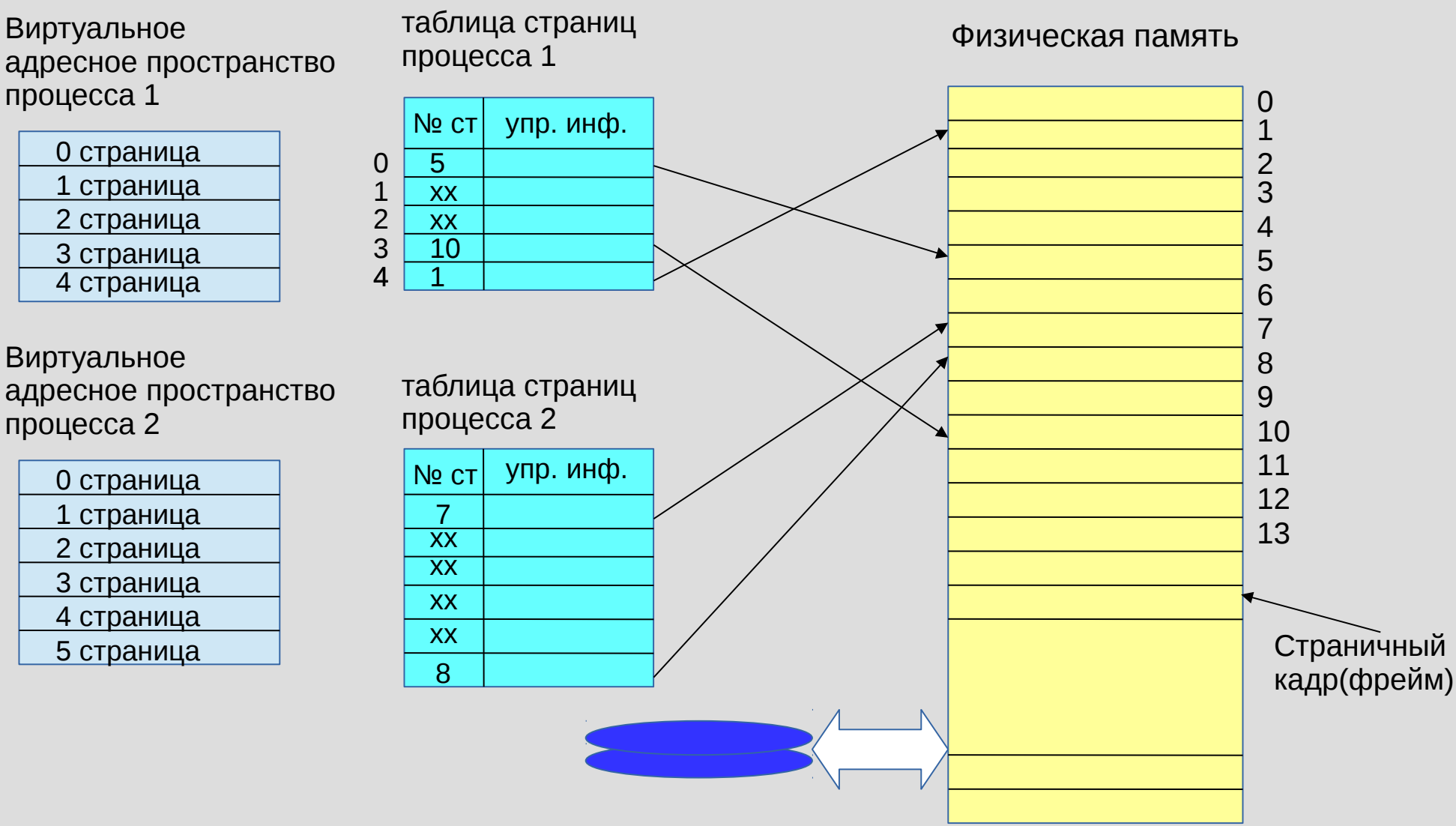
При работе в режиме ***виртуальной памяти*** процесс функционирует в виртуальном адресном пространстве. В этом случае процессор выставляет виртуальные адреса диспетчеру памяти (MMU), который осуществляет динамическую трансляцию виртуального адреса в физический и выставляет его на шину адреса.

Страничная виртуальная память — организует перемещение данных между оперативной памятью и внешним устройством страницами — фрагментами виртуального адресного пространства фиксированного и сравнительно небольшого размера.

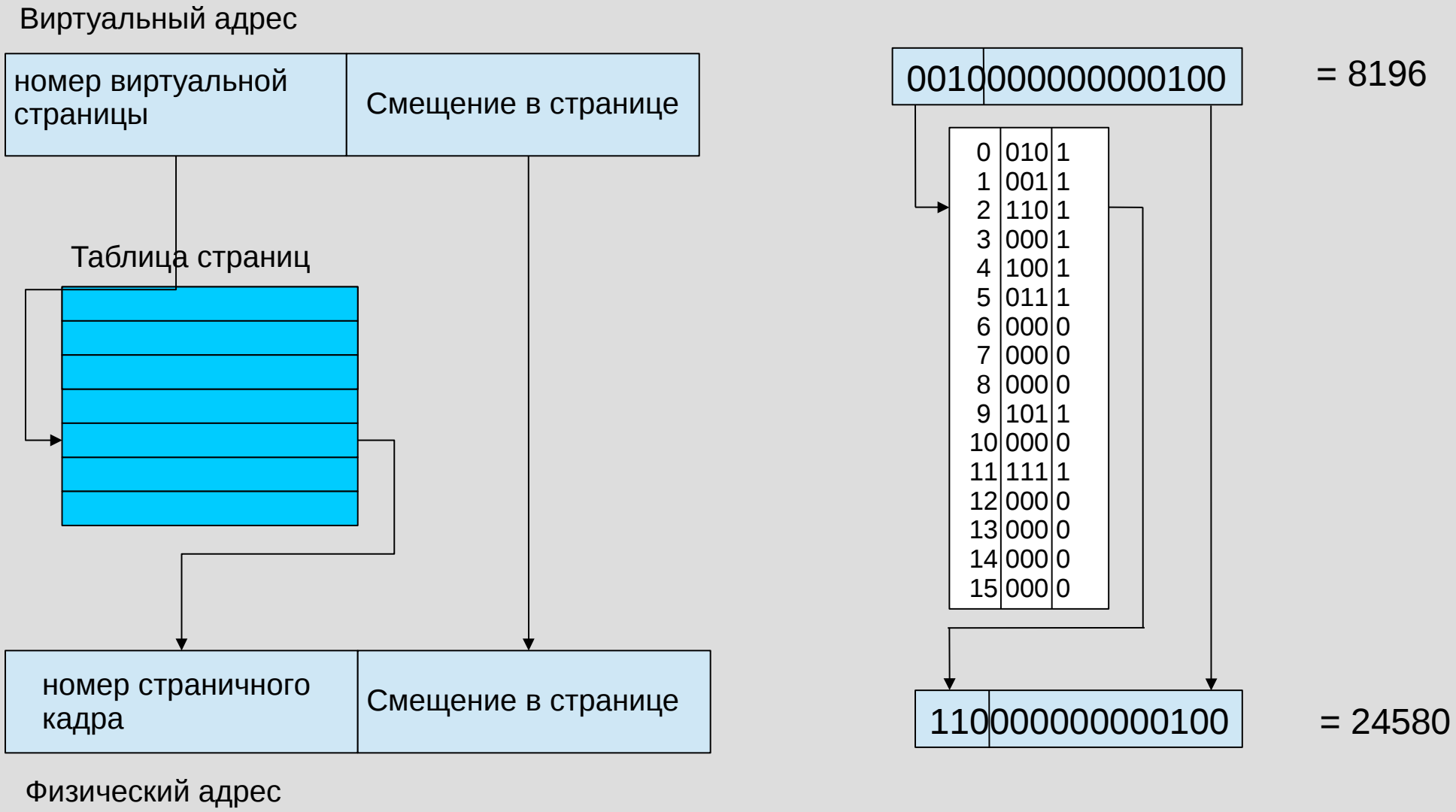
Сегментная виртуальная память — организует перемещение данных между оперативной памятью и внешним устройством сегментами — фрагментами виртуального адресного пространства произвольного размера, организуемых с учетом смыслового значения данных (сегмент кода, сегмент данных и т. д.)

Сегментно-страничная виртуальная память использует двухуровневое деление — виртуальное адресное пространство процессов делится на сегменты, которые, в свою очередь, делятся на страницы фиксированного размера. Единицей обмена данными между оперативной памятью и внешним устройством является страница. Данный подход управления памятью является комбинацией первых двух.

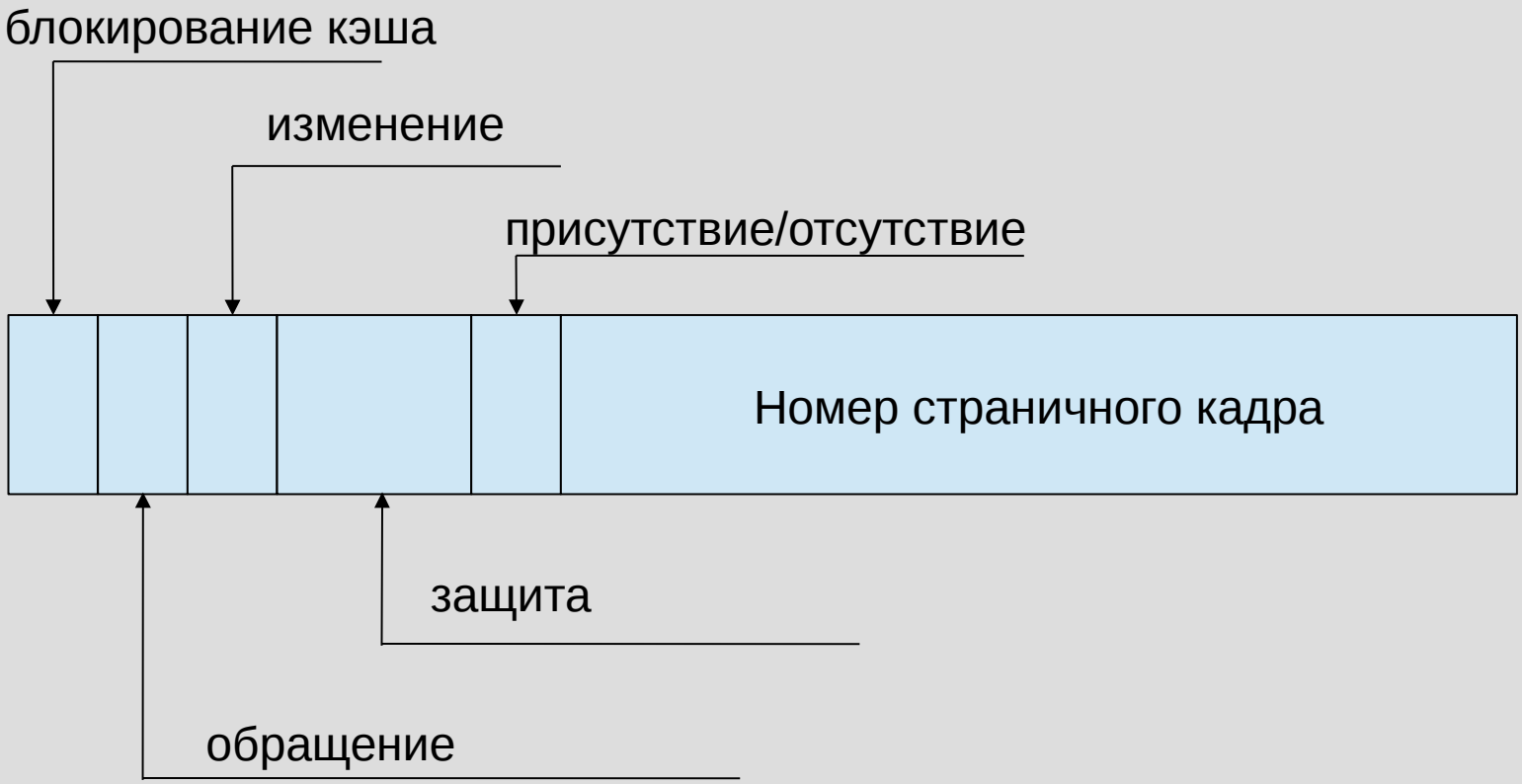
Страничная виртуальная память



Преобразование виртуального адреса в физический при страничной организации памяти



Структура записи в таблице страниц



Проблемы, возникающие при организации страничной виртуальной памяти

При организации страничной виртуальной памяти возникают следующие основные проблемы, влияющие на эффективность её использования:

1.Отображение виртуального адреса на физический должно быть быстрым

2.С ростом виртуального адресного пространства размеры таблицы страниц могут существенно увеличиваться

3.Каким образом реализуется алгоритм замещения страниц

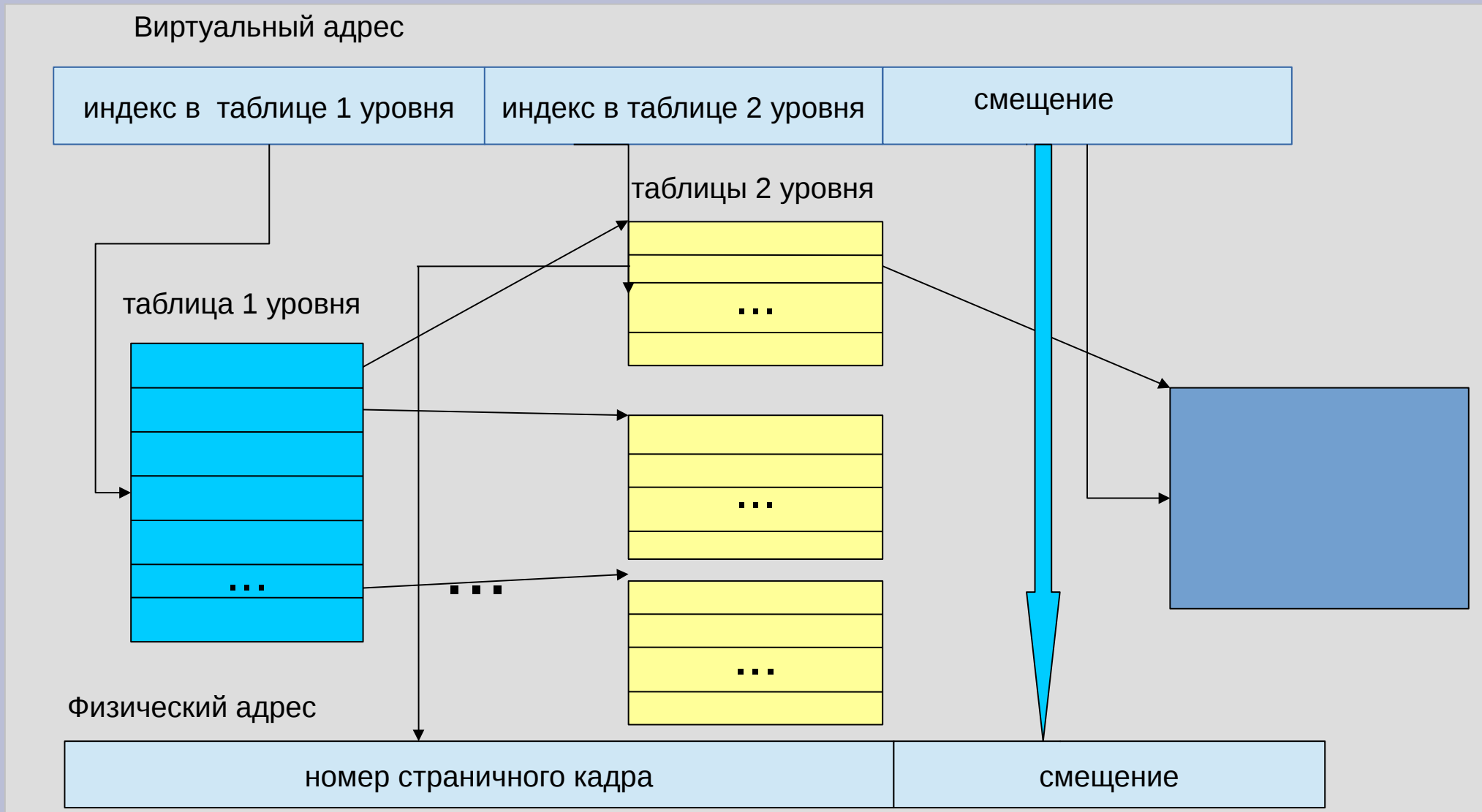
4.Каков оптимальный размер страницы

Ускорение отображения виртуального адреса на физический с использованием TLB

TLB — Translation Lookaside Buffer — буфер быстрого преобразования адресов — реализуется как специализированный кэш процессора на основе ассоциативной памяти.

Используется	Виртуальная страница	Изменена	защищена	Страничный кадр
1	140	1	rw	31
1	20	0	rx	38
1	130	1	rw	39
1	129	1	rw	62
1	19	0	rx	50

Многоуровневые таблицы страниц



Инвертированные таблицы страниц при организации страничной виртуальной памяти

Альтернатива постоянно растущим уровням иерархии страничной адресации называется инвертированными таблицами страниц. Впервые они использовались такими процессорами, как PowerPC, UltraSPARC и Itanium. В данной конструкции имеется одна запись для каждого страничного кадра в реальной памяти, а не одна запись на каждую страницу в виртуальном адресном пространстве. Например, при использовании 64-разрядных виртуальных адресов, страниц размером 4 Кбайт и оперативной памяти размером 4 Гбайт инвертированные таблицы требовали только 1 048 576 записей.

Хотя инвертированные таблицы страниц экономят значительное количество пространства, по крайней мере в том случае, когда виртуальное адресное пространство намного объемнее физической памяти, у них есть один серьезный недостаток: преобразование виртуальных адресов в физические становится намного сложнее, так как записи в инвертированной таблице не отсортированы по возрастанию номеров виртуальных страниц. Когда процесс n обращается к виртуальной странице p, аппаратура уже не может найти физическую страницу, используя p в качестве индекса внутри таблицы страниц. Вместо этого она должна провести поиск записи (n, p) по всей инвертированной таблице страниц. Одним из приемлемых способов осуществления этого поиска является ведение хэш-таблицы, созданной на основе виртуальных адресов. Записи в хэш-таблице содержат связку «виртуальная страница-страничный кадр».

Алгоритмы замещения страниц-1

Основным действием менеджера памяти является выделение страничного кадра для размещения в нём виртуальной страницы, находящейся во внешней памяти. В случае, когда размер виртуальной памяти для каждого процесса может существенно превосходить размер основной памяти, при выделении страницы основной памяти с большой вероятностью не удастся найти свободный страничный кадр. В этом случае ОС в соответствии с заложенными в нее критериями должна:

- найти некоторую занятую страницу основной памяти;
- переместить в случае надобности ее содержимое во внешнюю память;
- переписать в этот страничный кадр содержимое нужной виртуальной страницы из внешней памяти;
- должным образом модифицировать необходимый элемент соответствующей таблицы страниц;
- продолжить выполнение процесса, которому эта виртуальная страница понадобилась.

При замещении приходится дважды передавать страницу между основной и вторичной памятью. Процесс замещения может быть оптимизирован за счет использования бита модификации (один из атрибутов страницы в таблице страниц). Бит модификации устанавливается компьютером, если хотя бы один байт был записан на страницу. При выборе кандидата на замещение проверяется бит модификации. Если бит не установлен, нет необходимости переписывать данную страницу на диск, ее копия на диске уже имеется. Подобный метод также применяется к read-only-страницам, они никогда не модифицируются. Эта схема уменьшает время обработки page fault. Интенсивность страничного обмена может быть так же снижена в результате *упреждающей* загрузки

Существует большое количество разнообразных алгоритмов замещения страниц. Все они делятся на локальные и глобальные.

Алгоритмы замещения страниц-2

Алгоритм замещения FIFO «первым вошел-первым вышел» - ОС ведёт список страниц, ранее всех загруженные страницы — в «голове» списка, вновь загружаемые — в «хвосте». Низкозатратный алгоритм, в чистом виде применяется редко.

Оптимальный алгоритм - каждая страница может быть помечена количеством команд, которые должны быть выполнены до первого обращения к странице; должна быть удалена страница, имеющая пометку с наибольшим значением. Алгоритм легко описать но практически невозможно реализовать.

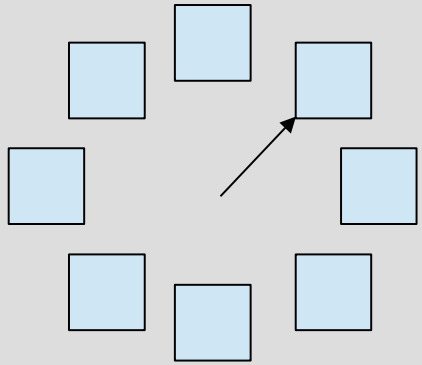
Алгоритм замещения наименее востребованной страницы (Least Recently Used (LRU)) Для его полной реализации необходимо вести связанный список всех страниц, находящихся в памяти. В начале этого списка должна быть только что востребованная страница, а в конце — наименее востребованная. Сложность в том, что этот список должен обновляться при каждом обращении к памяти.Один из вариантов реализации алгоритма предполагает, что с каждой страницей связан 64-х разрядный счетчик использования. Вытесняется страница с наименьшим значением счетчика.

Алгоритм нечастого востребования (Not Frequently Used (NFU)). Для его реализации требуется программный счетчик с начальным нулевым значением, связанный с каждой страницей. При каждом прерывании от таймера операционная система сканирует все находящиеся в памяти страницы. Для каждой страницы к счетчику добавляется значение бита R, равное 0 или 1. Счетчики позволяют приблизительно отследить частоту обращений к каждой странице. При возникновении ошибки отсутствия страницы для замещения выбирается та страница, чей счетчик имеет наименьшее значение.Основной недостаток — отсутствие механизма «забывания». Существует модификация **алгоритм «старения»**— перед добавлением бита R осуществляется сдвиг вправо на 1 разряд и бит R добавляется не справа, а слева.

Алгоритмы замещения страниц-3

Алгоритм «часы»

При возникновении ошибки отсутствия страницы проверяется та страница, на которую указывает стрелка. Если ее бит R имеет значение 0, страница вытесняется, на ее место в «циферблате» вставляется новая страница и стрелка передвигается вперед на одну позицию. Если значение бита R равно 1, то он сбрасывается и стрелка перемещается на следующую страницу. Этот процесс повторяется до тех пор, пока не будет найдена страница с R = 0.



Алгоритм «рабочий набор» базируется на оптимальном количестве страничных кадров и некотором наборе страниц, с которыми процесс работает в течение некоторого периода времени. На практике довольно громоздок для реализации. Существует усовершенствованный алгоритм, базирующийся на концепции рабочего набора и являющийся модификацией алгоритма «часы» - **WSClock.** Благодаря простоте реализации и хорошей производительности он довольно широко используется на практике.

Наиболее приемлемыми алгоритмами являются алгоритм старения и алгоритм WSClock. Они основаны на LRU и рабочем наборе соответственно. Оба обеспечивают неплохую производительность страничной организации памяти и могут быть эффективно реализованы. Существует также ряд других хороших алгоритмов, но эти два, наверное, имеют наибольшее практическое значение.

Алгоритмы замещения страниц-4 сравнительная таблица алгоритмов замещения(Таненбаум)

Алгоритм	Особенности
Оптимальный	Не может быть реализован, но полезен в качестве оценочного критерия
NRU (Not Recently Used) — алгоритм исключения недавно использовавшейся страницы	Является довольно грубым приближением к алгоритму LRU
FIFO (First In, First Out) — алгоритм «первый пришел, первый ушел»	Может выгрузить важные страницы
Алгоритм «второй шанс»	Является существенным усовершенствованием алгоритма FIFO
Алгоритм «часы»	Вполне реализуемый алгоритм
LRU (Least Recently Used) — алгоритм замещения наименее востребованной страницы	Очень хороший, но труднореализуемый во всех тонкостях алгоритм
NFU (Not Frequently Used) — алгоритм нечастого востребования	Является довольно грубым приближением к алгоритму LRU
Алгоритм старения	Вполне эффективный алгоритм, являющийся

Размер страницы
(Таненбаум,Бос)

Пусть средний размер процесса будет составлять s байт, а размер страницы — p байт. Кроме этого предположим, что на каждую страничную запись требуется e байт. Тогда приблизительное количество страниц, необходимое каждому процессу, будет равно s/p, что займет se/p байт пространства таблицы страниц. Из-за внутренней фрагментации неиспользуемое пространство памяти в последней странице процесса будет равно p/2. Таким образом, общие издержки на таблицу страниц и внутреннюю фрагментацию будут получены за счет суммирования этих двух величин:

$$\text{Издержки} = se/p + p/2.$$

Первое слагаемое (размер таблицы страниц) будет иметь большее значение при небольшом размере страницы. Второе слагаемое (внутренняя фрагментация) будет иметь большее значение при большом размере страницы. Оптимальный вариант находится где-то посередине. Если взять первую производную по переменной p и приравнять ее к нулю, то мы получим уравнение:

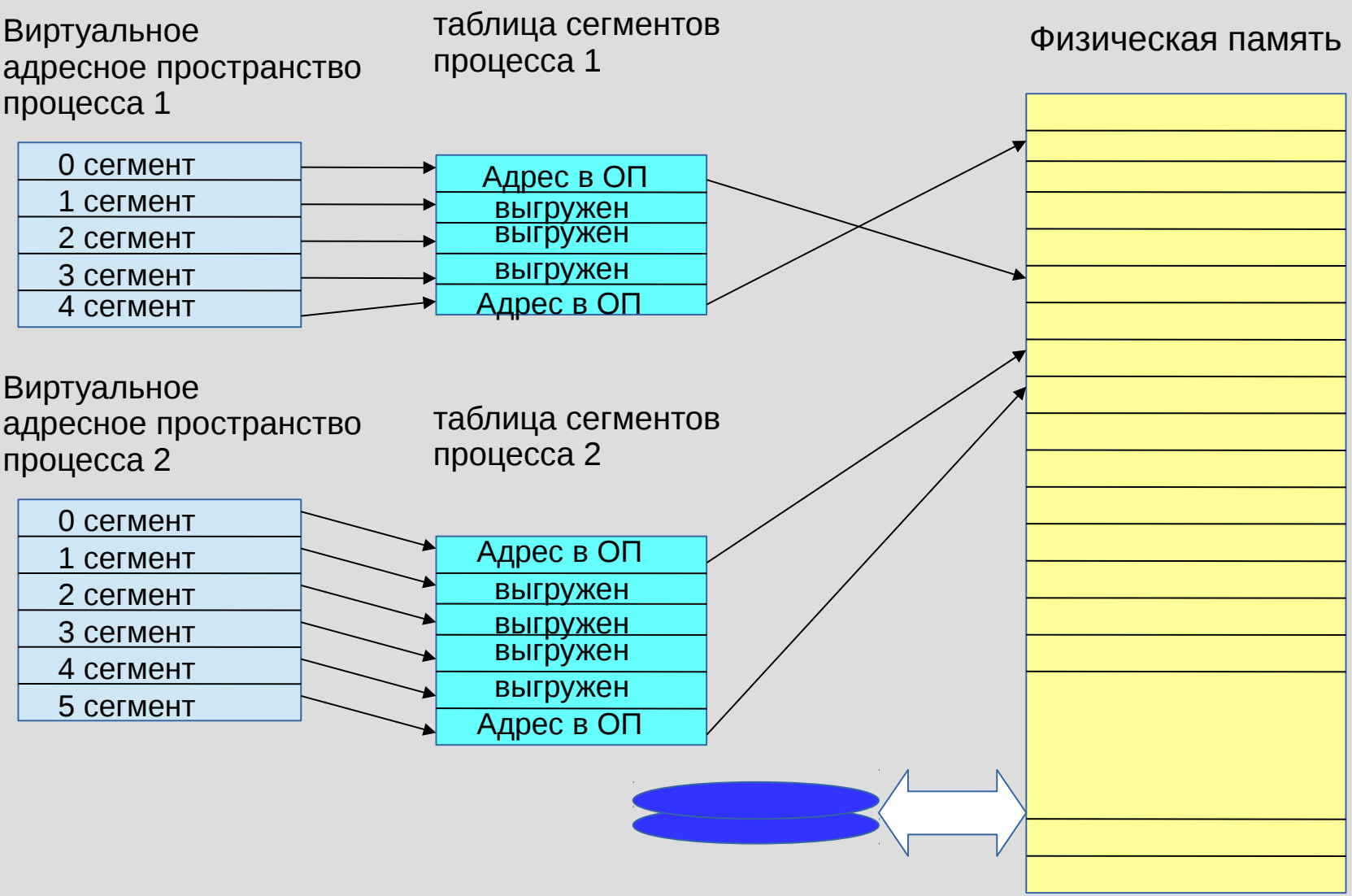
$$-se/p^2 + 1/2 = 0.$$

Из этого уравнения можно вывести формулу, дающую оптимальный размер страницы (с учетом только потерь памяти на фрагментацию и на размер таблицы страниц). Результат будет следующим:

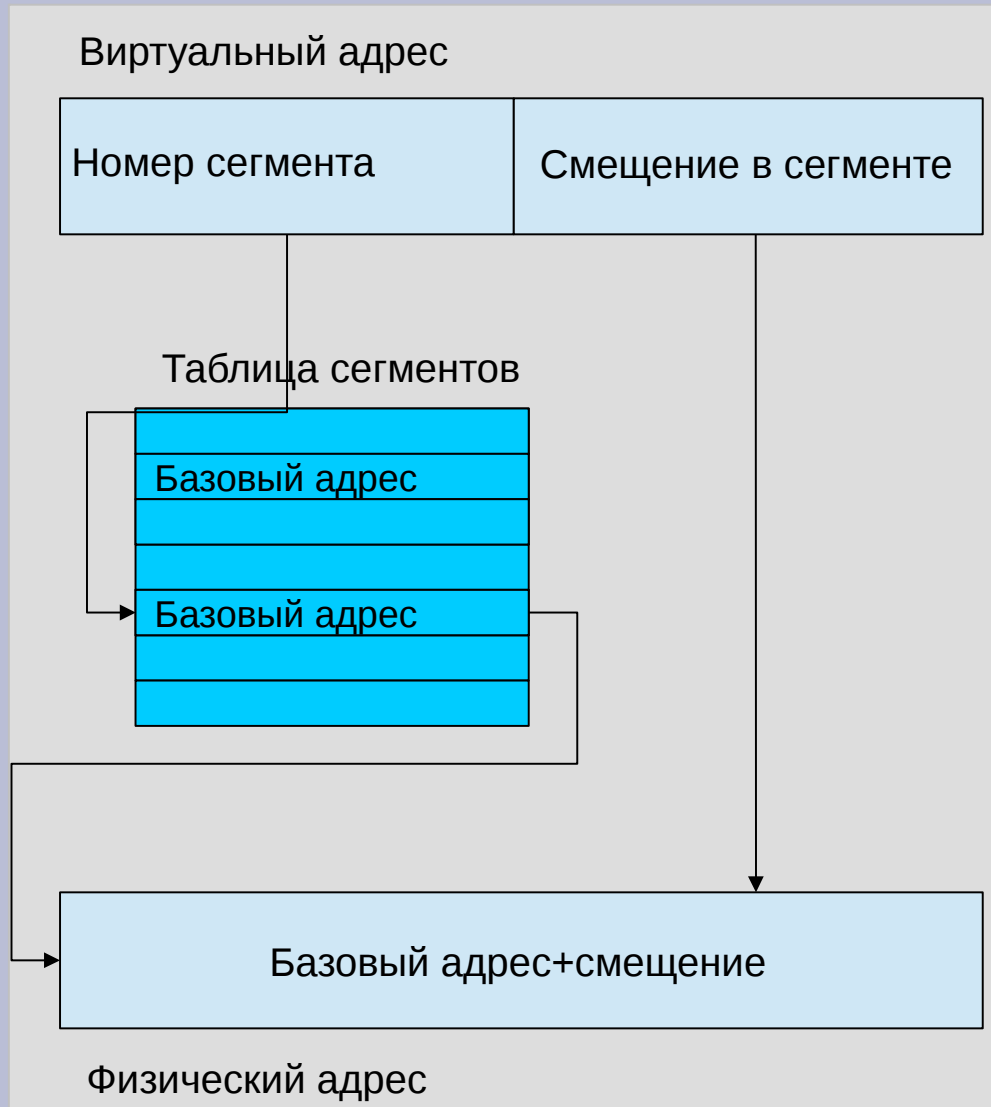
$$P = \sqrt{2se}$$

Для s = 1 Мбайт и e = 8 байт на каждую запись в таблице страниц оптимальный размер страницы будет 4 Кбайт. Имеющиеся в продаже компьютеры используют размер страницы от 512 байт до 64 Кбайт. Раньше чаще всего использовался размер 1 Кбайт, но сейчас чаще всего встречается размер страницы 4 Кбайт.

Сегментное распределение памяти



Преобразование виртуального адреса в физический при сегментной организации памяти



Управление памятью: литература

1.Таненбаум Э., Бос Х**.** Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил. — (Серия «Классика computer science»).
2. Основы операционных систем.Курс лекций. Учебное пособие / В.Е.Карпов, К.А.Коньков / Под редакцией В.П.Иванникова. - М.:ИНТУИТ.РУ «Интернет-Университет Информационных Технологий» - 2005, 536 с.
3. В.Г. Олифер, Н.А. Олифер. Сетевые операционные системы.- СПб.:Питер.-2002. - 544 с.