Операционные системы

Управление памятью

Основные положения

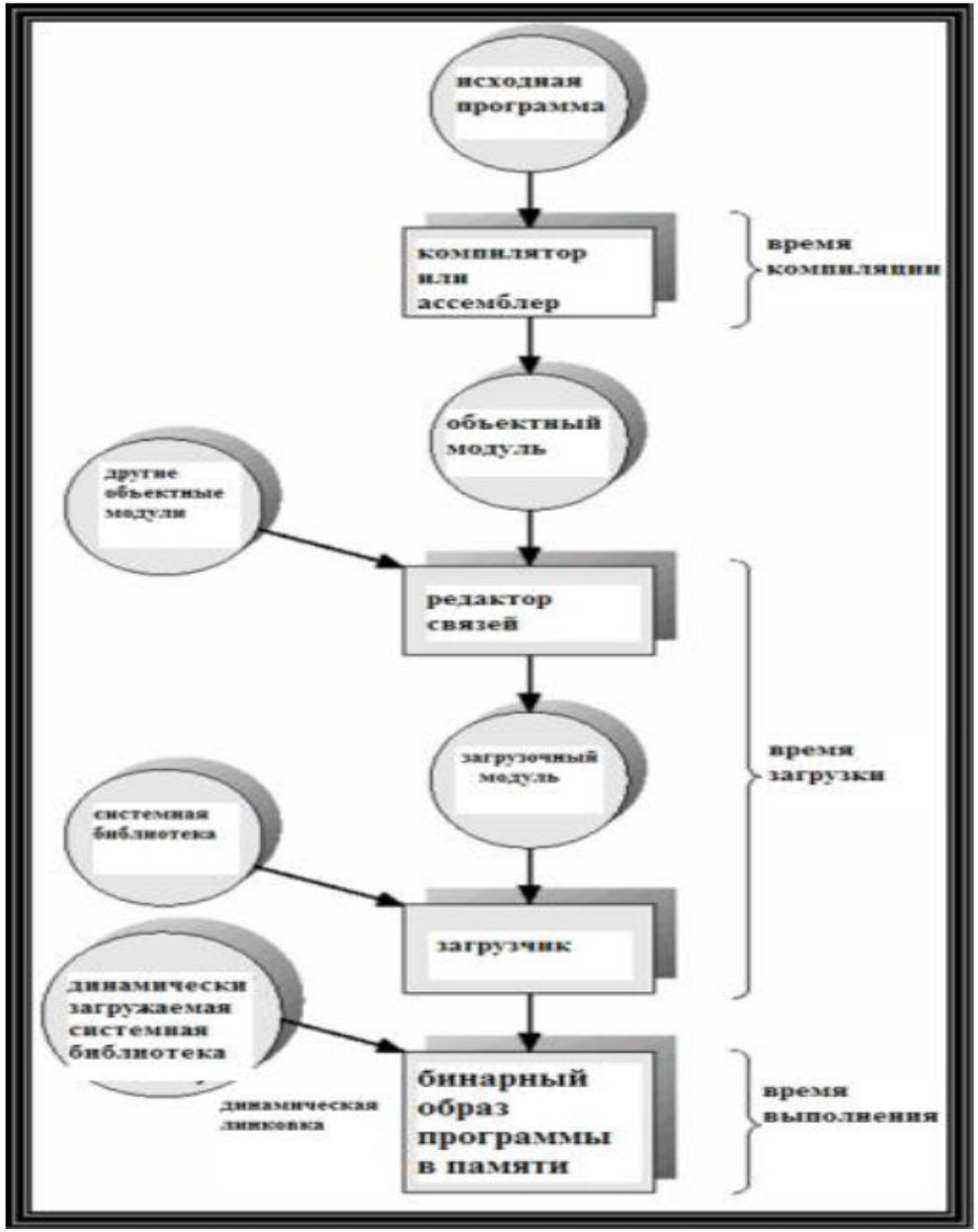
- **Программа должна быть размещена в памяти и оформлена в виде процесса для своего выполнения.**
- ***Входная очередь* – совокупность процессов на диске, ожидающих размещения в памяти для выполнения своих программ.**
- **Пользовательские программы проходят несколько стадий до своего выполнения.**

Связывание команд и данных с адресами памяти

Может выполняться на разных этапах

- ***Во время компиляции*: Если адрес в памяти априорно известен, может быть сгенерирован код с абсолютными адресами; при любых изменениях размещения в памяти должна быть выполнена перекомпиляция.**
- ***Во время загрузки*: Должен быть сгенерирован перемещаемый (relocatable) код, если адреса в памяти не известны во время компиляции.**
- ***Во время исполнения*: Связывание откладывается до периода выполнения, если процесс может быть перемещен во время выполнения из одного сегмента памяти в другой. Требуется аппаратная поддержка отображения адресов (например, регистры базы и границы).**

Многоэтапная обработка пользовательской программы



Исходный код программы (в форме текстового файла) на языке высокого уровня или на ассемблере преобразуется компилятором или ассемблером в **объектный модуль**, содержащий бинарные выполняемые машинные команды и **таблицу символов**, определенных и использованных в данном модуле кода. Рассмотренная фаза называется **временем компиляции**.

Объектный модуль не может непосредственно исполняться, так как он содержит неразрешенные ссылки на внешние модули и их компоненты.

Следующая фаза обработки программы – **редактирование связей. Редактор связей (linker)** – системная программа, которая получает на вход один или несколько объектных модулей, а на выходе выдает **загрузочный модуль** – двоичный код, образованный кодом нескольких объектных модулей, в котором разрешены все межмодульные ссылки.

Загрузочный модуль может быть загружен в память для исполнения с помощью системной программы – **загрузчика (loader),** который получает на вход загрузочный модуль и файлы с бинарными кодами **системных библиотек,** которые использует программа. Загрузчик, объединяя код программы с кодами системных библиотек, создает **бинарный образ программы в памяти**.

Логическое и физическое адресное пространство

- **Концепция логического адресного пространства, которое связано с соответствующим физическим адресным пространством, является главной для управления памятью.**
  - ***Логический адрес*** **– генерируется процессором; именуется также *виртуальным* адресом.**
  - ***Физический адрес*** **– адрес, который видит устройство памяти.**

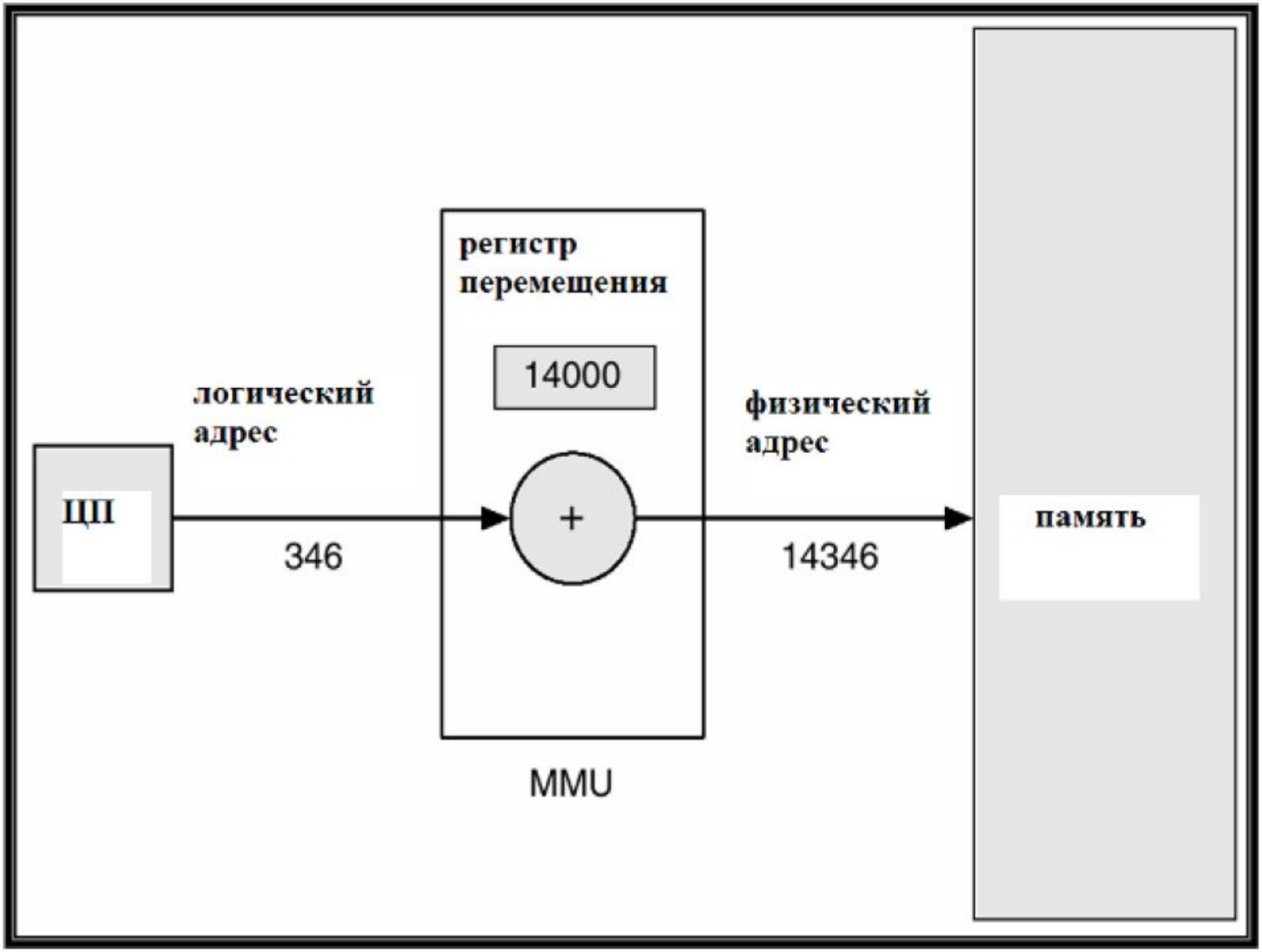
- **Логические адреса совпадают с физическими при связывании адресов во время компиляции и во время загрузки; логические адреса отличаются от физических при использовании схемы связывания адресов во время выполнения**

.

Устройство управления памятью (memory management unit – MMU)

- **Аппаратура, преобразующая виртуальный адрес в физический.**
- **В схеме MMU значение регистра перемещения прибавляется к любому адресу, сгенерированному пользовательским процессом, в момент, когда он размещается в памяти.**
- **Программа пользователя работает только с логическими адресами и не видит физических адресов.**

Динамическое перемещение с использованием регистра перемещения



Динамическая загрузка

Под **динамической загрузкой** понимается загрузка подпрограммы в память при первом обращении к ней из пользовательской программы.

Подпрограмма не загружается до момента ее вызова

Наиболее оптимальное использование памяти: ненужная подпрограмма не загружается.

Полезно, когда требуется сравнительно редкая обработка больших фрагментов кода.

Никакой специальной поддержки от ОС не требуется на этапе разработки программы.

Динамическая линковка

- **Линковка откладывается до времени исполнения.**
- **Небольшой фрагмент кода – заглушка (*stub*) используется для размещения соответствующей постоянно находящейся в памяти библиотечной подпрограммы.**
- **Заглушка заменяет себя на адрес подпрограммы и исполняет ее.**
- **ОС необходимо проверять, размещена ли подпрограмма в адресном пространстве процессов.**
- **Динамическая линковка наиболее целесообразна для библиотек (.dll, .so)**

Оверлейная структура (overlay)

В ранних ОС, в особенности – для персональных компьютеров, для пользовательского процесса были вынужденно введены очень жесткие ограничения по памяти. При дефиците основной памяти, если программа оказывается велика и полностью не помещается в память максимально разрешенного объема, необходимо предпринимать меры, чтобы разбить ее на непересекающиеся группы модулей, такие. что в каждой группе модули логически взаимосвязаны и должны присутствовать в памяти одновременно, модули же разных групп не обязательно должны вместе загружаться в память.

В памяти хранятся только те программы и данные, которые требуются в каждый данный момент времени.

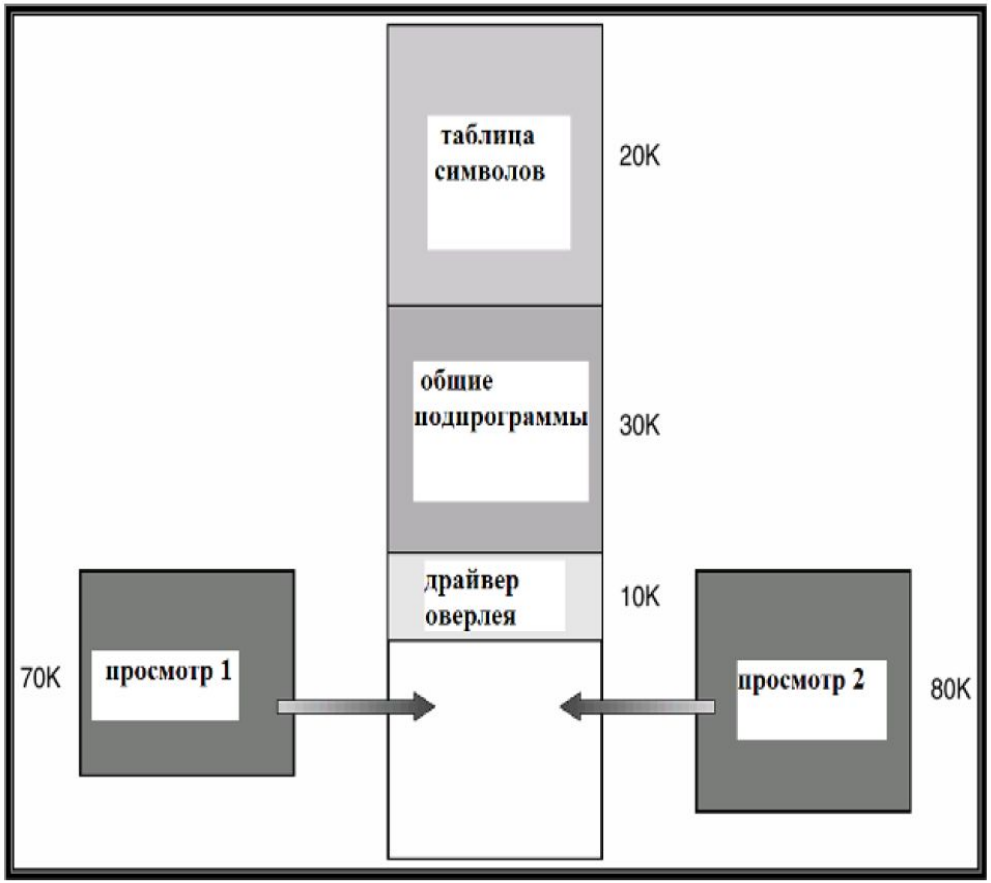
Во время исполнения такой программы должен использоваться специальный системный механизм, называемый **оверлейная структура (overlay,** дословно – **наложение**), обеспечивающий поочередную загрузку в одну и ту же область памяти то одной, то другой исполняемой группы модулей.

Необходимо, если памяти процессу требуется больше, чем область памяти, которая ему выделена.

Простая программа, которая выполняет эти действия, называется **драйвер оверлея (overlay driver).**

Реализуется пользователем (и поддерживается средой разработки программ); никакой поддержки от ОС не требуется; проектирование программы с оверлейной структурой сложно.

Оверлейная структура для двухпросмотрового ассемблера



Типичный для ранних компьютеров и ОС пример программы с оверлейной структурой – двухпросмотровый ассемблер. На первом просмотре он преобразует исходный ассемблерный код в промежуточное представление, которое программа второго просмотра ассемблера получает на входе. Полностью весь ассемблер (оба просмотра) в память не помещался, и пришлось применить оверлейную структуру

Операционные системы

Страничная организация памяти

Откачка и подкачка (swapping)

- **Процесс может быть временно откачан в файл откачки (backing store) и затем возвращен обратно в память для продолжения его выполнения.**

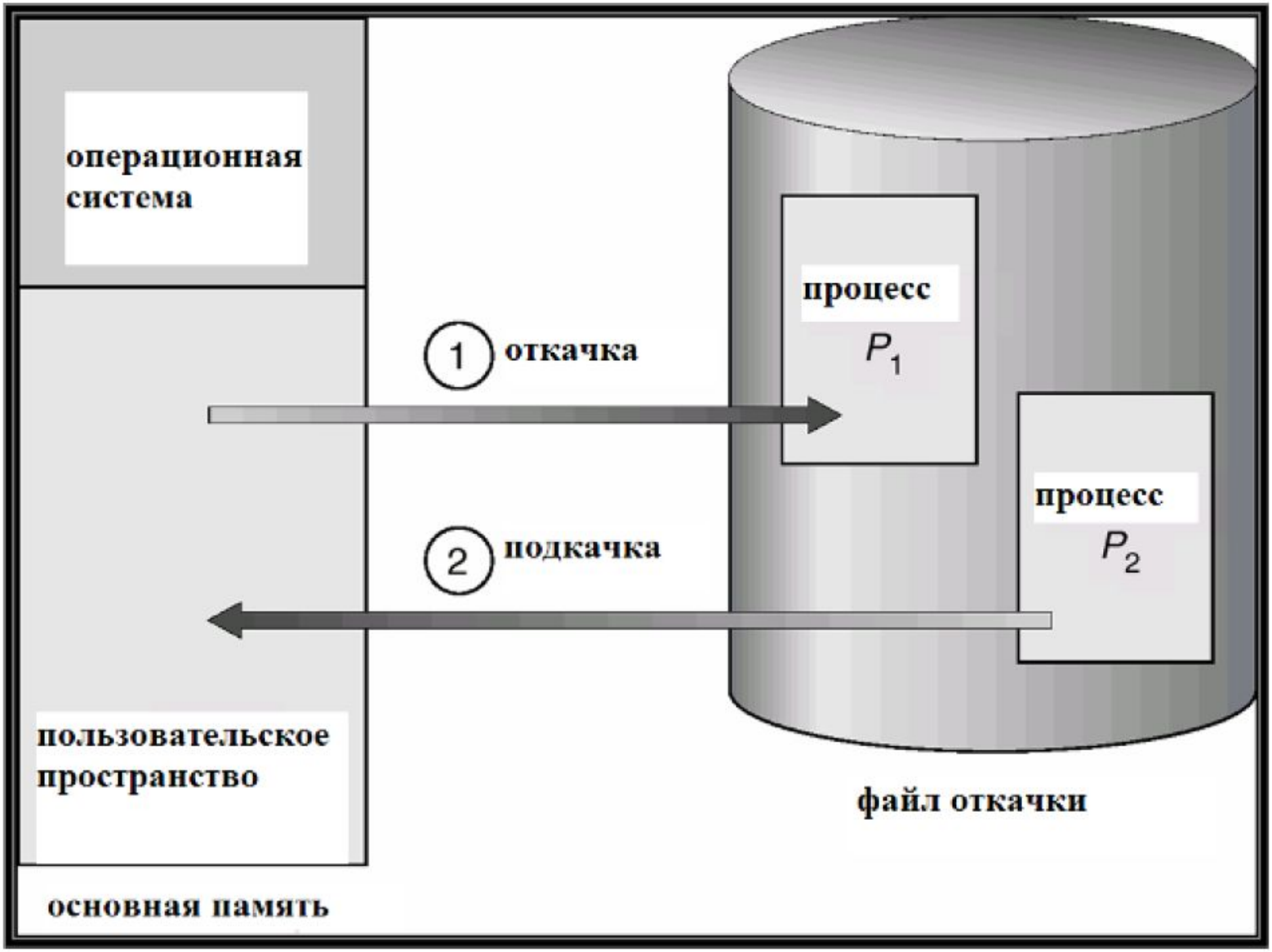
- ***Файл откачки* – пространство на быстром диске, достаточно большое, чтобы разместить образы памяти всех задач; должен быть обеспечен прямой доступ ко всем образам памяти.**

- ***Roll out, roll in* – вариант стратегии откачки и подкачки на базе приоритетов: более приоритетные процессы исполняются, менее приоритетные – откачиваются во внешнюю память.**

- **Наибольшие временные затраты на откачку – затраты на передачу данных; общее время откачки пропорционально объему откачиваемых данных.**

- **Различные стратегии откачки и подкачки реализованы в распространенных ОС - UNIX, Linux, Windows.**

Схема откачки и подкачки

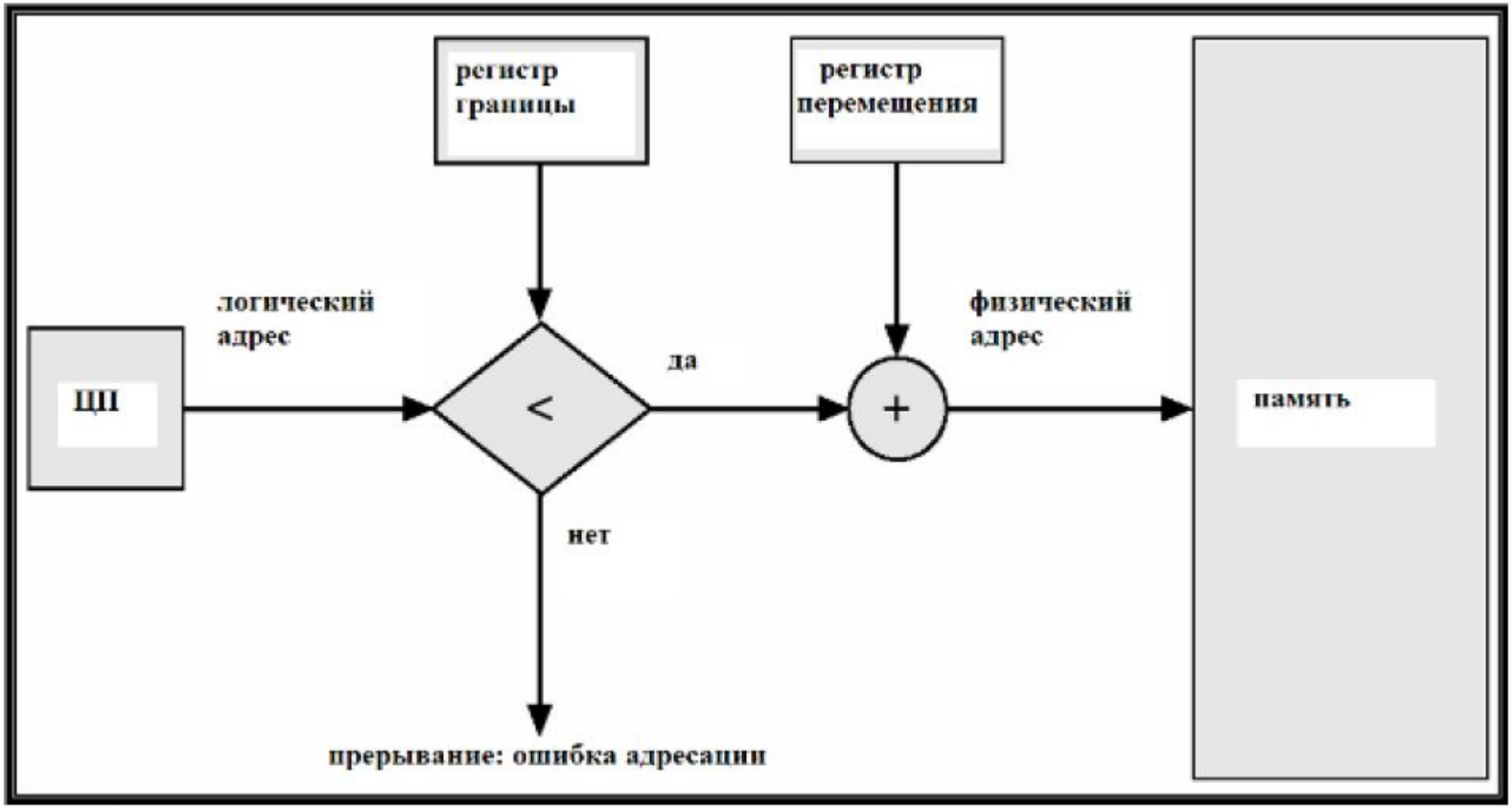


Смежное (contiguous) распределение памяти

- **Обычно основная память разбивается на две непрерывных части (partitions):**
  - **Резидентная ОС + вектор прерываний – по меньшим адресам**
  - **Пользовательские процессы – по большим адресам.**

- **Размещение в одном участке (partition):**
  - **Схема с регистром перемещения используется для защиты пользовательских процессов друг от друга, а также кода и данных ОС – от изменения.**
  - **Регистр перемещения содержит наименьший физический адрес; регистр границы содержит диапазон логических адресов; каждый логический адрес должен быть меньше, чем содержимое регистра границы.**

Аппаратная поддержка регистров перемещения и границы



Смежное распределение (продолжение)

- **Распределение в нескольких** смежных областях

- ***Свободная область (hole)*** **– блок доступной памяти; свободные области разбросаны по памяти.**
  - **При загрузке процесса ему предоставляется память из свободной области, которая достаточно велика для его размещения.**
  - **ОС хранит информацию о:**
    a) размещенных областях b) свободных областях

Стратегии динамического распределения памяти

Как удовлетворить запрос на участок памяти длиной *n*, используя список свободных областей?

- **Метод первого подходящего (First-fit): Выбрать первый по списку достаточно большой свободный участок.**
- **Метод наиболее подходящего (Best-fit): Выбрать *наименьшую* по размеру свободную область достаточно большого размера; требует полного поиска, если список не упорядочен по размеру свободных областей. Приводит к образованию оставшейся части самого маленького размера.**
- **Метод наименее подходящего (Worst-fit): Выбрать подходящую область *наибольшего* размера; также требует полного поиска; приводит к образованию самой большой из возможных оставшейся части.**

Первая и вторая стратегии лучше с точки зрения скорости и эффективности использования памяти

Фрагментация

Внешняя фрагментация – имеется достаточно большая область свободной памяти, но она не является непрерывной.

Внутренняя фрагментация: размер выделенной памяти может быть несколько больше размера запрашиваемой памяти; этот излишек – память, внутренняя для области (partition), но не используемая.

Внешняя фрамментация может быть уменьшена путем компактировки (compaction) :

- **Память перемешивается с целью выделения всей свободной области памяти в один непрерывный блок (возможны разные стратегии – либо простой сдвиг, либо попытка перемешивания методом наиболее подходящего**
- **Компактировка возможна только если перемещение (relocation) происходит динамически. Она выполняется во время исполнения программы.**
- **Проблема ввода-вывода:**
  - **Задача, “застрявшая” в памяти, занятая выполнением ввода-вывода.**
  - **Ввод-вывод должен выполняться только в буфера ОС.**

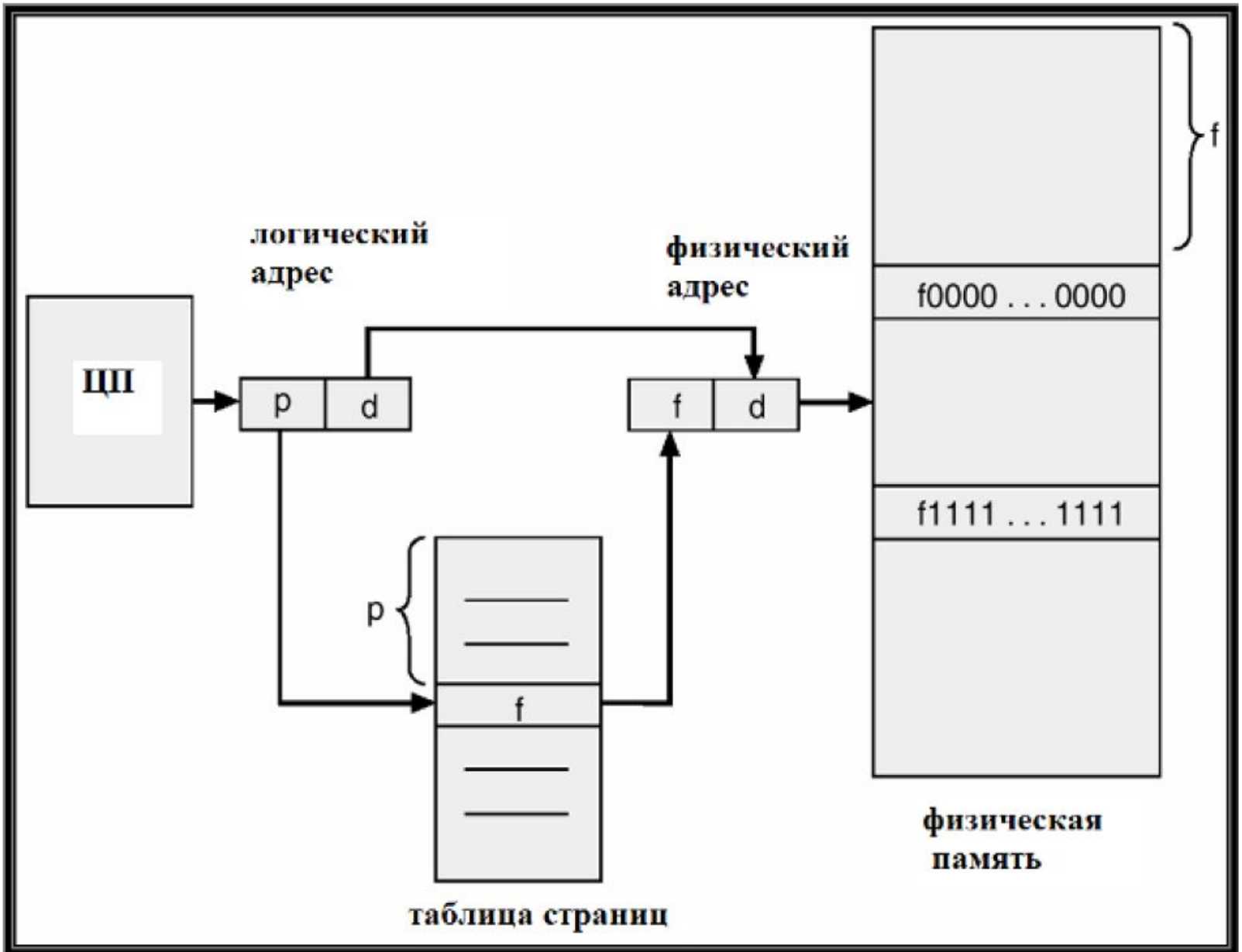
Страничная организация (paging)

- **Логическое адресное пространство процесса может и не быть непрерывным; процессу выделяется физическая память по мере ее освобождения.**
- **Физическая память делится на блоки фиксированной длины, называемые фреймами (frames). Размер - степень 2, от 512 до 8192 байтов**
- **Логическая память делится на блоки такого же размера, называемые страницами.**
- **Поддерживается информация обо всех свободных фреймах.**
- **Для исполнения программы свободных фреймов и загрузить программу.**
- **Вводится таблица страниц для трансляции логических адресов в физические.**
- **Возможна внутренняя фрагментация.**

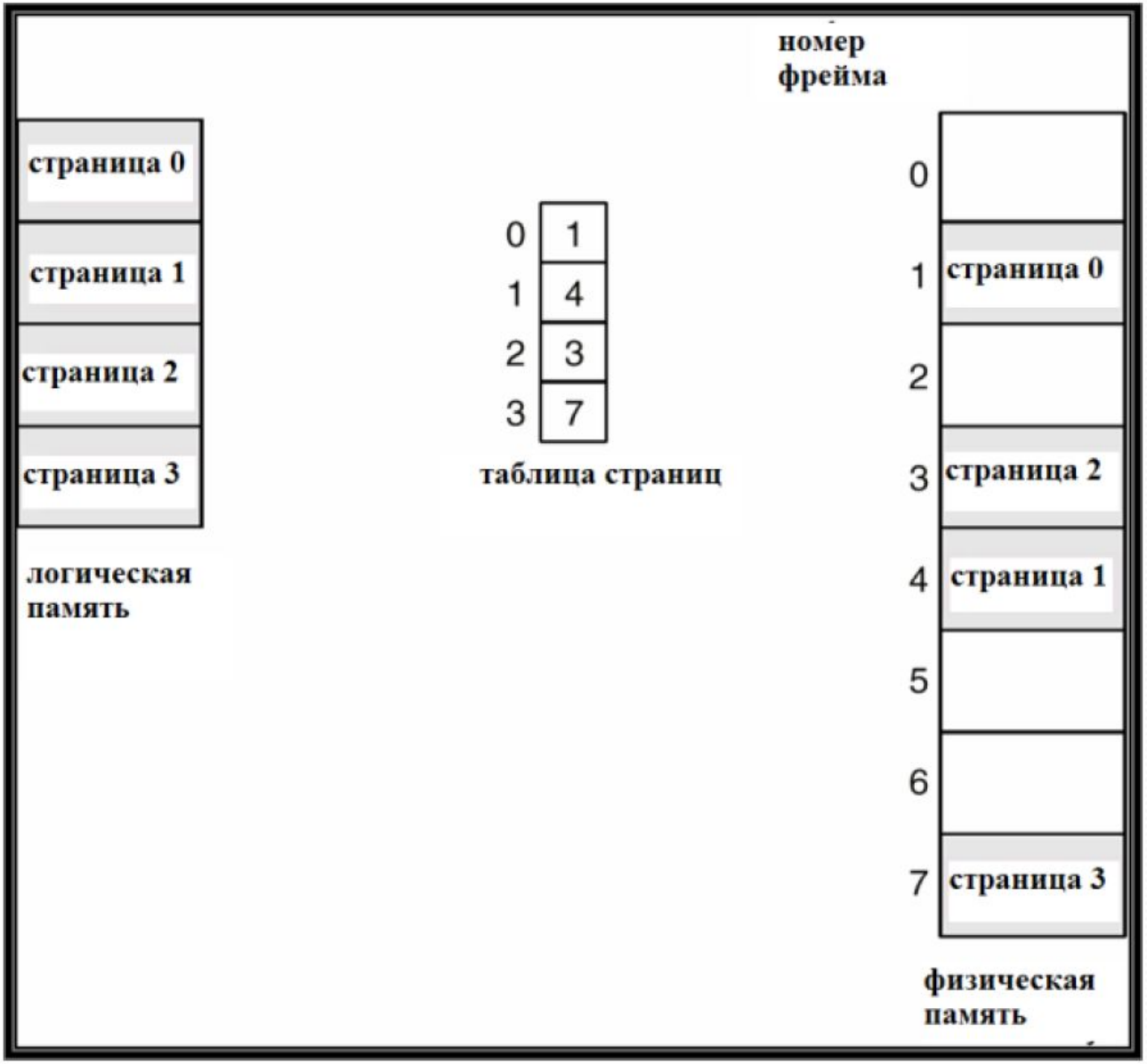
Схема трансляции адресов

- **Адрес, генерируемый процессором, подразделяется на:**
  - ***Номер страницы (p)* – используется как индекс в таблице страниц; содержит базовый адрес (начала) каждой страницы в физической памяти.**
  - ***Смещение внутри страницы (d)* – присоединяется к базовому адресу, в результате чего формируется физический адрес в памяти, посылаемый в MMU.**

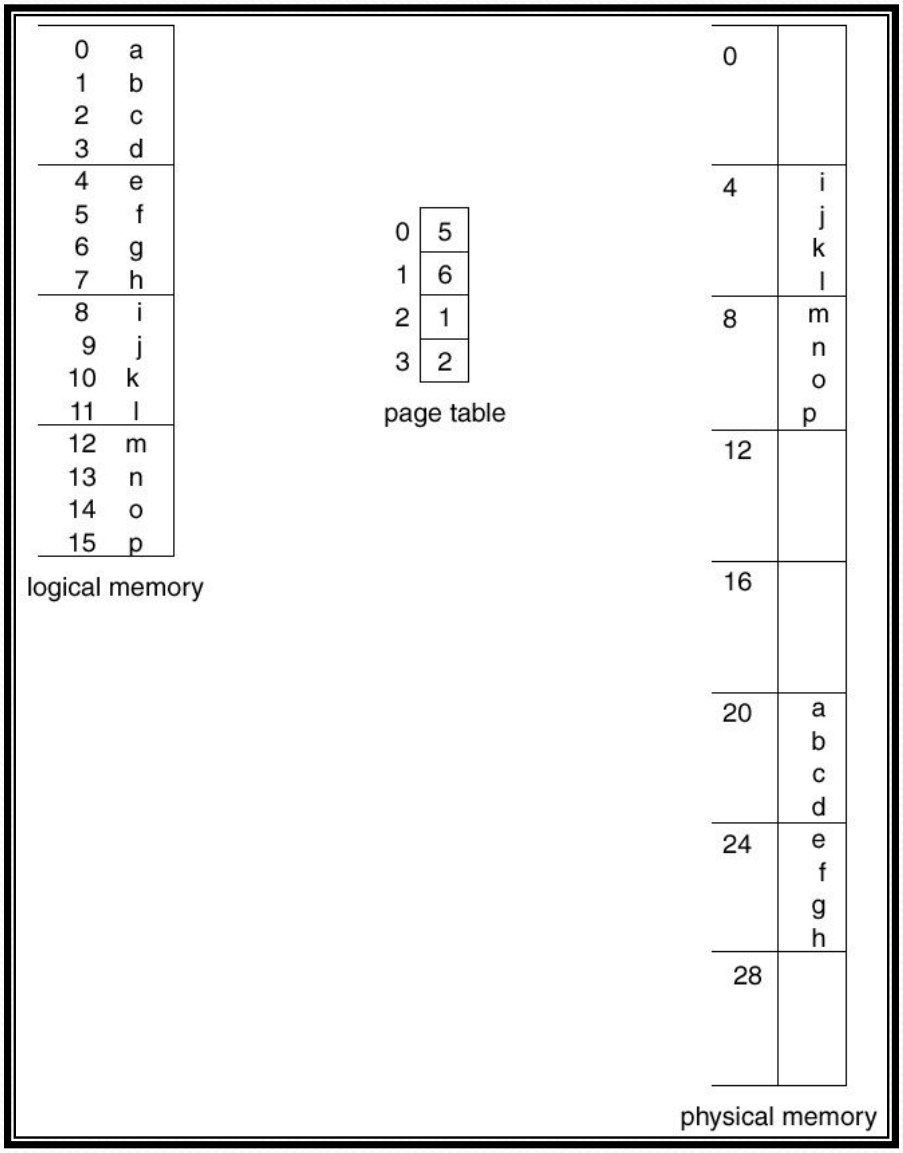
Архитектура трансляции адресов



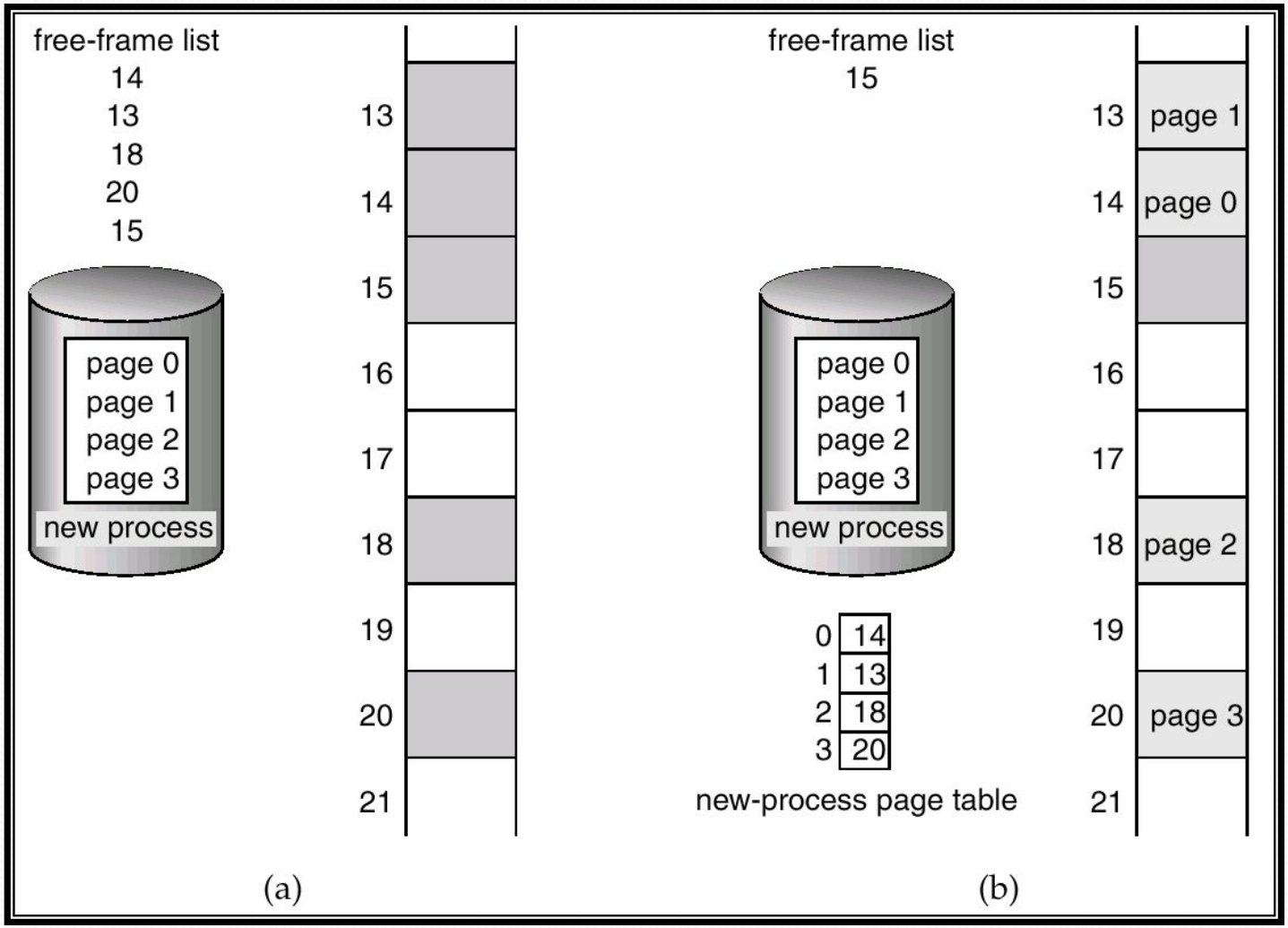
Пример страничной организации



Пример страничной организации



Список свободных фреймов



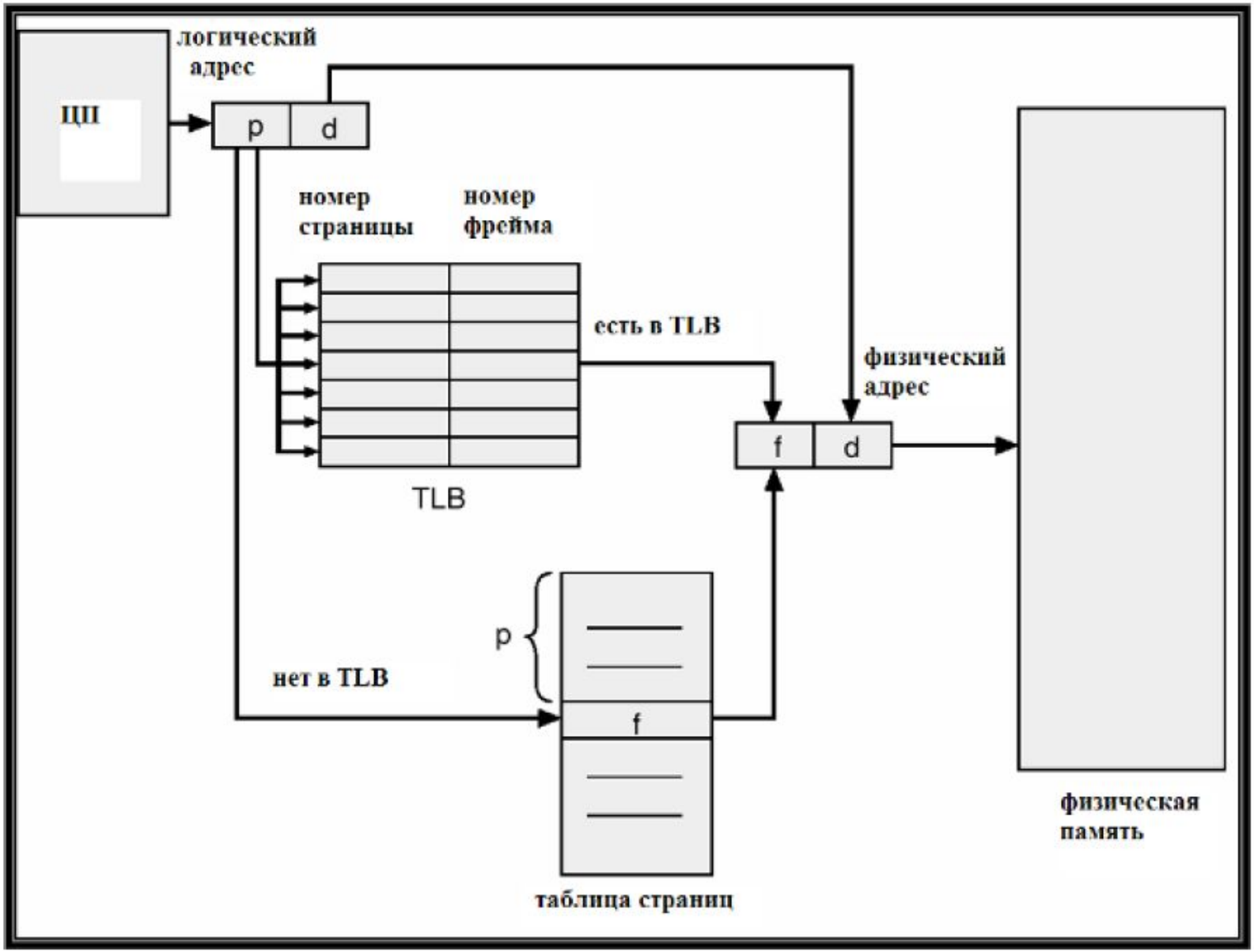
Реализация таблицы страниц

- **Таблица страниц хранится в основной памяти.**
- **Базовый регистр таблицы страниц(PTBR) указывает на таблицу задает длину таблицы страниц.**
- **При такой схеме доступ к любым данным или команде требует двух обращений к памяти – одно в таблицу страниц, другое – собственно к данным (команде).**
- **Эта проблема двух обращений в память может быть решена кешированием (cache), т.е. Использованием ассоциативной памяти, иначе называемой *translation look-aside buffers (TLBs)***
- **“Эльбрус” – РТСП – регистр таблицы страниц пользователя; размер таблицы страниц хранился в ее дескрипторе, как для любого массива**

Ассоциативная память

- **Ассоциативная память: параллельный поиск**
- **Таблица пар: (*номер страницы, номер фрейма*)**
- **Трансляция адресов (A′, A′′)**
  - **Если A′ - на ассоциативном регистре, найти соответствующий номер фрейма.**
  - **Иначе выбрать номер фрейма из таблицы страниц в памяти**

Аппаратная поддержка страничной орга



Эффективное время поиска (effective access time – EAT)

- **Ассоциативный поиск = ε единиц времени**
- **Предположим, что цикл памяти – 1 микросекунда**
- **Hit ratio – отношение, указывающее, сколько раз (в среднем) номер страницы будет найден по ассоциативным регистрам. Отношение зависит от размера кэш-памяти .**
- **Hit ratio = α**
- **Effective Access Time (EAT)**

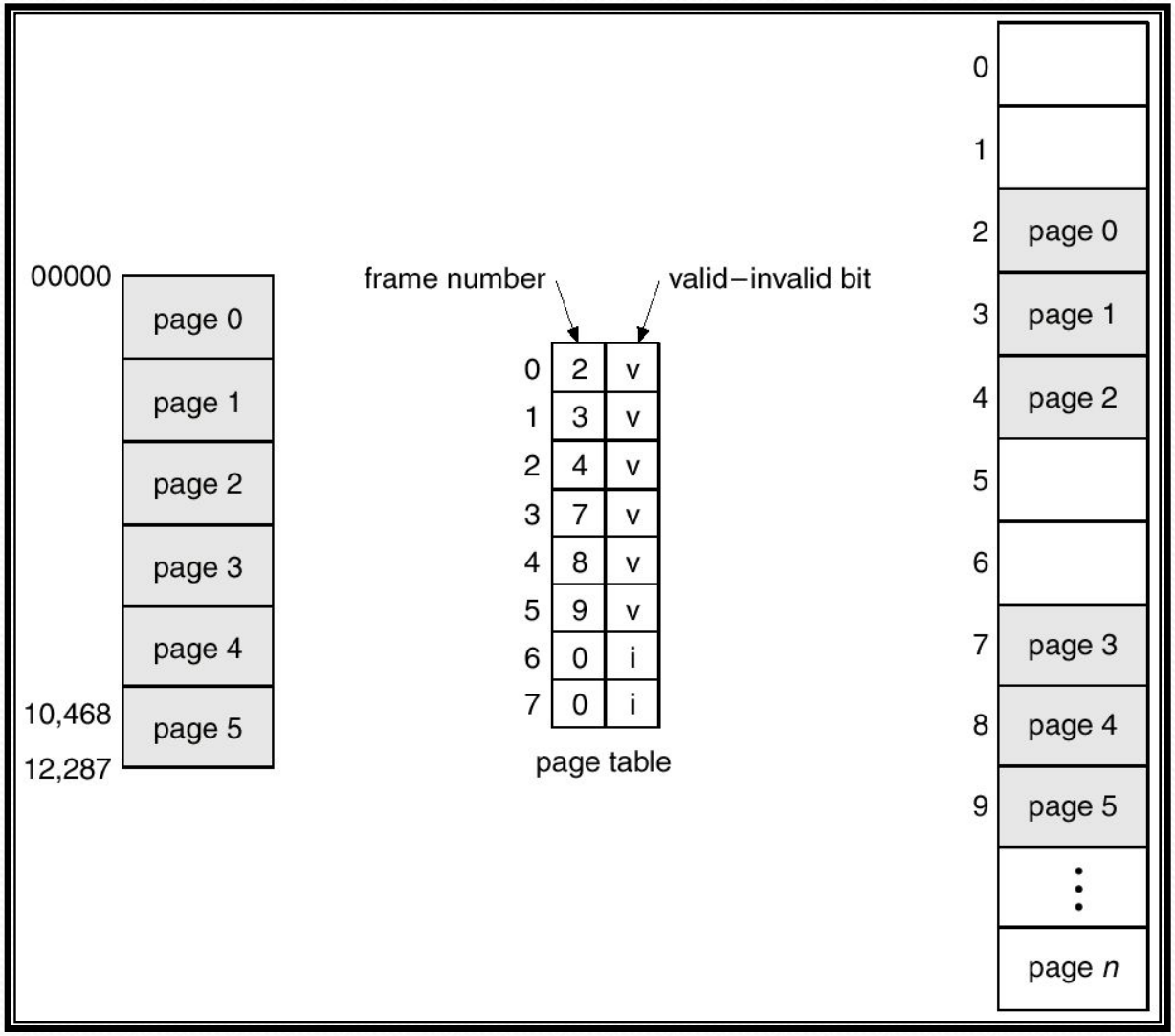
$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha) = 2 + \varepsilon - \alpha$$

Защита памяти

- **Защита памяти может быть реализована описанием каждого фрейма специальным битом (valid/invalid).**

- **Данный бит хранится в каждом элементе таблицы страниц:**
  - **“valid” указывает, что данная страница принадлежит логической памяти процесса.**
  - **“invalid” указывает, что это не так.**

Бит valid/invalid в таблице страниц



Структура таблицы страниц

- **Иерархические таблицы страниц**
- **Хешированные таблицы страниц**
- **Инвертированные таблицы страниц**

Иерархические таблицы страниц

- **Логическое адресное пространство разбивается на несколько таблиц страниц.**

- **Простой метод: двухуровневая таблица страниц.**

Пример двухуровневой таблицы страниц

- **Логический адрес (на 32-битной машине с размером страницы 4 K) подразделяется на:**
  - **Номер страницы (20 битов).**
  - **Смещение на странице (12 битов).**
- **Поскольку таблица страниц сама делится на страницы, номер страницы далее подразделяется на:**
  - **Номер страницы (10 битов).**
  - **Смещение на странице.**
- **Таким образом, логический адрес имеет следующую структуру:**

  (p1, p2, d)

  где *p1* – индекс во внешней таблице страниц, *p2* – смещение внутри страницы для внешней таблицы страниц

Схема двухуровневых таблиц страниц

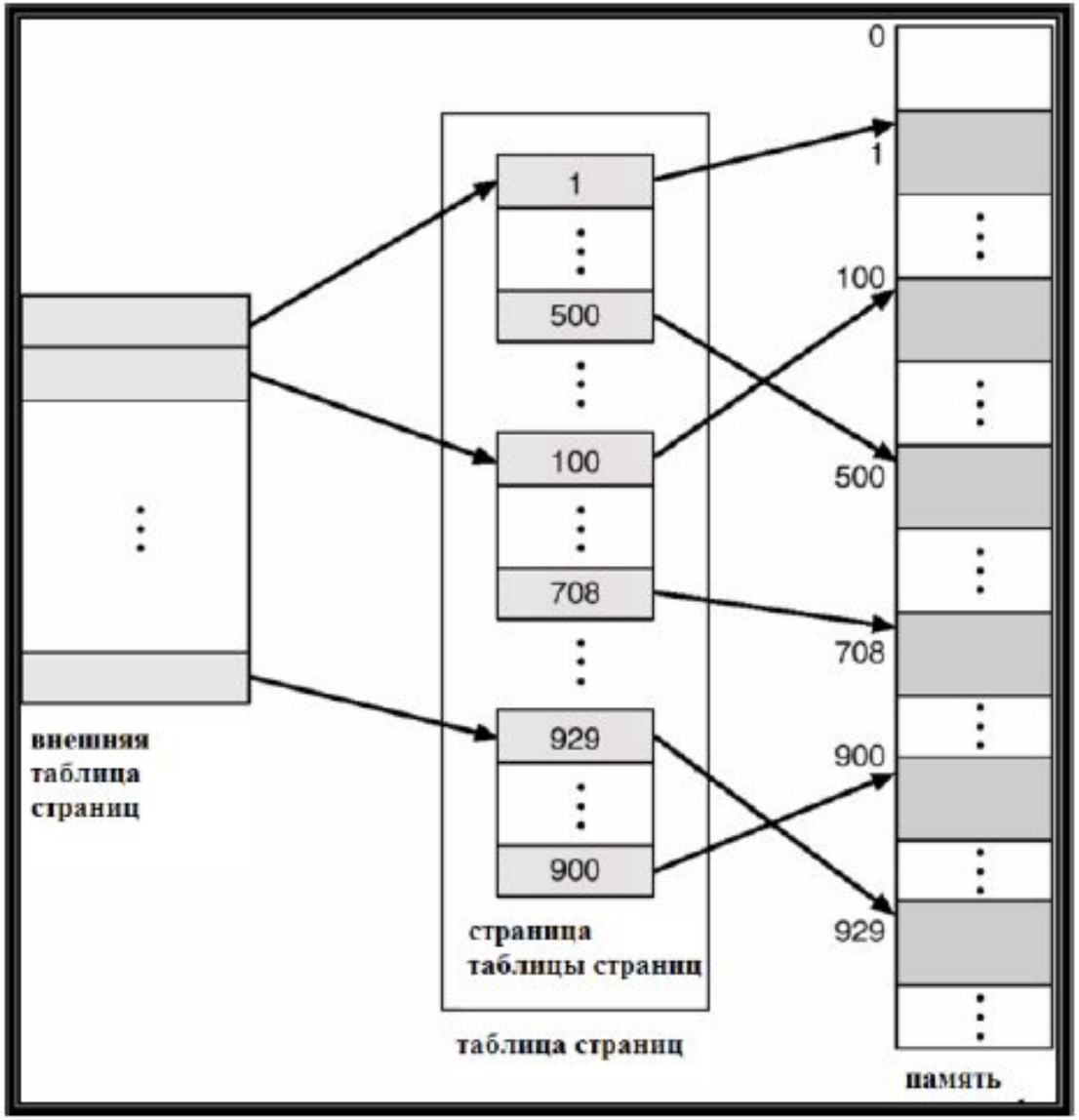
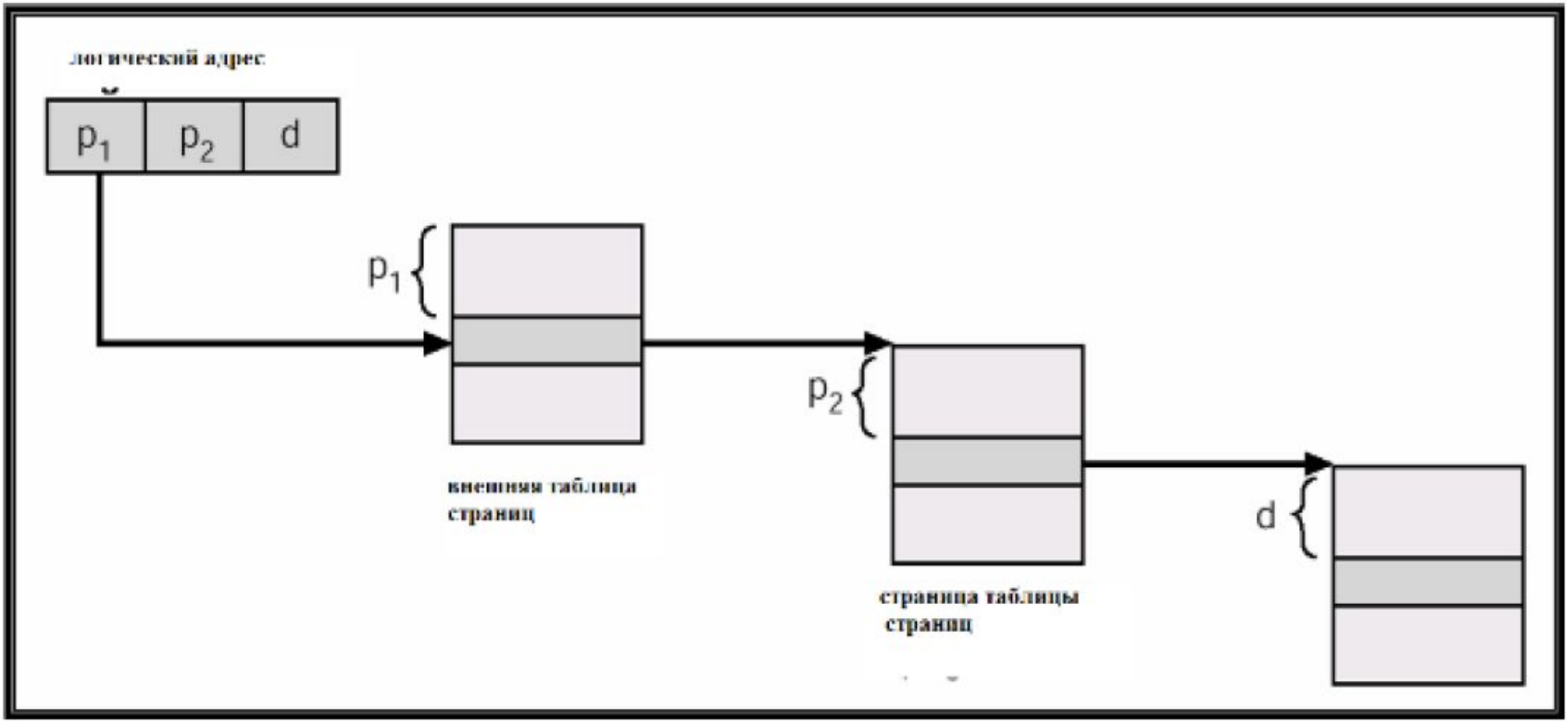


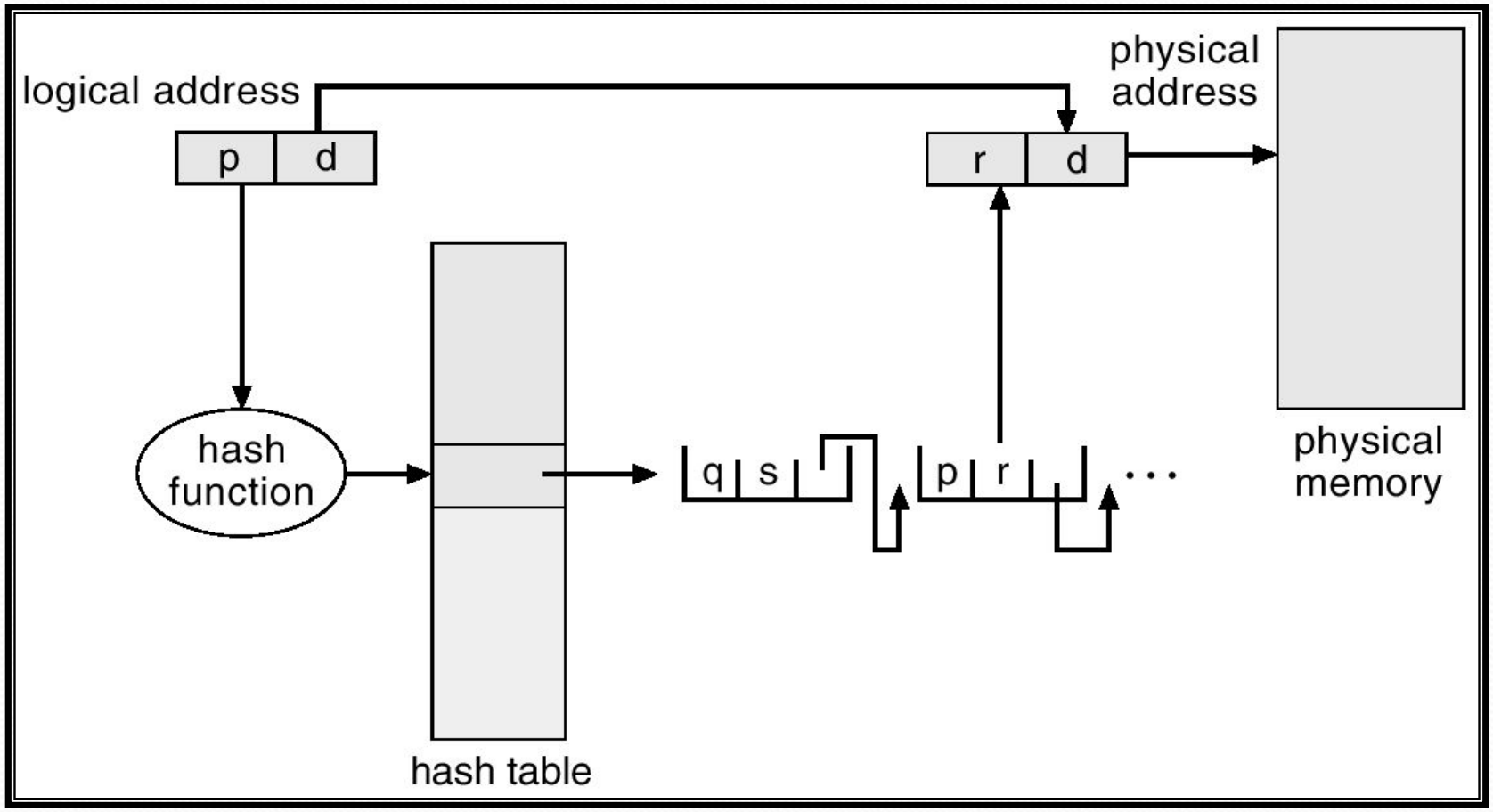
Схема адресной трансляции



Хешированные таблицы страниц

- **Общеупотребительны, если адресное пространство > 32 битов.**
- **Виртуальный номер страницы в результате вычисления хеш-функции преобразуется в реальный номер страницы. Таблица страниц содержит цепочку элементов, хешируемых в один и тот же номер**
- **Виртуальные номера страниц сравниваются в этой цепочке в поисках совпадения. Если совпадение найдено, извлекается соответствующий физический фрейм**

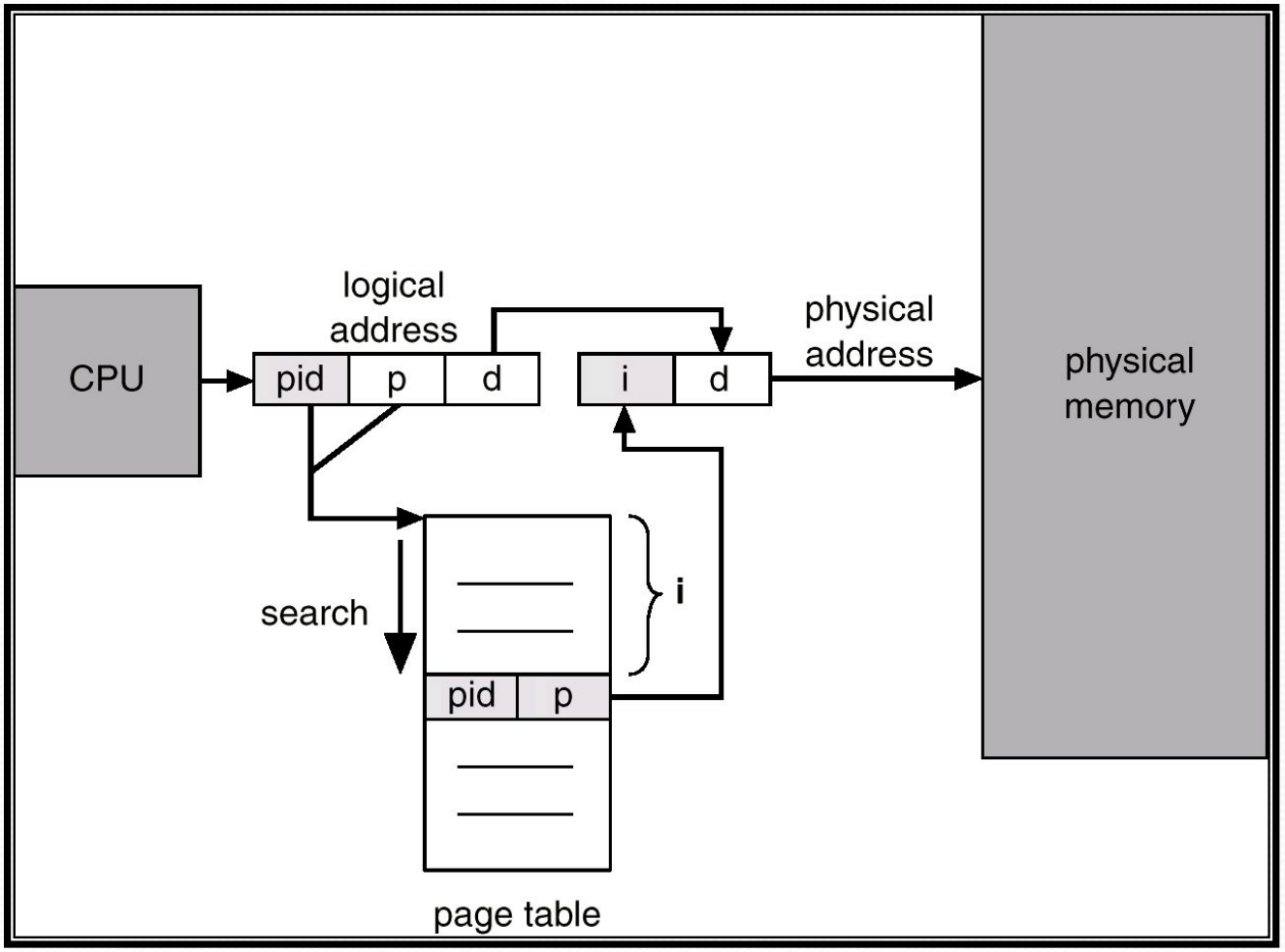
Хешированная таблица страниц



Инвертированная таблица страниц

- **Один элемент для каждой реальной страницы в памяти.**
- **Элемент состоит из виртуального адреса страницы, хранимой по данному адресу памяти, а также информации о процессе, который владеет данной страницей.**
- **Уменьшает объем памяти для хранения каждой таблицы страниц, но увеличивает время поиска страницы, на которую выполнена ссылка.**
- **Хеш-таблица используется для сокращения поиска до одного (или небольшого числа) элементов таблицы страниц.**

Архитектура инвертированной таблицы страниц

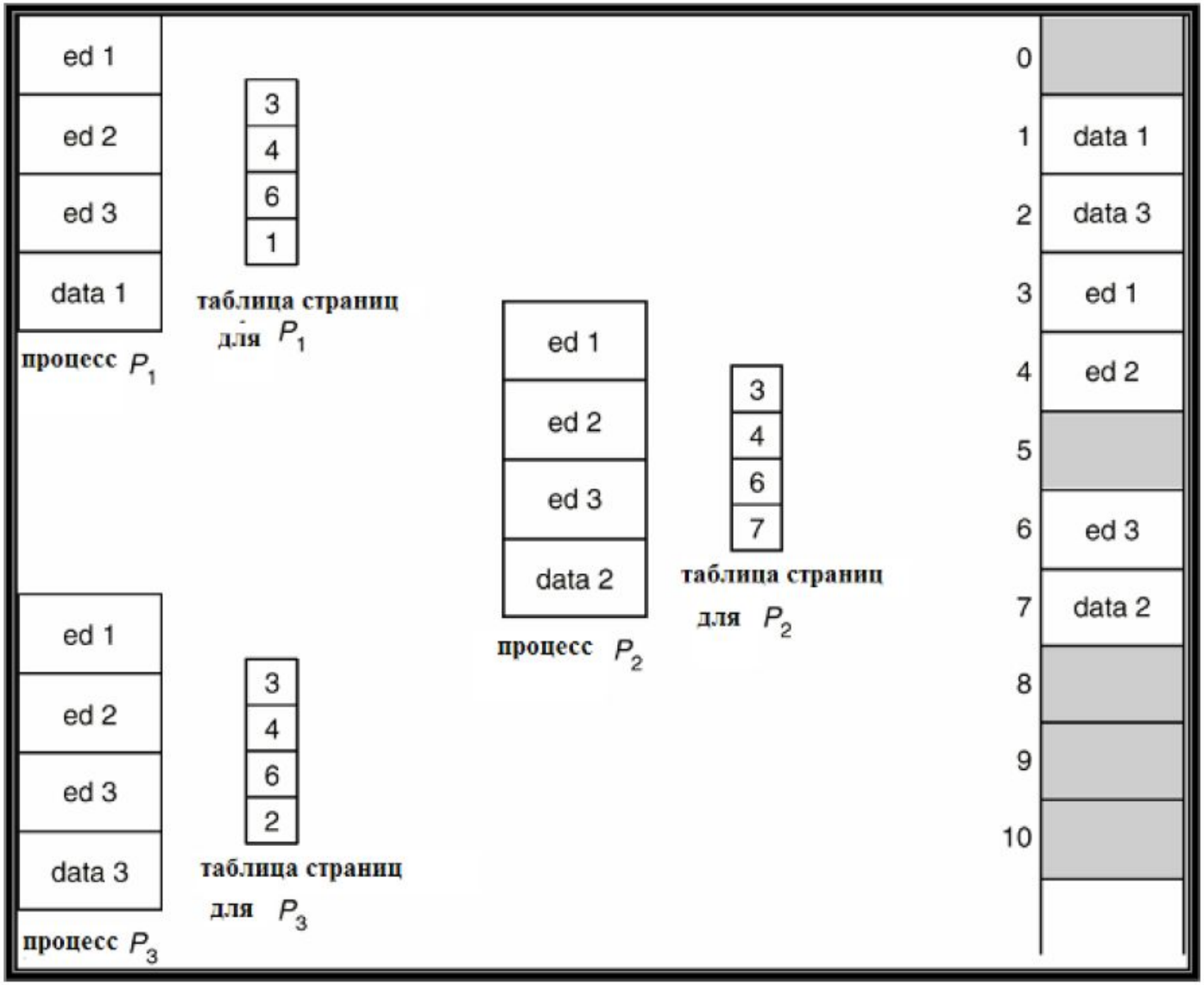


Разделяемые страницы

- **Разделяемый код**
  - **Единственная копия неизменяемого (реентерабельного - reentrant) кода, совместно используемого несколькими процессами, - текстовые редакторы, компиляторы, графические среды и др.**
  - **Разделяемый код должен располагаться по одним и тем же относительным адресам в логических адресных пространствах всех процессов .**

- **Локальный код и данные**
  - **Каждый процесс хранит отдельную копию кода и данных**
  - **Страницы с локальным кодом и данными могут быть расположены в любом месте логического адресного пространства.**

Пример: разделяемые страницы



Операционные системы

Сегментная организация памяти

Сегментная организация памяти (segmentation)

- **Схема распределения памяти, соответствующая пользовательской трактовке распределения памяти.**
- **Программа – набор *сегментов*. Сегмент – это логическая единица, такая, как, например:**

  основная программа,

  процедура,

  функция,

  метод,

  объект,

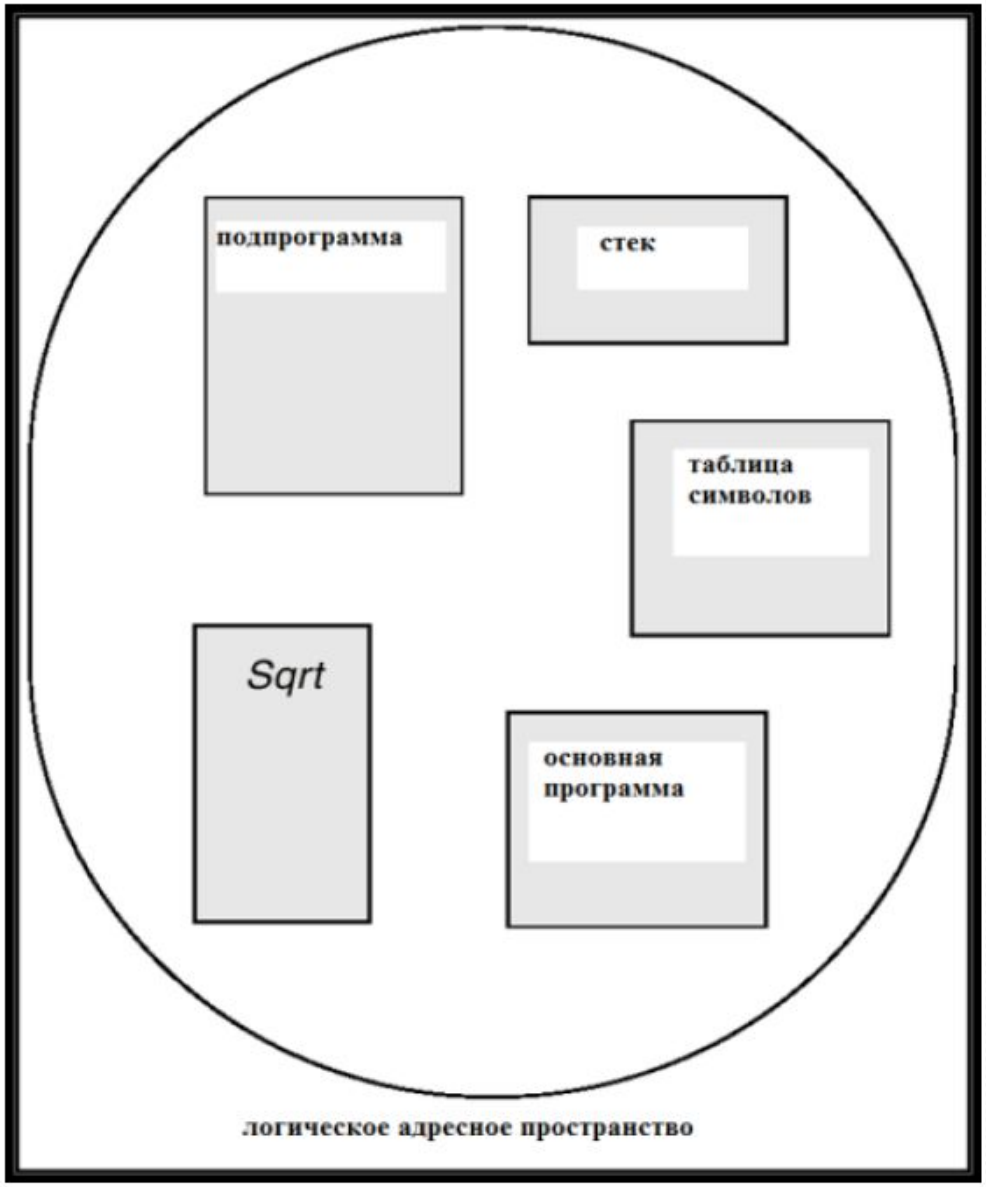
  локальные переменные; глобальные переменные,

  общий блок (COMMON blocks – Fortran),

  стек

  таблица символов; массивы

Программа с точки зрения пользователя



Архитектура сегментной организации памяти

- **Логический адрес ~ пара: <segment-number, offset>,**

где segment-number – номер сегмента, offset – смещение в сегменте.

- ***Таблица сегментов* – служит для отображения логических адресов в физические; каждый ее элемент содержит:**
  - ***base* – начальный адрес сегмента в оперативной (физической) памяти.**
  - ***limit* – длину сегмента.**
- ***Базовый регистр таблицы сегментов - segment-table base register (STBR)* срдержит адрес таблицы сегментов в памяти.**
- ***Регистр длины таблицы сегментов - segment-table length register (STLR)* содержит число сегментов, используемое программой;**

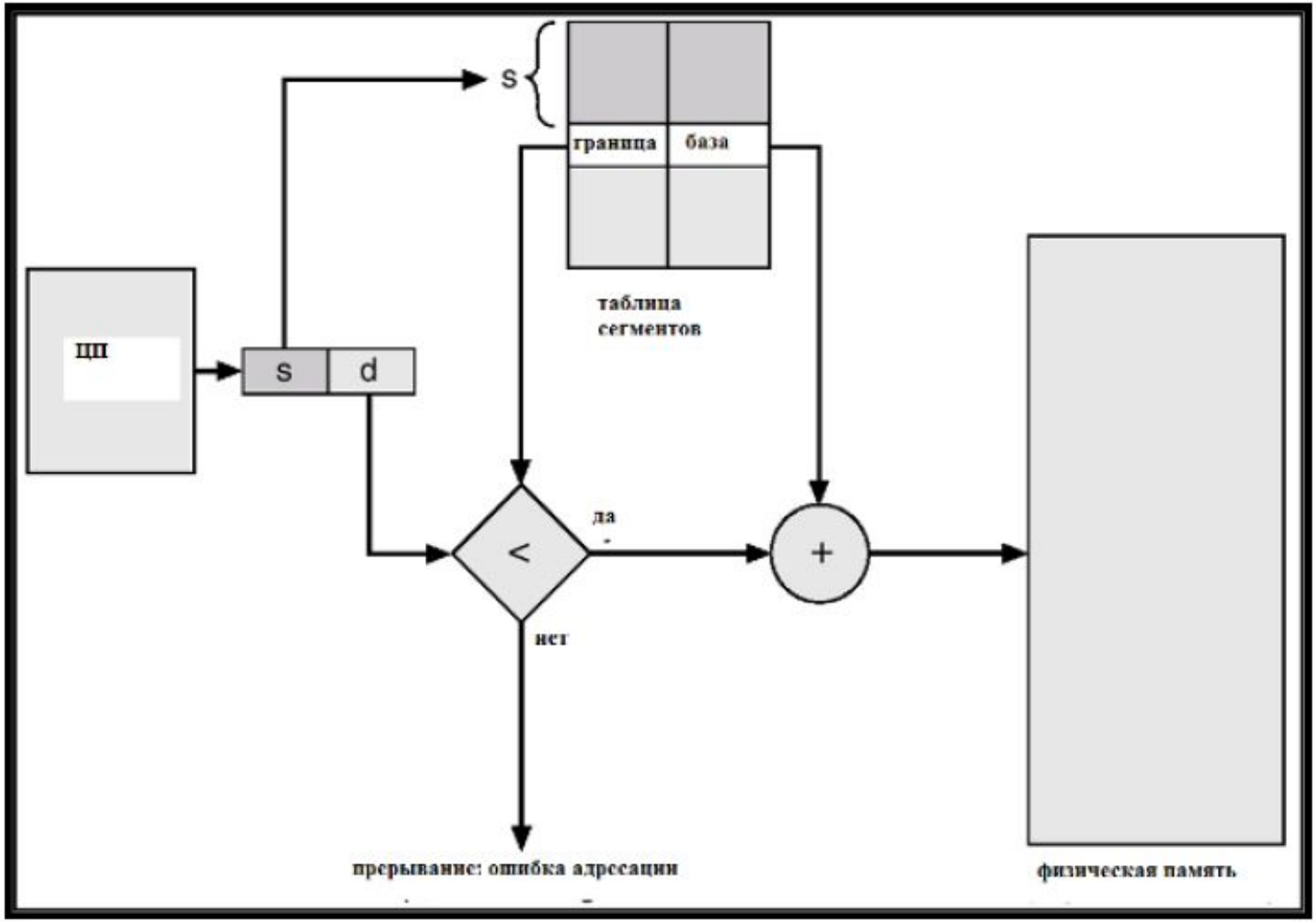
номер сегмента *s* корректен, если *s* < STLR.

Архитектура сегментной организации (продолжение)

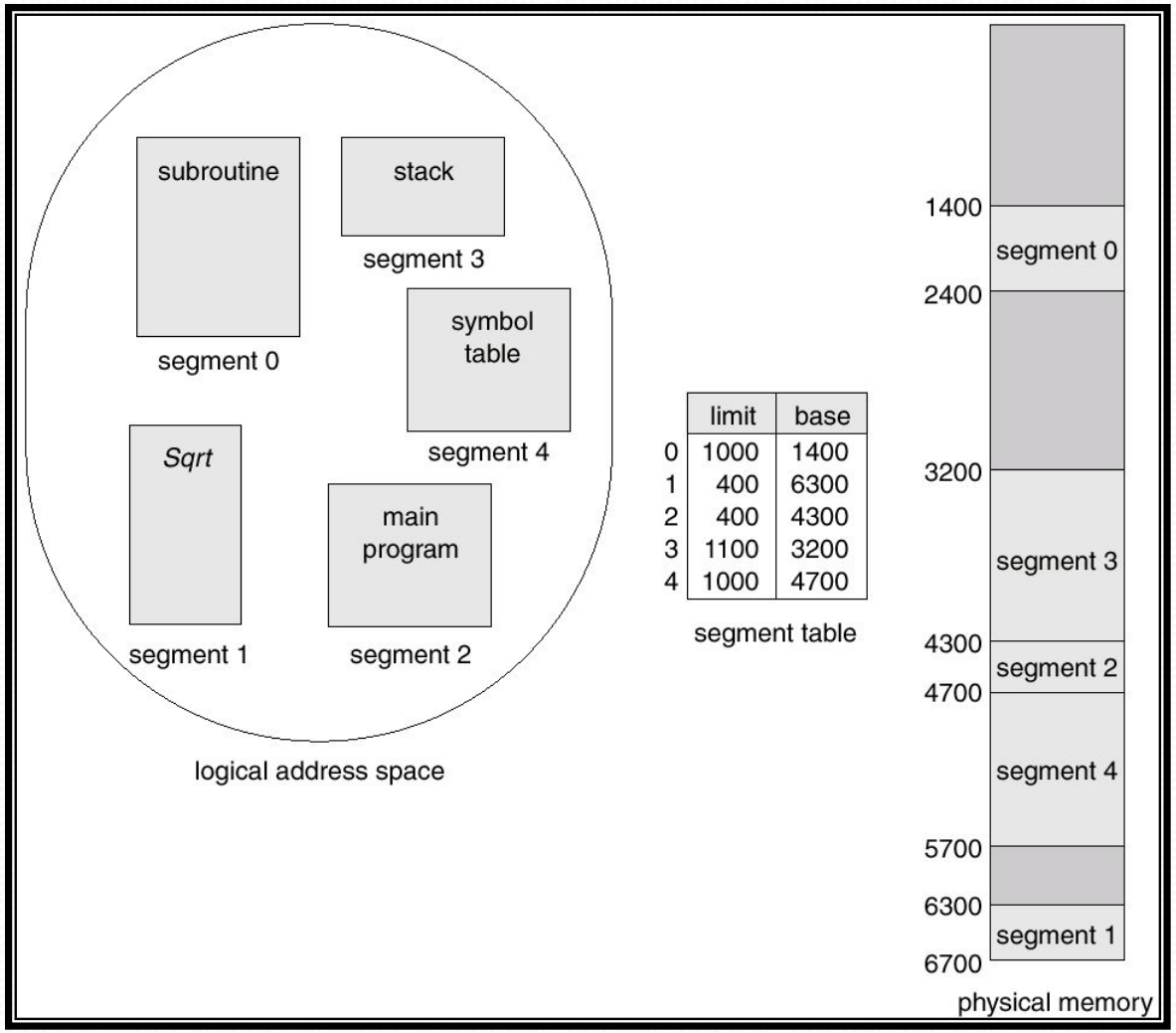
- **Перемещение (Relocation).**
  - **динамическое**
  - **с помощью таблицы сегментов.**
- **Общий доступ (Sharing).**
  - **Разделяемые сегменты**
  - **Один и тот же номер сегмента**
- **Стратегии распределения памяти.**
  - **first fit / best fit**
  - **Внешняя фрагментация**

- ***Защита*. С каждым элементом таблицы сегментов связываются:**
  - **validation bit = 0 ⇒ неверный сегмент**
  - **read/write/execute - полномочия**
- **Биты защиты связываются с сегментами; совместный доступ к коду осуществляется на уровне сегментов.**
- **Поскольку сегменты различаются по длине, распределение памяти в виде сегментов – это общая задача динамического распределения памяти.**
- **Пример сегментной организации приведен на диаграмме на следующих слайдах**

Аппаратная поддержка сегментного распределения памяти



Пример сегментной организации памяти



Совместное использование сегментов