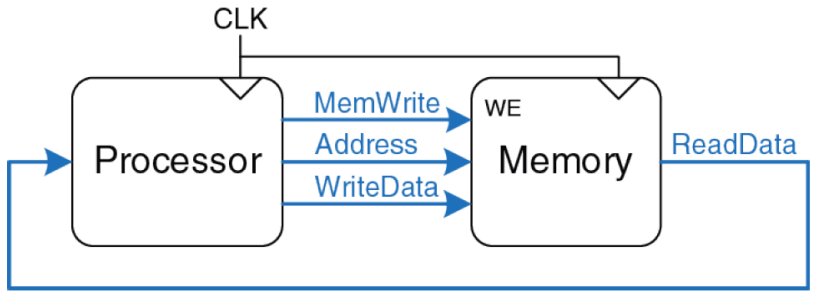
Архитектура компьютера

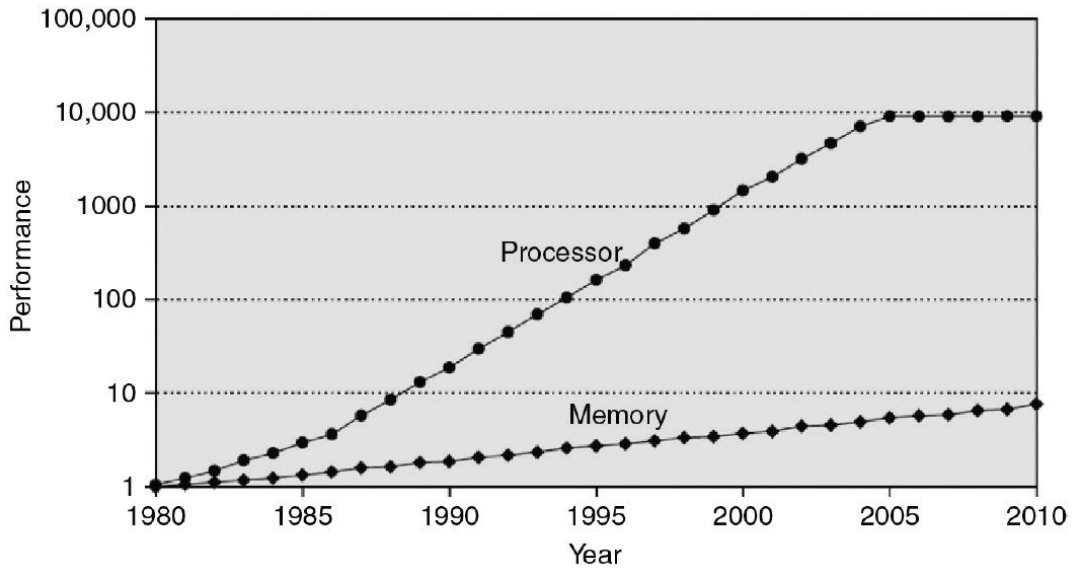
Виртуальная память.

План лекции

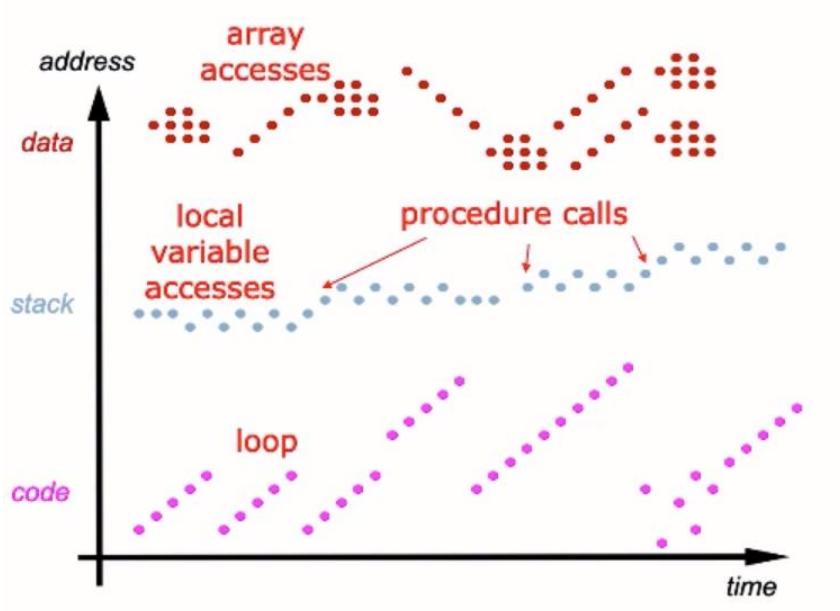
- Виртуальная память
- Буфер ассоциативной трансляции

Процессор–память

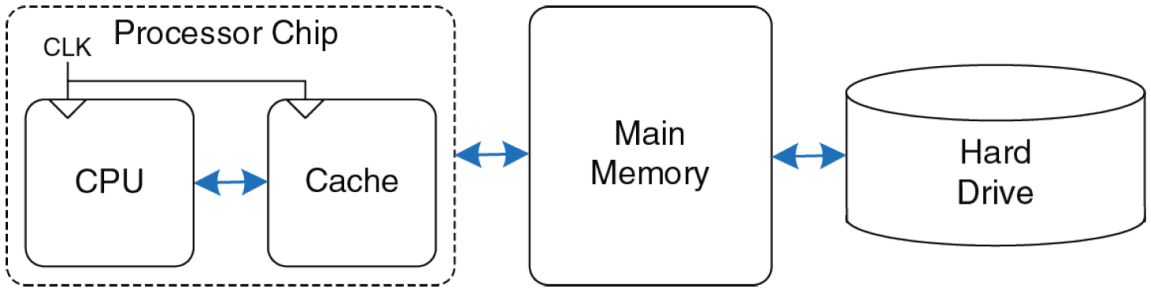


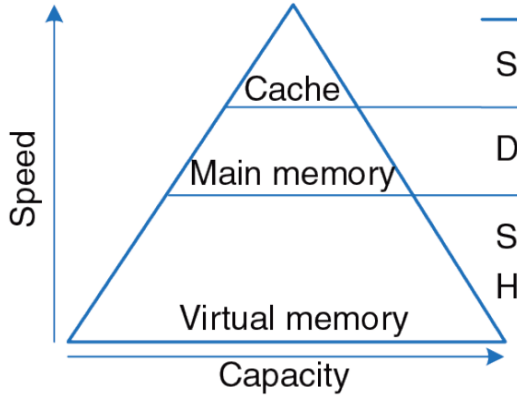


Пространственная и временная локальность данных



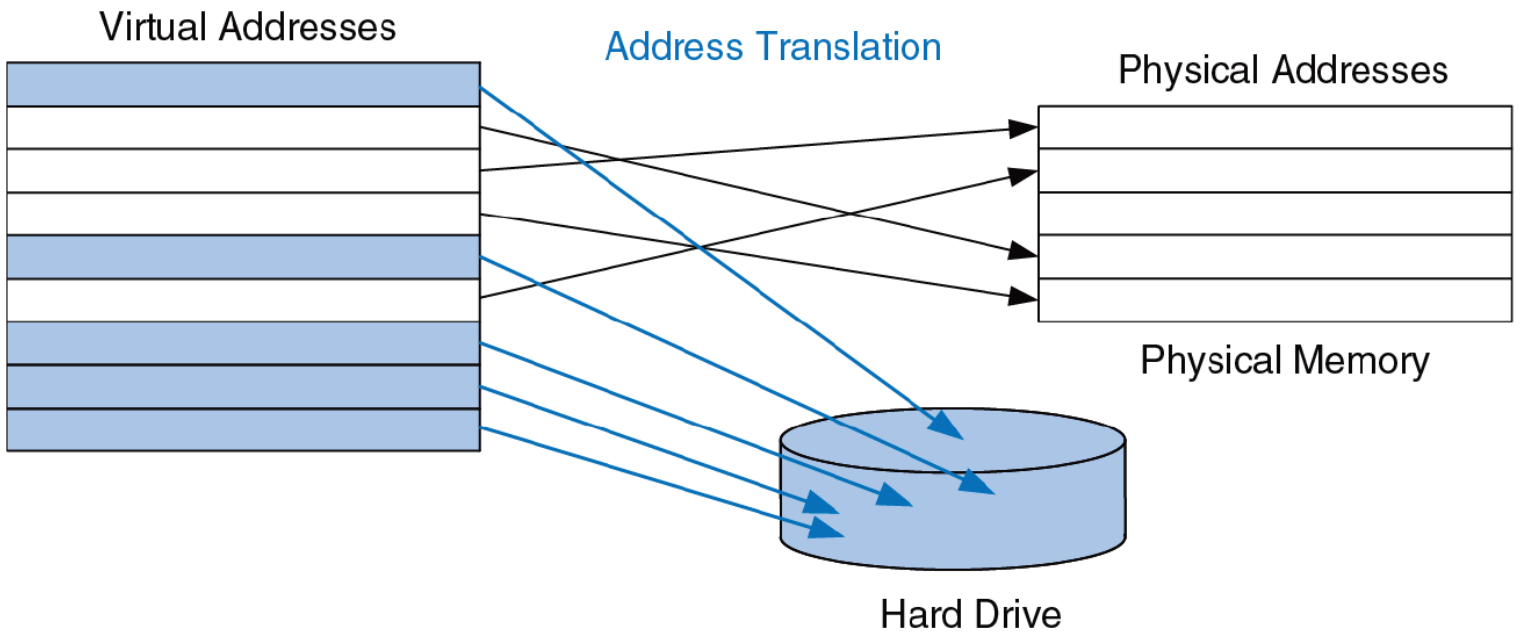
Иерархия памяти





Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$10,000	1	25+
DRAM	$10	10–50	10
SSD	$1	100,000	0.5
HDD	$0.1	10,000,000	0.1

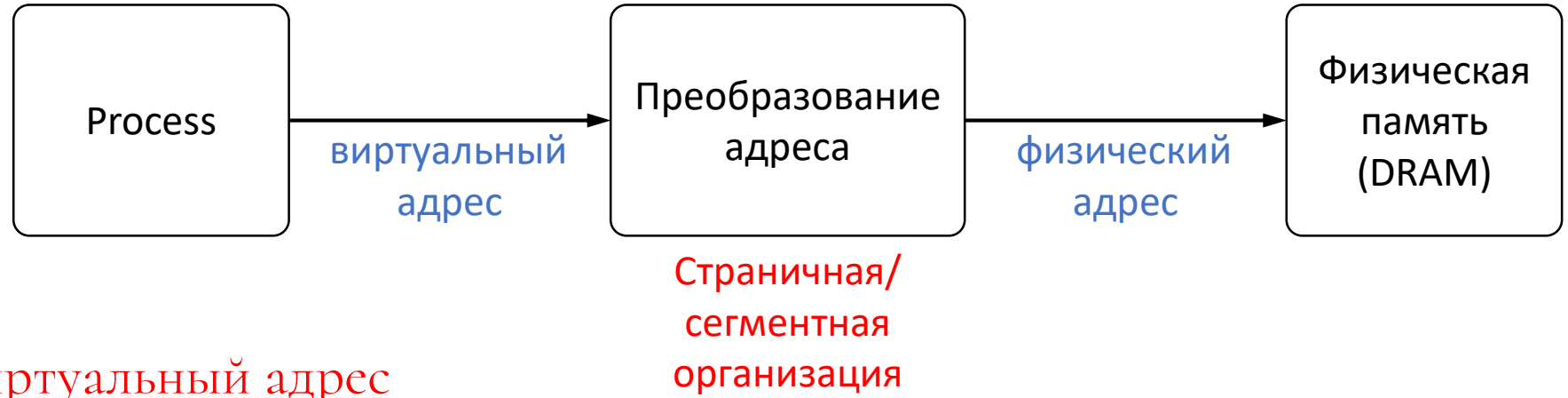
Виртуальная память



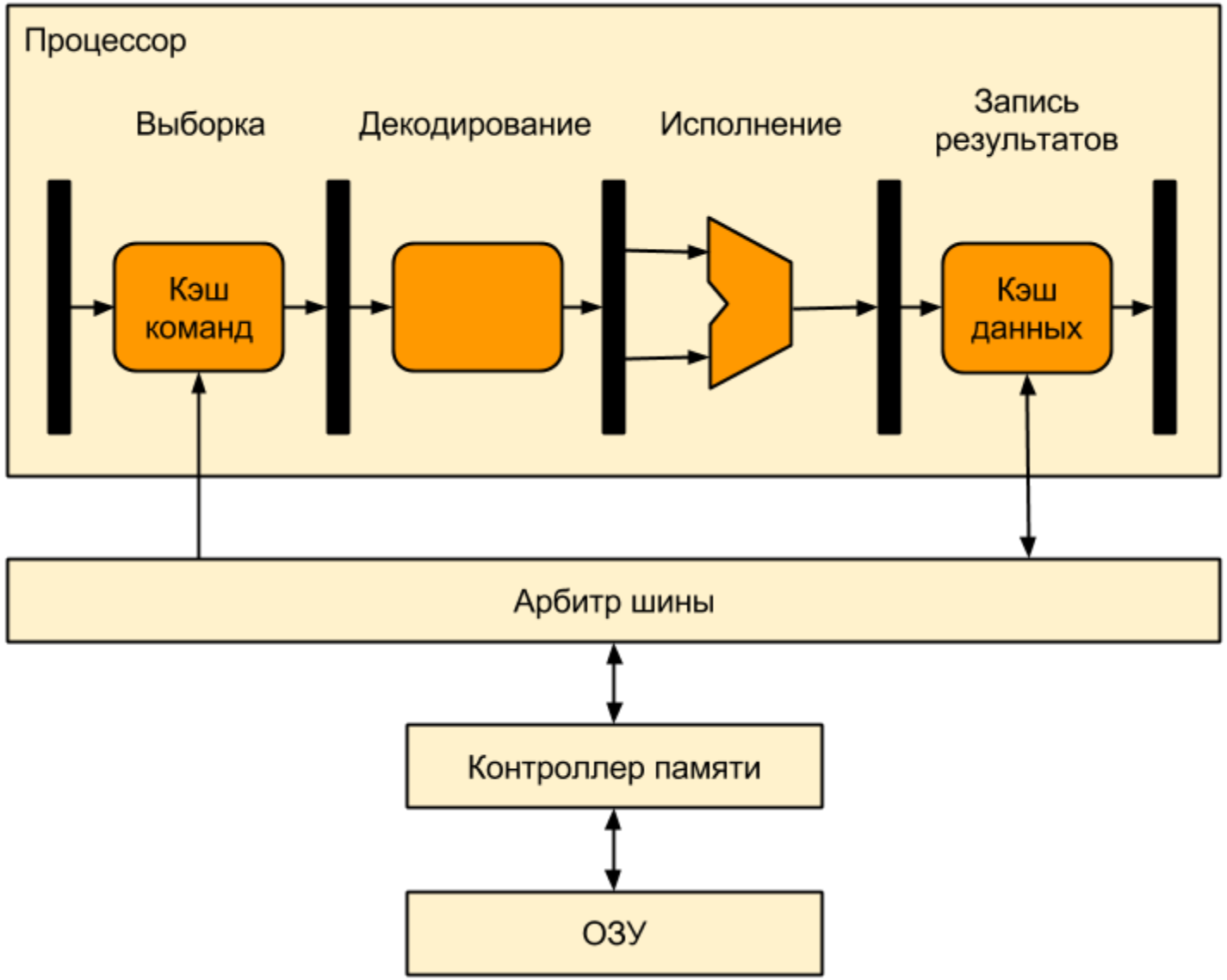
Решаемые задачи:

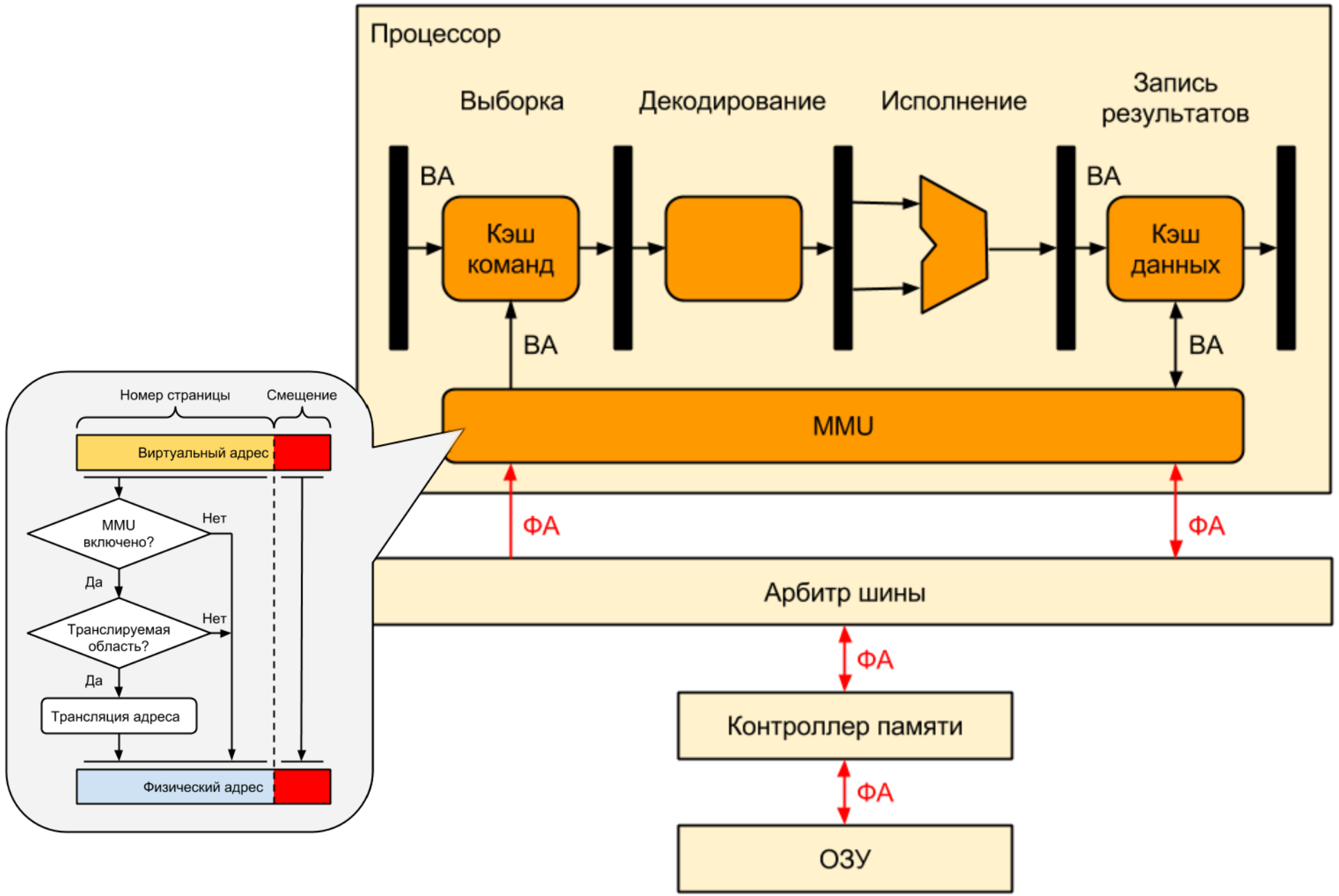
- поддержка изоляции процессов и защиты памяти путём создания своего собственного виртуального адресного пространства для каждого процесса
- поддержка изоляции области ядра от кода пользовательского режима
- поддержка памяти только для чтения и с запретом на исполненение
- поддержка выгрузки не используемых участков памяти в область подкачки на диске (свопинг)
- поддержка отображённых в память файлов, в том числе загрузочных модулей
- поддержка разделяемой между процессами памяти, в том числе с копированием-при-записи для экономии физических страниц

Названия адресов

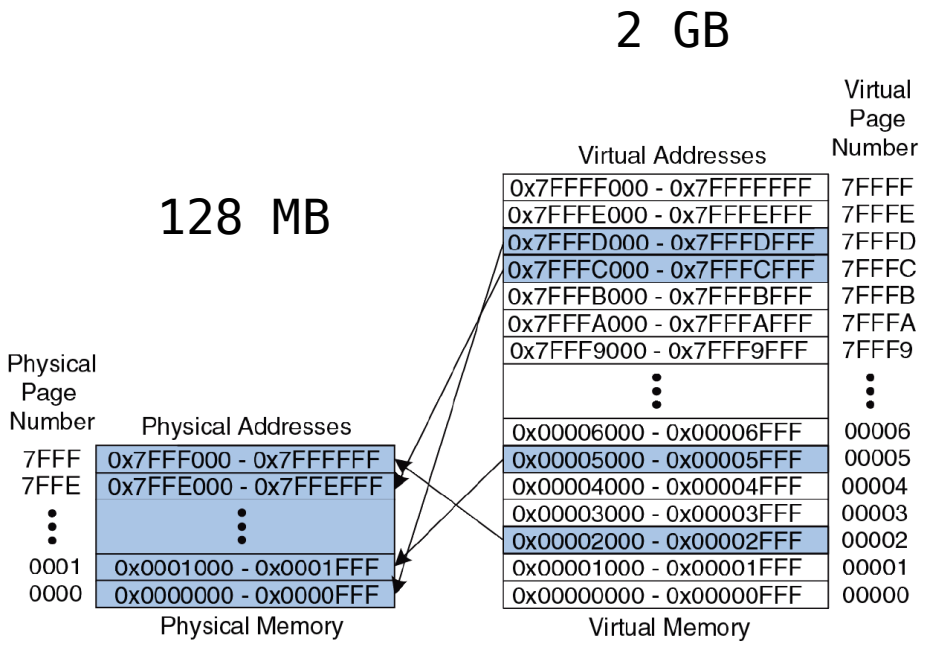


- Виртуальный адрес
  - Адрес генерируется процессом
  - Специфичное приватное адресное пространство процесса
- Физический адрес
  - Адрес используется для доступа к физической памяти
  - Операционная система определяет отображение виртуальных адресов на физические

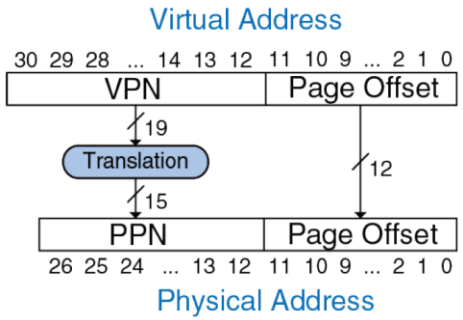

Процессор
Выборка
Декодирование
Исполнение
Запись результатов
Кэш команд
Кэш данных
Арбитр шины
Контроллер памяти
ОЗУ


Процессор
Выборка
Декодирование
Исполнение
Запись результатов
ВА
Кэш команд
ВА
ВА
Кэш данных
ВА
MMU
ФА
ФА
Арбитр шины
ФА
Контроллер памяти
ФА
ОЗУ
Номер страницы
Смещение
Виртуальный адрес
MMU включено?
Нет
Да
Транслируемая область?
Нет
Да
Трансляция адреса
Физический адрес

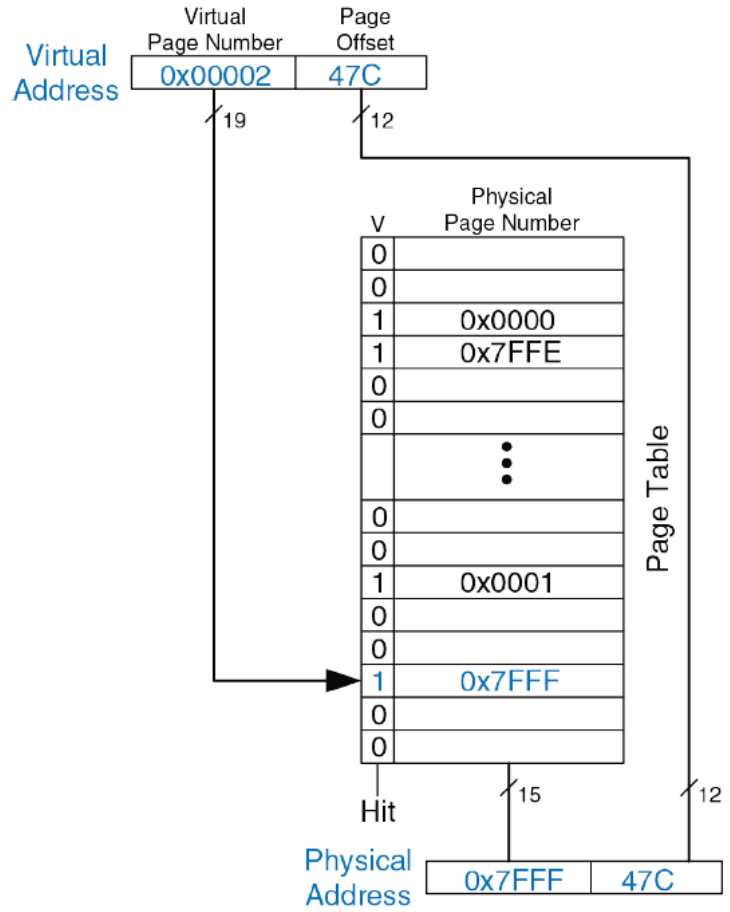
Физические и виртуальные страницы



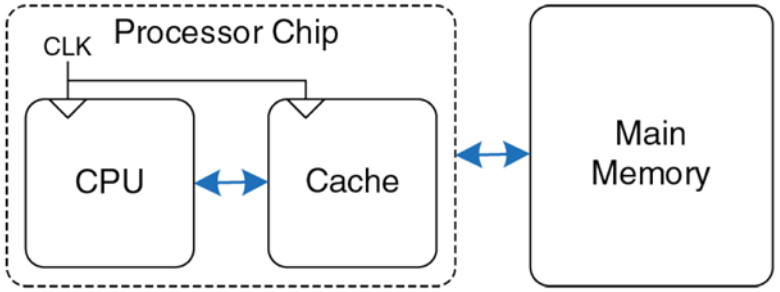
[VA] 0x000024C7

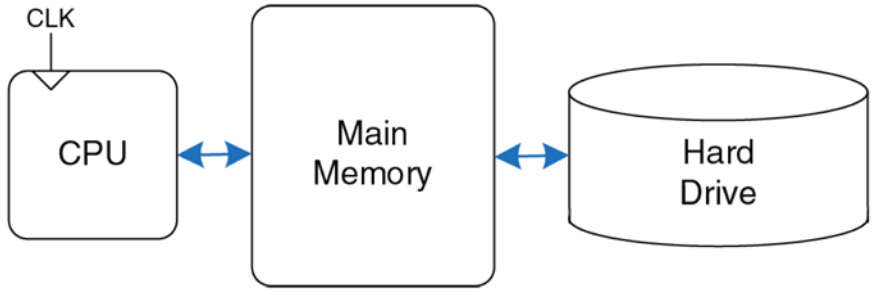


[PA] 0x7FFF4C7



Кэширование vs. Виртуальная память





- Кэширование
  - Запись в кэше
  - Тэг
  - Строка кэша (~32 байта)
  - Доля промахов (1% – 20%)
  - Попадание в кэш (~1 такта)
  - Кэш промах (~100 тактов)
  - Промахи обрабатываются аппаратно
- Виртуальная память (страничная)
  - Страничный кадр
  - Номер виртуальной страницы
  - Страница (~4К байта)
  - Доля страничных сбоев (<0.001%)
  - Страничное попадание (~100 тактов)
  - Страничная ошибка (~5М тактов)
  - Промахи обрабатываются в основном программно

Виртуальная память

- Защита и приватность
  - Каждый процесс использует приватное адресное пространство
- Подкачка страниц
  - Основная память используется как кэш для внешней памяти
  - Позволяет запускать программы большие, чем основная память
  - Скрывает разницу в машинных конфигурациях

- Цена виртуализации – это трансляция каждого обращения к памяти

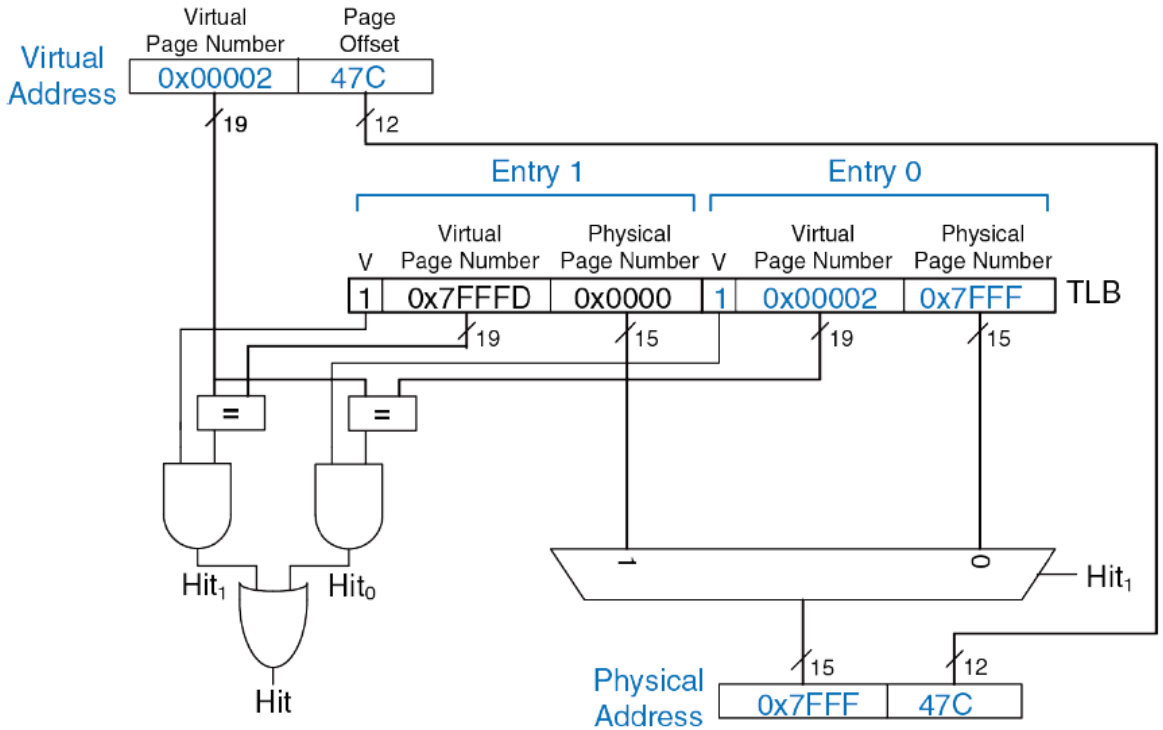
Буфер ассоциативной трансляции (TLB)

- Проблема
  - Трансляция адреса – дорогая операция! Каждое обращение к памяти требует обращения к страничной таблице (происходит 2 обращения к основной памяти)
- Решение
  - Кэширование транслированных страниц в специальной памяти Translation Lookaside Buffer
  - TLB hit → Трансляция происходит за один системный такт
  - TLB miss → Доступ к страничной таблице для обновления TLB

Буфер ассоциативной трансляции (TLB)

[VA] 0x24C7

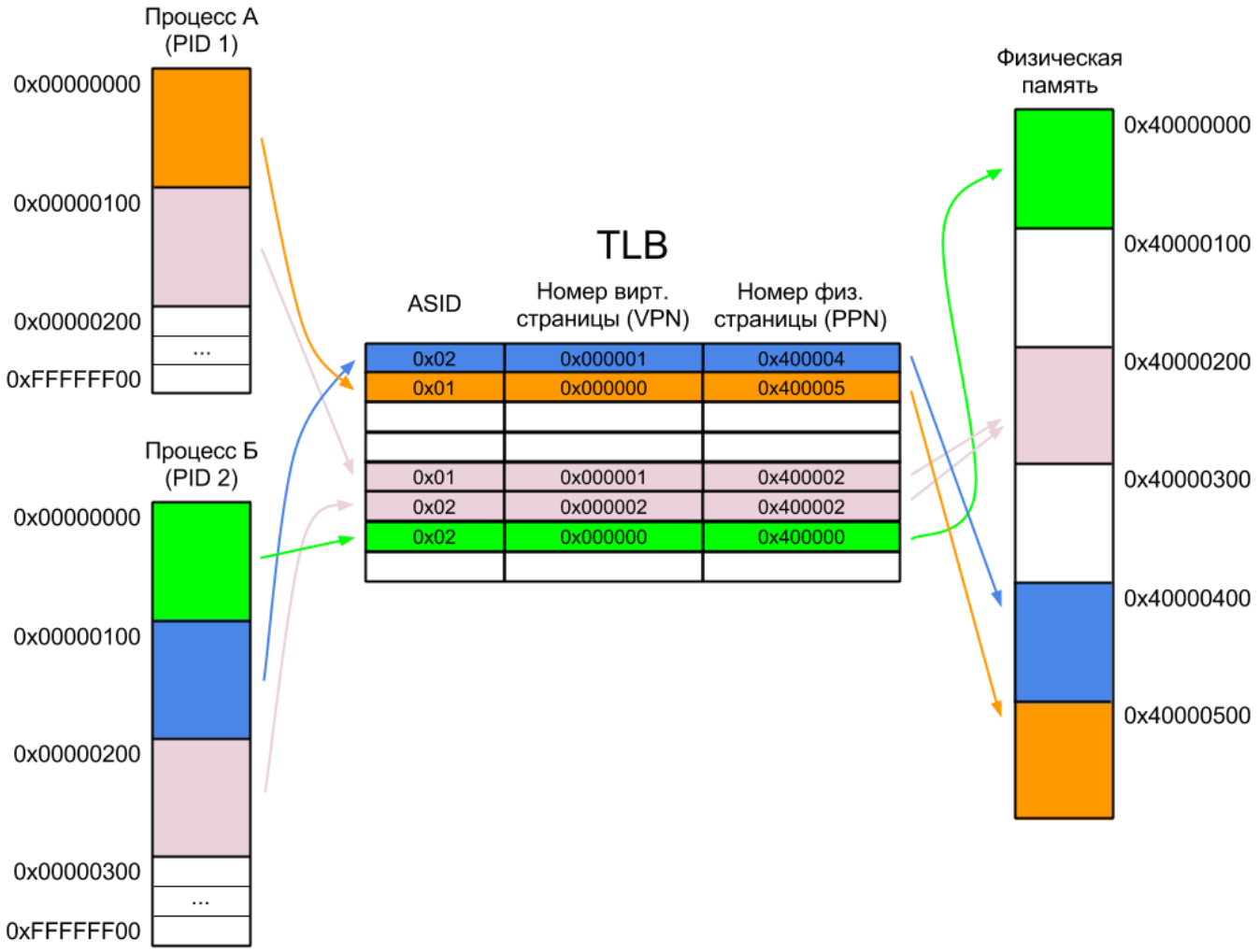
[VA] 0x5FB0



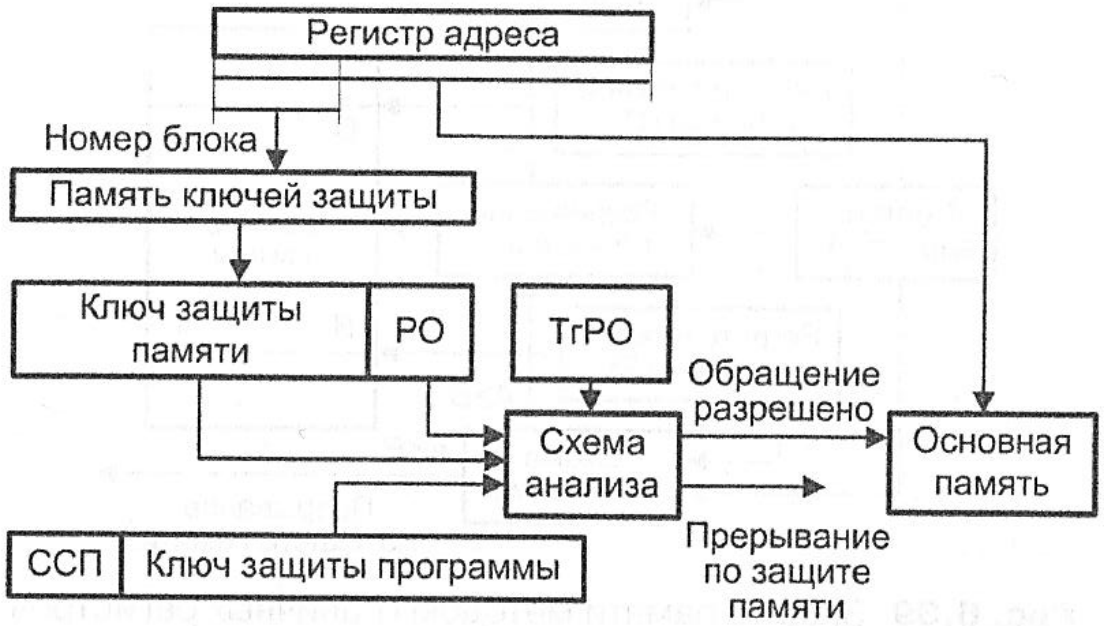
Использование TLB

- TLB имеет 32 – 128 входов, степень ассоциативности 4 – 8
  - Современные процессоры используют иерархический TLB (например, 128-входовый L1 TLB + 2K-входовый L2 TLB)
- Переключение процесса это дорогая операция, потому что нужно очистить TLB
  - Можно включать ID процесса в TLB, чтобы избежать очищения
- Промах TLB вызывает обращение к страничной таблице (основная память). Если страница находится в основной памяти, то происходит преобразование VPN → PPN и обновление TLB. В противном случае произойдет страничный сбой
  - Страничный сбой всегда обрабатывается программно
  - Обход страничной таблицы обычно осуществляется аппаратно с использованием MMU (memory management unit)
    - RISC-V, x86 реализуют страничную таблицу аппаратно

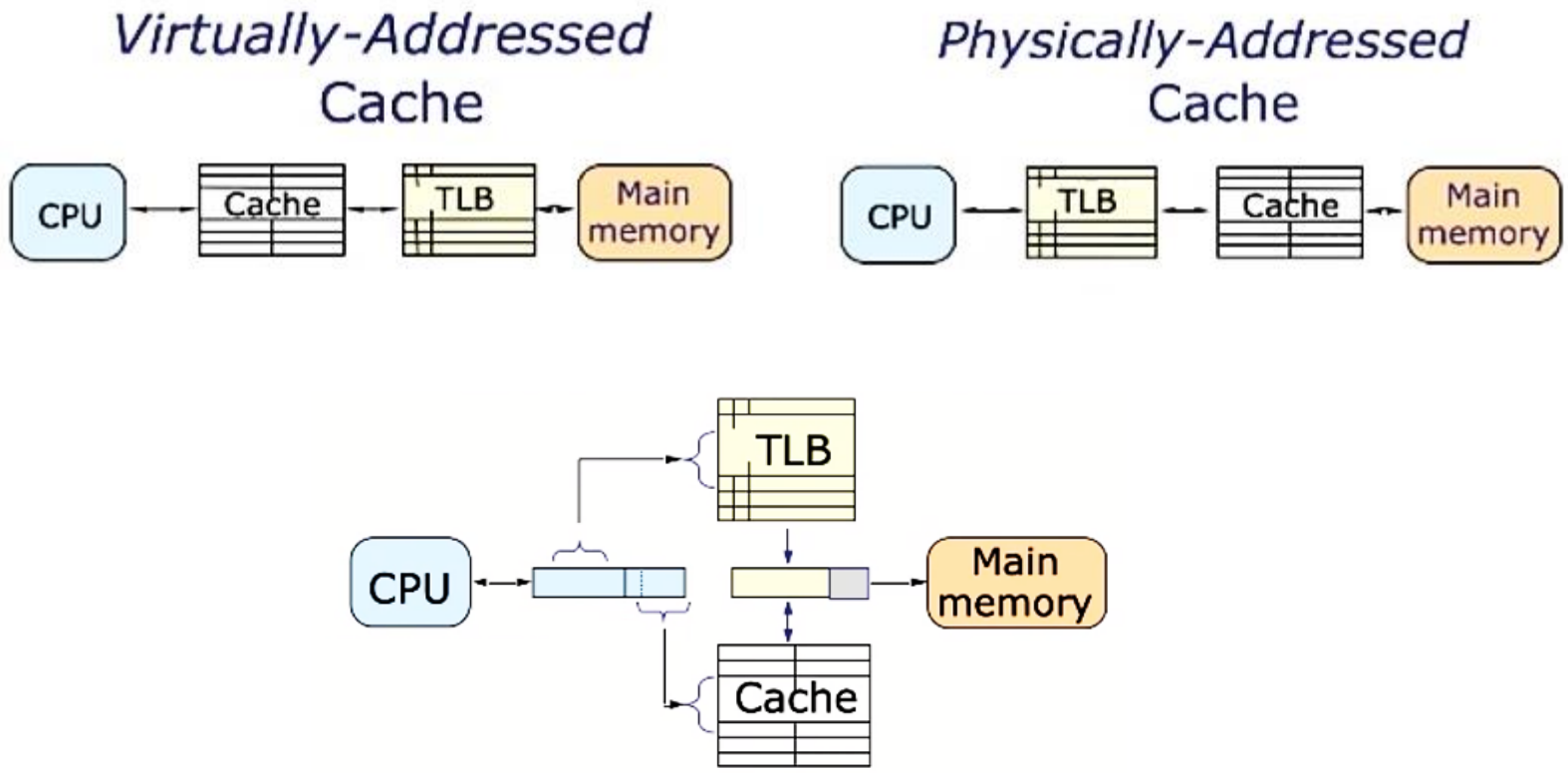
Буфер ассоциативной трансляции (TLB)



Метод ключей защиты



Использование кэш и виртуальной памяти

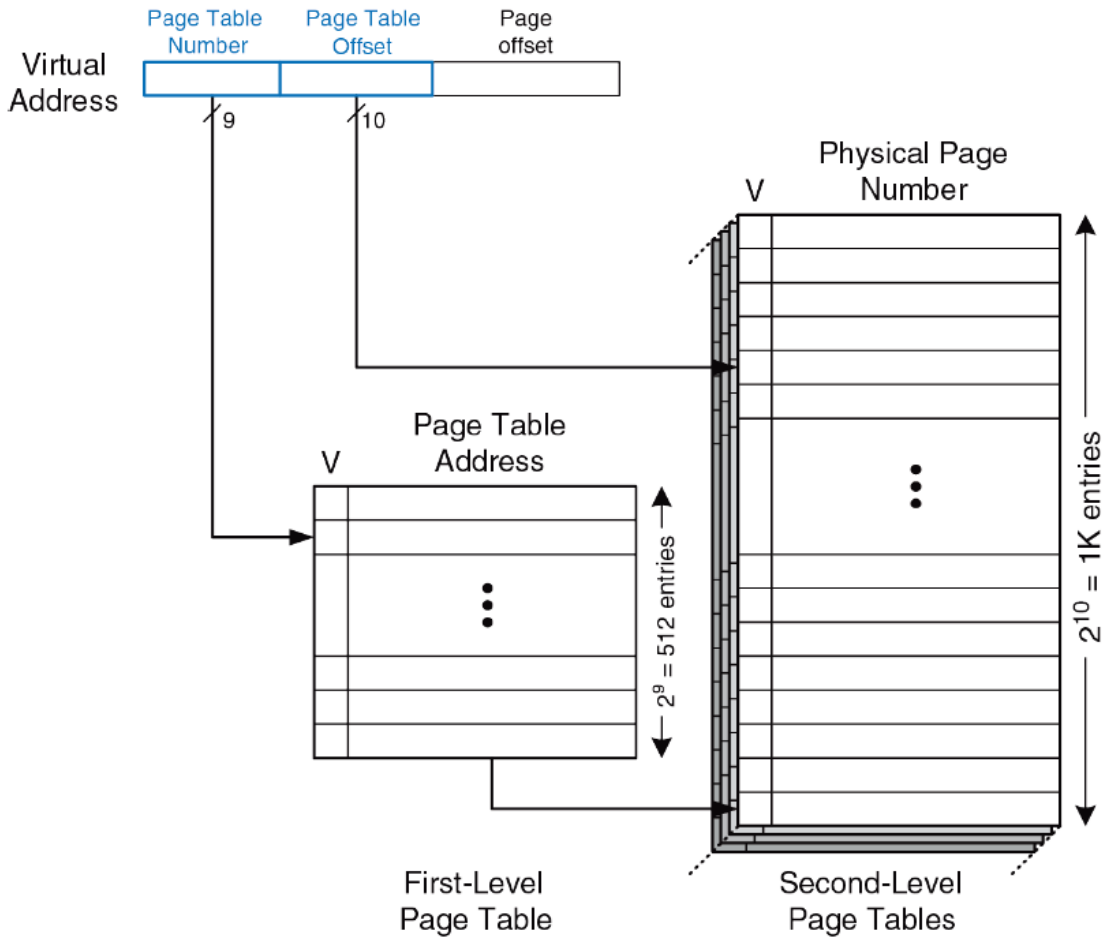


- Используется параллельное обращение к TLB и кэш памяти

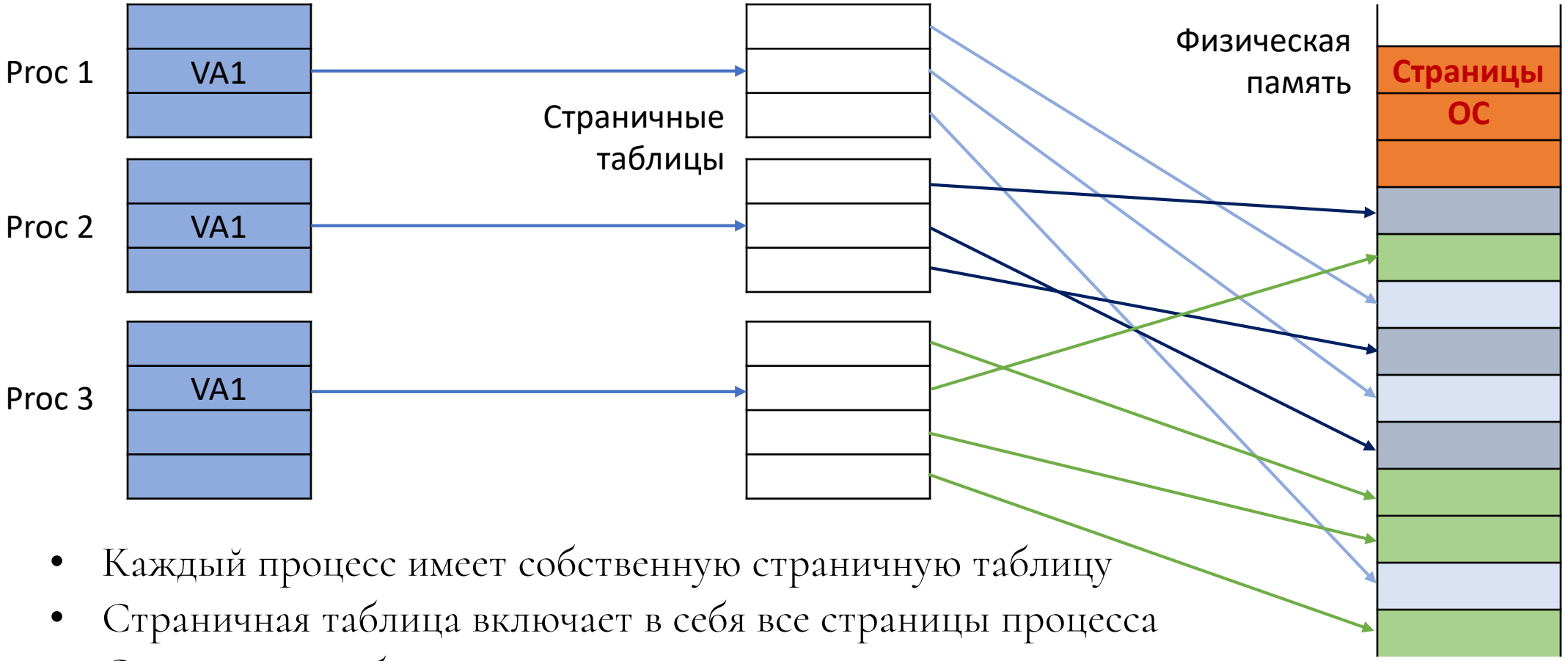
Стратегии замещения страниц

- Подобны стратегиям замещения в кэш-памяти
- Подсистема виртуальной памяти использует стратегию обратной записи (write-back)
  - Для поддержания работы в таблице страниц используется дополнительное поле D (dirty bit)
- Стратегия вытеснения редко используемых страниц (LRU)
  - Для поддержания работы в таблице страниц используется дополнительное поле U (use bit), которое периодически сбрасывается

Многоуровневые таблицы страниц

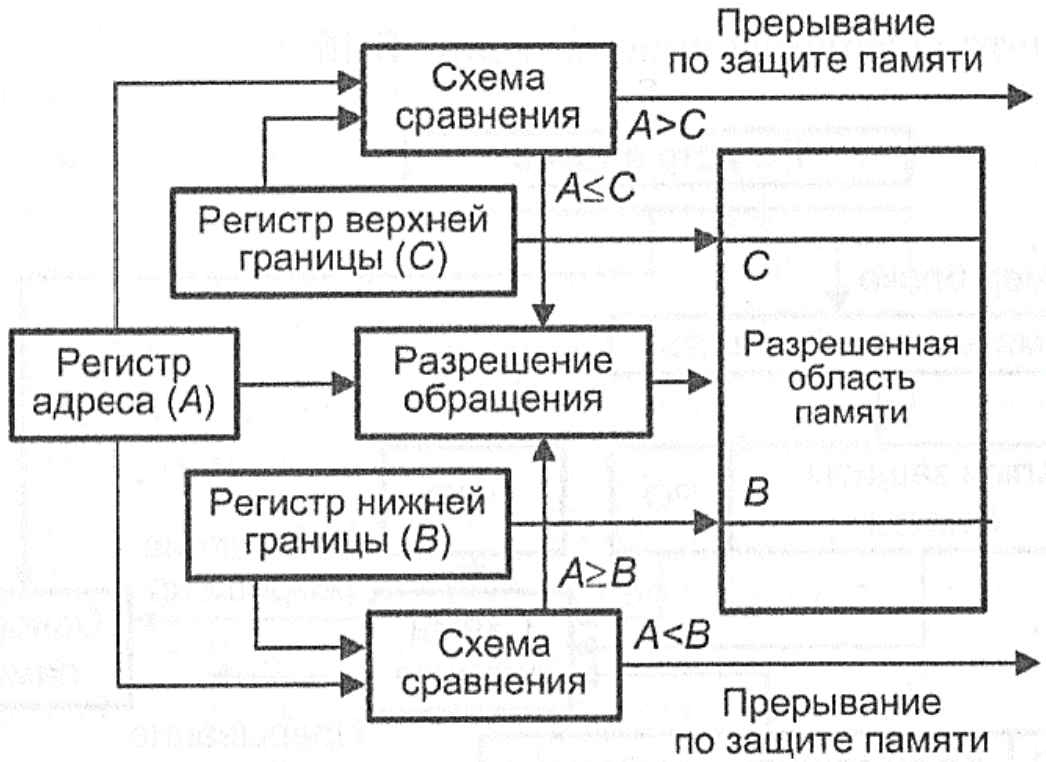


Приватное адресное пространство



- Каждый процесс имеет собственную страничную таблицу
- Страничная таблица включает в себя все страницы процесса
- Страничные таблицы позволяют хранить страницы процесса не непрерывно

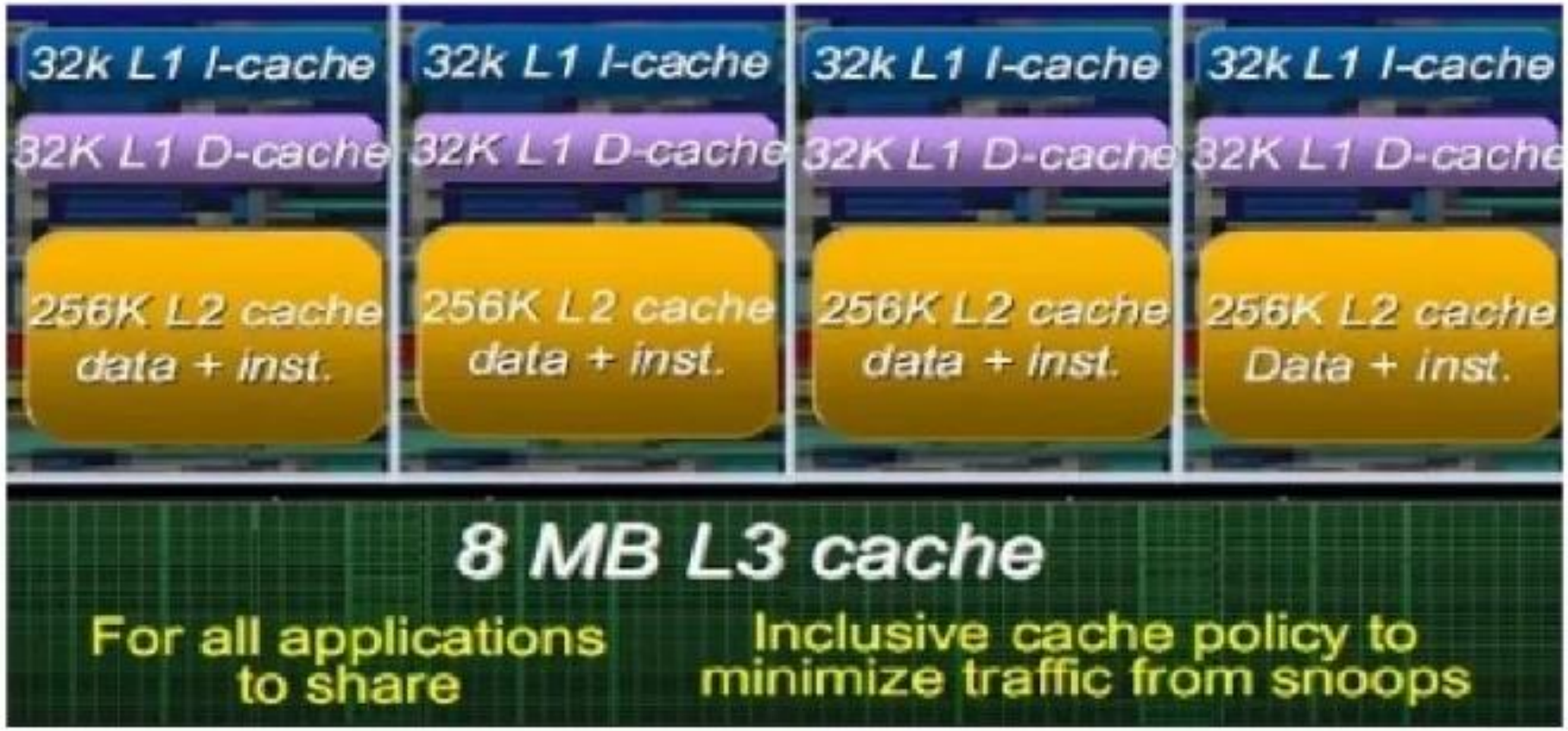
Метод граничных регистров



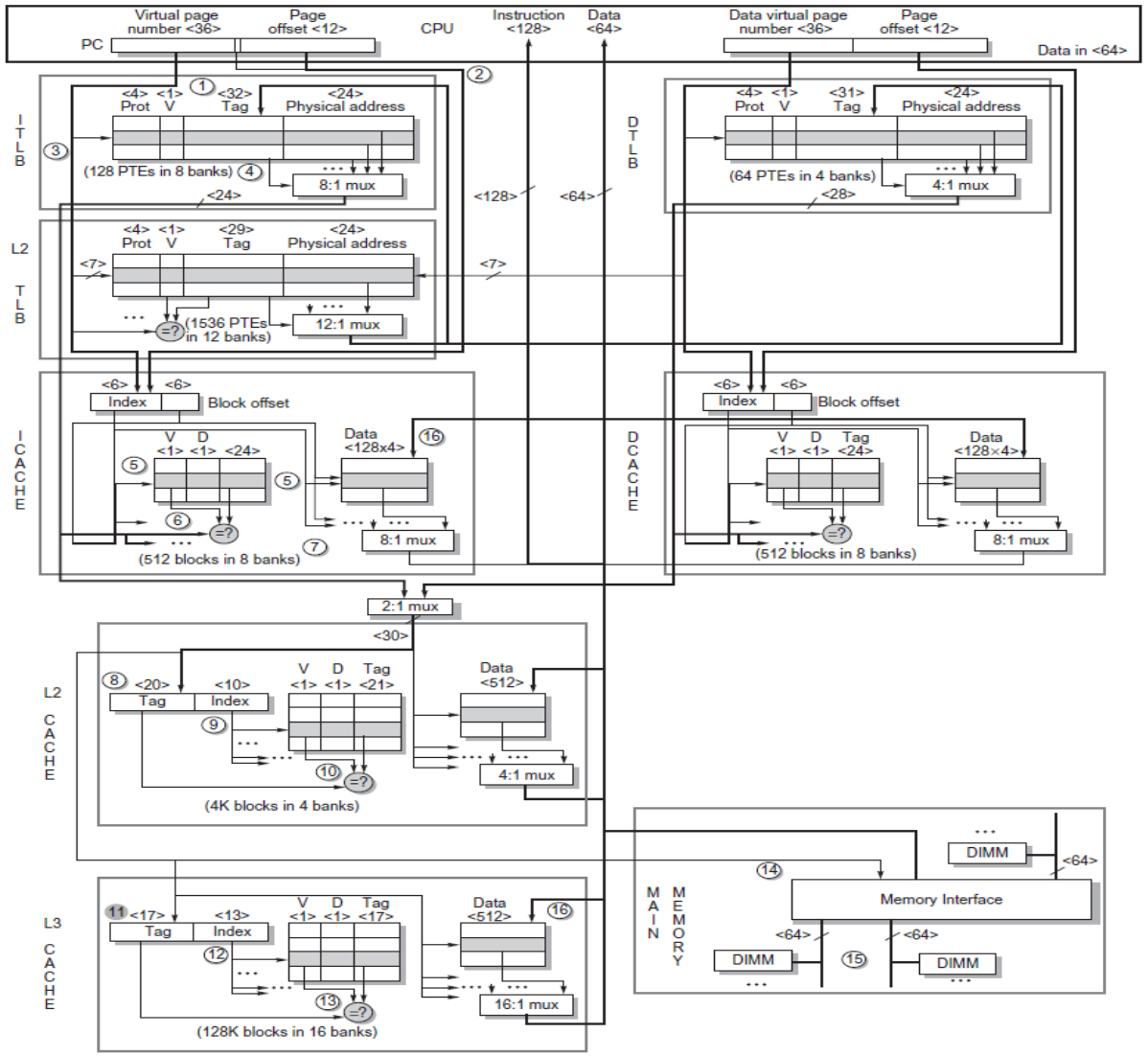
Виртуальная память

- Выгода использования виртуальной памяти
  - Защита и приватность: изолированное адресное пространство для процессов
  - Страничная организация позволяет использовать основную память как кэш для внешней
- Сегментация: адресное пространство каждого процесса представляется непрерывным сегментом в физической памяти
  - Прост в реализации за счет использования граничных регистров
  - Имеет проблемы фрагментации памяти
- Страничная организация: каждый процесс использует адресное пространство реализованное в многоуровневой таблице. Страничная таблица отображает виртуальные адреса на физические
  - Отсутствует фрагментация
  - Страницы могут храниться в основной или внешней памяти
  - Требует обращения к страничной таблице при каждом обращении к памяти
- TLB делает страничную организацию более эффективной

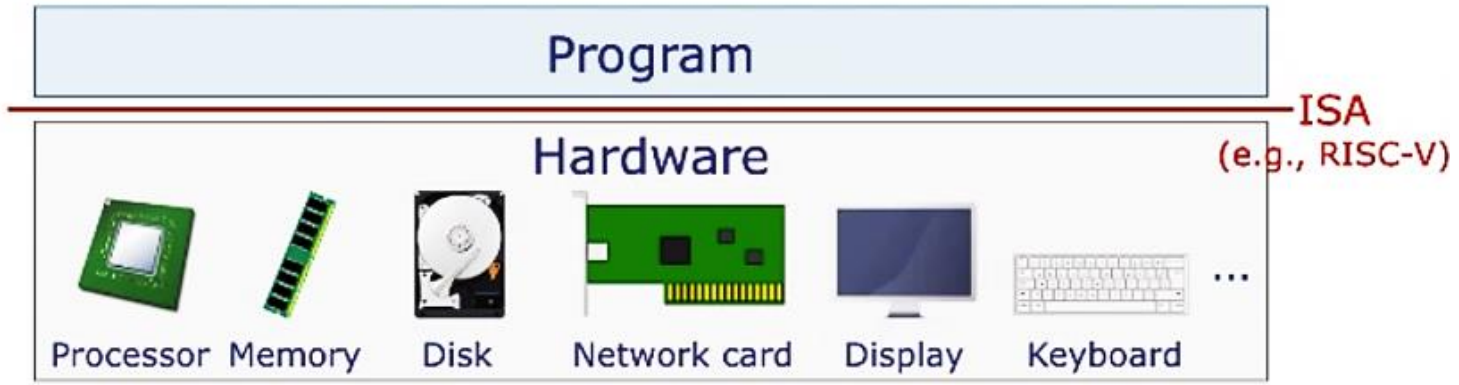
Кеш память Core i7



Иерархия памяти Core i7

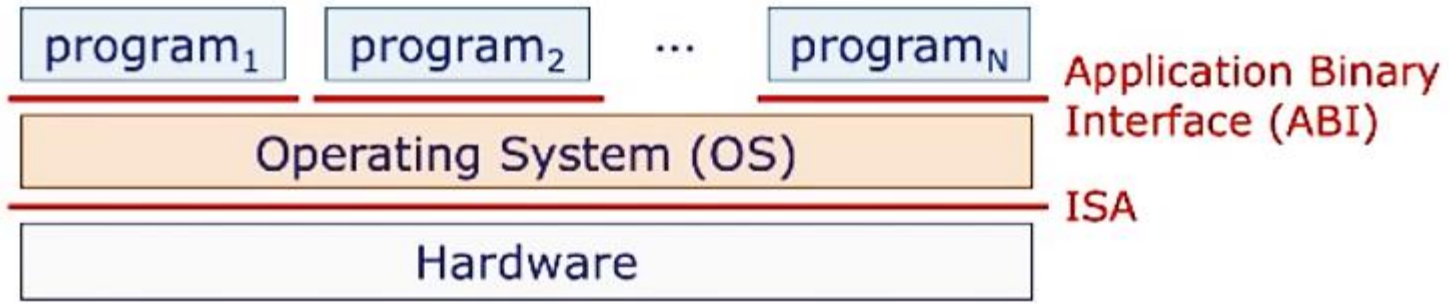


Однопользовательская машина



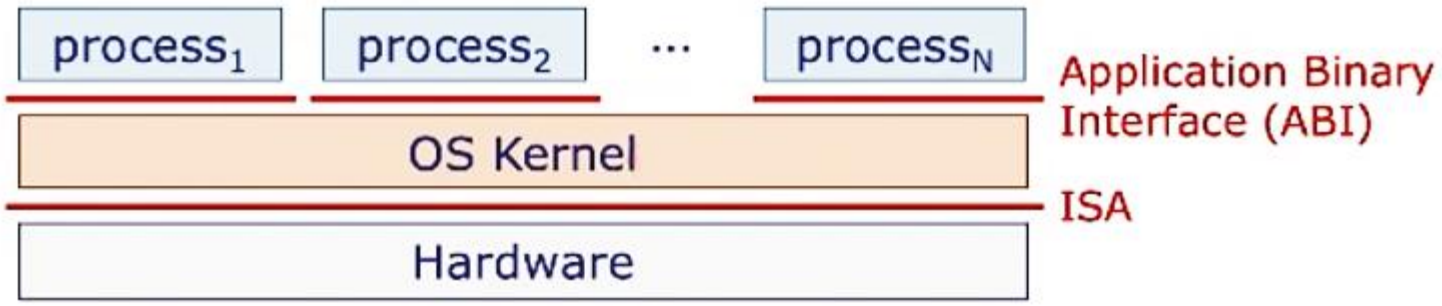
- Аппаратура выполняет одну программу
- Эта программа имеет полный подступ ко всем аппаратным ресурсам машины
- Архитектура системы команд (ISA) является интерфейсом между программным и аппаратным обеспечением
- Большинство систем работают не так

Операционная система



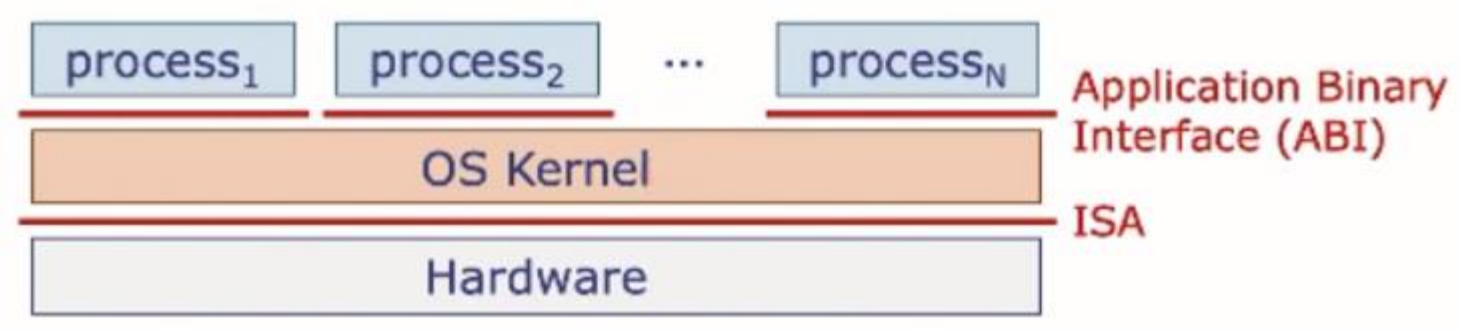
- Несколько запущенных программ используют одну машину
- Каждая запущенная программа не имеет прямого доступа к аппаратным ресурсам
- Вместо этого операционная система контролирует программы и то, как они разделяют аппаратные ресурсы
  - Только операционная система имеет неограниченный доступ к аппаратным ресурсам
- Программный интерфейс (ABI) – это интерфейс между операционной системой и программой

Процесс vs. программа



- Программа – это набор инструкций (только ее код)
- Процесс – это экземпляр программы, которая выполняется
  - Состоит из кода программы и контекста (регистры, содержимое памяти и другие ресурсы)
- Ядро операционной системы (OS Kernel) это привилегированный процесс

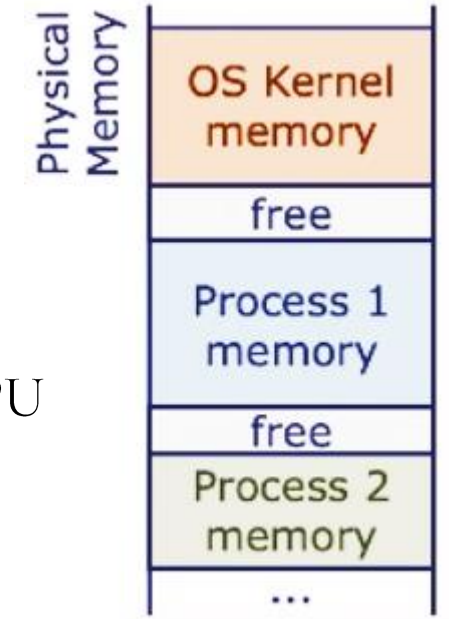
Цели операционной системы



- Защита и приватность: процессы не могут получить доступ к любым данным, только ко своим
- Абстракция: операционная система скрывает детали реализации аппаратного обеспечения
- Управление ресурсами: операционная система контролирует доступ процессов к ресурсам (CPU, память, внешняя память...)

Операционная система

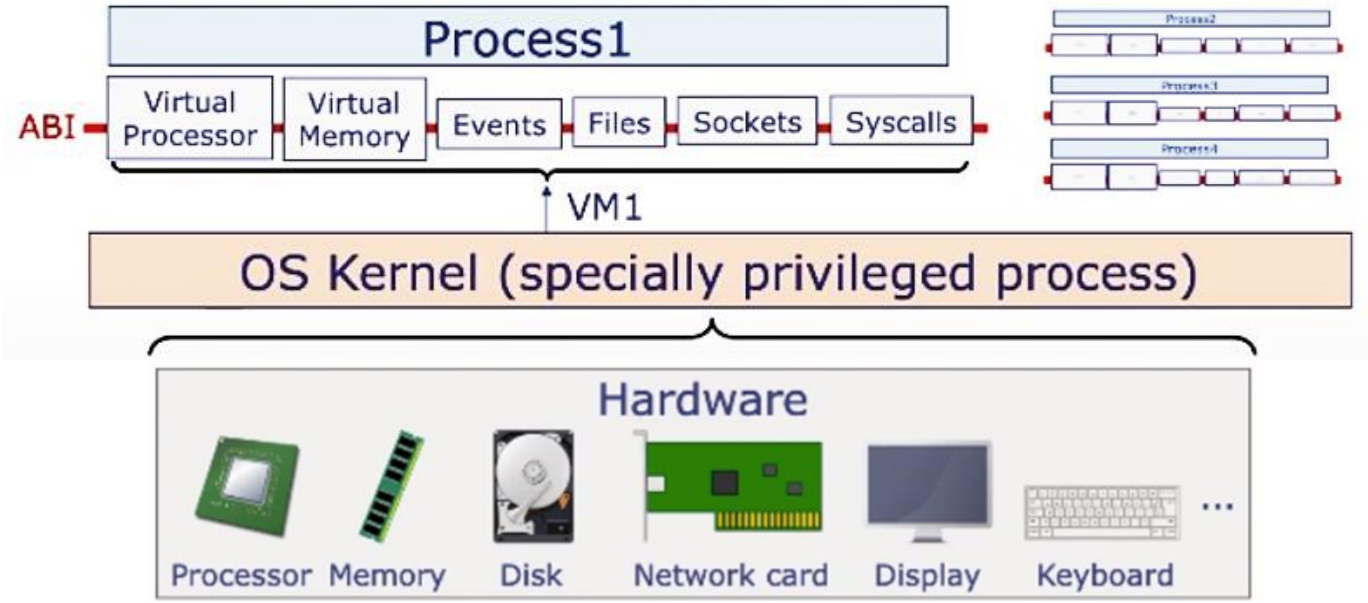
- Ядро операционной системы предоставляет приватное адресное пространство каждому процессу
  - Каждому процессу операционной системой выделяется пространство физической памяти
  - Процессы не могут получить доступ к «чужой» памяти
- Ядро операционной системы занимается планированием процессов для CPU
  - Каждый процесс получает часть времени CPU
  - Процесс не может использовать больше времени, чем ему выделяется



- Ядро операционной системы позволяет процессам вызывать системные службы (доступ к файловой системе или сетевой карте), это называется системным вызовом (system calls)

Виртуальные машины

- Операционная система предоставляет виртуальную машину каждому процессу
  - Каждый процесс верит, что он работает на собственной машине…
  - …но эта машина не имеет физического аппаратного обеспечения
- Виртуальная машина это эмуляция компьютерной системы
  - Важная концепция используемая не только операционными системами



Виртуальные машины на практике

- RISC-V процесс
  - RISC-V ISA
- RISC-V эмулятор
  - JavaScript
- Браузер
  - ABI OS
- Операционная система
  - x86
- Аппаратное обеспечение

Поддержка VM

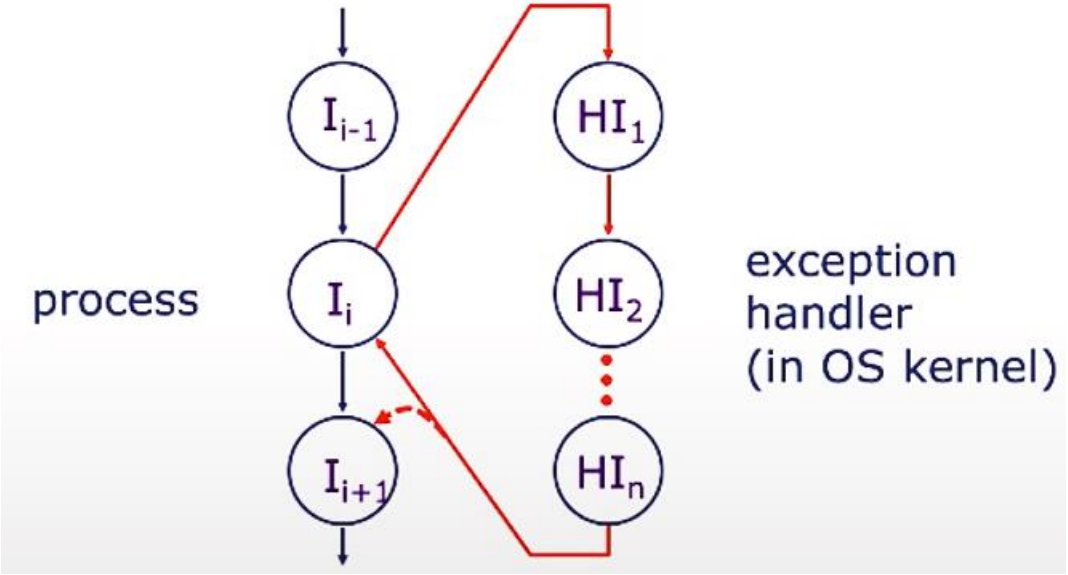
- Виртуальные машины могут полностью реализованы в программном обеспечении, но это сильно влияет на производительность
  - Например, программы для Python в 10-100 раз медленнее компилируемых программ из-за накладных расходов на интерпретацию
- Если мы хотим поддержку операционных систем с минимальными накладными расходами, значит необходимо аппаратная поддержка виртуальных машин

Поддержка OS в ISA

- Две модели исполнения программ: пользовательский (user) и привилегированный (supervisor)
  - Ядро ОС запускается в привилегированном режиме
  - Все остальные процессы запускаются в пользовательском режиме
- Привилегированные инструкции и регистры доступны только в режиме supervisor
- Прерывания и исключения для безопасной передачи управления между user и supervisor режимами
- Виртуальная память обеспечивает приватное адресное пространство и абстракцию ресурса памяти для машины

Прерывания и исключения

- Исключение – событие которое требует обработки операционной системой (неподдерживаемая инструкция, системный вызов, деление на ноль...)



- Прерывания – асинхронные события, требующие обработки операционной системой (таймер, мышка, клавиатура...)

Обработка исключений

- Когда происходит исключение, процессор:
  1. Останавливает выполнение текущего процесса на i-ой инструкции, заканчивая все инструкции перед i
  2. Сохраняет значение PC i-ой инструкции и причину исключения в специальном (привилегированном) регистре
  3. Переключается в режим supervisor, отключает прерывания и передает контроль подпрограмме обработчика прерывания (исключения)
- После того как ОС обработает исключение, управление будет передано процессу, начиная с i-ой инструкции
  - Исключение прозрачно для процесса (он не знает, что оно происходит)
- Если исключение вызвано неразрешенной операцией программы (например, нарушены границы памяти), то ОС прекращает процесс

Типичные системные вызовы

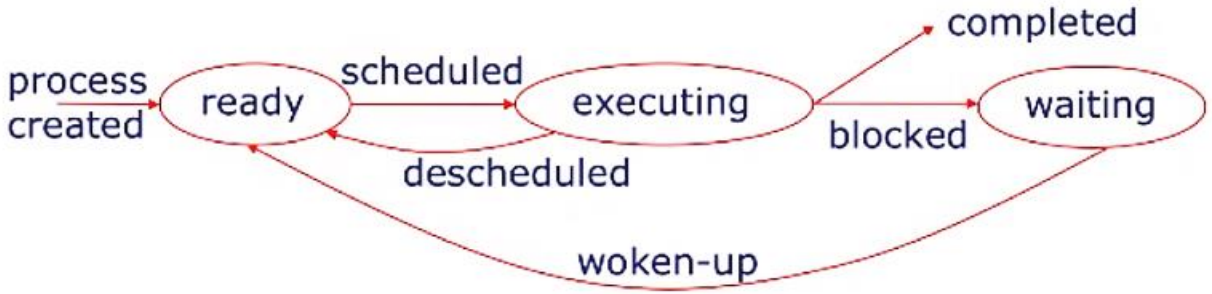
- Доступ к файлам
- Использование сетевого подключения
- Управление памятью
- Получение информации о системе или процессе
- Ожидание некоторого события
- Создание и прерывание других процессов
- и так далее, и тому подобное...

- Программы редко используют системные вызовы напрямую, вместо этого они пользуются библиотечными подпрограммами
- Некоторые системные вызовы могут блокировать процесс

Системные вызовы RISC-V

- Инструкция ecall вызывает исключение устанавливая mcause регистр в конкретное значение
- ABI определяет как процесс и ядро будут обмениваться аргументами и результатами
- Аналогичное соглашение используется при вызове подпрограмм
  - Номер системного вызова помещается в регистр a7
  - Остальные аргументы размещаются в регистрах a0 – a6
  - Результат размещается в a0 – a1 (или в основной памяти)
  - Все регистры являются сохраняемыми

Жизненный цикл процесса



- Операционная система содержит список всех процессов и их статусов (ready, executing, waiting)
  - Процесс запланирован на выполнение в течение указанного количества процессорного времени или до завершения
  - Если системный вызов не может быть удовлетворен немедленно (например, требуется доступ к диску), то процесс блокируется и переходит в состояние ожидания
  - Когда запрос ожидающего процесса удовлетворен, процесс переходит в режим готовности