Лекция 17
Изоляция процессов

Изоляция процессов

- Операционная система изолирует процессы друг от друга и от аппаратуры компьютера
- Виртуальная память – адресное пространство каждого процесса изолировано
- Процесс работает в **пользовательском режиме** (user mode) и не может выполнять “чувствительные” инструкции (настройка MMU, работа с внешними устройствами...)

Ядро ОС

- Ключевая компонента ОС
- Работает все время работы компьютера от загрузки до выключения
- Работает в **привилегированном режиме** (privileged mode или kernel mode)
- Управляет внешними устройствами, распределяет ресурсы между процессами

Системный вызов

- Процесс в пользовательском режиме не может выполнить ввода-вывода (нет прав)
- Для выполнения ввода-вывода процесс вызывает ядро ОС
- Ядро ОС от имени и с проверкой прав процесса выполняет запрошенную операцию
- Управление возвращается в процесс, он продолжает работу в режиме пользователя
- Вызов ядра – **системный вызов**

Системный вызов

- Все системные вызовы ядра занумерованы
- На Linux:
  - #include <asm/unistd_32.h> - для x86
  - #include <asm/unistd_64.h> - для x64
- На Linux/x86
  - Номер системного вызова передается в %eax
  - Параметры вызова в %ebx, %ecx, %edx, %esi, %edi, %ebp
  - Системный вызов: int $0x80
  - Результат возвращается в %eax

Системные вызовы

- Системные вызовы задокументированы в терминах языка C:
- Документация: man 2 SYSCALL, например
  man 2 write
- Исключение: системный вызов exit задокументирован как _exit
- Стандартная библиотека libc содержит вспомогательные функции (“мосты”)
  - Удовлетворяют стандартным соглашениям о вызовах
  - Обеспечивают подготовку параметров и выполнение системного вызова
- Такие функции стандартной библиотеки для удобства тоже называются системными вызовами

Некоторые системные вызовы

- Чтение
  ssize_t read(int fd, void *buf, size_t count);
  пока для наших целей fd == 0 – стандартный поток ввода
- Запись
  ssize_t write(int fd, const void *buf, size_t cnt);
  fd == 1 – запись на стандартный поток вывода
- Завершение работы:
  void _exit(int status);

Системные вызовы

- Системные вызовы предоставляют минимально необходимый интерфейс, те операции, которые невозможно или неэффективно выполнять без вызова ядра
- Предоставление удобного интерфейса – задача библиотек, работающих в пользовательском режиме
- Библиотечные функции для выполнения ввода-вывода используют системные вызовы
- Например: write и printf или std::cout

Виртуальная адресация

- Для многопроцессной обработки требуется защита памяти: процесс не должен иметь неавторизованный доступ к памяти других процессов и ядра
- Адреса ячеек памяти данных и программы, используемые в процессе, не обязаны совпадать с адресами в физической памяти (ОЗУ)
- Адреса ячеек памяти для процесса — **виртуальные адреса**
- Адреса ячеек памяти в оперативной памяти — **физические адреса**

Виртуальная адресация (память)

- Программно-аппаратный механизм трансляции виртуальных адресов в физические
- Аппаратная часть — отображение виртуальных адресов в физические в «обычной» ситуации — должно быть очень быстрой, так как необходимо для выполнения каждой инструкции
- Программная часть — подготовка отображения к работе, обработка исключительных ситуаций

Модели виртуальной адресации

- Модель база+смещение
  - Два регистра для процесса: регистр базы (B), регистр размера (Z)
  - Пусть V — виртуальный адрес (беззнаковое значение), если V >= Z — ошибка доступа к памяти, иначе
  - P — физический адрес, P = B + V

Сегментная адресация

- Каждый процесс состоит из нескольких сегментов: сегмент кода, сегмент стека, сегмент данных1, сегмент данных2
- Для каждого сегмента хранятся свои базовый адрес и размер
- У каждого сегмента свои права доступа, например:
  - Код: чтение + выполнение
  - Стек: чтение + запись
- Сегмент может отсутствовать в оперативной памяти и подгружаться по требованию

Страничная адресация

- Все пространство виртуальных адресов разбивается на страницы **равного размера**
- Каждая страница виртуальной памяти отображается на физическую память независимо от других
- Каждая страница имеет права доступа независимо от других страниц
- Страница может быть отмечена как неотображенная или отсутствующая в памяти
- При невозможности аппаратно отобразить виртуальную страницу в физическую — Page Fault

Отображение страниц

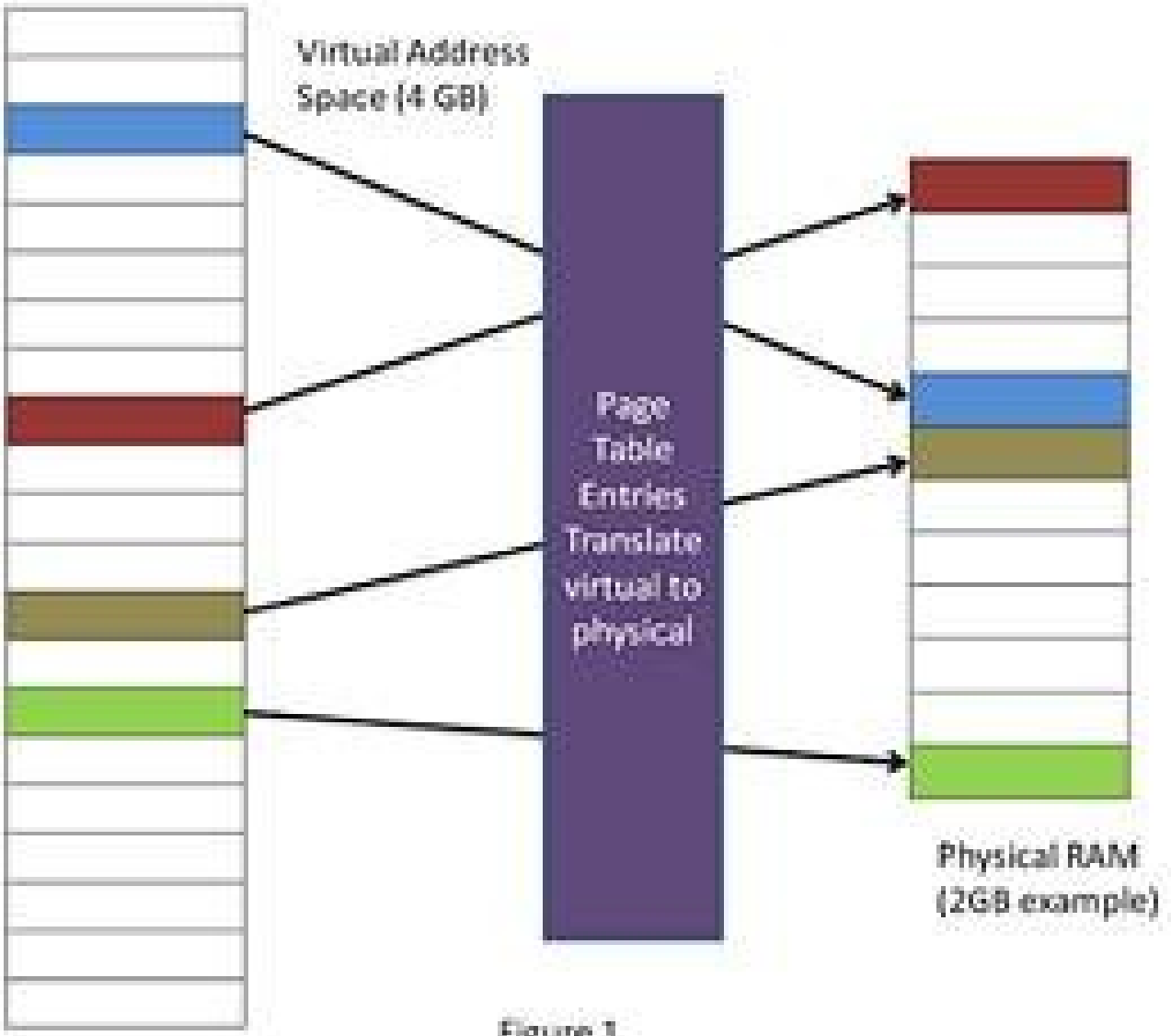
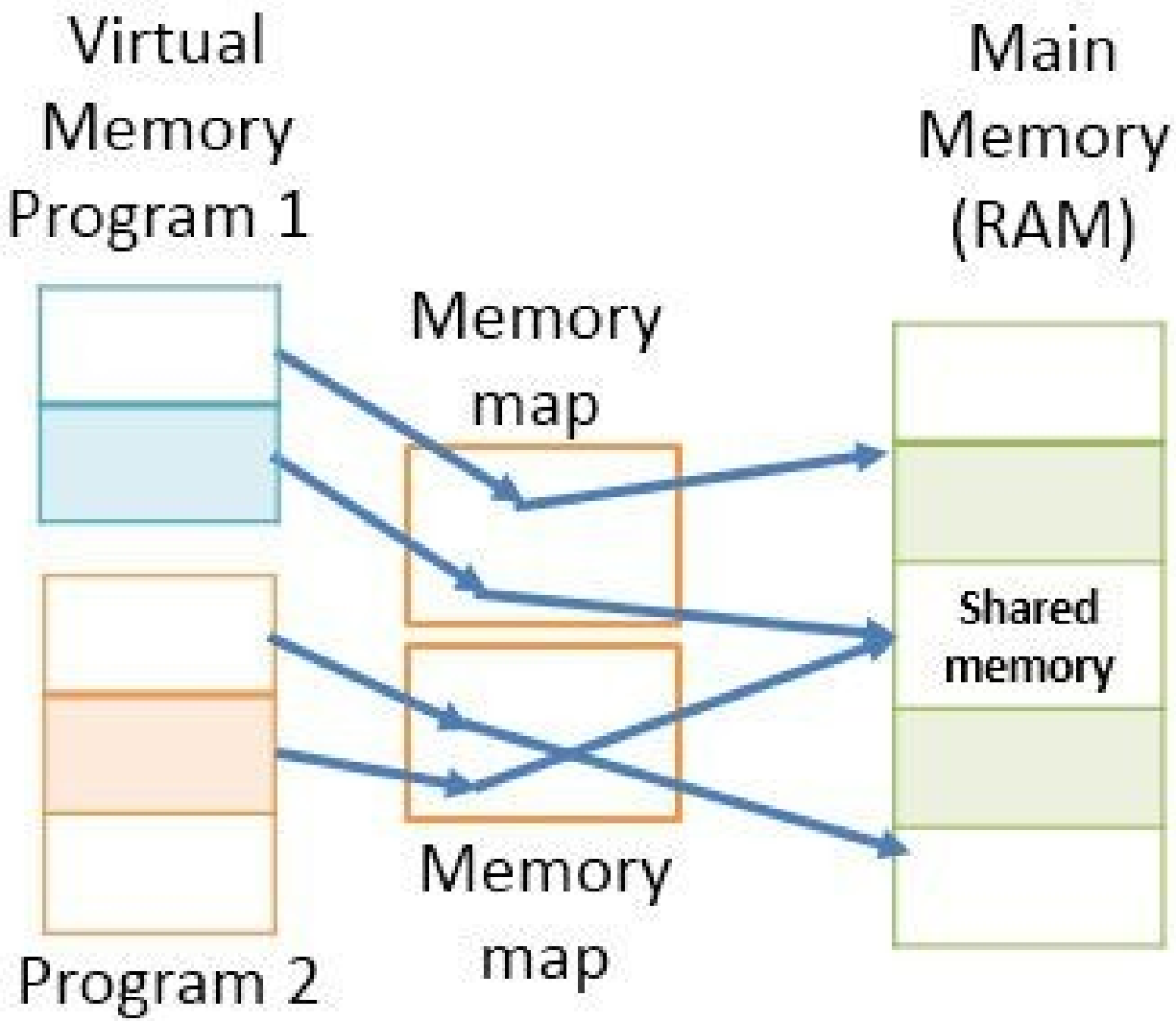
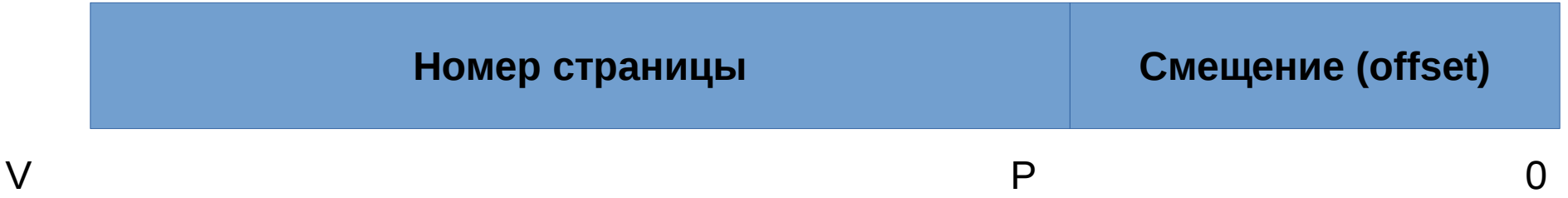


Figure 1

Разделяемые страницы



Виртуальный адрес



- V — количество бит виртуального адреса
- P — количество бит на смещение в страницу
- (V - P) — количество бит на номер страницы
- Для x86: V = 32, P = 12, V-P = 20
  - Виртуальное адресное пространство 4GiB
  - Размер страницы — 4096 байт (4KiB)
  - 2^20 (~1 Mi виртуальных страниц)

Двухуровневая таблица страниц (x86)

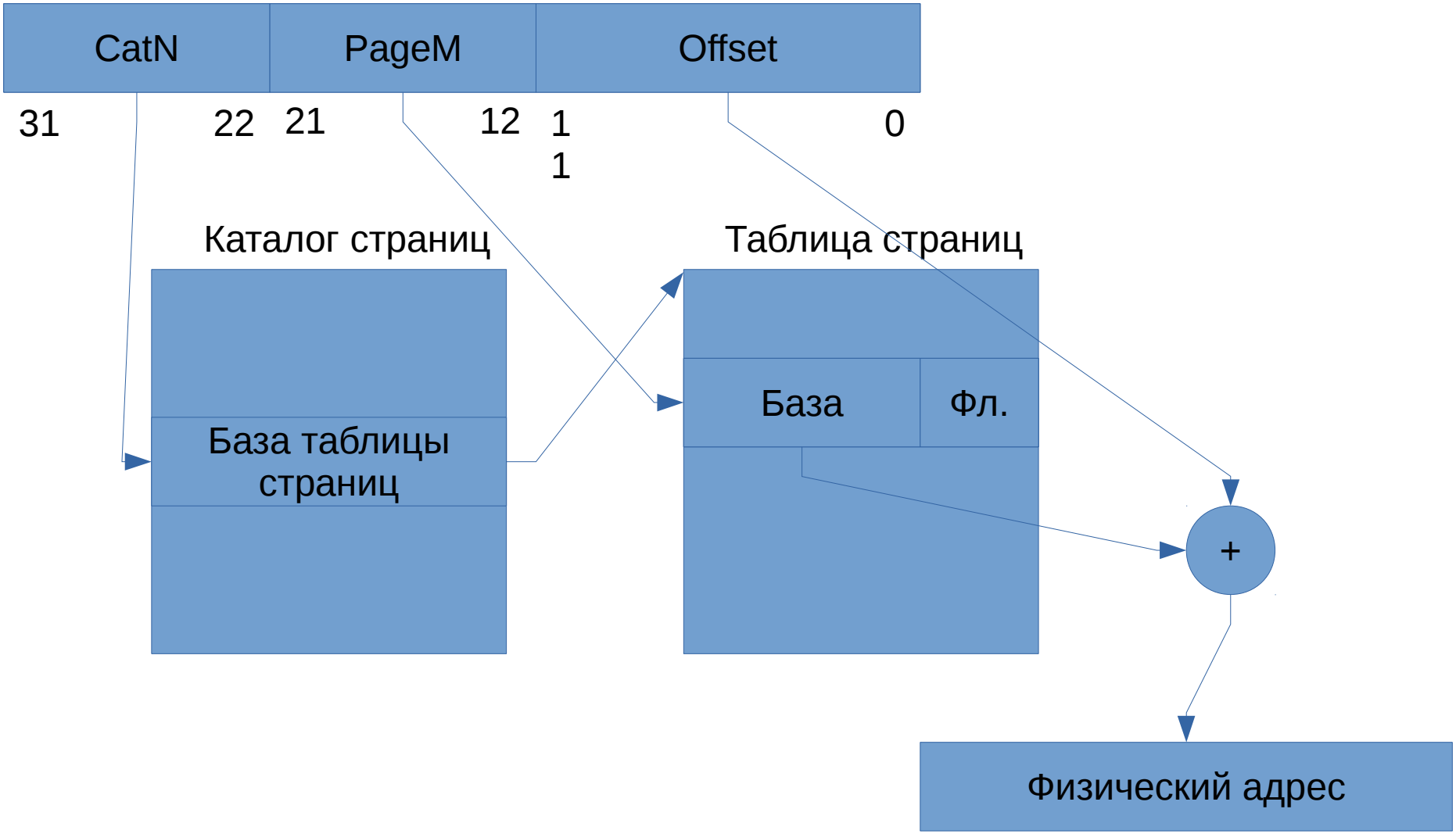


Таблица страниц

- Регистр процессора CR3 указывает на начало каталога страниц
- X86 — двухуровневая таблица страниц, размер страницы — 4KiB, в каталоге страниц 1024 записи, в каждой таблице страниц 1024 записи, одна запись — 4 байта
- X64 — четырехуровневая таблица страниц, размер страницы — 4KiB, в таблице каждого уровня 512 записей, одна запись — 8 байт.

Элемент таблицы страниц (x86)

- P — страница присутствует в ОЗУ
- R — право на запись в страницу
- U — доступна из user-space
- C — кеширование страницы запрещено
- W — разрешена сквозная (write-through) запись
- A — к странице было обращение
- D — (dirty) страница была модифицирована
- G — страница глобальная

Трансляция адресов x86

```
#define PAGE_SIZE  4096
#define TABLE_SIZE 1024
unsigned translate(unsigned va)
{
  unsigned *catalog = CR3;
  unsigned *table = catalog[va >> 22] & -PAGE_SIZE;
  unsigned phys = table[(va >> 12) & (TABLE_SIZE - 1)]
& -PAGE_SIZE;
  return phys + (va & (PAGE_SIZE - 1));
}
```

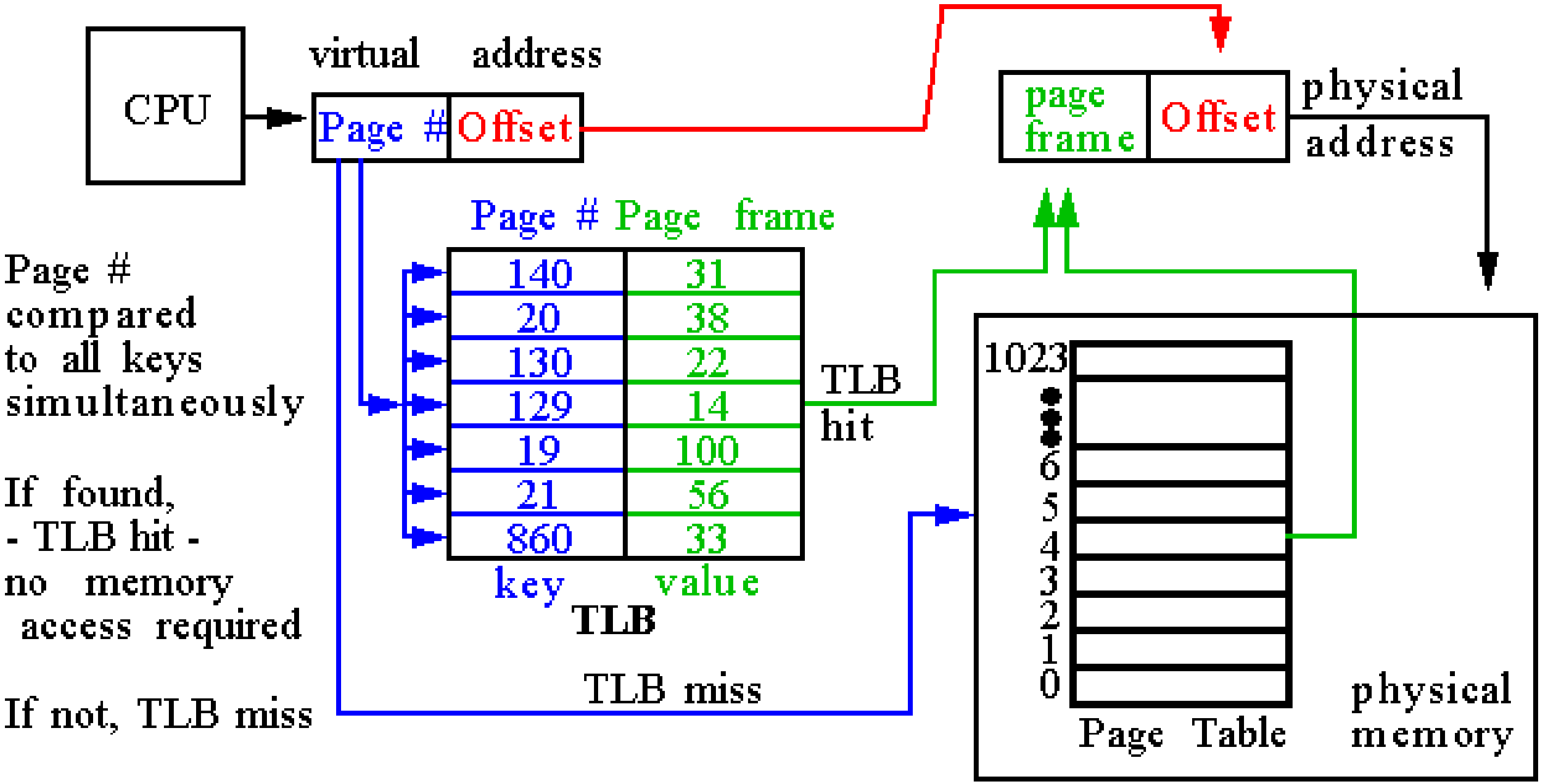
Доступ к странице

- Если страница отсутствует в ОЗУ (P == 0), обращение к странице → Page Fault
- Если в user-space и U == 0 → Page Fault
- Если записываем в страницу и R == 0 → Page Fault
- Устанавливаем флаг «accessed» (A = 1)
- Если записываем, устанавливаем флаг «dirty» (D = 1)

TLB (Translation Lookaside Buffer)

- Двухуровневая таблица страниц может потребовать 2 вспомогательных обращения к памяти (а 4-уровневая – 4!!!)
- TLB — кэш-память для отображения виртуального адреса в физический
- TLB может быть многоуровневым и разделенным: для Intel Nehalem:
  - 64 записи в L1 DTLB
  - 128 записей в L1 ITLB
  - 512 записей в L2 TLB

TLB


CPU
virtual address
Page # Offset
page frame
Offset
physical address
Page # Page frame
140 31
20 38
130 22
129 14
19 100
21 56
860 33
key value
TLB
TLB hit
Page # compared to all keys simultaneously
If found, - TLB hit - no memory access required
If not, TLB miss
TLB miss
1023
6
5
4
3
2
1
0
Page Table
physical memory

PageFault

- Исключение PageFault не обязательно ошибка в программе
- Допустимые ситуации:
  - Страница данных откачана в swap (P == 0)
  - Страница кода не загружена из файла (P == 0)
  - Запись в страницу созданную для copy-on-write (R == 0)
- Обработчик исключения определяет причину PageFault. Если PageFault произошел из-за ошибки, ошибка передается в программу

3G/1G virtual memory split

- Верхний 1GiB адресного пространства процесса содержит страницы, отображенные в режиме U == 0
- Страницы недоступны из user-space, но как только процесс переключается в kernel-space они становятся доступны
- Код и данные ядра отображаются в эту часть адресного пространства
- При переключении контекста (при входе в системный вызов) нет накладных расходов на перезагрузку таблицы страниц
- Из кода ядра непосредственно доступна память процесса (copy_from_user, copy_to_user)

Итог: зачем нужна страничная виртуальная память

- Каждый процесс (выполняемая программа) имеет свое виртуальное адресное пространство – упрощение управлением памятью на уровне процесса (стек, куча, нити)
- Права доступа к памяти могут гибко настраиваться (read, write, execute)
- Процессы изолированы друг от друга
- Но несколько процессов могут разделять (использовать) одну и ту же страницу физического ОЗУ (shared pages)
- Программа (и вообще адресное пространство отдельного процесса) не обязана располагаться последовательно в физической памяти
- Виртуальной памяти может быть больше физической

Что почитать

- http://rus-linux.net/lib.php?name=/MyLDP/hard/memory/memory.html
-