Управление памятью

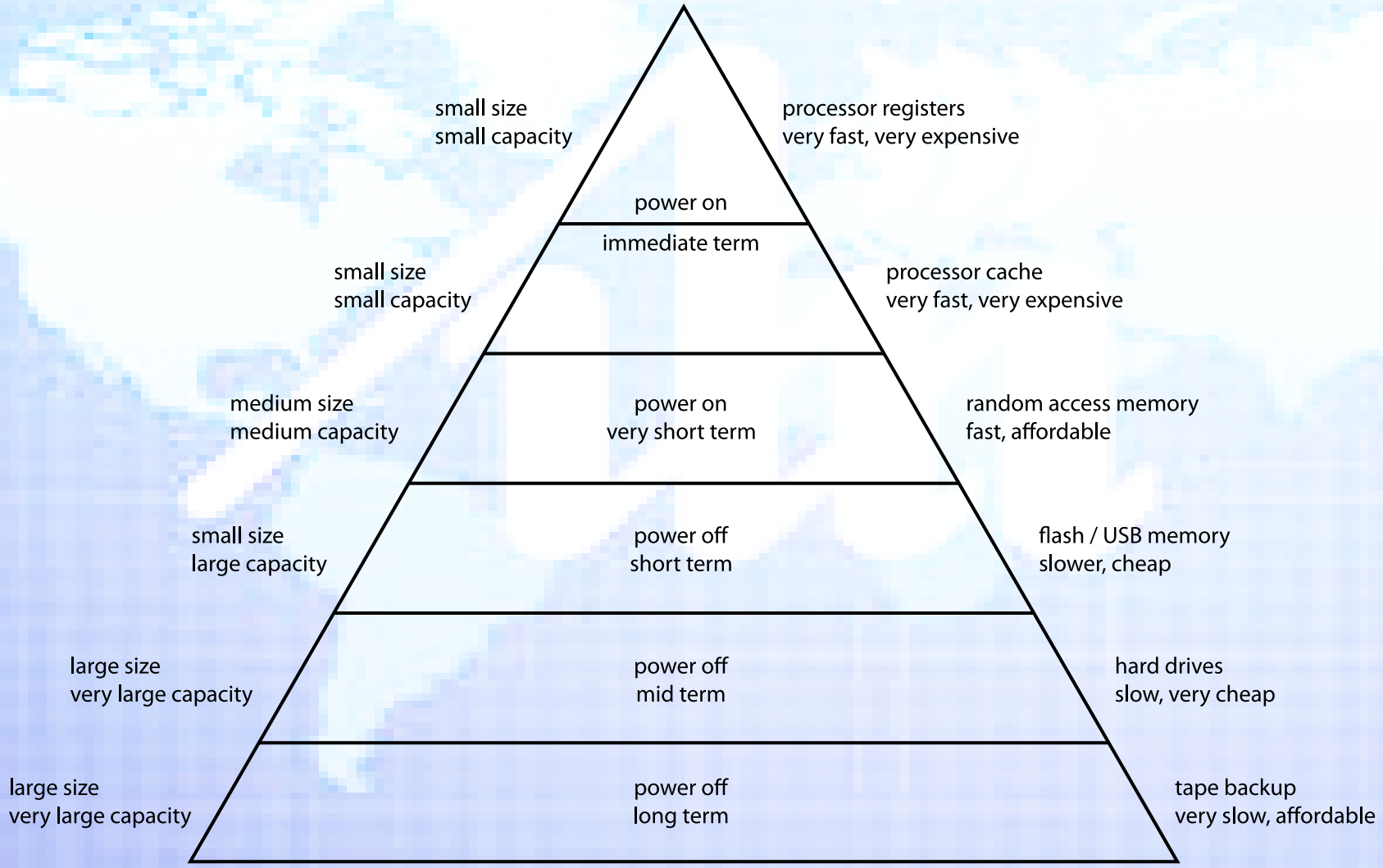
Евгений Иванович Клименков

osisp2019@gmail.com

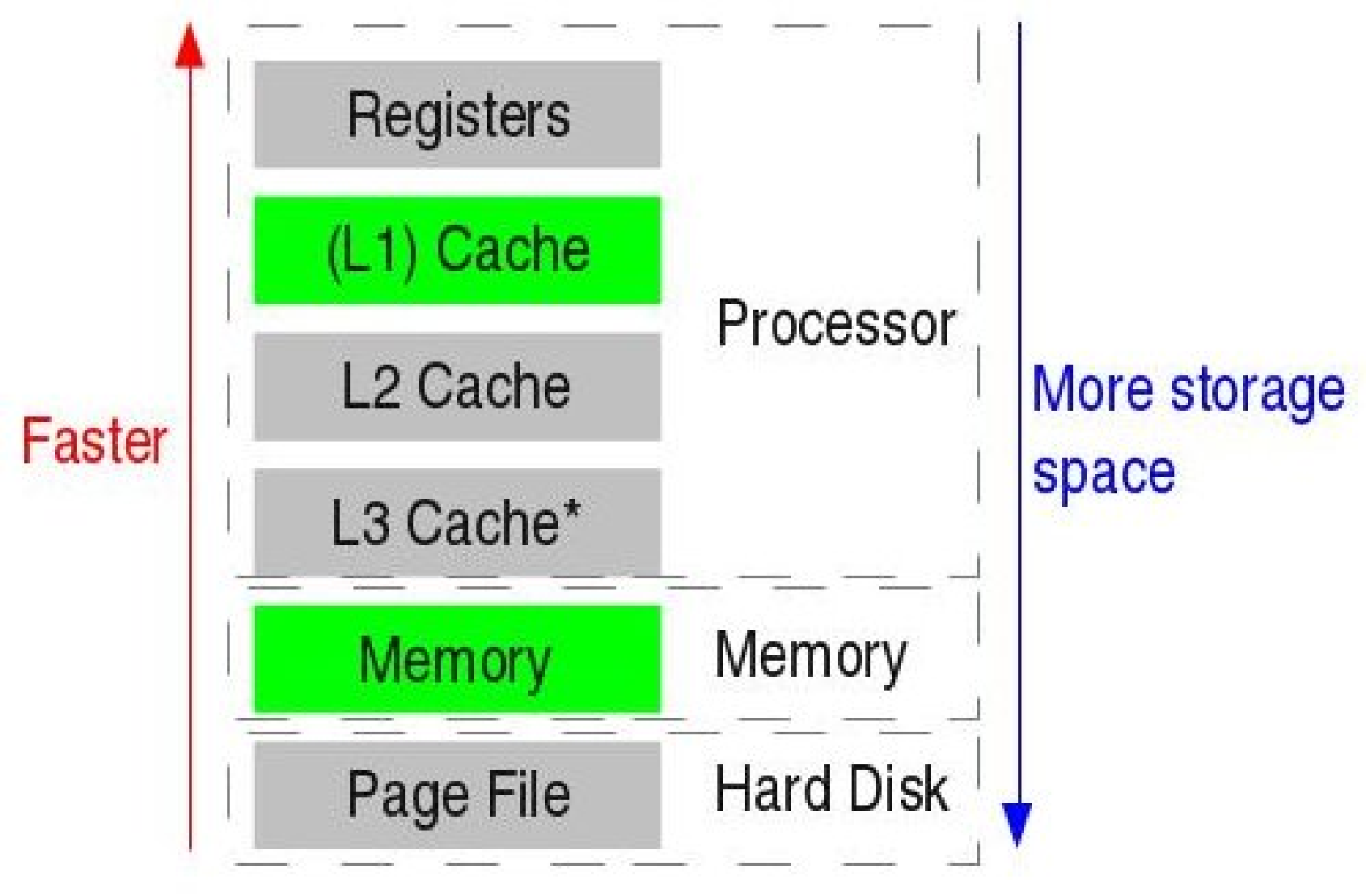
Белорусский Государственный Университет
Информатики и Радиоэлектроники

2019

Computer Memory Hierarchy



Иерархия памяти



Иерархия памяти

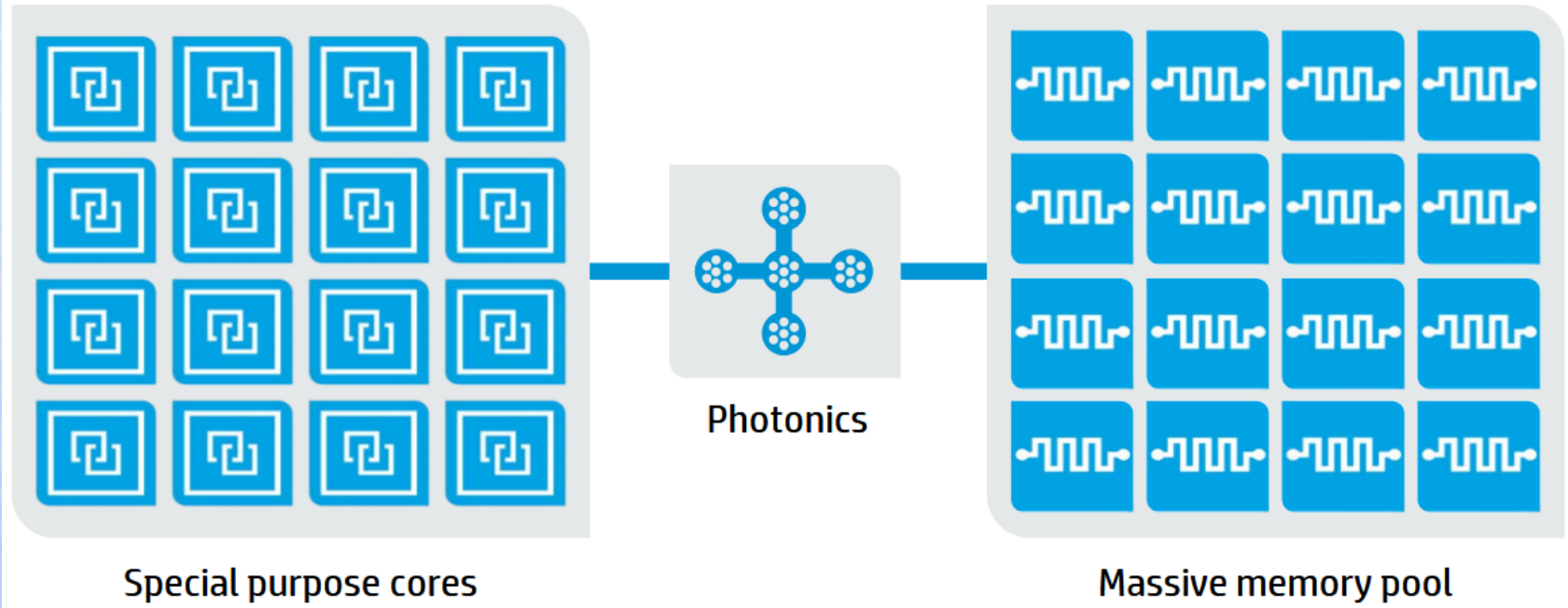
Мотивацией для иерархической модели памяти являются два свойства поведения программ:

- Временная локальность
- Пространственная локальность

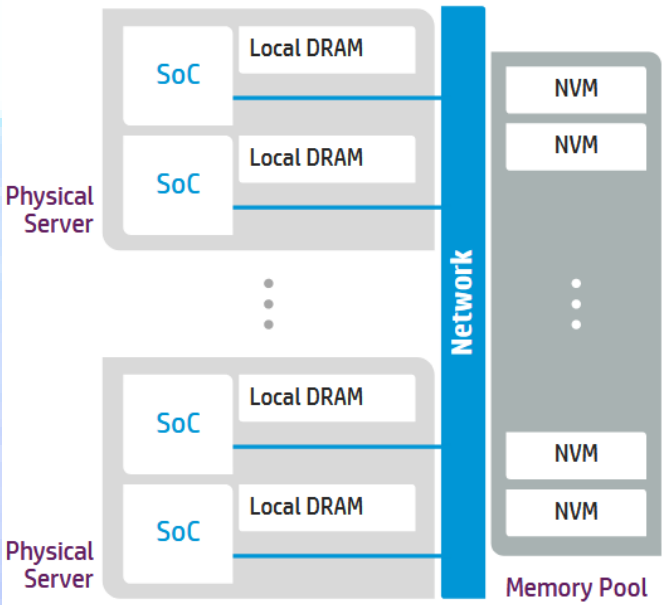
Альтернативное будущее

HP The Machine: An Architecture for Memory-centric Computing

https://www.mcs.anl.gov/events/workshops/ross/2015/slides/ross2015-keeton.pdf



Essential characteristics of The Machine



Converging memory and storage

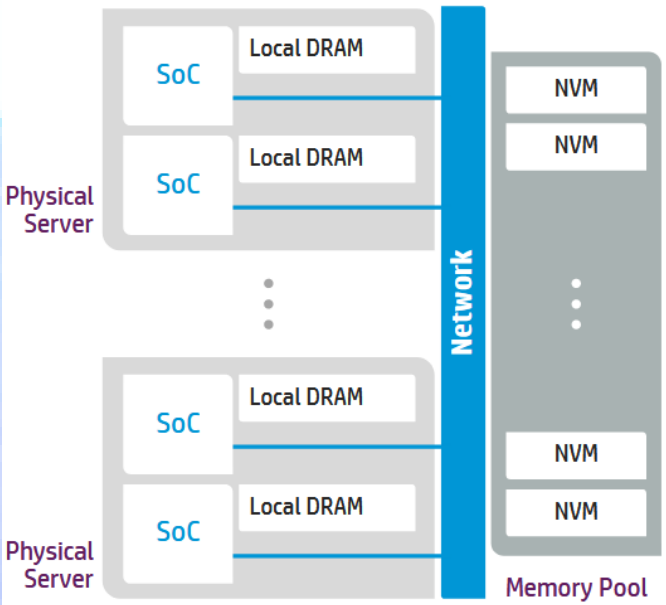
- Byte-addressable non-volatile memory (NVM) replaces hard drives and SSDs

Shared memory pool

- NVM pool is accessible by all compute resources
- Optical networking advances provide near-uniform low latency
- Local memory provides lower latency, high performance tier

Heterogeneous compute resources distributed closer to data

Essential characteristics of The Machine



Converging memory and storage

- Byte-addressable non-volatile memory (NVM) replaces hard drives and SSDs

Shared memory pool

- NVM pool is accessible by all compute resources
- Optical networking advances provide near-uniform low latency
- Local memory provides lower latency, high performance tier

Heterogeneous compute resources distributed closer to data

Raw Memory Systems

Мотивацией для иерархической модели памяти являются два свойства поведения программ:

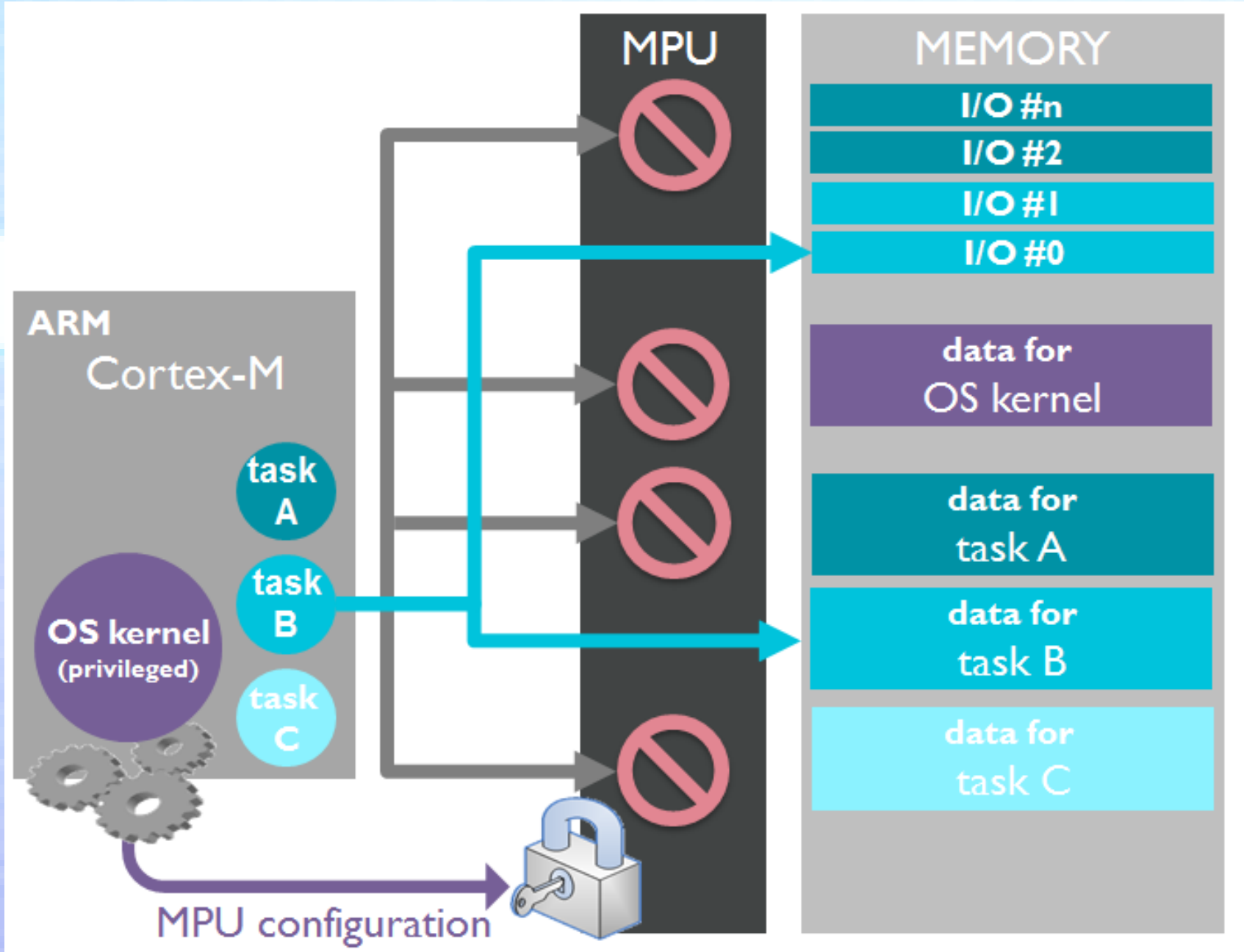
- Процессор без MMU и MPU
- Отсутствие возможности одновременного выполнения двух копий одной и той же программы
  - Многопоточность
  - Глубокий своппинг
  - Динамическое перемещение

Memory Protection Unit

Обеспечивает защиту но не виртуализацию:

- Защита ядра ОС от программ
  - Привелигерованный и непривилегированный режимы
- Защита программ друг от друга
  - Сегментация + таблица ключей

Memory Protection Unit



Связывание

Режимы адресации в процессоре:

- Абсалютная
- Относительная

Во время компиляции, компилятор формирует бинарные “объекты”, которые упаковывает в объектные файлы. Каждый объект содержит:

- Непрерывное бинарное тело
- Таблицу экспорта (имя-смещение)
- Таблицу импорта (имя-смещение)

Бинарное тело содержит уже связанные базовые блоки.

Связывание

Во время линковки Linker связывает объекты в более крупные сущности – сегменты. При этом каждый сегмент содержит:

- Непрерывное бинарное тело
- Таблицу экспорта (имя-смещение)
- Таблицу импорта (имя-смещение)

Бинарное тело сегмента содержит уже связанные функции и данные.

Динамическое перемещение

Во время загрузки программы Loader размещает тело сегмента в памяти и используя таблицы экспорта и знание фактического адреса загрузки сегмента патчит все ссылки указанные в таблице импорта.

Применение

- Простые, дешевые, маломощные микроконтроллеры с небольшим объемом памяти (0-16 Кб) – сырая память
- Сложные микроконтроллеры (Объем памяти до нескольких мегабайт) – MPU
- Микропроцессоры и сложные компьютерные системы – MMU

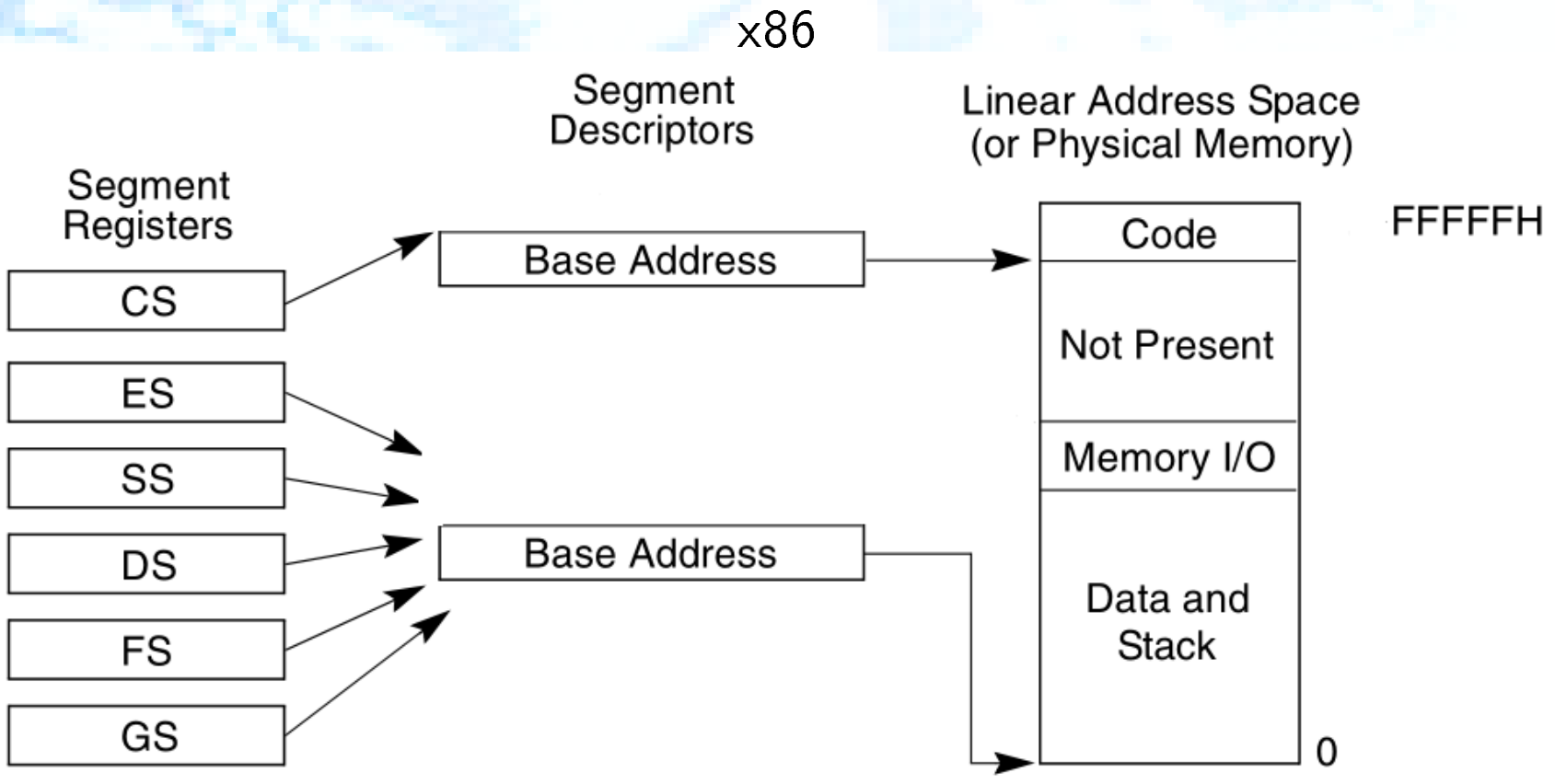
Адресное пространство

Адресное пространство – совокупность всех допустимых уникальных идентификаторов каких-либо объектов вычислительной системы, как-то ячеек памяти, секторов диска, узлов сети и т. п.

- Мощность адресного пространства – количество уникальных идентификаторов
- Размер объекта (байт, слово, страница, сектор)

Сегментация памяти

Сегмент представляет собой адресное пространство.



Сегментация памяти

Вычисление адреса.

x86

```
  0000 0110 1110 1111 0000  Segment, 16 bits
+      0001 0010 0011 0100  Offset,    16 bits
-------------------------------------------------
  0000 1000 0001 0010 0100  Address, 20 bits
```

Сегментация памяти

Сегментация может использоваться для:

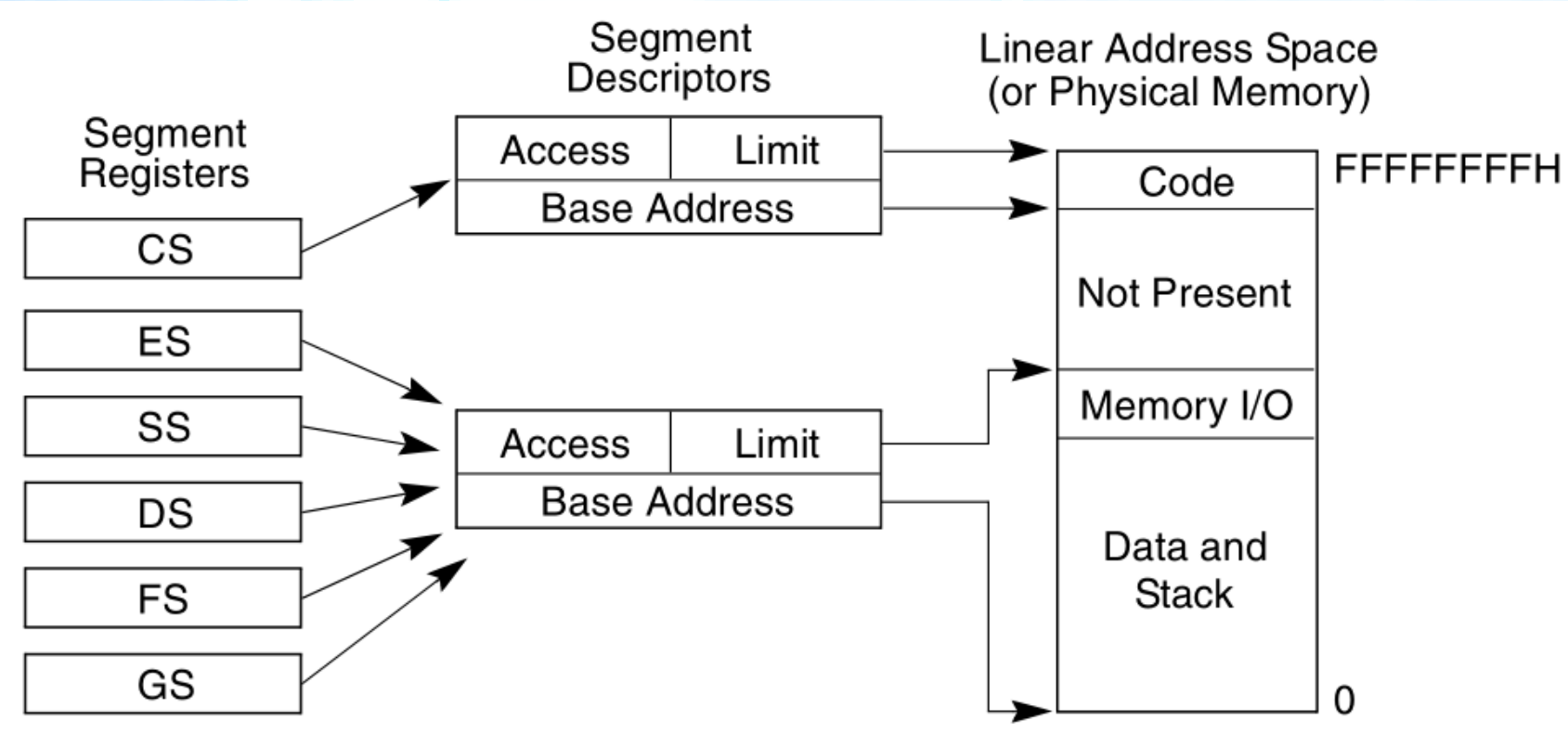
- Расширения возможностей адресации процессора (16-bit машиное слова с 20-разрядной шиной памяти, Intel 8086)
- Организации защиты памяти

Сегментация памяти

Сегментация может использоваться для:

- Расширения возможностей адресации процессора (16-bit машиное слова с 20-разрядной шиной памяти, Intel 8086)
- Организации защиты памяти (Intel 80286)
- Свопинг

Сегментация памяти

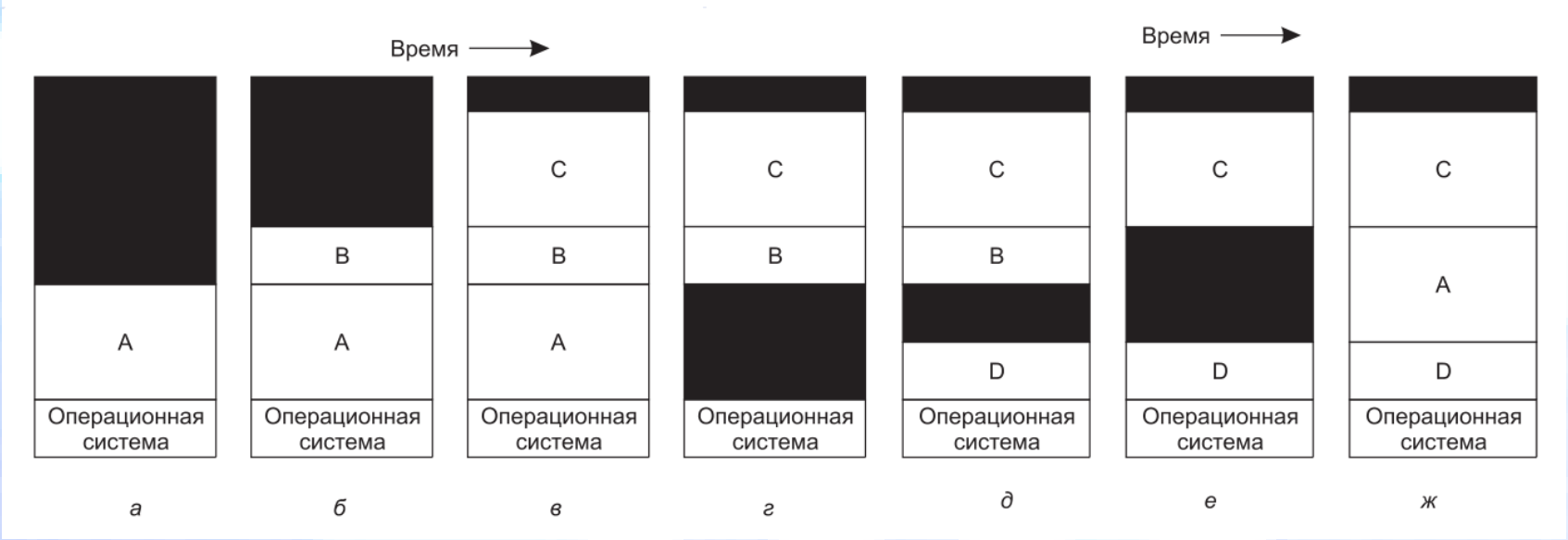


Свопинг

Свопинг (Подкачка) – механизм управления памятью, при котором отдельные фрагменты памяти (обычно неактивные) перемещаются из ОЗУ во вторичное хранилище (жёсткий диск или другой внешний накопитель, такой как флеш-память), освобождая ОЗУ для загрузки других активных фрагментов памяти.

- Использовать программе больше памяти чем есть на компьютере (предтеча виртуальной памяти)
- Использовать память более эффективно
- Загружать в память одновременно больше программ нежели это было бы возможно в системах с сырой памятью.

Свопинг



Фрагментация

- Внешняя фрагментация – ситуация при которой имеется достаточно большой объем свободной памяти для удовлетворения запроса на выделение памяти, однако запрос не может быть удовлетворен, так как отсутствуют непрерывные области памяти с размером достаточным для удовлетворения запроса.
  - Уплотнение памяти – Процесс перемещения блоков в памяти с целью увеличения размера непрерывных областей свободной памяти.
- Внутренняя фрагментация – ситуация при которой фактический размер блока выделенной памяти превышает размер запрашиваемого блока и в результате часть выделенной памяти остается фактически неиспользуемой.

Управление свободной памятью

Управление свободной памятью на основе битовых карт.

- Вся памяять делится на блоки фиксированного размера.
- Каждому блоку соответствует один бит в битовой карте.
- 1 – блок используется
- 0 – блок свободен
- Размер блока?

Управление свободной памятью

Управление свободной памятью на основе связанных списков.

```
struct MemoryFragment
{
  MemoryFragment* next;
  MemoryFragment* prev;
  size_t address;
  size_t size;
  bool type;
};

MemoryFragment* freelist_head;
```

Управление свободной памятью

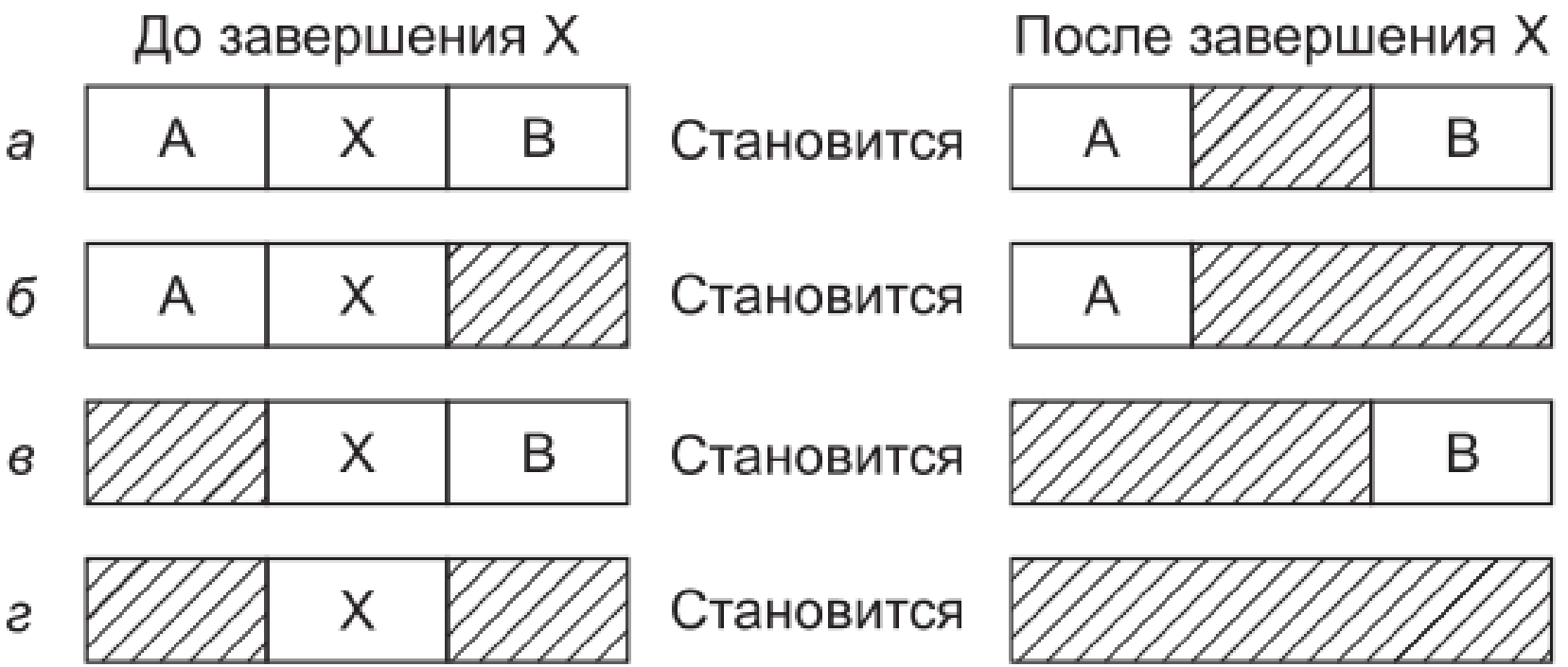
Управление свободной памятью на основе связанных списков.

```
struct MemoryFragment
{
  MemoryFragment* next;
  MemoryFragment* prev;
  size_t address;
  size_t size;
};

MemoryFragment* freelist_head;
```

Управление свободной памятью

Уплотнение списков:



Политики поиска на списке

- Первый подходящий
- Следующий подходящий
- Наиболее подходящий
- Наименее подходящий

Виртуальная память

Виртуальная память – техника управления памятью которая предоставляет идеальную абстракцию ресурса памяти фактически доступной в системе, и которая создает для каждого пользователя иллюзию монопольного владения всей памятью компьютерной системы.

Виртуальная память создает новый слой абстракции при работе с памятью.

Страничная организация памяти подразумевает разбиение адресного пространства на множество элементарных частей фиксированного размера.

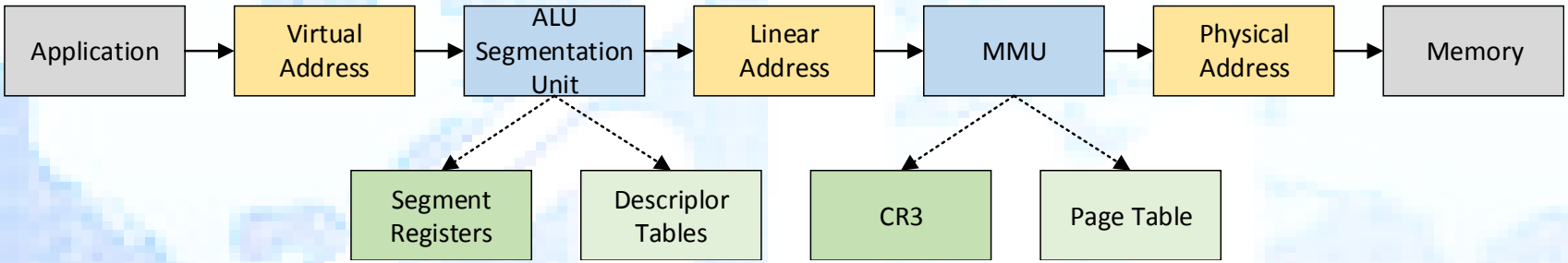
Виртуальная память

Физическое адресное пространство является статическим и всегда присутствует в системе в единственном экземпляре.

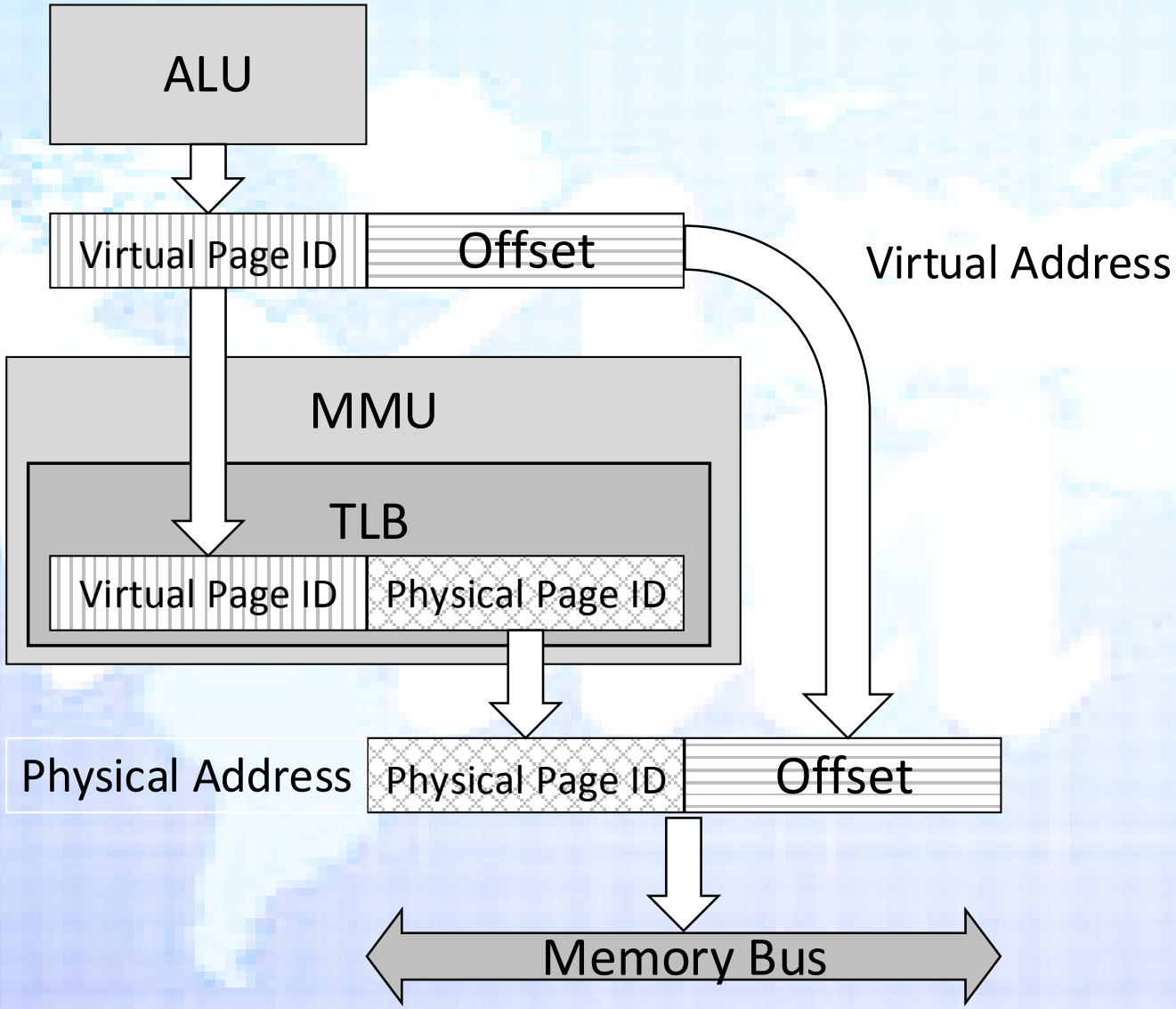
Виртуальное адресное пространство является динамическим и всегда присутствует в системе как минимум в одном экземпляре.

Реализация виртуальной памяти является результатом кооперации MMU и ядра ОС.

Виртуальная память



Виртуальная память



Виртуальная память

Физическое адресное пространство и виртуальное адресное пространство имеют фиксированный размер.

Однако, в отличие от физического, слоты виртуального адресного пространство могут быть либо заполнены, либо пусты.

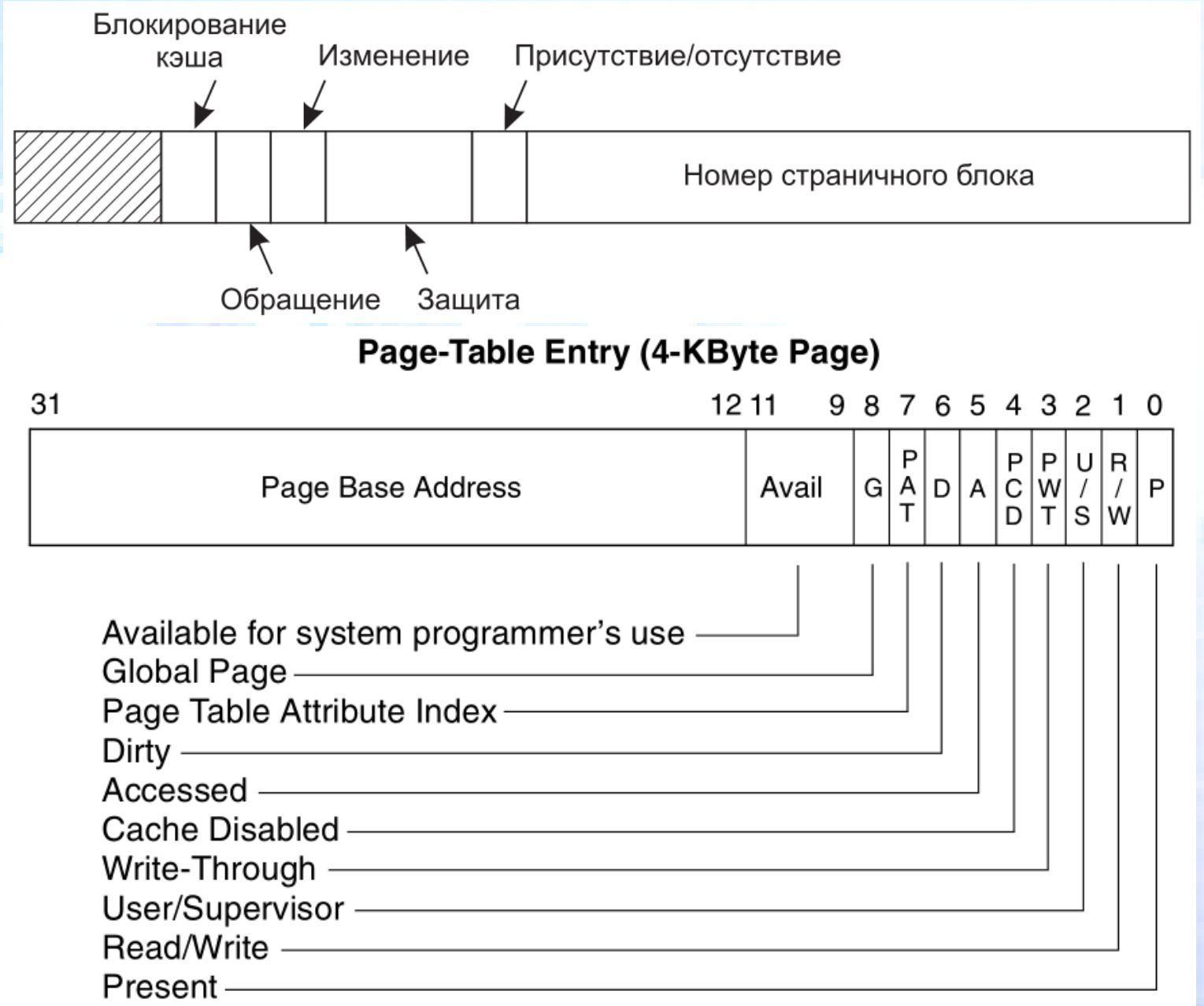
Попытка обращения к пустому слоту может детектироваться MMU и обслуживаться ОС – Page Fault (Отказ страницы). Этот механизм является основой для:

- Организации защиты
- Виртуализации

Виртуальная память

Концептуально, таблица страниц является простым ассоциативным массивом, который хранит правила трансляции виртуальных адресов в физические.

Page Frame Entry



Виртуальная память

Хорошая новость: “Любую проблему можно решить введением дополнительного уровня абстракции” (c) Kevlin Henney

Плохая новость: уровень абстракции не бесплатен.

Поэтому: “Любую проблему можно решить введением дополнительного уровня абстракции, *кроме проблемы чрезмерного количества уровней абстракции*” (c) Kevlin Henney

На виртуальную память тратятся ресурсы:

- Процессорного времени
- Физической памяти

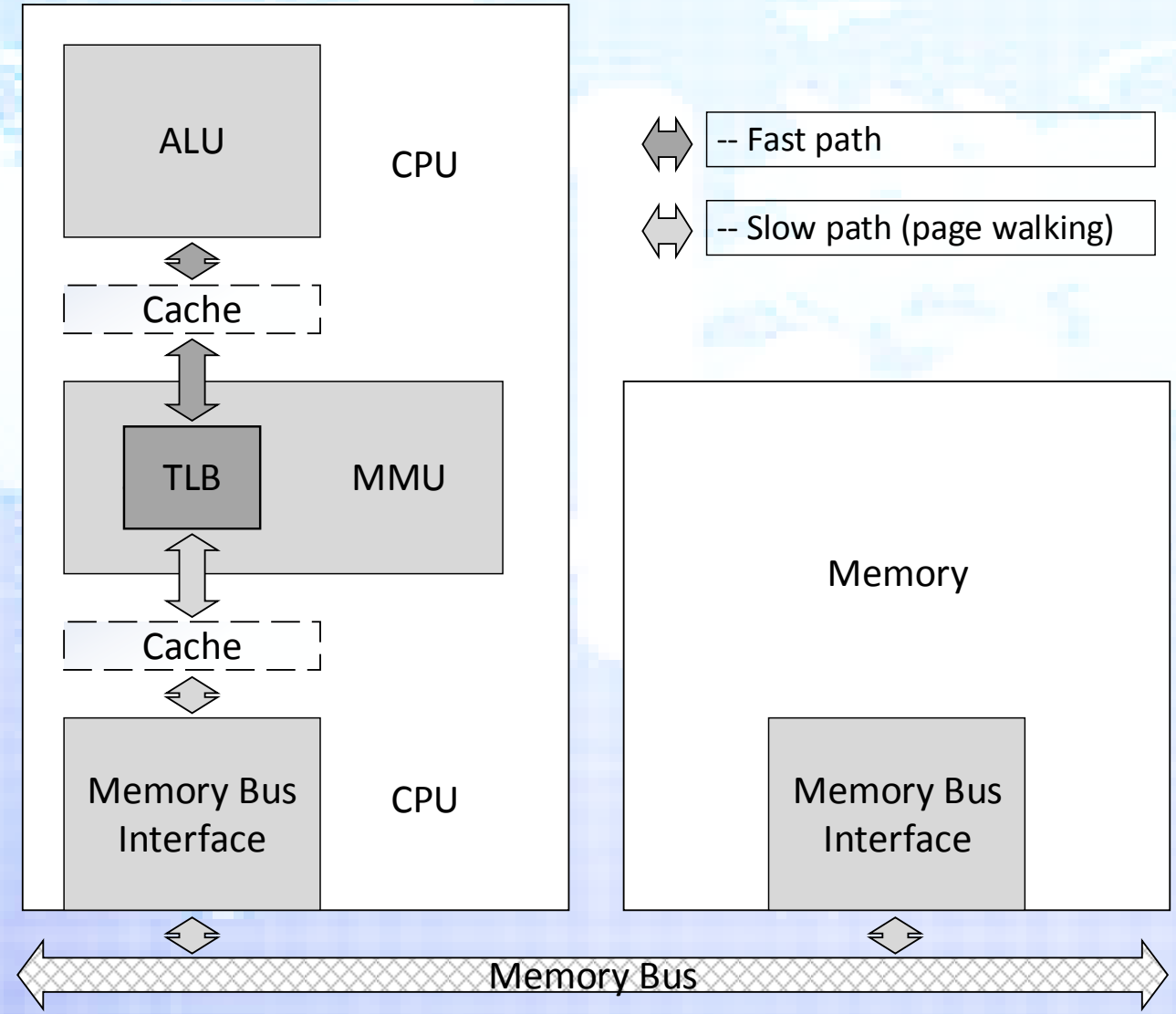
Виртуальная память

Наивный подход: Таблица страниц как непрерывный массив дескрипторов страниц в памяти

Проблемы:

- Как минимум удваивается количество обращений к памяти со стороны процессора
- Каждое виртуальное адресное пространство потребляет большой объем физической памяти (IA-32: 4MB или 0.1% всей физической памяти в системе)

Решение №1 – TLB.



TLB

Оптимизации:

- Global bit в Page Table Entry
- Huge pages

Без HP:

```
    10,969      iTLB-load-misses
 5,945,847      iTLB-loads
    26,007      dTLB-load-misses
 3,815,595      dTLB-loads
```

С HP:

```
     6,614      iTLB-load-misses
 4,301,442      iTLB-loads
    13,199      dTLB-load-misses
 1,792,403      dTLB-loads
```

TLB

Существуют программно управляемые TLB:

- +Проще аппаратная реализация
- +ОС может оптимизировать работу памяти применяя политики
- -Медленная работа

Отказ TLB:

- Программный
- Аппаратный

Отказ страницы:

- Мягкий
- Жесткий

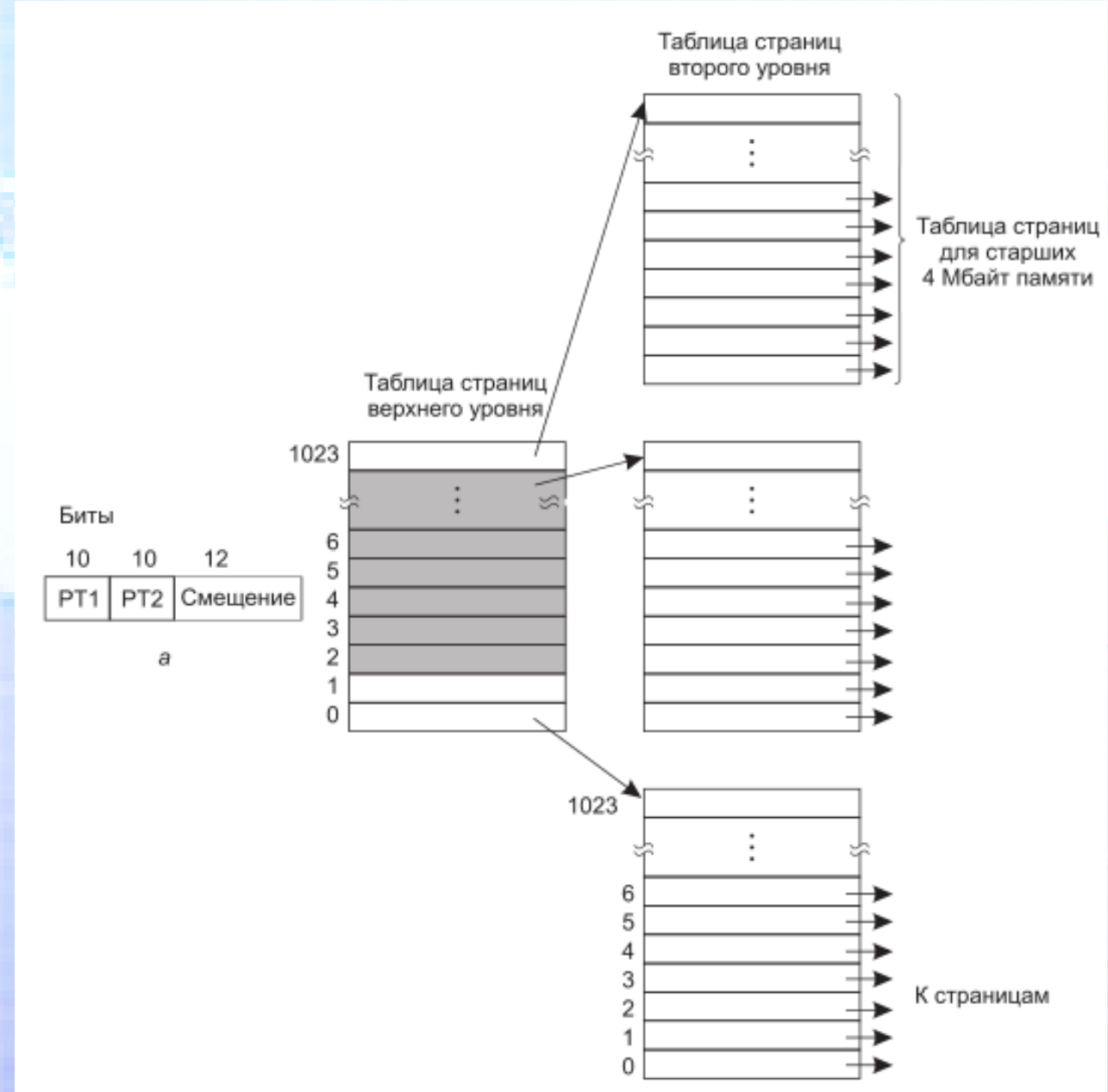
Таблицы страниц

Проблема: потребление физической памяти.

Наблюдение: большинство программ используют небольшое количество физической памяти и соответственно небольшую часть виртуального адресного пространства.

Решение: иерархические таблицы страниц.

Таблицы страниц



Таблицы страниц

x86:

- IA-32: 2 уровня
- AMD64: 3 уровня
- VT-x : 4 уровня

- +Эффективность использования физической памяти в среде приложений использующих малое ее количество
- -Увеличение стоимости отказа TLB (прохода по таблицам страниц)

Алгоритмы замещения страниц

Рассматриваем подкачку в среде с недостатком физической памяти и, следовательно, с высокой конкуренцией за ее использование.

Задача: определить какую страницу нужно откачать на диск для того чтобы освободить занимаемую ей память и передать ее в использование другому приложению.

Оптимальный алгоритм замещения страниц

Предполагаем что мы знаем полную историю обращений к памяти (в том числе будущее). При таких условиях оптимальным

будет замещение страницы, к которой дольше всего не будет обращений начиная с текущего момента истории. Красиво, но

не реализуемо. Все алгоритмы используемые на практике являются эвристическими.

Not Recently Used

NRU опирается на аппаратную поддержку – биты A и D в PTE.

NRU может быть реализован программно.

- Не было обращений
- Не было обращений, но страница модифицирована
- Страница недавно использовалась, но не изменялась
- Страница недавно использовалась и модифицирована

First In – First Out

FIFO предполагает отслеживание страниц памяти в виде списка.

При добавлении страницы в память, она помещается в хвост.

При необходимости освобождения страницы, она изымается из головы списка.

Second Chance

SC – модификация FIFO.

В SC при извлечении страницы из головы происходит проверка бита A, и если он установлен, то попытка считается неудавшейся, бит A сбрасывается, а страница добавляется в хвост, как только что прибывшая.

Clock

Clock – модификация SC.

В Clock список замыкается в кольцо и поддерживается указатель на самый давно непроверявшийся элемент кольца. При поиске страницы необходим только проход по кольцу и сброс битов A. Операции вставки и удаления элементов из кольца становятся редкоиспользуемыми.

Least Recently Used

LRU – модификация NRU.

В дополнение к битам A и D, ОС прикрепляет к каждой странице timestamp в котором регистрирует последне время обращения к странице.

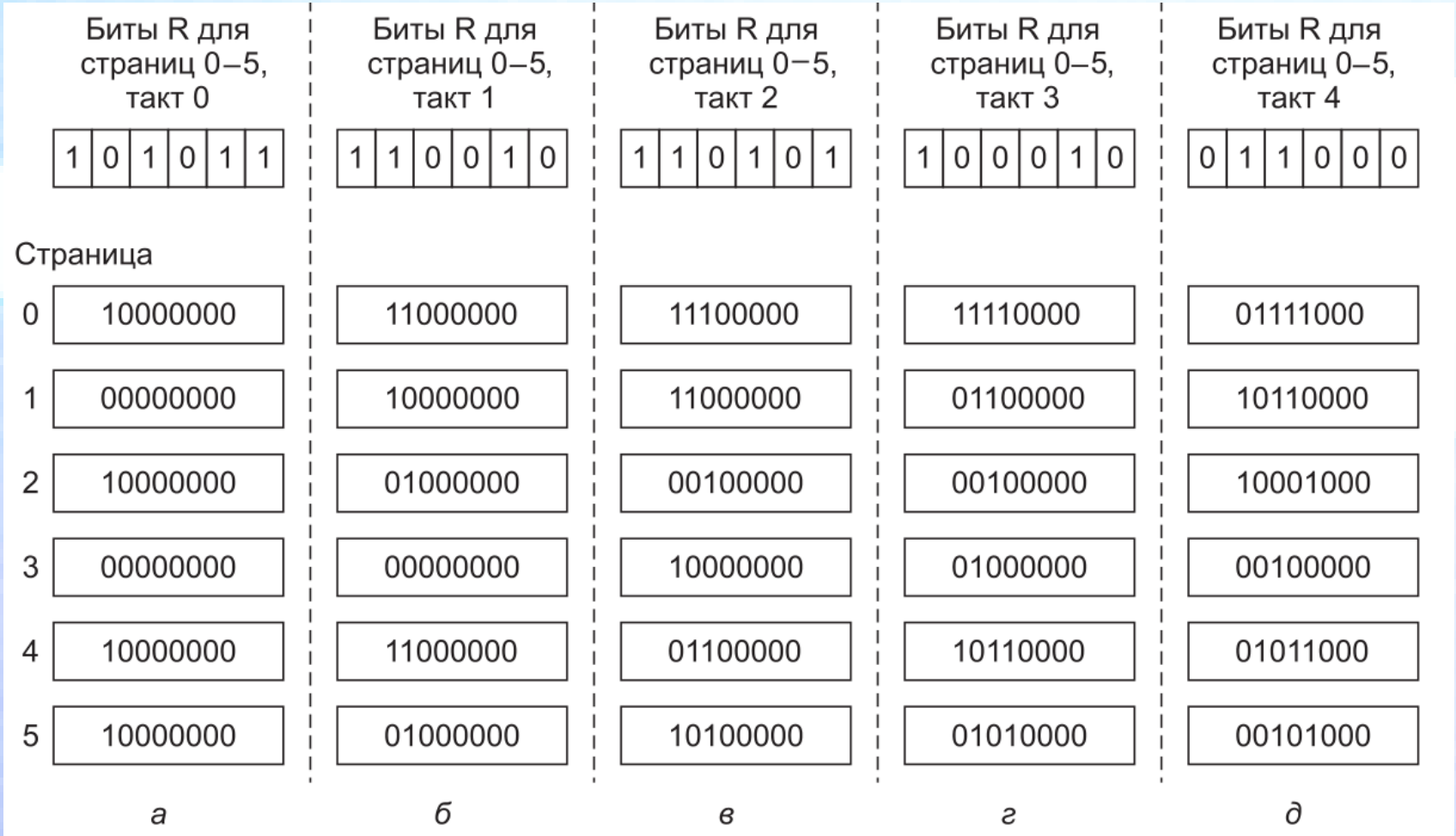
При необходимости замещения страницы, ищется страница с наименьшим значением timestamp.

Aging

Модификация NFU.

Отслеживаем недавнюю историю обращений к странице за счет осуществления побитового сдвига перед прибавлением к счетчику значения бита A. За счет сдвига самая старая запись в истории выталкивается из нее.

Aging


Биты R для страниц 0–5, такт 0
1 0 1 0 1 1
Биты R для страниц 0–5, такт 1
1 1 0 0 1 0
Биты R для страниц 0–5, такт 2
1 1 0 1 0 1
Биты R для страниц 0–5, такт 3
1 0 0 0 1 0
Биты R для страниц 0–5, такт 4
0 1 1 0 0 0
Страница
0 10000000 11000000 11100000 11110000 01111000
1 00000000 10000000 11000000 01100000 10110000
2 10000000 01000000 00100000 00100000 10001000
3 00000000 00000000 10000000 01000000 00100000
4 10000000 11000000 01100000 10110000 01011000
5 10000000 01000000 10100000 01010000 00101000
а
б
в
г
д

On Demand Paging

Он же Lazy Paging. Согдасно этому подходу память процессу выделяется через механизм отказа страниц

On Demand Paging опирается на временную локальность и пространственную локальность вычислений.

Набор страниц памяти интенсивно используемых в определенный момент времени называется **рабочий набор (working set)**.

Оптимизации на основе сбора статистики – Prefetching.

On Demand Paging

Менеджер памяти периодически проходит по всем виртуальным адресным пространствам и фиксирует время (логическое) последнего обращения к каждой странице.

Параметр k – максимальное время с момента последнего обращения к странице и текущего времени задает условия включения в рабочий набор.

При необходимости освобождения памяти, менеджер памяти делает еще один проход и освобождает все страницы соответствующие условию: $t_{current} - t_{page\_id,last\_access} > k$

WSClock

Clock + Working Set. Истории обращений к странице замкнутые в кольцо.

Является основным алгоритмом в современных промышленных ОС.

Summary

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 «первой пришла, первой и ушла»	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 неплохим приближением к алгоритму LRU
Алгоритм рабочего набора	Весьма затратный для реализации алгоритм
WSClock	Вполне эффективный алгоритм