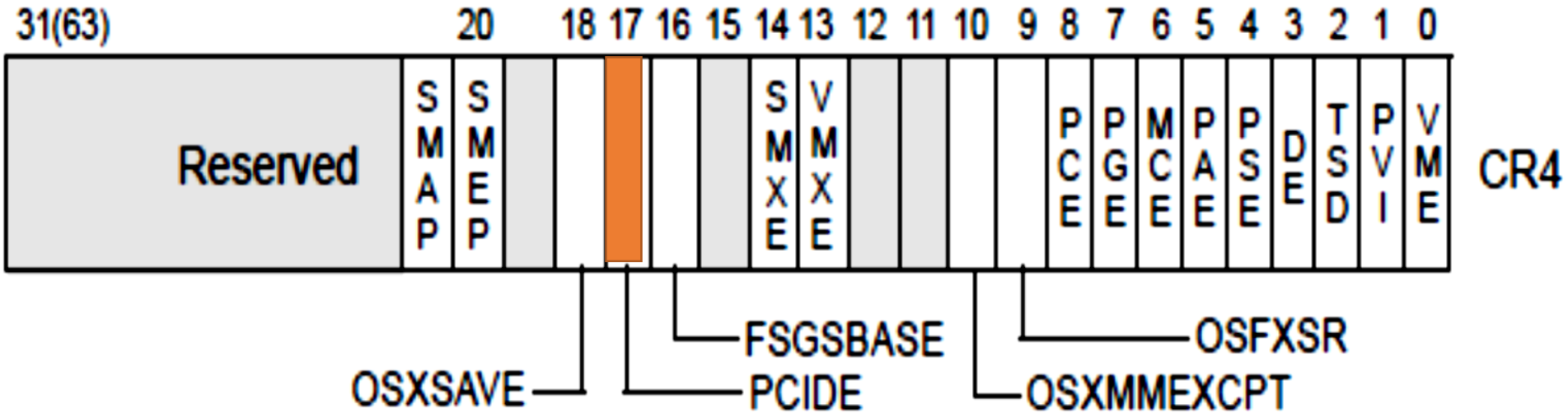
# Кэширование данных при трансляции

- Идентификаторы контекста процесса (PCID)
- Буфер быстрого преобразования адреса (Translation Lookaside Buffer, TLB)
- Кэши структур страничной трансляции

# Process-Context Identifiers (PCID)

- 12-разрядные идентификаторы контекста процесса – средства кэширования  данных для нескольких линейных адресных пространств



# Translation Lookaside Buffer (TLB)

- Процессор может ускорить процесс страничной трансляции с помощью кэширования отдельных трансляций в буфере быстрого преобразования адреса (TLB)
- TLB содержит:
  - ✓ физический адрес страничного фрейма
  - ✓ права доступа
  - ✓ значение флага «Dirty»
  - ✓ тип памяти

# Кэши структур страничной трансляции

- Кэш PML4:

  физический адрес PDPT

  права доступа элемента структуры PML4
- Кэш PDPTE

  физический адрес каталога страниц

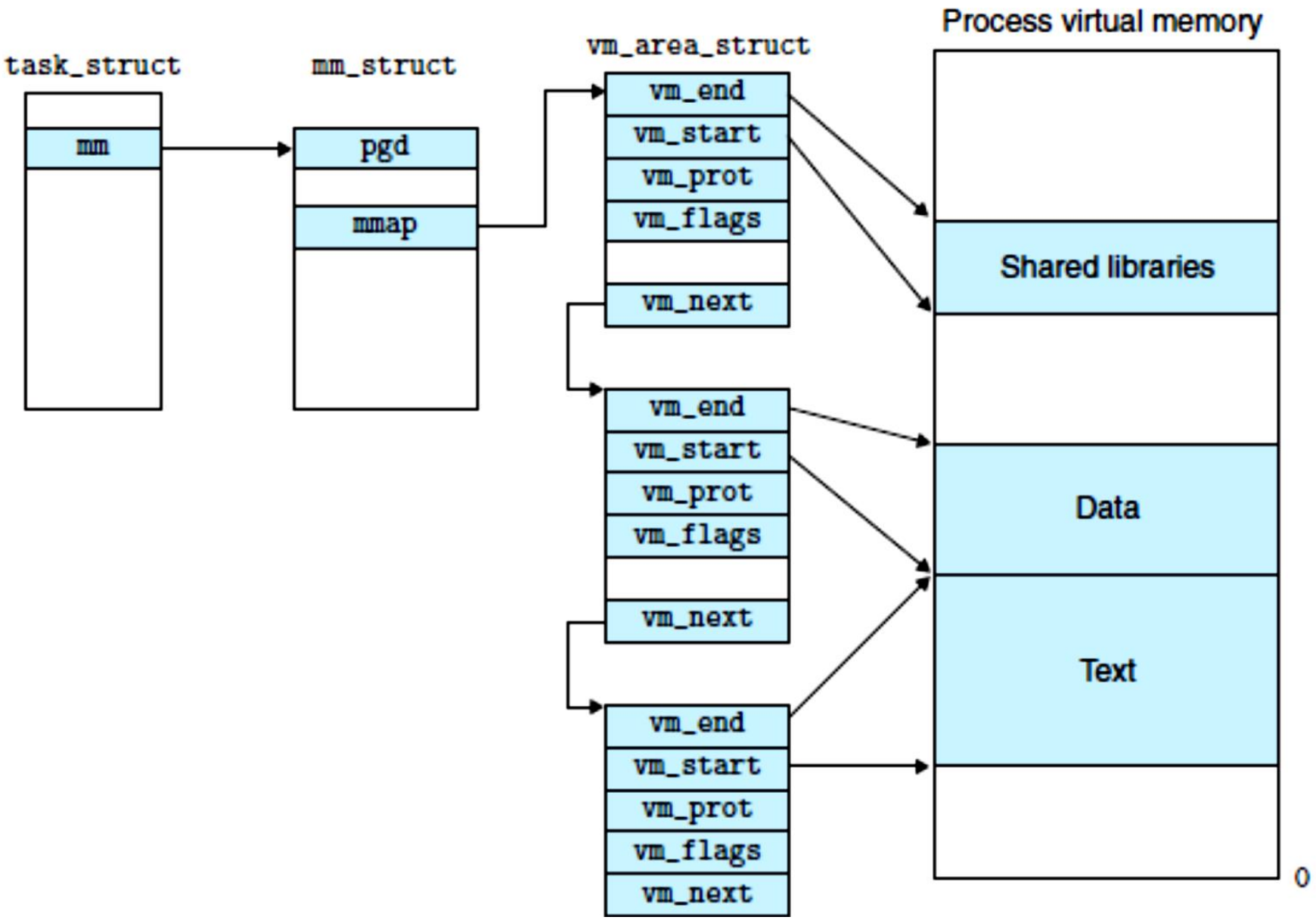
  права доступа в элементах структур PML4 и PDPT
- Кэш PDE

  физический адрес таблицы страниц

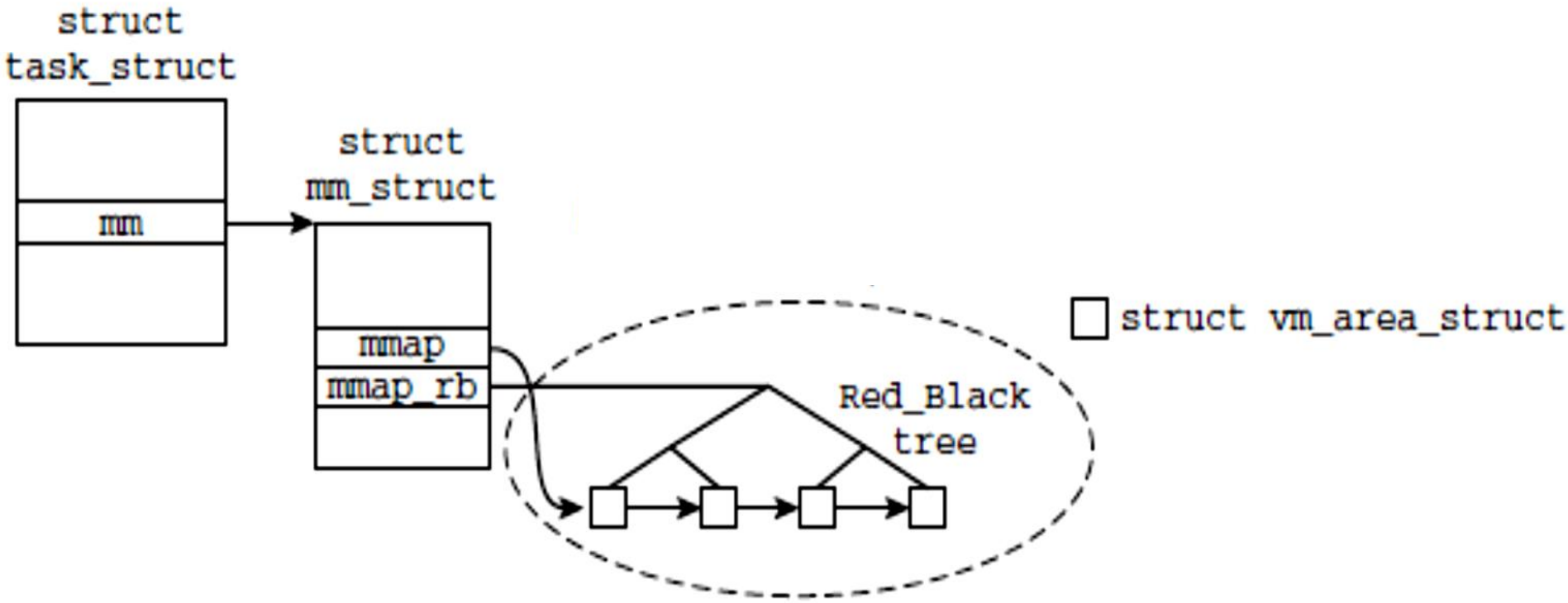
  права доступа в элементах структур PML4, PDPT и каталога страниц

# Управление памятью в Linux

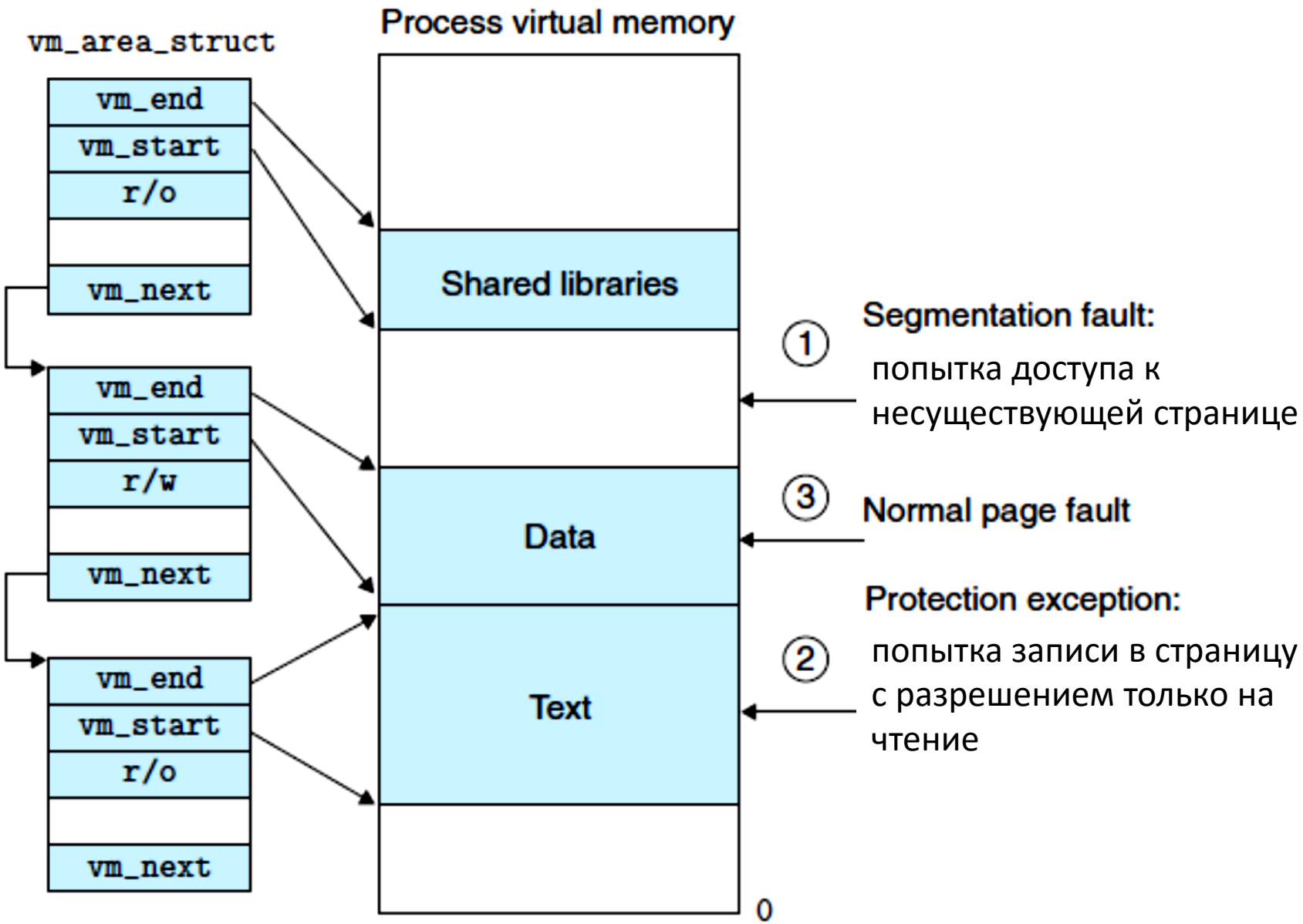
# Управление памятью в Linux



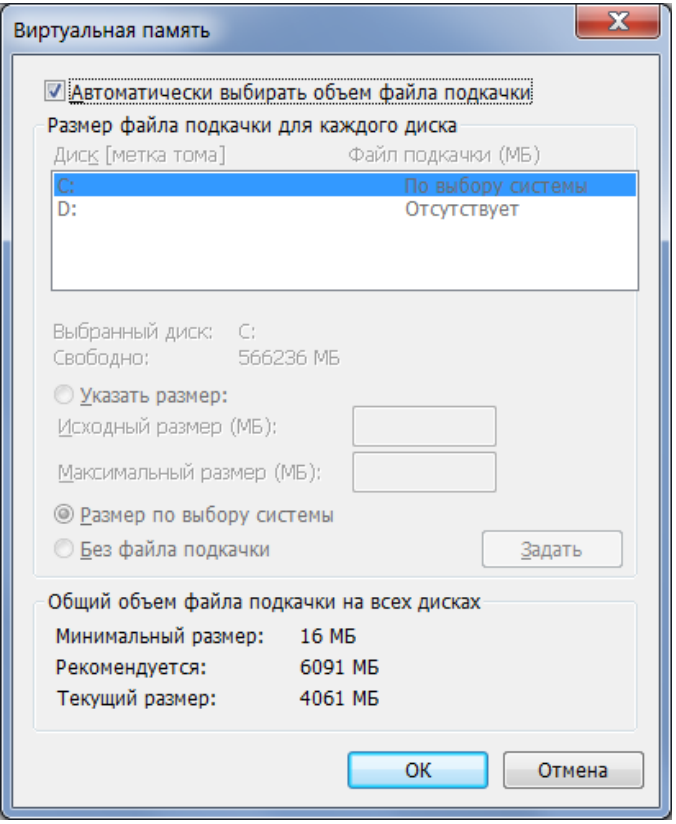
# Управление памятью в Linux



# Управление памятью в Linux



# Область свопинга




Настройка в Windows 7

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    nodev,noexec,nosuid 0       0
# / was on /dev/sda1 during installation
UUID=b135050d-4472-4c34-80f5-d79bdfe7b130 /               ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
UUID=792a319a-5836-45f2-b2df-d3aa482b9c55 none            swap    sw              0       0
```

В Linux области свопинга задаются командой **`swapon`**

Пример – область свопинга в **`/etc/fstab`**

# Алгоритмы замещения страниц

- **Оптимальный алгоритм** - каждая страница может быть помечена количеством команд, которые будут выполняться перед первым обращением к этой странице. Оптимальный страничный алгоритм просто сообщает, что должна быть выгружена страница с наибольшей меткой.
- **Алгоритм NRU** (Not Recently Used – не использовавшаяся в последнее время) - замещению подлежит самая неиспользуемая страница. При определении такой страницы используются биты обращения и изменения из дескриптора страницы.
- **Алгоритм FIFO** - выгружается из памяти страница в голове списка, а новая страница добавляется в его конец.
- **Алгоритм «вторая попытка»** - модификация алгоритма FIFO. В этом случае у самой старой страницы проверяется бит обращения. Если он равен 0, страница не используется, поэтому немедленно заменяется новой. Если он равен 1, то ему присваивается значение 0, страница переносится в конец списка, а время её загрузки обновляется, т.е. считается, что страница только что попала в память.
- **Алгоритм «часы»** - все страницы хранятся в кольцевом списке. Указатель («стрелка часов») указывает на старейшую страницу. Когда происходит страничное прерывание, проверяется эта страница. Если её бит обращения равен 0, страница выгружается, на её место становится новая страница, а указатель сдвигается вперёд на одну позицию. Если бит равен 1, то он сбрасывается, указатель перемещается к следующей странице. Этот процесс повторяется до тех пор, пока не находится та страница, у которой бит обращения = 0.
- **Алгоритм LRU** (Least Recently Used – наименее недавно используемая страница) - выгружается из памяти страница, которая не использовалась дольше всего.

# Управление свободной памятью

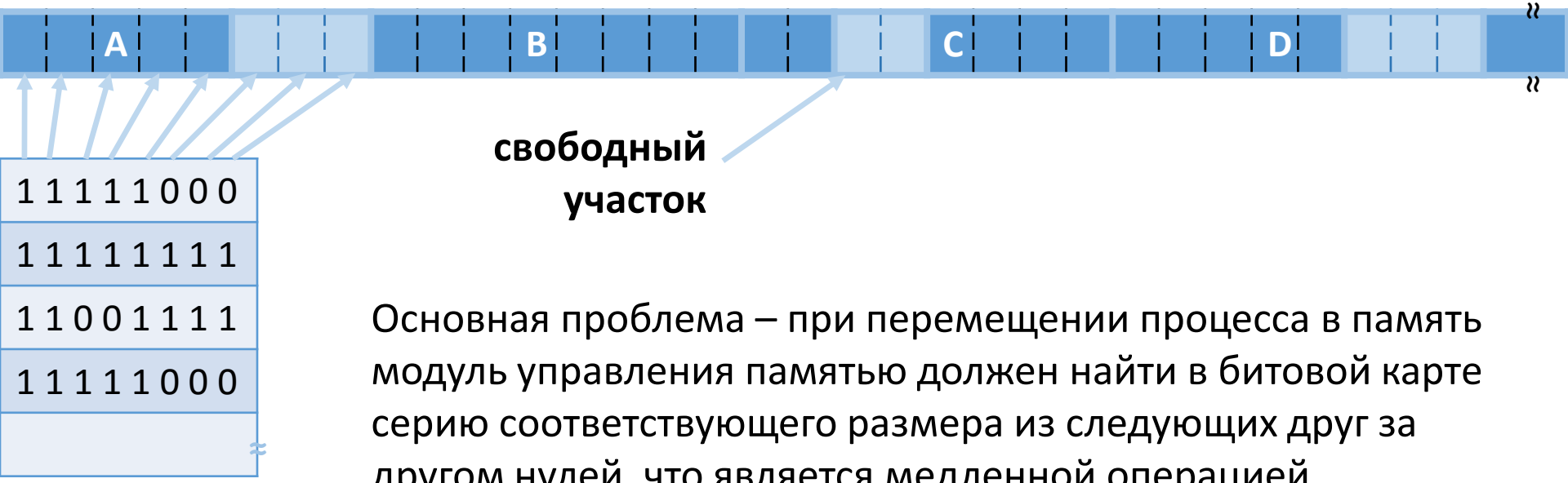
- Управление памятью с помощью битовых массивов

- Управление памятью с помощью связных списков

# Управление памятью с помощью битовых массивов

Один из способов учёта использования памяти ***– битовые массивы***, иногда называемые ***битовыми картами***.

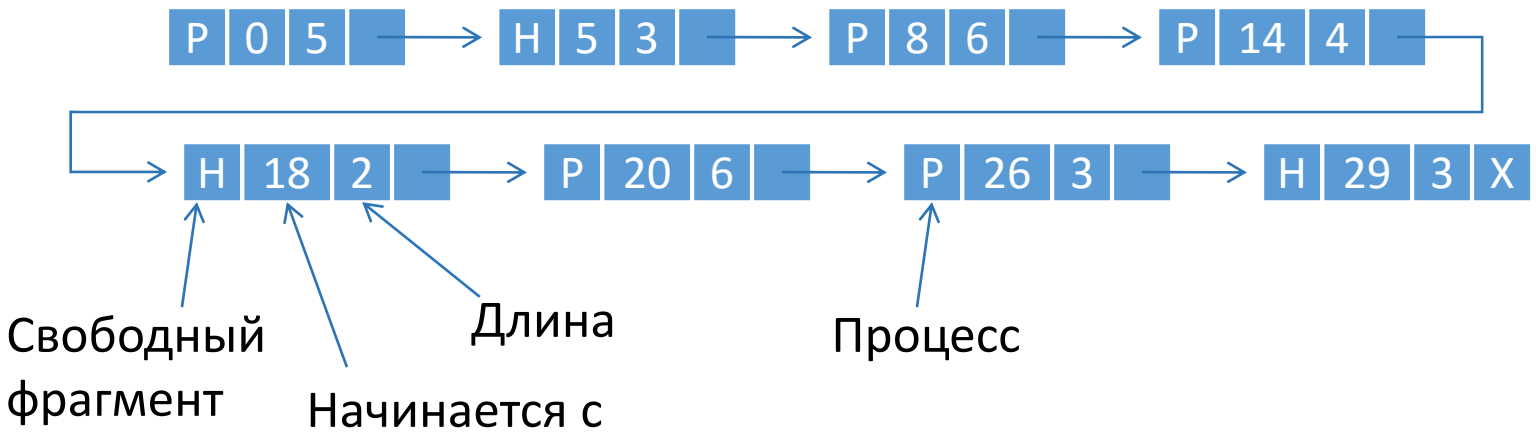
При работе с битовым массивом память разделяется на единичные блоки размещения размером до нескольких килобайт. В битовой карте каждому блоку соответствует один бит, равный нулю, а каждому занятому блоку – бит, установленный в 1 (или наоборот).




Основная проблема – при перемещении процесса в память модуль управления памятью должен найти в битовой карте серию соответствующего размера из следующих друг за другом нулей, что является медленной операцией.

# Управление памятью с помощью связных списков

Другой способ отслеживания состояния памяти предоставляет поддержка связных списков занятых и свободных фрагментов памяти, где сегментом является или процесс, или участок между двумя процессами.



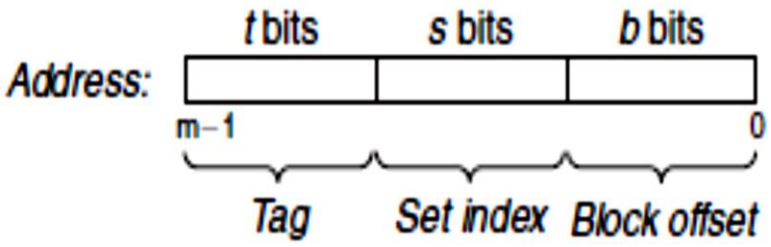
# Алгоритмы выделения памяти

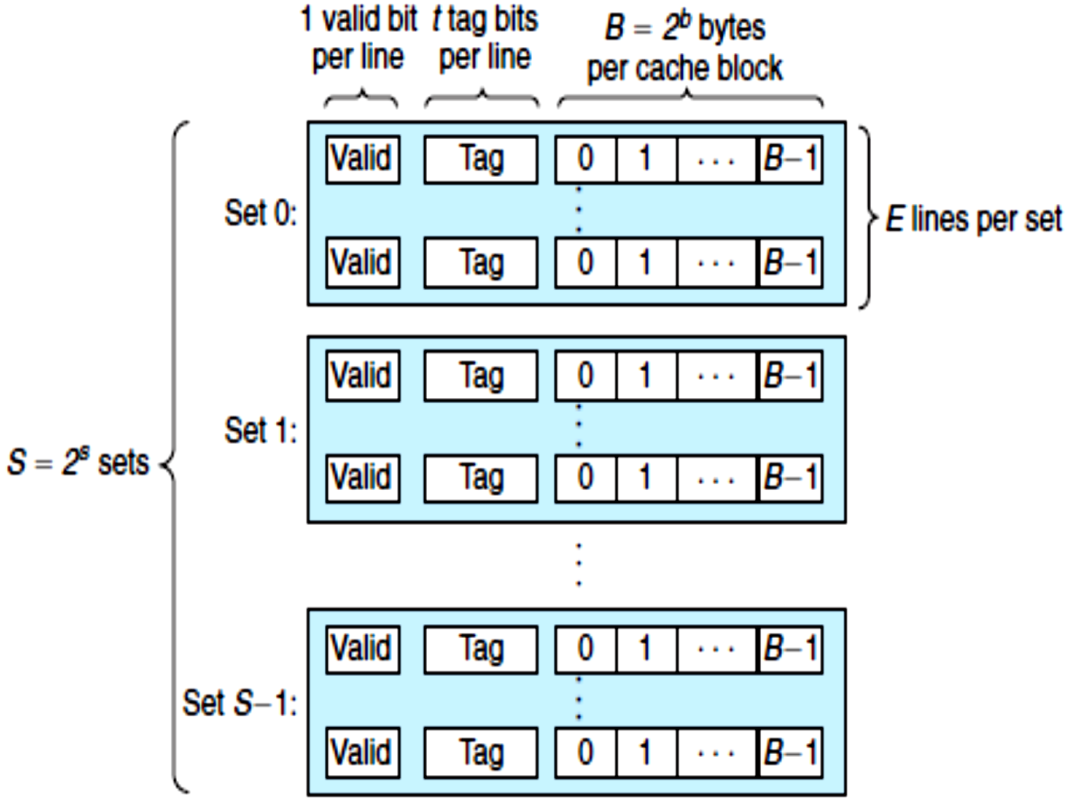
- **Алгоритм «Первая подходящая»**. Сканирование списка свободных областей и выбор первой, подходящей по размеру

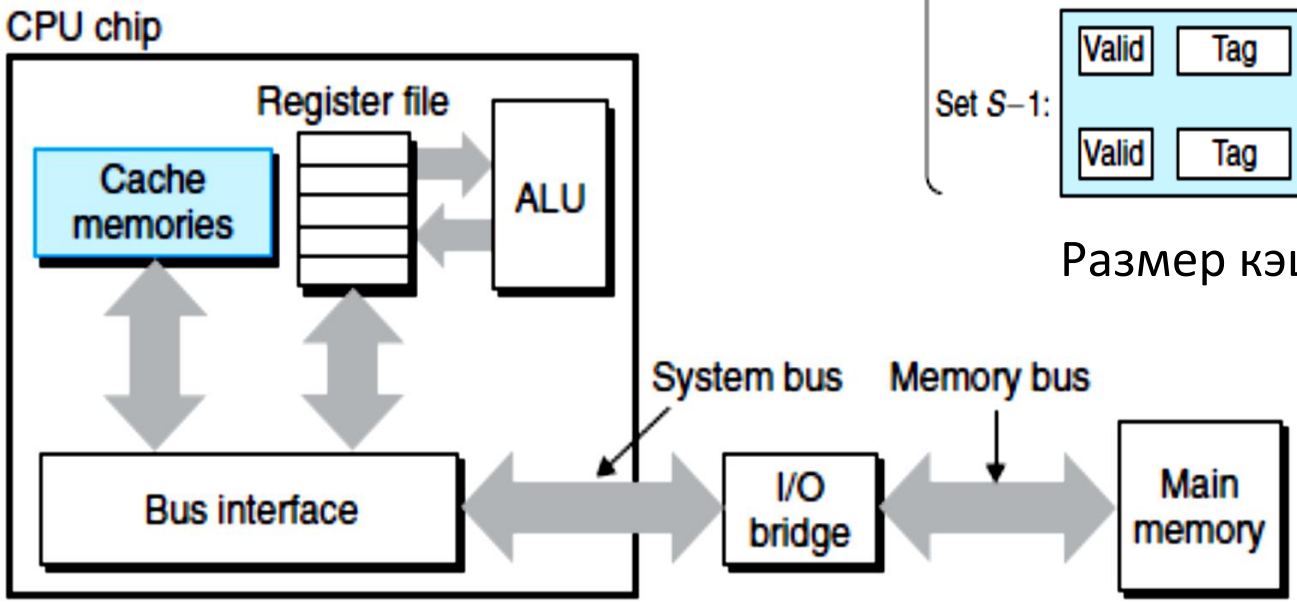
- **Алгоритм «Следующая подходящая»**. Вариант алгоритма «Первая подходящая» с запоминанием местоположения

- **Алгоритм «Наиболее подходящая»**. Выбирается наименьшее соответствующее пустое пространство памяти

# Организация кэша

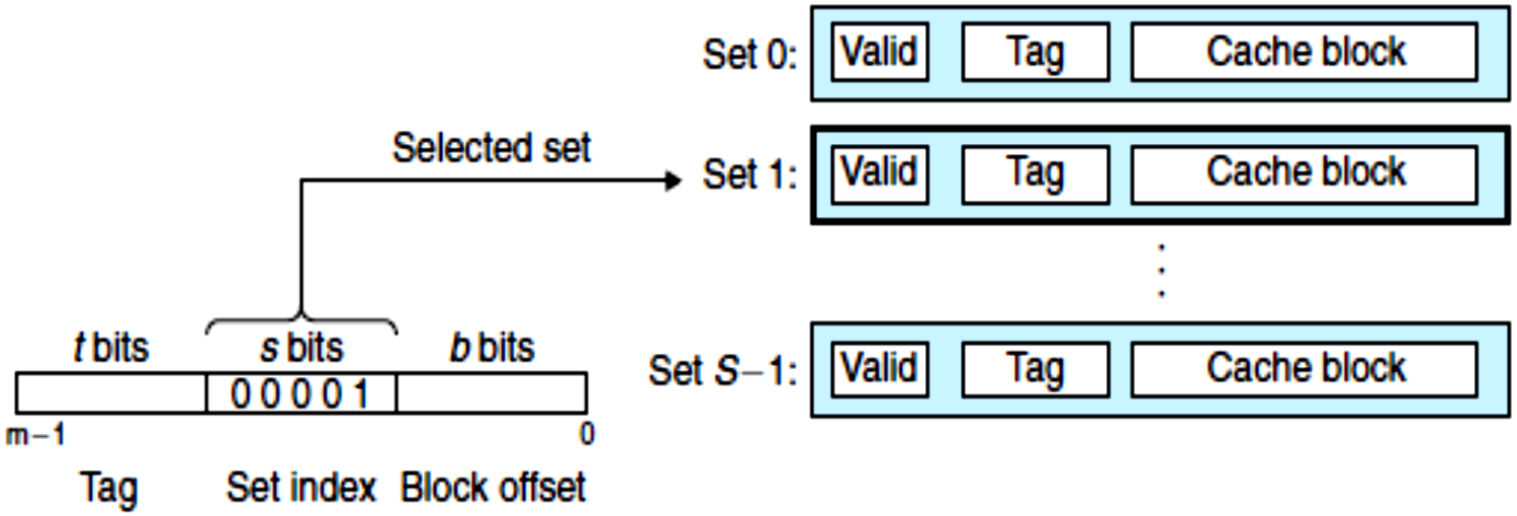


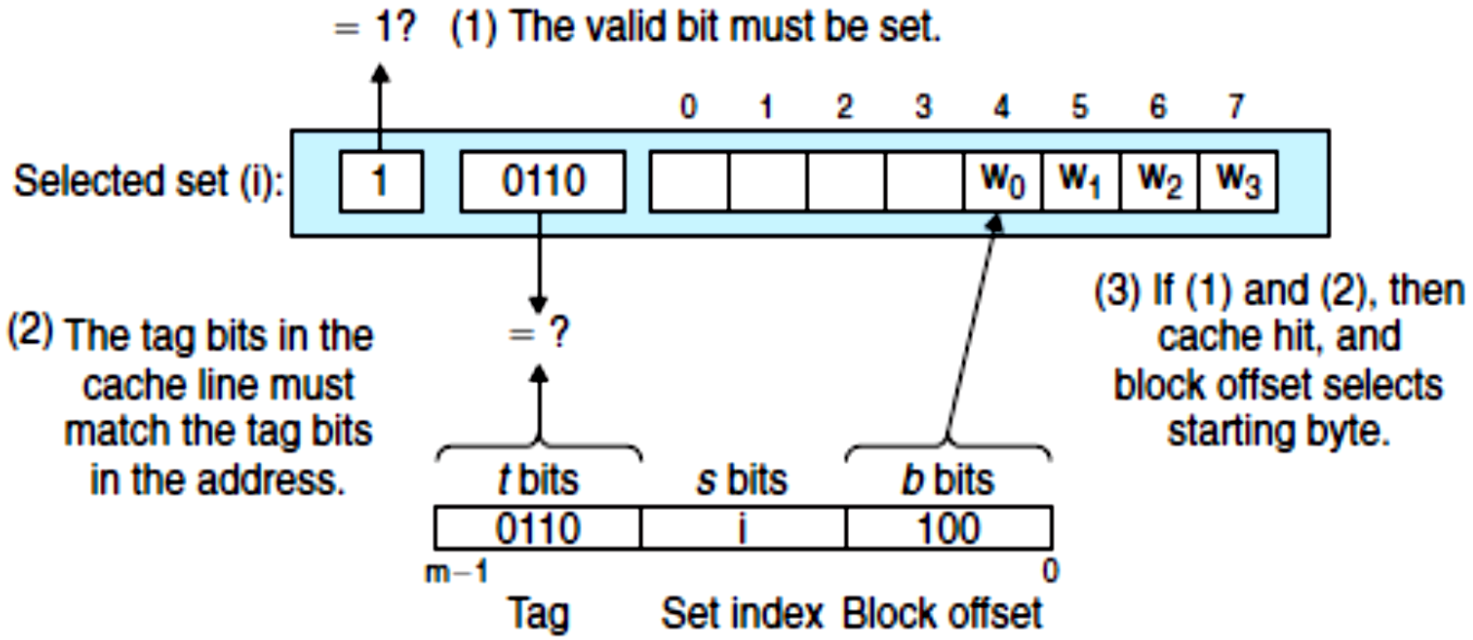



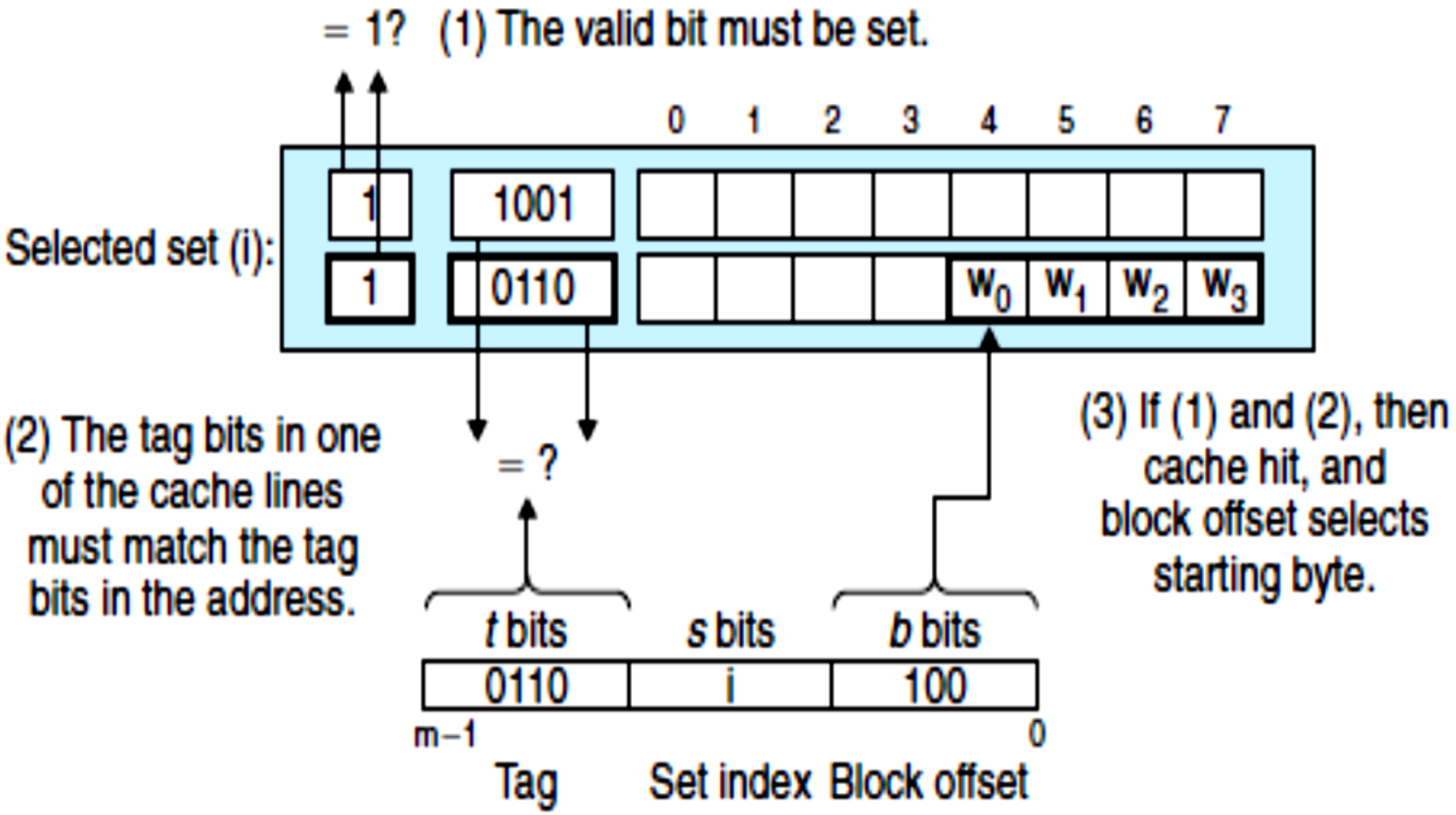

Размер кэша: C = B x E x S байт

# Кэш прямого отображения



# Ассоциативный кэш с множественным доступом



# Полностью ассоциативный кэш