Аппаратная поддержка управлением памятью

Сегментная организация памяти

- Является следствием логического разделения памяти
- Размер сегмента ограничен адресным пространством CPU, может меняться динамически

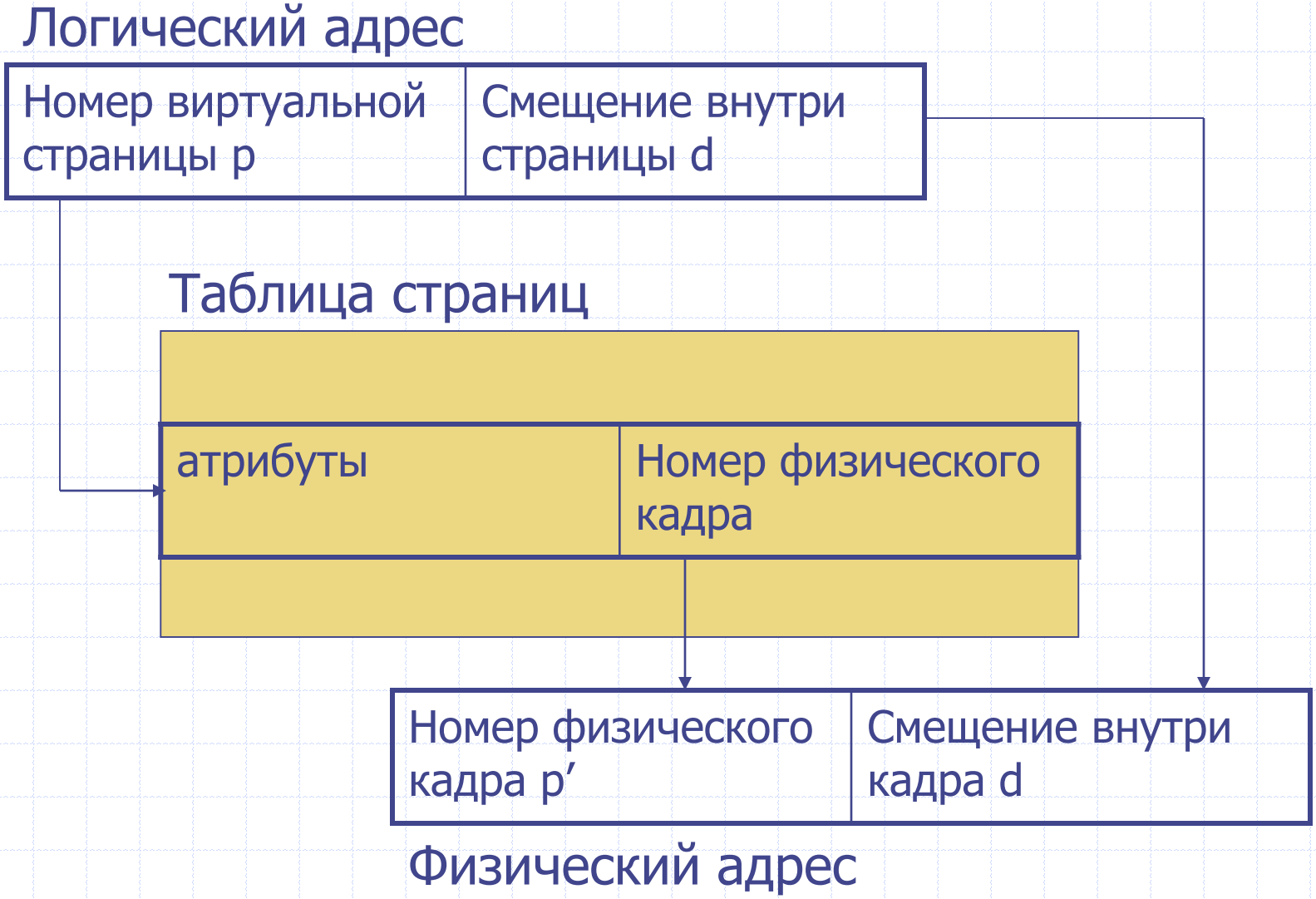
Связь логического и физического адресов

Страничная память

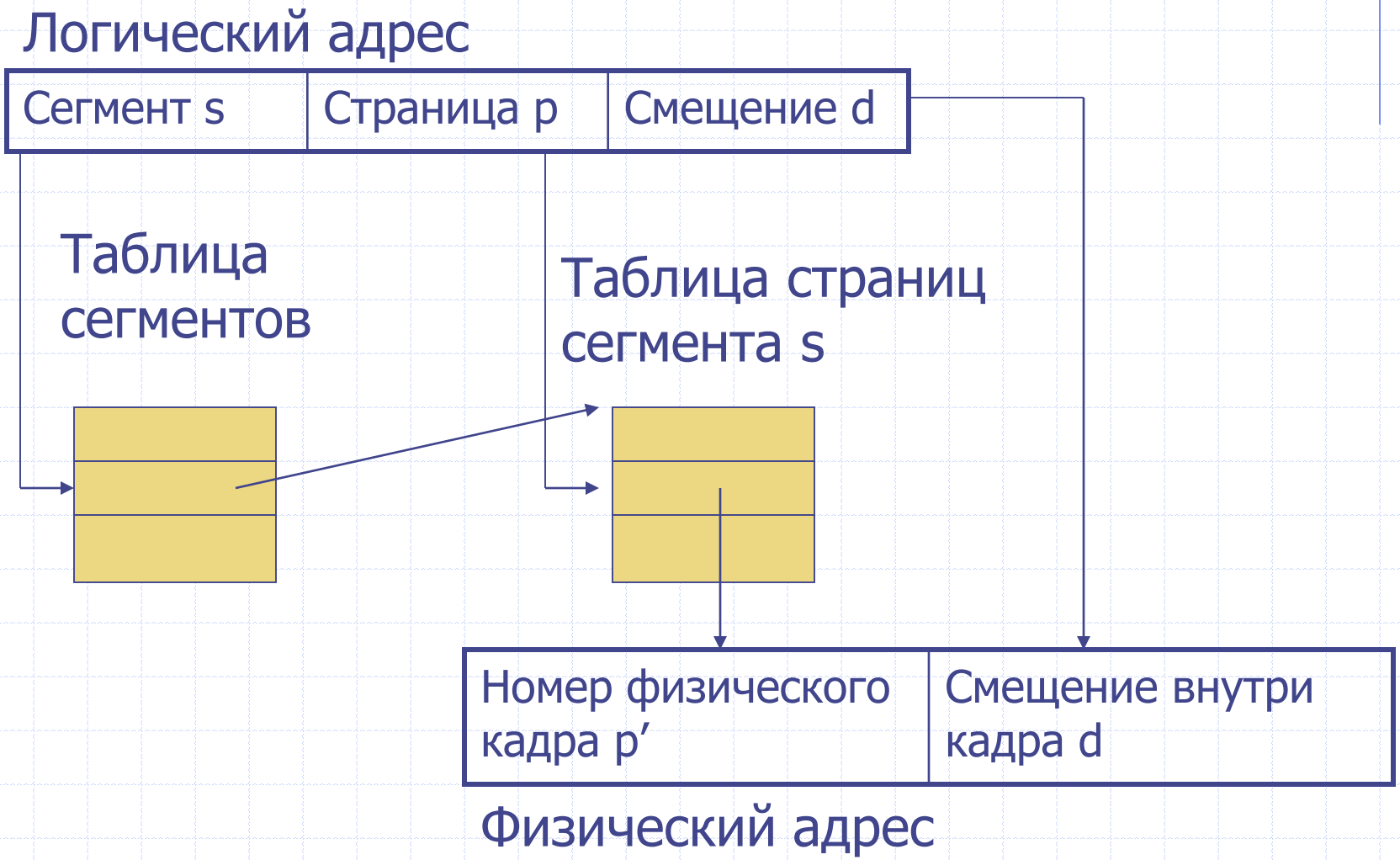
Физическая и логическая память представляются состоящими из блоков одинакового размера, называемых страницы (page) для логической памяти, кадры (frame) для физической.

Размер страниц фиксирован и обычно кратен степени 2. Одна страница соответствует одному кадру.

Связь логического и физического адресов



Странично-сегментная организация памяти



Виртуальная память

- Память процесса разбивается на части (например страницы)
- Выполняется динамическая трансляция логического адреса в физический
- При отсутствии страницы в ОЗУ она догружается (подкачивается) с жесткого диска.

Виртуальная память

- Программа не ограничена объемом ОЗУ и имеет доступ к всему адресному пространству CPU
- Возможно частое и гибкое перемещение процессов в памяти, что позволяет разместить больше процессов
- Объем ввода-вывода меньше, чем в свопинге
- Обеспечивается контроль доступа к памяти и защита адресных пространств процессов

Структура таблицы страниц

Структура адреса:

(p,d)

где p – номер виртуальной страницы,

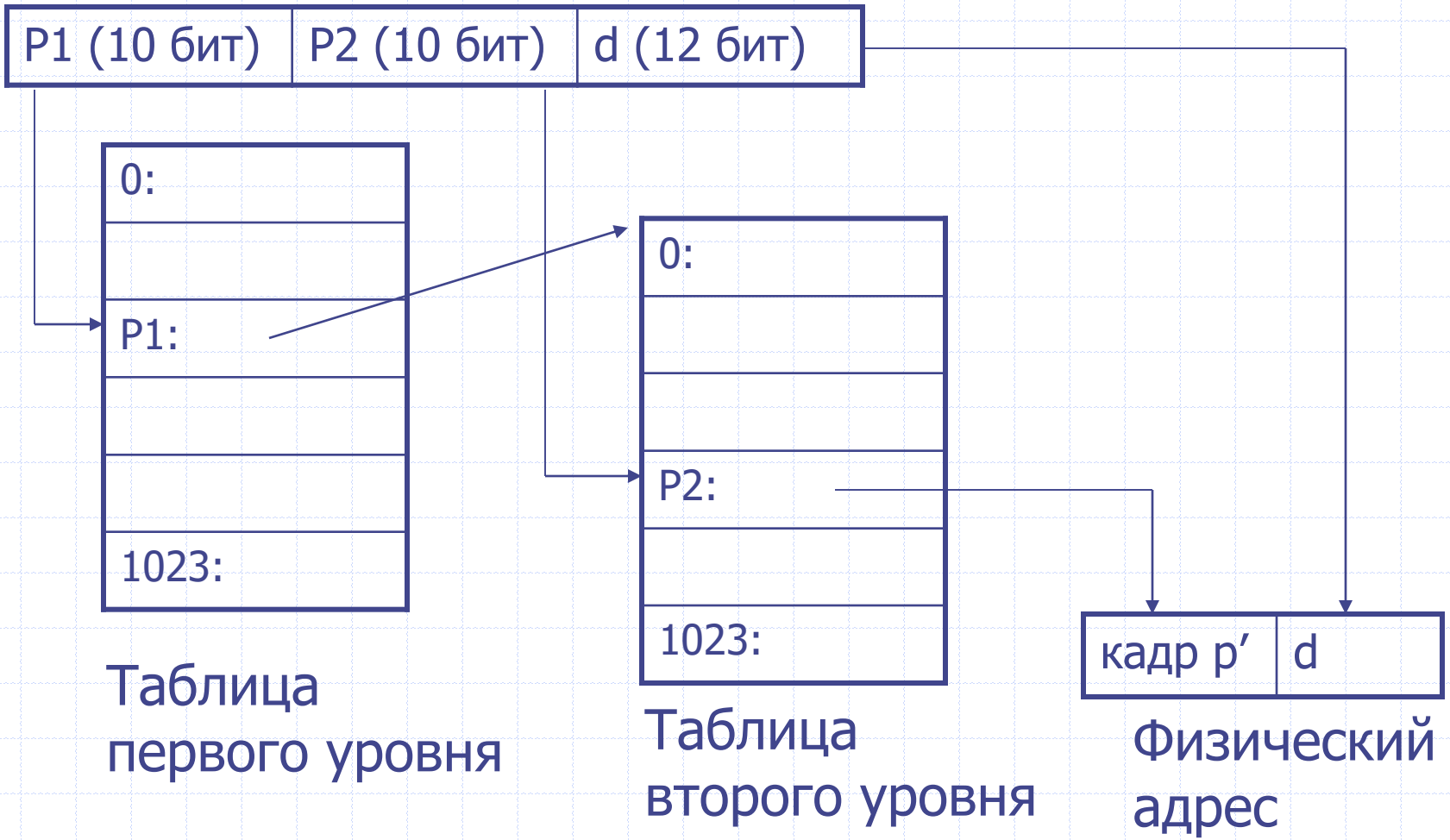
d – смещение

Запись таблицы страниц:

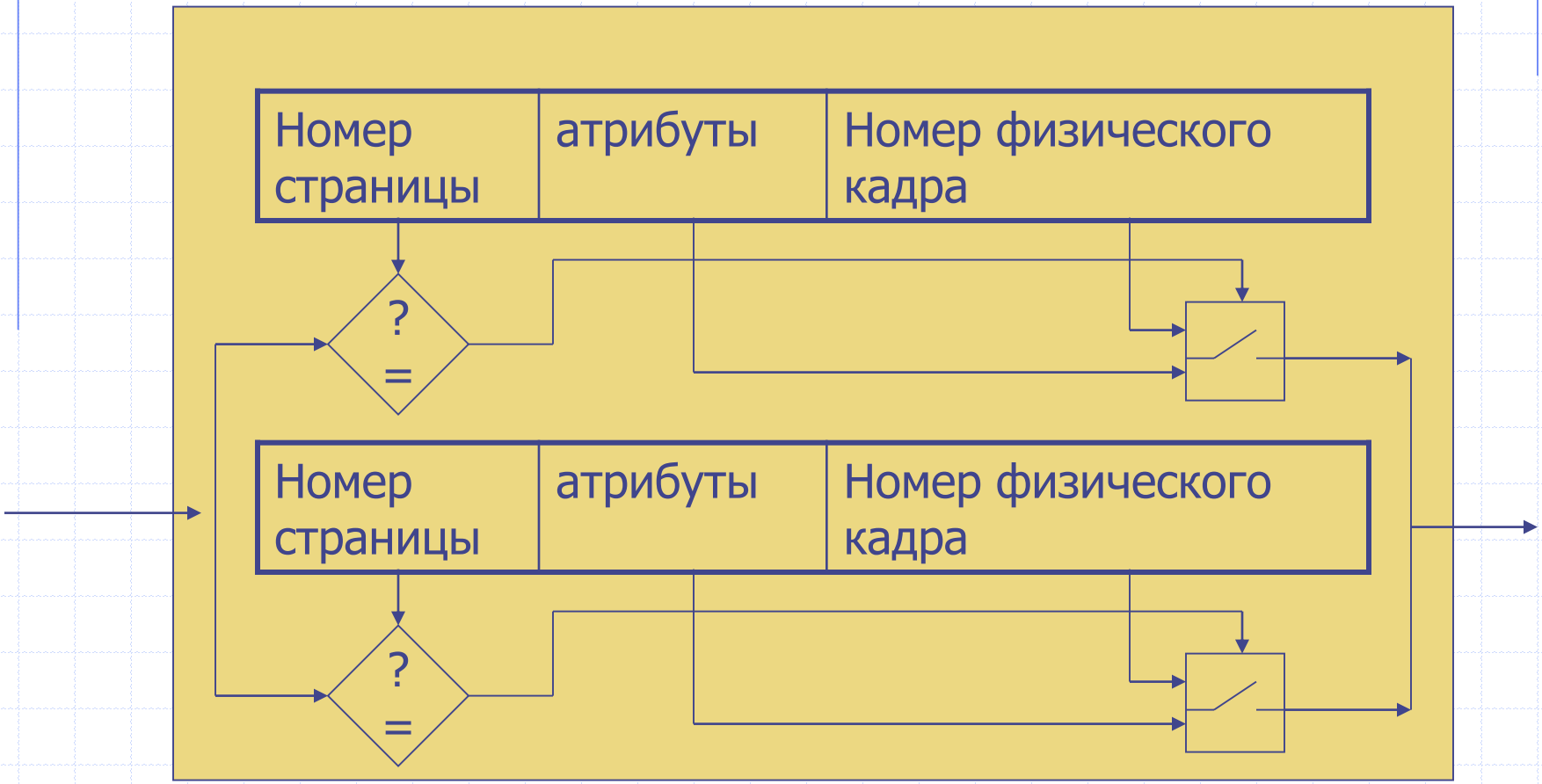
- номер физического кадра
- бит присутствия
- биты защиты
- бит модификации
- …

Многоуровневые таблицы страниц

Логический адрес



Translation Lookaside Buffer (TLB)



Инвертированная таблица страниц

- Записи соответствуют физическим кадрам
- Достаточно одной таблицы на все процессы
- Усложнён поиск необходимой записи – можно применить хеширование

Реализация систем виртуальной памяти

Прерывания и виртуальная память

Пользовательский процесс

...

R1← *p:d

...

ОС

...

page_fault:

...

Таблица страниц

№		Frame
0	☑	0x23E0
...		
P	☐	-------
...		

Таблица прерываний

IRQ	Обработчик
...	
page fault	0x12345678
...	

Обработка страничного исключения

1. Обработка прерывания
2. Поиск свободного / замещение занятого кадра
3. Подкачка страницы с диска
4. Возобновление процесса

Замещение страницы

1. Выбрать (занятый) кадр в ОЗУ
2. Скопировать его содержание на диск
3. Считать содержимое требуемой страницы с диска в выбранный кадр
4. Модифицировать таблицы страниц
5. Продолжить выполнение процесса

Стратегии управления памятью

- Стратегия очистки/ cleaning policy
  когда сохранять страницы на диск
- Стратегия выборки / fetch policy
  когда считывать страницу с диска
- Стратегия размещения / placement policy
  где в ОЗУ разместить считываемую страницу
- Стратегия замещения / replacement policy
  какую страницу вытеснить если нет места

Стратегия очистки

Грязная страница / **dirty page** – страница, которая была модифицирована и отличается от версии в страничном файле

Чистая страница / clean page – страница, точная копия которой находится в страничном файле. Не требует сохранения при замещении.

Очистка страницы / **cleaning** – сохранение в страничный файл

Стратегия очистки

Когда проводить очистку?

- В момент замещения

  хорошо когда у нас полнодуплексный канал во внешнюю память

- Заранее / Precleaning
  - Необходимо определять страницы, которые могут быть скоро замещены
  - Если очистить слишком рано, страница может успеть «испачкаться» ещё раз

Стратегия выборки

- **По запросу / demand paging** – страница считывается когда к ней происходит обращение
- С упреждением:
  - **Loader paging –** при запуске процесса загружаем его целиком
  - **Anticipatory paging** – выборка нескольких последовательных страниц
  - **Swap prefetch** – выборка с прогнозированием кандидатов на загрузку (не обязательно последовательно)

Виды алгоритмов замещения

- Глобальные алгоритмы

  При замещении может быть использована страница любого процесса

- Локальные алгоритмы

  Вытесняются только страницы того же процесса

  резидентное множество процесса – множество физических кадров, выделенных процессу

Строка обращений

Строка обращений / reference string – последовательность обращений процесса к памяти

- Запоминаются только номера страниц
- Последовательные обращения к одной странице объединяются в одно

Аппаратные средства фиксации обращений

- **Флаг ссылки / reference bit** – устанавливается при любом обращении к странице
- **Флаг изменения / modify bit** – устанавливается при записи на страницу
- Устанавливаются процессором автоматически
- Сбрасываются ОС после проверки состояния

Алгоритм FIFO

Вытесняется «старейшая» страница:

- создаётся очередь страниц
- при загрузке страница помещается в конец очереди
- для замещения выбирается страница из начала очереди

Аномалия Билэди (Belady)

Вариант 1: доступно 3 кадра

Строка обращения	**0**	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**
Самая новая	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
Самая старая			0	1	2	3	0	0	0	1	4	4
Page fault	✓	✓	✓	✓	✓	✓	✓			✓	✓	

Всего 9 страничных исключений

Аномалия Билэди (Belady)

Вариант 2: доступно 4 кадра

Строка обращения	**0**	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**
Самая новая	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	3
			0	1	1	1	2	3	4	0	1	2
Самая старая				0	0	0	1	2	3	4	0	1
Page fault	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓

Всего 10 страничных исключений

Оптимальный алгоритм / OPT

Стратегия замещения Билэди:

Следует выталкивать страницу, к которой дольше всего не будет происходить обращений

Алгоритм LRU

Выталкивание дольше всего не использовавшейся страницы / Least Recently Used

- Храним двунаправленный список всех кадров
- При каждом обращении к кадру помещаем его в начало списка / O(1)
- Для замещения выбираем кадры из конца списка, новые кадры помещаем в начало списка / O(1)

Алгоритм LRU - пример

Доступно 4 кадра

Строка обращения	**0**	**1**	**2**	**3**	**0**	**1**	**4**	**0**	**1**	**2**	**3**	**4**
Недавно				3	0	1	4	0	1	2	3	4
			2	2	3	0	1	4	0	1	2	3
		1	1	1	2	3	0	1	4	0	1	2
Давно	0	0	0	0	1	2	3	3	3	4	0	1
Page fault	✓	✓	✓	✓			✓			✓	✓	✓

Всего 8 страничных исключений

Алгоритм LRU – альтернативная реализация

- Имеется аппаратный 64-битный счётчик, который увеличивается при выполнении каждой инструкции
- При каждом обращении к странице значение счётчика сохраняется в таблице страниц
- Для замещения выбирается кадр с минимальным значением счётчика

Алгортим LRU. Выводы.

- Достоинства
  - Не более чем в N раз больше страничных исключений, чем у OPT
- Недостатки
  - Много исключений при последовательном сканировании N+1 страницы
  - Много исключений при загрузке ОС

    Можно попытаться обнаружить эту ситуацию и переключится на другой алгоритм, например Most Recently Used (MRU)
  - Не учитывает частоту использования страницы

N – размер резидентного множества

Стековые алгоритмы

Для фиксированной строки обращений резидентное множество для N страниц является подмножеством резидентного множества для N+1 страницы.

- не подвержены аномалии Билэди

- Оптимальный алгоритм
- LRU

Алгоритм NFU

Выталкивание не часто используемой страницы / Not Frequently Used

- Для каждого кадра храним счётчик, начальное значение =0
- По прерыванию по времени сканируем флаги обращений и там где он установлен увеличиваем счётчик, флаг сбрасываем
- Для замещения выбираем страницу с минимальным значением счётчика

Алгоритм Aging

Усовершенствованный NFU

- Для каждого кадра храним счётчик, начальное значение =0
- По прерыванию по времени:
  - сдвигаем значение всех счетчиков на 1 вправо
  - сканируем флаги обращений и там где он установлен увеличиваем счётчик, флаг сбрасываем
- Для замещения выбираем страницу с минимальным значением счётчика

Not Recently Used / NRU

- По прерыванию таймера сбрасываются биты ссылок
- При выборе кадра на замещение:
  - Все кадры разбиваются на 4 класса:
    1. reference = false, modify = false
    2. reference = false, modify = true
    3. reference = true, modify = false
    4. reference = true, modify =true
  - Выбирается случайная страница из непустого класса с наименьшим номером

Алгоритм Second Chance

- Действуем как в FIFO
- При выборе страницы на замещение проверяем флаг ссылки:
  - Если сброшен – замещаем
  - Если установлен:
    - сбрасываем флаг и помещаем страницу в начало очереди
    - проверяем следующую страницу в конце очереди

Алгоритм Clock

Second Chance с кольцевым буфером

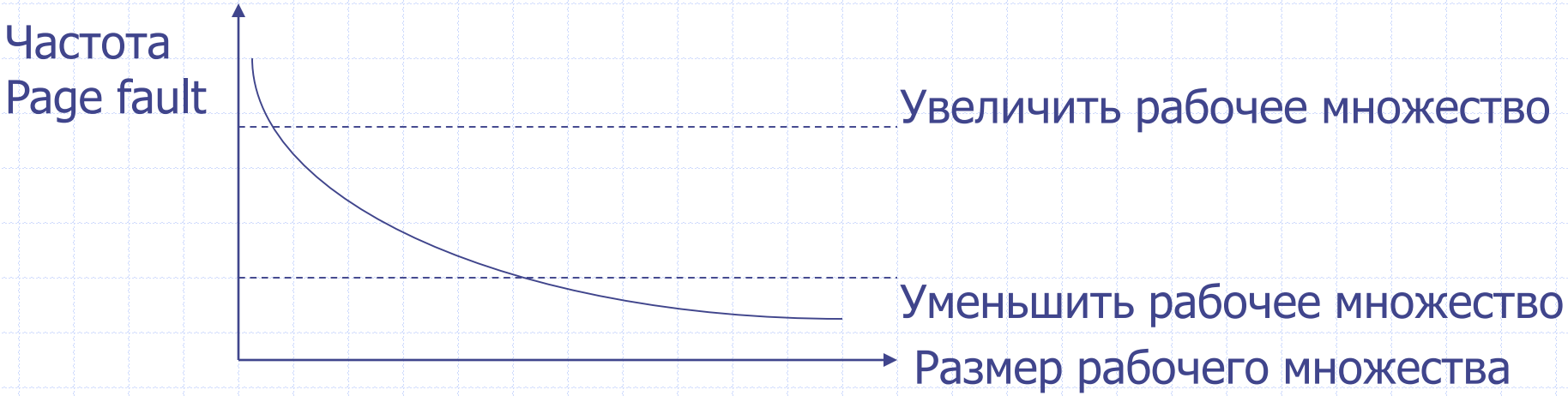
- Организуем кольцевой буфер кадров
- При замещении анализируем флаг ссылки текущего кадра:
  - Если сброшен – замещаем
  - Если установлен – сбрасываем и сдвигаемся к следующему кадру

Трешинг / Trashing

Высокая частота страничных исключений (trashing) вызывает снижение производительности на порядки

Для уменьшения эффекта от трешинга желательно применять локальные алгоритмы.

- Большое рабочее множество – мало памяти для других процессов
- Маленькое рабочее множество – возможен трешинг.



Модель рабочего множества

Рабочее множество процесса $W(t,\tau)$ – набор информации, используемый процессом в интервале времени $(t-\tau,t)$

Исходя из приницпа локальности

$$W(t,\tau) \approx W(t+\tau,\tau)$$

Нужно стараться чтобы резидентное множество = рабочее множество

Деннинг / Denning 1968

Алгоритм WSClock

- Отслеживаем текущее число прерываний по времени
- По каждому прерыванию записываем его номер в качестве времени последнего доступа ко всем кадрам у которых установлен бит доступа
- Как кандидаты на замещение рассматриваются кадры, которые старее τ
- Сначала пытаемся использовать чистые страницы, если их нет – грязные
- Если встречается грязная старая страница, она очищается (но не больше лимита)
- Список страниц для просмотра замкнут в кольцо

Причины развития алгоритмов замещения

- Увеличение размера ОЗУ
  - Снижает эффективность алгоритмов, периодически просматривающих все таблицы страниц
- Увеличение числа уровней в иерархии памяти
  - Нужно стараться работать в пределах кэш-памяти CPU
- Ослабление локальности за счёт использования ООП, сложных структур данных, сборки мусора
- Объединение систем виртуальной памяти и кэширования диска

Локализация страниц в памяти

Если страница используется для чтения данных с использованием механизма DMA она не может быть замещена.

- Использовать для ввода-вывода специальные страницы в памяти ОС
- Использовать специальный бит локализации в таблице страниц для запрещения замещения
  - опасно «забывать» сбросить этот бит
  - может носить рекомендательный характер (SunOS)

Разделяемая память

Если разместить ссылки на один и тот же кадр в таблицах страниц разных процессов, то его можно использовать для обмена данными

Таблица страниц процесса 1

№	Frame
0	0x23E0
…	
25	0x7518
…	

25:0

Кадр 0x7518

Таблица страниц процесса 2

№	Frame
0	0x23E0
…	
74	0x7518
…	

74:0

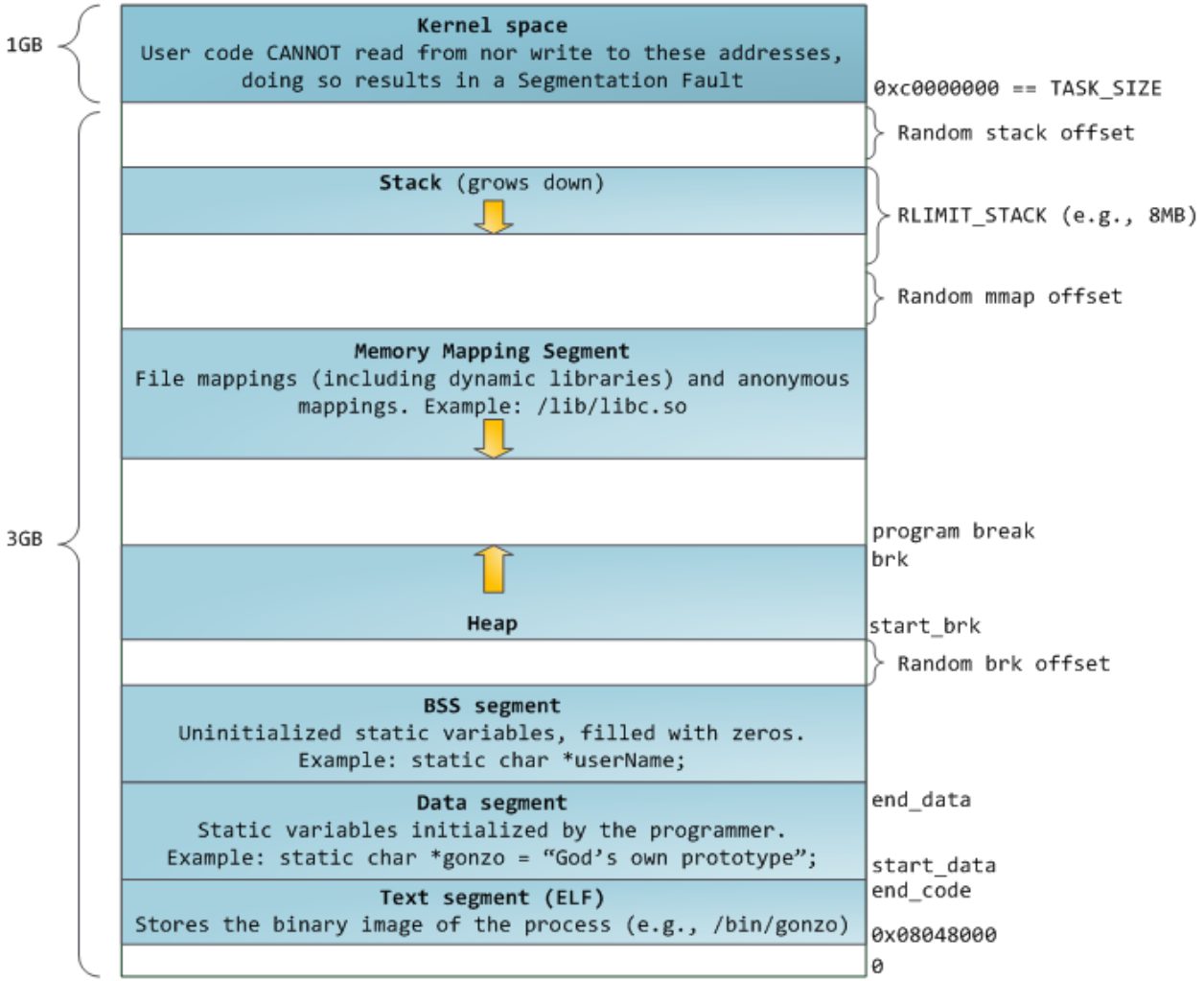
Копирование при записи

- Некоторые фрагменты памяти могут совместно использоваться разными процессами, до тех пор пока они не будут модифицированы
- Модификация возможна, но происходит не часто и затрагивает небольшие регионы памяти
  - Сегменты кода при запуске одного приложения несколько раз

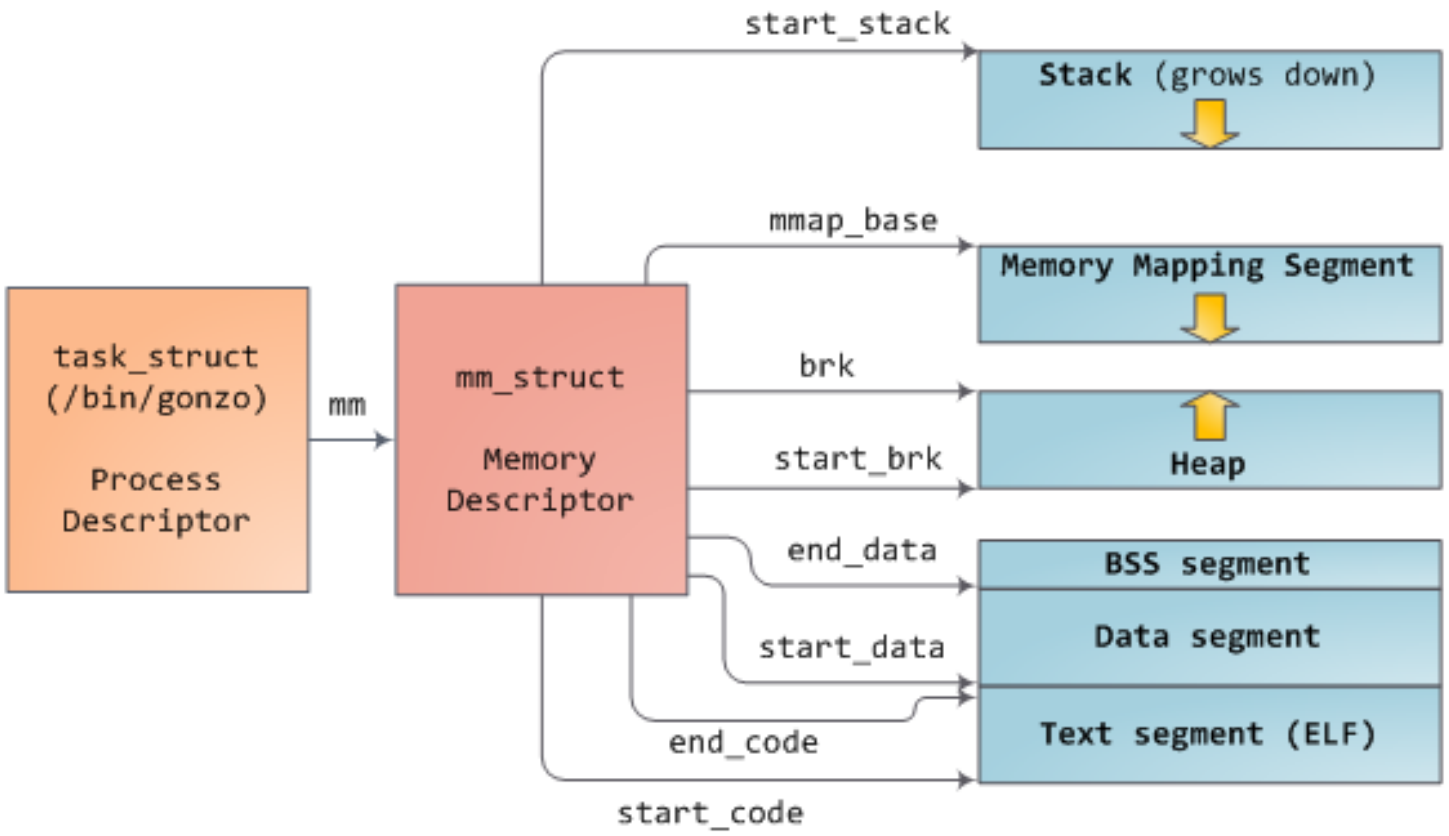
Копирование при записи

- Помечаем страницы специальным битом Copy-On-Write
- При попытке записи на такую страницу:
  - Происходит прерывание
  - Выделяем новый кадр
  - Копируем содержание разделяемого кадра
  - Модифицируем таблицу страниц пишущего процесса так, чтобы она ссылалась на новый кадр и снимаем флаг Copy-On-Write
  - Возобновляем выполнение процесса
  - Другие процессы продолжают работать с разделяемым кадром

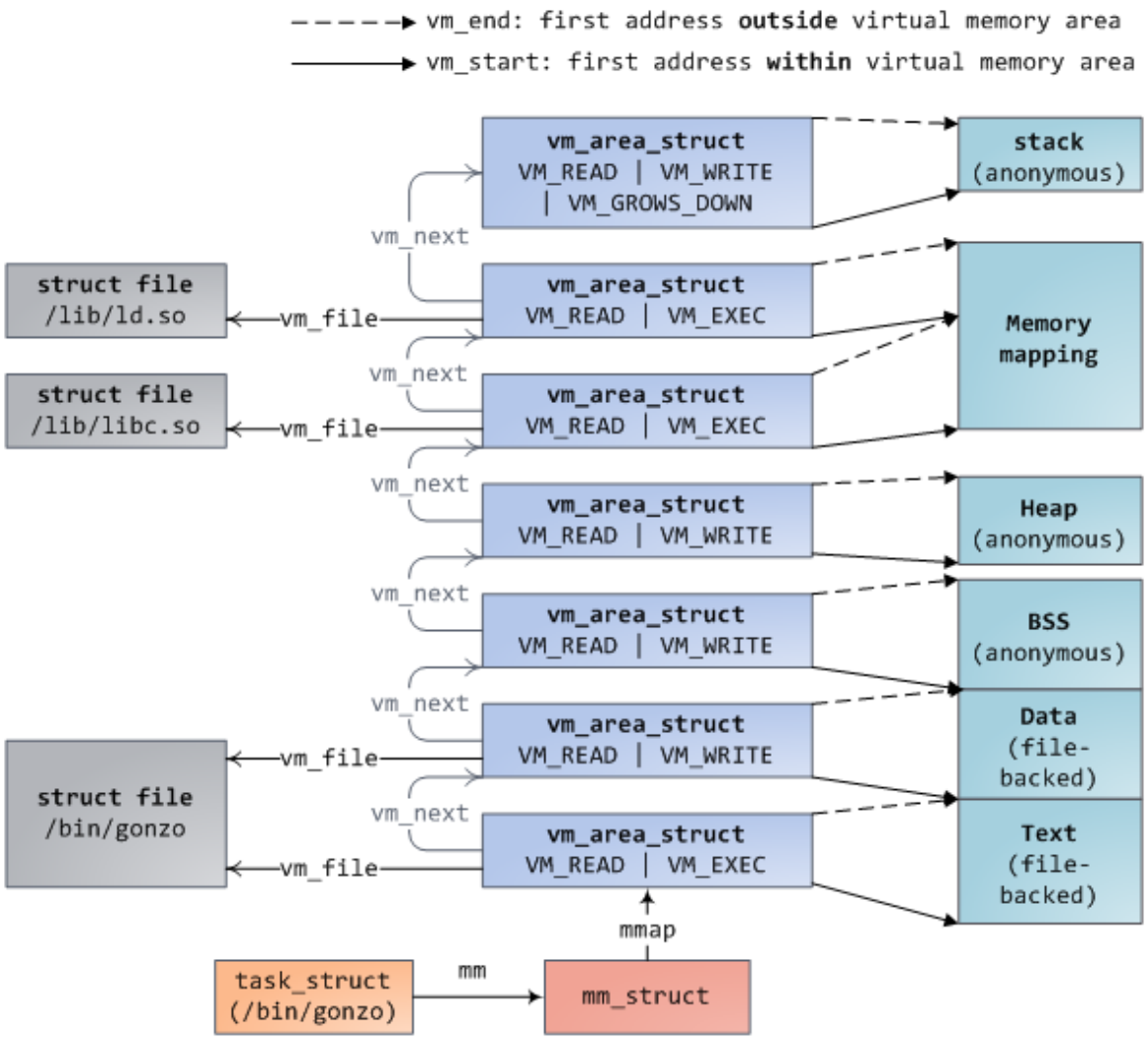
Адресное пространство процесса в Linux 32 бит



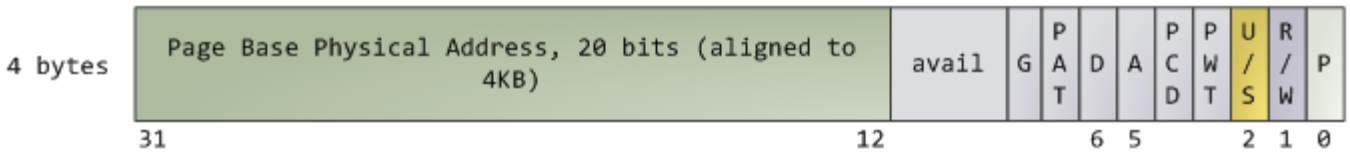
Управление памятью Linux 32

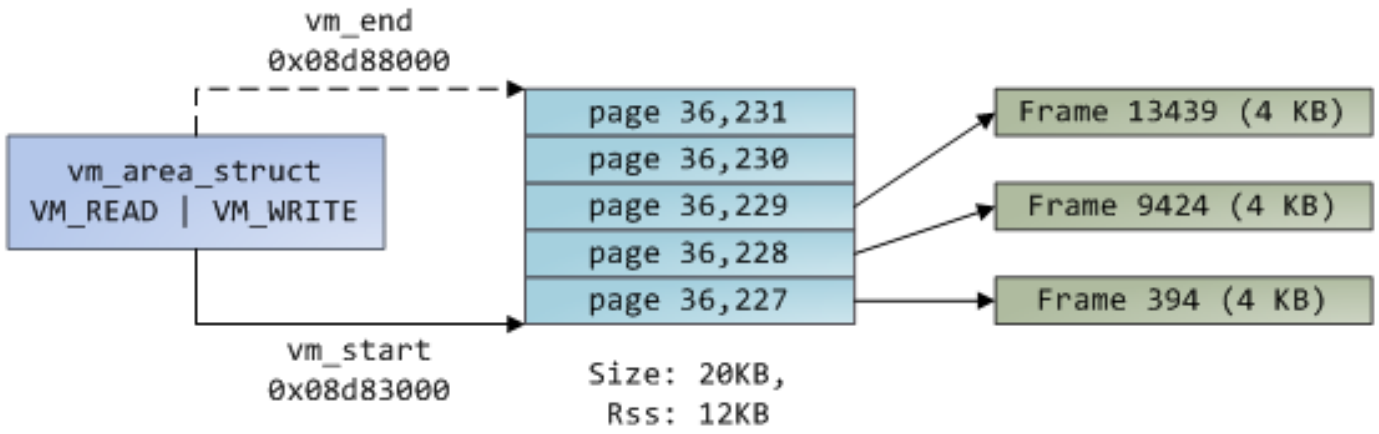


Управление памятью Linux 32



Управление памятью Linux 32



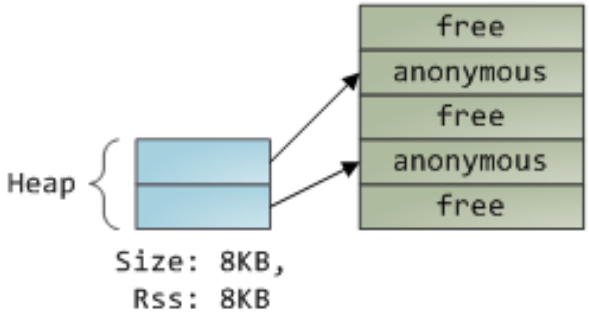


Buddy memory allocation

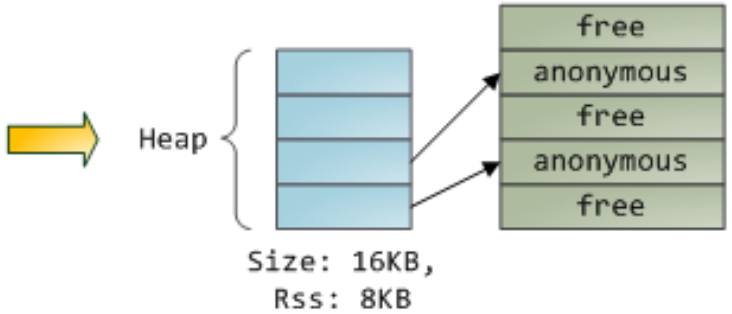
Step	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K
1	2^4															
2.1	2^3								2^3							
2.2	2^2				2^2				2^3							
2.3	2^1		2^1		2^2				2^3							
2.4	2^0	2^0	2^1		2^2				2^3							
2.5	A: 2^0	2^0	2^1		2^2				2^3							
3	A: 2^0	2^0	B: 2^1		2^2				2^3							
4	A: 2^0	C: 2^0	B: 2^1		2^2				2^3							
5.1	A: 2^0	C: 2^0	B: 2^1		2^1		2^1		2^3							
5.2	A: 2^0	C: 2^0	B: 2^1		D: 2^1		2^1		2^3							
6	A: 2^0	C: 2^0	2^1		D: 2^1		2^1		2^3							
7.1	A: 2^0	C: 2^0	2^1		2^1		2^1		2^3							
7.2	A: 2^0	C: 2^0	2^1		2^2				2^3							
8	2^0	C: 2^0	2^1		2^2				2^3							
9.1	2^0	2^0	2^1		2^2				2^3							
9.2	2^1		2^1		2^2				2^3							
9.3	2^2				2^2				2^3							
9.4	2^3								2^3							
9.5	2^4															

Управление памятью Linux 32

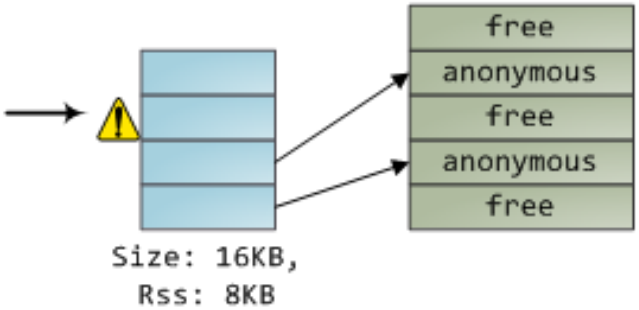
1. Program calls brk() to grow its heap



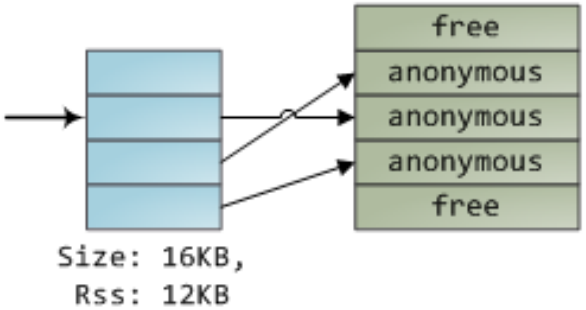
2. brk() enlarges heap VMA.
New pages are **not** mapped onto physical memory.



3. Program tries to access new memory.
Processor page faults.

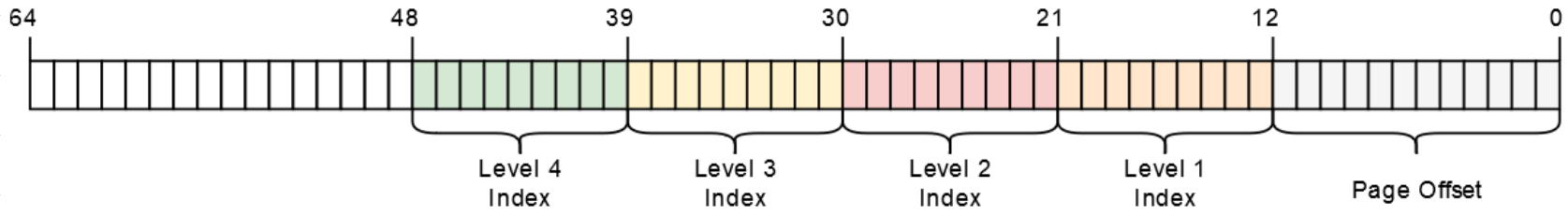


4. Kernel assigns page frame to process, creates PTE, resumes execution. Program is unaware anything happened.



Страничная организация памяти x86_64

- Используются 48 бит виртуального адреса = 256 ТВ
- Размер страницы 4 КБ
- Размер записи таблицы страниц 8 байт
- 4-х уровневая таблица страниц
- Размер таблицы на всех уровнях 512 записей
- Формат виртуального адреса:



Пример трансляции адреса

Виртуальный адрес:

00000000 00000000 000 000 001 000 000 000 111 111 111 001 111 111 0101 1100 1110

- Расширение знака
- Уровень 4, Индекс 1
- Уровень 3, Индекс 0
- Уровень 2, Индекс 511
- Уровень 1, Индекс 127
- Смещение, 0x5CE

Физическая память

0x01000
0x02000
0x03000
0x04000
0x05000
0x06000
0x07000
0x08000
0x09000
0x10000
0x11000
0x12000

Регистр CR3: 0x02000

Индекс	Кадр	Флаги
0	-	-
1	0x06000	r/w
...	...	...

Индекс	Кадр	Флаги
...	...	...
511	0x10000	r/w

Индекс	Кадр	Флаги
0	0x04000	r/w
...	...	...

Индекс	Кадр	Флаги
...	...	...
127	0x11000	r
...	...	...

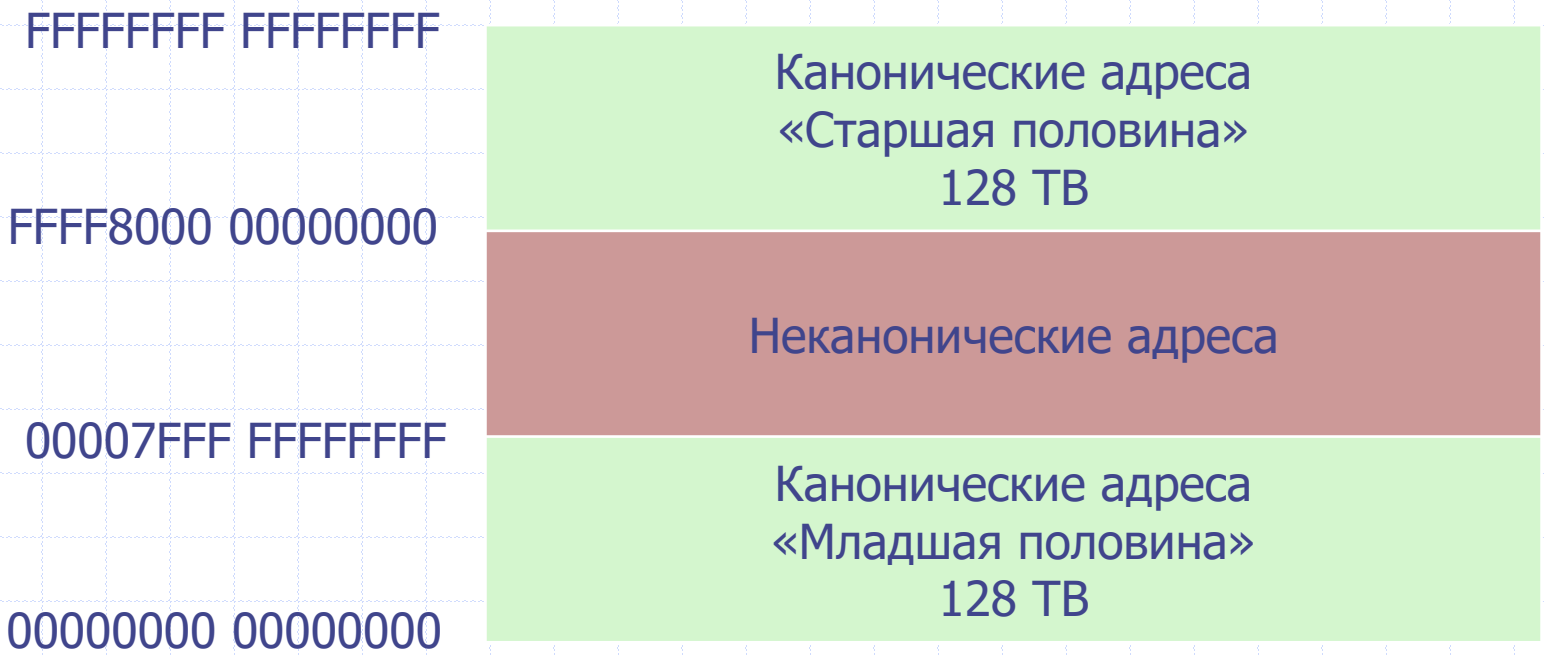
Физический адрес 0x...01105CE

Формат записи таблицы страниц

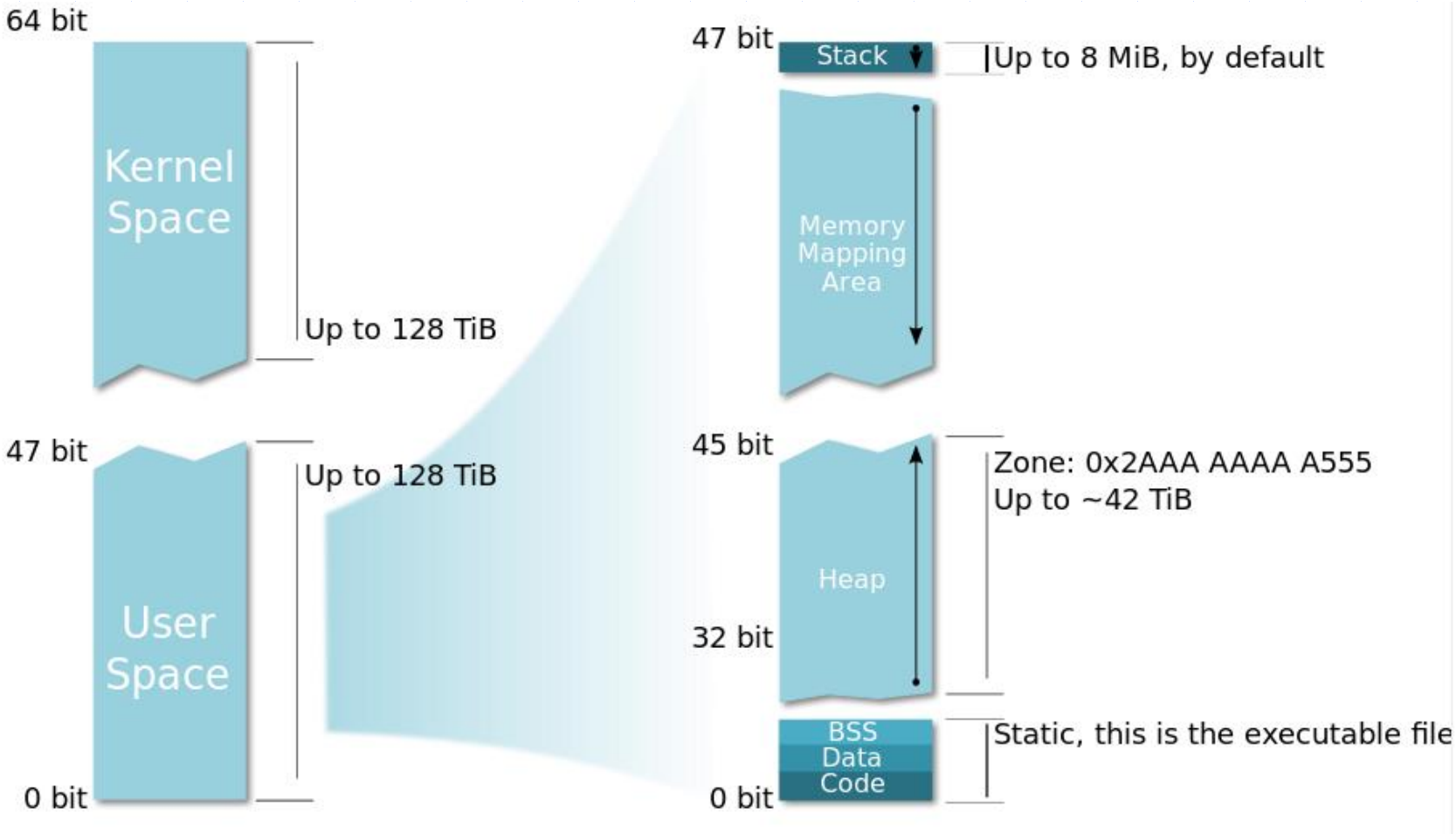
Биты	Название	Значение
0	present	Страница находится в памяти
1	writable	Разрешена запись
2	user accessible	Разрешён доступ из пользовательского режима (если сброшен, только из режима ядра)
3	write through caching	Запись идёт прямо в память
4	disable cache	Запрещено использование кэша
5	accessed	Устанавливается при любом обращении к памяти
6	dirty	Устанавливается при записи на страницу
7	huge page/null	Должен быть 0 на уровнях P1 и P4, создаёт 1GiB страницу на уровне P3, создаёт 2MiB страницу если использован на уровне P2
8	global	Страница не вытесняется из кэша при переключении адресных пространств
9-11	available	Доступно для любого использования ОС
12-51	physical address	Адрес кадра памяти или таблицы следующего уровня
52-62	available	Доступно для любого использования ОС
63	no execute	Запрет выполнения кода на странице

Канонические адреса

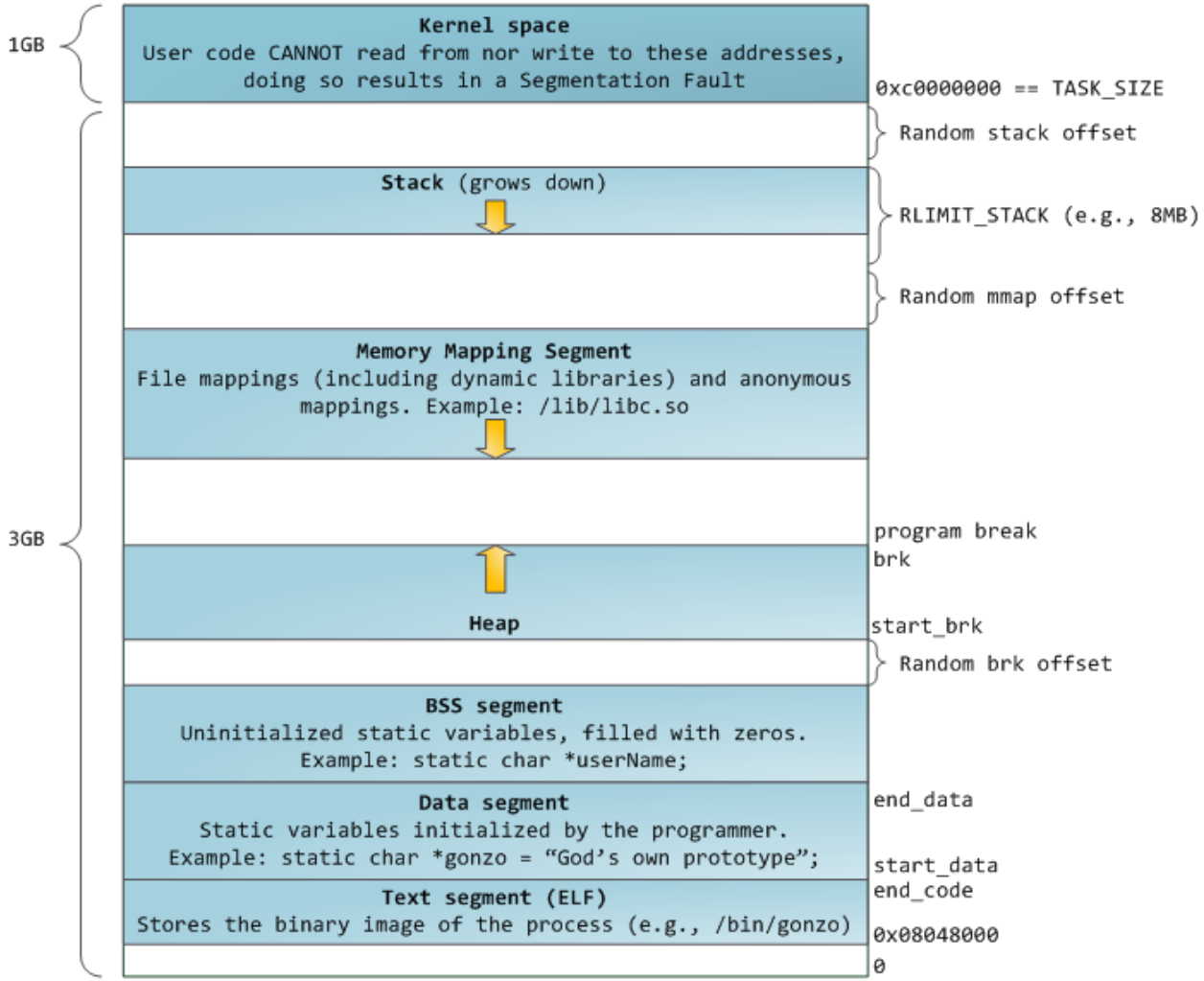
- Неиспользуемые старшие 16 бит адреса должны быть заполнены значением, равным 47-му биту – такие адреса называются каноническими.
- Обращение по не каноническому адресу вызывает исключение.

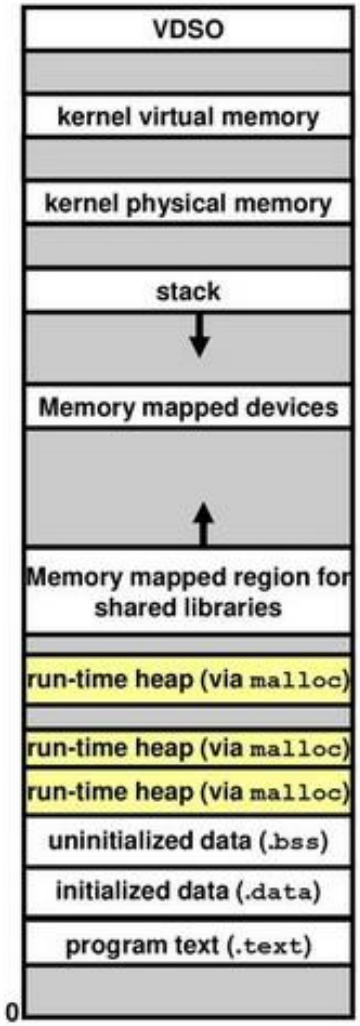


Использование 64-битного адресного пространства в Linux

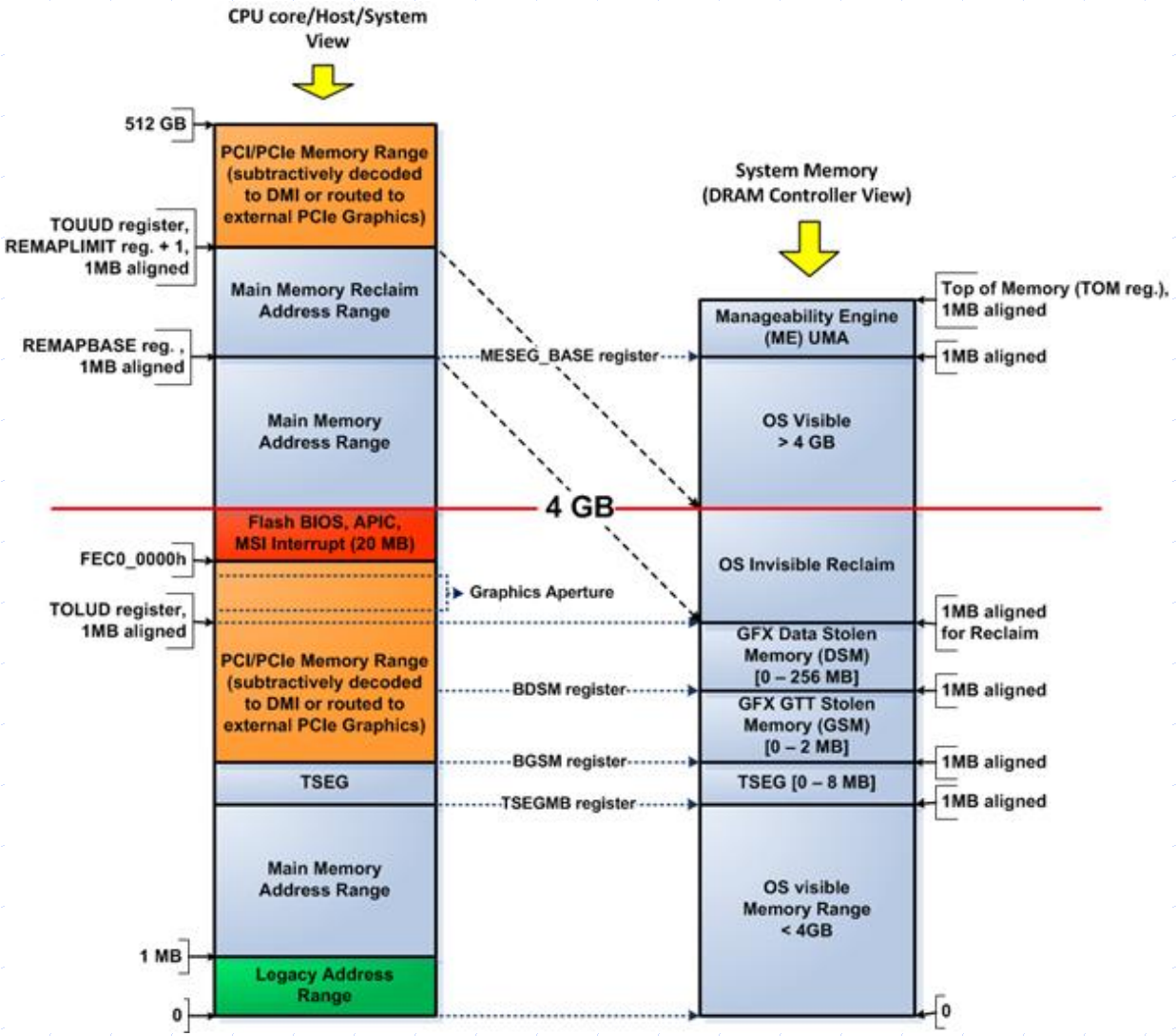


Адресное пространство процесса в Linux





Трансляция адресов на уровне чипсета



Переполнение буфера

```
int login()
{
  char name[10];
  scanf("%s",name);
  …
```

Кадр функции login()												**Кадры других функций**							
501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520
name										addr									

Переполнение буфера

```
int login()
{
  char name[10];
  scanf("%s",name);
  …
```

Кадр функции login()												**Кадры других функций**							
501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520
name										addr									
s	o	m	e	v	e	r	y	l	o	n	g	s	t	r	i	n	g		

Переполнение буфера

```
int login()
{
  char name[10];
  scanf("%s",name);
  …
```

Кадр функции login()												Кадры других функций							
501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520
name										addr									
X	a	-	X	a	-	X	a	X	a	517		nop	nop	nop	nop	nop	nop	code	

Возврат в libc

```
int login()
{
  char name[10];
  scanf("%s",name);
  …
```

Кадр функции login()												Кадры других функций							
501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520
name										addr									
X	a	-	X	a	-	X	a	X	a	exec()		515		format c:\ \yes					

```
Стандартная библиотека:
int exec(char *arg)
{
  …
```