Operating Systems

Physical and Virtual Memory

Me

February 20, 2016

План

1. Расположение физической памяти.
2. Понятие процесса и виртуальная память.
3. Сегментация памяти как способ защиты.
4. Paging и Page Fault.
5. Аллокация непоследовательных страниц.

Как выглядит физическая память?

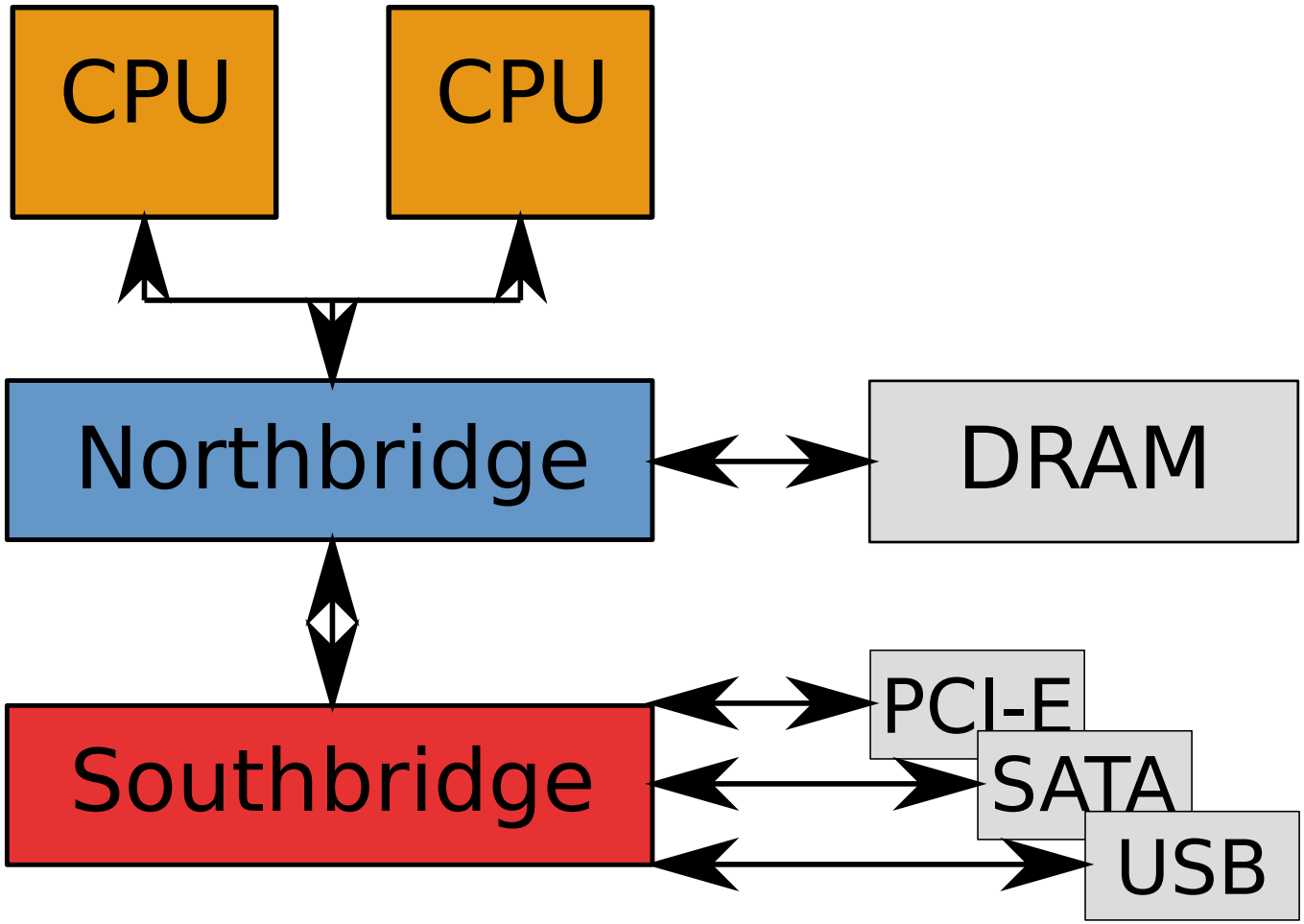

Figure : Classical UMA Architecture

Как выглядит физическая память?

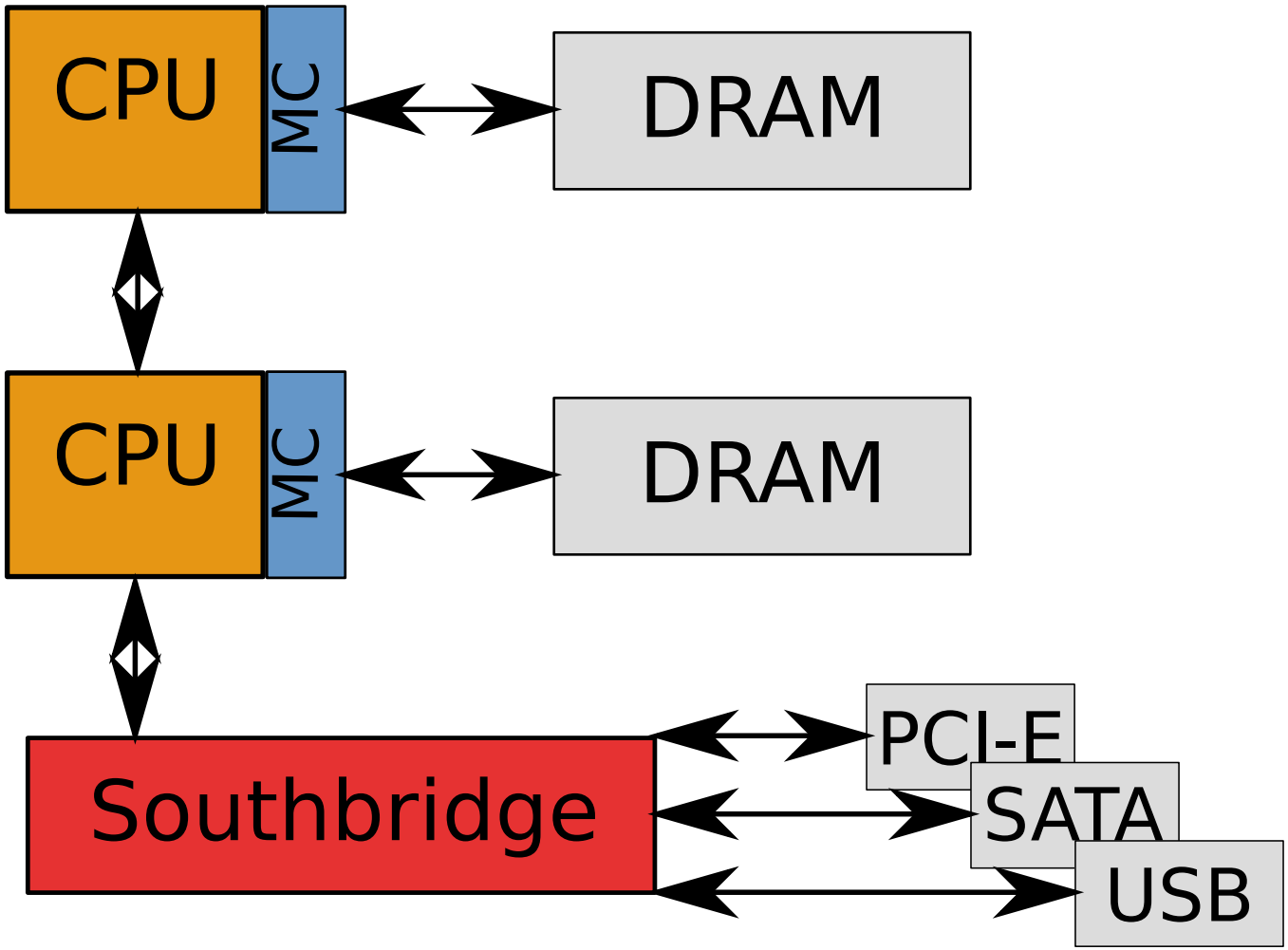

Figure : NUMA Architecture

Как выглядит физическая память?

Память не однородна:

- адреса могут вообще быть не доступны - память не отображена никуда;
- адреса могут быть отображены на устройства - особые правила доступа/кеширования;
- разные адреса могут иметь разное время доступа с разных CPU (NUMA);

Как выглядит физическая память?

Память не однородна:

- адреса могут вообще быть не доступны - память не отображена никуда;
- адреса могут быть отображены на устройства - особые правила доступа/кеширования;
- разные адреса могут иметь разное время доступа с разных CPU (NUMA);

Нужна карта памяти!

Где взять карту памяти?

- из документации чипсета

Где взять карту памяти?

- из документации чипсета
- из device tree
  - кто-то все равно должен взять документацию чипсета и описать память в нужном формате и передать загрузчику

Где взять карту памяти?

- из документации чипсета
- из device tree
  - кто-то все равно должен взять документацию чипсета и описать память в нужном формате и передать загрузчику
- BIOS/UEFI или их аналог
  - BIOS int $0x15, функции 0xe820 или 0xe801
  - UEFI GetMemoryMap

Где взять карту памяти?

- из документации чипсета
- из device tree
  - кто-то все равно должен взять документацию чипсета и описать память в нужном формате и передать загрузчику
- BIOS/UEFI или их аналог
  - BIOS int $0x15, функции 0xe820 или 0xe801
  - UEFI GetMemoryMap
- спросить у загрузчика (где ее берет загрузчик - не наше дело)

Типичная карта памяти

Такие регионы памяти, например, может сообщать QEMU через multiboot загрузчик:

- `0x00000000-0x0009fbff, Available`
- `0x0009fc00-0x0009ffff, Reserved`
- `0x000f0000-0x000fffff, Reserved`
- `0x00100000-0x07ffdfff, Available`
- `0x07ffe000-0x07ffffff, Reserved`
- `0xfffc0000-0xffffffff, Reserved`

Понятие процесса

Процесс - контейнер ресурсов ОС:

Понятие процесса

Процесс - контейнер ресурсов ОС:

- ресурсы в ОС привязаны к процессам
  - память (свое адресное пространство у процессов)
  - файловые дескрипторы
  - другие ресурсы (сокеты, различные IPC и тд)

Понятие процесса

Процесс - контейнер ресурсов ОС:

- ресурсы в ОС привязаны к процессам
  - память (свое адресное пространство у процессов)
  - файловые дескрипторы
  - другие ресурсы (сокеты, различные IPC и тд)
- процессы изолированы друг от друга
  - на сколько это позволяет аппаратное обеспечение (далее просто HW)
  - некоторые процессы могут разделять общие ресурсы намеренно

Адресное пространство процесса

Адресное пространство процесса (виртуальное адресное пространство, VA) - это набор адресов доступных процессу для работы (мы называем эти адреса виртуальными):

Адресное пространство процесса

Адресное пространство процесса (виртуальное адресное пространство, VA) - это набор адресов доступных процессу для работы (мы называем эти адреса виртуальными):

- VA отображается на физическую память (далее PA)
  - paging - произвольное отображение
  - "один к одному", если HW не поддерживает трансляцию

Адресное пространство процесса

Адресное пространство процесса (виртуальное адресное пространство, VA) - это набор адресов доступных процессу для работы (мы называем эти адреса виртуальными):

- VA отображается на физическую память (далее PA)
  - paging - произвольное отображение
  - "один к одному", если HW не поддерживает трансляцию
- VA может быть аппаратно защищено
  - сегментирование - попытка обращения в чужой сегмент приводит к ошибке
  - paging устраняет саму возможность обратиться к чужой памяти

Адресное пространство процесса

Адресное пространство процесса (виртуальное адресное пространство, VA) - это набор адресов доступных процессу для работы (мы называем эти адреса виртуальными):

- VA отображается на физическую память (далее PA)
  - paging - произвольное отображение
  - "один к одному", если HW не поддерживает трансляцию
- VA может быть аппаратно защищено
  - сегментирование - попытка обращения в чужой сегмент приводит к ошибке
  - paging устраняет саму возможность обратиться к чужой памяти
- VA может быть неоднородным - в нем могут быть дыры

Адресное пространство процесса

VA процесса - это ресурс, который нужно аллоцировать и освобождать:

Адресное пространство процесса

VA процесса - это ресурс, который нужно аллоцировать и освобождать:

- ОС необходимо делить память между несколькими процессами - не нужно выдавать процессу сразу много памяти, которая скорее всего не будет использована;

Адресное пространство процесса

VA процесса - это ресурс, который нужно аллоцировать и освобождать:

- ОС необходимо делить память между несколькими процессами - не нужно выдавать процессу сразу много памяти, которая скорее всего не будет использована;
- разные части VA используются под разные нужды:
  - они находятся в разных регионах VA (стек растет вниз, его логично положить наверх)
  - могут иметь разные права/привилегии доступа (например, делать стек исполняемым - плохая затея с точки зрения безопасности)

Сегментация на примере x86

Вся память разбивается на сегменты:

- уровень привилегий доступа назначается каждому сегменту отдельно
- сегменты могут перекрываться, т. е. два сегмента с разными привилегиями могут описывать одну и ту же физическую память

Сегментация на примере x86

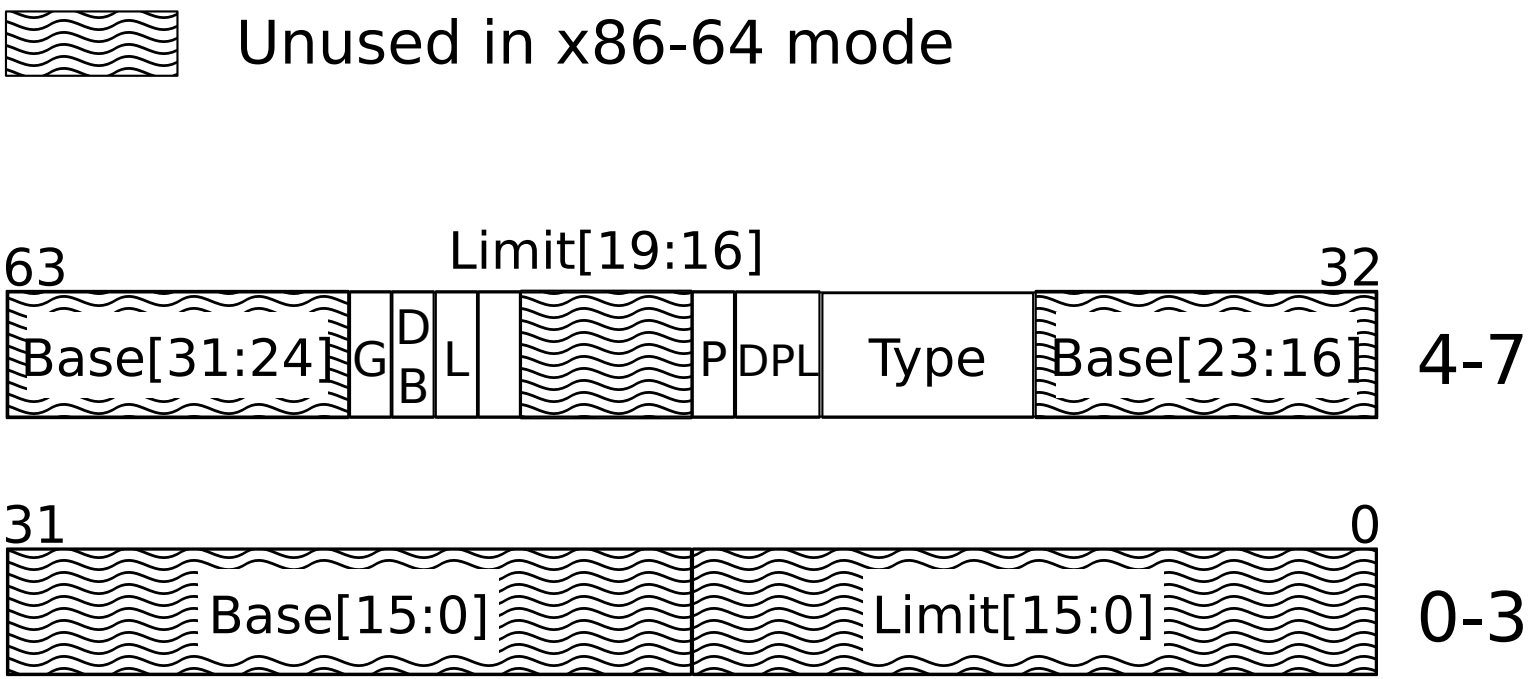

Figure : x86 segment data/code descriptor format

Сегментация на примере x86

Дескрипторы сегментов хранятся в таблицах GDT и LDT:

- GDT предполагается общей для всех процессов (не обязательно)
- LDT своя для каждого процесса (не обязательно)
- при обращении к памяти таблица и номер дескриптора в ней определяются используя селектор сегмента (16-битное значение в CS, SS, DS, ES, FS или GS)

Сегментация на примере x86

Data Selector (DS, ES)

DESCRIPTOR INDEX	T	RPL

Code Selector (CS)

DESCRIPTOR INDEX	T	CPL

Figure : Code and Data Segment Selectors

Сегментация на примере x86

Проверка привилегий:

- при записи селектора в сегментный регистр CPL (из CS) и RPL (то что мы записываем) должны быть меньше или равны DPL (в дескрипторе сегмента, на который мы ссылаемся);
- для селектора стека (SS) используются особые правила - RPL и DPL должны быть равны CPL.

Paging

- отображение VA на PA с гранулярностью в Page (блок память фиксированного размера);
- привилегии/права доступа проверяются на уровне страниц - меньше гранулярность;
- отображение описывается иерархической структурой (Page Table, далее PT) - большая гибкость;
- каждый процесс имеет свою PT - процессы изолированы друг от друга;

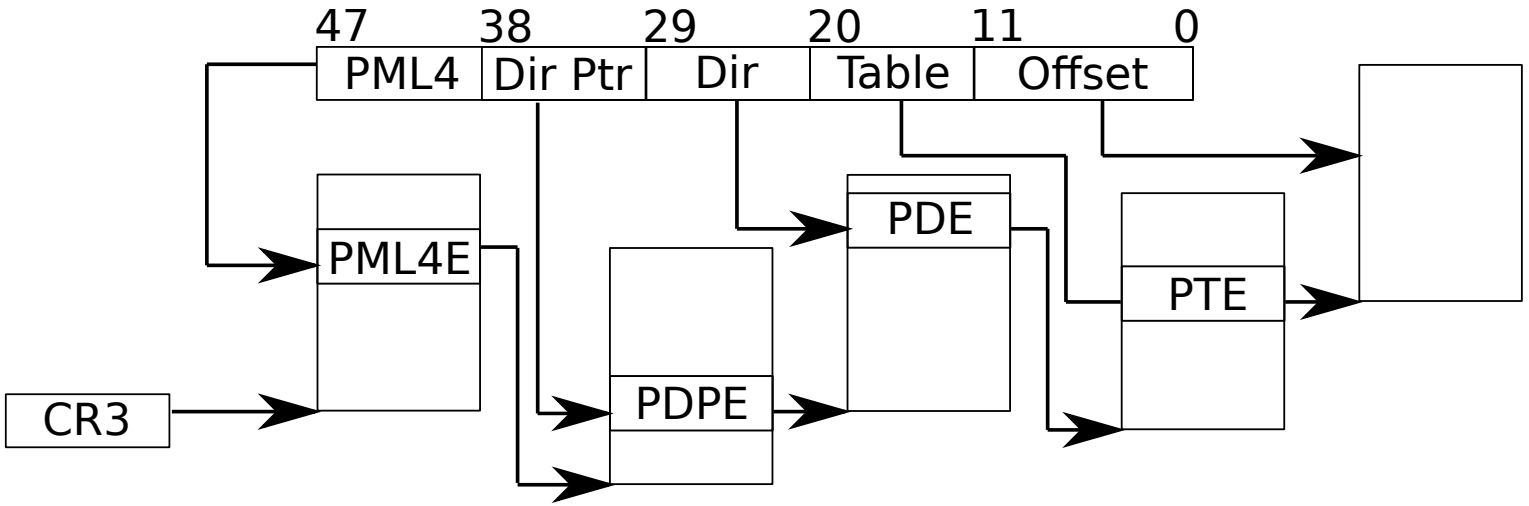

Figure : x86-64 Page Tables

Paging
Права доступа и прочее

- Paging позволяет защитить память от доступа непривилегированного кода;
- Paging позволяет защитить память от доступа привилегированного кода (SMEP/SMAP - защита kernelspace от атак из userspace);
- Paging позволяет запрещать исполнение кода в участке памяти;
- Paging позволяет управлять кешированием участка памяти;
- Paging вытеснил сегментацию (сегментация все еще используется в очень специфичных случаях);

- Обращение к PT при каждом доступе к памяти - очень дорого
  - чем больше памяти - тем больше уровней
  - чем больше уровней - тем дороже трансляция
  - чем больше памяти - тем меленее работа с ней (короче, все плохо)

- Обращение к PT при каждом доступе к памяти - очень дорого
  - чем больше памяти - тем больше уровней
  - чем больше уровней - тем дороже трансляция
  - чем больше памяти - тем меленее работа с ней (короче, все плохо)
- Кеширование ускоряет процесс, но задача слишком специфичная - используем специальный кеш TLB;

- Обращение к PT при каждом доступе к памяти - очень дорого
  - чем больше памяти - тем больше уровней
  - чем больше уровней - тем дороже трансляция
  - чем больше памяти - тем меленее работа с ней (короче, все плохо)
- Кеширование ускоряет процесс, но задача слишком специфичная - используем специальный кеш TLB;
- TLB не прозрачен для программиста - при изменении в PT нужно явно сбросить TLB;

Page Fault

Page Fault происходит если:

- для виртуального адреса отсутствует отображение в физический (в x86 за это отвечает бит Present);
- отображение есть, но не достаточно прав доступа для обращения к памяти;
- произошла попытка записи в страницу только для чтения;
- произошла попытка исполнить кода со страницы не предназначенной для исполнения;
- обнаружена запись некорректного формата в PT;

Page Fault

Page Fault происходит если:

- для виртуального адреса отсутствует отображение в физический (в x86 за это отвечает бит Present);
- отображение есть, но не достаточно прав доступа для обращения к памяти;
- произошла попытка записи в страницу только для чтения;
- произошла попытка исполнить кода со страницы не предназначенной для исполнения;
- обнаружена запись некорректного формата в PT;

Fault (в терминологии x86) - ошибка, которую можно исправить, виртуальный адрес по которому происходит обращение прилагается.

Page Fault
On Demand Allocation

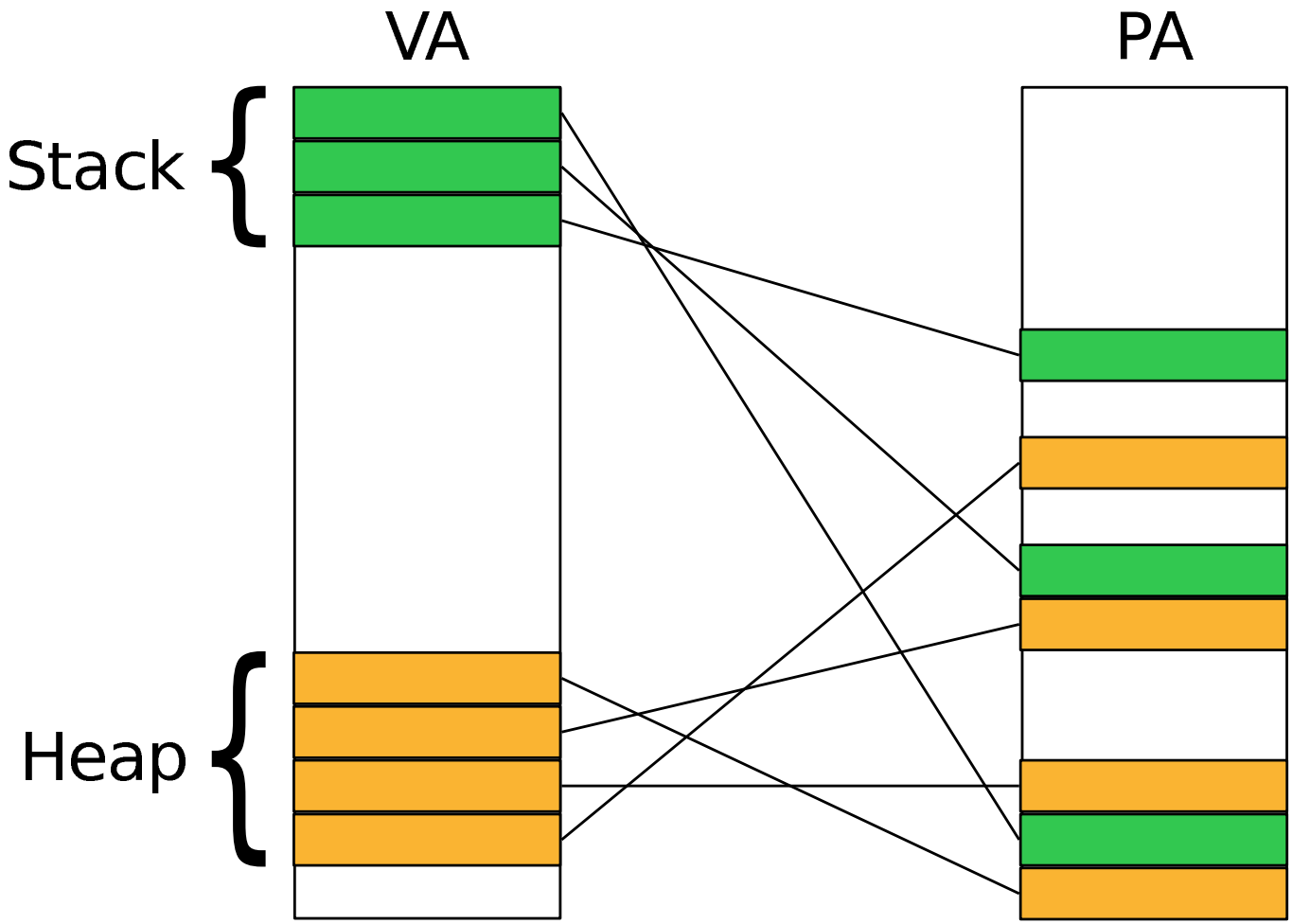

Figure : Отображение VA на PA

Page Fault
On Demand Allocation

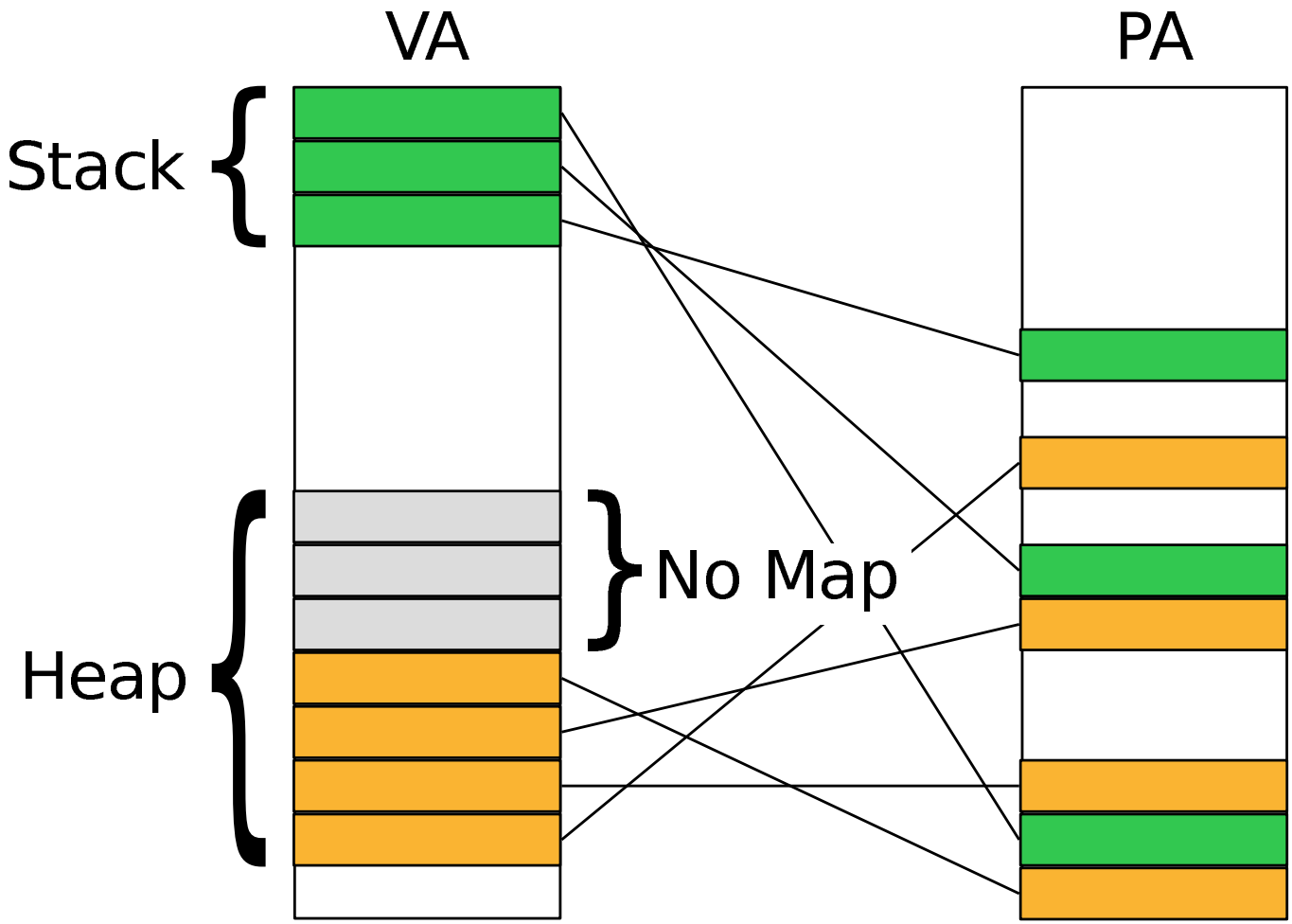

Figure : Процесс увеличивает Heap - страницы не выделяются

Page Fault
On Demand Allocation

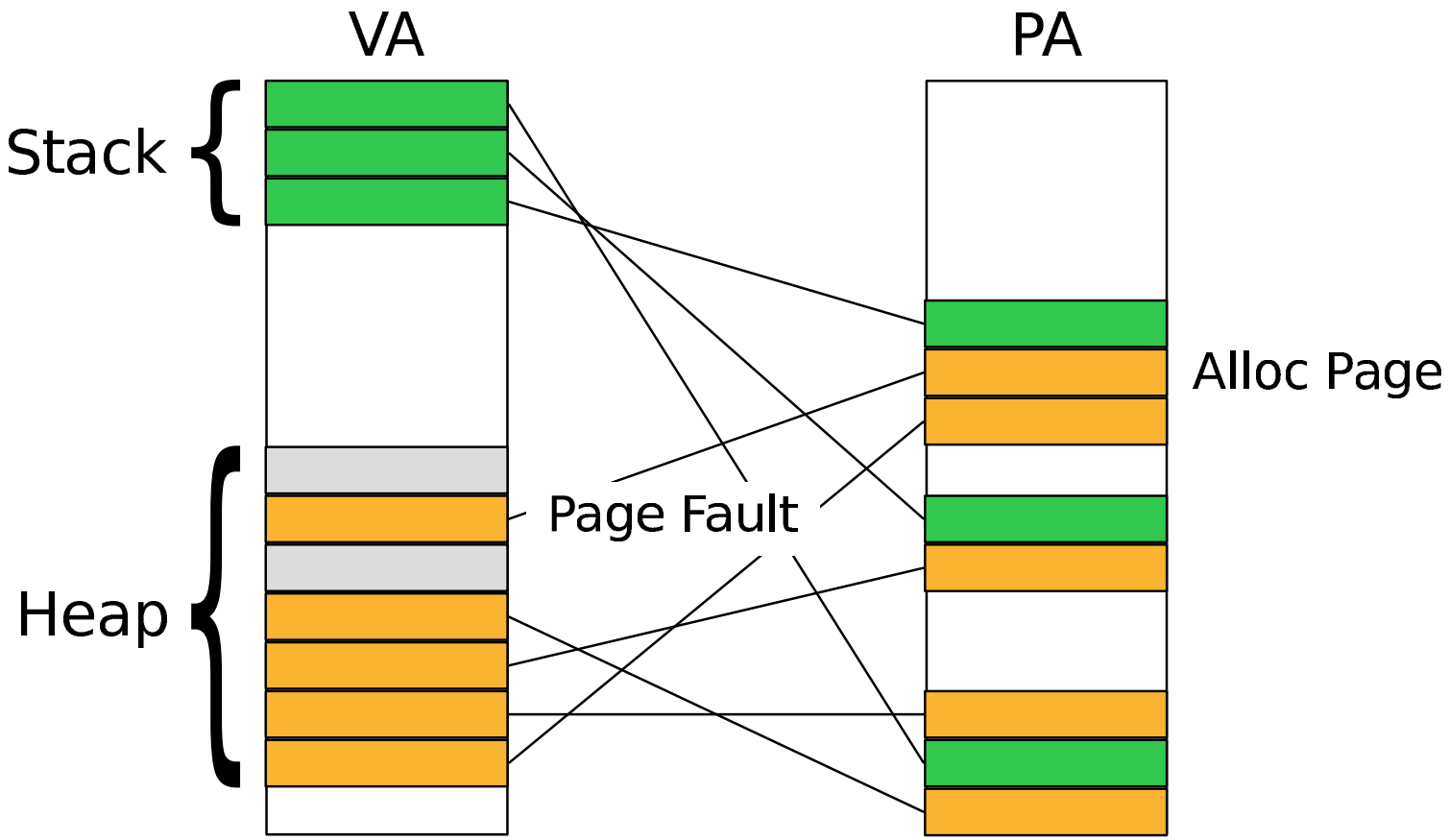

Figure : При обращении происходит Page Fault - выделяем страницу

Page Fault
On Demand Allocation

- аллоцируются только страницы, к котором происходит обращение - если процесс не использует память аллоцировать ее тоже не нужно;

Page Fault
On Demand Allocation

- аллоцируются только страницы, к котором происходит обращение - если процесс не использует память аллоцировать ее тоже не нужно;
- если ОС сказала, что аллоцировала память, еще не значит что она правда ее аллоцировала;

Page Fault
On Demand Allocation

- аллоцируются только страницы, к котором происходит обращение - если процесс не использует память аллоцировать ее тоже не нужно;
- если ОС сказала, что аллоцировала память, еще не значит что она правда ее аллоцировала; с этим можно частично бороться:
  - держать запас физических страниц для Page Fault;
  - swapping может предотвратить самое худшее;

Fork
Копирование VA

fork - системный вызов в Unix-like системах для создания нового процесса. Новый процесс является копией старого. Т. е. нужно копировать VA:

- честная копия VA может привести к большому количеству аллокаций физической памяти;
- честная копия VA может потребовать копирования большого количества памяти;

Fork
Копирование VA

fork - системный вызов в Unix-like системах для создания нового процесса. Новый процесс является копией старого. Т. е. нужно копировать VA:

- честная копия VA может привести к большому количеству аллокаций физической памяти;
- честная копия VA может потребовать копирования большого количества памяти;
- честная копия, на самом деле, не нужна:
  - не нужно копировать Read Only части VA;
  - не нужно копировать то, что мы не будем использовать;

Page Fault
Copy On Write

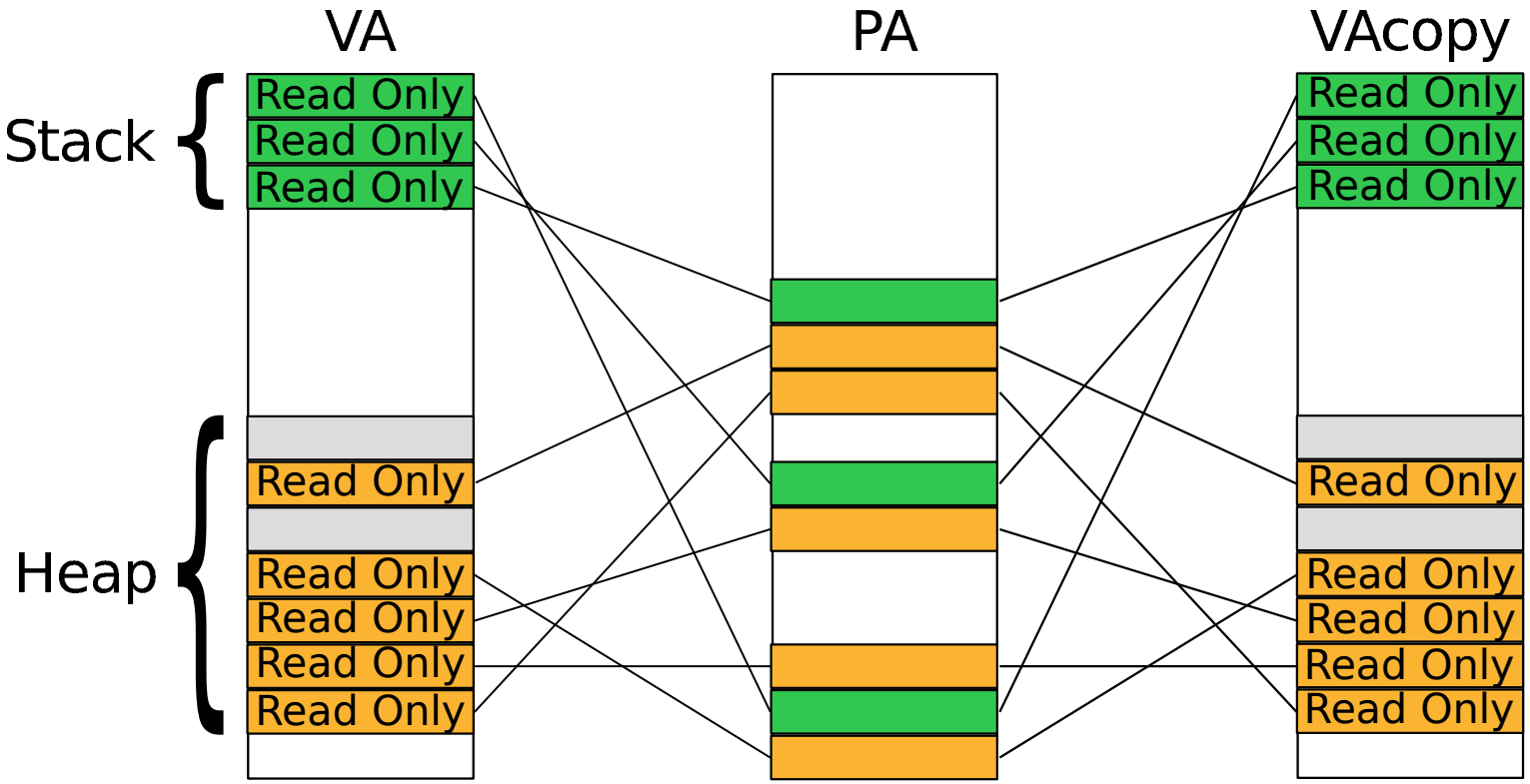

Figure : При копировании VA и оригинал и копия используют права Read Only

Page Fault
Copy On Write

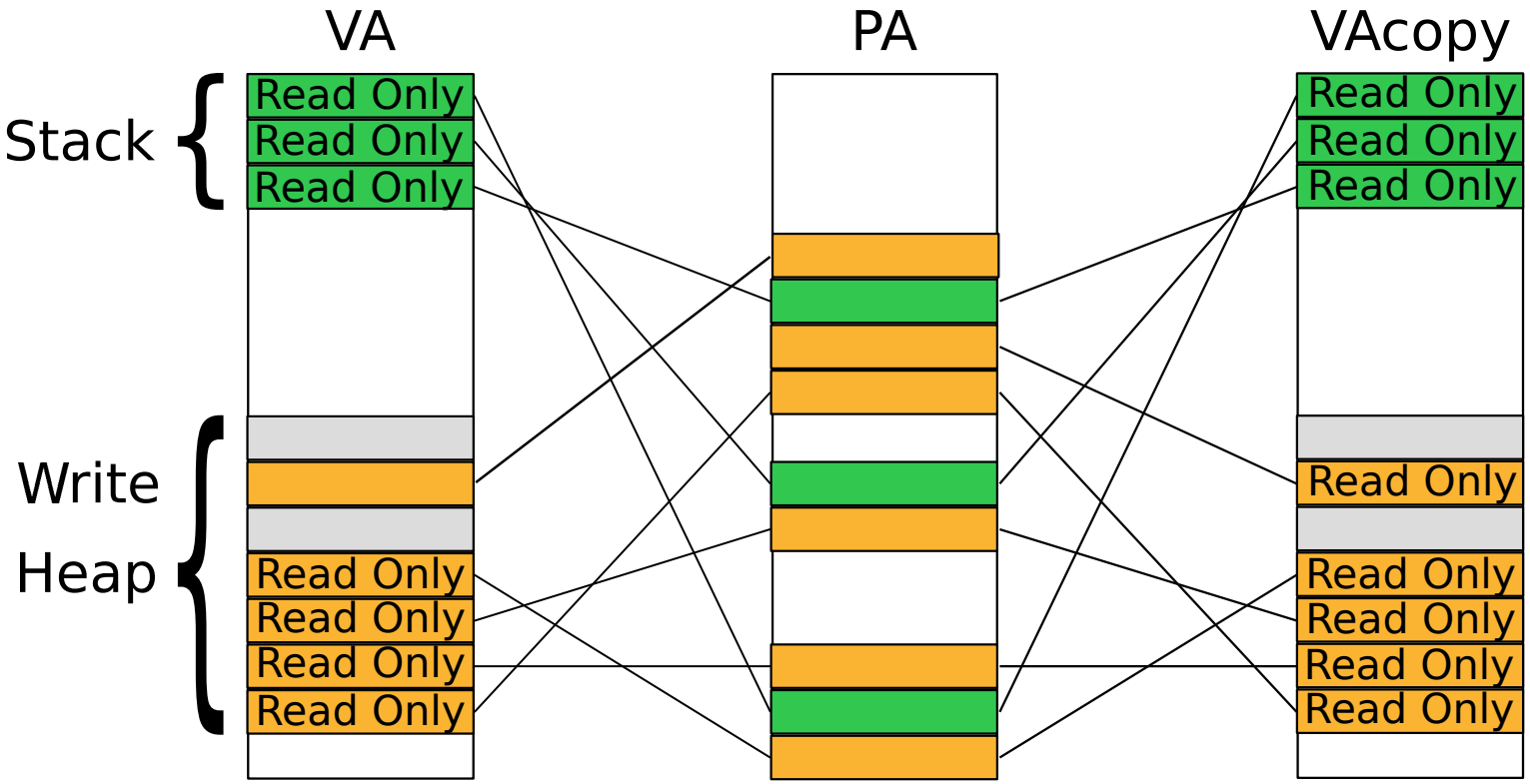

Figure : При записи происходит аллокация и настоящее копирование

Page Fault
Copy On Write

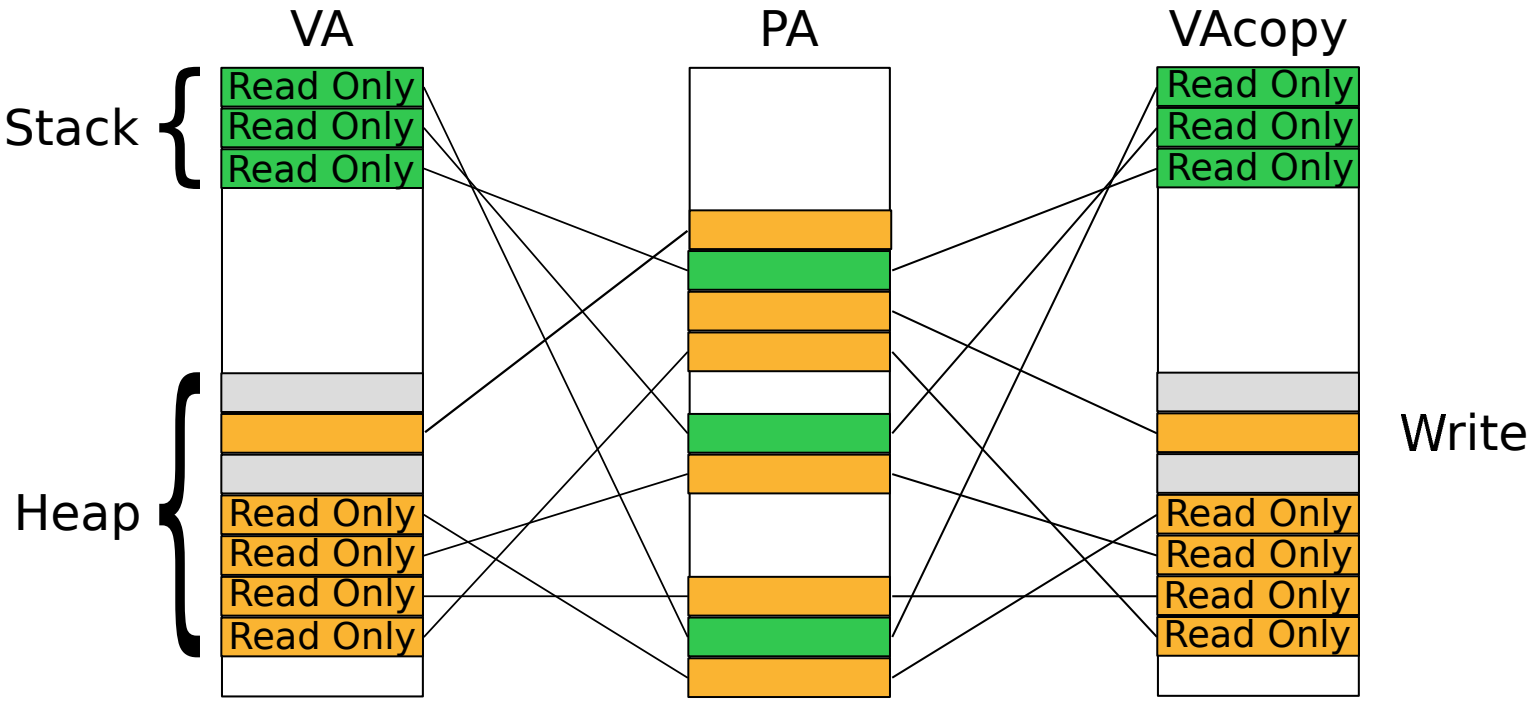

Figure : При записи происходит аллокация и настоящее копирование

Page Fault
Copy On Write

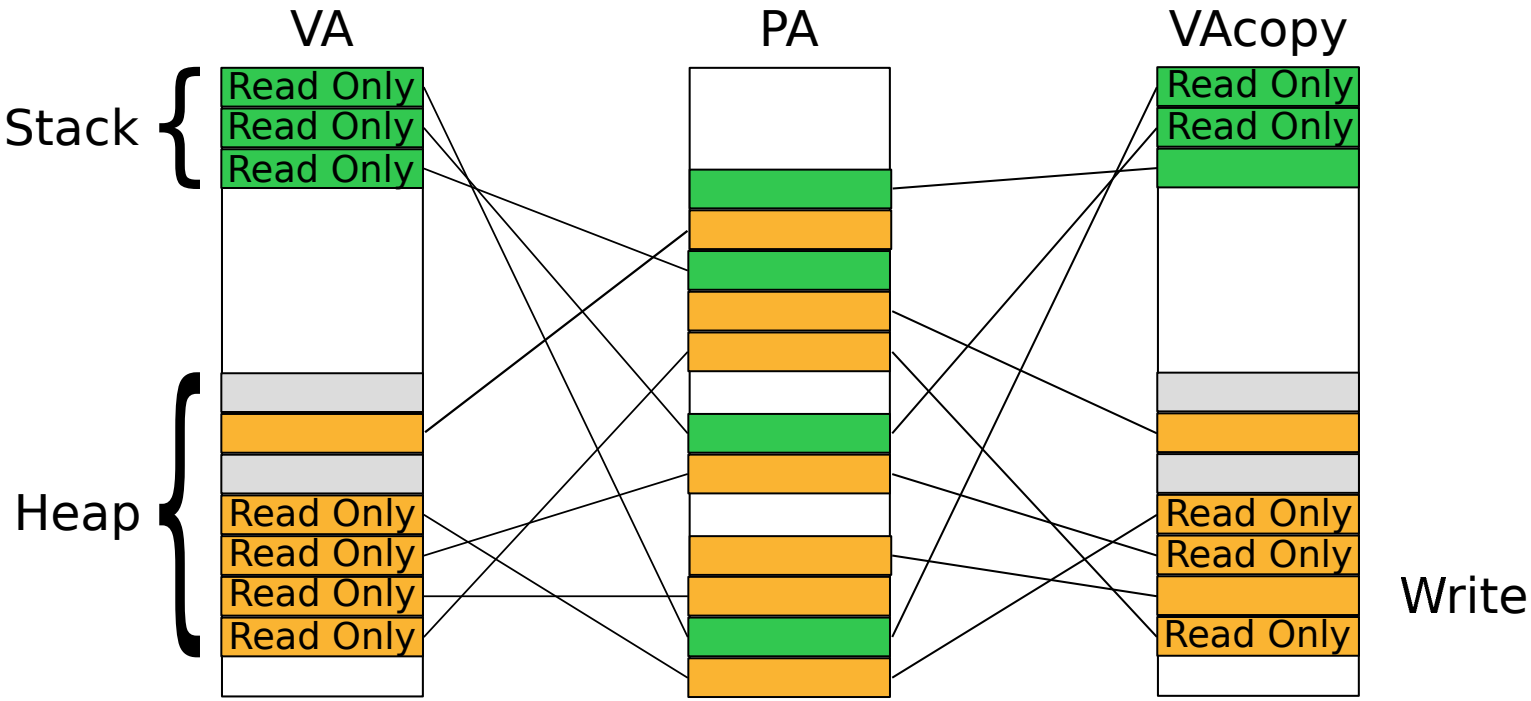

Figure : При записи происходит аллокация и настоящее копирование

Page Fault
Copy On Write

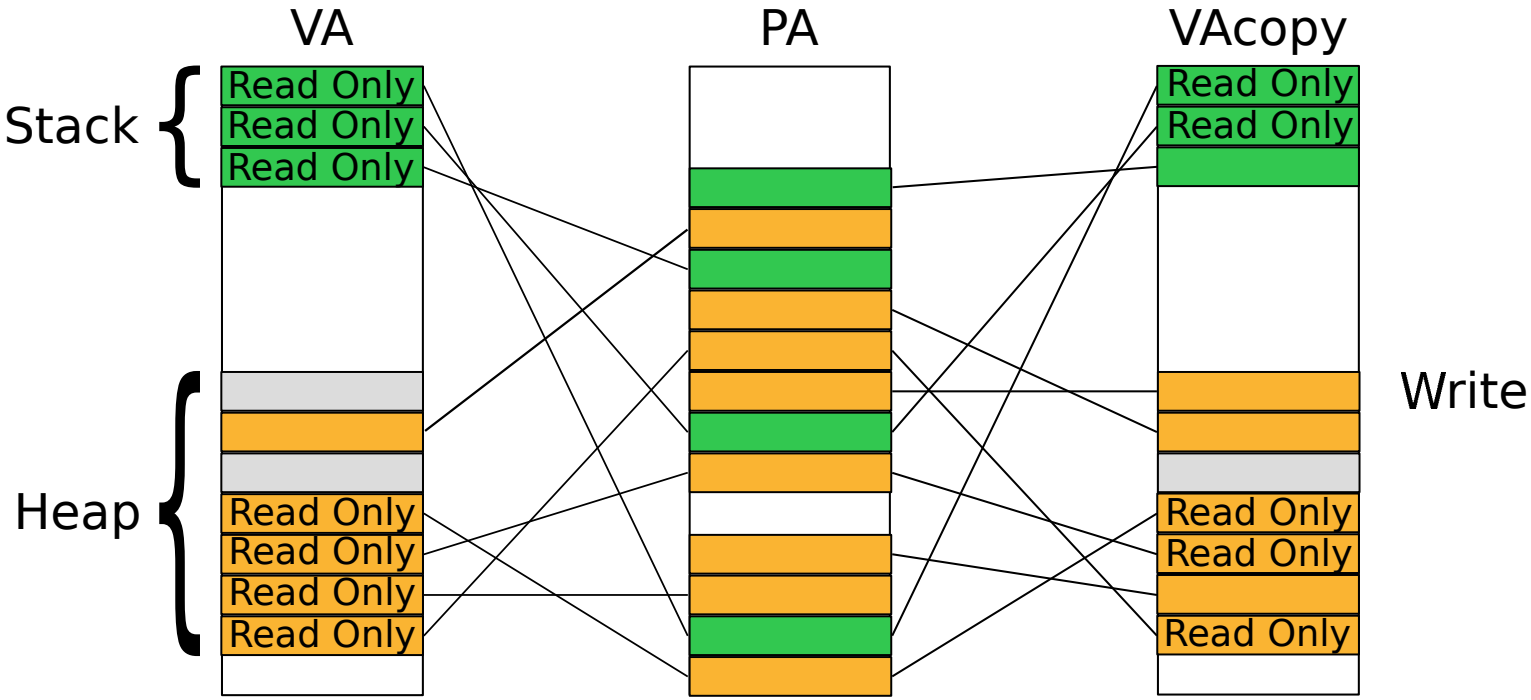

Figure : При записи происходит аллокация и настоящее копирование

Page Fault
Copy On Write

Для реализации Copy On Write, кроме наличия Page Fault, еще необходимы:

- счетчик ссылок для физических страниц - мы должны знать, что страницу нужно копировать;
  - если вы используете Buddy Allocator, то у вас уже есть дескриторы для страниц - в них можно хранить счетчик ссылок;
- "предполагаемые" права доступа к странице - мы должны знать, правда ли страница Read Only, или только из-за Copy On Write.
  - для каждого процесса должна быть структура описывающая его VA (в Linux Kernel она называется mm_struct, в FreeBSD она называется vmspace, в MAC OS X она называется vm_map)

Non-Contigous Page Allocation

- Аллокация больших блоков памяти может провалиться из-за фрагментации.

Non-Contigous Page Allocation

- Аллокация больших блоков памяти может провалиться из-за фрагментации.
- Paging позволяет отобразить последовательные участки VA на непоследовательные участки PA.

Non-Contigous Page Allocation

- Аллокация больших блоков памяти может провалиться из-за фрагментации.
- Paging позволяет отобразить последовательные участки VA на непоследовательные участки PA.
- Необходимо только иметь участок VA достаточного размера.

Адресное пространство процесса

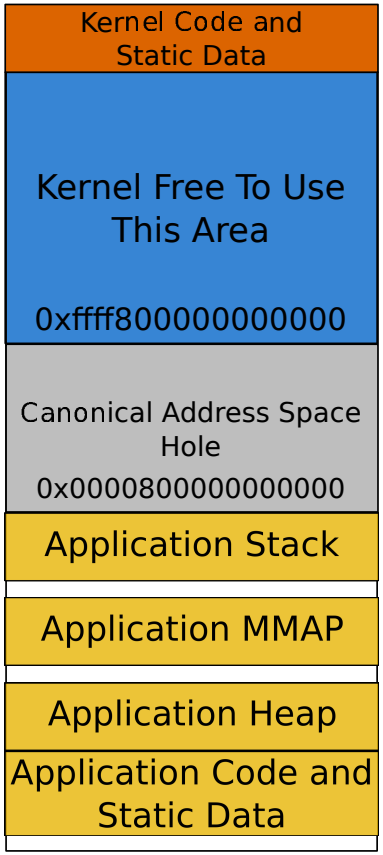

Figure : Типичная структура VA на x86-64

- Процессоры Intel поддерживают порядка 2TB RAM.
- У нас есть $2^{47} - 2GB$ VA между "дырой" и ядром (по факту, бесконечное VA).
- Будем аллоцировать непрерывные участки VA и отображать страницы на них.

There is much more about paging

За кадром осталось много полезных тем:

There is much more about paging

За кадром осталось много полезных тем:

- использование свободных страниц (коснемся этого когда дойдем до ФС)
  - свободная память - бесполезная память;
  - Page Cache, Buffer Cache, и много других;

There is much more about paging

За кадром осталось много полезных тем:

- использование свободных страниц (коснемся этого когда дойдем до ФС)
  - свободная память - бесполезная память;
  - Page Cache, Buffer Cache, и много других;
- swapping:
  - swapping возможен и без Paging-а;
  - Paging позволяет не выгружать все VA процесса на диск;

There is much more about paging

За кадром осталось много полезных тем:

- использование свободных страниц (коснемся этого когда дойдем до ФС)
  - свободная память - бесполезная память;
  - Page Cache, Buffer Cache, и много других;
- swapping:
  - swapping возможен и без Paging-а;
  - Paging позволяет не выгружать все VA процесса на диск;
- для Linux Kernel есть подробное (немного устаревшее) описание по ссылке