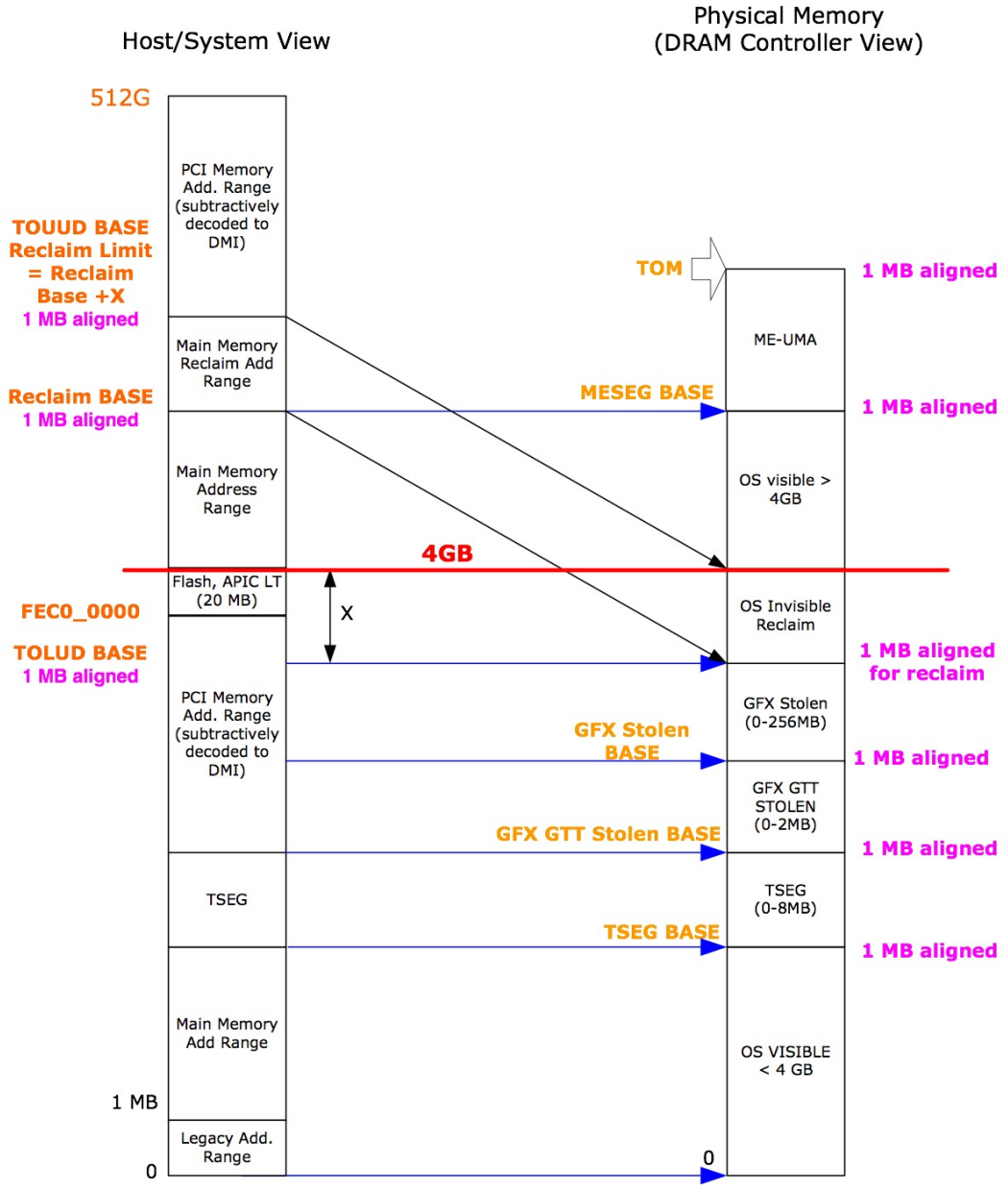
Конструирование ядра операционных систем (VI+VII)

Виртуальная память

Виды адресации x86

- На уровне контроллера ОЗУ
  - ЦПУ и ОЗУ видят память «по-разному»
  - Адресация прозрачна для ОС
  - Доступная память предоставлена UEFI
- Сегментная адресация
  - Используется в 32-битном режиме
  - Используется для Thread Local Storage https://uclibc.org/docs/tls.pdf
  - См. предыдущую лекцию
- Страничная адресация
  - Обеспечивает виртуальную память



UEFI Memory Map

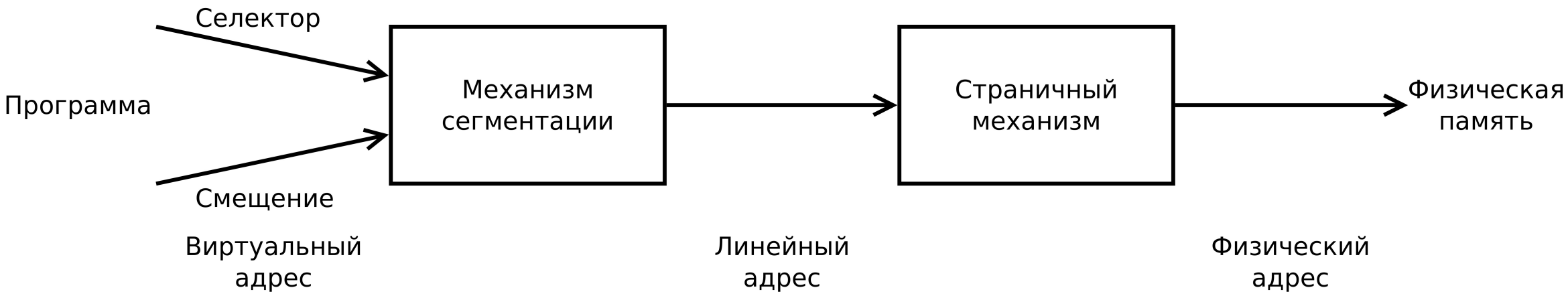
```
///
/// Definition of an EFI memory descriptor.
///
typedef struct {
  ///
  /// Type of the memory region.
  /// Type EFI_MEMORY_TYPE is defined in the
  /// AllocatePages() function description.
  ///
  UINT32                Type;
  ///
  /// Physical address of the first byte in the memory region. PhysicalStart must be
  /// aligned on a 4 KiB boundary, and must not be above 0xfffffffffffff000. Type
  /// EFI_PHYSICAL_ADDRESS is defined in the AllocatePages() function description
  ///
  EFI_PHYSICAL_ADDRESS  PhysicalStart;
  ///
  /// Virtual address of the first byte in the memory region.
  /// VirtualStart must be aligned on a 4 KiB boundary,
  /// and must not be above 0xfffffffffffff000.
  ///
  EFI_VIRTUAL_ADDRESS   VirtualStart;
  ///
  /// NumberOfPagesNumber of 4 KiB pages in the memory region.
  /// NumberOfPages must not be 0, and must not be any value
  /// that would represent a memory page with a start address,
  /// either physical or virtual, above 0xfffffffffffff000.
  ///
  UINT64                NumberOfPages;
  ///
  /// Attributes of the memory region that describe the bit mask of capabilities
  /// for that memory region, and not necessarily the current settings for that
  /// memory region.
  ///
  UINT64                Attribute;
} EFI_MEMORY_DESCRIPTOR;
```

```
typedef
EFI_STATUS
(EFIAPI *EFI_GET_MEMORY_MAP)(
  IN OUT UINTN                  *MemoryMapSize,
  OUT    EFI_MEMORY_DESCRIPTOR  *MemoryMap,
  OUT    UINTN                  *MapKey,
  OUT    UINTN                  *DescriptorSize,
  OUT    UINT32                 *DescriptorVersion
  );
```

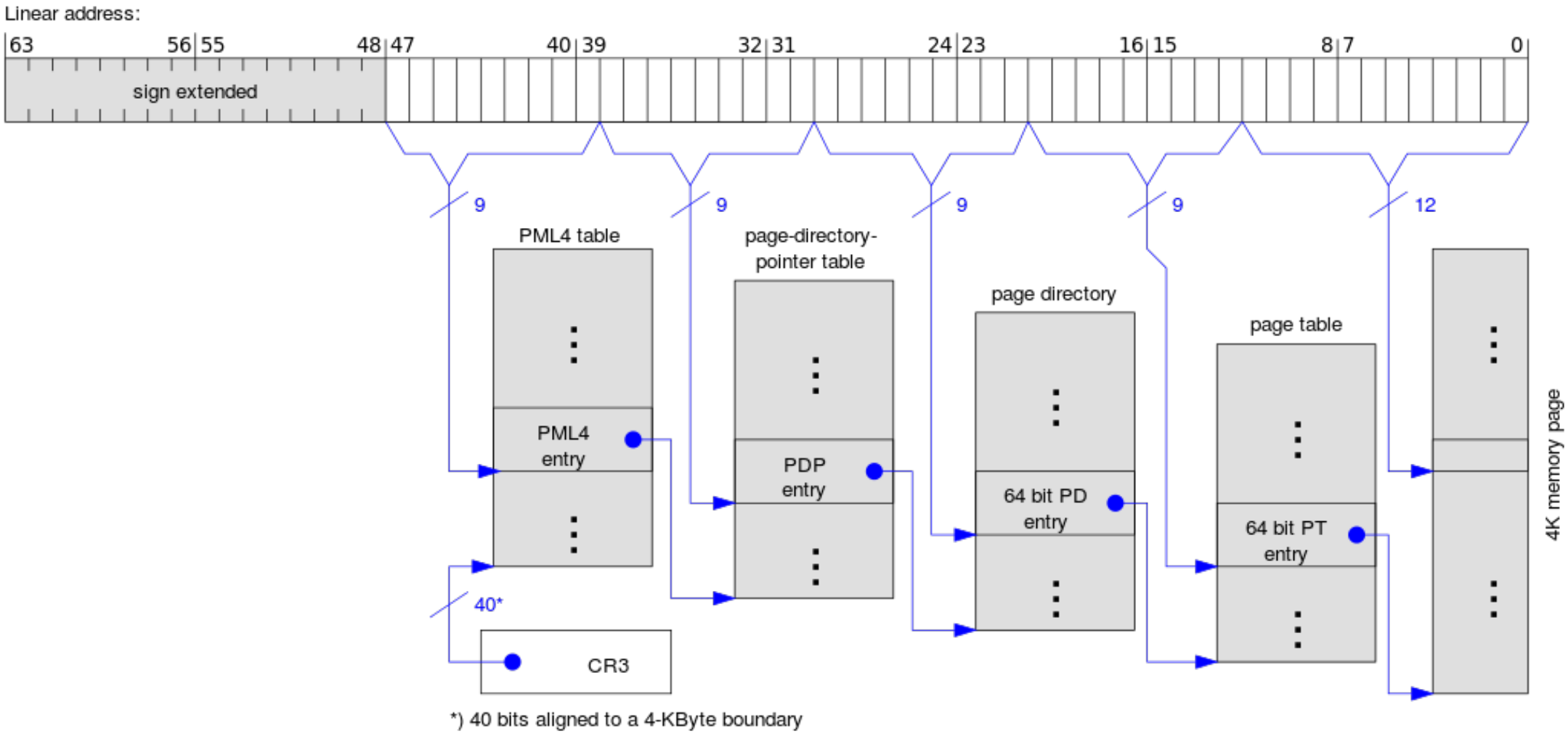
```
//
// Memory cacheability attributes
//
#define EFI_MEMORY_UC               0x0000000000000001ULL
#define EFI_MEMORY_WC               0x0000000000000002ULL
#define EFI_MEMORY_WT               0x0000000000000004ULL
#define EFI_MEMORY_WB               0x0000000000000008ULL
#define EFI_MEMORY_UCE              0x0000000000000010ULL
//
// Physical memory protection attributes
//
#define EFI_MEMORY_WP               0x0000000000001000ULL
#define EFI_MEMORY_RP               0x0000000000002000ULL
#define EFI_MEMORY_XP               0x0000000000004000ULL
#define EFI_MEMORY_RO               0x0000000000020000ULL
//
// Runtime memory attribute
//
#define EFI_MEMORY_RUNTIME          0x8000000000000000ULL
```

```
typedef enum {
  /// Not used.
  EfiReservedMemoryType,
  /// The code portions of a loaded application.
  /// (Note that UEFI OS loaders are UEFI applications.)
  EfiLoaderCode,
  /// The data portions of a loaded application and the default data allocation
  /// type used by an application to allocate pool memory.
  EfiLoaderData,
  /// The code portions of a loaded Boot Services Driver.
  EfiBootServicesCode,
  /// The data portions of a loaded Boot Serves Driver, and the default data
  /// allocation type used by a Boot Services Driver to allocate pool memory.
  EfiBootServicesData,
  /// The code portions of a loaded Runtime Services Driver.
  EfiRuntimeServicesCode,
  /// The data portions of a loaded Runtime Services Driver and the default
  /// data allocation type used by a Runtime Services Driver to allocate pool memory.
  EfiRuntimeServicesData,
  /// Free (unallocated) memory.
  EfiConventionalMemory,
  /// Memory in which errors have been detected.
  EfiUnusableMemory,
  /// Memory that holds the ACPI tables.
  EfiACPIReclaimMemory,
  /// Address space reserved for use by the firmware.
  EfiACPIMemoryNVS,
  /// Used by system firmware to request that a memory-mapped IO region
  /// be mapped by the OS to a virtual address so it can be accessed by EFI runtime services.
  EfiMemoryMappedIO,
  ...
} EFI_MEMORY_TYPE;
```

Виртульная память



Таблицы страниц



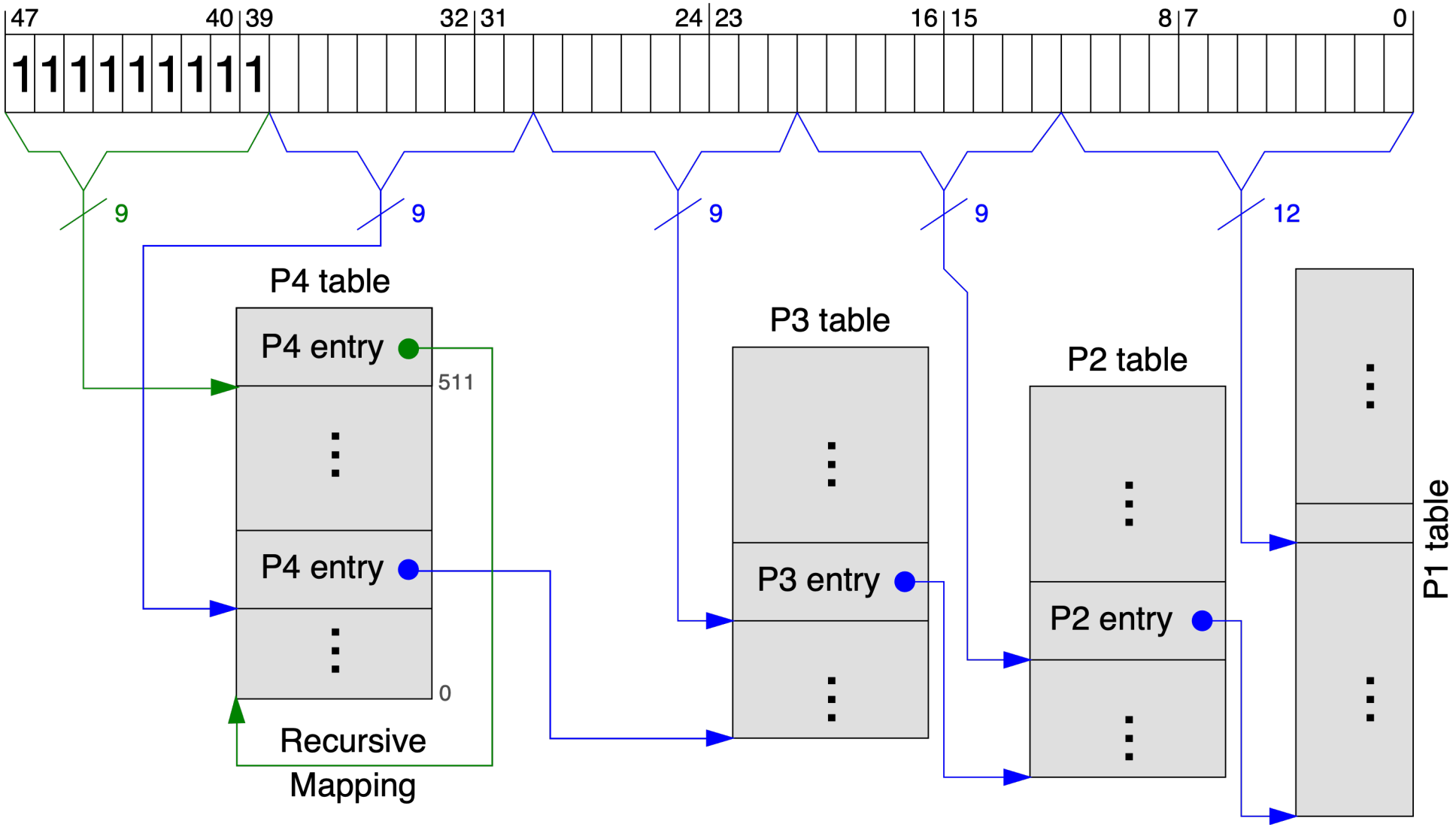
Основные флаги PML4E, PDPE, PDE

- S — Size, размер страницы: может быть выставлен только в PDE и PDPE; если бит выставлен, то размер страницы 2МБ и 1ГБ соответственно.
- A — Accessed, обращение. Процессор устанавливает этот флаг каждый раз, когда обращается к таблице для чтения или записи.
- D — Cache Disabled, запрещение кэша. Запрещает кэширование таблицы страниц.
- W — Write Through, сквозная запись. Управляет кэшированием страниц.
- U — User, пользовательская таблица. Если установлен — таблица доступна из пользовательского режима, если нет — только из режима ядра.
- R — Read/Write, чтение/запись. Определяет тип доступа к таблице: если установлен — таблица доступна на запись, если нет — только на чтение.
- P — Present, присутствие. Означает, что страница отображена и может использоваться при трансляции адреса.

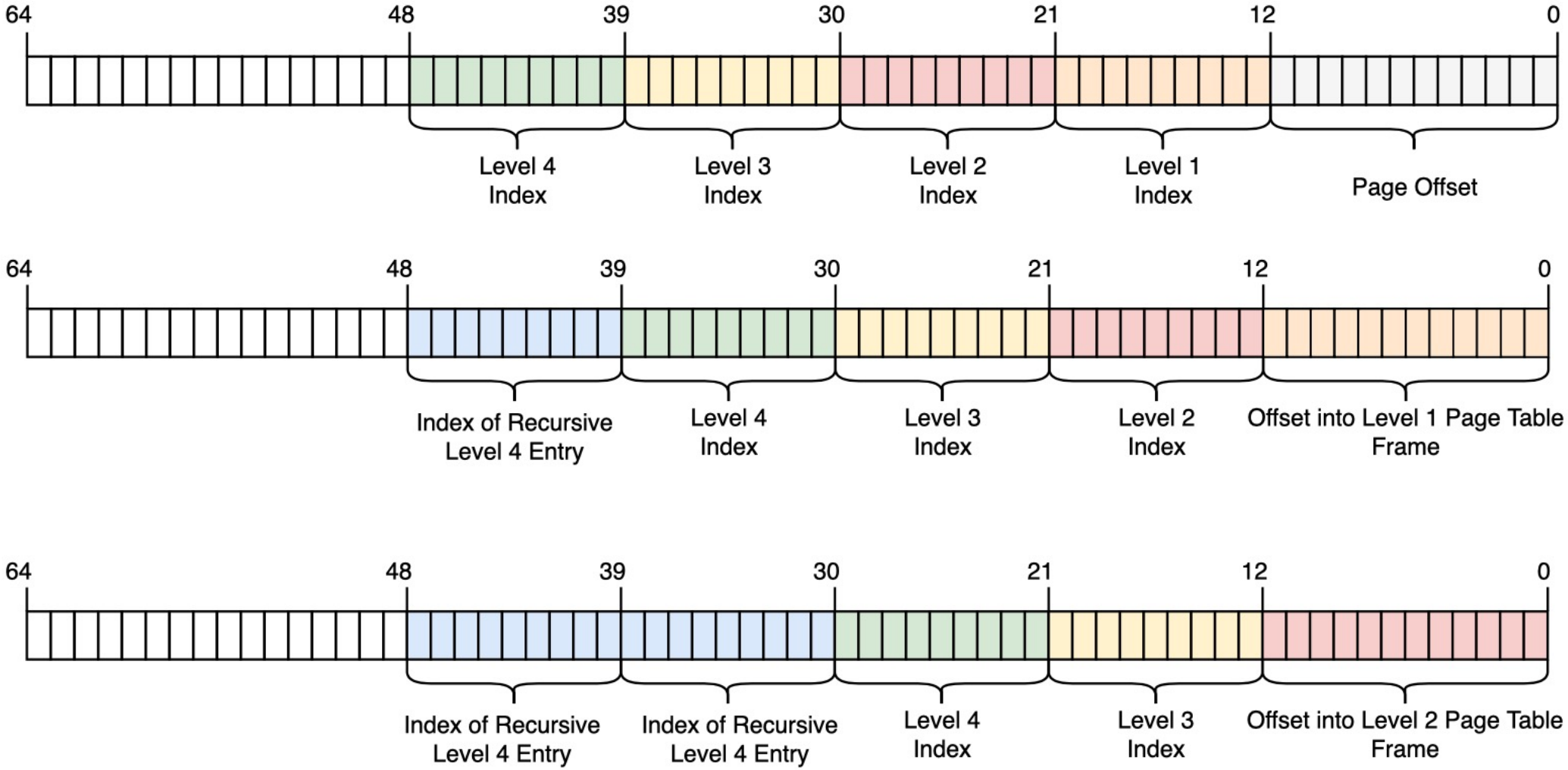
Основные флаги PTE

- D — Dirty, изменение. Процессор устанавливает флаг при обращении для записи.
- A — Accessed, обращение. Процессор устанавливает флаг при любом обращении.
- C — Cache Disabled, запрещение кэша. Запрещает кэширование страницы.
- U — User, пользовательская страница. Если установлен — страница доступна из пользовательского режима, если нет — только из режима ядра.
- R — Read/Write, чтение/запись. Определяет тип доступа к странице: если установлен — страница доступна на запись, если нет — только на чтение.
- P — Present, присутствие. Означает, что страница отображена и может использоваться при трансляции.
- NX - No execute, без выполнения. Если данный бит выставлен, то исполнение кода в данной странице запрещено (индивидуальное задание).

Self-referencing Page Tables (1/3)



Self-referencing Page Tables (2/3)



Self-referencing Page Tables (3/3)

- Операционные системы Windows используют данную технику для упрощения управления виртуальными пространствами.
- Техника позволяет работать только с текущим активным виртуальным пространством (загруженным в CR3).
- Если значение self-referencing индекса в PML4 не случайное (до Windows 10 Microsoft использовали 0x1ED), то техника позволяет узнать все корректные виртуальные адреса и избежать KASLR ☺.

Виртульная память и Intel SDM

- Volume 1: 3.3 Memory Organization
- Volume 2: 5.10 Pointer Validation
- Volume 2: 5.11 Page Protection
- Volume 3: 4 Paging (особенно 4.1, 4.3, 4.6, 4.7, 4.8)
- Volume 3: 5 Protection

Спасибо за внимание!

Вопросы?