Using segmentation to build a capability-based single address space operating system

Wuyang Chung

wy-chung@outlook.com

You can see the video on YouTube.

Outline

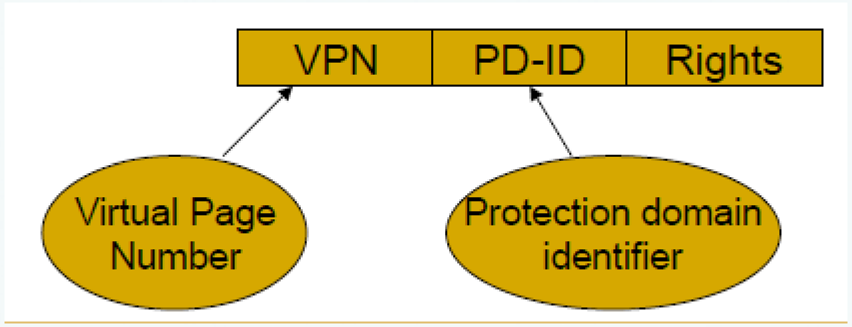
- Single Address Space Operating System
- Capability-based Addressing
- Segmentation
- Benefits of Segmentation
- Compatibility with Old Programs
- Conclusion

Single Address Space OS

- Problems with multiple address space OS
  - Pointers cannot be used in shared buffer
  - On CPUs without ASID
    - Need to flush TLB when doing address space switch
  - On CPUs with ASID
    - Duplications of translation information in TLB
- Advantages of single address space OS
  - Pointers can be used in shared buffer
  - Lower context switch overhead
- Challenge of single address space OS
  - Protection

Multiple Protection Domains on a Single Address Space

- Domain page model
  - PLB (Protection Lookaside Buffer)

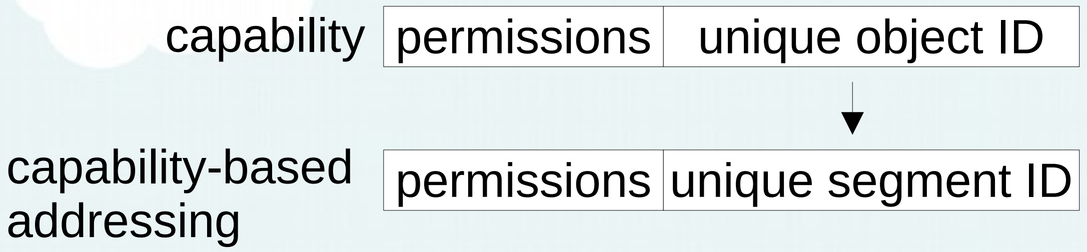


Architecture support for single address space operating systems, ACM SIGPLAN, 1992.

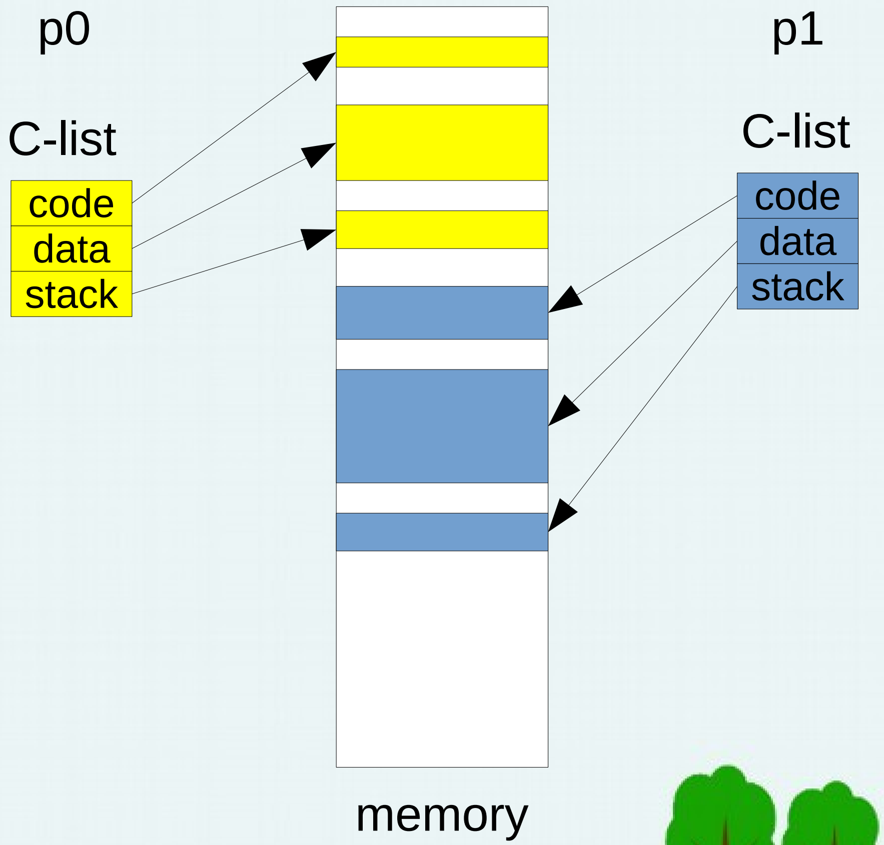
https://cseweb.ucsd.edu/classes/fa11/cse240A-a/Slides1/06_SAOS.pdf

- Page group model
  - There is an AID (access identifier) field in page table entry
  - A process can access a page if any of the PIDs is equal to the page AID
  - The CPU has several PID (protection identifier) registers
- Capability-based addressing

Capability-based Addressing

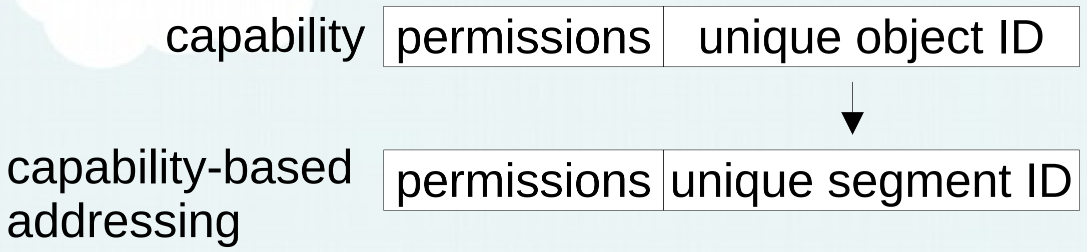


partitioned approach

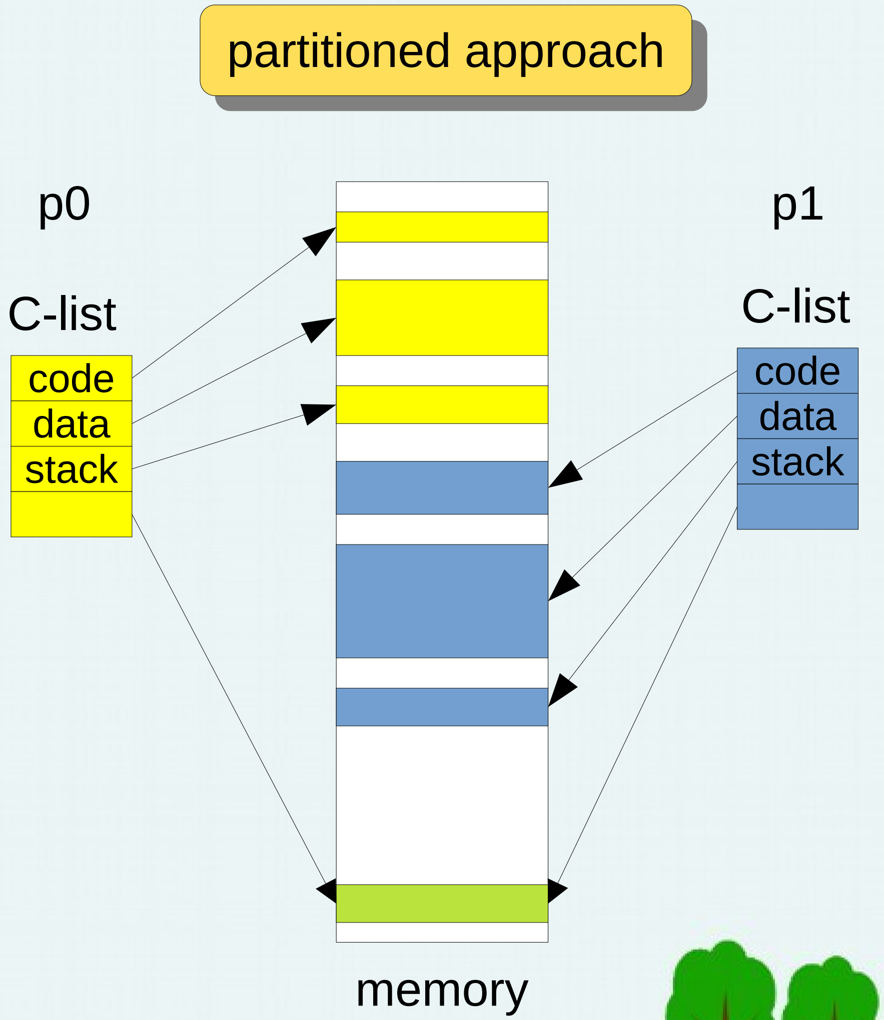


- The most important thing in a capability system is to prevent capability from being forged
- Two approaches
  - Tagged approach
    - e.g. CHERI
  - Partitioned approach
    - e.g. Plessey 250

Capability-based Addressing

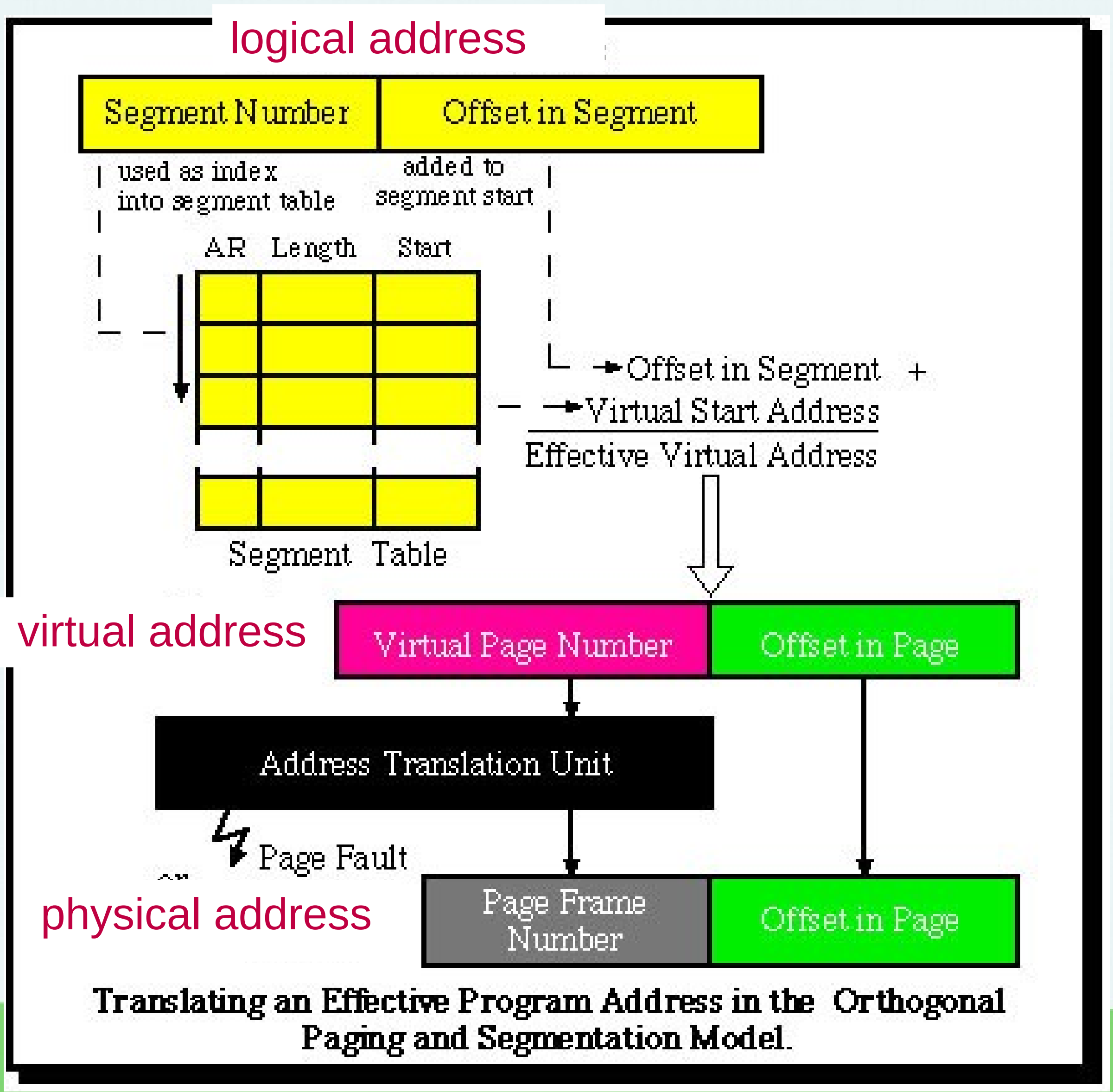


- The most important thing in a capability system is to prevent capability from being forged
- Two approaches
  - Tagged approach
    - e.g. CHERI
  - Partitioned approach
    - e.g. Plessey 250



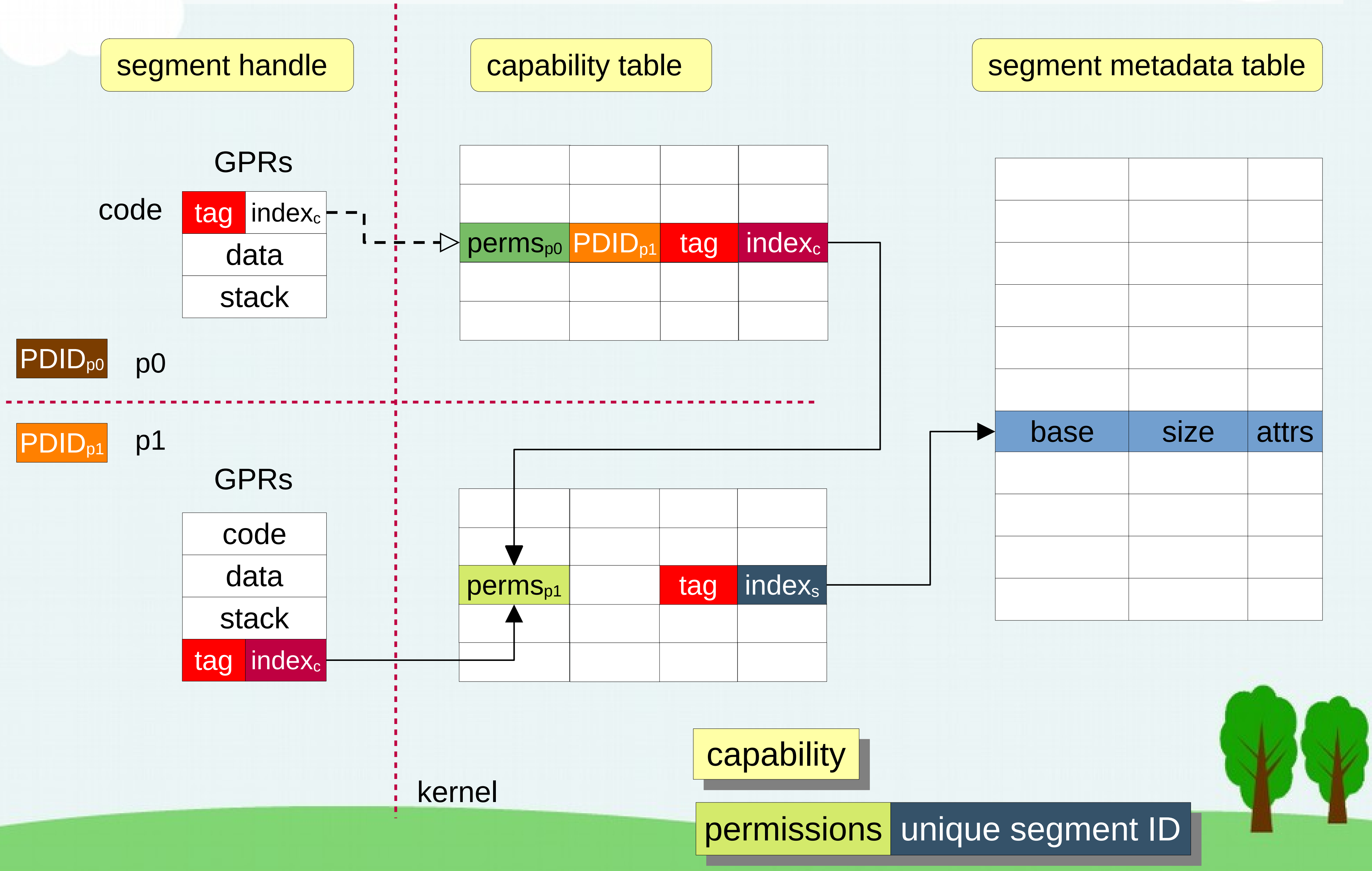
Orthogonal Segmentation and Paging

https://www.monads-security.org/orthogonal-segmentation-and-paging.html



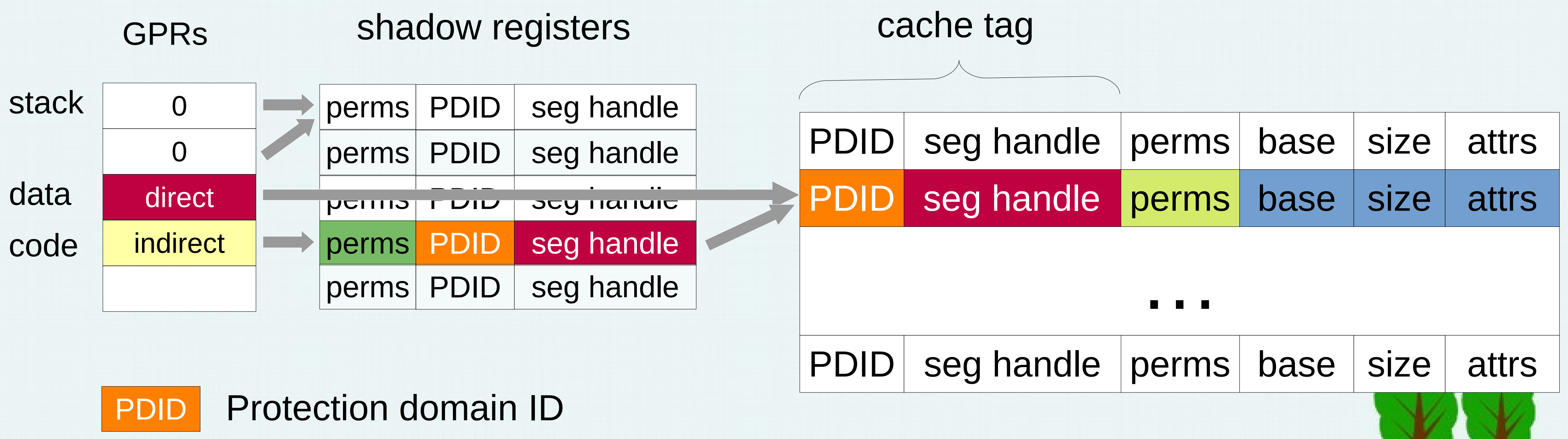
Translating an Effective Program Address in the Orthogonal Paging and Segmentation Model.

Segmentation



Segmentation Hardware

- Shadow registers
  - used to cache indirect segment handle
- SLB (Segment Lookaside Buffer)
  - Cache segment translation



Segment Types

- Segmentation translates logical address to linear address
  - There are 3 types of linear address
    - Virtual, physical and I/O address
- Segment types
  - Virtual segment
    - Linear address is virtual address
  - Physical segment
    - Linear address is physical address
  - I/O segment
    - Linear address is I/O address

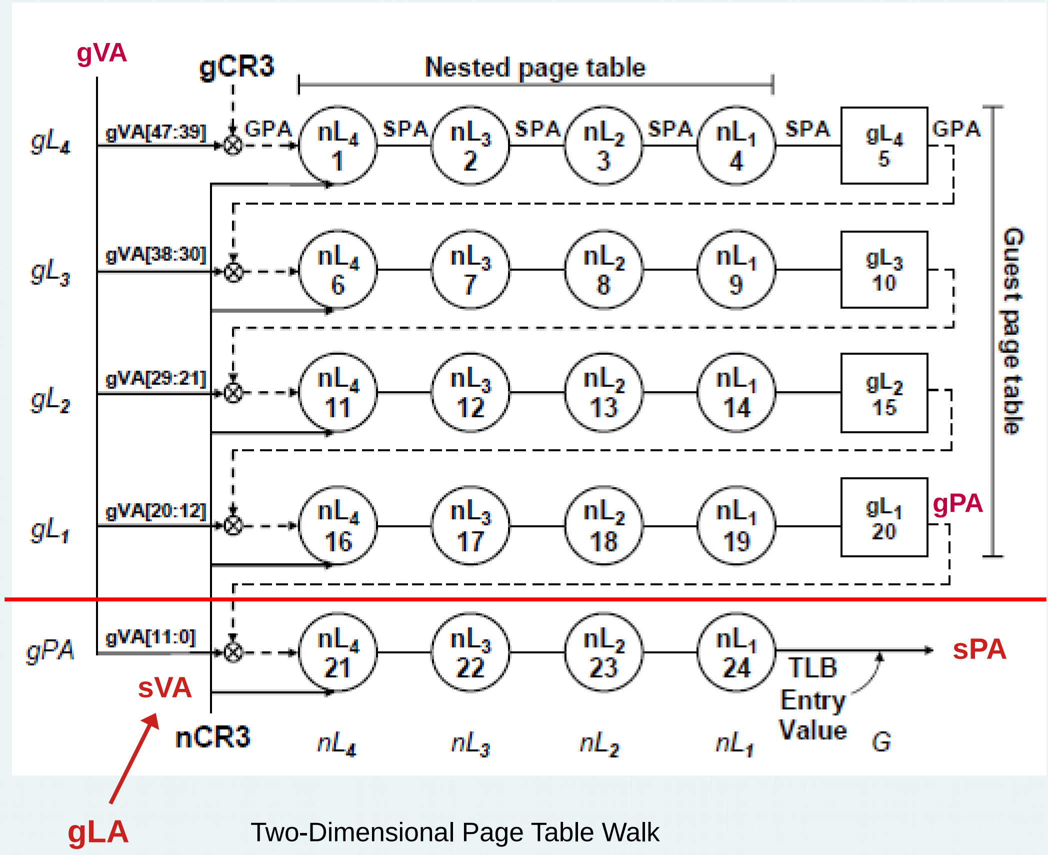
logical address is
{segment handle, offset within the segment}

Benefits of Virtual Segment

- Fast Segment move
  - Don't need to copy the segment data, only need to copy the virtual to physical mapping
  - Don't need to fix any pointers in the segment after the segment is moved

Simplification of page table walk in Guest Virtual Machine

- On a guest machine, hardware needs to do two virtual address translations for TLB miss
  - Guest virtual address to guest physical address
  - System virtual address to system physical address
- With virtual segment only one virtual address translation is needed
  - Guest logical address can be translated directly to system virtual address



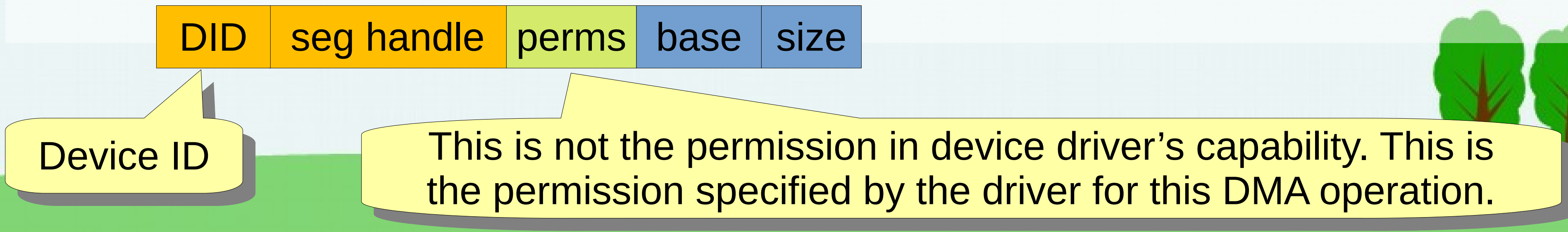
Two-Dimensional Page Table Walk

Accelerating two-dimensional page walks for virtualized systems, acm ASPLOS, 2008

logical address is
{segment handle, offset within the segment}

Benefits of Physical Segment

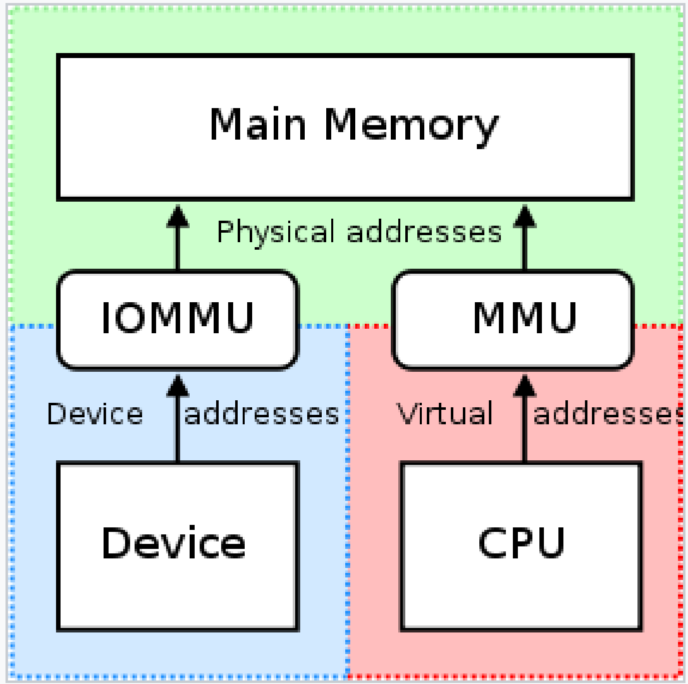
- Software-loaded TLB
  - Must guarantee no TLB miss exceptions when the kernel is running the TLB miss handling routine
- Direct segment
  - Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, Michael M. Swift. Efficient virtual memory for big memory servers. In Proc. ISCA, 2013.
    - TLB misses consume up to 51% of execution cycles for big memory servers
- IOMMU simplification
  - Simple authorization
  - No page translation is needed
  - IOMMU is simply a cache for physical segment metadata



This is not the permission in device driver's capability. This is the permission specified by the driver for this DMA operation.

Simplification of IOMMU

- How the IOMMU protects main memory from hardware device?
  0. Device driver allocates a physical segment and sends the **segment handle** to hardware device
  1. Device sends its device ID, the **segment handle**, offset and data length to IOMMU
  2. IOMMU verifies that the segment handle is valid and it points to a **physical segment**
  3. IOMMU then checks that the offset and length are within segment size
  4. IOMMU calculates the physical address by adding segment base address and the offset
  5. The calculated physical address is used to access system main memory



Comparison of the I/O memory management unit (IOMMU) to the memory management unit (MMU).

From Wikipedia

I/O Segment

- Purpose
  - Leave memory address space to memory only and move all the other stuff to I/O address space, such as CPU’s CSRs, PCI configuration space, video frame buffer, boot ROM, etc.
- Benefits
  - No memory holes in memory address space
  - Segment is more fine-grained then page
  - For a large I/O buffer, only one SLB entry is needed
  - The same load instruction can be used to load from either I/O or memory address space
  - Even when a device driver runs at the user level, it can access its hardware device without kernel intervention

Simplification of Shared Library

- No need for PIC (Position Independent Code), GOT (Global Offset Table) and PLT (Procedure Linkage Table)
  - Shared library has its own code and data segments and segment always starts with offset 0
- Shared libraries can be partially linked
  - The offset part of the library’s global variables and functions can be statically linked at compile time
  - The capability descriptor of the library’s global variables and functions are linked at program load time

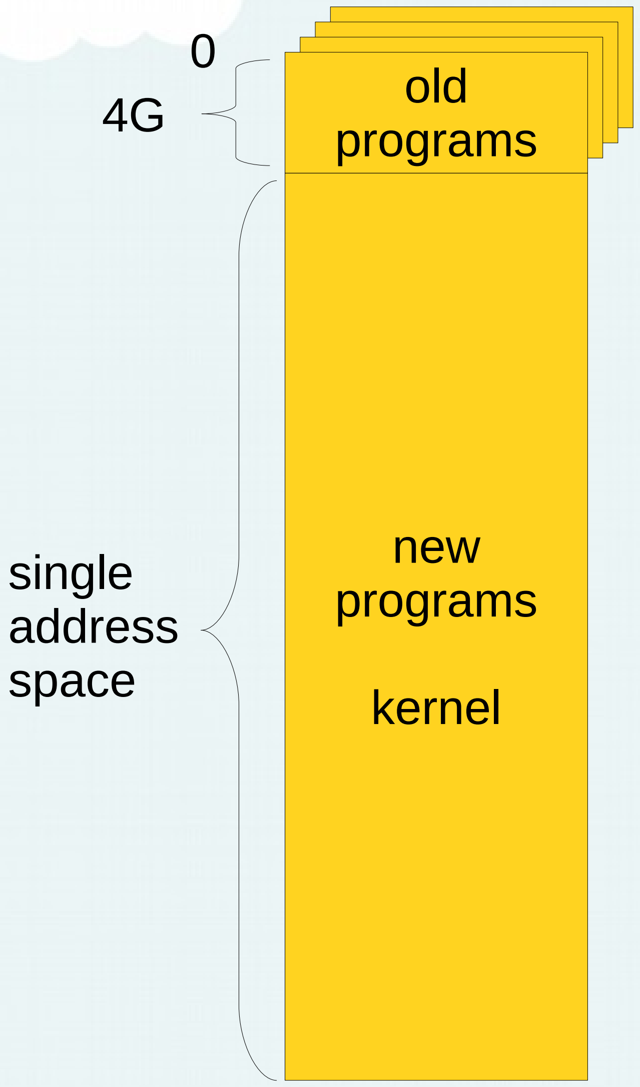
Cross-process Call

- Cross-process call has many names:
  cross-domain call, migrating thread model, protected control transfer, passive object model
- Traditionally thread is confined in two protection domains
  - When a thread is in the kernel, it can only return back to its creator process
- Cross-process Call
  - When a thread is in the kernel, it is allowed to upcall into other process
- Benefits
  - Encourage modularity and improve security
  - Resource accounting can be made more accurate
  - Preferable for real-time systems

Simplification of Cross-process Call

- The same stack segment can be used in both client and server process if we can
  - Protect the client part of the stack from server
    - By using spmin
  - Protect the server part of the stack from client
    - By hiding the stack handle for all the threads
      - The stack handle is always 0
- Need two new privileged registers
    - spmin
      - Thread can only access the part of the stack above this address
    - spmax
      - Automatically updated by CPU when sp is greater than spmax
      - Used for stack cleaning before and after cross-process call

Compatibility with Old Programs



- Ideal platform
  - 32-bit programs on 64-bit virtual address space
- The first 4G of the virtual address space is reserved for old multiple address space programs
- The rest of the virtual address space is for new single-address-space programs and the kernel

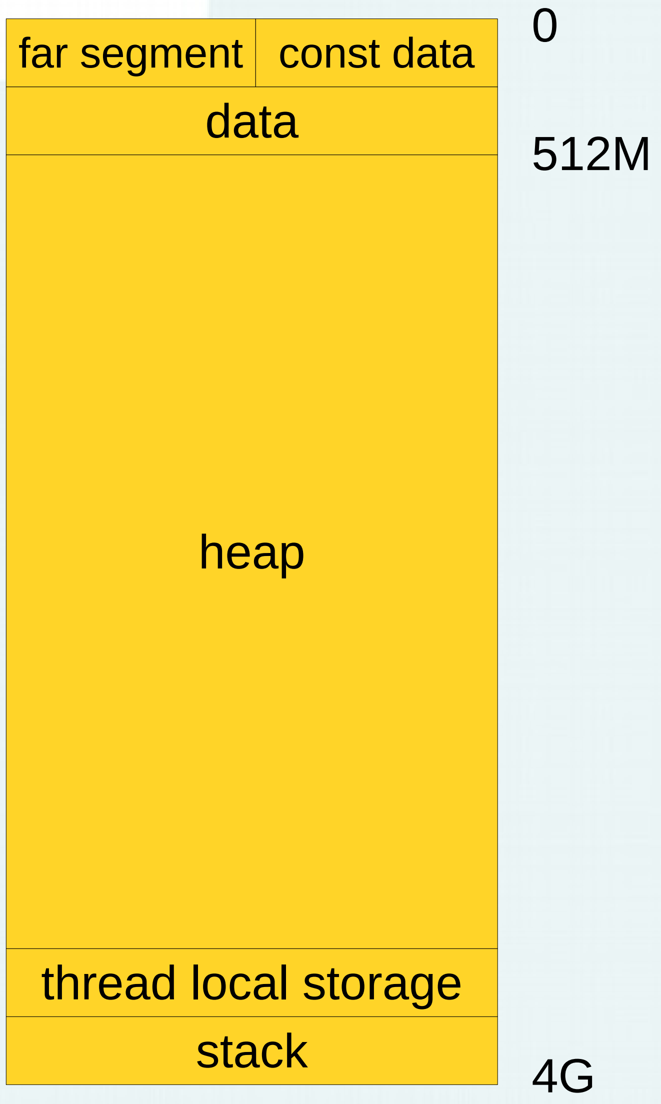
Conclusion

- Lots of benefits we can get from single address space, capability and segmentation
- Changes
  - OS
    - Segment, Capability and single address space
  - Compiler
    - Far pointer, far function call
    - New shared library implementation
  - Hardware
    - Capability and segmentation architecture

The End

Thank you

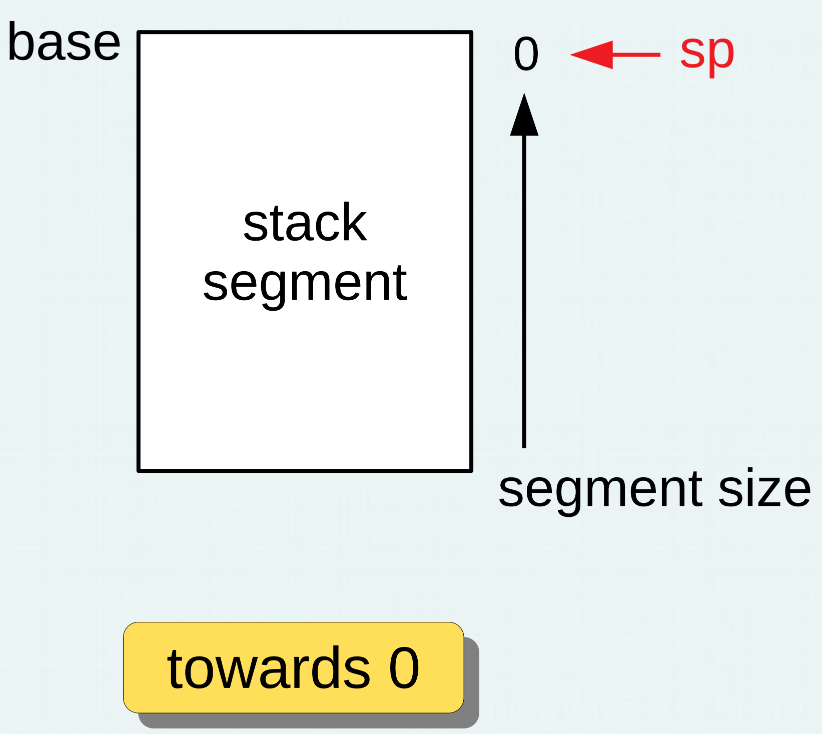
Data Address Space Partition

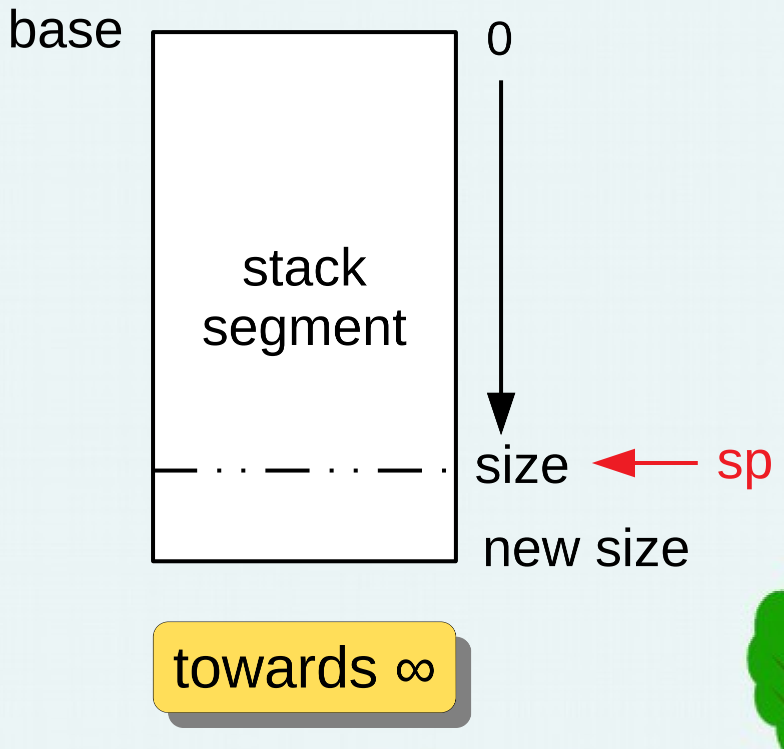


- There are 6 data segments for a thread
  - constant data
  - data
  - stack
  - thread local storage
  - heap
- They can share one data address space
  - Don’t need to use far pointer for pointers points to them

Stack Growth Direction

- Stack should grow towards ∞ instead of 0
  - Can expand the stack segment when it reaches its maximum size


base
stack segment
0
sp
segment size
towards 0


base
stack segment
0
size
sp
new size
towards ∞