

18-447 Lecture 17: Address Translation

James C. Hoe

Department of ECE

Carnegie Mellon University

Housekeeping

- Your goal today
 - see “Virtual Memory” in easy to digest pieces
- Notices
 - Break Day Monday
 - Lab 3, due Friday 4/9 noon
 - HW 4, due Monday 4/12 noon
 - Midterm 2, online during class time, Wed, 4/14
 - Midterm 3, online, Tue, 5/11, 5:30pm start
- Readings
 - P&H Ch 5

Format of Midterm

- Covers lectures (L11~L18), HW, labs, assigned readings (from textbook and papers)
- Types of questions
 - freebies: remember the materials
 - >> **probing: understand the materials** <<
 - applied: apply the materials in original interpretation
- ****55 minutes, 55 points****
 - 11 short-answer, typed-response questions
 - start of class on 4/14, online through Canvas
 - communicate with me privately by Zoom chat
 - openbook, **individual effort**

What to Expect

- 11 “5-point” short answer questions
 - ordered “easier” to “harder”
 - 1 question at a time and cannot go back
 - only first 45 words of each response graded
- Recommended strategy
 - give each question about 5min—as if taking 11 separate 5-min quizzes
- Be prepared
 - try practice midterm on Canvas
 - have your space and equipment ready
 - have a clock on your desk

2 Parts to Modern VM

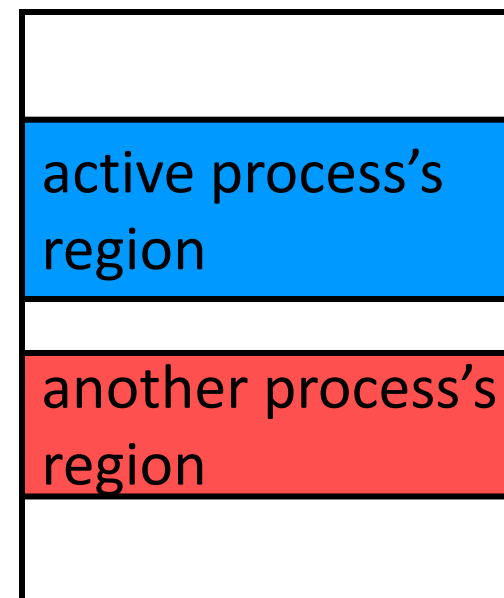
- In a multi-tasking system, **virtual** memory supports the **illusion** of a large, private, and uniform memory space to each process
- Ingredient A: naming and protection
 - each process sees a large, contiguous address space without holes (**for convenience**)
 - each process's memory is private, i.e., protected from access by other processes (**for sharing**)
- Ingredient B: demand paging (**for hierarchy**)
 - capacity of secondary storage (swap space on disk)
 - speed of primary storage (DRAM)

The Common Denominator: Address Translation

- Large, private, and uniform abstraction achieved through address translation
 - user process operates on effective address (**EA**)
 - HW translates from **EA** to physical address (**PA**) on every memory reference
- Through address translation
 - control which physical locations (DRAM and/or swap disk) can be referred to by a process
 - allow dynamic allocation and relocation of physical backing store (where in DRAM and/or swap disk)
- Address translation HW and policies controlled by the OS and protected from user

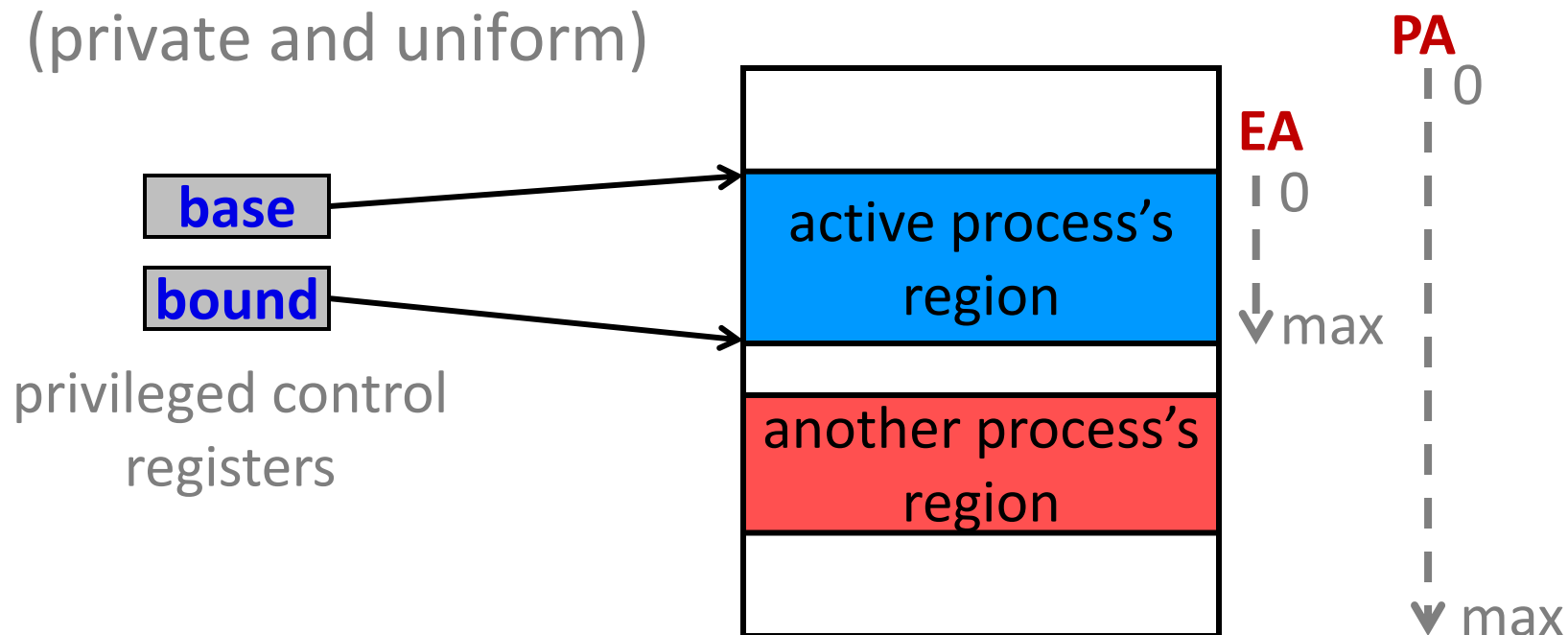
Beginnings of Memory Protection

- No need for protection or translation early on
 - single process, single user at a time
 - access all locations directly with **PA**
- Multitasking 101
 - each process limited to a non-overlapping, contiguous physical memory region (space doesn't start from addr 0 . . .)
 - everything must fit in the region
 - how to keep one process from reading or trashing another process's code and data? (*see corewars.org*)



Base and Bound

- A process's private memory region defined by
 - **base**: starting address of region
 - **bound**: size of region
- User process issue “effective” address (**EA**) between 0 and the size of its allocated region (private and uniform)



Base and Bound Registers

- Translation and protection check in hardware on every user memory reference
 - $PA = EA + \text{base}$
 - if ($EA < \text{bound}$) then okay else violation
- When switching user processes, OS sets **base** and **bound** registers
- User processes cannot be allowed to modify **base** and **bound** registers themselves

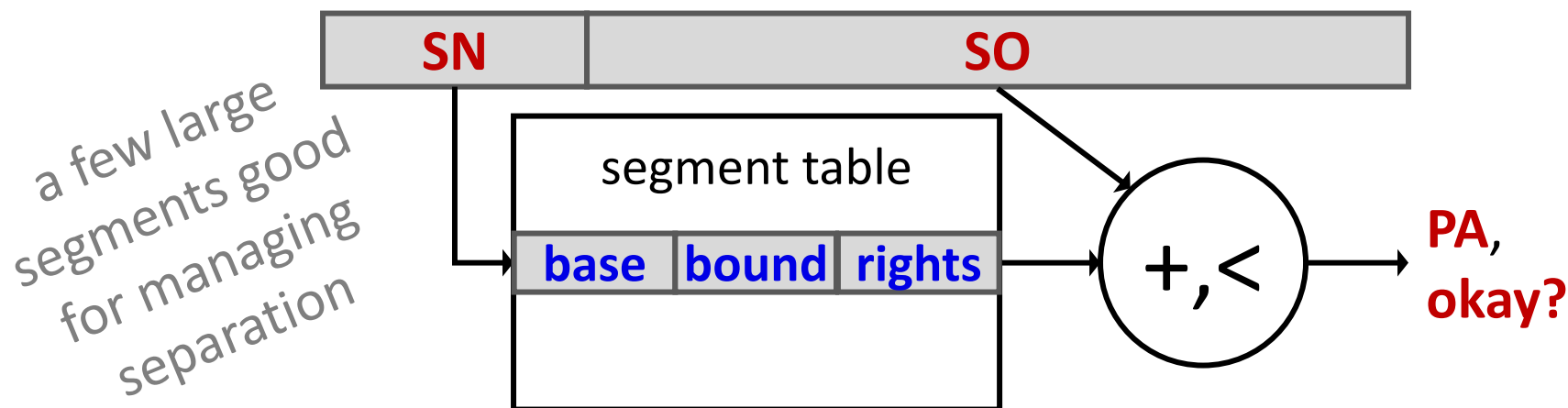
Requires at least 2 privilege levels with
protected instruction and state for OS only

Segmented Memory

- Limitations of single base-and-bound region
 - hard to find large contiguous space after a while—free space become fragmented
 - can two processes shared some memory regions but not others?
- A “base-and-bound” pair is a unit of protection
⇒ give user multiple memory “segments”
 - each segment is a contiguous memory region
 - each segment is defined by a **base** and **bound** pair
- Earliest use, separate code and data segments
 - 2 sets of **base/bound** for code vs data
 - forked processes can share code segments

more elaborate later: code, data, stack, etc.

Segmented Address



- **EA** partitioned into segment number (**SN**) and segment offset (**SO**)
 - max segment size limited by the range of **SO**
 - active segment size set by **bound**
- Per-process segment translation table
 - map **SN** to corresponding **base** and **bound**
 - separate mapping for each process
 - privileged structure if used to enforce protection

Access Protection

- Per-segment access rights can be specified as protection bits in segment table entries
- Generic options include

- Readable?
- Writeable?
- Executable?

also misc.
options, such as
“cacheable”

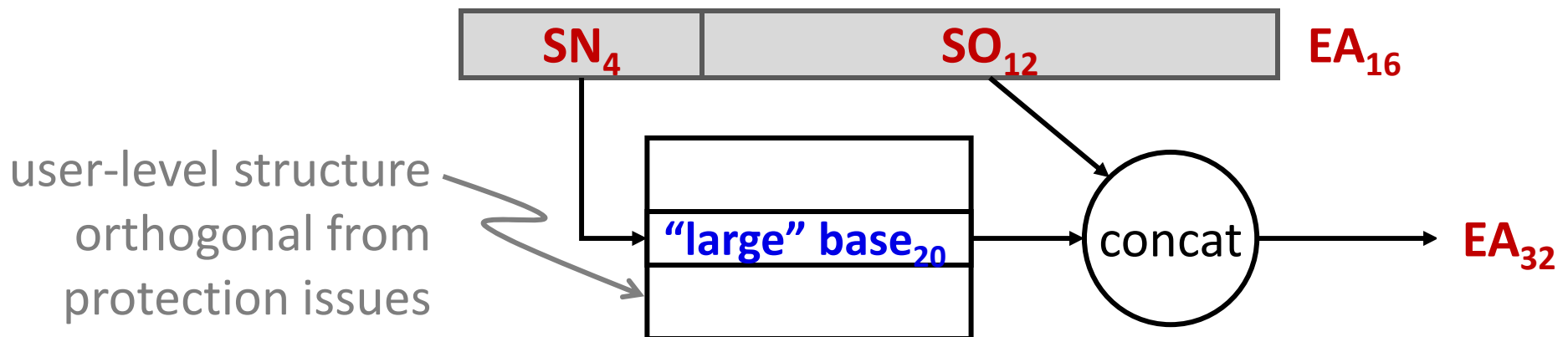
- For example

- normal data segment \Rightarrow **RW**(!**E**)
- static shared data segment \Rightarrow **R**(!**W**)(!**E**)
- code segment \Rightarrow **R**(!**W**)**E** *self modifying code?*
- illegal segment \Rightarrow (!**R**)(!**W**)(!**E**) *what for?*

Access violation exception brings OS into play

Aside: Another (ab)use of segments

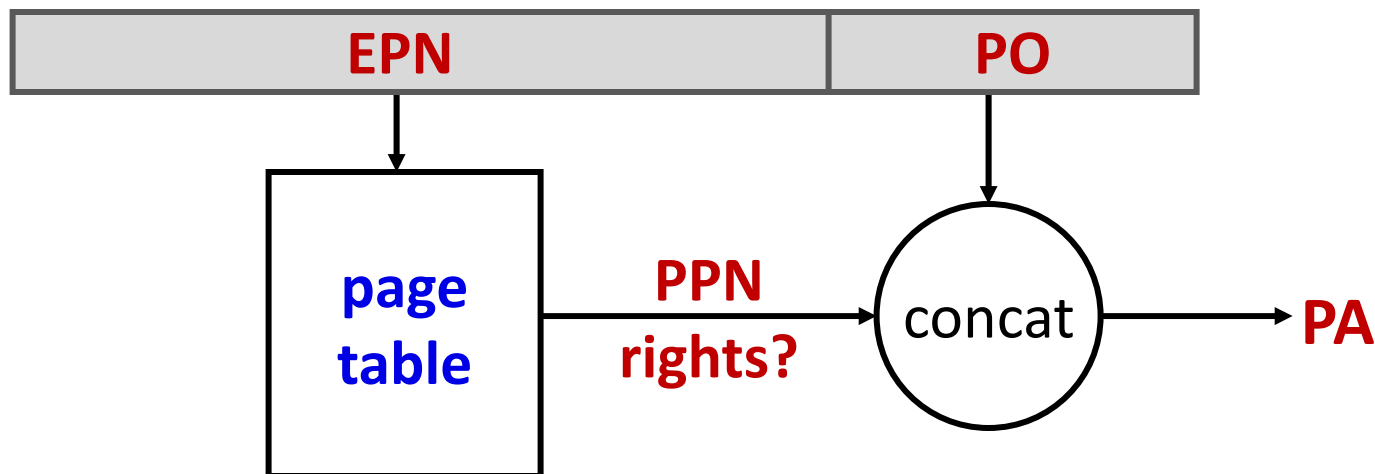
- Extend old ISA to give new applications a large address space while stay compatible with old
- “User-managed” segmented addressing $SA \equiv EA_{small}$
 - old application use identity mapping in table; old applications unaware of segments
 - new application reloads table at run time to access different regions in EA_{large} ; unequal access to active vs inactive regions



Paged Address Space

- Divide **PA** and **EA** space into equal, fixed size segments known as “page frames”
historically 4KByte pages
- **EA** and **PA** are interpreted as page number (**PN**) and page offset (**PO**)
 - page table translates **EPN** to **PPN**; **EPO=PPO**
 - **PA**=**{PPN,PO}**

many small
pages good for
managing
allocation



Fragmentation

- External fragmentation by segments
 - plenty of unallocated DRAM but none in contiguous region of a sufficient size
 - paged memory eliminates external fragmentation
- Internal fragmentation of pages
 - entire page (4KByte) is allocated; unused bytes go to waste
 - smaller page size reduces internal fragmentation
 - modern ISA moving to larger page sizes (MBytes) in addition to 4KBytes

Segments and pages not meant for the same role

Demand Paging

- Use main memory and “swap” disk as automatically managed memory hierarchy levels
analogous to cache vs. main memory
- Early attempts
 - von Neumann already described manual memory hierarchies
 - Brookner’s interpretive coding, 1960:
program interpreter managed paging between a 40KByte main memory and a 640KByte drum
 - Atlas, 1962:
hardware managed paging between 32-page core memory and 192-page drum (512 word/page)

Demand Paging: just like caching

- **M** bytes of storage (DRAM+Disk), keep most frequently used **C** bytes in DRAM where $C < M$
- Same basic issues as before
 - (1) where to place a page in DRAM or disk?
 - (2) how to find a page in DRAM or disk?
 - (3) when to bring a page into DRAM from disk?
 - (4) which page to evict from DRAM to disk to free-up DRAM for new pages?
- Key conceptual difference: swap vs. cache
 - DRAM doesn't hold “copies” of what is on disk
 - a page in **M** either in DRAM or on disk
 - address not bound to 1 location for all time

Demand Paging: not at all like caching

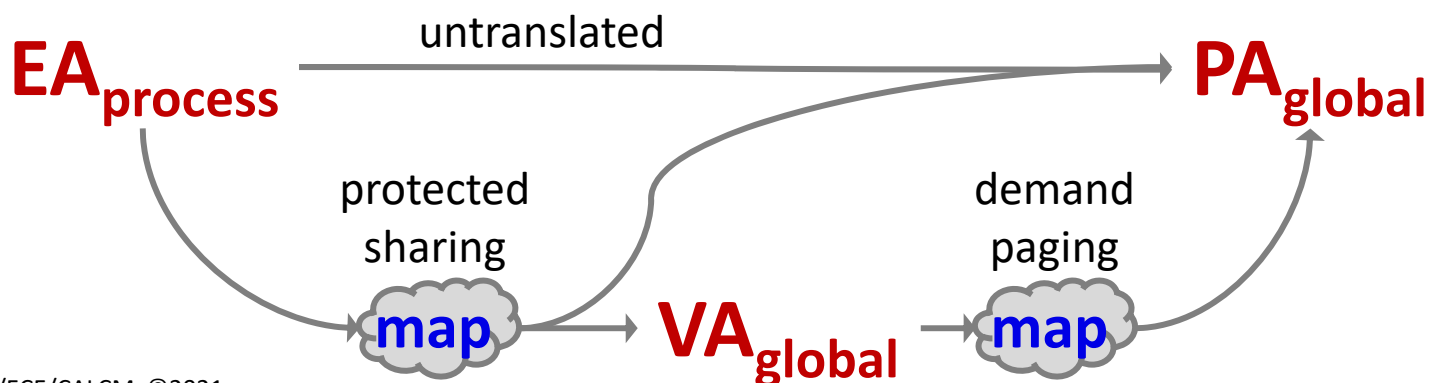
- Drastically different size and time scale leads to drastically different implementation choices

	L1 Cache	L2 Cache	Demand Paging
capacity	10s KByte	MByte	GByte
block size	10s Byte	\geq L1	4K~4M Byte
hit time	few cyc	few 10s cyc	few 100s cyc
miss penalty	few 10s cyc	few 100s cyc	10 msec
miss rate	0.1~10%	$\ll 0.1\%$	0.00001~0.001%
(per mem reference not per cache access)			
hit handling	HW	HW	HW
miss handling	HW	HW	SW

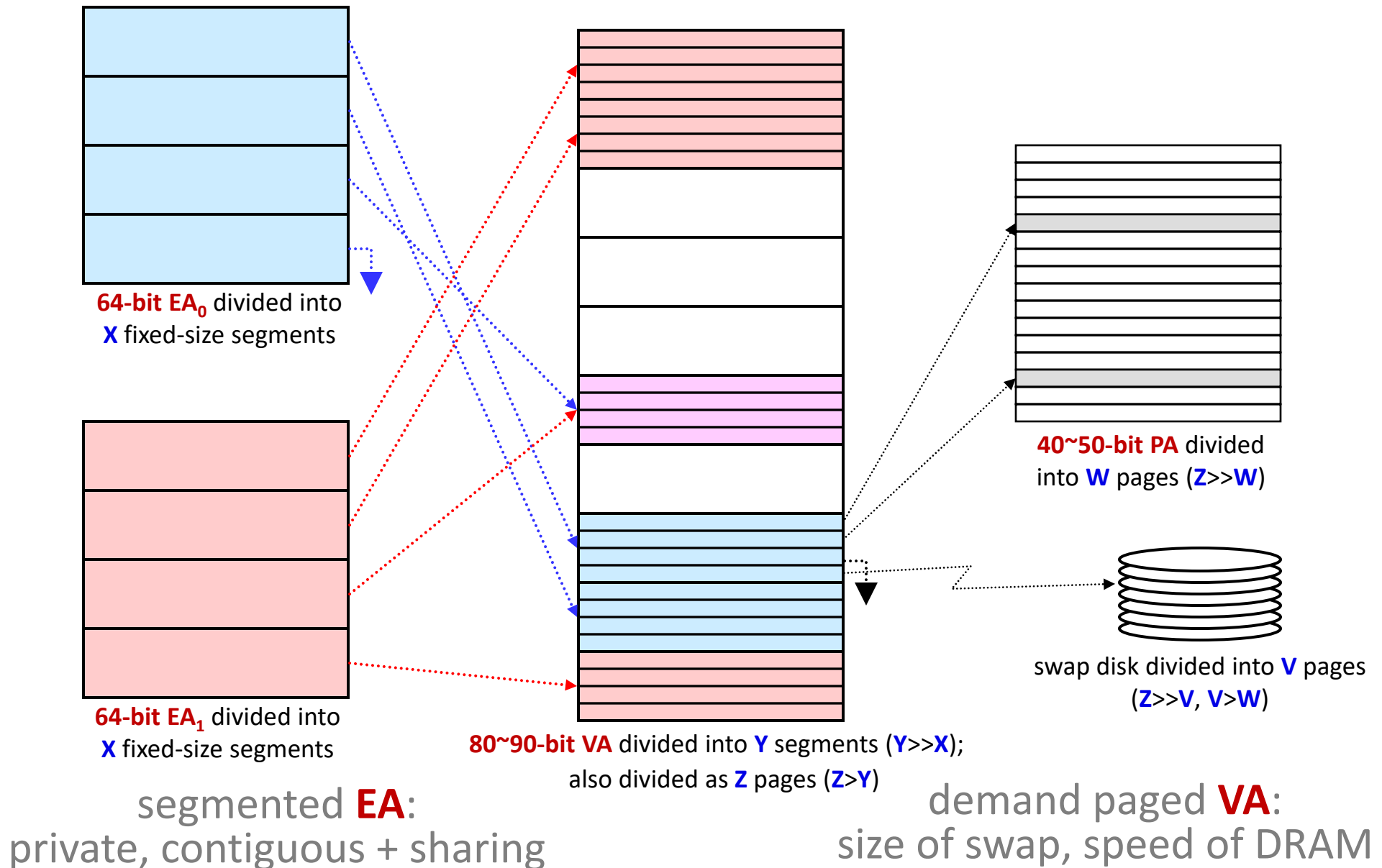
Hit time, miss penalty, miss rate not independent!!

Don't use “VM” to mean everything

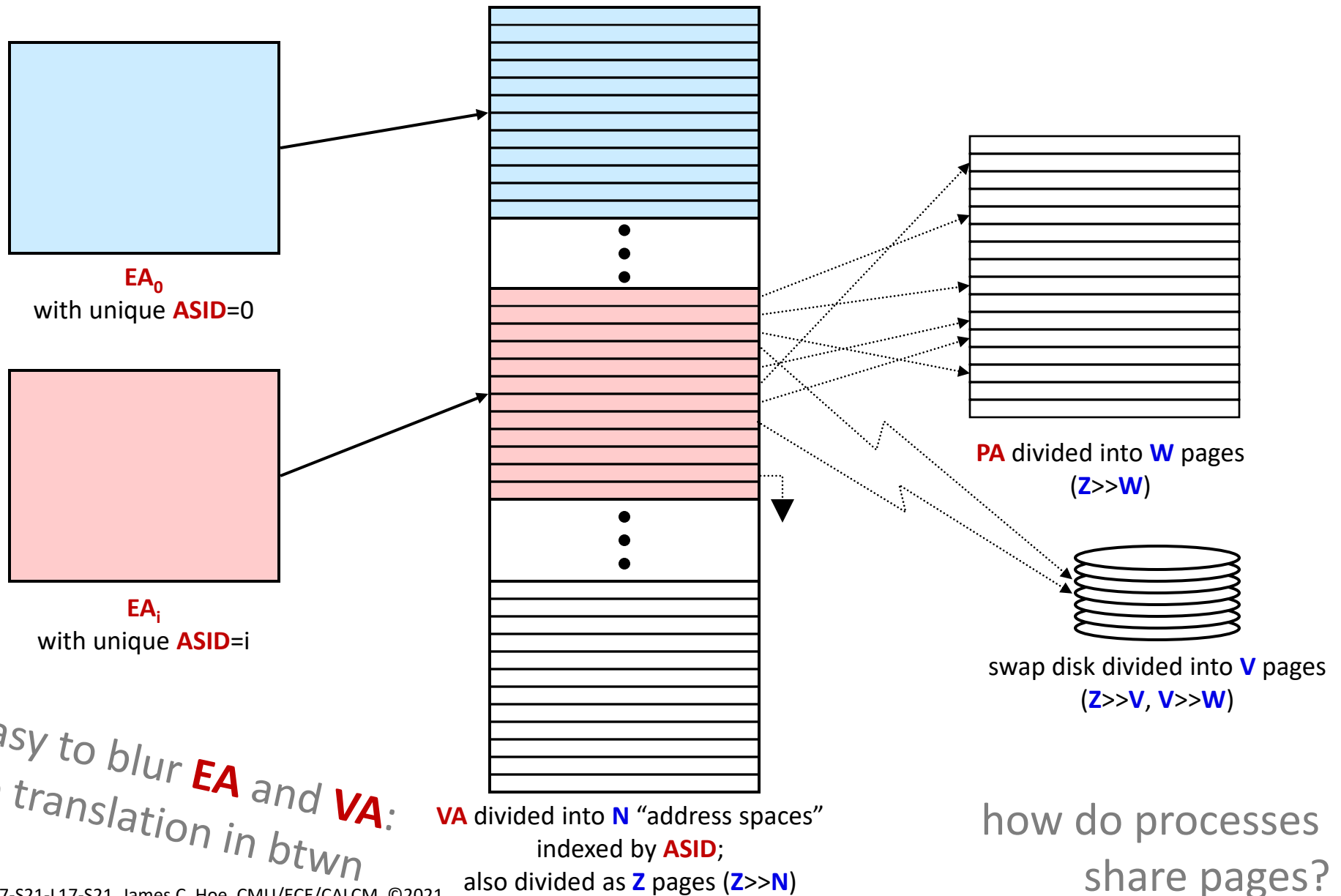
- Effective Address (**EA**): emitted by user instructions in a per-process space (**protection**)
- Physical Address (**PA**): corresponds to actual storage locations on DRAM or on swap disk
- Virtual Address (**VA**): refers to locations in a single system-wide, large, linear address space; not all locations in **VA** space have physical backing (**demand paging**)



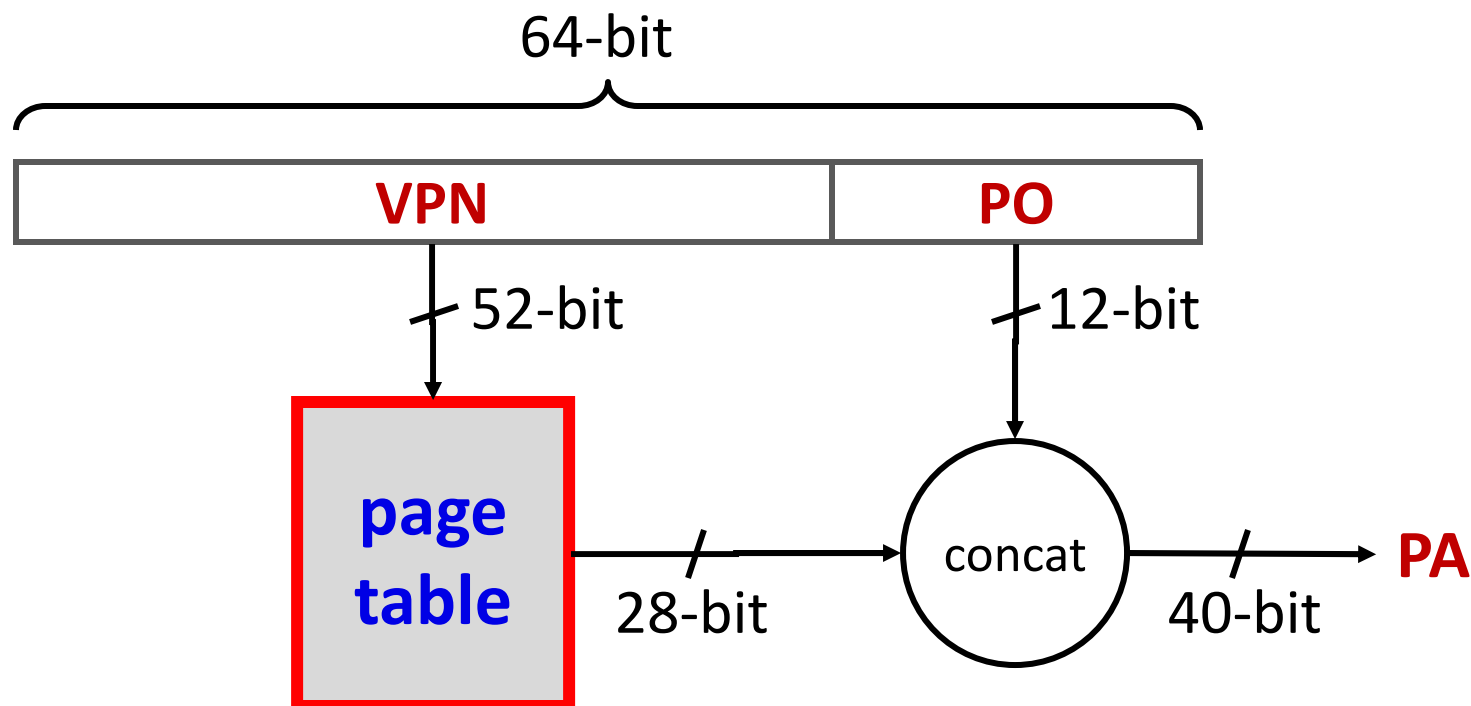
EA, VA and PA (IBM Power view)



EA, VA and PA (almost everyone else)



Just one more thing: How large is the page table?



- A page table holds mapping from **VPN** to **PPN**
- Suppose 64-bit **VA** and 40-bit **PA**, how large is the page table? 2^{52} entries \times ~ 4 bytes $\approx 16 \times 10^{15}$ Bytes

And that is for just one process!!?

How large should it be?

- Don't need to track entire **VA** space
 - total allocated **VA** space is 2^{64} bytes x # processes, but most of which not backed by storage
 - can't use more memory locations than physically exist (DRAM and swap disk)
- A clever page table should scale linearly with physical storage size and not **VA** space size
- Table cannot be too convoluted
 - a page table is accessed not infrequently
 - a page table should be “walkable” quickly in HW

Two dominant schemes in use today:

hierarchical page table and *hashed page table*