COL216
Computer Architecture

Memory Organization

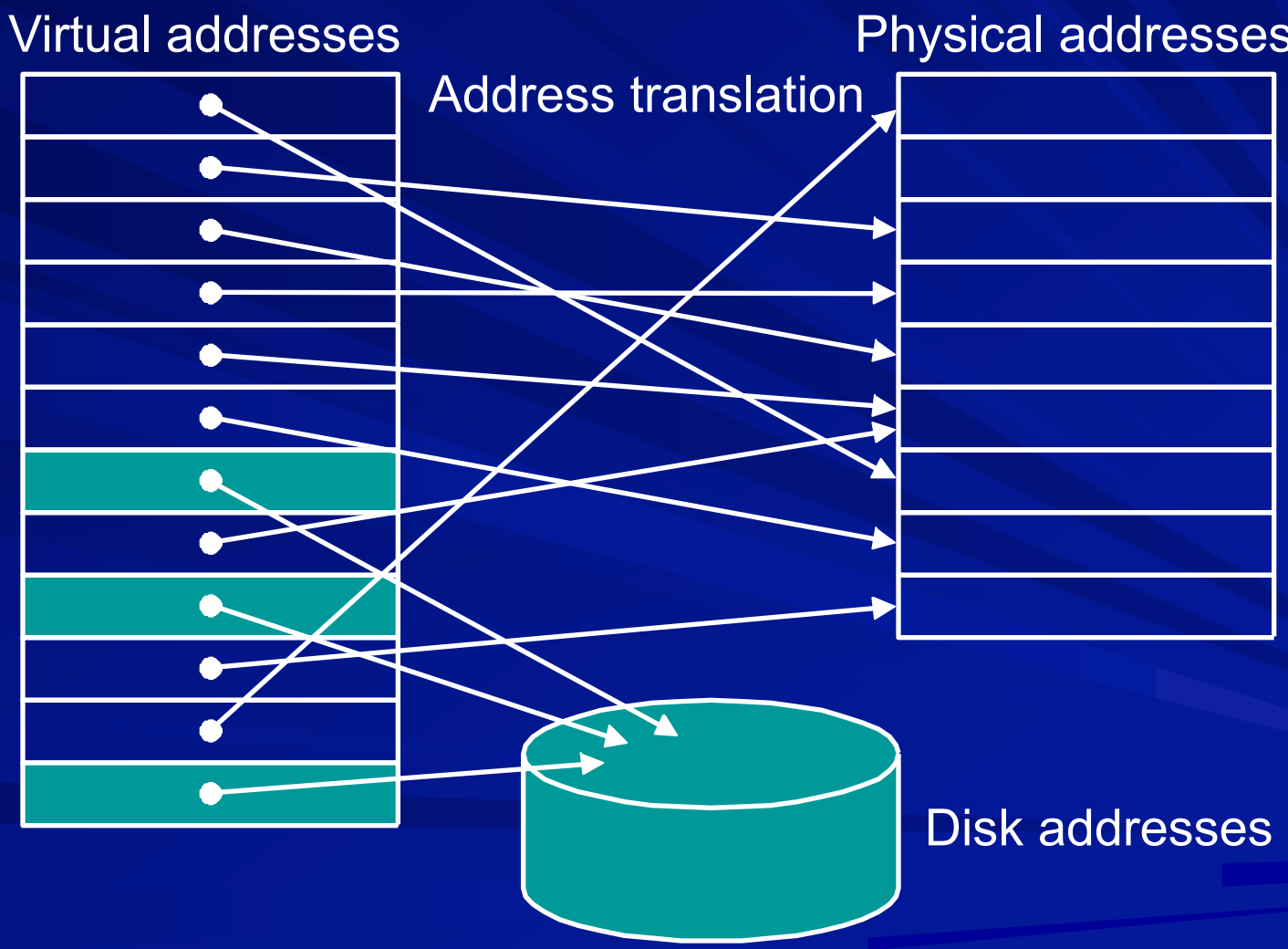
Virtual Memory

12th March 2022

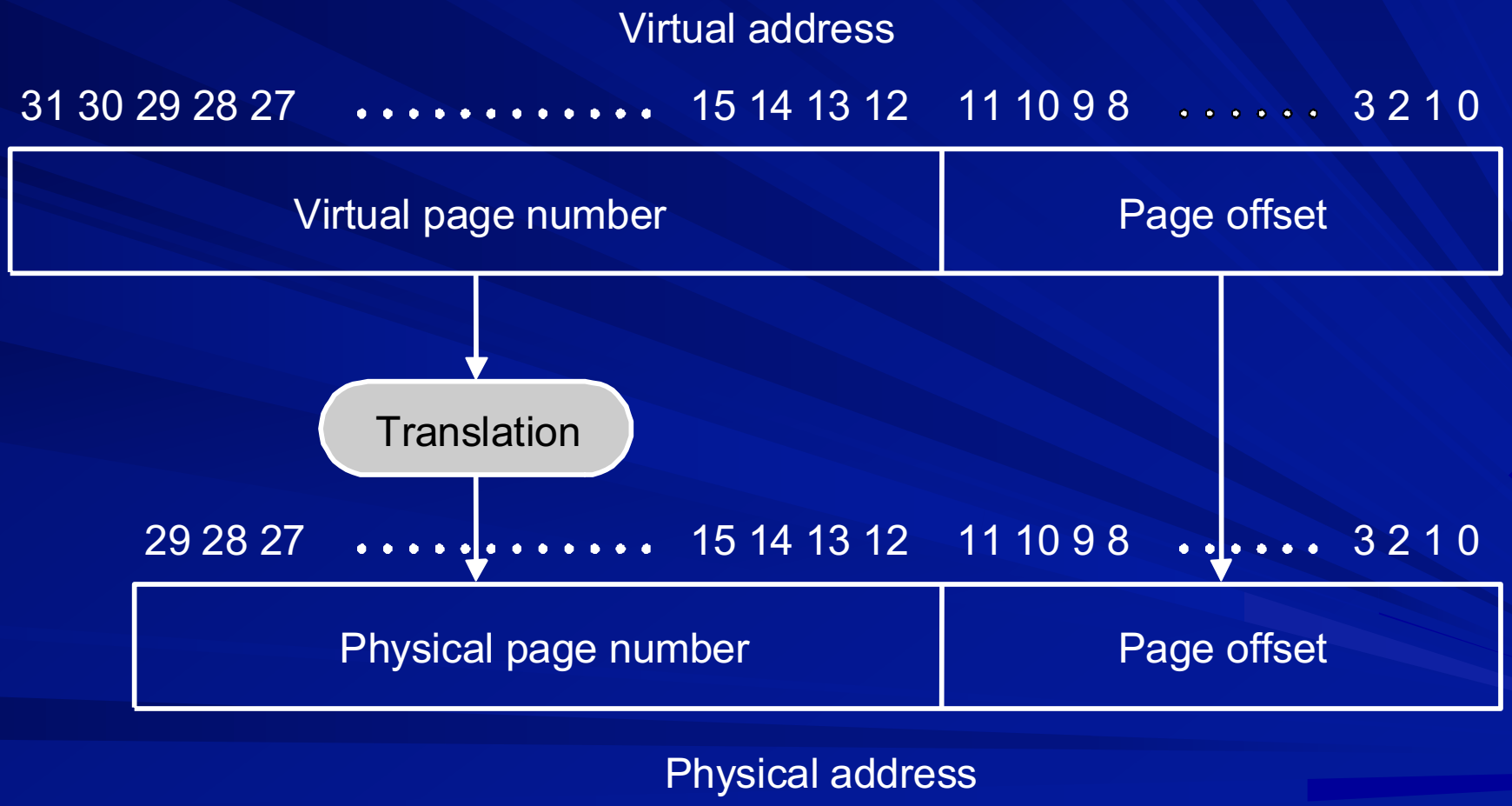
Virtual memory objectives

- To overcome the limitation of the physical memory size
- To allow multiple programs to share memory
  - Protection
  - Program relocation

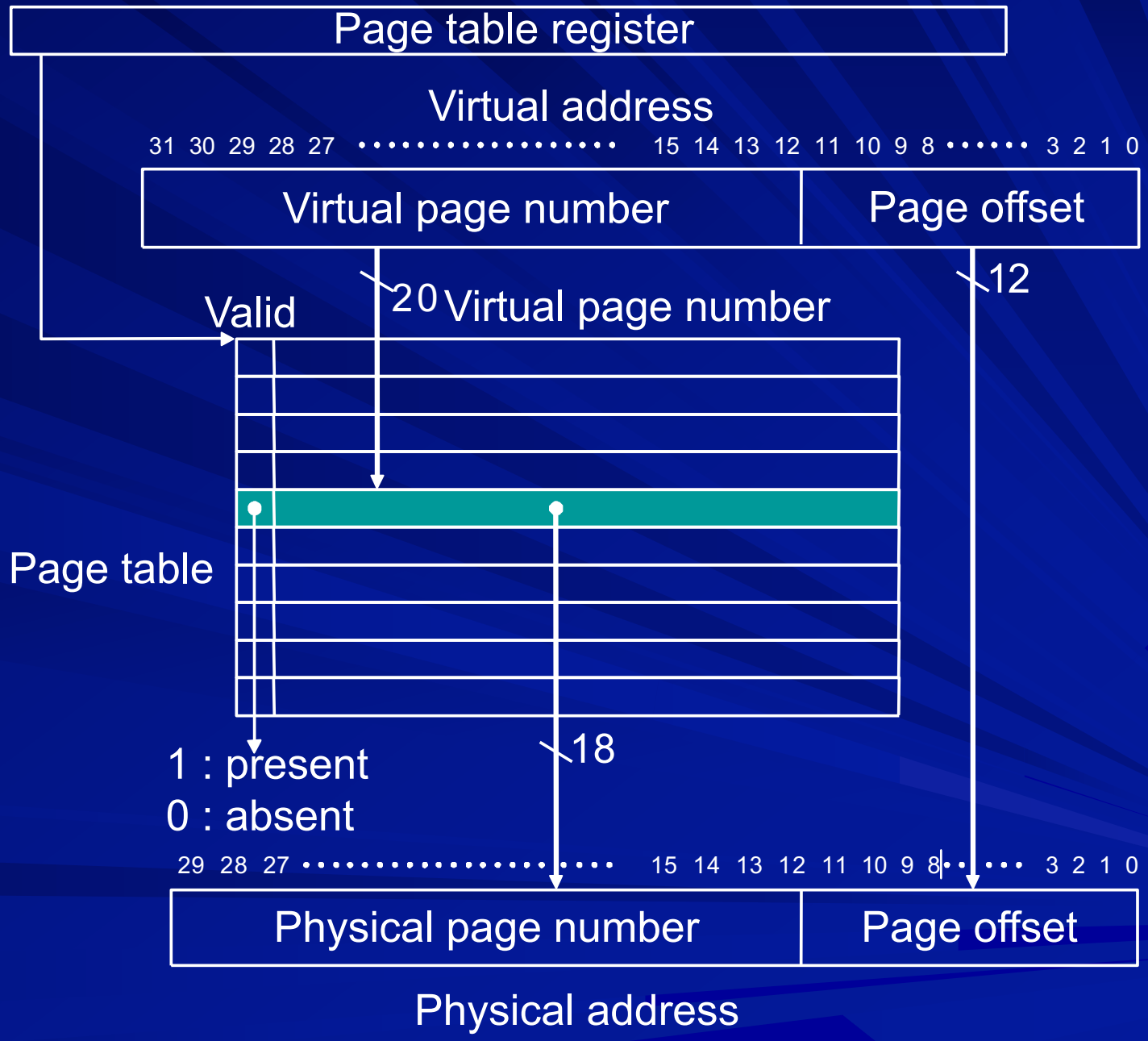
Virtual to physical mapping



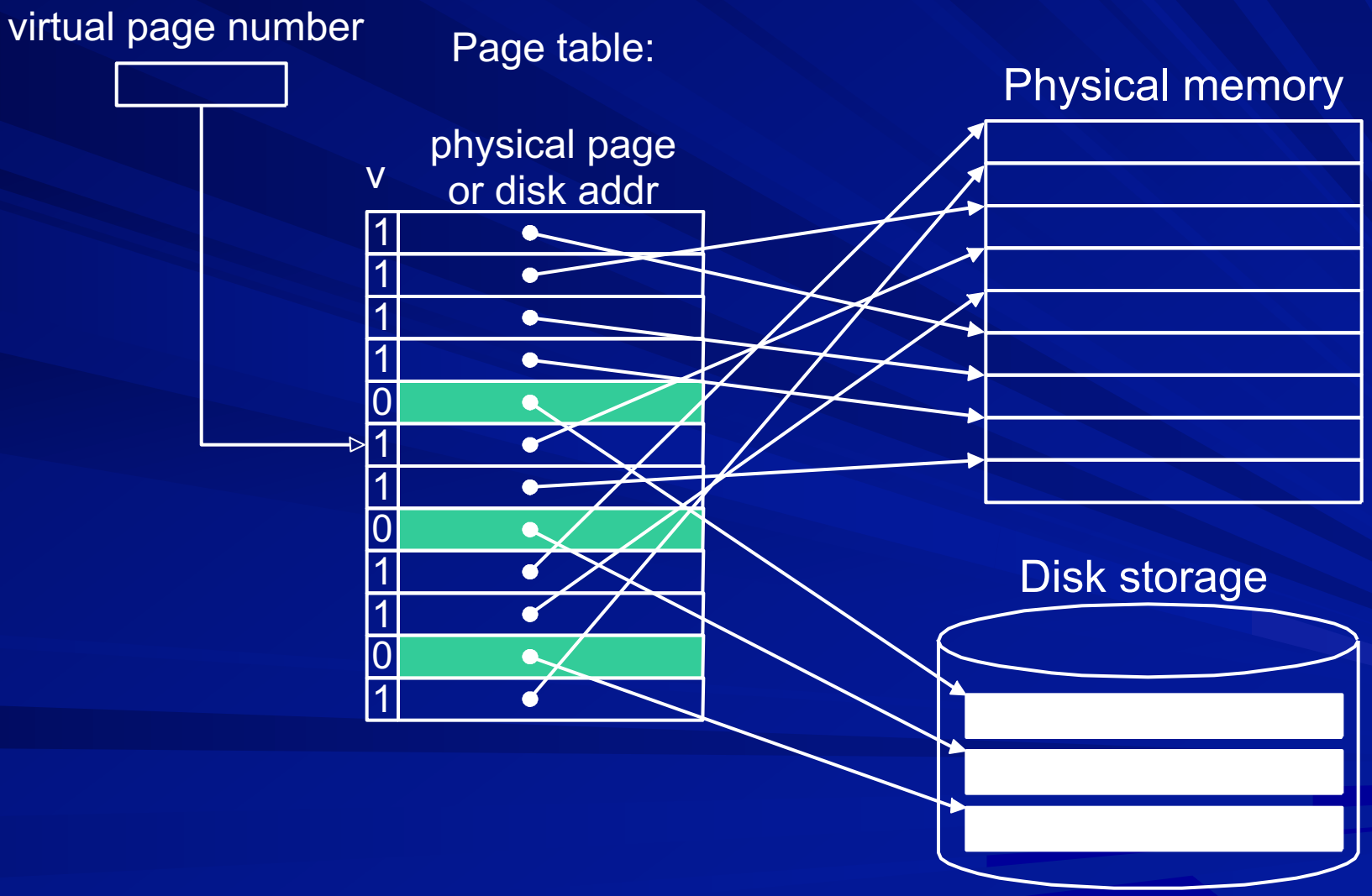
Address translation



Page table


Page table register
Virtual address
31 30 29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Virtual page number
Page table
1 : present
0 : absent
18
29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Physical page number
Page offset
Physical address

Mapping with page table



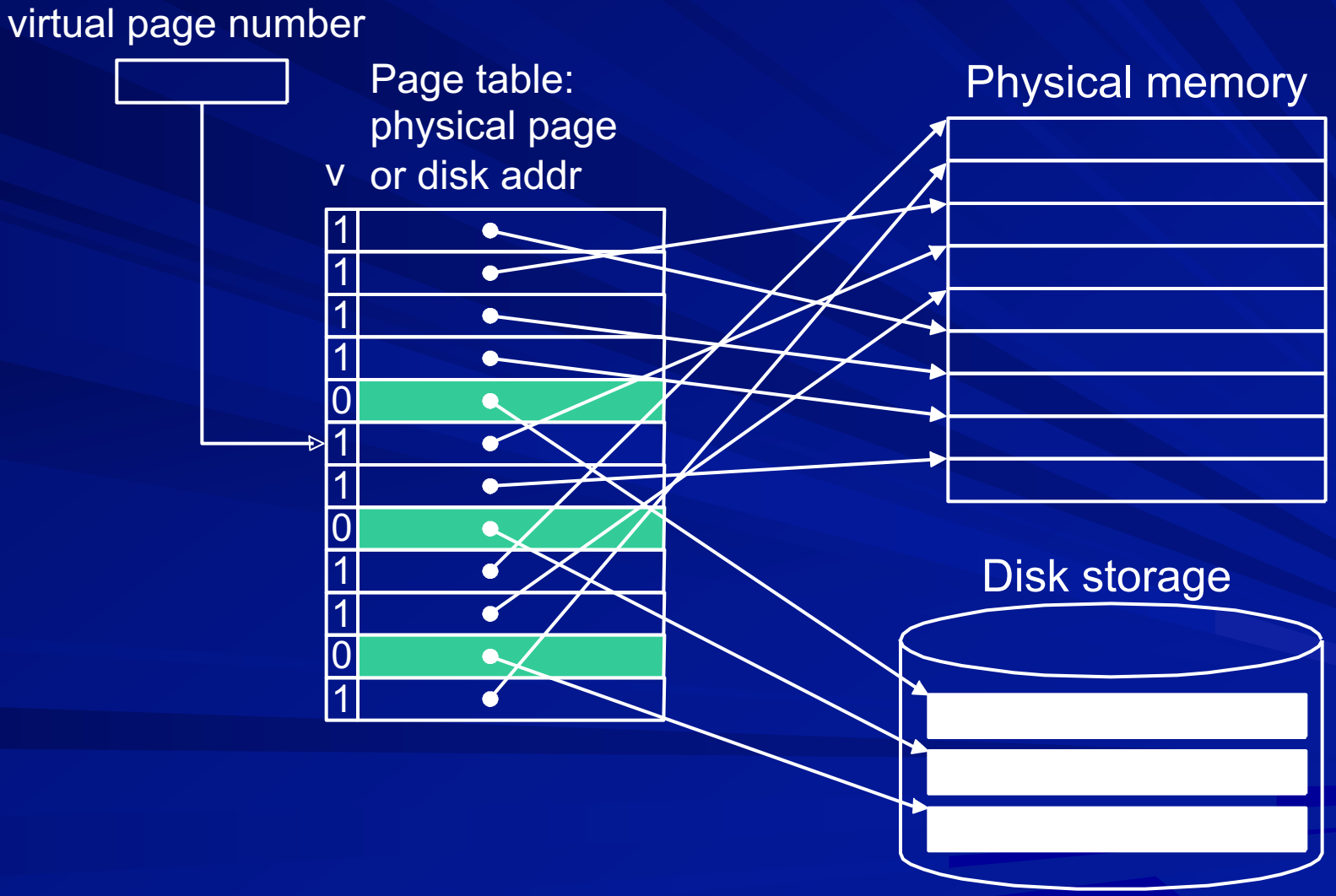
Virtual memory vs cache : similar but important differences

- Speed difference
  - cache and main memory : one order of magnitude
  - main memory and HDD : several orders of magnitude
- Response to a miss
  - cache : must be handled by h/w, can't afford to switch context
  - VM : can't keep CPU waiting, must switch context, can be conveniently handled by s/w
- Terminology differences
  - page vs block(cache line), page fault vs miss, page table vs cache directory etc

Differences contd.

- Miss rate
  - VM can only afford extremely low miss rates
  - main memory much larger than cache, implies much lower miss rate
- Policies
  - pages much larger than blocks (~4KB - 16 KB) : capture large spatial locality, amortize transfer time
  - fully flexible mapping (assoc mem not used)
  - always write back
  - good approximation to LRU

Back to page table



Where is page table stored?

How big is it?

- Suppose virtual address = 32 bits, page size = 4 KB, page table entry = 4 B
- then number of page table entries = number of pages = $2^{32} / 2^{12} = 2^{20}$
- page table size = $2^{20} \times 2^{2} = 4$ MB
- hundreds of processes

Store in dedicated memory? main memory?

Handling large page tables

- Bound the page table size
- Exploit sparseness
- Use multiple levels
- Page the page table

Bounding the page table size

- Keep a bound register
  - Allocate as much space as necessary
  - Address space expands in one direction
- Split the virtual space into two parts, each with its own bound, e.g. heap + stack
  - Two segments can grow independently
- Sparseness not exploited

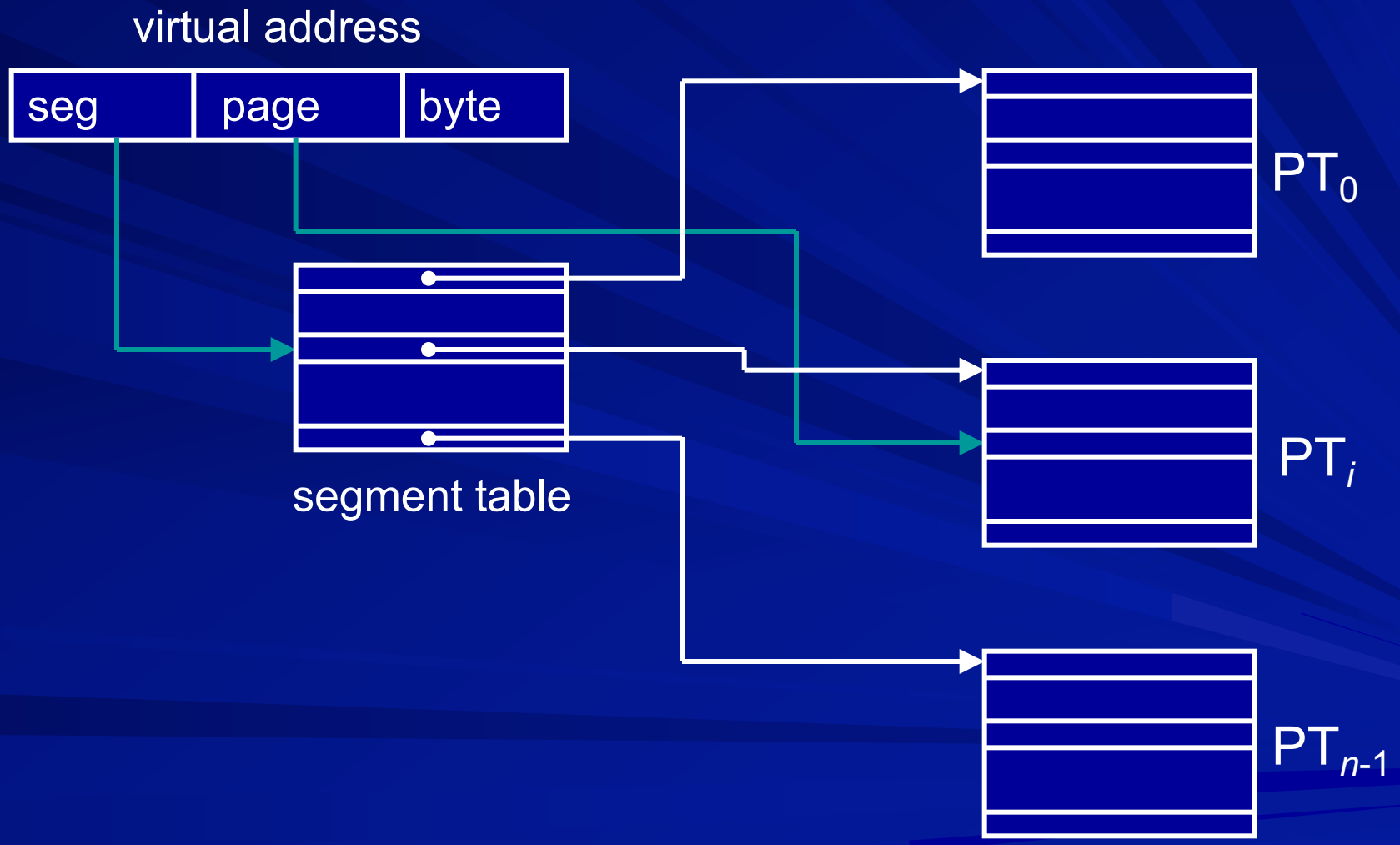
Exploit sparseness

- Page table: one entry per virtual page, Cache: one entry per cache block
  - Associative memory impractical for page table
- Hashing can be used - inverted page table
  - Access is more complex than simply indexing into the page table

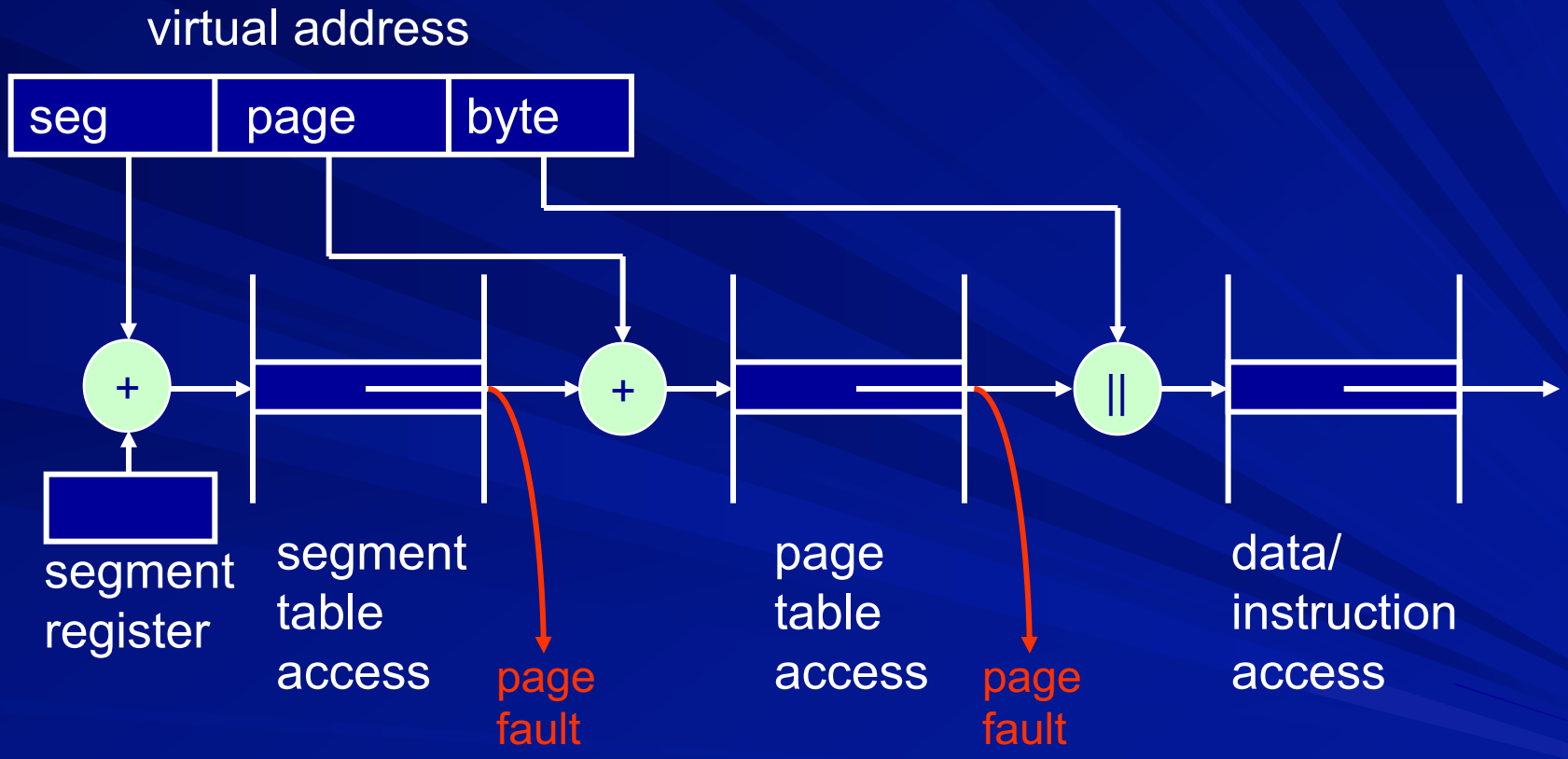
Two level page tables

- Virtual space divided into segments
- A smaller page table for each segment
- All page tables need not be in physical memory
- A segment table keeps track of where the page tables of different segments are located

2 Level page table structure



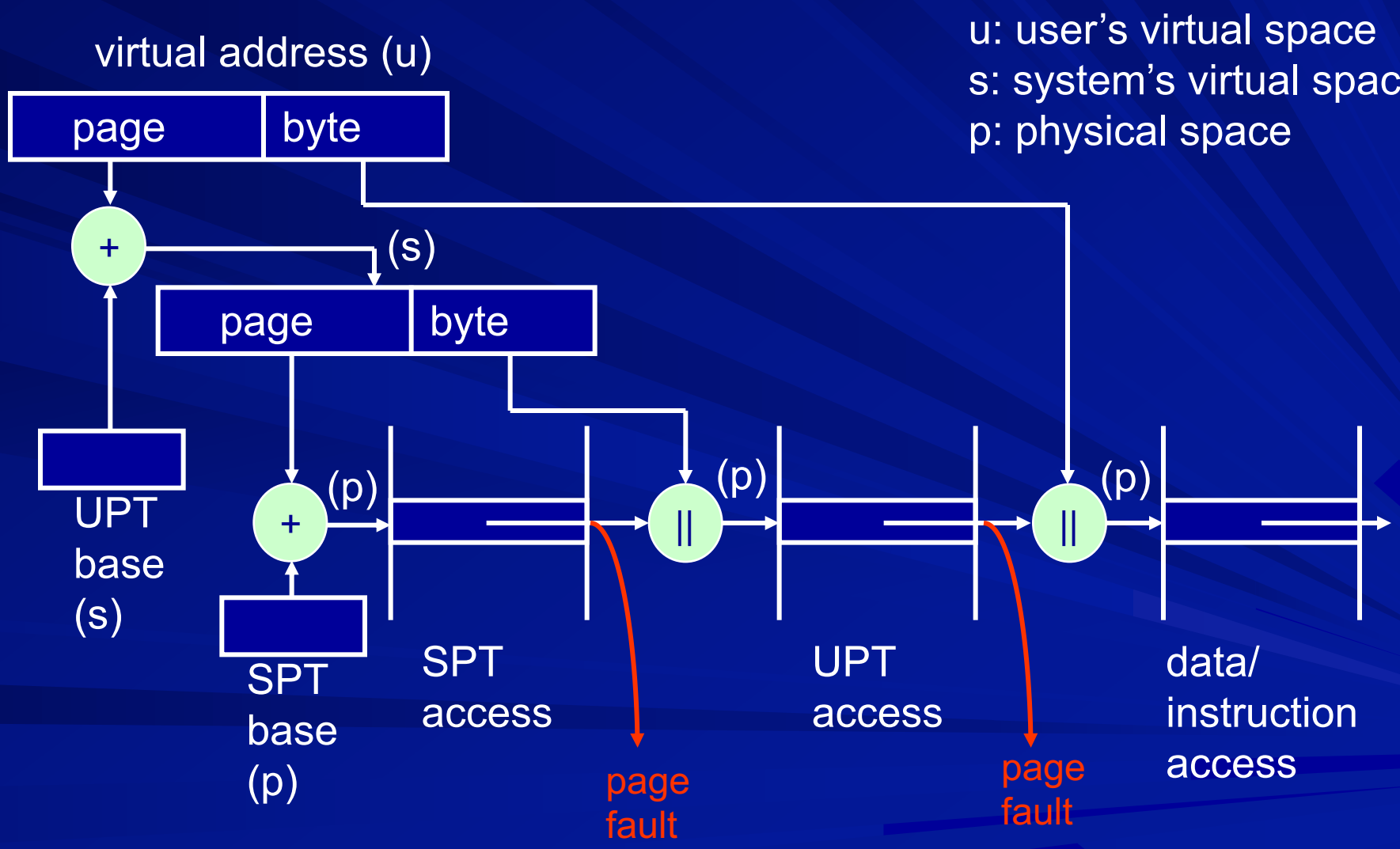
Memory access with 2 level page table

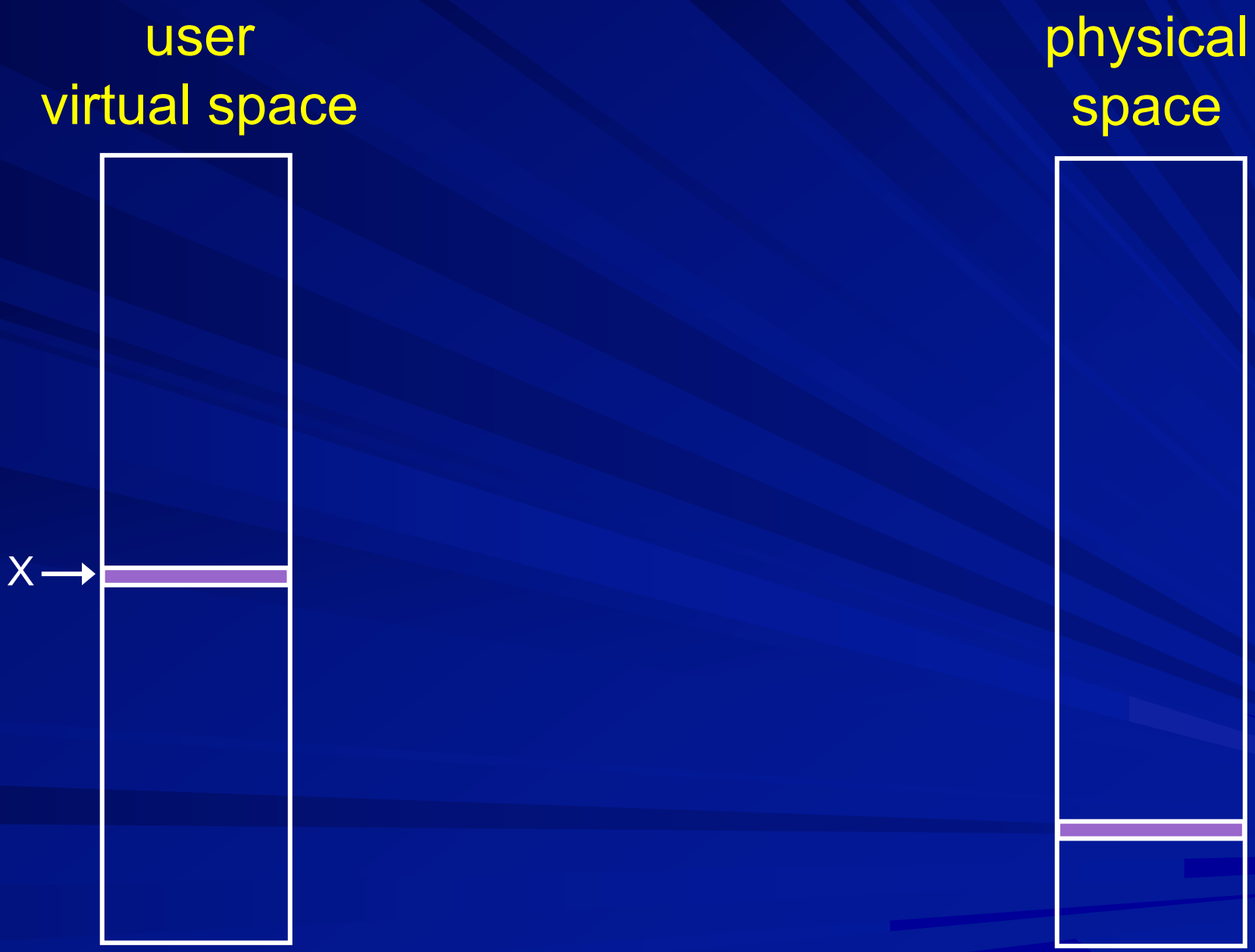


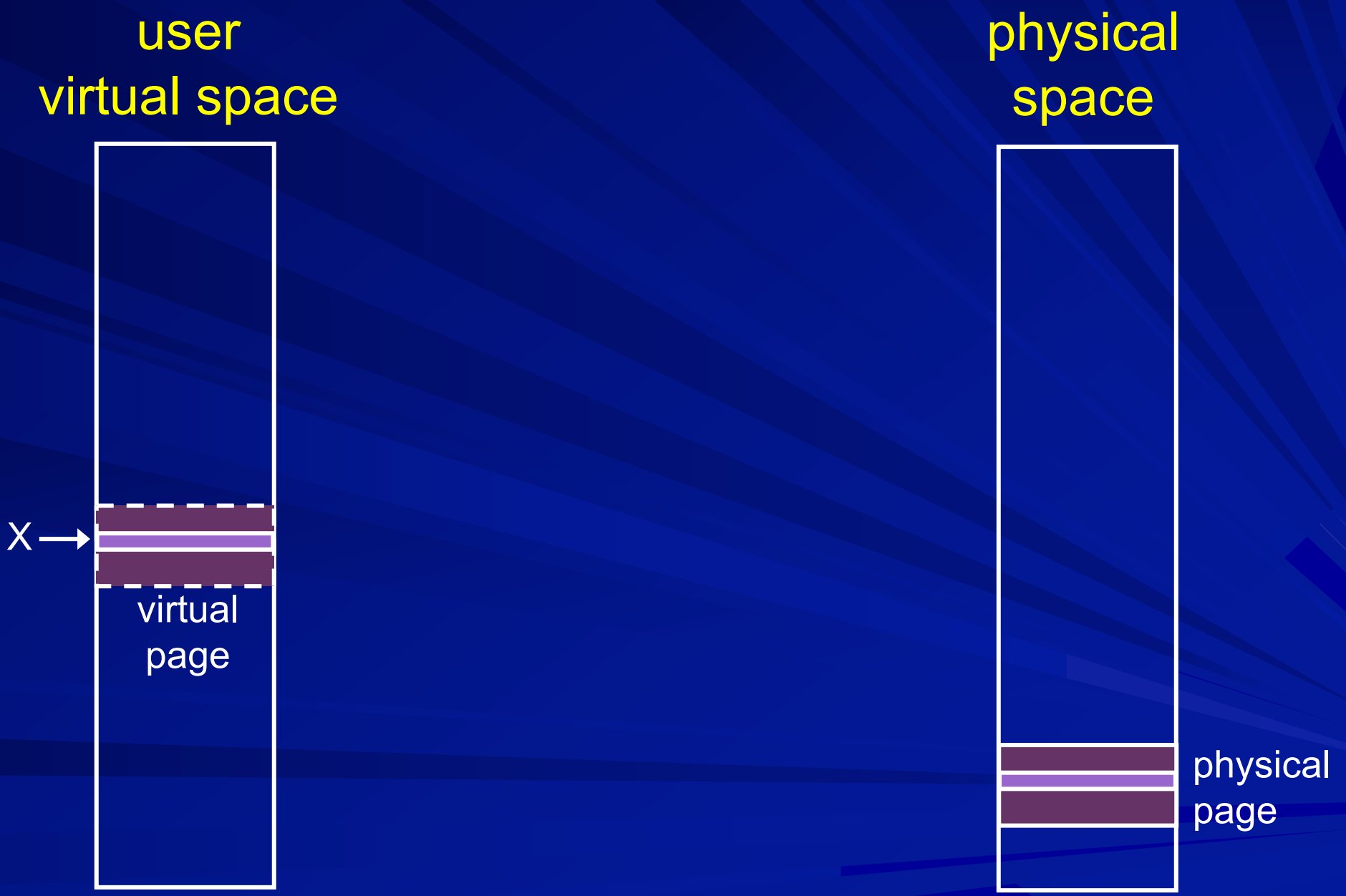
Keep page table in virtual space

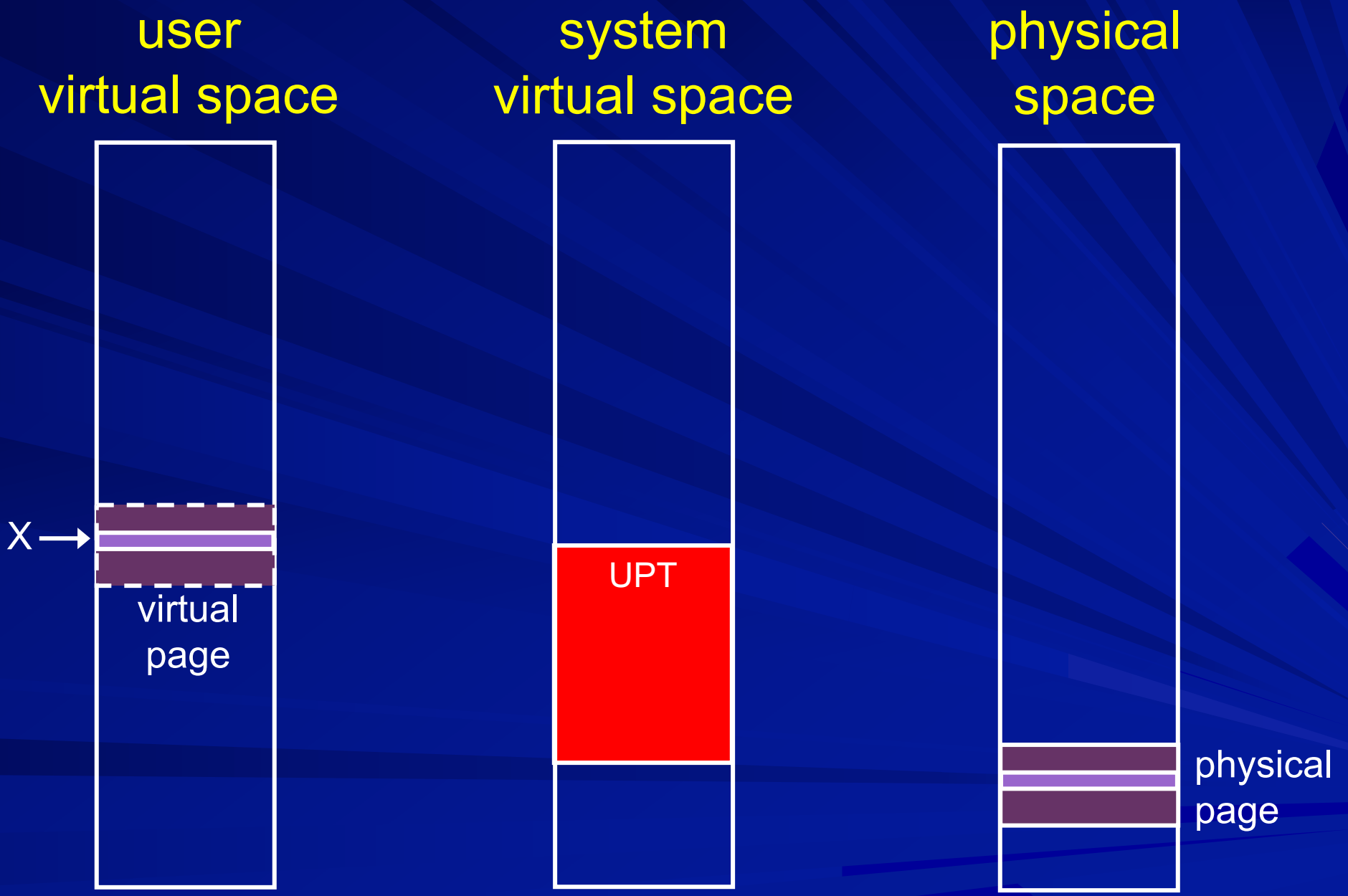
- Page table is paged!
- Only active pages of page table are brought into physical memory
- How do you locate these pages? - need another page table!
- User page tables are kept in system's virtual space
- System's page table (or at least a part of it) must be kept in physical memory and accessed directly

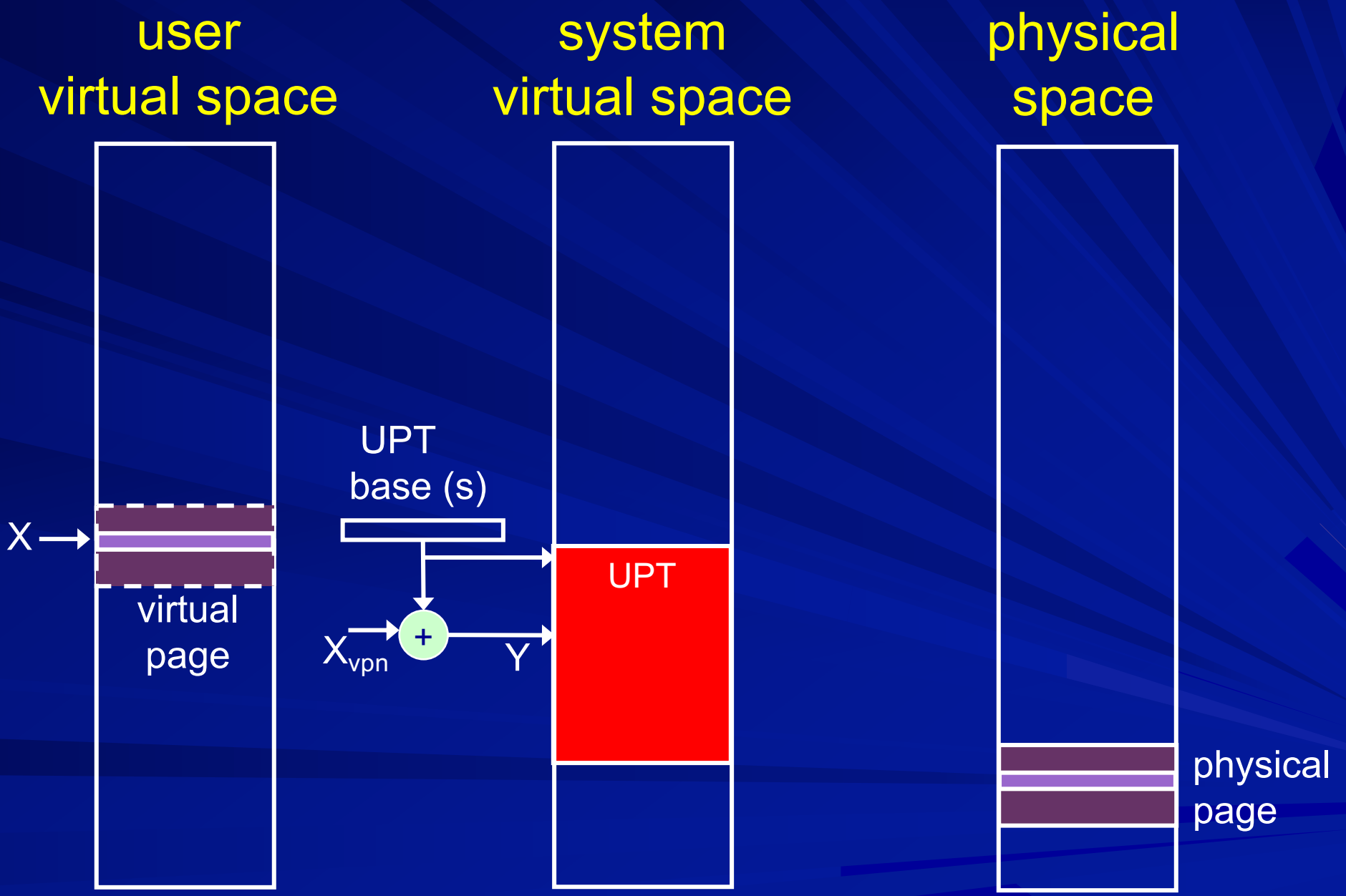
Memory access with paged page table

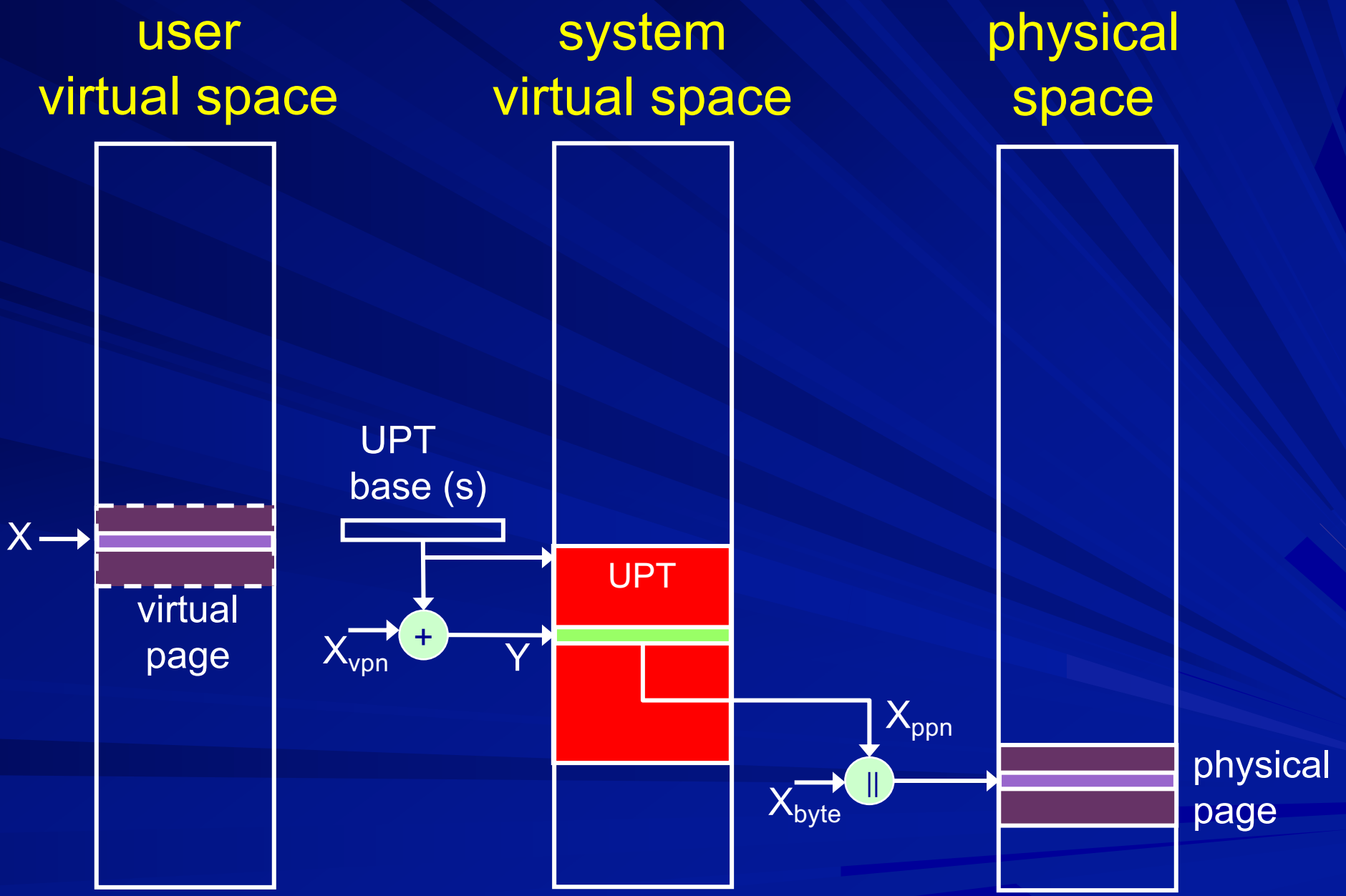


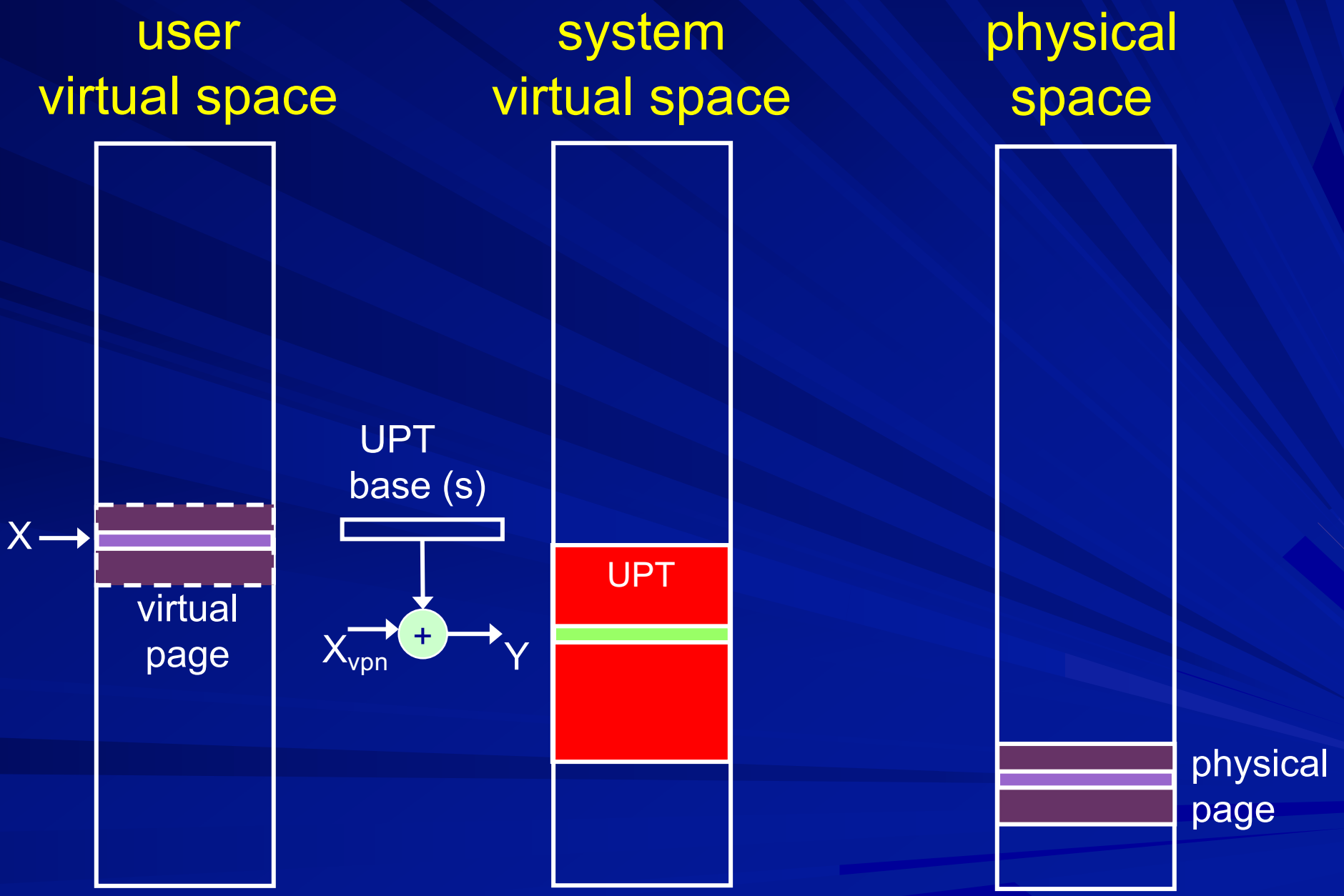

user
virtual space
physical
space
X

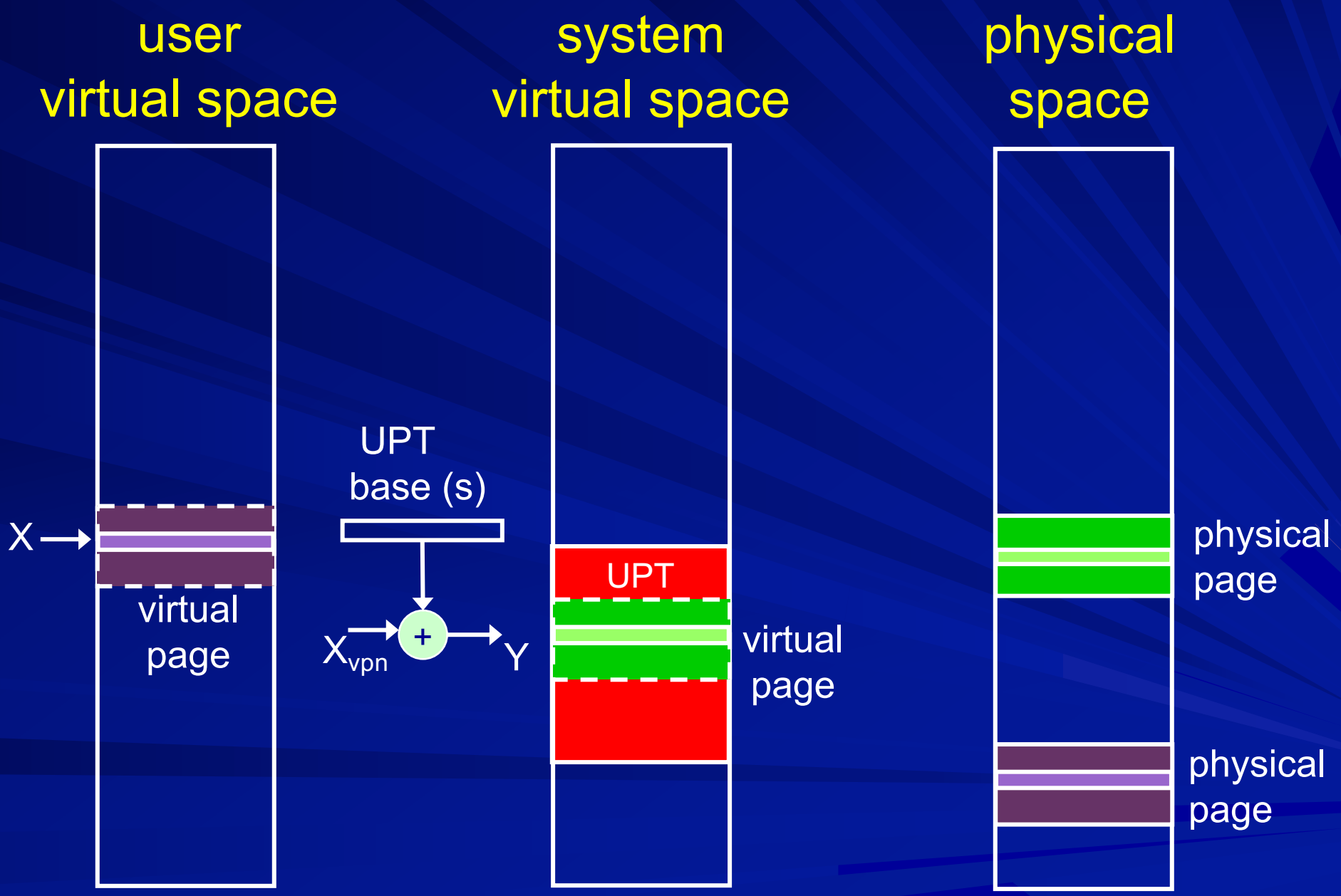

user
virtual space
X
virtual
page
physical
space
physical
page

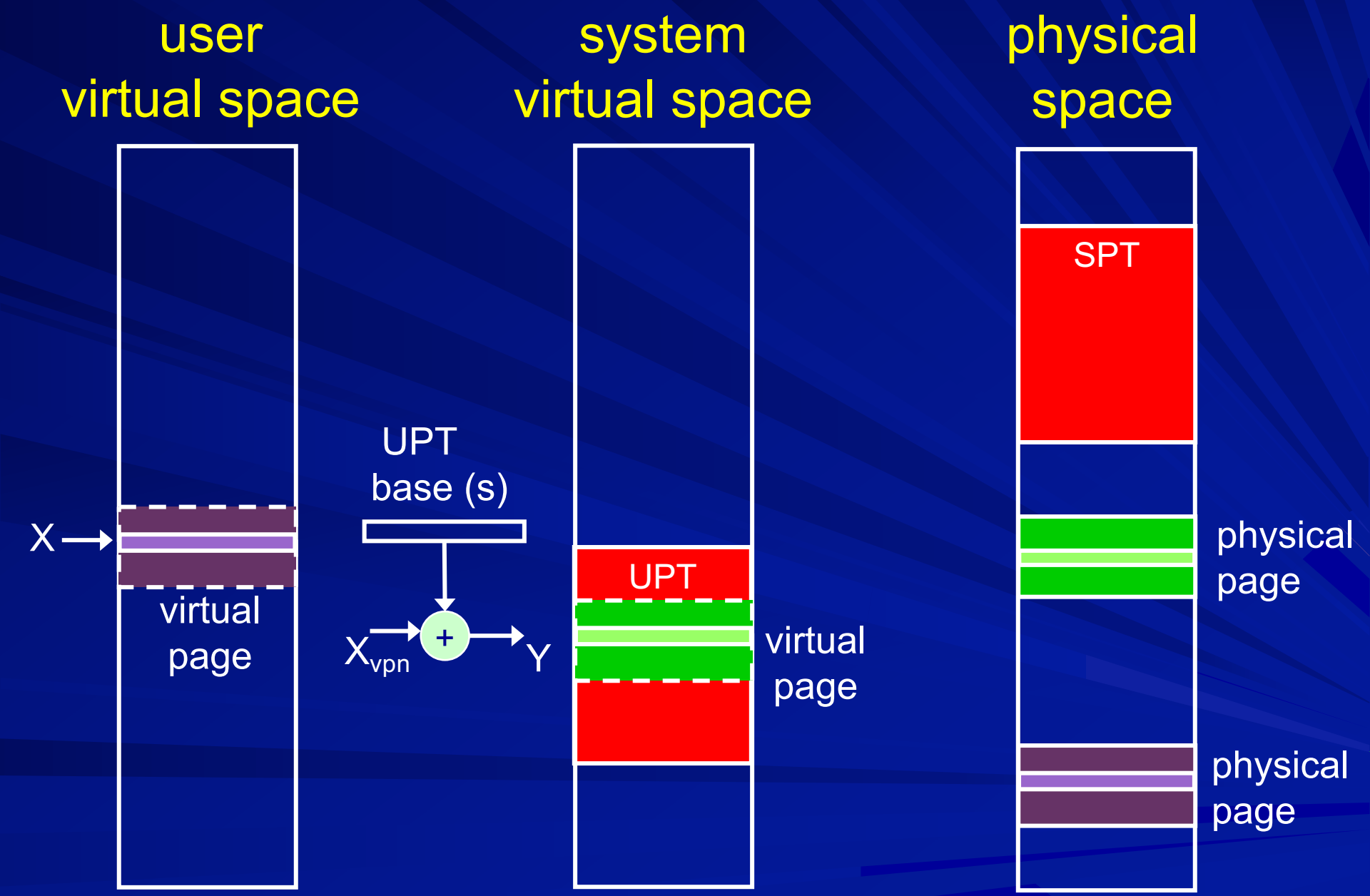

user
virtual space
system
virtual space
physical
space
X
virtual
page
UPT
physical
page

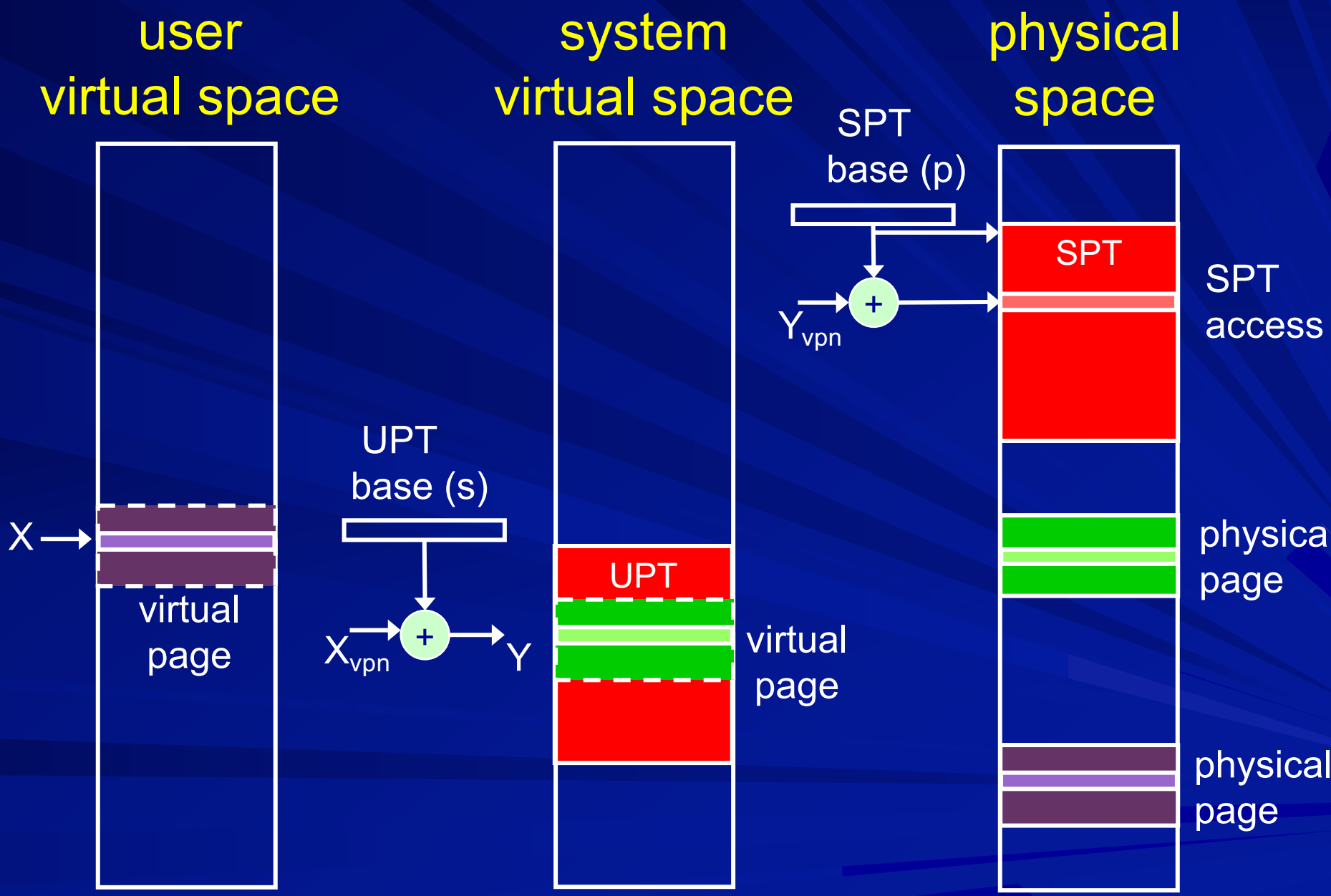

user
virtual space
system
virtual space
physical
space
X
virtual
page
UPT
base (s)
UPT
+
X_{vpn}
Y
physical
page

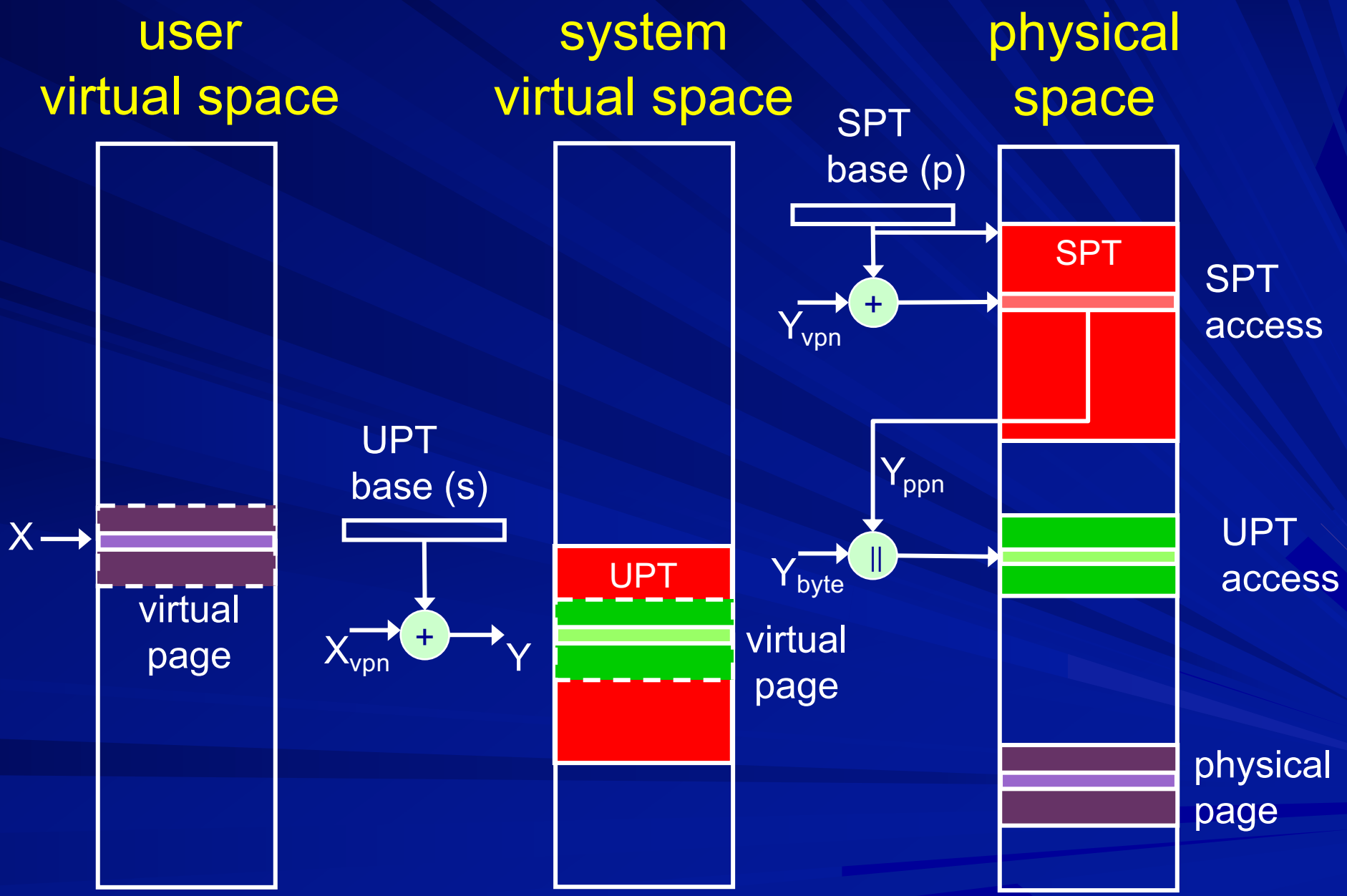

user
virtual space
system
virtual space
physical
space
X
virtual
page
UPT
base (s)
UPT
+
X_{vpn}
Y
X_{ppn}
||
X_{byte}
physical
page

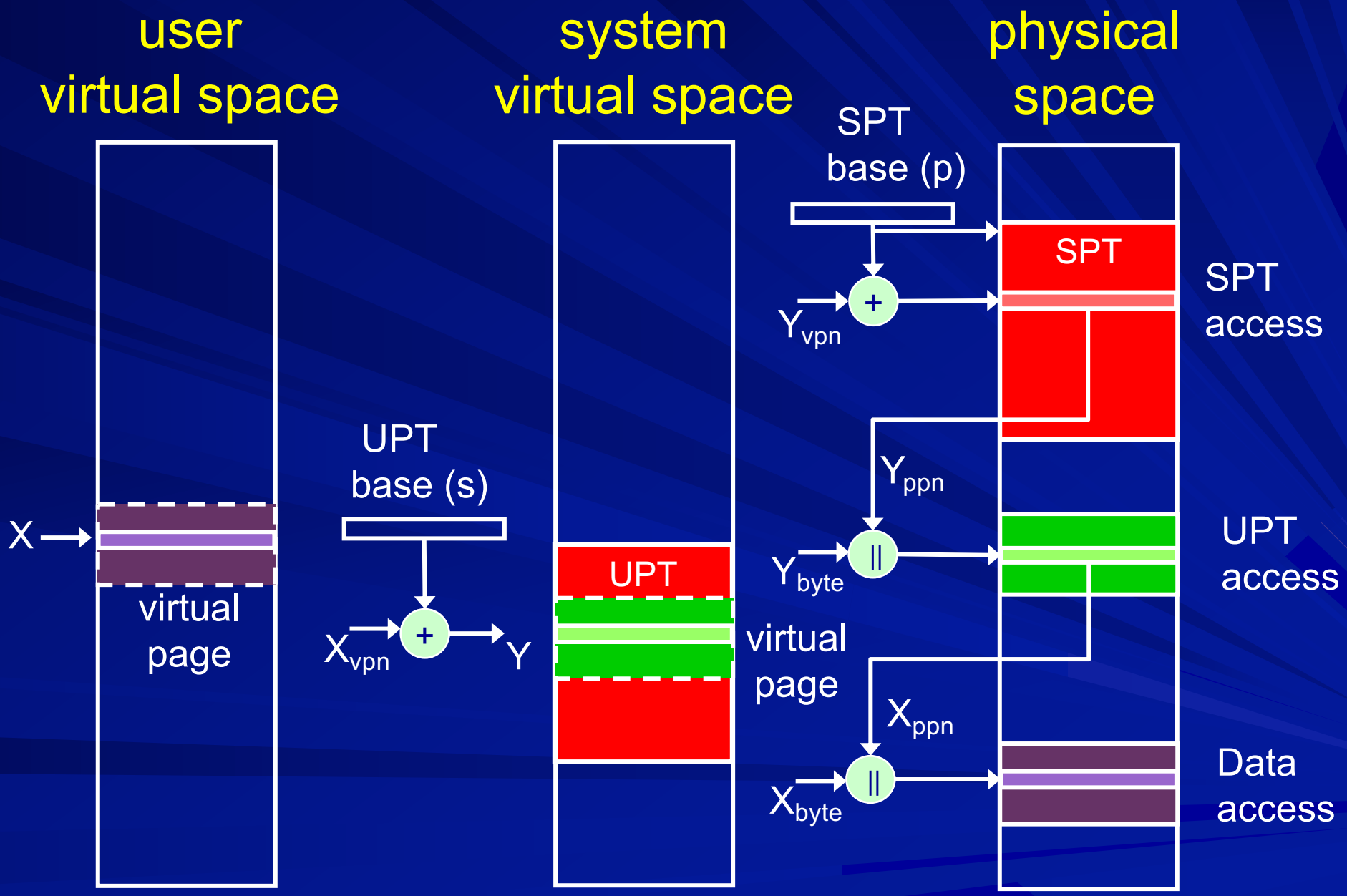

user
virtual space
system
virtual space
physical
space
X
virtual
page
UPT
base (s)
+
X_{vpn}
Y
UPT
physical
page


user
virtual space
system
virtual space
physical
space
X
virtual
page
UPT
base (s)
X_{vpn}
+
Y
UPT
virtual
page
physical
page
physical
page


user
virtual space
system
virtual space
physical
space
X
virtual
page
UPT
base (s)
X vpn
+
Y
UPT
virtual
page
SPT
physical
page
physical
page


user
virtual space
system
virtual space
physical
space
SPT
base (p)
SPT
SPT
access
Y_{vpn}
+
UPT
base (s)
X
virtual
page
UPT
physical
page
X_{vpn}
+
Y
virtual
page
physical
page


user
virtual space
system
virtual space
physical
space
SPT
base (p)
SPT
SPT
access
Y_{vpn}
+
UPT
base (s)
X
virtual
page
Y_{ppn}
UPT
UPT
access
Y_{byte}
||
X_{vpn}
+
Y
UPT
virtual
page
physical
page


user
virtual space
system
virtual space
physical
space
SPT
base (p)
SPT
SPT
access
Y_{vpn}
+
Y_{ppn}
UPT
base (s)
X
virtual
page
X_{vpn}
+
Y
UPT
Y_{byte}
||
UPT
access
virtual
page
X_{ppn}
X_{byte}
||
Data
access

Memory protection with VM

- A process can access only those physical pages which are pointed to by its page table
- Page tables are initialized by the OS and are updated automatically. User processes can’t modify these.
- Programs run in user mode or privileged mode. Certain operations are possible in privileged mode only.

Virtual memory hit time

- simple page table :

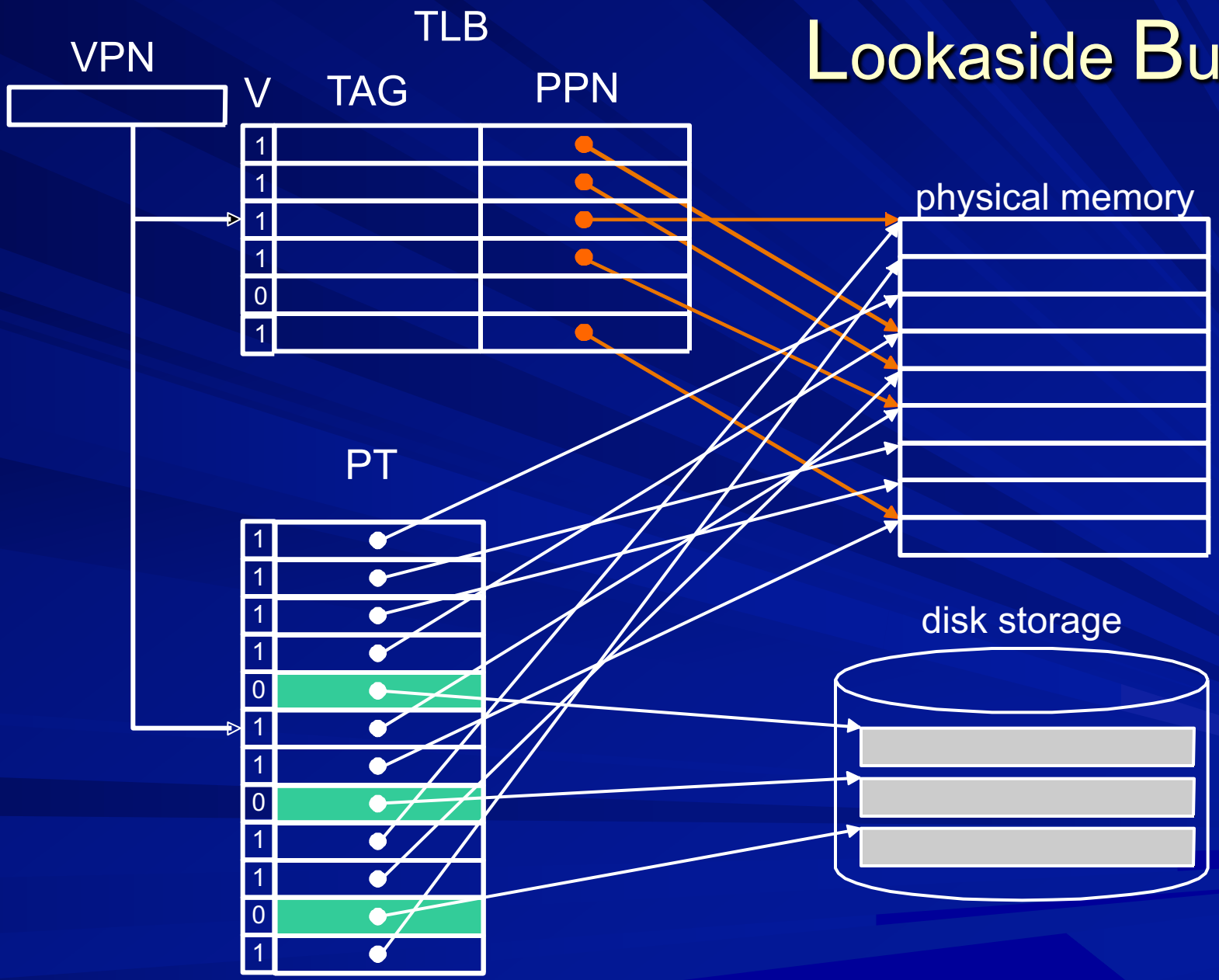
  2 x physical memory access time
- 2 level page table :

  3 x physical memory access time
- paged page table :

  3 x physical memory access time

Can this be better??

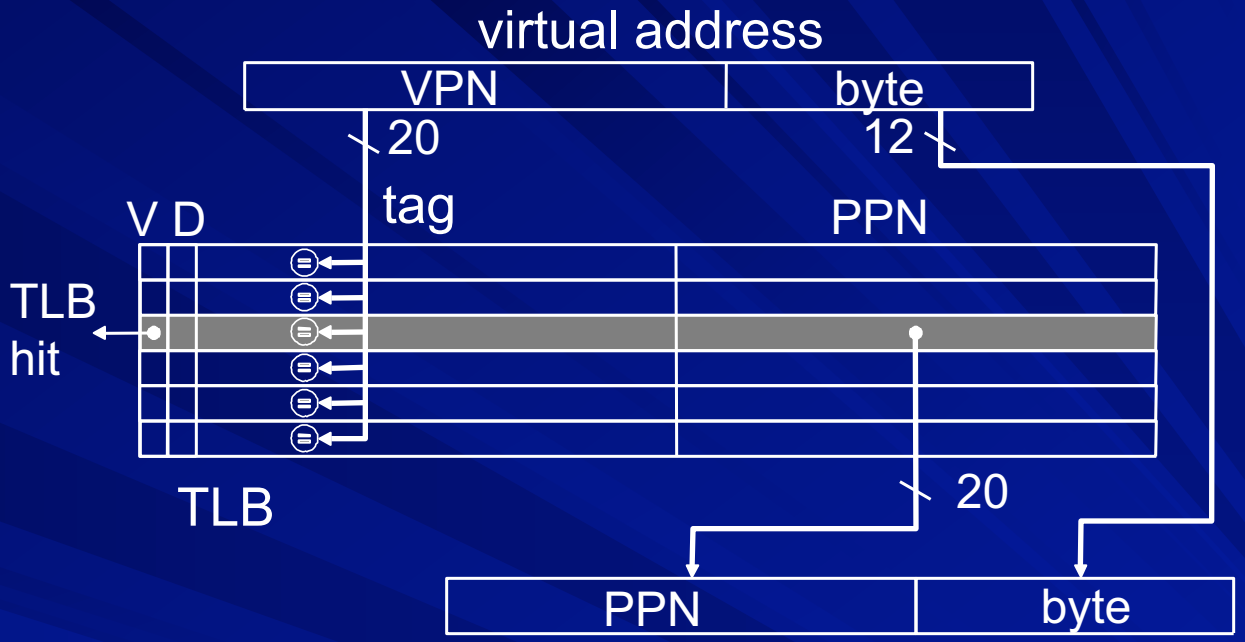
Translation Lookaside Buffer



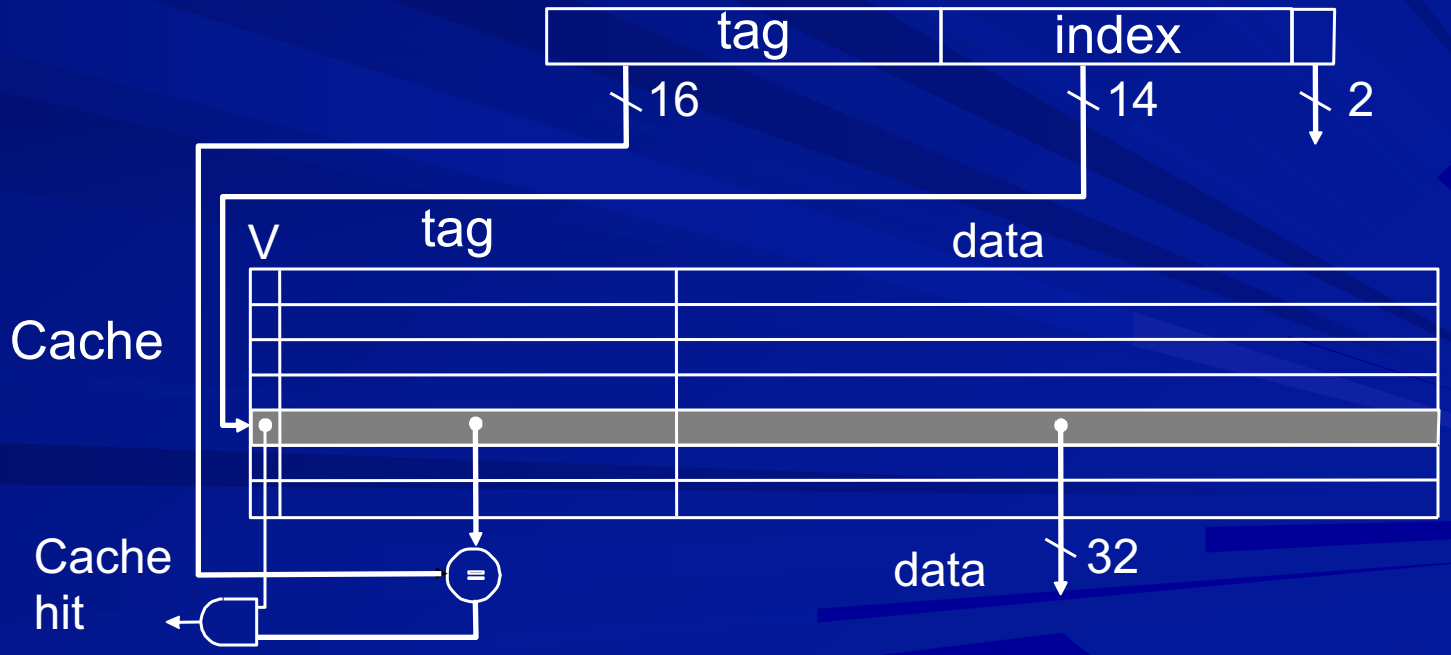
How is TLB Miss handled?

- Hardware
  - Memory Management Unit accesses Page Table
  - Extra hardware, but efficient
- Software
  - Exception is caused
  - OS (exception handler) accesses Page Table
  - Page Table structure defined by OS, not fixed by hardware

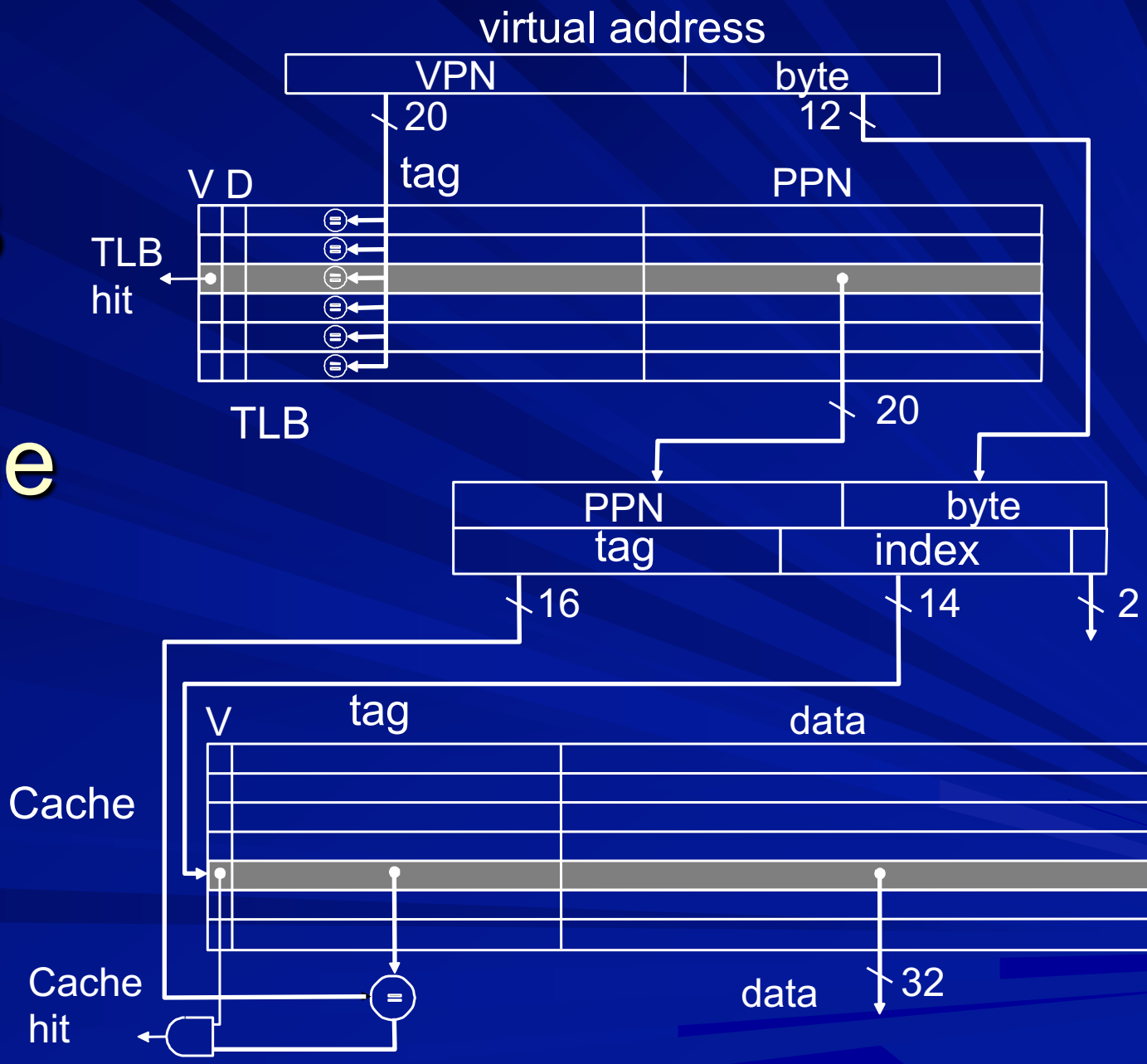
TLB with Cache



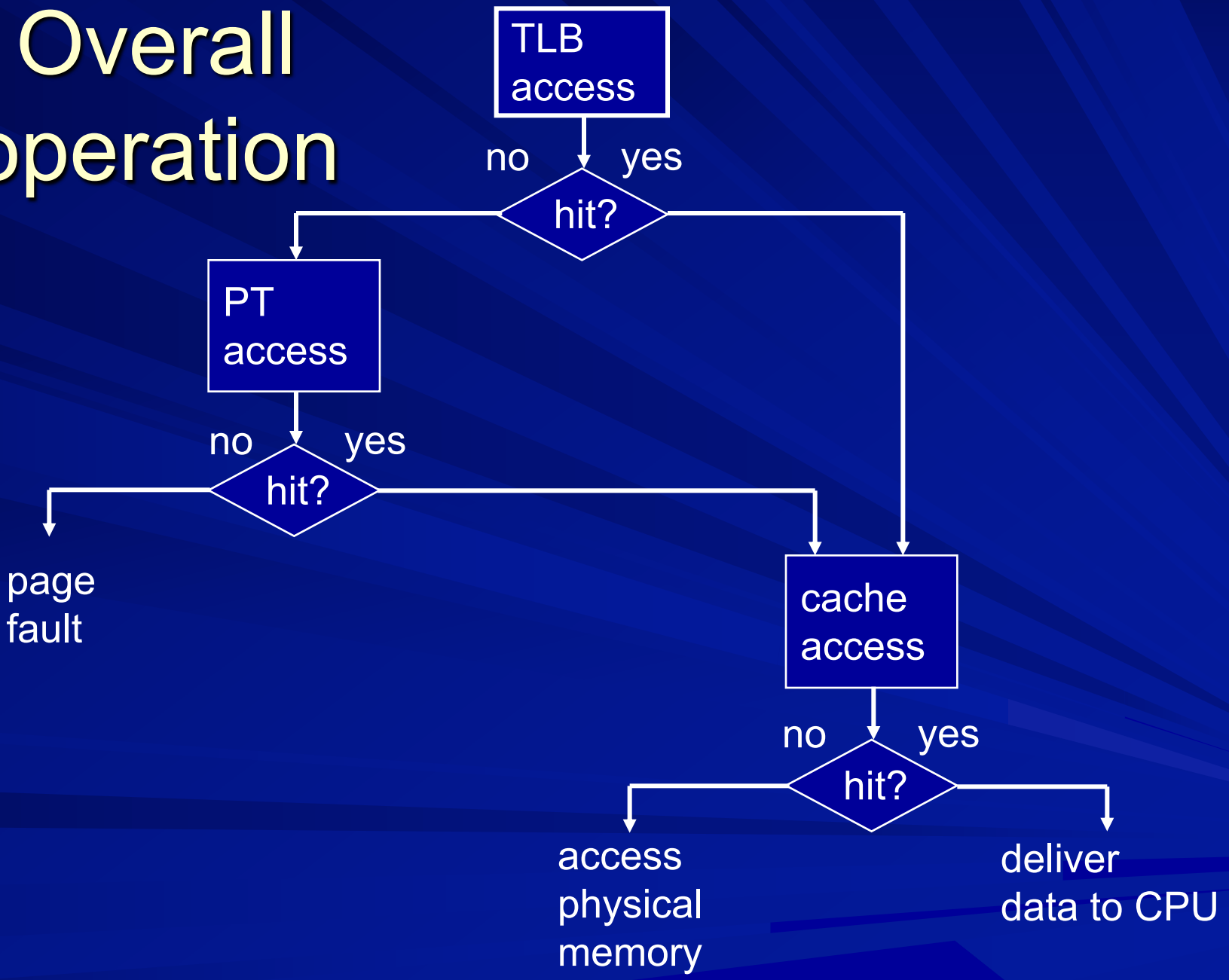
TLB with Cache



TLB with Cache



Overall operation



Virtually addressed cache

- Tags in the cache are from virtual addresses
- Cache access is made first
- TLB accessed only after cache miss
- Problem of aliasing - two copies of a shared object in physical memory
- Physically tagged and virtually indexed cache may be used

Indexing before address translation

- Virtually indexed, physically tagged cache
  - simple and effective approach
  - indexing overlaps with address translation, tag matching after address translation
  - valid only if index field does not change during address translation (possible for caches that are not too large)