23. Virtual Memory: Hierarchical Page Table

EECS 370 – Introduction to Computer Organization – Fall 2020

Satish Narayanasamy

EECS Department
University of Michigan in Ann Arbor, USA

Check your computer

System Information

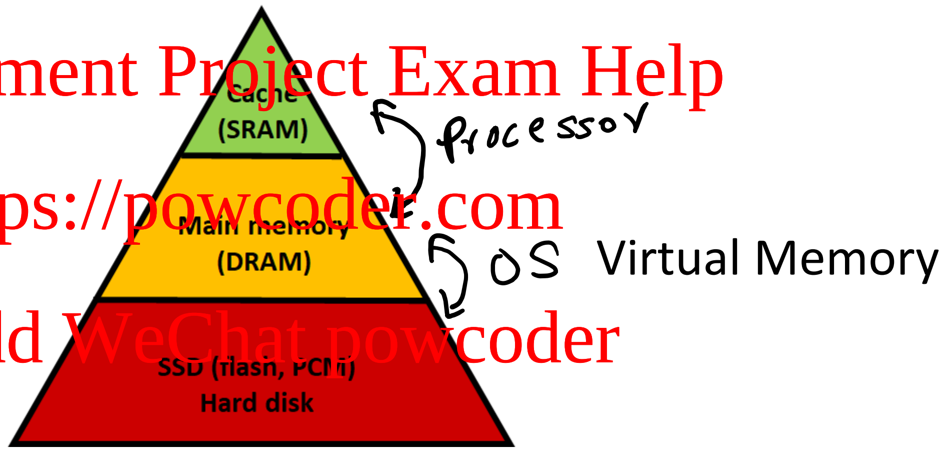
File Edit View Help

- System Summary
  - Hardware Resources
    - Conflicts/Sharing
    - DMA
    - Forced Hardware
    - I/O
    - IRQs
    - Memory
  - Components
    - Multimedia
    - CD-ROM
    - Sound Device
    - Display
    - Infrared
    - Input
    - Modem
    - Network
    - Ports
    - Storage
      - Drives
      - Disks
      - SCSI
      - IDE
    - Printing
    - Problem Devices
    - USB
  - Software Environment

Item	Value
OS Name	Microsoft Windows 10 Pro
Version	10.0.18363 Build 18363
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	DESKTOP[redacted]
System Manufacturer	Microsoft Corporation
System Model	Surface Studio 2
System Type	x64-based PC
System SKU	Surface_Studio_2_1707_Commercial
Processor	Intel(R) Core(TM) i7-7820HQ CPU @ ...
BIOS Version/Date	Microsoft Corporation 532.3238.768, ..
SMBIOS Version	3.2
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	Microsoft Corporation
BaseBoard Product	Surface Studio 2
BaseBoard Version	Not Available
Platform Role	Desktop
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	United States
Hardware Abstraction Layer	Version = "10.0.18362.1171"
User Name	[redacted]
Time Zone	Eastern Standard Time
Installed Physical Memory (RAM)	16.0 GB
Total Physical Memory	16.0 GB
Available Physical Memory	6.73 GB
Total Virtual Memory	29.5 GB
Available Virtual Memory	5.74 GB
Page File Space	13.5 GB
Page File	C:\pagefile.sys
Kernel DMA Protection	Off
Virtualization-based security	Not enabled
Device Encryption Support	Elevation Required to View
Hyper-V - VM Monitor Mode Extensi...	Yes
Hyper-V - Second Level Address Tran...	Yes
Hyper-V - Virtualization Enabled in F...	Yes
Hyper-V - Data Execution Protection	Yes

?

Virtual Memory Role



Virtual Memory Roles

Capacity: Main memory is not enough

Problem:

Modern systems can afford ~128 GBs DRAM space = 2^37 bytes. Programs written in 64-bit ISA need 2^64 bytes!

Need to run many programs simultaneously on the same machine. Each program may require GBs of memory.

Solution:

Provide an illusion of storage large enough for 2^64 bytes of data for all concurrently running programs

Manage main memory like an exclusive fully-associative cache. Spills to disk.

Security features

Isolation

Unrelated programs must not have access to each other's data

Permissions

Programs may want to share data and code (e.g., library)

Programs may want to disable read/write permissions to some portions of memory

e.g., mark instructions are read-only, no read/write permission for unallocated heap

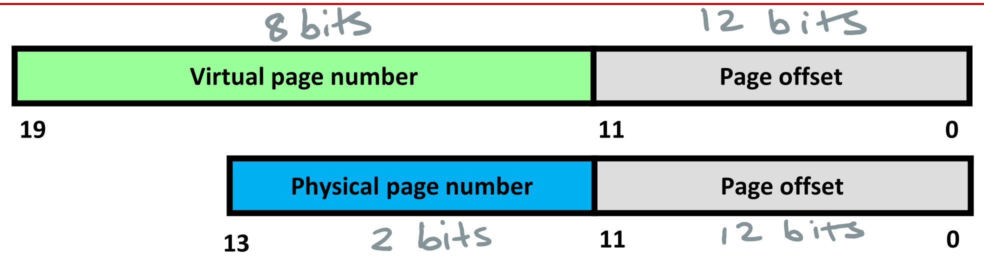
Virtual Memory: An Example

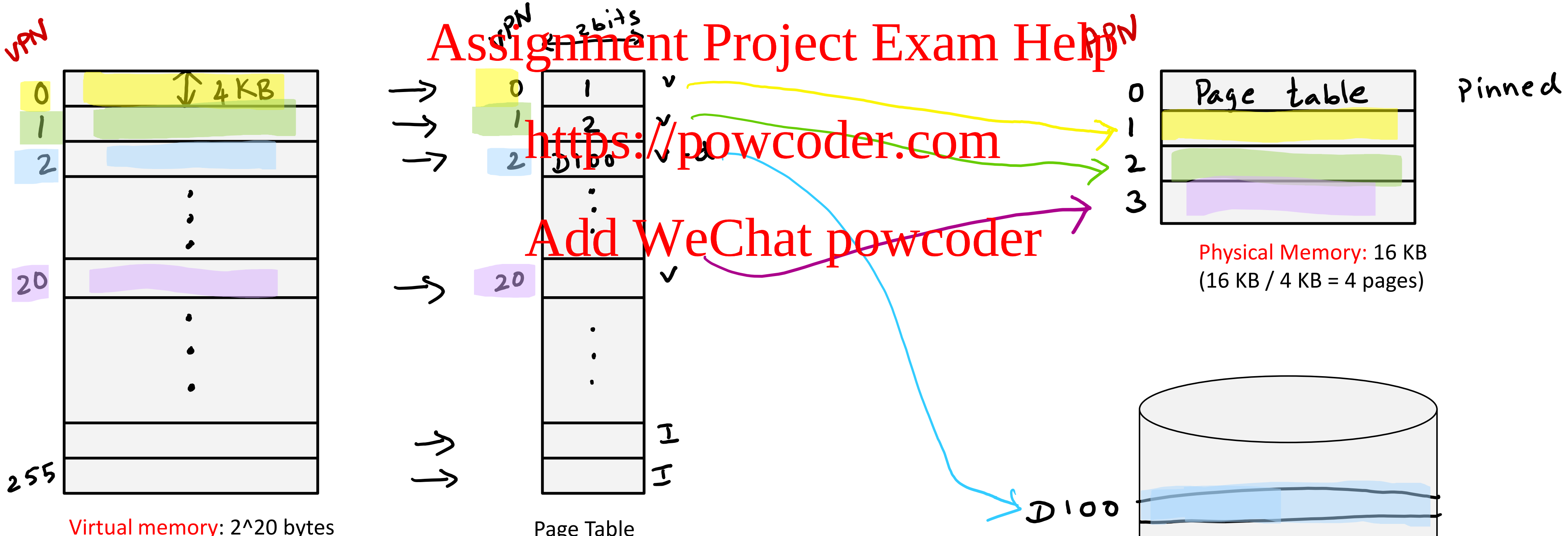
Page offset size = log(4 KB) = 12 bits

Page size = 4 KB

Virtual memory
(2^{20} bytes = 256 pages)

Physical Memory
(16 KB = 4 pages)





Virtual memory: 2^{20} bytes

2^{20} / 4 KB = 256 pages

Page Table
(256 entries)

Physical Memory: 16 KB
(16 KB / 4 KB = 4 pages)

Disk
(swap partition)

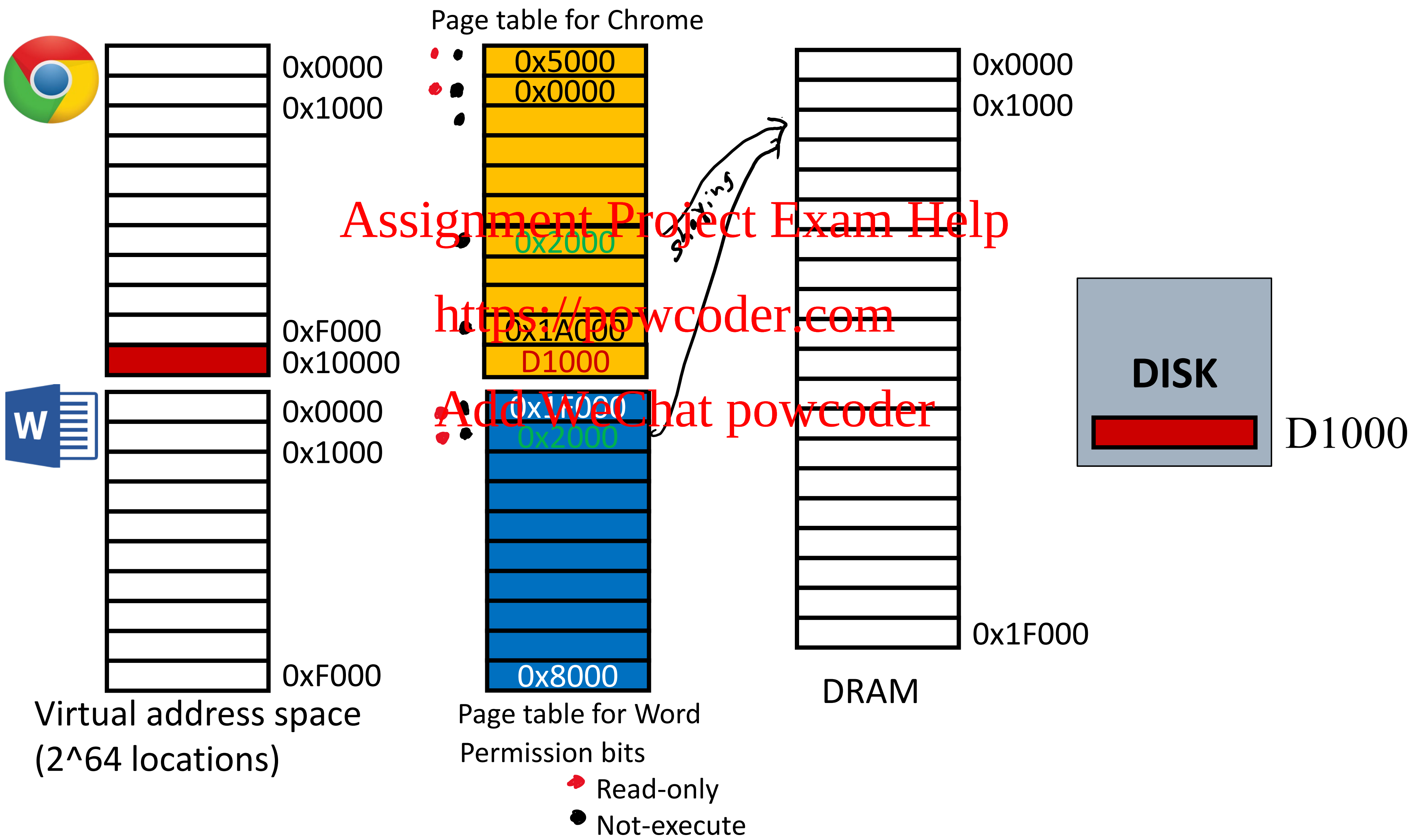
Page Replacement: Approximating LRU

Page table indirection enables a **fully associative** mapping between virtual and physical pages.

How does OS implement LRU?

- Precise LRU is expensive
- LRU is a heuristic anyway, so approximate LRU
- Keep a "***accessed" bit per page***, cleared occasionally by the OS
- OS picks any "unaccessed" page (accessed bit not set) to evict

Virtual Memory: Security: Isolation, Sharing, Permissions



Page Table Entry Contents

Physical page number (PPN)

Allocated or not? (valid/invalid)

Main memory or disk?

Access permission bits

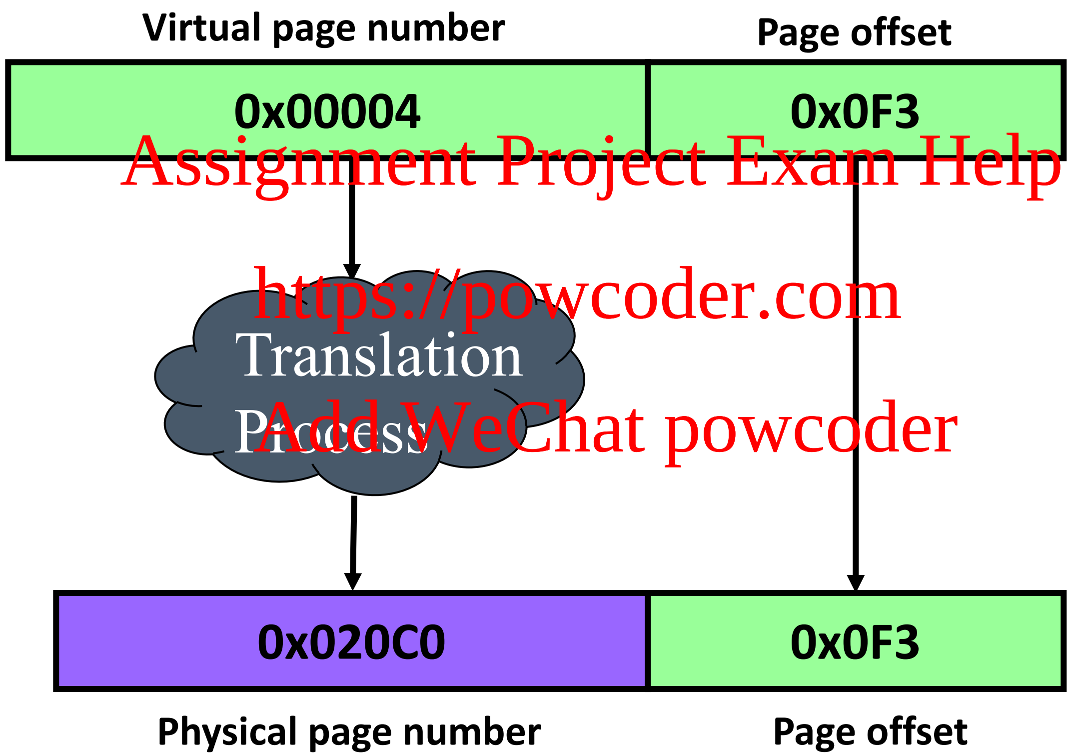
- read-only
- not-execute

Dirty page or not?

LRU meta-data

Address Translation

Virtual address = 0x000040F3



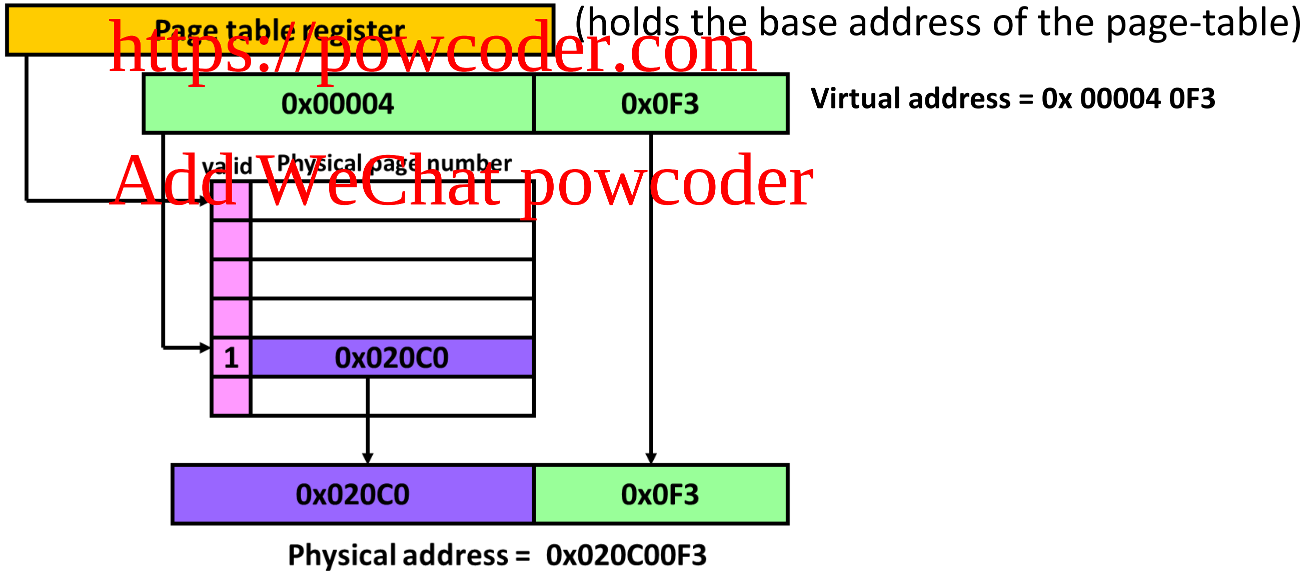
Physical address = 0x020C00F3

Page table for address translation

Single-level page table: an array-like structure.

a big array indexed by the virtual page number

Each page-table entry stores the physical page number (and some status bits like "valid").



Single-level page table

Next Problem: Page table is too large

Solution: Multi-level Page Table

Problem: Single-level page table is too big

Example:

Assume 64-bit ISA. 4 KB pages.

# virtual pages	= 2 ^ 64 / 4 KB	= 2 ^ 52 virtual pages
# page-table entries	= # virtual pages	= 2 ^ 52 entries

Say, each page table entry is 4 bytes

Total page size = 2 ^ 52 entries * 4 bytes per entry = 2 ^ 54 bytes!

= ~160,000 DRAMs each of 100 GB size (that is probably more DRAM than there is in UM!)

Observation

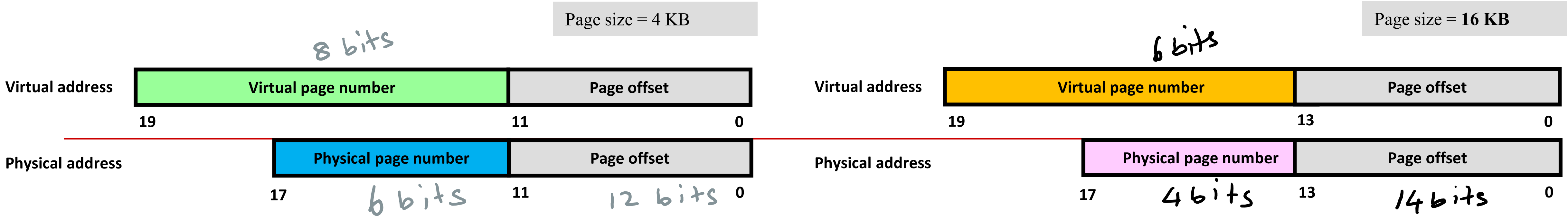
Problem: OS is allocating all page-table entries for *a process when it starts*.

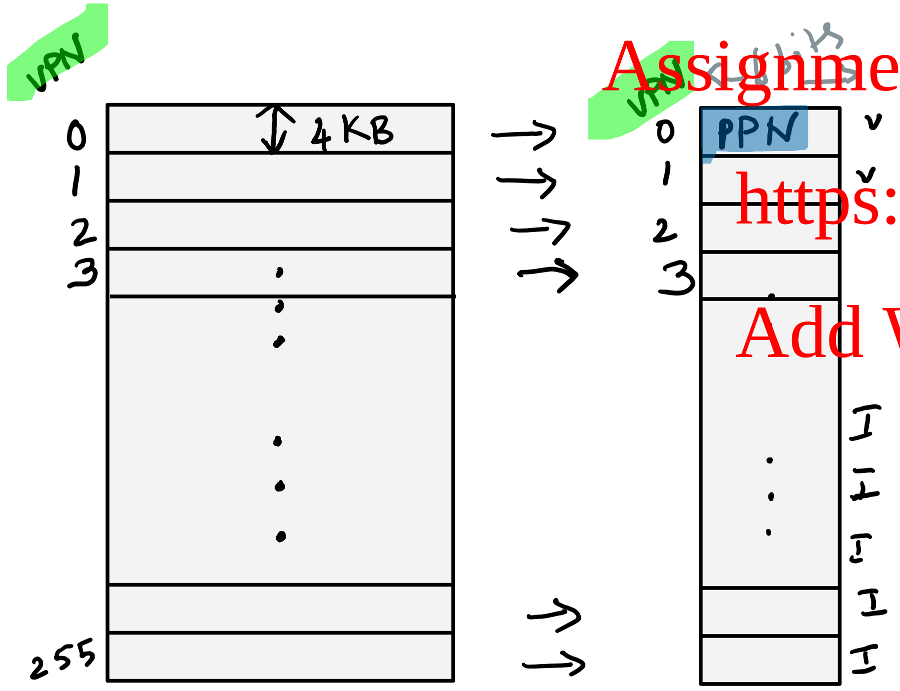
Most processes only use a very (very) small fraction of available virtual memory at any instant.

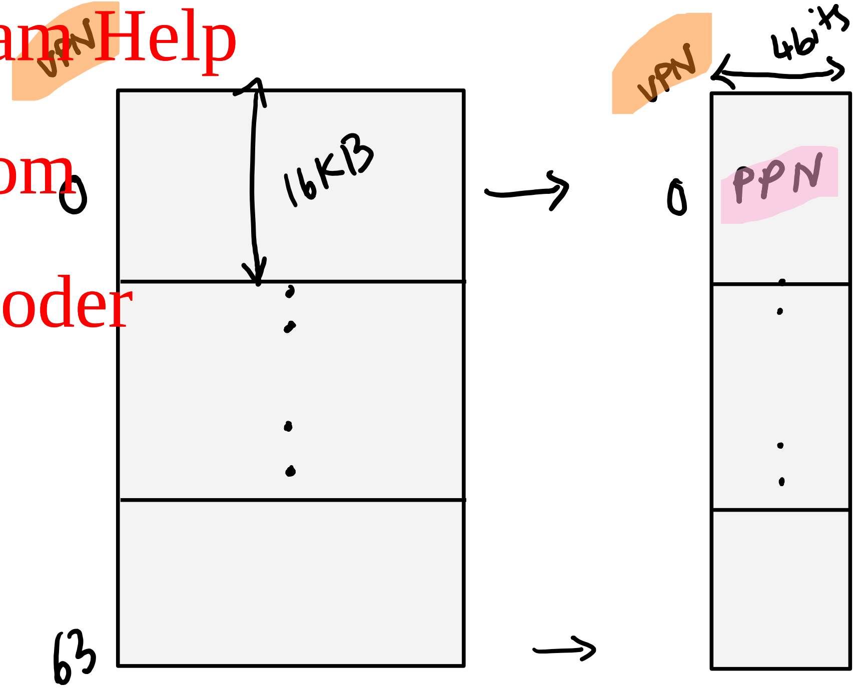
OS allocates physical pages on-demand only when a virtual page is accessed by the program.

Idea: Similarly, can we allocate page-table entries also on demand?

i.e., only allocate space for a page-table entry only its corresponding virtual address is accessed.







Virtual memory: 2^20 bytes
2^20 / 4 KB = 256 pages

Page Table
(256 entries)

Size = 256 * 6 bits = 1536 bits
(just for PPN field)

Virtual memory: 2^20 bytes
2^20 / 16 KB = 64 pages

Page Table
(64 entries)

Size = 64 * 4 bits = 256 bits

Hierarchical Page Table: Goal

Can we get the best of both worlds?

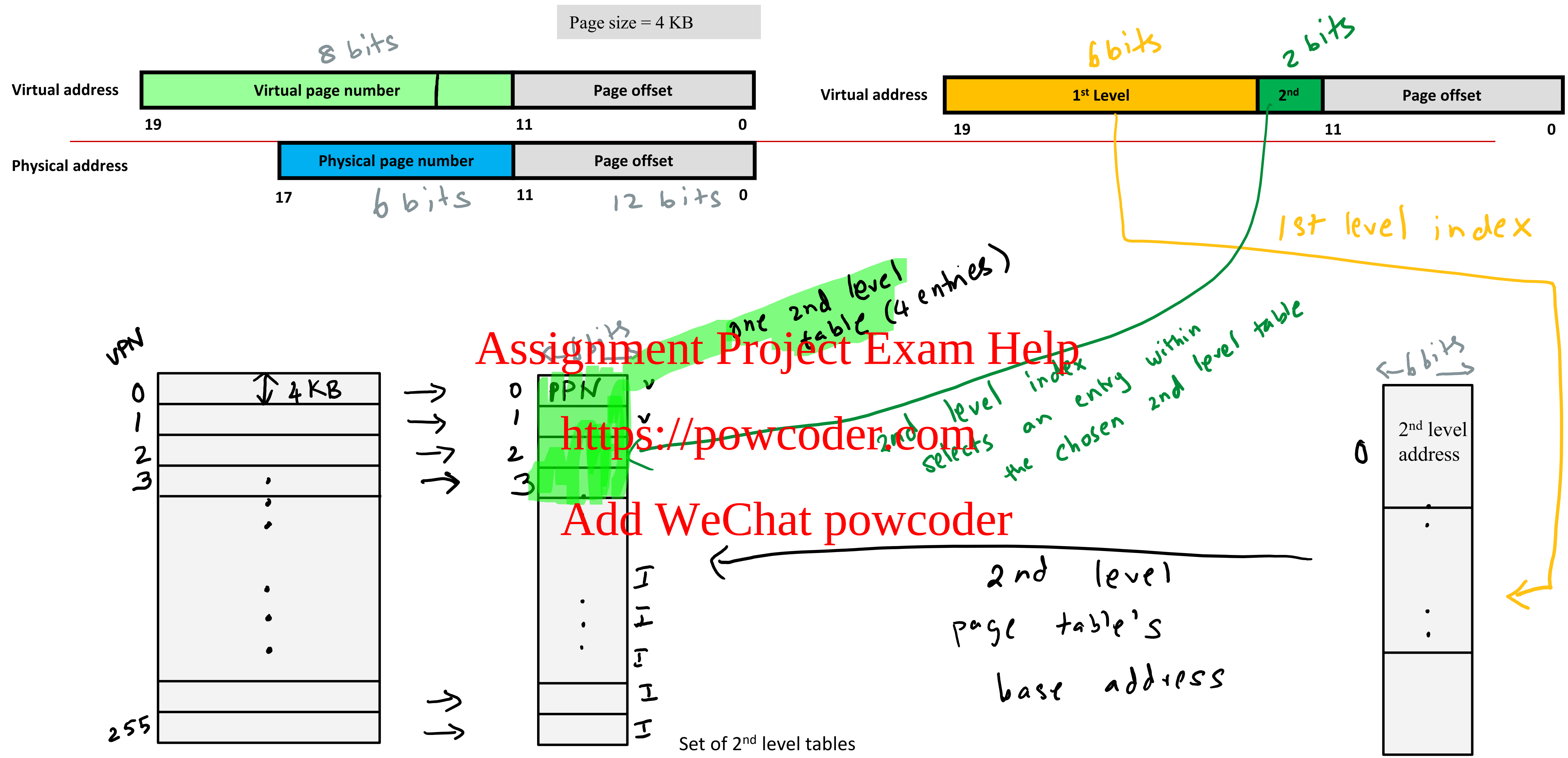
- Smaller pages (4 KB)

- Smaller page tables (as we would need for super-pages -- 16 KB)

Idea:

Allocate super-page-table at the start

Allocate smaller page table for translating each smaller page within a super-page on-demand

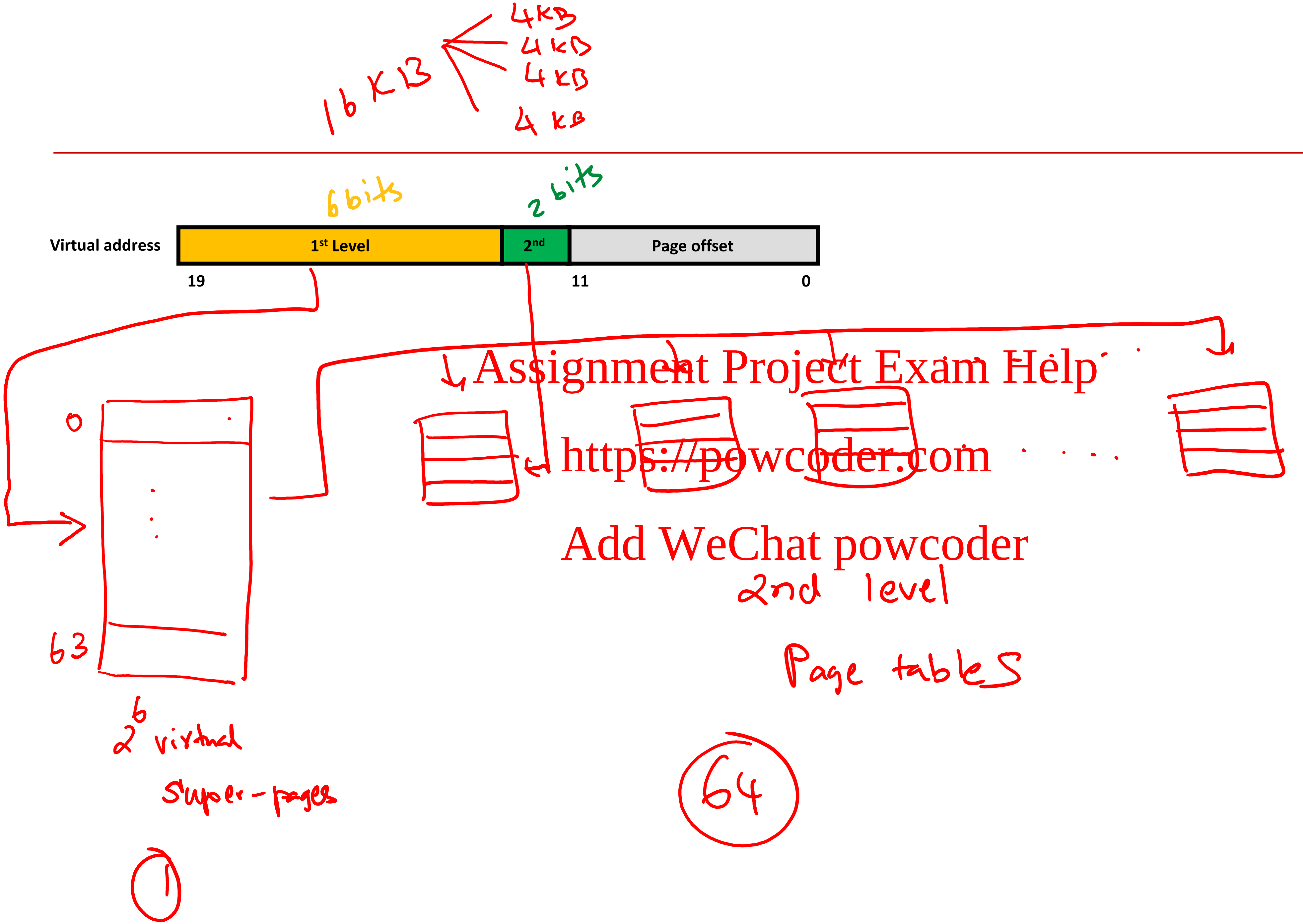


Virtual memory: 2^20 bytes
2^20 / 4 KB = 256 pages

Set of 2nd level tables
(256 entries in total.
But, each table has 4 entries,
and is allocated on demand)
Size = 256 * 6 bits
(in the worst case)

1st level Page Table
(64 entries)

Size = 64 * **6** bits
(allocated at start)
6 bits, because PPN size is 6 bits


16 KB
4KB
4KB
4KB
4KB
6 bits
2 bits
Virtual address
1st Level
2nd
Page offset
19
11
0
0
63
2^6 virtual
super-pages
1
2nd level
Page tables
64

Hierarchical 2-level page table

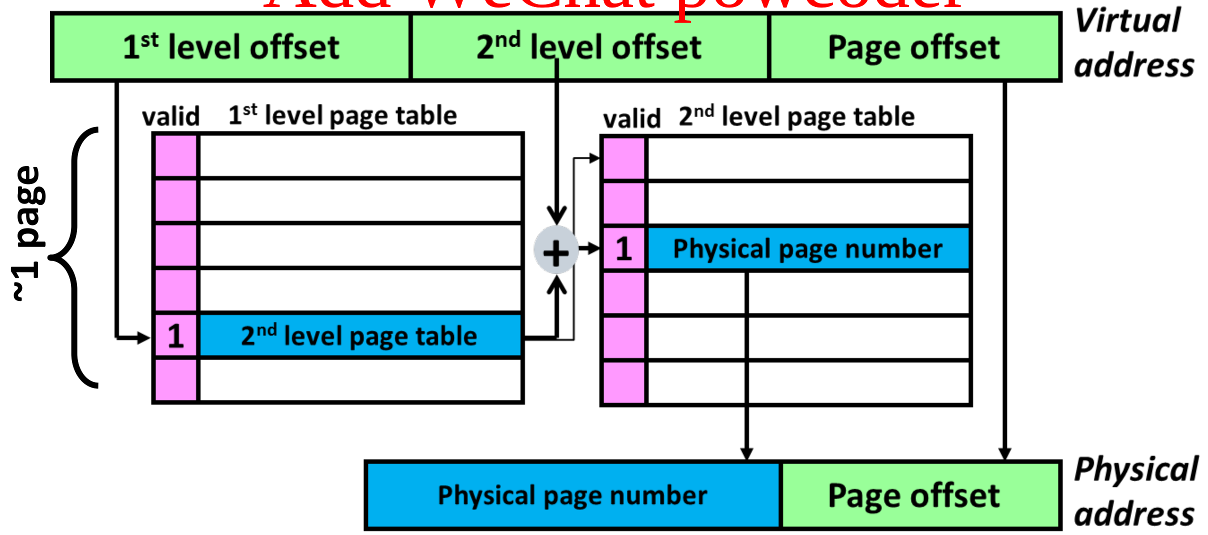
A tree-like hierarchical structure.

- A 1st level page table entry (*root of tree*) contains location of a 2nd level page table (leaf)
- A 2nd level page table entry is same as an entry in the single-level page-table

Allocate 1st level page table when the process starts.

Allocate a 2nd level page table on-demand only when the process accesses corresponding virtual address

Can be generalized to n-level (multi-level) page-table



Hierarchical 2-level page table: Space benefits

2^{nd} level page table size is proportional to the amount of virtual memory used by the process

Common case: size of multi-level page table << single-level

very few 2^{nd} level page table would be allocated

Worst case: size of multi-level page table = single-level page table + 1^{st} level page-table

(single-level page table size, because almost all 2^{nd} level page tables are allocated)

1^{st} level page-table is an additional overhead

Flat vs Hierarchical

Flat (single-level)

Pros:

One page table lookup (one memory access) *per address translation*

Cons:

All page-table entries allocated at the start. Always takes up a lot of memory

Hierarchical (multi-level) -- Used in most modern systems

Pros:

Allocates page-table entries on-demand. So, typically uses much less memory than single-level

Cons:

More page-table lookups (memory accesses) per address translation:

N page table lookups for N-level page table

Hierarchical page table: Example: 32bit Intel x86

1st level index	2nd level index	Page offset	Virtual address
31 ... 22	22 ... 12	12 ... 0	

Assume:

Size of 1^{st} level index	10 bits	
Size of 2^{nd} level index	10 bits	
Page offset size	12 bits	(not important for this problem)
Size of one page table entry	4 bytes	

Derivation:

# entries in the 1^{st} level page table	2^10
# entries in the 2^{nd} level page table	2^10

Size of 1^{st} level page table = 2^10 * 4 bytes = 4 KB

Size of one 2^{nd} level page table = 2^10 * 4 bytes = 4 KB

Computing Space for multi-level page table

N 2nd level page tables have been allocated. This means, 1st level page table will have N valid entries.

Total size of the page table = 1st level page-table size + N * (size of one 2nd level page table)

= 4 KB + N * 4 KB

(for the example in the previous slide)

Class Problem 1 (32 bit x86)

What is the least amount of memory that could be used? When would this happen?

What is the most memory that could be used? When would this happen?

How much memory is used for this memory access pattern:

0x00000ABC
0x00000ABD
0x10000ABC
0x20000ABC

How much memory if we used a single-level page table with 4KB pages? Assume entries are rounded to the nearest word (4B)

Class Problem 1 (32 bit x86)

1. What is the least amount of memory that could be used? When would this happen?

when N = 0 (true when no memory has been accessed -- before program runs)

4 KB + N * 4 KB

= 4KB + 0 * 4 KB

= 4 KB

2. What is the most memory that could be used? When would this happen?

All 2^{nd} level page tables are allocated. That is, all entries in 1^{st} level page table are valid.

So, N = 2 ^ 10

(true when program uses all virtual pages (2^{20} pages))

4 KB + N * 4 KB

= 4KB + 2 ^ 10 * 4 KB

= 4 KB + 4 MB

= 4100KB

Class Problem 1 (32 bit x86)

3. How much memory is used for this memory access pattern:

0x00000ABC // Page fault
0x00000ABD
0x10000ABC // Page fault
0x20000ABC // Page fault

N = 3

4 KB + N * 4 KB

= 4KB + 3*4KB

= 16 KB

Class Problem 1 (32 bit x86)

4. What is the size of a **single-level page table**? Assume entries are rounded to the nearest word (4B)

Virtual pages = Total virtual memory size / page size

= 2 ^ 32 / 2 ^ 12

= 2 ^ 20 pages

Each virtual page has an entry in the single-level page table

Single-level page table size = # entries * size of each entry

= 2^20 * 4 bytes = 4 MB

~= Size of all 2-level page tables in the worst case

Class Problem 1: Summary

	2-level page table size	Single-level page table size
Best case	4 KB	4 MB
Worst case	4 KB + 4 MB	4 MB
For given access pattern (slide 25)	4 KB + 12 KB	4 MB

2-level page table

Only the first level (super) page table is allocated at the start.

2nd level page tables are allocated on-demand, whenever a new super-page is accessed by the program.

Single-level page table

The entire page table is allocated at the start.

Class Problem 2 – Hierarchical 2-level VM

Design a two-level page-table for a 24-bit byte addressable ISA.

Physical memory size = 256KB

Page size = 512 Bytes.

Size of 1^{st} level page table entry = 3 bytes (a physical memory address pointer to a 2L page table)

Size of one 2^{nd} level page table = 1 page

Size of a 2^{nd} level page-table entry

2^{nd} level page table entry contains physical page number + 1 valid bit

Size of 2^{nd} level page table entry *must be smallest possible integer number of bytes*

Compute:

- Number of entries in each 2^{nd} level page table
- 2^{nd} level page table index size
- 1^{st} level page table index size
- Size of the 1^{st} level page table

Class Problem 2 – Hierarchical 2-level VM

PPN | Page offset

9	9

Physical address

Design a two-level page-table for a 24-bit byte addressable ISA.

Physical memory size = 256KB

$2^8 \times 2^{10} / 2^9 = 2^9$ phy. pages

Page size = 512 Bytes.

$\log(512) = 9$

Size of 1^{st} level page table entry = 3 bytes (a physical memory address pointer to a 2L page table)

Size of one 2^{nd} level page table = 1 page

$\frac{512 \text{ bytes}}{\text{size of 2nd level entry}} = \frac{512}{2} = 256$ entries

Size of a 2^{nd} level page-table entry ? 9 bits

2^{nd} level page table entry contains physical page number + 1 valid bit

Size of 2^{nd} level page table entry *must be smallest possible integer number of bytes*

9 bits $\simeq$ 2 bytes

Compute:

Number of entries in each 2^{nd} level page table

$\log(256) = 8$ bits

2^{nd} level page table index size

1^{st} level page table index size

Size of the 1^{st} level page table $= 2^7 * 3$ bytes

1st | 2nd | Page offset

7	8 bits	9

virtual address

24 bits (ISA)

$24 - 8 - 9 = 7$ bits

Class Problem 2 – Hierarchical 2-level VM

Physical page number = 9b	Page offset = 9b

Page Offset size = log (512) = 9 bits
Physical address size = log (256 K) = 18 bits
Physical page number (PPN) size = 18 bits (physical address size) – 9b (page offset)
= 9 bits

2^{nd} level page table entry size = 9b (PPN size) + 1 bit = ~2 bytes
Size of one 2^{nd} level page table = 1 page (given) = 512 bytes

#entries in 2^{nd} level page table = 512 bytes / 2 bytes = 256
2^{nd} level index size = log (256) = 8 bits

1^{st} level index size = 24 (virtual address size) – 8 (2^{nd} level index size) – 9 (page offset size)
= 7 bits

1^{st} level page table entry size = 3 bytes (given)
1^{st} level page table size = 2^7 (# entries) * 3 bytes = 384 bytes

1^{st} level = 7b	2^{nd} level = 8b	Page offset = 9b

Virtual address = 24b

Class Problem 3:
Simulate for hierarchical 2-level page table in problem 2

Virtual Address	1st level	2nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C						
0x001F0C						
0x020F0C						

1st level = 7b	2nd level = 8b	Page offset = 9b

Virtual address = 24b

Physical page number = 9b	Page offset = 9b

Physical address = 18b

On a page fault, allocate physical page number starting from 0.
Assume all physical pages are available initially.

Class Problem 3:

0x000F0C = 0000 0000 0000 1111 0000 1100

Simulate for hierarchical 2-level page table in problem 2

Virtual Address	1st level	2nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C						
0x020F0C						

1st level = 7b	2nd level = 8b	Page offset = 9b

Virtual address = 24b

Physical page number = 9b	Page offset = 9b

Physical address = 18b

On a page fault, allocate physical page number starting from 0.
Assume all physical pages are available initially.

Class Problem 3:
Simulate for hierarchical 2-level page table in problem 2

Virtual Address	1st level	2nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C	0x00	0x0F	0x10C	Y	0x001	0x0030C
0x020F0C						

1st level = 7b	2nd level = 8b	Page offset = 9b	Virtual address = 24b

Physical page number = 9b	Page offset = 9b	Physical address = 18b

On a page fault, allocate physical page number starting from 0.
Assume all physical pages are available initially.

Class Problem 3:
Simulate for hierarchical 2-level page table in problem 2

Virtual Address	1st level	2nd level	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C	0x00	0x0F	0x10C	Y	0x001	0x0030C
0x020F0C	0x01	0x07	0x10C	Y	0x002	0x0050C

1st level = 7b	2nd level = 8b	Page offset = 9b

Virtual address = 24b

Physical page number = 9b	Page offset = 9b

Physical address = 18b

On a page fault, allocate physical page number starting from 0.
Assume all physical pages are available initially.