Virtual memory

What causes cache misses? 3 Cs

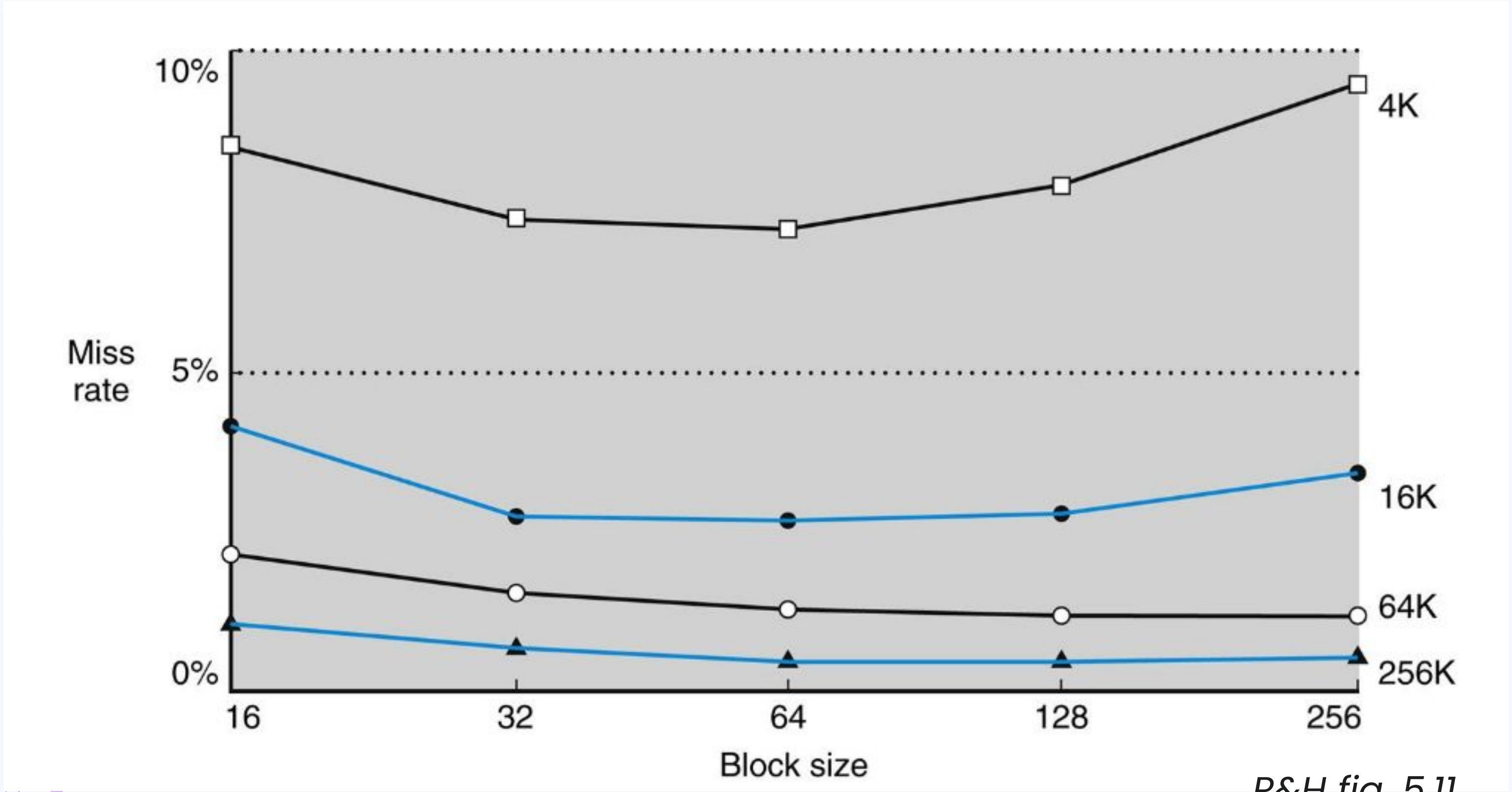
Compulsory – bringing the first blocks into a cache (“warming up” the cache)

Capacity – cache not big enough to contain all of the blocks it needs

Conflict – blocks constantly evicted due to cache collisions

Can we decrease compulsory misses?

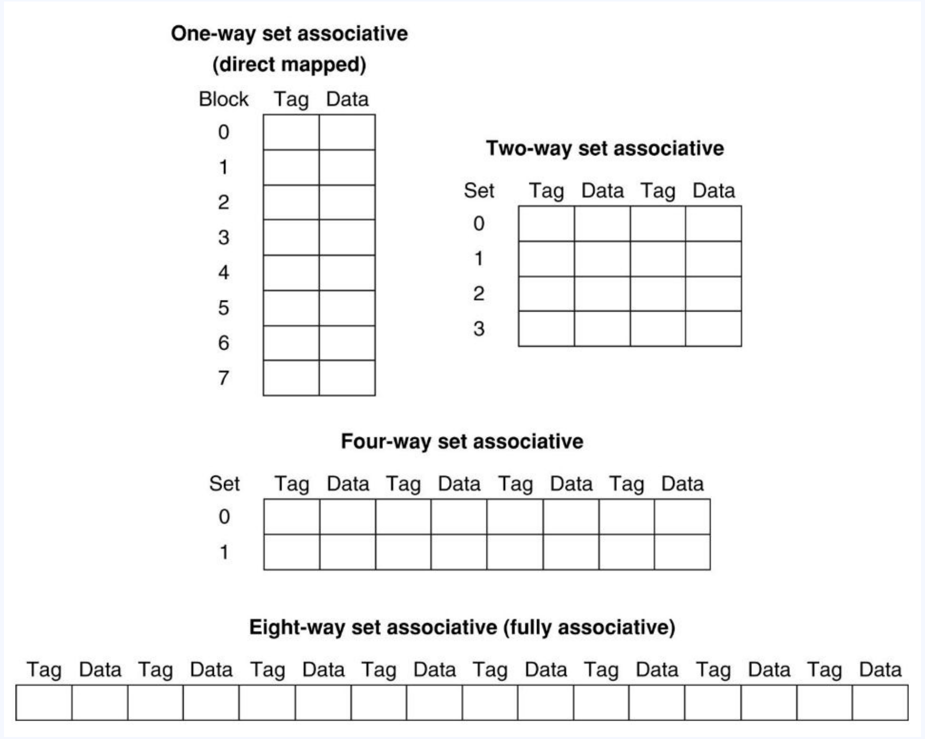
Increasing block size has limited effects



P&H fig. 5.11

Fully associative cache

Set-associative caches



P&H fig. 5.15

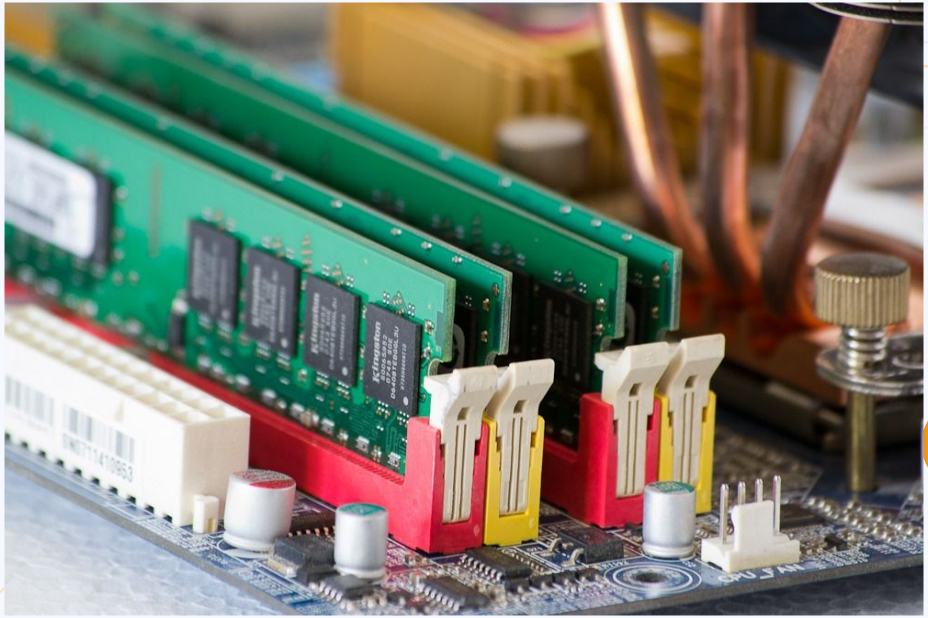
Physical memory

The memory physically available on a computer

Models how we've thought about memory so far

Allows random access to bytes using addresses

image source



? ? ?

What limitations does physical memory put on our systems/software?

Virtual memory

A way of using main memory as a “cache” for disk storage

Exists in modern OS (managed by SW/HW together)

Advantages:

- Allows coordination of memory between processes (less complexity from process POV)
- Allows process to see larger address space than fits in main memory

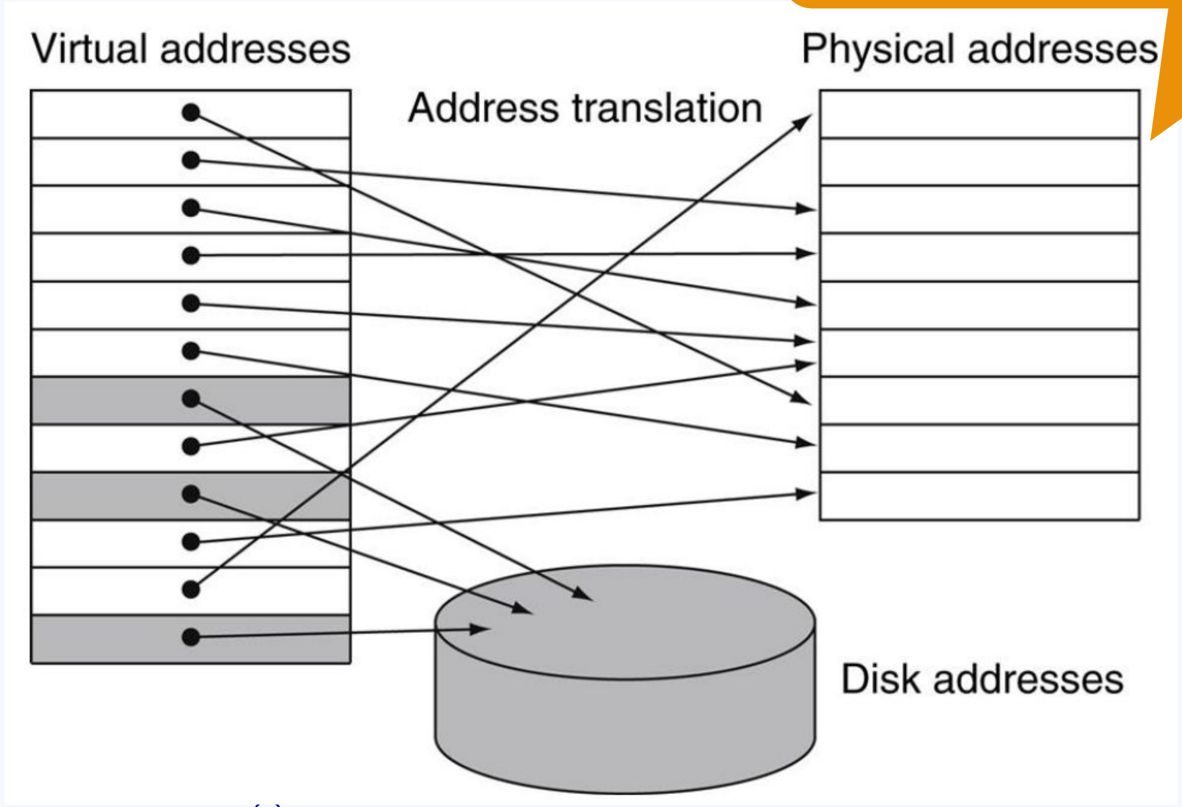
Complexities:

- Managing the mapping of physical to virtual memory
- Slow speed of disk

P&H Fig. 5.25

Each block of virtual memory is called a “page” (4KB on Intel chips, 16KB on Apple silicon)

Each process can have its own view of virtual addresses (0-N)



? ? ?

What determines how many physical pages our system has? How many virtual pages?

Avoiding page faults

A virtual memory “miss” is called a **page fault**

Since disk is even slower (100k* slowdown) than main memory, we want:

- Fairly large page sizes
- Fully associative placement of pages in memory (virtual page can map to any physical page)

Does it make sense for virtual memory systems to use write-through or write-back?

? ? ?

If virtual addresses can map to *any* physical address, how do we efficiently find a physical page?

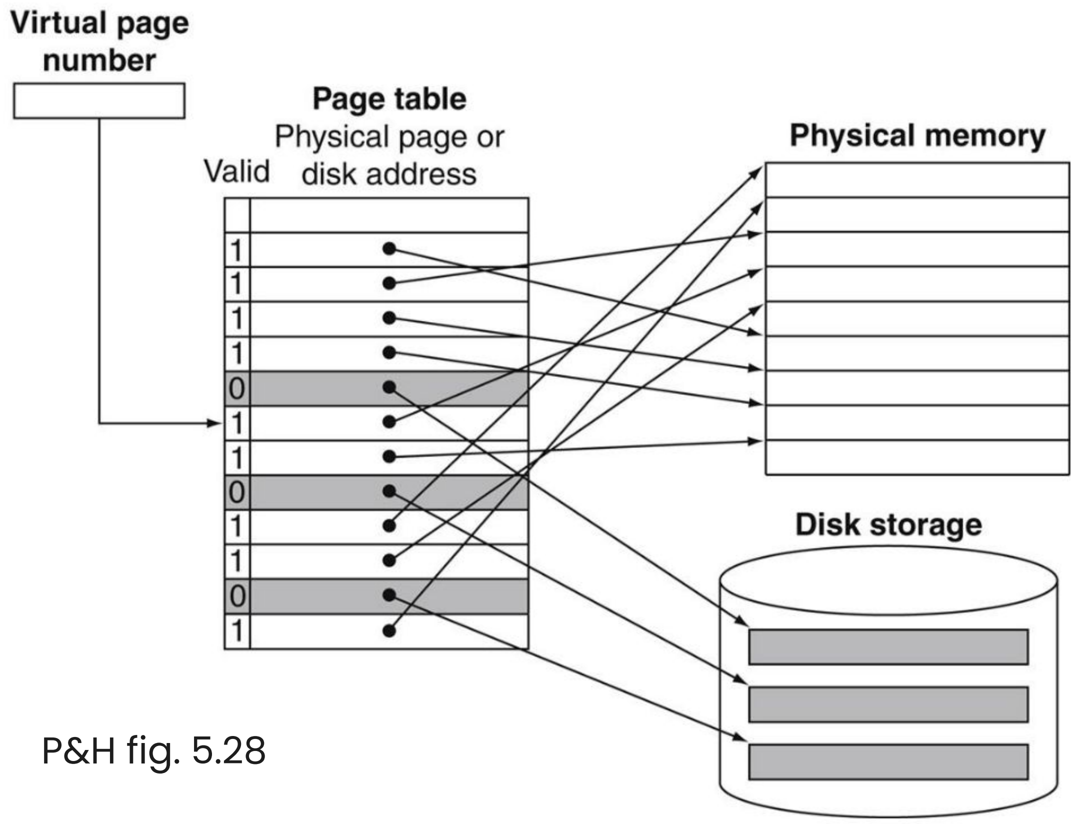
Page tables

Live in main memory (separate from pages themselves)

One for each process

Map virtual addresses to physical addresses (**not** a cache – **why?**)

Page faults managed by OS **(why?)**



P&H fig. 5.28

? ? ?

How does the hardware use/manage the page table? How does the OS?

? ? ?

What should happen if our virtual address space is so big that the page table can’t efficiently fit in main memory?

Multi-level page table

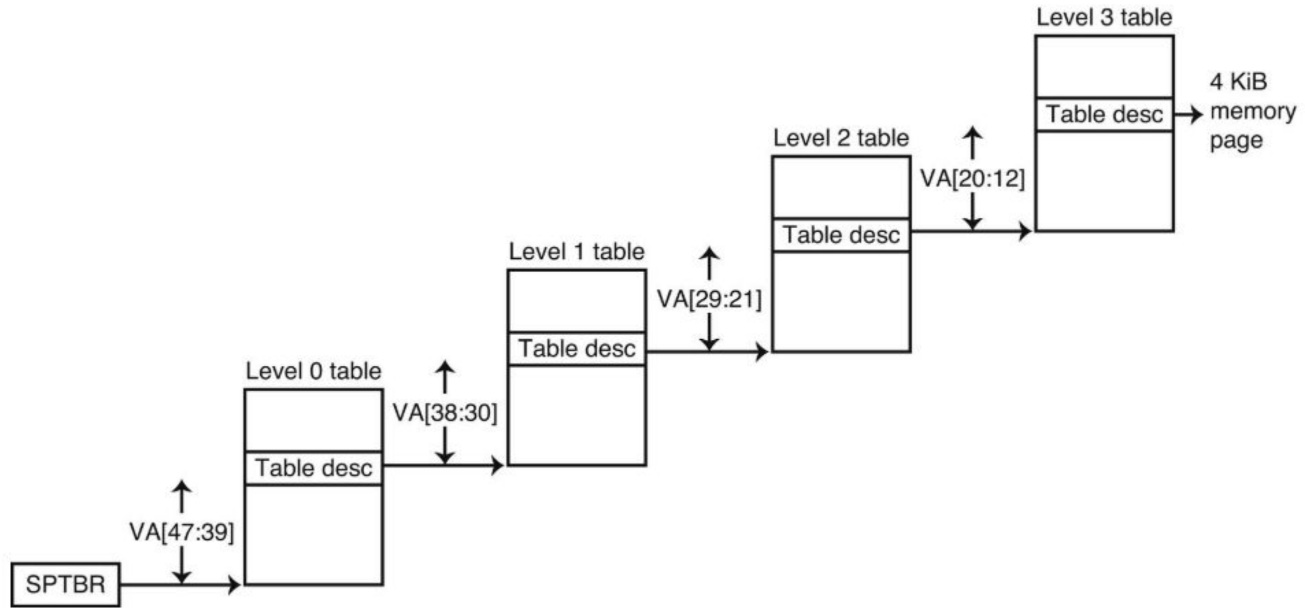
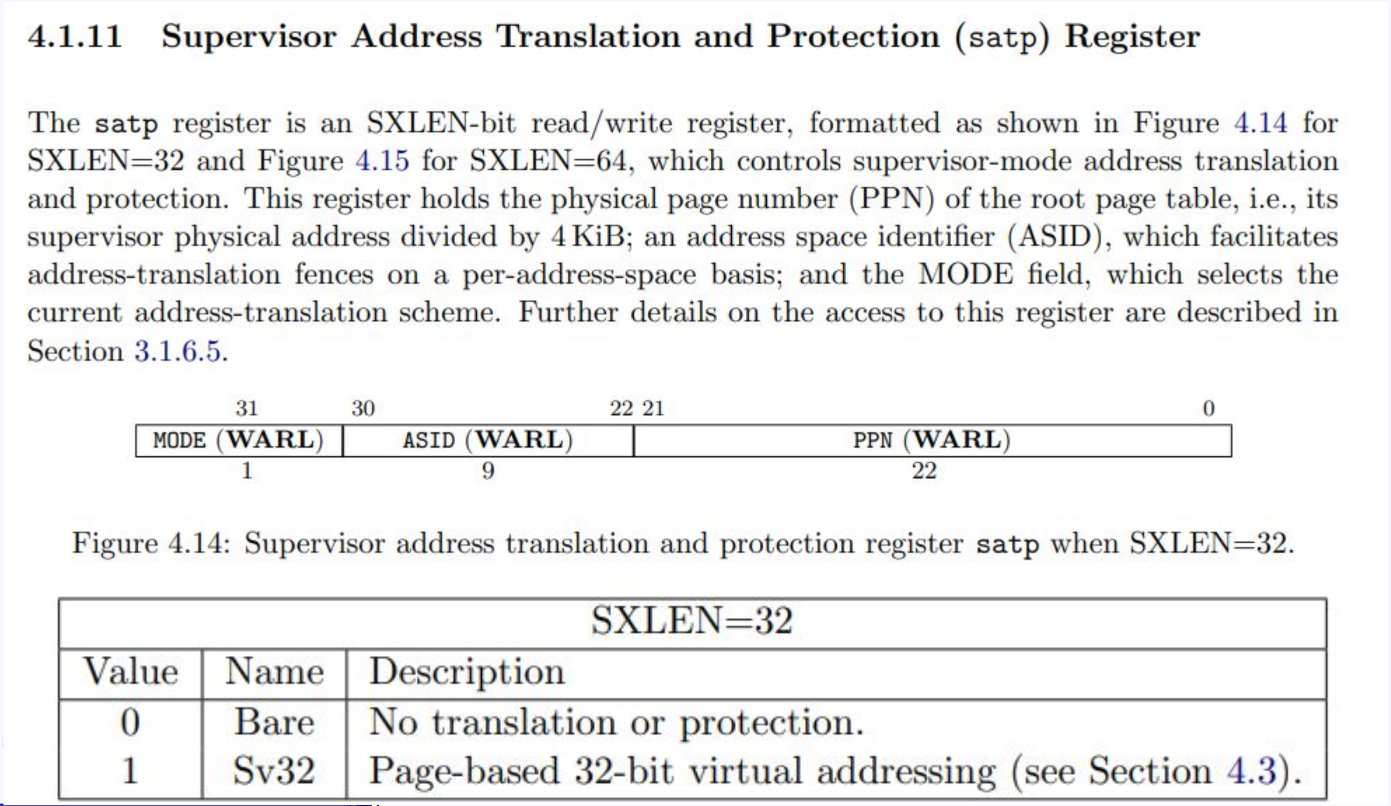


FIGURE 5.29 RISC-V uses four levels of tables to translate a 48-bit virtual address into a 40-bit physical address.

Address translation in RISC-V

RISC-V spec vol. 2



4.1.11 Supervisor Address Translation and Protection (satp) Register

The **satp** register is an SXLEN-bit read/write register, formatted as shown in Figure 4.14 for SXLEN=32 and Figure 4.15 for SXLEN=64, which controls supervisor-mode address translation and protection. This register holds the physical page number (PPN) of the root page table, i.e., its supervisor physical address divided by 4 KiB; an address space identifier (ASID), which facilitates address-translation fences on a per-address-space basis; and the MODE field, which selects the current address-translation scheme. Further details on the access to this register are described in Section 3.1.6.5.

31	30 … 22	21 … 0
MODE (**WARL**)	ASID (**WARL**)	PPN (**WARL**)
1	9	22

Figure 4.14: Supervisor address translation and protection register **satp** when SXLEN=32.

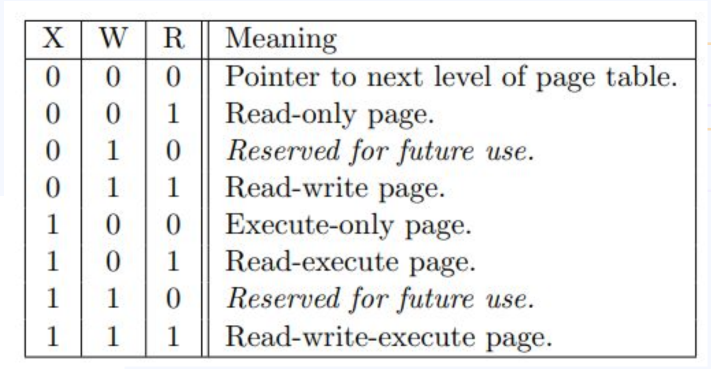
SXLEN=32		
Value	Name	Description
0	Bare	No translation or protection.
1	Sv32	Page-based 32-bit virtual addressing (see Section 4.3).

Virtual memory in RISC-V 32bit

32 bit virtual address space (4kb pages) → 20 bit virtual page numbers (VPNs)

22-bit physical page number (PPN)

Page tables are the size of a page



X	W	R	Meaning
0	0	0	Pointer to next level of page table.
0	0	1	Read-only page.
0	1	0	*Reserved for future use.*
0	1	1	Read-write page.
1	0	0	Execute-only page.
1	0	1	Read-execute page.
1	1	0	*Reserved for future use.*
1	1	1	Read-write-execute page.

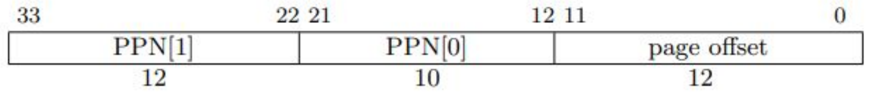


Figure 4.17: Sv32 physical address.

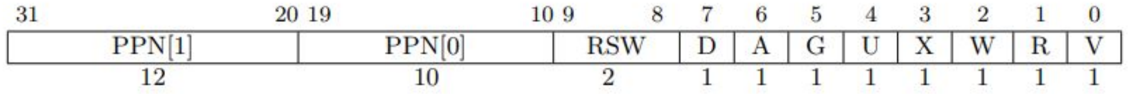
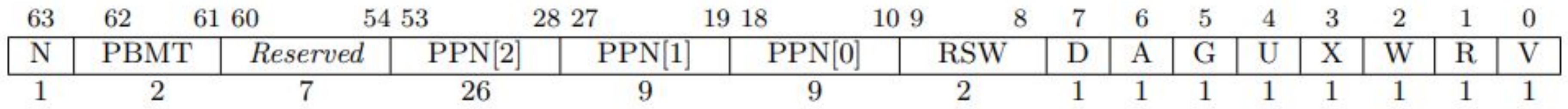


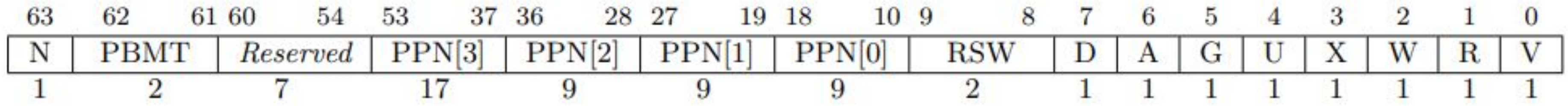
Figure 4.18: Sv32 page table entry.

Virtual memory in RISC-V 64bit



63	62 61	60 54	53 28	27 19	18 10	9 8	7	6	5	4	3	2	1	0
N	PBMT	*Reserved*	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
1	2	7	26	9	9	2	1	1	1	1	1	1	1	1

Figure 4.21: Sv39 page table entry.



63	62 61	60 54	53 37	36 28	27 19	18 10	9 8	7	6	5	4	3	2	1	0
N	PBMT	*Reserved*	PPN[3]	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
1	2	7	17	9	9	9	2	1	1	1	1	1	1	1	1

Figure 4.24: Sv48 page table entry.