User Request (Virtual Memory)

Please read 1-review-fully-associative-cache.md and then 1-review-direct-mapped-cache.md . Please assume that someone has read those two files in that order. I now want to write a third file that describes the page table in virtual memory. I want to leverage the knowledge in the first two files. Please describe the page table as a direct-mapped cache in which the data block is actually a pointer to the page frame in RAM (or an indication of non-resident). Please also explain that during a page fault, the page fault handler will read the page into RAM from the swapfile. Please explain the language of a "virtual page" (e.g., VPN: virtual page number) and "physical page (e.g.: PFN: page frame number).. After that, please explain that the page table and the swapfile both include not only the"page number" (the tag of the address for that page), but also the process id. (to Google Gemini, Oct. 27, 2025)

> *FROM GOOGLE GEMINI: That's a fantastic way to connect the concepts! By leveraging the structure of a direct-mapped cache, we can easily understand the Page Table's function in virtual memory.*

The Page Table: A Direct-Mapped Cache for Memory Translation

You have already learned about the **fully associative cache** and the **direct-mapped cache**. We can use the structure of a direct-mapped cache to easily understand how a computer system translates a virtual address into a physical address using a **Page Table**.

The Page Table as a Direct-Mapped Structure

In a direct-mapped cache, a memory block is mapped to a specific cache line using an **Index**. The Page Table operates similarly, serving as a lookup structure to map large blocks of virtual memory to physical memory frames.

Cache Concept	Virtual Memory Concept	Description
Direct-Mapped Cache	**Page Table**	The primary structure used to store the mappings.
Cache Line	**Page Table Entry (PTE)**	A single entry in the table that holds the mapping information.

Cache Concept	Virtual Memory Concept	Description
Index	**Virtual Page Number (VPN)**	The lower bits of the virtual address, used to directly select the *one* specific PTE in the Page Table.
Tag	**Virtual Page Number (VPN)**	The remaining higher bits of the virtual address, which must be stored in the PTE for verification.
Block Offset	**Page Offset**	The remaining lowest bits of the address, which do not change during translation and select the specific byte *within* the page.
Data Block	**Physical Page Frame Number (PPN or PFN)**	The data returned on a "hit," which is the pointer (address) to where the data actually resides in RAM.

The Translation "Hit"

When the CPU issues a **Virtual Address**, it is split into two main parts: the **Virtual Page Number (VPN)** and the **Page Offset**.

1. **Index/Lookup:** The **VPN** acts as the **Index** to select the single, specific **Page Table Entry (PTE)** that might hold the translation.
2. **Tag Check (Implicit):** In a traditional Page Table implementation, the VPN *is* the index, making the tag match trivial if the entry is valid. The check is simplified: Does the PTE exist, and is it valid?
3. **Data Retrieval:** If the PTE is **Valid** ($V = 1$), the PTE returns the **Physical Page Frame Number (PFN)**.
4. **Physical Address Construction:** The PFN is concatenated with the original **Page Offset** to form the final, actual **Physical Address (PA)** in RAM.

Status Bits and Page Faults

Just like a cache line has a **Valid Bit** (V) and a **Modified/Dirty Bit** (M), a Page Table Entry (PTE) has similar status fields:

- **Valid Bit:** This bit indicates if the virtual page is currently resident in **Physical RAM**.
  - $V = 1$: **Page Hit**. The PFN is valid, and the data is in memory.
  - $V = 0$: **Page Miss (or Page Fault)**. The page is not in RAM, and the memory management unit (MMU) must raise a trap to the operating system's kernel.

Handling a Page Fault

When a **Page Fault** occurs (a miss, $V = 0$), the operating system's **Page Fault Handler** takes over:

1. It locates the requested **virtual page** on the secondary storage, which is usually a dedicated space called the **swapfile** or **pagefile**.
2. It finds an available **Physical Page Frame** in RAM (or evicts an existing page using an algorithm like LRU, similar to what you read about in the fully associative cache description).
3. It reads the entire page of data from the **swapfile** into the chosen **Physical Page Frame** in RAM.
4. It updates the PTE: it writes the new **PFN** into the entry and sets the **Valid Bit (V) to 1**.
5. It returns control to the CPU, which then re-issues the instruction, allowing the memory access to successfully complete on the second try.

The Role of the Process ID

In a modern multitasking operating system, the same **Virtual Page Number (VPN)** may be used by different applications (processes).

To maintain isolation and ensure the correct translation, the operating system effectively gives each **Process** its own complete **Page Table**.

Therefore, when the system performs a memory lookup, it uses:

- **Process ID (PID):** Identifies the active program and selects its specific Page Table.
- **Virtual Page Number (VPN):** Used to index into the selected Page Table.

This same principle extends to the **swapfile**. The operating system must track which pages belong to which process, ensuring that during a page fault, the handler reads the correct virtual page belonging to the correct **(Process ID + VPN)** pair from the swapfile and loads it into the correct physical page frame.

The Role of the PTE (Digging Deeper)

FIXME: Review Quiz 3, and later add more concepts from Quiz 3 into this document.