

2018年度(平成30年度)版

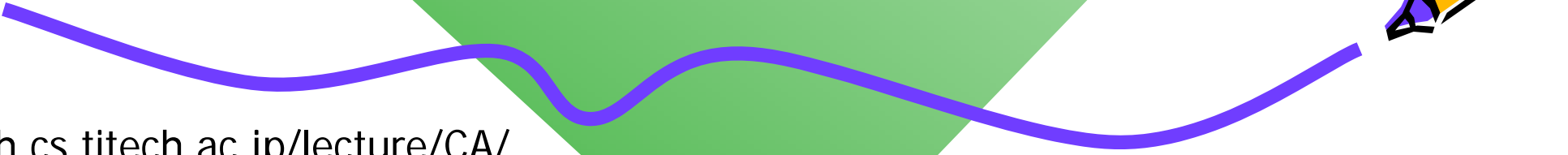
Ver. 2018-11-05a

Course number: CSC.T363



コンピュータアーキテクチャ Computer Architecture

10. 仮想記憶、セキュリティ Virtual Memory and Security

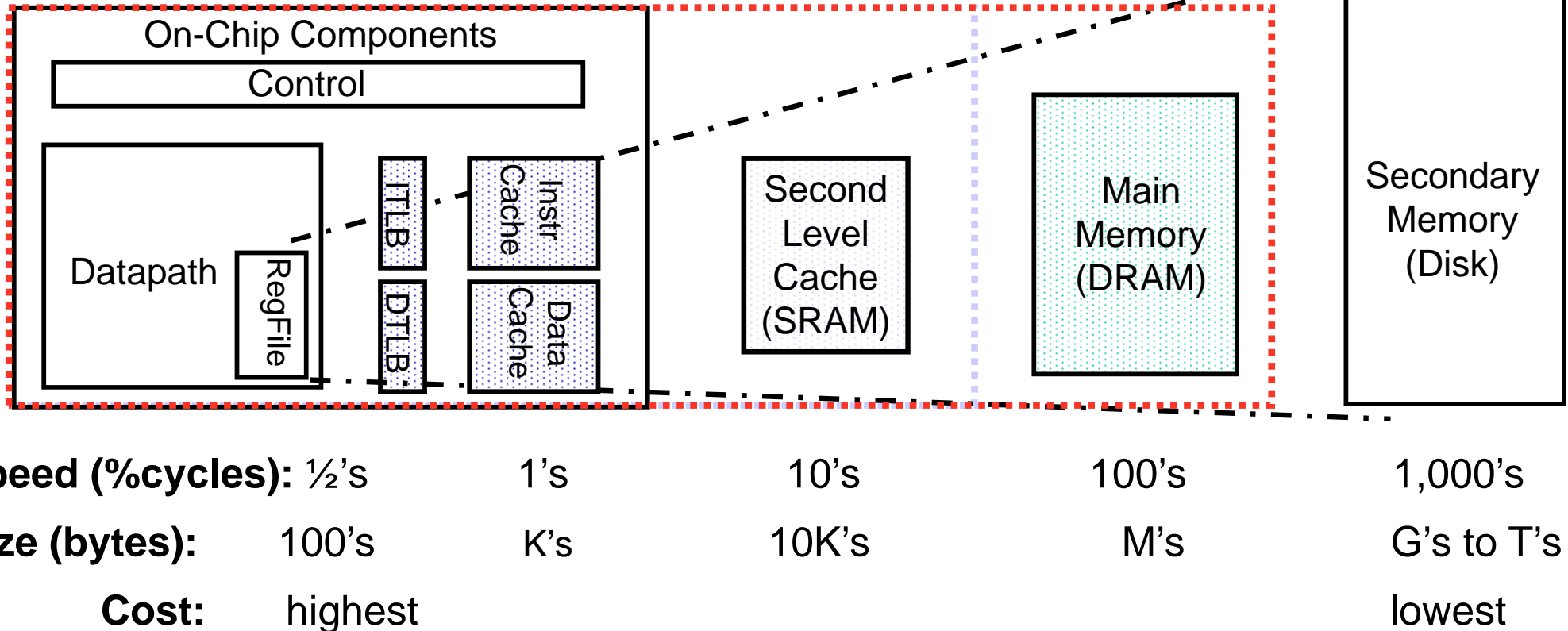


www.arch.cs.titech.ac.jp/lecture/CA/
Room No.W321
Tue 13:20-16:20, Fri 13:20-14:50

吉瀬 謙二 情報工学系
Kenji Kise, Department of Computer Science
kise_at_c.titech.ac.jp

A Typical Memory Hierarchy

- By taking advantage of **the principle of locality** (局所性)
 - Present **much memory** in **the cheapest technology**
 - at **the speed of fastest technology**



TLB: Translation Lookaside Buffer

Example of 32-bit memory space (4GB)

0x00000000

00000000 00000000 00000000 00000000₂ = 0₁₀



2GB Memory !

```
kterm
top - 11:35:26 up 10 days, 19:49, 2 users, load average: 0.01, 0.01, 0
Tasks: 164 total, 1 running, 163 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si,
Mem: 4002924k total, 3252404k used, 750520k free, 181808k buffers
Swap: 6062072k total, 0k used, 6062072k free, 2570804k cached

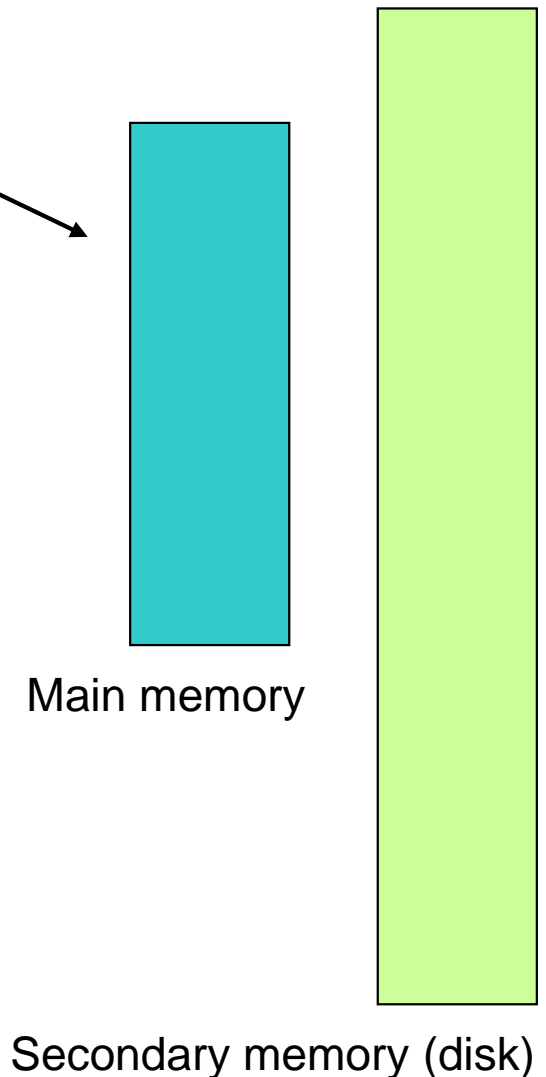
  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
    1 root        15   0 10348   696  584  S   0.0   0.0   0:01.00  init
    2 root         RT  -5     0     0     0  S   0.0   0.0   0:00.00  migration/0
    3 root         34  19     0     0     0  S   0.0   0.0   0:00.00  ksoftirqd/0
    4 root         RT  -5     0     0     0  S   0.0   0.0   0:00.00  watchdog/0
    5 root         RT  -5     0     0     0  S   0.0   0.0   0:00.00  migration/1
    6 root         34  19     0     0     0  S   0.0   0.0   0:00.00  ksoftirqd/1
    7 root         RT  -5     0     0     0  S   0.0   0.0   0:00.00  watchdog/1
    8 root         RT  -5     0     0     0  S   0.0   0.0   0:00.01  migration/2
    9 root         34  19     0     0     0  S   0.0   0.0   0:00.00  ksoftirqd/2
   10 root         RT  -5     0     0     0  S   0.0   0.0   0:00.00  watchdog/2
```

0xFFFFFFFF

11111111 11111111 11111111 11111111₂ = 4,294,967,296 - 1₁₀

Virtual Memory (仮想記憶)

- Use main memory as a “cache” for secondary memory
 - Provides the ability to easily run programs larger than the size of physical memory
 - Simplifies loading a program for execution by providing for code relocation (i.e., the code can be loaded anywhere in main memory)
 - Allows efficient and safe sharing of memory among multiple programs
- Security, memory protection
 - control memory access rights



Virtual Memory

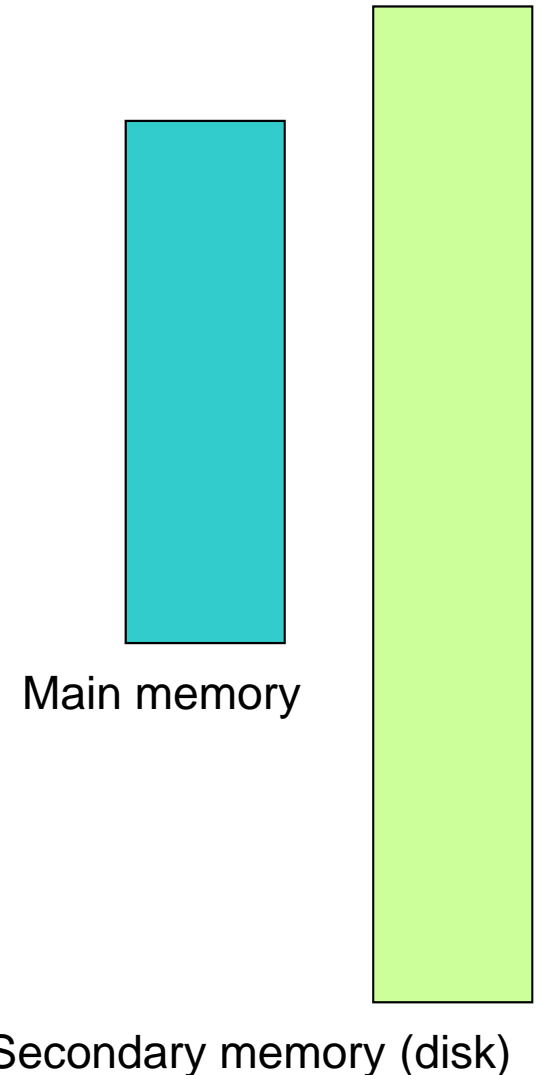


- What makes it work? – again the Principle of Locality
 - A program is likely to access a relatively small portion of its address space during any period of time



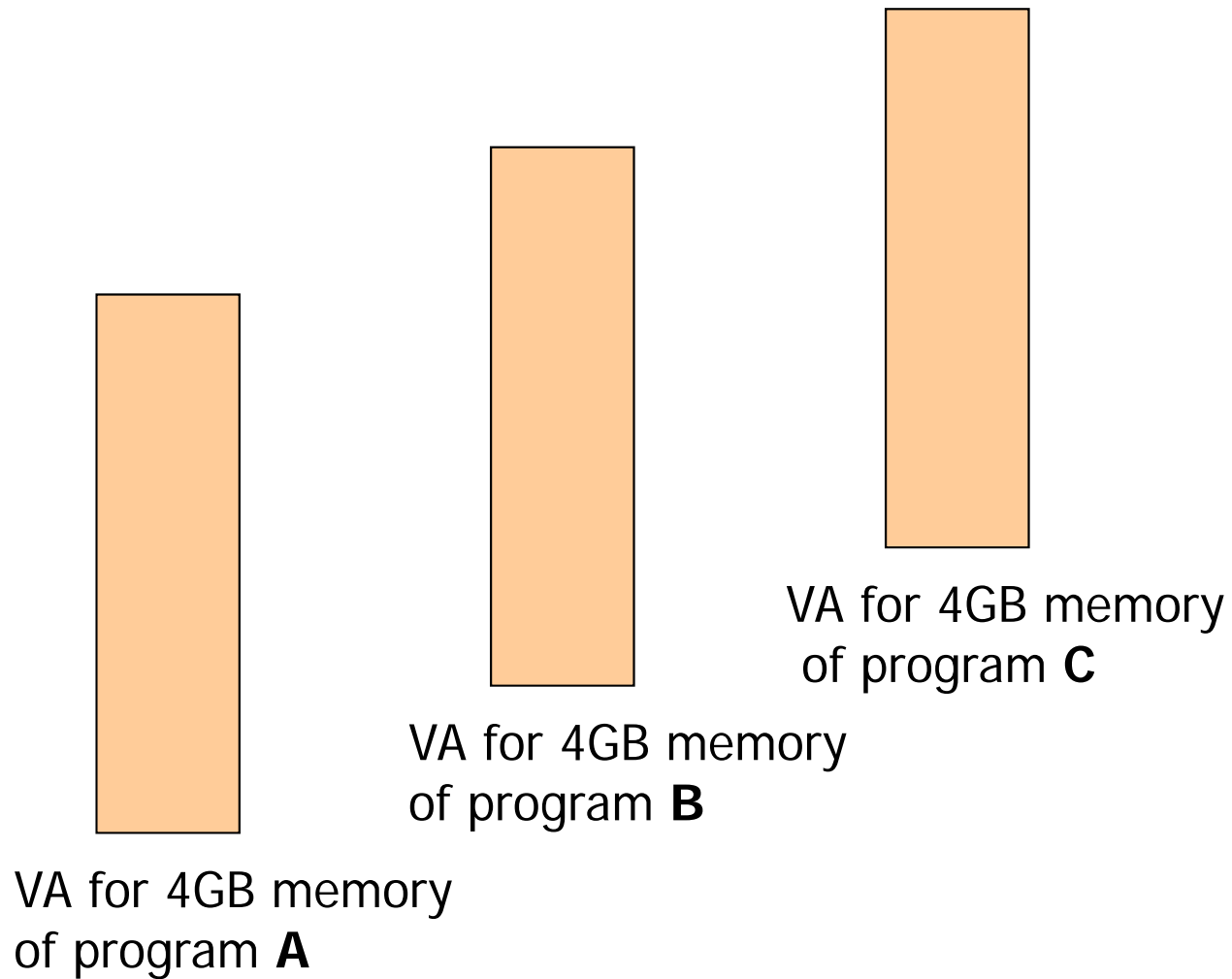
Virtual Memory

- Each program is compiled into its own address space – a “**virtual address (VA)**” space
- **Physical address (PA)** for the access of physical devices
 - During run-time each **virtual address, VA** must be translated to a **physical address, PA**

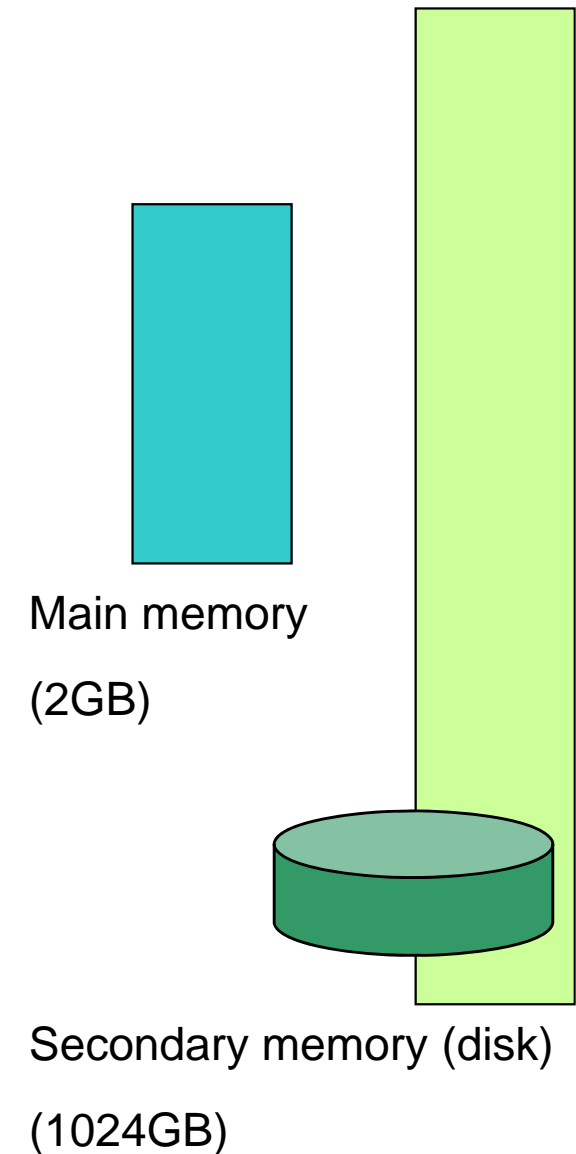


Virtual Memory

Virtual address world



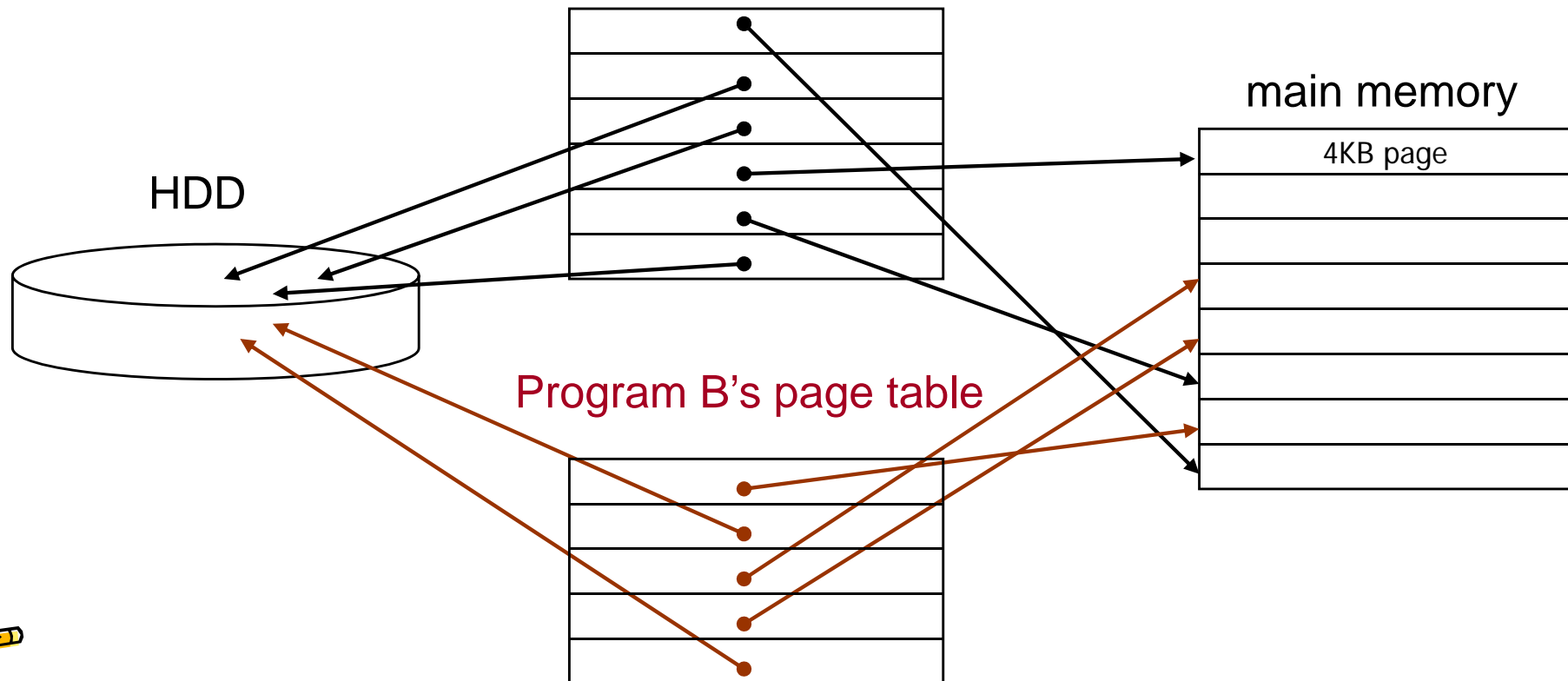
Physical address world



Two Programs Sharing Physical Memory

- A program's address space is divided into **pages** (all one fixed size, typical 4KB) or **segments** (variable sizes)
 - The starting location of each page (either in **main memory** or in **secondary memory**) is contained in the program's **page table**

Program A's **page table** (virtual address space)

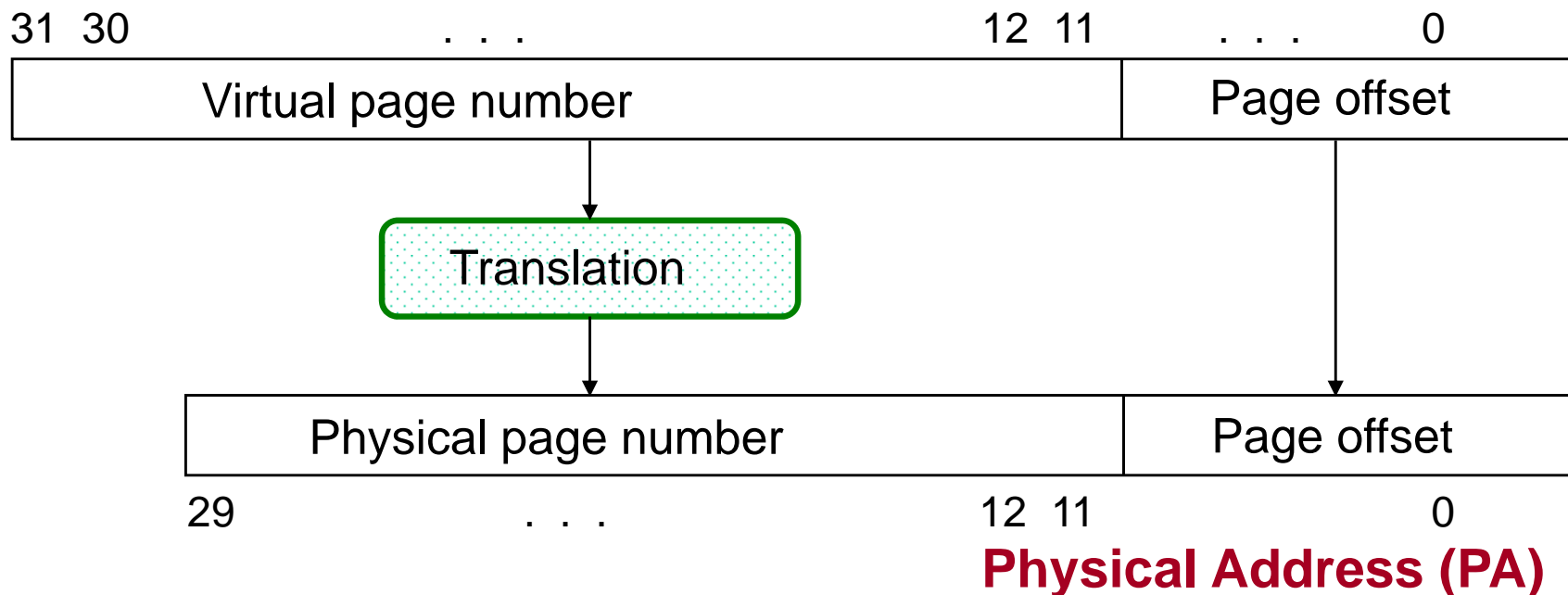


Address Translation

- A virtual address is translated to a physical address by a combination of hardware and software

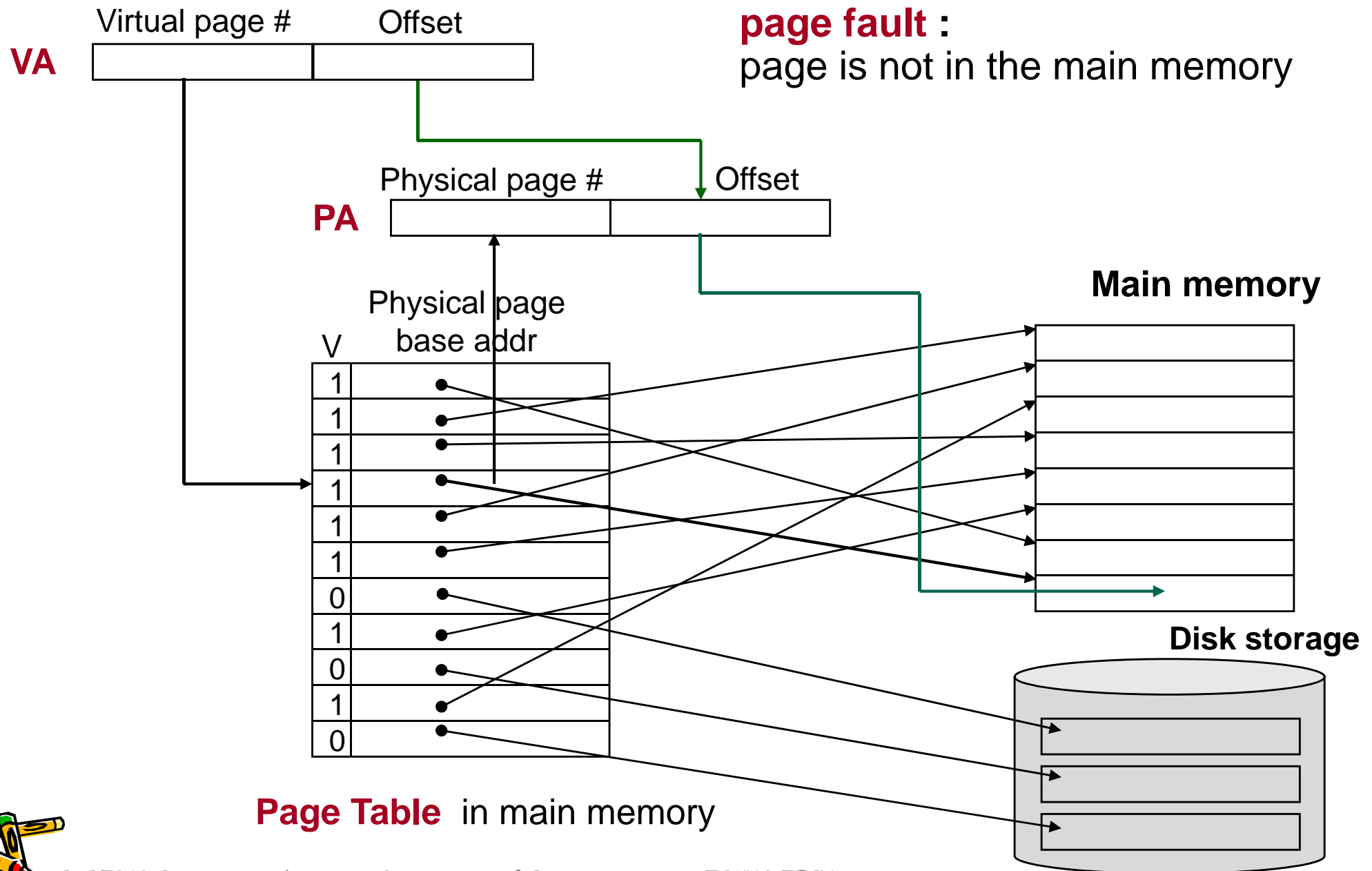
Virtual Address (VA)

Assume 4KB page size



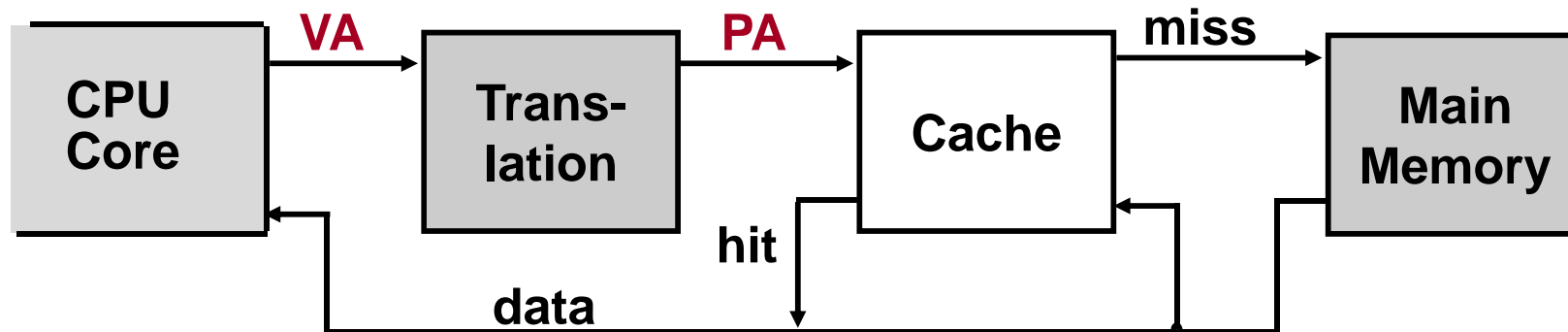
- So each memory request **first** requires an **address translation** from the virtual space to the physical space

Address Translation Mechanisms

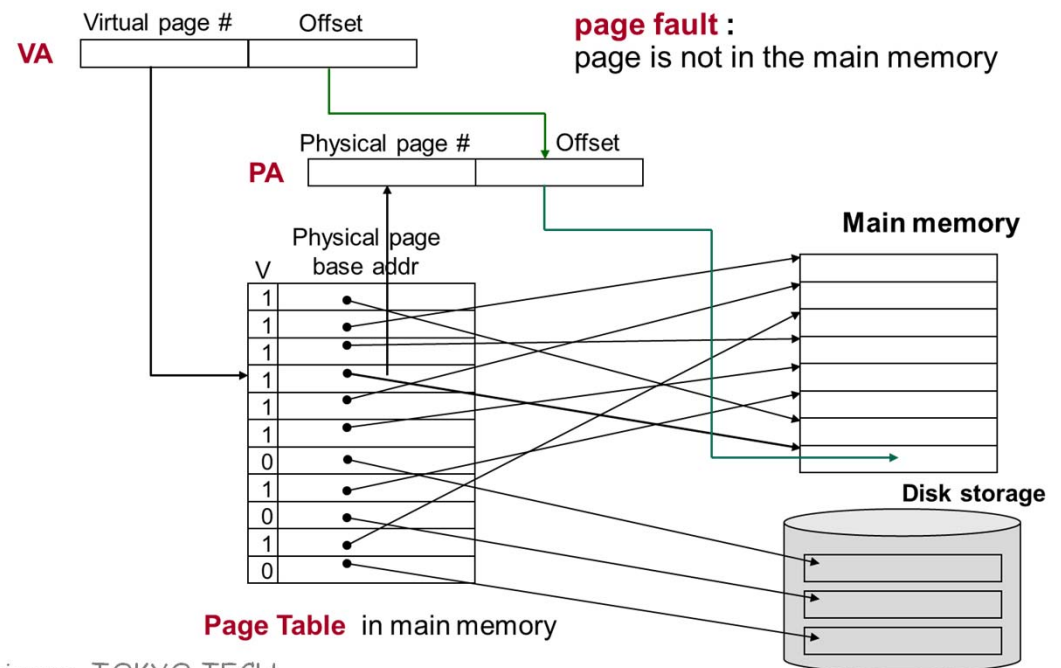


Virtual Addressing, the hardware fix


- Thus it may take an **extra memory access** to translate a virtual address to a physical address



- This makes memory (cache) accesses **very expensive** (if every access was really two accesses)
- What's the solution ?**



Virtual Addressing, the hardware fix

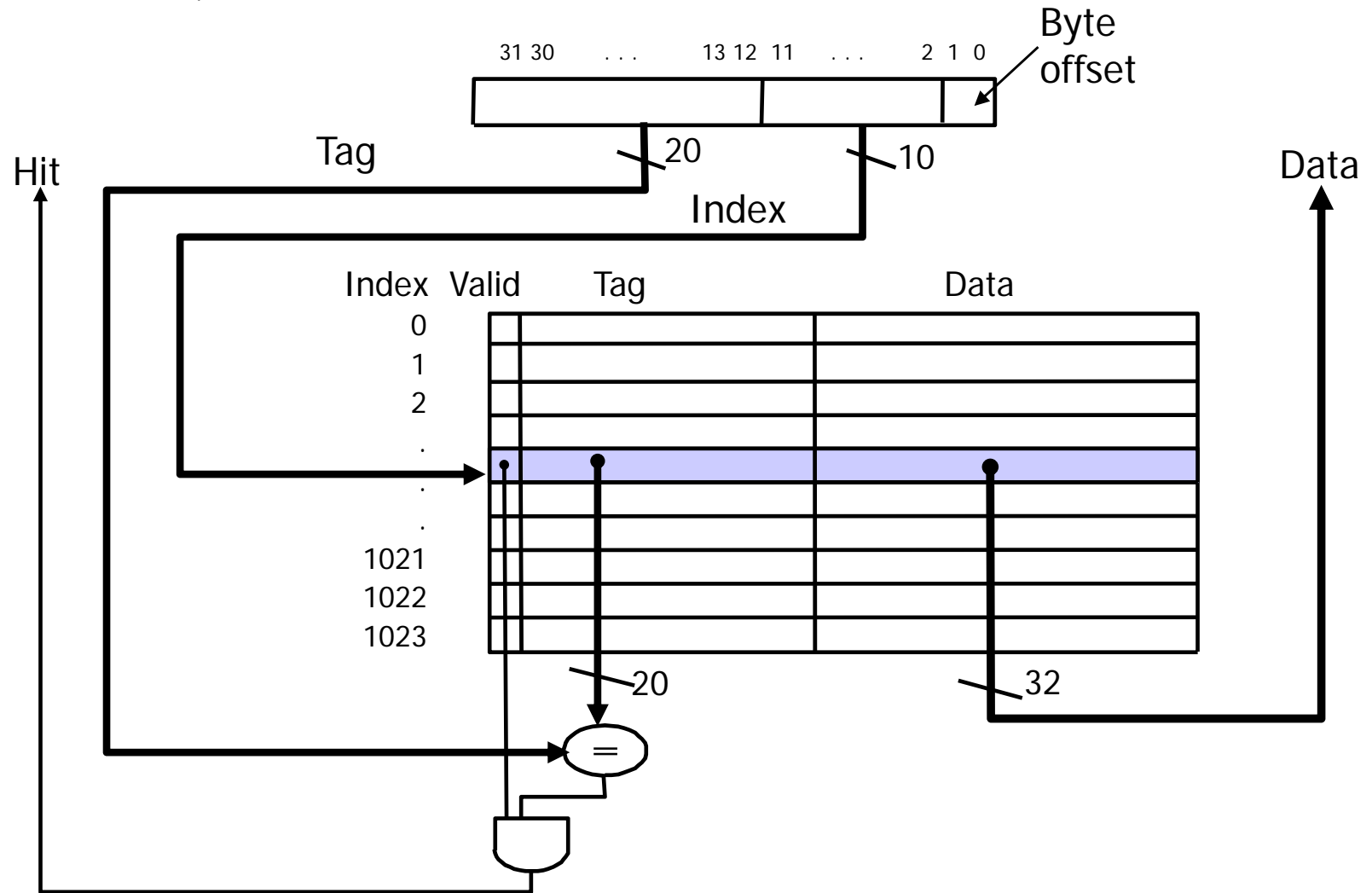


- The hardware fix is to use a **Translation Lookaside Buffer (TLB)** (アドレス変換バッファ)
 - a small **cache** that keeps track of recently used address mappings to avoid having to do a **page table** lookup



MIPS Direct Mapped Cache Example

- One word/block, cache size = 1K words



What kind of locality are we taking advantage of?



Translation Lookaside Buffers (TLBs)

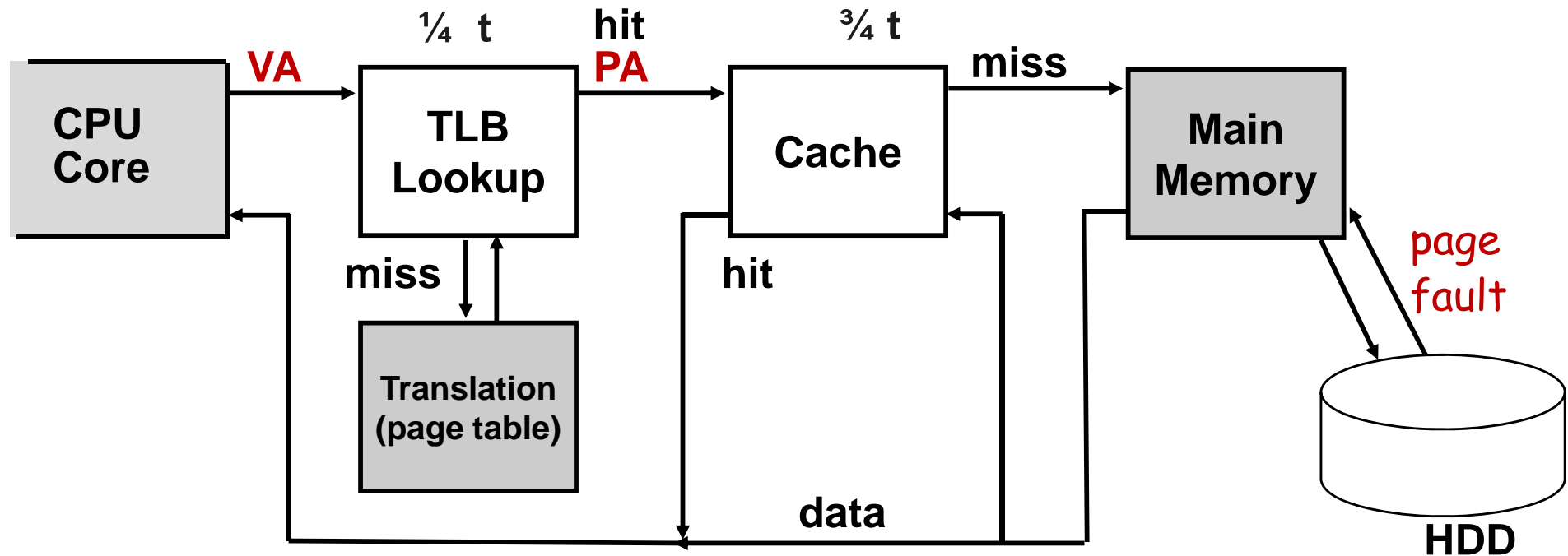
- Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

V	Virtual Page #	Physical Page #			

- TLB access time is typically smaller than cache access time (because TLBs are much smaller than caches)
 - TLBs are typically not more than 128 to 256 entries even on high end machines

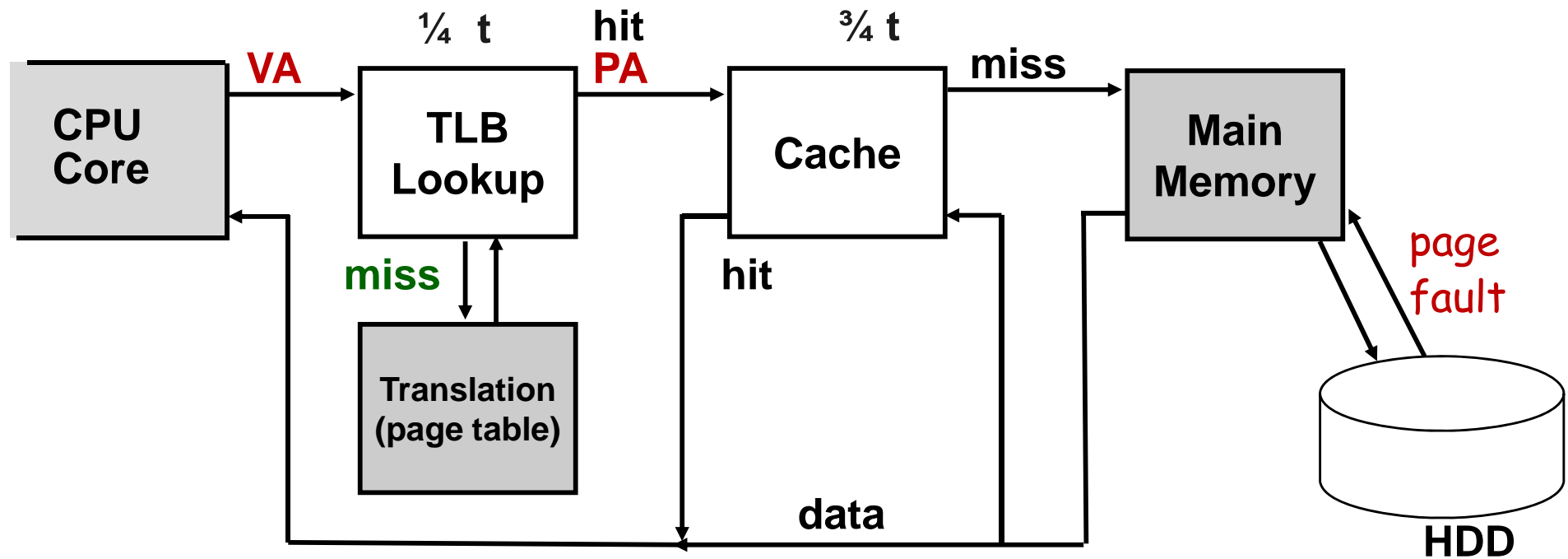


A TLB in the Memory Hierarchy



- A **TLB miss** – is it a TLB miss or a page fault ?
 - If the page is in main memory, then the TLB miss can be handled (in hardware or software) by loading the translation information from the page table into the TLB
 - Takes 100's of cycles to find and load the translation info into the TLB
 - If the page is not in main memory, then it's a true **page fault**
 - Takes 1,000,000's of cycles to service a page fault

A TLB in the Memory Hierarchy



- **page fault** : page is not in physical memory
- **TLB misses** are much more frequent than true page faults





Two Machines' TLB Parameters

	Intel P4	AMD Opteron
TLB organization	<p>1 TLB for instructions and 1 TLB for data</p> <p>Both 4-way set associative</p> <p>Both use ~LRU replacement</p> <p>Both have 128 entries</p> <p>TLB misses handled in hardware</p>	<p>2 TLBs for instructions and 2 TLBs for data</p> <p>Both L1 TLBs fully associative with ~LRU replacement</p> <p>Both L2 TLBs are 4-way set associative with round-robin LRU</p> <p>Both L1 TLBs have 40 entries</p> <p>Both L2 TLBs have 512 entries</p> <p>TBL misses handled in hardware</p>



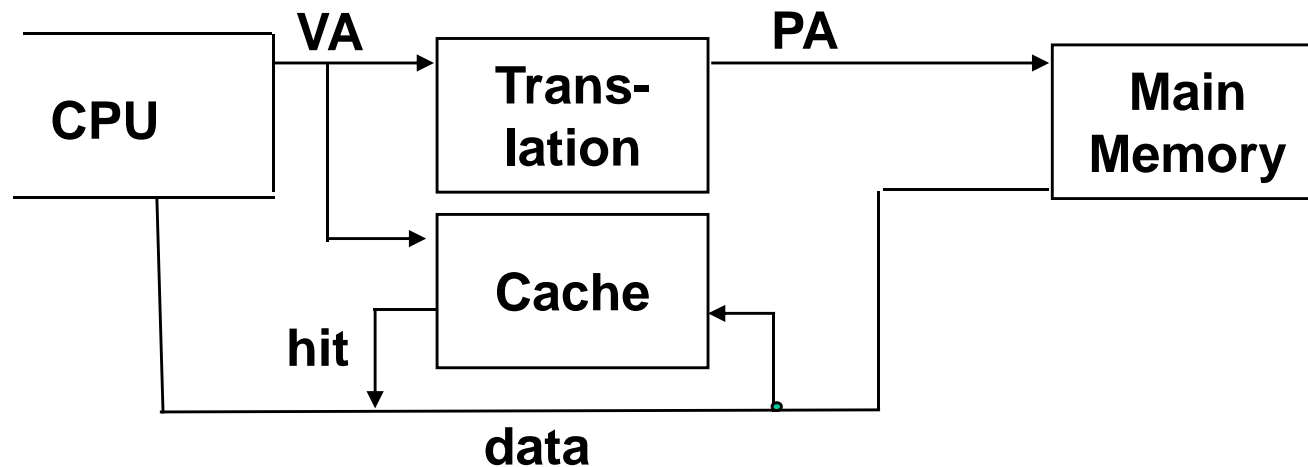
TLB Event Combinations



TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – what we want!
Hit	Hit	Miss	Yes – although the page table is not checked if the TLB hits
Miss	Hit	Hit	Yes – TLB miss, PA in page table
Miss	Hit	Miss	Yes – TLB miss, PA in page table, but data not in cache
Miss	Miss	Miss	Yes – page fault
Hit	Miss	Miss/ Hit	Impossible – TLB translation not possible if page is not present in memory
Miss	Miss	Hit	Impossible – data not allowed in cache if page is not in memory

Why Not a Virtually Addressed Cache?

- A **virtually addressed cache** would only require address translation on cache misses



but

- Two different virtual addresses can map to the same physical address (when processes are sharing data),
- Two different cache entries hold data for the same physical address
 - **synonyms** (別名)
 - Must update all cache entries with the same physical address or the memory becomes inconsistent



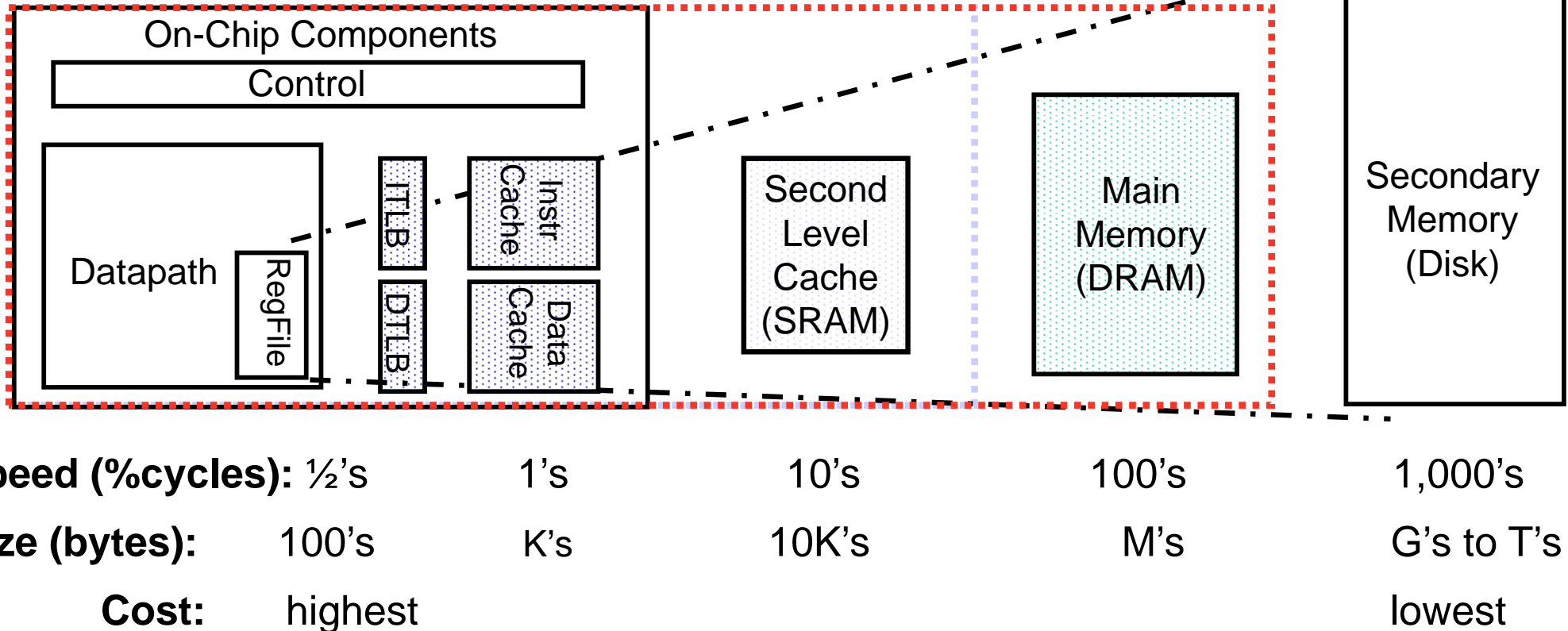
The Hardware/Software Boundary

- What parts of the virtual to physical address translation is done by or assisted by the hardware?
 - Translation Lookaside Buffer (TLB) that caches the recent translations
 - TLB access time is part of the cache hit time
 - May cause an extra stage in the pipeline for TLB access
 - Page table storage, fault detection and updating
 - Page faults result in interrupts (precise) that are then handled by the OS
 - Hardware must support (i.e., update appropriately) Dirty and Reference bits (e.g., ~LRU) in the Page Tables



A Typical Memory Hierarchy

- By taking advantage of **the principle of locality** (局所性)
 - Present **much memory** in **the cheapest technology**
 - at **the speed of fastest technology**



TLB: Translation Lookaside Buffer