2009-07-09



# 計算機アーキテクチャ 第一 (E)

## 10. 主記憶とファイルメモリの管理, 多重仮想記憶, 記憶保護

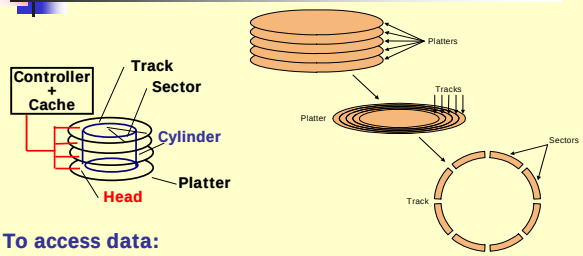
吉瀬 謙二 計算工学専攻
kise_at_cs.titech.ac.jp
W641講義室 木曜日13:20 ー 14:50

## Acknowledgement


- Lecture slides for Computer Organization and Design, Third Edition, courtesy of **Professor Mary Jane Irwin**, Penn State University
- Lecture slides for Computer Organization and Design, third edition, Chapters 1-9, courtesy of **Professor Tod Amon,** Southern Utah University.

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005


## Disk Drives

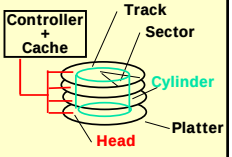



**To access data:**

- **seek time (シーク時間)**: position head over the proper track
- **rotational latency (回転待ち時間)**: wait for desired sector
- **transfer time (転送時間)**: grab the data (one or more sectors)
- **Controller time(制御時間)**: the overhead the disk controller imposes in performing a disk I/O access



## Magnetic Disk Characteristic




- **Disk read/write components**
  1. **Seek time**: position the head over the proper track (**3 to 14 ms avg**)
     - due to locality of disk references the actual average seek time may be only 25% to 33% of the advertised number
  2. **Rotational latency**: wait for the desired sector to rotate under the head (½ of 1/RPM converted to ms)
     - 0.5/5400RPM = **5.6ms** to 0.5/15000RPM = **2.0ms**
  3. **Transfer time**: transfer a block of bits (one or more sectors) under the head to the disk controller's cache (**30 to 80 MB/s** are typical disk transfer rates)
  4. **Controller time**: the overhead the disk controller imposes in performing a disk I/O access (**typically < .2 ms**)



## SSD (Solid State Drive)




## SSD (Solid State Drive)

## Reliability（信頼性）, Availability

- Reliability – measured by the mean time to failure (平均故障寿命, MTTF). Service interruption is measured by mean time to repair (平均修復時間, MTTR)
- Availability（アベイラビリティ）

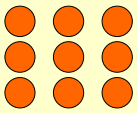
  Availability = MTTF / (MTTF + MTTR)

- To increase MTTF, either improve the quality of the components or design the system to continue operating in the presence of faulty components
  1. Fault avoidance: preventing fault occurrence by construction
  2. Fault tolerance: using redundancy to correct or bypass faulty components (hardware)

7

## **RAID**: Disk Arrays

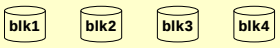
**Redundant Array of Inexpensive Disks**



- Arrays of small and inexpensive disks
  - Increase potential **throughput** by having many disk drives
    - Data is spread over multiple disk
    - Multiple accesses are made to several disks at a time
- **Reliability** is lower than a single disk
- But **availability** can be improved by adding redundant disks (RAID)

8

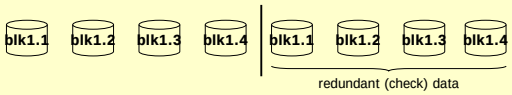
## RAID: **Level 0** (冗長性なし; Striping ストライピング)




- Multiple smaller disks as opposed to one big disk
  - Spreading the blocks over multiple disks – **striping** – means that multiple blocks can be accessed in parallel increasing the performance
    - A 4 disk system gives four times the throughput of a 1 disk system
  - Same cost as one *big* disk – assuming 4 small disks cost the same as one big disk
- No redundancy, so what if one disk fails?

9

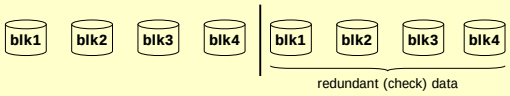
## RAID: **Level 1** (Redundancy via Mirroring)




- Uses twice as many disks for redundancy so there are always two copies of the data
  - The number of redundant disks = the number of data disks so twice the cost of one big disk
    - writes have to be made to both sets of disks, so writes would be only 1/2 the performance of RAID 0
- What if one disk fails?
  - If a disk fails, the system just goes to the "**mirror**" for the data

10

## RAID: **Level 0+1** (Striping with Mirroring)




- Combines the best of RAID 0 and RAID 1, data is striped across four disks and mirrored to four disks
  - Four times the throughput (due to striping)
  - # redundant disks = # of data disks so twice the cost of one big disk
    - writes have to be made to both sets of disks, so writes would be only 1/2 the performance of RAID 0
- What if one disk fails?
  - If a disk fails, the system just goes to the "**mirror**" for the data

11

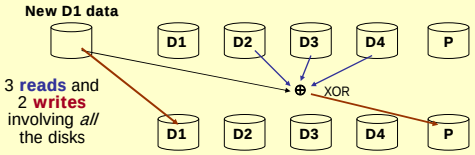
## RAID: Level 4 (Block-Interleaved Parity)


- Cost of higher availability still only 1/N but the parity is stored as **blocks** associated with sets of data blocks
  - Four times the throughput (striping)
  - # redundant disks = 1 × # of protection groups
  - Supports "small reads" and "small writes" (reads and writes that go to just one (or a few) data disk in a protection group)

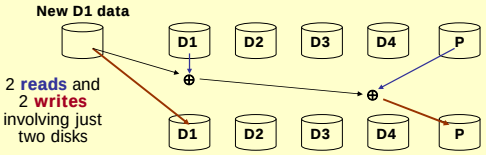
12

## Small Writes

- RAID 3




- RAID 4 small writes




13

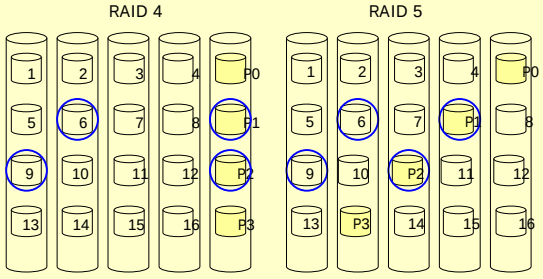
## RAID: Level 5 (Distributed Block-Interleaved Parity)

one of these assigned as the block parity disk

- Cost of higher availability still only 1/N but the parity block can be located on any of the disks
  so there is no single bottleneck for writes
  - Still four times the throughput (striping)
  - # redundant disks = 1 × # of protection groups
  - Supports "**small reads**" and "**small writes**" (reads and writes that go to just one (or a few) data disk in a protection group)
  - Allows **multiple simultaneous writes**

14

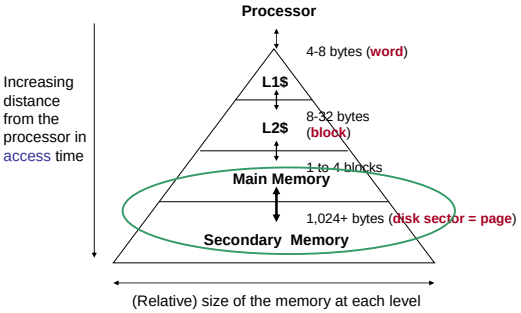
## Distributing Parity Blocks




- By distributing parity blocks to all disks, some small writes can be performed **in parallel**

15

## Memory Hierarchy



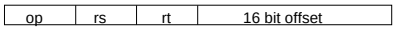

16

## Loading and Storing Bytes

- MIPS has two basic data transfer instructions for accessing memory

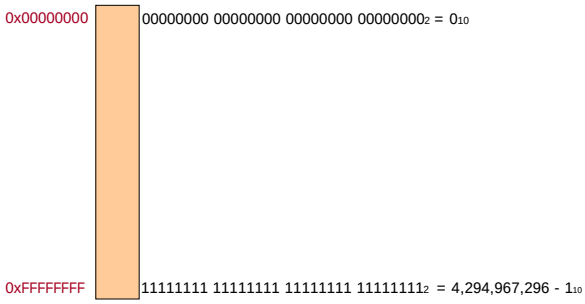
```
lw  $t0, 4($s3)  # load word from memory
sw  $t0, 8($s3)  # store word to memory
```

- The data is loaded into (**lw**) or stored from (**sw**) a register in the register file
- The memory address – a 32 bit address – is formed by adding the contents of the base address register to the offset value




17

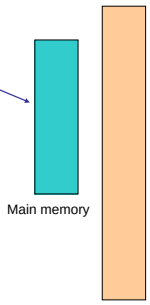
## 例：32ビットのメモリ空間




18

## Virtual Memory（仮想記憶）

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




19

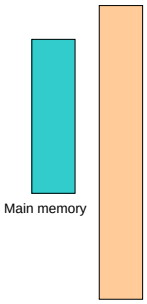
## Virtual Memory（仮想記憶）

- 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

20

## Virtual Memory（仮想記憶）

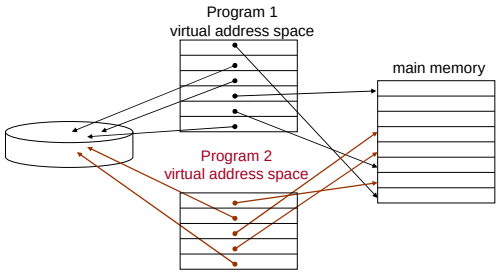
- Each program is compiled into its own address space – a "virtual" address space
  - During run-time each **virtual address, VA**（仮想アドレス）must be translated to a **physical address, PA**（物理アドレス）




21

## Two Programs Sharing Physical Memory

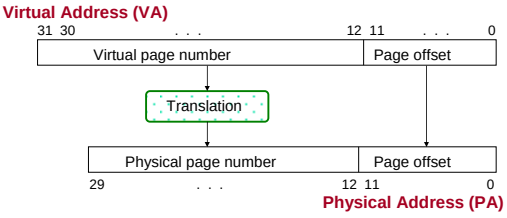
- A program's address space is divided into **pages** (all one fixed size) or **segments** (variable sizes)
  - The starting location of each page (either in **main memory** or in **secondary memory**) is contained in the program's **page table**




22

## Address Translation

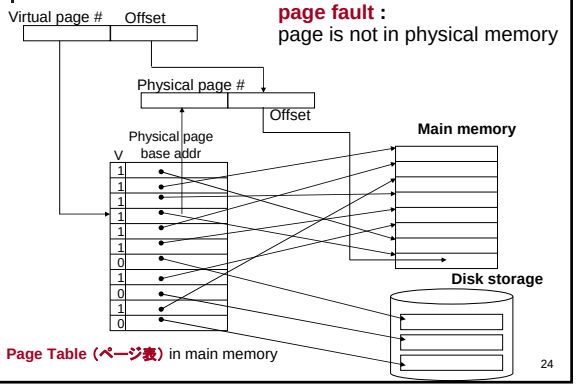
- A virtual address is translated to a physical address by a combination of hardware and software




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

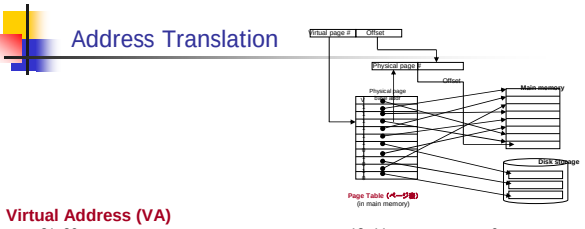
23

## Address Translation Mechanisms

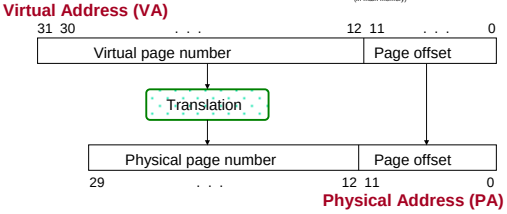



24

## Address Translation



**Virtual Address (VA)**



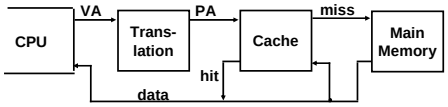

**Physical Address (PA)**

- ページサイズは？ ページ表のメモリサイズは？



## Virtual Addressing with a **Cache**

- Thus it takes 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)
- The hardware fix is to use a **Translation Lookaside Buffer (TLB)** – a small cache that keeps track of recently used address mappings to avoid having to do a page table lookup

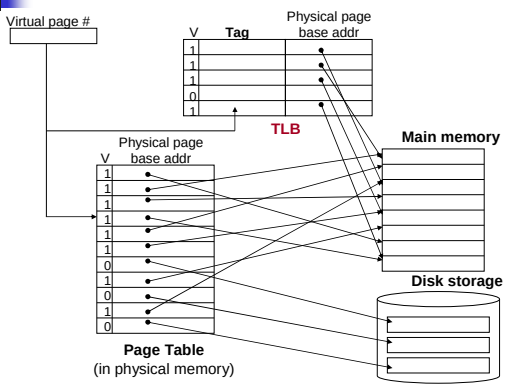


## Virtual Addressing, the hardware fix

- 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



## Making Address Translation Fast






## Translation Lookaside Buffers (TLBs)

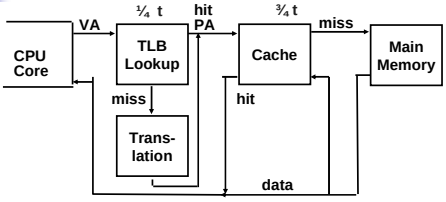
- Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

| V | Virtual Page # | Physical Page # | Dirty | Ref | |
|---|---|---|---|---|---|
| | | | | | |

- 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

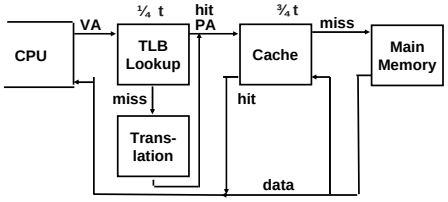


## A TLB in the Memory Hierarchy




- A TLB miss – is it a page fault or merely a TLB miss?
  - If the page is loaded into 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 10'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 | 1 TLB for instructions and 1TLB for data<br>Both 4-way set associative<br>Both use ~LRU replacement<br>Both have 128 entries<br>TLB misses handled in hardware | 2 TLBs for instructions and 2 TLBs for data<br>Both L1 TLBs fully associative with ~LRU replacement<br>Both L2 TLBs are 4-way set associative with round-robin LRU<br>Both L1 TLBs have 40 entries<br>Both L2 TLBs have 512 entries<br>TBL misses handled in hardware |



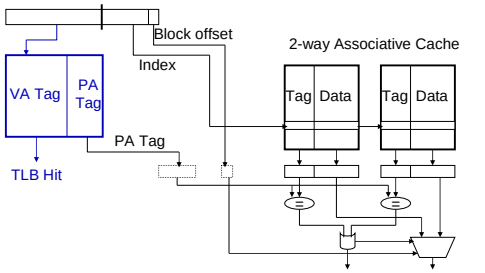
## TLB Event Combinations

| 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 |



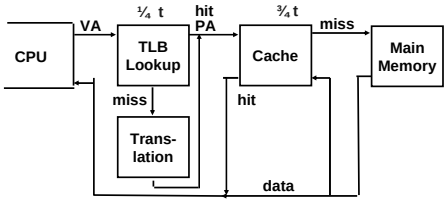
## Reducing Translation Time

- Can **overlap** the cache access with the TLB access
  - Works when the high order bits of the VA are used to access the TLB while the low order bits are used as index into cache






## A TLB in the Memory Hierarchy

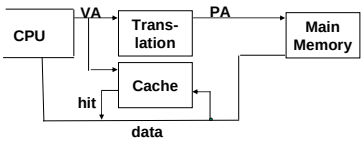



- **page fault** : page is not in physical memory
- **TLB misses** are much more frequent than true page faults



## 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（别名）**
  - 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



## Summary

- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time.
    - **Temporal Locality**: Locality in Time
    - **Spatial Locality**: Locality in Space
- Caches, TLBs, Virtual Memory all understood by examining how they deal with the four questions
  1. Where can block be placed?
  2. How is block found?
  3. What block is replaced on miss?
  4. How are writes handled?
- **Page tables** map virtual address to physical address
  - **TLBs** are important for fast translation