2024年度(令和6年)版

Ver. 2024-11-12a

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

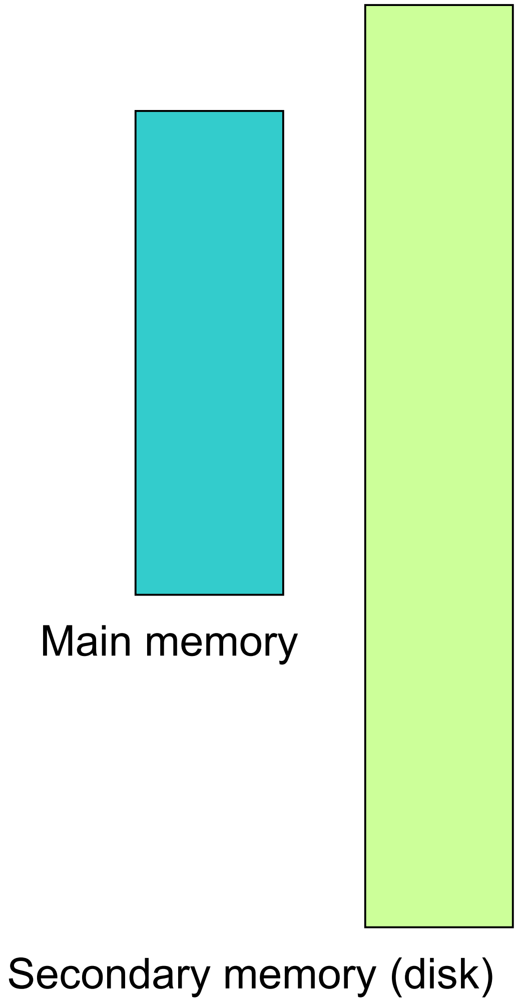
11. 仮想記憶 (2), 信頼性
Virtual Memory (2), dependability

www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

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

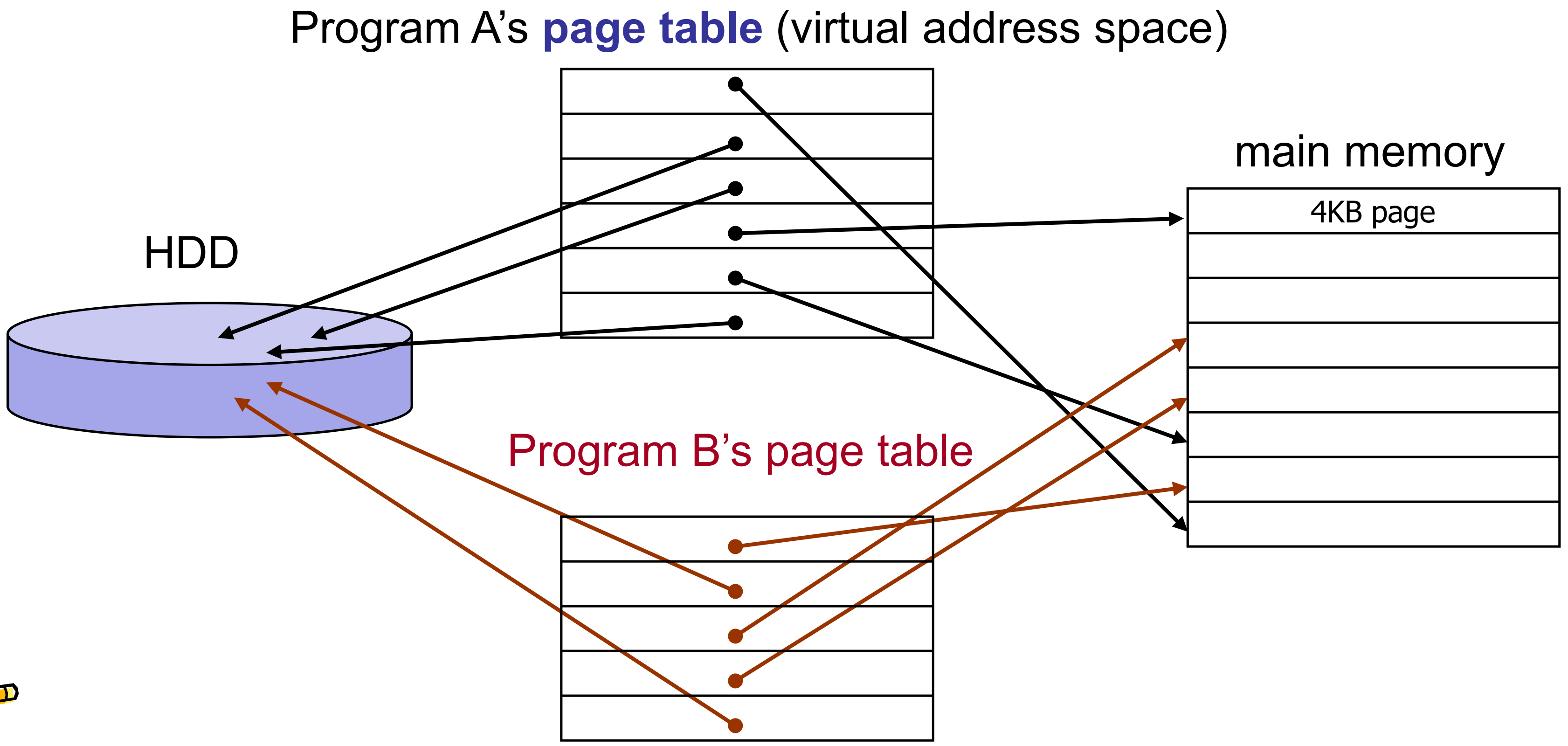
Virtual Memory

- 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



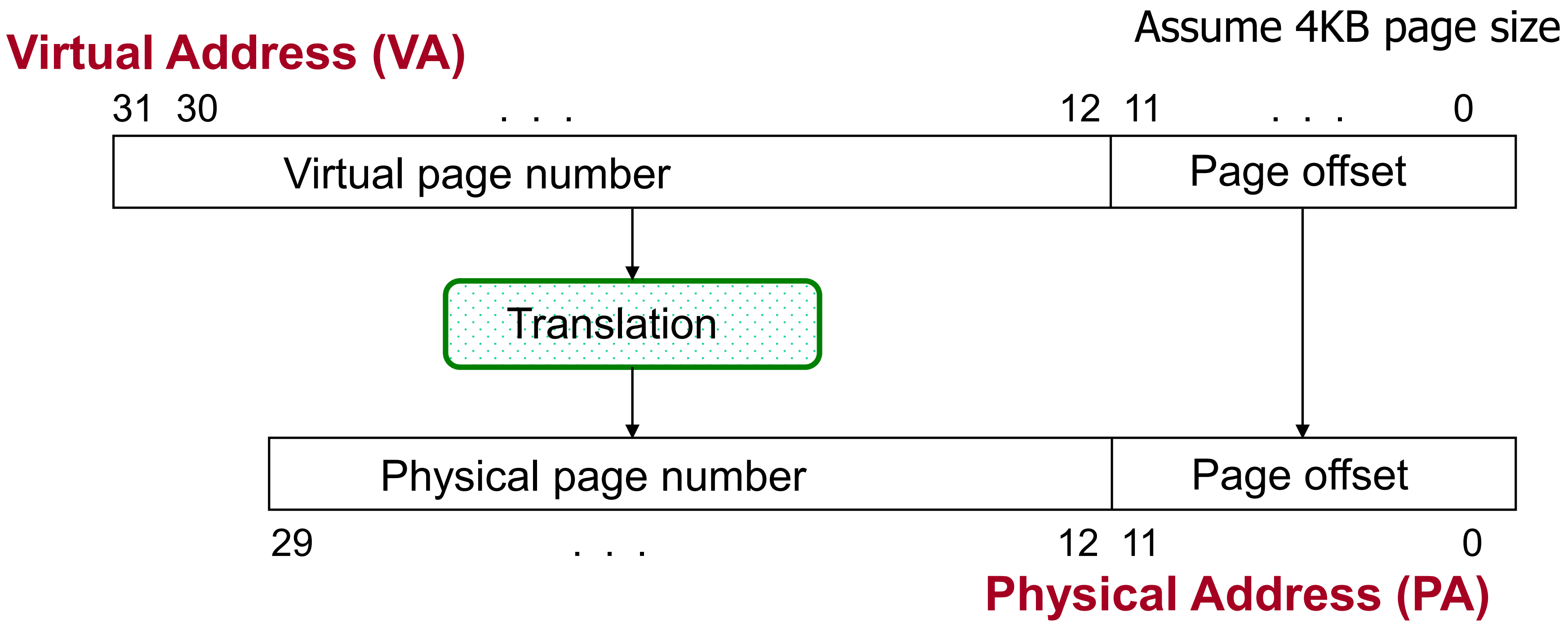
Two Programs Sharing Physical Memory

- 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**



Address Translation

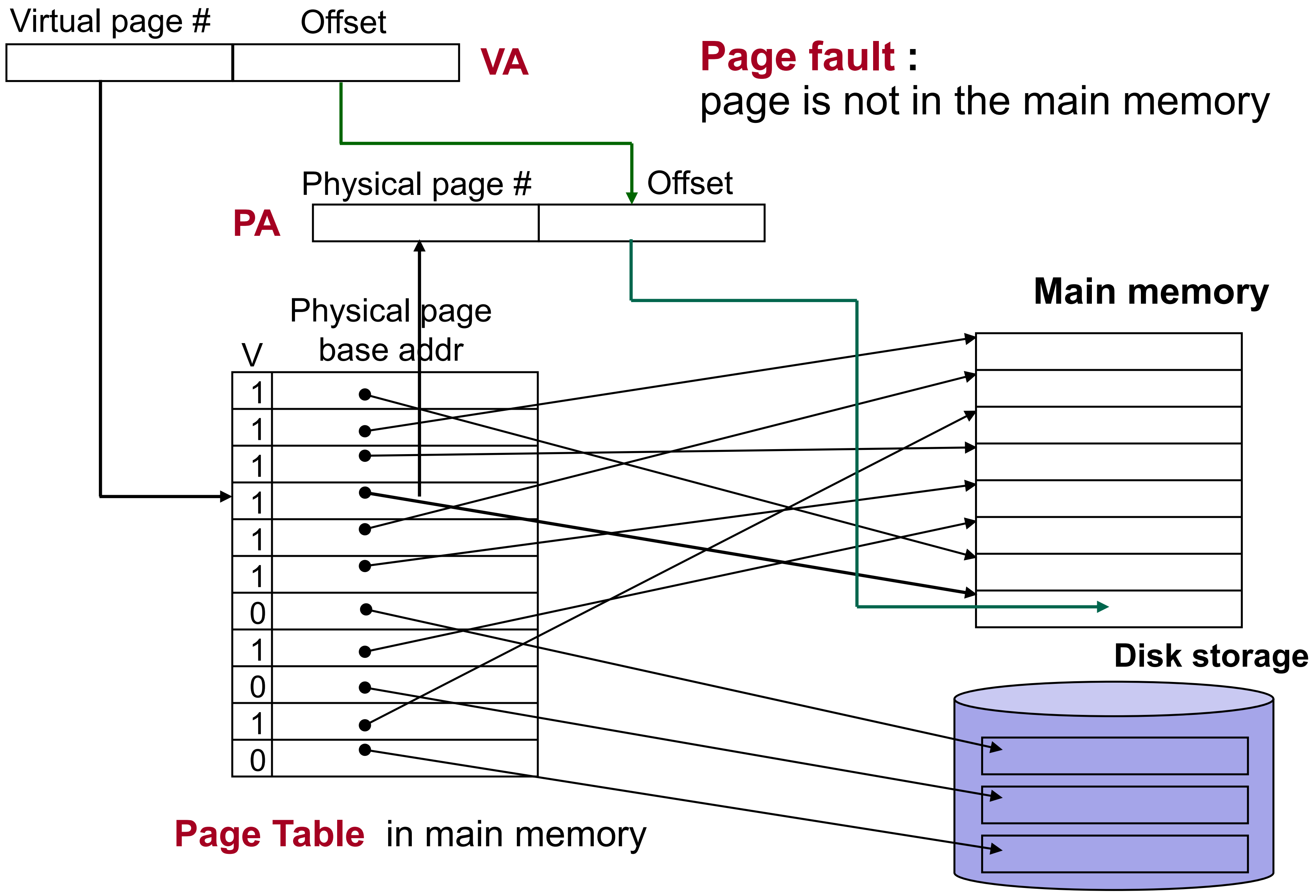
- A virtual address is translated to a physical address by a combination of hardware and software



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

Address Translation Mechanisms

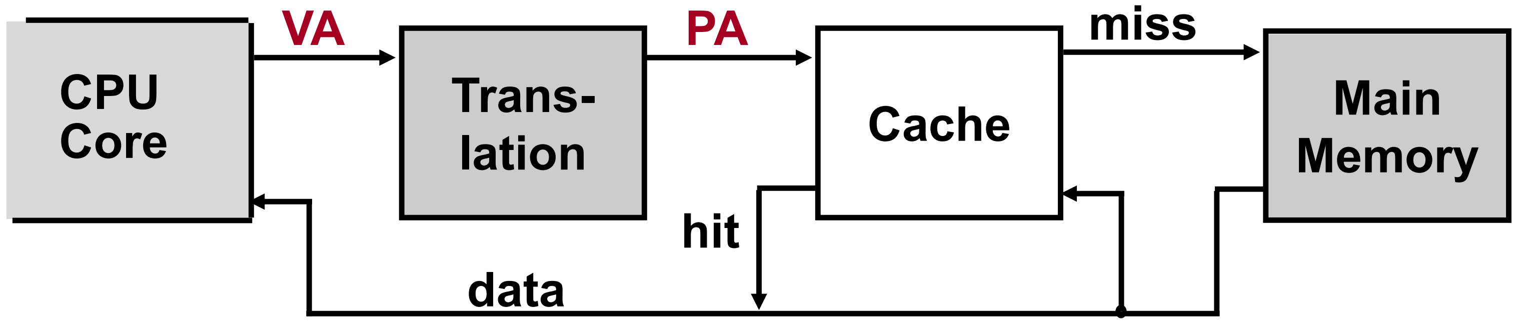
Page fault :
page is not in the main memory



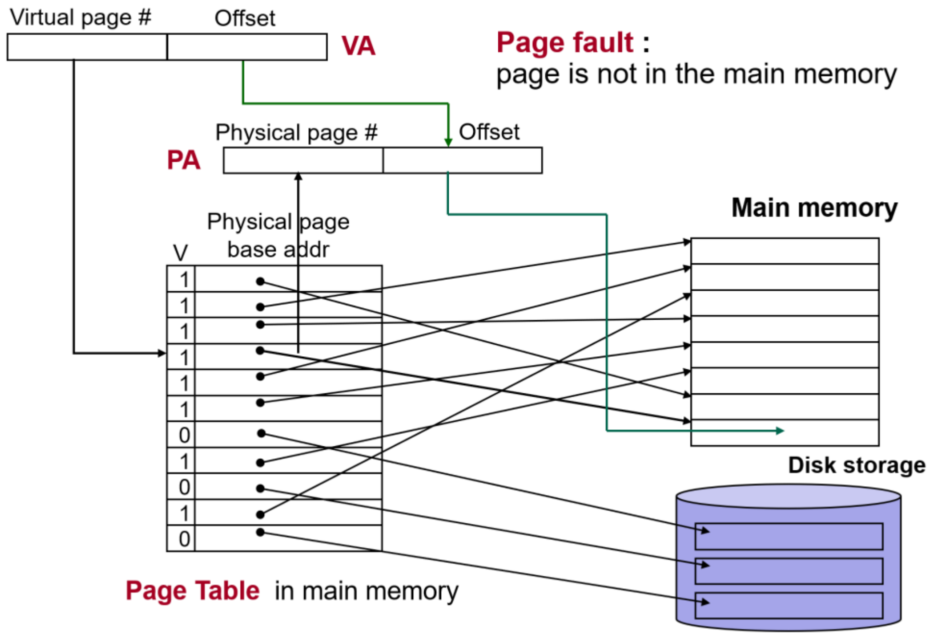
Page Table in main memory

Virtual Addressing, the hardware fix

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



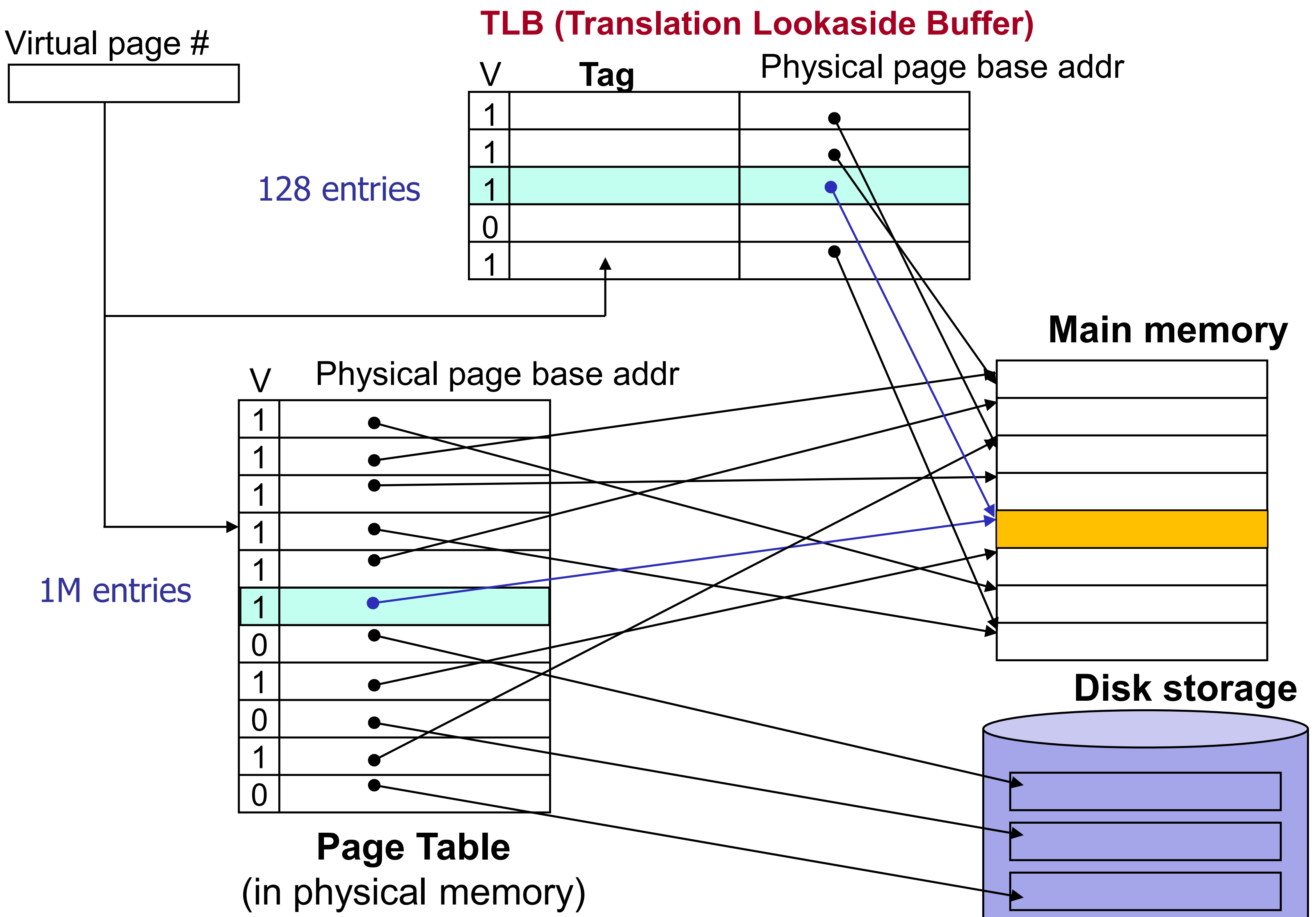
- This makes memory (cache) accesses very expensive (if every access was really **two** accesses)
- What's the solution ?



Virtual Addressing, the hardware fix

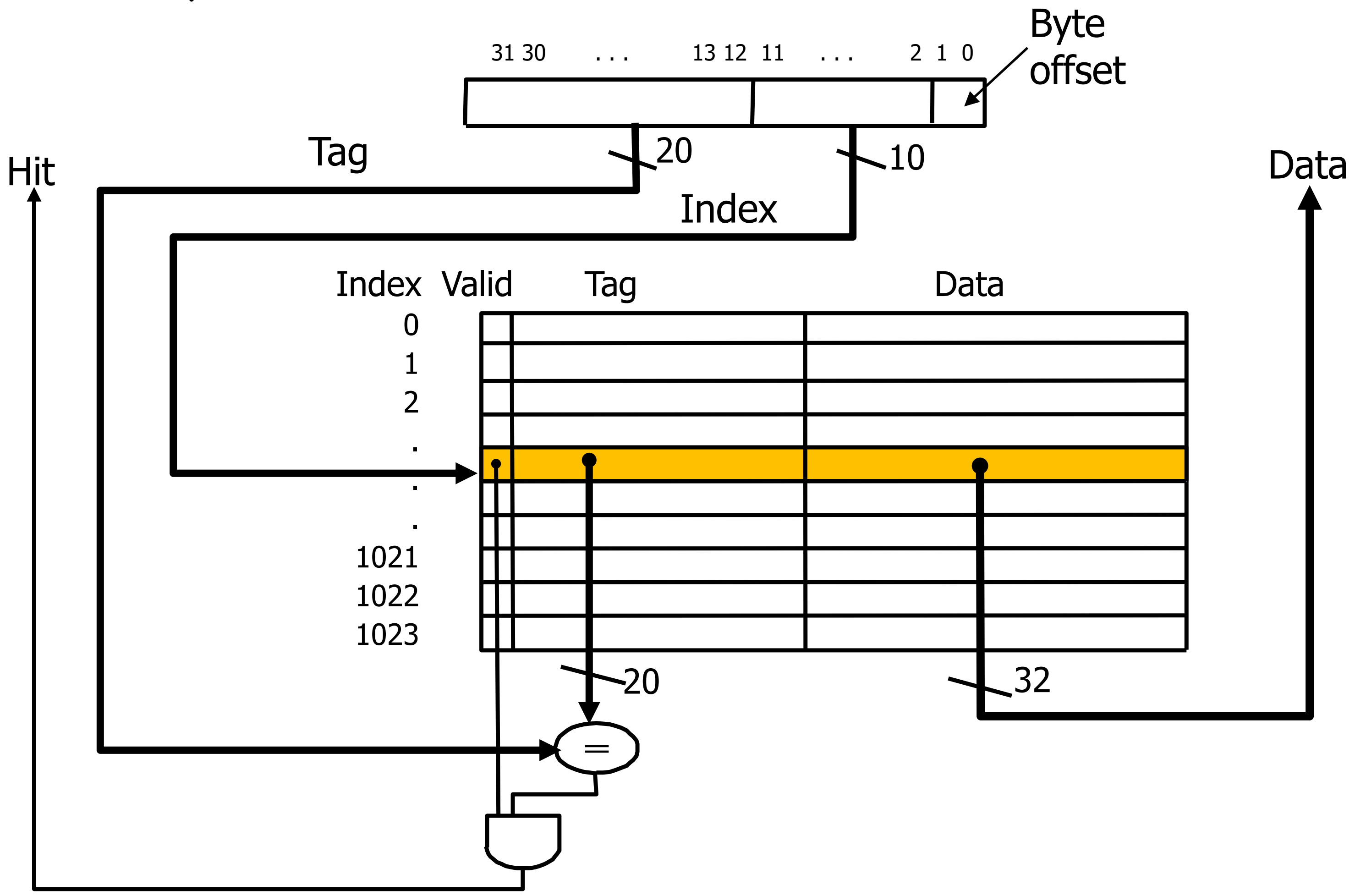
- The hardware fix is to use a Translation Lookaside Buffer (TLB)（アドレス変換バッファ）
  - a small cache that keeps track of recently used address mappings to avoid having to do a page table lookup

Making Address Translation Fast



Direct Mapped Cache Example

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



What kind of locality are we taking advantage of?

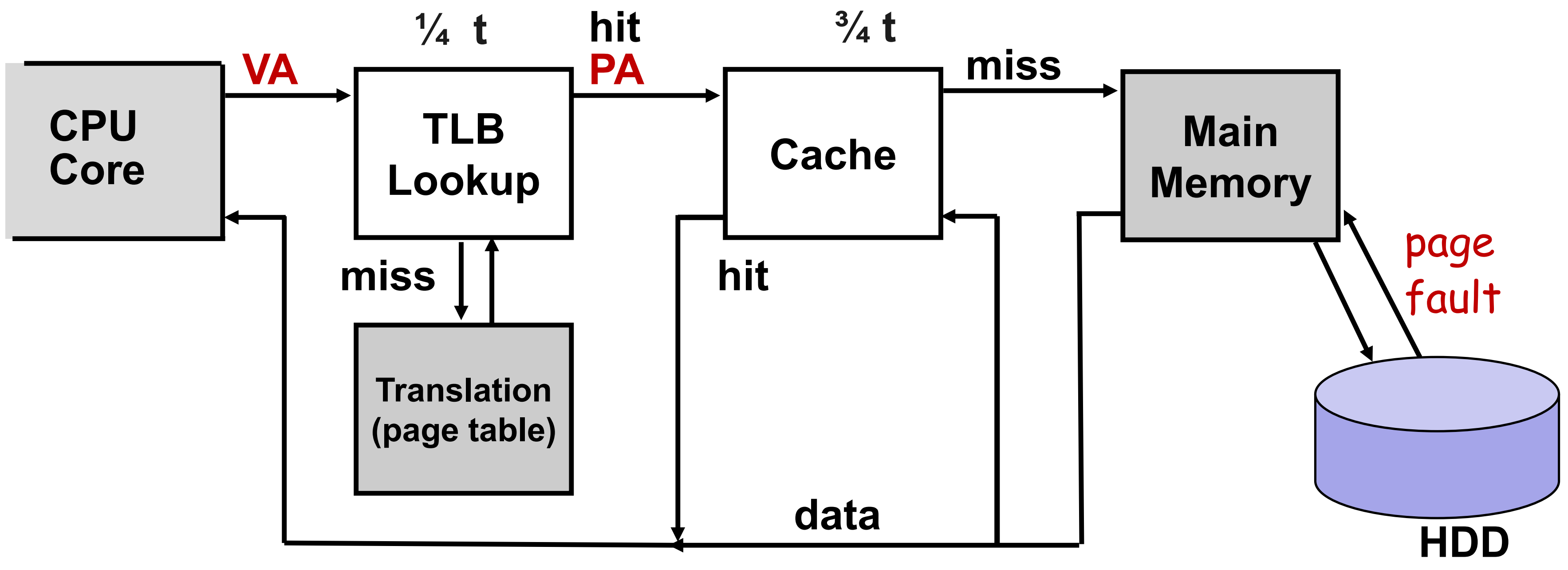
Translation Lookaside Buffers (TLBs)

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

V	Virtual Page #	Physical Page #			

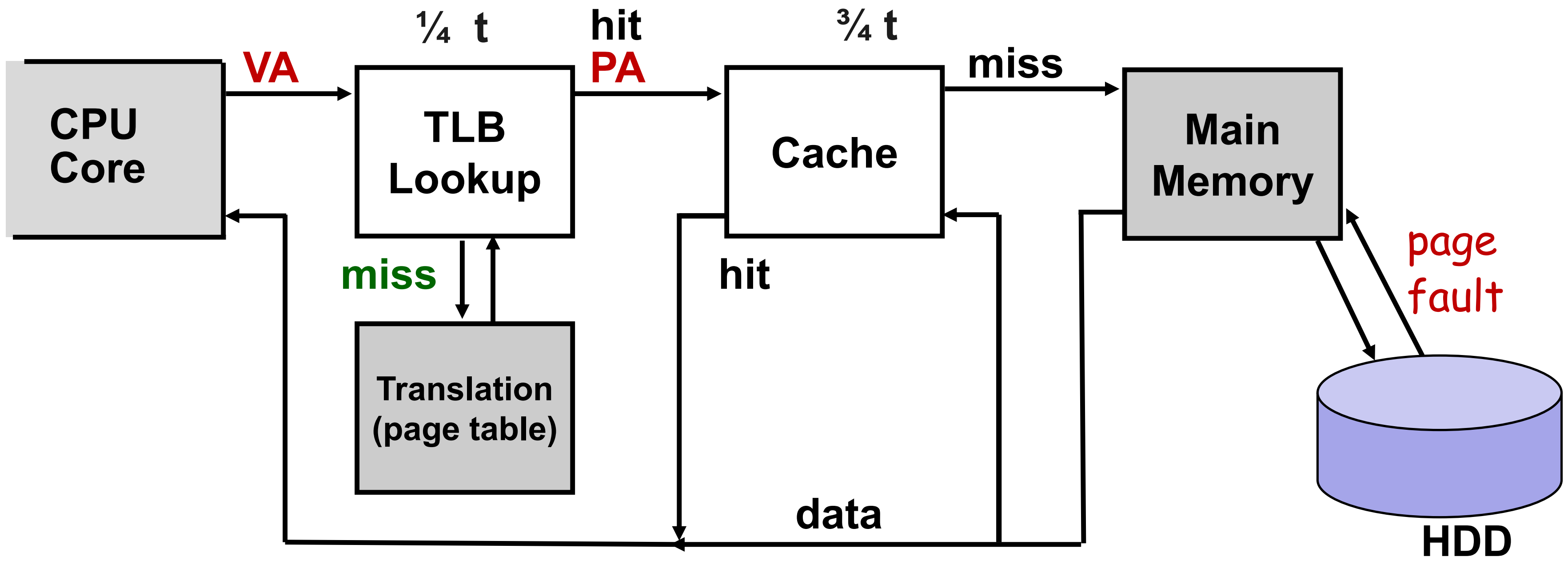
- TLB access time is typically smaller than cache access time (because TLBs are much smaller than caches)
  - TLBs are typically not more than 128 to 256 entries even on high end machines

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

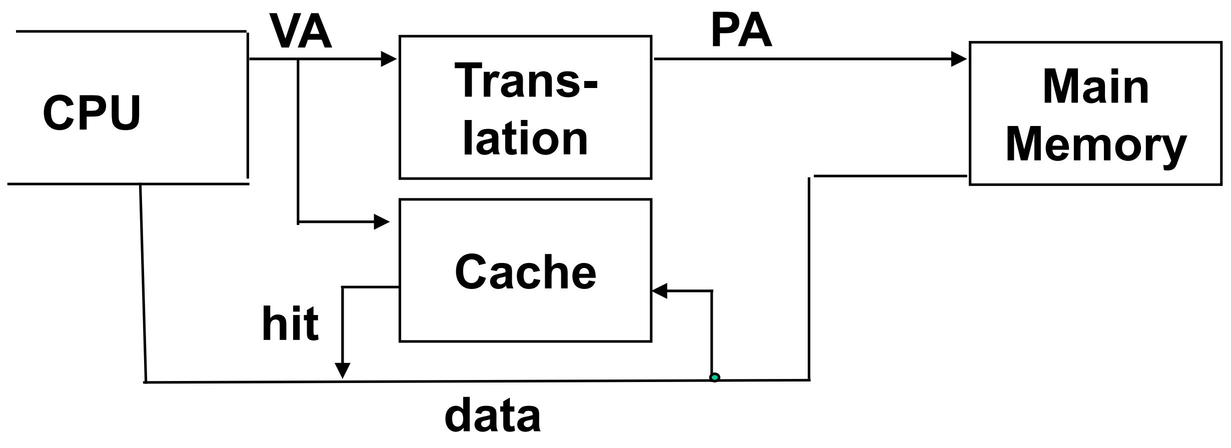
	Intel P4	**AMD Opteron**
TLB organization	1 TLB for instructions and 1TLB for data Both 4-way set associative Both use ~LRU replacement Both have 128 entries TLB misses handled in hardware	2 TLBs for instructions and 2 TLBs for data Both L1 TLBs fully associative with ~LRU replacement Both L2 TLBs are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

Why Not a **Virtually Addressed Cache?**

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



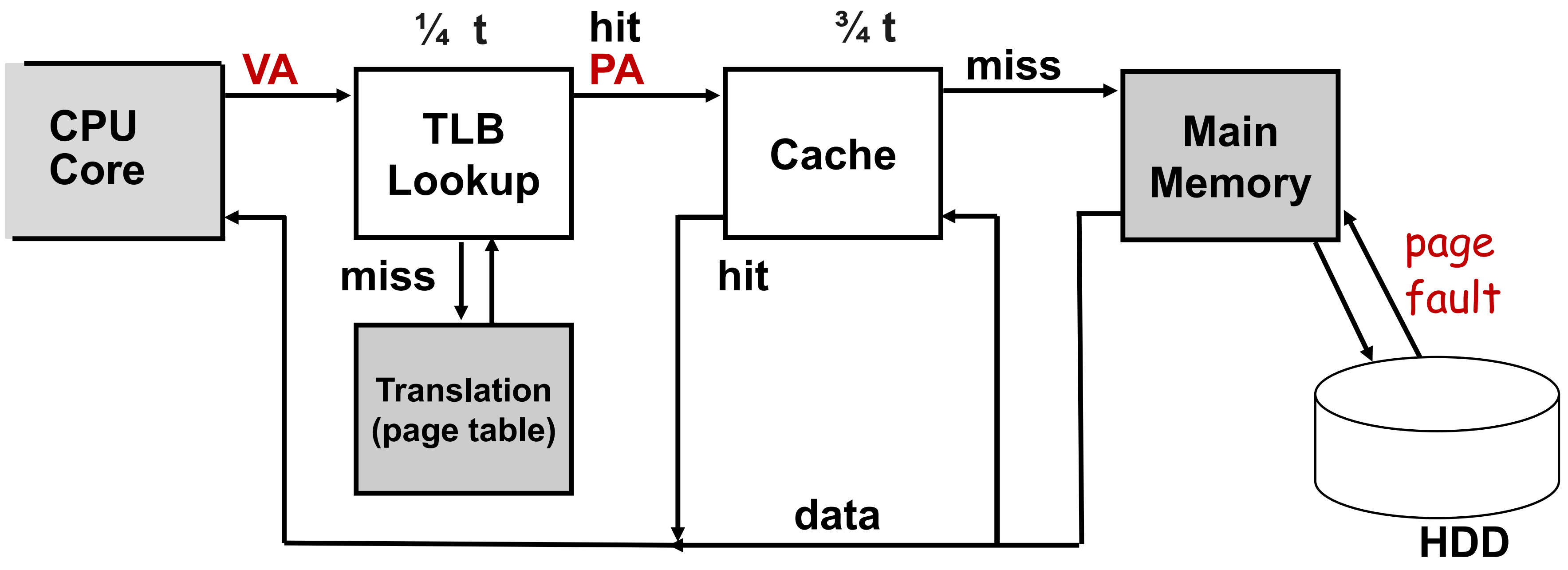
but

- 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

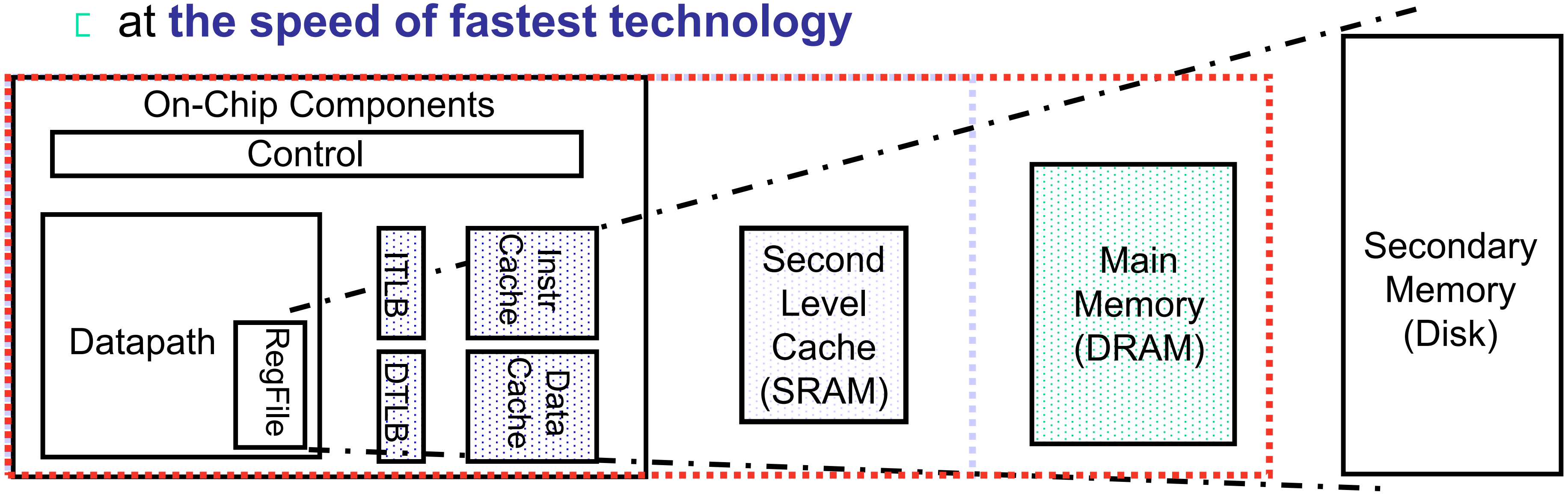
A TLB in the Memory Hierarchy



- **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 Typical Memory Hierarchy

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



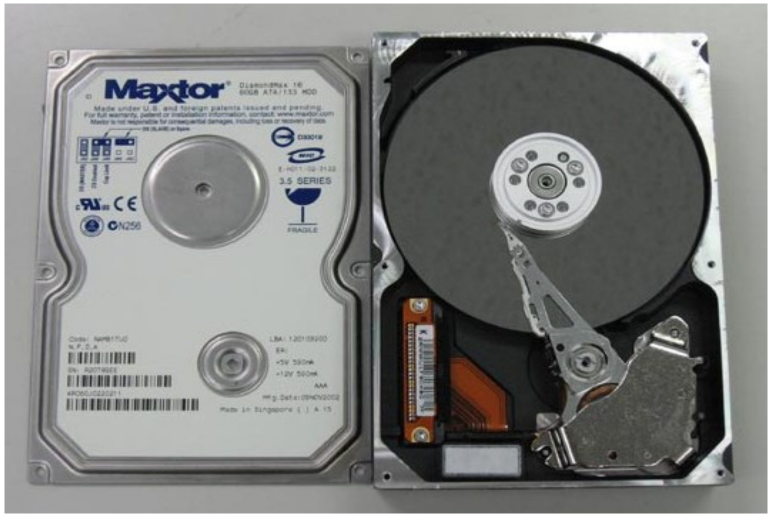
Speed (%cycles):	½'s	1's	10's	100's	1,000's
Size (bytes):	100's	K's	10K's	M's	G's to T's
Cost:	highest				lowest

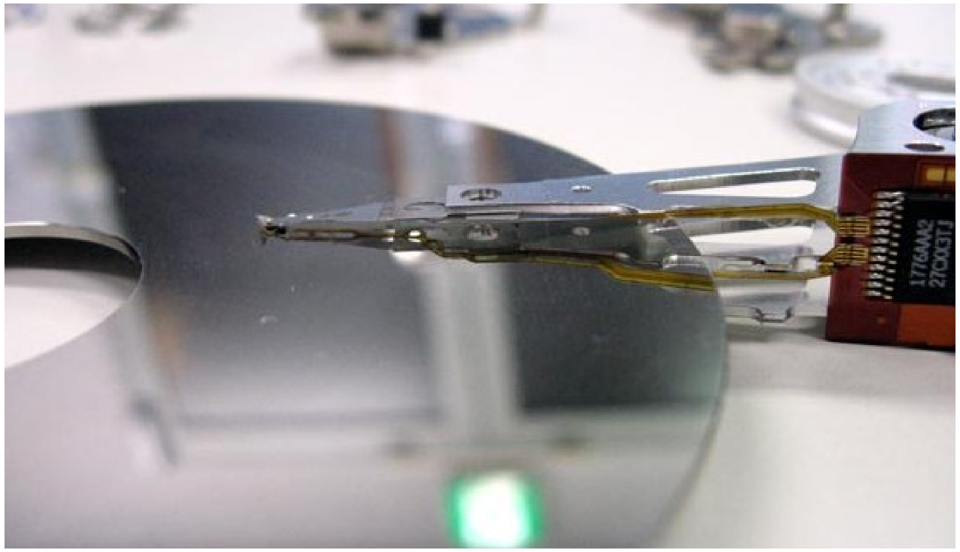
TLB: Translation Lookaside Buffer

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

Magnetic Disk（磁気ディスク）

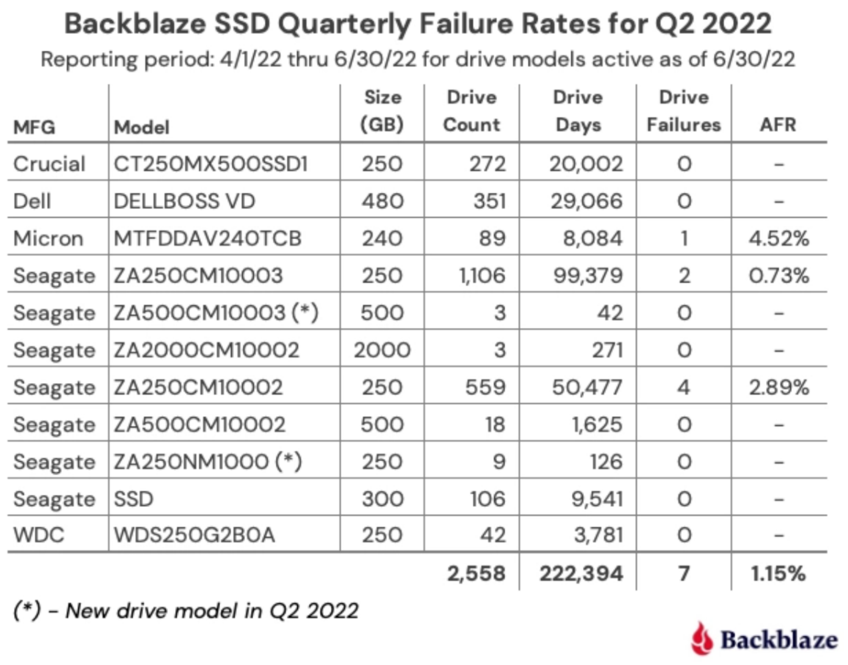




http://sougo057.aicomp.jp/0001.html

Q3 2022 Hard Drive Failure Rates

Annualized Failure Rate (AFR)



Backblaze SSD Quarterly Failure Rates for Q2 2022

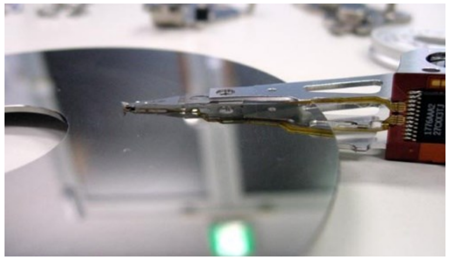
Reporting period: 4/1/22 thru 6/30/22 for drive models active as of 6/30/22

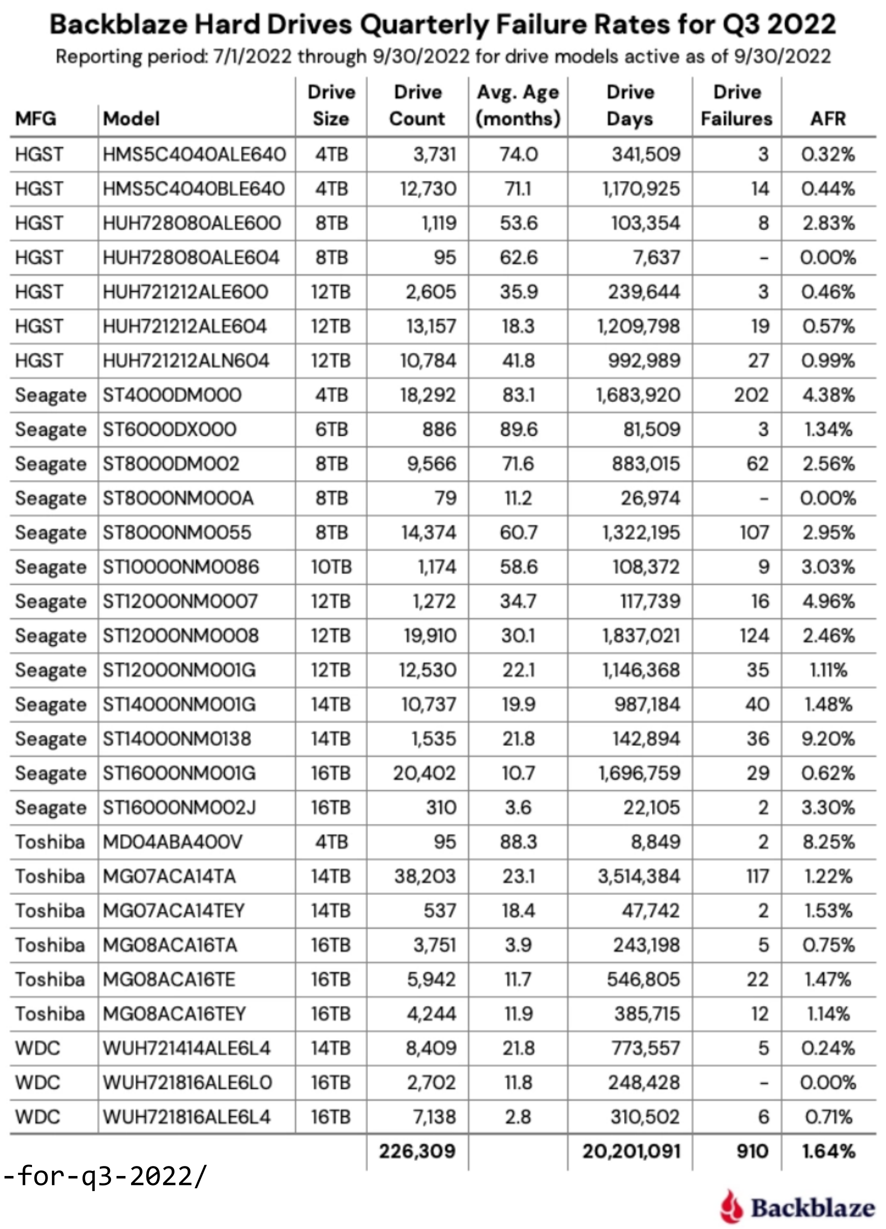
MFG	Model	Size (GB)	Drive Count	Drive Days	Drive Failures	AFR
Crucial	CT250MX500SSD1	250	272	20,002	0	-
Dell	DELLBOSS VD	480	351	29,066	0	-
Micron	MTFDDAV240TCB	240	89	8,084	1	4.52%
Seagate	ZA250CM10003	250	1,106	99,379	2	0.73%
Seagate	ZA500CM10003 (*)	500	3	42	0	-
Seagate	ZA2000CM10002	2000	3	271	0	-
Seagate	ZA250CM10002	250	559	50,477	4	2.89%
Seagate	ZA500CM10002	500	18	1,625	0	-
Seagate	ZA250NM1000 (*)	250	9	126	0	-
Seagate	SSD	300	106	9,541	0	-
WDC	WDS250G2B0A	250	42	3,781	0	-
			2,558	**222,394**	**7**	**1.15%**

(*) – New drive model in Q2 2022

Backblaze

https://www.backblaze.com/blog/ssd-drive-stats-mid-2022-review/





Backblaze Hard Drives Quarterly Failure Rates for Q3 2022

Reporting period: 7/1/2022 through 9/30/2022 for drive models active as of 9/30/2022

MFG	Model	Drive Size	Drive Count	Avg. Age (months)	Drive Days	Drive Failures	AFR
HGST	HMS5C4040ALE640	4TB	3,731	74.0	341,509	3	0.32%
HGST	HMS5C4040BLE640	4TB	12,730	71.1	1,170,925	14	0.44%
HGST	HUH728080ALE600	8TB	1,119	53.6	103,354	8	2.83%
HGST	HUH728080ALE604	8TB	95	62.6	7,637	-	0.00%
HGST	HUH721212ALE600	12TB	2,605	35.9	239,644	3	0.46%
HGST	HUH721212ALE604	12TB	13,157	18.3	1,209,798	19	0.57%
HGST	HUH721212ALN604	12TB	10,784	41.8	992,989	27	0.99%
Seagate	ST4000DM000	4TB	18,292	83.1	1,683,920	202	4.38%
Seagate	ST6000DX000	6TB	886	89.6	81,509	3	1.34%
Seagate	ST8000DM002	8TB	9,566	71.6	883,015	62	2.56%
Seagate	ST8000NM000A	8TB	79	11.2	26,974	-	0.00%
Seagate	ST8000NM0055	8TB	14,374	60.7	1,322,195	107	2.95%
Seagate	ST10000NM0086	10TB	1,174	58.6	108,372	9	3.03%
Seagate	ST12000NM0007	12TB	1,272	34.7	117,739	16	4.96%
Seagate	ST12000NM0008	12TB	19,910	30.1	1,837,021	124	2.46%
Seagate	ST12000NM001G	12TB	12,530	22.1	1,146,368	35	1.11%
Seagate	ST14000NM001G	14TB	10,737	19.9	987,184	40	1.48%
Seagate	ST14000NM0138	14TB	1,535	21.8	142,894	36	9.20%
Seagate	ST16000NM001G	16TB	20,402	10.7	1,696,759	29	0.62%
Seagate	ST16000NM002J	16TB	310	3.6	22,105	2	3.30%
Toshiba	MD04ABA400V	4TB	95	88.3	8,849	2	8.25%
Toshiba	MG07ACA14TA	14TB	38,203	23.1	3,514,384	117	1.22%
Toshiba	MG07ACA14TEY	14TB	537	18.4	47,742	2	1.53%
Toshiba	MG08ACA16TA	16TB	3,751	3.9	243,198	5	0.75%
Toshiba	MG08ACA16TE	16TB	5,942	11.7	546,805	22	1.47%
Toshiba	MG08ACA16TEY	16TB	4,244	11.9	385,715	12	1.14%
WDC	WUH721414ALE6L4	14TB	8,409	21.8	773,557	5	0.24%
WDC	WUH721816ALE6L0	16TB	2,702	11.8	248,428	-	0.00%
WDC	WUH721816ALE6L4	16TB	7,138	2.8	310,502	6	0.71%
			226,309		**20,201,091**	**910**	**1.64%**

Backblaze

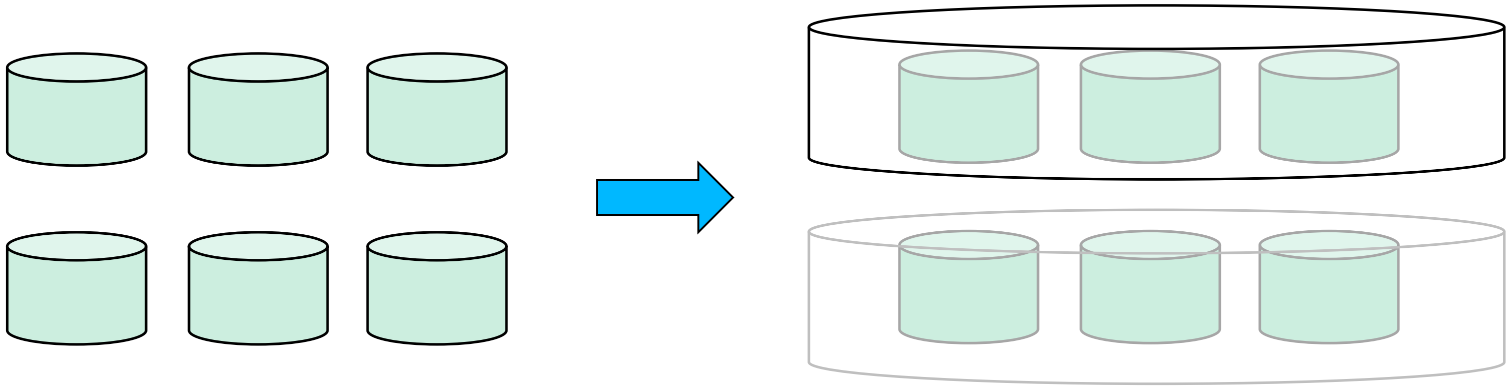
https://www.backblaze.com/blog/backblaze-drive-stats-for-q3-2022/

エラー，フォールト，故障

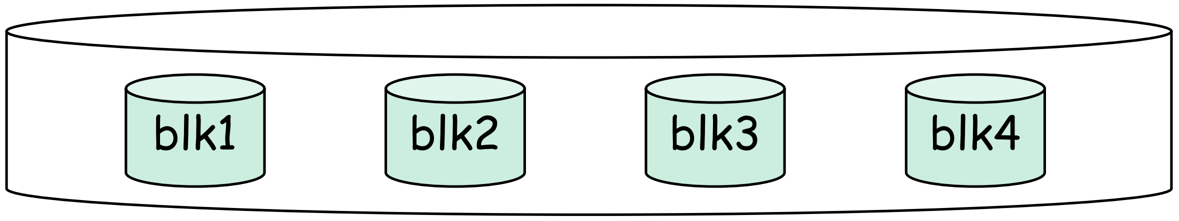
- Fault（フォールト，故障）
  - 誤りの原因
- Error（エラー，誤り）
  - システム内の構成要素の正しくない出力
- Failure（障害）
  - システムが正常な動作をしない．コンポーネントやシステムが，期待した機能，サービス，結果から逸脱すること．

RAID: 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

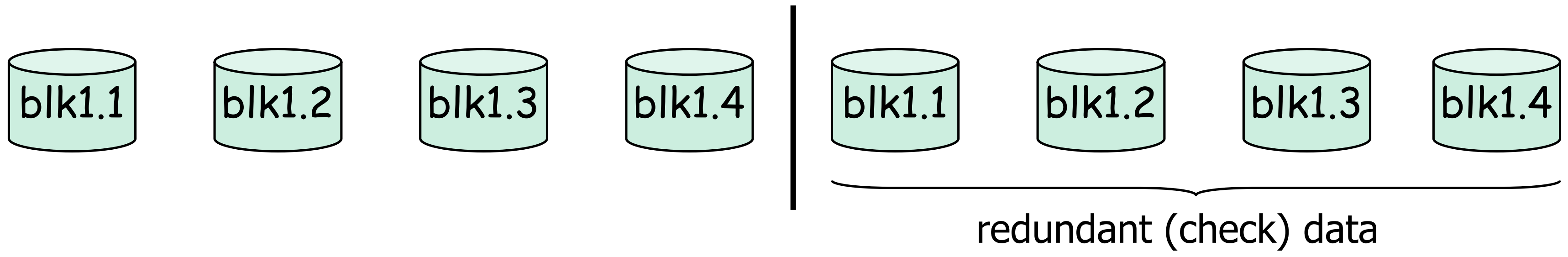


RAID: Level 0 (RAID 0, 冗長性なし, ストライピング)



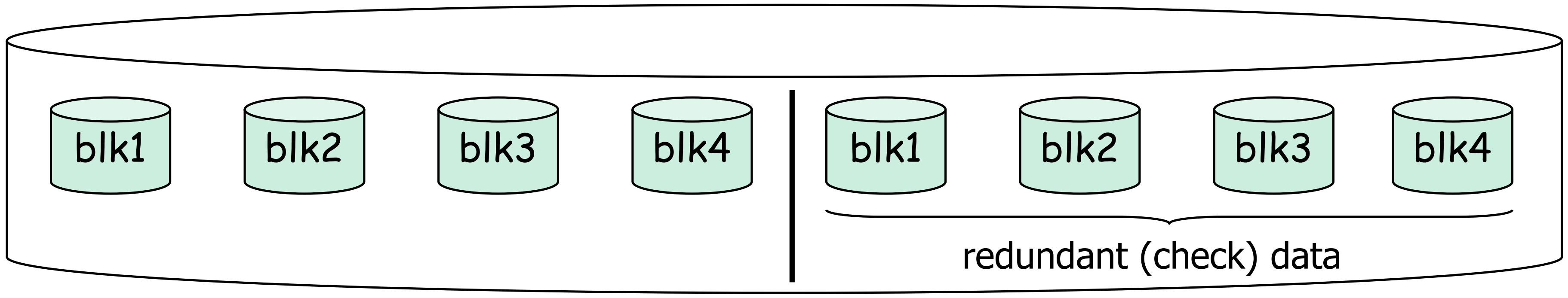
- Multiple smaller disks as opposed to **one big disk**
  - Spreading the blocks over multiple disks – **striping** – means that multiple blocks can be accessed in parallel increasing the performance
  - 4 disk system gives four times the throughput of a 1 disk system
  - Same cost as one *big* disk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

RAID: Level 1 (Redundancy via Mirroring)



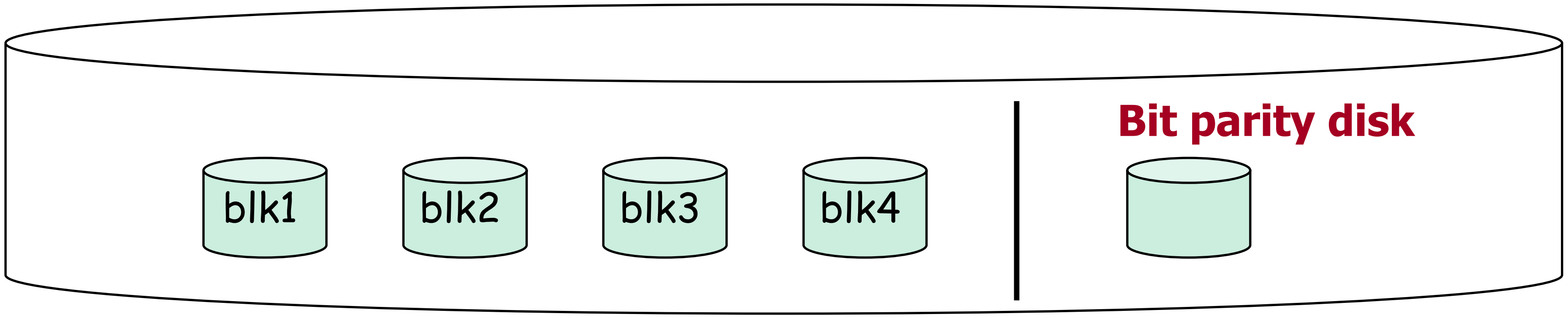
- Uses twice as many disks for redundancy so there are always two copies of the data
  - The number of redundant disks = the number of data disks so twice the cost of one big disk
    - writes have to be made to both sets of disks, so writes would be only 1/2 the performance of RAID 0
- What if one disk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

RAID: Level 0+1 (RAID01, 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

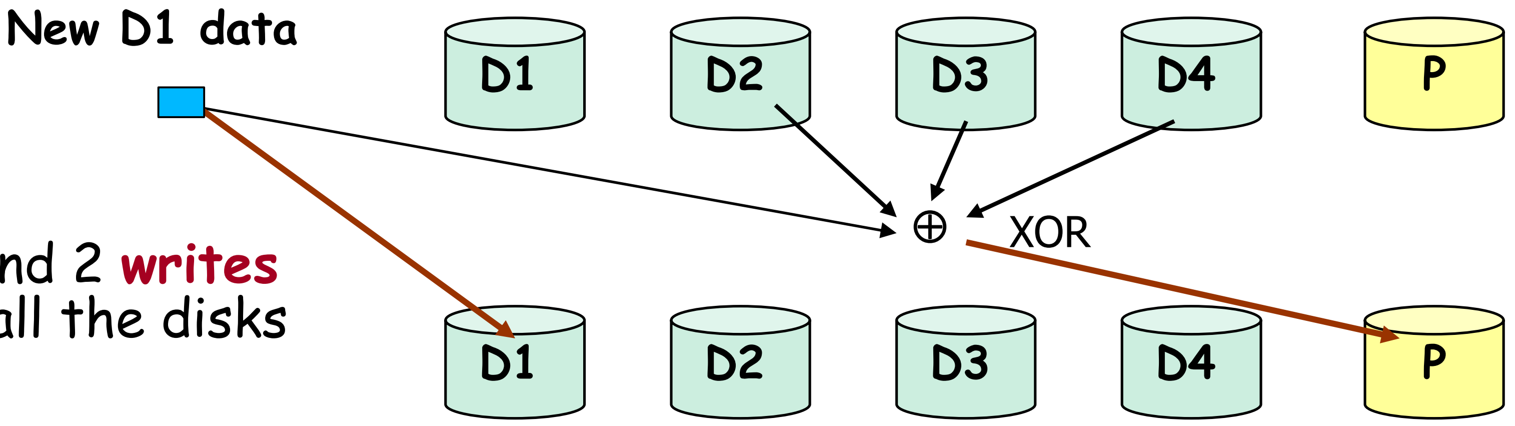
RAID: Level 3 (Bit/Byte-Interleaved **Parity**)



- Cost of higher availability is reduced to 1/N where N is the number of disks in a protection group
  - # redundant disks = 1 × # of protection groups
    - **writes** require writing the new data to the data disk as well as computing the parity, meaning reading the other disks, so that the parity disk can be updated
    - **reads** require reading all the operational data disks as well as the parity disk to calculate the missing data that was stored on the **failed disk**

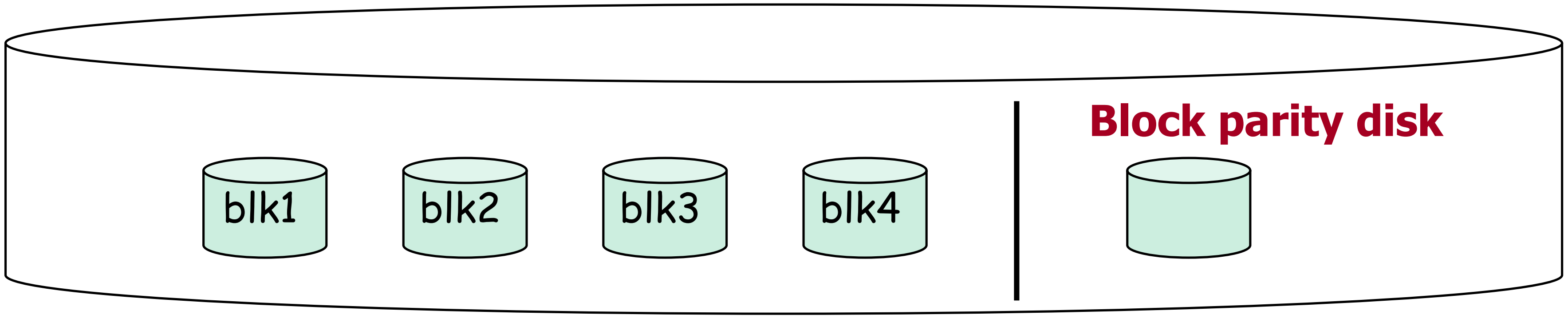
RAID 3 and parity

- RAID 3



3 **reads** and 2 **writes** involving all the disks

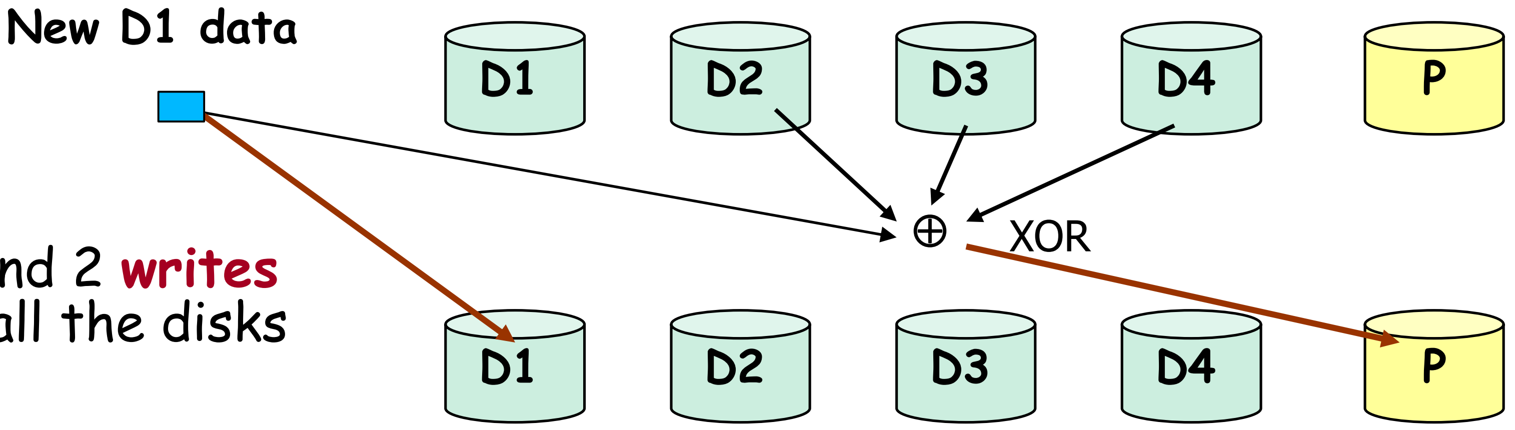
RAID: Level 4 (Block-Interleaved Parity)



- 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)

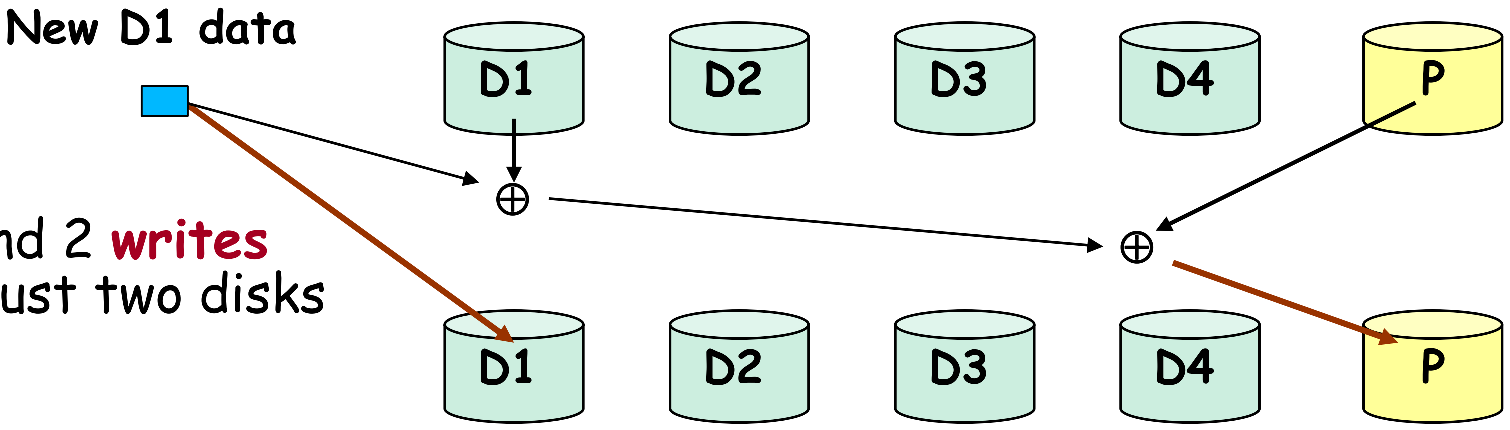
Small Reads and Small Writes

- RAID 3



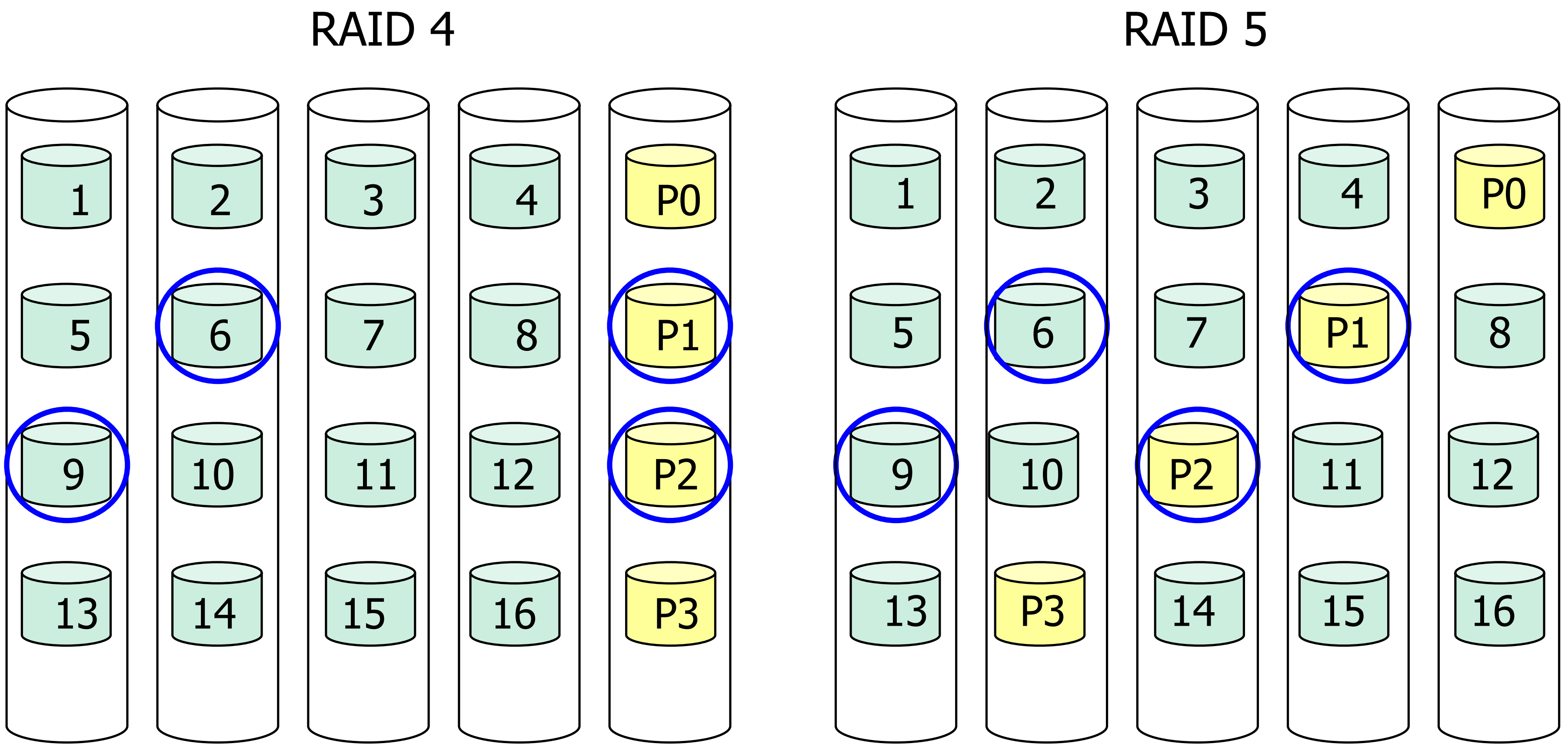
3 **reads** and 2 **writes** involving all the disks

- RAID 4 small reads and small writes



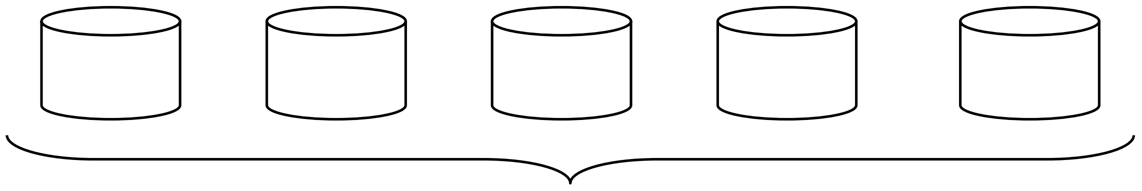
2 **reads** and 2 **writes** involving just two disks

Distributing Parity Blocks



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

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**

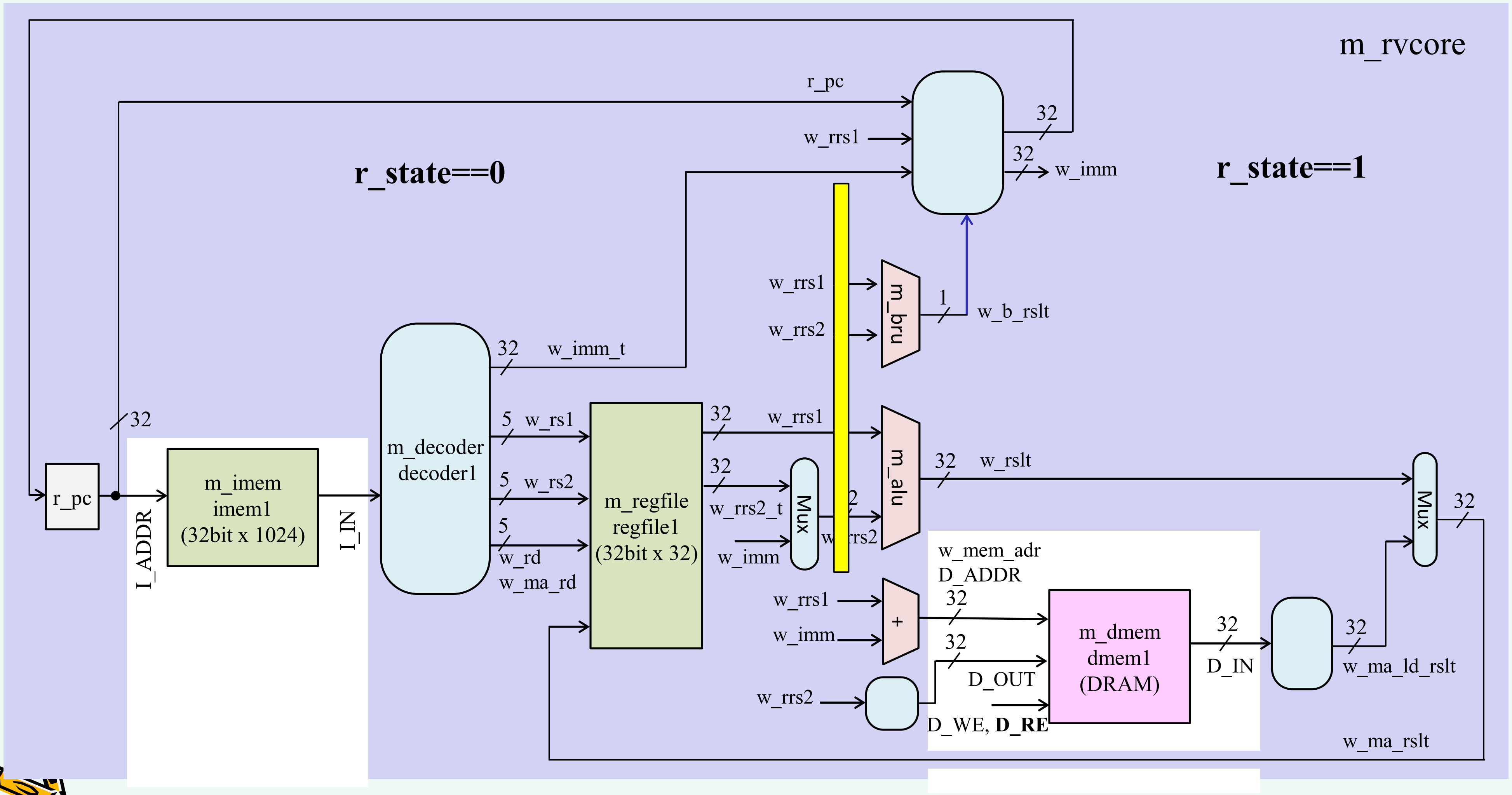
ACRi Room で利用できる幾つかのソフトウェア

- /tools/cad/bin/simrv
  - simrv -a -m main.bin
  - simrv –a –t 0 100 –m main.bin

- /tools/cad/riscv/rv32ima/bin/riscv32-unknown-linux-gnu-gcc
- /tools/cad/riscv/rv32ima/bin/riscv32-unknown-linux-gnu-ld
- /tools/cad/riscv/rv32ima/bin/riscv32-unknown-linux-gnu-objdump

module m_rvcore (RV32I, multi-cycle processor)

- 40MHz operating frequency
- lb, lbu, lh, lhu, sb, sh are not supported, DRAM is not initialized through UART



RISC-V開発のための Tool /SDK, GNU Toolchain

- 準備
  - Ubuntu 22.04 server
  - 次のコマンドで幾つかのパッケージをインストールする。
    - `apt -y install autoconf automake autotools-dev curl python3 libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev`

- 参考サイト
  - https://github.com/riscv-collab/riscv-gnu-toolchain

2022-09

RISC-V GNU Toolchain / RV32IMA

- ビルドとインストール
  - Ubuntu 22.04 Server (64-bit)
  - 一般ユーザの権限で構築
  - Intel Corei9-12900KF を搭載するサーバ計算機で、構築に「12分」程度
    - make の -j オプションで利用するコア数を指定する。

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain
$ cd riscv-gnu-toolchain
$ ./configure --prefix=/tools/cad/riscv/rv32ima --with-arch=rv32ima --with-abi=ilp32
$ time make –j 4
```

インストールするフォルダ、命令セット、ABIを指定する。
命令セットには、基本の rv32i と乗除算 m とアトミック命令の a 指定する。