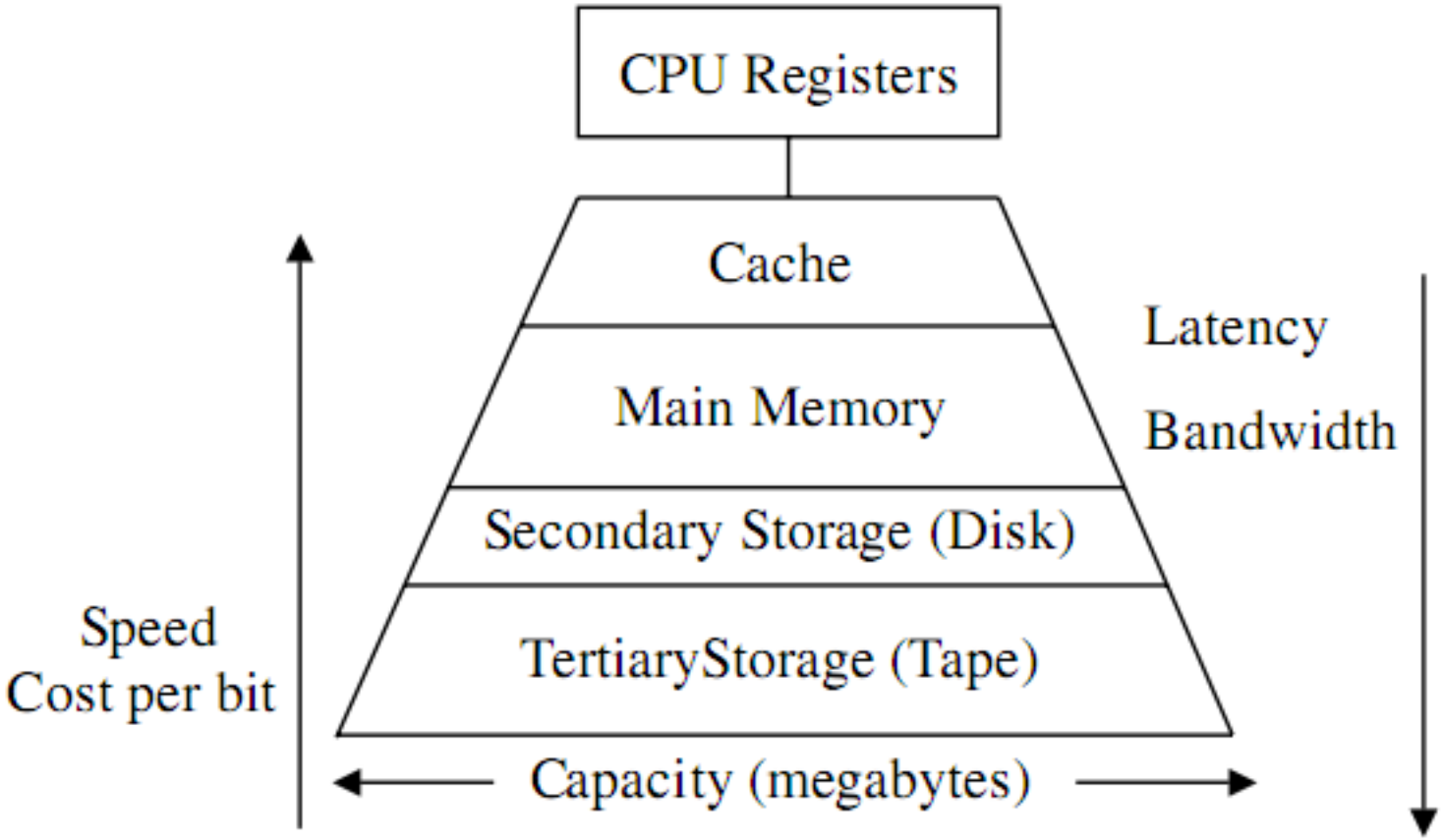
Chương 3: Xử lý xen kẽ dòng mã lệnh và bộ nhớ Cache (phần 2)

Bộ Nhớ Cache

Bộ nhớ cache

- Mô hình phân cấp bộ nhớ
- Cache là gì?
- Vai trò của cache
- Các nguyên lý cơ bản hoạt động của cache
- Kiến trúc cache
- Tổ chức cache
- Đọc/ ghi trong cache
- Các chính sách thay thế
- Cách thức để nâng cao hiệu năng cache

1. Mô hình phân cấp hệ thống bộ nhớ



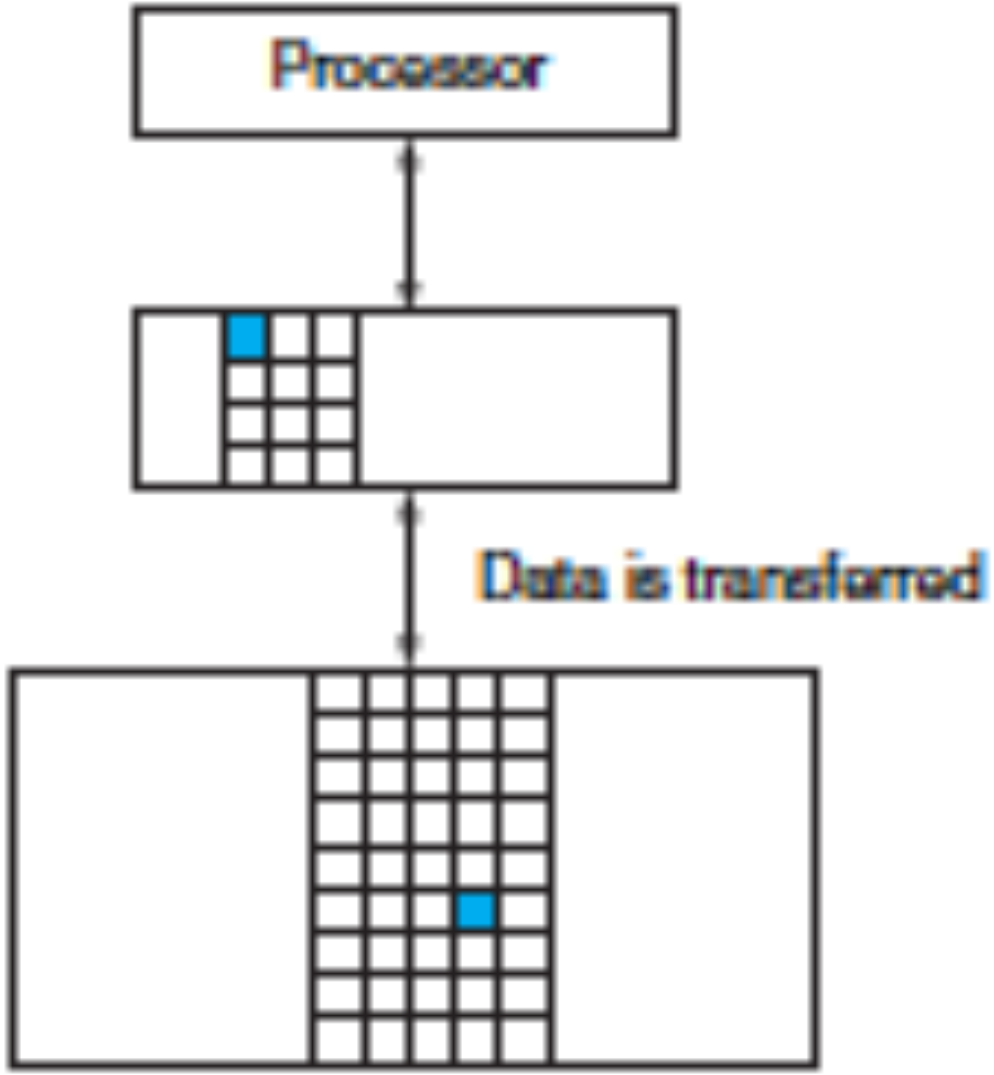
Các tham số phân cấp bộ nhớ

	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64–1024 bytes	1–10 ns	System clock rate	High
Cache memory	Random	8–512 KB	15–20 ns	10–20 MB/s	$500
Main memory	Random	16–512 MB	30–50 ns	1–2 MB/s	$20–50
Disk memory	Direct	1–20 GB	10–30 ms	1–2 MB/s	$0.25
Tape memory	Sequential	1–20 TB	30–10,000 ms	1–2 MB/s	$0.025

Mô hình phân cấp bộ nhớ

- Mô hình phân cấp có nhiều mức lưu trữ, mỗi mức có tốc độ truy cập khác nhau.
- Mức gần với bộ VXL có tốc độ truy cập cao nhất và chi phí đắt nhất, và các mức càng xa tốc độ truy cập càng giảm, và chi phí rẻ.
- Mức gần bộ VXL thường là một phần của bộ nhớ các mức bộ nhớ xa hơn. Tất cả dữ liệu máy tính (phần mềm chạy hệ thống, dữ liệu cá nhân, hệ thống) đều lưu ở phần xa nhất (ổ cứng: HDD, SSD).
- Dữ liệu thường được lập trình copy giữa 2 mức liên tiếp.

Mô hình phân cấp bộ nhớ (tiếp)



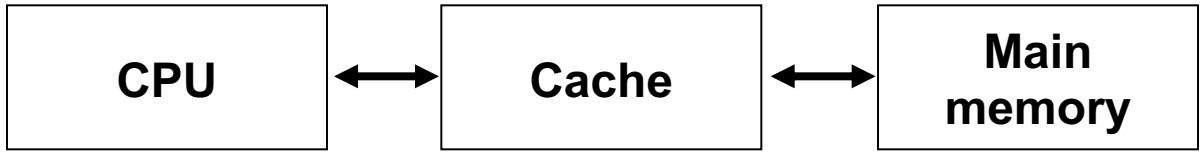
Vai trò của mô hình phân cấp

- Nâng cao hiệu năng hệ thống:
  - Dung hòa được CPU có tốc độ cao với bộ nhớ chính và bộ nhớ phụ có tốc độ thấp
  - Thời gian truy cập dữ liệu trung bình của CPU từ hệ thống bộ nhớ gần bằng thời gian truy cập cache
- Giảm giá thành sản xuất:
  - Các thành phần đắt tiền sẽ được sử dụng với dung lượng nhỏ hơn
  - Các thành phần rẻ hơn được sử dụng với dung lượng lớn hơn

  => Tổng giá thành của hệ thống nhớ theo mô hình phân cấp sẽ rẻ hơn so với hệ thống nhớ không phân cấp cùng tốc độ

Cache là gì?

- Cache là thành phần nhớ trong sơ đồ phân cấp bộ nhớ máy tính.
  - Nó hoạt động như thành phần trung gian, trung chuyển dữ liệu từ bộ nhớ chính về CPU và ngược lại
- Vị trí của cache:
  - Với các hệ thống cũ, cache thường nằm ngoài CPU
  - Với các CPU mới, cache thường được tích hợp vào trong CPU



Cache là gì?

- Dung lượng cache thường nhỏ:
  - Với các hệ thống cũ: 16K, 32K,..., 128K
  - Với các hệ thống mới: 256K, 512K, 1MB, 2MB, …
- Tốc độ truy nhập của cache nhanh hơn so với tốc độ bộ nhớ chính
- Giá thành cache (tính trên bit) thường đắt hơn so với bộ nhớ chính
- Với các hệ thống CPU mới, cache thường được chia thành nhiều mức:
  - Mức L1: 16 – 32 KB có tốc độ rất cao
  - Mức L2: 1 -16MB có tốc độ khá cao

Vai trò của Cache

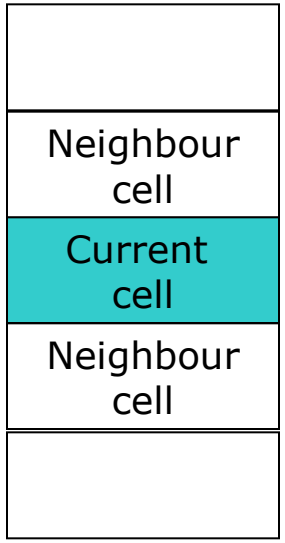
- Nâng cao hiệu năng hệ thống:
  - Dung hòa giữa CPU có tốc độ cao và bộ nhớ chính tốc độ thấp (giảm số lượng truy cập trực tiếp của CPU vào bộ nhớ chính)
  - Thời gian trung bình CPU truy cập hệ thống bộ nhớ gần bằng thời gian truy cập cache
- Giảm giá thành sản xuất:
  - Nếu 2 hệ thống có cùng hiệu năng thì hệ thống có cache sẽ rẻ hơn
  - Nếu 2 hệ thống cùng giá thành, hệ thống có cache sẽ nhanh hơn

Các nguyên lý hoạt động của Cache

- Cache được coi là bộ nhớ thông minh:
  - Cache có khả năng đoán trước yêu cầu về lệnh và dữ liệu của CPU
  - Dữ liệu và lệnh cần thiết được chuyển trước từ bộ nhớ chính về cache -> CPU chỉ truy nhập cache -> giảm thời gian truy nhập bộ nhớ
- Cache hoạt động dựa trên 2 nguyên lý cơ bản:
  - Nguyên lý cục bộ/ lân cận về không gian (spatial locality)
  - Nguyên lý cục bộ/ lân cận về thời gian (temporal locality)

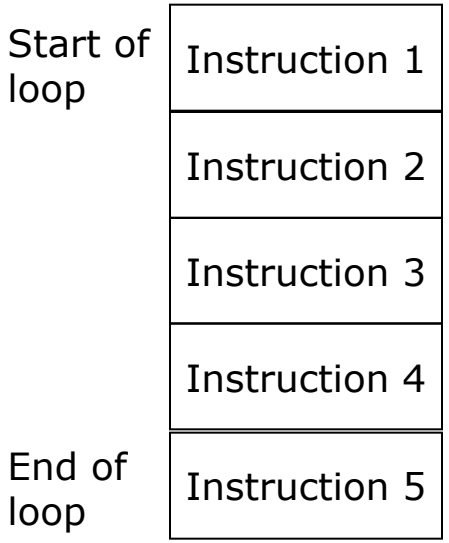
Các nguyên lý hoạt động của Cache

- Cục bộ (lân cận) về không gian:
  - Nếu một vị trí bộ nhớ được truy cập, thì khả năng/ xác suất các vị trí gần đó được truy cập trong thời gian gần tới là cao
  - Áp dụng với các mục dữ liệu và các lệnh có thứ tự tuần tự theo chương trình
  - Hầu hết các lệnh trong chương trình có thứ tự tuần tự, do đó cache đọc một khối lệnh trong bộ nhớ, mà bao gồm cả các phần tử xung quanh vị trí phần tử hiện tại được truy cập



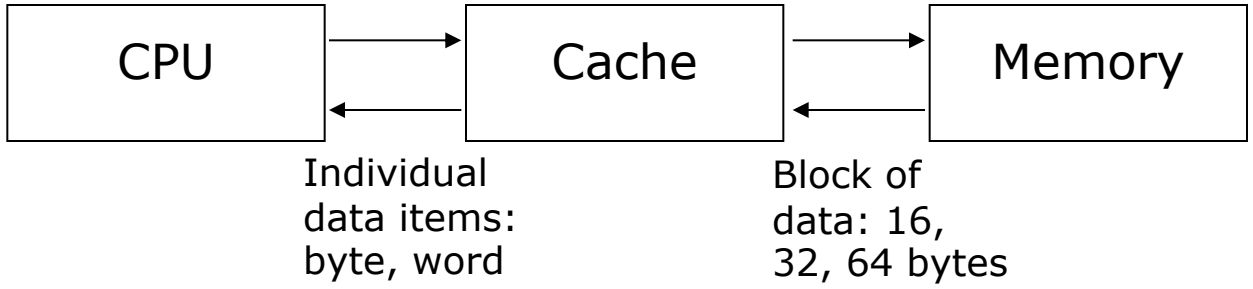
Các nguyên lý cơ bản của Cache

- Cục bộ (lân cận) về thời gian:
  - Nếu một vị trí bộ nhớ được truy cập, thì khả năng nó sẽ được truy cập **lại** trong thời gian gần tới là cao
  - Áp dụng với các mục dữ liệu và các lệnh trong vòng lặp
  - Cache đọc khối dữ liệu trong bộ nhớ bao gồm tất cả các thành phần trong vòng lặp



Trao đổi dữ liệu

- CPU đọc/ ghi từng mục dữ liệu riêng biệt từ/ vào cache
- Cache đọc/ ghi khối dữ liệu từ/ vào bộ nhớ



Hệ số Hit và Miss của Cache

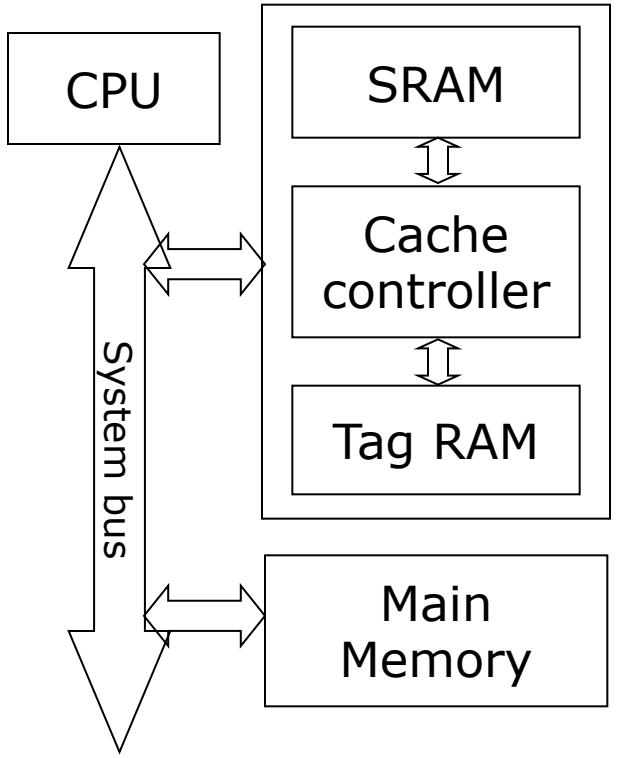
- Hit là sự kiện CPU truy cập tới mục dữ liệu/ mục tin mà tìm được trong cache
  - Xác suất xảy ra Hit được gọi là hệ số hit H
  - 0<= H <= 1
  - H càng cao thì cache càng tốt
- Miss là sự kiện CPU truy cập tới mục dữ liệu mà không tìm thấy nó trong cache
  - Khả năng xảy ra Miss gọi là hệ số miss hay 1-H
  - 0<= (1-H) <= 1
  - Hệ số miss thấp thì hiệu quả cache cao

Kiến trúc Cache – look aside

- Cache và bộ nhớ cùng được kết nối tới bus hệ thống
- Cache và bộ nhớ chính "thấy" chu kỳ bus CPU tại cùng một thời điểm
- Ưu:
  - Thiết kế đơn giản
  - Miss nhanh
- Nhược:
  - Hit chậm

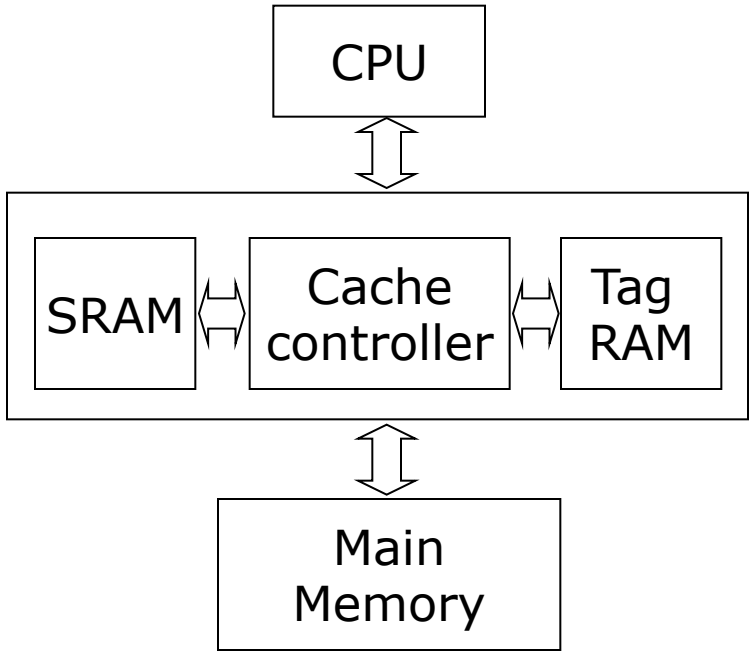
SRAM: RAM lưu dữ liệu cache

Tag RAM: RAM lưu địa chỉ bộ nhớ



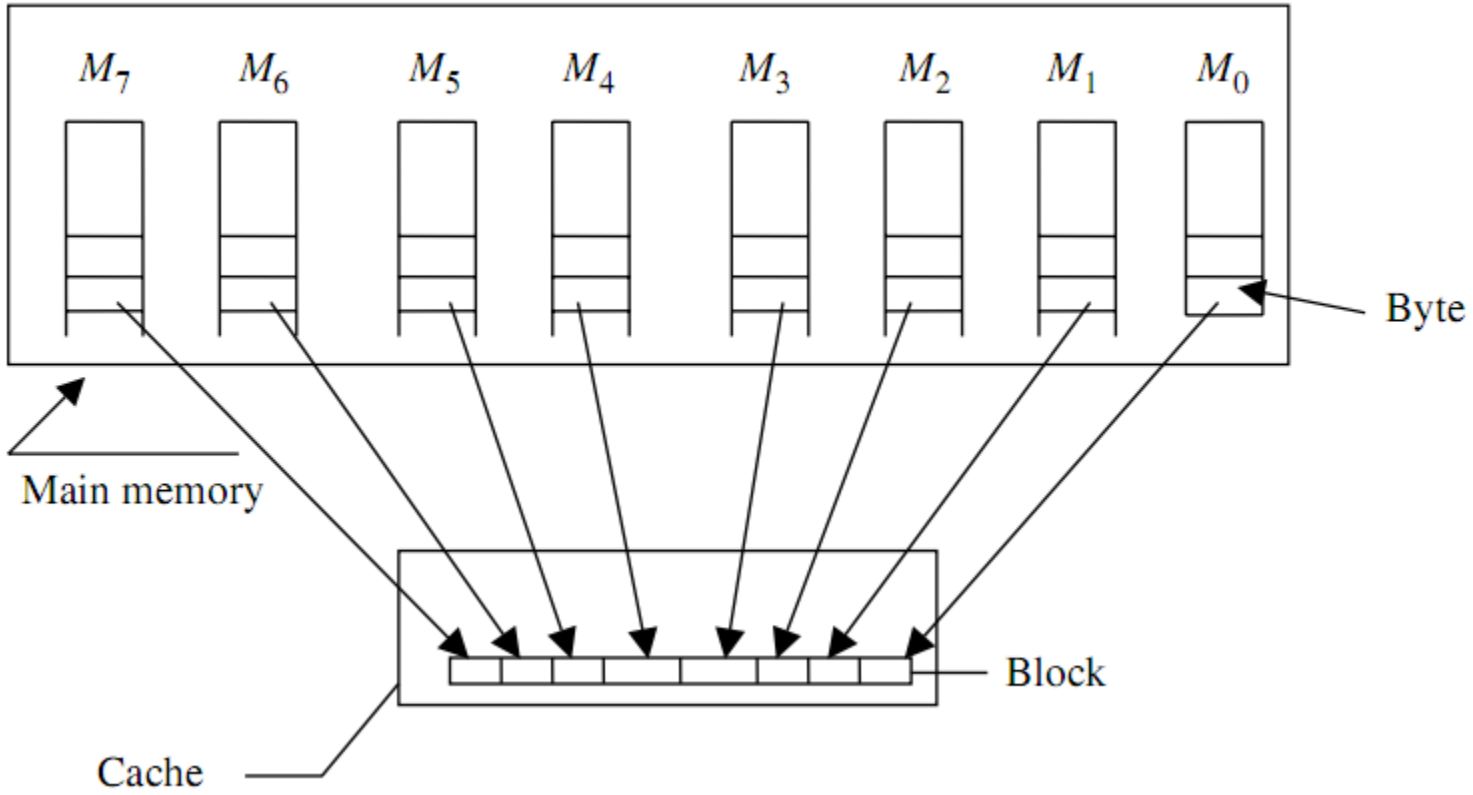
Kiến trúc Cache – look through

- Cache nằm giữa CPU và bộ nhớ chính
- Cache "thấy" chu kỳ bus CPU trước sau đó nó "truyền" lại cho bộ nhớ chính
- Ưu:
  - Hit nhanh
- Nhược:
  - Thiết kế phức tạp
  - Đắt
  - Miss chậm



Tổ chức Cache

- Tổ chức cache giải quyết vấn đề cache và bộ nhớ chính phối hợp làm việc với nhau như thế nào



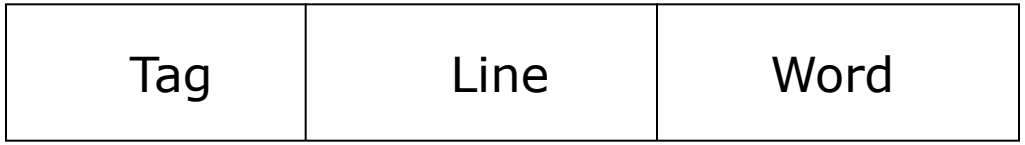
Các kỹ thuật tổ chức Cache

- Ánh xạ trực tiếp (direct mapping):
  - Đơn giản, nhanh
  - Ánh xạ cố định
- Ánh xạ kết hợp đầy đủ (fully associative mapping):
  - Phức tạp, chậm
  - Ánh xạ linh hoạt
- Ánh xạ tập kết hợp/ theo bộ (set): (set associative mapping)
  - Phức tạp
  - Nhanh, ánh xạ linh hoạt

Ánh xạ trực tiếp

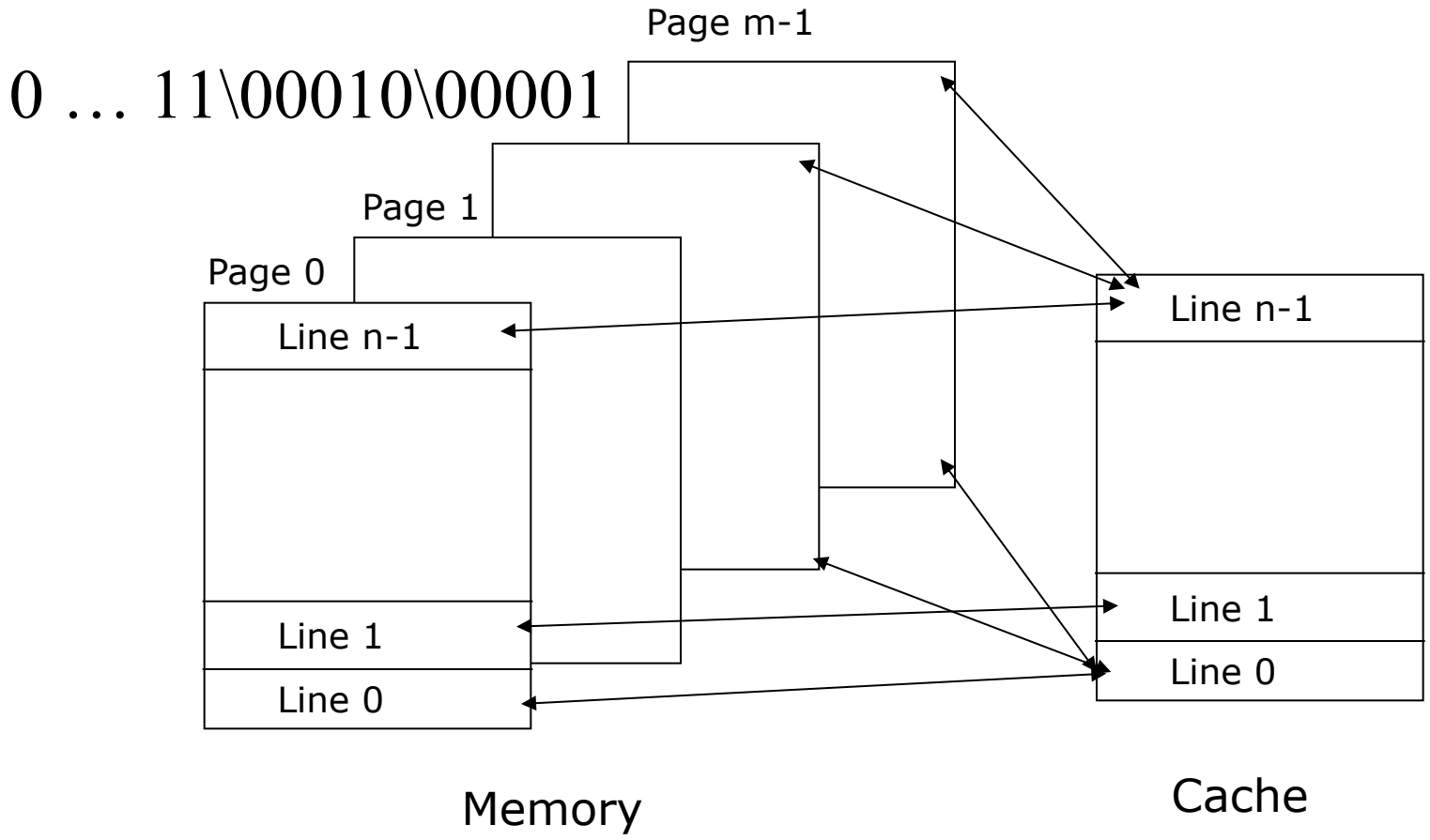
- Cache: Được chia thành n khối hoặc dòng (block or line), từ $Line_0$ tới $Line_{n-1}$
- Bộ nhớ:
  - Được chia thành m trang (page), từ $page_0$ tới $page_{m-1}$
  - Mỗi trang bộ nhớ có kích thước bằng cache
  - Mỗi trang có n lines, từ $Line_0$ tới $Line_{n-1}$
- Ánh xạ:
  - $Line_0$ của ($page_0$ tới $page_{m-1}$) được ánh xạ tới $Line_0$ của cache
  - $Line_1$ của ($page_0$ tới $page_{m-1}$) được ánh xạ tới $Line_1$ của cache
  - ….
  - $Line_{n-1}$ của ($page_0$ tới $page_{m-1}$) được ánh xạ tới $Line_{n-1}$ của cache

Địa chỉ ánh xạ trực tiếp

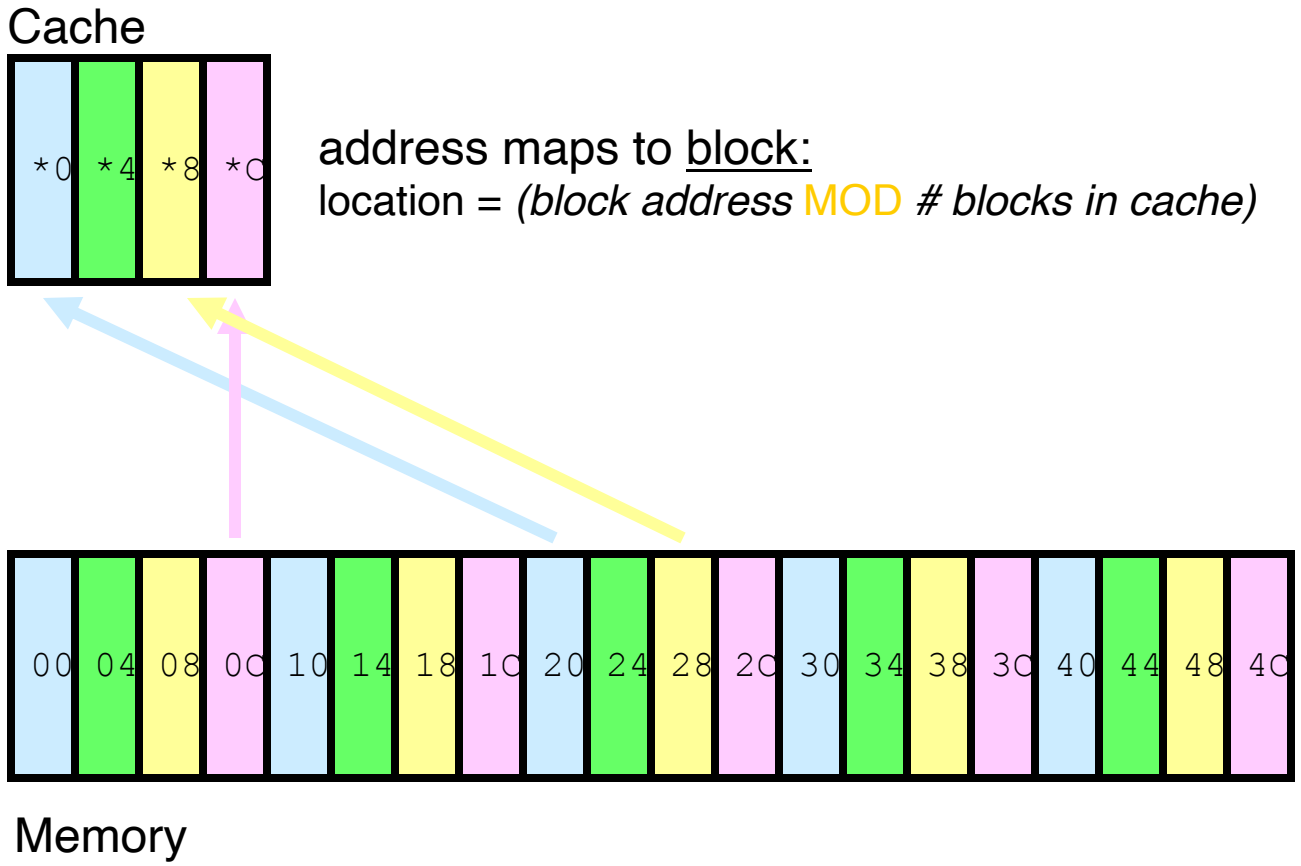


- *Tag* (bit): là địa chỉ của trang trong bộ nhớ
- *Line* (bit): là địa chỉ của line trong cache
- *Word* (bit): là địa chỉ của word trong line

Ánh xạ trực tiếp



Ví dụ: Direct Mapped block



Địa chỉ ánh xạ trực tiếp

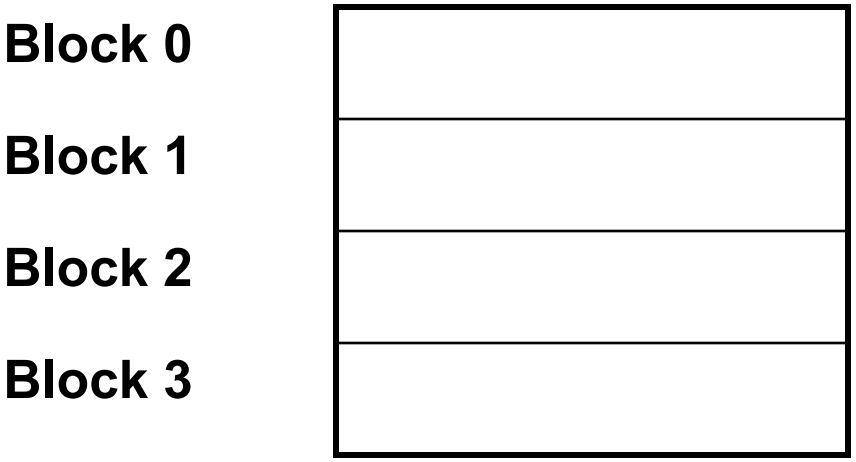
- Ví dụ: Tìm kích thước trường địa chỉ (32 bit) dùng direct mapping
  - Đầu vào:
    - Kích thước bộ nhớ: 4GB
    - Kích thước cache: 1KB
    - Kích thước line: 8 word (32 byte)
  - Đầu ra:
    - Kích thước line: 32 byte = 2^{3+2} -> Word = 5 bit (3 + 2)
    - Kích thước cache: 1 KB = 2^{10} -> có $2^{10} / 2^5 = 2^5$ lines -> Line = 5 bit
    - Tag = địa chỉ 32 bit – Line – Word = 32 – 5 – (3+2) = 22 bit

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 1: Mapping: 0 modulo 4 = 0

Mem Block	DM Hit/Miss
0	



Accessing A Direct-Mapped Cache

DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 1: Mapping: 0 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss

Block 0
Block 1
Block 2
Block 3

Mem[0]

Set 0 is empty: write Mem[0]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 2: Mapping: 8 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	

Block 0
Block 1
Block 2
Block 3

Mem[0]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 2: Mapping: 8 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss

Block 0
Block 1
Block 2
Block 3

Mem[8]

Set 0 contains Mem[0]. Overwrite with Mem[8]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 3: Mapping: 0 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	

	Mem[8]
Block 0	
Block 1	
Block 2	
Block 3	

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 3: Mapping: 0 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss

Block 0	Mem[0]
Block 1	
Block 2	
Block 3	

Set 0 contains Mem[8]. Overwrite with Mem[0]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 4: Mapping: 6 mod 4 = 2

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss
6	

Block 0
Block 1
Block 2
Block 3

Mem[0]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 4: Mapping: 6 mod 4 = 2

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss
6	miss

Block 0	Mem[0]
Block 1	
Block 2	Mem[6]
Block 3	

Set 2 empty. Write Mem[6]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 5: Mapping: 8 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss
6	miss
8	

Block 0	Mem[0]
Block 1	
Block 2	Mem[6]
Block 3	

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 5: Mapping: 8 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss
6	miss
8	miss

Block 0	Mem[8]
Block 1	
Block 2	Mem[6]
Block 3	

Set 0 contains Mem[0]. Overwrite with Mem[8]

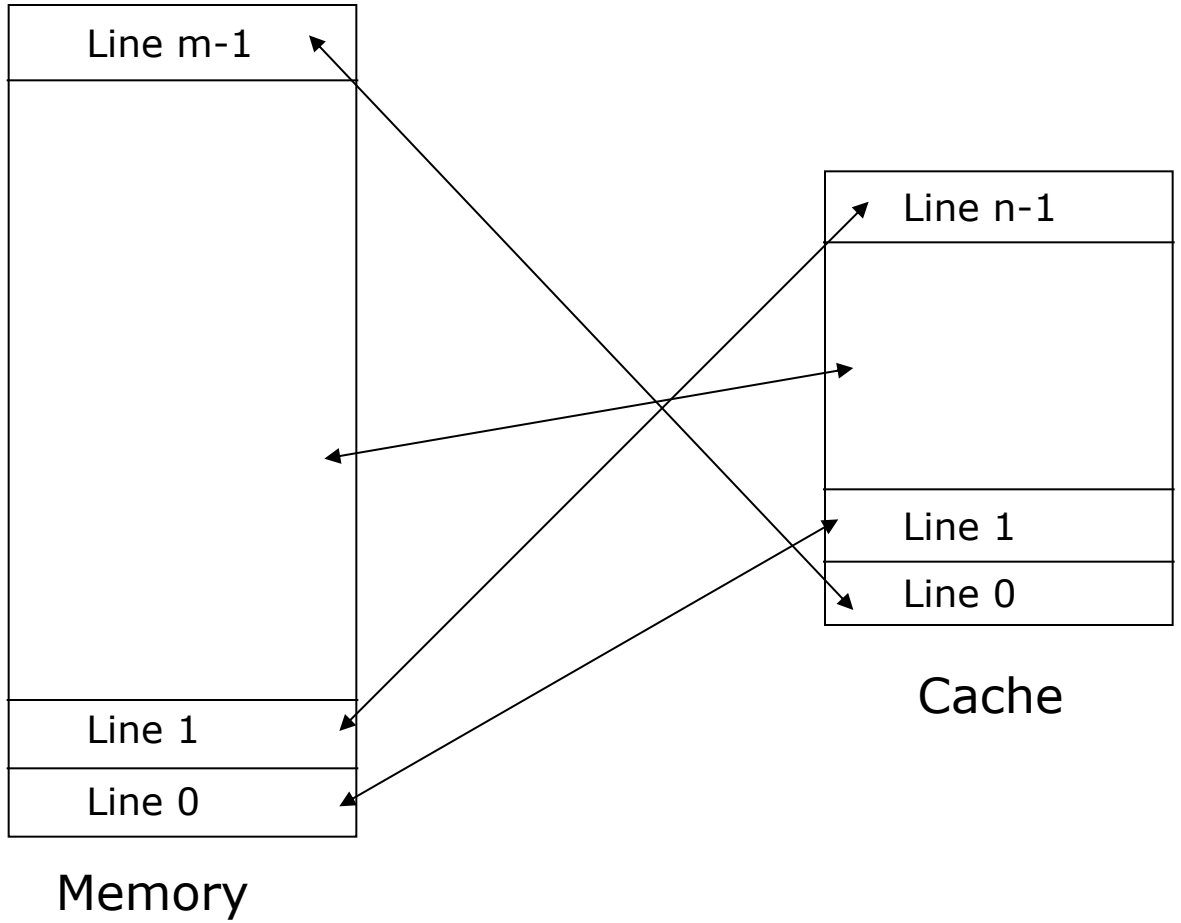
Ánh xạ trực tiếp

- Ưu điểm:
  - Thiết kế đơn giản
  - Nhanh vì ánh xạ cố định: khi biết địa chỉ bộ nhớ có thể tìm nó trong cache rất nhanh
- Nhược điểm:
  - Vì ánh xạ cố định nên khả năng xảy ra xung đột cao
  - Tỷ lệ hit thấp

Ánh xạ kết hợp (liên kết) đầy đủ

- Cache: Được chia thành n khối hoặc dòng (block or line), từ $Line_0$ tới $Line_{n-1}$
- Bộ nhớ:
  - Được chia thành m khối hay dòng, từ $Line_0$ tới $Line_{m-1}$
  - Kích thước mỗi dòng cache bằng kích thước một dòng bộ nhớ
  - Số lượng dòng trong bộ nhớ có thể lớn hơn nhiều số lương dòng của cache (m >>n)
- Ánh xạ:
  - Một dòng của bộ nhớ có thể ánh xạ tới dòng bất kì của cache
  - $Line_i$ của bộ nhớ có thể ánh xạ tới $line_j$ của cache

Ánh xạ kết hợp đầy đủ



- 000…..01\00011

Địa chỉ ánh xạ kết hợp đầy đủ

- *Tag* (bit) là địa chỉ của line trong bộ nhớ (page =1)
- *Word* (bit) là địa chỉ của từ trong line

Địa chỉ ánh xạ kết hợp đầy đủ

- Ví dụ: Tìm kích thước các trường địa chỉ (32 bit)
  - Input:
    - Kích thước cache = 1KB
    - Kích thước line = 32 byte
  - Output:
    - Kích thước Line = 32 byte = 2^5 -> Word = 5 bit
    - Tag = 32 bit – word = 32 – 5 = 27 bit

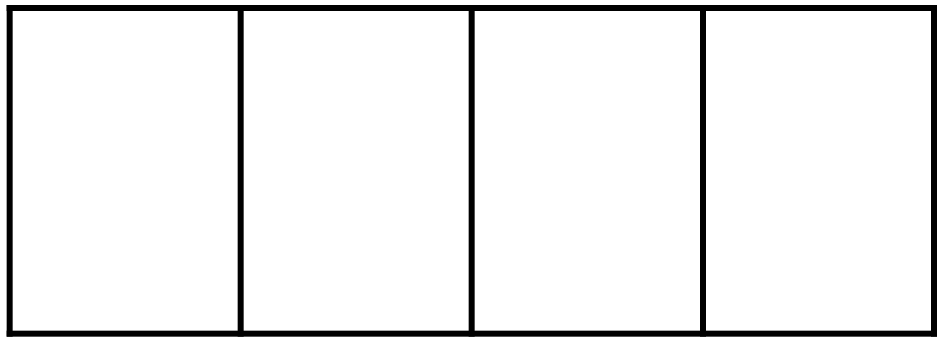
Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 1:

Mem Block	DM Hit/Miss
0	

Set 0



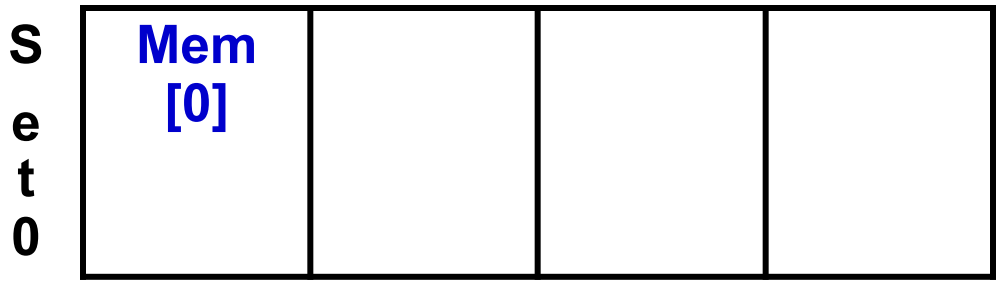
FA Block Replacement Rule: replace least recently used block in set

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 1:

Mem Block	DM Hit/Miss
0	miss



Set 0 is empty: write Mem[0] to Block 0

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 2:

Mem Block	DM Hit/Miss
0	miss
8	

Set 0				
Mem [0]				

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 2:

Mem Block	DM Hit/Miss
0	miss
8	miss

Set 0				
	Mem [0]	Mem [8]		

Blocks 1-3 are LRU: write Mem[8] to Block 1

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 3:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	

Set 0	Mem [0]	Mem [8]		

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 3:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit

Set 0				
	Mem [0]	Mem [8]		

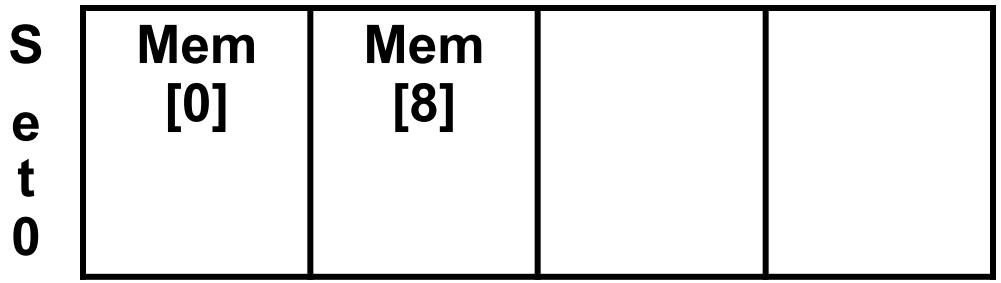
Block 0 contains Mem[0]

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 4:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	



Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 4:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss

Set 0				
	Mem [0]	Mem [8]	Mem [6]	

Blocks 2-3 are LRU : write Mem[6] to Block 2

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 5:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss
8	

Set 0	Mem [0]	Mem [8]	Mem [6]	

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 5:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss
8	hit

Set 0				
	Mem [0]	Mem [8]	Mem [6]	

Block 1 contains Mem[8]

Ánh xạ kết hợp đầy đủ

- Ưu:
  - Ít xung đột vì ánh xạ linh hoạt
  - Tỉ lệ hit cao hơn
- Nhược:
  - Chậm vì phải tìm kiếm địa chỉ bộ nhớ trong cache
  - Phức tạp vì có thêm n bộ so sánh địa chỉ trong cache
- Thường sử dụng cho cache có kích thước nhỏ

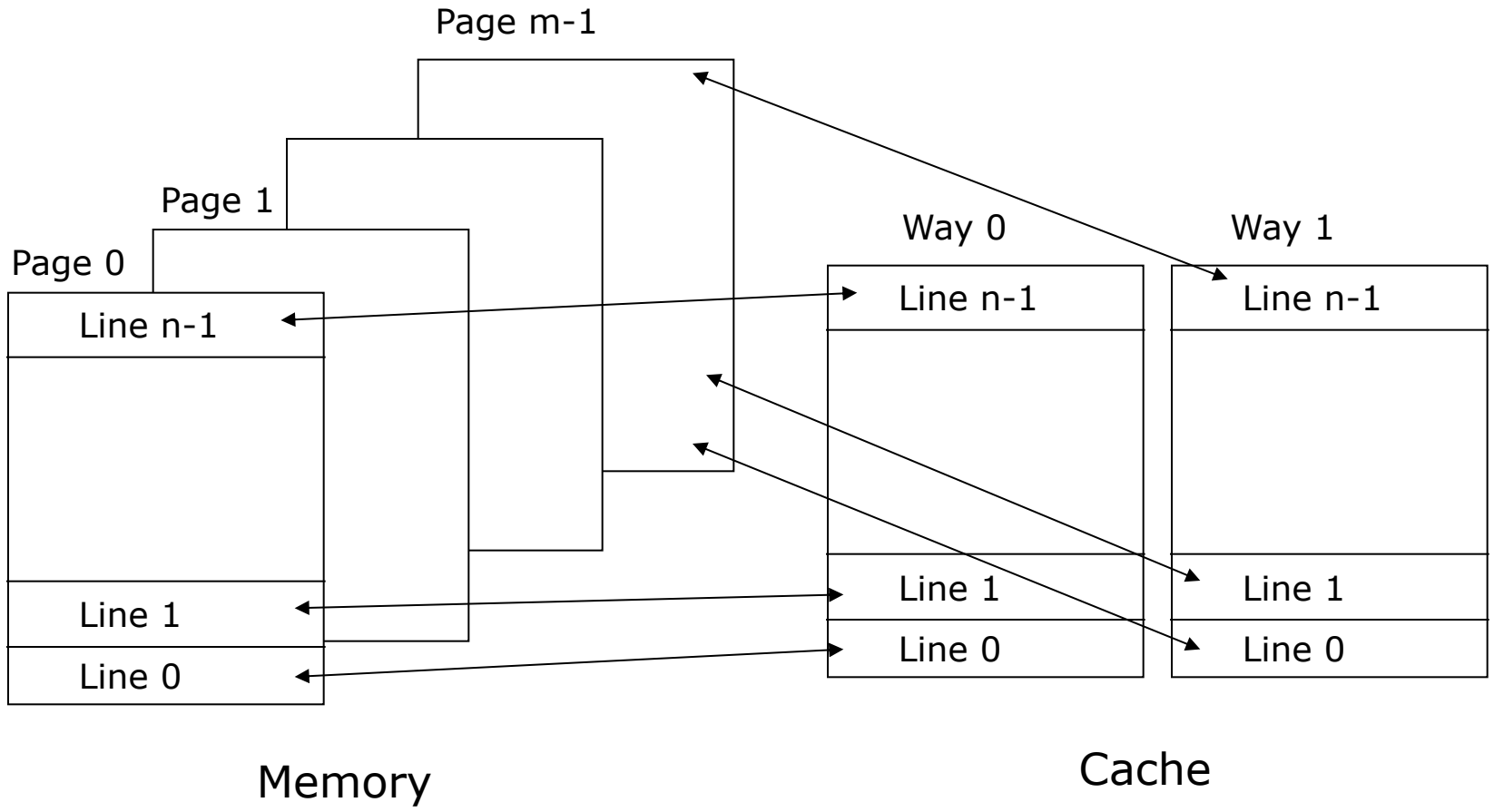
Ánh xạ tập kết hợp (liên kết theo nhóm)

- Cache:
  - Được chia thành k đường (ways) có kích thước bằng nhau
  - Mỗi đường được chia thành n dòng (block or line), từ $Line_0$ tới $Line_{n-1}$
- Bộ nhớ:
  - Được chia thành m trang, từ $page_0$ tới $page_{m-1}$
  - Kích thước trang page bằng kích thước way của cache
  - Mỗi trang có n line, từ $Line_0$ tới $Line_{n-1}$

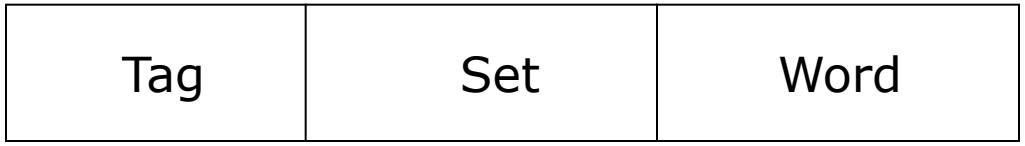
Ánh xạ tập kết hợp

- Ánh xạ:
  - Page được ánh xạ tới Way (ánh xạ linh hoạt):
    - Một page bộ nhớ có thể được ánh xạ tới way bất kì của cache
  - Line của page ánh xạ tới line của way (ánh xạ cố định):
    - $Line_0$ của $page_i$ được ánh xạ tới $Line_0$ của way_j;
    - $Line_1$ của $page_i$ được ánh xạ tới $Line_1$ của way_j;
    - …
    - $Line_{n-1}$ của $page_i$ được ánh xạ tới $Line_{n-1}$ của way_j;

Ánh xạ tập kết hợp



Địa chỉ ánh xạ tập kết hợp



- *Tag* (bit): là địa chỉ của trang trong bộ nhớ
- *Set* (bit): là địa chỉ của line trong way của cache
- *Word* (bit): là địa chỉ của word trong line

Địa chỉ ánh xạ tập kết hợp

- Ví dụ: Tìm kích thước các trường địa chỉ (32 bit)
  - Input:
    - Kích thước bộ nhớ: 4GB
    - Kích thước cache: 1KB, 2 ways
    - Kích thước Line: 32 byte
  - Output:
    - Line size =32 byte = 2^5 → Word = 5 bit
    - Cache size = 1KB = 2^{10} →có 2^{10} / 2^5 / 2 = 2^4 lines trong 1 way → Set = 4 bit
    - Tag = 32bit địa chỉ – Set – Word = 32 – 4 – 5 = 23 bit.

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 1: Mapping: 0 mod 2 = 0

Mem Block	DM Hit/Miss
0	

Set 0		
Set 1		

SA Block Replacement Rule: replace least recently used block in set

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 1: Mapping: 0 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss

Set 0	Mem[0]	
Set 1		

Set 0 is empty: write Mem[0] to Block 0

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 2: Mapping: 8 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	

Set 0	Mem[0]	
Set 1		

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 2: Mapping: 8 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss

	Mem[0]	Mem[8]
Set 0		
Set 1		

Set 0, Block 1 is LRU: write Mem[8]

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 3: Mapping: 0 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	

	Mem[0]	Mem[8]
Set 0		
Set 1		

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 3: Mapping: 0 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit

Set 0	Mem[0]	Mem[8]
Set 1		

Set 0, Block 0 contains Mem[0]

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 4: Mapping: 6 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	

	Mem[0]	Mem[8]
Set 0		
Set 1		

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 4: Mapping: 6 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss

	Mem[0]	Mem[6]
Set 0		
Set 1		

Set 0, Block 1 is LRU: overwrite with Mem[6]

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 5: Mapping: 8 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss
8	

	Mem[0]	Mem[6]
Set 0		
Set 1		

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 5: Mapping: 8 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss
8	miss

	Mem[8]	Mem[6]
Set 0		
Set 1		

Set 0, Block 0 is LRU: overwrite with Mem[8]

Ánh xạ tập kết hợp

- Ưu:
  - Nhanh vì ánh xạ trực tiếp được sử dụng cho ánh xạ dòng
  - Ít xung đột vì ánh xạ từ trang nhớ tới đường của cache là linh hoạt
  - Tỷ lệ tìm thấy (hit) cao
- Nhược:
  - Thiết kế và điều khiển phức tạp vì cache được chia thành các way

Bài tập

- Cho bộ nhớ DM cache có 64 blocks và kích thước block là 16 byte. Tính thứ tự của block trong cache mà được địa chỉ 1200 ánh xạ tới?

Giải

- Mỗi block chứa 16 bytes. Vậy địa chỉ 1200 ở block thứ

1200/16 = 75

- Thứ tự block trong bộ nhớ cach = thứ tự block trong main memory mod (#cache block) = 75 mod 64 = 11
- Địa chỉ 1200 được ánh xạ tới block 11 trong bộ nhớ cache.

Bài tập

- Tính số lượng bit trong bộ nhớ cache ánh xạ trực tiếp cho phép chứa 16KB dữ liệu, và mỗi dòng nhớ chứa 4 từ dữ liệu (32 bit)?

Bài tập

- Tính số lượng bit trong bộ nhớ cache ánh xạ trực tiếp cho phép chứa 16KB dữ liệu, và mỗi dòng nhớ chứa 4 từ dữ liệu (bộ VXL 32 bit)?
- Giải:
- VXL 32 bit = 4 byte, 1 word = 2^2 byte → 16KB = 4K (2^{12}) words

1 block = 4 words = 2^2 words =2^4 bytes

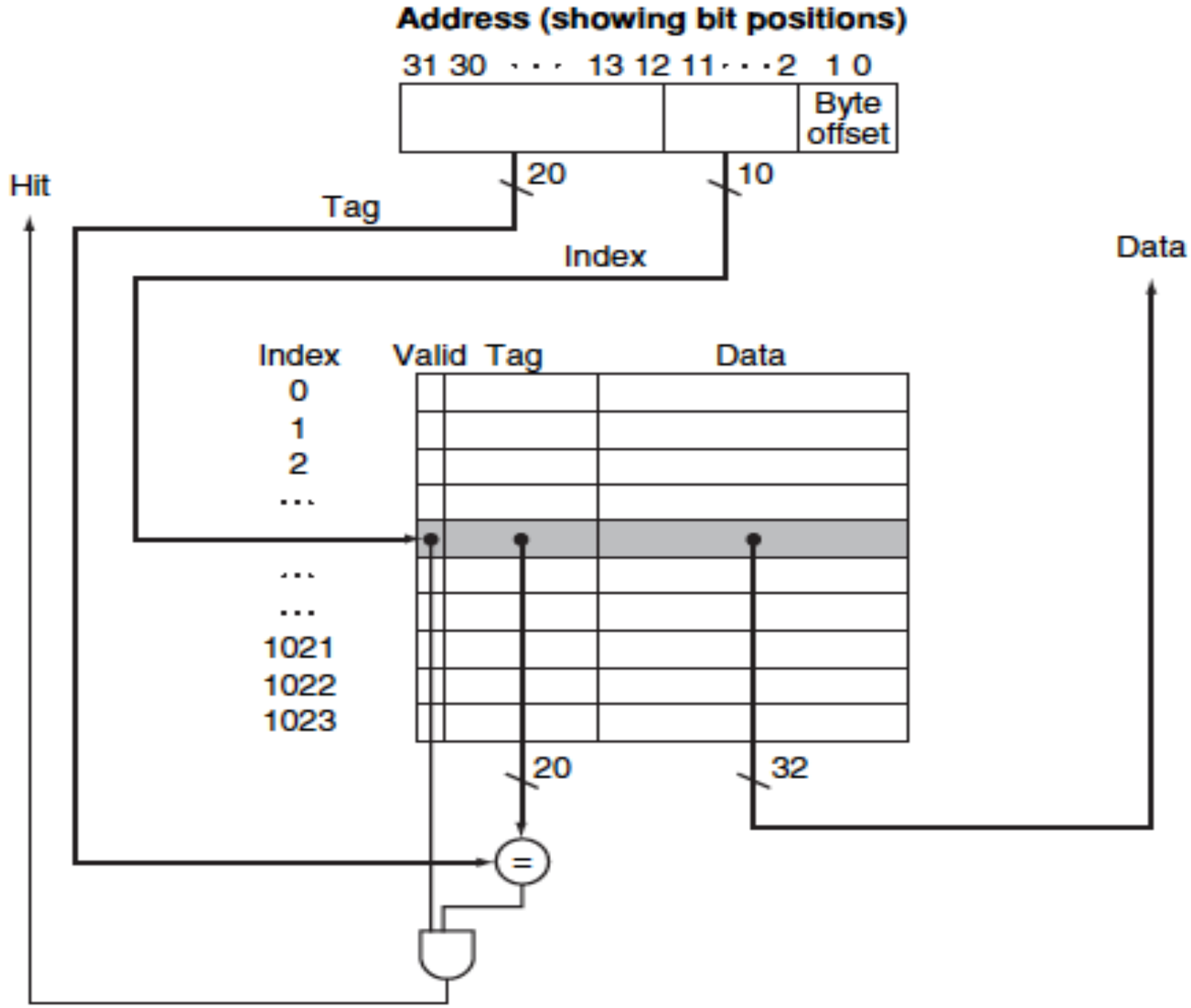
→ số lượng blocks = $2^{12}/2^2 = 2^{10}$ blocks

Trường tag địa chỉ: 32 – 10 – 4 = 18

1 blocks chứa: data (4x32 bit) + 18 tag + 1 (valid bit)

- Total: 2^{10}x(4x32+18+1)=147 Kbit = 18.4KB lưu 16KB dữ liệu (1.15 lần)

Ví dụ


Address (showing bit positions)
31 30 · · · 13 12 11 · · · 2 1 0
Byte offset
20
10
Tag
Index
Hit
Data
Index
Valid
Tag
Data
0
1
2
1021
1022
1023
20
32
=

Đọc/ ghi thông tin trong cache

- Thao tác đọc:
  - Trường hợp tìm thấy (hit case): mục dữ liệu yêu cầu tìm thấy trong cache
    - Mục dữ liệu được đọc từ cache vào CPU
    - Bộ nhớ chính không tham gia
  - Trường hợp không tìm thấy (miss case):
    - Mục dữ liệu được đọc từ bộ nhớ vào cache
    - Sau đó dữ liệu được chuyển từ cache vào CPU
    - ⇒ Miss penalty: thời gian truy cập mục dữ liệu bằng tổng thời gian truy cập cache và bộ nhớ chính

Ghi thông tin trong cache

- Trường hợp hit (bản tin thuộc 1 khối trong cache)
  - Write through (ghi thẳng): mục dữ liệu được ghi vào cache và ghi vào bộ nhớ đồng thời
  - Write back (ghi trễ): mục dữ liệu trước tiên được ghi vào cache và cả dòng (block) chứa nó ở trong cache sẽ được ghi lại vào bộ nhớ sau đó, khi mà dòng đó bị thay thế
- Trường hợp miss (bản tin ko có trong cache)
  - Write allocate (ghi có đọc lại): mục dữ liệu trước hết được ghi vào bộ nhớ sau đó cả dòng chứa nó sẽ được đọc vào cache
  - Write non-allocate (ghi không đọc lại): mục dữ liệu chỉ được ghi vào bộ nhớ

Các chính sách thay thế dòng cache

- Tại sao phải thay thế dòng cache?
  - Ánh xạ dòng (bộ nhớ) -> dòng (cache) thường là ánh xạ nhiều -> một
  - Nhiều dòng bộ nhớ chia sẻ một dòng cache -> các dòng bộ nhớ được nạp vào cache sử dụng một thời gian và được thay thế bởi dòng khác theo yêu cầu thông tin phục vụ CPU
- Chính sách thay thế (replacement policies): xác định cách thức lựa chọn các dòng trong cache để thay thế khi có dòng mới từ bộ nhớ cần chuyển vào
- 3 chiến lược chính:
  - Thay thế ngẫu nhiên (random replacement)
  - Vào trước ra trước FIFO (First In First Out)
  - Thay thế các dòng ít được sử dụng gần đây nhất LRU (Least Recently Used)

Các chính sách thay thế cache

- Thay thế ngẫu nhiên:
  - Các dòng trong cache được chọn ngẫu nhiên để thay
  - Đơn giản
  - Tỷ lệ miss cao vì phương pháp này không xét tới dòng cache nào đang thực sự được sử dụng
    - Nếu một dòng cache đang được sử dụng mà bị thay thế thì sự kiện miss xảy ra và cần phải đọc lại vào cache

Các chính sách thay thế cache

- Thay thế kiểu vào trước ra trước FIFO
  - Dựa trên nguyên lý FIFO
  - Các dòng cache được đọc vào cache trước sẽ được chọn để thay trước
  - Tỷ lệ miss thấp hơn so với phương pháp ngẫu nhiên
  - Tỷ lệ miss vẫn cao vì phương pháp này vẫn chưa thực sự xem xét tới block nào đang thực sự được sử dụng
    - Một dòng cache "cũ" có thể vẫn đang được sử dụng
  - Cài đặt phức tạp vì cần thêm mạch để giám sát thứ tự nạp các dòng bộ nhớ vào cache

Các chính sách thay thế cache

- Thay thế các dòng ít được sử dụng gần đây nhất LRU
  - Các dòng cache ít được sử dụng nhất trong thời gian gần đây sẽ được chọn để thay
  - Xét tới các dòng đang được sử dụng
  - Tỷ lệ miss thấp nhất so với phương pháp ngẫu nhiên và FIFO
  - Cài đặt phức tạp vì cần thêm mạch để giám sát tần suất sử dụng các dòng cache

Hiệu năng cache

- Thời gian truy cập trung bình của hệ thống nhớ có cache:

  Taccess = (hit cost) + (miss cost)

  = (Hit cost) + (miss rate) * (miss penalty)

  = t_{cache} + (1 - H) * (t_{memory})

  Trong đó H là hệ số hit

  Nếu t_{cache} = 5ns, t_{memory} = 60ns và H=80%, ta có:

  t_{access} = 5 + (1 – 0.8) * (60) = 5+12 = 17ns

  Nếu t_{cache} = 5ns, t_{memory} = 60ns và H=95%, ta có:

  t_{access} = 5 + (1 – 0.95) * (60) = 5+3 = 8ns

Các yếu tố ảnh hưởng – kích thước cache

- Kích thước cache lớn:
  - Có thể lưu trữ nhiều khối dữ liệu trong bộ nhớ hơn
  - Giảm tần suất trao đổi các khối dữ liệu của chương trình khác nhau giữa bộ nhớ và cache
  - Cache lớn chậm hơn cache nhỏ:
    - Tìm kiếm vị trí bộ nhớ trong không gian lớn hơn
  - Xu hướng: cache càng ngày càng lớn
    - Hỗ trợ đa nhiệm (multi-tasking) tốt hơn
    - Hỗ trợ tốt hơn cho xử lý song song
    - Hỗ trợ tốt hơn cho các hệ thống CPU đa nhân

Các yếu tố ảnh hưởng - Cache nhiều mức

- Nâng cao hiệu năng hệ thống vì cache nhiều mức có thể dung hòa tốc độ của CPU và MEM tốt hơn cache một mức

CPU	L1	L2	L3	Main memory
1ns	5ns	15ns	30ns	60ns
1ns	5ns			60ns

- Thực tế, cache thường có 2 mức: L1 và L2. Một số cache có 3 mức: L1, L2 và L3
- Giảm giá thành hệ thống nhớ

Các yếu tố ảnh hưởng – phân chia cache

- Cache có thể được chia thành cache lệnh và cache dữ liệu để hiệu năng tốt hơn vì:
  - Dữ liệu và lệnh khác nhau về tính cục bộ
    - Dữ liệu có tính chất cục bộ về thời gian hơn
    - Lệnh có tính chất cục bộ về không gian hơn
  - Cache lệnh chỉ hỗ trợ thao tác đọc còn cache dữ liệu hỗ trợ cả 2 thao tác đọc ghi

  ⇒ Dễ tối ưu cache hơn
  - Hỗ trợ nhiều thao tác đọc/ ghi cùng lúc => giảm xung đột tài nguyên
  - Kết hợp các chức năng khác như giải mã trước lệnh vào trong cache lệnh => xử lý lệnh tốt hơn

Các yếu tố ảnh hưởng – phân chia cache

- Trong thực tế, hầu hết cache L1 được chia thành 2 phần:
  - I_cache (Instruction cache): cache lệnh
  - D_cache (data cache): cache dữ liệu.
- Các cache mức cao hơn không được chia
  - Chia cache L1 có lợi ích cao nhất vì nó gần CPU nhất. CPU đọc/ ghi trực tiếp lên cache L1. Cache L1 cần hỗ trợ nhiều lệnh truy nhập đồng thời và các biện pháp tối ưu hóa
  - Chia các cache mức cao hơn không đem lại hiệu quả cao và làm phức tạp trong hệ thống điều khiển cache

Các biện pháp giảm miss

- Cache tốt:
  - Hệ số hit cao
  - Hệ số miss thấp
  - Miss penalty không quá chậm
- Các kiểu miss:
  - Compulsory misses (không tìm thấy bắt buộc): thường xảy ra tại thời điểm chương trình được kích hoạt, khi mã chương trình đang được tải vào bộ nhớ và chưa được nạp vào cache
  - Capacity misses (không tìm thấy do dung lượng): do kích thước của cache hạn chế, đặc biệt trong môi trường đa nhiệm. Khi cache nhỏ, mã chương trình thường xuyên bị chuyển đổi giữa cache và bộ nhớ
  - Conflict misses (không tìm thấy do xung đột): do có xung đột khi có nhiều dòng nhớ cùng cạnh tranh 1 dòng cache

Các biện pháp giảm miss

- Tăng kích thước dòng cache
  - Giảm compulsory misses: dòng có kích thước lớn sẽ có khả năng bao phủ lân cận tốt hơn
  - Tăng conflict miss:
    - Kích thước dòng lớn => giảm số lượng dòng trong cache => có nhiều dòng bộ nhớ cùng tham chiếu tới dòng cache hơn => tăng conflict miss
  - Kích thước dòng lớn có thể gây lãng phí dung lượng của cache (một số phần của dòng có thể không bao giờ được sử dụng)
  - Kích thước dòng thường dùng là 64 bytes

Các phương pháp giảm miss

- Tăng mức độ liên kết (tăng số lượng cache way):
  - Giảm conflict miss:
    - Tăng số lượng ways => ánh xạ bộ nhớ - cache linh hoạt hơn hay nhiều lựa chọn hơn => giảm conflict miss
  - Làm cache chậm hơn:
    - Tăng số lượng ways => không gian tìm kiếm lớn hơn => làm cache chậm đi

Bài tập bộ nhớ cache

- Tính số bit cần thiết để tạo một bộ nhớ cache ánh xạ trực tiếp chứa 64KB dữ liệu và mỗi dòng nhớ chứa 1 từ dữ liệu. Biết đây là bộ VXL 32 bit?

- Tính số bit cần thiết để tạo 4-way bộ nhớ cache ánh xạ tập kết hợp để chứa cùng lượng dữ lệu (64KB)?
- Cũng câu hỏi trên cho bộ nhớ cache ánh xạ linh hoạt?

- Tính số bit cần thiết để tạo một bộ nhớ cache ánh xạ trực tiếp chứa 64KB dữ liệu và mỗi dòng nhớ chứa 8 từ dữ liệu. Biết đây là bộ VXL 32 bit?

RAID

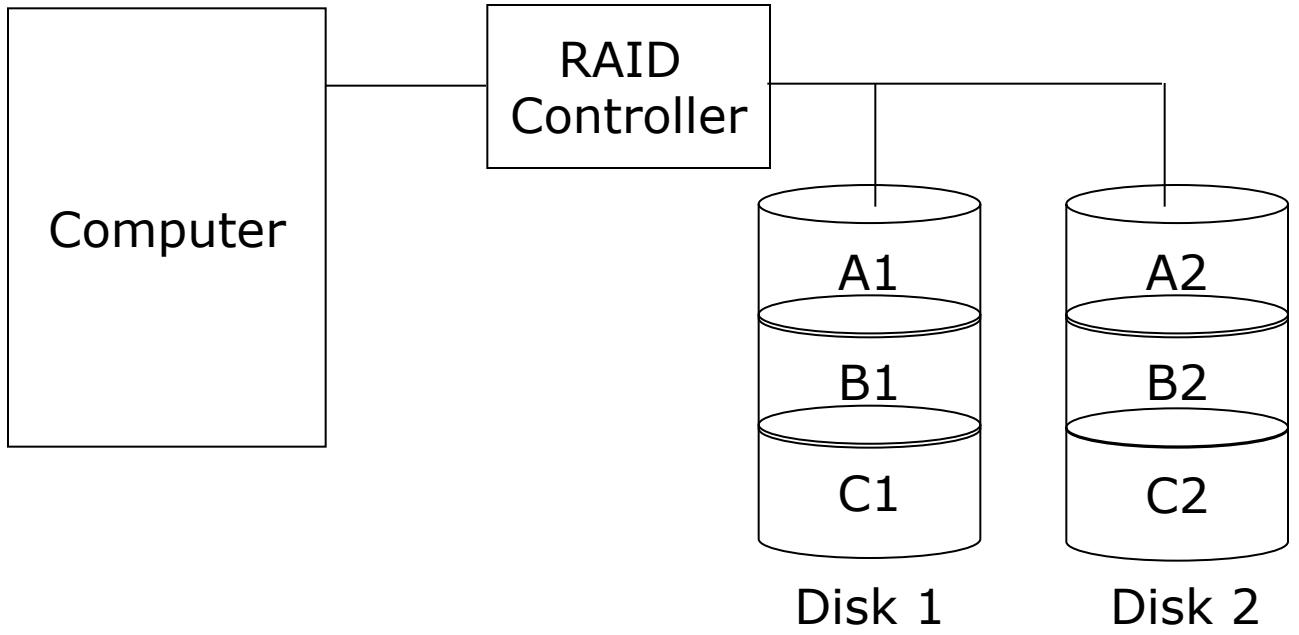
Giới thiệu về RAID

- RAID (Redundant Array of Independent Disks) là công nghệ tạo các thiết bị lưu trữ tiên tiến trên cơ sở đĩa cứng những mục đích sau:
  - Hiệu năng, tốc độ cao (high performance/ speed)
  - Độ tin cậy cao (high reliability)
  - Dung lượng lớn (large volume)
- RAID là một tập/ mảng các HDD nhưng được HDH coi như 1 ổ đĩa logic
- Các đĩa cứng theo chuẩn SATA và SCSI mới hỗ trợ tạo RAID
- Dữ liệu được phân tán trên các đĩa vật lý
- Đĩa dư thừa được sử dụng để lưu trữ thông tin parity => đảm bảo khôi phục dữ liệu

Các kỹ thuật RAID

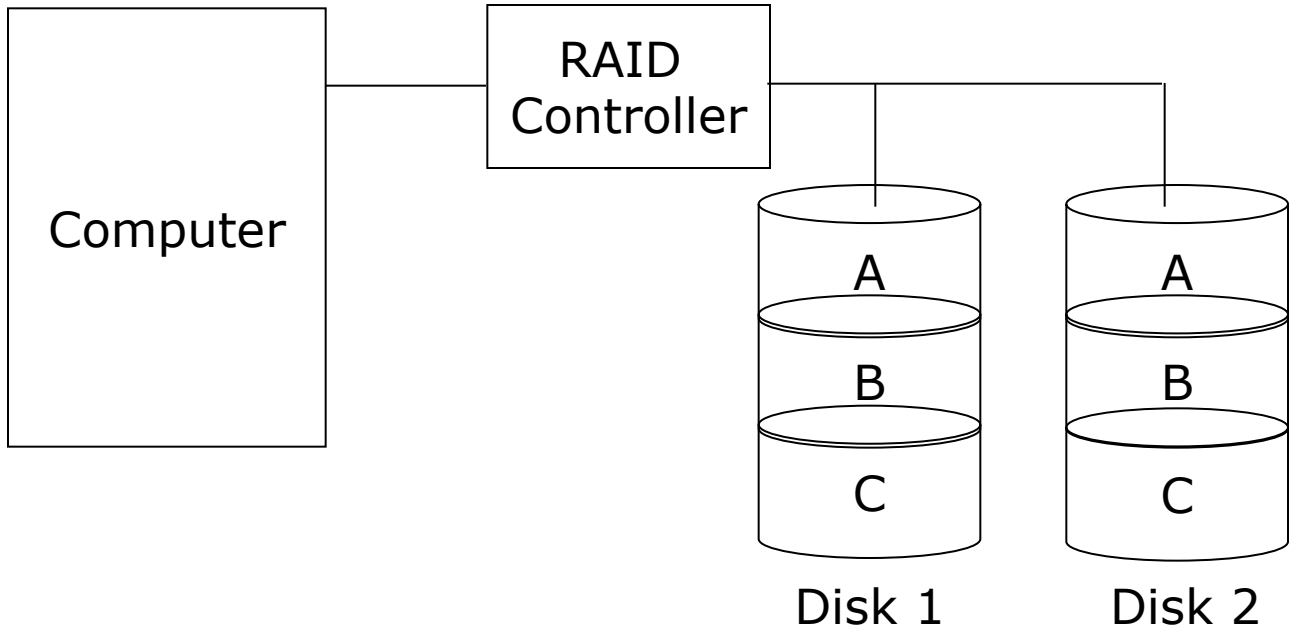
- 2 kỹ thuật cơ bản được sử dụng trong RAID:
  - Disk stripping (tạo lát đĩa):
    - Dữ liệu được chia thành các khối và mỗi khối được ghi đồng thời vào một đĩa độc lập
    - Sau đó, các khối dữ liệu có thể được đọc từ HDD một cách đồng thời
    - ⇒ Cải thiện tốc độ truy cập
  - Disk mirroring :
    - Dữ liệu được chia thành các khối và mỗi khối được ghi vào một số đĩa
    - Tại thời điểm bất kì, luôn có nhiều hơn 1 bản sao dữ liệu => độ tin cậy tăng

RAID – disk stripping



Disk stripping technique

RAID – disk mirroring

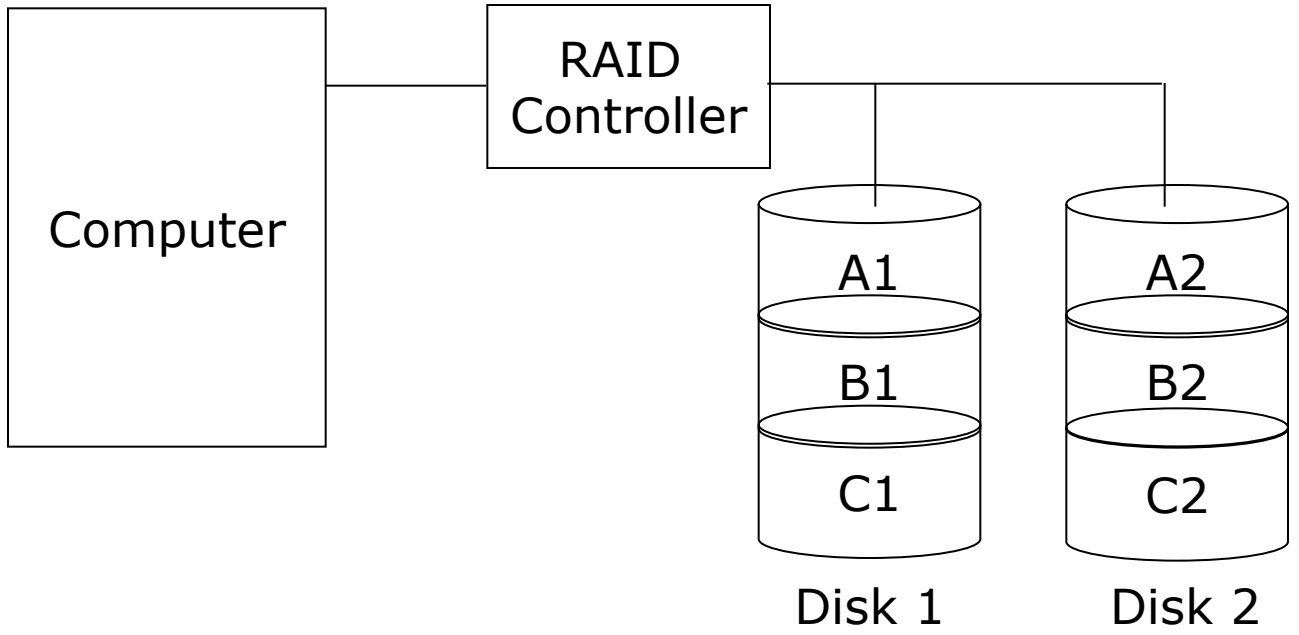


Disk mirroring technique

Các loại RAID

- Một số loại RAID thông dụng:
  - RAID 0
  - RAID 1
  - RAID 2
  - RAID 3
  - RAID 4
  - RAID 5
  - RAID 6

RAID 0 – disk stripping

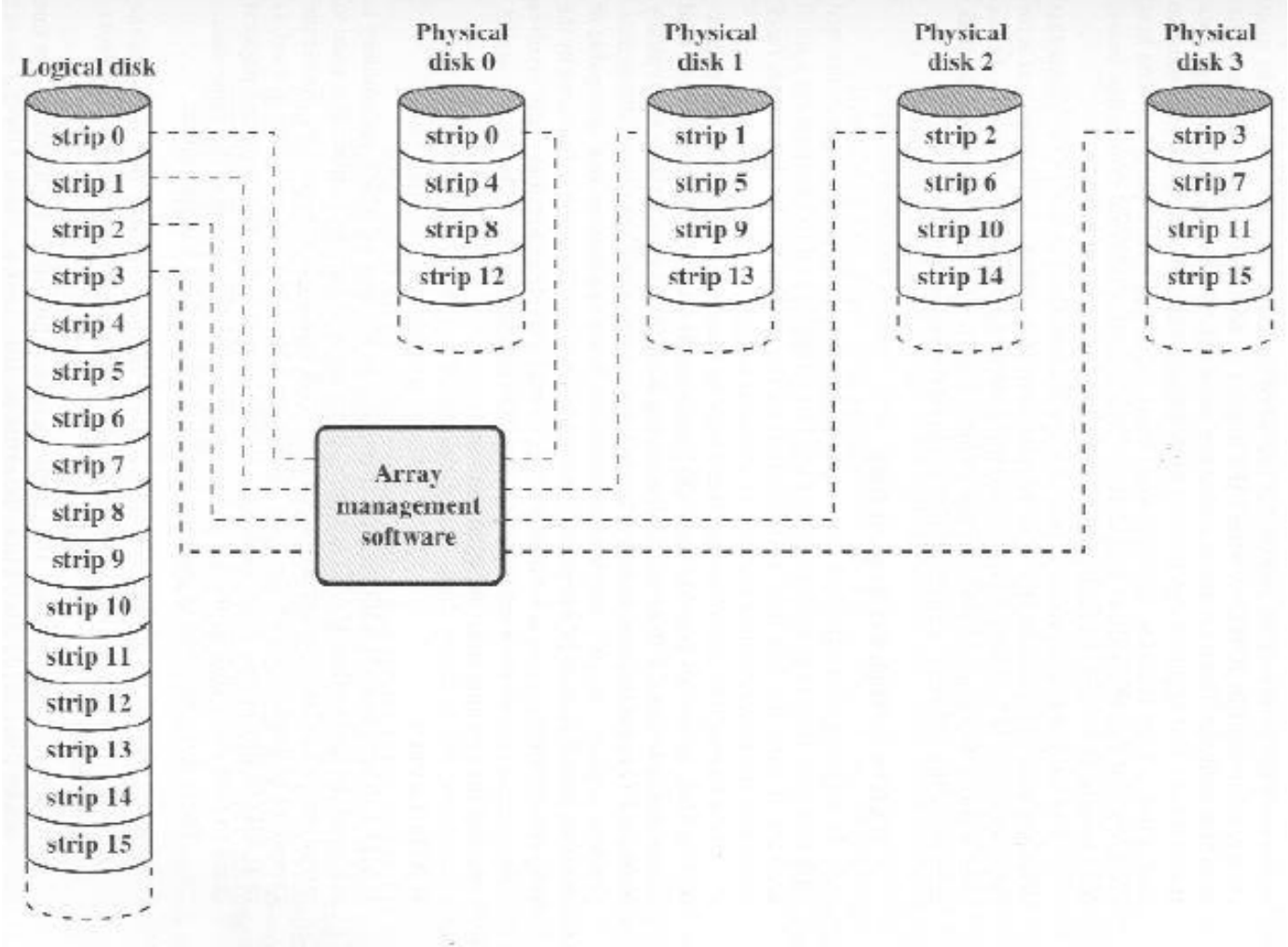


Disk stripping technique

RAID 0 – disk stripping

- Các đặc điểm:
  - Dựa trên kĩ thuật disk stripping (đọc/ ghi song song)
  - Dữ liệu được phân bố trên các đĩa trong mảng
  - Tối thiểu cần 2 HDD

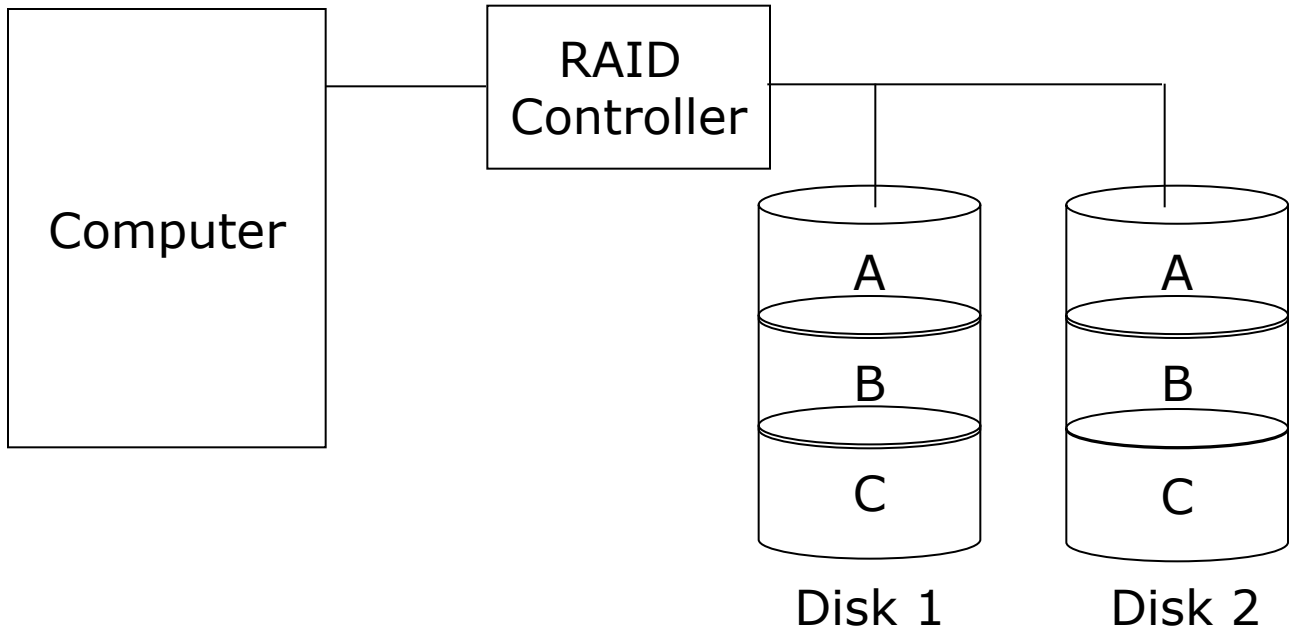
RAID 0 – disk tripping



RAID 0 – disk tripping

- Ưu:
  - Tốc độ nhanh
  - Đáp ứng tốt các hệ thống nhu cầu I/O cao
  - Dung lượng là tổng của tất cả các đĩa
- Nhược:
  - Độ tin cậy như một đĩa

RAID 1 – disk mirroring



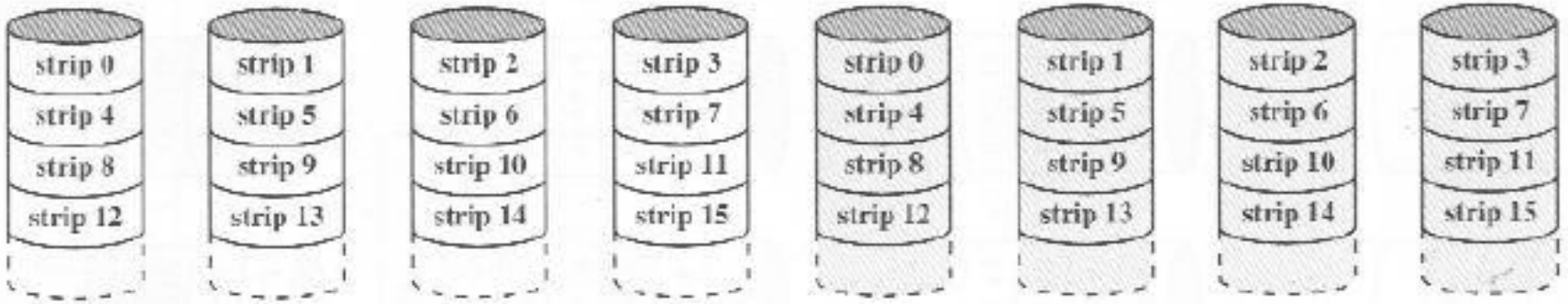
RAID 1 - Disk mirroring

RAID 1 – disk mirroring

- Các đặc điểm:
  - Dựa trên kĩ thuật disk mirroring (nhiều bản sao)
  - Tính dư thừa có được đơn giản bằng cách sao tất cả dữ liệu
  - Tối thiểu cần 2 HDD
  - Dữ liệu cũng phân mảnh (data stripping) như RAID 0 nhưng mỗi mảnh logic được ánh xạ tới 2 đĩa vật lý khác nhau

  => Mỗi đĩa trong mảng có một bản sao cùng dữ liệu (mirror)

RAID 1 – disk mirroring

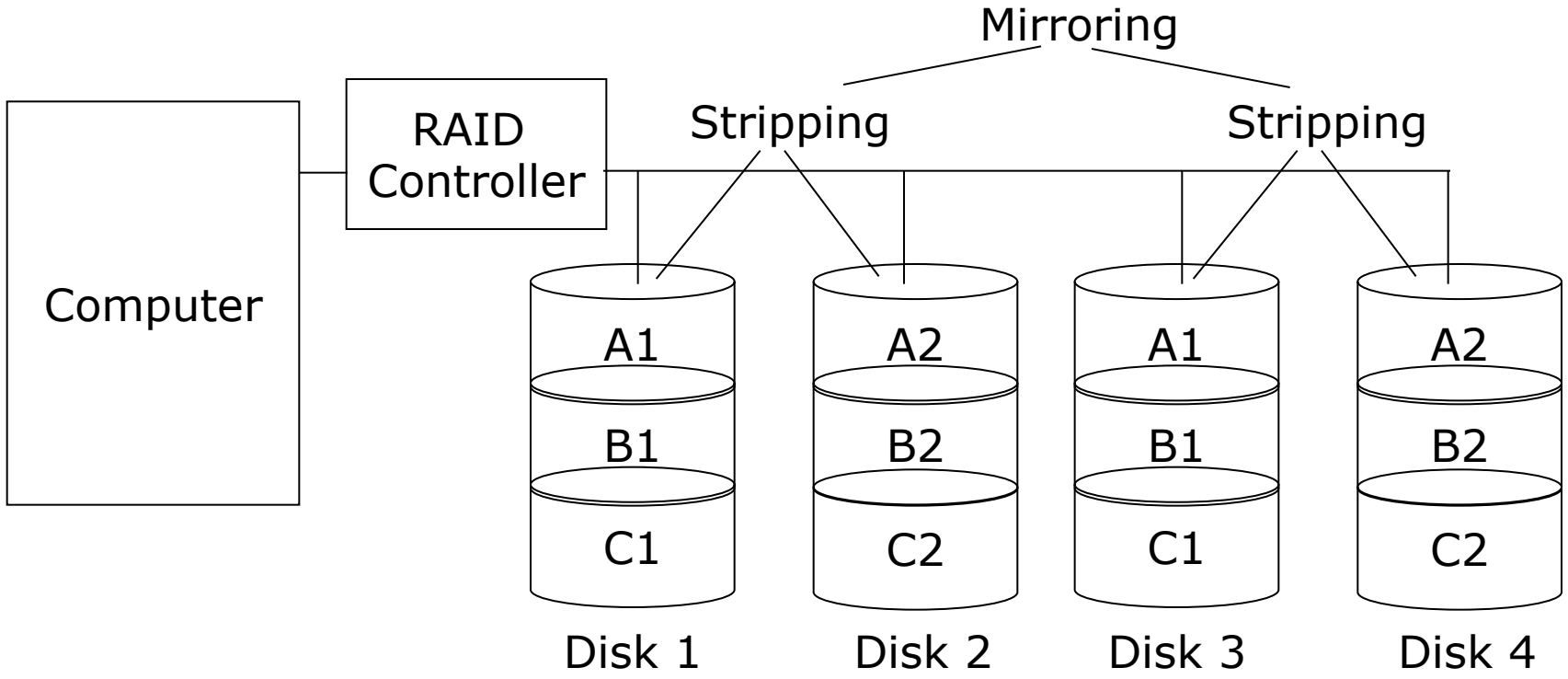


(b) RAID 1 (Mirrored)

RAID 1 – disk mirroring

- Ưu:
  - Độ tin cậy cao
- Nhược:
  - Dung lượng thực sự bằng ½ tổng số đĩa
  - Chi phí cao

RAID 10 – disk stripping & mirroring



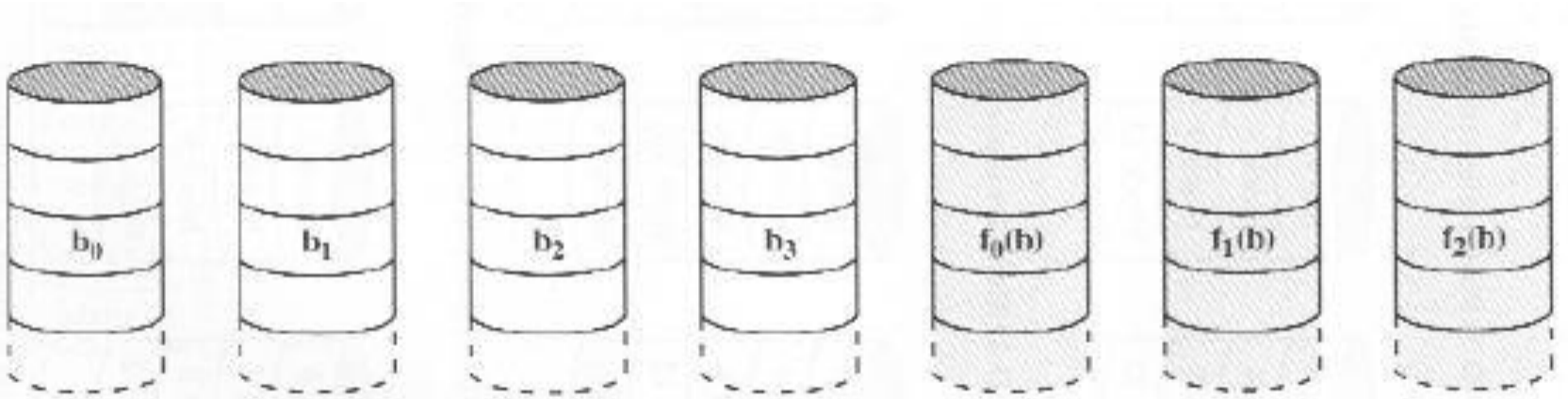
RAID 10 – disk stripping & mirroring

- Các đặc điểm:
  - Tối thiểu cần 4 HDD
  - Dựa trên kĩ thuật disk mirroring và stripping
- Ưu:
  - Nhanh hơn so với một đĩa
  - Tin cậy hơn so với một đĩa
- Nhược:
  - Dung lượng bằng một nửa dung lượng tổng số đĩa

RAID 2

- Các mảnh (strip) rất nhỏ, thường là byte hoặc word
- Các mã sửa sai được tính ứng với các bit trên đĩa dữ liệu tương ứng
- Thường sử dụng mã Hamming
- Cần ít đĩa hơn RAID1 nhưng vẫn tốn kém
- Số lượng đĩa dư thừa tỉ lệ thuận với log số lượng đĩa dữ liệu
- Yêu cầu đọc: dữ liệu và mã sửa sai được gửi tới bộ điều khiển

RAID 2

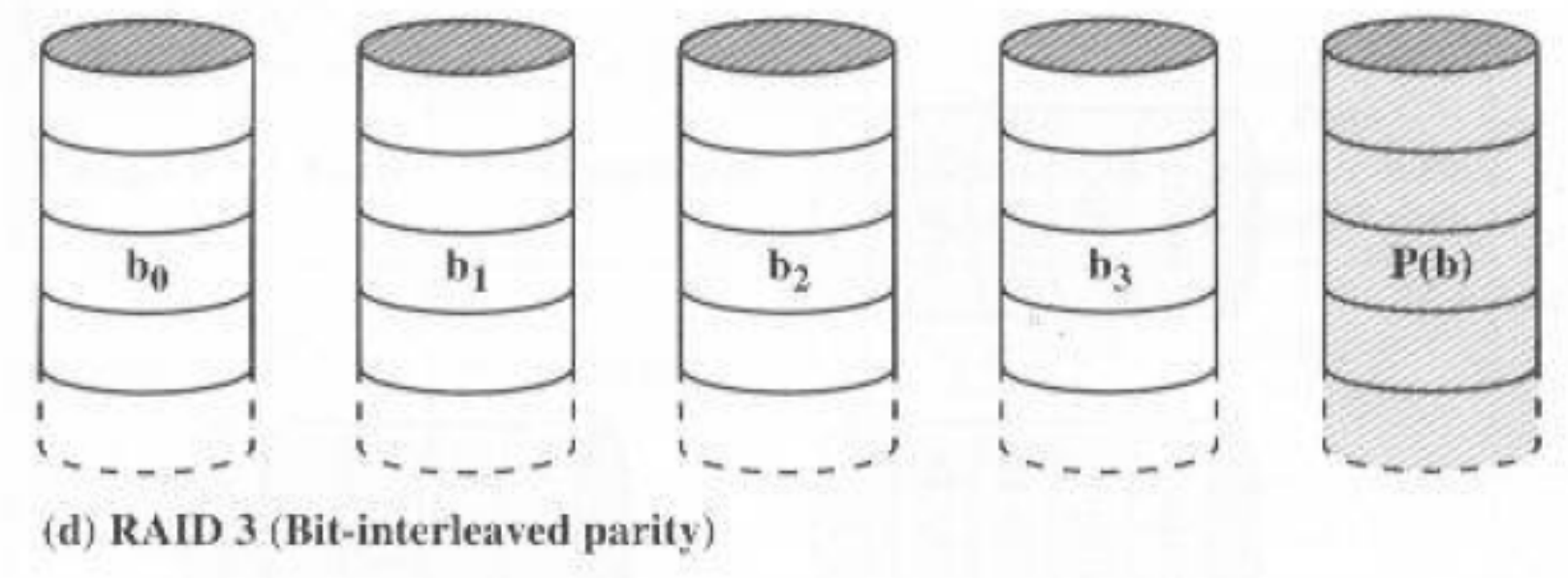


(c) RAID 2 (Redundancy through Hamming code)

RAID 3

- Cấu trúc tương tự như RAID2
- Chỉ cần 1 đĩa dư thừa
- Bit chẵn lẻ đơn giản được tính cho tập các bit cùng vị trí trên các đĩa

RAID 3

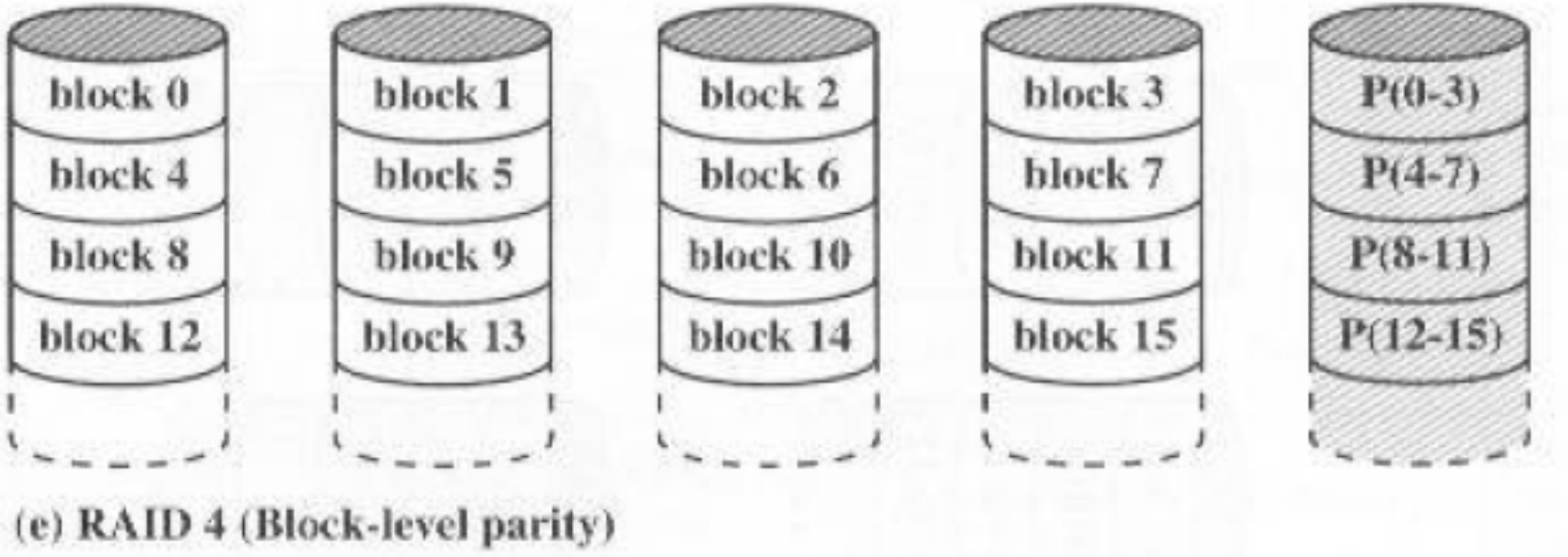


(d) RAID 3 (Bit-interleaved parity)

RAID 4

- Từ RAID4 -> RAID6 sử dụng kỹ thuật truy cập độc lập:
  - Mỗi đĩa thành viên hoạt động độc lập => các yêu cầu I/O riêng biệt có thể được đáp ứng song song
- Vẫn sử dụng data stripping nhưng các mảnh khá lớn
- Các mảnh parity theo từng bit được tính theo các mảnh tương ứng trên đĩa và lưu vào strip tương ứng trên đĩa parity

RAID 4

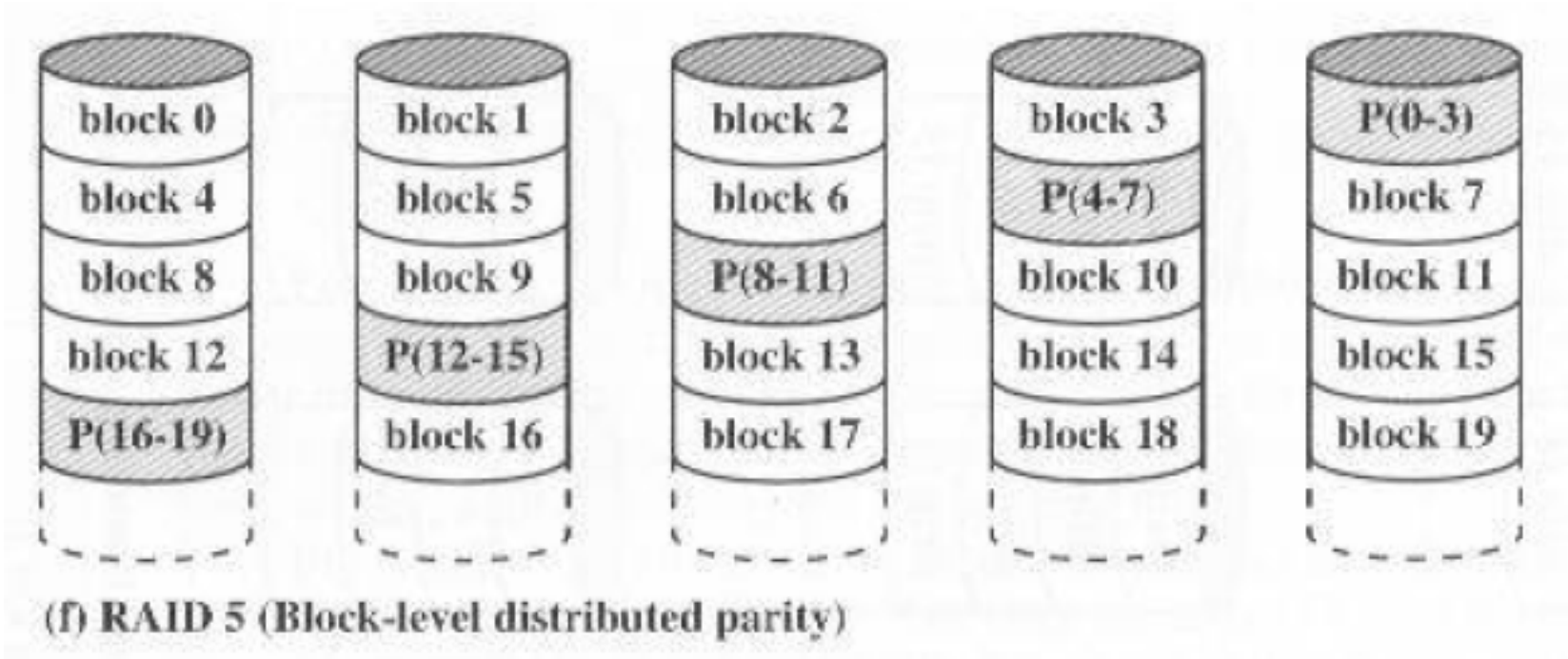


(e) RAID 4 (Block-level parity)

RAID 5

- Tổ chức tương tự như RAID4
- RAID5 phân bố các mảnh parity trên tất cả các đĩa

RAID 5

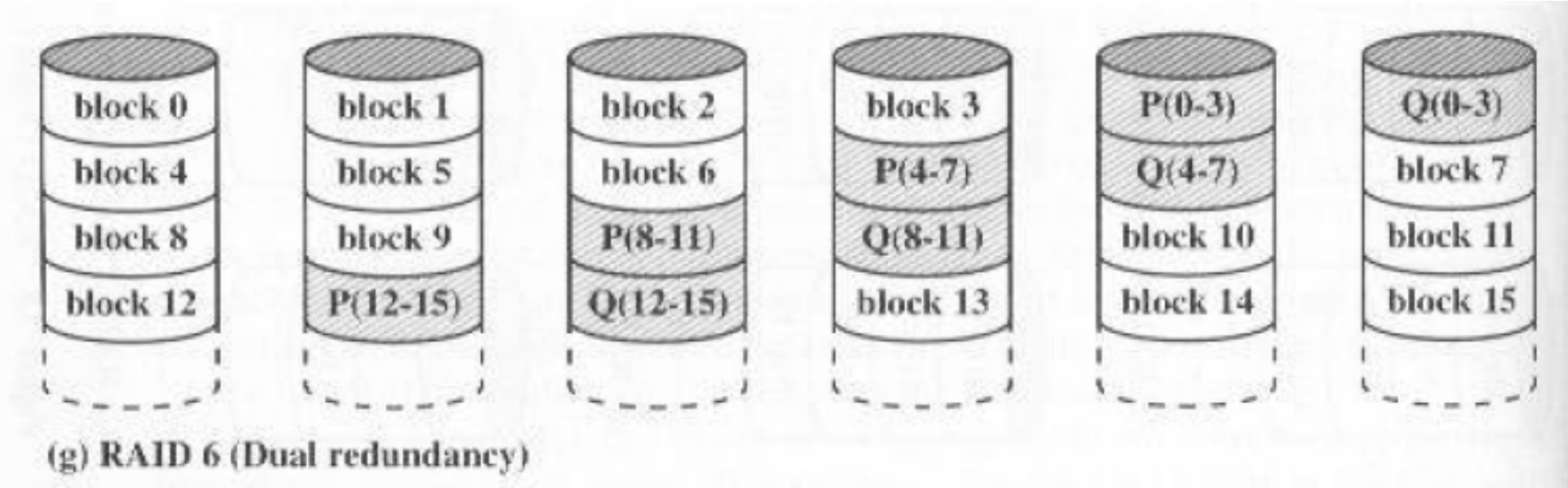


(f) RAID 5 (Block-level distributed parity)

RAID 6

- Hai mã parity được tính và lưu trên các block riêng biệt trên các đĩa khác nhau

RAID 6



(g) RAID 6 (Dual redundancy)