KIẾN TRÚC MÁY TÍNH
ET4270

TS. Nguyễn Đức Minh

[Adapted from Computer Organization and Design, 4th Edition, Patterson & Hennessy, © 2008, MK]
[Adapted from Computer Architecture lecture slides, Mary Jane Irwin, © 2008, PennState University]

Tổ chức lớp

Số tín chỉ	3 (3-1-1-6)
Giảng viên	TS. Nguyễn Đức Minh
Văn phòng	C9-401
Email	minhnd1@gmail,com
Website	https://sites.google.com/site/fethutca/home
Sách	*Computer Org and Design*, 3rd Ed., Patterson &Hennessy, ©2007
	Digital Design and Computer Architecture, David Money Harris
Thí nghiệm	3 bài
Bài tập	Theo chương, đề bài xem trên trang web

Điểm số

Điều kiện thi **Lab**

Bài thi giữa kỳ **30%**

Bài tập 20% (Tối đa 100 điểm)

Tiến trình 10%

Tối đa: 100 điểm,

Bắt đầu: 50 điểm

Tích lũy, trừ qua trả lời câu hỏi trên lớp và đóng góp tổ chức lớp

Bài thi cuối kỳ **70%**

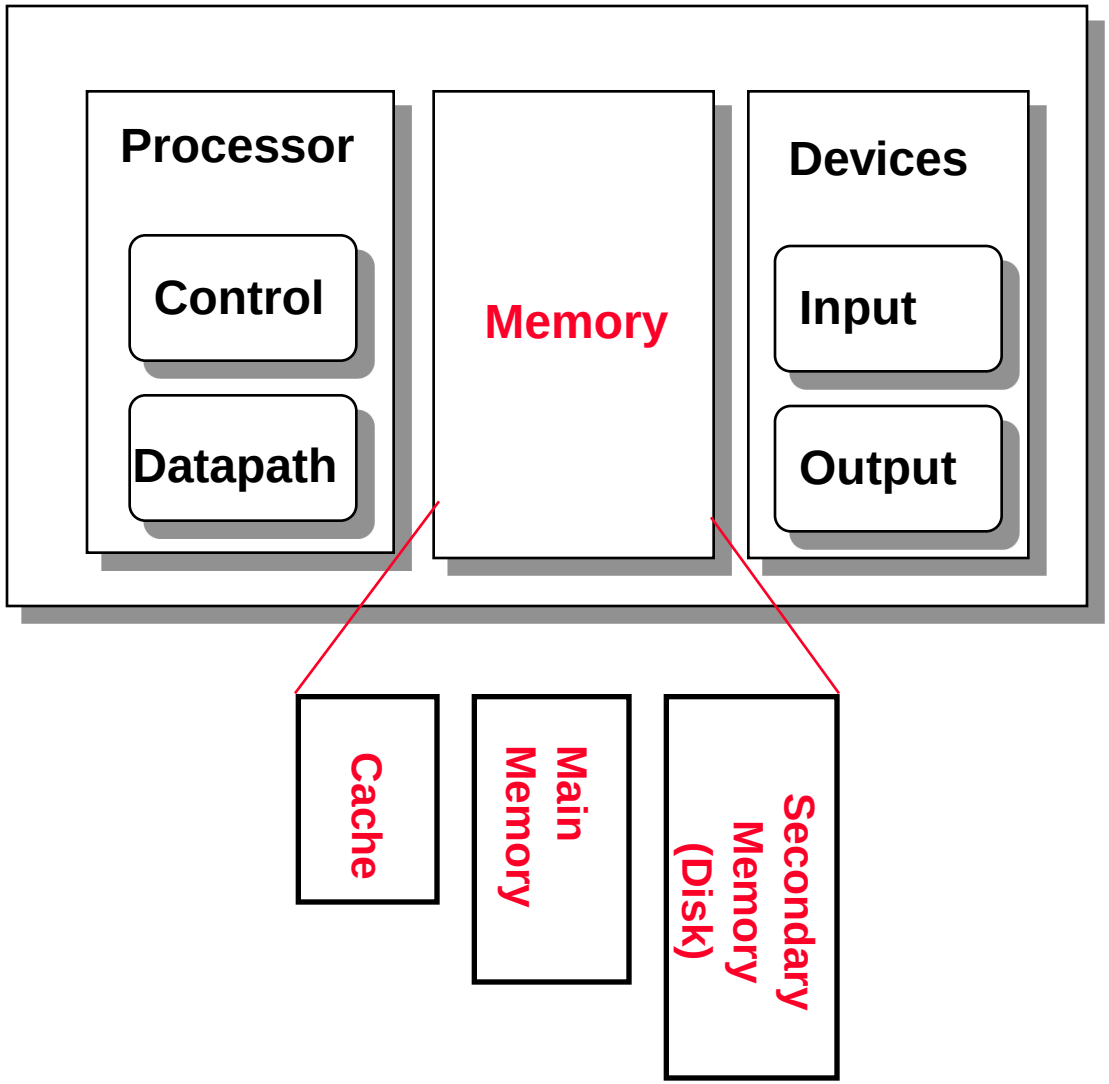
Lịch học

- Thời gian:
  - Từ 14h00 đến 17h20
  - Lý thuyết: 11 buổi x 135 phút / 1 buổi
  - Bài tập: 4 buổi x 135 phút / 1 buổi
  - Thay đổi lịch (nghỉ, học bù) sẽ được thông báo trên website trước 2 ngày

Tổng kết chương 3

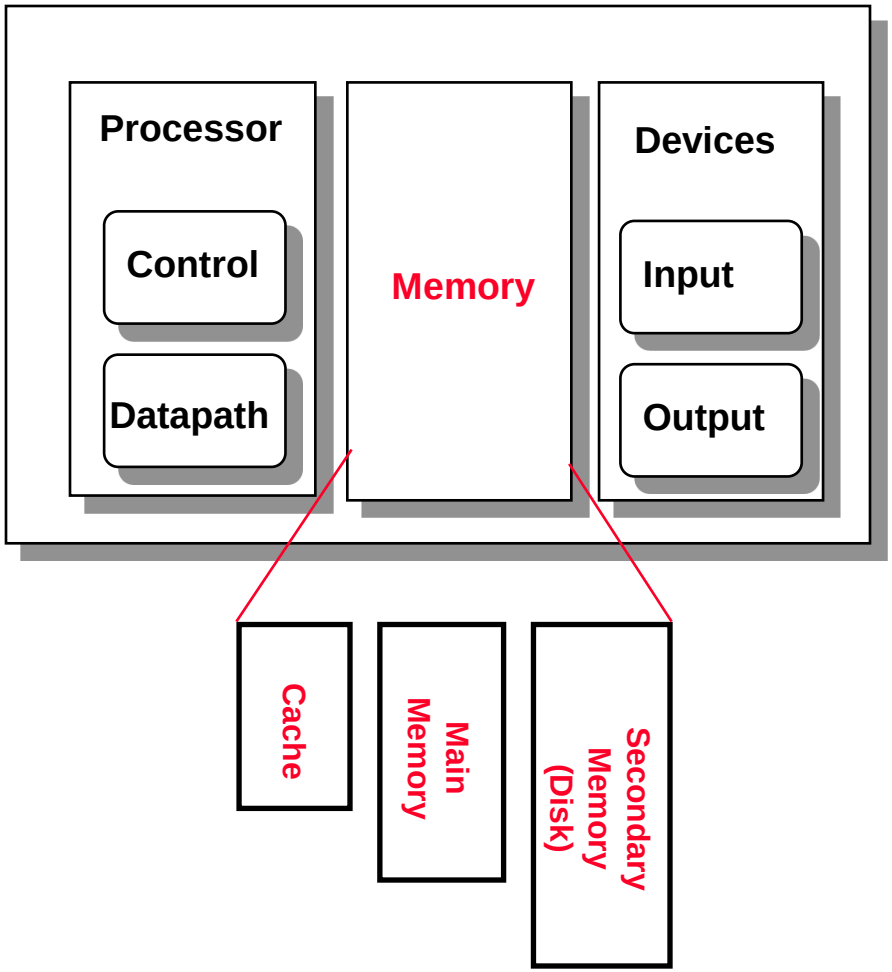
- Tất cả các bộ xử lý hiện đại đều dùng pipeline để tăng hiệu suất (CPI=1 và đồng hồ nhanh - fc lớn)
- Tốc độ đồng hồ pipeline bị giới hạn bởi giai đoạn pipeline chậm nhất – thiết kế pipeline cân bằng là rất quan trọng
- Cần phát hiện và giải quyết xung đột trong pipeline
  - Xung cấu trúc – giải quyết: thiết kế pipeline đúng
  - Xung đột dữ liệu
    - Dừng (ảnh hưởng CPI)
    - Chuyển tiếp (cần phần cứng hỗ trợ)
  - Xung đột điều khiển – đặt phần cứng quyết định rẽ nhánh lên các trạng thái đầu trong pipeline
    - Dừng (ảnh hưởng CPI)
    - Rẽ nhánh chậm (cần hỗ trợ của trình dịch)
    - Dự đoán rẽ nhánh tĩnh và động (cần phần cứng hỗ trợ)
- Xử lý ngắt trong pipeline phức tạp

Nhắc lại: Các thành phần cơ bản của máy tính

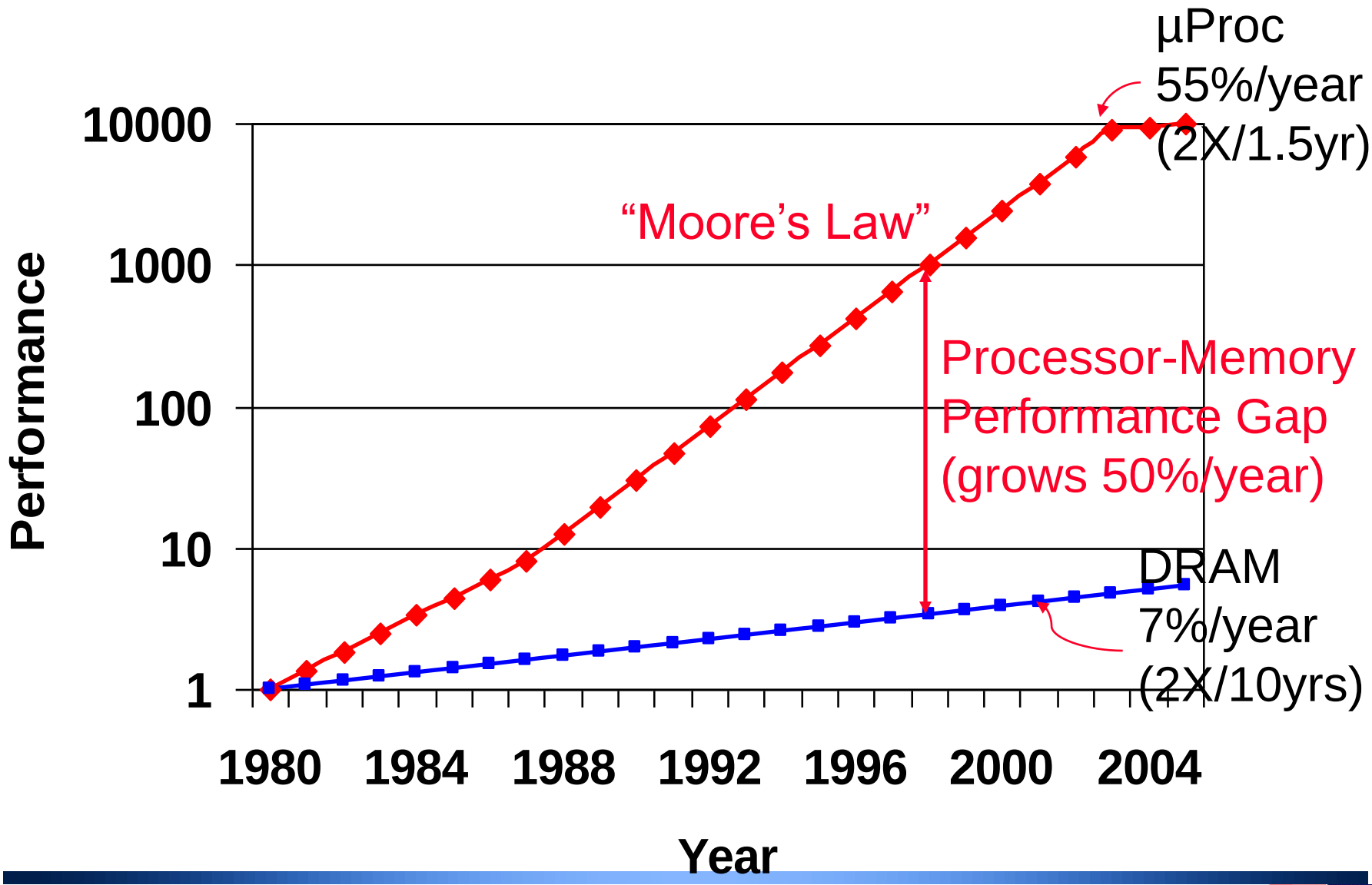


Nội dung

- Phân cấp bộ nhớ trong máy tính
  - Mục đích
  - Tính khả thi
- Bộ đệm cơ bản
  - Nguyên lý
  - Cấu trúc
  - Hoạt động
  - Hiệu năng
- Phương pháp tăng hiệu năng
  - Bộ đệm kết hợp
  - Bộ đệm đa mức
- Bộ nhớ ảo

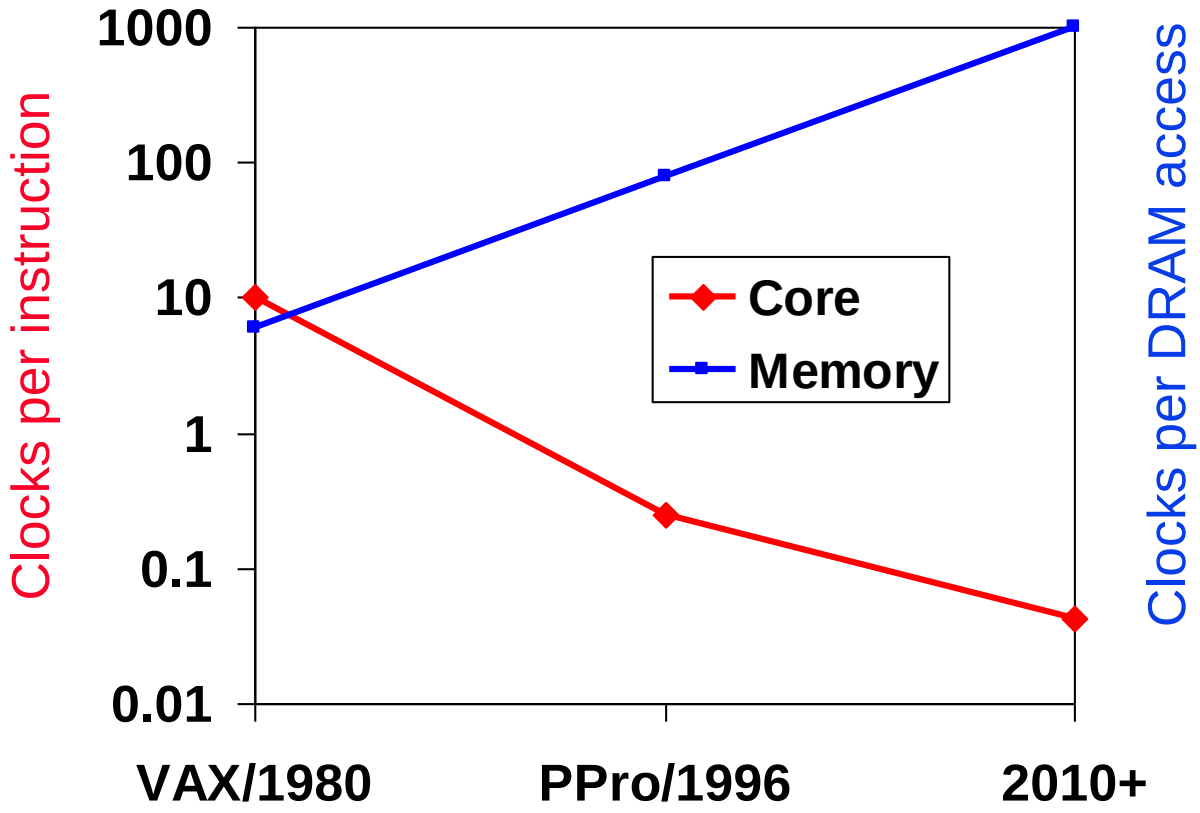


Processor-Memory Performance Gap



"Bức tường bộ nhớ"

- Chênh lệch tốc độ bộ xử lý và RAM động tiếp tục tăng



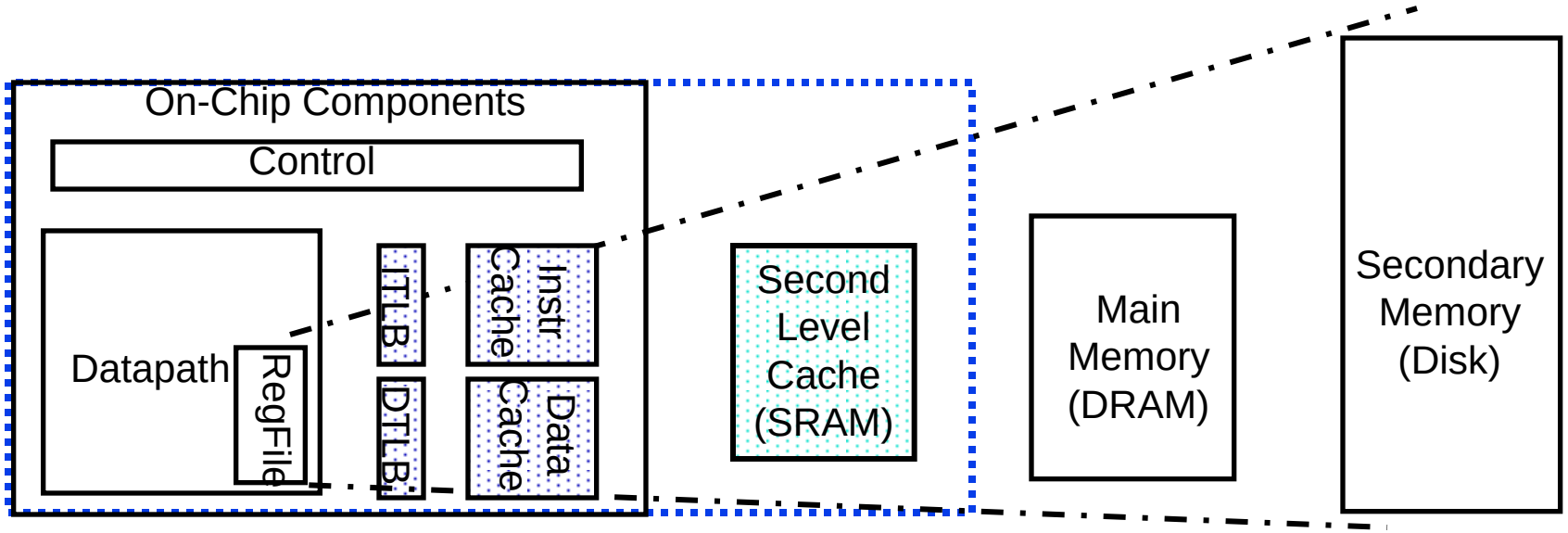
- Phân cấp bộ nhớ (bộ đệm) ngày càng quan trọng để tăng hiệu năng chung

Mục tiêu của phân cấp bộ nhớ

- Thực tế: Bộ nhớ lớn chậm, bộ nhớ nhanh nhỏ
- Bằng cách nào tạo ra 1 bộ nhớ có vẻ lớn, rẻ và nhanh (trong hầu hết thời gian)?
  - Bằng phân cấp bộ nhớ
  - Bằng song song

Phân cấp bộ nhớ thông thường

- Tập dụng nguyên tắc "cục bô" để cung cấp cho người dùng kích thước bộ nhớ lớn như công nghệ bộ nhớ rẻ rất nhưng ở tốc độ cao như công nghệ bộ nhớ nhanh nhất



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

Phân cấp bộ nhớ: Tại sao nó hoạt động?

- Cục bộ theo thời gian
  - Nếu một vị trí bộ nhớ được truy cập thì nó sẽ sớm được truy cập lại

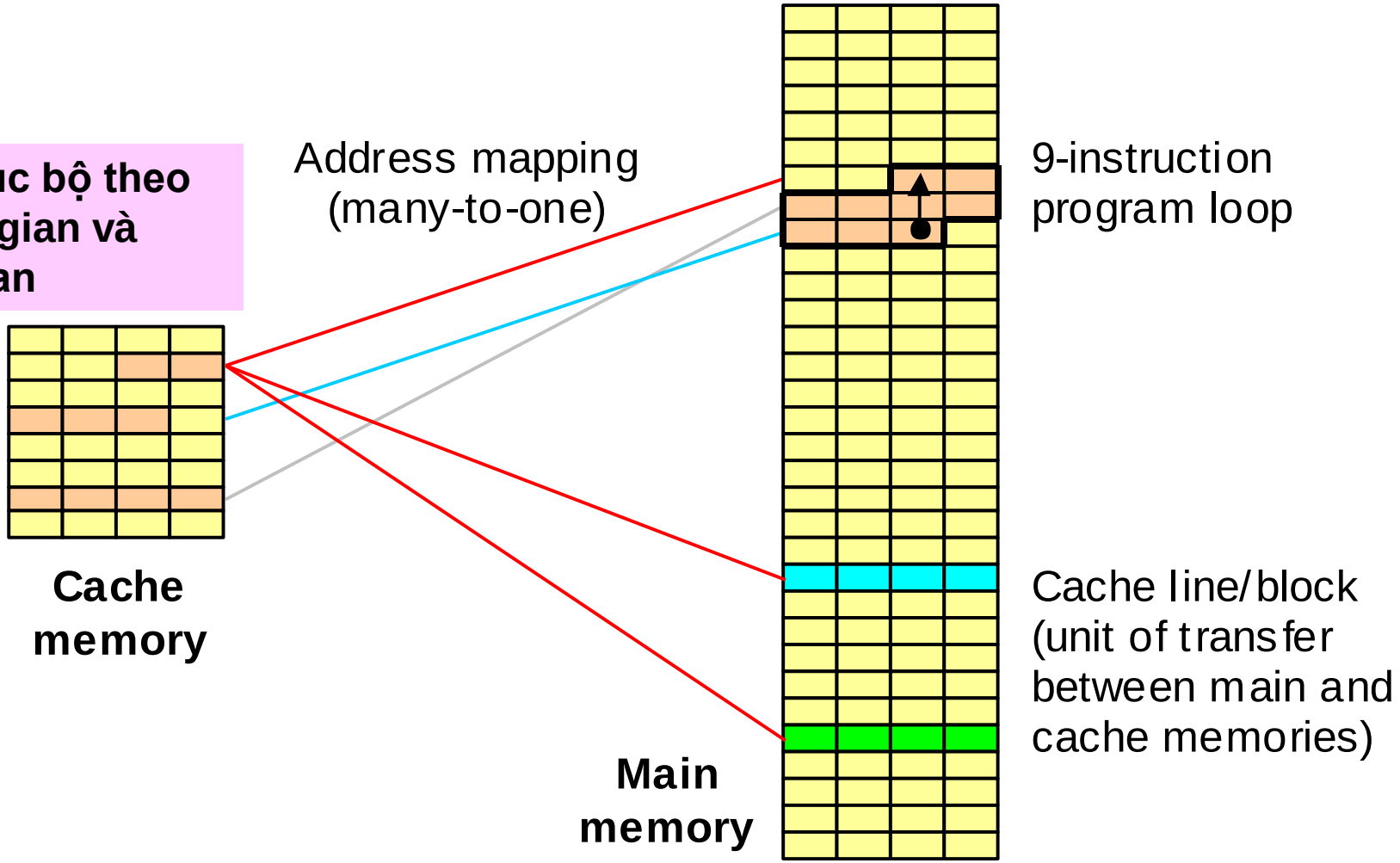
  $\Rightarrow$ Lưu các dữ liệu vừa được truy cập nhiều nhất ở gần bộ xử lý

- Cục bộ theo không gian
  - Nếu một vị trí bộ nhớ được truy cập thì các vị trí có địa chỉ gần đó sẽ sớm được truy cập

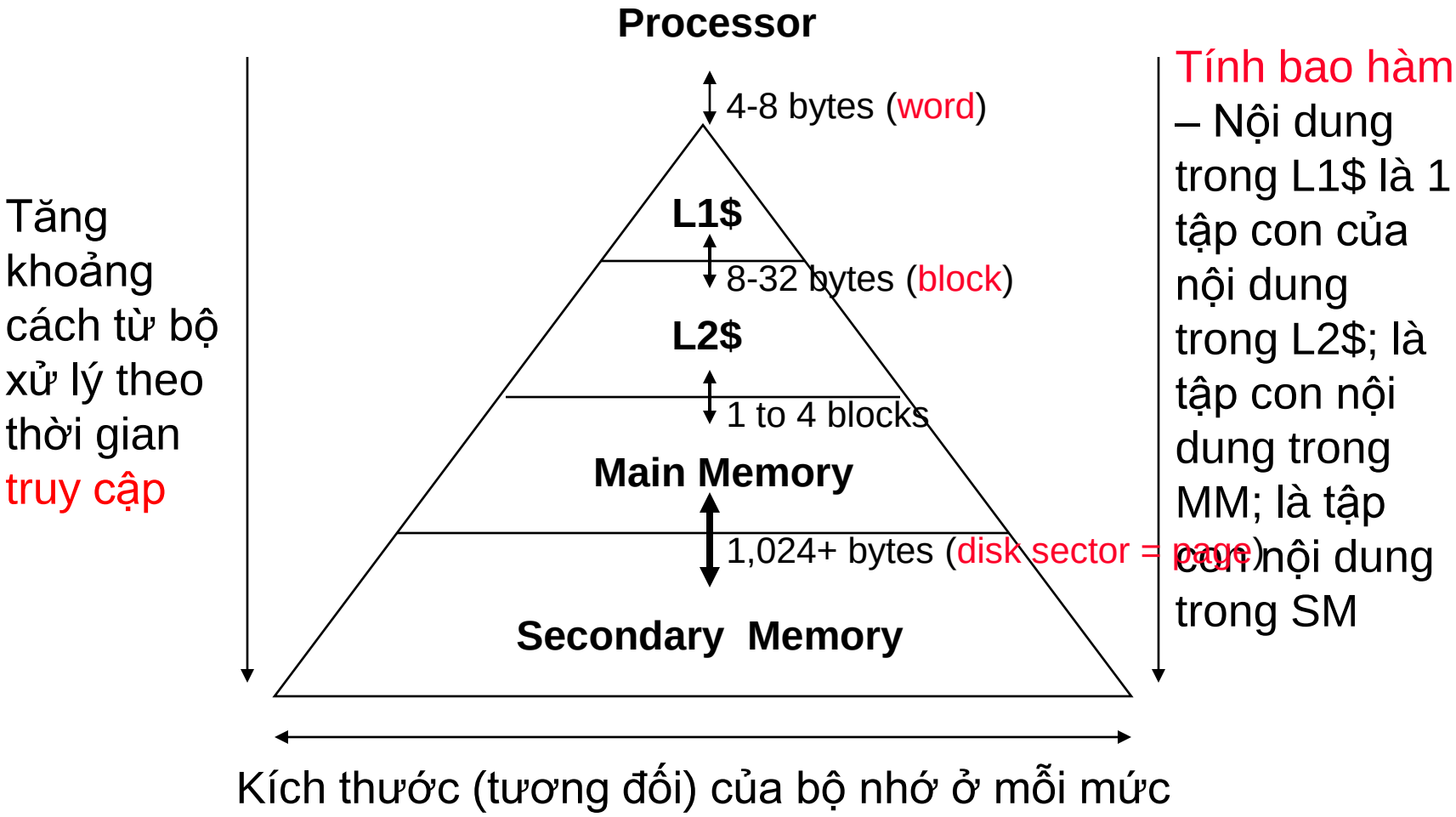
  $\Rightarrow$ Đưa các khối bộ nhớ chứa các từ cạnh nhau đến gần bộ xử lý hơn

Tính cục bộ

Tính cục bộ theo không gian và thời gian



Các mức phân cấp bộ nhớ



Phân cấp bộ nhớ: Khái niệm

- Khối (hoặc đường): đơn vị thông tin nhỏ nhất có (hoặc không có) trong bộ đệm – lượng thông tin nhỏ nhất được di chuyển giữa 2 bộ nhớ ở 2 mức liên tiếp trong phân cấp
- Tỷ lệ trúng (*Hit Rate*): Tỷ lệ số lần truy cập bộ nhớ tìm thấy ở 1 mức trong phân cấp bộ nhớ
  - Thời gian trúng (*Hit Time*): Thời gian truy cập mức bộ nhớ đó trong phân cấp bộ nhớ

    Thời gian truy cập 1 khối + Thời gian xác định trúng/trượt
- Tỷ lệ trượt (*Miss Rate*): Tỷ lệ số lần truy cập bộ nhớ không tìm thấy ở 1 mức trong phân cấp bộ nhớ $\Rightarrow$ 1 - (Hit Rate)
  - Tổn thất trượt (*Miss Penalty*): Thời gian thay thế 1 khối ở mức bộ nhớ đó bằng khối tương ứng từ mức bộ nhớ thấp hơn

    Thời gian truy cập khối ở mức thấp hơn + Thời gian truyền khối đến mức bộ nhớ có sự trượt + Thời gian chèn khối váo mức đó + Thời gian đưa dữ liệu tới nơi yêu cầu

Hit Time << Miss Penalty

Quản lý sự dịch chuyển dữ liệu giữa các mức

- Thanh ghi ↔ Bộ nhớ
  - Trình biên dịch (người lập trình?)
- Bộ đệm ↔ bộ nhớ chính
  - Phần cứng điều khiển bộ đệm
- Bộ nhớ chính ↔ Đĩa
  - Hệ điều hành (bộ nhớ ảo)
  - Ánh xạ địa chỉ ảo và địa chỉ vật lý nhờ phần cứng (Translation Lookaside Buffer)
  - Người lập trình (các tệp)

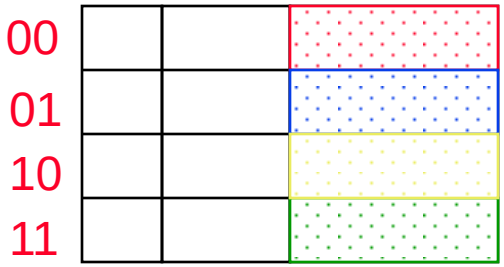
Cơ sở bộ đệm

- Trả lời 2 câu hỏi ở phần cứng:
  - Q1: Một mục dữ liệu có trong bộ đệm hay không?
  - Q2: Một mục dữ liệu ở đâu trong bộ đệm?
- Ánh xạ trực tiếp
  - Mỗi khối bộ nhớ được ánh xạ vào chính xác 1 khối trong bộ đệm
    - Nhiều khối trong bộ nhớ ở mức thấp cùng chia sẻ 1 khối trong bộ đệm
  - Ánh xạ bộ nhớ (trả lời câu hỏi Q2):

    (block address) modulo (# of blocks in the cache)
  - Có trường thẻ(tag) gắn với mỗi khối bộ đệm, chứa thông tin địa chỉ (các bít cao của địa chỉ) cần cho việc xác định khối (trả lời câu hỏi Q1)

Ví dụ 4.1. Bộ đệm ánh xạ trực tiếp đơn giản

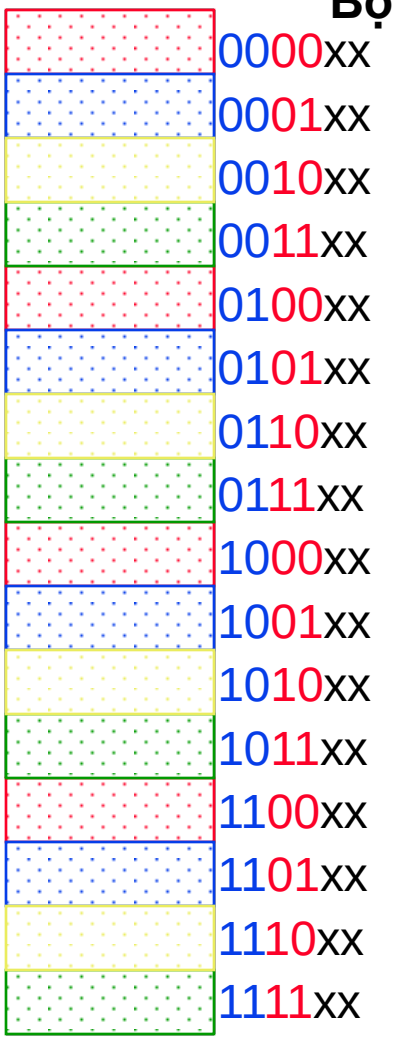
Bộ đệm: 4 khối nhớ



Q1: Có trong bộ đệm không?

So sánh trường thẻ bộ đệm với 2 bit cao của địa chỉ bộ nhớ để xác định khối dữ liệu có trong bộ đệm không?

Bộ nhớ chính: 16 khối 1 từ



Các khối 1 từ: 2 bít thấp dùng để xác định các byte trong từ (32b words)

Q2: Vị trí các từ trong bộ đệm?

Dùng 2 bít thấp tiếp theo của địa chỉ –chỉ số – để xác định khối bộ đệm nào (i.e., chia lấy dư cho số khối trong bộ đệm)

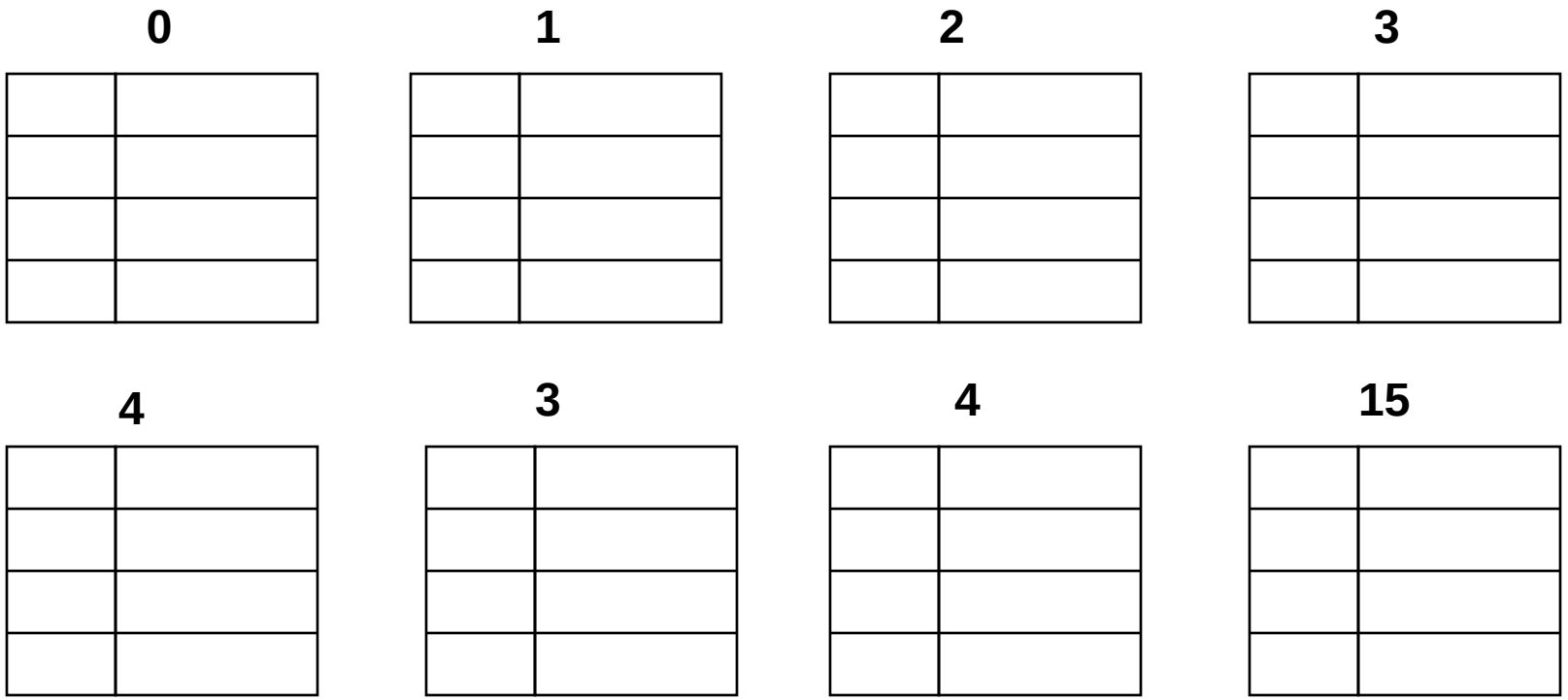
(block address) modulo (# of blocks in the cache)

Truy cập ô nhớ với bộ đệm ánh xạ trực tiếp

- Xét việc truy cập các ô nhớ trong bộ nhớ

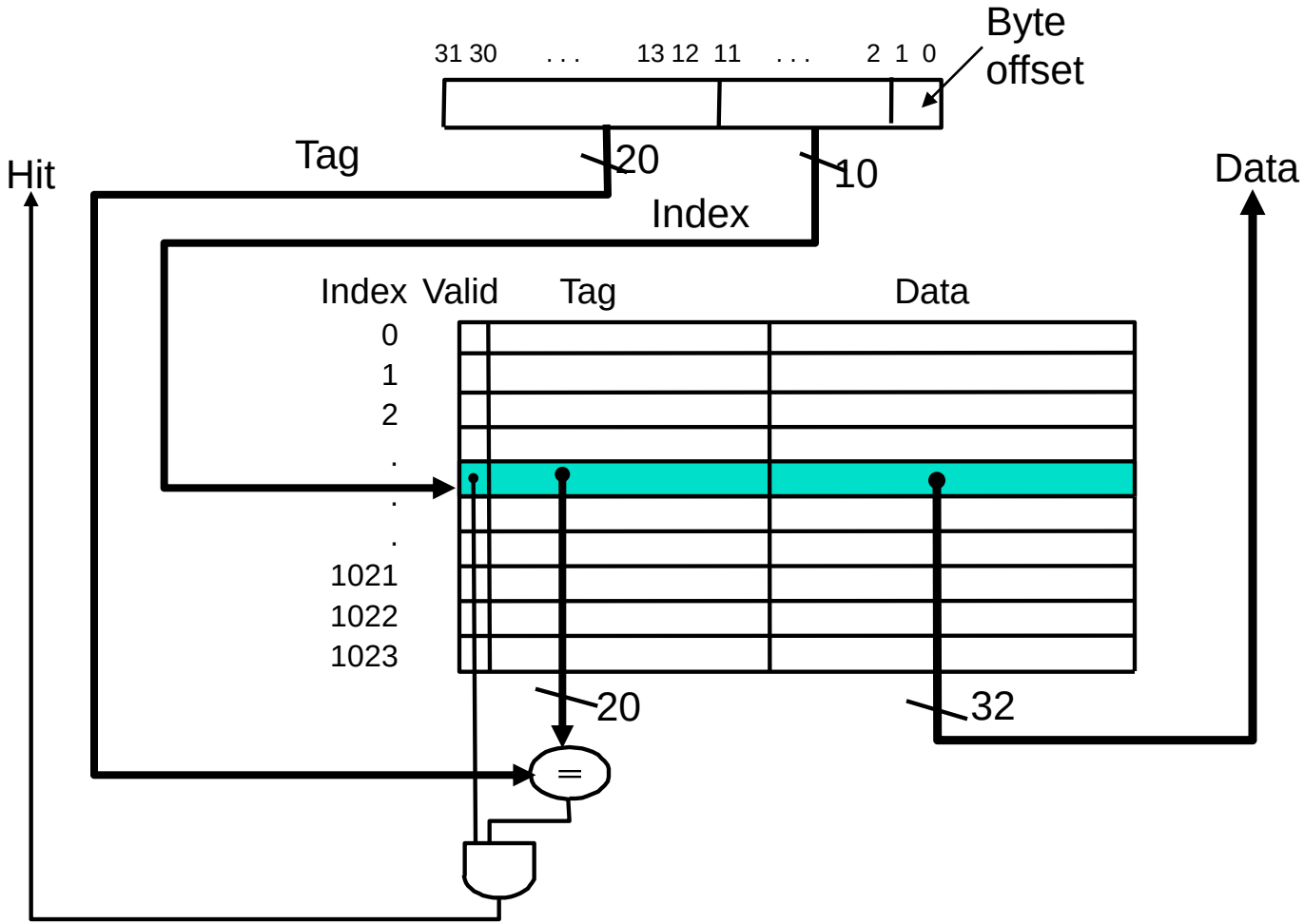
Bắt đầu với bộ đệm rỗng – tất cả các khối trong bộ đệm được đánh dấu không hợp lệ

0 1 2 3 4 3 4 15



Ví dụ 4.2. Bộ đệm ánh xạ trực tiếp MIPS

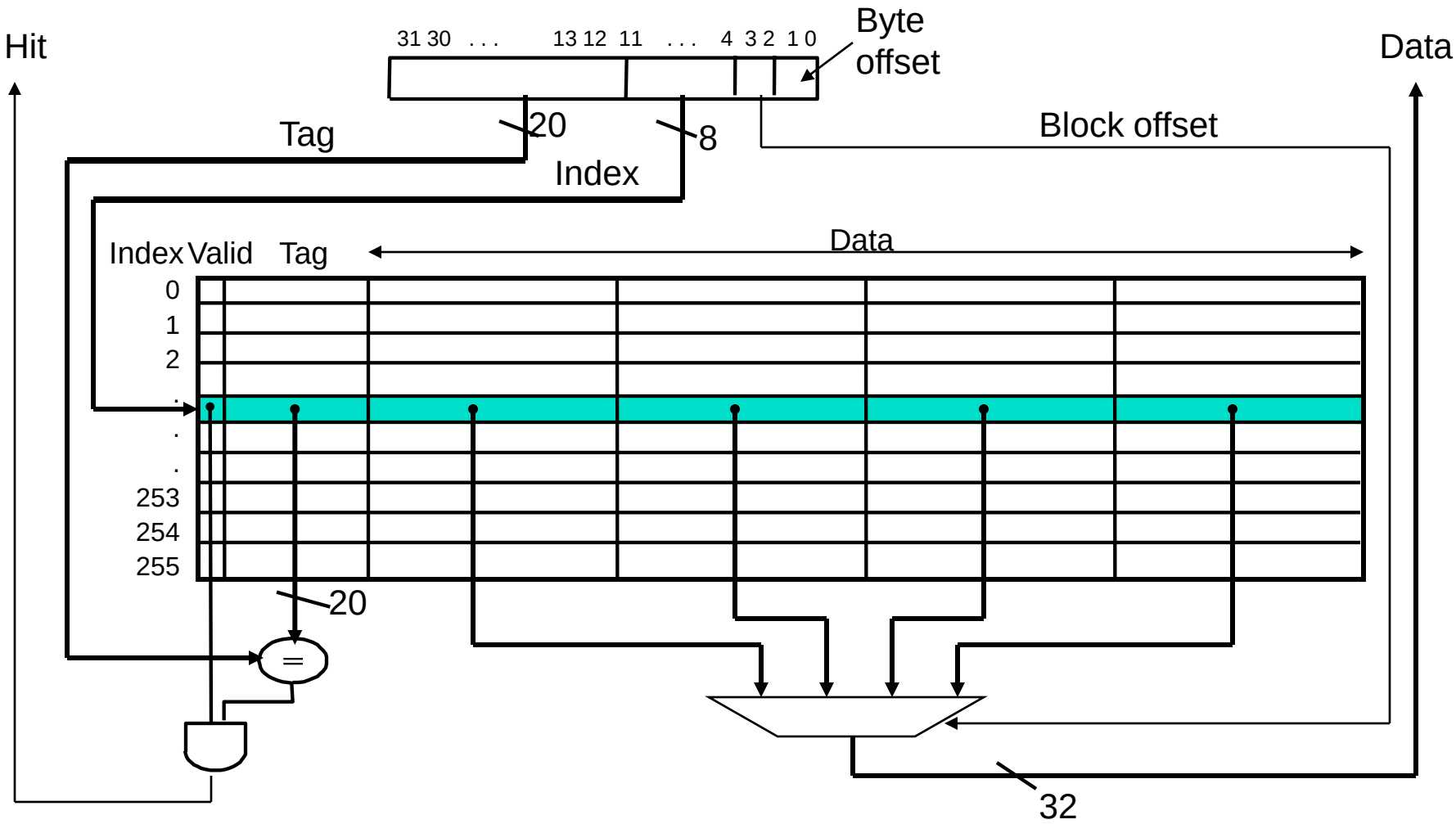
- Các khối 1 từ, kích thước bộ đệm = 1K từ (hay 4KB)



Tính cục bộ nào sẽ được tận dụng?

Bộ đệm ánh xạ trực tiếp khối nhiều từ

- Khối 4 từ, Kích thước bộ đệm = 1K words



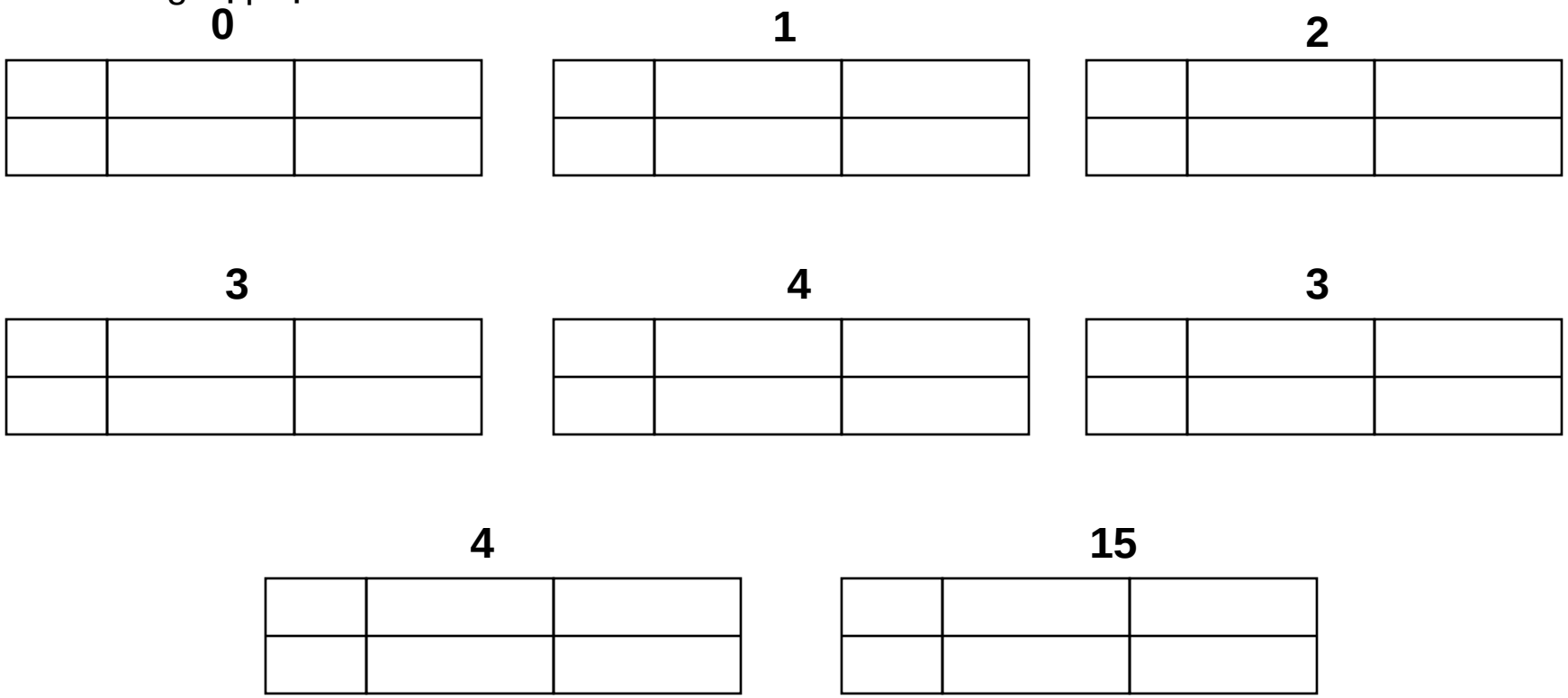
Tính cục bộ nào sẽ được tận dụng?

Tận dụng tính cục bộ không gian

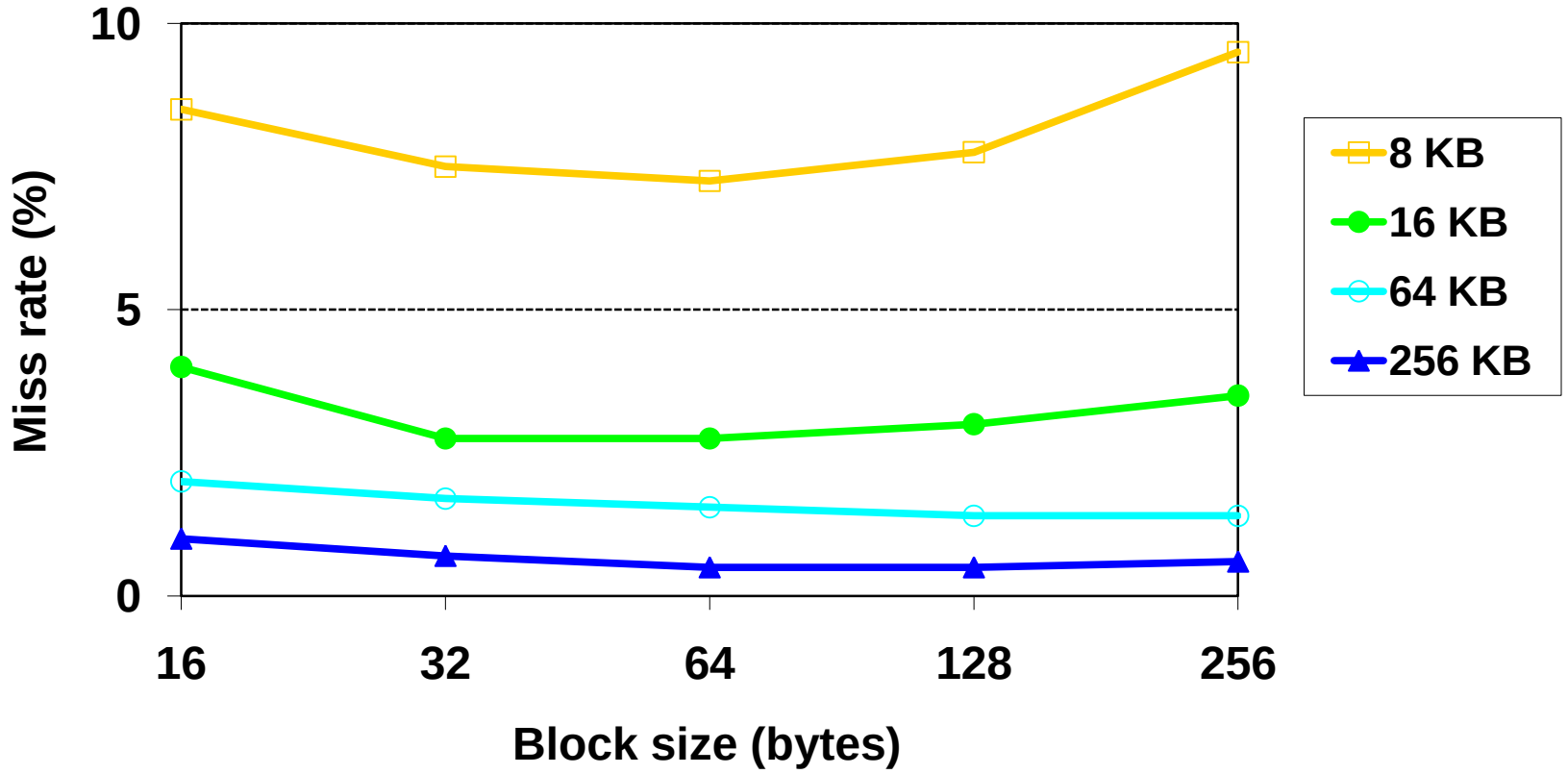
- Các khối trong bộ đệm chứa hơn 1 từ

Bắt đầu với bộ đệm rỗng – tất cả các khối trong bộ đệm được đánh dấu không hợp lệ

0 1 2 3 4 3 4 15



Tỉ lệ trượt vs Kích thước khối vs Kích thước bộ đệm



- Tỉ lệ trượt tăng khi kích thước khối trở nên đáng kể so với kích thước bộ đệm vì với cũng kích thước bộ đệm số khối có thể lưu giữ giảm (tăng trượt do dung lượng)
- Tăng kích thước khối làm tổn thất trượt

Kích thước các trường trong bộ đệm

- Số bit trong bộ đệm gồm bit cho dữ liệu và bit cho các trường thẻ
  - Địa chỉ byte 32 bit
  - Bộ đệm ánh xạ trực tiếp 2^n khối, *n* bits cho trường index
  - Kích thước khối là 2^m từ (2^{m+2} bytes), *m* bits cho trường block offset xác định vị trí từ trong khối; 2 bits cho trường byte offset xác định vị trí byte trong từ
- Kích thước trường tag sẽ là?
- Tổng số bít trong bộ đệm ánh xạ trực tiếp sẽ là

- Cần bao nhiêu bit cho bộ đệm ánh xạ trực tiếp kích thước 16KB dữ liệu, kích thước khối là 4 từ và dữ liệu được đánh địa chỉ bằng 32 bit?

Xử lý trúng bộ đệm

- Đọc trúng (I$ và D$)
  - Đó là điều ta cần!

- Ghi trúng (chỉ với D$)
  - yêu cầu bộ đệm và bộ nhớ phải thống nhất
    - luôn ghi dữ liệu vào cả khối bộ đệm và vào bộ nhớ ở mức kế tiếp (ghi xuyên - write-through)
    - ghi với tốc độ của bộ nhớ ở mức kế tiếp – chậm hơn! – sử dụng bộ đệm ghi (write buffer) và chỉ dừng khi bộ đệm ghi đầy
  - cho phép bộ đệm và bộ nhớ không thống nhất
    - chỉ ghi dữ liệu vào bộ đệm (ghi lại write-back khối bộ đệm vào bộ nhớ ở mức kế tiếp khi khối bộ đệm bị lấy lại)
    - cần 1 bít bẩn (dirty) cho mỗi khối bộ đệm để chỉ ra là khối đó cần được ghi lại vào bộ nhó khi nó bị lấy lại – có thể dùng bộ đệm ghi để tăng tốc việc ghi lại các khối bộ đệm bẩn

Xử lý trượt bộ đệm (Khối kích thước 1 từ)

- Đọc trượt (I$ và D$): mất thời gian *read_miss_penalty*
  - dừng đường ống, nạp khối từ bộ nhớ ở mức kế tiếp, đưa vào bộ đệm và gửi từ được yêu cầu tới bộ xử lý, tiếp tục đường ống
- Ghi trượt (D$) mất thời gian *write_miss_penalty và write_buffer_stalls*
  - Cấp phát và ghi – Đầu tiên đọc khối từ bộ nhớ và ghi từ vào khối

or

  - Không cấp phát và ghi– bỏ qua việc ghi vào bộ đệm; ghi từ vào bộ đệm ghi (tức là sẽ ghi vào bộ nhớ ở mức kết tiếp), không cần dừng nếu bộ đệm ghi không đầy

Đo hiệu năng bộ đệm

- Giả sử thời gian truy cập bộ nhớ khi trúng bộ đệm được bao gồm trong 1 chu kỳ thực hiện thông thường của CPU thì:

$$T_{cpu} = I \times CPI \times T_c$$

$$= I \times (\underbrace{CPI_{ideal} + MemStallC}_{CPI_{stall}}) \times T_c$$

- Số chu kỳ MemStallC là tổn thất trượt là tổng của read-stalls và write-stalls

- Với bộ đệm ghi xuyên, ta có công thức đơn giản

Ảnh hưởng của hiệu năng bộ đệm

- Tổn thất tương đối của bộ đệm sẽ tăng khi hiệu năng bộ xử lý tăng (tăng tốc độ đồng hồ và/hoặc giảm CPI)
  - Tốc độ bộ nhớ không được cải thiện nhanh nhu tốc độ bộ xử lý. Tổn thất trượt dùng để tính CPI_{stall} được đo theo số chu kỳ bộ xử lý cần thiết để xử lý trượt
  - CPI_{ideal} càng thấp thì ảnh hưởng của dừng do trượt càng lớn
- Bộ xử lý với CPI_{ideal} = 2, tổn thất trượt là 100, 36% là lệnh load/store, tỉ lệ trượt bộ nhớ I$ là 2% và bộ nhớ D$ là 4%

- Nếu CPI_{ideal} giảm xuống 1? 0.5? 0.25?
- Nếu tỉ lệ trượt bộ nhớ D$ tăng lên 1%? 2%?
- Nếu tốc độ đồng hồ CPU tăng gấp 2 (tổn hao trượt tăng gấp 2)?

Thời gian truy cập bộ nhớ trung bình (AMAT)

- Bộ đệm lớn sẽ có thời gian truy cập lớn. ➔ Làm tăng thời gian truy cập khi trúng ➔ cần 1 giai đoạn pipeline.
- Khi tăng kích thước bộ đệm

  ➔ cải tiến tỉ lệ trúng nhưng làm tăng thời gian truy cập trúng

  ➔ sẽ đến điểm mà thời gian truy cập bộ đệm lớn sẽ vượt qua cải tiến do tăng tỉ lệ trúng ➔ làm giảm hiệu năng
- Thời gian truy cập bộ nhớ trung bình (*Average Memory Access Time - AMAT*) là thời gian truy cập bộ nhớ khi tính cả 2 trường hợp trúng và trượt bộ đệm

- Tính AMAT cho 1 bộ xử lý có chu kỳ đồng hồ 20 psec, tổn thất trượt 50 chu kỳ, tỉ lệ trượt 0.02/1 lệnh và thời gian truy cập bộ đệm 1 chu kỳ?

Nguyên nhân trượt bộ đệm

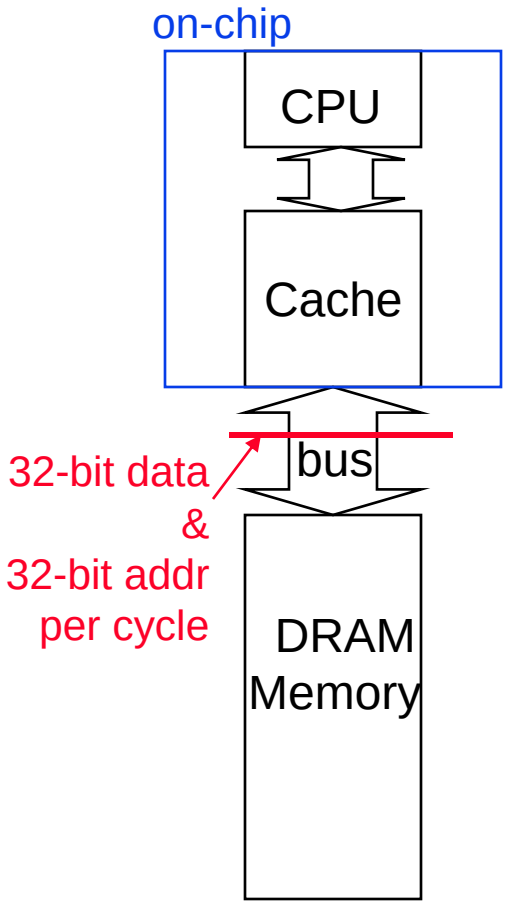
- Không tránh được:
  - Lần đầu truy cập khối
  - Giải pháp: tăng kích thước khối (làm tăng tổn thất trượt, khối rất lớn làm tăng tỉ lệ trượt)
- Dung lượng:
  - Bộ đệm không thể chứa toàn bộ các khối truy cập bởi chương trình
  - Giải pháp: tăng kích thước bộ đệm (có thể làm tăng thời gian truy cập)
- Xung đột:
  - Nhiều vị trí bộ nhớ cùng được ánh xạ vào 1 vị trí bộ đệm
  - Giải pháp 1: tăng kích thước bộ đệm
  - Giải pháp 2: tăng độ kết hợp trong bộ đệm (có thể tăng thời gian truy cập)

Yêu cầu với hệ thống bộ nhớ

- Tương thích với các đặc điểm của bộ đệm
  - bộ đệm truy cập 1 khối mỗi lần (nhiều hơn 1 từ)
- Tương thích với đặc điểm của DRAM
  - Sử dụng DRAM hỗ trợ truy cập nhanh nhiều từ, ưu tiên các DRAM tương thích với kích thước khối của bộ đệm
- Tương thích với đặc điểm của bus bộ nhớ
  - Bus bộ nhớ phải hỗ trợ được tốc độ truy cập DRAM và cách truy cập
  - Cho phép tăng băng thông giữa bus bộ nhớ và bộ đệm

Hệ thống bộ nhớ hỗ trợ bộ đệm

- Kết nối bên ngoài chip và kiến trúc bộ nhớ ảnh hưởng đến hiệu năng tổng thể của hệ thống rất nhiều

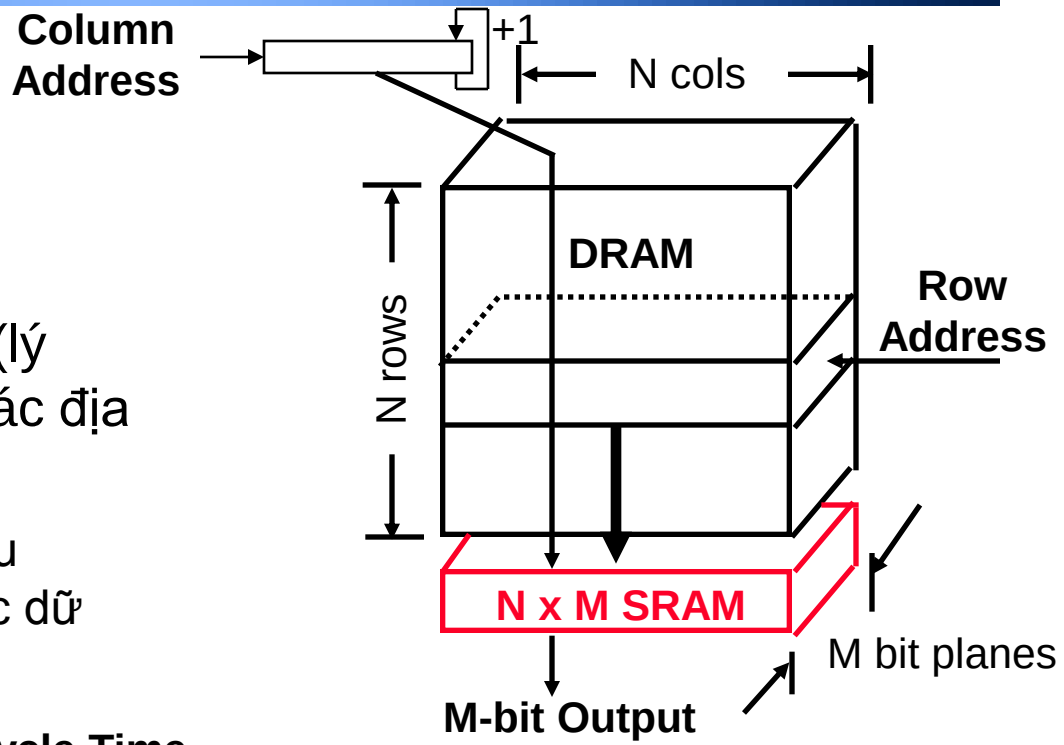


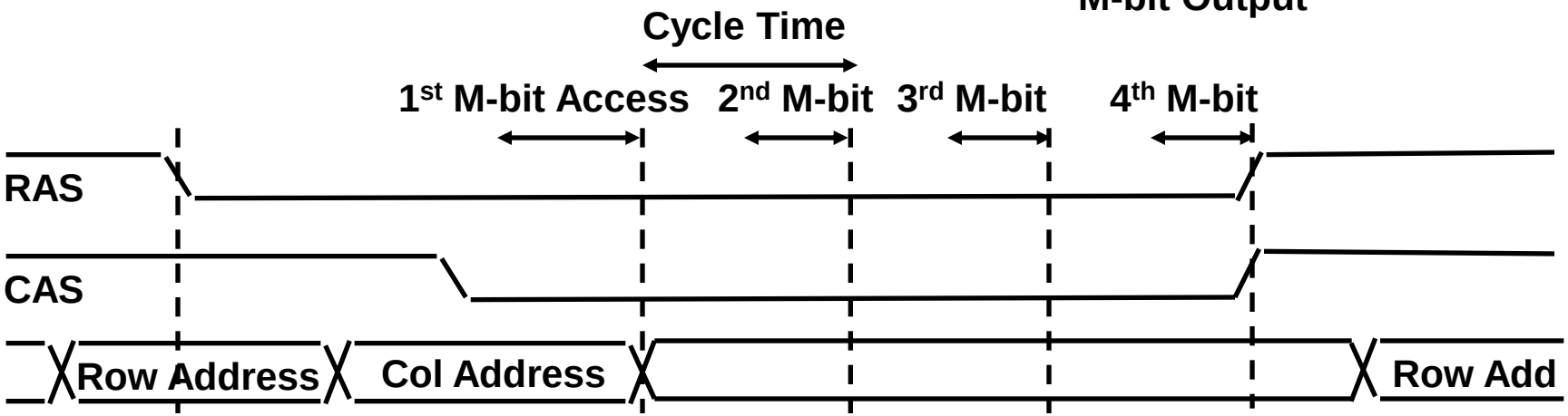
Cấu trúc 1 từ (bus rộng 1 từ và bộ nhớ có các ô nhớ 1 từ)

- Giả sử
  1. 1 chu kỳ bus bộ nhớ dùng để gửi địa chỉ addr
  2. 15 chu kỳ để đọc từ thứ nhất trong khối từ DRAM (thời gian chu kỳ 1 dòng), 5 chu kỳ cho các từ thứ 2, 3, 4 (thời gian truy cập cột)
  3. 1 chu kỳ để trả về 1 từ dữ liệu
- Băng thông từ bus bộ nhớ đến bộ đệm
  - số byte truy cập từ bộ nhớ và được truyền đến bộ đệm/CPU trong mỗi chu kỳ bus

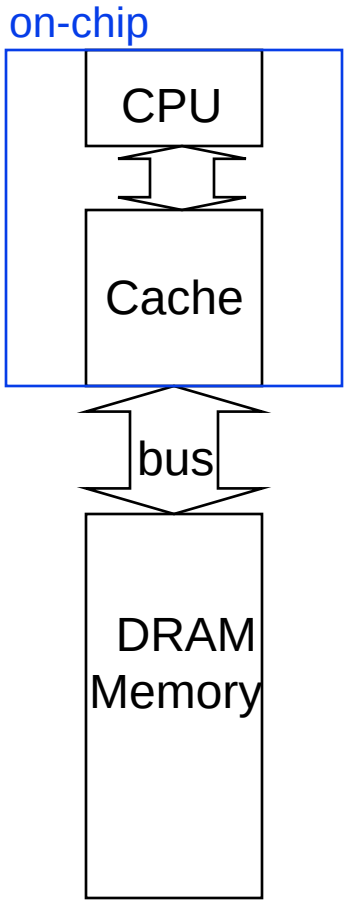
Hoạt động của (DDR) SDRAM

- Truy cập dòng: đọc vào thanh ghi SRAM
- Truy cập cột
  - Chuyển 1 chuỗi các dữ liệu (lý tưởng là 1 khối bộ đệm) ở các địa chỉ liên tiếp trong hàng
    - Đồng hồ bus bộ nhớ sẽ điều khiển việc chuyền chuỗi các dữ liệu





Bus rộng 1 từ; Khối 1 từ



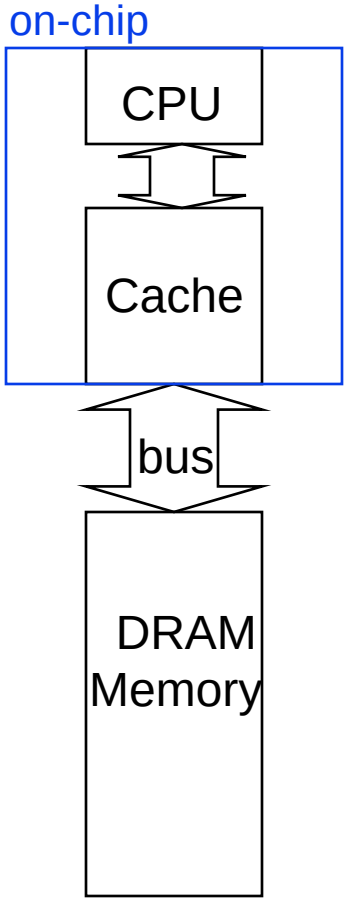
- Khi kích thước khối là 1 từ, truy cập bộ nhớ gây ra trượt bộ đệm sẽ gây ra dừng pipeline trong số chu kỳ cần để trả về 1 từ dữ liệu từ bộ nhớ

1	chu kỳ bus bộ nhớ để gửi địa chỉ
15	chu kỳ bus bộ nhớ để đọc hàng DRAM
1	chu kỳ bus bộ nhớ để trả về dữ liệu
17	tổng số chu kỳ tổn thất trượt

- Số byte chuyển trong 1 chu kỳ (băng thông) cho 1 lần trượt là

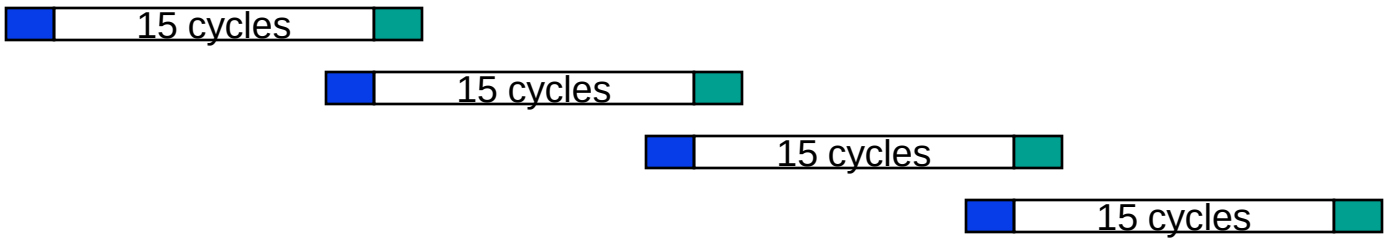
  4/17 = 0.235 byte/chu kỳ bus bộ nhớ

Bus rộng 1 từ; Khối 4 từ



- Khi các từ của khối nằm ở các hàng khác nhau?

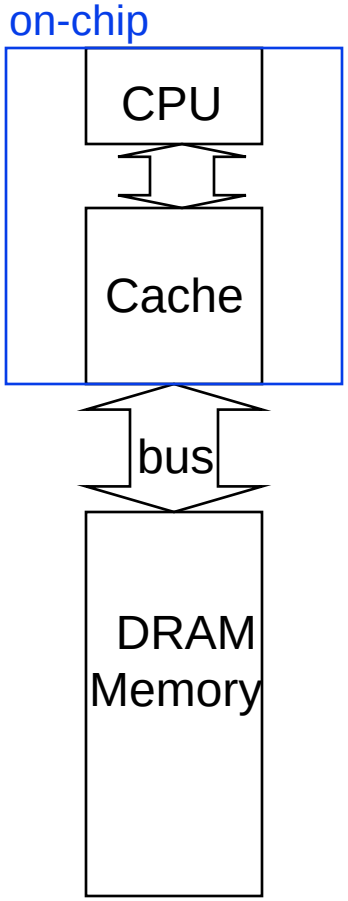
	1	chu kỳ gửi địa chỉ 1st
4 x 15 =	60	chu kỳ đọc từ hàng DRAM
	1	chu kỳ trả về từ cuối
	62	tổng chu kỳ tổn thất trượt



- Số byte chuyển trong 1 chu kỳ (băng thông) cho 1 lần trượt là

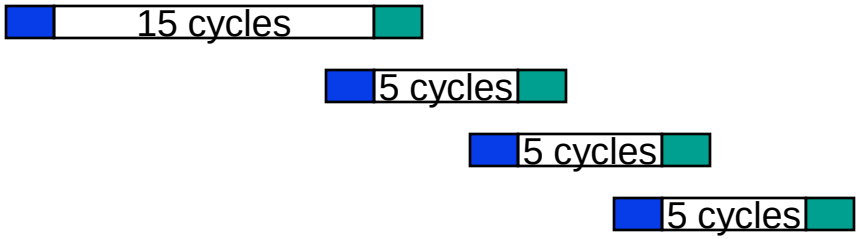
(4 x 4)/62 = 0.258 byte/chu kỳ

Bus rộng 1 từ; Khối 4 từ



- Khi các khối ở cùng 1 hàng?

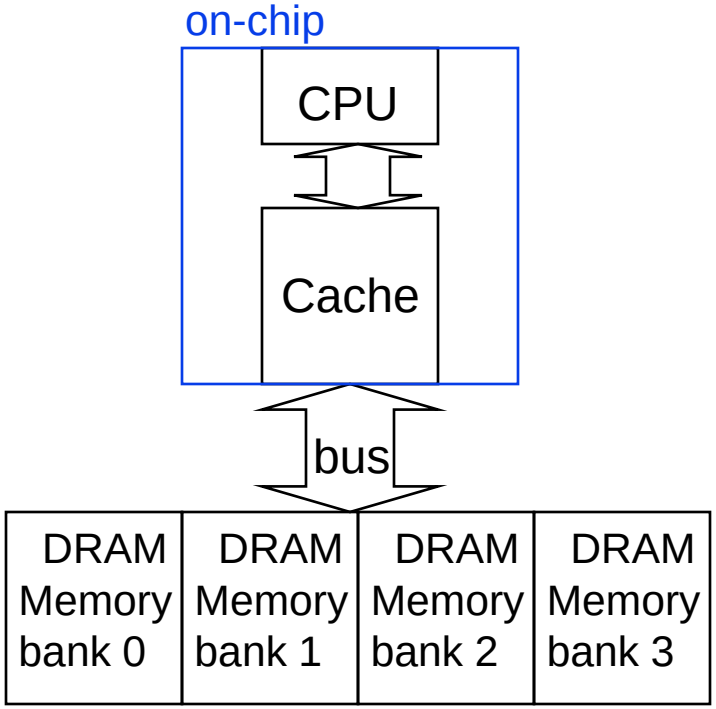
1	chu kỳ gửi địa chỉ 1st
15 + 3*5 = 30	chu kỳ đọc từ hàng DRAM
1	chu kỳ trả về từ cuối
32	tổng chu kỳ tổn thất trượt



- Số byte chuyển trong 1 chu kỳ (băng thông) cho 1 lần trượt là

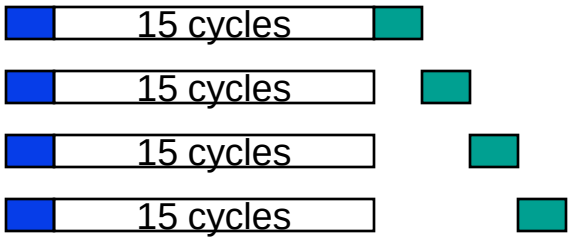
  (4 x 4)/32 = 0.5 byte/chu kỳ

Bộ nhớ xen kẽ; Bus độ rộng 4 từ



- 4 hàng trong các băng được đọc đồng thời

1	chu kỳ gửi địa chỉ thứ 1st
15	chu kỳ đọc các băng DRAM
4*1 = 4	chu kỳ trả về dữ liệu
20	tổng chu kỳ tổn thất trượt



- Số byte chuyển trong 1 chu kỳ (băng thông) cho 1 lần trượt là

(4 x 4)/20 = 0.8 byte/ chu kỳ

Giảm tỉ lệ trượt bộ đệm #1

1. Cho phép đặt các khối linh hoạt hơn

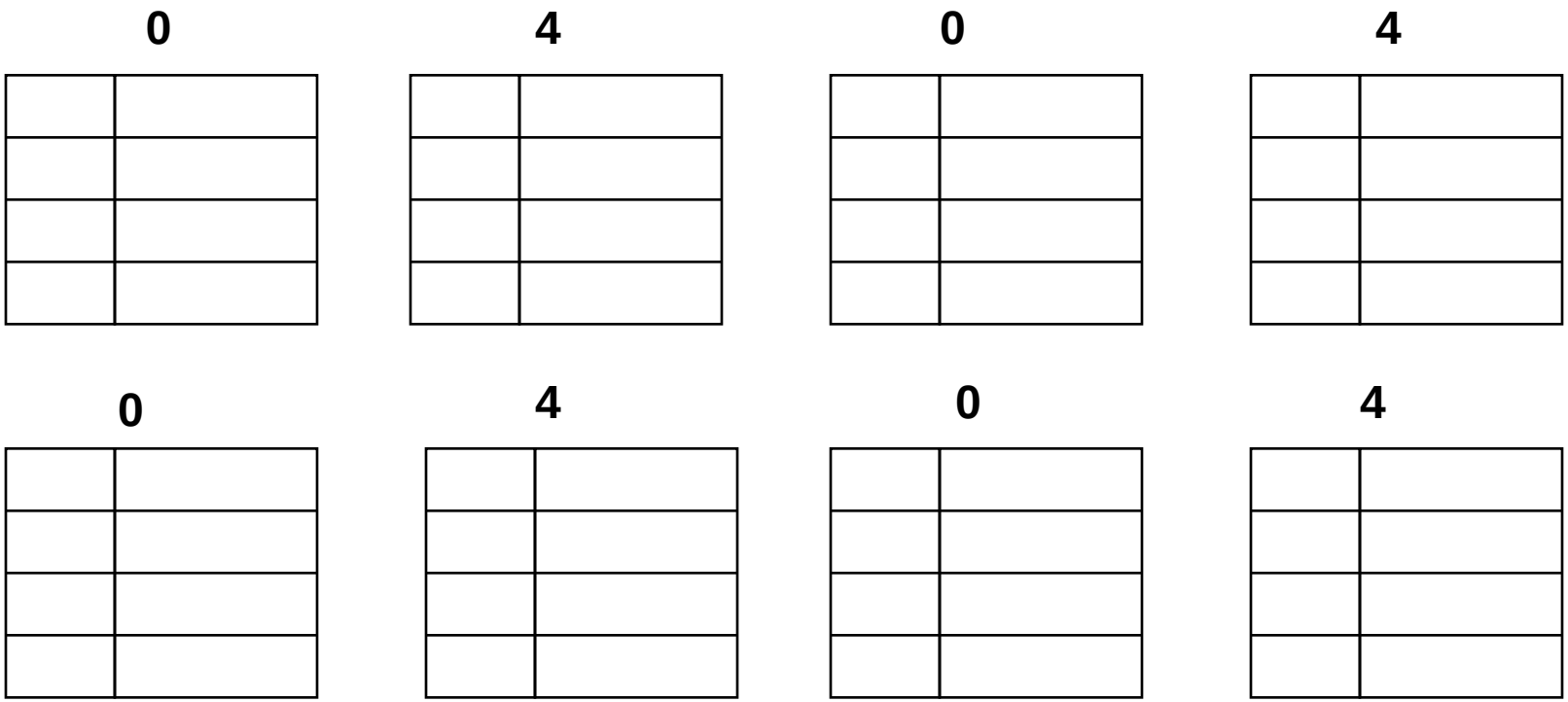
- Ở bộ đệm ánh xạ trực tiếp 1 khối bộ nhớ được ánh xạ vào chính xác 1 khối bộ đệm
- Ở 1 thái cực khác, 1 khối bộ nhớ có thể được ánh xạ vào *bất cứ* khối bộ đệm nào – bộ đệm kết hợp toàn phần (*fully associative cache)*
- Cách thỏa hiệp: chia bộ đệm thành các tập (*sets*); mỗi tập gồm n đường (kết hợp n đường - n-way set associative). Mỗi khối bộ nhớ được ánh xạ vào 1 tập duy nhất (xác định bằng trường index) và có thể được đặt vào đường bất kỳ trong tập (có n lựa chọn)

index = (block address) modulo (# sets in the cache)

Truy cập ô nhớ trong bộ đệm ánh xạ trực tiếp

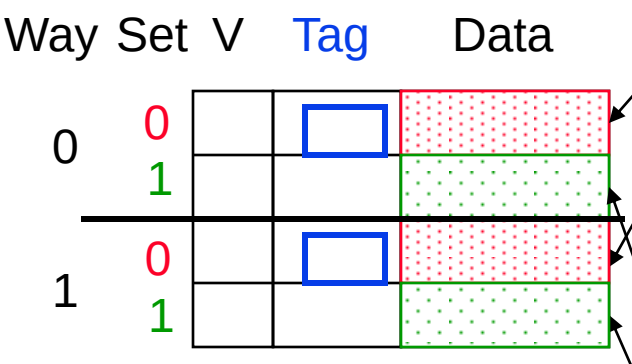
- Giả sử truy cập vào các ô nhớ 0 4 0 4 0 4 0 4

Bắt đầu với bộ đệm rỗng – tất cả các khối được đánh dấu không hợp lệ



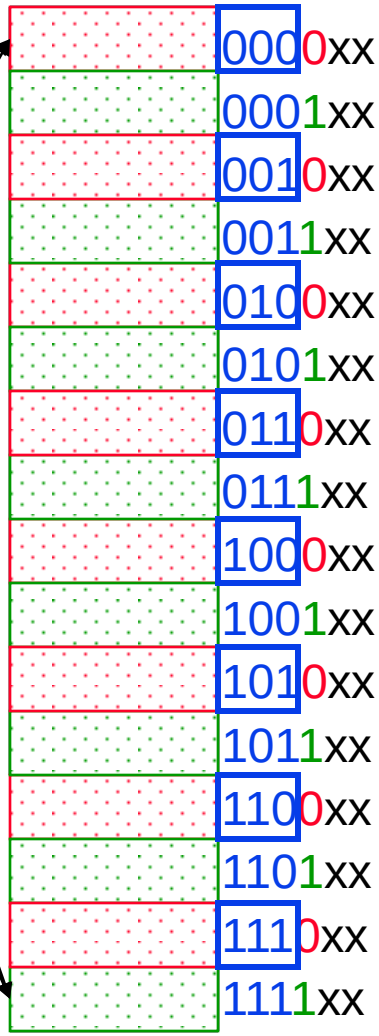
Bộ đệm kết hợp n đường

Bộ đệm: 4 khối, 2 tập



Bộ nhớ chính 16 khối 1 từ

Khối 1 từ, 2 bit thấp cuối dùng để xác định byte trong từ (từ 32b)



Q1: Có trong bộ đệm không?

So sánh *tất cả* các thẻ trong tập với 3 bít địa chỉ cao

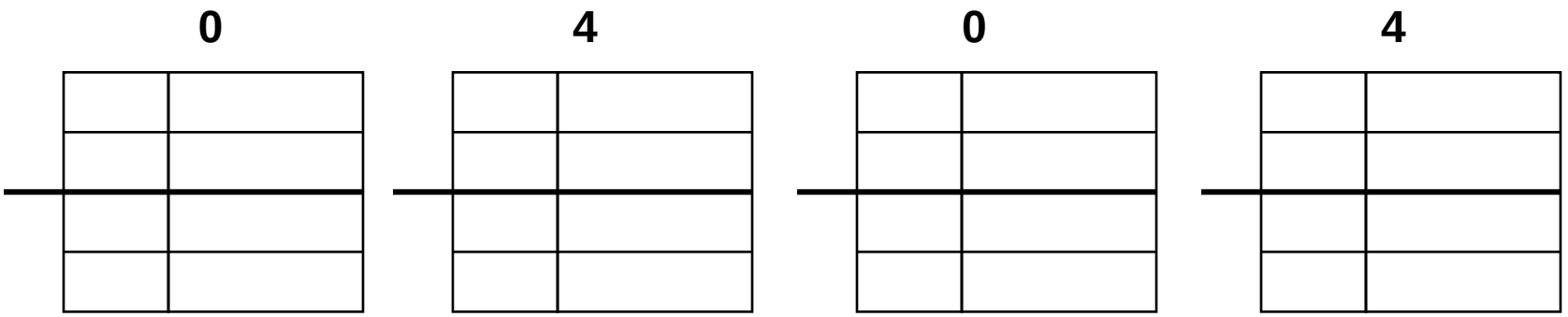
Q2: Vị trí từ trong bộ đệm?

Sử dụng bít thấp tiếp theo để xác định tập (i.e., chia lấy phần dư cho số tập trong bộ đệm)

Truy cập ô nhớ trong bộ đệm kết hợp 2 đường

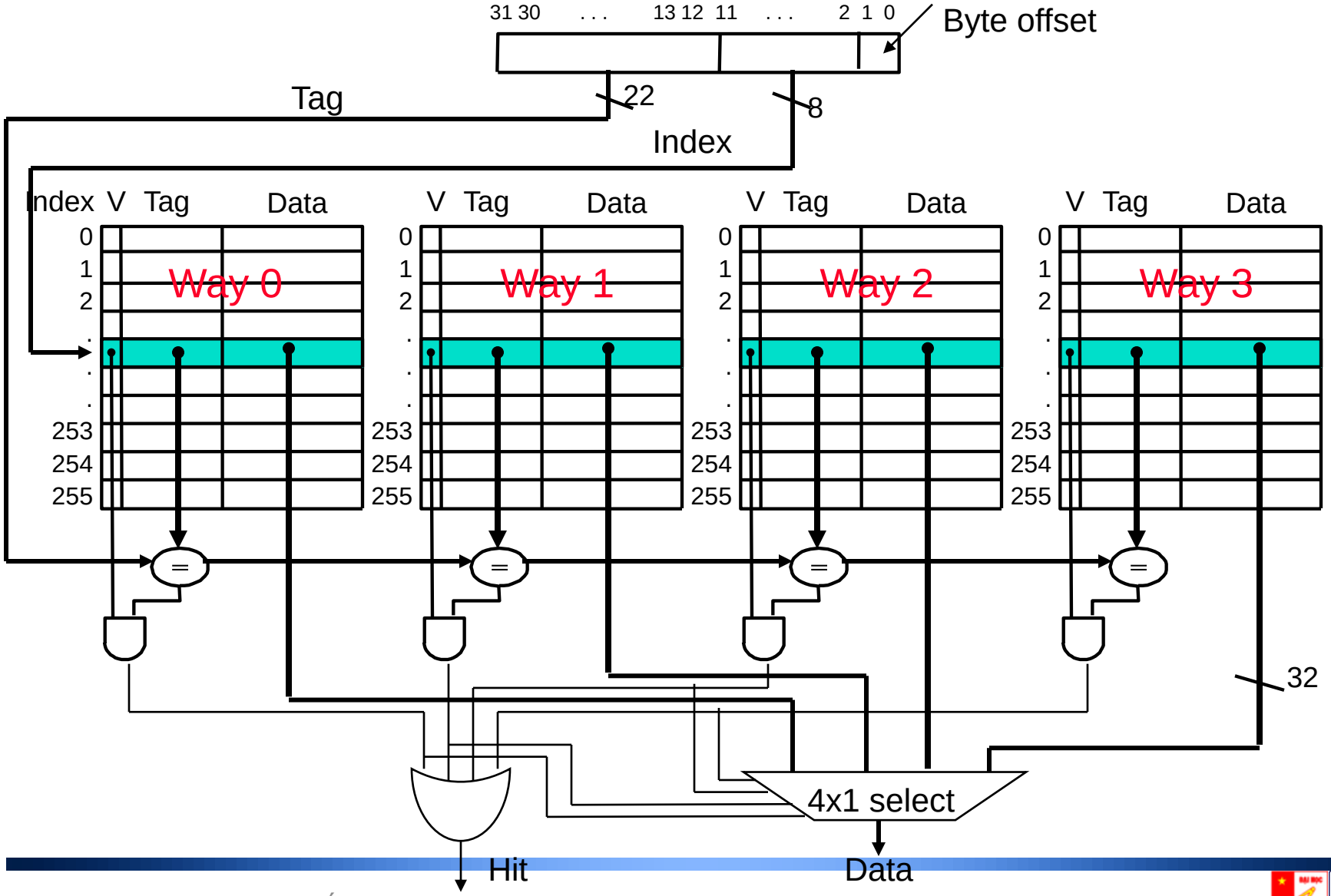
- Giả sử truy cập vào các ô nhớ 0 4 0 4 0 4 0 4

Bắt đầu với bộ đêm rỗng – tất cả các khối được đánh dấu không hợp lệ



Bộ đệm kết hợp 4 đường

- 2^8 = 256 tập 4 đường (mỗi đường chứa 1 khối)



Bố trí bộ đệm kết hợp

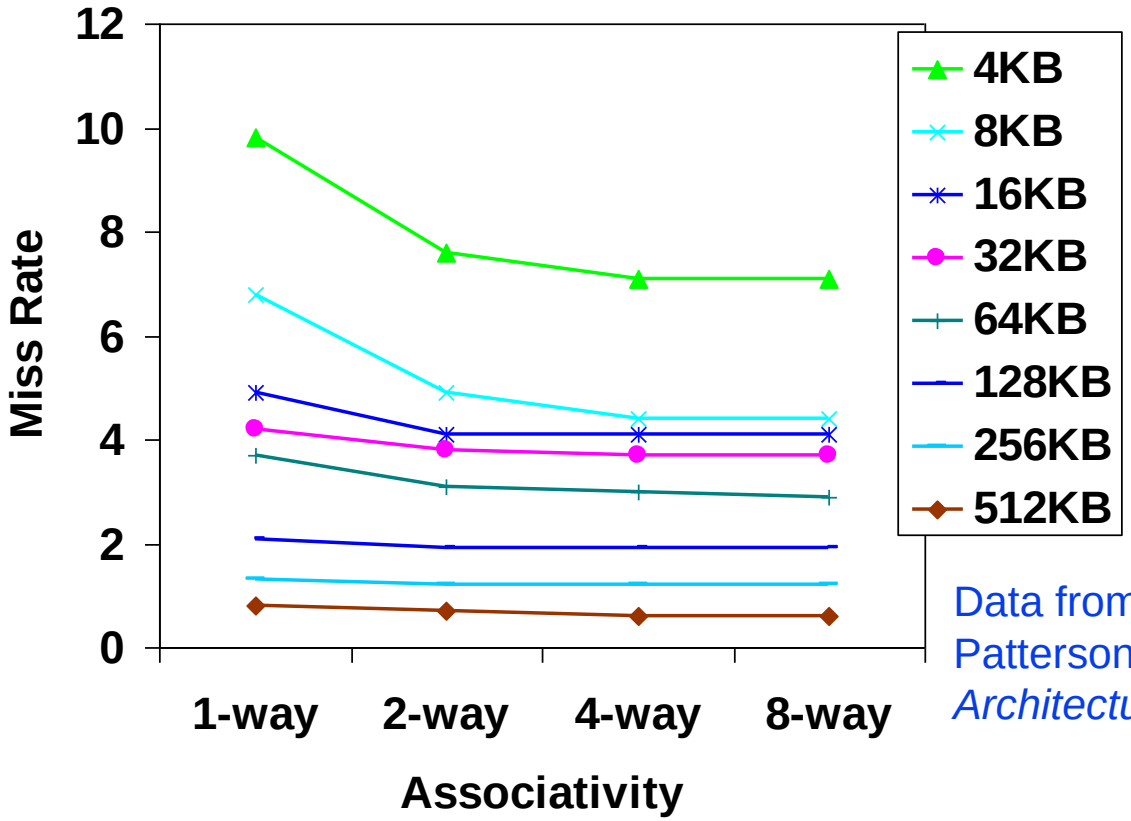
- Với kích thước bộ đệm cố định, tăng độ kết hợp theo hệ số 2 sẽ tăng số khối trong mỗi tập (tăng số đường) và giảm số tập – giảm kích thước trường index 1 bít và tăng kích thước trường tag 1 bit

Giá thành của bộ đệm kết hợp

- Khi xuất hiện trượt, đường (khối) nào sẽ bị thay thế?
  - Least Recently Used (LRU): khối bị thay thế là khối không được sử dụng trong thời gian dài nhất
    - Cần phần cứng để theo dõi khối được sử dụng khi nào so với các khối khác trong cùng tập
    - Với kết hợp 2 đường, dùng một bit cho mỗi tập → đặt bit khi một khối được truy cập
- Giá thành bộ đệm kết hợp N đường
  - N khối so sánh (trễ và diện tích)
  - Trễ khối MUX (chọn tập) trước khi dữ liệu sẵn sàng
  - Dữ liệu sẵn sàng sau khi chọn tập (và quyết định Hit/Miss). Trong bộ đệm trực tiếp, khối bộ đệm sẵn sàng trước khi quyết định Hit/Miss
    - Không thể giả sử là trúng để tiếp tục và sau đó khôi phục nếu là trượt

Lợi ích của bộ đệm kết hợp

- Lựa trọn giữa bộ đệm kết hợp và bộ đệm trực tiếp phụ thuộc vào tổn thất trượt và giá thành triển khai



Data from Hennessy & Patterson, *Computer Architecture*, 2003

- Lợi ích lớn nhất là khi chuyển từ bộ đệm trực tiếp sang kết hợp 2 đường (tỉ lệ trượt giảm 20%+)

Giảm tỉ lệ trượt #2

2. Sử dụng bộ đệm đa mức

- Mạch tích hợp ngày nay có thể chứa được bộ đệm mức 1 (L1 cache) lớn hơn hoặc bộ đệm mức 2 thống nhất (i.e., nó chứa cả chương trình và dữ liệu); và thậm chí cả bộ đệm L3 thống nhất
- Ví dụ:
  - $CPI_{ideal} = 2$
  - Tổn thất trượt = 100 chu kỳ (truy cập bộ nhớ chính)
  - Tổn thất trượt truy cập UL2$ = 25 chu kỳ
  - 36% load/stores
  - Tỉ lệ trượt: L1-I$ = 2%, L1-D$ = 4%, UL2$ = 0.5% (tỉ lệ trượt toàn cục)

Thiết kế bộ đệm đa mức

- Bộ đệm L1 và L2 được thiết kế rất khác nhau
  - Bộ đệm cơ sở tập trung vào tối thiểu hóa thời gian truy cập khi trúng để hỗ trợ chu kỳ ngắn hơn
    - Nhỏ hơn và có khối kích thước nhỏ hơn
  - Bộ đệm mức 2 tập trung vào giảm tỉ lệ trượt để giảm tổn thất trượt do phải truy cập bộ nhớ chính
    - Lớn hơn với khối kích thước lớn hơn
    - Độ kết hợp cao hơn
- Tổn thất trượt bộ đệm L1 được giảm rất nhiều khi có bộ đệm L2 – vì thế nó có thể nhỏ hơn (nhanh hơn) nhưng có tỉ lệ trượt cao hơn
- Với bộ đệm L2, thời gian truy cập khi trúng không quan trọng bằng tỉ lệ trượt
  - Thời gian truy cập trúng L2$ xác định tổn thất trượt L1$

Two Machines' Cache Parameters

	Intel Nehalem	AMD Barcelona
L1 cache organization & size	Split I$ and D$; 32KB for each per core; 64B blocks	Split I$ and D$; 64KB for each per core; 64B blocks
L1 associativity	4-way (I), 8-way (D) set assoc.; ~LRU replacement	2-way set assoc.; LRU replacement
L1 write policy	write-back, write-allocate	write-back, write-allocate
L2 cache organization & size	Unified; 256MB (0.25MB) per core; 64B blocks	Unified; 512KB (0.5MB) per core; 64B blocks
L2 associativity	8-way set assoc.; ~LRU	16-way set assoc.; ~LRU
L2 write policy	write-back	write-back
L2 write policy	write-back, write-allocate	write-back, write-allocate
L3 cache organization & size	Unified; 8192KB (8MB) shared by cores; 64B blocks	Unified; 2048KB (2MB) shared by cores; 64B blocks
L3 associativity	16-way set assoc.	32-way set assoc.; evict block shared by fewest cores
L3 write policy	write-back, write-allocate	write-back; write-allocate

Tổng kết: Cải thiện hiệu năng bộ đệm

0. Giảm thời gian truy cập trúng
   - Bộ đệm kích thước nhỏ
   - Bộ đệm ánh xạ trực tiếp
   - Khối kích thước nhỏ
   - Khi ghi
     - Không cấp phát và ghi – không truy cập trúng bộ đệm, chỉ ghi vào bộ đệm ghi
     - Cấp phát và ghi – để không cần 2 chu kỳ (1. kiểm tra trúng, 2. ghi), đường ống ghi sử dụng bộ đệm ghi trễ

1. Giảm tỉ lệ trượt
   - Bộ đệm kích thước lớn
   - Đặt khối linh hoạt hơn (tăng độ kết hợp)
   - Khối kích thước lớn (thông thường 16 đến 64 bytes)
   - Thêm bộ đệm “victim” – bộ đệm nhỏ lưu các khối vừa bị bỏ

Tổng kết: Cải thiện hiệu năng bộ đệm

2. Giảm tổn thất trượt

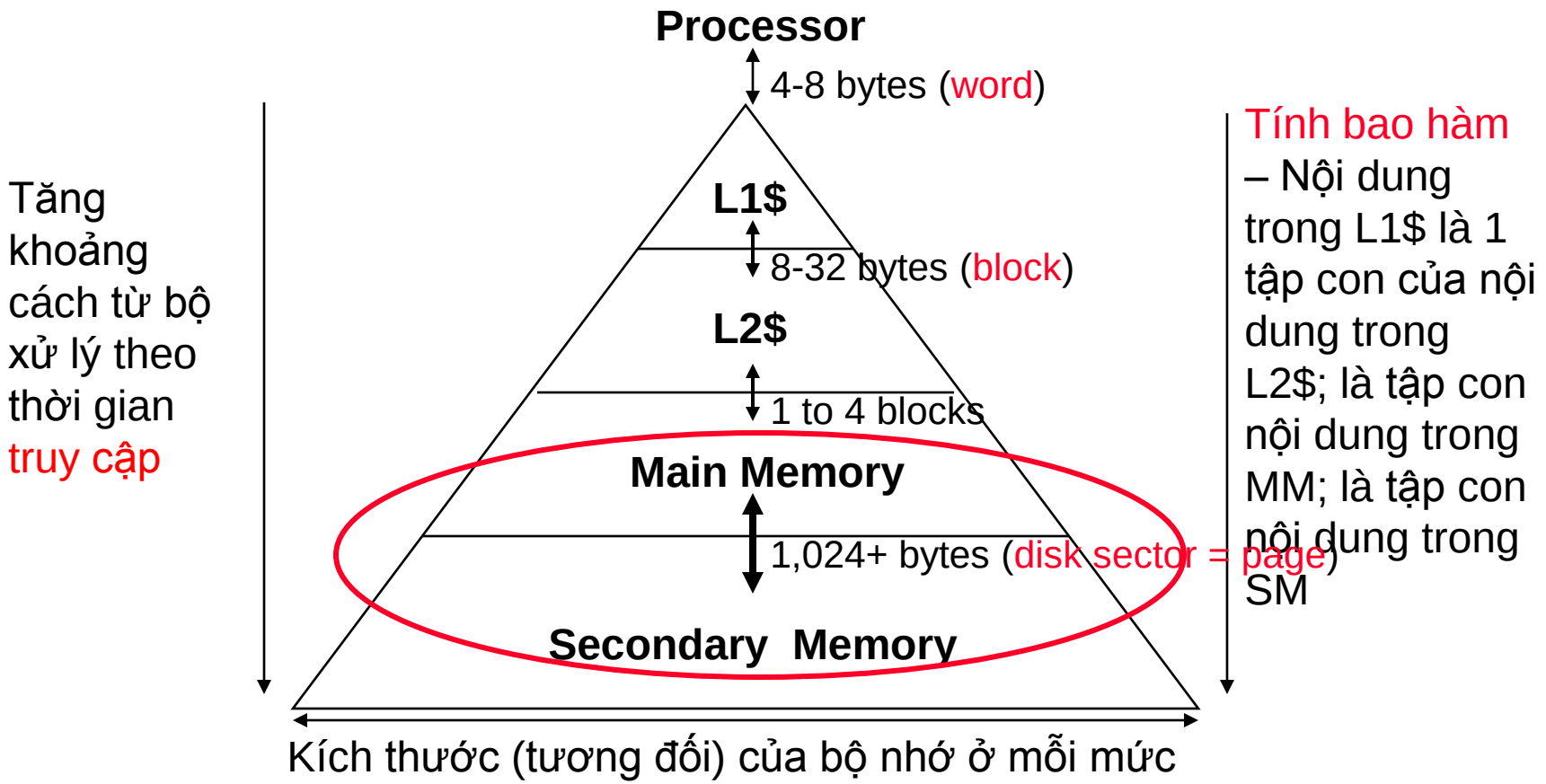
- Khối kích thước nhỏ
- Sử dụng bộ đệm ghi để lưu khối "dirty" (khối đã bị thay đổi-cần ghi vào bộ nhớ) ➔ không cần đợi kết thúc ghi trước khi đọc khối mới
- Kiểm tra bộ đệm ghi (và/hoặc bộ đệm "victim") trong trường hợp đọc trượt
- Với các khối lớn, nạp các từ quan trọng trước
- Sử dụng bộ đệm đa mức
- Tăng tốc độ và băng thông bộ nhớ
  - Bus rộng hơn
  - Bộ nhớ xen kẽ, DDR SDRAMs

Dịch chuyển dữ liệu giữa các mức bộ nhớ

- Thanh ghi ↔ Bộ nhớ
  - Trình biên dịch (người lập trình?)
- Bộ đệm ↔ bộ nhớ chính
  - Phần cứng điều khiển bộ đệm
- Bộ nhớ chính ↔ Đĩa
  - Hệ điều hành (bộ nhớ ảo)
  - Ánh xạ địa chỉ ảo và địa chỉ vật lý nhờ phần cứng (Translation Lookaside Buffer)
  - Người lập trình (các tệp)

Review: The Memory Hierarchy

- Tập dụng nguyên tắc "cục bô" để cung cấp cho người dùng kích thước bộ nhớ lớn như công nghệ bộ nhớ rẻ rất nhưng ở tốc độ cao như công nghệ bộ nhớ nhanh nhất

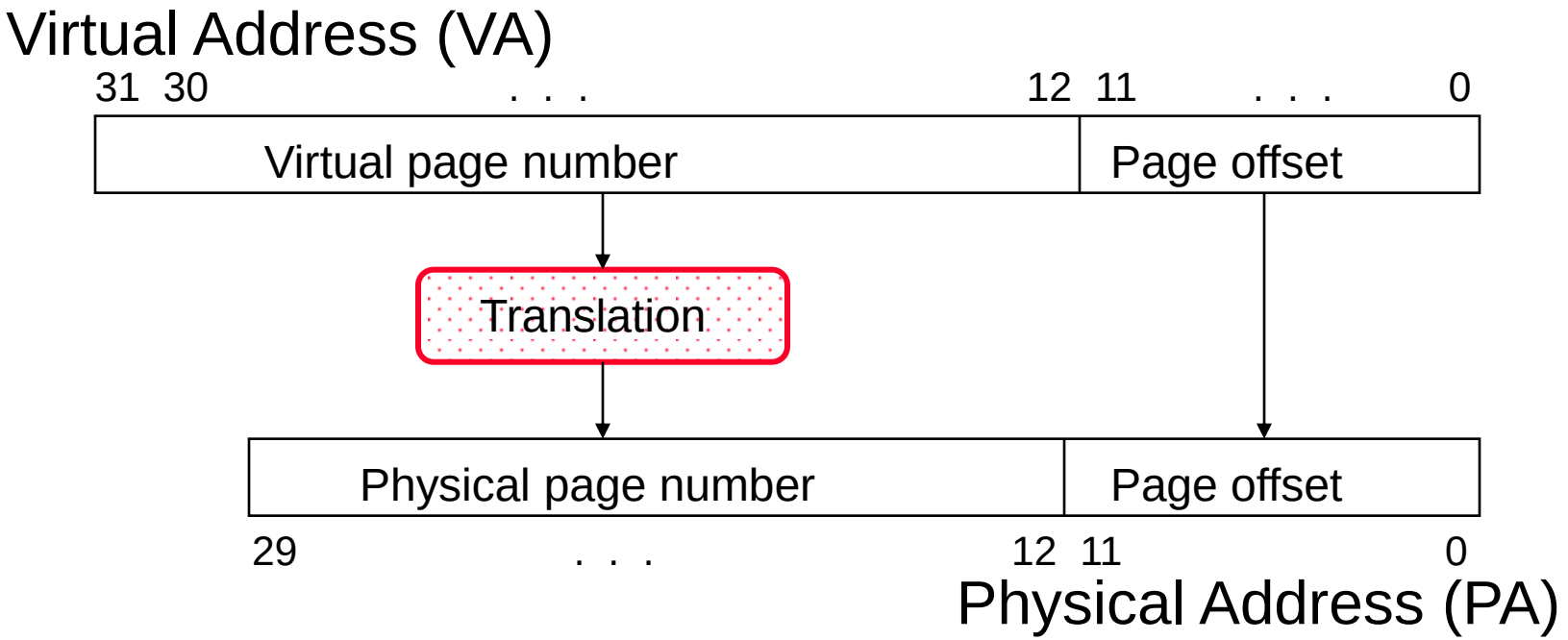


Bộ nhớ ảo

- Sử dụng bộ nhớ chính như "bộ đệm" cho bộ nhớ thứ cấp
  - Cho phép chia sẻ an toàn và hiệu quả bộ nhớ giữa các chương trình
  - Cho phép chạy chương trình lớn hơn kích thước bộ nhớ vật lý
  - Đơn giản hóa việc nạp chương trình để chạy (i.e., mã chương trình có thể được đưa vào bất kỳ chỗ nào trong bộ nhớ chính)
- Tại sao có hiệu quả? – Tính cục bộ
  - 1 chương trình thường truy cập vào một không gian địa chỉ nhỏ tại trong 1 khoảng thời gian
- Mỗi chương trình sử dụng 1 không gian địa chỉ riêng biệt – không gian địa chỉ "ảo".
  - Trong thời gian chạy, địa chỉ ảo được dịch thành địa chỉ vật lý (địa chỉ bộ nhớ chính)
  - Không gian ảo địa chỉ của chương trình được chia thành các trang (kích thước cố định) hoặc các đoạn (kích thước thay đổi)

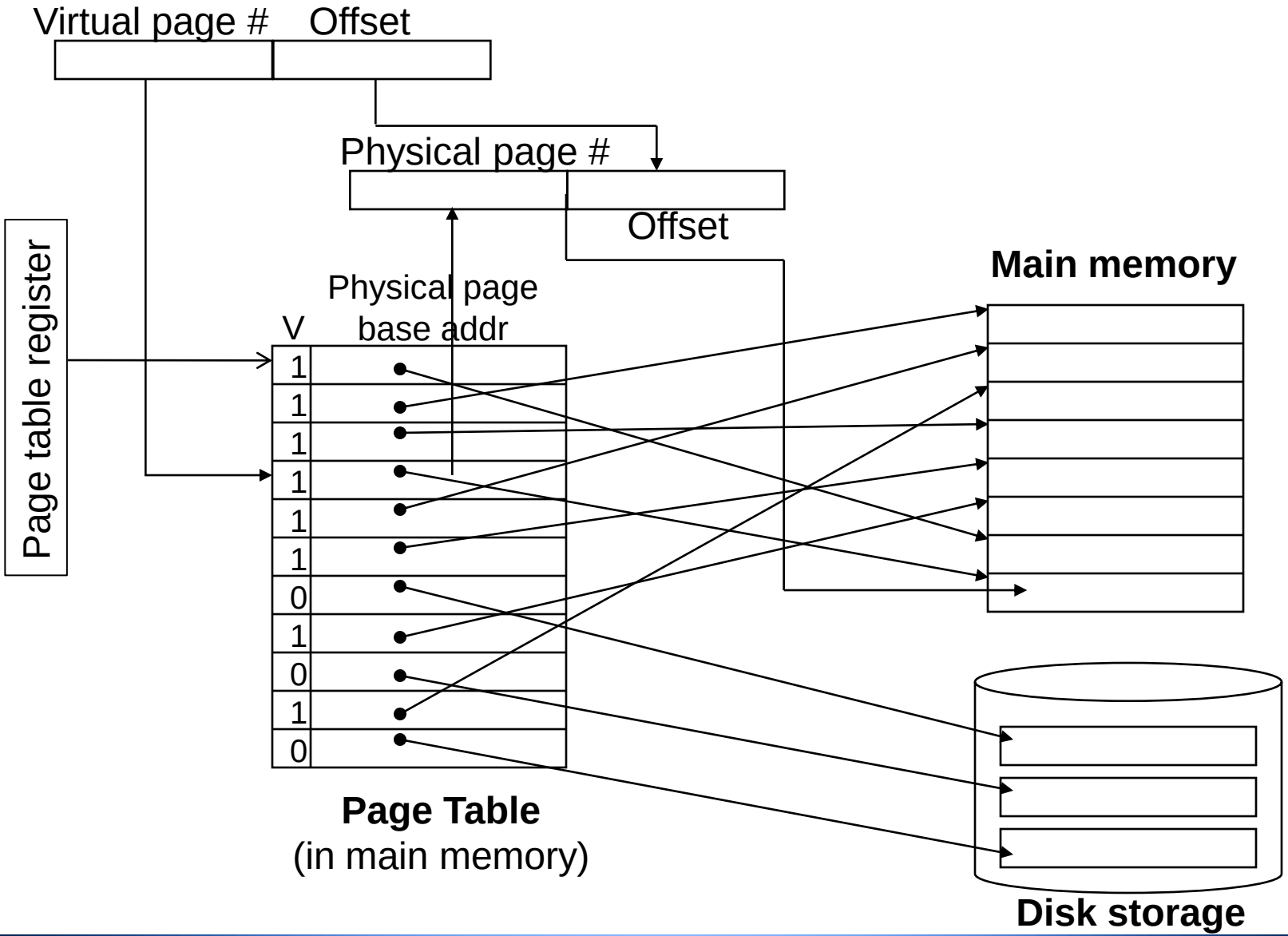
Dịch địa chỉ

- Một địa chỉ ảo được dịch thành 1 địa chỉ vật lý bằng cả phần cứng và phần mềm



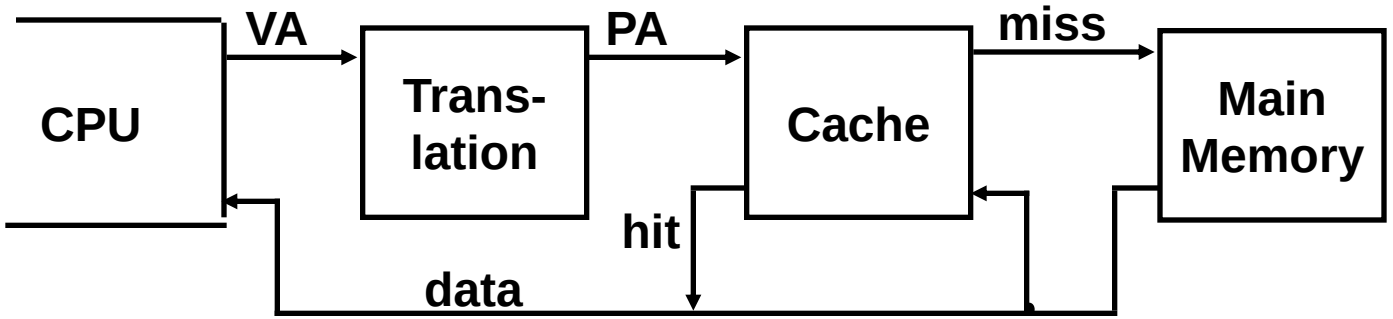
- Mỗi yêu cầu bộ nhớ, *đầu tiên* cần yêu cầu 1 sự dịch bộ nhớ từ không gian ảo thành không gian vật lý
  - Trượt bộ nhớ ảo (trang không có trong bộ nhớ vật lý) gọi là lỗi trang (*page fault*)

Nguyên lý dịch địa chỉ



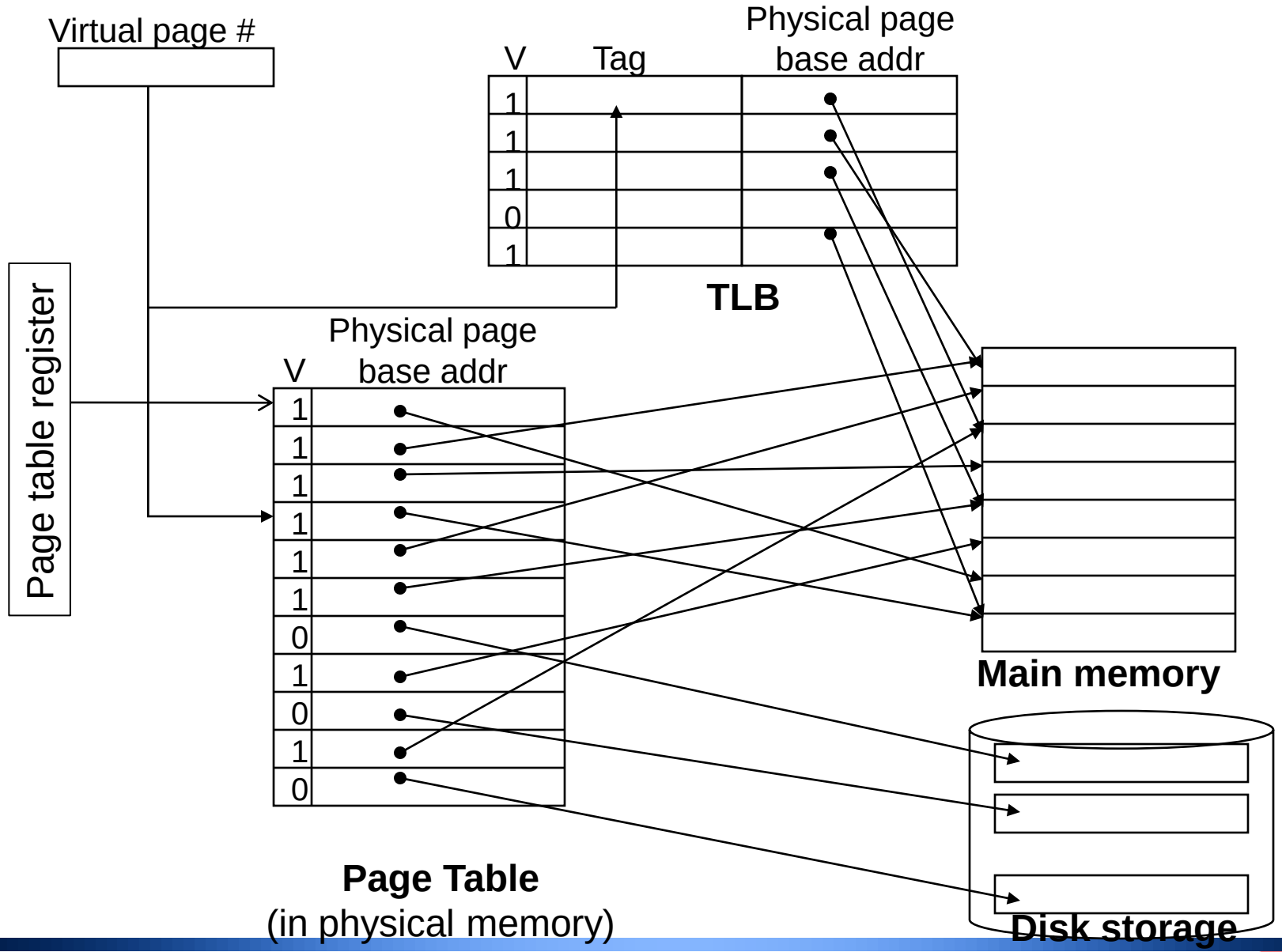
Địa chỉ ảo với bộ đệm

- Cần *thêm* 1 lần truy cập bộ nhớ để dịch địa chỉ ảo thành địa chỉ vật lý

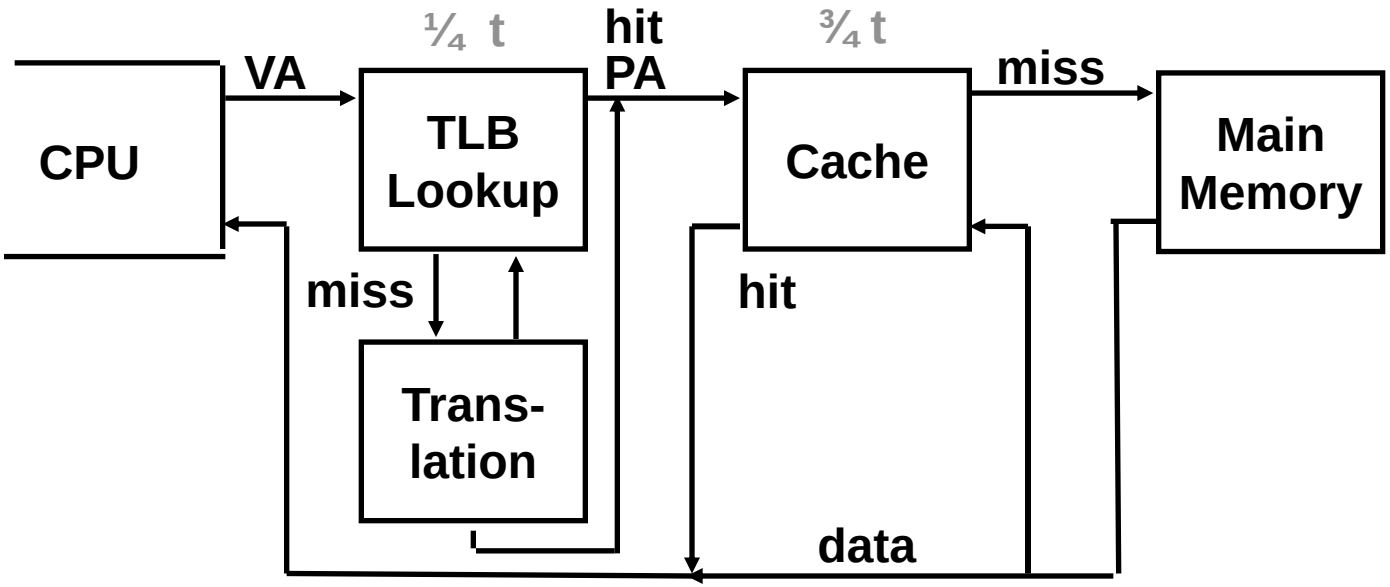


- Truy cập bộ nhớ (bộ đệm) rất tốn kém (mỗi lần truy cập thực chất là *hai* lần truy cập)
- Sử dụng bộ đệm nhắc vở (*Translation Lookaside Buffer TLB*) – một bộ đệm nhỏ lưu trữ các chuyển đổi địa chỉ vừa được sử dụng gần đây để trách việc tìm trong bảng trang

Tăng tốc dịch địa chỉ



TLB trong phân cấp bộ nhớ



Bốn câu hỏi trong phân cấp bộ đệm

- Q1: Một mục dữ liệu được đặt vào đâu trong mức cao hơn? *(Entry placement)*
- Q2: Một mục dữ liệu được tìm như thế nào trong mức cao hơn? *(Entry identification)*
- Q3: Thay thế mục nào khi có trượt? *(Entry replacement)*
- Q4: Làm gì khi ghi? *(Write strategy)*

Q1&Q2: Vị trí đặt/tìm một mục dữ liệu?

	Số tập	Số mục / 1 tập
Ánh xạ trực tiếp	Tổng số mục	1
Kết hợp đa đường	(Tổng số mục)/ Độ kết hợp	Độ kết hợp (thường từ 2 đến 16)
Kết hợp toàn phần	1	Tổng số mục

	Phương pháp tìm	Số bộ so sánh
Ánh xạ trực tiếp	Đánh chỉ số (index)	1
Kết hợp đa đường	Đánh chỉ số tập; So sánh thẻ của tập	Độ kết hợp
Kết hợp toàn phần	So sánh thẻ của tất cả các mục. Hoặc bảng (trang) tra cứu riêng	Tổng số mục 0

Q3: Thay thế mục nào khi có trượt

- Ánh xạ trực tiếp – duy nhất 1 lựa chọn – Luôn thay thế
- Kết hợp tập hoặc kết hợp toàn phần
  - Ngẫu nhiên
  - LRU (Least Recently Used): thay thế khối ít được sử dụng nhất trong thời gian dài nhất
- Với bộ đệm kết hợp 2 đường, phương pháp thay thế ngẫu nhiên có tỉ lệ trượt cao hơn 1,1 lần so với phương pháp LRU
- LRU có chi phí (phần cứng) cao khi áp dụng cho bộ đệm có độ kết hợp cao (> 4-đường) vì theo dõi thông tin sử dụng rất tốn kém

Q4: Làm gì khi ghi?

- *Write-through: Ghi xuyên* – Thông tin được ghi vào mục dữ liệu cả ở mức bộ nhớ hiện tại và mức bộ nhớ kế tiếp trong phân cấp bộ nhớ.
  - Luôn được kết hợp cùng bộ đệm ghi để loại bỏ thời gian chờ ghi vào bộ nhớ ở mức kế tiếp (cho đến khi bộ đệm ghi chưa đầy)
- *Write-back: Ghi sau* – Thông tin chỉ được ghi vào mục dữ liệu ở mức bộ nhớ hiện tại. Mục bị thay đổi được ghi vào mức bộ nhớ kế tiếp khi nó bị thay thế.
  - Cần bit "bẩn" để theo dõi 1 mục là bị thay đổi hay không
  - Hệ thống bộ nhớ ảo luôn dùng phương pháp ghi sau với các trang được đánh dấu "bẩn"
- Ưu nhược điểm?
  - Ghi xuyên: trượt khi đọc không gây ra việc ghi dữ liệu: đơn giản, rẻ và dễ triển khai
  - Ghi sau: ghi được cùng tốc độ của bộ đệm, ghi lặp lại cần 1 lần ghi vào bộ nhớ mức thấp

Tổng kết

- Nguyên lý cục bộ:
  - Chương trình có thể truy cập vào một phần khá nhỏ không gian địa chỉ tại 1 thời điểm.
    - Cục bộ thời gian - Temporal Locality
    - Cục bộ không gian - Spatial Locality
- Hiểu bộ đệm, TLBs, bộ nhớ ảo bằng cách nghiên cứu cách chúng xử lý 4 câu hỏi:
  1. Mục dữ liệu được đặt ở đâu?
  2. Mục dữ liệu được tìm như thế nào?
  3. Thay thế mục nào khi trượt?
  4. Thực hiện ghi như thế nào?
- Bảng trang ánh xạ địa chỉ ảo vào địa chỉ vật lý
  - TLBs dùng để thực hiện việc dịch nhanh