# Lec 17: Caches 

Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

## Announcements

- Prelim: graded
- PA 2: graded
- HW 2: graded
- HW 3 out tonight: cache simulation
- Recitations this week on C/Unix/etc.


## Cache Design 101



These are rough numbers: mileage may vary for latest/greatest Caches USUALLY made of SRAM

## Insight of Caches

- Exploit locality
- Two types: temporal and spatial
- Temporal locality
- If memory location $X$ is accessed, then it is more likely to be accessed again in the near future than some random location Y
- Caches exploit temporal locality by placing a memory element that has been referenced into the cache
- Spatial locality
- If memory location $X$ is accessed, then locations near $X$ are more likely to be accessed in the near future than some random location Y
- Caches exploit spatial locality by allocating a cache line of data (including data near the referenced location)


## Cache Lookups (Read)

- Look at address issued by processor
- Search cache to see if that block is in the cache
- Hit: Block is in the cache
- return requested data
- Miss: Block is not in the cache
- read line from memory
- evict an existing line from the cache
- place new line in cache
- return requested data


## Direct Mapped Cache



## Fully Associative Cache


© Kavita Bala, Computer Science, Cornell University

## Cache Organization

- Three common designs
- Fully associative: Block can be anywhere in the cache
- Direct mapped: Block can only be in one line in the cache
- Set-associative: Block can be in a few (2 to 8) places in the cache


## Eviction

- Which cache line should be evicted from the cache to make room for a new line?
- Direct-mapped
- no choice, must evict line selected by index
- Associative caches
- random: select one of the lines at random
- round-robin: similar to random
- FIFO: replace oldest line
- LRU: replace line that has not been used in the longest time


## Compromise

- Set-associative cache
- Like a direct-mapped cache
- Index into a location
- Fast
- Like a fully-associative cache
- Can store multiple entries
- decreases thrashing in cache
- Search in each element



## Direct Mapped


(c) Kavita Bala, Computer Science, CornellUniversity

| Fully AsSOC |  |  |
| :---: | :---: | :---: |
| Processor | Cache | Memory |
|  | 4 cache lines | 100 |
|  | $\underline{3}$ bit tag field | $1{ }^{1} 110$ |
|  | 2 word block | 2 120 |
| $\begin{array}{llll}\text { Ld R1 R M } & 1 & \\ \text { Ld R2 }\end{array}$ | tag data | 3 130 <br>  140 |
| Ld R3 $\leftarrow$ M[ 10 | 1 0 100 | 140 |
| Ld R3 $\leftarrow$ M[ 4 ] | 110 | 6160 |
|  | 1 2 140 | 170 |
| Ld R2 $\leftarrow$ M[ 5 ] | 150 | $8{ }^{8}$ |
|  | 6 220 | 190 |
|  | 230 | $10 \quad 200$ |
| Ld R2 $\leftarrow$ M[ 5 ] |  | 11 210 <br>   <br> 120  |
|  | Misses: 3 | $13-230$ |
|  | Hits: 8 | $14 \lcm{240}$ |
|  |  | $1 5 \longdiv { 2 5 0 }$ |

## 2 Way Set Assoc



## Cache Design

- Need to determine parameters
- Block size
- Number of ways of set-associativity
- Eviction policy
- Write policy
- Separate I-cache from D-cache


## Basic Cache Organization

Decide on the block size

- How? Simulate lots of different block sizes and see which one gives the best performance
- Most systems use a block size between 32 bytes and 128 bytes



## Tradeoff

- Larger sizes reduce the overhead by
- Reducing the number of tags
- Reducing the size of each tag
- But
- Have fewer blocks available
- And the time to fetch the block on a miss is longer


## Valid Bits

- Valid bits indicate whether cache line contains an up-to-date copy of the values in memory
- Must be 1 for a hit
- Reset to 0 on power up
- An item can be removed from the cache by setting its valid bit to 0


## Eviction

- Which cache line should be evicted from the cache to make room for a new line?
- Direct-mapped
- no choice, must evict line selected by index
- Associative caches
- random: select one of the lines at random
- round-robin: similar to random
- FIFO: replace oldest line
- LRU: replace line that has not been used in the longest time


## Cache Writes



- No-Write
- writes invalidate the cache and go to memory
- Write-Through
- writes go to main memory and cache
- Write-Back
- write cache, write main memory only when block is evicted


## What about Stores?

- Where should you write the result of a store?
- If that memory location is in the cache?
- Send it to the cache
- Should we also send it to memory right away? (write-through policy)
- Wait until we kick the block out (write-back policy)
- If it is not in the cache?
- Allocate the line (put it in the cache)? (write allocate policy)
- Write it directly to memory without allocation? (no write allocate policy)


Write-Through (REF 1)


| Write-Through (REF 1) |  |  |
| :---: | :---: | :---: |
| Processor | Cache | Memory |
|  |  | 78 |
|  |  | 29 |
|  |  | ${ }_{3}^{2} \quad 120$ |
| $\Rightarrow$ Ld R1ヶMI 11 | V tag data | 412 <br>  |
| Ld R2 $2 \leftarrow$ MII St R2 | 1 $\mathbf{0}$ 78 <br>    | 5150 <br> 102 <br> 185 |
|  | Iru 0 [ 29 | 162 |
|  |  | 18 |
|  |  | $99^{21}$ |
|  |  | 10 33 <br> 11  <br> 28  |
| R0 <br> R1 <br>  <br> 18 |  | 11 28 <br> 12  <br> 19  <br> 1  |
| R2 ${ }_{\text {R2 }}$ | Misses: 1 | 131200 <br> 20 <br> 15 |
|  | Hits: 0 | 14210 <br> 15 |
|  | Hers | $15 \lcm{225}$ |

Write-Through (REF 2)

| Processor | Cache |  | Memory |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  |  | 0 | 78 |
|  |  |  | 1 | 29 |
|  |  |  | 2 | 120 |
|  |  |  | 3 | 123 |
|  | V tag | data | 4 | 71 |
| $\Rightarrow \begin{array}{ll}\text { Ld R2 } \\ \text { St }\end{array}$ | \|150 | 78 | 5 | 150 |
|  |  | 29 | 6 | 162 |
| Ld R2 $\leftarrow$ M[ 10$]$ | lru 0 |  | 7 | 173 |
| St R1 $\rightarrow$ M[ 5 ] |  |  | 8 | 18 |
| St $\mathrm{R} 1 \rightarrow \mathrm{M}$ [ 10$]$ |  |  | 9 | 21 |
|  |  |  | 10 | 33 |
| R0 |  |  | 11 | 28 |
| R1 29 |  |  | 12 | 19 |
| R2 |  |  | 13 | 200 |
| R3 | Hits: | 0 | 14 | 210 |
|  |  |  | 15 | 225 |


| Write-Through (REF 2) |  |  |
| :---: | :---: | :---: |
|  |  |  |
| Processor | Cache | Memory |
|  |  | 178 <br>  <br> 18 |
|  |  | $2{ }_{2}$ |
|  | V tag data | 123 |
| $\Rightarrow \mathrm{Ld} \mathrm{R} 2<\mathrm{MI}_{7} \mathrm{l}$ |  | 71 |
|  | 10 | 162 |
|  | 1 3 162 | 173 |
|  | 173 | $\frac{18}{21}$ |
| $\rightarrow$ 隹 |  |  |
|  |  | 1118 <br> 28 |
| R11 29 | Misses: 2 | $12 \lcm{12}$ |
| R2 273 | Misses: 2 |  |
|  | Hits: 0 |  |

Write-Through (REF 3)



Write-Through (REF 4)

| Processor | Cache |  | Memory |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |
|  |  |  | 1 | 29 |
|  |  |  | 2 | 120 |
|  |  |  | 3 | 123 |
| Ld R1ヶMI 1 ] | V tag d | data | 4 | 71 |
| Ld R2 $\leftarrow$ M ${ }^{\text {l }} 71$ | 10 | 173 | 5 | 150 |
|  |  | 29 | 6 | 162 |
|  | lru 13 | 162 | 7 | 173 |
| St R1 $\rightarrow$ M[ 5 5 ] |  | 173 | 8 | 18 |
| St R1 $\rightarrow$ M[ 10 ] |  |  | 9 | 21 |
|  |  |  | 10 | 33 |
| R0 |  |  | 11 | 28 |
| R1 29 | Misses: |  | 12 | 19 |
| R2 173 | Misses: |  | 13 | 200 |
| R3 | Hits: | 1 | 14 | 210 |
|  |  |  | 15 | 225 |



Write-Through (REF 5)

| Processor | Cache |  | Memory |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  |  | 0 | 173 |
|  |  |  | 1 | 29 |
|  |  |  | 2 | 120 |
|  |  |  | 3 | 123 |
| $\begin{array}{ll}\text { Ld R1 } & \leftarrow \mathrm{MI} \\ \mathbf{1} & 1\end{array}$ | $V$ tag | data | 4 | 71 |
| $\begin{array}{llll}\text { Ld R2 } 2 \leftarrow \mathrm{ML} & 7 \\ \text { St }\end{array}$ | lru 10 | 173 | 5 | 29 |
|  |  | 29 | 6 | 162 |
| $\Rightarrow \xrightarrow[{L d R 2 \leftarrow M[10]} ~]{\text { L }}$ | 12 | 71 | 7 | 173 |
| St R1 $\rightarrow$ M[ 5 ] |  | 29 | 8 | 18 |
| St $\mathrm{R} 1 \rightarrow \mathrm{M}$ [ 10 ] |  |  | 9 | 21 |
|  |  |  | 10 | 33 |
| R0 |  |  | 11 | 28 |
| R1 29 |  |  | 12 | 19 |
| R2 173 |  |  | 13 | 200 |
| R3 | Hits: | 1 | 14 | 210 |
|  |  |  | 15 | 225 |


| Write-Through (REF 5) |  |  |  |
| :---: | :---: | :---: | :---: |
| Processor | Cache | Memory |  |
|  |  | 0 | 173 |
|  |  | 1 | 29 |
|  |  | 2 | 120 |
|  |  | 3 | 123 |
| Ld R1 $\leftarrow$ MI $1 \begin{array}{lll}\text { l }\end{array}$ | V tag data | 4 | 71 |
|  | 1 5 33 | 5 | 29 |
|  | - 28 | 6 | 162 |
| $\Rightarrow$ Ld R2ヶM[ 10$]$ | Iru 1 128 | 7 | 173 |
| St R1 $\rightarrow$ M[ 5 ] | 29 | 8 | 18 |
| St R1 $\rightarrow$ M[ 10$]$ |  | 9 | 21 |
|  |  | 10 | 33 |
| R0 |  | 11 | 28 |
| R1 29 | Misses: 4 | 12 | 19 |
| R2 33 | Misses: 4 | 13 | 200 |
| R3 | Hits: 1 | 14 | 210 |
|  |  | 15 | 225 |

Write-Through (REF 6)

| Processor | Cache | Memory |
| :---: | :---: | :---: |
|  |  | $0 \quad 173$ |
|  |  | 29 |
|  |  | $2{ }^{2} 120$ |
|  |  | $3{ }^{123}$ |
| Ld R1 $\leftarrow$ MI 111 | V tag data | 418 |
|  | 1 53 | 5 |
|  | - 28 | -6 6 |
| $\xrightarrow{\text { Ld R2 } 2 \leftarrow \mathrm{M}[10]}$ | Iru 12 2 71 | 173 |
|  | -29 | 18 |
| St R1 $\rightarrow$ M[ 10 ] | - | $9{ }^{9}$ |
|  |  | $10 \bigcirc 3$ |
| R0 |  | 1118 <br> 18 |
| R1 29. | s: 4 | $12 \quad 19$ |
| R2 33 | S. 4 | $13 \quad 200$ |
| R3 | Hits: 2 | $14 \bigcirc 210$ |
|  |  | $15 \lcm{225}$ |



## How Many Memory References?

- Each miss reads a block
(only two words in this cache)
- Each store writes a word
- Total reads: eight words
- Total writes: four words
but caches generally miss < 20\% usually much lower miss rates . . . but depends on both cache and application!



## How Many Memory References?

- Each miss reads a block
(only two words in this cache)
- Each store writes a word
- Total reads: eight words
- Total writes: six words, eight words, etc.
but caches generally miss < 20\% usually much lower miss rates . . . but depends on both cache and application!


## Write-Through vs. Write-Back

Can we also design the cache NOT to write all stores immediately to memory?

- Keep the most current copy in cache, and update memory when that data is evicted (write-back policy)
- Do we need to write-back all evicted lines?
- No, only blocks that have been stored into (written)


## Dirty Bits and Write-Back Buffers

| $V$ | D | Tag | Data Byte 0, Byte $1 \ldots$ | Byte N |
| :--- | :--- | :--- | :--- | :--- |
| 1 | 0 |  |  |  |
| 1 | 1 |  |  |  |
| 1 | 0 |  |  |  |

- Dirty bits indicate which lines have been written
- Dirty bits enable the cache to handle multiple writes to the same cache line without having to go to memory
- Dirty bit reset when line is allocated
- Set when block is written
- Write-back buffer
- A queue where dirty lines are placed
- Items added to the end as dirty lines are evicted from the cache
- Items removed from the front as memory writes are completed


## Handling Stores (Write-Back)

| Processor | Cache | Memory |
| :---: | :---: | :---: |
|  |  | 078 |
|  |  | 29 |
|  |  | 120 |
|  |  | 123 |
| Ld R1 $\leftarrow$ MI 1 ] | V d tag data | 71 |
| Ld R2 $\leftarrow \mathrm{ML}$ 7 ] | 0 0 | 150 |
|  |  | 162 |
|  | 0 | 173 |
| St R1 $\rightarrow$ M[ 5 ] |  | 18 |
| St R1 $\rightarrow$ M[ 10 ] |  | 21 |
|  |  | 10 |
| R0 |  | 11.28 |
| R1 | Misses: 0 | 12 |
| R2 |  | $13 \bigcirc$ |
| R3 | Hits: 0 | $1 4 \longdiv { 2 1 0 }$ |
|  |  | $1 5 \longdiv { 2 2 5 }$ |

Write-Back (REF 1)



Write-Back (REF 2)



Write-Back (REF 3)

| Processor | Cache |  | Memory |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  |  | 0 | 78 |
|  |  |  | 1 | 29 |
|  |  |  | 2 | 120 |
|  |  |  | 3 | 123 |
| $\mathbf{L d ~ R 1 ~} \leftarrow$ MI $1 \begin{array}{ll}\text { l }\end{array}$ | $V$ d tag | data | 4 | 71 |
|  | -100 | 78 | 5 | 150 |
|  |  | 29 | 6 | 162 |
| Ld R2 $\leftarrow$ M[ 10$]$ | 103 | 162 | 7 | 173 |
| St R1 $\rightarrow$ M[ 5 ] |  | 173 | 8 | 18 |
| St $\mathrm{R} 1 \rightarrow \mathrm{M}$ [ 10 ] |  |  | 9 | 21 |
|  |  |  | 10 | 33 |
| R0 |  |  | 11 | 28 |
| R1 29 |  |  | 12 | 19 |
| R2 173 | Misses: |  | 13 | 200 |
| R3 | Hits: | 0 | 14 | 210 |
|  |  |  | 15 | 225 |



Write-Back (REF 4)

| Processor | Cache |  | Memory |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  |  | 0 | 78 |
|  |  |  | 1 | 29 |
|  |  |  | 2 | 120 |
|  |  |  | 3 | 123 |
| Ld R1ヶM[ 1 ] | V d tag | data | 4 | 71 |
| $\left.\begin{array}{ll}\text { Ld R2 } 2 \leftarrow \mathrm{ML} & 7\end{array}\right]$ | 110 | 173 | 5 | 150 |
|  |  | 29 | 6 | 162 |
|  | E103 | 162 | 7 | 173 |
| St R1 $\rightarrow$ M[ 5 ] |  | 173 | 8 | 18 |
| St R1 $\rightarrow$ M[ 10 ] |  |  | 9 | 21 |
|  |  |  | 10 | 33 |
| R0 |  |  | 11 | 28 |
| R1 29 |  |  | 12 | 19 |
| R2 173 |  |  | 13 | 200 |
| R3 | Hits: | 1 | 14 | 210 |
|  |  |  | 15 | 225 |



Write-Back (REF 5)

| Processor | Cache |  | Memory |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  |  | 0 | 78 |
|  |  |  | 1 | 29 |
|  |  |  | 2 | 120 |
|  |  |  | 3 | 123 |
| Ld R1 $\leftarrow$ M[ 1 ] | V d tag | data | 4 | 71 |
|  | -110 | 173 | 5 | 150 |
| St R2 St M1 St |  | 29 | 6 | 162 |
| $\Rightarrow \mathrm{Ld} \mathrm{R2} \mathrm{\leftarrow MI} 10]$ | 113 | 71 | 7 | 173 |
| St R1 $\rightarrow$ M[ 5 ] |  | 29 | 8 | 18 |
| St $\mathrm{R} 1 \rightarrow \mathrm{MI} 10$ ] |  |  | 9 | 21 |
|  |  |  | 10 | 33 |
| R0 |  |  | 11 | 28 |
| R1 29 |  |  | 12 | 19 |
| R2 173 |  |  | 13 | 200 |
| R3 | Hits: | 2 | 14 | 210 |
|  |  |  | 15 | 225 |



## Write-Back (REF 5)




Write-Back (REF 7)

| Processor | Cache |  | Memory |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  |  | 0 | 78 |
|  |  |  | 1 | 29 |
|  |  |  | 2 | 120 |
| Ld R1 Ld R2 $\leftarrow$ M[ |  |  | 3 | 123 |
|  | V d tag | data | 4 | 71 |
| St R1 $\rightarrow$ MI 5 ] | 1115 | 29 | 5 | 150 |
| Ld R2 $\leftarrow$ MI 10$]$ |  | 28 | 6 | 162 |
| St R1 St M1 | 5113 | 71 | 7 | 173 |
|  |  | 29 | 8 | 18 |
| St R1 $\rightarrow$ M[ 10 ] |  |  | 9 | 21 |
|  |  |  | 10 | 33 |
| R0 |  |  | 11 | 28 |
| R1 29 |  |  | 12 | 19 |
| R2 33 |  |  | 13 | 200 |
| R3 | Hits: | 4 | 14 | 210 |
|  |  |  | 15 | 225 |

## How many memory references?

- Each miss reads a block

Two words in this cache

- Each evicted dirty cache line writes a block
- Total reads: six words
- Total writes: 4/6 words (after final eviction)

Write-Back (REF 8,9)


## How many memory references?

- Each miss reads a block

Two words in this cache

- Each evicted dirty cache line writes a block
- Total reads: six words
- Total writes: $4 / 6$ words (after final eviction)
- By comparison write-through was
- Reads: eight words
- Writes: 6/8/10 etc words
- Write-through or Write-back?


## Write-through vs. Write-back

- Write-through is slower
- But cleaner (memory always consistent)
- Write-back is faster
- But complicated when multi cores sharing memory

