Caches
The Memory Hierarchy

Want large, fast, and cheap memory, but...

- Large memories are slow (e.g., Hard Disk)
- Fast memories are small and expensive (e.g., SRAM)
The Memory Hierarchy

Want large, fast, and cheap memory, but...
Large memories are slow (e.g., Hard Disk)
Fast memories are small and expensive (e.g., SRAM)

Solution: Use a hierarchy of memories with different tradeoffs to fake a large, fast, cheap memory
Memory Hierarchy Interface

- Programming model: Single memory, single address space
Memory Hierarchy Interface

- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns
Caches

- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations.

![Diagram showing the interaction between CPU, Cache, and Main Memory]
Caches

- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations

- Processor sends accesses to cache. Two options:
Caches

- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations

- Processor sends accesses to cache. Two options:
  - Cache hit: Data for this address in cache, returned quickly
Caches

- **Cache**: A small, interim storage component that transparently retains (caches) data from recently accessed locations

  ![Cache Diagram]

- **Processor sends accesses to cache. Two options:**
  - **Cache hit**: Data for this address in cache, returned quickly
  - **Cache miss**: Data not in cache
    - Fetch data from memory, send it back to processor
    - Retain this data in the cache (replacing some other data)
Caches

- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations

- Processor sends accesses to cache. Two options:
  - **Cache hit**: Data for this address in cache, returned quickly
  - **Cache miss**: Data not in cache
    - Fetch data from memory, send it back to processor
    - Retain this data in the cache (replacing some other data)
  - Processor must deal with variable memory access time
Why Caches Work

- Two predictable properties of memory accesses:
  - **Temporal locality**: If a location has been accessed recently, it is likely to be accessed (reused) soon
  
  - **Spatial locality**: If a location has been accessed recently, it is likely that nearby locations will be accessed soon
Why Caches Work

- Two predictable properties of memory accesses:
  - **Temporal locality**: If a location has been accessed recently, it is likely to be accessed (reused) soon
  - **Spatial locality**: If a location has been accessed recently, it is likely that nearby locations will be accessed soon

- Result:
  - High hit rate (low miss ratio)
  - Reduced Average Memory Access Time (AMAT):

\[
AMAT = HitTime + MissRatio \times MissPenalty
\]
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

CPU

Tag  Data
A    Mem[A]
B    Mem[B]

Main Memory

(1-HR)
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: $X = \text{Tag}(i)$ for some cache line $i$

Return Data(i)
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i
Return Data(i)

MISS: X not found in Tag of any cache line
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: $X = \text{Tag}(i)$ for some cache line $i$
Return $\text{Data}(i)$

MISS: $X$ not found in Tag of any cache line
Read Mem[X]
Return Mem[X]
Select a line $k$ to hold Mem[X]
Write Tag($k$) = $X$
Data($k$) = Mem[X]
Basic Cache Algorithm (Reads)

Q: How do we “search” the cache?

On reference to Mem[X],
look for X among cache tags

HIT: X = Tag(i) for some cache line i
Return Data(i)

MISS: X not found in Tag of any cache line
Read Mem[X]
Return Mem[X]
Select a line k to hold Mem[X]
Write Tag(k)=X, Data(k) = Mem[X]
Direct-Mapped Caches

- Each word in memory maps into a single cache line.
- Access (for cache with $2^W$ lines):
  - Index into cache with $W$ address bits (the index bits).
  - Read out valid bit, tag, and data.
  - If valid bit == 1 and tag matches upper address bits, HIT.
- Example 8-line direct-mapped cache:

```
32-bit BYTE address
0000000000000000000000000000011101000
```

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (27 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Direct-Mapped Caches

- Each word in memory maps into a single cache line
- Access (for cache with $2^W$ lines):
  - Index into cache with $W$ address bits (the index bits)
  - Read out valid bit, tag, and data
  - If valid bit == 1 and tag matches upper address bits, HIT
- Example 8-line direct-mapped cache:

32-bit BYTE address

```
0000000000000000000000000011101000
```

<table>
<thead>
<tr>
<th>Byte offset bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (27 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Direct-Mapped Caches

- Each word in memory maps into a single cache line
- Access (for cache with $2^W$ lines):
  - Index into cache with $W$ address bits (the \textit{index bits})
  - Read out valid bit, tag, and data
  - If valid bit == 1 and tag matches upper address bits, HIT
- Example 8-line direct-mapped cache:

![Diagram of 8-line direct-mapped cache]

32-bit BYTE address

\begin{verbatim}
00000000000000000000000000000011101000
\end{verbatim}
Direct-Mapped Caches

- Each word in memory maps into a single cache line
- Access (for cache with \(2^W\) lines):
  - Index into cache with \(W\) address bits (the index bits)
  - Read out valid bit, tag, and data
  - If valid bit == 1 and tag matches upper address bits, HIT

Example 8-line direct-mapped cache:

32-bit BYTE address

\[0000000000000000000000000000000011101000\]
Direct-Mapped Caches

- Each word in memory maps into a single cache line.
- Access (for cache with $2^W$ lines):
  - Index into cache with $W$ address bits (the index bits).
  - Read out valid bit, tag, and data.
  - If valid bit == 1 and tag matches upper address bits, HIT.
- Example 8-line direct-mapped cache:

```
32-bit BYTE address: 00000000000000000000000000000011101000
```

```
Valid bit | Tag (27 bits) | Data (32 bits)
--- | --- | ---
0 | | |
1 | | |
3 | | |
4 | | |
5 | | |
6 | | |
7 | | |
```

```
Index bits: 0 1 2 3 4 5 6 7
```

```
Byte offset bits: =?
```
Direct-Mapped Caches

- Each word in memory maps into a single cache line
- Access (for cache with $2^w$ lines):
  - Index into cache with $W$ address bits (the index bits)
  - Read out valid bit, tag, and data
  - If valid bit $== 1$ and tag matches upper address bits, HIT
- Example 8-line direct-mapped cache:

```
32-bit BYTE address

000000000000000000000000000000011101000
```

```
Valid bit

0 1

Tag (27 bits)

0 1

Data (32 bits)

0 1

Index bits

Byte offset bits

=?

HIT
```
Example: Direct-Mapped Caches

64-line direct-mapped cache → 64 indices → 6 index bits

Read Mem[0x400C]

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (24 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000058</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>1</td>
<td>0x000058</td>
<td>0x00000000</td>
</tr>
<tr>
<td>2</td>
<td>0x000058</td>
<td>0x00000007</td>
</tr>
<tr>
<td>3</td>
<td>0x000040</td>
<td>0x42424242</td>
</tr>
<tr>
<td>4</td>
<td>0x000007</td>
<td>0x6FBA2381</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>0x000058</td>
<td>0xF7324A32</td>
</tr>
</tbody>
</table>
Example: Direct-Mapped Caches

64-line direct-mapped cache → 64 indices → 6 index bits

Read Mem[0x400C]
0100 0000 0000 1100

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (24 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000058</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>1</td>
<td>0x000058</td>
<td>0x00000000</td>
</tr>
<tr>
<td>2</td>
<td>0x000058</td>
<td>0x00000007</td>
</tr>
<tr>
<td>3</td>
<td>0x000040</td>
<td>0x42424242</td>
</tr>
<tr>
<td>4</td>
<td>0x000007</td>
<td>0x6FBA2381</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>0x000058</td>
<td>0xF7324A32</td>
</tr>
</tbody>
</table>
Example: Direct-Mapped Caches

64-line direct-mapped cache → 64 indices → 6 index bits

**Read Mem[0x400C]**

0100 0000 0000 1100

TAG: 0x40
INDEX: 0x3
BYTE OFFSET: 0x0

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (24 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000058</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>1</td>
<td>0x000058</td>
<td>0x00000000</td>
</tr>
<tr>
<td>2</td>
<td>0x000058</td>
<td>0x00000007</td>
</tr>
<tr>
<td>3</td>
<td>0x000040</td>
<td>0x42424242</td>
</tr>
<tr>
<td>4</td>
<td>0x000007</td>
<td>0x6FBA2381</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>0x000058</td>
<td>0xF7324A32</td>
</tr>
</tbody>
</table>
Example: Direct-Mapped Caches

64-line direct-mapped cache → 64 indices → 6 index bits

Read Mem[0x400C]

0100 0000 0000 1100

TAG: 0x40
INDEX: 0x3
BYTE OFFSET: 0x0

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (24 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000058</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>1</td>
<td>0x000058</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>2</td>
<td>0x000058</td>
<td>0x0000000007</td>
</tr>
<tr>
<td>3</td>
<td>0x000040</td>
<td>0x42424242</td>
</tr>
<tr>
<td>4</td>
<td>0x000007</td>
<td>0x6FBA2381</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>0x000058</td>
<td>0xF7324A32</td>
</tr>
</tbody>
</table>
**Example: Direct-Mapped Caches**

64-line direct-mapped cache → 64 indices → 6 index bits

**Read Mem[0x400C]**

0100 0000 0000 1100

TAG: 0x40
INDEX: 0x3
BYTE OFFSET: 0x0

HIT, DATA 0x42424242

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (24 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000058</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>1</td>
<td>0x000058</td>
<td>0x00000000</td>
</tr>
<tr>
<td>2</td>
<td>0x000058</td>
<td>0x00000007</td>
</tr>
<tr>
<td>3</td>
<td>0x000040</td>
<td>0x42424242</td>
</tr>
<tr>
<td>4</td>
<td>0x000007</td>
<td>0x6FBA2381</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>0x000058</td>
<td>0xF7324A32</td>
</tr>
</tbody>
</table>
# Example: Direct-Mapped Caches

64-line direct-mapped cache $\rightarrow$ 64 indices $\rightarrow$ 6 index bits

**Read Mem[0x400C]**

0100 0000 0000 1100

TAG: 0x40
INDEX: 0x3
BYTE OFFSET: 0x0

HIT, DATA 0x42424242

**Would 0x4008 hit?**

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (24 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000058</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>1</td>
<td>0x000058</td>
<td>0x00000000</td>
</tr>
<tr>
<td>2</td>
<td>0x000058</td>
<td>0x00000007</td>
</tr>
<tr>
<td>3</td>
<td>0x000040</td>
<td>0x42424242</td>
</tr>
<tr>
<td>4</td>
<td>0x000007</td>
<td>0x6FBA2381</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>0x000058</td>
<td>0xF7324A32</td>
</tr>
</tbody>
</table>
**Example: Direct-Mapped Caches**

64-line direct-mapped cache → 64 indices → 6 index bits

---

### Read Mem[0x400C]

- **Tag**: 0x40
- **Index**: 0x3
- **Byte Offset**: 0x0

**HIT, DATA 0x42424242**

Would 0x4008 hit?

- **Index**: 0x2 → tag mismatch → **MISS**

---

**Table**

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (24 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000058</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>1</td>
<td>0x000058</td>
<td>0x000000000</td>
</tr>
<tr>
<td>2</td>
<td>0x000058</td>
<td>0x000000007</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td><strong>0x000040</strong></td>
<td><strong>0x42424242</strong></td>
</tr>
<tr>
<td>4</td>
<td>0x000007</td>
<td>0x6FBA2381</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>0x000058</td>
<td>0xF7324A32</td>
</tr>
</tbody>
</table>
Example: Direct-Mapped Caches

64-line direct-mapped cache → 64 indices → 6 index bits

Read Mem[0x400C]

0100 0000 0000 1100

TAG: 0x40
INDEX: 0x3
BYTE OFFSET: 0x0

HIT, DATA 0x42424242

Would 0x4008 hit?
INDEX: 0x2 → tag mismatch → MISS

Valid bit | Tag (24 bits) | Data (32 bits)
---|---|---
0 | 0x000058 | 0xDEADBEEF
1 | 0x000058 | 0x00000000
2 | 0x000058 | 0x00000007
3 | 0x000040 | 0x42424242
4 | 0x000007 | 0x6FBA2381
... | ... | ...
63 | 0x000058 | 0xF7324A32

Part of the address (index bits) is encoded in the location Tag + Index bits unambiguously identify the data’s address
Selection of Index Bits

- Why do we chose low order bits for index?
Selection of Index Bits

- Why do we chose low order bits for index?
  - Allows consecutive memory locations to live in the cache simultaneously
Selection of Index Bits

- Why do we choose low order bits for index?
  - Allows consecutive memory locations to live in the cache simultaneously
  - Reduces likelihood of replacing data that may be accessed again in the near future
Selection of Index Bits

- Why do we chose low order bits for index?
  - Allows consecutive memory locations to live in the cache simultaneously
  - Reduces likelihood of replacing data that may be accessed again in the near future
  - Helps take advantage of locality
Block Size

- Take advantage of spatial locality: Store multiple words per data line
  - Always fetch entire block (multiple words) from memory
  - Another advantage: Reduces size of tag memory!
  - Potential disadvantage: Fewer indices in the cache

- Example: 4-block, 16-word direct-mapped cache

32-bit BYTE address
Block Size

- Take advantage of spatial locality: Store multiple words per data line
  - Always fetch entire block (multiple words) from memory
  - Another advantage: Reduces size of tag memory!
  - Potential disadvantage: Fewer indices in the cache
- Example: 4-block, 16-word direct-mapped cache

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (26 bits)</th>
<th>Data (4 words, 16 bytes)</th>
</tr>
</thead>
</table>

32-bit BYTE address

Byte offset bits: 2
Block Size

- Take advantage of spatial locality: Store multiple words per data line
  - Always fetch entire block (multiple words) from memory
  - Another advantage: Reduces size of tag memory!
  - Potential disadvantage: Fewer indices in the cache

- Example: 4-block, 16-word direct-mapped cache

```plaintext
Valid bit  Tag (26 bits)  Data (4 words, 16 bytes)
```

32-bit BYTE address

```
0 1 2 3

Block offset bits: 2
(4 words/block)

Byte offset bits: 2
```
Block Size

- Take advantage of spatial locality: Store multiple words per data line
  - Always fetch entire block (multiple words) from memory
  - Another advantage: Reduces size of tag memory!
  - Potential disadvantage: Fewer indices in the cache
- Example: 4-block, 16-word direct-mapped cache

![Diagram of cache organization]

- Valid bit
- Tag (26 bits)
- Data (4 words, 16 bytes)

32-bit BYTE address

Index bits: 2 (4 indices)
Block offset bits: 2 (4 words/block)
Byte offset bits: 2
Block Size

- Take advantage of spatial locality: Store multiple words per data line
  - Always fetch entire block (multiple words) from memory
  - Another advantage: Reduces size of tag memory!
  - Potential disadvantage: Fewer indices in the cache
- Example: 4-block, 16-word direct-mapped cache

![Diagram of cache organization]

- Valid bit
- Tag (26 bits)
- Data (4 words, 16 bytes)

32-bit BYTE address

- Tag bits: 26 (=32-6)
- Index bits: 2 (4 indices)
- Block offset bits: 2 (4 words/block)
- Byte offset bits: 2
Block Size Tradeoffs

- Larger block sizes...
  - Take advantage of spatial locality
  - Incur larger miss penalty since it takes longer to transfer the block from memory
  - Can increase the average hit time and miss ratio
Block Size Tradeoffs

- Larger block sizes...
  - Take advantage of spatial locality
  - Incur larger miss penalty since it takes longer to transfer the block from memory
  - Can increase the average hit time and miss ratio

![Graph showing the tradeoff between Block Size and Miss Penalty.](image)
Block Size Tradeoffs

- Larger block sizes...
  - Take advantage of spatial locality
  - Incur larger miss penalty since it takes longer to transfer the block from memory
  - Can increase the average hit time and miss ratio

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Miss Penalty</th>
<th>Miss Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploits spatial locality</td>
<td>Fewer blocks, compromises temporal locality</td>
<td></td>
</tr>
</tbody>
</table>

October 31, 2019
MIT 6.004 Fall 2019
### Block Size Tradeoffs

- Larger block sizes...
  - Take advantage of spatial locality
  - Incur larger miss penalty since it takes longer to transfer the block from memory
  - Can increase the average hit time and miss ratio
  
AMAT = HitTime + MissPenalty*MissRatio

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Miss Penalty</th>
<th>Miss Ratio</th>
<th>AMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Exploits spatial locality</td>
<td>Increased miss penalty and miss rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Exploits spatial locality
- Fewer blocks, compromises temporal locality
- ~64 bytes
Direct-Mapped Cache Problem: Conflict Misses

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Cache Line index</th>
<th>Hit/ Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0</td>
<td>HIT</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>HIT</td>
</tr>
<tr>
<td>1025</td>
<td>1</td>
<td>HIT</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>HIT</td>
</tr>
<tr>
<td>1026</td>
<td>2</td>
<td>HIT</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
<td>HIT</td>
</tr>
<tr>
<td>1024</td>
<td>0</td>
<td>HIT</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>HIT</td>
</tr>
</tbody>
</table>

Assume:
1. 1024-line DM cache
2. Block size = 1 word
3. Consider looping code, in steady state
4. Assume WORD, not BYTE, addressing

Loop A: Code at 1024, data at 37
Direct-Mapped Cache Problem: Conflict Misses

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Cache Line index</th>
<th>Hit/Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0</td>
<td>HIT</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>HIT</td>
</tr>
<tr>
<td>1025</td>
<td>1</td>
<td>HIT</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>HIT</td>
</tr>
<tr>
<td>1026</td>
<td>2</td>
<td>HIT</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
<td>HIT</td>
</tr>
<tr>
<td>1024</td>
<td>0</td>
<td>HIT</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>HIT</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Loop A: Code at 1024, data at 37 |

Assume:
- 1024-line DM cache
- Block size = 1 word
- Consider looping code, in steady state
- Assume WORD, not BYTE, addressing

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Cache Line index</th>
<th>Hit/Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0</td>
<td>MISS</td>
</tr>
<tr>
<td>2048</td>
<td>0</td>
<td>MISS</td>
</tr>
<tr>
<td>1025</td>
<td>1</td>
<td>MISS</td>
</tr>
<tr>
<td>2049</td>
<td>1</td>
<td>MISS</td>
</tr>
<tr>
<td>1026</td>
<td>2</td>
<td>MISS</td>
</tr>
<tr>
<td>2050</td>
<td>2</td>
<td>MISS</td>
</tr>
<tr>
<td>1024</td>
<td>0</td>
<td>MISS</td>
</tr>
<tr>
<td>2048</td>
<td>0</td>
<td>MISS</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Loop B: Code at 1024, data at 2048 |
Direct-Mapped Cache Problem: Conflict Misses

Assume:
- 1024-line DM cache
- Block size = 1 word
- Consider looping code, in steady state
- Assume WORD, not BYTE, addressing

Inflexible mapping (each address can only be in one cache location) ➞ Conflict misses (multiple addresses map to same cache index)!

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Cache Line index</th>
<th>Hit/Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0</td>
<td>HIT</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>HIT</td>
</tr>
<tr>
<td>1025</td>
<td>1</td>
<td>HIT</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>HIT</td>
</tr>
<tr>
<td>1026</td>
<td>2</td>
<td>HIT</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
<td>HIT</td>
</tr>
<tr>
<td>1024</td>
<td>0</td>
<td>HIT</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>HIT</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Cache Line index</th>
<th>Hit/Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0</td>
<td>MISS</td>
</tr>
<tr>
<td>2048</td>
<td>0</td>
<td>MISS</td>
</tr>
<tr>
<td>1025</td>
<td>1</td>
<td>MISS</td>
</tr>
<tr>
<td>2049</td>
<td>1</td>
<td>MISS</td>
</tr>
<tr>
<td>1026</td>
<td>2</td>
<td>MISS</td>
</tr>
<tr>
<td>2050</td>
<td>2</td>
<td>MISS</td>
</tr>
<tr>
<td>1024</td>
<td>0</td>
<td>MISS</td>
</tr>
<tr>
<td>2048</td>
<td>0</td>
<td>MISS</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fully-Associative Cache

Opposite extreme: Any address can be in any location
- No cache index!
- Flexible (no conflict misses)
- Expensive: Must compare tags of all entries in parallel to find matching one

32-bit BYTE address

Tag bits

Block offset bits

Byte offset bits

Tag

Valid bit

Data
N-way Set-Associative Cache

- Use multiple direct-mapped caches in parallel to reduce conflict misses
- Nomenclature:
  - # Rows = # Sets
  - # Columns = # Ways
  - Set size = #ways = “set associativity” (e.g., 4-way → 4 lines/set)
N-way Set-Associative Cache

- Use multiple direct-mapped caches in parallel to reduce conflict misses
- Nomenclature:
  - # Rows = # Sets
  - # Columns = # Ways
  - Set size = #ways = “set associativity” (e.g., 4-way → 4 lines/set)
- Each address maps to only one set, but can be in any way within the set
N-way Set-Associative Cache

- Use multiple direct-mapped caches in parallel to reduce conflict misses
- Nomenclature:
  - # Rows = # Sets
  - # Columns = # Ways
  - Set size = #ways = “set associativity” (e.g., 4-way → 4 lines/set)
- Each address maps to only one set, but can be in any way within the set
- Tags from all ways are checked in parallel
N-way Set-Associative Cache

- Use multiple direct-mapped caches in parallel to reduce conflict misses
- Nomenclature:
  - # Rows = # Sets
  - # Columns = # Ways
  - Set size = #ways = “set associativity” (e.g., 4-way → 4 lines/set)
- Each address maps to only one set, but can be in any way within the set
- Tags from all ways are checked in parallel
N-way Set-Associative Cache

- Use multiple direct-mapped caches in parallel to reduce conflict misses
- Nomenclature:
  - # Rows = # Sets
  - # Columns = # Ways
  - Set size = #ways = “set associativity” (e.g., 4-way → 4 lines/set)
- Each address maps to only one set, but can be in any way within the set
- Tags from all ways are checked in parallel

- Fully-associative cache: Extreme case with a single set and as many ways as cache lines
Associativity Implies Choices

**Direct-mapped**
- Compare addr with only one tag
- Location A can be stored in exactly one cache line

**N-way set-associative**
- Compare addr with N tags simultaneously
- Location A can be stored in exactly one set, but in any of the N cache lines belonging to that set

**Fully associative**
- Compare addr with each tag simultaneously
- Location A can be stored in any cache line
Associativity Implies Choices

Direct-mapped

- Compare addr with only one tag
- Location A can be stored in exactly one cache line

N-way set-associative

- Compare addr with N tags simultaneously
- Location A can be stored in exactly one set, but in any of the N cache lines belonging to that set

Fully associative

- Compare addr with each tag simultaneously
- Location A can be stored in any cache line
Associativity Implies Choices

Direct-mapped
- Compare addr with only one tag
- Location A can be stored in exactly one cache line

N-way set-associative
- Compare addr with N tags simultaneously
- Location A can be stored in exactly one set, but in any of the N cache lines belonging to that set

Fully associative
- Compare addr with each tag simultaneously
- Location A can be stored in any cache line
Associativity Implies Choices

**Direct-mapped**
- Compare addr with only one tag
- Location A can be stored in exactly one cache line

**N-way set-associative**
- Compare addr with N tags simultaneously
- Location A can be stored in exactly one set, but in any of the N cache lines belonging to that set

**Fully associative**
- Compare addr with each tag simultaneously
- Location A can be stored in any cache line
Associativity Implies Choices

Issue: Replacement Policy

Direct-mapped

- Compare addr with only one tag
- Location A can be stored in exactly one cache line

N-way set-associative

- Compare addr with N tags simultaneously
- Location A can be stored in exactly one set, but in any of the N cache lines belonging to that set

Fully associative

- Compare addr with each tag simultaneously
- Location A can be stored in any cache line

October 31, 2019
MIT 6.004 Fall 2019
Replacement Policies

- Optimal policy: Replace the line that is accessed furthest in the future
  - Requires knowing the future...
Replacement Policies

- Optimal policy: Replace the line that is accessed furthest in the future
  - Requires knowing the future...
- Idea: Predict the future from looking at the past
  - If a line has not been used recently, it’s often less likely to be accessed in the near future (a locality argument)
Replacement Policies

- **Optimal policy:** Replace the line that is accessed furthest in the future
  - Requires knowing the future...

- **Idea:** Predict the future from looking at the past
  - If a line has not been used recently, it’s often less likely to be accessed in the near future (a locality argument)

- **Least Recently Used (LRU):** Replace the line that was accessed furthest in the past
  - Works well in practice
  - Need to keep ordered list of N items → N! orderings
    → $O(\log_2 N!) = O(N \log_2 N)$ “LRU bits” + complex logic
  - Caches often implement cheaper approximations of LRU
Replacement Policies

- **Optimal policy:** Replace the line that is accessed furthest in the future
  - Requires knowing the future...
- **Idea:** Predict the future from looking at the past
  - If a line has not been used recently, it’s often less likely to be accessed in the near future (a locality argument)
- **Least Recently Used (LRU):** Replace the line that was accessed furthest in the past
  - Works well in practice
  - Need to keep ordered list of N items $\rightarrow N!$ orderings $\rightarrow O(\log_2 N!) = O(N \log_2 N)$ “LRU bits” + complex logic
  - Caches often implement cheaper approximations of LRU
- **Other policies:**
  - First-In, First-Out (least recently replaced)
  - Random: Choose a candidate at random
    - Not very good, but does not have adversarial access patterns
Write Policy

Write-through: CPU writes are cached, but also written to main memory immediately (stalling the CPU until write is completed). Memory always holds current contents

- Simple, slow, wastes bandwidth
Write Policy

**Write-through**: CPU writes are cached, but also written to main memory immediately (stalling the CPU until write is completed). Memory always holds current contents

- Simple, slow, wastes bandwidth

**Write-back**: CPU writes are cached, but not written to main memory until we replace the line. Memory contents can be “stale”

- Fast, low bandwidth, more complex
- Commonly implemented in current systems
**Example: Cache Write-Hit**

16-line direct-mapped cache → 4 index bits
Block size = 4 → 2 block offset bits
Write Policy = Write Back
Write: 0x09 to 0x4818

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Tag (24 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0x48</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line (4 words, 16 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Example: Cache Write-Hit

16-line direct-mapped cache → 4 index bits
Block size = 4 → 2 block offset bits
Write Policy = Write Back
Write: 0x09 to 0x4818

0100 1000 0001 1000

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Tag (24 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0x48</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line (4 words, 16 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01 0x02 0x03 0x04</td>
</tr>
<tr>
<td>: : : :</td>
</tr>
<tr>
<td>: : : :</td>
</tr>
</tbody>
</table>
Example: Cache Write-Hit

16-line direct-mapped cache → 4 index bits
Block size = 4 → 2 block offset bits
Write Policy = Write Back
Write: 0x09 to 0x4818

- Tag: 0x48
- Index: 0x1
- Block Off: 2
- Byte Off: 0x0

```
0100 1000 0001 1000
```

V D Tag (24 bits)

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Tag (24 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0x48</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>:</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Line (4 words, 16 bytes)

```
0x01 0x02 0x03 0x04
...
```

October 31, 2019  
MIT 6.004 Fall 2019

L16-19
Example: Cache Write-Hit

16-line direct-mapped cache → 4 index bits
Block size = 4 → 2 block offset bits
Write Policy = Write Back
Write: 0x09 to 0x4818

Tag: 0x48
Index: 0x1
Block Off: 2
Byte Off: 0x0
Example: Cache Write-Hit

16-line direct-mapped cache $\rightarrow$ 4 index bits
Block size = 4 $\rightarrow$ 2 block offset bits
Write Policy = Write Back
Write: 0x09 to 0x4818

0100 1000 0001 1000

Tag: 0x48
Index: 0x1
Block Off: 2
Byte Off: 0x0
Example: Cache Write-Hit

16-line direct-mapped cache → 4 index bits
Block size = 4 → 2 block offset bits
Write Policy = Write Back
Write: 0x09 to 0x4818

0100 1000 0001 1000
Tag: 0x48
Index: 0x1
Block Off: 2
Byte Off: 0x0
Example: Cache Write-Hit

16-line direct-mapped cache → 4 index bits
Block size = 4 → 2 block offset bits
Write Policy = Write Back
Write: 0x09 to 0x4818

0100 1000 0001 1000

Tag: 0x48
Index: 0x1
Block Off: 2
Byte Off: 0x0

D=1: cache contents no longer match main memory so write back line to memory upon replacement
Example: Cache Write-Miss

Tag: 0x48
Index: 0x1
Block Off: 2
Byte Off: 0x0

Write: 0x09 to 0x4818
Example: Cache Write-Miss

Write: 0x09 to 0x4818

1. Tags don’t match -> Miss
Example: Cache Write-Miss

1. Tags don’t match -> Miss
   - **D=1**: Write cache line 1 (tag = 0x280: addresses 0x28010-0x2801C) back to memory
Example: Cache Write-Miss

1. Tags don’t match -> Miss
   - D=1: Write cache line 1 (tag = 0x280: addresses 0x28010-0x2801C) back to memory
   - If D=0: Don’t need to write line back to memory.
Example: Cache Write-Miss

Write: 0x09 to 0x4818

1. Tags don’t match -> Miss
   - D=1: Write cache line 1 (tag = 0x280: addresses 0x28010-0x2801C) back to memory
   - If D=0: Don’t need to write line back to memory.

2. Load line (tag = 0x48: addresses 0x4810-0x481C) from memory
Example: Cache Write-Miss

Write: 0x09 to 0x4818

1. Tags don’t match -> Miss
   - D=1: Write cache line 1 (tag = 0x280: addresses 0x28010-0x2801C) back to memory
   - If D=0: Don’t need to write line back to memory.

2. Load line (tag = 0x48: addresses 0x4810-0x481C) from memory

3. Write 0x09 to 0x4818 (block offset 2), set D=1.
Summary: Cache Tradeoffs

\[ \text{AMAT} = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty} \]
Summary: Cache Tradeoffs

\[ \text{AMAT} = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty} \]

- Cache size
Summary: Cache Tradeoffs

\[ AMAT = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty} \]

- Cache size
- Block size
Summary: Cache Tradeoffs

\[ AMAT = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty} \]

- Cache size
- Block size
- Associativity
Summary: Cache Tradeoffs

\[ AMAT = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty} \]

- Cache size
- Block size
- Associativity
- Replacement policy
Summary: Cache Tradeoffs

\[ AMAT = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty} \]

- Cache size
- Block size
- Associativity
- Replacement policy
- Write policy
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

2-Way

FA

0x0 = 0b000000
DM index = 000
2-Way index = 00
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

2-Way

FA

0x0 = 0b000000
DM index = 000
2-Way index = 00
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU

Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

2-Way

FA

0x0 = 0b000000
DM index = 000
2-Way index = 00
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

M[0x0]

2-Way

M[0x0]

FA

M[0x0]

0x0 = 0b000000
DM index = 000
2-Way index = 00
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

2-Way

FA

0x0 = 0b000000
DM index = 000
2-Way index = 00

0x10 = 0b010000
DM index = 100
2-Way index = 00
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM
- M[0x0]
- M[0x10]

2-Way
- M[0x0]
- M[0x10]

FA
- M[0x0]

0x0 = 0b000000
DM index = 000
2-Way index = 00

0x10 = 0b010000
DM index = 100
2-Way index = 00
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

M[0x0]
M[0x10]

2-Way

M[0x0]
M[0x10]

FA

M[0x0]
M[0x10]

0x0 = 0b000000
DM index = 000
2-Way index = 00

0x10 = 0b010000
DM index = 100
2-Way index = 00
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU

Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

M[0x0]
M[0x10]

2-Way

M[0x0]
M[0x10]

FA

M[0x0]
M[0x10]

0x0 = 0b000000
DM index = 000
2-Way index = 00

0x10 = 0b00100
DM index = 001
2-Way index = 00

0x4 = 0b000100
DM index = 001
2-Way index = 00

0x24 = 0b001100
DM index = 001
2-Way index = 01
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU

Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

2-Way

FA

0x0 = 0b000000
DM index = 000
2-Way index = 00

0x10 = 0b0110
DM index = 001
2-Way index = 01

0x4 = 0b000100
DM index = 001
2-Way index = 01
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU

Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

<table>
<thead>
<tr>
<th>M[0x0]</th>
<th>M[0x4]</th>
<th>M[0x10]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2-Way

<table>
<thead>
<tr>
<th>M[0x0]</th>
<th>M[0x10]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M[0x0]</th>
<th>M[0x4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FA

<table>
<thead>
<tr>
<th>M[0x0]</th>
<th>M[0x10]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>

0x0 = 0b000000
DM index = 000
2-Way index = 00

0x10 = 0b001
DM index = 01
2-Way index = 1

0x4 = 0b000100
DM index = 001
2-Way index = 01
**Example: Comparing Hit Rates**

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU

Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

<table>
<thead>
<tr>
<th>DM</th>
<th>2-Way</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x0]</td>
<td>M[0x0]</td>
<td>M[0x0]</td>
</tr>
<tr>
<td>M[0x4]</td>
<td>M[0x4]</td>
<td></td>
</tr>
<tr>
<td>M[0x10]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Oct 31, 2019**

<table>
<thead>
<tr>
<th>0x0 = 0b000000</th>
<th>0x10 = 0b010010</th>
<th>0x4 = 0b000100</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM index = 000</td>
<td>DM index = 101</td>
<td>DM index = 001</td>
</tr>
<tr>
<td>2-Way index = 00</td>
<td>2-Way index = 00</td>
<td>2-Way index = 01</td>
</tr>
</tbody>
</table>

MIT 6.004 Fall 2019
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

<table>
<thead>
<tr>
<th>0x0</th>
<th>0x4</th>
<th>0x10</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x0]</td>
<td>M[0x4]</td>
<td>M[0x10]</td>
</tr>
</tbody>
</table>

2-Way

<table>
<thead>
<tr>
<th>0x0</th>
<th>0x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x0]</td>
<td>M[0x10]</td>
</tr>
</tbody>
</table>

FA

<table>
<thead>
<tr>
<th>0x0</th>
<th>0x10</th>
<th>0x4</th>
<th>0x24</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x0]</td>
<td>M[0x10]</td>
<td>M[0x4]</td>
<td>-</td>
</tr>
</tbody>
</table>

October 31, 2019
MIT 6.004 Fall 2019
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU

Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

2-Way

FA

0x0 = 0b000000
DM index = 000
2-Way index = 00

0x10 = 0b010000
DM index = 001
2-Way index = 01

0x4 = 0b000100
DM index = 001
2-Way index = 01

0x24 = 0b100100
DM index = 001
2-Way index = 01
### Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

<table>
<thead>
<tr>
<th>DM</th>
<th>2-Way</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x0]</td>
<td>M[0x0]</td>
<td>M[0x0]</td>
</tr>
<tr>
<td>M[0x4]</td>
<td>M[0x4]</td>
<td>M[0x4]</td>
</tr>
<tr>
<td>M[0x10]</td>
<td>M[0x24]</td>
<td>M[0x10]</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0x10</td>
<td>0x4</td>
<td>0x24</td>
</tr>
<tr>
<td>0b000000</td>
<td>0b010000</td>
<td>0b000100</td>
<td>0b100100</td>
</tr>
<tr>
<td>DM index</td>
<td>2-Way index</td>
<td>DM index</td>
<td>2-Way index</td>
</tr>
<tr>
<td>000</td>
<td>00</td>
<td>001</td>
<td>01</td>
</tr>
</tbody>
</table>
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

2-Way

FA

0x0 = 0b000000
DM index = 000
2-Way index = 00

0x10 = 0b01
DM index = 1
2-Way index = 01

0x4 = 0b000100
DM index = 001
2-Way index = 01

0x24 = 0b100100
DM index = 001
2-Way index = 01

October 31, 2019
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU
Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

M[0x0]

M[0x4]

M[0x10]

2-Way

M[0x0]

M[0x4]

M[0x10]

M[0x24]

FA

M[0x0]

M[0x10]

M[0x4]

M[0x24]

0x0 = 0b000000
DM index = 000
2-Way index = 00

0x10 = 0b010001
DM index = 001
2-Way index = 01

0x4 = 0b000100
DM index = 001
2-Way index = 01

0x24 = 0b100100
DM index = 001
2-Way index = 01
Example: Comparing Hit Rates

3 Caches: DM, 2-Way, FA: each has 8 words, block size=1, LRU

Access following addresses repeatedly: 0x0, 0x10, 0x4, 0x24

DM

M[0x0]  
M[0x4]  
M[0x10]

2-Way

M[0x0]  
M[0x4]  
M[0x10]  
M[0x24]

FA

M[0x0]  
M[0x10]  
M[0x4]  
M[0x24]

DM: 50% hit rate
2-Way: 100% hit rate
FA: 100% hit rate
Example 2: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x10, 0x14, 0x18, 0x1C, 0x20 repeatedly
Example 2: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x10, 0x14, 0x18, 0x1C, 0x20 repeatedly

DM

<table>
<thead>
<tr>
<th>M[0x0]</th>
<th>M[0x20]</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x4]</td>
<td></td>
</tr>
<tr>
<td>M[0x8]</td>
<td></td>
</tr>
<tr>
<td>M[0xC]</td>
<td></td>
</tr>
<tr>
<td>M[0x10]</td>
<td></td>
</tr>
<tr>
<td>M[0x14]</td>
<td></td>
</tr>
<tr>
<td>M[0x18]</td>
<td></td>
</tr>
<tr>
<td>M[0x1C]</td>
<td></td>
</tr>
</tbody>
</table>
Example 2: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x10, 0x14, 0x18, 0x1C, 0x20 repeatedly

<table>
<thead>
<tr>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x0]</td>
</tr>
<tr>
<td>M[0x20]</td>
</tr>
<tr>
<td>M[0x4]</td>
</tr>
<tr>
<td>M[0x8]</td>
</tr>
<tr>
<td>M[0xC]</td>
</tr>
<tr>
<td>M[0x10]</td>
</tr>
<tr>
<td>M[0x14]</td>
</tr>
<tr>
<td>M[0x18]</td>
</tr>
<tr>
<td>M[0x1C]</td>
</tr>
</tbody>
</table>

DM: Hit rate = 7/9
Example 2: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x10, 0x14, 0x18, 0x1C, 0x20 repeatedly

DM

M[0x0]  M[0x20]
M[0x4]  M[0x10]
M[0x8]  M[0x0]  M[0x20]
M[0xC]  M[0x14]  M[0x18]
M[0x10] M[0x18]  M[0xC]
M[0x14]  M[0xC]  M[0x1C]
M[0x18]  M[0x1C]  
M[0x20]  

2-Way

M[0x0]  M[0x10]  
M[0x4]  M[0x0]  M[0x20]
M[0x8]  M[0x14]
M[0xC]  M[0x18]
M[0x10] M[0xC]
M[0x14]
M[0x18]  M[0x1C]
M[0x20]

DM: Hit rate = 7/9
Example 2: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x10, 0x14, 0x18, 0x1C, 0x20 repeatedly

<table>
<thead>
<tr>
<th>DM</th>
<th></th>
<th>2-Way</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x0]</td>
<td>M[0x20]</td>
<td>M[0x0]</td>
</tr>
<tr>
<td>M[0x4]</td>
<td>M[0x10]</td>
<td>M[0x20]</td>
</tr>
<tr>
<td>M[0x8]</td>
<td></td>
<td>M[0x4]</td>
</tr>
<tr>
<td>M[0xC]</td>
<td></td>
<td>M[0x8]</td>
</tr>
<tr>
<td>M[0x10]</td>
<td></td>
<td>M[0xC]</td>
</tr>
<tr>
<td>M[0x14]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M[0x18]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M[0x1C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DM: Hit rate = 7/9

2-Way: Hit rate = 6/9
### Example 2: Comparing Hit Rates

Access: $0x0, 0x4, 0x8, 0xC, 0x10, 0x14, 0x18, 0x1C, 0x20$ repeatedly

<table>
<thead>
<tr>
<th>DM</th>
<th>2-Way</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M[0x0]$</td>
<td>$M[0x0]$</td>
<td>$M[0x0] M[0x20]$</td>
</tr>
<tr>
<td>$M[0x20]$</td>
<td>$M[0x20]$ $M[0x10]$</td>
<td>$M[0x4] M[0x0]$</td>
</tr>
<tr>
<td>$M[0x4]$</td>
<td>$M[0x4]$ $M[0x0]$</td>
<td>$M[0x8] M[0x4]$</td>
</tr>
<tr>
<td>$M[0x8]$</td>
<td>$M[0x8]$ $M[0x10]$</td>
<td>$M[0xC] M[0x8]$</td>
</tr>
<tr>
<td>$M[0xC]$</td>
<td>$M[0xC]$ $M[0x14]$</td>
<td>$M[0x10] M[0xC]$</td>
</tr>
<tr>
<td>$M[0x10]$</td>
<td>$M[0x10]$ $M[0x18]$</td>
<td>$M[0x14] M[0x10]$</td>
</tr>
<tr>
<td>$M[0x14]$</td>
<td>$M[0x14]$ $M[0x1C]$</td>
<td>$M[0x18] M[0x14]$</td>
</tr>
<tr>
<td>$M[0x18]$</td>
<td>$M[0x18]$ $M[0x1C]$</td>
<td>$M[0x1C] M[0x18]$</td>
</tr>
</tbody>
</table>

**DM:** Hit rate = $7/9$  
**2-Way:** Hit rate = $6/9$
Example 2: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x10, 0x14, 0x18, 0x1C, 0x20 repeatedly

DM: Hit rate = 7/9

2-Way: Hit rate = 6/9

FA: Hit rate = 0%
Example 3: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x20, 0x24, 0x28, 0x2C, 0x10 repeatedly
Example 3: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x20, 0x24, 0x28, 0x2C, 0x10 repeatedly

<table>
<thead>
<tr>
<th></th>
<th>M[0x0]</th>
<th>M[0x20]</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x4]</td>
<td>M[0x24]</td>
<td></td>
</tr>
<tr>
<td>M[0x8]</td>
<td>M[0x28]</td>
<td></td>
</tr>
<tr>
<td>M[0xC]</td>
<td>M[0x2C]</td>
<td></td>
</tr>
<tr>
<td>M[0x10]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 3: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x20, 0x24, 0x28, 0x2C, 0x10 repeatedly

DM

M[0x0]
M[0x20]
M[0x4]
M[0x24]
M[0x8]
M[0x28]
M[0xC]
M[0x2C]
M[0x10]

DM: Hit rate = 1/9
Example 3: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x20, 0x24, 0x28, 0x2C, 0x10 repeatedly

DM

M[0x0]  M[0x20]
M[0x4]  M[0x24]
M[0x8]  M[0x28]
M[0xC]  M[0x2C]
M[0x10]

2-Way

M[0x0]  M[0x20]  M[0x20]  M[0x10]
M[0x0]  M[0x20]  M[0x20]  M[0x10]
M[0x4]  M[0x24]  M[0x4]  M[0x24]
M[0x8]  M[0x28]  M[0x8]  M[0x28]
M[0xC]  M[0x2C]  M[0xC]  M[0x2C]

DM: Hit rate = 1/9
Example 3: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x20, 0x24, 0x28, 0x2C, 0x10 repeatedly

DM

M[0x0]
M[0x20]
M[0x4]
M[0x24]
M[0x8]
M[0x28]
M[0xC]
M[0x2C]
M[0x10]

2-Way

M[0x0]
M[0x10]
M[0x20]
M[0x4]
M[0x24]
M[0x8]
M[0x28]
M[0xC]
M[0x2C]

DM: Hit rate = 1/9

2-Way: Hit rate = 6/9

October 31, 2019

MIT 6.004 Fall 2019

L16-24
Example 3: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x20, 0x24, 0x28, 0x2C, 0x10 repeatedly

<table>
<thead>
<tr>
<th>DM</th>
<th>2-Way</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x0]</td>
<td>M[0x0]</td>
<td>M[0x0]</td>
</tr>
<tr>
<td>M[0x20]</td>
<td>M[0x10]</td>
<td>M[0x10]</td>
</tr>
<tr>
<td>M[0x4]</td>
<td>M[0x20]</td>
<td>M[0x20]</td>
</tr>
<tr>
<td>M[0x10]</td>
<td>M[0x4]</td>
<td>M[0x0]</td>
</tr>
<tr>
<td>M[0x8]</td>
<td>M[0x8]</td>
<td>M[0x8]</td>
</tr>
<tr>
<td>M[0x28]</td>
<td>M[0xC]</td>
<td>M[0x8]</td>
</tr>
<tr>
<td>M[0xC]</td>
<td>M[0x2C]</td>
<td>M[0xC]</td>
</tr>
<tr>
<td>M[0x2C]</td>
<td>M[0x2C]</td>
<td>M[0x2C]</td>
</tr>
</tbody>
</table>

DM: Hit rate = 1/9
2-Way: Hit rate = 6/9
### Example 3: Comparing Hit Rates

Access: 0x0, 0x4, 0x8, 0xC, 0x20, 0x24, 0x28, 0x2C, 0x10 repeatedly

<table>
<thead>
<tr>
<th>DM</th>
<th>2-Way</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0x0]</td>
<td>M[0x0]</td>
<td>M[0x0]</td>
</tr>
<tr>
<td>M[0x20]</td>
<td>M[0x10]</td>
<td>M[0x10]</td>
</tr>
<tr>
<td>M[0x4]</td>
<td>M[0x20]</td>
<td>M[0x4]</td>
</tr>
<tr>
<td>M[0x24]</td>
<td>M[0x8]</td>
<td>M[0x0]</td>
</tr>
<tr>
<td>M[0x8]</td>
<td>M[0x8]</td>
<td>M[0x8]</td>
</tr>
<tr>
<td>M[0x28]</td>
<td>M[0xC]</td>
<td>M[0x8]</td>
</tr>
<tr>
<td>M[0xC]</td>
<td>M[0x2C]</td>
<td>M[0xC]</td>
</tr>
</tbody>
</table>

DM: Hit rate = 1/9

2-Way: Hit rate = 6/9

FA: Hit rate = 0%
Thank you!

Next lecture: Operating Systems