# COS 284 TUTORIAL 6

CLASS TEST 4 RECAP

# DIRECT MAPPED CACHING

- Assume a computer with memory of 2^16 bytes and a **direct mapped cache** with **4 blocks** where each block has 16 bytes.
- Each **memory address consists of 16 bits** of which the rightmost 4 bits reflect the offset field, the leftmost 10 bits make up the tag, and the 2 bits in between specify the block in cache. The 10 tag bits and the 2 cache block bits together indicate the 12-bit number of the associated memory block.

# DIRECT MAPPED CACHING

- Assume a computer with memory of 2^16 bytes and a **direct mapped cache** with **4 blocks** where each block has 16 bytes.
- Each **memory address consists of 16 bits** of which the rightmost 4 bits reflect the offset field, the leftmost 10 bits make up the tag, and the 2 bits in between specify the block in cache. The 10 tag bits and the 2 cache block bits together indicate the 12-bit number of the associated memory block.

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| | ← 16 bits → | |

# DIRECT MAPPED CACHING

- Assume a computer with memory of 2^16 bytes and a **direct mapped cache** with **4 blocks** where each block has 16 bytes.
- Each **memory address consists of 16 bits** of which the rightmost 4 bits reflect the offset field, the leftmost 10 bits make up the tag, and the 2 bits in between specify the block in cache. The 10 tag bits and the 2 cache block bits together indicate the 12-bit number of the associated memory block.

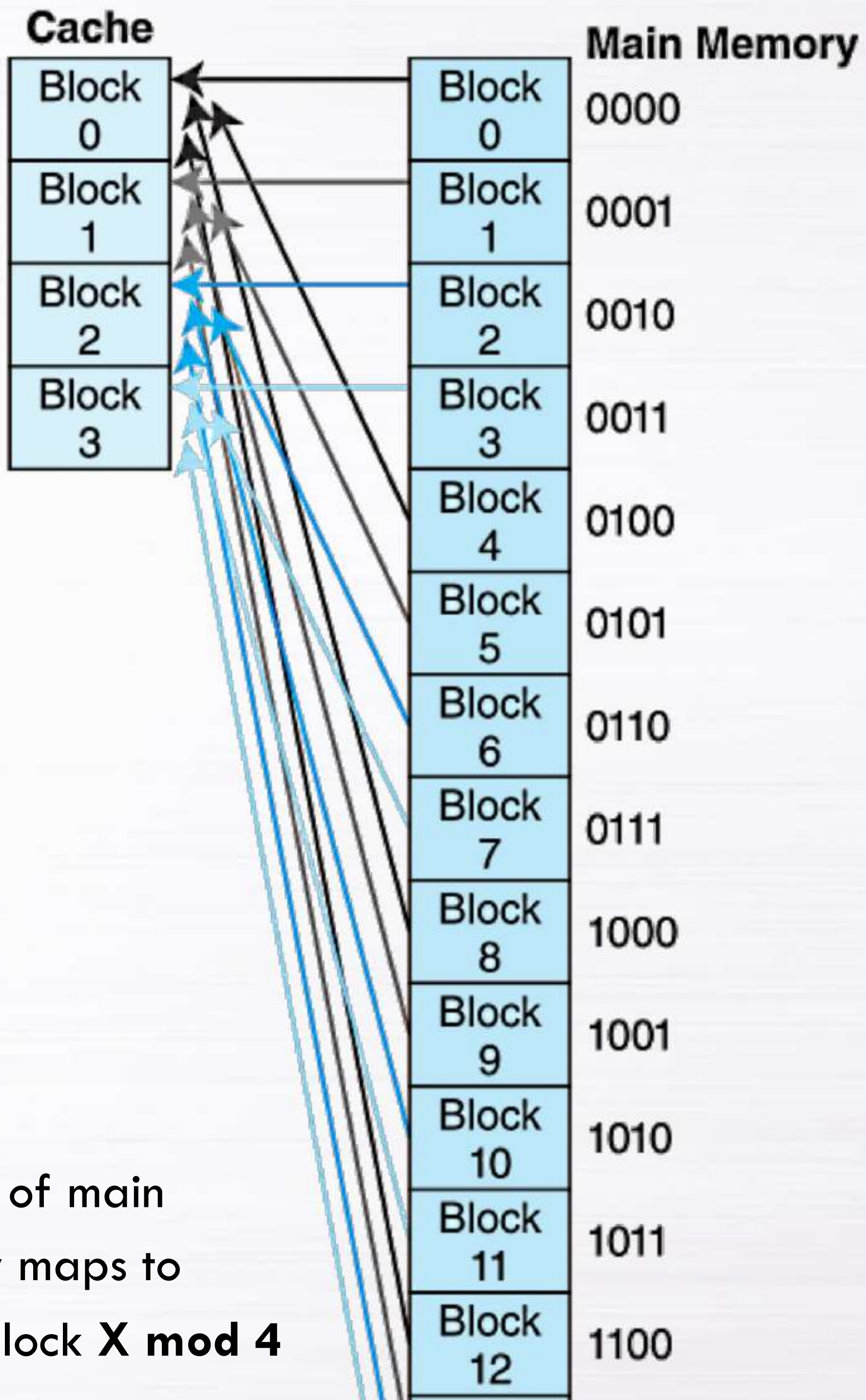
| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| | ← 16 bits → | |

- Example: 1010101111 00 1101 (binary)

  ABCD (hexadecimal)

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

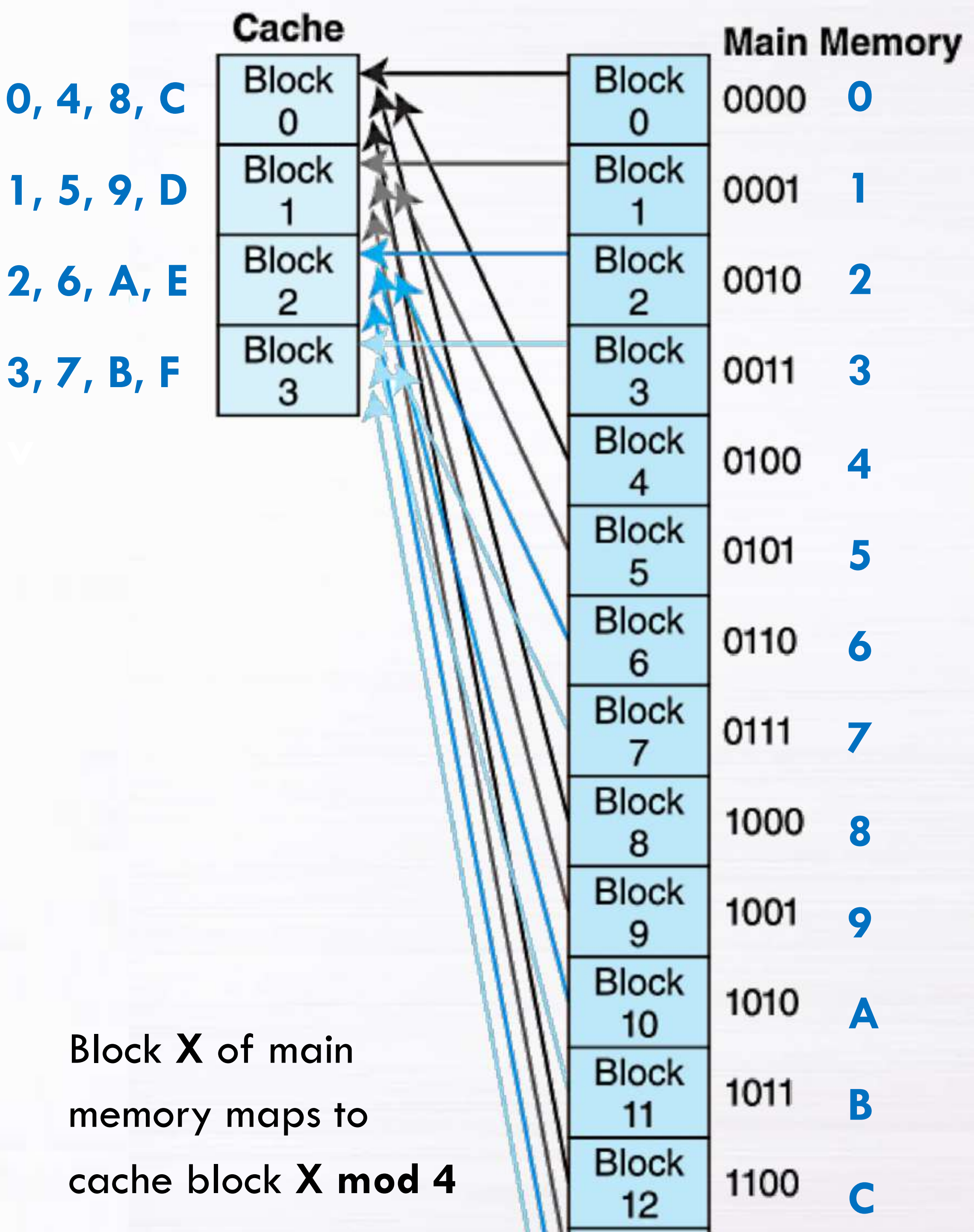
# DIRECT MAPPED CACHING




Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
| --- | --- | --- |
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING



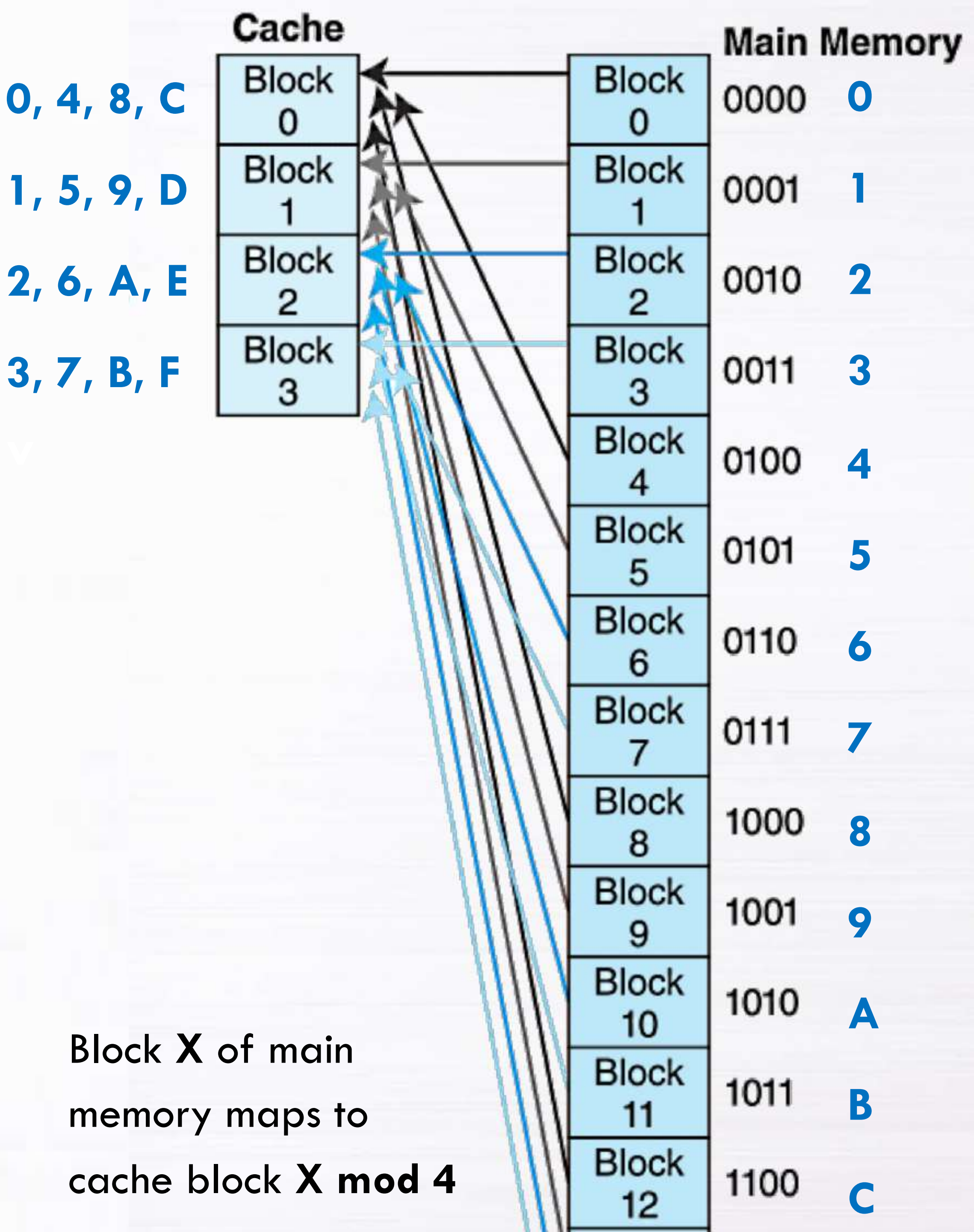

Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

Assume all cache blocks are empty. The computer accesses the memory addresses provided. For each accessed address, indicate the following:

- If value to be accessed is **in cache**, then write: **HIT**
- If value to be accessed is **not in cache** but can be loaded into an **unused cache block**, then write: **OPEN**
- If the value to be accessed is **not in cache** and **cache is full**, then write the **3-digit hexadecimal number of memory block to be evicted**

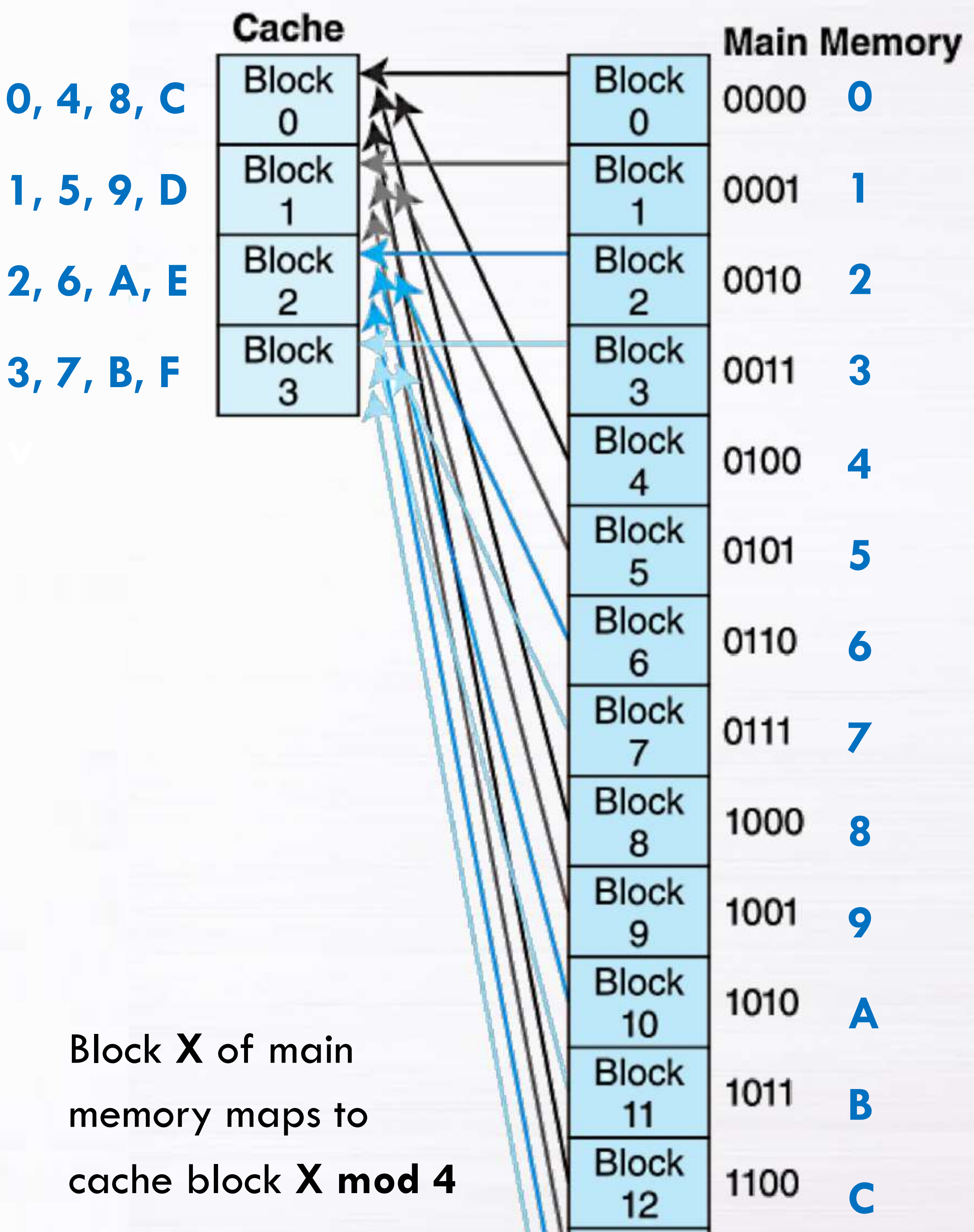



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

- ABBA
- C0DD
- F00D
- DEAD
- BEE5
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE




Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

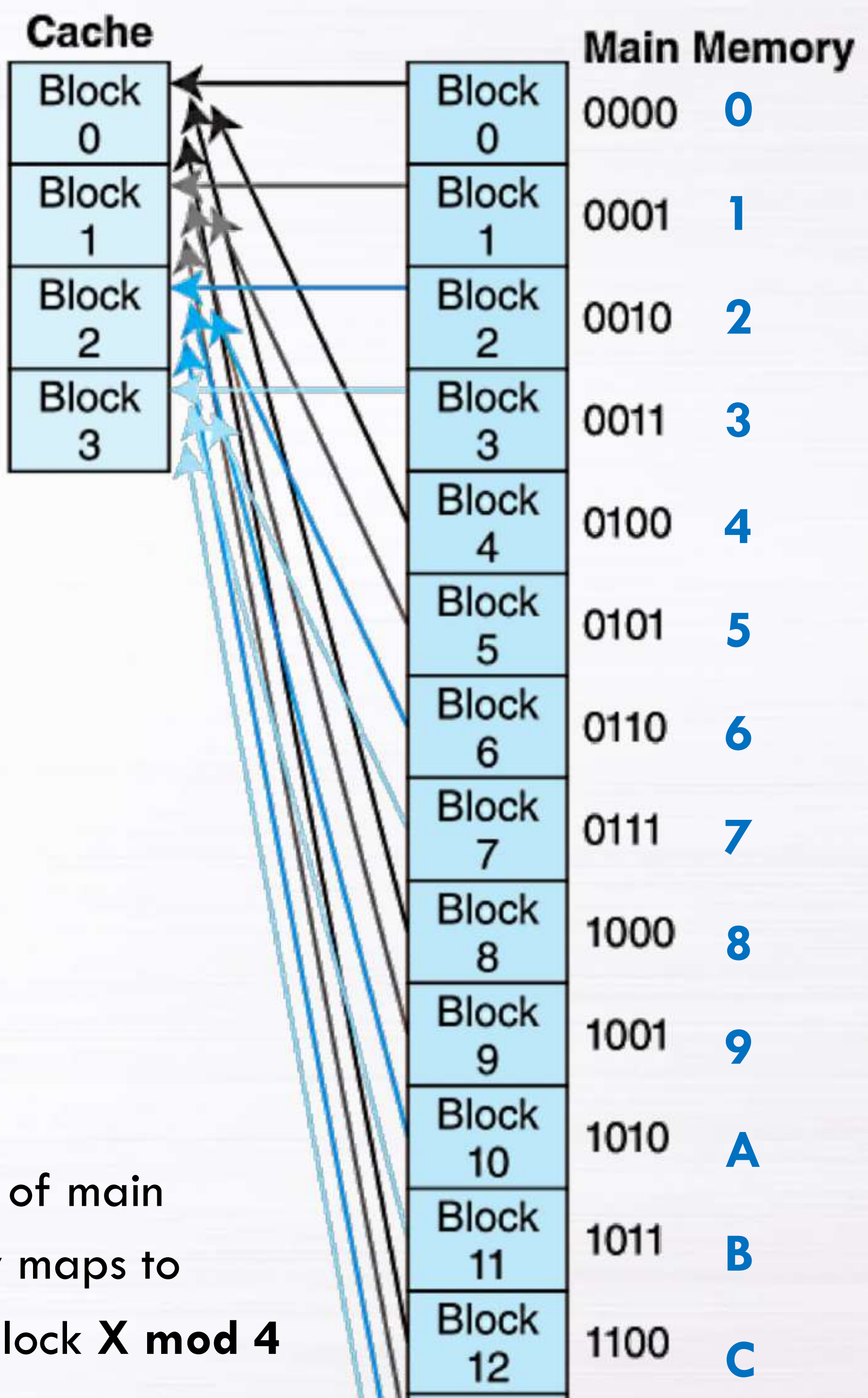
- ABBA
- C0DD
- F00D
- DEAD
- BEE5
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE

0, 4, 8, C

1, 5, 9, D

2, 6, A, E

3, 7, B, F

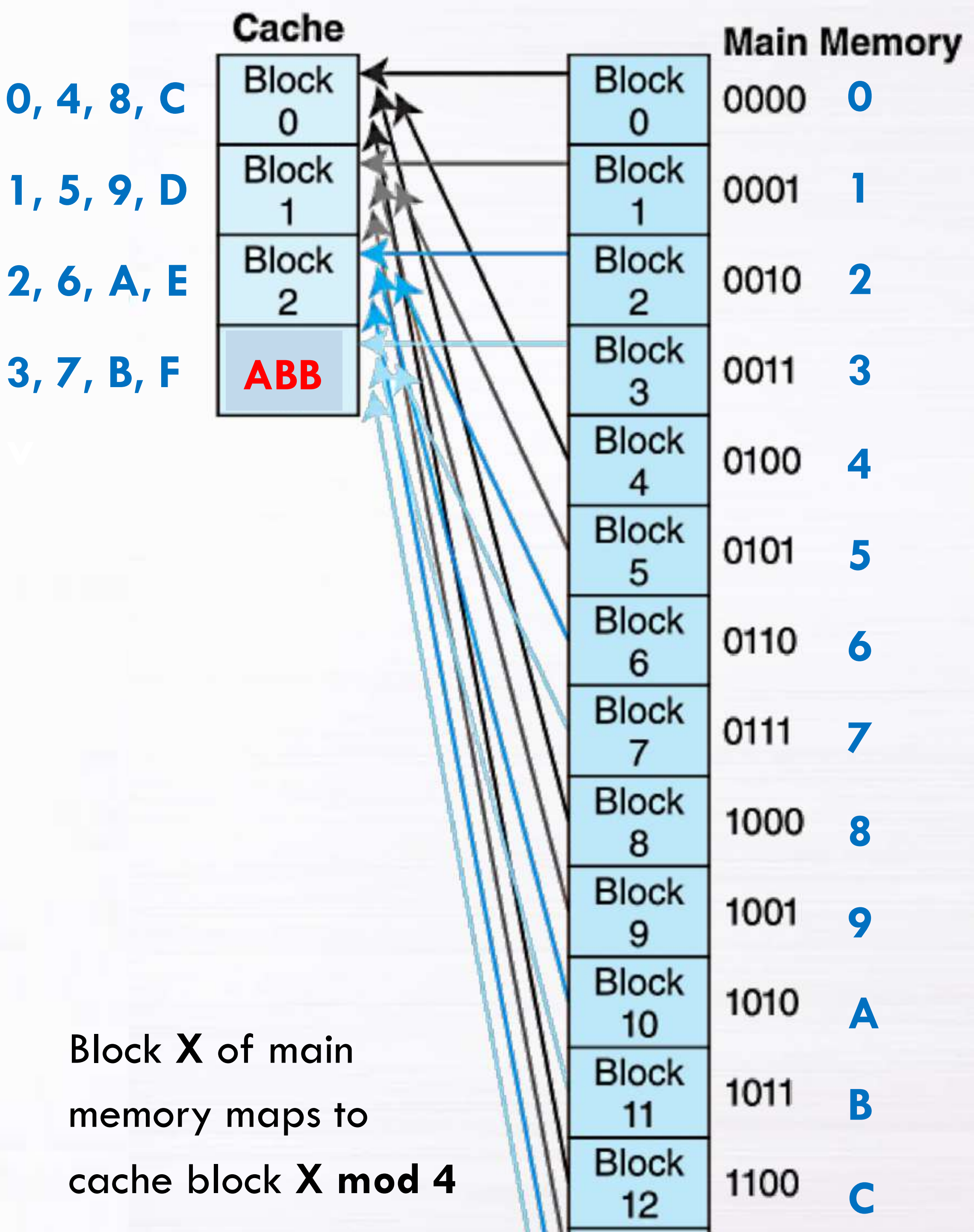



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

- ABBA OPEN
- C0DD
- F00D
- DEAD
- BEE5
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE

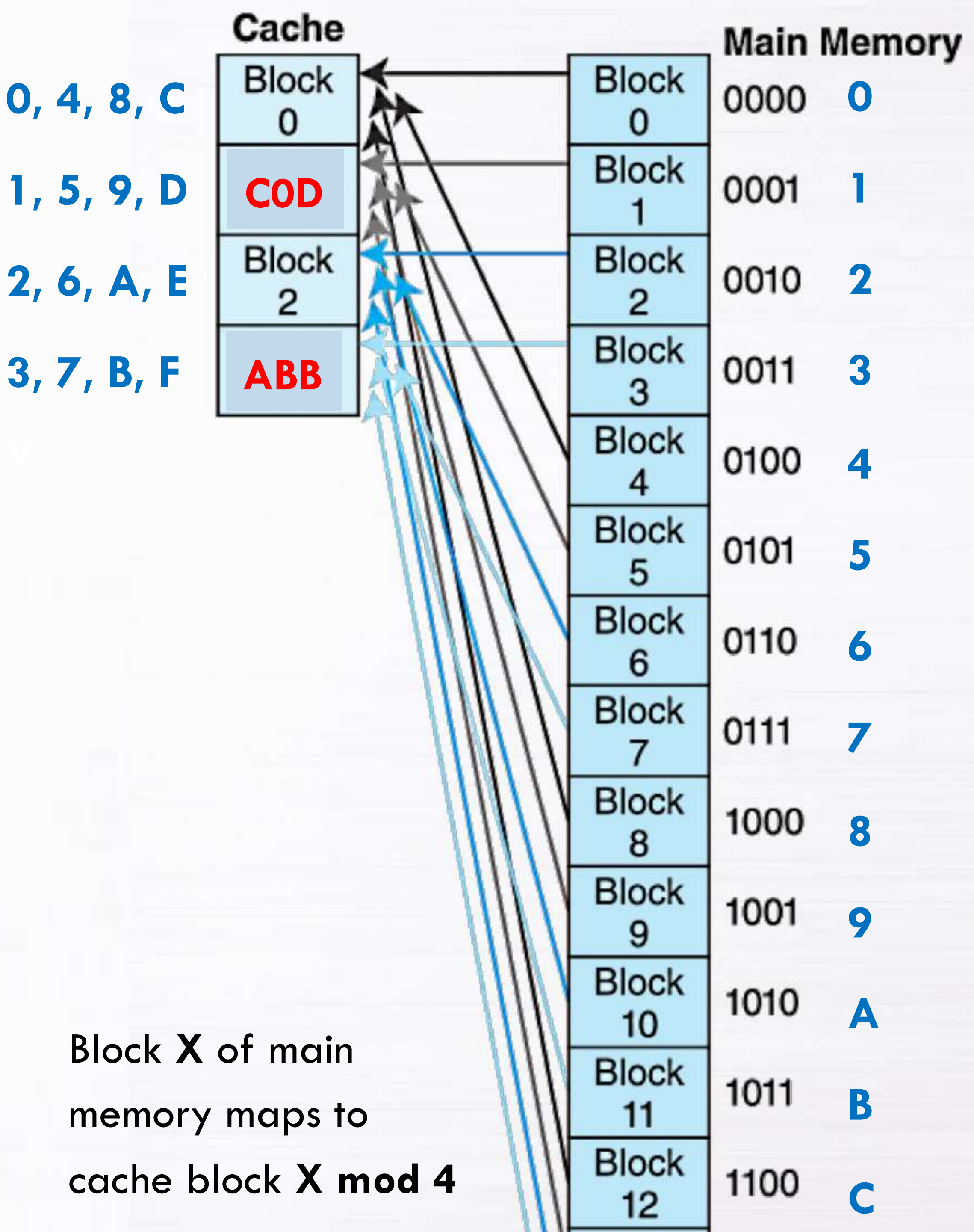



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

- ABBA OPEN
- C0DD OPEN
- F00D
- DEAD
- BEE5
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE

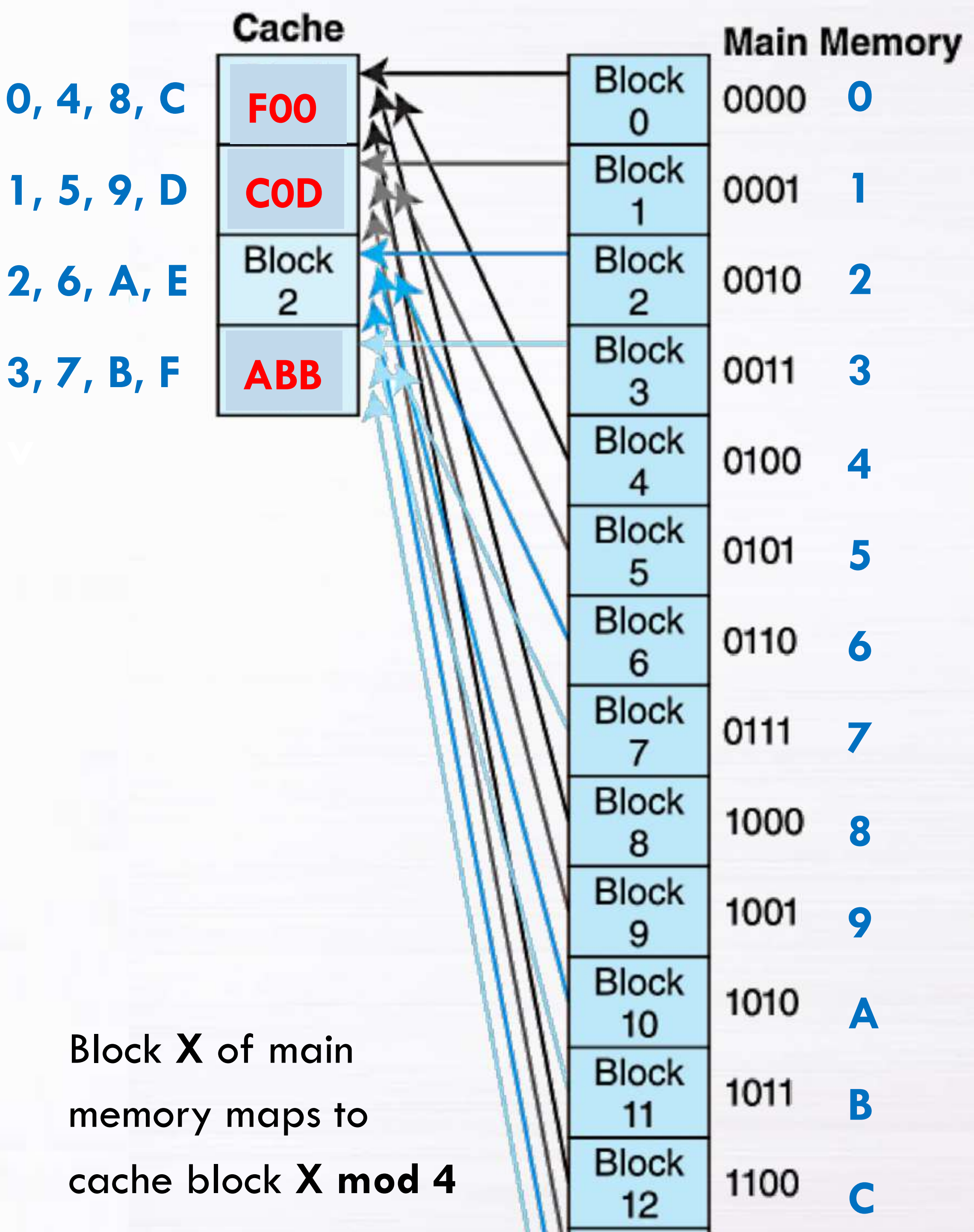



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

- ABBA OPEN
- C0DD OPEN
- F00D OPEN
- DEAD
- BEE5
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE

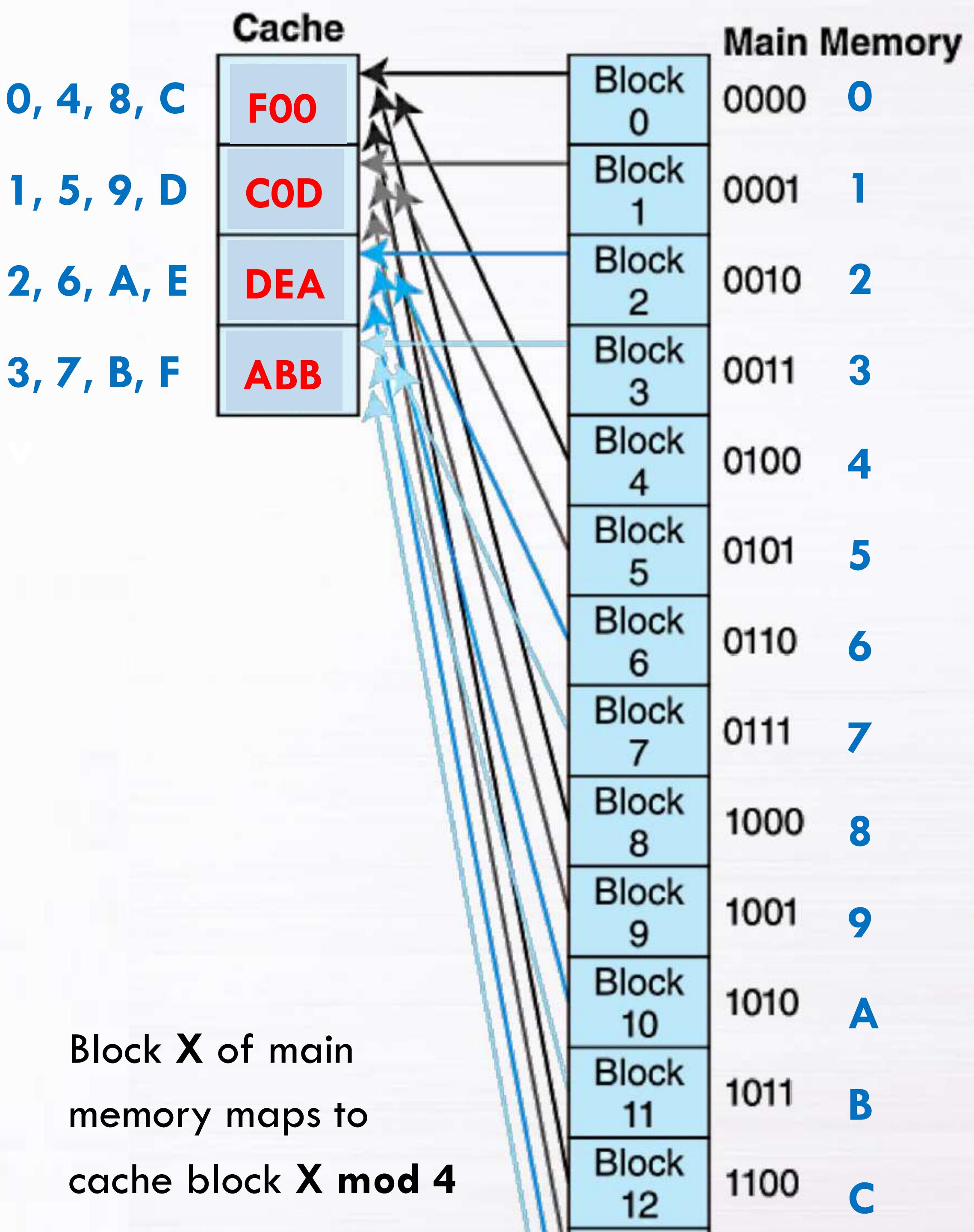



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

- ABBA OPEN
- C0DD OPEN
- F00D OPEN
- DEAD OPEN
- BEE5
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE

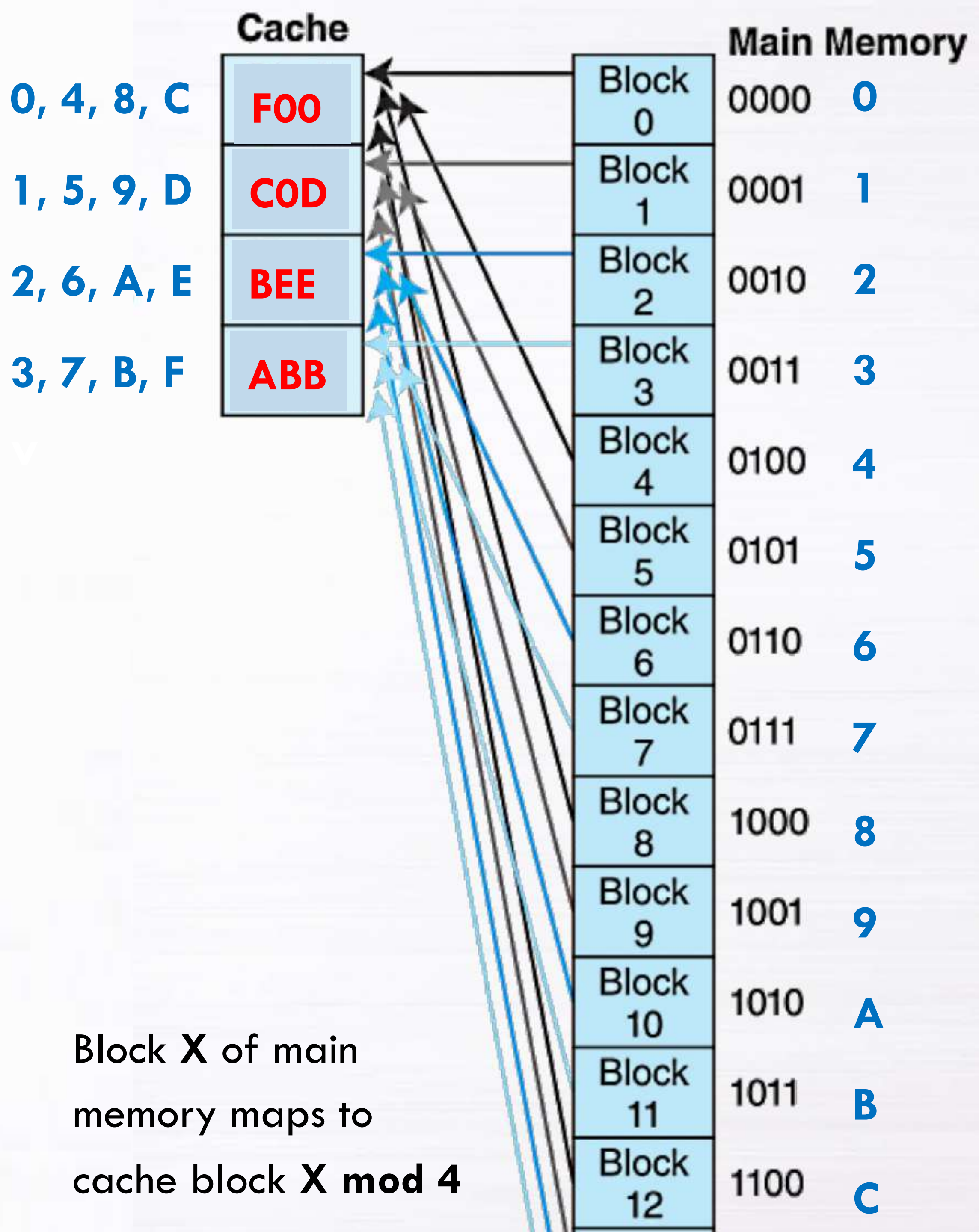



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

- ABBA OPEN
- C0DD OPEN
- F00D OPEN
- DEAD OPEN
- BEE5 DEA
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE

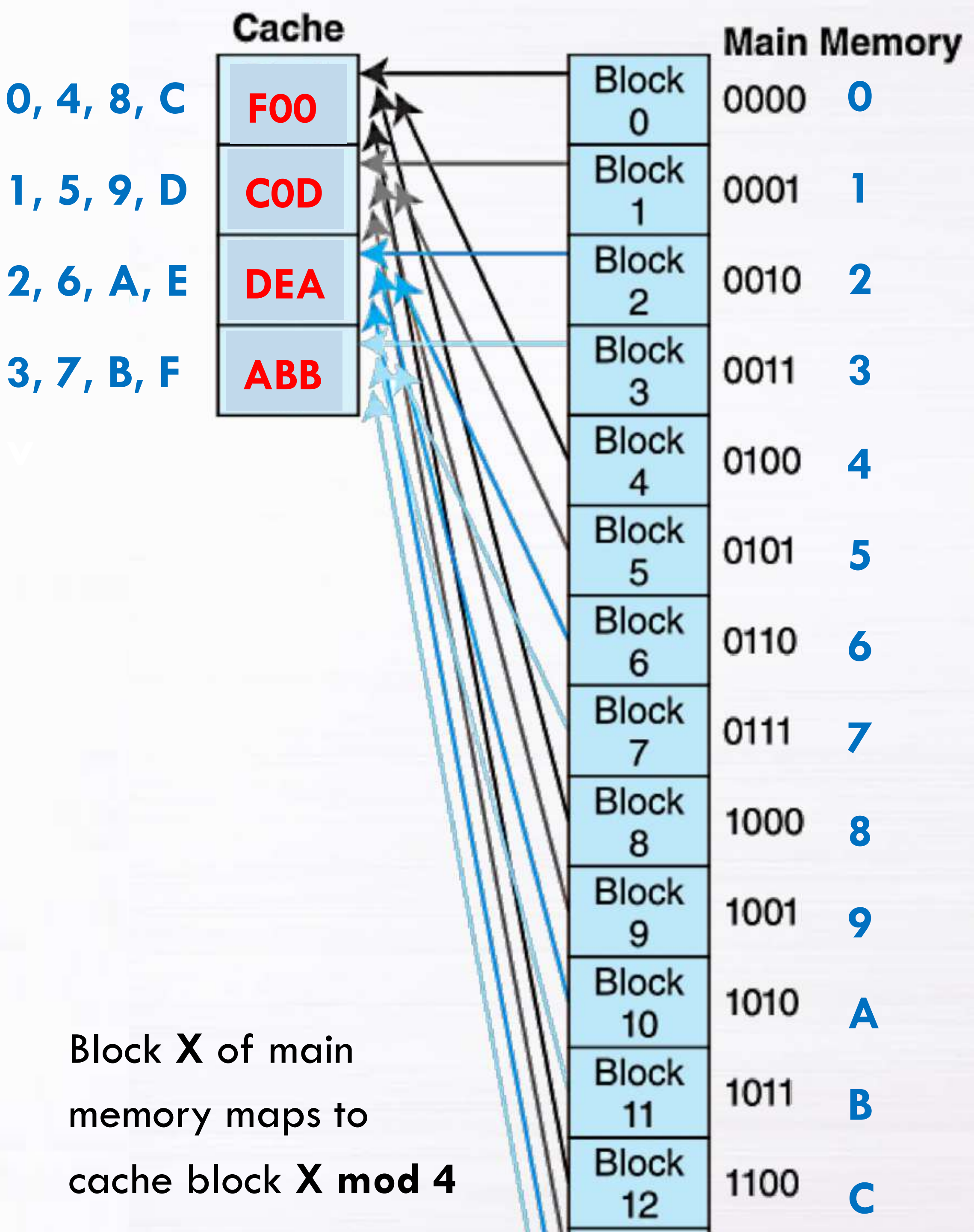



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

| | |
|---|---|
| • ABBA | OPEN |
| • C0DD | OPEN |
| • F00D | OPEN |
| • DEAD | OPEN |
| • BEE5 | DEA |
| • DEAF | BEE |
| • FEE5 | |
| • EAD5 | |
| • FEED | |
| • F00D | |
| • C0DE | |
| • FADE | |

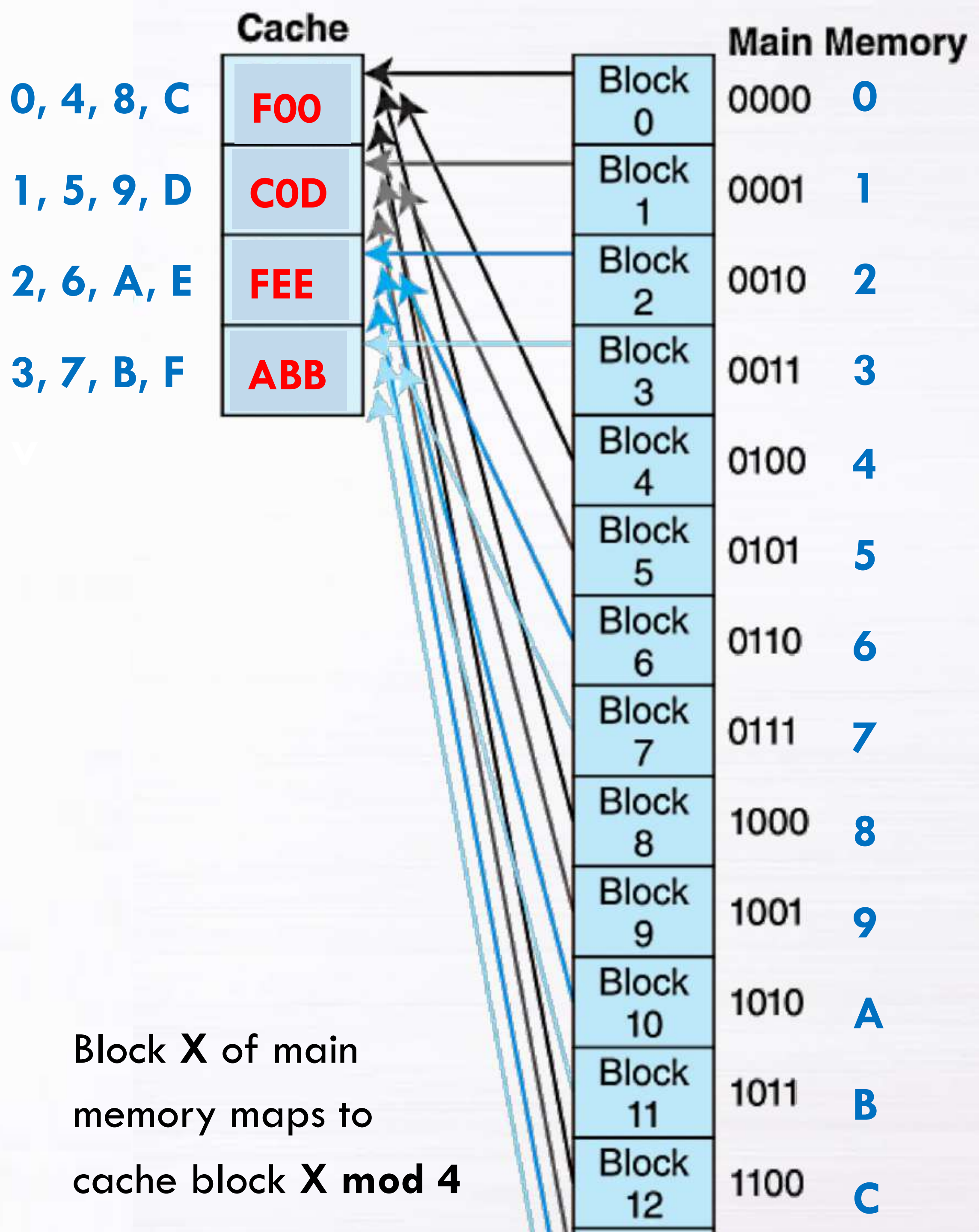



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

| | |
|---|---|
| • ABBA | OPEN |
| • C0DD | OPEN |
| • F00D | OPEN |
| • DEAD | OPEN |
| • BEE5 | DEA |
| • DEAF | BEE |
| • FEE5 | DEA |
| • EAD5 | |
| • FEED | |
| • F00D | |
| • C0DE | |
| • FADE | |

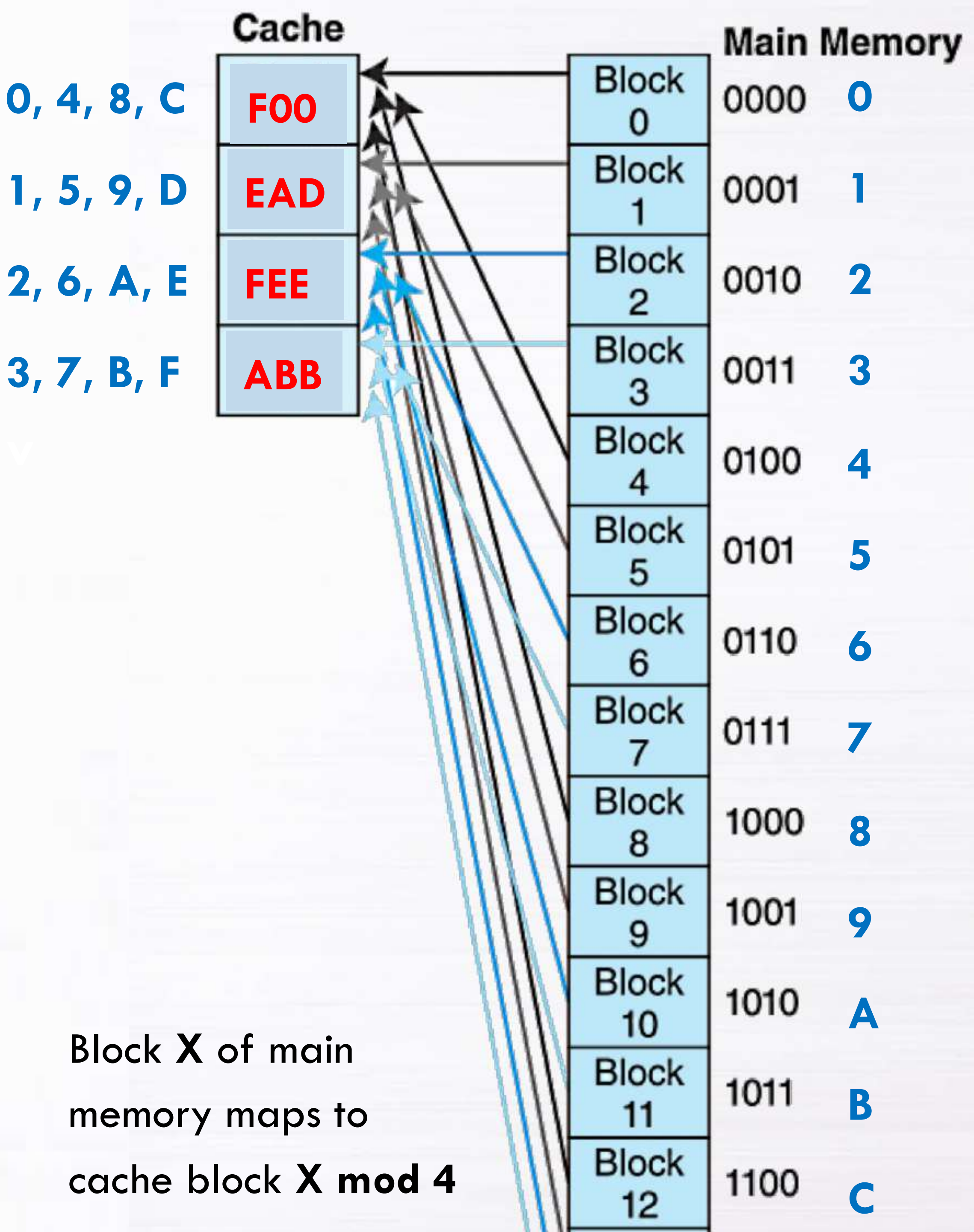



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

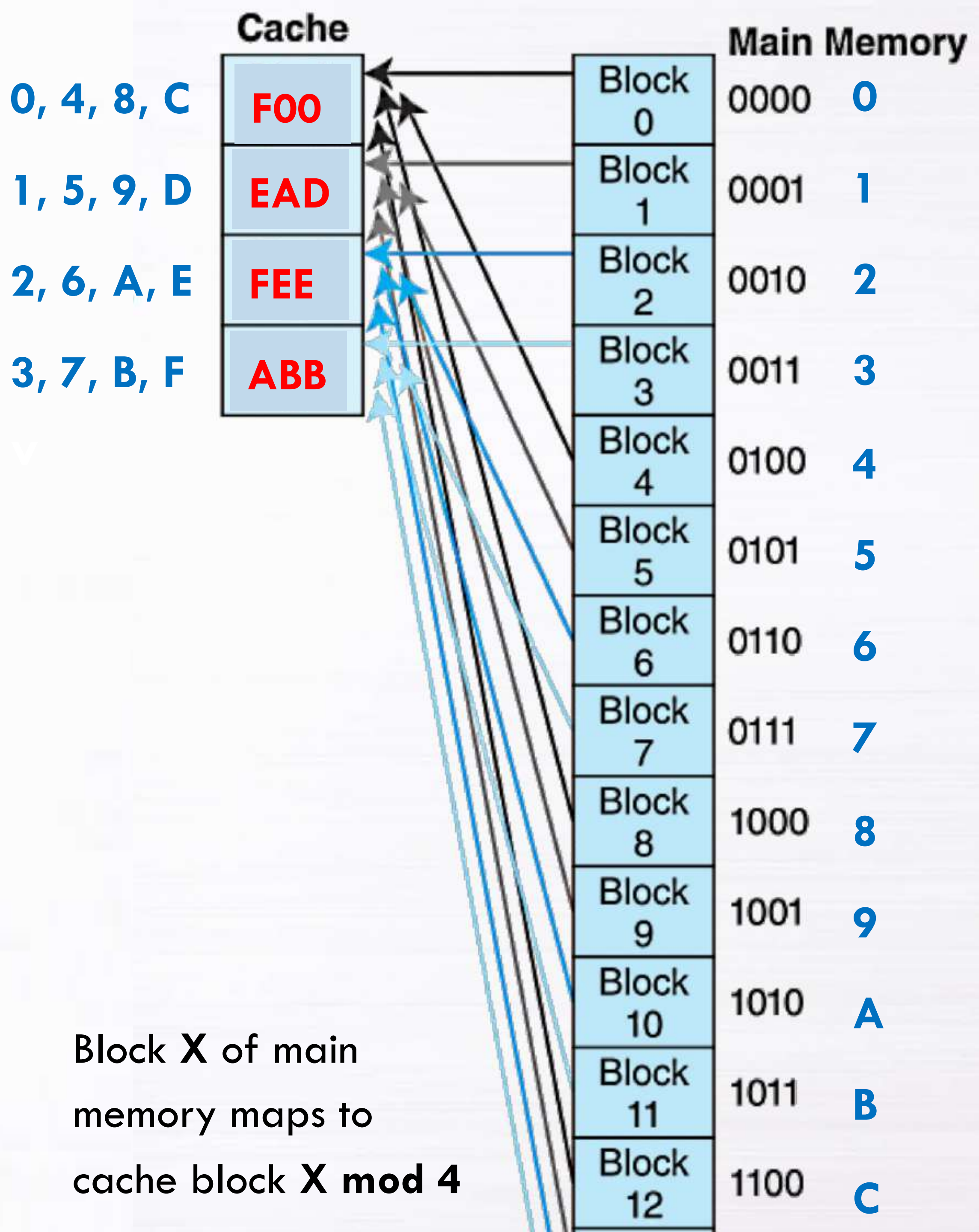
| | |
|---|---|
| • ABBA | OPEN |
| • C0DD | OPEN |
| • F00D | OPEN |
| • DEAD | OPEN |
| • BEE5 | DEA |
| • DEAF | BEE |
| • FEE5 | DEA |
| • EAD5 | C0D |
| • FEED | |
| • F00D | |
| • C0DE | |
| • FADE | |



| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

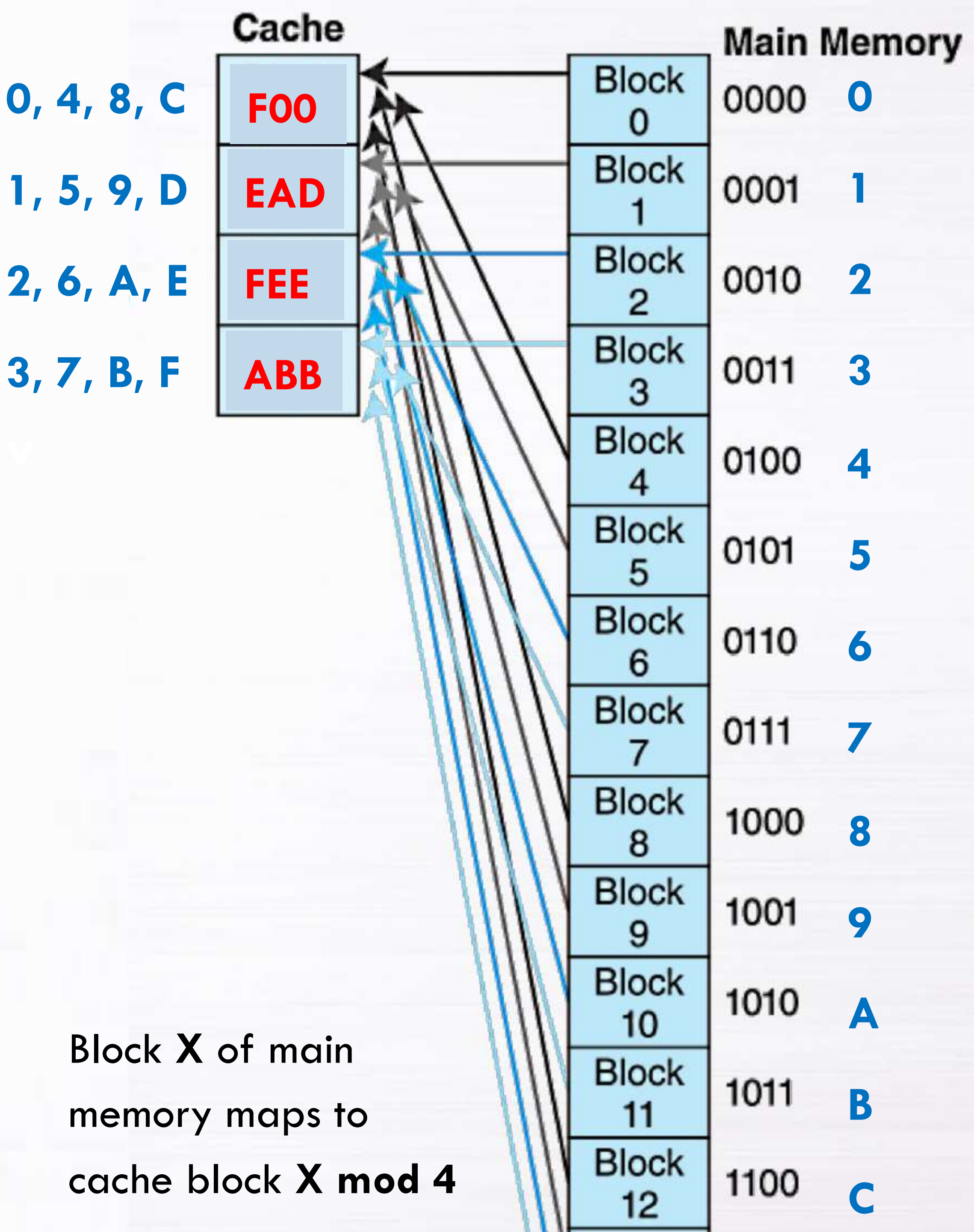
| | |
|---|---|
| • ABBA | OPEN |
| • C0DD | OPEN |
| • F00D | OPEN |
| • DEAD | OPEN |
| • BEE5 | DEA |
| • DEAF | BEE |
| • FEE5 | DEA |
| • EAD5 | C0D |
| • FEED | HIT |
| • F00D | |
| • C0DE | |
| • FADE | |



| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

| | |
|---|---|
| • ABBA | OPEN |
| • C0DD | OPEN |
| • F00D | OPEN |
| • DEAD | OPEN |
| • BEE5 | DEA |
| • DEAF | BEE |
| • FEE5 | DEA |
| • EAD5 | C0D |
| • FEED | HIT |
| • F00D | HIT |
| • C0DE | |
| • FADE | |

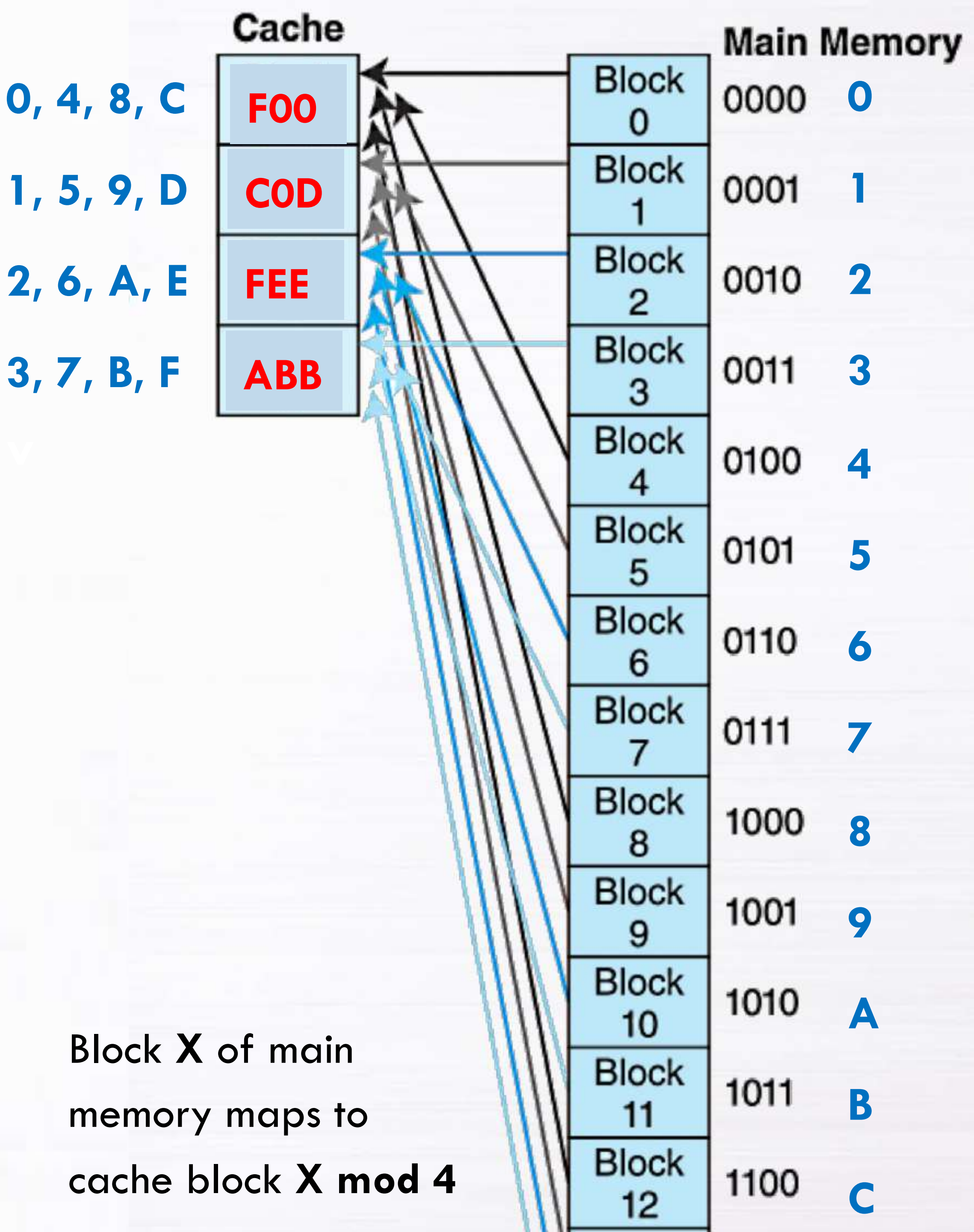



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

- ABBA OPEN
- C0DD OPEN
- F00D OPEN
- DEAD OPEN
- BEE5 DEA
- DEAF BEE
- FEE5 DEA
- EAD5 C0D
- FEED HIT
- F00D HIT
- C0DE EAD
- FADE

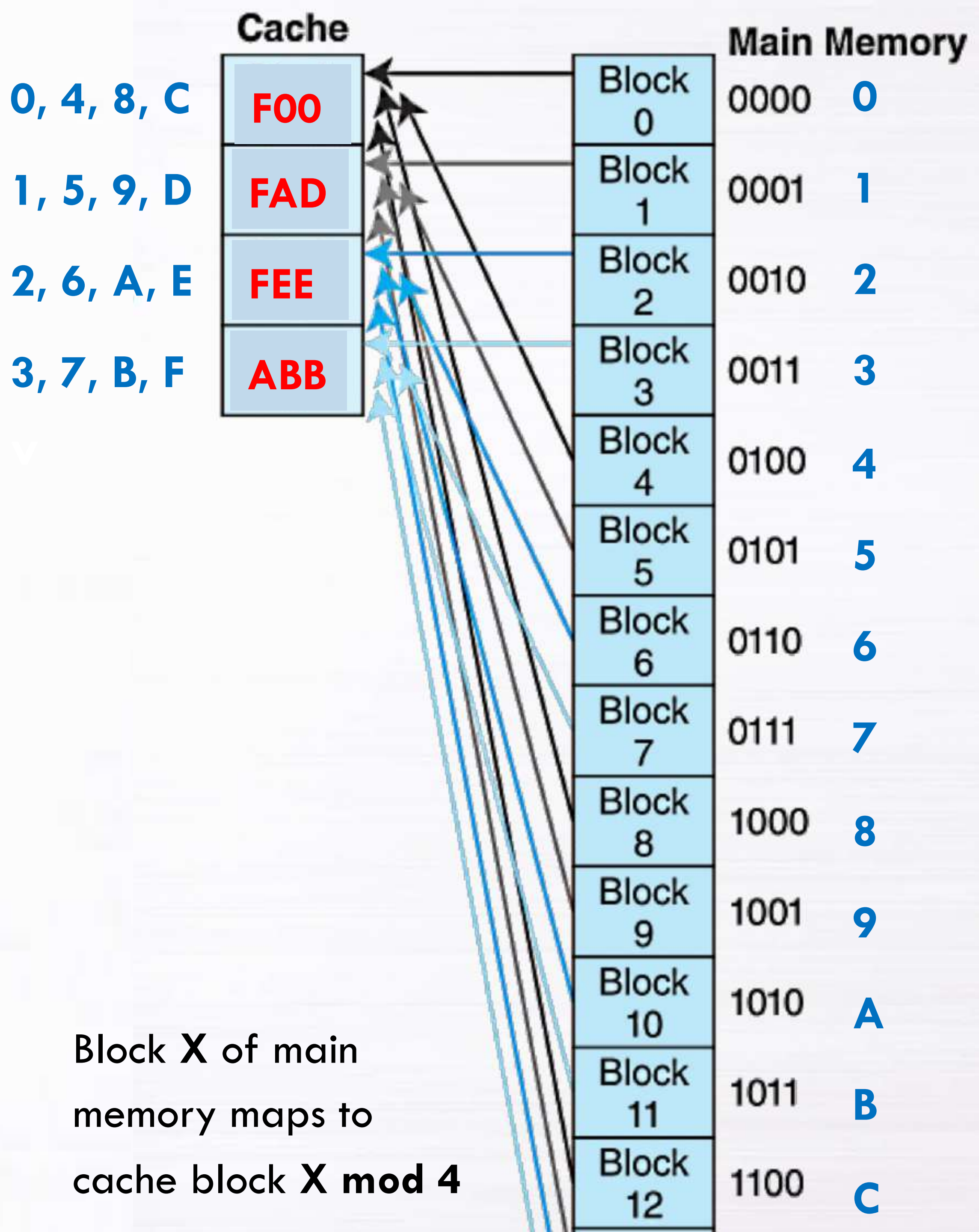



Block **X** of main memory maps to cache block **X mod 4**

| 10 bits | 2 bits | 4 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 16 bits → | | |

# DIRECT MAPPED CACHING

- ABBA OPEN
- C0DD OPEN
- F00D OPEN
- DEAD OPEN
- BEE5 DEA
- DEAF BEE
- FEE5 DEA
- EAD5 C0D
- FEED HIT
- F00D HIT
- C0DE EAD
- FADE C0D




Block **X** of main memory maps to cache block **X mod 4**

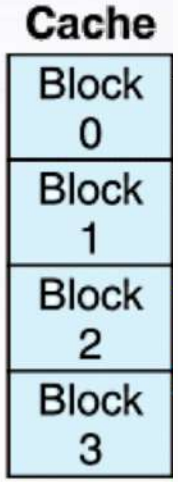
# FULLY ASSOCIATIVE CACHING

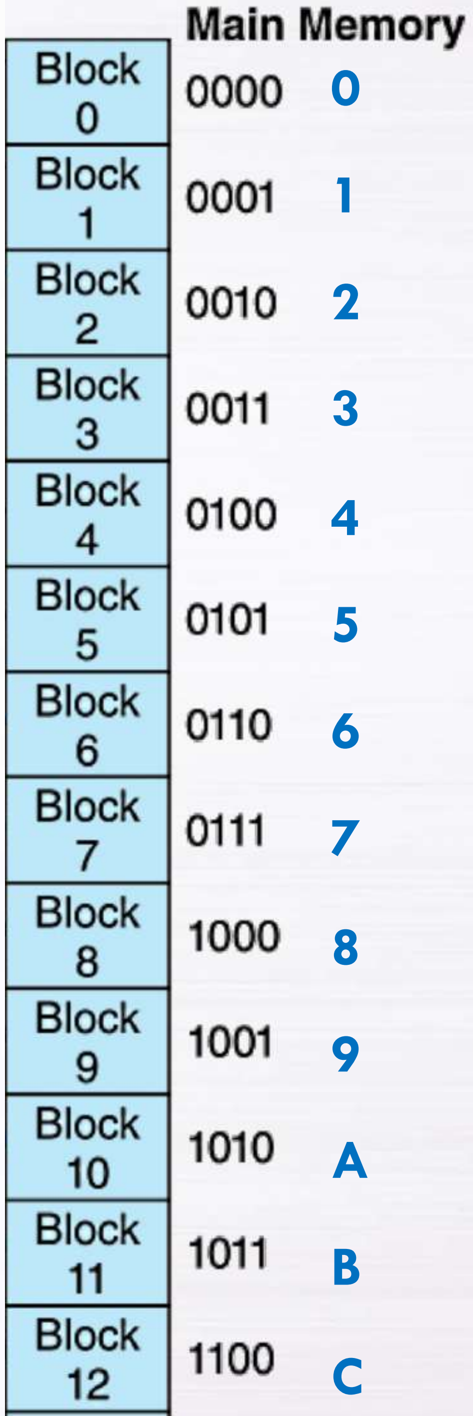
- Assume a computer with memory of 2^16 bytes and a **fully associative cache** with **4 blocks** where each block has 16 bytes. A **FIFO approach for block replacement** is used.
- Each **memory address consists of 16 bits** of which the rightmost 4 bits reflect the offset field and the leftmost 12 bits (tag) specify the number of the associated memory block.

| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

| 12 bits | 4 bits |
| --- | --- |
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING



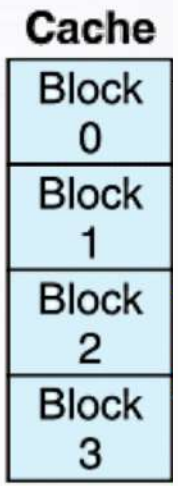

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.

| 12 bits | 4 bits |
| --- | --- |
| Tag | Offset |
| ← 16 bits → | |

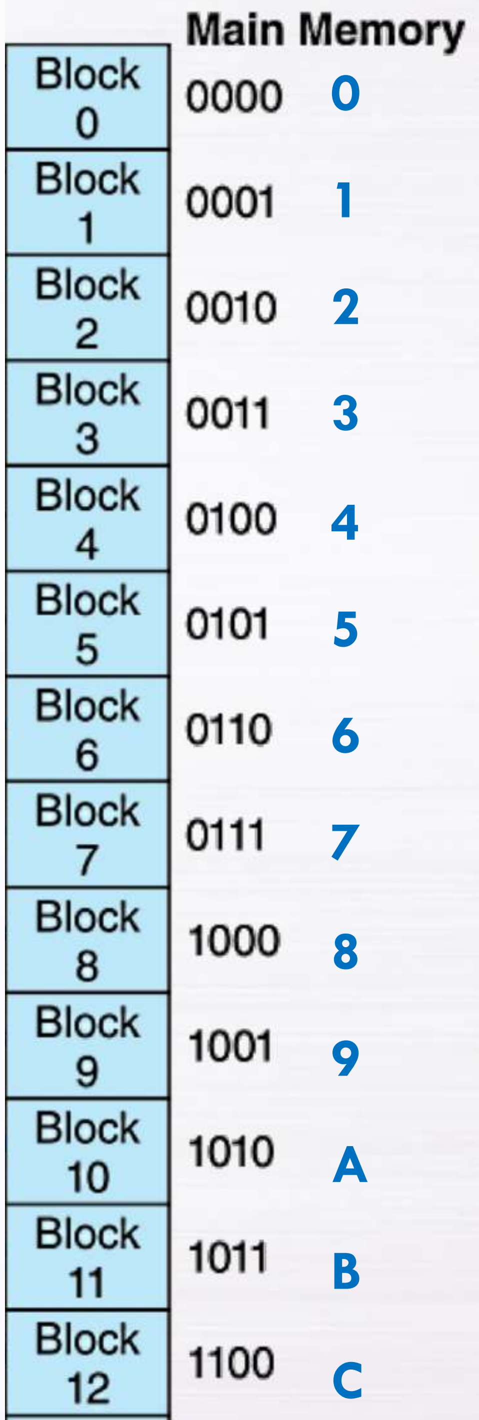
# FULLY ASSOCIATIVE CACHING

Assume all cache blocks are empty. The computer accesses the memory addresses provided. For each accessed address, indicate the following:

- If value to be accessed is **in cache**, then write: **HIT**
- If value to be accessed is **not in cache** but can be loaded into an **unused cache block**, then write: **OPEN**
- If the value to be accessed is **not in cache** and **cache is full**, then write the **3-digit hexadecimal number of memory block to be evicted**



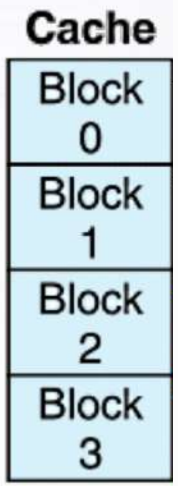

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.



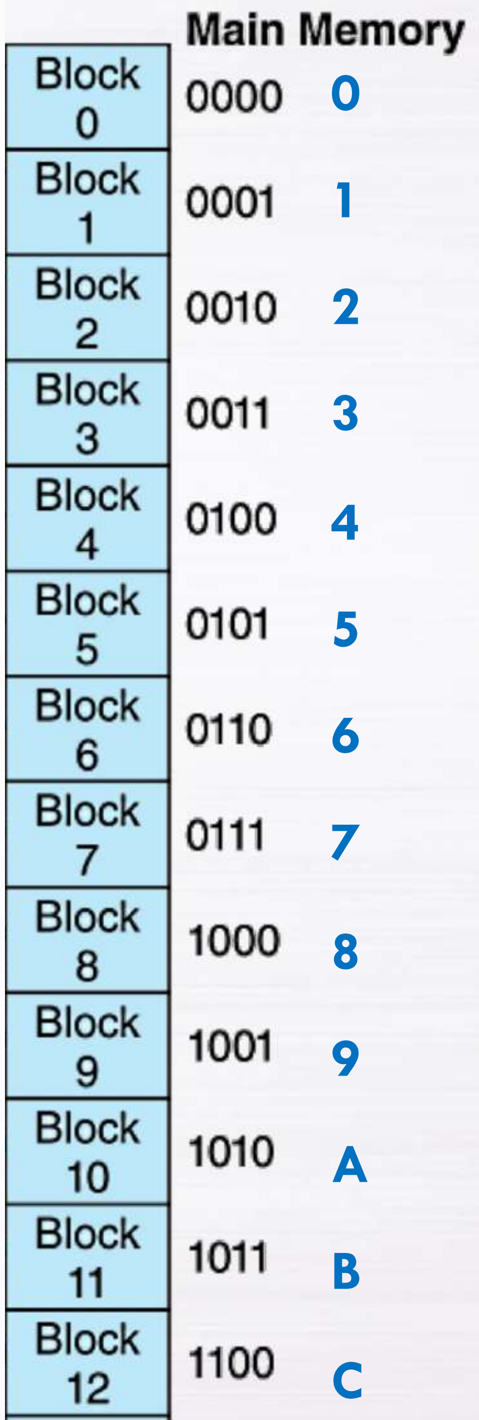
| | |
|---|---|
| 12 bits | 4 bits |
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

- ABBA
- C0DD
- F00D
- DEAD
- BEE5
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE




- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.



| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

- ABBA OPEN
- C0DD OPEN
- F00D OPEN
- DEAD OPEN
- BEE5
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE

**Cache**

| |
|---|
| ABB |
| C0D |
| F00 |
| DEA |

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.

**Main Memory**

| | | |
|---|---|---|
| Block 0 | 0000 | 0 |
| Block 1 | 0001 | 1 |
| Block 2 | 0010 | 2 |
| Block 3 | 0011 | 3 |
| Block 4 | 0100 | 4 |
| Block 5 | 0101 | 5 |
| Block 6 | 0110 | 6 |
| Block 7 | 0111 | 7 |
| Block 8 | 1000 | 8 |
| Block 9 | 1001 | 9 |
| Block 10 | 1010 | A |
| Block 11 | 1011 | B |
| Block 12 | 1100 | C |

| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

- ABBA OPEN
- C0DD OPEN
- F00D OPEN
- DEAD OPEN
- BEE5 ABB
- DEAF
- FEE5
- EAD5
- FEED
- F00D
- C0DE
- FADE

| Cache |
|---|
| BEE |
| C0D |
| F00 |
| DEA |

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.

| | Main Memory | |
|---|---|---|
| Block 0 | 0000 | 0 |
| Block 1 | 0001 | 1 |
| Block 2 | 0010 | 2 |
| Block 3 | 0011 | 3 |
| Block 4 | 0100 | 4 |
| Block 5 | 0101 | 5 |
| Block 6 | 0110 | 6 |
| Block 7 | 0111 | 7 |
| Block 8 | 1000 | 8 |
| Block 9 | 1001 | 9 |
| Block 10 | 1010 | A |
| Block 11 | 1011 | B |
| Block 12 | 1100 | C |

| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

| | |
|---|---|
| ABBA | OPEN |
| C0DD | OPEN |
| F00D | OPEN |
| DEAD | OPEN |
| BEE5 | ABB |
| DEAF | HIT |
| FEE5 | |
| EAD5 | |
| FEED | |
| F00D | |
| C0DE | |
| FADE | |

**Cache**

| |
|---|
| BEE |
| C0D |
| F00 |
| DEA |

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.

**Main Memory**

| | | |
|---|---|---|
| Block 0 | 0000 | 0 |
| Block 1 | 0001 | 1 |
| Block 2 | 0010 | 2 |
| Block 3 | 0011 | 3 |
| Block 4 | 0100 | 4 |
| Block 5 | 0101 | 5 |
| Block 6 | 0110 | 6 |
| Block 7 | 0111 | 7 |
| Block 8 | 1000 | 8 |
| Block 9 | 1001 | 9 |
| Block 10 | 1010 | A |
| Block 11 | 1011 | B |
| Block 12 | 1100 | C |

| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

| | |
|---|---|
| • ABBA | OPEN |
| • C0DD | OPEN |
| • F00D | OPEN |
| • DEAD | OPEN |
| • BEE5 | ABB |
| • DEAF | HIT |
| • FEE5 | C0D |
| • EAD5 | |
| • FEED | |
| • F00D | |
| • C0DE | |
| • FADE | |

**Cache**

| |
|---|
| BEE |
| FEE |
| F00 |
| DEA |

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.

**Main Memory**

| | | |
|---|---|---|
| Block 0 | 0000 | 0 |
| Block 1 | 0001 | 1 |
| Block 2 | 0010 | 2 |
| Block 3 | 0011 | 3 |
| Block 4 | 0100 | 4 |
| Block 5 | 0101 | 5 |
| Block 6 | 0110 | 6 |
| Block 7 | 0111 | 7 |
| Block 8 | 1000 | 8 |
| Block 9 | 1001 | 9 |
| Block 10 | 1010 | A |
| Block 11 | 1011 | B |
| Block 12 | 1100 | C |

| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

| | |
|---|---|
| ABBA | OPEN |
| C0DD | OPEN |
| F00D | OPEN |
| DEAD | OPEN |
| BEE5 | ABB |
| DEAF | HIT |
| FEE5 | C0D |
| EAD5 | F00 |
| FEED | |
| F00D | |
| C0DE | |
| FADE | |

**Cache**

| |
|---|
| BEE |
| FEE |
| EAD |
| DEA |

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.

**Main Memory**

| | | |
|---|---|---|
| Block 0 | 0000 | 0 |
| Block 1 | 0001 | 1 |
| Block 2 | 0010 | 2 |
| Block 3 | 0011 | 3 |
| Block 4 | 0100 | 4 |
| Block 5 | 0101 | 5 |
| Block 6 | 0110 | 6 |
| Block 7 | 0111 | 7 |
| Block 8 | 1000 | 8 |
| Block 9 | 1001 | 9 |
| Block 10 | 1010 | A |
| Block 11 | 1011 | B |
| Block 12 | 1100 | C |

| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

| | |
|---|---|
| ABBA | OPEN |
| C0DD | OPEN |
| F00D | OPEN |
| DEAD | OPEN |
| BEE5 | ABB |
| DEAF | HIT |
| FEE5 | C0D |
| EAD5 | F00 |
| FEED | HIT |
| F00D | |
| C0DE | |
| FADE | |

| Cache |
|---|
| BEE |
| FEE |
| EAD |
| DEA |

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.

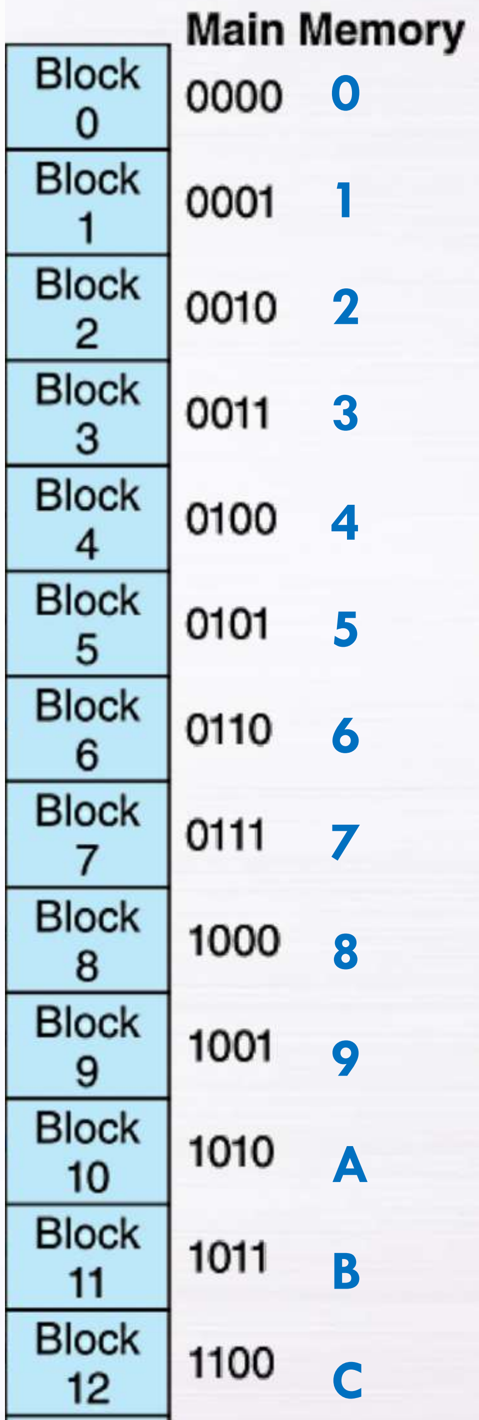
| Main Memory | | |
|---|---|---|
| Block 0 | 0000 | 0 |
| Block 1 | 0001 | 1 |
| Block 2 | 0010 | 2 |
| Block 3 | 0011 | 3 |
| Block 4 | 0100 | 4 |
| Block 5 | 0101 | 5 |
| Block 6 | 0110 | 6 |
| Block 7 | 0111 | 7 |
| Block 8 | 1000 | 8 |
| Block 9 | 1001 | 9 |
| Block 10 | 1010 | A |
| Block 11 | 1011 | B |
| Block 12 | 1100 | C |

| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

| | |
|---|---|
| • ABBA | OPEN |
| • C0DD | OPEN |
| • F00D | OPEN |
| • DEAD | OPEN |
| • BEE5 | ABB |
| • DEAF | HIT |
| • FEE5 | C0D |
| • EAD5 | F00 |
| • FEED | HIT |
| • F00D | DEA |
| • C0DE | |
| • FADE | |


- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.



| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

- ABBA — OPEN
- C0DD — OPEN
- F00D — OPEN
- DEAD — OPEN
- BEE5 — ABB
- DEAF — HIT
- FEE5 — C0D
- EAD5 — F00
- FEED — HIT
- F00D — DEA
- C0DE — BEE
- FADE

**Cache**

| |
|---|
| C0D |
| FEE |
| EAD |
| F00 |

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.

**Main Memory**

| | | |
|---|---|---|
| Block 0 | 0000 | 0 |
| Block 1 | 0001 | 1 |
| Block 2 | 0010 | 2 |
| Block 3 | 0011 | 3 |
| Block 4 | 0100 | 4 |
| Block 5 | 0101 | 5 |
| Block 6 | 0110 | 6 |
| Block 7 | 0111 | 7 |
| Block 8 | 1000 | 8 |
| Block 9 | 1001 | 9 |
| Block 10 | 1010 | A |
| Block 11 | 1011 | B |
| Block 12 | 1100 | C |

| 12 bits | 4 bits |
|---|---|
| Tag | Offset |
| ← 16 bits → | |

# FULLY ASSOCIATIVE CACHING

| | |
|---|---|
| ABBA | OPEN |
| C0DD | OPEN |
| F00D | OPEN |
| DEAD | OPEN |
| BEE5 | ABB |
| DEAF | HIT |
| FEE5 | C0D |
| EAD5 | F00 |
| FEED | HIT |
| F00D | DEA |
| C0DE | BEE |
| FADE | FEE |

Cache

| |
|---|
| C0D |
| FAD |
| EAD |
| F00 |

- Copy data into the next unused cache block.
- If cache is full, evict memory block that has been in the cache for the longest time.

Main Memory

| | | |
|---|---|---|
| Block 0 | 0000 | 0 |
| Block 1 | 0001 | 1 |
| Block 2 | 0010 | 2 |
| Block 3 | 0011 | 3 |
| Block 4 | 0100 | 4 |
| Block 5 | 0101 | 5 |
| Block 6 | 0110 | 6 |
| Block 7 | 0111 | 7 |
| Block 8 | 1000 | 8 |
| Block 9 | 1001 | 9 |
| Block 10 | 1010 | A |
| Block 11 | 1011 | B |
| Block 12 | 1100 | C |

# CACHING I

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **direct mapped caching**

  with **2 kilowords per block**
- The cache consists of **8 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| | | |
|---|---|---|
| Tag | Block | Offset |
| ← 32 bits → | | |

# CACHING I

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **direct mapped caching**

  with **2 kilowords per block = 2^11 words per block**
- The cache consists of **8 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| | | |
|---|---|---|
| Tag | Block | Offset |
| ← 32 bits → | | |

# CACHING I

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **direct mapped caching**

  with **2 kilowords per block = $2^{11}$ words per block**
- The cache consists of **8 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| | | |
|---|---|---|
| | | **11 bits** |
| Tag | Block | Offset |
| ← 32 bits → | | |

# CACHING I

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **direct mapped caching**

  with **2 kilowords per block = 2^11 words per block**
- The cache consists of **8 blocks = 2^3 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| | | **11 bits** |
|---|---|---|
| Tag | Block | Offset |
| ← 32 bits → | | |

# CACHING I

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **direct mapped caching**

  with **2 kilowords per block = 2^11 words per block**
- The cache consists of **8 blocks = 2^3 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| | 3 bits | 11 bits |
| --- | --- | --- |
| Tag | Block | Offset |
| ← 32 bits → | | |

# CACHING I

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **direct mapped caching**

  with **2 kilowords per block = 2^11 words per block**
- The cache consists of **8 blocks = 2^3 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| 32 - 14 bits | 3 bits | 11 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 32 bits → | | |

# CACHING I

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **direct mapped caching**

  with **2 kilowords per block = 2^11 words per block**
- The cache consists of **8 blocks = 2^3 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| 18 bits | 3 bits | 11 bits |
|---|---|---|
| Tag | Block | Offset |
| ← 32 bits → | | |

# CACHING II

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **fully associative caching**

  with **1 kiloword per block**
- The cache consists of **16 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| | |
|---|---|
| Tag | Offset |
| ← 32 bits → | |

# CACHING II

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **fully associative caching**

  with **1 kiloword per block = 2^10 words per block**
- The cache consists of **16 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| | |
|---|---|
| Tag | Offset |
| ← 32 bits → | |

# CACHING II

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **fully associative caching**

  with **1 kiloword per block = 2^10 words per block**
- The cache consists of **16 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| | |
|---|---|
| | 10 bits |
| Tag | Offset |
| ← 32 bits → | |

# CACHING II

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **fully associative caching**

  with **1 kiloword per block = 2^10 words per block**
- The cache consists of **16 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| 32 – 10 bits | 10 bits |
|---|---|
| Tag | Offset |
| ← 32 bits → | |

# CACHING II

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **fully associative caching**

  with **1 kiloword per block = 2^10 words per block**
- The cache consists of **16 blocks**
- How many bits are then allocated to the **tag** and the **offset**?

| 22 bits | 10 bits |
| --- | --- |
| Tag | Offset |
| ← 32 bits → | |

# CACHING III

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **4-way set associative mapping** with **2 kilowords per block**
- The cache consists of **32 blocks**
- How many bits are then allocated to the **tag** and the **set**?

| | | |
|---|---|---|
| | | |
| Tag | Set | Offset |
| ← 32 bits → | | |

# CACHING III

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **4-way set associative mapping**

  with **2 kilowords per block = 2^11 words per block**
- The cache consists of **32 blocks**
- How many bits are then allocated to the **tag** and the **set**?

| | | |
|---|---|---|
| | | **11 bits** |
| Tag | Set | Offset |
| ← 32 bits → | | |

# CACHING III

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **4-way set associative mapping**

  with **2 kilowords per block = 2^11 words per block**
- The cache consists of **32 blocks → 8 sets**
- How many bits are then allocated to the **tag** and the **set**?

| | | |
|---|---|---|
| | | **11 bits** |
| Tag | Set | Offset |
| ← 32 bits → | | |

# CACHING III

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **4-way set associative mapping**

  with **2 kilowords per block = 2^11 words per block**
- The cache consists of **32 blocks → 8 sets = 2^3 sets**
- How many bits are then allocated to the **tag** and the **set**?

| | **3 bits** | **11 bits** |
|---|---|---|
| Tag | Set | Offset |
| ← 32 bits → | | |

# CACHING III

- Assume a word addressable computer that uses **32-bit addresses**
- The computer uses **4-way set associative mapping**

  with **2 kilowords per block = 2^11 words per block**
- The cache consists of **32 blocks → 8 sets = 2^3 sets**
- How many bits are then allocated to the **tag** and the **set**?

| 18 bits | 3 bits | 11 bits |
|---|---|---|
| Tag | Set | Offset |
| ← 32 bits → | | |

# AMDAHL'S LAW

- Assume a system where processes on **average spend 65%** of their time being processed in the **CPU**.
- The system was upgraded by a **new CPU** that replaced an old CPU.
- The upgrade resulted in an **overall system speed-up of 15%**
- **How much faster is the new CPU** in comparison to the old CPU?

# AMDAHL'S LAW

- Assume a system where processes on **average spend 65%** of their time being processed in the **CPU**.
- The system was upgraded by a **new CPU** that replaced an old CPU.
- The upgrade resulted in an **overall system speed-up of 15%**
- **How much faster is the new CPU** in comparison to the old CPU?

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where $S$ is the overall speedup;
$f$ is the fraction of work performed by a faster component; and
$k$ is the speedup of the faster component.

# AMDAHL'S LAW

- Assume a system where processes on **average spend 65%** of their time being processed in the **CPU**.
- The system was upgraded by a **new CPU** that replaced an old CPU.
- The upgrade resulted in an **overall system speed-up of 15%**
- **How much faster is the new CPU** in comparison to the old CPU?

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where $S$ is the overall speedup;
$f$ is the fraction of work performed by a faster component; and
$k$ is the speedup of the faster component.

**S = 1.15**

# AMDAHL'S LAW

- Assume a system where processes on **average spend 65%** of their time being processed in the **CPU**.
- The system was upgraded by a **new CPU** that replaced an old CPU.
- The upgrade resulted in an **overall system speed-up of 15%**
- **How much faster is the new CPU** in comparison to the old CPU?

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where $S$ is the overall speedup; $f$ is the fraction of work performed by a faster component; and $k$ is the speedup of the faster component.

$S = 1.15$

$f = 0.65$

# AMDAHL'S LAW

- Assume a system where processes on **average spend 65%** of their time being processed in the **CPU**.
- The system was upgraded by a **new CPU** that replaced an old CPU.
- The upgrade resulted in an **overall system speed-up of 15%**
- **How much faster is the new CPU** in comparison to the old CPU?

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where $S$ is the overall speedup;
$f$ is the fraction of work performed by a faster component; and
$k$ is the speedup of the faster component.

$S = 1.15$

$f = 0.65$

$k = ?$

# AMDAHL'S LAW

$$k = \frac{f}{\left(\frac{1}{S} - (1-f)\right)}$$

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where $S$ is the overall speedup;
$f$ is the fraction of work performed by a faster component; and
$k$ is the speedup of the faster component.

$S = 1.15$

$f = 0.65$

$k = ?$

# AMDAHL'S LAW

$$k = \frac{f}{\left(\frac{1}{S} - (1-f)\right)} = \frac{0.65}{\left(\frac{1}{1.15} - (1-0.65)\right)} = 1.25$$

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where $S$ is the overall speedup; $f$ is the fraction of work performed by a faster component; and $k$ is the speedup of the faster component.

**$S$ = 1.15**

**$f$ = 0.65**

**$k$ = ?**

# AMDAHL'S LAW

$$k = \frac{f}{\left(\frac{1}{S} - (1-f)\right)} = \frac{0.65}{\left(\frac{1}{1.15} - (1-0.65)\right)} = 1.25$$

**Answer in percent: 25%**

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where $S$ is the overall speedup; $f$ is the fraction of work performed by a faster component; and $k$ is the speedup of the faster component.

**$S$ = 1.15**

**$f$ = 0.65**

**$k$ = ?**