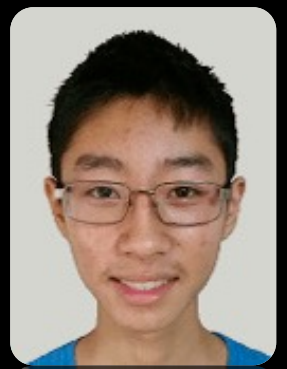
UC Berkeley
Teaching Professor
Lisa Yan

CS61C

Great Ideas in Computer Architecture

(a.k.a. Machine Structures)



UC Berkeley
Lecturer
Justin Yokota

Caches IV: Examples

Comparing Caches (Policies)

[review]

Fully Associative Cache

Direct Mapped Cache

- A specific line of data can be stored in **any line** of the cache.
- No index.
- Need to choose replacement policy.
- Need to choose write policy.
- Hardware: expensive

- A specific line of data can only be stored in **one index** of the cache.
- Has index.
- If the line you want to store the data in is occupied, you kick out that line.
- Need to choose write policy.
- Hardware: cheap

Agenda

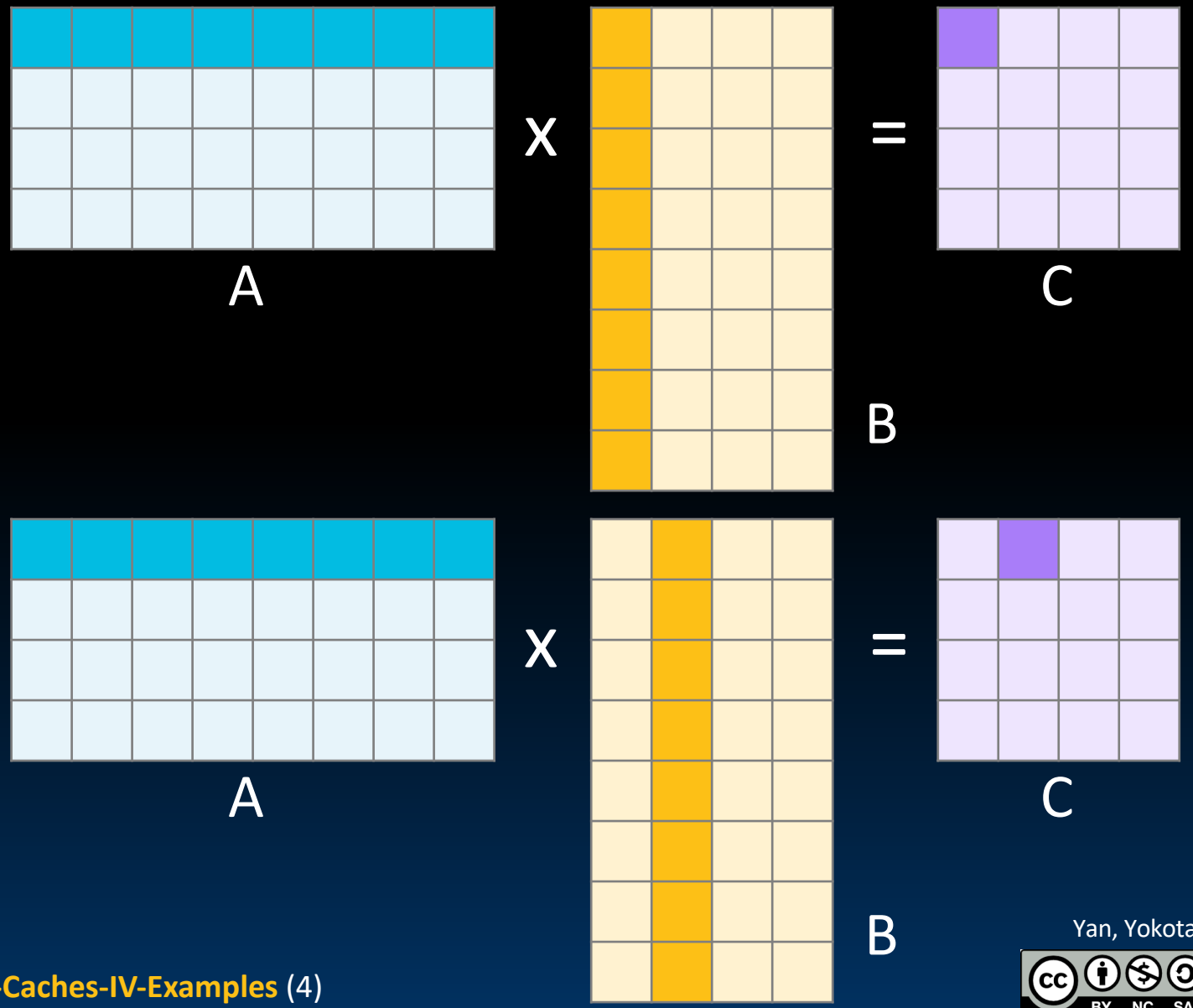
Matrix Multiply

- Matrix Multiply
- Set Associative Caches
- More on Misses

Recall Matrix Multiplication

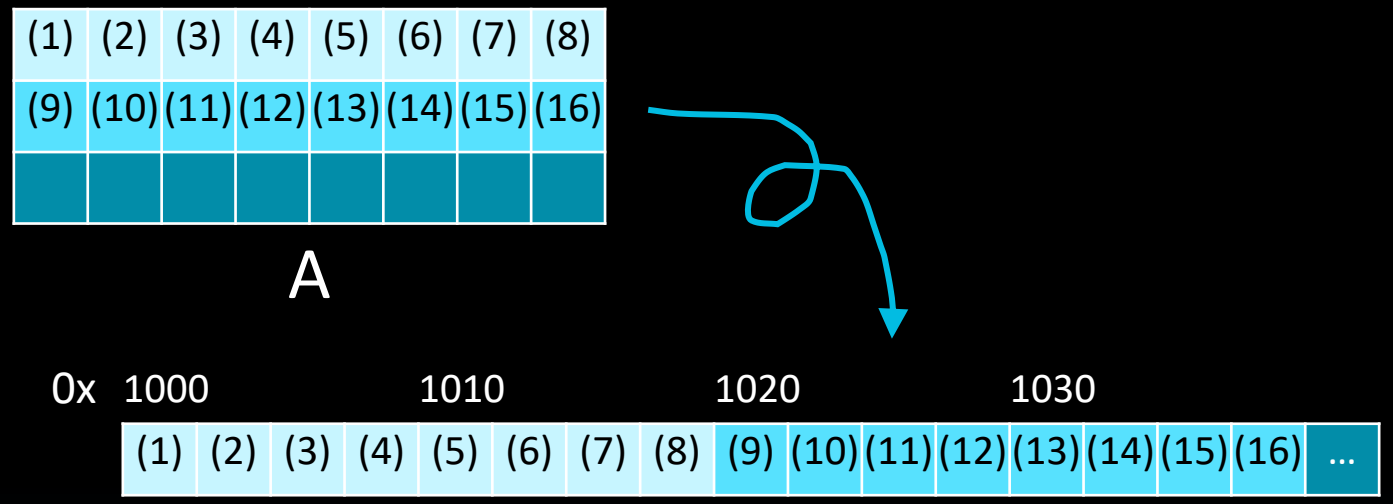
- For A x B = C, construct each element of C as follows:

```
// for row i, index j of C
int sum = 0; // sizeof(int) = 4
for (int k = 0; k < size; k++) {
  sum += A[i][k] * B[k][j];
}
C[i][j] = sum;
```



Matrix Multiplication Assumptions

- `sizeof(int) = 4`
- `int` 2-D matrices
- Matrices are stored in row-major order.

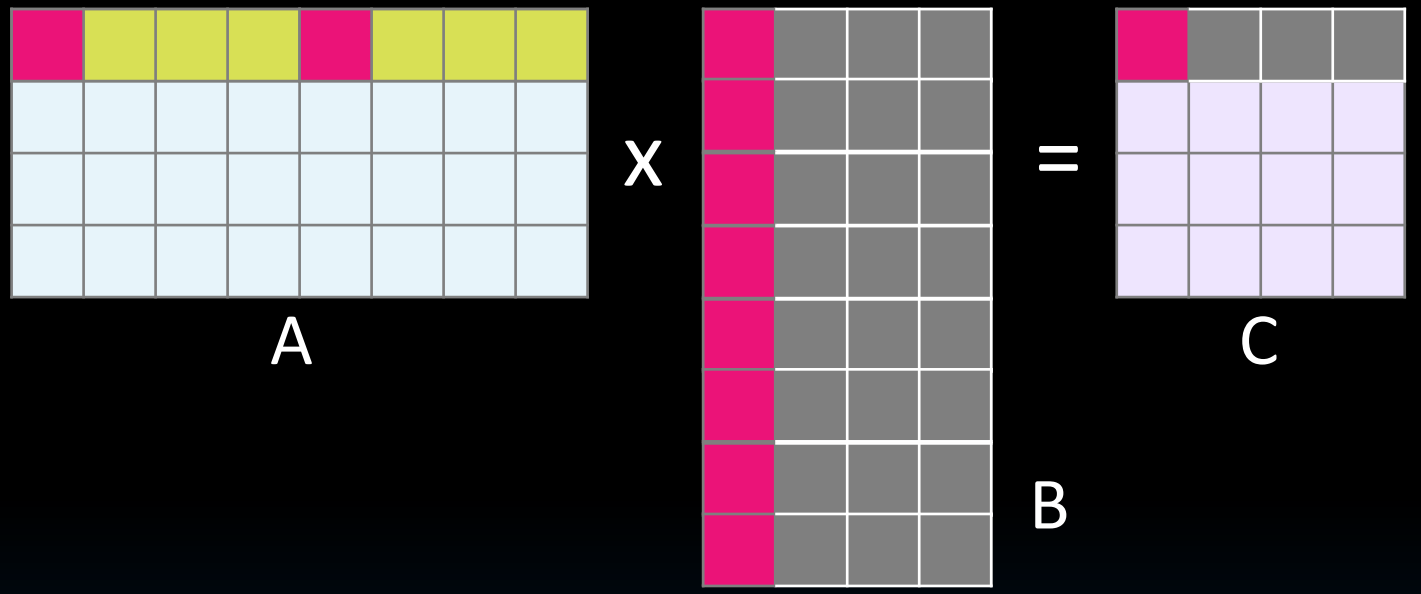


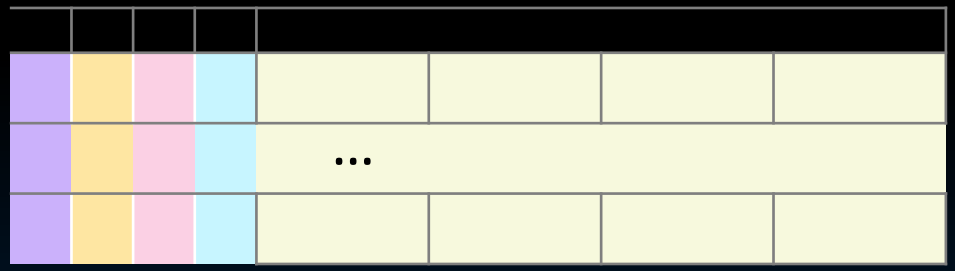
- 128B Fully Associative Cache:
  - 16B block size
  - 8 blocks

Valid	Dirty	LRU	Tag	Data F			Data 0
1	…	…	0x100	(4)	(3)	(2)	(1)
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…

C[i][j] memory access pattern

```
// for row i, index j of C
int sum = 0; // sizeof(int) = 4
for (int k = 0; k < size; k++) {
  sum += A[i][k] * B[k][j];
}
C[i][j] = sum;
```





Fully Associative Cache
16B block size, 4 blocks, LRU

Adjusted after lecture (from 8 blocks) such that B, C blocks are clearly evicted before second access

Miss

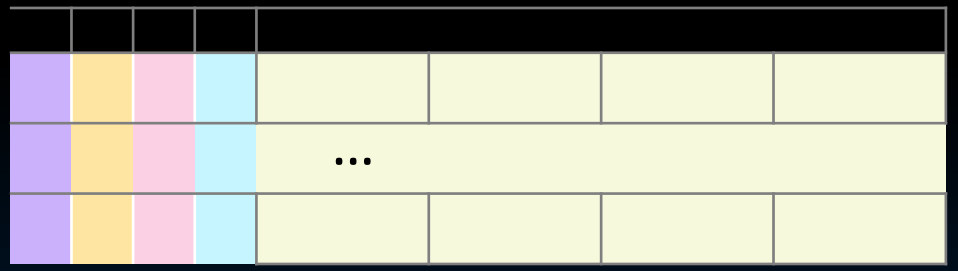
Hit

Brought in, but unused before eviction

Yan, Yokota

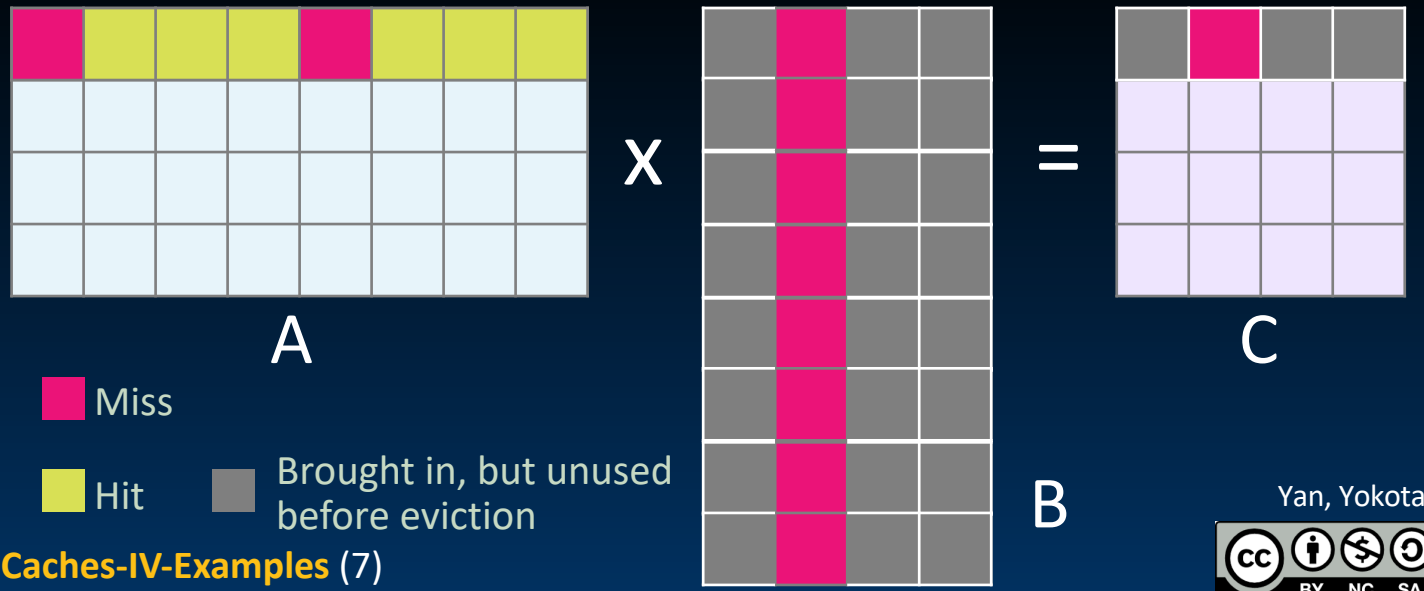
C[i][j] memory access pattern

```
// for row i, index j of C
int sum = 0; // sizeof(int) = 4
for (int k = 0; k < size; k++) {
  sum += A[i][k] * B[k][j];
}
C[i][j] = sum;
```



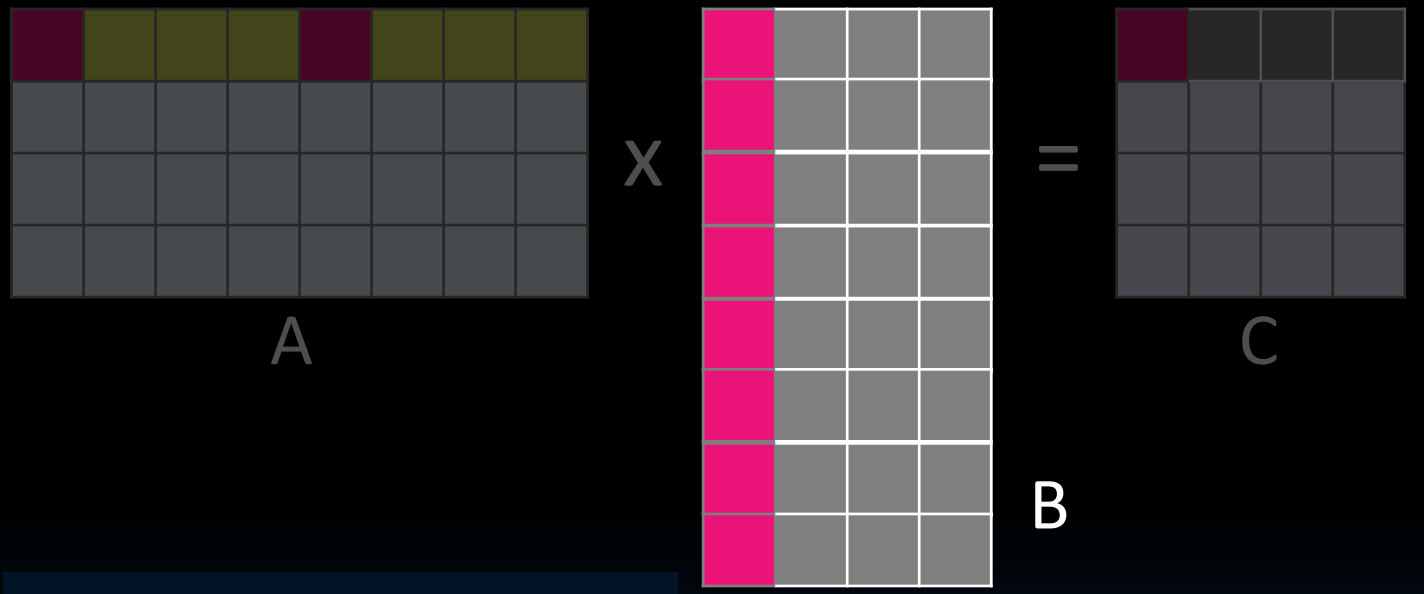
Fully Associative Cache
16B block size, 4 blocks, LRU

Adjusted after lecture (from 8 blocks) such that B, C blocks are clearly evicted before second access



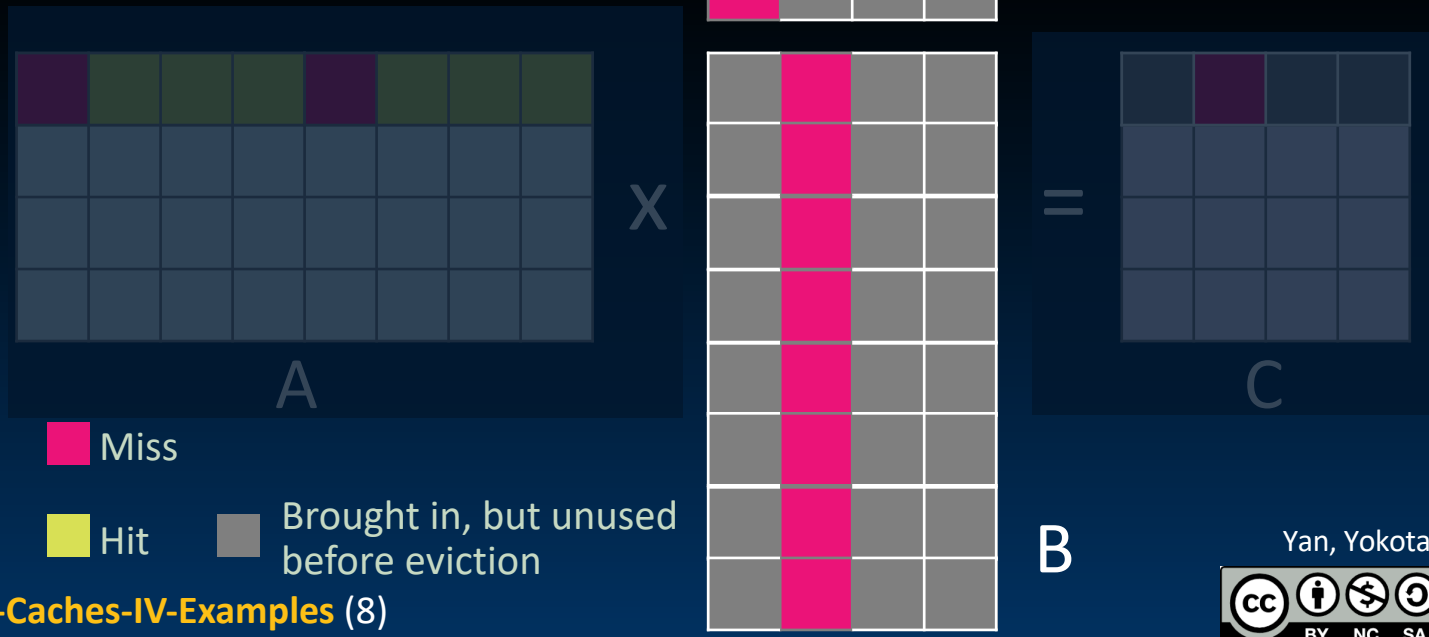
C[i][j] memory access pattern

```
// for row i, index j of C
int sum = 0; // sizeof(int) = 4
for (int k = 0; k < size; k++) {
  sum += A[i][k] * B[k][j];
}
C[i][j] = sum;
```



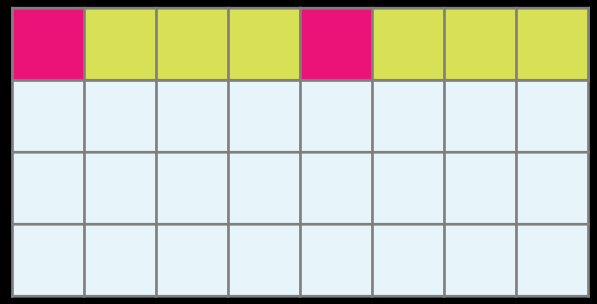
Fully Associative Cache
16B block size, 4 blocks, LRU

B's row-major layout in memory causes excessive memory accesses!

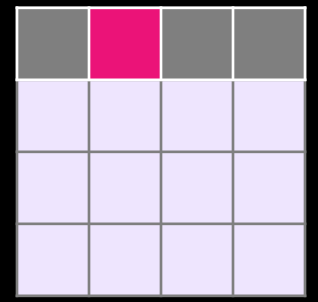


Cache Blocking: Make use of the cache!

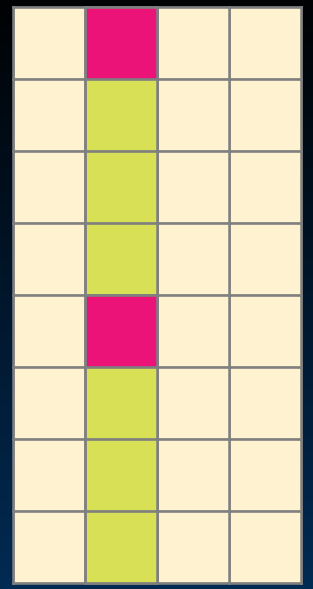
- "Transpose" matrix B before performing matrix multiplication.
  - Still mathematically multiplying A x B!
  - However, B is effectively "column-major" order, reducing excessive evictions.



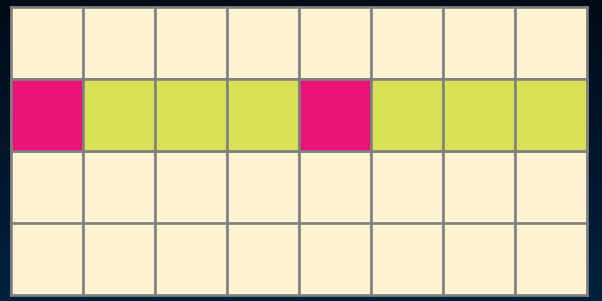
A



C



B



B transpose

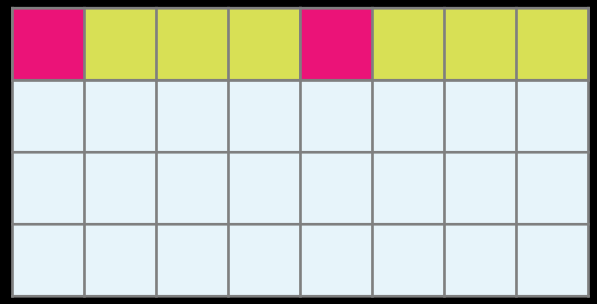
Use on Project 4!

More: Sp22 Lec17.38-50 [link]

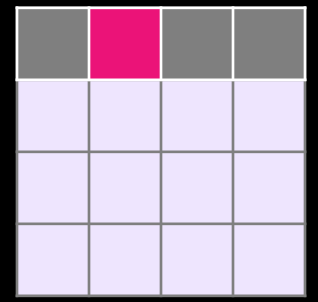
Yan, Yokota

Cache Blocking: Make use of the cache!

- "Transpose" matrix B before performing matrix multiplication.
  - Still mathematically multiplying A x B!
  - However, B is effectively "column-major" order, reducing excessive evictions.

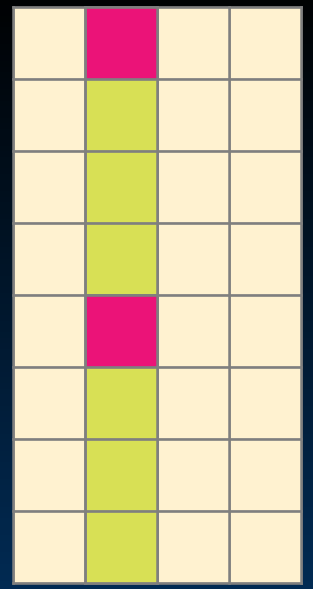


A

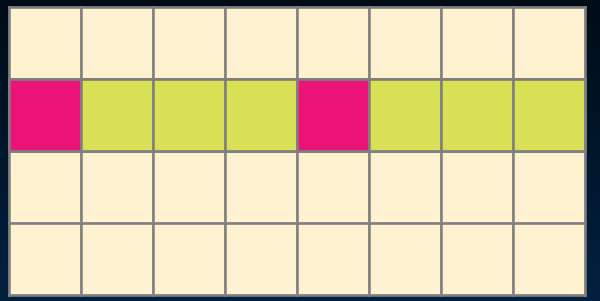


C

- **Cache blocking**:
  - Programmer technique: **Rearrange data accesses** to make better use of data brought into the cache.
  - Prevent repeatedly evicting and fetching the same data from the main memory!



B



B transpose

Use on Project 4!

More: Sp22 Lec17.38-50 [link]

Yan, Yokota

Agenda

Set-Associative Caches

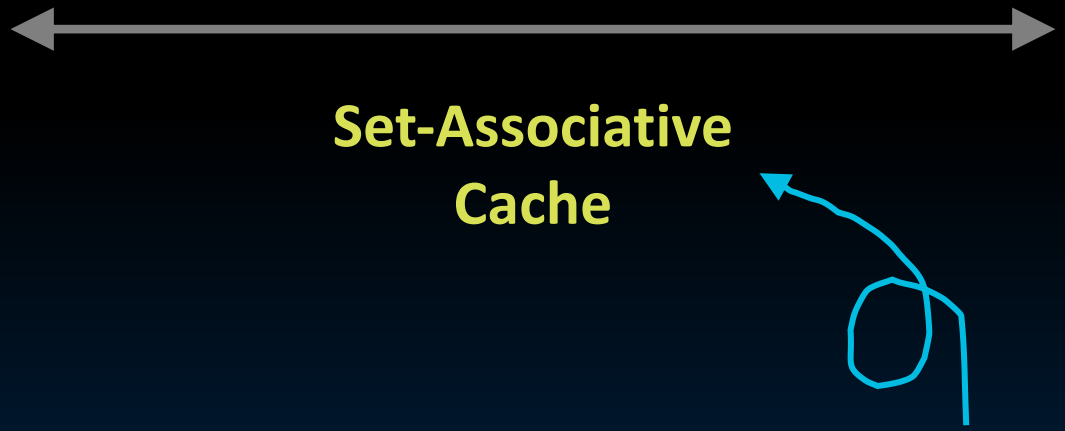
- Matrix Multiply
- Set Associative Caches
- More on Misses

Cache Design: Placement Policies

[in scope]

Fully Associative Cache

Put a new block anywhere



Direct Mapped Cache

Put a new block in one specific place

In scope: Why it exists, why it is often preferred vs. FA/DM
Out of scope: Details

2-Way Set-Associative Caches, in Detail [out of scope]

- 16B capacity, 4B blocks, write-back
- **2-Way Set-Associative**
  - Each **index** is now associated with **a set of 2 blocks**.
  - Block replacement policy (e.g., LRU) occurs within each **Set**.

	Valid	Dirty	LRU	Tag	Data 11	Data 10	Data 01	Data 00
0	1	0	0	0x11C	…	…	…	…
	0	…	…	…	…	…	…	…
1	0	…	…	…	…	…	…	…
	0	…	…	…	…	…	…	…

Note: In this toy example, there happens to be 2 sets as well.

In a 2-way Set Associative cache with 8 blocks total, then there would be 4 sets.

For more, see Spring 2022 slides:
https://inst.eecs.berkeley.edu/~cs61c/sp22/pdfs/lectures/lec16.pdf

A specific block of data can be stored at only one index of the cache, but multiple blocks can be in each index.

2-Way Set-Associative Caches, in Detail [out of scope]

- 16B capacity, 4B blocks, write-back
- Suppose the cache starts cold.

1. Load byte 0x8E2 0x11C, 0, 0x2
2. Load byte 0x8E8 0x11D, 0, 0x0
3. Load byte 0x8E9 0x11D, 0, 0x1
4. Load byte 0xDF7 0x1BE, 1, 0x3
5. Load byte 0xAB8 0x157, 0, 0x0

	Valid	Dirty	LRU	Tag	Data 11	Data 10	Data 01	Data 00
0	1	0	1	0x11C	…	…	…	…
	1	0	0	0x11D	…	…	…	…
1	1	0	0	0x1BE	…	…	…	…
	0	…	…	…	…	…	…	…

A specific block of data can be stored at only one index of the cache, but multiple blocks can be in each index.

For more, see Spring 2022 slides:
https://inst.eecs.berkeley.edu/~cs61c/sp22/pdfs/lectures/lec16.pdf

2-Way Set-Associative Caches, in Detail [out of scope]

- 16B capacity, 4B blocks, write-back
- Suppose the cache starts cold.

1. Load byte 0x8E2 0x11C, 0, 0x2
2. Load byte 0x8E8 0x11D, 0, 0x0
3. Load byte 0x8E9 0x11D, 0, 0x1
4. Load byte 0xDF7 0x1BE, 1, 0x3
5. Load byte 0xAB8 0x157, 0, 0x0

	Valid	Dirty	LRU	Tag	Data 11	Data 10	Data 01	Data 00
0	1	0	0	0x157	…	…	…	…
	1	0	1	0x11D	…	…	…	…
1	1	0	0	0x1BE	…	…	…	…
	0	…	…	…	…	…	…	…

A specific block of data can be stored at only one index of the cache, but multiple blocks can be in each index.

For more, see Spring 2022 slides:
https://inst.eecs.berkeley.edu/~cs61c/sp22/pdfs/lectures/lec16.pdf

Comparing Caches (Policies)

[in scope]

Fully Associative Cache

"M-way" associative (where M = # blocks)

- A specific line of data can be stored in **any line** of the cache.
- No index.
- Need to choose replacement policy.
- Need to choose write policy.
- Hardware: expensive

Set-Associative Cache

- A specific line of data can be stored at only one index of the cache, but **multiple lines** can be in each **index**.
- Has index.
- Need to choose replacement policy.
- Need to choose write policy.

Direct Mapped Cache

1-way associative

- A specific line of data can only be stored in **one index** of the cache.
- Has index.
- If the line you want to store the data in is occupied, you kick out that line.
- Need to choose write policy.
- Hardware: cheap

Comparing Caches (Performance/HW)

[in scope]

Fully Associative Cache

Direct Mapped Cache

Set-Associative Cache

- Reduces excessive conflict misses
- Increases usage of the cache

- Reduced circuitry required
- Possibly larger cache sizes? (since less complex, cheaper HW)

Mostly balances good parts of FA (i.e., better performance by reducing conflict misses) with good parts of DM (simpler hardware)

Agenda

More on Misses

- Matrix Multiply
- Set Associative Caches
- More on Misses

Types of Misses

[review]

- Compulsory Miss
  - Caused by the first access to a block that has never been in the cache
- Capacity Miss
  - Caused when the cache cannot contain all the blocks needed during the execution of a program
  - Occur when blocks were in the cache, replaced, and later retrieved
- Conflict Miss
  - Multiple blocks compete for the same location in the cache, even when the cache has not reached full capacity.
  - Occurs in direct mapped caches, as well as in set associative caches.
  - Misses of this type would **not occur in a fully associative cache** with similar specs.

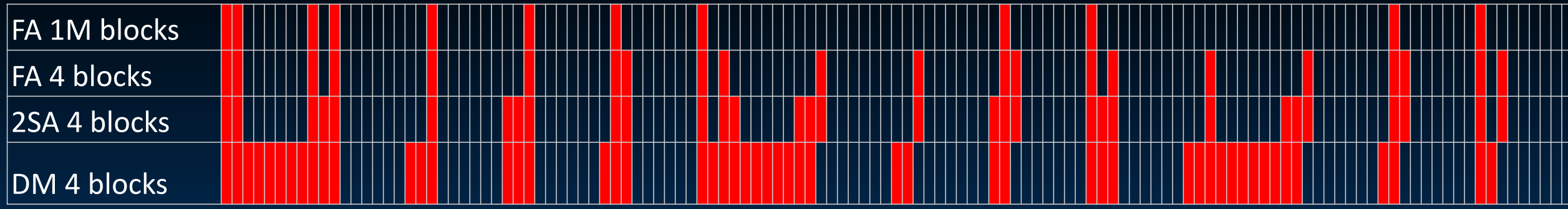
What do we mean by this?

Yan, Yokota

Analyzing Cache Effectiveness

[post-lecture: updated diagram to be consistent with Fa23]

- Let's compare the hit patterns of a few cache types.
- The below diagram shows the hit/miss pattern of various caches when run on a Matrix Multiply Example.
  - FA 1M (Fully Associative, 1 million—"ideal" with as many hits as possible
  - FA 4 blocks (Fully Associative), 2SA 4 blocks (Set Associative), DM 4 blocks (Direct Mapped)



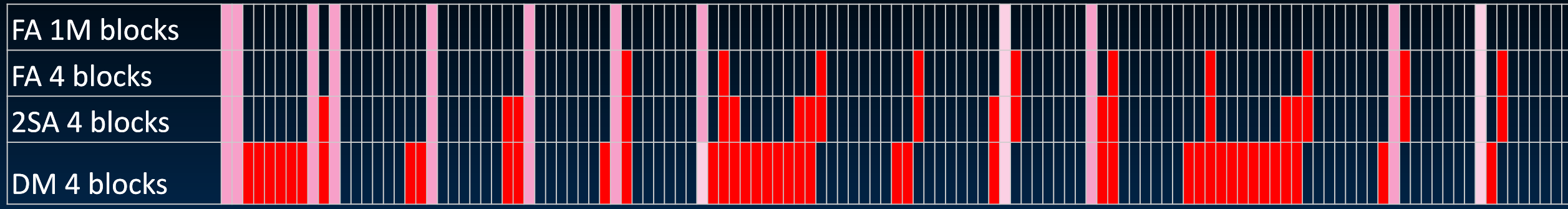
Miss

Hit

Compulsory Miss

[post-lecture: updated diagram to be consistent with Fa23]

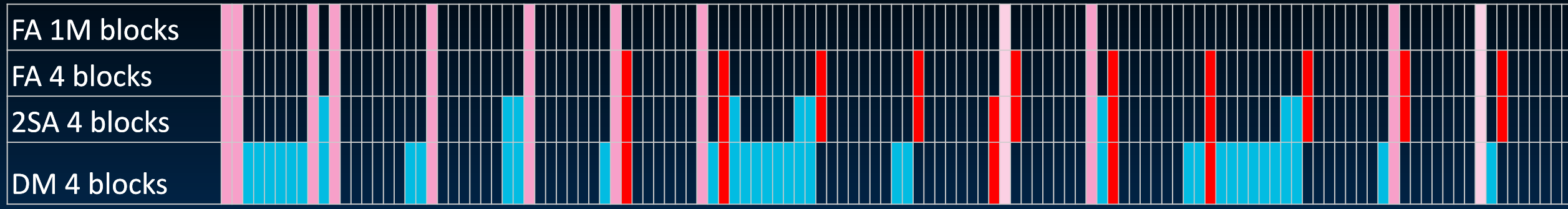
- **Compulsory Miss**
  - Caused by the first access to a block that has never been in the cache
  - (If our program mostly causes compulsory misses, then we can't improve much)



Compulsory Miss

[post-lecture: updated diagram to be consistent with Fa23]

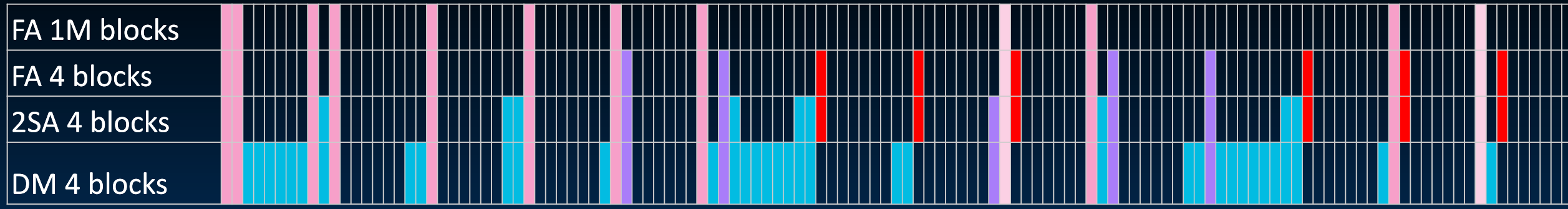
- **Compulsory Miss**
  - Caused by the first access to a block that has never been in the cache
  - (If our program mostly causes compulsory misses, then we can't improve much)
- **Conflict Miss**
  - Multiple blocks compete for the same location in the cache, even when the cache has not reached full capacity.
  - "misses that would not occur if the cache was **fully-associative** and had LRU replacement" (with all other noncompulsory misses being **capacity**) [link]



Capacity Miss

[post-lecture: updated diagram to be consistent with Fa23]

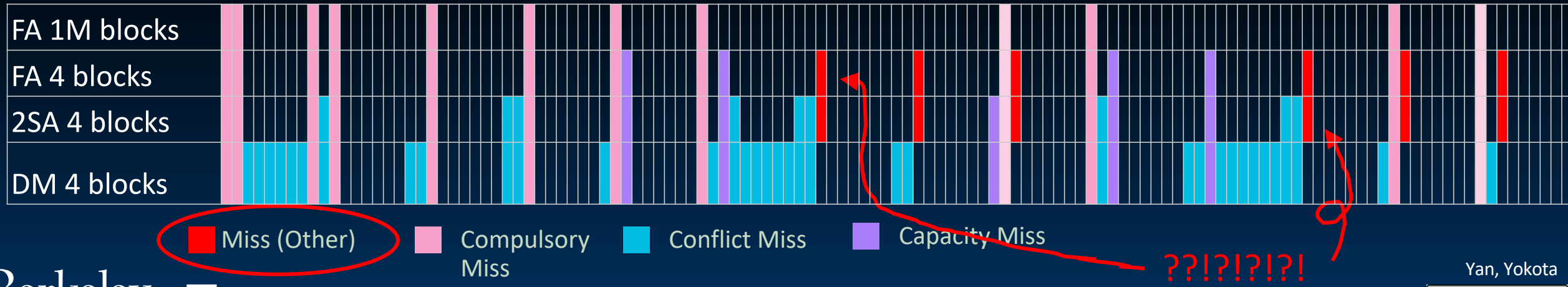
- **Compulsory Miss**
  - Caused by the first access to a block that has never been in the cache
  - (If our program mostly causes compulsory misses, then we can't improve much)
- **Conflict Miss**
  - Multiple blocks compete for the same location in the cache, even when the cache has not reached full capacity.
  - "misses that would not occur if the cache was **fully-associative** and had LRU replacement" (with all other noncompulsory misses being **capacity**) [link]
- **Capacity Miss**



Hold on…

[post-lecture: updated diagram to be consistent with Fa23]

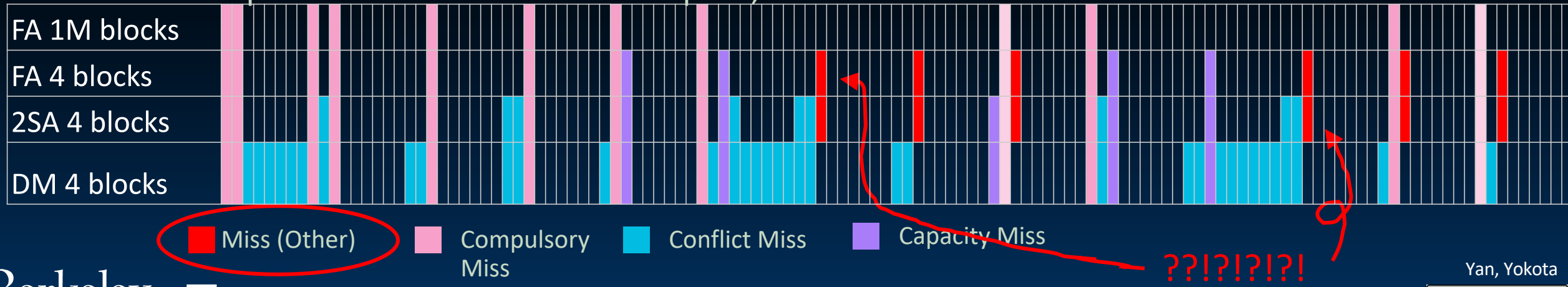
- **Compulsory Miss**
  - Caused by the first access to a block that has never been in the cache
  - (If our program mostly causes compulsory misses, then we can't improve much)
- **Conflict Miss**
  - Multiple blocks compete for the same location in the cache, even when the cache has not reached full capacity.
  - "misses that would not occur if the cache was **fully-associative** and had LRU replacement" (with all other noncompulsory misses being **capacity**) [link]
- **Capacity Miss**



Hold on…

[post-lecture: updated diagram to be consistent with Fa23]

- **Conflict Miss**
  - Multiple blocks compete for the same location in the cache, even when the cache has not reached full capacity.
  - "misses that would not occur if the cache was **fully-associative** and had LRU replacement" (with all other noncompulsory misses being **capacity**) [link]
- Problem: This definition assumes all misses in equivalent FA cache **also** cause misses in lower associativities.
  - But below, sometimes we get "conflict hits" because a block that would have been evicted ends up staying in the cache longer! Depends on replacement policy.
  - Also requires a lot more work to compute, so we'll use an easier definition in this class



Types of Misses, Simplified (for this class)

- **Compulsory Miss**
  - Caused by the first access to a block that has never been in the cache
- **Capacity Miss**
  - Caused when the cache cannot contain all the blocks needed during the execution of a program
  - Occur when blocks were in the cache, replaced, and later retrieved
- **Conflict Miss**
  - Multiple blocks compete for the same location in the cache, even when the cache has not reached full capacity.
  - Occurs in direct mapped caches, as well as in set associative caches.
  - Misses of this type would **not occur in a fully associative cache** with similar specs.

- If a miss occurs:
  - Block never evicted before? **Compulsory**.
  - Cache currently full (no empty spaces)? **Capacity**.
  - Otherwise? **Conflict**.

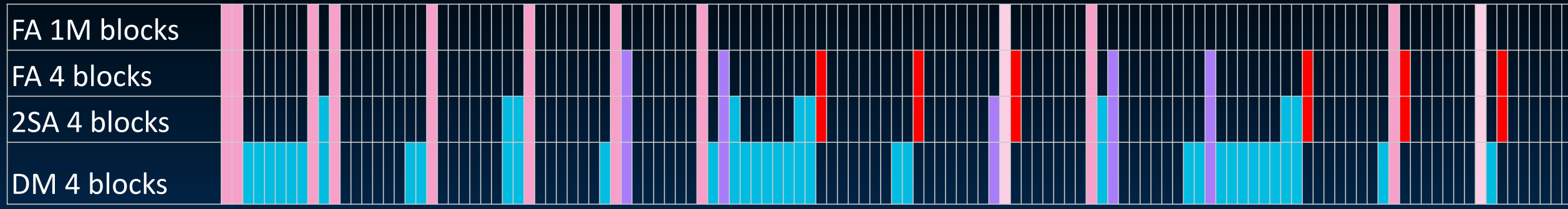
Types of Misses, Simplified (for this class)

More details here:

- https://electronics.stackexchange.com/questions/421286/cache-miss-types-capacity-miss-vs-conflict-miss
- https://www.sciencedirect.com/topics/computer-science/capacity-miss
- The original Technical Report: https://bitsavers.org/pdf/dec/tech_reports/WRL-TN-53.pdf

- If a miss occurs:
  - Block never evicted before? **Compulsory**.
  - Cache currently full (no empty spaces)? **Capacity**.
  - Otherwise? **Conflict**.

[post-lecture: updated diagram to be consistent with Fa23]



Types of Misses, In Detail (for this class)

[out of scope]

- Compulsory Miss
  - Would never be a hit regardless of cache design.
- Capacity Miss
  - Would be a hit if we increased capacity.

- Conflict Miss
  - In the literature: Would be a hit if we increased associativity.
  - Misses that would not have happened if the cache was fully associative under at least one “consistent eviction policy”,
    - “Consistent eviction policy”: An eviction policy that keeps all the data in the lower-associativity cache
    - In other words, any eviction policy such that the data in the Fully Associative cache is a superset of the data in our actual cache.
    - This definition guarantees that we don't have "conflict hits", but also tends to classify misses more as capacity misses than as conflict misses

Agenda

Extra Practice

- Matrix Multiply
- Set Associative Caches
- More on Misses

Direct Mapped Cache (write-back)

- Suppose the cache starts **cold**.

1. Load byte 0xFE2 0b1111 1110 0010 0xFE, 0x0, 0x2
2. Load byte 0xFE8 0b1111 1110 1000 0xFE, 0x2, 0x0
3. Load word 0xFE9 0b1111 1110 1001 0xFE, 0x2, 0x1
4. Load word 0xDF9 0b1101 1111 1001 0xDF, 0x2, 0x1
5. Load byte 0xFE8 0b1111 1110 1000 0xFE, 0x2, 0x0

1. What is the resulting state of the cache?
2. What misses occur, and are they (1) compulsory, (2) capacity, or (3) conflict?

Valid	Dirty	Tag	Data			
			11	10	01	00
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…

Direct Mapped Cache (write-back)

- Suppose the cache starts **cold**.

1. Load byte 0xFE2 0b1111 1110 0010 0xFE, 0x0, 0x2
2. Load byte 0xFE8 0b1111 1110 1000 0xFE, 0x2, 0x0
3. Load word 0xFE9 0b1111 1110 1001 0xFE, 0x2, 0x1
4. Load word 0xDF9 0b1101 1111 1001 0xDF, 0x2, 0x1
5. Load byte 0xFE8 0b1111 1110 1000 0xFE, 0x2, 0x0