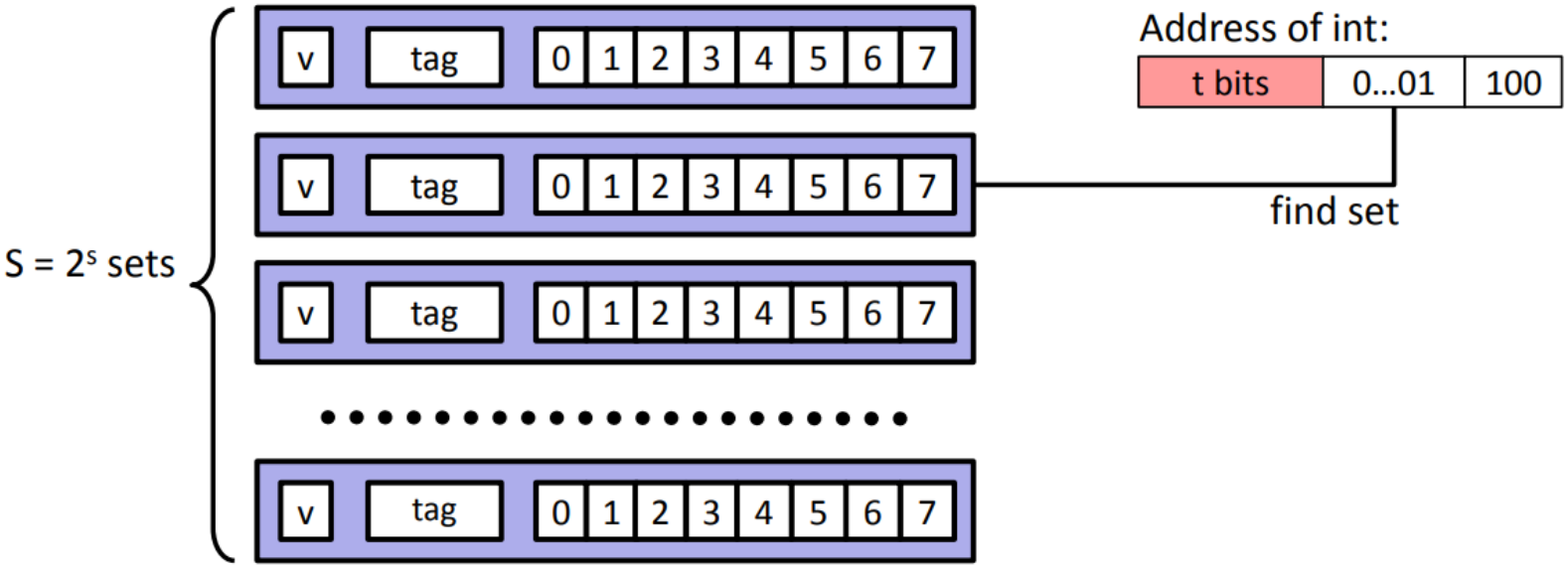
Lab 08
Cache (Part B)
- Optimization -

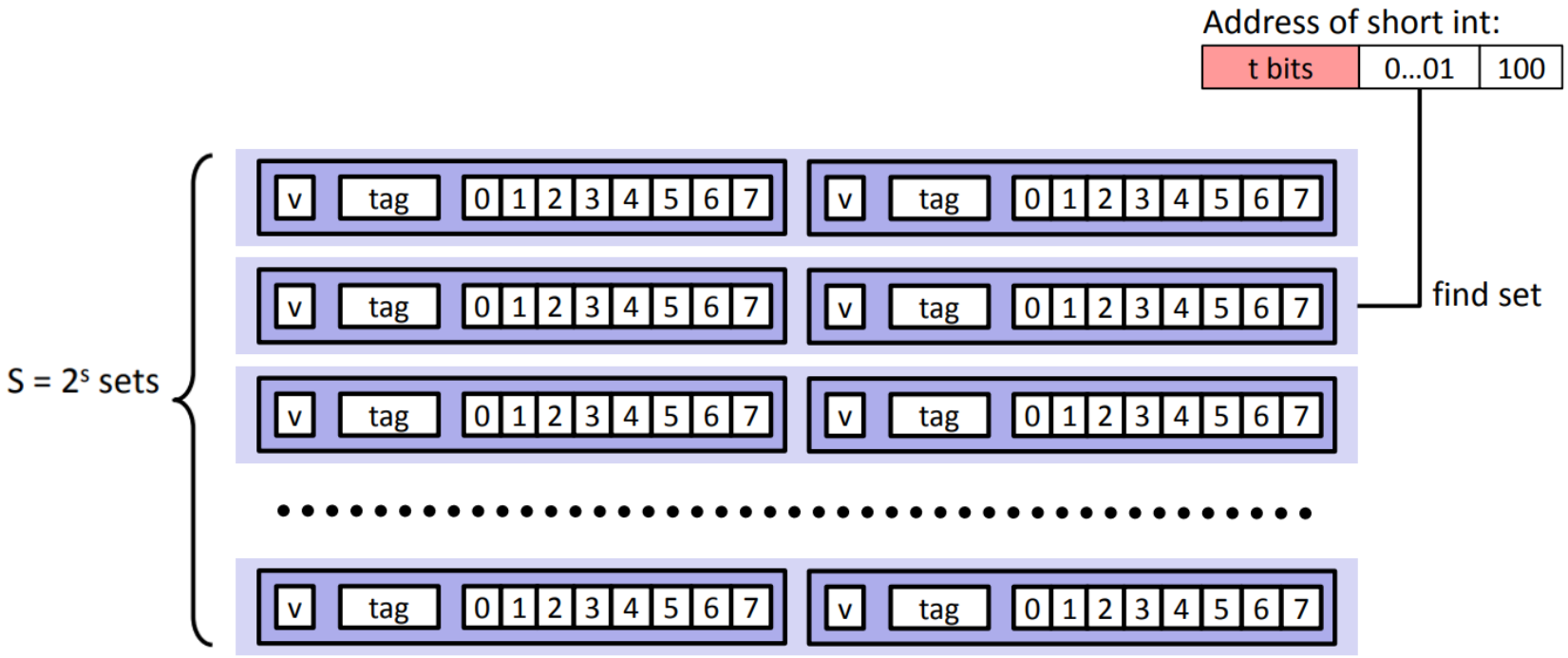
Review Previous Session

- Direct Mapped Cache
  - One line per set
  - Assume: cache block size 8 bytes



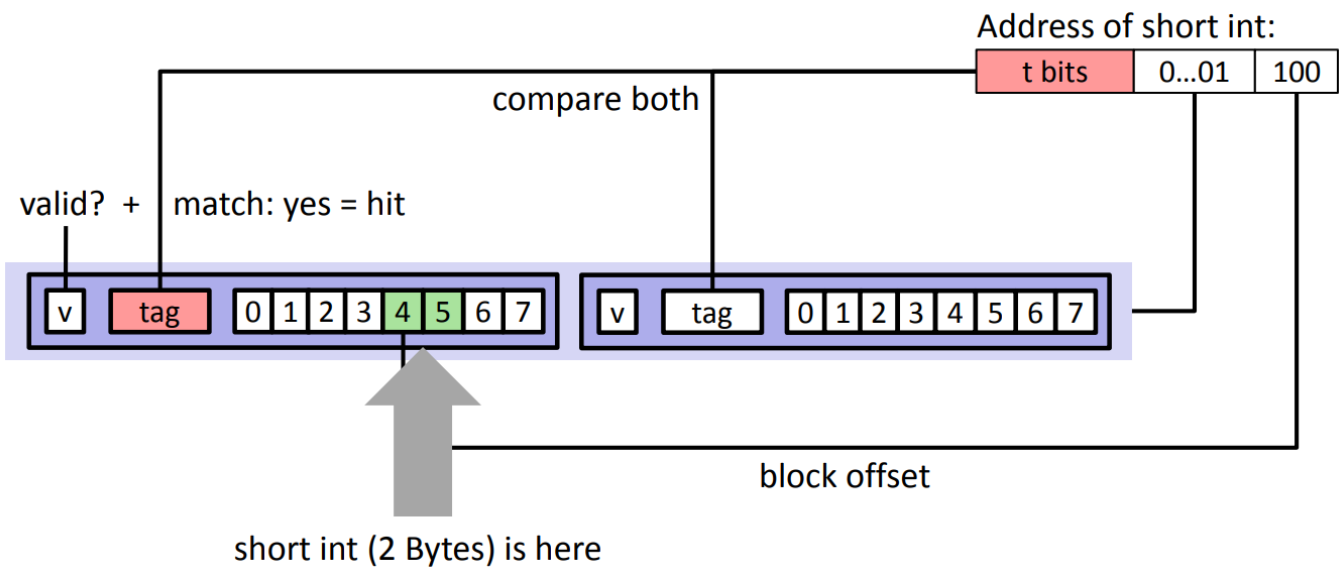
Review Previous Session

- 2-Way Set Associative Cache
  - Two lines per set
  - Assume: cache block size 8 bytes



Review Previous Session

- When the cache is full,
  - Direct Mapped Cache
    - Old line is evicted and replaced
  - E-way Set Associative Cache
    - One line in set is selected for eviction and replacement
    - *Replace policies*: random, least recently used (LRU), …



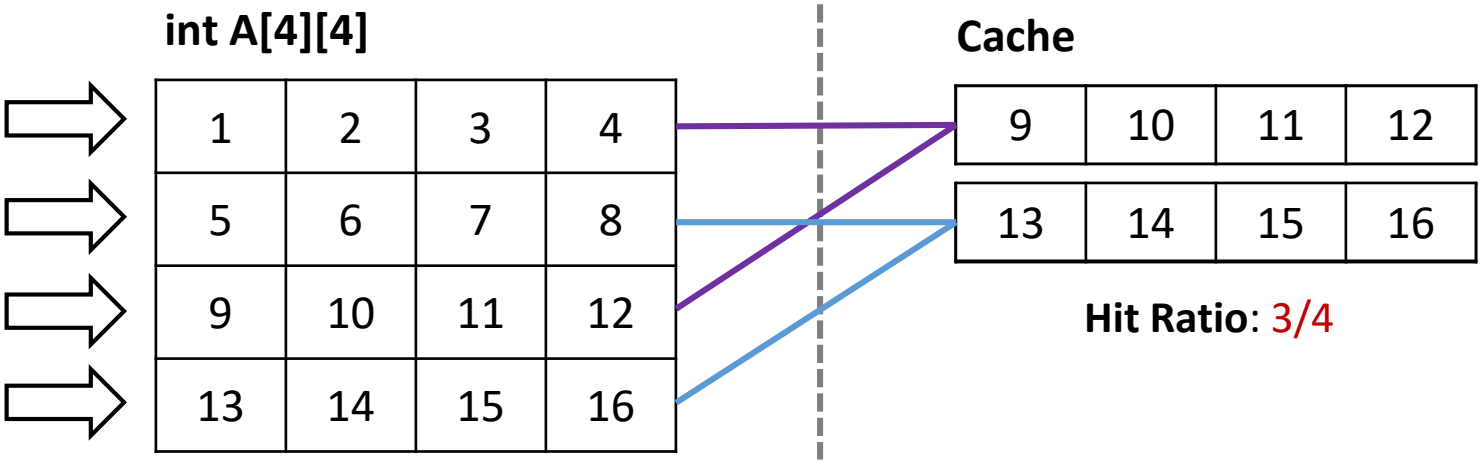
Cache Replacement Policy

- Least Recently Used (LRU)
  - Replace the cache block which was used least recently

Memory	0	1	2	3	0	1	4	0	1	2	3	4
Cache Status	0	0	0	3	3	3	4	4	4	2	2	2
		1	1	1	0	0	0	0	0	0	3	3
			2	2	2	1	1	1	1	1	1	4
	M	M	M	M	M	M	M	H	H	M	M	M

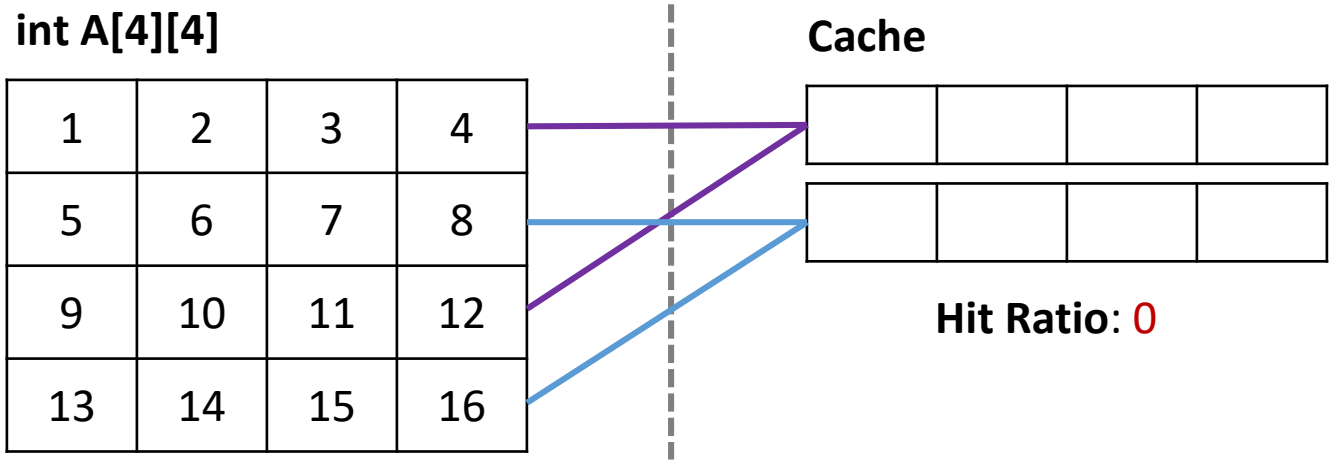
Hit Ratio

- The percentage of accesses that result in cache hits
- Example
  - 32 bytes direct mapped cache with a block size of 16 bytes
  - *Row-major order*



Hit Ratio

- The percentage of accesses that result in cache hits
- Example
  - 32 bytes direct mapped cache with a block size of 16 bytes
  - How about *Column-major* order?



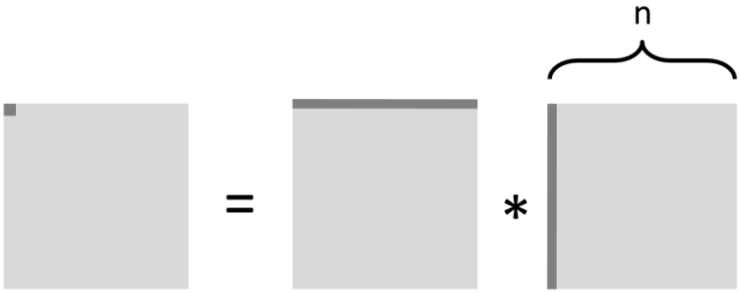
Example: Matrix Multiplication

- Assume
  - Cache block = 8 doubles

```
c = (double *) calloc(sizeof(double), n * n);

/* Multiply n x n matrices a and b */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            for (k = 0; k < n; k++)
                c[i * n + j] += a[i * n + k] * b[k * n + j];
        }
    }
}
```

- Inner iteration
  - $n/8 + n = 9n/8$ misses
- Total misses
  - $9n/8 * n^2 = (9/8) * n^3$



Example: Blocked Matrix Multiplication

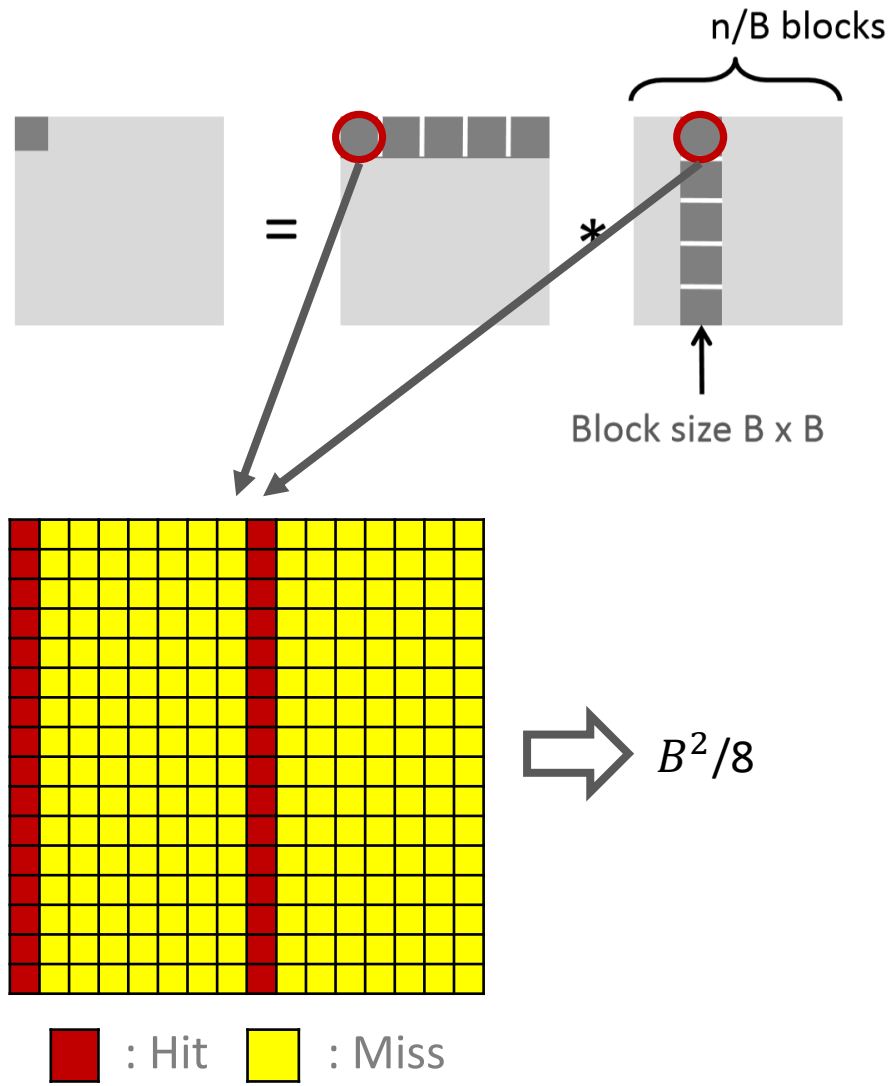
- Assume
  - Cache block = 8 doubles
  - Three blocks ■ fit into cache: $3B^2 < C$

```
c = (double *) calloc(sizeof(double), n * n);

/* Multiply n x n matrices a and b */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k, i1, j1, k1;
    for (i = 0; i < n; i += B)
        for (j = 0; j < n; j += B)
            for (k = 0; k < n; k += B)
                /* B x B mini matrix multiplication */
                for (i1 = i; i1 < i + B; i1++)
                    for (j1 = j; j1 < j + B; j1++)
                        for (k1 = k; k1 < k + B; k1++)
                            c[i1 * n + j1] += a[i1 * n + k1] * b[k1 * n + j1];
}
```

Example: Blocked Matrix Multiplication

- Block Iteration
  - $B^2/8$ misses for each block
  - 2n/B * $B^2/8$ = nB/4
- Total misses
  - nB/4 * $(n/B)^2$ = $\boldsymbol{n^3/(4B)}$



Cachegrind

- Simulating how your program interacts with a machine's cache hierarchy and branch predictor
- For modern machines that have three or four levels of cache, Cachegrind simulates the *first-level* and *last-level* caches
  - Instruction Cache: I1, LLi
  - Data Cache: D1, LLd

```
==19042== I   refs:      821,074,275
==19042== I1  misses:            852
==19042== LLi misses:            841
==19042== I1  miss rate:        0.00%
==19042== LLi miss rate:        0.00%
==19042==
==19042== D   refs:      345,648,721  (327,742,988 rd   + 17,905,733 wr)
==19042== D1  misses:        137,974  (    137,427 rd   +        547 wr)
==19042== LLd misses:         13,928  (     13,410 rd   +        518 wr)
==19042== D1  miss rate:         0.0% (        0.0%     +        0.0%  )
==19042== LLd miss rate:         0.0% (        0.0%     +        0.0%  )
==19042==
==19042== LL refs:           138,826  (    138,279 rd   +        547 wr)
==19042== LL misses:          14,769  (     14,251 rd   +        518 wr)
==19042== LL miss rate:          0.0% (        0.0%     +        0.0%  )
```

Cachelab Part B

- Optimize matrix transpose $(A \rightarrow A^T)$
  - Write the efficient code with the highest hit ratio (i.e. minimize the cache miss)
- Reference README file
  - Notice '**Blocking**' technique
  - You would be better to think about diagonal entries