

CSED211: Lab. 9

Cache Lab

(Part B: Efficient Matrix Transpose)

백승훈

habaek4@postech.ac.kr

POSTECH

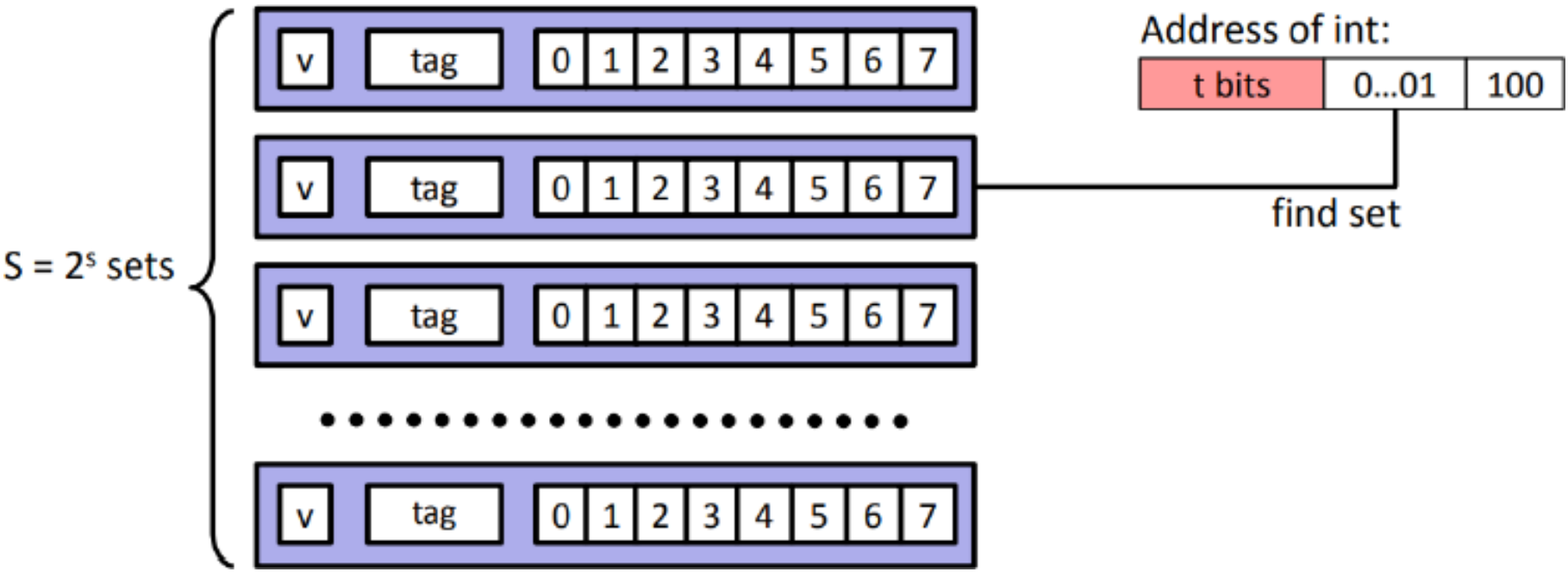
2023.11.20

Table of Contents

- Review Previous Session
- Cache Lab
 - Part A. Building Cache Simulator
 - Part B. Efficient Matrix Transpose

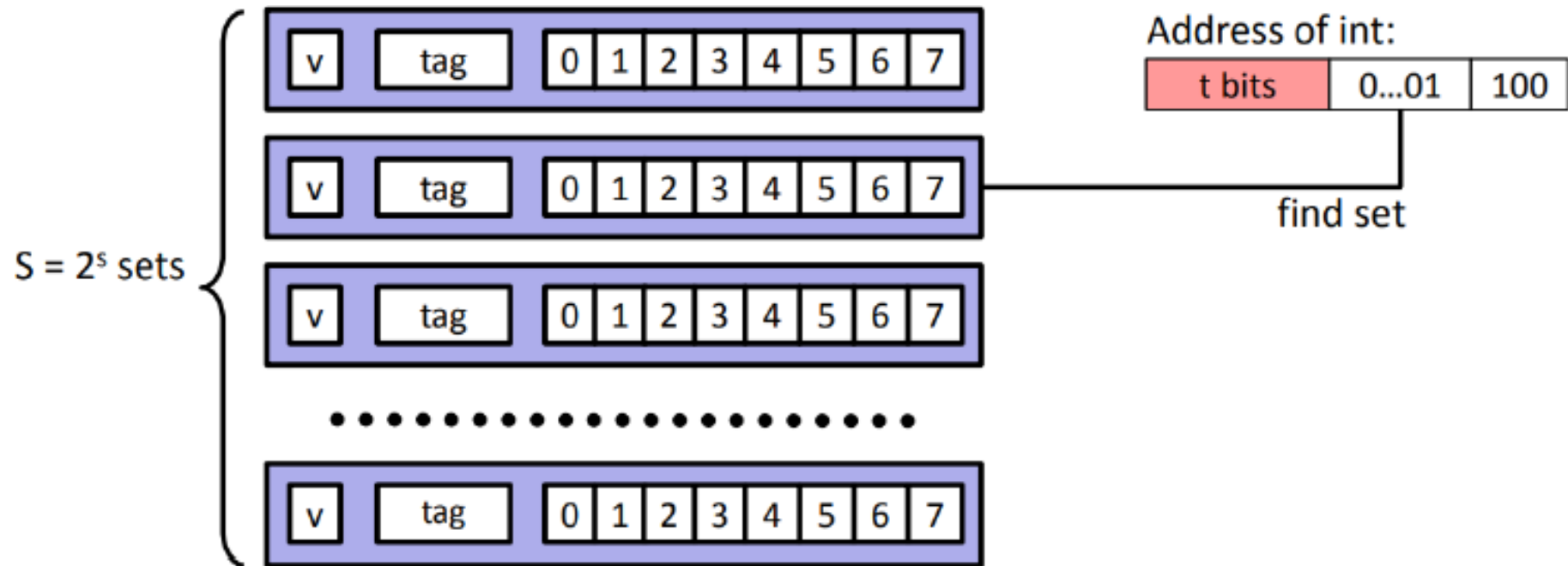
Review Previous Session

- Directed Mapped Cache
 - One line per set



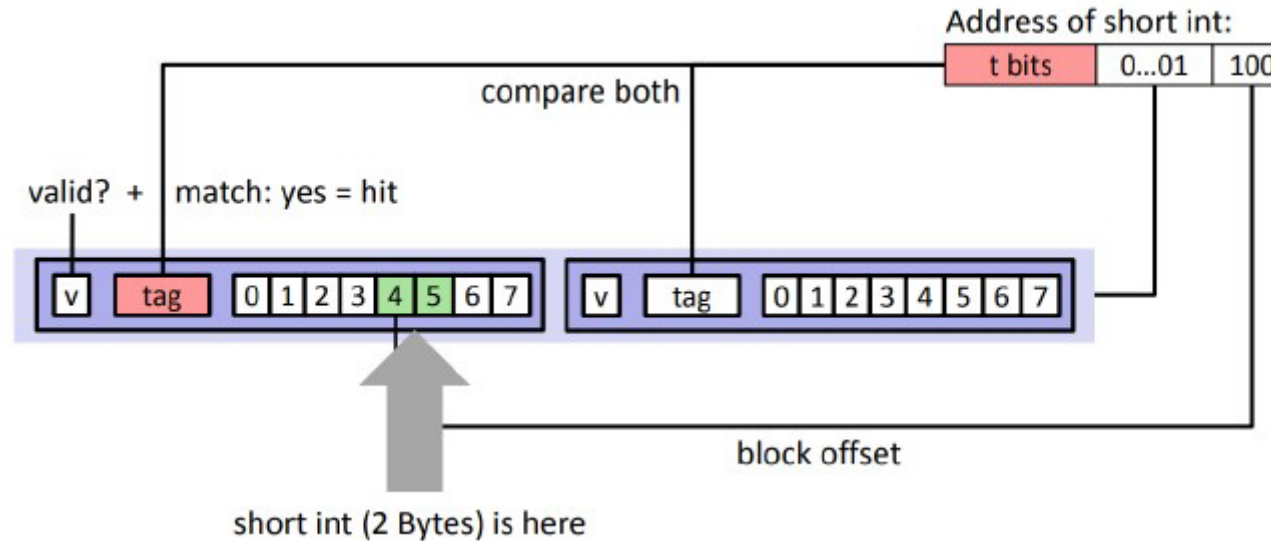
Review Previous Session

- 2-way Set Associative Cache
 - Two lines per set



Review Previous Session

- When the cache is full,
 - Directed mapped Cache
 - Old line is evicted and replaced
 - E-way Set Associative Cache
 - One line in set is selected for eviction and replaced
 - Replacement Policy: Least Recently Used (LRU)



Review Previous Session

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

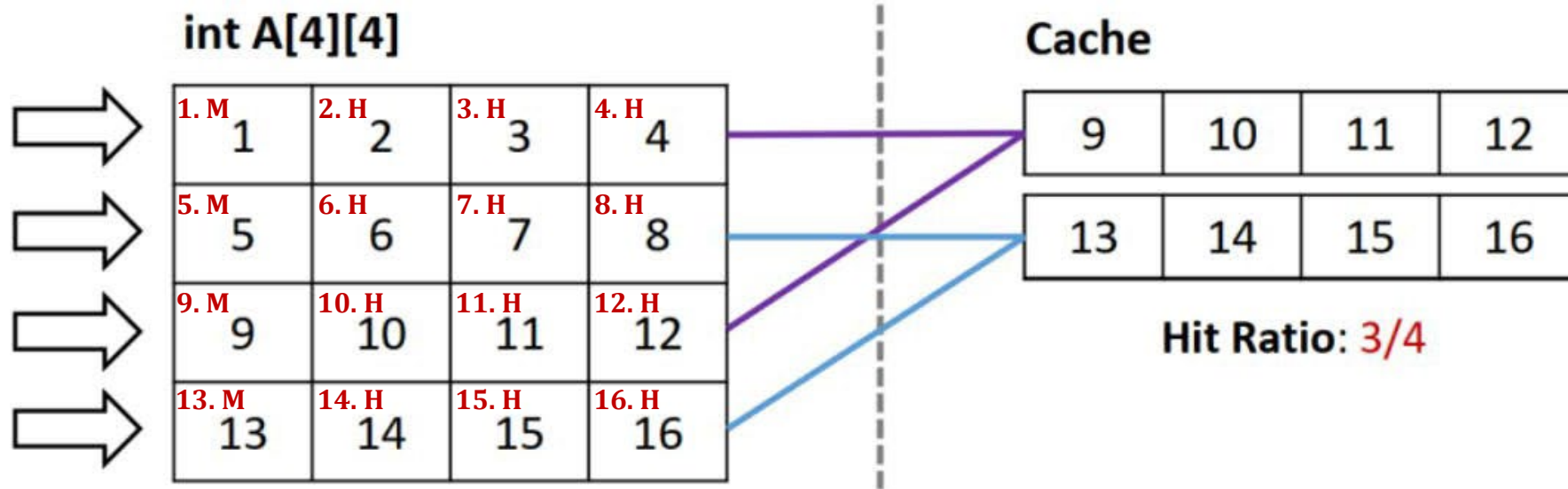
Memory	0	1	2	3	0	1	4	0	1	2	3	4
Cache Status	0	0	0	3	3	3	4	4	4	2	2	2
		1	1	1	0	0	0	0	0	0	3	3
			2	2	2	1	1	1	1	1	1	4
	M	M	M	M	M	M	M	H	H	M	M	M

Cache Lab

Part B. Efficient Matrix Transpose

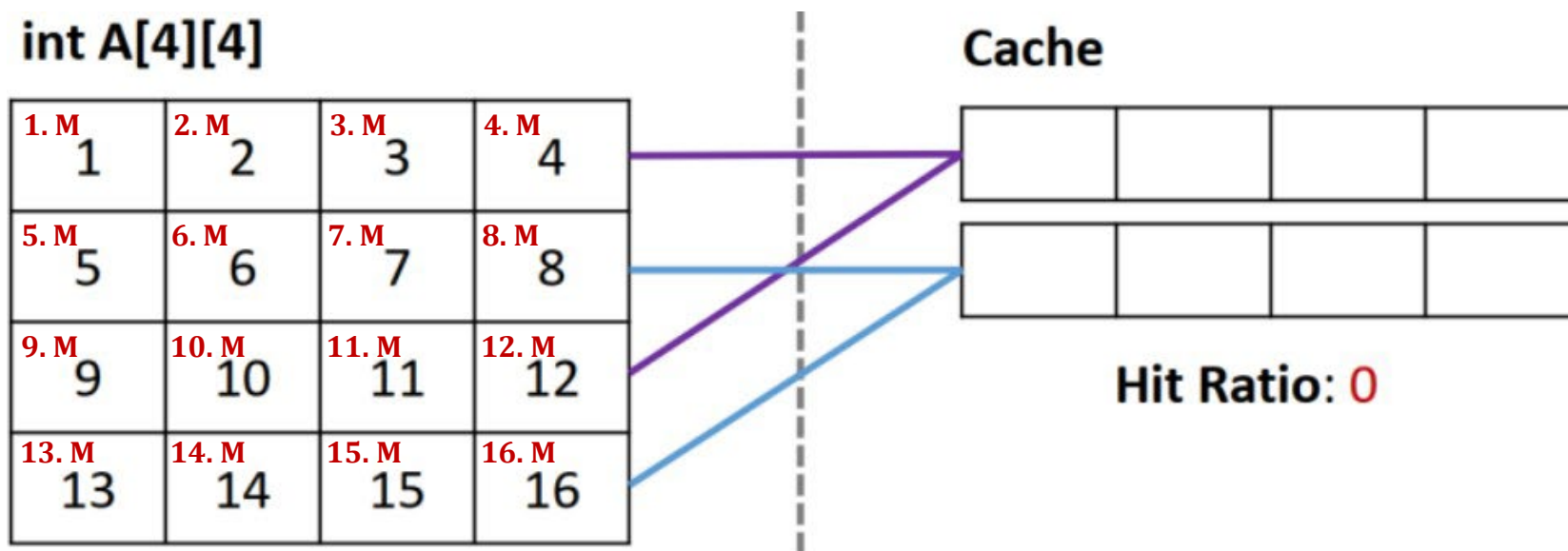
Hit Ratio

- The percentage of accesses that result in cache hits
- e.g.
 - For 32 bytes directed mapped cache with a block size of 16 bytes,
 - Accessing elements in **row-major** order:



Hit Ratio

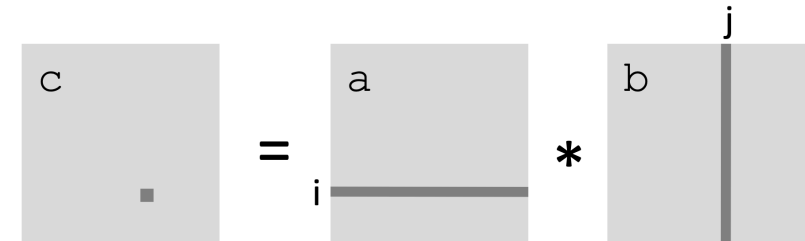
- The percentage of accesses that result in cache hits
- e.g.
 - For 32 bytes directed mapped cache with a block size of 16 bytes,
 - Accessing elements in **column-major** order:



Matrix Multiplication w/o Blocking

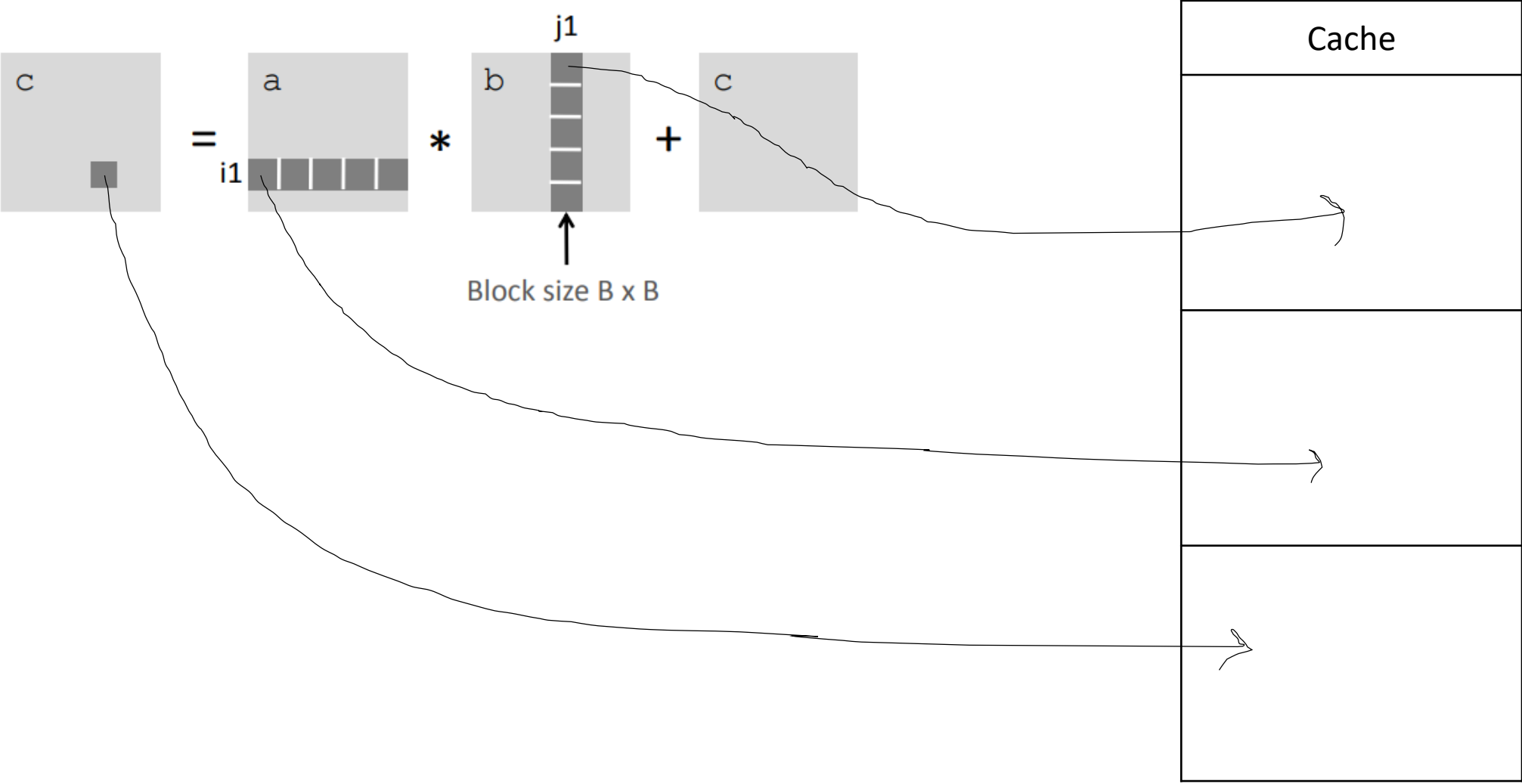
- Assume:
 1. Matrix elements are doubles
 2. Cache blocks = 8 doubles
 3. Cache size $C \ll n$

```
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];  
        }  
    }  
}
```



- # of miss in every inner iteration: $\frac{n}{8} + n = \frac{9n}{8}$
- # of total miss: $\frac{9n}{8} * n^2 = \frac{9n^3}{8}$

Blocking

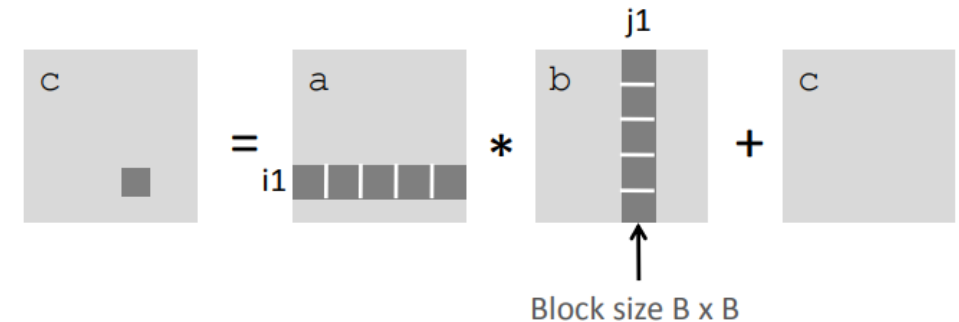


Matrix Multiplication w/ Blocking

- Assume:
 1. Matrix elements are doubles
 2. Cache blocks = 8 doubles
 3. Cache size $C \ll n$
 4. $3B^2 < C$

```
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];
}
```



- # of miss in every inner iteration: $\frac{B^2}{8} * \frac{2n}{B} = \frac{nB}{4}$
- # of total miss: $\frac{nB}{4} * \left(\frac{n}{B}\right)^2 = \frac{n^3}{4B}$

Blocking Summary

- Matrix multiplication w/o blocking: $\frac{9n^3}{8}$
- Matrix multiplication w/ blocking: $\frac{n^3}{4B}$

→ Suggest largest possible block size B , but limits $3B^2 < C$!

- For a detailed discussion of blocking:

<http://csapp.cs.cmu.edu/public/waside.html>

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 caches, Cachegrind simulates the first-level and last level caches
 - Instruction Cache: I1, L1
 - Data Cache: D1, LLd

```
[jungbeomseo@programming2 pass]$ valgrind --tool=cachegrind ./mmm
==22232== Cachegrind, a cache and branch-prediction profiler
==22232== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==22232== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==22232== Command: ./mmm
==22232==
--22232-- warning: L3 cache found, using its data for the LL simulation.
==22232==
==22232== I refs:      41,246,378
==22232== I1 misses:      761
==22232== L1i misses:     756
==22232== I1 miss rate:    0.00%
==22232== L1i miss rate:   0.00%
==22232==
==22232== D refs:    20,642,839 (19,698,859 rd + 943,980 wr)
==22232== D1 misses:    123,511 (116,181 rd + 7,330 wr)
==22232== LLd misses:    7,059 (2,107 rd + 4,952 wr)
==22232== D1 miss rate:   0.6% ( 0.6% + 0.8% )
==22232== LLd miss rate: 0.0% ( 0.0% + 0.5% )
==22232==
==22232== LL refs:      124,272 (116,942 rd + 7,330 wr)
==22232== LL misses:      7,815 (2,863 rd + 4,952 wr)
==22232== LL miss rate:   0.0% ( 0.0% + 0.5% )
```

```
[jungbeomseo@programming2 pass]$ valgrind --tool=cachegrind ./bmmm
==22251== Cachegrind, a cache and branch-prediction profiler
==22251== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==22251== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==22251== Command: ./bmmm
==22251==
--22251-- warning: L3 cache found, using its data for the LL simulation.
==22251==
==22251== I refs:      43,757,779
==22251== I1 misses:      763
==22251== L1i misses:     758
==22251== I1 miss rate:    0.00%
==22251== L1i miss rate:   0.00%
==22251==
==22251== D refs:    22,002,702 (21,009,026 rd + 993,676 wr)
==22251== D1 misses:    26,325 (18,995 rd + 7,330 wr)
==22251== LLd misses:    6,868 (1,916 rd + 4,952 wr)
==22251== D1 miss rate:   0.1% ( 0.1% + 0.7% )
==22251== LLd miss rate: 0.0% ( 0.0% + 0.5% )
==22251==
==22251== LL refs:      27,088 (19,758 rd + 7,330 wr)
==22251== LL misses:      7,626 (2,674 rd + 4,952 wr)
==22251== LL miss rate:   0.0% ( 0.0% + 0.5% )
```

Homework (Part B)

- Optimize matrix transpose ($A \rightarrow A^T$)
 - Write the efficient code for the highest hit ratio (i.e. minimize the cache miss)
- Cache:
 - You get 1 kilobytes of cache
 - Directly mapped ($E=1$)
 - Block size is 32 bytes ($b=5$)
 - There are 32 sets ($s=5$)
- Read **Programming Rules** in `writeup_cachelab.pdf` over and over!

Homework (Report)

- Deadline: 11/27 (Mon) 23:59
- You need to
 - Explain how did you optimize your matrix transpose in the report.
 - For the report, follow the format **[student#].pdf**
 - For example, 20170354.pdf (No square brackets in the file name).
 - For the code, follow the format **[student#].tar**
 - For example, 20170354.tar (No square brackets in the file name).
 - Combining Part A(csim.c) and Part B(trans.c) together.
 - No zip, No tar.gz!
- You can find more details in writeup_cachelab.pdf

Quiz
