CS 33 – Discussion 1A
Computer System Organization

Week 6

Questions before we start?

Logistics

- HW3 due tonight 11:59pm
- Lab 2 – Attack lab released - Due Friday May 15th 11:59pm
- Please submit suggestions on CCLE TA-site on how to improve OH or DISC
  - Please fill out LA survey on CCLE if you haven't already

Agenda

10am-11am PST:

- Memory Hierarchy
- Attack lab spec

11am-11:50am PST:

- LA worksheet

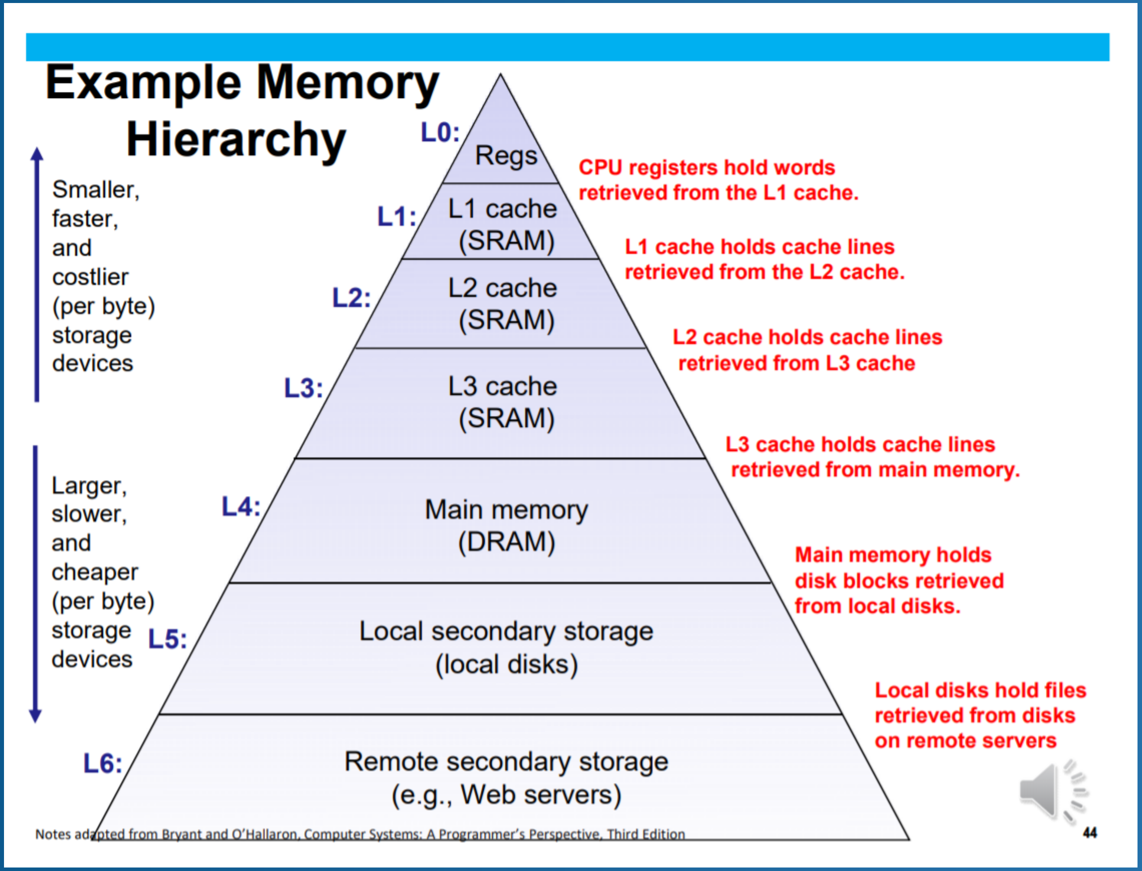
Memory Basics

- Memory technology
  - SRAM – Registers/caches – 1-10 cycles access time
    - L1/L2/L3 Cache data/instruction/unified
  - DRAM – Volatile main memory
  - Disk/Flash – Non-volatile main memory
  - Communicate between types of memory with buses
    - Bits driven out on parallel wires

- Memory reference locality
  - Spatial
  - Temporal

- Caching
  - Subset of main memory with fast access
  - Memory Mountain
  - Matrix multiplication example

Memory Basics



Memory Basics

The memory hierarchy is composed of:

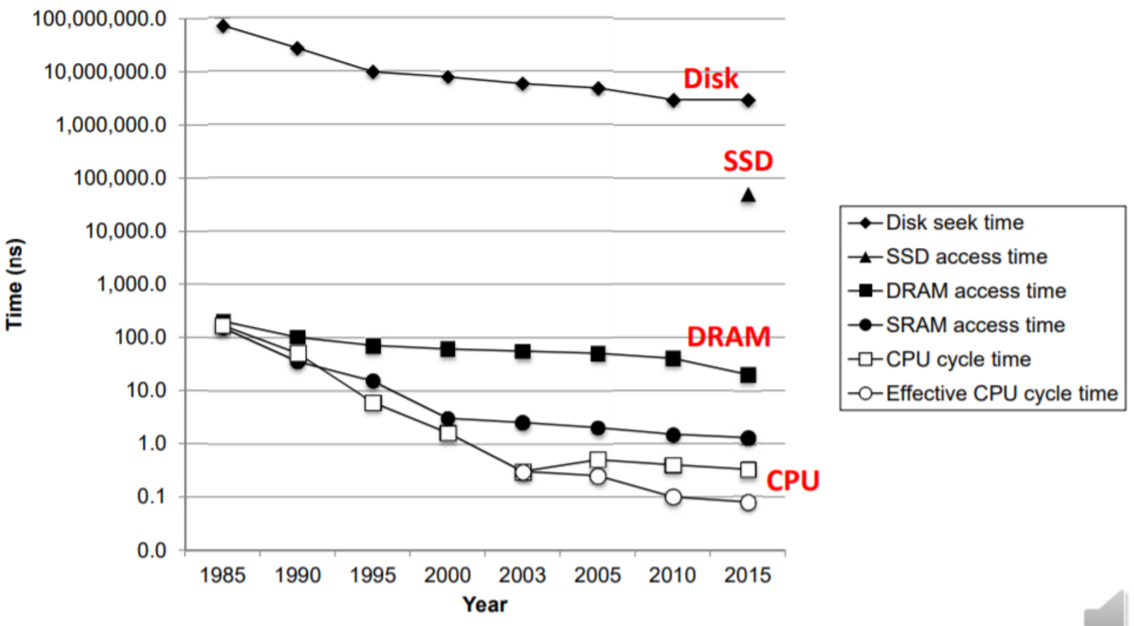
- Caches
- DRAM (4Gb, 2^32 bits)
  - (can hypothetically fit entire address space in main memory if our word size is 32) (hundreds of cycles access time)
    - Virtual memory gives each program illusion of having this
- Disk (>1Tb, >2^40 bits) (millions of cycles access time)

- Caches themselves have multiple levels:
  - L1 (~Kb, ~2^10 bits) (~1-3 cycles access)
  - L2/L3 (~Mb, ~2^20 bits)(~20-30 cycles access time L2)
    - all levels have different sizes/policies/architectures

Memory Basics

The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.



Notes adapted from Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

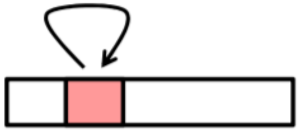
34

Memory Basics

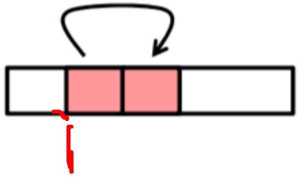
Locality

- **Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently**

- **Temporal locality:**
  - Recently referenced items are likely to be referenced again in the near future



- **Spatial locality:**
  - Items with nearby addresses tend to be referenced close together in time



Memory Basics

Locality Example

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- **Data references**
  - Reference array elements in succession (stride-1 reference pattern). **Spatial locality**
  - Reference variable `sum` each iteration. **Temporal locality**
- **Instruction references**
  - Reference instructions in sequence. **Spatial locality**
  - Cycle through loop repeatedly. **Temporal locality**

Memory Basics

Locality Example

- **Question:** Does this function have good locality with respect to array a?

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

- **Question:** Does this function have good locality with respect to array a?

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Memory Basics

Locality Example

- Question: Does this function have good locality with respect to array a?

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

- Question: Does this function have good locality with respect to array a?

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Caching Basics

- Physical memory isn't the instantly accessible extremely large contiguous array we want it to be
- Provide the fast access with the illusion of memory the size of disk by using caching and page tables for multiple programs
- Want to exploit principles of temporal and spatial locality in good cache design to reduce the cost of accessing memory to the cost of just accessing the highest level cache

Caching Basics

Block (line): Unit of storage in the cache

- Memory is logically divided into cache blocks that map to locations in the cache

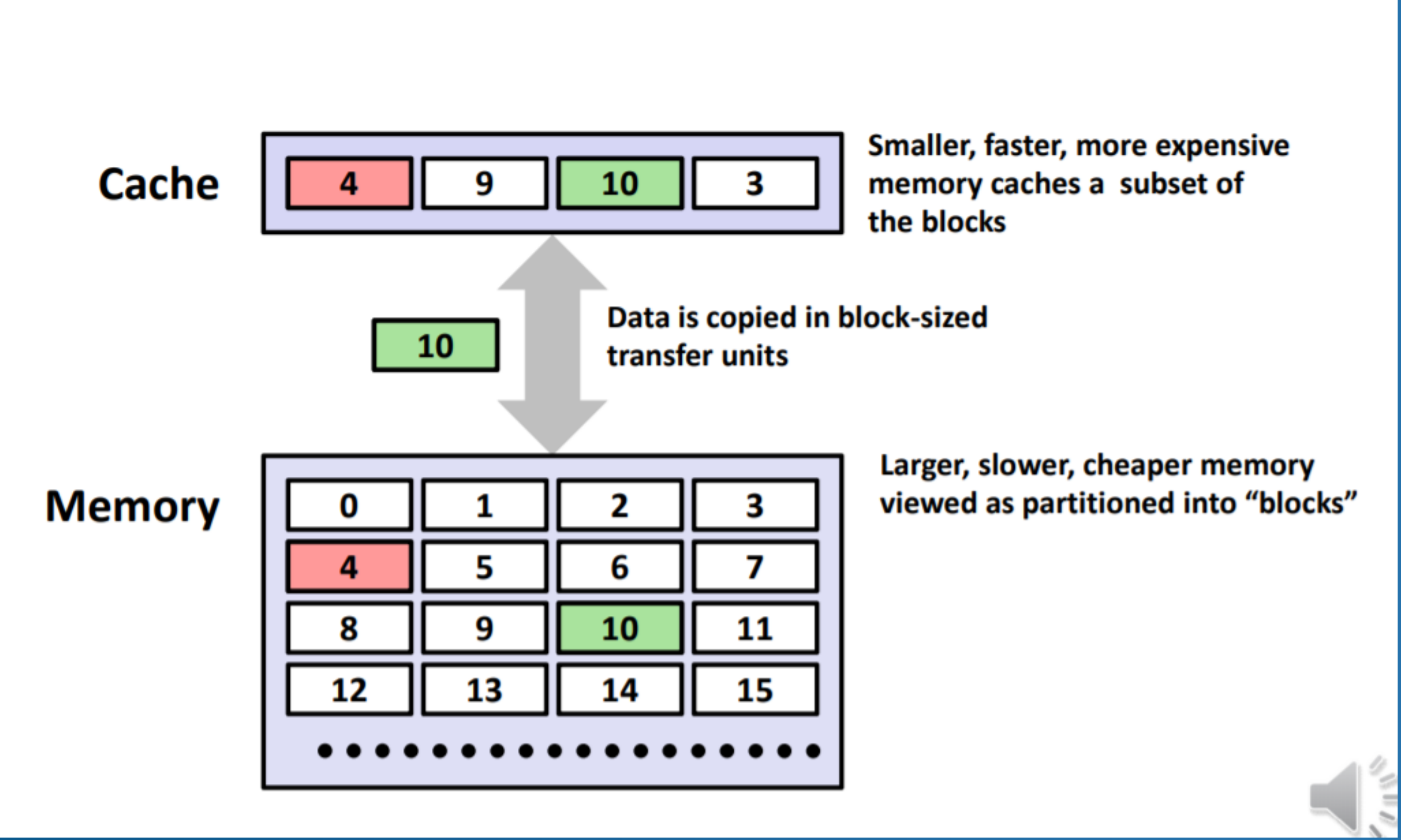
When data referenced

- HIT: if in cache, use cached data instead of accessing memory
- MISS: if not in cache, bring block into cache

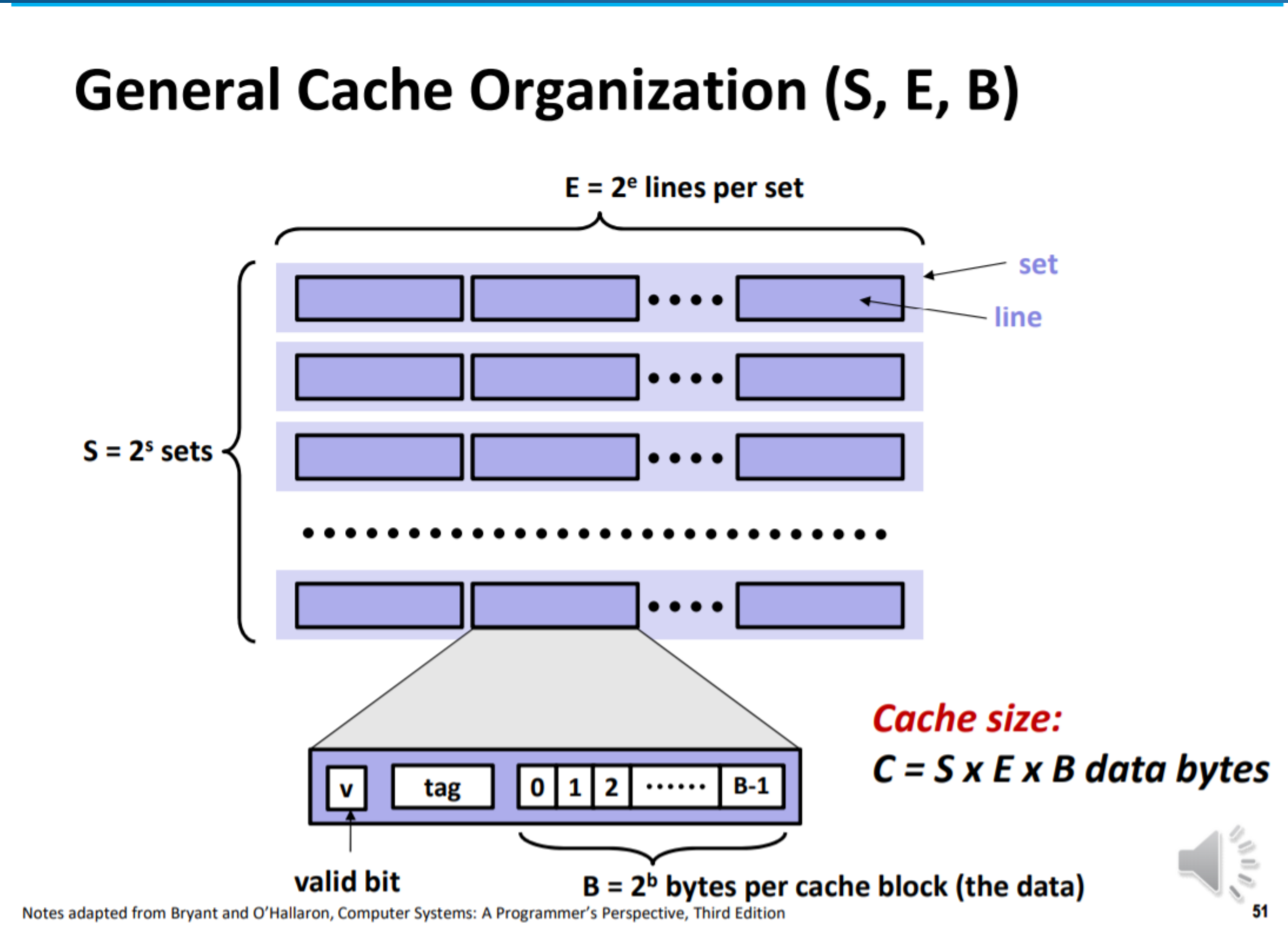
Some important cache design decisions

- Placement: where and how to place/find a block in cache?
- Replacement: what data to remove to make room in cache?
- Granularity of management: large, small, uniform blocks?
- Write policy: what do we do about writes?
- Instructions/data: do we treat them separately?

Caching Basics



Caching Basics



Caching Basics

Cache Performance Metrics

- **Miss Rate**
  - Fraction of memory references not found in cache (misses / accesses) = 1 – hit rate
  - Typical numbers (in percentages):
    - 3-10% for L1
    - can be quite small (e.g., < 1%) for L2, depending on size, etc.
- **Hit Time**
  - Time to deliver a line in the cache to the processor
    - includes time to determine whether the line is in the cache
  - Typical numbers:
    - 4 clock cycle for L1
    - 10 clock cycles for L2
- **Miss Penalty**
  - Additional time required because of a miss
    - typically 50-200 cycles for main memory (Trend: increasing!)

- Average access time:

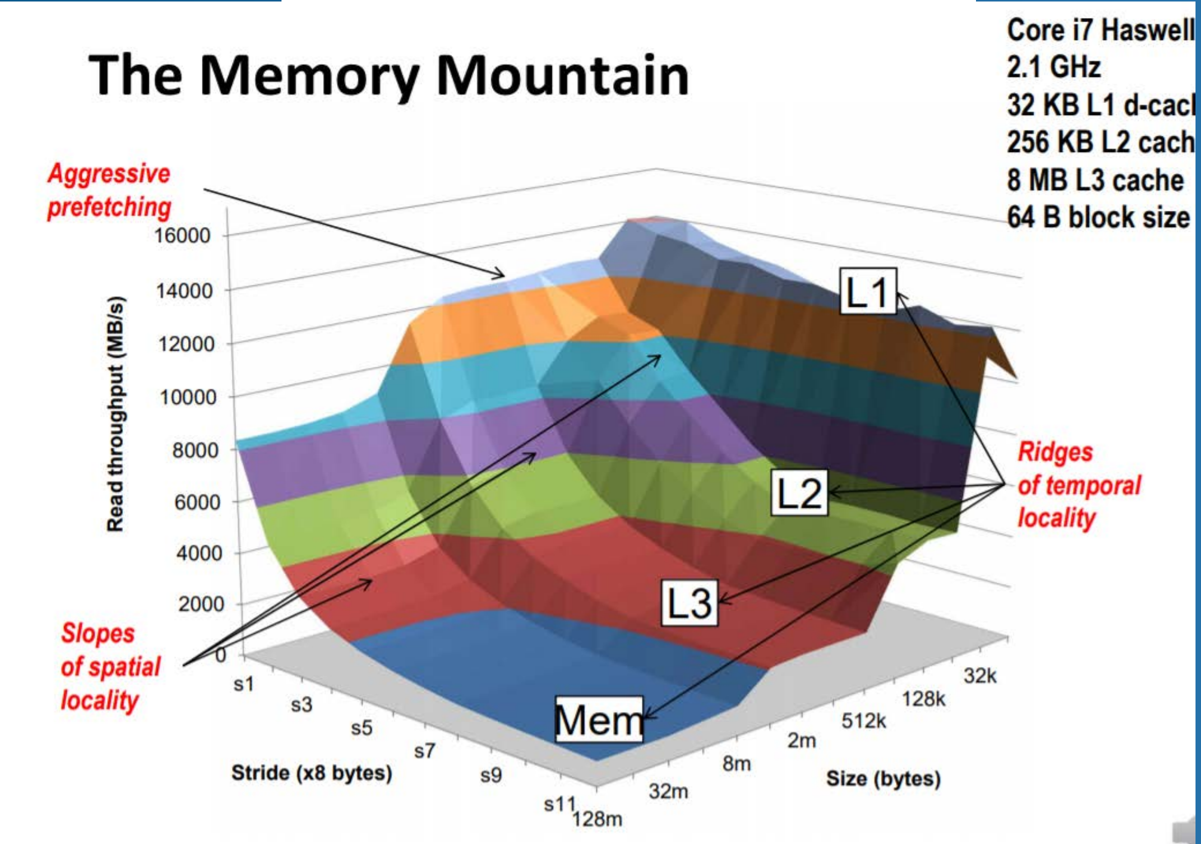
  97% hits: 1 cycle + 0.03 * 100 cycles = **4 cycles**

  99% hits: 1 cycle + 0.01 * 100 cycles = **2 cycles**

Caching Basics

The Memory Mountain

- **Read throughput (read bandwidth)**
  - Number of bytes read from memory per second (MB/s)



Notes adapted from Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caching Basics

Writing Cache Friendly Code

- **Make the common case go fast**
  - Focus on the inner loops of the core functions
- **Minimize the misses in the inner loops**
  - Repeated references to variables are good (temporal locality)
  - Stride-1 reference patterns are good (spatial locality)

Matrix Multiplication Example

- **Description:**
  - Multiply N x N matrices
  - Matrix elements are doubles (8 bytes)
  - $O(N^3)$ total operations
  - N reads per source element
  - N values summed per destination
    - but may be able to hold in register

Variable sum held in register

```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}                          matmult/mm.c
```

Caching Basics

Summary of Matrix Multiplication

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
   sum = 0.0;
   for (k=0; k<n; k++)
     sum += a[i][k] * b[k][j];
   c[i][j] = sum;
 }
}
```

ijk (& jik):
- 2 loads, 0 stores
- misses/iter = **1.25**

```
for (k=0; k<n; k++) {
 for (i=0; i<n; i++) {
  r = a[i][k];
  for (j=0; j<n; j++)
   c[i][j] += r * b[k][j];
 }
}
```

kij (& ikj):
- 2 loads, 1 store
- misses/iter = **0.5**

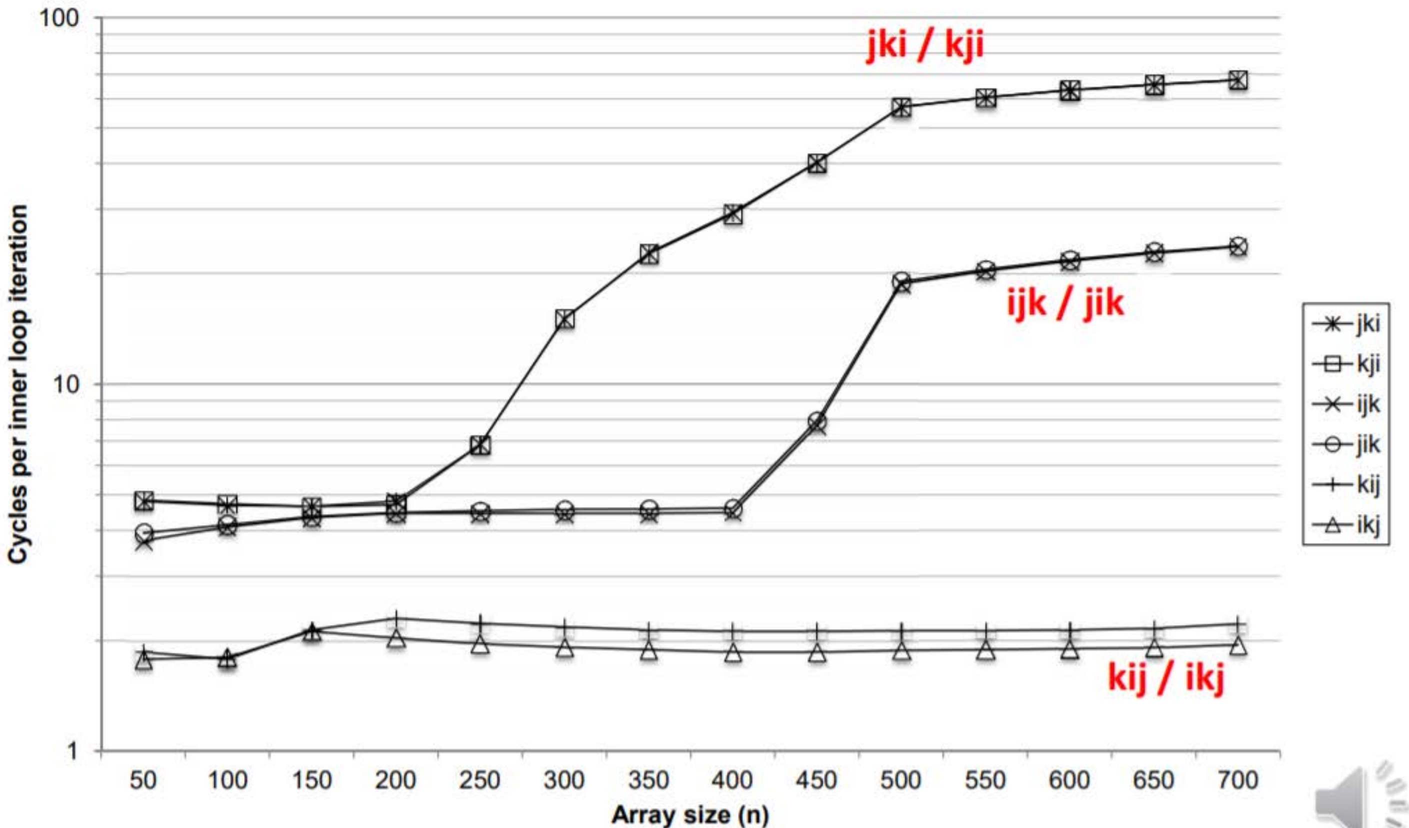
```
for (j=0; j<n; j++) {
 for (k=0; k<n; k++) {
   r = b[k][j];
   for (i=0; i<n; i++)
    c[i][j] += a[i][k] * r;
 }
}
```

jki (& kji):
- 2 loads, 1 store
- misses/iter = **2.0**

Notes adapted from Bry e, Third Edition

Caching Basics

Core i7 Matrix Multiply Performance



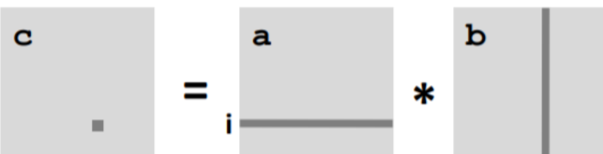
Notes adapted from Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Caching Basics

Example: Matrix Multiplication

```
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];
}
```

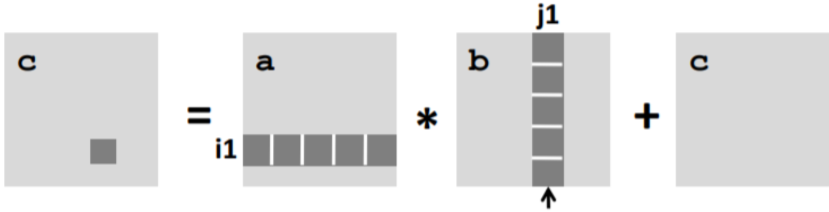


Blocked Matrix Multiplication

```
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+=B)
        for (j = 0; j < n; j+=B)
            for (k = 0; k < n; k+=B)
                /* B x B mini matrix multiplications */
                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];
}
```

matmult/bmm.c



Block size B x B

Notes adapted from Bryant and O'Hallaron, Computer Systems: A Progr ... :ion

UCLA

Ca

ent

Blocking Summary

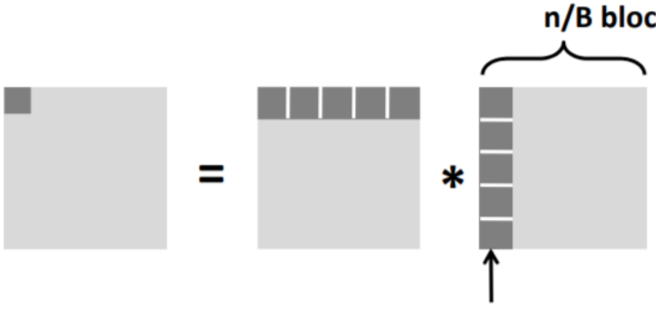
- No blocking: $(9/8) * n^3$
- Blocking: $1/(4B) * n^3$

- Suggest largest possible block size B, but limit $3B^2 < C$!

Cache Miss Analysis

- **Assume:**
  - Cache block = 8 doubles
  - Cache size C << n (much smaller than n)
  - Three blocks fit into cache: $3B^2 < C$

- **First (block) iteration:**
  - $B^2/8$ misses for each block
  - $2n/B * B^2/8 = nB/4$ (omitting matrix c)



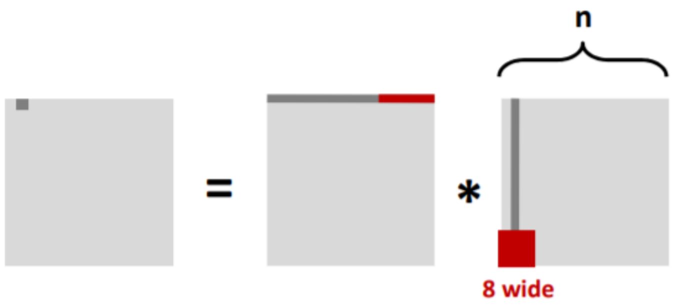
  - Afterwards in cache (schematic)

- **Total misses:**
  - $nB/4 * (n/B)^2 = n^3/(4B)$

Cache Miss Analysis

- **Assume:**
  - Matrix elements are doubles
  - Cache block = 8 doubles
  - Cache size C << n (much smaller than n)

- **Second iteration:**
  - Again: $n/8 + n = 9n/8$ misses



- **Total misses:**
  - $9n/8 * n^2 = (9/8) * n^3$

Questions before we start worksheet?

Resources used

- Professor Reinman's slides (CCLE)
- **Credit for compilation: Attiano Purpura-Pontoniere**