

CSC 411

Computer Organization (Spring 2022) Lecture 19: Cache Memory

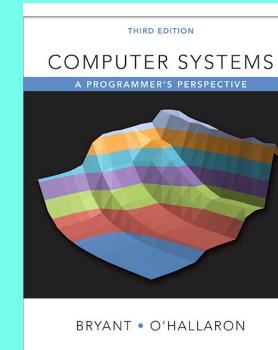
Prof. Marco Alvarez, University of Rhode Island

Disclaimer

The following slides are from:

Computer Systems (Bryant and O'Hallaron)

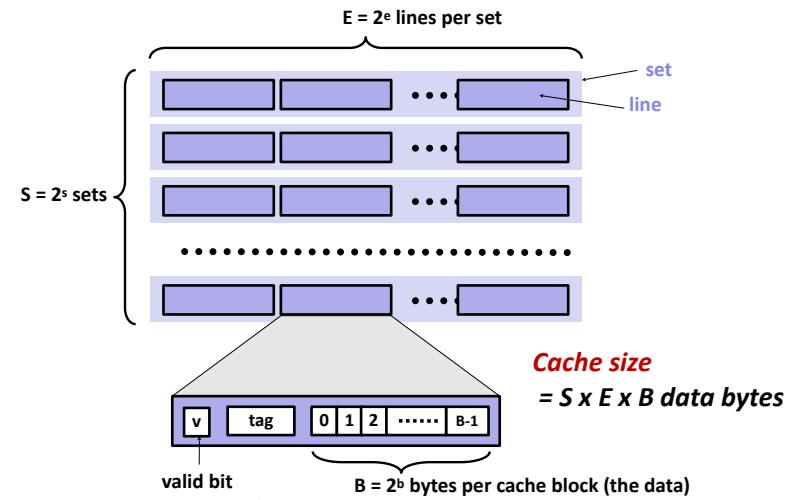
A Programmer's Perspective

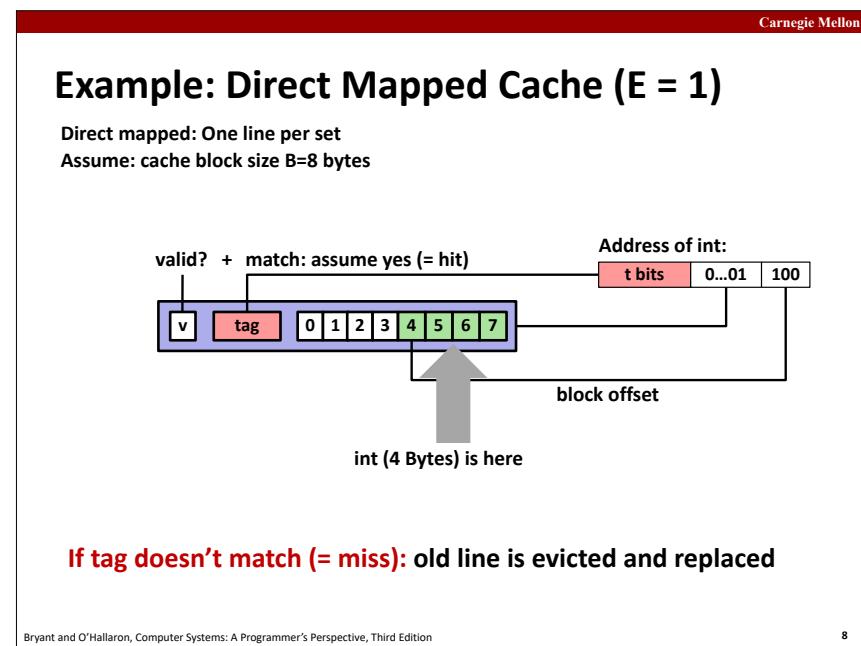
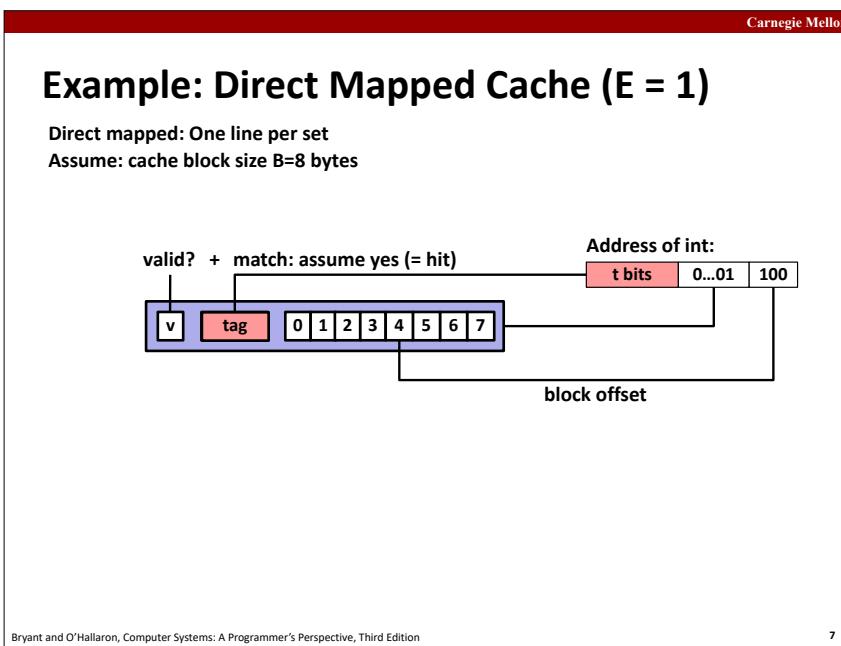
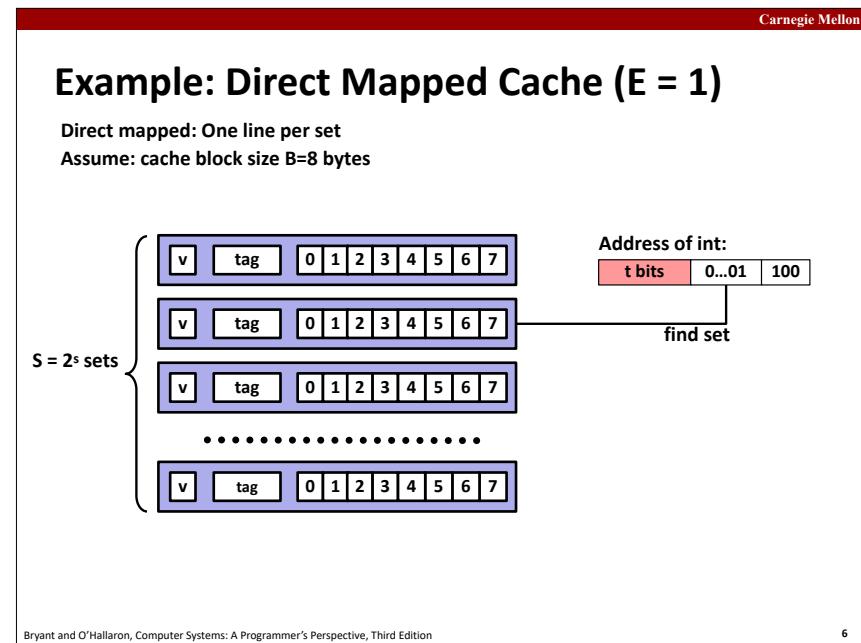
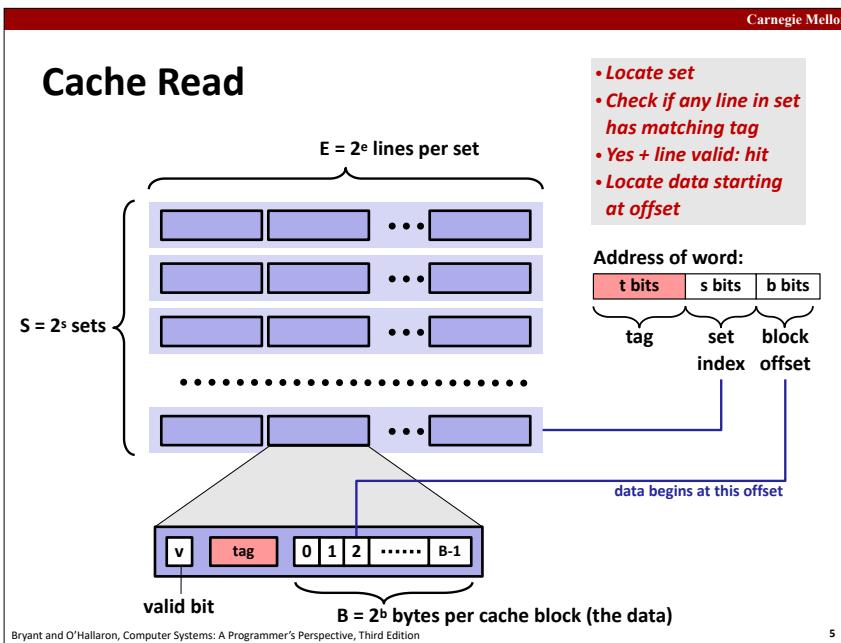


Today

- Cache memory organization and operation
- Performance impact of caches
 - The memory mountain
 - Rearranging loops to improve spatial locality
 - Using blocking to improve temporal locality

General Cache Organization (S, E, B)





Direct-Mapped Cache Simulation

t=1 s=2 b=1
 x xx x

4-bit addresses (address space size M=16 bytes)
 S=4 sets, E=1 Blocks/set, B=2 bytes/block

Address trace (reads, one byte per read):

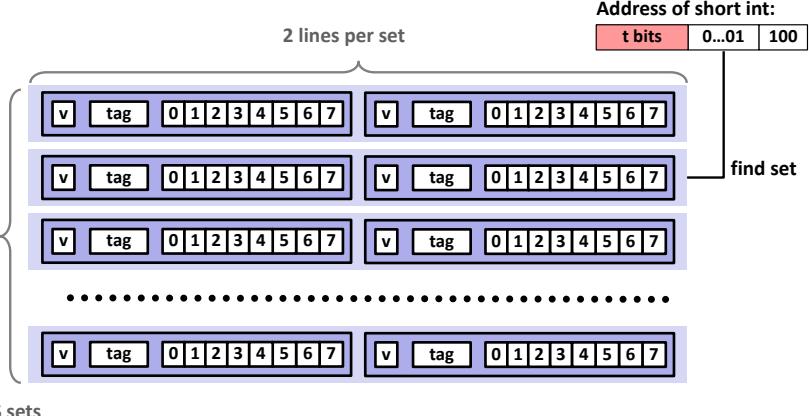
0	[0000] ₂	miss
1	[0001] ₂	hit
7	[0111] ₂	miss
8	[1000] ₂	miss
0	[0000] ₂	miss

	V	Tag	Block
Set 0	1	0	M[0-1]
Set 1	0		
Set 2	0		
Set 3	1	0	M[6-7]

E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

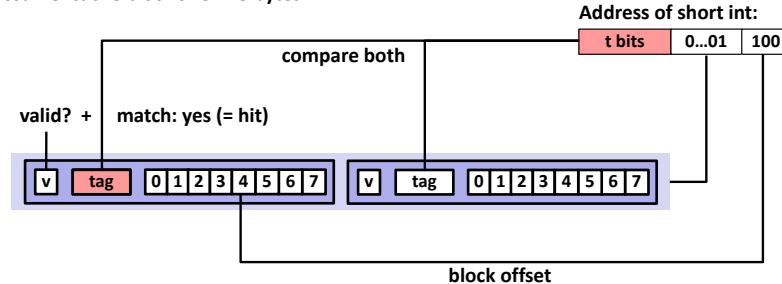
Assume: cache block size B=8 bytes



E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

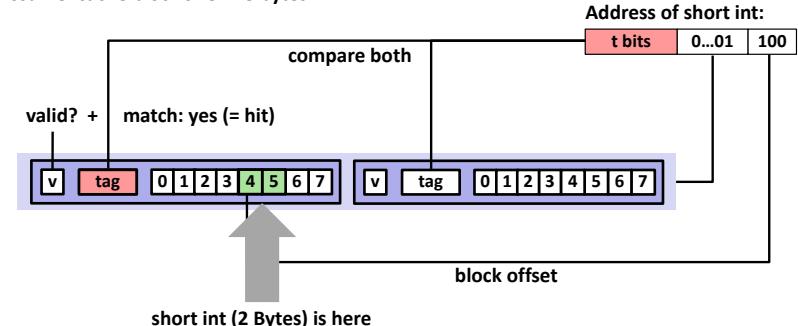
Assume: cache block size B=8 bytes



E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

Assume: cache block size B=8 bytes



No match or not valid (= miss):

- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), ...

2-Way Set Associative Cache Simulation

t=2 s=1 b=1
 XX X X

4-bit addresses (M=16 bytes)
 S=2 sets, E=2 blocks/set, B=2 bytes/block

Address trace (reads, one byte per read):

0	[0000] ₂	miss
1	[0001] ₂	hit
7	[0111] ₂	miss
8	[1000] ₂	miss
0	[0000] ₂	hit

	v	Tag	Block
Set 0	1	00	M[0-1]
	1	10	M[8-9]
Set 1	1	01	M[6-7]
	0		

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

13

What about writes?

Multiple copies of data exist:

- L1, L2, L3, Main Memory, Disk

What to do on a write-hit?

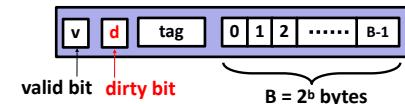
- Write-through (write immediately to memory)
- Write-back (defer write to memory until replacement of line)
 - Each cache line needs a dirty bit (set if data has been written to)

What to do on a write-miss?

- Write-allocate (load into cache, update line in cache)
 - Good if more writes to the location will follow
- No-write-allocate (writes straight to memory, does not load into cache)

Typical

- Write-through + No-write-allocate
- Write-back + Write-allocate

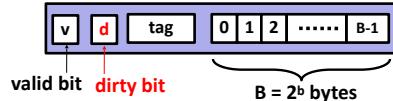


Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

14

Practical Write-back Write-allocate

A write to address X is issued



If it is a hit

- Update the contents of block
- Set dirty bit to 1 (bit is sticky and only cleared on eviction)

If it is a miss

- Fetch block from memory (per a read miss)
- The perform the write operations (per a write hit)

If a line is evicted and dirty bit is set to 1

- The entire block of 2^b bytes are written back to memory
- Dirty bit is cleared (set to 0)
- Line is replaced by new contents

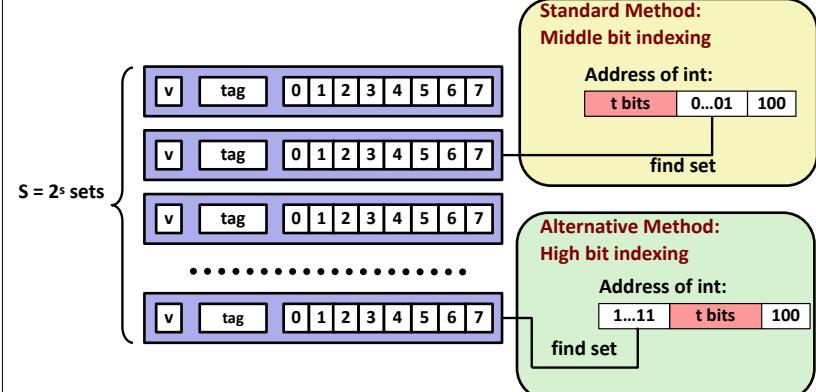
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

15

Why Index Using Middle Bits?

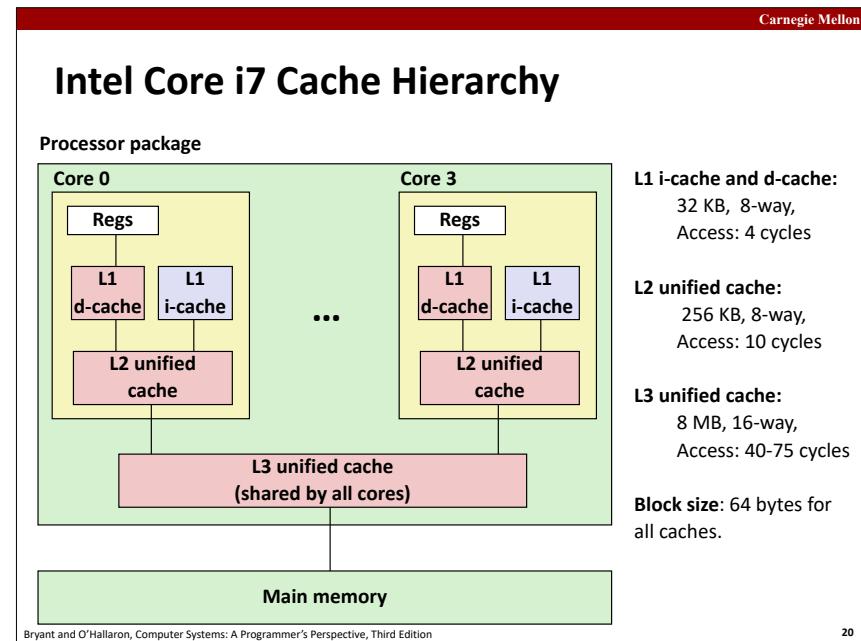
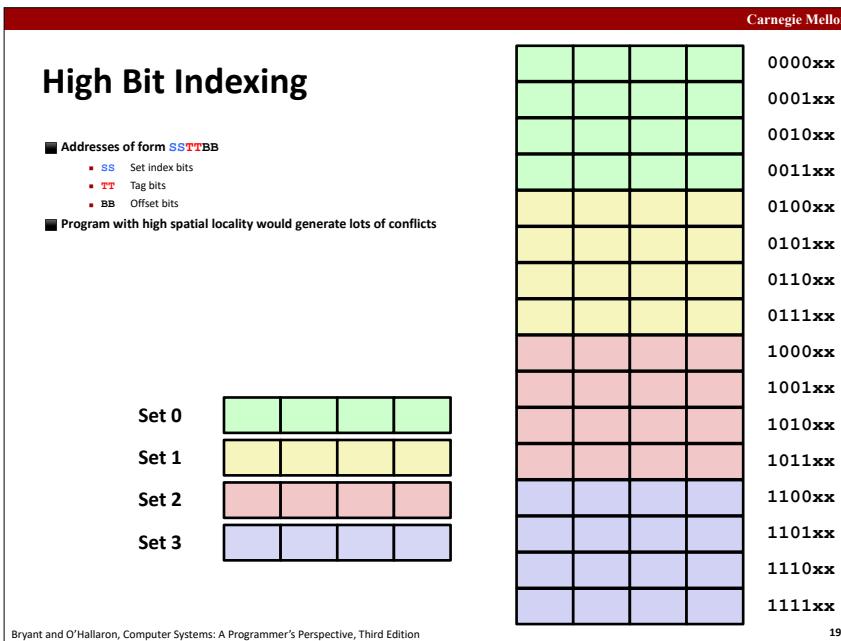
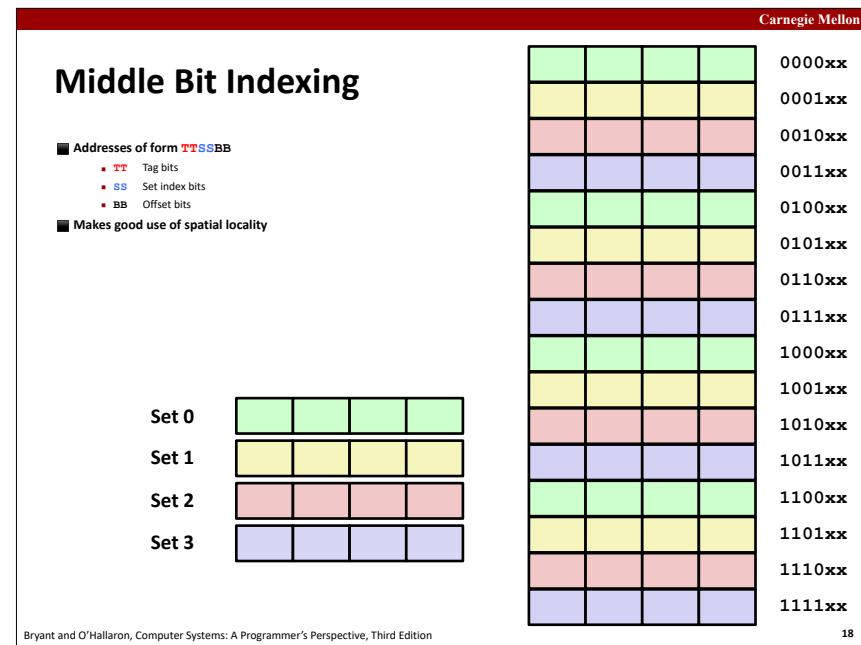
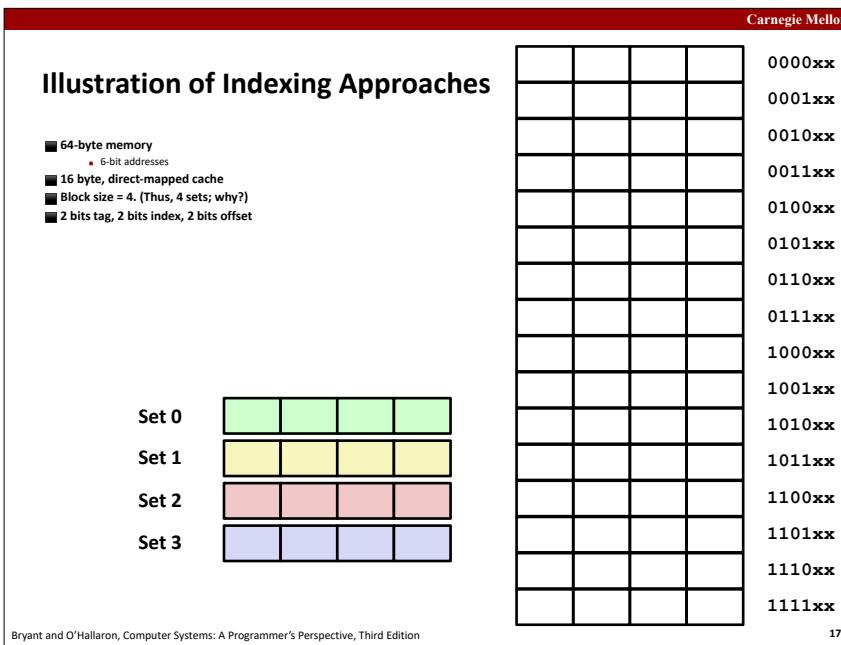
Direct mapped: One line per set

Assume: cache block size 8 bytes



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

16



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!)

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

21

Let's think about those numbers

■ Huge difference between a hit and a miss

- Could be 100x, if just L1 and main memory

■ Would you believe 99% hits is twice as good as 97%?

- Consider this simplified example:
 - cache hit time of 1 cycle
 - miss penalty of 100 cycles
- Average access time:
 - 97% hits: 1 cycle + 0.03 x 100 cycles = **4 cycles**
 - 99% hits: 1 cycle + 0.01 x 100 cycles = **2 cycles**

■ This is why “miss rate” is used instead of “hit rate”

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

22

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**)

Key idea: Our qualitative notion of locality is quantified through our understanding of cache memories

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

23

Today

■ Cache organization and operation

■ Performance impact of caches

- The memory mountain
- Rearranging loops to improve spatial locality
- Using blocking to improve temporal locality

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

24

The Memory Mountain

■ Read throughput (read bandwidth)

- Number of bytes read from memory per second (MB/s)

■ Memory mountain: Measured read throughput as a function of spatial and temporal locality.

- Compact way to characterize memory system performance.

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

25

Memory Mountain Test Function

```
long data [MAXELEMS]; /* Global array to traverse */

/* test - Iterate over first "elems" elements of
 *        array "data" with stride of "stride",
 *        using 4x4 loop unrolling.
 */
int test(int elems, int stride) {
    long i, sx2=stride*2, sx3=stride*3, sx4=stride*4;
    long acc0 = 0, acc1 = 0, acc2 = 0, acc3 = 0;
    long length = elems, limit = length - sx4;

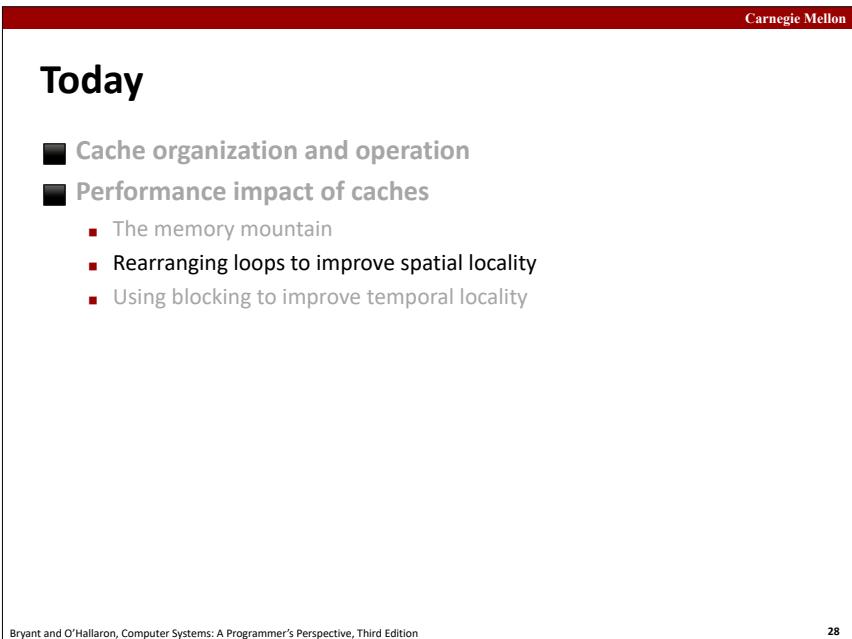
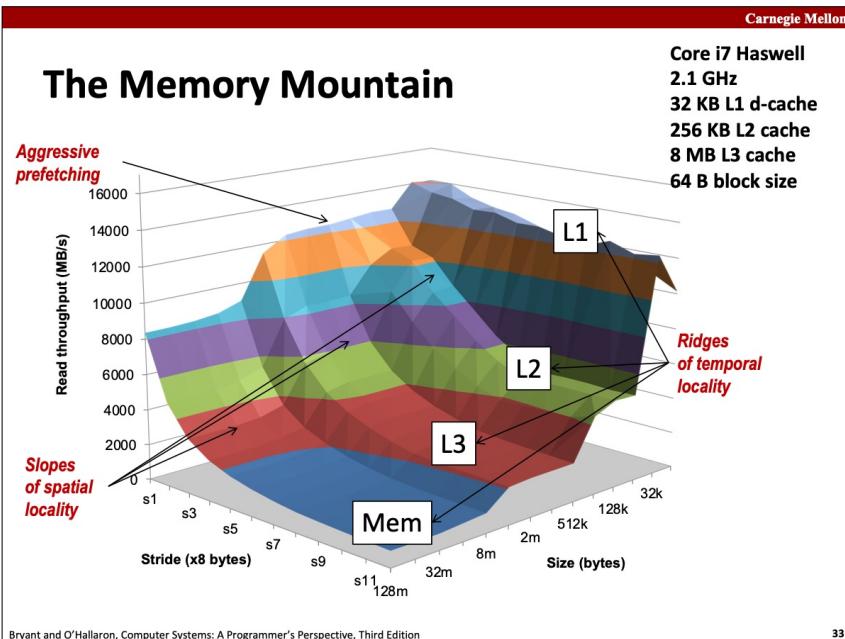
    /* Combine 4 elements at a time */
    for (i = 0; i < limit; i += sx4) {
        acc0 = acc0 + data[i];
        acc1 = acc1 + data[i+stride];
        acc2 = acc2 + data[i+sx2];
        acc3 = acc3 + data[i+sx3];
    }

    /* Finish any remaining elements */
    for (; i < length; i++) {
        acc0 = acc0 + data[i];
    }
    return ((acc0 + acc1) + (acc2 + acc3));
}
```

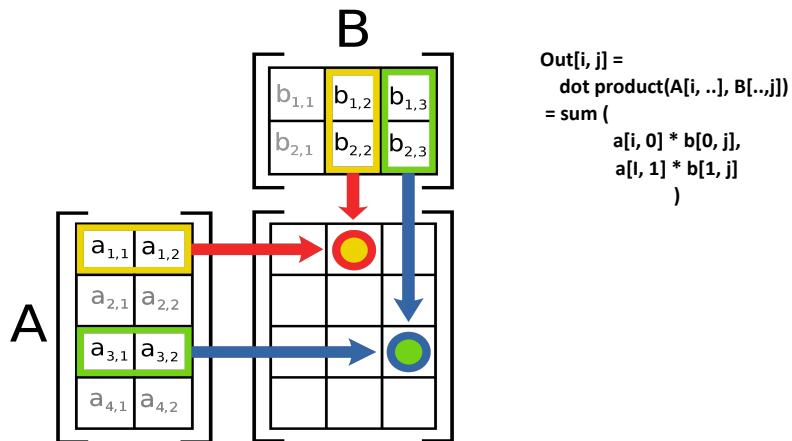
mountain/mountain.c

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

26



Remember matrix multiplication



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

29

Matrix Multiplication Example

■ Description:

- Multiply $N \times N$ 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

```
/* 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

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

30

Miss Rate Analysis for Matrix Multiply

■ Assume:

- Block size = 32B (big enough for four doubles)
- Matrix dimension (N) is very large
 - Approximate $1/N$ as 0.0
- Cache is not even big enough to hold multiple rows

■ Analysis Method:

- Look at access pattern of inner loop

$$\begin{array}{c} \overrightarrow{j} \\ | \\ \text{C} \end{array} = \begin{array}{c} \overrightarrow{i} \\ | \\ \text{A} \end{array} \times \begin{array}{c} \overrightarrow{k} \\ | \\ \text{B} \end{array}$$

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

31

Layout of C Arrays in Memory (review)

■ C arrays allocated in row-major order

- each row in contiguous memory locations
- $a[i][j] = a[i*N + j]$ where N is the number of columns

■ Stepping through columns in one row:

- $\text{for } (i = 0; i < N; i++)$
 $\quad \text{sum} += a[0][i];$
- accesses successive elements
- if block size (B) > $\text{sizeof}(a_{ij})$ bytes, exploit spatial locality
 - miss rate = $\text{sizeof}(a_{ij}) / B$

■ Stepping through rows in one column:

- $\text{for } (i = 0; i < n; i++)$
 $\quad \text{sum} += a[i][0];$
- accesses distant elements
- no spatial locality!
 - miss rate = 1 (i.e. 100%)

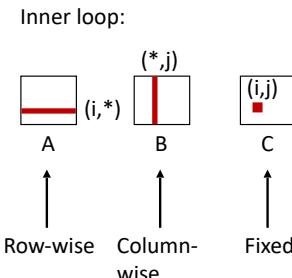
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

32

Matrix Multiplication (ijk)

```
/* 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



Miss rate for inner loop iterations:

A B C

Block size = 32B (four doubles)

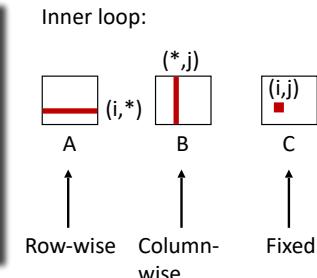
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

33

Matrix Multiplication (ijk)

```
/* 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



Miss rate for inner loop iterations:

A	B	C
0.25	1.0	0.0

Block size = 32B (four doubles)

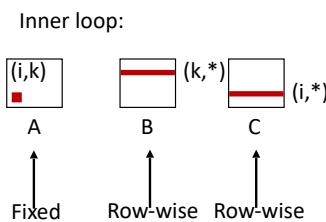
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

34

Matrix Multiplication (kij)

```
/* kij */
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];
    }
}
```

matmult/mm.c



Miss rate for inner loop iterations:

A B C

Block size = 32B (four doubles)

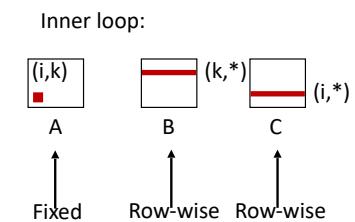
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

35

Matrix Multiplication (kij)

```
/* kij */
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];
    }
}
```

matmult/mm.c



Miss rate for inner loop iterations:

A	B	C
0.0	0.25	0.25

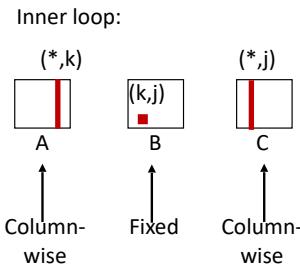
Block size = 32B (four doubles)

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

36

Matrix Multiplication (jki)

```
/* jki */
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;
    }
}
matmult/mm.c
```



Miss rate for inner loop iterations:

A B C

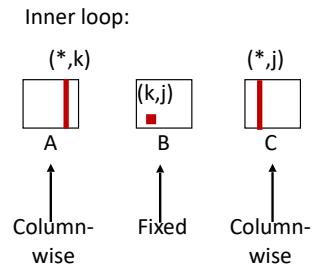
Block size = 32B (four doubles)

37

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

Matrix Multiplication (jki)

```
/* jki */
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;
    }
}
matmult/mm.c
```



Miss rate for inner loop iterations:

A B C
1.0 0.0 1.0

Block size = 32B (four doubles)

38

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

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
- avg 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
- avg 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 (& kjci):

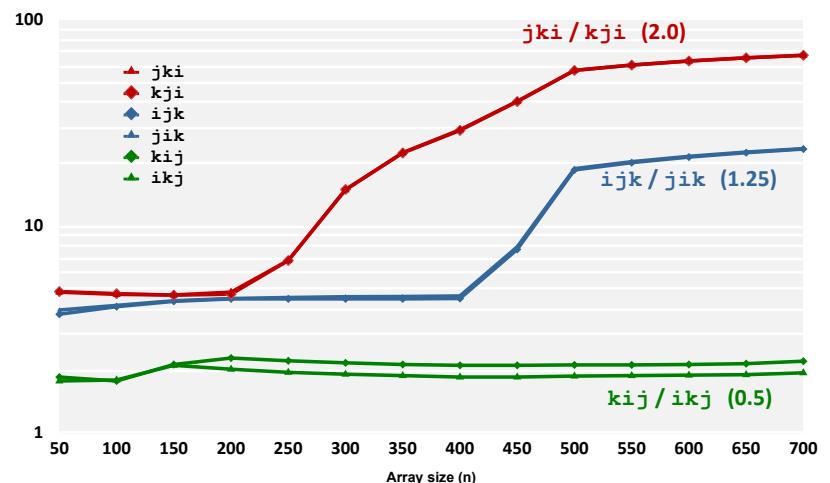
- 2 loads, 1 store
- avg misses/iter = 2.0

Bryant and O'Hallaron,

39

Core i7 Matrix Multiply Performance

Cycles per inner loop iteration



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

40

Today

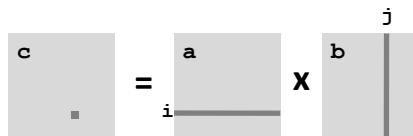
- Cache organization and operation
- Performance impact of caches
 - The memory mountain
 - Rearranging loops to improve spatial locality
 - Using blocking to improve temporal locality

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

41

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];
}
```



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

42

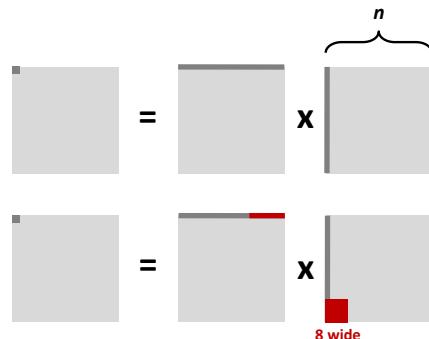
Cache Miss Analysis

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

First iteration:

- $n/8 + n = 9n/8$ misses

- Afterwards in cache: (schematic)



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

43

Cache Miss Analysis

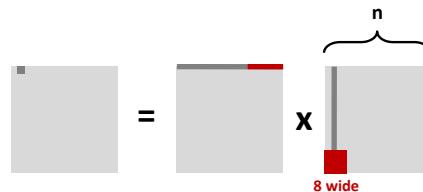
- Assume:
 - Matrix elements are doubles
 - Cache block = 8 doubles
 - Cache size C << n (much smaller than n)

Second iteration:

- Again:
 $n/8 + n = 9n/8$ misses

Total misses:

$$9n/8 \cdot n^2 = (9/8) n^3$$



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

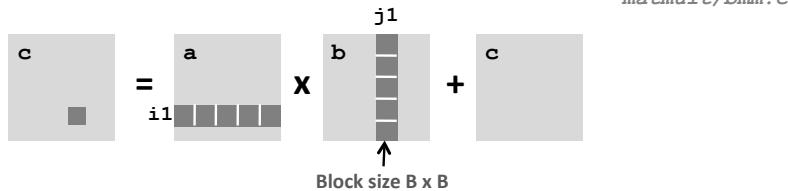
44

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



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

45

Cache Miss Analysis

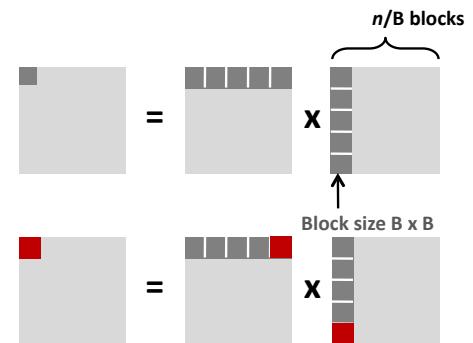
Assume:

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

First (block) iteration:

- $B^2/8$ misses for each block
- $2n/B \times B^2/8 = nB/4$
(omitting matrix c)

Afterwards in cache
(schematic)



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

46

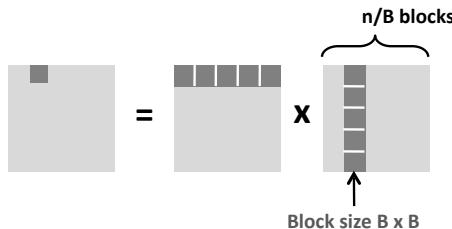
Cache Miss Analysis

Assume:

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

Second (block) iteration:

- Same as first iteration
- $2n/B \times B^2/8 = nB/4$



Total misses:

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

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

47

Blocking Summary

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

Use largest block size B , such that B satisfies $3B^2 < C$

- Fit three blocks in cache! Two input, one output.

Reason for dramatic difference:

- Matrix multiplication has inherent temporal locality:
 - Input data: $3n^2$, computation $2n^3$
 - Every array elements used $O(n)$ times!
- But program has to be written properly

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

48

Cache Summary

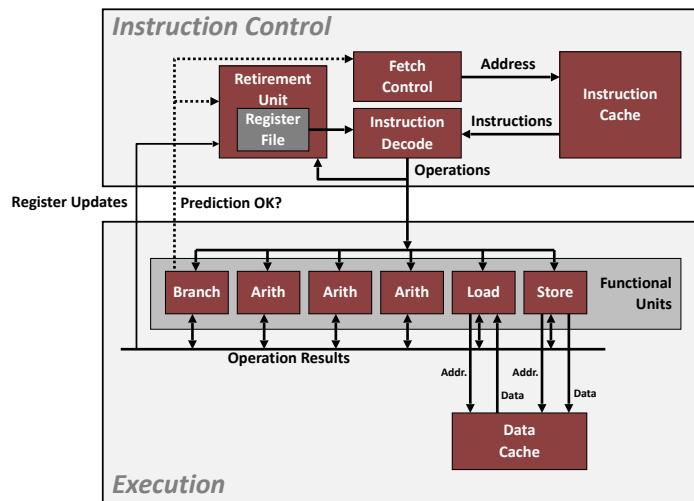
Cache memories can have significant performance impact

You can write your programs to exploit this!

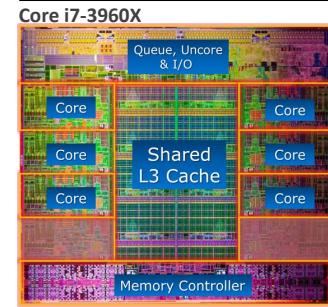
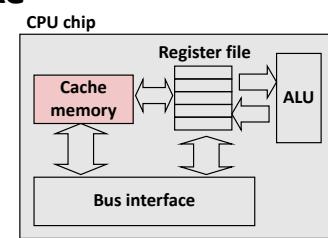
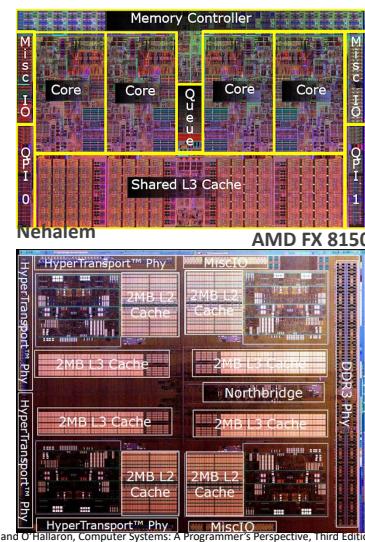
- Focus on the inner loops, where bulk of computations and memory accesses occur.
- Try to maximize spatial locality by reading data objects sequentially with stride 1.
- Try to maximize temporal locality by using a data object as often as possible once it's read from memory.

Supplemental slides

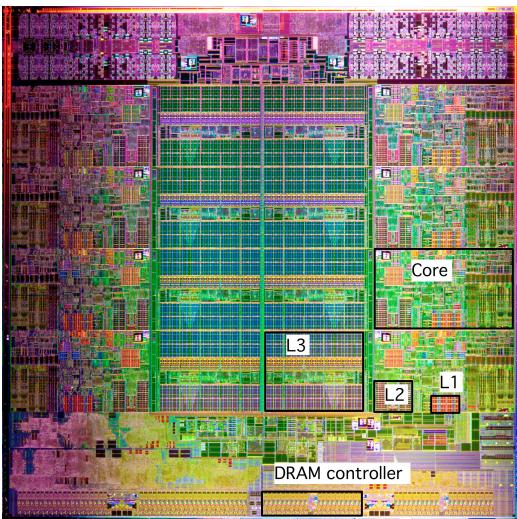
Recall: Modern CPU Design



What it Really Looks Like



What it Really Looks Like (Cont.)



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

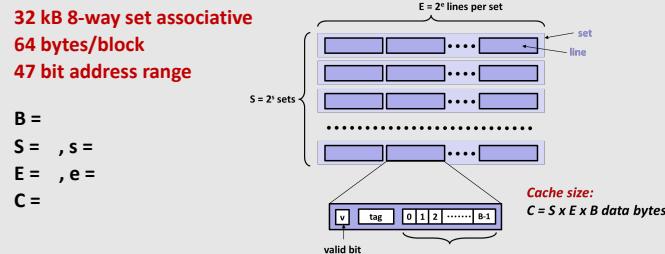
53

Example: Core i7 L1 Data Cache

32 kB 8-way set associative

64 bytes/block

47 bit address range

 $B =$ $S = 2^6$, $s = 6$ $E = 8$, $e = 3$ $C =$ 

	Hex	Decimal	Binary
0	0	0000	0000
1	1	0001	0001
2	2	0010	0010
3	3	0011	0011
4	4	0100	0100
5	5	0101	0101
6	6	0110	0110
7	7	0111	0111
8	8	1000	1000
9	9	1001	1001
A	10	1010	1010
B	11	1011	1011
C	12	1100	1100
D	13	1101	1101
E	14	1110	1110
F	15	1111	1111

Address of word:
 t bits s bits b bits
 tag set index block offset

Stack Address:
0x00007f7262a1e010

Block offset: 0x??
 Set index: 0x??
 Tag: 0x??

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

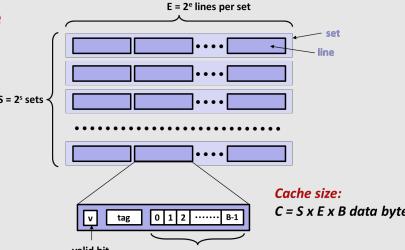
54

Example: Core i7 L1 Data Cache

32 kB 8-way set associative

64 bytes/block

47 bit address range

 $B = 64$ $S = 64$, $s = 6$ $E = 8$, $e = 3$ $C = 64 \times 64 \times 8 = 32,768$ 

	Hex	Decimal	Binary
0	0	0000	0000
1	1	0001	0001
2	2	0010	0010
3	3	0011	0011
4	4	0100	0100
5	5	0101	0101
6	6	0110	0110
7	7	0111	0111
8	8	1000	1000
9	9	1001	1001
A	10	1010	1010
B	11	1011	1011
C	12	1100	1100
D	13	1101	1101
E	14	1110	1110
F	15	1111	1111

Address of word:
 t bits s bits b bits
 tag set index block offset

Block offset: 6 bits

Set index: 6 bits

Tag: 35 bits

Stack Address:
0x00007f7262a1e010

Block offset: 0x10
 Set index: 0x0
 Tag: 0x7f7262a1e

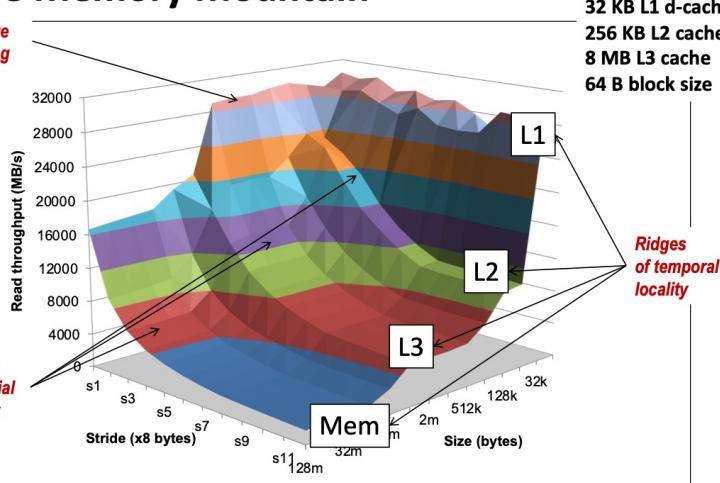
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

55

The Memory Mountain

Aggressive prefetching

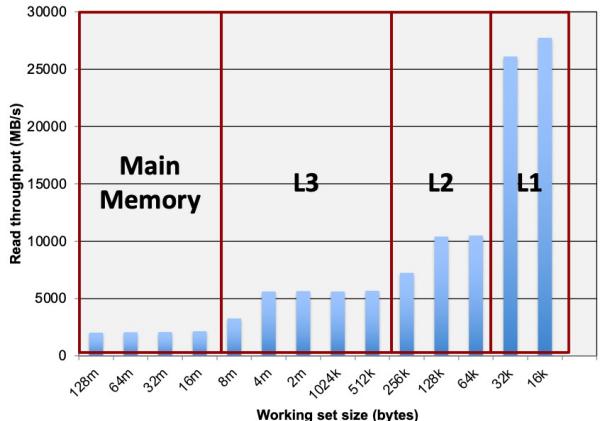
Ridges of temporal locality
 Slopes of spatial locality



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

66

Cache Capacity Effects from Memory Mountain



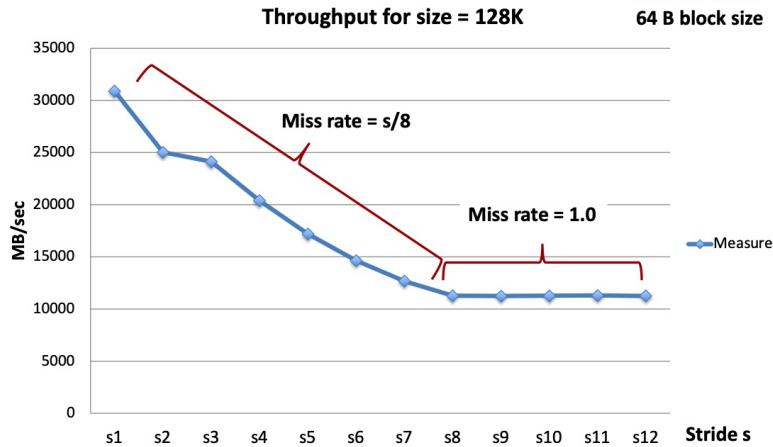
Core i7 Haswell
3.1 GHz
32 KB L1 d-cache
256 KB L2 cache
8 MB L3 cache
64 B block size

Slice through memory mountain with stride=8

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

67

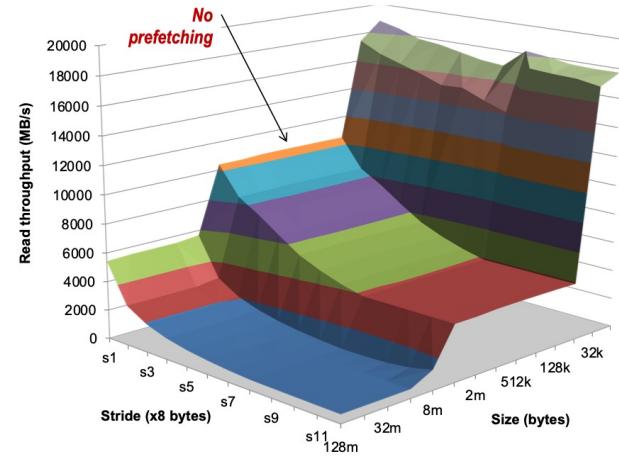
Cache Block Size Effects from Memory Mountain



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

68

2008 Memory Mountain



Core 2 Duo
2.4 GHz
32 KB L1 d-cache
6MB L2 cache
64 B block size

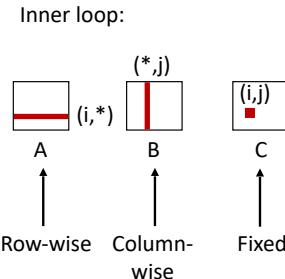
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

70

Matrix Multiplication (jik)

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

matmult/mm.c



Misses per inner loop iteration:

 A 0.25	 B 1.0	 C 0.0
---------------	--------------	--------------

Block size = 32B (four doubles)

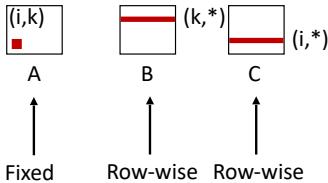
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

69

Matrix Multiplication (ikj)

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

Inner loop:

Misses per inner loop iteration:

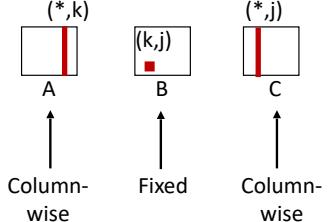
A	B	C
0.0	0.25	

Block size = 32B (four doubles)

Matrix Multiplication (kji)

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

Inner loop:

Misses per inner loop iteration:

A	B	C
1.0	0.0	

Block size = 32B (four doubles)