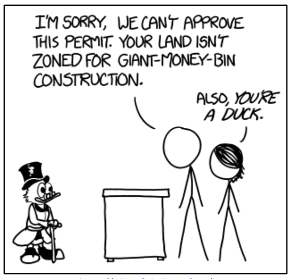
Caches III

CSE 351 Winter 2018

Instructor:
Mark Wyse

Teaching Assistants:
Kevin Bi
Parker DeWilde
Emily Furst
Sarah House
Waylon Huang
Vinny Palaniappan



I'M SORRY, WE CAN'T APPROVE THIS PERMIT. YOUR LAND ISN'T ZONED FOR GIANT-MONEY-BIN CONSTRUCTION.
ALSO, YOU'RE A DUCK.

https://what-if.xkcd.com/111/

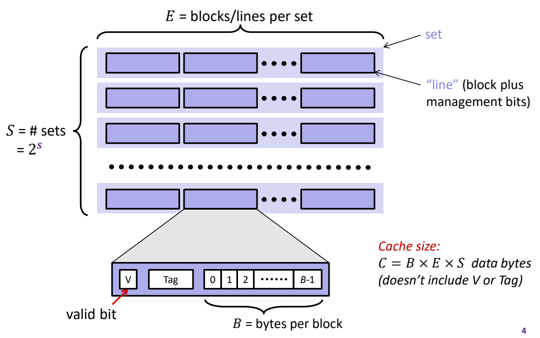
Administrative

- Midterm regrade requests due today
- Lab 3 due today!
- HW 4 out, due Friday, February 23
- No lecture on Monday – President's Day!
  - OH also cancelled

Making memory accesses fast!

- Cache basics
- Principle of locality
- Memory hierarchies
- Cache organization
  - Direct-mapped (*sets*; index + tag)
  - Associativity (*ways*)
  - Replacement policy
  - **Handling writes**
- **Program optimizations that consider caches**

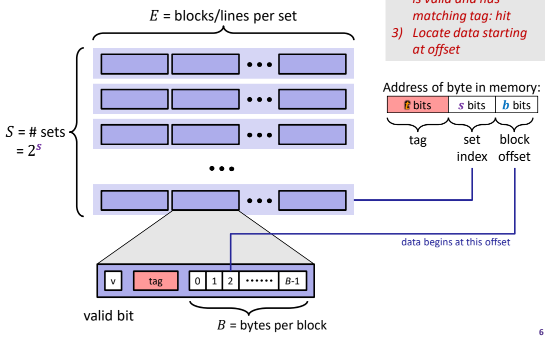
General Cache Organization (S, E, B)



Cache size:
$C = B \times E \times S$ *data bytes*
(doesn't include V or Tag)

Cache Read

1) *Locate set*
2) *Check if any line in set is valid and has matching tag: hit*
3) *Locate data starting at offset*



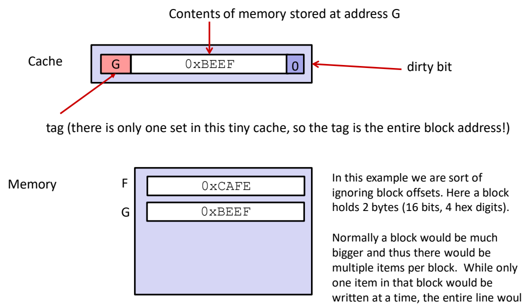
Types of Cache Misses: 3 C's!

- Compulsory (cold) miss
  - Occurs on first access to a block
- Conflict miss
  - Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot
    - *e.g.* referencing blocks 0, 8, 0, 8, ... could miss every time
  - Direct-mapped caches have more conflict misses than E-way set-associative (where $E > 1$)
- Capacity miss
  - Occurs when the set of active cache blocks (the *working set*) is larger than the cache (just won't fit, even if cache was *fully-associative*)
  - **Note:** *Fully-associative* only has Compulsory and Capacity misses

What about writes?

- Multiple copies of data exist:
  - L1, L2, possibly L3, main memory
- What to do on a write-hit?
  - Write-through: write immediately to next level
  - Write-back: defer write to next level until line is evicted (replaced)
    - Must track which cache lines have been modified ("*dirty bit*")
- What to do on a write-miss?
  - Write-allocate: ("fetch on write") load into cache, update line in cache
    - Good if more writes or reads to the location follow
  - No-write-allocate: ("write around") just write immediately to memory
- Typical caches:
  - Write-back + Write-allocate, usually
  - Write-through + No-write-allocate, occasionally

Write-back, write-allocate example

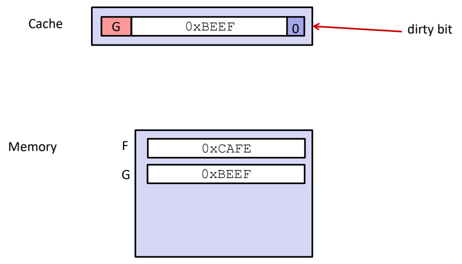

Contents of memory stored at address G
Cache
G
0xBEEF
0
dirty bit
tag (there is only one set in this tiny cache, so the tag is the entire block address!)
Memory
F
0xCAFE
G
0xBEEF

In this example we are sort of ignoring block offsets. Here a block holds 2 bytes (16 bits, 4 hex digits).

Normally a block would be much bigger and thus there would be multiple items per block. While only one item in that block would be written at a time, the entire line would be brought into cache.

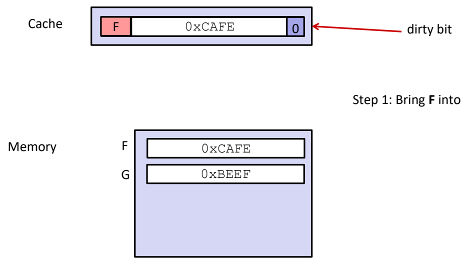
Write-back, write-allocate example

```
mov 0xFACE, F
```



Write-back, write-allocate example

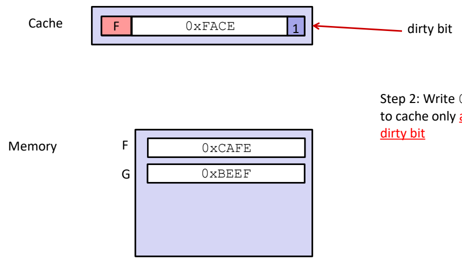
```
mov 0xFACE, F
```



Step 1: Bring **F** into cache

Write-back, write-allocate example

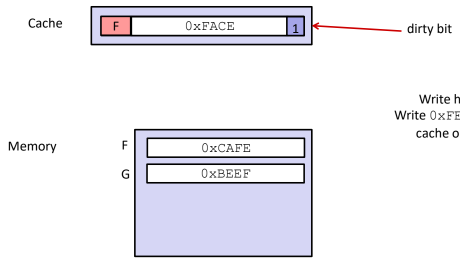
```
mov 0xFACE, F
```



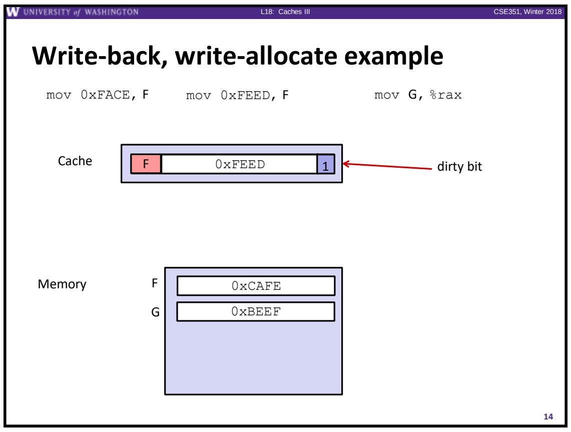
Step 2: Write 0xFACE to cache only and set dirty bit

Write-back, write-allocate example

```
mov 0xFACE, F      mov 0xFEED, F
```

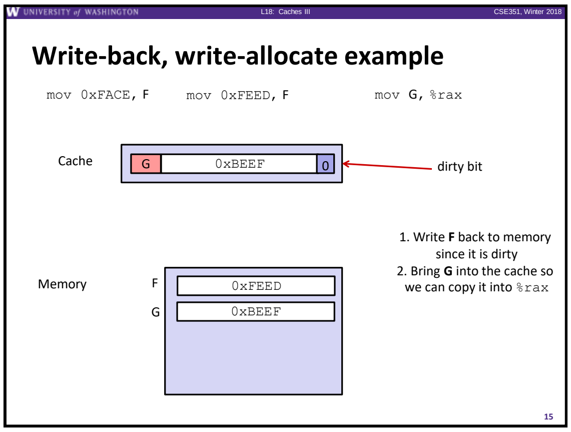


Write hit!
Write 0xFEED to cache only



Write-back, write-allocate example

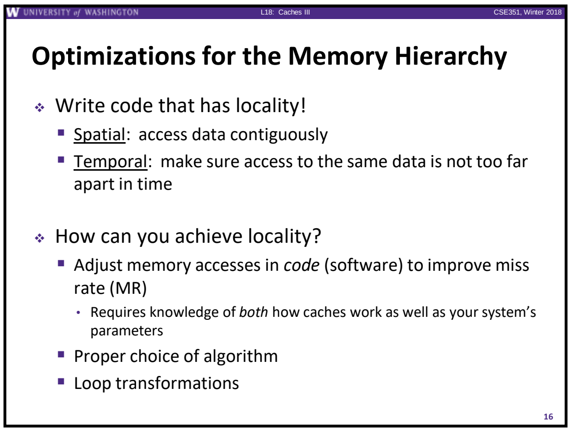
`mov 0xFACE, F` `mov 0xFEED, F` `mov G, %rax`



Write-back, write-allocate example

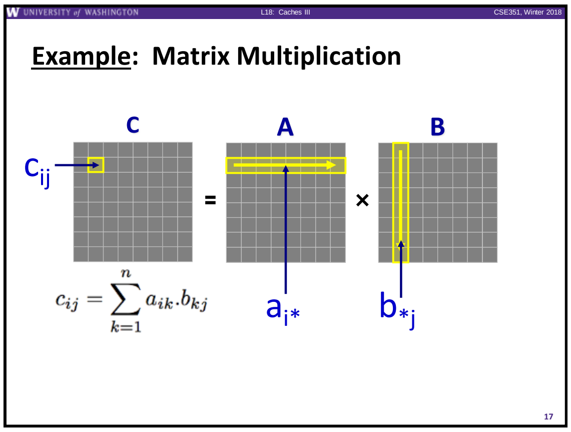
`mov 0xFACE, F` `mov 0xFEED, F` `mov G, %rax`

1. Write **F** back to memory since it is dirty
2. Bring **G** into the cache so we can copy it into `%rax`



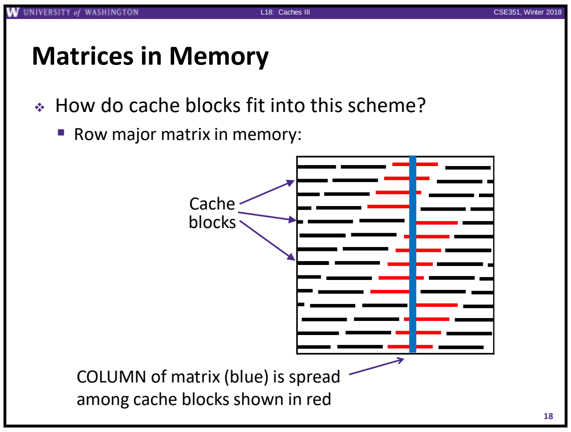
Optimizations for the Memory Hierarchy

- Write code that has locality!
  - Spatial: access data contiguously
  - Temporal: make sure access to the same data is not too far apart in time
- How can you achieve locality?
  - Adjust memory accesses in *code* (software) to improve miss rate (MR)
    - Requires knowledge of *both* how caches work as well as your system's parameters
  - Proper choice of algorithm
  - Loop transformations



Example: Matrix Multiplication

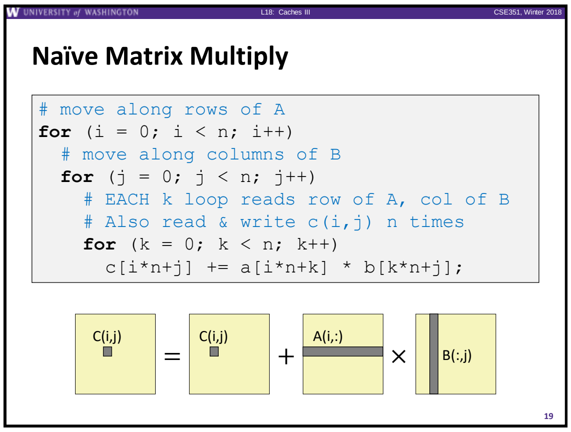
$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$



Matrices in Memory

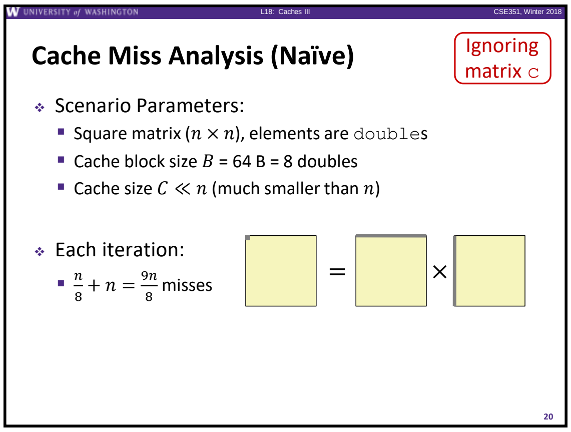
- How do cache blocks fit into this scheme?
  - Row major matrix in memory:

COLUMN of matrix (blue) is spread among cache blocks shown in red



Naïve Matrix Multiply

```
# move along rows of A
for (i = 0; i < n; i++)
  # move along columns of B
  for (j = 0; j < n; j++)
    # EACH k loop reads row of A, col of B
    # Also read & write c(i,j) n times
    for (k = 0; k < n; k++)
      c[i*n+j] += a[i*n+k] * b[k*n+j];
```

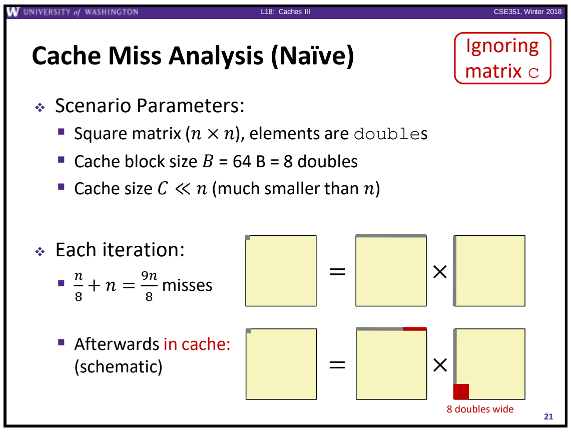


Cache Miss Analysis (Naïve)

Ignoring matrix c

- Scenario Parameters:
  - Square matrix ($n \times n$), elements are `doubles`
  - Cache block size B = 64 B = 8 doubles
  - Cache size $C \ll n$ (much smaller than n)
- Each iteration:
  - $\frac{n}{8} + n = \frac{9n}{8}$ misses

$= \times$



Cache Miss Analysis (Naïve)

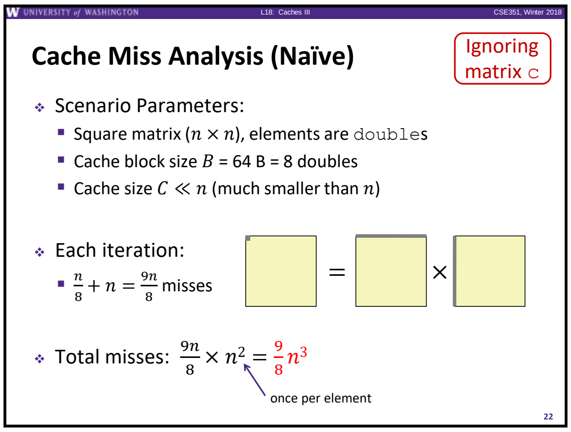
Ignoring matrix c

- Scenario Parameters:
  - Square matrix ($n \times n$), elements are `doubles`
  - Cache block size B = 64 B = 8 doubles
  - Cache size $C \ll n$ (much smaller than n)
- Each iteration:
  - $\frac{n}{8} + n = \frac{9n}{8}$ misses
  - Afterwards in cache: (schematic)

$= \times$

$= \times$

8 doubles wide



Cache Miss Analysis (Naïve)

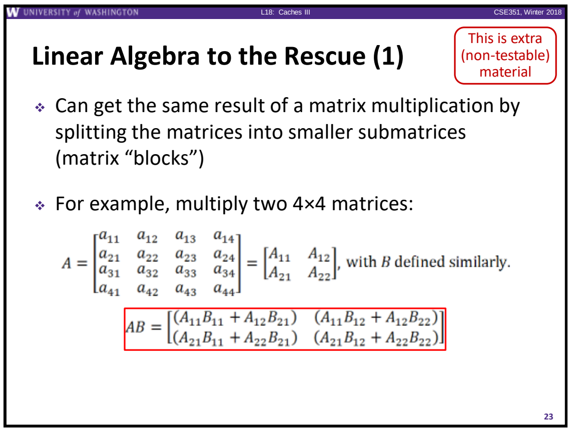
Ignoring matrix c

- Scenario Parameters:
  - Square matrix ($n \times n$), elements are `doubles`
  - Cache block size B = 64 B = 8 doubles
  - Cache size $C \ll n$ (much smaller than n)
- Each iteration:
  - $\frac{n}{8} + n = \frac{9n}{8}$ misses

$= \times$

- Total misses: $\frac{9n}{8} \times n^2 = \frac{9}{8}n^3$

once per element



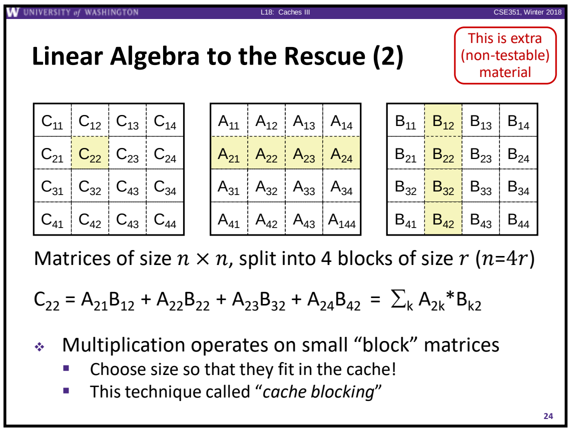
Linear Algebra to the Rescue (1)

This is extra (non-testable) material

- Can get the same result of a matrix multiplication by splitting the matrices into smaller submatrices (matrix "blocks")
- For example, multiply two 4×4 matrices:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \text{ with } B \text{ defined similarly.}$$

$$\boxed{AB = \begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21}) & (A_{11}B_{12} + A_{12}B_{22}) \\ (A_{21}B_{11} + A_{22}B_{21}) & (A_{21}B_{12} + A_{22}B_{22}) \end{bmatrix}}$$



Linear Algebra to the Rescue (2)

This is extra (non-testable) material

C_{11}	C_{12}	C_{13}	C_{14}
C_{21}	C_{22}	C_{23}	C_{24}
C_{31}	C_{32}	C_{43}	C_{34}
C_{41}	C_{42}	C_{43}	C_{44}

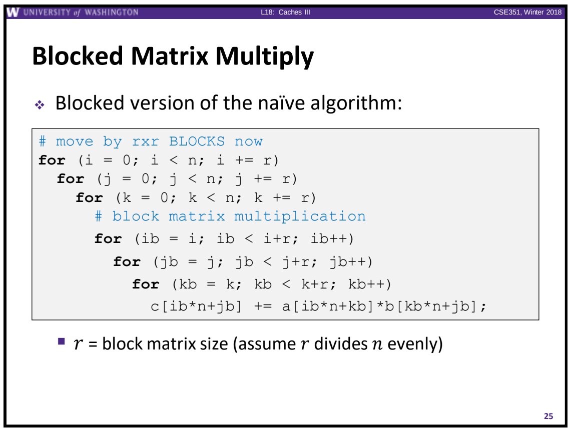
A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}
A_{41}	A_{42}	A_{43}	A_{144}

B_{11}	B_{12}	B_{13}	B_{14}
B_{21}	B_{22}	B_{23}	B_{24}
B_{32}	B_{32}	B_{33}	B_{34}
B_{41}	B_{42}	B_{43}	B_{44}

Matrices of size $n \times n$, split into 4 blocks of size r (n=4r)

$C_{22} = A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32} + A_{24}B_{42} = \sum_k A_{2k}*B_{k2}$

- Multiplication operates on small "block" matrices
  - Choose size so that they fit in the cache!
  - This technique called "*cache blocking*"

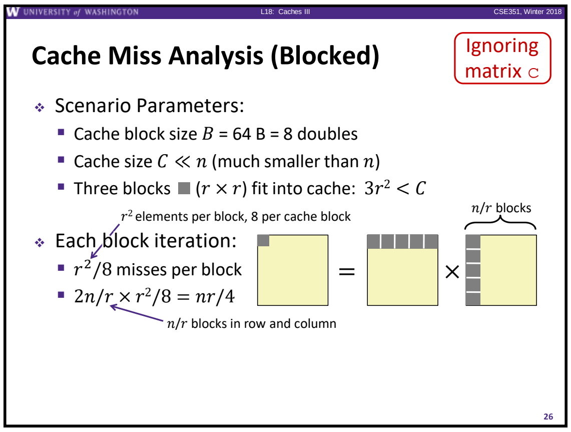


Blocked Matrix Multiply

- Blocked version of the naïve algorithm:

```
# move by rxr BLOCKS now
for (i = 0; i < n; i += r)
  for (j = 0; j < n; j += r)
    for (k = 0; k < n; k += r)
      # block matrix multiplication
      for (ib = i; ib < i+r; ib++)
        for (jb = j; jb < j+r; jb++)
          for (kb = k; kb < k+r; kb++)
            c[ib*n+jb] += a[ib*n+kb]*b[kb*n+jb];
```

  - r = block matrix size (assume r divides n evenly)



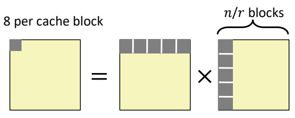
Cache Miss Analysis (Blocked)

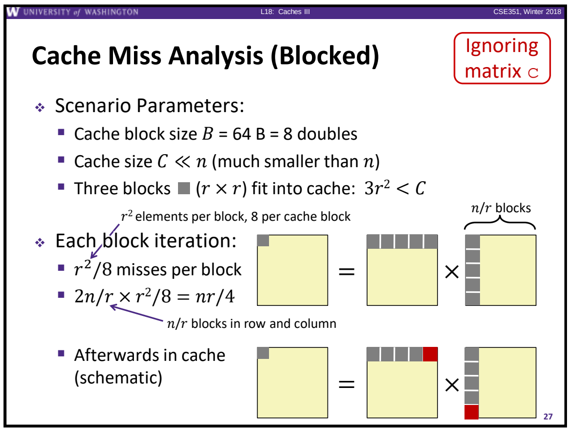
Ignoring matrix c

- Scenario Parameters:
  - Cache block size B = 64 B = 8 doubles
  - Cache size $C \ll n$ (much smaller than n)
  - Three blocks ■ $(r \times r)$ fit into cache: $3r^2 < C$
- Each block iteration:
  - $r^2/8$ misses per block
  - $2n/r \times r^2/8 = nr/4$

r^2 elements per block, 8 per cache block

n/r blocks in row and column





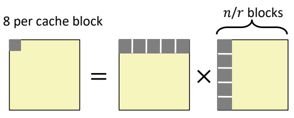
Cache Miss Analysis (Blocked)

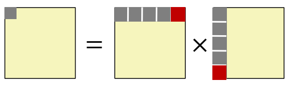
Ignoring matrix c

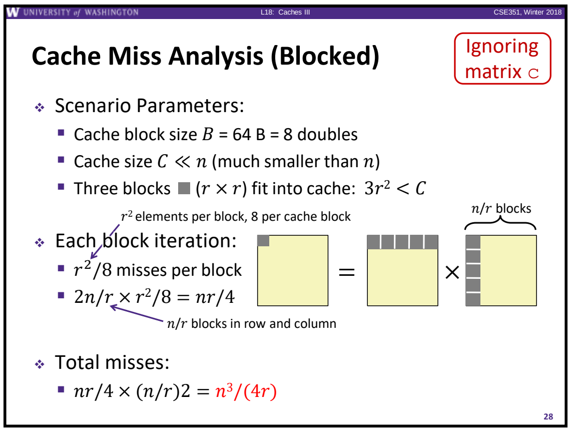
- Scenario Parameters:
  - Cache block size B = 64 B = 8 doubles
  - Cache size $C \ll n$ (much smaller than n)
  - Three blocks ■ $(r \times r)$ fit into cache: $3r^2 < C$
- Each block iteration:
  - $r^2/8$ misses per block
  - $2n/r \times r^2/8 = nr/4$
  - Afterwards in cache (schematic)

r^2 elements per block, 8 per cache block

n/r blocks in row and column







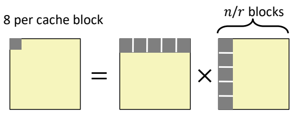
Cache Miss Analysis (Blocked)

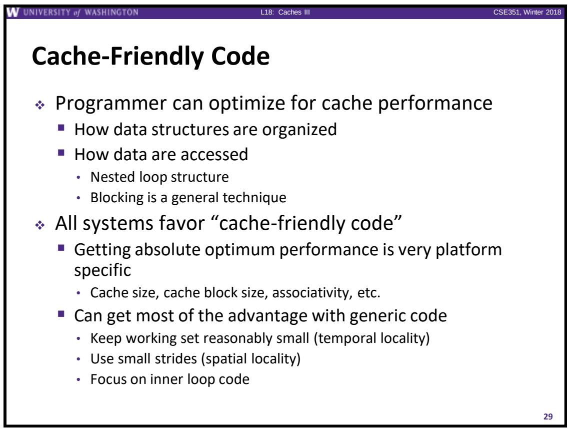
Ignoring matrix c

- Scenario Parameters:
  - Cache block size B = 64 B = 8 doubles
  - Cache size $C \ll n$ (much smaller than n)
  - Three blocks ■ $(r \times r)$ fit into cache: $3r^2 < C$
- Each block iteration:
  - $r^2/8$ misses per block
  - $2n/r \times r^2/8 = nr/4$
- Total misses:
  - $nr/4 \times (n/r)2 = n^3/(4r)$

r^2 elements per block, 8 per cache block

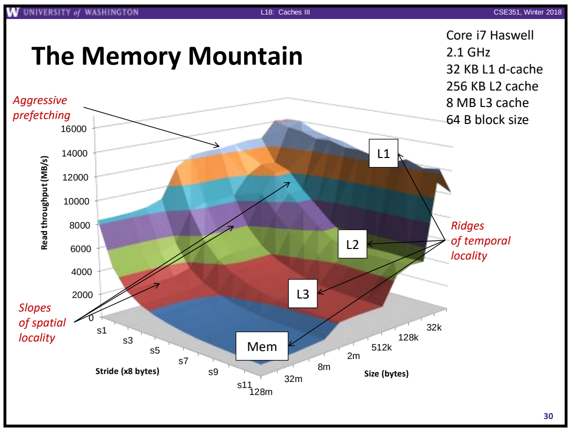
n/r blocks in row and column





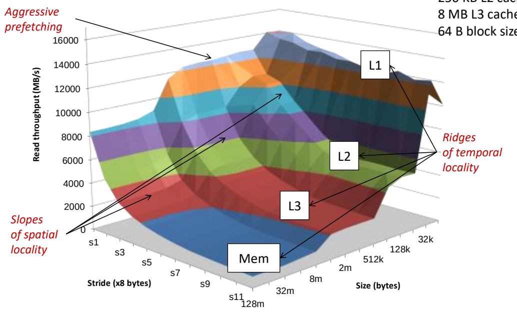
Cache-Friendly Code

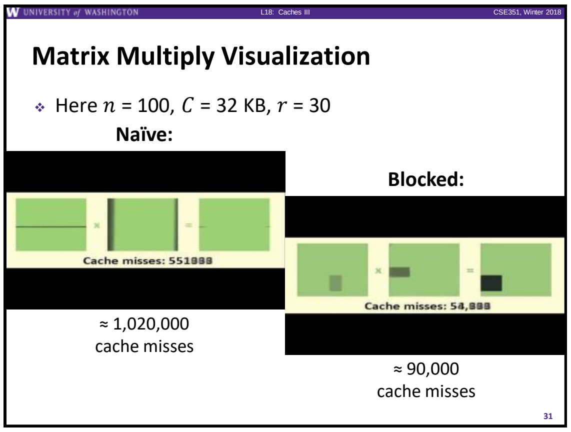
- Programmer can optimize for cache performance
  - How data structures are organized
  - How data are accessed
    - Nested loop structure
    - Blocking is a general technique
- All systems favor "cache-friendly code"
  - Getting absolute optimum performance is very platform specific
    - Cache size, cache block size, associativity, etc.
  - Can get most of the advantage with generic code
    - Keep working set reasonably small (temporal locality)
    - Use small strides (spatial locality)
    - Focus on inner loop code



The Memory Mountain

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



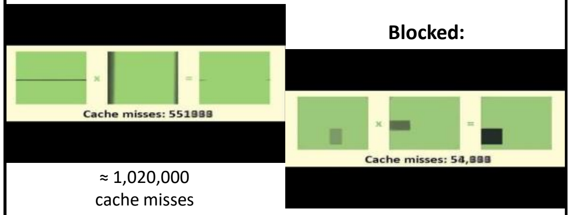


Matrix Multiply Visualization

- Here n = 100, C = 32 KB, r = 30

Naïve:

Blocked:



≈ 1,020,000
cache misses

≈ 90,000
cache misses

Anatomy of a Cache Question

- Cache questions come in a few flavors:
  1) TIO Address Breakdown
  2) For fixed cache parameters, analyze the performance of the given code/sequence
  3) For given code/sequence, how does changing your cache parameters affect performance?
  4) Average Memory Access Time (AMAT)

Example Cache Parameters Problem

- 1 MB address space, 125 cycles to go to memory. Fill in the following table:

Cache Size	4 KB
Block Size	16 B
Associativity	4-way
Hit Time	3 cycles
Miss Rate	20%
Write Policy	Write-through
Replacement Policy	LRU
Tag Bits	10
Index Bits	6
Offset Bits	4
AMAT	AMAT = 3 + 0.2 * 125 = 28

Peer Instruction Question

- We have a cache of size 2 KB with block size of 128 B. If our cache has 2 sets, what is its associativity?

  A. **2**
  B. **4**
  C. **8**
  D. **16**
  E. **We're lost…**

- If addresses are 16 bits wide, how wide is the Tag field?

Peer Instruction Question

- Which of the following cache statements is FALSE?
  A. **We can reduce compulsory misses by decreasing our block size**
  B. **We can reduce conflict misses by increasing associativity**
  C. **A write-back cache will save time for code with good temporal locality on writes**
  D. **A write-through cache will always match data with the memory hierarchy level below it**
  E. **We're lost…**

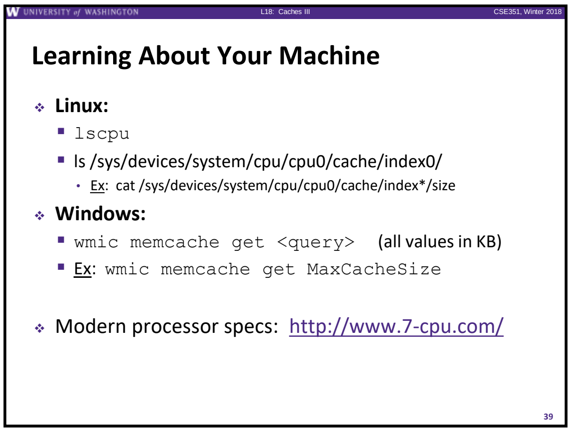
Example Code Analysis Problem

- Assuming the cache starts <u>cold</u> (all blocks invalid), calculate the **miss rate** for the following loop:
  - m = 20 bits, C = 4 KB, B = 16 B, E = 4

```
#define AR_SIZE 2048
int int_ar[AR_SIZE], sum=0;      // &int_ar=0x80000
for (int i=0; i<AR_SIZE; i++)
   sum += int_ar[i];
for (int j=AR_SIZE-1; j>=0; j--)
   sum += int_ar[i];
```

Suggested Problems

- CS:APP 3rd
  - Practice Problems 6.12-15
- AU16 Final Question F5



Learning About Your Machine

- **Linux:**
  - `lscpu`
  - ls /sys/devices/system/cpu/cpu0/cache/index0/
    - Ex: cat /sys/devices/system/cpu/cpu0/cache/index*/size
- **Windows:**
  - `wmic memcache get <query>` (all values in KB)
  - Ex: `wmic memcache get MaxCacheSize`
- Modern processor specs: http://www.7-cpu.com/