

# Chapter 5

## **Large and Fast: Exploiting Memory Hierarchy**

### **Part 3**

# Engineering Week Watch Parties With

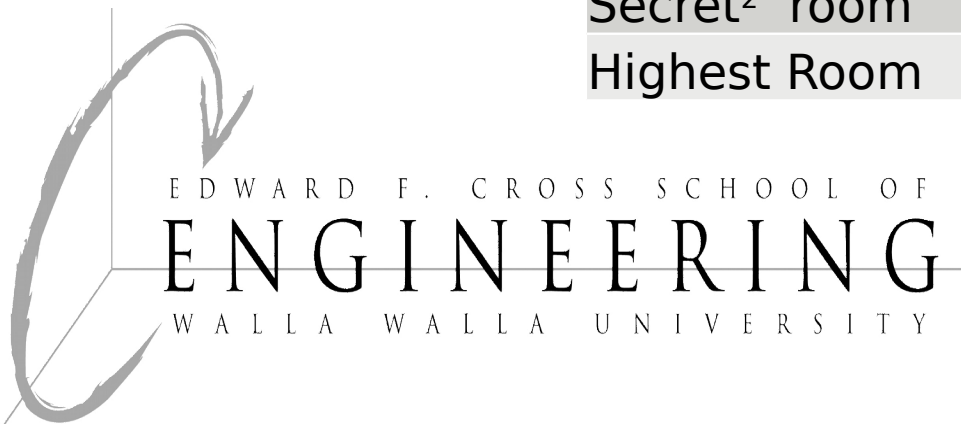
- Friends
- FOOD
- Colloquium Credit
- Networking

7:00 pm via Zoom

Get link to event: [tinyurl.com/2p8jcpcj](https://tinyurl.com/2p8jcpcj)

- Watch your email for more information

Type	Room #	#	Notes
Laptop Room	Fishbowl	20	Bring your laptop
The "Big" Room	KRH 205	20	
Awards Room	KRH 210	20	Dr. Roth broadcasting live
Secret room	CSP 163	12	
Secret <sup>2</sup> room	CSP 165	12	
Highest Room	KRH 326	12	



## Goals today:

- Cache performance measurement
- Reduce cache misses via flexible block placement
  - Direct mapped placement
  - Set associative placement
  - Fully associative placement
- Where to place a block in the cache
- Replacement policy
- Multi-level caches to reduce miss penalty
- Software optimization using blocking

# Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

$$\begin{aligned} & \text{Memory stall cycles} \\ &= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty} \end{aligned}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

# Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache:  $0.02 \times 100 = 2$
  - D-cache:  $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI =  $2 + 2 + 1.44 = 5.44$ 
  - Ideal CPU is  $5.44/2 = 2.72$  times faster

# Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%
  - $AMAT = 1 + 0.05 \times 20 = 2\text{ns}$ 
    - 2 cycles per instruction

# Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance

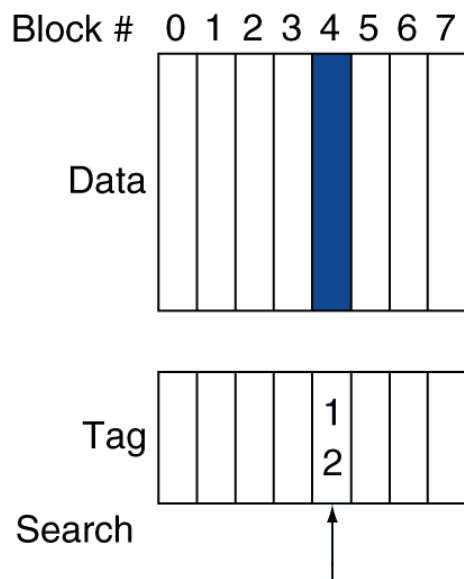
# Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

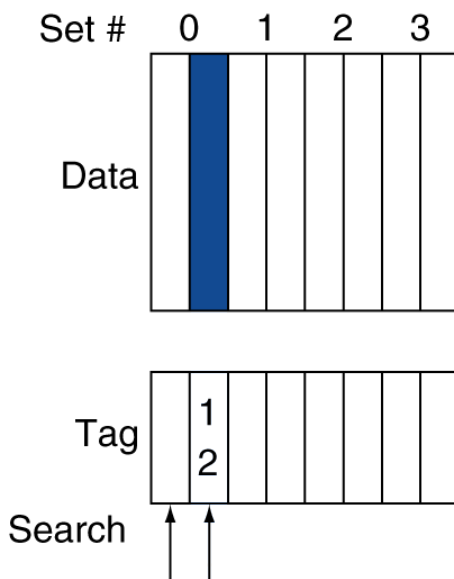


# Associative Cache Example

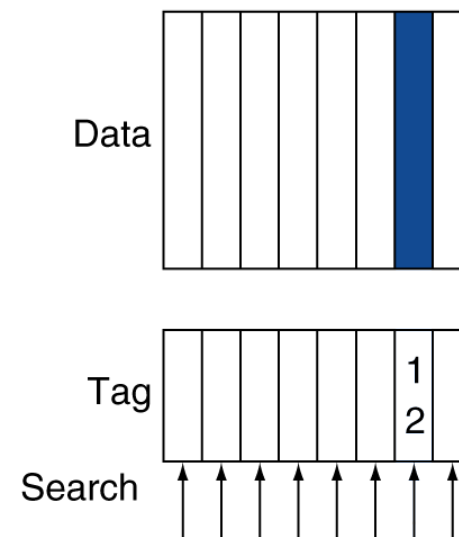
**Direct mapped**



**Set associative**



**Fully associative**



# Spectrum of Associativity

- For a cache with 8 entries

**One-way set associative  
(direct mapped)**

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

**Two-way set associative**

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

**Four-way set associative**

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

**Eight-way set associative (fully associative)**

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

# Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	

# Associativity Example

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

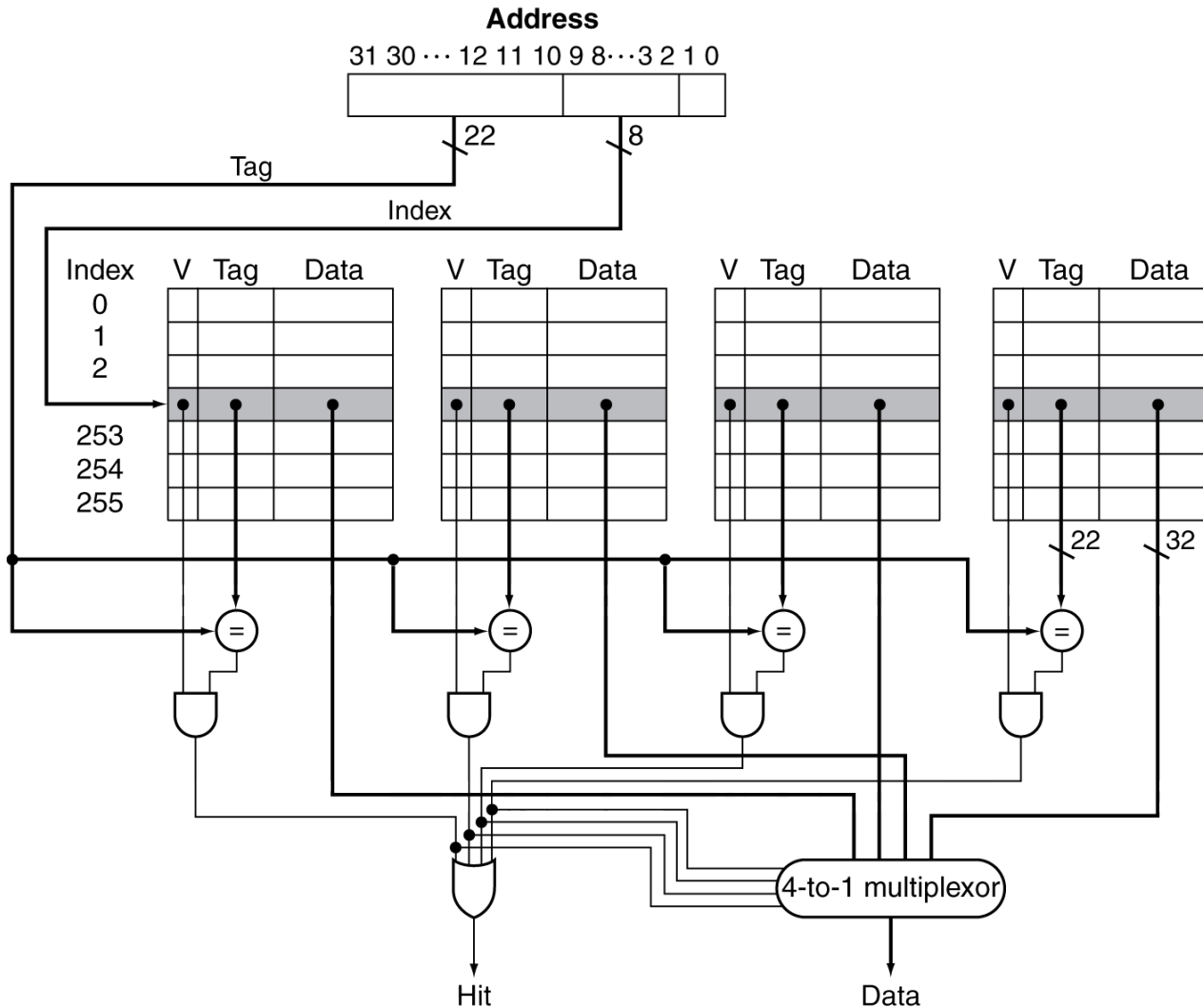
- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[8]	Mem[6]	

# How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

# Set Associative Cache Organization



# Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

# Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache



# Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty =  $100\text{ns}/0.25\text{ns} = 400$  cycles
  - Effective CPI =  $1 + 0.02 \times 400 = 9$

# Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty =  $5\text{ns}/0.25\text{ns} = 20$  cycles
- Primary miss with L-2 miss
  - Extra penalty = 500 cycles
- $\text{CPI} = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance ratio =  $9/3.4 = 2.6$

# Multilevel Cache Considerations

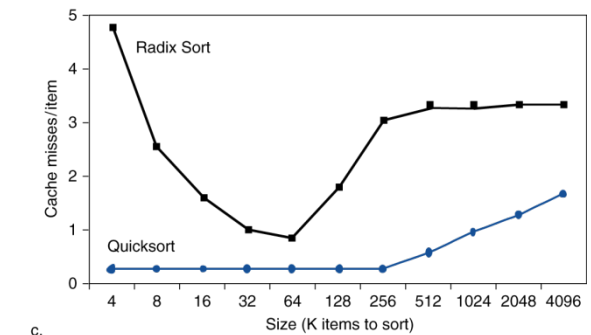
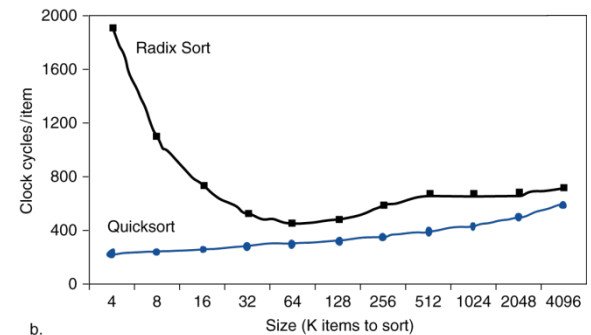
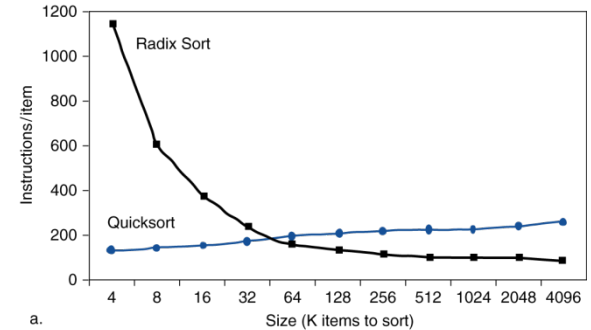
- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

# Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

# Interactions with Software

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



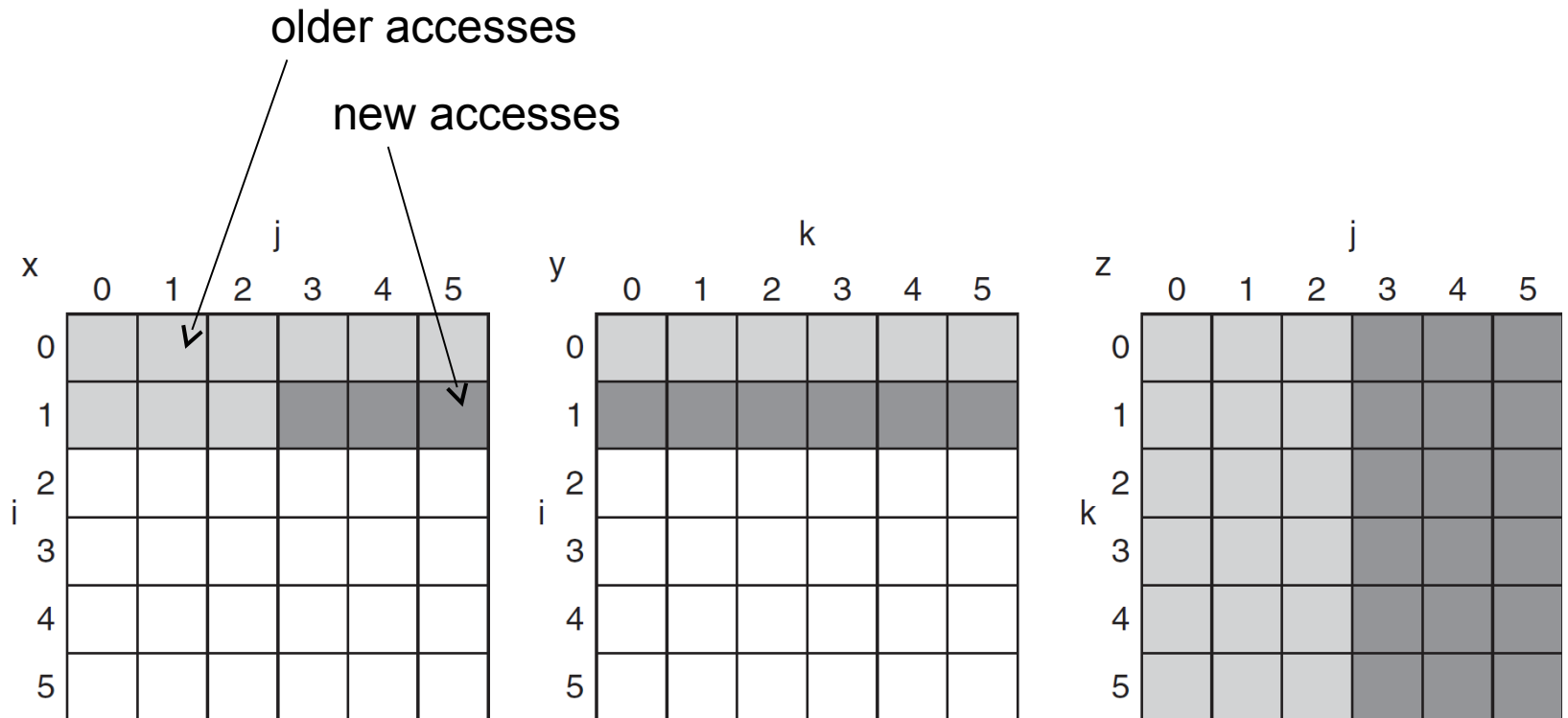
# Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

```
for (int j = 0; j < n; ++j)
{
    double cij = C[i+j*n];
    for( int k = 0; k < n; k++ )
        cij += A[i+k*n] * B[k+j*n];
    C[i+j*n] = cij;
}
```

# DGEMM Access Pattern

- C, A, and B arrays

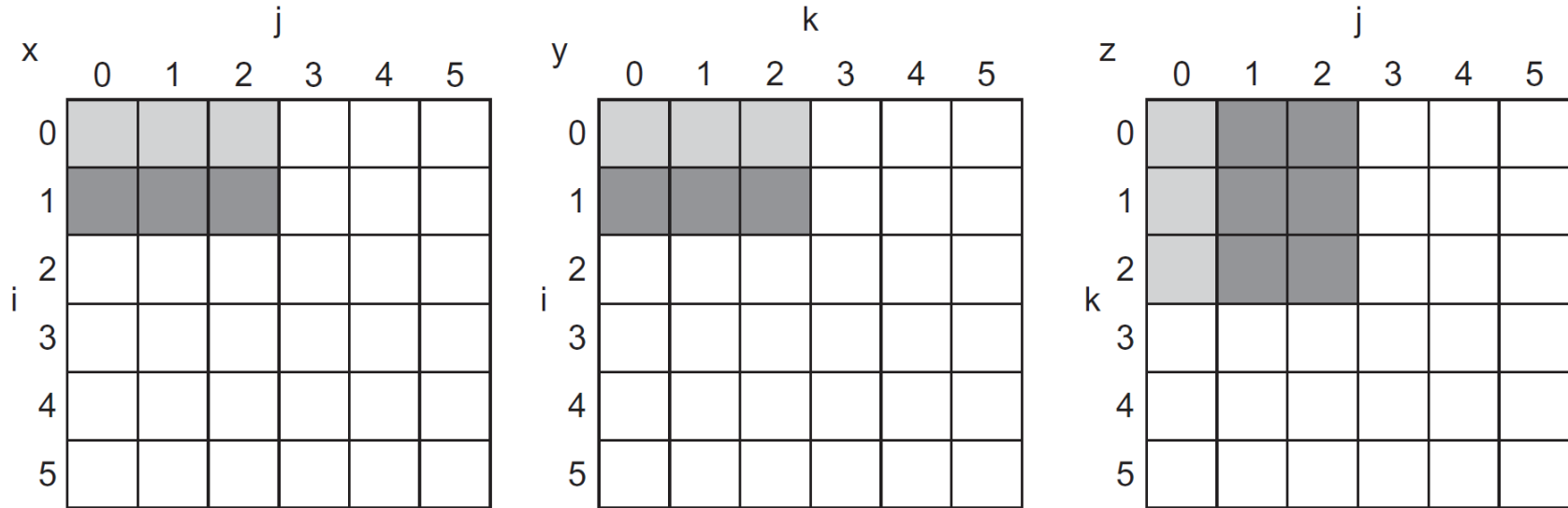


# Cache Blocked DGEMM

```
1 #define BLOCKSIZE 32
2 void do_block (int n, int si, int sj, int sk, double *A, double
3 *B, double *C)
4 {
5   for (int i = si; i < si+BLOCKSIZE; ++i)
6     for (int j = sj; j < sj+BLOCKSIZE; ++j)
7       {
8         double cij = C[i+j*n]; /* cij = C[i][j] */
9         for( int k = sk; k < sk+BLOCKSIZE; k++ )
10          cij += A[i+k*n] * B[k+j*n]; /* cij+=A[i][k]*B[k][j] */
11        C[i+j*n] = cij; /* C[i][j] = cij */
12      }
13 }
14 void dgemm (int n, double* A, double* B, double* C)
15 {
16   for ( int sj = 0; sj < n; sj += BLOCKSIZE )
17     for ( int si = 0; si < n; si += BLOCKSIZE )
18       for ( int sk = 0; sk < n; sk += BLOCKSIZE )
19         do_block(n, si, sj, sk, A, B, C);
20 }
```



# Blocked DGEMM Access Pattern



■ 32x32 ■ 160x160 ■ 480x480 ■ 960x960

