

How to Write Fast Numerical Code

Spring 2014

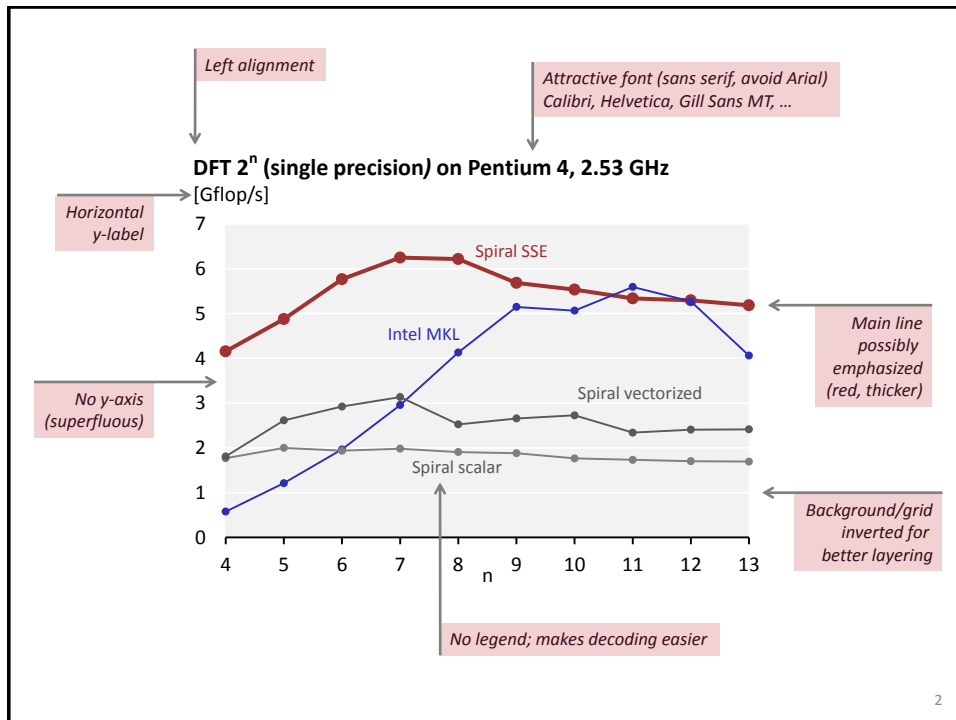
Lecture: Memory hierarchy, locality, caches

Instructor: Markus Püschel

TA: Daniele Spampinato & Alen Stojanov



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Organization

- Temporal and spatial locality
- Memory hierarchy
- Caches

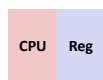
Chapter 5 in **Computer Systems: A Programmer's Perspective**, 2nd edition,
Randal E. Bryant and David R. O'Hallaron, Addison Wesley 2010

Part of these slides are adapted from the course associated with this book

3

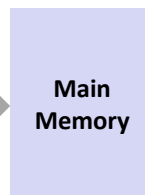
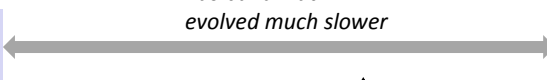
Problem: Processor-Memory Bottleneck

*Processor performance
doubled about
every 18 months*



Core 2 Duo:
Peak performance:
2 SSE two operand ops/cycles
consumes up to 64 Bytes/cycle

*Bus bandwidth
evolved much slower*

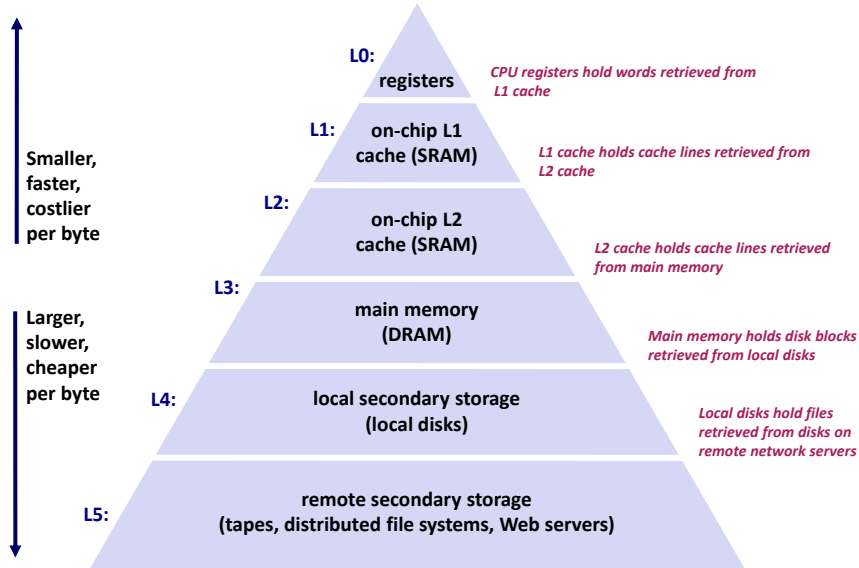


Core 2 Duo:
Bandwidth
2 Bytes/cycle

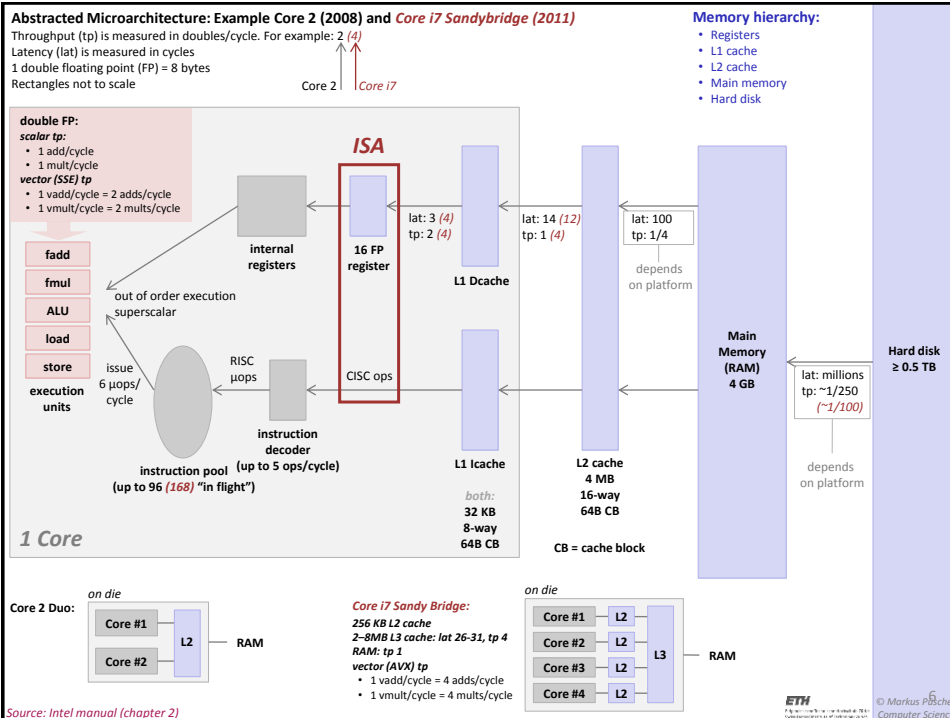
Solution: Caches/Memory hierarchy

4

Typical Memory Hierarchy



5



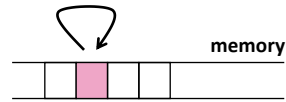
Why Caches Work: Locality

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

History of locality

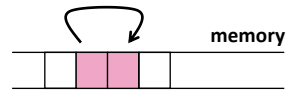
- **Temporal locality:**

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



7

Example: Locality?

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

- **Data:**
 - Temporal: **sum** referenced in each iteration
 - Spatial: array **a[]** accessed in stride-1 pattern
- **Instructions:**
 - Temporal: loops cycle through the same instructions
 - Spatial: instructions referenced in sequence
- **Being able to assess the locality of code is a crucial skill for a performance programmer**

8

Locality Example #1

```
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;
}
```

9

Locality Example #2

```
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;
}
```

10

Locality Example #3

```
int sum_array_3d(int a[M][N][K])
{
    int i, j, k, sum = 0;

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

How to improve locality?

11

Operational Intensity Again

- **Definition:** Given a program P, assume cold (empty) cache

$$\text{Operational intensity: } I(n) = \frac{W(n)}{Q(n)}$$

#flops (input size n) ←

#bytes transferred cache ↔ memory (for input size n) ←

- **Examples: Determine asymptotic bounds on $I(n)$**
 - Vector sum: $y = x + y$ $O(1)$
 - Matrix-vector product: $y = Ax$ $O(1)$
 - Fast Fourier transform $O(\log(n))$
 - Matrix-matrix product: $C = AB + C$ $O(n)$

12

Compute/Memory Bound

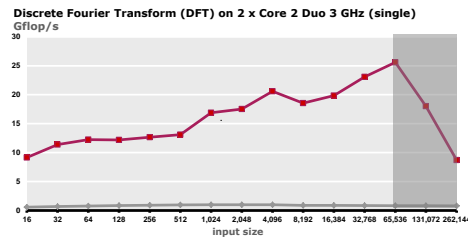
- A function/piece of code is:
 - **Compute bound** if it has high operational intensity
 - **Memory bound** if it has low operational intensity

- Relationship between operational intensity and locality?
 - Operational intensity \sim locality

13

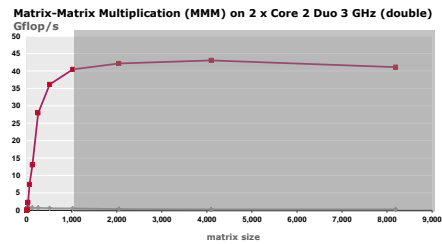
Effects

FFT: $I(n) \leq O(\log(n))$



Up to 40-50% peak
Performance drop outside last level cache (LLC)
Most time spent transferring data

MMM: $I(n) \leq O(n)$



Up to 80-90% peak
Performance can be maintained outside LLC
Cache miss time compensated/hidden by computation

14

Cache

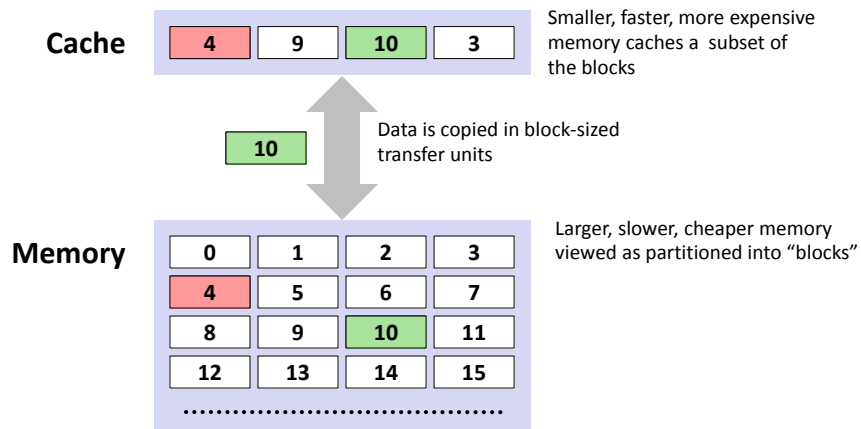
- **Definition:** Computer memory with short access time used for the storage of frequently or recently used instructions or data



- Naturally supports **temporal locality**
- **Spatial locality** is supported by transferring data in blocks
 - Core 2: one block = 64 B = 8 doubles

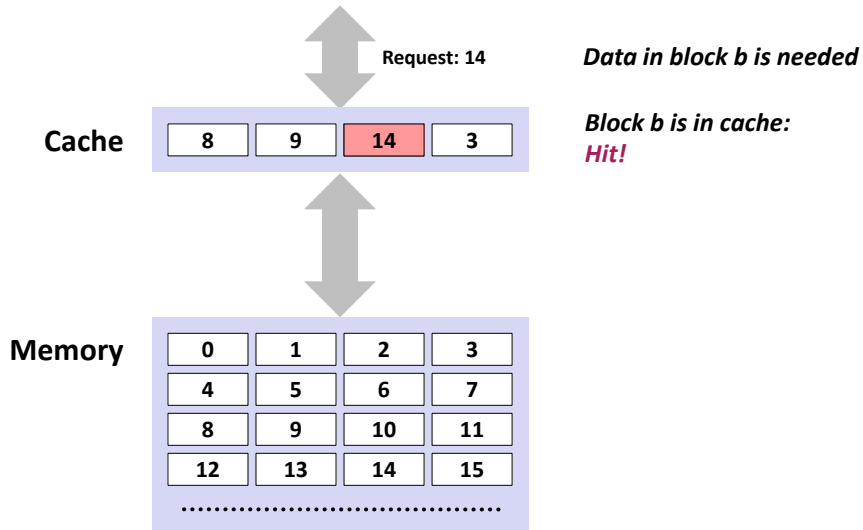
15

General Cache Mechanics



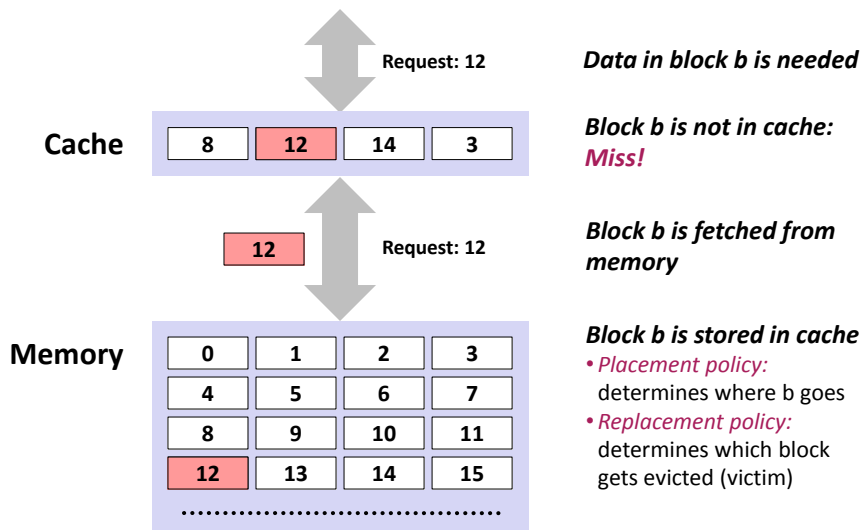
16

General Cache Concepts: Hit



17

General Cache Concepts: Miss



18

Types of Cache Misses (The 3 C's)

- **Compulsory (cold) miss**
Occurs on first access to a block
- **Capacity miss**
Occurs when working set is larger than the cache
- **Conflict miss**
Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot

- **Not a clean classification but still useful**

19

Cache Performance Metrics

- **Miss Rate**
 - Fraction of memory references not found in cache: misses / accesses
= $1 - \text{hit rate}$
- **Hit Time**
 - Time to deliver a block in the cache to the processor
 - Core 2:
3 clock cycles for L1
14 clock cycles for L2
- **Miss Penalty**
 - Additional time required because of a miss
 - Core 2: about 100 cycles for L2 miss

20

Cache Structure

- Draw a direct mapped cache (E = 1, B = 4 doubles, S = 8)
- Show how blocks are mapped into cache

21

Example (S=8, E=1)

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

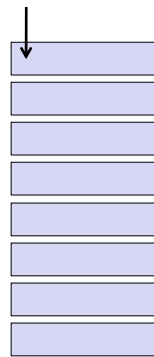
    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

Ignore the variables sum, i, j

assume: cold (empty) cache,
a[0][0] goes here



B = 32 byte = 4 doubles

blackboard

22

Cache Structure

- Add associativity (E = 2, B = 4 doubles, S = 8)
- Show how elements are mapped into cache

23

Example (S=4, E=2)

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

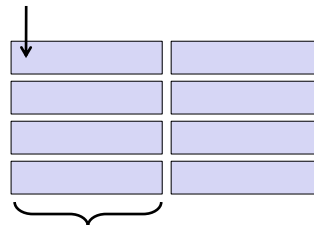
    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

Ignore the variables sum, i, j

assume: cold (empty) cache,
a[0][0] goes here

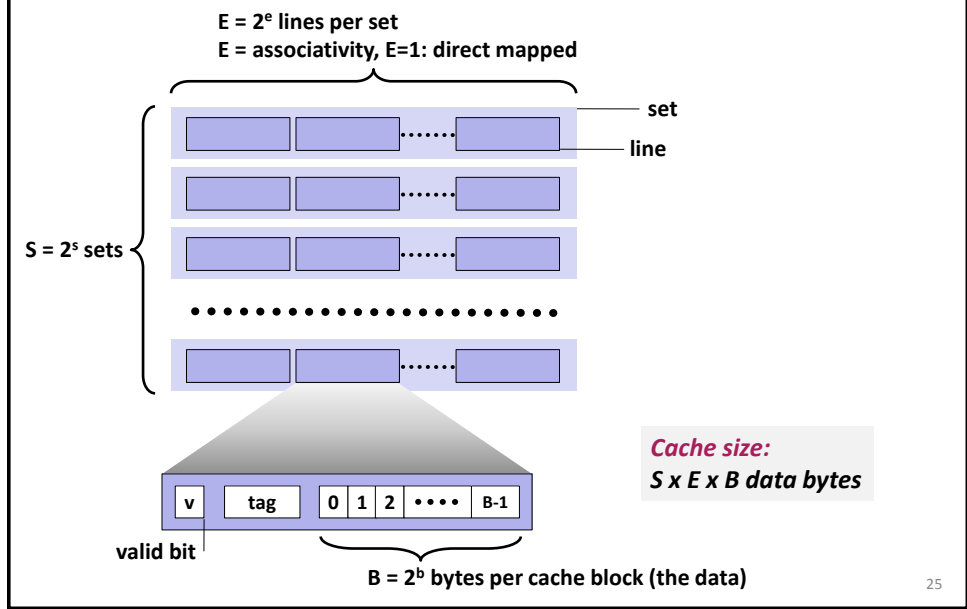


B = 32 byte = 4 doubles

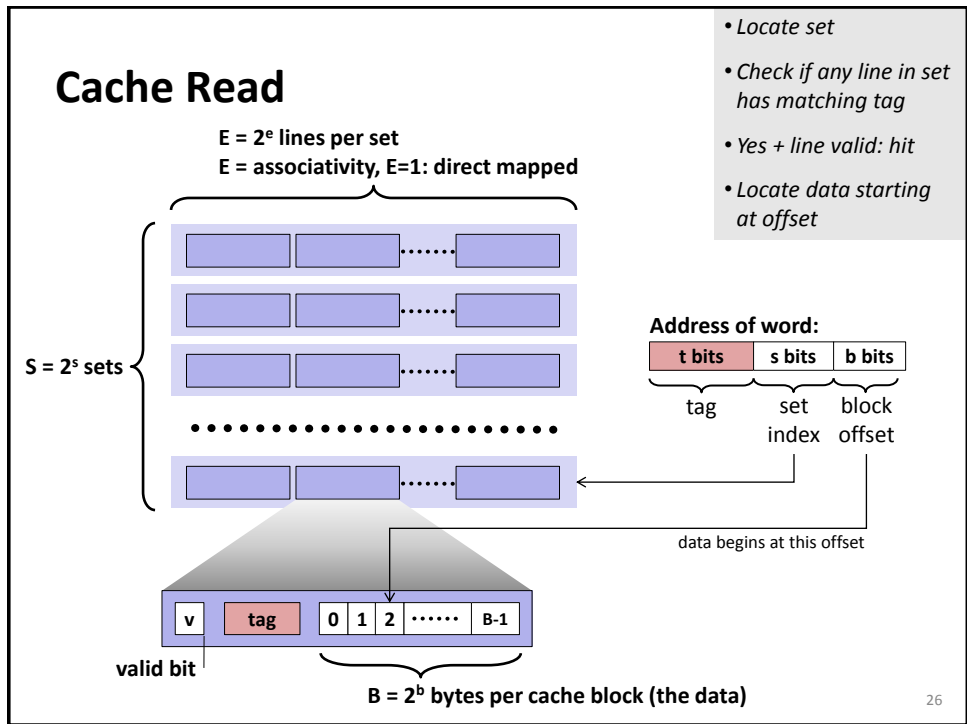
blackboard

24

General Cache Organization (S, E, B)

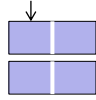


Cache Read



Small Example, Part 1

x[0]



Cache:

E = 1 (direct mapped)

S = 2

B = 16 (2 doubles)

Array (accessed twice in example)

x = x[0], ..., x[7]

```
% Matlab style code
for j = 0:1
  for i = 0:7
    access(x[i])
```

Access pattern:

0123456701234567

Hit/Miss:

MHMHMHMHMHMHMHMH

Result: 8 misses, 8 hits

Spatial locality: yes

Temporal locality: no

27

Small Example, Part 2

x[0]



Cache:

E = 1 (direct mapped)

S = 2

B = 16 (2 doubles)

Array (accessed twice in example)

x = x[0], ..., x[7]

```
% Matlab style code
for j = 0:1
  for i = 0:2:7
    access(x[i])
  for i = 1:2:7
    access(x[i])
```

Access pattern:

0246135702461357

Hit/Miss:

MMMMMMMMMMMMMMMM

Result: 16 misses

Spatial locality: no

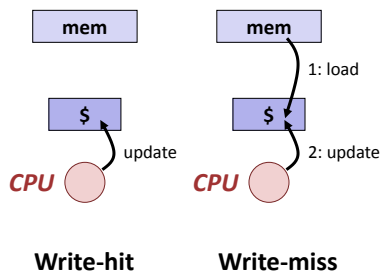
Temporal locality: no

28

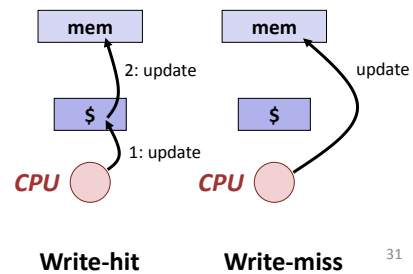
What about writes?

- What to do on a write-hit?
 - *Write-through*: write immediately to memory
 - *Write-back*: defer write to memory until replacement of line
- What to do on a write-miss?
 - *Write-allocate*: load into cache, update line in cache
 - *No-write-allocate*: writes immediately to memory

Write-back/write-allocate (Core)



Write-through/no-write-allocate



31

Example: (Blackboard)

- $z = x + y$, x, y, z vector of length n
- assume they fit jointly in cache + cold cache
- memory traffic $Q(n)$?
- operational intensity $I(n)$?

32

Locality Optimization: Blocking

- Example: MMM (blackboard)

33

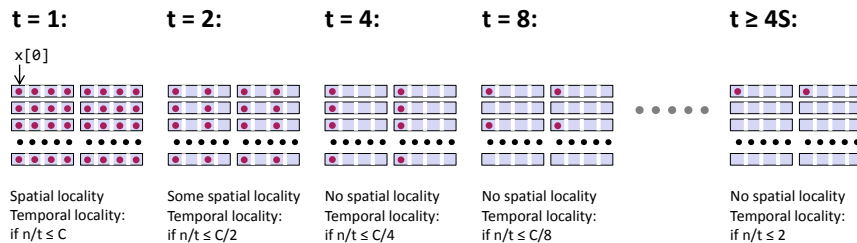
The Killer: Two-Power Strided Working Sets

```

% t = 1,2,4,8,... a 2-power
% size of working set: n/t
for (i = 0; i < n; i += t)
  access(x[i])
for (i = 0; i < n; i += t)
  access(x[i])
    
```

blackboard

Cache: E = 2, B = 4 doubles



34

The Killer: Where Can It Occur?

- **Accessing two-power size 2D arrays (e.g., images) columnwise**
 - 2d Transforms
 - Stencil computations
 - Correlations
- **Various transform algorithms**
 - Fast Fourier transform
 - Wavelet transforms
 - Filter banks

35

Summary

- It is important to assess temporal and spatial locality in the code
- Cache structure is determined by three parameters
- You should be able to roughly simulate a computation on paper
- Blocking to improve locality
- Two-power strides are problematic (conflict misses)

36