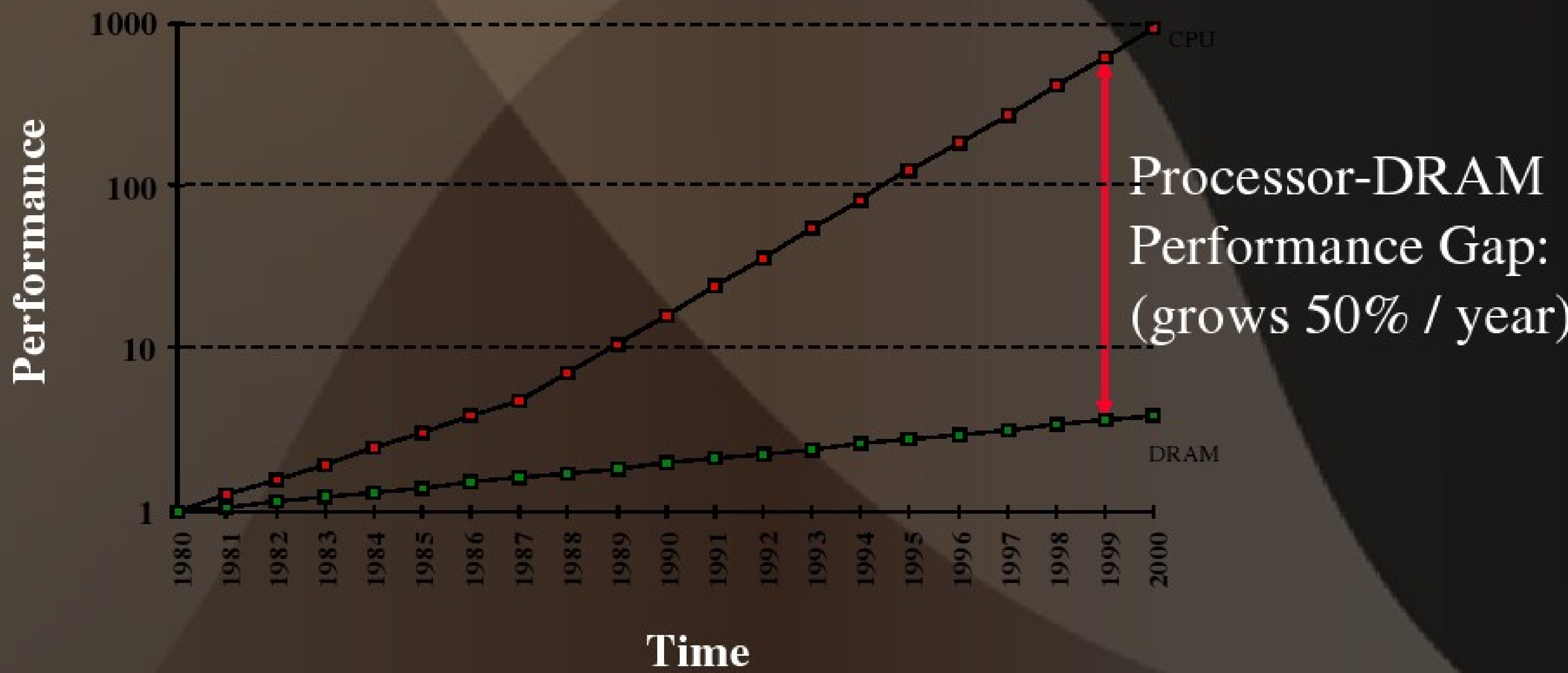
Lecture #35

An Overview of Cache Aware Compiler Optimization

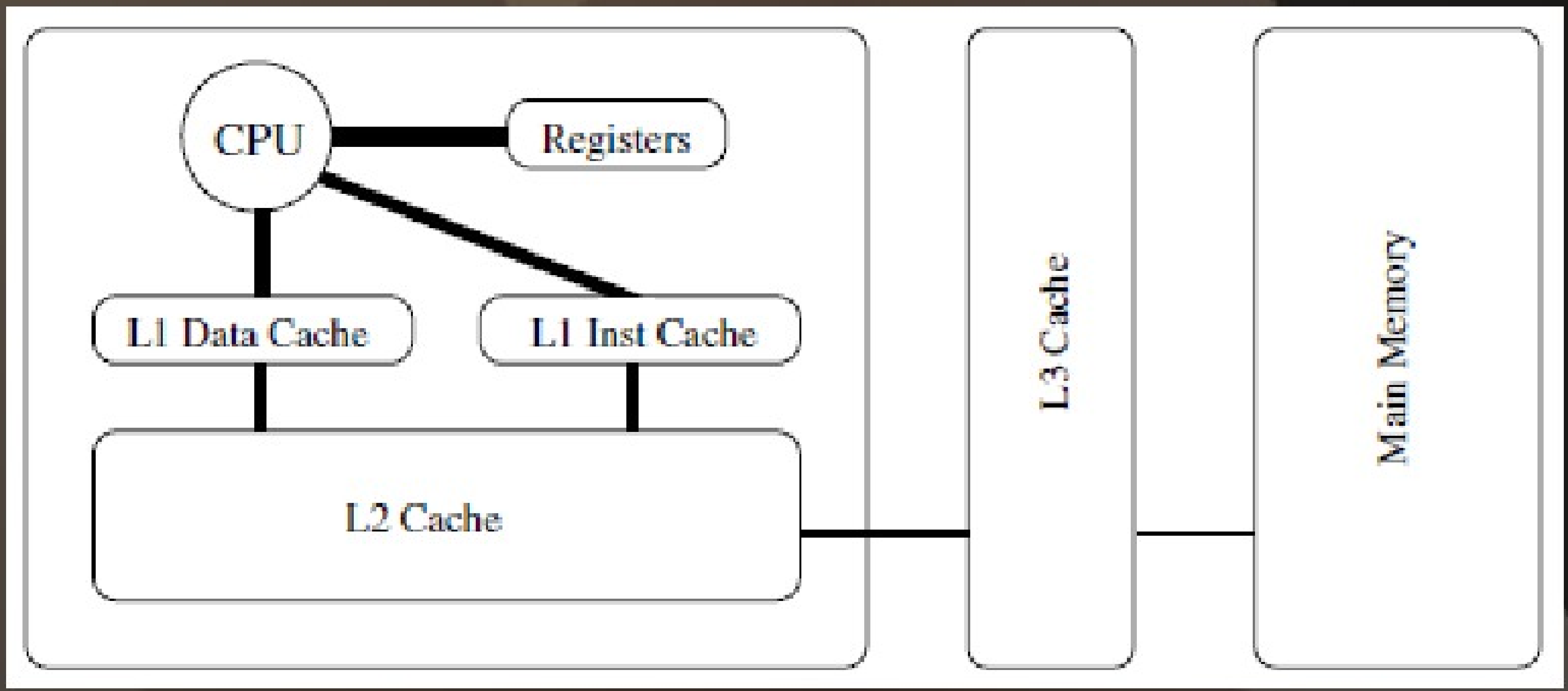
Memory Hierarchy

- **Hierarchical memory Structures:**
  - **An attempt to mitigate the growing gap between CPU speed and memory performance**
    - **Low main memory bandwidth and high latency**



Memory Hierarchy

• Registers	1 cycle	256-8000 bytes
• Cache (on-chip)	3-10 cycles	256k-1M
• Cache (off-chip)	10-20 cycles	1M – 16M
• Main memory	20-100 cycles	32M-4G
• Disk	0.5-5M cycles	4G-1T



Locality of References

- Temporal Locality
  - The tendency of recently accessed data to be accessed again in the near future
- Spatial Locality
  - The tendency of data located close together in address space to be referenced close together in time

Aspects of Cache Architectures

- Cache Line
  - Holds the contents of a contiguous block of main memory
- Cache hits / misses
  - Data requested by processor is found / not found in cache line

- Memory block placement strategies:
  - Direct mapping
    - A memory block may be Placed in exactly one cache line
  - a-way set-associative mapping
    - Cache lines are grouped into sets of size a
    - Placement anywhere in corresponding set
  - Fully-associative mapping
    - A memory block can be placed into any cache line

Aspects of Cache Architectures

- Replacement strategies
  - Random
    - Chooses a random cache line for replacement
  - Least Recently Used (LRU)
    - Chooses block which has not been accessed for the longest time interval
  - Least Frequently Used (LFU)
  - First-In First-Out (FIFO)
- Measuring Cache Behaviour: Profiling
  - Hardware performance counters
    - Cache hits / misses, pipeline stalls, processor cycles, instruction issues, branch mis-predictions, etc.
    - Instrumentation
      - Insert calls to a monitoring library into the program to gather information – gprof does this

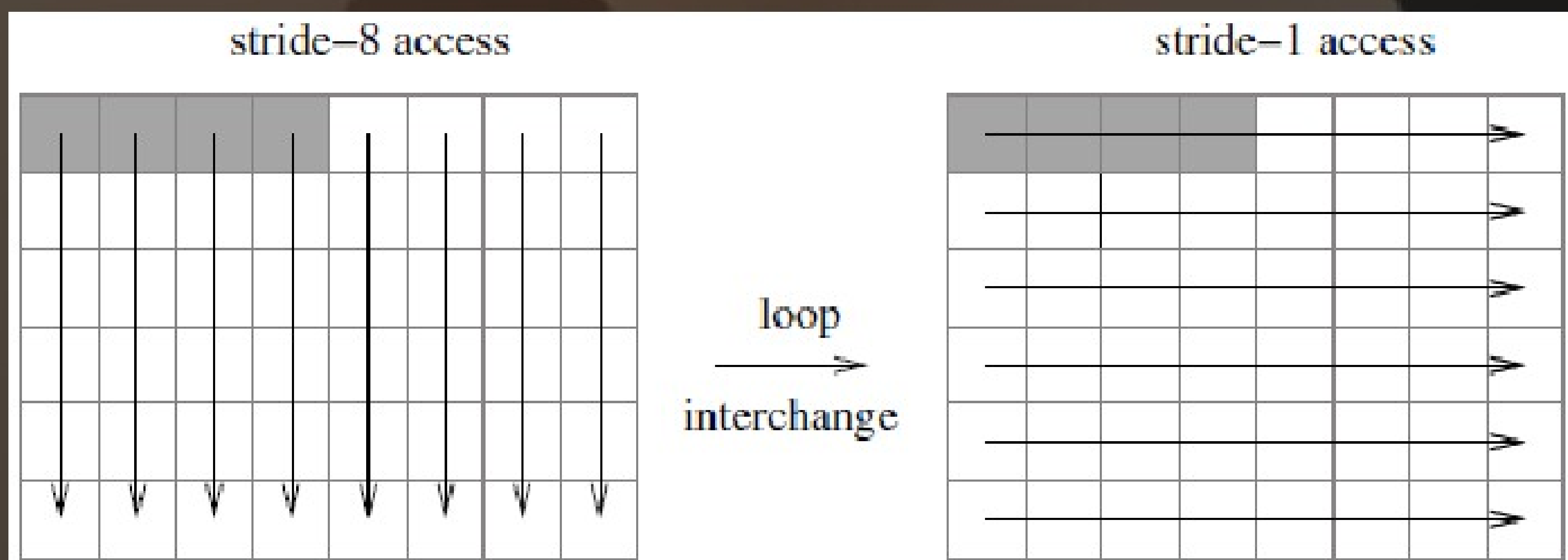
Optimizations

- Loop Interchange

Algorithm 10.3.1 Loop interchange

```
double sum;
double a[n, n];
// Original loop nest:
for j = 1 to n do
    for i = 1 to n do
        sum+ = a[i, j];
    end for
end for
```

```
double sum;
double a[n, n];
// Interchanged loop nest:
for i = 1 to n do
    for j = 1 to n do
        sum+ = a[i, j];
    end for
end for
```



Optimizations

- Loop Interchange: Example 2

```
DO J = 1, N
   DO I = 1, M
    D(I) = D(I) + B(I,J)
   ENDDO
ENDDO
```

Optimizations

- Loop Fusion

Algorithm 10.3.2 Loop fusion

// Original code:	*// After loop fusion:*
for $i = 1$ **to** n **do**	**for** $i = 1$ **to** n **do**
$b[i] = a[i] + 1.0;$	$b[i] = a[i] + 1.0;$
end for	$c[i] = b[i] * 4.0;$
for $i = 1$ **to** n **do**	**end for**
$c[i] = b[i] * 4.0;$	
end for	

Optimizations

- Loop Blocking (or Loop Tiling)

Algorithm 10.3.3 Loop blocking for matrix transposition

```
// Original code:
for i = 1 to n do
    for j = 1 to n do
        a[i, j] = b[j, i];
    end for
end for
```

```
// Loop blocked code:
for ii = 1 to n by B do
    for jj = 1 to n by B do
        for i = ii to min(ii + B - 1, n) do
            for j = jj to min(jj + B - 1, n) do
                a[i, j] = b[j, i];
            end for
        end for
    end for
end for
```