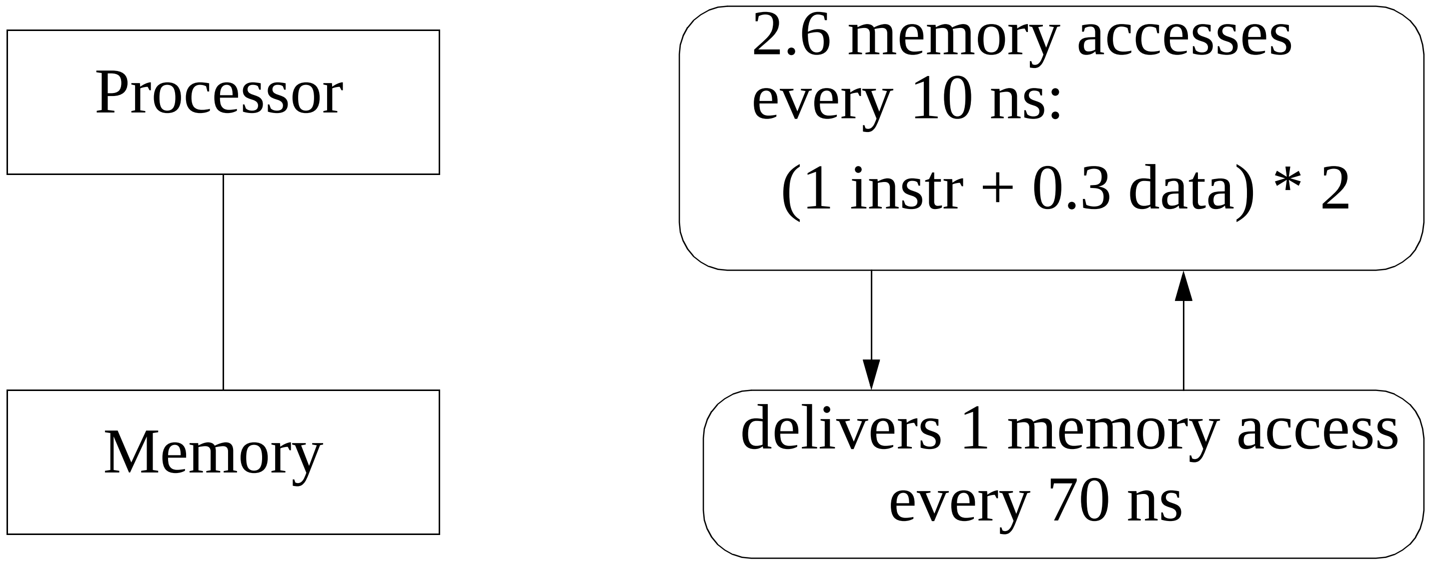
Cache Related Issues in Multiprocessors

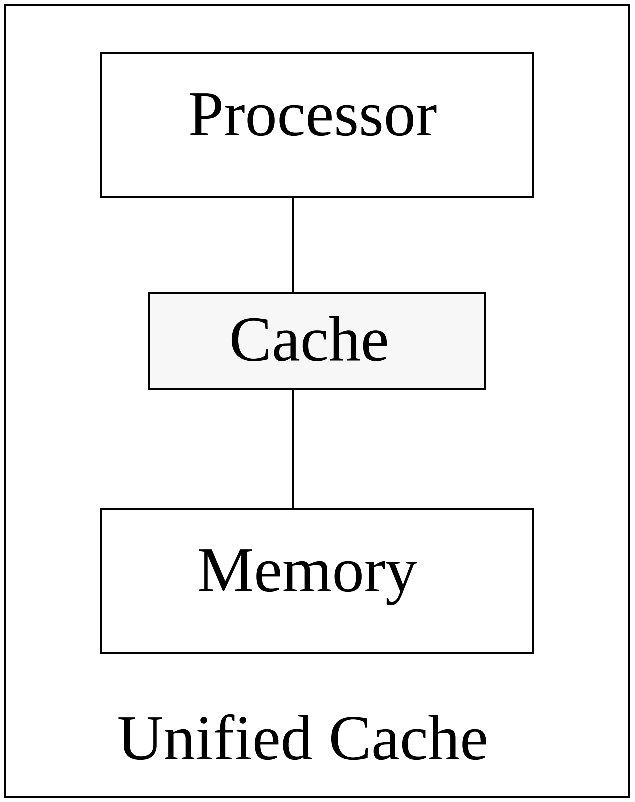
Calin Cascaval
Dept. of Computer Science
Univ. of Illinois at Urbana-Champaign

Why Caches?

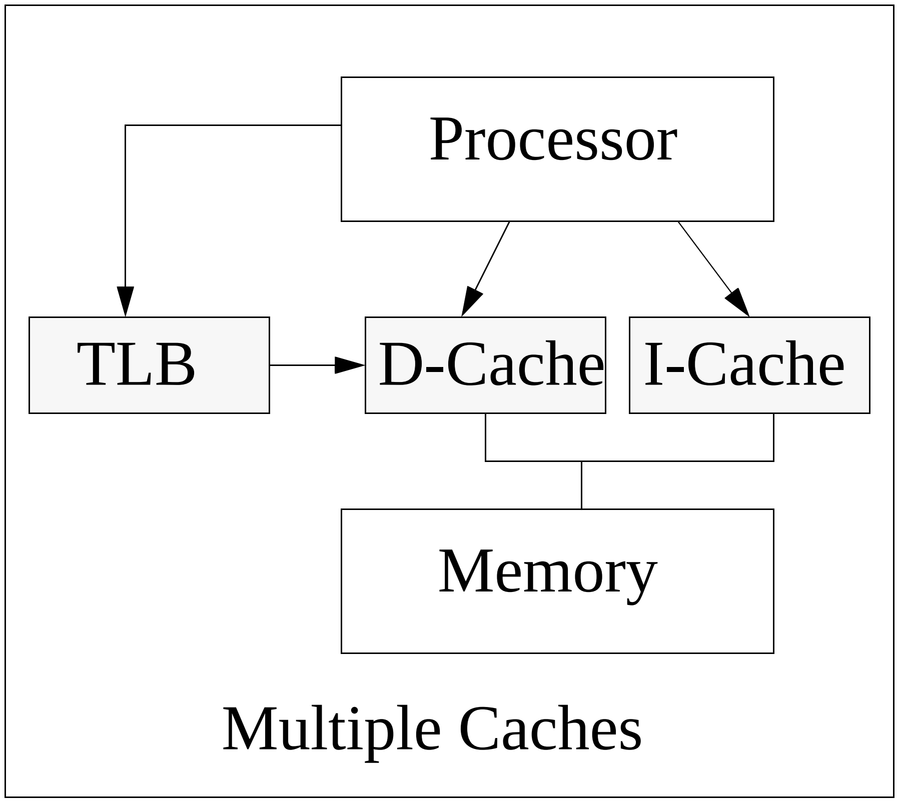

Processor
Memory
2.6 memory accesses every 10 ns:
(1 instr + 0.3 data) * 2
delivers 1 memory access every 70 ns

Key Observation: Programs exhibit locality of reference, i.e., a program accesses a set of locations repeatedly, then move to another set, and so on.

What are Caches?



Unified Cache



Multiple Caches

Fast, expensive memories (SRAMs) that operate at the same speed as the processor.

About Caches

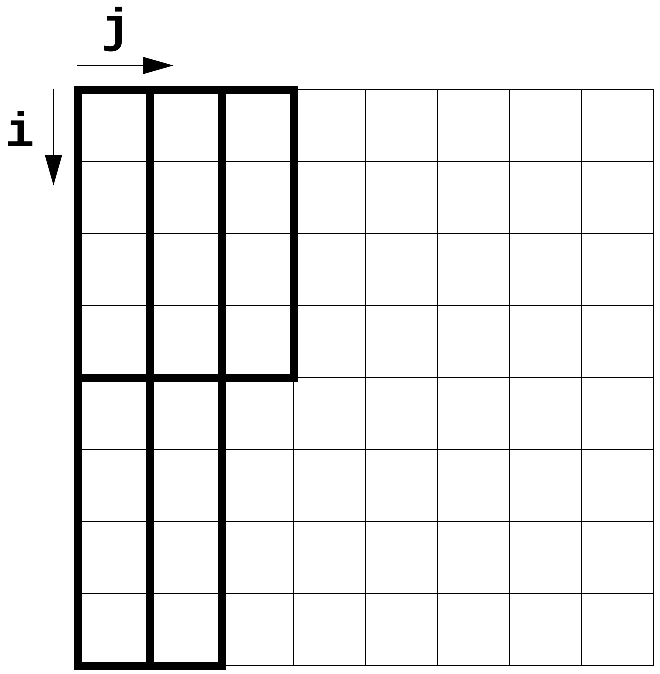
There can be hierarchies of caches, and caches closer to processor are smaller and faster, while caches closer to memory are bigger, slower and also cheaper.

A cache line is the unit of transfer between memory and cache, usually a power of 2. It is based on two observations:

- spatial locality — the location most likely to be accessed next is the neighboring location
- many memory chips can deliver more locations in the same time they deliver one word.

Larger is not necessarily better!

Locality



```
do i = 1, n
  do j = 1, n
    c(i,j) = 0
    do k = 1, n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
```

A better solution:

```
do j = 1, n
  do i = 1, n
    c(i,j) = 0
    do k = 1, n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
```

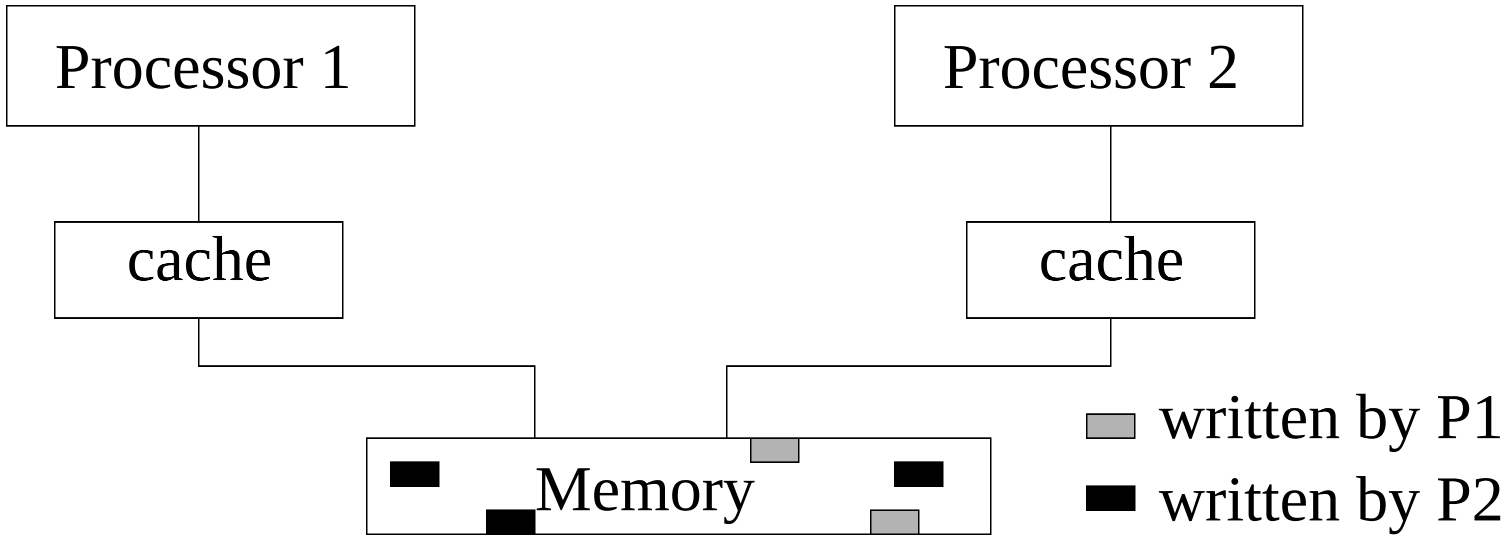
Locality (cont.)

The best solution for matrix multiplication is *tiling*:

```
do jj = 1, N, T
  do ii = 1, N, T
    do kk = 1, N, T
      do j = jj, min(jj+T-1, N)
         do i = ii, min(ii+T-1, N)
           c(i, j) = 0
          do k = kk, min(kk+T-1,N)
            c(i, j) = c(i, j) + a(i, k) * b(k, j)
```

Important: **T** (the tile size) should be chosen such that the blocks fit in the cache.

Caches in SMPs



Cache Coherence

The system must provide a coherent, uniform view of the memory to all processors, despite the presence of local, private cache storage.

Options:

- **Hardware cache coherence**:
    - snoopy protocols — require a bus;
    - central directory protocols — complicated and may become a bottleneck;
    - a combination of both;
- **Software cache coherence**: provided by the system or left to the programmer

Cache Coherence (cont.)

Another classification:

- invalidation protocols: lines in the cache are marked “*invalid*”, without updating the data. Whenever a processor accesses an invalid cache line, it should trigger a miss;
- update protocols: a line is “*pushed*” in the caches that have copies of the cache line. Generates a lot more traffic if lines are big.

Smaller lines are better as units of cache coherence!

What is False Sharing?

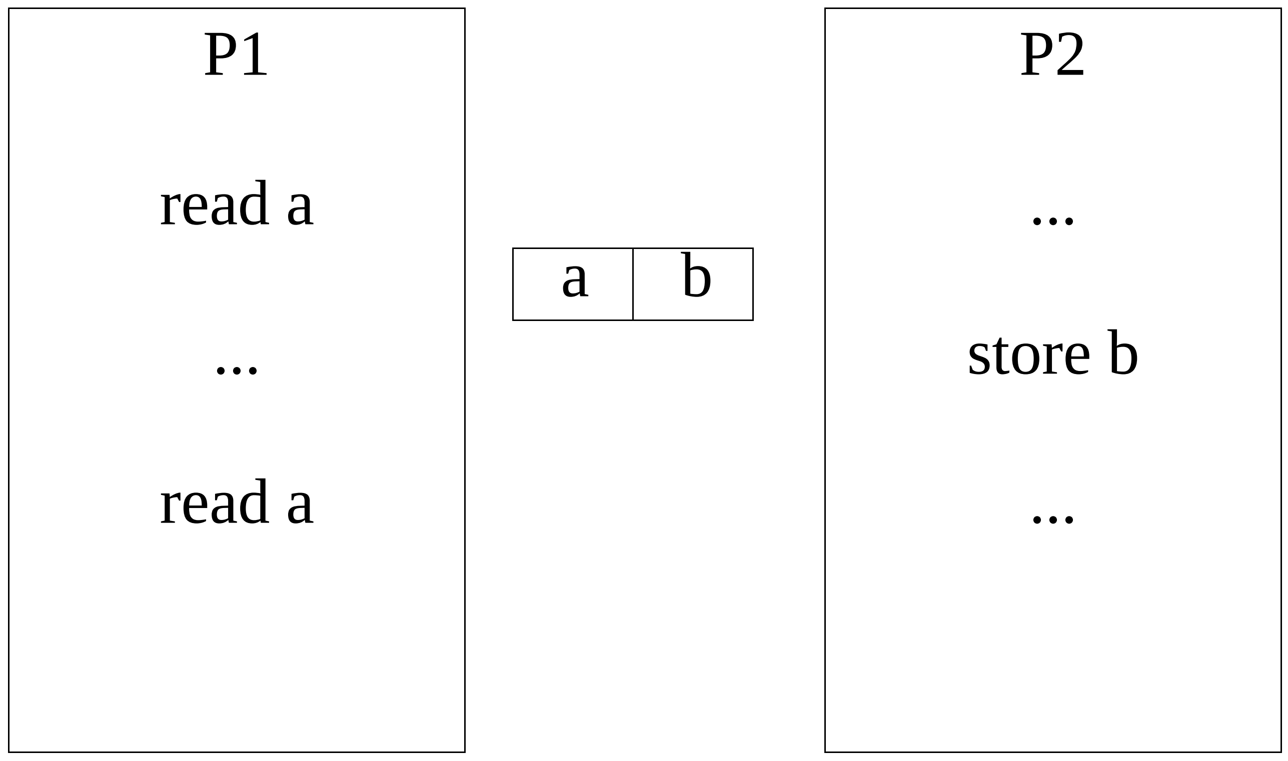
Two processors sharing a multi-word block because they need to access two different words that happen to be in the same cache block [Torrellas 1990]

If one of the accesses to the block is a write access, false sharing can induce a large number of cache misses (invalidations)

False sharing is an artifact introduced by data collocation.

Depends on the cache block (line) size and the particular placement of data in memory.

Example



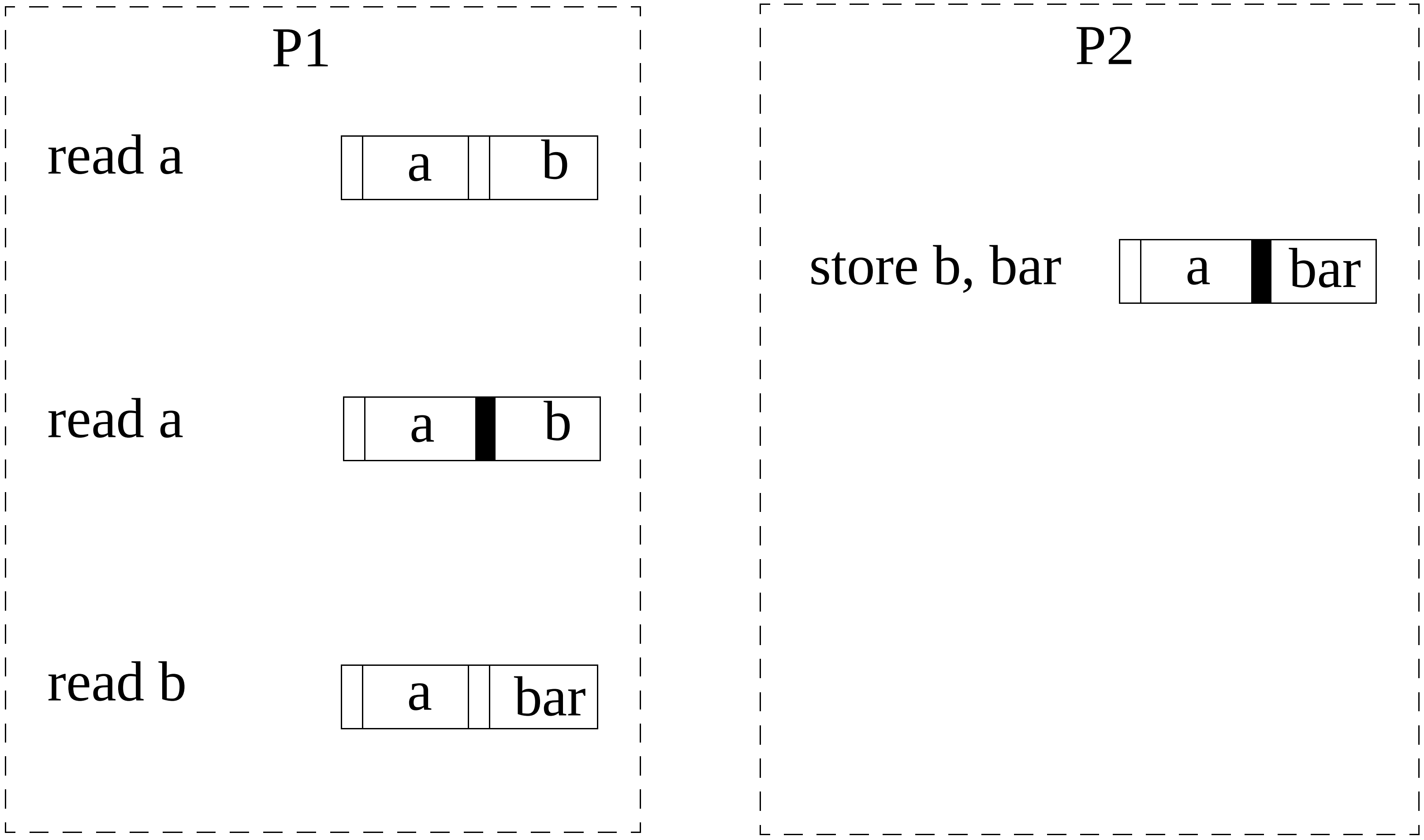
Hardware Solutions

Michel Dubois, Jonas Skeppstedt, Livio Ricciulli, Krishnan Ramamurthy, and Per Stenström [Dub93]

Write-through cache

- introduce an invalidation buffer, which could be implemented as a dirty bit associated with each word in each block in the cache, and
- on a store into a block, the address of the modified word is propagated to all processors with a copy of the block and also buffered in the invalidation buffer
- a local access to a word whose address is in the invalidation buffer invalidates the block and triggers a ‘true sharing miss’

Example

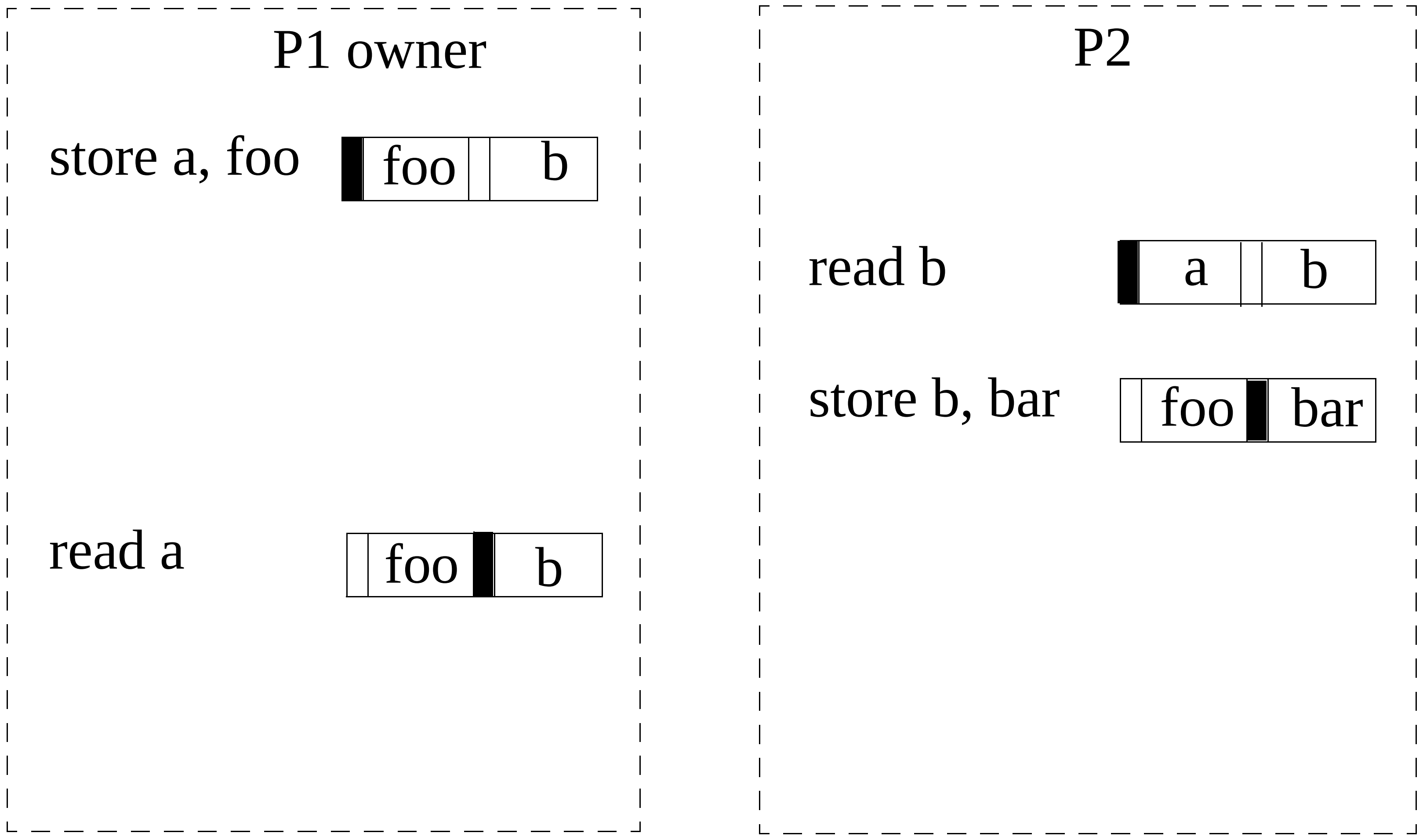


Hardware Solutions (continued)

Write back caches

- need to maintain ownership
- modify the algorithm such that stores accessing non-owned blocks with a pending invalidation for **ANY** of its words in the local invalidation buffer must trigger a miss

Example


P1 owner
store a, foo
foo
b
read a
foo
b
P2
read b
a
b
store b, bar
foo
bar

Software Solutions

Torrellas [Tor90] proposes the following “Data Placement Optimizations”, although not implemented in a compiler

- place scalar variables that exhibit false sharing in different cache blocks
- place active scalars that are protected by a lock in the same cache block as the lock
- allocate shared space requested by different processors from different heap regions
- position an array of records so that the number of different blocks that the average record spans is minimized (ALIGN)
- expand records in an array to minimize the sharing of a cache block by different records if the cache misses can be reduced and the space increase is tolerable. (PAD)

Software Solutions (cont.)

Eggers and Jeremiassen [EJ91] measure and identify false sharing for some applications written in C. Applying transformations to eliminate false sharing they are able to reduce false sharing misses with 40% to 75%, for a total cache misses reduction of 20%-30%.

Basic idea: data restructuring transformations such that

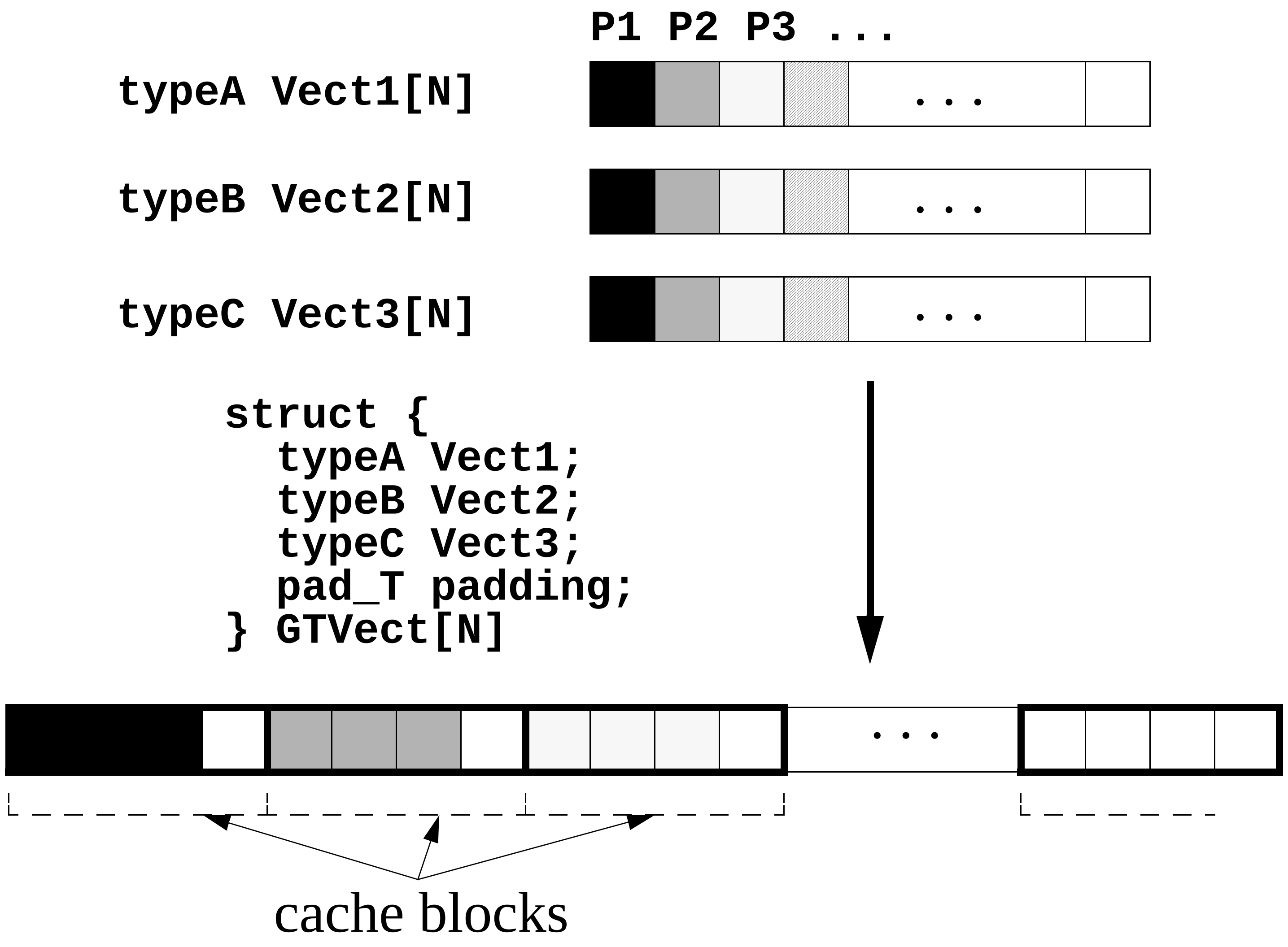
- data that is only, or mostly, accessed by one processor are grouped together, and
- writable shared data objects with no processor locality do not share cache lines

Software Solutions (cont.)

The transformations:

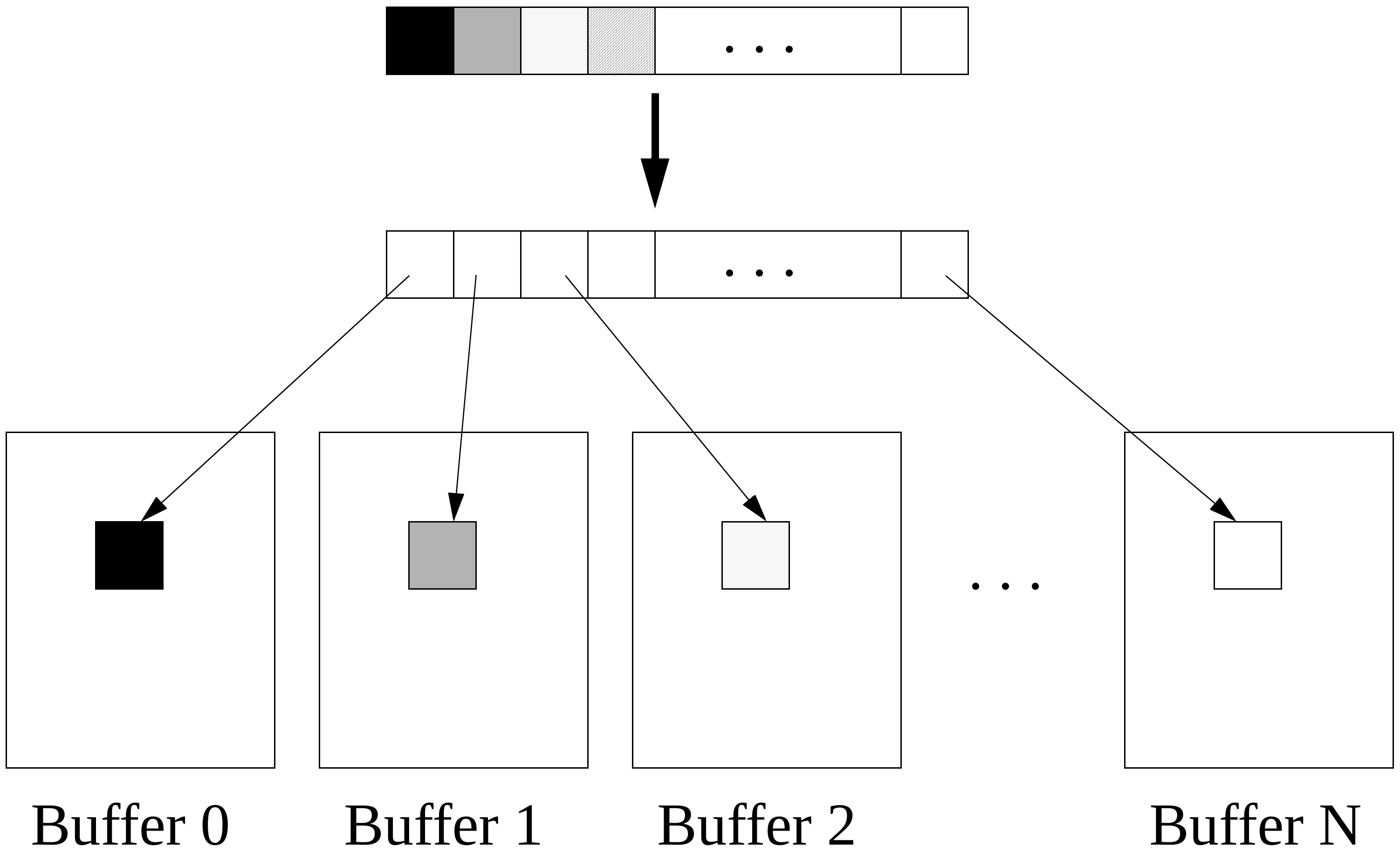
- **group and transpose** — group objects with similar sharing properties into vectors and transpose these vectors. Targets statically allocated data.
- **indirection** — technique used for dynamically allocated objects, in which blocks of memory are allocated for each processor and each element of the original shared data gets a pointer to a value in a block instead of the value itself.
- **pad and align** — shared data blocks are placed in their own cache blocks by padding their size and aligning them on the cache block boundaries.

Group and Transpose

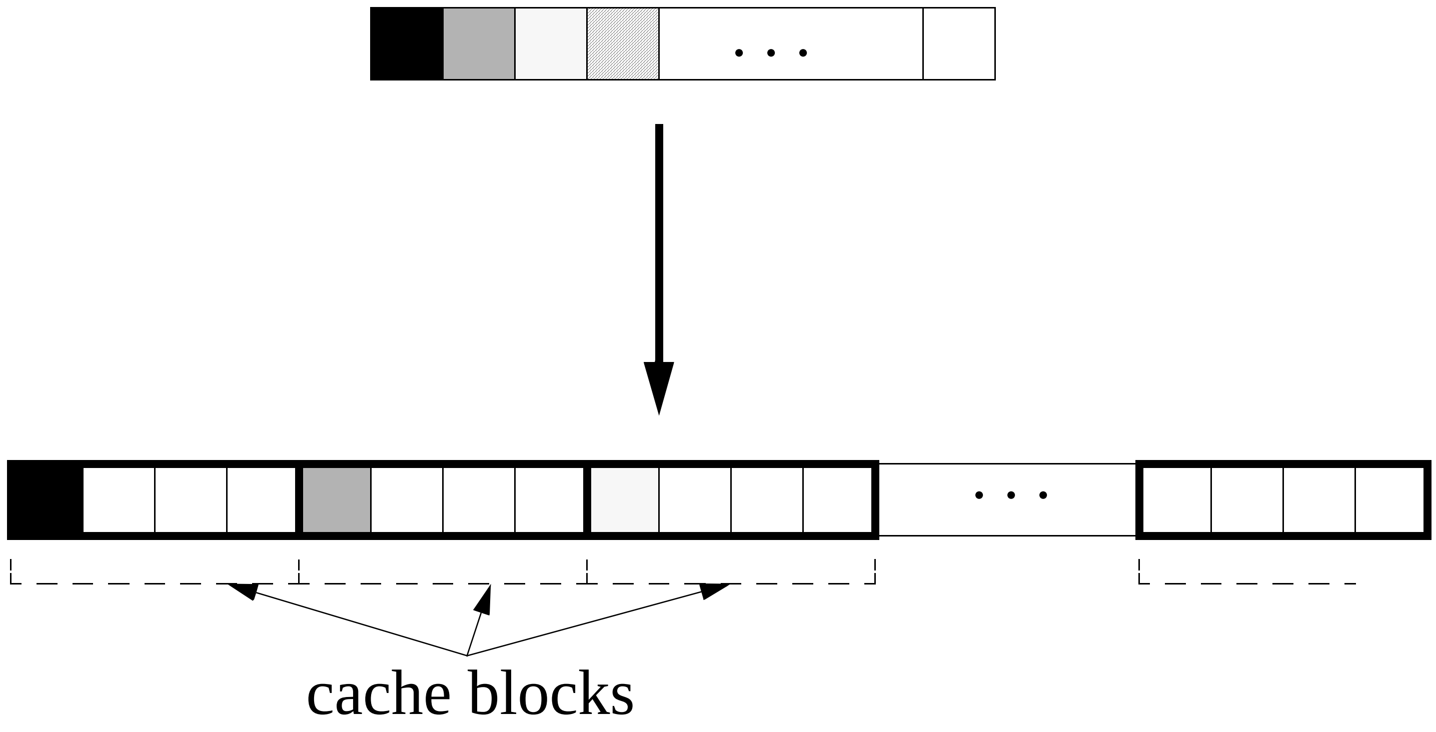


```
struct {
  typeA Vect1;
  typeB Vect2;
  typeC Vect3;
  pad_T padding;
} GTVect[N]
```

Indirection



Pad and Align



Heuristics

Used to decide which transformations are applied to which data structures.

Factors:

- data type: scalars, vectors, lock variables
- access type: read, write, shared, per-process
- access stride
- frequency of access to the elements of a data structure
- space and cost (group and transpose is *cheaper* than indirection)

Software Solutions (cont.)

Granston [Gra94] develops a loop transformation theory for eliminating false sharing

Transformations are:

- **processor-page alignment** — loop blocking and aligning + iteration mapping, with blocking factors constrained so that minimize sharing
- **loop distribution** — move multiple references that exhibit different sharing patterns in different loop nests

Conclusions

Hardware techniques exist for eliminating false sharing, although nobody has implemented them in real machines.

Software techniques have been implemented in parallelizing compilers, but false sharing continues to be a problem.

Reducing false sharing is beneficial because it reduces the number of cache misses and also the coherence traffic.

References

[Tor90] J. Torrellas, M. Lam, J. Hennessy, *Shared Data Placement Optimizations to Reduce Multiprocessor Cache Miss Rates*, International Conference on Parallel Processing (ICPP), 1990

[EJ91] S. Eggers, T. Jeremiassen, *Eliminating False Sharing*, ICPP, 1991

[Dub93] Michel Dubois, Jonas, Skeppstedt, Livio Ricciulli, Krishnan Ramamurthy, and Per Stenström, *The Detection and Elimination of Useless Misses in Multiprocessors*, ISCA, 1993

[Gra94] E. Granston, *Toward a Compile-Time Methodology for Reducing False Sharing and Communication Traffic in Shared Virtual Memory Systems*, Languages and Compilers for Parallel Computing, Springer-Verlag, 1994

Machine Problem on the WEB Site