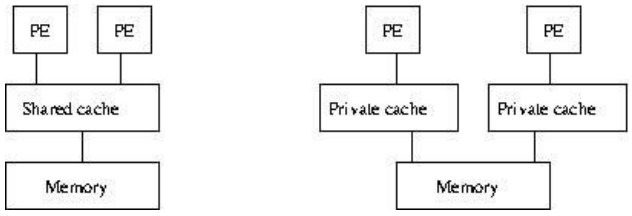
# Multiprocessor caching and false sharing

**Abstract**

This module presents the cache coherence problem for private caches and the potential for false sharing.

**Multiprocessor caching**

In the previous discussion of caching, the cache was presented as living between the processor and the memory system. This picture is more complicated for parallel systems because processing is done by multiple entities (potentially multiple processors but definitely multiple cores within each processor). The following images depict two ways this could be done, using PE (processing element) to stand for a processor or core:




The image on the left shows multiple PEs using a single cache while the image on the right shows each PE with a private cache. Real systems typically have several levels of cache, often arranged as a combination of these ideas; the level(s) of cache closest to the PEs are private while the lower level(s) of cache are shared.

Having private caches creates an added complication when code on the different PEs access the same memory address. In order to preserve the illusion of having a single memory, changes made by one of the PEs must be made visible to the others rather than just being made to the private cache. This need to share changes is called the *cache coherence* problem since it is concerned that the caches maintain a single coherent view of the contents of memory.

There are different ways to provide cache coherence, but the upshot is that private caches for the different PEs will sometimes need to invalidate a cache entry because that entry is needed by a different private cache. This is an unavoidable cost of using the same data on different PEs. Sometimes there is also an avoidable component of the cost, however, and that is the focus of this module.

Recall that each cache entry stores data from more than one address; this is how caches exploit spatial locality. In a multiprocessor setting, however, this can lead to a phenomenon called *false sharing*, which is when different PEs access different memory addresses but those

addresses are close enough to be stored in the same cache entry. No data is actually shared between the PEs, but the memory system still invalidates the cache entries as if it were.