Ex 1.5 Modern Processors are have multiple cores to enable simetric multiprocesses. as processes are distributed to the cpu. their cache gets more localized. the cache hierarchy is designed in such a way that the smaller caches are more local to a specific part of the cpu. this is to reduce the cache hits and keep the most local data the fastest

Ex 1.7 Least Recently Used (LRU) caching policies most benefit programs that exhibit temporal locality of data. it can be assumed that is you write data to an area of memory, you'll read it shortly. fifo can be worse in the situation where you frequently access data multiple times after it's first cached. int i = 0; int j = 1; "some other code" print(i , j) int k, l; print(i , j) # here if fifo was used, the variables may be unloaded from cache

Ex 1.8 the problem is that data tends to be grouped together since it was made near the same time. using the least significant bits will make the cache sparse and not good.

Ex 1.10 since memory bank conflicts happen when division occurs. primes can't exhibit this behavior because they can't be divided by any other numbers

Ex 1.11 the first algorithm reads and writes sparesly w/ stride, s which won't conflict. the second one will have stride 2 which will conflict.

Ex 1.15 for the most part. a compute-limited application will use all of the compute power allocated within margin of error. memory intensize applications will not show as high of a load average. showing memory cache misses can also give insight into whether an application is bandwidth bound or compute bound.