The most obvious place where spatial overheads cause a problem is in the address space: the 4 GB memory space (32-bit addresses) of a virtual computer won't fit inside the 4 GB (or less) available in real (host) hardware. Even with large amounts of host memory, we can't just declare `byte[] memory = new byte[4 * 1024 * 1024 * 1024];`. Somehow we must shrink our emulated address space to fit inside a single process on the host machine, and ideally with plenty of room to spare!

To save space, we first observe that the 4 GB address space is invariably not full. The typical machine will not exceed 2 GB of physical RAM, and we can get away with significantly less than this in most circumstances. So we can crush our 4 GB down quite quickly by observing that not all of it will be occupied by physical RAM.

The first step in designing our emulated physical address space has its origin in a little peek at the future. If we look up the road we will see that one of the features of the IA-32 memory management unit will help guide our structure for the address space. In protected mode, the memory management unit of the CPU carves the address space into indivisible chunks that are 4 KB wide (known as pages). So the obvious thing to do is to chunk our memory on the same scale.

Splitting our address space into 4 KB chunks means our address space no longer stores the data directly. Instead, the data is stored in atomic memory units, which are represented as various subclasses of `Memory`. The address space then holds references to these objects. The resultant structure and memory accesses are shown in Figure 9-2.

NOTE

To optimize `instanceof` lookups, we design the inheritance chain for `Memory` objects without using interfaces.

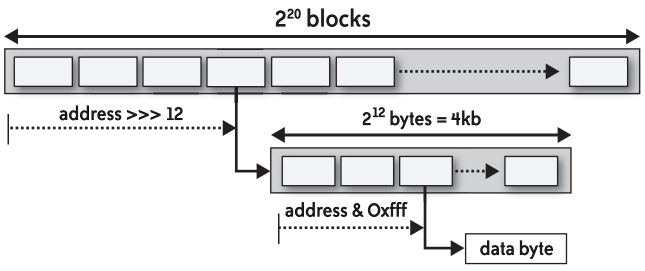

FIGURE 9-2. Physical address space block structure

This structure has a set of 2^{20} blocks, and each block will require a 32-bit reference to hold it. If we hold these in an array (the most obvious choice), we have a memory overhead of 4 MB, which is not significant for most instances.