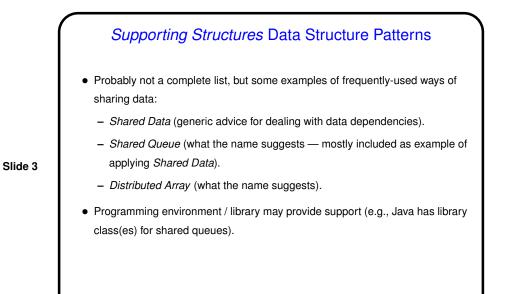


Slide 4



Many applications — especially ones using a master/worker approach — need a shared queue. Programming environment might provide one, or might not. Nice example of dealing with a shared data structure anyway.
Java code in figures 5.37 (p. 185) through 5.40 (p. 189) presents a step-by-step approach to developing implementation.

## Shared Queue, Continued

Simplest approach to managing a shared data structure where concurrent modifications might cause trouble — one-at-a-time execution. Shown in figures 5.37 (nonblocking) and 5.38 (block-on-empty). Only tricky bits are use of dummy first node and details of take. Reasons to become clearer later. Usually a good idea to try simplest approach first, and only try more complex

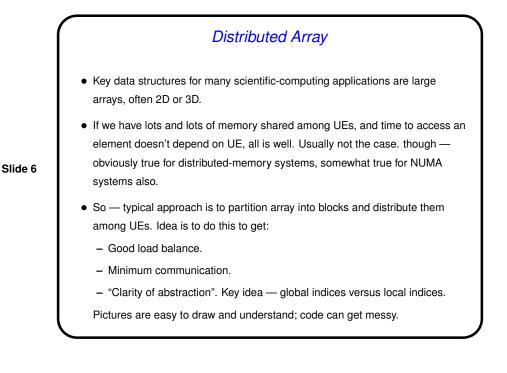
Slide 5

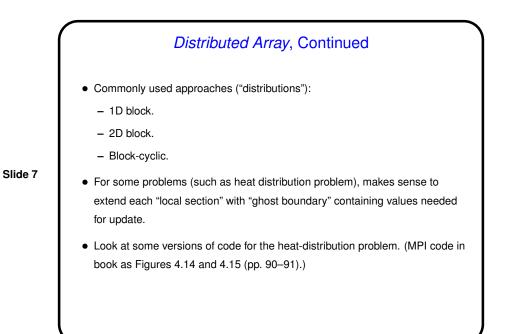
all evil." Attributed to D. E. Knuth; may actually be C. A. R. Hoare.)
Here, next thing to try is concurrent calls to put and take. Not too hard for nonblocking queue — figure 5.39. Tougher for block-on-empty queue —

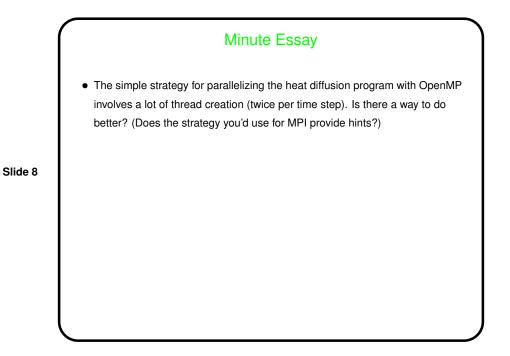
ones if better performance is needed. ("Premature optimization is the root of

• If still too slow, or a bottleneck for large numbers of UE, explore distributed queue.

figure 5.40. In both cases, must be very careful.







## Minute Essay Answer

 There's certainly a way that might do better: You could essentially duplicate the MPI strategy in OpenMP – make the whole program an OpenMP "parallel section", with each thread doing the time step loop, with barriers at the end of each phase of the calculation. We did something like this with the numerical integration example — "SPMD" versions in OpenMP.

Slide 9