

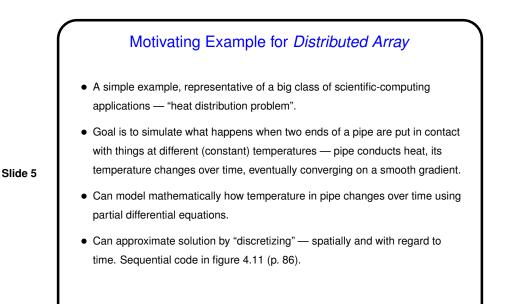
Slide 3

## 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 ones if better performance is needed. ("Premature optimization is the root of 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 figure 5.40. In both cases, must be very careful.
- If still too slow, or a bottleneck for large numbers of UE, explore distributed queue.



Motivating Example, Continued

- How to parallelize? Obvious places to look for lots of tasks are loops. Time-step loop doesn't look promising — values at each step depend on values at previous step. Loop over points seems more likely.
- Could consider each iteration as a task. Here, though, makes at least as much sense to focus on decomposing large data structures (arrays) and operating on elements concurrently.
- Dependencies among tasks / data elements: At each step, we first update ukp1 using uk (where for each point we need values from neighbors too), then update uk using ukp1. All elements can be updated concurrently.
- Since main source of concurrency appears to involve updating a large data structure, with tasks that aren't completely independent algorithm structure is *Geometric Decomposition*.

