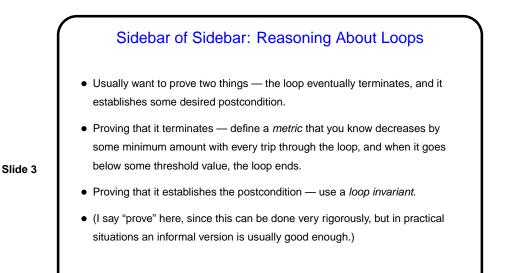


## Sidebar: Reasoning about Concurrent Algorithms (Review)

 For concurrent algorithms (such as various solutions proposed for mutual exclusion problem), testing is less helpful than for sequential algorithms. (Why?)

- May be helpful, then, to try to think through whether they work. How? Idea of "invariant" may be useful:
  - Loosely speaking "something about the program that's always true". (If this reminds you of "loop invariants" in CSCI 1323 — good.)
  - Goal is to come up with an invariant that's easy to verify by looking at the code and implies the property you want (here, "no more than one process in its critical region at a time").
  - We will do this quite informally, but it can be done much more formally mathematical "proof of correctness" of the algorithm.



Reasoning About Loops, Continued
What's a loop invariant? in the context of reasoning about programs, it's a *predicate* (boolean expression using program variables) that

is true before the loop starts, and
if true before a trip through the loop, with the loop condition true, is also true after the trip through the loop.

If you can prove that a particular predicate is a loop invariant, after the loop exits, you know it's still true, and the loop condition is not. With a well-chosen invariant, this is enough to prove useful things.
(Might be worth noting that compiler writers have a different definition — some computation that can be moved outside the loop.)

 $\bullet \ \mbox{Loop}$  to compute sum of elements of array a of size n:

```
i = 0; sum = 0;
while (i != n) {
    sum = sum + a[i];
```

Slide 5

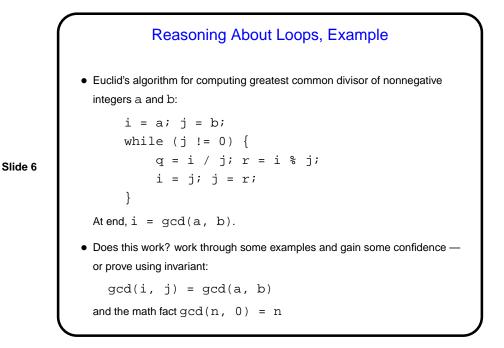
At end, sum is sum of elements of a.

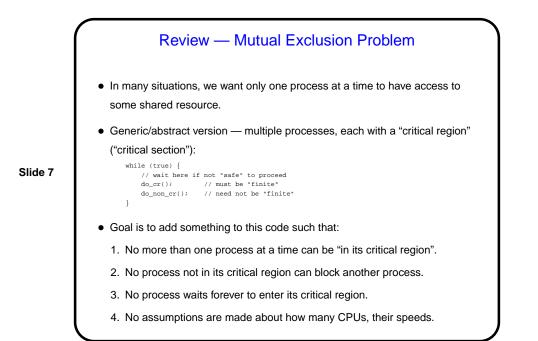
}

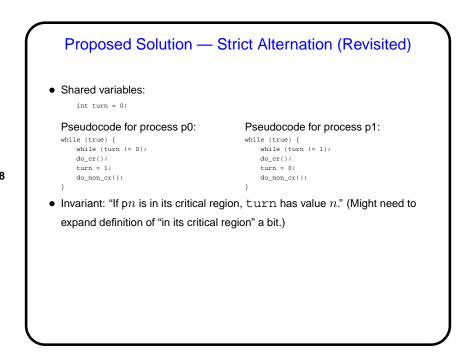
• Does this work? well, you probably believe it does, but you could prove it using the invariant:

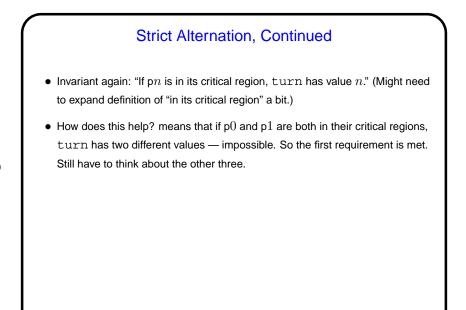
Reasoning About Loops, Simple Example

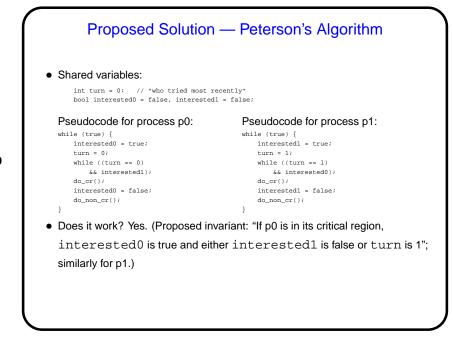
sum is the sum of a[0] through a[i-1]

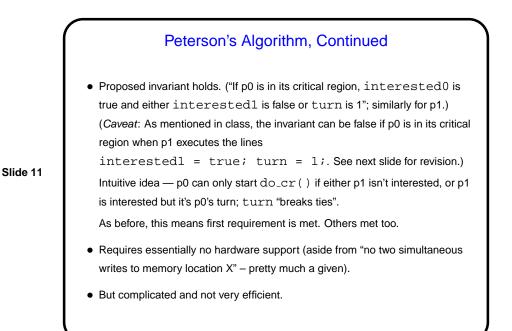


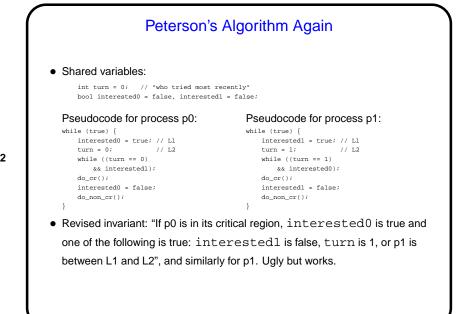


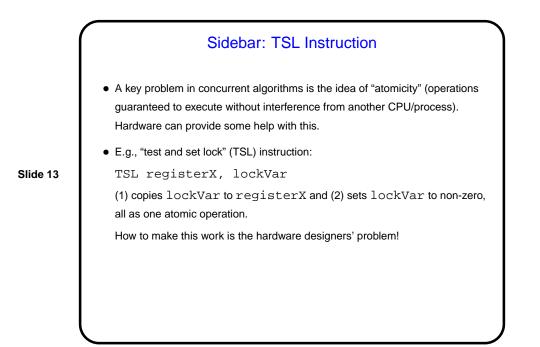


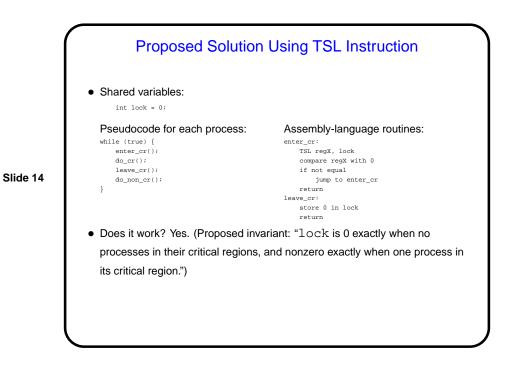


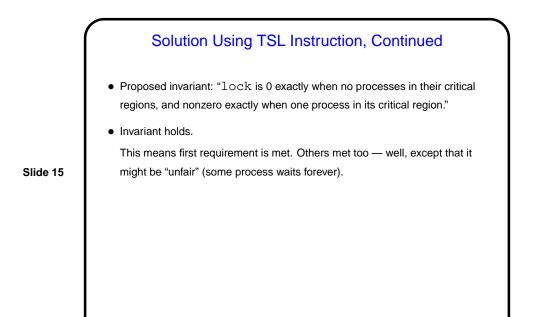












Minute Essay
The TSL-based solution seems okay, but — are there circumstances in which a process might wait forever?
Even if this solution is correct, might it have disadvantages?

