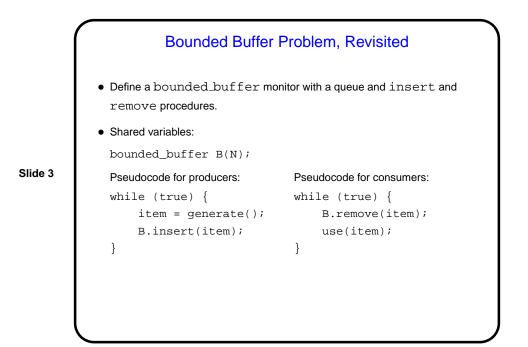
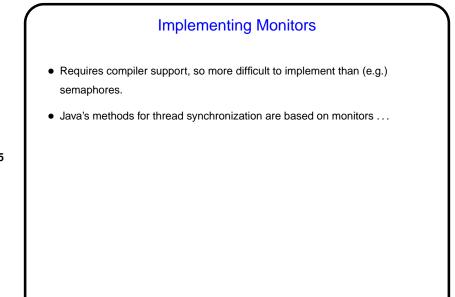
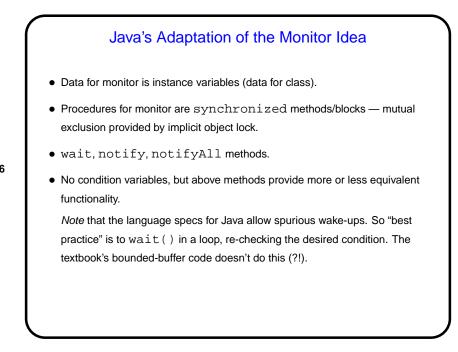


## Another Synchronization Mechanism — Monitors History — Hoare (1975) and Brinch Hansen (1975). Idea — combine synchronization and object-oriented paradigm. A monitor consists of Data for a shared object (and initial values). Procedures — only one at a time can run. "Condition variable" ADT allows us to wait for specified conditions (e.g., buffer not empty): Value — queue of suspended processes. Operations: \* Wait — suspend execution (and release mutual exclusion). \* Signal — *if* there are processes suspended, allow *one* to continue. (if not, signal is "lost"). Some choices about whether signalling process continues, or signalled process awakens right away.



```
Bounded-Buffer Monitor
           • Data:
                 buffer B(N); // N constant, buffer empty
                 int count = 0;
                 condition full;
                 condition empty;
Slide 4
           • Procedures:
             insert(item itm) { remove(item &itm) {
                                        if (count == 0)
                 if (count == N)
                     wait(full);
                                               wait(empty);
                 put(itm, B);
                                           itm = get(B);
                 count += 1;
                                           count -= 1;
                 signal(empty);
                                           signal(full);
                                       }
             }
           • Does this work? (Yes.)
```

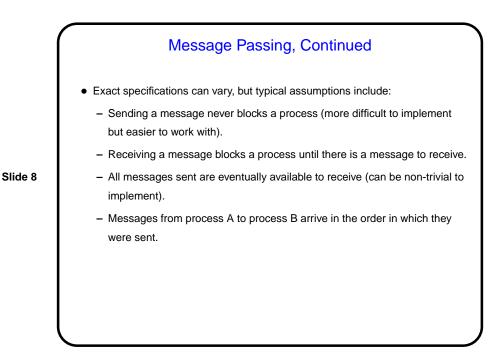


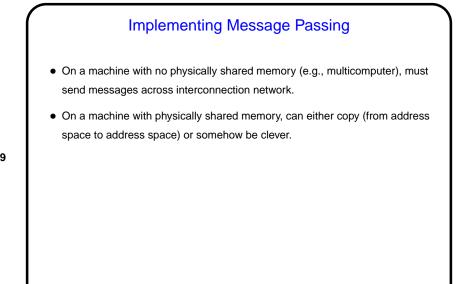


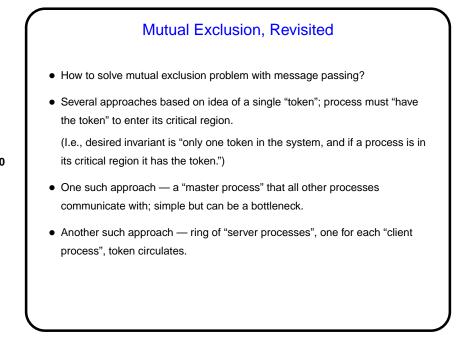


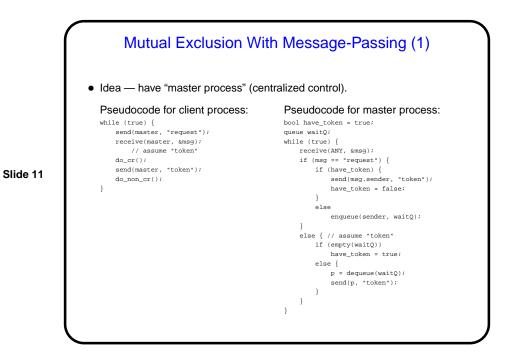
• Previous synchronization mechanisms all involve shared variables; okay in some circumstances but not very feasible in others (e.g., multiple-processor system without shared memory).

- Idea of message passing each process has a unique ID; two basic operations:
  - Send specify destination ID, data to send (message).
  - Receive specify source ID, buffer to hold received data. Usually some way to let source ID be "any".









Mutual Exclusion With Message-Passing (2) • Idea — ring of servers, one for each client. Pseudocode for client process: Pseudocode for server process: while (true) { bool need\_token = false; if (my\_id == first) send(my\_server, "request"); receive(my\_server, &msg); send(next\_server, "token"); while (true) {
 receive(ANY, &msg); // assume "token" do\_cr(); send(my\_server, "token"); if (msg == "request") need\_token = true; else { // assume "token" do\_non\_cr(); } if (msg.sender == my\_client) {
 need\_token = false;
 send(next\_server, "token"); } else if (need\_token)
 send(my\_client, "token"); else send(next\_server, "token"); } }

