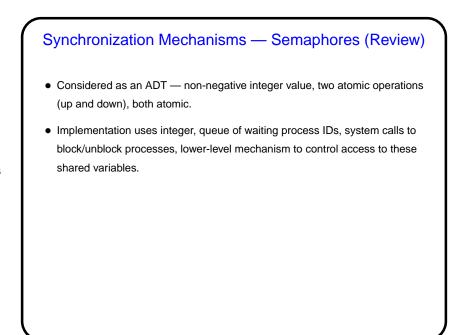
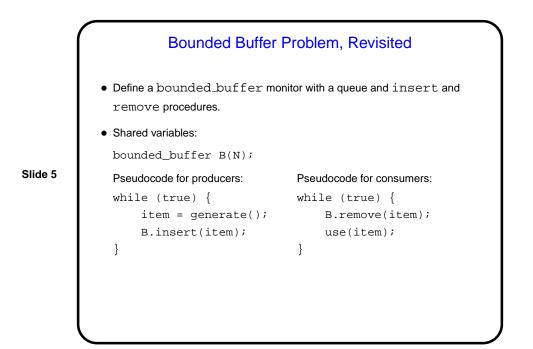


Synchronization Mechanisms — Review/Recap
Synchronization using only shared variables seems to be tedious and inefficient.
"Synchronization mechanisms" are more-abstract ways of coordinating what processes do. A key point is providing *something* that potentially makes a process wait.

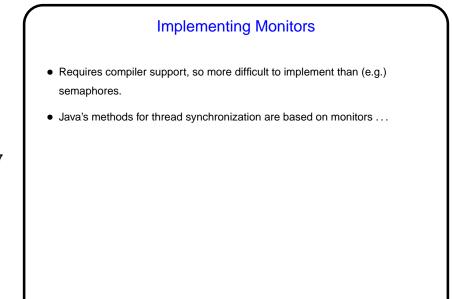


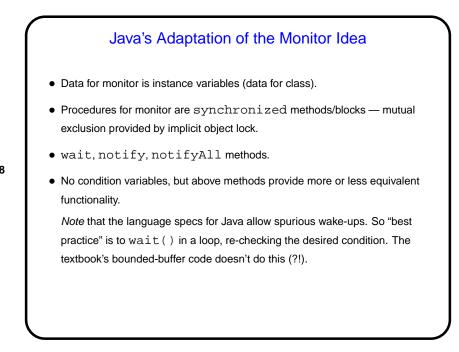
Slide 4

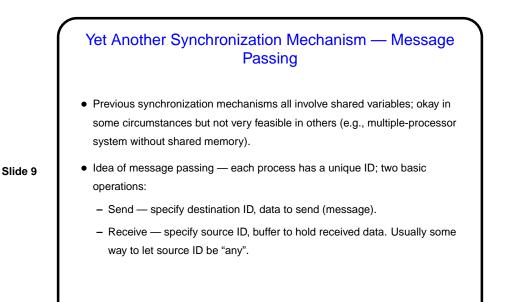
Another Synchronization Mechanism — Monitors History — Hoare (1975) and Brinch Hansen (1975). Idea — combine synchronization and object-oriented paradigm. A monitor consists of Data for a shared object (and initial values). Procedures — only one at a time can run. "Condition variable" ADT allows us to 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.

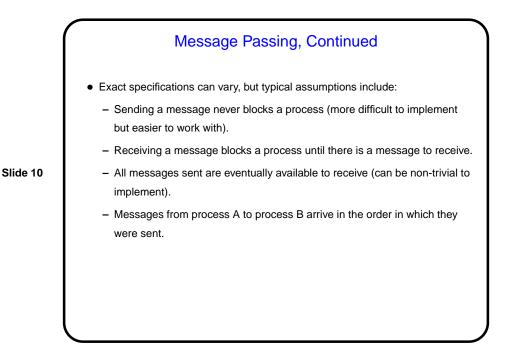


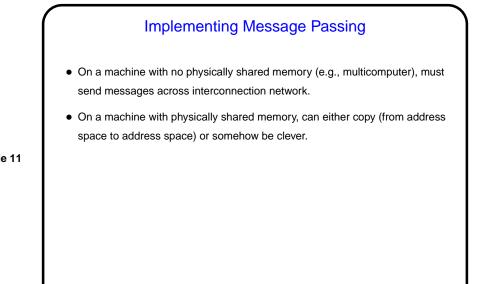
```
Bounded-Buffer Monitor
           • Data:
                 buffer B(N); // N constant, buffer empty
                 int count = 0;
                 condition full;
                 condition empty;
Slide 6
           • Procedures:
             insert(item itm) { remove(item &itm) {
                                       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.)
```

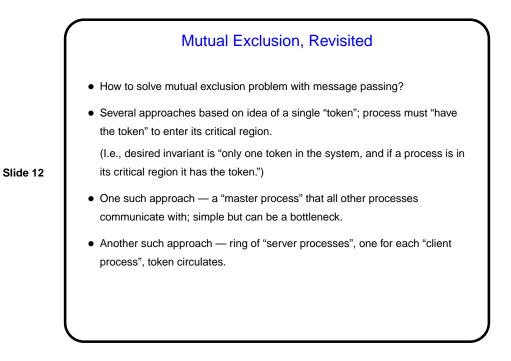


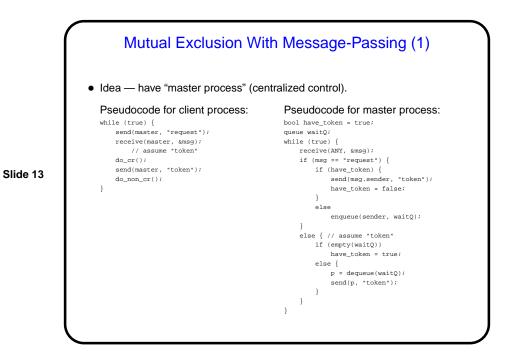












Mutual Exclusion With Message-Passing (2) • Idea — ring of servers, one for each client. Pseudocode for client process: Pseudocode for server process: while (true) { bool need_token = false; if (my_id == first) send(my_server, "request"); receive(my_server, &msg); send(next_server, "token"); while (true) {
 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"); } }

