

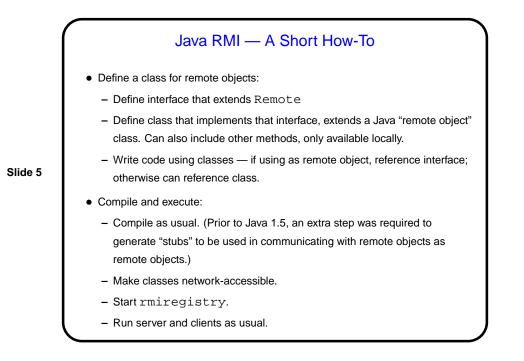
Slide 2



- Motivation for client/server applications, can be annoying to have to design your own protocol.
- Instead, idea is to define "remote objects" that can be treated (at program level) like any other objects invoke methods.
- Slide 3
- Typical use in client/server program:
 - Server creates some remote objects and "registers" them.
 - Clients look up server's remote objects and invoke their methods.
 - Both sides can pass around references to other remote objects.



- Example simplified generic master/worker program, similar to the versions in OpenMP and MPI.
- Version using sockets is relatively straightforward server creates a new thread for each client, only tricky bits are in making sure things are shut down properly. Notice use of synchronized in code to ensure thread-safe access to shared variables.
- Version using RMI is also straightforward, again except for code to shut down
 properly. Notice use of synchronized in code to ensure thread-safe
 access to shared variables; experiment suggests that RMI may use multiple
 threads to process concurrent requests.
- Slide 4



Distributed-Memory Java and Implementation Mechanisms

 Very similar to MPI, really — UE management is outside the scope of the libraries, synchronization is implicit. For sockets, communication is explicit; for RMI, implicit.

Slide 6

