

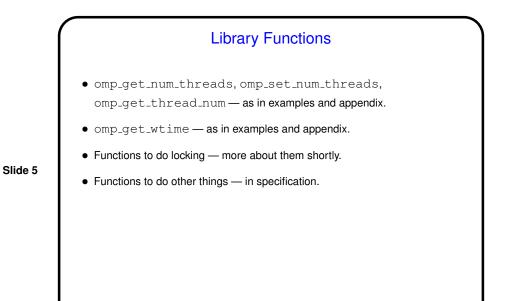
## **OpenMP Constructs — Basic Categories**

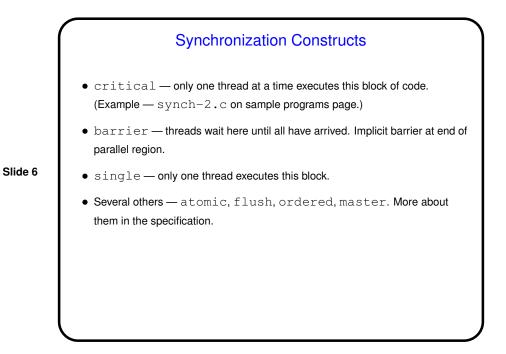
- Parallel regions ("replicate the following in all threads").
- Worksharing ("divide the following among threads").
- Data environment (shared variables versus per-thread variables).

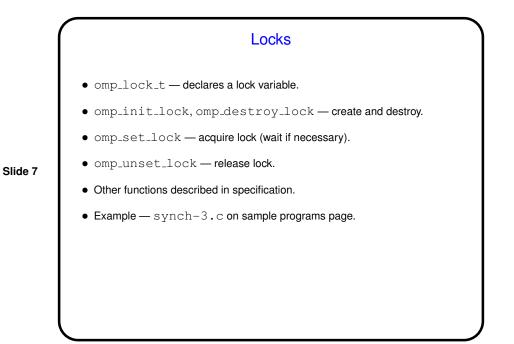
Slide 4

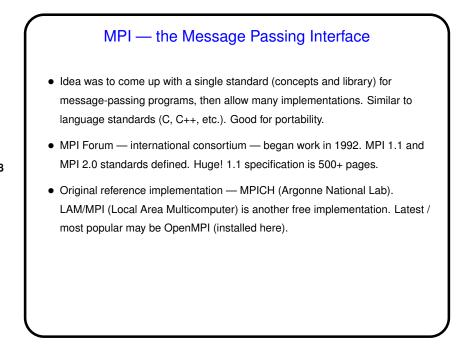
• Runtime functions / environment variables.

• Synchronization.











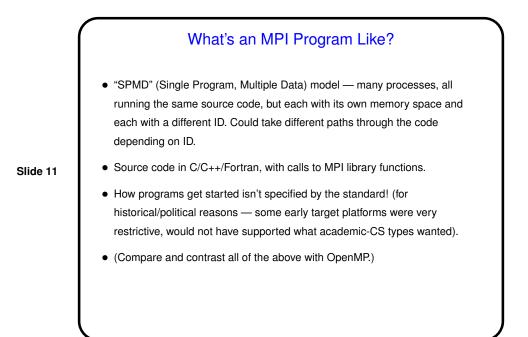
 "SPMD" (Single Program, Multiple Data) model — many processes, all running the same source code, but each with its own memory space and each with a different ID. Could take different paths through the code depending on ID.

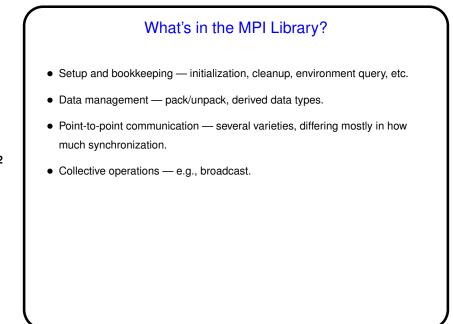
Slide 9

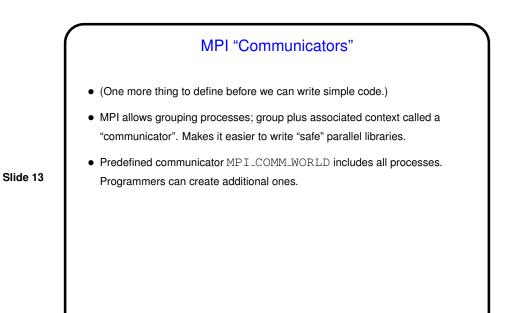
- Source code in C/C++/Fortran, with calls to MPI library functions.
- How programs get started isn't specified by the standard! (for historical/political reasons — some early target platforms were very restrictive, would not have supported what academic-CS types wanted).
- (Compare and contrast all of the above with OpenMP.)



- Idea was to come up with a single standard (concepts and library) for message-passing programs, then allow many implementations. Similar to language standards (C, C++, etc.). Good for portability.
- MPI Forum international consortium began work in 1992. MPI 1.1 and MPI 2.0 standards defined. Huge! 1.1 specification is 500+ pages.
- Original reference implementation MPICH (Argonne National Lab).
   LAM/MPI (Local Area Multicomputer) is another free implementation. Latest / most popular may be OpenMPI (installed here).

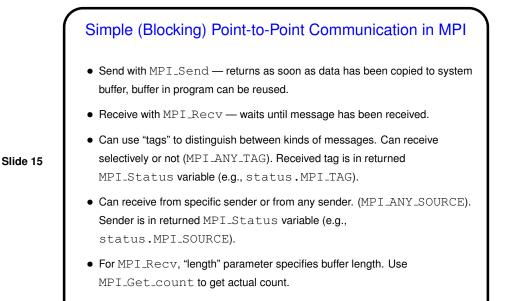






## Simple Examples / Compiling and Executing

- Look at sample program hello.c. (All sample programs from class should be on the Web, linked from course "sample programs" page, with short instructions on how to use MPI.)
- We'll use OpenMPI as installed on the F9 lab machines. There should be man pages for all commands and functions.
- Slide 14
- Compile with mpicc.
- Execute with mpirun.

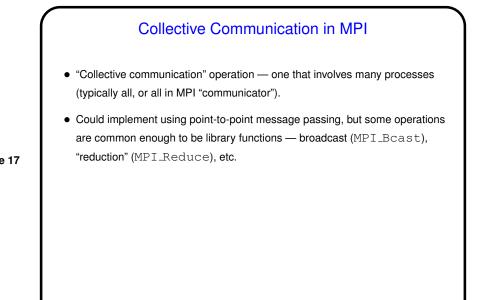


• Look at sample program send-recv.c.

## Not-So-Simple Point-to-Point Communication in MPI

- For not-too-long messages and when readability is more important than performance, MPI\_Send and MPI\_Recv are probably fine.
- If messages are long, however, buffering can be a problem, and can even lead to deadlock. Also, sometimes it's nice to be able to overlap computation and communication.
- Therefore, MPI offers several other kinds of send/receive functions --"synchronous" (blocks both sender and receiver until communication can take place), "non-blocking" (doesn't block at all, program must later test/wait for communication to take place).

(More about these later.)



	Minute Essay
Slide 18	<ul> <li>If you add the following lines to sample program send-recv.c, right after the call to printf() for process 0</li> <li>buff[0] = 30;</li> <li>buff[1] = 40;</li> <li>what does process 1 print?</li> </ul>

