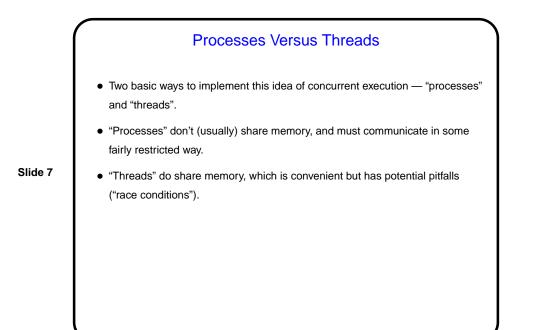


Slide 5

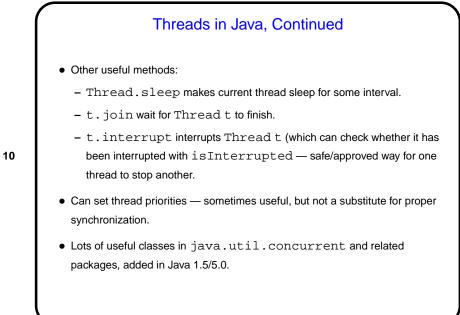
## Concurrency Basics Textbooks on operating systems talk about "processes" — "threads of control" executing "concurrently", i.e., at the same time (in fact or in effect). Each is a sequence of steps, like the (sequential) programs you've written. How does it work? Conceptually, all processes not waiting for something (such as I/O) run at the same time. Operating system basically simulates one CPU per thread, with real CPU(s) switching back and forth among them. This turns out to be a good mental model for managing applications, and activities of the O/S itself. It also means you could get better performance with more than one CPU/core — can potentially have more than one thing actually running at the same time. But there are some potential pitfalls, involving interaction among processes.



Multithreading in Java
Interestingly enough, Java has included support for multiple threads from the beginning — probably because it's a good mental model for GUIs.
Interaction among Java threads based on "monitors" (see textbooks on operating systems, parallel programming — idea goes back to 1975 papers by Hoare and Brinch Hansen). Java leaves out some aspects of full-fledged idea, but keeps enough to be useful.

Slide 8

	Threads in Java
	<ul> <li>Thread class provides basic functionality. To start a new thread, make a Thread object and call its start method. Two choices:</li> </ul>
	<ul> <li>Create a Thread with an object that implements Runnable — run method has code to execute.</li> </ul>
Slide 9	<ul> <li>Define a subclass of Thread that has a run method with code to execute.</li> </ul>
	<ul> <li>Interthread interaction based on (implicit) locks:</li> </ul>
	<ul> <li>Every object (and every class) has a lock.</li> </ul>
	<ul> <li>synchronized methods must acquire lock — so only one at a time can run.</li> </ul>
	<ul> <li>wait gives up the lock and sleeps; notify and notifyAll wake up one/all sleeping thread(s).</li> </ul>



Slide 10

