- to clarify:
  - a CPU is connected to Mm module via BUS. In smaller multiprocessors, it makes sense to use a shared bus to connect multiple CPUs to the shared Mm module. bus arbitration will ensure that a single operation is on the bus at a time (otherwise there will be garbage). As the number of CPUs grows, there is more contention for the bus, each CPU will have to wait potentially longer to get to Mm, you'd want to have a solution which permits multiple CPUs to go to Mm at a single moment. Clearly shared bus is not going to permit that.

  - therefore, larger multiprocessor systems (anything with more than 64 CPU definitely, maybe even for less CPUs), use an interconnection network which connects CPUs to different memory modules. Depending on the paths through the interconnection network, multiple CPUs can access different Mm modules at the same time. If the paths from two CPUs to two different Mm modules requires going through different switches in the I/C network, then this is possible. If the paths go through same switches, then one request would have to wait. Supercomputers, massively parallel processors,… use interconnection networks as opposed to shared bus.

- Processes Overview, Silberschatz and Galvin Chapter 3.1-3.4
- Intro to Threads, Birrell "An Introduction to Programming with Threads"
- Multithreading & Thread Design Patterns, Context Switch details, Silberschatz 4
- Scheduling, Silberschatz Ch. 5
- Scheduling in CMTs, Chip Multithreading Systems Need a New Operating System Scheduler
- Synchronization constructs, Silberschatz Chapter 6.1-6.8
- Serializers, Path Expressions, RWLocks, and Semaphores, Bloom's paper
- Deadlocks and Priority Inversion, Silberschatz Ch. 7
- Kernel thread implementation, Eykholt et al. paper on Multithreading the SunOS kernel
- User-level thread library implementation discussion, Stein and Shah "lightweight threads" paper
- Psyche threads, First-class user-level threads paper
- Spinlocks for SMPs, Performance of Spinlock Alternatives for SMPs

```
Mutex m; Condition red_cond, green_cond;
int red_waiting = 0; int green = 0, red = 0;
```

Red:

```
Lock(m) {
 while (green + red == 3)
  Wait(m, red_cond);
 red++;
 // ???
}
... //read data
Lock(m) {
 red--;
 // ???
 // ???
 // ???
}
```

Green:

```
Lock(m) {
 while (green + red == 3
     || red_waiting != 0)
  Wait(m, green_cond);
 green++;
}
... //write data
Lock(mutex) {
 green--;
 // ???
 // ???
 // ???
}
```