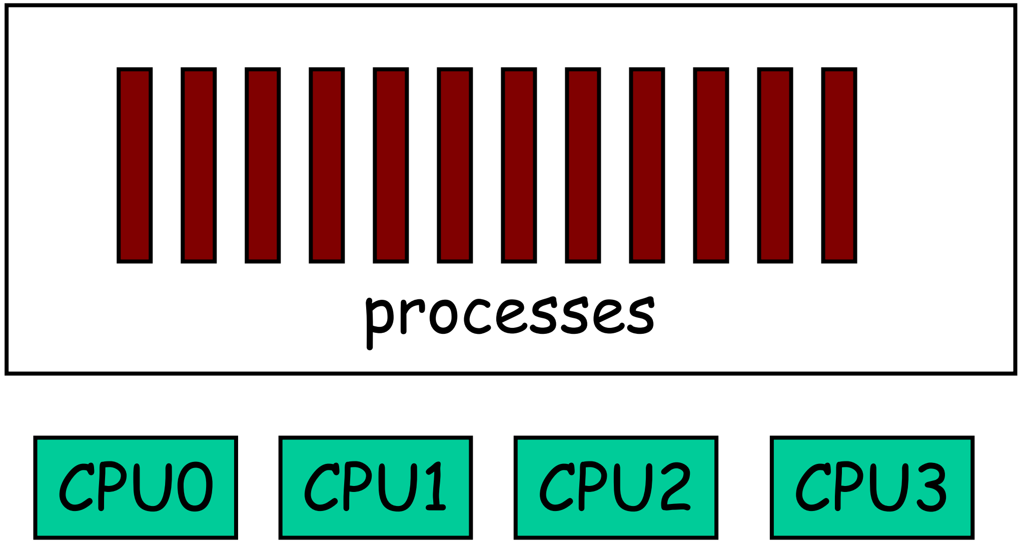
Multiprocessor scheduling issues

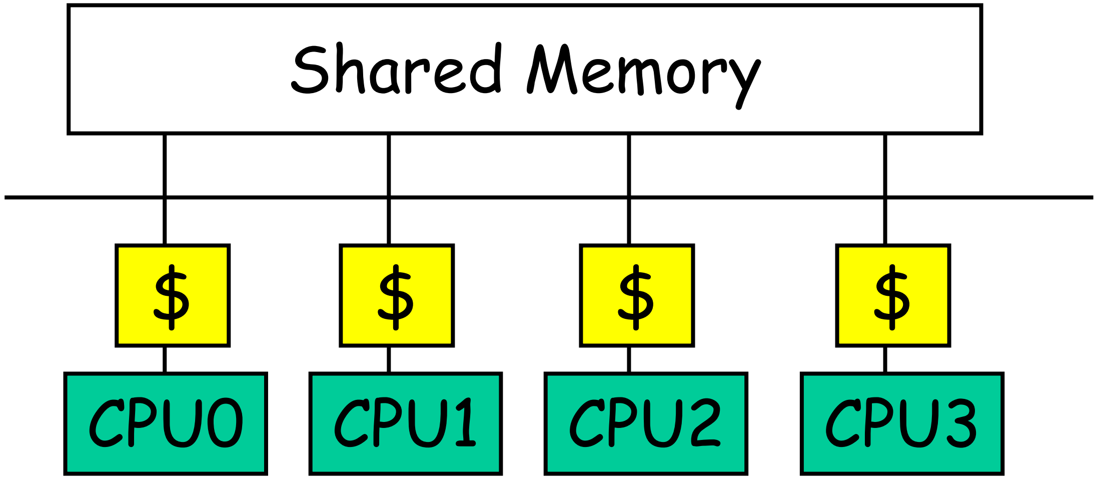
- Shared-memory Multiprocessor



- How to allocate processes to CPU?

Symmetric multiprocessor

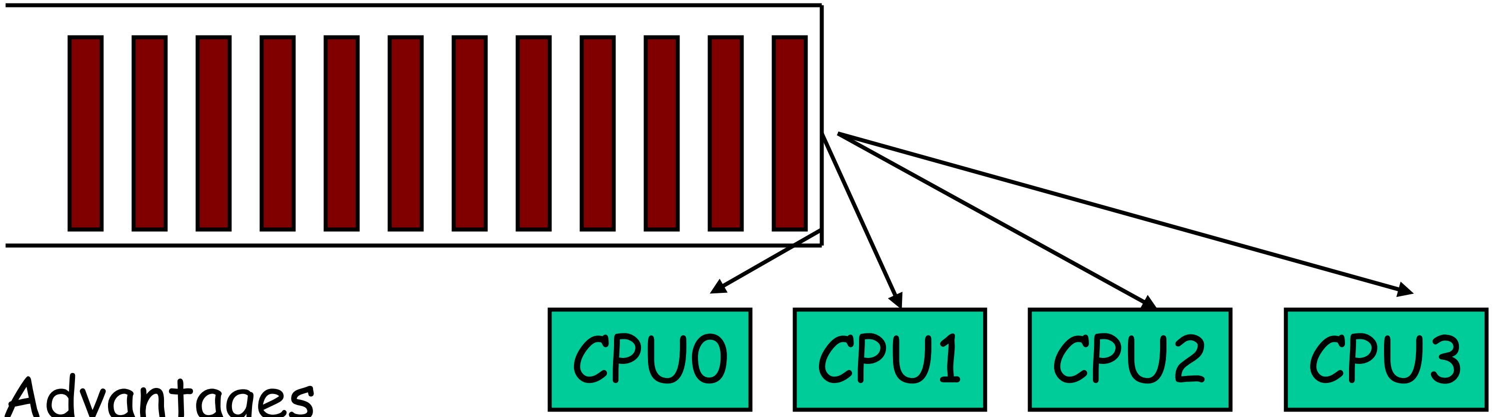
- Architecture



- Small number of CPUs
- Same access time to main memory
- Private cache

Global queue of processes

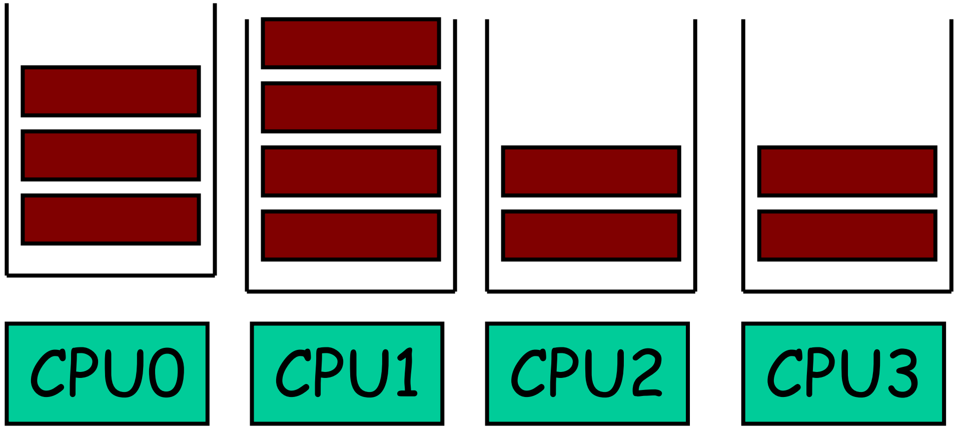
- One ready queue shared across all CPUs



- Advantages
  - Good CPU utilization
  - Fair to all processes
- Disadvantages
  - Not scalable (contention for global queue lock)
  - Poor cache locality
- Linux 2.4 uses global queue

Per-CPU queue of processes

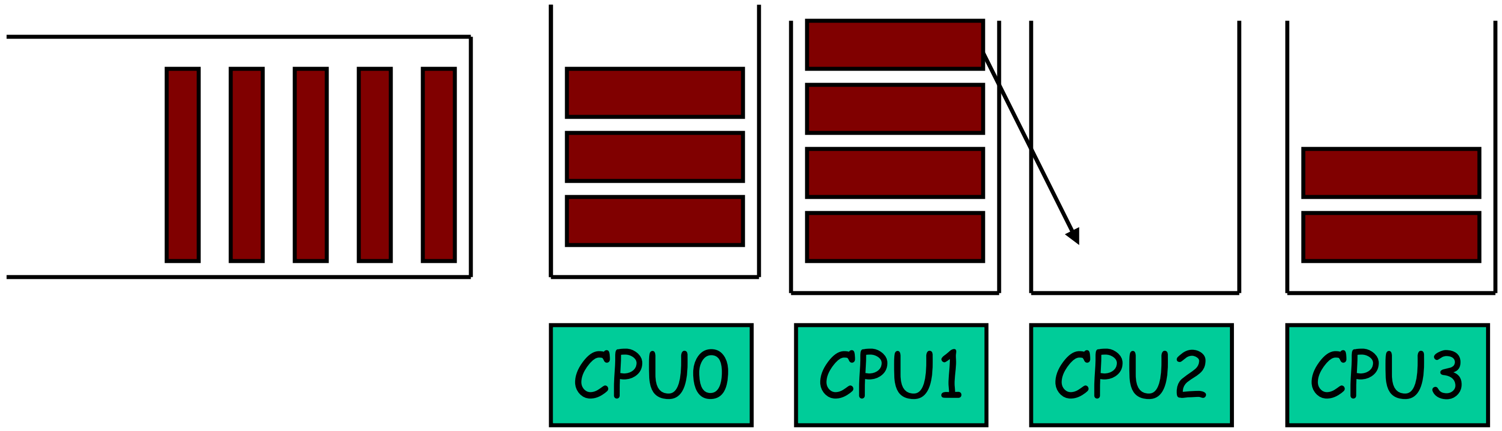
- Static partition of processes to CPUs



- Advantages
  - Easy to implement
  - Scalable (no contention on ready queue)
  - Better cache locality
- Disadvantages
  - Load-imbalance (some CPUs have more processes)
    - Unfair to processes and lower CPU utilization

Hybrid approach

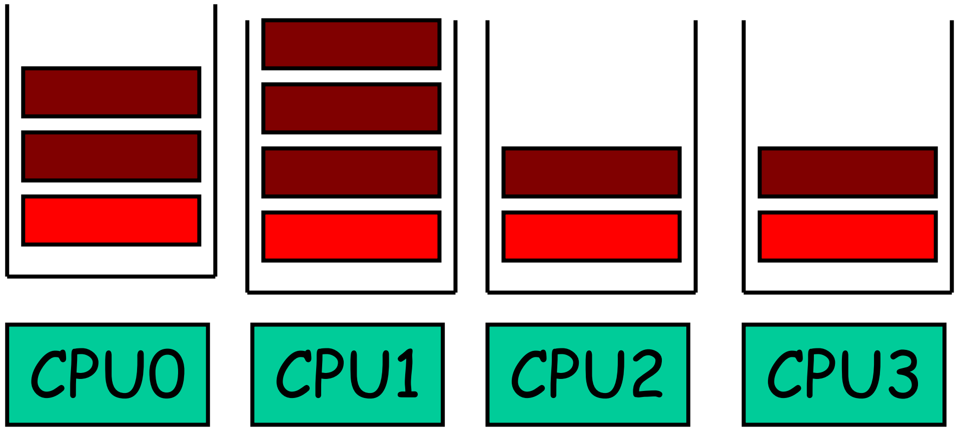
- Use both global and per-CPU queues
- Balance jobs across queues



- Processor Affinity
  - Add process to a CPU's queue if recently run on the CPU
    - Cache state may still present
- Linux 2.6 uses a very similar approach

SMP: “gang“ scheduling

- Multiple processes need coordination
- Should be scheduled simultaneously



- Scheduler on each CPU does not act independently
- Coscheduling (gang scheduling): run a set of processes simultaneously
- Global context-switch across all CPUs