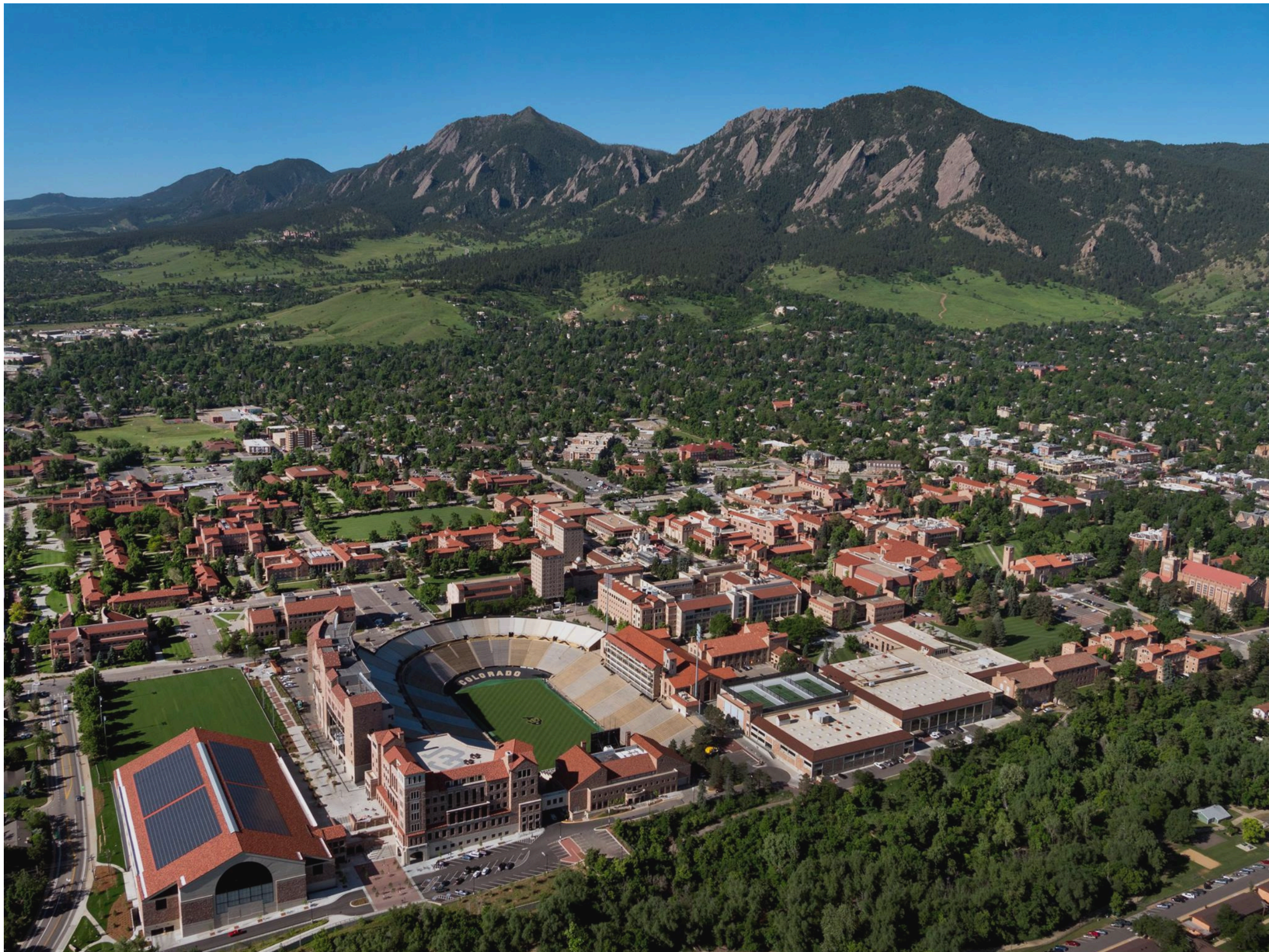
Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder

These slides adapted from materials provided by the textbook authors.



Realtime and Multi-Core Scheduling

Real Time Scheduling in Linux

- Linux also includes three **real-time** scheduling classes:
  - Real time FIFO – soft real time (SCHED_FIFO)
  - Real time Round Robin – soft real time (SCHED_RR)
  - Real time Earliest Deadline First – hard real time as of Linux 3.14 (SCHED_DEADLINE)

- Only processes with the priorities 0-99 have access to these RT schedulers

Real Time Scheduling in Linux

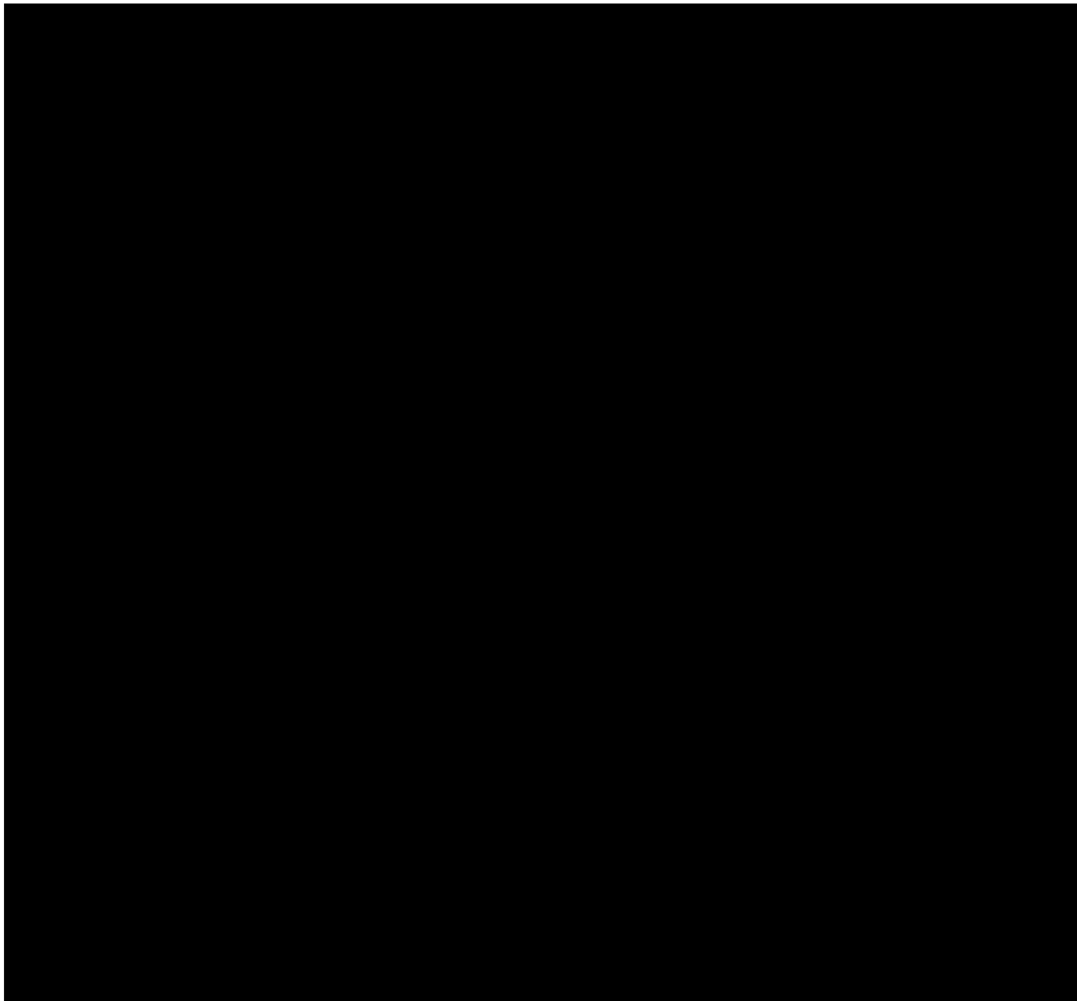
- A real time FIFO task continues to run until it voluntarily yields the processor, blocks or is preempted by a higher-priority real-time task
  - no timeslices
  - all other tasks of lower priority will not be scheduled until it relinquishes the CPU
  - two equal-priority Real time FIFO tasks do not preempt each other

Real Time Scheduling in Linux

- SCHED_RR is similar to SCHED_FIFO, except that such tasks are allotted timeslices based on their priority and run until they exhaust their timeslice
- Non-real time tasks continue to use CFS algorithm
- SCHED_DEADLINE uses an Earliest Deadline First algorithm to schedule each task.

Multi-Core Scheduling

- Scheduling over multiple processors or cores is a new challenge.
    - A single CPU/processor may support multiple cores
- Variety of multi-core schedulers being tried. We'll just mention some design themes.
- In *asymmetric multiprocessing* - one CPU handles all scheduling, decides which processes run on which cores



Multi-Core Scheduling

- Scheduling over multiple processors or cores is a new challenge.
  - A single CPU/processor may support multiple cores
- Variety of multi-core schedulers being tried. We'll just mention some design themes.
- In *asymmetric multiprocessing* - one CPU handles all scheduling, decides which processes run on which cores
- In symmetric multi-processing (SMP), each core is self-scheduling.

Multi-Core Scheduling

- In symmetric multi-processing (SMP), each core is self-scheduling.
- Another self-scheduling SMP approach is when each core has its own ready queue
- Most modern OSs support this paradigm
- Caching is important to consider
- To maximally exploit caching, tasks tend to stick to a given core/processor processor affinity
    - In hard affinity, a process specifies via a system call that it insists on staying on a given CPU core
    - In soft affinity, there is still a bias to stick to a CPU core, but processes can on occasion migrate.
    - Linux supports both

Multi-Core Scheduling

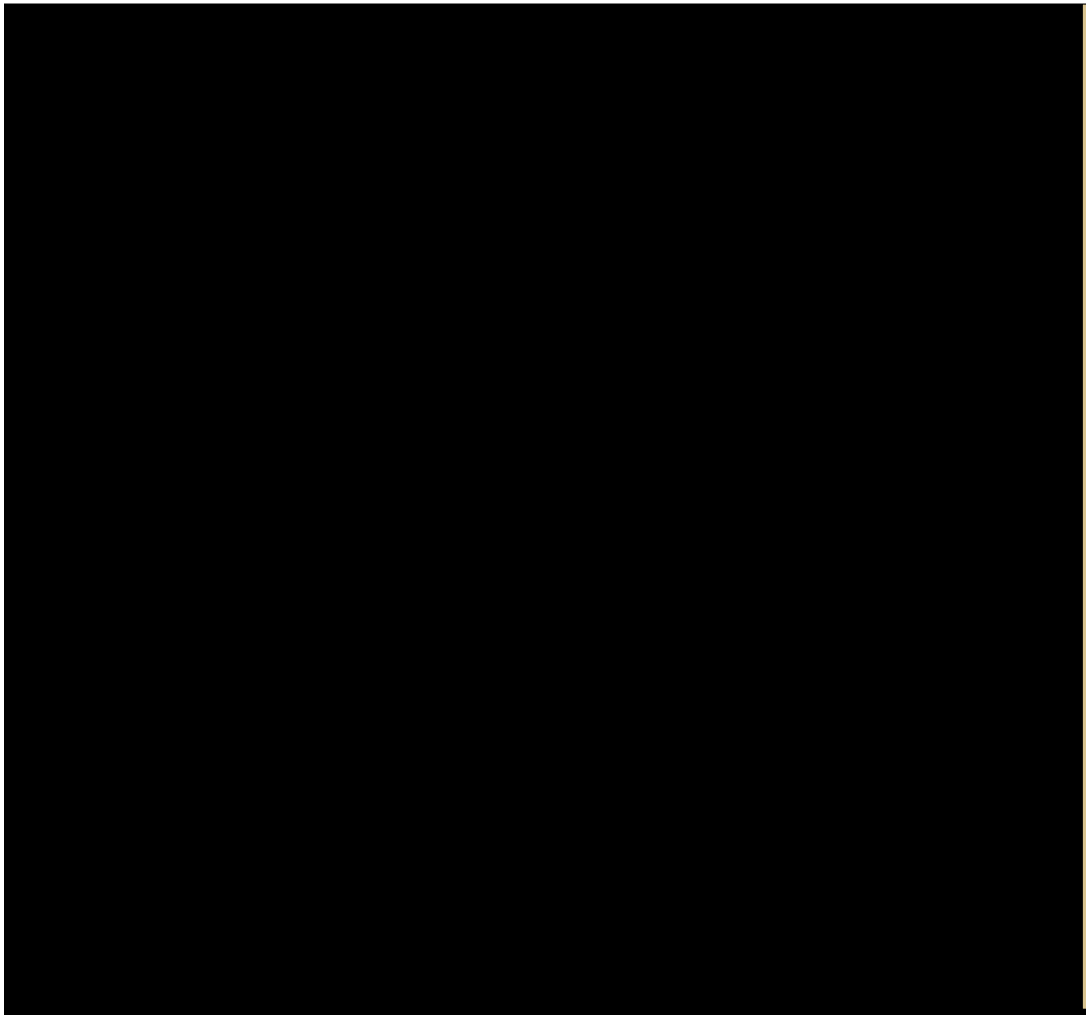
Load balancing

- Goal: Keep workload evenly distributed across cores
- Otherwise, some cores will be under-utilized.
- When there is a single shared ready queue, there is automatic load balancing
- Cores pull in processes from the ready queue whenever they're idle.
- push migration – a dedicated task periodically checks the load on each core, and if imbalance, pushes processes from more-loaded to less-loaded cores
- Pull migration – whenever a core is idle, it tries to pull a process from a neighboring core
- Linux and FreeBSD use a combination of pull and push

Multi-Core Scheduling

Load balancing

- Goal: Keep workload evenly distributed across cores
- Otherwise, some cores will be under-utilized.
- When there is a single shared ready queue, there is automatic load balancing
- Cores pull in processes from the ready queue whenever they’re idle.
- push migration – a dedicated task periodically checks the load on each core, and if imbalance, pushes processes from more-loaded to less-loaded cores
- Pull migration – whenever a core is idle, it tries to pull a process from a neighboring core
- Linux and FreeBSD use a combination of pull and push



Multi-Core Scheduling

Load balancing

- Load balancing can conflict with caching
  - Push/pull migration causes caches to be rebuilt
- Load balancing can conflict with power management
  - Mobile devices typically want to save power
  - One approach is to power down unused cores
  - Load balancing would keep as many cores active as possible, thereby consuming power
- In systems, often conflicting design goals

Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder

These slides adapted from materials provided by the textbook authors.