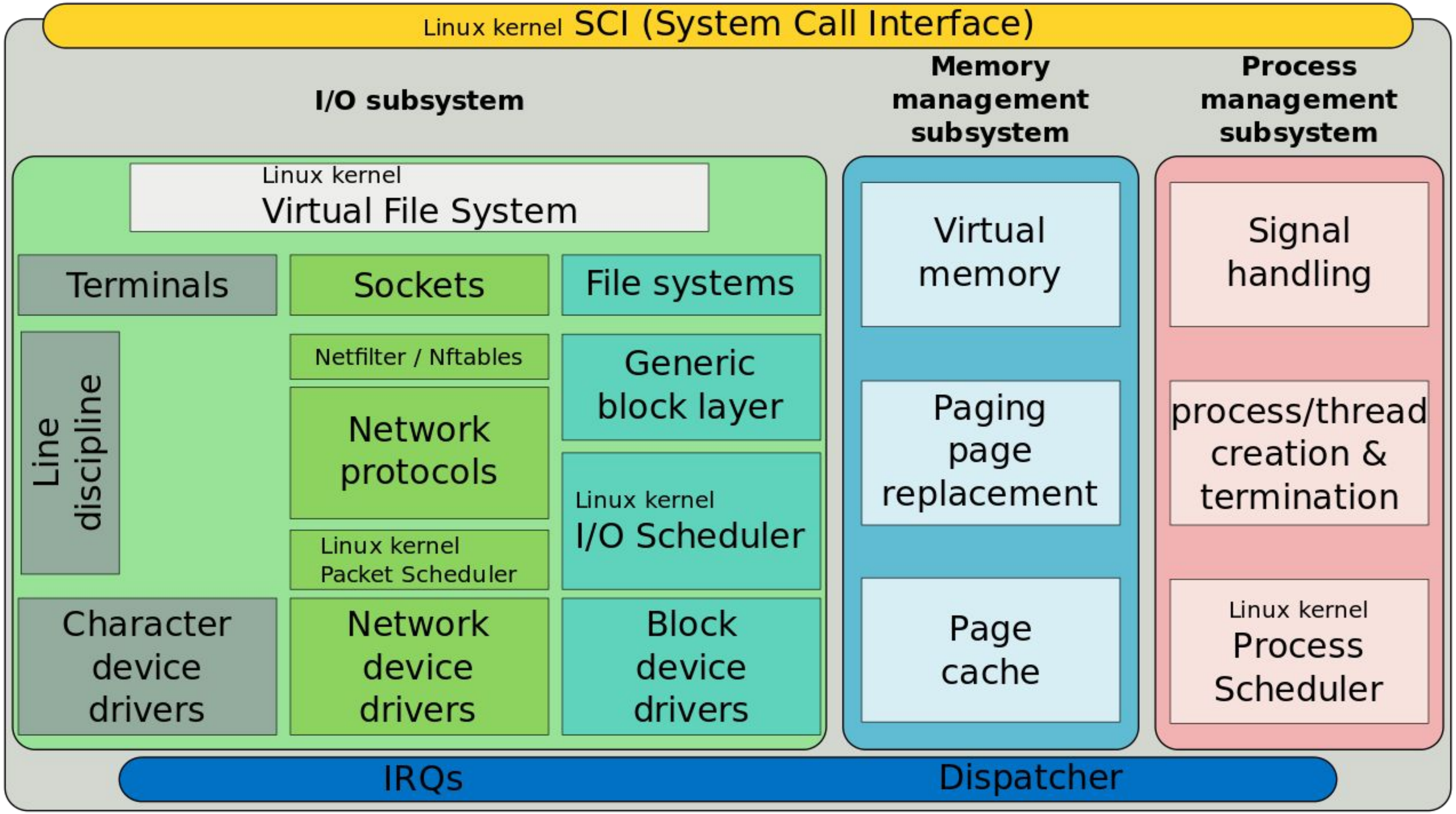
Linux and Real-Time Operation

ESE3025

further reading

- Chapter 11 of course text, Siewart and Pratt
- read about the fascinating Linux Kernal and its Completely Fair Scheduler, history and development:
    - https://www.kernel.org/doc/html/latest/
    - https://www.kernel.org/doc/html/latest/scheduler/sched-design-CFS.html
    - https://en.wikipedia.org/wiki/Completely_Fair_Scheduler



source: https://en.wikipedia.org/wiki/Completely_Fair_Scheduler

Linux Kernel Operation

- a typical Linux system operates with many active processes, threads, and interrupts, all competing for access to the kernel (via system calls) and kernel memory
- access to kernel memory is a kind of “bottleneck” because it is a protected resource
    - a mechanism, like a mutex or context-switch prevention (critical section), must be used
- for multi-core systems, the Linux kernel has used a number of strategies to handle access to kernel memory, from disabling interrupts, a single shared mutex (known as the Big Kernel Lock), to the use of spin locks

BKL

- advantages:
    - relatively simple approach
    - protects kernel memory from simultaneous access on different processor cores
    - straightforward extension of disabling interrupts (which worked well for Linux on single-core machines)
- disadvantages:
    - disables interrupts (which are used, for example, in real-time systems for timing!)
    - puts threads to sleep if the BKL is not available
    - if a thread holding the BKL goes to sleep (like block state) it gives up the BKL, which it needs when it wakes up!

spin locks

- places a lock on individual system resources
- advantages:
  - allows fine-grained control of system resources, rather than having a single large mutex
  - threads attempting to take the lock/mutex are not put to sleep
  - accommodates interrupts
- disadvantages:
  - code holding a spin lock must not sleep and complete the critical section as quickly as possible
  - any thread or interrupt attempting to access the spin lock will keep cores busy, leaving them unavailable for other activities
  - spin locks cannot be used recursively (threads calling themselves, for example, won’t be able to get the lock)

PREEMPT_RT patch

- uses semaphores instead of spin locks
- has interrupts run as higher-priority threads
- introduces a specialized system call for critical sections dealing with scheduler resources
- introduces “sleeping spin locks”
- overall advantages:
  - allows blocked processes to go to sleep (semaphore keeps track of things)
  - interrupts can also sleep
  - worst-case timing is more tightly bounded
- overall disadvantages:
  - higher overhead
  - cannot be considered a hard real-time solution (although still useful for soft real time)