Parallel computation

Types of parallel computations

Plan

0. Parallelism that does not require programmer intervention
1. SIMD
2. Thread-level concurrency
3. Distributed computing
4. Hardware acceleration

0. Parallelism that does not require programmer intervention

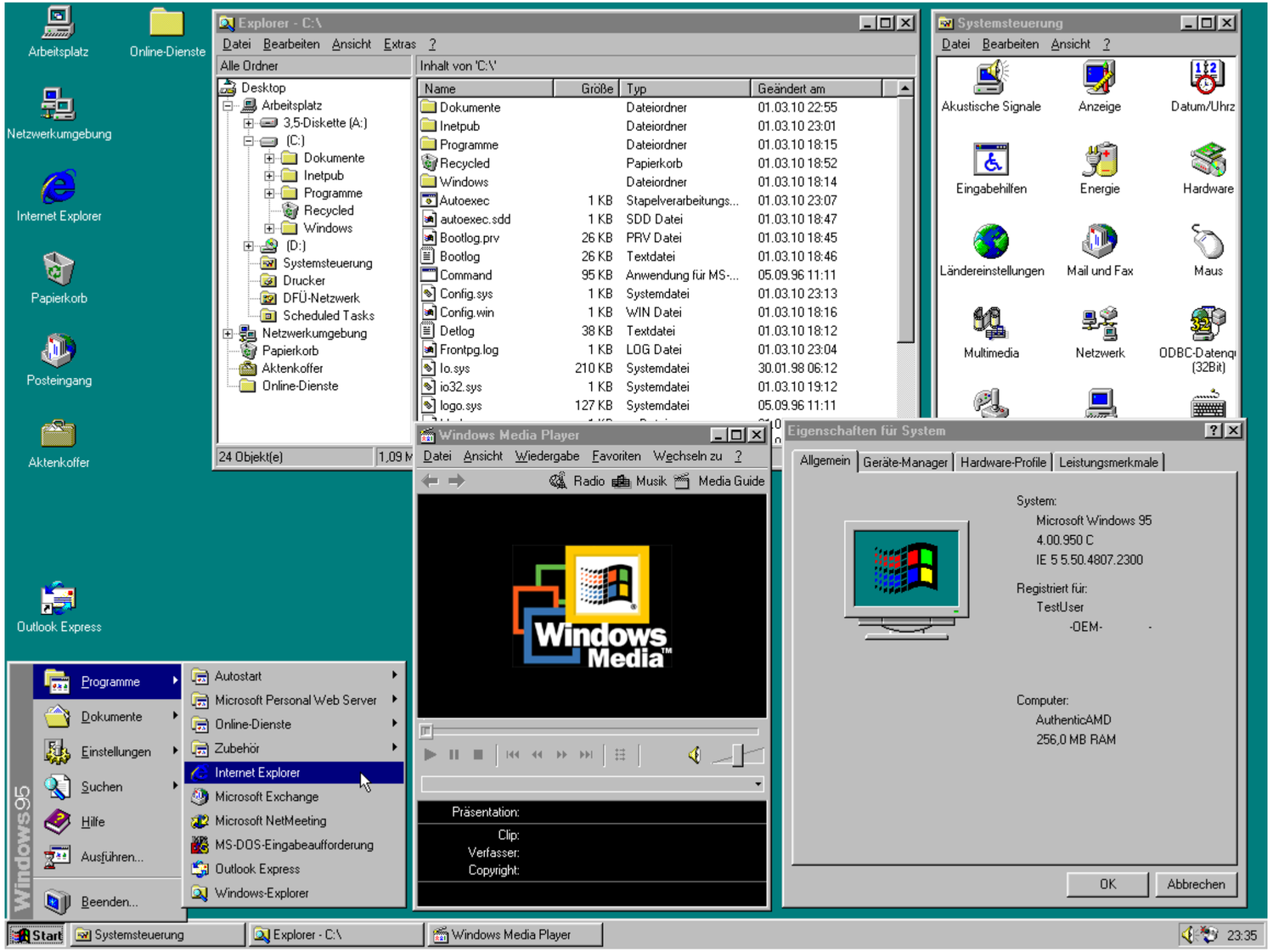
Pipelines

- CPU pipelines can be viewed as implementing some form of parallelism
  in the sense that multiple executions are being executed simultaneously

- For example, one instruction's arithmetic is performed (in an ALU)
  while the next is being decoded

- However, from the programmer's perspective,
  everything must happen **as if** there was no parallelism at all

Multitasking

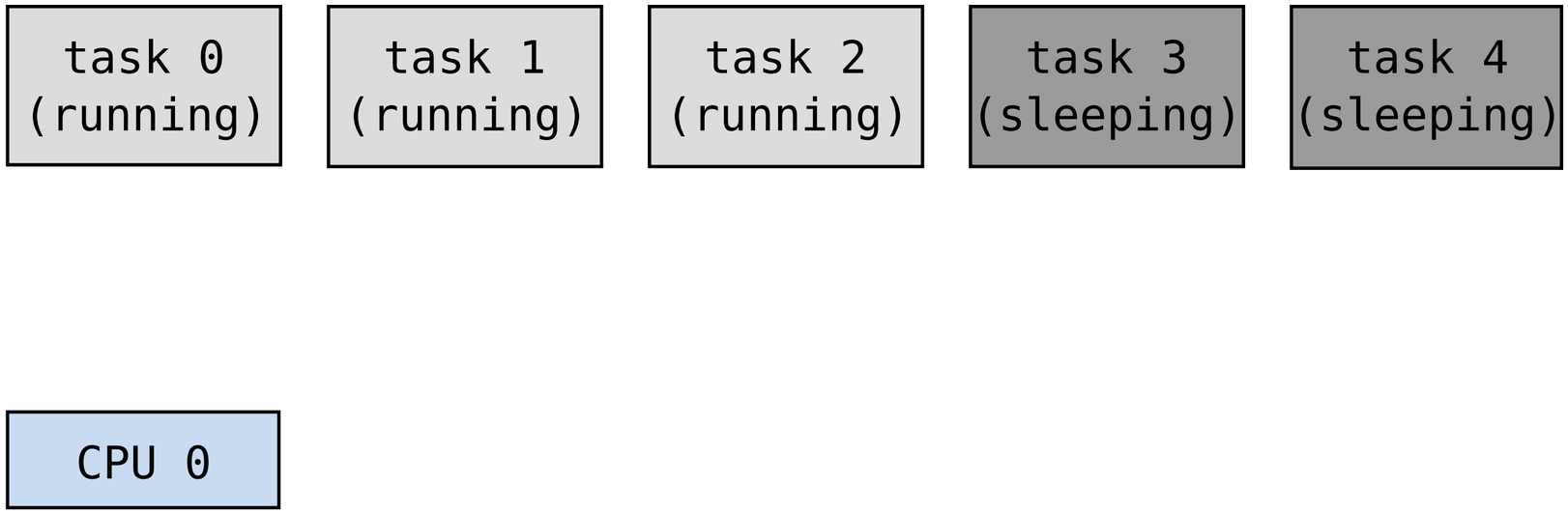


Multitasking

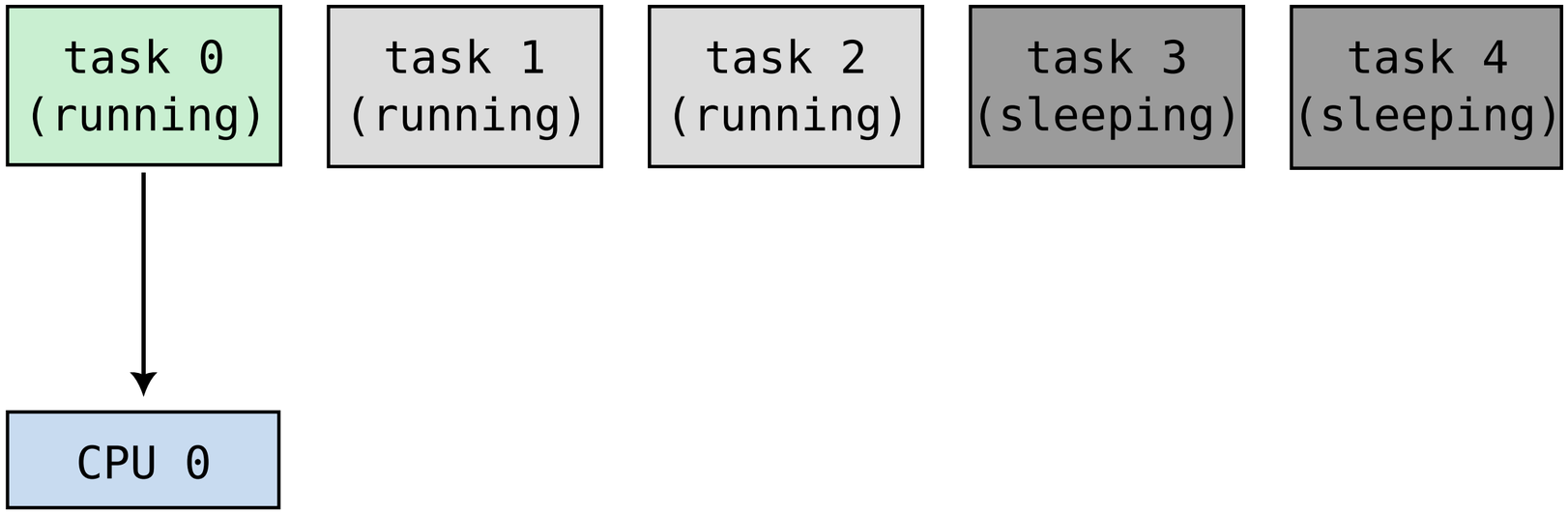
- Multitasking allows multiple executables to run “simultaneously”
  (even on a single processor)

- Regularly, the **scheduler** (part of the OS kernel) decides which **task** gets to run on a processor.

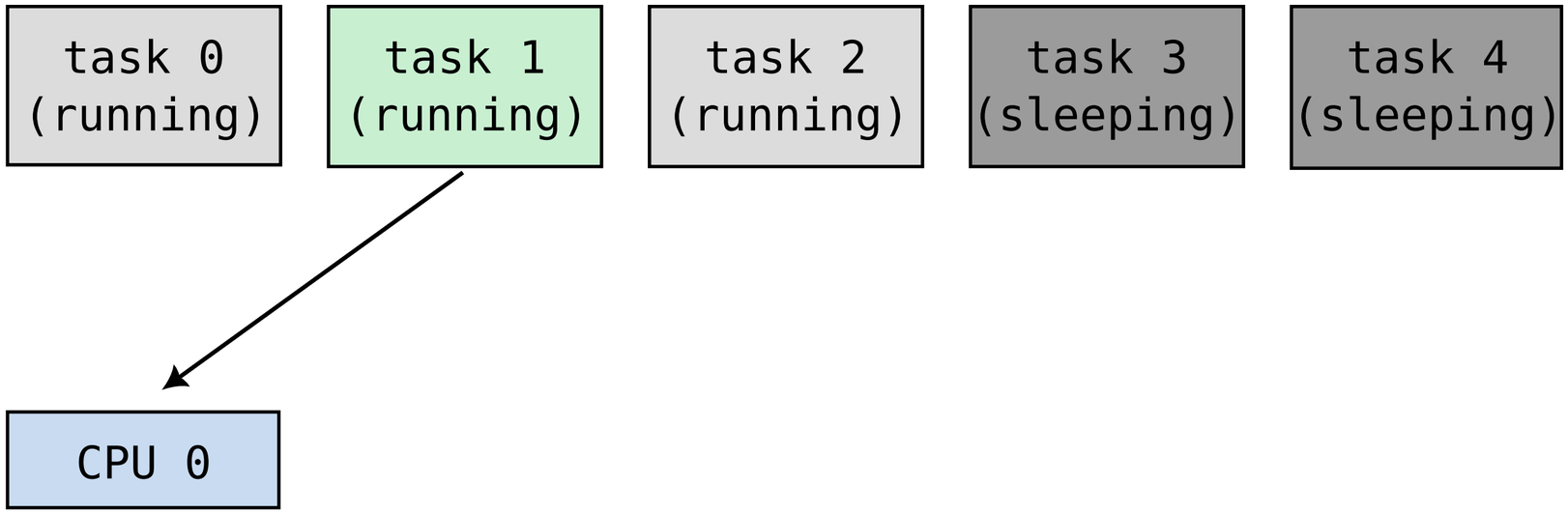
Multitasking on a single-core processor



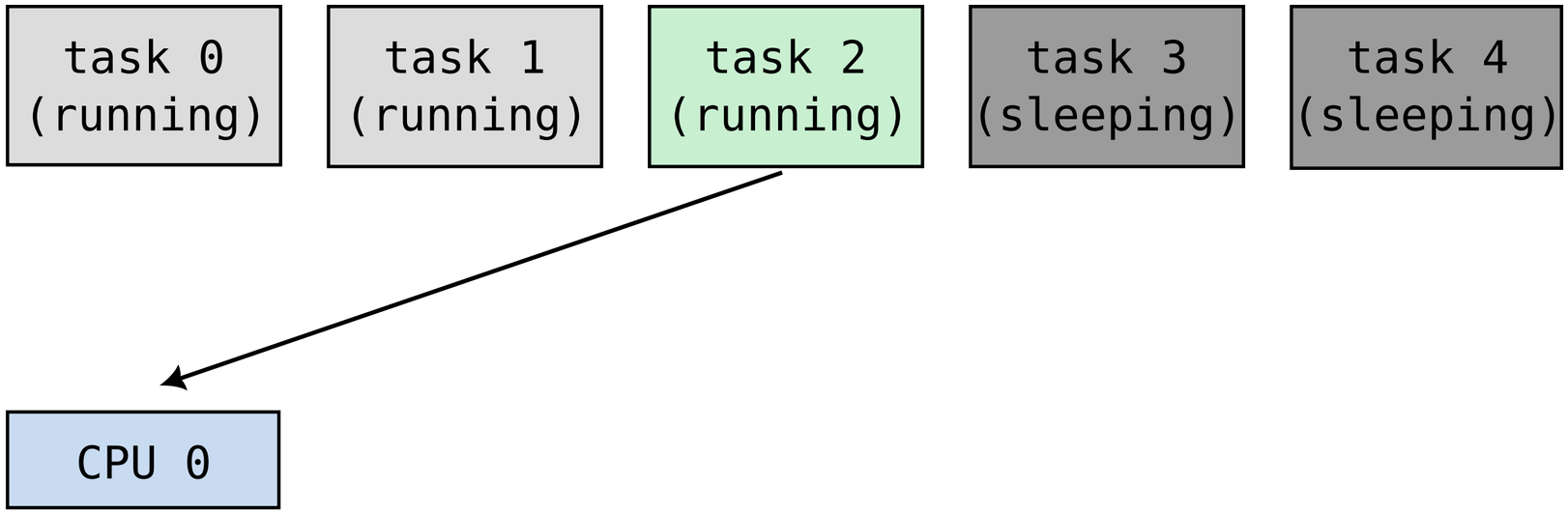
Multitasking on a single-core processor



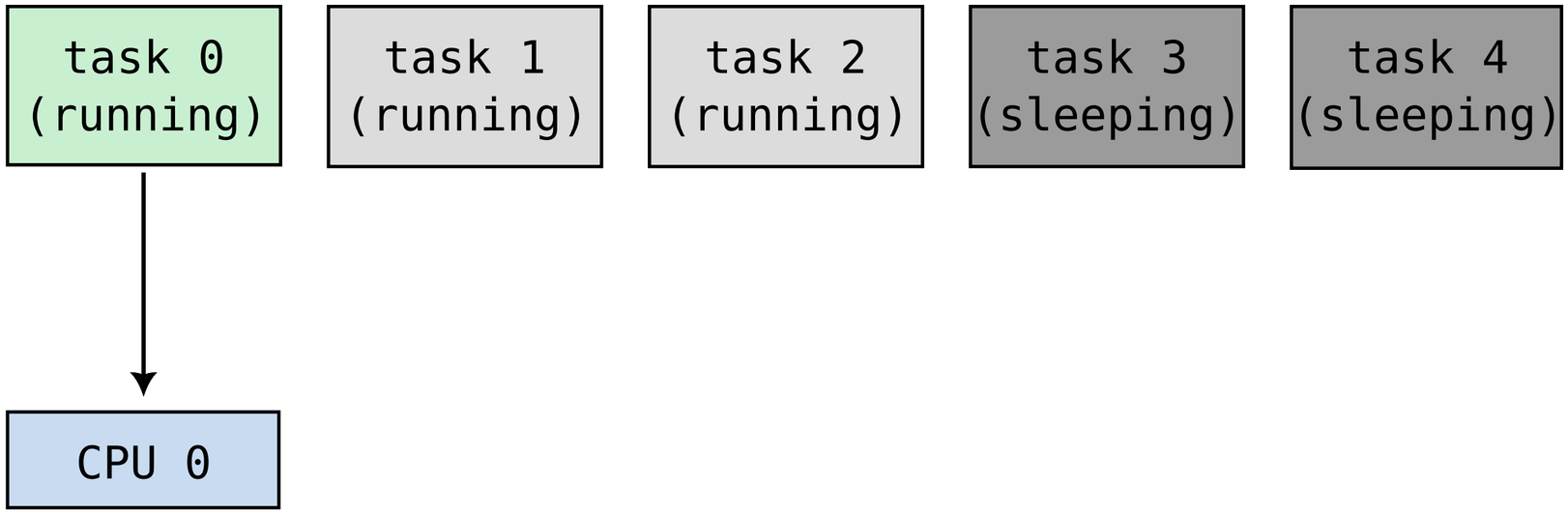
Multitasking on a single-core processor



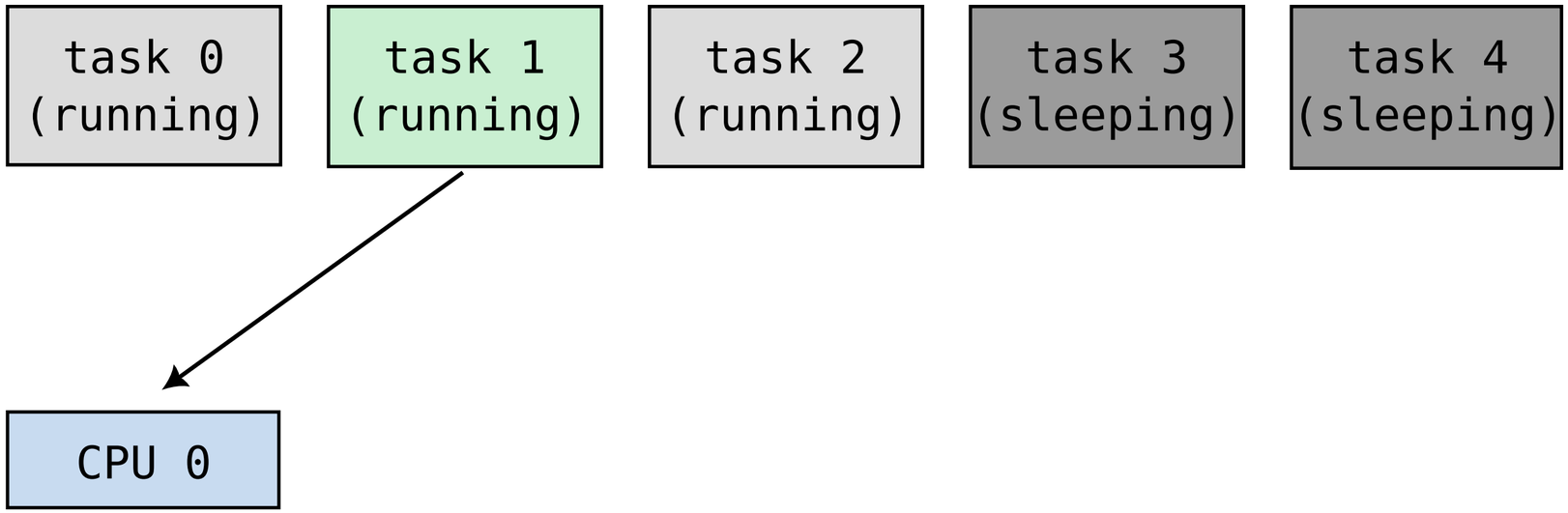
Multitasking on a single-core processor



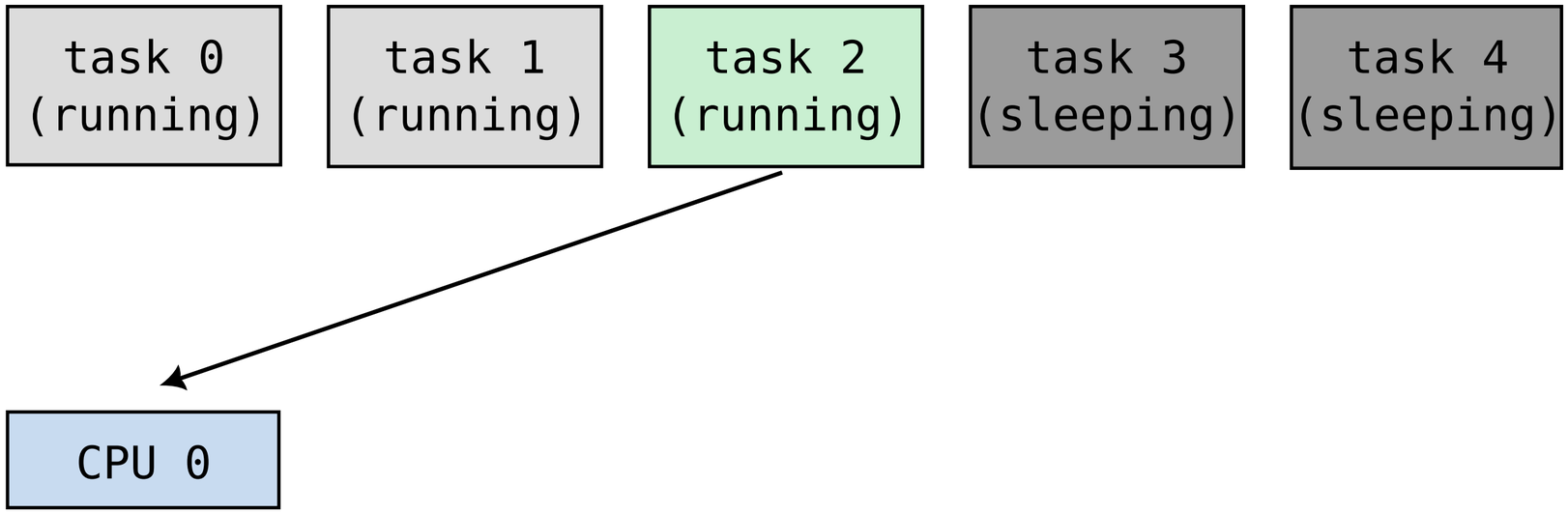
Multitasking on a single-core processor



Multitasking on a single-core processor



Multitasking on a single-core processor



- The scheduler is called:
  - at regular intervals f times per second, by default:
    - Linux: f = 1000 Hz (> see CONFIG_HZ)
    - MacOS: f = 100 Hz (> see sysctl kern.clockrate)
    - Windows 10: f = 64 Hz (> see timeBeginPeriod())
  - when an task performs a system call (`open()`, `write()`, `exit()`, ...)
  - when a “hardware interrupt” happens:
    - keyboard received a keypress
    - network device received data
    - storage device finished writing
    - sound/video device ready to receive next buffer
    - …

Preemptive multitasking

- When the scheduler decides to interrupt a **running** process (e.g. to run another)
  - the process is said to "**preempted**"
  - it becomes "**runnable**"

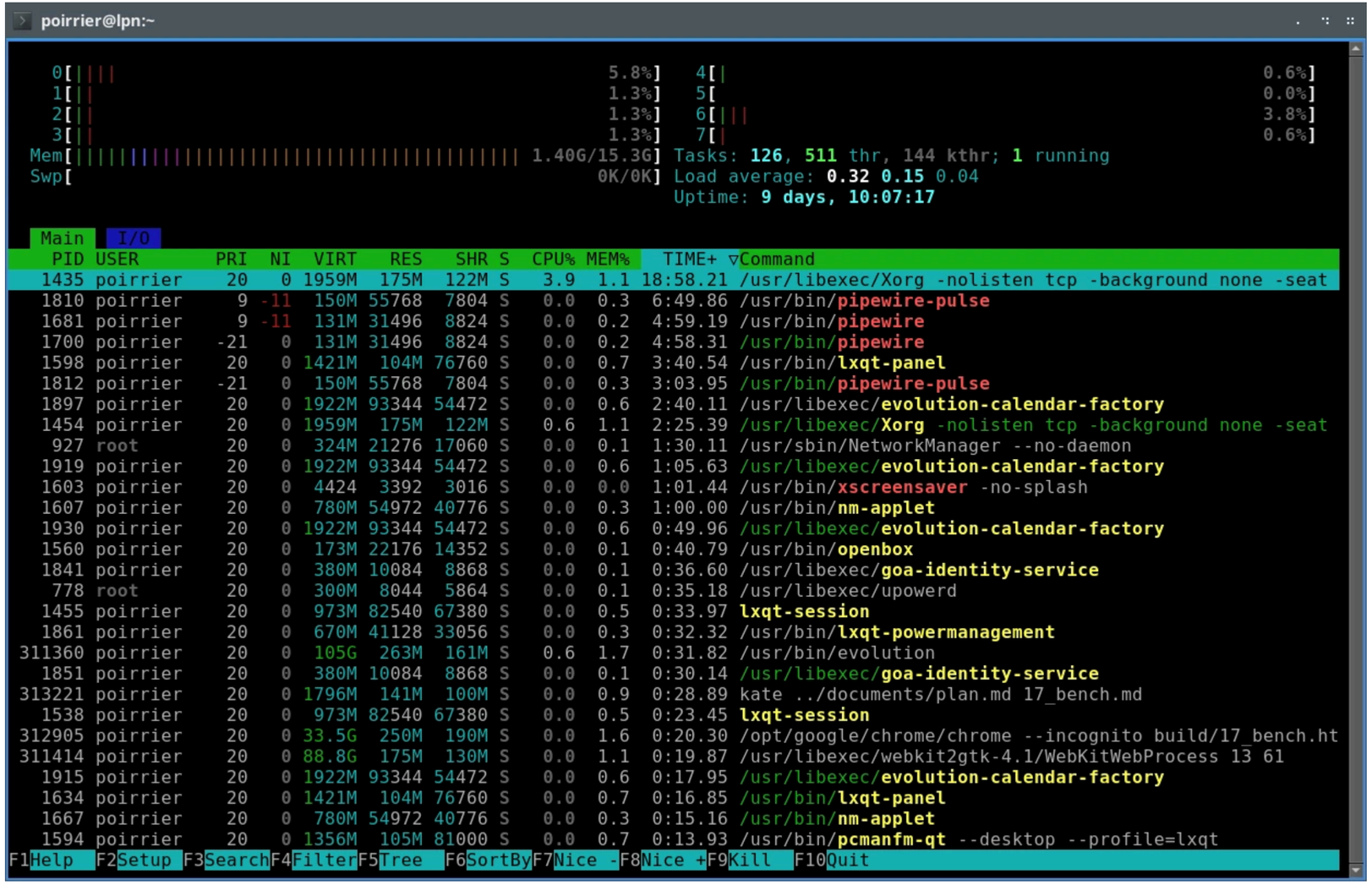
- When a process executes a system call,
  - it starts "**sleeping**"
  - after the requested operation is performed,
    - in some cases, it will **run** again
    - in other cases, it becomes **runnable** and will only run when a CPU is available
  - many system calls can take a long time to perform ("**blocking**" system calls): `read()`, `write()`, `recv()`, `send()`

Preemptive multitasking

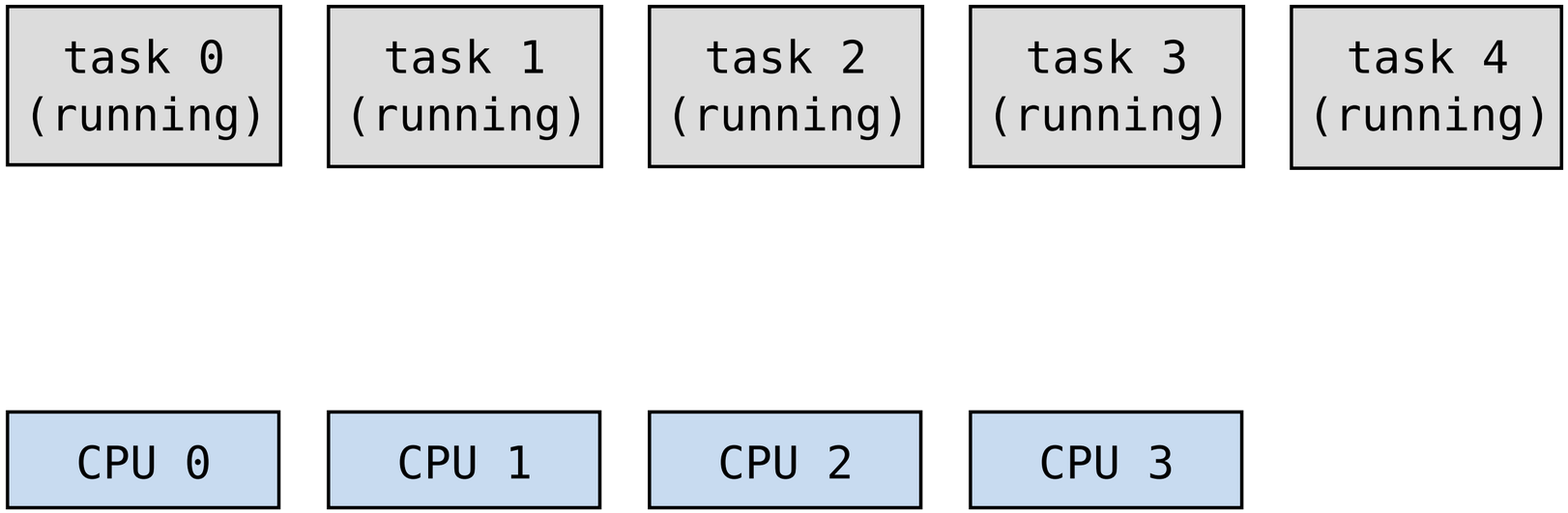
- At any given time, most tasks are **sleeping**
  - waiting for data (e.g. from network)
  - waiting for user interaction (e.g. keyboard or touch input)
  - waiting on a timer (tasks that run at regular interval)

- The only tasks that are normally **running/runnable**

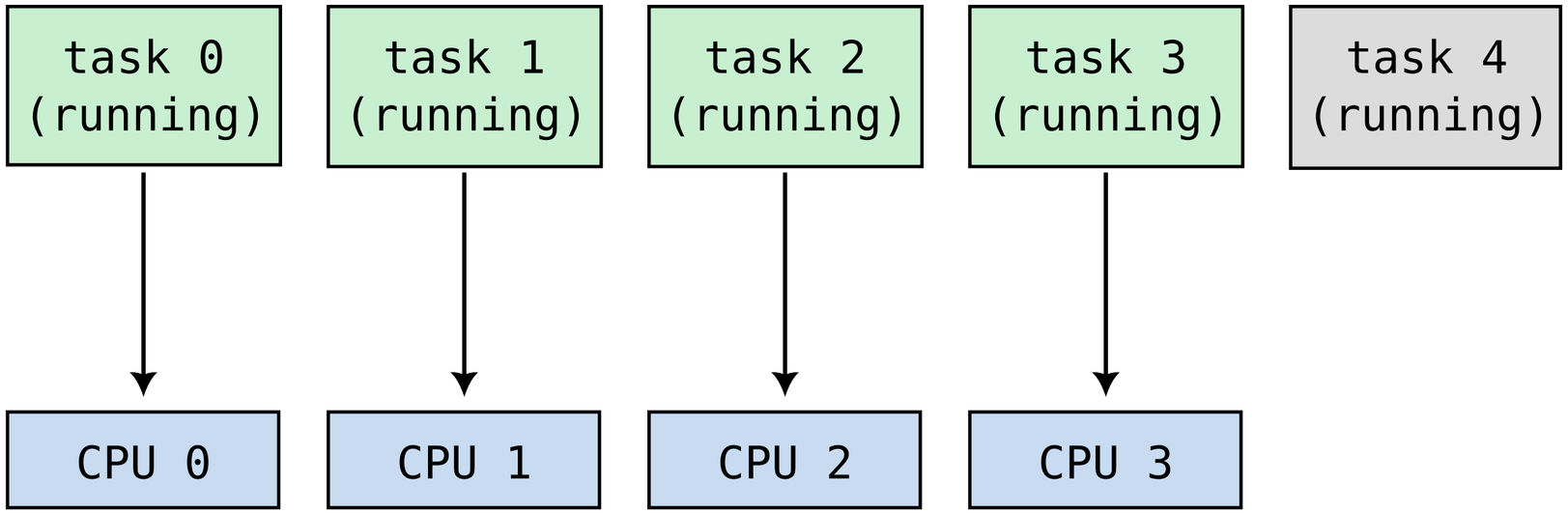
  are those performing CPU-intensive operations
  - graphics rendering
  - audio/video/data compression and decompression
  - computations
  - etc.


poirrier@lpn:~
0[|||| 5.8%] 4[| 0.6%]
1[|| 1.3%] 5[0.0%]
2[|| 1.3%] 6[||| 3.8%]
3[|| 1.3%] 7[| 0.6%]
Mem[|||||||||||||||||||||||||||||||| 1.40G/15.3G] Tasks: 126, 511 thr, 144 kthr; 1 running
Swp[0K/0K] Load average: 0.32 0.15 0.04
Uptime: 9 days, 10:07:17
Main I/O
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ ▽Command
1435 poirrier 20 0 1959M 175M 122M S 3.9 1.1 18:58.21 /usr/libexec/Xorg -nolisten tcp -background none -seat
1810 poirrier 9 -11 150M 55768 7804 S 0.0 0.3 6:49.86 /usr/bin/pipewire-pulse
1681 poirrier 9 -11 131M 31496 8824 S 0.0 0.2 4:59.19 /usr/bin/pipewire
1700 poirrier -21 0 131M 31496 8824 S 0.0 0.2 4:58.31 /usr/bin/pipewire
1598 poirrier 20 0 1421M 104M 76760 S 0.0 0.7 3:40.54 /usr/bin/lxqt-panel
1812 poirrier -21 0 150M 55768 7804 S 0.0 0.3 3:03.95 /usr/bin/pipewire-pulse
1897 poirrier 20 0 1922M 93344 54472 S 0.0 0.6 2:40.11 /usr/libexec/evolution-calendar-factory
1454 poirrier 20 0 1959M 175M 122M S 0.6 1.1 2:25.39 /usr/libexec/Xorg -nolisten tcp -background none -seat
927 root 20 0 324M 21276 17060 S 0.0 0.1 1:30.11 /usr/sbin/NetworkManager --no-daemon
1919 poirrier 20 0 1922M 93344 54472 S 0.0 0.6 1:05.63 /usr/libexec/evolution-calendar-factory
1603 poirrier 20 0 4424 3392 3016 S 0.0 0.0 1:01.44 /usr/bin/xscreensaver -no-splash
1607 poirrier 20 0 780M 54972 40776 S 0.0 0.3 1:00.00 /usr/bin/nm-applet
1930 poirrier 20 0 1922M 93344 54472 S 0.0 0.6 0:49.96 /usr/libexec/evolution-calendar-factory
1560 poirrier 20 0 173M 22176 14352 S 0.0 0.1 0:40.79 /usr/bin/openbox
1841 poirrier 20 0 380M 10084 8868 S 0.0 0.1 0:36.60 /usr/libexec/goa-identity-service
778 root 20 0 300M 8044 5864 S 0.0 0.1 0:35.18 /usr/libexec/upowerd
1455 poirrier 20 0 973M 82540 67380 S 0.0 0.5 0:33.97 lxqt-session
1861 poirrier 20 0 670M 41128 33056 S 0.0 0.3 0:32.32 /usr/bin/lxqt-powermanagement
311360 poirrier 20 0 105G 263M 161M S 0.6 1.7 0:31.82 /usr/bin/evolution
1851 poirrier 20 0 380M 10084 8868 S 0.0 0.1 0:30.14 /usr/libexec/goa-identity-service
313221 poirrier 20 0 1796M 141M 100M S 0.0 0.9 0:28.89 kate ../documents/plan.md 17_bench.md
1538 poirrier 20 0 973M 82540 67380 S 0.0 0.5 0:23.45 lxqt-session
312905 poirrier 20 0 33.5G 250M 190M S 0.0 1.6 0:20.30 /opt/google/chrome/chrome --incognito build/17_bench.ht
311414 poirrier 20 0 88.8G 175M 130M S 0.0 1.1 0:19.87 /usr/libexec/webkit2gtk-4.1/WebKitWebProcess 13 61
1915 poirrier 20 0 1922M 93344 54472 S 0.0 0.6 0:17.95 /usr/libexec/evolution-calendar-factory
1634 poirrier 20 0 1421M 104M 76760 S 0.0 0.7 0:16.85 /usr/bin/lxqt-panel
1667 poirrier 20 0 780M 54972 40776 S 0.0 0.3 0:15.16 /usr/bin/nm-applet
1594 poirrier 20 0 1356M 105M 81000 S 0.0 0.7 0:13.93 /usr/bin/pcmanfm-qt --desktop --profile=lxqt
F1Help F2Setup F3SearchF4FilterF5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit

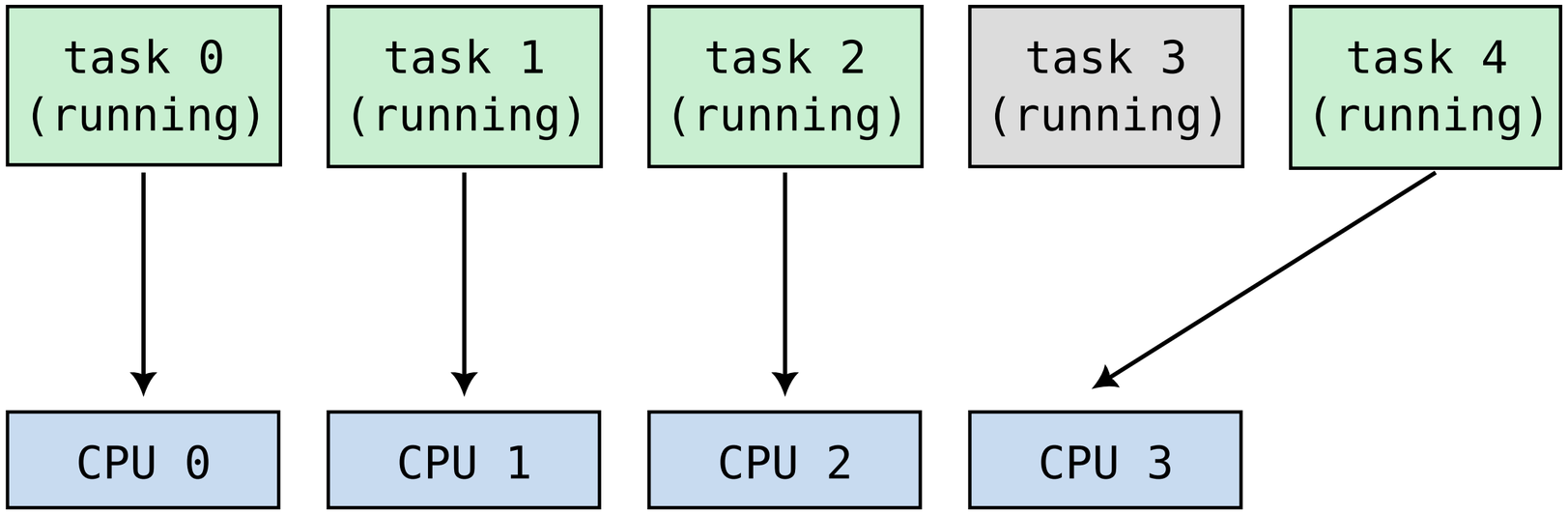
Multitasking on a multi-core processor



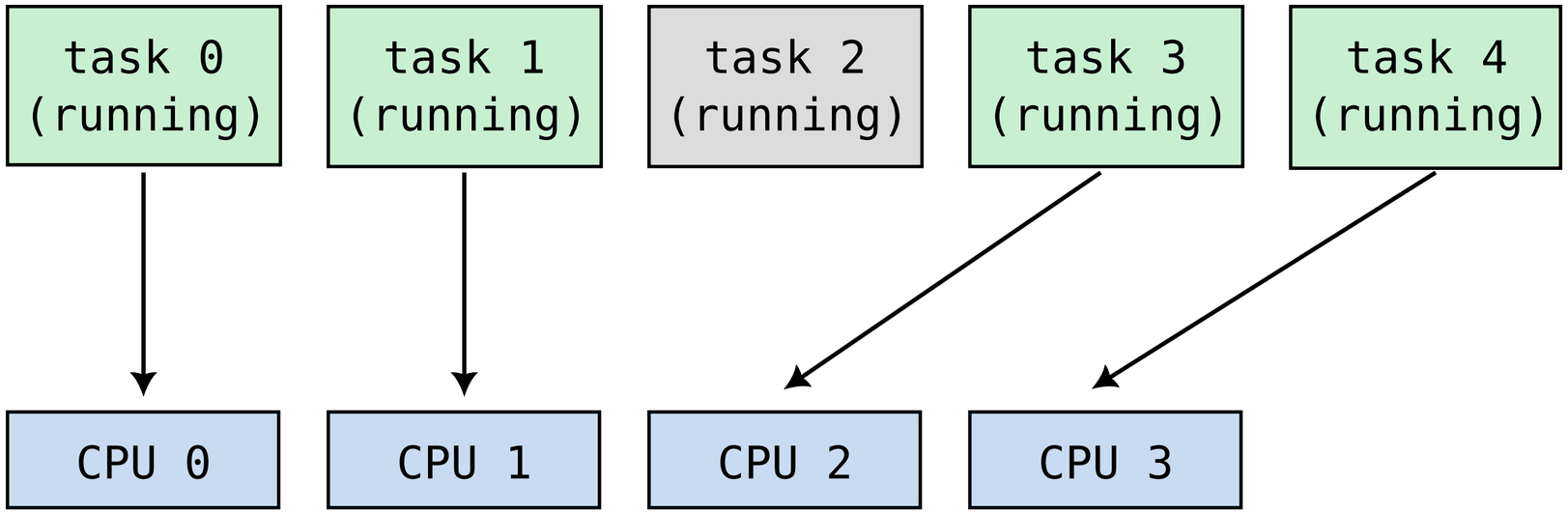
Multitasking on a multi-core processor



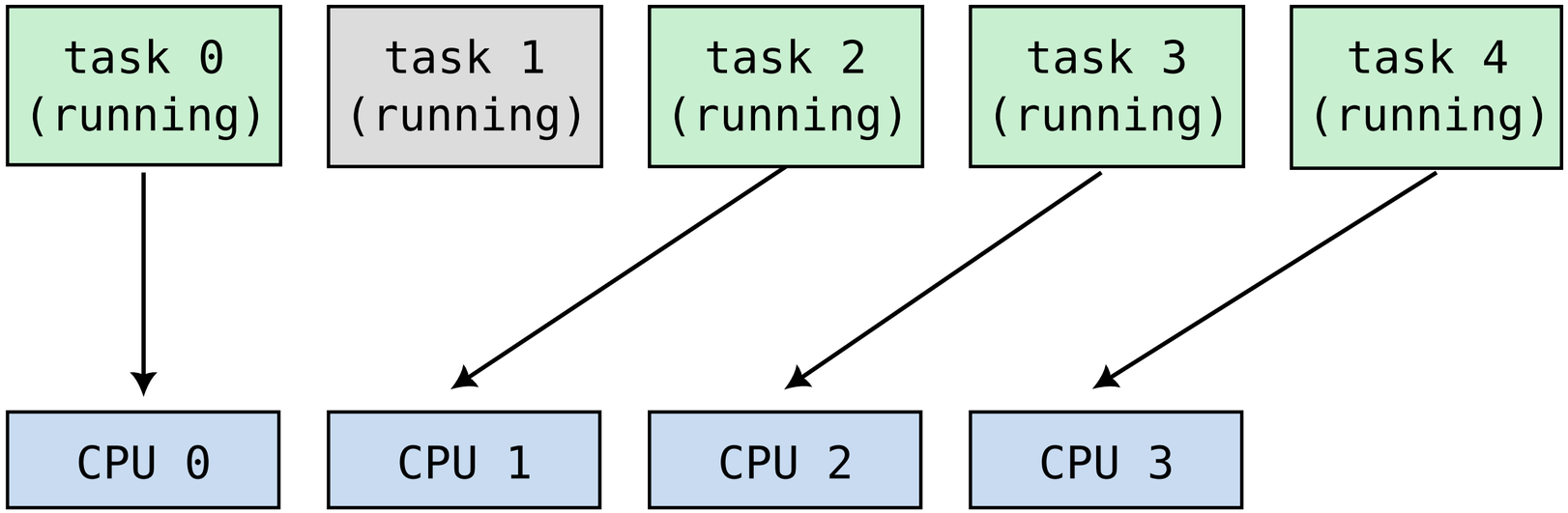
Multitasking on a multi-core processor



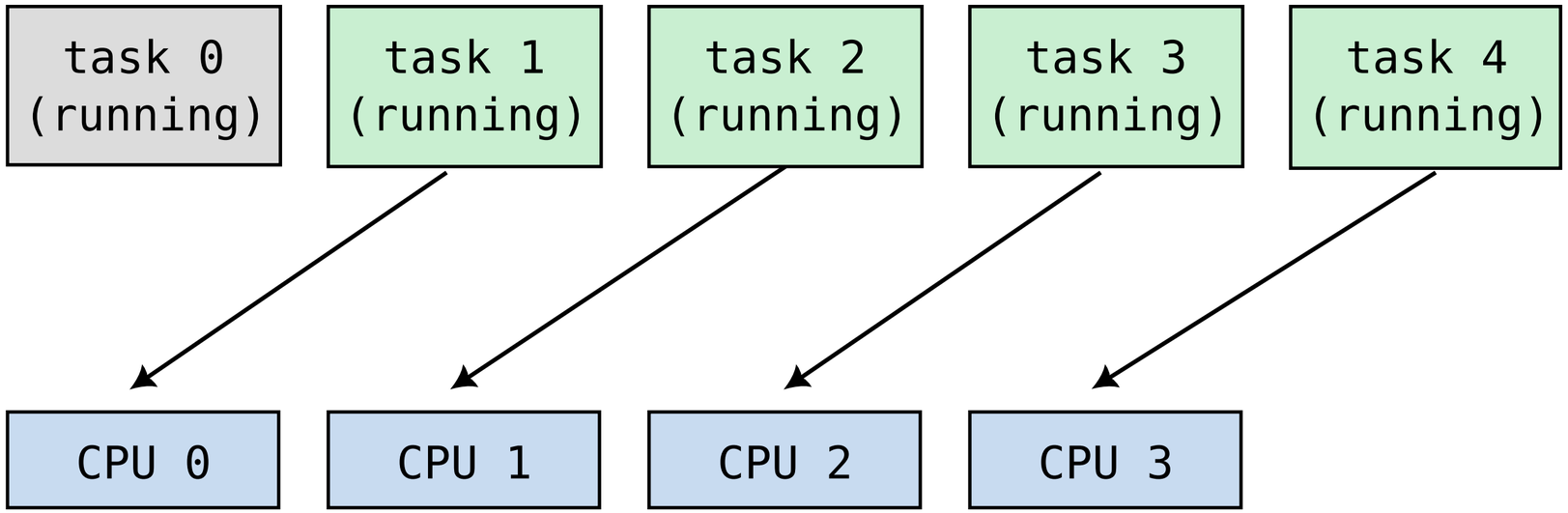
Multitasking on a multi-core processor



Multitasking on a multi-core processor



Multitasking on a multi-core processor

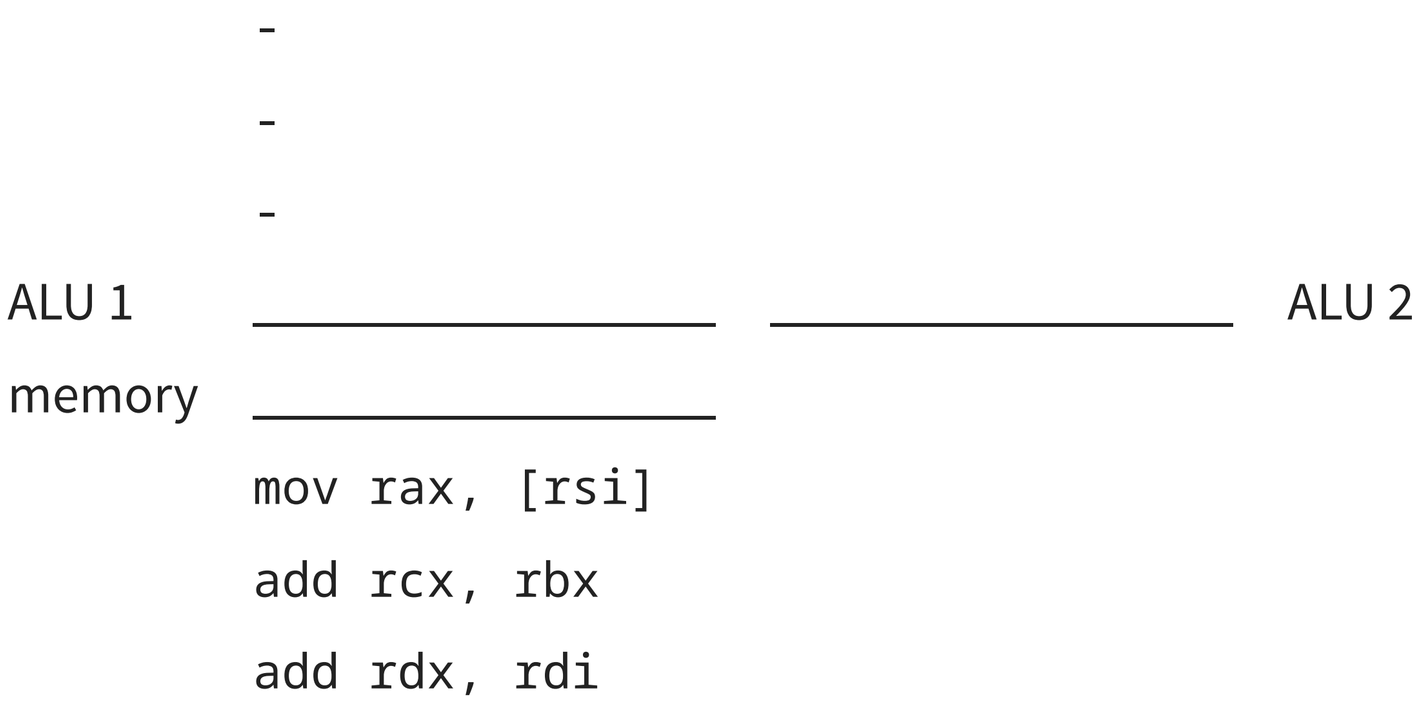


- From a hardware perspective:
  - A CPU corresponds to a single integrated circuit (“IC”) package
  - A computer can (rarely) have multiple CPUs

    Typically only found in datacenters, rarely more than 2
  - Each CPU can have multiple **cores**
    - generally 2-8 cores on laptops
    - up to 128 on datacenter CPUs

- From a software perspective:
  - Everything that can run a task is generally called a “CPU”
  - Only the kernel’s scheduler will (sometimes) care about CPU vs. core
  - All other software is unaware of the difference

- a CPU can have multiple copies of some logic blocks
- very common for arithmetic and logic units (ALUs)



Simultaneous Multithreading (SMT)

- From a hardware perspective:
  - With Simultaneous Multithreading (SMT) (a.k.a. Hyperthreading),
    - each core can run multiple (generally 2) tasks (“threads”)
    - but they share many logic blocks (in particular ALUs)
    - SMT works well when those logic blocks would otherwise be idle
    - SMT is ineffective when those logic blocks are the bottleneck

- From a software perspective:
  - Everything that can run a task is generally called a “CPU”
  - Only the kernel’s scheduler will (sometimes) care about CPU vs. core vs. **thread**
  - All other software is unaware of the difference
  - “Thread” has a different meaning in software