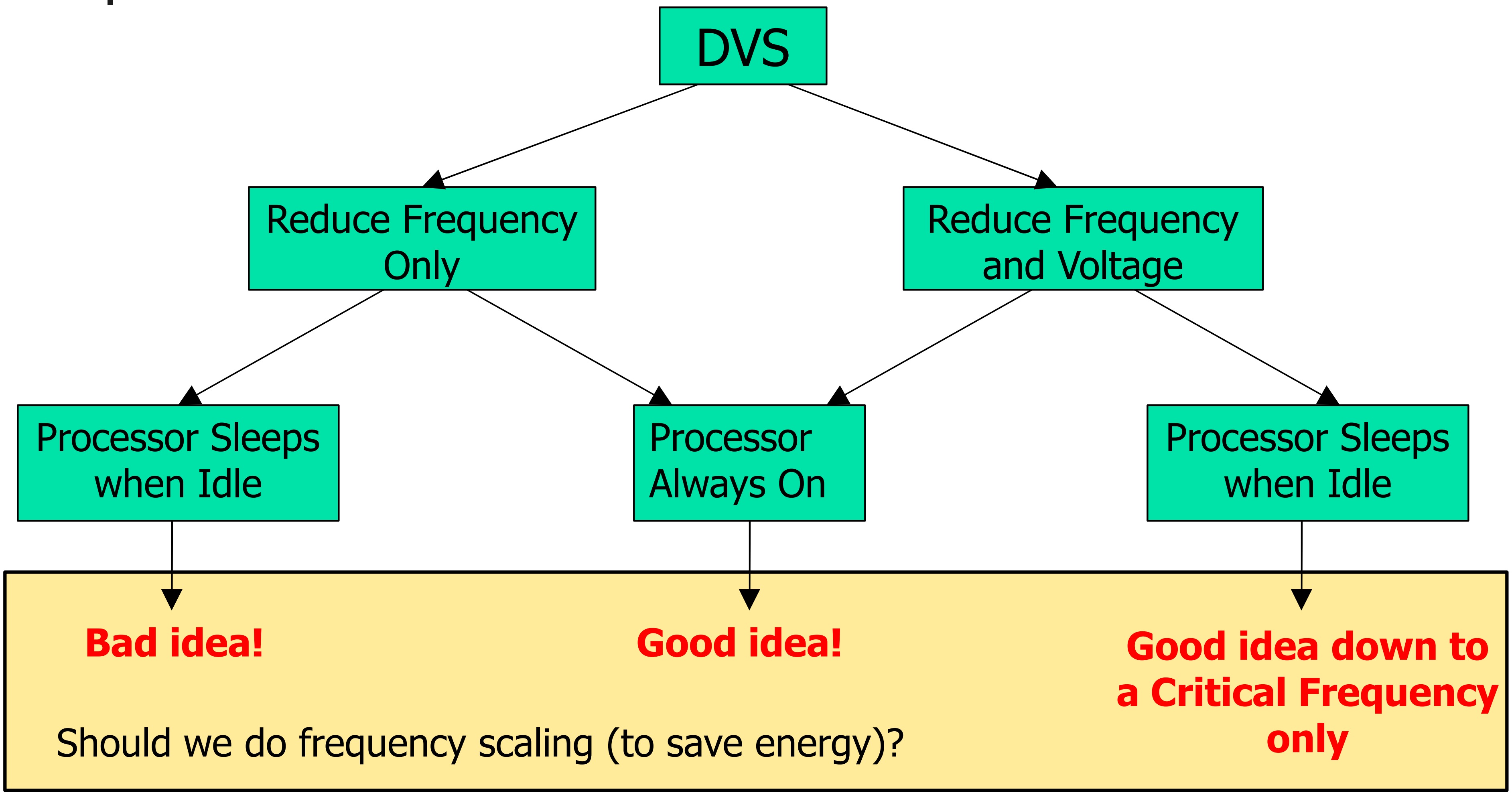
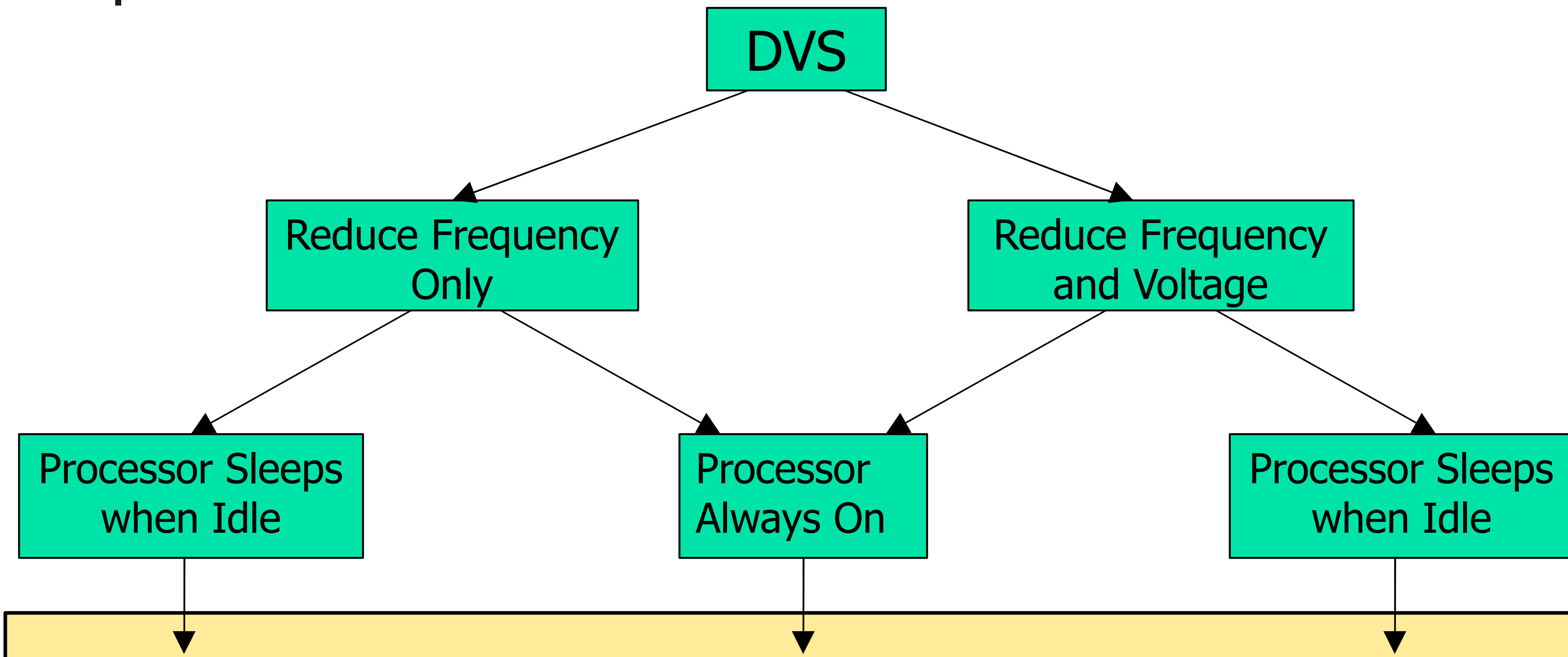
Energy

Continued

Recap


DVS
Reduce Frequency Only
Reduce Frequency and Voltage
Processor Sleeps when Idle
Processor Always On
Processor Sleeps when Idle
Bad idea!
Good idea!
Good idea down to a Critical Frequency only
Should we do frequency scaling (to save energy)?

Recap



Bad idea!

Good idea!

$$f = \sqrt[3]{\frac{R_0}{2k_f}}$$

Should we do frequency scaling (to save energy)?

Advanced Configuration and Power Interface (ACPI)

- Defines different power saving states in a platform-independent manner
- The standard was originally developed by Intel, Microsoft, and Toshiba (in 1996), then later joined by HP, and Phoenix.
- The latest version is "Revision 6.1," published by UEFI (March 2016).

Global States

- **G0**: *working*
- **G1**: *Sleeping* and *hibernation* (several degrees available)
- **G2**:, *Soft Off*: almost the same as G3 *Mechanical Off*, except that the power supply still supplies power, at a minimum, to the power button to allow wakeup. A full reboot is required.
- **G3**, *Mechanical Off*: The computer's power has been totally removed via a mechanical switch.

Processor Performance States (P-States)

- **P0** max power and frequency
- **P1** less than P0, voltage/frequency scaled
- **P2** less than P1, voltage/frequency scaled
- …
- **Pn** less than *P(n-1)*, voltage/frequency scaled

Processor “Sleep” States (C-states)

- **C0**: is the operating state.
- **C1** (often known as *Halt*): is a state where the processor is not executing instructions, but can return to an executing state instantaneously. All ACPI-conformant processors must support this power state.
- **C2** (often known as *Stop-Clock*): is a state where the processor maintains all software-visible state, but may take longer to wake up. This processor state is optional.
- **C3** (often known as *Sleep*) is a state where the processor does not need to keep its cache, but maintains other state. This processor state is optional.

Turning Processors Off
The Cost of Wakeup

- Energy expended on wakeup, E_{wake}
- To sleep or not to sleep?

Turning Processors Off
The Cost of Wakeup

- Energy expended on wakeup, E_{wake}
- To sleep or not to sleep?
  - Not to sleep (for time t):

    $$E_{no\text{-}sleep} = (k_v V^2 f + R_0)\, t$$

  - To sleep (for time t) then wake up:

    $$E_{sleep} = P_{sleep}\, t + E_{wake}$$

Turning Processors Off
The Cost of Wakeup

- Energy expended on wakeup, E_{wake}
- To sleep or not to sleep?
  - Not to sleep (for time t):

    $$E_{no\text{-}sleep} = (k_v V^2 f + R_0)\, t$$

  - To sleep (for time t) then wake up:

    $$E_{sleep} = P_{sleep}\, t + E_{wake}$$

  - To save energy by sleeping: $E_{sleep} < E_{no\text{-}sleep}$

    $$t > \frac{E_{wake}}{k_v V^2 f + R_0 - P_{sleep}}$$

Turning Processors Off
The Cost of Wakeup

- Energy expended on wakeup, E_{wake}
- To sleep or not to sleep?
  - Not to sleep (for time t):

    $$E_{no\text{-}sleep} = (k_v V^2 f + R_0)\, t$$

  - To sleep (for time t) then wake up:

    $$E_{sleep} = P_{sleep}\, t + E_{wake}$$

  - To save energy by sleeping: $E_{sleep} < E_{no\text{-}sleep}$

$$t > \frac{E_{wake}}{k_v V^2 f + R_0 - P_{sleep}}$$

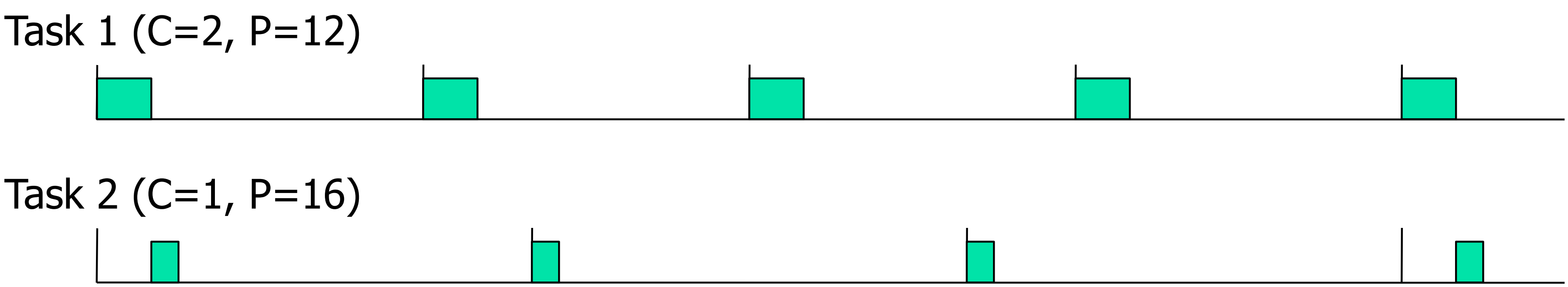
Minimum sleep interval

Dynamic Power Management

- DPM refers to turning devices off (or putting them in deep sleep modes)
- Device wakeup has a cost that imposes a minimum sleep interval (a breakeven time)
- DPM must maximize power savings due to sleep while maintaining schedulability

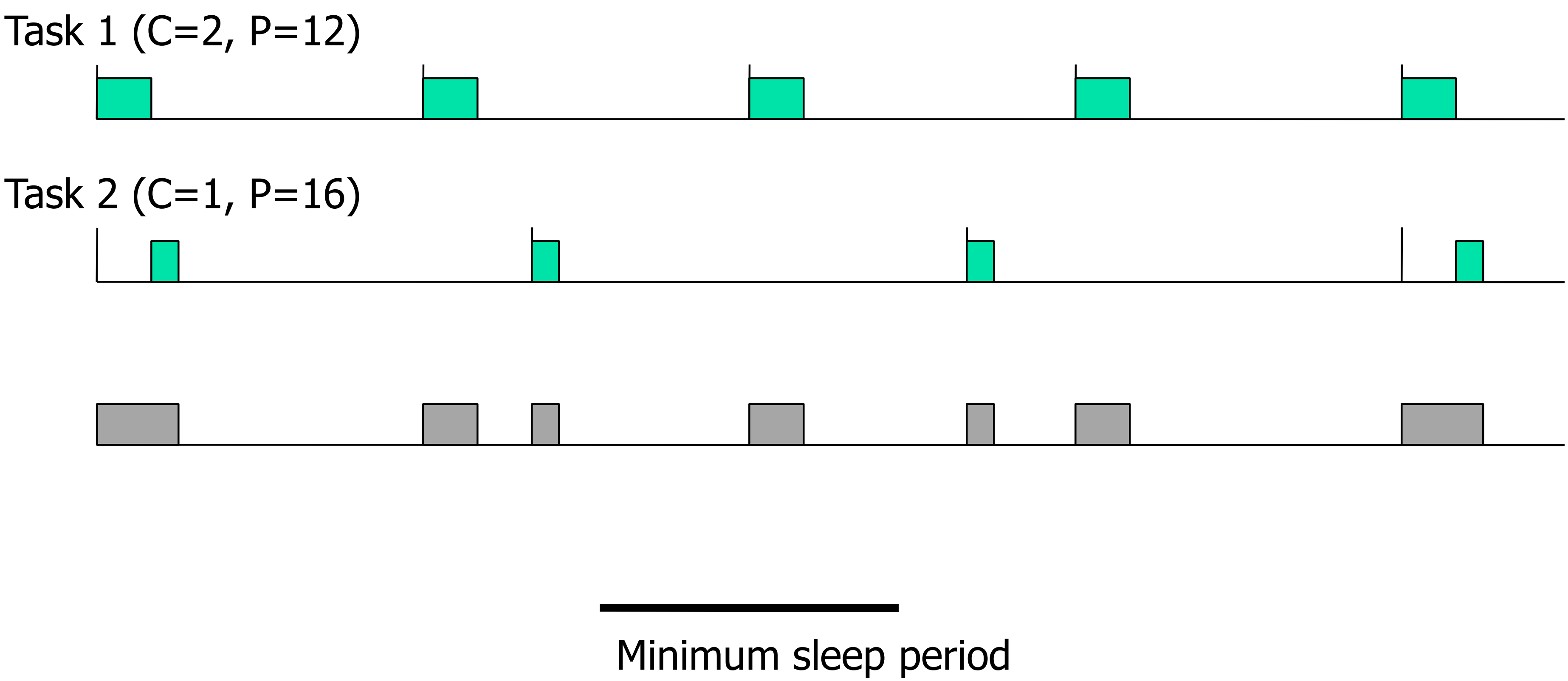
DPM and the Problem with Work-conserving Scheduling

- Example:



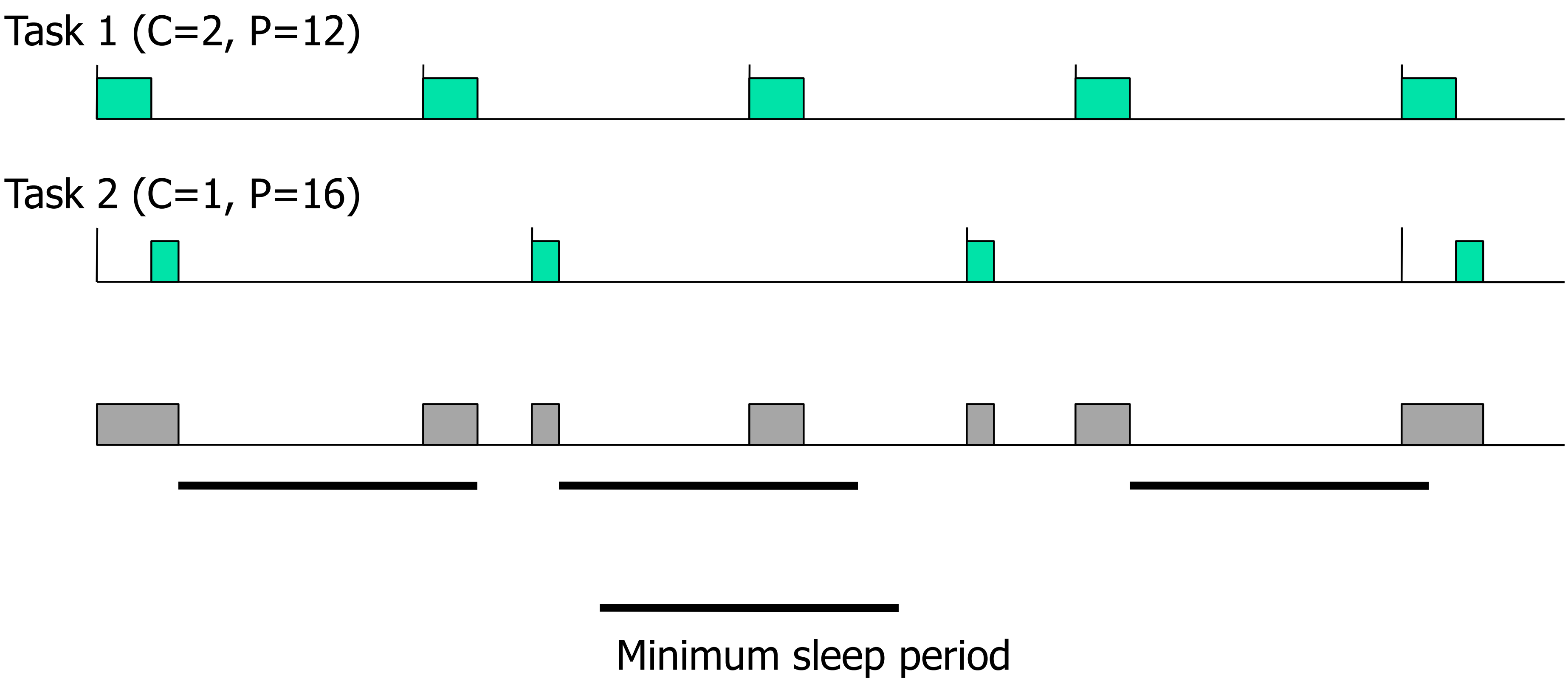
DPM and the Problem with Work-conserving Scheduling

- Example:



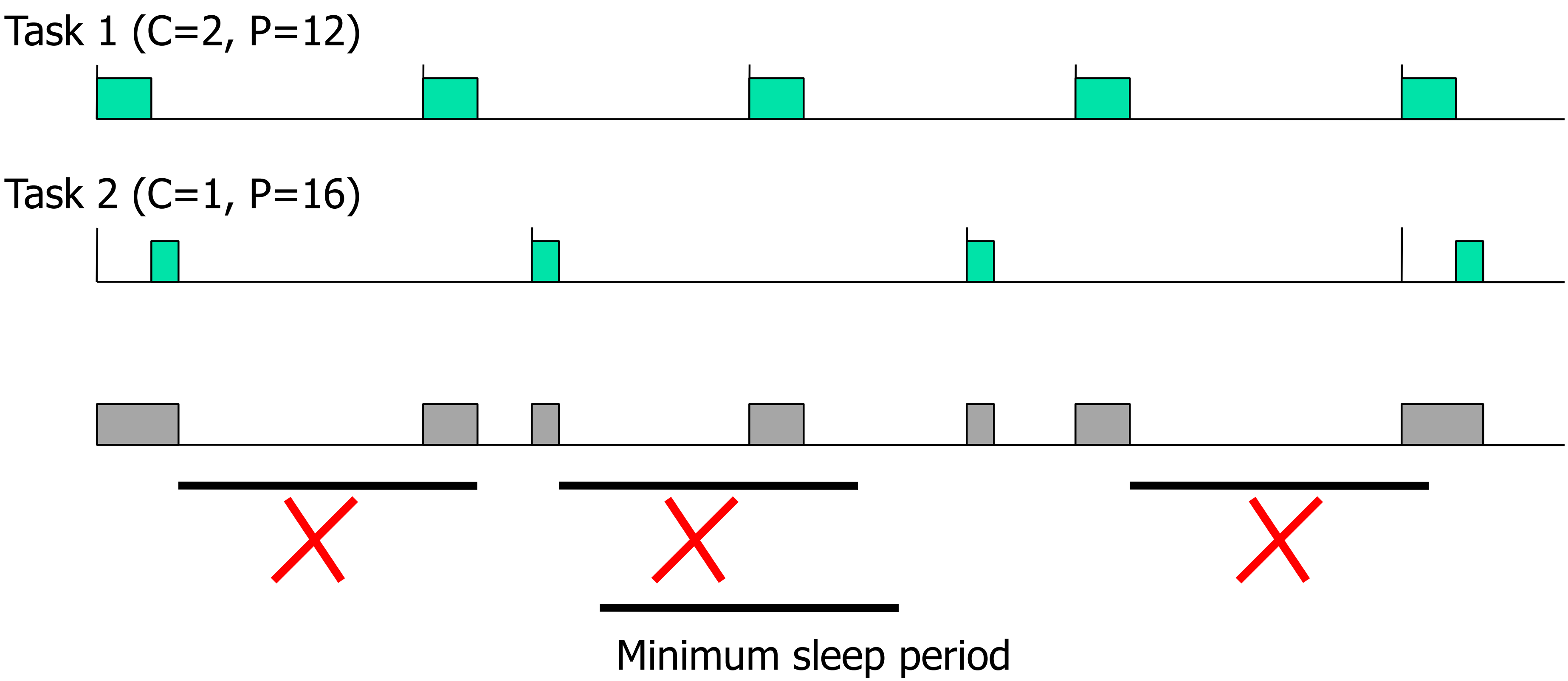
DPM and the Problem with Work-conserving Scheduling

- Example:



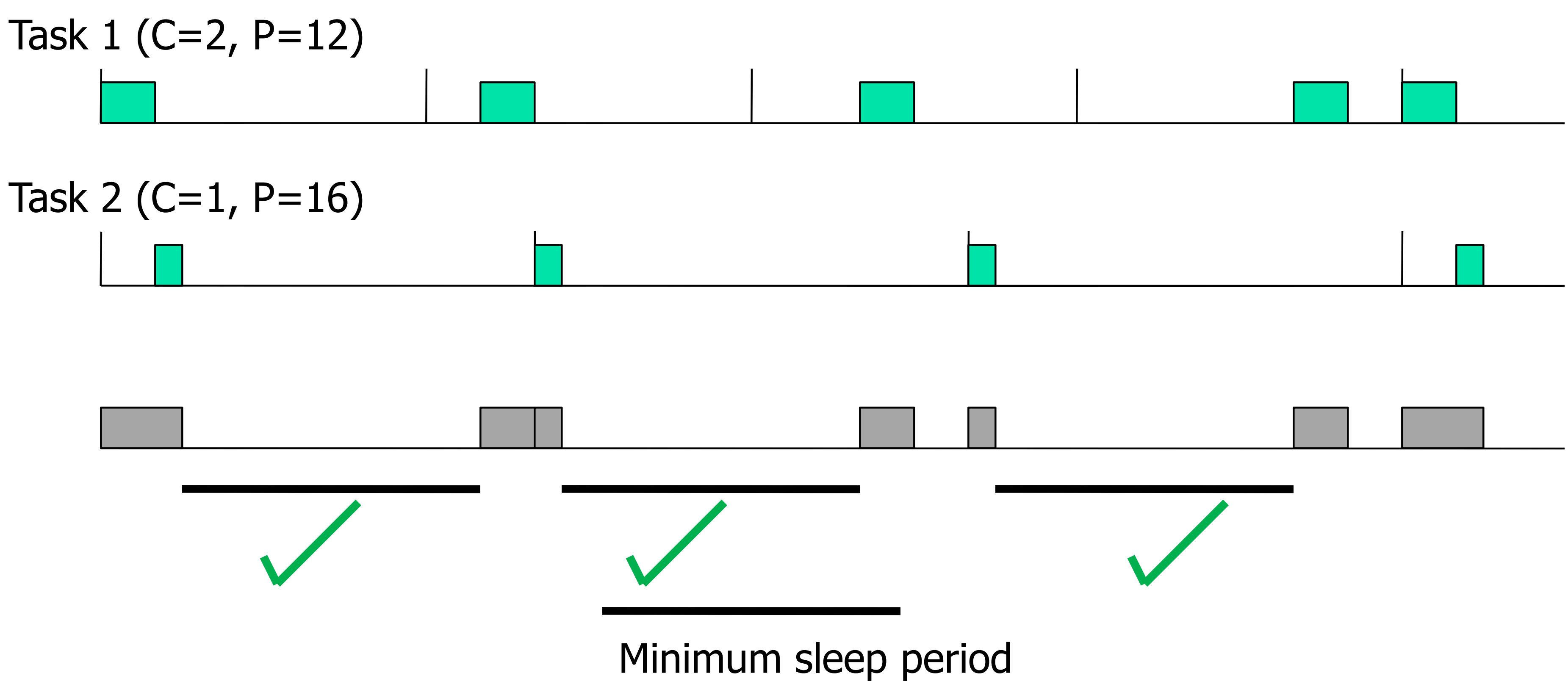
DPM and the Problem with Work-conserving Scheduling

- No opportunity to sleep ☹



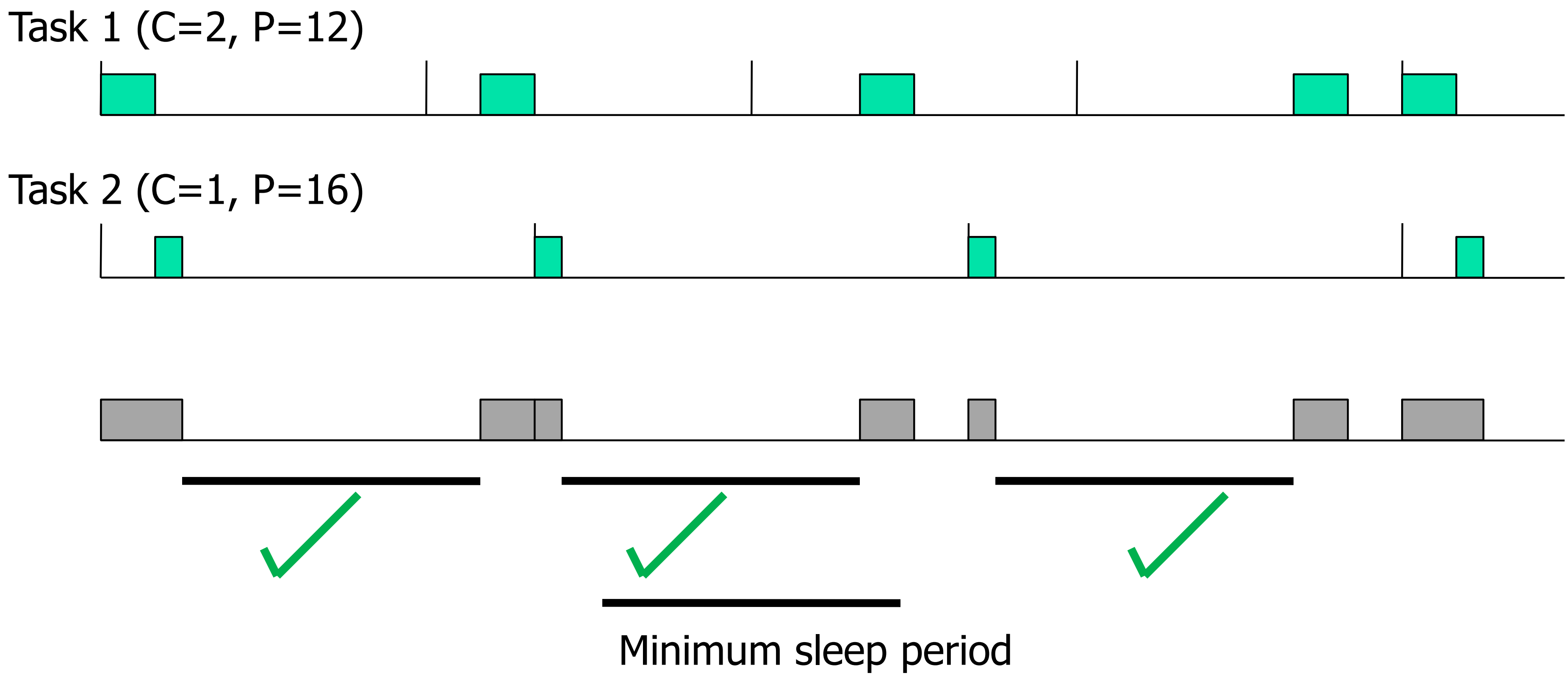
DPM and the Problem with Work-conserving Scheduling

- Must batch! ☺



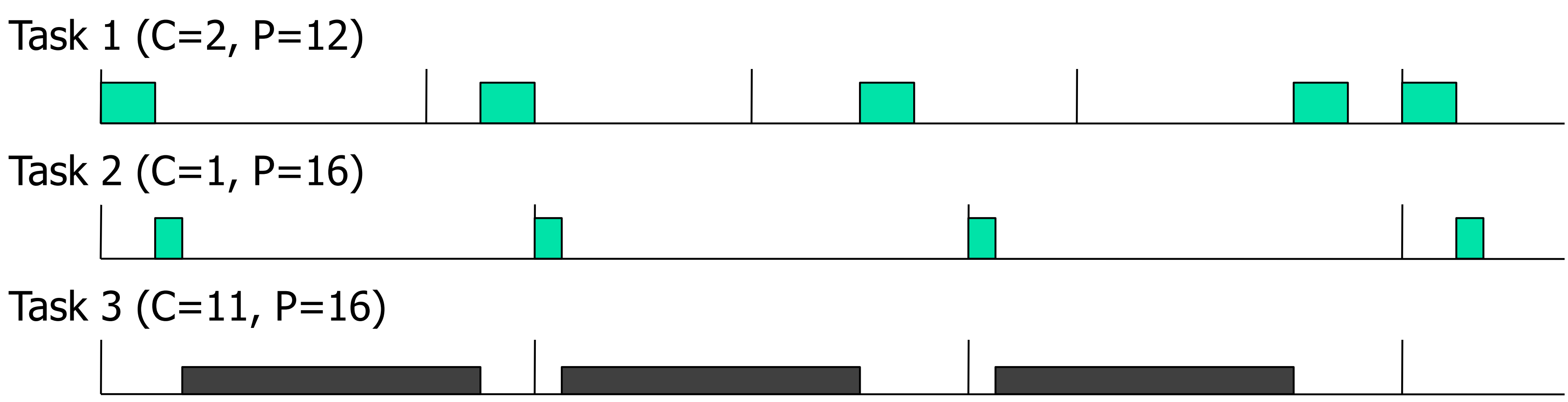
A Schedulability Question:

How to Analyze Schedules with Sleep Periods?



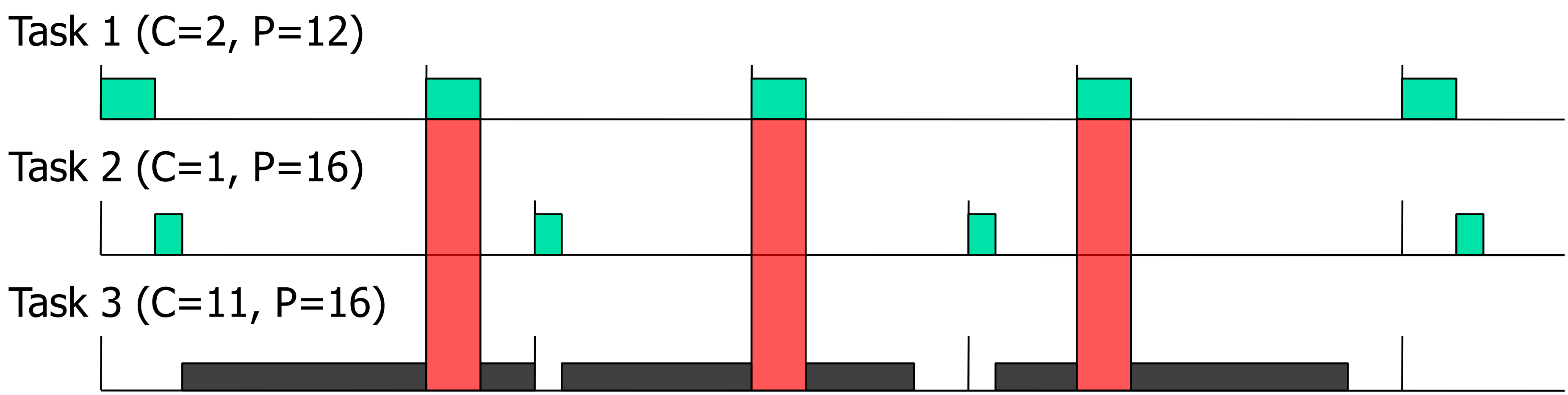
A Schedulability Question:
How to Analyze Schedules with Sleep Periods?

- Option 1: Treat sleep periods like a periodic task. Use the Liu and Layland utilization bound for schedulability. Problems?



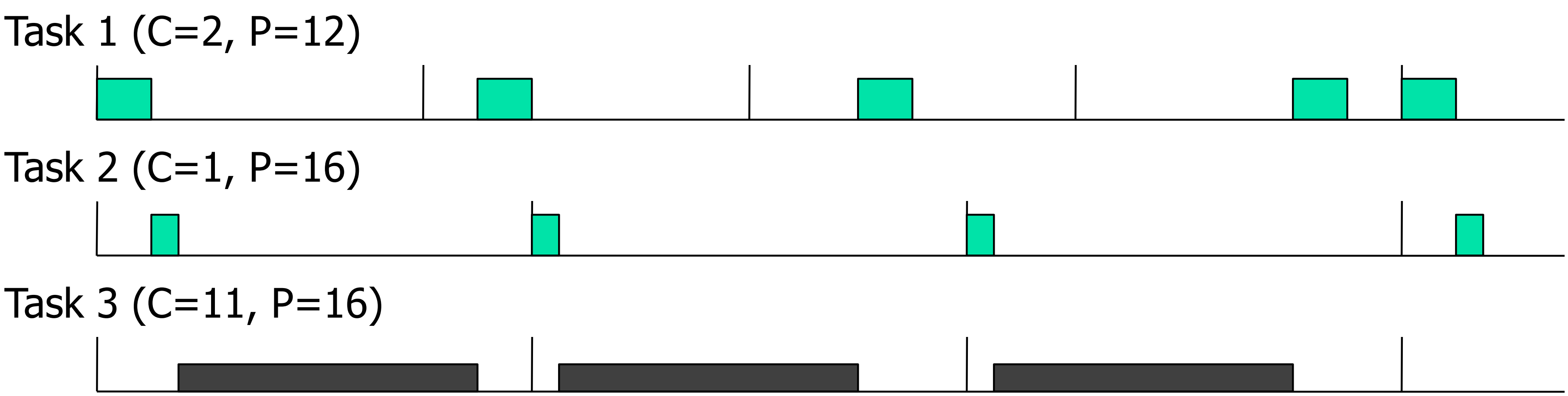
A Schedulability Question:
How to Analyze Schedules with Sleep Periods?

- Option 1: Treat sleep periods like a periodic task. Use the Liu and Layland utilization bound for schedulability. Problems?
  - Does not work because the "sleep task" cannot be preempted, whereas the rest of the tasks are preemptible. The utilization bound works only for fully preemptive scheduling.



A Schedulability Question:
How to Analyze Schedules with Sleep Periods?

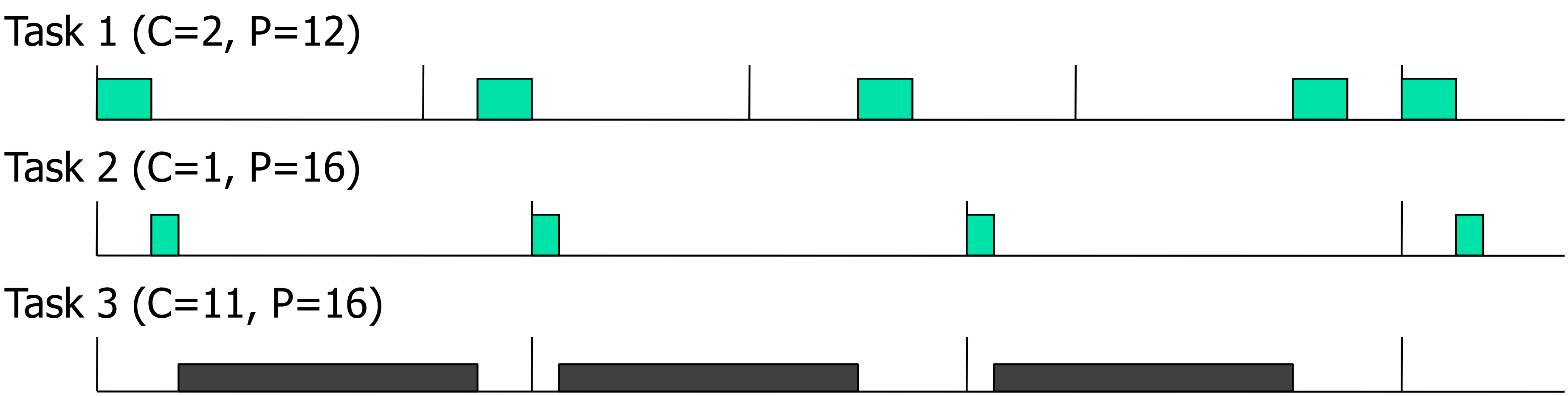
- Option 2: Treat sleep periods like the *highest-priority* periodic task. Use the Liu and Layland utilization bound for schedulability. Problems?



A Schedulability Question:

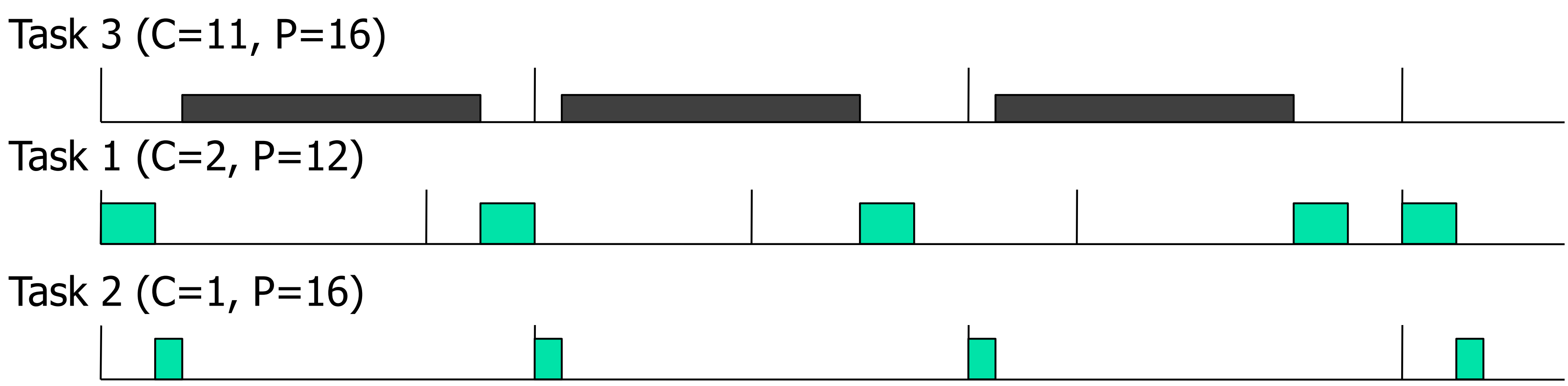
How to Analyze Schedules with Sleep Periods?

- Option 2: Treat sleep periods like the *highest-priority* periodic task. Use the Liu and Layland utilization bound for schedulability. Problems?
  - Does not work because the "sleep task" may need to have a larger period than the actual top-priority task, which contradicts rate-monotonic scheduling. The bound does not work.



A Schedulability Question:
How to Analyze Schedules with Sleep Periods?

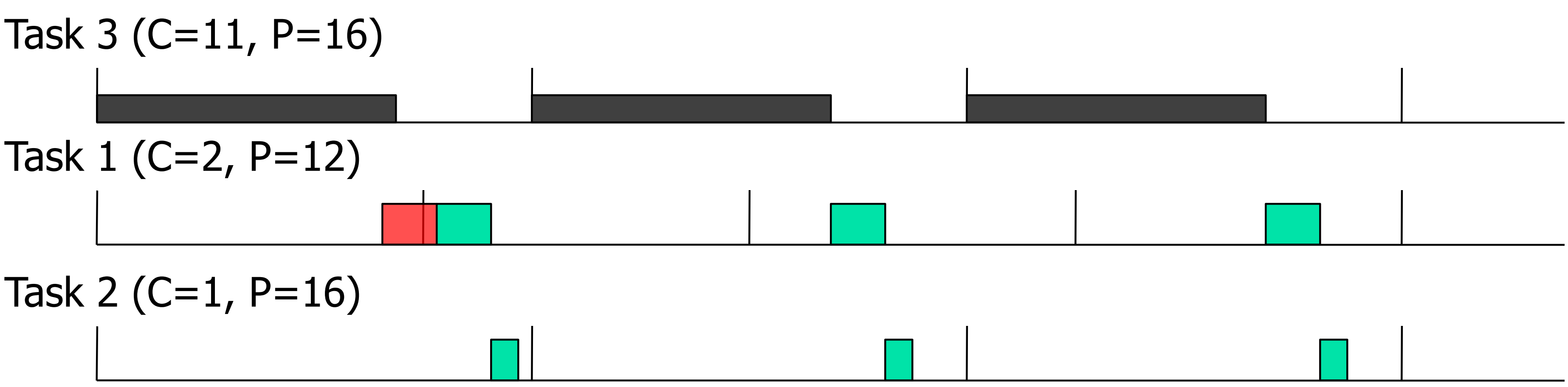
- Option 3: Treat sleep periods like the *highest-priority* periodic task. Use *exact response time analysis* for schedulability. Problems?



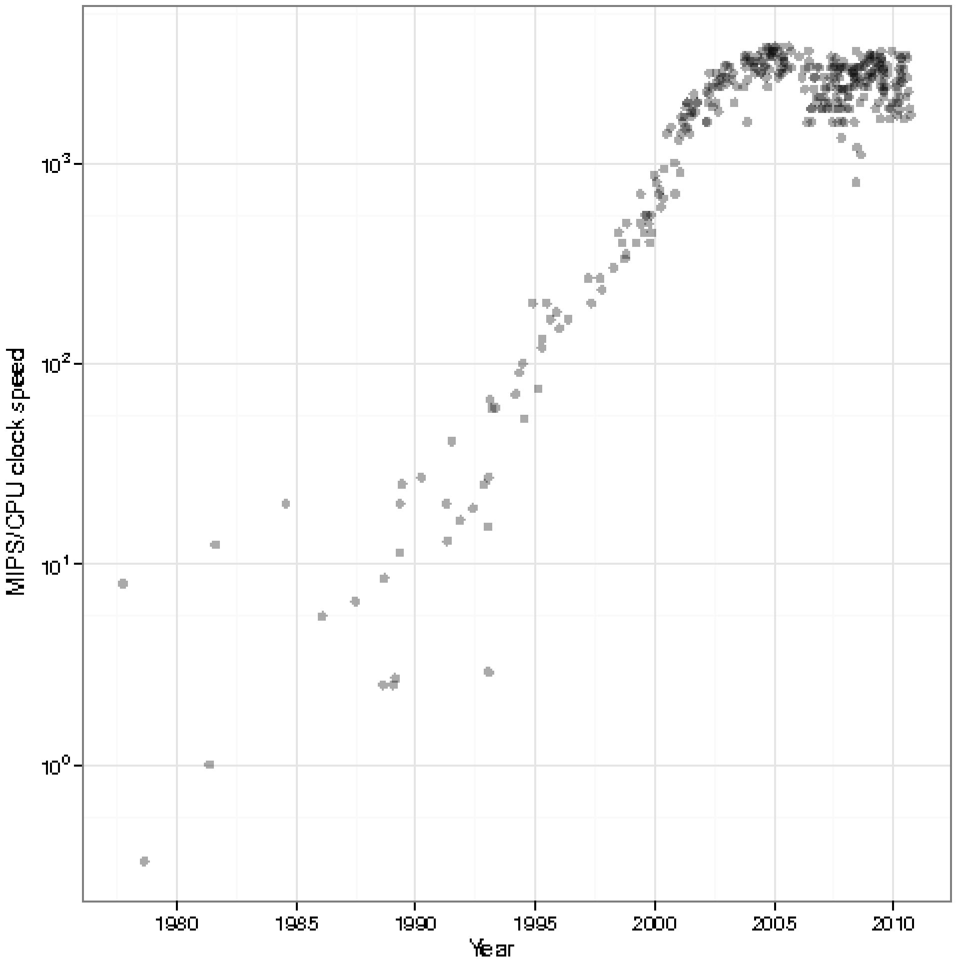
Device Forbidden Regions

- Option 3: Treat sleep periods like the *highest-priority* periodic task. Use *exact response time analysis* for schedulability. Problems?
  - A Valid solution, but pessimistic.

  (Called: Device Forbidden Regions. Published in RTAS 2008.)

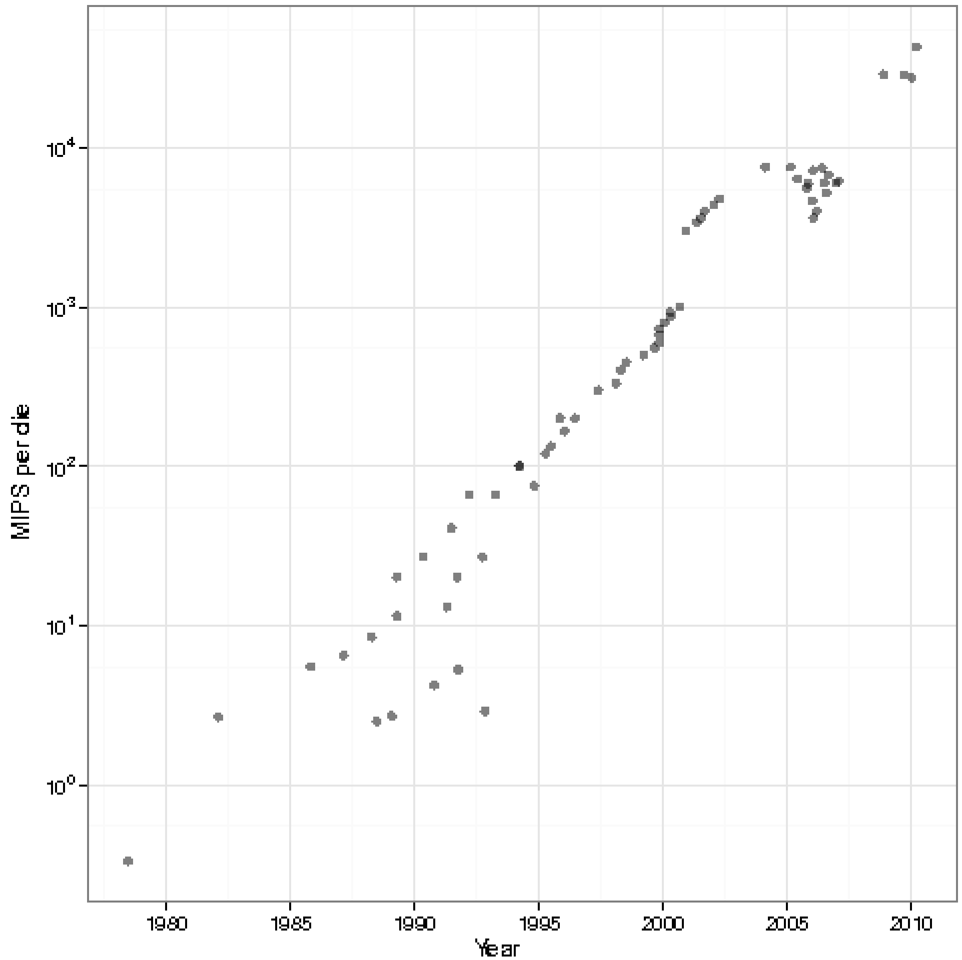


Intel CPU Clock Speed



- Moore's Law (1980-2005)
- Question: Why did the speed curve saturate (around 2005)?

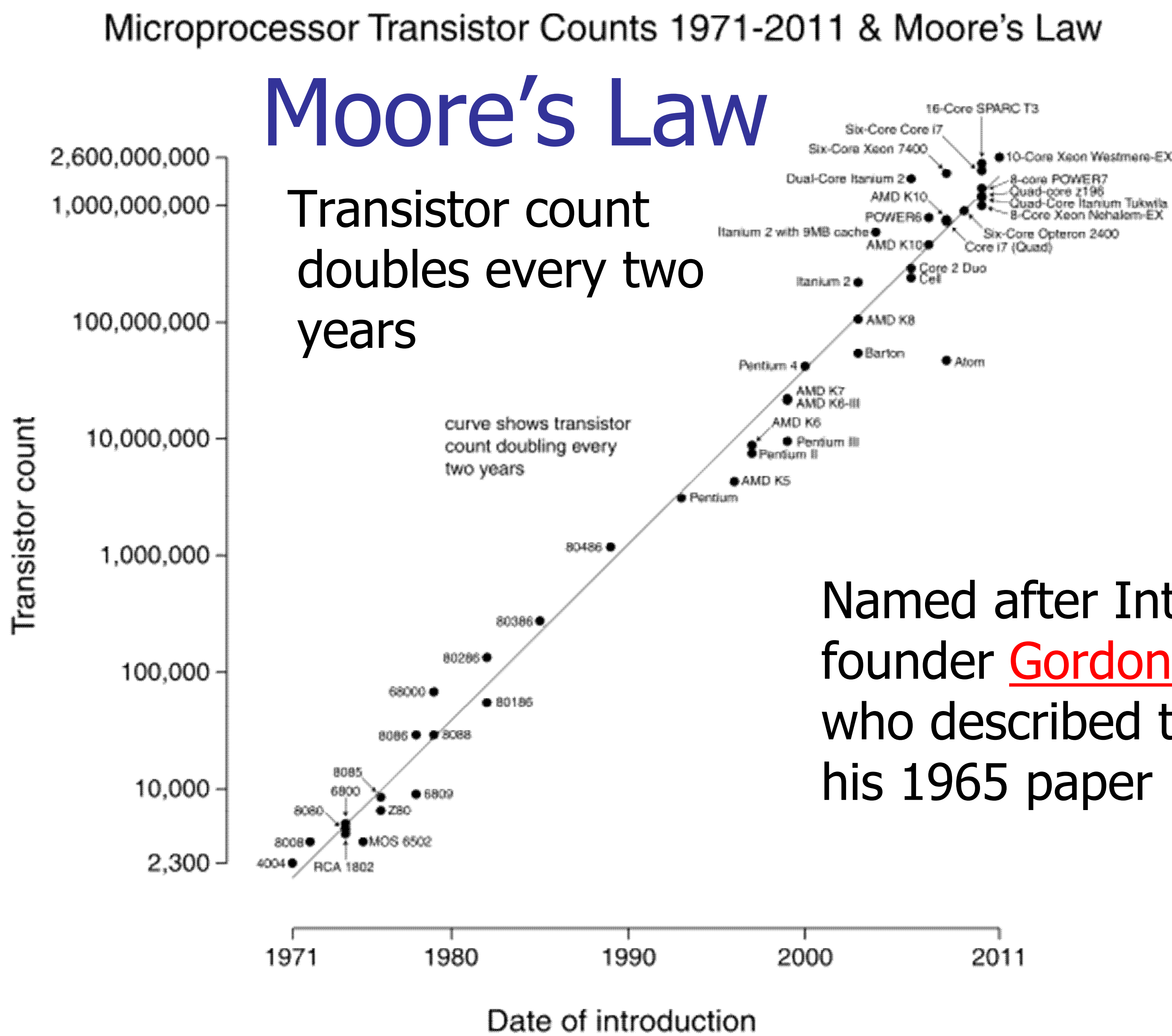
Computational Power (per Die)



- Note the exponential rise in power consumption
- Question: how come it does not saturate?

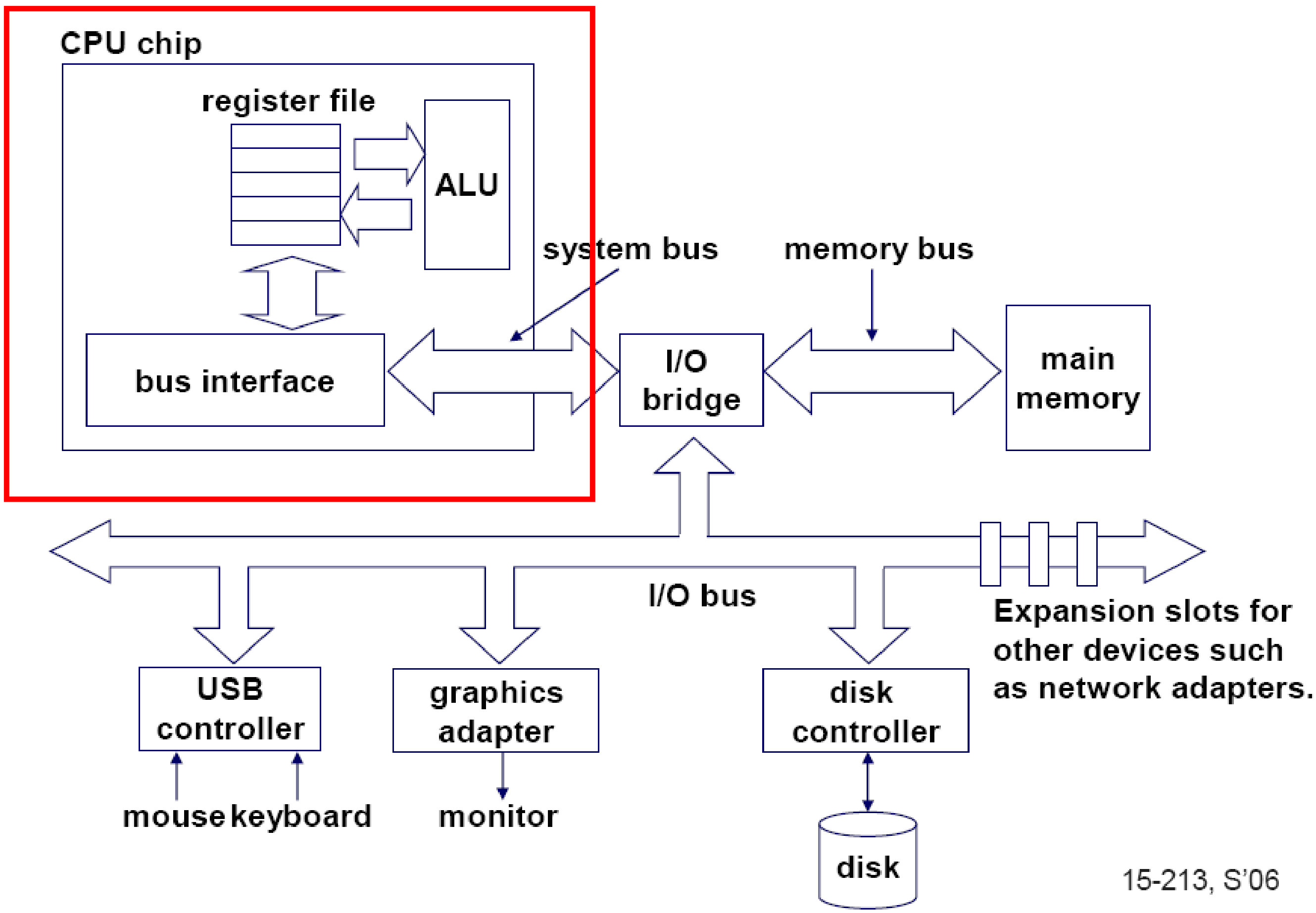
Moore's Law

Transistor count doubles every two years

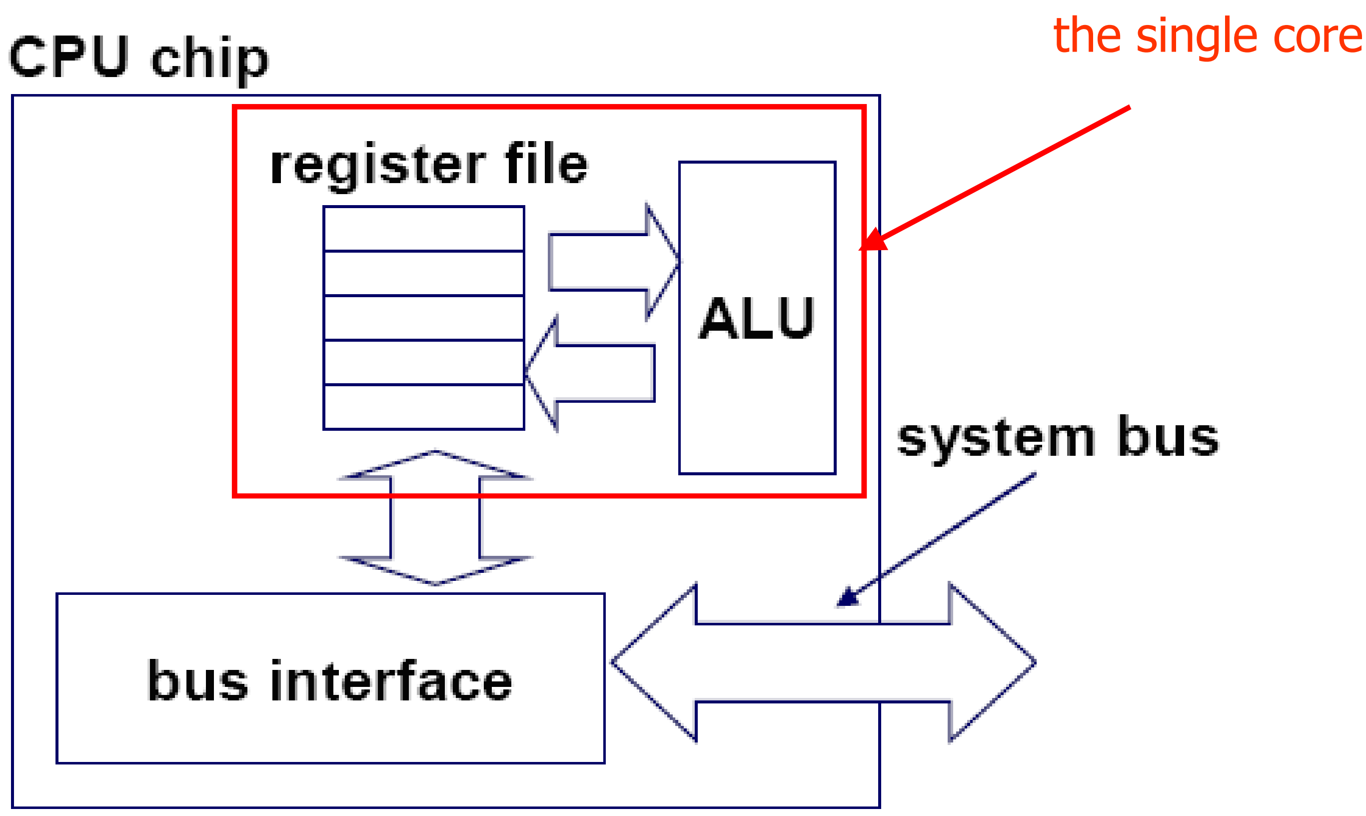


Named after Intel co-founder Gordon E. Moore, who described the trend in his 1965 paper

Single-core computer

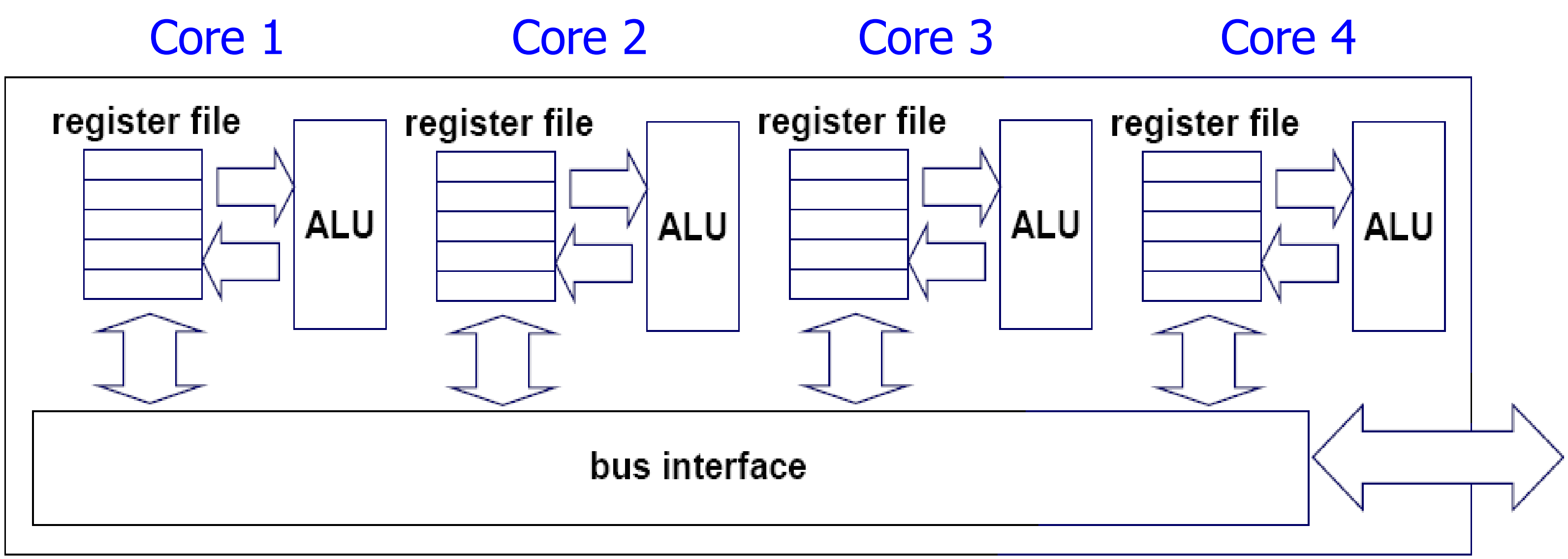


Single-core CPU chip



Multi-core Architectures

- Replicate multiple processor cores on a single die.



Multi-core CPU chip

Interaction with the Operating System

- OS perceives each core as a separate processor
- OS scheduler maps threads/processes to different cores
- Major OSes support multi-core today: Windows, Linux, Mac OS X, …

DVS on Multiprocessors

- Consider a set of tasks, where task i has period P_i and total number of cycles C_i
  - Sort tasks from largest to smallest utilization C_i / P_i
  - Assign tasks one at a time (largest-first) to the least utilized processor
  - Apply one of the previous algorithms on each processor separately

Question

- From the perspective of minimizing energy, is it always a good idea to use up all processors?

How Many Processors to Use?

- Consider using one processor at frequency f versus two at frequency $f/2$
- Case 1: Total power for one processor
  - $k_f f^3 + R_0$
- Case 2: Total power for two processors
  - $2 \{k_f (f/2)^3 + R_0\} = k_f f^3/4 + 2 R_0$

How Many Processors to Use?

- Consider using one processor at frequency f versus two at frequency $f/2$
- Case 1: Total power for one processor
  - $k_f f^3 + R_0$
- Case 2: Total power for two processors
  - $2 \{k_f (f/2)^3 + R_0\} = k_f f^3 / 4 + 2 R_0$
- The general case: n processors
  - $n \{k_f (f/n)^3 + R_0\} = k_f f^3 / n^2 + n R_0$

How Many Processors to Use?

- The general case: n processors
  - $Power = n\,\{k_f\,(f/n)^3 + R_0\} = k_f\, f^{\,3}/\, n^{\,2} + n\, R_0$
  - $dPower/dn = -2\, k_f\, f^{\,3}/\, n^{\,3} + R_0 = 0$

$$n = \sqrt[3]{\frac{2k_f\, f^{\,3}}{R_0}}$$

How Many Processors to Use?

- The general case: n processors
  - $Power = n\ \{k_f\ (f/n)^3 + R_0\} = k_f\ f^3/\ n^2 + n\ R_0$
  - $dPower/dn = -2\ k_f\ f^3/\ n^3 + R_0 = 0$

$$n = \sqrt[3]{\frac{2k_f\, f^3}{R_0}}$$

- What if n is not an integer?