Layering Multi-core Scheduling Innovation Atop Existing RTOS Platforms

Mac Mollison

Real-Time Systems

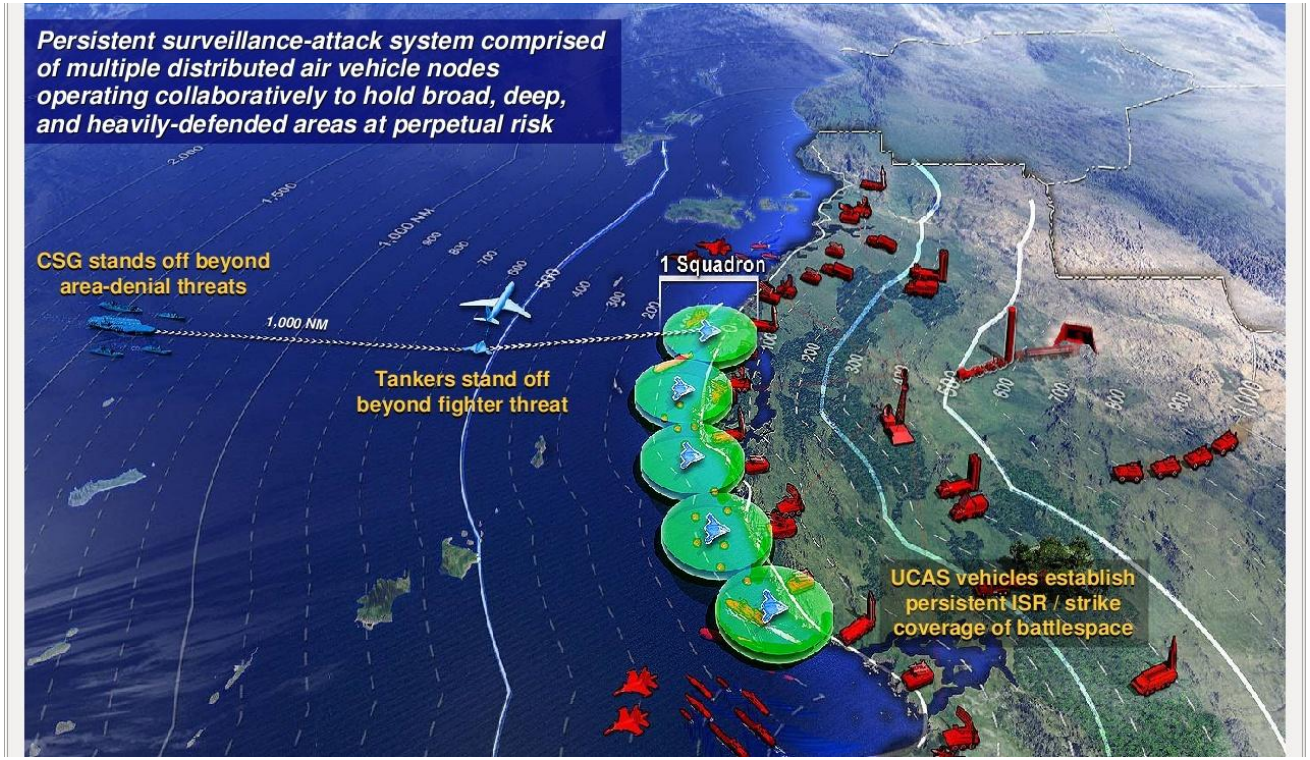
at the University of North Carolina at Chapel Hill

Preview

- There is a **large gap** between current capabilities and future systems.
- To close this gap, **new ways of thinking** are emerging.
- To support these new ways of thinking, **software platforms must evolve.**
- I will propose a new way to **enable this evolution.**

Next-Gen Capabilities

- Autonomous, fractionated, and intelligent.
- Case in point: **UCAS.**



Next-Gen Technologies

- Mixed criticality
- Adaptivity
- Composability
- Trust
  - Behavior assurance
  - V&V
- etc.

A Growing Gap

- Significant **research advances** in many of these areas.
- Software platforms are not keeping up.
  - Particularly, RTOS and middleware.

This Presentation

- I will **propose a way to evolve underlying software platforms** more rapidly and safely.
- I will **focus in on real-time scheduling** in this presentation.

Next-Gen Real-Time

- Need to “juggle” **many** tasks with **demanding** requirements.
- Over time and across multiple cores.
- For example, advanced mission management software.

Significant research progress

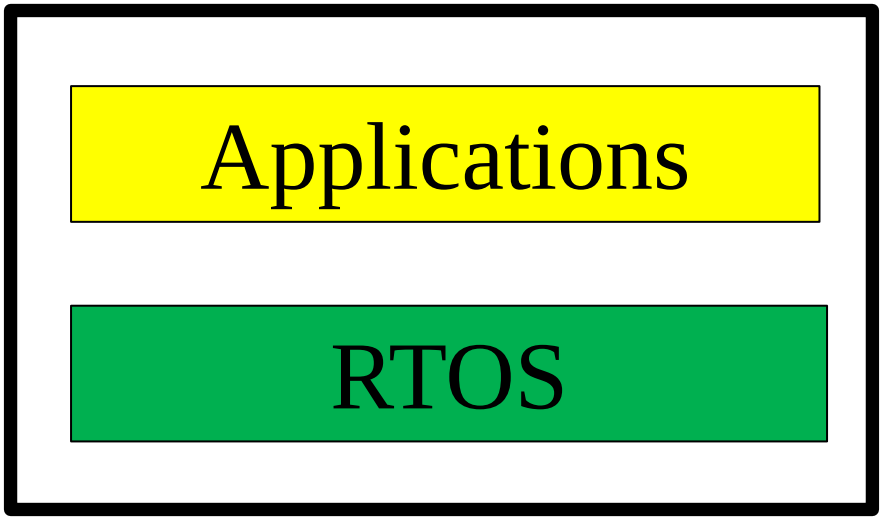
- Just a sample of active research:
  - Global soft real-time scheduling
  - Adaptive scheduling
  - Mixed-criticality scheduling
- Entire research community exists.
- **But…** (next slide)

But…

- Ultimately, industry practitioners need **finished software products** they can obtain, use, and re-use.
  - Not just academic papers and dissertations.
- Rest of this presentation:
  - Where we are now
  - **How to get where we're going**

Where we are now

- Current RTOSs are **solid and reliable**, but **very rigid.**
- Not able to satisfy the needs we've been describing.

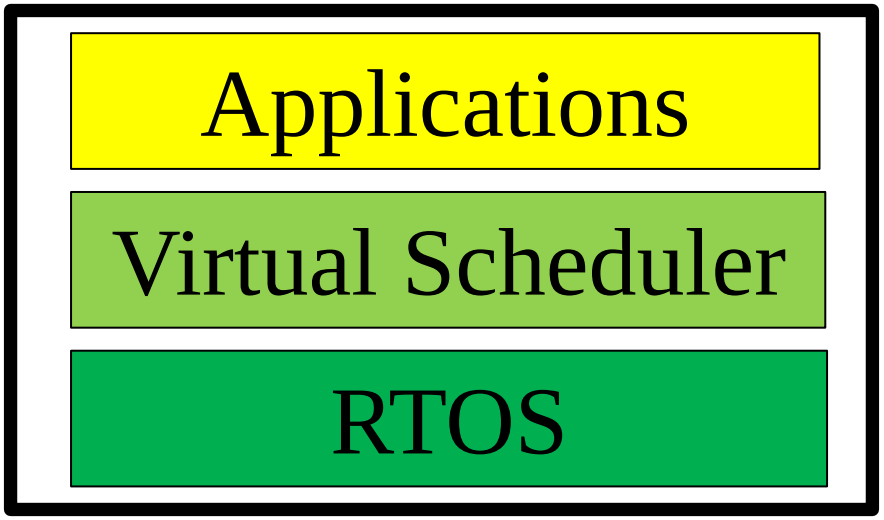


Getting to the future

- Evolve RTOSs?
  - Not **cost effective**
  - Not **flexible** for app. developers
  - Hard to choose **what to implement** in RTOS
  - We *want* **stable RTOSs**
- Up next: Another way forward

Atop the RTOS

- I'm proposing offering the "fancy" scheduling mentioned earlier via a **middleware layer**
- Tentatively, I call this **virtual real-time scheduling**.



Potential Benefits

- **Layered architecture** has been proven time and again
  - **Manage complexity**
  - Allows software developer **specialization**
  - Good way to **leverage** existing RTOS tech

Implementation Tradeoffs

- Many ways to implement
  - Samples given in later slides
- There are many **implementation tradeoffs**
- Thus, a very thorough **trade study** is needed

Research Plan

- Implementation trade study: **next 3 years**.
- I am **very interested in collaborating** with system developers and RTOS vendors
- **If the approach proves feasible,**
  - I hope to move us close (or closer) to usable "virtual scheduling" infrastructure.

Sample Approaches

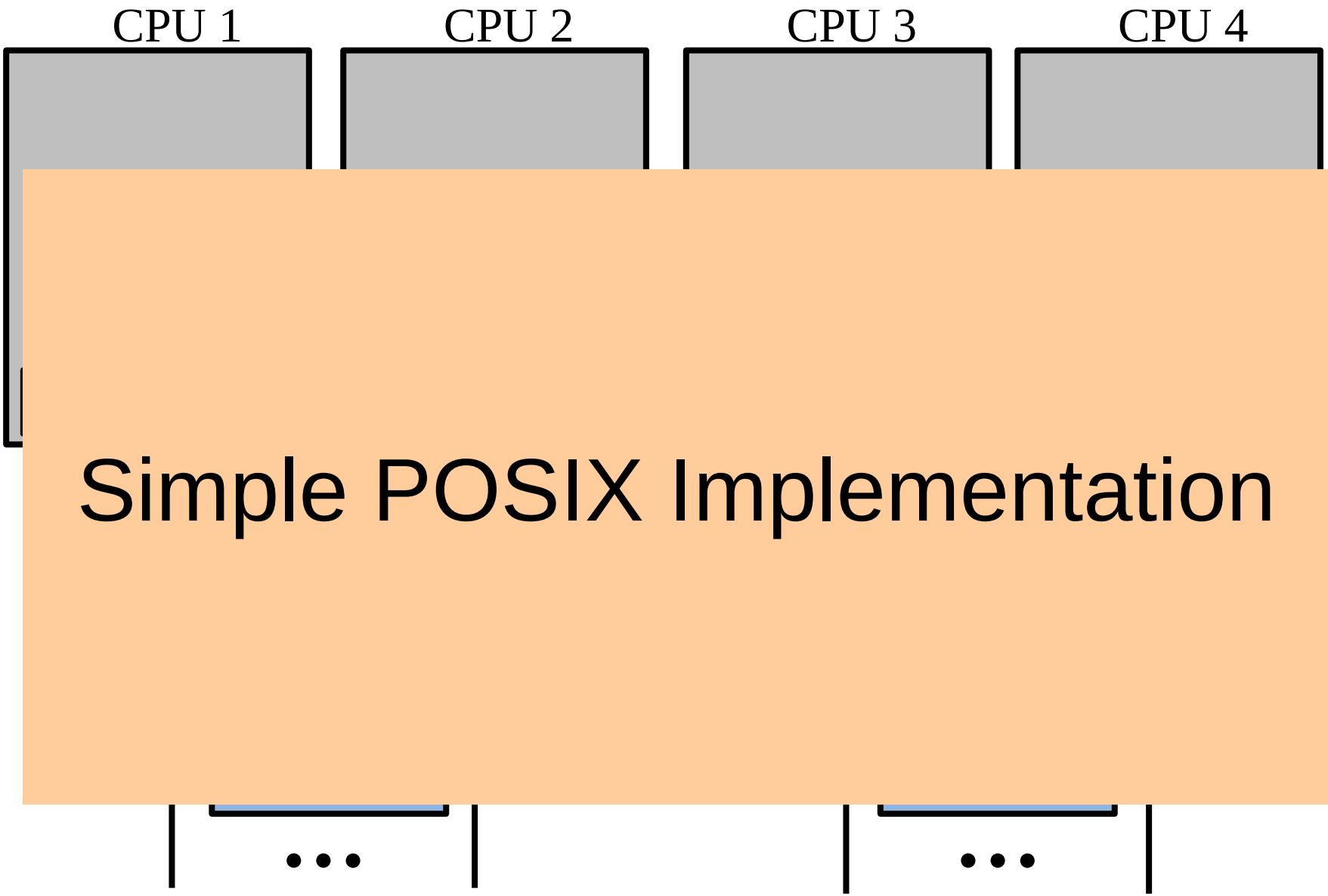
1. **POSIX**-based runtime library
2. Process **virtual machines**

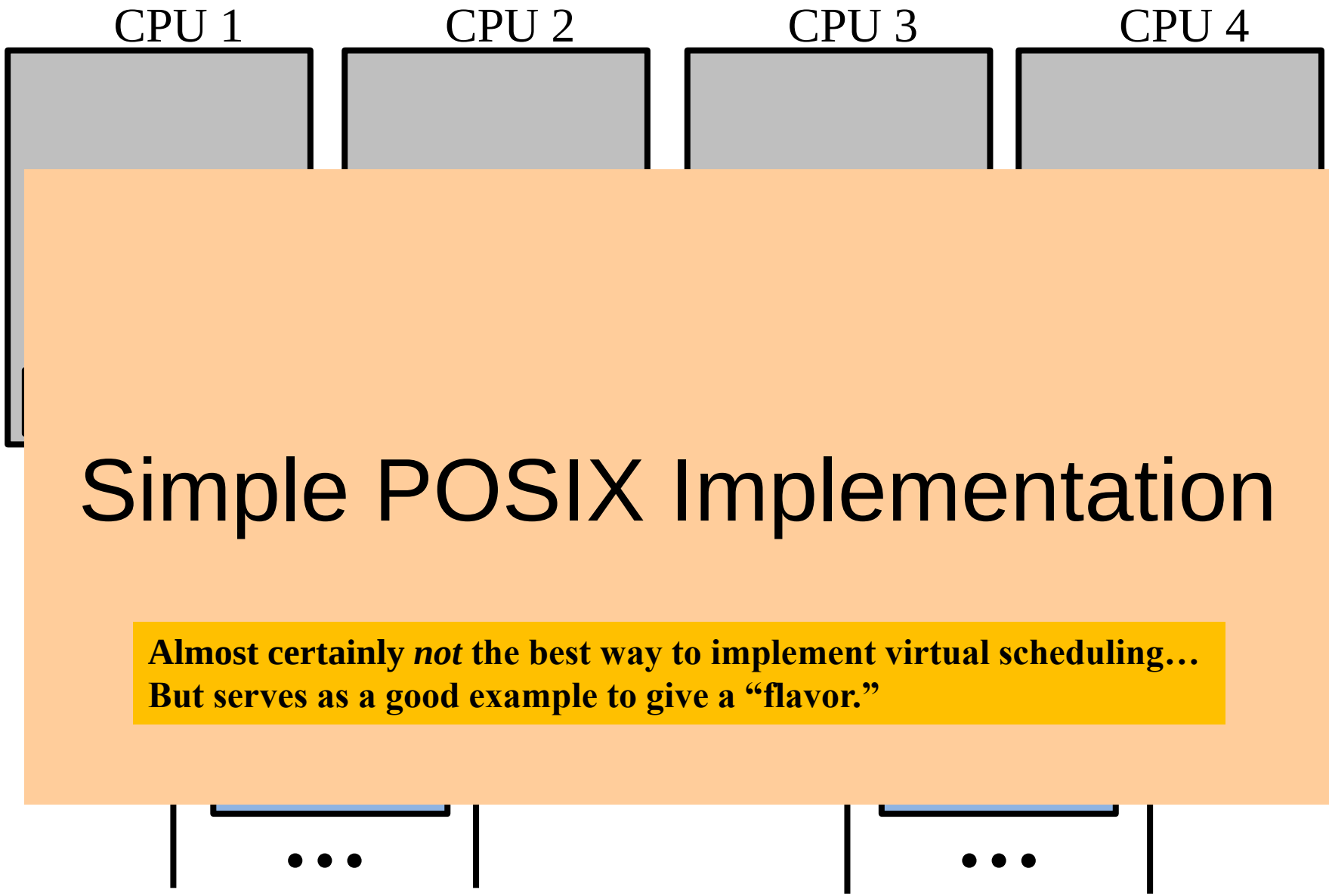
System Model

- *m* CPUs
- *n* tasks (assumed to be > *m*)

Therefore…

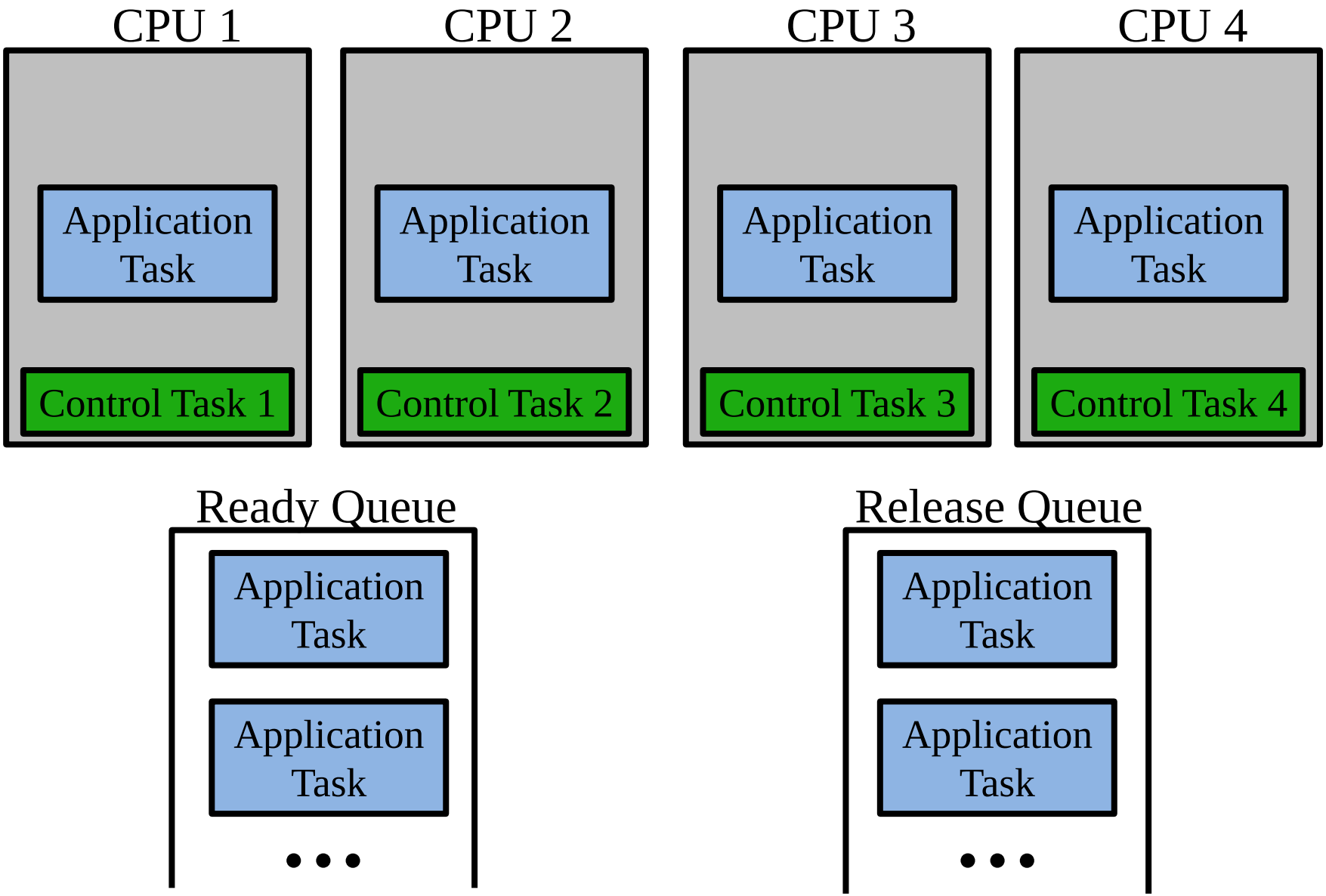
- At any time, the *n* highest-priority tasks run on the *m* CPUs
- Task priorities **determined by the application** and **can change dynamically.**

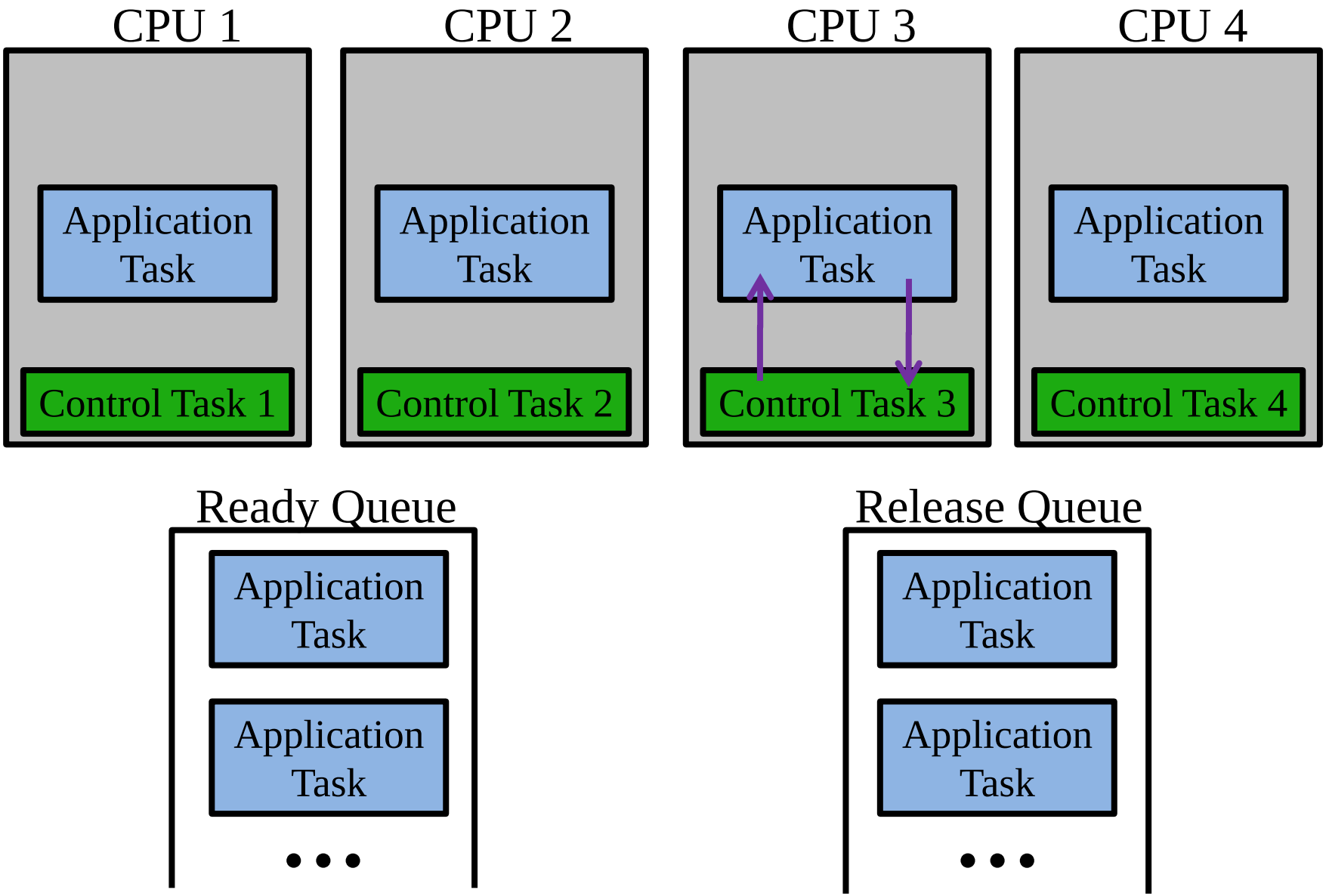

CPU 1
CPU 2
CPU 3
CPU 4
Simple POSIX Implementation
…
…

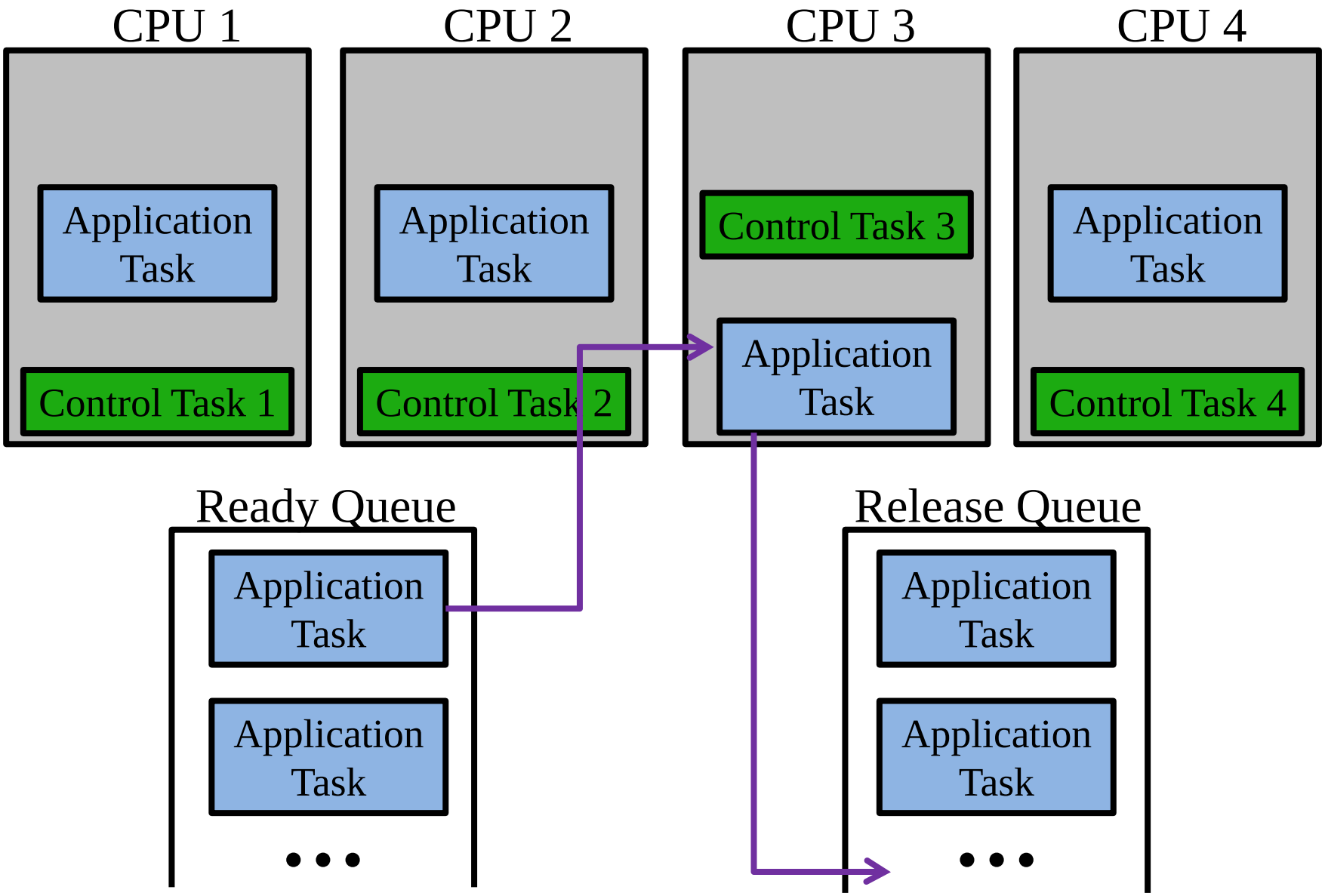


Simple POSIX Implementation

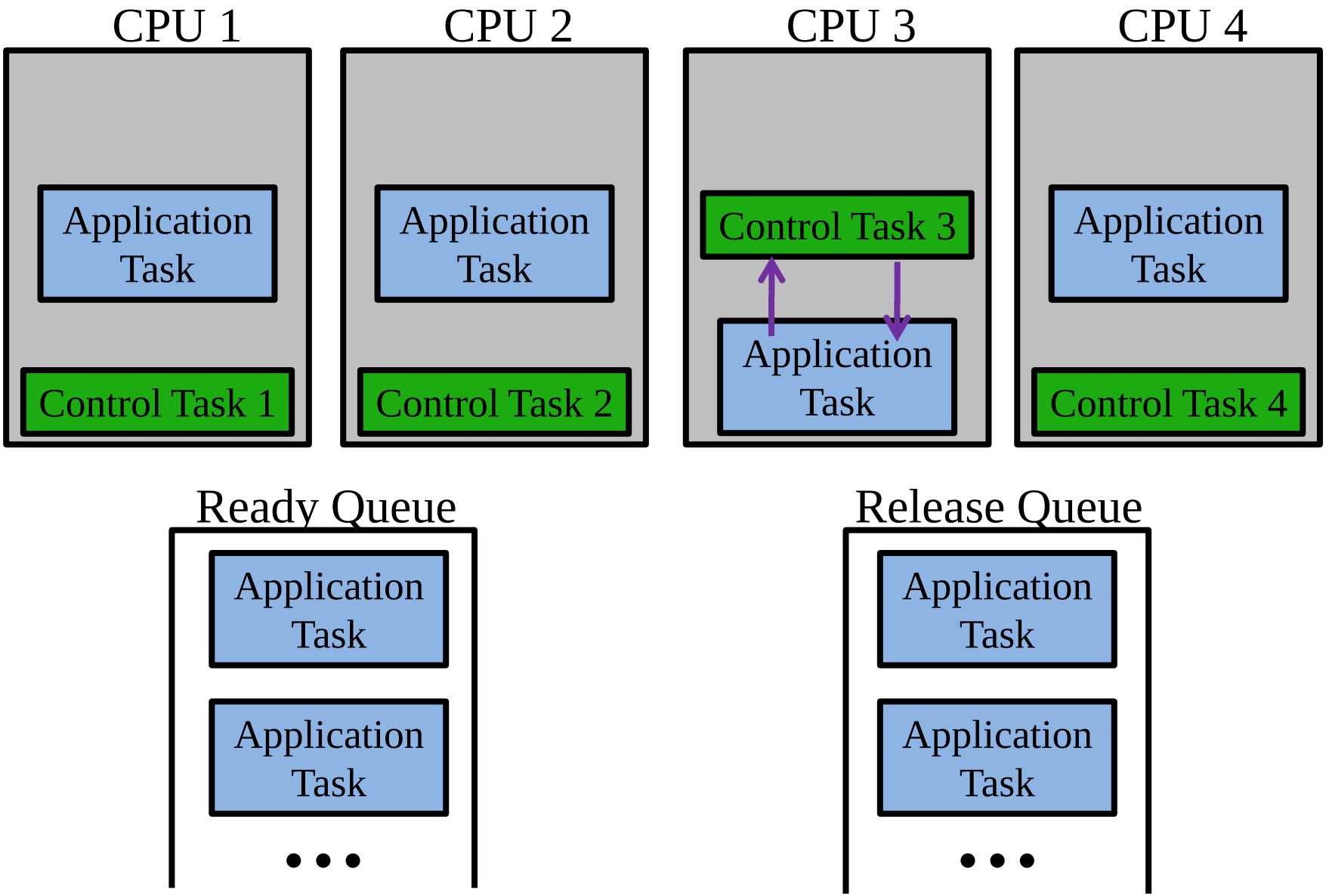
Almost certainly *not* the best way to implement virtual scheduling…
But serves as a good example to give a "flavor."

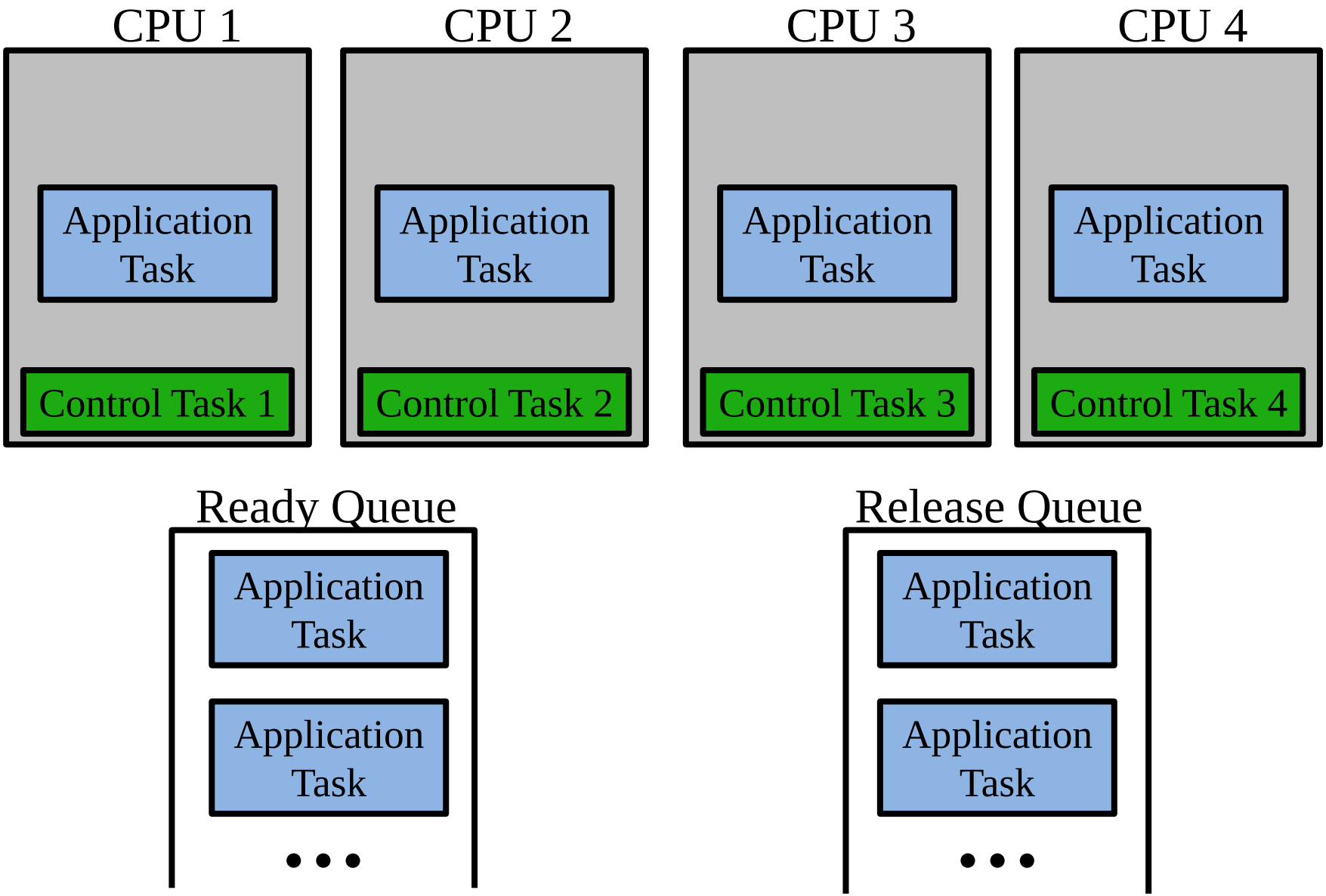

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Application Task
Application Task
• • •
Release Queue
Application Task
Application Task
• • •

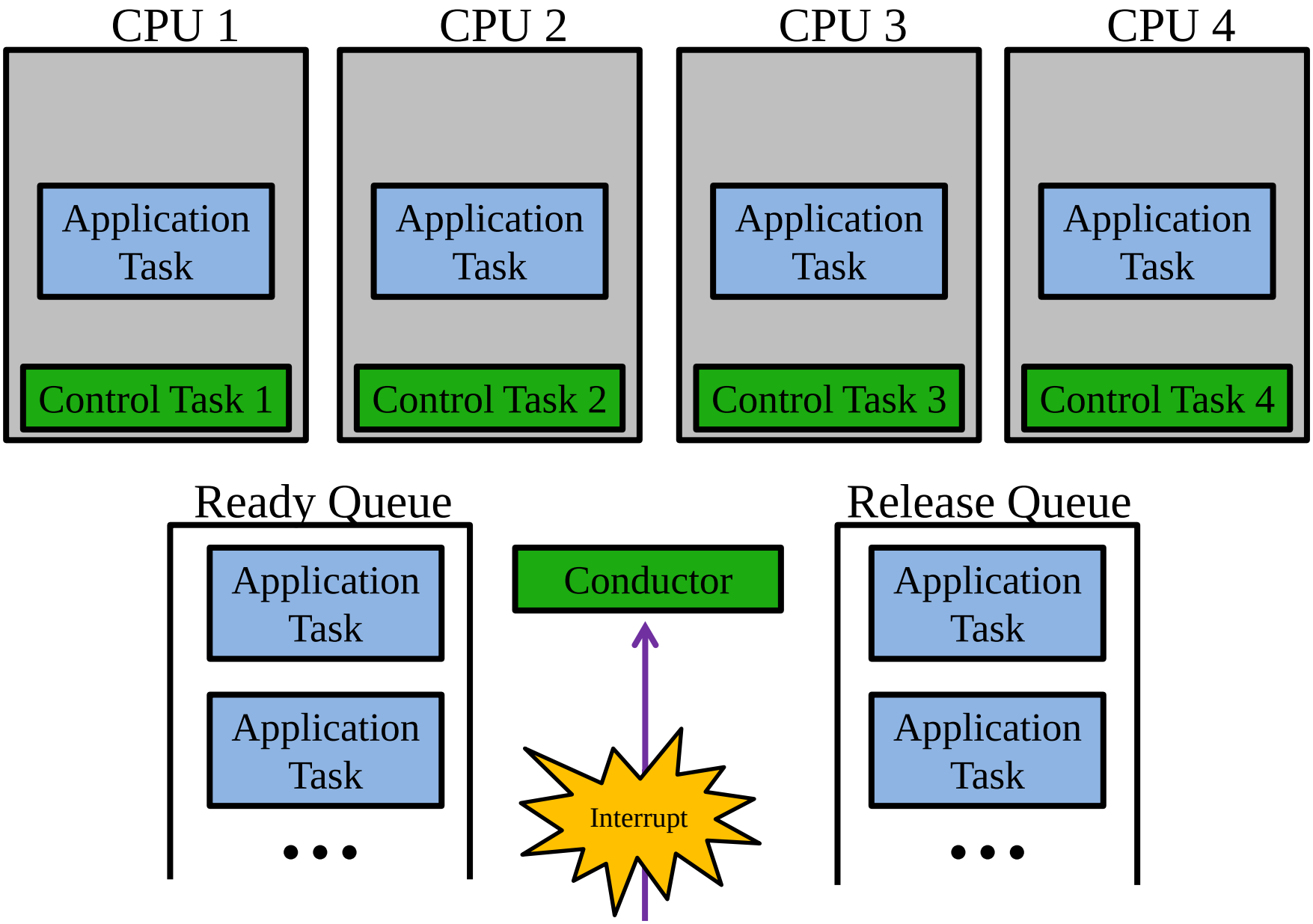

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Application Task
Application Task
Release Queue
Application Task
Application Task

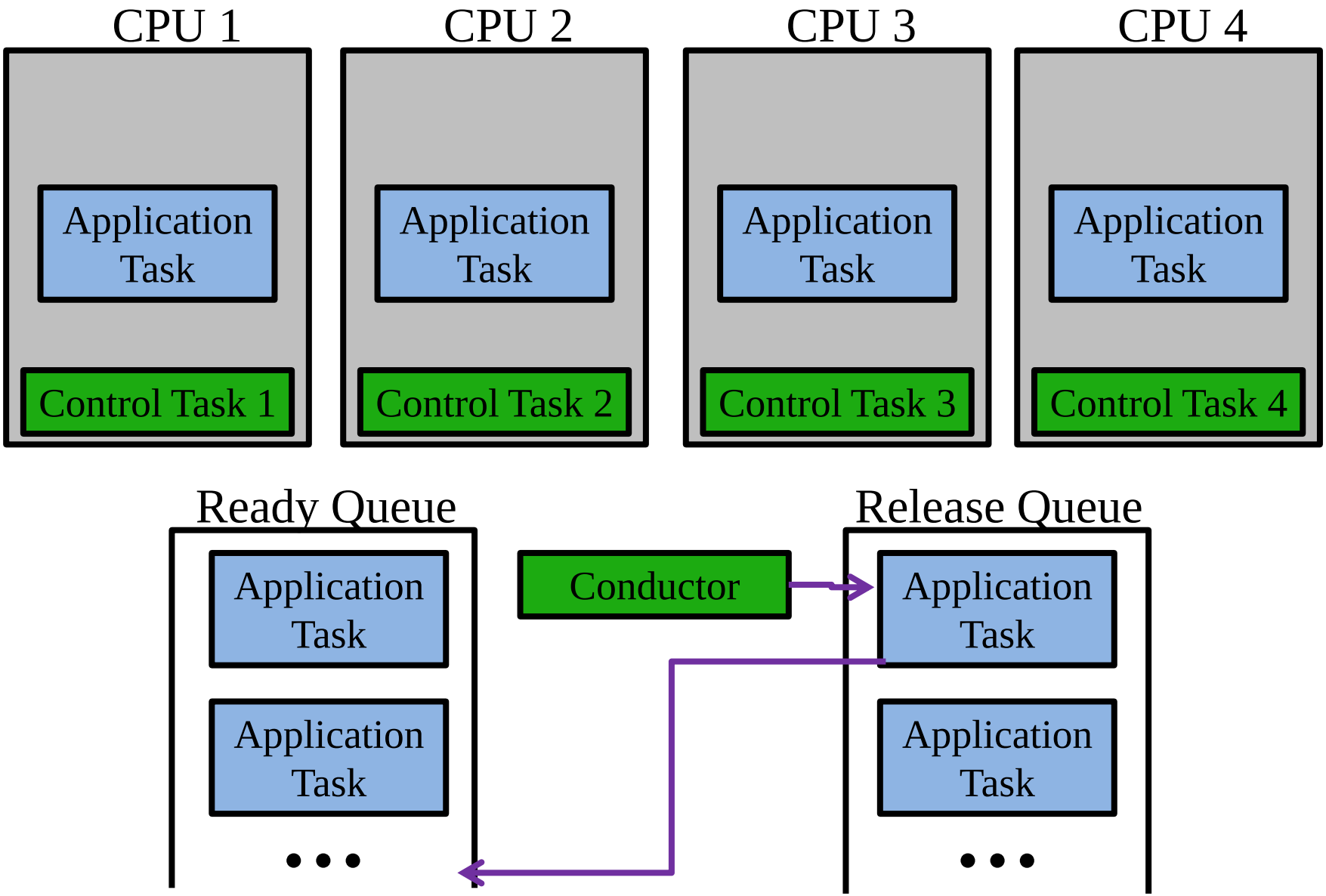

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Control Task 3
Application Task
Application Task
Control Task 1
Control Task 2
Application Task
Control Task 4
Ready Queue
Release Queue
Application Task
Application Task
Application Task
Application Task
• • •
• • •

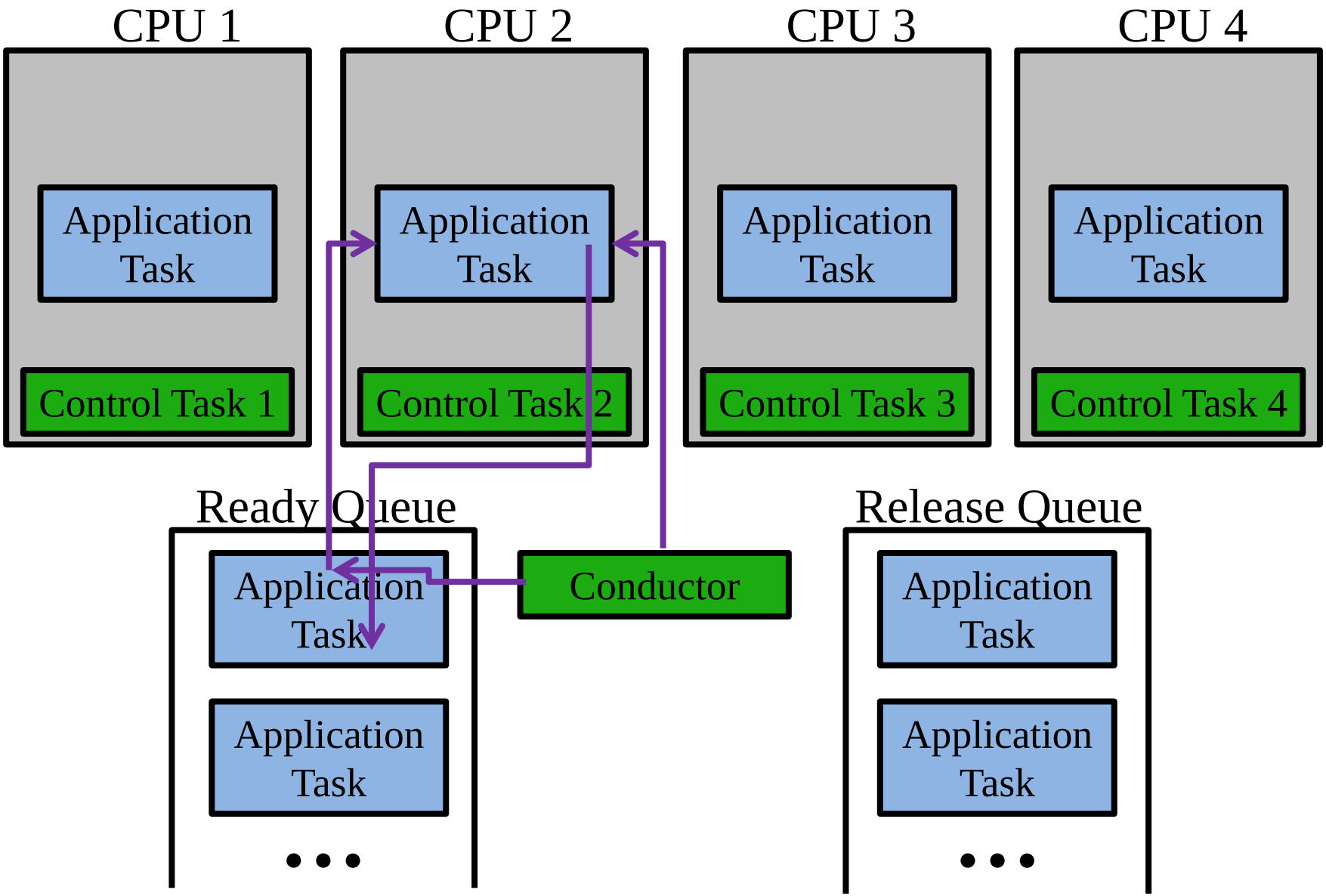
Control Task Becomes Inactive

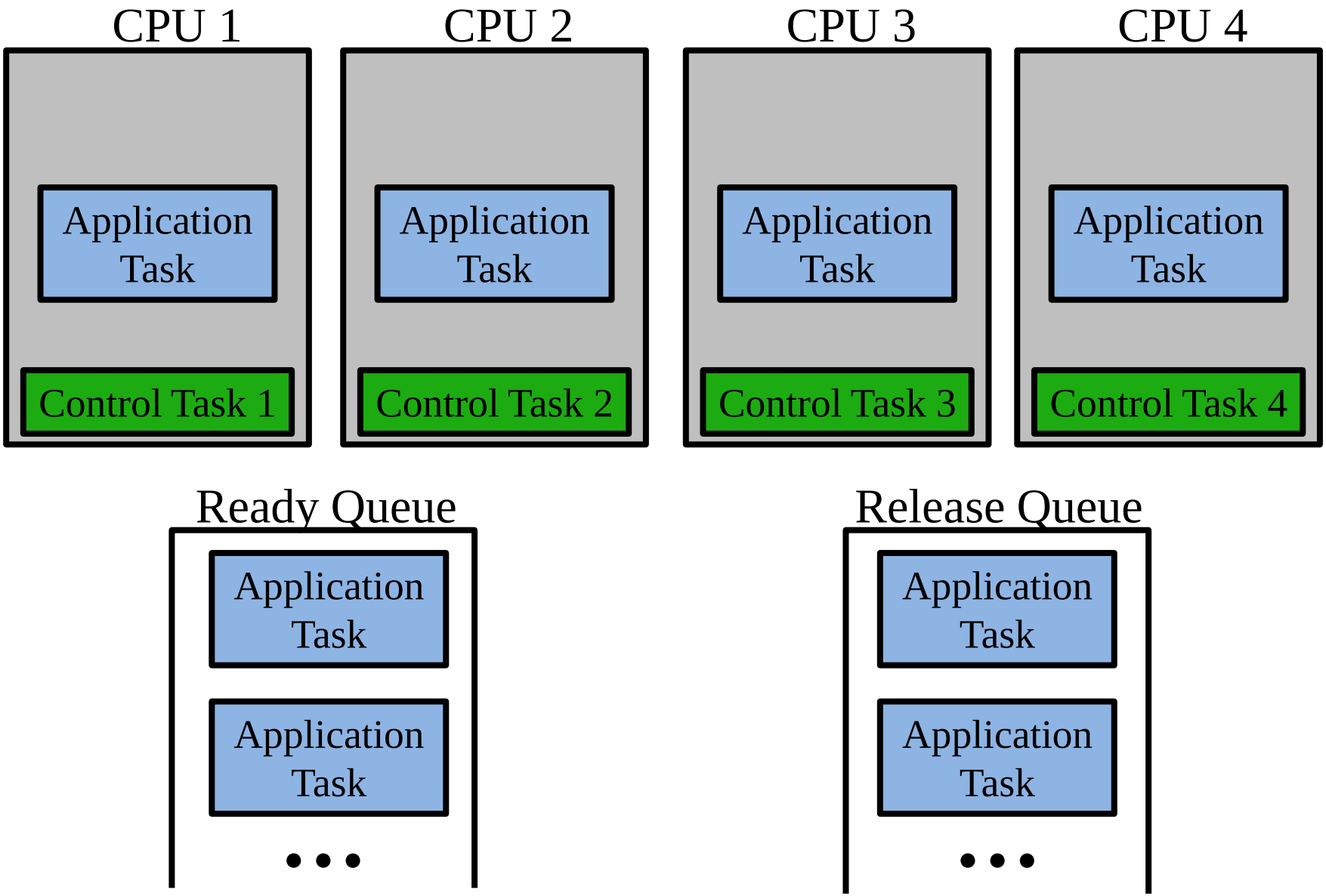


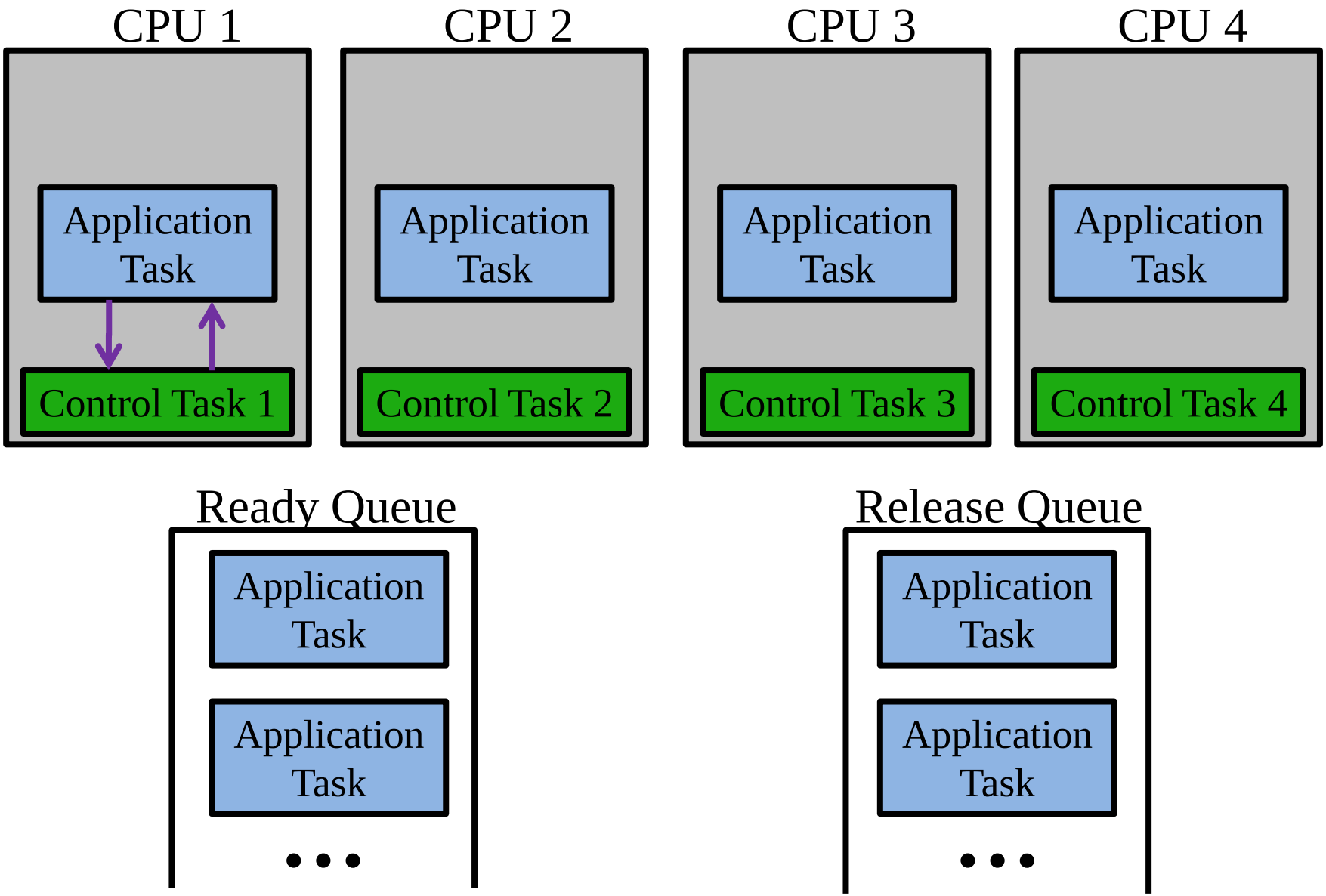

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Application Task
Application Task
• • •
Release Queue
Application Task
Application Task
• • •

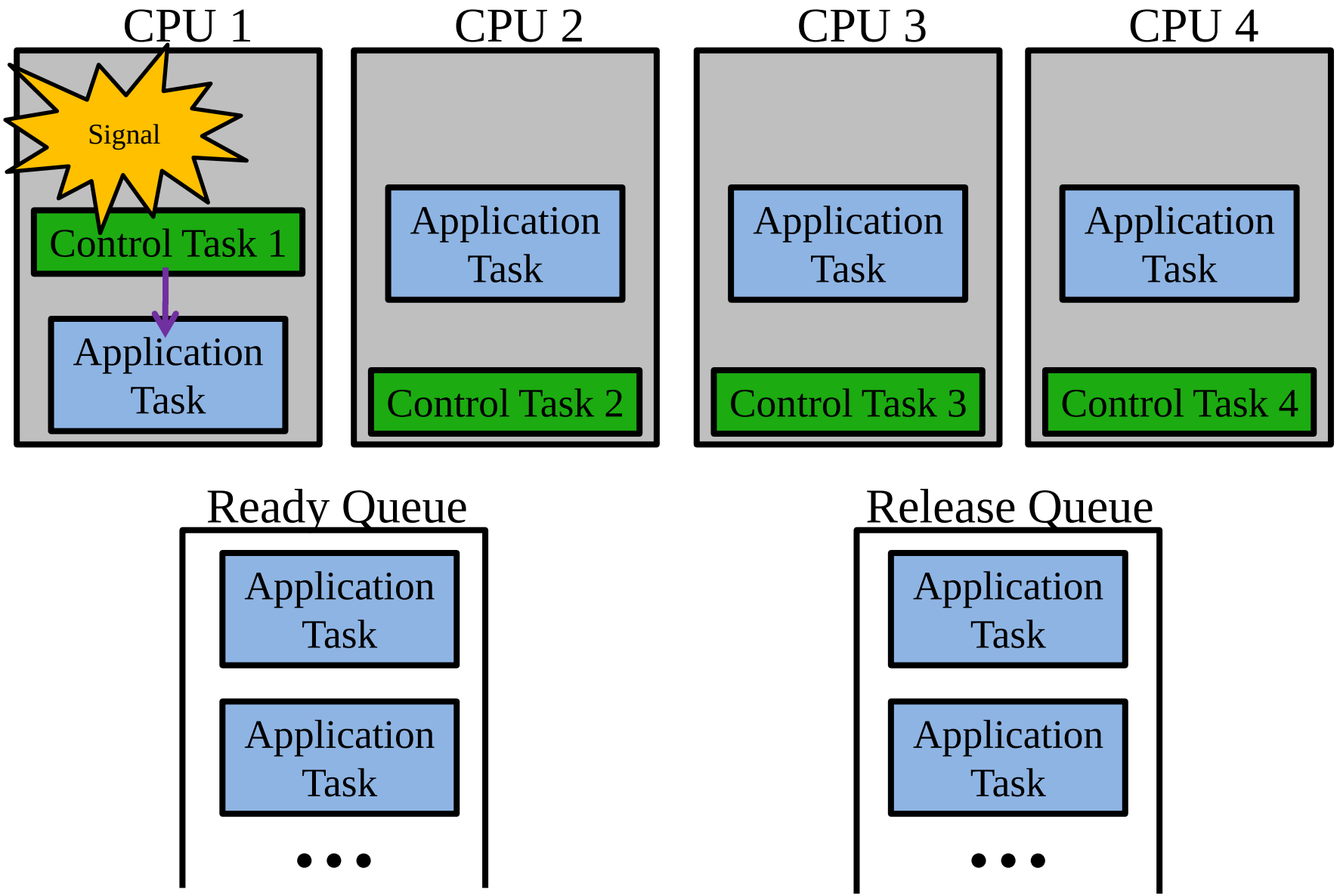

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Application Task
Application Task
• • •
Conductor
Interrupt
Release Queue
Application Task
Application Task
• • •

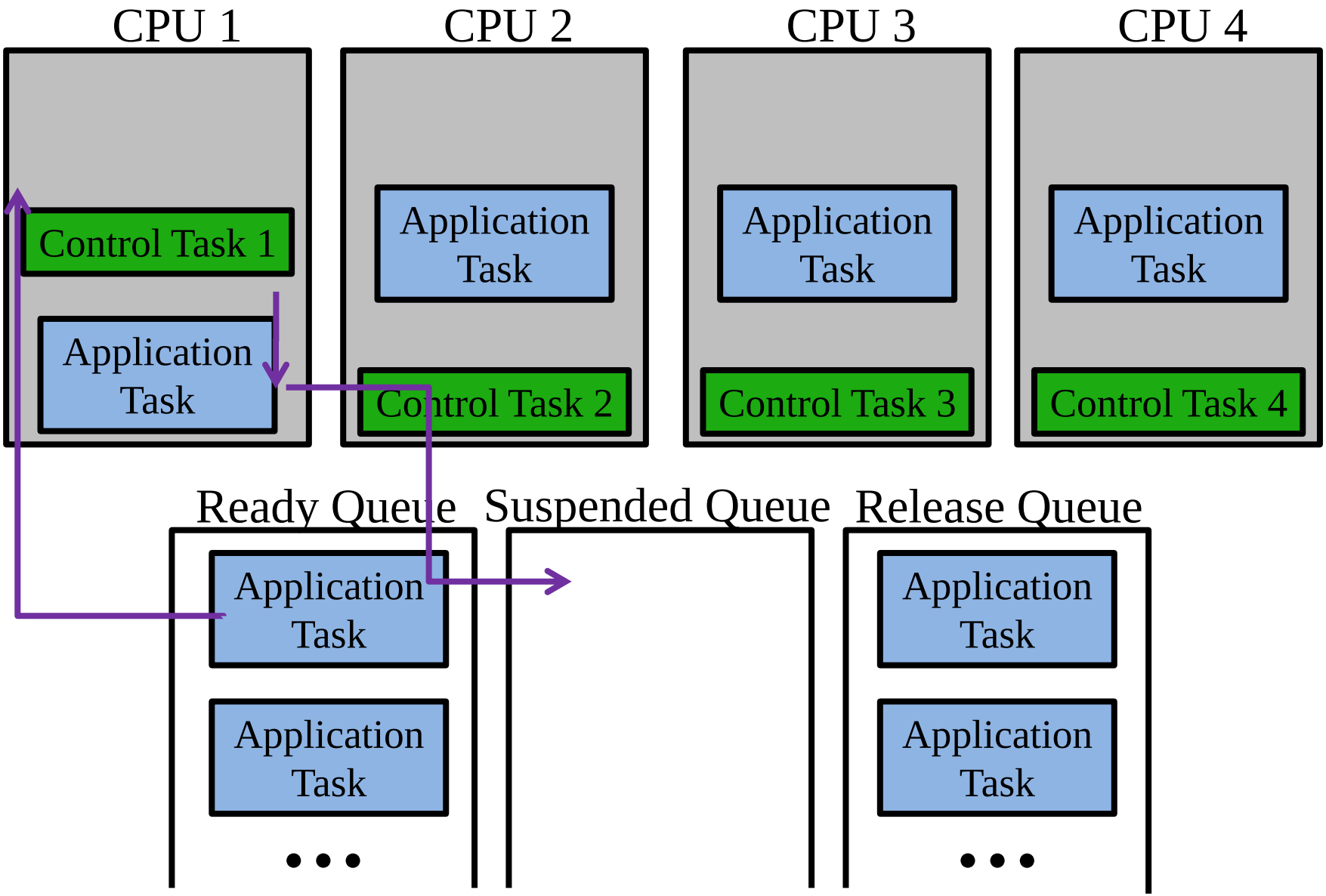

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Release Queue
Application Task
Application Task
Conductor
Application Task
Application Task

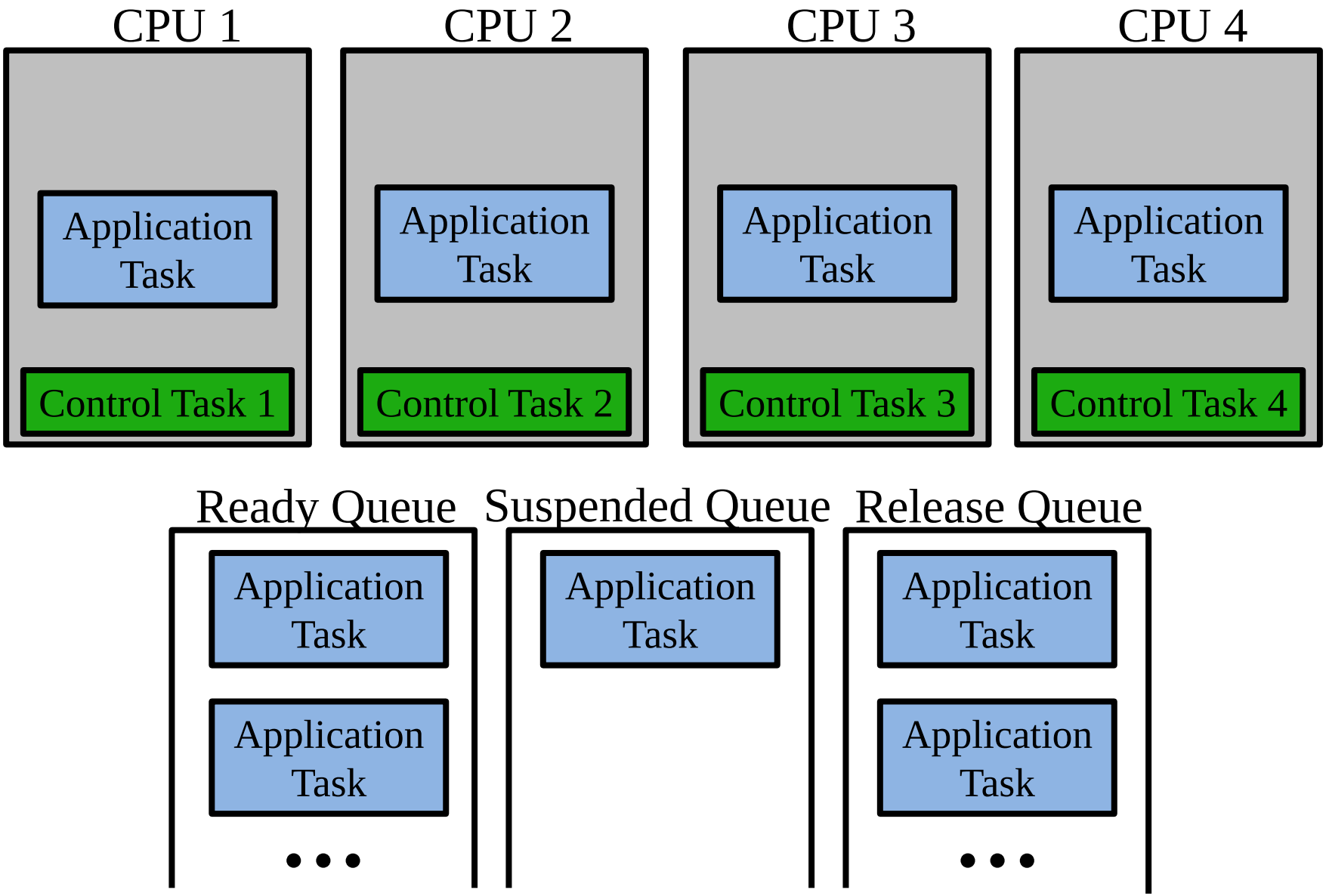

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Release Queue
Application Task
Application Task
Conductor
Application Task
Application Task
• • •
• • •

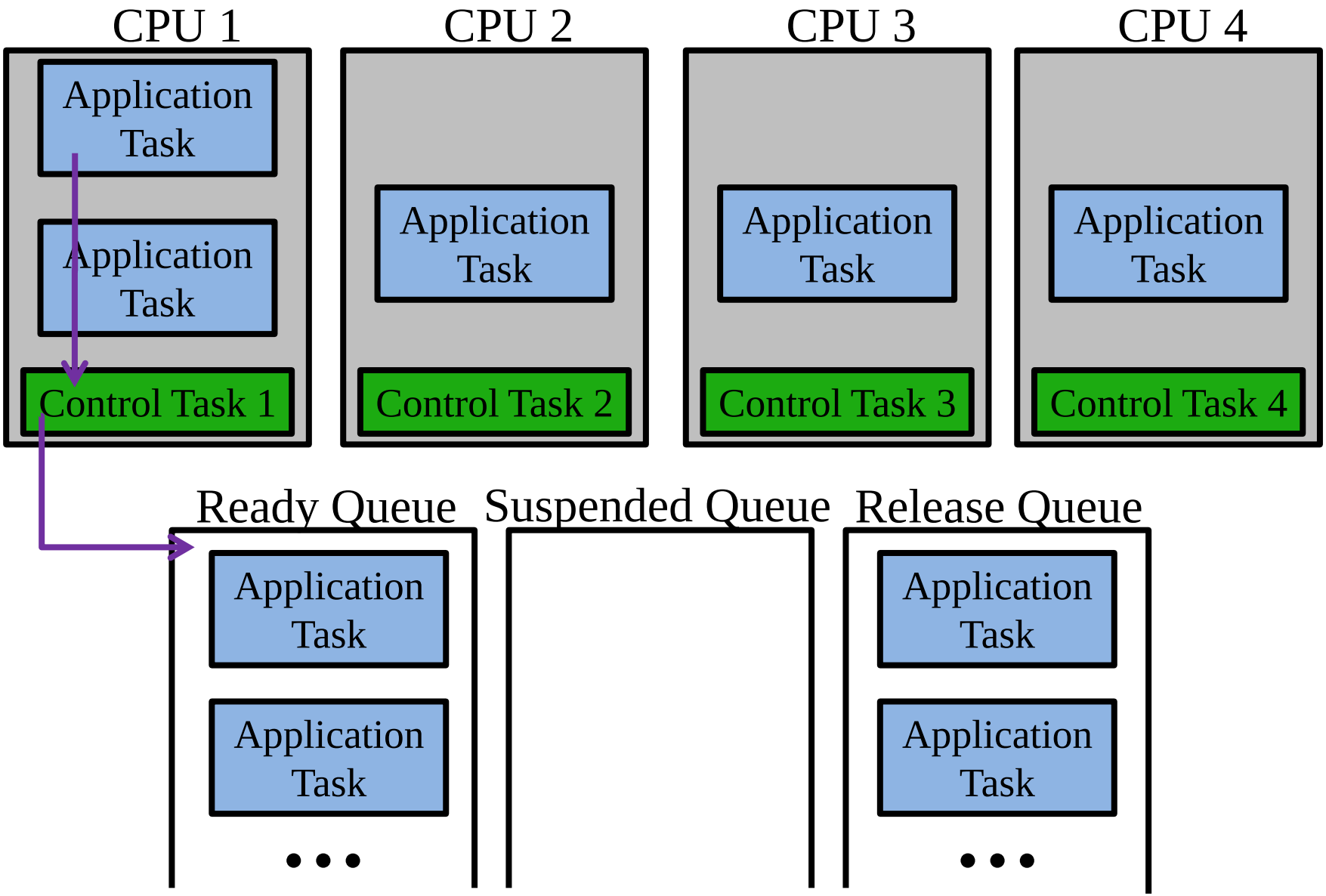

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Application Task
Application Task
• • •
Release Queue
Application Task
Application Task
• • •

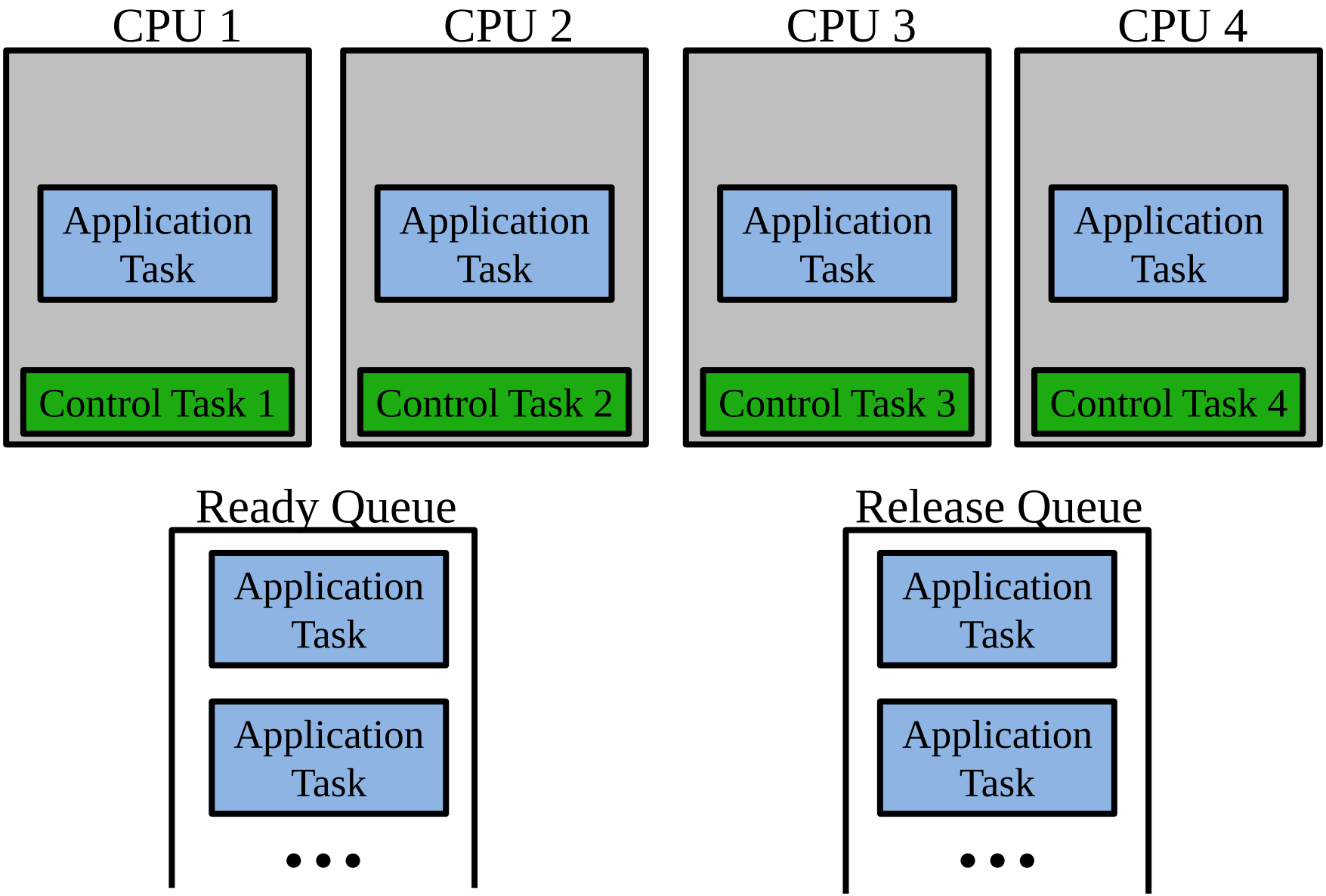

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Application Task
Application Task
• • •
Release Queue
Application Task
Application Task
• • •


CPU 1
CPU 2
CPU 3
CPU 4
Signal
Control Task 1
Application Task
Application Task
Control Task 2
Application Task
Control Task 3
Application Task
Control Task 4
Ready Queue
Application Task
Application Task
• • •
Release Queue
Application Task
Application Task
• • •


CPU 1
CPU 2
CPU 3
CPU 4
Control Task 1
Application Task
Application Task
Application Task
Application Task
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Suspended Queue
Release Queue
Application Task
Application Task
Application Task
Application Task
• • •
• • •


CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Suspended Queue
Release Queue
Application Task
Application Task
Application Task
Application Task
Application Task
Application Task
• • •
• • •


CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Suspended Queue
Release Queue
Application Task
Application Task
Application Task
Application Task
• • •
• • •

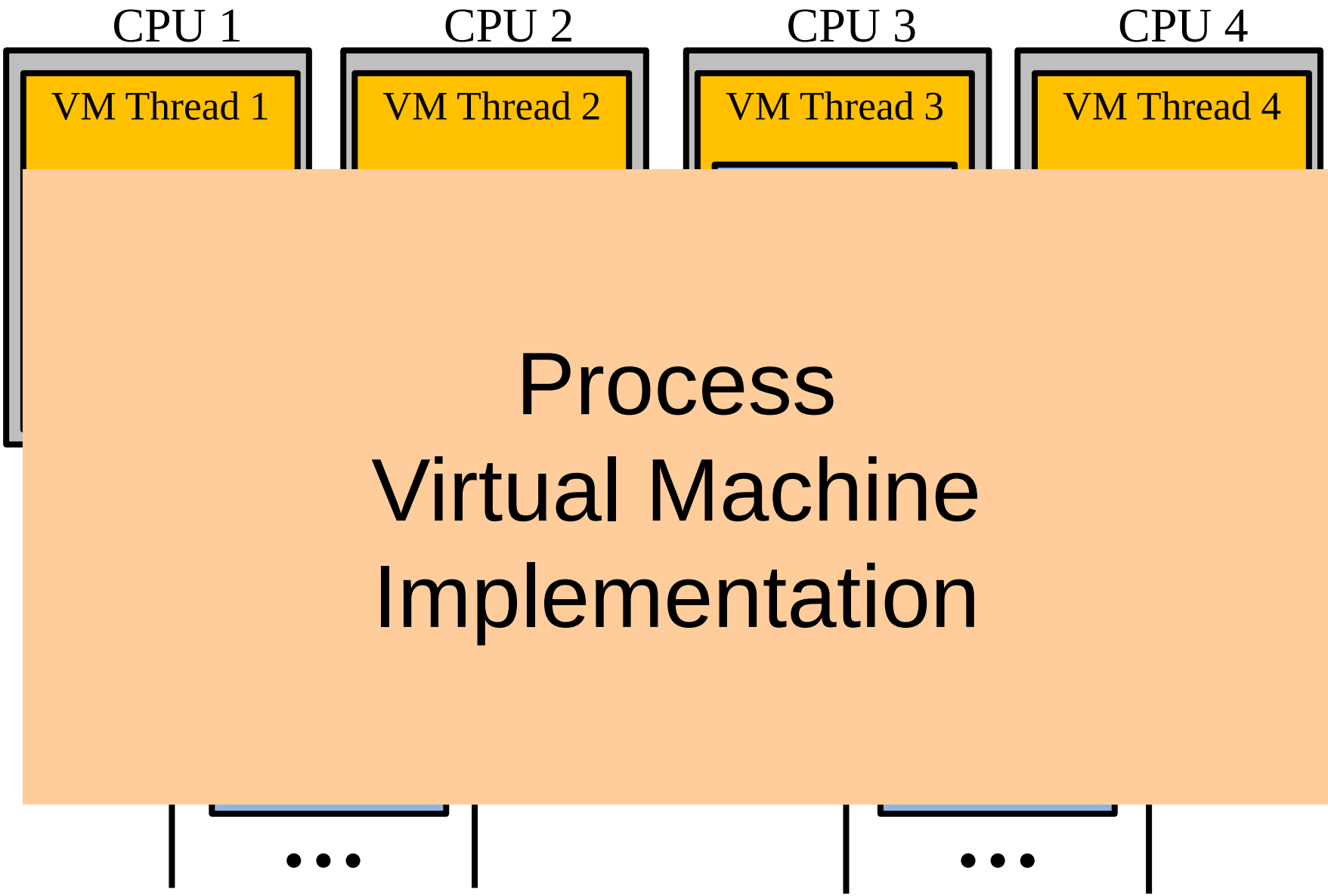

CPU 1
CPU 2
CPU 3
CPU 4
Application Task
Application Task
Application Task
Application Task
Control Task 1
Control Task 2
Control Task 3
Control Task 4
Ready Queue
Application Task
Application Task
• • •
Release Queue
Application Task
Application Task
• • •

Commentary on POSIX Impl.

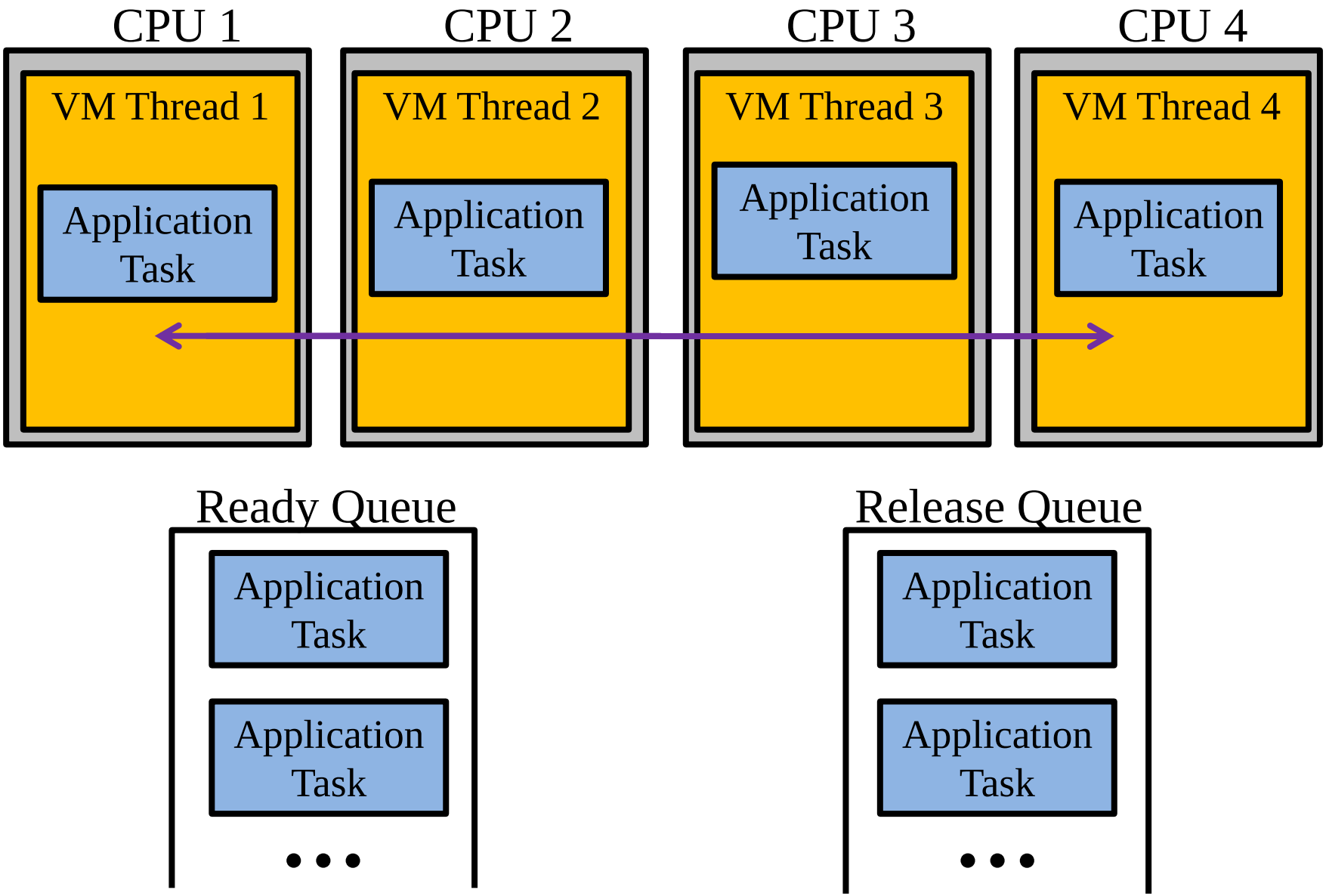
- Pros
  - Should prove easy to implement and customize
- Cons
  - Sensitive to POSIX signal implementation
  - Probably not robust enough for a many applications

Improving this Implementation

- Real-time tasks as *user threads* (CPU contexts)
  - 10x faster context switch (typically)
- Execute them inside non-migrating OS threads.
- Still has some robustness issues.


CPU 1
CPU 2
CPU 3
CPU 4
VM Thread 1
VM Thread 2
VM Thread 3
VM Thread 4
Process
Virtual Machine
Implementation
…
…

VMs coordinate amongst themselves



Comparison

- Process VM impl. has the potential to reap the gains from user-level threads, while maintaining robustness
  - More “elegant” solution
  - Much more implementation work
- I “prefer” the VM design, but there are, no doubt, pros and cons to each.

Thank You

Questions and Feedback

For upcoming paper or to contact me:

http://cs.unc.edu/~mollison

mollison@cs.unc.edu