

CS 112/212 Project 1: Threads

Spring 2025



Getting Started

Pre-Reading

Before you read the description of this project, you should read all of the following:

- 1. Introduction, C. Coding Standards, E. Debugging Tools, and F. Development Tools.
- A.1 Loading through A.5 Memory Allocation
 - especially A.2 Threads, A.3 Synchronization, A.4 Interrupt Handling
- B. 4.4BSD Scheduler (read before implementing 2.2.4 Advanced Scheduler)

and *then* the assignment description/FAQ.

Understand how timer interrupts, sleeping, and synchronization work.

Look over + draft the design doc for each section *before* coding it!

General Tips

- Small, incremental changes
 - Go from working state to working state
- Merge changes at each step rather than waiting until the end
- **Pay attention to style; conform to the style of the starter code**
 - Make sure each line is ≤ 80 characters for easier readability
- **!!! Design, design, design (50% of your grade) !!!**
- Brush up on **GDB/debugging** (section E in previous slide) and **git**
- `lib/kernel` has nice data structures (for proj1, especially linked list)
- Check out Poojan's guide on Ed!

Code TODOs

1. Alarm Clock
 - a. Re-implement `timer_sleep()` without busy waiting
2. Priority Scheduler
 - a. Threads set their own priorities, and run according to these priorities
 - b. Priority donation for locks
3. Advanced Scheduler
 - a. Thread priorities are calculated by the system, and run according to these priorities
 - b. No priority donation
4. Design Doc
 - a. Answer questions regarding your design and implementation for parts 1-3

How much code?

```
devices/timer.c      |   42 +++++-
threads/fixed-point.h |  120 ++++++ (new file)
threads/synch.c      |   88 ++++++
threads/thread.c     |  196 ++++++-----
threads/thread.h     |   23 +++

5 files changed, 440 insertions(+), 29 deletions(-)
```

Alarm Clock

Alarm Clock Tasks (Section 2.2.2)

- Currently, `timer_sleep()` “busy waits”
 - if a thread wants to suspend execution for some time, it doesn’t actually go to sleep; execution just temporarily yields to other threads, but the “sleeping” thread will repeatedly loop/execute to check if the time is up yet (bad)
- You will implement the actual “sleeping” functionality and eliminate busy-waiting

Alarm Clock Considerations

- **How** will you avoid busy waiting?
- How will you keep track of **sleeping threads**?
- Where in the code will you **wake up** sleeping threads?
- Check out the design doc to see what **race conditions** to watch out for!

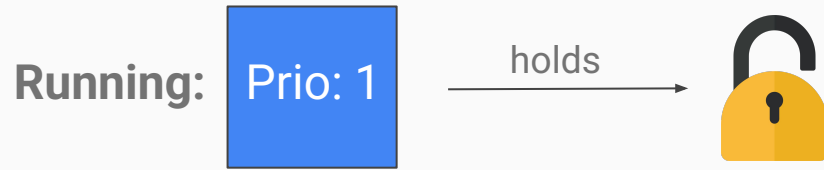
Priority Scheduling

Priority Scheduling Tasks (Section 2.2.3)

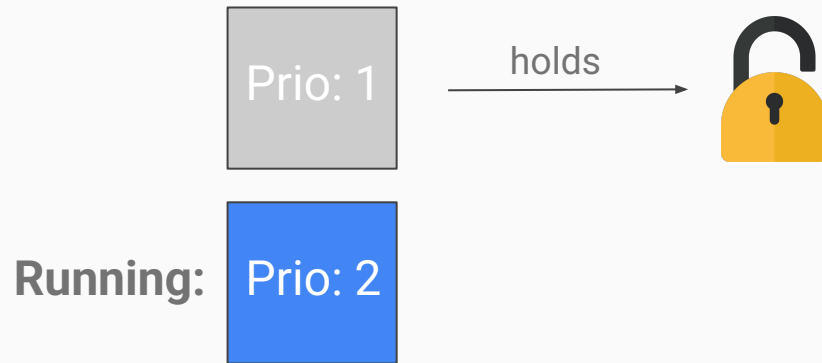
- Rather than just switching among all threads equally (round-robin), introduce a notion of **priority**: higher-priority threads get run before lower-priority threads, until higher-priority threads can't run
- Priority scale [0, 63]; round-robin threads *of the same priority*
- When the currently running thread stops being the highest priority (either due to a new thread being added or the current thread's priority being lowered), the kernel should immediately switch to executing the highest-priority thread that is ready
- When threads are waiting for a lock, semaphore, or condition variable, the **highest priority** waiting thread should be **awakened first**

Problem: priority inversion

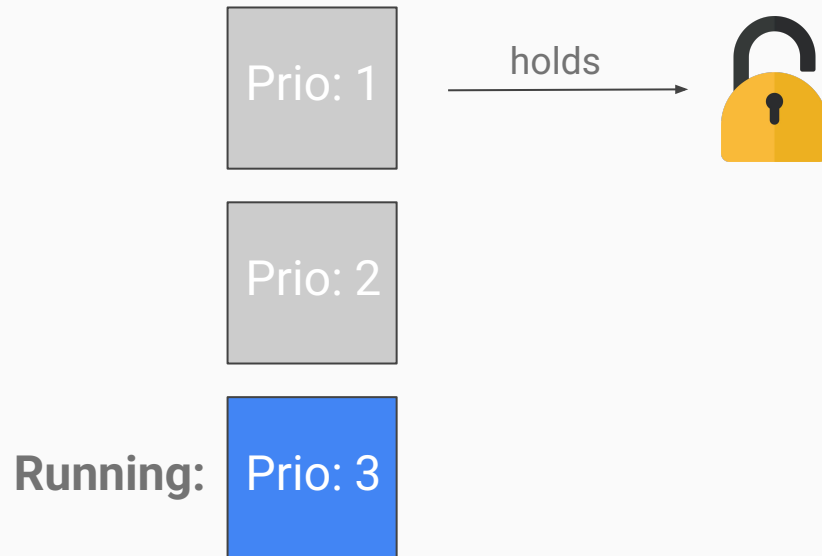
Priority Inversion



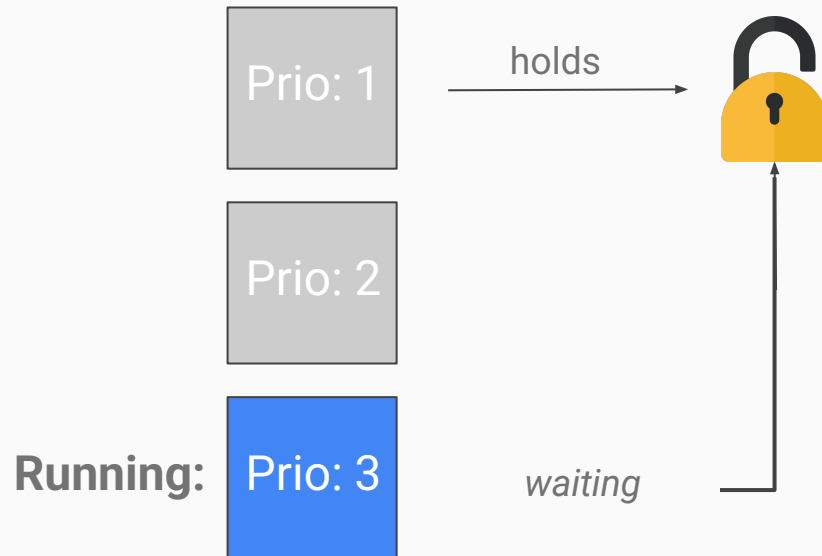
Priority Inversion



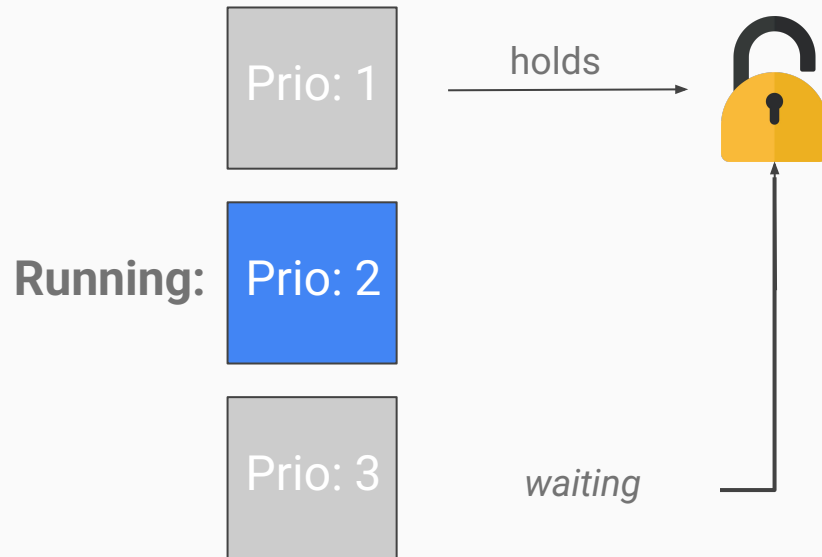
Priority Inversion



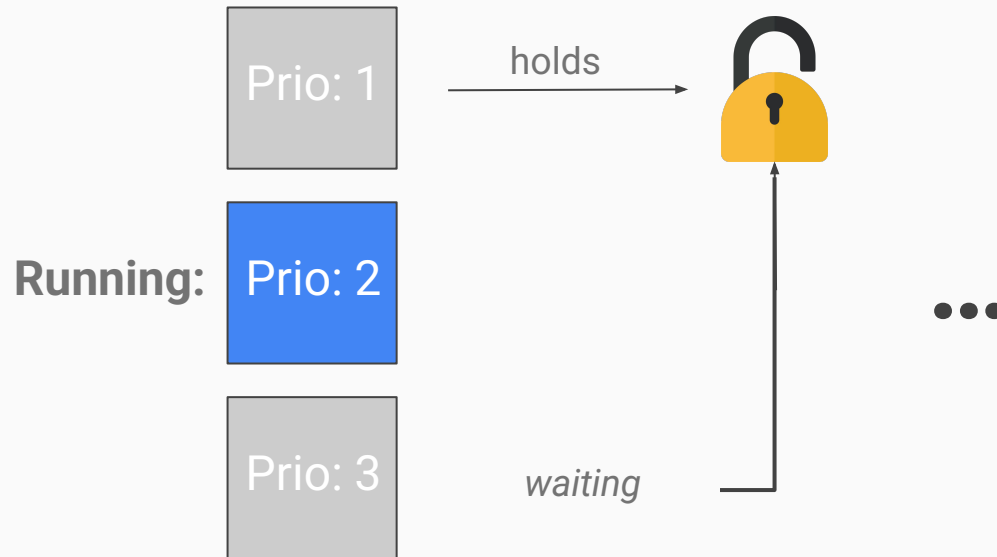
Priority Inversion



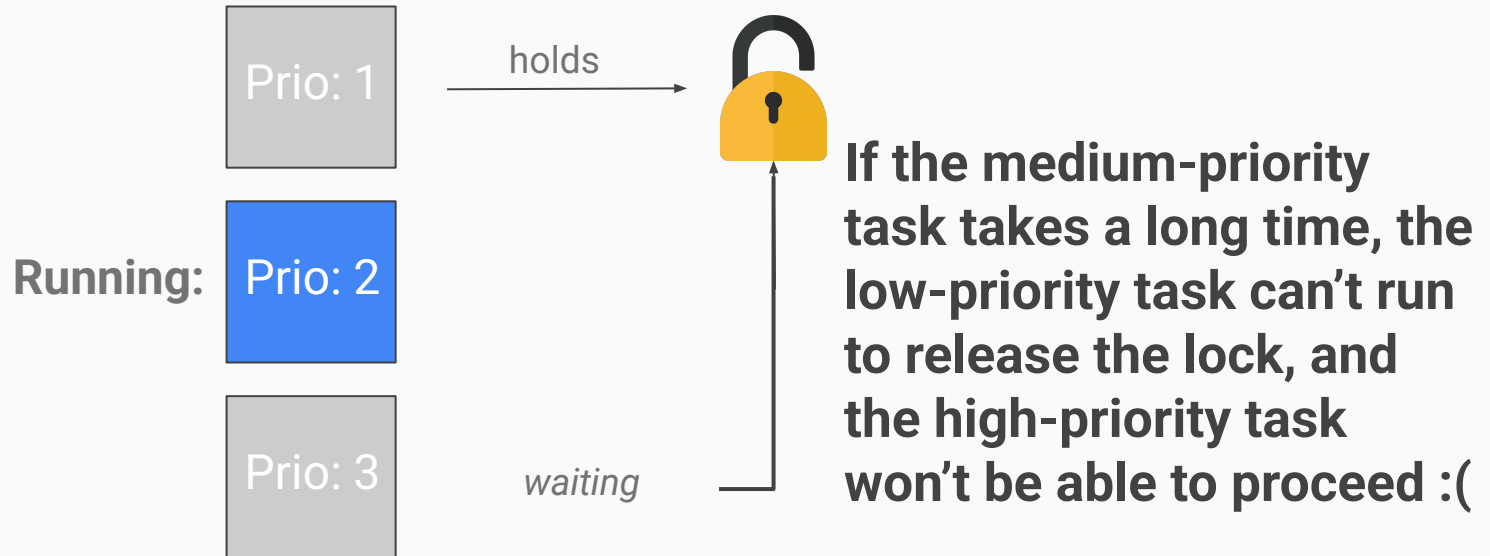
Priority Inversion



Priority Inversion

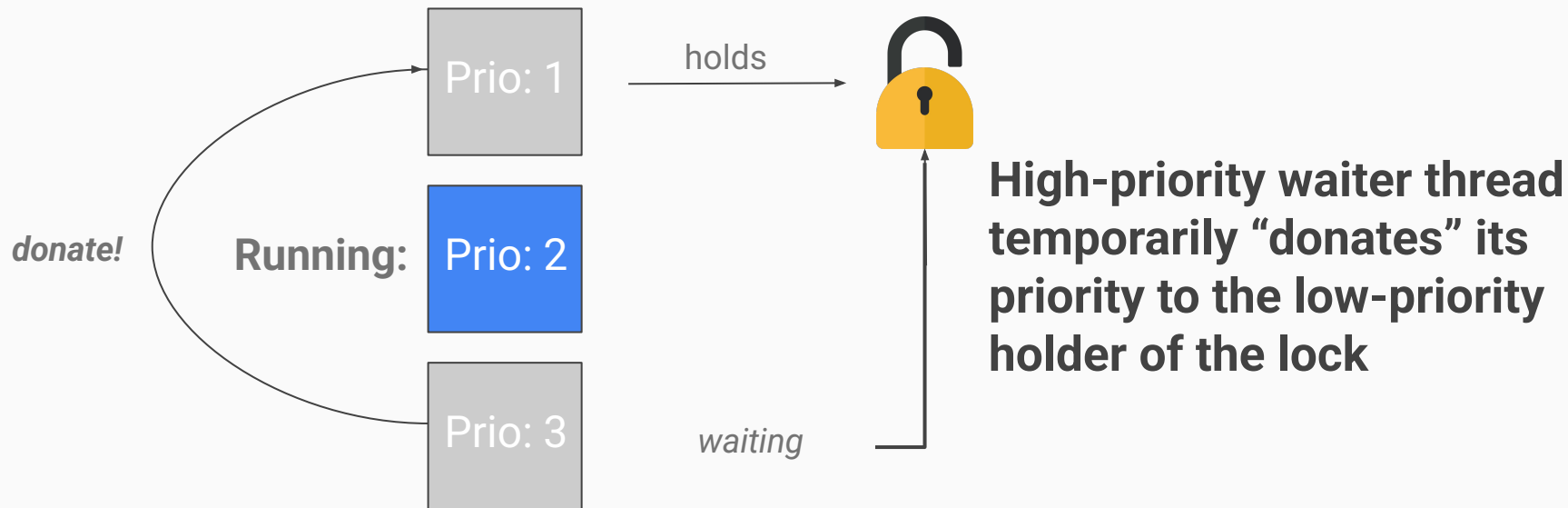


Priority Inversion

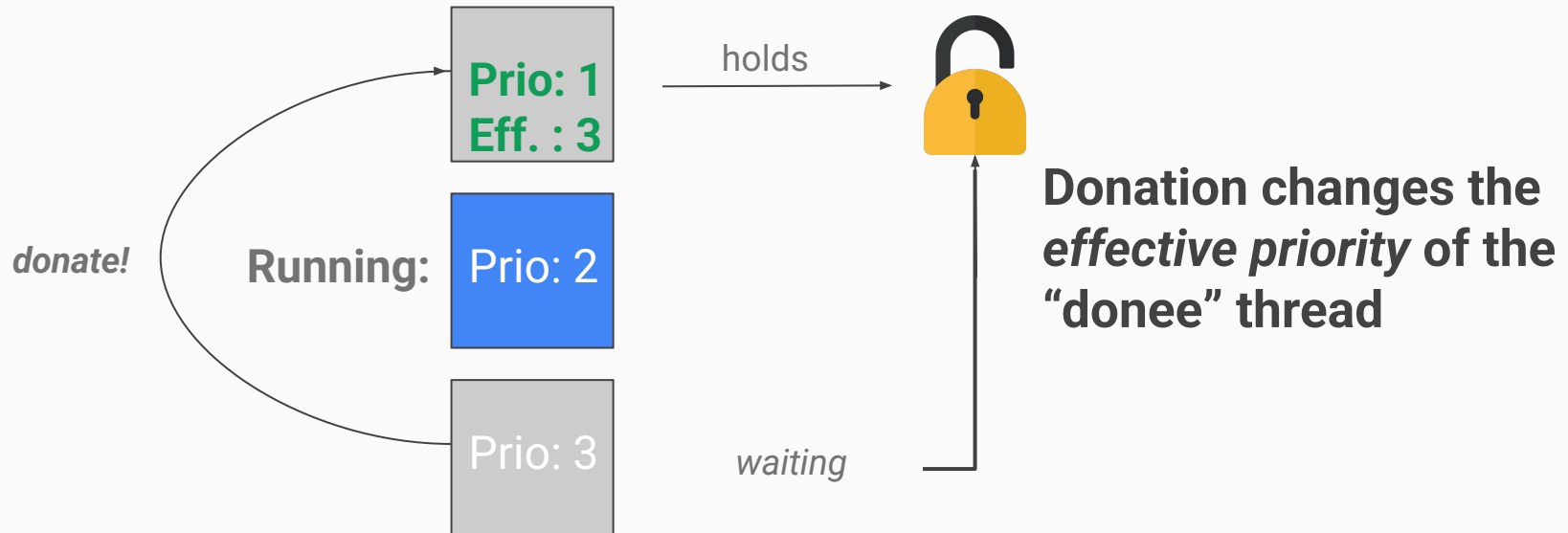


Solution: priority donation

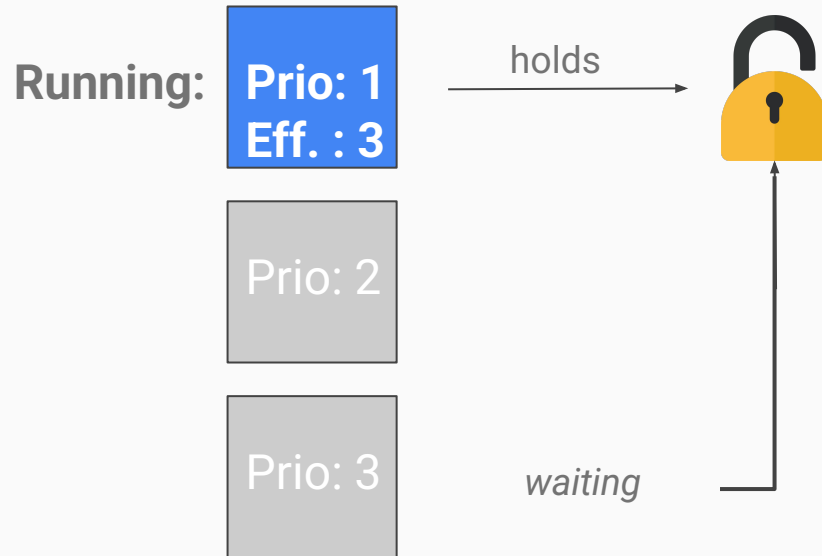
Priority Donation



Priority Donation



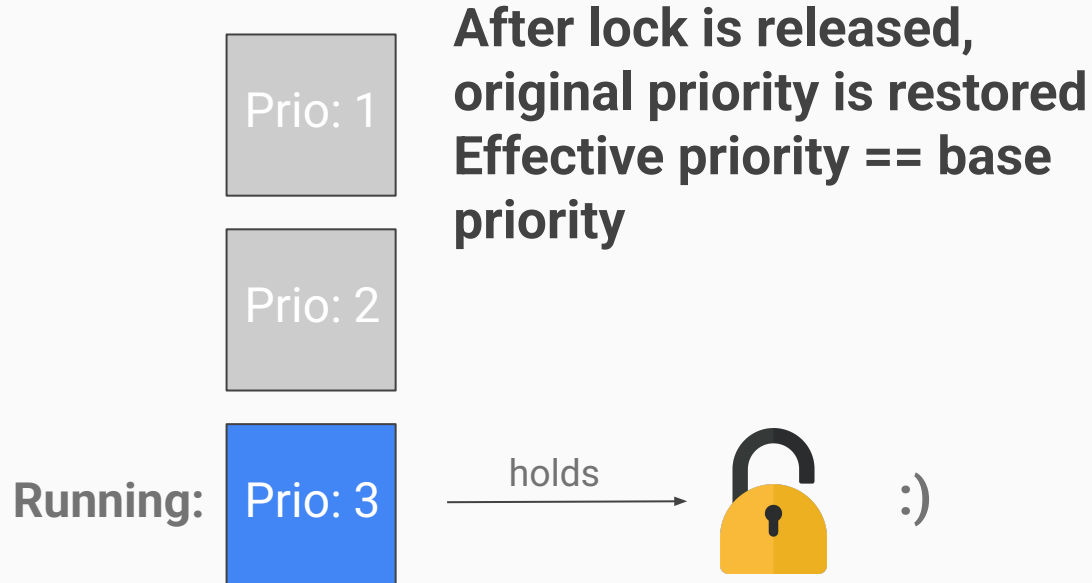
Priority Donation



Priority Donation



Priority Donation



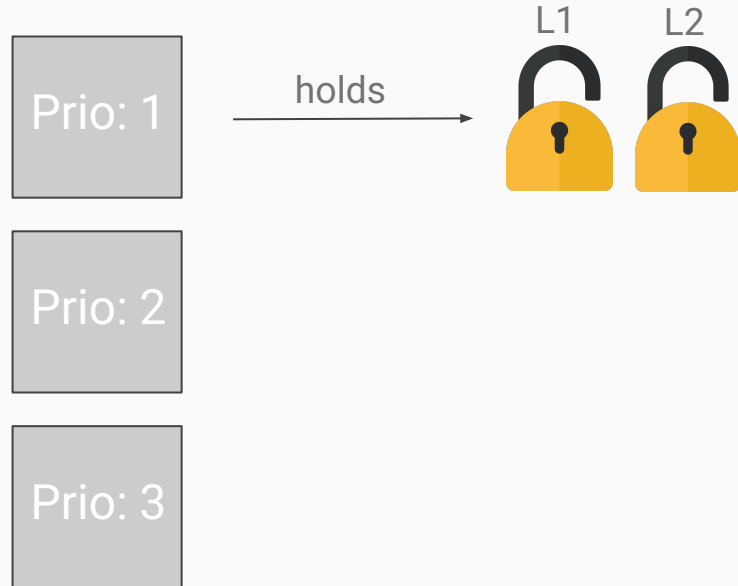
Priority Donation: More Cases

- To how many threads can a donor donate its priority?
- From how many threads may a donee receive priority?
- What happens when a priority recipient *itself* donates to another thread?

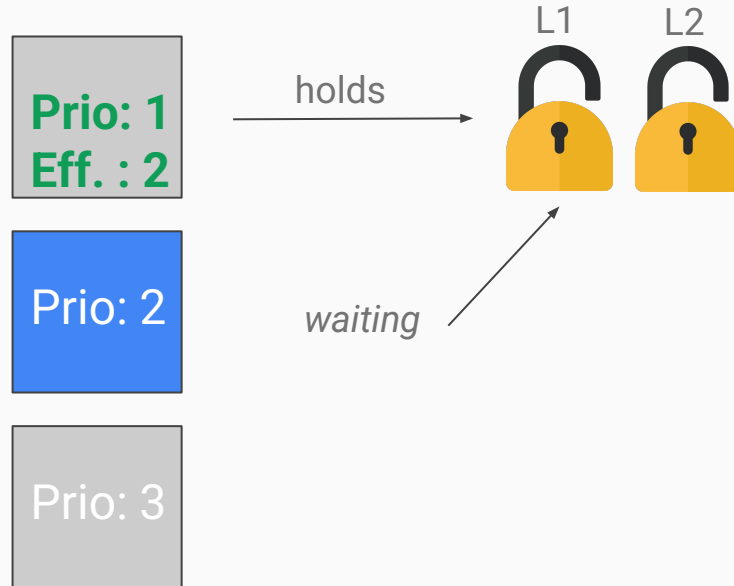
Priority Donation: More Cases

- To how many threads can a donor donate its priority?
- **From how many threads may a donee receive priority?**
- What happens when a priority recipient *itself* donates to another thread?

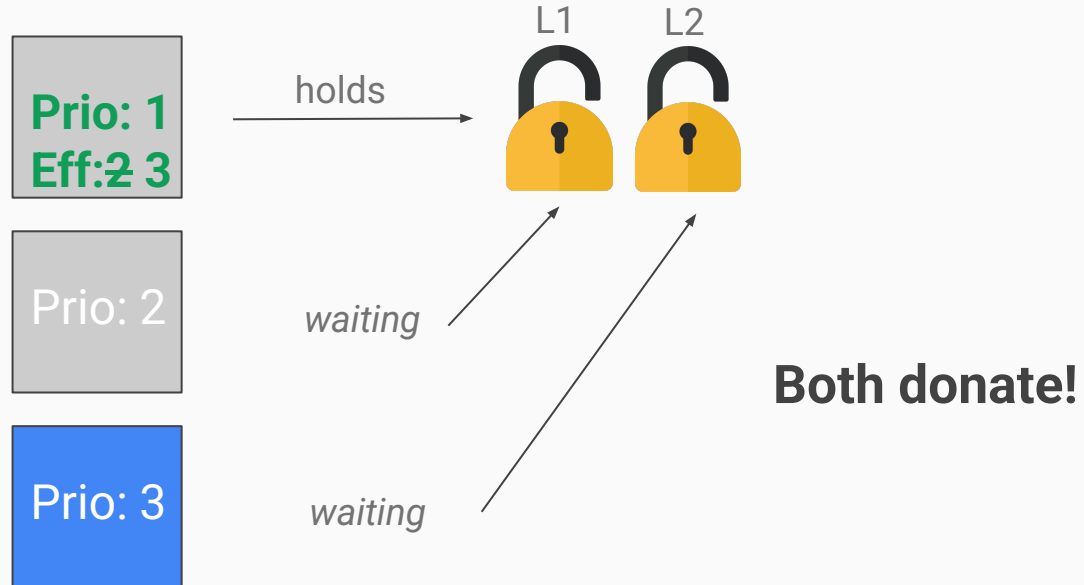
Priority Inversion: *Multiple* Donation



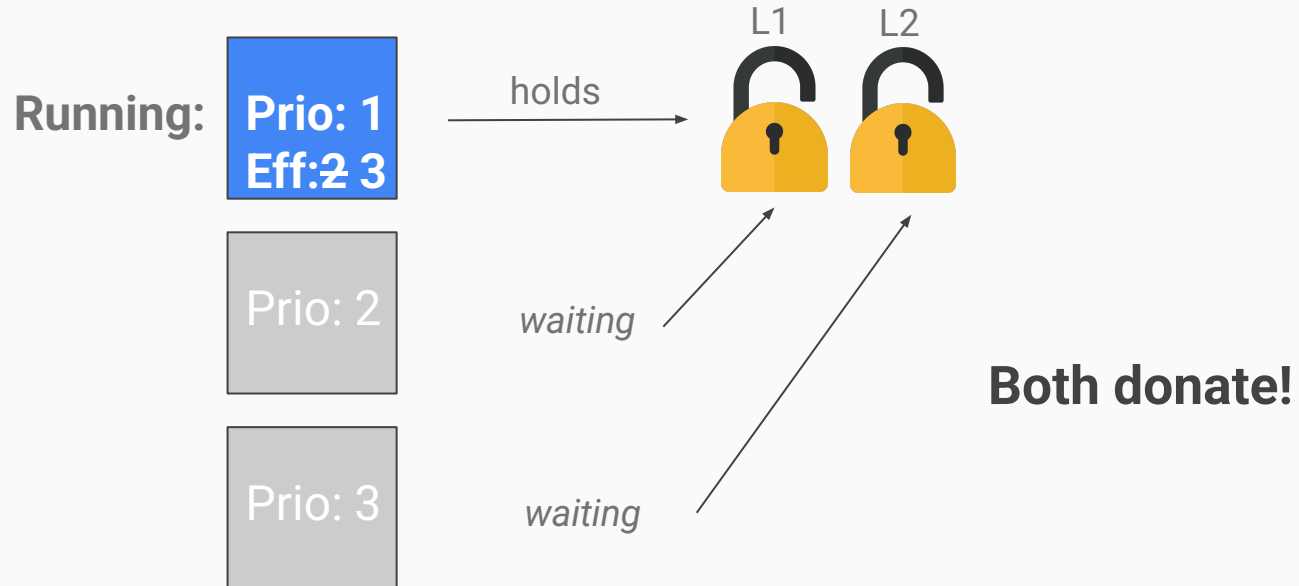
Priority Inversion: *Multiple Donation*



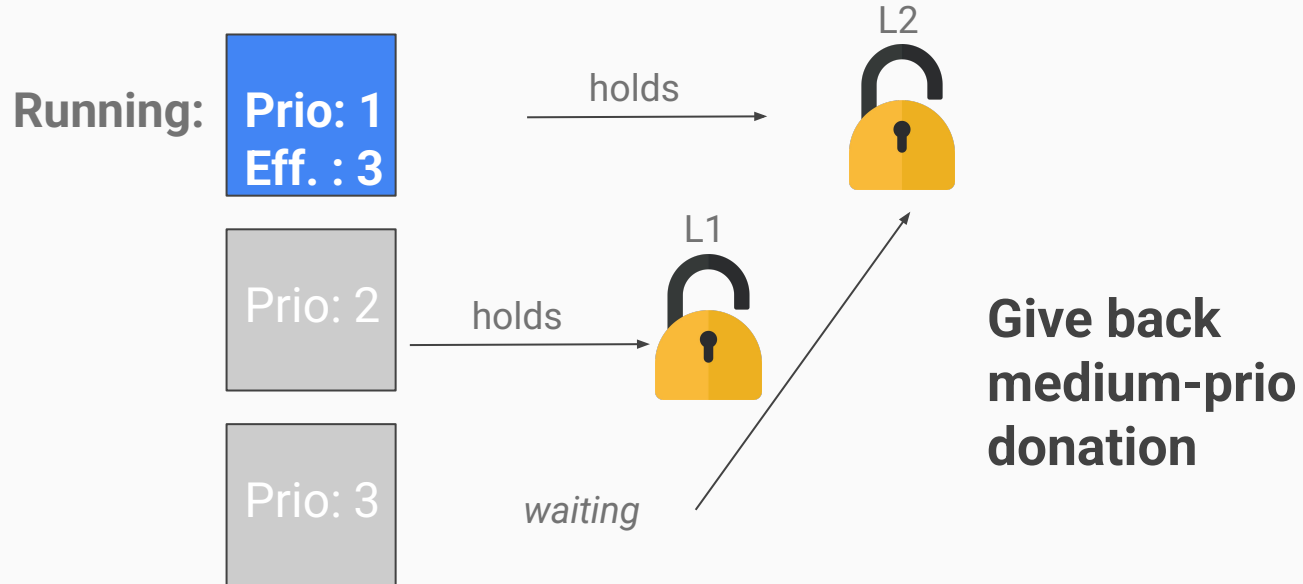
Priority Inversion: *Multiple Donation*



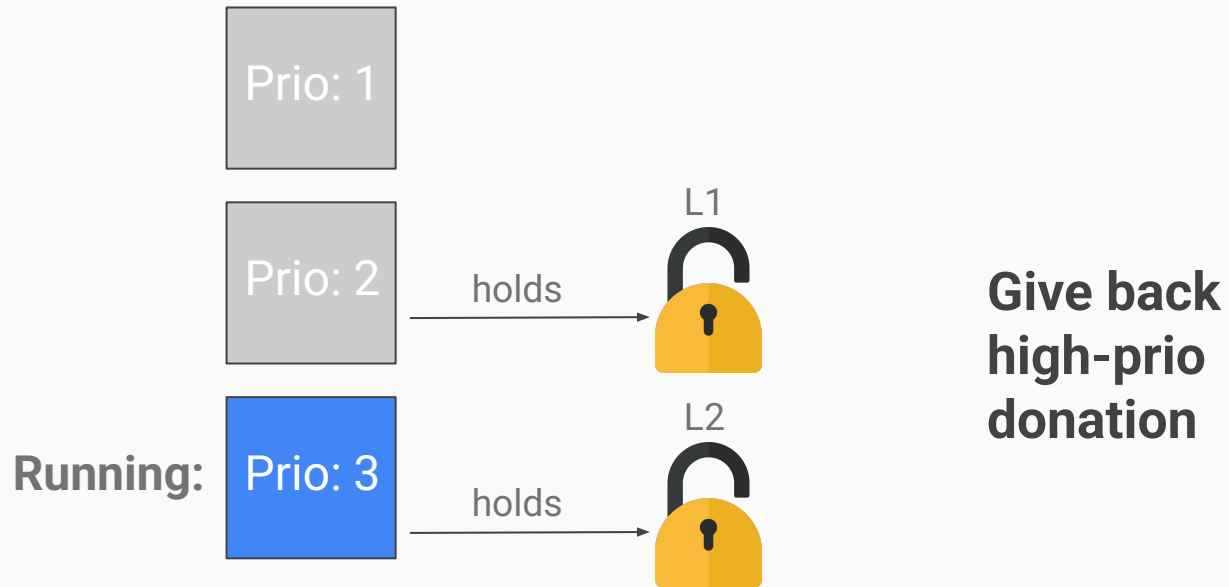
Priority Inversion: *Multiple Donation*



Priority Inversion: *Multiple* Donation



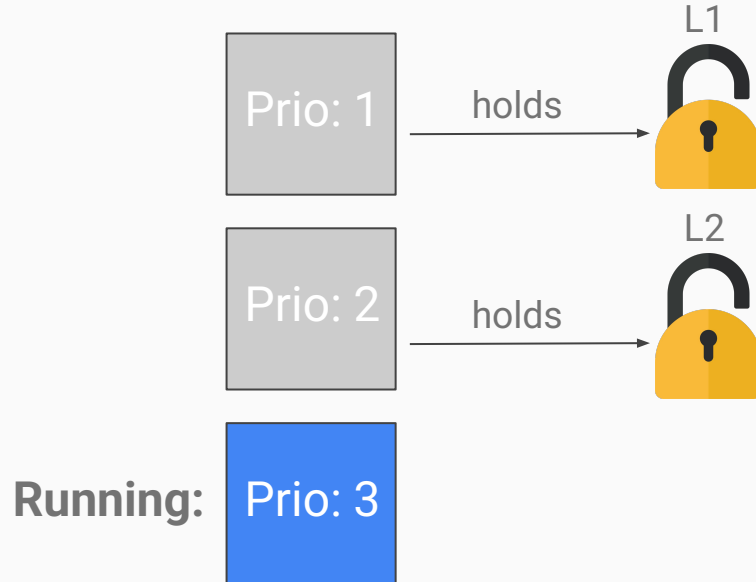
Priority Inversion: *Multiple* Donation



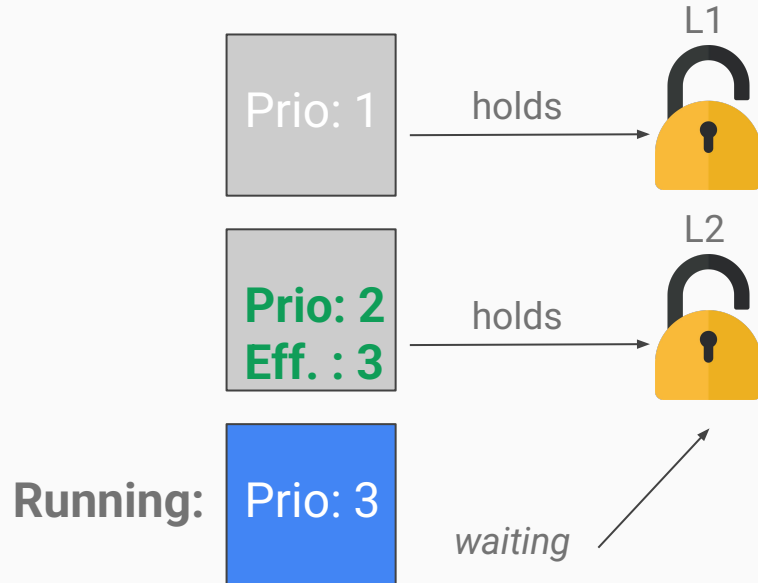
Priority Donation: More Cases

- To how many threads can a donor donate its priority?
- From how many threads may a donee receive priority?
- **What happens when a priority recipient *itself* donates to another thread?**

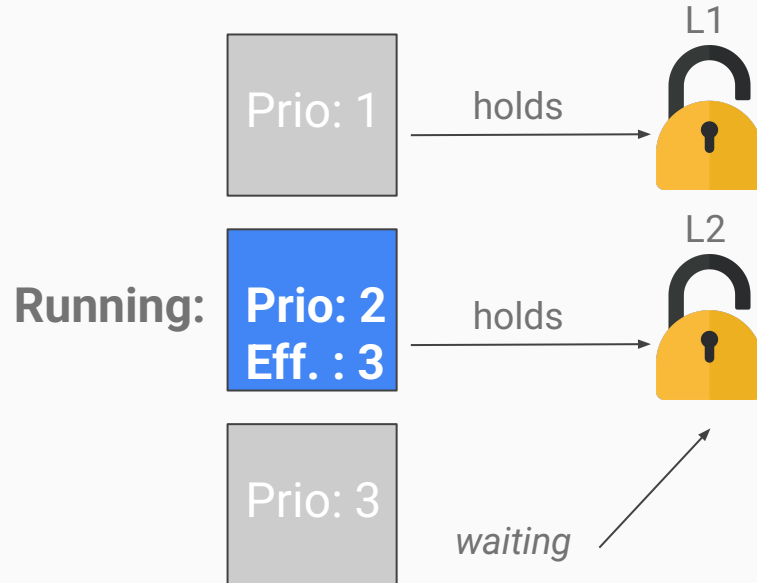
Priority Inversion: *Nested* Donation



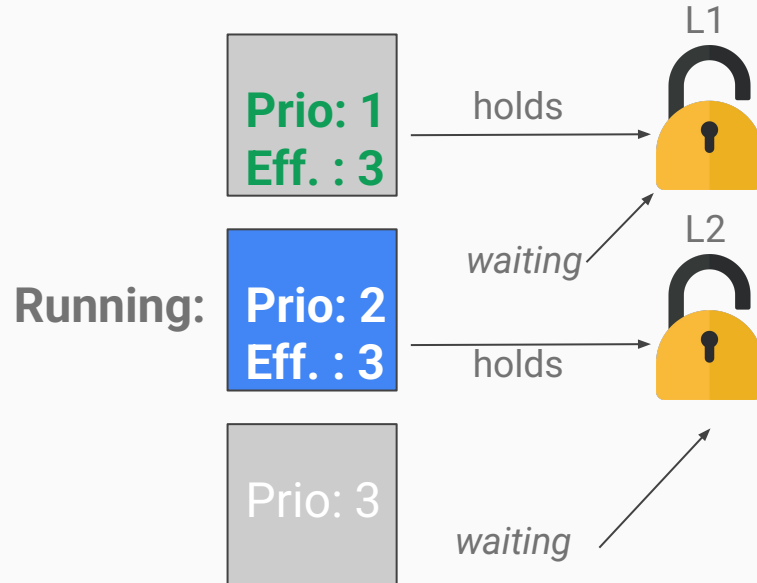
Priority Inversion: *Nested* Donation



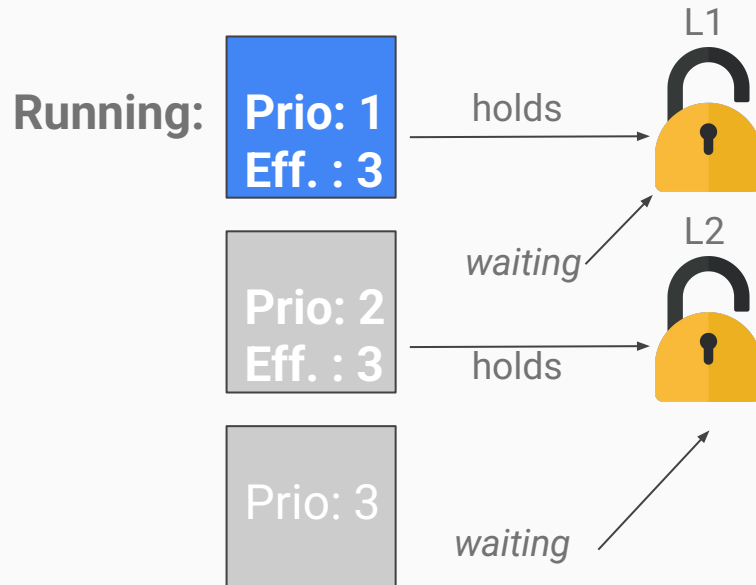
Priority Inversion: *Nested* Donation



Priority Inversion: *Nested* Donation

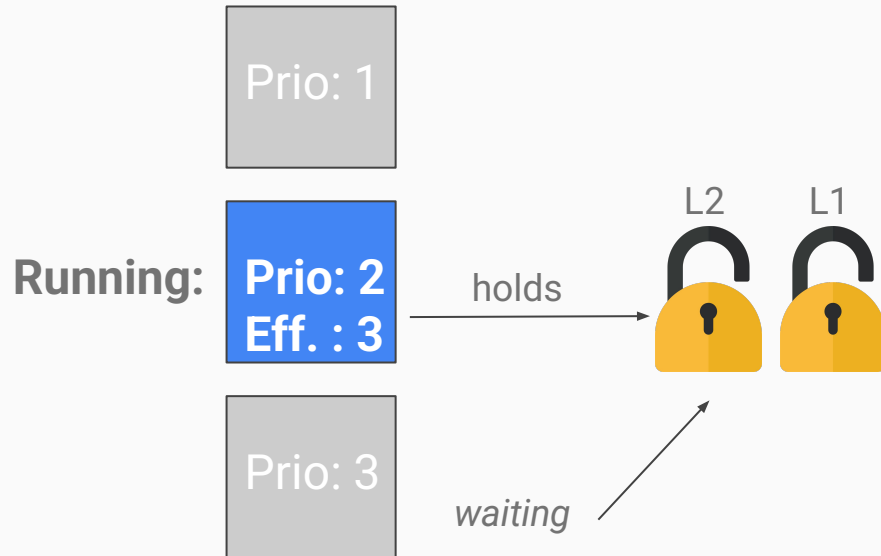


Priority Inversion: *Nested* Donation

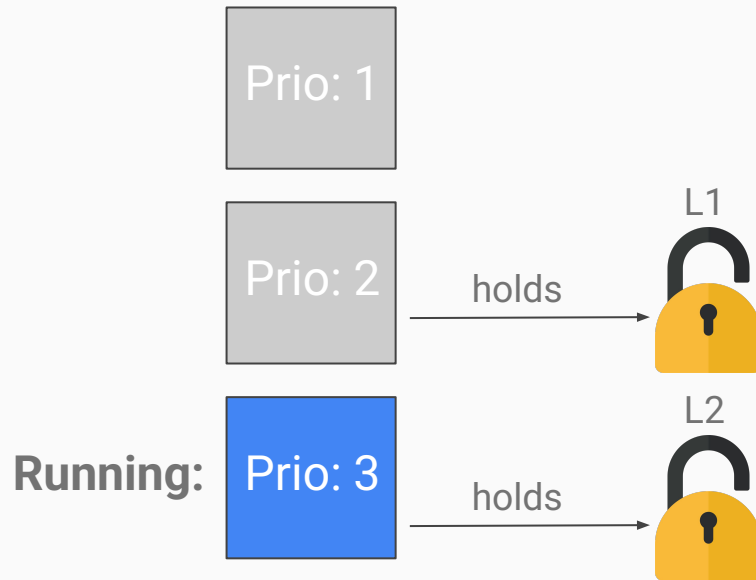


**Note: You may impose a reasonable limit on depth of nested priority donation, such as 8 levels*

Priority Inversion: *Nested* Donation



Priority Inversion: *Nested* Donation



Priority Scheduling Considerations

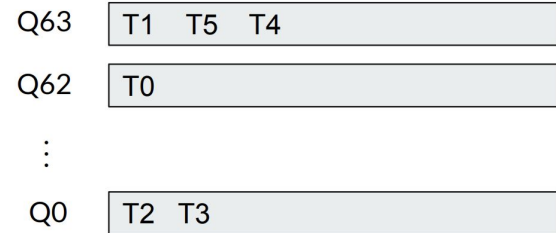
- Data structure for donations? Where should it go?
- When is a donation executed? When is it “un-donated” (when is the donee’s original priority restored)?
- How will you ensure that the highest priority thread waiting for a lock, semaphore, or condition variable is woken up?

Advanced Scheduler (MLFQ)

Multi-Level Feedback Queue

([Section 2.24](#) and [Appendix B](#))

- Still based on priorities, but **no** priority donation
 - So we recommend that before starting this part, you have a working version of the basic priority scheduler (except possibly priority donation)
- Maintain several queues, each representing a different priority
 - Pick from highest-priority non-empty queue
 - Round-robin within each queue
- Scheduler dynamically updates priorities
 - “Niceness” = how generous to be to other threads
 - $\text{priority} = \text{PRI_MAX} - (\text{recent_cpu} / 4) - (\text{nice} * 2)$
 - Scheduler updates priority every 4th tick if recent_cpu changed (recent_cpu calculated every 1 sec)
- Algorithm tries to boost priority of threads that have used less CPU time recently



Note: Fixed-Point Arithmetic

- Detailed formulas for Advanced Scheduler's priority calculation and related metrics in Appendix B (uses both integers and real numbers)
- Can't use actual floating-point arithmetic: too slow for kernel code
- Carefully approximate floating-point calculations with integers: implementation also detailed in appendix

TL;DR

Reminders (TL;DR part 1)

- Do the pre-readings first
- Give **lots** of attention to design
- If some of these examples were unclear:
 - The appendix/Pintos docs are very detailed
 - More about synchronization & scheduling in next week's lectures
 - Come to Office Hours or post on Ed!

Code Tasks (TL;DR part 2)

1. Alarm Clock
 - a. Re-implement `timer_sleep()` without busy waiting
2. Priority Scheduler
 - a. Threads set their own priorities, and run according to these priorities
 - b. Priority donation for locks
3. Advanced Scheduler
 - a. Thread priorities are calculated by the system, and run according to these priorities
 - b. No priority donation
4. Design Doc
 - a. Answer questions regarding your design and implementation for parts 1-3