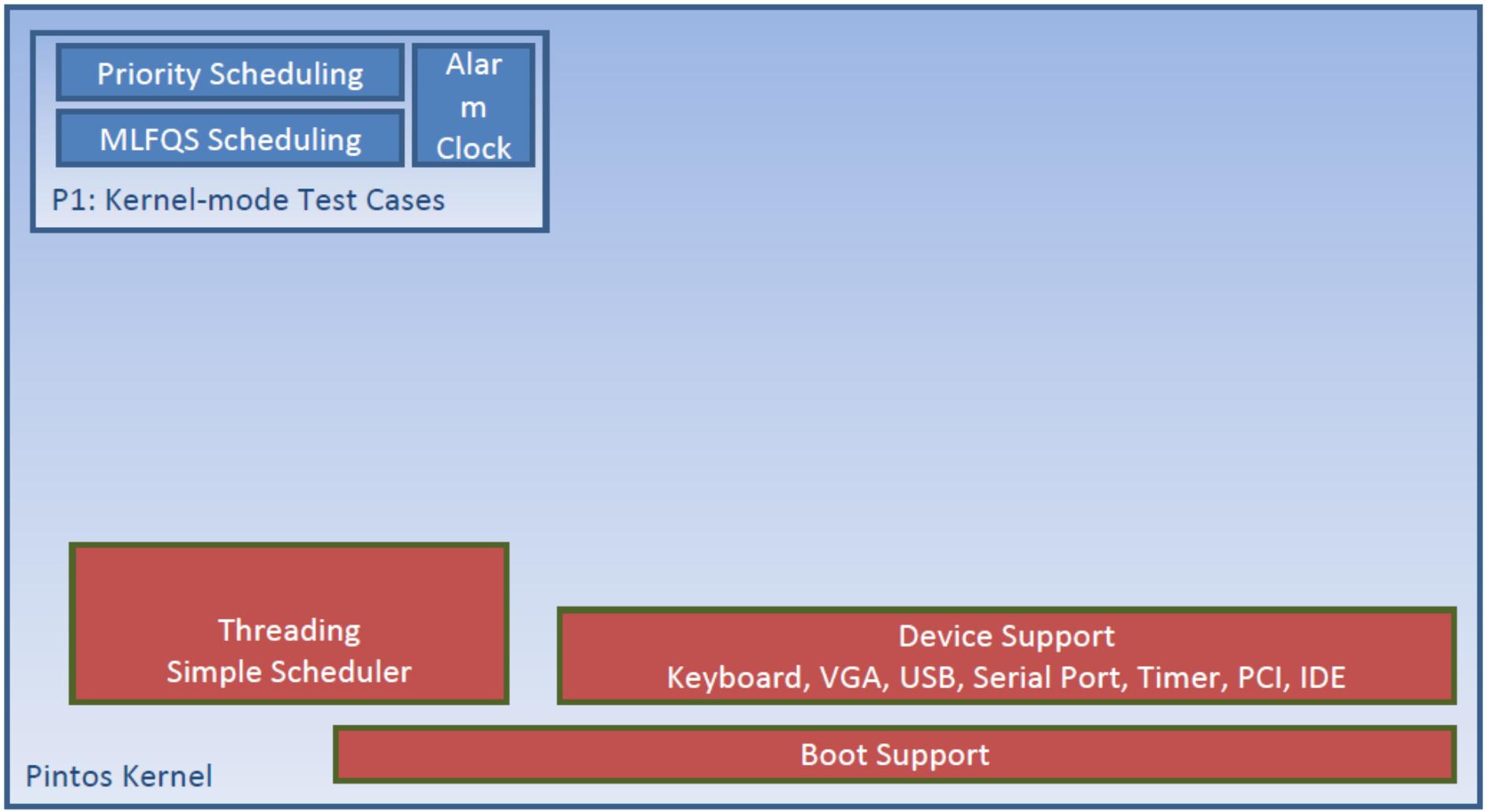
Project 1: Thread Scheduling

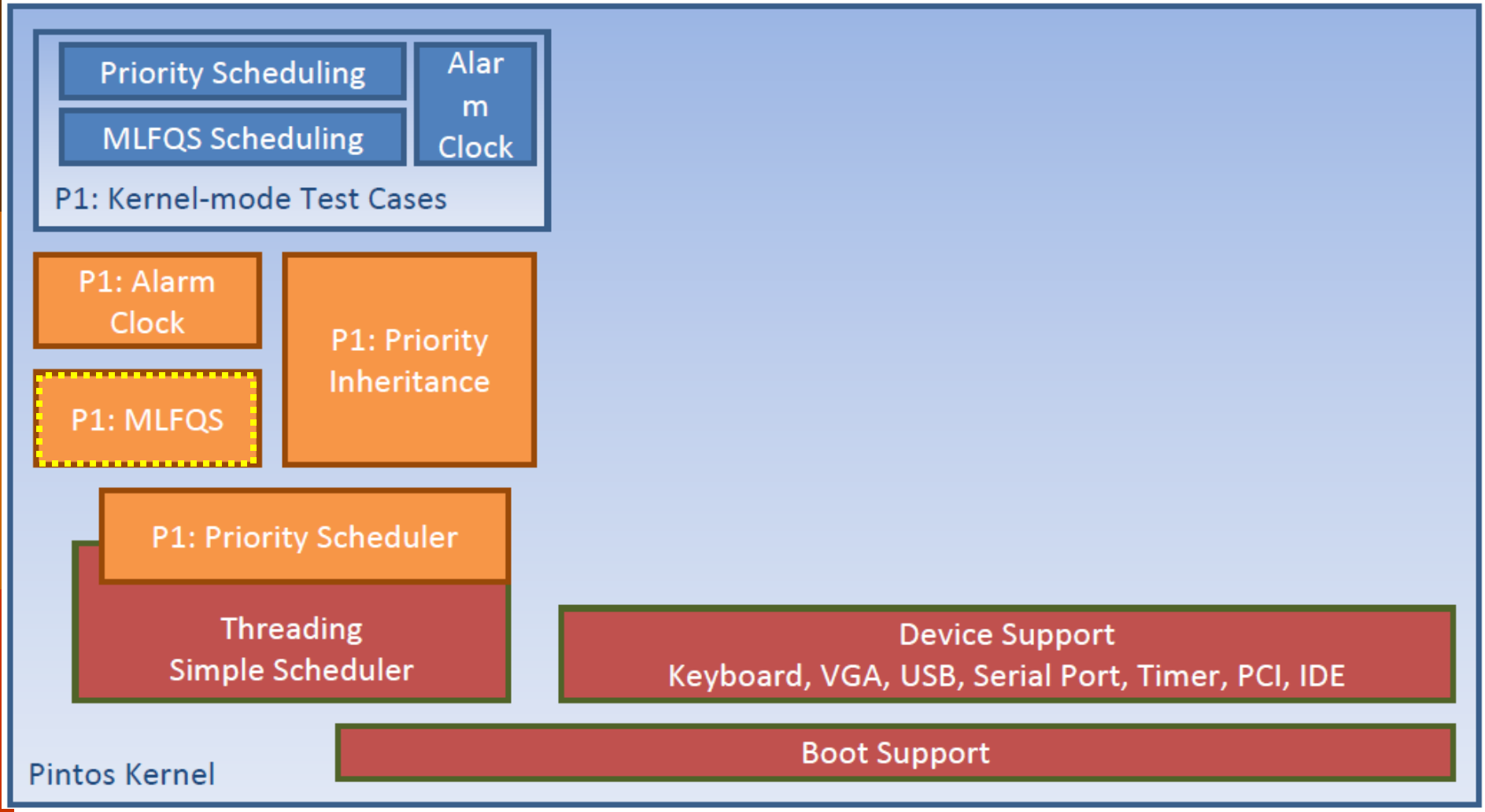
Project 1 Overview

- Extend the functionality of a minimally functional thread system
- Implement
  - Alarm Clock
  - Priority Scheduling
    - Including priority inheritance
  - Advanced MLFQ Scheduler [Extra Credit]

Project 1: Components



Project 1: Components

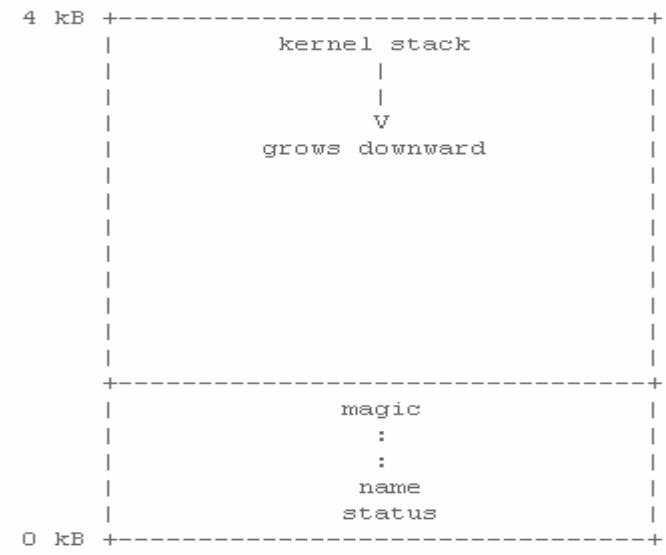


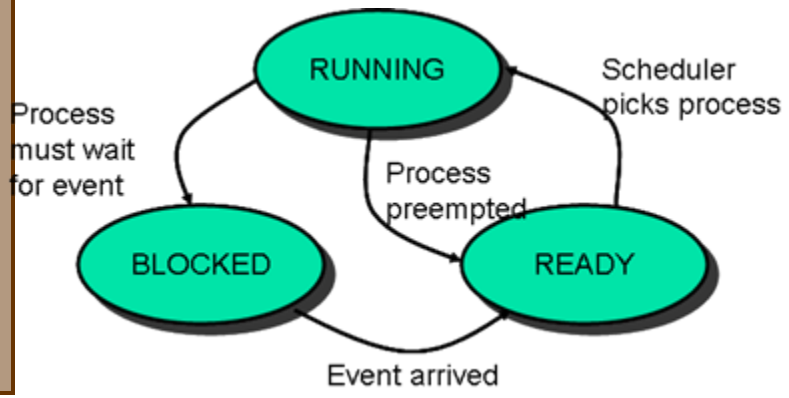
Pintos Thread System

```
struct thread
  {
    tid_t tid;            /* Thread identifier. */
    enum thread_status status;        /* Thread state. */
    char name[16];    /* Name (for debugging purposes). */
    uint8_t *stack;    /* Saved stack pointer. */
    int priority;       /* Priority. */
    struct list_elem allelem;  /* List element for all-threads list.*/
    /* Shared between thread.c and synch.c. */
    struct list_elem elem;            /* List element. */

You add more fields here as you need them.

#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;                /* Page directory. */
#endif
    /* Owned by thread.c. */
    unsigned magic;  /* Detects stack overflow. */
  };
```





Alarm Clock

- Reimplement timer_sleep( ) in devices/timer.c without busy waiting

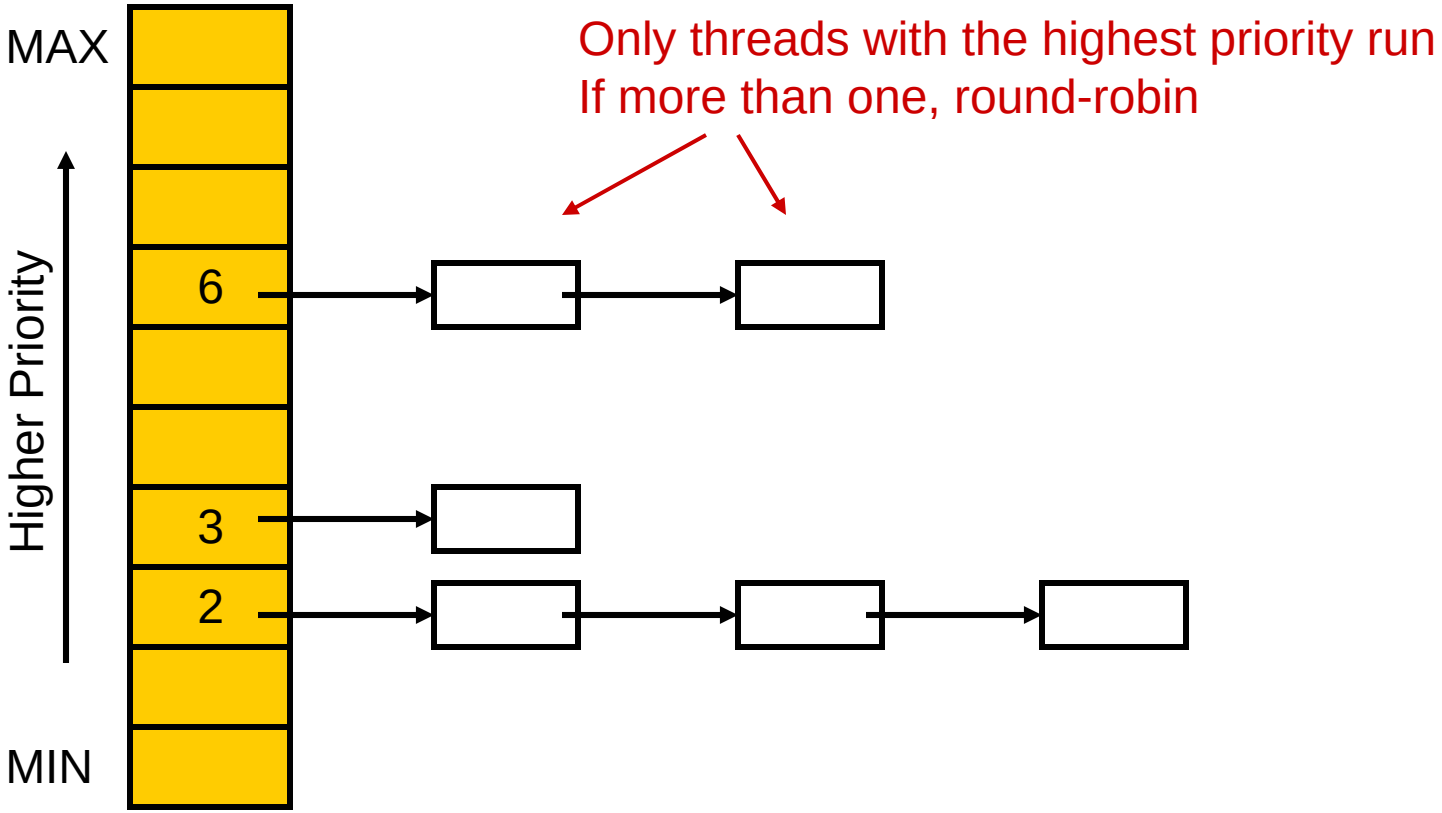
```
/* Suspends execution for approximately TICKS timer ticks. */
void timer_sleep (int64_t ticks){
  int64_t start = timer_ticks ();
  ASSERT (intr_get_level () == INTR_ON);
  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

- Implementation details
  - Remove thread from ready list and record time when it should be put back in ready list - in timer_sleep( )
  - Put it back after sufficient ticks have elapsed - in timer interrupt handler, timer_interrupt( ).

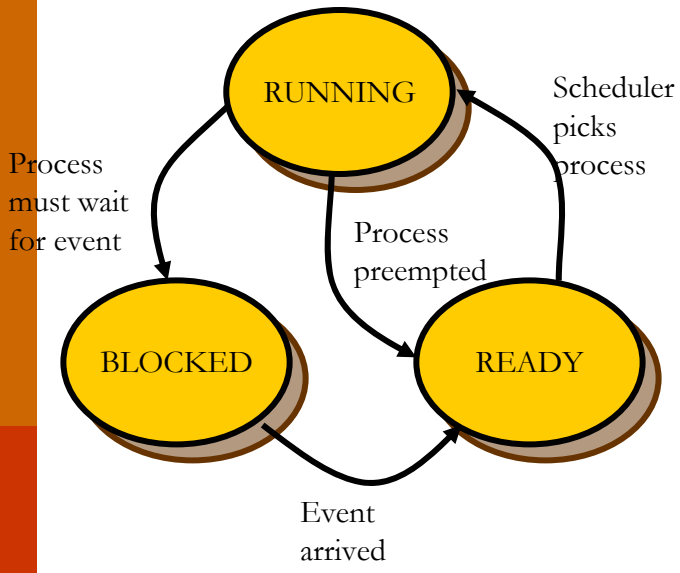
Priority Scheduler

- Ready thread with highest priority gets the processor
- When a thread is added to the ready list that has a higher priority than the currently running thread, immediately yield the processor to the new thread
- When threads are waiting for a lock, semaphore or a condition variable, the highest priority waiting thread should be woken up first
- Implementation details
  - compare priority of the threads inthe ready list with that of the running thread
  - select next thread to run based on priorities
  - compare priorities of waiting threads when releasing locks, semaphores, condition variables (for priority inheritance), e.g., when thread_unblock(t) is called, check if current thread's priority is less than priority of thread t, then the current thread should yield() by calling thread_yield();.

Priority Based Scheduling



Using thread_yield() to implement preemption



- Current thread ("RUNNING") is moved to READY state, added to READY list.
- Then scheduler is invoked. Picks a new READY thread from READY list.
- Case a): there's only 1 READY thread. Thread is rescheduled right away
- Case b): there are other READY thread(s)
  - b.1) another thread has higher priority – it is scheduled
  - b.2) another thread has same priority – it is scheduled provided the previously running thread was inserted in tail of ready list.
- "thread_yield()" is a call you can use whenever you identify a need to preempt current thread.
- Exception: inside an interrupt handler, use "intr_yield_on_return()" instead

Function to compare threads in a list by priority

```
/* Returns true if thread a has lower priority than thread b,
   within a list of threads. */
bool
thread_lower_priority (const struct list_elem *a_,
                       const struct list_elem *b_,
                       void *aux UNUSED)
{
  const struct thread *a = list_entry (a_, struct thread, elem);
  const struct thread *b = list_entry (b_, struct thread, elem);

  return a->priority < b->priority;
}
```

Function to yield to a higher priority thread using list_entry and list_max

```
/* If the ready list contains a thread with a higher priority,
   yields to it. */
void thread_yield_to_higher_priority (void)
{
  enum intr_level old_level = intr_disable ();
  if (!list_empty (&ready_list)) {
      struct thread *cur = thread_current ();
      struct thread *max = list_entry (list_max (&ready_list,
          thread_lower_priority, NULL), struct thread, elem);
      if (max->priority > cur->priority) {
          if (intr_context ())
            intr_yield_on_return ();
          else
            thread_yield ();
      }
  }
  intr_set_level (old_level);
}
```