Project 2: Priority Scheduling

CSL-Pintos

Notice

Submission Deadline

- 10(Mon) June until 23:59 (4 weeks)
- Delay
  - ✓ 10 % reduction for every more than 1 day
  - ✓ Delayed submission will be accepted until 13(Thu) June

Softcopy

- Design (50%) / Testing (50%)
  - ✓ Design: design document / source code
  - ✓ Testing: test case
- Receives 0 point if flagged as a plagiarism with others

Task #2 Priority Scheduling

Main goal

① Modify PintOS default scheduling for **priority scheduling**

- ✓ Sort ‘`ready_list`’ for thread priority scheduling

② Implement the **preemption**

Task #2 Priority Scheduling

Files to modify

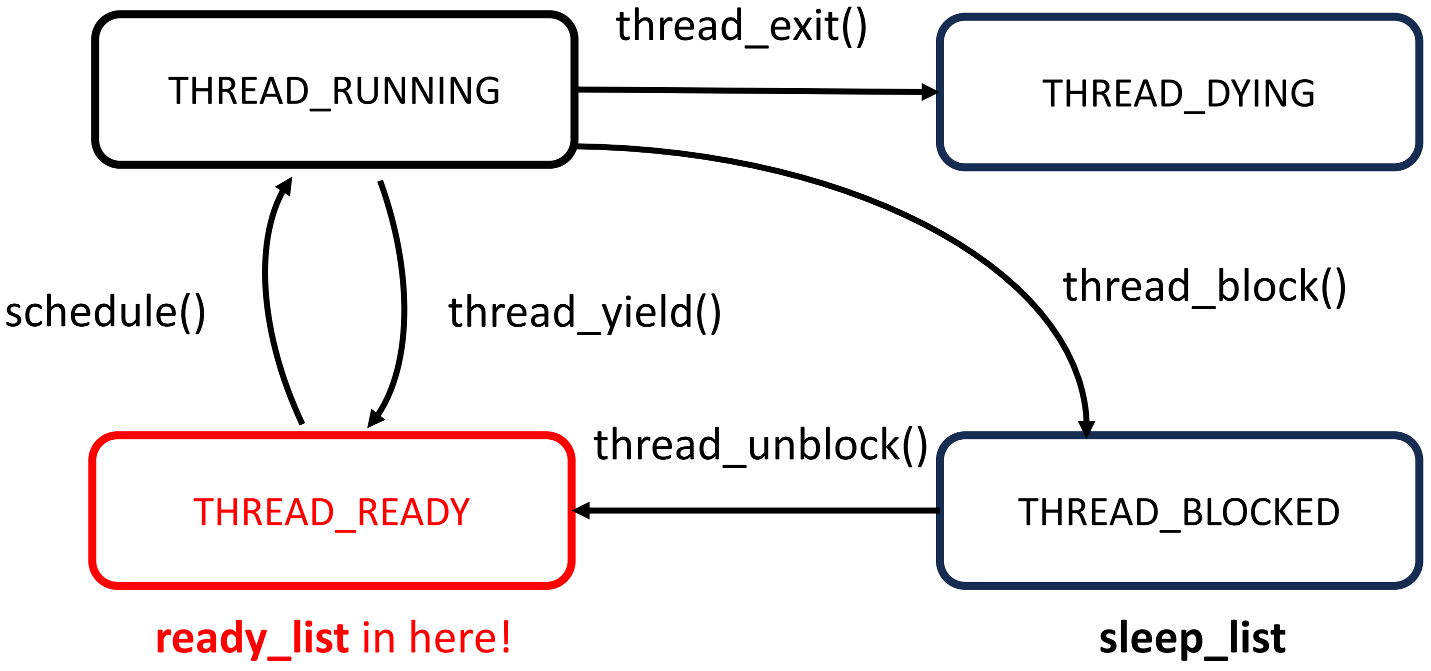
① threads/thread.*

- ✓ Code related to the implementation of main thread behaviors

  -> Thread init, scheduling, etc.

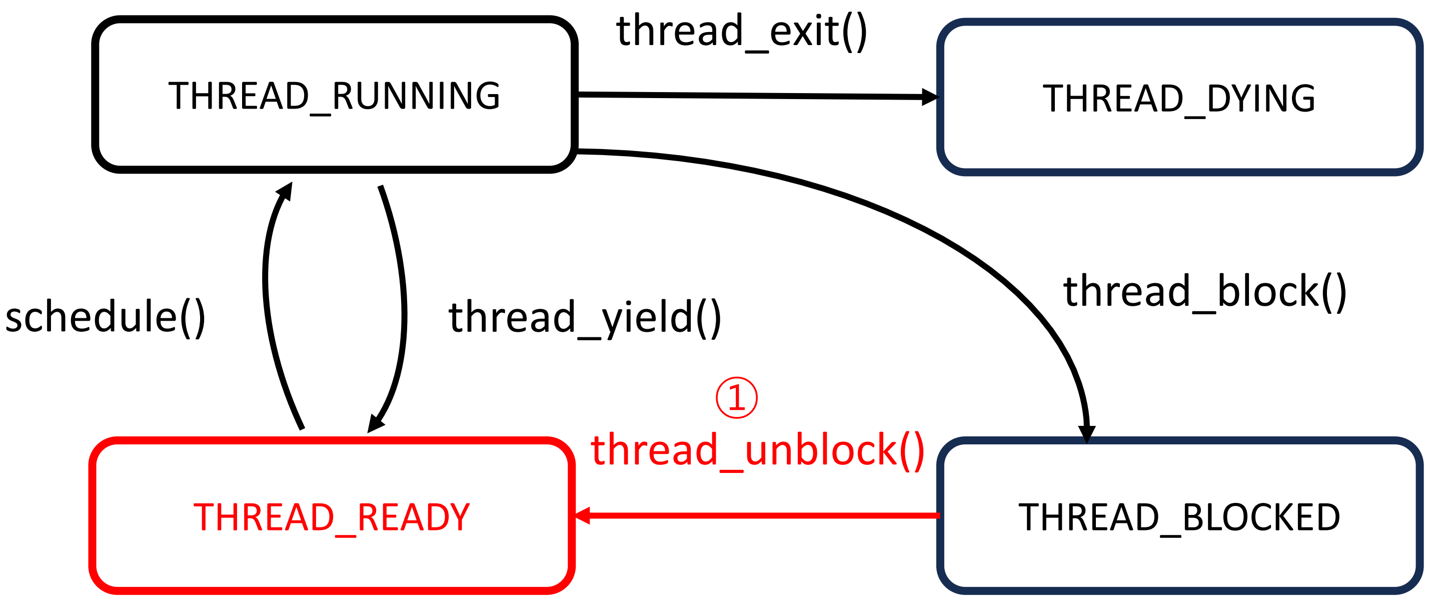
Task #2 Priority Scheduling

- Threads in PintOS have 4 states



Task #2 Priority Scheduling

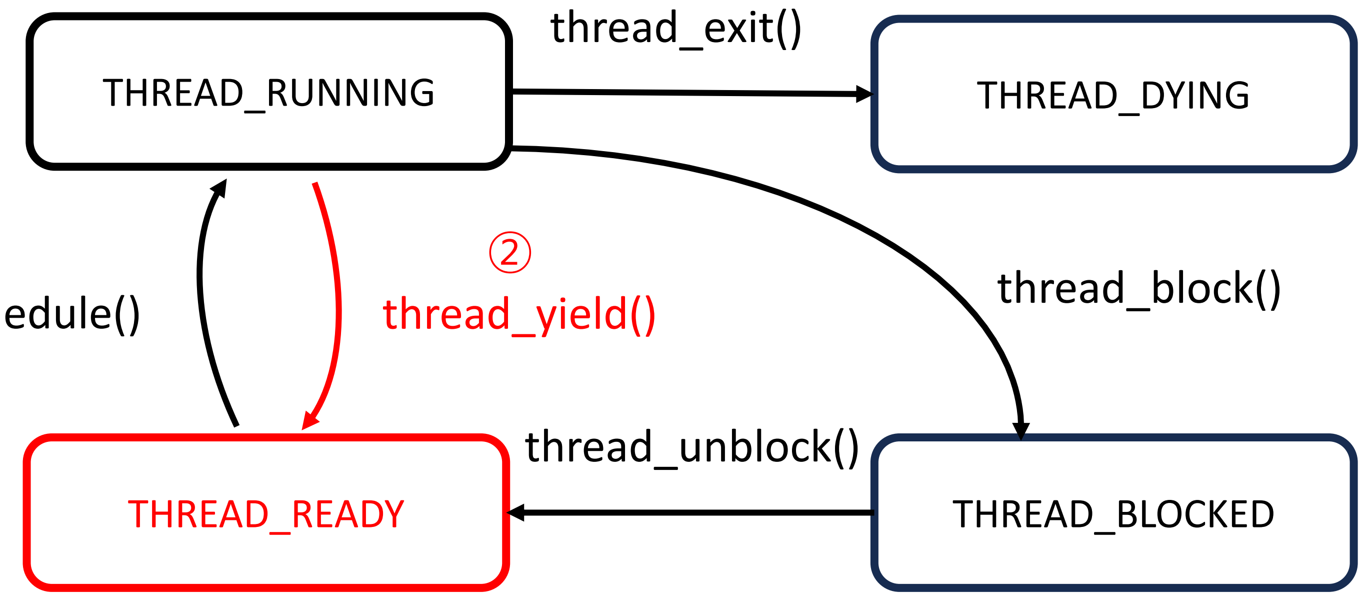
- **BLOCKED → READY** with `thread_unblock()`



```
void thread_unblock (struct thread *t)
{
   enum intr_level old_level;

   ASSERT (is_thread(t));

   old_level = intr_disabled();
   ASSERT (t->status == THREAD_BLOCKED);
   list_push_back (&ready_list, &t->elem);
   t->status = THREAD_READY;
   intr_set_level (old_level);
}
```

Task #2 Priority Scheduling

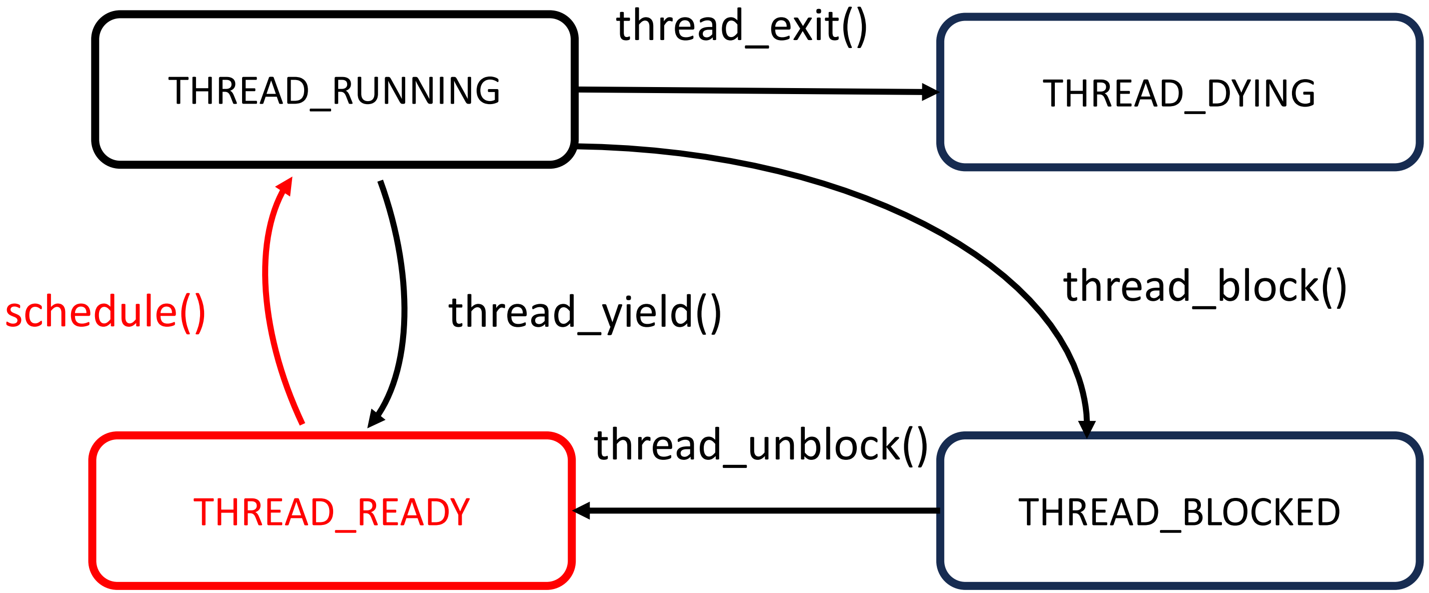
- **RUNNING → READY** with `thread_yield()`



```
void thread_yield (void)
{
   struct thread *cur = thread_current ();
   enum intr_level old_level;

   ASSERT (!intr_context ());

   old_level = intr_disable ();
   if (cur != idle_thread)
      list_push_back (&ready_list, &cur->elem);
   cur->status = THREAD_READY;
   schedule ();
   intr_set_level (old_level);
}
```

Task #2 Priority Scheduling

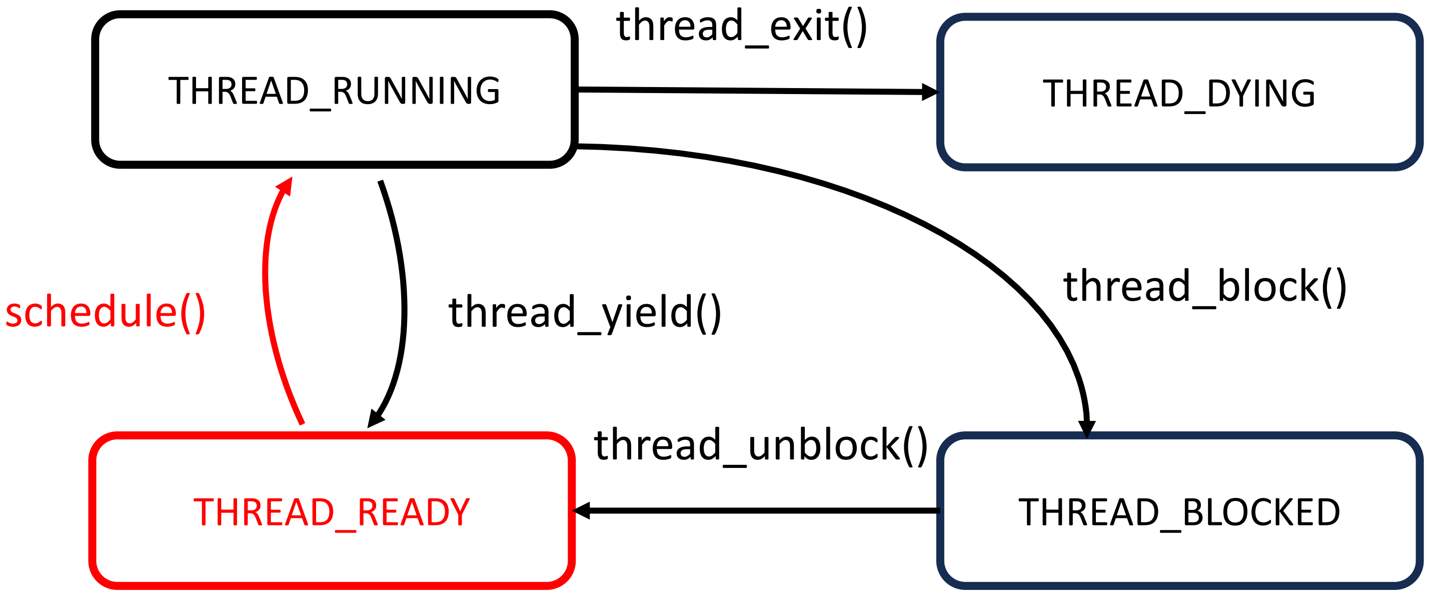
- **READY → RUNNING** with `schedule()`



```
static void schedule (void)
{
  struct thread *cur = running_thread ();
  struct thread *next = next_thread_to_run ();
  struct thread *prev = NULL;

  ASSERT (intr_get_level () == INTR_OFF);
  ASSERT (cur->status != THREAD_RUNNING);
  ASSERT (is_thread (next));

  if (cur != next)
    prev = switch_threads (cur, next);
  thread_schedule_tail (prev);
}
```

Task #2 Priority Scheduling

- **READY → RUNNING** with `schedule()`

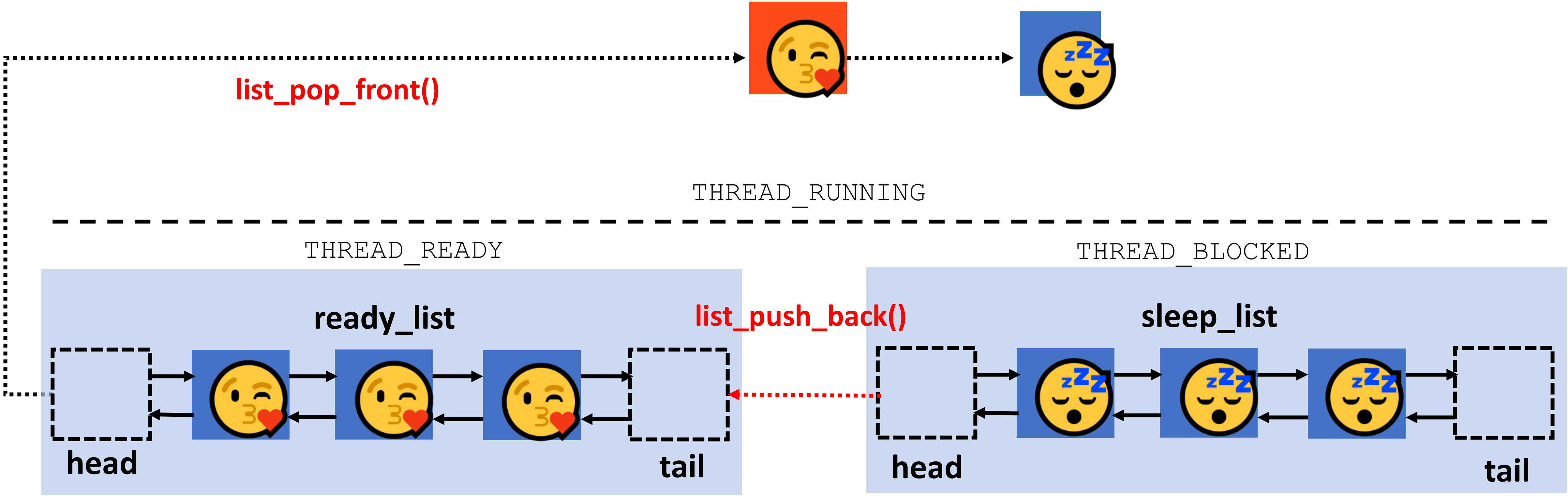


```
static void schedule (void)
{
  struct thread *cur = running_thread ();
  struct thread *next = next_thread_to_run ();
```

```
static struct thread *
next_thread_to_run (void)
{
   if (list_empty (&ready_list))
      return idle_thread

    else
      return list_entry(list_pop_front (&ready
      _list), struct thread, elem);
}
```

Task #2 Priority Scheduling

- PintOS uses FIFO scheduling as default



Task #2 Priority Scheduling

Limitation of FIFO scheduling

- Initial priority is set by PintOS when the thread is created via `thread_create()`

- Priority ranges is from 0 to 63, and higher numbers have higher priority
  - ✓ `PRI_MIN(=0)`, `PRI_DEFAULT(=31)`, `PRI_MAX(=63)`

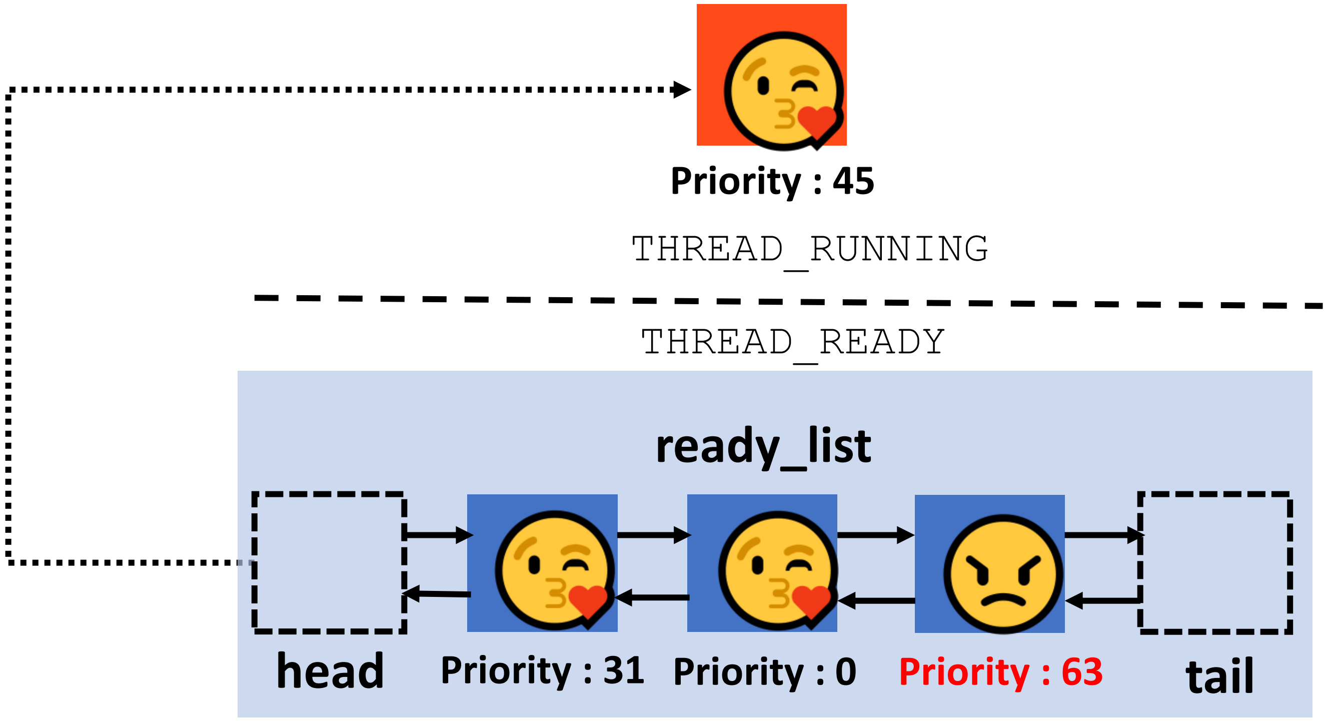
- `int thread_get_priority (void)` in `threads/thread.c`
  - ✓ Enables to return the current thread's priority

- `void thread_set_priority (int new_priority)` in `threads/thread.c`
  - ✓ Allows to change thread's priority to the value `new_priority`

Task #2 Priority Scheduling

Limitation of FIFO scheduling

- PintOS does not consider priority while scheduling

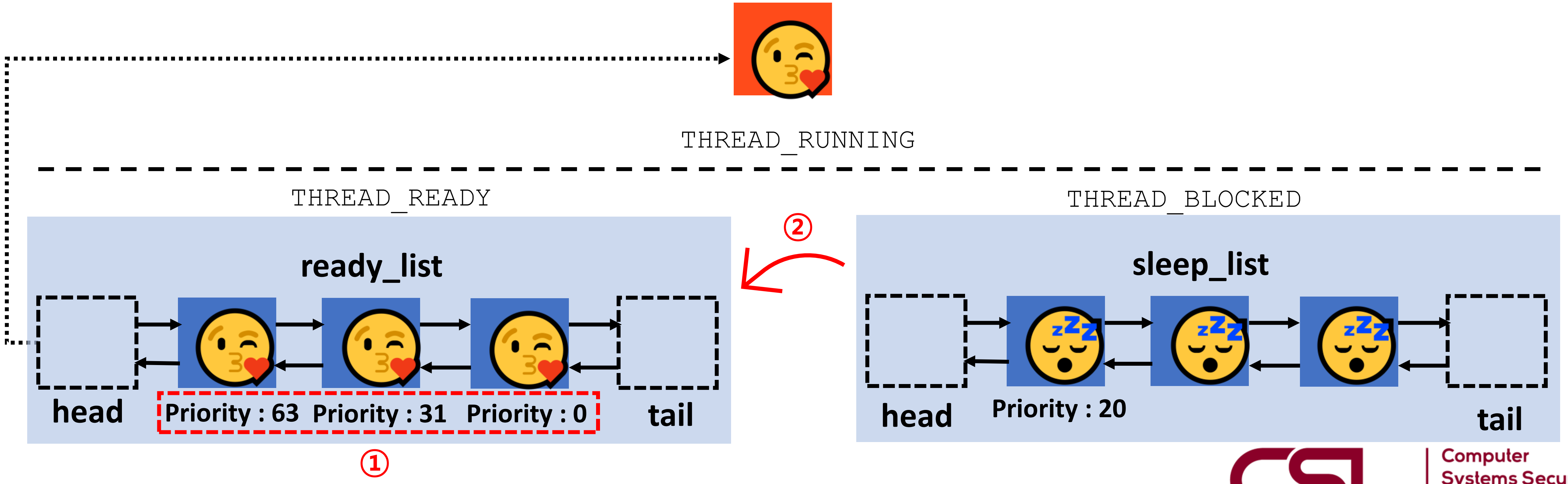


Task #2 Priority Scheduling

How can we resolve this problem?

① Sort the **`‘ready_list’`** by priority

② When pushing a thread, push it according to its priority



Task #2 Priority Scheduling

Implementation details

- [Hint] You can use `list_insert_ordered()` in `/lib/kernel/list.c`

```
void list_inserted_ordered (struct list *list, struct_list_elem *elem, list_less_func *less, void *aux)
{
   struct list_elem *e;

   ASSERT (list != NULL);
   ASSERT (elem != NULL);
   ASSERT (less != NULL);

   for ( e = list_begin(list); e != list_end(list); e = list_next(e))
      if (less(elem, e, aux))
         break;
     return list_insert(e, elem);
}
```

```
void list_insert (struct_list_elem *before, struct_list_elem *elem)
{
   ASSERT (is_interior(before) || is_tail(before));
   ASSERT (elem != NULL);

   elem->prev = before->prev;
   elem->next = before;
   before->prev-> next = elem;
   before->prev = elem;
}
```

Task #2 Priority Scheduling

Implementation details

- Implement “`Your_Own_Less_Function()`”

```
void list_inserted_ordered (struct list *list, struct_list_elem
*elem, list_less_func *less, void *aux)
{
  struct list_elem *e;

  ASSERT (list != NULL);
  ASSERT (elem != NULL);
  ASSERT (less != NULL);

  for ( e = list_begin(list); e != list_end(list); e = list_next(e))
     if (less(elem, e, aux))
        break;
    return list_insert(e, elem);
}
```

```
Bool Your_Own_Less_Function (struct list *l, struct_
list_elem *s, void *aux UNUSED)
{
  /*
  - Performing appropriate comparison operations
  */
}
```

If you are creating a new function, you must define it in the header file.

Task #2 Priority Scheduling

Implementation details

- Change ’list_pushback()’ to ‘list_insert_ordered()’

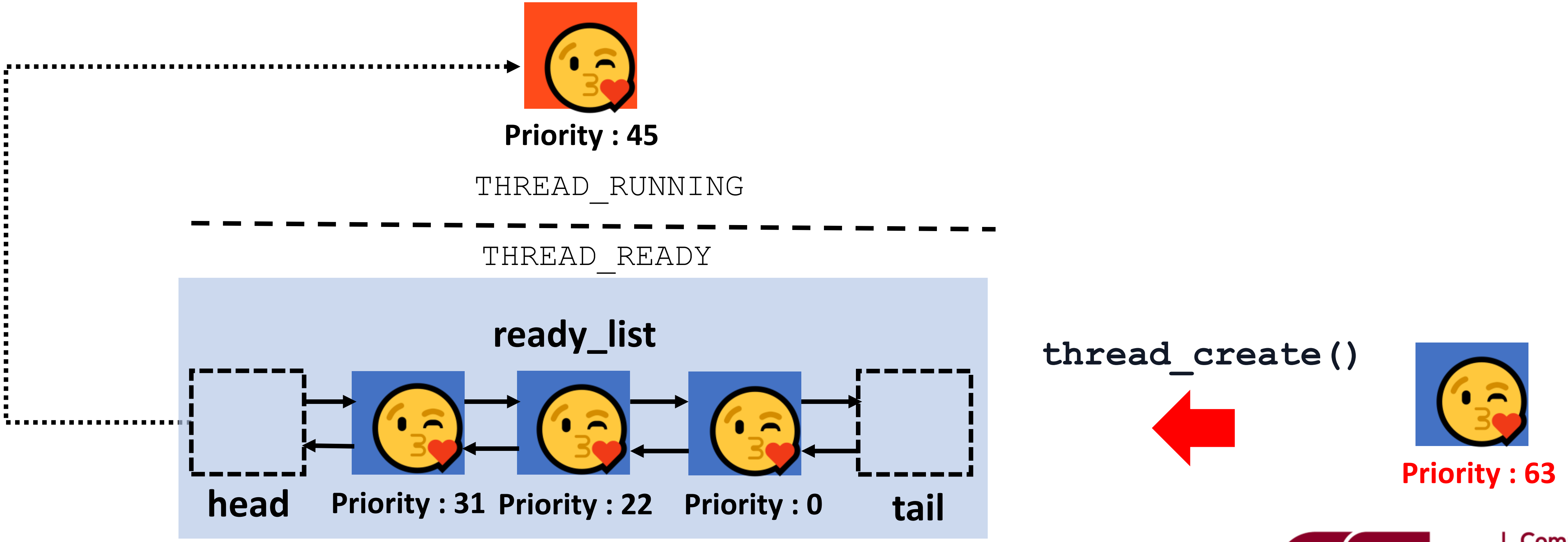
```
void thread_yield (void)
{
   …

   if (cur != idle_thread)
      list_push_back (&ready_list, &cur->elem);

   …
}
```

```
void thread_unblock (struct thread *t)
{
   …

   list_push_back (&ready_list, &t->elem);

   …
}
```

Task #2 Priority Scheduling

Things to consider

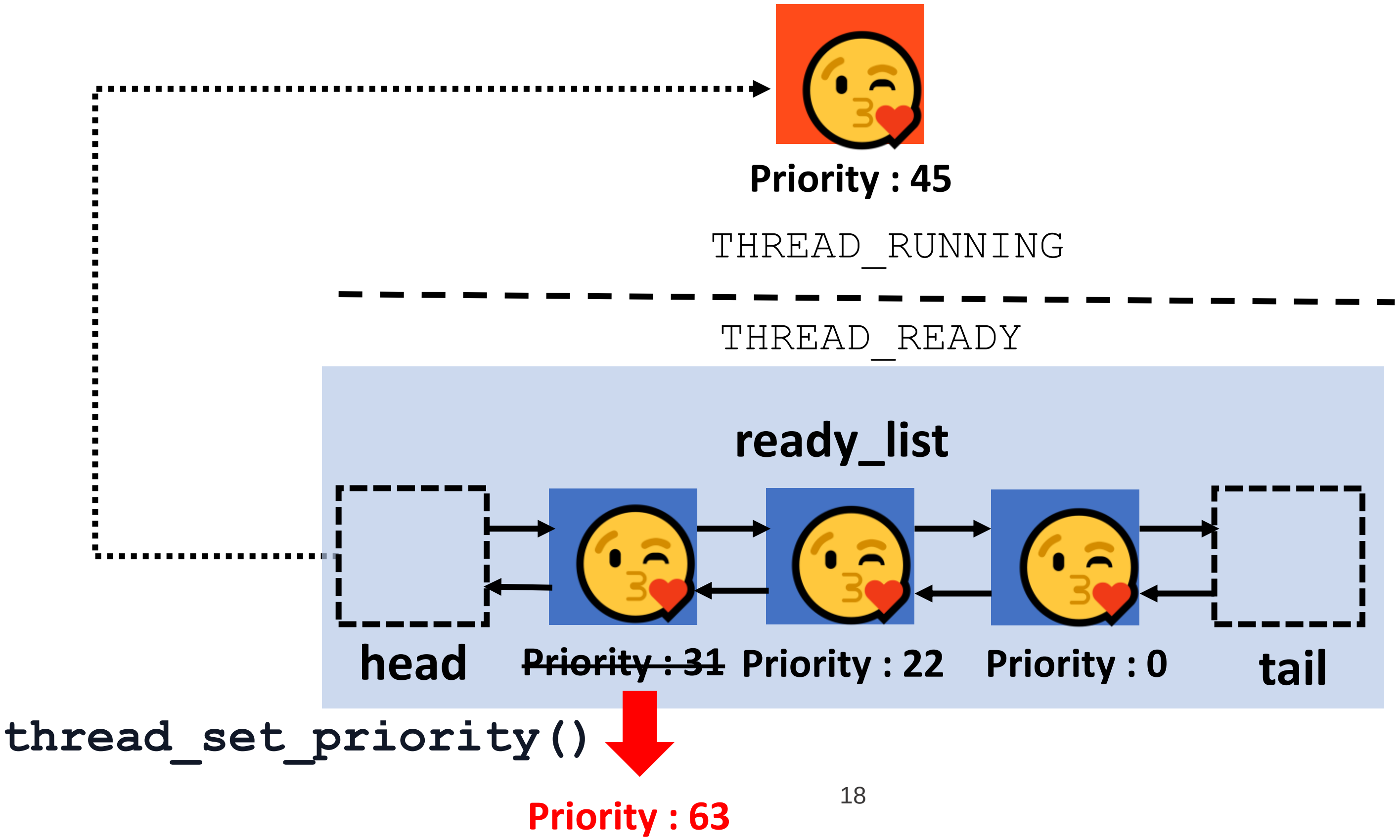
① When the priority of a new thread created via **'thread_create()'** is higher than the running thread



Task #2 Priority Scheduling

Things to consider

② When a thread's changed priority is higher than the running thread via **`thread_set_priority()`**



Task #2 Priority Scheduling

Implementation details

- Implement ‘`Your_Own_Preemption()`’ in `threads/thread.c`

```
void Your_Own_Preemption (void)
{
   /*
  - Compare the priorities of the newly inserted thread and currently running thread
  - Yield the CPU if the newly inserted thread has higher priority than running thread
  */
}
```

If you are creating a new function, you must define it in the header file.

Task #2 Priority Scheduling

Implementation details

- Add 'Your_Own_Preemption()' to 'thread_create()' and 'thread_set_priority()'

```
tid_t thread_create (const char *name, int priority, thread_func *function, void *aux)
{
    …

    thread_unblock (t);
    Your_Own_Preemption();

    return tid;
}
```

```
void thread_set_priority (int new_priority)
{
    thread_current ()->priority = new_priority;
    Your_Own_Preemption();
}
```

Task #2 Priority Scheduling

Compile

- $ cd pintos/src/thread
- $ make clean
- $ make
- $ cd build

Testing

- $ **make check**

Thank you

:-)