Project 1: Alarm Clock

CSL-Pintos

Notice

Submission Deadline

- Mon May 13 until 23:59
- Delay
  - ✓ 10 % reduction for every more than 1 day
  - ✓ Delayed submission will be accepted until Thu May 16

Softcopy

- Design (50%) / Testing (50%)
  - ✓ Design: design document / source code
  - ✓ Testing: test case
- **Student must submit a "report" and "pintos.tar.gz"**
- **Receives 0 point if flagged as a plagiarism with others**

Get familiar with Pintos!

The structure of Pintos

① `src/utils/`

- ✓ It contains a number of help functions and utilities related to the Pintos kernel

② `src/threads/`

- ✓ It configures the behavior of kernel threads

③ `src/devices/`

- ✓ It contains hardware device drivers and related code for the Pintos operating system

④ `src/lib/`

- ✓ It contains various libraries and utility codes providing common functions and features for both the kernel and application programs

⑤ `src/tests/`

- ✓ It contains test code and test suites used to verify various components and functionalities

Task #1 Alarm Clock

Main goal

- Implement **a thread scheduling algorithm without busy-waiting**
- Managing sleeping threads separately in a list other than the ready list
- When the sleep time is up, wake the thread and put it in the ready list

```
void timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();

  ASSERT (intr_get_level () == INTR_ON);
  while (timer_elapsed (start) < ticks)
  thread_yield ();
}
```

Task #1 Alarm Clock

Files to modify

① `threads/thread.*`

- ✓ Code related to thread behavior and scheduling

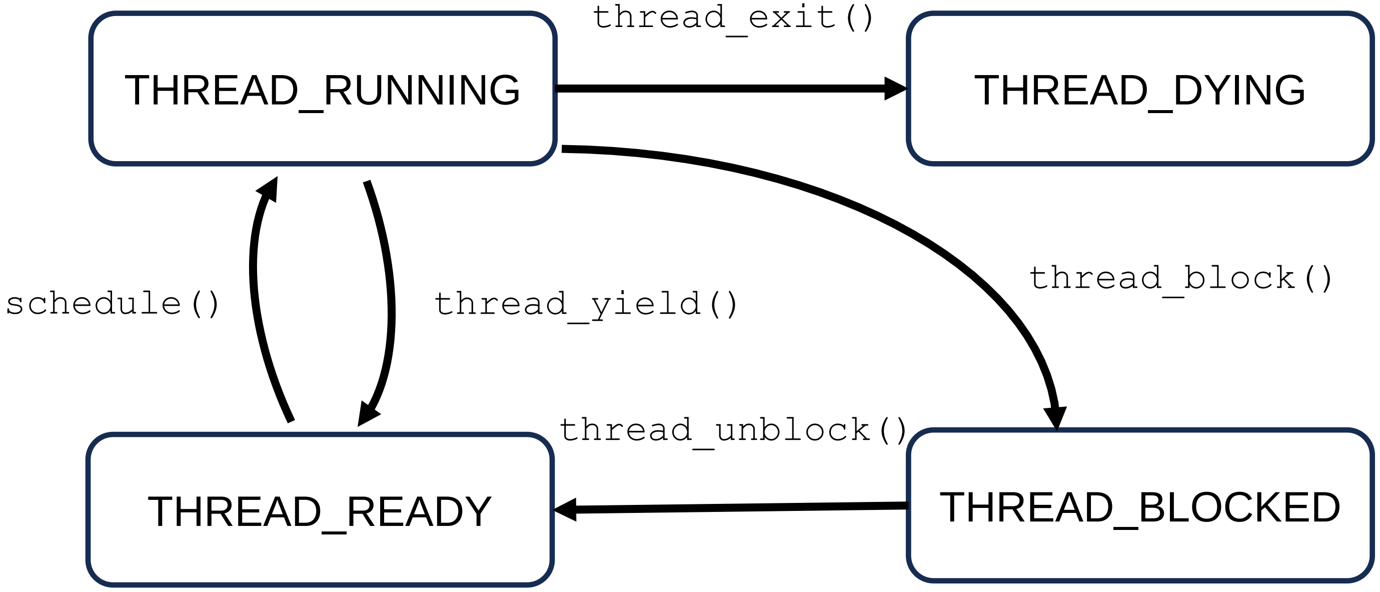
② `devices/timer.*`

- ✓ Code related to device time, such as yielding CPU occupancy to a thread

Task #1 Alarm Clock

Threads in Pintos have 4 states

- Running, Ready, Blocked, Dying



THREAD_RUNNING

thread_exit()

THREAD_DYING

schedule()

thread_yield()

thread_block()

thread_unblock()

THREAD_READY

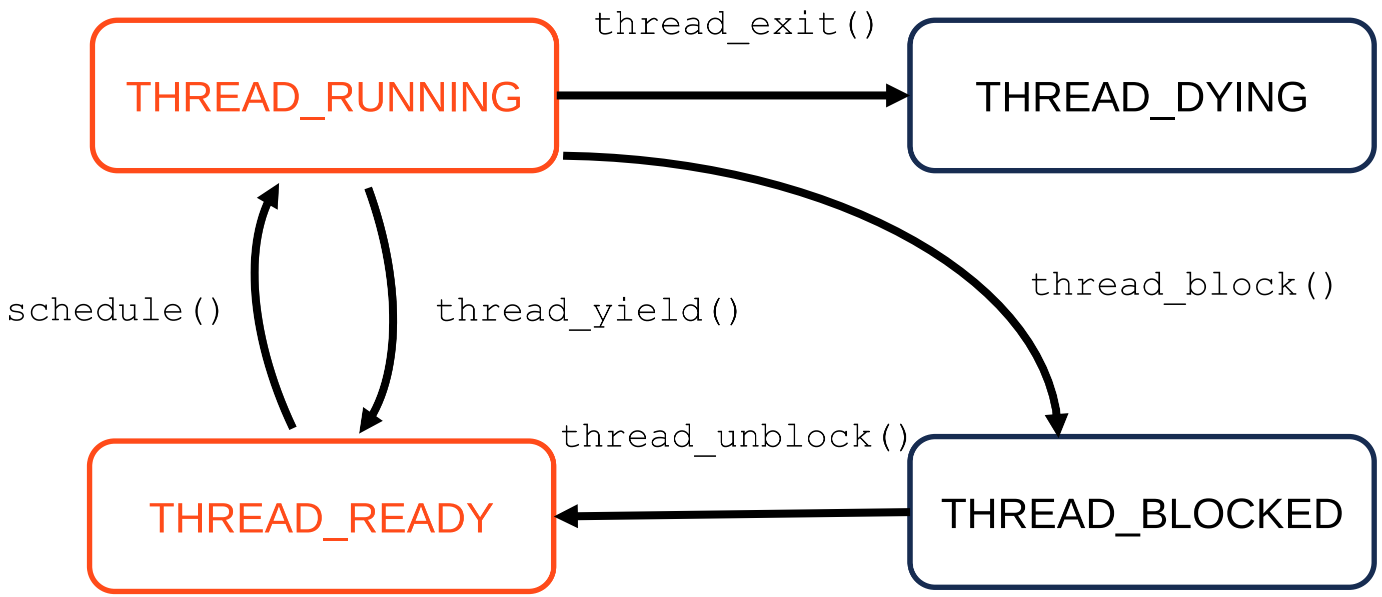
THREAD_BLOCKED

```
enum thread_status
{
   THREAD_RUNNING, // Running thread
   THREAD_READY, // Not running but ready to run
   THREAD_BLOCKED, // Waiting for an event to trigger
   THREAD_DYING // About to be destroyed
};
```

Task #1 Alarm Clock

With a busy-waiting, the thread uses just 2 states

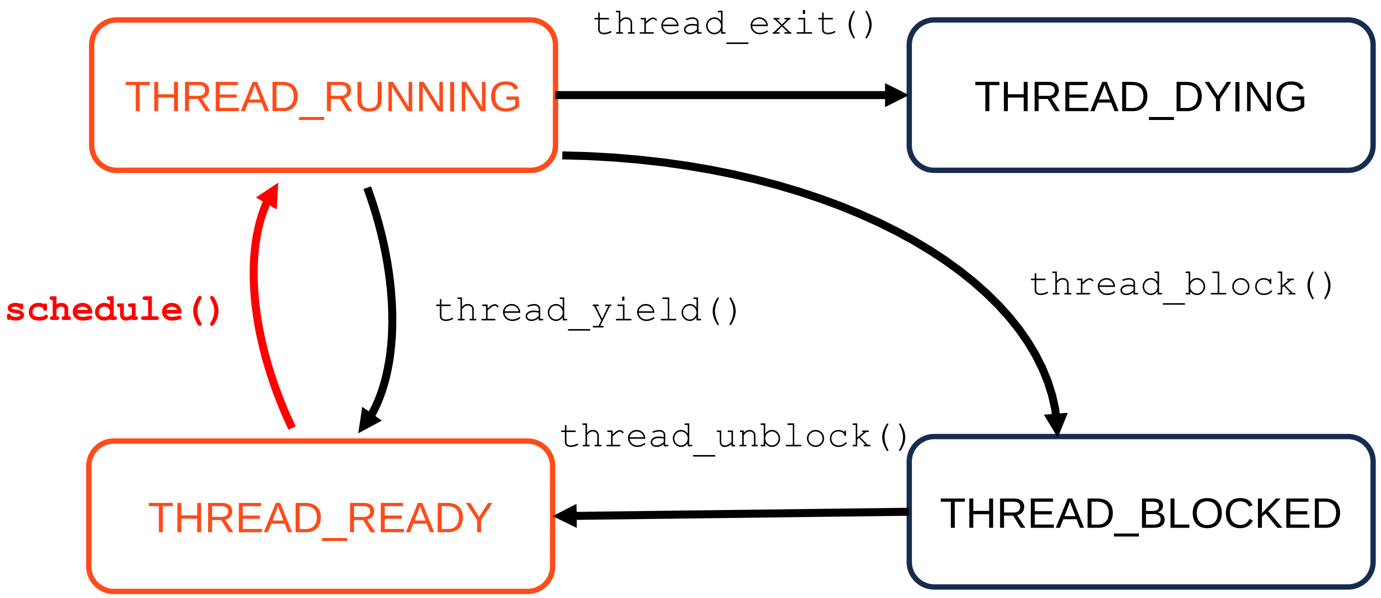
- Running, Ready, ~~Blocked, Dying~~

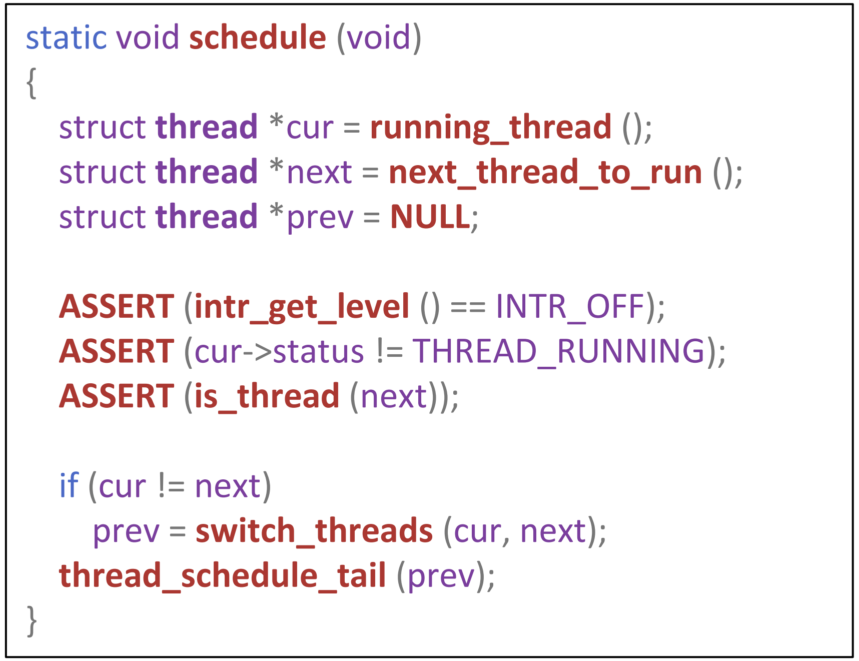


Task #1 Alarm Clock

READY -> RUNNING with schedule()

- Interrupts are disabled and the currently running thread is switched to the next thread



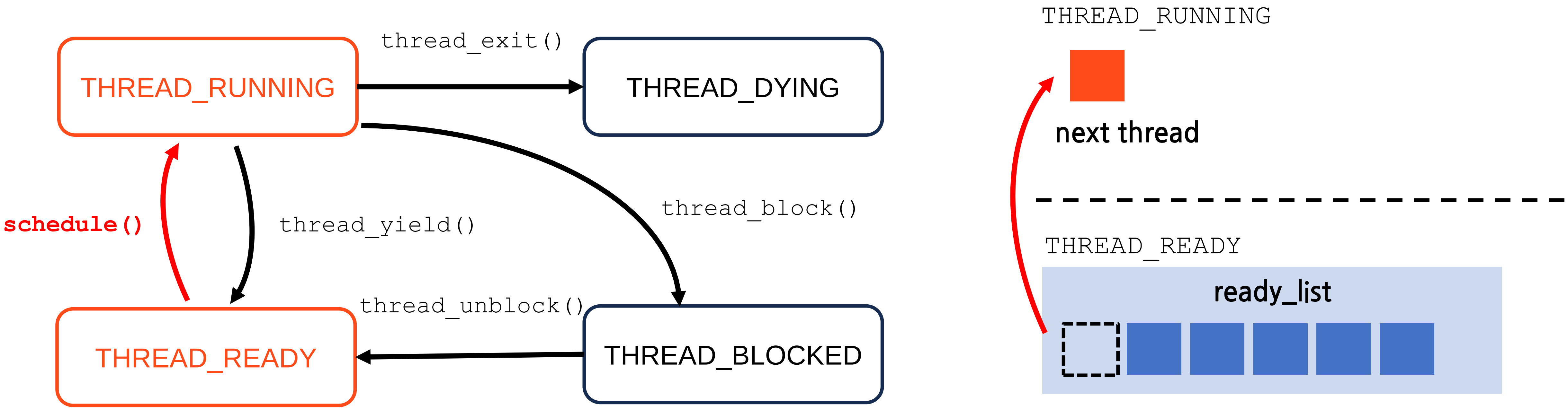


```
static void schedule (void)
{
   struct thread *cur = running_thread ();
   struct thread *next = next_thread_to_run ();
   struct thread *prev = NULL;

   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 #1 Alarm Clock

READY -> RUNNING with schedule()

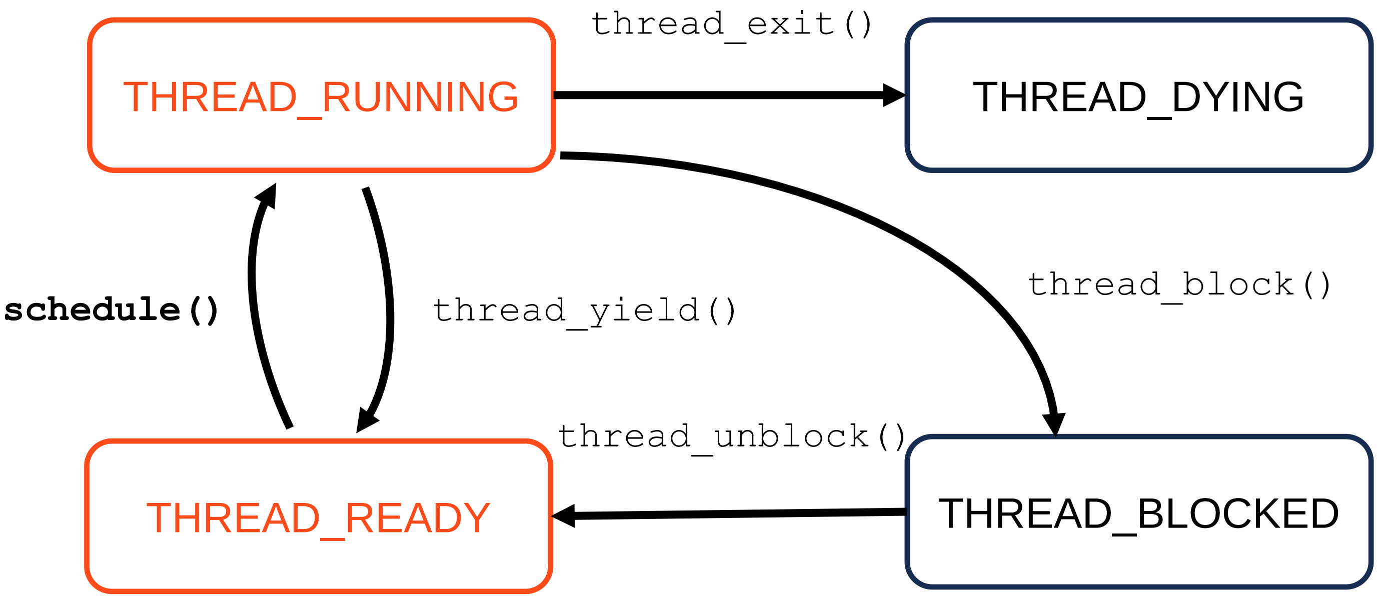
- Interrupts are disabled and the currently running thread is switched to the next thread



Task #1 Alarm Clock

Check the ticks of the thread

- Check the time to see if it's time for the thread to wake up or not



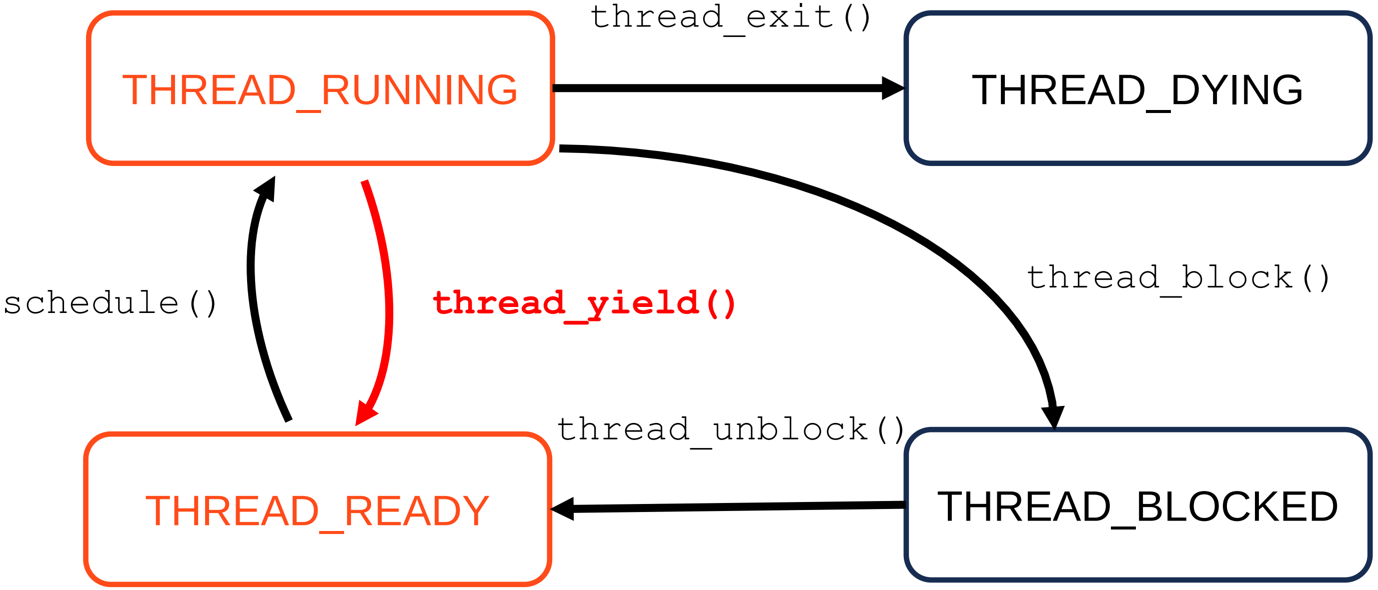
THREAD_RUNNING
check time(ticks)!
next thread

```
void timer_sleep (int64_t ticks)
{
   int64_t start = timer_ticks ();

   ASSERT (intr_get_level () == INTR_ON);
   while (timer_elapsed (start) < ticks)
     thread_yield ();
}
```

Task #1 Alarm Clock

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

- Interrupts are disabled and the status of the currently running thread changes to THREAD_READY
- Currently running threads are pushed to the ready list.

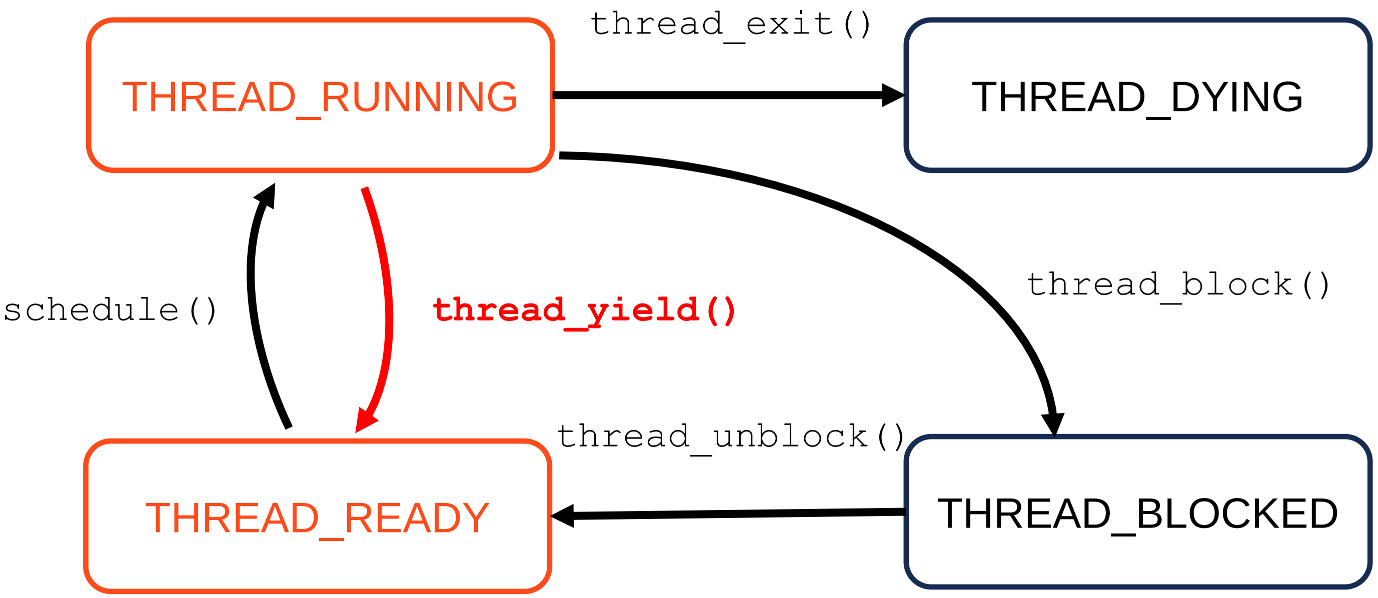


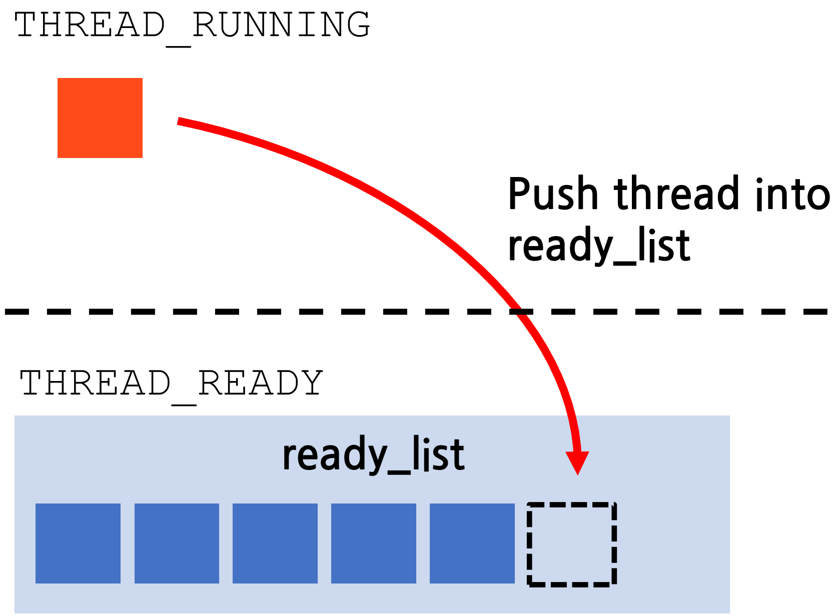
```
void thread_yield (void)
{
   struct thread *cur = thread_current ();
   enum intr_level old_level;
   ASSERT (!intr_context ());

   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 #1 Alarm Clock

RUNNING -> READY with thread_yield()

- Interrupts are disabled and the status of the currently running thread changes to THREAD_READY
- Currently running threads are pushed to the ready list.

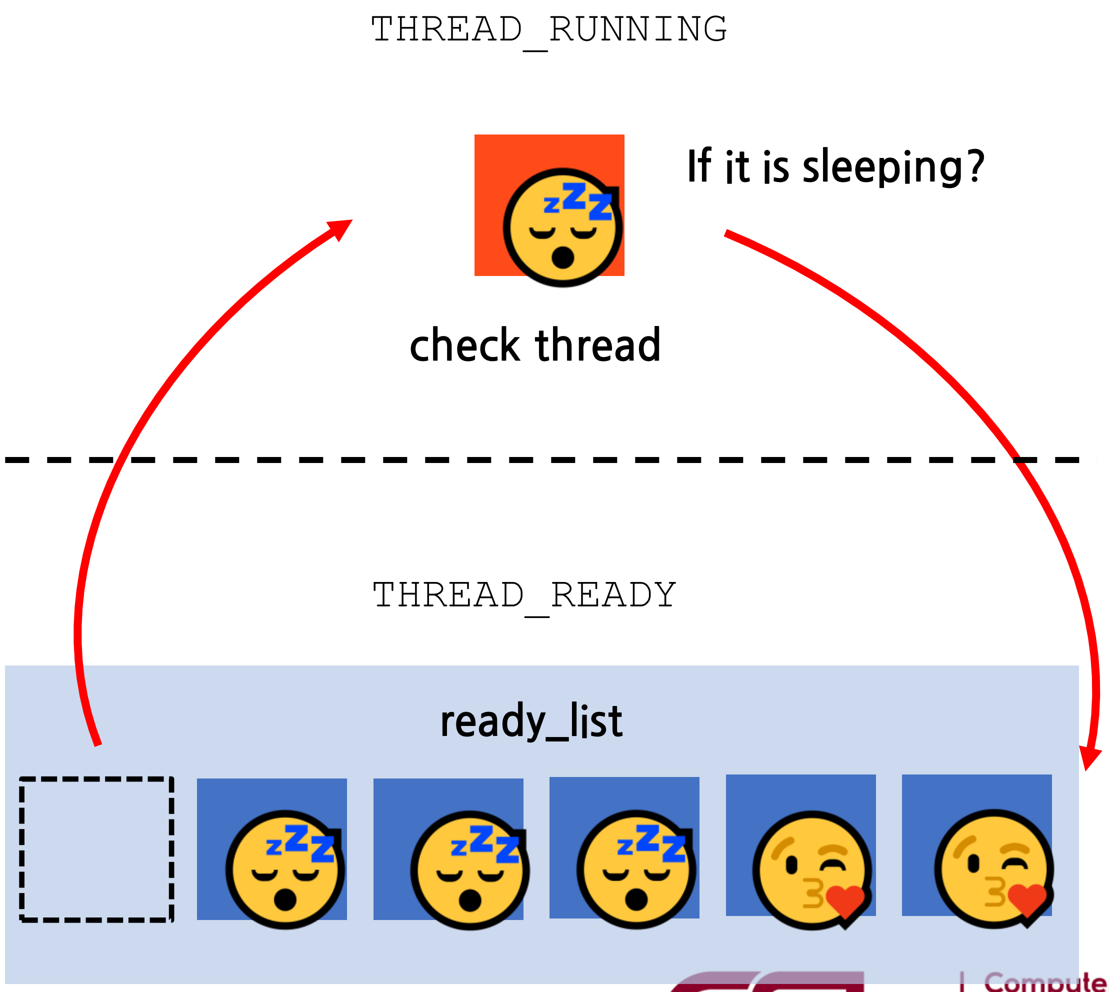




Task #1 Alarm Clock

The limitation of a busy-waiting

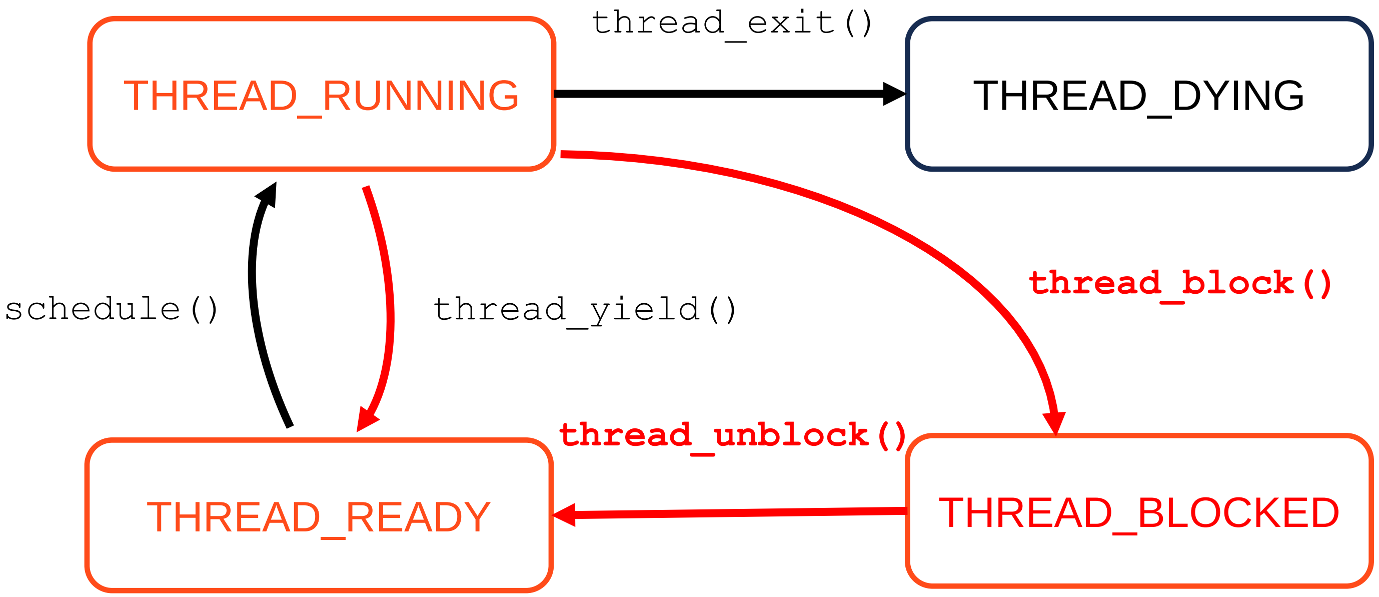
- Sleeping threads occupy the CPU
- It wastes time



Task #1 Alarm Clock

How can we resolve this problem?

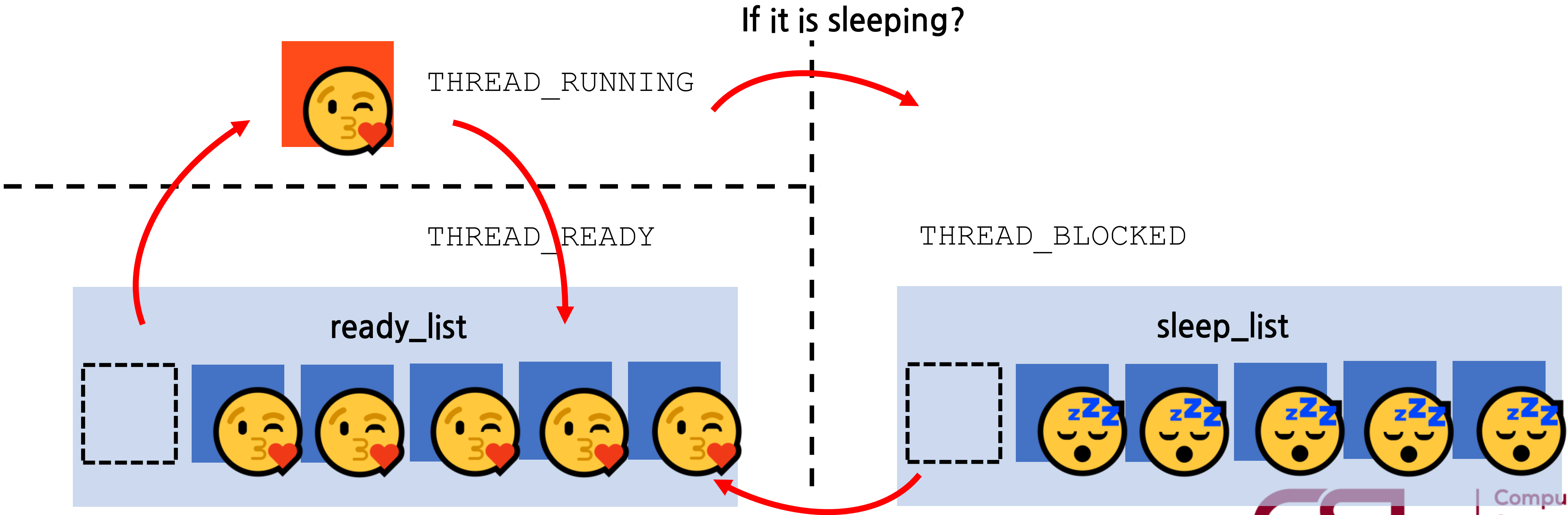
- Use “blocked” state to handle sleeping threads!
- Push threads in the sleep_list while sleeping, and move them to the ready_list when wake up



Task #1 Alarm Clock

How can we resolve this problem?

- Use “blocked” state to handle sleeping threads!
- Push threads in the sleep_list while sleeping, and move them to the ready_list when wake up



Task #1 Alarm Clock

Implementation details

- Add timer field in thread structure

```
/*  /src/threads/threads.h  */
struct thread
{
   int64_t waketime; // time to wake up

   …
}
```

- Initialize sleep list in thread_init

```
/*  /src/threads/threads.c  */
static struct list sleep_list; // define sleep_list

void thread_init (void)
{
   …
   list_init (&sleep_list); // initialize sleep_list
   …
}
```

Task #1 Alarm Clock

Implementation details

- Add a functionality to make threads sleep in timer_sleep

```
/*  /src/devices/timer.c  */
void timer_sleep (int64_t ticks)
{
   int64_t start = timer_ticks ();

         /*
           - Call thread_sleep
         */

}
```

```
/*  /src/threads/threads.c  */
void thread_sleep (int64_t ticks)
{
     /*
       - Disable interrupts between switching thread
       - Store a value for the time the thread will wake up
       - Add the current thread to the sleep list
       - Change the thread status to THREAD_BLOCKED
       - Enable interrupts
     */

}
```

Also, if you are creating a new function, you must define it in the header file.
Ex) “thread/thread.h”

Task #1 Alarm Clock

Implementation details

- Add a functionality to wake up the threads in timer_interrupt

```
/*  /src/devices/timer.c  */
void timer_interrupt (struct intr_frame *args UNUSED)
{
   ticks++;
   thread_tick ();

        /*
          - call thread_interrupt
        */

}
```

```
/*  /src/threads/threads.c  */
void thread_interrupt (int64_t ticks)
{
    /*
      - Iterate the sleep list to determine
        the time to wake up
      - Remove the thread from the sleep list
      - Add it to the ready list
    */
}
```

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

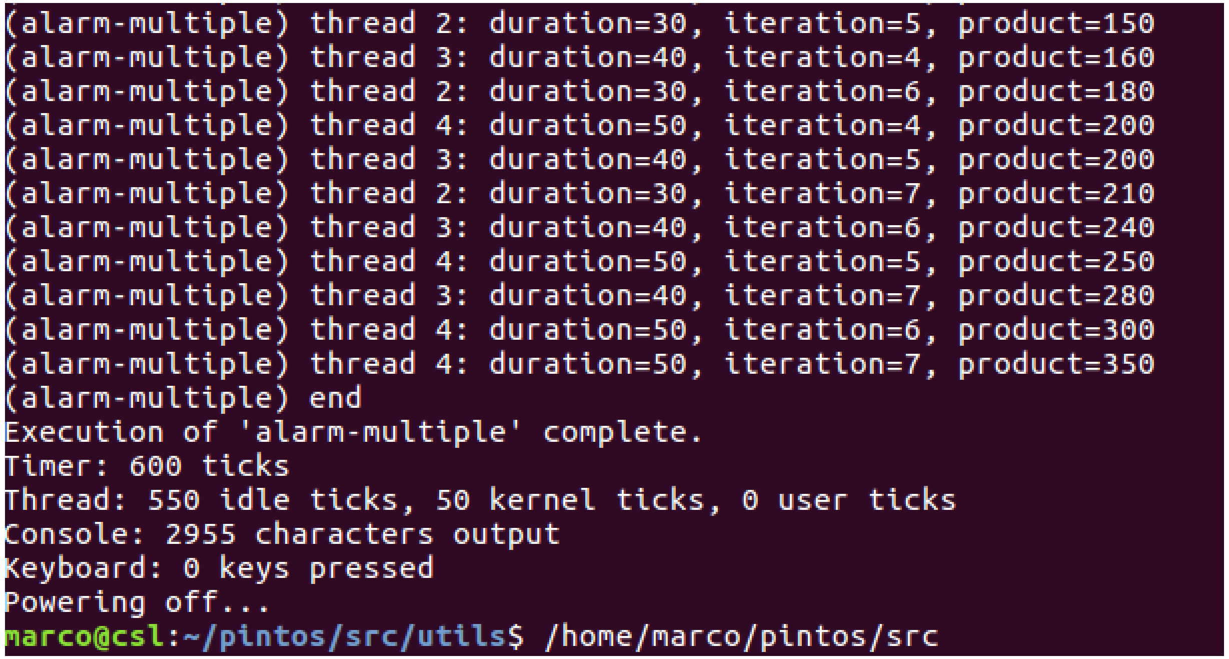
Task #1 Alarm Clock

Compile

- $ cd pintos/src/thread
- $ make

Testing

- $ pintos -q run alarm-multiple



```
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=180
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
Timer: 600 ticks
Thread: 550 idle ticks, 50 kernel ticks, 0 user ticks
Console: 2955 characters output
Keyboard: 0 keys pressed
Powering off...
marco@csl:~/pintos/src/utils$ /home/marco/pintos/src
```

Students must include screenshot with your username[student id]

Thank you

:-)