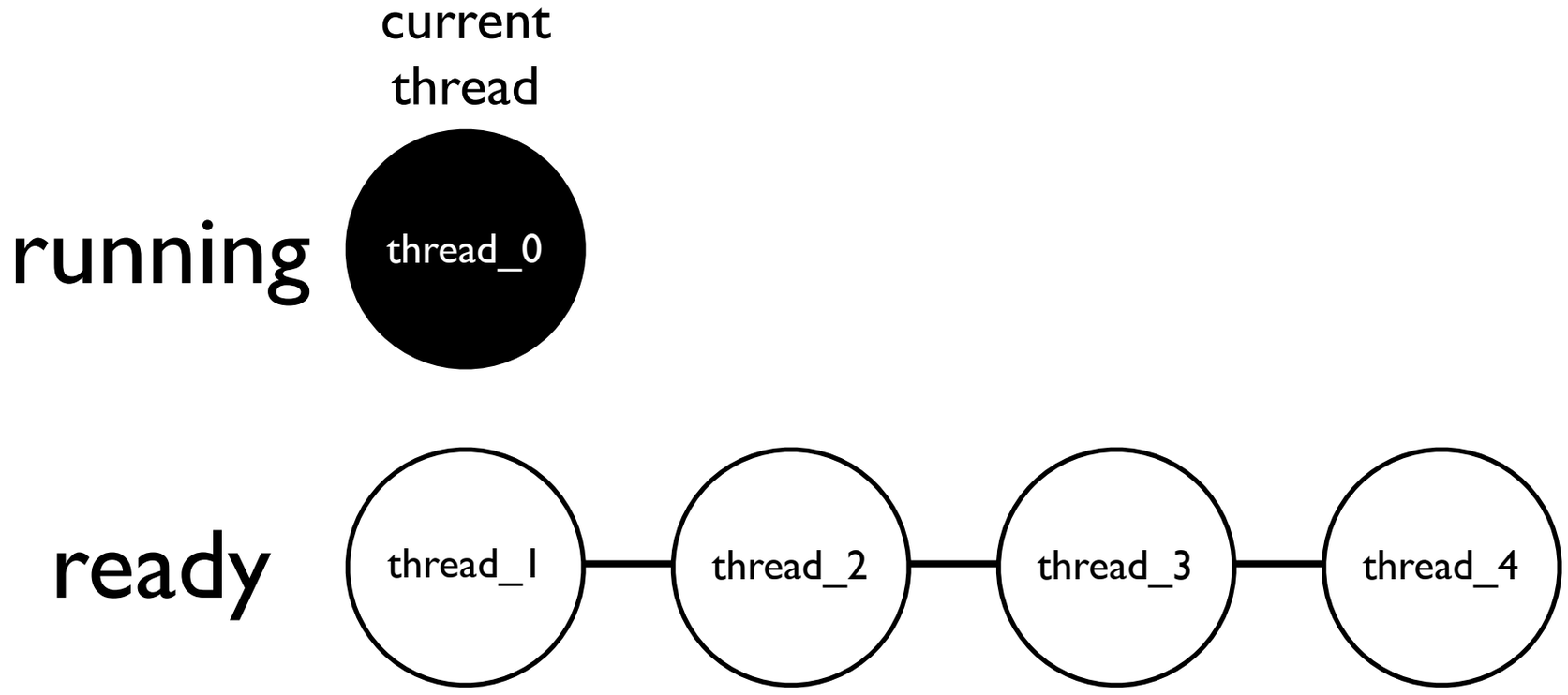
Priority scheduling
&
Advanced scheduling

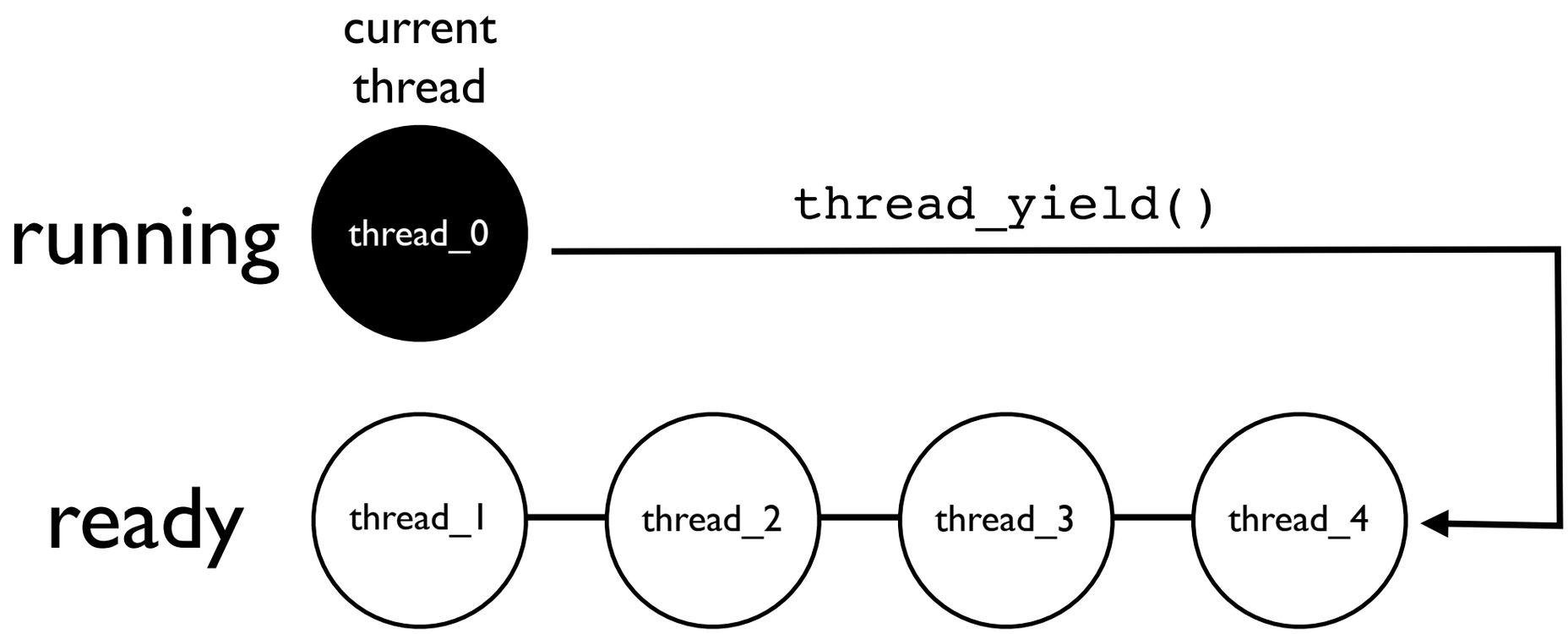
Recap

- So far, we've learnt how to:
  - run and debug pintos tests
  - add our own tests
  - use `lib/kernel/list.h`
  - implement `sleep()` with no busy-wait

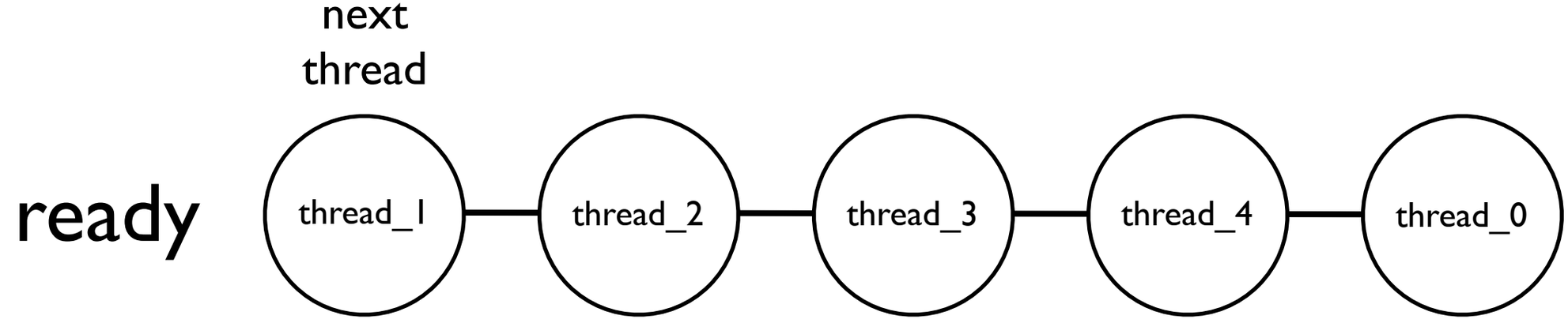
Round-robin schedule



Round-robin schedule

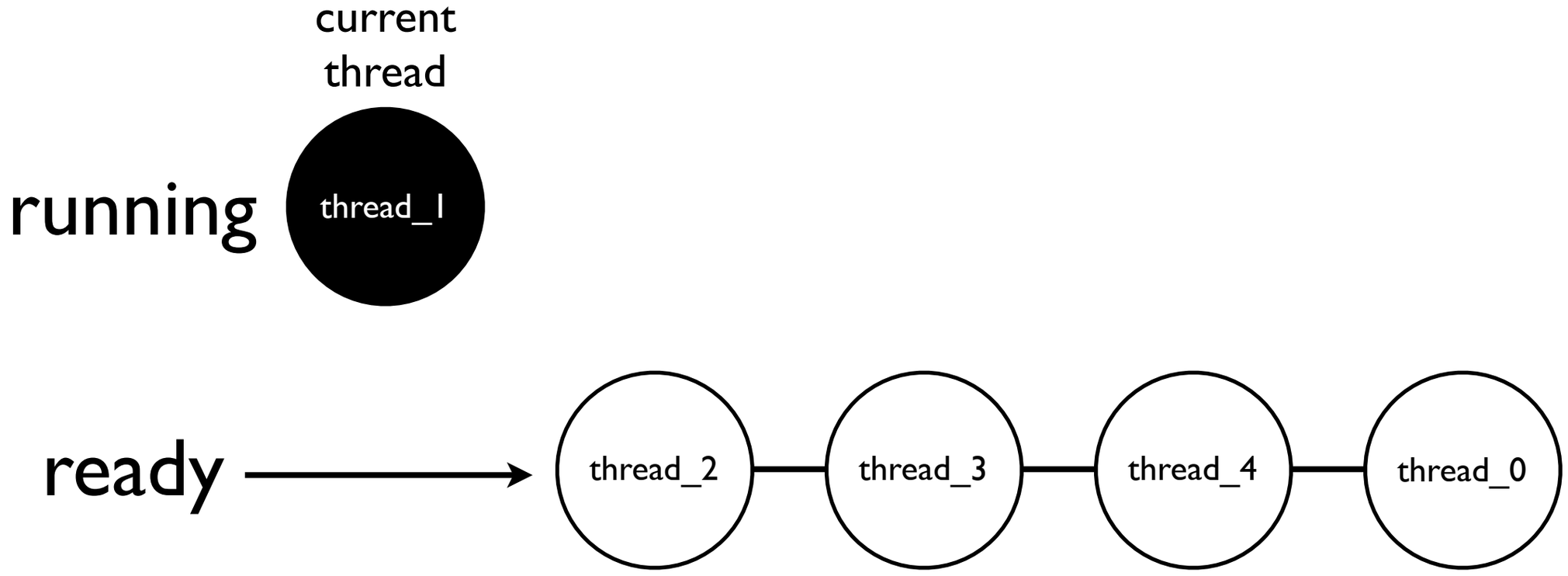


Round-robin schedule




```
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);
}
```

Round-robin schedule




```
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);
}
```

Priority scheduling

Priority scheduling

- Some threads/processes require faster response
- This may be achieved with a higher priority
- The next thread to run is that with the highest priority
- If multiple threads have the same highest priority, they alternate in round-robin

Priority scheduling

- A "priority" is an int ranging from 0 to 63
  `(PRI_MIN = 0; PRI_DEFAULT = 31; PRI_MAX = 63)`
- A simple priority scheduler in Pintos:
  - Make `next_thread_to_run()` return the thread with highest priority
  - implement `thread_set_priority()`
    yield if the new priority is not the highest!
  - make the current thread yield when a higher priority thread is created

Priority scheduling

do you see any problem with this scheduler?

Priority scheduling

- Priorities are fixed, unless the user calls `thread_set_priority(...)`
- A cpu-hungry high priority thread may:
  - make the system less responsive (e.g., to I/O)
  - hog the cpu: lower priority threads may starve!

Priority scheduling

- Priorities are fixed, unless the user calls `thread_set_priority(...)`
- A cpu-hungry high priority thread may:
  - make the system less responsive (e.g., to I/O)
  - hog the cpu: lower priority threads may starve!

Solution: advanced scheduler with automatic priorities

Advanced scheduling

(or multilevel feedback queue scheduling - mlfqs)

Advanced scheduling

- The user sets a "nice" value
  (ranging from -20 to 20, with 0 as default)
- Priority is automatically adjusted by the OS
  (based on the thread's niceness)
- cpu-hungry threads lose priority
- low-computation threads (like I/O) automatically get higher priorities
  - the system becomes more responsive to I/O

Advanced scheduling

- To set priorities automatically:
  - each thread has a `nice` value
    (again, from -20 to 20)
  - each thread has a `recent_cpu` value
    (indicating how much cpu the thread has used recently)
  - global variable `load_avg` stores the system load
    (in number of ready+running threads per second in the last minute)

- To adjust each thread's priority:

```
priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)
```

Advanced scheduling

`nice` (per thread): type int
initial value: set by user
user can change with `thread_set_nice()`

`load_avg` (global): type fixed-point real
initial value: `0`
every second:
`load_avg = (59/60)*load_avg + (1/60)*ready_or_running_threads`

`recent_cpu` (per thread): type fixed-point real
initial value: `0`
every tick: increase `recent_cpu` by `1` for current thread
every second:
`recent_cpu = (2*load_avg)/(2*load_avg + 1) * recent_cpu + nice`

`priority` (per thread): type int
initial value: calculated based on the initial value of `nice`
every 4 ticks:
`priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)`

Advanced scheduling

`nice` (per thread): type int
- initial value: set by user
- user can change with `thread_set_nice()`

`load_avg` (global): type fixed-point real
- initial value: `0`
- every second:

```
load_avg = (59/60)*load_avg + (1/60)*ready_or_running_threads
```

`recent_cpu` (per thread): type fixed-point real
- initial value: `0`
- every tick: increase `recent_cpu` by `1` for current thread
- every second:

```
recent_cpu = (2*load_avg)/(2*load_avg + 1) * recent_cpu + nice
```

`priority` (per thread): type int
- initial value: calculated based on the initial value of `nice`
- every 4 ticks:

```
priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)
```

Fixed-Point Real

- `load_avg` and `recent_cpu` are real numbers
- Usually, no float operations in the kernel
- Fixed-point real variables are a solution
  - Implemented using integers
  - You can use the header `fpr_arith.h`, available on moodle

Advanced scheduling

- Functions that must be implemented:
  - `thread_set_nice(int)`:
    update the thread nice, recalculate its priority based on that and yield if the new priority is not the highest anymore
  - `thread_get_nice()` : return the thread's nice value
  - `thread_get_load_avg()` : return `load_avg * 100`
  - `thread_get_recent_cpu()`:
    return current thread's `recent_cpu * 100`
- The current thread must yield if a thread with higher priority (based on its `nice` value) is created

Advanced scheduling

- Check the bool variable `thread_mlfqs`
  - It tells whether the option `-mlfqs` was passed to the kernel
  - If true, use the advanced scheduler
  - If false, use the simple priority scheduler

Assignment

- In class: implement the simple priority scheduler
- Home: extend your scheduler with the advanced scheduling
  - it must be possible to select which one, by giving (or omitting) the -mlfqs option

Assignment

```
$ make check
...
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-change
FAIL tests/threads/priority-donate-one
FAIL tests/threads/priority-donate-multiple
FAIL tests/threads/priority-donate-multiple2
FAIL tests/threads/priority-donate-nest
FAIL tests/threads/priority-donate-sema
FAIL tests/threads/priority-donate-lower
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
FAIL tests/threads/priority-sema
FAIL tests/threads/priority-condvar
FAIL tests/threads/priority-donate-chain
pass tests/threads/mlfqs-load-1
pass tests/threads/mlfqs-load-60
pass tests/threads/mlfqs-load-avg
pass tests/threads/mlfqs-recent-1
pass tests/threads/mlfqs-fair-2
pass tests/threads/mlfqs-fair-20
pass tests/threads/mlfqs-nice-2
pass tests/threads/mlfqs-nice-10
FAIL tests/threads/mlfqs-block
10 of 27 tests failed.
```

Readings

- Read the Pintos documentation
- Chapter 2
  - 2.2.3
  - 2.2.4
  - 2.3
- Appendix B