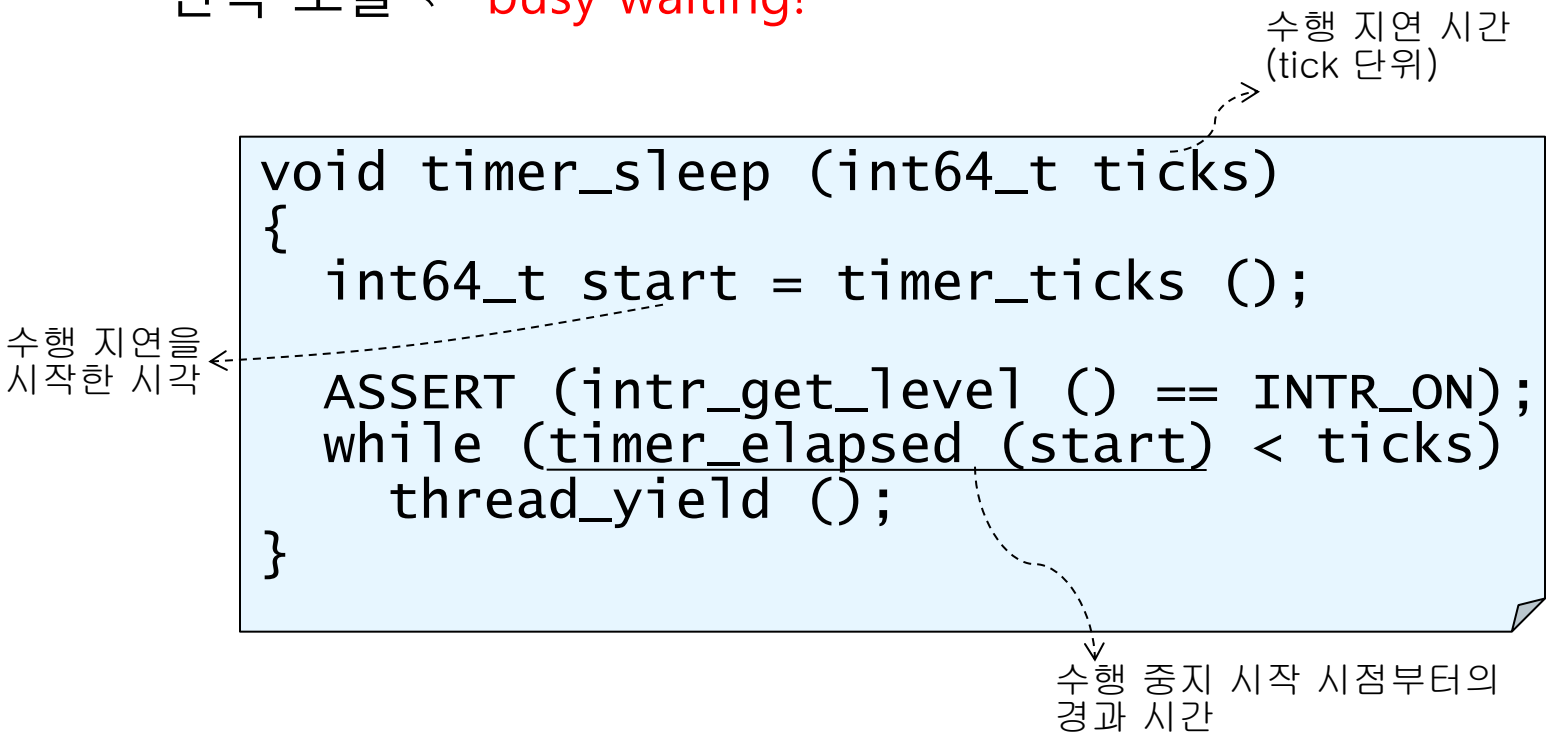
1. ALARM CLOCK 개선

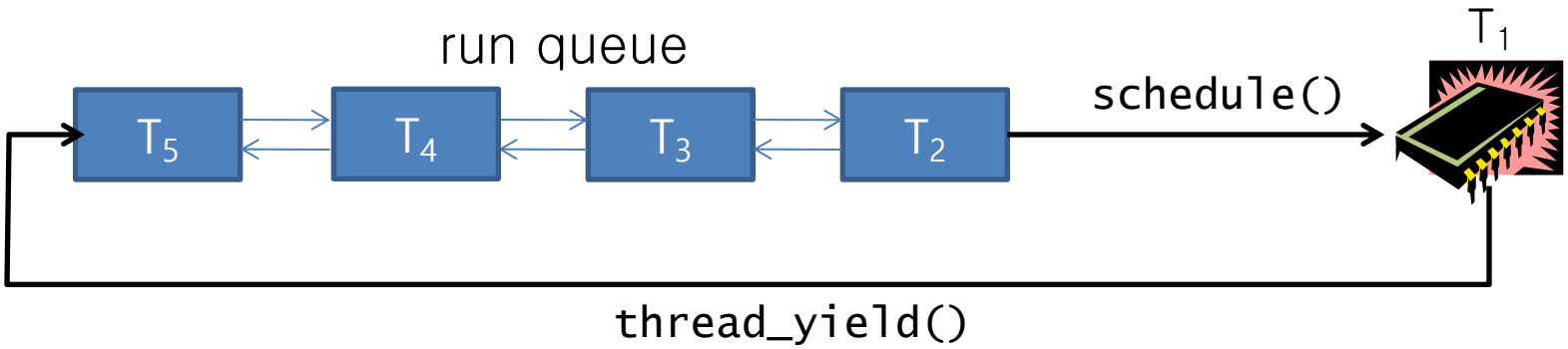
Alarm Clock

- `void timer_sleep (int64_t ticks);`
  - 이를 호출한 쓰레드의 수행을 호출 시점부터 `ticks` 틱 동안 지연하는데 사용
  - 현재의 구현: `ticks` 틱이 경과할 때까지 `thread_yield()` 를 반복 호출 ← busy waiting!



현재 alarm clock 동작 – alarm-multiple의 예

```
for (i = 1; i <= test->iterations; i++) {
  int64_t sleep_until =
              test->start + i * t->duration;
  timer_sleep (sleep_until-timer_ticks ());
  lock_acquire (&test->output_lock);
  *test->output_pos++ = t->id;
  lock_release (&test->output_lock);
}
```



thread_block() and thread_unblock()

```
/* This function must be
   called with interrupts
   turned off */

Void thread_block (void)
{
  ASSERT (!intr_context ());
  ASSERT (intr_get_level ()
          == INTR_OFF);

  thread_current()->status
    = THREAD_BLOCKED;
  schedule ();
}
```

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

  ASSERT (is_thread (t));

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

Semaphore – thread_block() / thread_unblock()과 wait queue의 예

```
struct semaphore
{
  /* Current value. */
  unsigned value;
  /* List of waiting threads. */
  struct list waiters;
};
```

```
struct thread
{
  …

  struct list_elem elem;
  …
};
```

```
void sema_down (struct semaphore *sema)
{
  …

  old_level = intr_disable ();
  while (sema->value == 0) }
    list_push_back (&sema->waiters,
                    &thread_current ()->elem);
    thread_block ();
  }
  sema->value--;
  intr_set_level (old_level);
}
```

/* wait(S) or P(S) */

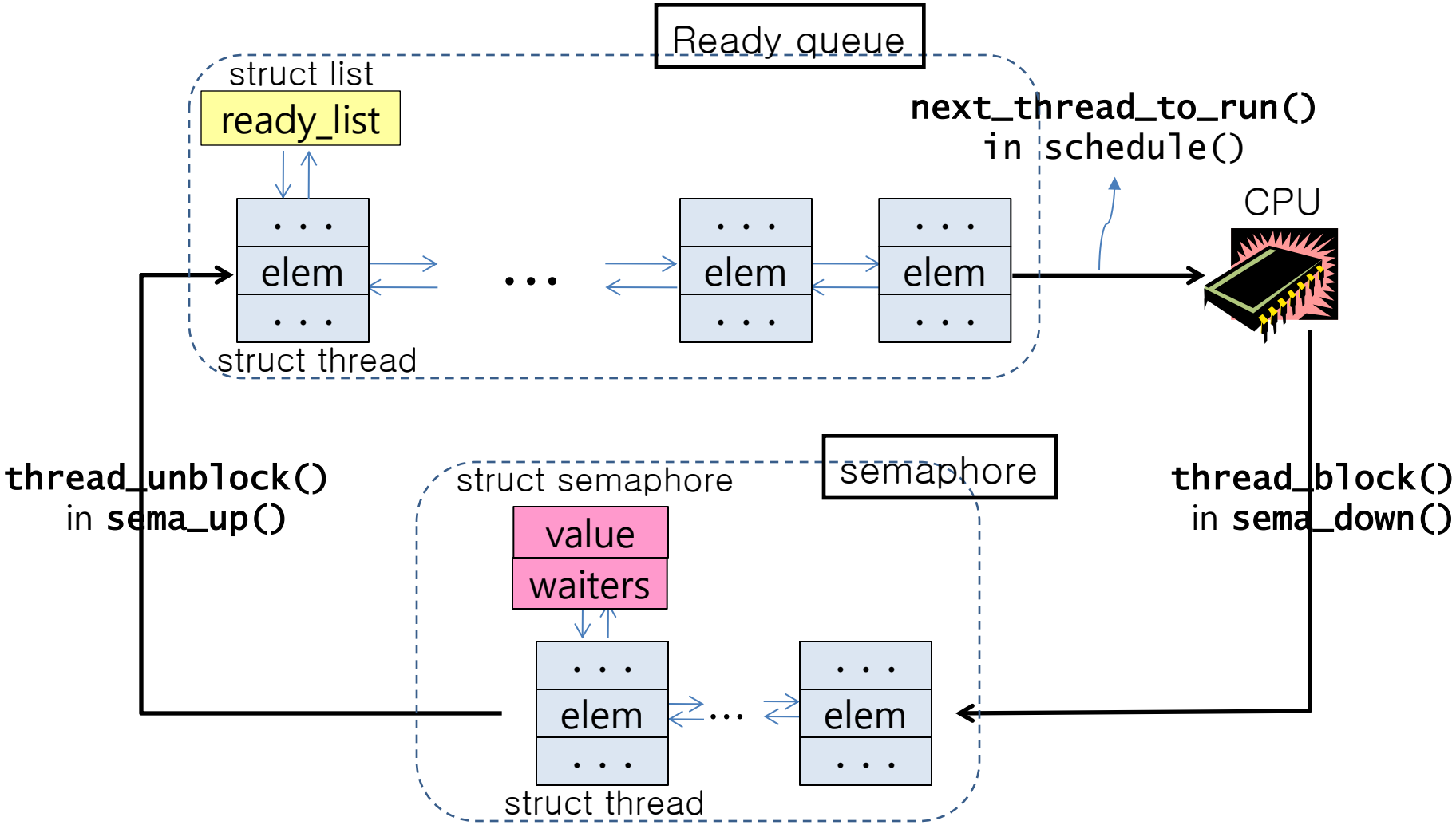
```
void sema_up (struct semaphore *sema)
{
  enum intr_level old_level;

  ASSERT (sema != NULL);

  old_level = intr_disable ();
  if (!list_empty (&sema->waiters))
    thread_unblock (
      list_entry (list_pop_front (&sema->waiters),
      struct thread, elem)
  );
  sema->value++;
  intr_set_level (old_level);
}
```

/* signal(S) or V(S) */

Semaphore – thread_block() / thread_unblock()과 list 사용 예 (cont'd)



Alarm Clock의 개선 – No Busy-wait Alarm Clock

- 자료구조
  - 스레드들이 요청한 알람을 유지하는 linked list
  - 알람을 위한 구조체의 예

```
struct list alarms;

struct alarm
{
  int64_t expiration; // 알람 만료 시간
  struct thread *th;  // 알람 요청한 스레드
  struct list_elem elem;  // 알람 리스트 연결 필드
  …
};
```

Alarm Clock의 개선 – No Busy-wait Alarm Clock (cont'd)

- `timer_sleep()` 에서는
  - 알람을 해당 스레드와 함께 위 리스트에 매단 후,
  - `thread_block()`을 호출, 해당 스레드를 블록시킴

- 타이머 인터럽트 핸들러 `timer_handler()` 에서는
  - 매 틱 발생 시마다 인터럽트 처리 과정에서 위 리스트 중 시간이 만료된 알람이 있는지 검사하고,
  - 만료된 알람이 있는 경우, 알람은 리스트에서 제거하고 해당 스레드는 `thread_unblock()`을 호출, 스레드를 깨움

LINKED LIST IN PINTOS

Linked list in Pintos

- Doubly linked list implementation in Pintos
  - lib/kernel/list.h, lib/kernel/list.c /* 코드에 포함된 커멘트들을 반드시 읽어볼 것 */
  - Ready queue 나 semaphore의 wait queue 등의 구현에 사용
- Declaration

```
struct foo {
    struct list_elem elem;
    int bar;
    ...other members...
};
```

- Initialization

```
struct list foo_list;
list_init (&foo_list);
```

- Eg) Iteration

```
struct list_elem *e;
for (e = list_begin (&foo_list);
  e != list_end (&foo_list);
  e = list_next (e)) {
    struct foo *f = list_entry (e, struct foo, elem);
    ...do something with f...
}
```

List Operations

- List initialization

```
void list_init (struct list *);
```

- List traversal

```
struct list_elem *list_begin (struct list *);
struct list_elem *list_next (struct list_elem *);
struct list_elem *list_end (struct list *);

struct list_elem *list_rbegin (struct list *);
struct list_elem *list_prev (struct list_elem *);
struct list_elem *list_rend (struct list *);

struct list_elem *list_head (struct list *);
struct list_elem *list_tail (struct list *);
```

- List insertion

```
void list_insert (struct list_elem *,
                  struct list_elem *);
void list_splice (struct list_elem *before,
                  struct list_elem *first,
                  struct list_elem *last);
void list_push_front (struct list *, struct
list_elem *);
void list_push_back (struct list *, struct
list_elem *);
```

- List removal

```
struct list_elem *list_remove (struct list_elem *);
struct list_elem *list_pop_front (struct list *);
struct list_elem *list_pop_back (struct list *);
```

- List elements

```
struct list_elem *list_front (struct list *);
struct list_elem *list_back (struct list *);
```

- List properties

```
size_t list_size (struct list *);
bool list_empty (struct list *);
```

- Miscellaneous

```
void list_reverse (struct list *);
```

- Compares the value of two list

```
typedef bool list_less_func (const struct
list_elem *a, const struct list_elem *b, void
*aux);
```

- Operations on lists with ordered elements

```
void list_sort (struct list *, list_less_func *,
void *aux);
void list_insert_ordered (struct list *, struct
list_elem *, list_less_func *, void *aux);
void list_unique (struct list *, struct list
*duplicates,list_less_func *, void *aux);
```

- Max and min

```
struct list_elem *list_max (struct list *,
list_less_func *, void *aux);
struct list_elem *list_min (struct list *,
list_less_func *, void *aux);
```