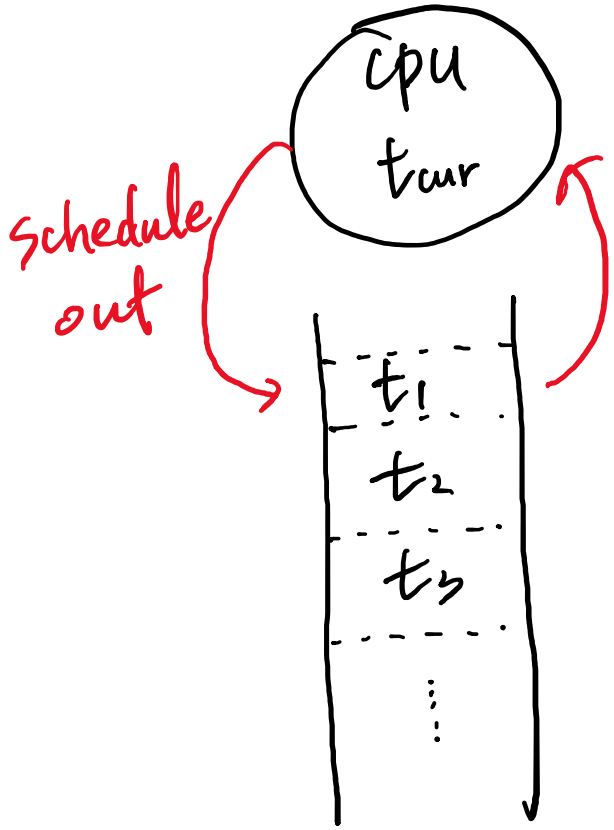
Scheduler concept



scheduler:

- pick a thread from run queue
- kick out currently running thread
- run the picked thread

context switch

When is the scheduler invoked?

Pinto scheduler

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

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

  if (curr != next)
    prev = switch_threads (curr, next);
  schedule_tail (prev);
}
```

Pick the next thread to run

```
/* Chooses and returns the next thread to be scheduled.  Should
   return a thread from the run queue, unless the run queue is
   empty.  (If the running thread can continue running, then it
   will be in the run queue.)  If the run queue is empty, return
   idle_thread. */
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);
}
```

List_entry?

List_entry

```
/* Converts pointer to list element LIST_ELEM into a pointer to
   the structure that LIST_ELEM is embedded inside.  Supply the
   name of the outer structure STRUCT and the member name MEMBER
   of the list element.  See the big comment at the top of the
   file for an example. */
#define list_entry(LIST_ELEM, STRUCT, MEMBER)           \
        ((STRUCT *) ((uint8_t *) &(LIST_ELEM)->next     \
                     - offsetof (STRUCT, MEMBER.next)))
```

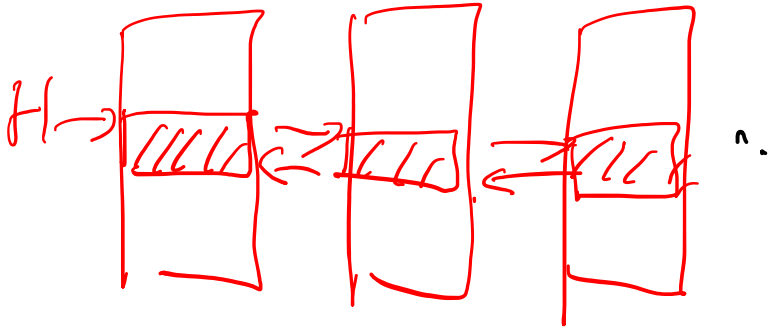
```
/* List element. */
struct list_elem
  {
    struct list_elem *prev;     /* Previous list element. */
    struct list_elem *next;     /* Next list element. */
  };

/* List. */
struct list
  {
    struct list_elem head;      /* List head. */
    struct list_elem tail;      /* List tail. */
  };
```

```
/* A linked list element. */
struct value
  {
    struct list_elem elem;      /* List element. */
    int value;                  /* Item value. */
  };
```

```
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *) 0)->MEMBER)
```

Struct list_elem *e;
Struct list l;

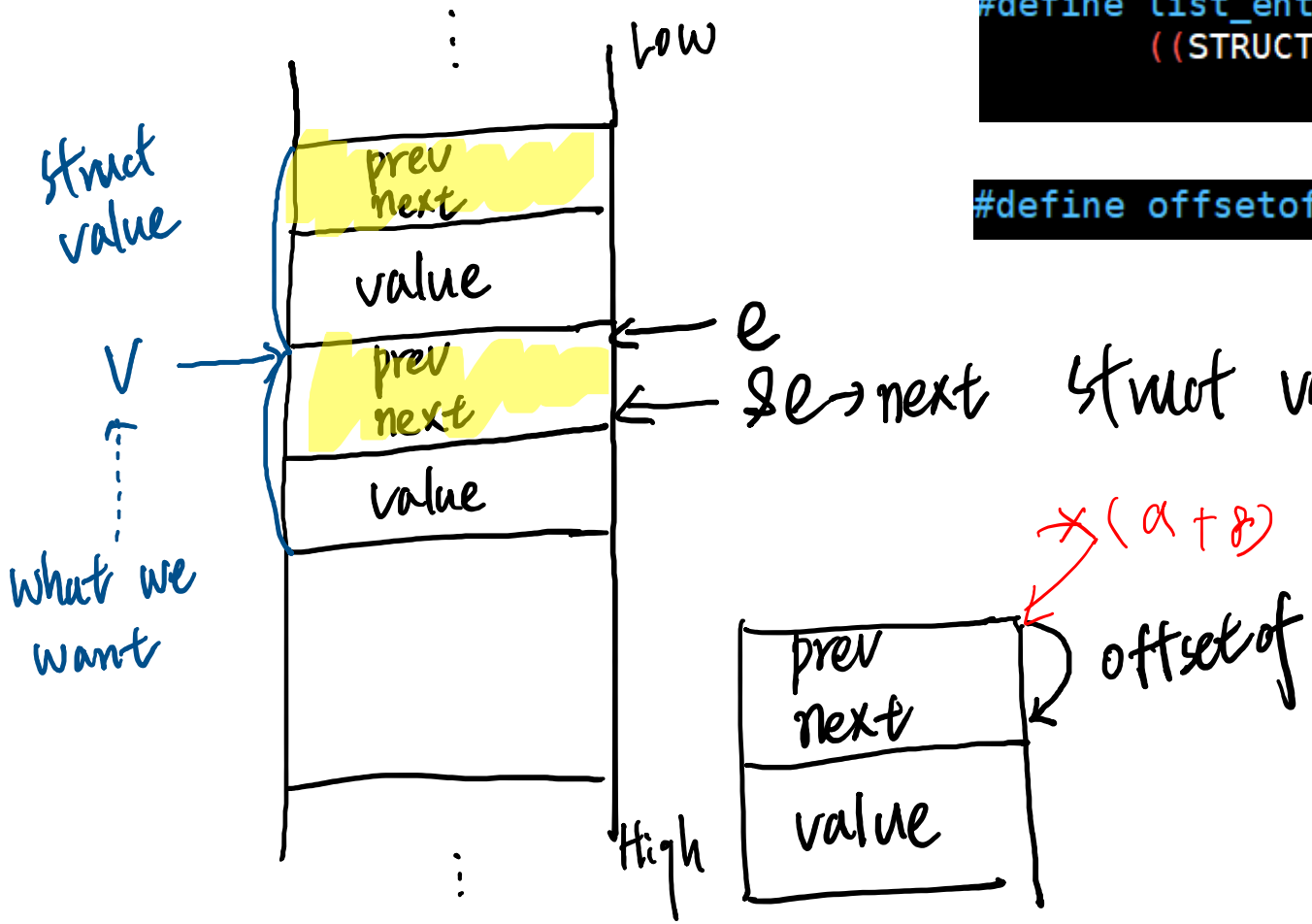
[Add some nodes to the list l]



e = &(l->head);
Struct value *v = list_entry(e, struct value, elem)

v->value = 4;

List_entry




```
/* Converts pointer to list element LIST_ELEM into a pointer to
   the structure that LIST_ELEM is embedded inside.  Supply the
   name of the outer structure STRUCT and the member name MEMBER
   of the list element.  See the big comment at the top of the
   file for an example. */
#define list_entry(LIST_ELEM, STRUCT, MEMBER)           \
        ((STRUCT *) ((uint8_t *) &(LIST_ELEM)->next     \
                     - offsetof (STRUCT, MEMBER.next)))
```

```
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *) 0)->MEMBER)
```

struct value *v = list_entry (e, struct value, elem)

```
/* A linked list element. */
struct value
  {
    struct list_elem elem;      /* List element. */
    int value;                  /* Item value. */
  };
```

```
/* List element. */
struct list_elem
  {
    struct list_elem *prev;     /* Previous list element. */
    struct list_elem *next;     /* Next list element. */
  };
```

Now you can decipher the function!

```
/* Chooses and returns the next thread to be scheduled.  Should
   return a thread from the run queue, unless the run queue is
   empty.  (If the running thread can continue running, then it
   will be in the run queue.)  If the run queue is empty, return
   idle_thread. */
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);
}
```

pick the first entry from run queue (return list_elem pointer)

compute the addr of "struct thread"

Priority scheduling

thread_create()

On a thread creation, if the thread has a higher priority than a current thread, then kickout the current running thread and schedule the created one.

thread_get_priority()

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

  ASSERT (!intr_context ());

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