threads

why threads?

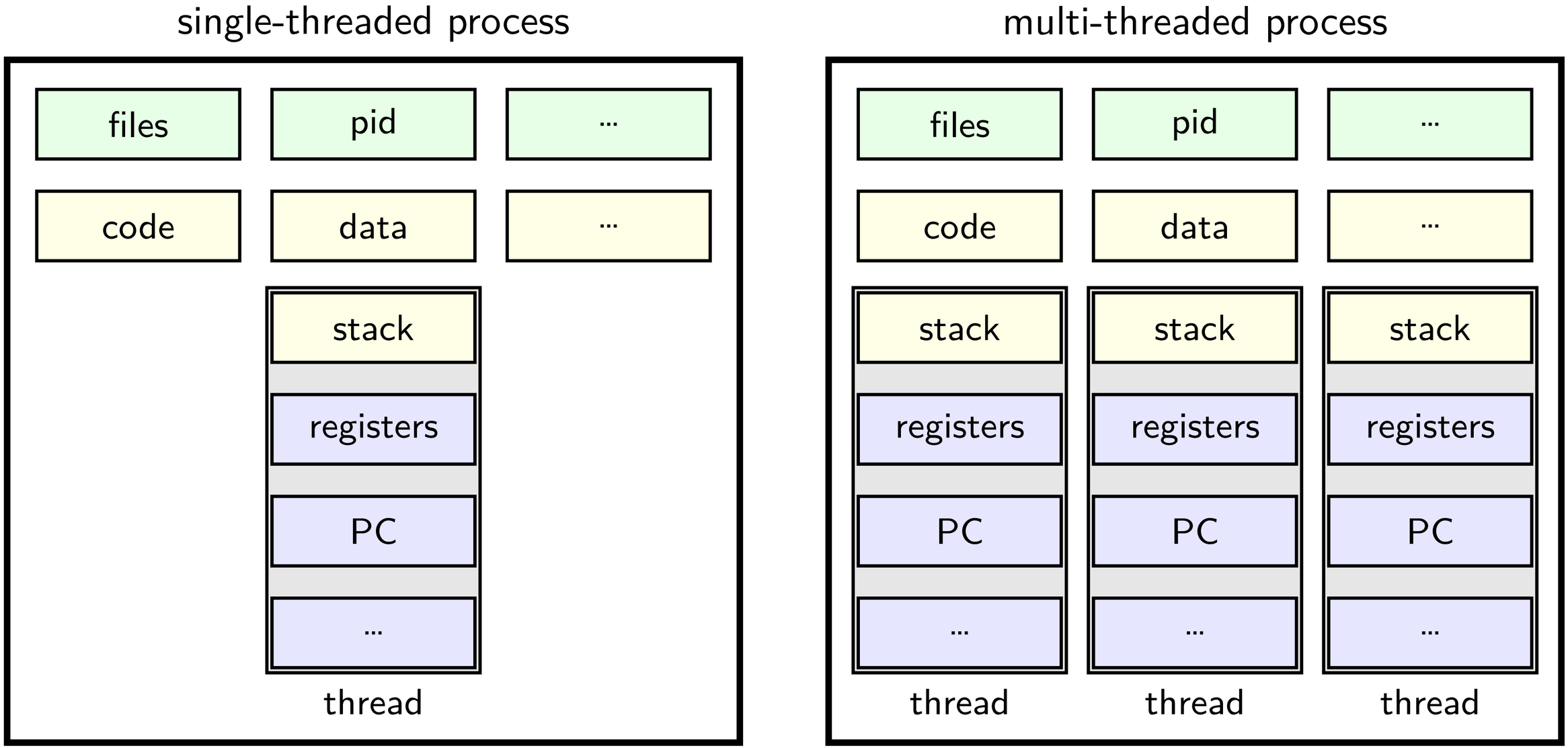
concurrency: different things happening at once

- one thread per user of web server?
- one thread per page in web browser?
- one thread to play audio, one to read keyboard, …?
- …

parallelism: do same thing with more resources

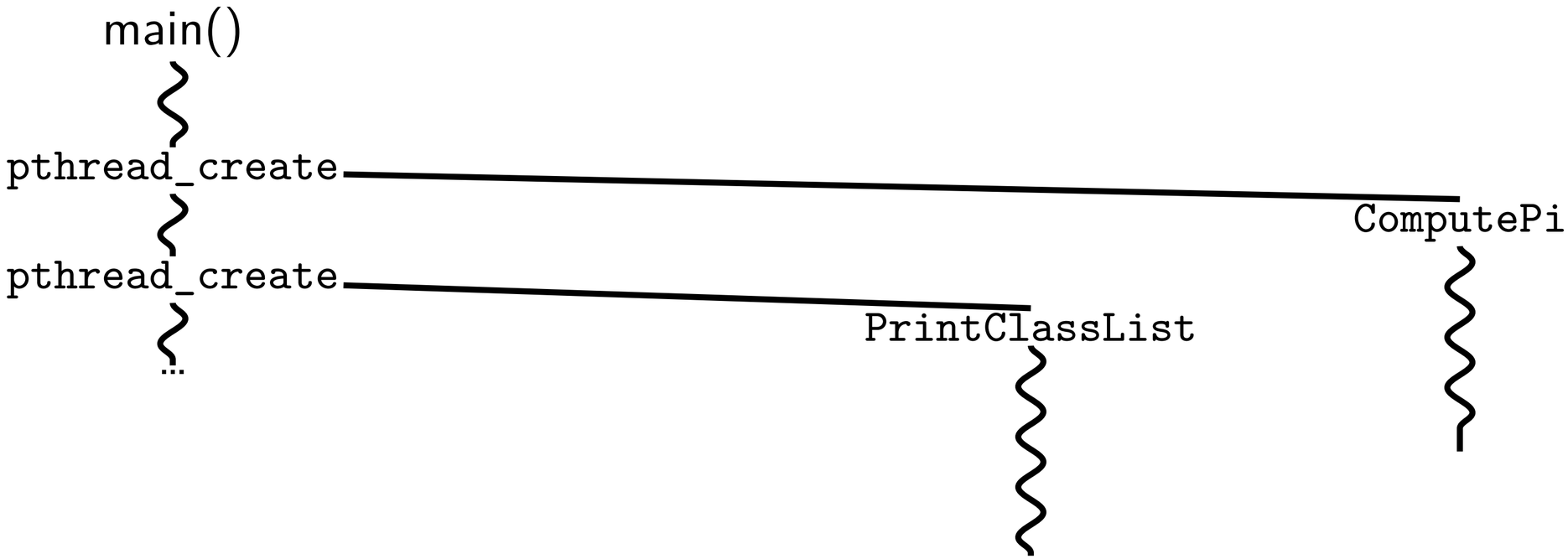
- multiple processors to speed-up simulation (life assignment)

single and multithread processes



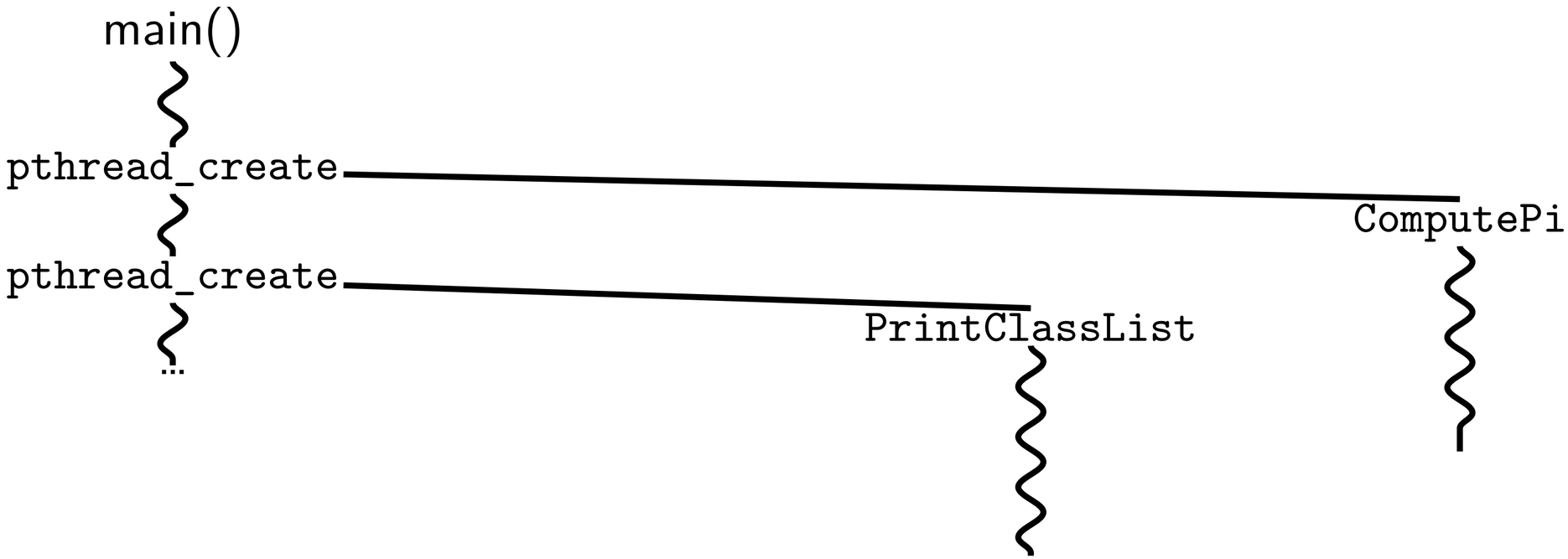
pthread_create

```
void *ComputePi(void *argument) { ...  }
void *PrintClassList(void *argument) { ...  }
int main() {
  pthread_t pi_thread, list_thread;
  if (0 != pthread_create(&pi_thread, NULL, ComputePi, NULL))
      handle_error();
  if (0 != pthread_create(&list_thread, NULL, PrintClassList, NULL))
      handle_error();
  ... /* more code */
}
```



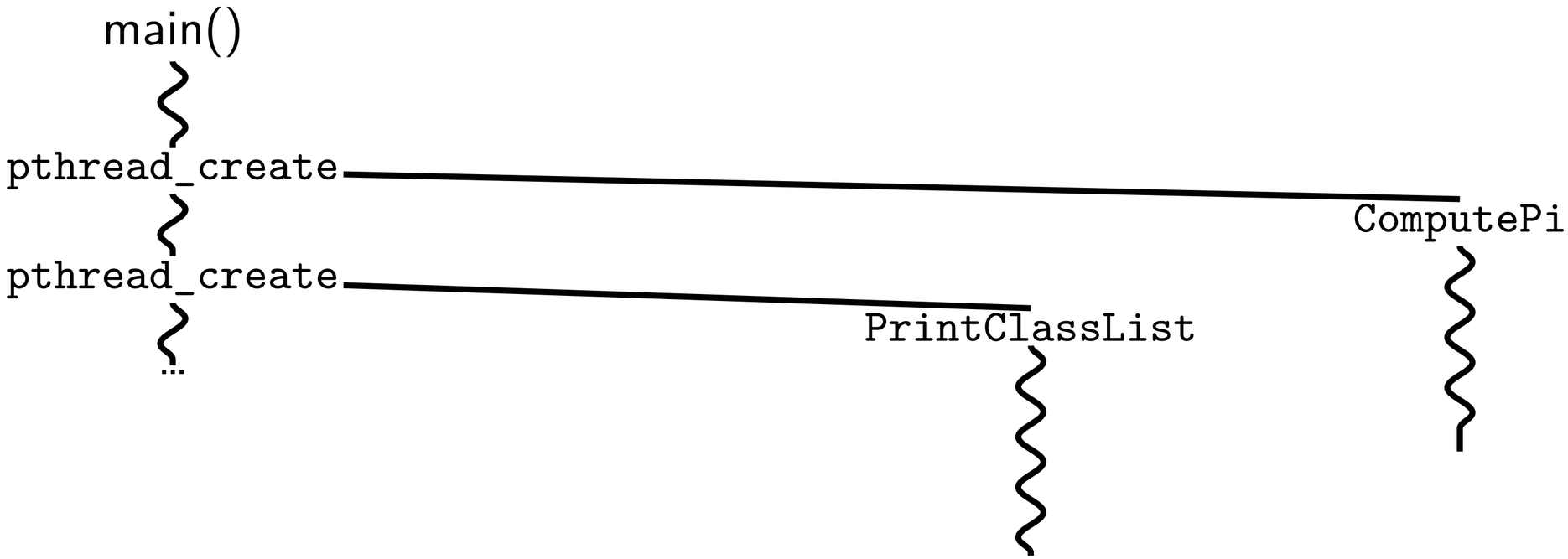
pthread_create

```
void *ComputePi(void *argument) { ...  }
void *PrintClassList(void *argument) { ...  }
int main() {
  pthread_t pi_thread, list_thread;
  if (0 != pthread_create(&pi_thread, NULL, ComputePi, NULL))
      handle_error();
  if (0 != pthread_create(&list_thread, NULL, PrintClassList, NULL))
      handle_error();
  ... /* more code */
}
```



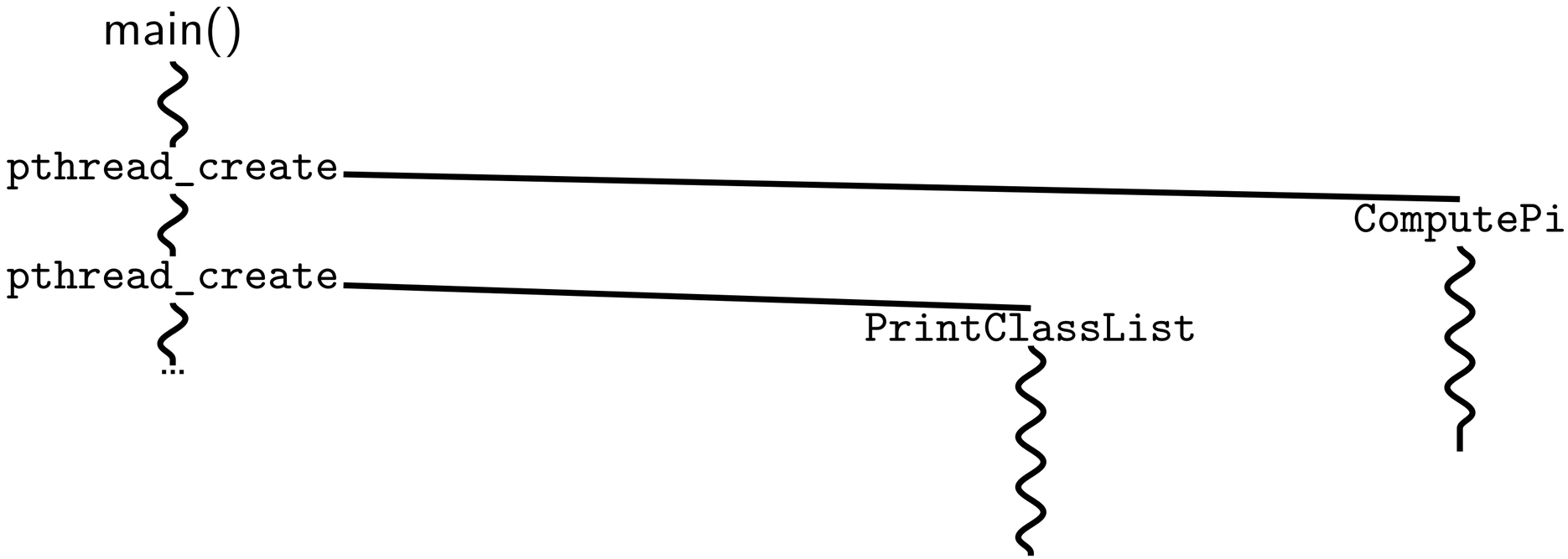
pthread_create

```
void *ComputePi(void *argument) { ...  }
void *PrintClassList(void *argument) { ...  }
int main() {
  pthread_t pi_thread, list_thread;
  if (0 != pthread_create(&pi_thread, NULL, ComputePi, NULL))
      handle_error();
  if (0 != pthread_create(&list_thread, NULL, PrintClassList, NULL))
      handle_error();
  ... /* more code */
}
```



pthread_create

```
void *ComputePi(void *argument) { ...  }
void *PrintClassList(void *argument) { ...  }
int main() {
  pthread_t pi_thread, list_thread;
  if (0 != pthread_create(&pi_thread, NULL, ComputePi, NULL))
      handle_error();
  if (0 != pthread_create(&list_thread, NULL, PrintClassList, NULL))
      handle_error();
  ... /* more code */
}
```



pthread_create

```
void *ComputePi(void *argument) { ...  }
void *PrintClassList(void *argument) { ...  }
int main() {
  pthread_t pi_thread, list_thread;
  if (0 != pthread_create(&pi_thread, NULL, ComputePi, NULL))
      handle_error();
  if (0 != pthread_create(&list_thread, NULL, PrintClassList, NULL))
      handle_error();
  ... /* more code */
}
```

pthread_create arguments:

thread identifier

function to run thread starts here, terminates if this function returns

thread attributes (extra settings) and function argument

pthread_create

```
void *ComputePi(void *argument) { ...  }
void *PrintClassList(void *argument) { ...  }
int main() {
  pthread_t pi_thread, list_thread;
  if (0 != pthread_create(&pi_thread, NULL, ComputePi, NULL))
      handle_error();
  if (0 != pthread_create(&list_thread, NULL, PrintClassList, NULL))
      handle_error();
  ... /* more code */
}
```

pthread_create arguments:

thread identifier

function to run thread starts here, terminates if this function returns

thread attributes (extra settings) and function argument

pthread_create

```
void *ComputePi(void *argument) { ...  }
void *PrintClassList(void *argument) { ...  }
int main() {
  pthread_t pi_thread, list_thread;
  if (0 != pthread_create(&pi_thread, NULL, ComputePi, NULL))
      handle_error();
  if (0 != pthread_create(&list_thread, NULL, PrintClassList, NULL))
      handle_error();
  ... /* more code */
}
```

pthread_create arguments:

thread identifier

function to run thread starts here, terminates if this function returns

thread attributes (extra settings) and function argument

pthread_create

```
void *ComputePi(void *argument) { ...  }
void *PrintClassList(void *argument) { ...  }
int main() {
  pthread_t pi_thread, list_thread;
  if (0 != pthread_create(&pi_thread, NULL, ComputePi, NULL))
      handle_error();
  if (0 != pthread_create(&list_thread, NULL, PrintClassList, NULL))
      handle_error();
  ... /* more code */
}
```

pthread_create arguments:

thread identifier

function to run thread starts here, terminates if this function returns

thread attributes (extra settings) and function argument

a threading race

```c
#include <pthread.h>
#include <stdio.h>
void *print_message(void *ignored_argument) {
    printf("In the thread\n");
    return NULL;
}
int main() {
    printf("About to start thread\n");
    pthread_t the_thread;
    /* assume does not fail */
    pthread_create(&the_thread, NULL, print_message, NULL);
    printf("Done starting thread\n");
    return 0;
}
```

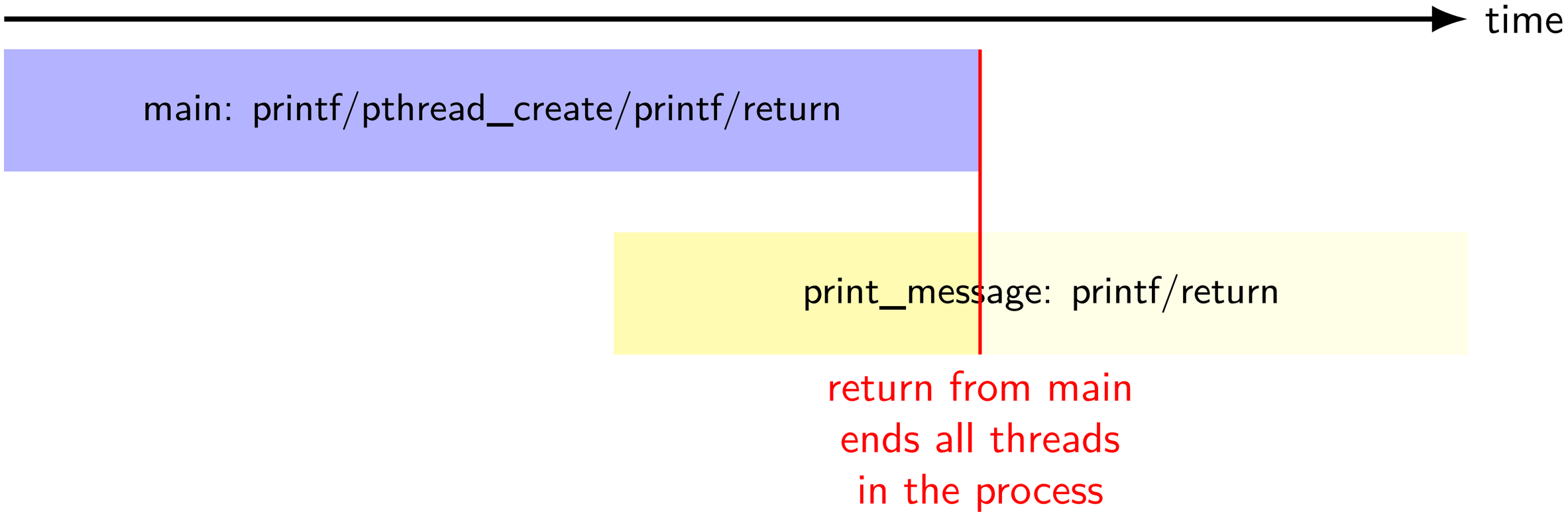
My machine: outputs `In the thread` *about 4% of the time*.
What happened?

a race

returning from main *exits the entire process* (all its threads)

same as calling exit; not like other threads

race: main's return 0 or print_message's printf first?



fixing the race (version 1)

```
#include <pthread.h>
#include <stdio.h>
void *print_message(void *ignored_argument) {
    printf("In the thread\\n");
    return NULL;
}
int main() {
    printf("About to start thread\\n");
    pthread_t the_thread;
    /* missing: error checking */
    pthread_create(&the_thread, NULL, print_message, NULL);
    printf("Done starting thread\\n");
    pthread_join(the_thread, NULL);  /* WAIT FOR THREAD */
    return 0;
}
```

fixing the race (version 1)

```
#include <pthread.h>
#include <stdio.h>
void *print_message(void *ignored_argument) {
    printf("In the thread\\n");
    return NULL;
}
int main() {
    printf("About to start thread\\n");
    pthread_t the_thread;
    /* missing: error checking */
    pthread_create(&the_thread, NULL, print_message, NULL);
    printf("Done starting thread\\n");
    pthread_join(the_thread, NULL);  /* WAIT FOR THREAD */
    return 0;
}
```

fixing the race (version 2; not recommended)

```
#include <pthread.h>
#include <stdio.h>
void *print_message(void *ignored_argument) {
    printf("In the thread\\n");
    return NULL;
}
int main() {
    printf("About to start thread\\n");
    pthread_t the_thread;
    /* missing: error checking */
    pthread_create(&the_thread, NULL, print_message, NULL);
    printf("Done starting thread\\n");
    pthread_exit(NULL);
}
```

fixing the race (version 2; not recommended)

```
#include <pthread.h>
#include <stdio.h>
void *print_message(void *ignored_argument) {
    printf("In the thread\\n");
    return NULL;
}
int main() {
    printf("About to start thread\\n");
    pthread_t the_thread;
    /* missing: error checking */
    pthread_create(&the_thread, NULL, print_message, NULL);
    printf("Done starting thread\\n");
    pthread_exit(NULL);
}
```

pthread_join, pthread_exit

R = `pthread_join`(X, `&`P): wait for thread X, copies return value into P

- like `waitpid`, but for a thread
- thread return value is pointer to anything
- R = 0 if successful, error code otherwise

`pthread_exit`: exit current thread, returning a value

- like `exit` or returning from main, but for a single thread
- same effect as returning from function passed to pthread_create

a note on error checking

from pthread_create manpage:

ERRORS

EAGAIN Insufficient resources to create another thread, or a system-imposed limit on the number of threads was encountered. The latter case may occur in two ways: the **RLIMIT_NPROC** soft resource limit (set via **setrlimit**(2)), which limits the number of process for a real user ID, was reached; or the kernel's system-wide limit on the number of threads, /proc/sys/kernel/threads-max, was reached.

EINVAL Invalid settings in attr.

EPERM No permission to set the scheduling policy and parameters specified in attr.

special constants for *return value*

same pattern for many other pthreads functions

pthread_join, pthread_mutex_… (later), …

will often omit error checking in slides for brevity

error checking pthread_create

```
int error = pthread_create(...);
if (error != 0) {
    /* print some error message */
}
```

sum example (only globals)

```
1  int values[1024];  int results[2];
2  void *sum_front(void *ignored_argument) {
3      int sum = 0;
4      for (int i = 0; i < 512; ++i) { sum += values[i]; }
5      results[0] = sum;
6      return NULL;
7  }
8  void *sum_back(void *ignored_argument) {
9      int sum = 0;
10     for (int i = 512; i < 1024; ++i) { sum += values[i]; }
11     results[1] = sum;
12     return NULL;
13 }
14 int sum_all() {
15     pthread_t sum_front_thread, sum_back_thread;
16     /* missing: error handling */
17     pthread_create(&sum_front_thread, NULL, sum_front, NULL);
18     pthread_create(&sum_back_thread, NULL, sum_back, NULL);
19     pthread_join(sum_front_thread, NULL); pthread_join(sum_back_thread, NULL);
20     return results[0] + results[1];
21 }
22
```

sum example (only globals)

```
1  int values[1024];  int results[2];
2  void *sum_front(void *ignored_argument) {
3      int sum = 0;
4      for (int i = 0; i < 512; ++i) { sum += values[i]; }
5      results[0] = sum;
6      return NULL;
7  }
8  void *sum_back(void *ignored_argument) {
9      int sum = 0;
10     for (int i = 512; i < 1024; ++i) { sum += values[i]; }
11     results[1] = sum;
12     return NULL;
13 }
14 int sum_all() {
15     pthread_t sum_front_threa
16     /* missing: error handlin
17     pthread_create(&sum_front
18     pthread_create(&sum_back_
19     pthread_join(sum_front_th                                  _L);
20     return results[0] + resul
21 }
22
```

values, results: global variables shared between all threads

sum example (only globals)

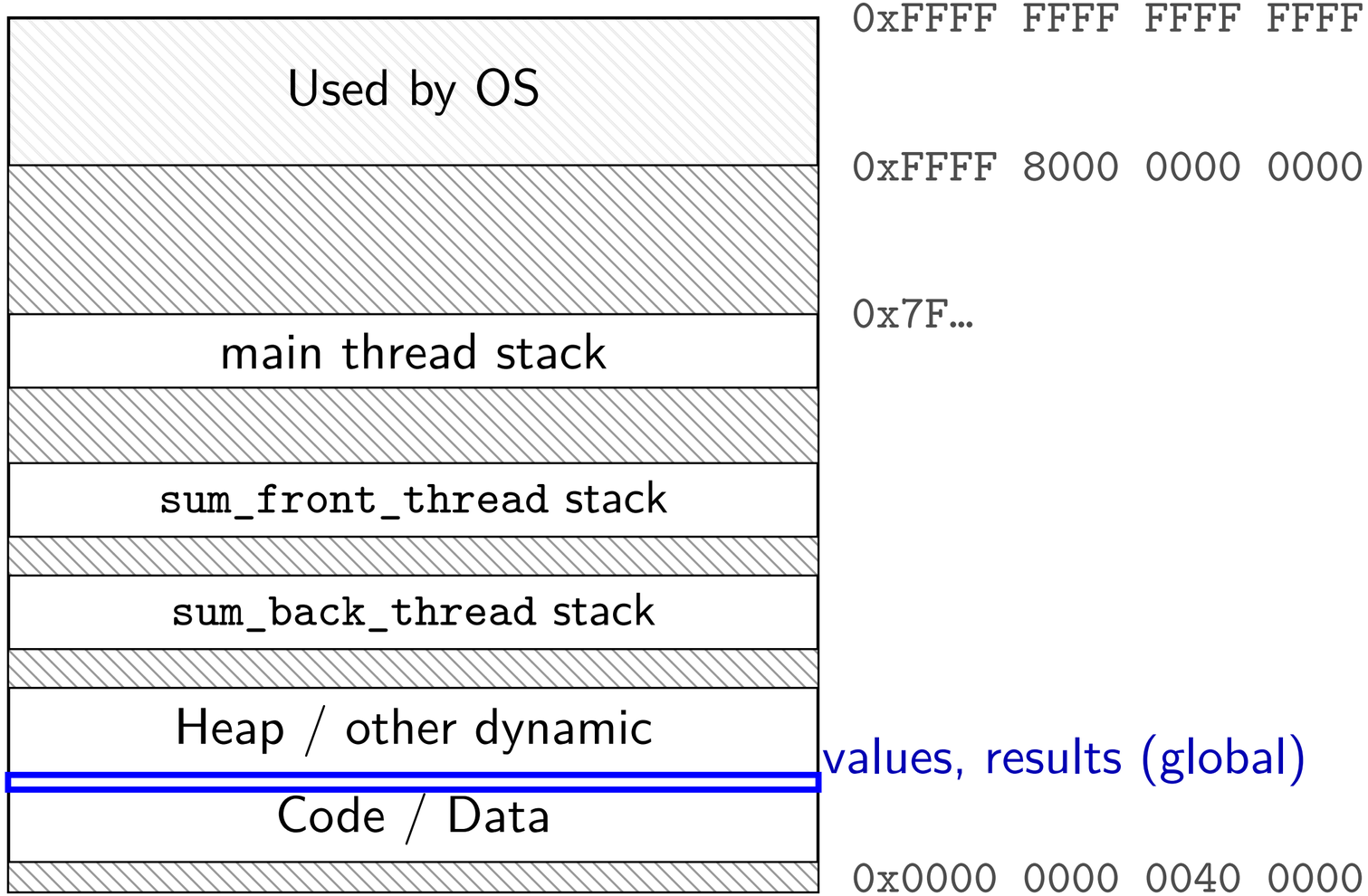
```
1  int values[1024];  int results[2];
2  void *sum_front(void *ignored_argument) {
3      int sum = 0;
4      for (int i = 0; i < 512; ++i) { sum += values[i]; }
5      results[0] = sum;
6      return NULL;
7  }
8  void *sum_back(void *ignored_argument) {
9      int sum = 0;
10     for (int i = 512; i < 1024; ++i) { sum += values[i]; }
11     results[1] = sum;
12     return NULL;
13 }
14 int sum_all() {
15     pthread_t sum_f
16     /* missing: err
17     pthread_create(
18     pthread_create(
19     pthread_join(su
20     return results[
21 }
22
```

two functions — same except some numbers

values returned from via global array `results`
used here instead of return value
(partly to illustrate memory is shared;
partly for later version that doesn't join)

thread_sum memory layout

thread_sum memory layout



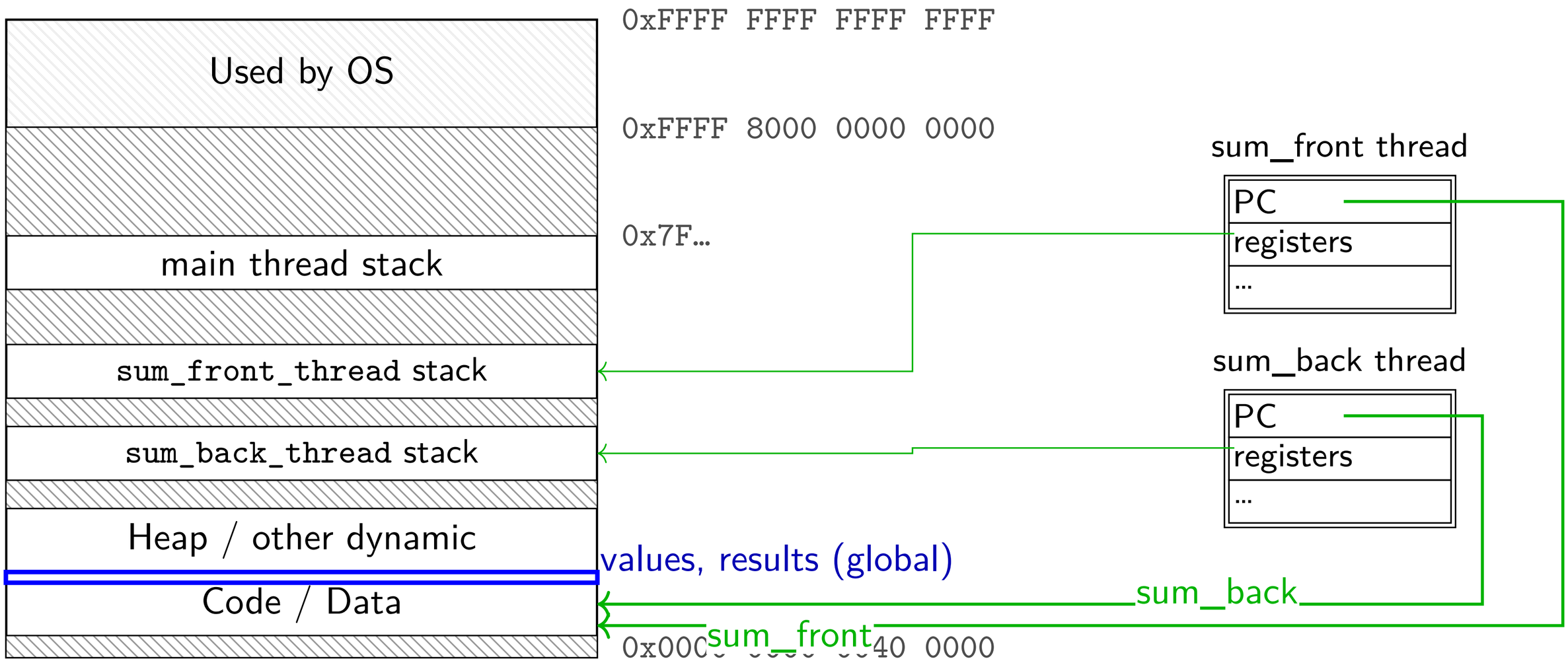
0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

0x0000 0000 0040 0000

thread_sum memory layout



sum example (to global, with thread IDs)

```
 1  int values[1024];
 2  int results[2];
 3  void *sum_thread(void *argument) {
 4      int id = (int) argument;
 5      int sum = 0;
 6      for (int i = id * 512; i < (id + 1) * 512; ++i) {
 7          sum += values[i];
 8      }
 9      results[id] = sum;
10      return NULL;
11  }
12  int sum_all() {
13      /* missing: error handling */
14      pthread_t thread[2];
15      for (int i = 0; i < 2; ++i) {
16          pthread_create(&threads[i], NULL, sum_thread, (void *) i);
17      }
18      for (int i = 0; i < 2; ++i)
19          pthread_join(threads[i], NULL);
20      return results[0] + results[1];
21  }
22
```

pass thread index (as fake pointer “address”)

sum example (to global, with thread IDs)

```
 1  int values[1024];
 2  int results[2];
 3  void *sum_thread(void *argument) {
 4      int id = (int) argument;
 5      int sum = 0;
 6      for (int i = id * 512; i < (id + 1) * 512; ++i) {
 7          sum += values[i];
 8      }
 9      results[id] = sum;
10      return NULL;
11  }
12  int sum_all() {
13      /* missing: error handling */
14      pthread_t thread[2];
15      for (int i = 0; i < 2; ++i) {
16          pthread_create(&threads[i], NULL, sum_thread, (void *) i);
17      }
18      for (int i = 0; i < 2; ++i)
19          pthread_join(threads[i], NULL);
20      return results[0] + results[1];
21  }
22
```

pass thread index (as fake pointer “address”)

sum example (to global, with thread IDs)

```
 1  int values[1024];
 2  int results[2];
 3  void *sum_thread(void *argument) {
 4      int id = (int) argument;
 5      int sum = 0;
 6      for (int i = id * 512; i < (id + 1) * 512; ++i) {
 7          sum += values[i];
 8      }
 9      results[id] = sum;
10      return NULL;
11  }
12  int sum_all() {
13      /* missing: error handling */
14      pthread_t thread[2];
15      for (int i = 0; i < 2; ++i) {
16          pthread_create(&threads[i], NULL, sum_thread, (void *) i);
17      }
18      for (int i = 0; i < 2; ++i)
19          pthread_join(threads[i], NULL);
20      return results[0] + results[1];
21  }
22
```

pass thread index (as fake pointer “address”)

sum example (to global, with thread IDs)

```
 1  int values[1024];
 2  int results[2];
 3  void *sum_thread(void *argument) {
 4      int id = (int) argument;
 5      int sum = 0;
 6      for (int i = id * 512; i < (id + 1) * 512; ++i) {
 7          sum += values[i];
 8      }
 9      results[id] = sum;
10      return NULL;
11  }
12  int sum_all() {
13      /* missing: error handling */
14      pthread_t thread[2];
15      for (int i = 0; i < 2; ++i) {
16          pthread_create(&threads[i], NULL, sum_thread, (void *) i);
17      }
18      for (int i = 0; i < 2; ++i)
19          pthread_join(threads[i], NULL);
20      return results[0] + results[1];
21  }
22
```

pass thread index (as fake pointer “address”)

sum example (info struct)

```
1  int values[1024];
2  struct ThreadInfo {
3      int start, end, result;
4  };
5  void *sum_thread(void *argument) {
6      struct ThreadInfo *my_info = (struct ThreadInfo *) argument;
7      int sum = 0;
8      for (int i = my_info->start; i < my_info->end; ++i) { sum += values[i]; }
9      my_info->result = sum;
10     return NULL;
11 }
12 int sum_all() {
13     pthread_t thread[2]; struct ThreadInfo info[2];
14     for (int i = 0; i < 2; ++i) {
15         info[i].start = i*512; info[i].end = (i+1)*512;
16         pthread_create(&threads[i], NULL, sum_thread, &info[i]);
17     }
18     for (int i = 0; i < 2; ++i) { pthread_join(threads[i], NULL); }
19     return info[0].result + info[1].result;
20 }
21
```

sum example (info struct)

```
1  int values[1024];
2  struct ThreadInfo {
3      int start, end, result;
4  };
5  void *sum_thread(void *argument) {
6      struct ThreadInfo *my_info = (struct ThreadInfo *) argument;
7      int sum = 0;
8      for (int i = my_info->start; i < my_info->end; ++i) { sum += values[i]; }
9      my_info->result = sum;
10     return NULL;
11 }
12 int sum_all() {
13     pthread_t thread[2]; struct ThreadInfo info[2];
14     for (int i = 0; i < 2; ++i) {
15         info[i].start = i*512; info[i].end = (i+1)*512;
16         pthread_create(&threads[i], NULL, sum_thread, &info[i]);
17     }
18     for (int i = 0; i < 2; ++i) { pthread_join(threads[i], NULL); }
19     return info[0].result + info[1].result;
20 }
21
```

sum example (info struct)

```
1  int values[1024];
2  struct ThreadInfo {
3      int start, end, result;
4  };
5  void *sum_thread(void *argument) {
6      struct ThreadInfo *my_info = (struct ThreadInfo *) argument;
7      int sum = 0;
8      for (int i = my_info->start; i < my_info->end; ++i) { sum += values[i]; }
9      my_info->result = sum;
10     return NULL;
11 }
12 int sum_all() {
13     pthread_t thread[2]; struct ThreadInfo info[2];
14     for (int i = 0; i < 2; ++i) {
15         info[i].start = i*512; info[i].end = (i+1)*512;
16         pthread_create(&threads[i], NULL, sum_thread, &info[i]);
17     }
18     for (int i = 0; i < 2; ++i) { pthread_join(threads[i], NULL); }
19     return info[0].result + info[1].result;
20 }
21
```

sum example (info struct)

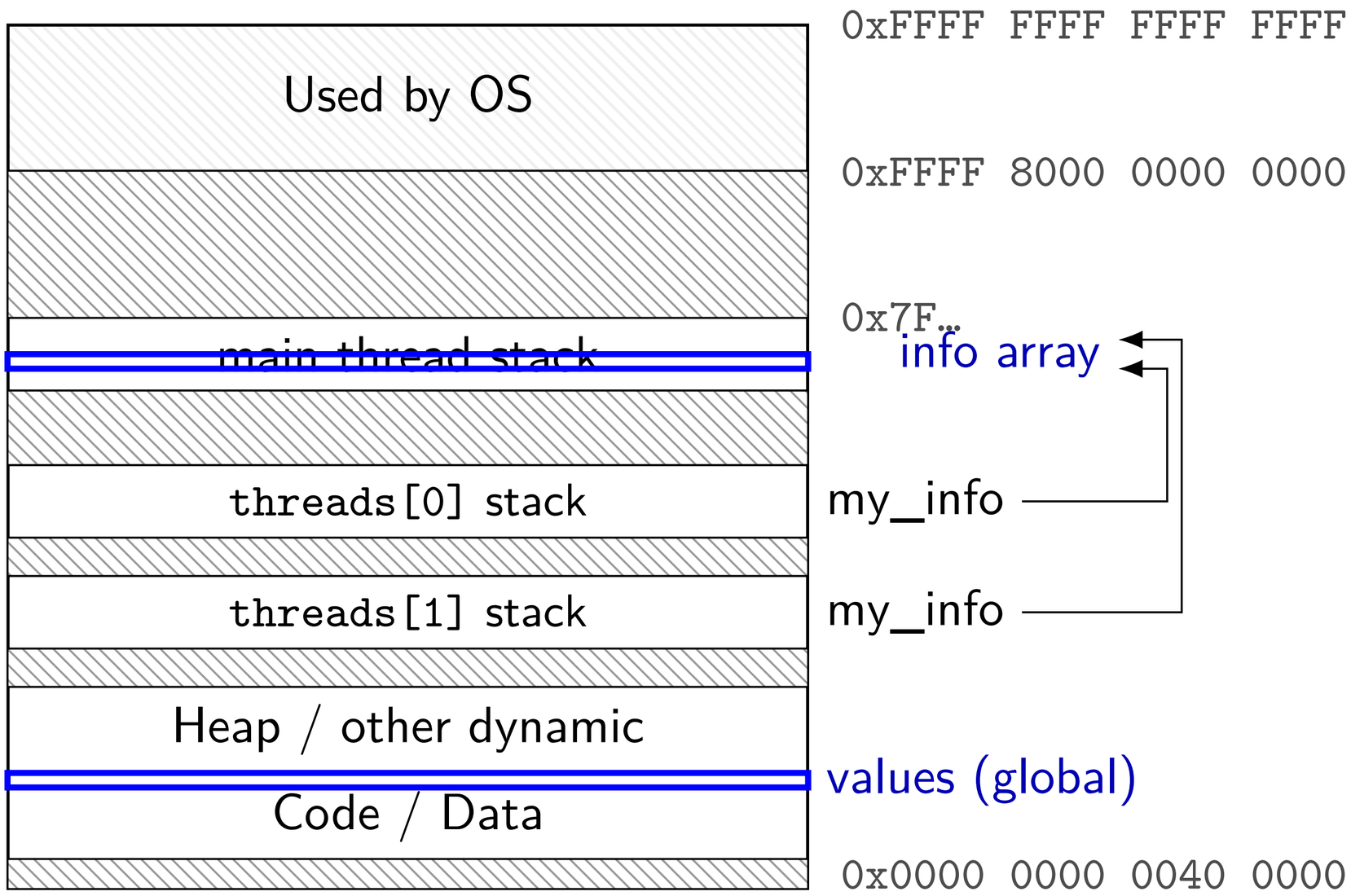
```
1  int values[1024];
2  struct ThreadInfo {
3      int start, end, result;
4  };
5  void *sum_thread(void *argument) {
6      struct ThreadInfo *my_info = (struct ThreadInfo *) argument;
7      int sum = 0;
8      for (int i = my_info->start; i < my_info->end; ++i) { sum += values[i]; }
9      my_info->result = sum;
10     return NULL;
11 }
12 int sum_all() {
13     pthread_t thread[2]; struct ThreadInfo info[2];
14     for (int i = 0; i < 2; ++i) {
15         info[i].start = i*512; info[i].end = (i+1)*512;
16         pthread_create(&threads[i], NULL, sum_thread, &info[i]);
17     }
18     for (int i = 0; i < 2; ++i) { pthread_join(threads[i], NULL); }
19     return info[0].result + info[1].result;
20 }
21
```

sum example (info struct)

```
1  int values[1024];
2  struct ThreadInfo {
3      int start, end, result;
4  };
5  void *sum_thread(void *argument) {
6      struct ThreadInfo *my_info = (struct ThreadInfo *) argument;
7      int sum = 0;
8      for (int i = my_info->start; i < my_info->end; ++i) { sum += values[i]; }
9      my_info->result = sum;
10     return NULL;
11 }
12 int sum_all() {
13     pthread_t thread[2]; struct ThreadInfo info[2];
14     for (int i = 0; i < 2; ++i) {
15         info
16         pthr
17     }
18     for (int
19     return
20 }
21
```

`my_info` = pointer to sum_all's stack
okay because sum_all doesn't return until thread is done

thread_sum memory layout (info struct)



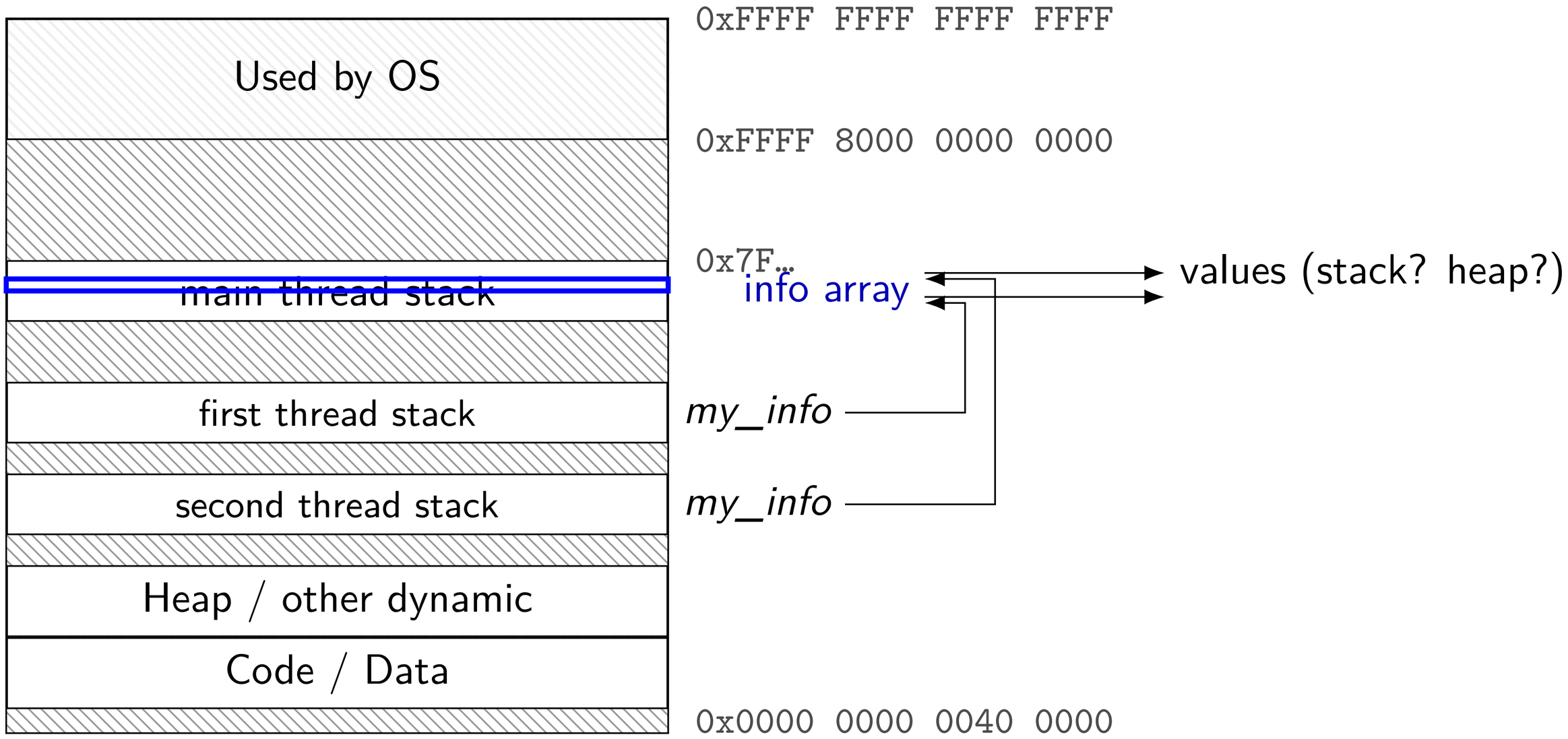
sum example (to main stack)

```
1  struct ThreadInfo { int *values; int start; int end; int result };
2  void *sum_thread(void *argument) {
3      ThreadInfo *my_info = (ThreadInfo *) argument;
4      int sum = 0;
5      for (int i = my_info->start; i < my_info->end; ++i) {
6          sum += my_info->values[i];
7      }
8      my_info->result = sum;
9      return NULL;
10 }
11 int sum_all(int *values) {
12     ThreadInfo info[2]; pthread_t thread[2];
13     for (int i = 0; i < 2; ++i) {
14         info[i].values = values; info[i].start = i*512; info[i].end = (i+1)*512;
15         pthread_create(&threads[i], NULL, sum_thread, (void *) &info[i]);
16     }
17     for (int i = 0; i < 2; ++i)
18         pthread_join(threads[i], NULL);
19     return info[0].result + info[1].result;
20 }
21
22
```

sum example (to main stack)

```
1  struct ThreadInfo { int *values; int start; int end; int result };
2  void *sum_thread(void *argument) {
3      ThreadInfo *my_info = (ThreadInfo *) argument;
4      int sum = 0;
5      for (int i = my_info->start; i < my_info->end; ++i) {
6          sum += my_info->values[i];
7      }
8      my_info->result = sum;
9      return NULL;
10 }
11 int sum_all(int *values) {
12     ThreadInfo info[2]; pthread_t thread[2];
13     for (int i = 0; i < 2; ++i) {
14         info[i].values = values; info[i].start = i*512; info[i].end = (i+1)*512;
15         pthread_create(&threads[i], NULL, sum_thread, (void *) &info[i]);
16     }
17     for (int i = 0; i < 2; ++i)
18         pthread_join(threads[i], NULL);
19     return info[0].result + info[1].result;
20 }
21
22
```

program memory (to main stack)



sum example (on heap)

```
1  struct ThreadInfo {
2      pthread_t thread;
3      int *values; int start; int end; int result;
4  };
5
6  void *sum_thread(void *argument) { ...  }
7
8  struct ThreadInfo *start_sum_all(int *values) {
9      struct ThreadInfo *info = calloc(2, sizeof(struct ThreadInfo));
10     for (int i = 0; i < 2; ++i) {
11         info[i].values = values; info[i].start = i*512; info[i].end = (i+1)*512;
12         pthread_create(&info[i].thread, NULL, sum_thread, (void *) &info[i]);
13     }
14     return info;
15 }
16
17 int finish_sum_all(ThreadInfo *info) {
18     for (int i = 0; i < 2; ++i)
19         pthread_join(info[i].thread, NULL);
20     int result = info[0].result + info[1].result;
21     free(info);
22     return result;
23 }
24
```

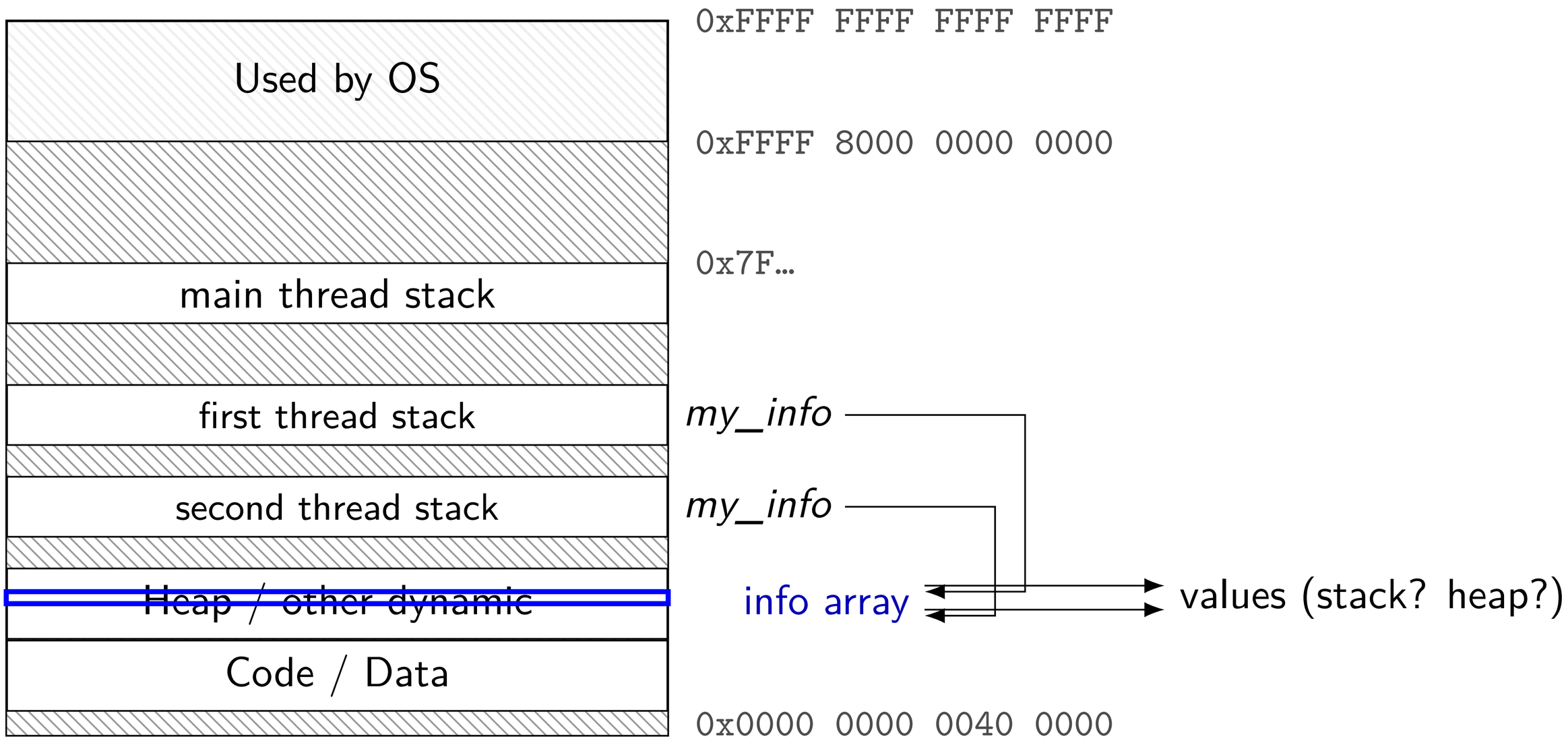
sum example (on heap)

```
1  struct ThreadInfo {
2      pthread_t thread;
3      int *values; int start; int end; int result;
4  };
5
6  void *sum_thread(void *argument) { ...  }
7
8  struct ThreadInfo *start_sum_all(int *values) {
9      struct ThreadInfo *info = calloc(2, sizeof(struct ThreadInfo));
10     for (int i = 0; i < 2; ++i) {
11         info[i].values = values; info[i].start = i*512; info[i].end = (i+1)*512;
12         pthread_create(&info[i].thread, NULL, sum_thread, (void *) &info[i]);
13     }
14     return info;
15 }
16
17 int finish_sum_all(ThreadInfo *info) {
18     for (int i = 0; i < 2; ++i)
19         pthread_join(info[i].thread, NULL);
20     int result = info[0].result + info[1].result;
21     free(info);
22     return result;
23 }
24
```

sum example (on heap)

```
 1  struct ThreadInfo {
 2      pthread_t thread;
 3      int *values; int start; int end; int result;
 4  };
 5
 6  void *sum_thread(void *argument) { ...  }
 7
 8  struct ThreadInfo *start_sum_all(int *values) {
 9      struct ThreadInfo *info = calloc(2, sizeof(struct ThreadInfo));
10      for (int i = 0; i < 2; ++i) {
11          info[i].values = values; info[i].start = i*512; info[i].end = (i+1)*512;
12          pthread_create(&info[i].thread, NULL, sum_thread, (void *) &info[i]);
13      }
14      return info;
15  }
16
17  int finish_sum_all(ThreadInfo *info) {
18      for (int i = 0; i < 2; ++i)
19          pthread_join(info[i].thread, NULL);
20      int result = info[0].result + info[1].result;
21      free(info);
22      return result;
23  }
24
```

thread_sum memory (heap version)

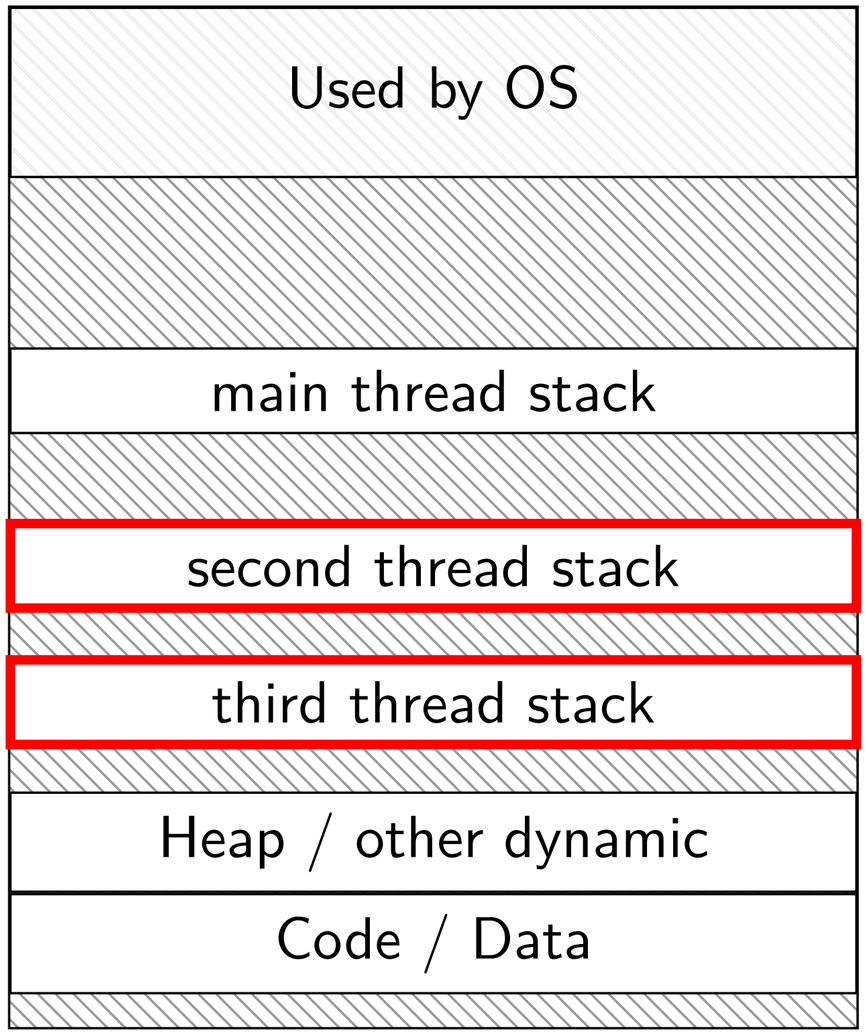


what's wrong with this?

```
1  /* omitted: headers */
2  void *create_string(void *ignored_argument) {
3    char string[1024];
4    ComputeString(string);
5    return string;
6  }
7
8  int main() {
9    pthread_t the_thread;
10   pthread_create(&the_thread, NULL, create_string, NULL);
11   char *string_ptr;
12   pthread_join(the_thread, (void**) &string_ptr);
13   printf("string is %s\n", string_ptr);
14 }
```

program memory

program memory



0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

dynamically allocated stacks

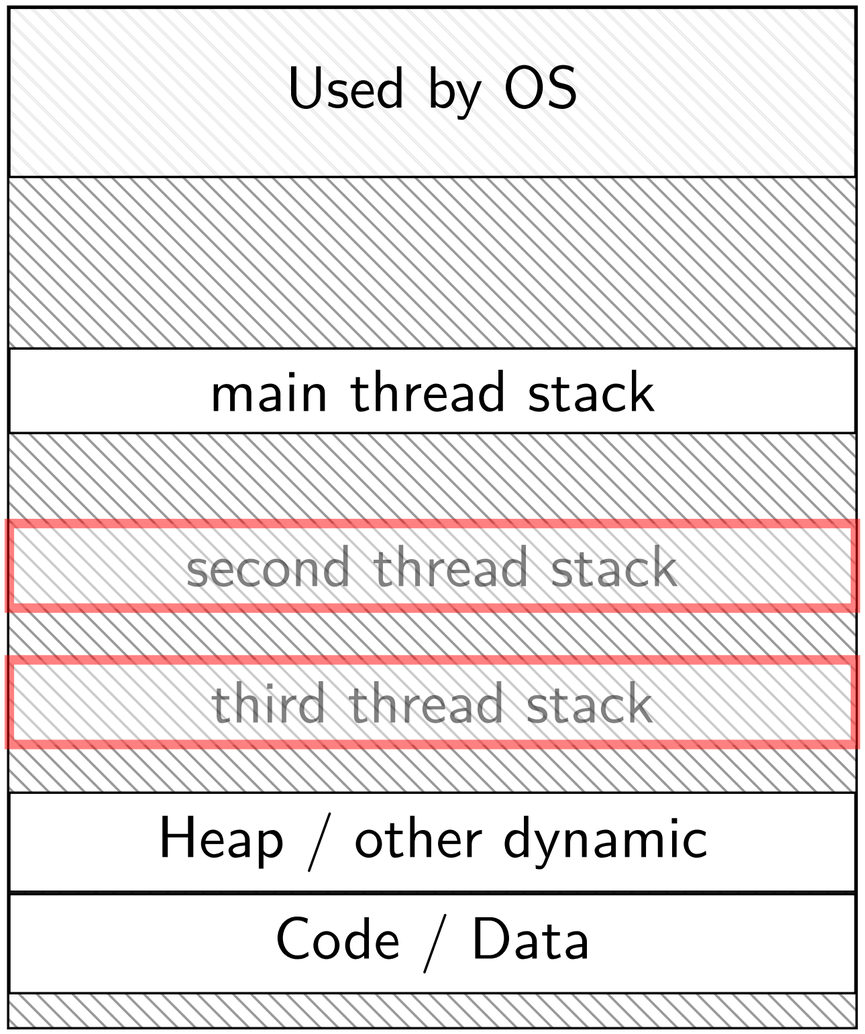
`char string[]` allocated here

`string_ptr` pointed to here

...stacks deallocated when threads exit/are joined

0x0000 0000 0040 0000

program memory



0xFFFF FFFF FFFF FFFF

0xFFFF 8000 0000 0000

0x7F...

dynamically allocated stacks
`char string[]` allocated here
`string_ptr` pointed to here

...stacks deallocated when threads exit/are joined

0x0000 0000 0040 0000

thread joining

pthread_join allows collecting thread return value

if you don’t join joinable thread, then *memory leak*!

avoiding memory leak?

always join… or

“detach” thread to make it not joinable

pthread_detach

```
1  void *show_progress(void * ...) { ... }
2  void spawn_show_progress_thread() {
3      pthread_t show_progress_thread;
4      pthread_create(&show_progress_thread, NULL,
5                     show_progress, NULL);
6
7      /* instead of keeping pthread_t around to join thread later: */
8      pthread_detach(show_progress_thread);]
9  }
10
11 int main() {
12     spawn_show_progress_thread();
13     do_other_stuff();
14     ...
15 }
```

detach = don’t care about return value, etc.
system will decallocate when thread terminates

pthread_detach

```
1  void *show_progress(void * ...) { ... }
2  void spawn_show_progress_thread() {
3      pthread_t show_progress_thread;
4      pthread_create(&show_progress_thread, NULL,
5                     show_progress, NULL);
6
7      /* instead of keeping pthread_t around to join thread later: */
8      pthread_detach(show_progress_thread);]
9  }
10
11 int main() {
12     spawn_show_progress_thread();
13     do_other_stuff();
14     ...
15 }
```

detach = don’t care about return value, etc.
system will decallocate when thread terminates

starting threads detached

```
1  void *show_progress(void * ...) { ... }
2  void spawn_show_progress_thread() {
3      pthread_t show_progress_thread;
4      pthread_attr_t attrs;
5      pthread_attr_init(&attrs);
6      pthread_attr_setdetachstate(&attrs, PTHREAD_CREATE_DETACHED);
7      pthread_create(&show_progress_thread, attrs,
8                     show_progress, NULL);
9      pthread_attr_destroy(&attrs);
10 }
```

starting threads detached

```
1  void *show_progress(void * ...) { ... }
2  void spawn_show_progress_thread() {
3      pthread_t show_progress_thread;
4      pthread_attr_t attrs;
5      pthread_attr_init(&attrs);
6      pthread_attr_setdetachstate(&attrs, PTHREAD_CREATE_DETACHED);
7      pthread_create(&show_progress_thread, attrs,
8                     show_progress, NULL);
9      pthread_attr_destroy(&attrs);
10 }
```

setting stack sizes

```
1 void *show_progress(void * ...) { ... }
2 void spawn_show_progress_thread() {
3     pthread_t show_progress_thread;
4     pthread_attr_t attrs;
5     pthread_attr_init(&attrs);
6     pthread_attr_setstacksize(&attrs, 32 * 1024 /* bytes */);
7     pthread_create(&show_progress_thread, attrs,
8                    show_progress, NULL);
9 }
```