# Stack Pointer

- void *stack = new char[1024*1024];
- TCB.rsp = high address of stack allocated

# How to save/restore context – 1

- This assignment is about coroutine – everything still runs in a big program [not like pthread]
  - We do create many kernel-level threads by pthread_thread and the threads can run concurrently on possible multiple COREs.
  - In this assignment, the threads run as corountine on one core only.
- The returned address on the stack is the actual place to save the context for RIP/Program Count for the threads.

# How to save/restore context – 2

- Main thread creates a new thread – thread 1
- Thread 1 function runs
- Some steps are done.
- Thread 1 calls Thd_yield()
- Inside Thd_yield(), we will do
  - Context saving
  - Scheduler()
  - Constext restoring

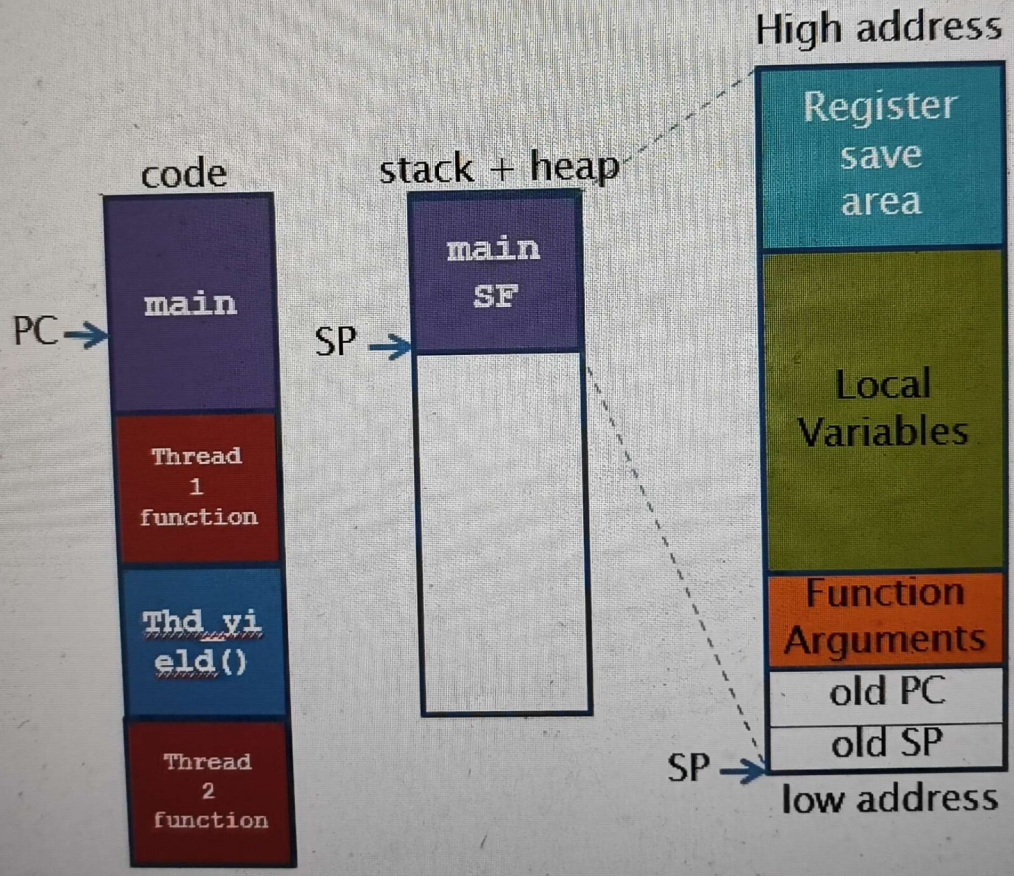
# How to save/restore context – 3

- During context saving, need to save the registers into TCB, except RIP/PC
- Following this, we need to schedule another thread to run.
- Suppose no thread is in Ready Queue.
- Now we only run the newly created thread (Thread 2 function) in this case.
- In this case, we need to prepare the stack and the context for thread 2.

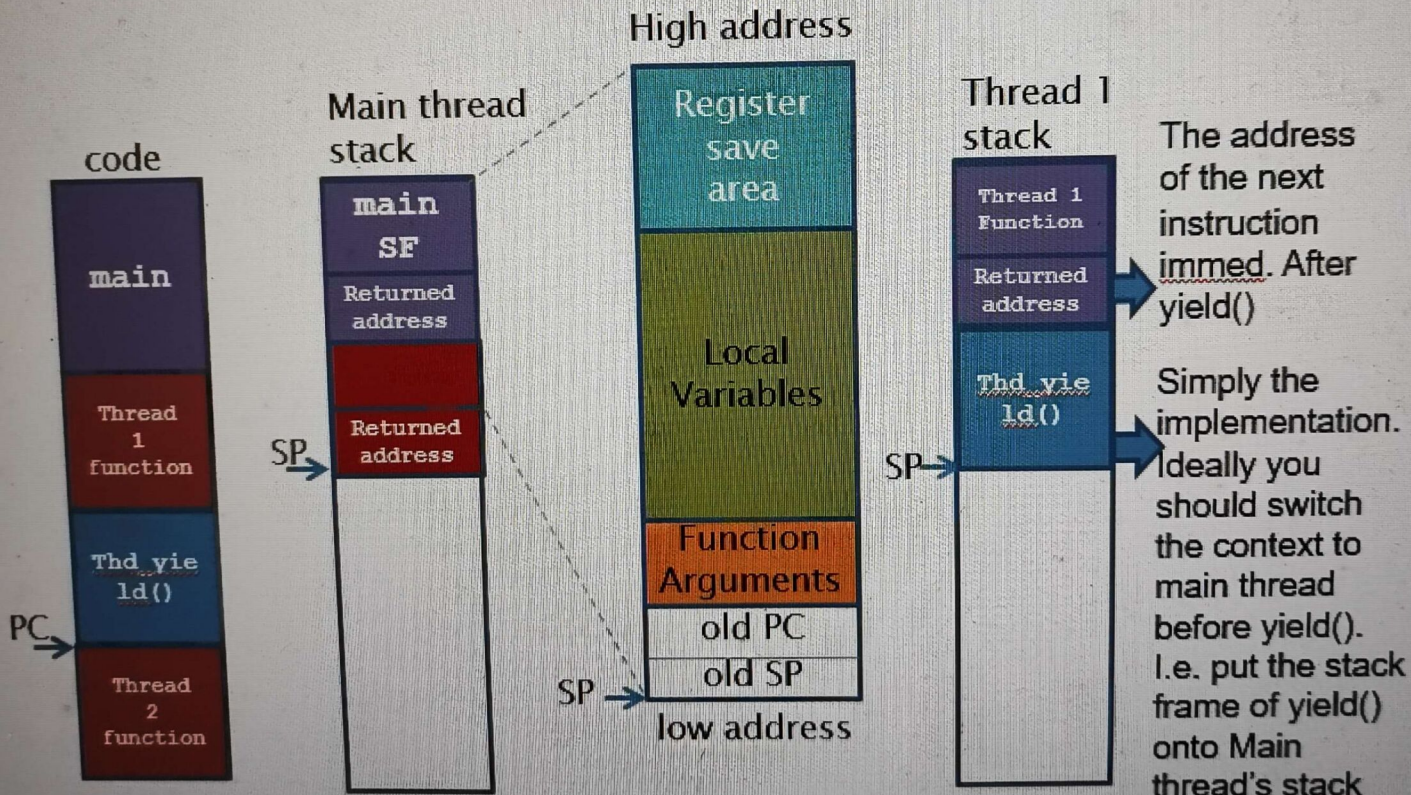
# How to save/restore context – 4

- Then we can run thread 2 function with the arguments.
- Suppose thread 2 finishes. Thread_exit() will terminate thread 2.
  - Will call scheduler() to schedule thread 1 again
- In this case, only thread 1 will run.
  - Context restoring for thread 1 from TCB
    - RSP is restored.
    - No more thread 2 stack is used.
  - The returned address is eventually popped up from the stack of thread 1

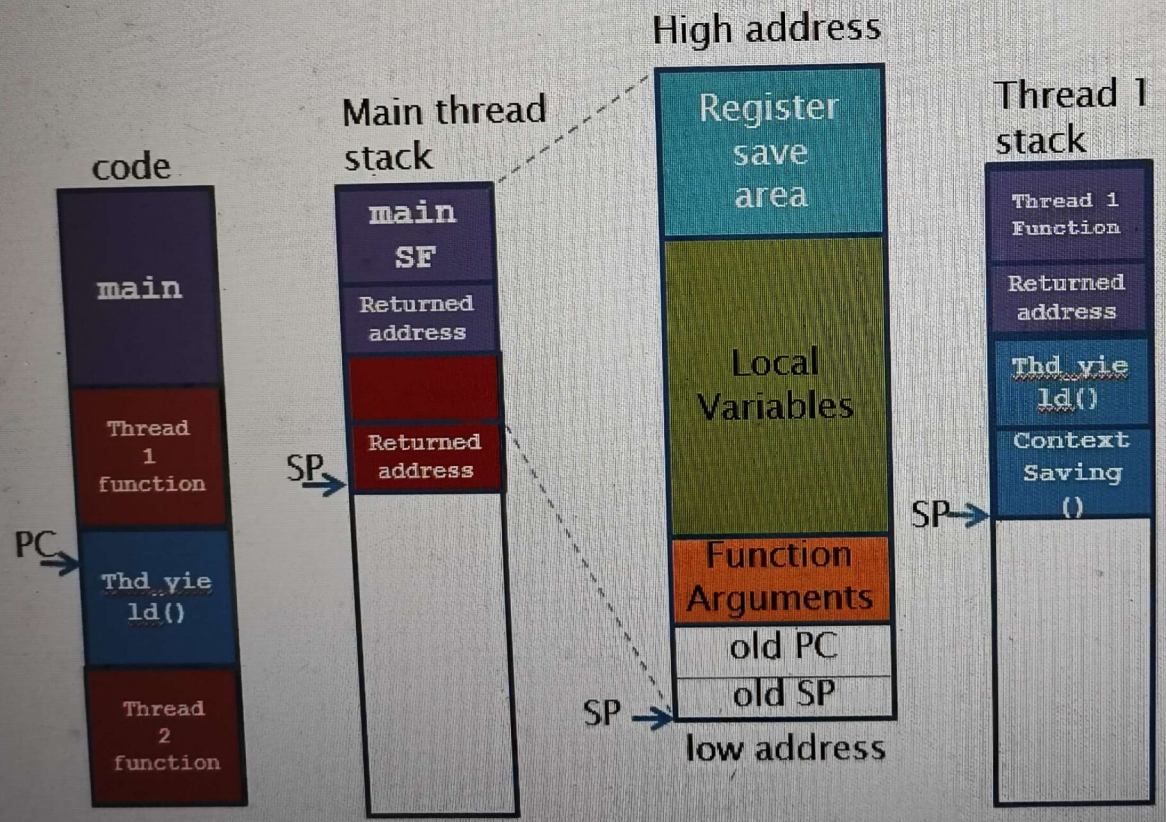
# Stack Frame – 1



# Stack Frame – 3



# Stack Frame – 4



# Stack Frame – 5