MICROS-1

Timothy Borunov

Ph.D. Student, Computer Engineering

With materials from slides by Dr. Sasan Golchin

MICROS-1

- In MEMOS-2 we got to know which regions of memory are usable and how big they are
- This information is important if you are building something that requires a lot of code and/or data
- But before doing anything in the kernel we need to setup the GDT

Global Descriptor Table

- GRUB sets up a default GDT, but we cannot rely on that, so we set up our own
- GDT defines specific segments of memory for specific purposes
- In operating systems, this is used to establish user-kernel separation
- For now, only the kernel segments are sufficient

Global Descriptor Table

Segment Descriptor

63 56	55 52	51 48	47 40	39 32
Base 31 24	**Flags** 3 0	**Limit** 19 16	**Access Byte** 7 0	**Base** 23 16

31 16	15 0
Base 15 0	**Limit** 15 0

- Base: A 32-bit value indicating the linear address where the segment begins
- Limit: A 20-bit value indicating size of the segment with a granularity specified by the flags
- Flags: Granularity (Bit 55), CodeSize (Bit 54) and Reserved (Bits 52 to 53)

Global Descriptor Table

- Has at least 3 entries: NULL descriptor, kernel code segment, and kernel data segment
- Defining GDT can be done in C or ASM; but you will need ASM to load it (LGDT)
- After setting up the GDT, you need to reload all the segment registers to point to the GDT entry
- Neither POP nor MOV can place a value in the code-segment register CS; only the far control transfer (JMP) instructions can change CS

MICROS-1

- In MEMOS-2 we got to know which regions of memory are usable and how big they are
- This information is important if you are building something that requires a lot of code and/or data
- But before doing anything in the kernel we need to setup the GDT
- Detect memory to see which regions are free (already done in MEMOS-2)
- Finally, assign work to the threads and call the scheduler to run the first task

Task Control Block

- A thread is a function with a private stack, characterized by the TCB
- TCB stores
  - State: New, Ready, Active, Dead, etc.
  - Next Instruction to run: EIP
  - Stack top: ESP
  - Machine State (minimally the following): General registers: EAX, EBX, ECX, EDX, ESI, EDI, EBP (pushl/popl, pushal/popal) and Flags: EFLAGS (pushf/popf)

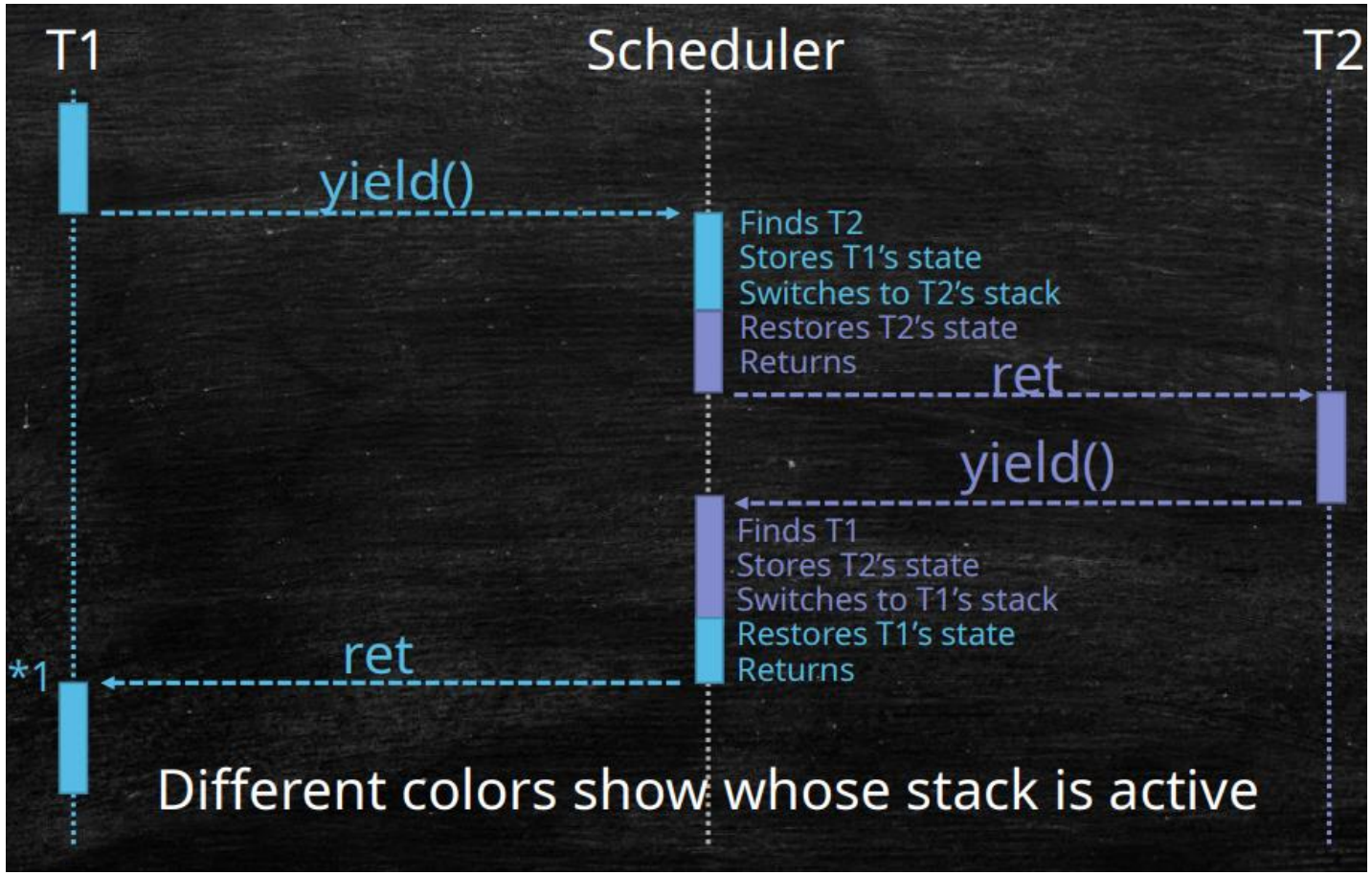
Scheduler Functionality

- Thread creation: we have a predefined pool of threads, so the only thing to do is assign work to a free thread
  - Note that the functions assigned to thread can be as simple as an empty while loop running for a predetermined number of iterations
  - Thus, the stack size need not be very big
- Delete threads when they complete assigned work: this only involves changing state of the thread to free as we do not want to delete threads
- Support task switches

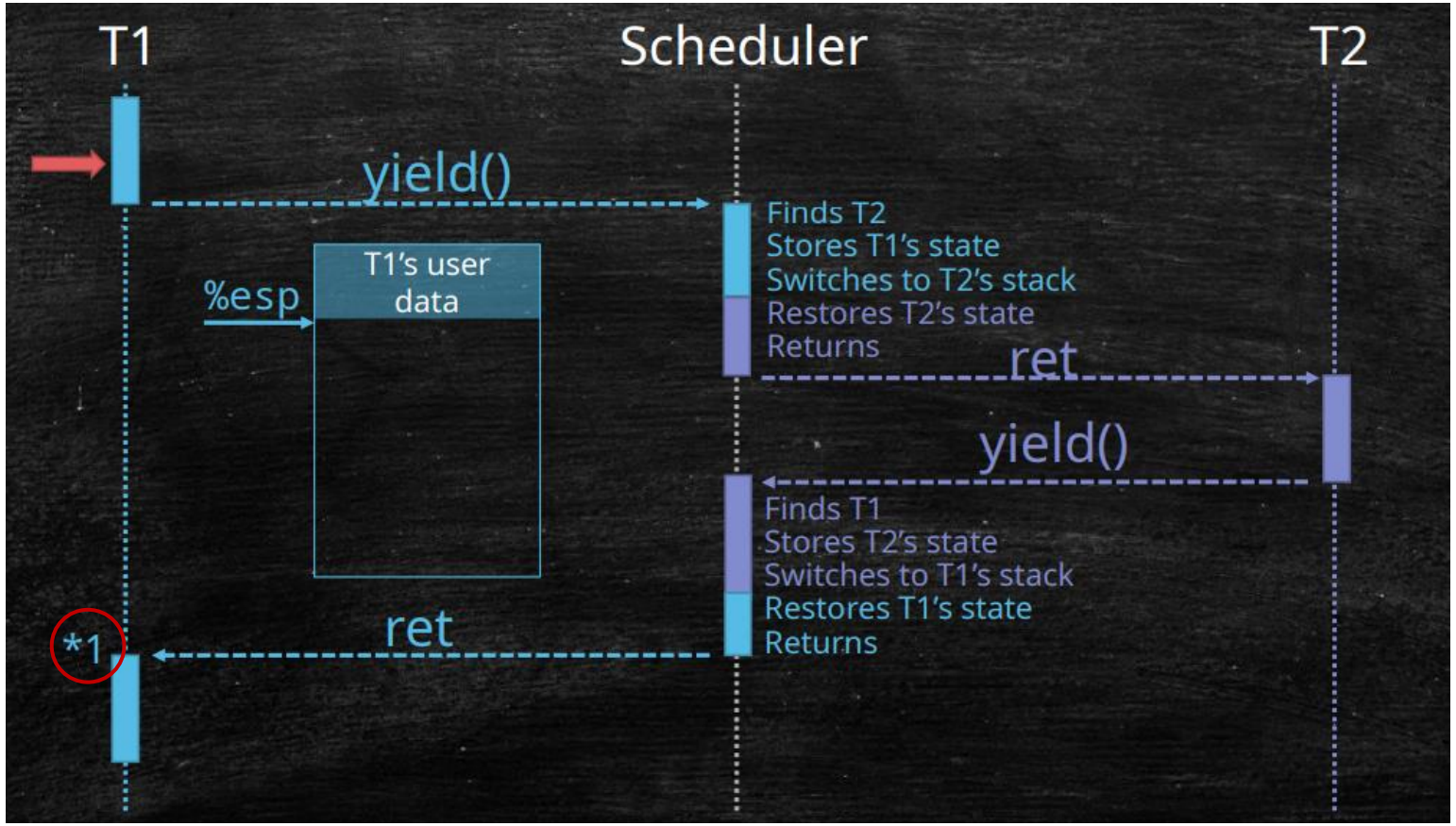
Non-Preemptive Context Switch

- A context switch happens when the current running task either finishes execution or explicitly yields
- Control passes to the scheduler which
  - Chooses the next task (FIFO ordering)
  - Pushes the machine state on the stack
  - Updates the TCB of the current task (ESP, EIP, State)
  - Switches to the stack of the next thread (mov next->esp, %esp)
  - Pops the machine state from the new stack
  - Returns to the new current task

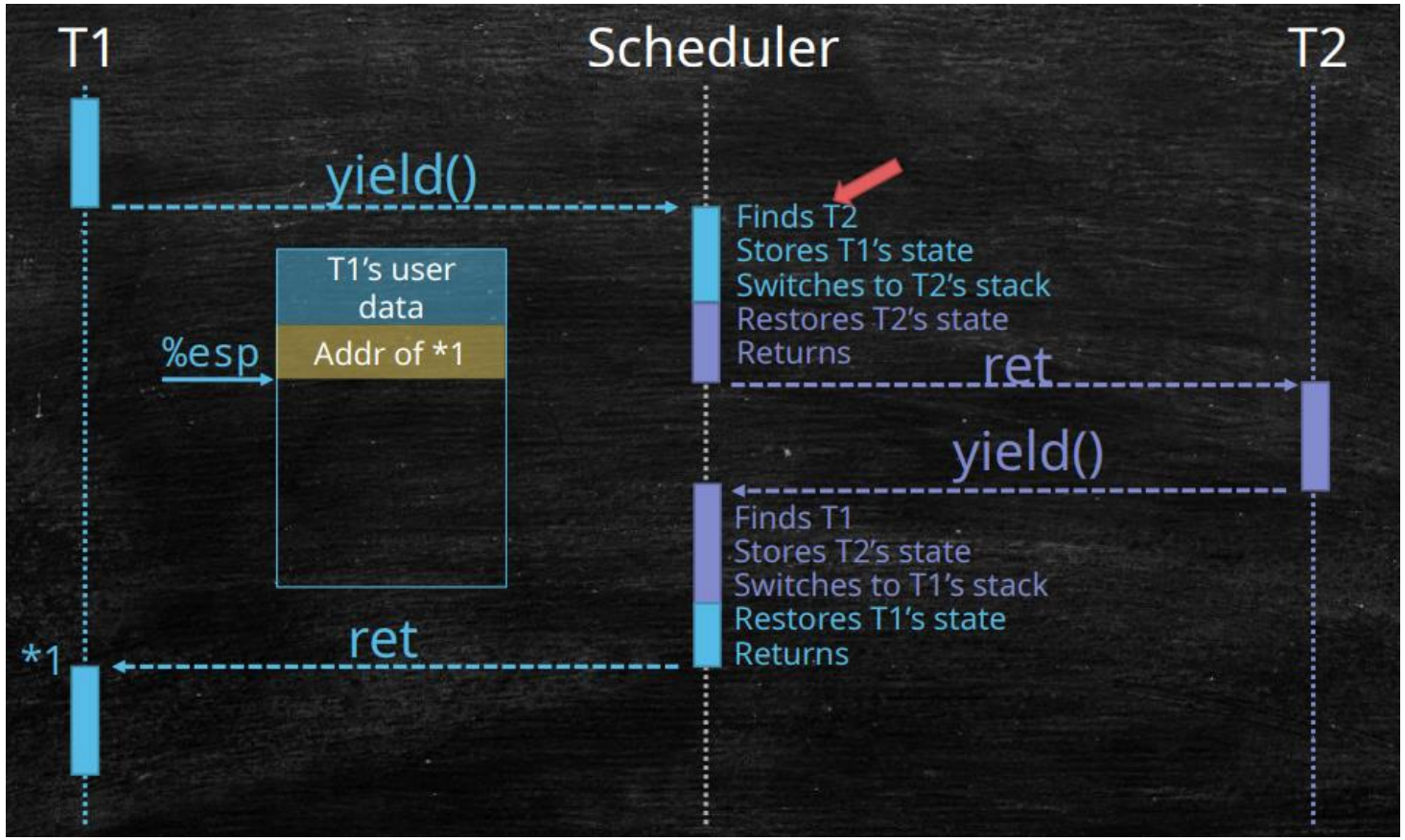
Example (T1 to T2 and back to T1)



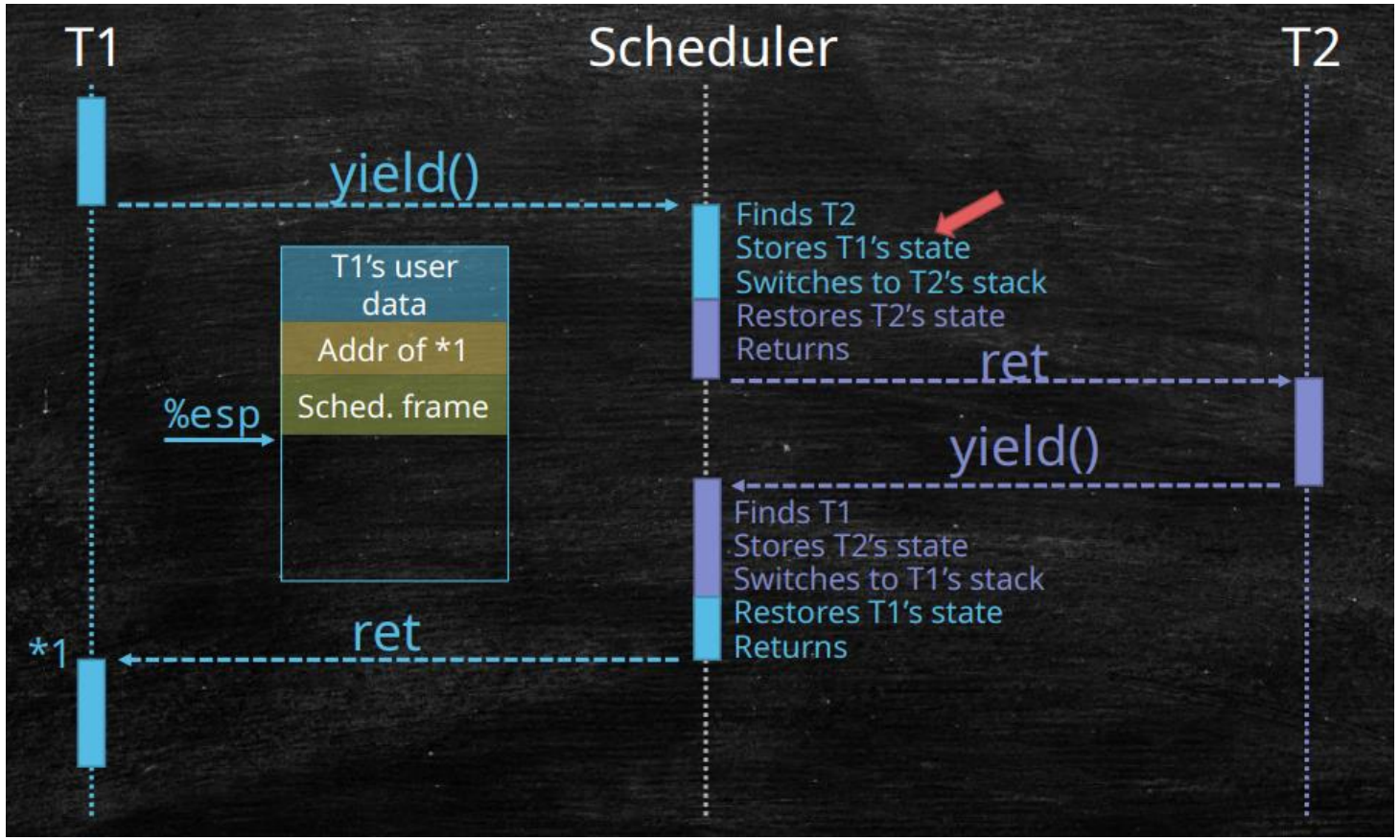
Example (T1 to T2 and back to T1)



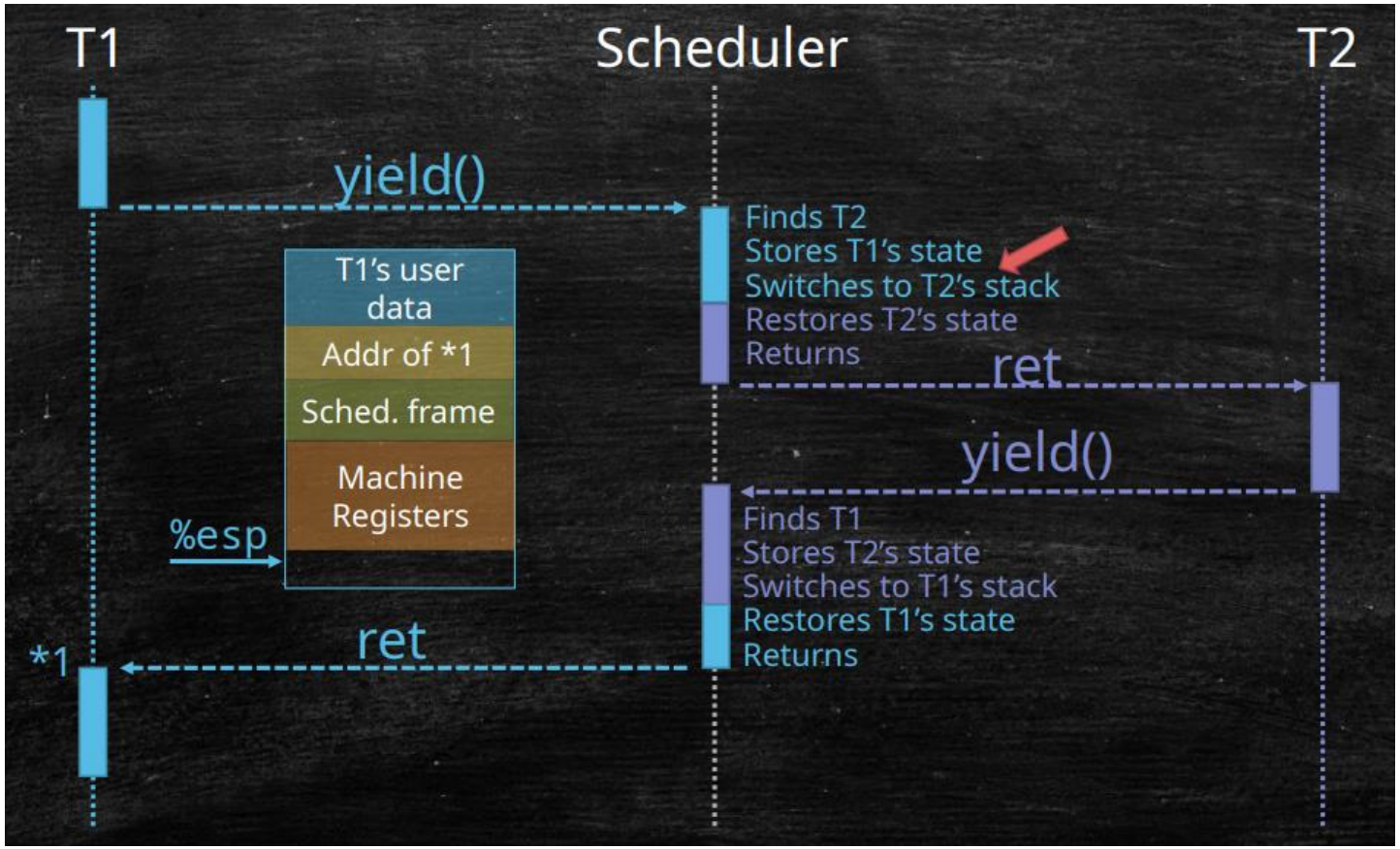
Example (T1 to T2 and back to T1)



Example (T1 to T2 and back to T1)

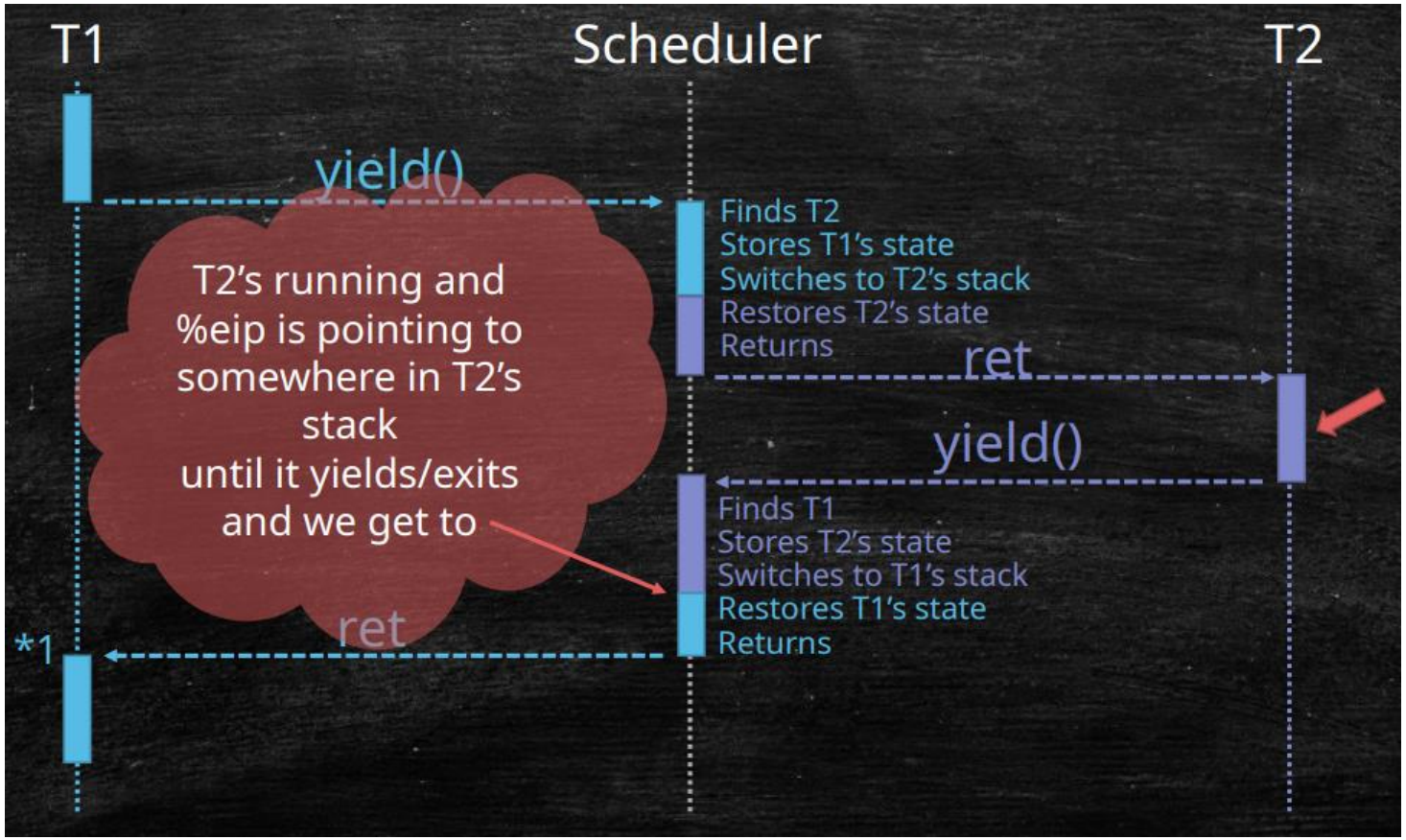


Example (T1 to T2 and back to T1)



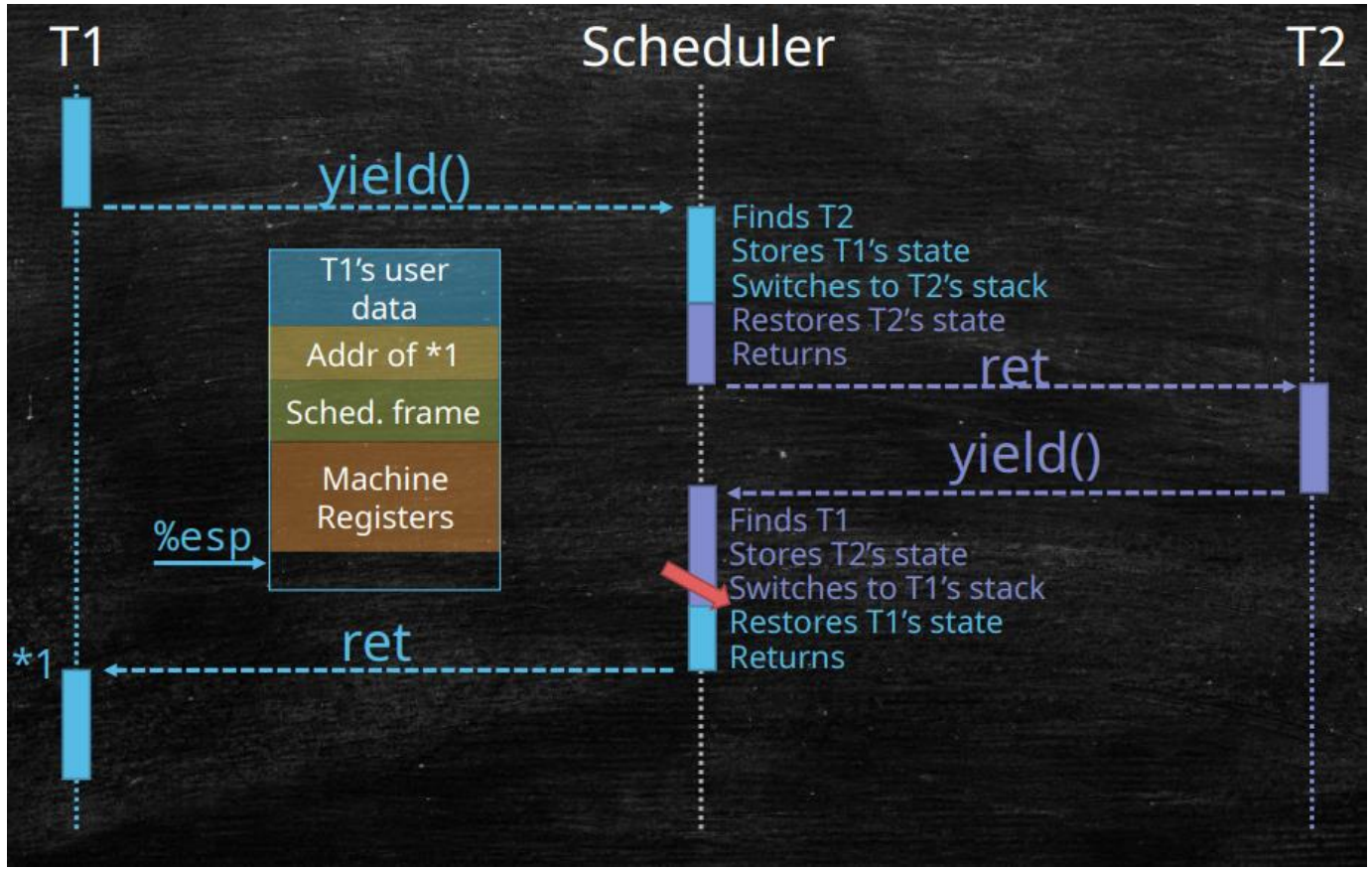
BOSTON UNIVERSITY

Example (T1 to T2 and back to T1)

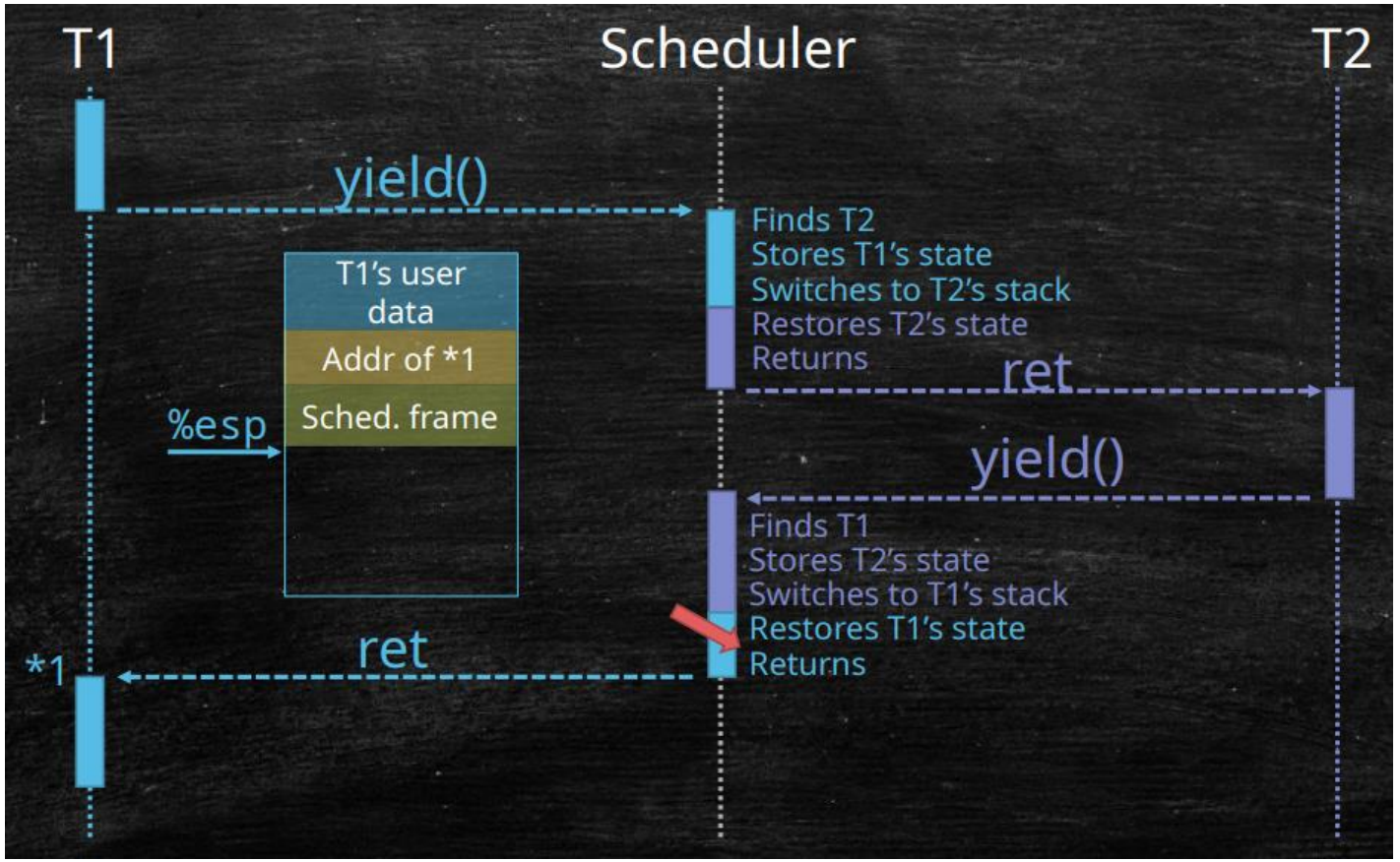


BOSTON UNIVERSITY

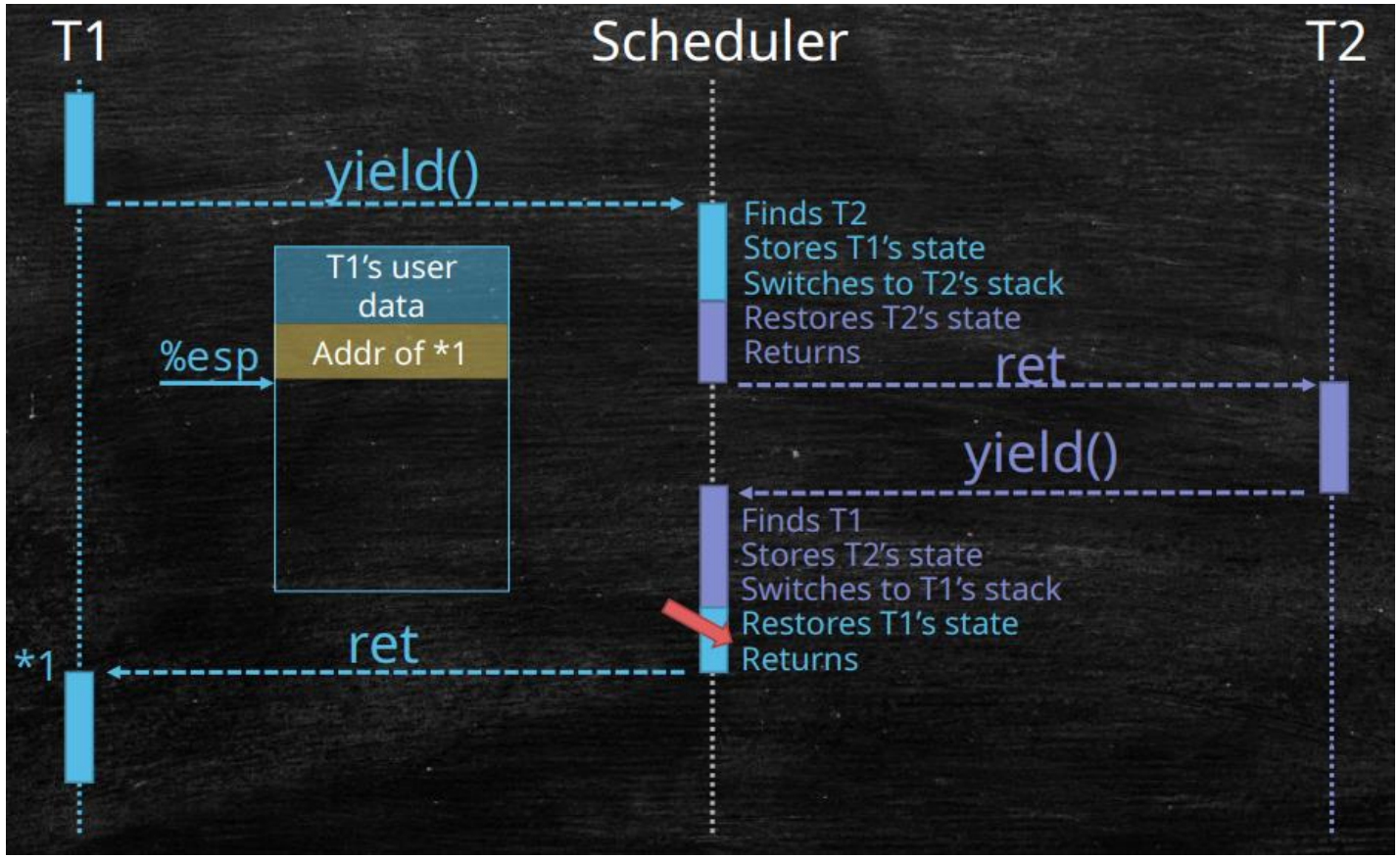
Example (T1 to T2 and back to T1)



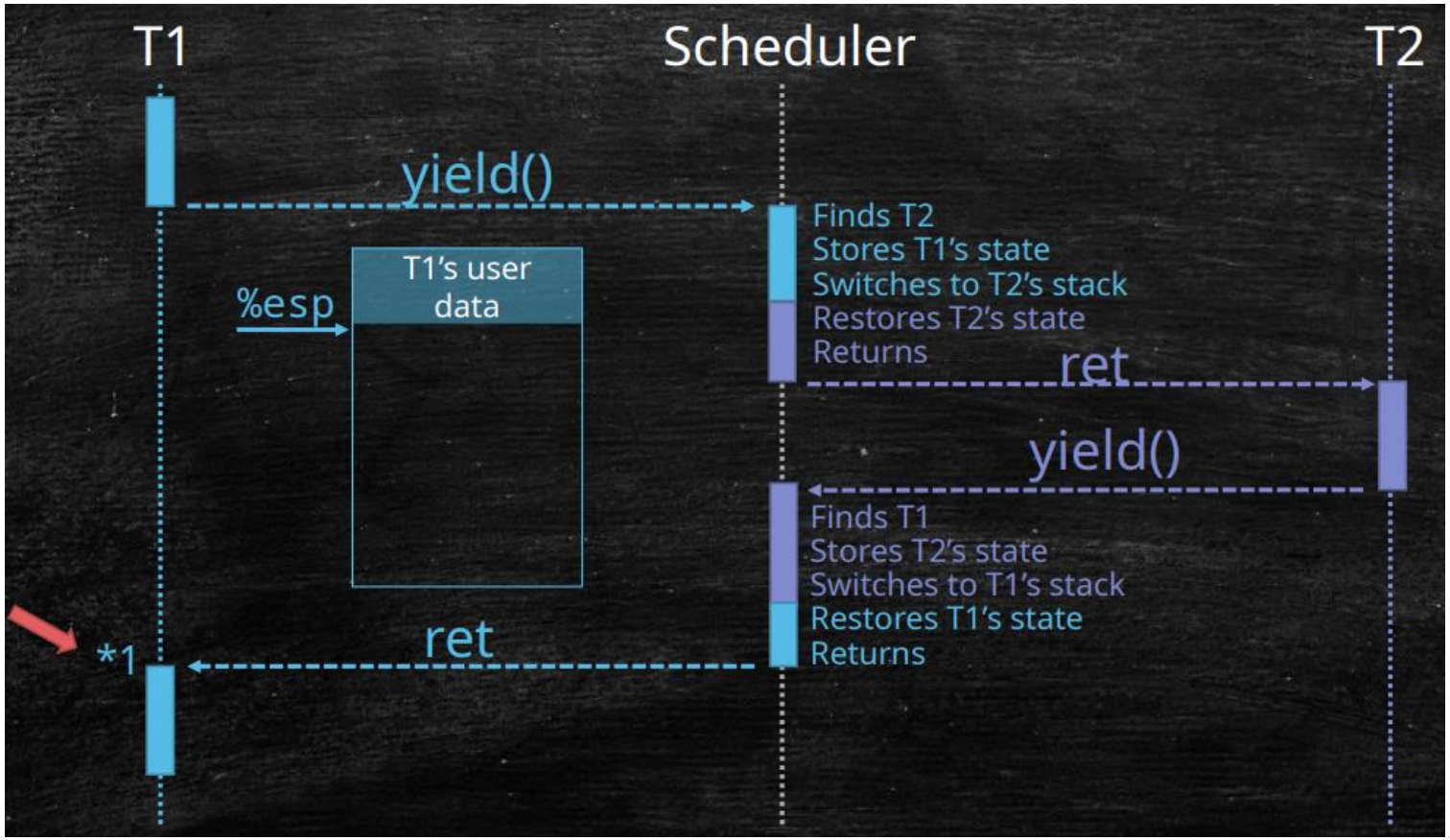
Example (T1 to T2 and back to T1)



Example (T1 to T2 and back to T1)



Example (T1 to T2 and back to T1)



References

1. GDT - Overview, Tutorial, Packed structs
2. Inline Assembly - OSDev, gcc
3. Felix Cloutier - LGDT/LIDT, PUSHA