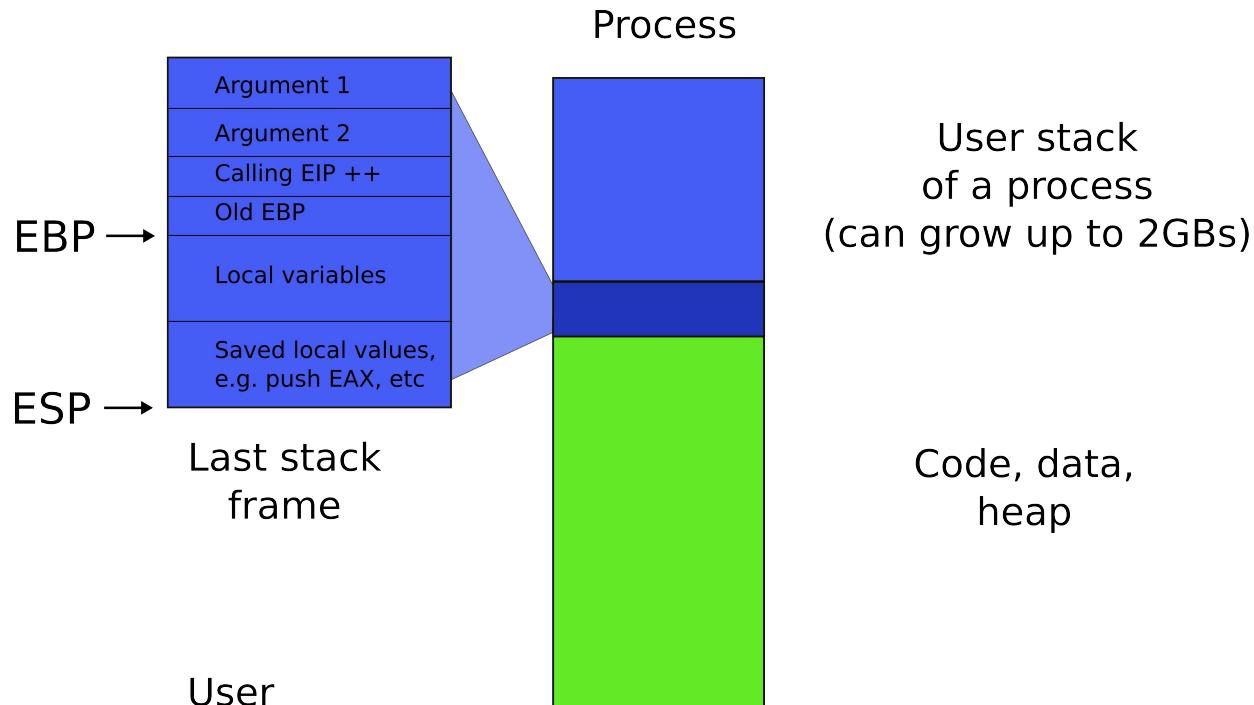


ICS143A: Principles of Operating Systems

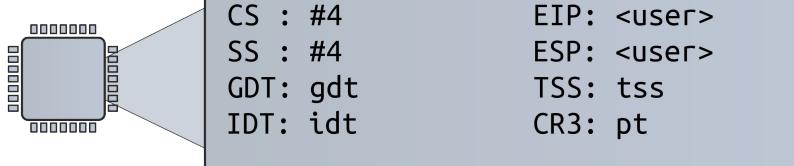
Lecture 13: Context switching

Anton Burtsev
November, 2017

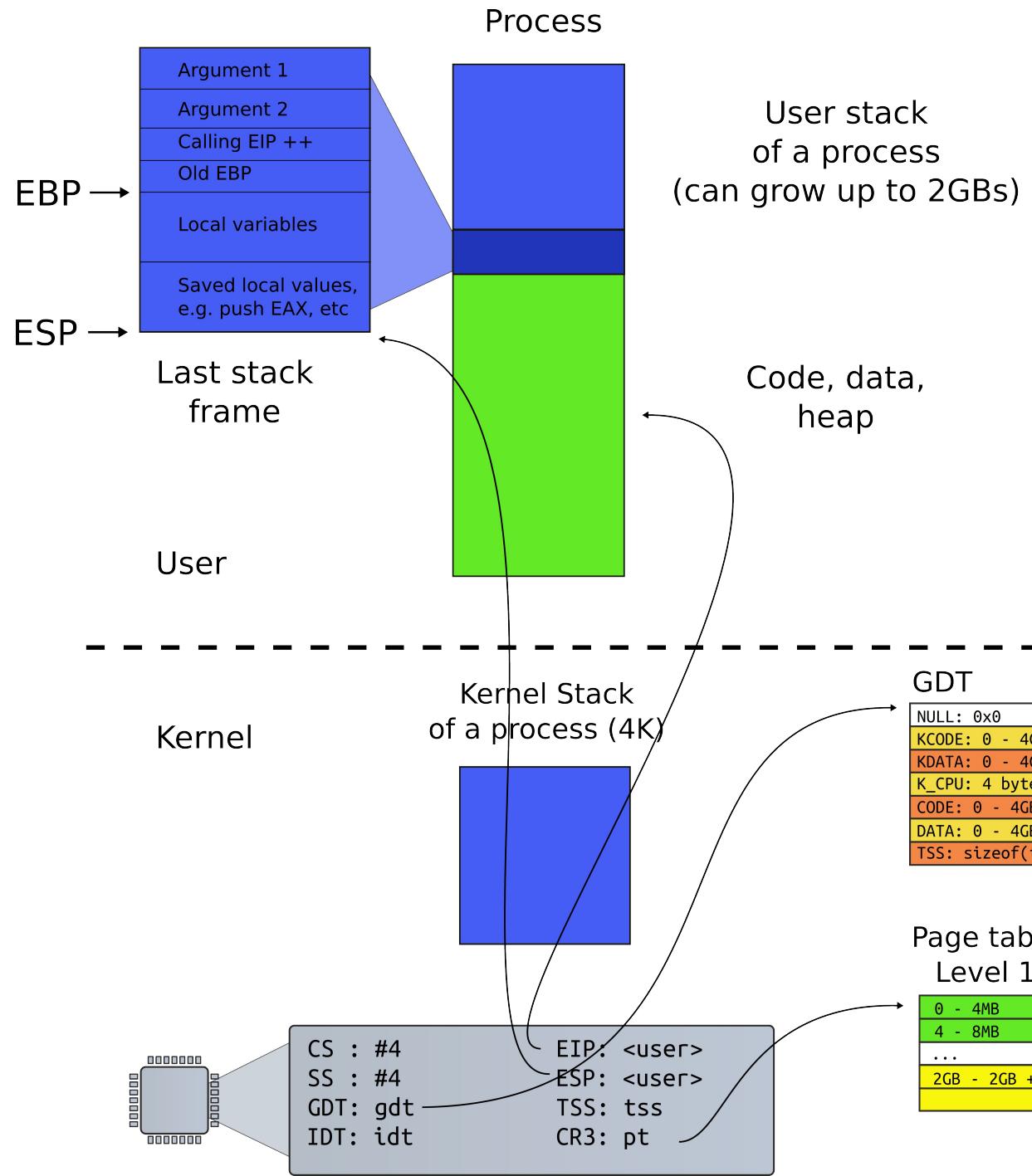


Kernel

Kernel Stack
of a process (4K)

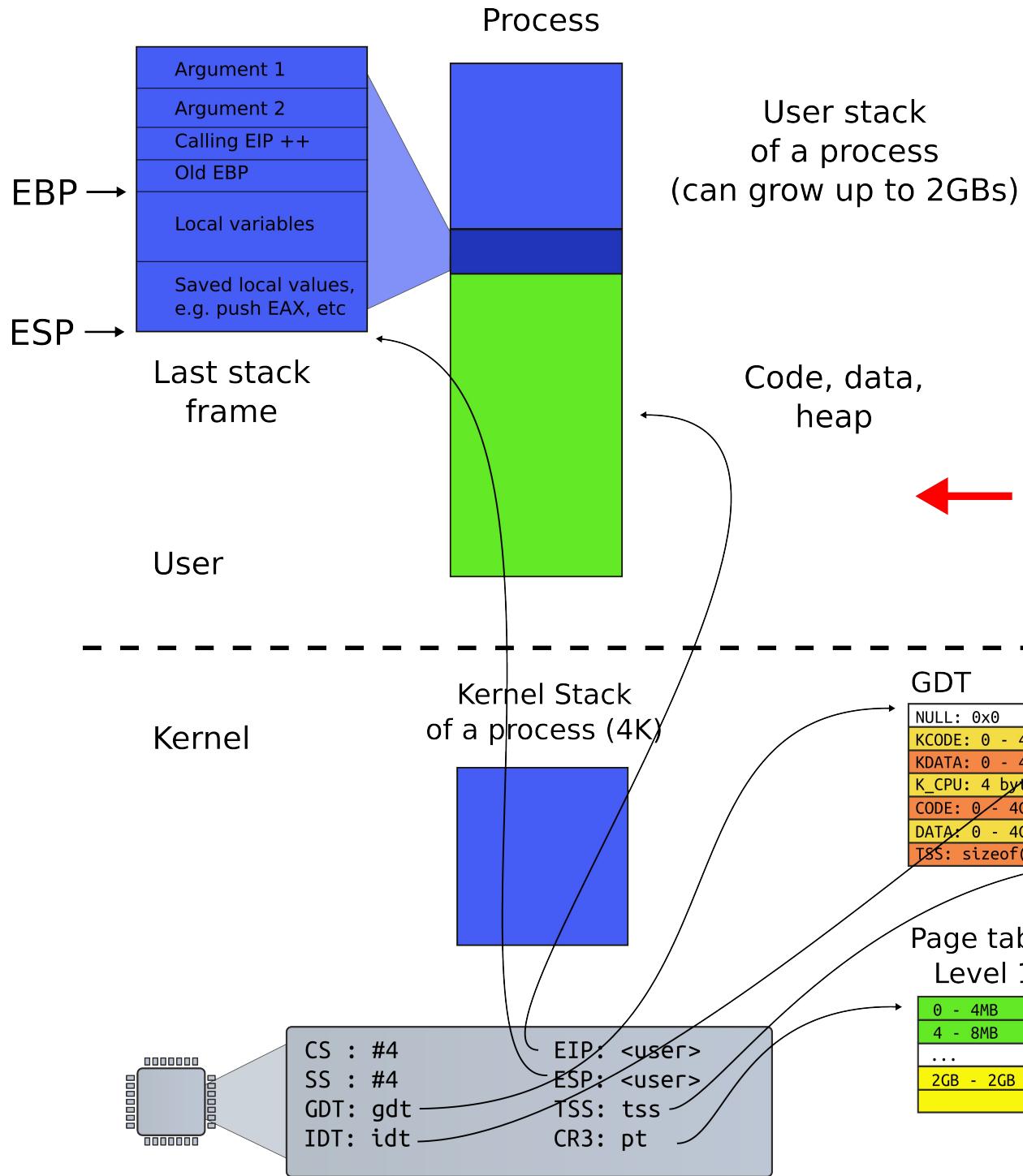


- User mode
- Two stacks
 - Kernel and user
 - Kernel stack is empty

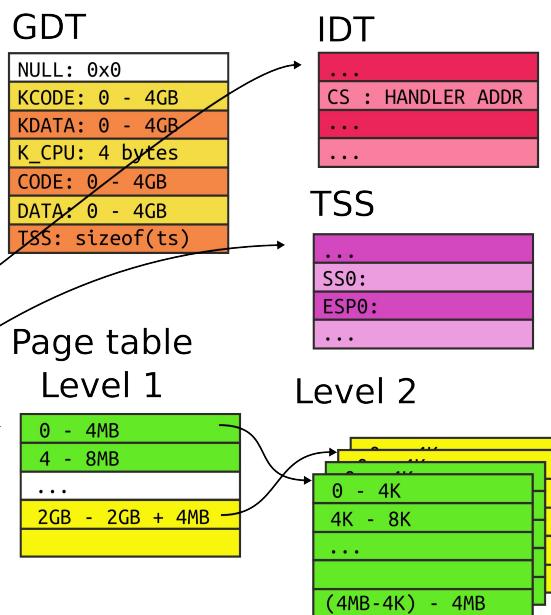


- Page table
- GDT

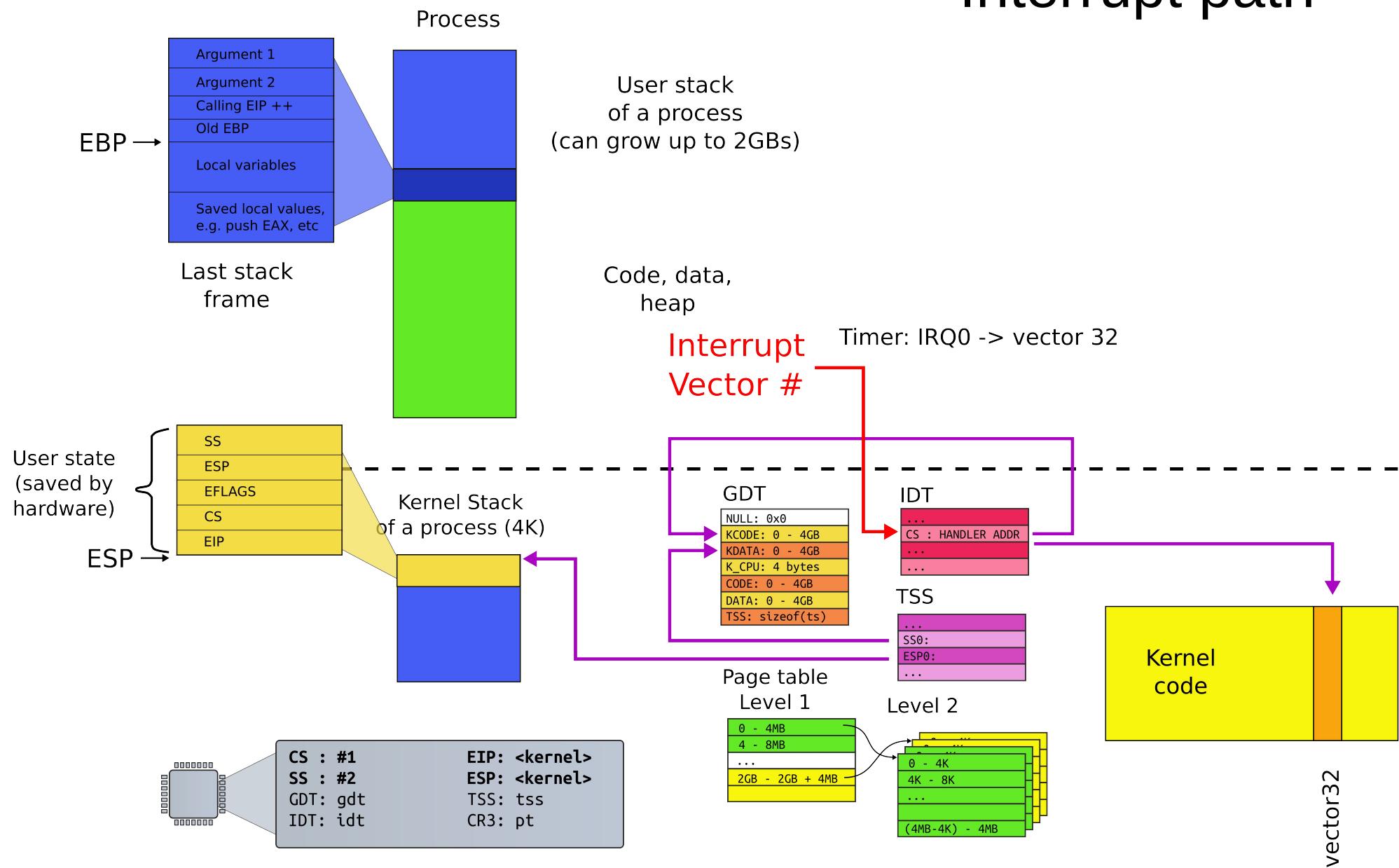
Timer interrupt



Timer
Interrupt



Interrupt path



Where does IDT (entry 32) point to?

vector32:

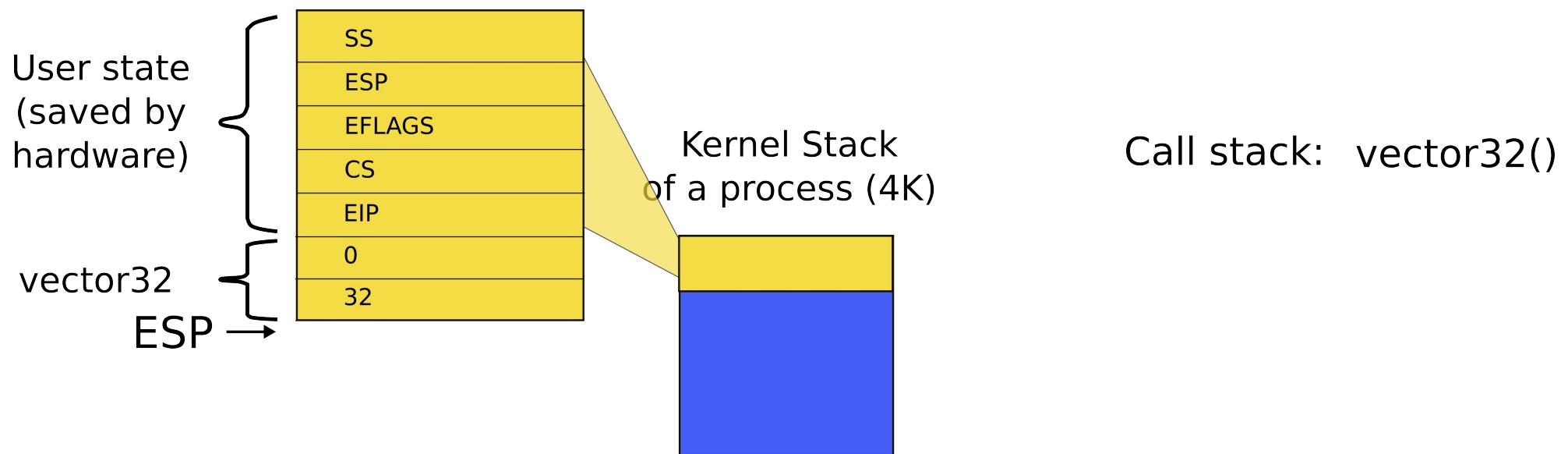
```
pushl $0      // error code
```

```
pushl $32      // vector #
```

```
jmp alltraps
```

- Automatically generated
- From vectors.pl
 - vector.S

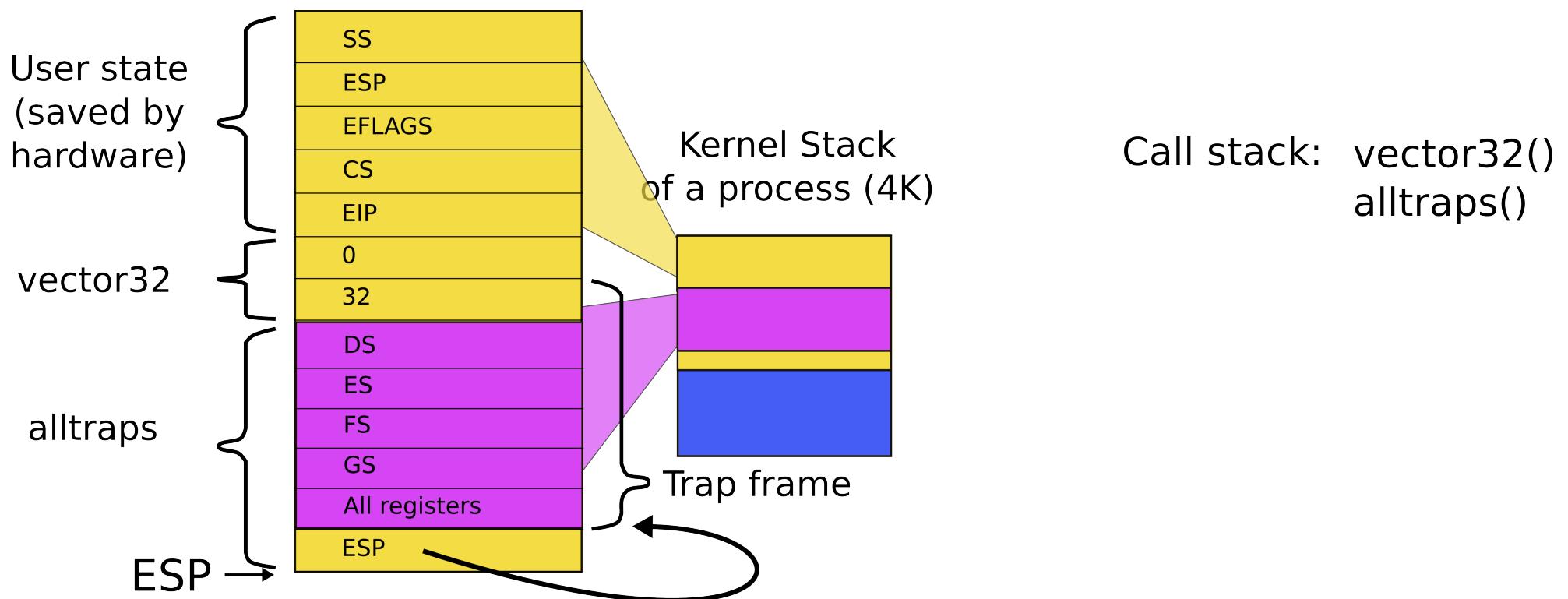
Kernel stack after interrupt



alltraps()

```
3254 alltraps:  
3255     # Build trap frame.  
3256     pushl %ds  
3257     pushl %es  
3258     pushl %fs  
3259     pushl %gs  
3260     pushal  
3261  
3262     # Set up data and per-cpu segments.  
3263     movw $(SEG_KDATA<<3), %ax  
3264     movw %ax, %ds  
3265     movw %ax, %es  
3266     movw $(SEG_KCPU<<3), %ax  
3267     movw %ax, %fs  
3268     movw %ax, %gs  
3269  
3270     # Call trap(tf), where tf=%esp  
3271     pushl %esp  
3272     call trap
```

Kernel stack after interrupt



```
3254 alltraps:  
3255 # Build trap frame.  
3256 pushl %ds  
3257 pushl %es  
3258 pushl %fs  
3259 pushl %gs  
3260 pushal  
3261  
3262 # Set up data and per-cpu segments.  
3263 movw $(SEG_KDATA<<3), %ax  
3264 movw %ax, %ds  
3265 movw %ax, %es  
3266 movw $(SEG_KCPU<<3), %ax  
3267 movw %ax, %fs  
3268 movw %ax, %gs  
3269  
3270 # Call trap(tf), where tf=%esp  
3271 pushl %esp  
3272 call trap
```

alltraps()

```
3351 trap(struct trapframe *tf)
3352 {
...
3363     switch(tf->trapno){
3364     case T_IRQ0 + IRQ_TIMER:
3365         if(cpu->id == 0){
3366             acquire(&tickslock);
3367             ticks++;
3368             wakeup(&ticks);
3369             release(&tickslock);
3370         }
3372     break;
...
3423     if(proc && proc->state == RUNNING
3424         && tf->trapno == T_IRQ0+IRQ_TIMER)
3424         yield();
```

trap()

```
3351 trap(struct trapframe *tf)
3352 {
...
3363     switch(tf->trapno){
3364     case T_IRQ0 + IRQ_TIMER:
3365         if(cpu->id == 0){
3366             acquire(&tickslock);
3367             ticks++;
3368             wakeup(&ticks);
3369             release(&tickslock);
3370         }
3372     break;
...
3423     if(proc && proc->state == RUNNING
3424         && tf->trapno == T_IRQ0+IRQ_TIMER)
3424         yield();
```

trap()

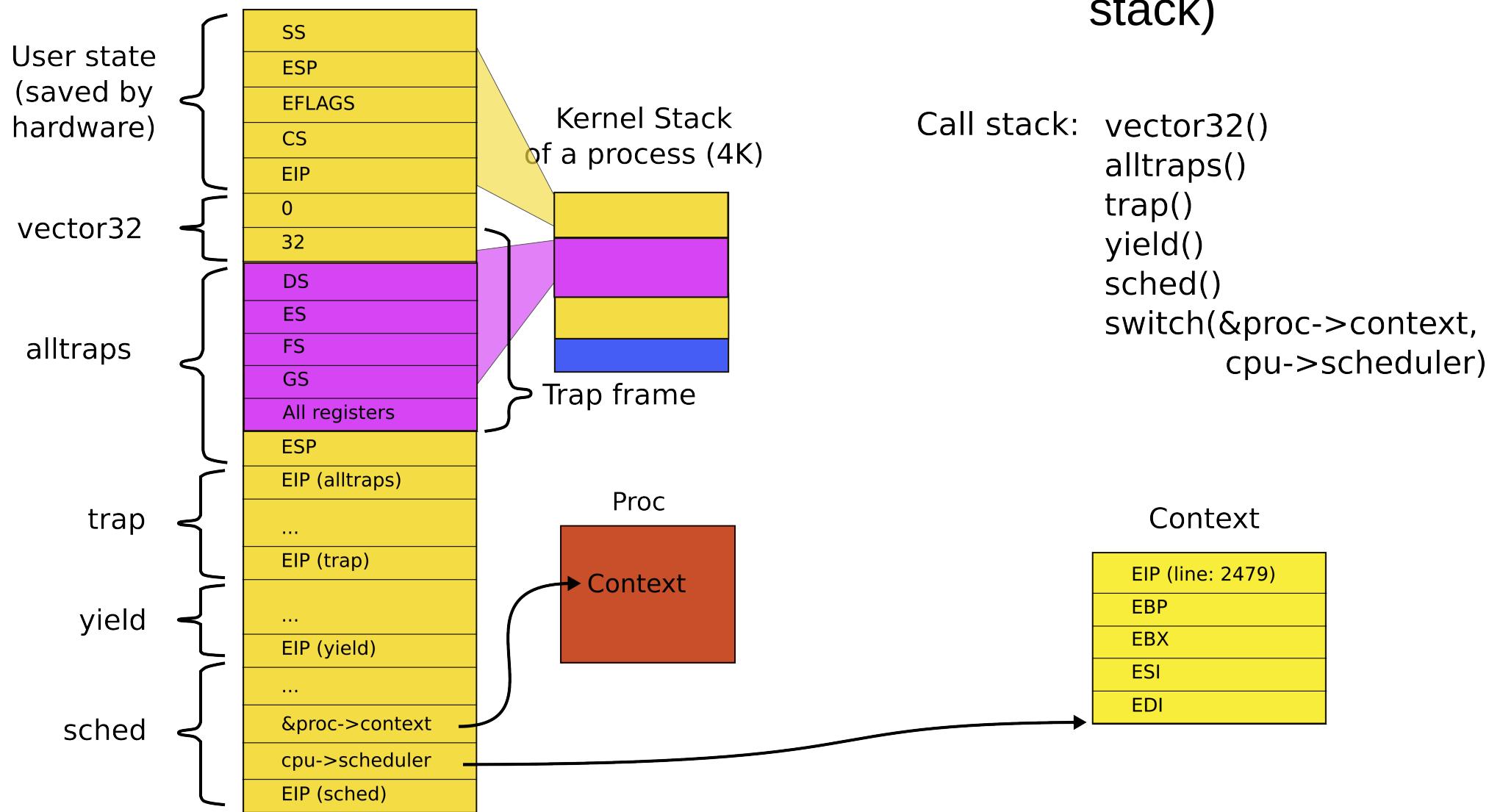
Invoke the scheduler

```
2777 yield(void)  
2778 {  
2779     acquire(&ptable.lock);  
2780     proc->state = RUNNABLE;  
2781     sched();  
2782     release(&ptable.lock);  
2783 }
```

Start the context switch

```
2758 sched(void)
2759 {
...
2771     swtch(&proc->context,
...
2773 }
```

Stack inside swtch() and its two arguments (passed on the stack)



```
2958 swtch:  
2959    movl 4(%esp), %eax # **old  
2960    movl 8(%esp), %edx # *new  
2961  
2962 # Save old callee-save registers  
2963    pushl %ebp  
2964    pushl %ebx  
2965    pushl %esi  
2966    pushl %edi  
2967  
2968 # Switch stacksh  
2969    movl %esp, (%eax) # *old = %esp  
2970    movl %edx, %esp # %esp = new  
2971  
2972 # Load new callee-save registers  
2973    popl %edi  
2974    popl %esi  
2975    popl %ebx  
2976    popl %ebp  
2977    ret
```

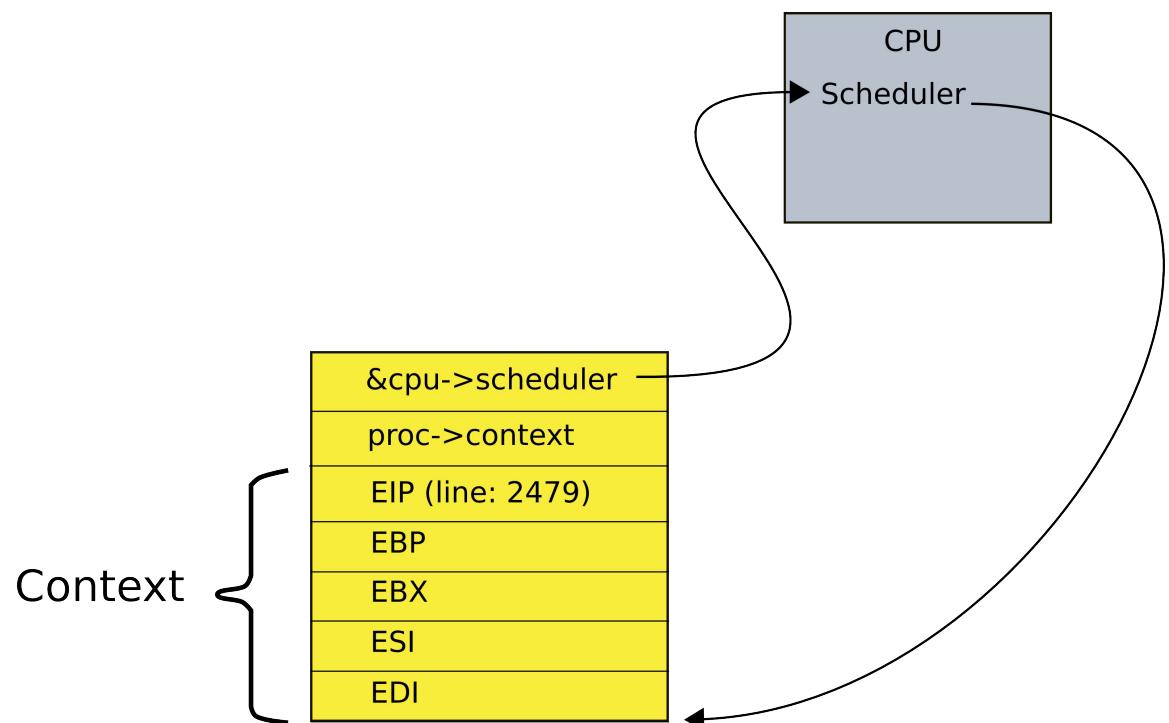
swtch()

```
void swtch(struct context **old,  
           struct context *new);
```

- First argument:
 - A pointer to a pointer to a context
 - That we're going to save
- Second argument:
 - A pointer to a context
 - We're going to restore

Context data structure

```
2093 struct context {  
2094     uint edi;  
2095     uint esi;  
2096     uint ebx;  
2097     uint ebp;  
2098     uint eip;  
2099 };
```

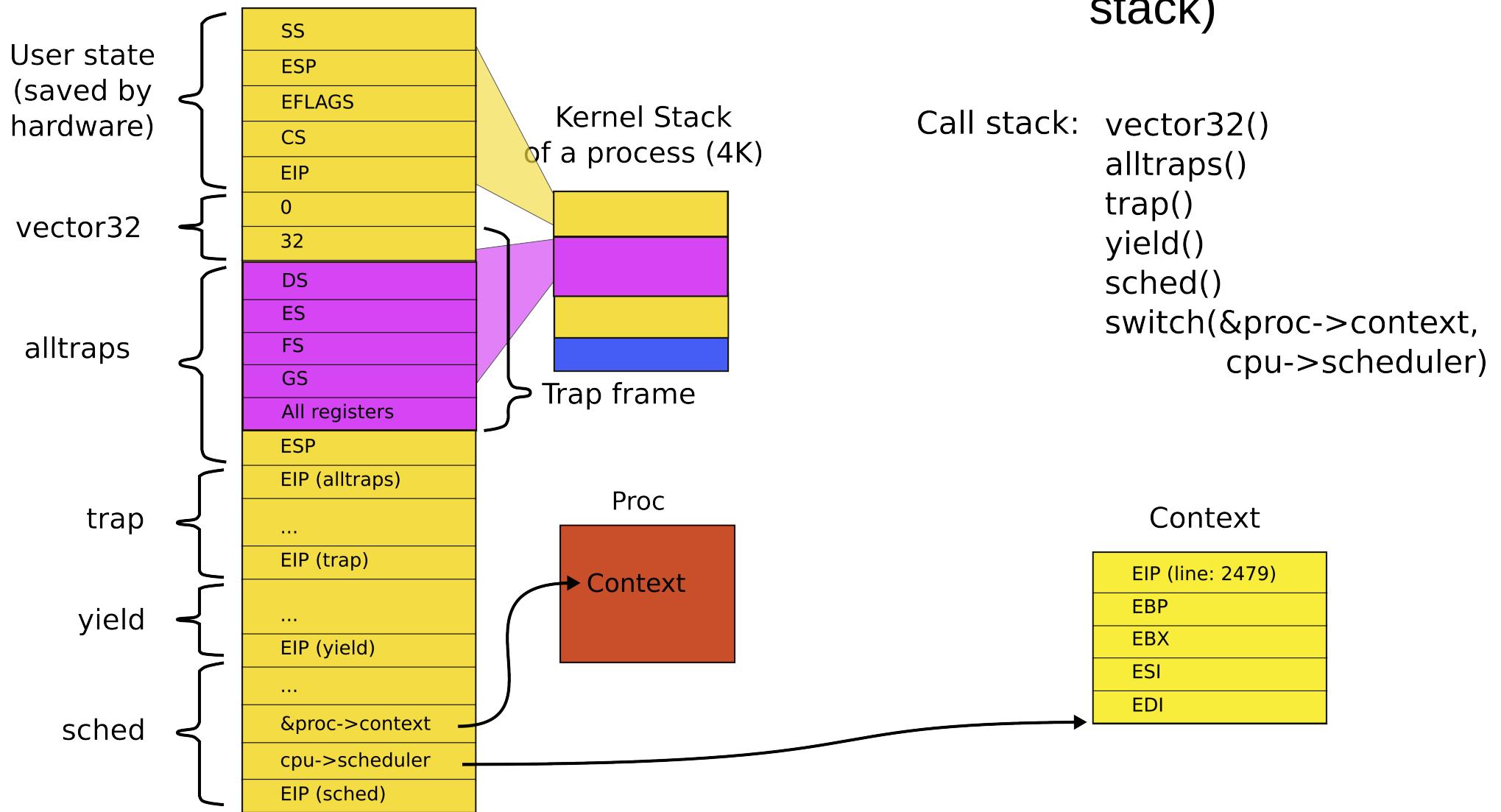


```
2958 swtch:  
2959    movl 4(%esp), %eax  
2960    movl 8(%esp), %edx  
2961  
2962 # Save old callee-save registers  
2963    pushl %ebp  
2964    pushl %ebx  
2965    pushl %esi  
2966    pushl %edi  
2967  
2968 # Switch stacksh  
2969    movl %esp, (%eax)  
2970    movl %edx, %esp  
2971  
2972 # Load new callee-save registers  
2973    popl %edi  
2974    popl %esi  
2975    popl %ebx  
2976    popl %ebp  
2977    ret
```

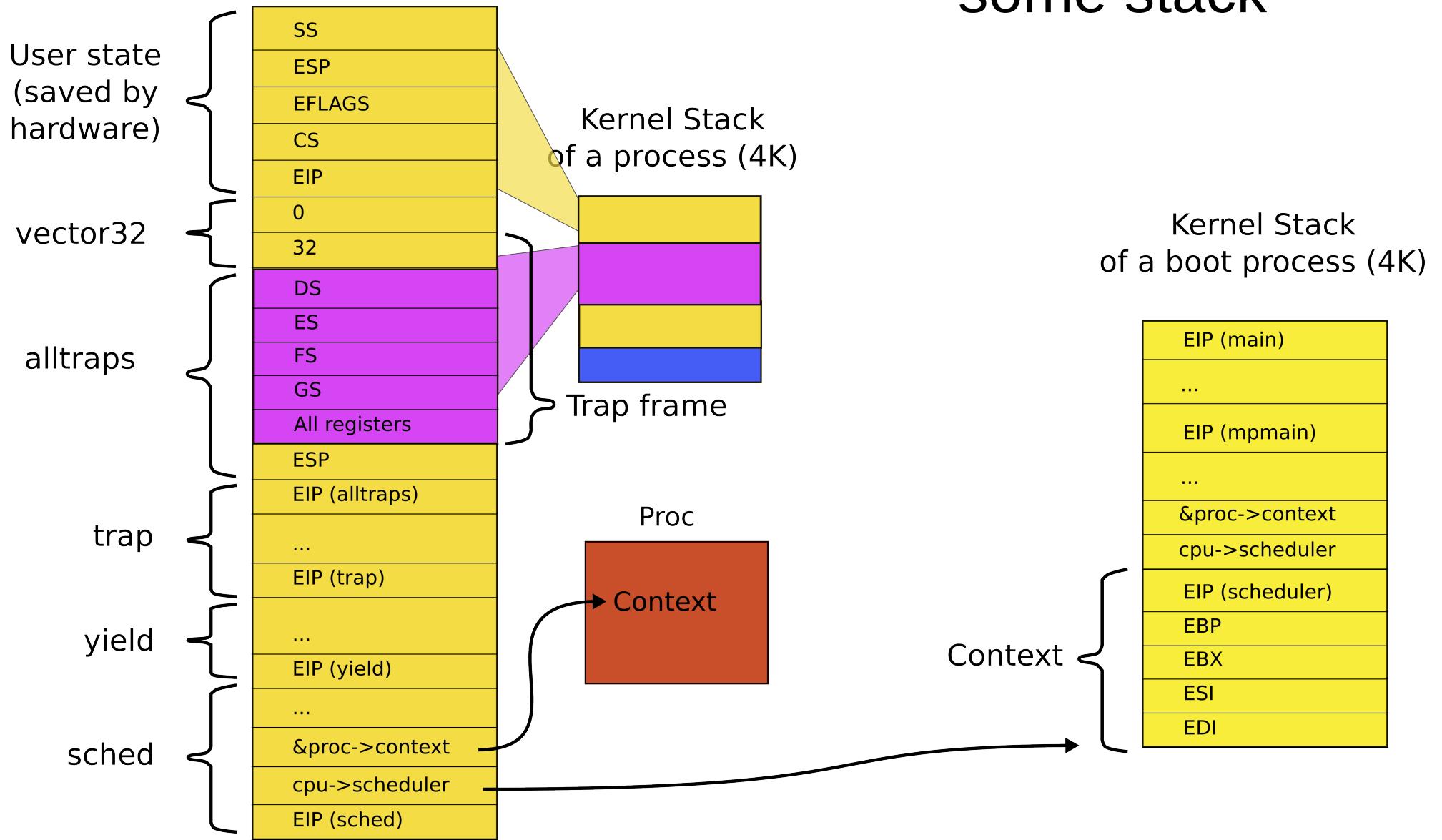
swtch()

```
2093 struct context {  
2094     uint edi;  
2095     uint esi;  
2096     uint ebx;  
2097     uint ebp;  
2098     uint eip;  
2099 };
```

Stack inside swtch() and its two arguments (passed on the stack)



Context is always top of some stack



Context is always top of some stack, why?

- How does initialization of each CPU end?

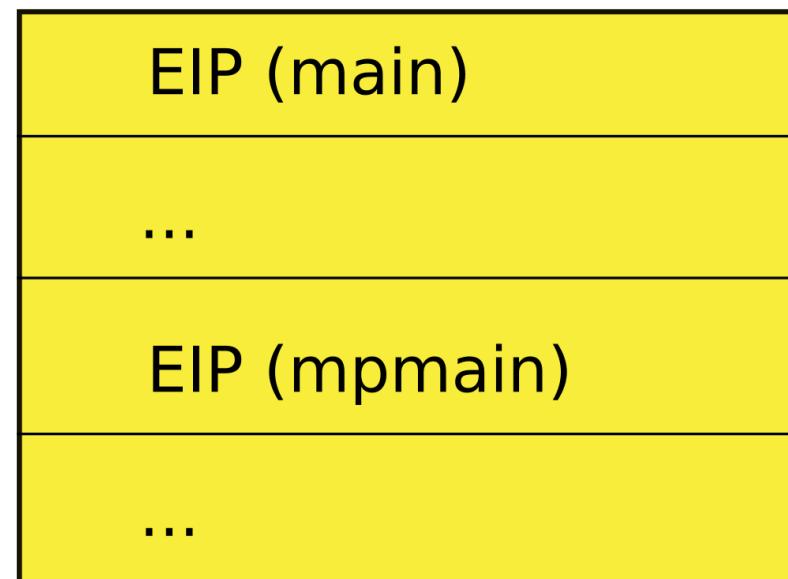
```
1251 static void  
1252 mpenter(void)  
1253 {  
1254     switchkvm();  
1255     seginit();  
1256     lapicinit();  
1257     mpmain();  
1258 }
```

```
1260 // Common CPU setup code.  
  
1261 static void  
1262 mpmain(void)  
1263 {  
1264     cprintf("cpu%d: starting\n", cpu->id);  
1265     idtinit(); // load idt register  
1266     xchg(&cpu->started, 1);  
1267     scheduler(); // start running  
processes  
1268 }
```

We ended boot by starting a scheduler

Remember the stack of the boot process?

Kernel Stack
of a boot process (4K)



```
2458 scheduler(void)
2459 {
2462     for(;;){
2468         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
2469             if(p->state != RUNNABLE)
2470                 continue;
2475             proc = p;
2476             switchuvm(p);
2477             p->state = RUNNING;
2478             swtch(&cpu->scheduler, proc->context);
2479             switchkvm();
2483             proc = 0;
2484         }
2487     }
2488 }
```

How does scheduler start?

- Chooses next process to run
- Switches to it
 - From the current context

- So when the scheduler context switched the first time

```
2478 swtch(&cpu->scheduler,  
            proc->context);
```

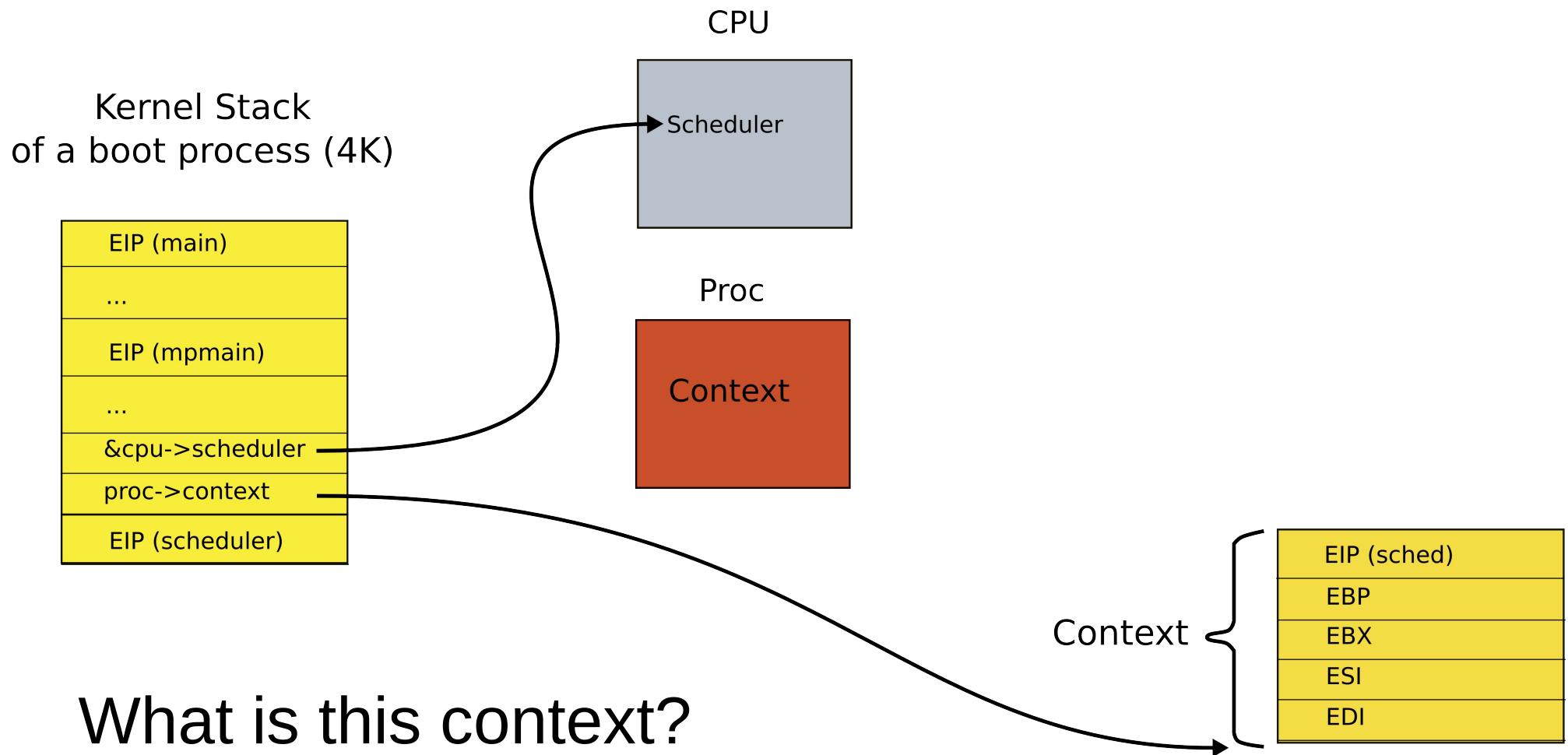
- It saved its own context

```
&cpu->scheduler
```

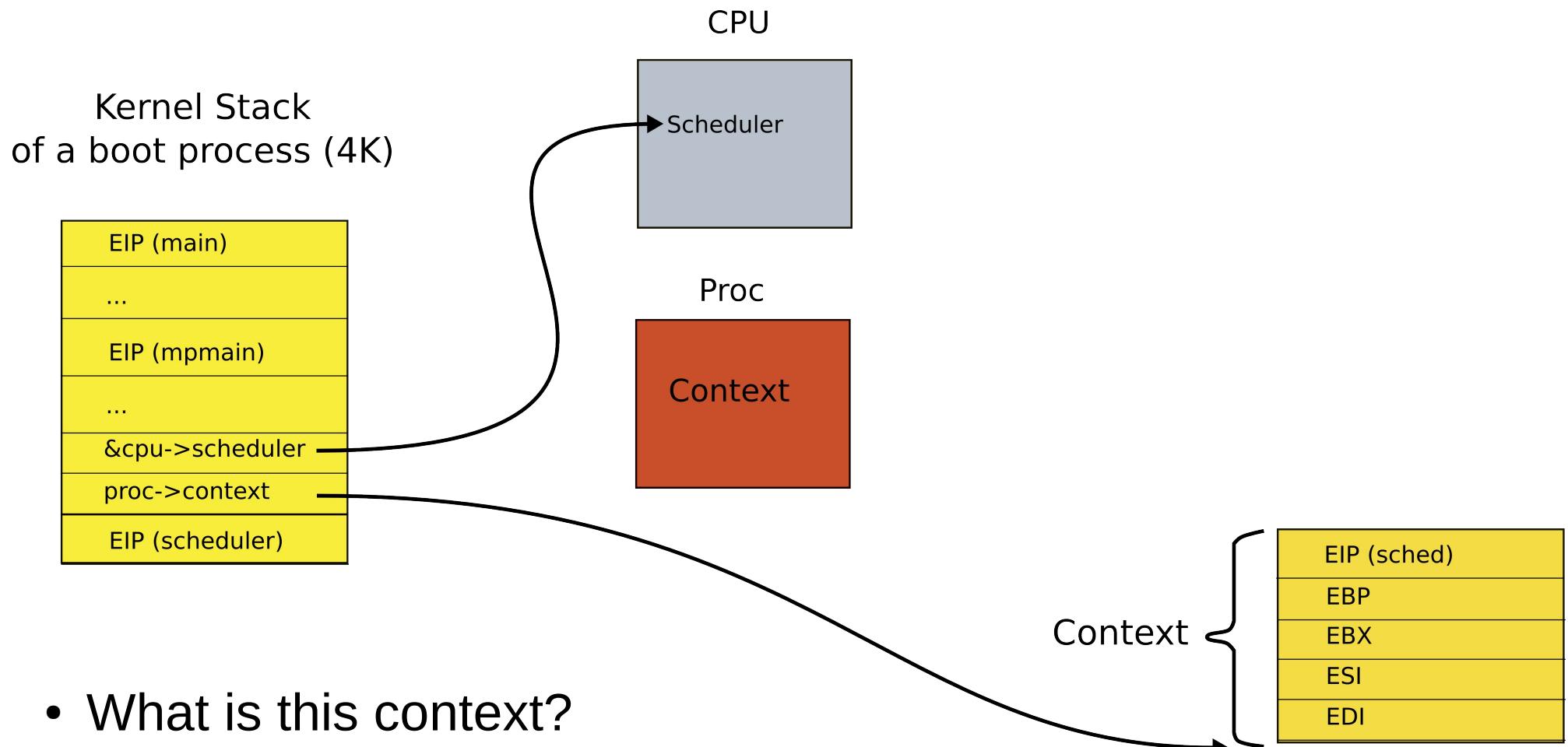
- And restored a context of the first process

```
proc->context
```

This is how stack looked like
when scheduler() invoked
swtch() for the first time

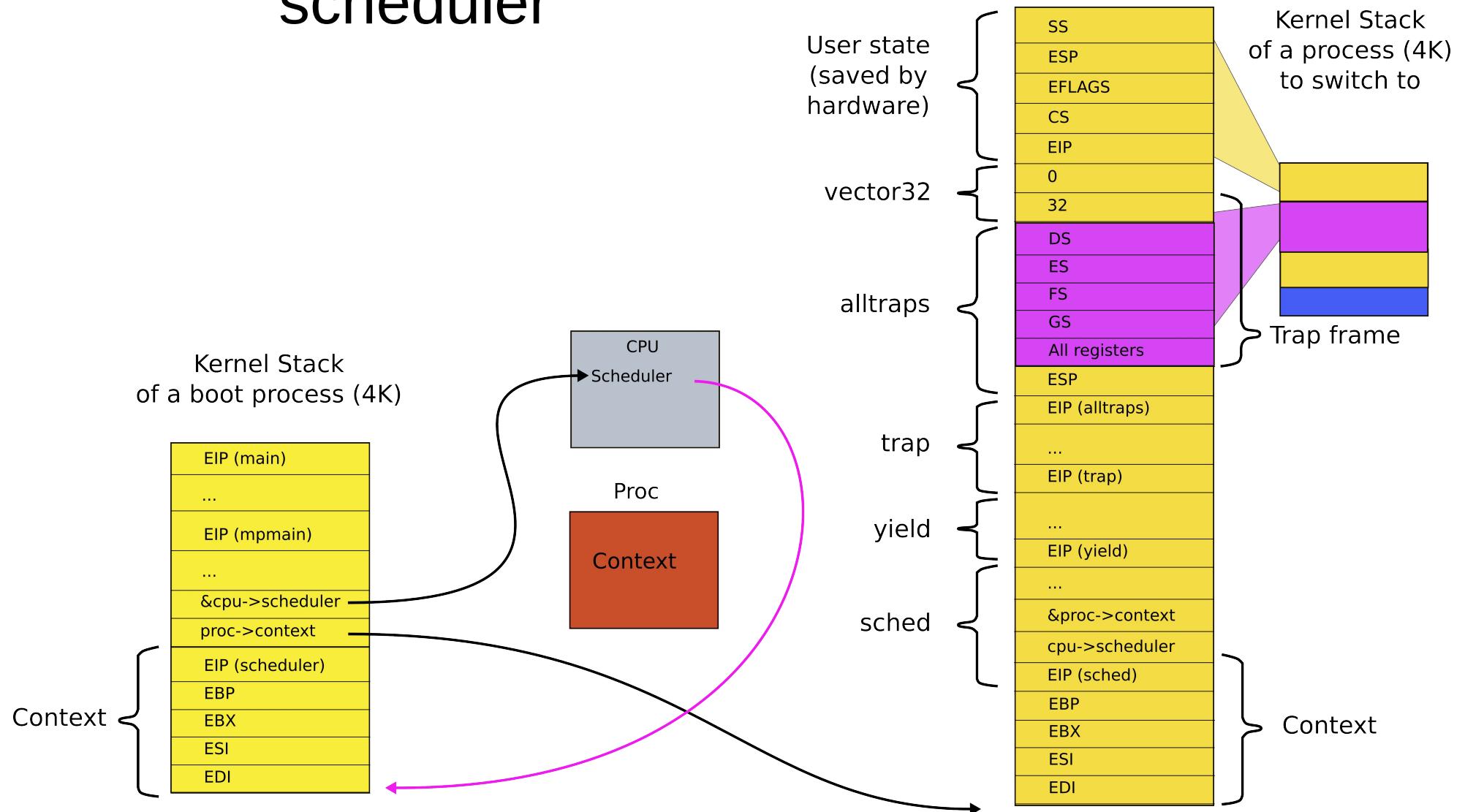


This is how stack looked like
when scheduler() invoked
swtch() for the first time



- What is this context?
- It's the context of the first process scheduler decides to run

Save context of the scheduler



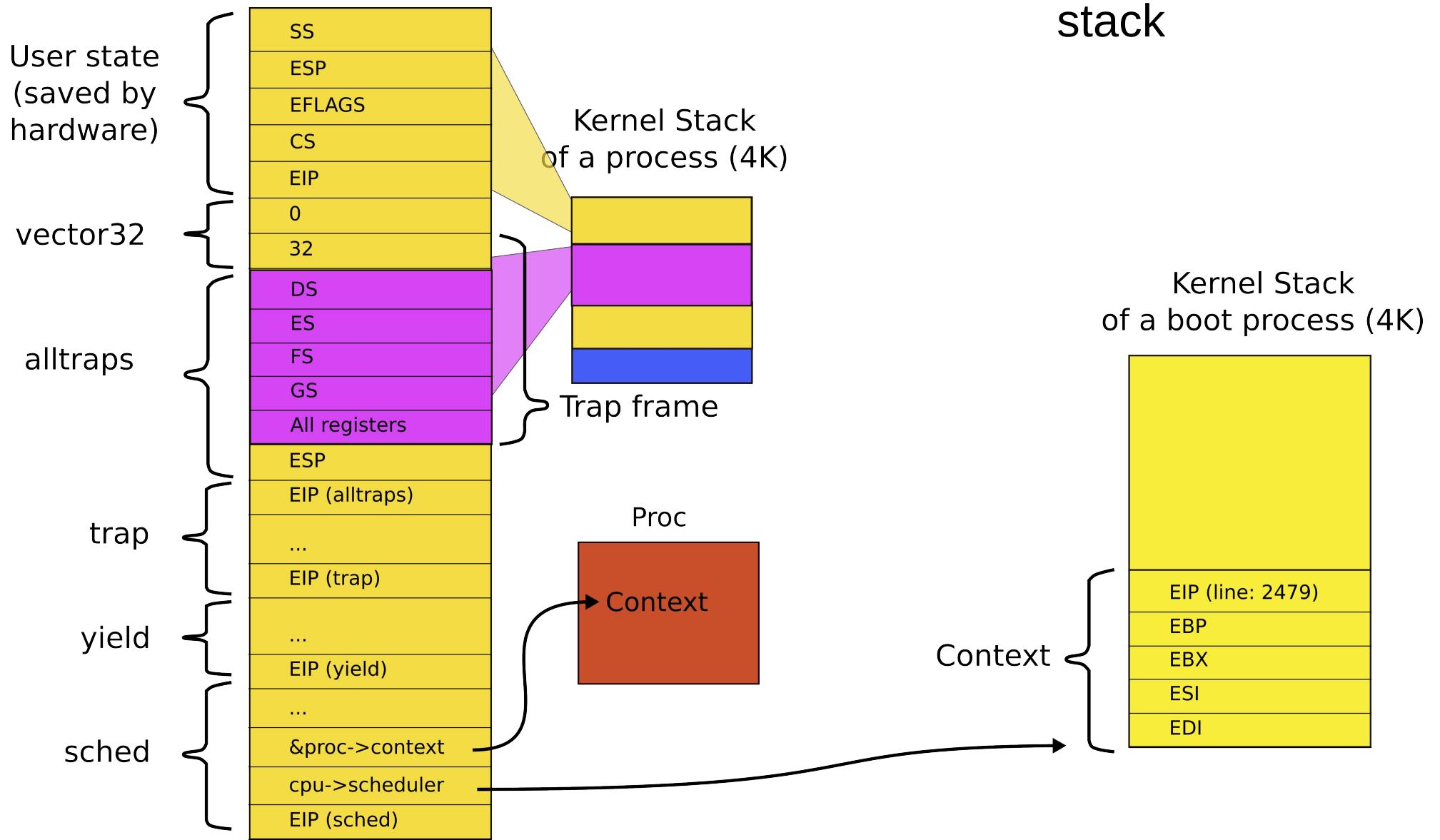
```
2958 swtch:  
2959 movl 4(%esp), %eax      // struct context **old  
2960 movl 8(%esp), %edx      // struct context *new  
2961  
2962 # Save old callee-save registers  
2963 pushl %ebp  
2964 pushl %ebx  
2965 pushl %esi  
2966 pushl %edi  
2967  
2968 # Switch stacksh  
2969 movl %esp, (%eax) // load current context (top of current stack) into  
                  // the memory location pointed by *old  
2970 movl %edx, %esp // set stack to be equal to *new (the top of the new context)  
2971  
2972 # Load new callee-save registers  
2973 popl %edi  
2974 popl %esi  
2975 popl %ebx  
2976 popl %ebp  
2977 ret
```

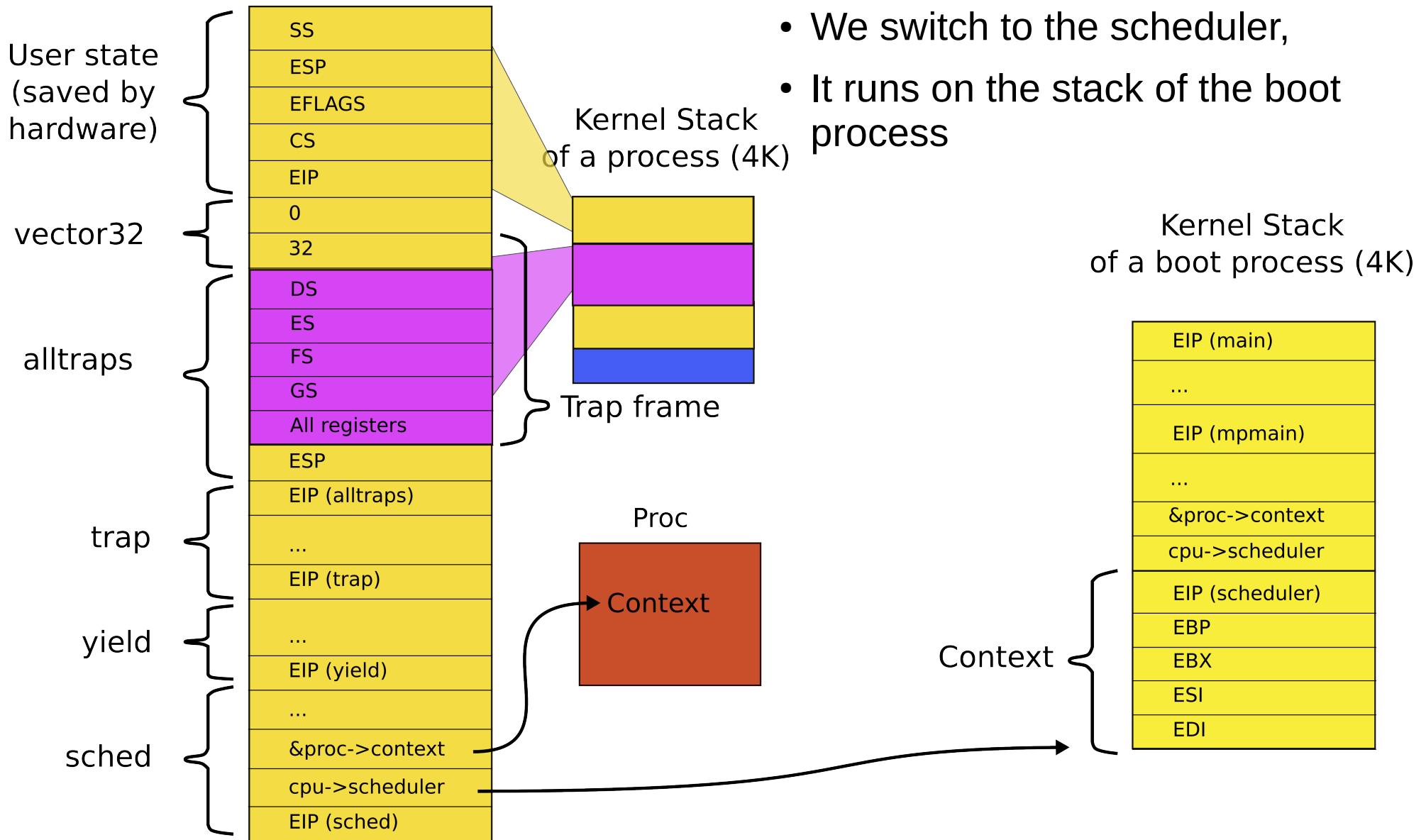
swtch()

This is why the context is the top of some stack

- Initially it was the stack of mpenter()
 - On which scheduler started
- Then first process...
 - Then scheduler again
 - And the next process...

Back to main context switch: so context is always top of some stack





```
2958 swtch:  
2959 movl 4(%esp), %eax      // struct context **old  
2960 movl 8(%esp), %edx      // struct context *new  
2961  
2962 # Save old callee-save registers  
2963 pushl %ebp  
2964 pushl %ebx  
2965 pushl %esi  
2966 pushl %edi  
2967  
2968 # Switch stacks  
2969 movl %esp, (%eax) // load current context (top of current stack) into  
                  // the memory location pointed by *old  
2970 movl %edx, %esp // set stack to be equal to *new (the top of the new context)  
2971  
2972 # Load new callee-save registers  
2973 popl %edi  
2974 popl %esi  
2975 popl %ebx  
2976 popl %ebp  
2977 ret
```

swtch()

```
2958 swtch:  
2959    movl 4(%esp), %eax  
2960    movl 8(%esp), %edx  
2961  
2962 # Save old callee-save registers  
2963    pushl %ebp  
2964    pushl %ebx  
2965    pushl %esi  
2966    pushl %edi  
2967  
2968 # Switch stacks  
2969    movl %esp, (%eax)  
2970    movl %edx, %esp  
2971  
2972 # Load new callee-save registers  
2973    popl %edi  
2974    popl %esi  
2975    popl %ebx  
2976    popl %ebp  
2977    ret
```

And now: exit from swtch()

Where does this switch() return?

Where does this swtch() return?

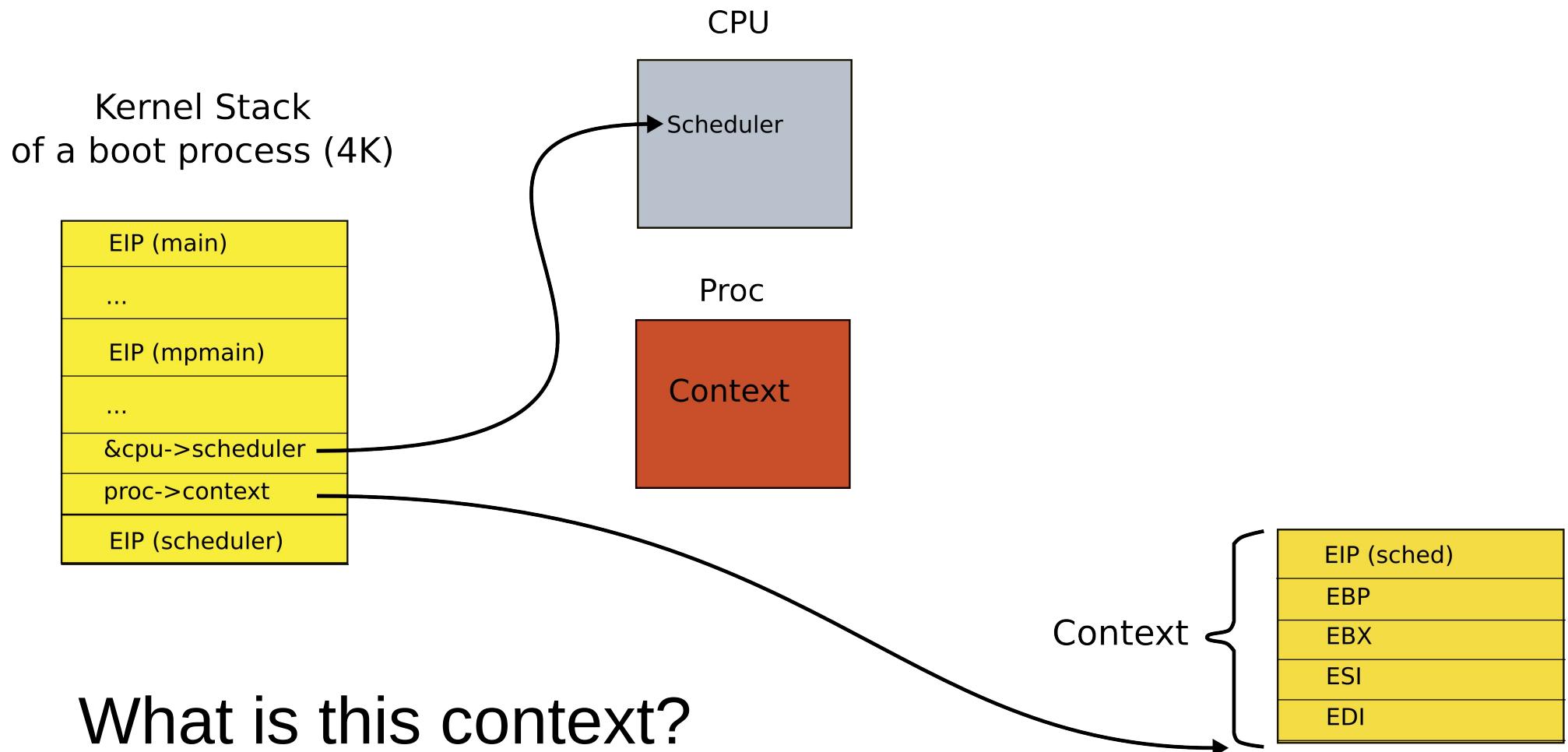
- Scheduler
- After all remember
 - We started with timer interrupt
 - Entered the kernel
 - Entered schedule()
 - Entered switch
- And are currently on our way from the process into the scheduler

```
2458 scheduler(void)
2459 {
2462     for(;;){
2468         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
2469             if(p->state != RUNNABLE)
2470                 continue;
2475             proc = p;
2476             switchuvm(p);
2477             p->state = RUNNING;
2478             swtch(&cpu->scheduler, proc->context);
2479             switchkvm();
2483             proc = 0;
2484         }
2487     }
2488 }
```

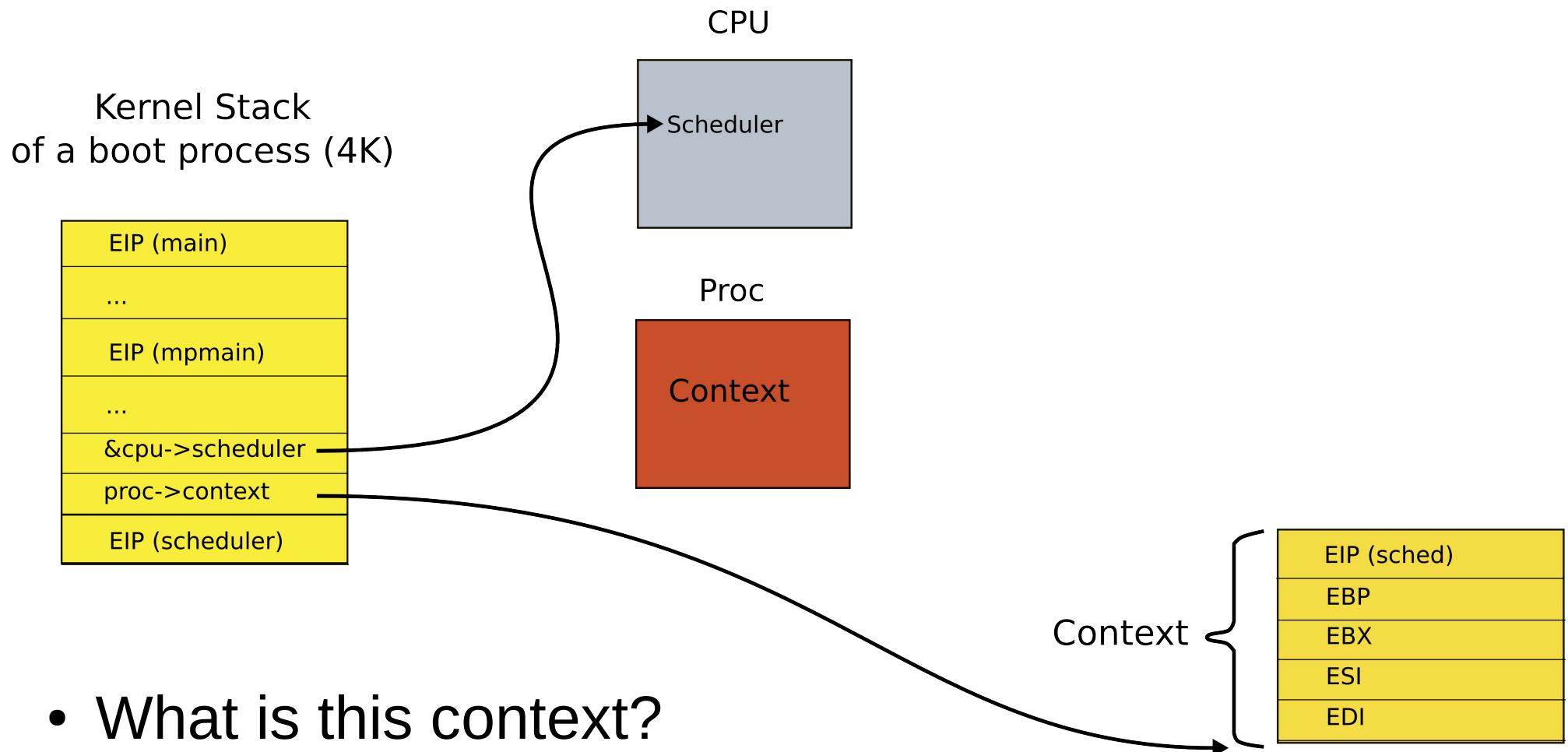
What does scheduler do?

- Chooses next process to run
- Switches to it

What does stack look like when
scheduler() invokes swtch()?

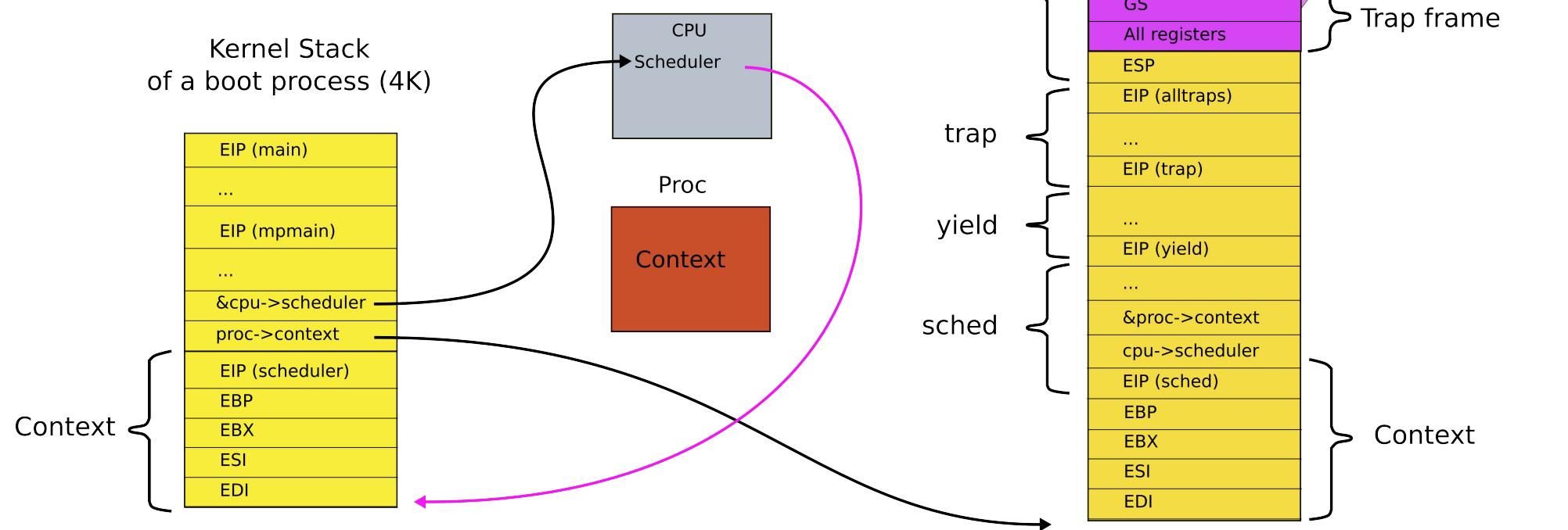


What does stack look like when
scheduler() invokes swtch()?



- What is this context?
 - Right the context of the next process to run

- We save the context of the scheduler again
- Restore the context of the next process

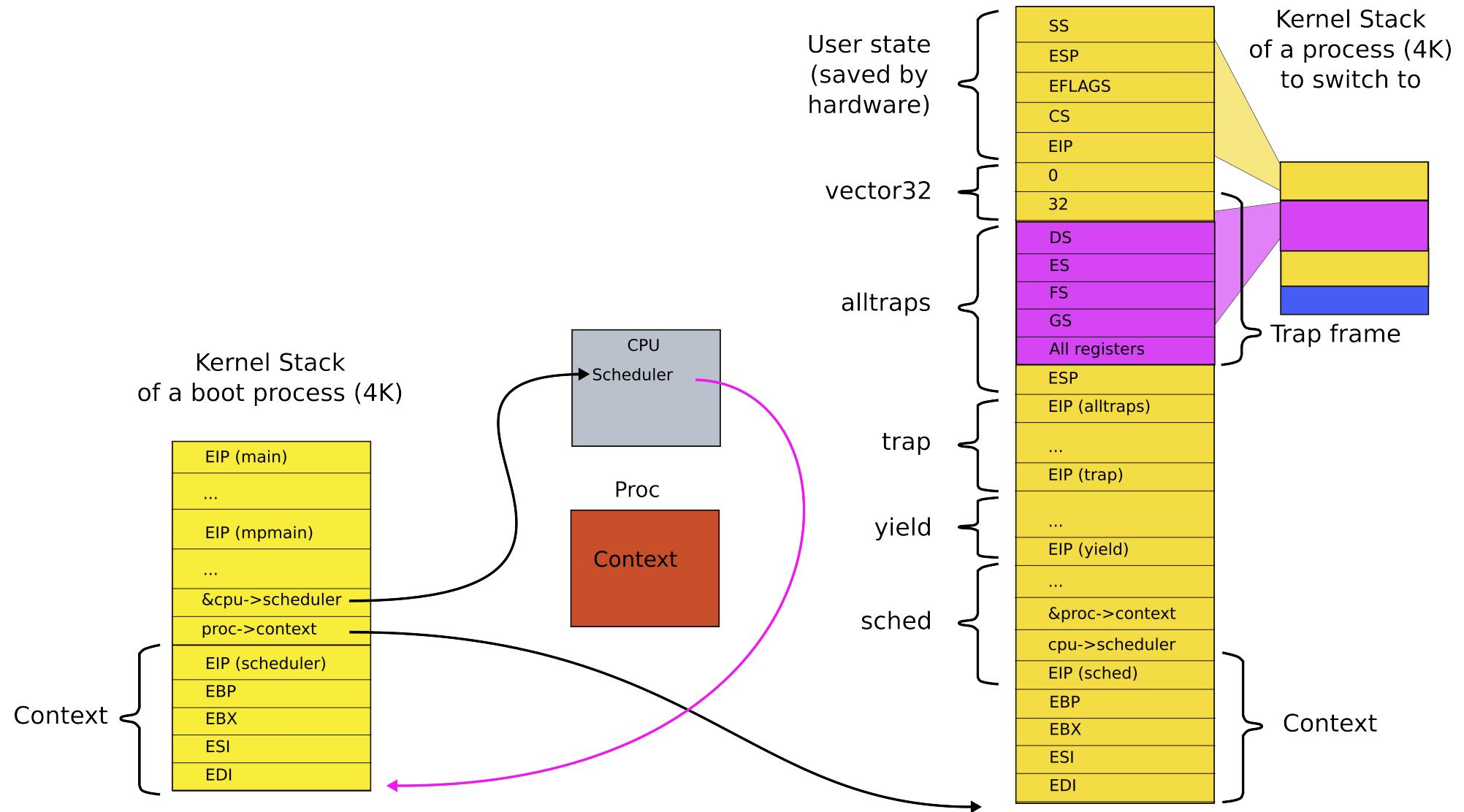


- Remember, from inside the scheduler we invoked swtch() as

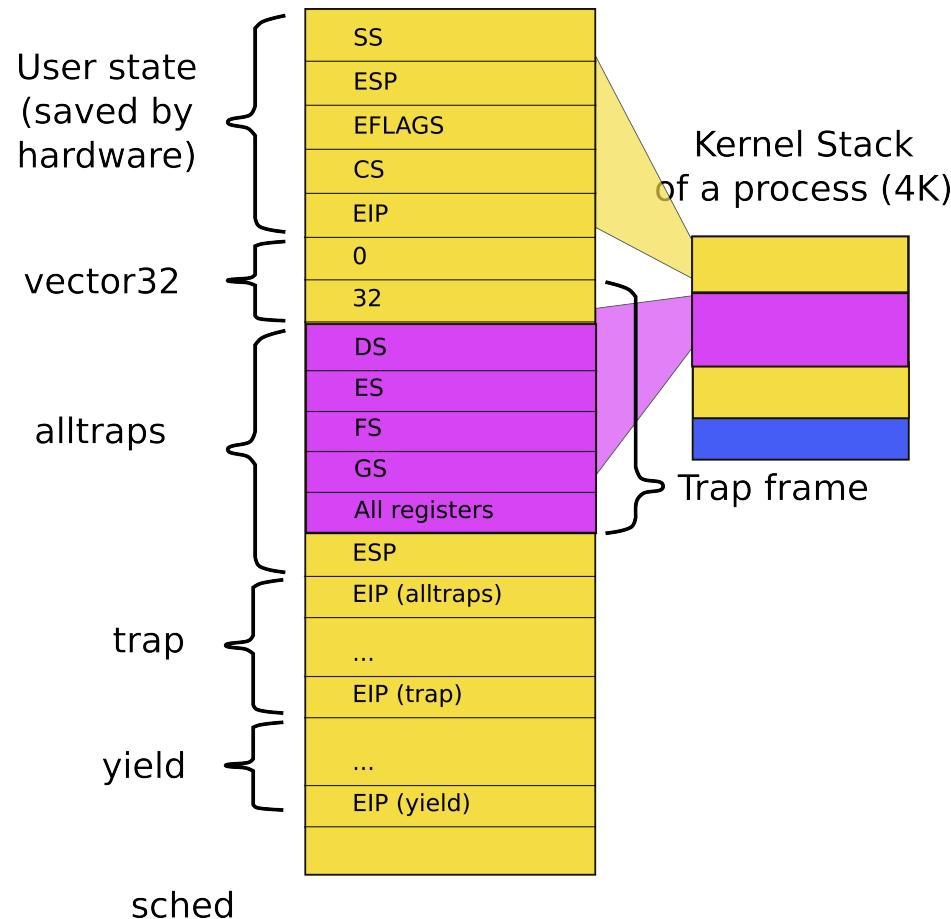
```
2478 swtch(&cpu->scheduler,  
           proc->context);
```

- Hence, we save context of the scheduler into
`&cpu->scheduler`
- And restore
`proc->context`

Stacks and context inside swtch()



Exiting back to user-level

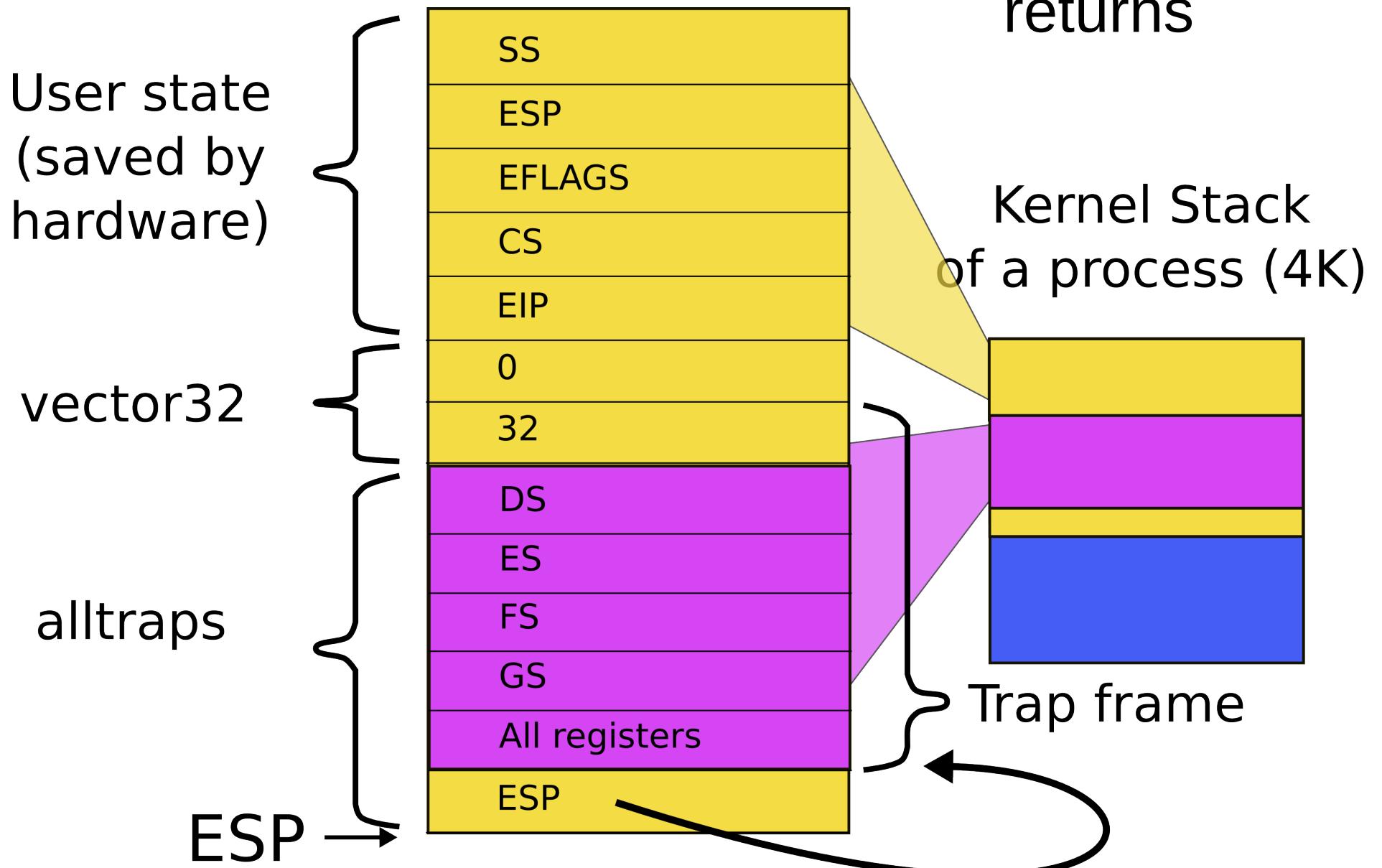


- Stack inside sched()
 - Normal returns until back to alltrap()

```
3004 alltraps:  
...  
3020 # Call trap(tf), where tf=%esp  
3021 pushl %esp  
3022 call trap  
3023 addl $4, %esp  
3024  
3025 # Return falls through to trapret...  
3026 .globl trapret  
3027 trapret:  
3028 popal  
3029 popl %gs  
3030 popl %fs  
3031 popl %es  
3032 popl %ds  
3033 addl $0x8, %esp # trapno and errcode  
3034 iret
```

alltraps(): Exiting back into user level process

Stack after trap()
returns



```
3004 alltraps:
```

```
...
```

```
3020 # Call trap(tf), where tf=%esp
```

```
3021 pushl %esp
```

```
3022 call trap
```

```
3023 addl $4, %esp
```

```
3024
```

```
3025 # Return falls through to trapret...
```

```
3026 .globl trapret
```

```
3027 trapret:
```

```
3028 popal
```

```
3029 popl %gs
```

```
3030 popl %fs
```

```
3031 popl %es
```

```
3032 popl %ds
```

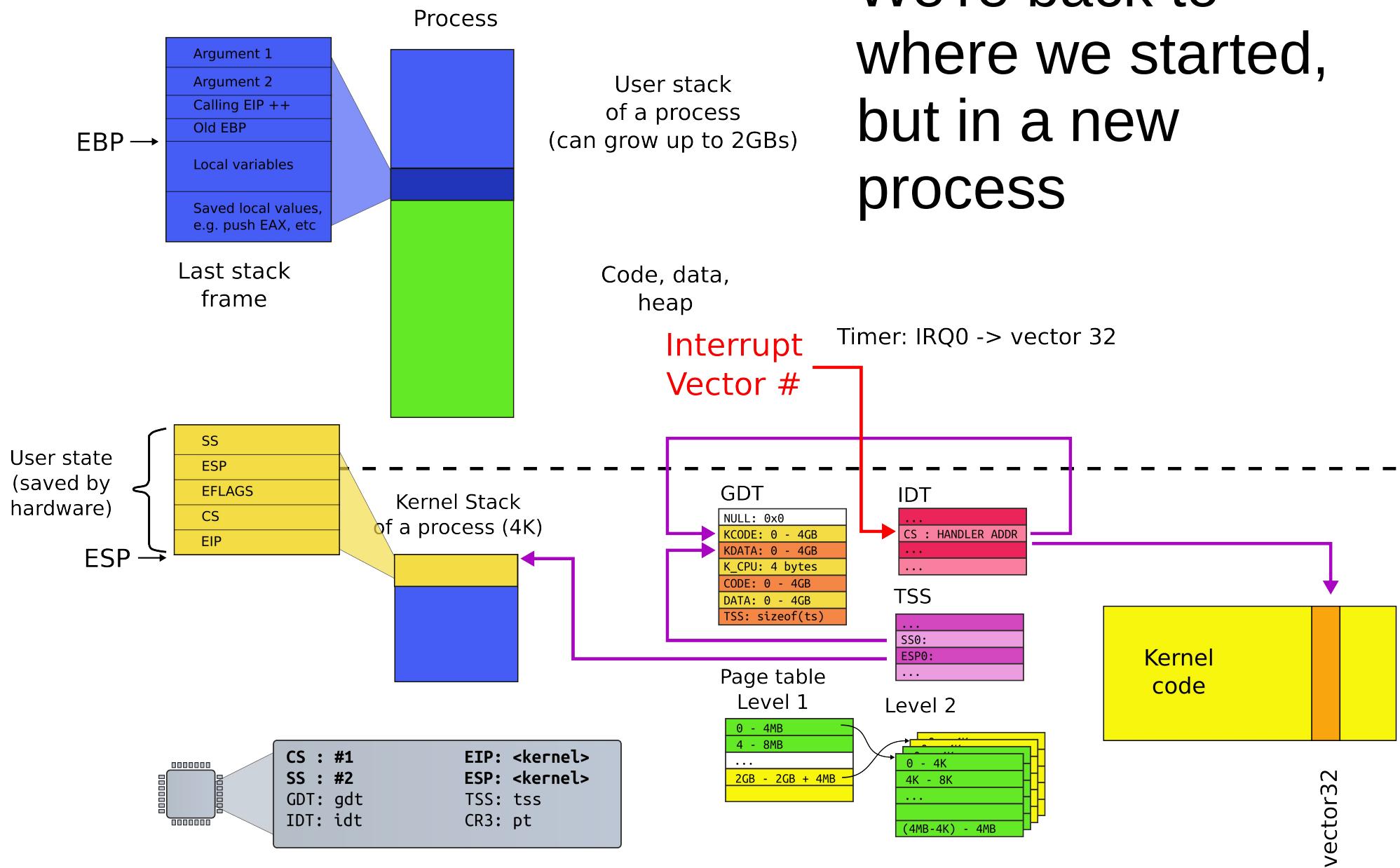
```
3033 addl $0x8, %esp # trapno and errcode
```

```
3034 iret
```

alltraps(): exiting

- Restore all registers
- Exit into user
 - iret

We're back to where we started, but in a new process



Summary

- We switch between processes now

Thank you