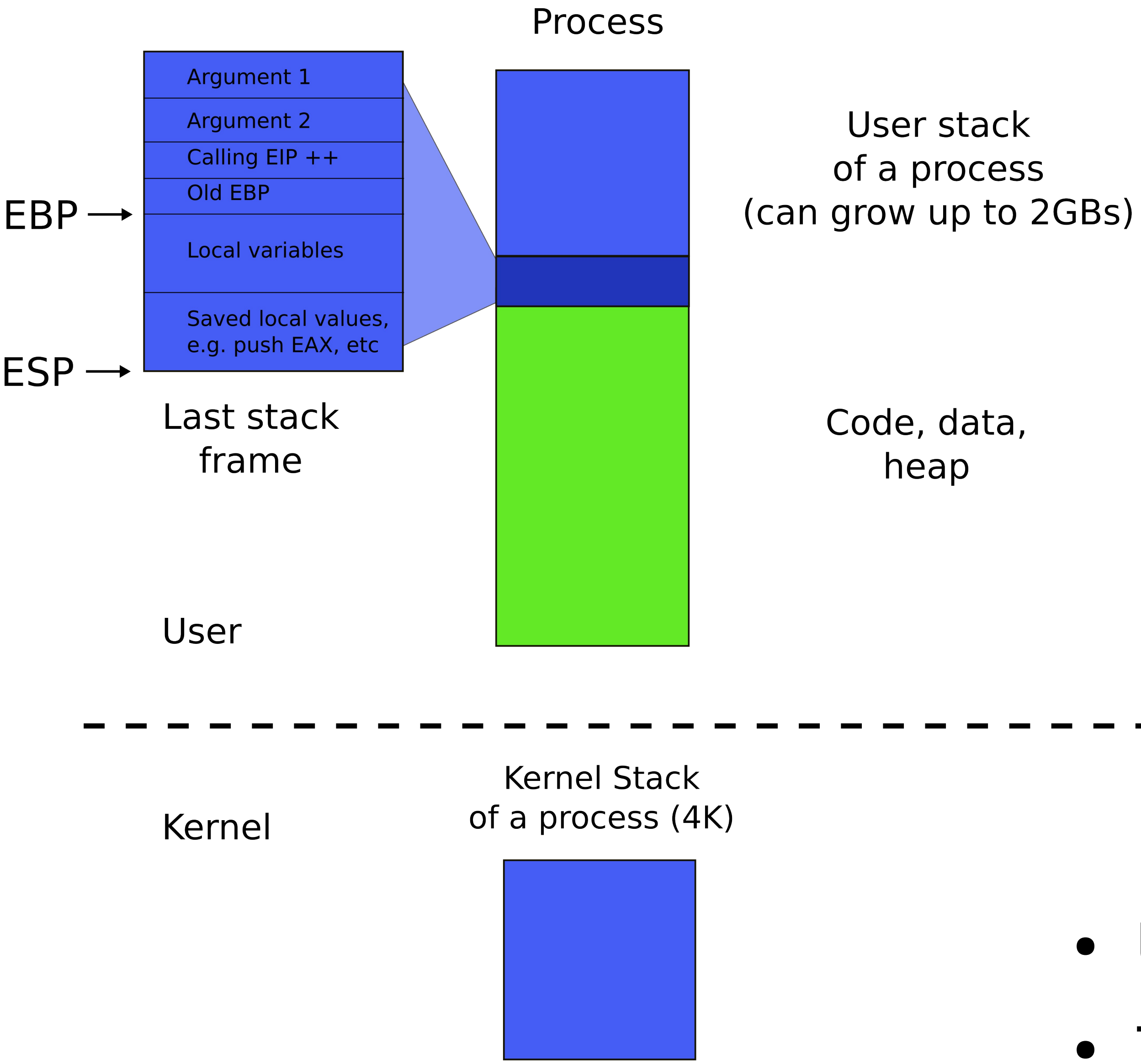
CS5460/6460: Operating Systems

Lecture 10: Context switching

Anton Burtsev
February, 2014



Process

Argument 1

Argument 2

Calling EIP ++

Old EBP

Local variables

Saved local values, e.g. push EAX, etc

EBP →

ESP →

Last stack frame

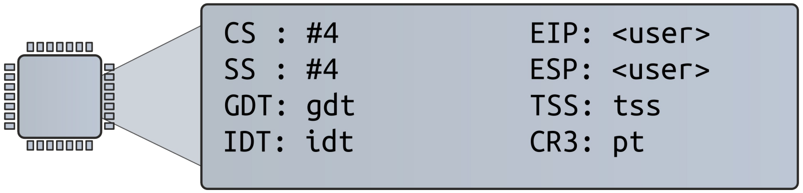
User stack of a process (can grow up to 2GBs)

Code, data, heap

User

Kernel

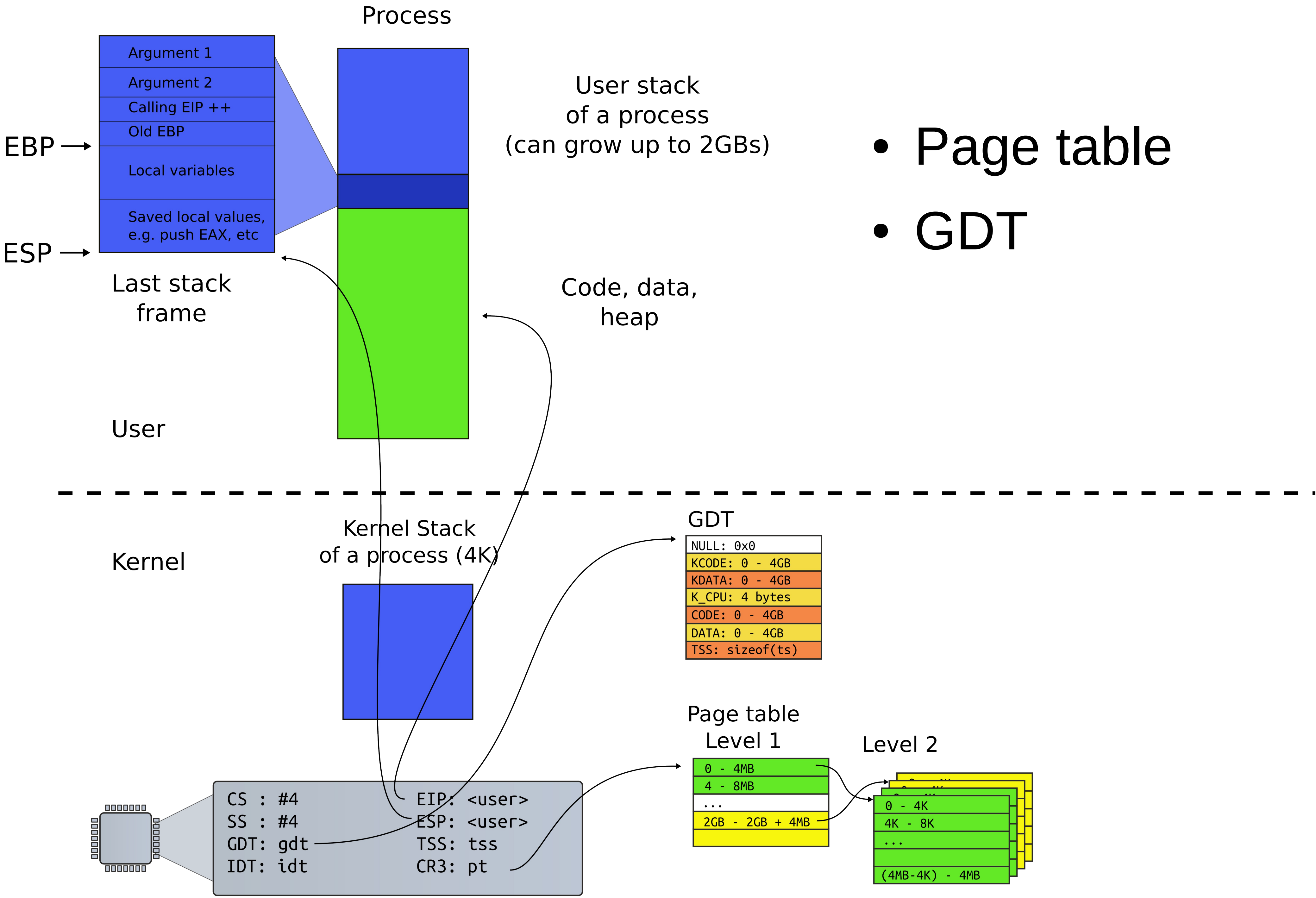
Kernel Stack of a process (4K)



```
CS : #4              EIP: <user>
SS : #4              ESP: <user>
GDT: gdt             TSS: tss
IDT: idt             CR3: pt
```

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

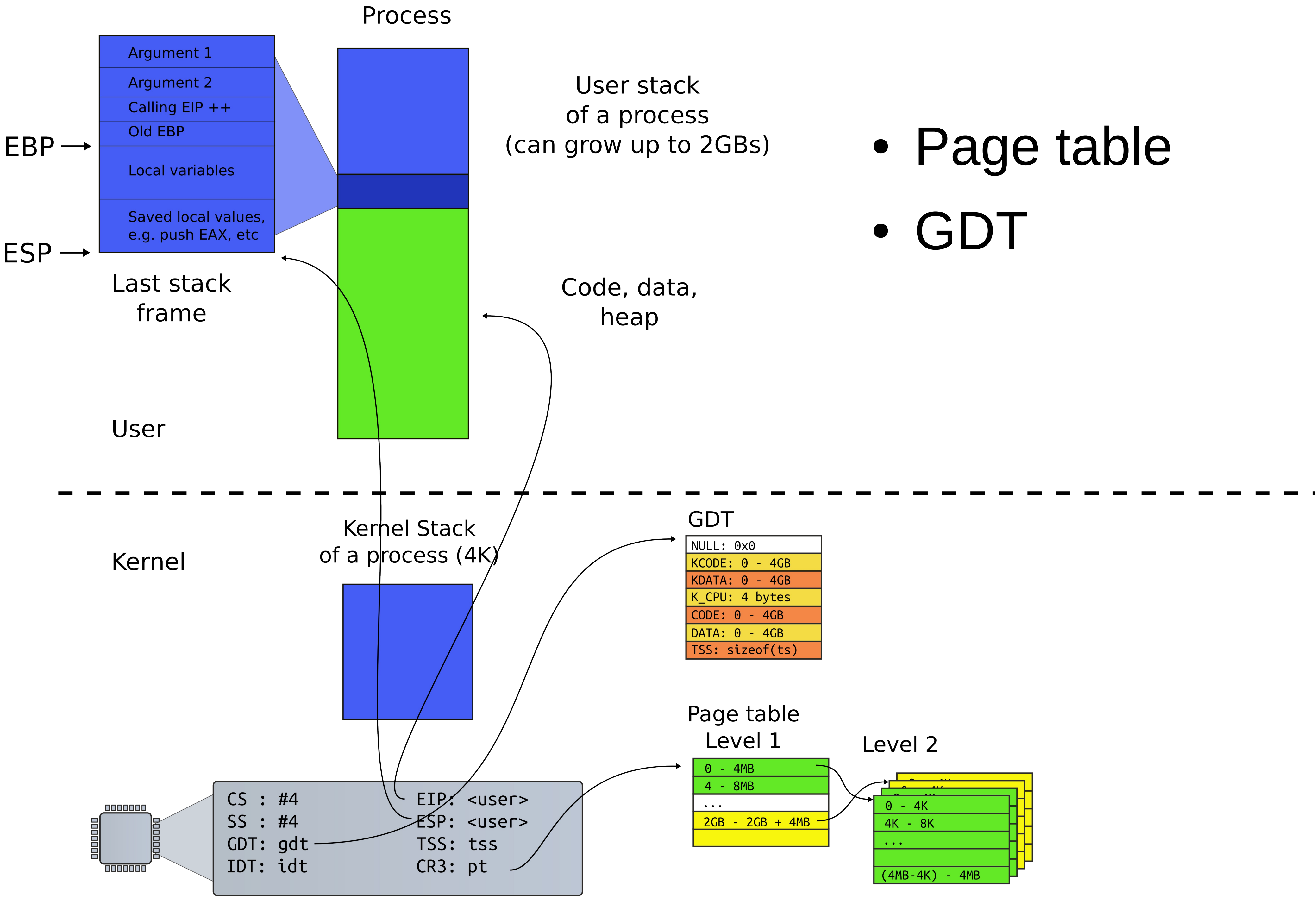
- Page table
- GDT



```
CS : #4          EIP: <user>
SS : #4          ESP: <user>
GDT: gdt         TSS: tss
IDT: idt         CR3: pt
```

GDT
NULL: 0x0
KCODE: 0 - 4GB
KDATA: 0 - 4GB
K_CPU: 4 bytes
CODE: 0 - 4GB
DATA: 0 - 4GB
TSS: sizeof(ts)

- Page table
- GDT



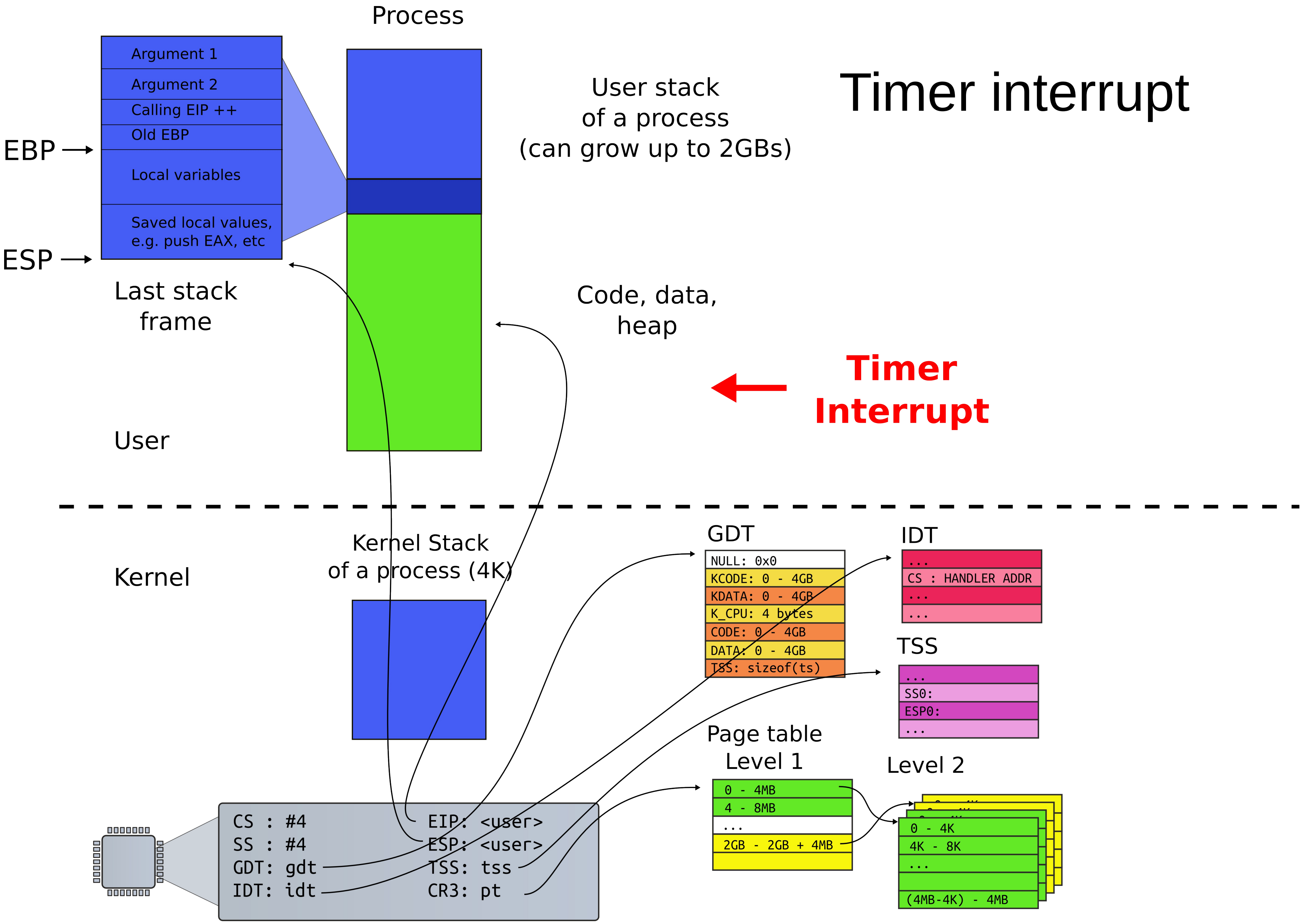
GDT
NULL: 0x0
KCODE: 0 - 4GB
KDATA: 0 - 4GB
K_CPU: 4 bytes
CODE: 0 - 4GB
DATA: 0 - 4GB
TSS: sizeof(ts)

Page table Level 1
0 - 4MB
4 - 8MB
...
2GB - 2GB + 4MB

Level 2
0 - 4K
4K - 8K
...
(4MB-4K) - 4MB

```
CS : #4          EIP: <user>
SS : #4          ESP: <user>
GDT: gdt         TSS: tss
IDT: idt         CR3: pt
```

Timer interrupt



Process

Argument 1
Argument 2
Calling EIP ++
Old EBP
Local variables
Saved local values, e.g. push EAX, etc

EBP

ESP

Last stack frame

User stack of a process (can grow up to 2GBs)

Code, data, heap

Timer Interrupt

User

Kernel

Kernel Stack of a process (4K)

GDT

NULL: 0x0
KCODE: 0 - 4GB
KDATA: 0 - 4GB
K_CPU: 4 bytes
CODE: 0 - 4GB
DATA: 0 - 4GB
TSS: sizeof(ts)

IDT

...
CS : HANDLER ADDR
...
...

TSS

...
SS0:
ESP0:
...

Page table

Level 1

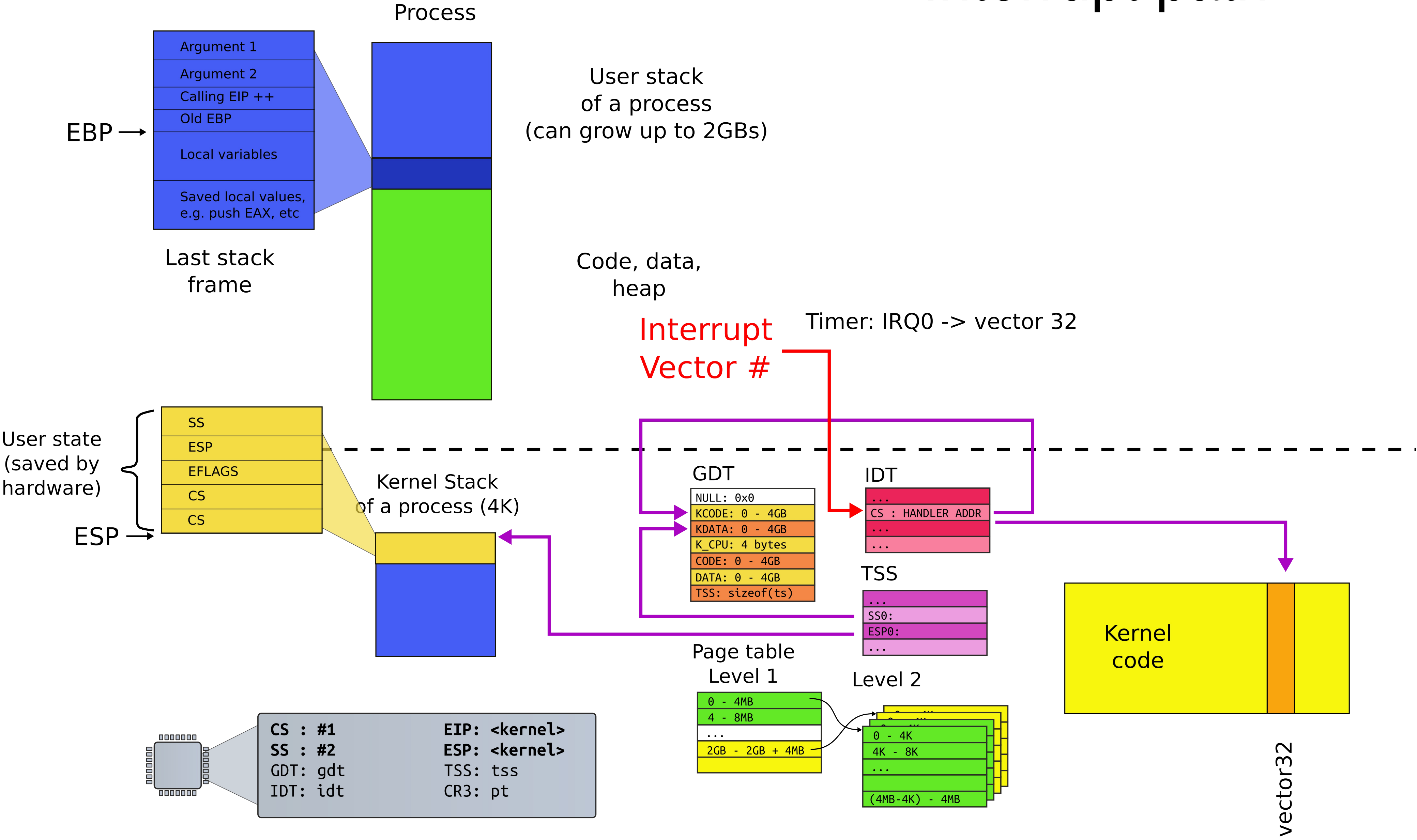
0 - 4MB
4 - 8MB
...
2GB - 2GB + 4MB

Level 2

0 - 4K
4K - 8K
...
(4MB-4K) - 4MB

```
CS  : #4          EIP: <user>
SS  : #4          ESP: <user>
GDT: gdt          TSS: tss
IDT: idt          CR3: pt
```

Interrupt path



GDT

```
NULL: 0x0
KCODE: 0 - 4GB
KDATA: 0 - 4GB
K_CPU: 4 bytes
CODE: 0 - 4GB
DATA: 0 - 4GB
TSS: sizeof(ts)
```

IDT

```
...
CS : HANDLER ADDR
...
...
```

TSS

```
...
SS0:
ESP0:
...
```

Page table Level 1

```
0 - 4MB
4 - 8MB
...
2GB - 2GB + 4MB
```

Level 2

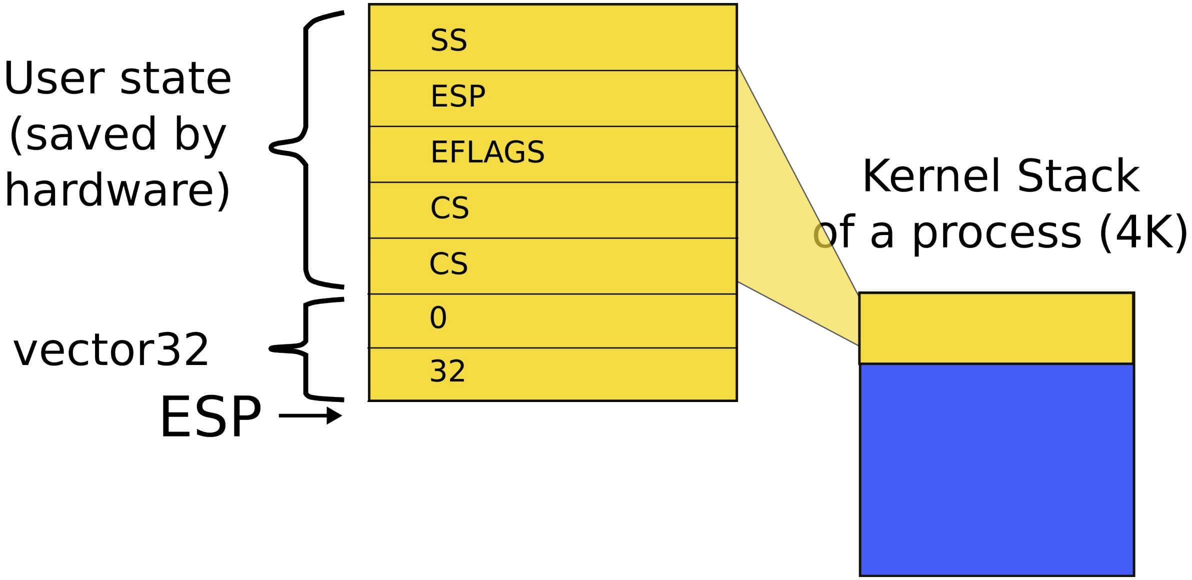
```
0 - 4K
4K - 8K
...
(4MB-4K) - 4MB
```

```
CS : #1           EIP: <kernel>
SS : #2           ESP: <kernel>
GDT: gdt          TSS: tss
IDT: idt          CR3: pt
```

Where does IDT (entry 32) point to?

```
vector32:
pushl $0     // error code
pushl $32    // vector #
jmp alltraps
```

Kernel stack after interrupt


SS
ESP
EFLAGS
CS
CS
0
32
User state (saved by hardware)
vector32
ESP
Kernel Stack of a process (4K)

Call stack: vector32()

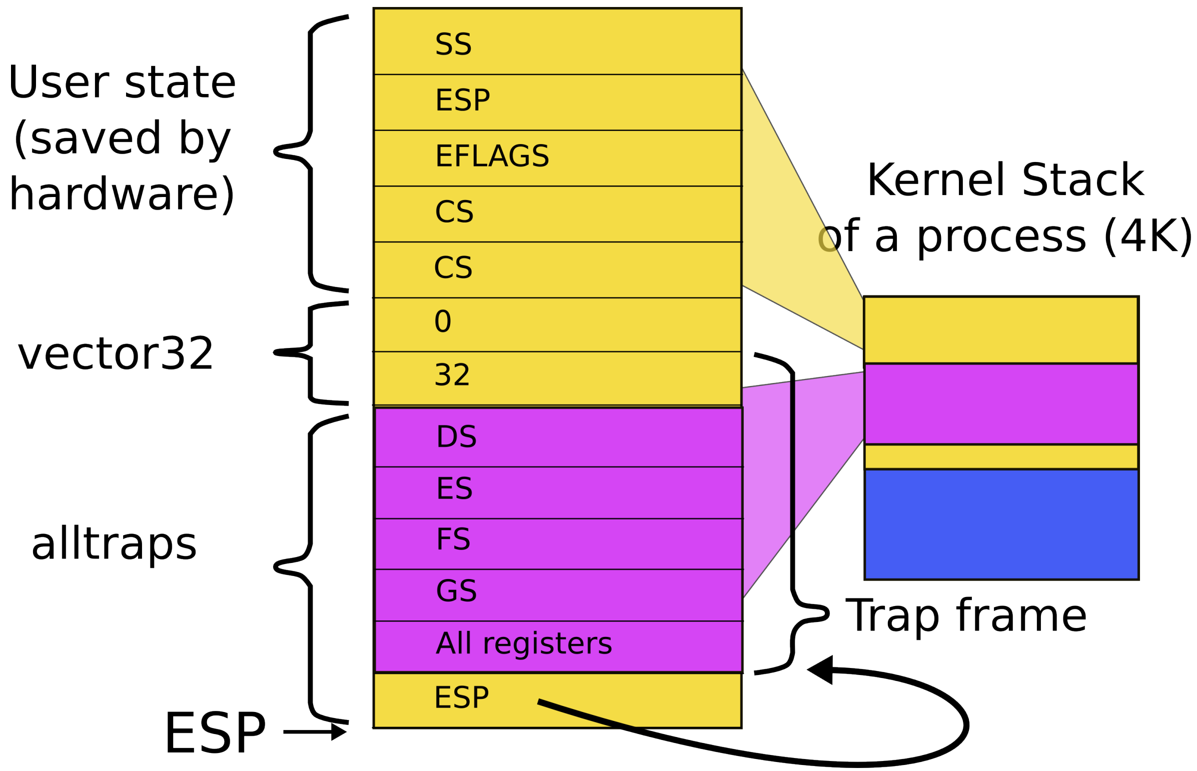
alltraps()

```
alltraps:
# Build trap frame.
pushl %ds
pushl %es
pushl %fs
pushl %gs
pushal

# Set up data and per-cpu segments.
movw $(SEG_KDATA<<3), %ax
movw %ax, %ds
movw %ax, %es
movw $(SEG_KCPU<<3), %ax
movw %ax, %fs
movw %ax, %gs

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

Kernel stack after interrupt



Call stack: vector32()
alltraps()

alltraps()

```
alltraps:
# Build trap frame.
pushl %ds
pushl %es
pushl %fs
pushl %gs
pushal

# Set up data and per-cpu segments.
movw $(SEG_KDATA<<3), %ax
movw %ax, %ds
movw %ax, %es
movw $(SEG_KCPU<<3), %ax
movw %ax, %fs
movw %ax, %gs

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

trap()

```
trap(struct trapframe *tf)
{
...
  switch(tf->trapno){
  case T_IRQ0 + IRQ_TIMER:
    if(cpu->id == 0){
    acquire(&tickslock);
    ticks++;
    wakeup(&ticks);
    release(&tickslock);
  }
  break;
...
  if(proc && proc->state == RUNNING
     && tf->trapno == T_IRQ0+IRQ_TIMER)
    yield();
```

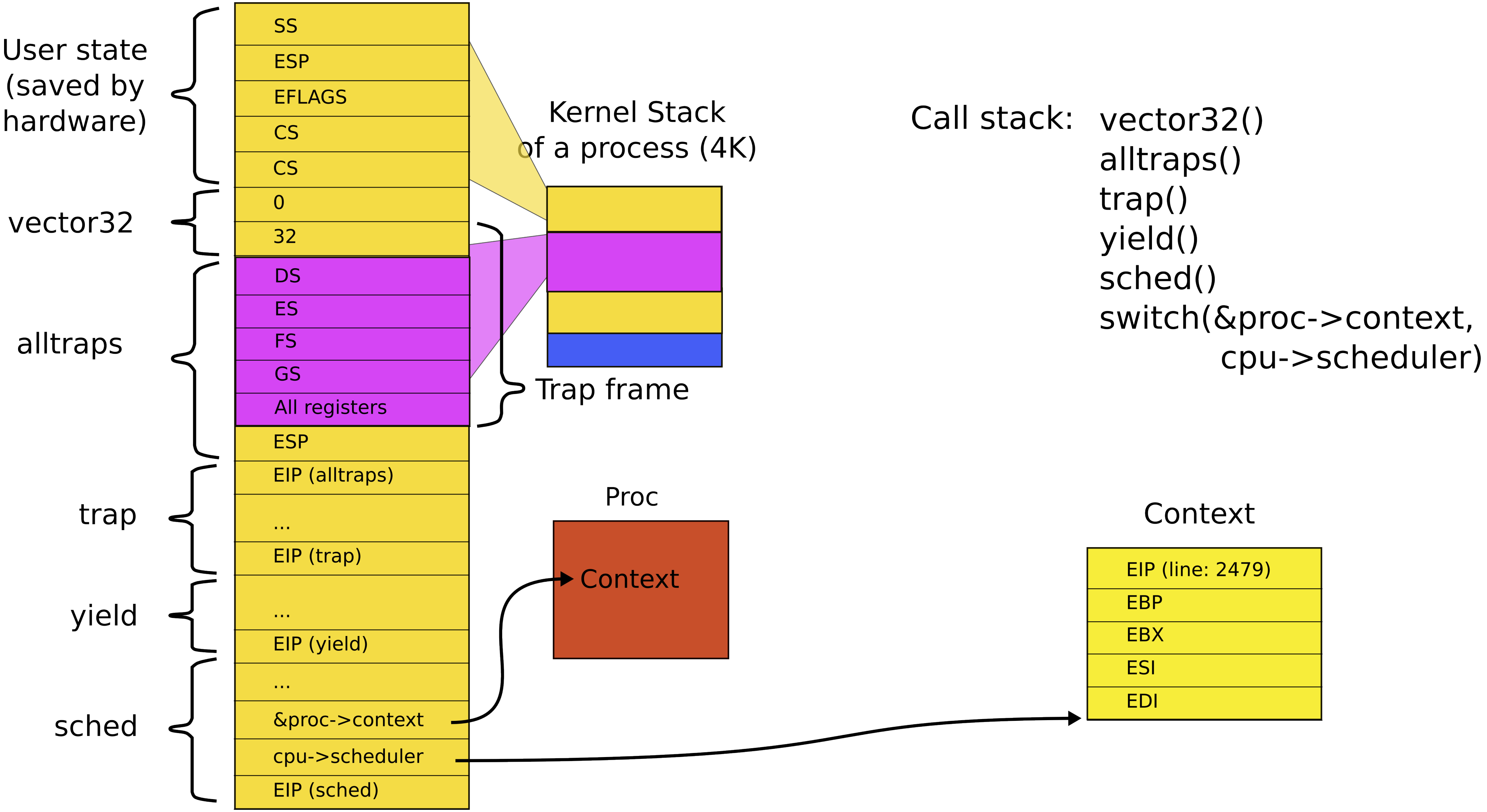
Invoke the scheduler

```
yield(void)
{
  acquire(&ptable.lock);
  proc->state = RUNNABLE;
  sched();
  release(&ptable.lock);
}
```

Start the context switch

```
sched(void)
{
...
  if(proc->state == RUNNING)
    panic("sched running");
...
  swtch(&proc->context, cpu->scheduler);
..
}
```

Stack inside swtch()



User state (saved by hardware)

- SS
- ESP
- EFLAGS
- CS
- CS

vector32

- 0
- 32

alltraps

- DS
- ES
- FS
- GS
- All registers
- ESP

trap

- EIP (alltraps)
- ...
- EIP (trap)

yield

- ...
- EIP (yield)

sched

- ...
- &proc->context
- cpu->scheduler
- EIP (sched)

Trap frame

Kernel Stack of a process (4K)

Call stack: vector32()
alltraps()
trap()
yield()
sched()
switch(&proc->context,
cpu->scheduler)

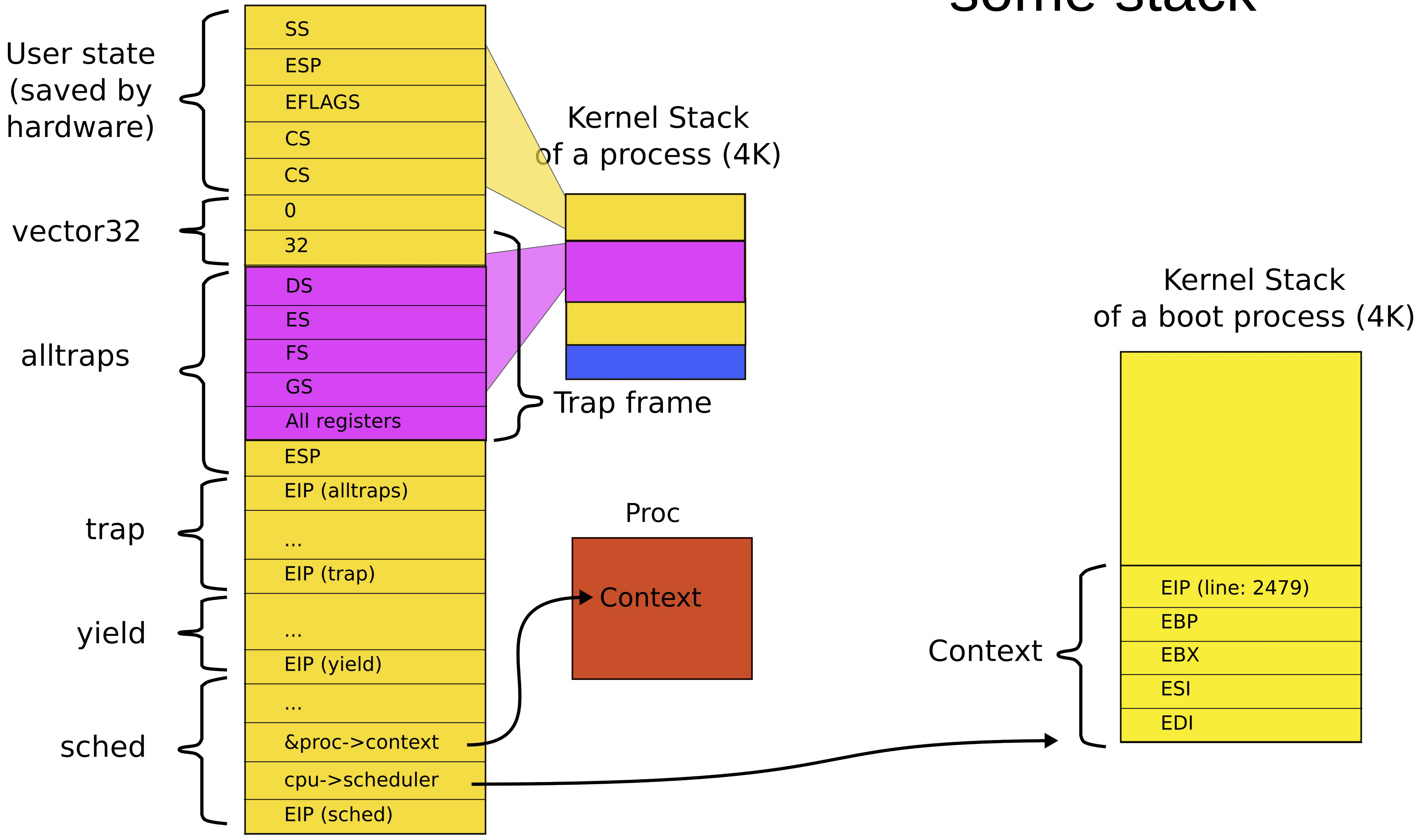
Proc

Context

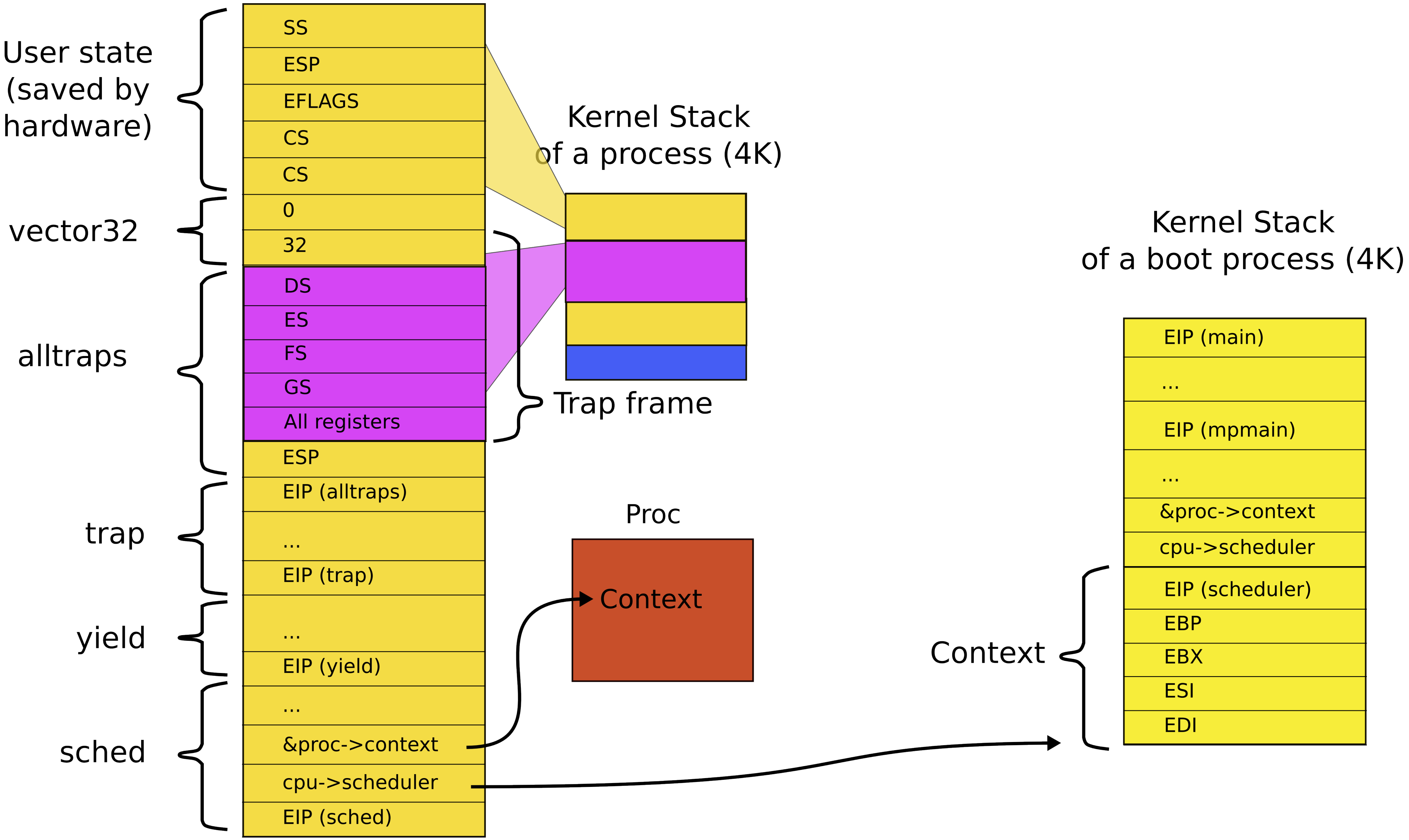
Context

- EIP (line: 2479)
- EBP
- EBX
- ESI
- EDI

Context is always top of some stack



We switch to the scheduler, it runs on the stack of the boot process



swtch()

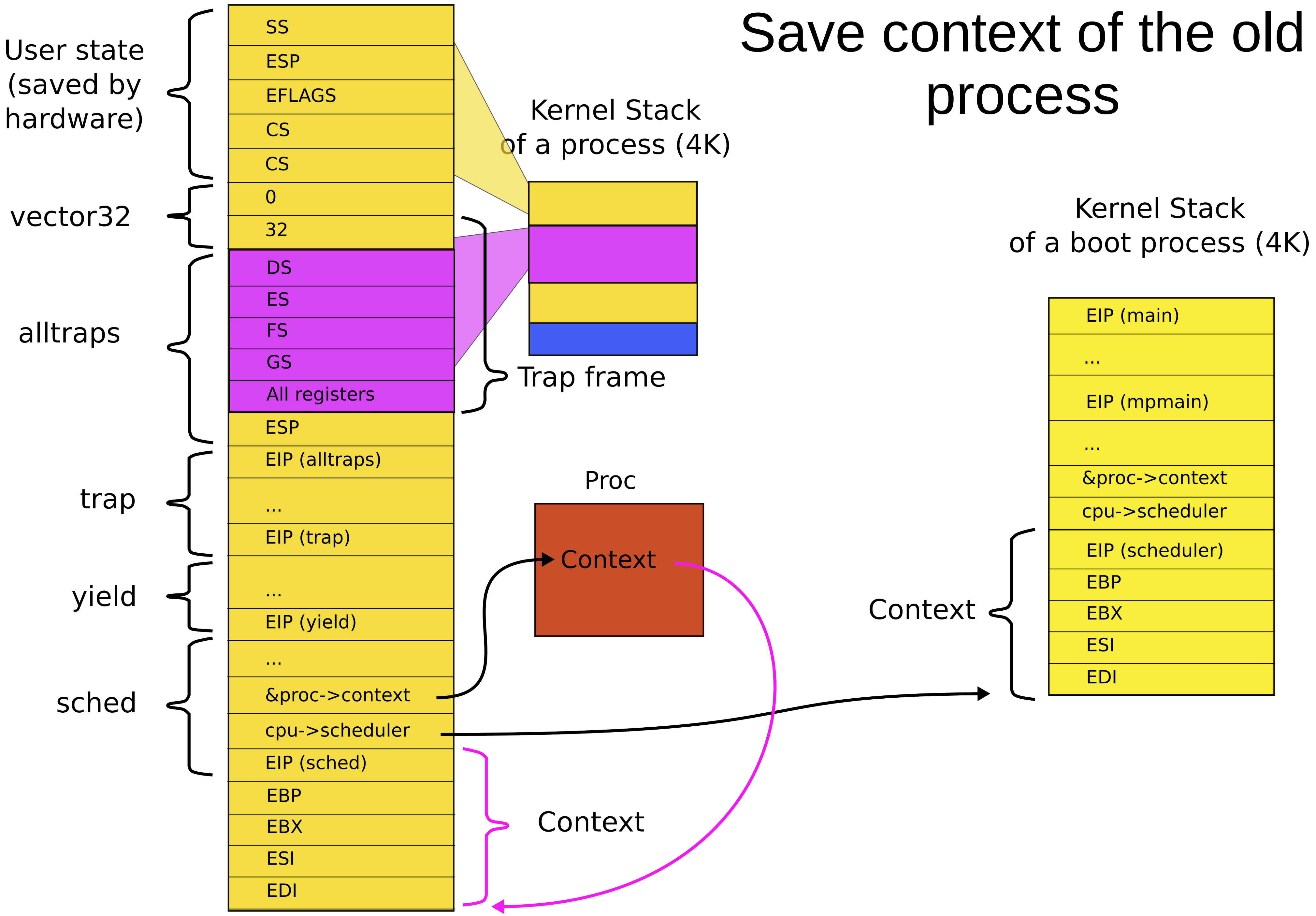
```
.globl swtch
swtch:
movl 4(%esp), %eax
movl 8(%esp), %edx

# Save old callee-save registers
pushl %ebp
pushl %ebx
pushl %esi
pushl %edi

# Switch stacks
movl %esp, (%eax)
movl %edx, %esp

# Load new callee-save registers
popl %edi
popl %esi
popl %ebx
popl %ebp
ret
```

Save context of the old process



Where does this swtch() return?

Where does this swtch() return?

- Scheduler
  - Remember, this is how we created the first process

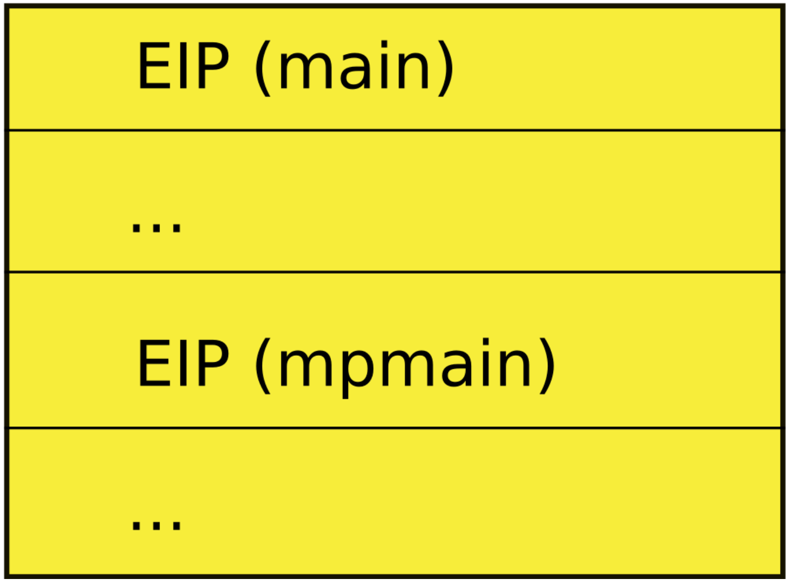
What does scheduler do?

```
scheduler(void)
{
  for(;;){
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;
      proc = p;
      switchuvm(p);
      p->state = RUNNING;
      swtch(&cpu->scheduler, proc->context);
      switchkvm();
      proc = 0;
    }
  }
}
```

- Scheduler picks next process to run
- Enters swtch()

Remember the stack of the boot process?

Kernel Stack
of a boot process (4K)



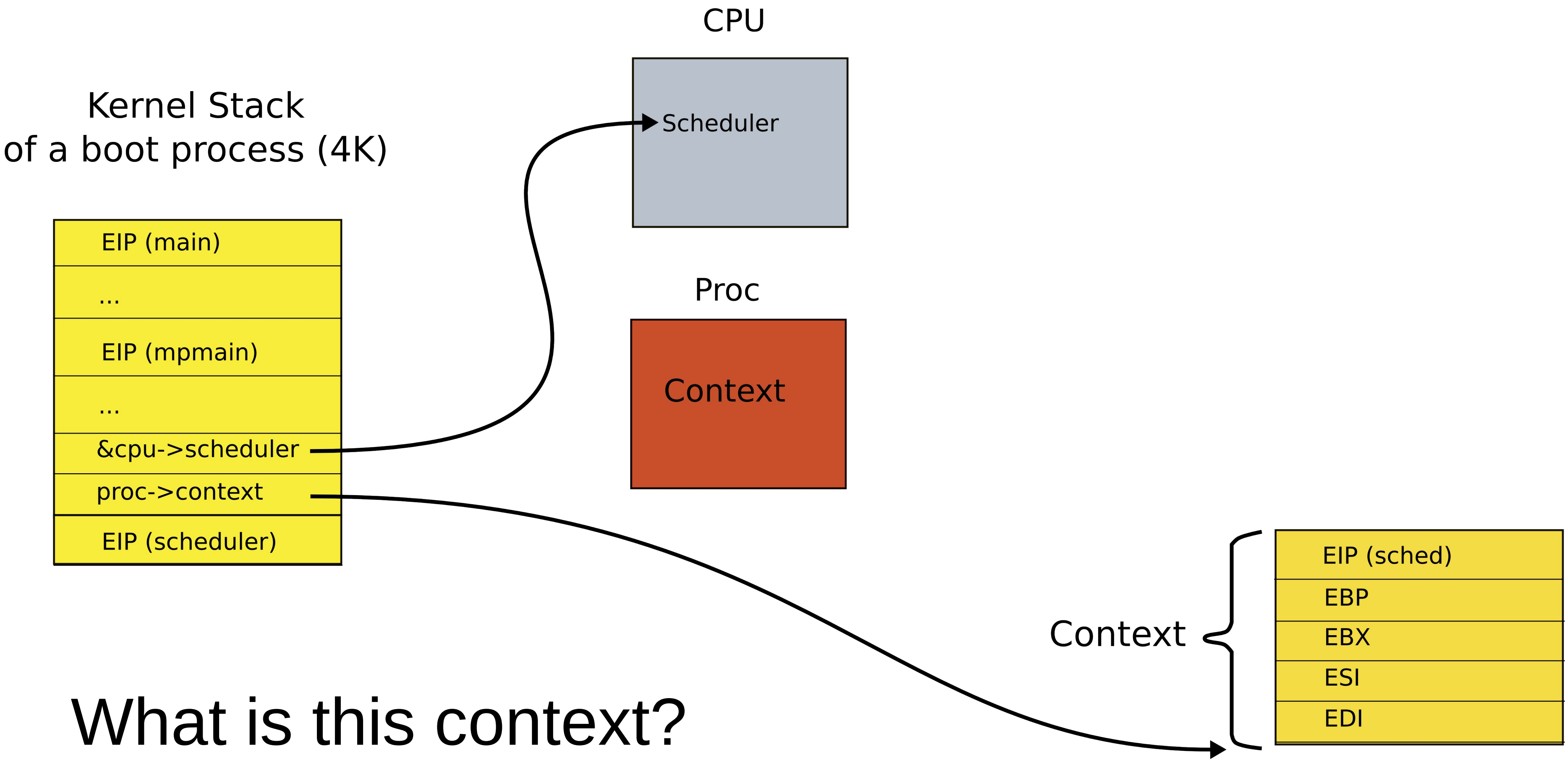
EIP (main)

...

EIP (mpmain)

...

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



Kernel Stack
of a boot process (4K)

EIP (main)
...
EIP (mpmain)
...
&cpu->scheduler
proc->context
EIP (scheduler)

CPU

Scheduler

Proc

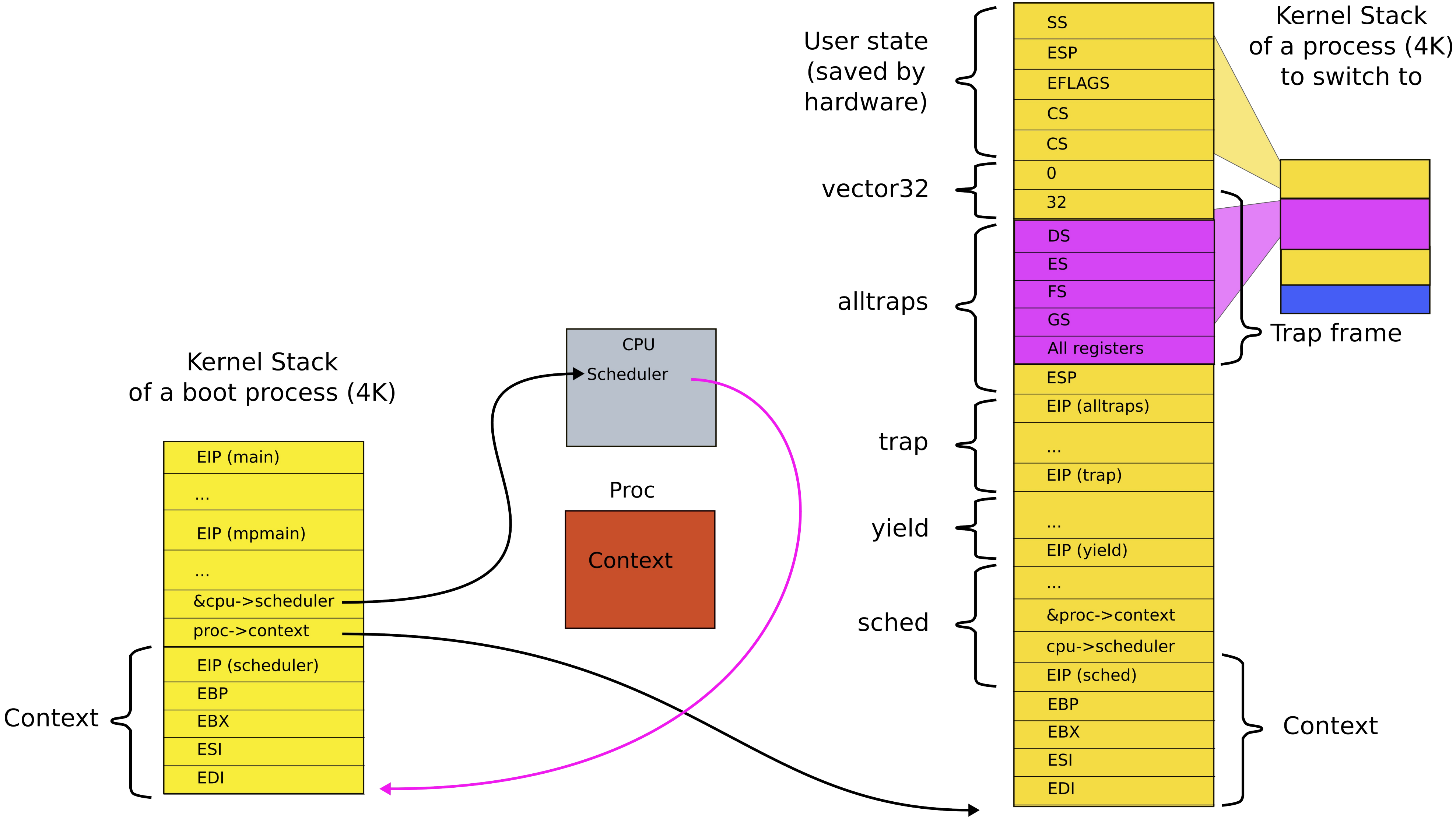
Context

Context

EIP (sched)
EBP
EBX
ESI
EDI

What is this context?

Stack inside swtch()



Kernel Stack
of a boot process (4K)

EIP (main)
...
EIP (mpmain)
...
&cpu->scheduler
proc->context
EIP (scheduler)
EBP
EBX
ESI
EDI

Context

CPU
Scheduler

Proc
Context

User state
(saved by
hardware)

vector32

alltraps

trap

yield

sched

SS
ESP
EFLAGS
CS
CS
0
32
DS
ES
FS
GS
All registers
ESP
EIP (alltraps)
...
EIP (trap)
...
EIP (yield)
...
&proc->context
cpu->scheduler
EIP (sched)
EBP
EBX
ESI
EDI

Kernel Stack
of a process (4K)
to switch to

Trap frame

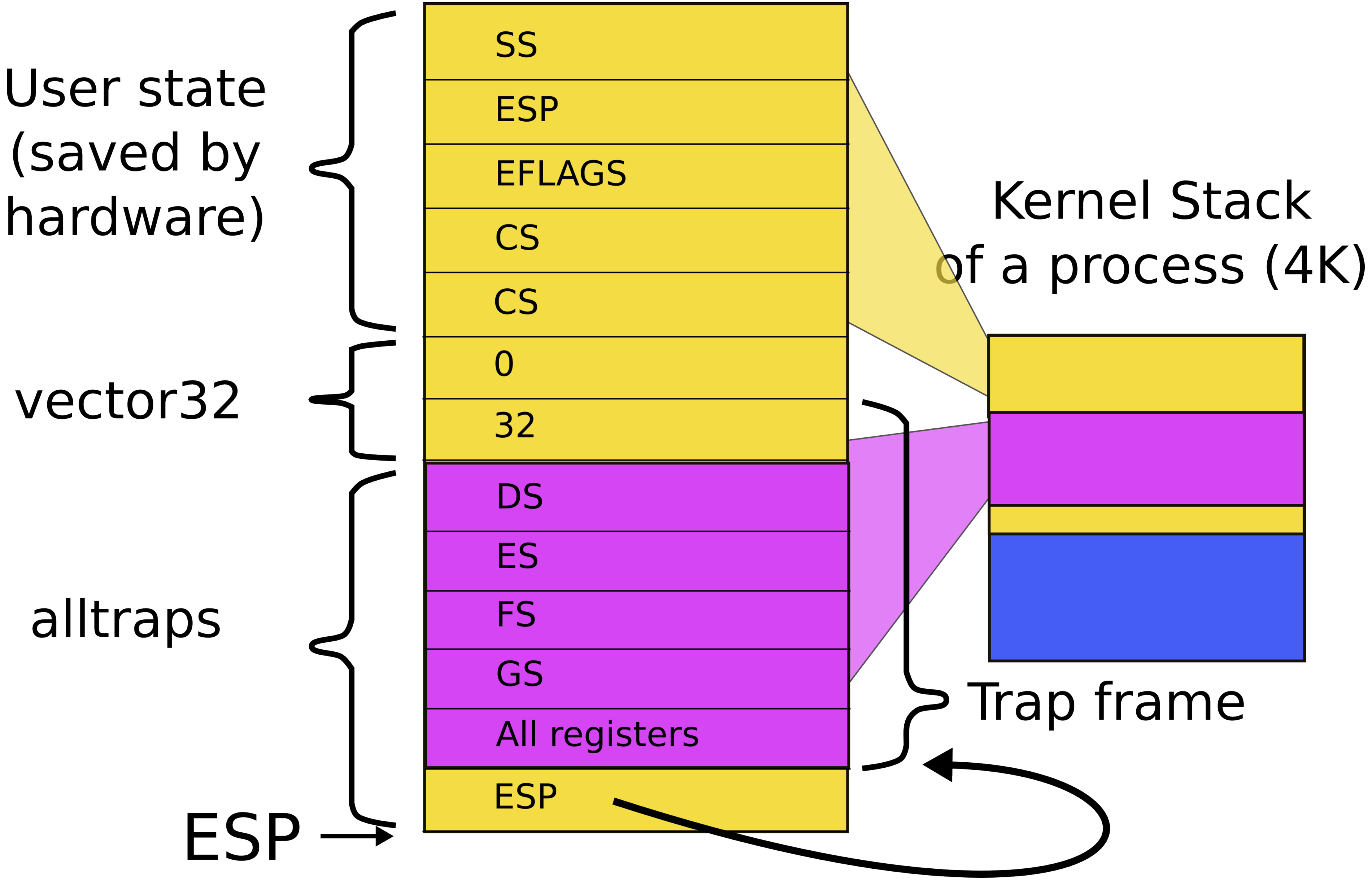
Context

alltraps()

```
alltraps:
...
# Call trap(tf), where tf=%esp
pushl %esp
call trap
addl $4, %esp

# Return falls through to trapret...
.globl trapret
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $0x8, %esp # trapno and errcode
iret
```

Stack inside swtch()



User state (saved by hardware)

SS

ESP

EFLAGS

CS

CS

vector32

0

32

alltraps

DS

ES

FS

GS

All registers

ESP

ESP →

Kernel Stack of a process (4K)

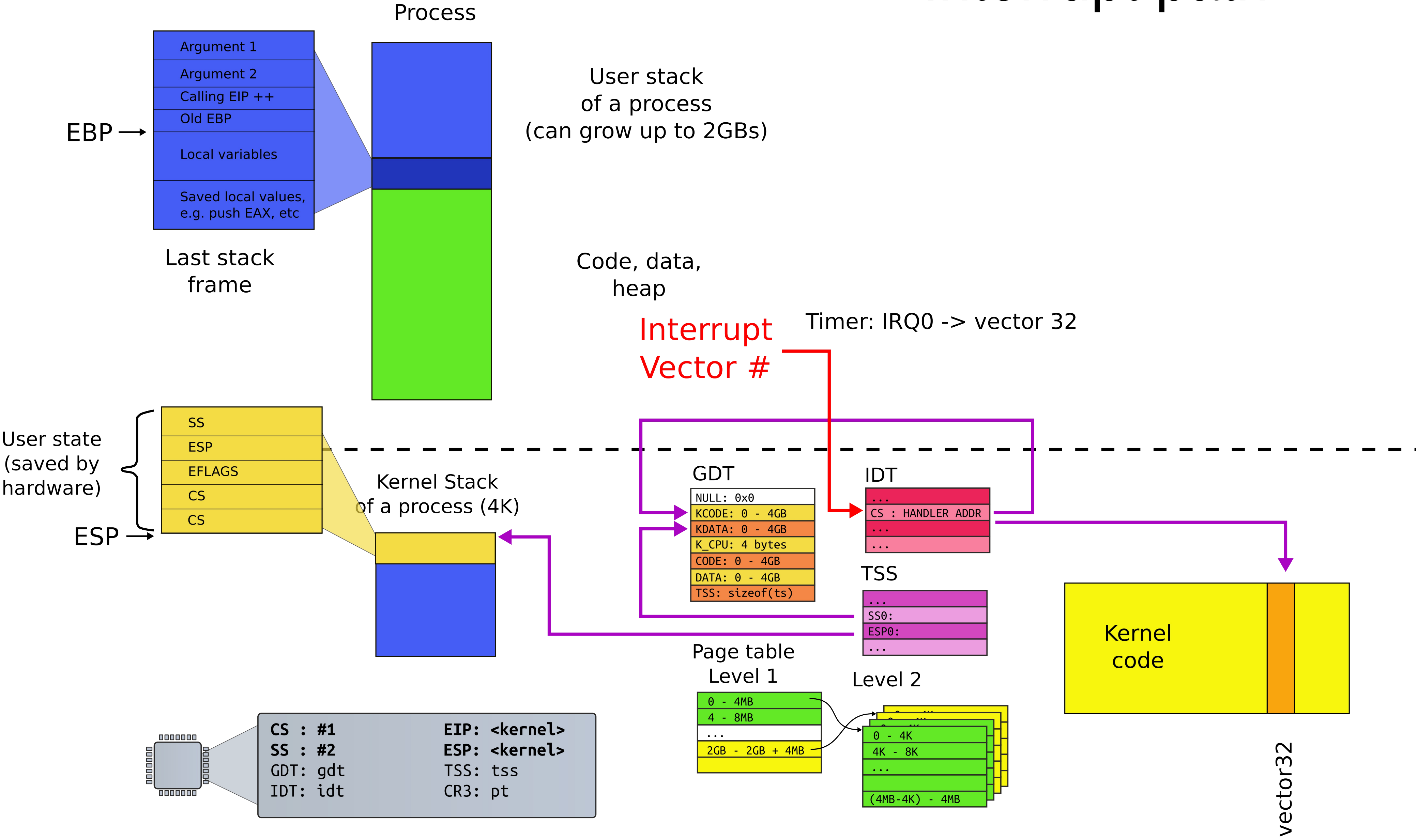
Trap frame

alltraps()

```
alltraps:
...
# Call trap(tf), where tf=%esp
pushl %esp
call trap
addl $4, %esp

# Return falls through to trapret...
.globl trapret
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $0x8, %esp # trapno and errcode
iret
```

Interrupt path



Summary

- We switch between processes now

Thank you!