Page Fault Liberation Army

Sergey Bratus
Julian Bangert

Trust Lab
Dartmouth College

“No instructions were harmed in the making of this talk”

Disclaimer

- ***Turing complete*** it's just a way of describing what kind of computations an environment can be programmed to do (T.-c. = any kind we know, in theory)
- Wish we had a more granular scale better suited to exploit power

Today's Slogan

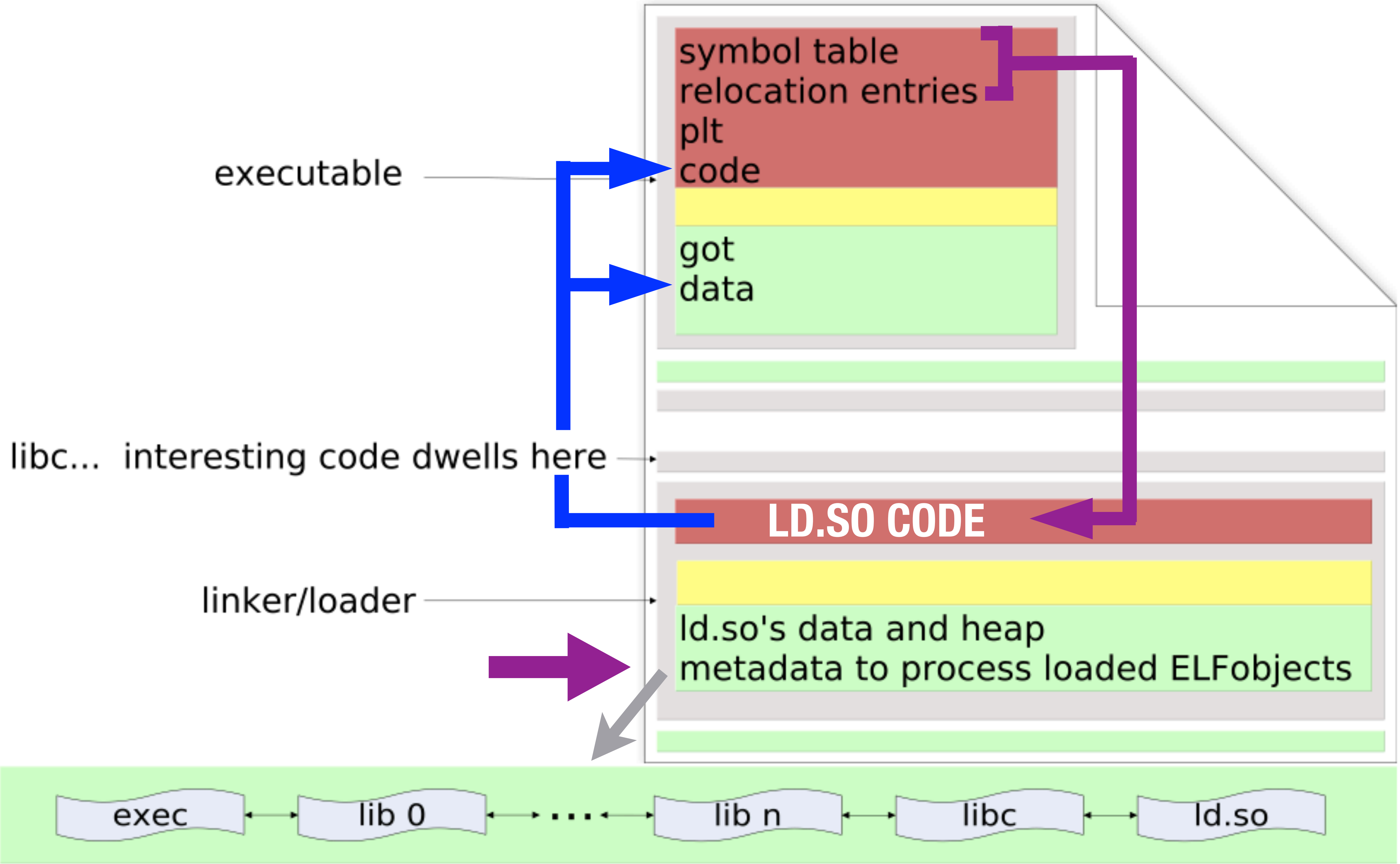
Any input is a program.

Any sufficiently complex input is indistinguishable from byte code;
any code that takes complex inputs is indistinguishable from a VM.

Intro Example:
ABI Metadata Machines

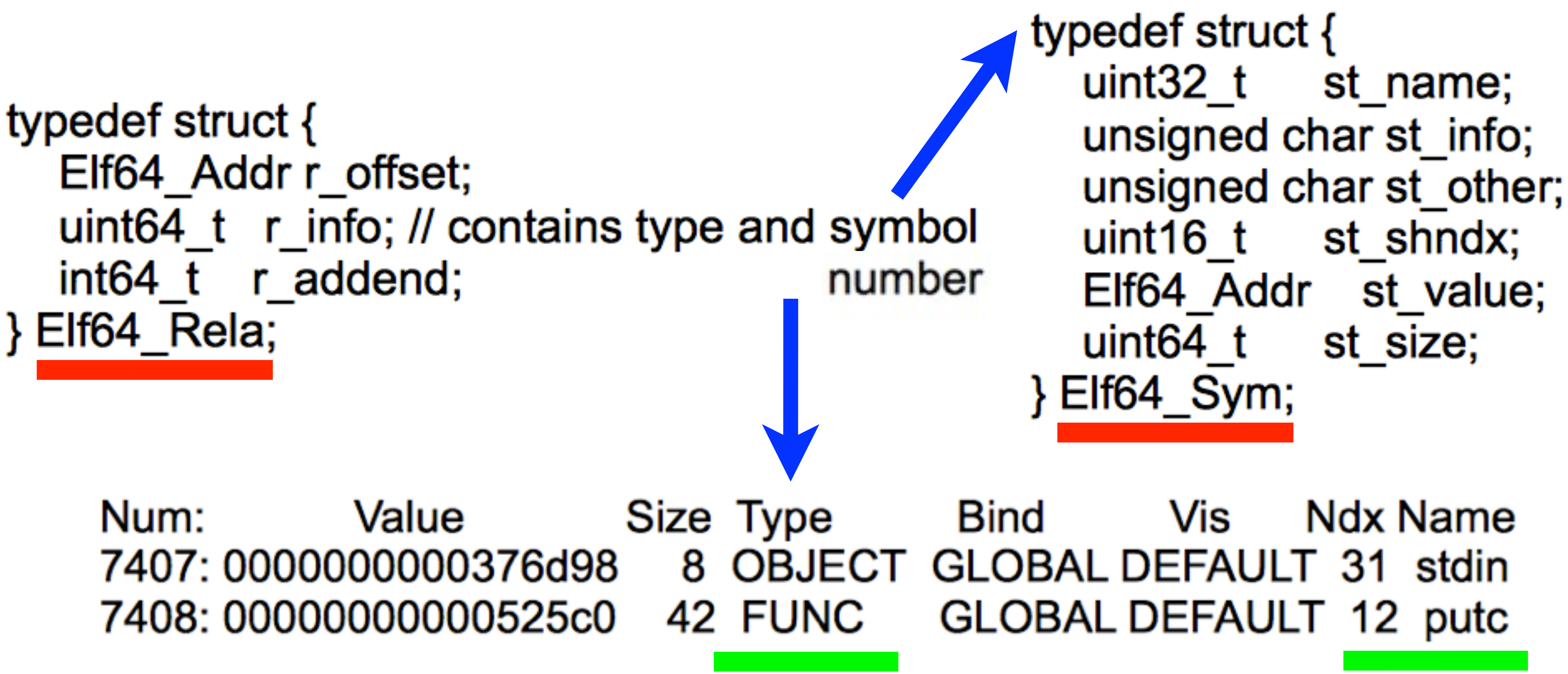
Sarah Inteman/John Kiehl

ELF relocation machine



ELF metadata machines

Relocations + symbols:
a **program** in ABI for automaton to patch images loaded at a different virtual address:



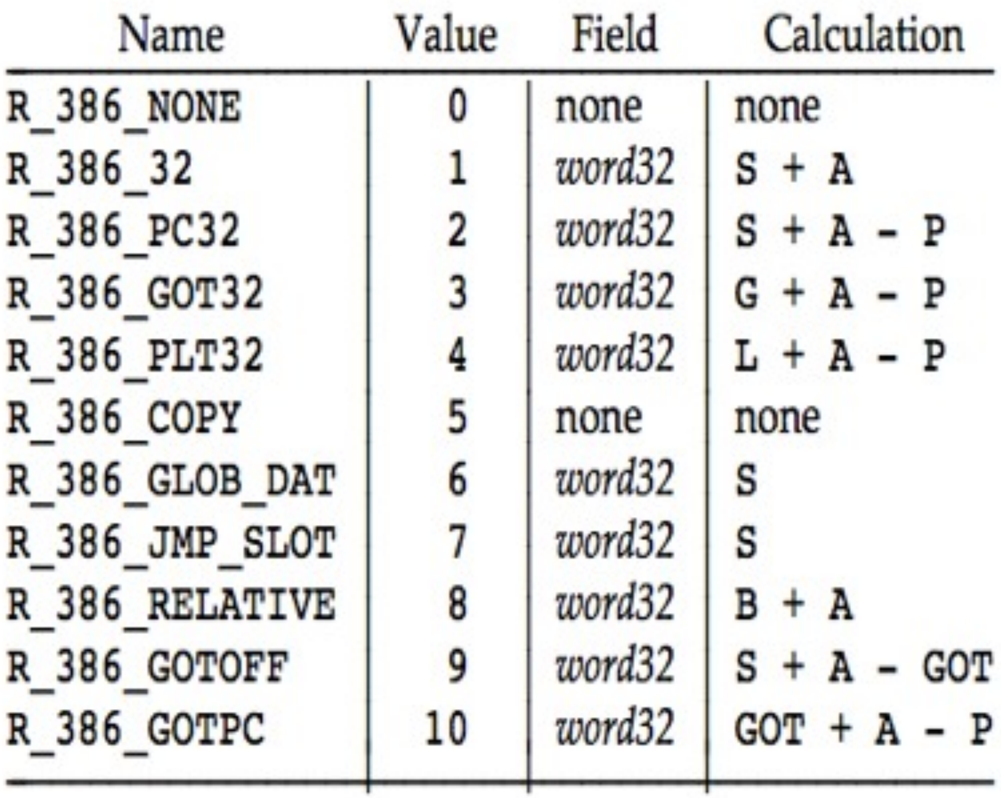
```
typedef struct {
    Elf64_Addr r_offset;
    uint64_t   r_info; // contains type and symbol
    int64_t    r_addend;                      number
} Elf64_Rela;
```

```
typedef struct {
    uint32_t      st_name;
    unsigned char st_info;
    unsigned char st_other;
    uint16_t      st_shndx;
    Elf64_Addr     st_value;
    uint64_t      st_size;
} Elf64_Sym;
```

```
Num:           Value          Size  Type     Bind    Vis      Ndx Name
7407: 0000000000376d98     8  OBJECT  GLOBAL DEFAULT  31  stdin
7408: 00000000000525c0    42  FUNC    GLOBAL DEFAULT  12  putc
```

Relocation arithmetic:

```
typedef struct {
   Elf64_Addr r_offset;
   uint64_t   r_info; // contains type and symbol
                                           number
   int64_t    r_addend;
} Elf64_Rela;
```



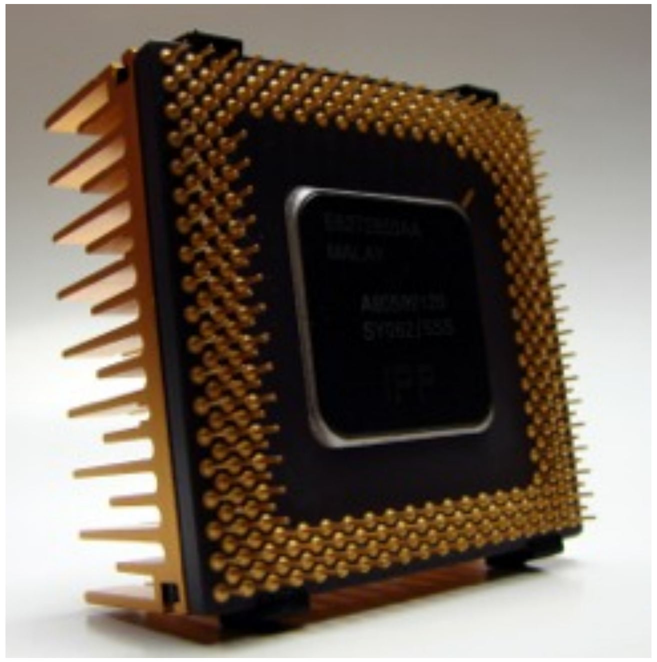
Name	Value	Field	Calculation
R_386_NONE	0	none	none
R_386_32	1	*word32*	S + A
R_386_PC32	2	*word32*	S + A - P
R_386_GOT32	3	*word32*	G + A - P
R_386_PLT32	4	*word32*	L + A - P
R_386_COPY	5	none	none
R_386_GLOB_DAT	6	*word32*	S
R_386_JMP_SLOT	7	*word32*	S
R_386_RELATIVE	8	*word32*	B + A
R_386_GOTOFF	9	*word32*	S + A - GOT
R_386_GOTPC	10	*word32*	GOT + A - P

```
  R_X86_64_COPY:
memcpy(r.r_offset, s.st_value, s.st_size)
  R_X86_64_64:
*(base+r.r_offset) = s.st_value +
                     r.r_addend + base
  R_X86_64_RELATIVE:
*(base+r.r_offset) = r.r_addend+base
```

See 29c3 talk by Rebecca "**.bx**" Shapiro,
https://github.com/bx/elf-bf-tools

Example for Today:

Page Fault Liberation Army (PFLA)*



"Input is (still) a program!"

*) In the x86 manuals it stands for "Page Faulting Linear Address", but our version is more interesting

“Page Fault Liberation”

Let’s take an old and known thing...

99 7240-7

“Page Fault Liberation”

...and see how far we can make it can go!

“Page Fault Liberation”

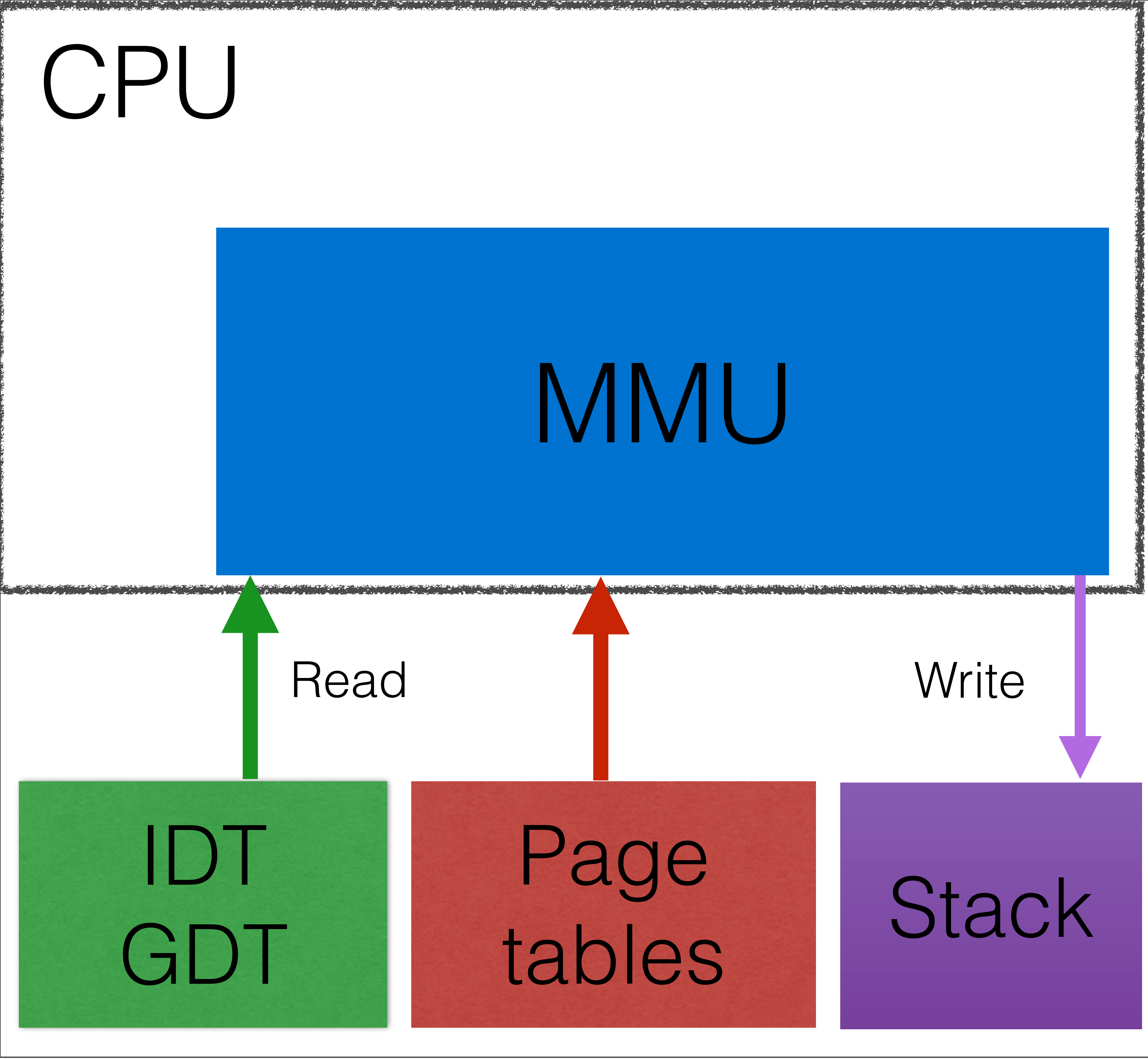
and perhaps others can take it further!

“Page Fault Liberation”

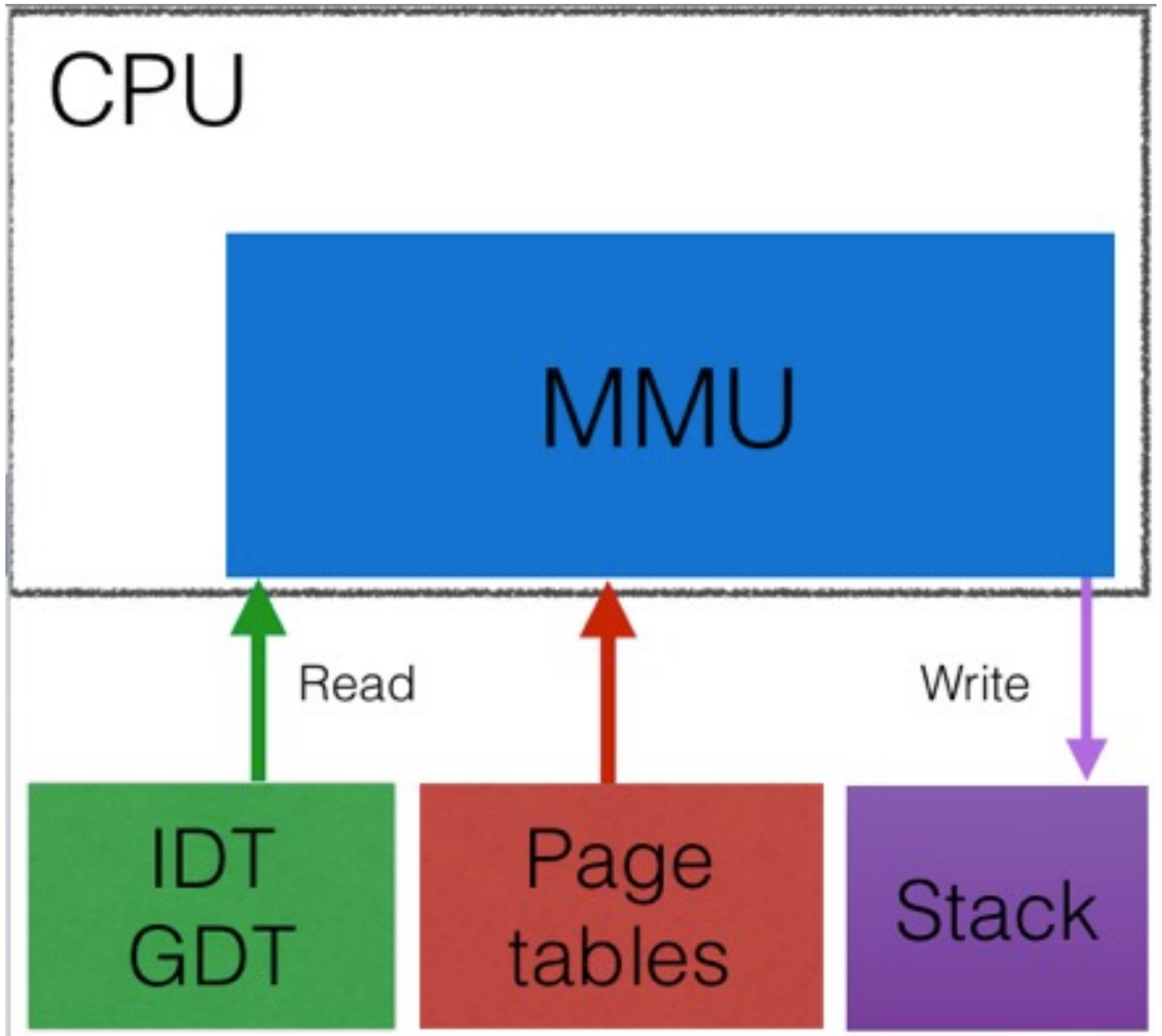
- The x86 MMU is not just a look-up table!
- x86 MMU performs complex logic on complex data structures
- The MMU has **state** and **transitions** that brilliant hackers put to unorthodox uses.
- Can it be **programmed** with its input data?

“Hacking is a practical study of computational models’ limits”

- [Apologies for repeating myself]
- “What Church and Turing did with theorems, hackers do with exploits”
- Great exploits (and effective defenses!) reveal truths about the target’s **actual** computational model.


CPU
MMU
Read
Write
IDT
GDT
Page tables
Stack

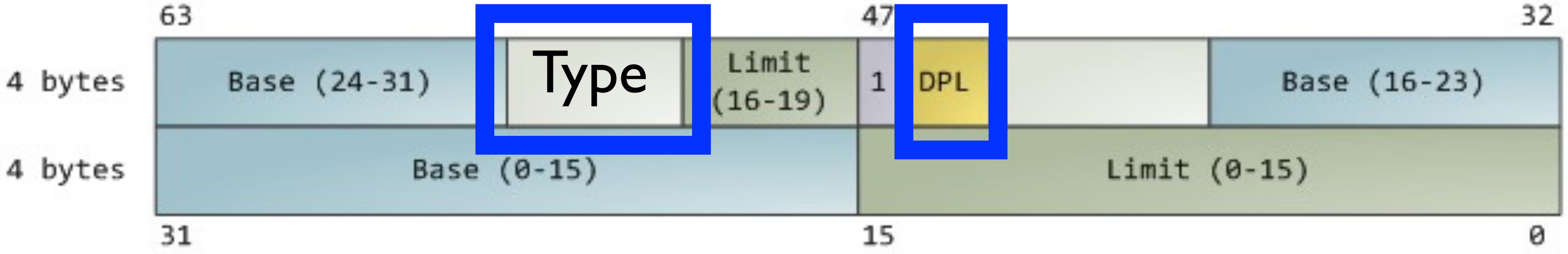
- unmapped/bad memory reference **trap**, based on **page tables** & (current) **IDT**
- hardware **writes fault info** on the stack - where it **thinks** the stack is (address in TSS)
- If we point "**stack**" into **page tables, GDT** or TSS, can we get the "tape" of a Turing machine?



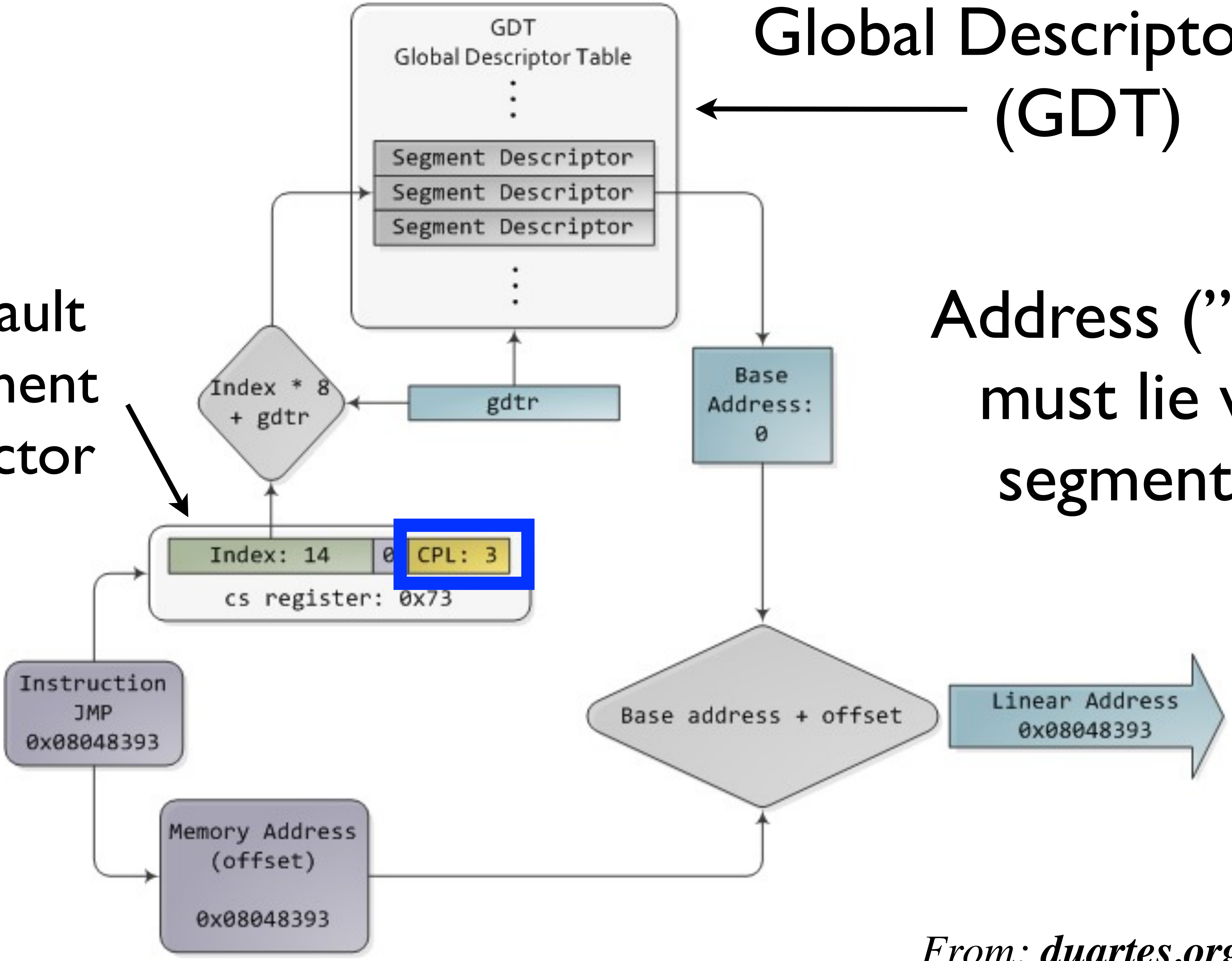
The devil's in the ~~details~~

trapping bits

Segment descriptor:



Global Descriptor Table (GDT)

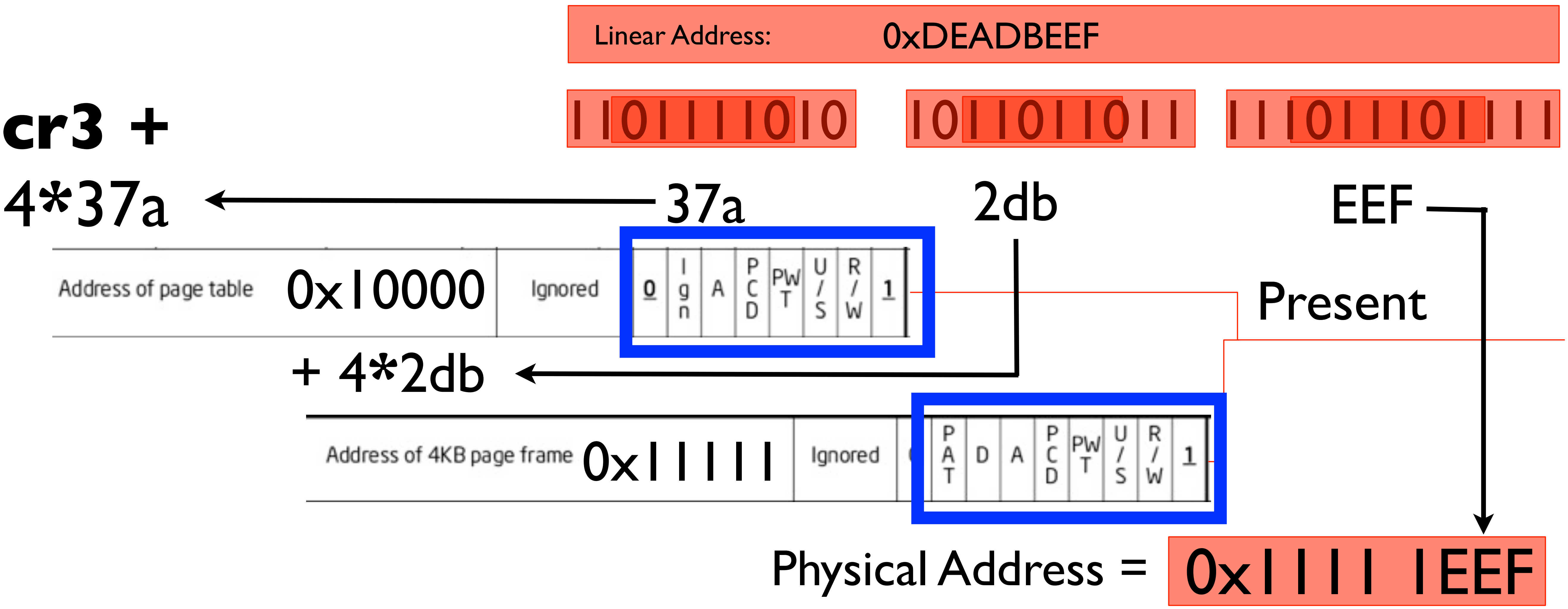


Default segment selector

Address ("offset") must lie within segment limit

From: ***duartes.org/gustavo/blog/***

Virtual Address Translation



Linear Address: 0xDEADBEEF

1101111010 1011011011 111011101111

cr3 + 4*37a

37a

2db

EEF

Address of page table 0x10000 Ignored 0 Ign A PCD PWT U/S R/W 1

Present

+ 4*2db

Address of 4KB page frame 0x11111 Ignored PAT D A PCD PWT U/S R/W 1

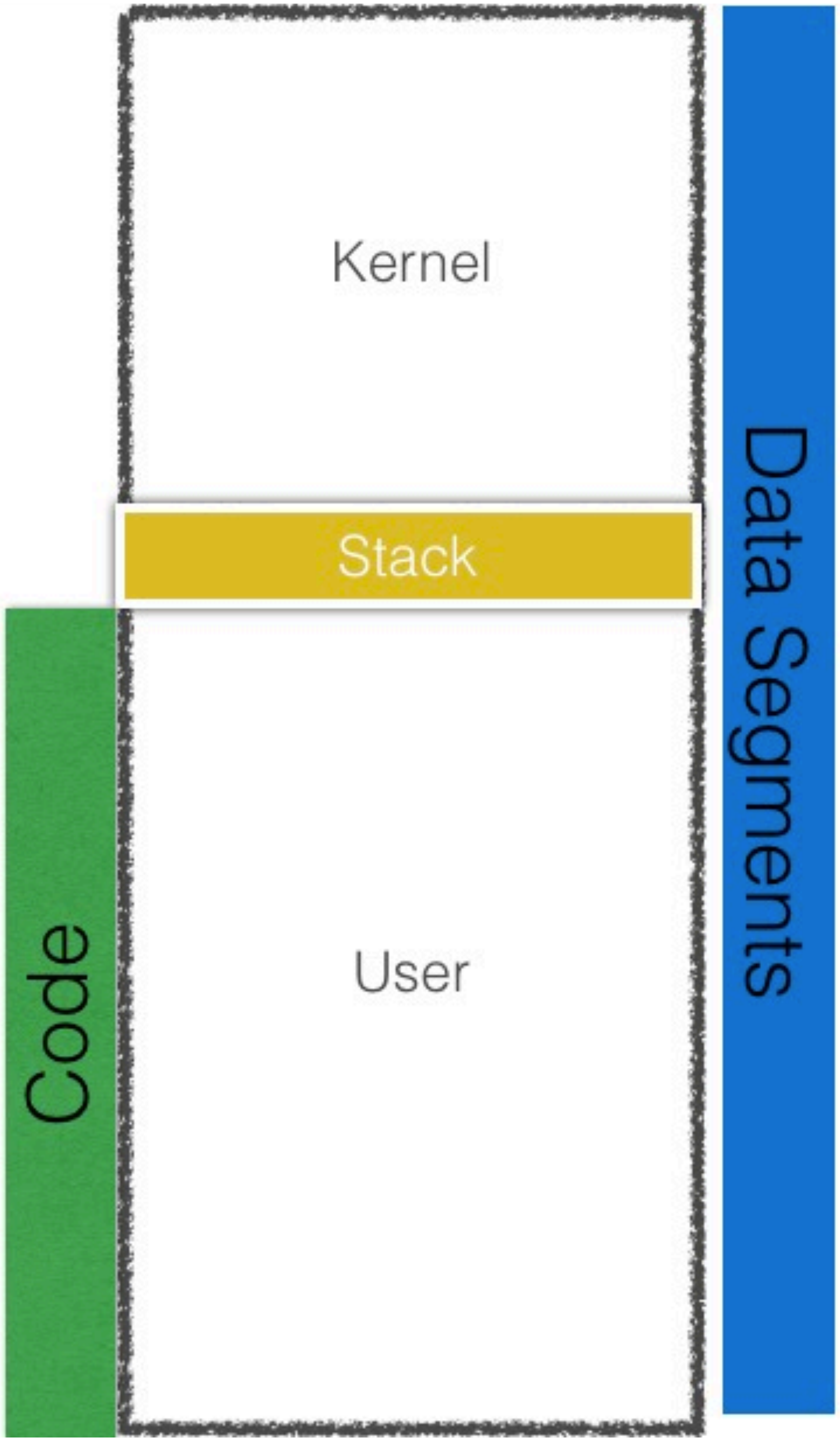
Physical Address = 0x1111 1EEF

- All **P** bits set
- Ring 3: All **U/S** bits have to be set
- Write: All **R/W** bits have to be set
- What if we violate these rules?

ITS A TRAP

OpenWall

- Solar Designer, 1999
  - cf. "Stack Smashing for Fun and Profit"
- **CS limit** is 3GB - 8**MB** (for stack)
- Code **fetch** from the stack is trapped
- See if the current instruction is a **RET**
- Very specific threat, allows JIT, etc.
- (And many other hardening patches)



PaX

- PaX is an awesome Linux hardening patch
- Many 'firsts' on real-world OS's, e.g. **NX** on Intel and ASLR (PaX in 2000, OpenBSD in 2003)
- PaX has **NX** on all CPUs since the Pentium (Intel has hardware support since P4)
    - SEGMEXEC and PAGEEXEC
    - Leverages difference between instruction and data memory paths

PaX **NX**: SegmExec

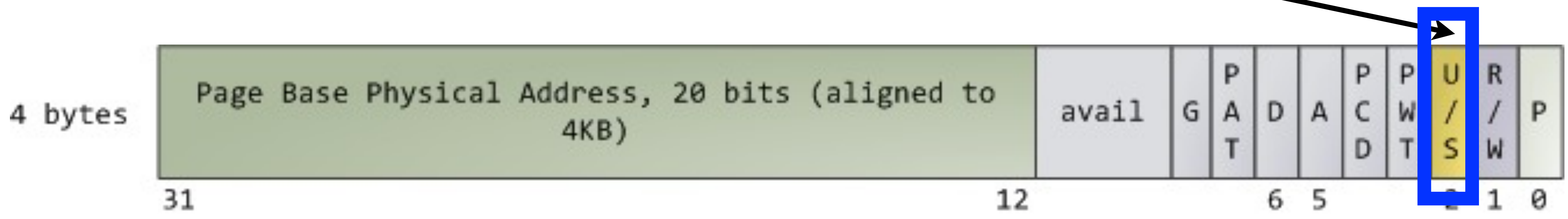
- Instruction: Virtual address = Linear + CS.base
- Data: VA= Linear + {DS,ES,FS,GS,SS}.base
- 3GB user space
- Set all segment limits to 1.5 GB (so all pointers are less than 1.5GB)
- Data access goes to lower half of VA space
- Instruction fetch goes to upper half of VA space

PaX **NX**: PageExec

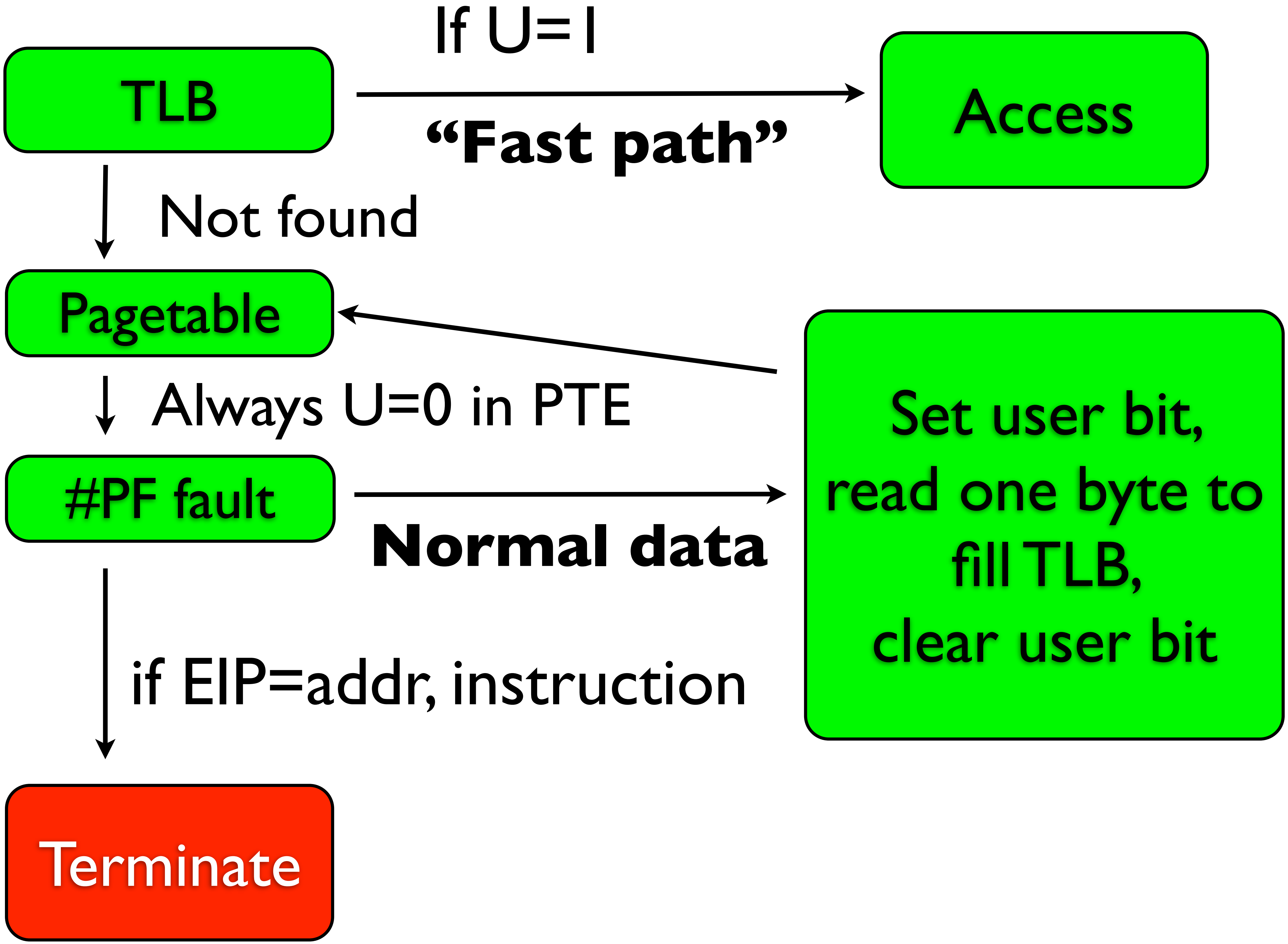
- "**split TLB**" (iTLB for fetches, dTLB for loads) [Plex86 1997, to detect self-modifying code: http://pax.grsecurity.net/docs/pageexec.old.txt]

- TLBs are **not** synchronized with page tables in RAM (manually flushed every time tables change)

- NX ~ User/Supervisor bit



PageExec data lookup



TLB

If U=1

Access

“Fast path”

Not found

Pagetable

Always U=0 in PTE

#PF fault

Normal data

Set user bit, read one byte to fill TLB, clear user bit

if EIP=addr, instruction

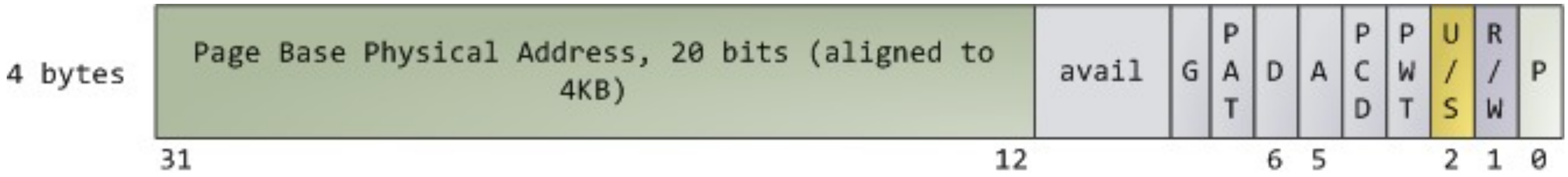
Terminate

OllyBone:
Trap on end of unpacker

- Same TLB technique as PaX
- Debugger plugin to analyze (un)packers
- Want to break execution on a memory range (so you trap every time you exec after writing)
- The idea goes back to Plex86 (before PaX) who tried to do virtualization that way

ShadowWalker

- When a rootkit detector **scans** the code (as **data**!), why not give a different page than when the code is executed?
- Instead of having different User bits, we could also have different **page frame numbers** (trap on P=0 in pagetables)



Trap-based “Design Patterns”

- **Overloading #PF** for security policy, labeling memory (e.g., PaX, OpenWall)
- **Combining** traps to trap on more complex events (OllyBone, “fetch from a page just written”)
- Using **several** trap bits in different locations to label memory for **data flow** control (PaX UDEREF, SMAP/SMEP use)
- Storing **extra state** in TLBs (PaX PageExec)
- “Unorthodox” breakpoints, control flow, ...

What’s in a trap handler (let’s roll our own)

IDT entries:

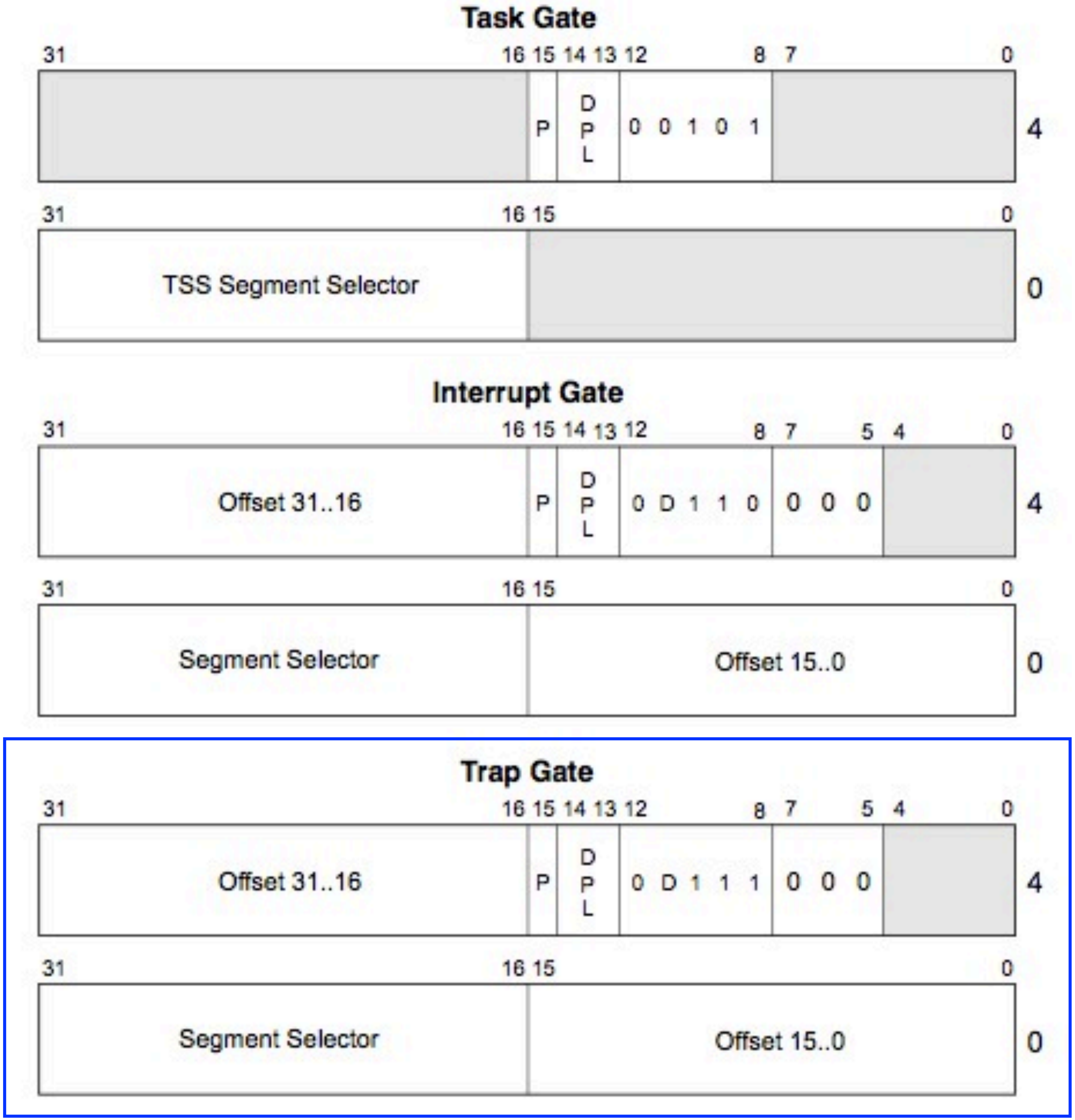
...

8: #DF

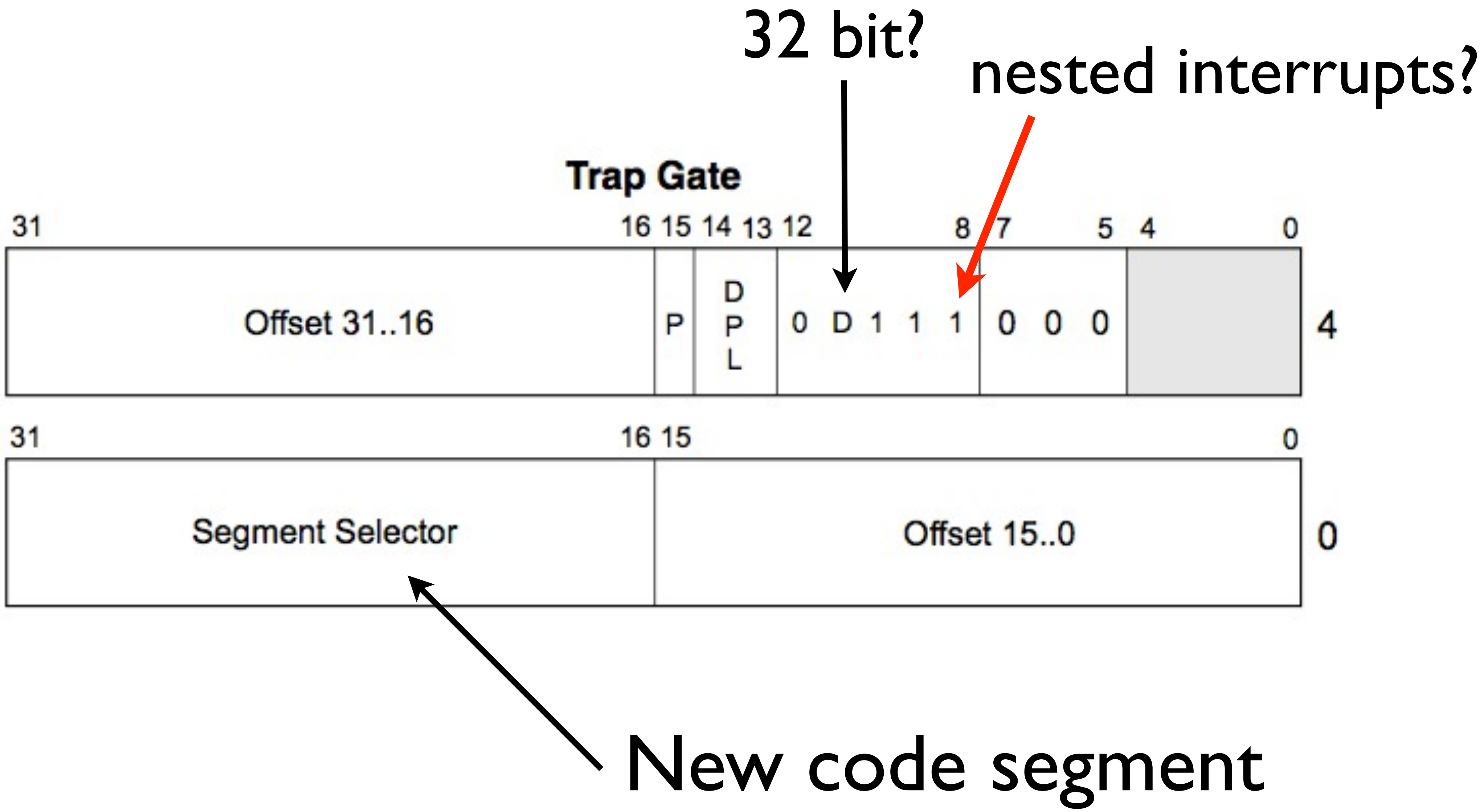
...

14: #PF

...



Call through a Trap Gate



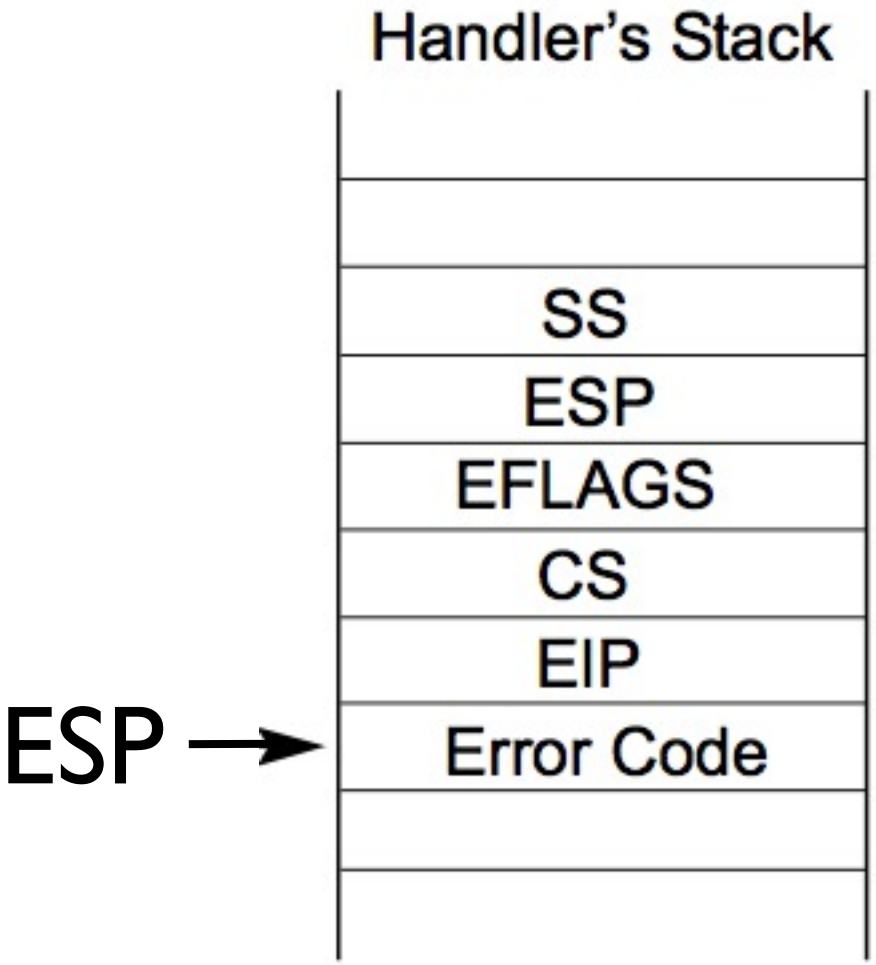
32 bit?

nested interrupts?

New code segment

Like a FAR call of old. If the new segment is in a lower (i.e. higher privilege) Ring, we load a new SP.

Pushes parameters to “handler’s stack”



Handler’s Stack

SS

ESP

EFLAGS

CS

EIP

Error Code

ESP

These two are only pushed if we changed the stack

“IRET” instruction can return from this

What if this fails?

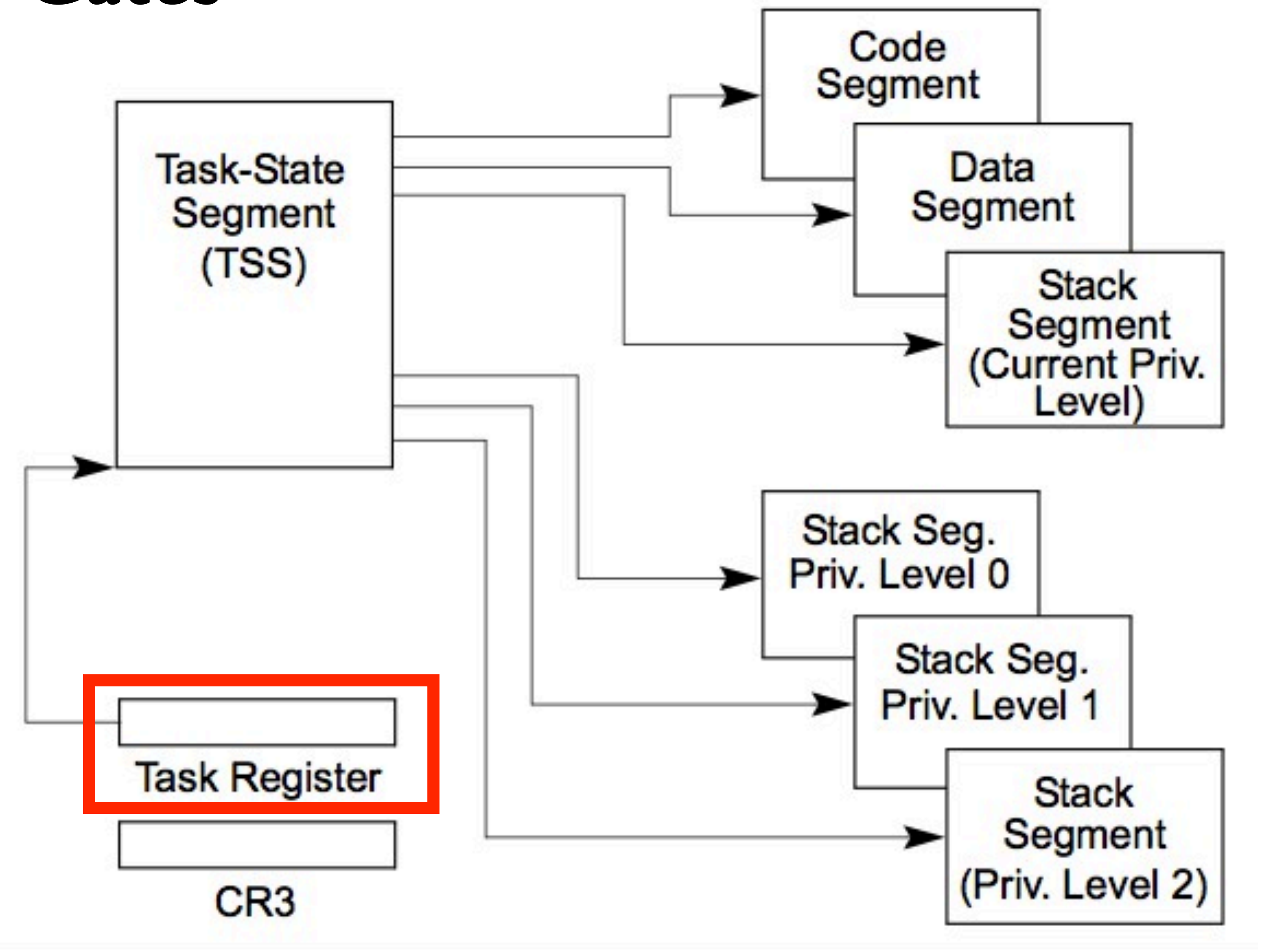
- Stack invalid?
- Code segment invalid?
- IDT entry not present?

Causes "**Double Fault**"(#8). "Triple fault" = Reboot

Usually DF means OS bug, so a lot of state might be corrupted (i.e. invalid kernel stack)

Hardware Task Switching

Can use it for #PF and #DF traps instead of Trap Gates

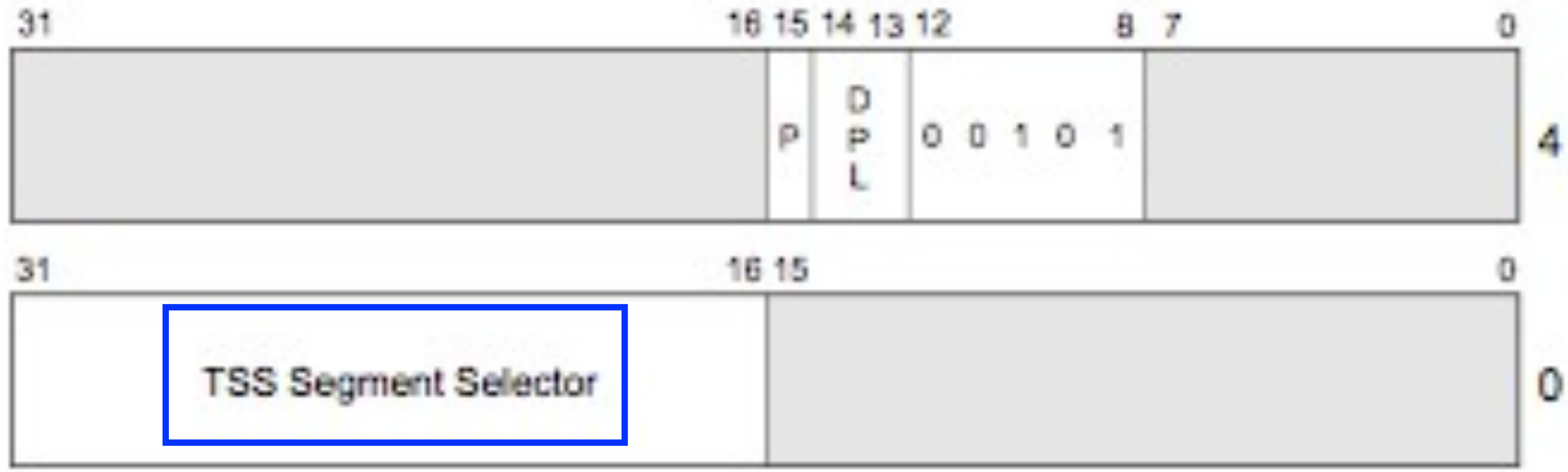


TR

Task gate

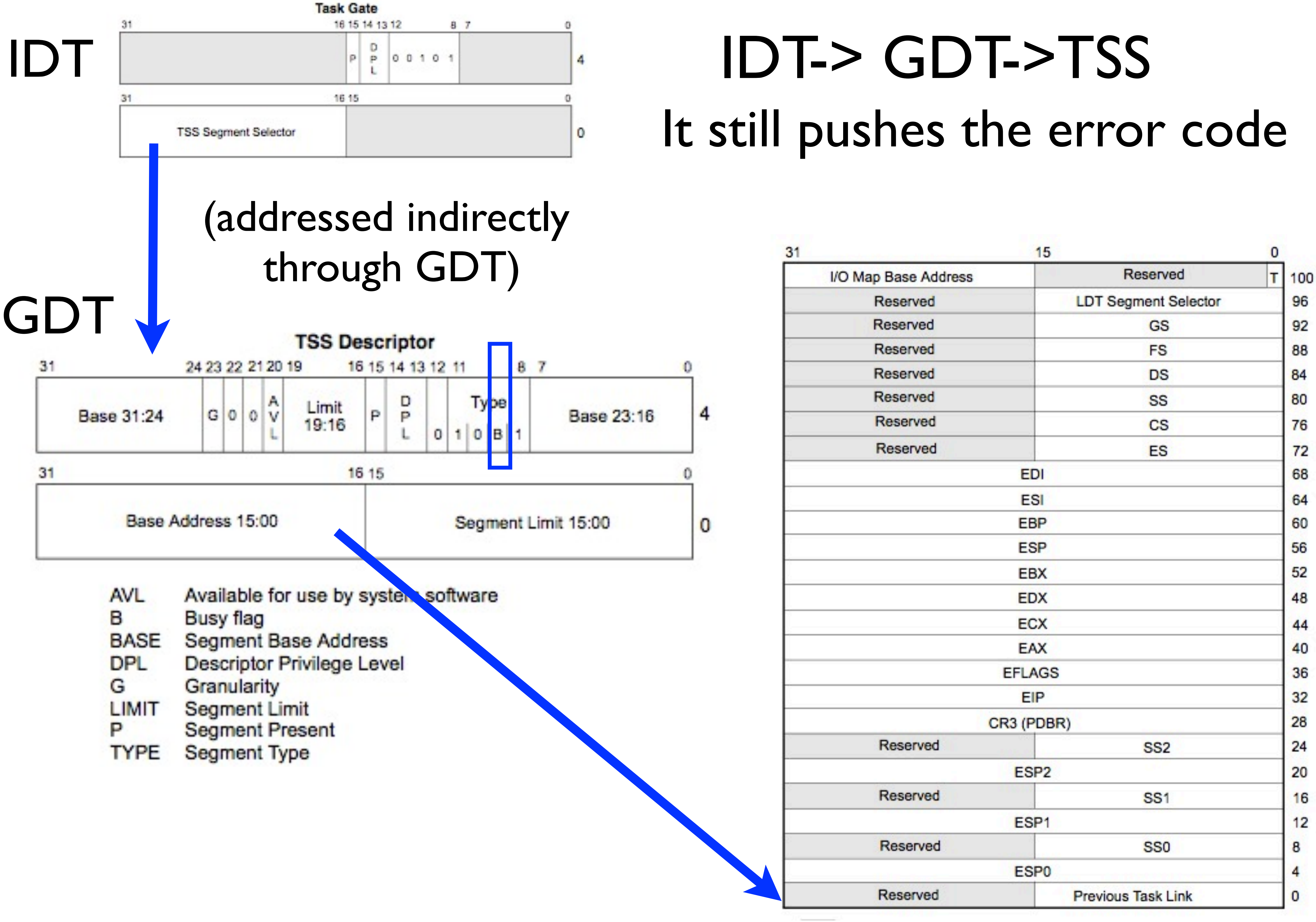
- (unused) mechanism for **hardware** tasking
- Reloads (nearly) all CPU state from memory
- Task gate causes **task switch** on **trap**

Task Gate



IDT-> GDT->TSS

It still pushes the error code



IDT

Task Gate

(addressed indirectly through GDT)

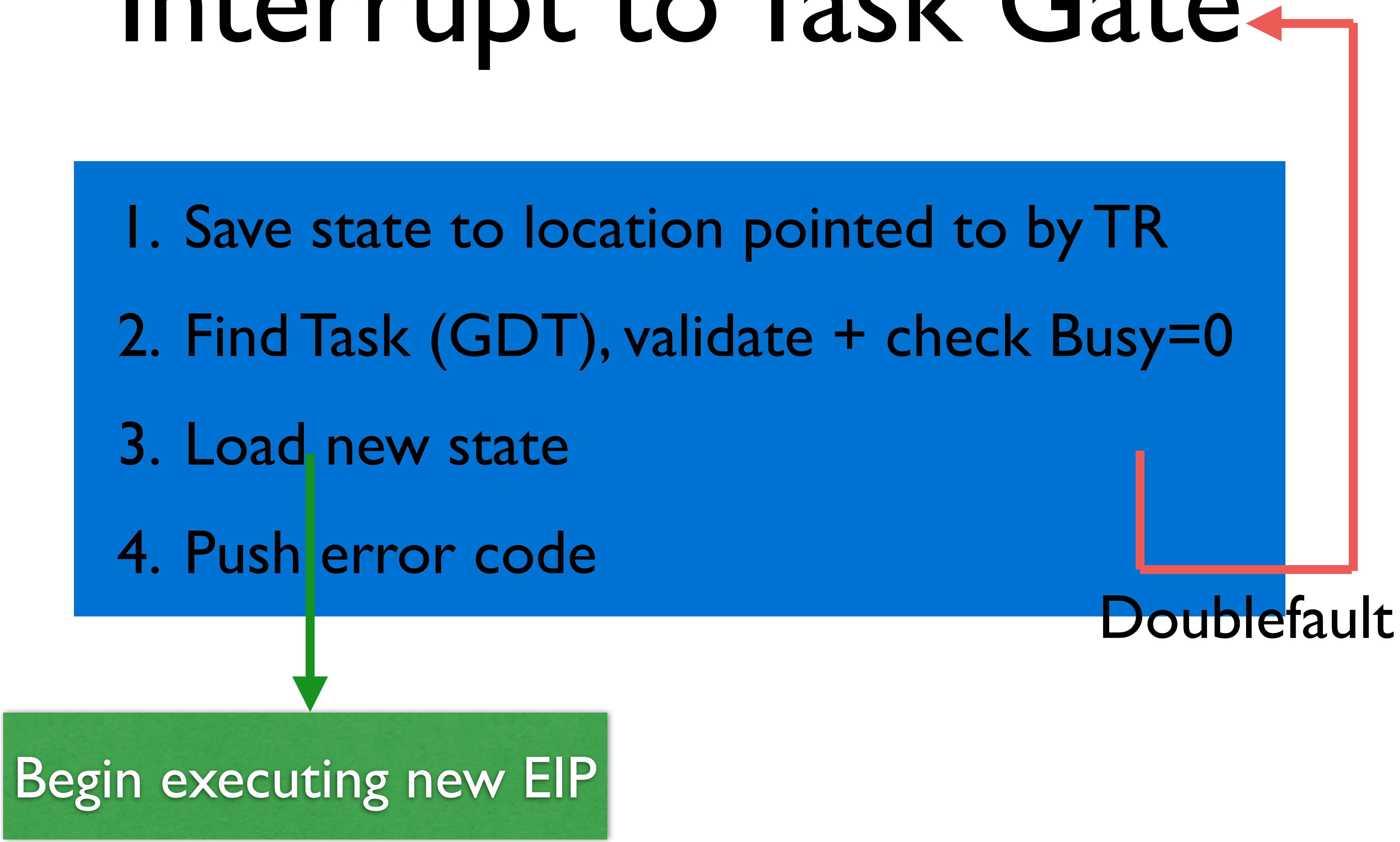
GDT

TSS Descriptor

AVL	Available for use by system software
B	Busy flag
BASE	Segment Base Address
DPL	Descriptor Privilege Level
G	Granularity
LIMIT	Segment Limit
P	Segment Present
TYPE	Segment Type

31	15	0
I/O Map Base Address	Reserved | T	100
Reserved	LDT Segment Selector	96
Reserved	GS	92
Reserved	FS	88
Reserved	DS	84
Reserved	SS	80
Reserved	CS	76
Reserved	ES	72
EDI		68
ESI		64
EBP		60
ESP		56
EBX		52
EDX		48
ECX		44
EAX		40
EFLAGS		36
EIP		32
CR3 (PDBR)		28
Reserved	SS2	24
ESP2		20
Reserved	SS1	16
ESP1		12
Reserved	SS0	8
ESP0		4
Reserved	Previous Task Link	0

Interrupt to Task Gate



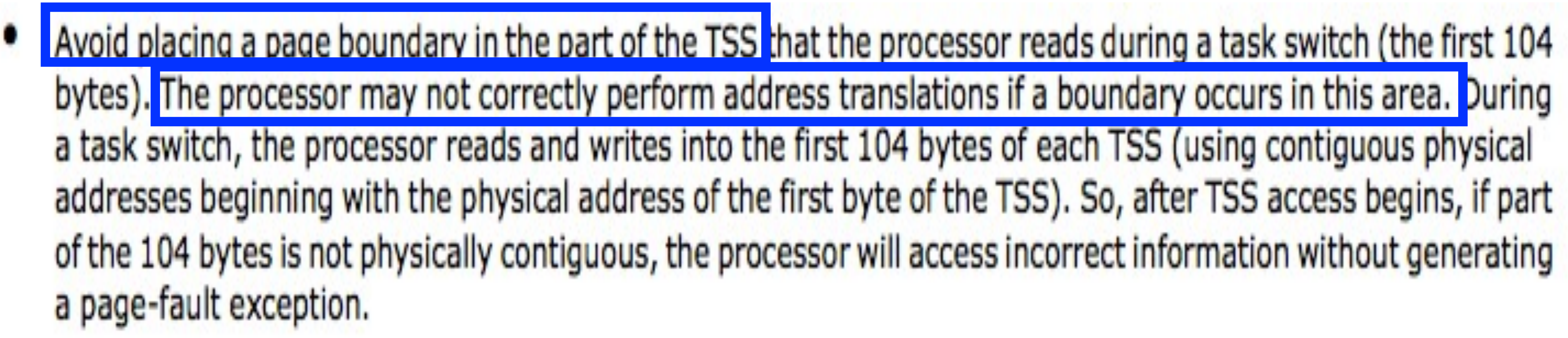
1. Save state to location pointed to by TR
2. Find Task (GDT), validate + check Busy=0
3. Load new state
4. Push error code

Doublefault

Begin executing new EIP

Brief digression

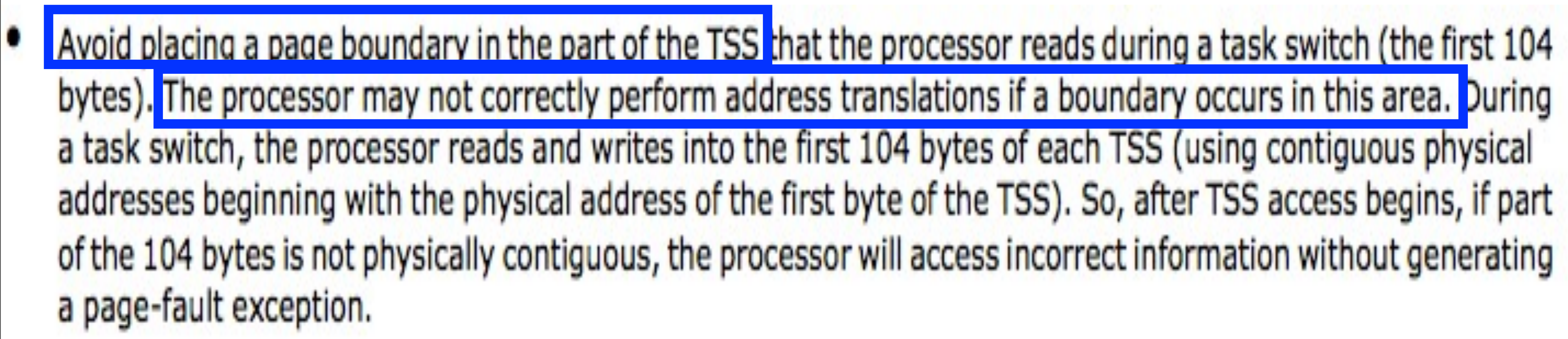
Intel Manual:



- Avoid placing a page boundary in the part of the TSS that the processor reads during a task switch (the first 104 bytes). The processor may not correctly perform address translations if a boundary occurs in this area. During a task switch, the processor reads and writes into the first 104 bytes of each TSS (using contiguous physical addresses beginning with the physical address of the first byte of the TSS). So, after TSS access begins, if part of the 104 bytes is not physically contiguous, the processor will access incorrect information without generating a page-fault exception.

Brief digression

Intel Manual:



- Avoid placing a page boundary in the part of the TSS that the processor reads during a task switch (the first 104 bytes). The processor may not correctly perform address translations if a boundary occurs in this area. During a task switch, the processor reads and writes into the first 104 bytes of each TSS (using contiguous physical addresses beginning with the physical address of the first byte of the TSS). So, after TSS access begins, if part of the 104 bytes is not physically contiguous, the processor will access incorrect information without generating a page-fault exception.

Bypass (all) paging from the kernel?
VM Escape?
Wouldn’t that be nice?

Maybe we should actually verify it..

CPU translates DWORD by DWORD

(CC-BY-SA)Lizzie Bitty/DevianArt

Look Ma, it’s a machine!

A one-instruction machine

Instruction Format:
Label = (X <-Y,A,B)

```
Label:
  X=Y
  If X<4:
      Goto B
  Else
      X-=4
      Goto A
```

- “Decrement-Branch-If-Negative”
- Turing complete (!)
- ““Computer Architecture: A Minimalist Perspective” by Gilreath and Laplathe (~$200)
- Or Wikipedia :)

Implementation sketch:

- If EIP of a handler is pointed at invalid memory, we get another **page fault** immediately; keep EIP invalid in all tasks
- Var Decrement: use TSS' SP, pushing the stack decrements SP by 4.
- Branch: <4 or not? Implemented by **double fault** when SP cannot be decremented

Dramatis Personae I

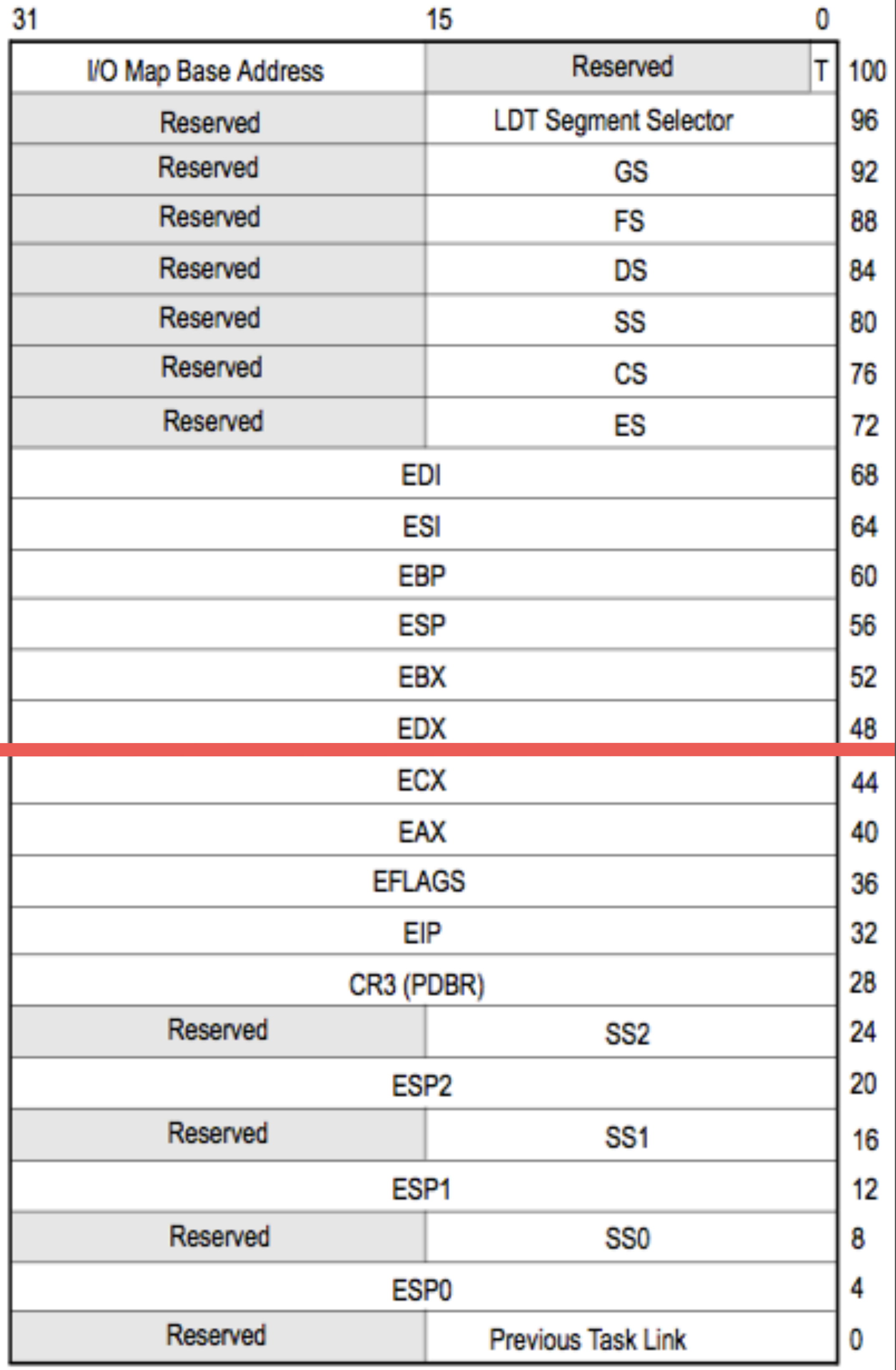
- One GDT to rule them all
- One TSS Descriptor per instruction, aligned with the end of a page
- IDT is mapped differently, per instruction
- A target (branch-not-taken) in Int 14, #PF
- B target (branch taken) in Int 8, #DF

Dramatis Personae II

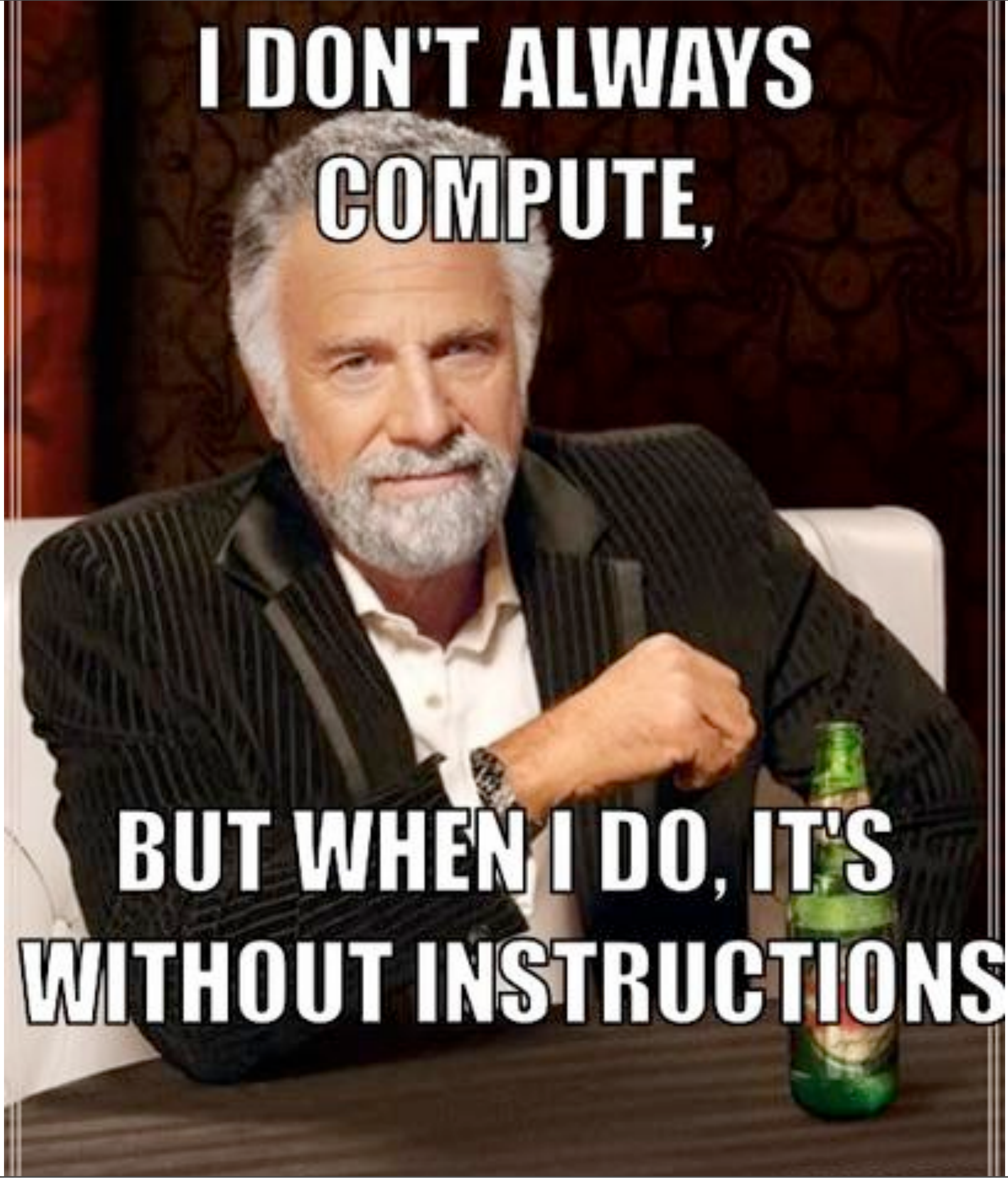
- Higher half of TSS (variables)
  - Map A.Y, B.Y (the value we want to load for next instruction) at their TSS addresses
  - map X (the value we want to write) at the addr of the current task
- So we have the move and decrement

- We split these TSS across a page boundary
- Variables are stack pointer entries in a TSS
- Upper Page: ESP and segments
- Lower Page: EAX, ECX, EIP, CR3 (page tables)

Labels: A, B, C, ...

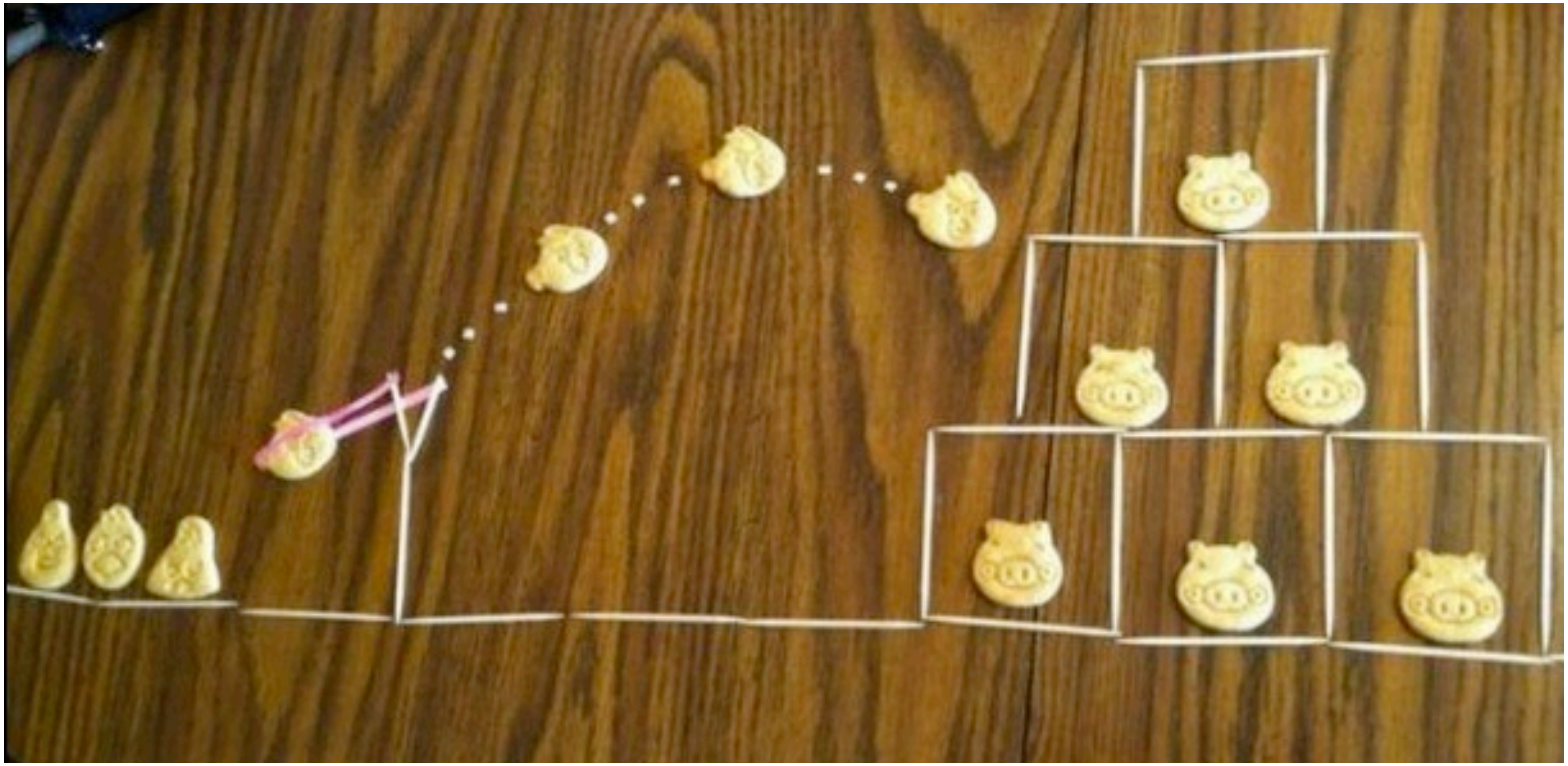


31	15	0	
I/O Map Base Address	Reserved	T	100
Reserved	LDT Segment Selector		96
Reserved	GS		92
Reserved	FS		88
Reserved	DS		84
Reserved	SS		80
Reserved	CS		76
Reserved	ES		72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3 (PDBR)			28
Reserved	SS2		24
ESP2			20
Reserved	SS1		16
ESP1			12
Reserved	SS0		8
ESP0			4
Reserved	Previous Task Link		0

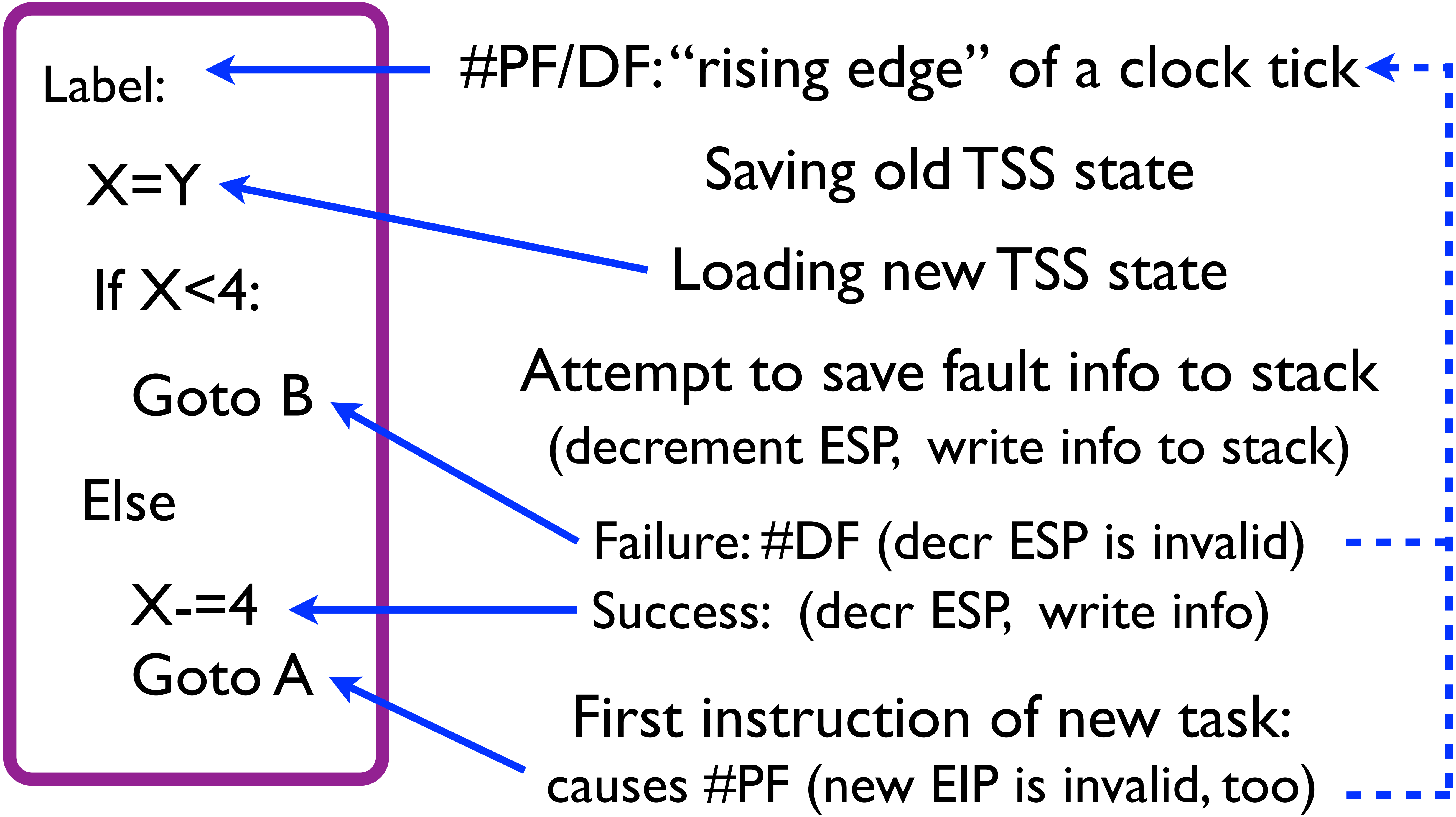

I DON'T ALWAYS COMPUTE,
BUT WHEN I DO, IT'S WITHOUT INSTRUCTIONS

Let's step through an instruction

(Some details glossed over;
think of it as a fairy tale, not a lie)



Instruction by the numbers (or, “PFLA fetch-decode-execute” loop)



```
Label:
  X=Y
  If X<4:
     Goto B
  Else
     X-=4
     Goto A
```

#PF/DF: “rising edge” of a clock tick

Saving old TSS state

Loading new TSS state

Attempt to save fault info to stack

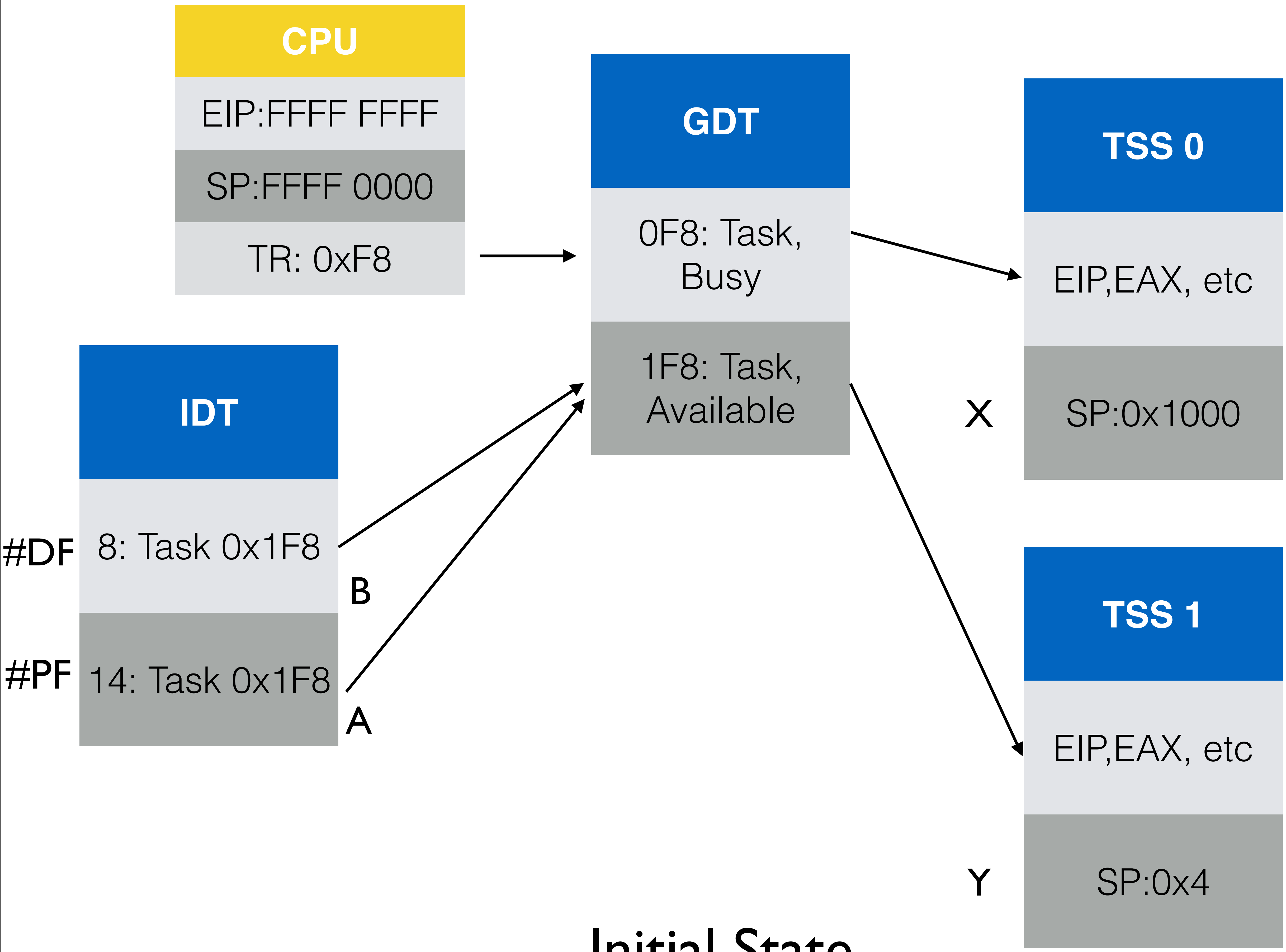
(decrement ESP, write info to stack)

Failure: #DF (decr ESP is invalid)

Success: (decr ESP, write info)

First instruction of new task:

causes #PF (new EIP is invalid, too)



CPU

EIP:FFFF FFFF

SP:FFFF 0000

TR: 0xF8

GDT

0F8: Task, Busy

1F8: Task, Available

TSS 0

EIP,EAX, etc

X

SP:0x1000

IDT

#DF

8: Task 0x1F8

B

#PF

14: Task 0x1F8

A

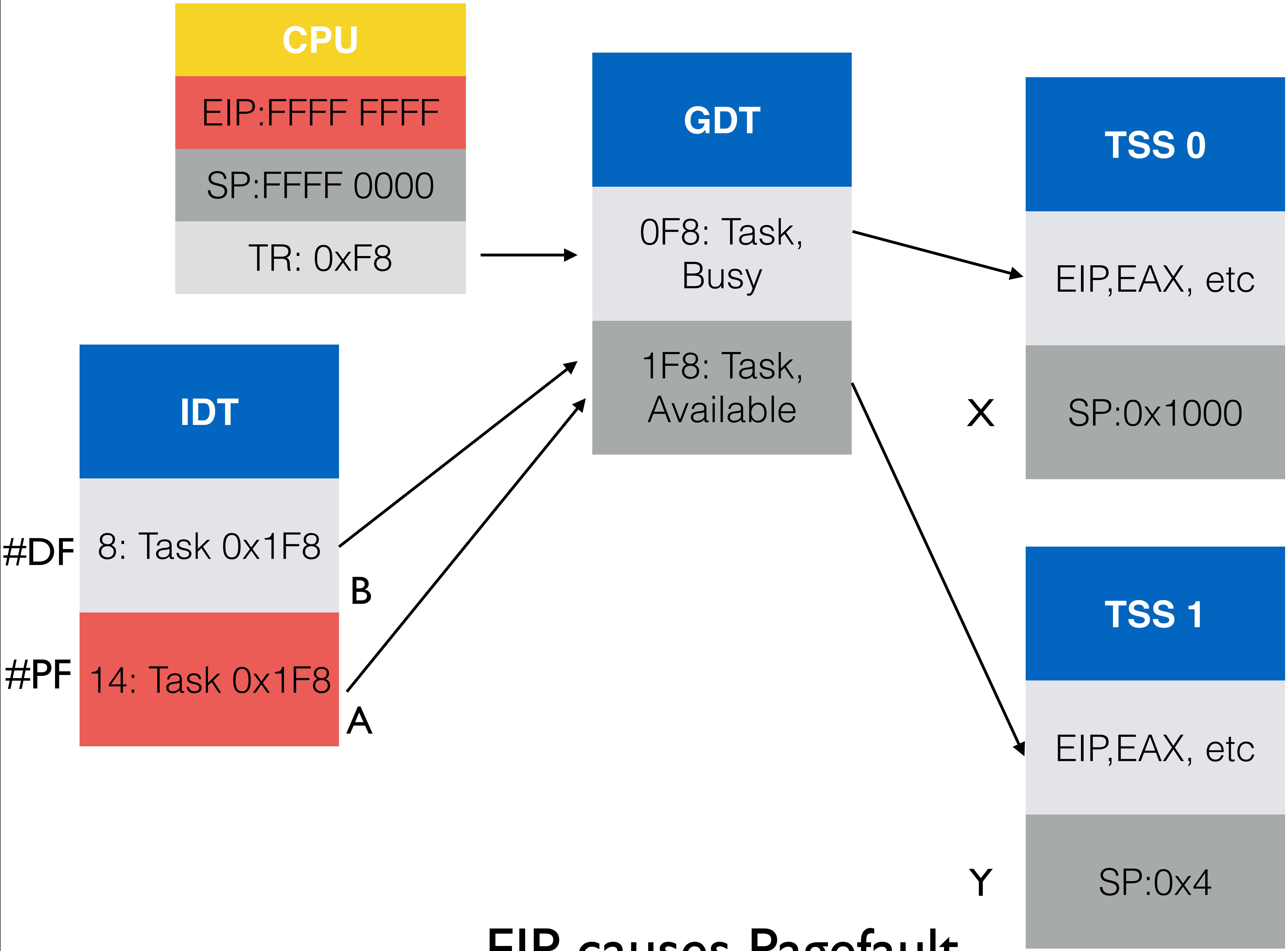
TSS 1

EIP,EAX, etc

Y

SP:0x4

Initial State



CPU

EIP:FFFF FFFF

SP:FFFF 0000

TR: 0xF8

GDT

0F8: Task, Busy

1F8: Task, Available

TSS 0

EIP,EAX, etc

X

SP:0x1000

IDT

#DF 8: Task 0x1F8

B

#PF 14: Task 0x1F8

A

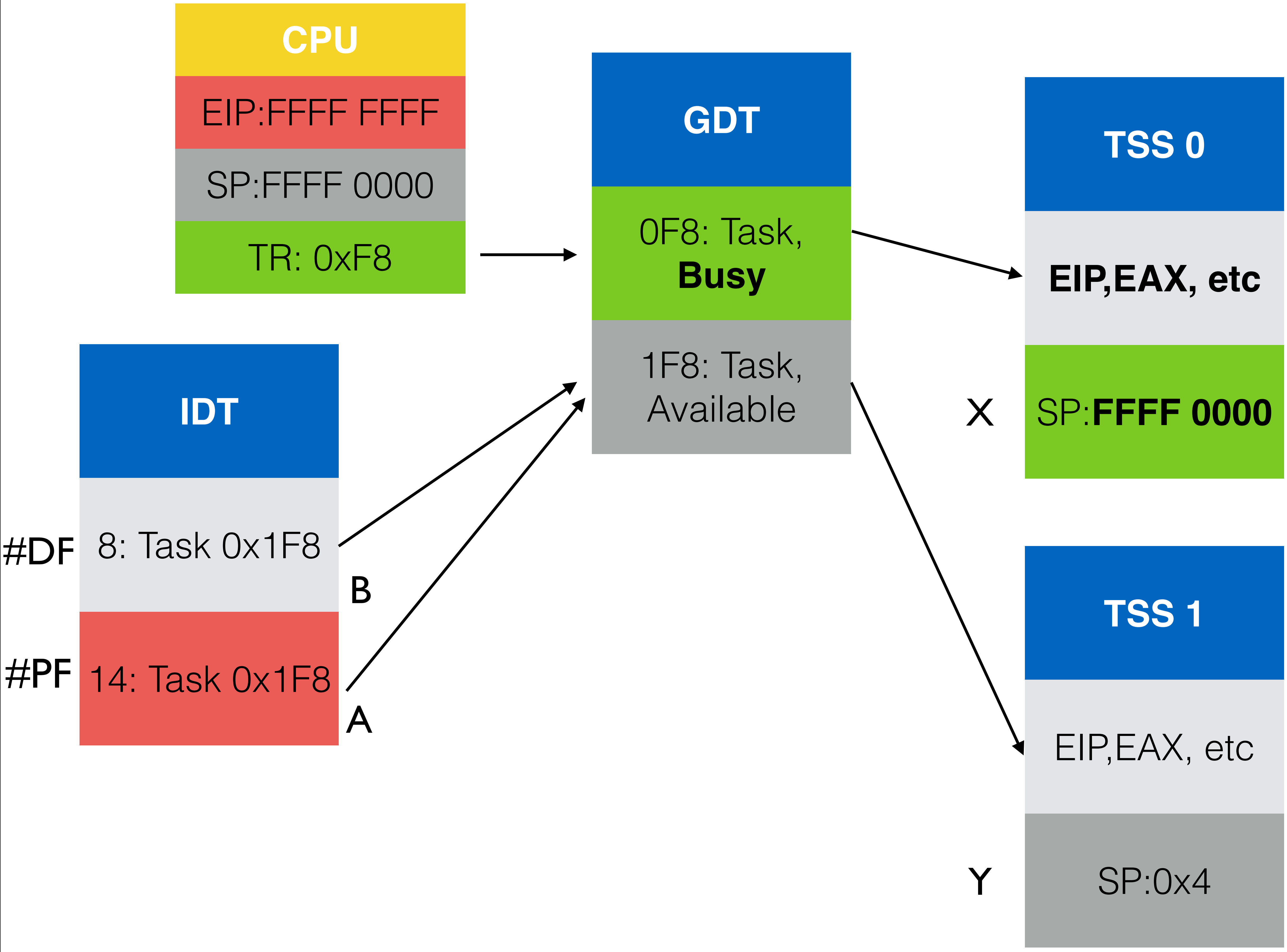
TSS 1

EIP,EAX, etc

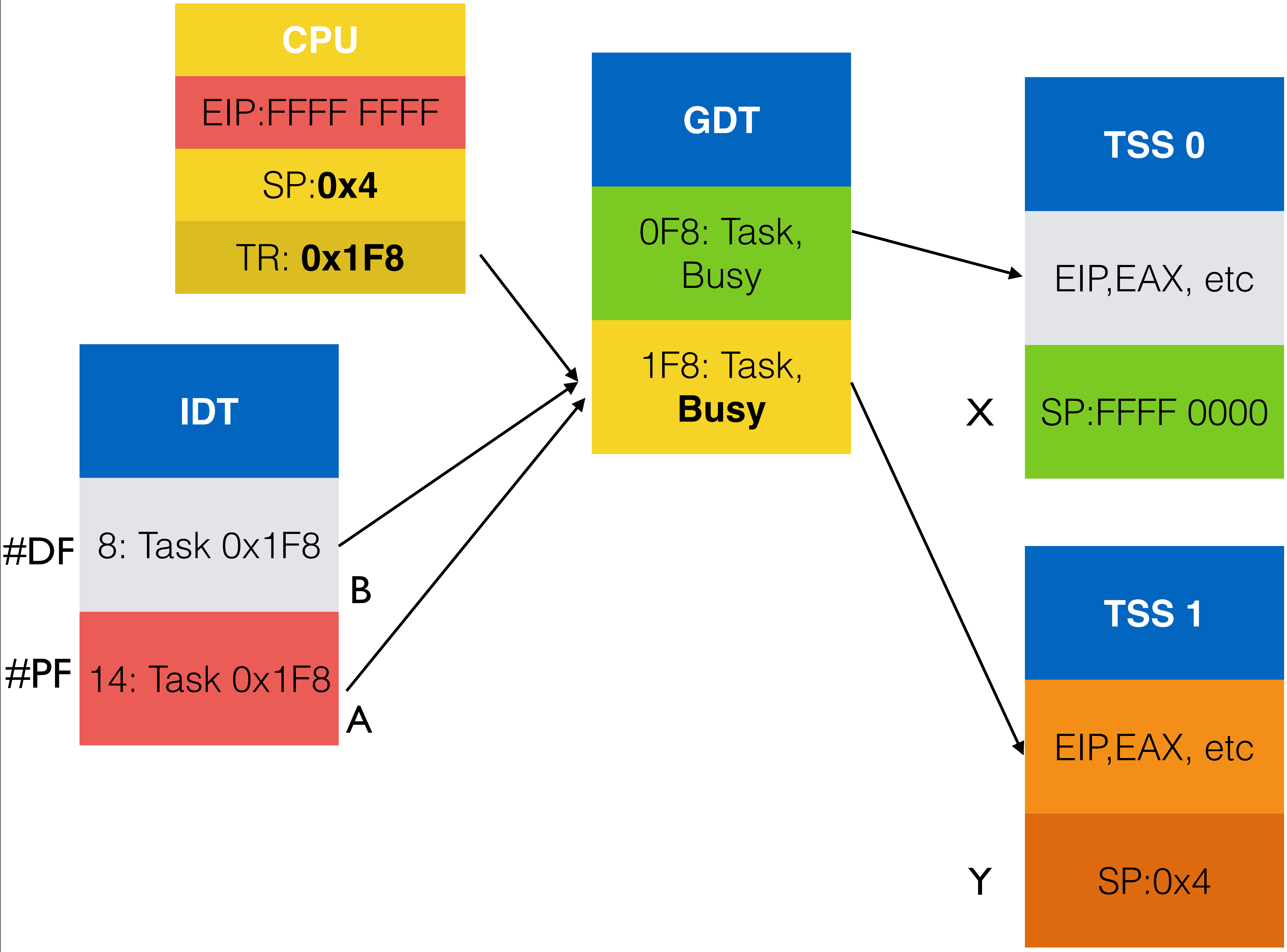
Y

SP:0x4

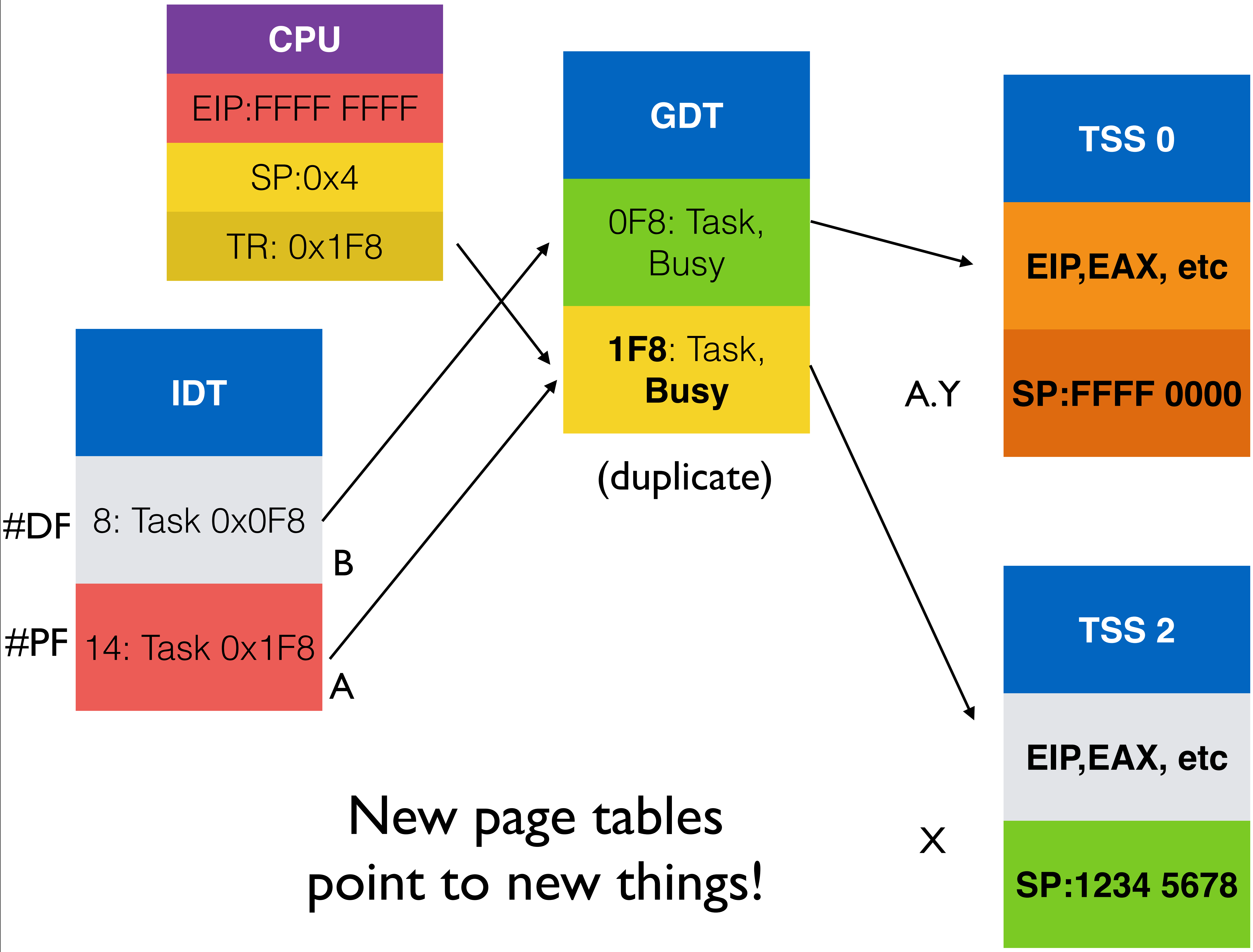
EIP causes Pagefault



CPU state is saved to current task



CPU loads interrupt task



CPU
EIP:FFFF FFFF
SP:0x4
TR: 0x1F8

IDT
#DF 8: Task 0x0F8
#PF 14: Task 0x1F8
B
A

GDT
0F8: Task, Busy
1F8: Task, **Busy**
(duplicate)

A.Y

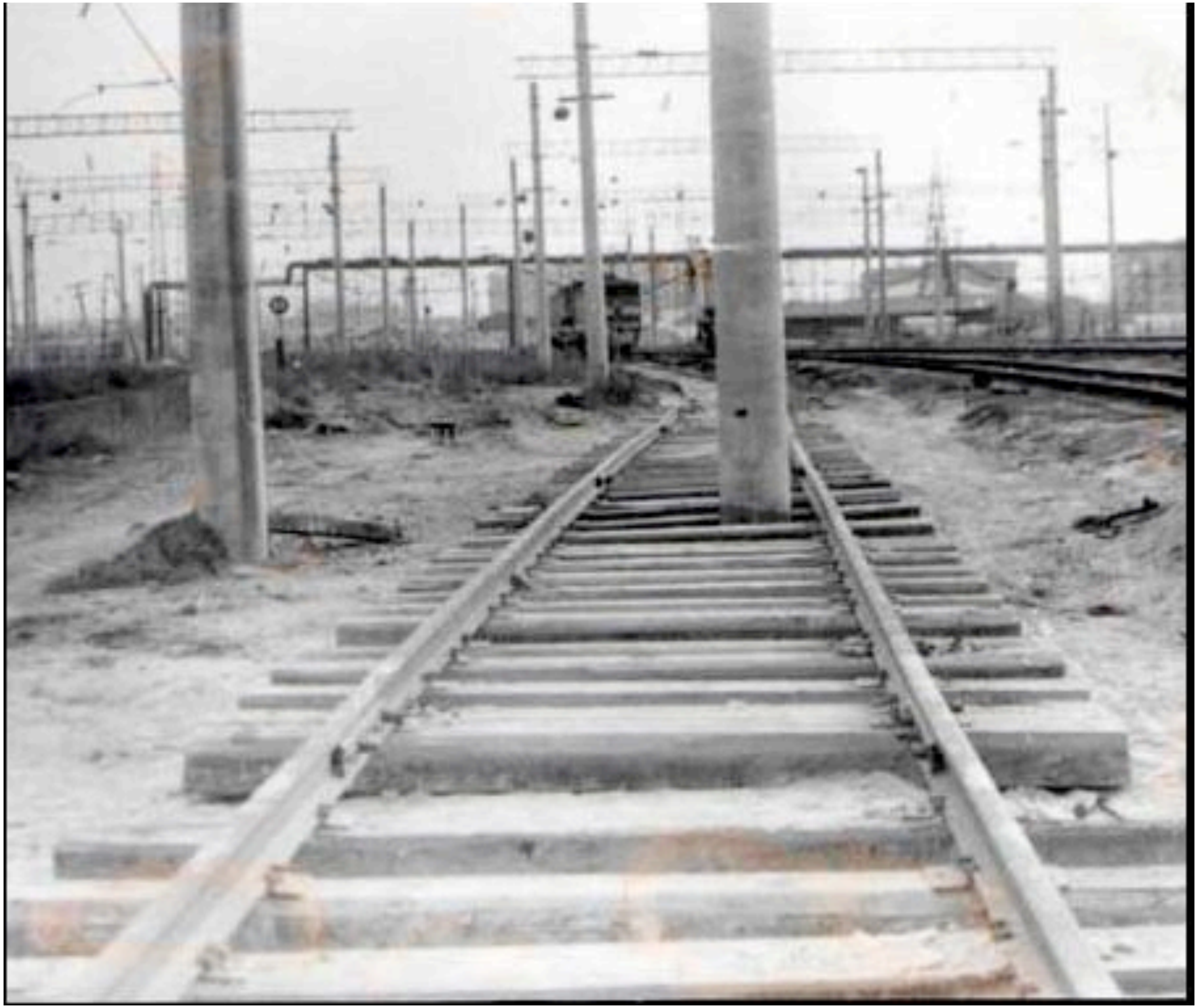
TSS 0
EIP,EAX, etc
SP:FFFF 0000

X

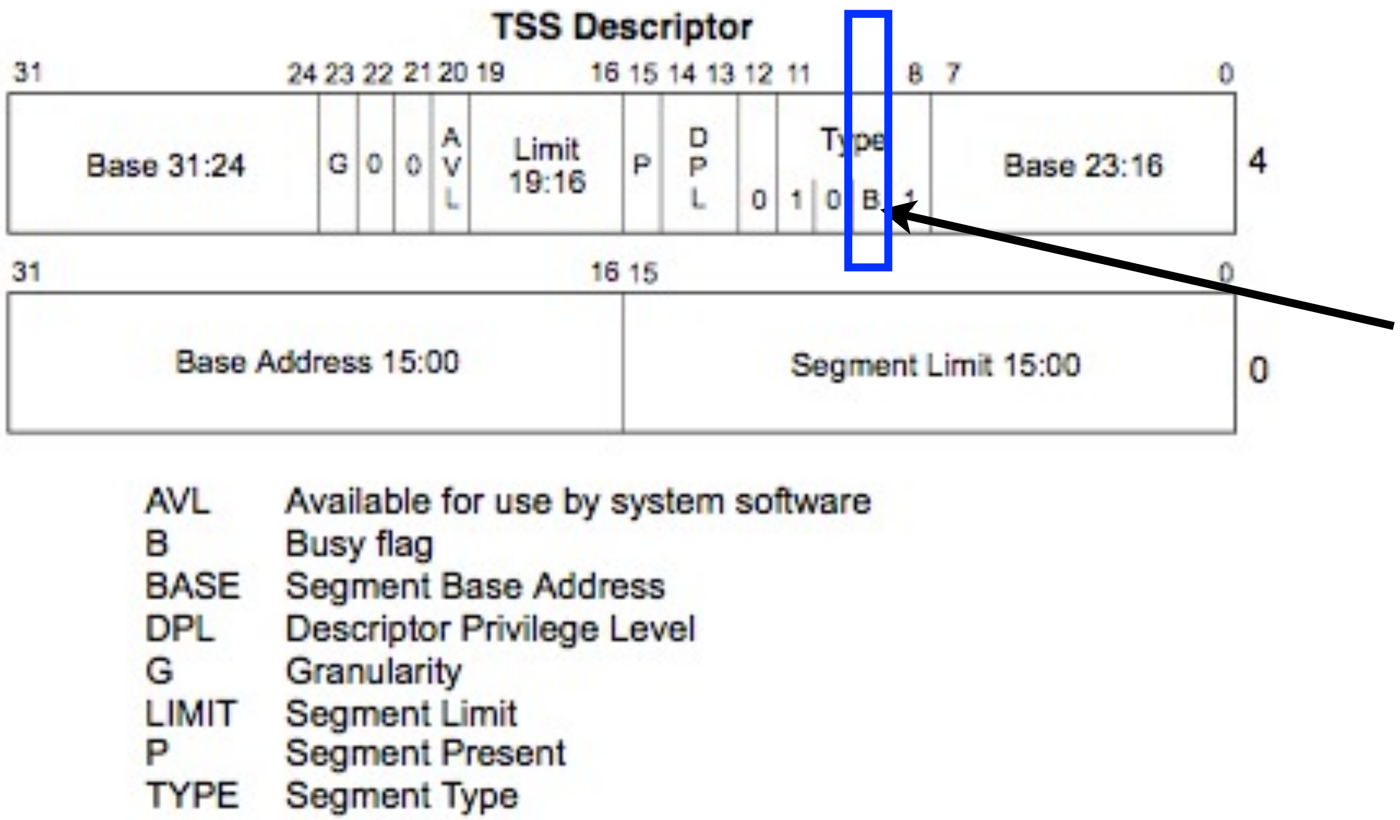
TSS 2
EIP,EAX, etc
SP:1234 5678

New page tables point to new things!

“Implementation Problem”



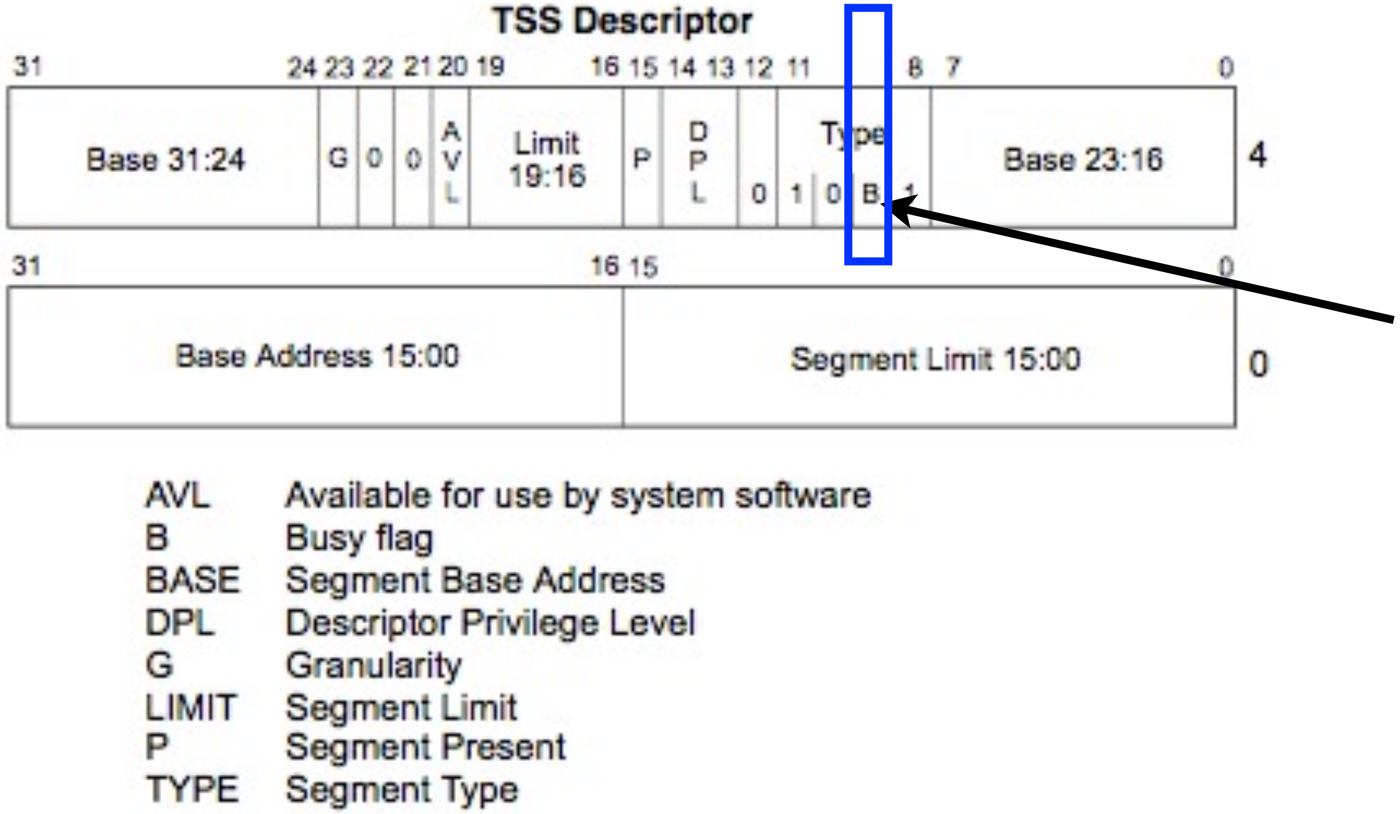
1 bit(ch) of a bit(ch)



AVL	Available for use by system software
B	Busy flag
BASE	Segment Base Address
DPL	Descriptor Privilege Level
G	Granularity
LIMIT	Segment Limit
P	Segment Present
TYPE	Segment Type

CPU won't load task if this is set

1 bit(ch) of a bit(ch)

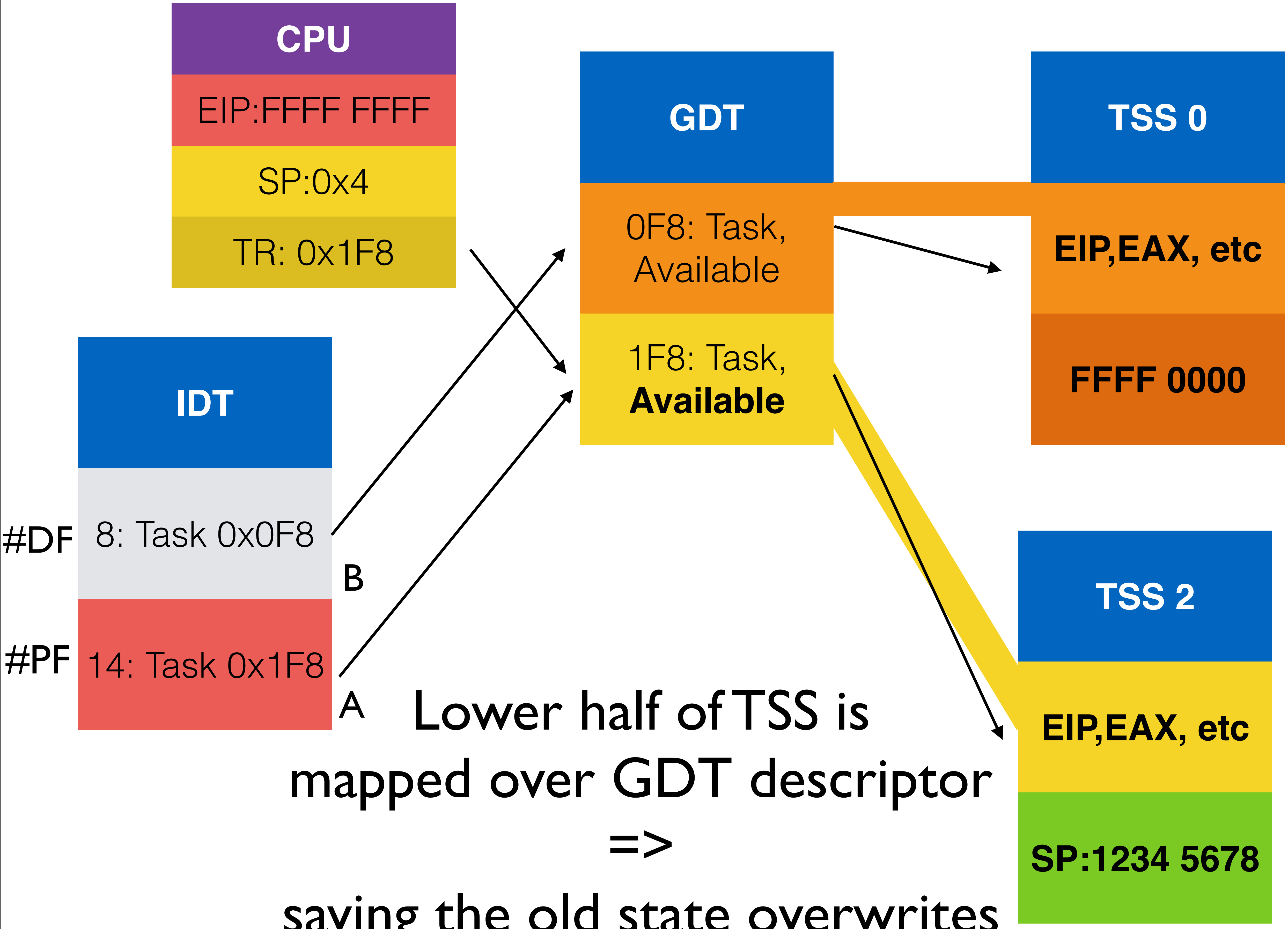


AVL	Available for use by system software
B	Busy flag
BASE	Segment Base Address
DPL	Descriptor Privilege Level
G	Granularity
LIMIT	Segment Limit
P	Segment Present
TYPE	Segment Type

CPU won't load task if this is set

We need to overwrite it. Luckily, the CPU always saves all the state (even if not dirty).
So: map the lower half of TSS over GDT, so that saved EAX,ECX from TSS overwrite descriptor; same content, only busy bit cleared.

Dealing with that bit needs a nuclear option...



CPU

EIP:FFFF FFFF

SP:0x4

TR: 0x1F8

GDT

0F8: Task, Available

1F8: Task, **Available**

TSS 0

EIP,EAX, etc

FFFF 0000

IDT

#DF 8: Task 0x0F8

B

#PF 14: Task 0x1F8

A

TSS 2

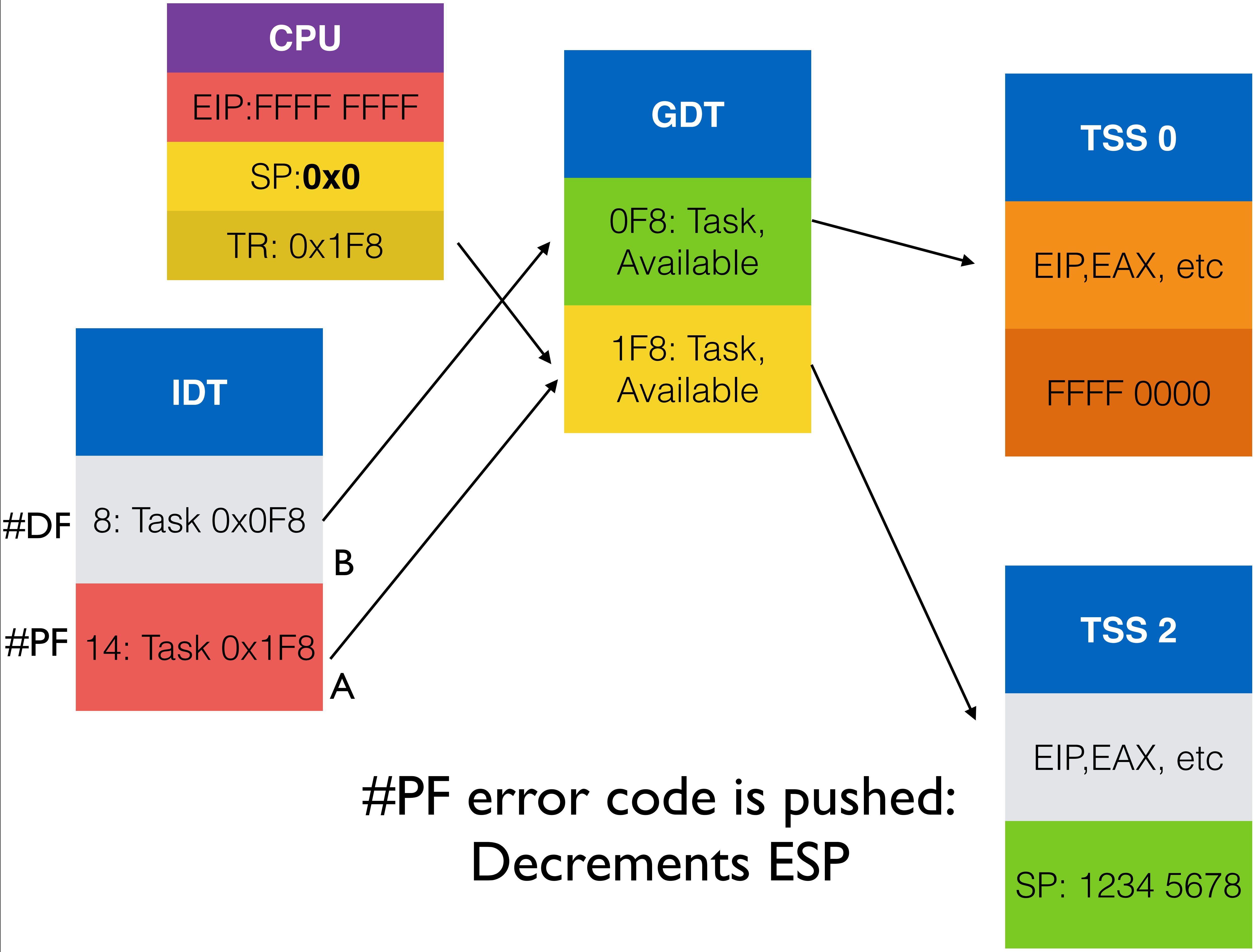
EIP,EAX, etc

SP:1234 5678

Lower half of TSS is mapped over GDT descriptor

=>

saving the old state overwrites the GDT entry busy bit!



CPU

EIP:FFFF FFFF

SP:**0x0**

TR: 0x1F8

IDT

#DF 8: Task 0x0F8

#PF 14: Task 0x1F8

B

A

GDT

0F8: Task, Available

1F8: Task, Available

TSS 0

EIP,EAX, etc

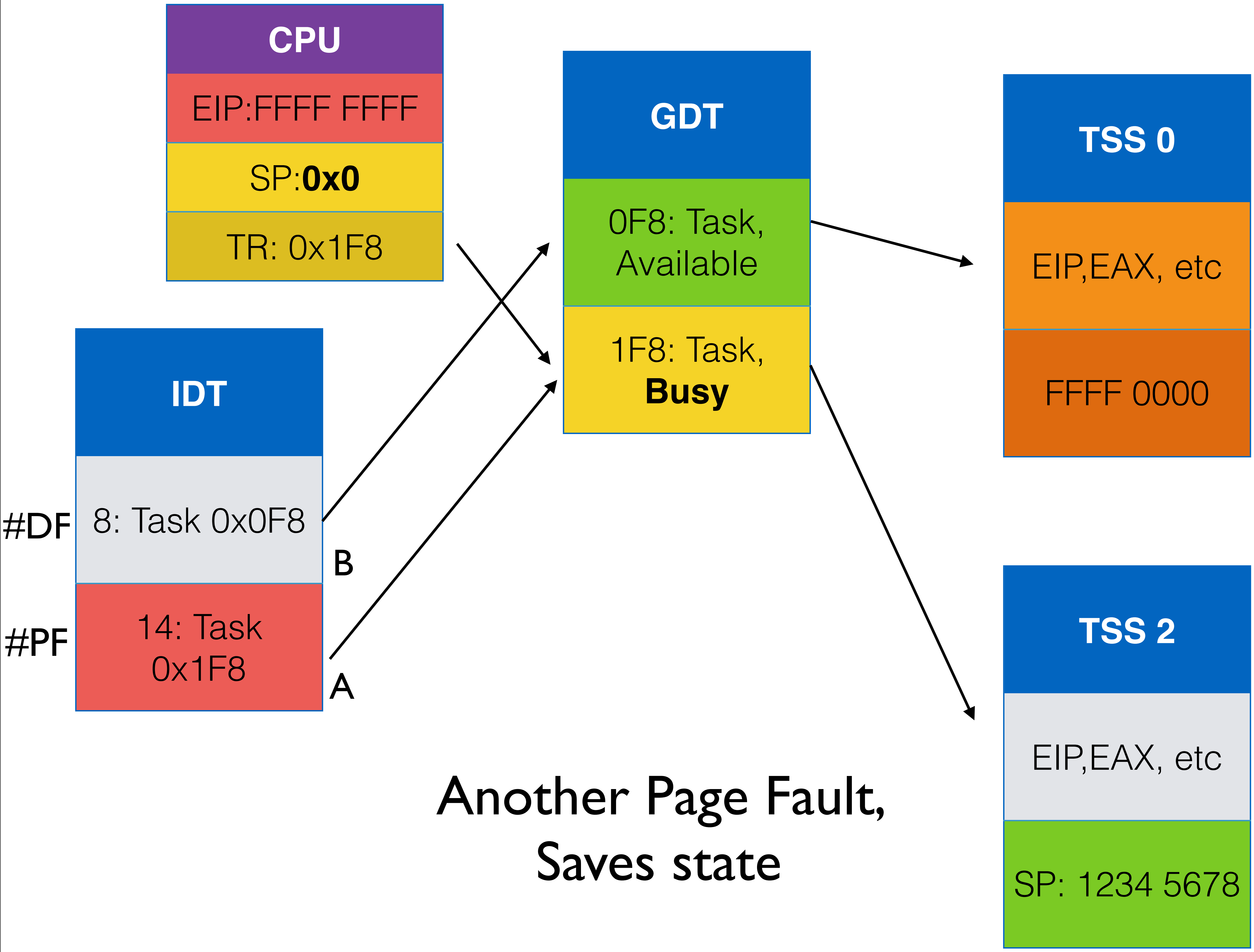
FFFF 0000

TSS 2

EIP,EAX, etc

SP: 1234 5678

#PF error code is pushed:
Decrements ESP



CPU

EIP:FFFF FFFF

SP:**0x0**

TR: 0x1F8

GDT

0F8: Task, Available

1F8: Task, **Busy**

TSS 0

EIP,EAX, etc

FFFF 0000

IDT

#DF 8: Task 0x0F8

#PF 14: Task 0x1F8

B

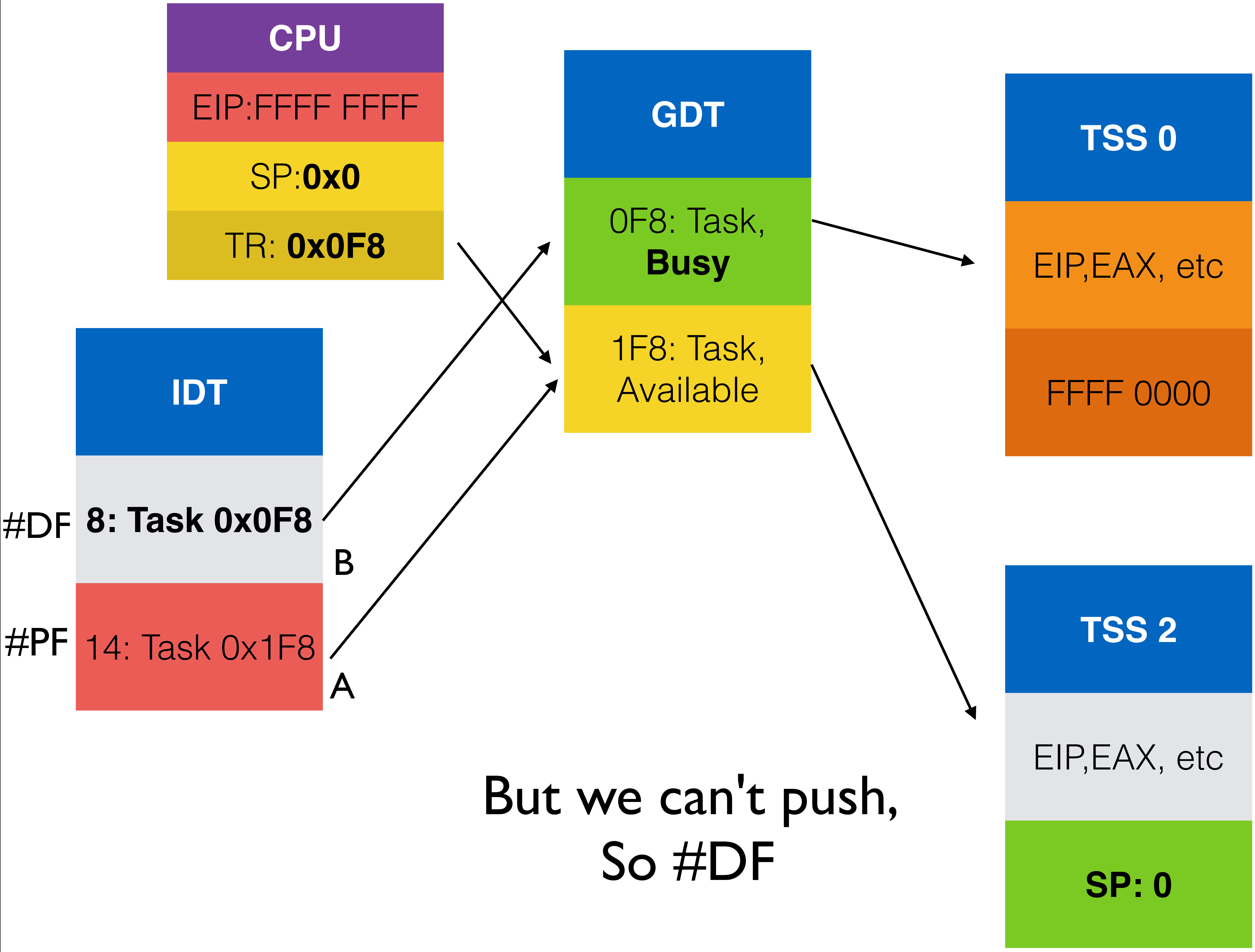
A

TSS 2

EIP,EAX, etc

SP: 1234 5678

Another Page Fault, Saves state



CPU

EIP:FFFF FFFF

SP:**0x0**

TR: **0x0F8**

IDT

#DF **8: Task 0x0F8**

#PF 14: Task 0x1F8

B

A

GDT

0F8: Task, **Busy**

1F8: Task, Available

TSS 0

EIP,EAX, etc

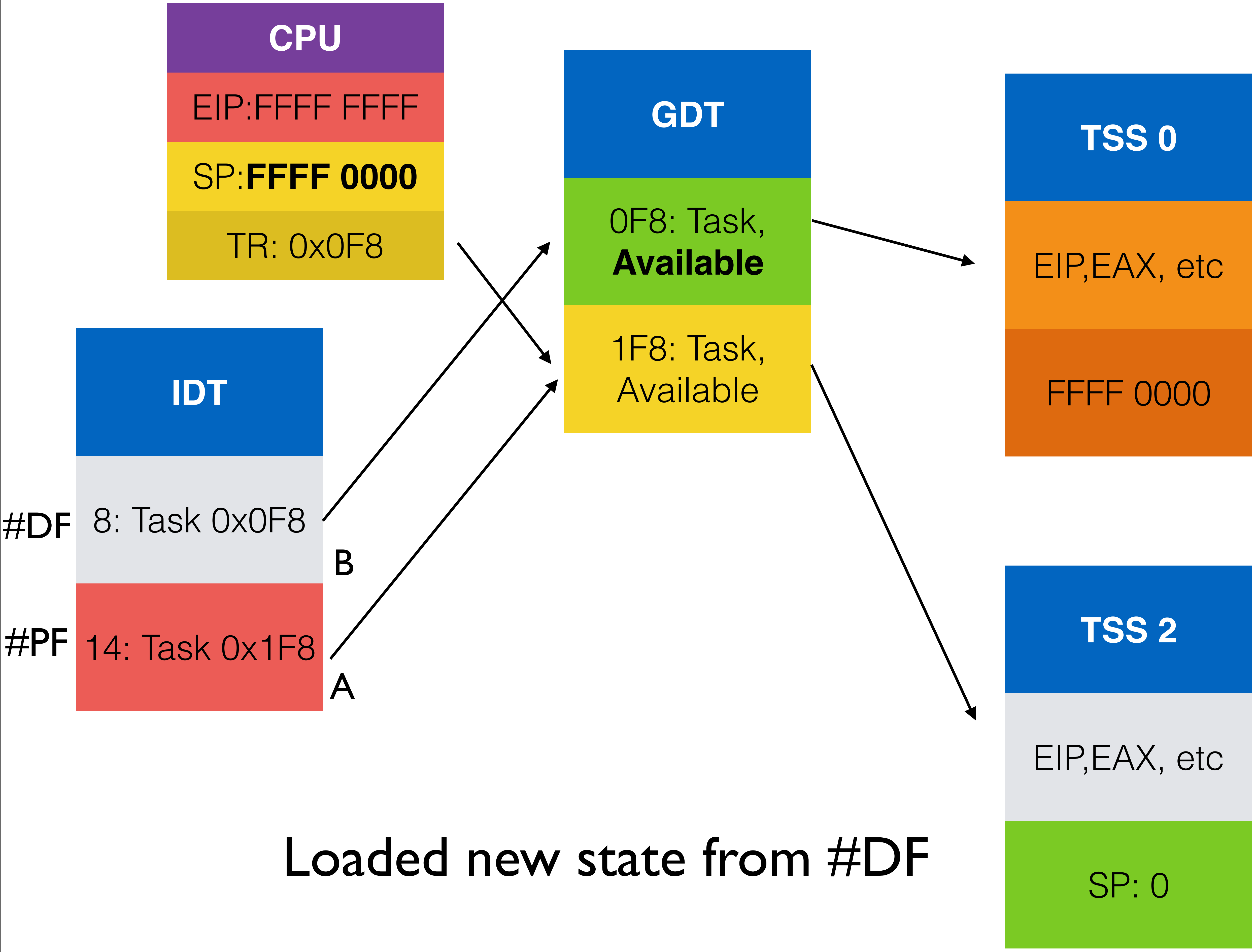
FFFF 0000

TSS 2

EIP,EAX, etc

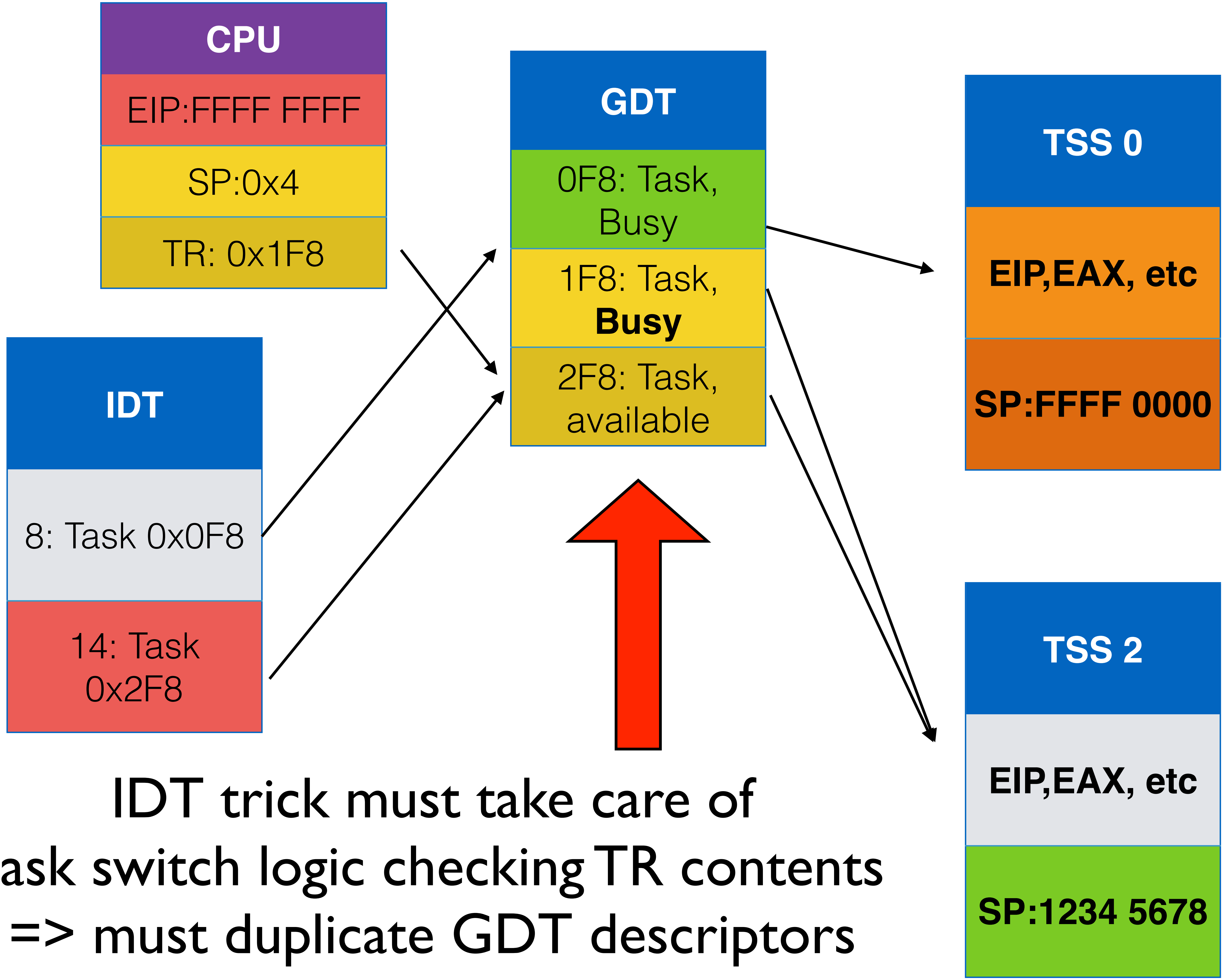
SP: 0

But we can't push,
So #DF



Loaded new state from #DF

And now to face the uglier truth...



IDT trick must take care of
task switch logic checking TR contents
=> must duplicate GDT descriptors

Meanwhile, on the FSB

(Slightly redacted)

Write 0x8	0xFFFF 0000
Read 0x1008	0x4
Write 0x2008	0x0
Read 0x8	0xFFFF 0000

And they all compute happily ever after

(for all we know)

What restrictions do we have?

- Needs kernel access to set up :)
- No two double faults in a row
- Can only use our one awkward instruction
- Can only work with SP of TSS aligned across page (very limited coverage of phys. mem)

In Soviet Russia,
Red Pill takes you

White Hat Takeaway

- Check how your tools handle old/unused CPU features
- Don't trust the spec

Black Hat Takeaway

- A really nice, big Redpill
- With more work, you can probably make it work differently in Analysis tools
- Or just shoot down the host

Strawhat Takeaway

- It's a weird machine! (And we like them)
- We are working on 64 bit, better tools
  - Compiler, debugger
- See how it works on different hardware?

“There is never enough time.
Thank you for yours!”

--Dan Geer

“I have a dream”

- of a world where a hacker isn’t judged by the color of his hat, but the weirdness of his machine
- of a world where a single step in can change your world completely
- of a world where we strive to understand what dragons sleep in seemingly innocent systems