Page Fault Liberation Army

It's ~~turtles~~ Turing machines all the way down!

Julian Bangert,
Sergey Bratus,
Rebecca ".bx" Shapiro

Trust Lab
Dartmouth College

“Page Fault Liberation”

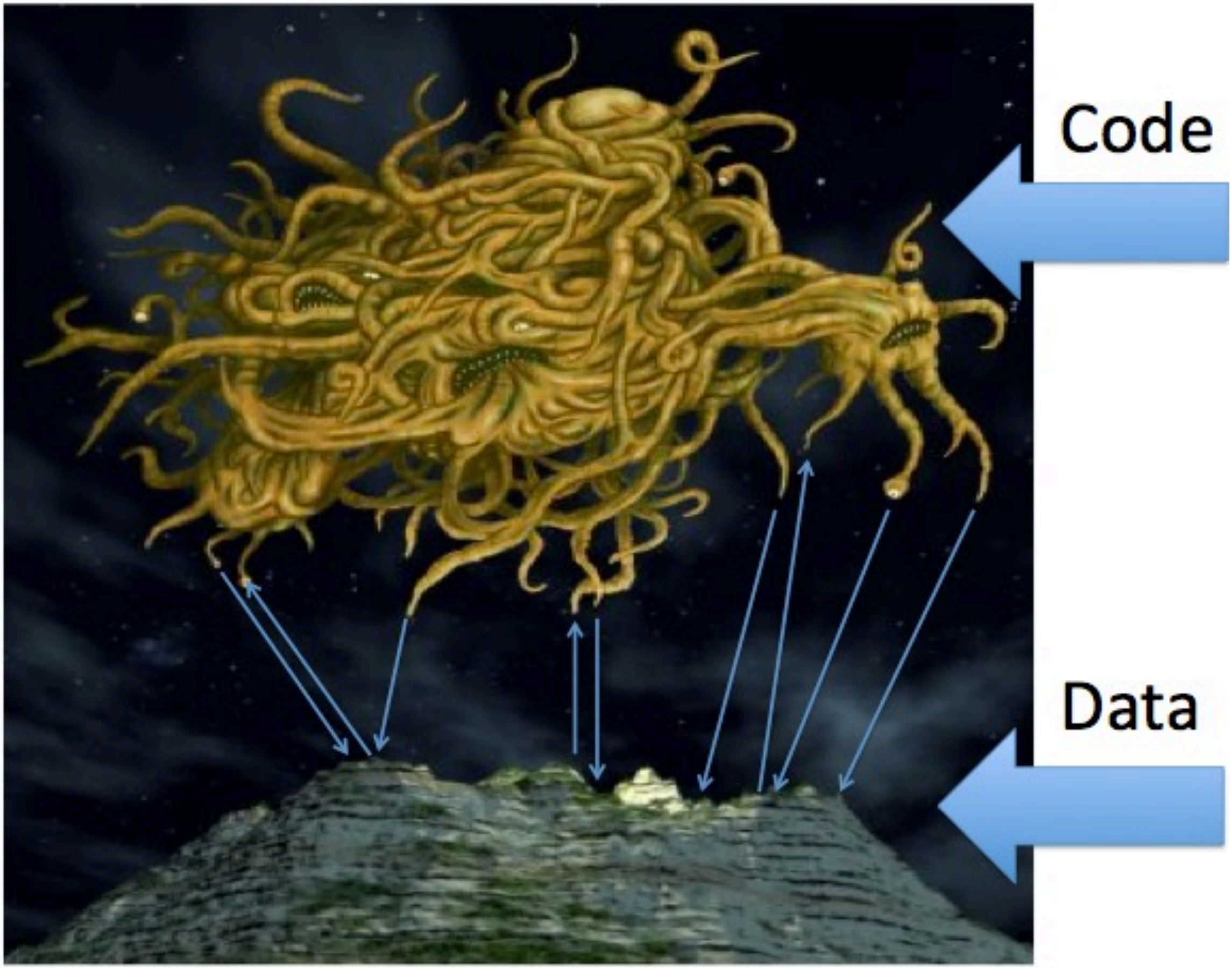
- The x86 MMU is not just a look-up table!
- x86 MMU performs complex logic on complex data structures
- The MMU has **state** and **transitions** that brilliant hackers put to unorthodox uses.
- Can it be **programmed** with its data? **YES**

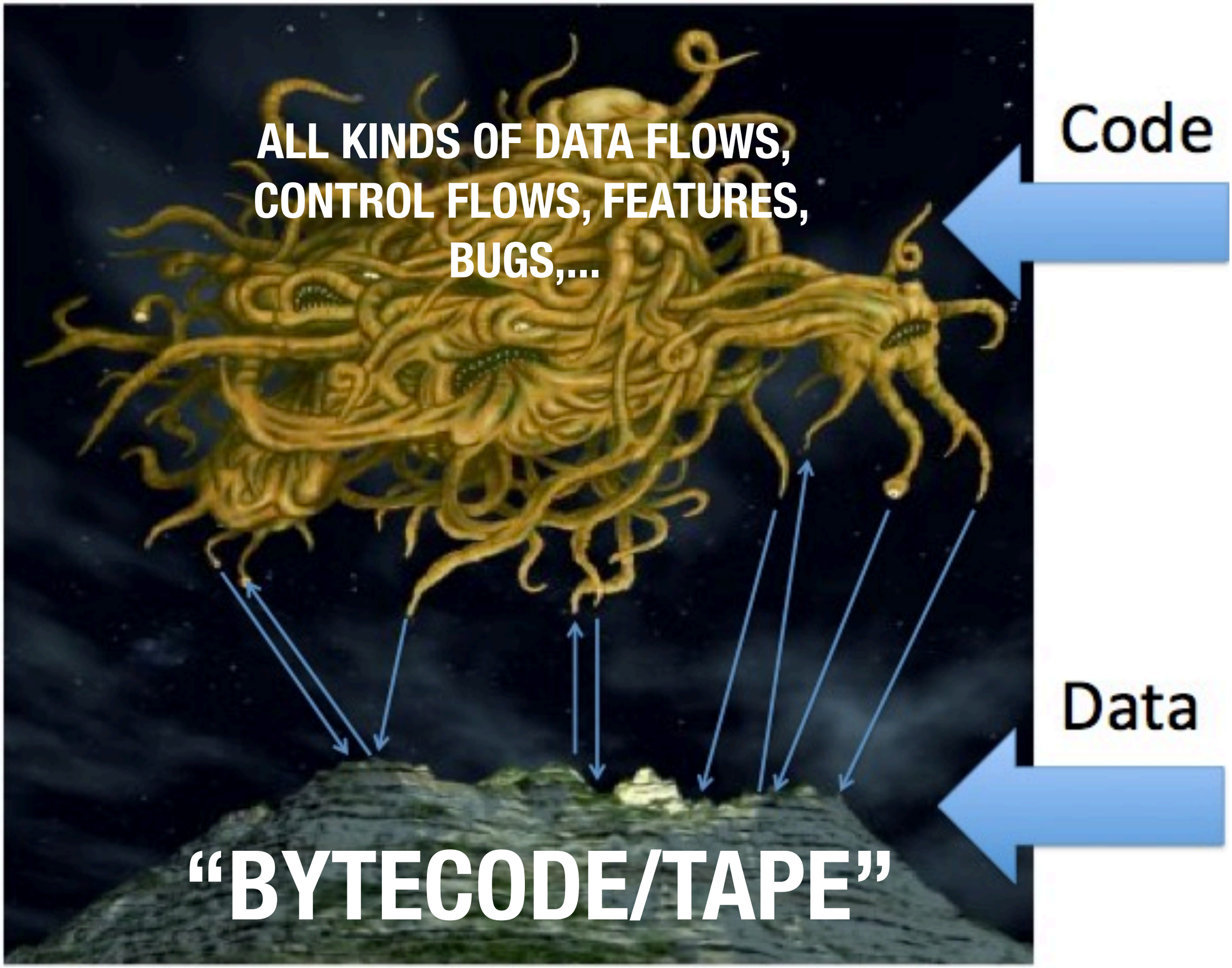
Disclaimer

- ***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 sufficiently advanced/complex input data/metadata acts as "***bytecode***" to the system that must interpret it; that system acts as a "***virtual machine***" for that bytecode (!)

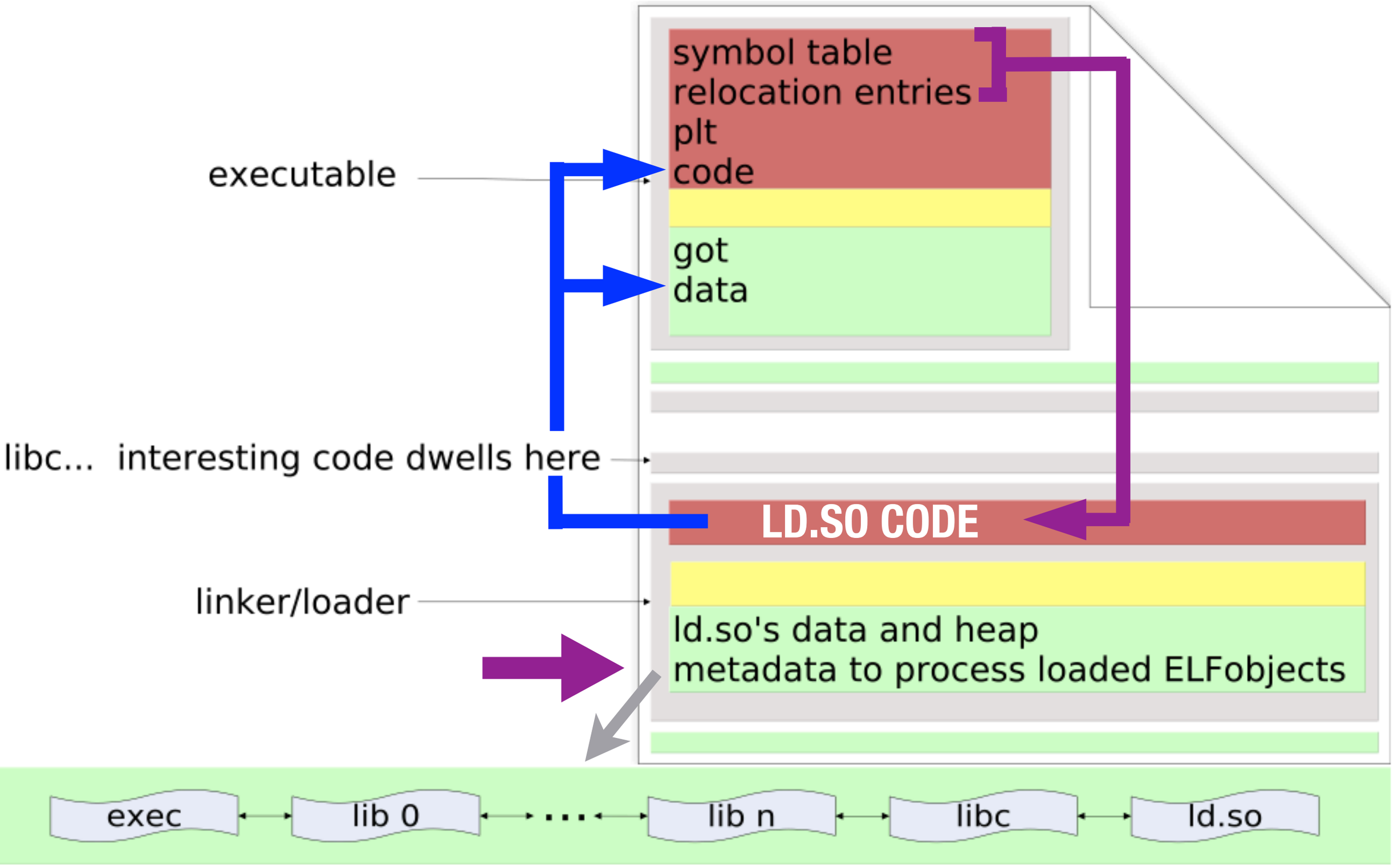

Code
Data


ALL KINDS OF DATA FLOWS,
CONTROL FLOWS, FEATURES,
BUGS,...
Code
Data
“BYTECODE/TAPE”

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 than linked for.

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

RTLD arithmetic

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:

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

s:

```
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;
```

- **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**

```
Relocation section '.rela.p' at offset 0xf3a8 contains 14 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend

00000060dfe0  002d00000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
00000060e9e0  004e00000005 R_X86_64_COPY     00000000060e9e0 __progname + 0
00000060e9f0  004b00000005 R_X86_64_COPY     00000000060e9f0 stdout + 0
00000060e9f8  005100000005 R_X86_64_COPY     00000000060e9f8 __progname_full + 0
00000060ea00  005600000005 R_X86_64_COPY     00000000060ea00 stderr + 0
00000060eb40  000000000005 R_X86_64_COPY                     0000000000000000
00000060eb40  000000000001 R_X86_64_64                       0000000000000018
00000060eb40  000000000005 R_X86_64_COPY                     0000000000000000
00000060eb40  000000000001 R_X86_64_64                       0000000000000018
00000060eb40  000000000005 R_X86_64_COPY                     0000000000000000
00000060eb40  000000000005 R_X86_64_COPY                     0000000000000000
00000060eb40  000000000001 R_X86_64_64                       00000000000be6e0
00000060e028  000000000001 R_X86_64_64                       0000000000000000
00000060e218  000000000008 R_X86_64_RELATIVE                 000000000401dc2

Symbol table '.sym.p' contains 90 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 00000000060dff0     8 FUNC    LOCAL  DEFAULT  UND
```

See 29c3 talk by Rebecca "**.bx**" Shapiro,

https://github.com/bx/elf-bf-tools

Hacker research inspirations

- "Backdooring binary objects, **klog** [Phrack 56:9]
- "Cheating the ELF", **the grugq** [also Phrack 58:5]
- PLT redirection, **Silvio Cesare** [Phrack 56:7, ...]
- Injecting objects, **mayhem** [Phrack 61:8]
- ElfSh/**ERESI team**, http://eresi-project.org/
- LOCREATE, **skape** [Uninformed 6, 2007]
  - *Rewriting (unpacking) of binaries using REL**

“Page Fault Liberation”

Let’s take an old and known thing...

“Page Fault Liberation”

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

“Page Fault Liberation”

and perhaps others can take it further!

"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

Pagetables

Stack

Traps + Tables = weird machine?

- unmapped/bad memory reference **trap**, based on **page tables** & (current) **IDT**

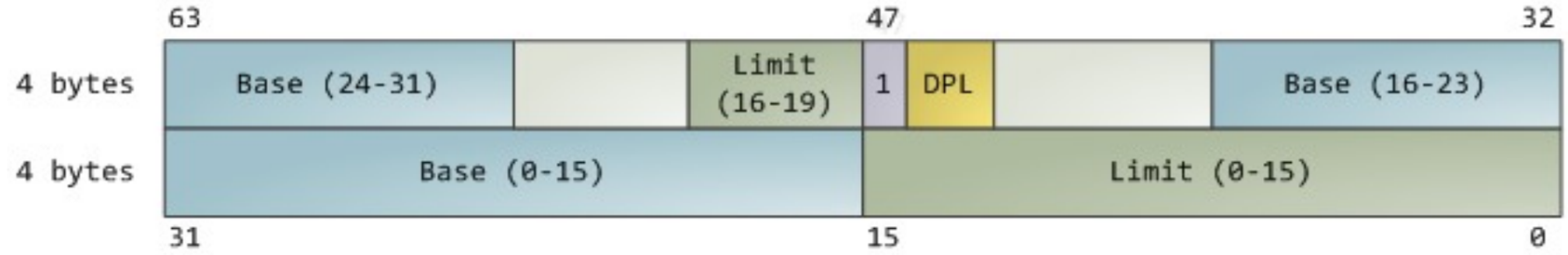
- hardware **writes fault info** on the stack - where it **thinks** the stack is (address in TSS)

- If we point "**stack**" into **page tables, GDT** or TSS, can we get the "tape" of a Turing machine?

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, ...

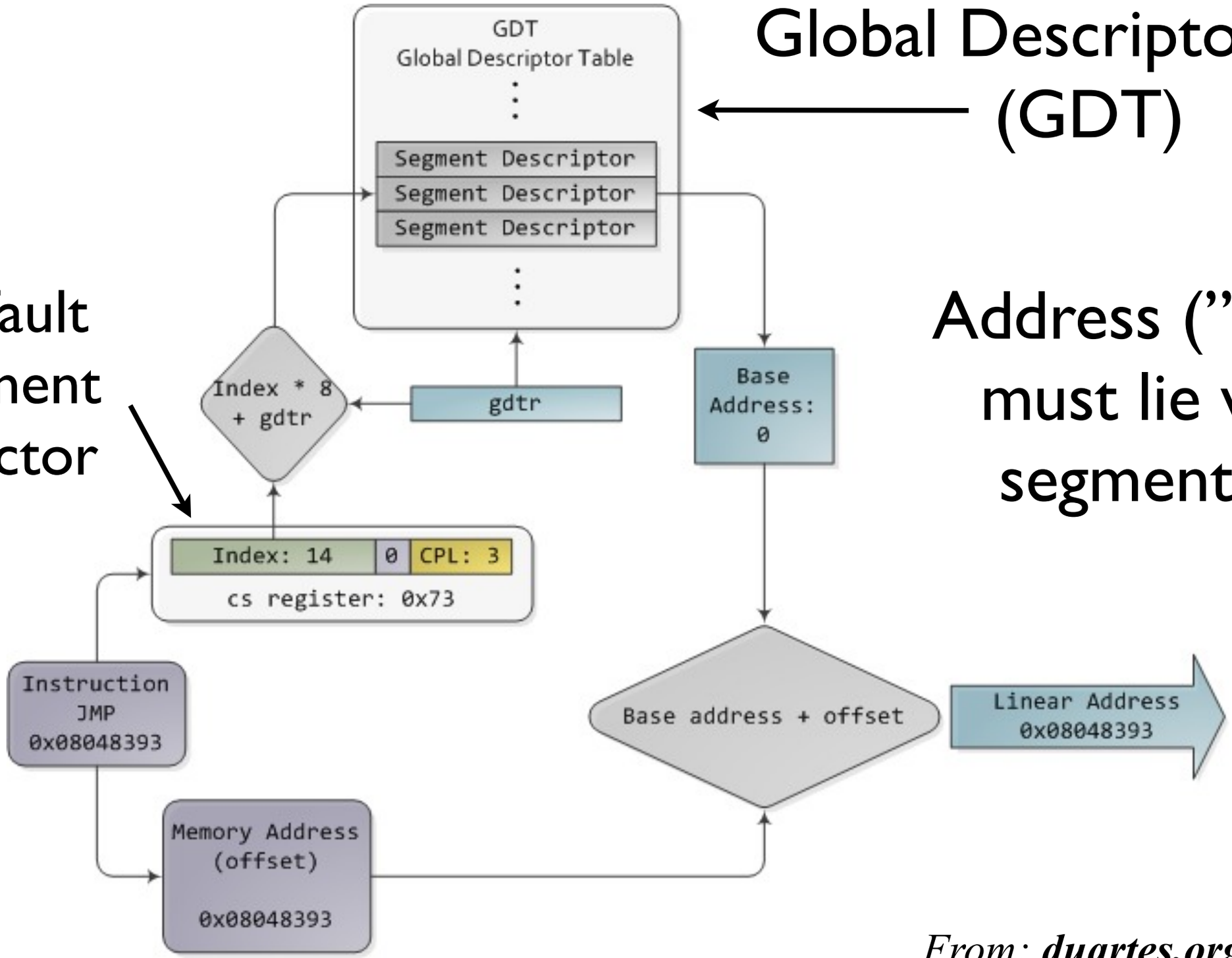
Segment descriptor:



Global Descriptor Table (GDT)

Default segment selector

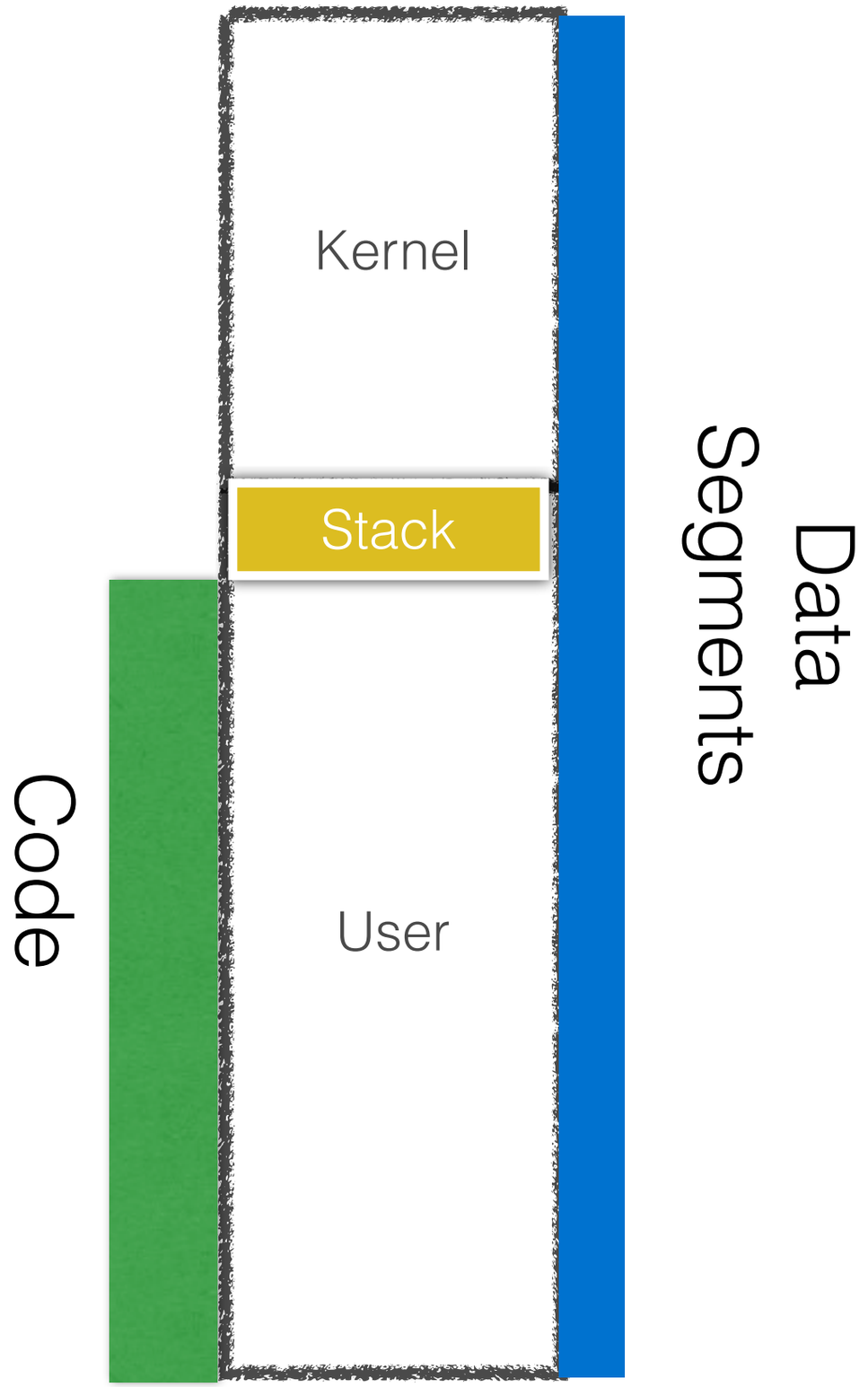
Address (”offset”) must lie within segment limit



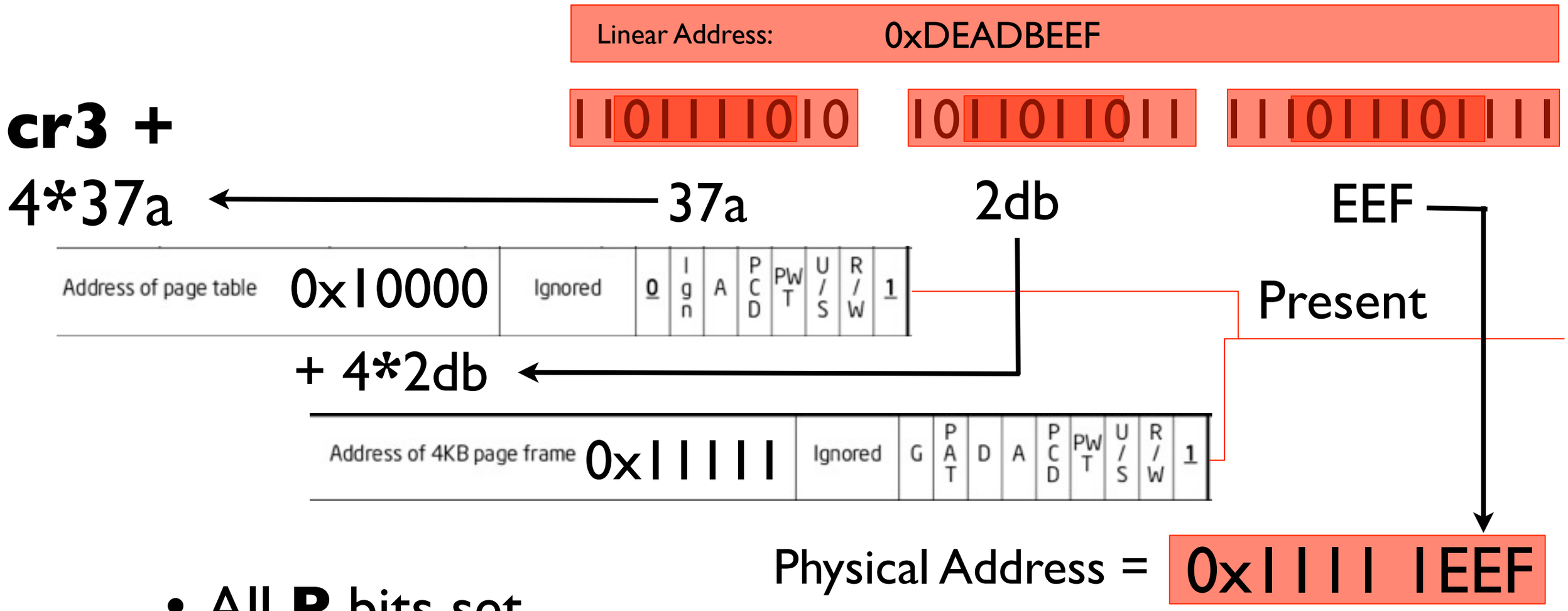
From: ***duartes.org/gustavo/blog/***

OpenWall

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



Virtual Address Translation



- 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?

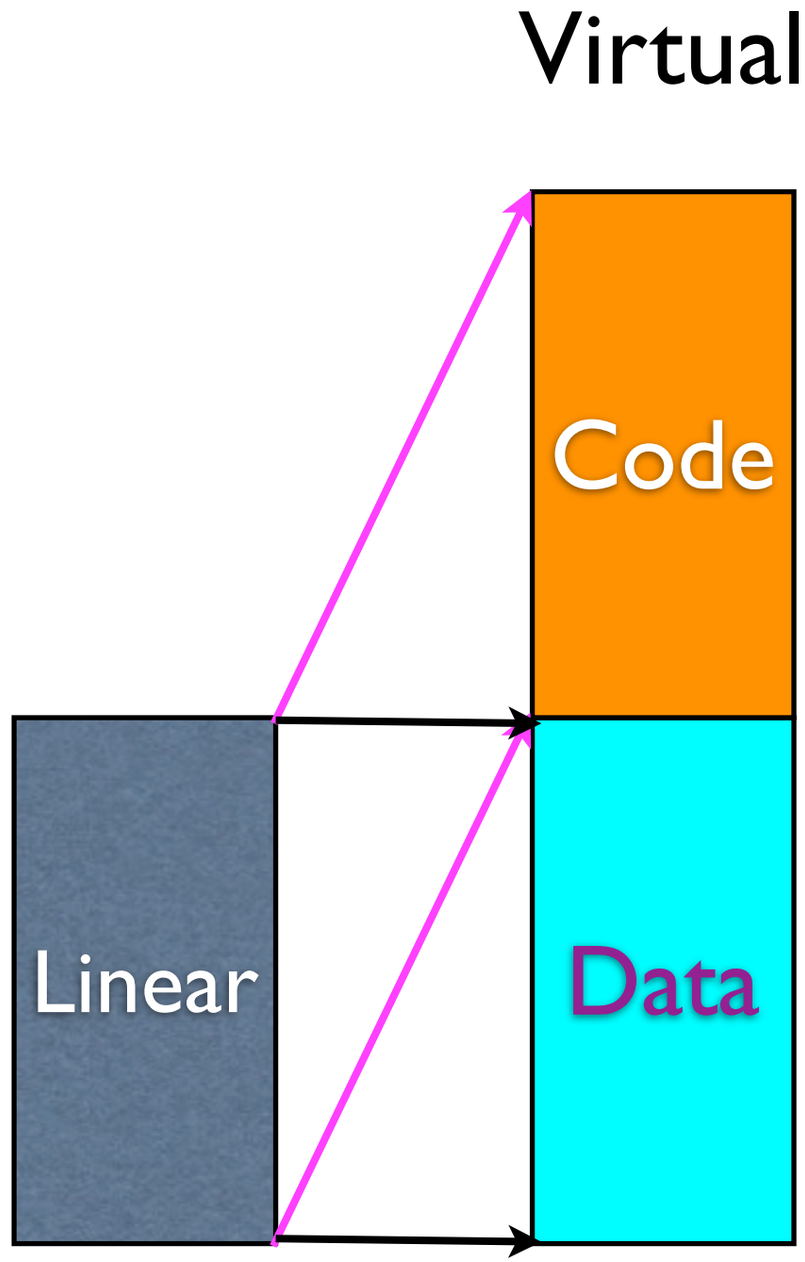
ITS A TRAP

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.base
- 3GB user space
- All Segment limits = 1.5 GB
- Data access goes to lower half of VA space
- Instruction fetch goes to upper half of VA space

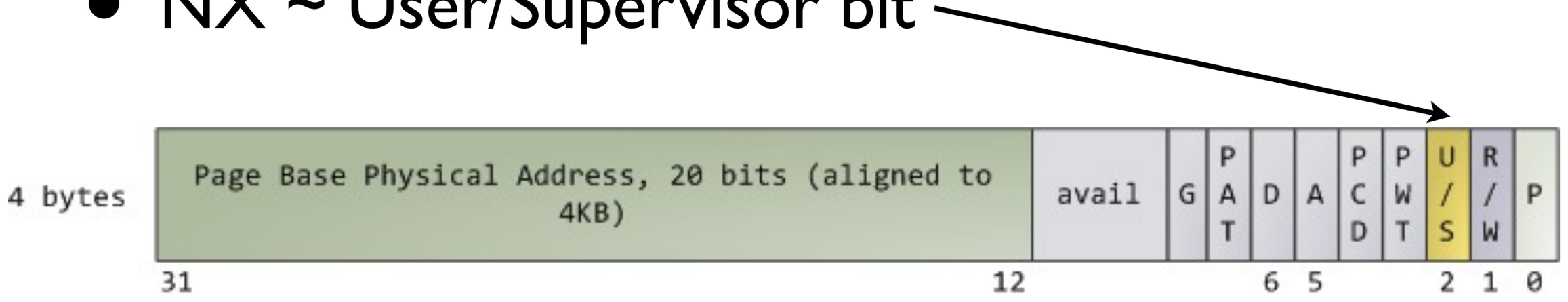


PaX **NX**: PageExec

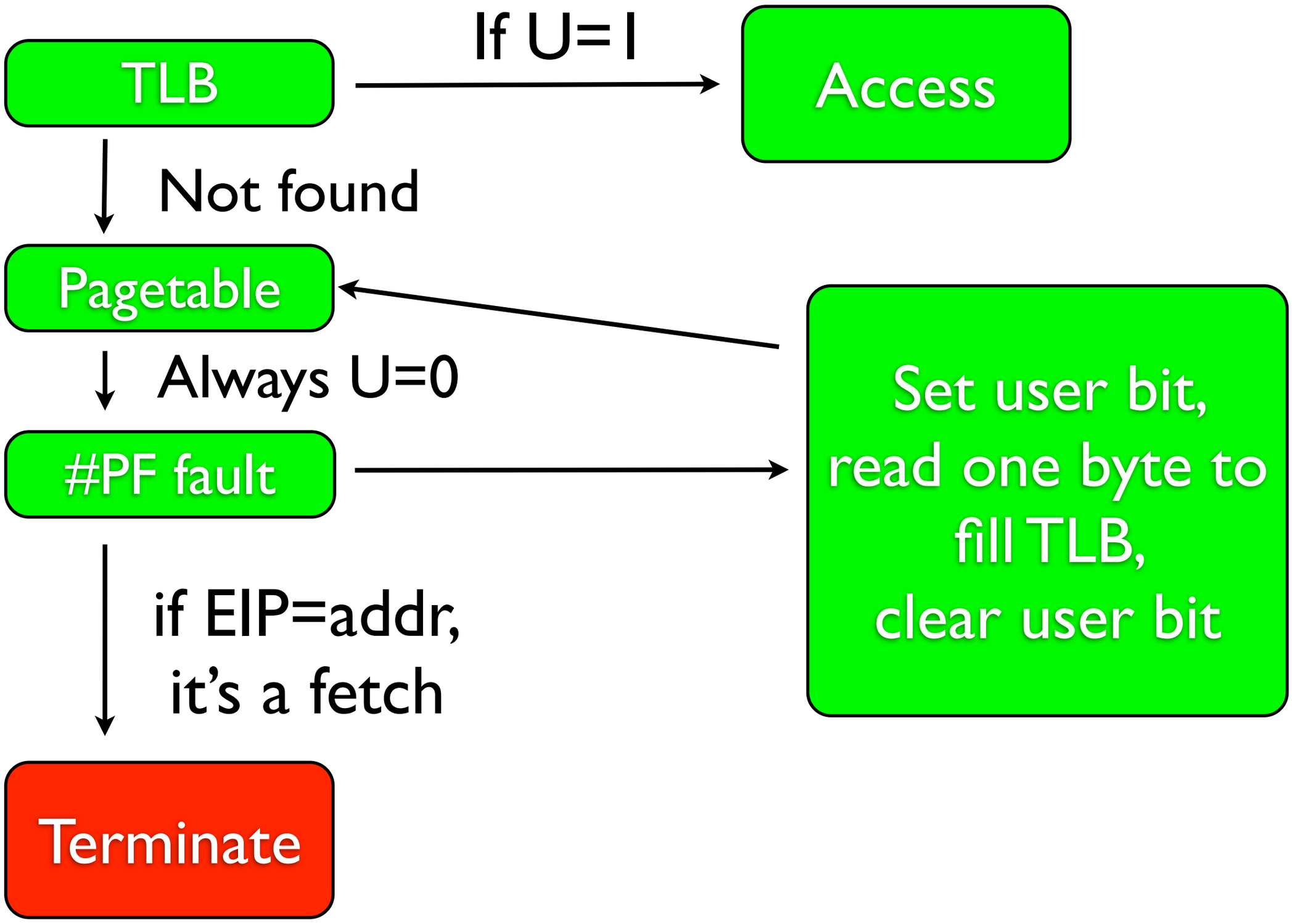
- "**split TLB**" (iTLB for fetches, dTLB for loads) [Plex86 1997, to detect self-modifying code: http://pax.grsecurity.net/docs/pageexec.old.txt]

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

- NX ~ User/Supervisor bit



PageExec data lookup



OllyBone:
Trap on end of unpacker

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

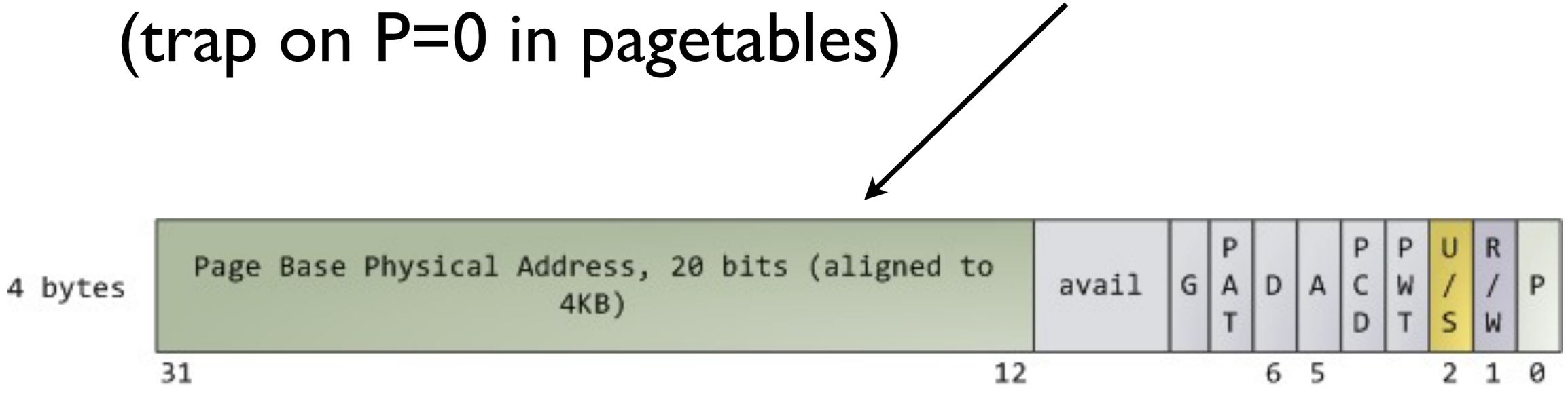
http://www.joestewart.org/ollybone/

ShadowWalker

Phrack 63:8, BlackHat 2005, DEFCON 13

- 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)

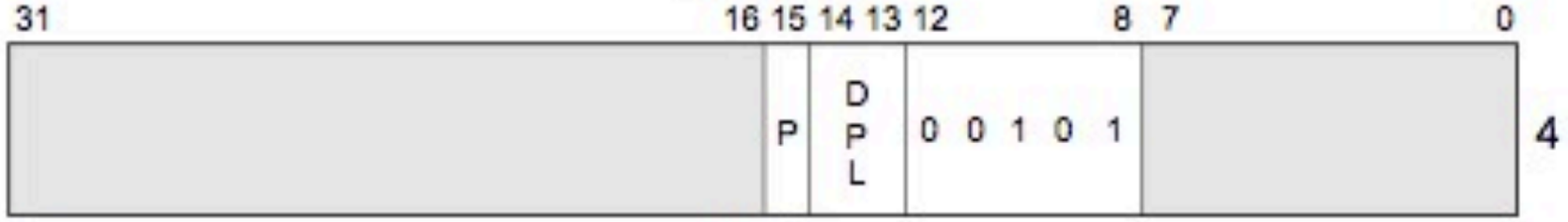


What's in a trap handler (let's roll our own)

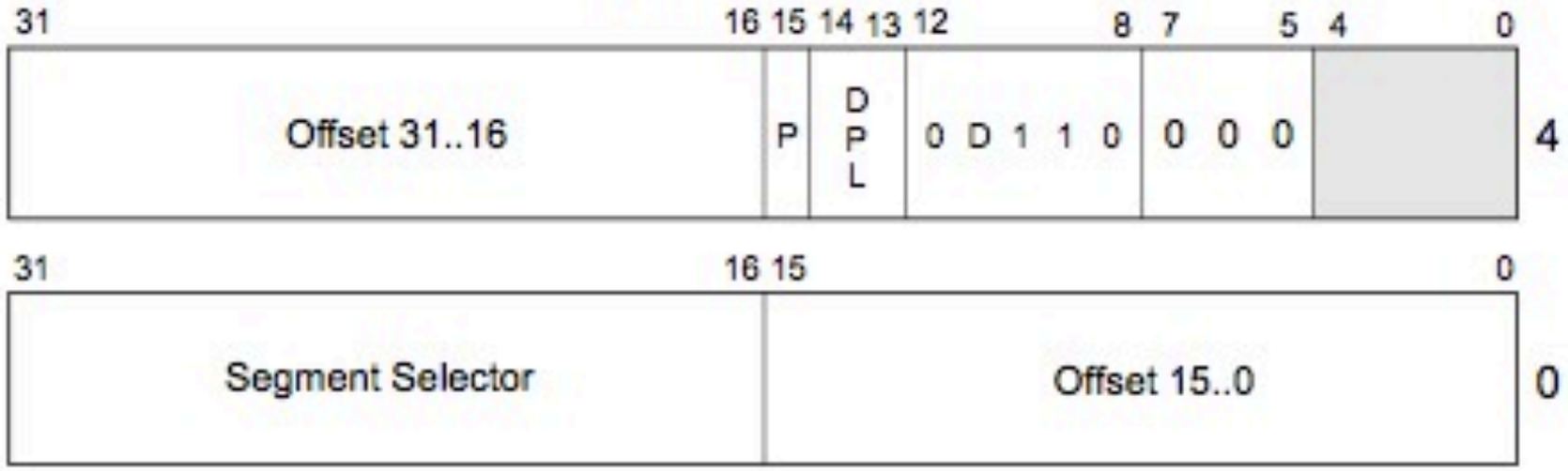
IDT entries:

...
8: #DF
...
14: #PF
...

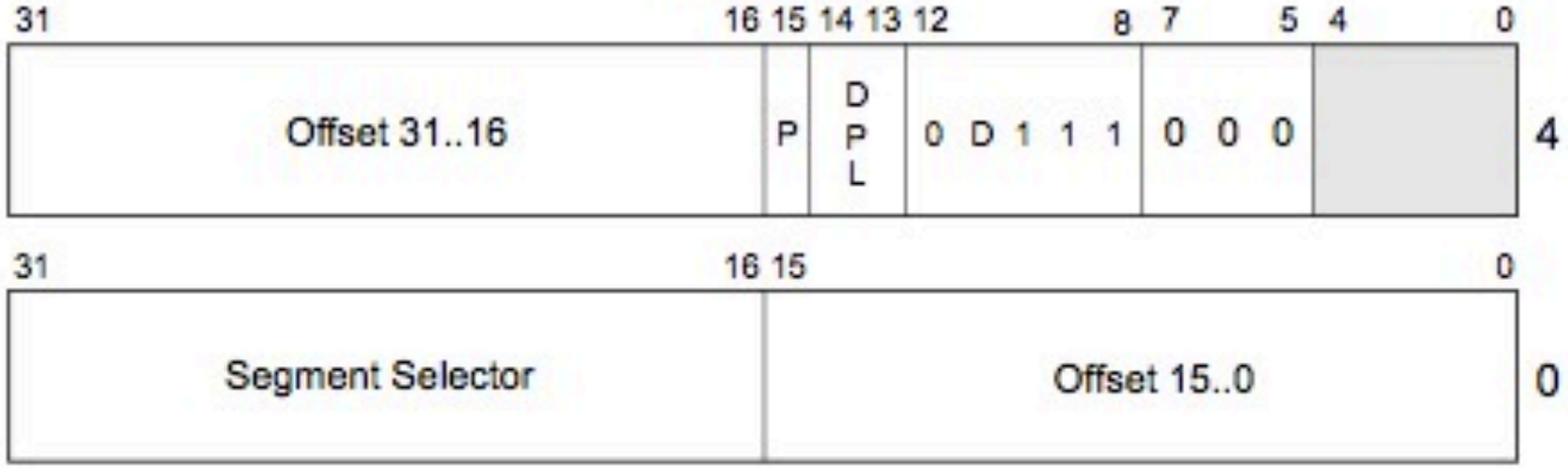
Task Gate



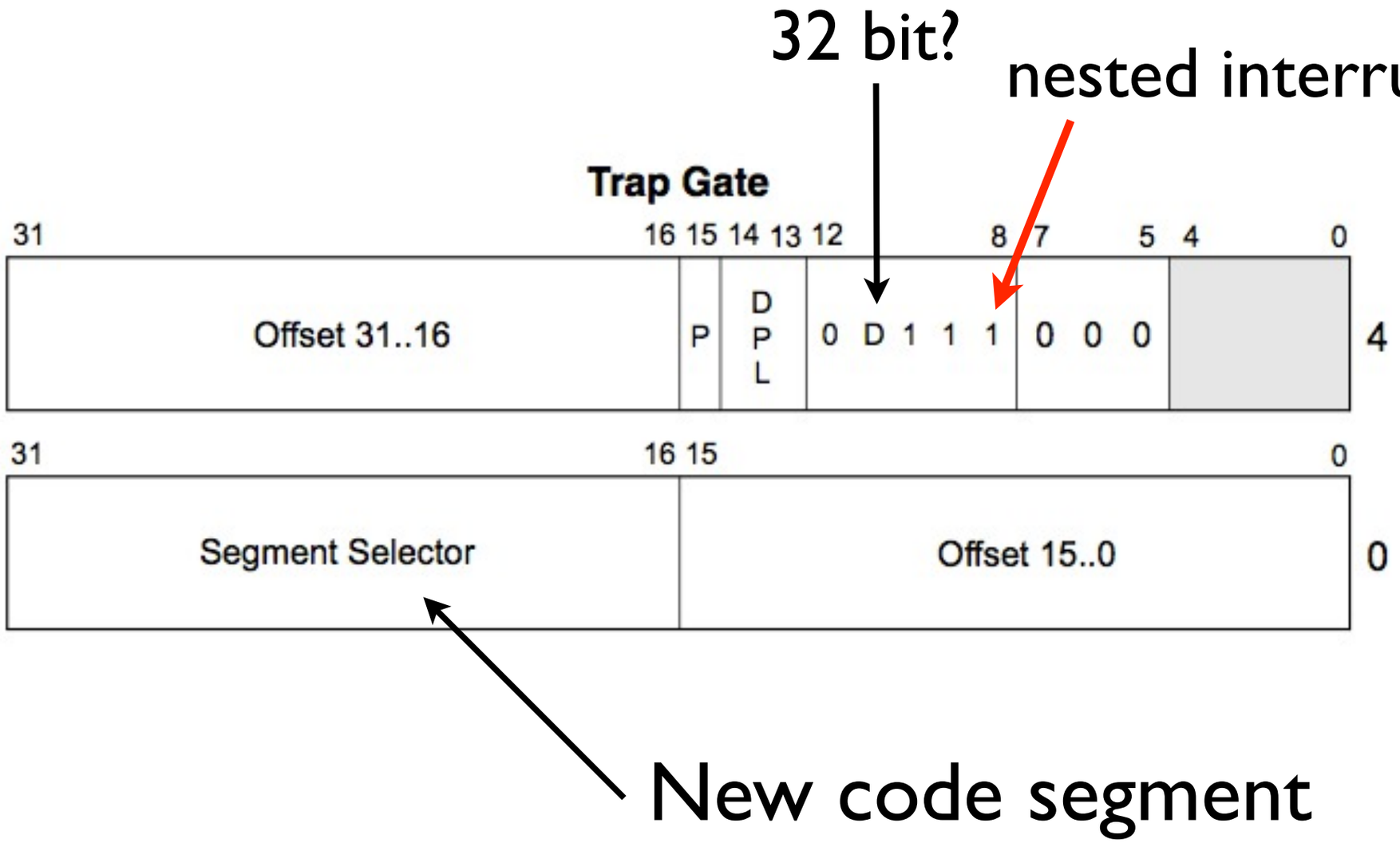
Interrupt Gate



Trap Gate

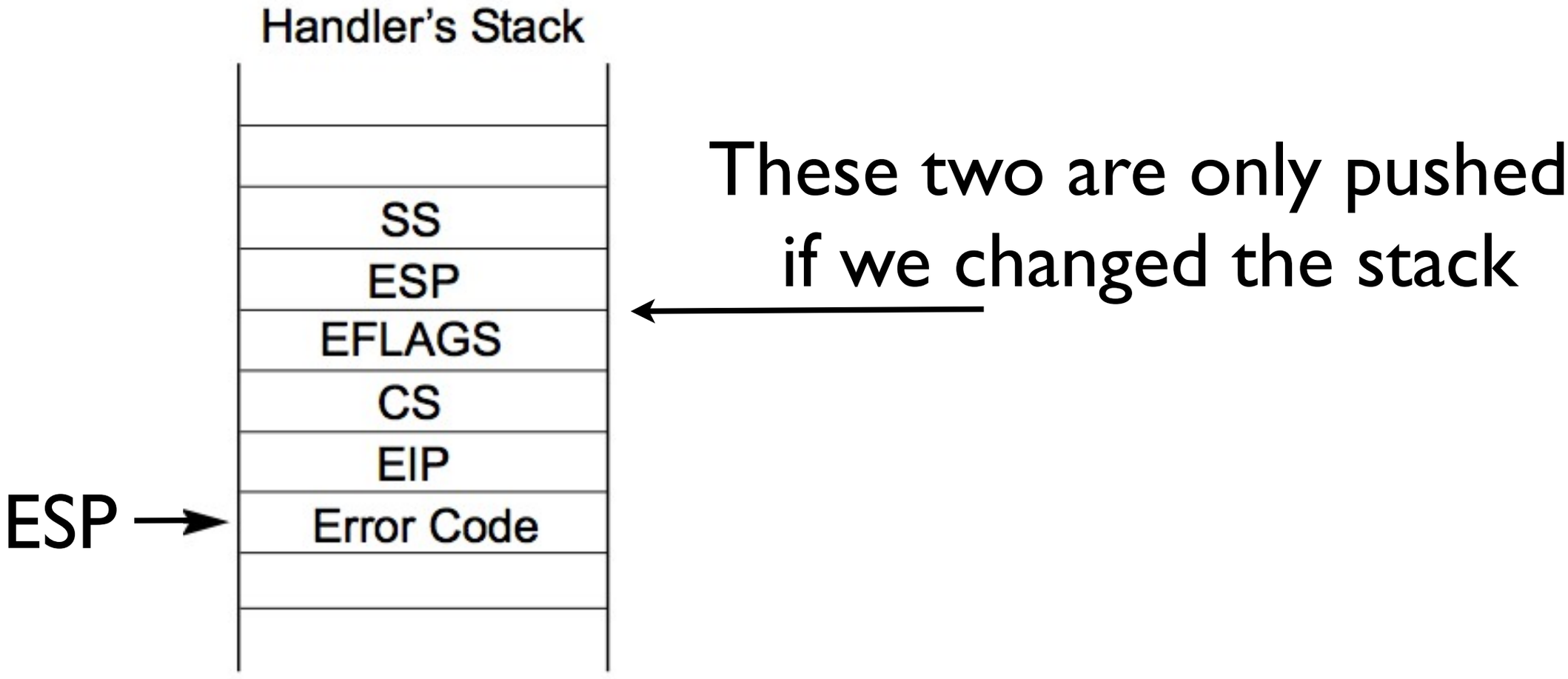


Call through a Trap Gate



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"



"IRET" instruction can return from this

What if this fails?

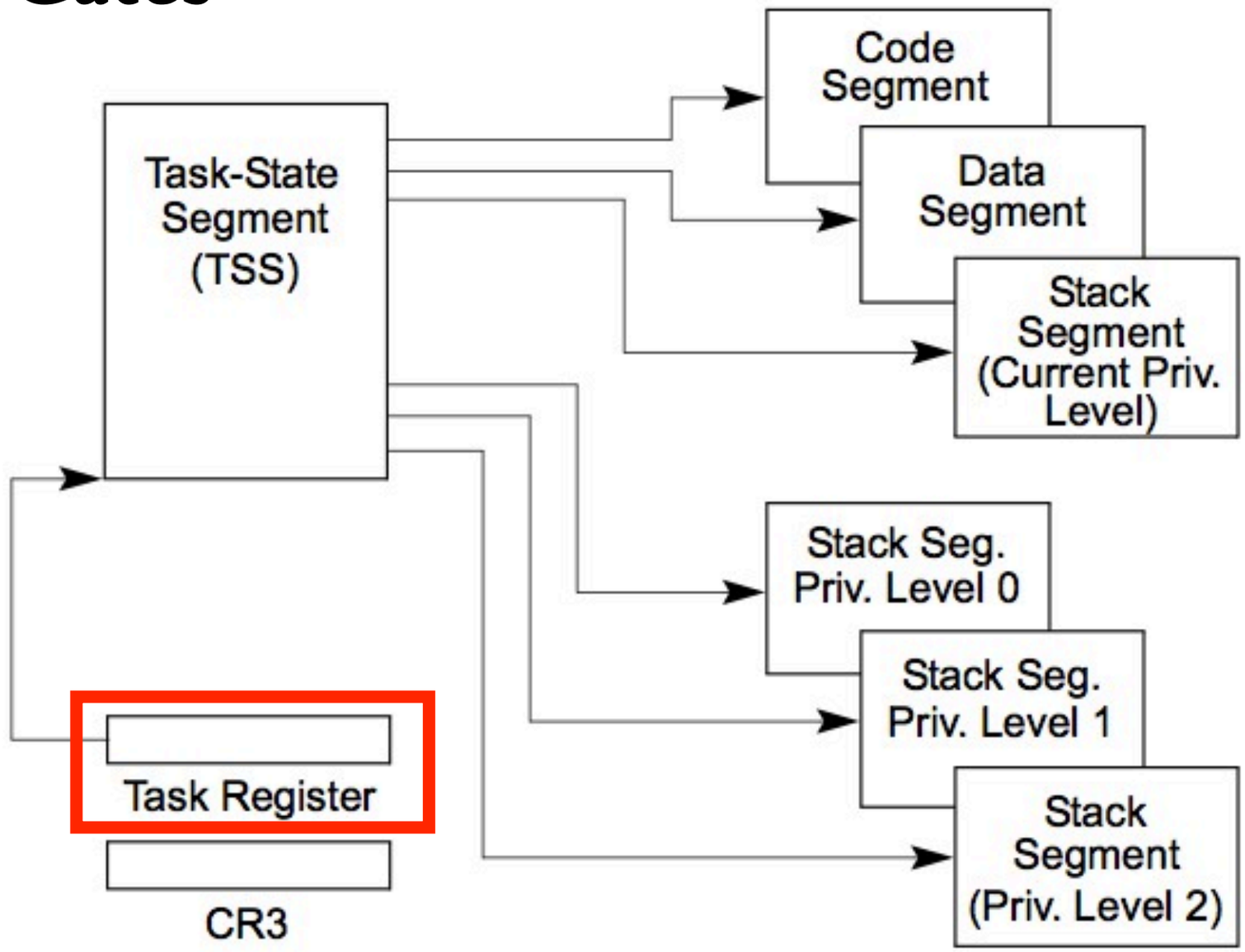
- Stack invalid?
- Code segment invalid?
- IDT entry not present?

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)

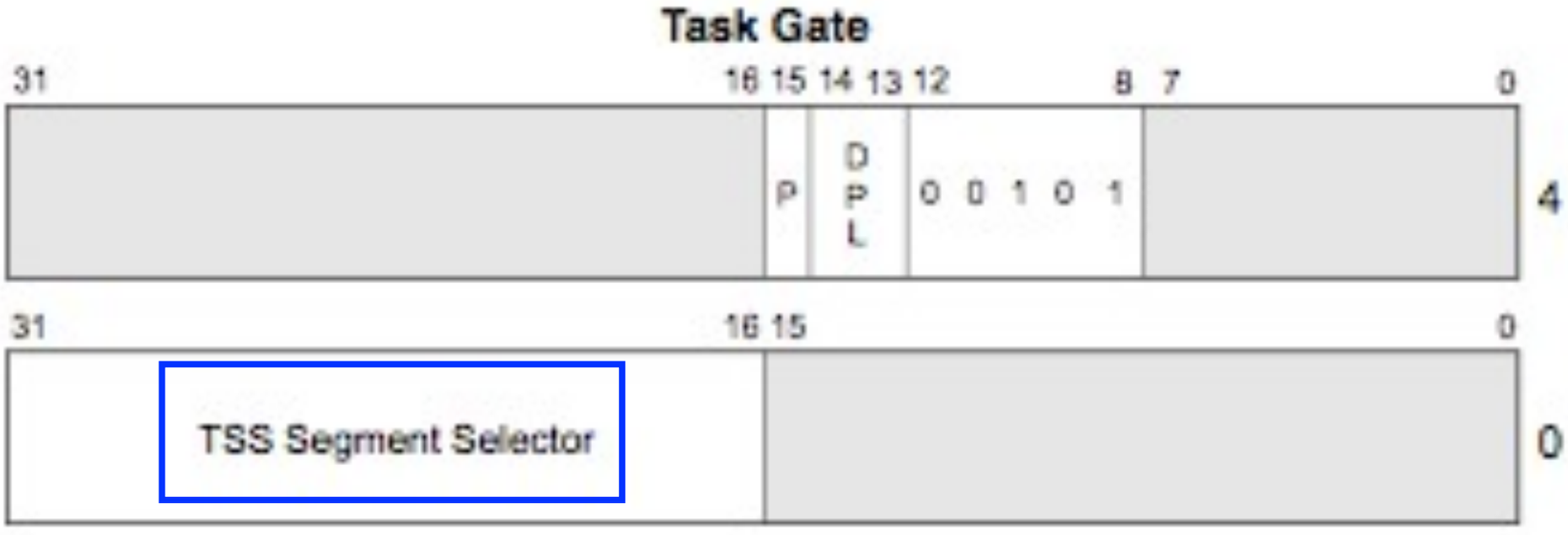
Hardware Task Switching

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



Task gate

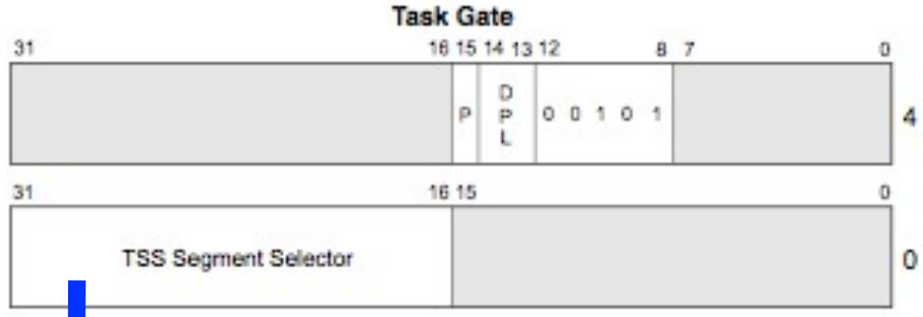
- (unused) mechanism for **hardware** tasking
- Reloads (nearly) all CPU state from memory
- Task gate causes **task switch** on **trap**



IDT-> GDT->TSS

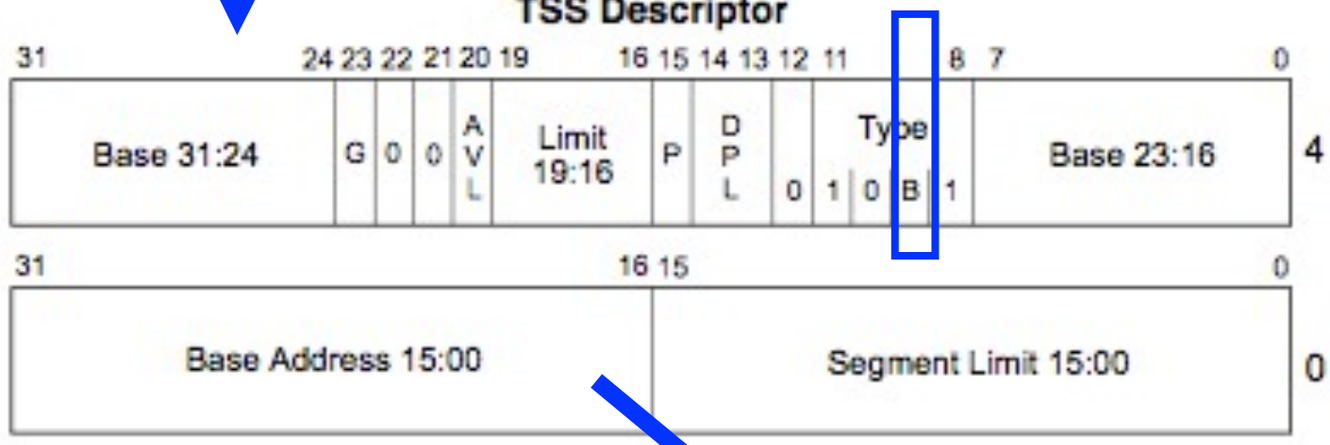
It still pushes the error code

IDT



(addressed indirectly through GDT)

GDT



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

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? Implement by **double fault** when SP cannot be decremented

Dramatis Personae I

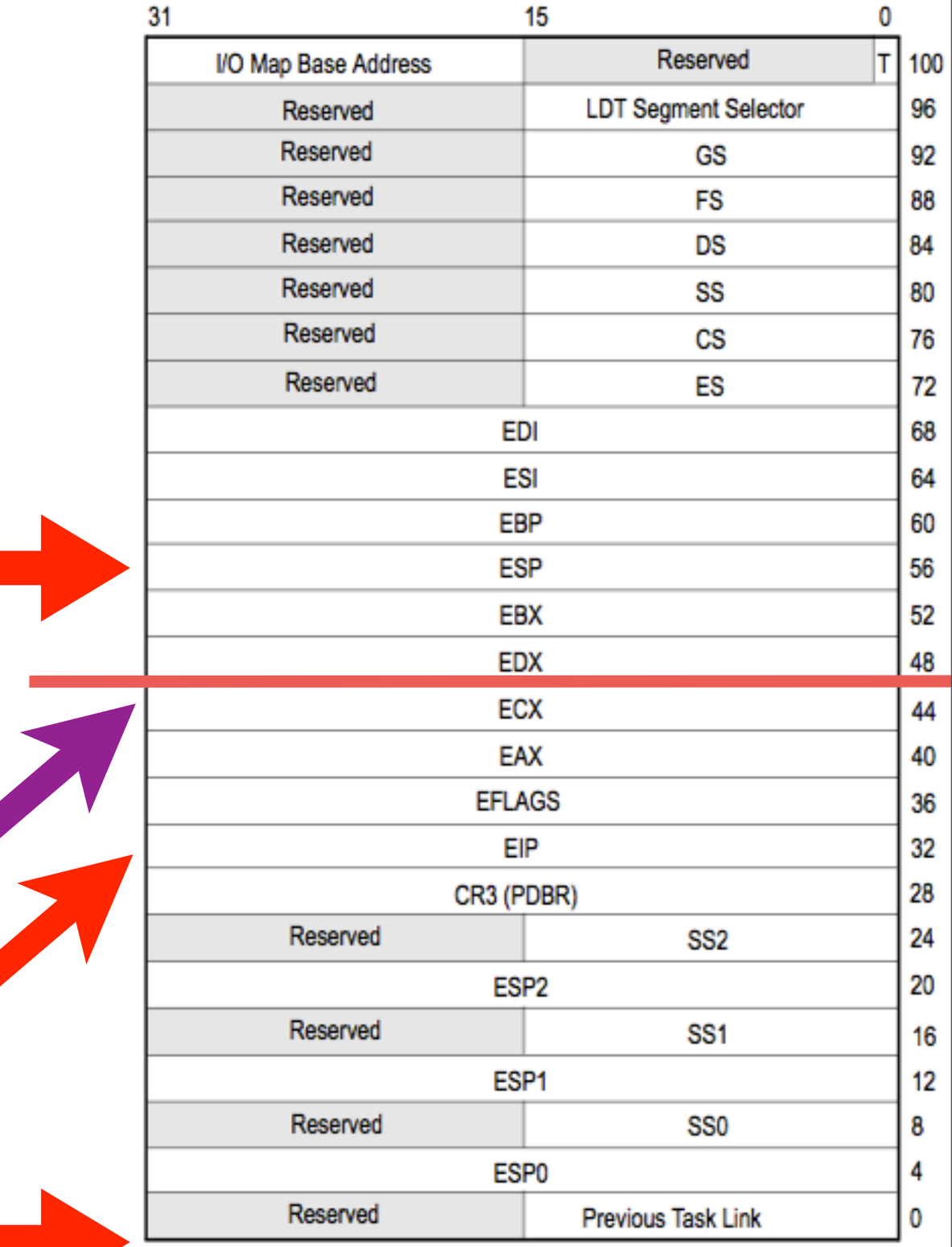
- One GDT to rule them all
- One TSS Descriptor per instruction, aligned with the end of a page
- IDT is mapped differently, per instruction
- A target (branch-not-taken) in Int 14, #PF
- B target (branch taken) in Int 8, #DF

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

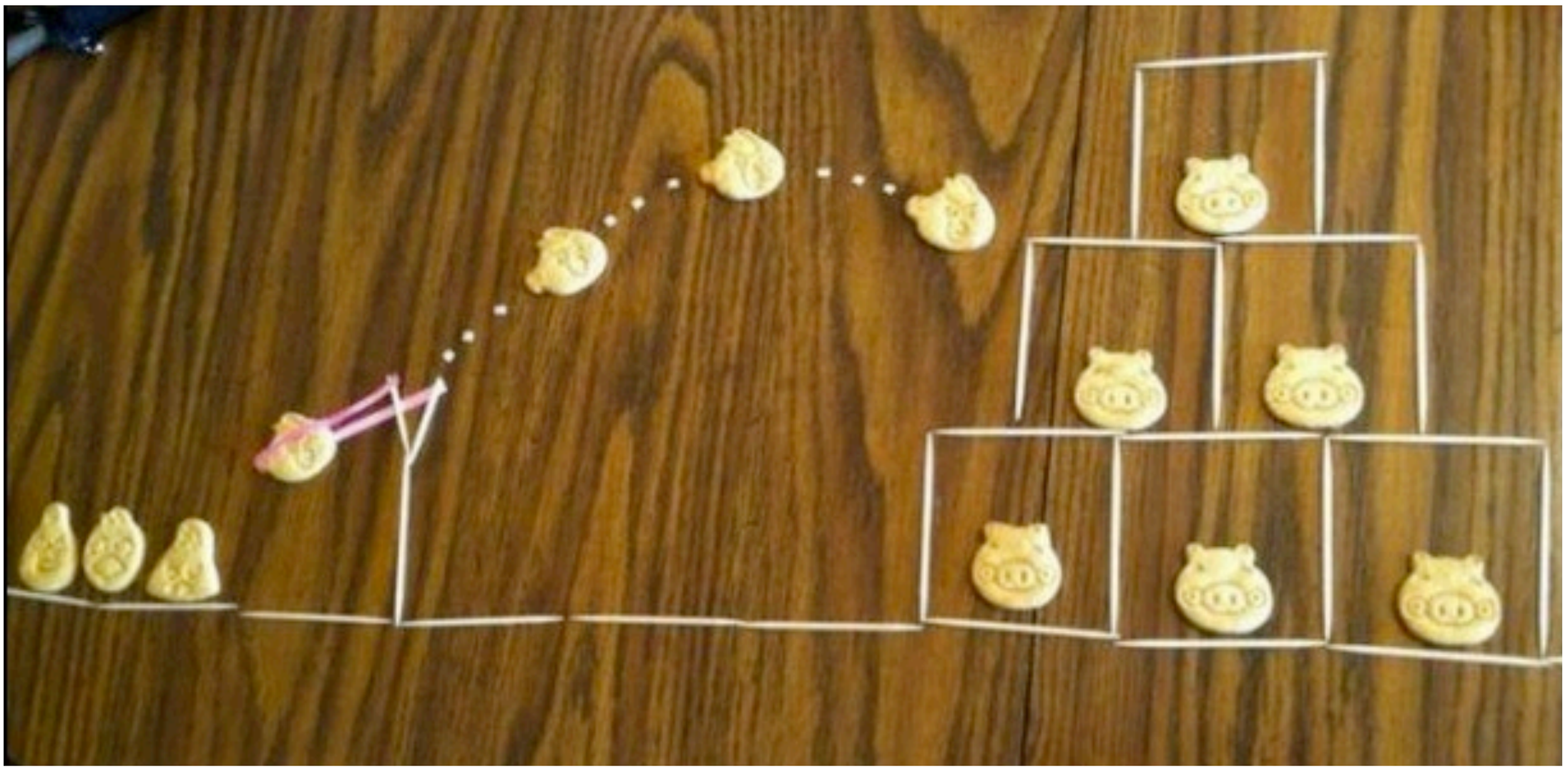
- We split these TSS across a page boundary
- Variables are stack pointer entries in a TSS
- Upper Page: ESP and segments
- Lower Page: EAX, ECX, EIP, CR3 (page tables)

Labels: A, B, C, ...

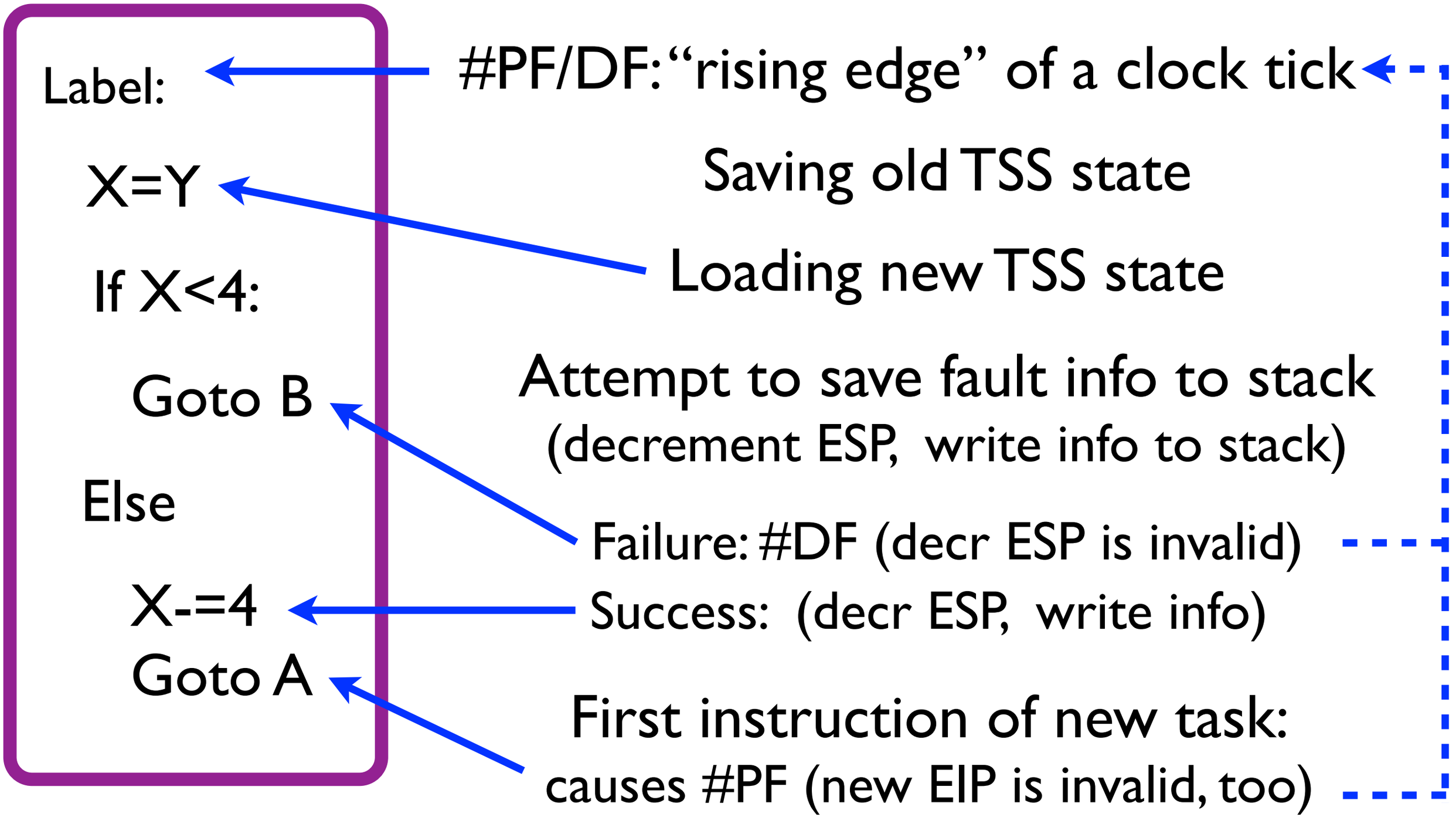


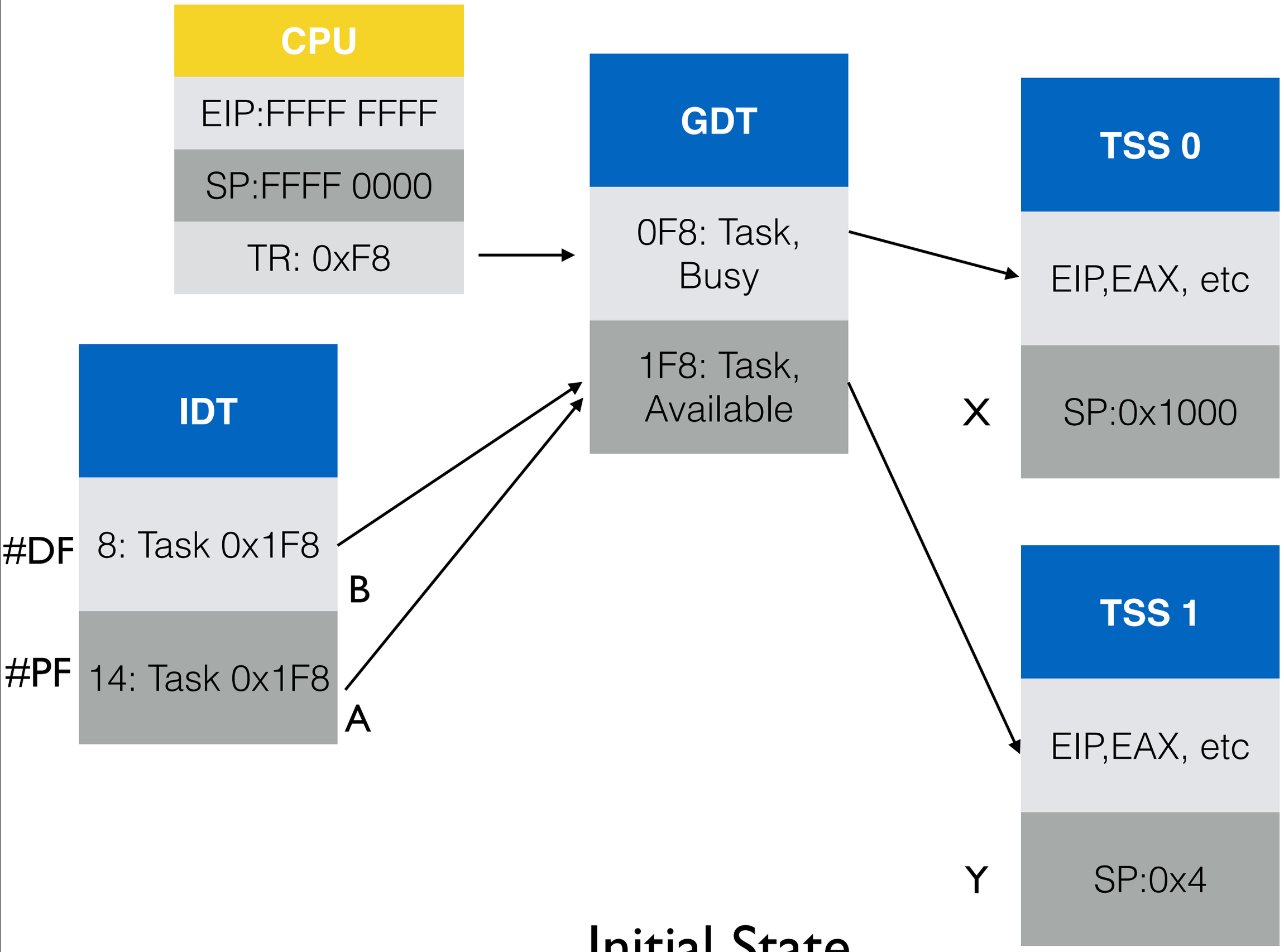
Let's step through an instruction

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

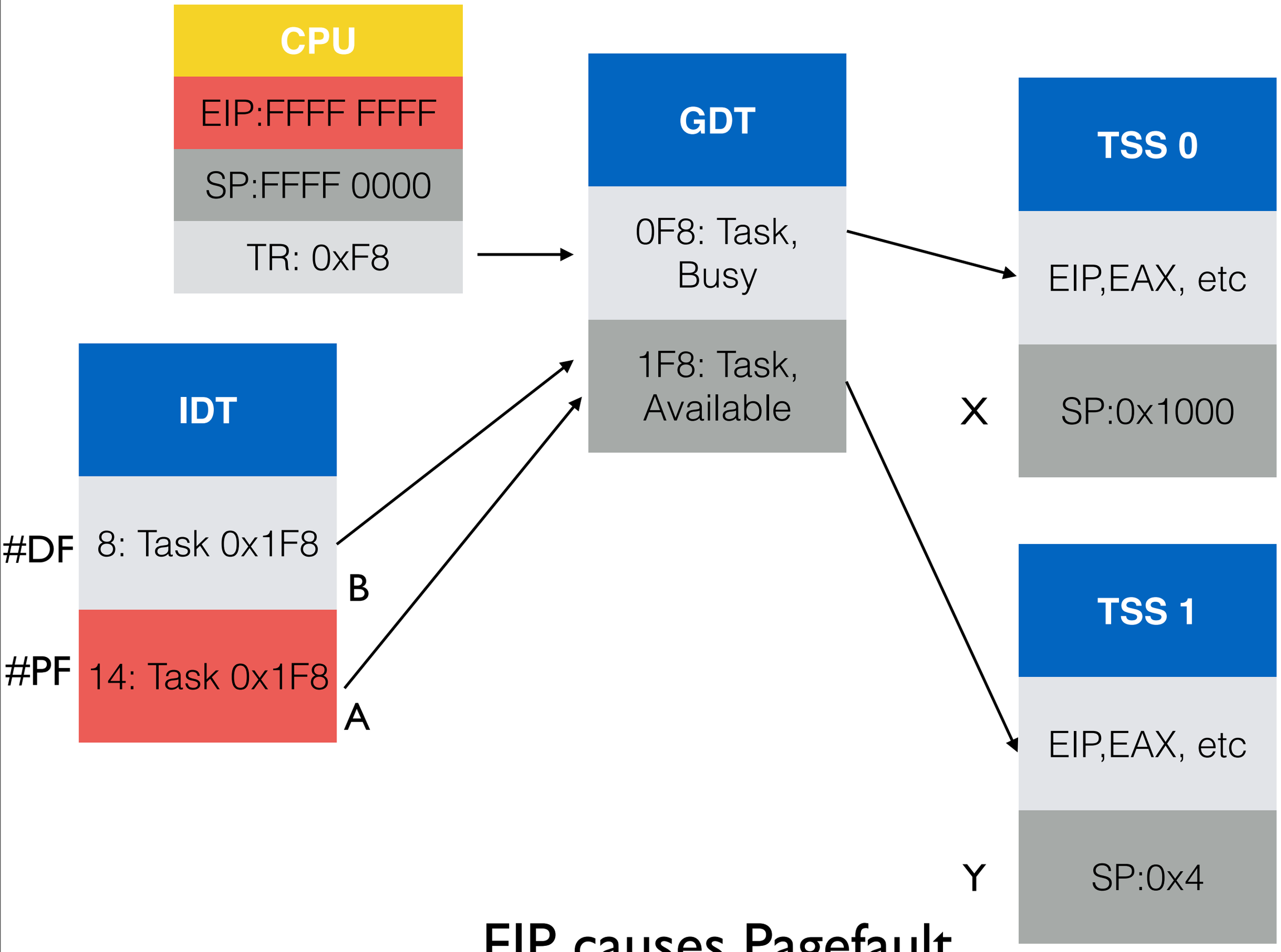


Instruction by the numbers (or, "PFLA fetch-decode-execute" loop)

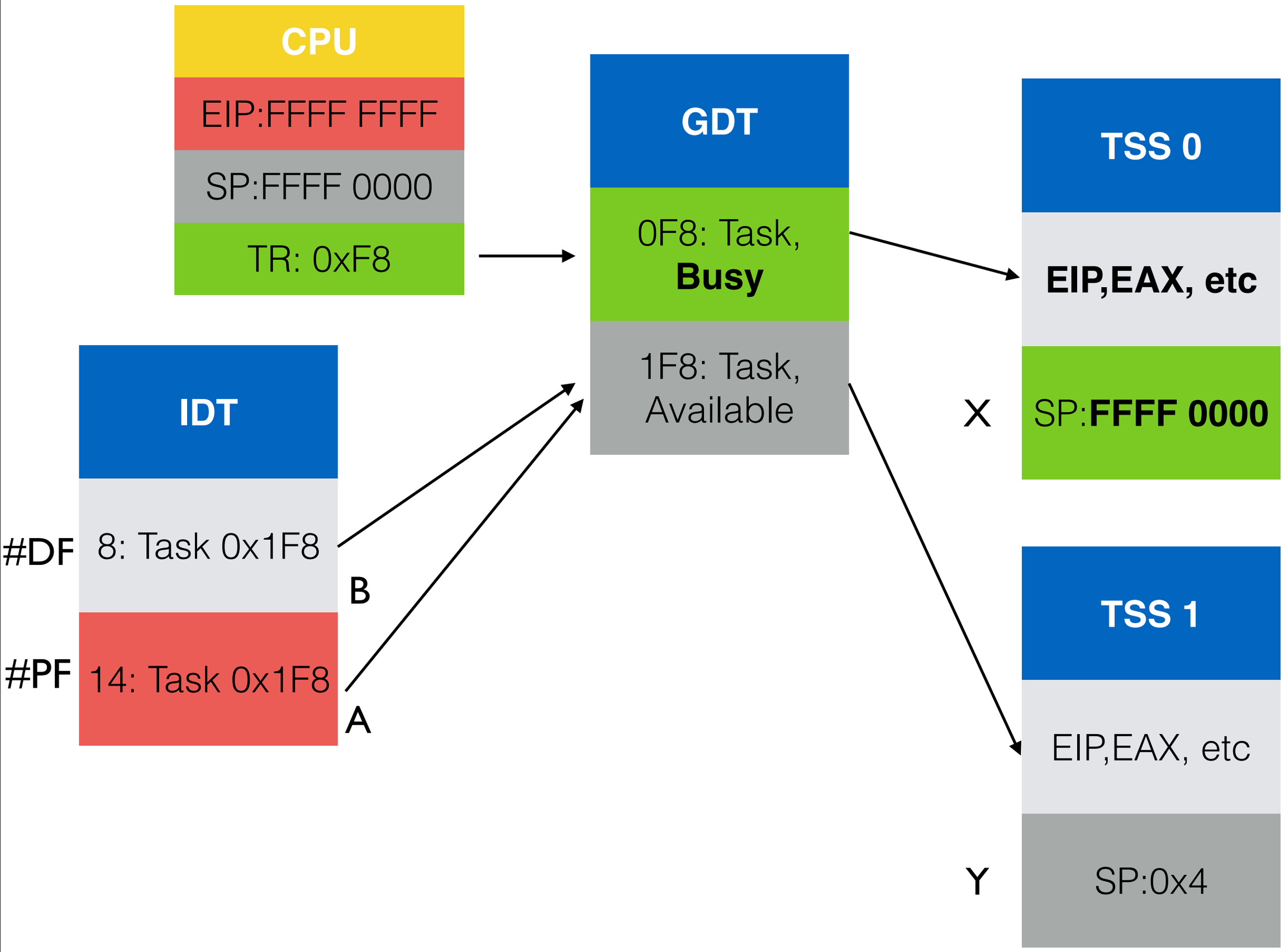




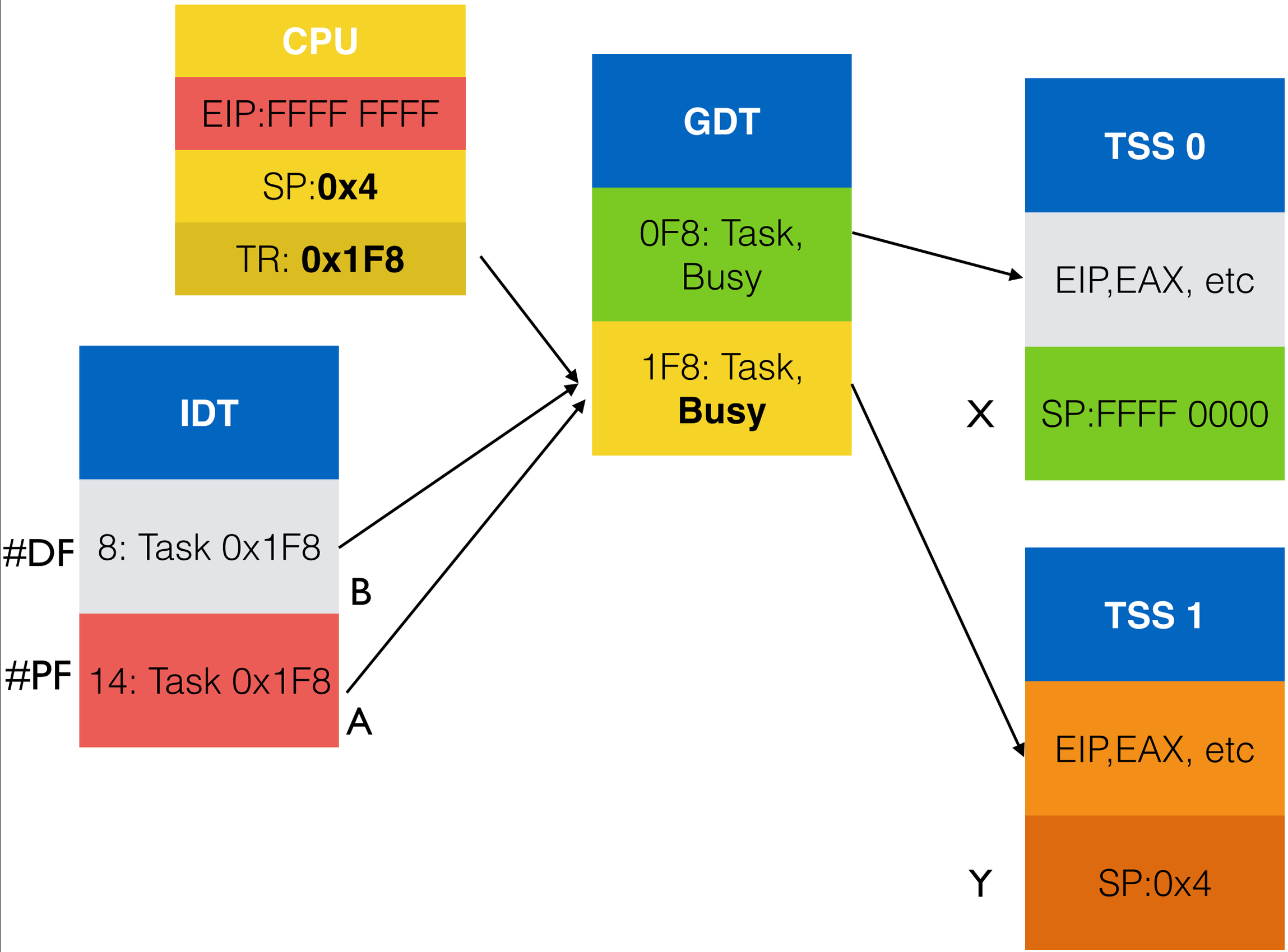
Initial State



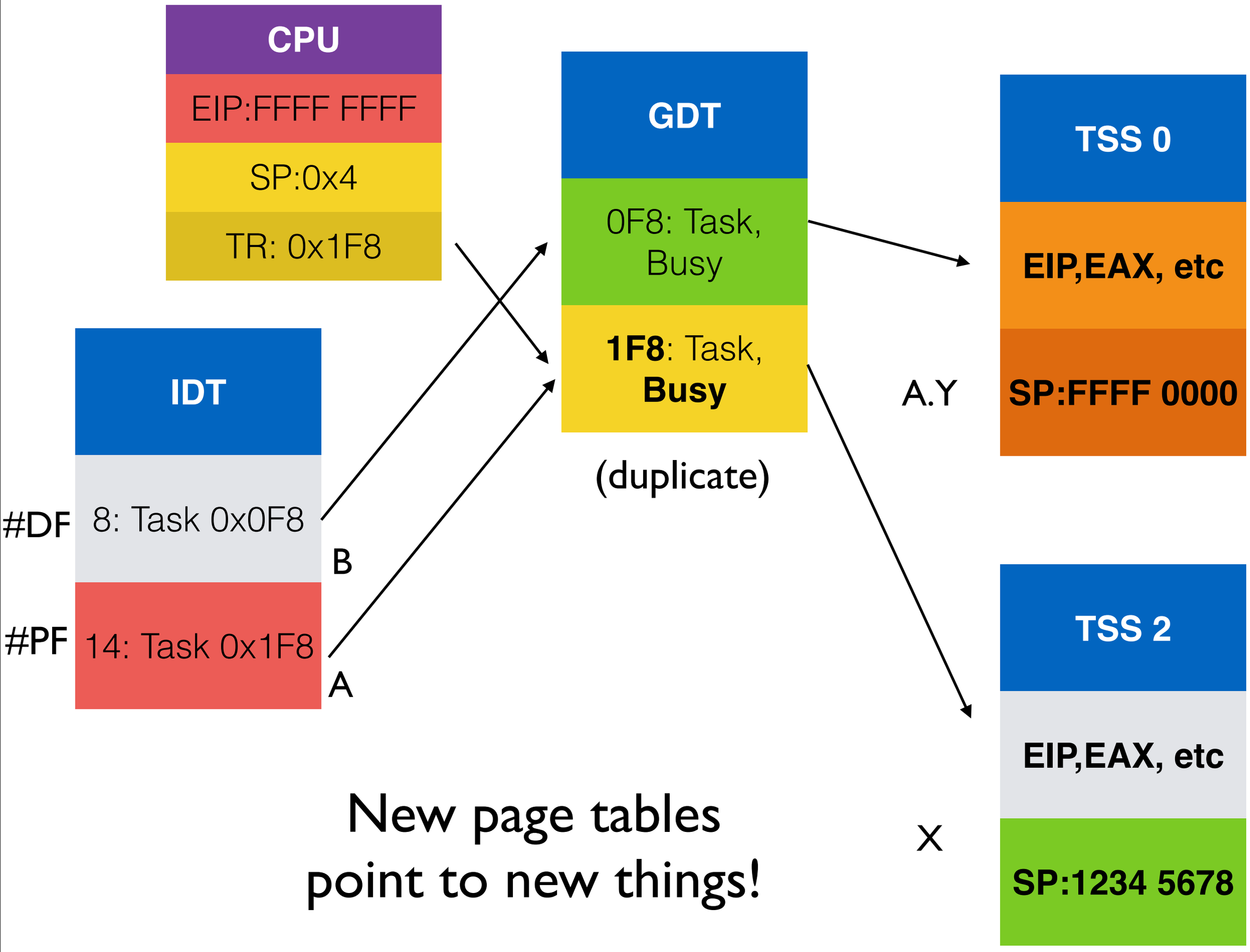
EIP causes Pagefault



CPU state is saved to current task



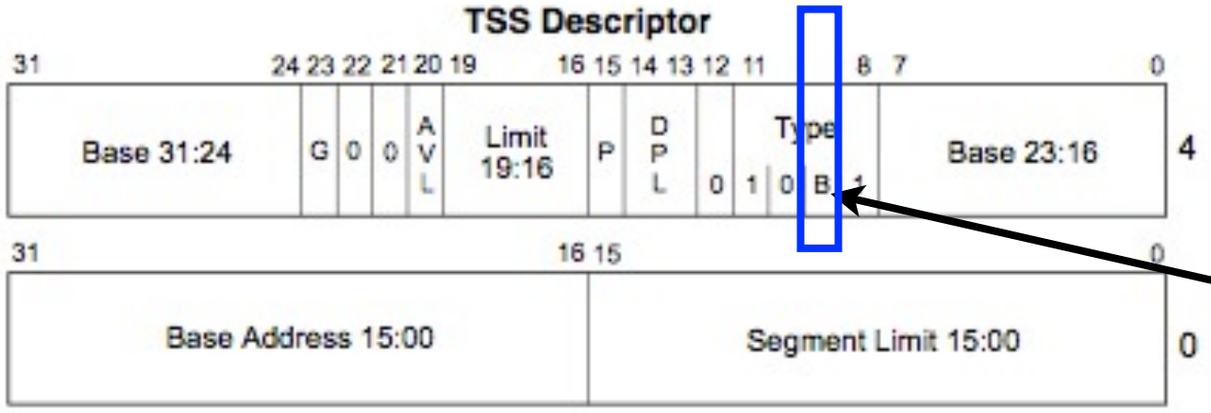
CPU loads interrupt task



New page tables
point to new things!

“Implementation Problem”

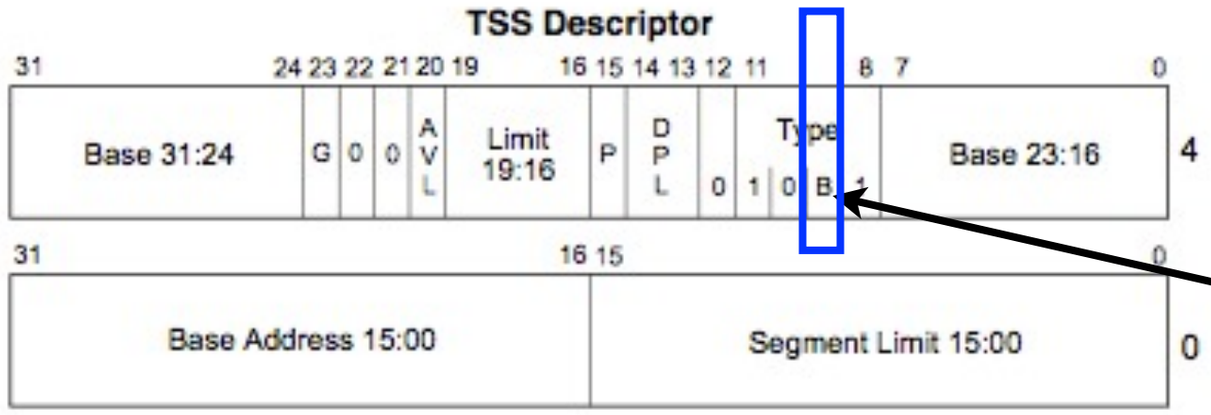
1 bit(ch) of a bit(ch)



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)



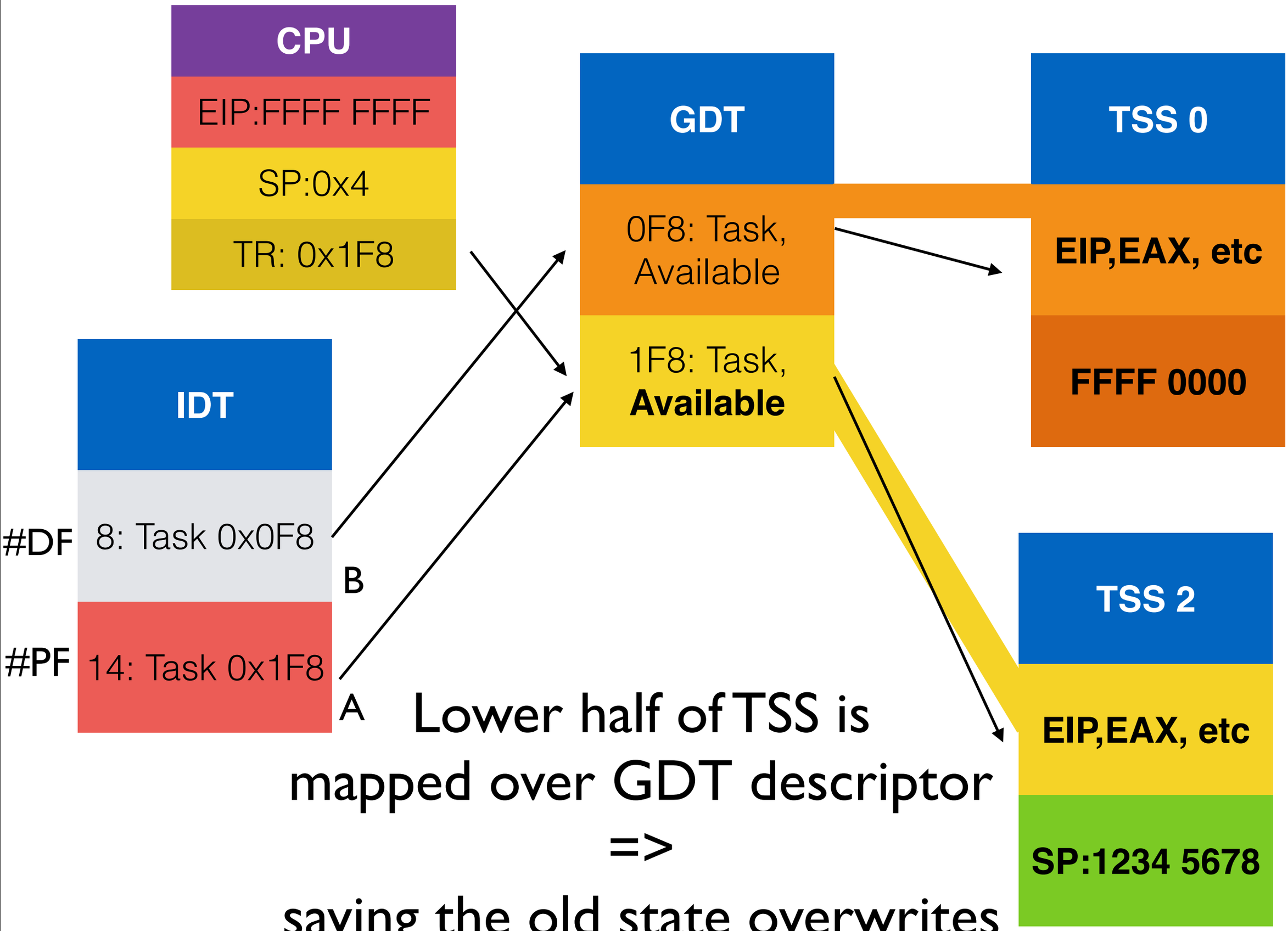
AVL	Available for use by system software
B	Busy flag
BASE	Segment Base Address
DPL	Descriptor Privilege Level
G	Granularity
LIMIT	Segment Limit
P	Segment Present
TYPE	Segment Type

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.

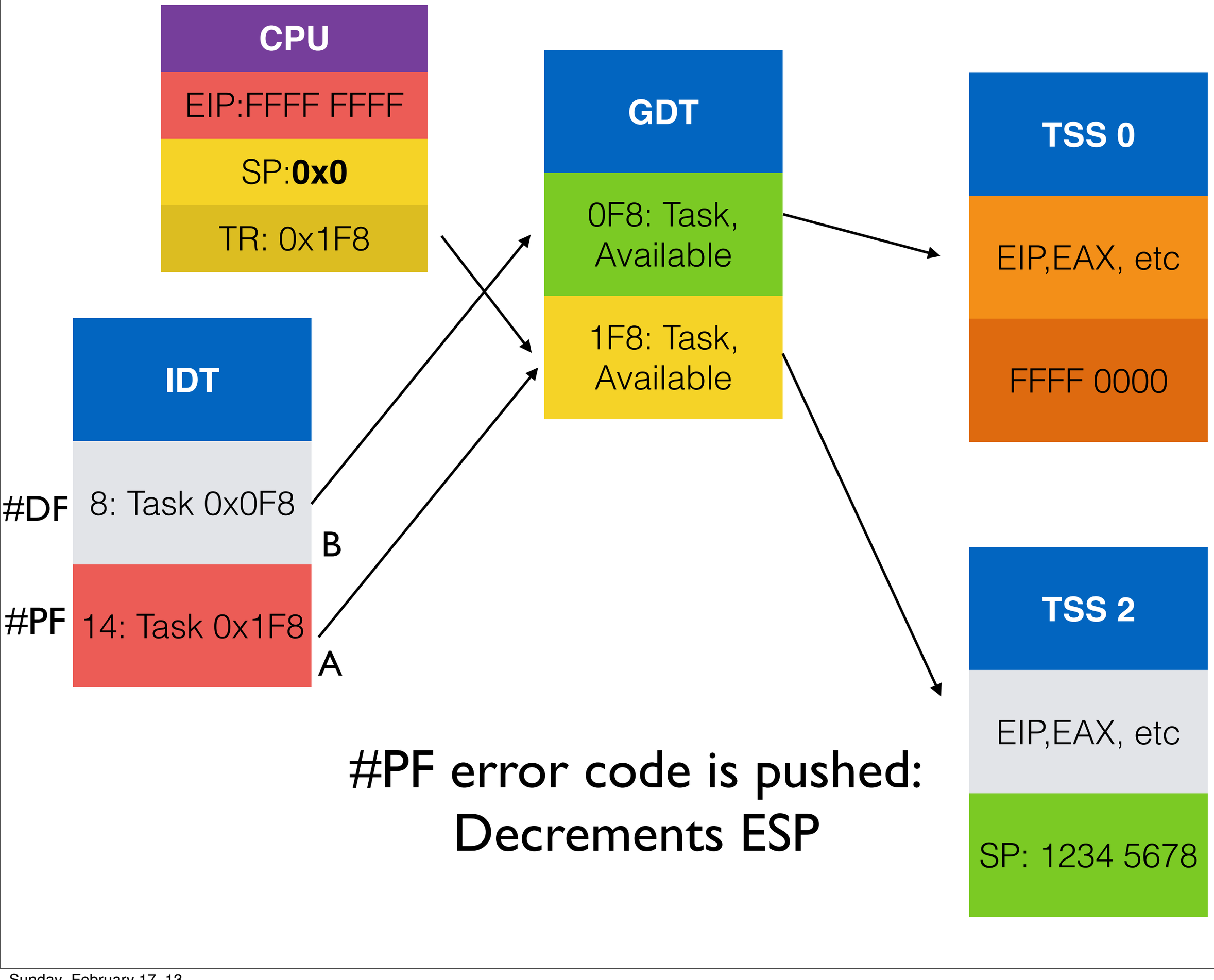
Dealing with that bit needs a nuclear option...



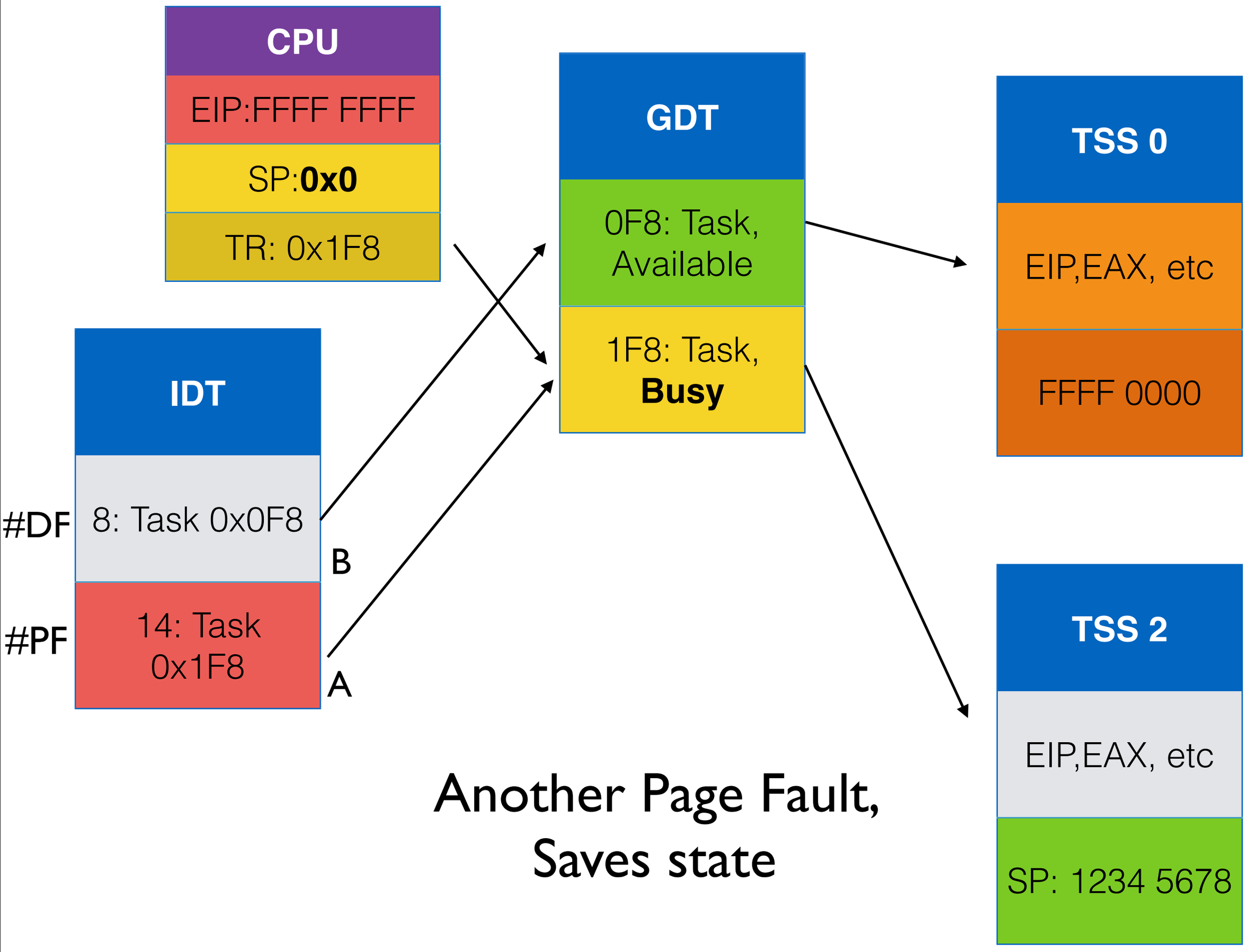
Lower half of TSS is mapped over GDT descriptor

=>

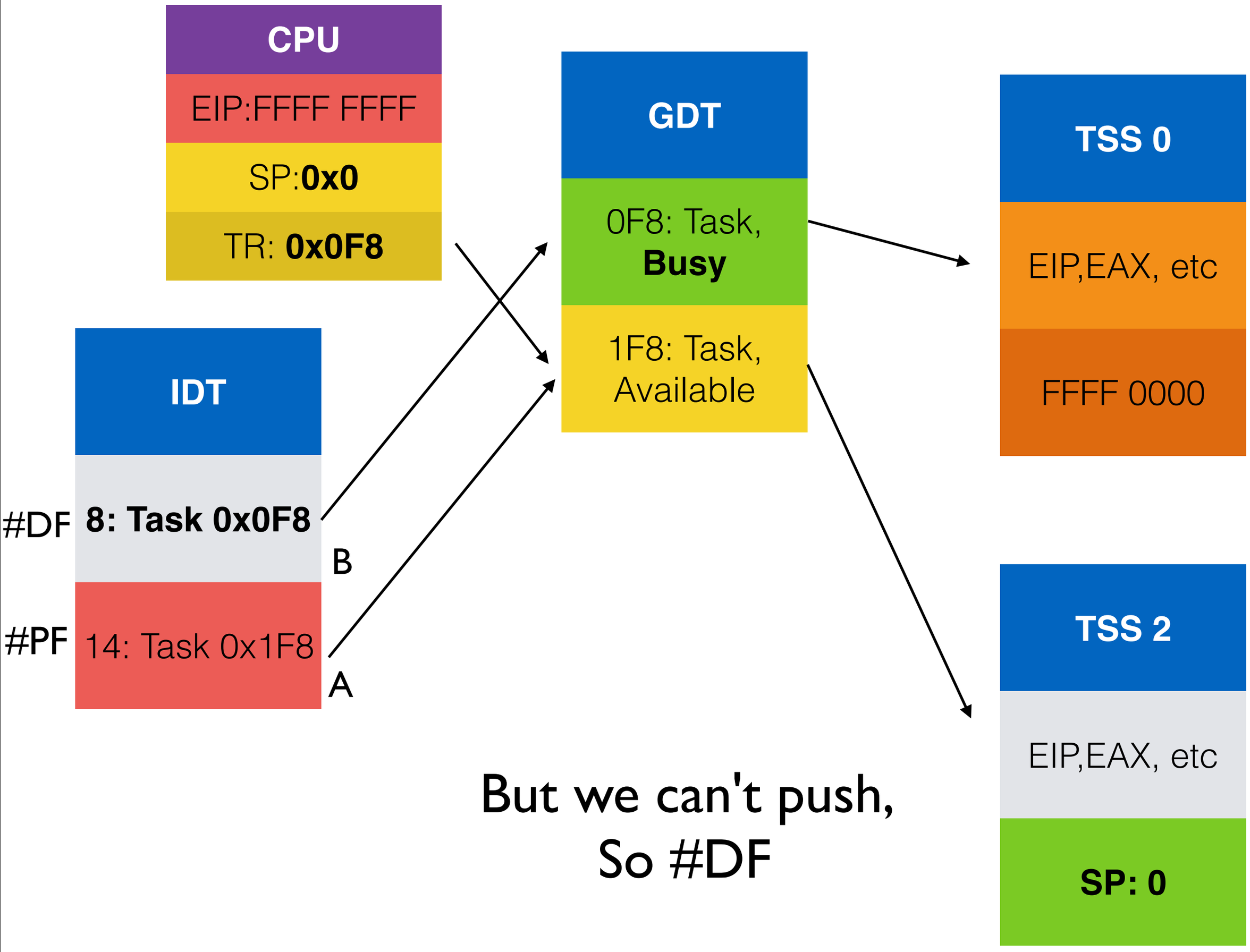
saving the old state overwrites the GDT entry busy bit!

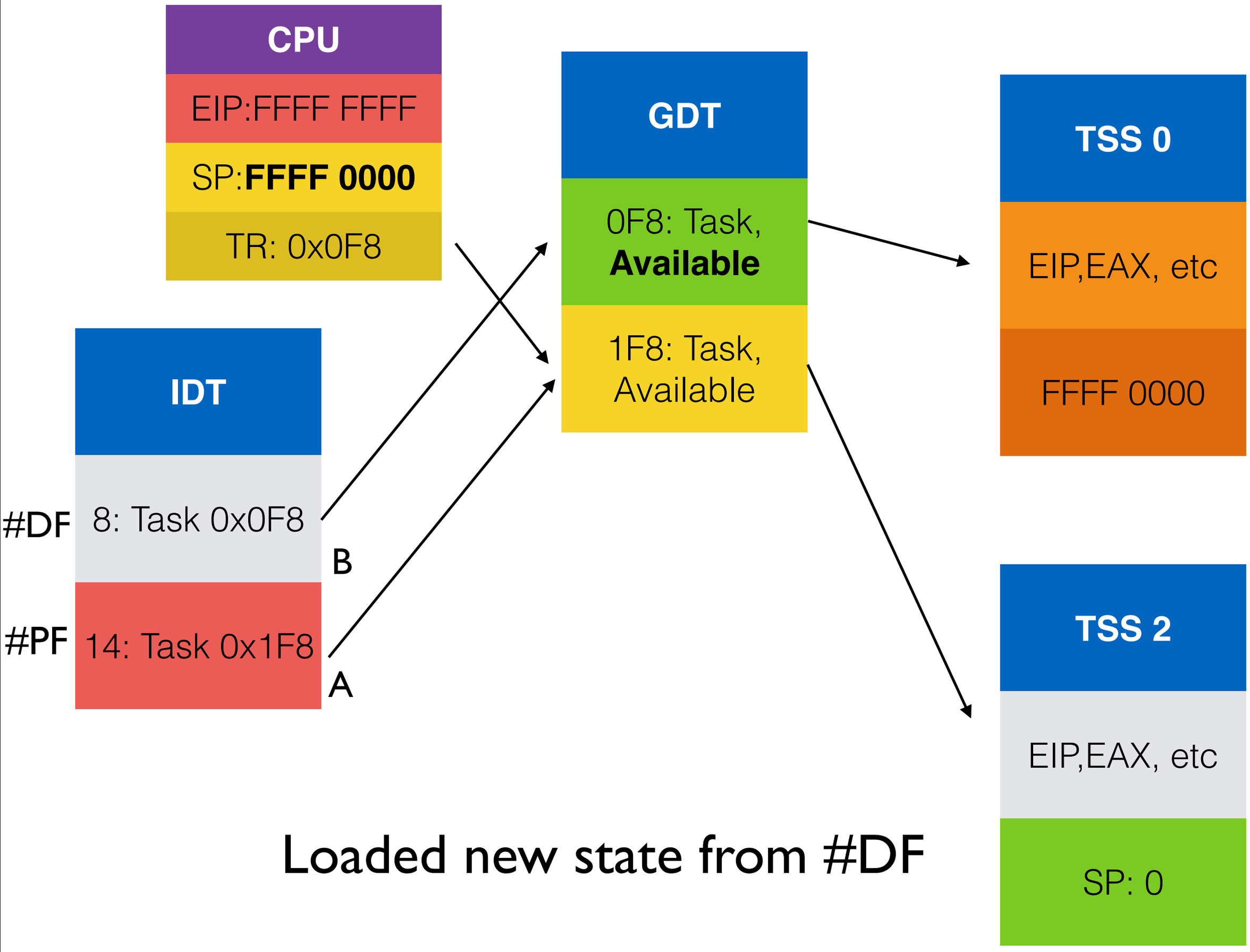


#PF error code is pushed: Decrements ESP



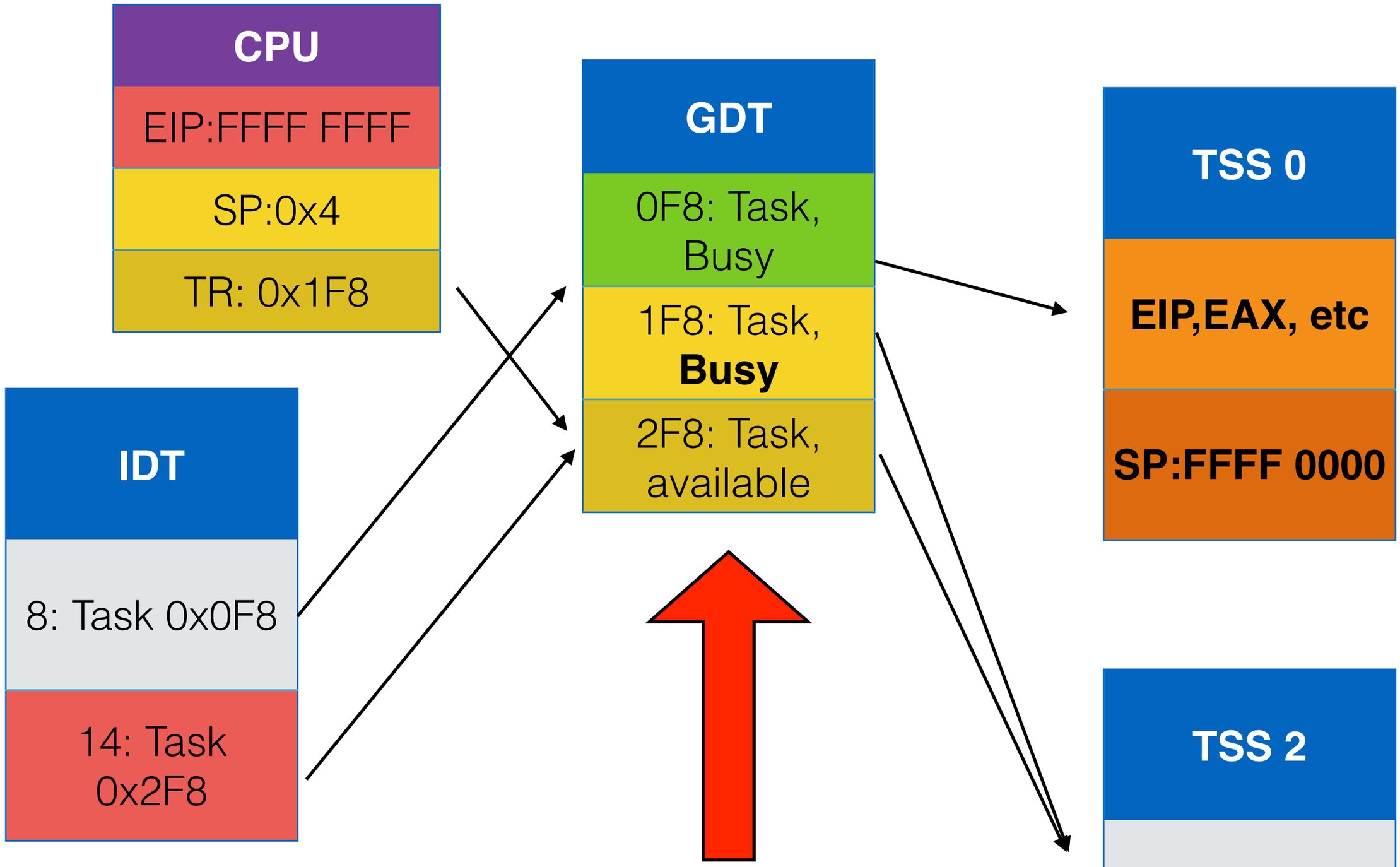
Another Page Fault, Saves state


CPU
EIP:FFFF FFFF
SP:0x0
TR: 0x0F8
GDT
0F8: Task, Busy
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: 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 :)
- Can only use our one awkward instruction
- Can only work with SP of TSS aligned across page (very limited coverage of phys. mem)
- No two double faults in a row, so we have to insert dummy instructions occasionally


In Soviet Russia,
Red Pill takes you

No publicly available simulator implements this correctly

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, better debugger
- See how it works on different hardware?

Go forth and play!

- Twitter: @julianbangert, @sergeybratus
- github.com/jbangert/trapcc

“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