

Fundamentos de Segurança Informática (FSI)

2022/2023 - LEIC

Manuel Barbosa
mbb@fc.up.pt

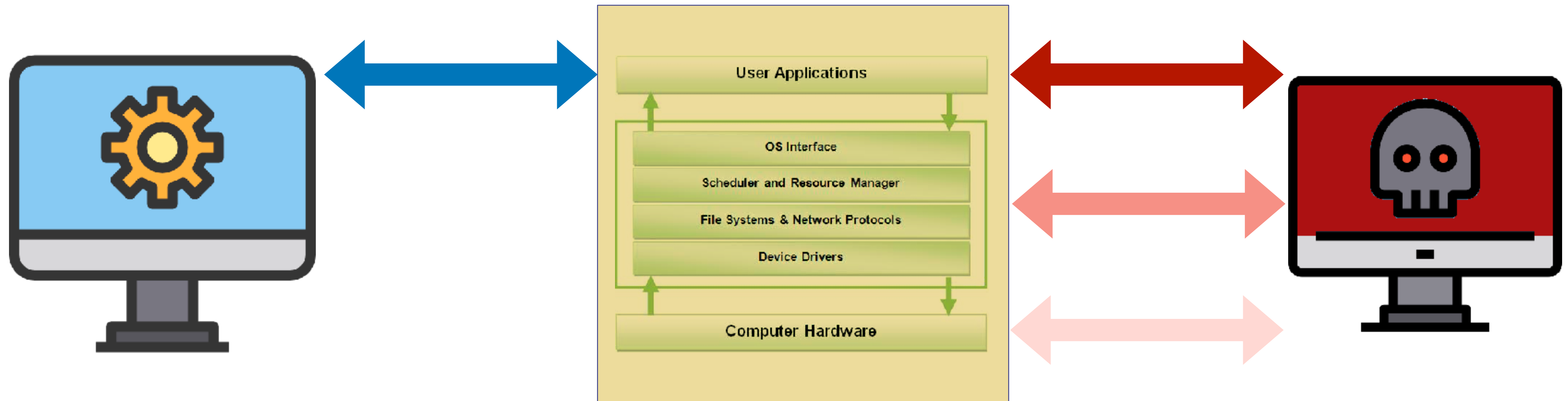
Hugo Pacheco
hpacheco@fc.up.pt

Aula 8

Segurança de Sistemas

(Parte 2)

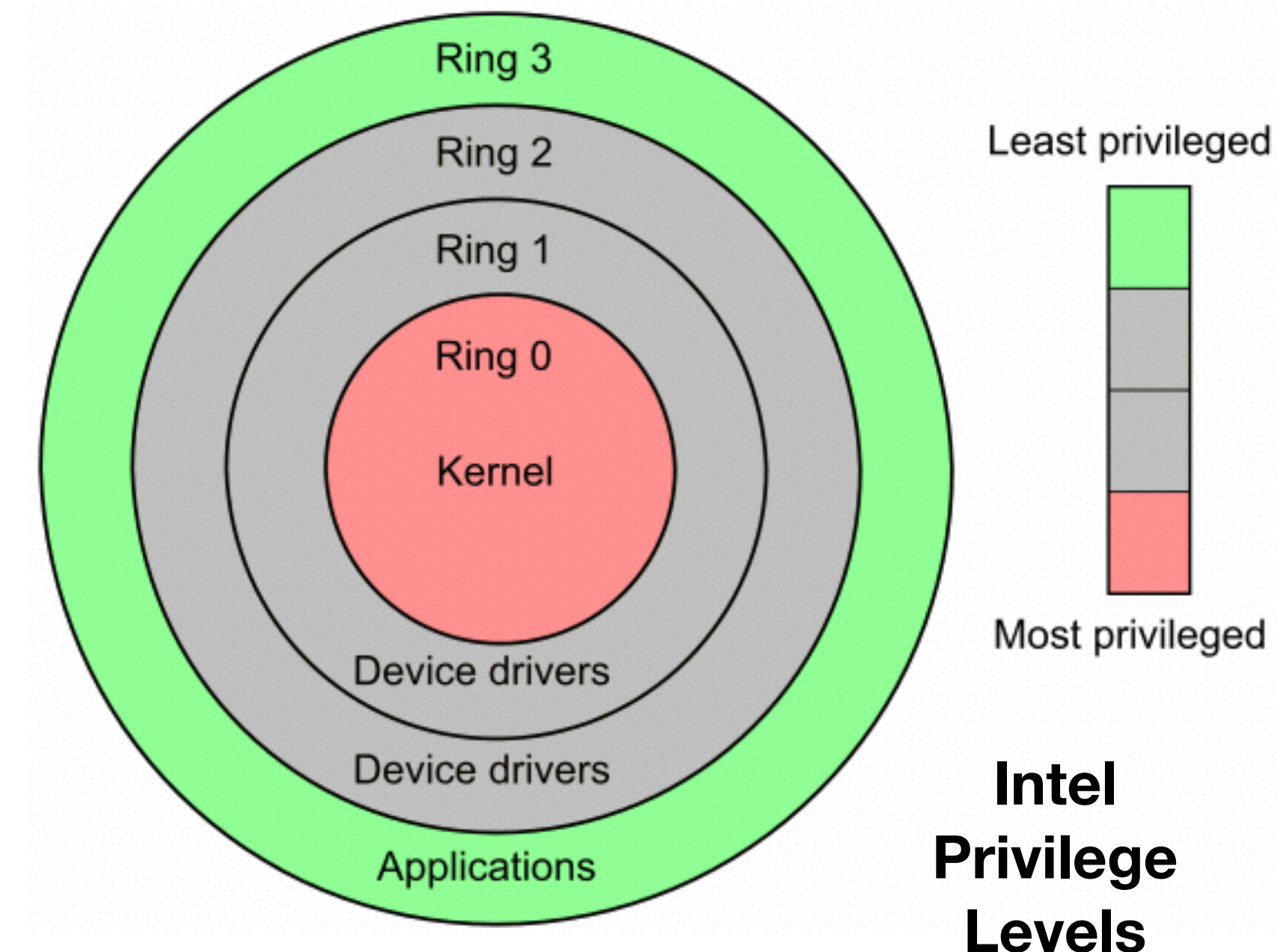
Segurança de Sistemas



- Processos honestos correm “isoladamente” (“garantia” do SO)
- Processos desonestos tentam quebrar isolamento e/ou usurpar recursos
- Hierarquia de confiança: níveis crescentemente menos confiáveis

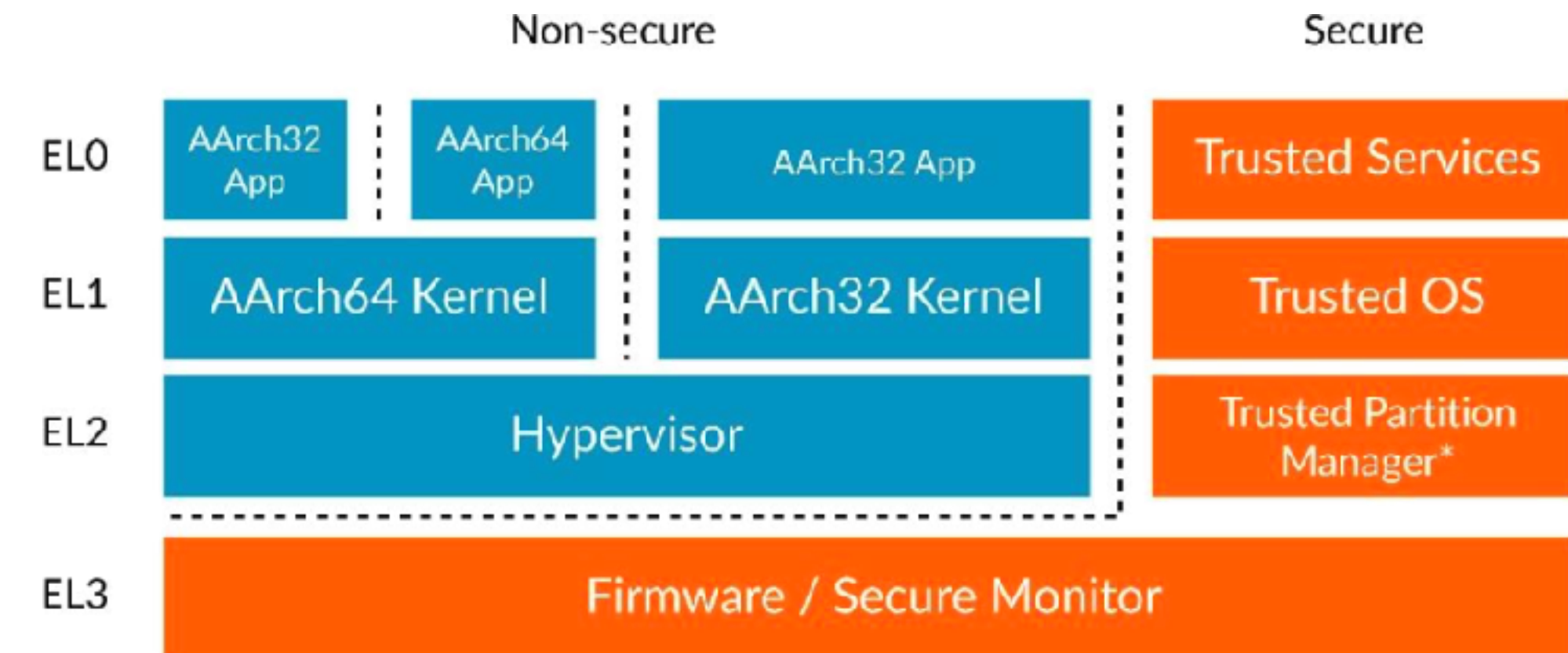
Kernel

- O **Kernel** é a parte do sistema operativo que desempenha as operações mais críticas (mediação completa 📖📖):
 - o processador está em **kernel mode** e todas as operações são permitidas a esse código
 - os processadores permitem geralmente definir vários níveis de privilégio (ring em Intel)
 - em muitos casos usam-se apenas dois: **kernel mode** e **user mode**
 - o código em **user mode** não tem acesso directo aos recursos do sistema
 - qualquer troca de informação entre os dois níveis faz parte de uma superfície de ataque



Kernel

- O **Kernel** está protegido dos processos em modo utilizador:
 - tem um espaço de memória gerido de forma independente (pelo próprio kernel)
 - o processador garante que apenas código que corre em **kernel mode** pode executar um conjunto de instruções privilegiadas (separação de privilégios 📖)
 - qualquer processo em **user mode** (incluindo device drivers) tem de aceder aos recursos do sistema usando **system calls** (privilégio mínimo 📖)
 - parte do código das **system calls** executa em **kernel mode**!



ARM Privilege Levels

Confinamento (veremos mais à frente)

- Os **pontos de entrada em system calls são críticos**:
 - para causar danos críticos, um processo em modo utilizador tem de o fazer através de uma system call! (relembrar kBouncer)
 - Isto implica implementar sistemas de monitorização e controlo de chamadas ao sistema
 - **Reference monitor** (registo de compromissos 📖):
 - sempre presente (se terminar, têm de ser terminados os processos monitorizados)
 - tem de ser simples para poder ser analisado e validado mais facilmente que o sistema todo



System Calls

- Controlo de processos (e.g., fork, load, execute, wait, alloc, free)
- Acesso a ficheiros (criar, ler, escrever, etc.)
- Gestão de dispositivos (obter acesso, escrever/ler, etc.)
- Configuração do sistema (hora, data, características, estado, etc.)
- Comunicações (estabelecer ligação, enviar/receber mensagens, etc.)
- Proteção (alterar/obter permissões de acesso a recursos)

System Calls

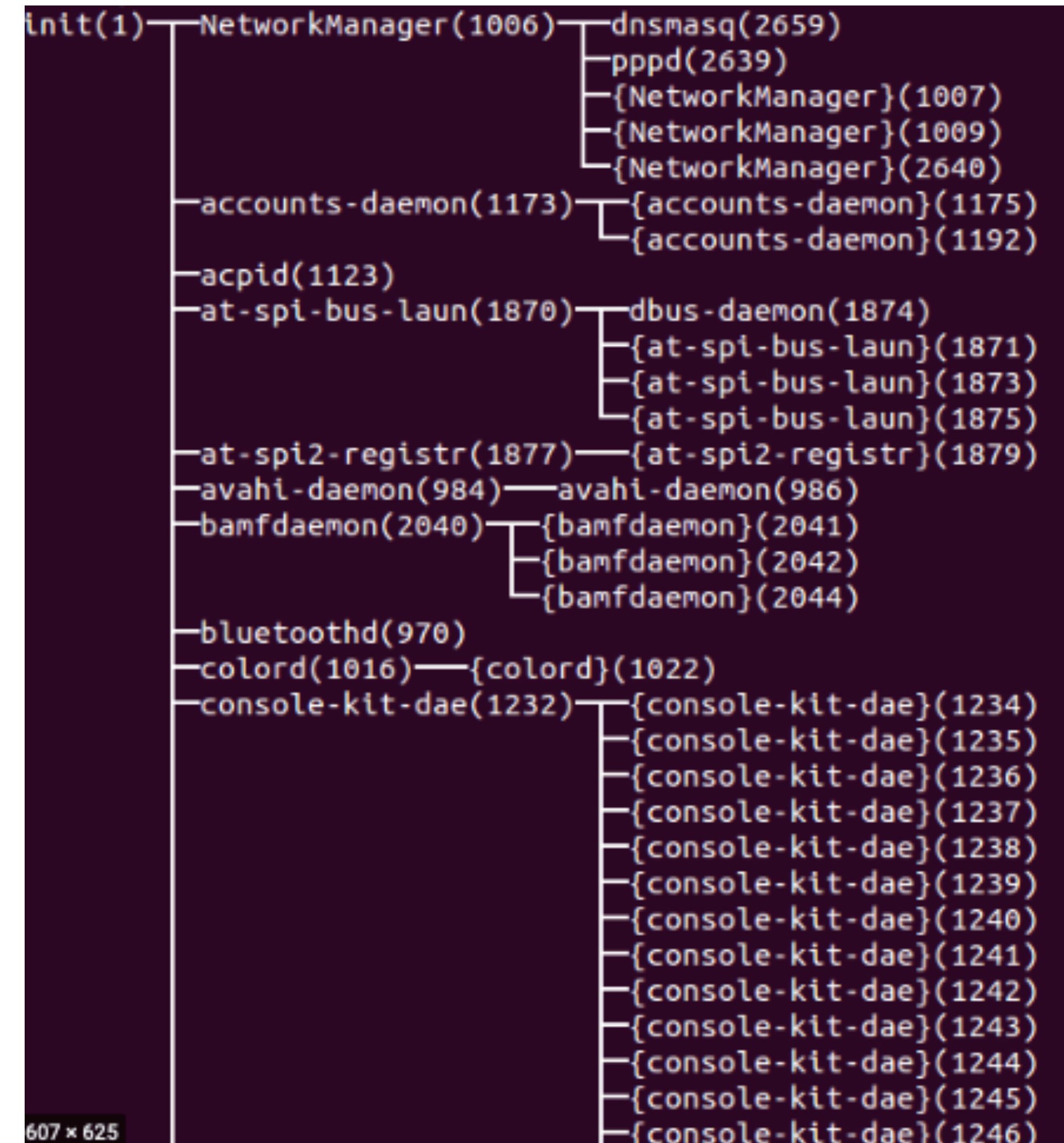
- Para entrar num modo de funcionamento com mais privilégios, o código **user mode** deve:
 - preparar argumentos, e identificar um ponto permitido para acesso a **kernel mode**
 - executar uma **instrução especial** que passa o controlo para o kernel
- Existe um número limitado deste tipo de pontos de entrada (economia de mecanismos 📖):
 - registos específicos para parâmetros, que são tipicamente apontadores para memória de processos em user mode
 - o processamento dessa informação é da total responsabilidade do kernel
- Um **número limitado de pontos de entrada** \Rightarrow **superfície de ataque bem definida**

Processos

- O **Kernel** define a noção de **processo**: uma instância de um programa que está a executar
- Os programas começam por estar guardados em armazenamento não volátil (e.g., código no disco)
- Para serem executados têm de ser carregados para memória e receber um identificador como processo
- Cada processo deve executar num contexto em que tem acesso a um conjunto de **recursos**, que devem estar disponíveis independentemente de outros processos
- A **fronteira entre processos** é uma fronteira de confiança: os processos têm de estar confinados/isolados entre si (separação de privilégio 📖📖)

Processos

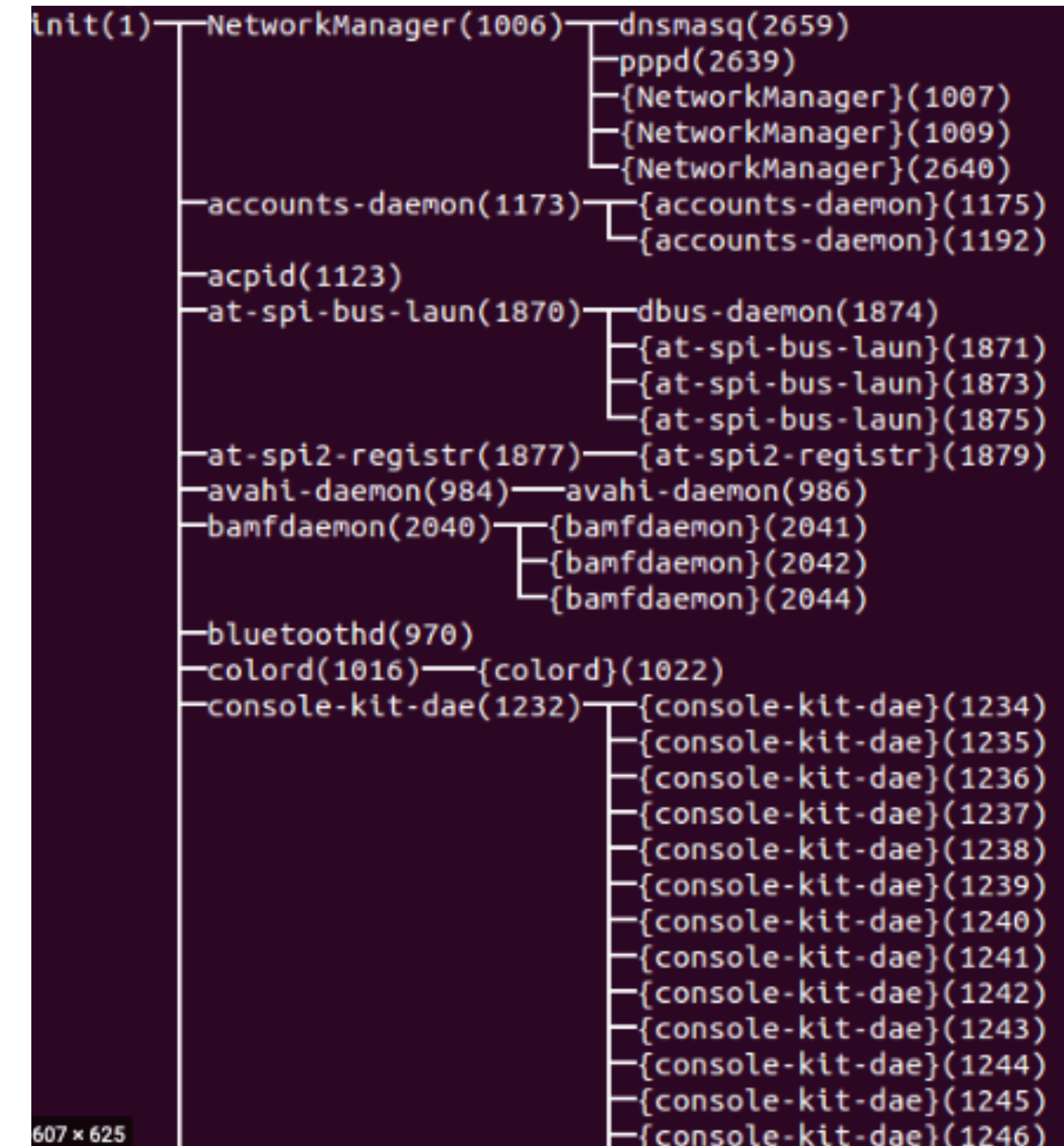
- O **Kernel** garante acesso a **recursos**:
 - atribui uma fração razoável de tempo de processador a cada **processo** (time slicing)
 - atribui um espaço de memória sobre o qual o processo pode trabalhar
 - concede acesso a outros recursos através de system calls
- Nos SO multi-utilizador, existe um conjunto de processos base que interagem com o utilizador humano:
 - quando o utilizador lança uma aplicação, o SO vê um dos processos que interagem com o utilizador (e.g., shell, GUI, etc.) a criar um novo processo => *forking*
 - Os SO gerem uma **hierarquia de processos**, em que tipicamente os descendentes herdam os privilégios dos seus criadores



Linux pstree

Processos

- Em Linux pode ver-se a árvore com `ps tree`
- O processo raiz é o `init` (PID 1)
- PID é um identificador único de processo
- As permissões atribuídas a cada processo dependem do utilizador que o cria
 - cada utilizador tem um UID (único para o utilizador) e um GID (único para o grupo)
 - tipicamente o 0 é reservado para o super-user (root)
 - o processo é associado aos mesmos UID, GID: sintaxe `processo (GID)`



```
tuts@fossilinux:~$ sudo cat /etc/passwd
[sudo] password for tuts:
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```

```
tuts@fossilinux:~$ id fossilinux_admin
uid=1001(fossilinux_admin) gid=1001(fossilinux_admin) groups=1001(fossilinux_admin)
tuts@fossilinux:~$ id tuts
uid=1000(tuts) gid=1000(tuts) groups=1000(tuts),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),126(sambashare),129(ubridge),130(libvirt)
tuts@fossilinux:~$
```


Processos

- Muitas vezes é necessário comunicar entre processos (IPC) \Rightarrow system calls
 - através do sistema de ficheiros
 - memória partilhada
 - mensagens síncronas: pipes, sockets
 - mensagens assíncronas: signals

```
-+= 00001 root /sbin/launchd
|--= 00093 root /usr/libexec/logd
|--= 00094 root /usr/libexec/UserEventAgent (System)
|--= 00096 root /System/Library/PrivateFrameworks/Uninstall.frameworko
|--= 00097 root /System/Library/Frameworks/CoreServices.framework/V
|--= 00098 root /System/Library/PrivateFrameworks/MediaRemote.frame
|+= 00101 root /usr/sbin/systemstats --daemon
| \--- 00414 root /usr/sbin/systemstats --logger-helper /private/va
|--= 00102 root /usr/libexec/configd
|--= 00104 root /System/Library/CoreServices/powerd.bundle/powerd
|--= 00105 root /usr/libexec/IOMFB_bics_daemon
|--= 00108 root /usr/libexec/remoted
|--= 00115 root /usr/libexec/watchdogd
|--= 00121 root /usr/libexec/kernelmanagerd
|--= 00122 root /usr/libexec/diskarbitrationd
|--= 00124 root /Library/PrivilegedHelperTools/com.wacom.UpdateHelp
|--= 00130 root /usr/libexec/thermalmonitord
|--= 00131 root /usr/libexec/opaidd
|--= 00132 root /System/Library/PrivateFrameworks/ApplePushService.
|--= 00133 root /Library/PrivilegedHelperTools/com.docker.vmnetsd
|--= 00134 root /System/Library/CoreServices/launchservicesd
|--= 00135 _timed /usr/libexec/timed
|--= 00136 _usbmuxd /System/Library/PrivateFrameworks/MobileDevice.
|--= 00137 root /usr/sbin/securityd -i
|--= 00140 _locationd /usr/libexec/locationd
|--= 00142 root autofs
|--= 00144 root /usr/libexec/das
|--= 00146 root /Library/Application Support/Checkpoint/Endpoint Co
|--= 00147 _distnote /usr/sbin/distnoted daemon
|--= 00151 root /System/Library/CoreServices/login
|--= 00152 root /System/Library/PrivateFrameworks/GenerationalStora
|--= 00153 root /usr/sbin/KernelEventAgent
```

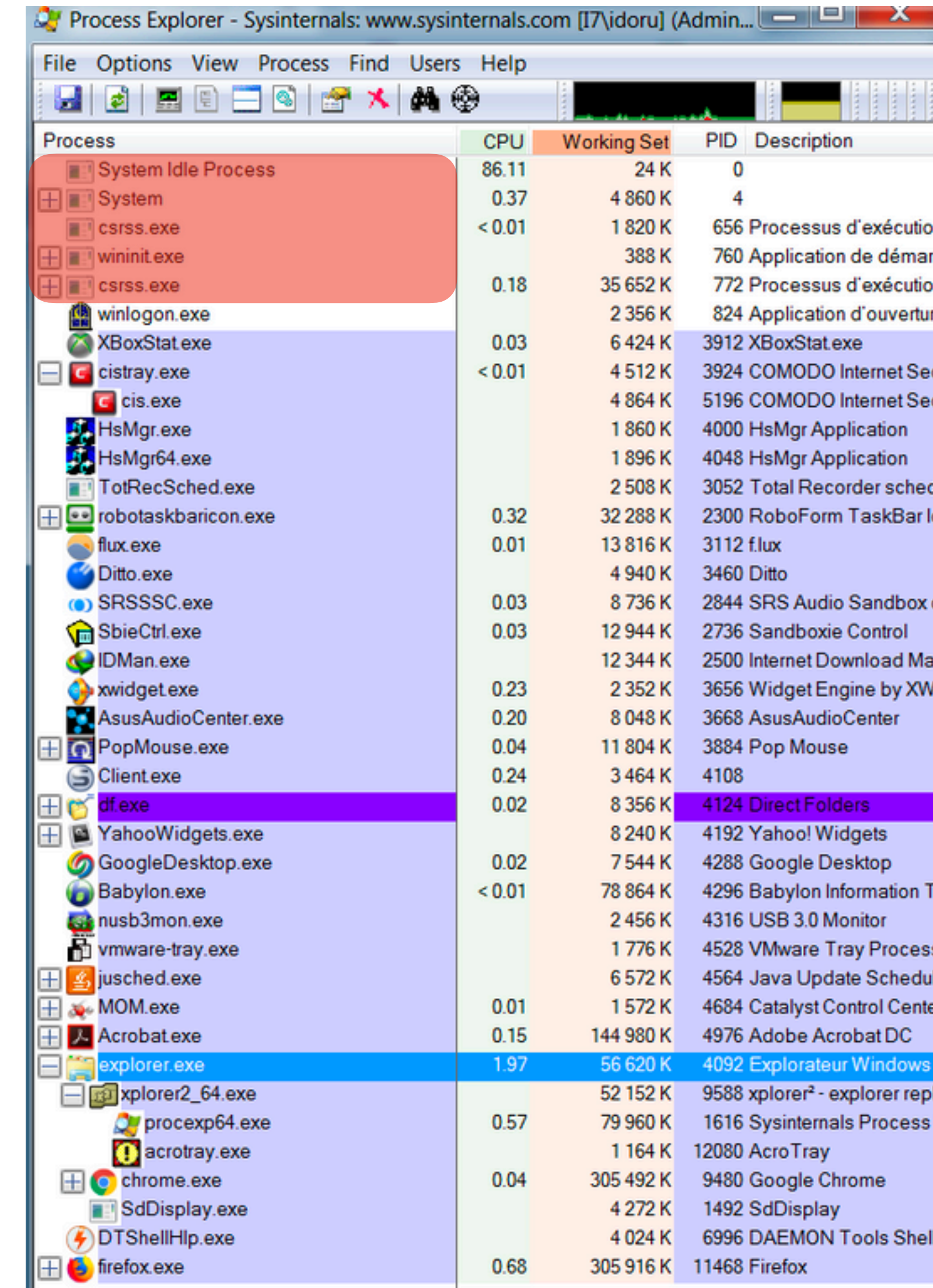
Mac pstree

Processos

- **Daemons, services:**

- processos que não são visíveis pelo utilizador (diretamente)
- E.g., indexação, login remoto, impressoras, sincronização de ficheiros, etc.
- geralmente arrancados antes dos próprios processos que interagem com o utilizador
- executam tipicamente com privilégios superiores aos dos utilizadores e sobrevivem às suas sessões

```
tuts@fossilinux:~$ sudo cat /etc/passwd
[sudo] password for tutorials:
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```



Process	CPU	Working Set	PID	Description
System Idle Process	86.11	24 K	0	
System	0.37	4 860 K	4	
csrss.exe	< 0.01	1 820 K	656	Processus d'exécution
wininit.exe		388 K	760	Application de démarr
csrss.exe	0.18	35 652 K	772	Processus d'exécution
winlogon.exe		2 356 K	824	Application d'ouverture
XBoxStat.exe	0.03	6 424 K	3912	XBoxStat.exe
cistray.exe	< 0.01	4 512 K	3924	COMODO Internet Sec
cis.exe		4 864 K	5196	COMODO Internet Sec
HsMgr.exe		1 860 K	4000	HsMgr Application
HsMgr64.exe		1 896 K	4048	HsMgr Application
TotRecSched.exe		2 508 K	3052	Total Recorder sched
robotaskbaricon.exe	0.32	32 288 K	2300	RoboForm TaskBar Ic
flux.exe	0.01	13 816 K	3112	f.lux
Ditto.exe		4 940 K	3460	Ditto
SRSSSC.exe	0.03	8 736 K	2844	SRS Audio Sandbox c
SbieCtrl.exe	0.03	12 944 K	2736	Sandboxie Control
IDMan.exe		12 344 K	2500	Internet Download Ma
xwidget.exe	0.23	2 352 K	3656	Widget Engine by XW
AsusAudioCenter.exe	0.20	8 048 K	3668	AsusAudioCenter
PopMouse.exe	0.04	11 804 K	3884	Pop Mouse
Client.exe	0.24	3 464 K	4108	
df.exe	0.02	8 356 K	4124	Direct Folders
YahooWidgets.exe		8 240 K	4192	Yahoo! Widgets
GoogleDesktop.exe	0.02	7 544 K	4288	Google Desktop
Babylon.exe	< 0.01	78 864 K	4296	Babylon Information T
nusb3mon.exe		2 456 K	4316	USB 3.0 Monitor
vmware-tray.exe		1 776 K	4528	VMware Tray Process
jusched.exe		6 572 K	4564	Java Update Schedul
MOM.exe	0.01	1 572 K	4684	Catalyst Control Cent
Acrobat.exe	0.15	144 980 K	4976	Adobe Acrobat DC
explorer.exe	1.97	56 620 K	4092	Explorateur Windows
xplorer2_64.exe		52 152 K	9588	xplorer² - explorer repl
procexp64.exe	0.57	79 960 K	1616	Sysinternals Process E
acrotray.exe		1 164 K	12080	AcroTray
chrome.exe	0.04	305 492 K	9480	Google Chrome
SdDisplay.exe		4 272 K	1492	SdDisplay
DTShellHlp.exe		4 024 K	6996	DAEMON Tools Shell
firefox.exe	0.68	305 916 K	11468	Firefox

Windows Process Explorer

Modelo de Confiança

- A confiança depositada nos processos lançados é **indutiva**:
 - o código armazenado no computador (nomeadamente a BIOS e o kernel) após uma instalação é **“confiável”** (proteção por omissão 🗝️)
 - o processo de *boot* utiliza este código para colocar o *kernel* em memória e passar-lhe o controlo, **preservando o estado “confiável”** em memória
 - o kernel lança processos com permissões que garantem que **nenhum novo processo pode alterar o estado de confiança**
 - os processos de hibernação **preservam o estado de confiança**
 - os administradores podem alterar o software instalado no sistema e o sistema de permissões, mas garantem que qualquer actualização **preserva o estado de confiança**

Blaster Worm (2003)

- Afetou Windows 2000 e Windows XP sem update SP2
- Grupo Chinês Xfocus (alegadamente) fez reverse engineering do patch do Windows XP SP2
- Quando infecção ocorria, um buffer overflow fazia o serviço de RPC “crashar”, o que forçava o shutdown do SO
- Tão disseminado que infetava quase instantaneamente qualquer SO vulnerável ligado à rede

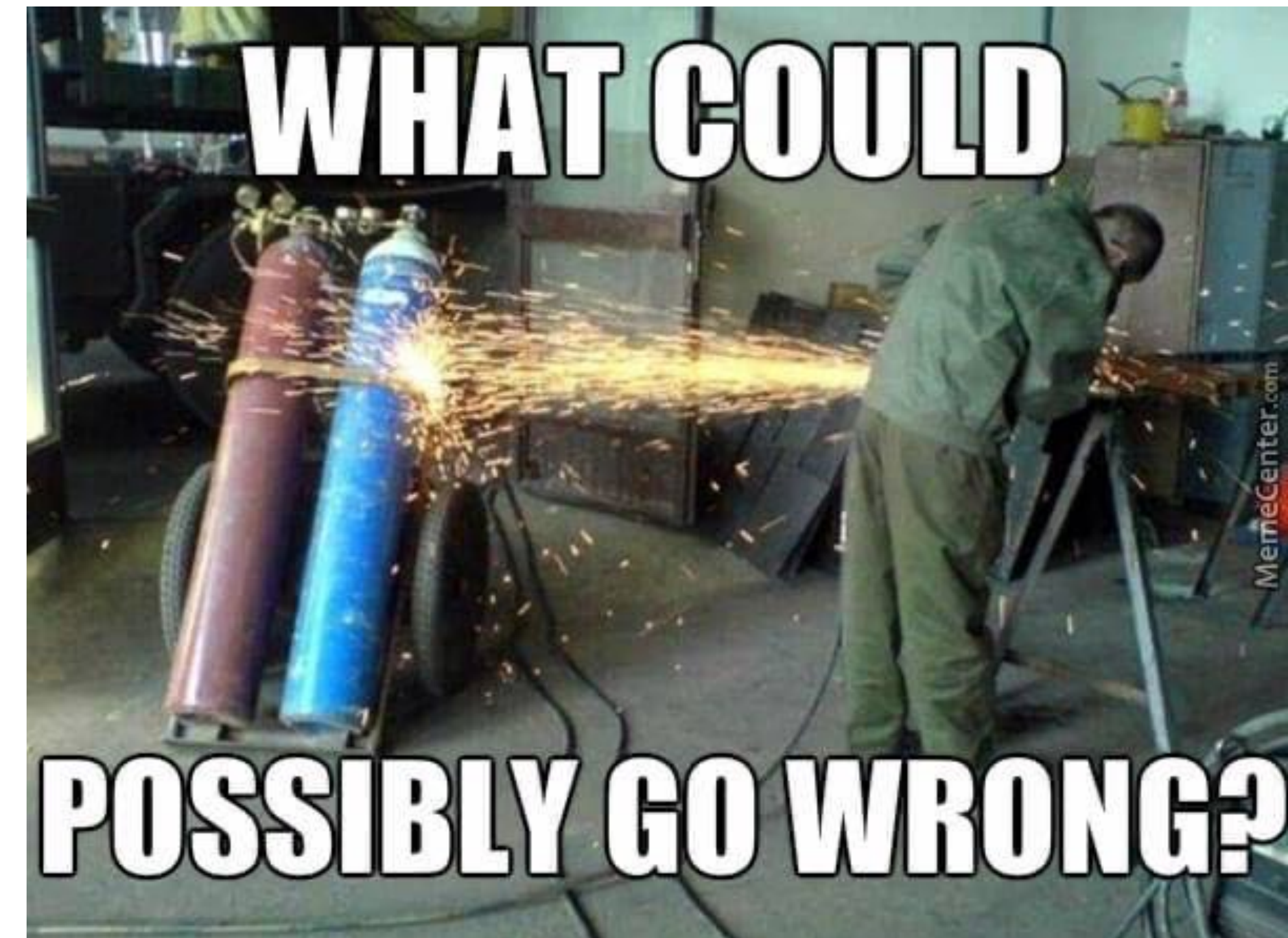


Modelo de Confiança

- O que significa "**confiável**":
 - que o sistema faz exatamente (e apenas) aquilo que foi especificado
 - exemplo: não transmite a nossa informação sensível para o exterior sem autorização
 - exemplo: garante que as nossas comunicações são estabelecidas com as entidades com quem queremos comunicar (e.g., servidores Google)
 - exemplo: cifra toda a informação em disco e limpa a memória quando fazemos shutdown

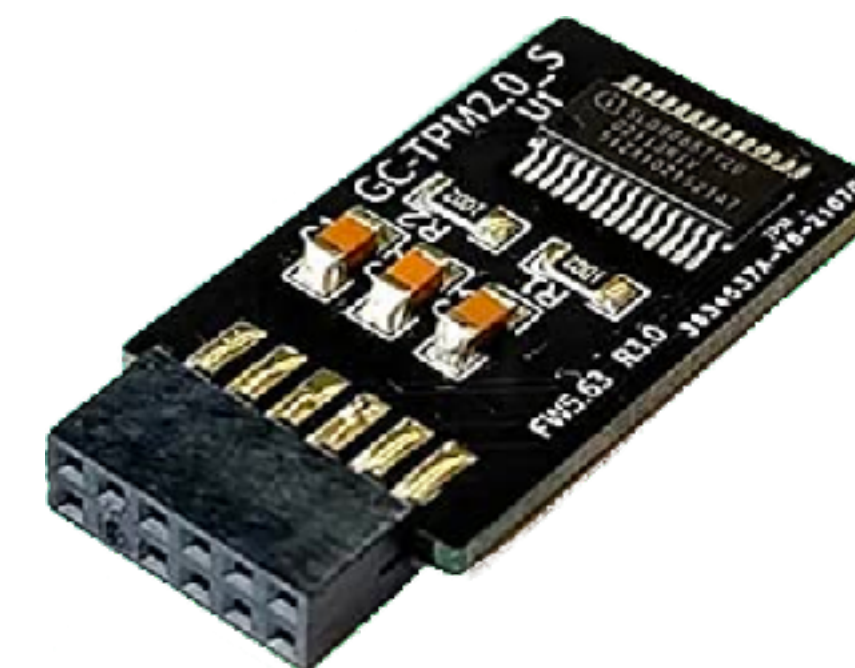
Modelo de Ameaças

- Ataques em todos os níveis do boot:
 - BIOS corrompida
 - ficheiros de hibernação corrompidos
 - *bootloader* corrompido
 - cold boot attacks (side-channel 😈)
- Vulnerabilidades que **afetam a implementação dos mecanismos de arranque** **fazem um bypass** completo ao modelo de confiança (o arranque é a âncora)



Quanto confiamos no SO?

- Trusted Platform Module (TPM)
- guardar credenciais, chaves de encriptação e outros dados sensíveis do sistema
- Um módulo criptográfico em hardware
- UEFI Secure Boot (**Root-of-Trust**)
- BIOS só faz boot a **componentes assinados digitalmente pelo fornecedor** da motherboard



Windows 11 and Secure Boot

Published August 2021

This article is intended for users who are not able to upgrade to Windows 11 because their PC is not currently Secure Boot capable. If you are unfamiliar with this level of technical detail, we recommend that you consult your PC manufacturer's support information for more instructions specific to your device.

Secure Boot is an important security feature designed to prevent malicious software from loading when your PC starts up (boots). Most modern PCs are capable of Secure Boot, but in some instances, there may be settings that cause the PC to appear to not be capable of Secure Boot. These settings can be changed in the PC firmware. Firmware, often called BIOS (Basic Input/Output System), is the software that starts up before Windows when you first turn on your PC.

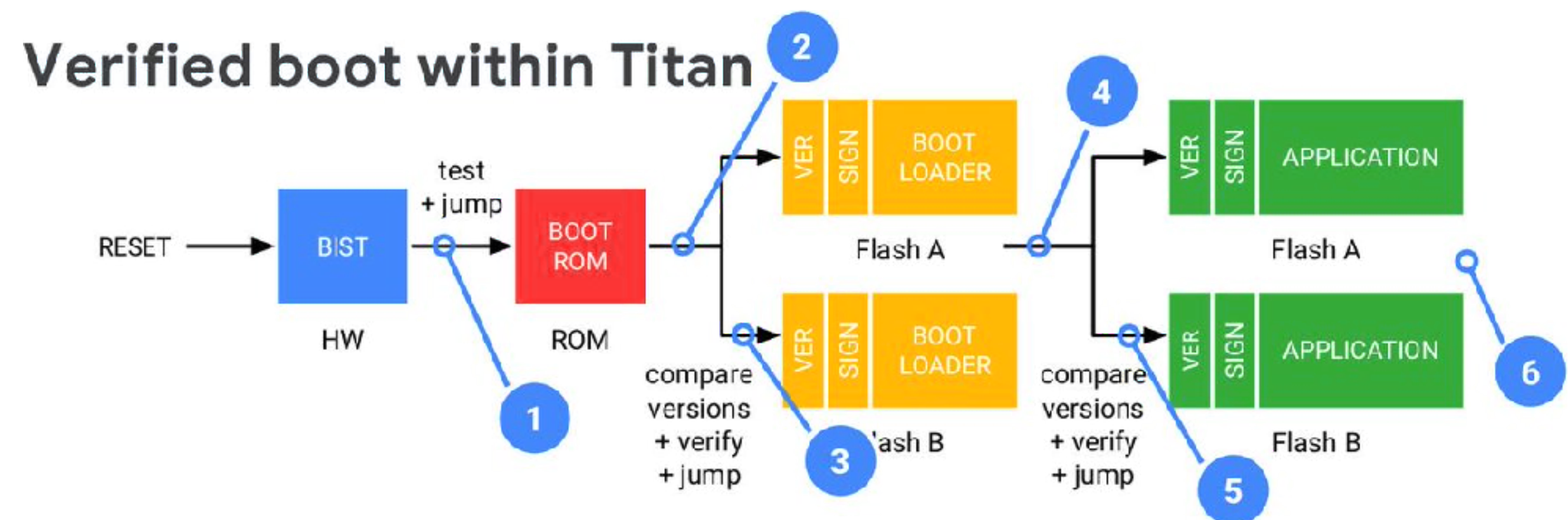
Quanto confiamos no Hardware?

- Google OpenTitan (2019 - ...)
- Projeto open-source para criar um chip mais confiável, com design aberto e transparente
- Qualquer um pode auditar o hardware para backdoors e vulnerabilidade de segurança

- Evitar supply-chain attacks 🤖

- **Root-of-Trust:**

- assegurar criptograficamente que o chip não foi modificado
 - entre BIOS e processador
- Fornece uma fundação de confiança para o SO correr no chip



1. Test logic (LBIST) and ROM (MBIST); if fail \Rightarrow stay in reset; else jump to ROM
2. Compare bootloader (BL) versions A + B; choose most recent
3. Verify BL signature; if fail, retry with other BL; if fail, freeze
4. Compare firmware application (FW) versions A + B; choose most recent
5. Verify FW signature; if fail, retry with other FW; if fail, freeze
6. Execute successfully verified FW

Medidas de Mitigação

- A maioria dos problemas de segurança surgem através de erros de administração
- Veremos exemplos de processos maliciosos (malware) e das formas que utilizam para corromper o modelo de confiança, bem como medidas de mitigação
- A **monitorização** é uma forma de mitigação comum para detectar quebras neste modelo (registo de compromissos 📖📖):
 - logs de eventos permitem detectar comportamentos suspeitos, como o crash repetido de um processo que está a tentar explorar uma vulnerabilidade
 - aplicações de monitorização de processos permitem visualizar os processos que estão a executar, os recursos que utilizam, e os ficheiros de código que os originam
- a **mediação** de instalação/execução de código com base em **assinaturas digitais** fornece também um entrave à introdução de código malicioso num sistema (mediação completa 📖📖 + proteção por omissão 📖📖)

Medidas de Mitigação

- Para já vamos estudar os mecanismos de segurança que são utilizados nos sistemas operativos para garantir **isolamento** entre processos
- impedem que um utilizador com poucos privilégios (e portanto fora do círculo de confiança) possa utilizar o sistema para além do que lhe é permitido
- impedem que um processo que contenha uma vulnerabilidade não abra uma porta que corrompa todo o sistema (defesa em profundidade)
- vamos focar-nos na **gestão da memória, processos e sistema de ficheiros**, que são aspectos fundamentais comuns a todos os SO

Memória

- A regra fundamental da gestão de memória diz que:
 - **um processo não pode aceder ao espaço de memória de outro processo**
 - a confidencialidade, integridade e **controlo de fluxo do kernel tem de ser protegida** de todos os processos que executam em modo utilizador
- Como se garante?
 - em run-time os acessos são mediados por um conjunto de mecanismos de hardware e software geridos pelo kernel (e.g., partição swap, memory protection keys)
 - as partes da memória virtual que estão em disco podem ser alvo de ataque *off-line* se um adversário puder aceder a essa informação ⇒ disco cifrado



A evolução do Windows

- DOS: nenhuma proteção de memória (e.g., qualquer processo podia escrever diretamente na memória do ecrã)
- Windows 1.x/3.x/95/98/ME: single-user SOs, algum isolamento entre user space e kernel space (e.g., aplicações correm como administrador)

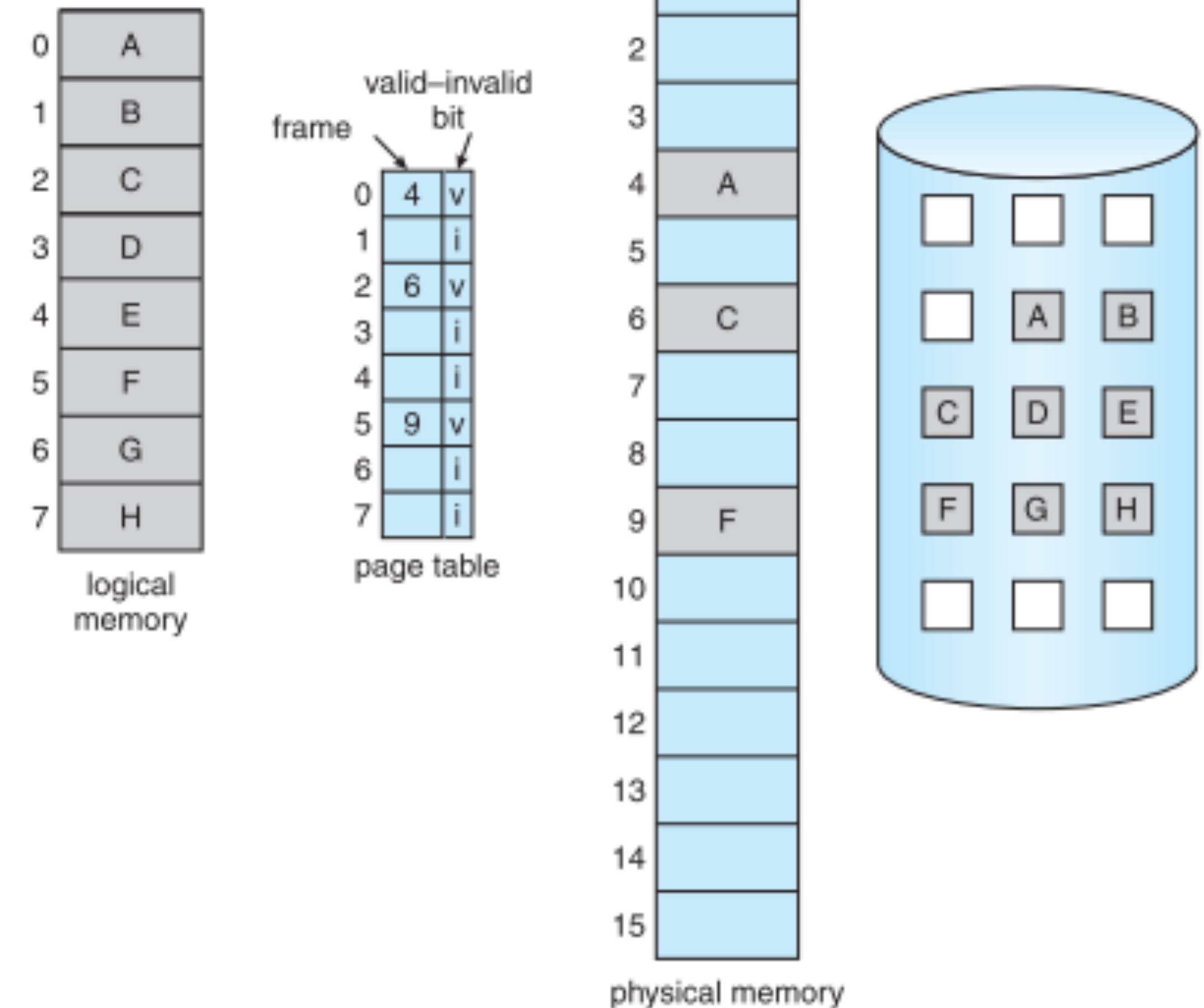
Windows 9x is a series of hybrid 16/32-bit operating systems.

Like most operating systems, Windows 9x consists of **kernel space** and **user space** memory. Although Windows 9x features some **memory protection**, it does not protect the first megabyte of memory from **userland** applications for compatibility reasons. This area of memory contains code critical to the functioning of the operating system, and by writing into this area of memory an application can **crash** or **freeze** the operating system. This was a source of instability as faulty applications could accidentally write into this region, potentially corrupting important operating system memory, which usually resulted in some form of system error and halt.^[24]

- Windows NT/2000/XP/...: multi-user SOs, isolamento entre utilizadores = sistemas *n?x
 - processos no mesmo user space não são isolados (containers nas próxima aulas)

Memória

- O espaço de memória gerido por um SO é muito maior do que o espaço físico:
- o processador dá suporte a mecanismos de memória virtual
- o espaço de endereçamento de um processo está dividido em páginas
- algumas estão em memória outras em armazenamento não volátil
- quando é necessário trocar, diz-se que ocorreu um "page fault"



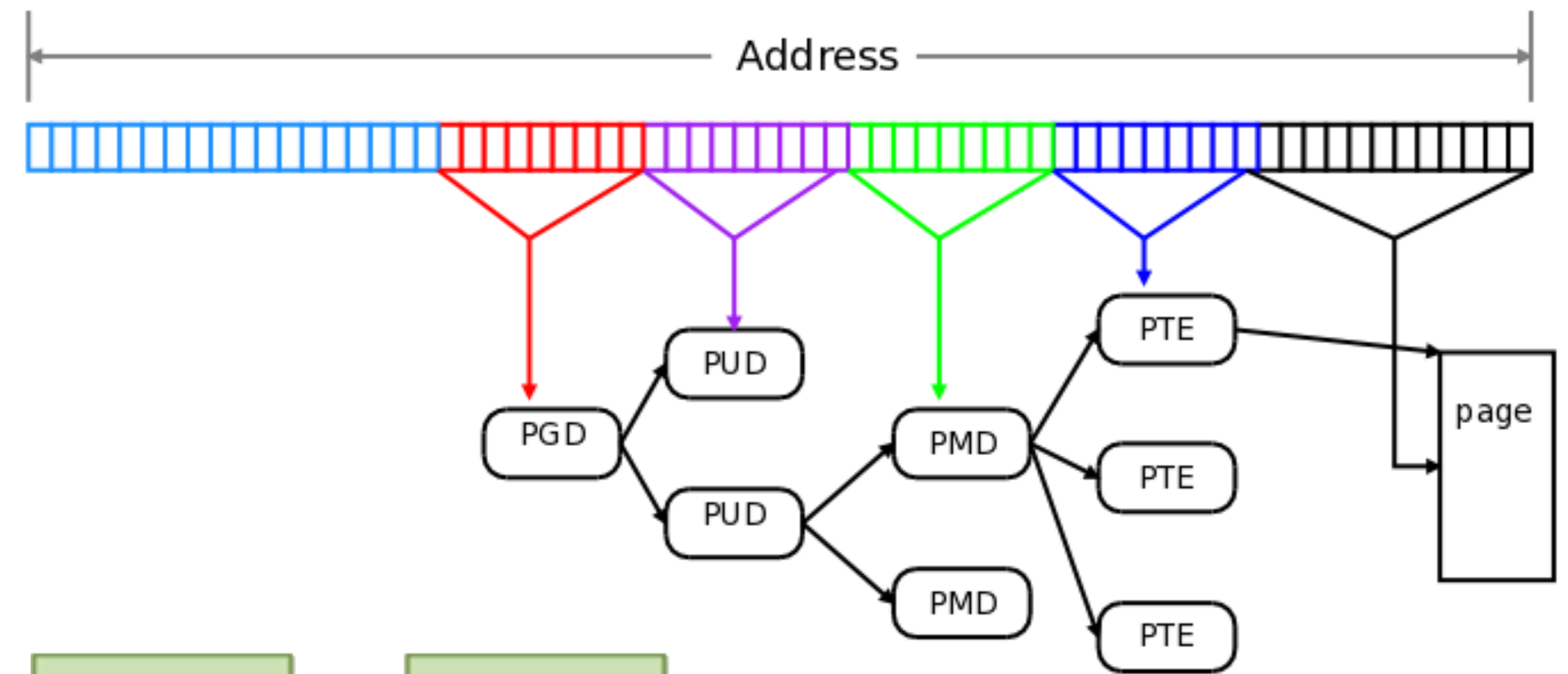
Tradução de Endereços

- A tradução de endereços necessária à implementação de mecanismos de memória virtual cumpre dois propósitos:
- **isolamento**: cada processo acede a uma zona de memória que não existe na realidade, e que dá visão/acesso limitados aos recursos (mediação completa 📖)
- **eficiência**: esconde mecanismos de optimização (caching, speculative access, paging, etc.)
- Nos últimos anos ficou claro que algumas optimizações criam, de facto, novos pontos de ataque (side-channel 😈 como Spectre e Meltdown)

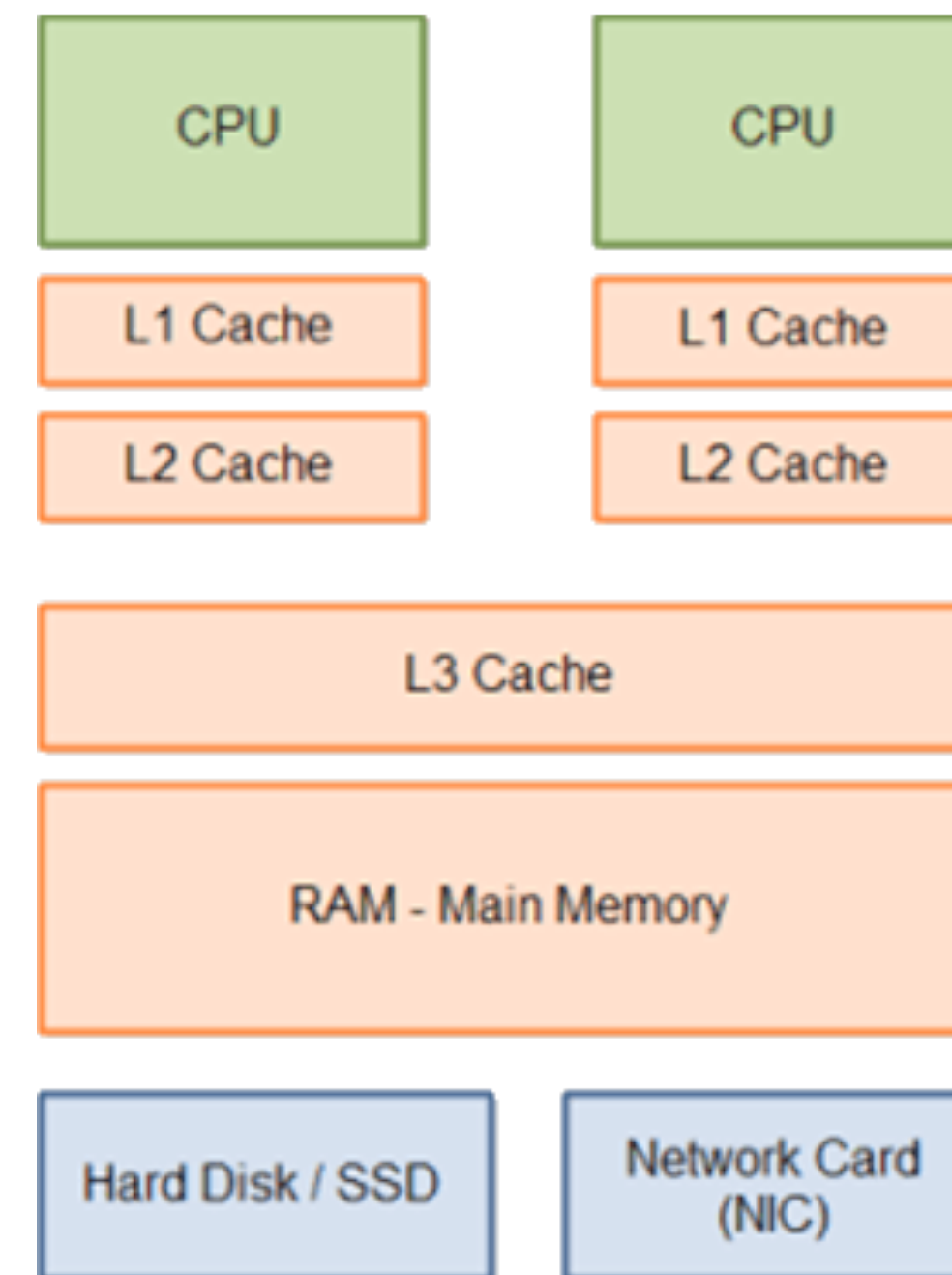


Tradução de Endereços

- A **memória virtual** está dividida em páginas, e.g., 4KB
- O sistema tem de armazenar, para cada página (se utilizada) a sua **localização física**
- Page-table: árvore esparsa com informação nas folhas
- Processador oferece suporte para gerir estas estruturas



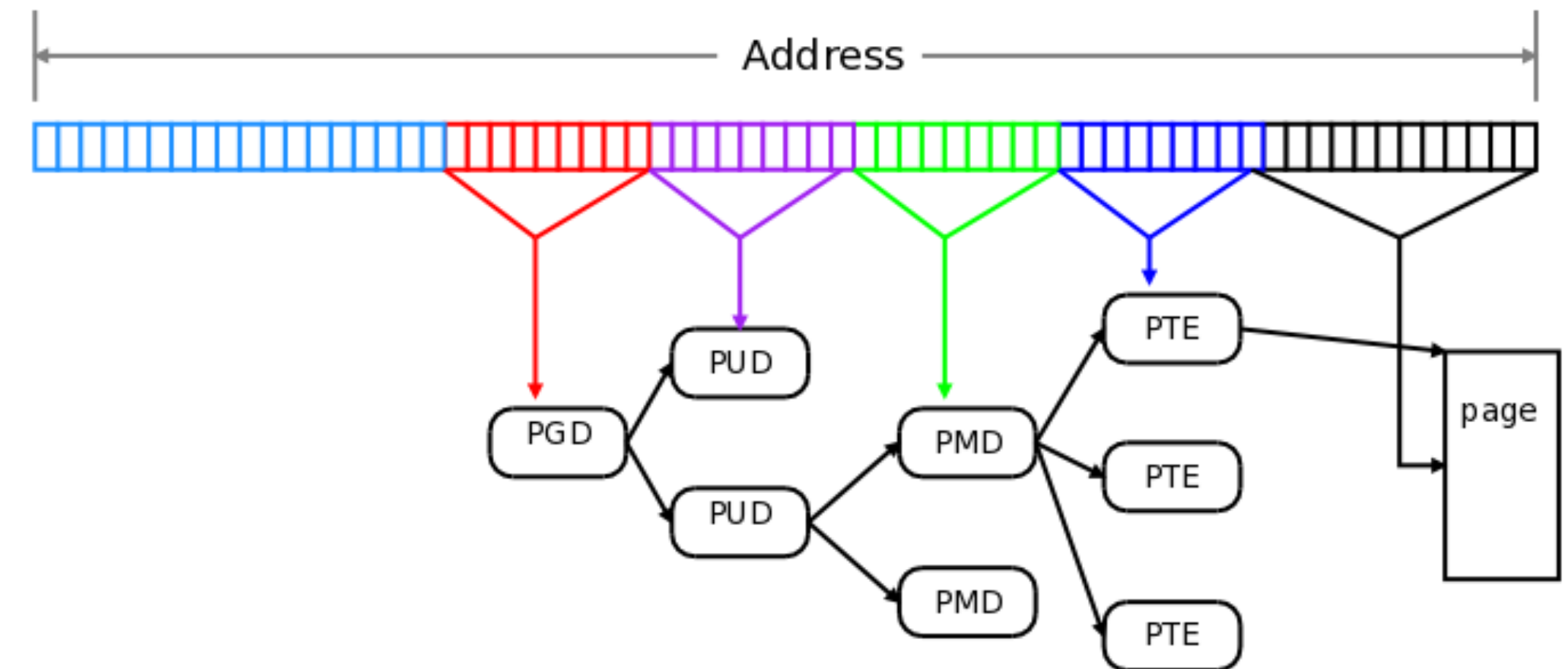
<https://lwn.net/Articles/717293/>



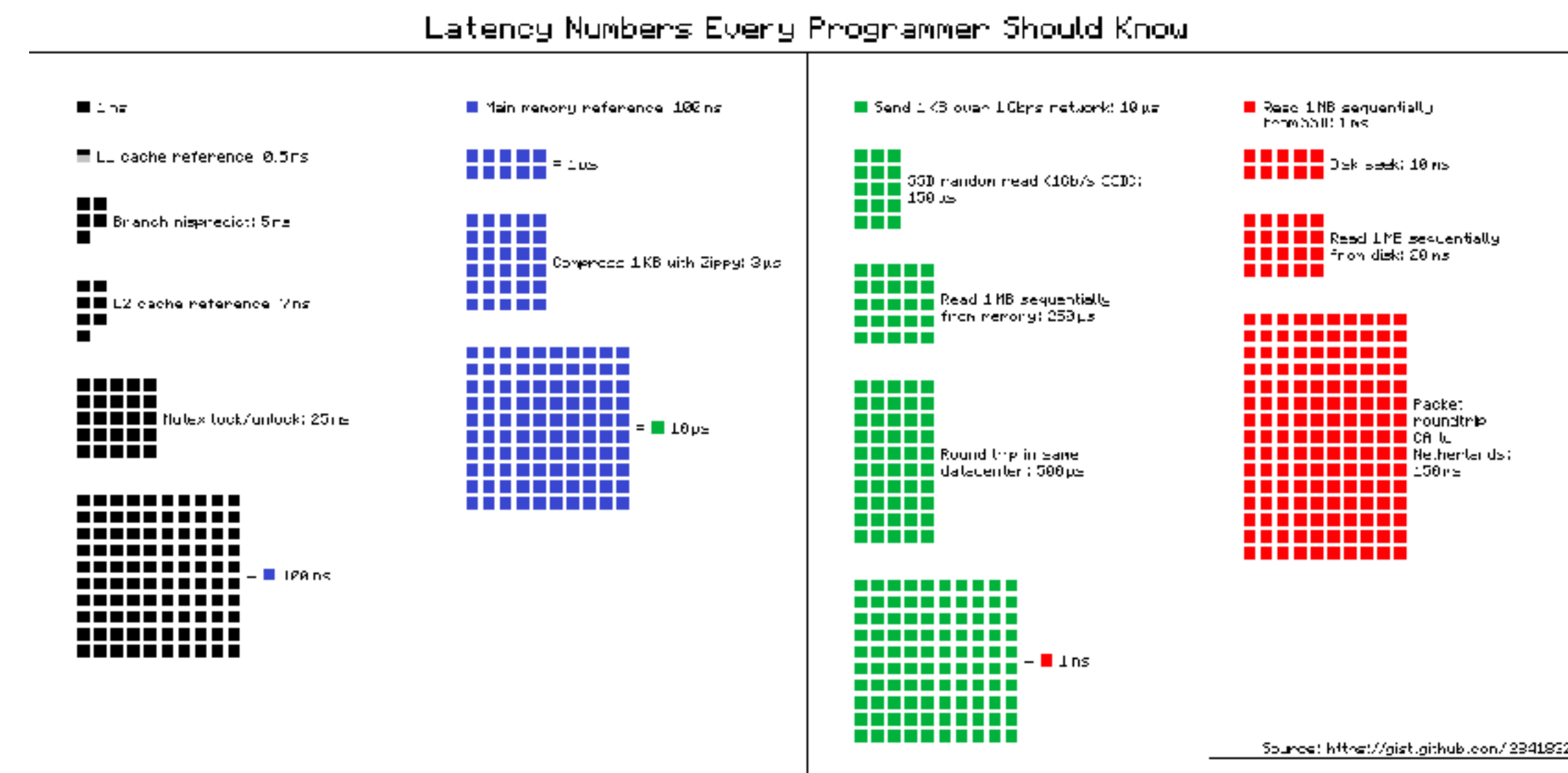
<https://cpuninja.com/cpu-cache/>

Tradução de Endereços

- Aceder a uma page table (que está em memória física) é **penalizador**
- Translation Lookaside Buffer (TLB):
 - cache de páginas traduzidas recentemente
- informação para controlo de acessos em cada página (ACL)
- Read/Write/eXecute (NX bit)



<https://lwn.net/Articles/717293/>

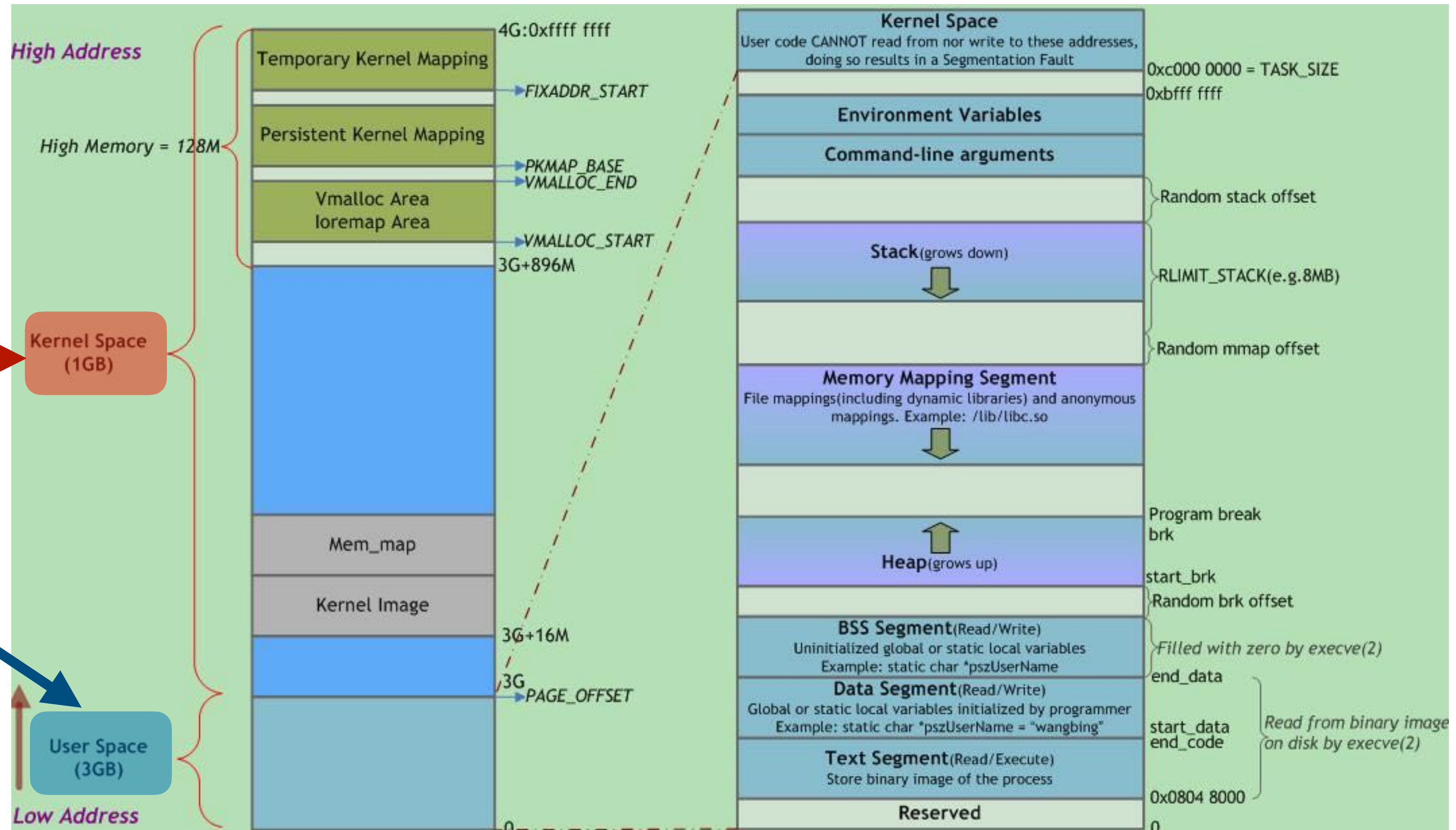


Tradução de Endereços

- Como lidar com system calls?
 - uma gestão totalmente independente dos espaços de endereçamento tornaria as mudanças de contexto muito ineficientes
- Kernel mapping:
 - parte da memória virtual do kernel está mapeada diretamente na memória virtual de cada processo, mas com permissões diferentes
 - **User mode (UR,UW,UX)**
 - **Kernel/privileged mode (PR,PW,PX)**

Memória Virtual


- Parte do espaço de memória de um processo é ocupado/**gerido pelo Kernel**, por questões de eficiência.
- O **processo** não tem acesso a esse espaço, mas **pode interagir com ele via system calls**, e.g., memory map.



Kernel Mapping

- Quando um processo faz uma system call não é necessário alterar o sistema de mapeamento de páginas:
 - a memória relevante para o kernel já está mapeada
 - mais importante: a parte da memória do processo relevante à system call coexiste no mesmo espaço de endereçamento
- Quando se muda de processo de utilizador, as tabelas de páginas são alteradas, mas as que dizem respeito à memória do kernel são as mesmas (relembrar Pegasus)

Defesa em profundidade

- Que permissões deve ter o kernel sobre a memória dos outros processos:
 - todas as permissões \Rightarrow !
- Defesa em profundidade:
 - nem o kernel deve ser capaz de violar a regra W^X
 - impedir o kernel de escrever em partes da memória do utilizador é uma forma de impedir fugas de informação/código malicioso no caso de kernel corrompido

Acknowledgements

- This lecture's slides have been inspired by the following lectures:
 - CSE127: System Security I + System Security II
 - CS155: Isolation and Sandboxing