

Operációs rendszerek: felépítés és alpműködés

Mészáros Tamás

<http://www.mit.bme.hu/~meszaros/>

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Az előadásfóliák legfrissebb változata a tantárgy honlapján érhető el.

Az előadásanyagok BME-n kívüli felhasználása és más rendszerekben történő közzététele előzetes engedélyhez kötött.

Hogyan építsünk fel egy operációs rendszert?

Az operációs rendszer

azon **programok** összessége,
amelyek vezérlik a számítógép hardverének működését,
és lehetővé teszik azon felhasználói feladatok végrehajtását.

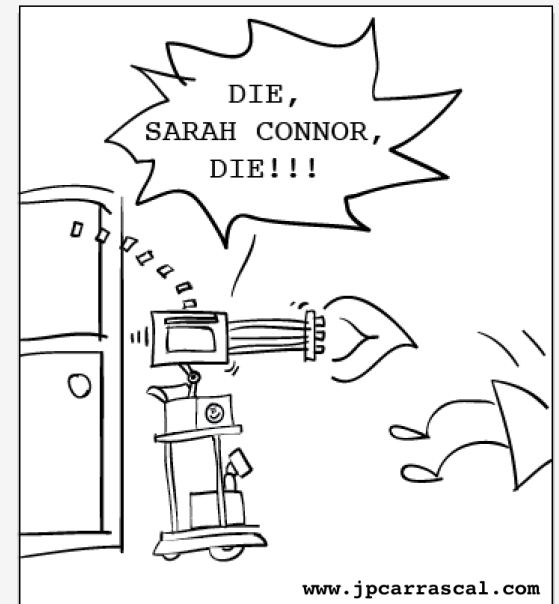
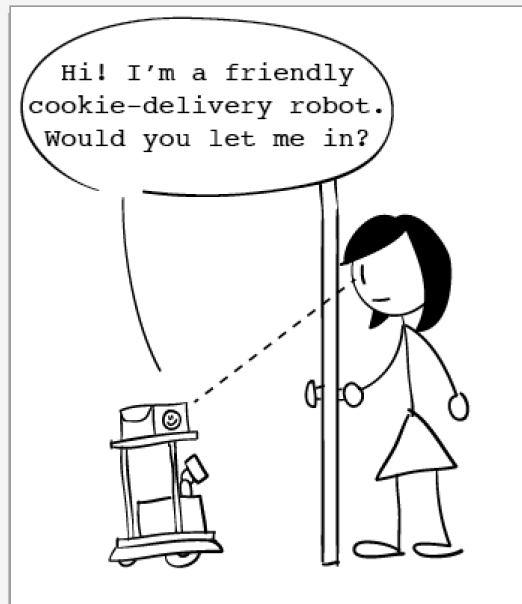
Elvárások

- egyszerre több feladat kiszolgálása (több program futtatása)
- megbízható
- biztonságos

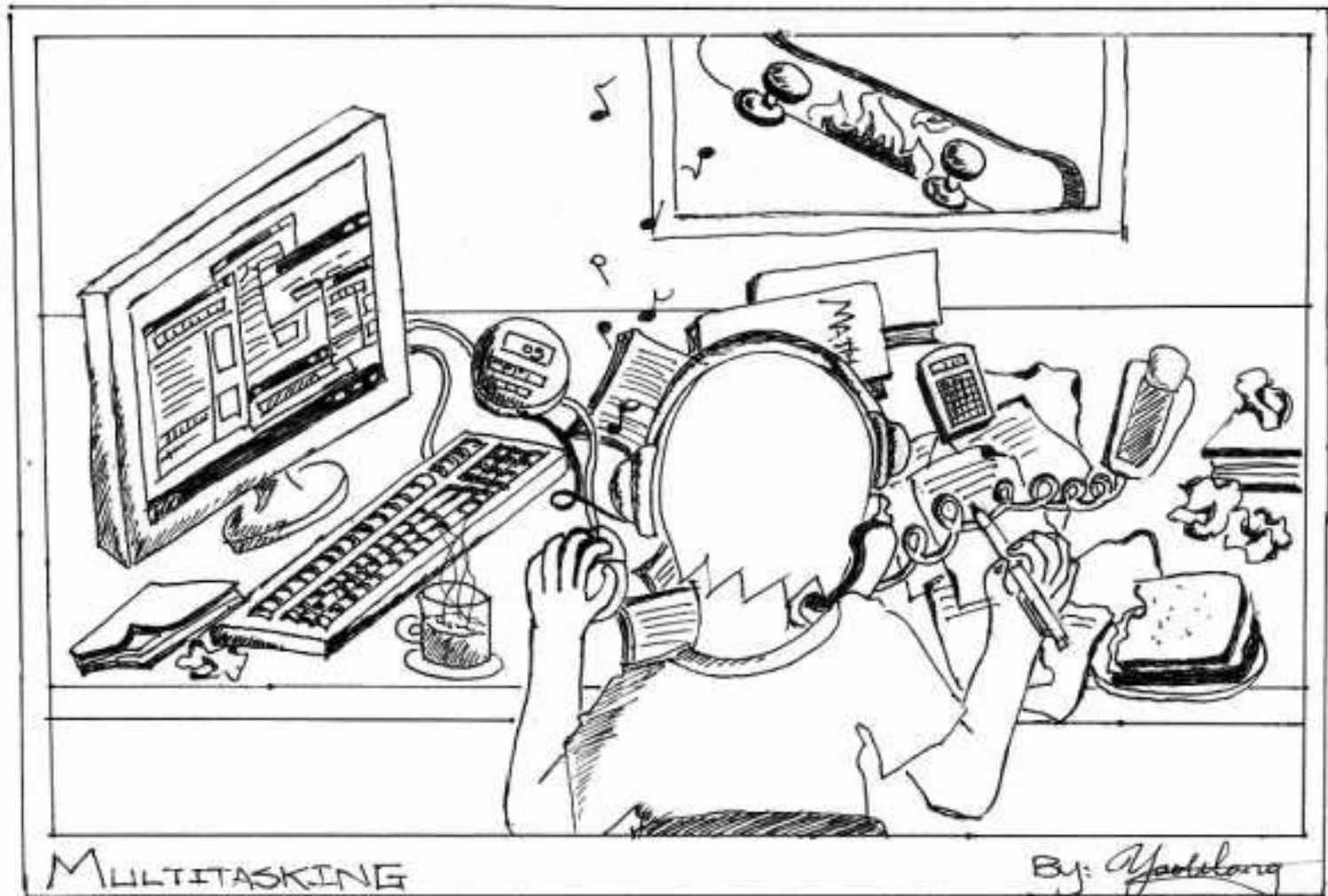
Programok

- megoldják a feladatainkat
- különféle forrásokból származnak (OS, alkalmazásból, sw repo, web stb.)

Megbízhatunk-e a szoftverekben?



Architekturális megfontolások: multiprogramozás



Architektúrális megfontolások: fennhatóság

AN x64 PROCESSOR IS SCREAMING ALONG AT BILLIONS OF CYCLES PER SECOND TO RUN THE XNU KERNEL, WHICH IS FRANTICALLY WORKING THROUGH ALL THE POSIX-SPECIFIED ABSTRACTION TO CREATE THE DARWIN SYSTEM UNDERLYING OS X, WHICH IN TURN IS STRAINING ITSELF TO RUN FIREFOX AND ITS GECKO RENDERER, WHICH CREATES A FLASH OBJECT WHICH RENDERS DOZENS OF VIDEO FRAMES EVERY SECOND

BECAUSE I WANTED TO SEE A CAT JUMP INTO A BOX AND FALL OVER.



I AM A GOD.

forrás: xkcd



forrás: youtube

A fennhatóság megvalósítása

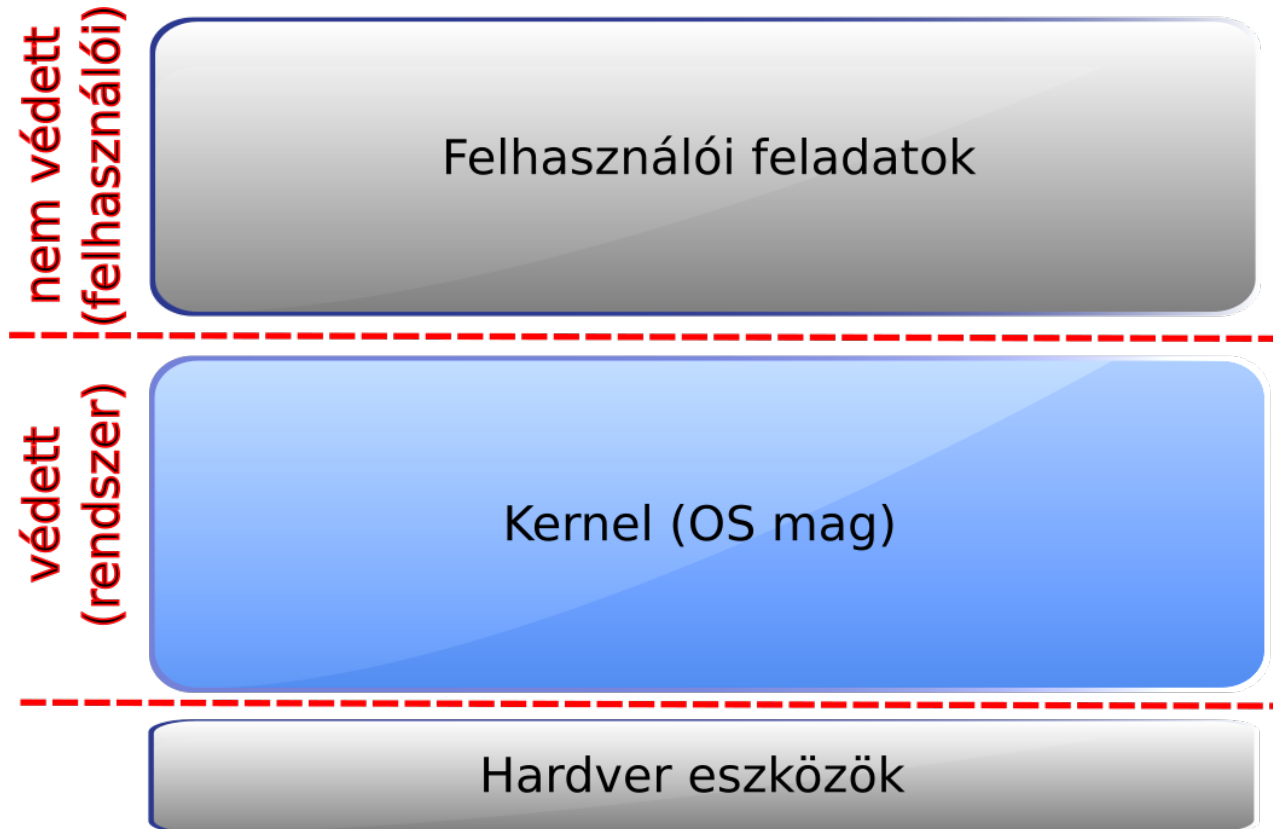
- Hogyan felügyelheti egy program egy másik működését?
- Ismétlés: CPU védelmi szintek (**SZGA**, **hardver alapok**)
 - legalább két eltérő működési mód
 - 0. privilegizált avagy védett mód (bármilyen megtehető)
 - más üzemmódban a CPU korlátozza
 - utasítások végrehajtását, memóriaterületek elérését, perifériák hozzáférését
- Az OS programjának egy része **védtett módban** fut
 - ez a rész fennhatóságot gyakorol minden más program felett
 - szabályozza az életciklusukat (keletkezés, működés, megszűnés)

*A **kernel** az operációs rendszer védett módban működő programja, amely felügyeli a felhasználói módú programok működését, és biztosítja hozzáférésüket a rendszer erőforrásaihoz.*

- Minden más program **felhasználói módban** működik
 - hardveresen betartatott korlátozásokkal

A védett módban működő kernel

Vezérlőprogram



Erőforrás-allokátor

A kernel

- Vezérlőprogramként felügyeli más programok végrehajtását
 - életciklus-menedzsment (létrehozás, működés, megszűnés)
 - működési események kezelése, kézbesítése
 - szolgáltatásokat nyújt számukra
- Menedzseli az erőforrásokat
 - eszközök előkészítése a felhasználásra
 - kezelésükkel kapcsolatos közös funkciók biztosítása
 - működésükkel kapcsolatos események kezelése, illetve továbbítása
 - párhuzamos kérések kiszolgálása, szeparációja, konfliktusok feloldása
- A megbízhatóság és biztonság szem előtt tartása
 - az erőforrások védelme a hibás vagy kártékony felhasználástól
 - a futó programok szeparációja, külső védelme
 - biztonsággal kapcsolatos közös funkciók biztosítása a programok számára

Mire lehet még szükségünk?

- Erőforrás-menedzsment ✓
- Felügyelet ✓
- ... ???

(Gondoljunk a felhasználói szerepkörökre!)

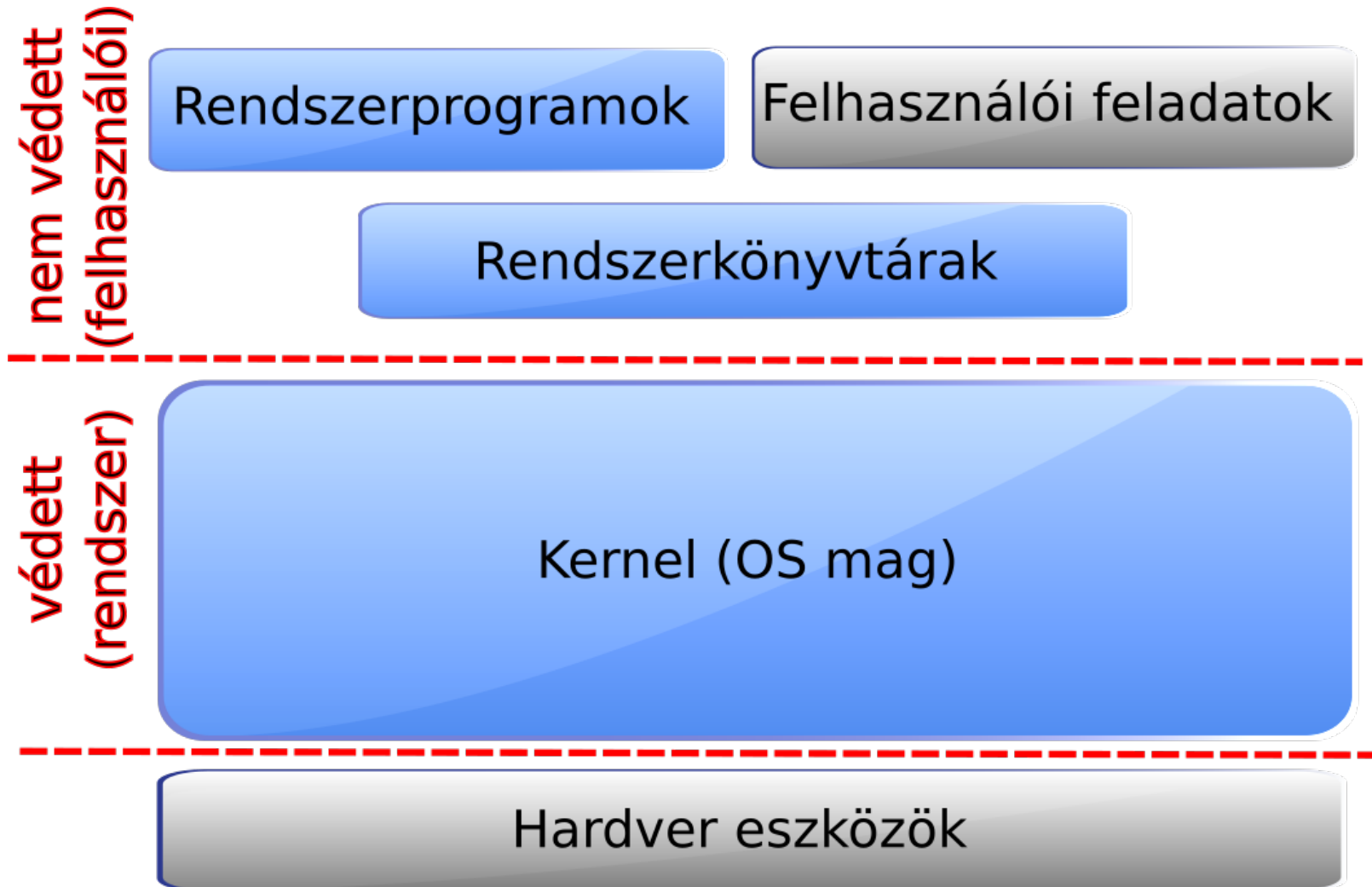
Az OS további részei

Rendszerkönyvtárnak nevezzük az operációs rendszer részét képező programkönyvtárakat, amelyeket a programok felhasználhatnak működésük során.

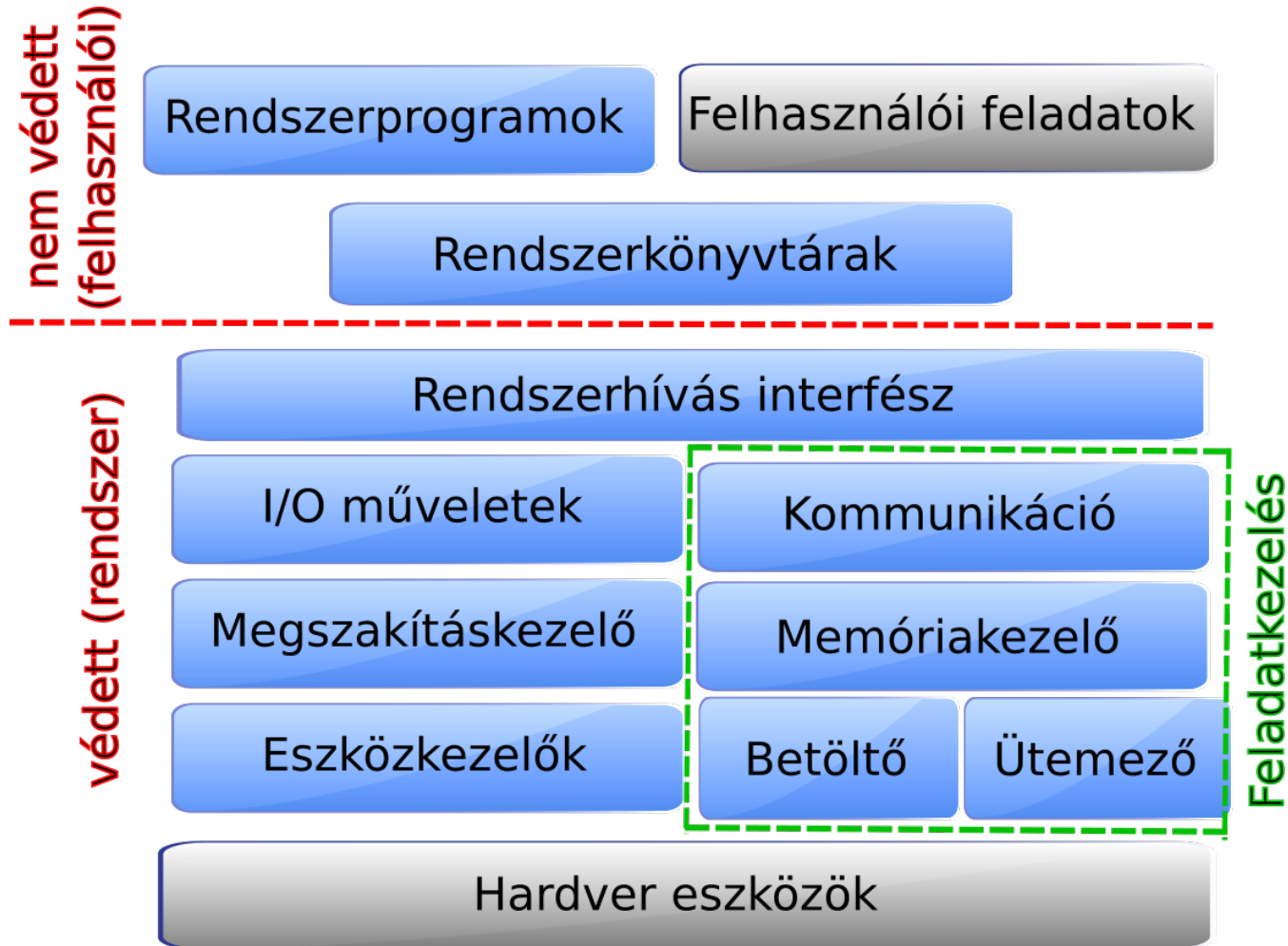
A ***rendszerprogram*** az operációs rendszer részét képező, működésével kapcsolatos feladatokat megoldó program.

A ***rendszer szolgáltatás*** az operációs rendszer által kezelt, folyamatosan elérhető funkciókat nyújtó program.

Az OS további részei



A kernel vázlatos felépítése



Az OS, mint feladat-végrehajtó rendszer

Áttekintés...

- Az operációs rendszer
 - feladatok végrehajtása
 - **vezérlőprogram**
 - **erőforrás-allokátor**
- Elvárások
 - feladatok egyidejű kiszolgálása
 - megbízható működés
 - esetenként valós idejűség



Az OS felépítése

- Kialakulása
 - köteget rendszerek
 - **multiprogramozott**
 - **időosztásos**
 - beágyazott

Hogyan kezeljük a feladatokat?



Milyen feladatokat futtatunk?

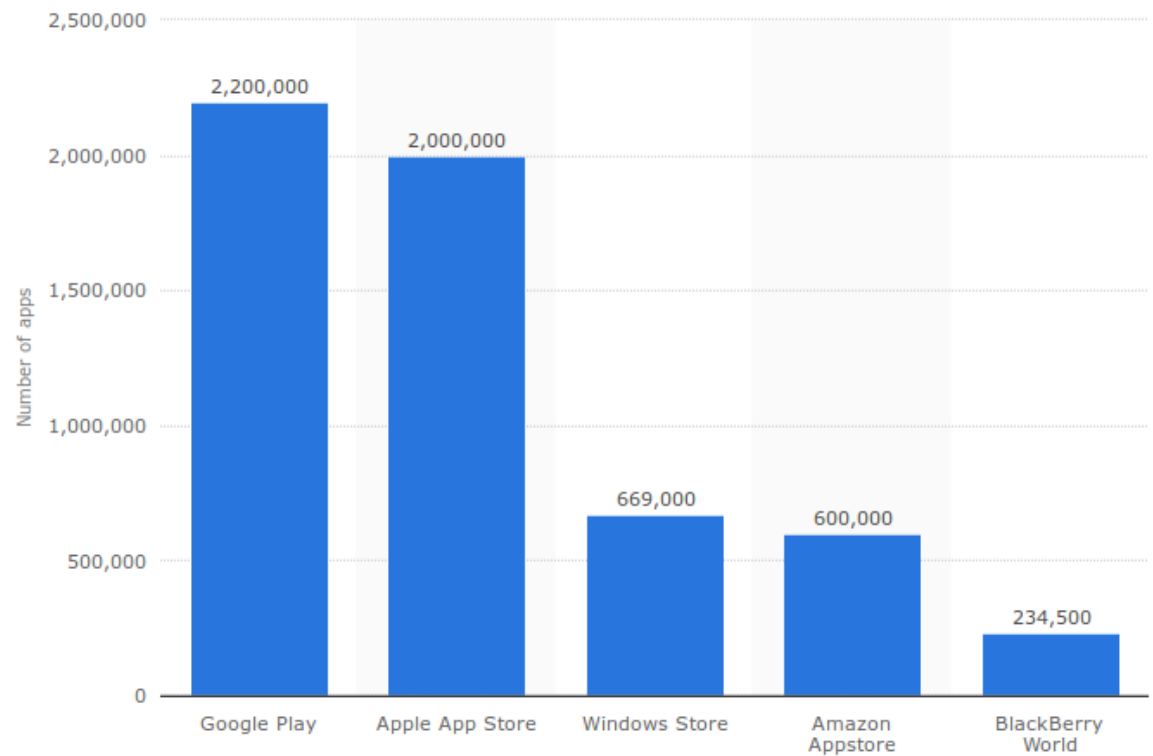
- A felhasználói feladatok sokszínűsége
- Az operációs rendszerek osztályozása (kliens, szerver, beágyazott...)
- Alkalmazások

Synaptic Package Manager

54628 packages listed, 2019 installed,

```
> yum list all | wc -l
22747
```

Number of apps available in leading app stores as of June 2016



forrás: statistica.com

A feladatok jellege

- I/O-intenzív feladatok
 - idejük nagy részét várakozással töltik (adatbetöltés, adatkírás)
 - kevés processzoridőre van szükségük
 - pl.: fájlserver, webszerver, email kliens és szerver stb.
- CPU-intenzív feladatok
 - idejük nagy részét a processzoron szeretnék tölteni
 - ehhez képest (relatív) kevés I/O műveletre van szükségük
 - pl.: titkosítási és matematikai műveletek, összetett adatfeldolgozás stb.
- Memória-intenzív feladatok
 - egy időben nagy mennyiségű adat elérésére van szükségük
 - ha van elég, akkor CPU-intenzívek, ha nincs, akkor I/O-intenzív feladattá válnak
 - pl. nagy mátrixok szorzása, keresési indexek építése és használata stb.
- Speciális igények
 - valós idejű működés
 - filmnézés
 - ...

Elvárásaink

- Kevés várakozás

várakozási idő (waiting time)

körülfordulási idő (turnaround time)

válaszidő (response time)

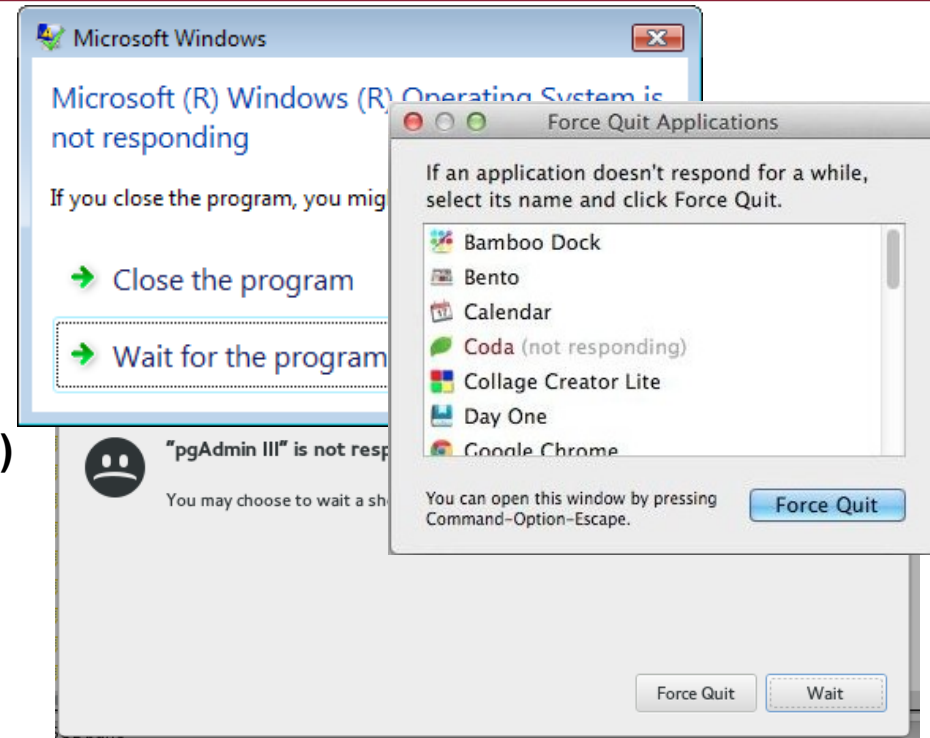
- Hatékonyság

CPU-kihasználtság (CPU utilization)

átbocsájtó képesség (throughput)

rezsiköltség (overhead)

- Jósolhatóság, determinisztikusság



Az optimális feladat-végrehajtó rendszer

- A naiv felhasználó elvárásai
 - biztosítja feladatai végrehajtását
 - minimalizálja a várakozási és válaszidőt
 - az erőforrásokat (CPU, I/O) maximális kihasználja
 - minél kisebb rezsiköltséggel dolgozik

- Mit tapasztal a rendszer használata során?
 - egyes programok „lassan” futnak
 - mások ok nélkül „lefagynak”
 - „feleslegesen” erőforrásokat foglalnak
 - akadozik a filmnézés
 - gyorsan merül az akkumulátor
 - néha mintha az egész rendszer leállna
 - nem tudja fogadni a hívást
 - ...



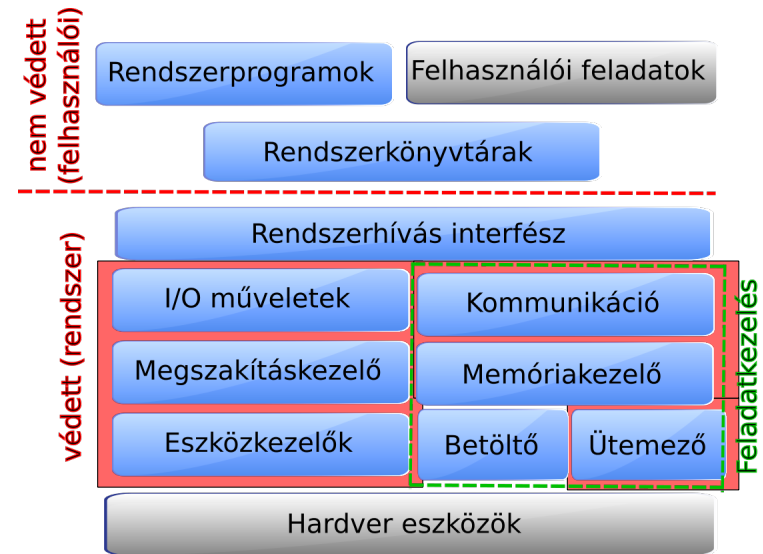
Mi okozza a nehézségeket?

- Az OS nem lát a jövőbe
 - milyen feladatok jönnek
 - milyen jellegűek
- Sok a feladat
 - különböző elvárások
 - más az optimalitási kritérium
 - néha túl sok, „vergődik” a rendszer
- A feladatok hatással vannak egymásra
 - együttműködnek
 - versenyeznek
- Hibák
 - programozói
 - hardver

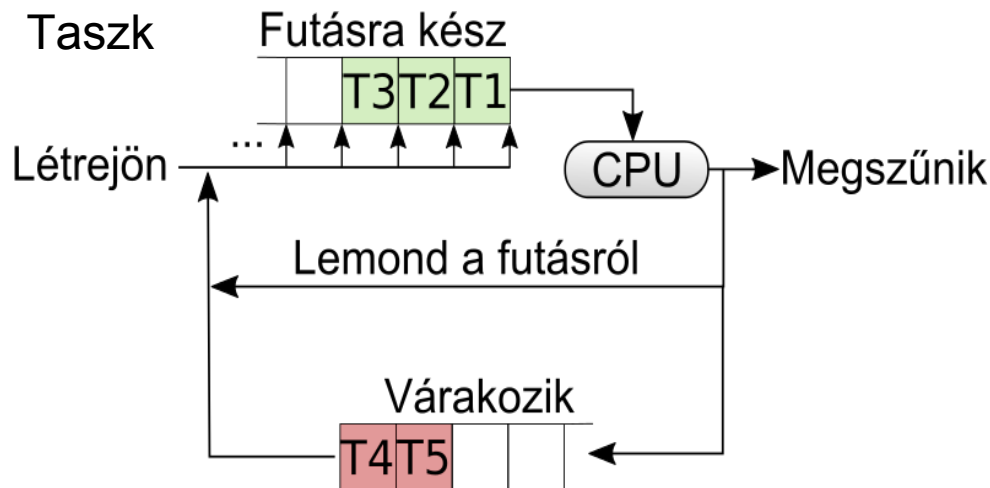
Az OS, mint erőforrás-allokátor

Áttekintés...

- Az operációs rendszer
 - felhasználói feladatok támogatása
 - vezérlőprogram
 - **erőforrás-allokátor**
- Elvárások
 - feladatok egyidejű kiszolgálása
 - megbízható működés
 - erőforrások optimális kihasználása



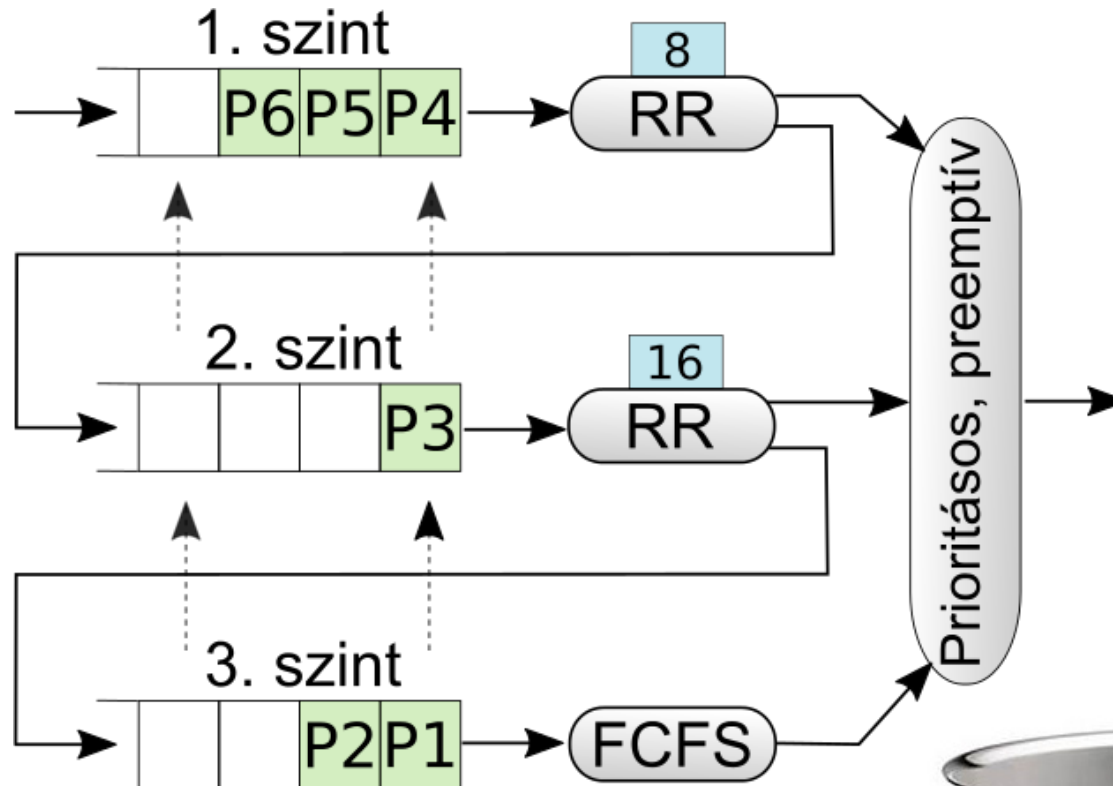
Az OS felépítése



- Erőforrások
 - processzor (CPU)
 - központi memória
 - tárolórendszerek
 - perifériák
 - egyéb hardvereszközök

Többszintű visszacsatolt sorok ütemező

multilevel feedback queue (MFQ)



Taszkok szintlépései:

- amelyik kihasználja az időszelét, az lentebb lép
- várakozó állapotba kerülő taszkok fentebb lépnek



Turing díj járt érte

Esettanulmány: AMD Ryzen

- Egyetlen tokban 8 CPU mag
 - 2 „Core Complex” (CCX)
 - Infinity Fabric összekötőelem
 - L3 tekintetében ~NUMA

- CCX
 - 4 CPU mag (SMP)
 - 8MB L3 cache

- CPU mag
 - 2 szál (SMT)
 - 512K L2 cache, 64K+32K L1 cache

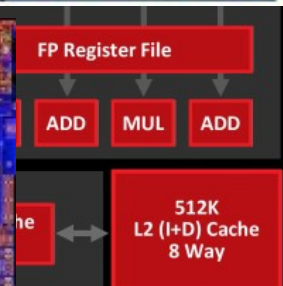
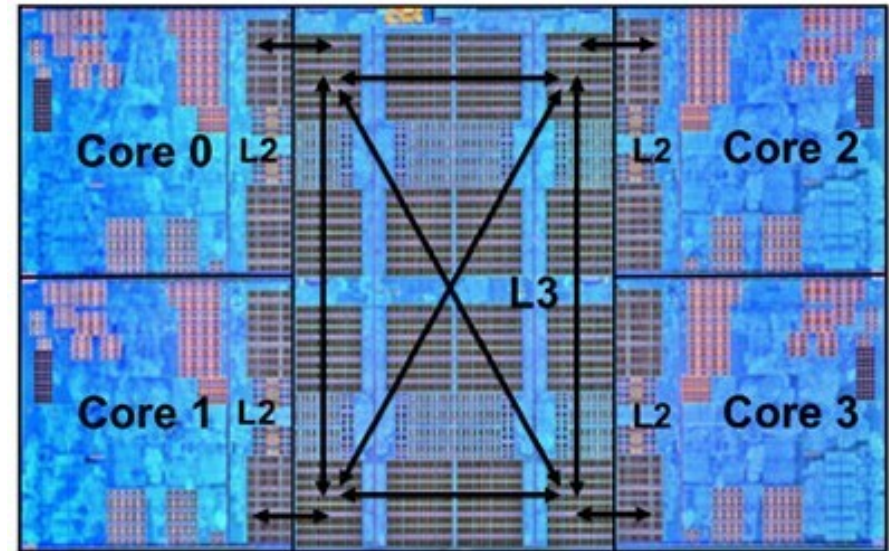
- Szempontok** az ütemező / programozók számára

- a CCX ismerete

Mozoghat egy taszk a CCX-ek között? Az L3 adat elveszik, a teljesítmény esik.

- többszálú végrehajtás

8 mag 16 szála között hogyan osszuk el a taszkokat?



Forrás: AMD


```

C:\Windows\system32\cmd.exe
Application - ScaLibTestApp - WIN32_MSC ( 64-bit ) - Release
Tests: Start
> Test0001 Start <
*****
Configuration - WIN32_MSC ( 64-bit ) - Release
CTestSet::InitTestEnv - Passed

* CMatrixSet Start *
> TMatrixSet Classes <
> TMatrixSet Methods <
> CMatrixSet Methods <
> CMatrixSet Algorithms <
CStrassenSet Methods - CStrassenSet Object - Created

* CStrassenSet Start *
> CStrassenSet Methods <
> CStrassenSet Algorithms <
Strassen HBI
Matrix Size      : 16384 x 16384
Matrix Size Threshold : 8192 x 8192
Matrix Partitions : 8
Result Sets Reflection: N/A
Calculating...
OMP: Info #204: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #202: KMP_AFFINITY: Affinity capable, using global cpuid leaf 11 info
OMP: Info #154: KMP_AFFINITY: Initial OS proc set respected: {0,1,2,3,4,5,6,7}
OMP: Info #156: KMP_AFFINITY: 8 available OS procs
OMP: Info #157: KMP_AFFINITY: Uniform topology
OMP: Info #179: KMP_AFFINITY: 1 packages x 4 cores/pkg x 2 threads/core (4 total)
OMP: Info #206: KMP_AFFINITY: OS proc to physical thread map:
OMP: Info #171: KMP_AFFINITY: OS proc 0 maps to package 0 core 0 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 1 maps to package 0 core 0 thread 1
OMP: Info #171: KMP_AFFINITY: OS proc 2 maps to package 0 core 1 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 3 maps to package 0 core 1 thread 1
OMP: Info #171: KMP_AFFINITY: OS proc 4 maps to package 0 core 2 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 5 maps to package 0 core 2 thread 1
OMP: Info #171: KMP_AFFINITY: OS proc 6 maps to package 0 core 3 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 7 maps to package 0 core 3 thread 1
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {2}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {4}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {6}
Strassen HBI - Pass 01 - Completed: 53.82100 secs
Strassen HBI - Pass 02 - Completed: 53.60100 secs
Strassen HBI - Pass 03 - Completed: 53.54000 secs
Strassen HBI - Pass 04 - Completed: 53.60200 secs
Strassen HBI - Pass 05 - Completed: 53.58600 secs
ALGORITHM_STRASSENHBI - Passed
* CStrassenSet End *
CStrassenSet Methods - CStrassenSet Object - Deleted
* CMatrixSet End *

Test Completed in 273719 ticks
> Test0001 End <
Tests: Completed
Press any key to continue . . .


```

Windows Task Manager

File Options View Help


Applications Processes Services Performance Networking Users

CPU Usage




0 %

CPU Usage History




Memory



2.59 GB

Physical Memory Usage History



Physical Memory (MB)

Total	32641
Cached	1696
Available	29984
Free	28368

Kernel Memory (MB)

Paged	350
Nonpaged	122

System

Handles	10382
Threads	536
Processes	35
Up Time	0:02:48:26
Commit (GB)	2 / 159

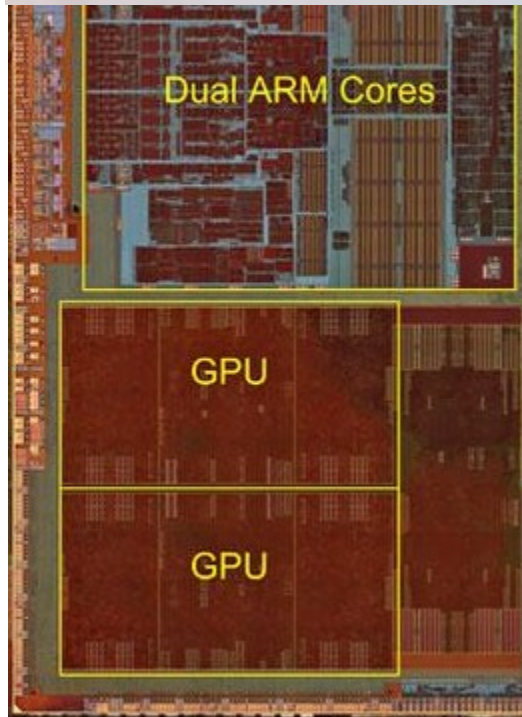
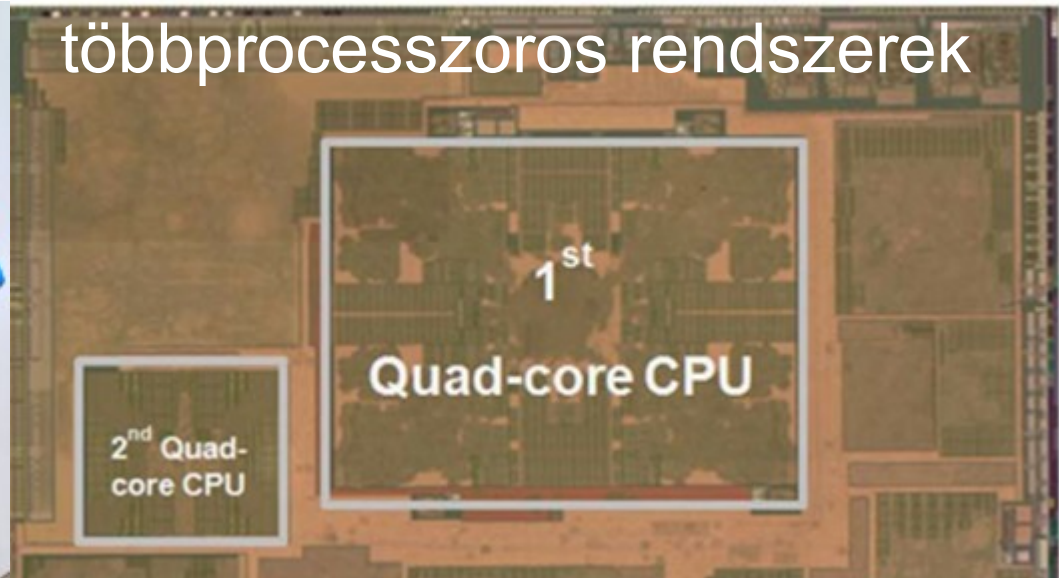
Resource Monitor...

Processes: 35 CPU Usage: 0% Physical Memory: 8%

Heterogén

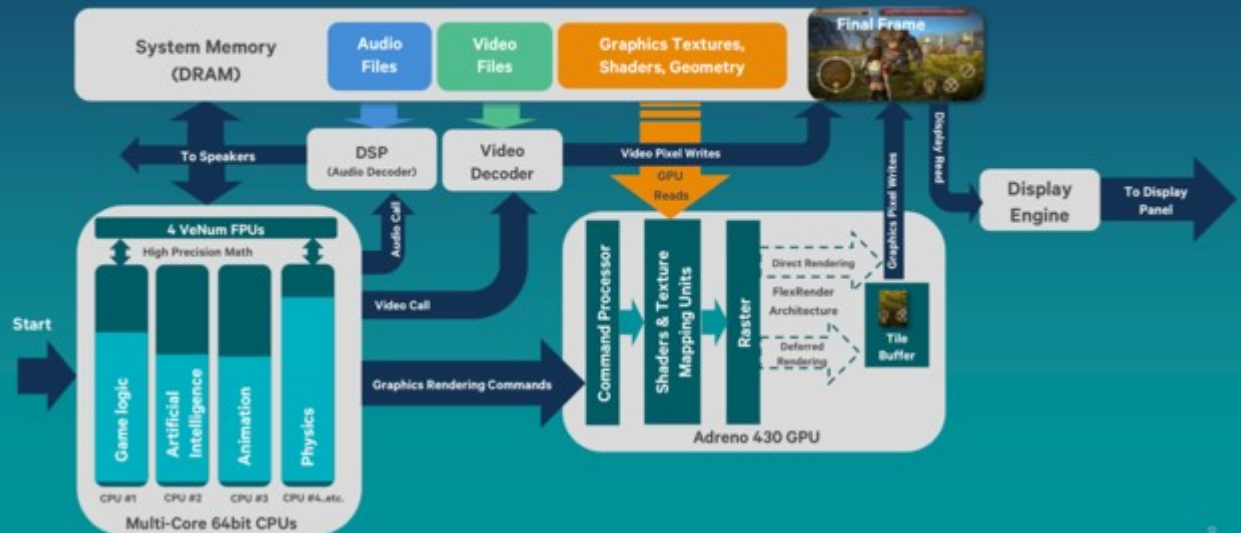


többprocesszoros rendszerek



Advantages of heterogeneous architecture for gaming use cases

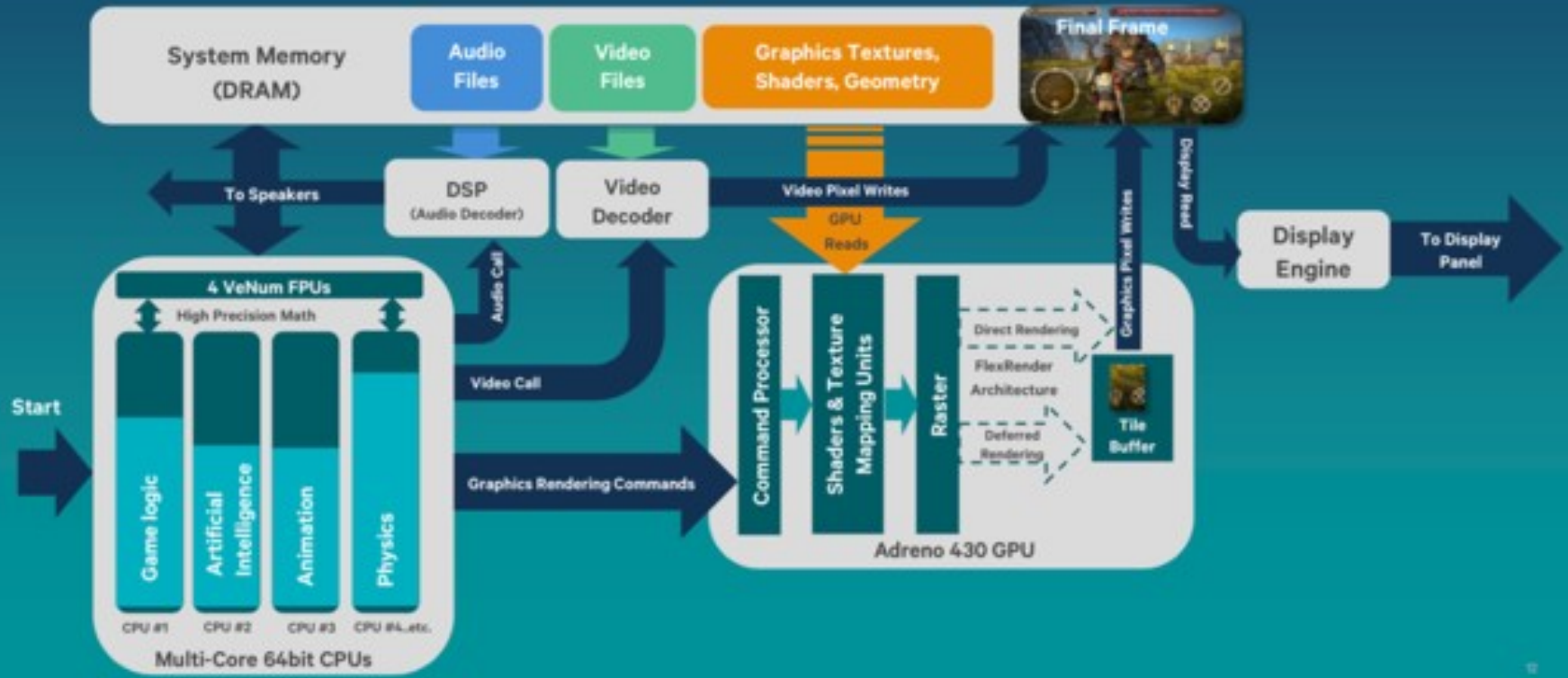
Heterogeneous hardware blocks and data flow



Feladatok – taszkok – végrehajtó egységek

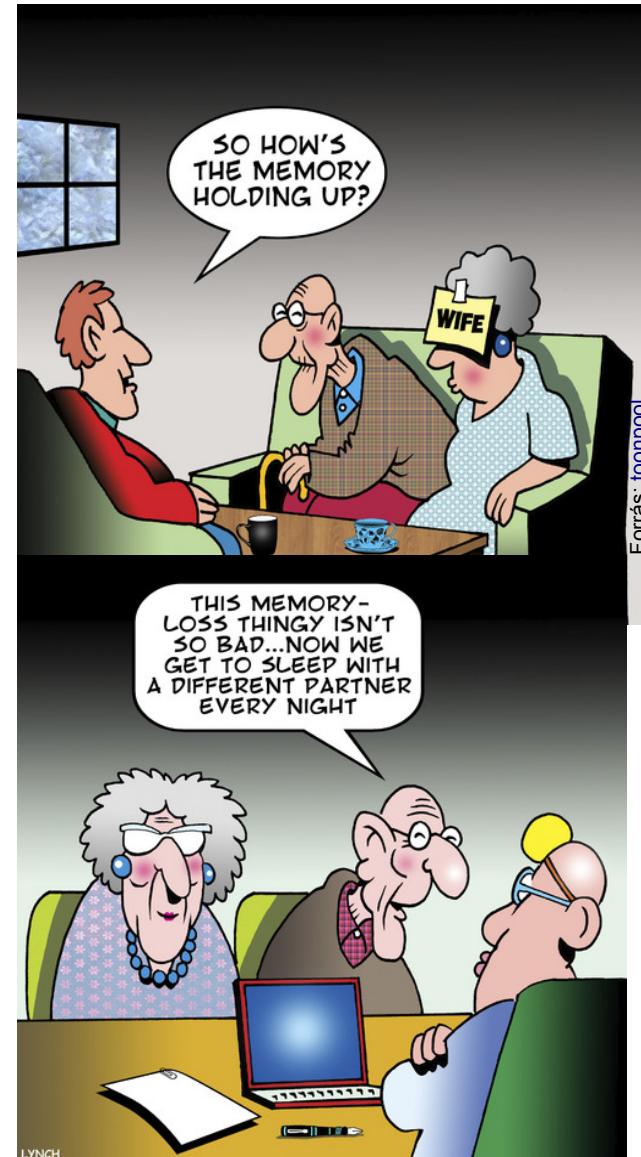
Advantages of heterogeneous architecture for gaming use cases

Heterogeneous hardware blocks and data flow



Mivel foglalkozik a memóriakezelés?

- Kiosztja az erőforrást
 - erőforrás: fizikai memória
 - igénylők: taszkok és kernel
- Elhelyezi a taszkok adatait
 - programkód + statikus
 - dinamikusan allokált
- Elhelyezi a kernel adatait
 - programkód
 - adminisztratív adatok
- Biztosítja a védelmet
 - szeparáció
 - hibák
- Támogatja a kommunikációt
 - adatcsere taszkok között



Fájl- és tárolórendszerek

Felhasználói szemmel...

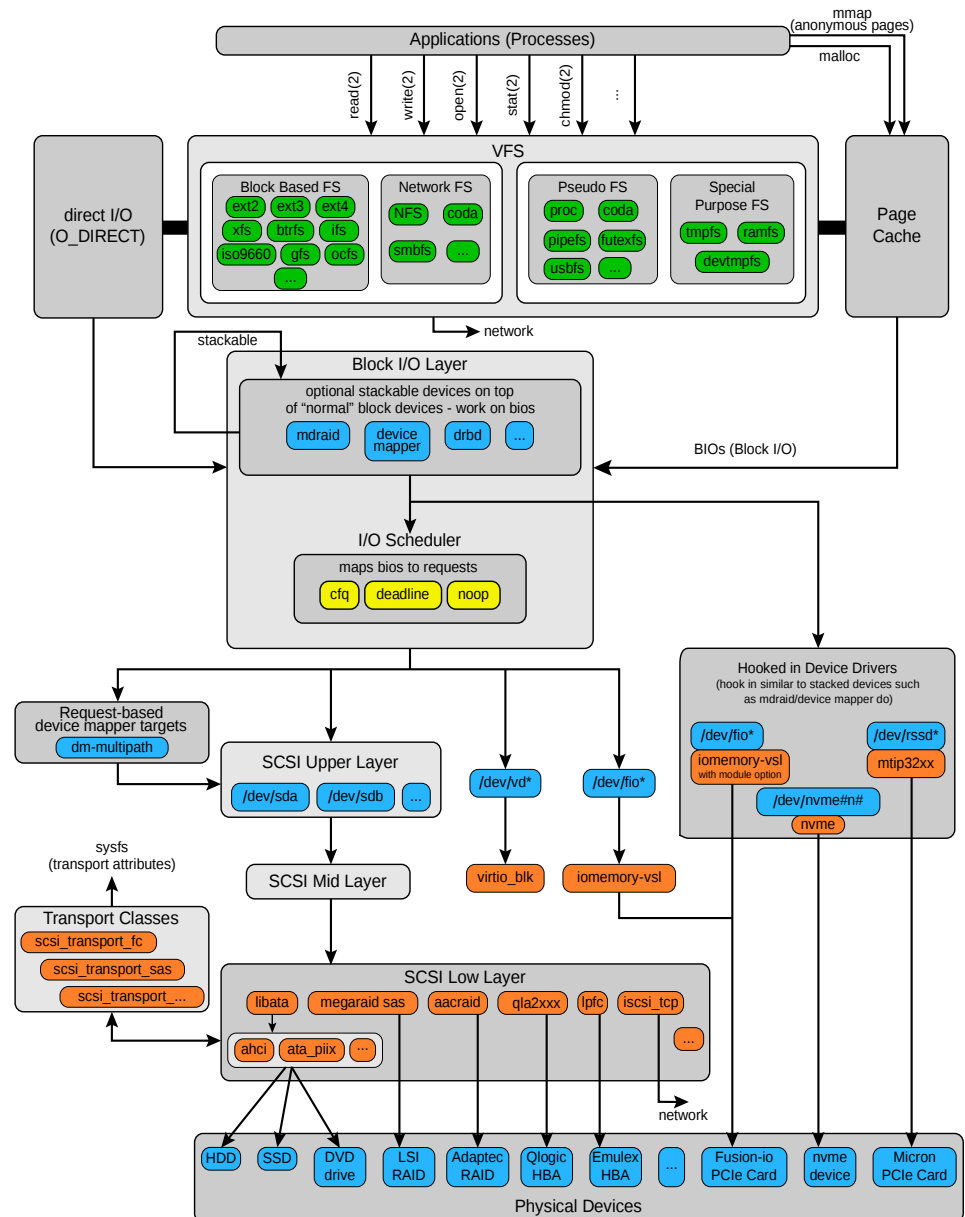
- végfelhasználó
- adminisztrátor
- programozó

Belső működés

- fájlrendszer interfészek
- kernel adatstruktúrák
- a háttértár szervezése
- virtuális fájlrendszerek

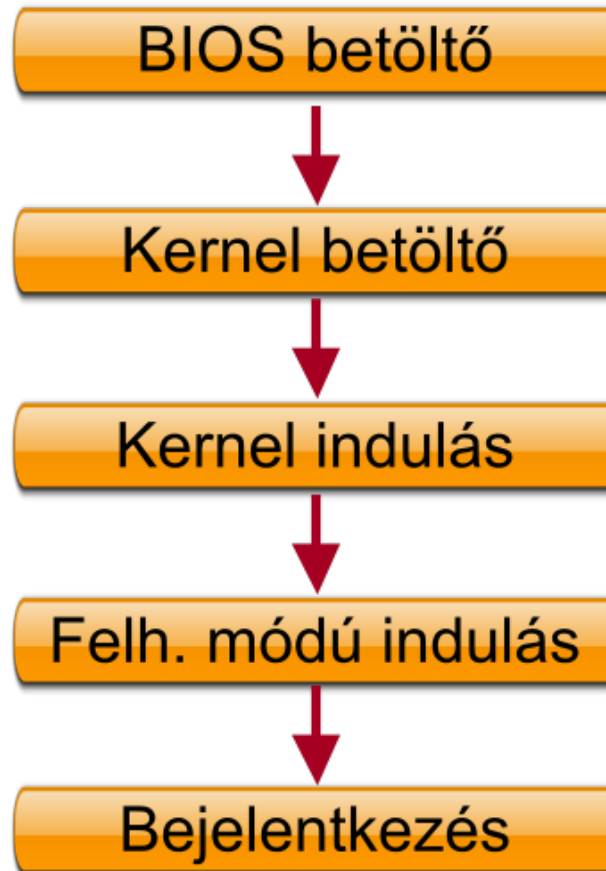
Adattárolás

- fizikai tárolók (HDD, SSD)
- I/O ütemezés
- tárolórendszer-virtualizáció:
 - helyi (RAID, LVM)
 - hálózati (SAN, NAS)
- elosztott fájl- és tárolórendszerek



Az operációs rendszerek működése

Rendszerindulás



Az operációs rendszer indulása (betöltő szint)

- Bekapcsoljuk a gépet
 - elindul a rendszerórajel, ami a processzor inicializációját indukálja
 - a processzor végrehajt egy fix címen kezdődő programot (ROM betöltő)
- 0. szintű (ROM) betöltő (BIOS, boot ROM)
 - hardverinicializálási feladatok (POST)
 - betöltőeszköz meghatározása
- 1. szintű (RAM) betöltő (MBR vagy GPT)
 - ismeri a háttértár felépítését (partíciók),
 - betölti a következő szintű betöltőt az aktív partícióról
- 2. szintű (OS) betöltő (Boot loader @ PBR / VBR)
 - már ismeri az OS-t (pl. fájlrendszer felépítése, kernel betöltése stb.)
 - betölthet további programrészeket (Windows: **Bootmgr**, Linux: **Grub** stage2)
 - rendelkezhet felhasználói felülettel (az indítandó OS és opcióinak megadására)
 - betölti a kernel kódját és elindítja
- Elindul a kernel

A Linux kernel indulása

- Inicializálás nem védett módban
 - alap memóriakezelés, kernel verem, megszakítások stb.
 - indulási paraméterek átvétele
 - alapvető hardverek (pl. konzol / videokártya) beállítása
- Védett (és 64 bites) módba váltás
- Inicializálás védett módban
 - teljes memória, védett módú memóriakezelés
 - megszakításvektorok kezelőfüggvényei
 - a feladatkezelés adatstruktúrái
 - rendszerindulási eszközmeghajtók (initrd: initial ram disk)
 - ütemező
 - további architektúrafüggő feladatok
 - az első feladat, az **init** programkódjának betöltése és elindítása
- Fut az első felhasználói módú folyamat, az init

A Windows kernel indulási folyamata

- **Bootmgr** – 2. szintű betöltő
 - védett módba vált
 - még a BIOS eszközközkezelőkre támaszkodik
 - megjeleníti a boot menüt
- **Winload** – a kernel betöltője
 - 32/64 bites védett módban működik
 - betölti az Ntoskrnl.exe-t és függőségeit, valamint az induláskori eszközközkezelőket
 - átadja a rendszerindulási paramétereket az Ntoskrnl-nek
- **Ntoskrnl és HAL** – a kernel és a hardverkezelő
 - 32/64 bites védett módban működik
 - **0. fázis** (phase 0) – inicializálás letiltott megszakításokkal
 - boot processzor, kernel adatstruktúrák, zárolási táblák stb.
 - megszakításvezérlők és -kezelők
 - a memóriamenedzsmment és a feladatkezelés
 - **1. fázis** (phase 1) – további inicializálás a normál feladatkezelés keretében
 - az összes CPU beállítása, megszakítások engedélyezése
 - videó (folyamatjelző sáv), I/O és több tucatnyi más alrendszer
 - kernel ütemező
 - elindul a munkamenet-kezelő (SMSS)

A Windows felhasználói módú kritikus folyamatai

- **SMSS** – Munkamenet-kezelő (session manager)
 - a felhasználói módú működés alapvető beállítása
 - fájlrendszerek ellenőrzése
 - környezeti változók
 - lapozófájlok
 - registry
 - Wininit indítása (S0 InitialCommand)
 - munkamenetek (CSRSS) és bejelentkezés-kezelő (Winlogon) indítása (S1+)
 - A Winlogon kilépéséig fut (vár)
- **Wininit** – további felhasználói inicializálási lépések (Session 0)
 - pl. Service Control Manager (services.exe)
- **CSRSS** – Win32 alrendszer indítása (Session 1+)
 - további hardver-inicializálás (pl. teljes grafikus felbontás)
- **Winlogon** – felhasználói bejelentkezés (Session 1+)
 - bejelentkezési képernyő megjelenítése (LogonUI)

A Unix rendszerek felhasználói módú **indulása**

- Az init indítja...
 - meghatározza és fenntartja a rendszer működési szintjét (**/etc/inittab**)
 - elindítja/leállítja a szükséges OS szolgáltatásokat
- **Futási szint** (runlevel)
 - az OS állapotleírása
 - a működési aktuális módja (karbantartás, többfelhasználós, grafikus stb.)
 - rendszerszolgáltatások köre
 - jellemzően számmal (0-6), vagy betűvel jelölik
 - **0**: teljes leállítás
 - **1** vagy **S**: single-user: egyfelhasználós (adminisztrátori) mód
 - **2-5**: többfelhasználós üzemmódok (GUI, ha van)
 - jellemzően az **5** az alapértelmezett teljes felhasználói mód
 - **6**: újraindítás
 - A rendszergazda válthatja: `telinit`, `init`, `shutdown`, `halt`, `reboot`
 - Lekérdezhető: `who -r`
- Az android-alapú rendszerek **indulása** hasonló

A Unix rendszerszolgáltatások kezelése

- Az init működése
 - konfiguráció: `/etc/init.d/` és `/etc/rc?.d/` (? a runlevel)
 - a parancsfájlok
 - nevüknek megfelelő sorrendben futnak le
 - beállítják a rendszert
 - pl. fájlrendszerek ellenőrzése és csatolása
 - elindítják, illetve leállítják a szolgáltatásokat
 - pl. felhasználói bejelentkezés, grafikus felület, adatbázis- és webszerver stb.
- Az adminisztrátor meghatározhatja az aktív rendszerfeladatok körét
 - a parancsfájlok manuálisan is meghívhatók

```
service <parancsfájl-neve> <start|stop|restart|...>
```

- beállítható az adott futási szinten aktív szolgáltatások, elvégzendő feladatok köre

```
ntsysv, tksysv, chkconfig, bum
```

A Sysinit alternatívái

- Mi a gond az init-tel?
 - egysíkú függőségek (fájlok nevei)
 - lassan (egyesével) indítja a szolgáltatásokat
 - hibakezelés?
- **Systemd** (RedHat, CentOS, Ubuntu 15.04+, Arch Linux, Debian stb.)
 - deklaratív szolgáltatásleírások, pontosabb függőségek
 - párhuzamos / késleltetett indítás
 - működési hibák észlelése és kezelése
 - megváltozott parancskészlet:

```
systemctl <start|stop|restart|...> <szolgáltatás/feladat>
```

- a bevezetése főleg Debian/Ubuntu-körökben elég sok **vihart kavart**
- Futottak még:
 - Upstart (Debian és Ubuntu korábbi változatai, Google Chrome/Chromium OS)

Hogyan áll le a Windows operációs rendszer?

- **ExitWindowsEx()** rendszerhívás (sokféle paraméterezés)
 - leállítja a futó alkalmazásokat
 - kijelentkezteti a felhasználót
 - kezdeményezi az operációs rendszer leállítását
- Explorer → Winlogon
 - Start menü – Leállítás (Explorer)
 - Értesíti a CSRSS-t a rendszer leállítási kérésről
- CSRSS (Win32 alrendszer)
 - kilépteti az összes felhasználót (session 1+) az alábbiak szerint
 - végiglépked az összes futó alkalmazáson (shutdown order)
 - és leállítja azokat
 - ha egy taszk nem áll le (HungAppTimeout), akkor jelzi ezt a felhasználó felé
 - hasonló módon leállítja a szolgáltatásokat (session 0)
 - a HungAppTimeout itt is működik, csak nincs jelzés róla
- Winlogon
 - az összes CSRSS leállítási procedura után
 - meghívja az NtShutdownSystem() rendszerhívást
 - amely a PoSetSystemPowerState() híváson keresztül a kernelszintű részek leállítását végzi

Hogyan áll le a Unix rendszer?

- Elindítjuk a leállítási folyamatot
 - pl.: shutdown +60 "System going down for regular maintenance"
- Az init (PID 1) értesül a leállítási szándékról
 - értesíti az interaktív felhasználókat a leállításról
 - „Broadcast message from root@localhost
 - The system is going down for regular maintenance in 60 seconds”
 - a szolgáltatások konfigurációi alapján leállítja azokat a megfelelő sorrendben
 - szinkronizálja majd lecsatolja a fájlrendszereket*
 - leállítja a még futó taszkokat
 - elindítja a kernel leállítási folyamatát ([reboot\(\)](#))
- A kernel [leállítása](#)
 - letiltja a felhasználói módú működést
 - értesíti a komponenseit a leállásról (reboot_notifier_list)
 - leállítja a hardvereszközöket a vezérlőiken keresztül
(lecsatolja a még megmaradt fájlrendszereket)
 - leállítja az alapvető funkcióit (pl. megszakításkezelés)
 - leállítja a számítógépet

A kernel belső felépítése

Mekkora a kernel forráskódja?

- Példák:
 - World of Warcraft: 5.5 millió programsor (LOC)
 - Windows XP: 45 millió LOC (nem csak a kernel)
 - Linux kernel: ~60 ezer fájl, ~25 millió LOC (~ fele eszközközelés)
 - MINIX alap kernel < 1400 LOC, teljes kernel kb. 5000 LOC
- Érdekességek
 - <http://www.informationisbeautiful.net/visualizations/million-lines-of-code/>
 - <http://www.pabr.org/kernel3d/kernel3d.html>
 - <http://www.jukie.net/bart/blog/linux-kernel-walkthroughs>
 - http://en.wikiversity.org/wiki/Reading_the_Linux_Kernel_Sources
 - Linux vs. Windows kernel (videó, Mark Russinovich)
 - Minix OS fut Intel AMT mikrocsipekben (Tannenbaum levele az Intelhez)



A kernel felépítésének alapelvei

- Réteges
 - jól definiált (szabványos?) **interfészekkel**

*A **rendszerhívás interfész** egy programozói felület, amely a kernel felhasználói módban működő programok számára nyújtott szolgáltatásait tartalmazza.*

- Monolitikus
 - a kernel részei egyetlen címtérben találhatók
 - egyszerű fejlesztés vs. megbízhatóság

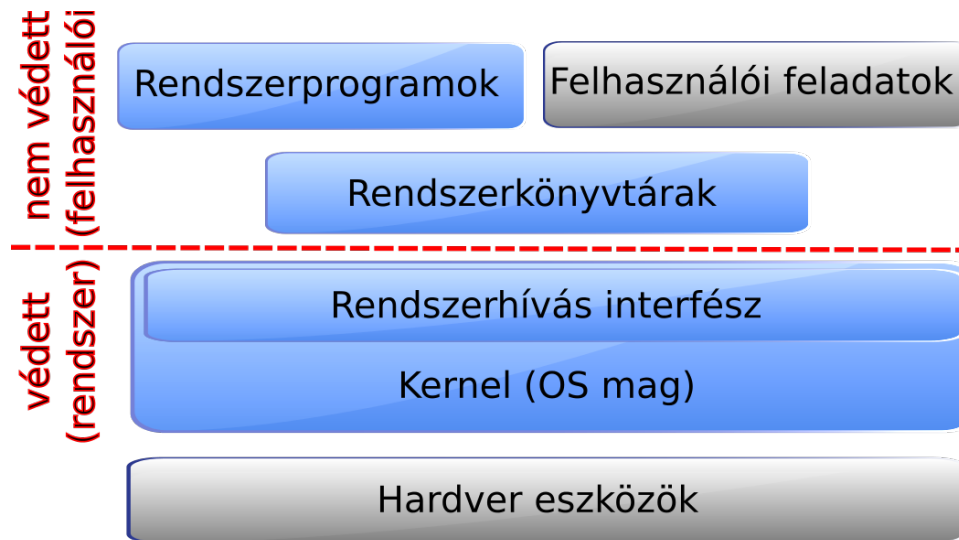
- Moduláris
 - nem érhető el mindig minden rész
 - fordítási időben / konfiguráció során / futásidőben

*A mai kernelek jellemzően
moduláris, monolitikus szoftverek*

*Linux: vmlinux
Windows: ntoskrnl.exe*

- Elosztott
 - önálló komponensek (külön címtérben)
 - üzenetalapú kommunikáció

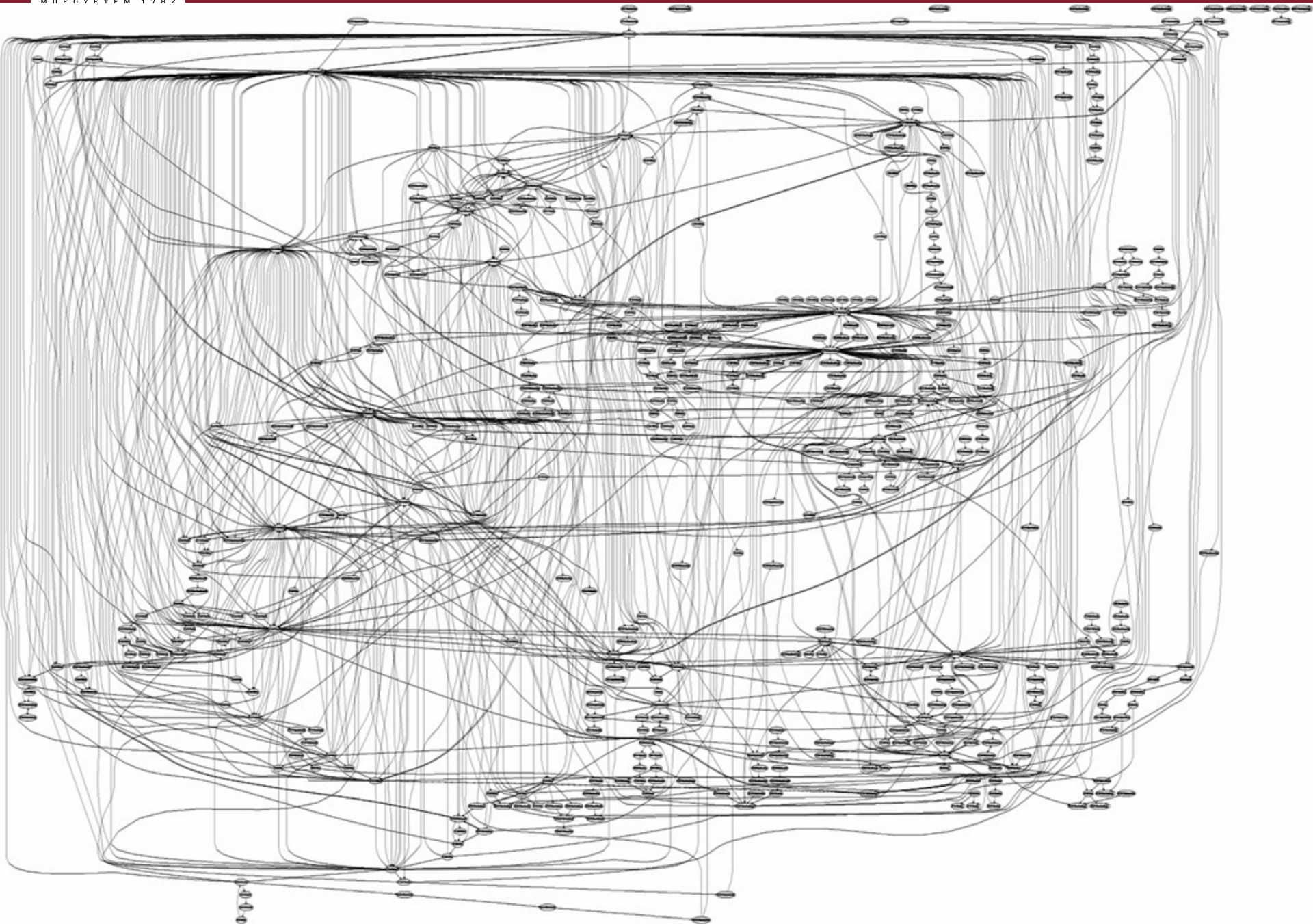
A rendszerhívások működése



- A rendszerhívás látszólag egy függvényhívás, de valójában más
 - üzem módváltás szoftver megszakítással
`trap, syscall, sysenter`
 - a kernel megszakításkezelője
 - átveszi a paramétereket (nem a vermen keresztül)
 - végrehajtja a feladatokat
 - visszatér a megszakításból (`iret, sysexit`)

Rendszerhívások működése Unix alatt

- a rendszerhívás meghívása (`read()`, `write()` stb.)
 - klasszikus függvényhívásnak tűnik
 - a **`libc`** rendszerkönyvtár implementálja
 - csak a valódi rendszerhívás előkészítését végzi
- a `libc` kiadja a `SYSCALL` utasítást (megszakítást generál)
- a kernel `SYSCALL` kezelője előkészíti a rendszerhívás végrehajtását
- végrehajtódik a kernel módú eljárás (a tényleges rendszerhívás)
- a kernel visszatér a megszakításból (`iret`, `sysexit`)
 - folytatódik a `libc` segédfüggvénye, amely beállítja a visszatérési értéket
- a `libc` visszatér a folyamat által meghívott függvényből



Virtuális rendszerhívások

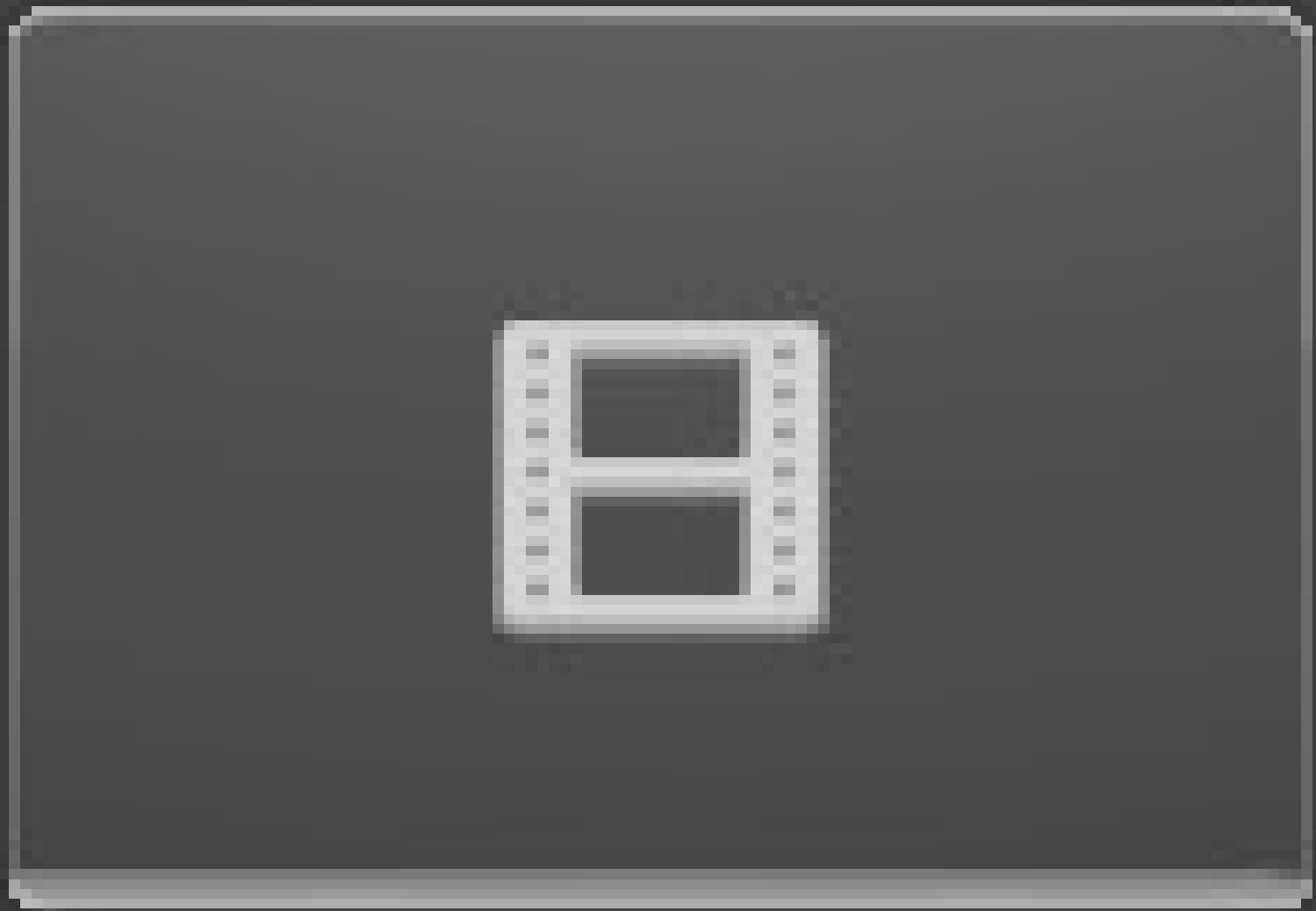
- A rendszerhívás rendkívül gyakori és költséges eljárás
 - szoftver megszakítás
 - üzemmódváltás
 - megszakítás-kezelés
- Hogyan csökkenthető a rezsiköltség?
 - ötlet: próbáljuk elkerülni a megszakítást és az üzemmódváltást
 - bizonyos kernel funkciók elérhetők a felhasználói címtérben
- Virtuális rendszerhívások (Linux)
 - speciális „kernel” memóriaterület felhasználói címtérben
 - biztonságosnak ítélt rendszerhívások érhetők el rajta
 - szoftver megszakítás és módváltás nélkül működnek
 - a felhasználók programjai nem látják a különbséget (a libc igen)

Mi a baj a kernelek felépítésével?

- Hatalmas kódbázis, és **emberek írják**
 - 1000 soronként kb. 10-100 hiba az átadott programban (**forrás**)
 - egy mai kernel több millió programsor.....
 - ~~hibaizoláció~~, futásidejű javítás hibák, kártevők



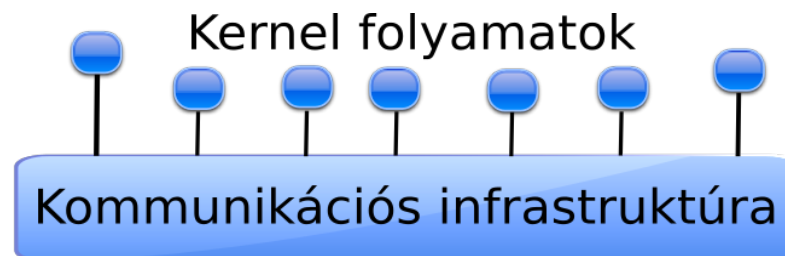
Forrás: [Linux.com](https://www.linux.com) (2016)



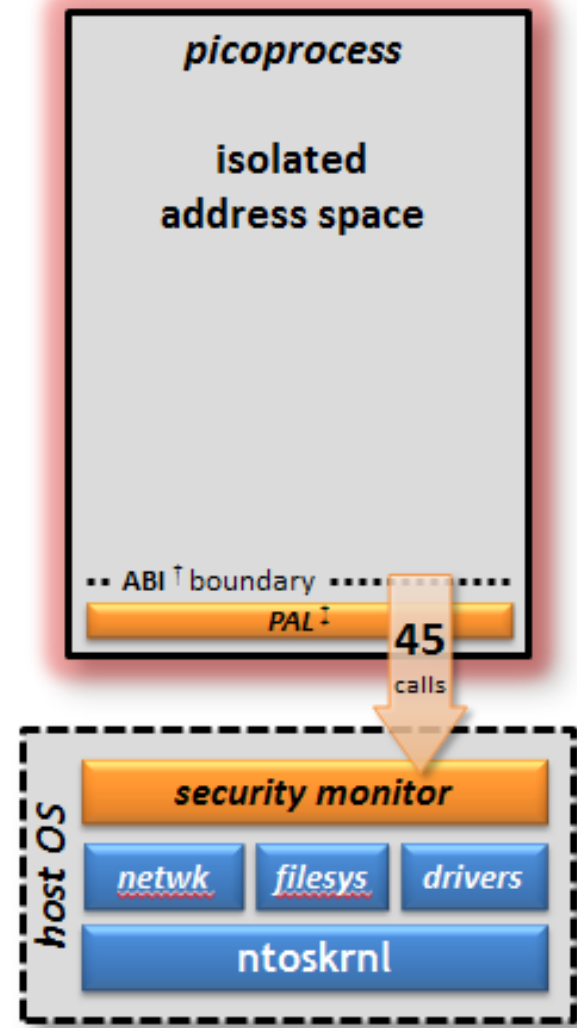
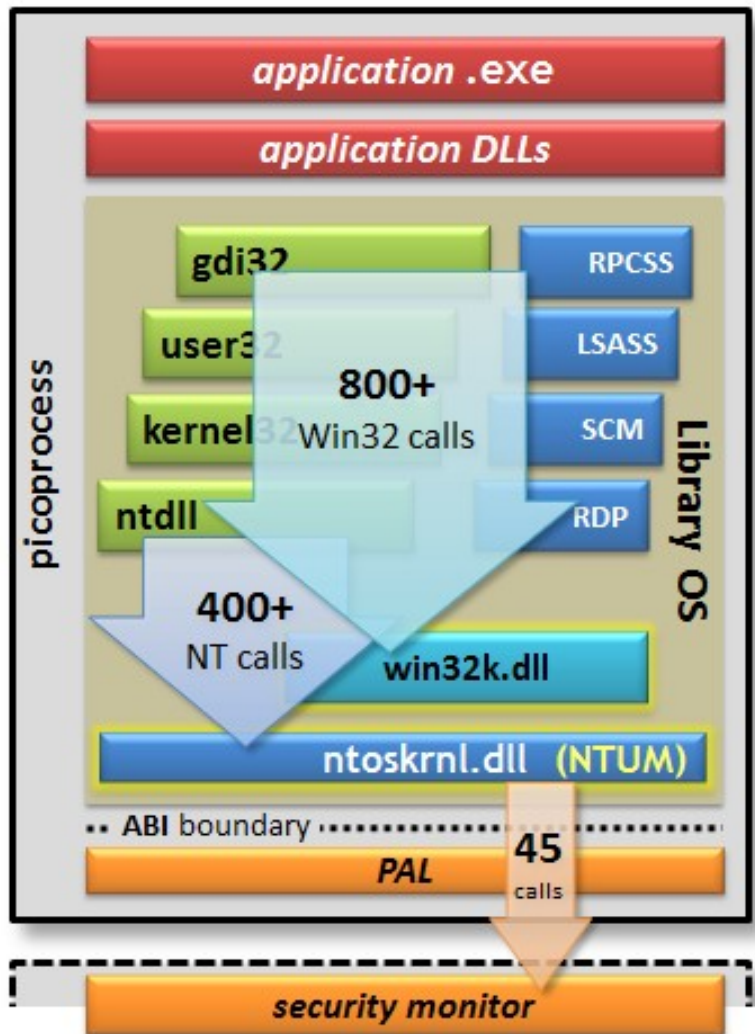
Mit **tehetünk** a helyzet javítása érdekében?

- Kernel sandboxing **armored OS**
 - a kezelőfüggvényeket „védőréteggel” látja el (hibadetektálás)
 - egy felhasználói módú helyreállító ágens kezeli a felmerült problémákat
- OS/app sandboxing **KVM/vmware, Docker, MirageOS, Drawbridge**
 - kisebb felületű rétegek, erősebb szeparáció
 - virtualizáció: még egy felügyeleti szint
 - konténerek: ugyanazon a kernelen független OS-ek
 - unikernel: mini kernel (library OS) + alkalmazás egy címtérben
- Kidobjuk a monolitikus felépítést
 - elosztott rendszer (feladat-végrehajtók és kommunikáció)
 - védett módban csak a legszükségesebb részek működnek

Kritika



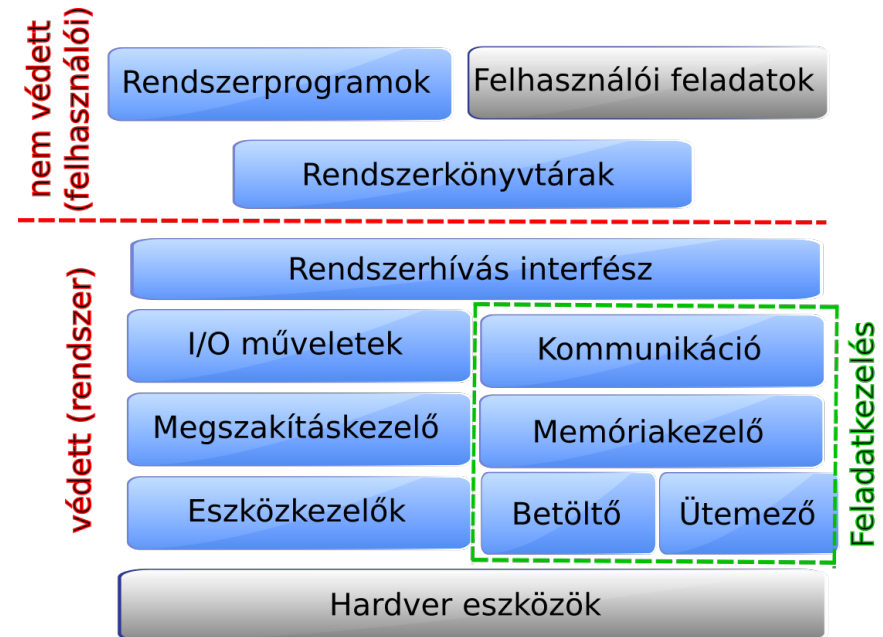
Windows Library OS és pProcess (Drawbridge koncepció)



*A **mikrokernel** egy olyan operációs rendszer kernel, amely csak az alaplűködéshez feltétlenül szükséges kódrészleteket tartalmazza, minden más funkciót felhasználói módban működtet.*

- [illegible]

Mikro- vs. monolitikus kernel



Újgenerációs mikrokernel

- L4 mikrokernel
 - akár CPU regiszterekben is átvihető az üzenet
 - 10-20-szor gyorsabbak, mint a klasszikus mikrokernel IPC
 - nagyon kevés védett módú funkció (az L4 API 7 funkcióval rendelkezik)
 - a védett módú kernel nagyon kicsi (5-15 ezer programsor)
 - speciális ütemezést alkalmaznak (sok a blokk az IPC üzenetek miatt, ezt kezelik)
 - erősen hardverfüggőek (még x86-on belül is sokféle implementáció szükséges)

a kis kernel lehetővé teszi a formális leírást és a [verifikációt](#)

- Hibrid kernelek
 - monolitikus rendszerekkel vegyített mikrokernel
 - OS X [XNU](#) (az Apple kernele), egy Mach mikrokernel + BSD Unix hibrid kernel
 - Kísérleteztek L4 mikrokernelre épülő reinkarnációjával is, lásd [Lee & Gray, 2006](#)
 - A Windows is tartalmaz mikrokernel elemeket, de nem mikrokernel felépítésű.

Az L4 mikrokernel családfája

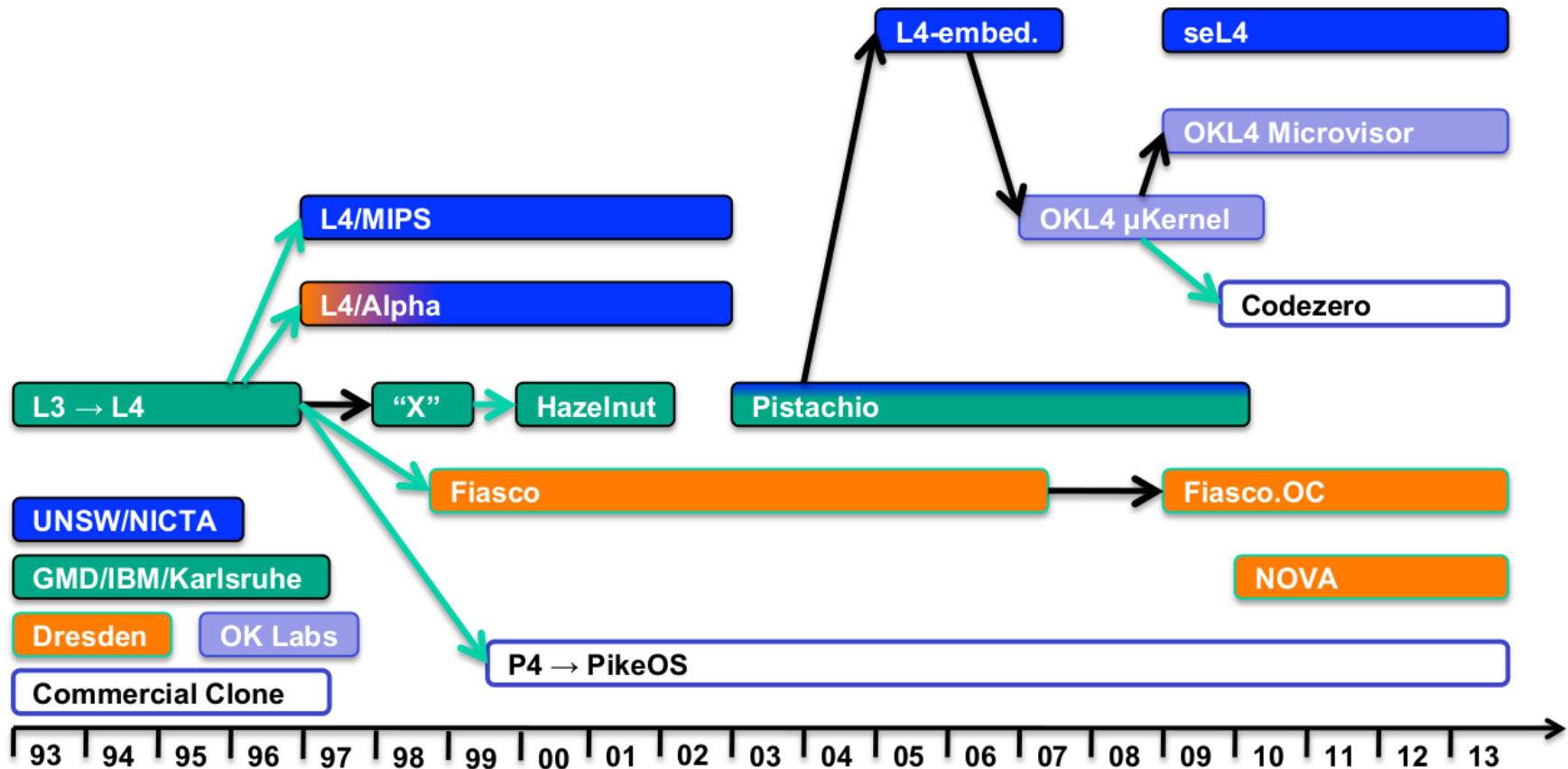


Figure 1: The L4 family tree (simplified). Black arrows indicate code, green arrows ABI inheritance. Box colours indicate origin as per key at the bottom left.

Forrás: Kevin Elphinstone, Gernot Heiser, From L3 to seL4 what have we learnt in 20 years of L4 microkernels?

Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, November 03-06, 2013, Farmington, Pennsylvania

Összefoglalás

- A kernel egy komplex program
 - jellemzően réteges, moduláris és monolitikus szerkezetű
 - ez utóbbi számos megbízhatósági és biztonsági problémával küzd
 - a mikrokernelek próbálnak ezen segíteni
- Az operációs rendszer indulása egy komplex eljárássorozat
 - ROM, RAM, OS és kernel saját betöltő
 - kernel és felhasználói módú indulás
- Az operációs rendszer működése
 - rendszerszolgáltatások
 - felhasználói munkamenetek
 - rendszerhívások
- Javasolt otthoni gyakorlatok (virtuális géppel)
 - OS telepítése, rendszerindulás, szolgáltatáskezelés, rendszerhívás-nyomkövetés