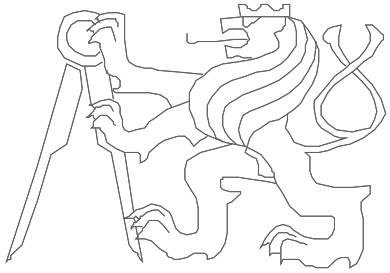
Architektura počítačů

Víceúrovňový model počítače, virtualizace



České vysoké učení technické, Fakulta elektrotechnická

Mnohaúrovňová organizace počítače

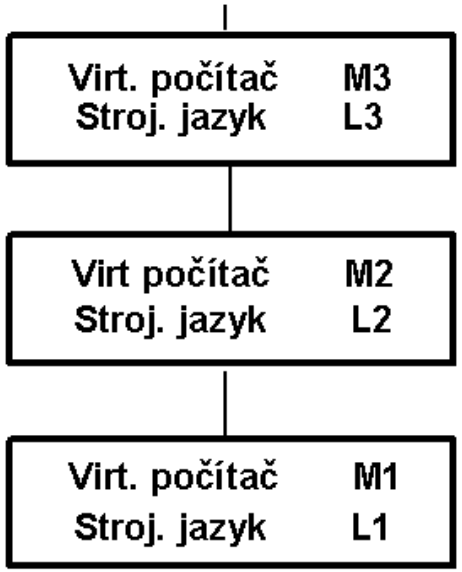
Strojový jazyk počítače - množ. jedn. instr. - do ní převést prog. pro výkon
- úroveň L1 - abeceda {0,1} , obtížná komunikace

Jazyk vyšší úrovně - vhodnější pro lidskou komunikaci L2 + další

Vykonání programu v L2 na stroji jenž má L1

Kompilace - instrukce v L2 se nahradí posloupností instrukcí v L1

Interpretace - program v L1 zpracovává program v L2 jako data (pomalejší)



PRGM v L3 je interpretován programem na M2 nebo M1, nebo je přeložen do L2 nebo L1

PRGM v L2 je interpretován programem běžícím na M1 nebo přeložen do L1

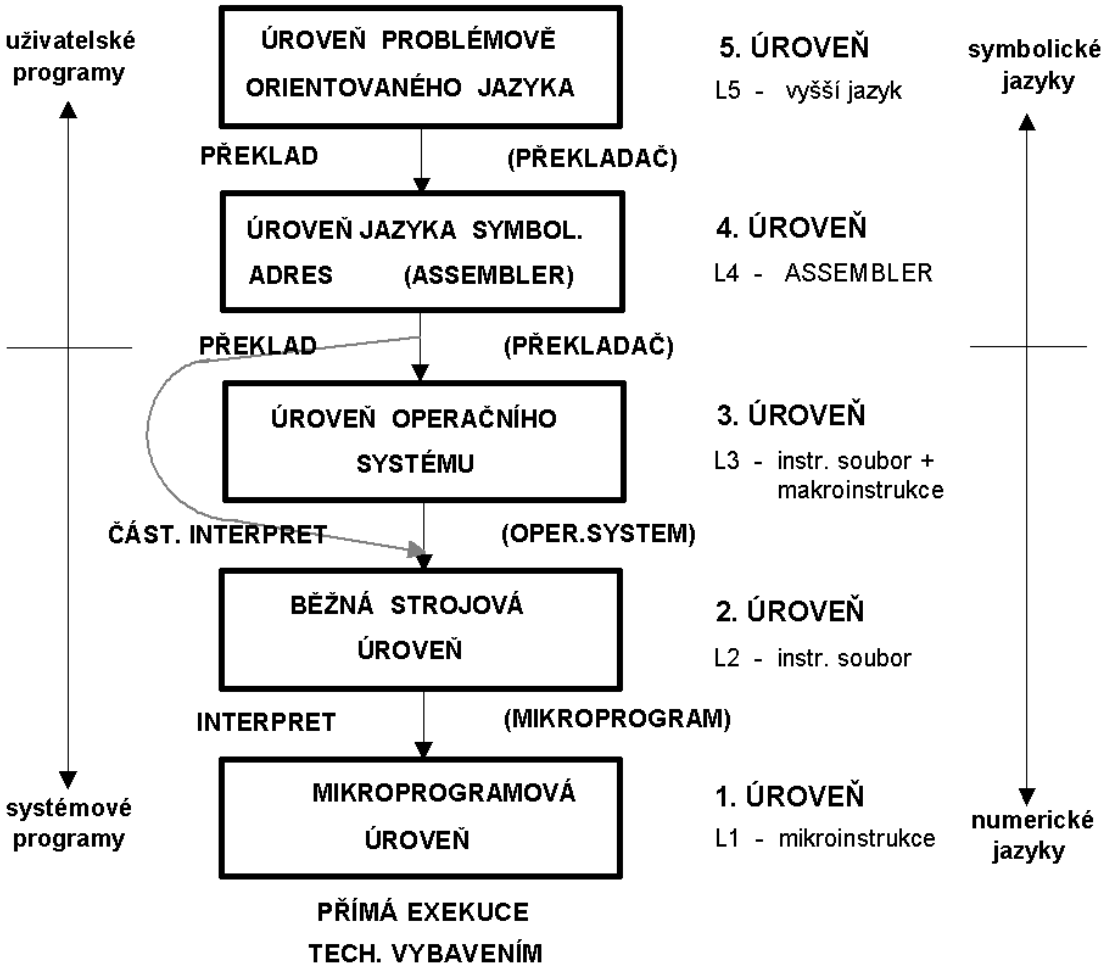
PRGM v L1 je skutečně vykonáván technickými prostředky

Virtuální počítač

Na úrovni i zavádíme virtuální počítač M_i s jazykem Li.

Program v Li je překládán nebo interpretován počítačem M_{i-1} atd.

Současný mnohaúrovňový počítač



Vývoj víceúr. stroje

- první počítače – běžná strojová úroveň - 1.úr.
- 50- tá léta - Wilkes – mikroprogram. - 2.úr.
- 60- tá léta – operační systémy - 3.úr.
- překladače, program. jazyky - 4.úr.
- uživatelské aplikační programy - 5.úr.
- HW a SW jsou logicky ekvivalentní (lze je navzájem nahradit)
- souboj a splývání RISC a CISC procesorů

Procesy a jejich stavy

PROCES - probíhající program (program - pasivní, proces - aktivní)

STAV PROCESU - info, které při zast. procesu umožní jeho znovuspuštění

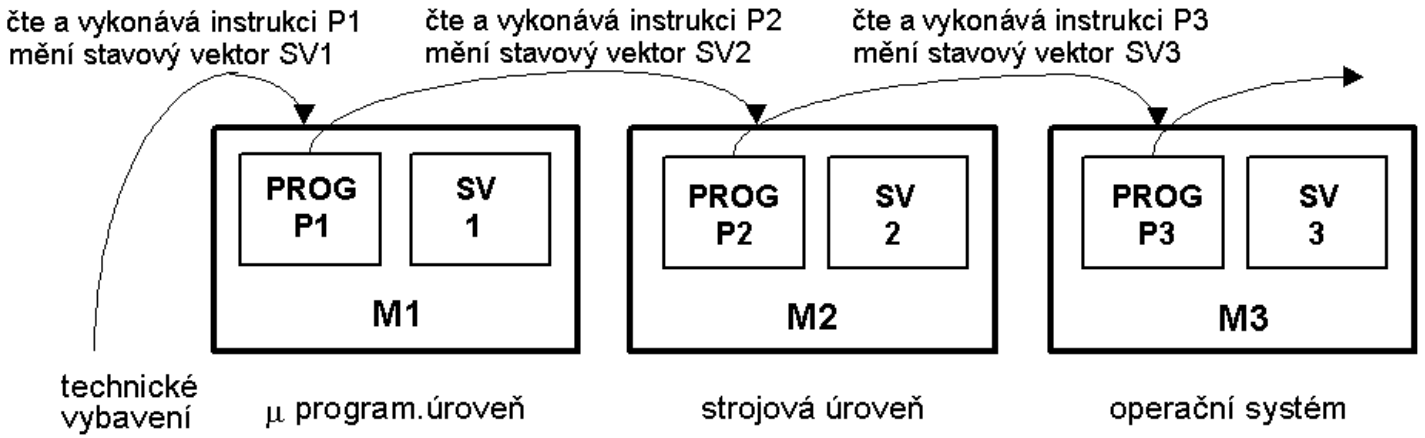
1. program
2. následující instrukce
3. hodnoty proměnných a data
4. stavy a polohy všech I/O

PŘEDPOKLAD: proces P sám nemění svůj program!

STAVOVÝ VEKTOR - proměnné složky stavu procesu – mění HW nebo prgm.

PROCES = PROG + STAV. VEKT.

ZMĚNA STAV. V. – stav. vektor proc. P2 mění P1 -> P1 interpret programu P2



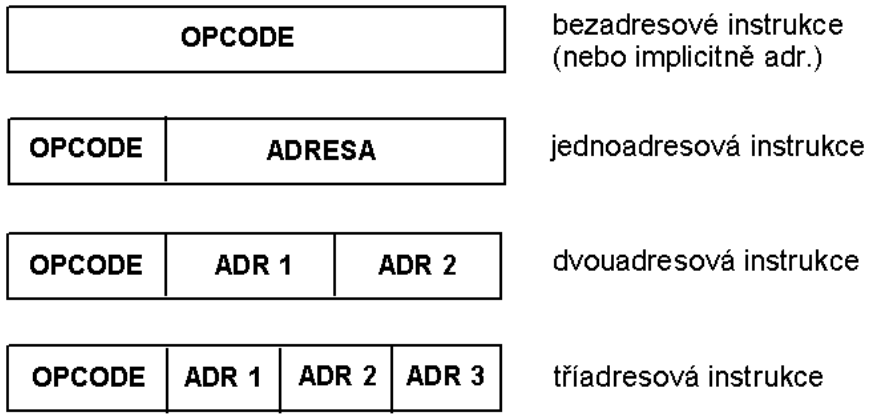
Konvenční strojová úroveň (ISA) (M2)

Definovaná Instrukční sadou (často označovaná architektura procesoru)
ISA – Instruction Set Architecture × Mikroarchitektura (Implementece v M1)

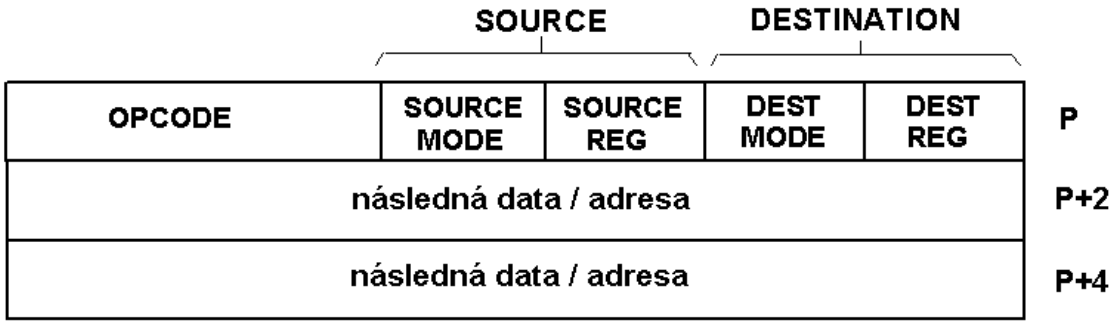
HW - strukt. poč., proc., I/O kan., sběrnic, org. a příst. k pam., org.reg. a j.
SW - instr.soubor, formát dat a ulož. v pam., adresování, zás.pam, reg aj.

Instrukční formát

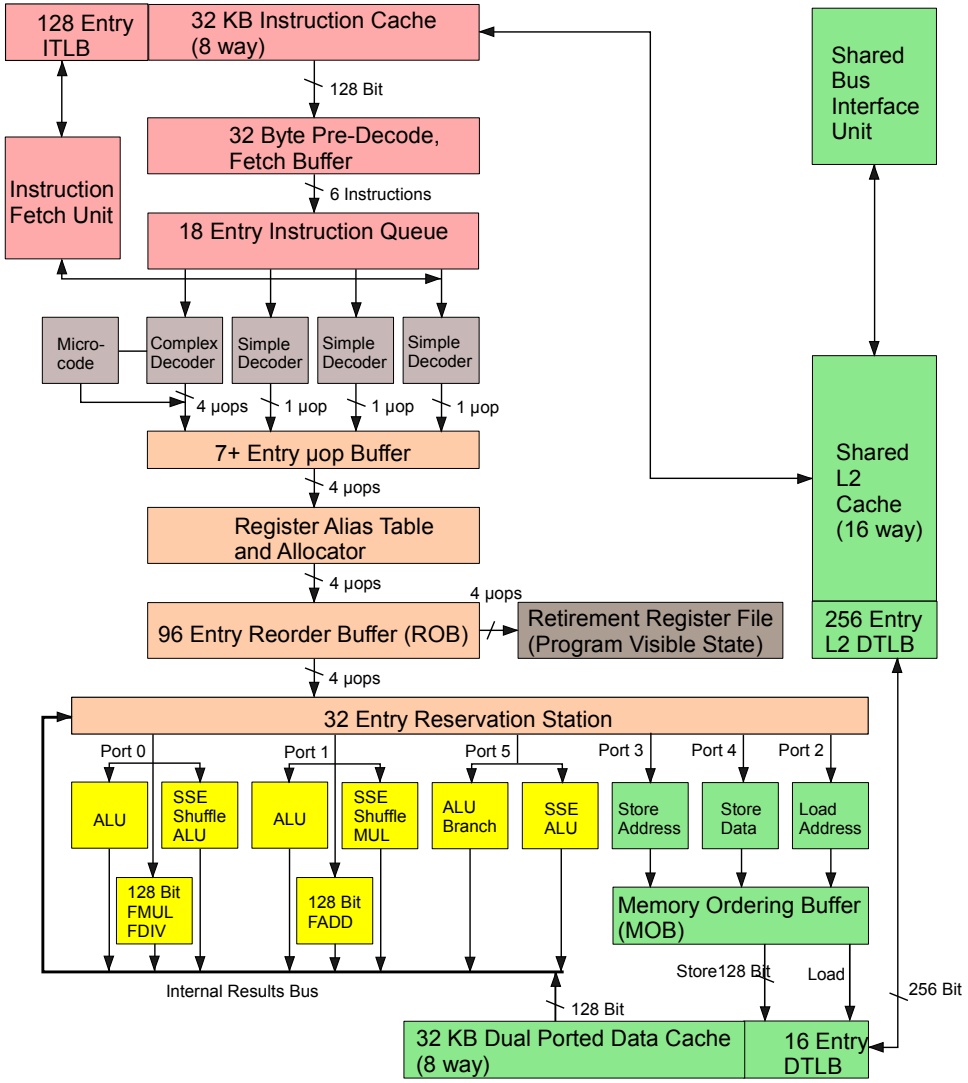
- **skládá se z polí, (jedno nebo po sobě jdoucí slova)**
- **adr. části mohou být adresami nebo odkazy na reg. (nepřímo)**



Nejpoužívanější formát je u CISC dvouadresový, u RISC tříoperandový jen mezi registry



Intel Core 2 Architecture (Mikroarchitektura) (M1)



Požadavky na dobrého programátora

Každý programátor by měl umět vytvořit takový kód, aby splnil následující požadavky:

- Efektivní využití procesoru, tj. rychlý kód
- Efektivní využití paměti, tj. používat jen takové datové typy, aby stačily k účelům programátora a zároveň nezabíraly příliš místa v paměti
- Následování stanovených pravidel konvence, tj. pravidla zápisu pro lepší čtení zdrojového kódu (mezery, odsazení..)
- Možnost jednoduché úpravy a zlepšování kódu (používání funkcí nebo OOP podle možností programátora)
- Dobře otestovaný a pevný kód, tj. zjištění a oprava možných chyb (většinou se vyskytují takové chyby, které neočekáváme, že by se mohly objevit!)
- Dokumentace (návod pro používání programu)

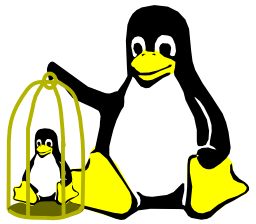
Z knihy Randall Hyde: Write Great Code - Volume 2: Thinking Low-Level, Writing High-Level, překlad k@ze

⇒ moje doporučení pro x86 a další CPU architektury

- pro x86 – přečíst a pochopit návody na optimalizaci od Agner Fog http://www.agner.org/optimize/
  - The microarchitecture of Intel, AMD and VIA CPUs
  - An optimization guide for x86 platforms
- pro pochopení vývoje CPU a technik zrychlení vykonávání instrukcí přečíst
  - John Bayko: Great Microprocessors of the Past and Present
- pochopit jak vyšší jazyky a běhová prostředí mapují jazykové konstrukce na datové typy a alokaci paměti a z toho odvodit, které datové struktury použít
- prostudovat vlastnosti kompilátoru a zkusit si různé testy a získat „cit“ pro odhad, jak budou které konstrukce výhodné a kolik paměti pak data zaberou
- často pak zjistit, že v mnoha případech bez dostatečných znalostí neobratné snahy optimalizovat vedou k pomalejšímu kódu, než když je ponechaná kompilátoru volnost ⇒ když je opravdu výkon prioritou, aplikace náročná, tak znovu studovat a pochopit, kde jsem prvně měl znalosti nedostatečné, kde je dobré věřit kompilátoru a kde se mu musím pomoc
- veškeré toto snažení může být zcela zbytečné, pokud je nevhodně navržený/vybraný již vlastní algoritmus nebo koncepce řešení
- viz uvedený příklad skládání dvou byte do 16-bit slova

Virtualizace

- Virtualizace skrývá implementaci/vlastnosti nižších vrstev (reality) a předkládá prostředí s požadovanými vlastnostmi
- V počítačové technice rozdělujeme virtualizaci na
  - Čistě aplikační/na úrovni jazyků a kompilovaného kódu (byte-kód) – virtuální stroje, např. JVM, dotNET
  - Emulace a simulace typicky jiné počítačové architektury (také zvaná křížová virtualizace)
  - Nativní virtualizace – izolované prostředí poskytující shodný typ architektury pro nemodifikovaný OS
  - Virtualizace s plnou podporou přímo v HW
  - Částečná virtualizace – typicky jen adresní prostory
  - Paravirtualizace – systém musí být pro běh v nabízeném prostředí upraven
  - Virtualizace na úrovni OS – pouze oddělená uživ. Prostředí



Virtualizace na úrovni OS

Mechanism	Operating system	License	Release date	Features						
				FS	Disk quotas	I/O QoS	Mem limits	CPU quotas	Network isolation	Part. checkpt. and live migration
chroot	most UNIX-like operating systems	Proprietary BSD GNU GPL CDDL	1982	Yes	No	No	No	No	No	No
FreeVPS	Linux	GNU GPL	-	Yes	Yes	No	Yes	Yes	Yes	No
Linux-VServer (security context)	Linux	GNU GPL v.2	-	Yes	Yes	Yes/ No [1]	Yes	Yes	Yes	No
OpenVZ (virtualization, isolation and resource management)	Linux	GNU GPL v.2	-	Yes	Yes	Yes [2]	Yes	Yes	Yes[3]	Yes
Parallels Virtuozzo Containers	Linux, Windows	Proprietary	-	Yes	Yes	Yes [4]	Yes	Yes	Yes[3]	Yes
Container/Zone	Solaris	CDDL	01/2005	Yes	Yes	No	Yes	Yes	Yes[3]	No[5]
FreeBSD Jail	FreeBSD	BSD	03/2000	Yes	No	No	No	No	Yes	No
sysjail	OpenBSD, NetBSD	BSD	-	Yes	No	No	No	No	Yes	No
WPARs	AIX	Proprietary	10/2007	Yes	-	-	-	-	-	

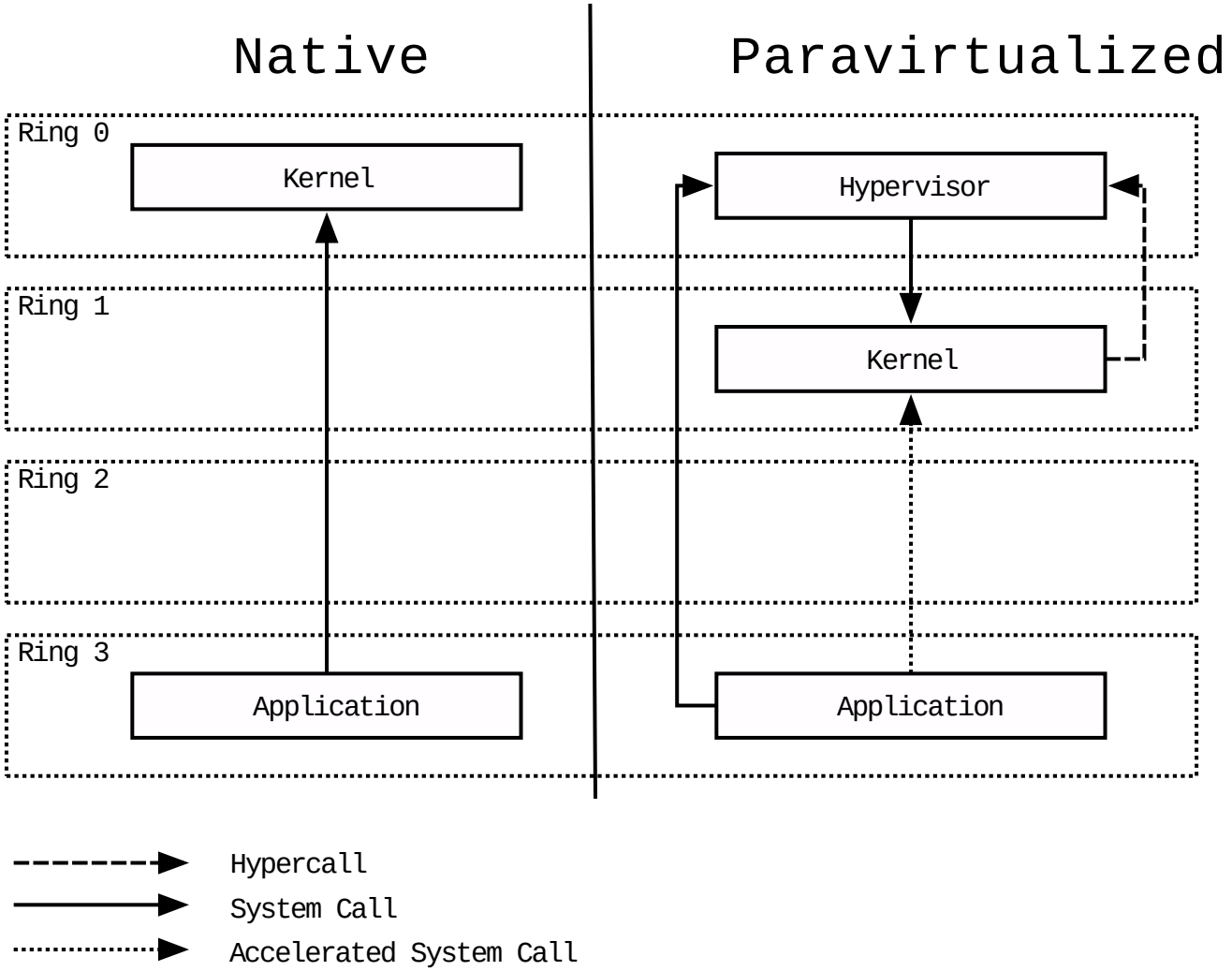
Virtualizace na úrovni celého stroje

- Hostitelský počítač (Host, Domain DOM 0)
- Hostovaný systém (Guest)
- Procesor pro hostovaný systém
  - pro nativní případ běžný kód/neprivilegované instrukce zpracovány přímo CPU
  - pro křížový případ interpretace instrukcí emulátorem (program v DOM 0), případně akcelerace
- Privilegované instrukce v hostovaném systému
  - způsobí výjimky, které obslouží monitor/hypervizor tak, že je odemuluje
  - CPU má podporu pro HW virtualizaci (AMD-V, Intel VT-x), stínové stránkovací tabulky
- Periferie/zařízení
  - přístupu na IO a paměťově mapované periferie způsobují výjimky a hypervizor odsimuluje jejich činnost
  - hostovaný systém se přizpůsobí tak, aby předal požadavky přímo ve formátu, kterému hypervizor rozumí (ovladače na míru atd.)

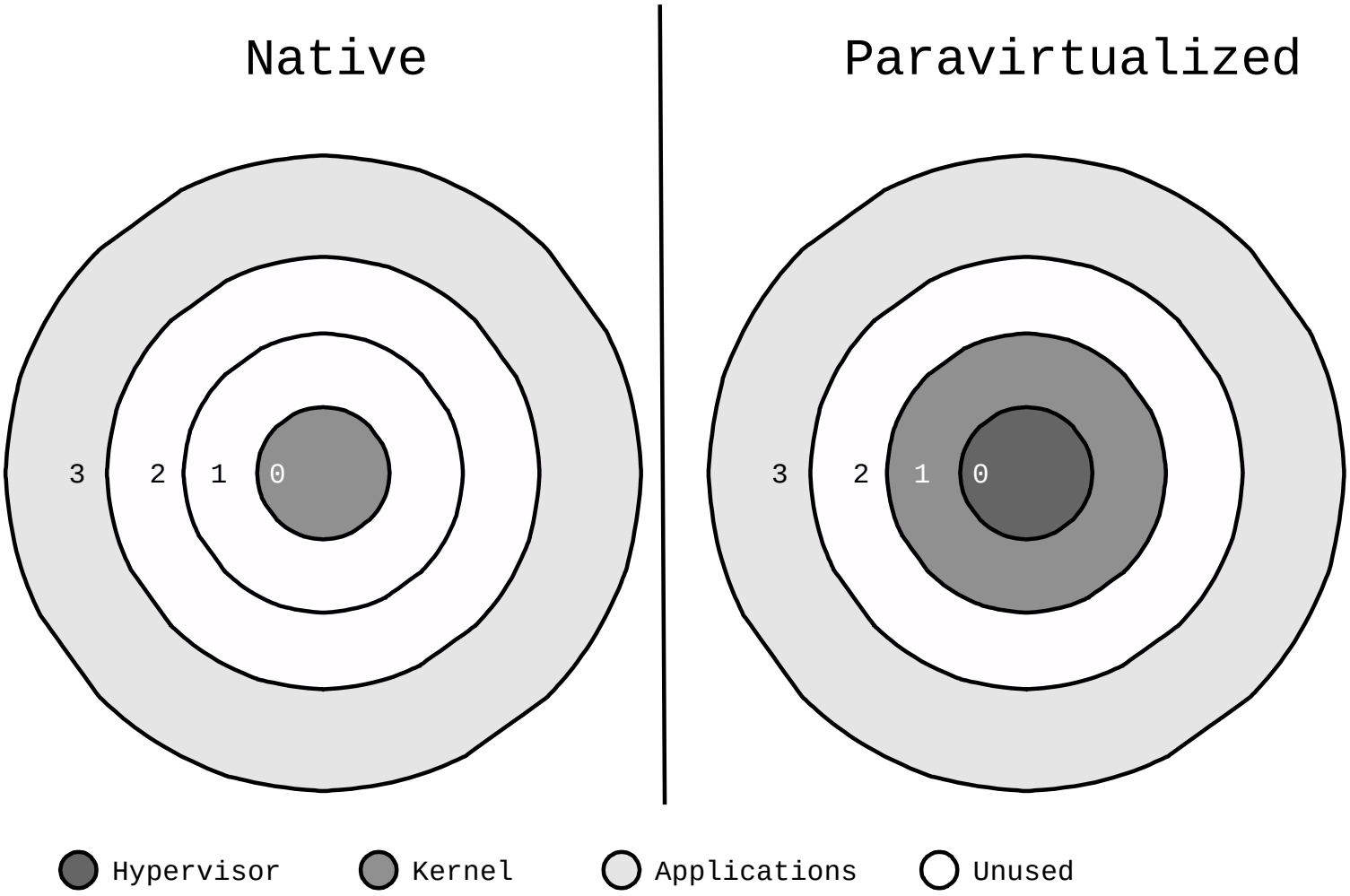
Hypervizor

- zajišťuje spouštění a zastavování domén
- monitoruje jejich činnost a ošetřuje výjimky – především emuluje činnost privilegovaných instrukcí
- zajišťuje rozdělení paměti a výpočetního výkonu do hostovaných systémů
- emuluje činnost periferií a předává data do ovladačů fyzických zařízení a sítí na úrovni hostitelského systému
- může být implementovaný
  - v uživatelském prostoru hostitelského systému (QEMU)
  - s využitím podpory v HW a jádře OS (KVM)
  - jako samostatný systém/mikrojádro, který využívá systém v jedné doméně (DOM 0) pro komunikaci s fyzickými zařízeními a z tohoto systému data přeposílá do ostatních (XEN)

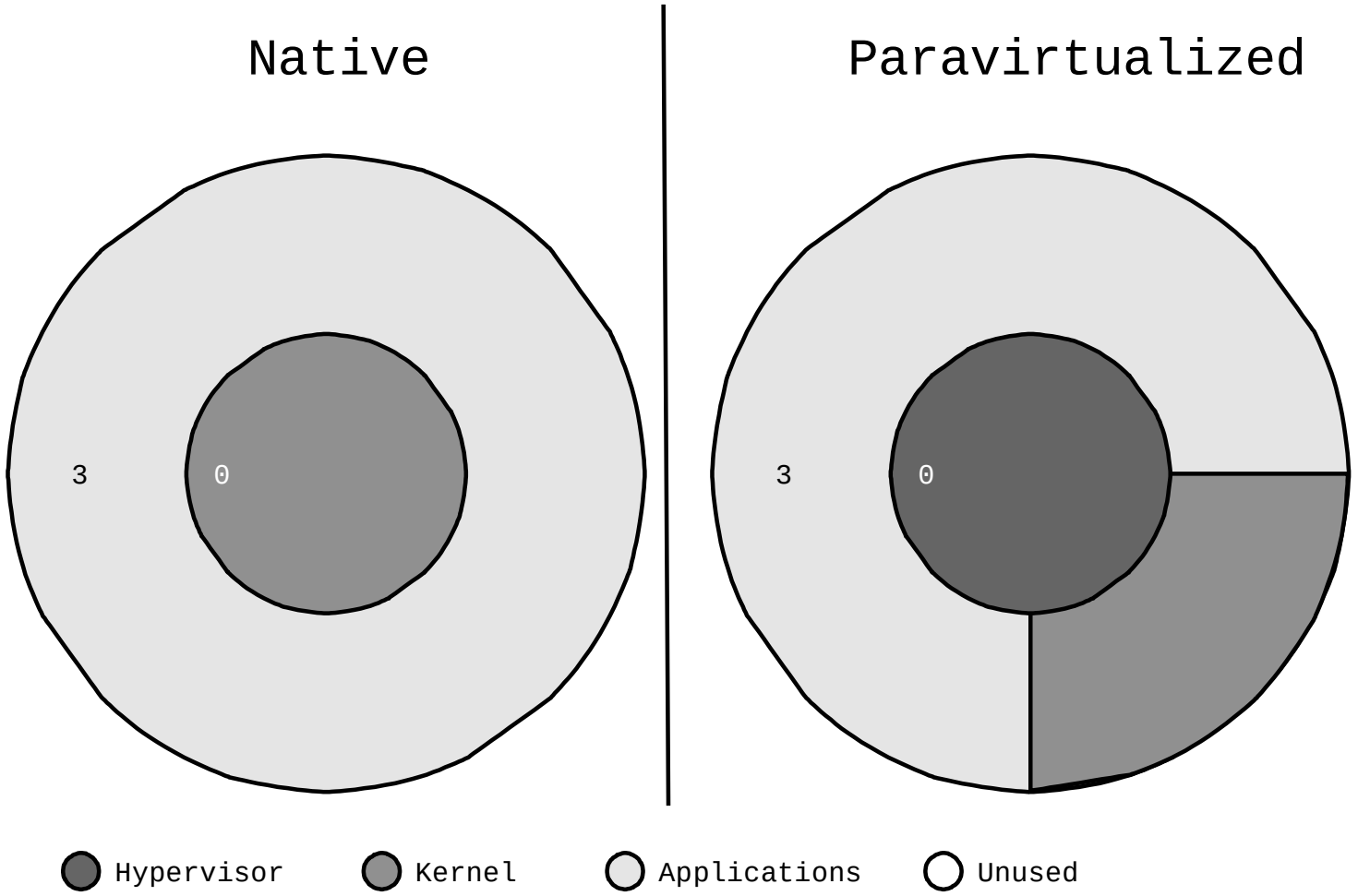
Paravirtualizovaná systémová volání (XEN)



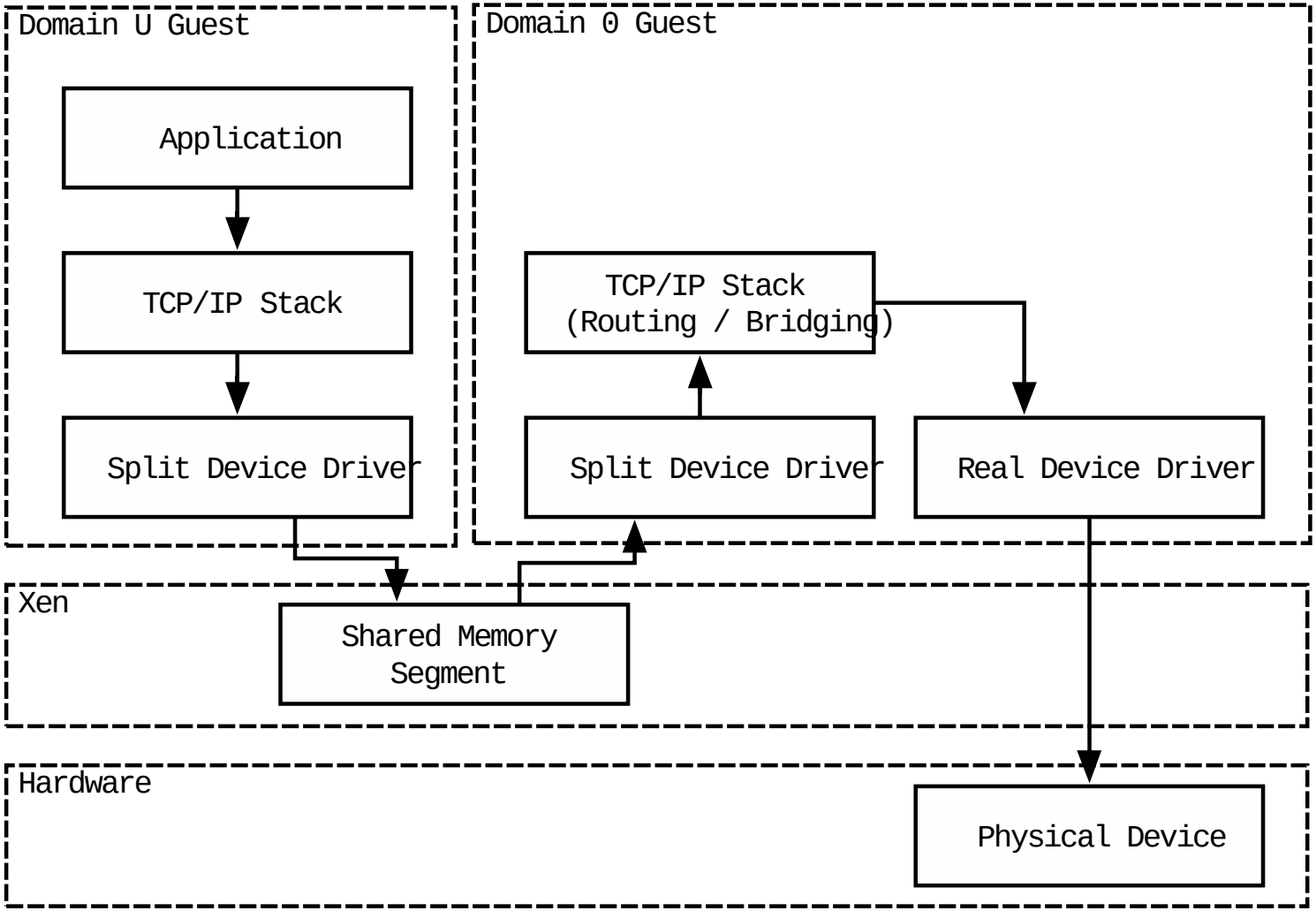
Využití úrovní (ringů) oprávnění X86 v Parvirtualizovaném OS



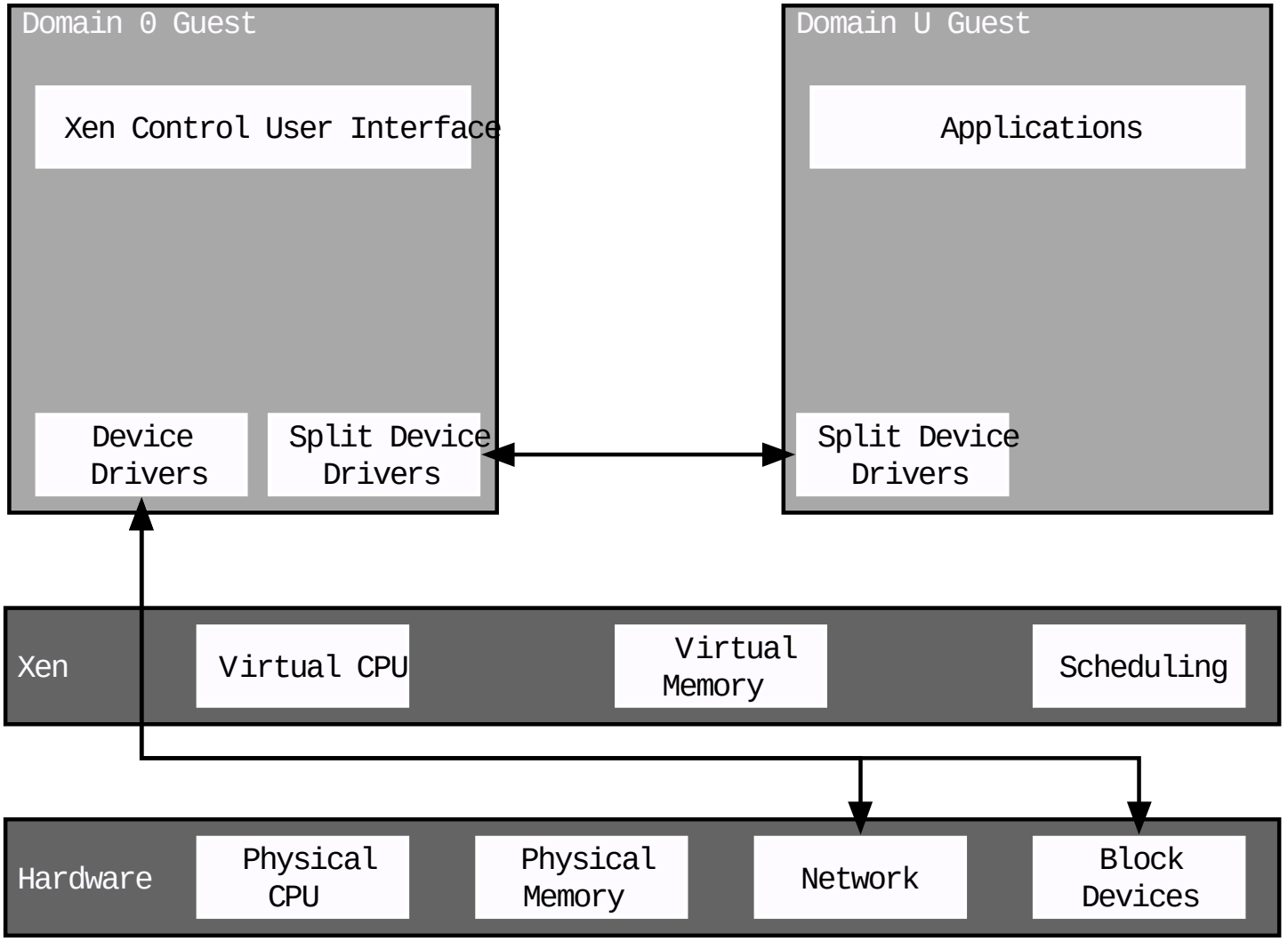
Využití úrovní (ringů) oprávnění pro AMD64/EMT64



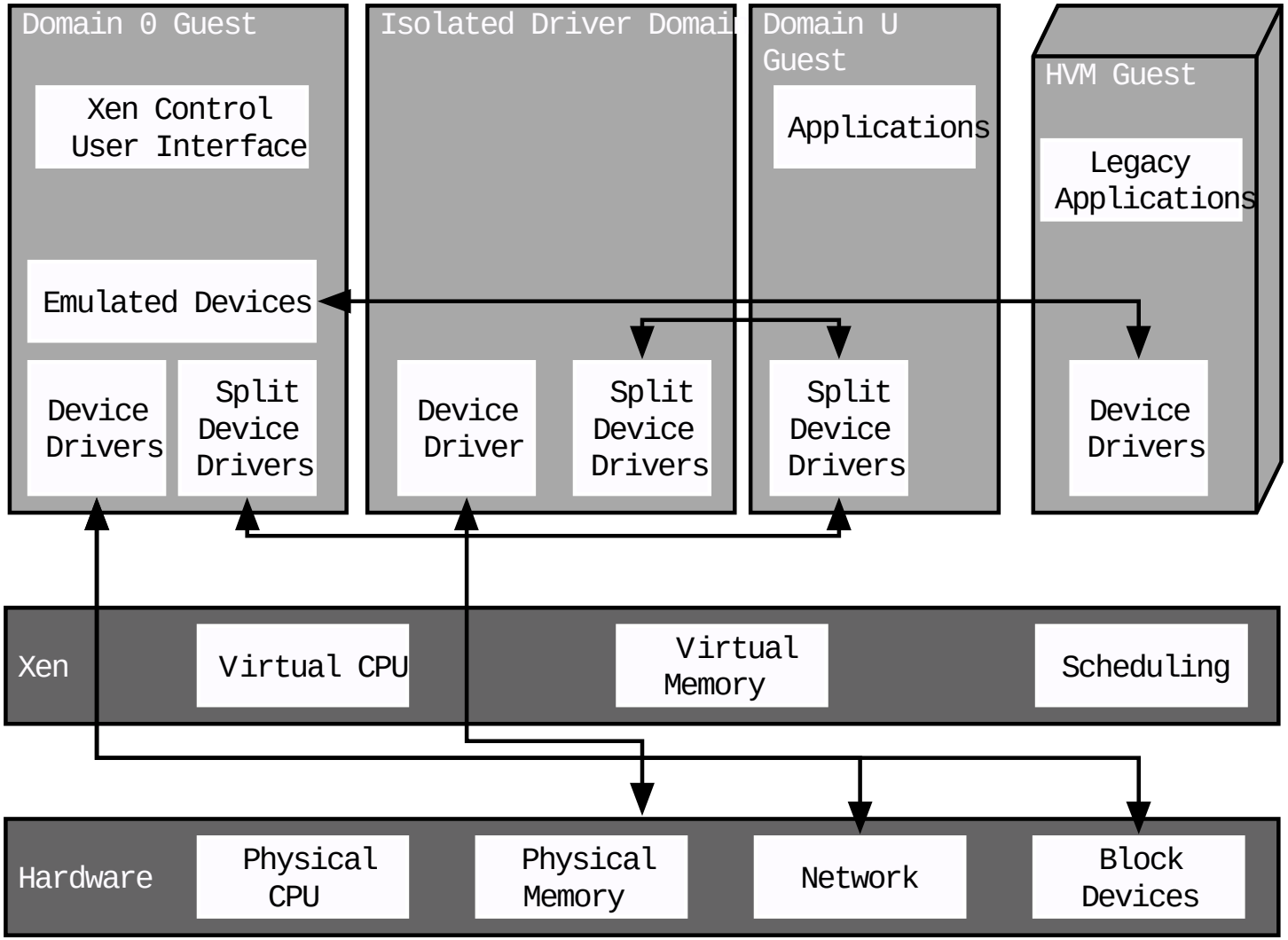
Packet Path from Unprivileged Domain



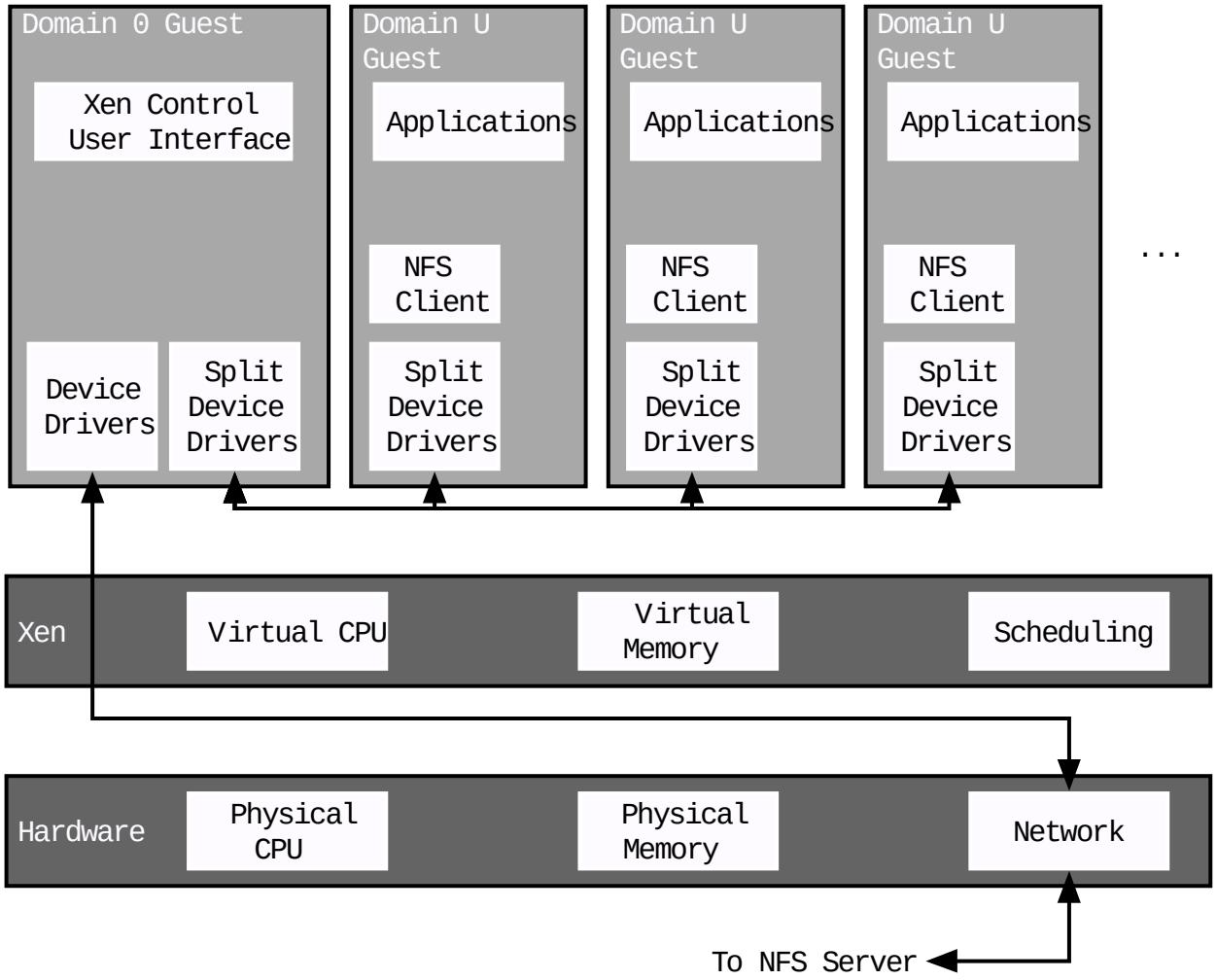
Architektura zařízení v prostředí Xen



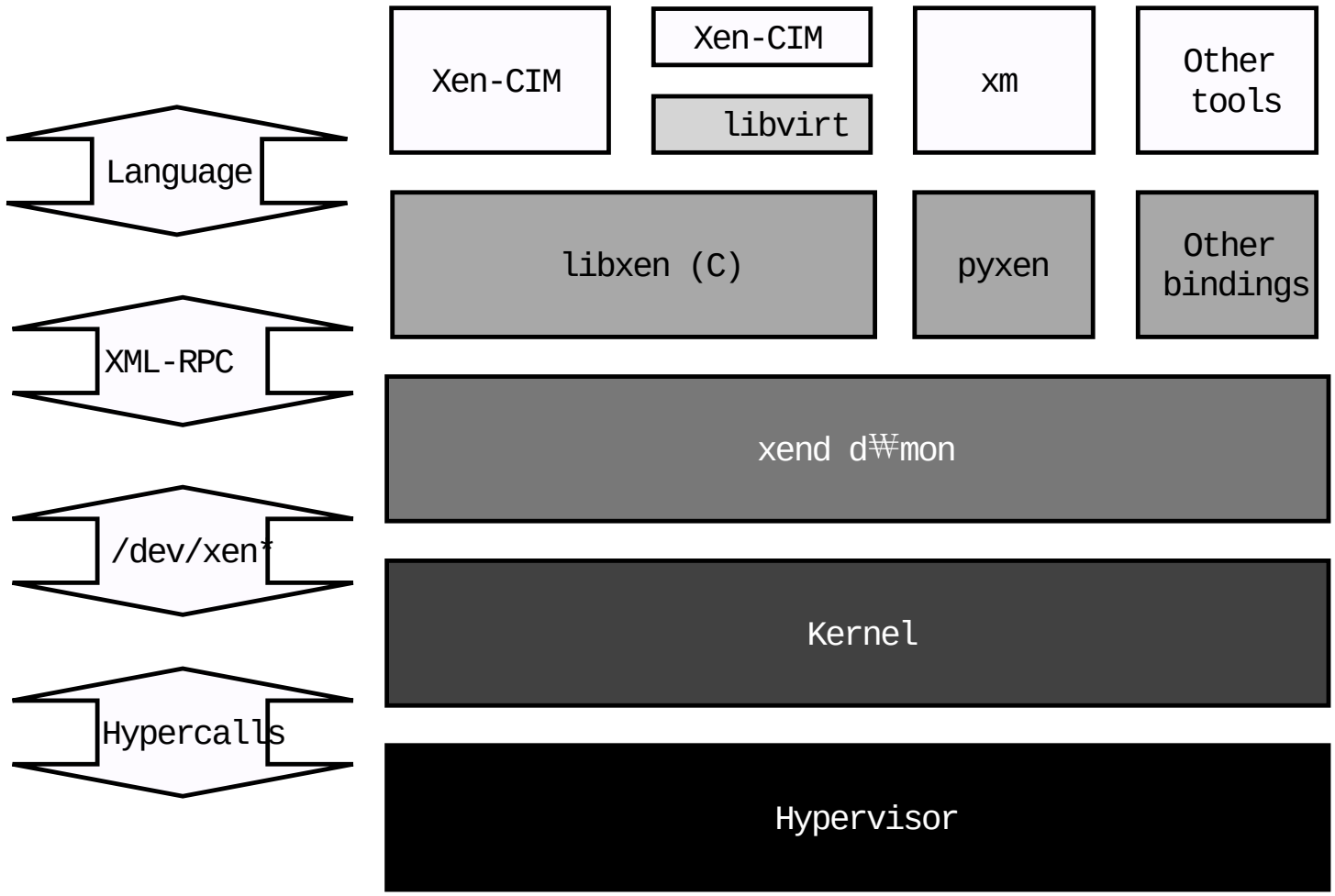
Xen – Nemodifikovaný Guest OS (HVM)



Xen – Plně paravirtualizovaný/adaptovaný Guest OS



Xen API Hierarchy



Qemu, GNU/Linux a další

- dobrý zdrojem informací pro zájemce o portaci GNU/Linuxu, psaní ovladačů a přenositelných aplikací jsou výukové texty ze serveru **Free Electrons** http://free-electrons.com/docs/
  - Například virtualizace
    - Thomas Petazzoni / Michael Opdenacker: Virtualization in Linux