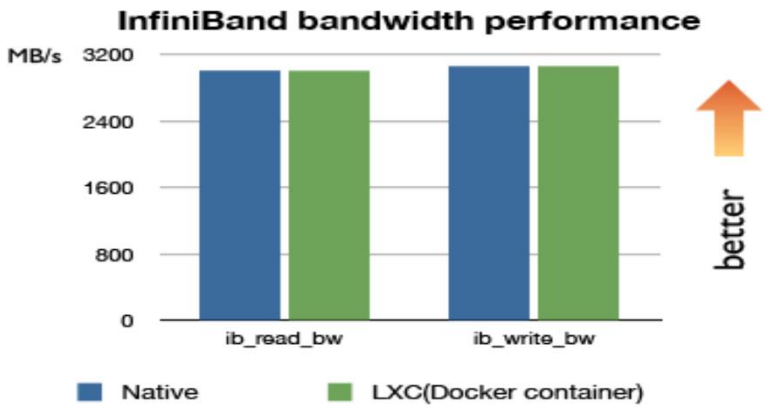
Hálózatba kapcsolt erőforrás platformok és alkalmazásaik

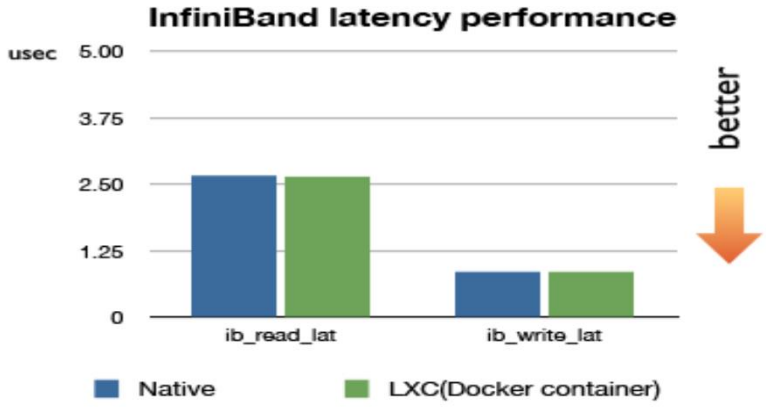
Simon Csaba
TMIT, HSNLab
2019

Konténerek

Virtualizáció és teljesítmény?

- Motiváció:
  - Virtualizáció = valamiben(fut_valami)
    - 2x kell elvégezni egyes feladatokat
  - Teljesítmény növelés: a „valamiben” overhead csökkentése





Konténer metafora: áruszállítás probléma

- Logisztikai (menedzsment) kérdés
  - Sok szállítási platform, sok árutípus
  - Egy közös csomagoló-egység

	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
						

Konténer metafora: inter-modális konténer

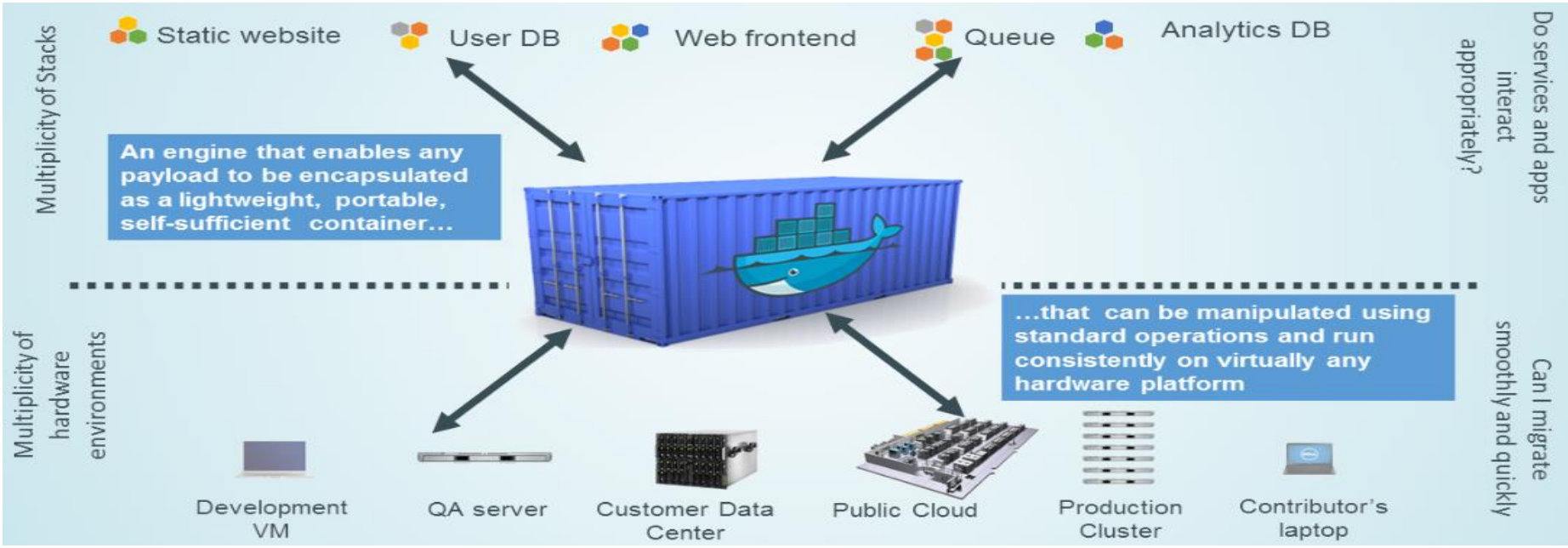
- Logisztikai (menedzsment) kérdés
  - Sok szállítási platform, sok árutípus
  - Egy közös csomagoló-egység
  - KONTÉNER (egységes, köztes szállítási egység)

Kódok „szállítása” virtualizációs megoldásokhoz

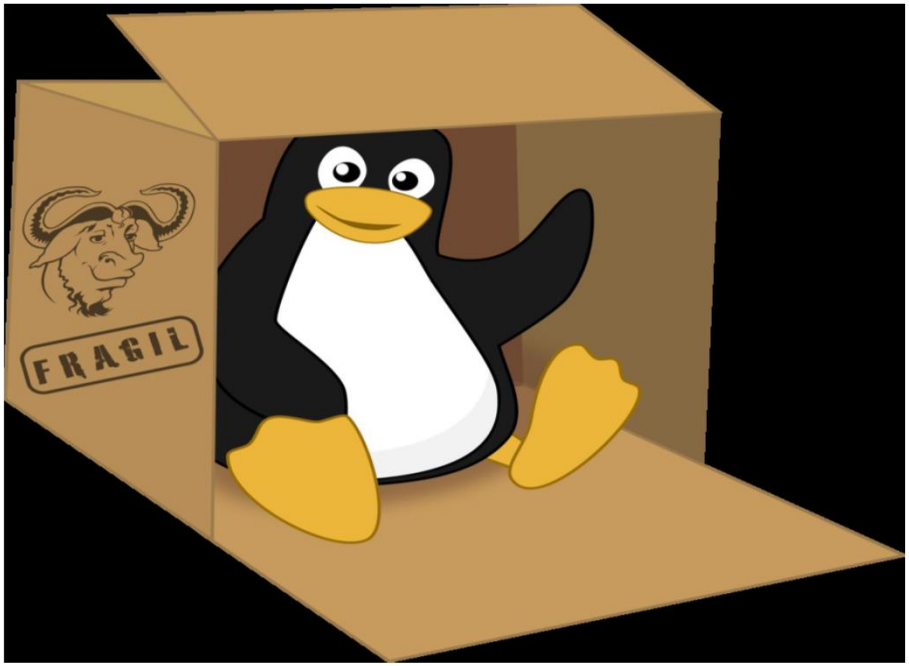
- Szállítási platform => végrehajtási környezet
- Árutípus => számítási feladat

	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
						

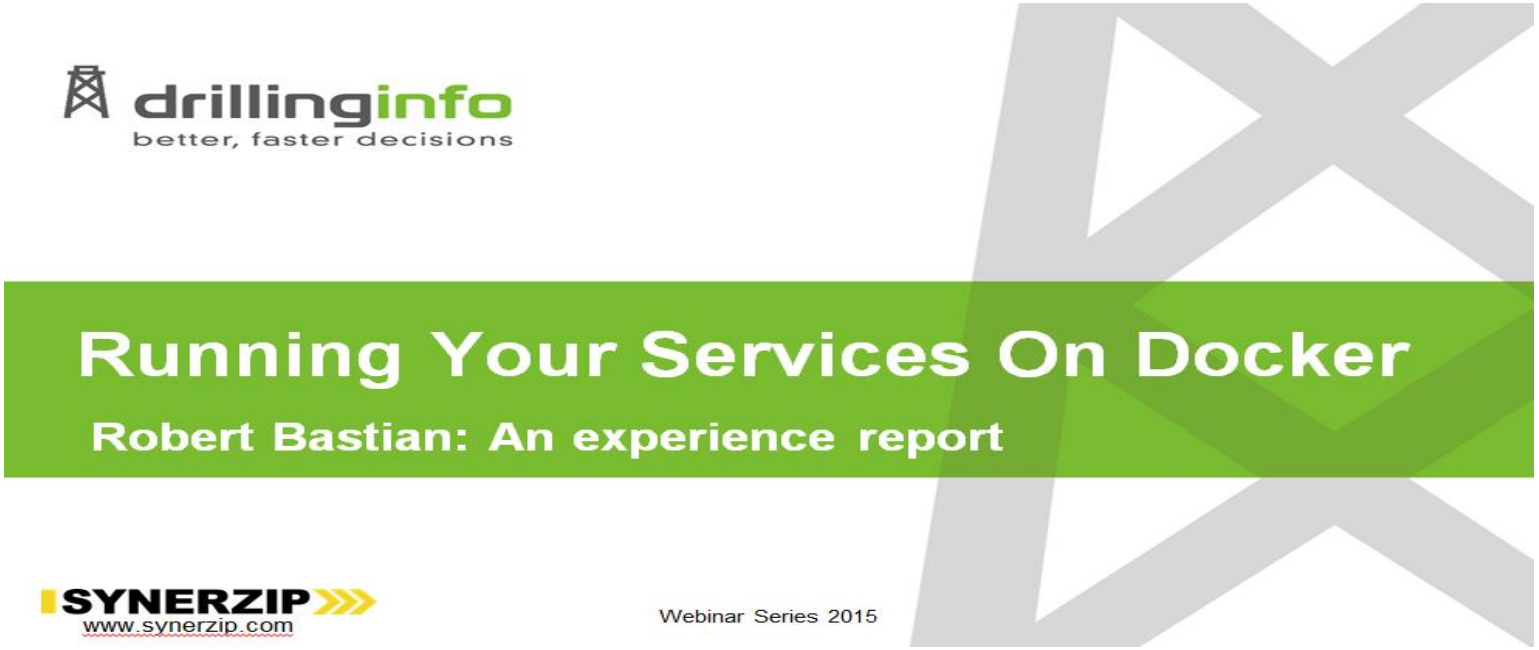
Alkalmazás-konténerek



A Linux konténerek mindent megoldanak (khm)



Zárójel: esettanulmány (miért is választják a Dockert a felhő helyett?)



Why Docker?

My World Needed To Change

- 5+ individual teams building “micro services” in Java and Scala
- Frictionless deployment of “micro-services” using Chef & AWS
- 25+ separate “micro-services” deployed in the previous 18 months
- Each service is typically deployed to a single AWS virtual machine
- Each service is deployed 6x - dev, test, staging (2x) and production (2x)
- 25+ “micro-services” became nearly 150 AWS virtual machines

Why Docker? COST!

The AWS bill is too damn high!

- Decline in the global price of oil causing churn in our business
- 6 AWS virtual machines per service isn't sustainable with our budget
- AWS monthly bill started to gain visibility from sr. management and ***the board***

Why Docker? WASTE!

We weren't using the compute and memory resources purchased from AMZN!

- Nearly all "micro-services" were at 1% CPU utilization
- Nearly all "micro-services' were only using 40% of memory (JVM)
- 150+ virtual machines essentially sitting idle

Why Docker? LOCK IN!

How would we leave AMZN if we wanted to?

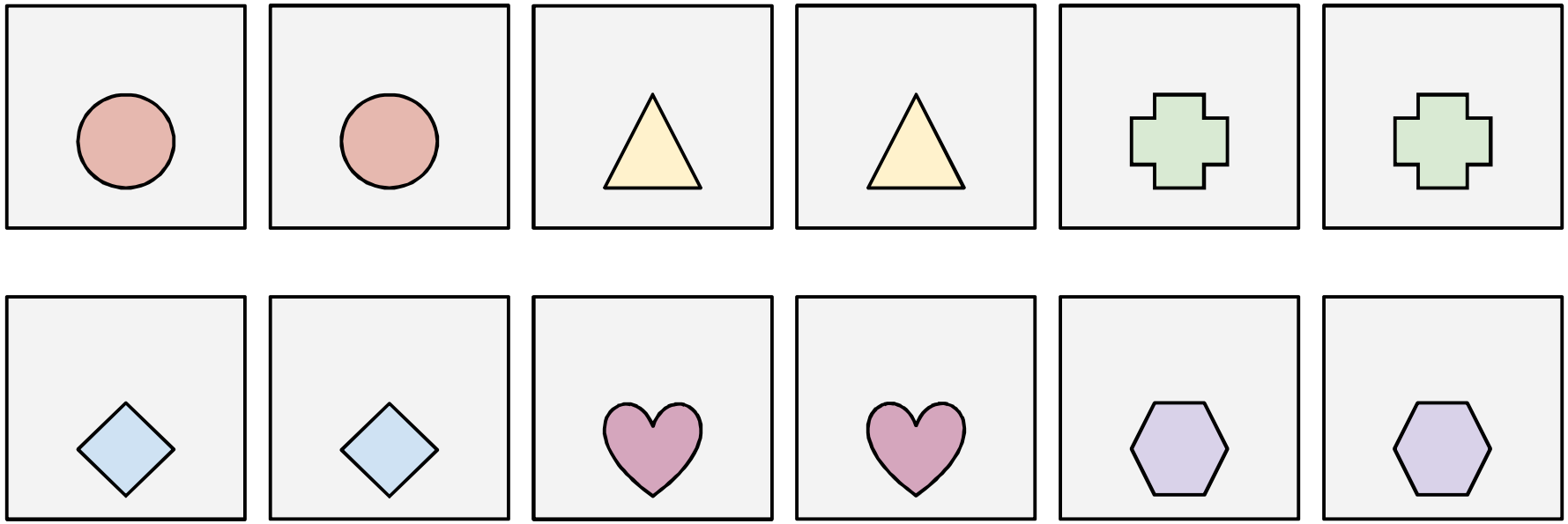
- Could we use Drillinginfo IT's Openstack platform?
- What about alternate IaaS providers like Rackspace or Azure?
- What about Container as a Service (CaaS) providers like Joyent, Tutum or Profitbricks?
- What about using Amazon's Container Service?

My World Needs To Change - Problem Statement

"How can we ***deploy fewer*** virtual machines while ***increasing the density and utilization*** of services per machine ***without locking*** us into a specific IaaS provider?"

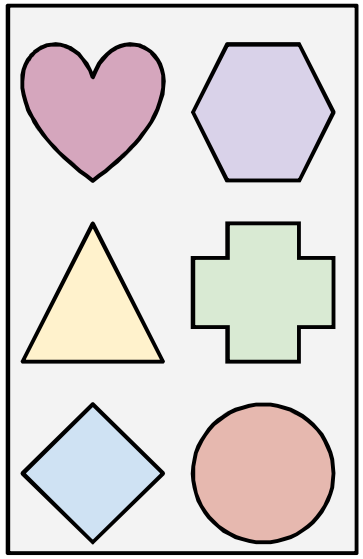
Why Docker Is Important - Before Containers

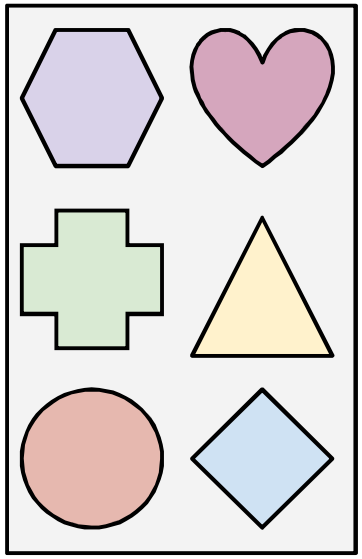
Very inefficient use of memory and CPU resources



Why Docker Is Important - After Containers

Isolated services in fewer VMs...



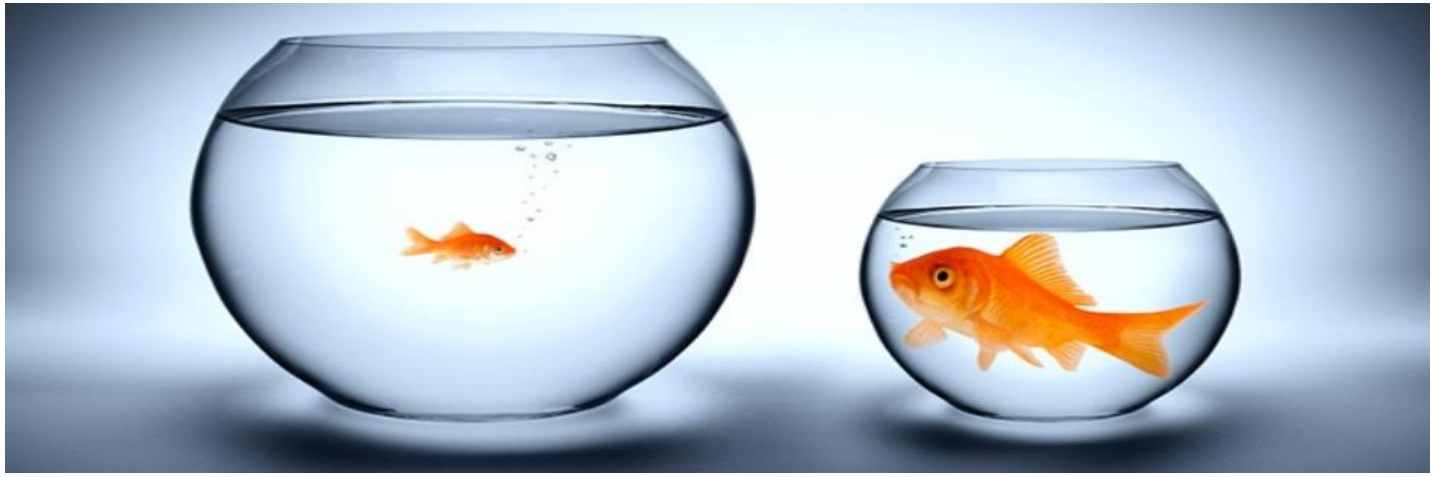


... and use VMs more efficiently.

Why Is Docker Important?

Docker container technology provides our “micro-services” platform:

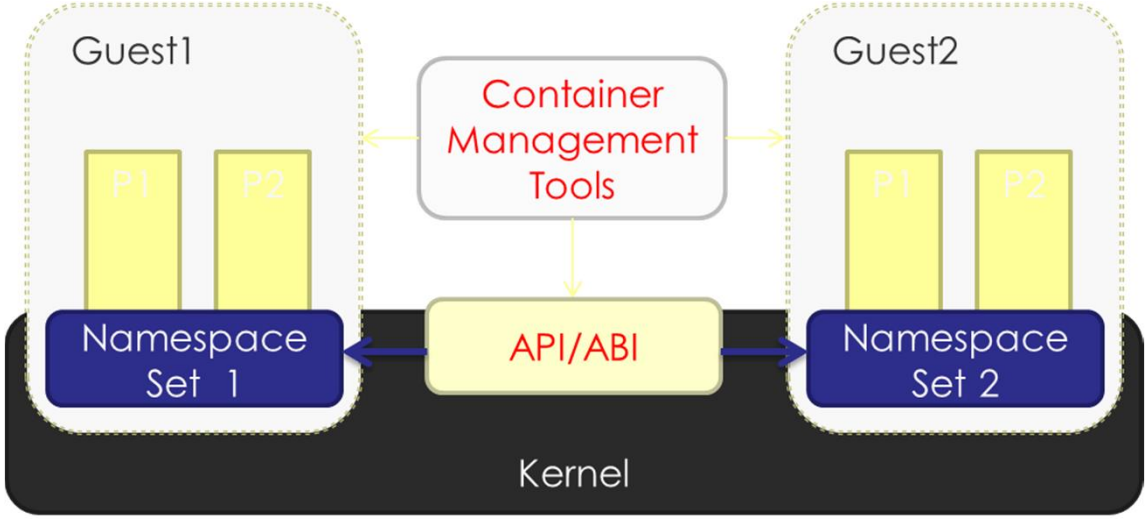
- Increased ***density*** of ***isolated*** “micro-services” per virtual machine (9:1!)
- Containerized “micro-services” are ***portable*** across machines and providers
- Containerized “micro-services” are much ***faster*** than virtual machines



Zárójel vége

Bevezető: Linux konténerek

- Konténer = Operation System Level virtualization method for Linux
  - Operációs rendszer (Linux) szintű virtualizációs megoldás



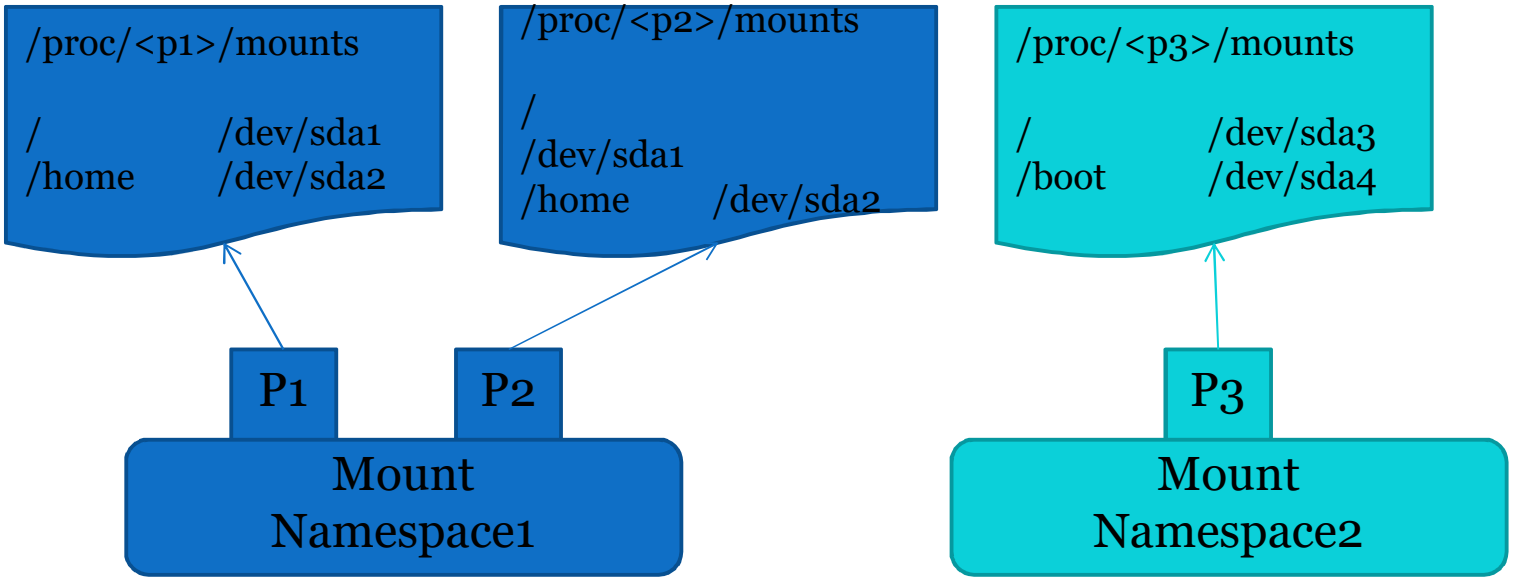
Linux Névterek

Namespaces (névterek)

- Rendszer erőforrásainak elszigetelése
- 6 névtér van a Linux Kernelben
  - Mount
  - UTS
  - IPC
  - Net
  - Pid
  - User

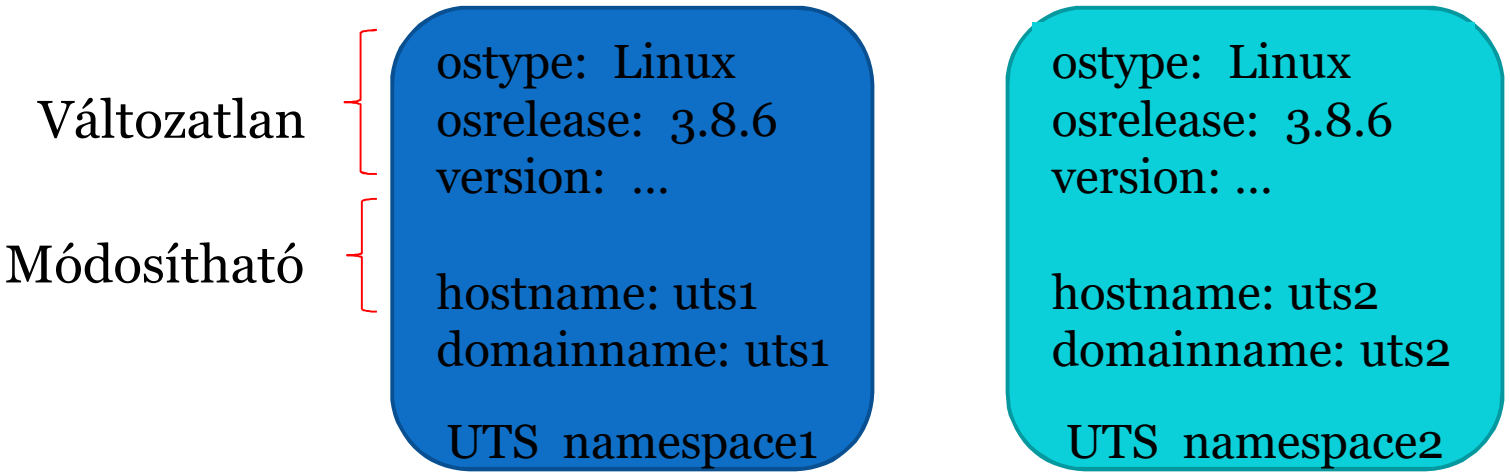
Mount Namespace

- Saját fájlrendszer



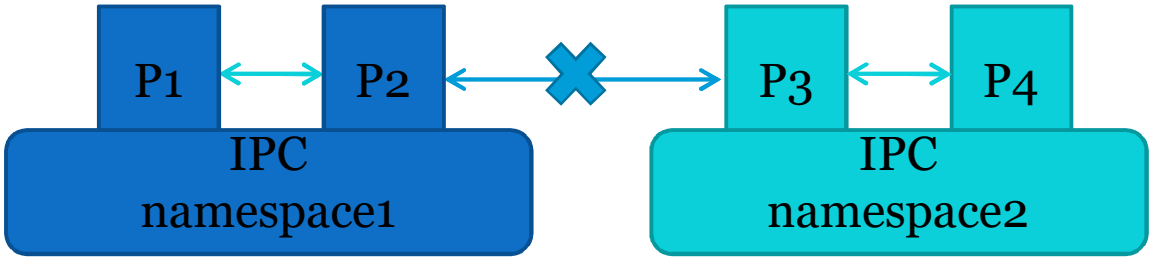
UTS Namespace

- UTS = UNIX Timesharing System
- Saját uts-infó



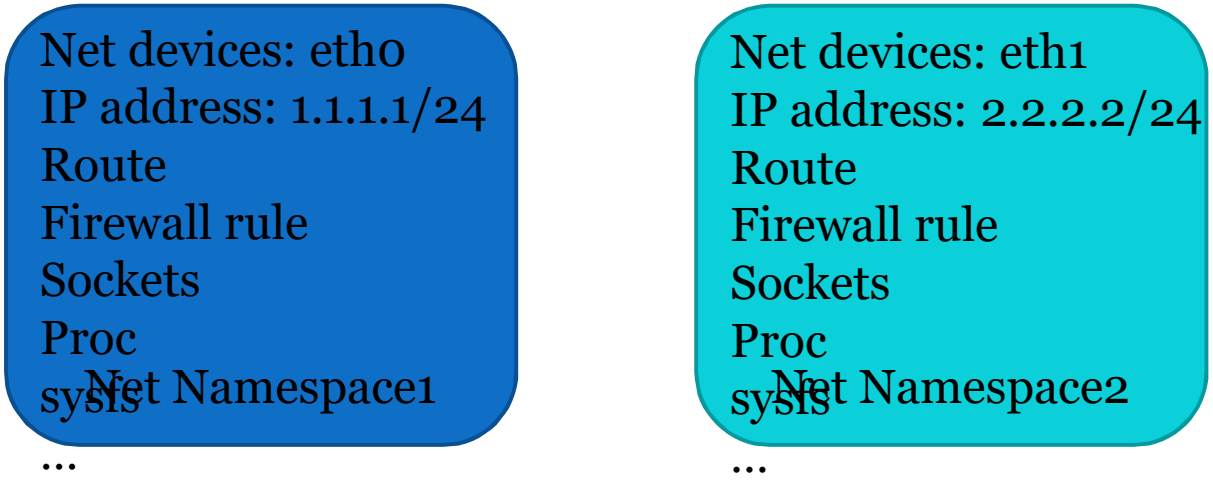
IPC Namespace

- IPC: InterProcess Communication
- Processzek közti kommunikációt izolál:
  - Shared memory
  - Semaphore
  - Message queue



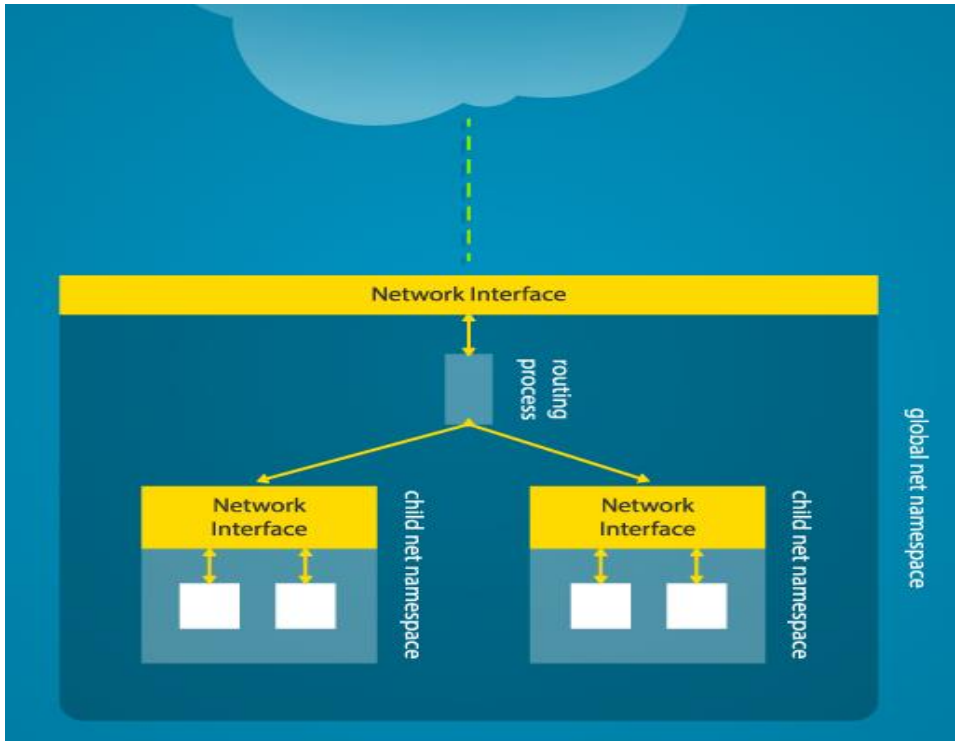
Net Namespace 1/2

- Net namespace: a hálózati erőforrásokat rejti el



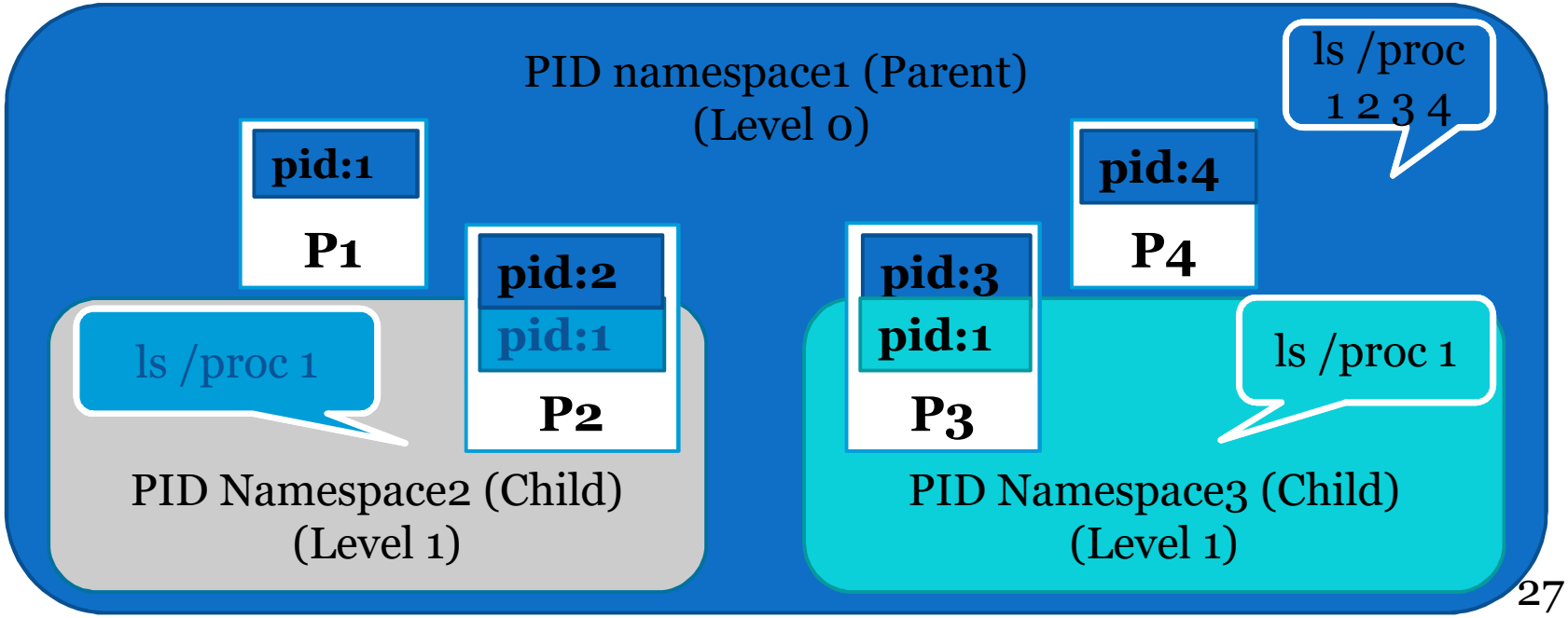
Net Namespace 2/2

- A kernel elrejti egymástól a két külön hálózati névteret
- Ha át kell hidalni a fizikai interfész és a névtér közötti rést
  - routing



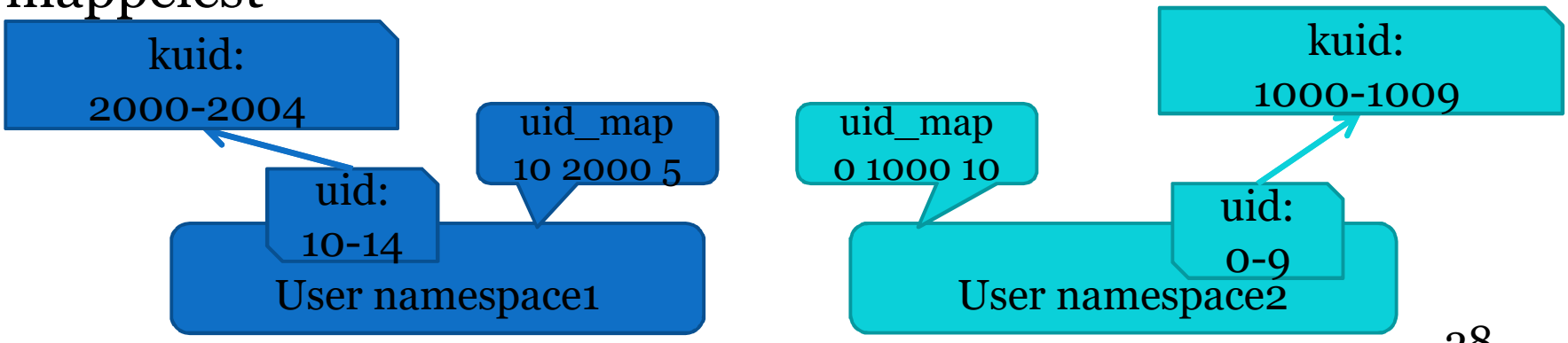
PID Namespace

- PID: Process ID
- Hierarchikus rendszerben virtualizálja



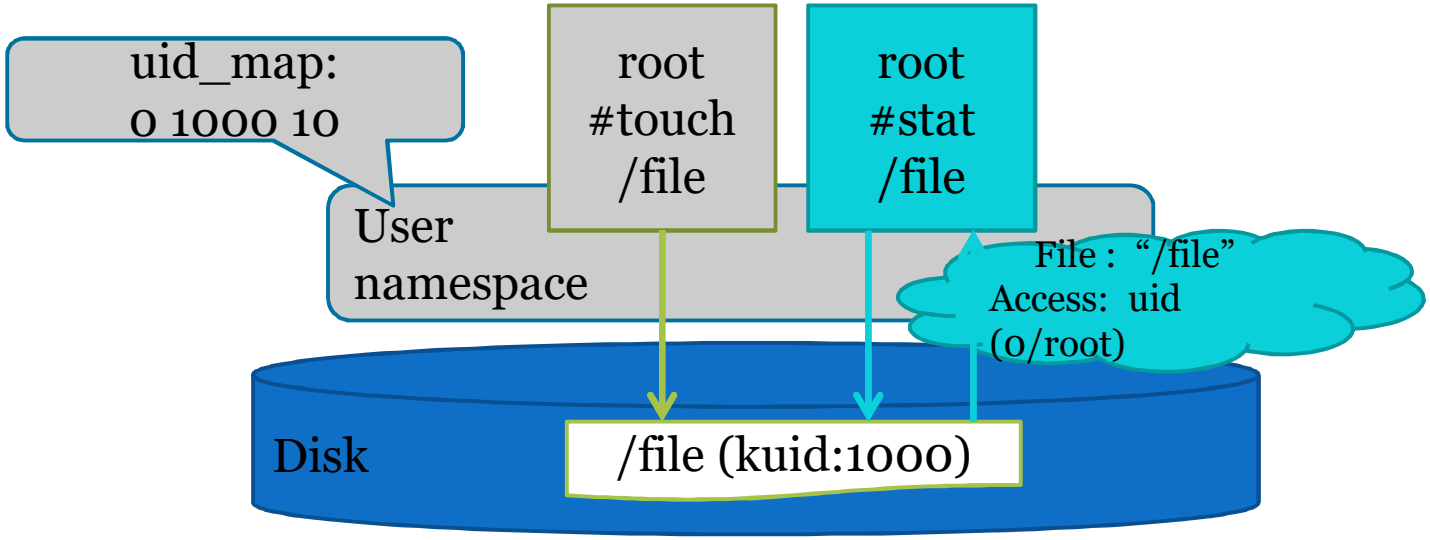
User Namespace

- Biztonsághoz köthető felhasználói attribútomok izolálása
  - kuid/kgid: Original uid/gid, Global
  - uid/gid: user id a „user” namespaceből a kuid/kgid attribútumokba lesz átfordítva
- Csak a szülő felhasználó (parent user) NS állíthat be mappelést



User Namespace

- Create, stat file



CGROUPS

Linux cgroups

- Erőforrás-felhasználás korlátozása
  - Tárolás (mem)
  - Számítás (cpu)
  - Kommunikáció (blkio)
  - Eszköz (dev)

LXC

System API/ABI

- Proc
  - /proc/<pid>/ns/
- System Call
  - clone
  - unshare
  - setns

Proc

- /proc/<pid>/ns/ipc: ipc namespace
- /proc/<pid>/ns/mnt: mount namespace
- /proc/<pid>/ns/net: net namespace
- /proc/<pid>/ns/pid: pid namespace
- /proc/<pid>/ns/uts: uts namespace
- /proc/<pid>/ns/user: user namespace

- Ha adott processznek a proc fájlja ugyanaz, akkor a két processz ugyanabban a névtérben van

Rendszerhívások

- clone

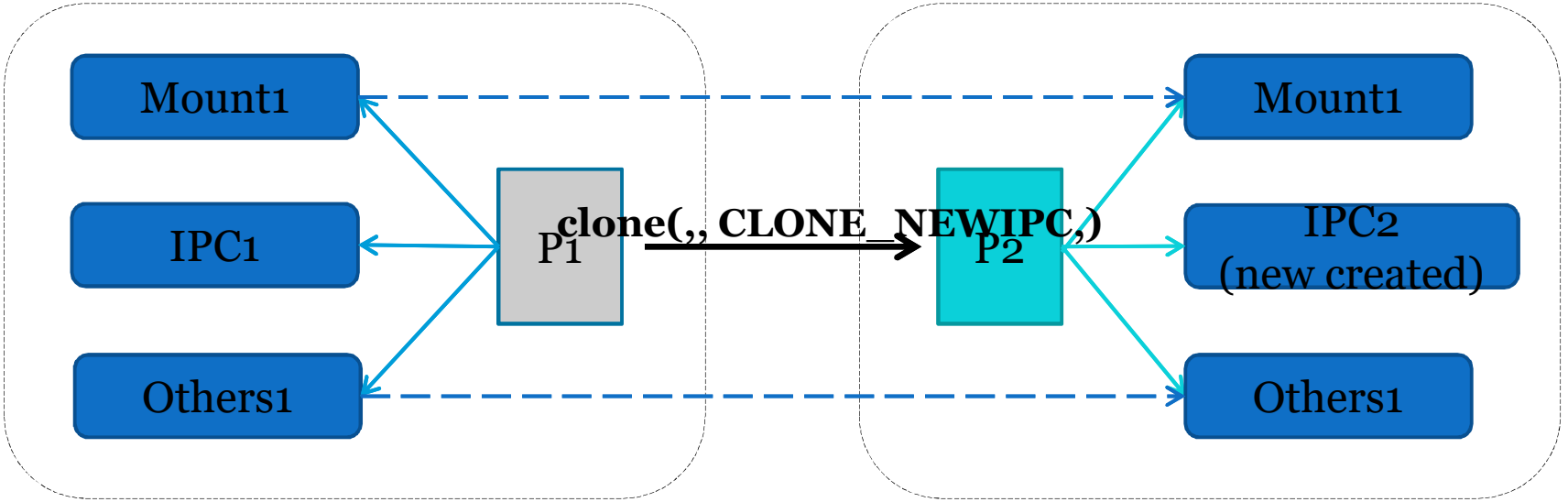
```
int clone(int (*fn)(void *), void *child_stack,
                  int flags, void *arg, ...);
```

6 flag:

CLONE_NEWIPC,CLONE_NEWNET,
CLONE_NEWNS,CLONE_NEWPID,
CLONE_NEWUTS,CLONE_NEWUSER

Rendszerhívások

- clone
  új processz (process2) és IPC a namespace2-ben



Rendszerhívások

- unshare

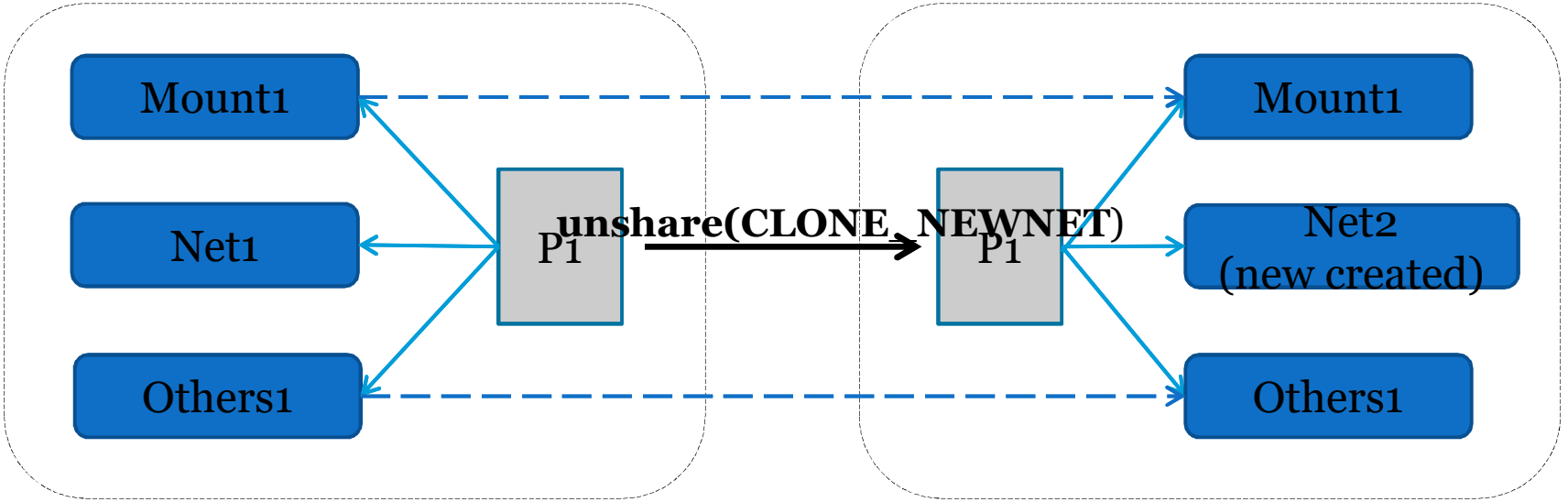
  int unshare(int flags);

  „user space”-ből új névtér hozható létre, új névtérbe lehet átlépni

Rendszerhívások

- unshare

net namespace2 létrehozása



Rendszerhívások

- setns

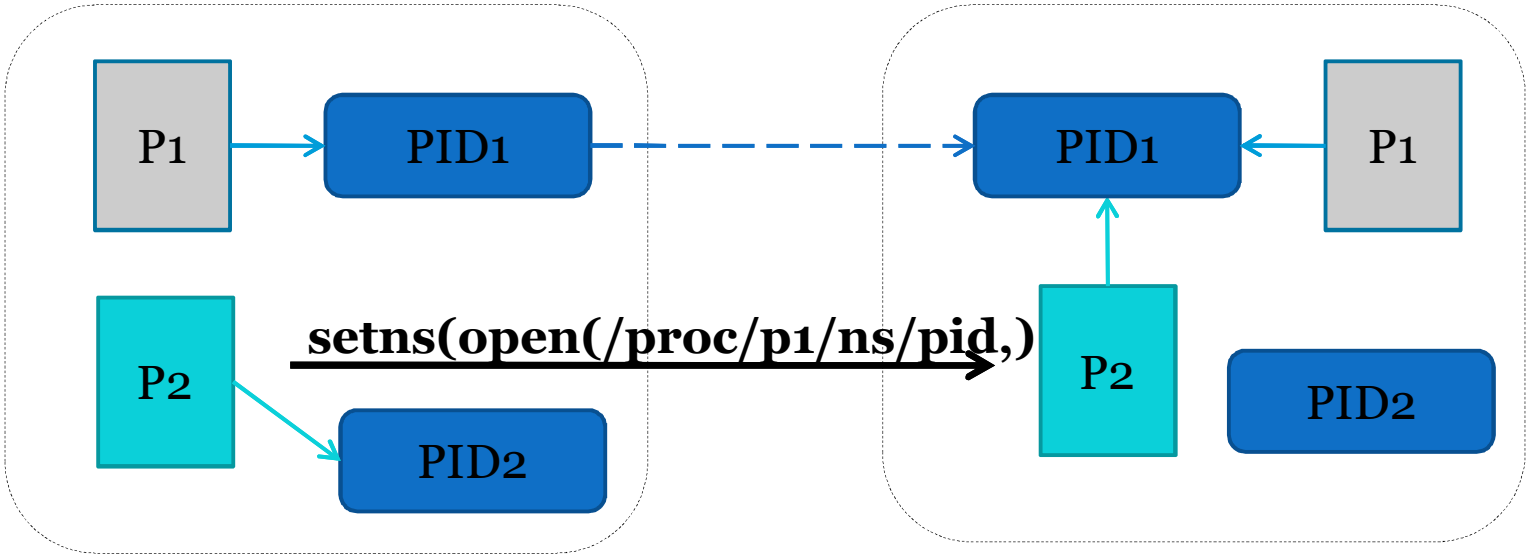
  int setns(int fd, int nstype);

  Új rendszerhívás
  Megadja, milyen névtérbe tartozzon a processz

  @fd: file descriptor of namespace(/proc/<pid>/ns/*)
  @nstype: type of namespace.

Rendszerhívások

- setns
  A P2 PID namespace megválzoztatása



Libvirt LXC

- Libvirt LXC: userspace container management tool
  - libvirt driver-ként megvalósítva
  - Konténer menedzsment
  - Névtér létrehozás
  - Privát fájlrendszer kezelése a konténeren belül
  - Konténer eszközeinek létrehozása
  - Cgroup által vezérelt erőforrások

Összefoglalás

- A konténer alapú virtualizáció előnyös tulajdonságai
- Rugalmas és hatékony erőforrás kihasználás
- Kényelmesebb a szoftver disztribúciók kezelése
- Felhasznált megoldások: Linux kernel névterek és cgroups