Felhő alapú hálózatok
Konténer alapú virtualizáció

Dr. Simon Csaba

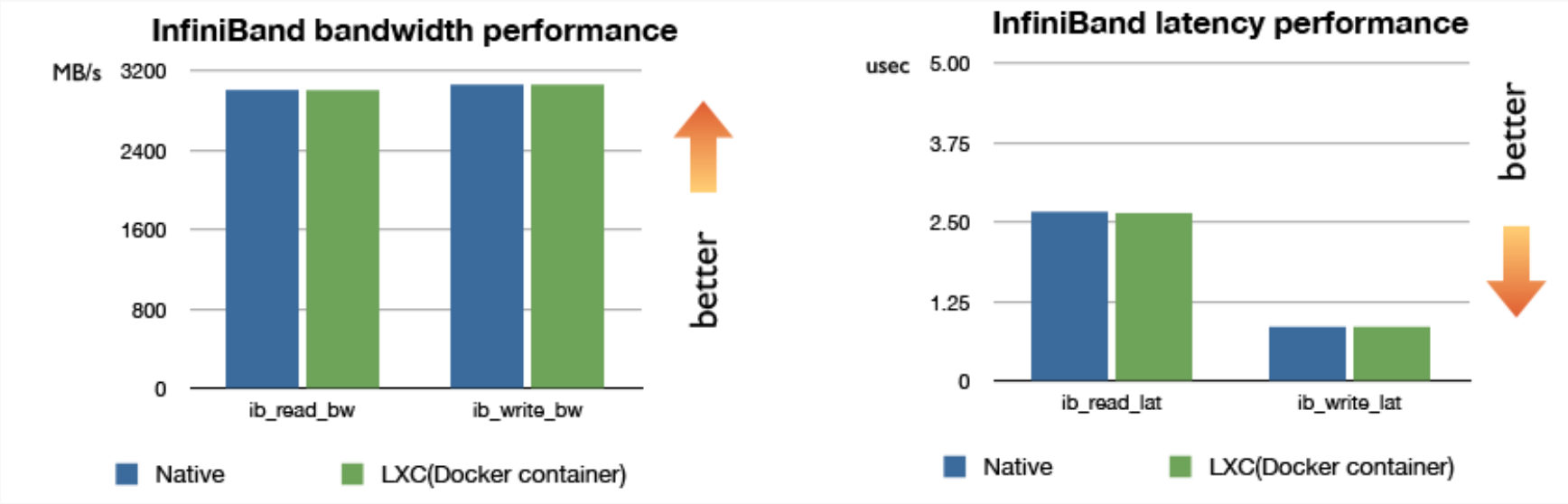
2016. 05. 03.

Budapesti Műszaki és Gazdaságtudományi Egyetem

KONTÉNEREK

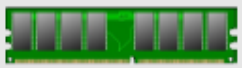
Virtualizáció és teljesítmény?

- 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

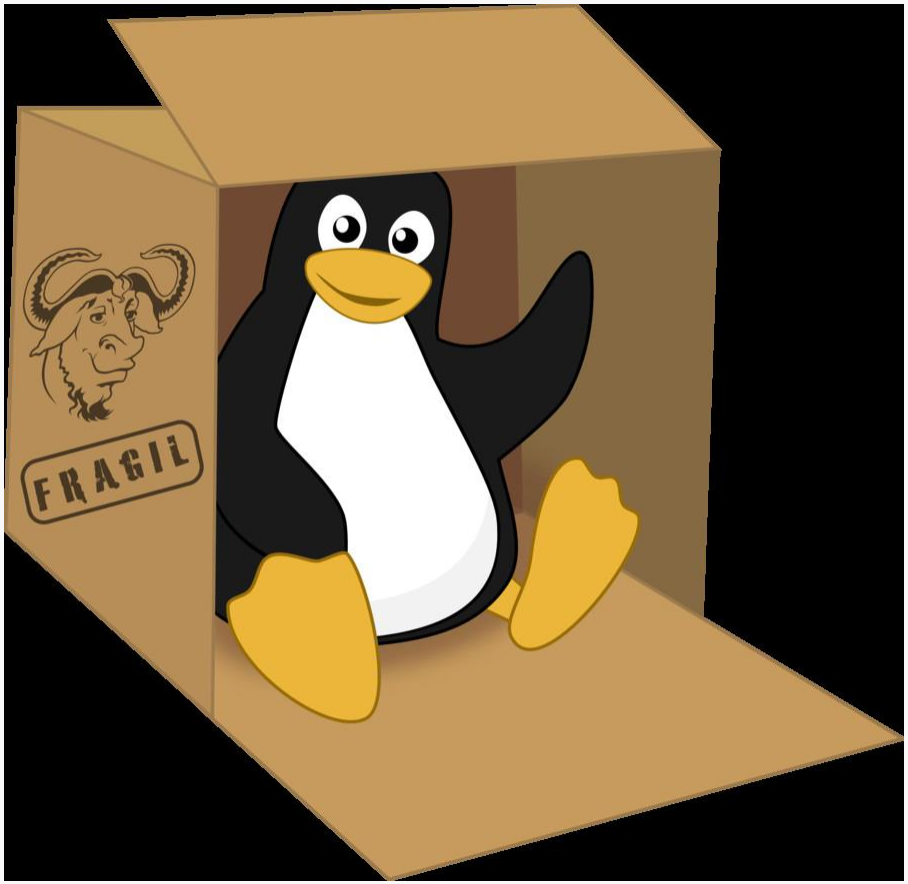
- Logisztikai (menedzsment) kérdés
  - Sok szállítási platform, sok árutípus
  - Egy közös csomagoló-egység
  - KONTÉNER (egységes, köztes szállítási egység)

Kódok „szállítása” virtualizációs megoldásokhoz

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

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

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?

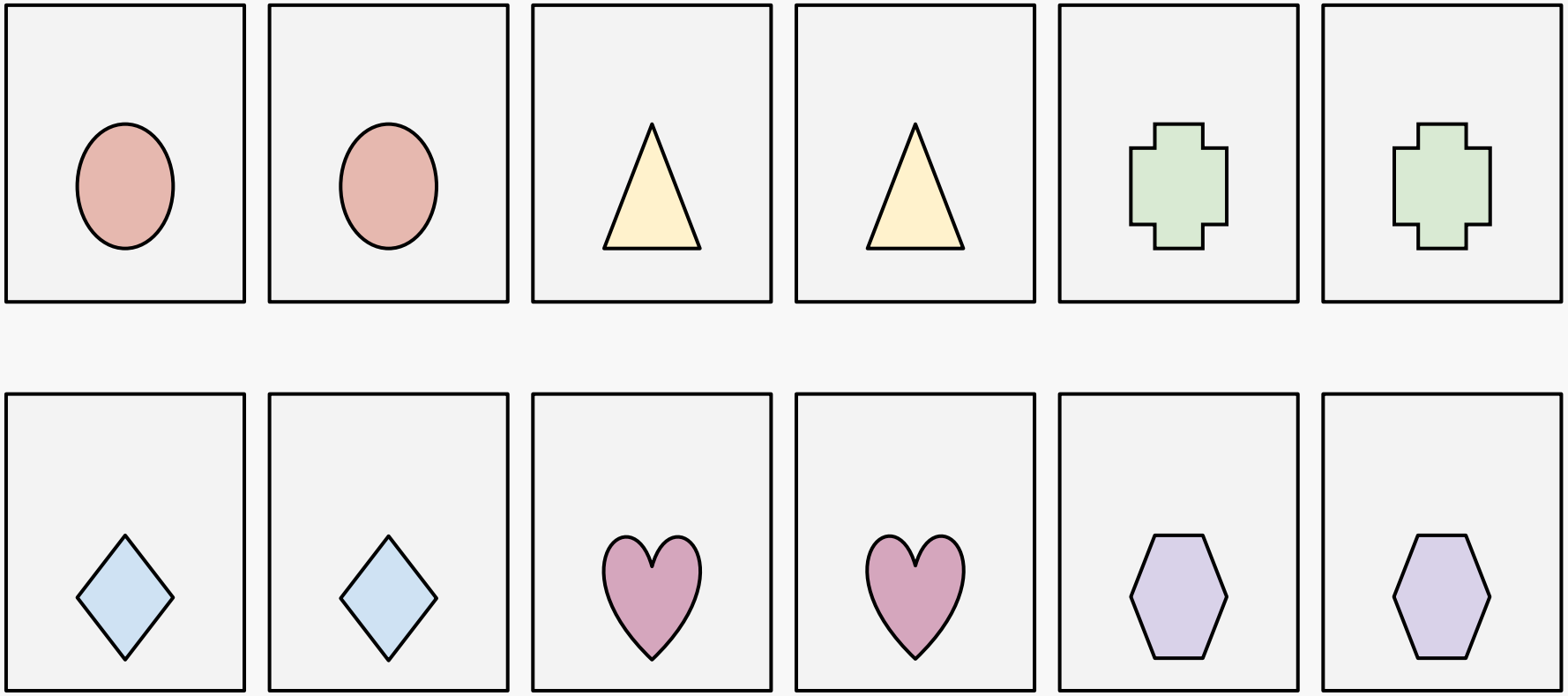
- Could we use Drillinginfo IT's Openstack platform?
- What about alternate IaaS providers like Rackspace or Azure?
- What about Container as a Service (CaaS) providers like Joyent, Tutum or Profitbricks?
- What about using Amazon's Container Service?

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?"

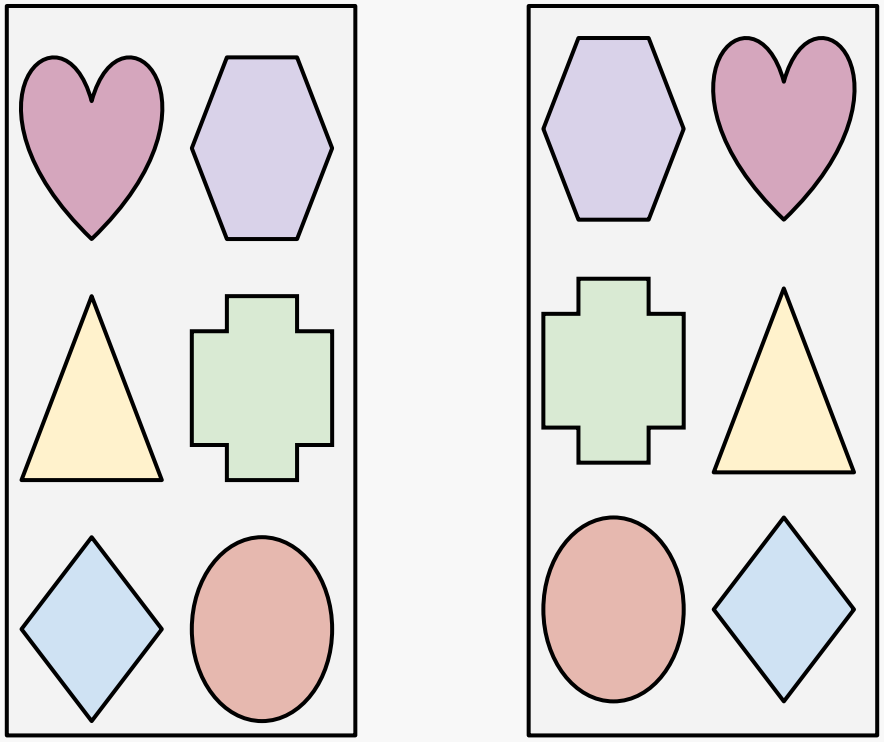
Why Docker Is Important - Before Containers

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?

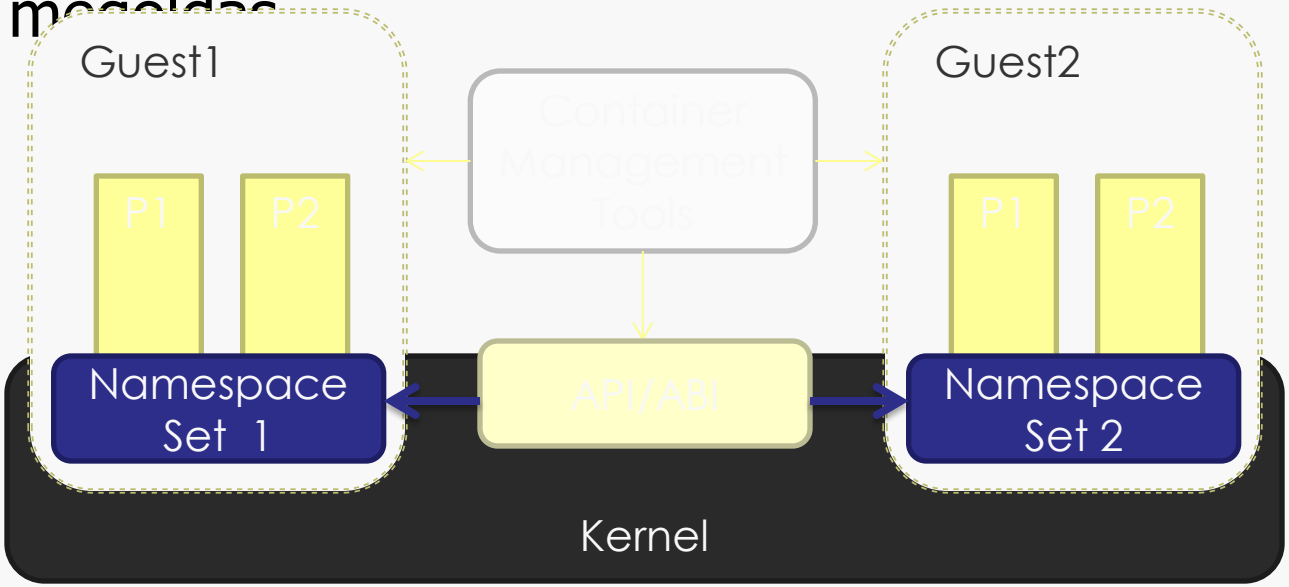
Docker container technology provides our "micro-services" platform:

- 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

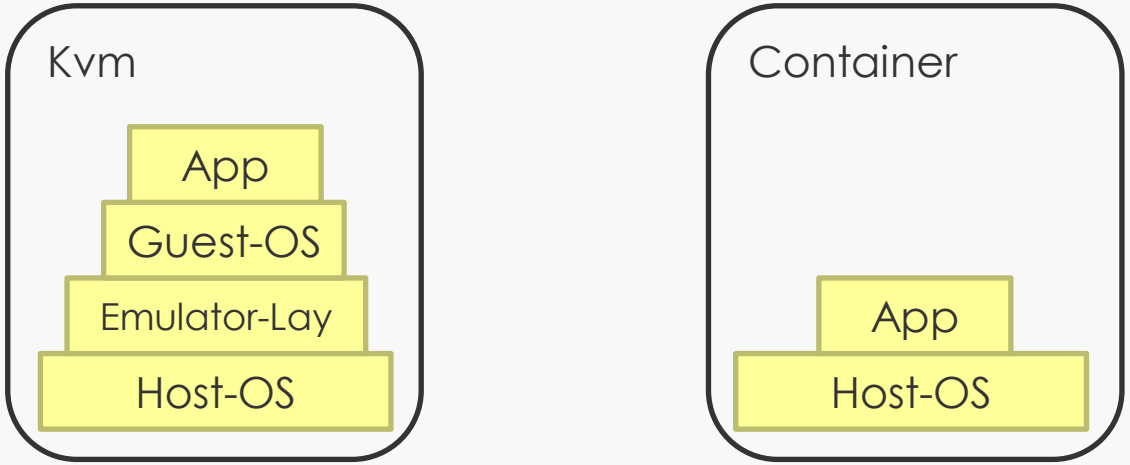
Bevezető: Linux konténerek

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



Bevezető: motiváció

- Miért van szükség rá?
  - Jobb teljesítmény



- Több-bérlős virtualizációs kötnyezet
  - multi-tenant

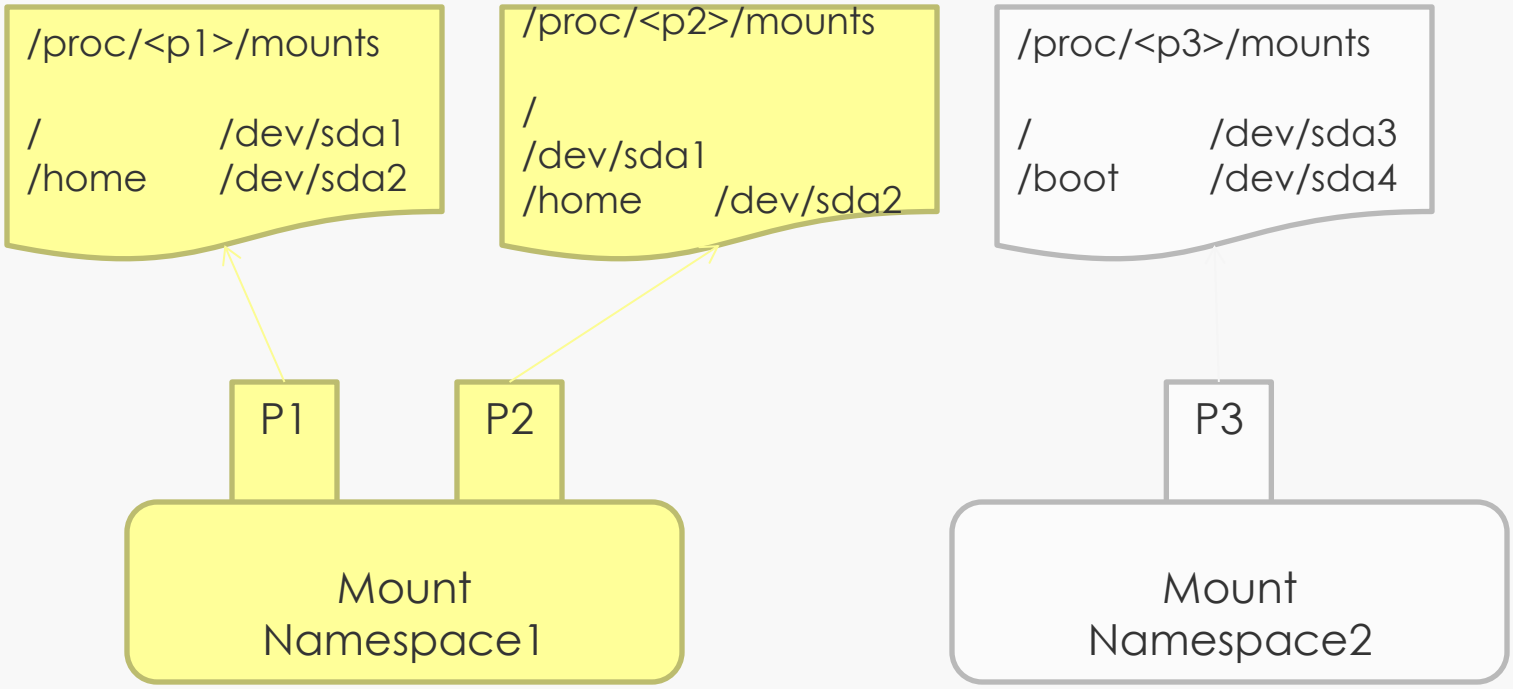
LINUX NÉVTÉR

Namespaces (névterek)

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

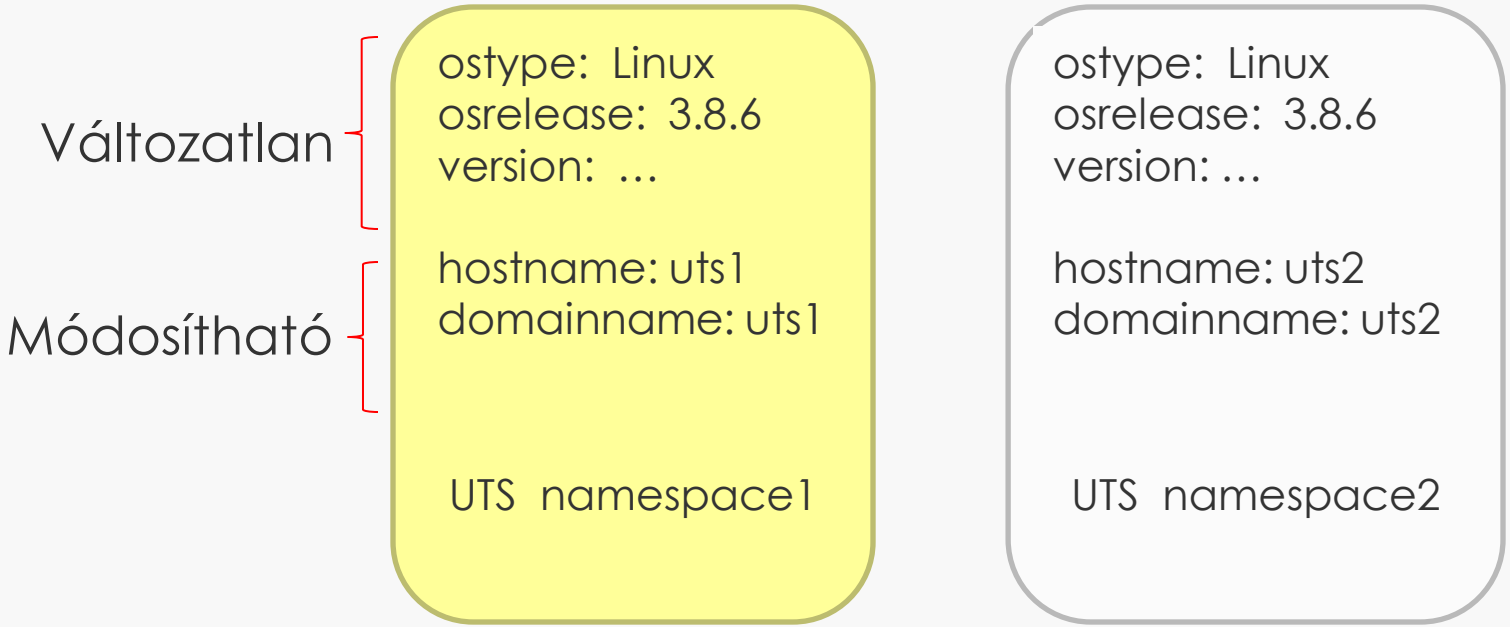
Mount Namespace

- Saját fájlrendszer



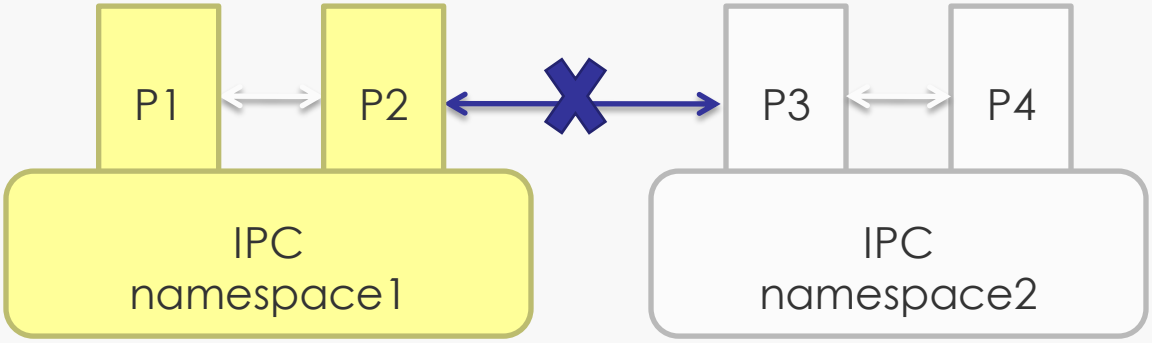
UTS Namespace

- UTS = UNIX Timesharing System
- Saját uts-infó



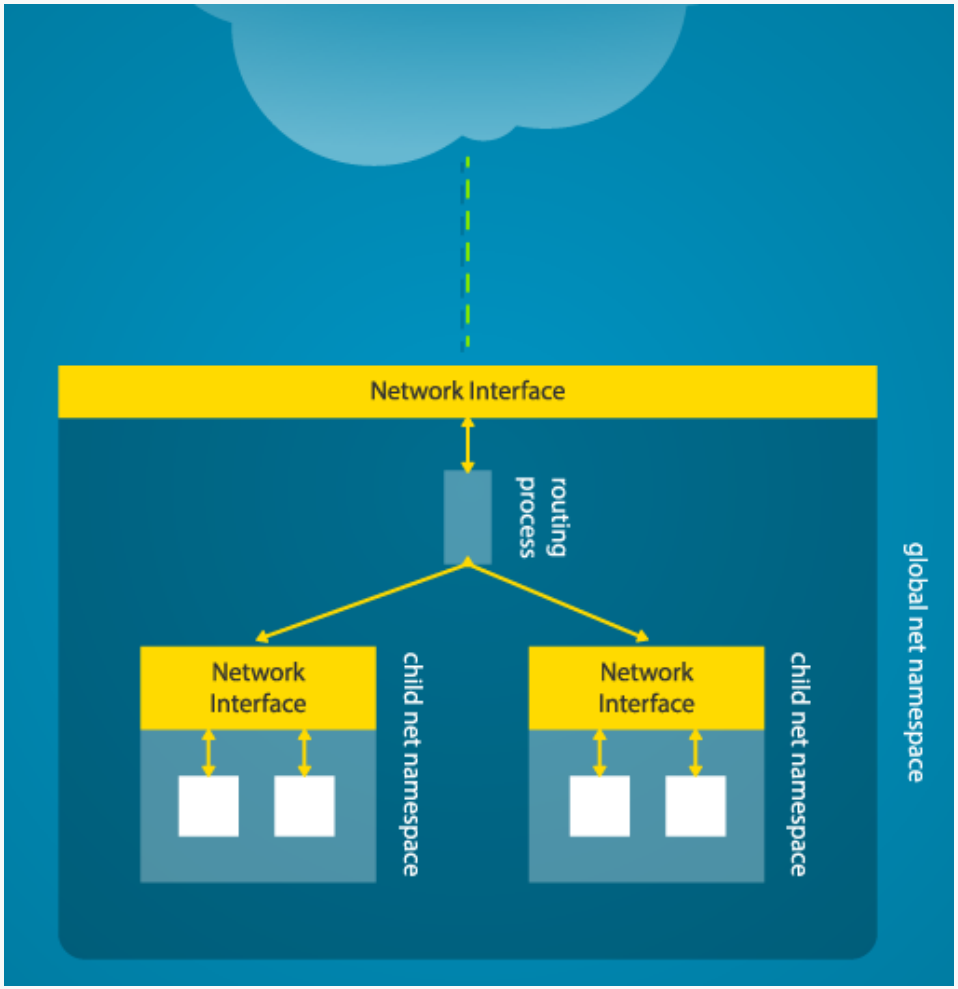
IPC Namespace

- IPC: InterProcess Communication
- Processzek közti kommunikációt izolál:
  - shared memory
  - Semaphore
  - message queue



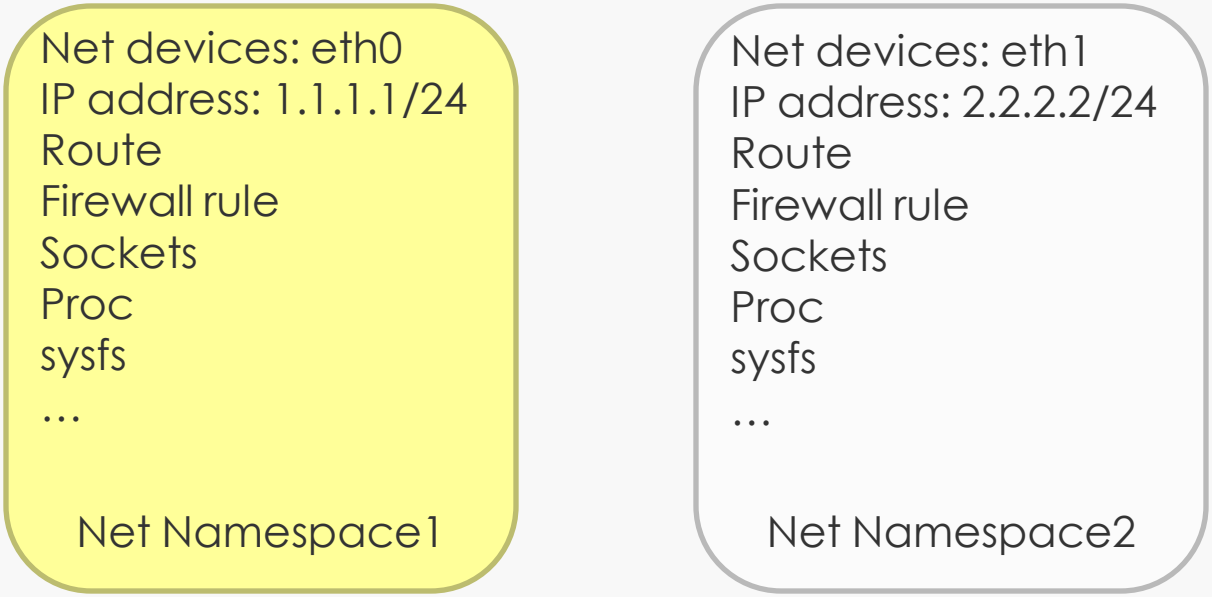
Net Namespace 1/2

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



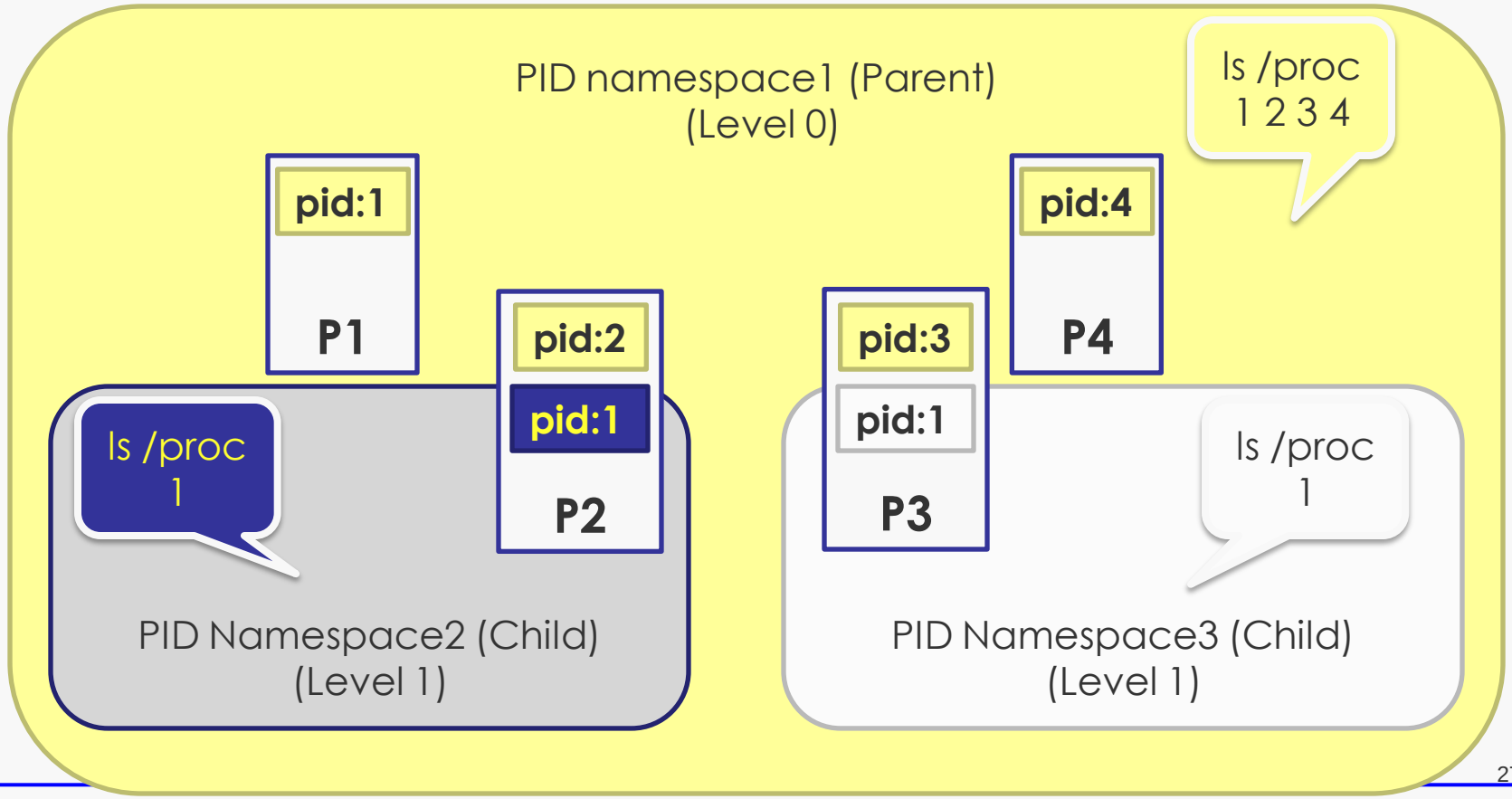
Net Namespace 2/2

- Net namespace: a hálózati erőforrásokat rejti el



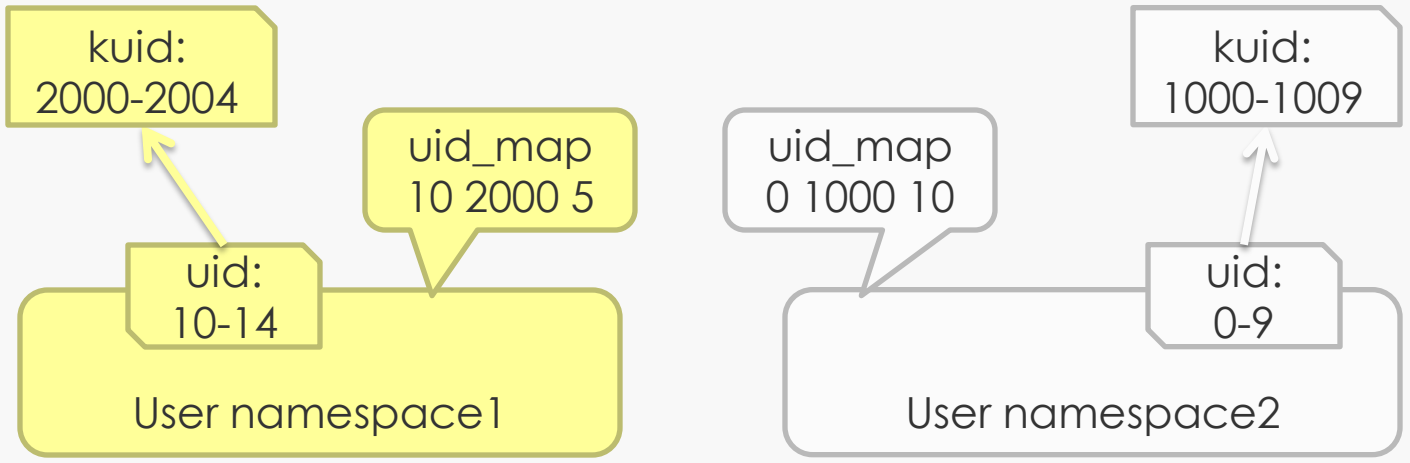
PID Namespace

- PID: Process ID
- Hierarchikus rendszerben virtualizálja



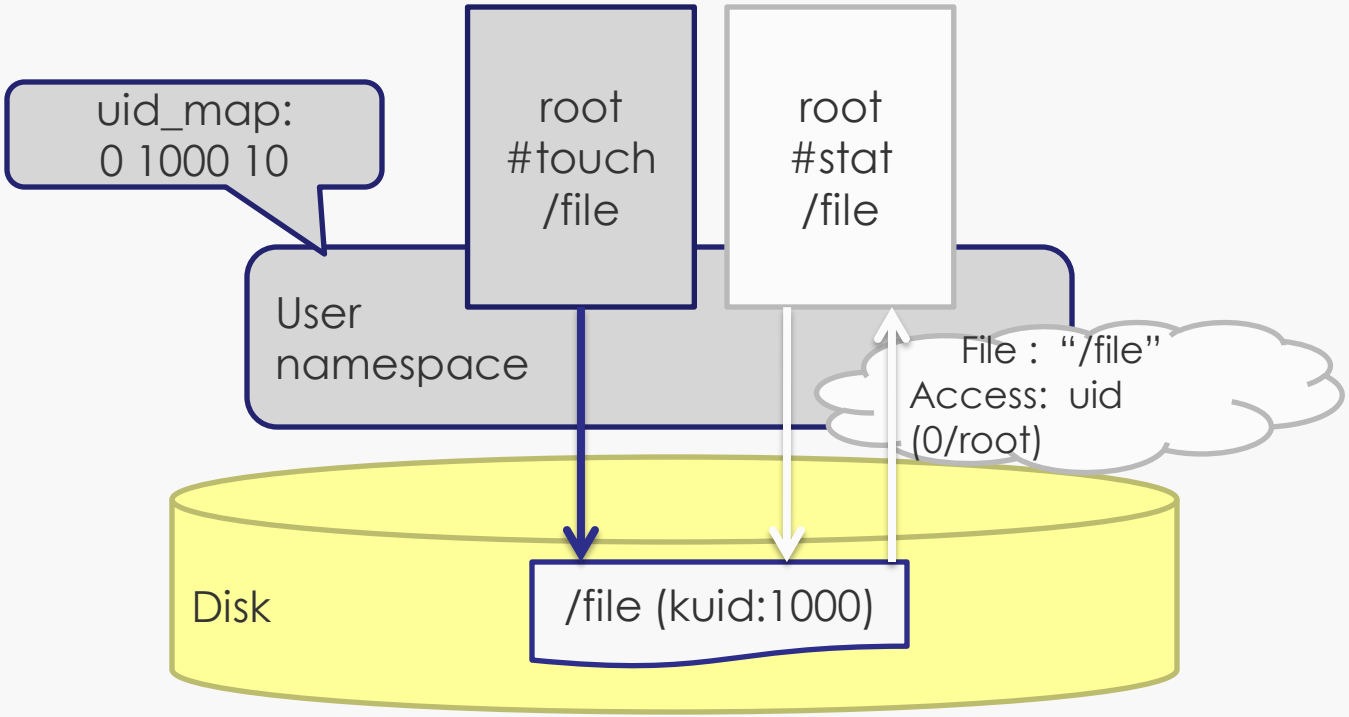
User Namespace

- 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



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

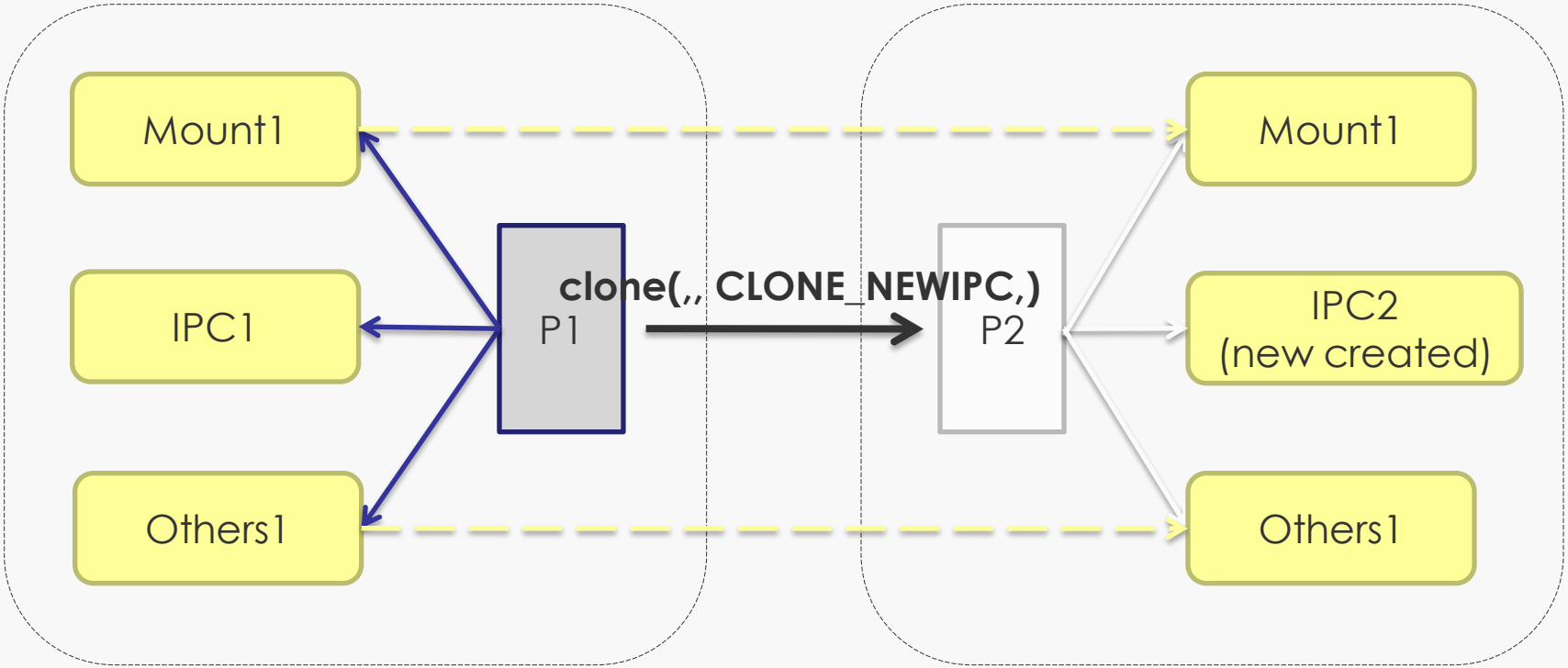
```
int clone(int (*fn)(void *), void *child_stack,
        int flags, void *arg, ...);
```

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

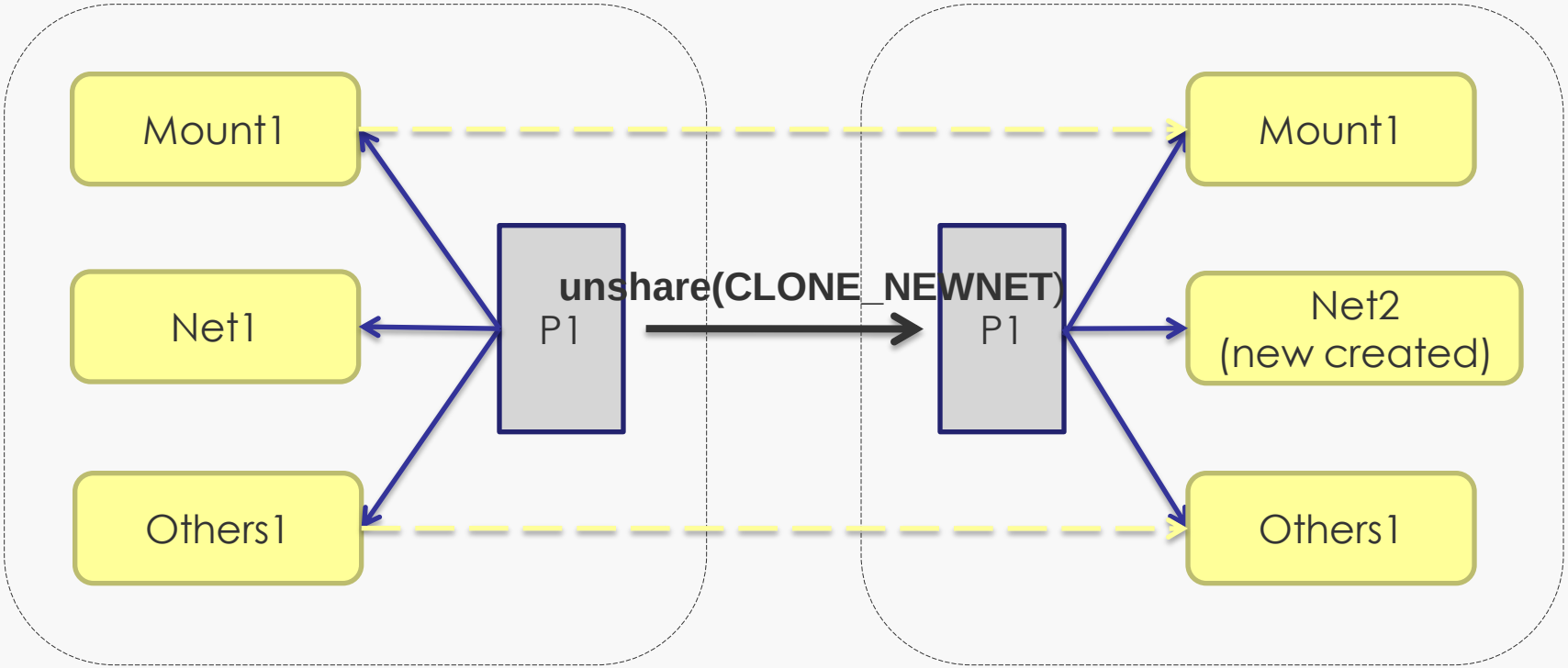
  int unshare(int flags);

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

Rendszerhívások

- unshare

net namespace2 létrehozása



Rendszerhívások

- setns

  int setns(int fd, int nstype);

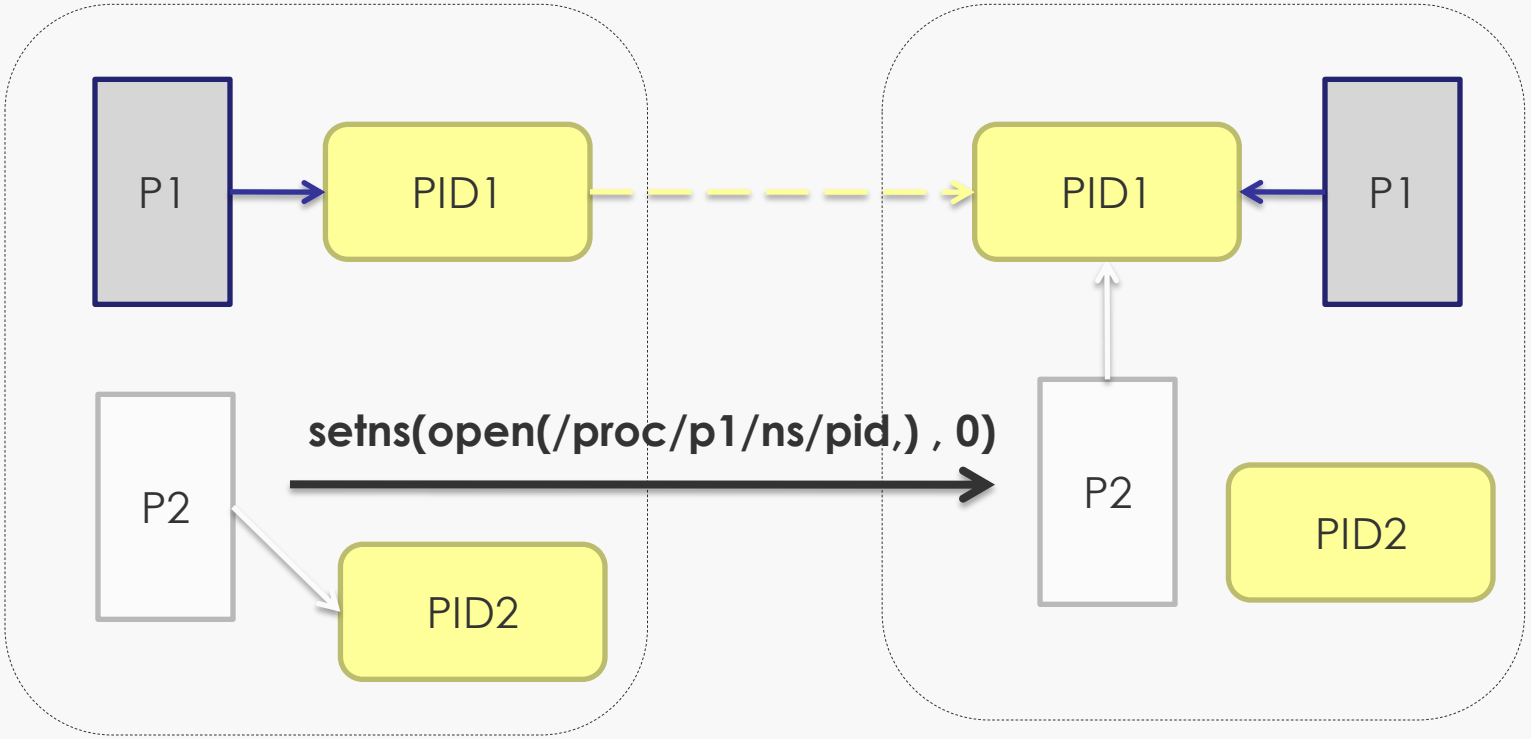
  Új rendszerhívás
  Megadja, milyen névtérbe tartozzon a processz

  @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 erpőforrások

Összehasonlítás

- Vékony (lightweigth) virtualizáció, csak egy OS van (= ugyanaz a kernel)
  - „host share the same kernel with guest”

	Container	**KVM**
performance	Great	Normal
OS support	Linux Only	No Limit
Security	Normal	Great
Completeness	Low	Great

Felmerülő kérdések

- /proc/meminfo, cpuinfo…
  - Kernel space (cgroup)
  - User space (gyenge hatékonyság)
- Új névtér
  - Audit (user namespace-hez rendelni?)
  - Syslog (szükség van rá egyáltalán?)

Felmerülő kérdések

- Bandwidth (sávszélesség kezelése)
  - TC Qdisc
    - On host (hogy rendeljük hozzá a NIC-et a konténerhez)
    - On container (felhasználó módosíthatja)
  - Netfilter
    - Ingress bandwidth kezelése?
- Disk quota
  - Uid/Gid Quota (sok felhasználó)
  - Project Quota (xfs-re OK)

DOCKER

Mi a Docker?

- Docker = Linux container engine.
- Open Source project
  - első verzió: 3/2013 by dotCloud
  - átnevezték Docker Inc-re
- Eredetileg Python kód,, később Go
- https://www.docker.io/
- git repository: https://github.com/dotcloud/docker.git

Docker workflow 1/2

- Egy dev környezetben dolgozik (local machine or container)
- Konténerben fut a többi szolgáltatás (services) (pl. adatbázisok)
  - És ugyanúgy működik

- A „valós” működés tesztelése során :
  - *Másodpercek* alatt fordul (build)
  - *Azonnal* fut

Docker workflow 2/2

- Ha a lokális build OK, akkor
  - Feltölthető a registry-be (public/private)
  - Automatikusan futtatható
  - Üzemi (production, enterprise) környezetben
  - Egyszerű átjárást biztosít a dev és production környezet között
- Hiba esetén: Rollback
  - Vissza lehet térni egy korábbi verzióra

a.) Képfájlok készítése (run/commit megoldással)

- 1) docker run ubuntu bash
- 2) apt-get install this and that
- 3) docker commit <containerid> <imagename>
- 4) docker run <imagename> bash
- 5) git clone git://.../mycode
- 6) pip install -r requirements.txt
- 7) docker commit <containerid> <imagename>
- 8) repeat steps 4-7 as necessary
- 9) docker tag <imagename> <user/image>
- 10) docker push <user/image>

a.) Előnyök/hátrányok

» Előnyök

– Kényelmes, ismert lépések

– roll back/forward – szükség szerint

» Hátrányok

– Kézi-vezérelt folyamat

– Iteratív változások „felgyűlnek”

– Teljes újrafordítás (rebuild) folyamata sok hibalehetőséget tartogat

b.) Képfájlok (Docker images)

- RUN apt-get -y update
- RUN apt-get install -y g++
- RUN apt-get install -y erlang-dev erlang-manpages erlang-base-hipe ...
- RUN apt-get install -y libmozjs185-dev libicu-dev libtool ...
- RUN apt-get install -y make wget
- RUN wget http://.../apache-couchdb-1.3.1.tar.gz | tar -C /tmp -zxf-
- RUN cd /tmp/apache-couchdb-* && ./configure && make install
- RUN printf "[httpd]\nport = 8101\nbind_address = 0.0.0.0" >
- /usr/local/etc/couchdb/local.d/docker.ini

EXPOSE 8101

CMD ["/usr/local/bin/couchdb"]

docker build -t author_name/couchdb

b.) Előnyök

- Gyorsan tanulható
- Könnyen újra-fordítható
  - Caching rendszer segíti ezt
- Egy fájlban meghatározható a build folyamat

Docker

- Multi-arch, multi-OS
- Stabil kontroll API
- Stabil plugin API
- Hibatűrés (resiliency)
- Aláírással ellátott
- Klaszeterezhető

Docker előnyei

- Könnyű installációs folyamat
- Minden alkalmazás fut rajta, sok környezetben
- Ismételhető build folyamat
- Nagy hype, erős közösség, gyors javítások
- Új virtualizációs folyamatok

- **Hátrány**
- A Docker konténer típusa
  - A gazdarendszer OS-e határozza meg
- „Orchestration”
- Hálózati kommunikáció

Docker hátrányai

- A Docker konténer típusa
  - A gazdarendszer OS-e határozza meg
- „Orchestration”
- Hálózati kommunikáció

- De: jelentős és még mindig aktív fejlesztések az elmúlt félévben is
  - A hype és erős közösség előnye

Források

- Docker történet dióhéjban:

  http://www.infoworld.com/article/3025870/paas/the-sun-sets-on-original-docker-paas.html

- Docker áttekintés:

  http://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment

- „The Docker Book”

  http://www.dockerbook.com/#toc

- Docker Meetup @Budapest

  http://www.ustream.tv/recorded/60277876

Források

- „zárójel” – Szolgáltatás biztosítás Docker alapokon

http://synerzip.com/must-see-agile-tv/webinar-secured-area-2/2015-2/running-microservices-on-docker/