# CPU Namespace

## A mechanism to isolate CPU topology information in the Linux kernel

Pratik Rajesh Sampat <pratik.sampat@in.ibm.com>
Gautham R. Shenoy <gautham.shenoy@amd.com>

LINUXCONFAU
January 14, 2022

# Outline

N°1

**Motivation**
Purpose of sysfs in the world of containers

N°2

**Existing Solutions**
Existing cgroup interface, LXCFS, New Proc interfaces,

N°3

**CPU namespace**

N°4

**Experimental Results**

N°5

**Challenges and Future**

# Motivation

# Sysfs in the world of containers

- Sysfs: Pseudo filesystem that can expose kernel information to userspace such as information about kernel subsystems and hardware
- Applications determine system resources and usage:- sys and procfs
- Containerized applications can be restricted via cgroups cpuset. However, unaware of these restrictions can still look at traditional interfaces for information
- Problem also exists outside the realm of containers.
    - Ex: taskset => sched_get/setaffinity() can set CPU restrictions on applications but applications can still make decisions based on traditional interfaces

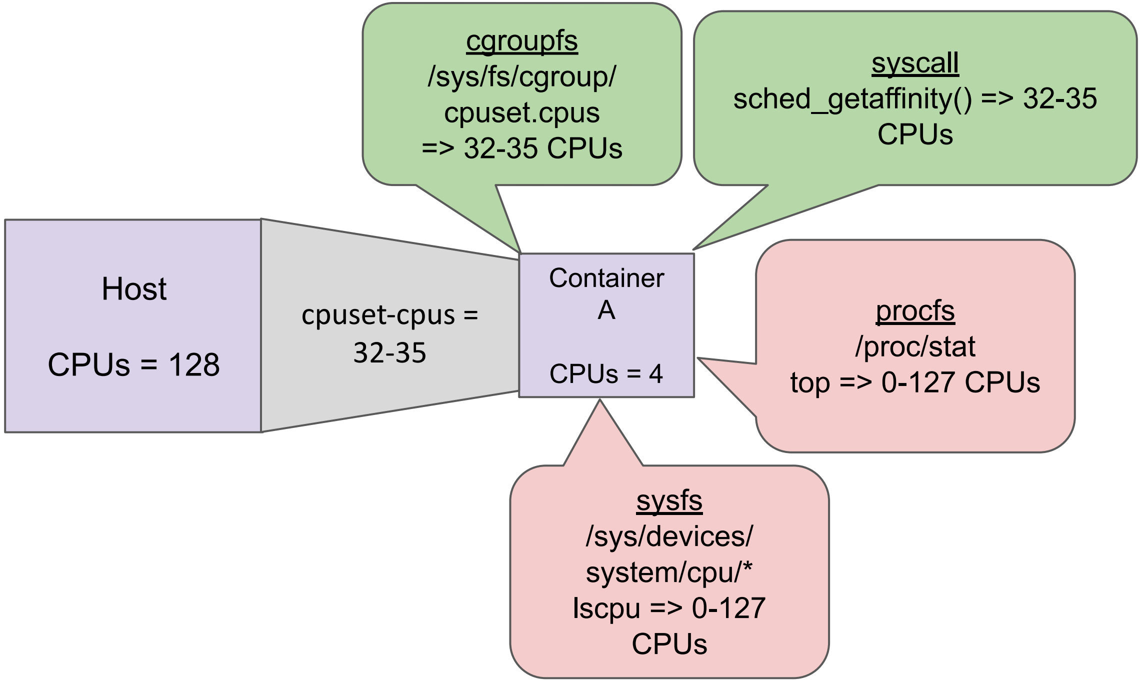
What does sysfs and procfs really mean in the context of container restriction?

What are the implications of exposing this information when applications can only use a subset of them?

# Implication: inconsistency of information

The control and the display interface is fairly disjoint with each other.

Restrictions can be set through control interfaces like cgroups cpuset, however applications can view multiple interfaces to retrieve CPU information and make decisions based on it.

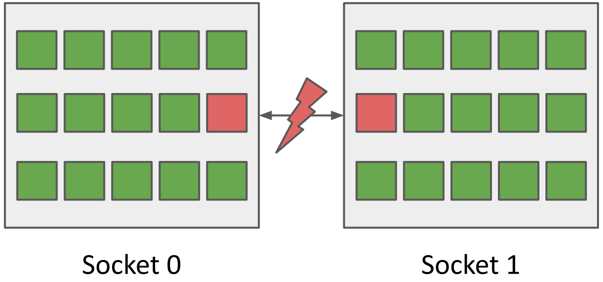


# Implication: fair use

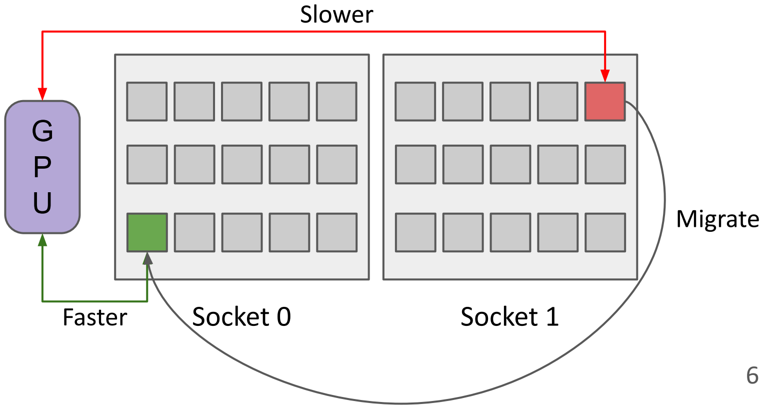
In the context of a restricted container in a multi-tenant system, should all the information about the topology be available for the container to see?

Can this information potentially be misused?

- Could a user schedule workloads across sockets such that the bus is flooded and other container tenants experience slowdown?



- Could a user identify its vicinity from peripheral such as GPUs and schedule themselves closer to get latency advantage compared to the rest of workloads?



# Existing Solutions

# Existing Solutions

## Just look at cgroupfs!

+ If you need information about your restrictions look at the interface that restricts it

- A lot of applications legacy and otherwise rely on traditional interfaces like sys and proc

/ Need to interpret concepts like period-quota(time) in terms of threads(space) to spawn

/ While cgroups can be used to extract information, in crux they are a control mechanism for the host rather than a display interface inside the container

## Userspace solution: LXCFS

Userspace file system that bind-mounts over the existing sys and procfs to provides consistent information in accordance to current restrictions

A cgroupfs-like tree which is container aware

+ Light, easy to use userspace tool. Currently in use with Kubernetes as described by Google Anthos[1] and Alibaba Cloud[2]

- Needs explicit setup for applications that experience the effects of incorrect information

## Other proposed In-kernel solution

A RFC patchset[3] which added /proc/self/meminfo respecting cgroup restrictions for the memory consistency problem.

+ Introduces standards for exposing and interpreting information

+ A clean new interface. Does not break any assumptions of the already established by sys and proc

/ A sizable number of applications still look at sys and proc instead of cgroup, motivation to use this new interface may be low. A comment[4] highlights of the same as well.

# Existing Solutions

## Just look at cgroupfs!

Present information about restrictions

## Userspace solution: LXCFS

Userspace file system that bind-mounts over the existing sys and procfs to provides consistent information in accordance to current restrictions

A cgroupfs-like tree which is container aware

+ Light, easy to use userspace tool. Currently in use with Kubernetes as described by Google Anthos[1] and Alibaba Cloud[2]

- Needs explicit setup for applications that experience the effects of incorrect information

## Other proposed In-kernel solution

A RFC patchset[3] which added /proc/self/meminfo respecting cgroup restrictions for the memory consistency problem.

+ Introduces standards for exposing and interpreting information

+ A clean new interface. Does not break any assumptions of the already established by sys and proc

/ A sizable number of applications still look at sys and proc instead of cgroup, motivation to use this new interface may be low. A comment[4] highlights of the same as well.

# Existing Solutions

## Just look at cgroupfs!

Present information about restrictions

## Userspace solution: LXCFS

Consistently with all existing interfaces

## Other proposed In-kernel solution

A RFC patchset[3] which added /proc/self/meminfo respecting cgroup restrictions for the memory consistency problem.

- + Introduces standards for exposing and interpreting information
- + A clean new interface. Does not break any assumptions of the already established by sys and proc
- / A sizable number of applications still look at sys and proc instead of cgroup, motivation to use this new interface may be low. A comment[4] highlights of the same as well.

# Existing Solutions

| Just look at cgroupfs! | Userspace solution: LXCFS | Other proposed In-kernel solution |
|---|---|---|
| Present information about restrictions | Consistently with all existing interfaces | Introduces standardization by an In-kernel solution |

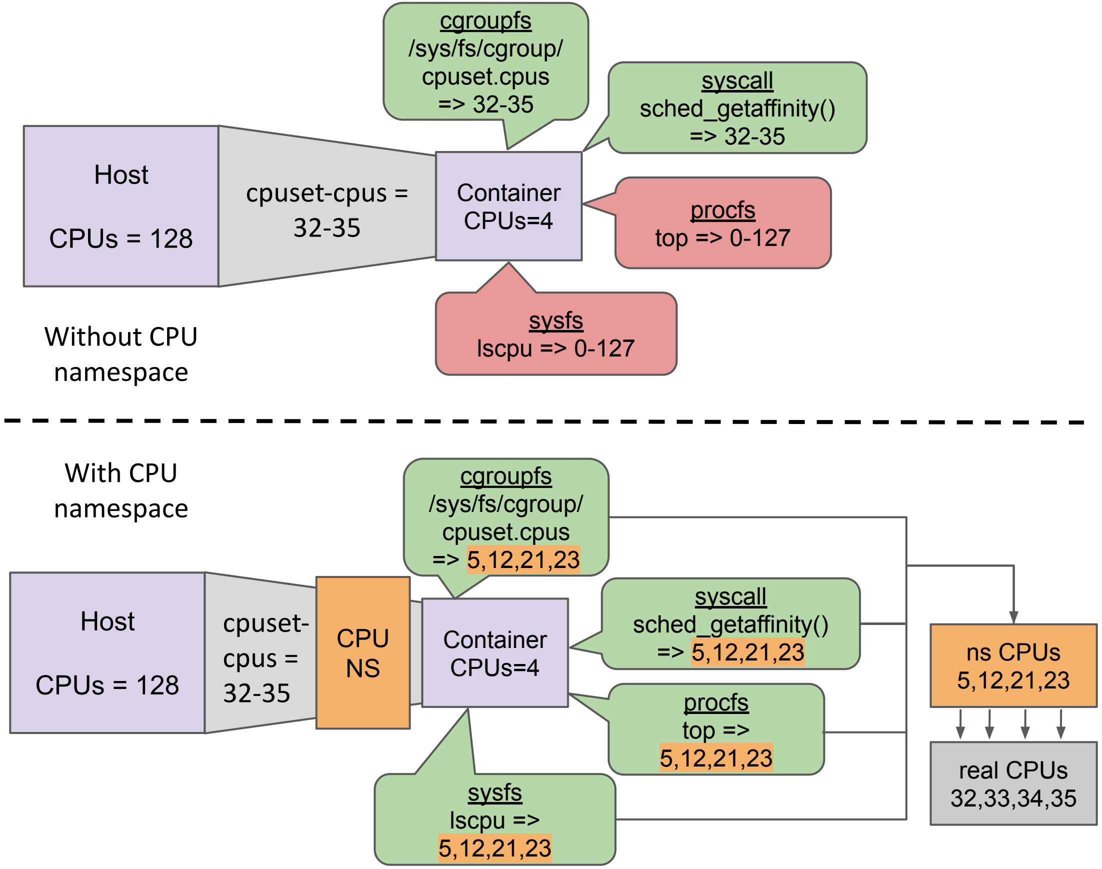
N°3

# CPU Namespace

# CPU namespace[5]

A mechanism that isolates CPU information for each task and presents restrictions consistently with all the control and display interfaces.

The translation of CPU information is maintained by a scrambled map of namespace CPUs to Logical CPUs.
Eg. real CPU 32 => ns CPU 5



# Experimental Results

Machine: IBM Power 9 - 44 SMT4 Cores => 176 CPUs

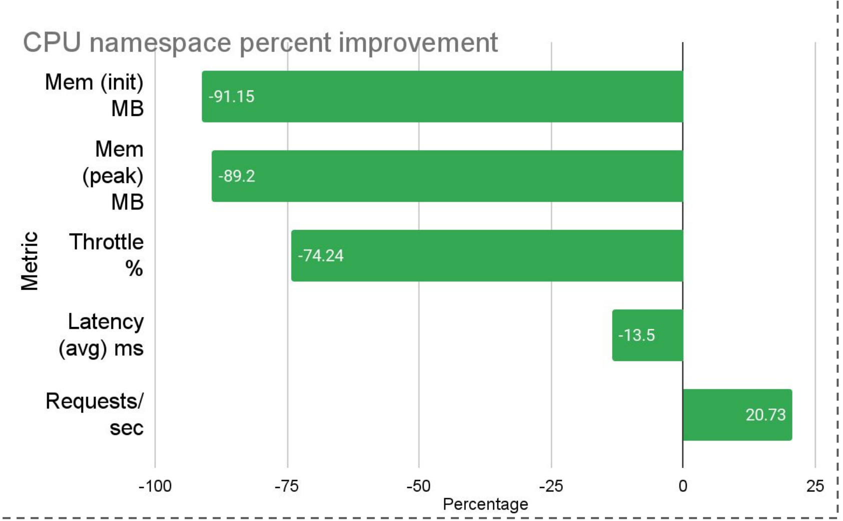
# Experiment

- Benchmarking nginx (HTTP server) with a multithreaded workload called wrk (HTTP load generator)
- Nginx is configured with worker_processes auto; to enable the application to manage resources based on the system configuration
- The nginx container is configured to cpuset to 4 CPUs
- The wrk benchmark spawns 500 requests in 30 seconds for 4 threads

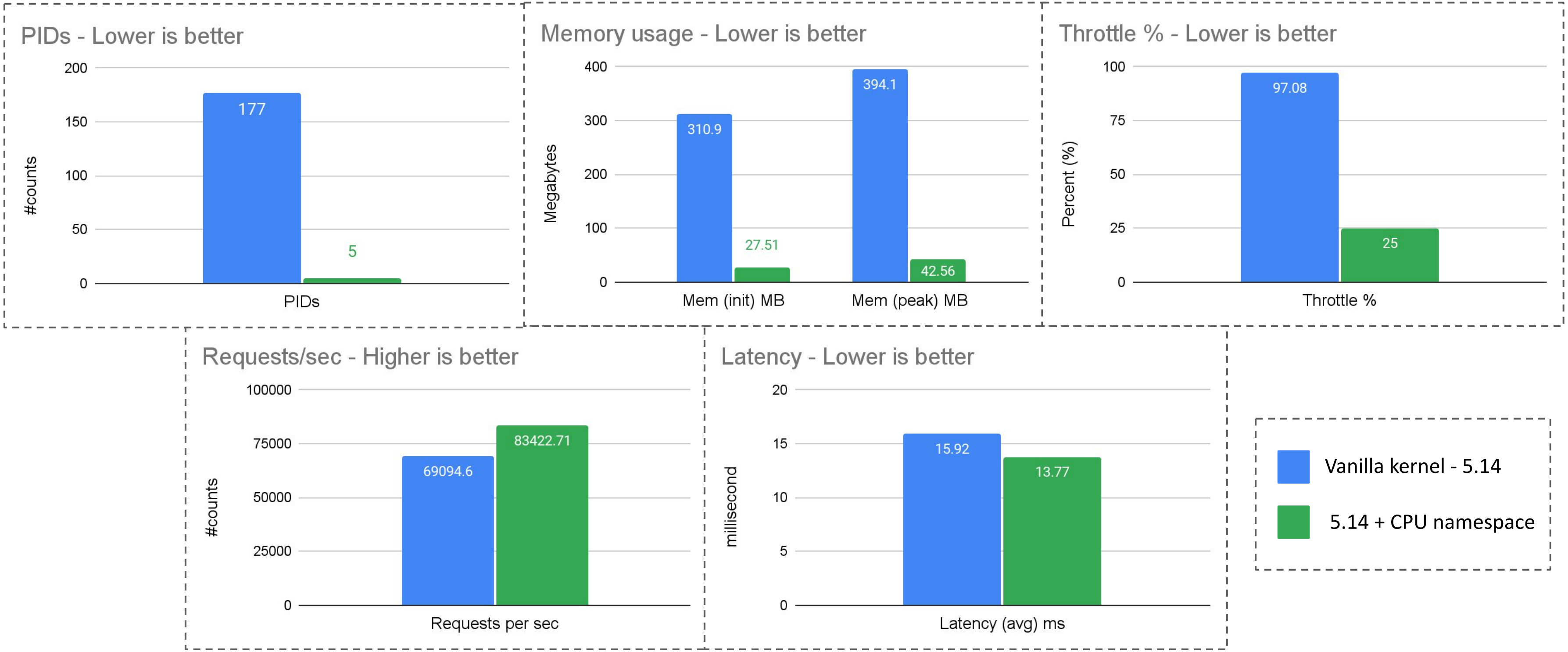
Metrics of Measurement:

Vanilla 5.14 vs CPU namespace 5.14

- Memory usage: init and peak - Lower is better
- Cgroup CPU Throttle % - Lower is better
- Workload: Latency - Lower is better
- Workload: Requests/sec - Higher is better
- Number of PIDs/threads spawned - Lower is better

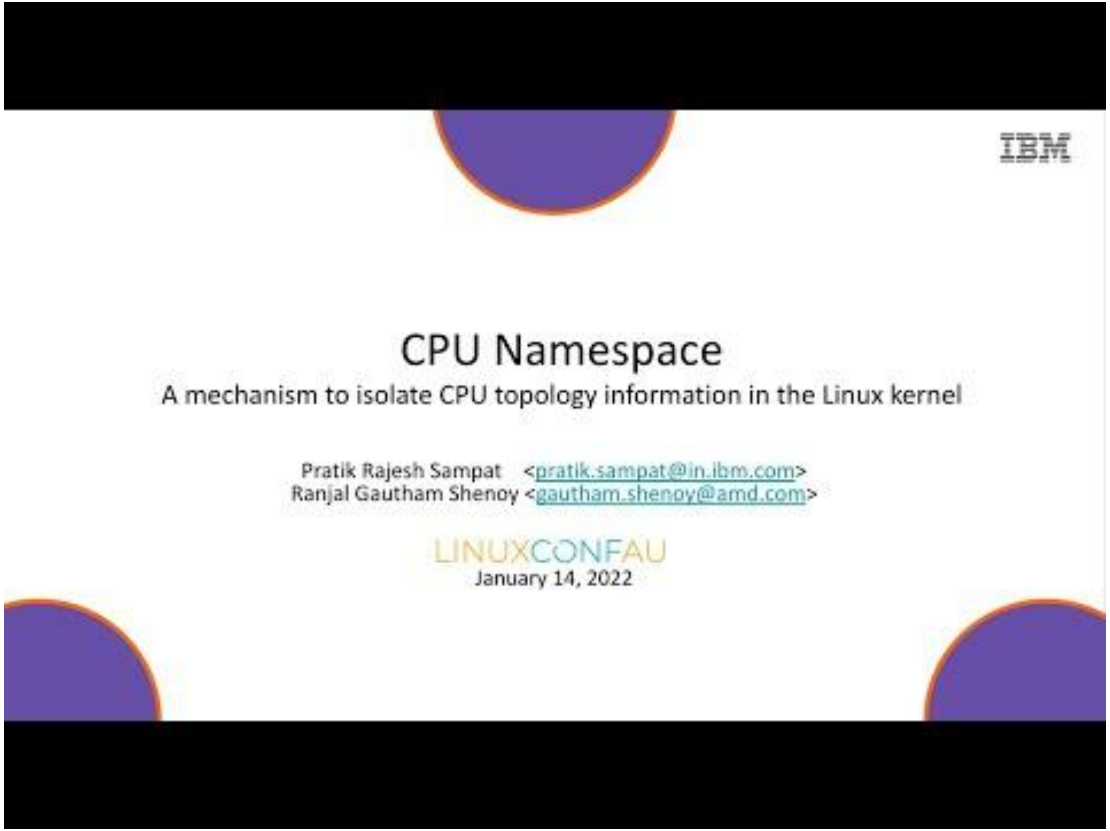


# Results



PIDs - Lower is better
#counts
200
150
100
50
0
177
5
PIDs
Memory usage - Lower is better
Megabytes
400
300
200
100
0
310.9
27.51
394.1
42.56
Mem (init) MB
Mem (peak) MB
Throttle % - Lower is better
Percent (%)
100
75
50
25
0
97.08
25
Throttle %
Requests/sec - Higher is better
#counts
100000
75000
50000
25000
0
69094.6
83422.71
Requests per sec
Latency - Lower is better
millisecond
20
15
10
5
0
15.92
13.77
Latency (avg) ms
🟦 Vanilla kernel - 5.14
🟩 5.14 + CPU namespace

# Demo



CPU Namespace

A mechanism to isolate CPU topology information in the Linux kernel

Pratik Rajesh Sampat <pratik.sampat@in.ibm.com>
Ranjal Gautham Shenoy <gautham.shenoy@amd.com>

LINUXCONFAU
January 14, 2022

N°5

# Challenges and Future

# Challenges with the current design

- Until now namespaces and cgroups have been fairly disjoint from one another. CPU namespace breaks that. Without the CPU/CPUSet cgroup the CPU namespace loses its meaning.

- The current design only addresses restriction in space and not time. Containers also frequently use cfs periods and quota in the form of millicores. How does the information need to be exposed for these restrictions?

- While CPU namespace mitigates the potential misuse stemming from the knowledge of topology by obfuscation of information, the topology can still be roughly figured out with IPI latencies to determine siblings or far away cores.

# Future

- The intention is to spark a discussion on the problem rather than to be a know all and end solution
- If the solution is for applications to change and look at cgroupfs, there are exciting discussions[6] around exporting more useful metrics to entice applications to change
- If the solution is external userspace programs bind-mounting custom sys and procs then should that be the norm for the future as well?

# Legal

- This work represents the view of the authors and does not necessarily represent the view of the employers (IBM, AMD)
- Author Gautham R. Shenoy’s contributions were while he was working at IBM
- IBM and IBM (Logo) are trademarks or registered trademarks of International  Business  Machines in United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product and service names may be trademarks or service marks of others

# References

1. LXCFS - Google Anthos use-case: https://cloud.google.com/blog/products/containers-kubernetes/migrate-for-anthos-streamlines-legacy-java-app-modernization
2. LXCFS - Alibaba use- case: https://www.alibabacloud.com/blog/kubernetes-demystified-using-lxcfs-to-improve-container-resource-visibility_594109
3. In-kernel solution - /proc/self/meminfo: https://lore.kernel.org/lkml/ac070cd90c0d45b7a554366f235262fa5c566435.1622716926.git.legion@kernel.org/
4. In-kernel solution review comment - /proc/self/meminfo: https://lore.kernel.org/lkml/20210615113222.edzkaqfvrris4nth@wittgenstein/
5. CPU namespace patches: https://lore.kernel.org/lkml/20211009151243.8825-1-psampat@linux.ibm.com/
6. Comment suggesting introducing more metrics in cgroups: https://lore.kernel.org/lkml/YW2g73Lwmrhjg%2Fsv@slm.duckdns.org/
7. CPU namespace phoronix article: https://www.phoronix.com/scan.php?page=news_item&px=Linux-CPU-Namespace

# Thank you!

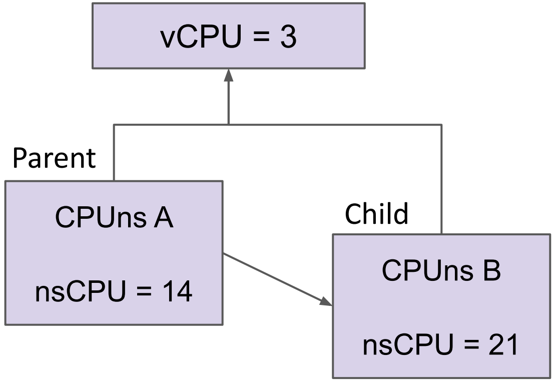
# Additional information

# CPU namespace - Design

Virtualization of CPU information is created by a scrambled map of namespace CPUs to Logical CPUs.

The virtualization make a flat hierarchy of 1:1 mappings. Thus making translations O(1)

More information about design posted here: [pratiksampat.github.io/cpu_namespace.html](pratiksampat.github.io/cpu_namespace.html)



vCPU = 3

Parent

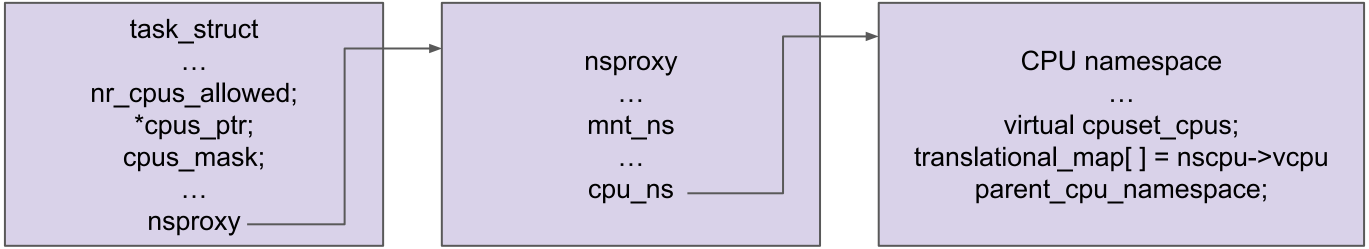
CPUns A

nsCPU = 14

Child

CPUns B

nsCPU = 21

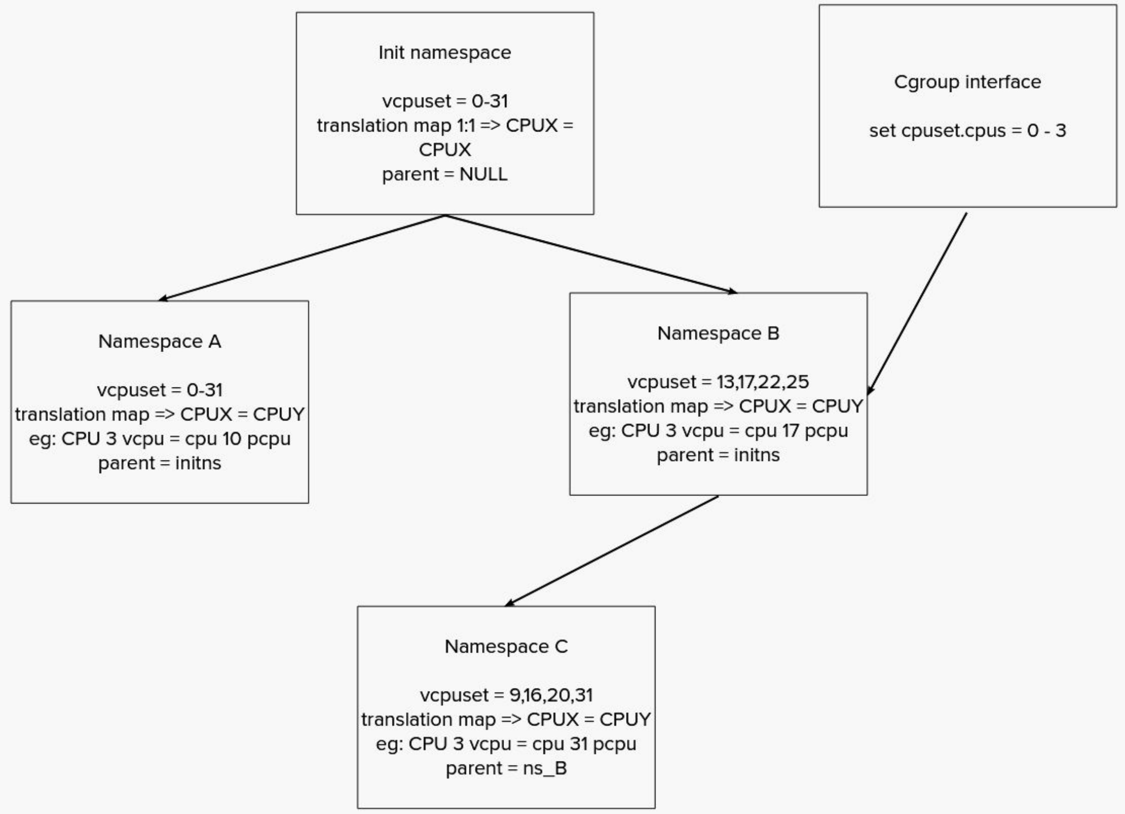


task_struct
…
nr_cpus_allowed;
*cpus_ptr;
cpus_mask;
…
nsproxy

nsproxy
…
mnt_ns
…
cpu_ns

CPU namespace
…
virtual cpuset_cpus;
translational_map[ ] = nscpu->vcpu
parent_cpu_namespace;

# CPU namespace - Block example



Init namespace

vcpuset = 0-31
translation map 1:1 => CPUX = CPUX
parent = NULL

Cgroup interface

set cpuset.cpus = 0 - 3

Namespace A

vcpuset = 0-31
translation map => CPUX = CPUY
eg: CPU 3 vcpu = cpu 10 pcpu
parent = initns

Namespace B

vcpuset = 13,17,22,25
translation map => CPUX = CPUY
eg: CPU 3 vcpu = cpu 17 pcpu
parent = initns

Namespace C

vcpuset = 9,16,20,31
translation map => CPUX = CPUY
eg: CPU 3 vcpu = cpu 31 pcpu
parent = ns_B