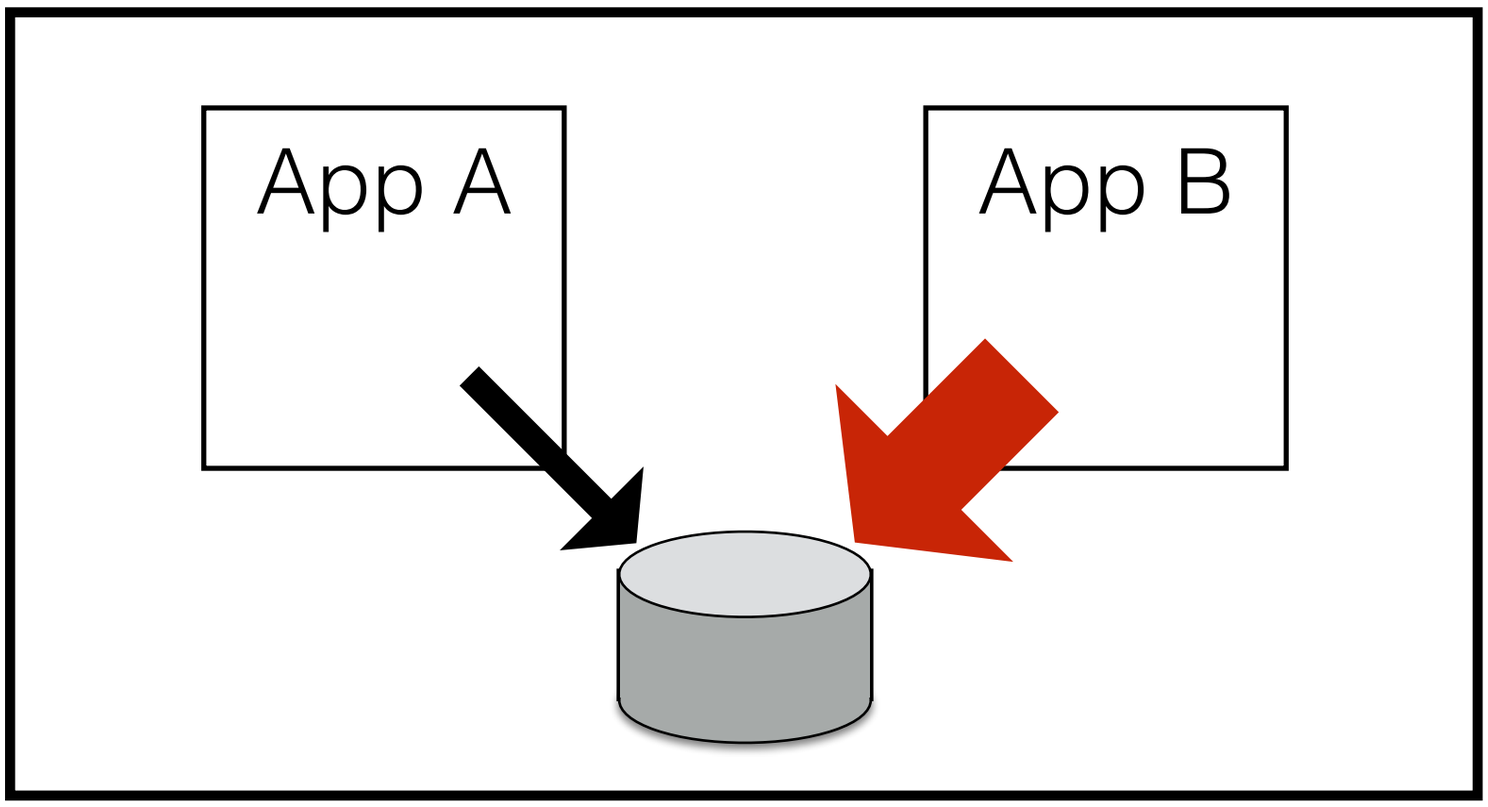
Namespaces

Tyler Caraza-Harter

Isolating Performance

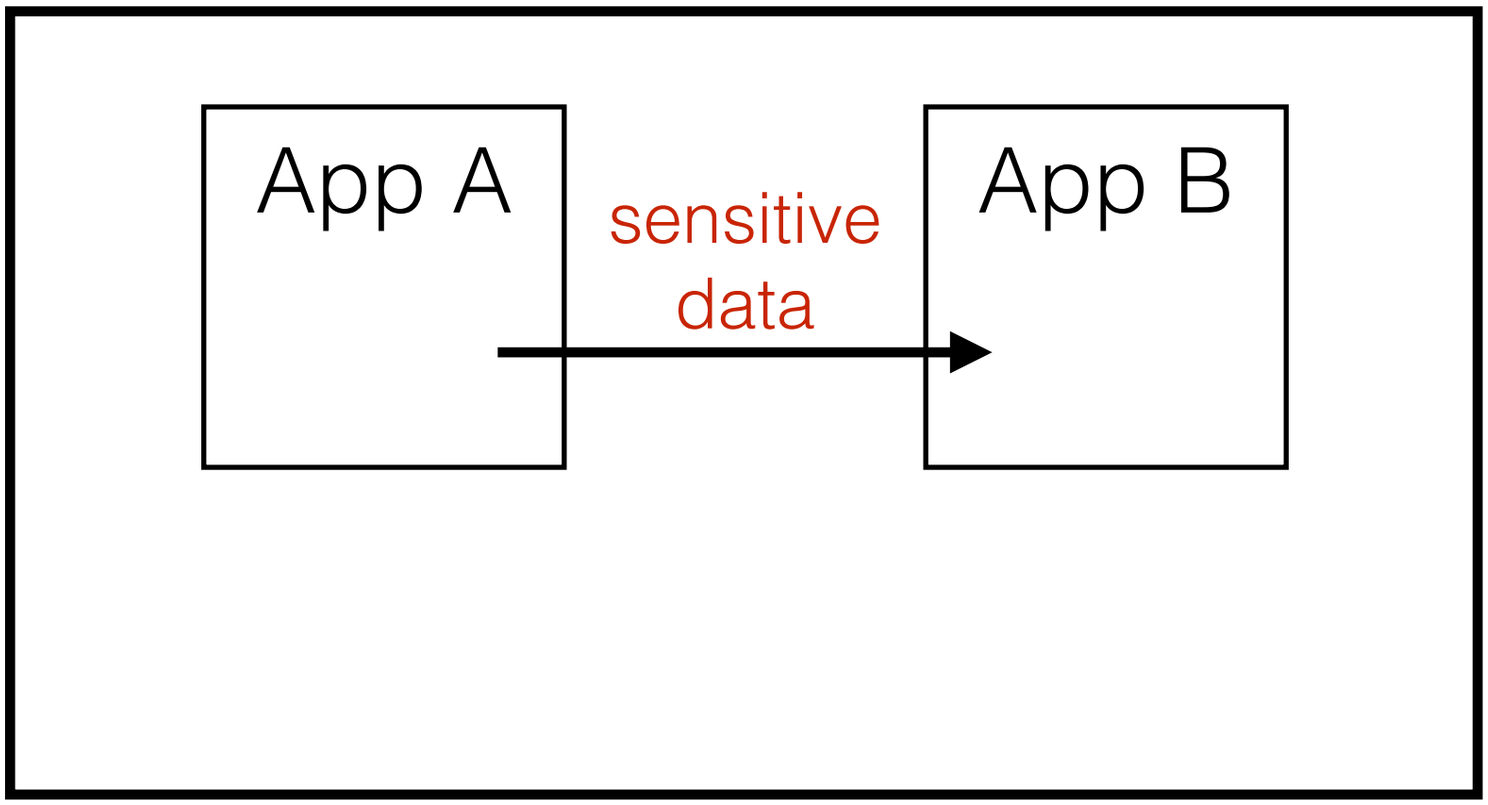


Physical Machine

don’t want: unfairness

Another session (on cgroups) looks at **performance** isolation

Isolating Data



Physical Machine

don't want: leaks

Linux

- older features: virtual memory, ACLs, etc
- newer features: namespaces (this session)

Outline

Definition (and Analogy to Virtual Memory)

Interface

- namespaces
- setns, unshare, clone3

PIDs

- parent, child, grandchild
- one process with multiple IDs

mount

- implementation
- systemd perf issue

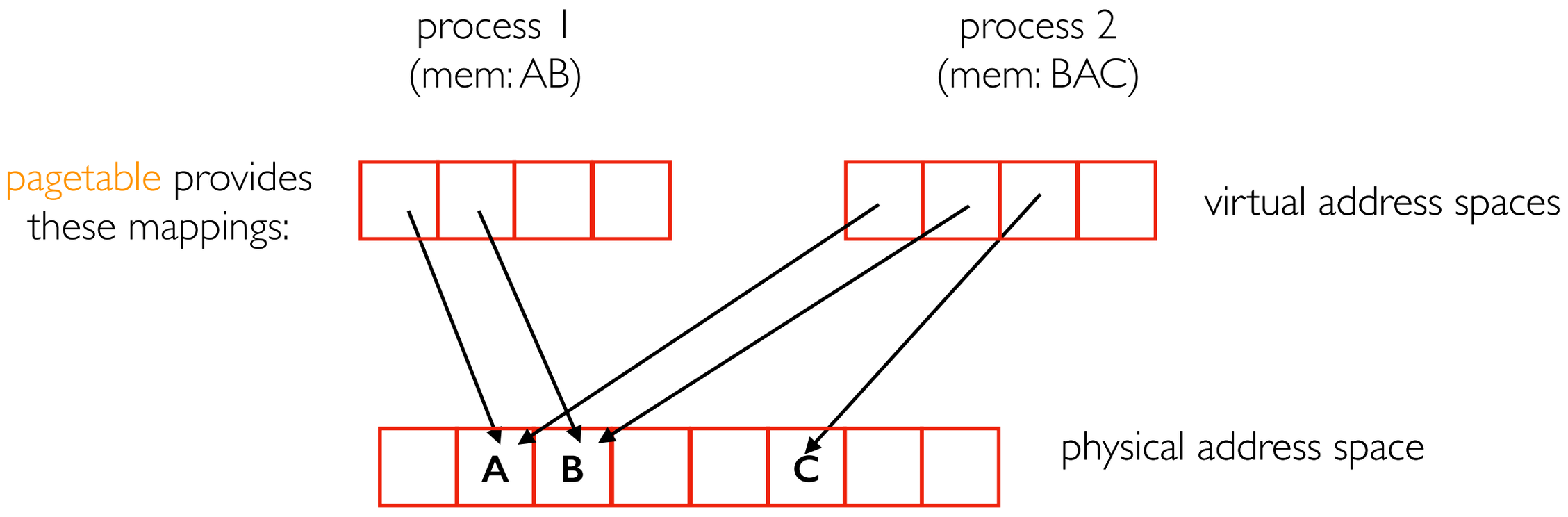
network

- perf issues: RCU
- one top-level one?

Namespaces

Namespace: a kernel data structure that maps virtual resources to physical resources

If pagetables were a recent innovation, they probably would have been called "*memory namespaces.*"



Tasks

In userspace, we talk about "processes" and "threads".

In the kernel, the "task" is the main abstraction -- the schedulers chooses tasks to run on CPUs. Kernel tasks are always in kernel space (e.g., FS journal task); user tasks run in userspace normally, but switch to kernel (e.g., during syscall).

Two different tasks **can share any subset** of the following:

- pagetable (when they share, we call them threads in the same process)
- file descriptor table
- mount namespace, network namespace, PID namespace, user namespace
- cgroups
- ...

Many ways to innovate mixing and matching what is shared. Example: Kubernetes.

- Tasks in the same "pod" share a network namespace. Pods hold containers.
- Tasks in the same "container" share mount namespace. Containers hold processes.
- Tasks in the same "process" share pagetable, FD table, etc.

Outline

Definition (and Analogy to Virtual Memory)

Interface

- namespaces
- setns, unshare, clone3

PIDs

- parent, child, grandchild
- one process with multiple IDs

mount

- implementation
- systemd perf issue

network

- perf issues: RCU
- one top-level one?

Flags and FDs

Namespace Flags: `CLONE_NEW*`

- `CGROUP`, `IPC` (inter-process communication), `NET`, `NS` (mount namespace), `PID` (processes), `TIME` (new?), `USER`, `UTS` (sethostname)
- Can "or" together: CLONE_NEWNS | CLONE_NEWPID

The procfs gives us FDs referring to the namespaces used by any process.

ls /proc/[pid]/ns

- cgroup
- ipc
- mnt
- net
- pid
- pid_for_children
- time
- time_for_children
- user
- uts

Syscalls using Namespaces

from opening:
/proc/[pid]/ns/[namespace]

`setns(int fd, int nstype)`

- used in SOCK paper; join existing namespace of other process
- disadvantage: need pre-running process doing nothing to have pre-initialized namespaces (see `spin` process, which we should remove)

CLONE_NEWPID
(or similar)

`unshare(int flags)`

- used in OL now: create new namespaces for the process
- disadvantage: need parent/child/grandchild pattern (more in PIDs section)

`clone3(struct clone_args, ...);`

- what OL should probably use

```
struct clone_args {
    u64 flags;
    u64 cgroup; // Linux 5.7
    ...
};
```

OpenLambda: Primary Usage

OpenLambda Usage
min-image/runtimes/python/ol.c

```
static PyObject *ol_unshare(PyObject *module) {
  int res = unshare(CLONE_NEWUTS|CLONE_NEWPID|CLONE_NEWIPC);
  return Py_BuildValue("i", res);
}
```

`unshare(int flags)`

- used in OL now: create new namespaces for the process
- disadvantage: need parent/child/grandchild pattern (more in PIDs section)

notes on others

- net: don't care about port collision: lambdas aren't servers
- user: why not have all run as the same?
- mount: chroot is faster
- time: seccomp blocks changing time
- cgroup: user code doesn't need a view of it

Outline

Definition (and Analogy to Virtual Memory)

Interface

- namespaces
- setns, unshare, clone3

PIDs

- parent, child, grandchild
- one process with multiple IDs

mount

- implementation
- systemd perf issue

network

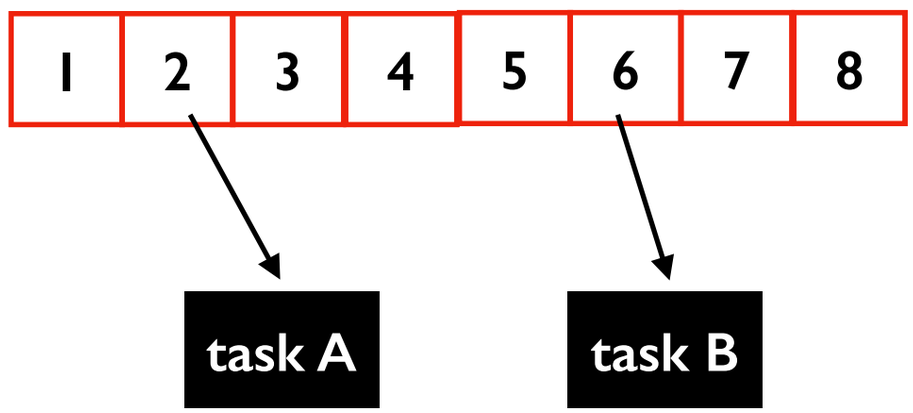
- perf issues: RCU
- one top-level one?

PID Namespace

Important: if a process can't be referenced by PID in another container, things like "kill -9 PID don't work.

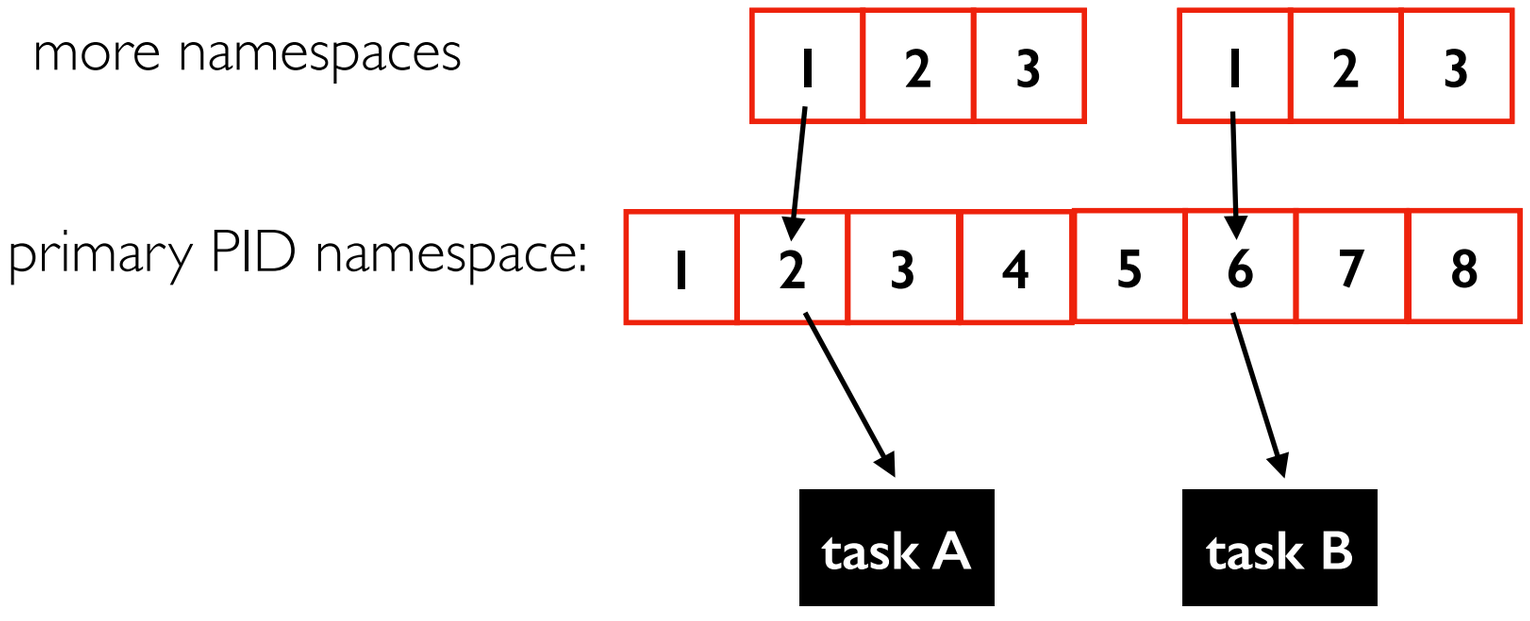
process IDs:
A (PID 2)
B (PID 6)

primary PID namespace:



PID Namespace

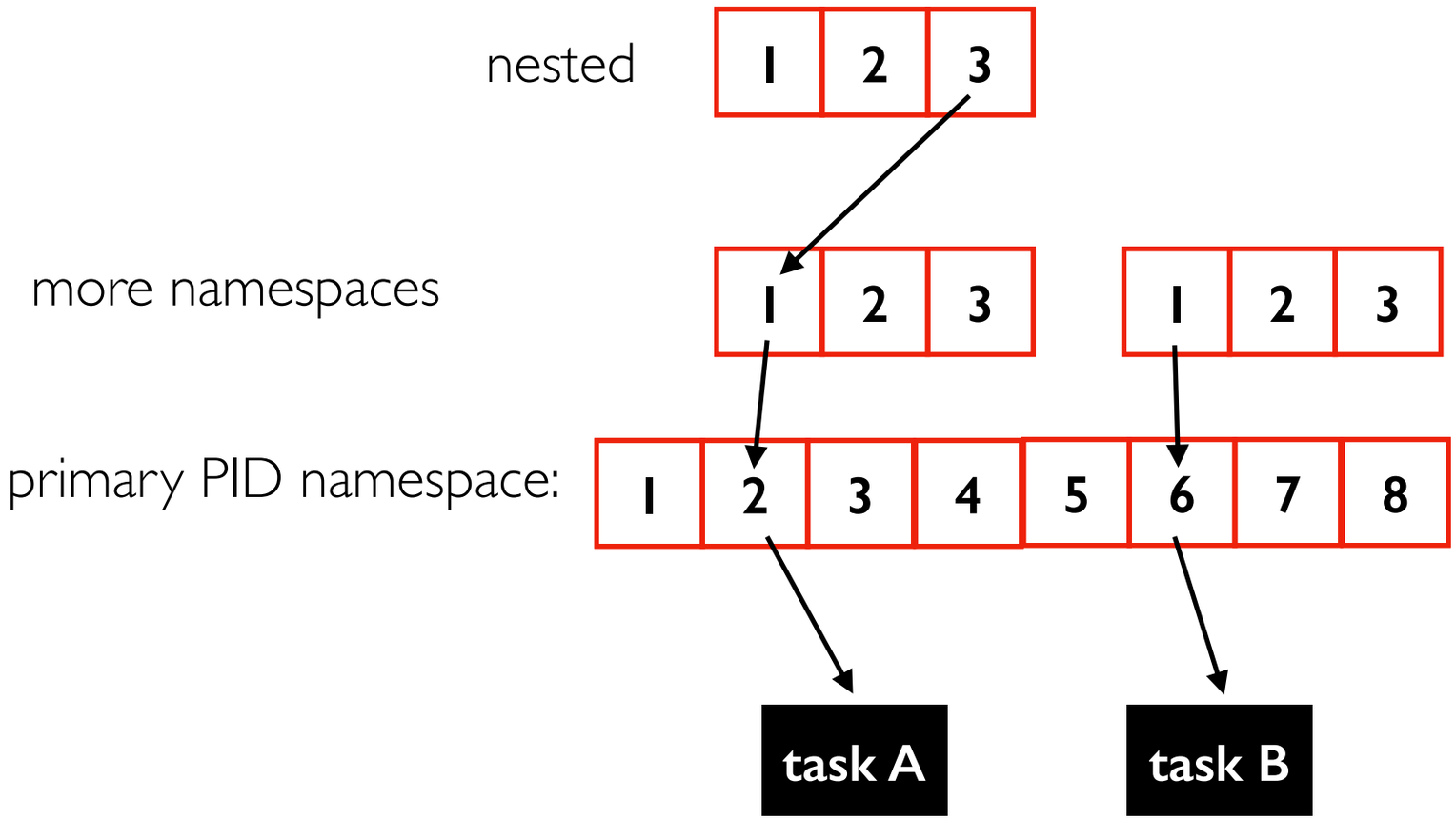
Important: if a process can't be referenced by PID in another container, things like "kill -9 PID don't work.



process IDs:
A (PID 1)
B (PID 1)

PID Namespace

Important: if a process can't be referenced by PID in another container, things like "kill -9 PID don't work.



Careful! If you want to use a PID (to kill it, move it to a cgroup, etc), you need the right one based on where you're running. Task A could be 2, 1, or 3 depending on that.

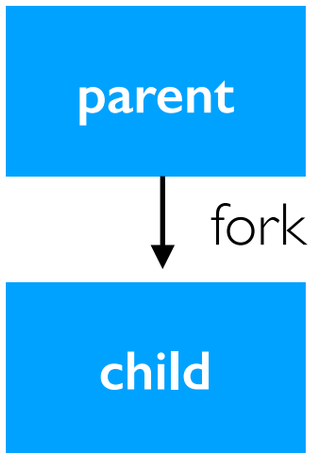
Early OL bug: give task A a FD to a cgroup it should join. Ask task A to write its PID to cgroup.procs. Issue: A sees itself as PID 3 (container namespace), but /sys/fs/cgroups2 sees it as 2 (primary namespace).

PID Namespace Limitation

Apparently you can't dynamically change PID namespace, even though unshare and setns generally behave like that (maybe a process seeing its PID change breaks something?).

Changing the PID namespace of a process only affects CHILDREN of that process.

Problem with fork: say we want a specific child in new PID namespace.



unshare PID namespace here?

No: we don't want to affect all children of parent, just one.

unshare PID namespace here?

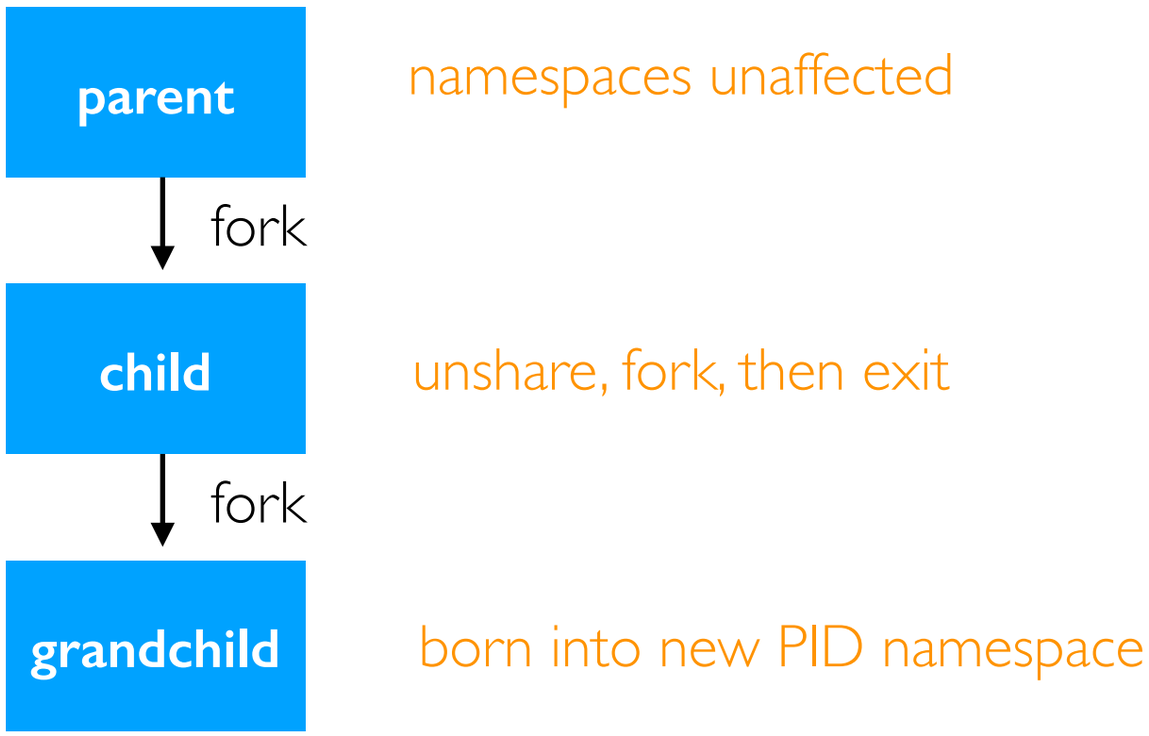
No: child's PID can't change after created.

PID Namespace Limitation

Apparently you can't dynamically change PID namespace, even though unshare and setns generally behave like that (maybe a process seeing its PID change breaks something?).

Changing the PID namespace of a process only affects CHILDREN of that process.

Problem with fork: say we want a specific child in new PID namespace.



Outline

Definition (and Analogy to Virtual Memory)

Interface

- namespaces
- setns, unshare, clone3

PIDs

- parent, child, grandchild
- one process with multiple IDs

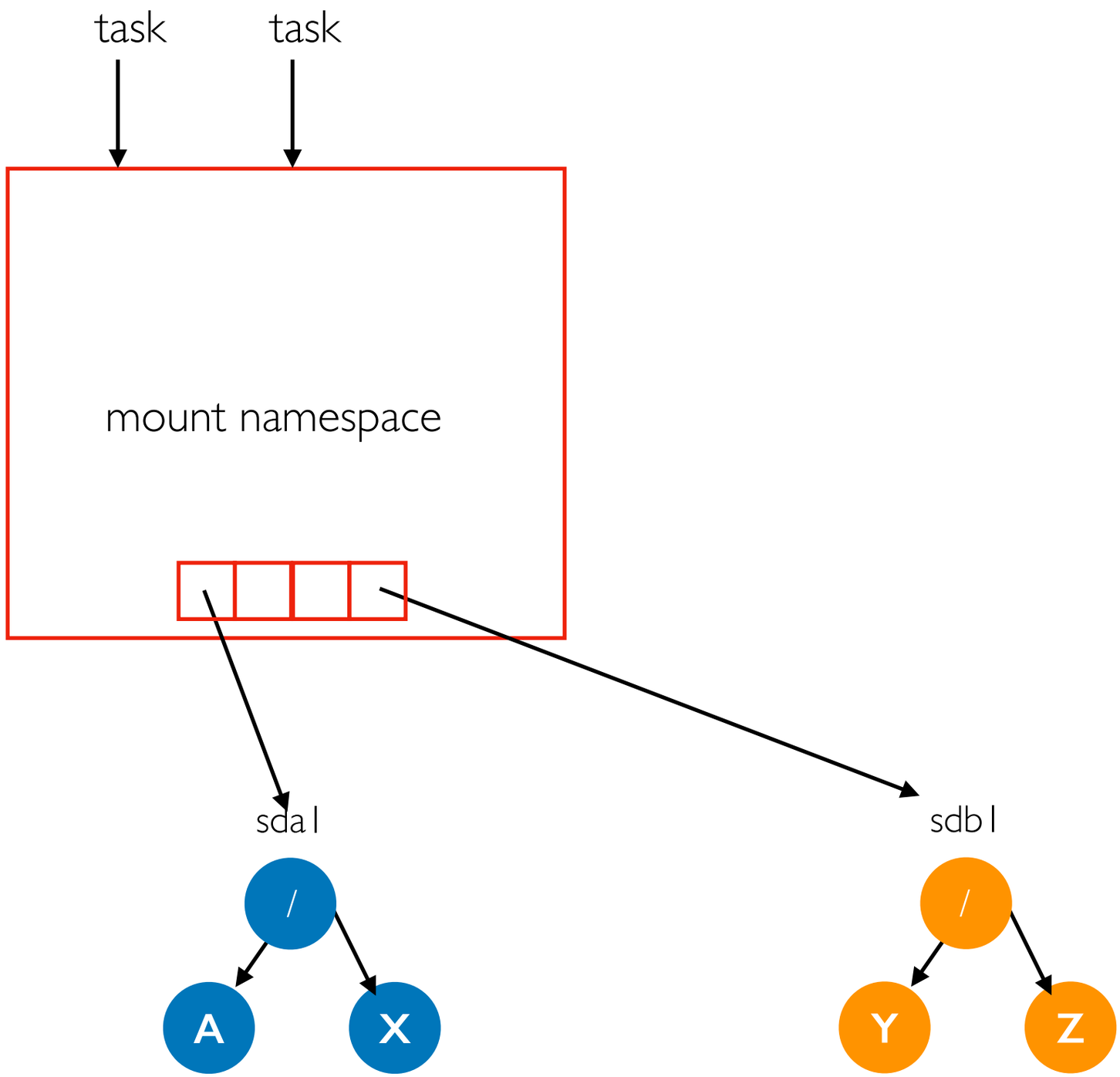
mount

- implementation
- systemd perf issue

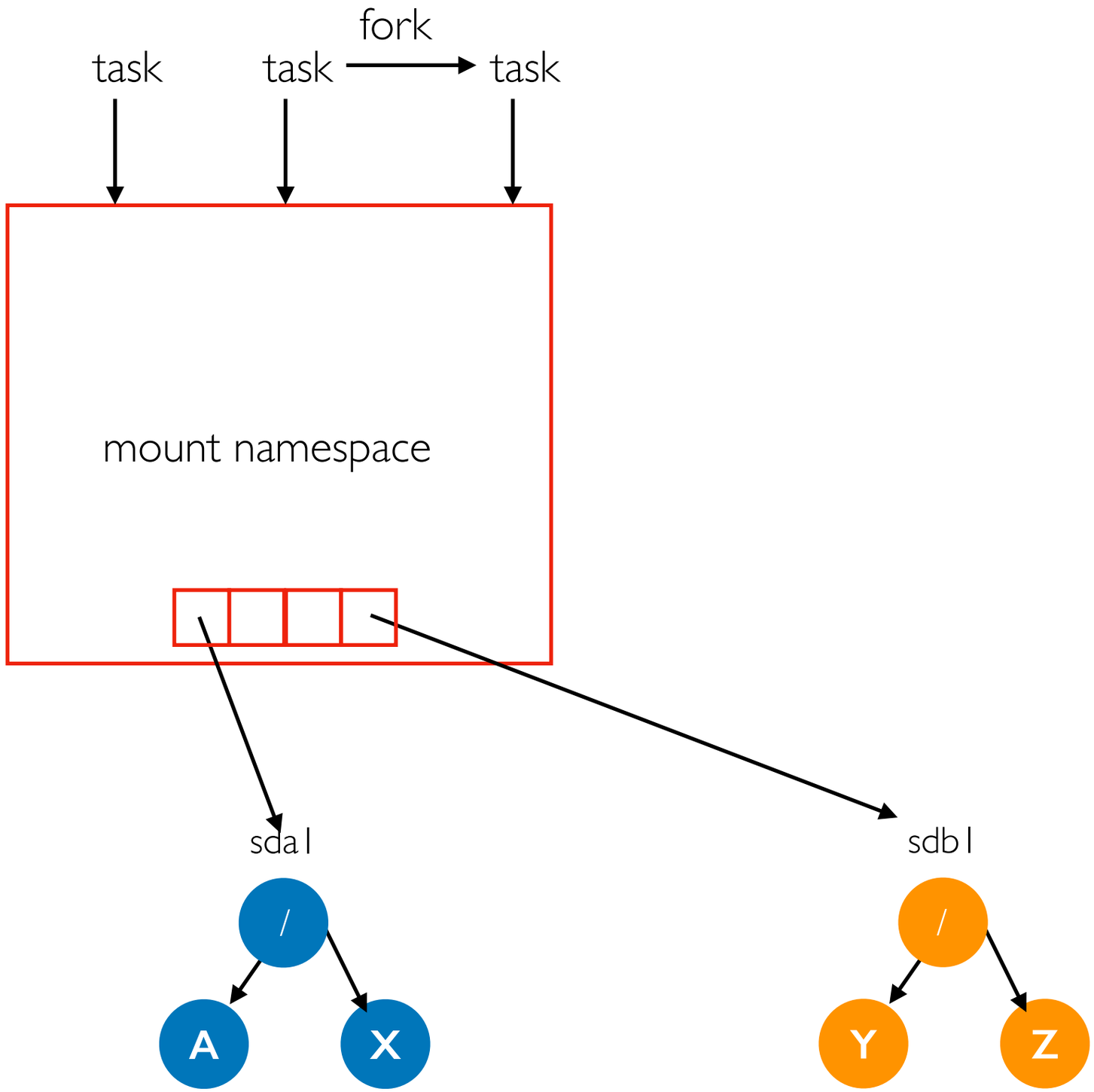
network

- perf issues: RCU
- one top-level one?

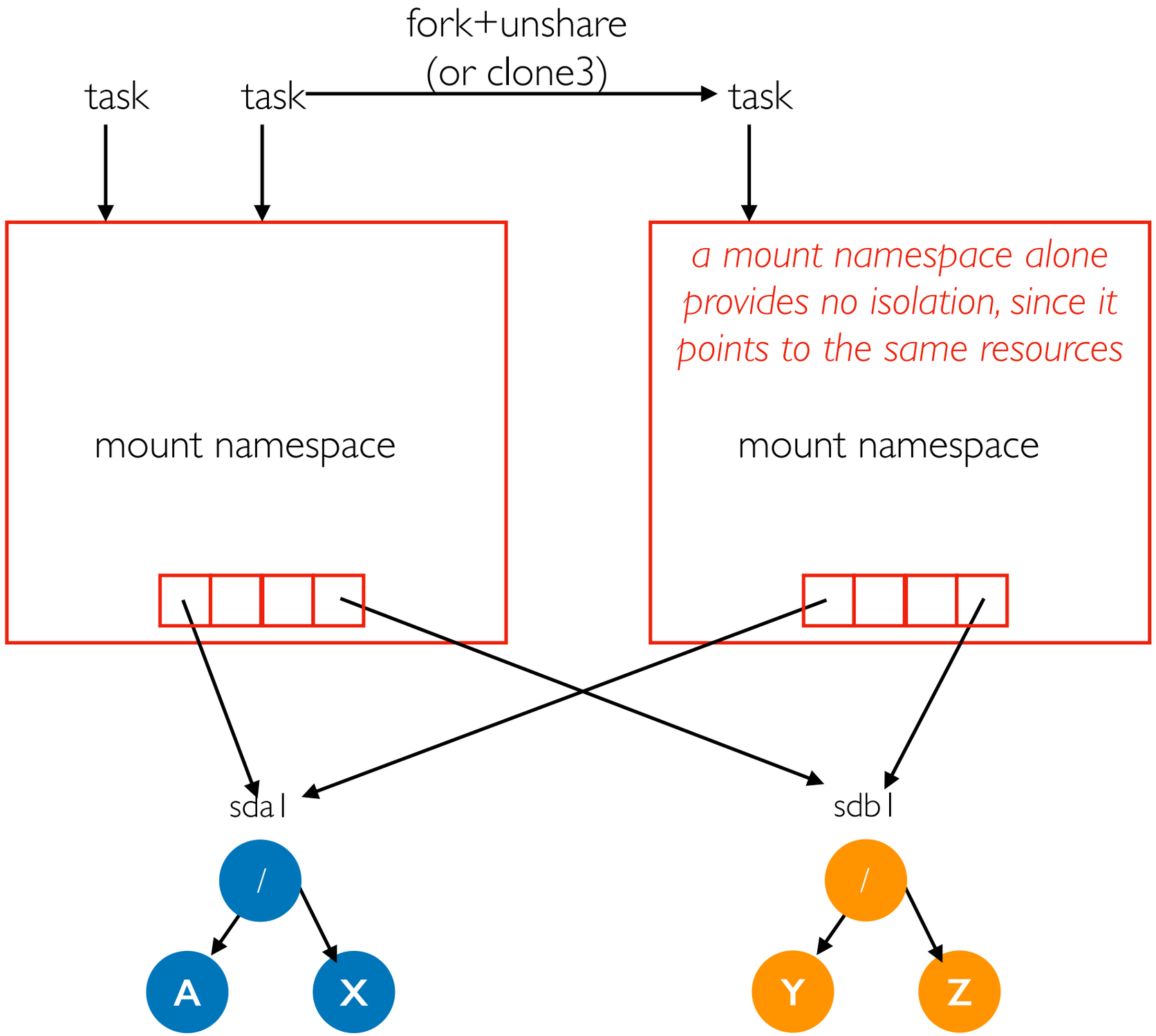
Mount Namespaces



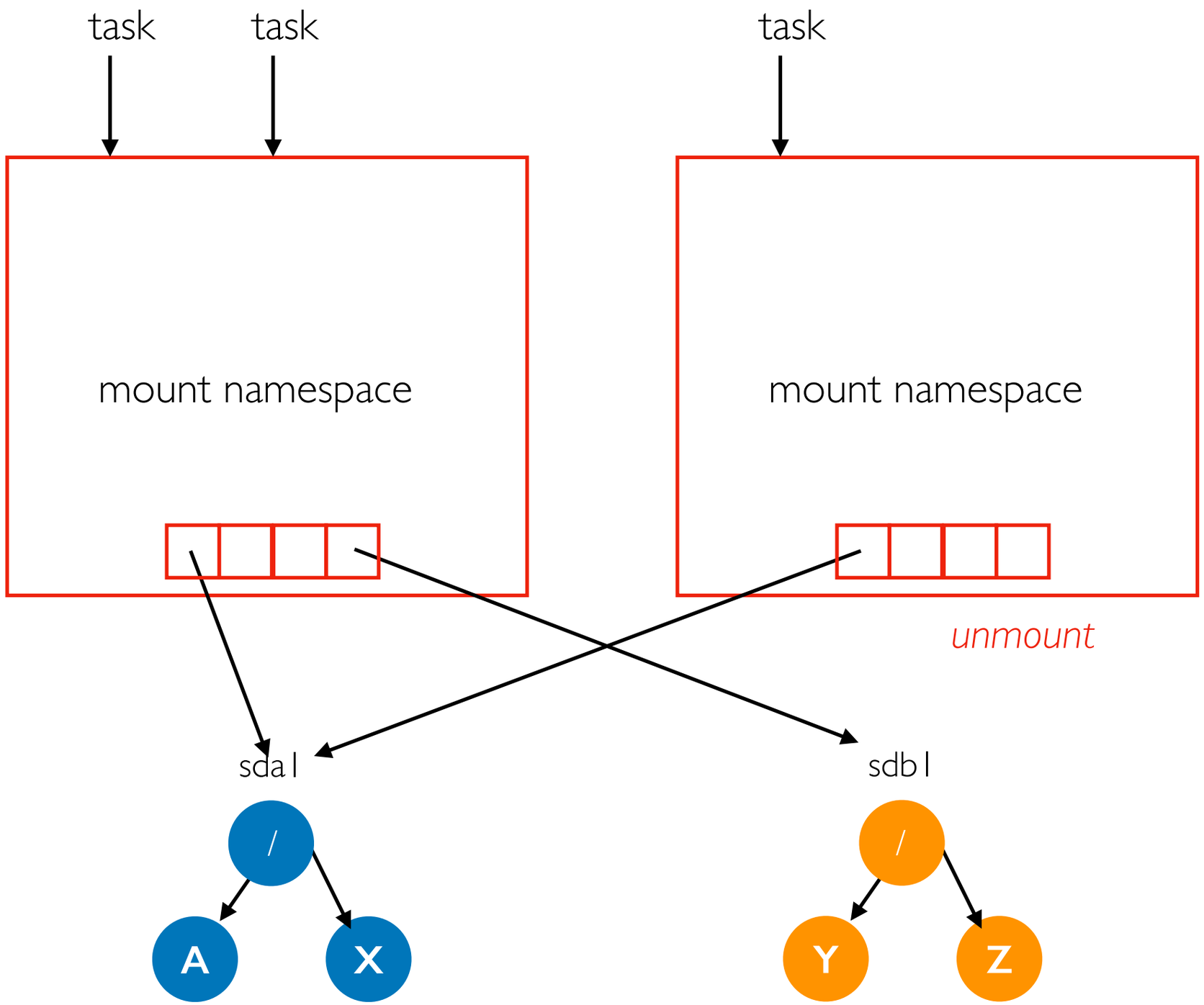
Mount Namespaces



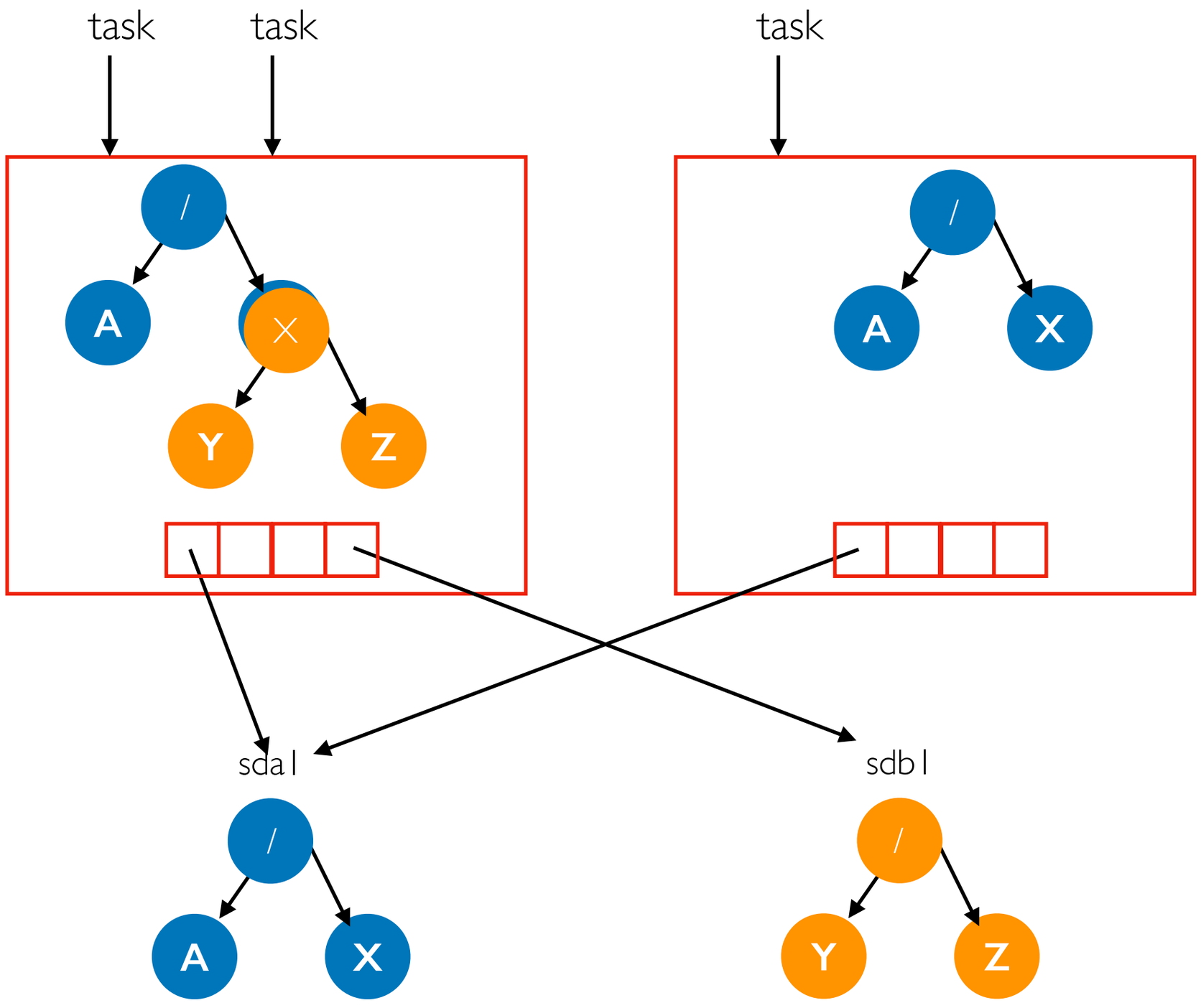
Mount Namespaces



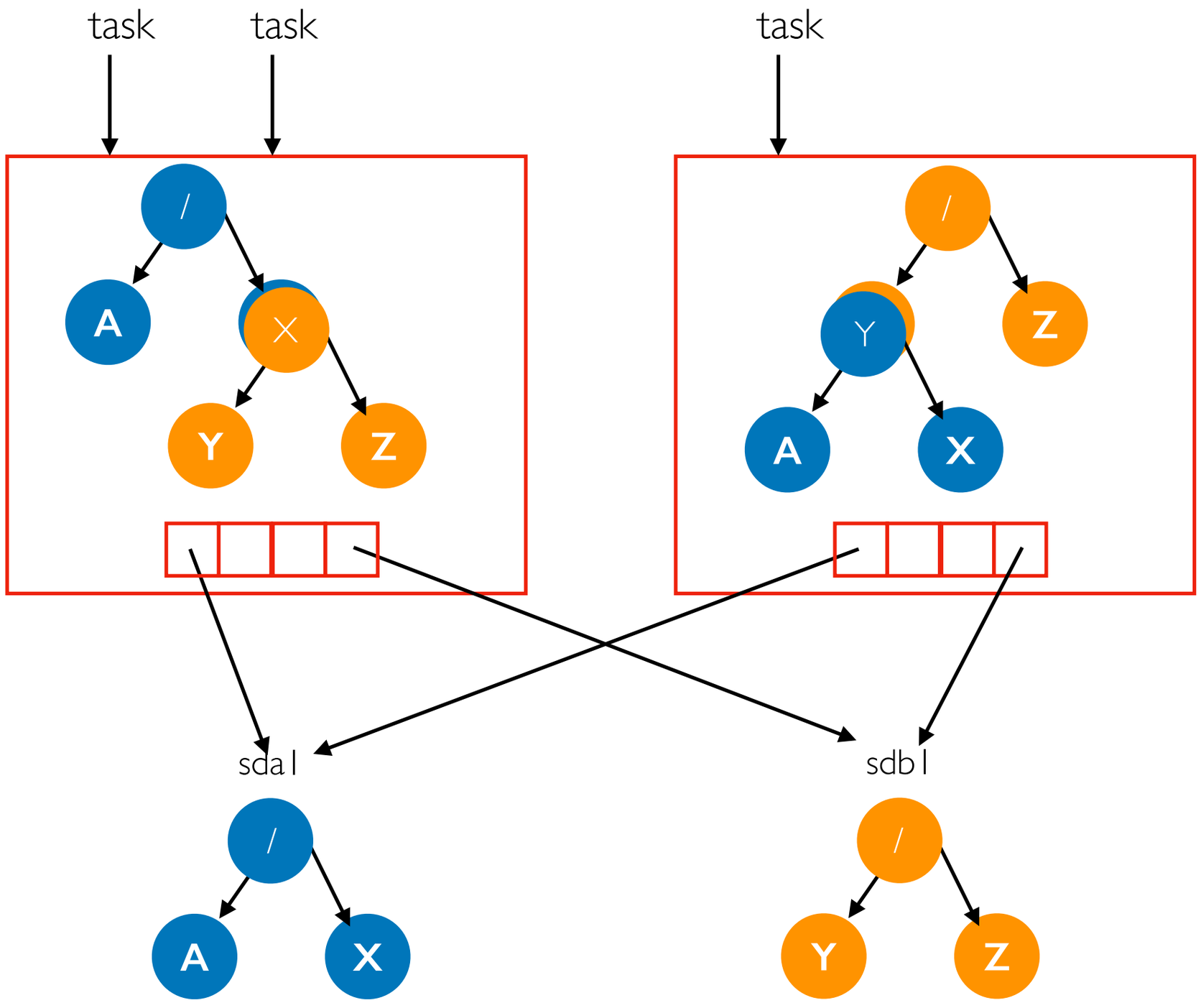
Mount Namespaces



Multiple Unified File Systems



Multiple Arrangements of the Same FS's are possible



Mount Namespaces: Usage

Docker Usage

- create a union file system with the main resources
- create a mount namespace
- make union FS the root, unmount other stuff
- mount other stuff on top of that (like procfs, tmpfs)
- chroot to that new directory

OpenLambda Usage

- minimal (originally no) mount NS usage
- create directory to use as root with some bind mounts
- chroot to it (OK to share a mount NS with others if you can only see a subtree)

systemd Performance Issue

Problem

- OpenLambda makes LOTs of changes to the mount namespace (bind mounts/ unmounts for every function)
- systemd started watching the primary mount namespace for changes and registering every mount as a "service"
- we create/delete containers so fast that systemd would be the bottleneck: 100% on two cores (not very parallel)

Solution: one new mount namespace for the work upon startup. All mounts/unmounts happen there for every container, but systemd can't see it. Only applies in OL "detached" mode.

systemd Performance Issue

Problem

- OpenL
  unmou
- systemc
  register
- we crea
  two co

on

Solution: one
happen there
mode.

s
ed"

```
if detach {
        // stdout+stderr both go to log
        logPath := filepath.Join(olPath, "worker.out")
        // creates a worker.out file
        f, err := os.Create(logPath)
        if err != nil {
                return err
        }
        // holds attributes that will be used when os.StartProcess.
        // we use CLONE_NEWNS because ol creates many mount points.
        // we don't want them to show up in /proc/self/mountinfo
        // for systemd because systemd creates a service for each
        // mount point, which is a major overhead.
        attr := os.ProcAttr{
                Files: []*os.File{nil, f, f},
                Sys: &syscall.SysProcAttr{
                        Unshareflags: syscall.CLONE_NEWNS,
                },
        }
```

src/worker/commands.go

Outline

Definition (and Analogy to Virtual Memory)

Interface

- namespaces
- setns, unshare, clone3

PIDs

- parent, child, grandchild
- one process with multiple IDs

mount

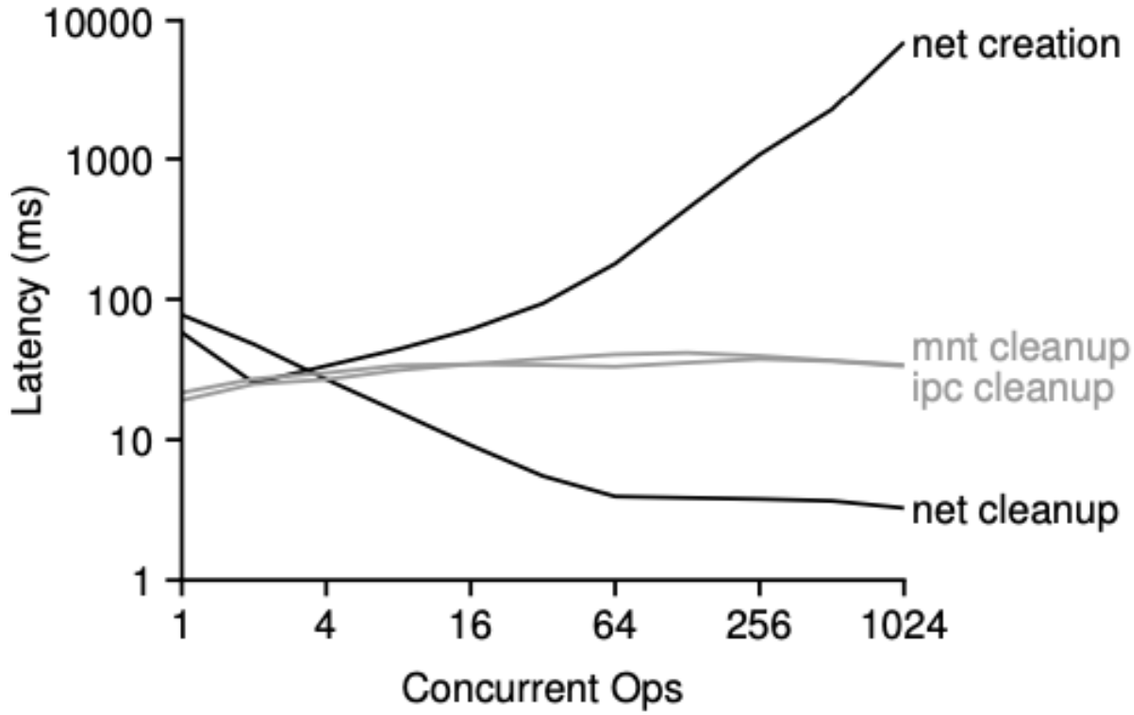
- implementation
- systemd perf issue

network

- perf issues: RCU
- one top-level one?

Network Namespaces

Network namespaces are SUPER slow.



https://www.usenix.org/system/files/conference/atc18/atc18-oakes.pdf

obverations

- single global lock
- RCU primitives used (optimized for reads, but for lambda workloads we care about writes
- cleanup is the real bottleneck, but it is async, so cost shows up during creation
- need to loop over every namespace (with lock held) when adding/removing a namespace
- hard to fix: there have been patches, but none merged (last I'm aware)

Network Namespaces: Usage

Docker Usage

- different namespaces can have different virtual interfaces available
- this lets different containers both use virtual port 80, for example

OpenLambda Usage

- no usage: too slow, and lambdas shouldn't be binding to specific ports anyway
- would it be useful to have ONE network namespace though, per worker (like we have one mount namespace)? This could let us run all lambdas using a network interface separated from others (that might be used for secure admin control).