

# Virtualization, micro-kernel-based OS

Samuel Thibault <[samuel.thibault@u-bordeaux.fr](mailto:samuel.thibault@u-bordeaux.fr)>  
<https://dept-info.labri.fr/~thibault/enseignements#SecuSys>

# Goals

- One physical machine
  - Host
- Several services
  - Guests
- Lowering cost
- Management flexibility
- Isolation
  - More security?
    - Some people have given up trying to fix syscall security



# Nothing new

- S/370 was emulating several S/360 in parallel (70')
- Unix process
- Filesystem with permissions

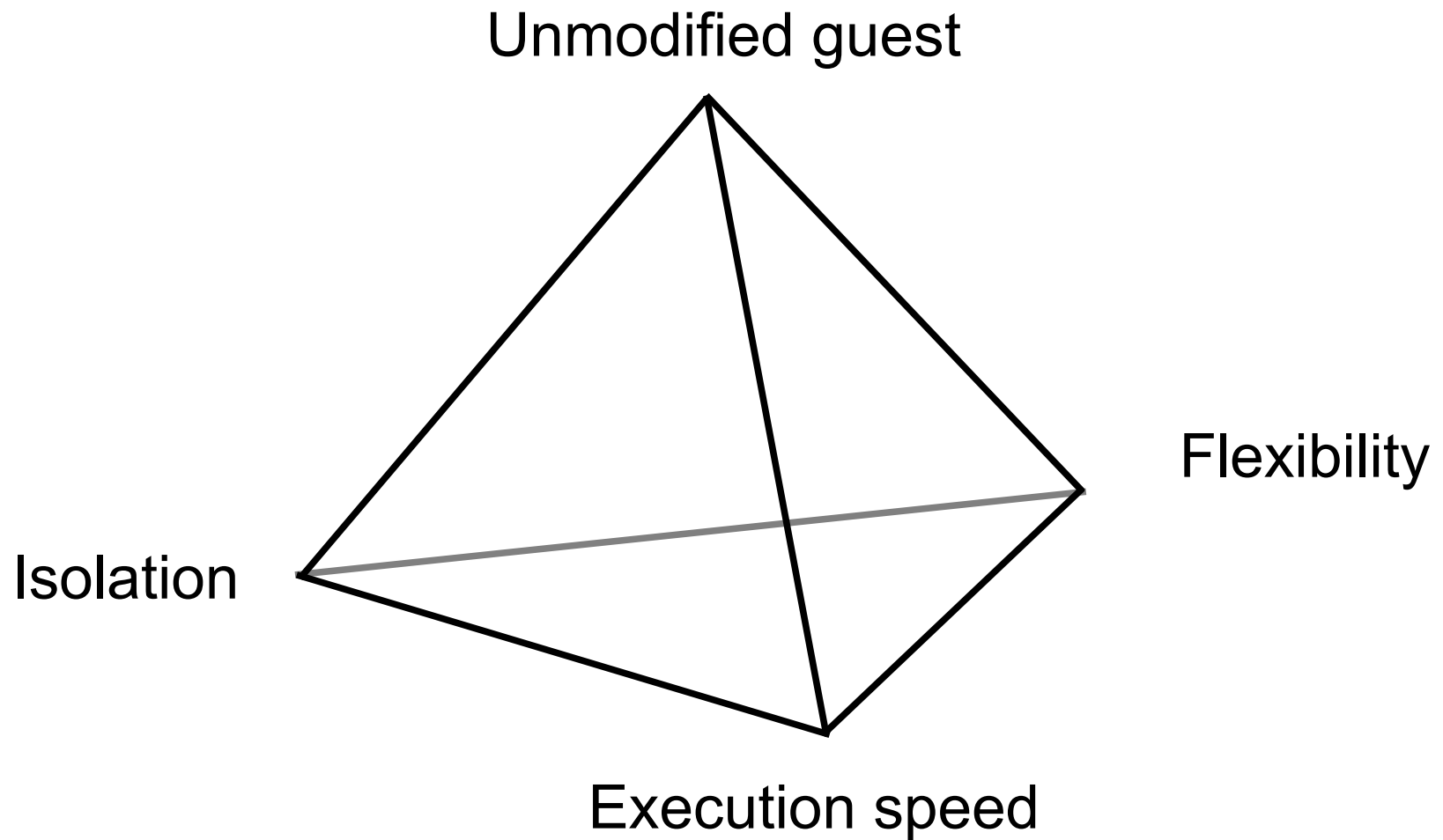
The question is the interface

- A PC
- A simplified PC
- hypercalls
- system calls

# Tons of solutions

- Virtualbox
- KVM/qemu
- Xen
- VMWare
- chroot
- LXC
- Parallels
- OpenVZ
- UML
- ...

# Contradictory qualities



# Unmodified guest?

## Modifying the kernel to adapt to virtual interface

- Xen: Executing in ring 1 instead of ring 0
  - No privileged instruction
  - Make hypercalls to manage page table etc.
- UML: Executing in a process
  - Make system call to manage memory etc.

## Special drivers for acceleration

- Virtio
- Hyper-V

Nowadays, integrated in standard distributions

# Flexibility?

## Memory

- Tuning : automatic / static / on the fly
- Avoid duplicate memory pages

## Disk

- Tuning : automatic / static / on the fly
- Adding disk on the fly

## Network

- On the fly

## Exchanging data

- Mount shared filesystem

## Accounting, quotas

# Isolation

## Do guests

- see files of other guests?
- see each other?
- are on the same Ethernet network?
- use the same kernel?
  - beware of rogue modules and crashes



# Execution speed

## CPU speed not really a problem

- Unless using total virtualization without hardware acceleration

## Memory management

- Creating many processes
- Switching between processes

## Disk, network

- Bandwidth
- Latency

**3 main classes**

# 3 main classes

## Total virtualization

- KVM/qemu, Xen HVM, VirtualBox, VMware, ...

## Para-virtualization

- Xen PV, Xen PV-HVM, Hyper-V, VMware drivers, virtio

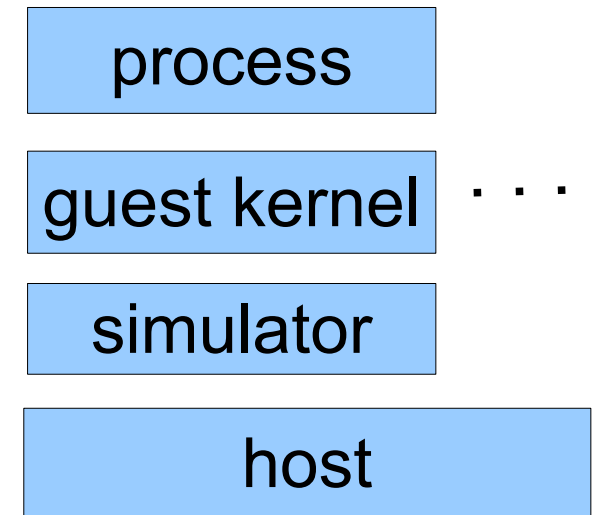
## Containers

- Process, chroot, cgroups, openVZ, LXC, ...

# Total virtualization

KVM/qemu, Xen HVM, VirtualBox, VMware

- Simulate a complete PC
  - BIOS, floppy, ...
  - Or not : qboot
- Complex simulation
  - Many CVEs
- Accelerated by hardware support
  - Intel (vmx, VT-x, VT-i), AMD (svm), NPT
- A must for Windows etc.



Wake-up, neo...

- Blue pill...

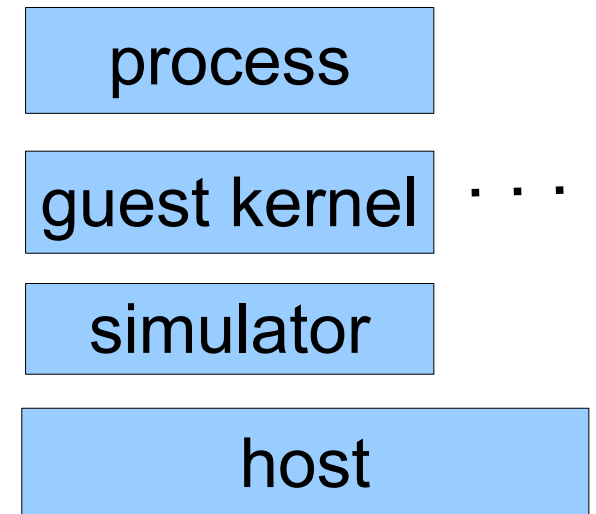
WSL2 (Windows Subsystem for Linux)

- Runs a real Linux kernel, thus perfect compatibility

# Total virtualization summary

KVM/qemu, Xen HVM, VirtualBox, VMware

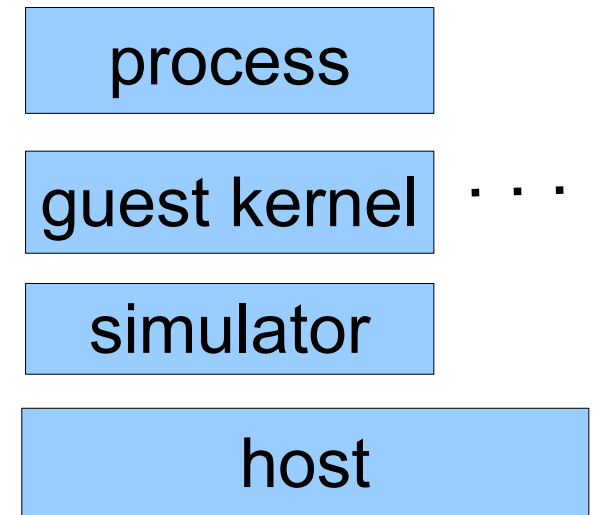
- Guest is not modified
- Not really flexible
  - Removing RAM banks?
- Very isolated
  - But beware of simulation bugs
    - Use `stubdom-dm`
- Reasonable speed
  - Thanks to hardware acceleration



# Non-stubdom simulation

KVM/qemu, Xen HVM, VirtualBox, VMware

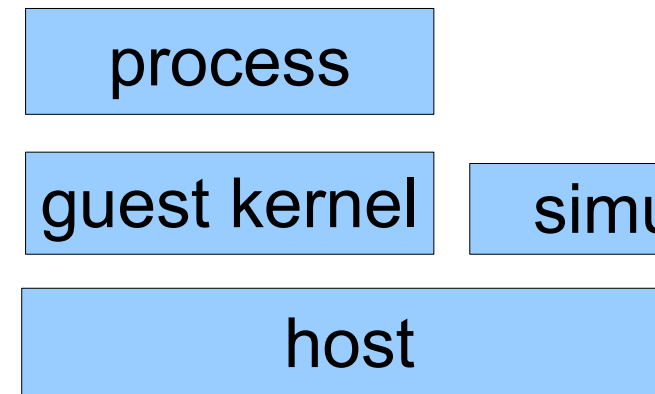
- Simulation of hardware is running inside the host
- Often with root privileges!
- Better isolate that



# Stubdom simulation

## Xen HVM with stubdom

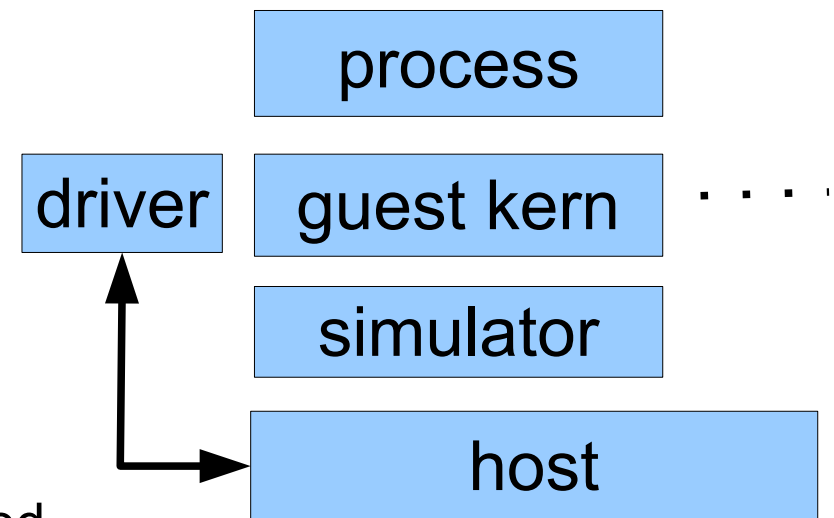
- Running hardware simulator in a separate guest
  - Limit what it can do
  - Isolate just like the VM is supposed to be



# Para-Virtualization

## VMware drivers, virtio

- « VMware drivers »
- Often based on shared memory
  - packet exchange protocol
  - frontend/backend
- Speed, flexibility
- But then security concerns on the shared-memory protocol
  - e.g. virtio tries to factorize it
  - POSIX syscall interface did get proposed...

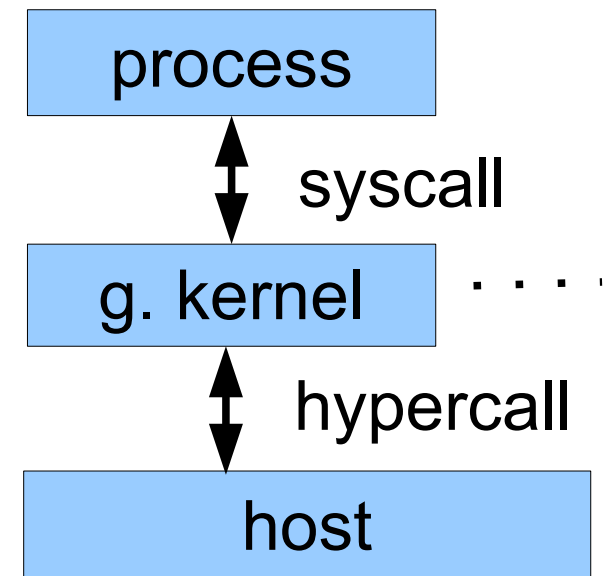




# Para-Virtualization

## Xen PV, Xen PV-HVM

- PV kernel
  - Really knows that it is virtualized
- « Hypercalls »
  - Essentially to replace privileged instructions
    - CPU control, virtual memory, ...
- PV drivers

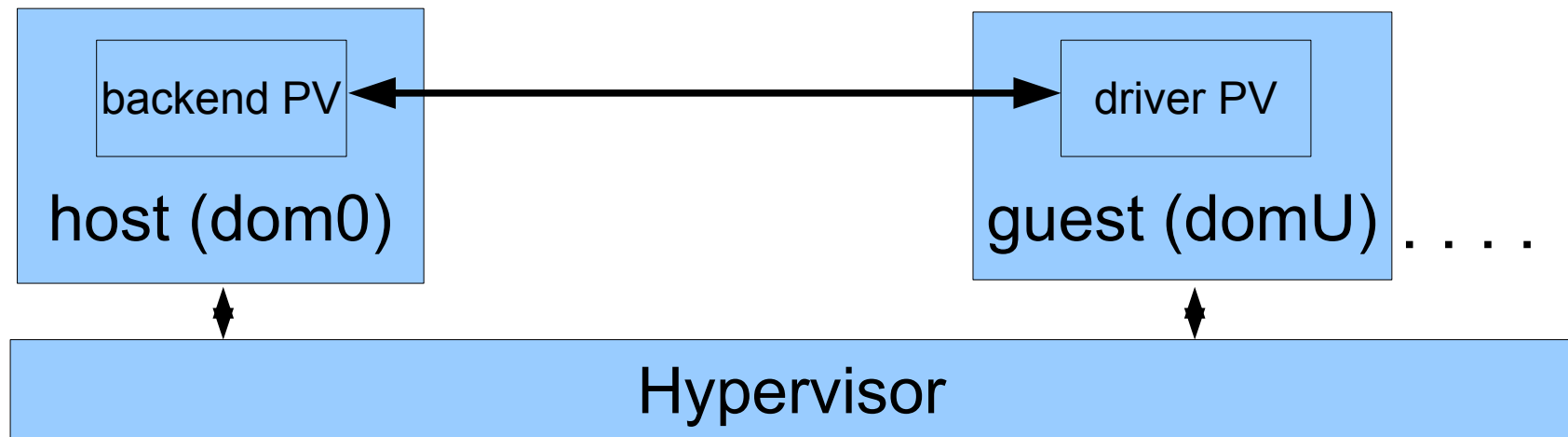


# Para-Virtualization

## Real Xen picture

### Virtualization type 1

- A hypervisor boots before the host
  - Much smaller than a full-featured kernel

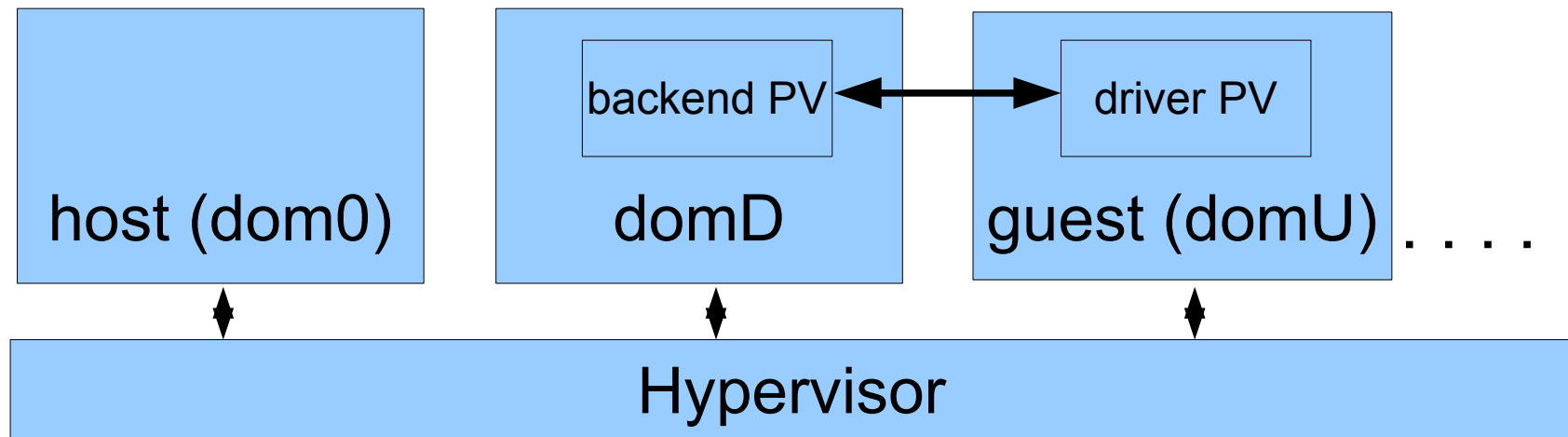


# Para-Virtualization

## Real Xen picture

### Virtualization type 1

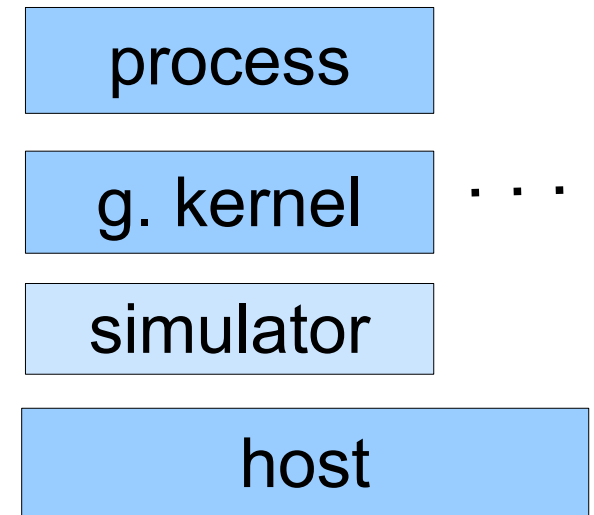
- A hypervisor boots before the host
  - Much smaller than a full-featured kernel
- Or even disaggregated



# Para-Virtualization summary

Xen PV, Xen PV-HVM, VMware drivers, virtio, UML

- **Guest is modified**
  - Largely (PV), or drivers
- **Very flexible**
  - Dynamic resizes
- **Very isolated**
  - Separate kernels
  - Interface is hypercalls
    - Meant exactly for this usage
- **Excellent speed**
  - That's meant for it !



# Containers

Process, chroot, cgroups/NS, openVZ, LXC

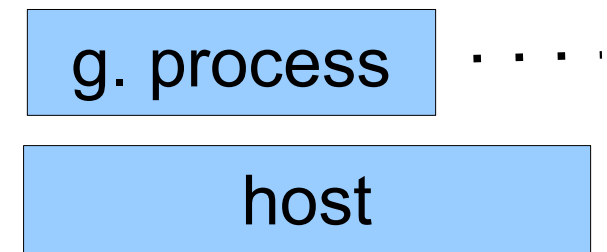
- Process++

Varying isolation

- Filesystem space
  - « private / »
- pid space
- network space
- ...
- What did we forget ?

WSL1

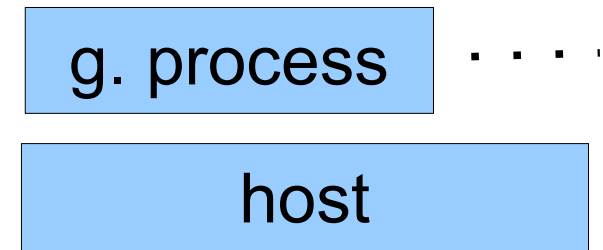
- Emulates the Linux system call interface



# Containers summary

Process, chroot, cgroups/NS, openVZ, LXC

- **Unmodified guest**
  - As in : it's just a normal process
- **Very flexible**
- **More or less isolated**
  - Depending on the support
  - Same kernel, incompatibility concerns (udev, systemd)
- **Perfect speed**
  - Identical to a process



Flexibility?

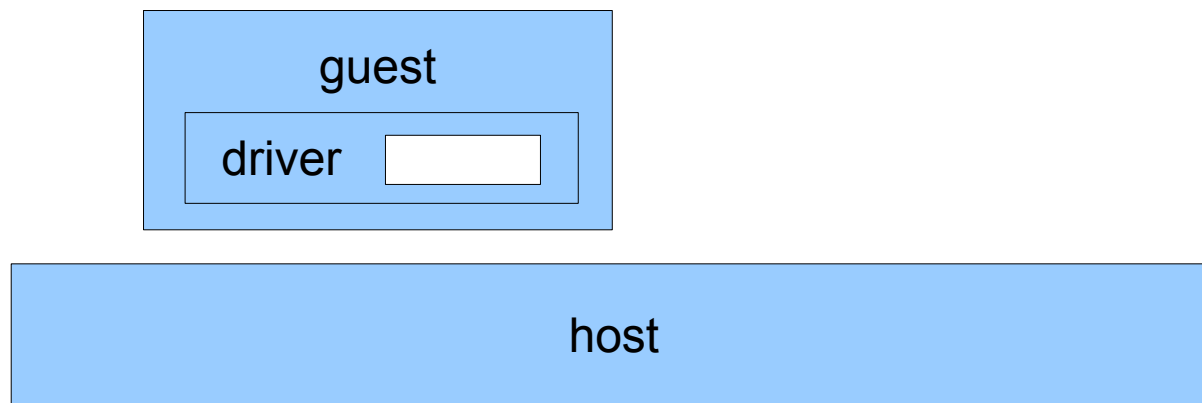
# Memory flexibility

## Adding memory

- Not « that » hard

## Removing memory

- Ew...
- Ballooning





# Disk flexibility

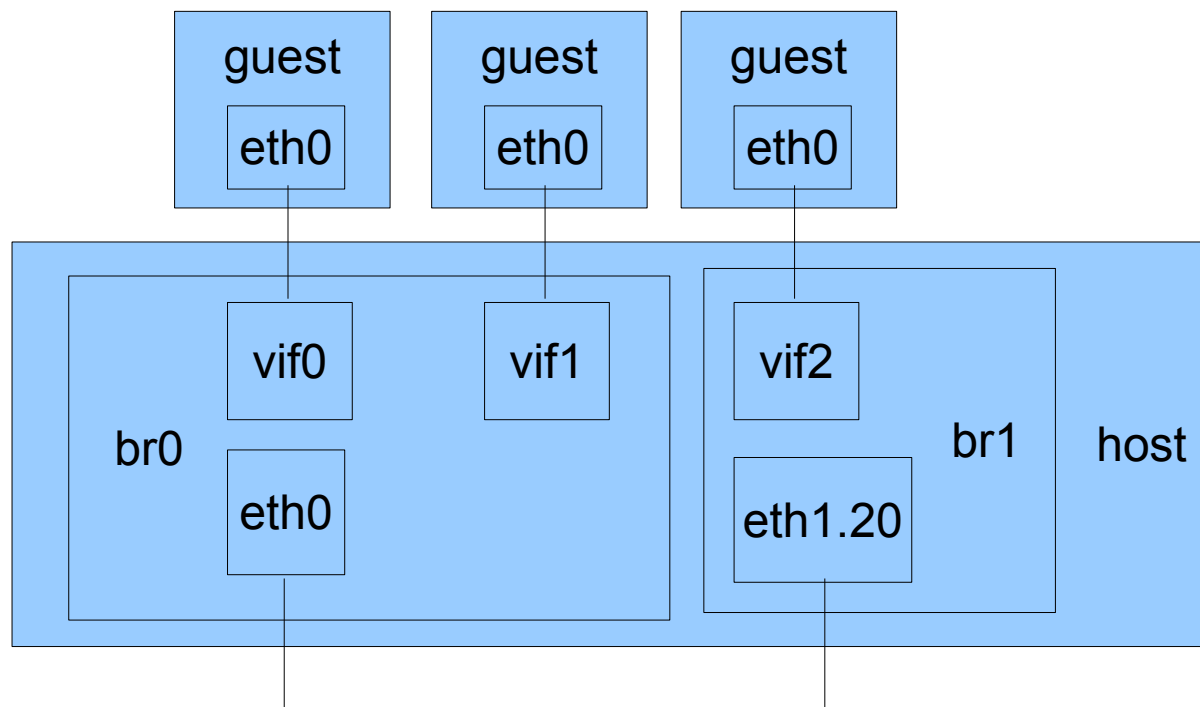
## Online resize

- Containers : ~= quotas
- Others :
  - Use lvm's lvresize on the host
  - Use resize2fs inside the guest
  - Seamless!

# Network flexibility

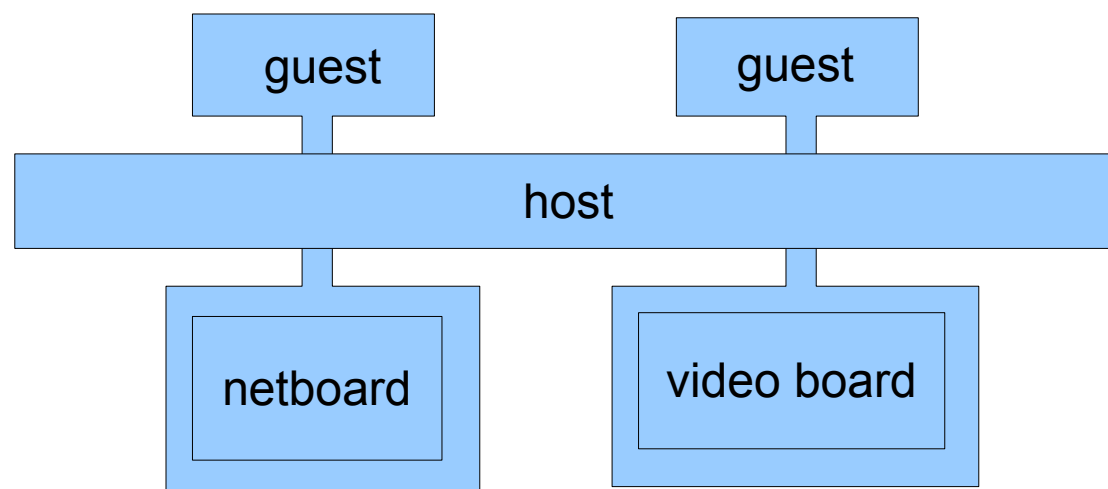
## Virtual plugging

- Bridge  $\sim$  network switch
- Usual host management (bridge, VLAN, ...)
  - Or openvswitch



# Hardware flexibility

- PCI passthrough
- USB passthrough
- ! Security ! (DMA, ...)
  - VT-d support (IOMMU)



# Virtualization conclusion

All solutions try to optimize for the 4 qualities

- More or less successful
- Keep evolving
  - Don't trust what people say
- Check what you really want

# Micro-kernel-based OS

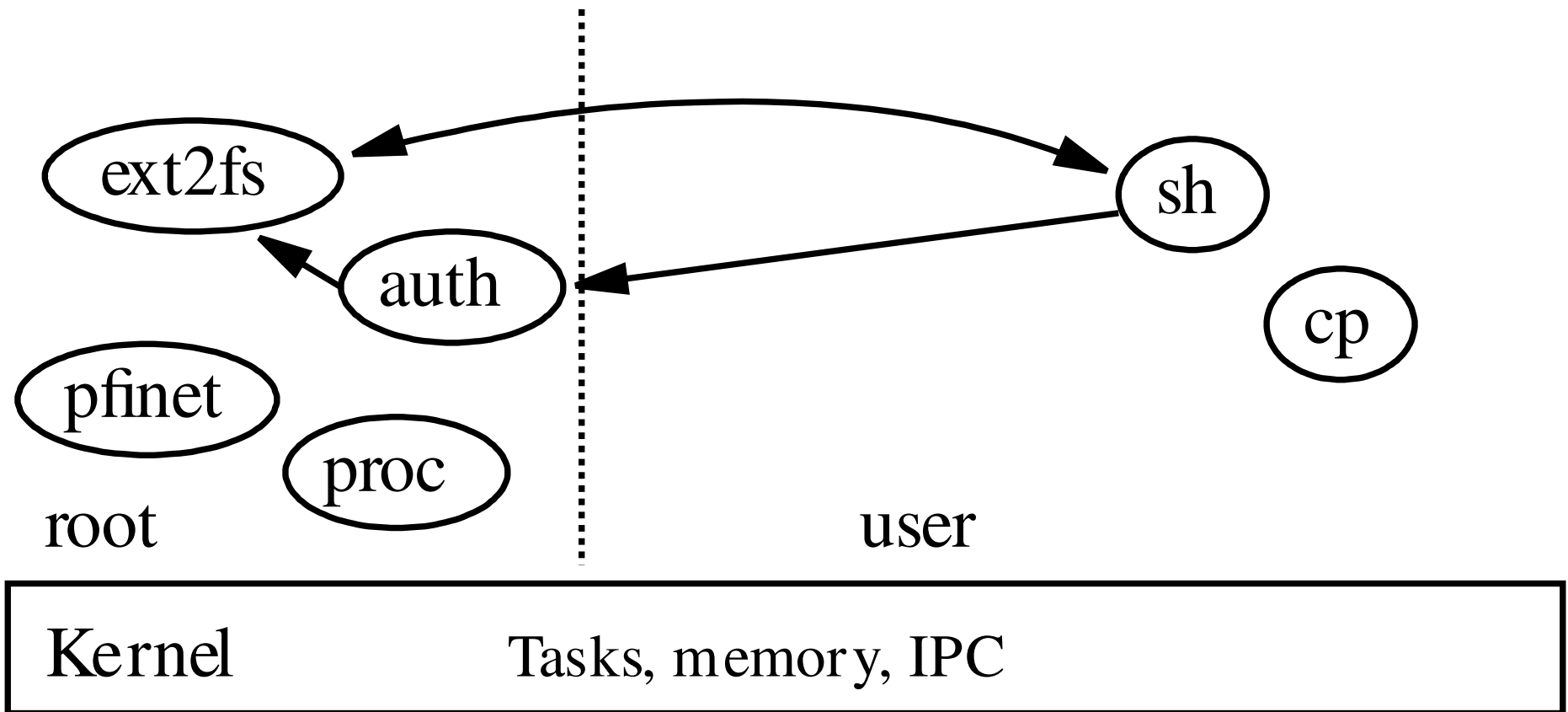
# Micro-kernel-based OS

- Micro kernel containing only the few required features
  - Task switching
  - Memory management
  - Inter-Process communication (IPC, RPC)
- Rather than a big kernel containing all system calls

Minix, LynxOS, L4, GNU/Hurd

Will here discuss GNU/Hurd

# Micro-kernel layering



# Micro-kernel layering

read() is not a system call

- RPC to whatever server backs the opened file

Ditto for all POSIX functions

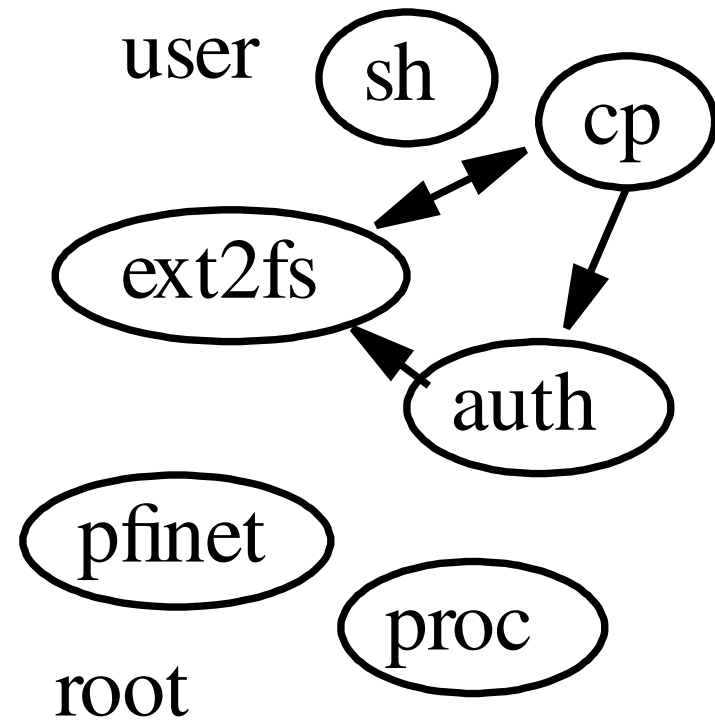
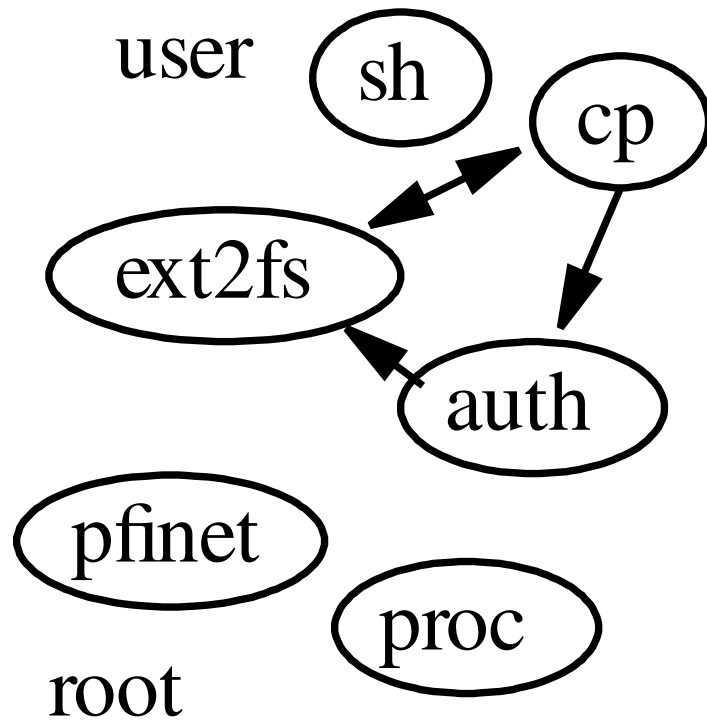
- Underlying RPC calls

→ **Natively** isolated

- Server crash? Not a problem
  - « computer bought the farm » is just an error, not something-of-the-death
- Driver isolated in its own process
  - Overflows limited to itself
- Driver hung? Just kill it and let it restart
- Easier to debug/tune
  - Just run gdb, gprof, ...
- Can virtualize at a very fine grain

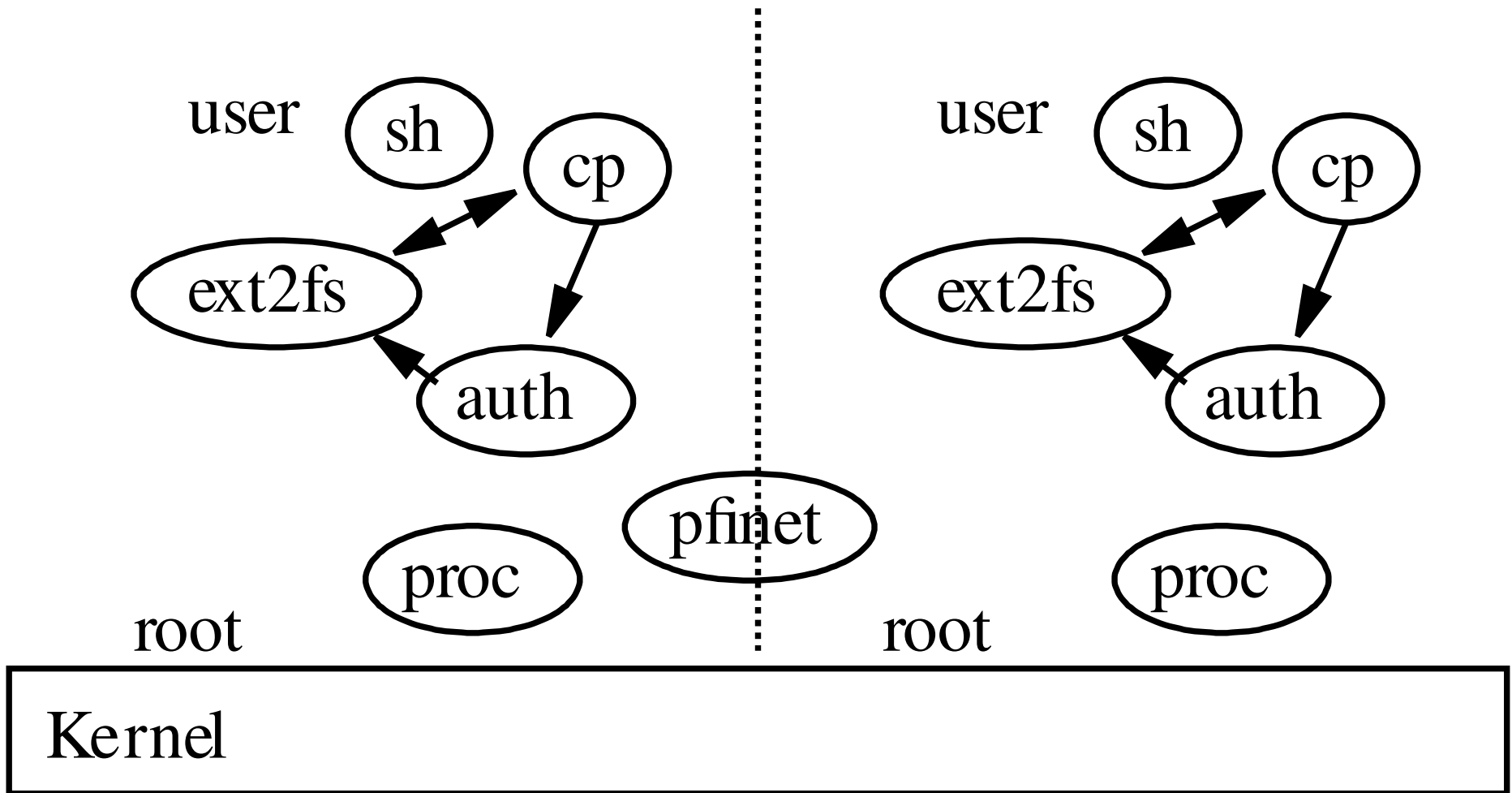


# Neighbour Hurds

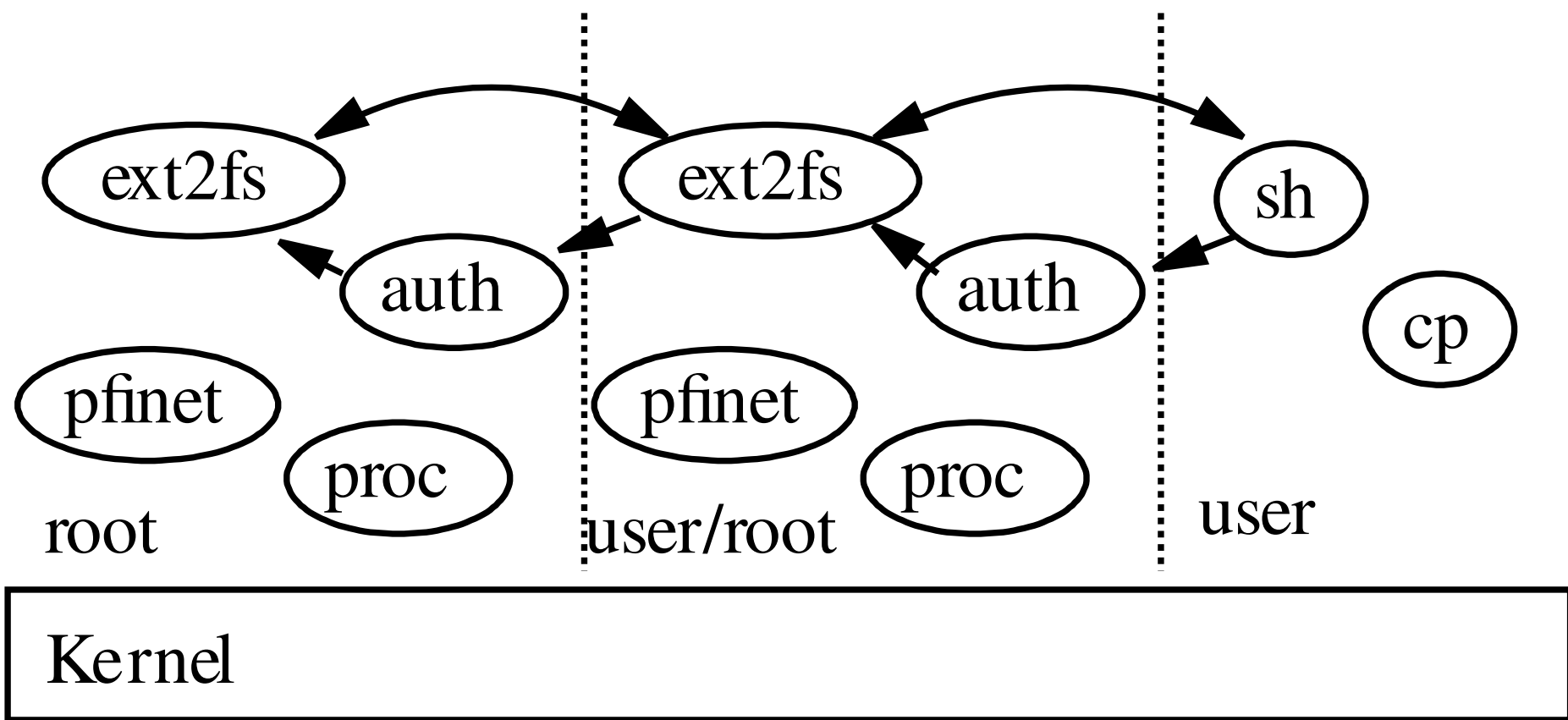


Kernel

# Neighbour Hurds, shared pfinet



# Sub-Hurd



# Micro-kernel-based systems


## Strong isolation

- *By default*
  - No risk of forgetting something


## Didn't expand in the industry

- But virtualization/containerization looks more and more like it

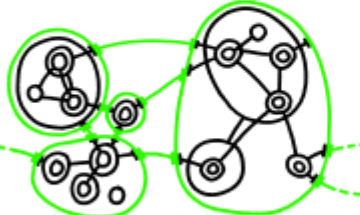
"I WISH THESE PARTS  
COULD COMMUNICATE  
MORE EASILY."



"OOH, THIS NEW TECHNOLOGY  
MAKES IT EASY TO CREATE  
ARBITRARY CONNECTIONS,  
INTEGRATING EVERYTHING!"



"OOH, THIS NEW TECHNOLOGY  
MAKES IT EASY TO ENCLOSE  
ARBITRARY THINGS IN  
SECURE SANDBOXES!"



"UH-OH, THERE ARE  
SO MANY CONNECTIONS  
IT'S CREATING BUGS  
AND SECURITY HOLES!"

