The new VVorld

FreeBSD jail based virtualization

Bjoern Zeeb, Robert Watson

The FreeBSD Project

BSDCan 2010

Overview

1. Virtualization
2. Virtualization at OS level
3. Virtual Network Stack
4. Use cases and numbers

What is virtualization

- Term more and more popular during the last decade.
- Have one real something but pretend to provide several.
- Q: Why doesn't it work with apples or oranges?
- Q: How does IT solve it?

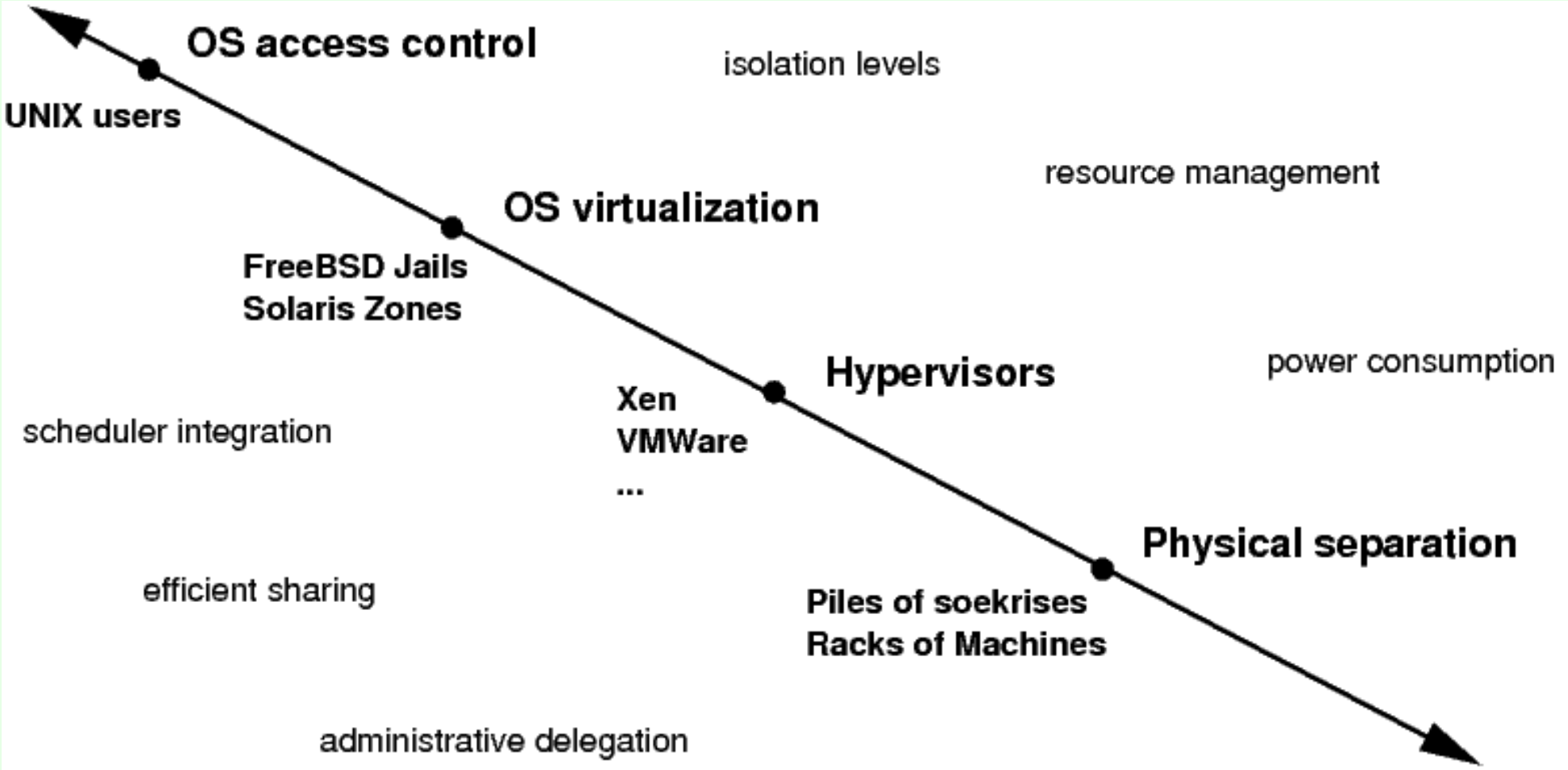
What is virtualization, samples

- In networking: VLANs, MPLS VPNs, ...
- For machines: 1 piece of hardware, many virtual instances.
- In operating systems: VM address spaces, ...
- In FreeBSD: Jails, as well as Xen, Virtualbox, ...
- A: Add a level of indirection.

Why virtualize?

- Use resources more efficiently.
- Make people believe they get it all but still share.
- Allow for (managed) overcommits.
- Consolidate resources.
- Delegation of administration.
- Build systems with better security footprints.

Virtualization spectrum



History of Jails

- April 1999: commit of the “mumbled about Jail feature”.
- May 1999: VMware Workstation 1.0
- 2002 M. Zec *BSD network stack virtualization*, BSDCon Europe
- 2007 Jail friendly file systems (ZFS).
- 2008 multi-IPv4/v6/no-IP patches.
- 2008 integration of virtual network stack starts.
- 2009 hierachical jails.

What are “Jails”?

- Operating system-level virtualization.
- One kernel.
- Chroot, IP address subsetting, . . .
- Safe super user delegation with restrictions.
- Efficient resoures sharing, allows overcommits.
- Very lightweight (no hypervisor, no virtual device overhead)
- Nowadays: multi IPv4/v6/no-IP jails, cpuset support, jail-friendly filesystems (ZFS).

Jails - what improvements?

- lightweight, secure, fast, simple (keep this).
- Does not depend on hardware support (keep it like this).
- ”ping does not work”, no loopback, . . . (improve).
- System V IPC troubles (improve).
- Why is that? Jails “subset” rather than “virtualize”.
- Cannot run “unnamed commercial OSes” (not going to happen).

What to virtualize?

- Start with network stack.
  - Immediate demand as just an address wasn't good enough anymore.
  - Marko Zec 2002 Prototype.
  - Make sure performance does not change noticably.
- Future: VIPC, ...

Subsystem virtualization - what to take care of?

- Global variables.
- Callouts (timers).
- Eventhandlers.
- Sysctl MIBs.
- Startup and shutdown.
- Debugging.

How to handle things?

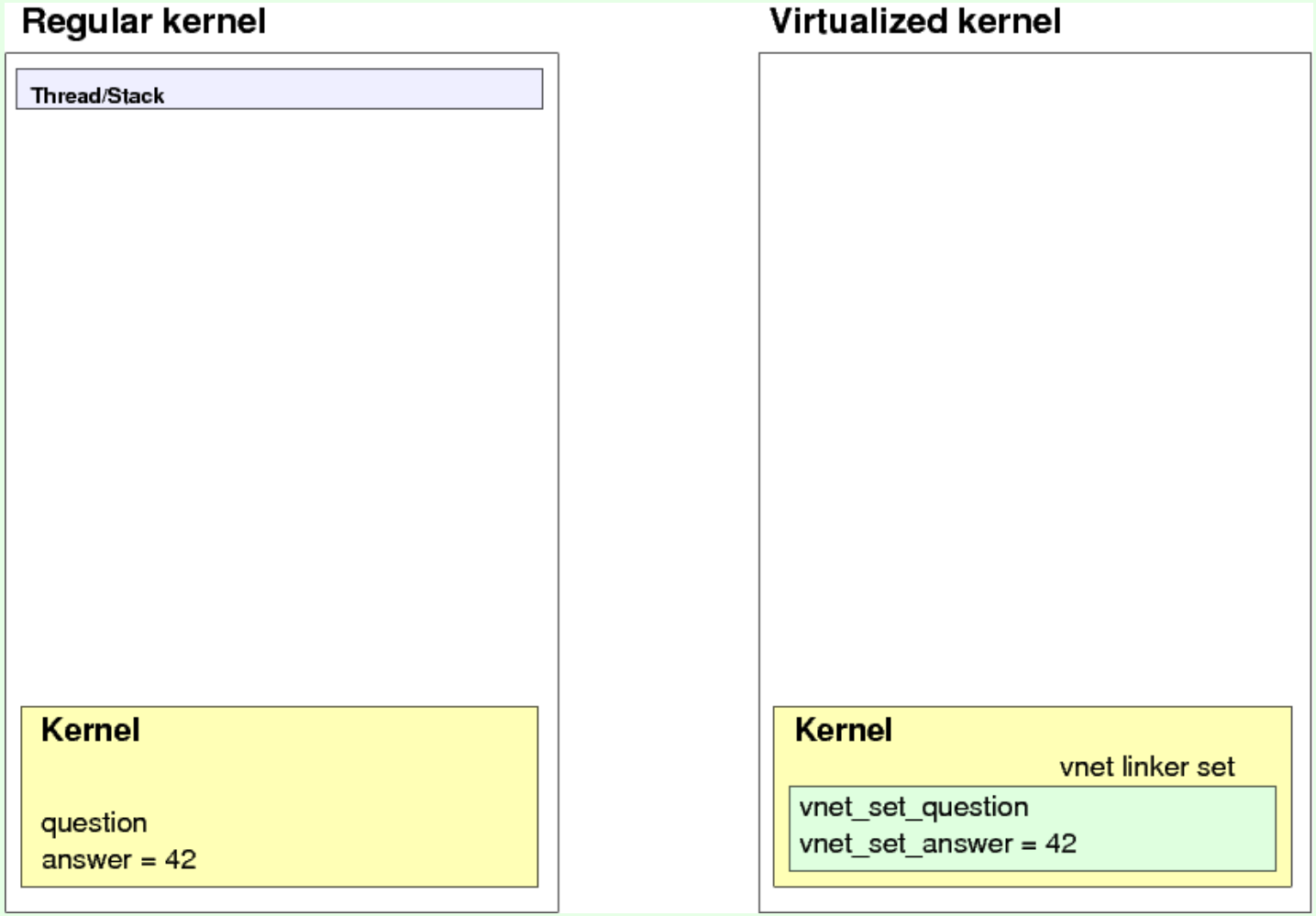
- Have “virtual instances” as abstraction level.
- Replicate global variables per instance.
- Duplicate or multiplex eventhandlers/timers per instance.
- Mark objects as part of an instance.
- Keep a clear security concept - do not grant insecure privileges.
- Be prepared for inter-instance interaction.
- Starting and Stopping?

Step by step walkthrough

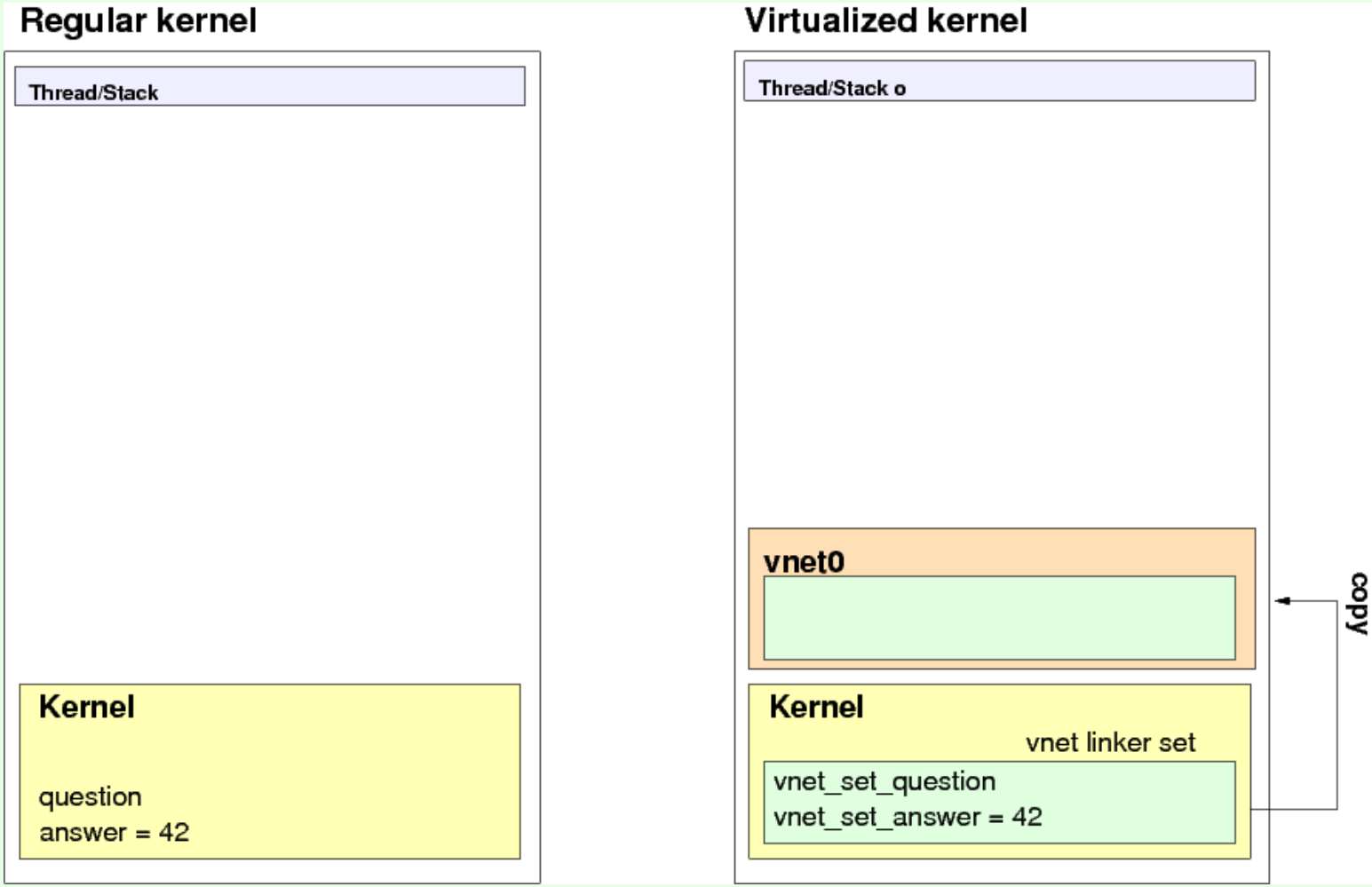
- Step by step walkthrough - how virtualization works.

Standard kernel

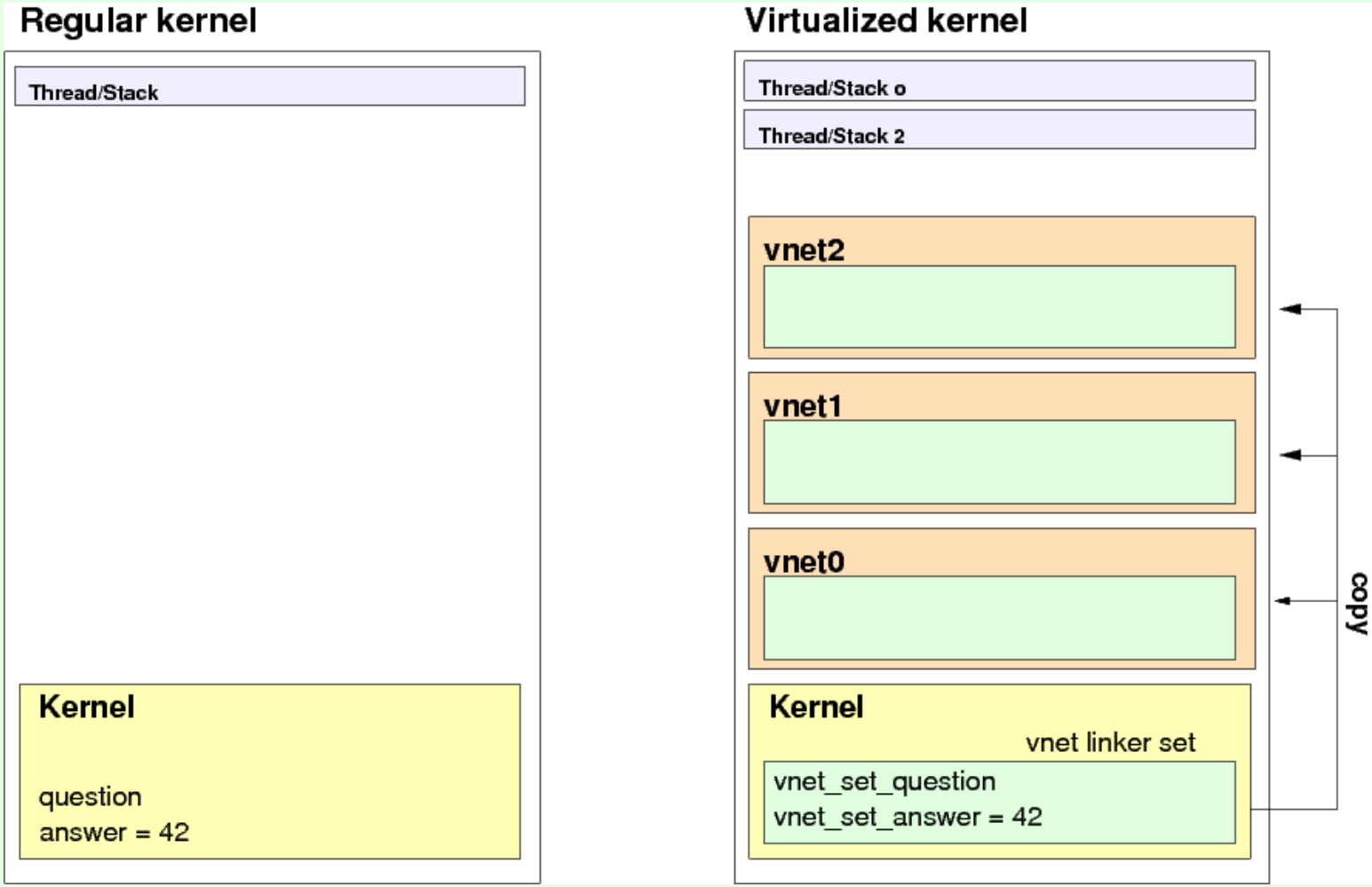
Virtualized kernel



The base system network



Start more instances, procs/threads not virtualized



Globals

Unvirtualized globals:

```
static int question;
int answer = 42;
```

Now virtualize this:

```
static VNET_DEFINE(int, question);
static VNET_DEFINE(int, answer) = 42;
#define V_question  VNET(question);
#define V_answer    VNET(answer);
```

Sysctls, System initialization

```
SYSCTL_VNET_INT(_hhgttg, OID_AUTO, question,
    CTLFLAG_RW, &VNET_NAME(question), 0, "Q:");
SYSCTL_VNET_INT(_hhgttg, OID_AUTO, answer,
    CTLFLAG_RW, &VNET_NAME(answer), 0, "A:");

static void
hhgttg_init(void __unused)
{
        if (V_question)
                goto_page(V_answer);
}
VNET_SYSINIT(hhgttg_init, SI_SUB_PSEUDO,
    SI_ORDER_FIRST, hhgttg_init, NULL);
```

Virtual Network Stack

- own loopback interface
- virtual iterfaces or netgraph subsystem to get connectivity to network stacks
- can give dedcicated hardware resources (NIC) to a virtual network stack
- own statistics
- own IPsec, firewalls, . . .
- own everything network stacky basically
- use forwarding or bridging between instances or outer world
- still very lightweight

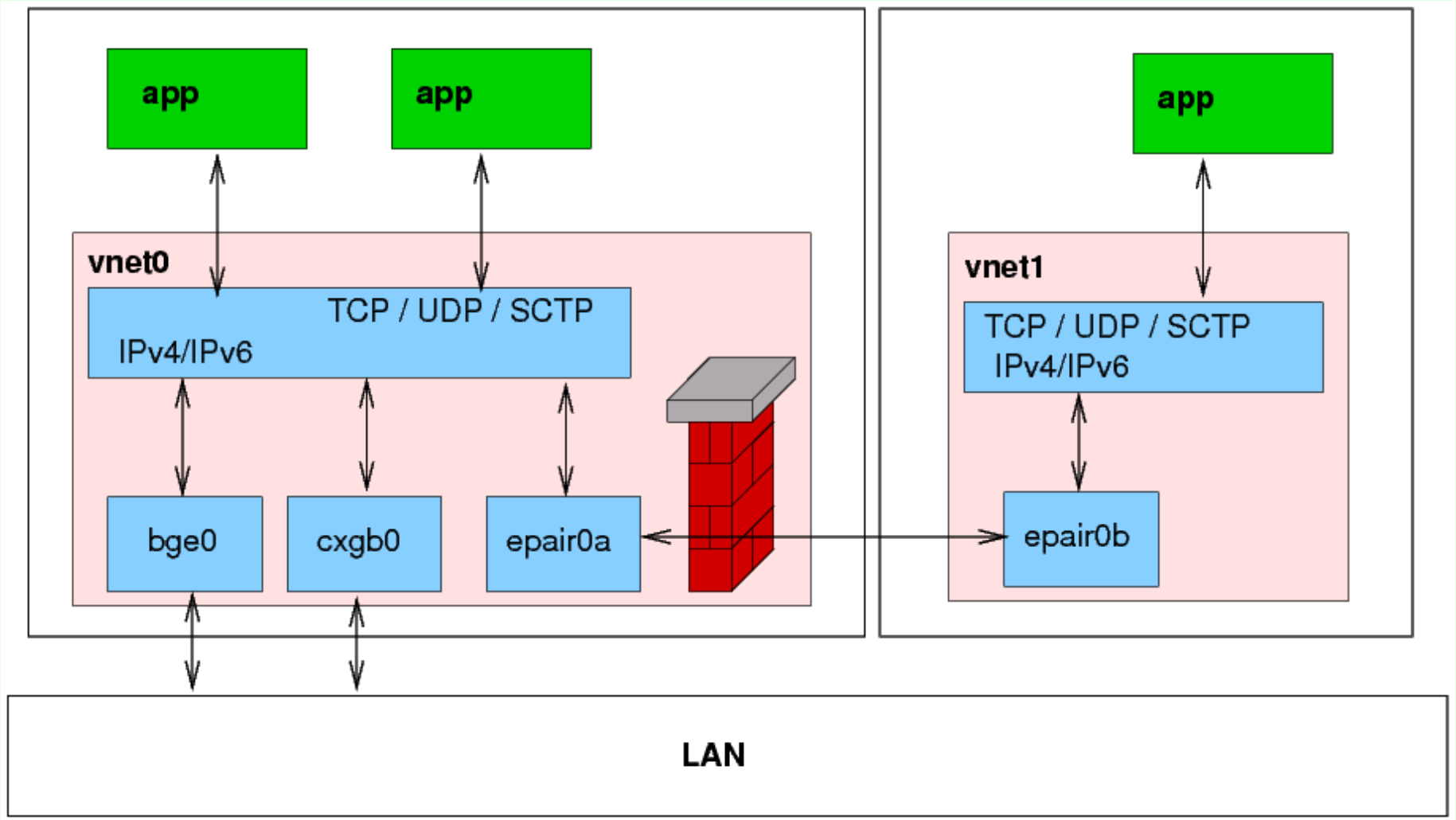
What's the major problem left?

- Shutdown is really hard. "Why?", you ask?
- Up to now we booted and the reset pin handled did the "shutdown" for us.
- We need destructors.

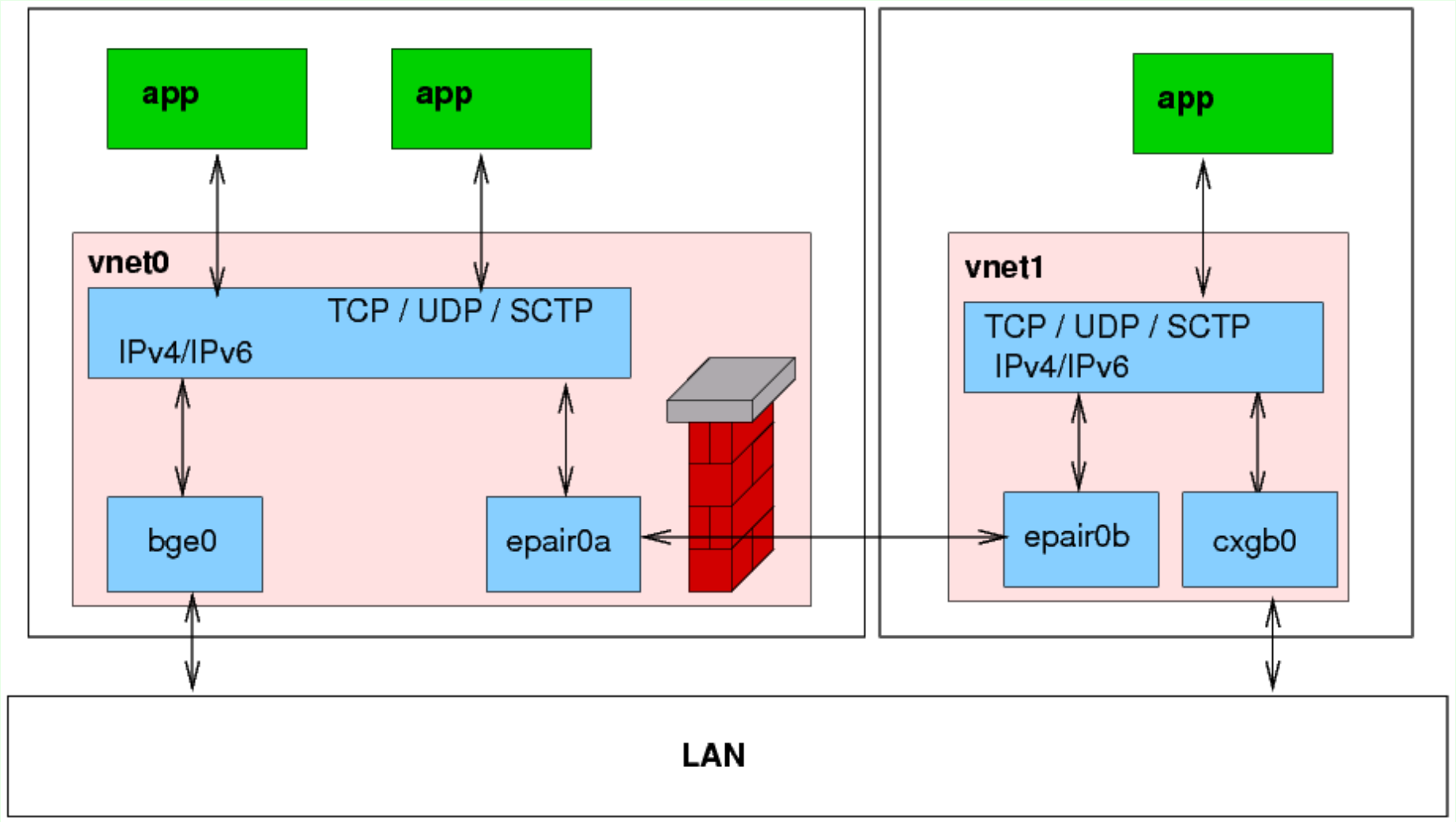
Where are we?

- VIMAGE is a “highly experimental” feature in FreeBSD 8.0.
- Known memory leaks, known crash conditions.
- A bit this and that left todo: (IPX, Appletalk, 1.5 firewalls, . . .).
- FreeBSD Foundation helps to make us make progress.
- Goal: production quality VIMAGE somewhen in 9.x.

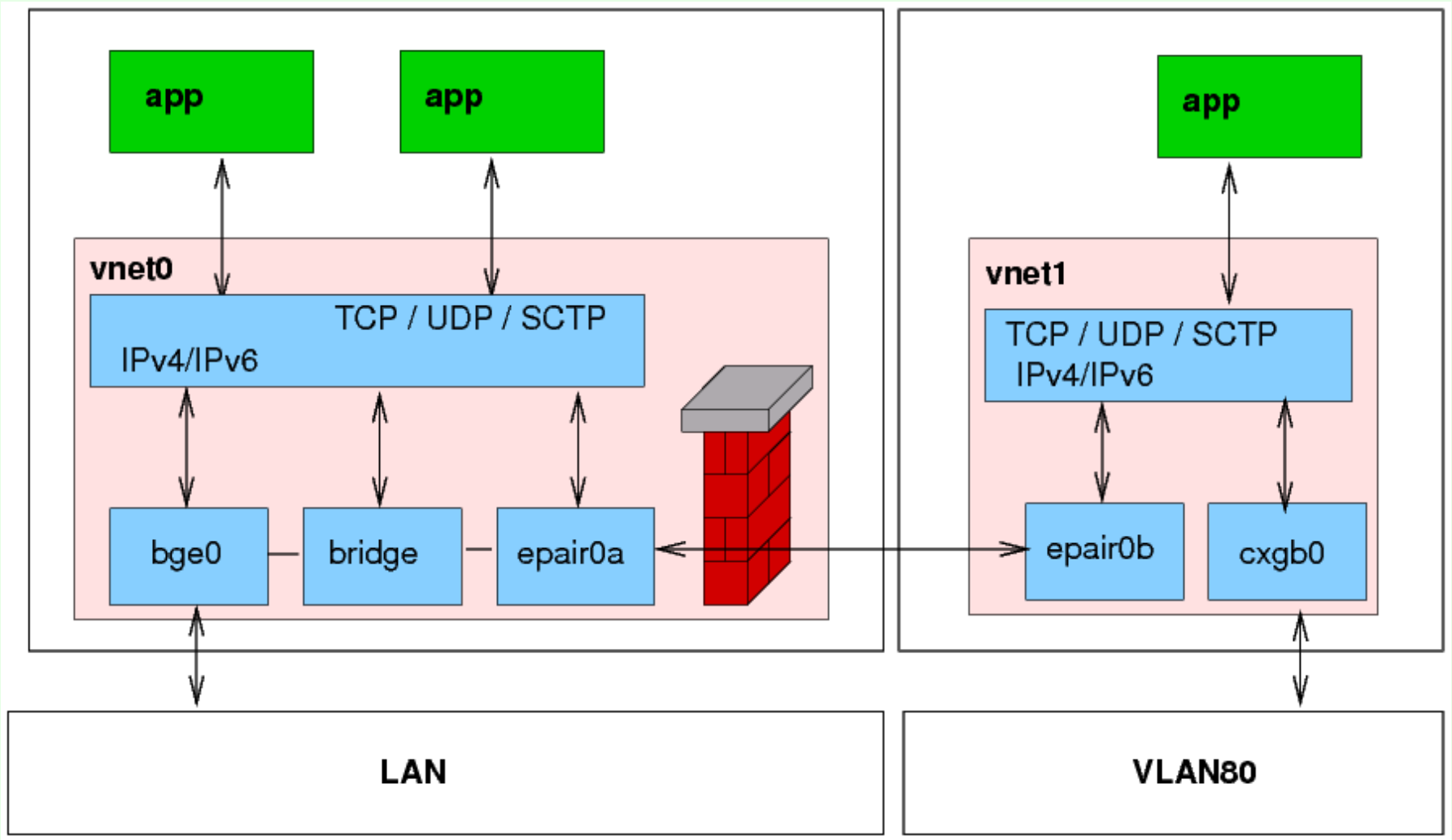
Virtual Network stack sample 1



Virtual Network stack sample 2



Virtual Network stack sample 3

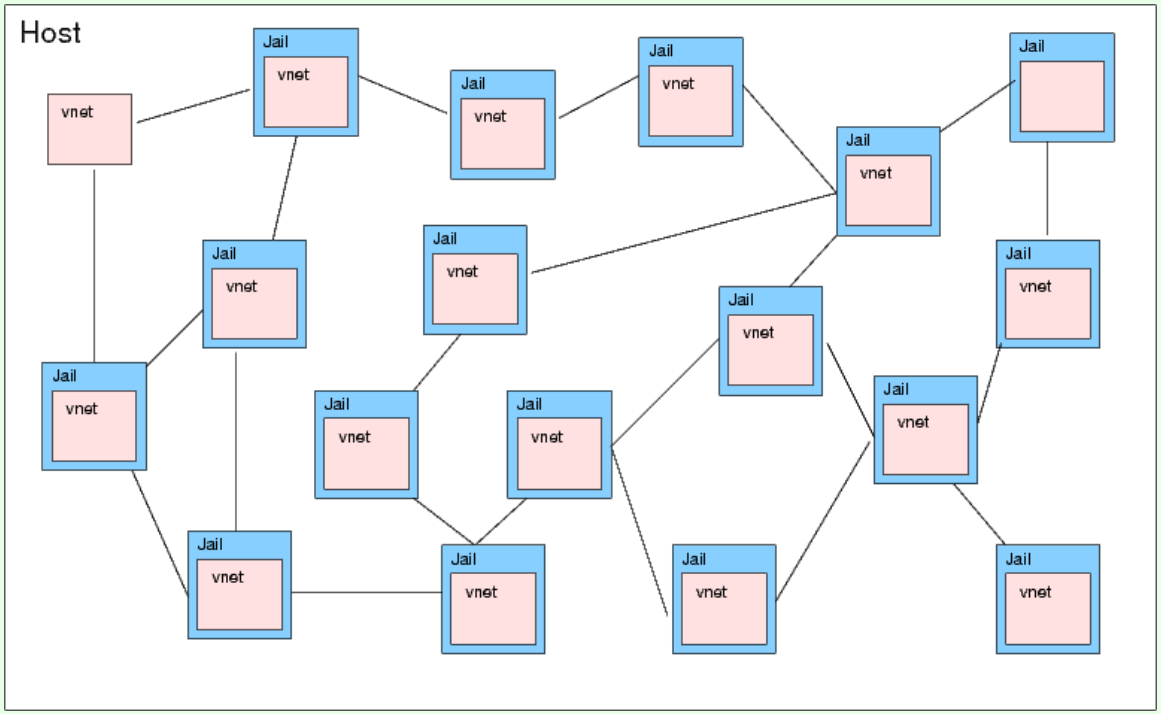


Protocol development or Admin/Operation

- BSD the classic OS for (network protocol) research.
- Easily test with various nodes on one piece of metal.
- Have all the debugging tools available, like tcpdump, DTrace, . . .
- Can more easily correlate data.

Simulations

Arbitrary structure, 1000s of nodes with independent network stacks



ISPs and Hosters

- Run tens, hundreds or thousands of jails on a box.
- As low as 2MB + user data per virtual instance using a read only base image and nullfs mounts technique or ZFS.
- Allow shell logins and root access to a virtual instance.
- Add dedicated/virtual resources only where needed.
- Provide IPsec, own firewall, ability to tcpdump, ping, . . . to users.

Scalability

- A classic jail w/o process is about 5k on 64bit.
- A jail + vnet w/o processes is about 300-500k on 64bit.
  Depending on kernel configuration.
- We can give you six 9s (at least sounds good:)
- A 64bit machine, netbooted, 8GB RAM can start >5000 jails + vnet + epair
  (*) actually higher number.
- The number will always depend on your environment, number of processes, workload, ...

How to try it out?

- Use 8-STABLE or 9-CURRENT.
- Compile kernel with `options VIMAGE`
- Still simple commands:

```
jail -i -c vnet name=foo \
   host.hostname=foo.example.net path=/ persist
ifconfig epair0 create
ifconfig epair0a vnet foo
jexec foo /bin/csh
```

- http://wiki.freebsd.org/Image
- Experimental - you have been warned.

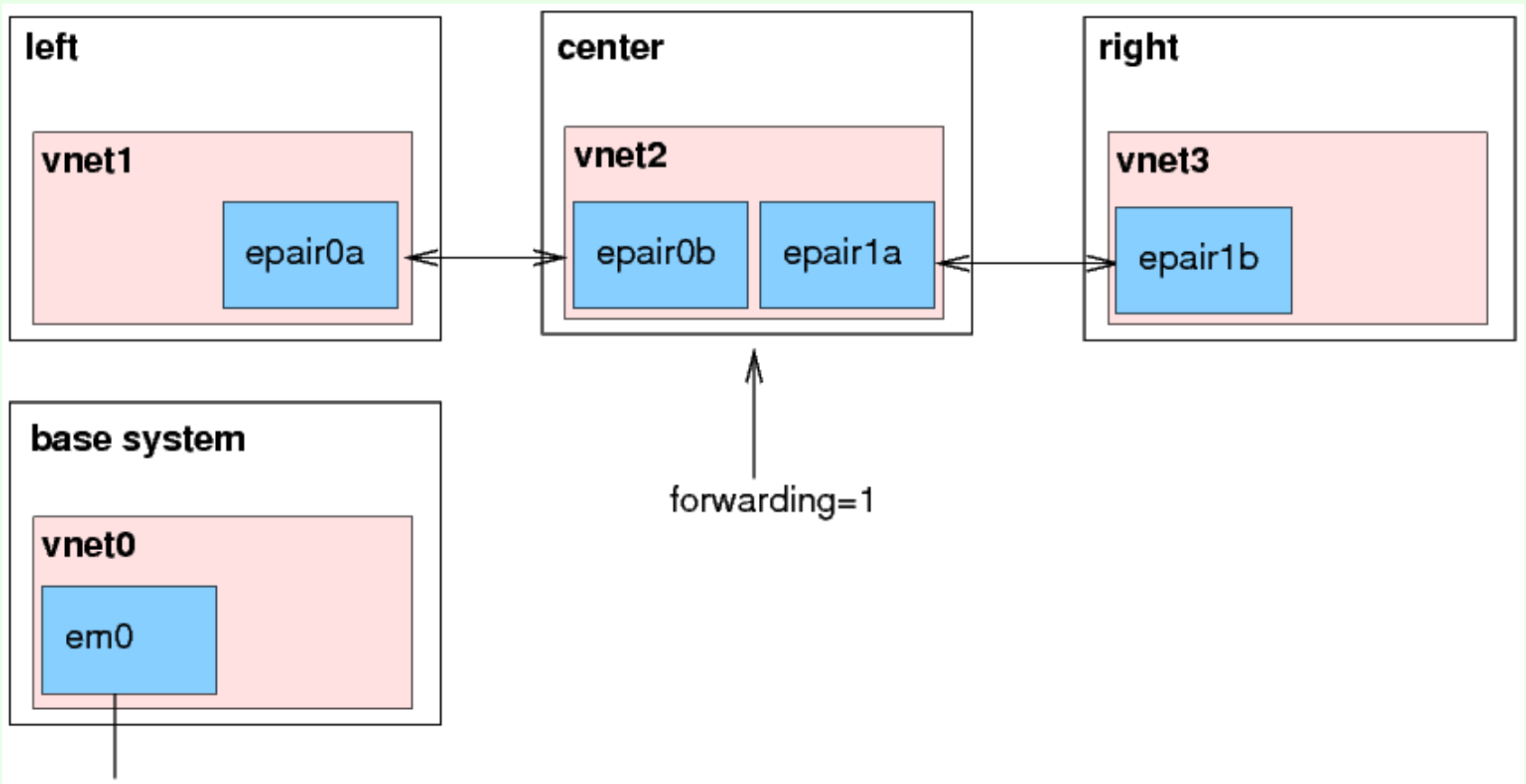
Conclusions

- Virtual kernel subsystem, like vnet, become reality.
- Prototype increasingly stable.
- Very little performance overhead.
- Enlarges FreeBSD's portfolio, can be combined with Xen.
- Coming soon(ish).

So long for now

- Special thanks to the FreeBSD Foundation.
- If you are interested let us know: bz@FreeBSD.org, freebsd-virtualization@FreeBSD.org.
- We will have ideas how you could help.
- That's it. Thanks!

Demo


left
vnet1
epair0a
center
vnet2
epair0b
epair1a
right
vnet3
epair1b
base system
vnet0
em0
forwarding=1