
HISSA
AIBO
MIETO
RODNEY
RODNEY

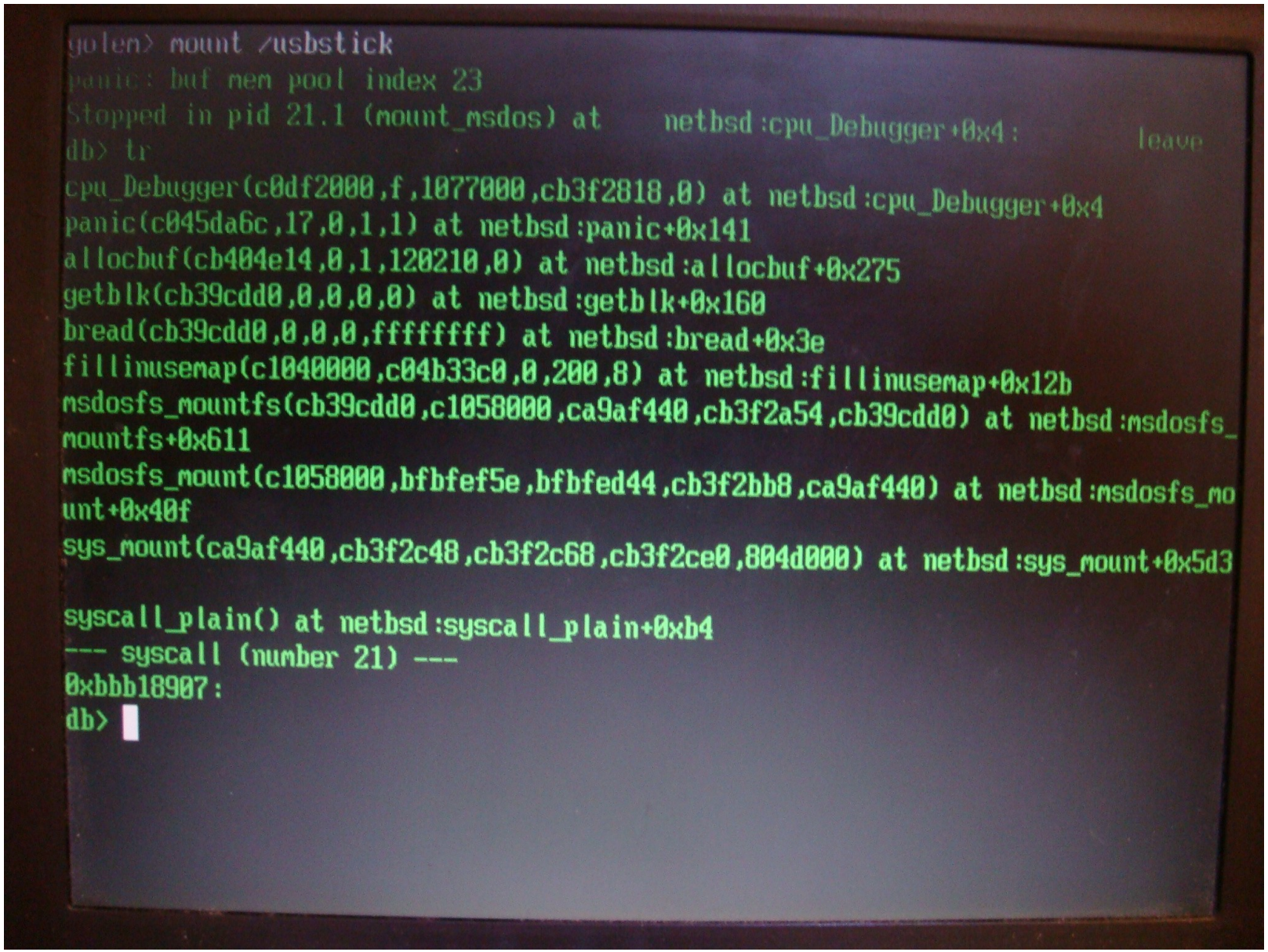
Ultralightweight Kernel Service Virtualization with *Rump Kernels*

Antti Kantee
pooka@iki.fi

FOSDEM 2012
Virtualization and Cloud Devroom

Brussels, Belgium

mount /usbstick: A Problem



We Don't Like Problems

- write bugfree code?
  - sure, great idea
- FUSE?
  - only part of the solution, need *drivers*
- carry a second laptop for USB sticks?
  - heavyweight approach
- mount USB sticks in virtual machine?
  - see above

A Lightweight Virtual Driver

perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away

- we do *not need* a *duplicated copy* of e.g.
  - program execution support and root fs image
  - virtual memory subsystem
  - scheduling
- make virtualizing only the driver possible


REWRITE
ALL THE DRIVERS
memegenerator.net


REWRITE
ALL THE DRIVERS

Pick & Choose Existing Code

- monolithic kernels offer best driver support
- need the correct support routine semantics for drivers
  - process/thread context
  - CPU context
  - locking, memory allocation, etc.
- need to be able to link it

They said about the monolithic

But I didn't believe them

Enter The *Anykernel*

- term we define for a kernel codebase which allows drivers to run standalone
- does not define runtime organization
  - unlike the terms *monolithic kernel*, *microkernel*, *exokernel*, *multikernel*, …
  - instead, states it is possible
- NetBSD kernel turned into an anykernel with a few relatively simple modifications

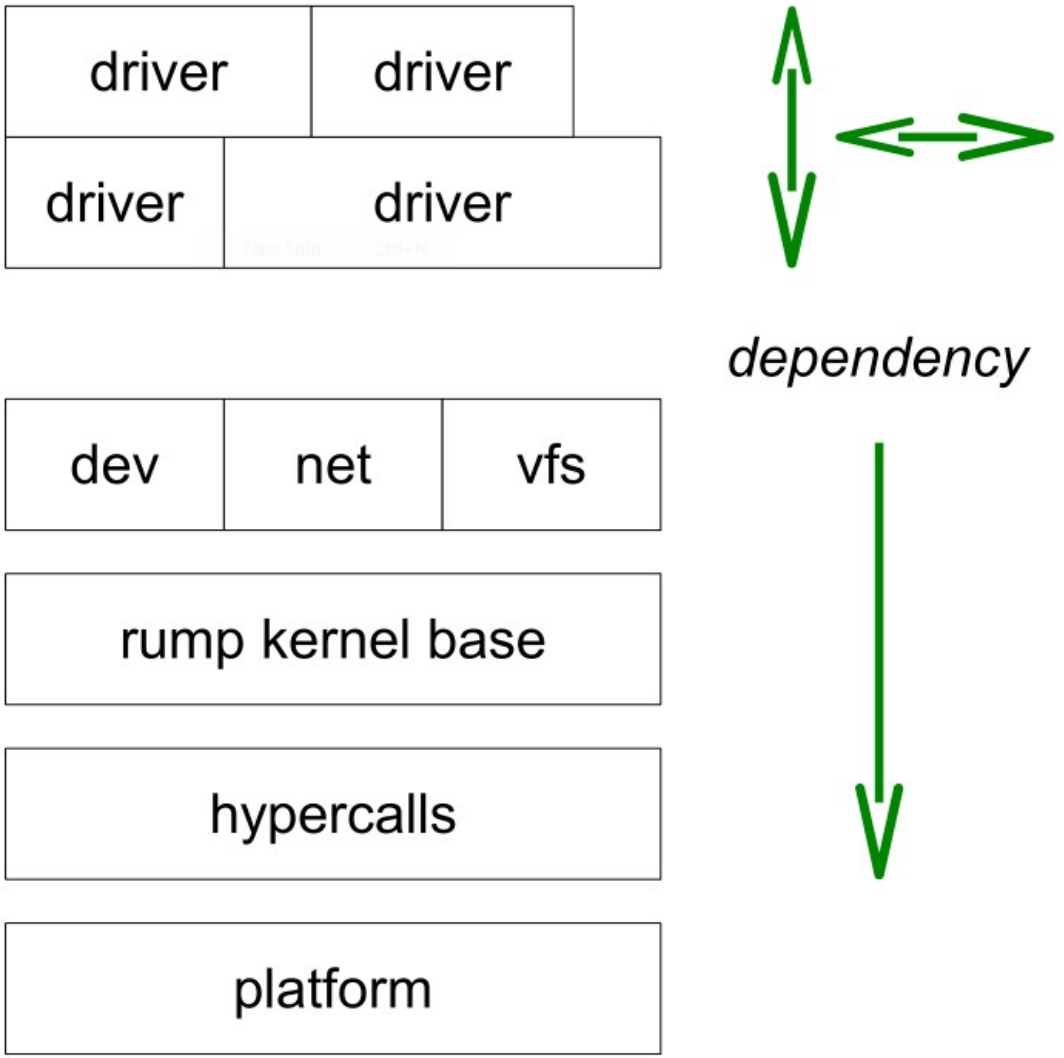
Now that we can, we should

- an anykernel used as a monolithic kernel does not solve problems (or cause them)
- we need the lightweight virtual driver
- *bring together Lord Order and Lord Chaos*

The *rump kernel*

- include just enough for the driver to link and run properly
- paravirtualize
- implement necessary glue code, e.g.
  - scheduling
  - virtual memory (note: not memory allocators, they are available courtesy of the anykernel)
- glue amounts to ~10% of all support code

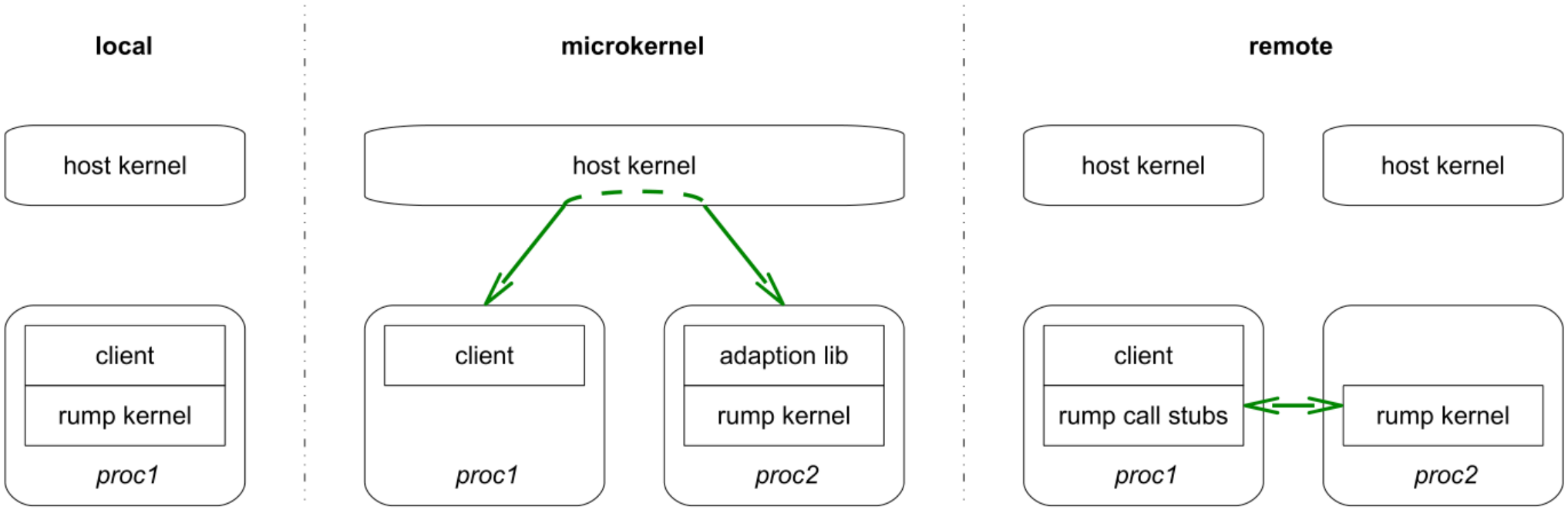
Like a tiramisu



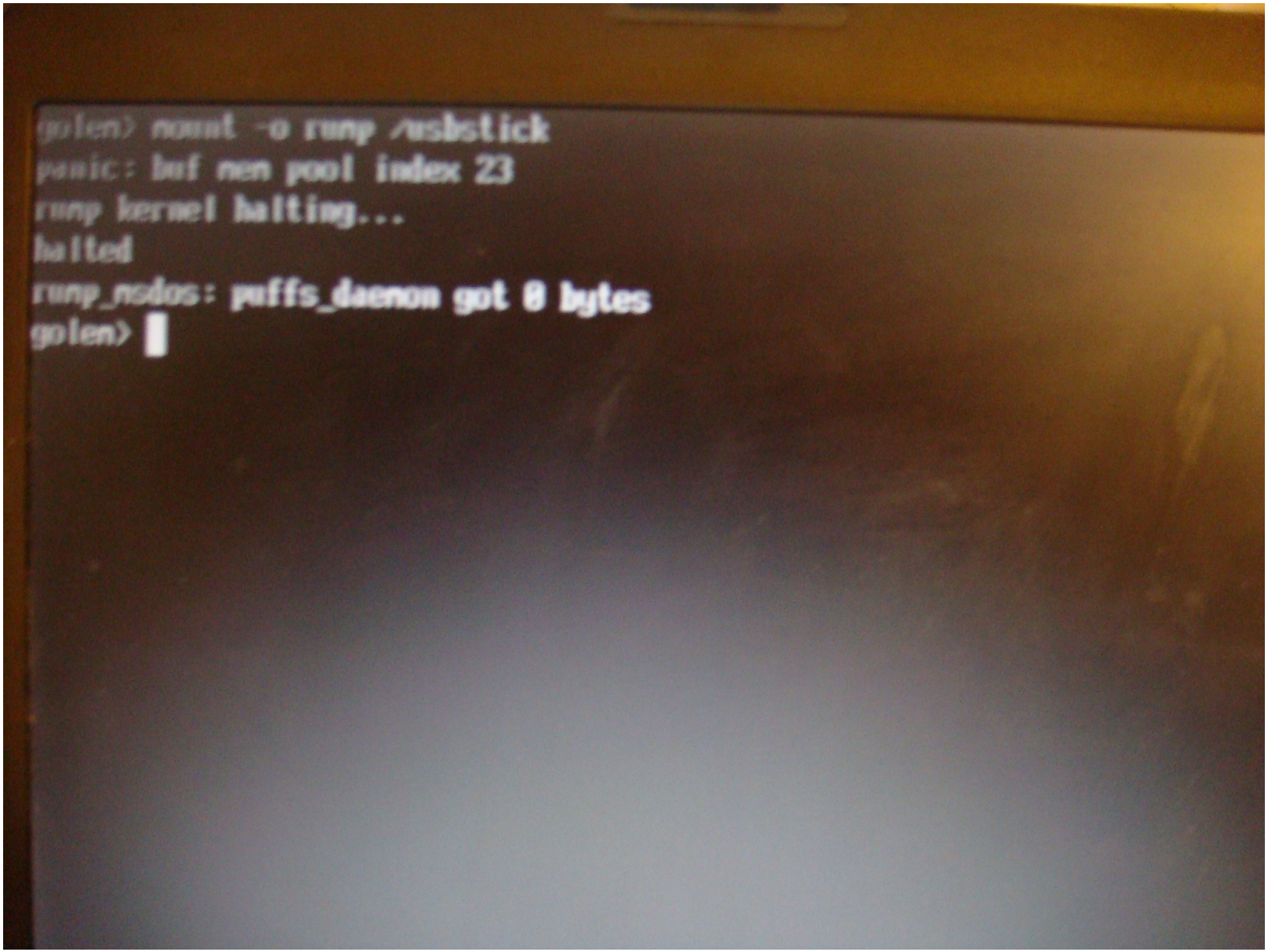
Current support in rump kernels

- unmodified drivers, *source* and *binary*
- file systems
  - disk based: FFS, ext2, FAT, etc.
  - network based: NFS, SMBFS
- networking
  - TCP/IP, TCP/IPv6, 802.11, bluetooth
- device drivers
  - USB hardware drivers
  - pseudo devices, e.g. cryptodisk, BPF, RAID

Here come the clients



mount -o rump /usbstick: no prob



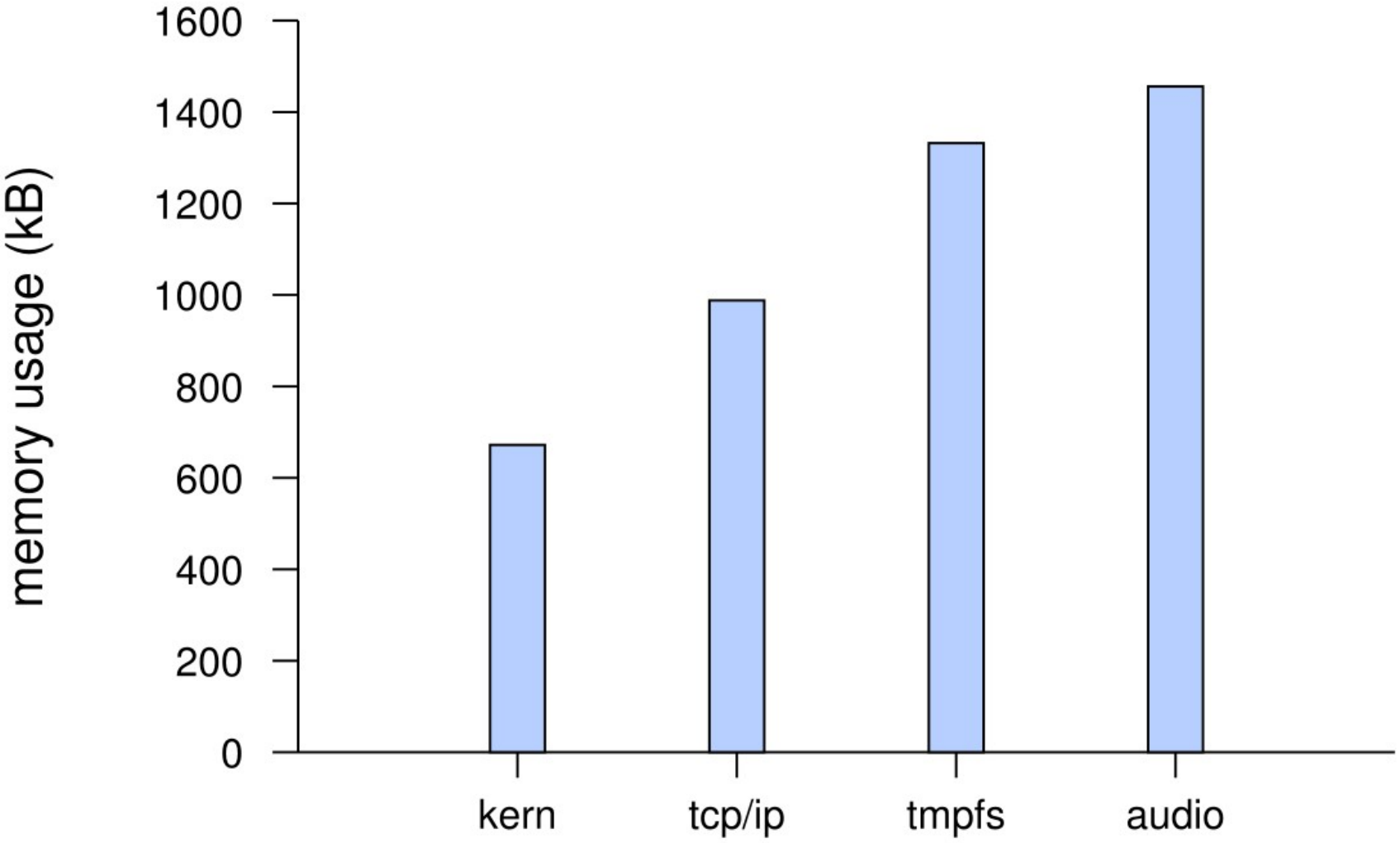
Half-point summary

- *anykernel*: kernel codebase supports organization of drivers in any runtime configuration
- *rump kernel*: kernel providing unmodified virtual drivers
- clients: local, microkernel and remote
- supported, shipped and working in NetBSD since 2007

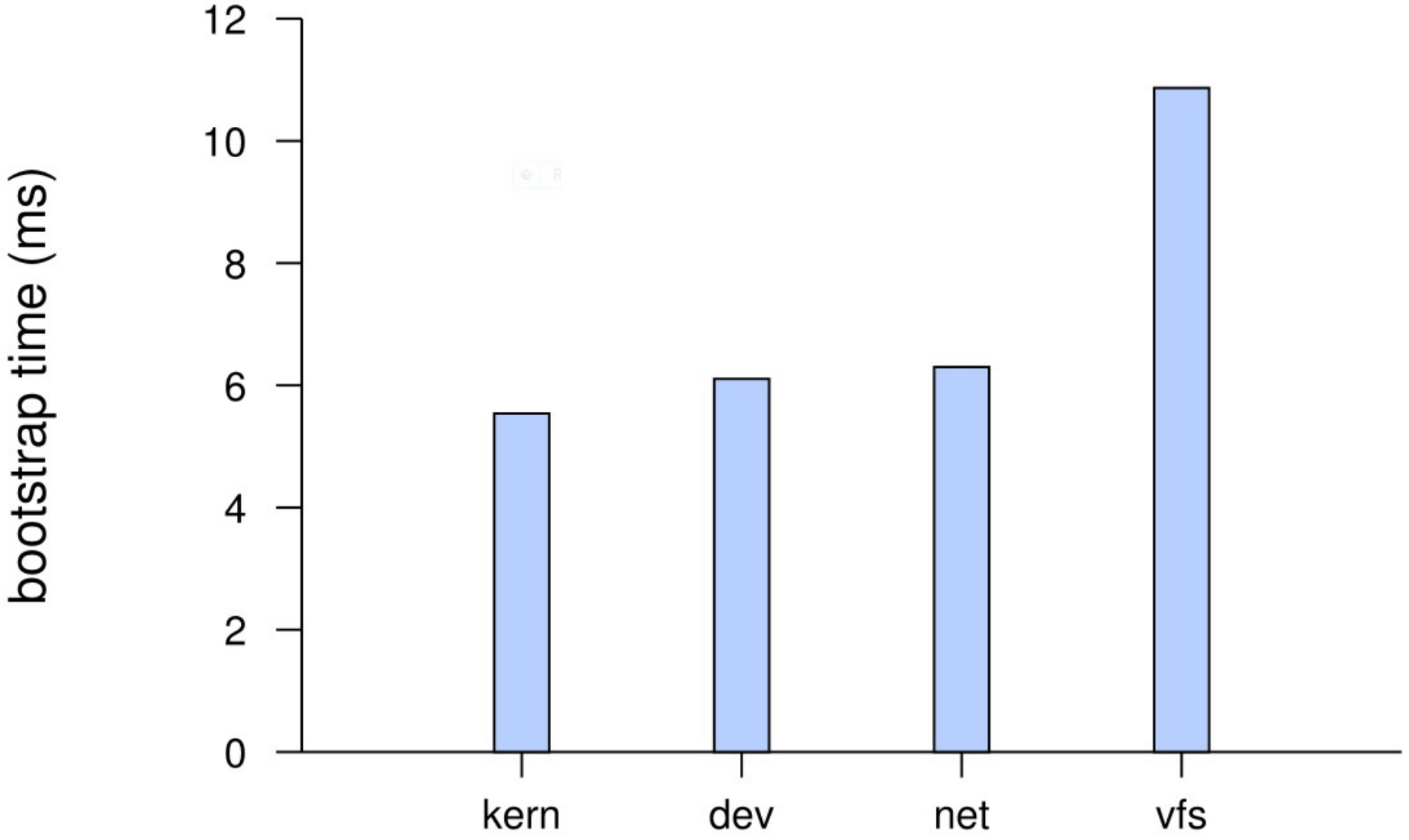
Still more to come

- performance figures
  - for the default out-of-the-box installation
- other uses
  - applications
  - testing & development
- few words on remote clients
  - and the cloud!

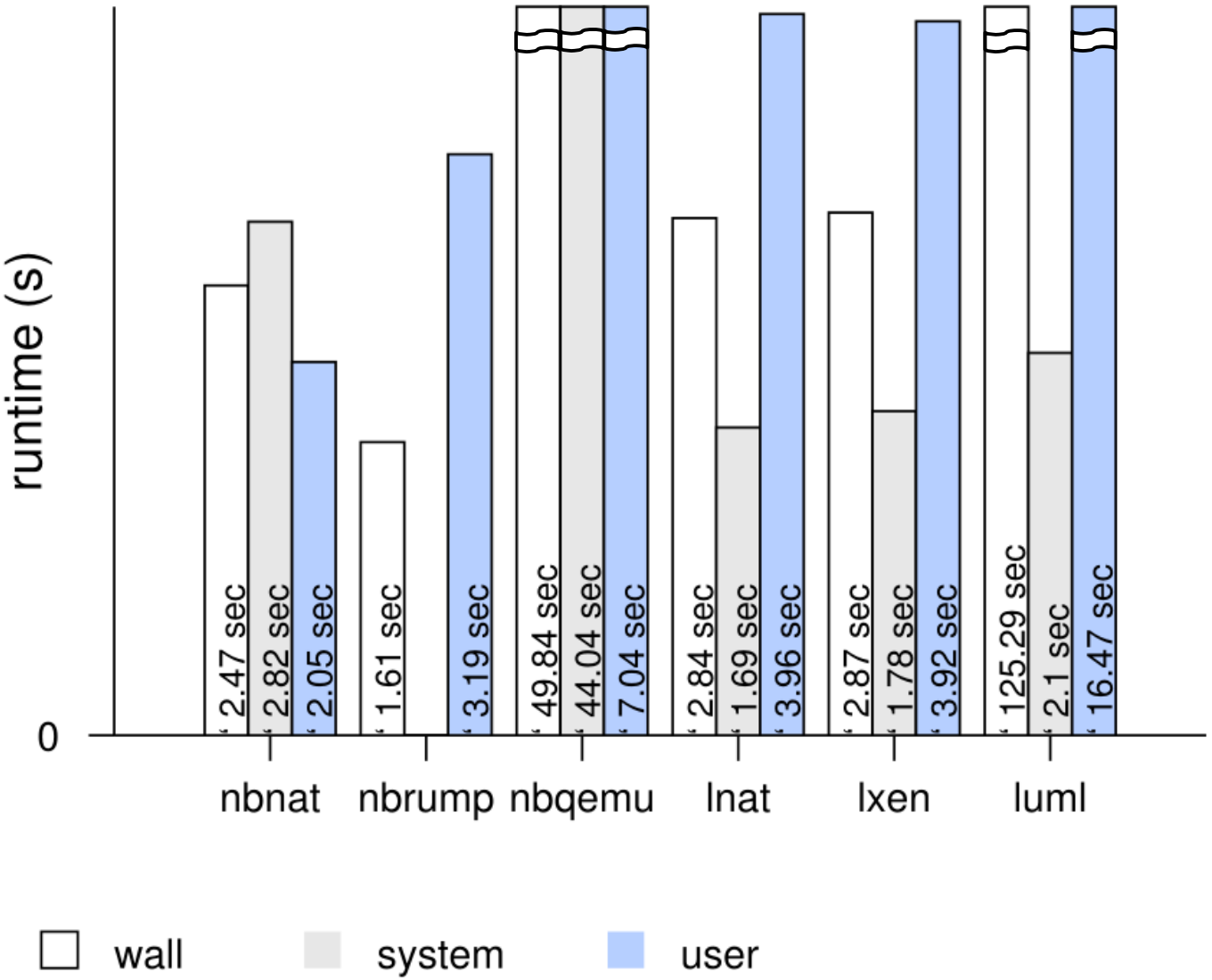
Memory use


memory usage (kB)
1600
1400
1200
1000
800
600
400
200
0
kern
tcp/ip
tmpfs
audio

Bootstrap time



Null system call speed



Applications: fs-utils

- mtools-like file system utility suite created by Arnaud Ysmal
  - cat, ls, cp, mv, ...
- local clients for rump kernels, supports NetBSD kernel file system drivers
  - `fsu_ls /dev/rwd0e -laF`
  - `fsu_ls 10.181.181.181:/m/dm -laF`
- preserves usage for you `ls` flag pleasure

Testing

- rump kernels are unparallelled for kernel testing
  - crashproof => test buggy code
  - bootstrap time => no overhead from safety
  - isolated => kernel settings do not affect host
  - *first class citizen* => easy to collect results
  - massively virtualizable => network testing

File system testing

- file system independent testing framework created by Nicolas Joly
  - fs specific code ~20-40 lines to create the file system and mount it in a rump kernel
  - each test can be run against all file systems
- use in test-driven development!
  - safe testing of experimental file system drivers, e.g. LFS (or ZFS on NetBSD)

NetBSD test suite

- daily tests use:
  - ATF (Julio Merino)
  - anita (Andreas Gustafsson)
- March 31st 2011 the test suite bootstrapped 911 rump kernels in 2178 tests and completed the run in 56min
  - less than 4s per test
  - 15 tests caused a rump kernel panic or hang due to a known driver bug

ptyioctl	Expected signal	PR kern/40688
lib/libc/stdlib/t_strtox		
hexadecimal	Expected failure	PR lib/44189: /bracket/i386/work /2011.03.08.22.21.52/src/tests/lib/libc/stdlib /t_strtox.c:55: strtod(str, &end) == -0.0 && end == str+2 not met
lib/libpthread/t_sem		
before_start_one_thread	Expected timeout	Race condition; it is probably unsafe to call sem_post from a signal handler when using the pthread version
lib/semaphore/pthread/t_sem_pth		
unlink	Expected failure	PR kern/43452: close unlinked semaphore: Invalid argument
net/bpf/t_bpf		
bpfwriteleak	Expected failure	PR kern/44196: /bracket/i386/work /2011.03.08.22.21.52/src/tests/net/bpf/t_bpf.c:96: getmtdata() != 0
net/if_loop/t_pr		
loopmtu	Expected signal	PR kern/43664

Standard output stream

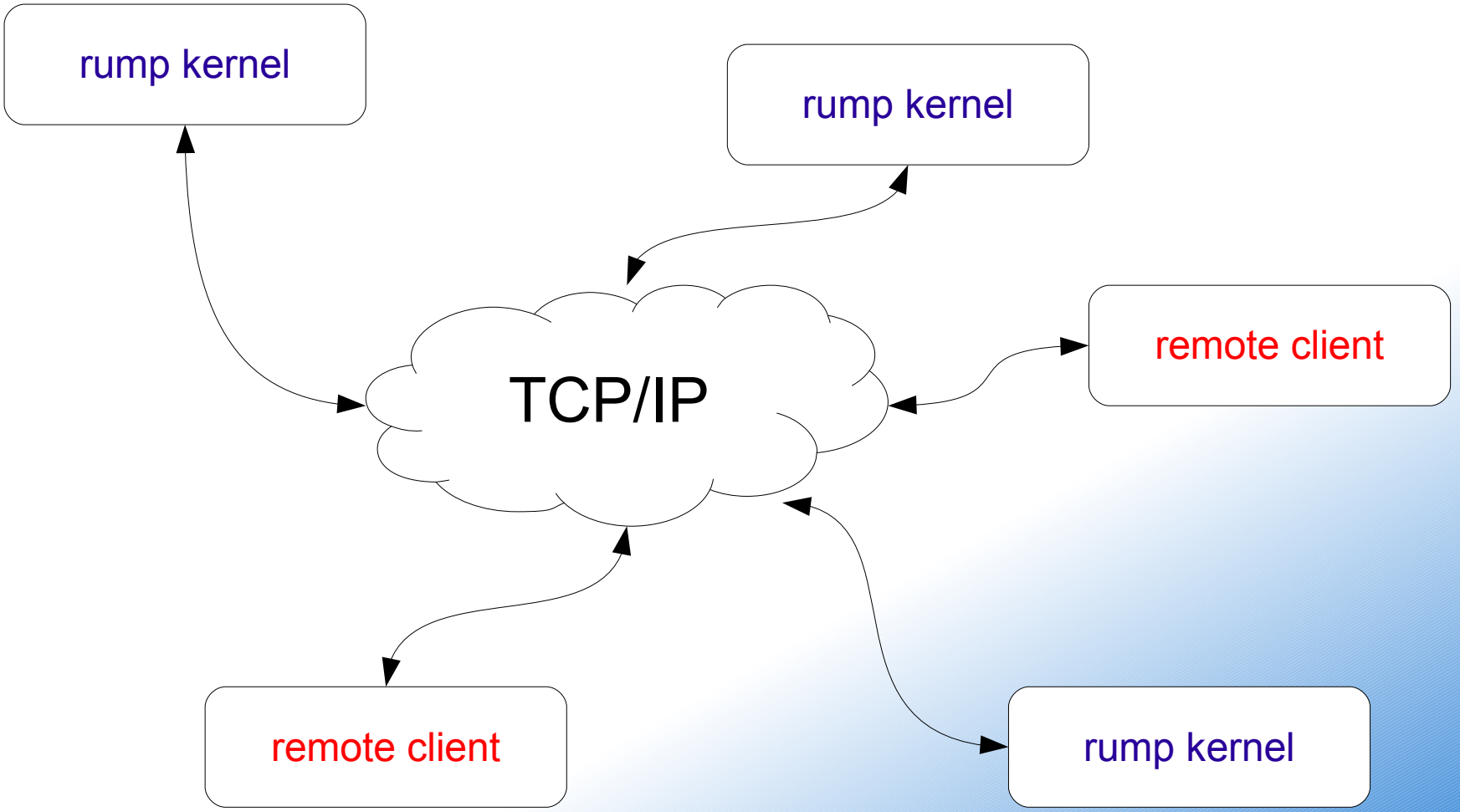
```
panic: kernel diagnostic assertion "m->m_pkthdr.csum_flags & M_CSUM_IPv4" failed: file
rump kernel halting...
halted
```

Standard error stream

```
test program crashed, autolisting stacktrace:
Core was generated by `t_pr'.
Program terminated with signal 6, Aborted.
#0  0xbba54047 in _lwp_kill () from /usr/lib/libc.so.12
#0  0xbba54047 in _lwp_kill () from /usr/lib/libc.so.12
#1  0xbba54005 in raise () from /usr/lib/libc.so.12
#2  0xbba537da in abort () from /usr/lib/libc.so.12
#3  0xbba8e050 in rumpuser_exit () from /usr/lib/librumpuser.so.0
#4  0xbbb1b4aa in rumpns_cpu_reboot () from /usr/lib/librump.so.0
#5  0xbbaf1618 in rumpns_panic () from /usr/lib/librump.so.0
#6  0xbbad9493 in rumpns_kern_assert () from /usr/lib/librump.so.0
#7  0xbbbb8a72 in rumpns_ip_fragment () from /usr/lib/librumpnet_net.so.0
#8  0xbbbb9f0b in rumpns_ip_output () from /usr/lib/librumpnet_net.so.0
#9  0xbbbb7b17 in rumpns_rip_output () from /usr/lib/librumpnet_net.so.0
#10 0xbbbb7f58 in rumpns_rip_usrreq () from /usr/lib/librumpnet_net.so.0
#11 0xbbbc1c7a in rumpns_sockaddr_in_addr () from /usr/lib/librumpnet_net.so.0
#12 0xbbb5d5ba in rumpns_sosend () from /usr/lib/librumpnet.so.0
#13 0xbbb56854 in rumpns_do_sys_sendmsg () from /usr/lib/librumpnet.so.0
#14 0xbbb56a81 in rumpns_sys_sendto () from /usr/lib/librumpnet.so.0
#15 0xbbb11b2f in rumpns_sys_unmount () from /usr/lib/librump.so.0
#16 0xbbb14d5c in rump___sysimpl_sendto () from /usr/lib/librump.so.0
#17 0x08049912 in atfu_loopmtu_body ()
#18 0x0804b86b in atf_tc_run ()
```

Remote clients

They're scattered and they're cloudy



well, it's also a cloud workshop ...

Generality of the approach

- compiled & ran NetBSD drivers in rump kernels on Linux
  - some compat code for types required
- wrote prototypes for Linux and FreeBSD
  - both experiments resulted in a driver which would run in NetBSD
  - shared the same hypercall implementation as on NetBSD (rumpuser)

Future platform possibilities

- processes in other POSIX style hosts
- microkernel style host interfaces
  - Minix, Xen, L4, …
- hardware
  - need more support in the hypercall layer because not provided readily by platform
  - benefit: use readily available drivers in lightweight embedded systems / ASICs

Conclusions

More info

- http://www.NetBSD.org/docs/rump/
- install NetBSD-current, support available out-of-the-box
  - or a future NetBSD 6 version
  - tip: use anita for a quick test in a VM

    http://www.gson.org/netbsd/anita/
- read the source
- read my maybe-available-soon dissertation