Trends in HPC OS

SAND2007-7494P

INDESCRIBABLE...
INDESTRUCTIBLE!
NOTHING CAN STOP IT!
THE BLOB
STEVEN McQUEEN ANETA CORSEAUT · EARL ROWE
PRODUCED BY JACK H. HARRIS · DIRECTED BY IRVIN S. YEAWORTH, JR. · SCREENPLAY BY THEODORE SIMONSON AND KATE PHILLIPS
FROM AN IDEA BY IRVINE H. MILLGATE
A TONYLYN PRODUCTION · COLOR BY DE LUXE

Ron Minnich, SNL

Overview

- Relative slowdown as the blob grows exponentially
- Fault tolerance is the wrong mindset
- We need good design, not bad design, in kernels
- Programming to 100M CPUs is a different game

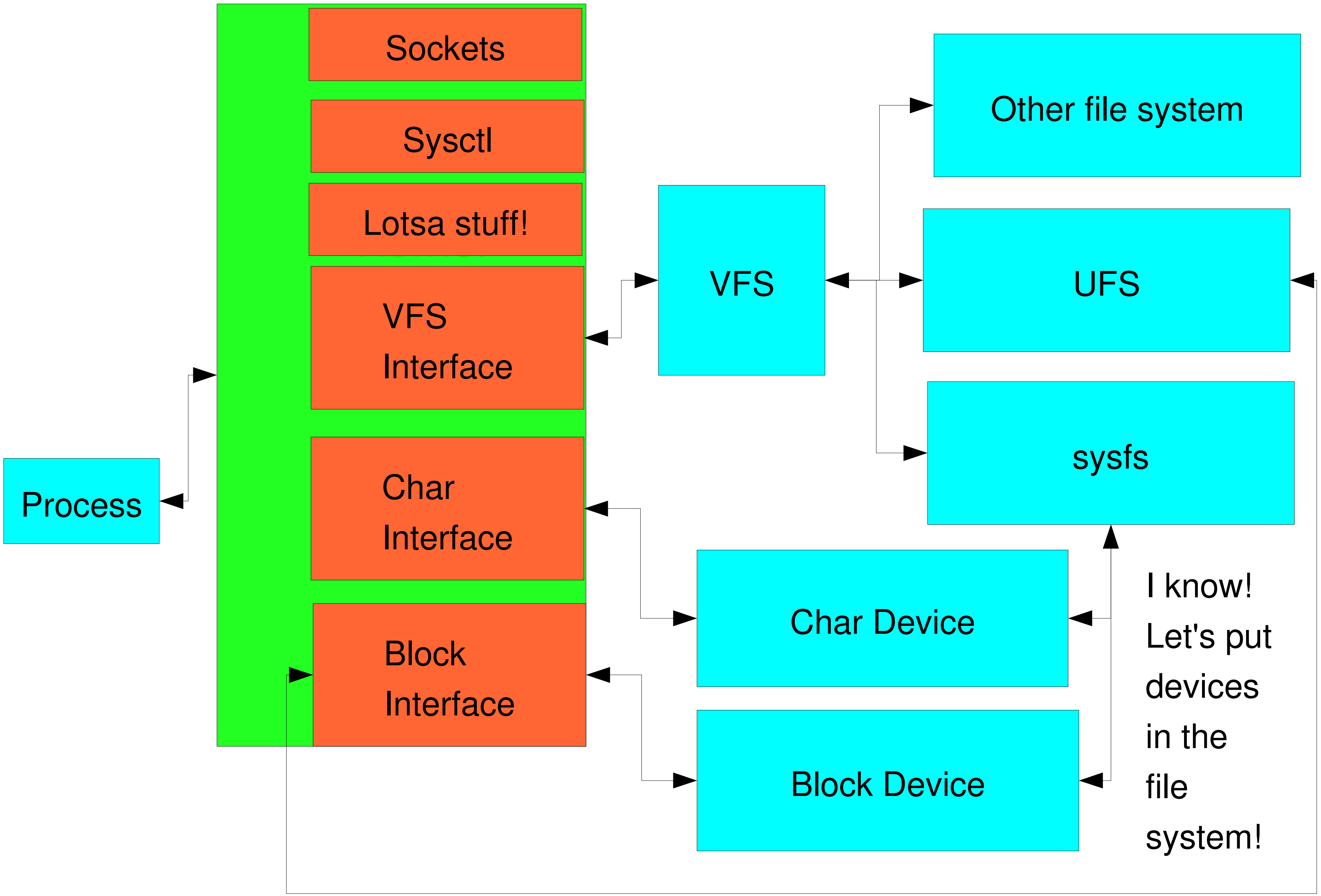
Fault Oblivious computing

- Application not aware of faults
- Does not respond to them
- Computes correctly even as they occur
    - You don't know about each disk error, do you?
- Works @ Google
- This is NOT “fault tolerance”!

One of the questions

- Should we add a system call to frob the natz?
- Ooops. The question presupposes many things
- In particular, that adding a system call is required to gain functionality
- In all too many cases, that's really covering up for poor design

Poor Design



Sockets

Sysctl

Lotsa stuff!

VFS Interface

Char Interface

Block Interface

Process

VFS

Other file system

UFS

sysfs

Char Device

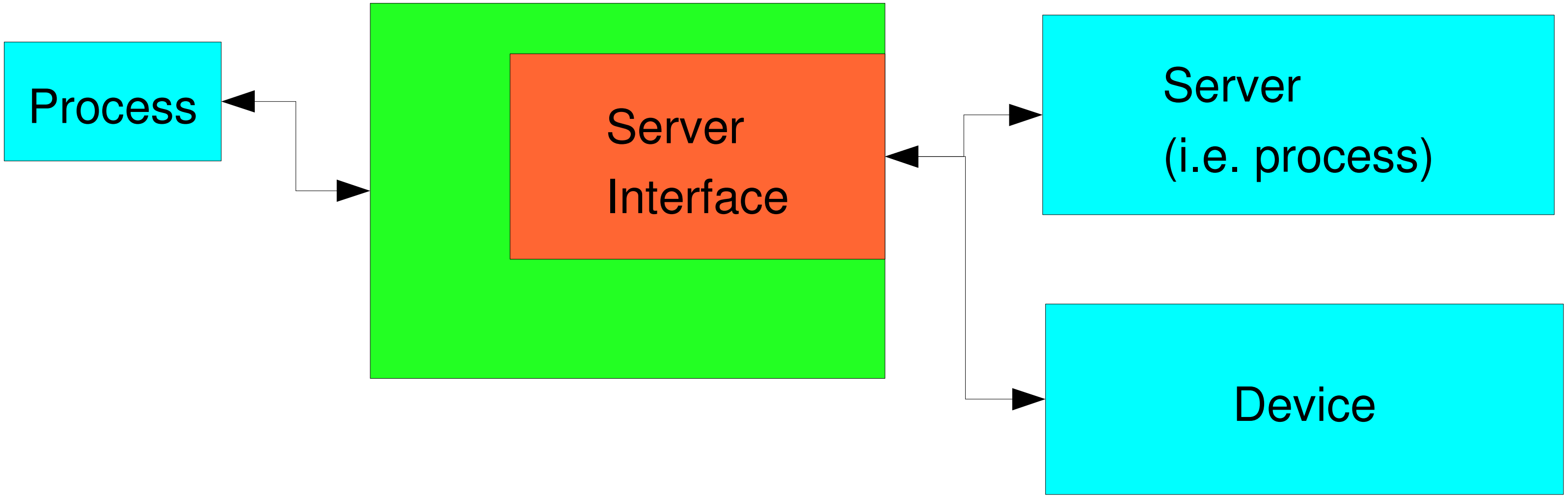
Block Device

I know! Let's put devices in the file system!

HEAVY

- _llseek _newselect _sysctl access acct add_key adjtimex afs_syscall alarm bdflush break brk capget capset chdir chmod chown chown32 chroot clock_getres
- clock_gettime clock_nanosleep clock_settime clone close creat create_module delete_module dup dup2 epoll_create epoll_ctl epoll_pwait epoll_wait execve exit exit_group faccessat fadvise64 fadvise64_64
- fchdir fchmod fchmodat fchown fchown32 fchownat fcntl fcntl64 fdatasync fgetxattr flistxattr flock fork fremovexattr fsetxattr fstat fstat64 fstatat64 fstatfs fstatfs64
- fsync ftime ftruncate ftruncate64 futex futimesat get_kernel_syms get_mempolicy get_robust_list get_thread_area getcpu getcwd getdents getdents64 getegid getegid32 geteuid geteuid32 getgid getgid32
- getgroups getgroups32 getitimer getpgid getpgrp getpid getpmsg getppid getpriority getresgid getresgid32 getresuid getresuid32 getrlimit getrusage getsid gettid gettimeofday getuid
- getuid32 getxattr gtty idle init_module inotify_add_watch inotify_init inotify_rm_watch io_cancel io_destroy io_getevents io_setup io_submit ioctl ioperm iopl ioprio_get ioprio_set ipc kexec_load
- keyctl kill lchown lchown32 lgetxattr link linkat listxattr llistxattr lock lookup_dcookie lremovexattr lseek lsetxattr lstat lstat64 madvise madvise1 mbind migrate_pages
- mincore mkdir mkdirat mknod mknodat mlock mlockall mmap mmap2 modify_ldt mount move_pages mprotect mpx mq_getsetattr mq_notify mq_open mq_timedreceive mq_timedsend mq_unlink
- mremap msync munlock munlockall munmap nanosleep nfsservctl nice oldfstat oldlstat oldolduname oldstat olduname open openat pause personality pipe pivot_root poll
- ppoll prctl pread64 prof profil pselect6 ptrace putpmsg pwrite64 query_module quotactl read readahead readdir readlink readlinkat readv reboot remap_file_pages removexattr
- rename renameat request_key restart_syscall rmdir rt_sigaction rt_sigpending rt_sigprocmask rt_sigqueueinfo rt_sigreturn rt_sigsuspend rt_sigtimedwait sched_get_priority_max sched_get_priority_min sched_getaffinity
- sched_getparam sched_getscheduler sched_rr_get_interval sched_setaffinity sched_setparam sched_setscheduler sched_yield select sendfile sendfile64 set_mempolicy set_robust_list set_thread_area set_tid_address setdomainname
- setfsgid setfsgid32 setfsuid setfsuid32 setgid setgid32 setgroups setgroups32 sethostname setitimer setpgid setpriority setregid setregid32 setresgid setresgid32 setresuid setresuid32 setreuid setreuid32
- setrlimit setsid settimeofday setuid setuid32 setxattr sgetmask sigaction sigaltstack signal sigpending sigprocmask sigreturn sigsuspend socketcall splice ssetmask stat stat64 statfs
- statfs64 stime stty swapoff swapon symlink symlinkat sync sync_file_range sysfs sysinfo syslog tee tgkill time timer_create timer_delete timer_getoverrun timer_gettime timer_settime
- times tkill truncate truncate64 ugetrlimit ulimit umask umount umount2 uname unlink unlinkat unshare uselib ustat utime utimes vfork vhangup vm86 vm86old vmsplice vserver wait4 waitid waitpid write writev
-

Good Design



With a common server interface, location of services is no longer important. The differentiation of char/block is archaic. Processes no longer distinguish servers and devices. Devices no longer have numbers.

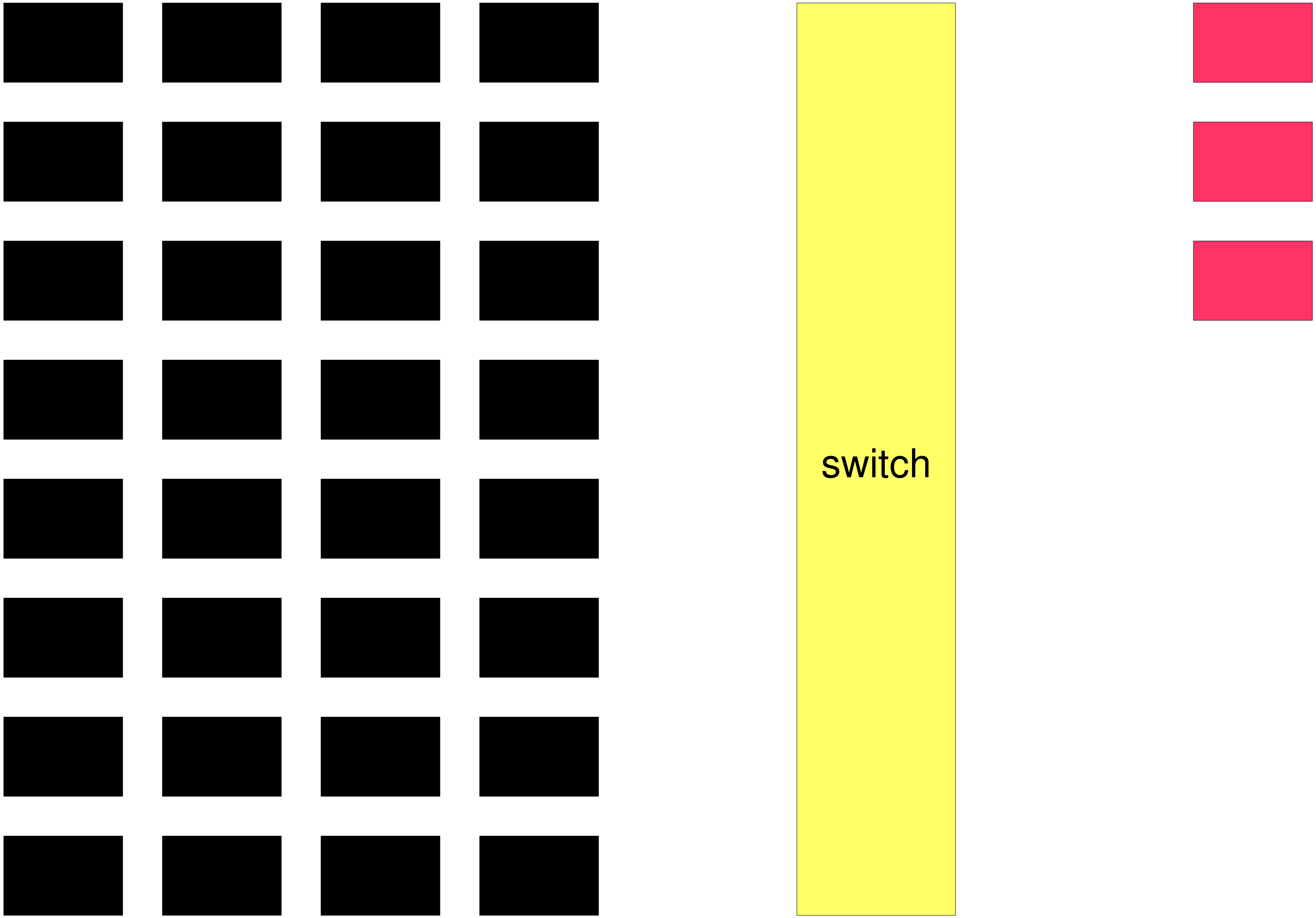
LIGHT

- BIND CHDIR CLOSE DUP ALARM EXEC
- EXITS FAUTH SEGBRK OPEN OSEEK SLEEP
- RFORK PIPE CREATE FD2PATH BRK_ REMOVE
- NOTIFY NOTED SEGATTACH SEGDETACH SEGFREE
- SEGFLUSH RENDEZVOUS UNMOUNT SEMACQUIRE
- SEMRELEASE SEEK FVERSION ERRSTR STAT FSTAT
- WSTAT FWSTAT MOUNT AWAIT PREAD PWRITE

Kernel is a multicore application

- And hence should be as easily parallelized as your app
- But Linux and LWK are not that way
  - Well, LWK doesn't do much anyway
- Trend: file systems, drivers are moving out of Linux as it is so hard to work in-kernel
- Sooner or later, Linux will be a server mux
- I.e. a poor man's Plan 9

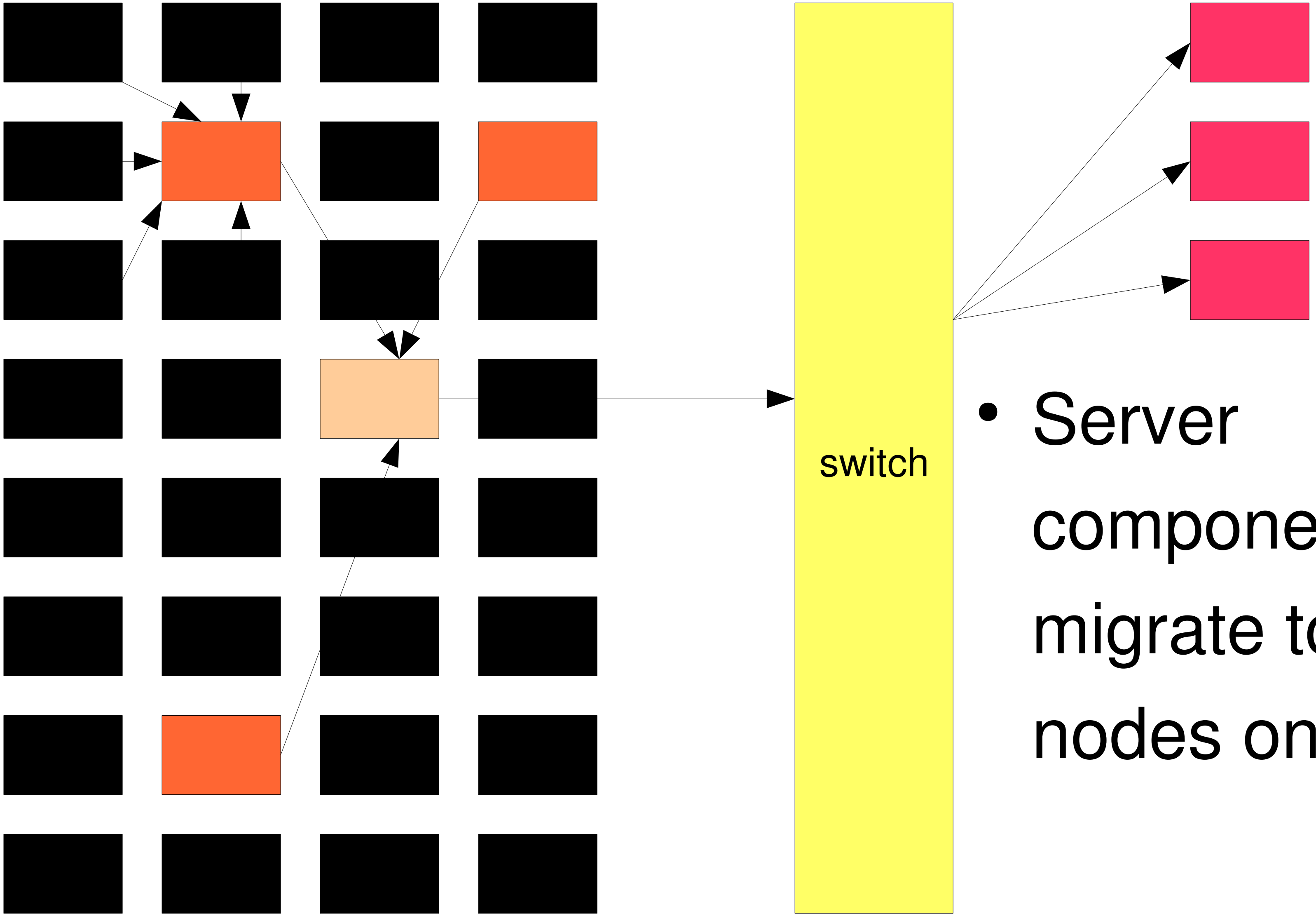
Example: CPU and file server



This structure makes no sense

- All the power, bandwidth is on the CPU side
- Need to get file server components running in the fabric!
- Dynamically activate CPUs to take on some functions of file servers, as determined *by the application*
  - *This is NOT "I/O nodes"*
- Easy given the right OS

Non-product-derived structure



- Server components migrate to CPU nodes on demand

None of the software we use functions this way

- This is a distributed system
- As our computers grow, they will resemble distributed systems, not 1992 MPPs
- We should plan for this change now
- Not continue to pretend that we can scale RHEL 5 forever

The rule

- Common interface to multiple subsystems
- The kernel's job is to multiplex process connections to servers

Exascale question:

How to run 100 *MILLION* cores?

- Just keep running Linux forever; it'll work fine
- Throw Linux away and run limited LWKs

The end of the free ride

- Linux continues to grow *exponentially*
- Up to now, piggy OS compensated for by fast clock
- From here on out, performance comes from lots of cores
- oops -- Linux will consume a growing fraction of ever-relatively-slower CPUs
- Unless you can split it into lots of parallel bits

Growth happens

- “A handful of characteristics of Unix are responsible for its resilience. First, Unix is simple: whereas some operating systems implement thousands of system calls and have unclear design goals, Unix systems typically implement only ***hundreds*** of system calls and have a very clear design” -- Linux Kernel Development, 2nd Ed. by Robert Love
- When was “hundreds” ever small?

So we just run an LWK, right?

- Something small, something simple
- As long as it can support Python
- And remote access (ssh?)
- And NFS, and xterms,
- And gdb, and Emacs
- It's 4K desktops!

So ...

- LWK works, if enhanced until it becomes Linux
  - Which is why BG/x LWK keeps growing ...
- Can Light-Weight Linux fill the bill?
  - Just a kernel and a remote exec daemon?
- LANL experience says no
- Bproc was shown to be as light as it can get
- “too light” for some users (LWK problem redux)

Plan 9 is smaller than Linux, far more capable than LWK

- Most services (e.g. file systems) run outside the kernel
- as unprivileged user processes
- And hence can be started, and controlled, by the application
- Get exactly the capability you want/need, no more

Status

- Running on BG/L with 16 man-weeks effort
  - We've run window manager on BG/L compute nodes :-)
- Port to BG/P starts 2008
- Port to XT/4 starts 2008
- Possible port to siCortex
- App port work in progress (HPCC to start)

Hence the

Slow-motion tsunami

- MegaCore systems are coming
- Today's capacity cluster is 3 years ago capability cluster is 6 years ago HPC system
- HPC systems lead clusters by about 6 years
- Growth in “node address space”:
  - .57+ bits/yr for HPC systems (14 years, 8 bits)
  - .58+ bits/yr for cluster system (16 years, 9 bits)
- What about the software?

Well, what about the software?

- Current plans are “more of the same”
- Just stack a thousand, er, thousands, er tens of thousands, err, ah, well millions of RHEL 5 desktops
- And just make everything look like a 1024-node cluster
- Yeah, that's gonna work ...

What's that look like in 2015?

- Question: do *you* want to run 1M desktops?
- Does anyone in their right mind?
- And what's happening in Linux anyway?
- Growth ...

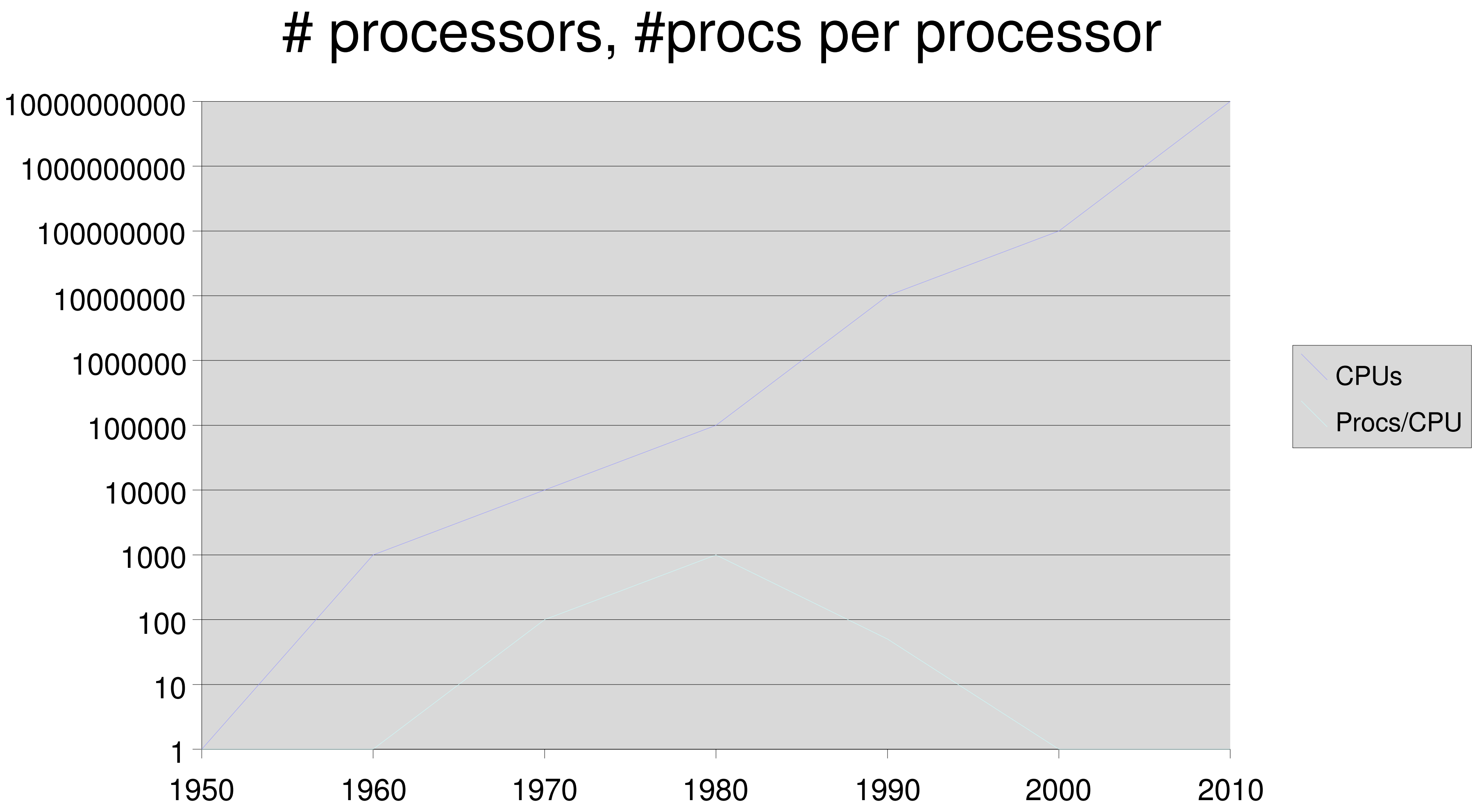
So what are we doing about it? Nothing

- A full desktop per node is unacceptable
- How long have we known? 5 years
- How long do we have? Maybe 5 years
- Will people be surprised anyway? Yes
- Slow-motion Tsunami
  - We can see it coming
  - We're still sitting on the beach drinking mai-tai's

What we can predict

- Whatever we're doing today won't work tomorrow
- If we try to freeze the structure of the software we are using, we guarantee obsolescence
  - For “freeze” substitute “standardize”
- Unfortunately, we've just done that
- Our old software dividing lines are making less and less sense

Another notional graph



What's the OS doing?

- It's time sharing
- The name even says it: “The Unix Time-Sharing System”

The promise

The reality

Time sharing CPUs?

- This thing we are drowning in?
- The whole structure of our OS is designed around something we no longer need to do
- Trend observable in 1996
- So, I proposed in 1996 that we start research in non-time-shared OSes

How you can tell you're on the right track

- “That idea is so ridiculous I won't even put it on the slide” -- Eminent computer scientist #1
- “You're proposing to take a 30 Mhz. MIPS CPU with 8 MB memory and *just let it sit idle?*!” -- Eminence #2
- And yet here we are ... with a 128,000 CPU machine with non-time-sharing OS
- Because time-share OSes are the wrong idea

That's nice, but what have you done for me lately”

- This quote could define our business
- This is the computer industry; you don't get to say “we're done”
- So, yes, it's nice that we've done some good work; now it's time to go solve some problems