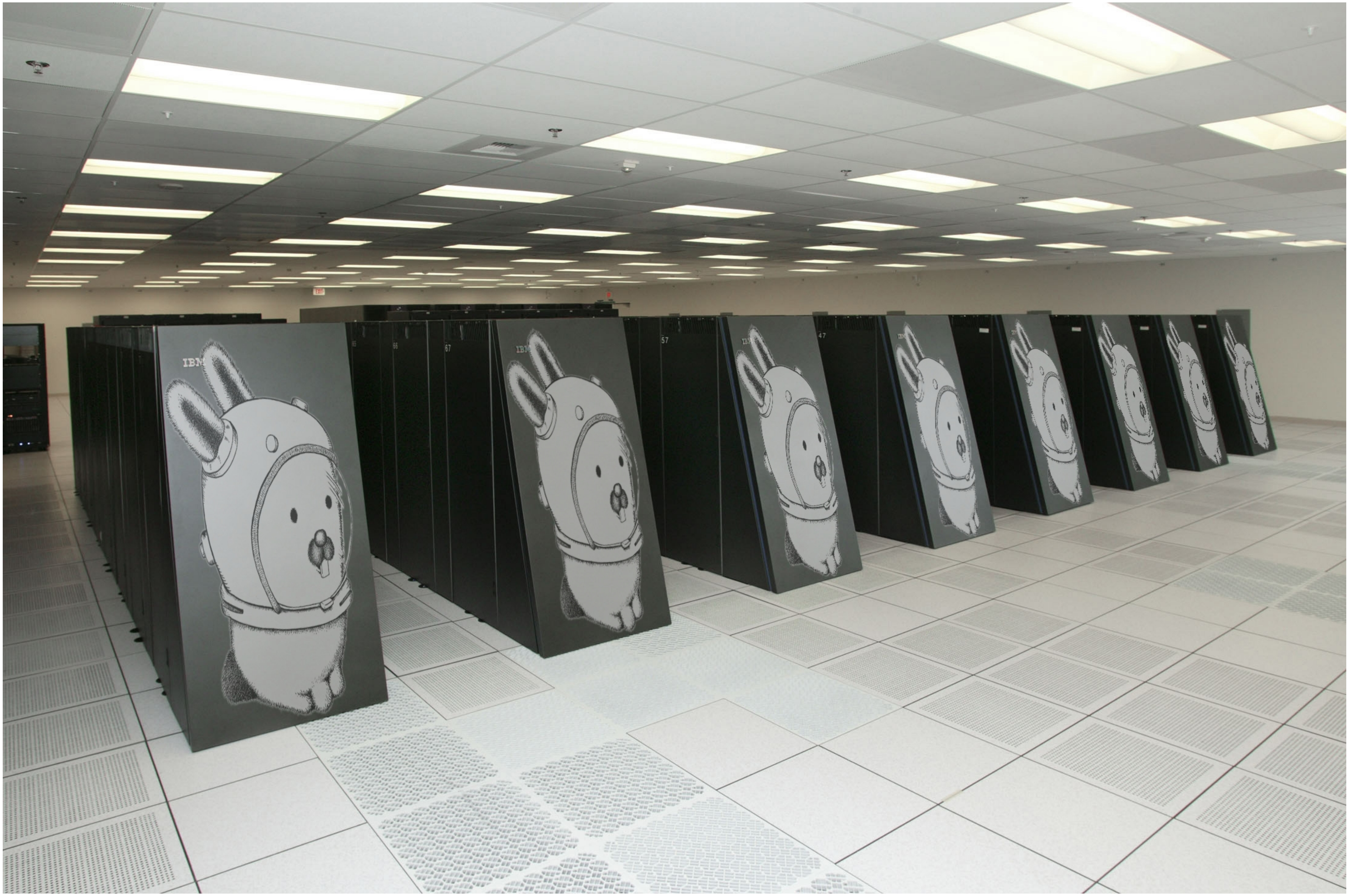
Porting Plan 9 to the IBM BG/L Supercomputer
LA-UR-06-7369
Ron Minnich
LANL

- LANL:
- Ron Minnich
- Vita Nuova:
- Charles Forsyth

- Bell Labs:
- Jim McKie
- IBM:
- Eric Van Hensbergen
- Volker Strumpen


IBM
67
57
47

Overview

- What BG/L is
- BG/L software
  - And its Limitations
- Why not just run Linux?
- What Plan 9 is
- How we are porting it to BG/L
- Conclusions

What BG/L is

- Highly integrated system: chip parts + wires
- Built from standard IBM embedded PPC macros with additional parts
  - Embedded DDR-2
  - Torus, Tree, barrier network
  - JTAG
  - EMAC4 ethernet
- Careful design at every level for reliability

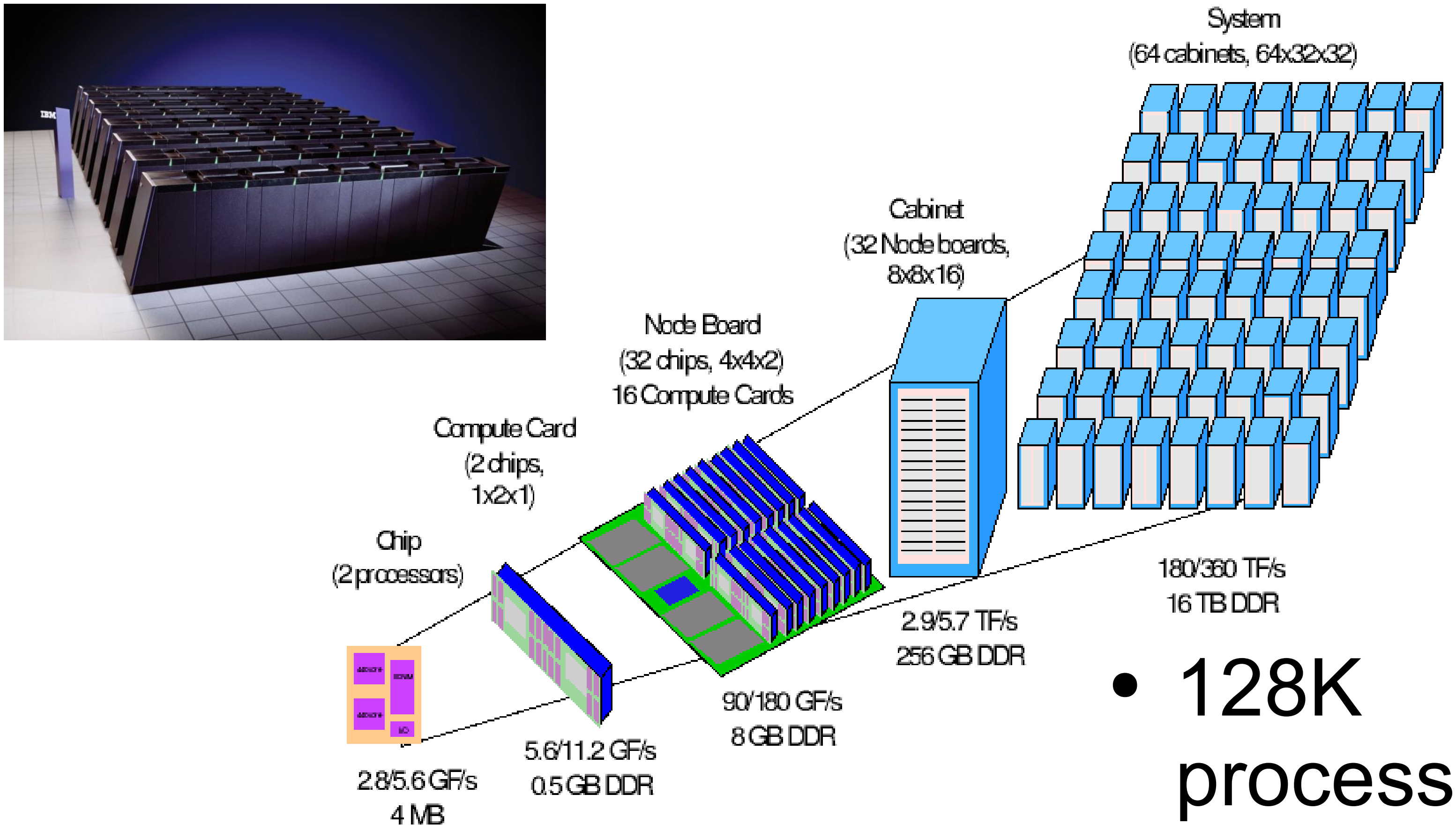


Figure 1: BlueGene/L packaging.

http://www.research.ibm.com/journal/rd/492/gara.html

- 128K processors
- But 64x32x32 is 65536!
- Original plan: self-check mode

The hardware

- CPU ea. with 2 FP pipelines
  - “Double Hummer”
- Embedded DRAM
- Shared resources between the 2 cpus
  - Lock Box
  - SRAM
  - Most importantly, the network: simple and complex
    - No DMA on any of them
    - Torus: over 3000 control and status bits

The CPUs

- Power PC 440/450
- Essentially the embedded part
- One distinguishing feature is the software-loaded TLB
- No big hardware-walked page tables to maintain
- Nice improvement over the earlier 405: there is no “real mode”
  - Always in virtual mode
  - Hardware turn-on loads “shadow TLBs” for startup

Memory

- Embedded DRAM and standard DDR2 controller
- These registers programmed via JTAG on startup as far as we can tell
- DRAM programming never changes, since it's built-in DRAM
- No OS involvement in the setup as far as we know

The networks

- Ethernet – standard EMAC4
- JTAG – control chain from control node
  - Configure on-chip resources for software
  - Load “IPL”
- Global Interrupt (or Barrier)
- Torus
  - Loads the actual kernel
- Tree

Barrier (or Global interrupt)

- Allows global “and” and “or”
  - Same operation, really; “and” is low-true
- Can set and sample input/output bits
- Can set conditions for arming interrupts
- Has an interesting “sticky” mode
- Four individual parallel networks – it literally looks like the old parallel-port hardware from PaPers
- All on a single page
- Mapped by a 256 MB TLB :-)

3D Torus (X, Y, Z)

- 6 FIFOs which can go anywhere
  - Control for same
  - + 4 virtual channels – adaptive and deterministic
- Enables for subsystems
- Complete “back door” access – to SRAM, etc.
- The enums and defines alone for this driver constitute several hundred lines
- Controlled by 108 32-bit DCRs
- And 64k of memory-mapped I/O
- Which, for convenience, is mapped into a 256 MB segment (TLB)

Torus FIFOs

- Distinguished by function and priority
  - 6 normal inject
  - 2 priority inject
  - 6 receive
  - 1 “HP receive”
- Shared between 2 CPUs
- With no interlock – if both write to a fifo, “it is an error”
  - But not detected (how could you?)
- All intended to be driven from user-controlled software!

Torus XMIT packet

- At the front: some fancy control bits, including a way to implement broadcast and priority – and support checkpoint/restart
- Hints for routing (can leave at 0)
- x,y,z coords
- Size
- Total size is <= 256 bytes
- Data fits in 256 – header size of 8 bytes
- Send data via programmed I/O (a memcpy with non-incrementing destination)

Torus receive FIFOs

- Receive FIFOs for each of the x+,x-,y+,y-,z+,z- directions plus HP (7 total)
- Times 2 for each CPU
- So note, injection is almost-shared, receive is not intended to be
- Pull data out via memcpy
- What's a receive packet look like? This is not yet clear – we assume mostly like a send packet
- BUT: we assume that receive data is perfect
- “HIPPI-like flow control”

Torus bringup

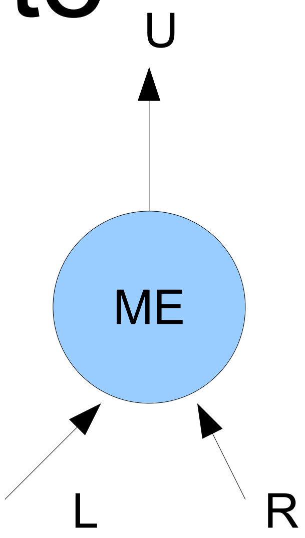
- Bring up subsystems in order defined by document
- Clear out SRAM, etc. via “back door”
- Enable state machines for transmit (and bring them out of hardware reset)
- Use the barrier network to wait for everyone to reach this point – i.e. All XMIT hardware is clean
- Then enable receive side hardware

Tree network
(Class Routed Network)

- This was confusing at first
- It has several different names – it seems the correct one is “Class Routed Network”
- Packets mostly don't have a destination
- Up to 15 different classes, with different “up/down” rules
- Simple rules for propagating packets up/down the tree – actually quite elegant
- Different routing information at each board due to differences in topology and possibly usage

Class Routing Example
16 routes, one byte per route

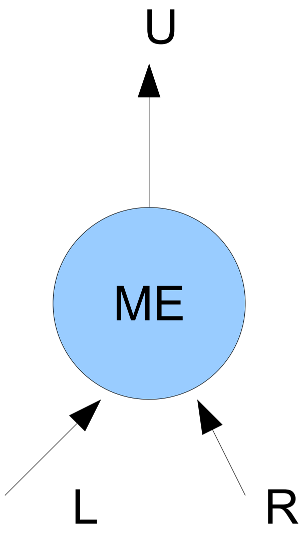
#	SOURCE				TARGET			
1	L	R	U	ME	L	R	U	ME
	1	1	0	1	1	1	0	1



- Root of a tree
- IF: SOURCE, use TARGET rule
- Anything else, use SOURCE rule
- So, at top, it comes in, we propagate to left, right, self
- I am pretty sure the source rule here is a no-op

Class routing example: intermediate node

#	SOURCE				TARGET			
1	L	R	U	ME	L	R	U	ME
	1	1	0	1	0	0	1	0



- If SOURCE, route (U)
- If NOT SOURCE, route (L,R,ME) (i.e. “down”)
- For different types/layouts of the tree, the SOURCE rules will change at different nodes (e.g. Leaf only has (ME), not (L,R,ME)
- The target rules won't

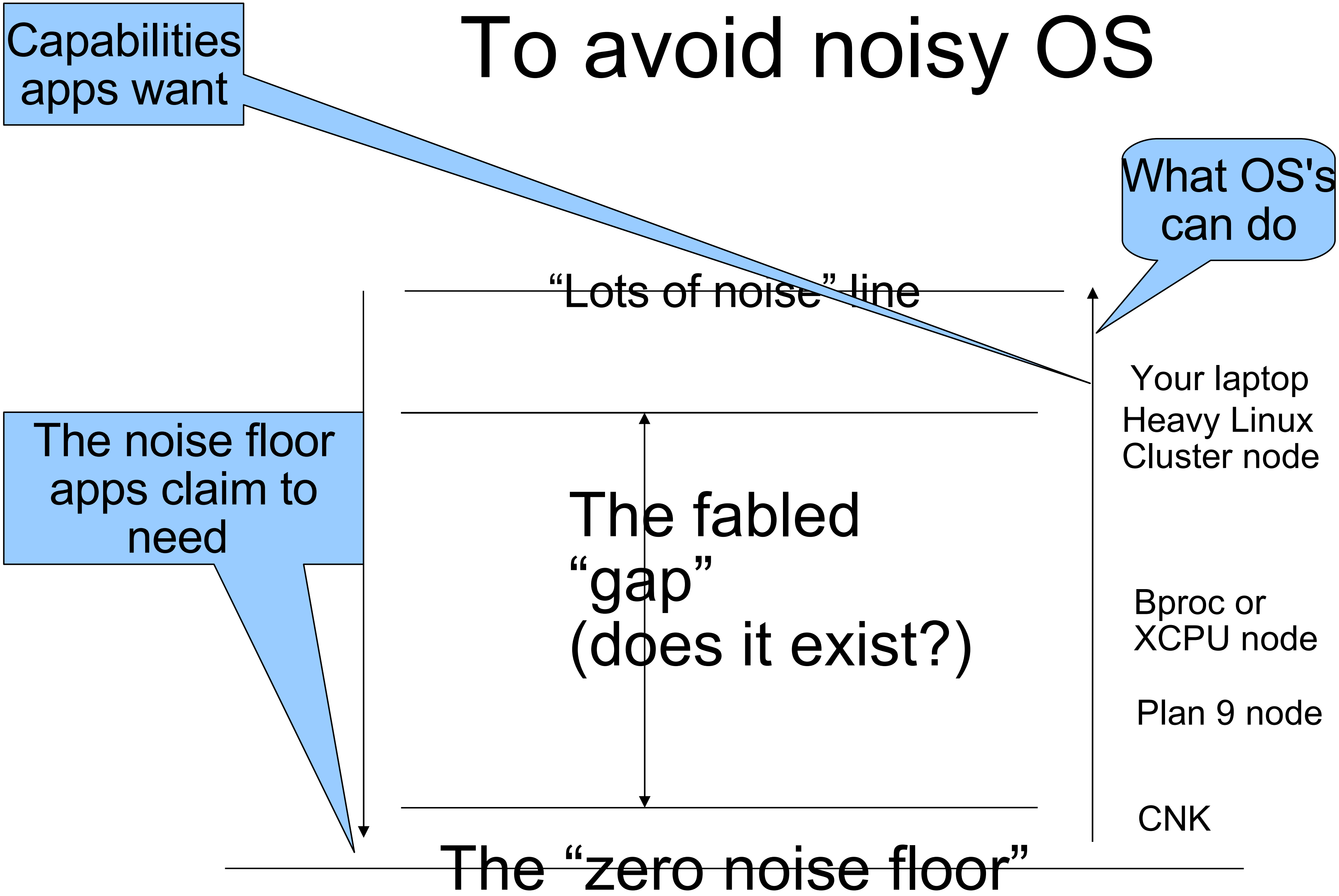
LOCK box

- 256 1-bit locks
- Read (TAS)
- Write (Set)
- Test
- Group sync
- All on one page
- 256MB address space

BG/L software

- BG/L looks like 1024-node cluster with 64 attached nodes per node (where'd that other processor go?)
- I/O nodes – run Linux kernel
- Compute nodes – run Compute Node Kernel
  - CNK
  - Runs one process at a time (on one CPU)
  - I/O and function forwarding
- A few bits missing: e.g. Sockets, mmap, ...

Why the CNK?
To avoid noisy OS



Capabilities apps want

What OS's can do

"Lots of noise" line

Your laptop

Heavy Linux Cluster node

The noise floor apps claim to need

The fabled "gap" (does it exist?)

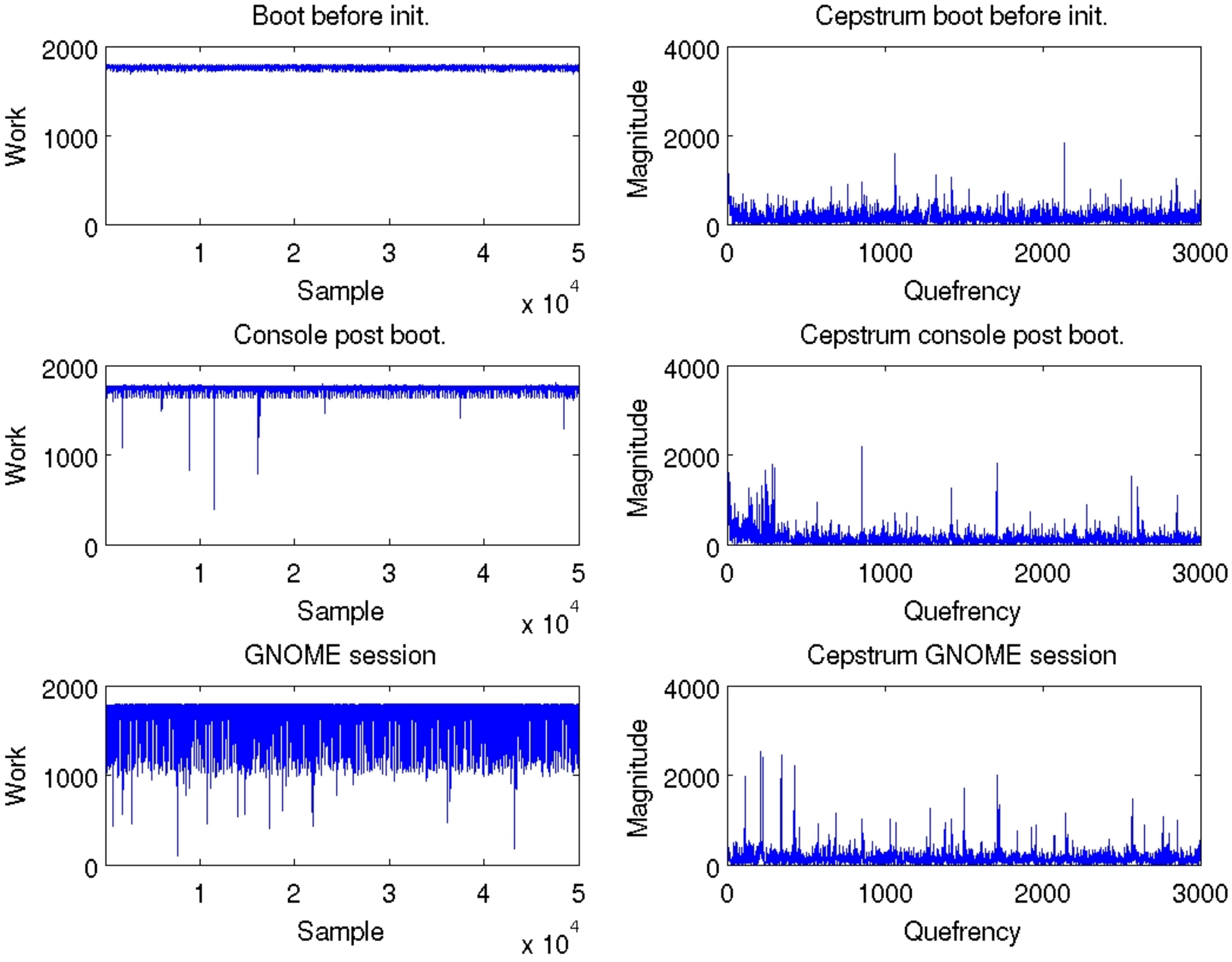
Bproc or XCPU node

Plan 9 node

CNK

The "zero noise floor"

Linux has costs



Problem 1

- The good news is, CNK is not Linux
- The bad news is, CNK is not Linux
- “What do you mean, no sockets?”
- “How do I run Python”
- “I want a REAL OS!”
- So, one might consider running Linux
- Which brings us to:

Problem 2

- The design assumption going in was that a system such as Linux would not be used
- The common way to use these resources is to let the process “have at it”
- But access to one instance (e.g. One lock) grants access to all locks
- This makes the consequences of pointer computation errors much more exciting
- Access is not fine-grained – it's no-grained
- How to allow multiple processes on such a system?

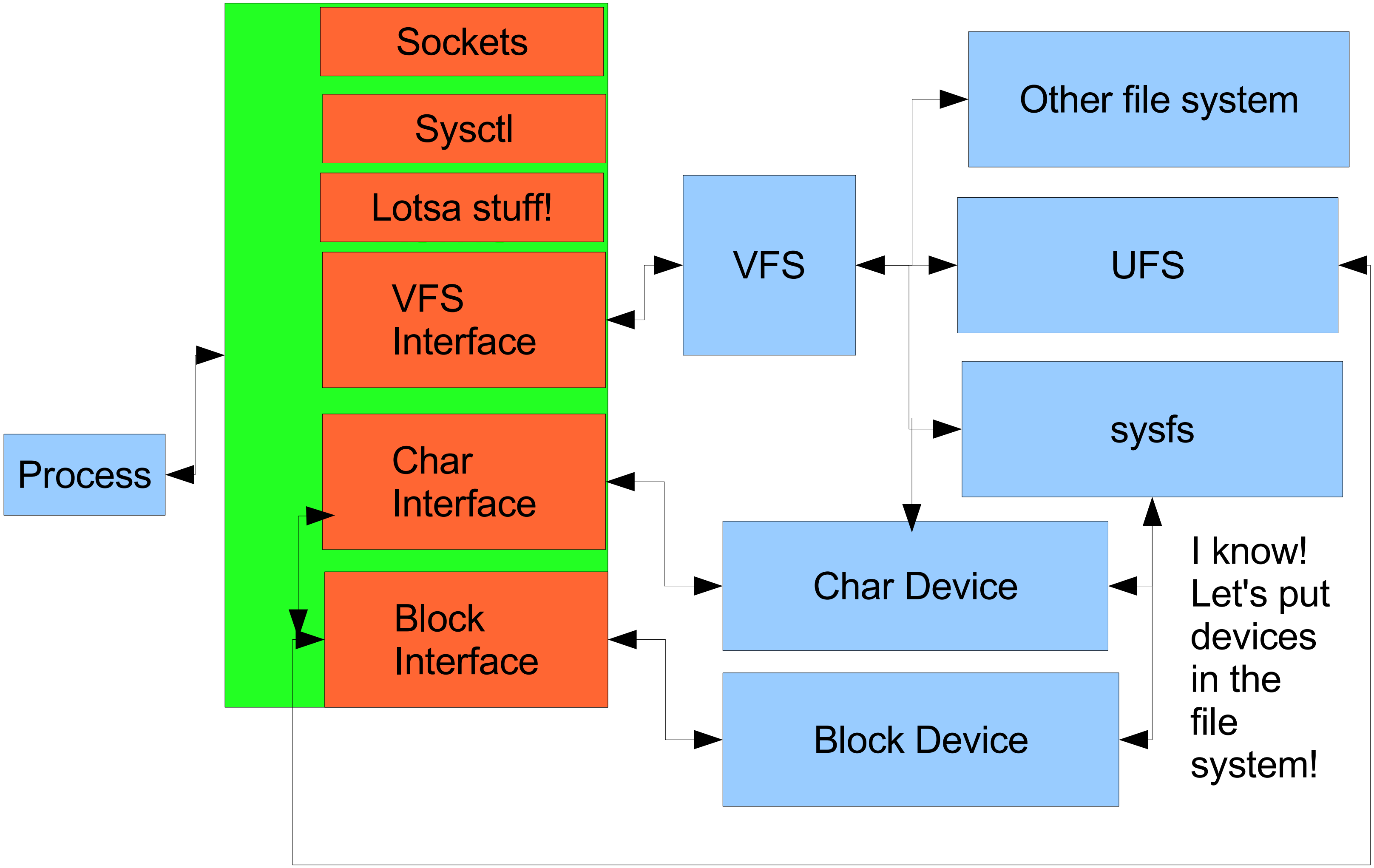
How do we keep processes separate?

- Can not separate via virtual mapping mechanisms – they're all on one page!
- It's not possible with this hardware to run more than one process in a safe manner
- Or just take your chances ... what could go wrong?

So how do we handle this?

- The single-process design of BG/L directly impacts the software
- Which is a problem for multiple processes
- We can't just let the processes directly at the networks
- After all, the destination Torus address is written to the FIFO as part of the packet
- Conclusion: on BG/L, we're going to have to go through the OS
- So are we going through Linux?

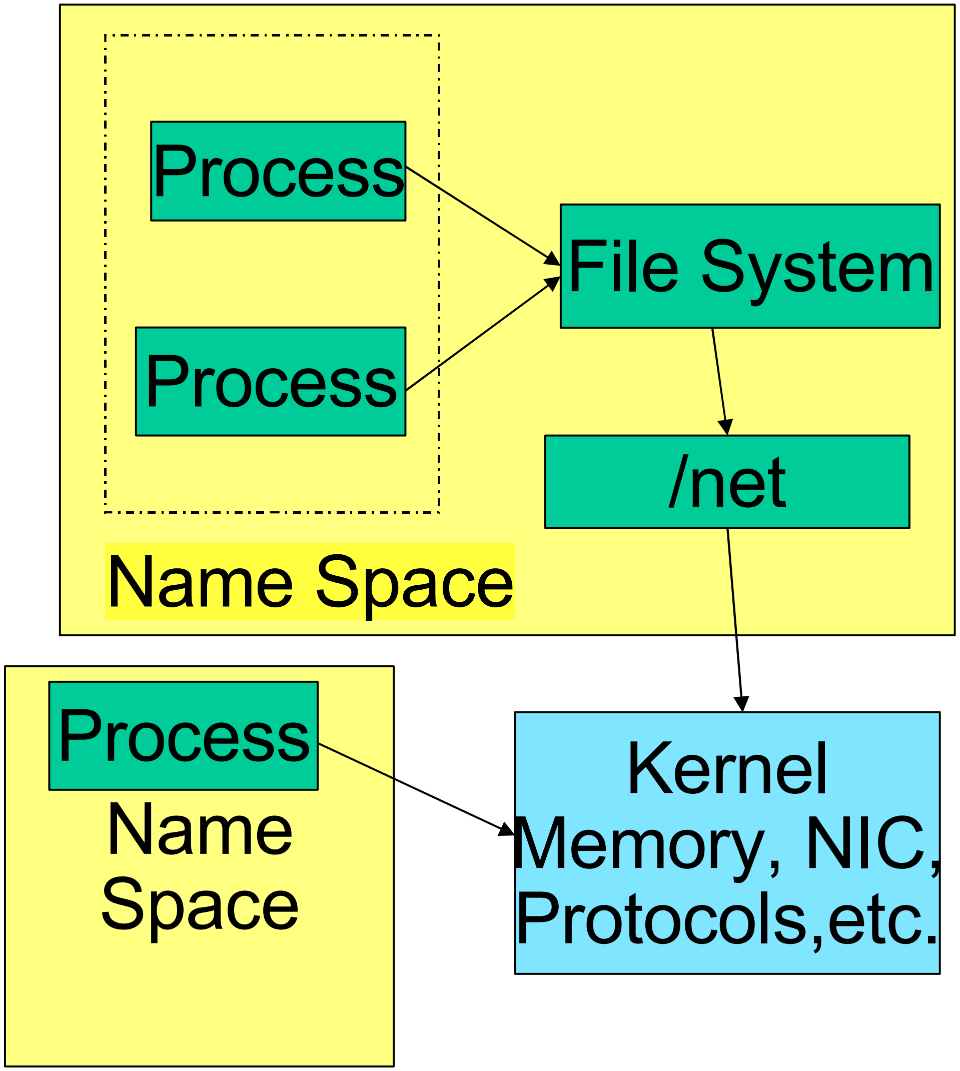
What's the Linux device path look like?



That's a long, slow path!

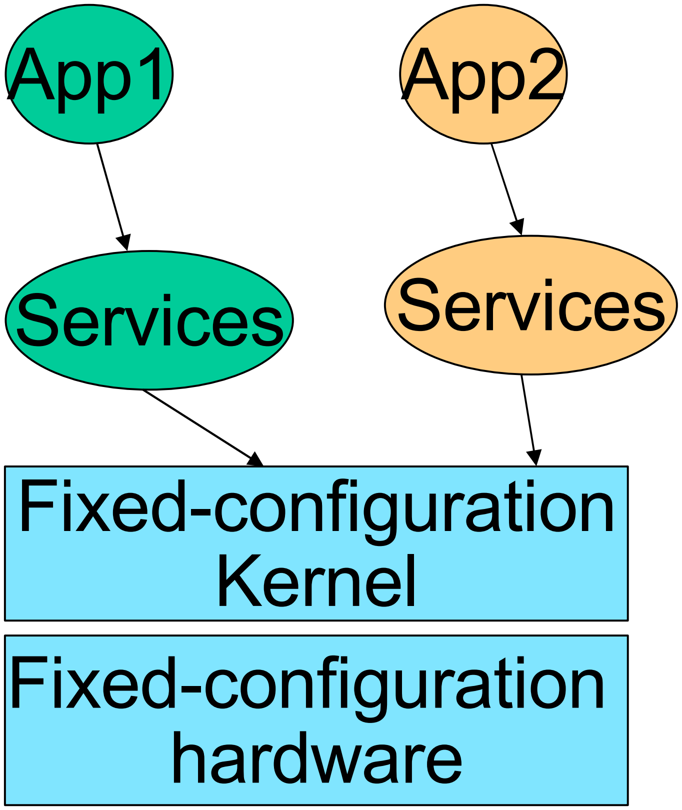
- The Linux I/O path has grown up over time
- It drank too many sugary drinks, ate too many donuts
- It's a very long path to the device
- Which is why people did OS bypass in the first place (on Unix as well)
- We can use Linux, but it's going to hurt
- We need a grown-up operating system, but a slim one

Plan 9 architecture



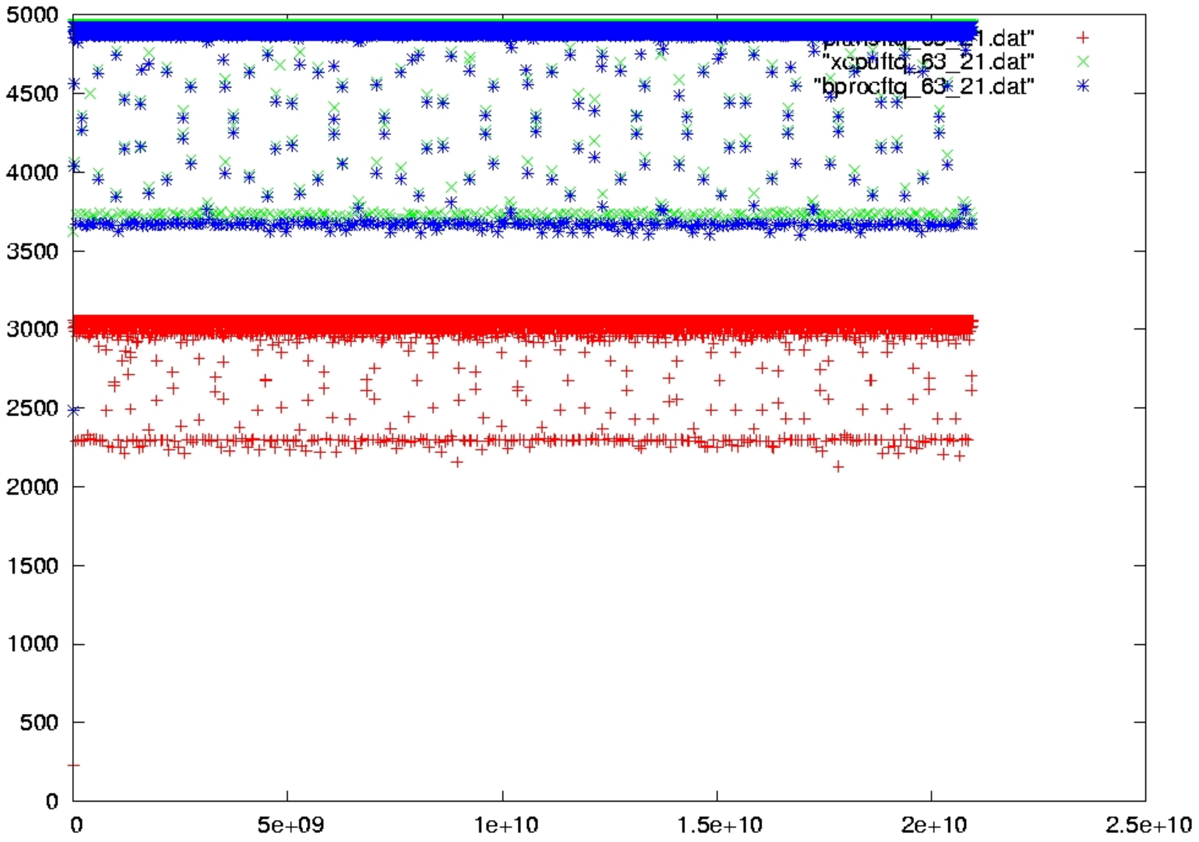
- Processes attach *servers* as needed
- Attaches are inherited
- Not visible outside the group
- In this example one group has attached remote files
- Other group only needs IPC so it has no other services
- Servers can be anywhere – no need to kludge in syscall forwarding

Why this is a good match to BG/L



- BG/L features fixed-configuration CPU/nodes
  - I.e. NO “hot plug”
- All variability is in software services used by apps
- Plan 9 fits this model perfectly
- Fixed kernel & hardware
- Customized services

Comparative noise



Why is Plan 9 so much quieter?

- It does a lot less
- The path from a user program to a device I/O is very short
- There are a lot less pointless tasks
  - Such as the 1-second timer for cursor blinking
- File systems are outside the kernel
  - And can be located anywhere

The actual port

- Start from Plan 9 port to PPC
- Forward-port to a more modern CPU: the ppc405

Port to 405

- Modify the assembly for the 405 TLB and different interrupts
- Verify on the IBM systemsim simulator
- Once that is going, boot on actual hardware
- Basically worked exactly – systemsim proved to be quite accurate – even mirrored failure modes
- But 405 is not the BG/L CPU

The PPC 440

- The actual CPU used in the BG/L machine
- Had a few differences from the 405
  - Different soft TLB layout
  - Slightly different interrupts
- Brought up again on SystemSim
- Used a commercial board for validation
- Again, SystemSim matched the hardware so well that port to SystemSim sufficed for port to hardware

BG/L and the 440

- Oops!
- No networks in the BG/L sim!
- Requires that we move forward to the BG/P sim
- And a new CPU – the PPC 450

ppc450 is in progress

- But the real important part is the network drivers
- How do we present these to the applications?
- Simply mapping them into the application is not going to work
- Instead, the OS will make them available as devices
- Which brings us back to Plan 9 and its short I/O paths

What the driver presents

```
cpu% lc /dev/bglgi
.               bglgidata     bglgiint0
0               bglgidata0    bglgiint1
bglgiarm0       bglgidata1    bglgiint2
bglgiarm1       bglgidata2    bglgiint3
bglgiarm2       bglgidata3    bglgireadmode
bglgiarm3       bglgien       bglgires
bglgiarmtype0   bglgien0      bglgires0
bglgiarmtype1   bglgien1      bglgires1
bglgiarmtype2   bglgien2      bglgires2
bglgiarmtype3   bglgien3      bglgires3
bglgiclone      bglgiint      bglgistate
```

- And in 0 we have
- 0/ctl
- 0/data
- Right now, all that happens is a write of & to 0/data does a global sync

Conclusion

- Work in progress
- Ports to CPUs done
- Barrier driver done
- Torus in progress
- Tree is next
- Then work on SystemSim BG/P to verify the drivers
- Simple SystemSim demo by SC 2006 (we hope)