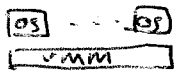
Disco

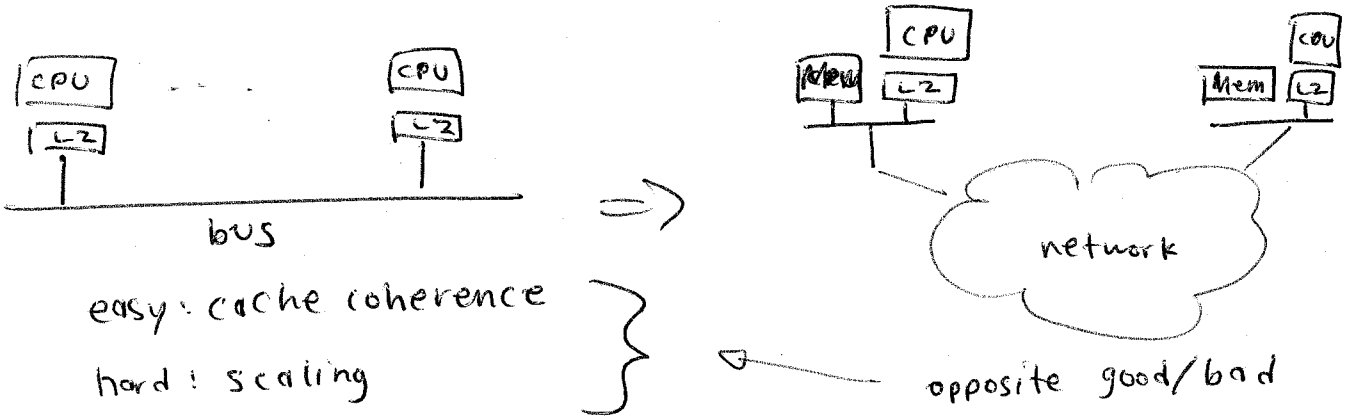
(Spr'05)

Overview

- Beautiful system, hacking, + paper!
- Illusions, Layering, ⇒ 

← deal w/ NUMA here
- Problems: Overhead, Sharing, Resource Mgmt

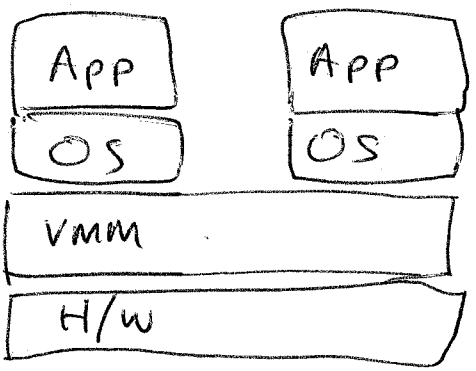
Background: CC NUMA



Question: How to build OS for CC NUMA?

⇒ Hard: Data structures in OS, page placement, etc.
OS is big/ugly/hard to change

Solution: Virtual Machines



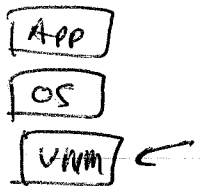
⇒ Advantages?
- Hide tough issues ("parallel") from OS
- Can run different OS's concurrently
- Portability layer

Problems:

Overhead
Sharing
Resource Mgmt
(Breaking the Illusion)

Why Overheads?

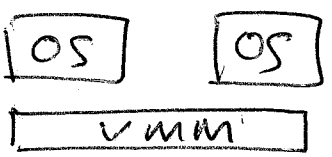
- Time



could be a simulator!

(but usually can just directly execute instrs.)

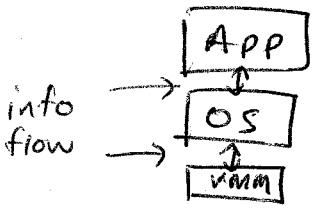
- Space



{ file cache, page tables, etc. }

Resource Mgmt Problems

- Information flow



e.g. OS in idle loop, page on free list

Sharing problems

- Old VM/370 → no file sharing allowed
- Here: Distributed Systems Technology

Disco Virtualization

CPU: MIPS R10K

"direct execution"

set real registers to VCPU registers, jump to VPC

hard: privileged instructions (TLB, phys mem, I/O)

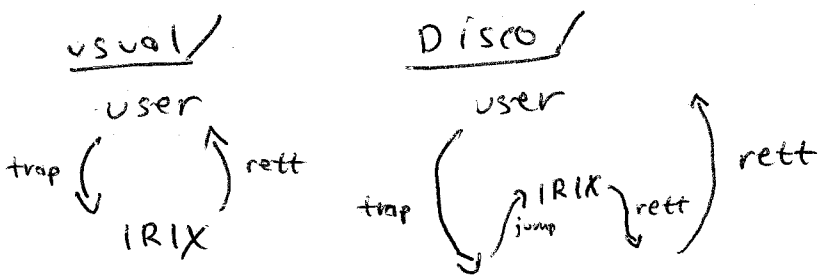
MIPS details:

	usual	Disco
user mode	App	App
supervisor	(not used)	OS
kernel	IRIX	Disco

why? MIPS wanted to support "modern" OS's → supervisor

Example:

System call: trap/rett

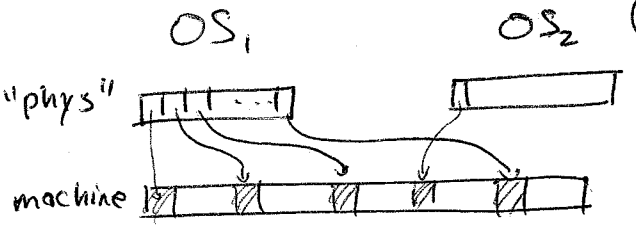


key: supervisor mode in MIPS

can access more mem than user mode, but not privileged inst, phys mem

Result: new protection boundaries in system

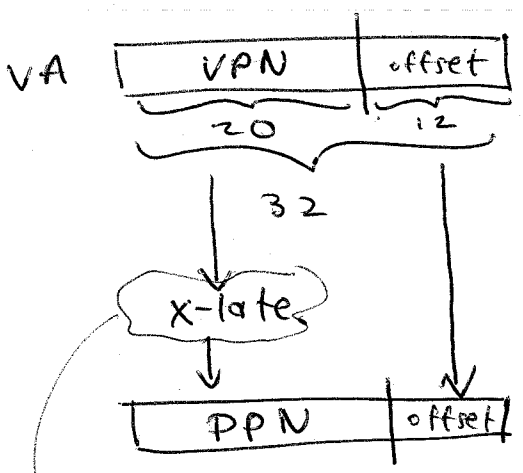
Memory



$$\underbrace{VA \rightarrow PA}_{\text{usual OS}} \underbrace{\rightarrow MA}_{\text{w/ Disco}}$$

All done thru TLB ; e.g.:

⇒ load $0(R_1), R_2$



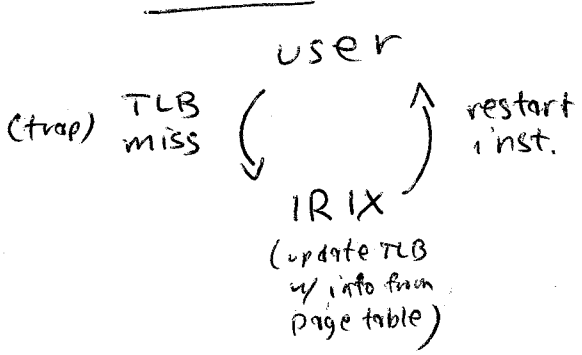
4KB page

Q: where are all x-lations held?
A: page table (just a data structure in the OS)
(VA → PA)

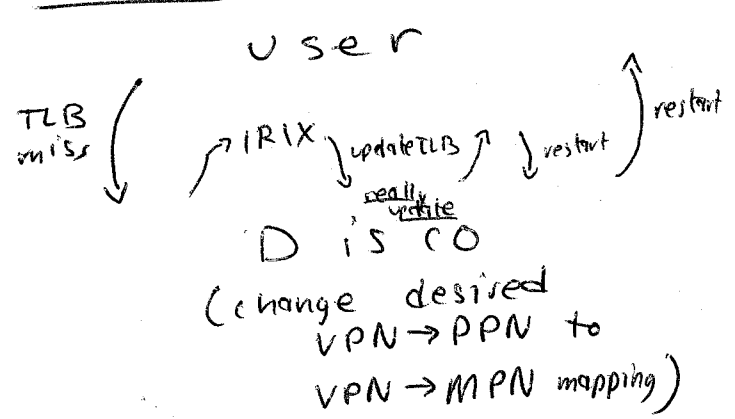
⇒ to speed up: TLB
- in hardware
- fully associative cache of VA → PA

TLB miss: "software handled" TLB

usual/



Disco/



Memory Difficulties

(KSEG0)

⇒ IRIX usually lives in unmapped physical memory?

⇒ what does this mean?
(can't interpose via TLB)

⇒ have to relink kernel
(now KTLB faults an issue)

⇒ ASID (address space identifier)

⇒ what is it?

⇒ Hard to virtualize

⇒ Flush TLB on VCPU switch

⇒ Cost of TLB miss is high

⇒ Add 2nd-level S/W TLB

NUMA issues

Key: w/ VA → PA → MA,
can move "phys" pages to handle NUMA issues

Goal: cache misses serviced from local memory

Example: when to replicate a page?
⇒ Read sharing (e.g. kernel code)

Example: when to migrate a page?
⇒ when write activity occuring from a far away CPU (e.g. scheduler moved VM)

How to do these things?
⇒ must modify TLB appropriately
migrate: invalidate
replicate: downgrade to read-only

What about heavily write-shared pages?
⇒ don't move, no point

Handling I/O (SKIP)

PIO:

usual/

memory map control registers
use loads/stores to interact
w/ device

disco/

add special driver to OS
(network, scsi)
use internal "monitor call" interface
to interact w/ I/O device efficiently

DMA:

usual/

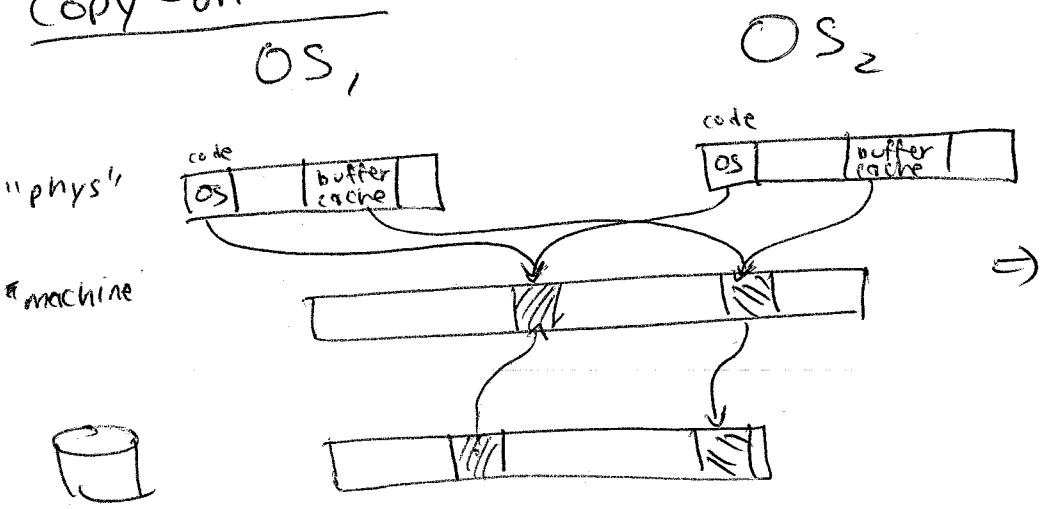
for efficient, large transfers w/o CPU

disco/

must intercept, translate PAs → MAs

Sharing across VMs

Copy-on-write

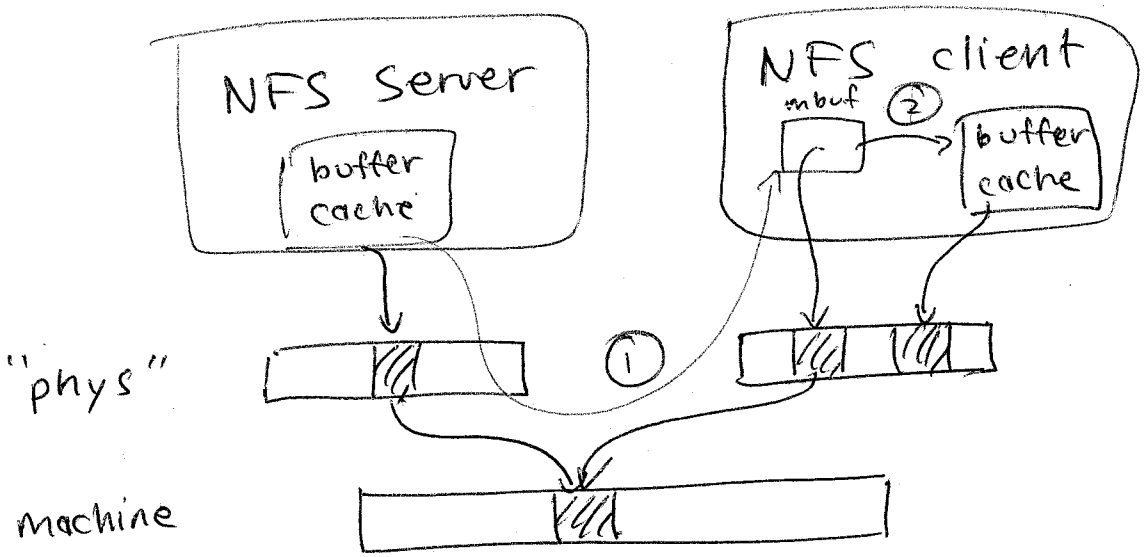


⇒ intercept DMA from block X; if already in memory, just remap PA→MA

[COW]

⇒ good for read-only sharing OS text, some of buffer cache

Network interface



1. send becomes remapping
2. but client still copies incoming packet (solve by changing client)

Breaking the Illusion

1. Some priv. ops just need to read/write regs
   => replace w/ LD/ST to mem addresses

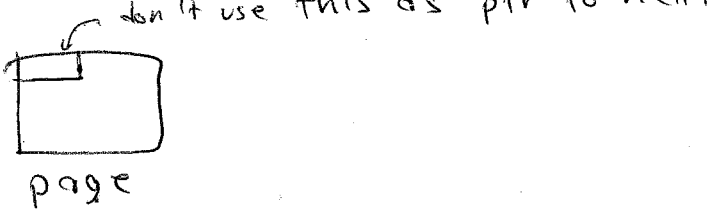
2. zeroed pages:
   => why zero @ all?
   => Problem: OS zeroes, monitor zeroes
   => Solution: OS asks monitor for zeroed page

3. Page on free list
   => OS calls monitor to tell it

4. CPU is idle
   => Disco detects low power mode
      => idleness

5. bcopy -> remap in NFS client

6. mbuf structure change



Performance #s

⇒ All simulated results (but reliable)

⇒ microbenchmarks : exec, open, write 1.6 → 2x slower

⇒ macro : apps run well

Conclude

⇒ Power of layering, illusion

⇒ Importance of info flow

⇒ Today : VMware

key : consolidation in server farms