Denali

Lightweight virtual machines for distributed and networked systems

Steven D. Gribble, Andrew Whitaker, Marianne Shaw

Department of Computer Science and Engineering

University of Washington

http://denali.cs.washington.edu

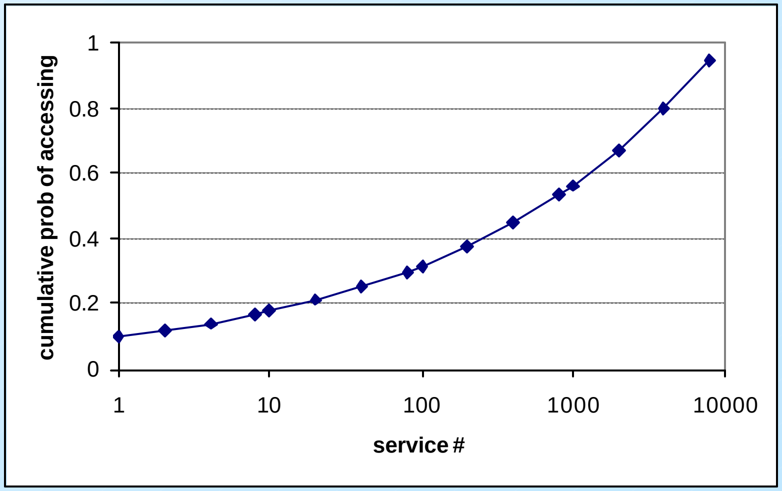
{gribble,andrew,mar}@cs.washington.edu

Content delivery: not just static anymore

- **Recent progression of content-delivery architectures**
  - CDNs, proxy caches, P2P, …
    - premise same for all: replicate static content
  - but: a large fraction of content is dynamic
    - 20-40% of web requests are to dynamic content [Wolman99]
    - these systems have, or soon will, “hit the wall”
- **Need to think about distributing dynamic content**
  - inject content-generation code into CDNs, caches
    - infrastructure completely distrusts this code
    - isolation and security challenge
      - existing research doesn’t adequately solve

Content delivery: challenges of scale

- **High degree of concurrency in caches, servers**
  - lessons from web proxy caches
    - hundreds/thousands web pages in hot set
    - O(100) simultaneous requests at any time
- **Driven by Zipfian popularity distributions**
  - 50% of access to 6% sites
  - 20% of accesses to least popular 50% of sites
  - need fast context switching!



Pushing Internet services

- **Vision for future applications: the network is computer**
  - requires scalable, available hosting infrastructure
- **Barrier to deployment of new services is high**
  - cost of physical equipment large
    - >=1 physical machine, rack space, power, admin, etc.
  - stifles grassroots service innovation
- **Ideal: push new services into virtual hosting site**
  - most will be unpopular: must multiplex large number of services
  - same isolation, multiplexing, context switching issues as before

What do these have in common?

- **Hosts must execute untrusted code**
  - need a bulletproof protection domain to isolate
- **Large degree of concurrency required**
  - protection domains must be lightweight
    - so can run hundreds simultaneously
  - fast context switching between domains
    - Zipf: implies swapping domains in/out at tail
  - implies careful control of resource mux’ing
- **Little/no data sharing between domains is necessary**
  - (possibly not true for CGIs/services backed by big DB)

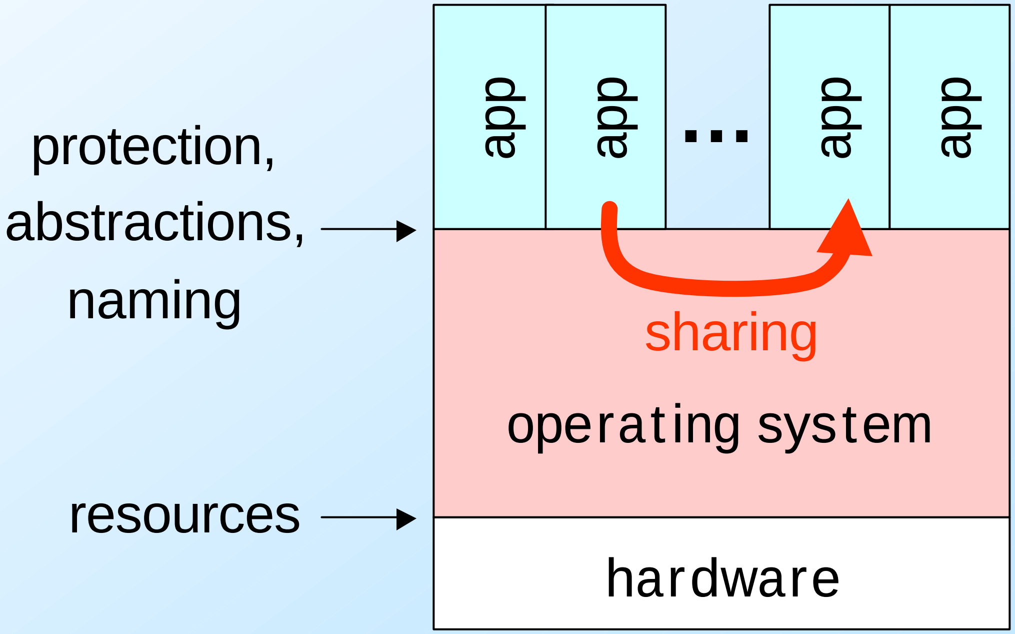
Outline

- Motivating applications
- Case for LVMs
- Core virtualization issues
- Architecture and implementation
  - paravirtualization
  - our VMM/VM architecture
- Long term plans

Conventional OS view of world

- **OS provides shared abstractions, enforces protection across applications**

What you're used to



protection,
abstractions,
naming

resources

app
app
…
app
app

sharing

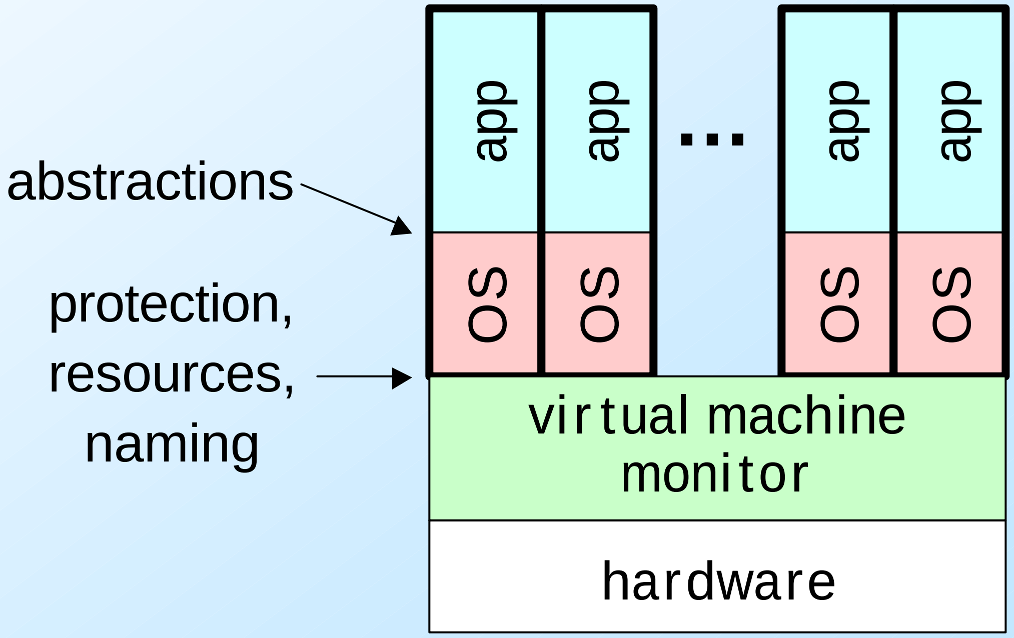
operating system

hardware

Our intended approach

- **Instead, virtualize at the HW interface level using *virtual machine monitors***

Virtual machine monitors


abstractions

protection,
resources,
naming

1. No fixed, high-level abstractions

- **High level abstractions have "layer-below" problems**
  - semantic gap between abstraction and the resources being protected below abstraction
    - shared file descriptors bypassing FS access control
    - packet sniffer capturing shared files through NFS
    - forced core dumps reveal passwords
- **Fixed abstractions make it hard to express isolation**
  - e.g., virtual address spaces are too coarse-grained
  - e.g., DB's need record-level isolation, c.f. file system
  - virtual machines: defer abstractions to higher layer
    - don't impose single protection interface on apps

2. Simple, intuitive sharing model

- **Protection can be represented by access control matrix**
  - a reference monitor enforces policy
  - two sources of security flaws:
    - badly expressed policy
    - bugs in (complex) monitor
      - monitor = OS, JRE, …

	/etc/pwd	/etc/motd
root	R,W	R,W
gribble		R

- **Virtual machines simplify both!**
  - simpler reference monitor (narrower abstractions)
  - start with **no** sharing
    - relax by allowing share-by-copy over virtual network
    - at least some hope of getting this right!
  - VMs: applications are principals, not users

3. Private namespaces

- **Global namespaces lead to many vulnerabilities**
  - e.g., aliasing: many names refer to same object
  - e.g., escalation of privilege: move to different column in matrix
- **A VM cannot name, let alone access, a resource in another VM!**
  - makes sharing impossible: so, allow virtual ethernet
    - single “choke point”, forces copies rather than access
    - switching, IDS, firewalls directly applicable
- **Virtualization is a level of indirection from HW**
  - transparently insert/change physical devices, migrate code, …

Outline

- Motivating applications
- Case for LVMs
- **Core virtualization issues**
- Architecture and implementation
  - paravirtualization
  - our VMM/VM architecture
- Long term plans

Which architecture to virtualize?

- **x86, Itanium, PowerPC, Sparc, Alpha?**
  - unfortunately, a tradeoff between simplicity and market reach
- **Many aspects of architecture to virtualize**
  - CPU
    - instruction set, registers, processor modes, SMP issues
  - Memory subsystem
    - translation hardware: segmentation, paging, TLB
    - privilege levels: user vs. supervisor, protection rings
  - I/O
    - console, disk, network, clocks, timers, and other devices
    - interrupt and exception dispatching

Instruction set virtualization

- **Definition of virtualizability (Goldberg, 1974)**
  - for efficiency, execute instructions natively
  - to protect VMM, execute VM with phys. CPU in user mode
    - "privileged" instructions must be trapped and emulated
      - e.g., accessing processor state: status registers, TLB, I/O instructions, interrupt dispatching
  - **virtualizable**: privileged instr. throw exceptions in user mode
- **x86 is not virtualizable**
  - 17 privileged x86 instructions do not trap in user mode
  - whither VMware? must be really hairy binary rewriting!

Scheduling, resource management

- **Zipf curve dominates all decisions**
  - 6-10% of concurrent machines are popular (pinned)
    - rest are unpopular, must be quickly swapped in
  - design issue: granularity of swapping?
    - phys. pages, virtual phys. pages, virtual virt. pages, or VMs?
    - VMM is unaware of resource mgmt. decisions of guest OS
      - double paging?
  - control relative resource consumption rates
    - important for isolation: CPU heavy service should not be able to overly penalize differently balanced services
    - goal: fair queueing of I/O

What guest OS should we run?

- **Remember: goal of 100’s of concurrent VMs**
  - implies cannot run stock Linux or Win2K
  - need to select/modify/build something else
    - there be dragons here
- **But: protection is now below level of OS**
  - opportunity to remove OS protection complexity
    - simplify OS design significantly
- **Also: can pick what devices to virtualize**
  - e.g., least-common-denominator NIC
    - simplfying the virtual architecture simplifies our job
    - hmm….a principle is beginning to emerge…

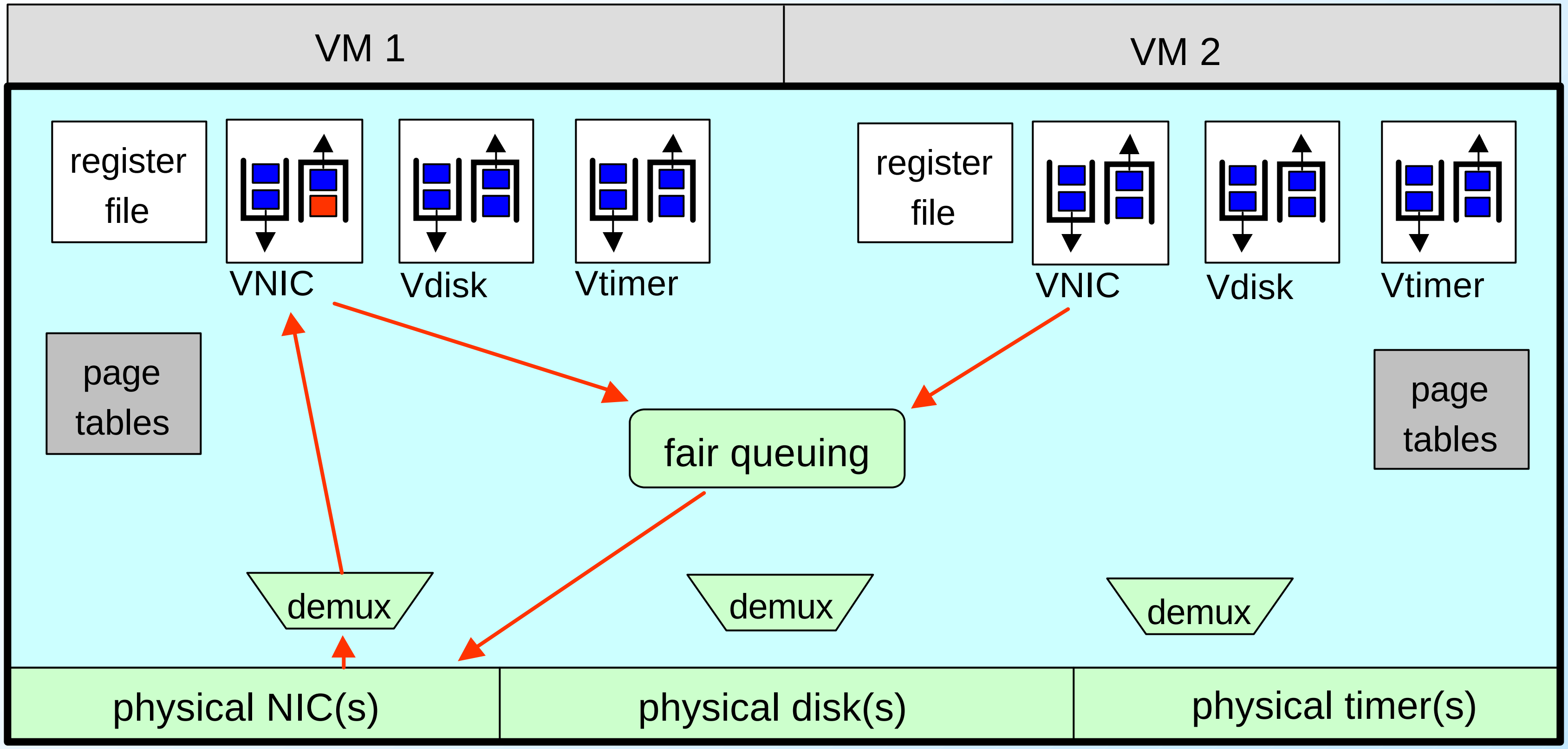
Outline

- Motivating applications
- Case for LVMs
- Core virtualization issues
- **Architecture and implementation**
  - paravirtualization
  - our VMM/VM architecture
- Long term plans

Key insight: “paravirtualization”

- **Make virtual arch. close, but not identical, to x86**
  - close for efficiency (direct execution of most instr.)
  - but, dodge all of the tough parts
    - 17 non-virtualizable instructions: semantics undefined
    - goofy processor modes: semantics undefined
    - paging, protection: not available (!!)
    - boot sequence: eliminate with simple, preinitialized devices
- **Implies cannot run stock OS on virtual architecture**
  - note: the 17 non-virtualizable insrt. are rare (~20 lines in Linux)
    - but, we didn’t want to run stock OS anyway
- **Implies cannot run guest OS on physical architecture**

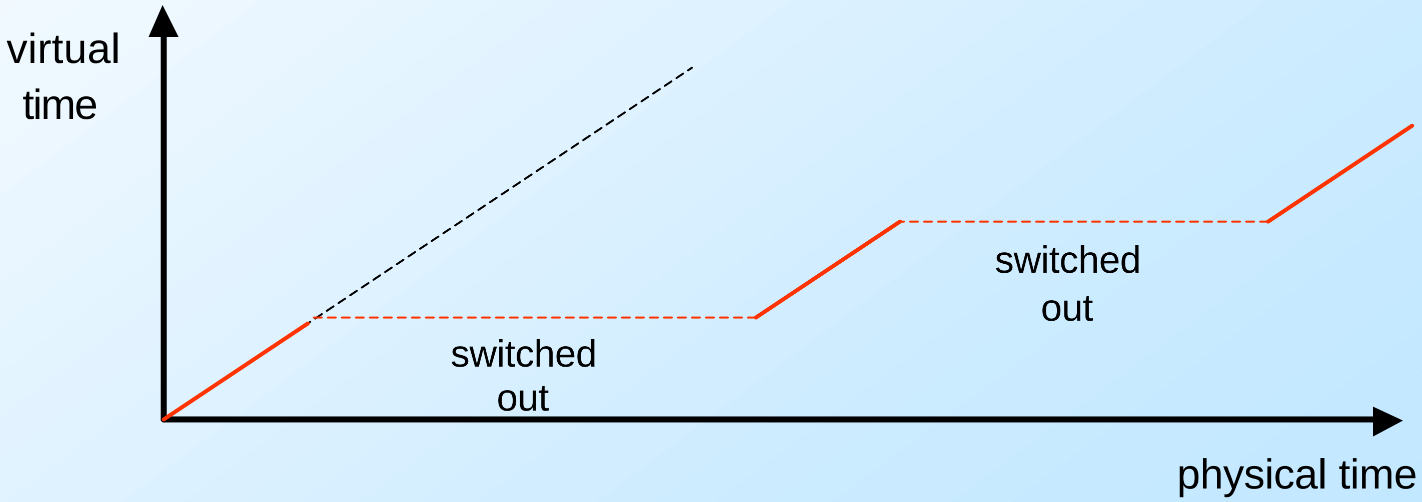
Basic VMM architecture



- **we are building our VMM on top of Flux OSkit**
  - library of C code for interacting with hardware

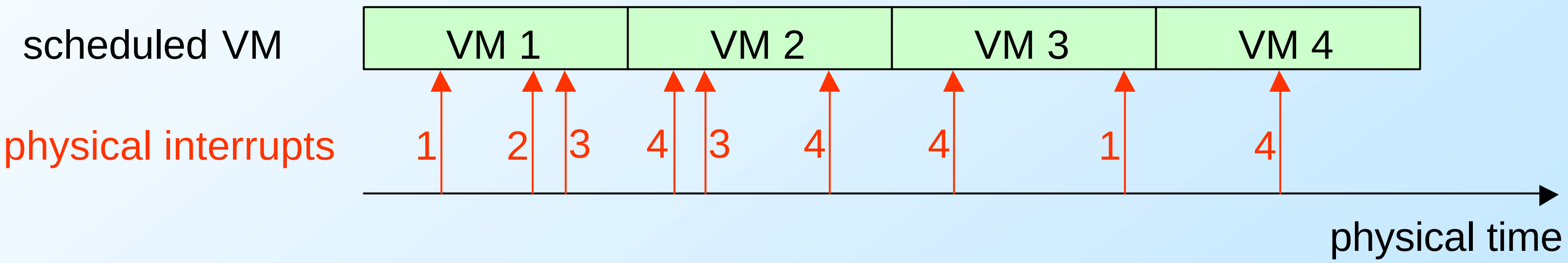
Virtual and physical time

- **Both timelines must be exposed**
  - physical: kerberos, WWW caching, TCP timeouts, ...
  - virtual: timer interrupts
- **Time from the perspective of VMs:**



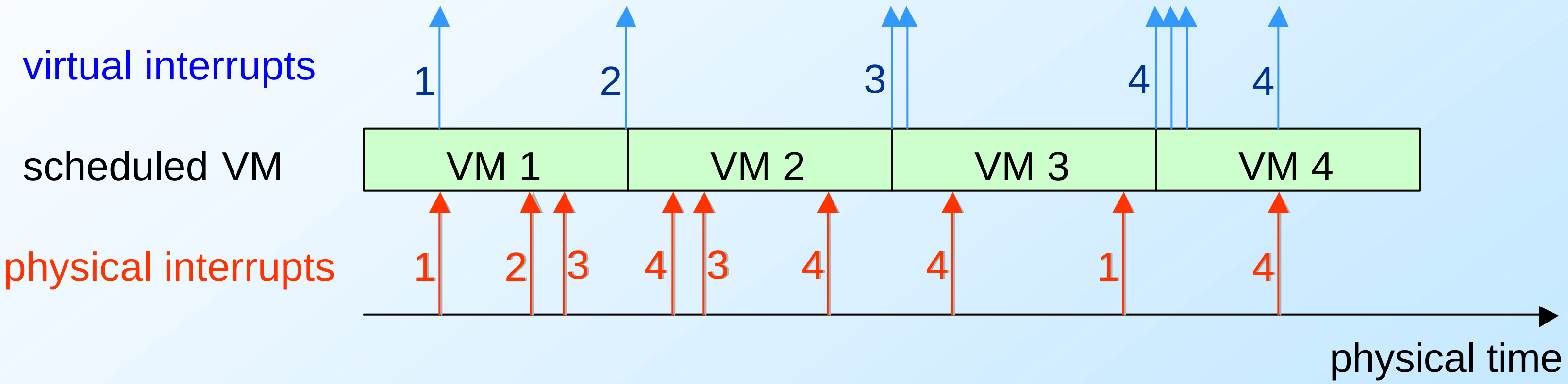
Timer interrupts

- **One virtual timer per virtual machine**
  - VMs can implement software timers if it wants more
  - question: what granularity should we offer?



Timer interrupts

- **One virtual timer per virtual machine**
  - VMs can implement software timers if it wants more
  - question: what granularity should we offer?



- **Granularity is inversely proportional to popularity**
  - happy accident: Vtimers enjoy finer granularity when VM busy

Interrupt issues

- **“Spike” on context-switch begs questions**
  - physical interrupts are synchronous w.r.t physical time
    - virtual are asynchronous
  - traditional stacked interrupts designed for synchrony
    - each results in context switch + boundary crossing
    - *notification mechanism* is conflated with *interrupt state*
- **change virtual interrupt semantics for asynchrony**
  - expose read-only bitmask of pending interrupts
    - separates interrupt state from interrupt notification
    - VM is interrupted once when this changes state
      - guest OS disables interrupts, loops until bitmask is cleared

Idleness

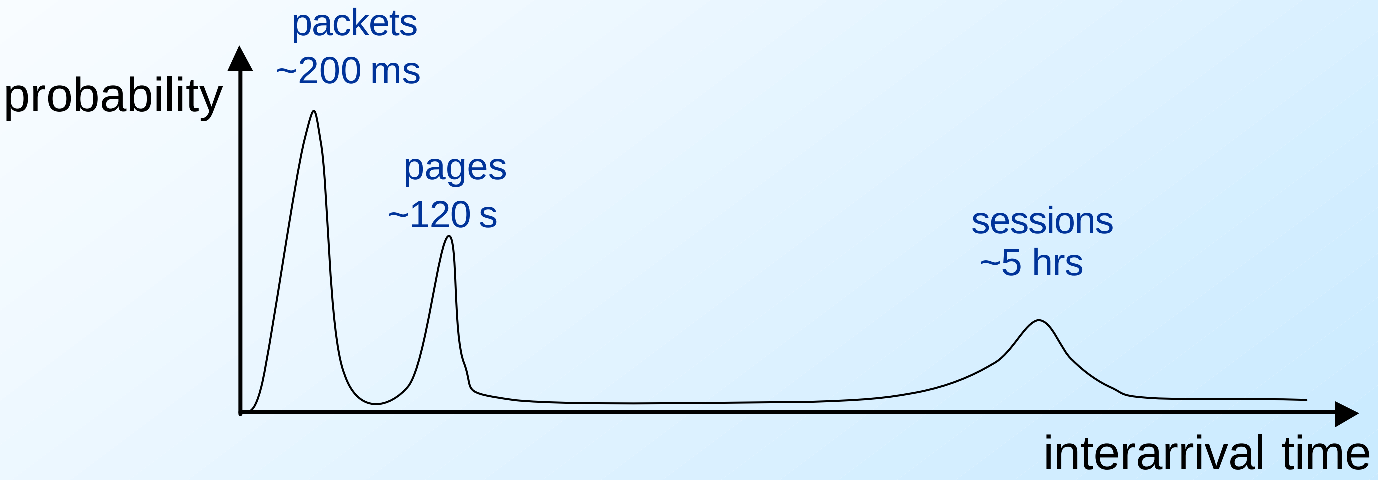
- **What about the idle loop in a guest OS?**
  - pop quiz: under what circumstances do physical CPUs stop executing instructions?

Idleness

- **What about the idle loop in a guest OS?**
  - pop quiz: under what circumstances do physical CPUs stop executing instructions?
    - power off, suspend, slow down in low-power mode
  - invariant: the only idle loop consuming physical CPU cycles should be VMM's
    - add "idle" instruction to virtual ISA
      - semantics: suspend VM until a new interrupt arrives
    - not doing this hurts massively
      - aggregate throughput drops with # of VMs

Guest OS must be aware of VMM

- **Consider packet interarrival of an unpopular service**
  - e.g., a web session every 5 hours



- **unpopular services must turn off periodic timer interrupts between "pages" and "sessions"**
  - to avoid being continually swapped in

“Fast boot” is a requirement

- **Issue: mechanics of swapping VMs in and out**
  - is it “APM suspend/restore”, or a “shutdown/reboot”?
    - tradeoff betw. performance and software rejuvenation
      - if suspend/restore, memory leaks are not cleaned up
      - if shutdown/reboot, pay price of OS and device restart
  - plan: suspend/restore most of the time, occasional shutdown/reboot
    - paravirtualization helps here too: devices start in initialized state, boot sequence is minimal

Supervisory VM

- **Idea: have one trusted, powerful VM**
  - ability to start, stop, monitor, migrate VMs
  - console + UI for controlling VMM
  - contains allocation policy of physical resources to virtual machines
- **Why put in supervisor VM instead of VMM?**
  - keeps VMM simple and effectively stateless
    - e.g., no TCP stack in VMM
  - separates supervision policy from virtualization mechanism

LibOS architecture

- **Push paravirtualization all the way**
  - virtual architecture doesn't support protection, virtual memory
    - no paging → single-address space for guest OS + app(s)
    - OS becomes a library (similar to exokernel libOS)
  - simple user-level threads package
- **our first libOS is designed for web services**
  - Alpine user-level network stack
    - BSD stack, with OS dependencies "stubbed out"
    - malloc, timer, packet xmit/rcv
  - we're shopping for a simple user-level FS for read-mostly data
  - anticipate a large set of VMs using the same libOS
    - share its code pages copy-on-write across VMs?

Some “freebies”

- **Can imagine clever virtual hardware devices**
  - copy-on-write disks, non-persistent disks
    - safely share read-only data across VMs
  - append-only log disks
    - LFS without the cleaner
- **Checkpoint / migration / recovery for free**
  - simple to capture entire machine state
    - once you can capture it, you can move it, copy it, etc.
    - all underlying hardware names are virtual
  - can even hot swap physical hardware under VMs!

Outline

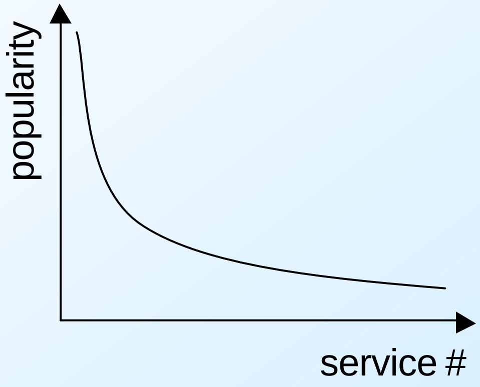
- Motivating applications
- Case for LVMs
- Core virtualization issues
- Architecture and implementation
  - paravirtualization
  - our VMM/VM architecture
- **Long term plans**

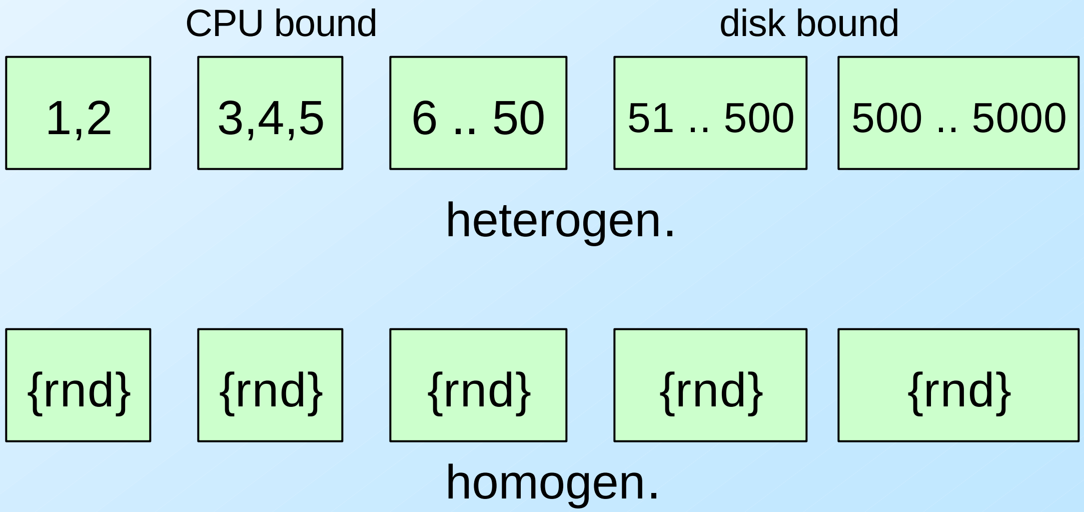
Virtual clusters

- **virtual clusters within a physical cluster**
  - VMs offer multiple levels of resource allocation and containment
    - fair queuing and quotas inside one node’s VMM
    - cloning virtual machines across cluster nodes
      - migration can become a load balancing and resource management mechanism
  - goal: have VMMs cooperate across nodes to build virtual clusters

Placement of VMs inside a cluster

- **goal: a balanced use of physical resources that obtains max throughput at min cost ($)**
  - open question: homogenous cluster, or heterogeneous cluster with specialized nodes?





CPU bound

disk bound

1,2	3,4,5	6 .. 50	51 .. 500	500 .. 5000

heterogen.

{rnd}	{rnd}	{rnd}	{rnd}	{rnd}

homogen.

Largest open issue

- **What if service/CGI relies on a large DB?**
  - partition DB and ship slices?
    - works well for mass-customization or geographic locality
  - copy entire DB, share amongst many VMs?
    - define “views” over DB as isolation mechanism
  - resort to accessing DB remotely over WAN?
    - negates most of benefit of shipping code
    - perhaps demand-load views of DB?

On-demand loading of VMs

- **Wide-area system of demand-loaded VMs**
  - similar to caching hierarchy or CDNs
  - instead of demand-loading content, demand load an entire VM
    - same issues as cache systems, but with larger images (5-10MB instead of 5-10KB)
  - one other wrinkle: what if the content-generation code relies on a large DB?
    - either copy the DB over, or access master copy over WAN?

Final thoughts

- **Para-virtualization blurs the lines**
  - OS / process vs. VMM / [VM:libOS]
- **some key distinctions:**
  - namespace isolation
    - no sharing of resources between VMs
  - no “layer below” issues
    - why we don’t have TCP/IP stack in VMM
  - only state in VMM is virtual device emulation state
    - simplifies migration

Questions?