Variations of Process Abstractions

- "Solaris Zones: Operating System Support for Consolidating Commercial Workloads"

  2004 LISA XVIII – November 14-19, 2004 – Atlanta, GA

Problem

Within many IT organizations, driving up system utilization (and saving money in the process) has become a priority. In the lean economic times following the post dot-com downturn, many IT managers are

electing to adopt server consolidation as a way of life. They are trying to improve on typical data center server utilizations of 15-30%

- Context:
  - Hardware supported virtualization was still restricted to specialized servers
    - Intel VT-x release 2005
  - Software virtualization had significant overheads
    - Memory footprint of multiple operating systems
    - Lack of sharing
    - Performance penalty for emulating I/O

Practical Barriers

- Server-class applications written assuming a machine to itself
  - Clashing network ports
  - Clashing user IDs
  - Hard-coded log/config file locations
- One application should not interfere with another

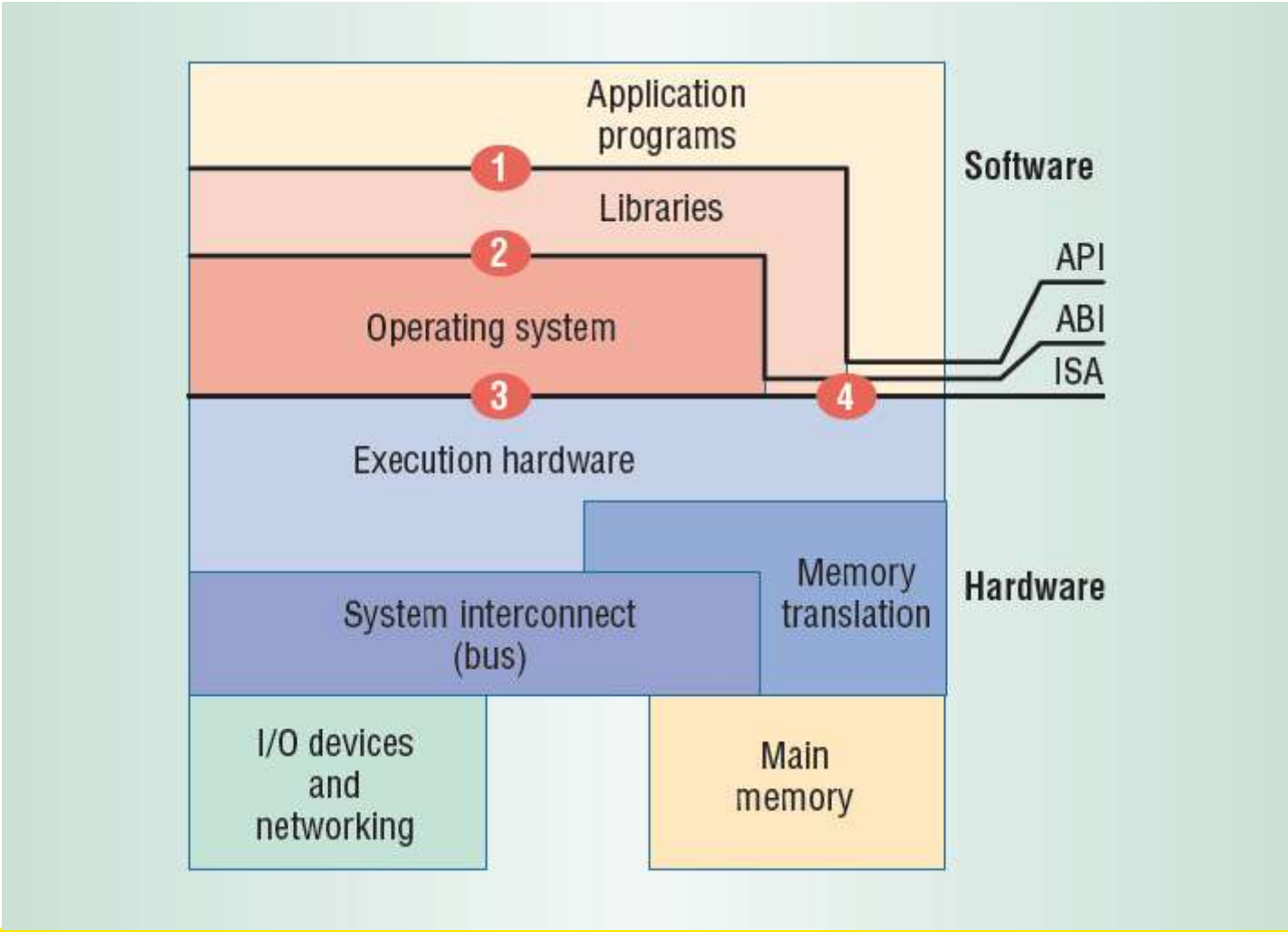
Security Issues

- Runs as ‘root’
  - How to run two mutually distrusting applications?
- Administration requires root
  - What about mutually distrusting administrators?
- Root for one application environment should be less than root for the machine

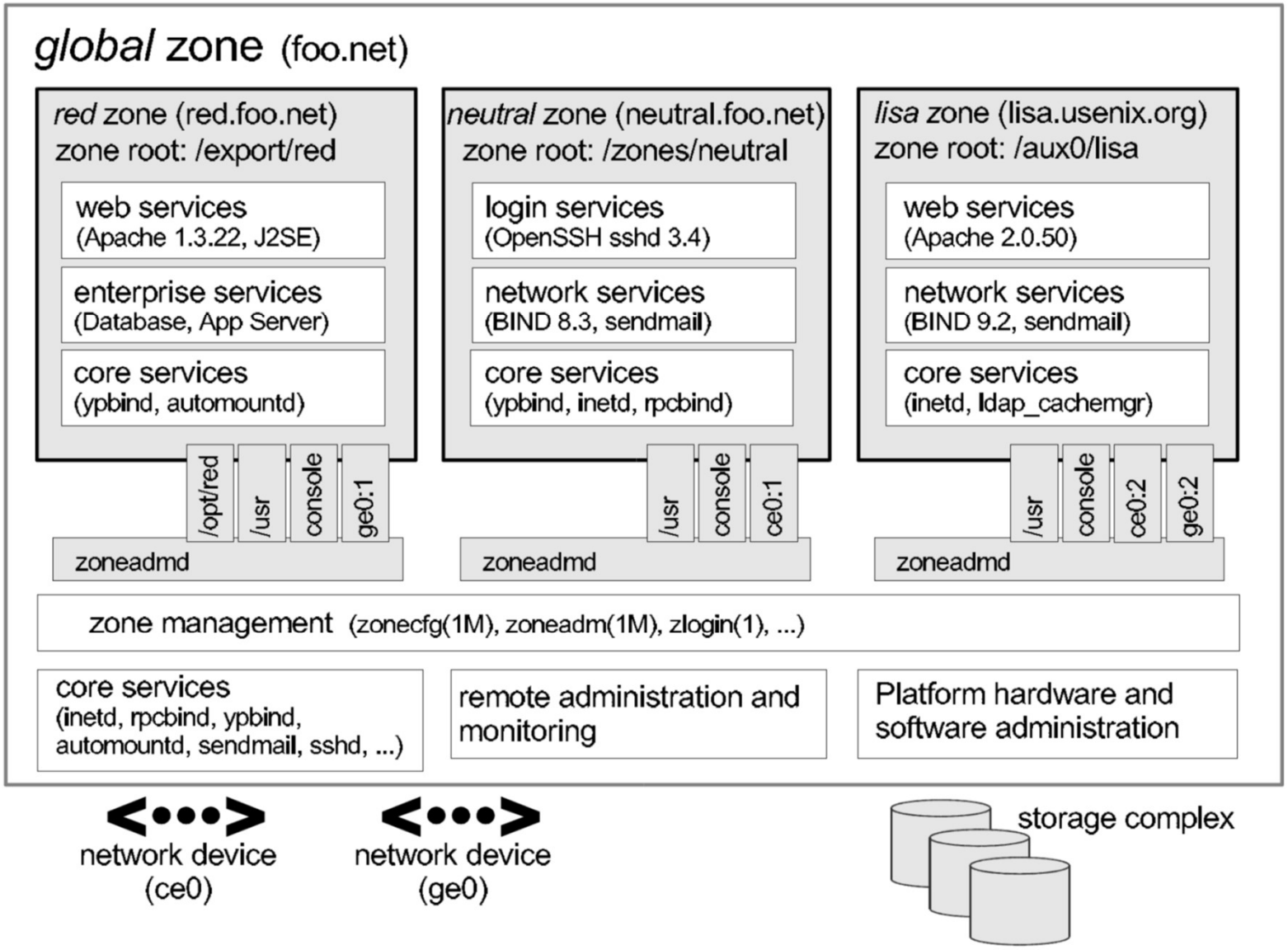
Solaris Zones

- A baked in solution
  - Part of the operating system
- “Applications can be run within zones with no changes, and with no significant performance impact for either the performance of the application or the base operating system”
- Virtualises user-kernel boundary (not the hardware platform)

Interface Levels



Overview



Design Requirements

- Each zone can provide a rich (and different) set of customized services, and to the outside world, it appears that multiple distinct systems are available.
- Each zone has a distinct root password and its own administrator.

Design Requirements

- Basic process isolation;
  - A process in one non-global zone cannot locate, examine, or signal a process in another zone.
- Each zone is given access to at least one logical network interface;
  - applications running in distinct zones cannot observe the network traffic of the other zones even though their respective streams of packets travel through the same physical interface.
- Finally, each zone is provided a disjoint portion of the file system hierarchy, to which it is confined.

Design Requirements

- The *global* zone encloses the three non-global zones and has visibility into and control over them.
- Practically speaking, the global zone is not different from a traditional UNIX system;
  - root generally remains omnipotent and omniscient.
  - The global zone always exists, and acts as the ‘‘default’’ zone in which all processes are run if no non-global zones have been setup

To address these design principles, we divided the zones architecture into five principal components.

- A state model that describes the lifecycle of the zone, and the actions that comprise the transitions.
- A configuration engine, used by administrators to describe the future zone to the system. This allows the administrator to describe the ‘platform,’’ or those parameters of the zone that are controlled by the global administrator, in a persistent fashion.
- Installation support, which allows the files that make up the zone installation to be deployed into the *zone path*. This subsystem also enables patch deployment and upgrades from one operating system release to another.
- The *application environment*, the ‘‘sandbox’’ in which processes run. For example, in Figure 3 each zone’s application environment is represented by the large shaded box.
- The virtual platform, comprised of the set of platform resources dedicated to the zone.

Specifics

- Process Model
  - Per-zone namespace with no visibility between non-global zones
- Accounting
  - Legacy accounting formats made it tricky, modified accounting to be intra-zone.
- Networking
  - Global zone multi-homed server
  - Each IP associated with a specific zone

Specific

- Filesystem
  - Use loopback filesystem to mount part of global filesystem namespace
  - High degree sharing
- Device
  - Generally discouraged
    - Device semantics generally unclear
      - Compare /dev/null to /dev/kmem
  - /dev/log an exception

Resource Management

- CPU
  - Global fair scheduler can schedule zones
  - Scheduler within a zone can further share
- Memory still to come ☺

Performance

Workload	Base	Zone	Diff (%)
Java	38.45	38.29	99.6
Time-sharing	23332.58	22406.51	96.0
Networking	283.30	284.24	100.3
Database	38767.62	37928.70	97.8

Figure 7: Performance impact of running in a zone.

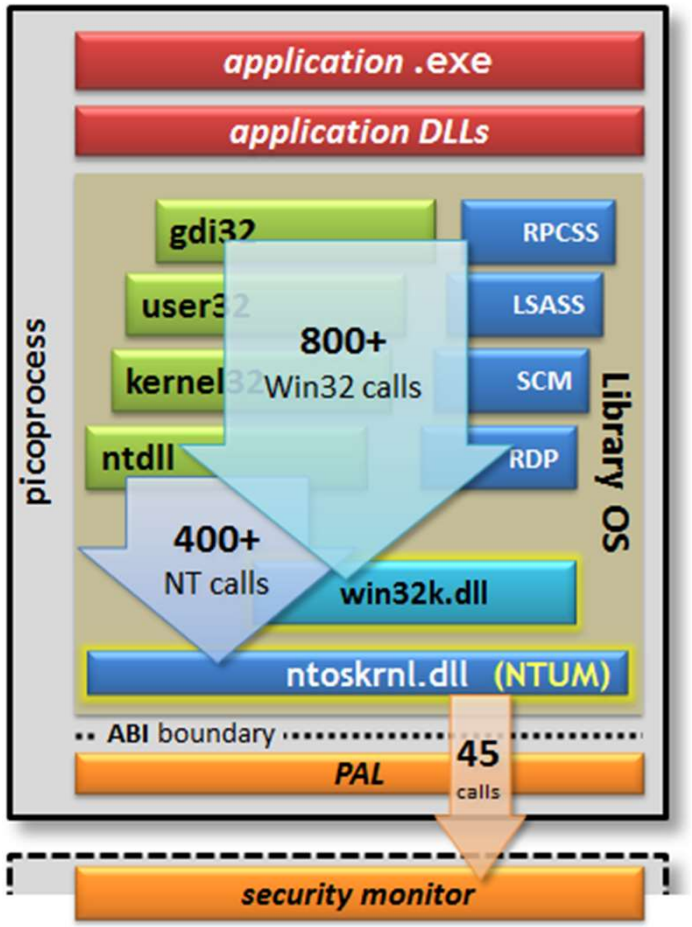
- Timesharing workload related to loopback file system

Drawbridge

- Donald E. Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C. Hunt. 2011. Rethinking the library OS from the top down. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems* (ASPLOS XVI).

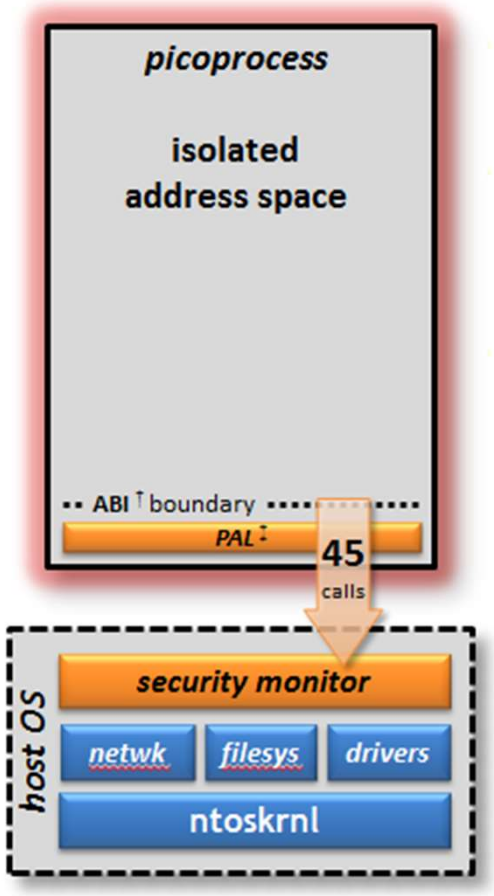
Library OS

- OS refactored to run in the context of the application
  - From the application perspective it looks like an OS (Windows in this case)
- The underlying OS API is smaller
  - Easier to get correct
  - Easier to make secure



Pico process

- An isolated process with different system call interface to normal processes
  - A security monitor in this case
  - Could be something else?



Windows Subsystem for Linux (WSL1)