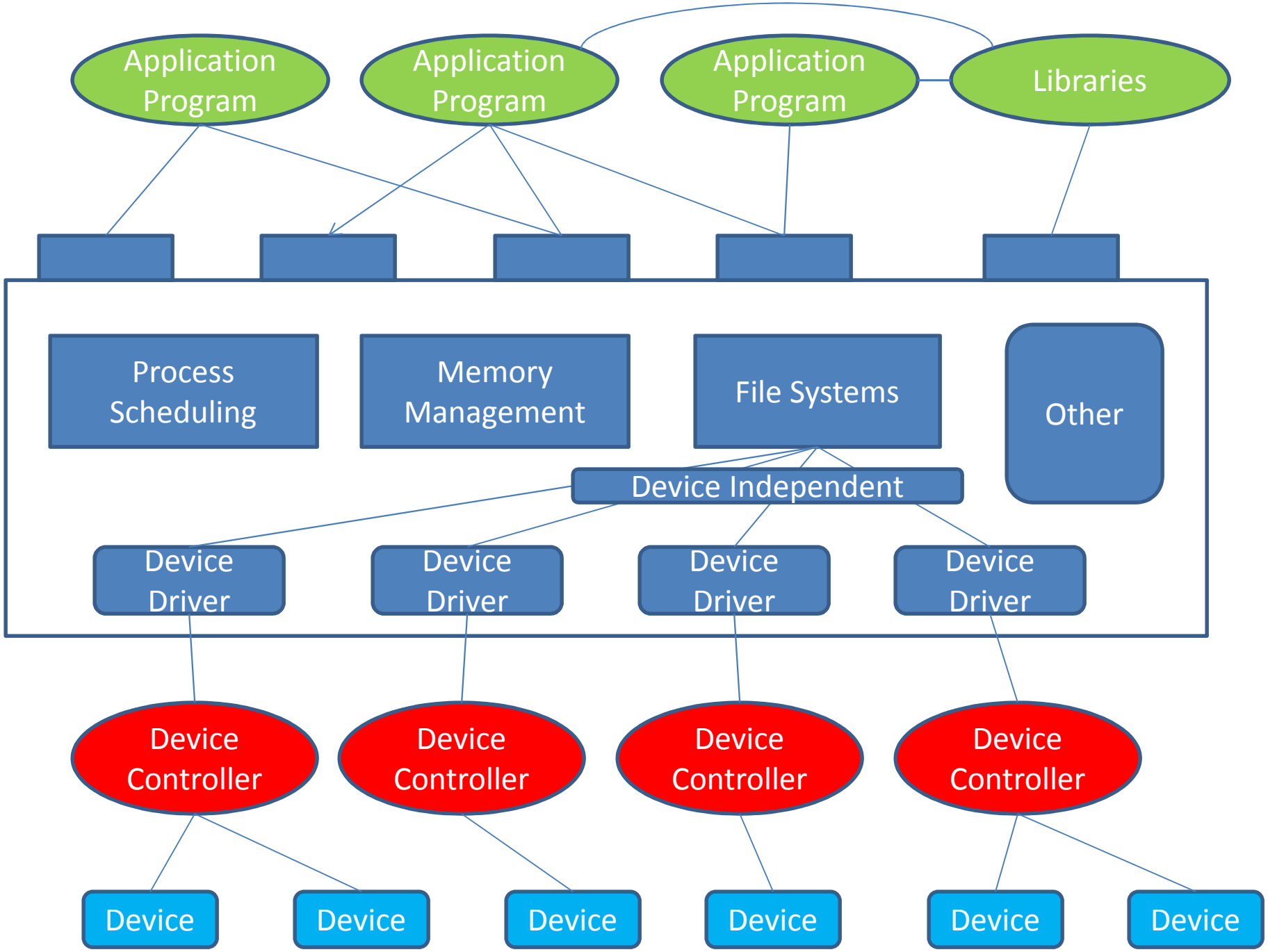
NYU
New York University

CSCI-GA.2250-001

Operating Systems

Advanced Topics

Hubertus Franke
frankeh@cs.nyu.edu

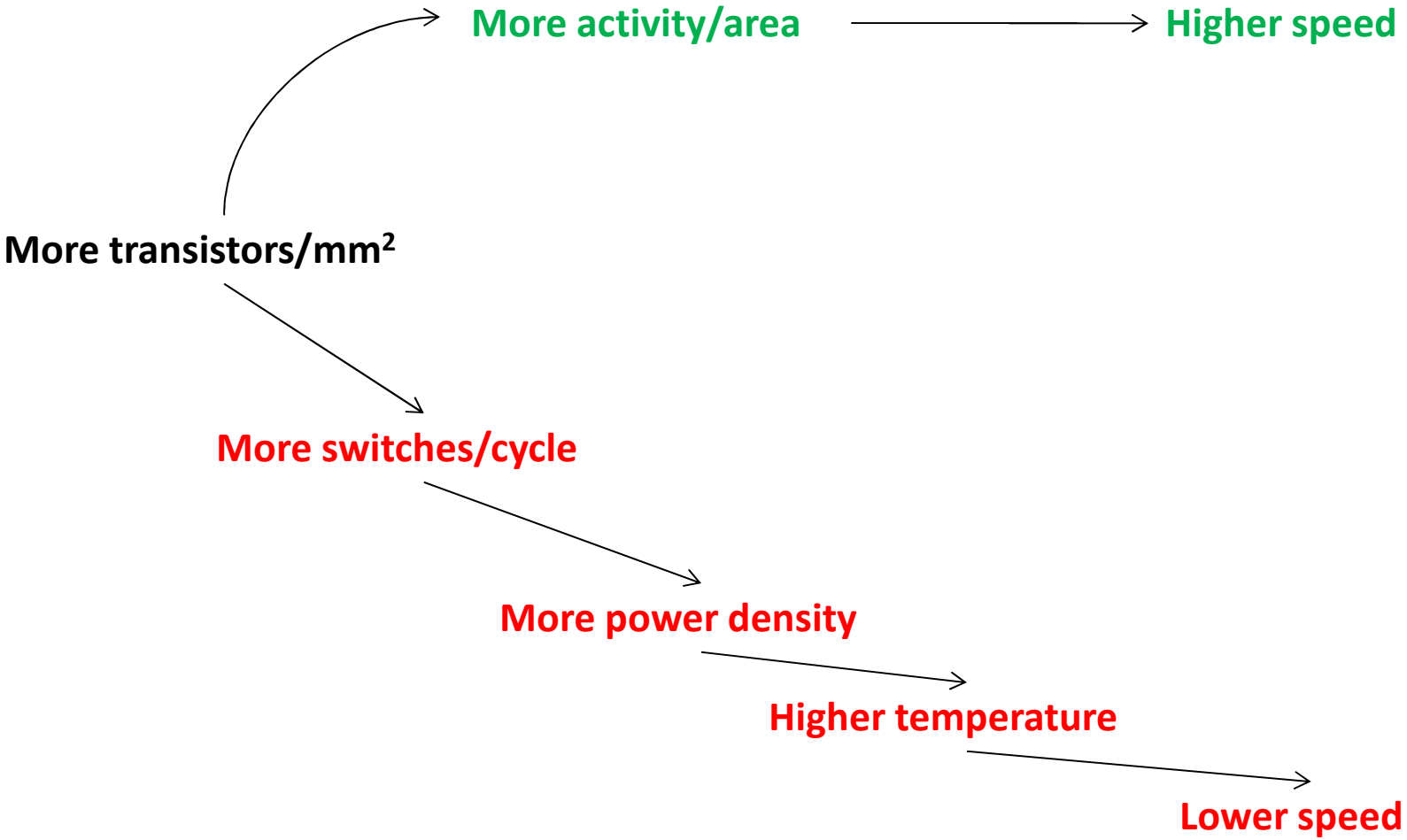

Application Program
Application Program
Application Program
Libraries
Process Scheduling
Memory Management
File Systems
Other
Device Independent
Device Driver
Device Driver
Device Driver
Device Driver
Device Controller
Device Controller
Device Controller
Device Controller
Device
Device
Device
Device
Device
Device

Power Management

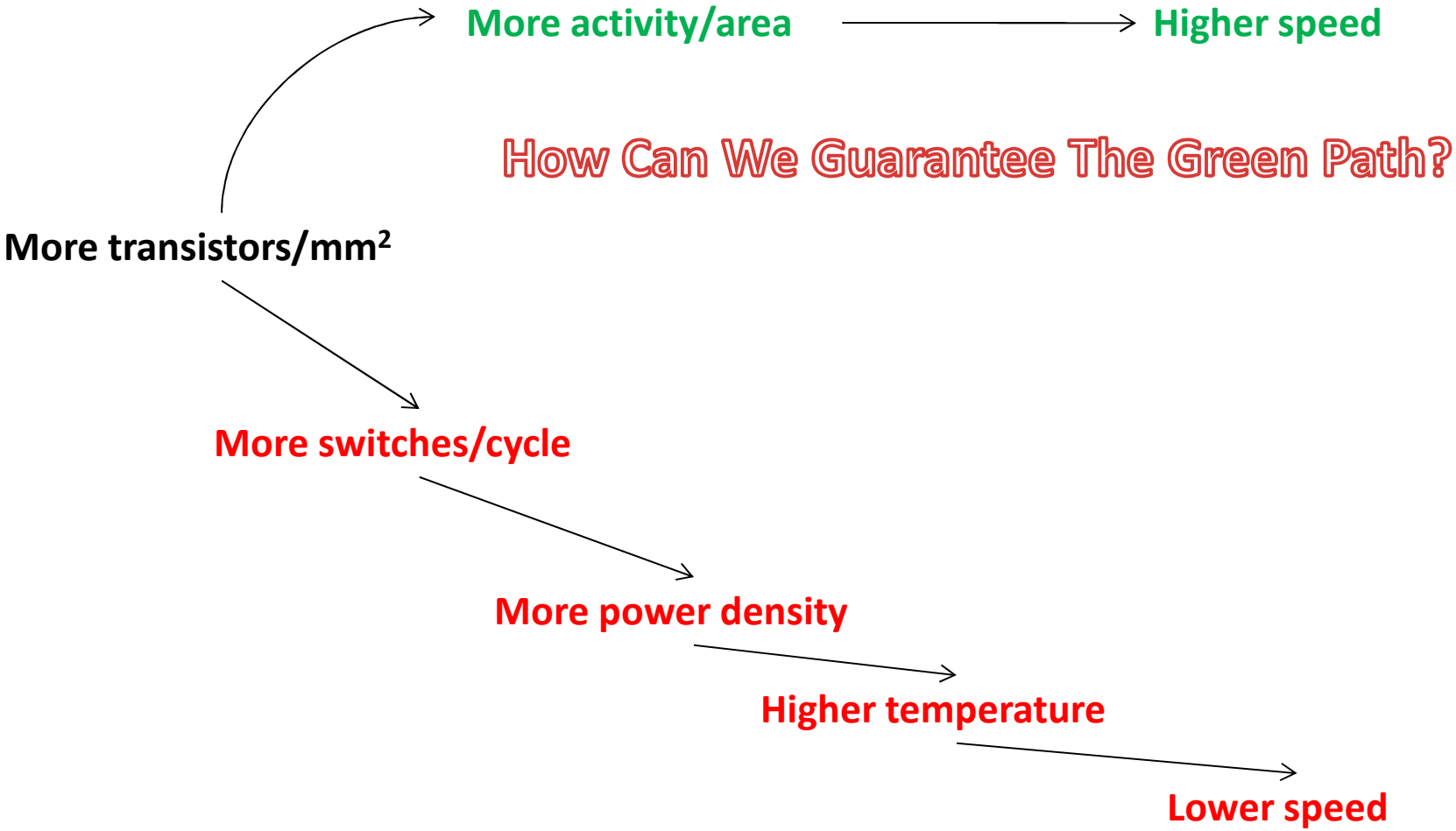
- Needed now more than ever
- Almost no progress in battery technology
- Typical desktop PC ~200 watt power supply with 85% efficiency
- Making computers use less energy so existing batteries last longer is high on everybody's agenda
- New devices require stringent power conservation
  - Smart Phones, Tablets
  - Smart Watches
  - Google-glasses
  - IoT (Internet of Things, e.g. remote powerline monitoring)

Moore's Law

Moore's Law


More activity/area
Higher speed
More transistors/mm²
More switches/cycle
More power density
Higher temperature
Lower speed

Moore's Law

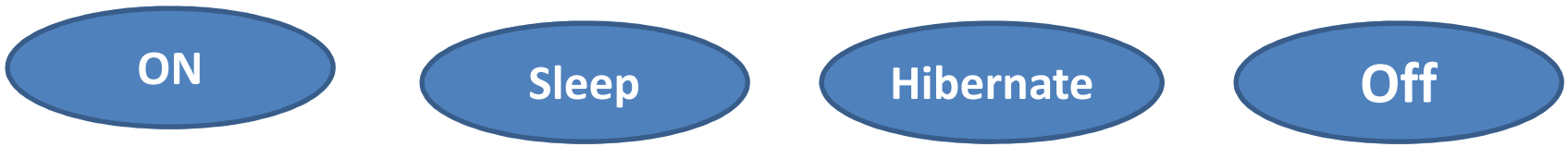

More activity/area
Higher speed
How Can We Guarantee The Green Path?
More transistors/mm²
More switches/cycle
More power density
Higher temperature
Lower speed

Power Management

Cooking Aware Computing

Power Management Philosophies

1. The OS turns off parts of the computer when they are not in use.
2. Application program to use less energy, with possible degradation in user experience in return of extended battery life.

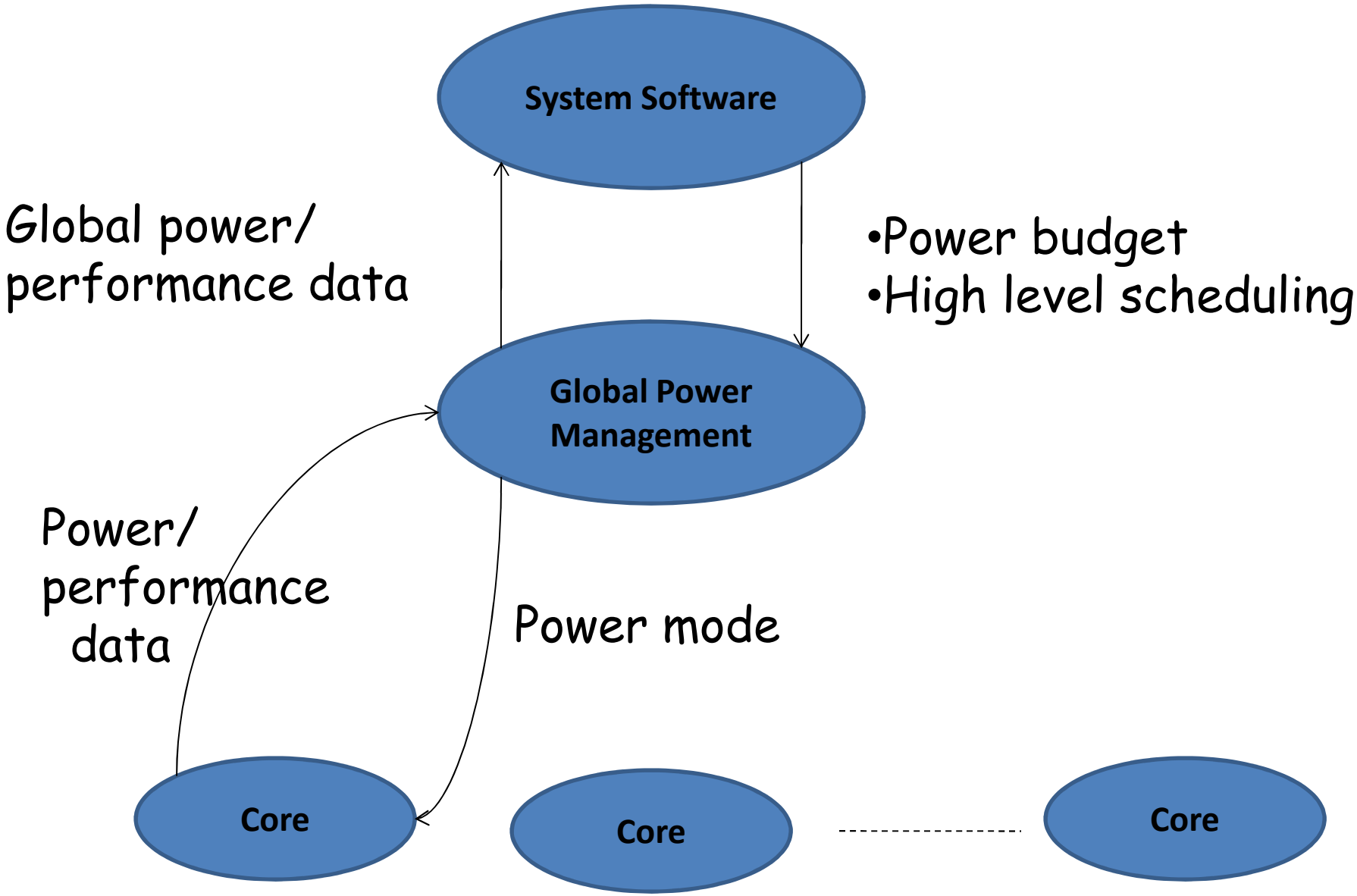


Typical States of devices

OS Role in Power Management

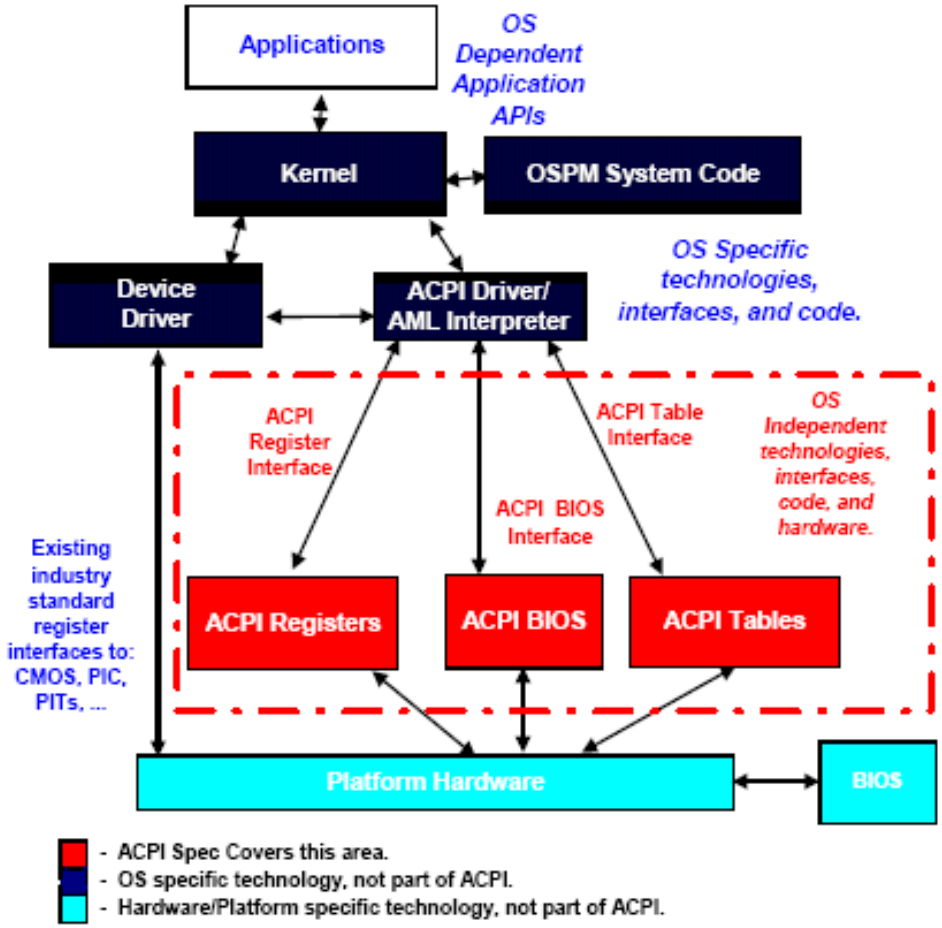
- The OS must be designed to make good decision about what to shut down (or make state change) and when.
  - Delays in shutting down and waking up
  - Energy consumption in shutting down and waking up
- Biggest energy consumers in PC: display, hard disk, CPU

Example of Managing Cores power consumption

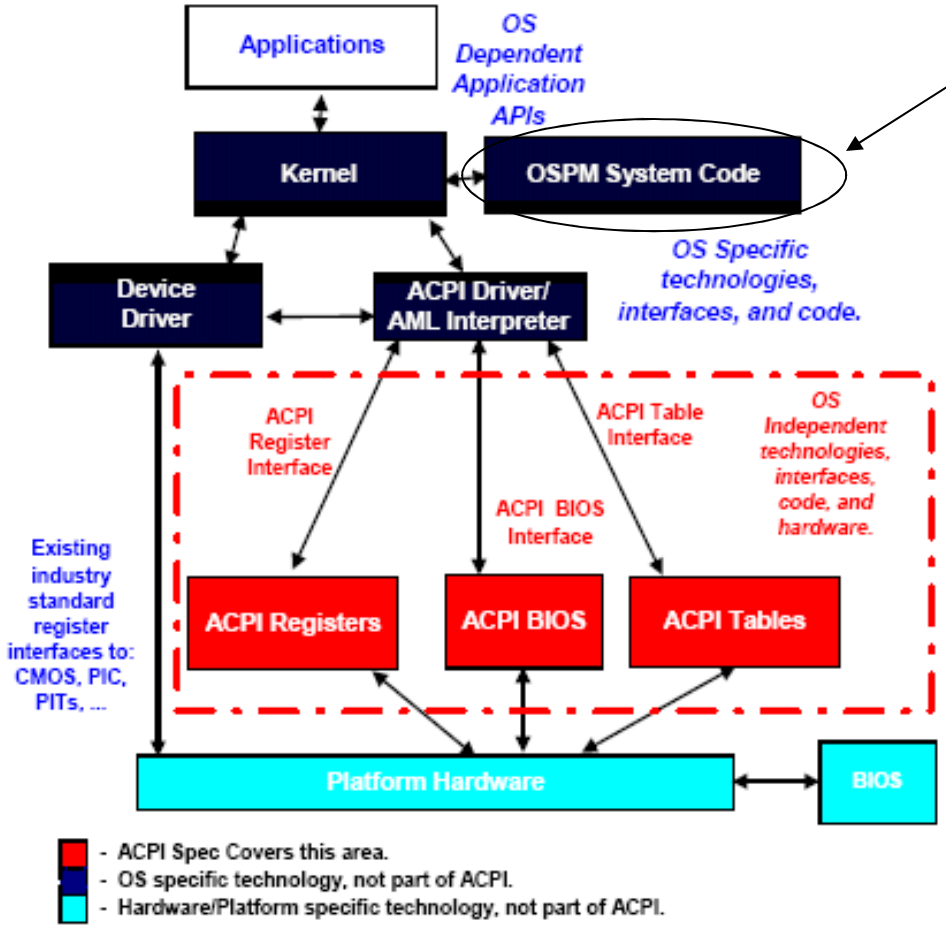


Advanced Configuration and Power Interface (ACPI)

- An open industry specification co-developed by Hewlett-Packard, Intel, Microsoft, Toshiba, among others.
- Devices must support power saving modes
- ACPI must be supported by the computer motherboard, BIOS, and the operating system
- ACPI is a power management platform at the hardware level
- Establishes industry-standard interfaces for OS-directed configuration and power management on laptops, desktops, and servers.



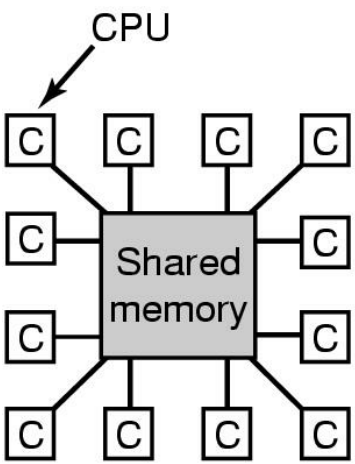
http://www.acpi.info



OS directed
Power Management

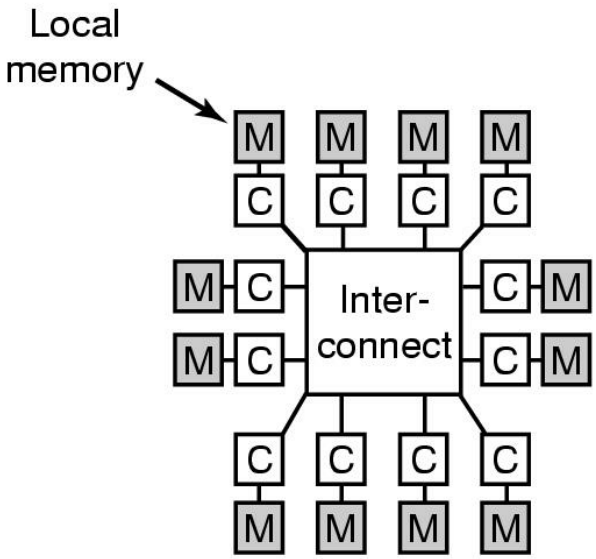
OS For Multiprocessors

OS For Multiprocessors



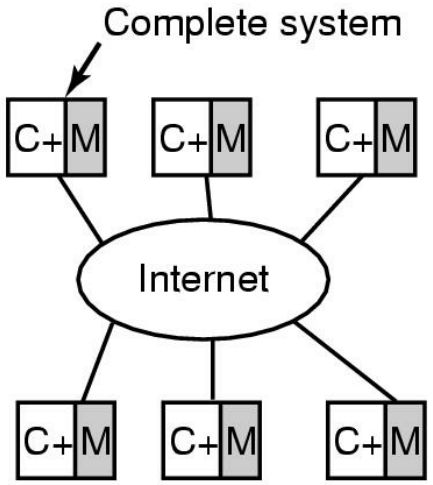
(a)

Shared Memory



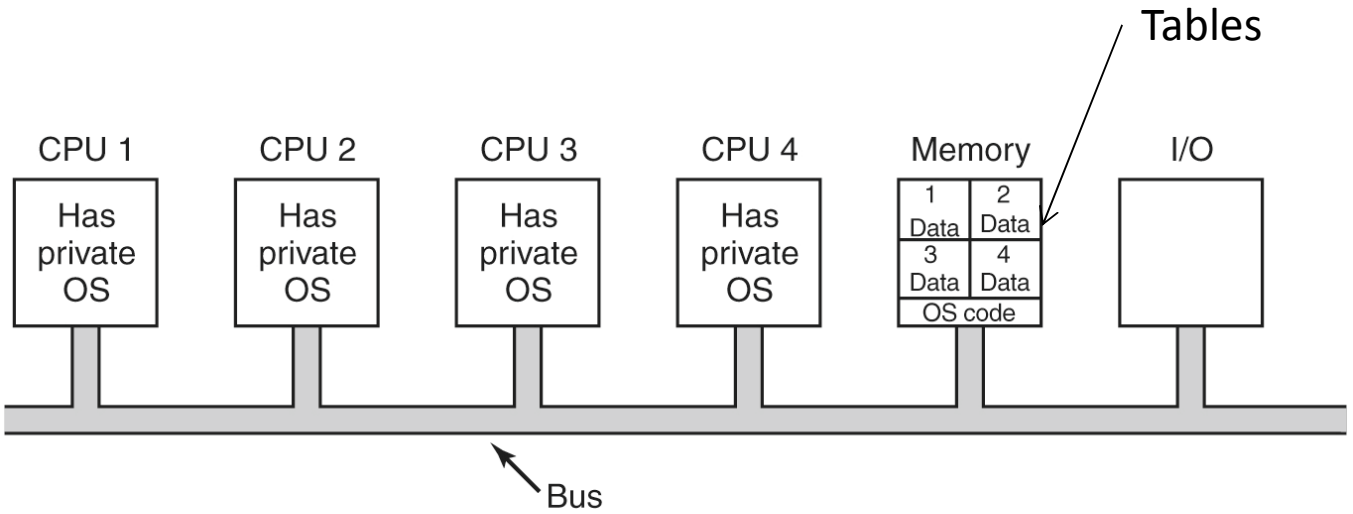
(b)

Message Passing

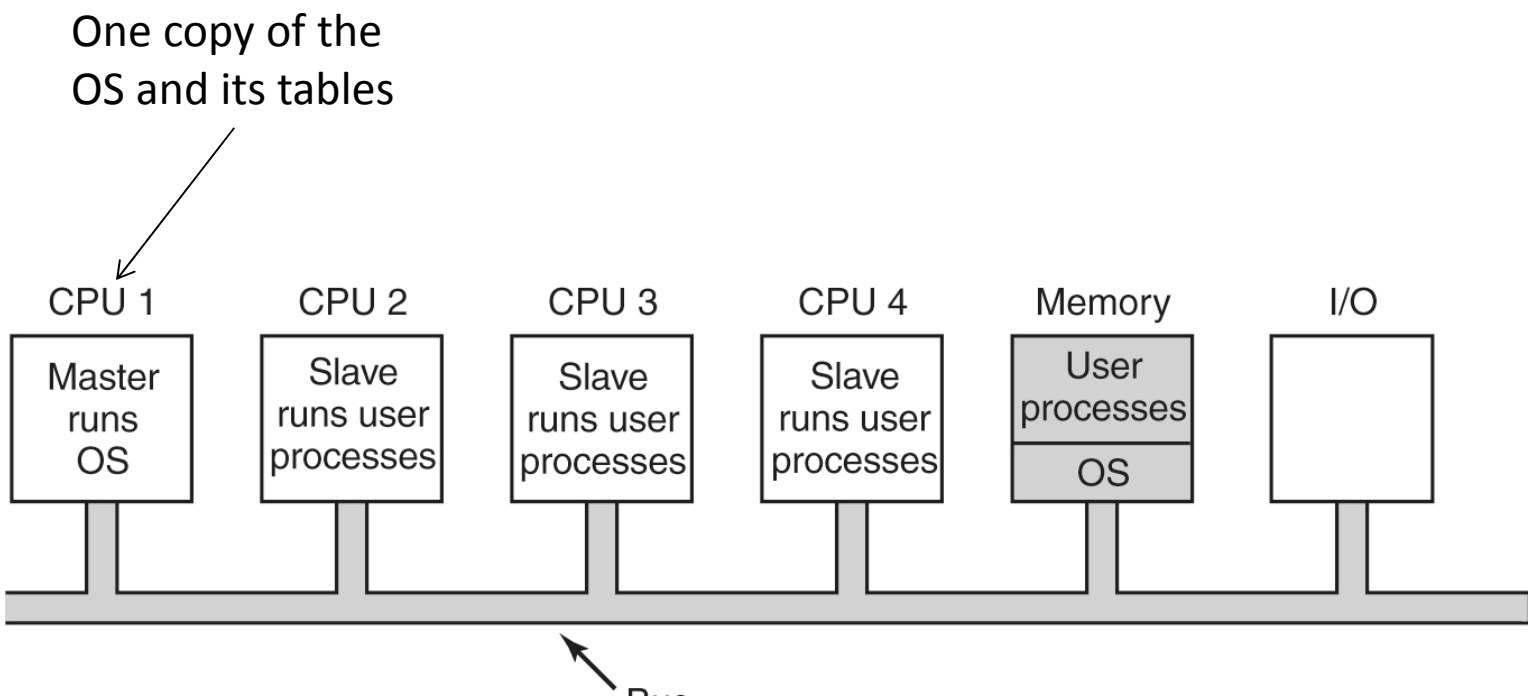


(c)

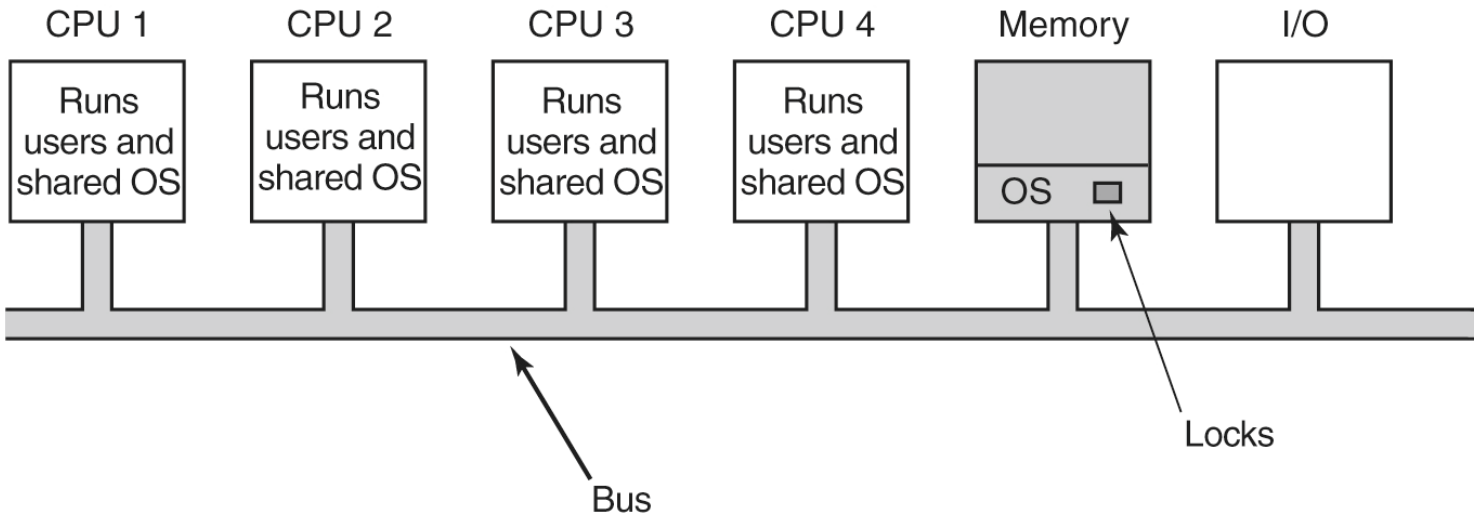
Distributed System



Each CPU has its own OS



Master-Slave Model

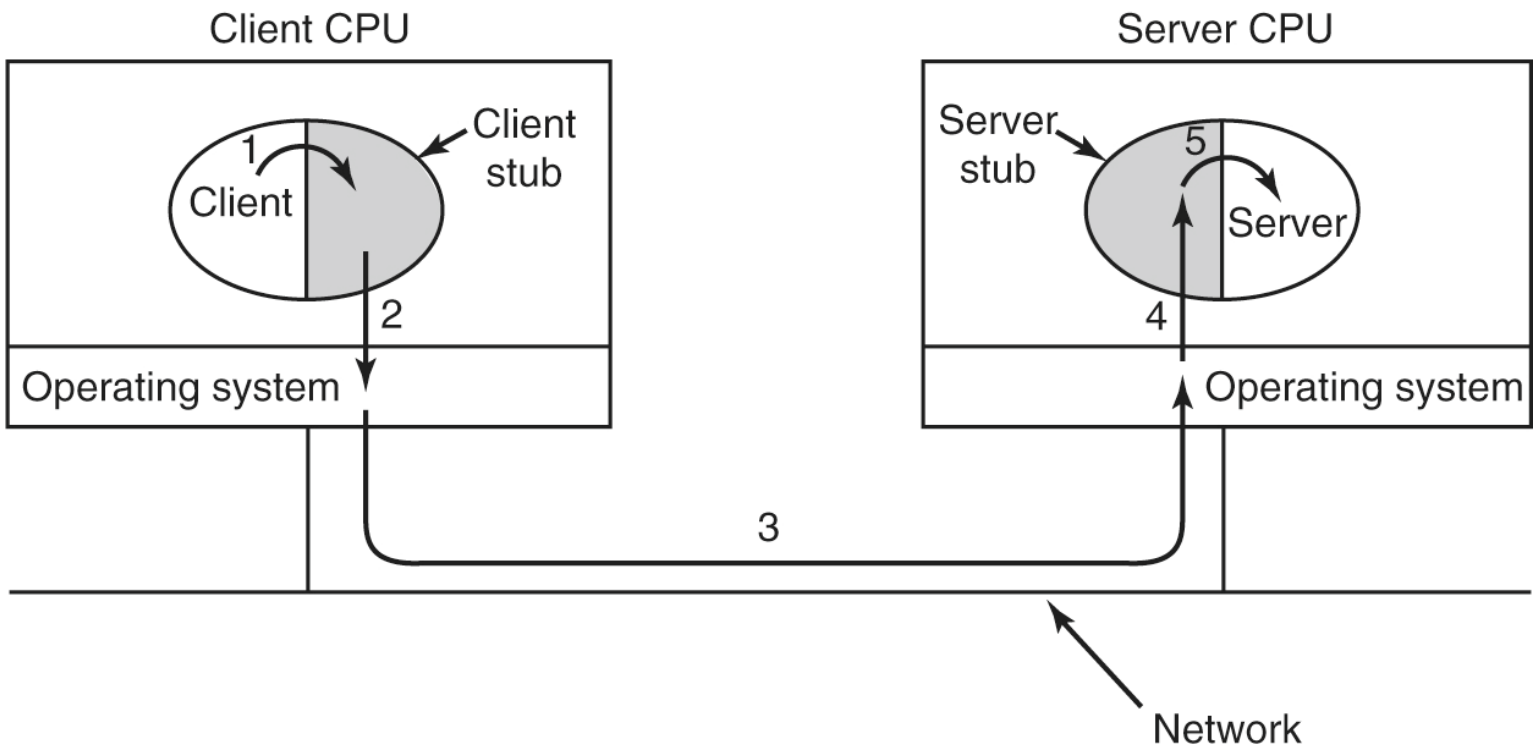


Symmetric Multiprocessors (SMP)
Any CPU can run the OS

From Multiprocessors to Multicomputers

- Tightly coupled CPUs that do NOT share memory
- Sometimes called cluster computers
- Easier to build than a multiprocessor system
  - Stripped down PCs and high-performance network interface

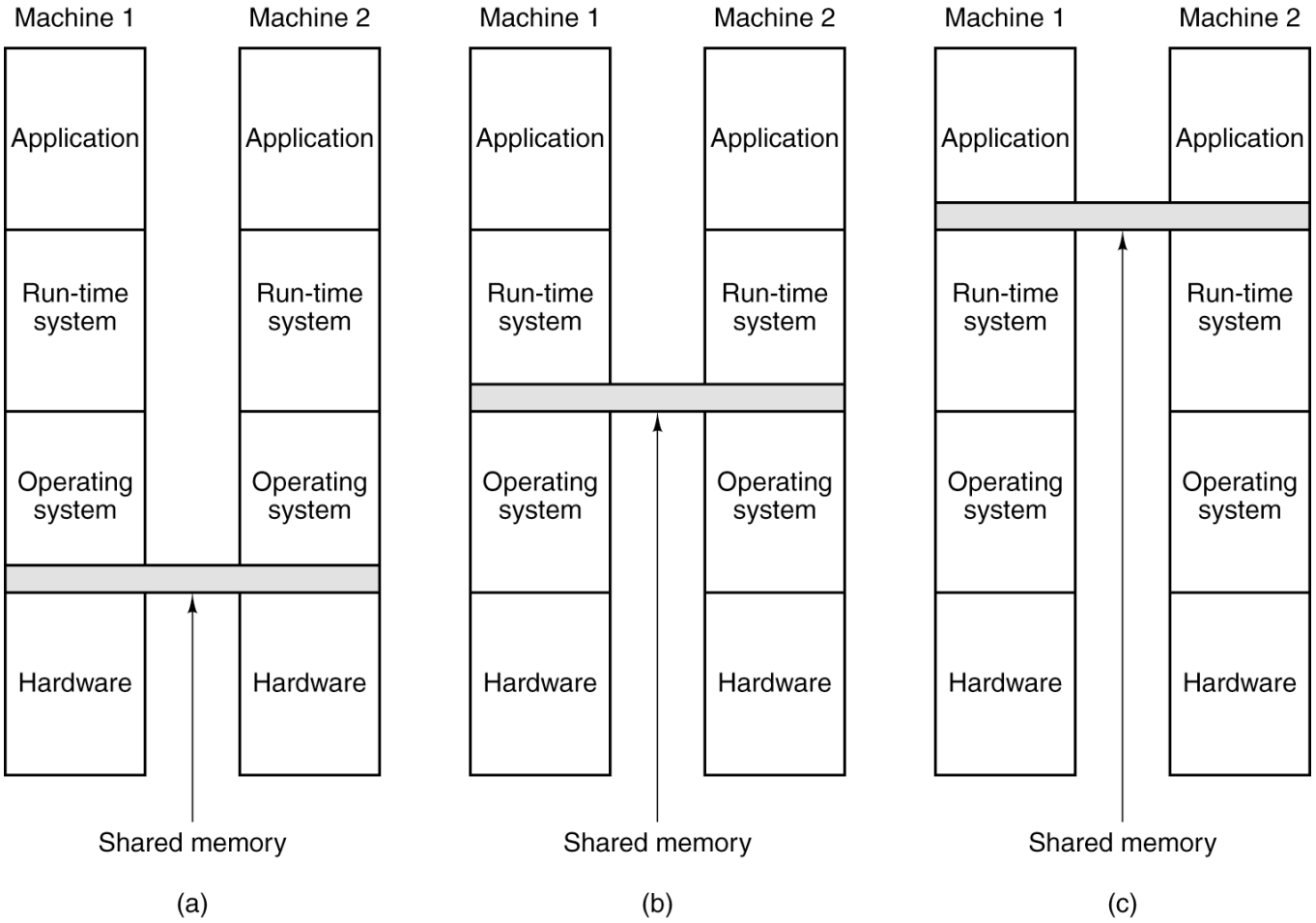
Multicomputers: Remote Procedure Call



Multicomputers: Distributed Shared Memory

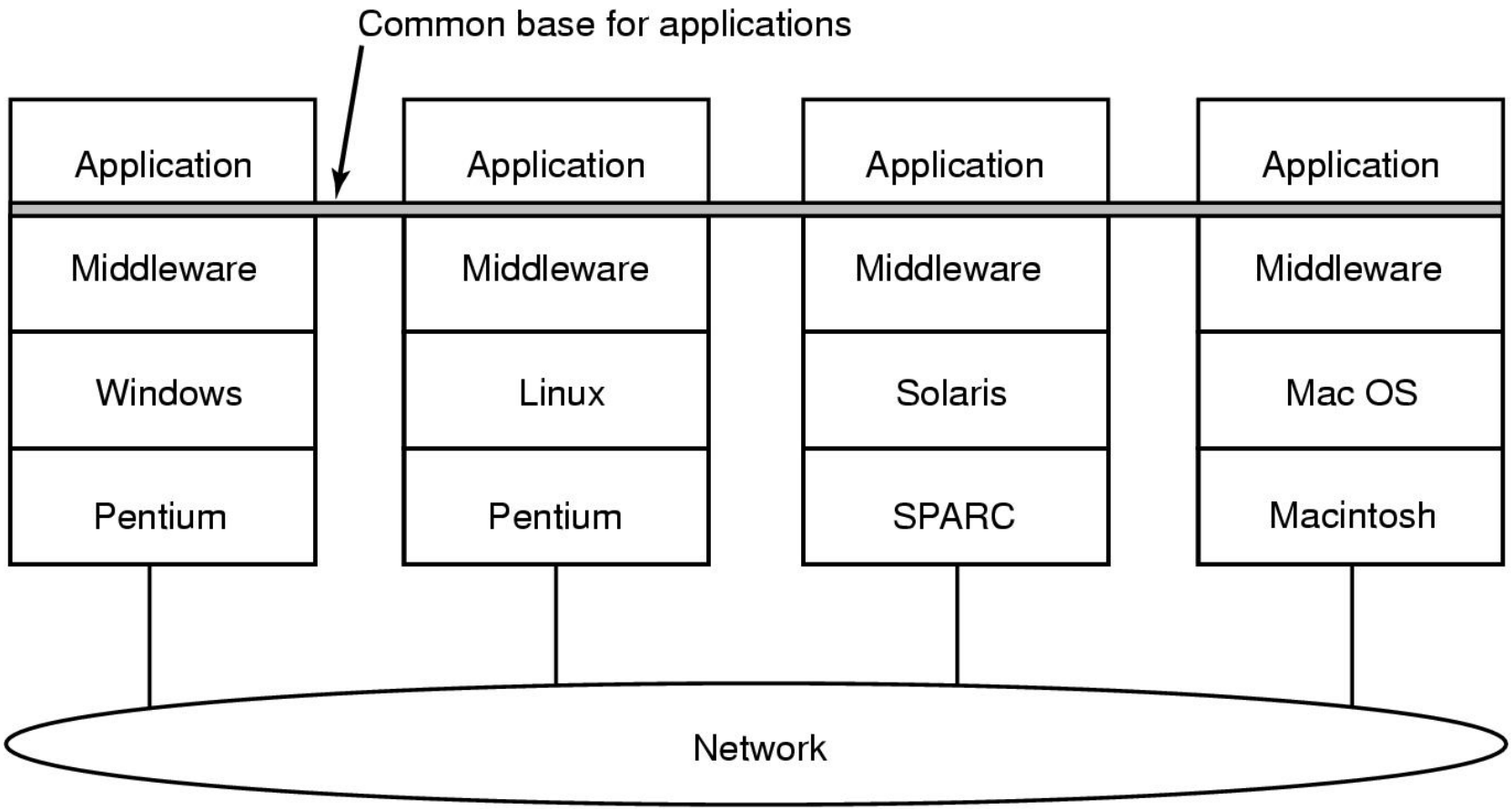
- Each page is located in one of the memories
- Each machine has its own page table
- If a CPU accesses a page it does not have:
  1. trap to OS occurs
  2. OS locates the page
  3. OS asks the CPU to unmap the page and send it to the requesting CPU
  4. The page is mapped at the receiving CPU

Multicomputers: Distributed Shared Memory



Operating System	*Description*	*URL for more information*
Alpha Kernel	Developed at Carnegie Mellon University, started in 1985, built from scratch on custom made distributed system. Intended to be a real-time distributed OS, development seems to have finished around 1995.	http://www.realtime-os.com/alpha.html
Amoeba	Developed at Vrije Universitei, started about 1990. Design around a microkernel, ran on both Motorola and Intel based hardware.	http://www.cs.vu.nl/pub/amoeba/
Angel	Developed at City University of London, started in 1991. Initially not UNIX compatible, based on microkernel, emphasis on single address space.	http://www.soi.city.ac.uk/research/sarc/angel
ChorusOS	Developed at Sun Microsystems. Microkernel based, real-time, embedded, scalable and distributed OS. Used in telecommunications equipment, e.g. PBXs, public switches.	http://www.sun.com/chorusos/
GLUnix	Developed at University of California, Berkeley, began mid-1990s. Implemented at the user level on top of Solaris 2.3- 2.6	http://now.cs.berkeley.edu/Glunix/glunix.html
Guide	Developed by Bull and Universities of Grenoble, object-oriented distributed operating system. Began late-1980s, redevelopment in 1991, moved to Mach based kernel, emphasis on using C++ persistent objects to keep track of jobs.	http://www-bi.imag.fr/GUIDE/presguide.html
Hurricane	Developed at University of Toronto, began early 1990s. Hierarchically clustered operating system implemented on the Hector multiprocessor.	http://www.eecg.toronto.edu/EECG/RESEARCH/ParallelSys/hurricane.html
MOSIX	Developed at The Hebrew University of Jerusalem, began early 1990s. Patches Linux/UNIX kernel to add clustering capabilities.	http://www.mosix.org/
Plan 9	Developed at Bell Labs Computing Sciences Research Centre, distributed operating system and associated utilities. Built from scratch.	http://plan9.bell-labs.com/plan9/

Distributed Systems

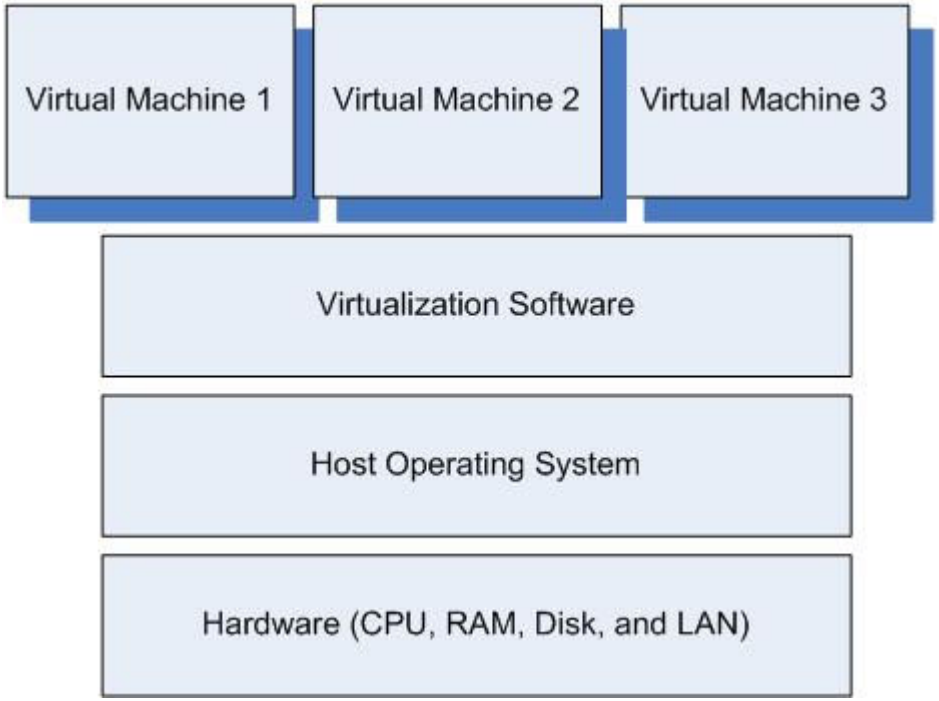


Item	Multiprocessor	Multicomputer	Distributed System
Node configuration	CPU	CPU, RAM, net interface	Complete computer
Node peripherals	All shared	Shared exc. maybe disk	Full set per node
Location	Same rack	Same room	Possibly worldwide
Internode communication	Shared RAM	Dedicated interconnect	Traditional network
Operating systems	One, shared	Multiple, same	Possibly all different
File systems	One, shared	One, shared	Each node has own
Administration	One organization	One organization	Many organizations

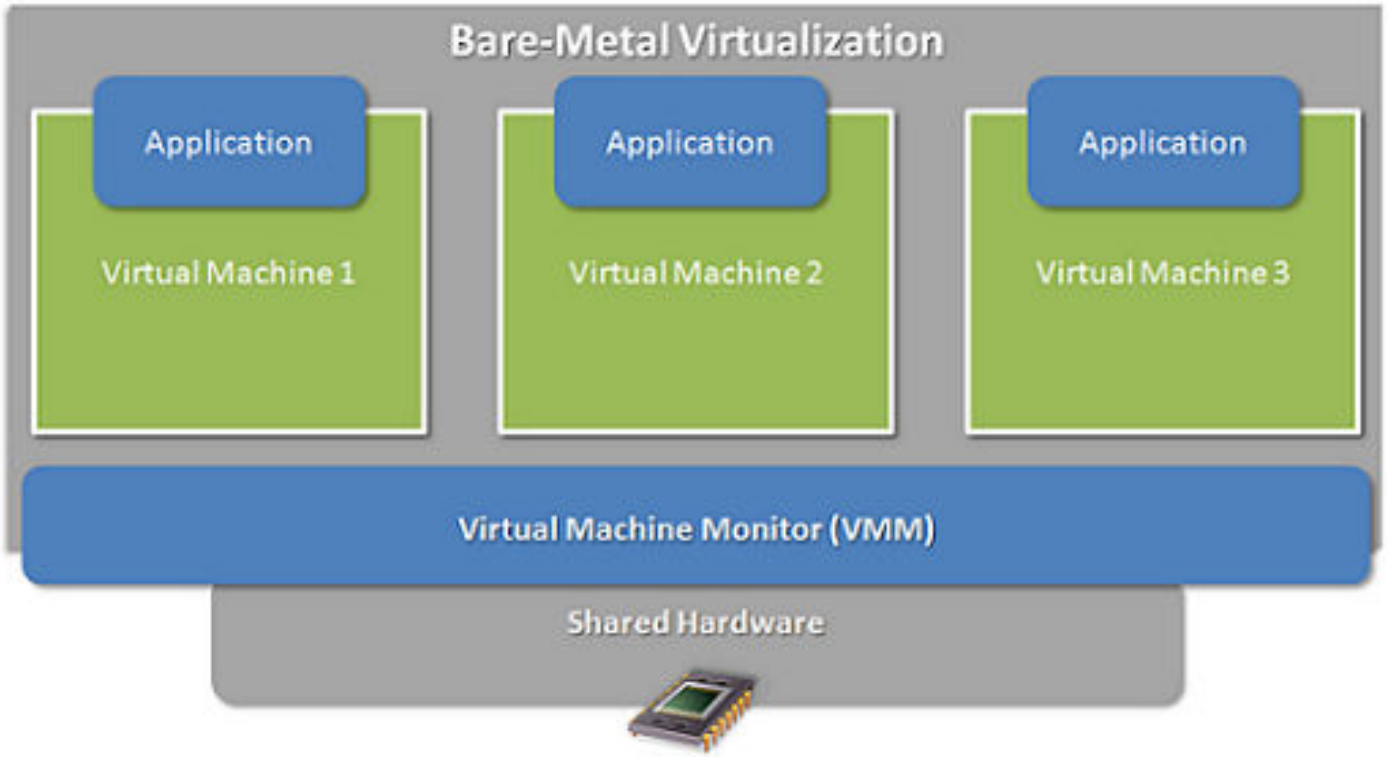
OS and Virtualization

- Allows a single machine to run several *guest OSes* simultaneously
- Each OS is running on a virtual machine (VM)
- If a VM crashes the others are not affected

OS and Virtualization



OS and Virtualization

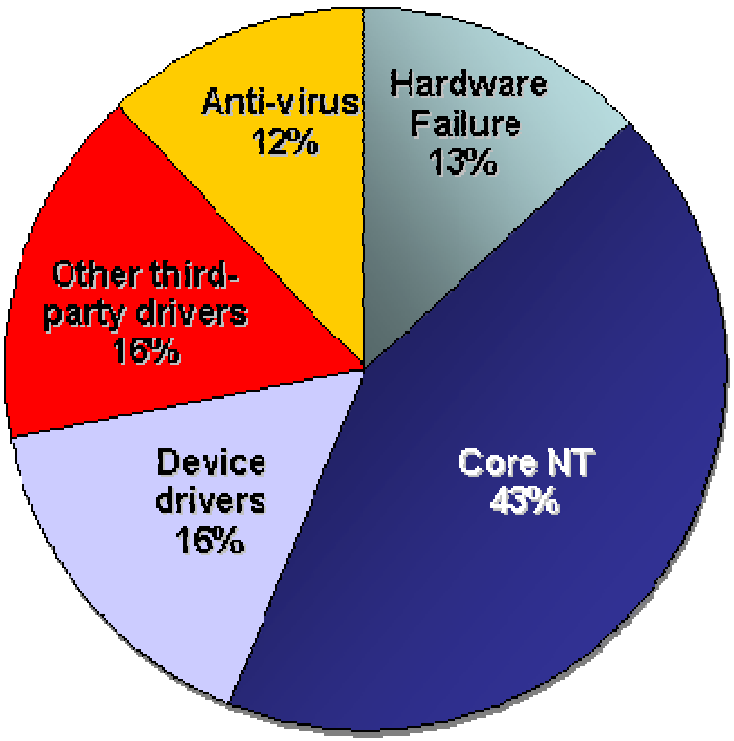


OS and Security

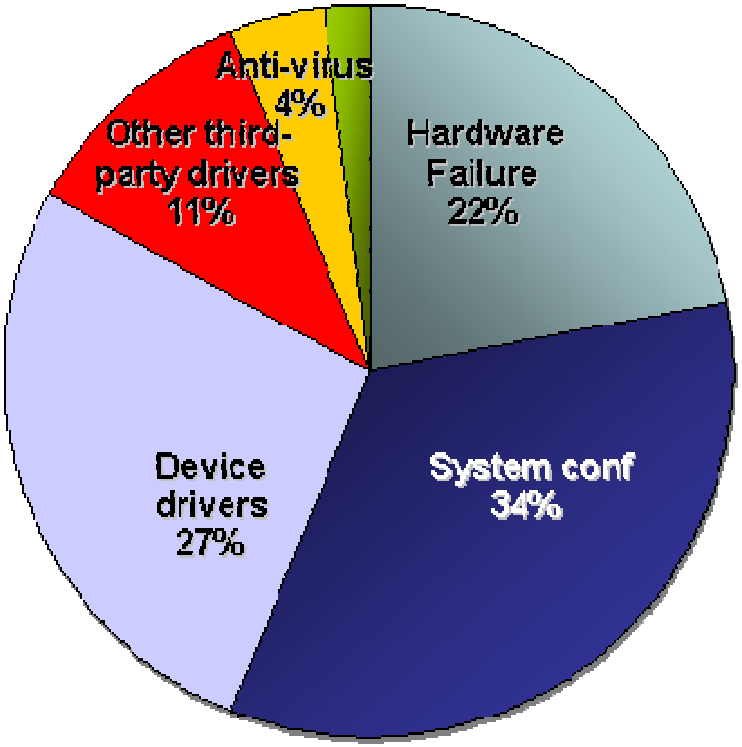
Basic Features of a Multiprogramming OS

- Authentication of users.
- Protection of memory
- File and I/O device access control
- Guarantee of fair service
- Interprocess communication and synchronization.

Fault Tolerance



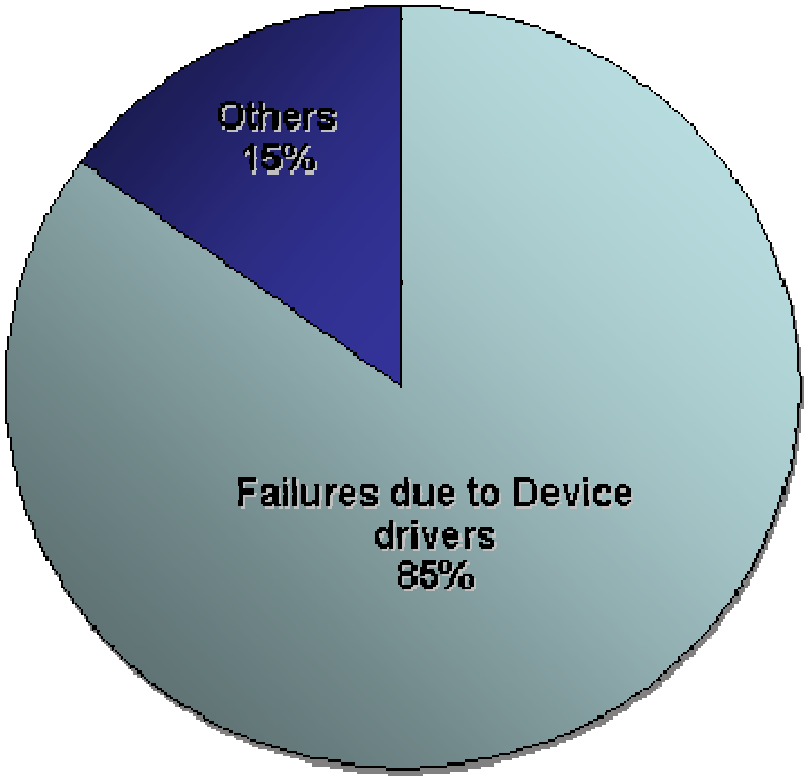
Microsoft NT



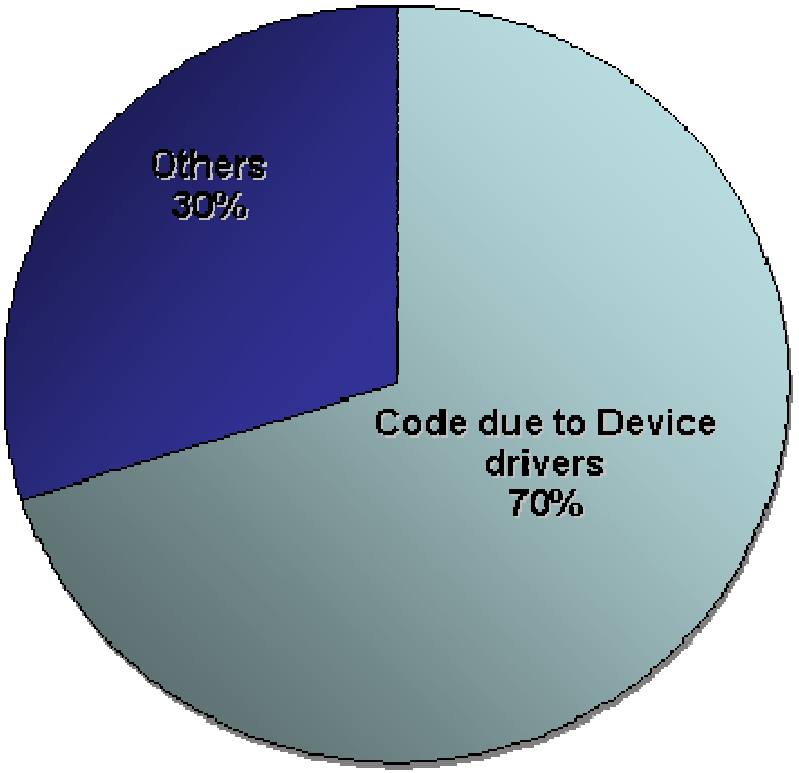
Microsoft 2000

Data obtained from *courses.cs.vt.edu/~cs5204/fall05-gback/presentations/hari_nooks.**ppt***

Fault Tolerance



Microsoft XP



Linux

Data obtained from *courses.cs.vt.edu/~cs5204/fall05-gback/presentations/hari_nooks.**ppt***

Fault Tolerance

- Verify all parameters on calls between the kernel and device drivers
- Prevent device drivers from writing to kernel memory
- Prevent device drivers from executing privileged instructions and/or emulate privileged instructions
- Inject code into device drivers to ensure that addresses and instructions are safe
- Design for fault resistance not fault tolerance

Real-Time OS (RTOS)

- Correctness of output depends on timing as well as result
- Typical characteristics
  - Threads priority can be set by users
  - Threads scheduled by priority
  - Interrupts must be bounded
  - RTOS scheduler needs to be deterministic ~ O(1) or O(n).

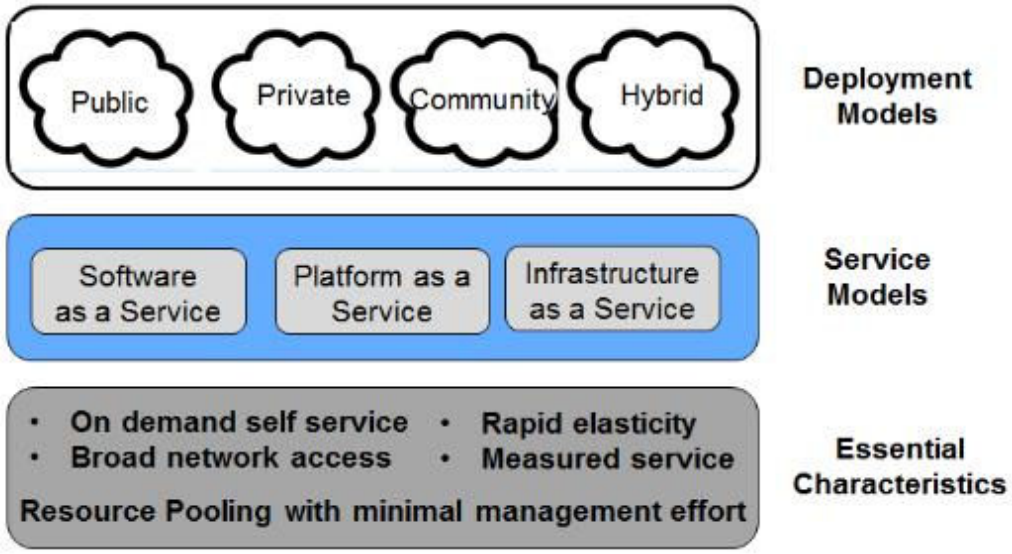
Conclusions

- First thing in designing an OS is to know your *customers* to be able to decide the relative importance of: cost, reliability, speed, security, compatibility and … time-to-market
  (aka Time to Value T2V)

- It is all about the art of trade-offs.

Cloud Computing

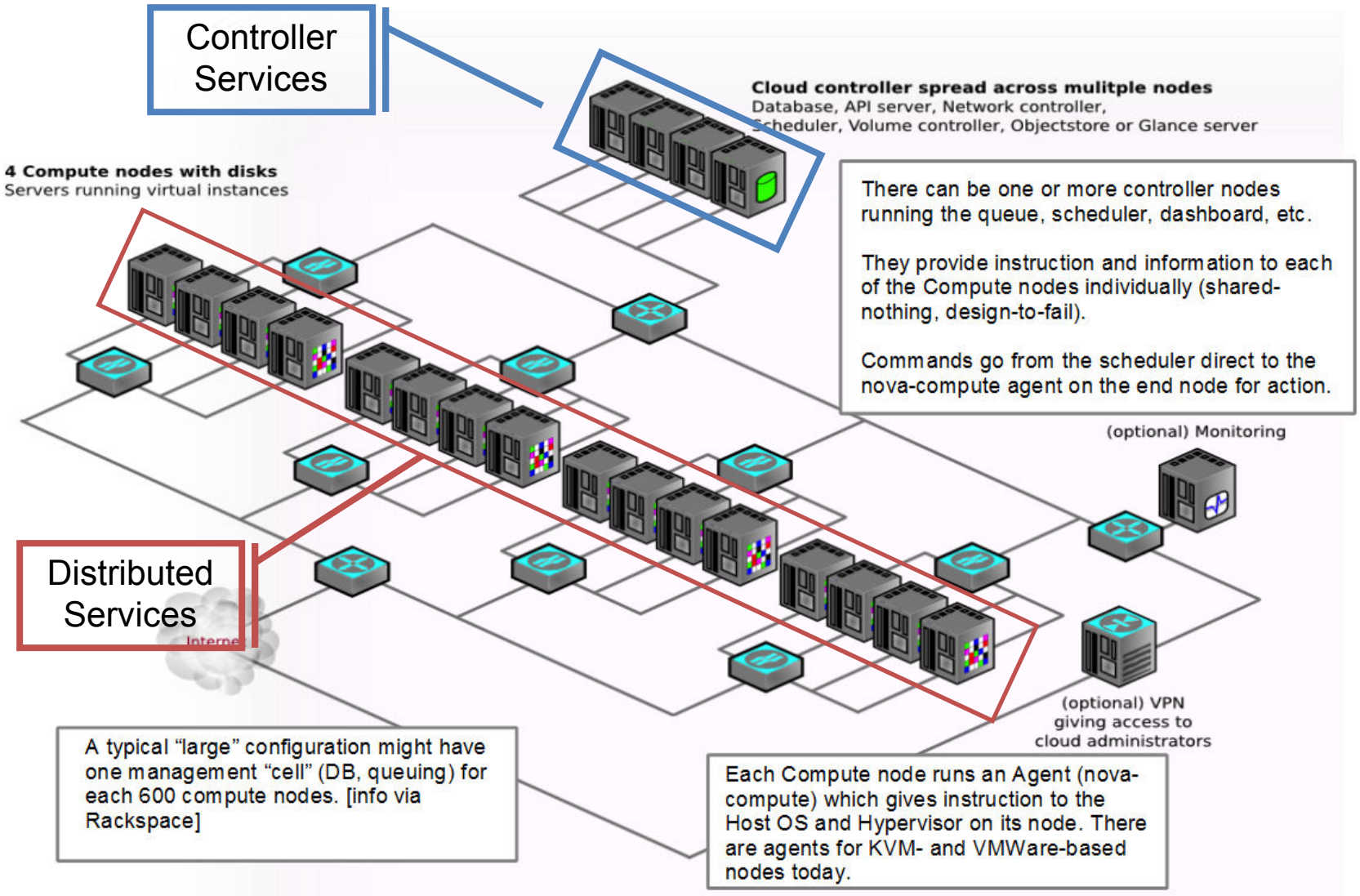
- New IT delivery model
- Access your services remotely (cloud)
- Pay as you go model
  - Amazon Webservices



IaaS (Infrastructure as a Service)

- Request "machines" with certain characteristics
- IaaS layer provisions necessary resources
  - Compute
  - Network
  - Storage
- Provides connectivity to the machine
- Heavily relies on virtualization technology

Cloud implementation modules can be categorized into two groups – controller services and distributed services



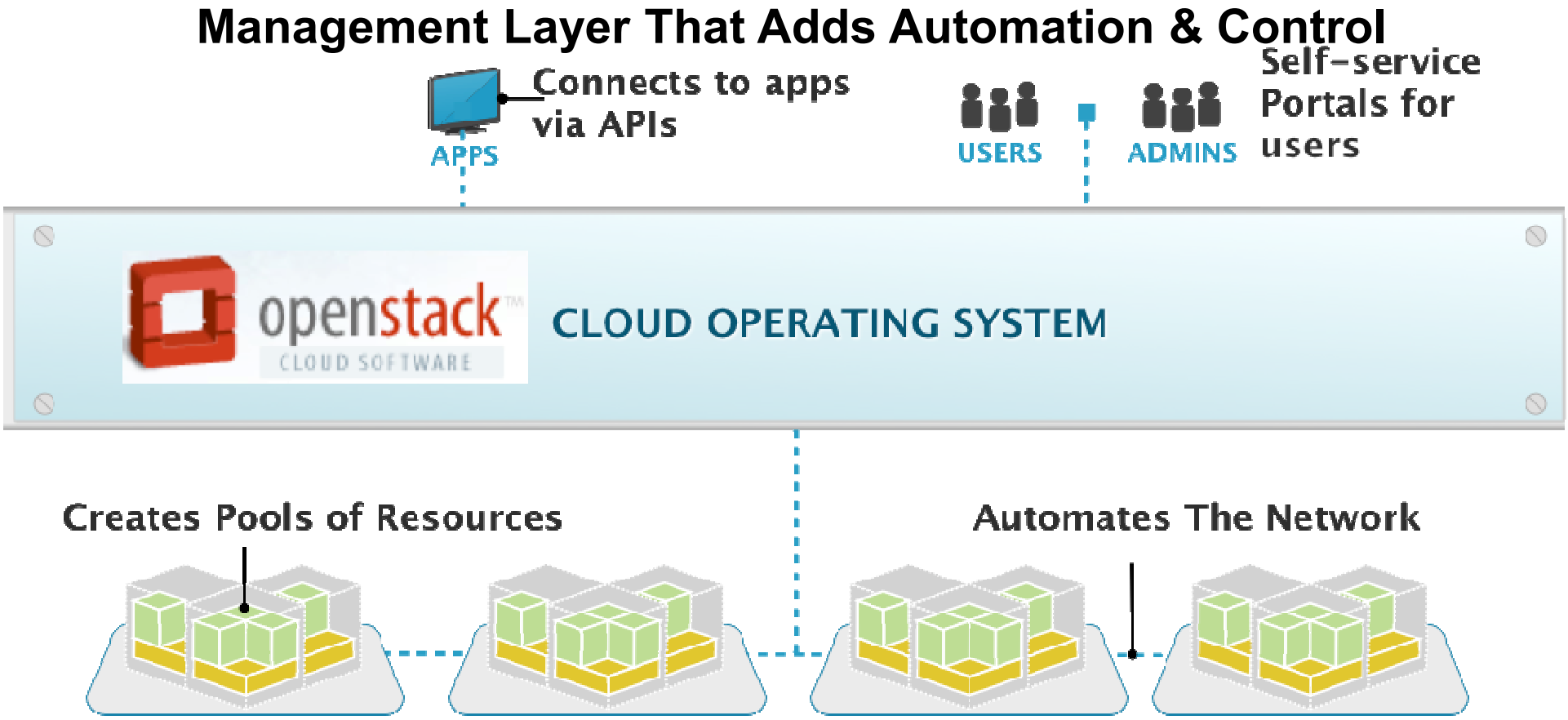
OpenStack Cloud Environment

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter

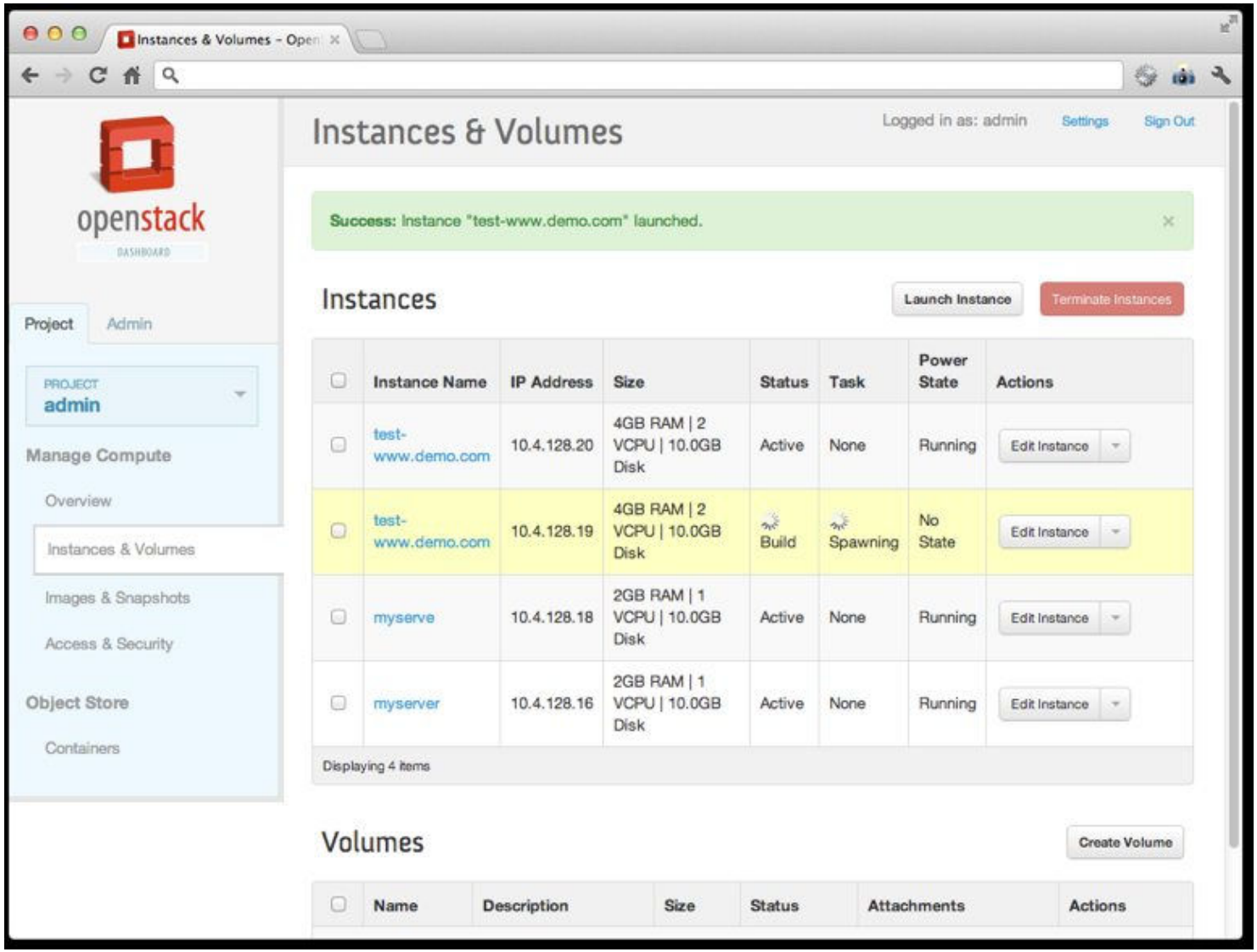
" Our goal is to produce the ubiquitous Open Source cloud computing platform that will meet the needs of public and private cloud providers regardless of size, by being simple to implement and massively scalable. "

- Open source (Apache 2.0 licensed)
- "Linux of the data center" eliminate vendor lock-in, maintain workload portability
- Build a great engine, packagers make a great car (think Linux Kernel to RHEL/SUSE)
- Design Tenets
  - scalability and elasticity are our main goals
  - share nothing, distribute everything (must be asynchronous and horizontally scalable)
  - any feature that limits our main goals must be optional
  - accept eventual consistency and use it where appropriate

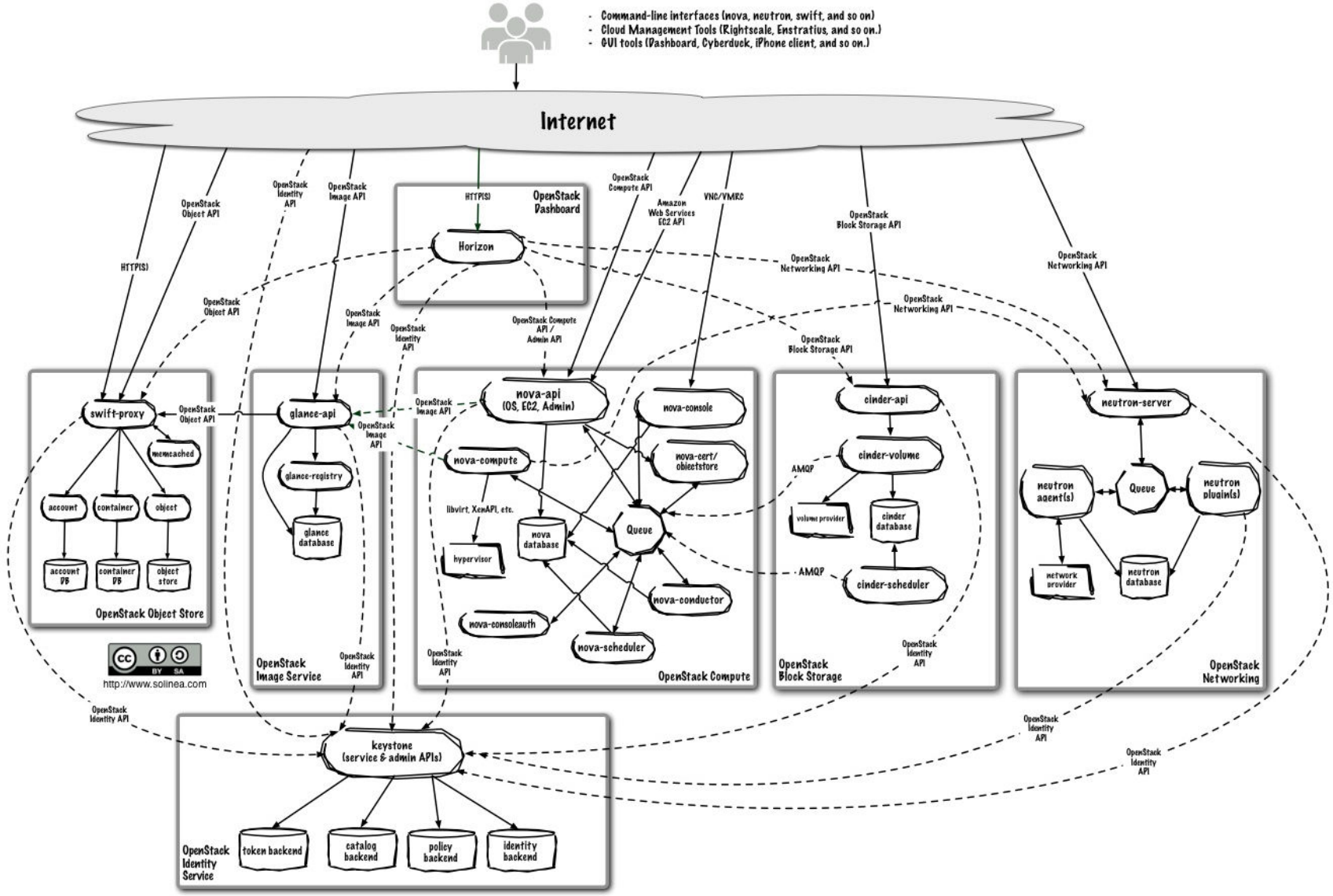
OpenStack
the Cloud Operating System



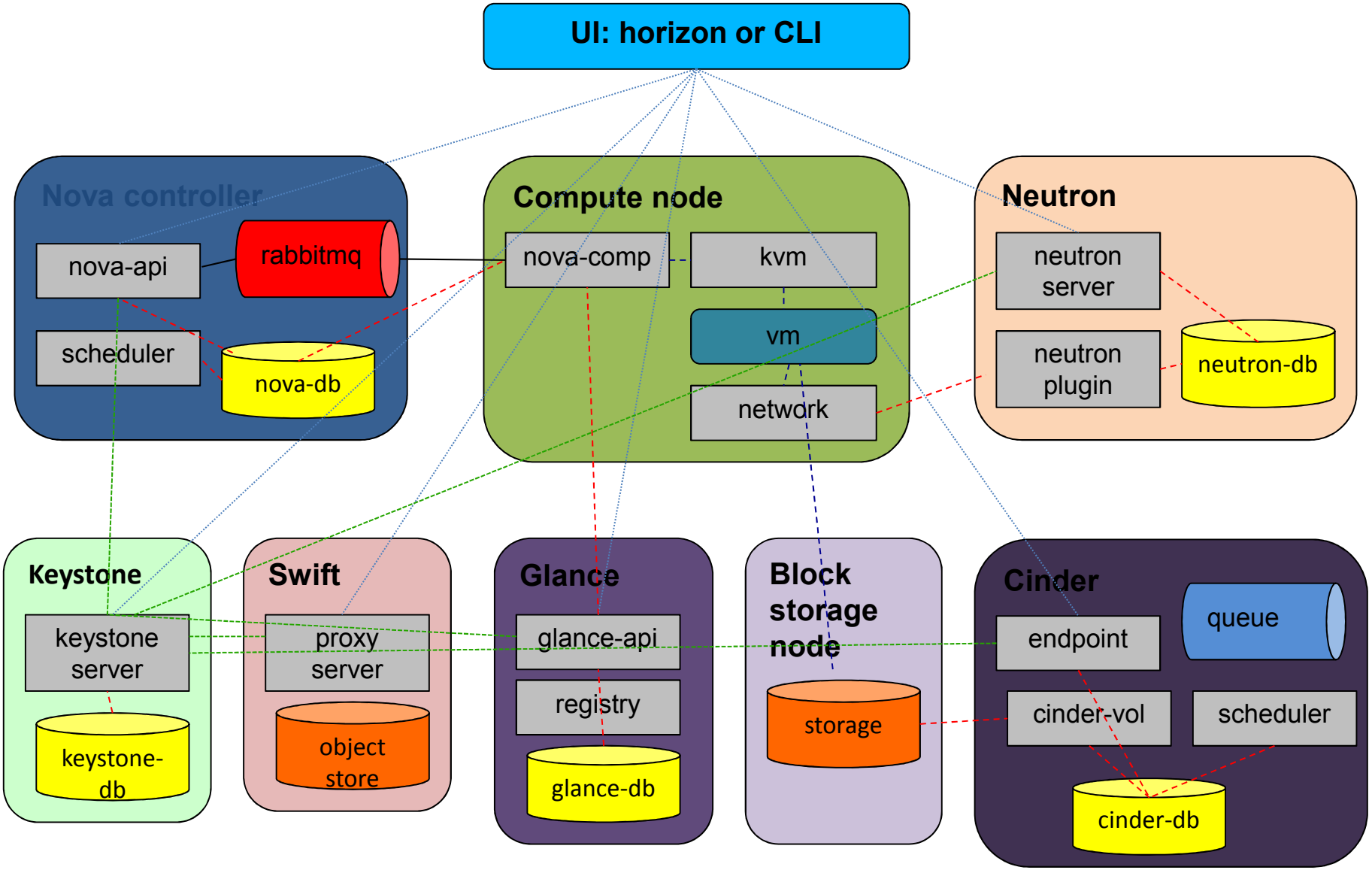
Service Consumption through Portal



General OpenStack Architecture

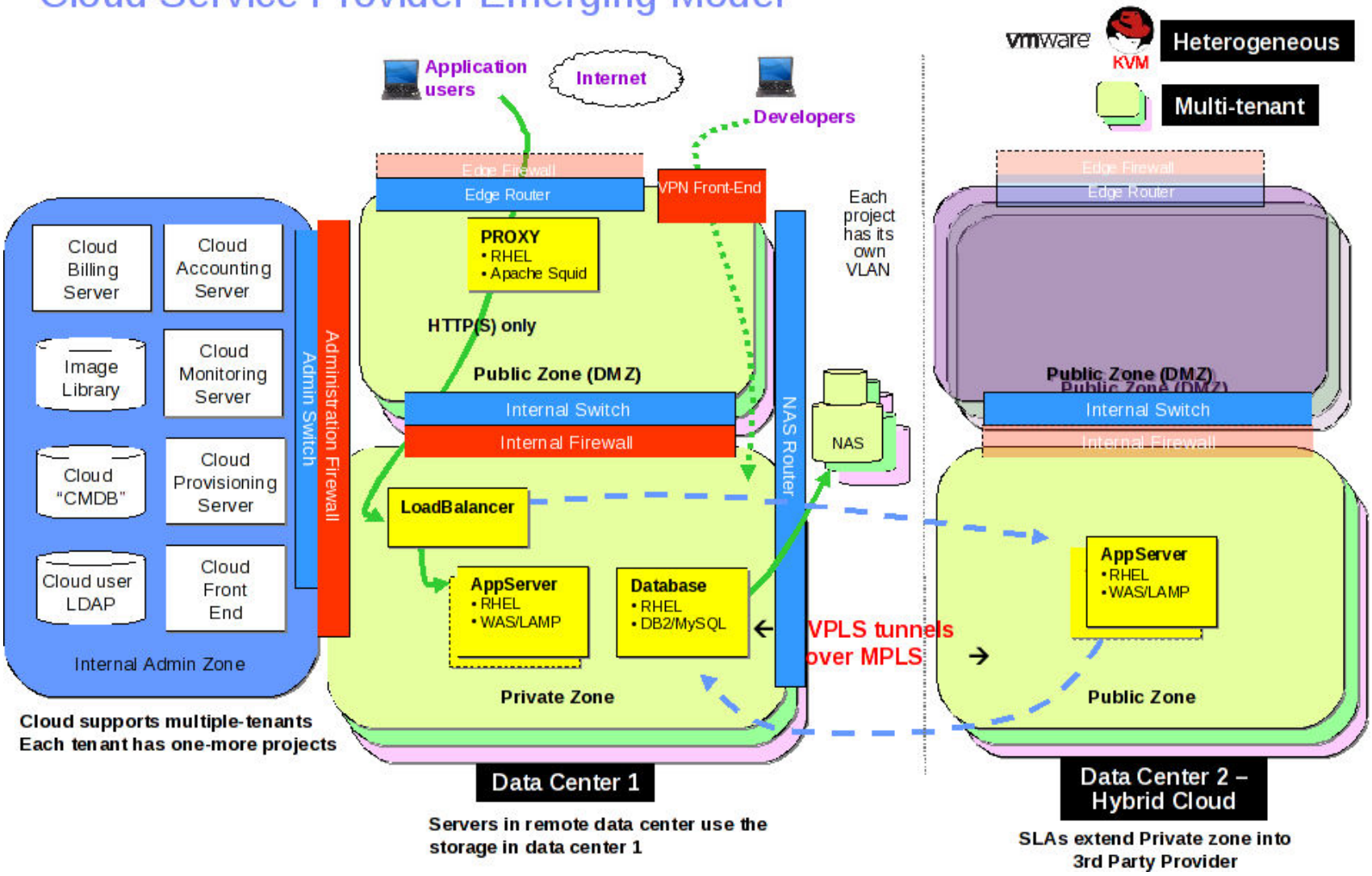


OpenStack Core Components



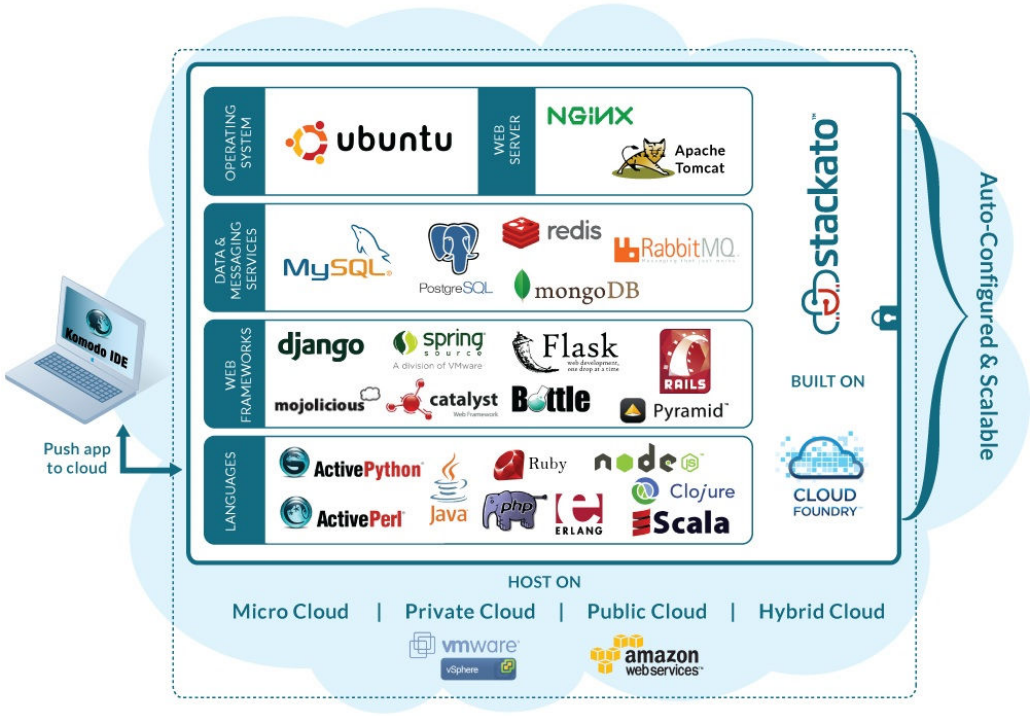
Hybrid Cloud Model

Cloud Service Provider Emerging Model



PaaS (Platform as a Service)

- Obtain Services (DBs, Queues, ..)
- Customize



SaaS (Software as a Service)

- Company hosts software (e.g. CRM, HRM, ERP)
  - Provides software and datacenter
- Takes care of maintenance, upgrades, support, …
- Pay per use (seats, transactions)
- Example:
  - Online Tax Filing
  - Salesforce.com
  - Gmail/hotmail/…

The final word (before the final)

- Thanks for joining the class
  (I know it is a mandatory class, but …)
- Its been a fun class
- I hope you learned how operating systems at a high level function and how they interact with platform/system
- Good Luck with the Final Exam