Distributed Systems

Operating Systems

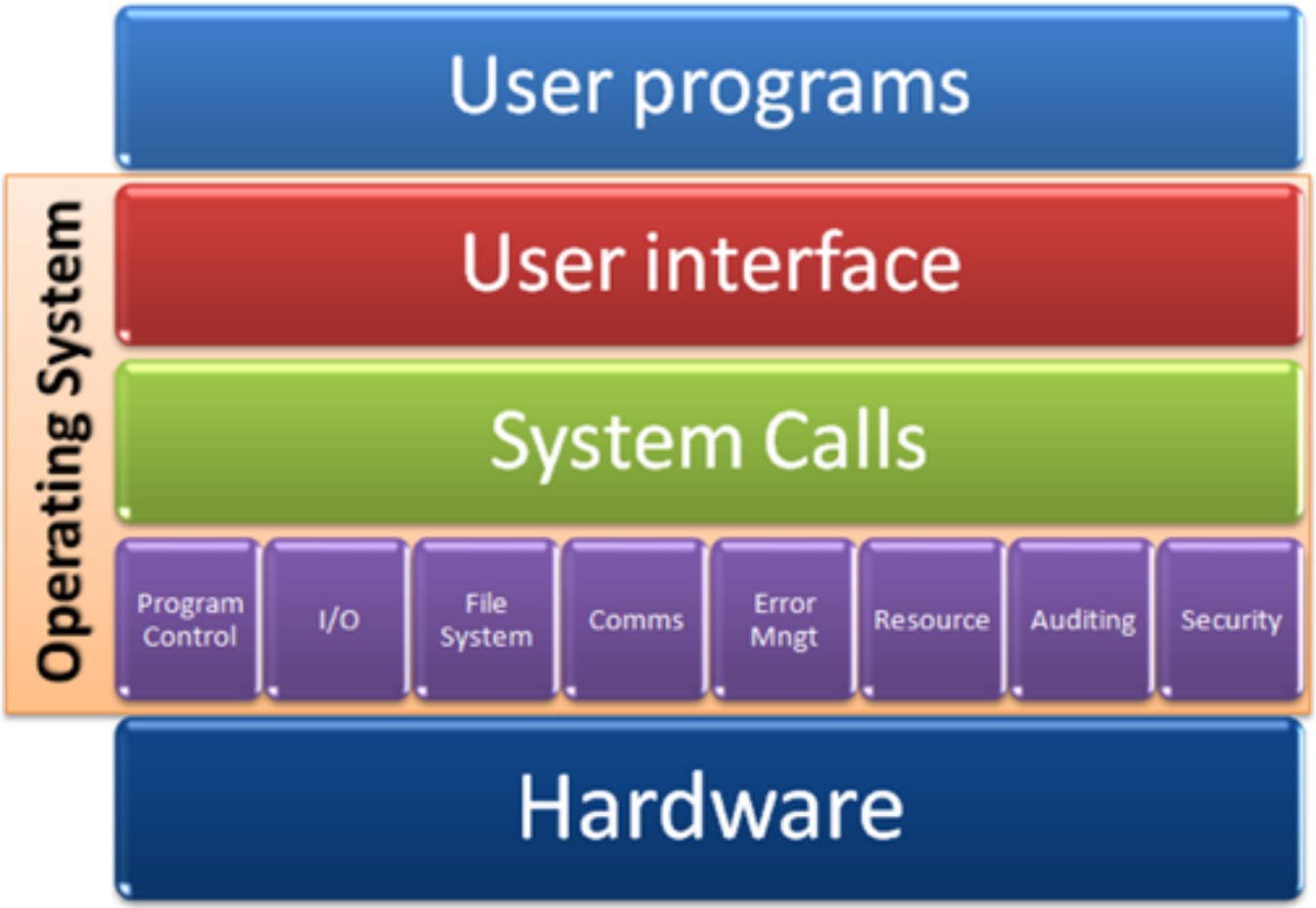
Björn Franke

University of Edinburgh
2015/2016

Overview

- Operating Systems
- Networked Operating Systems
- Distributed Operating Systems
- Virtualisation
- Current Trends

Operating System



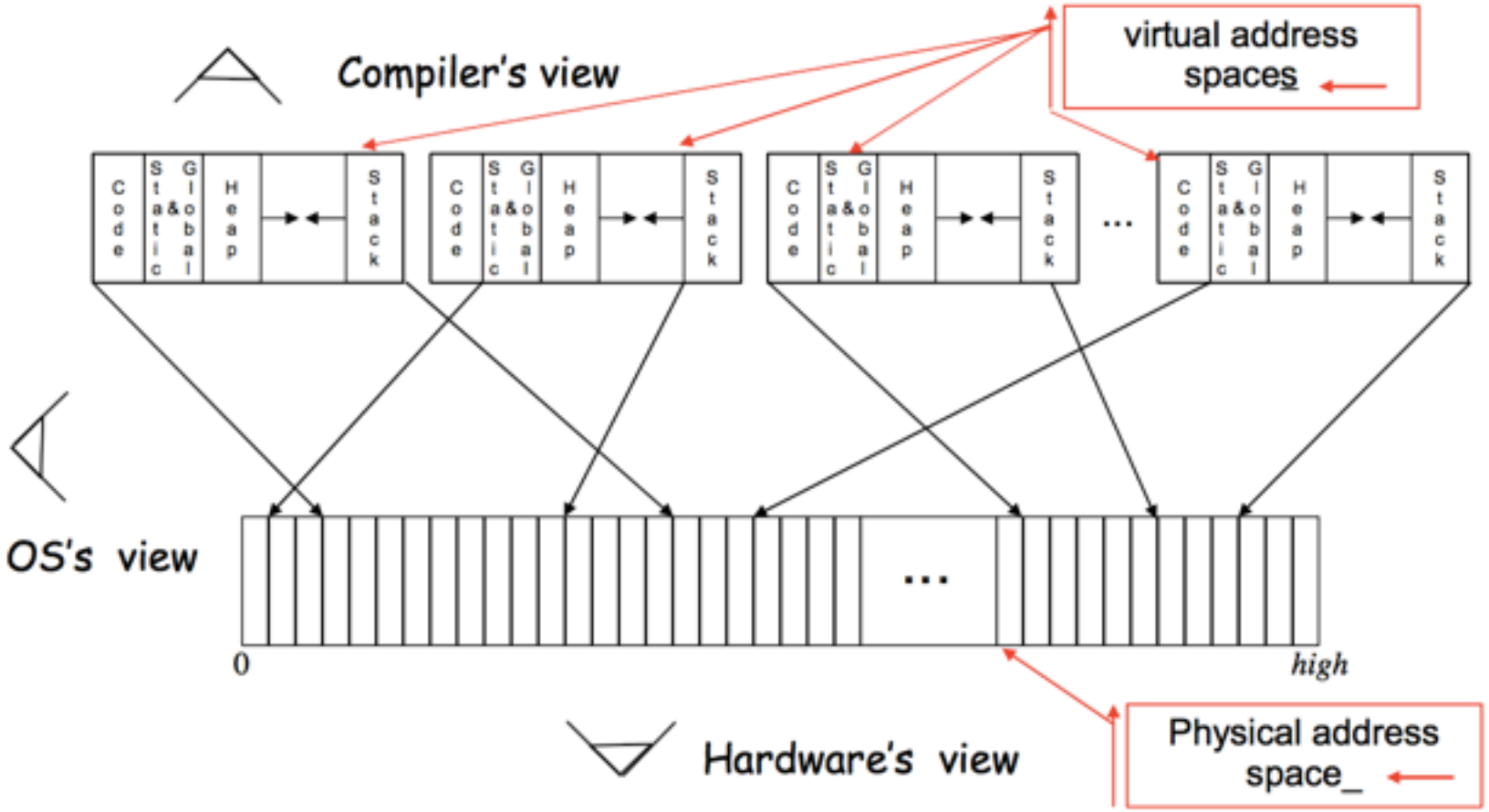
Operating System

- What is an operating system?
- An operating system is a **resource manager**
- Provides an **abstract computing interface**
- OS **arbitrates resource usage** between processes
  - CPU, memory, filesystem, network, keyboard, mouse, monitor
  - Other hardware
- This makes it possible to have **multiple processes** in the same system
  - If 2 processes ask for use of same resource
  - OS decides who gets is when, how much etc.

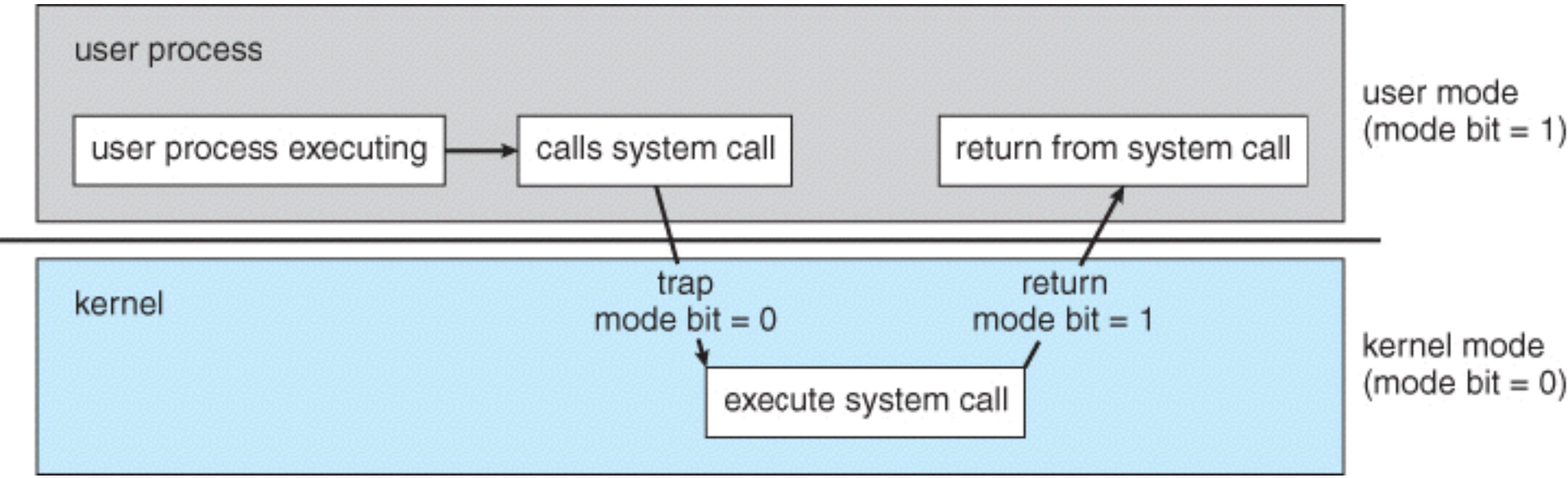
Operating System

- How OS handles different resources
- Memory:
  - Each process is given a different part of memory to use, they cannot access other's memory
  - If it needs more memory, OS will allocate from unallocated memory store
- Filesystem
  - OS checks that process has rights to read/write the file
  - Makes sure that 2 processes are not writing the same file
- Network:
  - OS receives messages from processes, sends them to network card one at a time
  - When messages are received, OS delivers to suitable processes

Virtual Memory



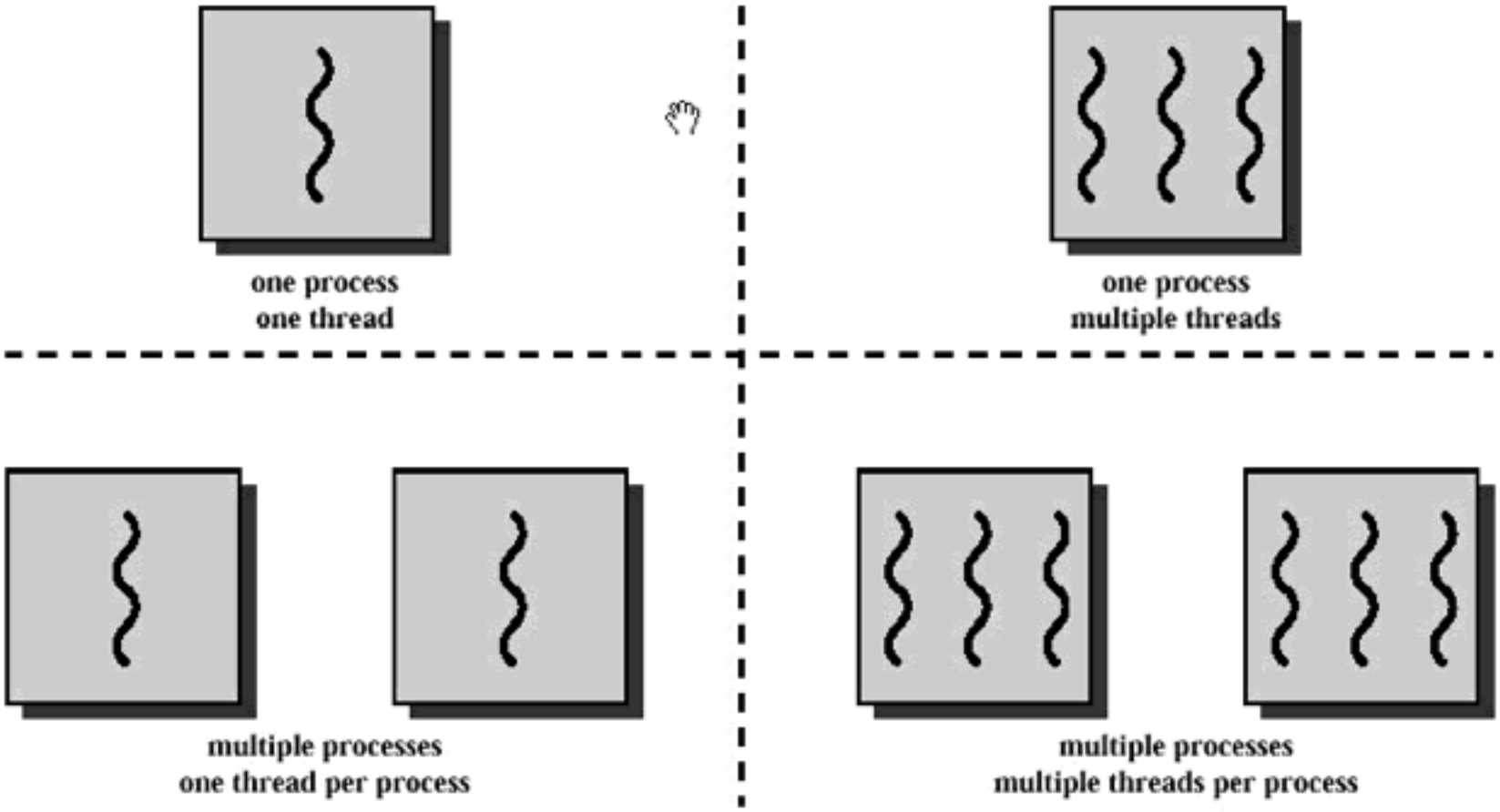
Kernel/User Mode Operation



Operating System

- OS makes processes *oblivious* of environment
- Process does not know details of hardware
- Process does not know about other processes (unless they communicate with each-other)

Threads


one process
one thread
one process
multiple threads
multiple processes
one thread per process
multiple processes
multiple threads per process

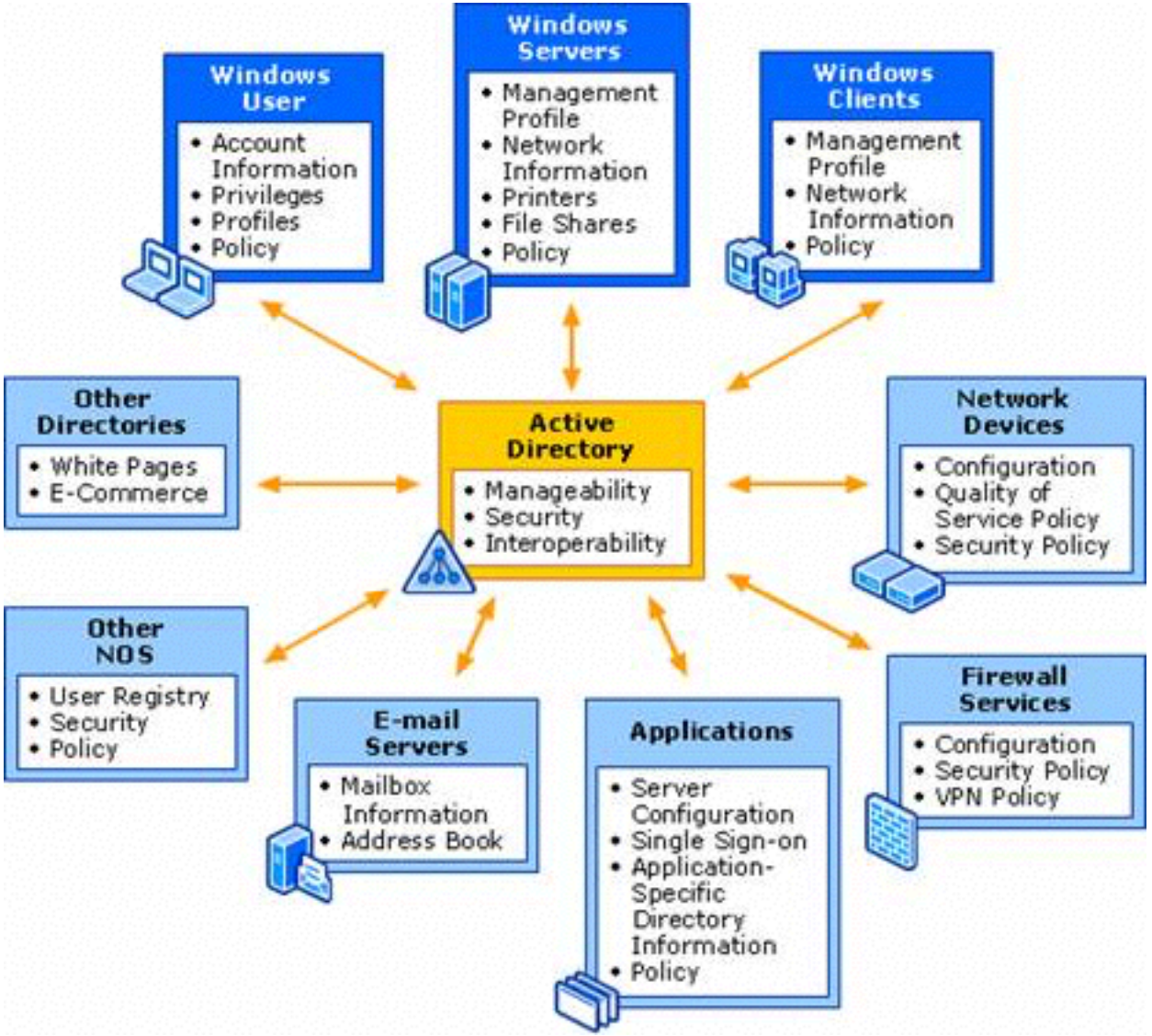
Benefits of Threads

- **Responsiveness**: even if part of program is blocked or performing lengthy operation multithreading allow a program to continue.
- **Resource Sharing**: threads share the memory & resources of the process within the same address space.
- **Economy**: Allocating memory & resources for process creation is costly. Threads share resources of the process to which it belongs. Create and context switch threads is more economical.
- **Utilisation of multicore Architectures**: In multicore system, threads running in parallel on different cores.

Networked OS (any standard OS)

- A networked OS is aware that it is connected to the network
- Every node has an OS running
- Every node manages the resources at that node
- A process can request communication to processes in other nodes
  - It has to be explicitly aware that it is requesting service at at different node
  - And which node it is requesting (eg. I.P. address)
  - So it also has to know which services/resources are aailable in the netwok
- A process cannot request resources in control of a different computer
- It has to communicate with a process on that computer and request it to do the job
- Distributed computing has to be done explicitly

Networked Operating System



Distributed Operating System

- OSes running on the different computers act like a single OS
- Process does not get to know (or need to know) that other resources/processes are at other computers
  - Process gets input/output from hardware X, which can be on any computer
  - Process A communicates with process B the same way whether they are on same computer or not
  - OS takes care of using the network if needed
- A process may be running on a different computer from where it was started. Processes can be moved among different computers
- The “distributed” nature of the system is hidden from the processes
- The OS manages all the “distributed” aspects

Distributed Operating System

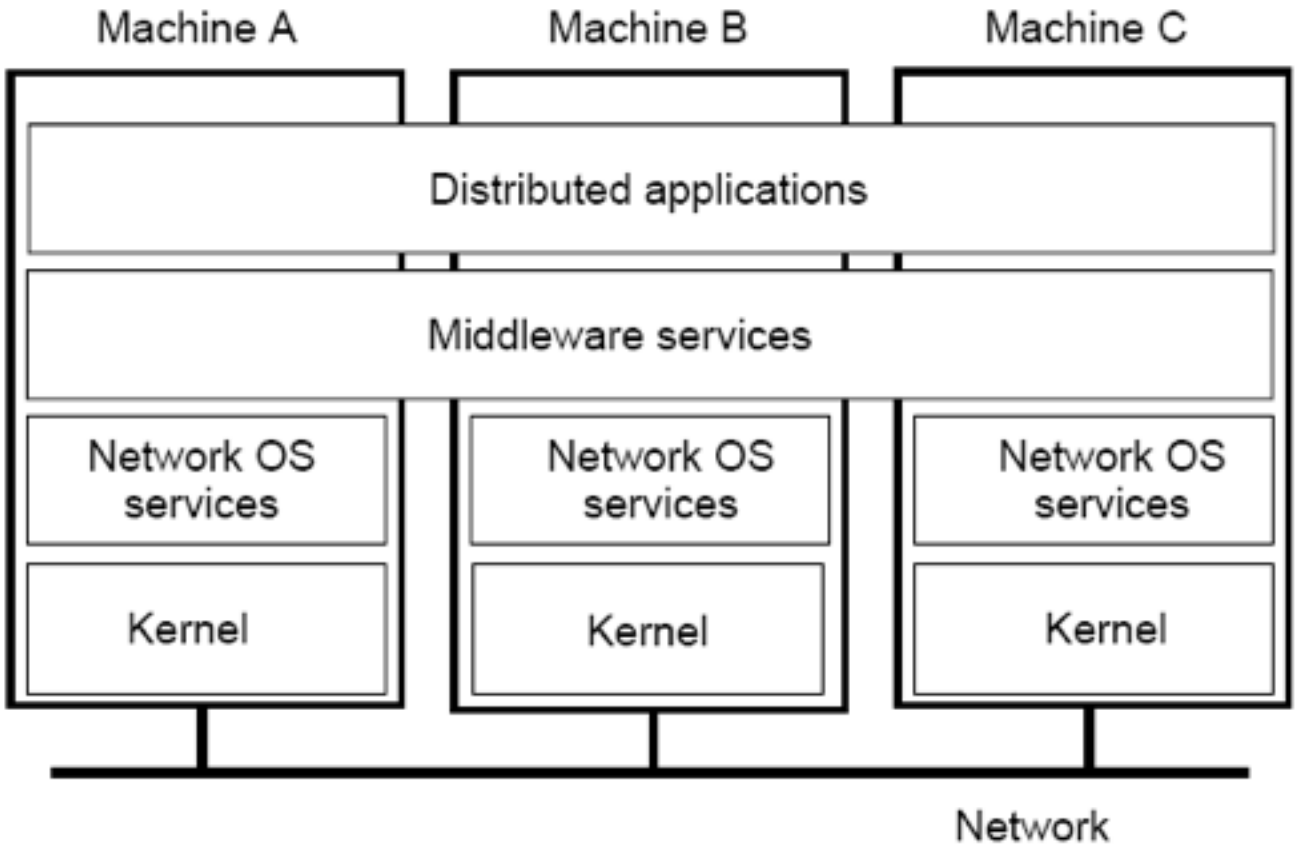


Fig. 1-22. General structure of a distributed system as middleware.

Distributed Operating System

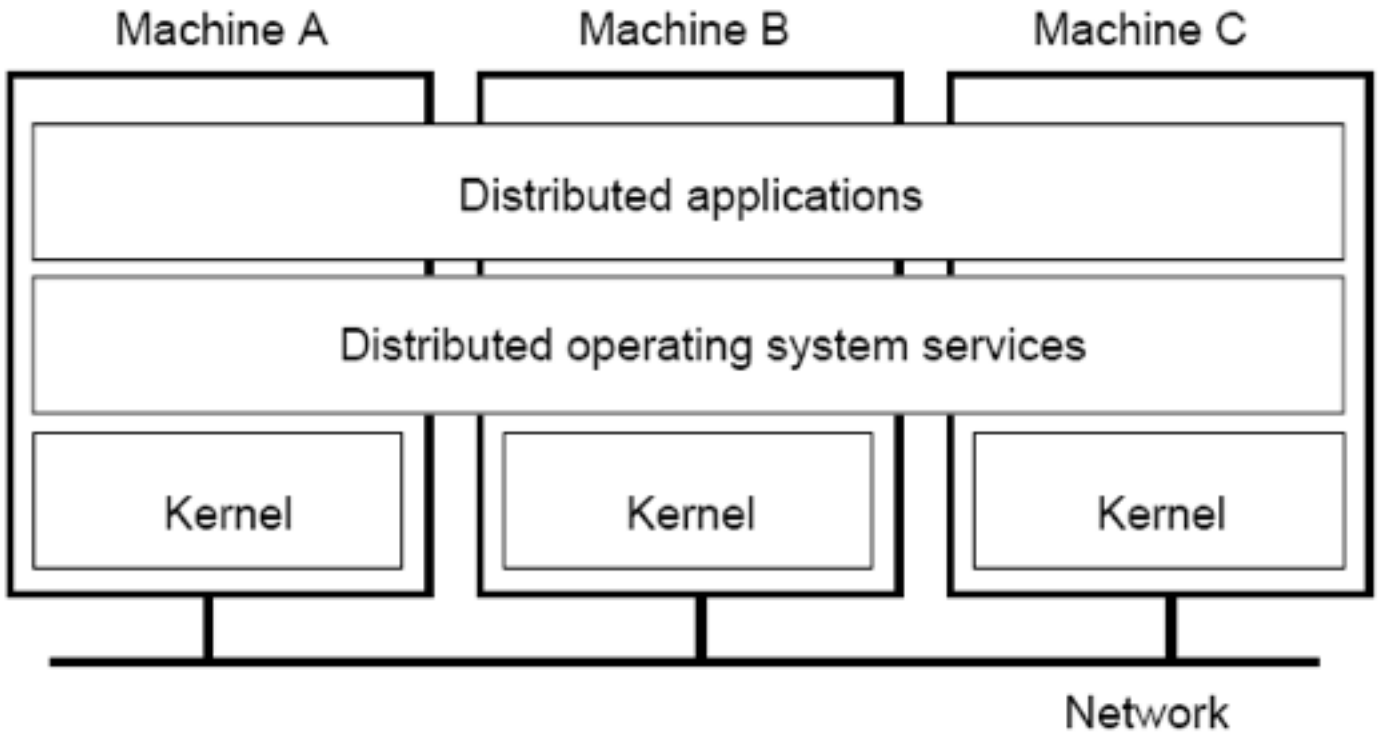


Fig. 1-14. General structure of a multicomputer operating system.

Distributed OS

- One interface to all resources in the network
- Regular program can be made to run in a distributed fashion
- Easier to program applications that make use of networked resources
- Or is it?

Problems with Distributed OS

- What happens if part of the network fails, and processes are separated into 2 sets?
  - Now we have to tell processes that the network has failed, and process has to take action
  - What if some OS-processes were moved elsewhere?
- Suppose we start processes A and B on the same computer
  - OS moves them to different computers
  - But A and B communicate a lot, so it would have been efficient to have them on the same computer!

Problems with Distributed OS

- Access to offsite resources
  - Has to be through explicit network connection
  - All computers in the world cannot be in same system!
- Adding new nodes to a distributed computing
  - May be part of a different instance of the OS
  - We will still need explicit connections
- Distributed OS does not help a lot with distributed computing

Problems with Distributed OS

- A network/computer failure means part of the OS failed
  - Hard to design OS with tolerance to such failures
- Distributed OS has to allow for lots of different possibilities in distributed computing
  - Harder to design
  - In fact, it is not possible to allow for all different possibilities
- "Distributed computing" means different things in different cases
- Better to let the application programmer decide how it will be distributed, and how to handle communication, failure etc
- OS provides only the basic infrastructure

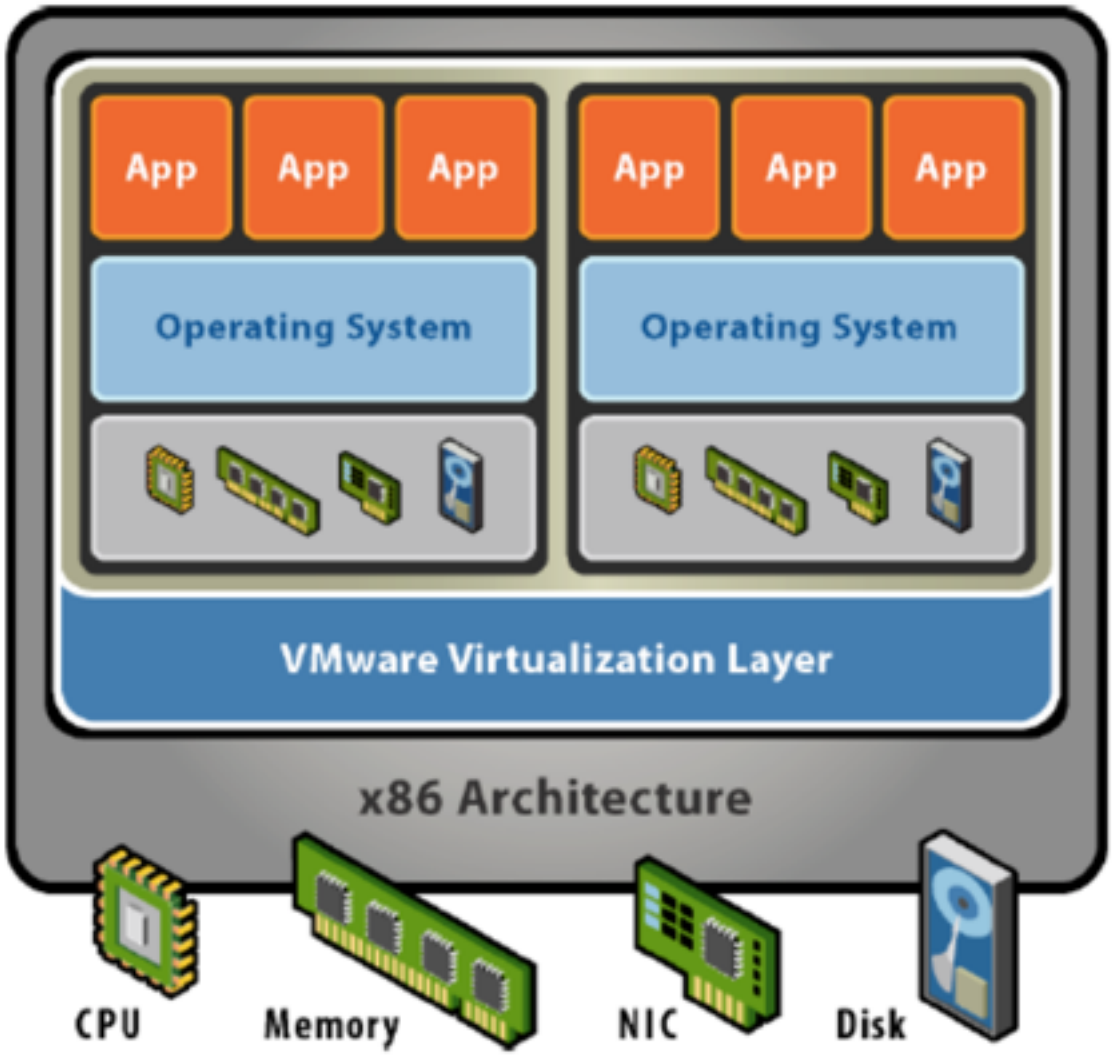
Networked OS vs Distributed OS

- As a result, we do not have any distributed OS in regular use
- Networked OS are popular
- Provide communication facilities
- Let software decide how they want to execute distributed computation
  - More flexibility
  - Failure etc are application's responsibility
  - OS continues to do basic tasks

Distributed Computation and Networked OS

- Use distributed algorithms at the **application layer** for
  - Synchronization
  - Consistent ordering
  - Mutual Exclusion
  - Leader election
  - Failure detection
  - Multicast
  - Etc..
- And design distributed computing applications
- Different applications will need different sets of features

Virtualisation



Virtualisation

- **Multiple operating system instances** to run **concurrently** within **virtual machines** on a **single computer**, dynamically partitioning and sharing the available physical resources such as CPU, storage, memory and I/O devices.
- **Hosted** or a **hypervisor** architecture.
  - **Hosted** architecture installs and runs the virtualization layer as an application on top of an operating system contrast
  - **Hypervisor** (bare-metal) architecture installs the virtualization layer directly on a clean x86-based system.
    - Direct hardware access: more efficient, greater scalability, robustness and performance

Virtualisation

- Sandboxing
- Testing
- Backup
- Fault-tolerance
- Migration
- Consolidation
- …

Virtualisation & Distributed Computing

- Consider a server farm
- Many different servers are running
- Instead of giving a physical server to each, many server farms consist of real servers running virtual machines
- For example, renting a server to host a web site is likely to give you a VM based server

Virtualisation & Distributed Domputing

- Advantages: more flexibility
  - Multiple VMs on same computer
    - Need fewer physical machines
  - Easier to turn on/off
  - Easier to backup
  - VMs can be moved from one computer to another while preserving state
    - Useful when the work load changes, some servers need more computation, others need less..

Virtualisation & Distributed Computing

- This is *not* a good strategy for CPU intensive computation such a large data mining
- Because running a large computation in a virtual machine is inefficient
- However, many systems need computation running all the time, but not so intensively
- Virtualisation is most useful when flexibility is critical

Current Trends

- Mobile
  - Heavily contested area
  - Adaptation to mobility
  - Harder to network when moving
  - Adaptation to low energy system
  - Different style of user interaction
  - Needs better synchronization across multiple mobile user devices

Current Trends

- Sensors
  - For sensor networks
  - TinyOS, LiteOS, Contiki
  - Small, low power sensor devices
  - Needs efficient operation
  - Needs specialization to process and handle sensor data and related operations in place of application interface

Current Trends

- Embedded systems
  - Computers all around us, in every device/ machine
  - Needs OS and Distributed Computing, since they need to communicate with each-other
  - Adaptation to low power, low resource environment
  - Has to run without supervision/interaction