

Multiprocessor ports of operating systems

Simon Kågström
ska@bth.se

Department of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
P.O. Box 520, SE-372 25 Ronneby, Sweden

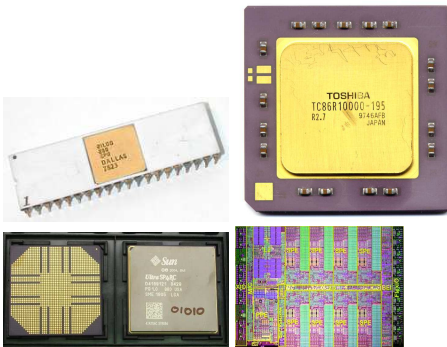
Multiprocessor ports of operating systems – p. 1

Outline

- Introduction and background
- Problems when porting to a multiprocessor
- Multiprocessor operating system organizations
 - How they handle concurrency issues
 - Changes to the uniprocessor system
- Case-study of Linux evolution (preliminary)

Multiprocessor ports of operating systems – p. 2

Introduction



Multiprocessor ports of operating systems – p. 3

Things you already know

- System calls
- Interrupts
- Privileged instructions
- Spinlocks
- Privilege levels
 - User-level
 - Kernel-level
 - (Hardware)

Multiprocessor ports of operating systems – p. 4

Problems, I

- Why cannot uniprocessor operating systems use multiprocessor hardware unmodified?

Multiprocessor ports of operating systems – p. 5

Problems, I

- Why cannot uniprocessor operating systems use multiprocessor hardware unmodified?
- Concurrency problems!
 - What happens when two processors attempt to access a shared structure at the same time?

Multiprocessor ports of operating systems – p. 5

Problems, I

- Why cannot uniprocessor operating systems use multiprocessor hardware unmodified?
- Concurrency problems!
 - What happens when two processors attempt to access a shared structure at the same time?
- Scalability problems!
 - 512 procesors accessing a single run queue will waste lots of time in lock contention, cache ping-pong etc.

Multiprocessor ports of operating systems – p. 5

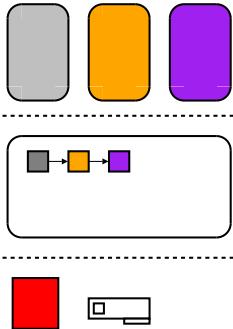
Problems, II

- Uniprocessor operating systems often assume only one process in the kernel
 - Simplifies kernel construction, no cross-process locks
 - But: this rule breaks as soon as multiple processors execute in the kernel
- UP kernels often protect code from interrupts by disabling them
 - No longer safe with multiple processors

So: when adding SMP support, the original OS must be changed

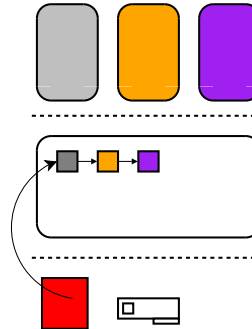
Multiprocessor ports of operating systems – p. 6

Problems, example



Multiprocessor ports of operating systems – p. 7

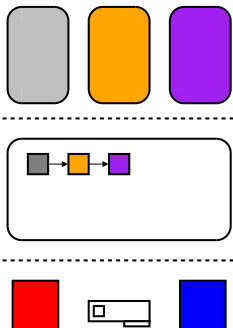
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues

Multiprocessor ports of operating systems – p. 7

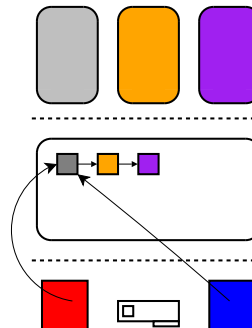
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues

Multiprocessor ports of operating systems – p. 7

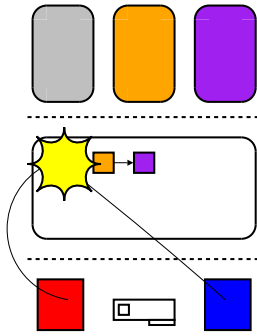
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?

Multiprocessor ports of operating systems – p. 7

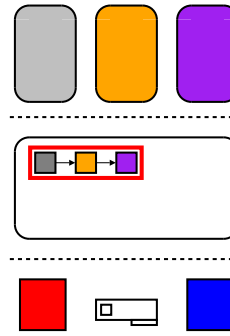
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?

Multiprocessor parts of operating systems – p. 7

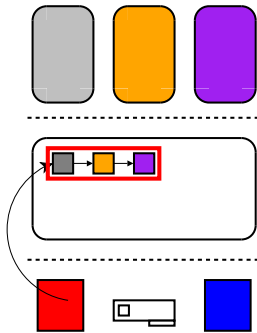
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?
- This requires locking

Multiprocessor parts of operating systems – p. 7

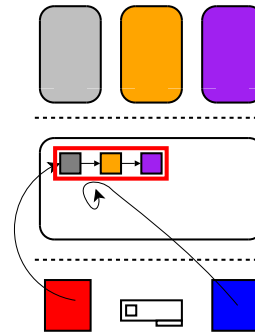
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?
- This requires locking

Multiprocessor parts of operating systems – p. 7

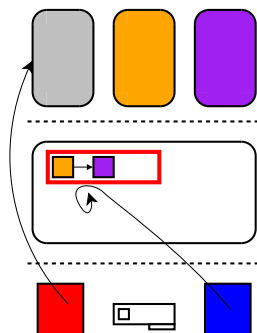
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?
- This requires locking

Multiprocessor parts of operating systems – p. 7

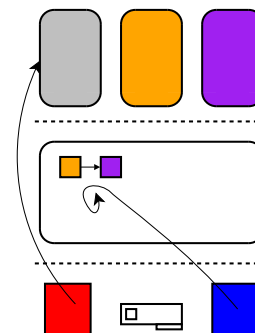
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?
- This requires locking

Multiprocessor parts of operating systems – p. 7

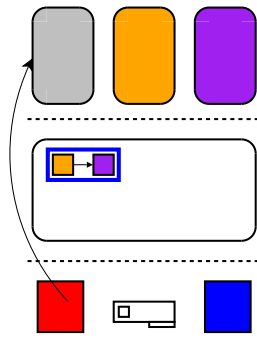
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?
- This requires locking

Multiprocessor parts of operating systems – p. 7

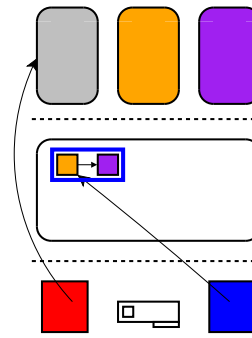
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?
- This requires locking

Multiprocessor parts of operating systems - p. 7

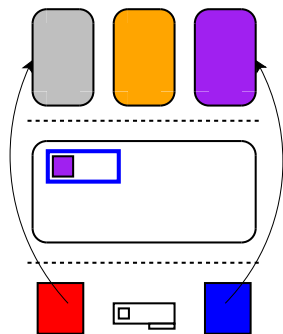
Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?
- This requires locking

Multiprocessor parts of operating systems - p. 7

Problems, example



- On a uniprocessor, we can access a structure without worrying about concurrency issues
- What happens when multiple processors access a common structure?
- This requires locking

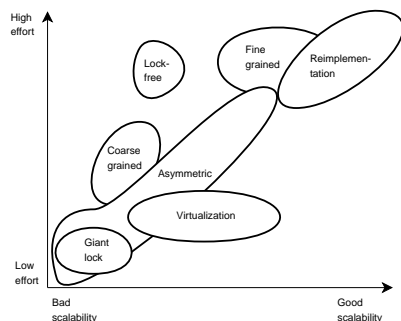
Multiprocessor parts of operating systems - p. 7

Multiprocessor operating systems

- There are different ways of organizing a multiprocessor operating system
 - Giant / coarse-grained / fine-grained locking
 - Lock-free operating systems
 - Assymmetric approaches
 - Virtualization
 - API/ABI compability and reimplementatation
- The requirements on the operating system support depends on the applications
 - For compute-bound applications, almost any method will do
 - I/O-bound applications require a highly parallel OS
- Performance requirements for large systems (> 8 processors) are different than on small

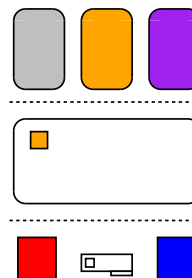
Multiprocessor parts of operating systems - p. 8

Performance/Complexity



Multiprocessor parts of operating systems - p. 9

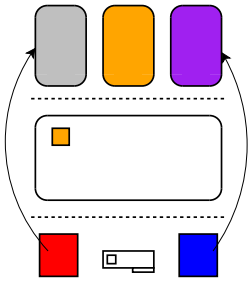
Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor parts of operating systems - p. 10

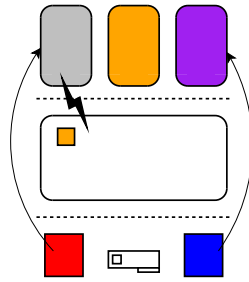
Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor ports of operating systems – p. 10

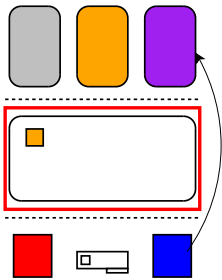
Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor ports of operating systems – p. 10

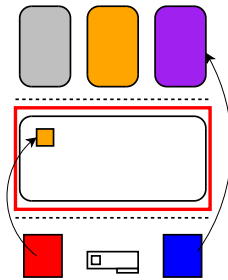
Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor ports of operating systems – p. 10

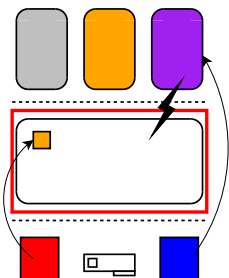
Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor ports of operating systems – p. 10

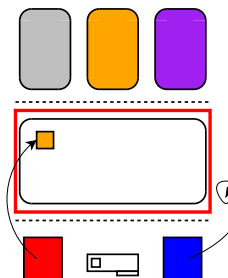
Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor ports of operating systems – p. 10

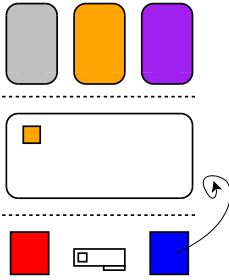
Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor ports of operating systems – p. 10

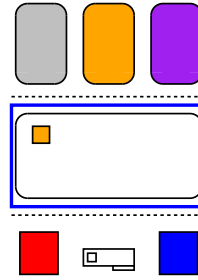
Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor ports of operating systems – p. 10

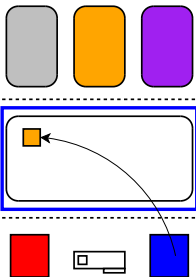
Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor ports of operating systems – p. 10

Giant locking



- The giant lock approach locks the entire kernel
- Scalability is bad for kernel-bound applications, but requires relatively small changes
- Examples: Linux 2.0, early FreeBSD, own work

Multiprocessor ports of operating systems – p. 10

Lock-free operating systems

- Do we need locking at all?
 - With many processors: **No!**
- Many processors has support for one or more atomic instructions like **test-and-set**
- These instructions can be used to implement lock-free algorithms for stacks, lists, queues etc.
- Larger areas are protected by replaying failed transactions
- Since the algorithms become more complex, lock-free operation is only beneficial if the contention is otherwise high
 - Difficult to compete with fine-grained locking (tas instructions can be expensive to execute)
- Examples: Synthesis, Cache kernel
 - Both from the early 90s

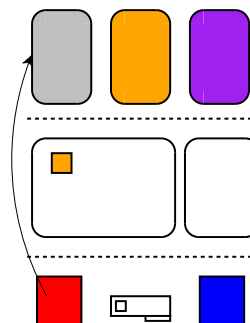
Multiprocessor ports of operating systems – p. 11

Assymetric approaches

- Most operating systems divide tasks between processors *symmetrically*
 - All processors can perform all tasks
- Some systems are *assymetric* instead
 - In these, different processors have different tasks
 - One processor can be tied to handling kernel code, the rest user code only
 - Or one processor might handle a certain hardware device
- Examples: Piglet, Application kernel, master-slave systems

Multiprocessor ports of operating systems – p. 12

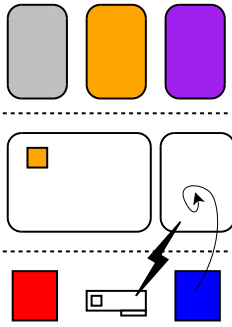
Assymetric, piglet



- Piglet is an assymetric approach which divides CPUs into *application* and *device* processors
- For example, TCP/IP handling and a NIC can be handled by a device processor
- Beneficial if I/O handling dominates the workload
- Small changes (almost no) if the core OS is still UP
- Examples: Piglet, intelligent network devices (Myrinet etc.)

Multiprocessor ports of operating systems – p. 13

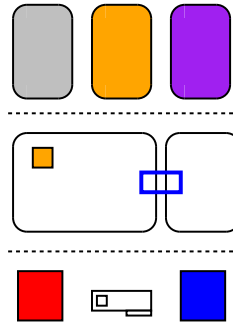
Assymmetric, piglet



- Piglet is an assymmetric approach which divides CPUs into *application* and *device* processors
- For example, TCP/IP handling and a NIC can be handled by a device processor
- Beneficial if I/O handling dominates the workload
- Small changes (almost no) if the core OS is still UP
- Examples: Piglet, intelligent network devices (Myrinet etc.)

Multiprocessor ports of operating systems – p. 13

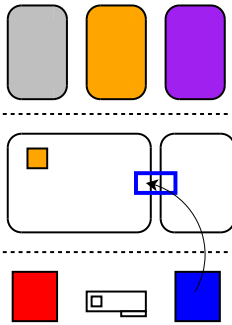
Assymmetric, piglet



- Piglet is an assymmetric approach which divides CPUs into *application* and *device* processors
- For example, TCP/IP handling and a NIC can be handled by a device processor
- Beneficial if I/O handling dominates the workload
- Small changes (almost no) if the core OS is still UP
- Examples: Piglet, intelligent network devices (Myrinet etc.)

Multiprocessor ports of operating systems – p. 13

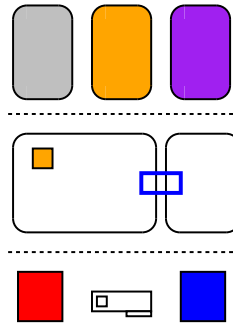
Assymmetric, piglet



- Piglet is an assymmetric approach which divides CPUs into *application* and *device* processors
- For example, TCP/IP handling and a NIC can be handled by a device processor
- Beneficial if I/O handling dominates the workload
- Small changes (almost no) if the core OS is still UP
- Examples: Piglet, intelligent network devices (Myrinet etc.)

Multiprocessor ports of operating systems – p. 13

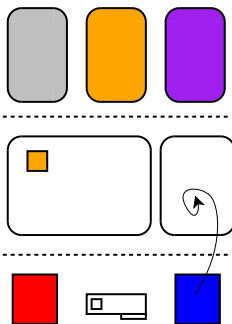
Assymmetric, piglet



- Piglet is an assymmetric approach which divides CPUs into *application* and *device* processors
- For example, TCP/IP handling and a NIC can be handled by a device processor
- Beneficial if I/O handling dominates the workload
- Small changes (almost no) if the core OS is still UP
- Examples: Piglet, intelligent network devices (Myrinet etc.)

Multiprocessor ports of operating systems – p. 13

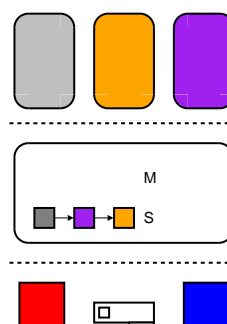
Assymmetric, piglet



- Piglet is an assymmetric approach which divides CPUs into *application* and *device* processors
- For example, TCP/IP handling and a NIC can be handled by a device processor
- Beneficial if I/O handling dominates the workload
- Small changes (almost no) if the core OS is still UP
- Examples: Piglet, intelligent network devices (Myrinet etc.)

Multiprocessor ports of operating systems – p. 13

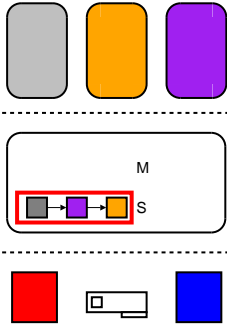
Master-slave



- Master-slave systems are simple assymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems – p. 14

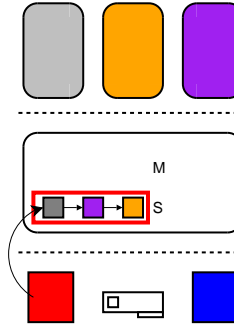
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

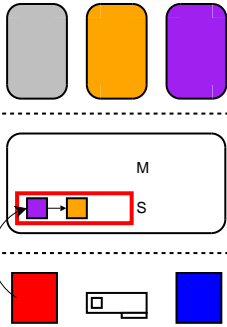
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

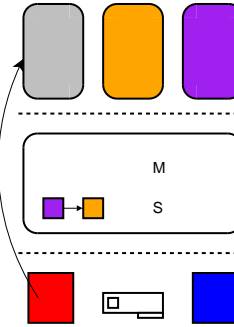
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

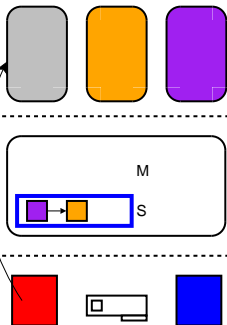
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

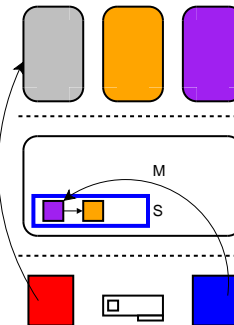
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

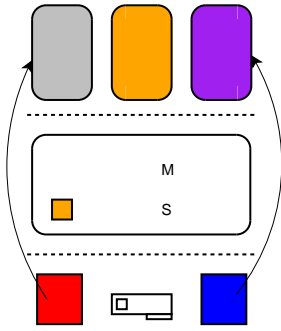
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

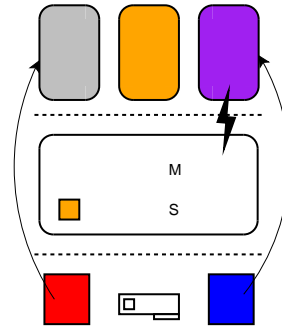
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

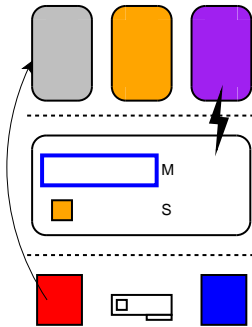
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

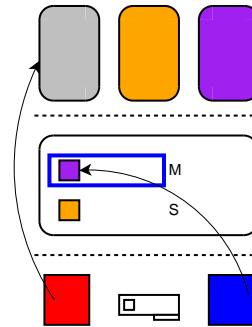
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

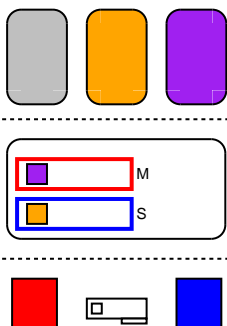
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

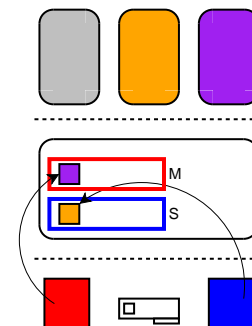
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

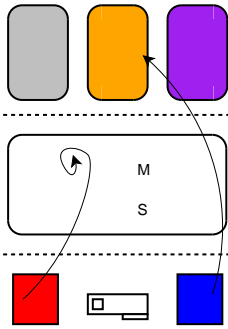
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

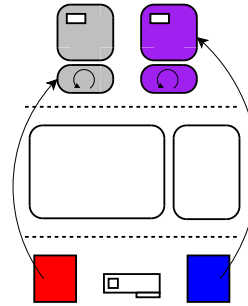
Master-slave



- Master-slave systems are simple asymmetric systems
- One processor acts as "master" (in-kernel operations)
- The other processors are "slaves" (user mode)
- Scalability bad for kernel-bound applications
- Used in various older kernels, limited changes

Multiprocessor ports of operating systems - p. 14

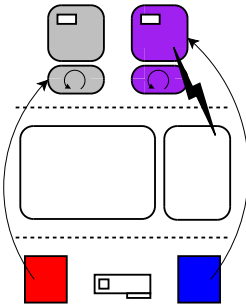
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems - p. 15

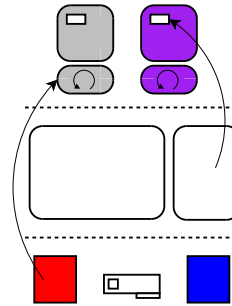
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems - p. 15

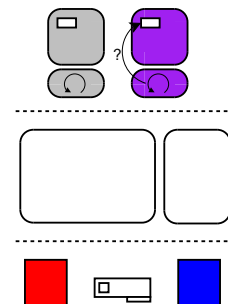
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems - p. 15

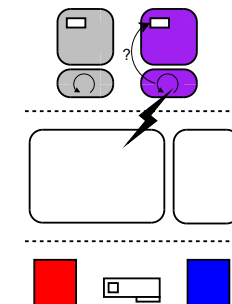
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems - p. 15

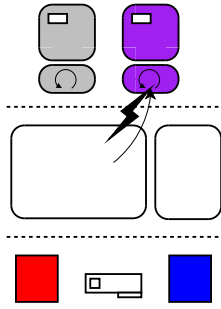
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems - p. 15

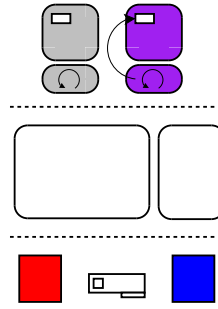
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems – p. 15

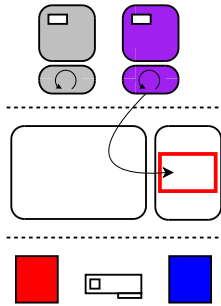
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems – p. 15

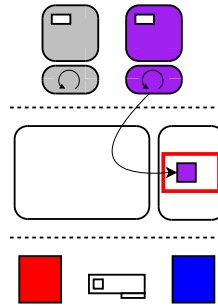
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems – p. 15

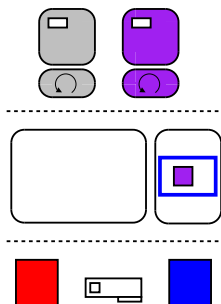
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems – p. 15

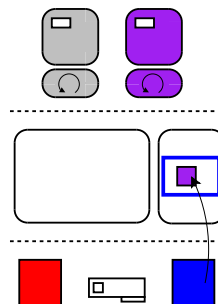
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems – p. 15

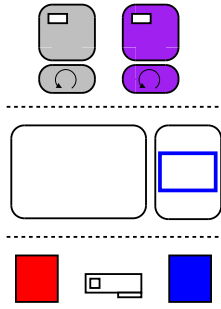
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems – p. 15

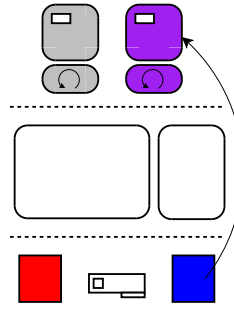
Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

Multiprocessor ports of operating systems – p. 15

Appkern



- The application kernel is a hybrid master-slave system
- Keeps the original kernel unchanged, MP support added as a driver: small or no changes
- User-level "bootstrap thread" handles kernel access
- Scalability is bad for kernel-bound applications
- Examples: The application kernel for Linux

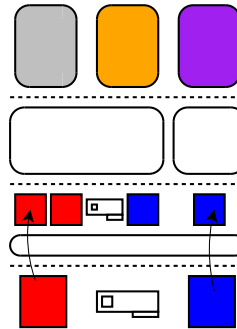
Multiprocessor ports of operating systems – p. 15

Virtualization

- A completely different approach is to run the operating system in a virtual machine
 - Perhaps as a fast cluster
- With this approach, a small microkernel can be used and "guest OSes" run on top of it
- Supports sharing of big machines
 - It might be hard to achieve scalability for those machines anyway
 - Many operating systems scale to a few CPUs only
- Can also provide fault isolation
 - By separating resources into groups ("cells")

Multiprocessor ports of operating systems – p. 16

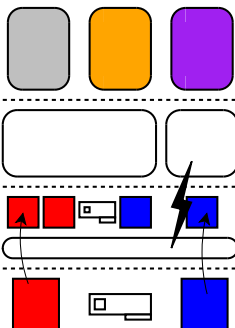
Virtualization



- Virtualization adds a software layer between the operating system and the hardware (the *hypervisor*)
- The OS(es) run on virtual CPUs, *VCPUs*

Multiprocessor ports of operating systems – p. 17

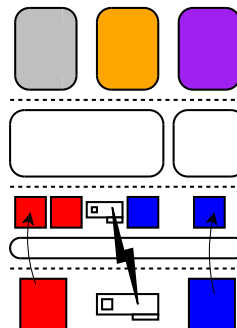
Virtualization



- Virtualization adds a software layer between the operating system and the hardware (the *hypervisor*)
- The OS(es) run on virtual CPUs, *VCPUs*

Multiprocessor ports of operating systems – p. 17

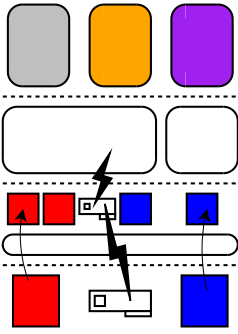
Virtualization



- Virtualization adds a software layer between the operating system and the hardware (the *hypervisor*)
- The OS(es) run on virtual CPUs, *VCPUs*

Multiprocessor ports of operating systems – p. 17

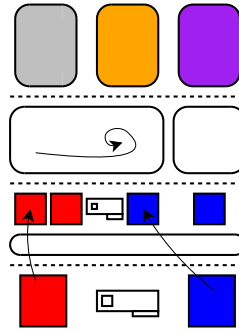
Virtualization



- Virtualization adds a software layer between the operating system and the hardware (the *hypervisor*)
- The OS(es) run on virtual CPUs, *VCPUs*

Multiprocessor ports of operating systems – p. 17

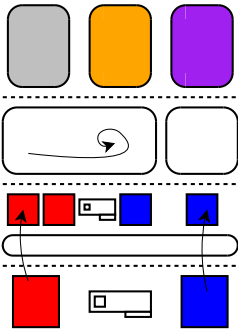
Virtualization



- Virtualization adds a software layer between the operating system and the hardware (the *hypervisor*)
- The OS(es) run on virtual CPUs, *VCPUs*
- What happens on spinning for locks?

Multiprocessor ports of operating systems – p. 17

Virtualization



- Virtualization adds a software layer between the operating system and the hardware (the *hypervisor*)
- The OS(es) run on virtual CPUs, *VCPUs*
- What happens on spinning for locks?
- Examples: L4Ka, Cellular Disco, Adeos nanokernel

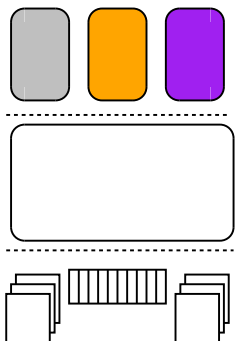
Multiprocessor ports of operating systems – p. 17

ABI/API compatibility

- A final method is to provide ABI/API compatibility to the uniprocessor system with a different kernel
- This can either mean reimplementing it fully
 - Feasible if the changes to the original kernel would be very large to accommodate the target hardware
 - K42 is a Linux-compatible OS for large systems implemented from scratch
- Or providing a compatibility layer in an already existing OS
 - Might not always be a perfect match

Multiprocessor ports of operating systems – p. 18

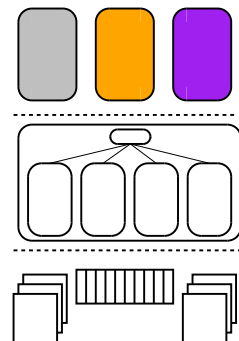
ABI/API compatibility



- K42 optimizations:
- Few global data structures
 - E.g., the pool of physical memory is partitioned between processors
 - Advantage for large-scale systems, disadvantage otherwise
- Hot-swapping of components

Multiprocessor ports of operating systems – p. 19

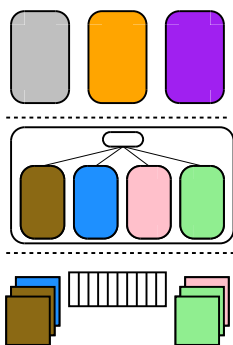
ABI/API compatibility



- K42 optimizations:
- Few global data structures
 - E.g., the pool of physical memory is partitioned between processors
 - Advantage for large-scale systems, disadvantage otherwise
- Hot-swapping of components

Multiprocessor ports of operating systems – p. 19

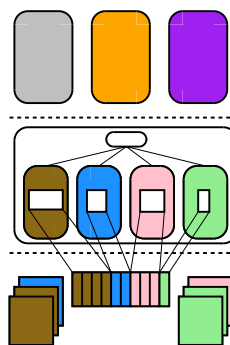
ABI/API compatibility



- K42 optimizations:
 - Few global data structures
 - E.g., the pool of physical memory is partitioned between processors
 - Advantage for large-scale systems, disadvantage otherwise
- Hot-swapping of components

Multiprocessor ports of operating systems – p. 19

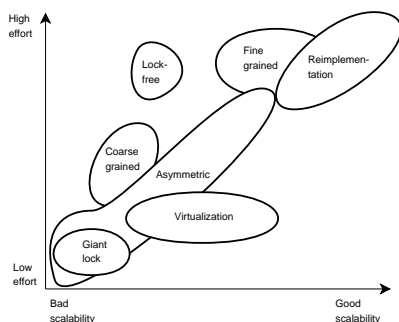
ABI/API compatibility



- K42 optimizations:
 - Few global data structures
 - E.g., the pool of physical memory is partitioned between processors
 - Advantage for large-scale systems, disadvantage otherwise
- Hot-swapping of components

Multiprocessor ports of operating systems – p. 19

Performance/Complexity, again



Multiprocessor ports of operating systems – p. 20

Linux evolution

- As a case-study, we look at the Linux evolution
- Performance is evaluated with the Postmark benchmark: heavily kernel bound, very hard
- 2.0: giant lock
 - One giant lock (BKL, “Big kernel lock”) protects the entire kernel
 - All interrupts are handled by *one* CPU
 - *Very* simple solution
- 2.2: Coarse-grained, some parallelism in the kernel
 - Run-queue lock etc.
 - Block-I/O lock
 - Most filesystem code (open, close etc. syscalls) still use the BKL

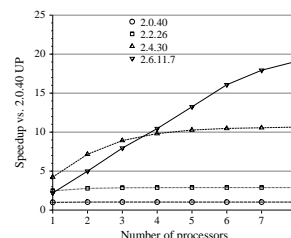
Multiprocessor ports of operating systems – p. 21

Linux evolution, II

- 2.4: Further granularity-improvements
 - Still common run-queue
 - Relaxations of BKL in filesystem code
- 2.6: Fine-grained
 - Many per-structure locks
 - Preemptible kernel
 - Support for NUMA architectures, O(1) scheduler with per-processor queues
 - Scheduling domains / CPU groups (lwn.net/Articles/80601/)

Multiprocessor ports of operating systems – p. 22

Linux evolution, performance



- 2.0, 2.2: Does not scale
- 2.4: Slight increase upto 3 processors
- 2.6: Almost perfect scaling
 - Why the jump 6-8 CPUs?

Multiprocessor ports of operating systems – p. 23

Linux evolution, code changes

- Preprocessed changes in `kernel/`, `mm/`, `arch/i386`, `include/asm-i386`
- The code has been indented and empty lines were removed

Version	Files		Lines		
	Nr	Changed	No SMP	With SMP	Modified
2.0.40	137	18	34176	34498	476
2.2.26	188	33	40969	41916	1115
2.4.30	246	37	53906	55141	1349
2.6.11.7	496	48	88988	90636	1761

- Conclusions?
 - Not conclusive
 - The affected code increases in absolute numbers

Questions?

Questions?