TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute of System Architecture, Operating Systems Group

RESOURCE MANAGEMENT

MICHAEL ROITZSCH

AGENDA

- done: time
- today: misc. resources
  - architectures for resource management
  - solutions for specific resources
- upcoming: applications, legacy support

KERNEL RESOURCES

PROBLEM

- kernel manages
  - tasks
  - threads
  - capabilities
  - mapping database
- it all comes down to memory
- kernel memory is limited
- opens the possibility of DoS attacks

IDEA

- memory management **policy** should not be in the kernel
- manage kernel memory in userland
- kernel provides memory control **mechanism**
- exception for bootstrapping:
  initial kernel memory is managed by kernel

SOLUTION

- kernel-memory objects in **L4.sec**
- extension of the recursive address space model
- create kernel-memory object by converting user pages
- userland apps pay for kernel memory allocated on their behalf
- converted pages no longer accessible by userland

SOLUTION

- kernel uses kernel-memory objects for storing allocated data structures
- supports map and unmap
- reconvert on unmap
- reconvert transfers object back to userland
- recursively deletes all kernel structures
- dependency graph must be cycle-free

PRINCIPLE

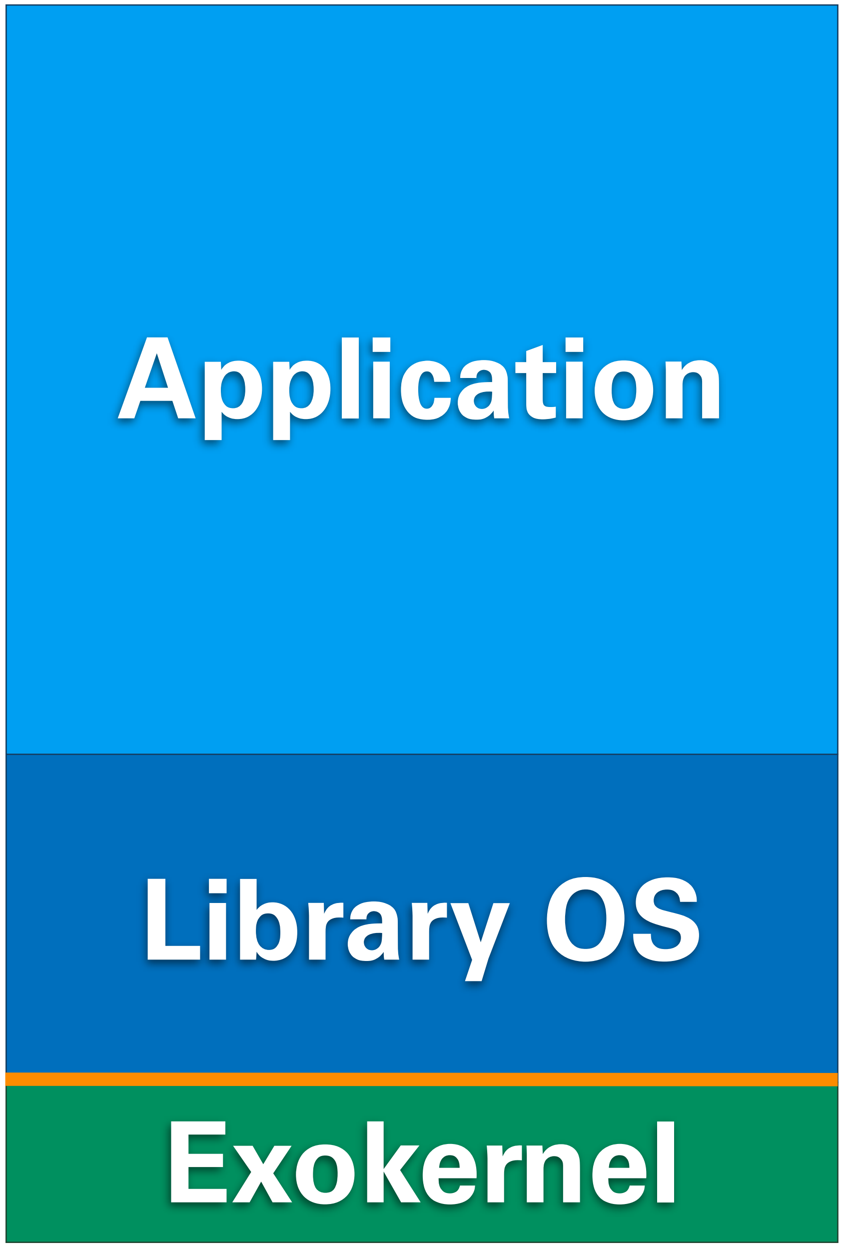
separate enforcement and management

ARCHITECTURES

EXOKERNEL

Management

Enforcement



DESIGN

- provide primitives at the **lowest possible level** necessary for protection
- use **physical names** wherever possible
- resource management primitives:
  - explicit allocation
  - exposed revocation
  - protected sharing
  - ownership tracking

CONSEQUENCES

- each application can use its own **library OS**
- library OS'es cannot trust each other
- no central management for global resources
- think of a file system
  - kernel manages disk ownership with block granularity
  - each library OS comes with its own filesystem implementation
- one partition per application?

SHARING

- invariants in shared resources must be maintained
- four mechanisms provided by the exokernel
  - **software regions** for sub-page memory protection, allows to share state
  - **capabilities** for access control
  - **critical sections**
  - **wakeup predicates:** code downloaded into the kernel for arbitrary checks

MULTISERVER

LEVELS

- different abstraction levels for resources

basic resources	**memory, CPU, IO-ports, interrupts**
hardware	**block device, framebuffer, network card**
compound resources	**file, GUI window, TCP session**

HIERARCHIES

- applications use multiple resources that depend on other resources
- resource tree of cooperating resource managers
- isolation of resources allows managers to provide real-time guarantees for their specific resource
  - DROPS:
    Dresden **R**eal-time **OP**erating **S**ystem

HIERARCHIES

- some resources are by themselves managed in a hierarchy
  - recursive virtual memory
- hierarchy can differentiate service
  - different memory properties by different pagers

EXAMPLES

ANKH

- driver for physical network card
- built with DDE using Linux 2.6 drivers
- provides multiple virtual network cards
- implements a simple virtual bridge

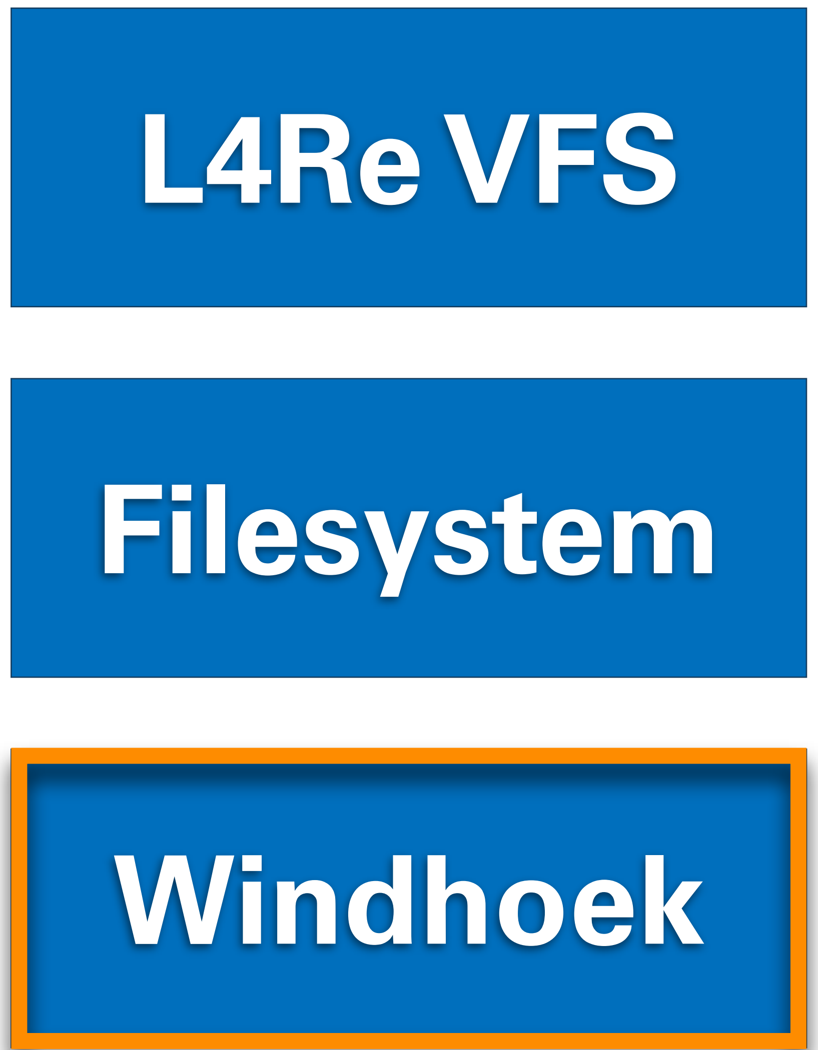
LWIP

- light-weight IP Stack
- TCP/IP stack
- also supports UDP, ICMP

WGET

- clients can use standard BSD socket interface

WINDHOEK



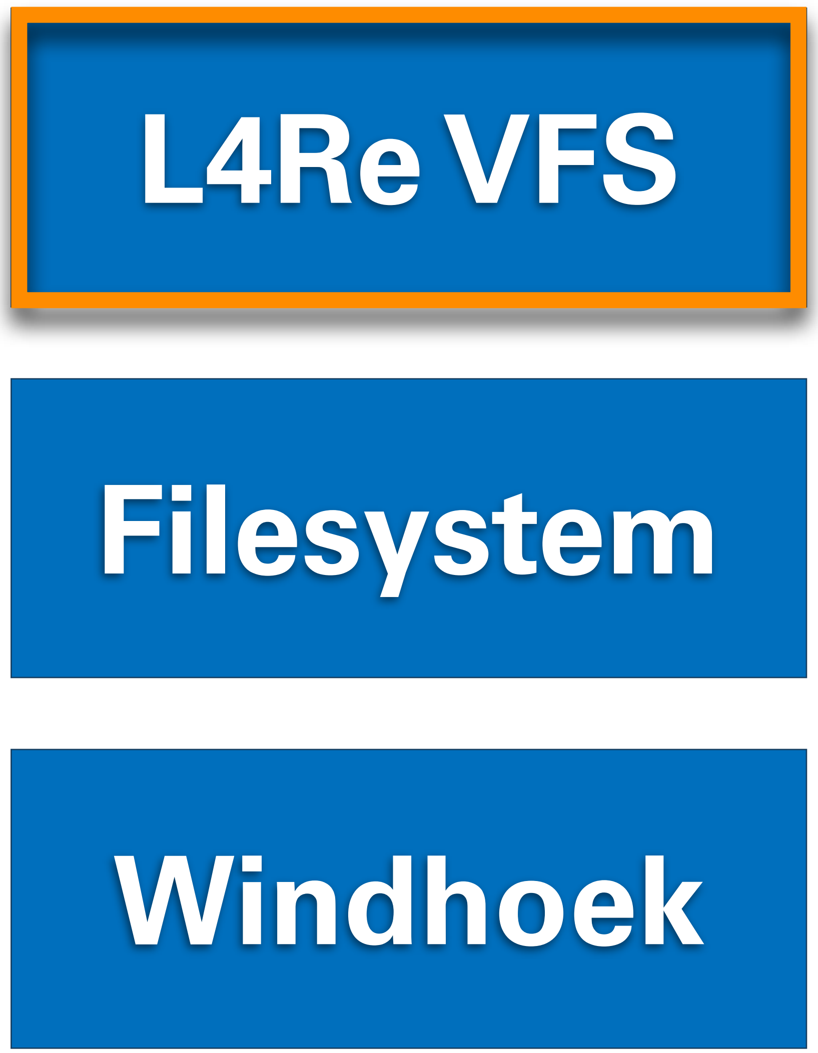
- IDE driver to access hard disks
- includes disk request scheduling
- based on DDE
- provides block device
- ongoing work on USB block devices

FILESYSTEM

- L4Re VFS
- **Filesystem**
- Windhoek

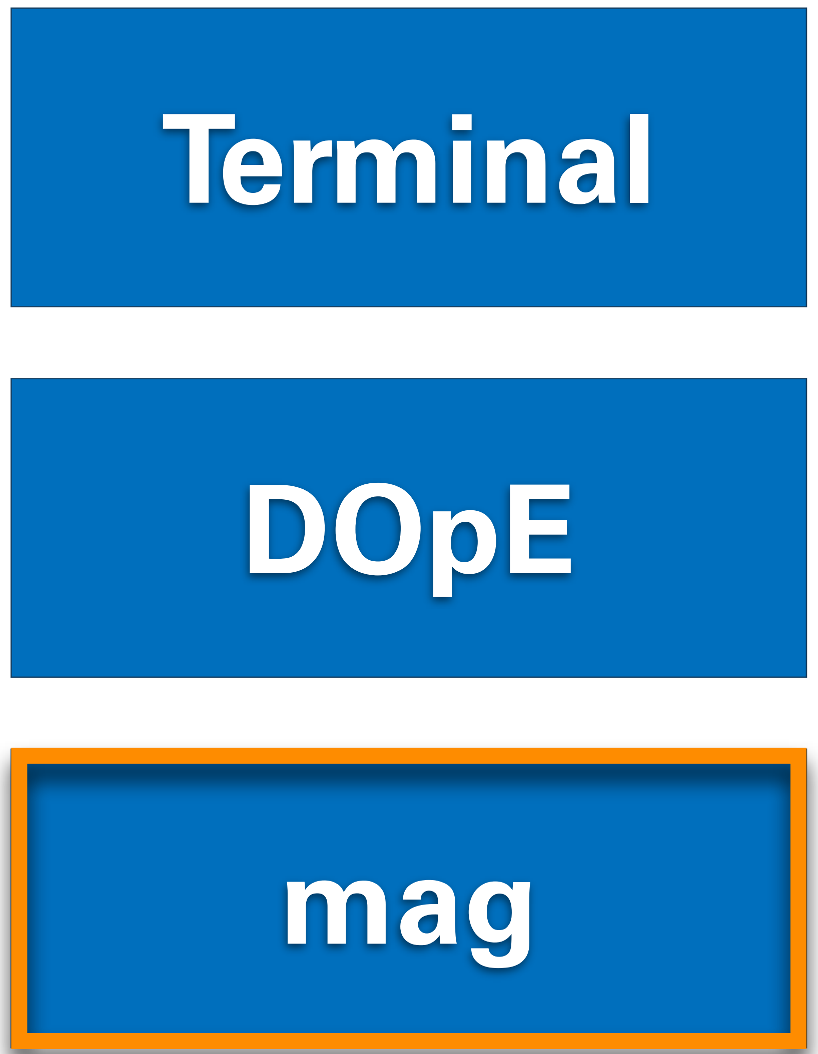
- no real one implemented
- we have a tmpfs using RAM as backing store
- VPFS: securely reuse a Linux filesystem

L4RE VFS



- hierarchical name space
- connects subtrees to different backend servers
- aka mounting

MAG



- multiplexes the frame buffer
- no virtual desktops, but window merging
- details in the legacy / security lectures

DOPE

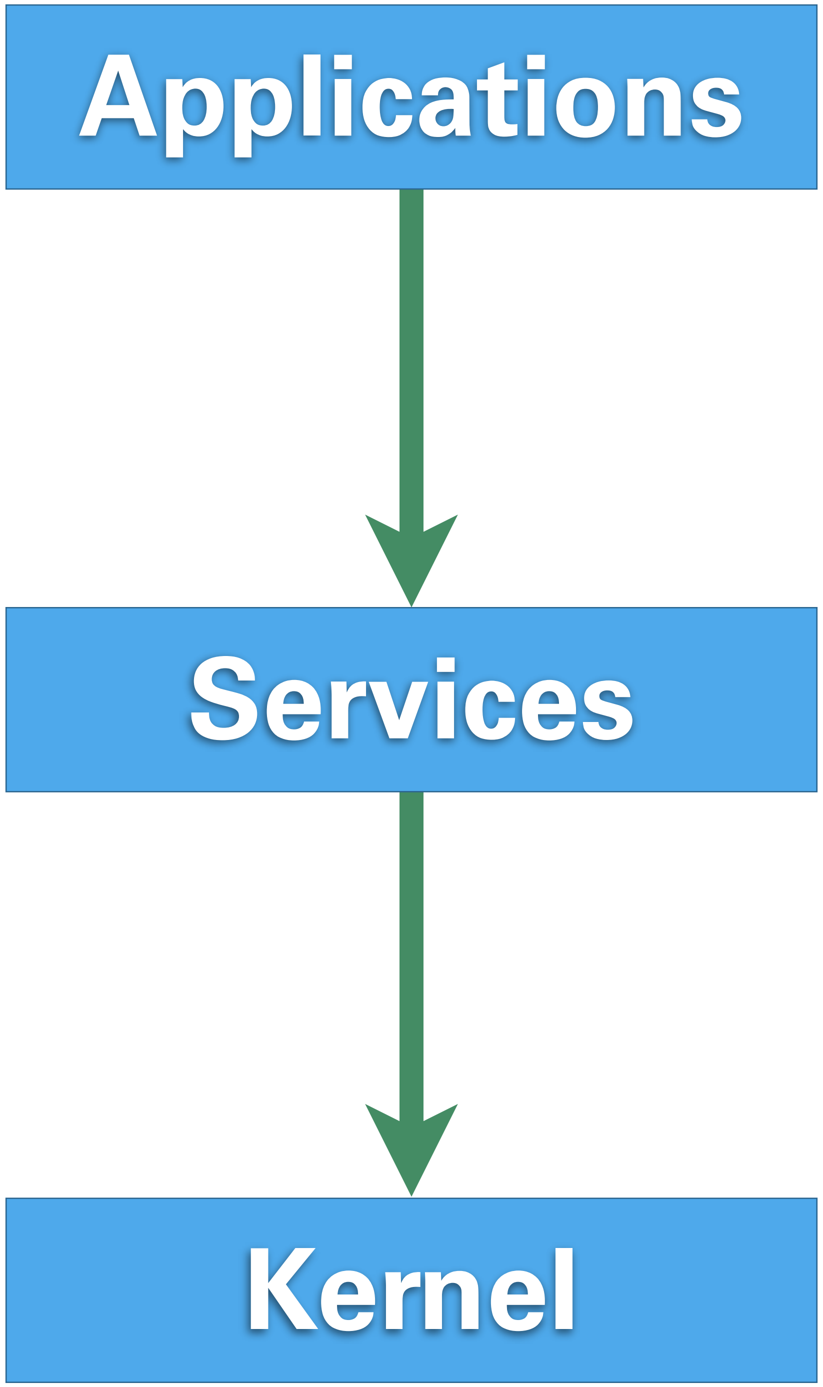
- widget drawing server
- also handles mouse and keyboard input
- distributed to applications
- can also operate on raw framebuffer
- real-time capable

TERMINAL

- DOpE client providing a terminal window
- VT100 emulation
- can support readline applications
  - shell
  - python

RESOURCE ACCESS

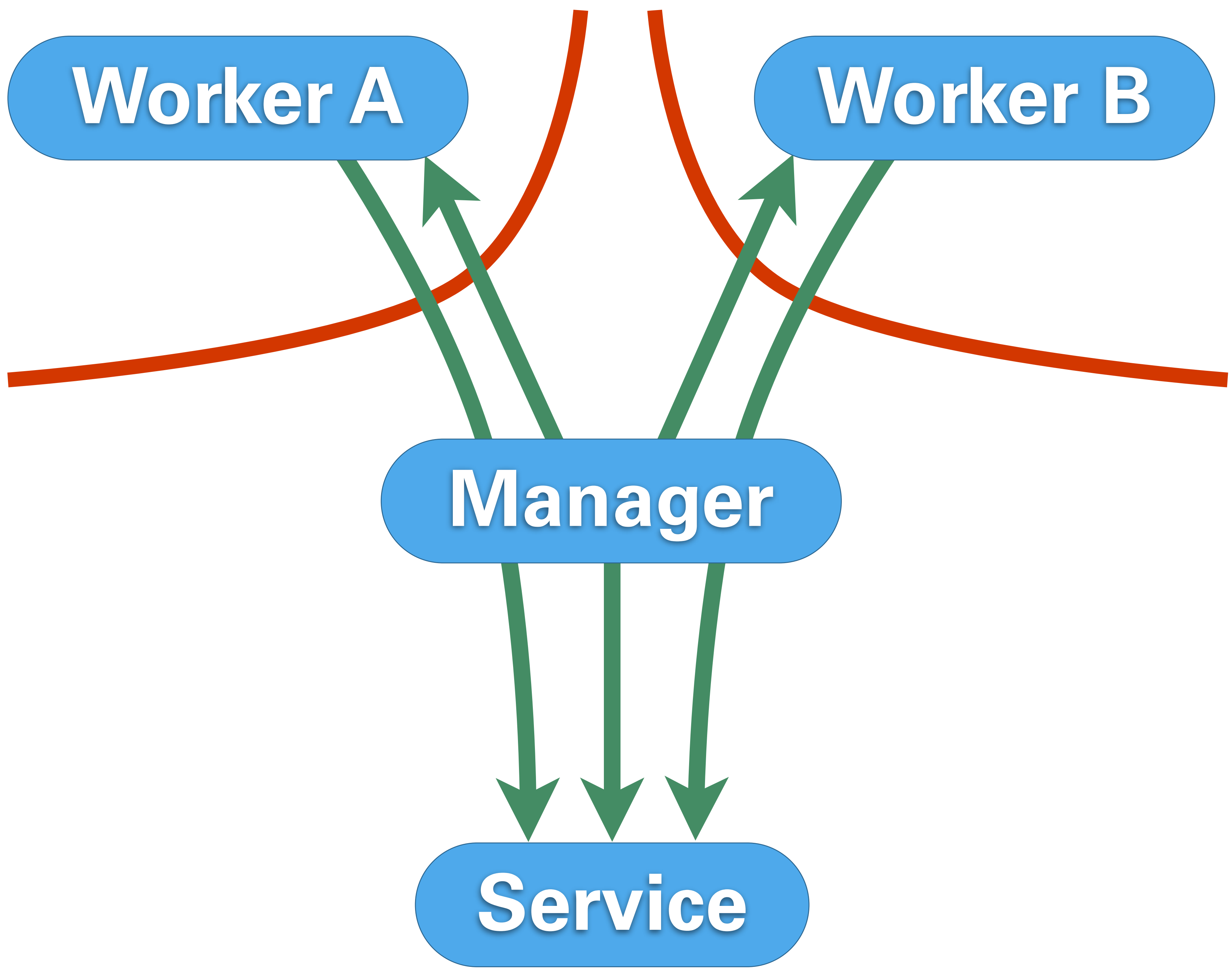
SYSTEM DESIGN



DESIGN GOALS

- application-centric interfaces
- object-based design
- easy setup and destruction of subsystems
- object invocation by message passing
- uniform security model
- all services virtualizable
- flexible and efficient support for multicore

EXAMPLE



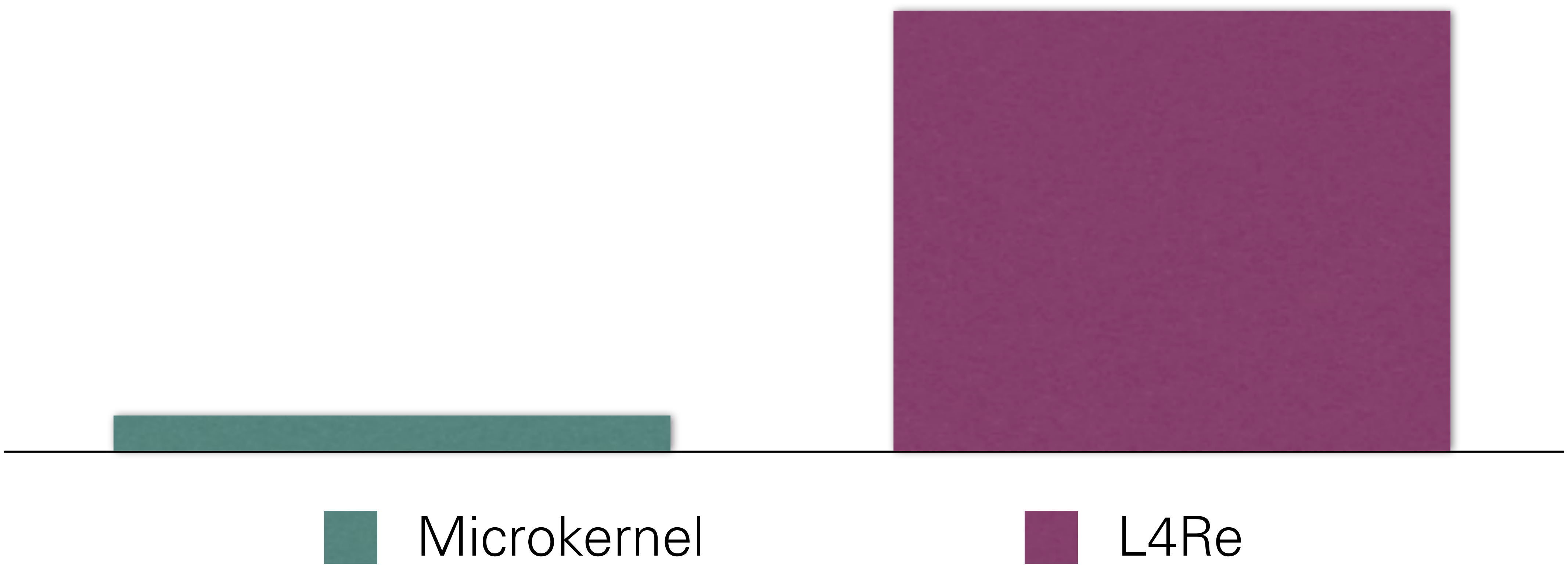
GOOGLE CHROME

- separate processes
  - chrome parent
  - sandboxes for tabs
- implementation on Linux: glorious mix of chroot(), clone() and setuid()
- there must be a better way…

TWO WORLDS

POSIX	POLA
operations allowed by default	nothing allowed by default
some limited restrictions apply	every right must be granted
ambient authority	explicit authority

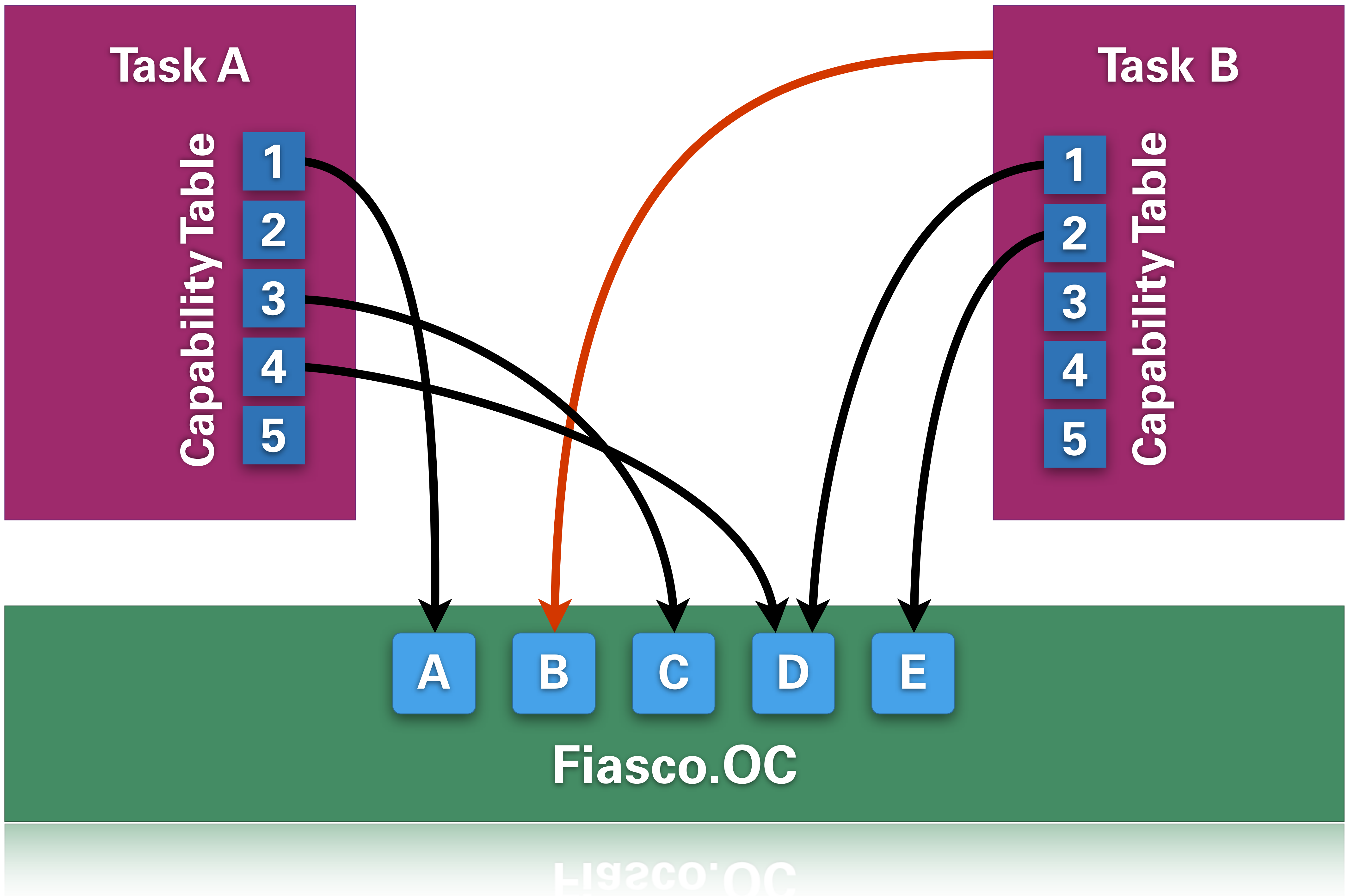
L4Re — the L4 Runtime Environment
set of libraries and system services on top of the Fiasco.OC microkernel



CAPABILITIES

- Fiasco.OC and L4Re form an **object-capability** system
- actors in the system are **objects**
  - objects have local state and behavior
- **capabilities** are references to objects
  - object interaction requires a capability
  - capabilities cannot be forged

CAPABILITIES

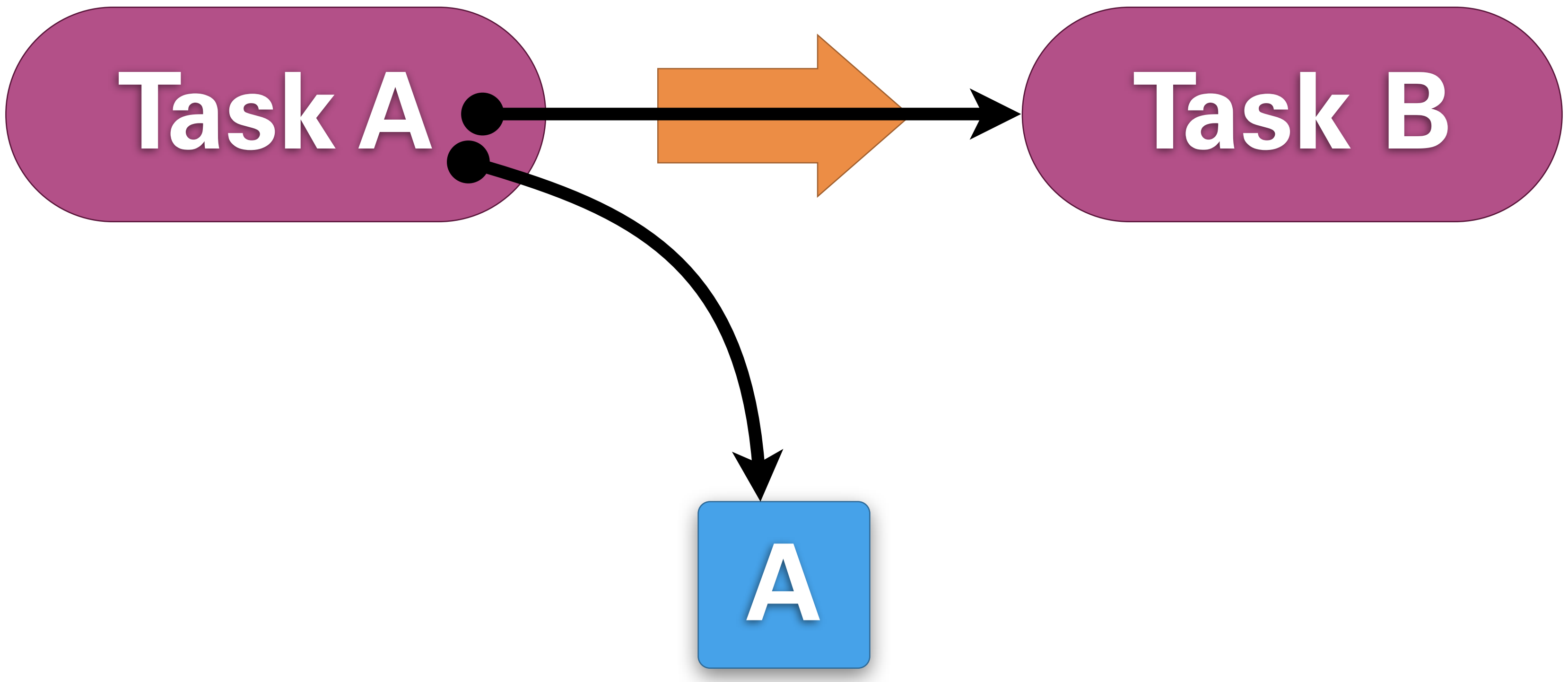


TECHNISCHE UNIVERSITÄT DRESDEN

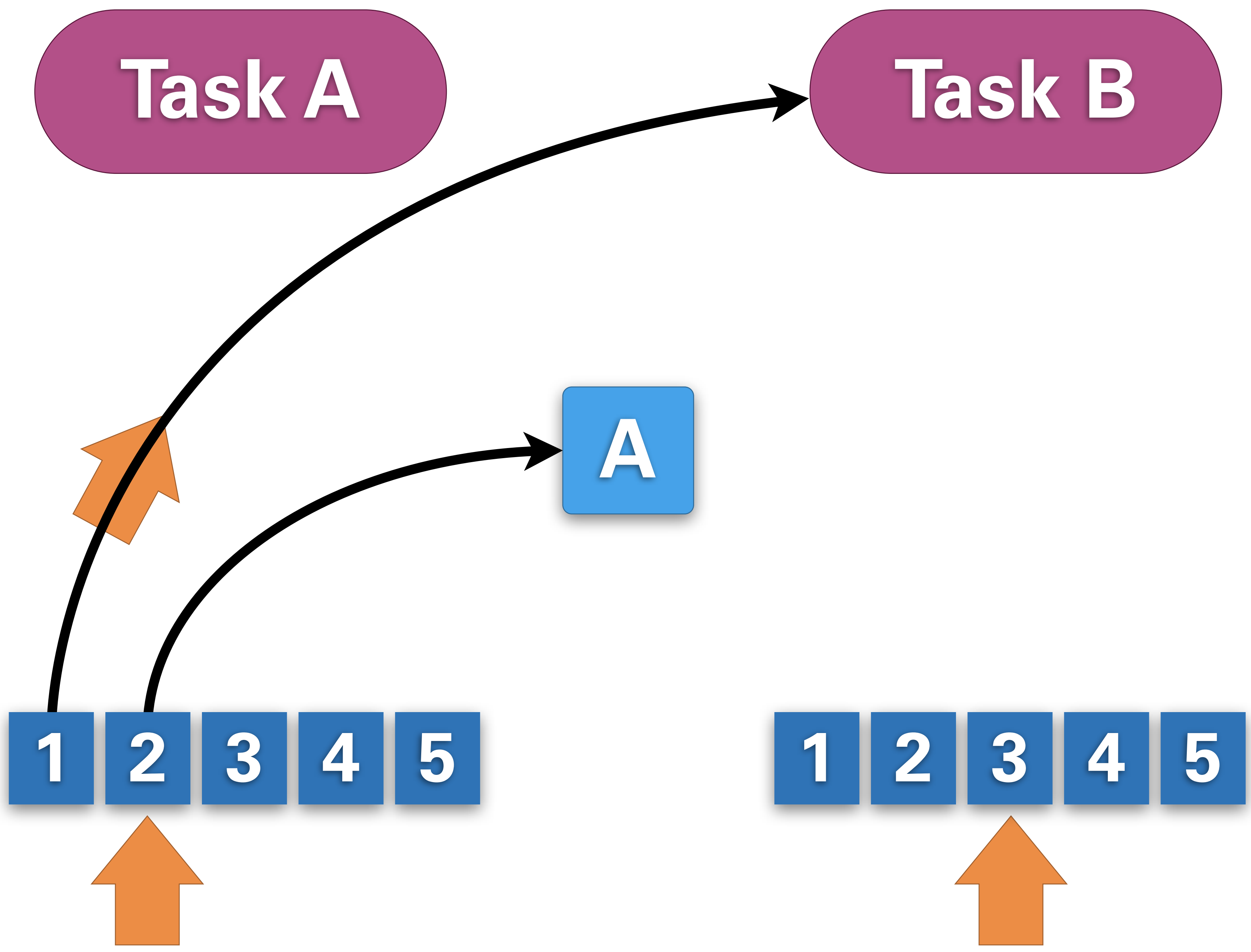
HOW TO USE?

- invocation of any object requires a capability to that object
  - no global names
- no sophisticated rights representation beyond capability ownership
  - just four rights bits on objects
- C++ language integration
- capabilities passed as message payload

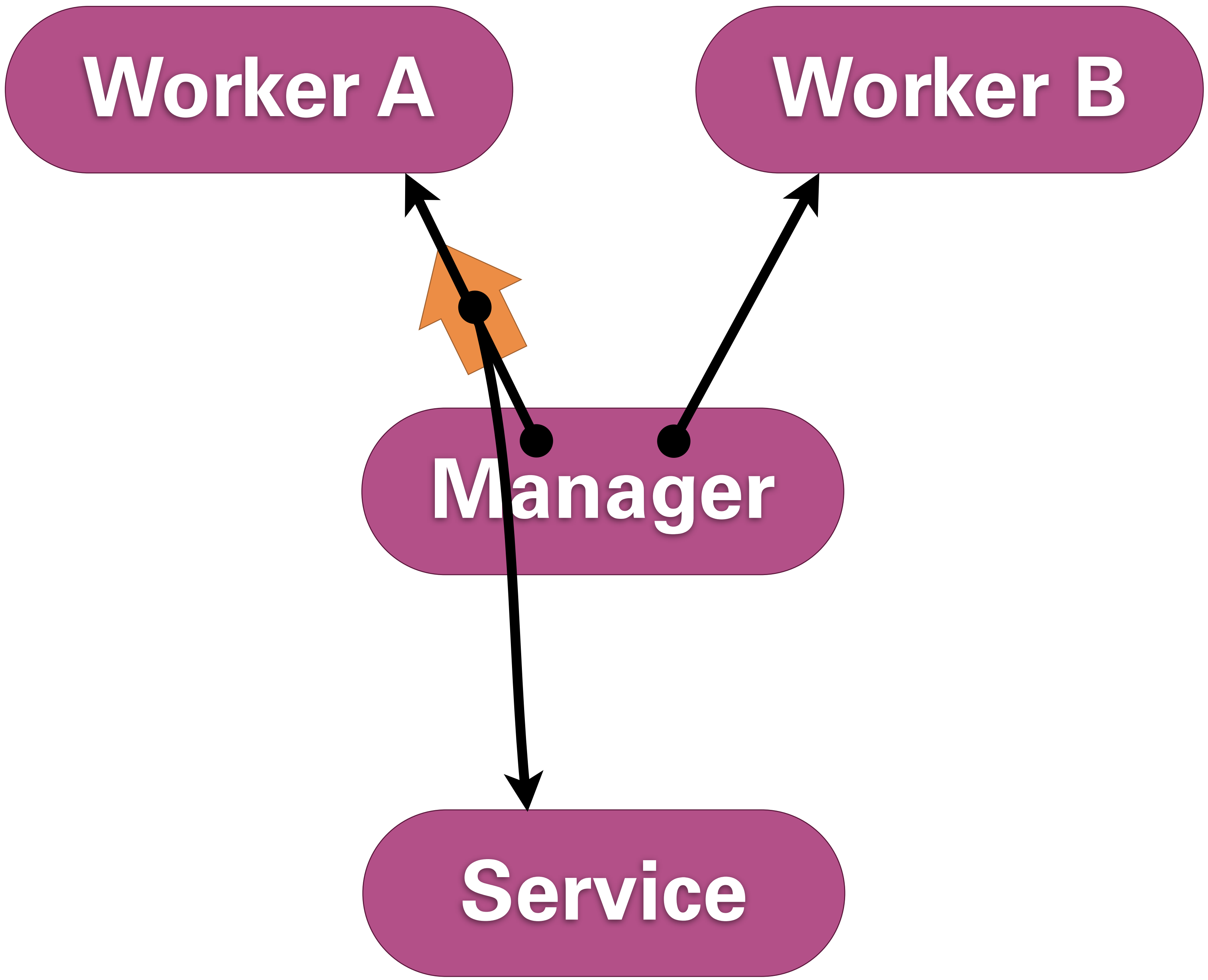
CAP TRANSFER



CAP TRANSFER



EXAMPLE

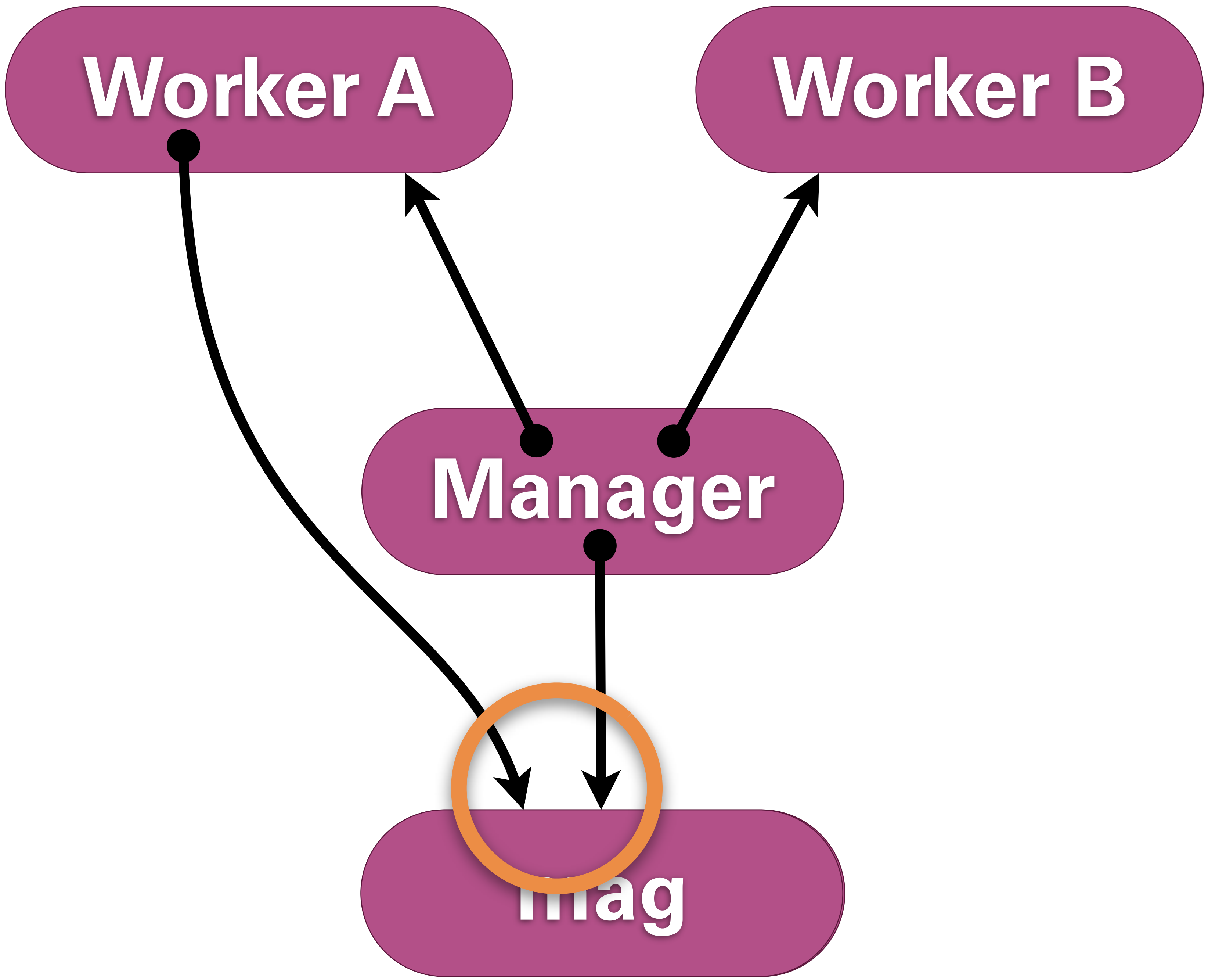


ANSWERING

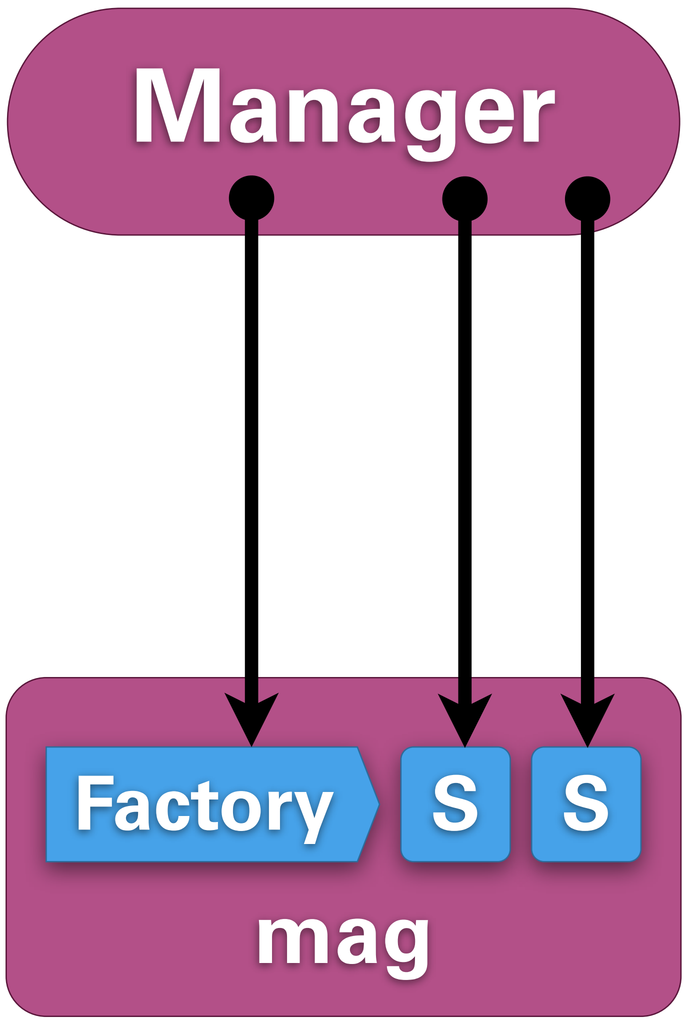
How do you send an answer to a client?

- Always include a backward capability in every request?
- Establish backward capability once and cache?
- call-return-semantics as the standard case
- **implicit reply capability**
- use-once, cannot be passed on

EXAMPLE

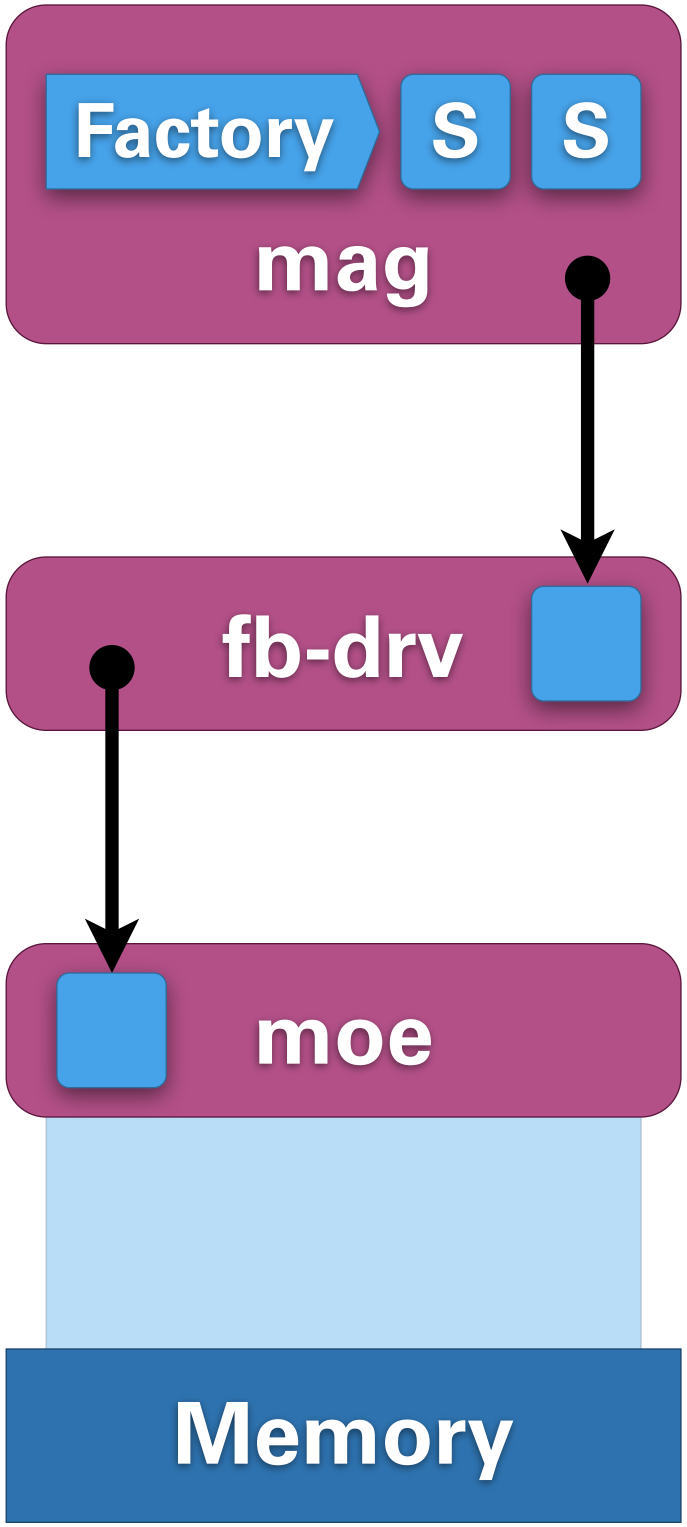


MAG



- **factory** for new framebuffer **sessions**
- session object
  - backing store memory
  - view: visible rectangle on the backing store
  - metadata, refresh method
- How does it appear on the screen?

MAG

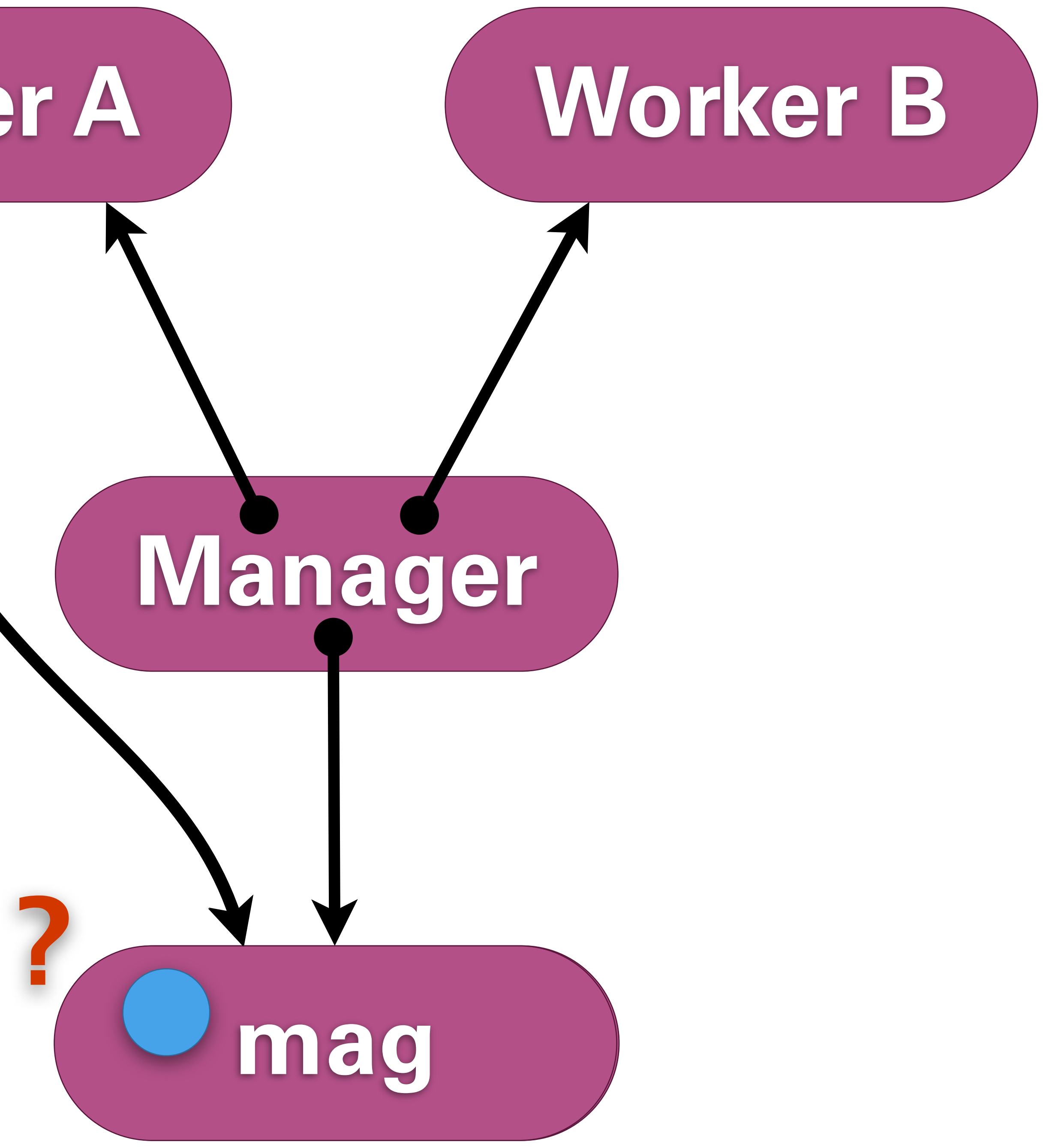


- hardware framebuffer is memory with side effect
- all memory is initially mapped to the root task
- **framebuffer driver**
  - find framebuffer memory
  - wrap in FB-interface
- same interface as mag's

INTERFACES

- L4Re uses one interface per resource
- low-level system resources are managed by the kernel
  - CPU, memory, IRQ
  - minimal policy
- user-level servers can reimplement and augment interfaces
- **virtualizable interfaces**

EXAMPLE



SUBSYSTEMS

Subsystem Life

- subsystems are opaque
- parents can restrict the resources
- parents cannot restrict their sub-structure

Subsystem Death

- How to deallocate resources in servers?
- notify all servers used by the subsystem?
- **garbage collection**

CONCLUSION

- ✔ coherent per-resource interfaces
- ✔ all services provided as objects
- ✔ garbage collection for server resources
- ✔ invocation is the only system call
- ✔ object-capability system
- ✔ all interfaces can be interposed
- ✔ RCU in kernel, user-level load balancing

SUMMARY

- kernel resource management
- basic resource management concepts
  - exokernel
  - multiserver
- management details for specific resources
- how capabilities work