Architectures, Microkernels, IPC, Capabilities

http://d3s.mff.cuni.cz/aosy

CHARLES UNIVERSITY
Faculty of Mathematics and Physics

Department of Distributed and Dependable Systems

Jakub Jermář
jakub.jermar@kernkonzept.com

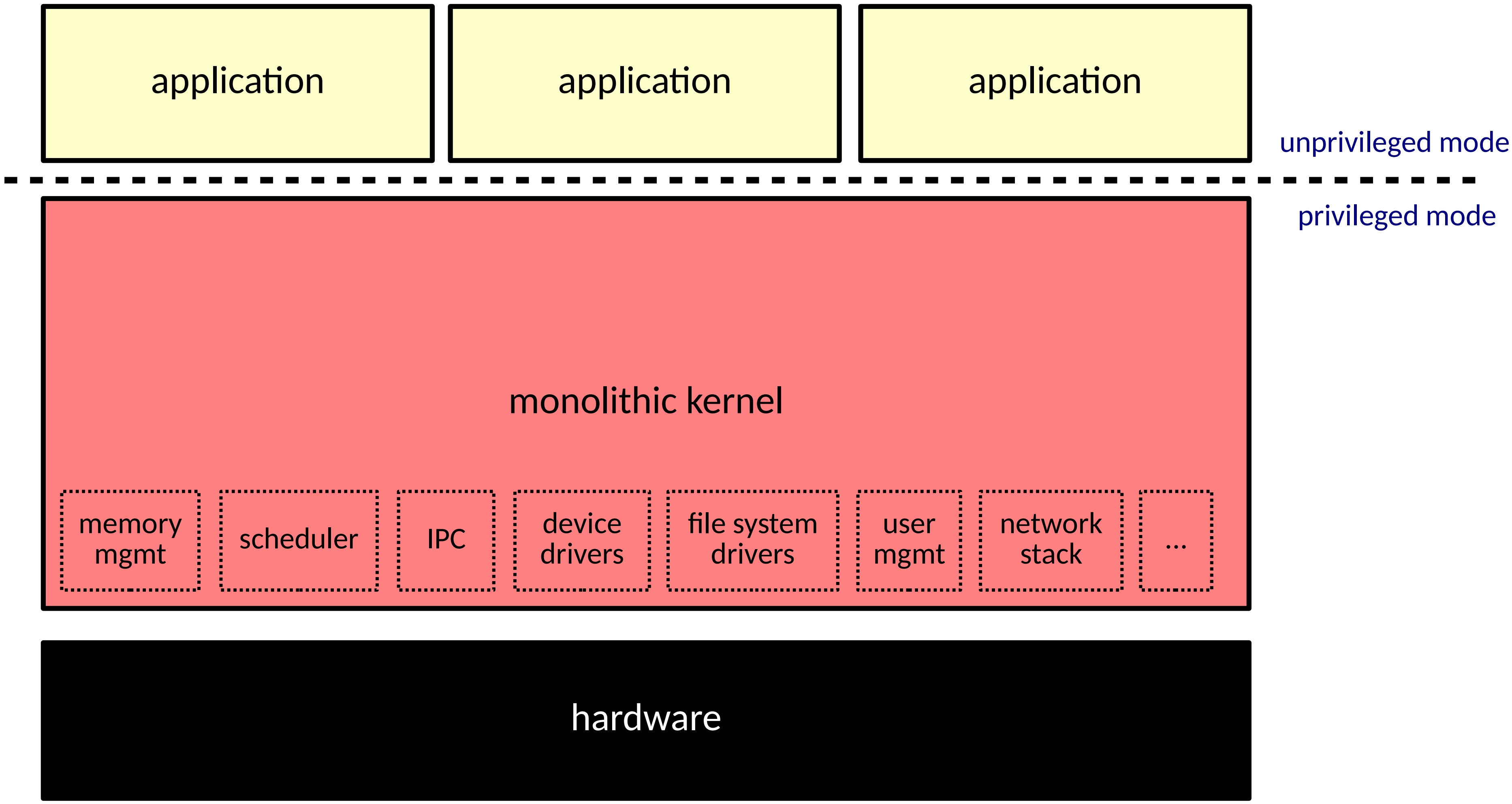
Agenda

- **Kernel architectures**
- **Microkernels**
- **IPC**
- **Capabilities**

Recall: Common OS Taxonomy

- **Special-purpose operating systems**
  - Real-time operating systems
  - Hypervisors (type 1)
  - ...
- **General-purpose operating systems**
  - Monolithic kernel
  - Single-server microkernel
  - Multiserver microkernel
  - Hybrid kernel (?)

Monolithic Kernel



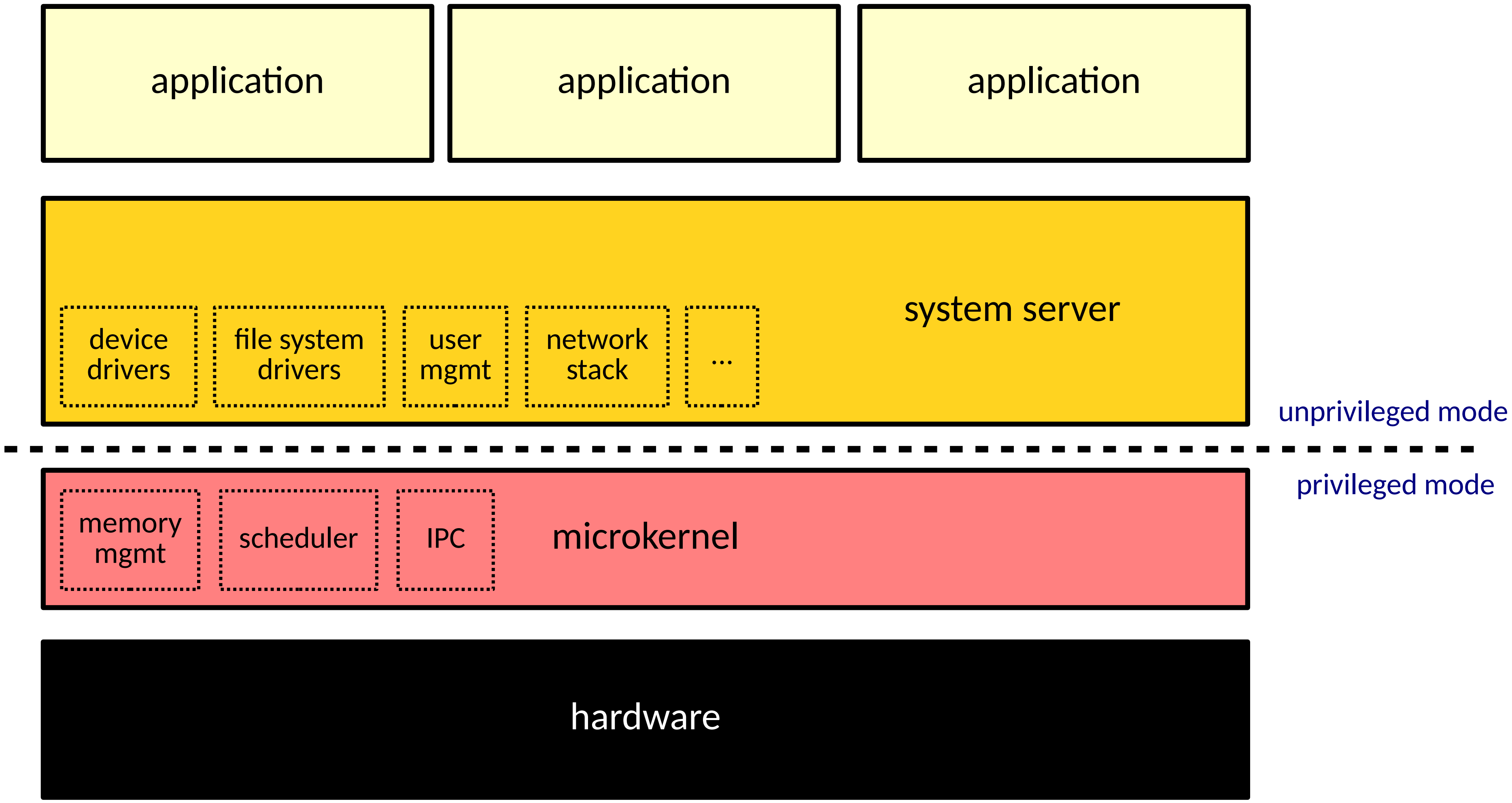
Some Obvious Issues

- **Security**
  - Applications trust all kernel components
  - Kernel components trust all other kernel components
- **Reliability**
  - Kernel components are a single point of failure
- **Availability**
  - Kernel components cannot be updated independently
- **Justifiability**
  - Who says file systems, networking, device drivers, etc. belong to the kernel?

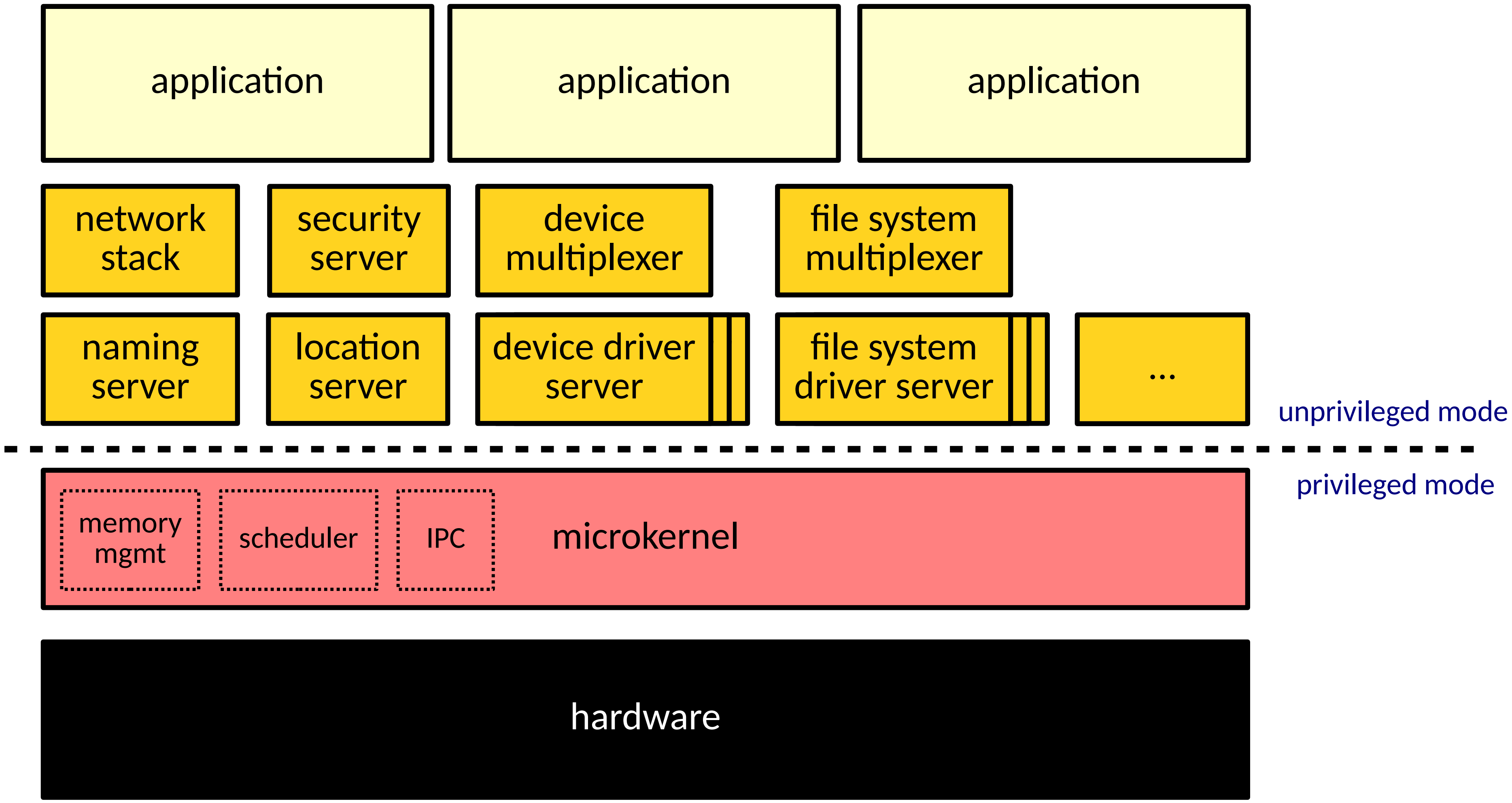
Some Obvious Issues (2)

- **Extensibility**
  - How to extend the system without modifying the kernel
  - Too many communication mechanisms
    - Unix: pipes, files, shared memory, sockets, signals, System V IPC, System V shared memory, System V semaphores...
  - Kernel has many built-in policies
- **Software design principles**
  - Interfaces between kernel components are usually implicit, not well-defined

Single-server Microkernel



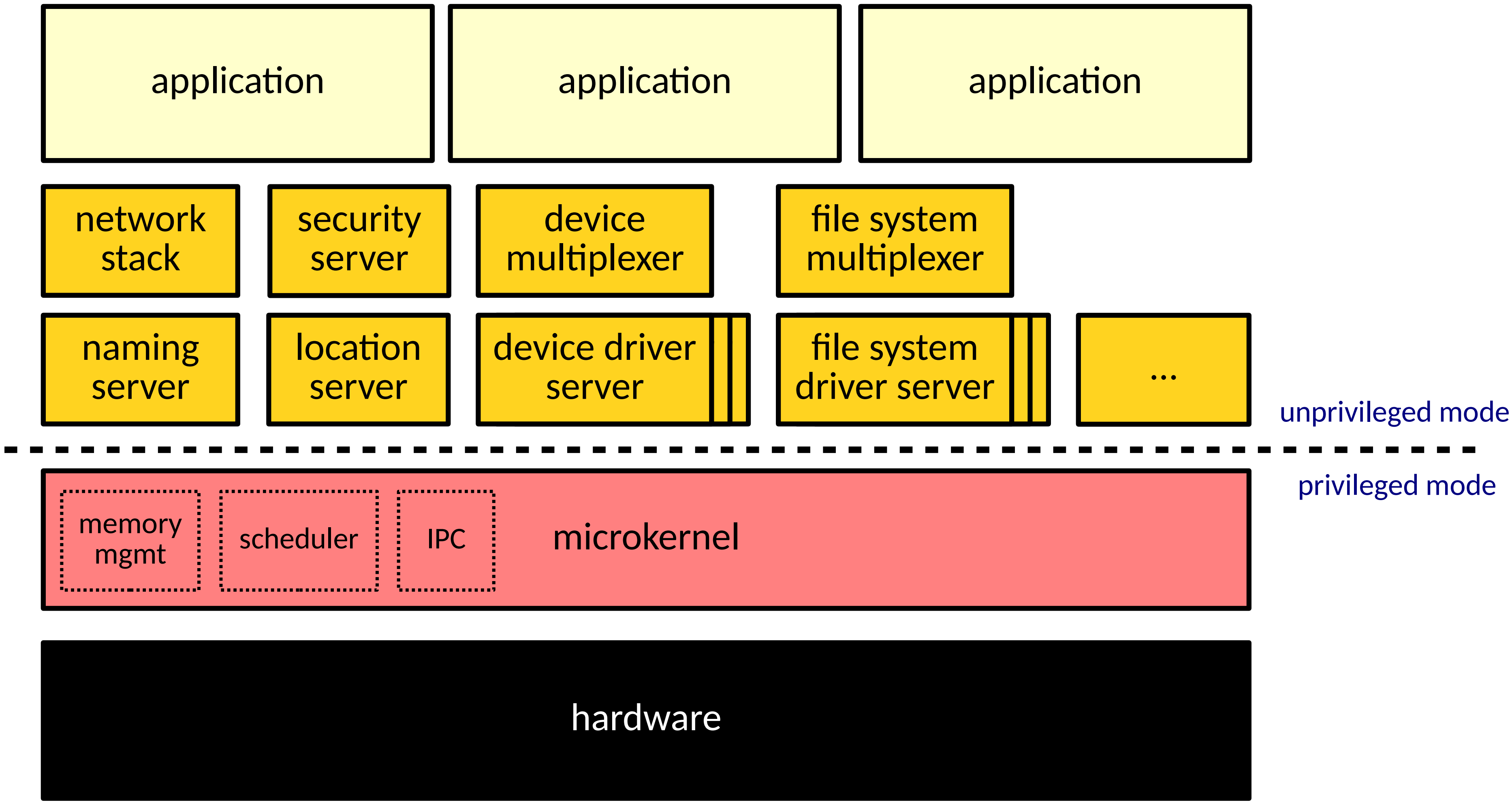
Multiserver Microkernel



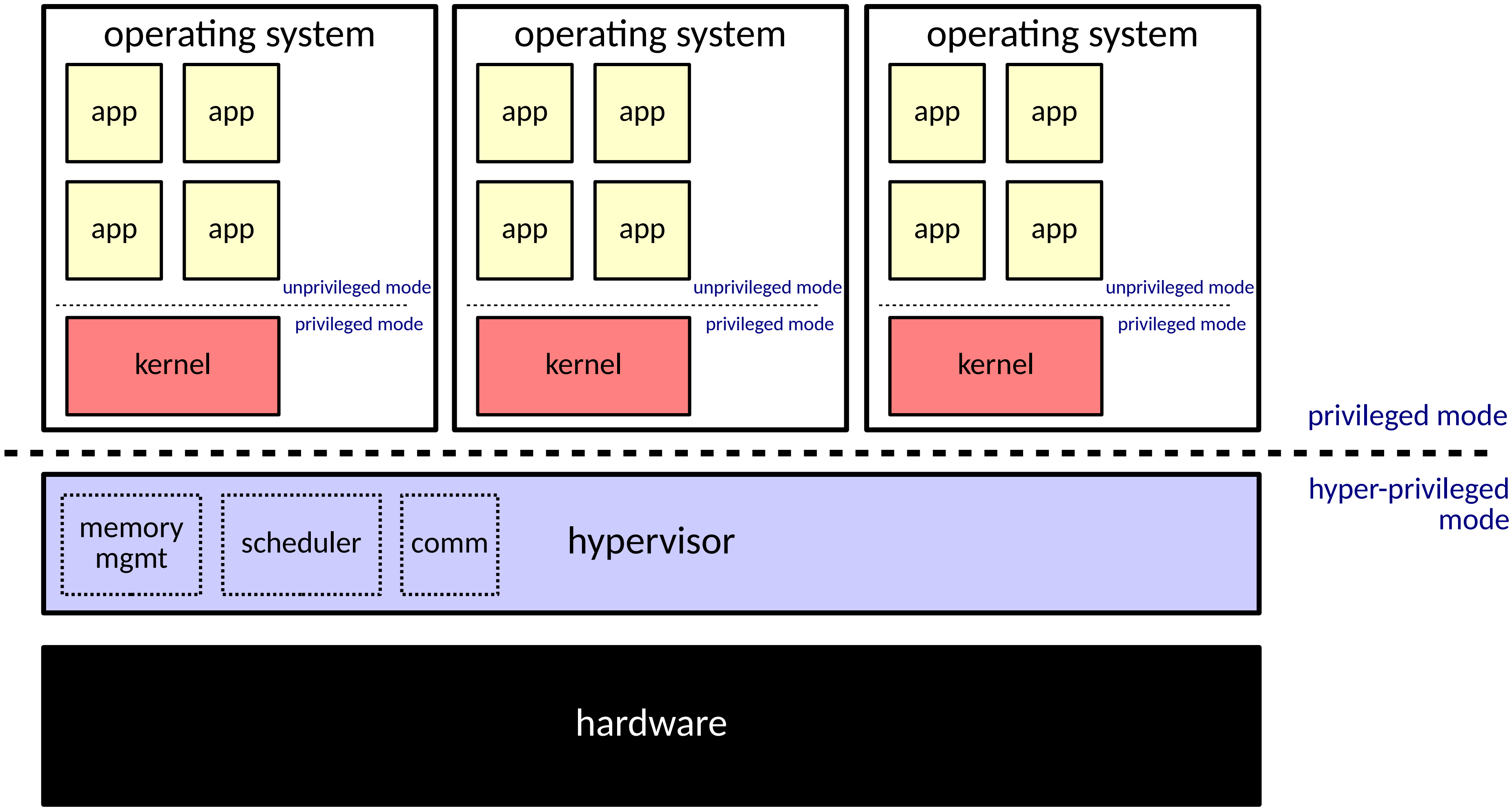
Examples

- **Monolithic kernel**
  - Linux, Solaris (UTS), Windows, FreeBSD, NetBSD, OpenBSD, OpenVMS, MS-DOS, RISC OS
- **Microkernel** (the microkernel on its own)
  - CMU Mach, GNU Mach, L4::Pistachio, Fiasco.OC, seL4
- **Single-server microkernel**
  - CMU Mach (with 4.3BSD server), MkLinux, L4Linux
- **Multiserver microkernel**
  - L4Re, HelenOS, MINIX 3, Genode, GNU/Hurd

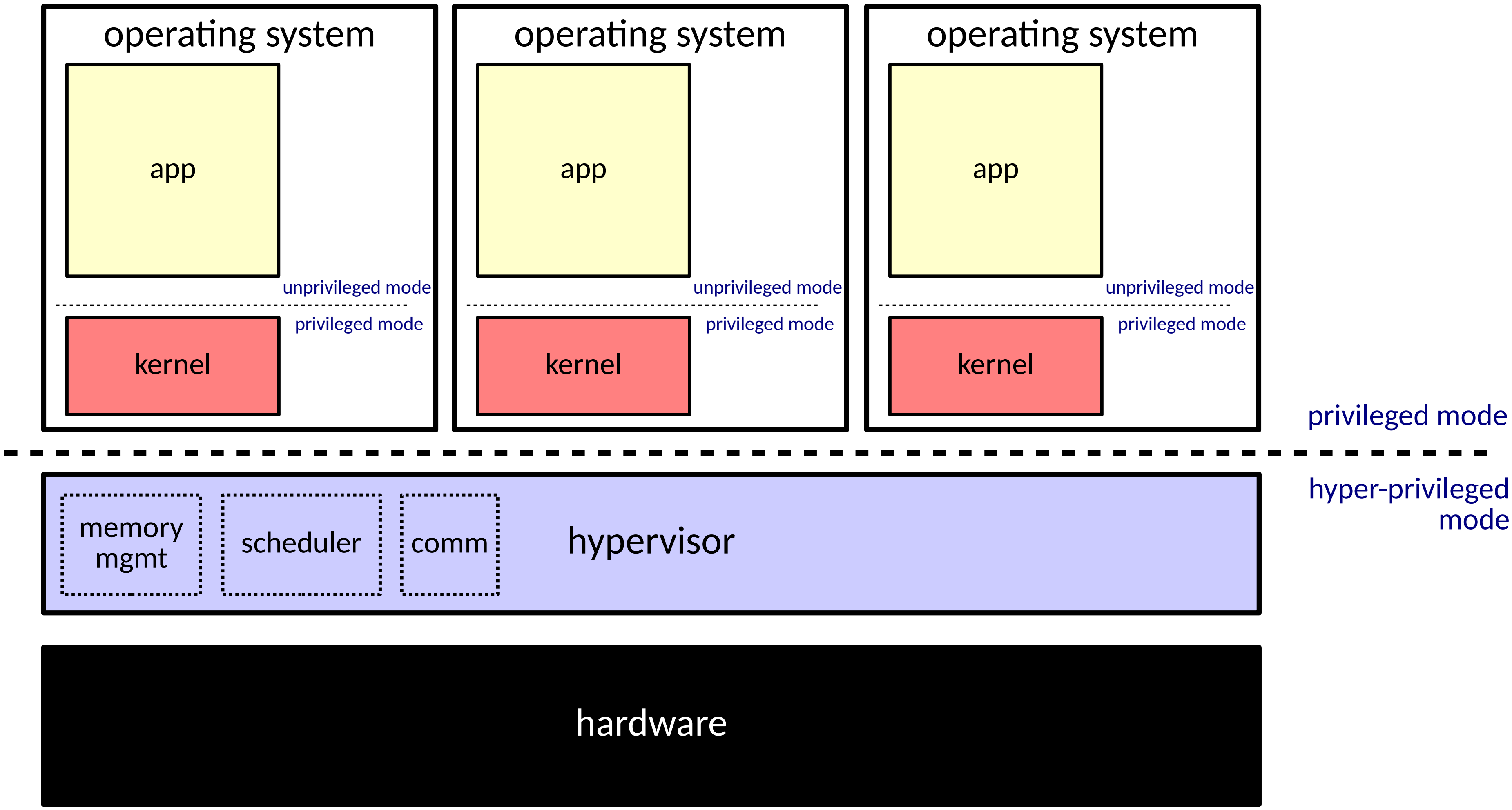
Multiserver Microkernel (reprise)



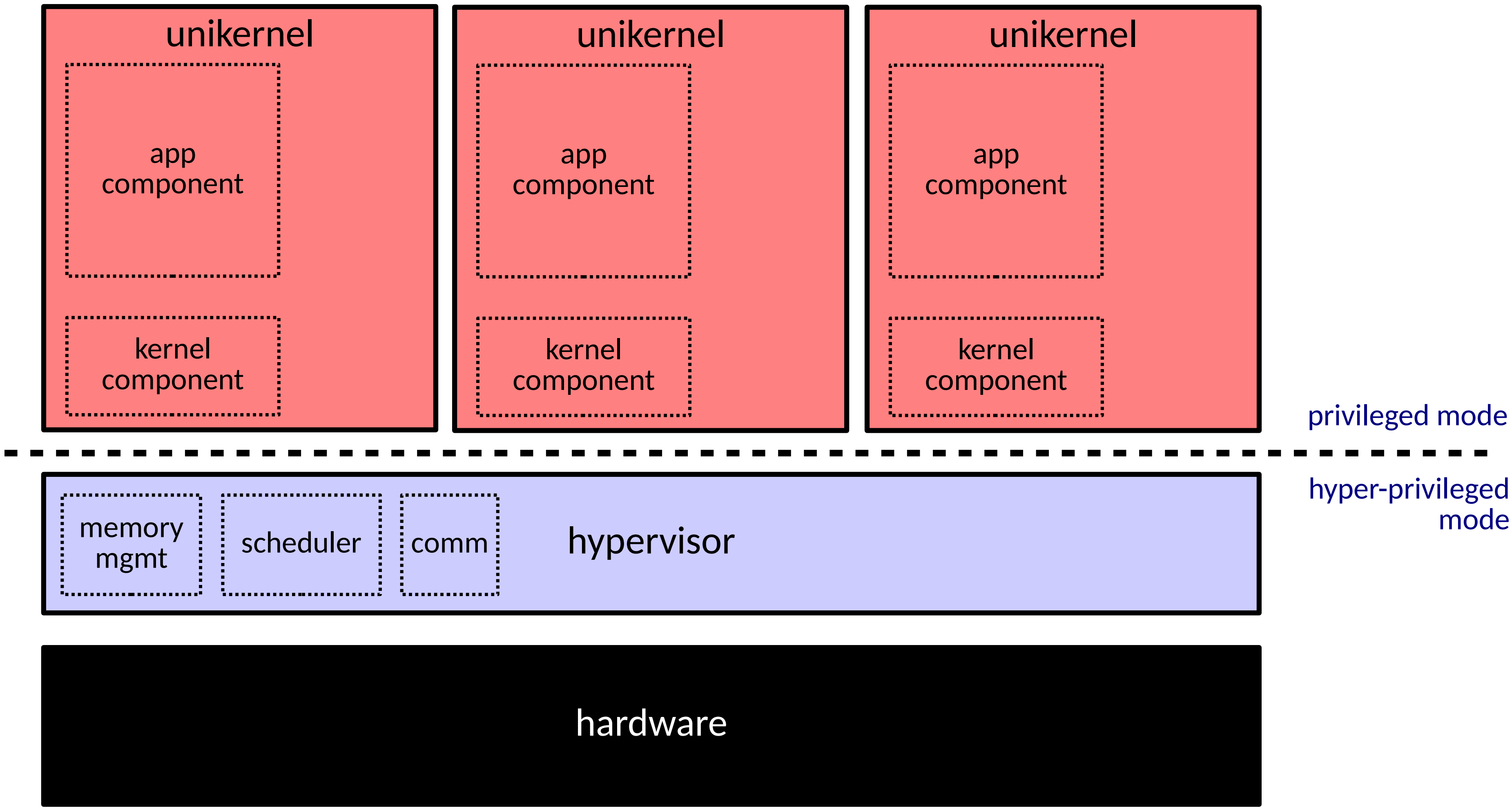
Hypervisor (Type 1)



Common Cloud Deployment



Unikernel


unikernel
app component
kernel component
unikernel
app component
kernel component
unikernel
app component
kernel component
privileged mode
hyper-privileged mode
memory mgmt
scheduler
comm
hypervisor
hardware

Unikernel (2)

- **Library operating system**
  - Approach to building operating systems
- **Unikernel**
  - Architecture
  - Binary artifact

Department of Distributed and Dependable Systems

Unikernel (3)

- **Library operating system**
  - Payload (application) merged with the kernel
    - Kernel component acts as a library providing access to the hardware, threading, file systems, etc.
      - Only necessary functionality
    - Mostly static (single image), but there are dynamic variants
    - Code runs in privileged (*less privileged*) mode and single address space
      - No mode switches, address space switches
      - Syscalls can be replaced by function calls
      - Isolation/security provided by the underlying hypervisor (*more privileged mode*)

Unikernel (4)

- Madhavapeddy, A., Scott, D., J.: **Unikernels: Rise of the Virtual Library Operating System**, ACM Queue, 2013
  - **MirageOS**
    - University of Cambridge, Docker
    - Clean-slate components written in OCaml
    - Used in Docker for Mac, VPNKit

Department of Distributed and Dependable Systems

Unikernel (5)

- Porter, D., E., et al.: **Rethinking the library OS from the top down**, ASPLOS, 2011
  - **Drawbridge**
    - Microsoft Research (2011– ?)
    - Librarified Windows
    - Used in MSSQL Server for Linux (2016)
- Kantee, A.: **The Rise and Fall of the Operating System**, ;login:, October 2015, Vol. 40, No. 5
  - **Rumpkernel**
    - Librarified NetBSD
    - Popular source of components for *any* kernels (NetBSD, rumprun, Hurd, Genode, ...)

Future Hardware Predictions

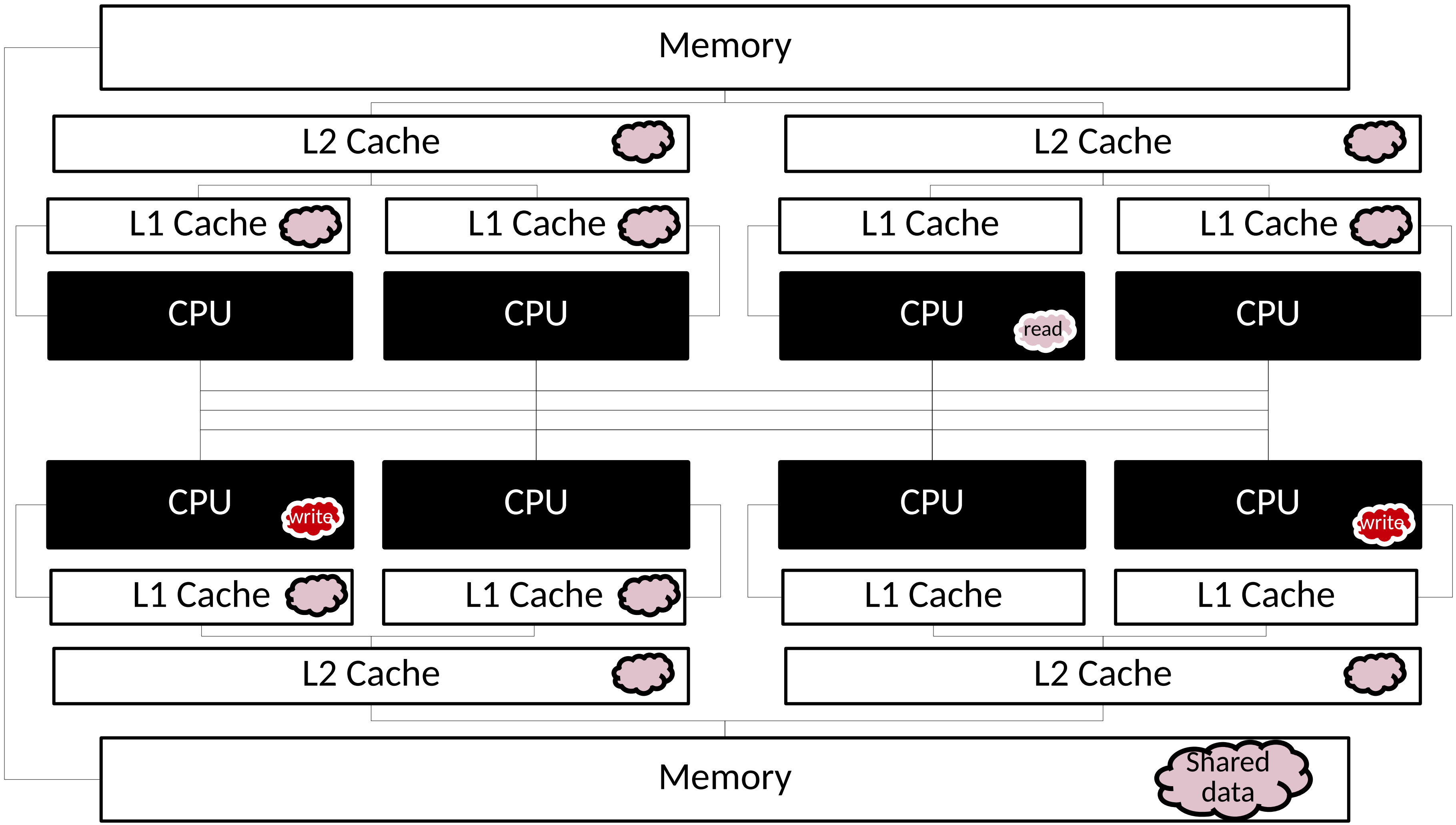
- **More of**
  - Complex interconnects & cache hierarchies
    - Cache-coherency protocols even more expensive
  - Diversity
    - Different cores together → same optimizations won't work anymore
  - Heterogeneity
    - Multiple ISAs → can't have a single-image OS
- **Less of / lack of**
  - Cache coherency
  - Shared memory

Department of Distributed and Dependable Systems

Options for general purpose OS’s

- **Resign**
  - Make it easy to build specialized OS’s
    - Unikernels
- **Redesign**
  - Attack the problem from different angle
    - Multikernels

Implicit Message Passing in Hardware

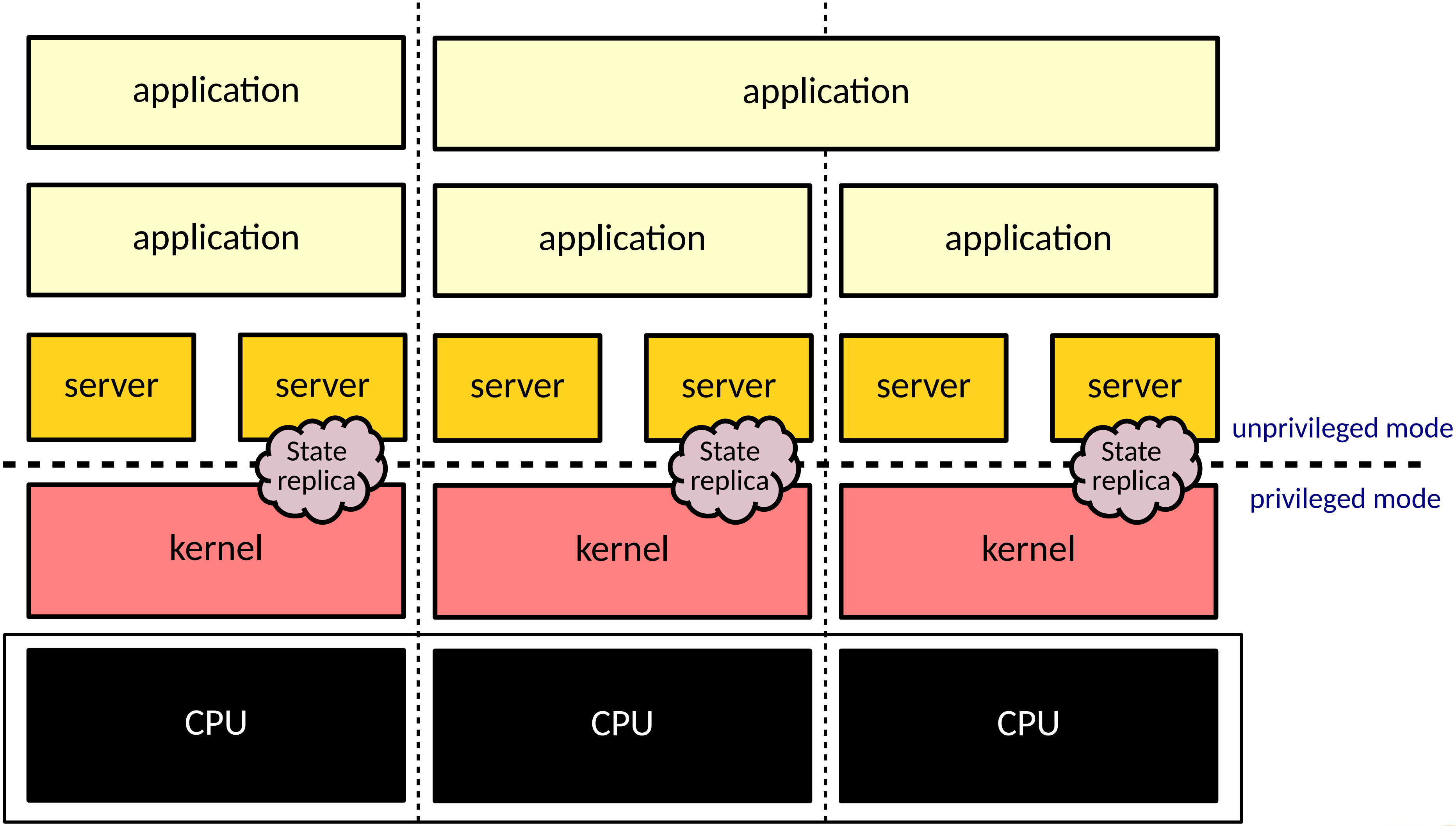


Multikernel Paradigm Shift

- **Inside the OS layer**
  - Do not assume coherent shared-memory SMP
    - If available, use to optimize message passing
  - No implicit inter-core state sharing
    - Simple, single-threaded, event-driven code
  - Explicit inter-core communication via message passing
  - Global state replica maintained by distributed algorithms

Department of Distributed and Dependable Systems

Multikernel



Multikernel (2)

- Kernel-userspace boundary not characteristic of multikernels
- Baumann, A., et al.: **The Multikernel: A new OS architecture for scalable multicore systems**, SOSP '09
  - **Barrelfish**
    - ETH Zürich, Microsoft Research

Inter-Process Communication

- **Sharing data between processes (tasks)**
  - Crossing the process isolation in a **managed** and **predictable** way
    - Technically, any means of sharing data can be considered IPC (e.g. files, networking, middleware)
      - In monolithic systems, this usually works without using a dedicated IPC mechanism
    - Crucial for microkernel systems
      - In microkernel systems, even files and networking cannot be implemented without an IPC mechanism

Department of Distributed and Dependable Systems

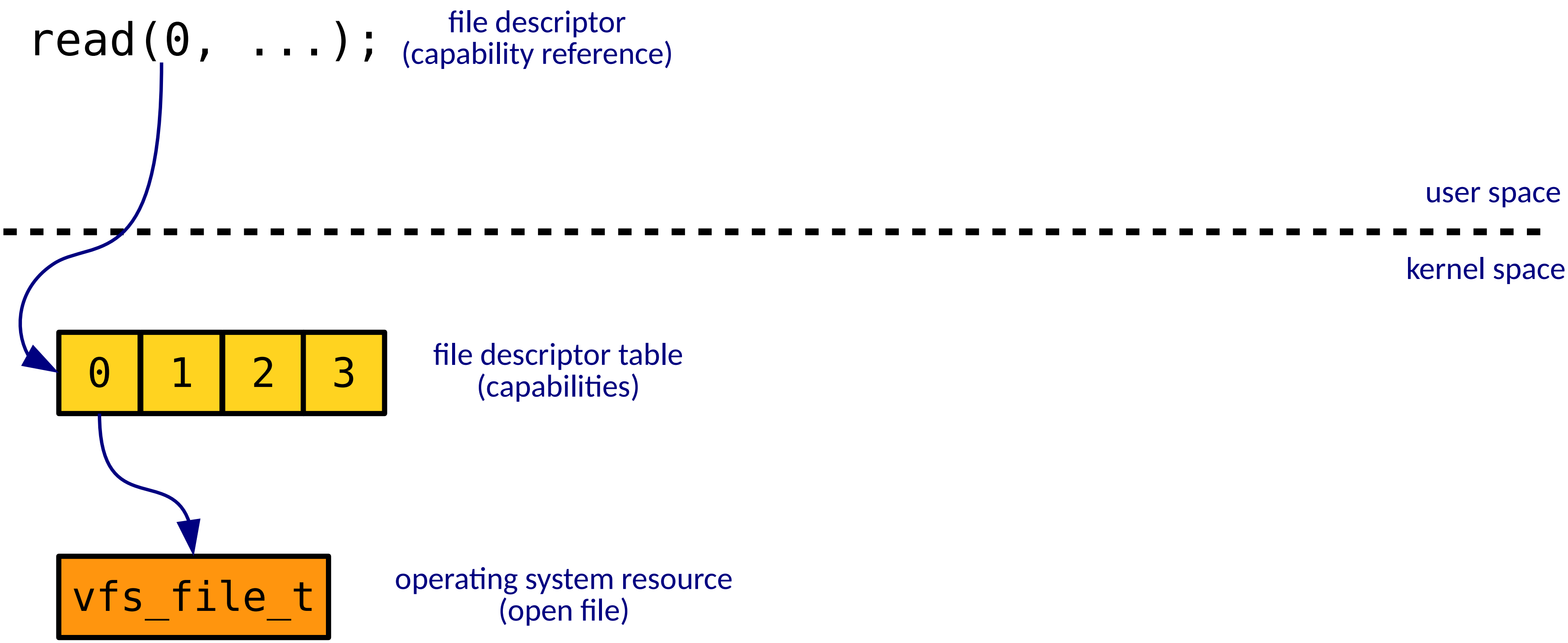
Classical IPC

- POSIX signals
- Anonymous pipes
- Named pipes
- Sockets
- POSIX shared memory
- System V shared memory, IPC, semaphores

Capabilities

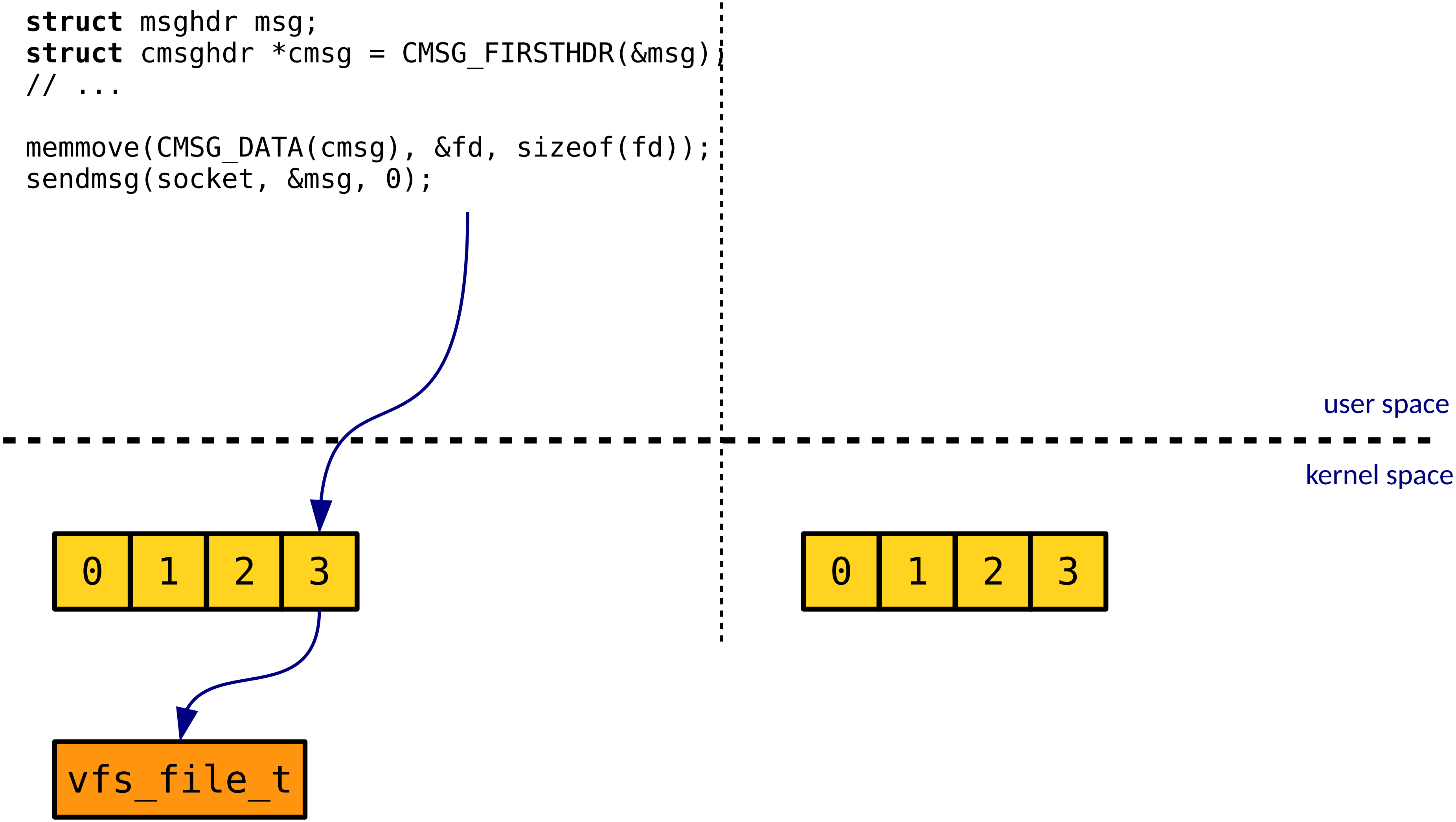
- **Capability**
  - Object identifying an OS resource
    - Logical objects (open files, connections), typed memory areas (physical memory regions)
  - Capability reference
    - Local user space identification of a capability (file handles, virtual memory regions)
  - Operations with capabilities
    - Invoking a method with a capability reference
      - Permissible methods defined by the capability itself
    - Give a capability to someone else
    - Revoke a previously given capability

Trivial Capability Example



Trivial Capability Example (2)

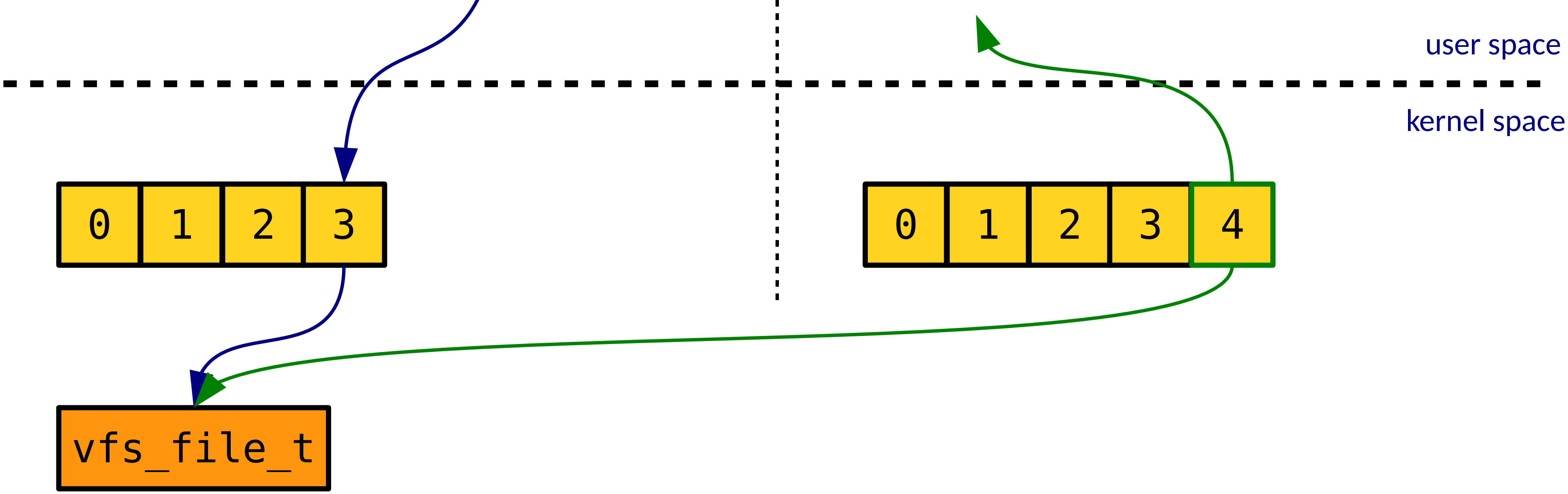
```
struct msghdr msg;
struct cmsghdr *cmsg = CMSG_FIRSTHDR(&msg);
// ...

memmove(CMSG_DATA(cmsg), &fd, sizeof(fd));
sendmsg(socket, &msg, 0);
```



Trivial Capability Example (2)

```c
struct msghdr msg;
struct cmsghdr *cmsg = CMSG_FIRSTHDR(&msg);
// ...

memmove(CMSG_DATA(cmsg), &fd, sizeof(fd));
sendmsg(socket, &msg, 0);
```

```c
struct msghdr msg;
struct cmsghdr *cmsg = CMSG_FIRSTHDR(&msg);
// ...

recvmsg(socket, &msg, 0);

int fd;
memmove(&fd, CMSG_DATA(cmsg), sizeof(fd));
```



L4 IPC Before Capabilities

- **L4::Pistachio**

```
L4_Msg_t msg;
L4_MsgClear(&msg);
L4_Set_MsgLabel(&msg, LABEL);  // set user-defined label and flags
L4_Msg_AppendWord(&msg, 1);    // append some data
L4_Msg_AppendWord(&msg, 2);    // append some data
L4_MsgLoad(&msg);              // load into message registers

L4_ThreadId_t dest_tid;
L4_MsgTag_t tag;
...
tag = L4_Send(dest_id);        // send the loaded message to dest_id
```

global ID

Issues with Global IDs

- **Prevent unauthorized clients**
  - Global ID can be guessed, even if officially unknown
  - Example: MINIX 3 communication control
    - Ordinary user processes allowed to communicate only with POSIX servers
    - Services and driver use policy configured in a file
  - Example: L4 v2 Chiefs and Clans
    - Threads can communicate with all threads in their own clan
    - Inter-clan communication must go through the chief threads
- **Permission checks**
  - Failed checks can still DoS the server
  - Decide who can do what
- **Difficult to interpose**
  - The global ID identifies the communication parties

Department of Distributed and Dependable Systems

Capabilities Trump Global IDs

- **Prevent unauthorized clients**
  - Only authorized clients have the capability
- **Permission checks**
  - Possession of the capability is the authorization to access the resource
  - Can have different capabilities for different access modes to the same resource
- **Easy to interpose**
  - All names are local
  - Communicating parties don't know each other

L4 IPC with capabilities

- **Fiasco.OC**

```
l4_msg_regs_t *mr = l4_utcb_mr();
mr->mr[0] = 1;
mr->mr[1] = 2;

l4_cap_idx_t dest_cap;          // destination object
l4_msgtag_t tag;
…
tag = l4_ipc_send(dest_cap, l4_utcb(), l4_msgtag(LABEL, 2, 0, 0),
                                       L4_IPC_NEVER);
```

local ID

Department of Distributed and Dependable Systems

Fiasco.OC IPC

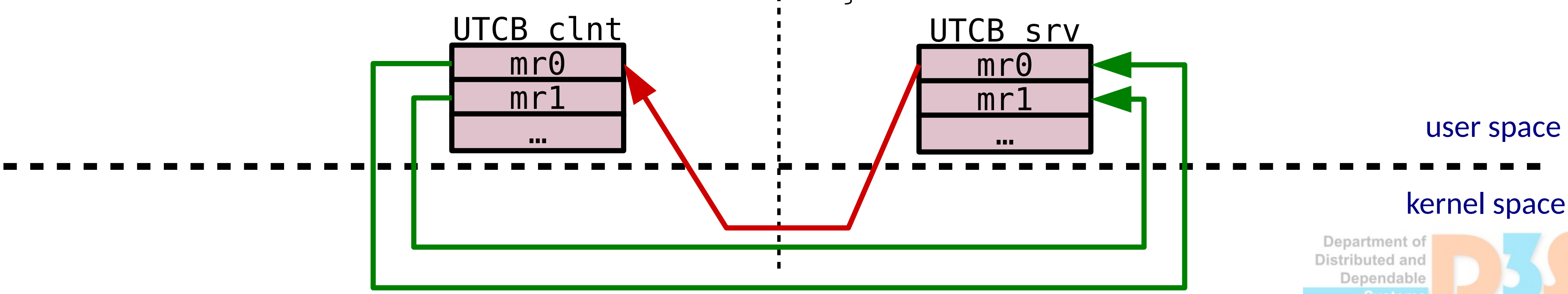
- l4_msgtag_t l4_ipc(l4_cap_idx_t dest, l4_utcb_t *utcb, l4_umword_t flags, l4_umword_t slabel, l4_msgtag_t tag, l4_umword_t *rlabel, l4_timeout_t timeout);
- SEND – Send to the specified destination
- RECV – Receive from the specified destination
- CALL (SEND | RECV) – Send, create reply capability and receive
- WAIT (OPEN_WAIT | RECV) – Receive from any possible sender
- SEND_AND_WAIT (SEND | OPEN_WAIT | RECV)
- REPLY | SEND – Send to the reply capability
- REPLY | SEND | RECV – Send to the reply capability and receive
- REPLY_AND_WAIT (REPLY | SEND | OPEN_WAIT | RECV)

Department of Distributed and Dependable Systems

Fiasco.OC Client/Server IPC Example

```
l4_msg_regs_t *mr = l4_utcb_mr();
int a = 1;
int b = 1;

for (;;) {
  mr->mr[0] = a;
  mr->mr[1] = b;

  l4_msgtag_t tag;
  tag = l4_ipc_call(server_cap,
    l4_utcb(), l4_msgtag(0, 2, 0, 0),
    L4_IPC_NEVER);
  …
  a = b;
  b = (int)mr->mr[0];
}
```

```
l4_msgtag_t tag;
l4_umword_t label;
l4_msg_regs_t *mr = l4_utcb_mr();

tag = l4_ipc_wait(l4_utcb(), &label,
  L4_IPC_NEVER);
for (;;) {
  …
  int a = mr->mr[0];
  int b = mr->mr[1];
  mr->mr[0] = (int)(a + b);

  tag = l4_ipc_reply_and_wait(l4_utcb(),
    l4_msgtag(0, 1, 0, 0), &label,
    L4_IPC_NEVER);
}
```



Fiasco.OC IPC (2)

- **l4_msgtag(label, words, items, flags)**
  - Label
    - User-defined label, e.g. protocol number, error code
  - Words
    - Number of untyped words stored in the UTCB
  - Items
    - Number of typed items stored in the UTCB
      - Capabilities, mappings
  - Flags

Fiasco.OC IPC (3)

- **l4_umword_t slabel, *rlabel**
  - Send label
    - User-defined label copied to the recipient
    - Used to hold sender thread ID before capabilities
    - Mostly zero these days
  - Receive label
    - User-defined label copied from the sender
    - Usually zero
    - <u>Bound</u> IPC Gates and <u>attached</u> IRQ objects modify the label
      - Can be used e.g. to store a pointer to the server object

IPC Marshalling

- **By hand**
- **Interface Definition Language**
  - IDL compiler generates client and server stubs from the interface description in IDL
  - Overkill for microkernels
    - Need just one language, one architecture
    - Advanced constructs not used in microkernels
    - IDL compiler often bigger than the microkernel

Department of Distributed and Dependable Systems

IPC Marshalling

- ## Stream-based IPC

```
template <typename T>
Ipc_client &operator << (T value);

Ipc_client client(foo, &snd_buf, &rcv_buf);
int result;
client << OPCODE_BAR << 1 << IPC_CALL >> result;
```

- ## C++11 IDL (parameter packs, ...)

```
struct Foo : … {
  L4_INLINE_RPC(long, bar, (int, int &));
};

L4::Cap<Foo> foo;
int result;
foo->bar(1, &result);
```

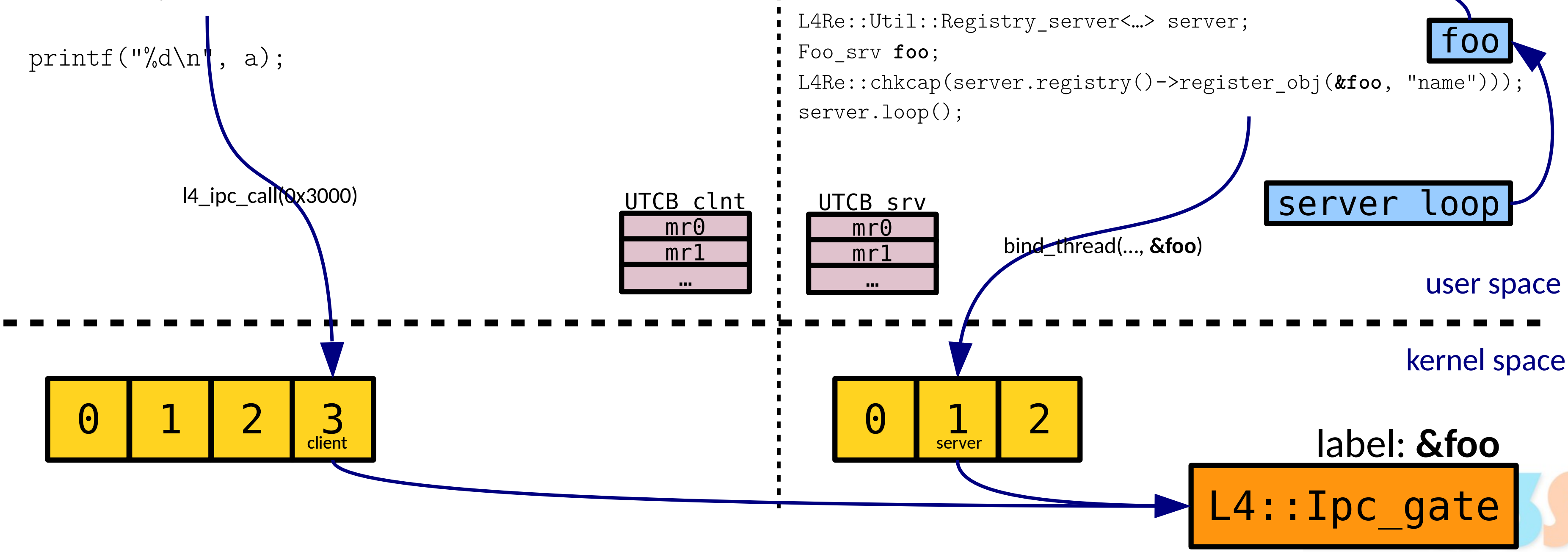
Department of Distributed and Dependable Systems

L4Re Client/Server RPC Example

```
struct Foo : L4::Kobject_t<Foo, L4::Kobject, 0xf00>
{
  L4_INLINE_RPC(long, bar, (int, int &));
  typedef L4::Typeid::Rpcs<bar_t> Rpcs;
};
```

```
L4::Cap<Foo> foo;
…
int a;
L4Re:chksys(foo->bar(42, a));

printf("%d\n", a);
```

```
struct Foo_srv : L4::Epiface_t<Foo_srv, Foo>
{
  long op_bar(Foo::Rights, int value, int &a)
  { a = 2 * value; return L4_EOK; }
};

L4Re::Util::Registry_server<…> server;
Foo_srv foo;
L4Re::chkcap(server.registry()->register_obj(&foo, "name")));
server.loop();
```



Fiasco.OC Object Model

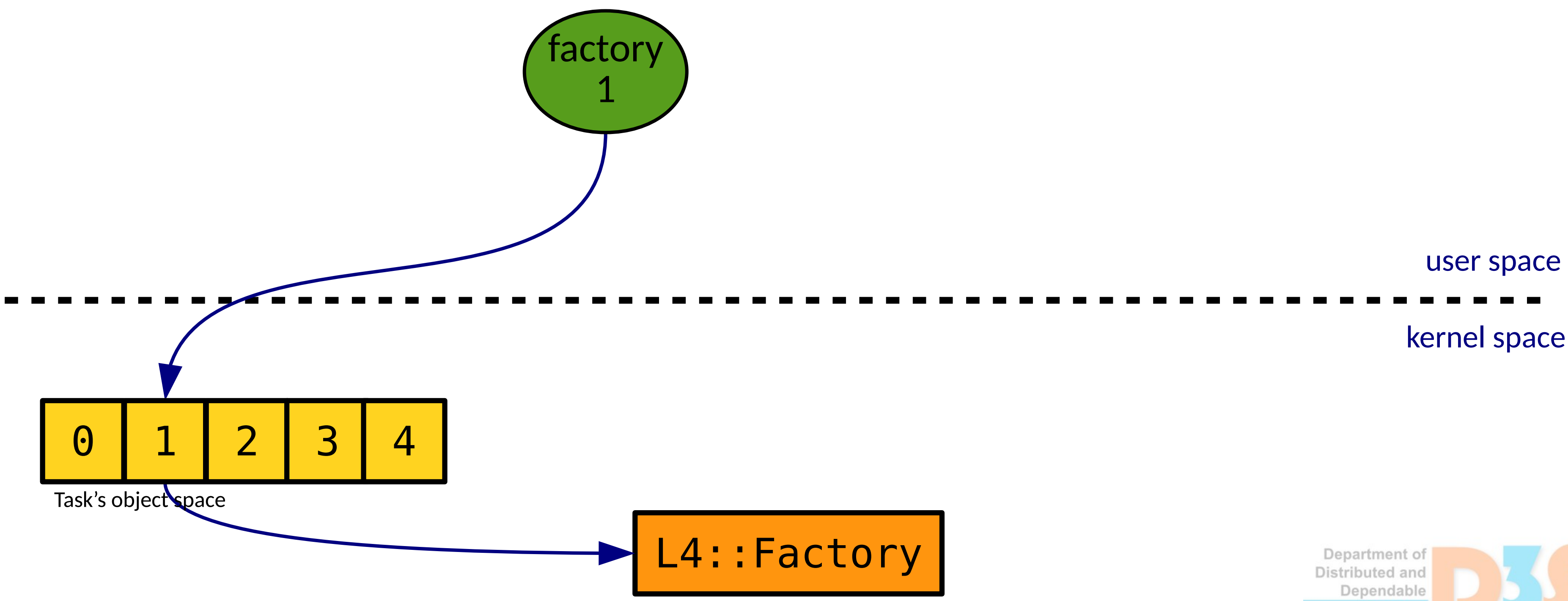
- **Kernel objects**
  - L4::Thread
  - L4::Task
  - L4::Ipc_gate
    - Object for implementing userspace objects
  - L4::Irq
  - L4::Semaphore
  - L4::Scheduler
  - L4::Factory
    - Creates new kernel objects subject to factory quota
  - L4::Vcon

Fiasco.OC Object Model (2)

- **Capabilities**
  - Typed by kernel/user object
  - Capability selectors / slots allocated in userspace
    - Like in seL4
    - Unlike in HelenOS, Mach, file descriptors
  - Mapped to kernel object upon object creation
  - Can be sent via IPC as a typed item
  - Can be mapped to a task via its capability
- **Syscall**
  - Invocation of capability via IPC

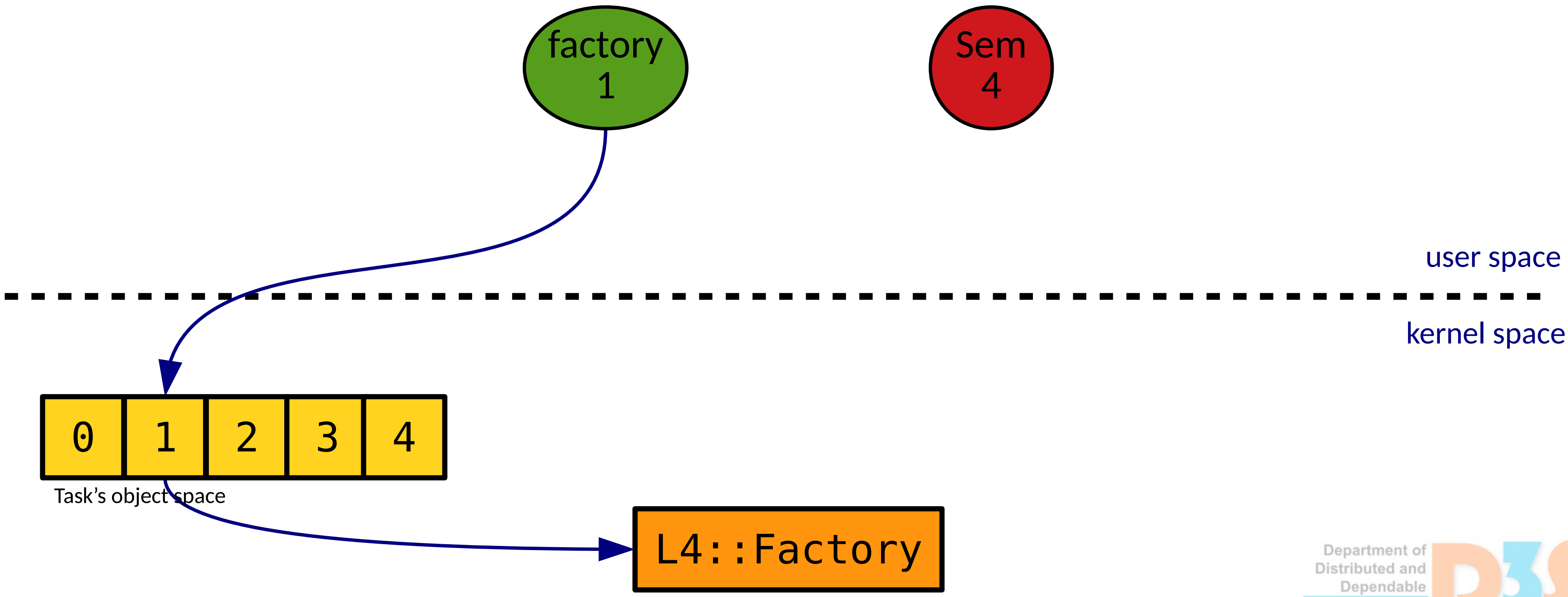
Department of Distributed and Dependable Systems

New Object Creation in L4Re / Fiasco.OC



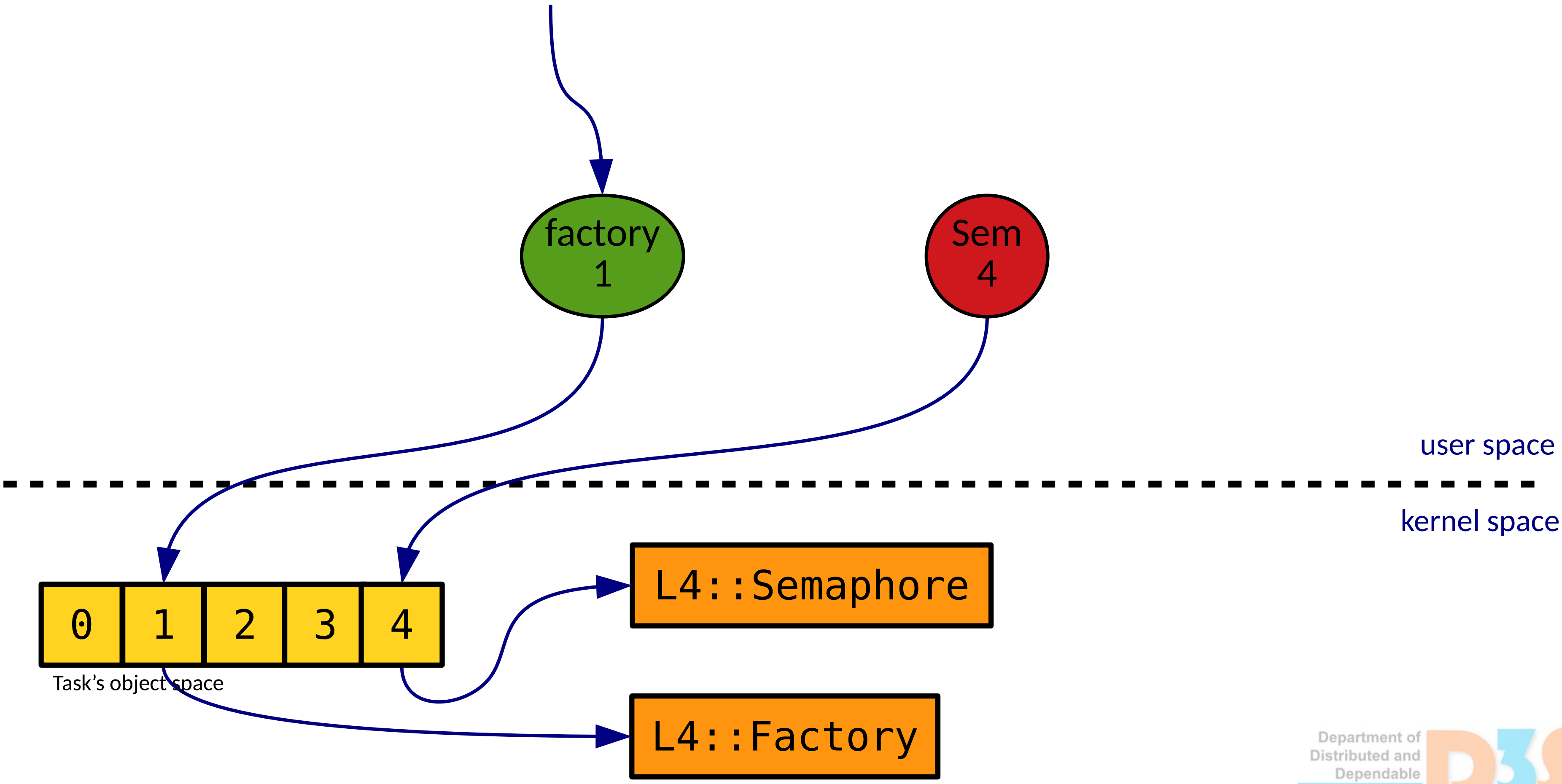
New Object Creation in L4Re / Fiasco.OC

```
auto sem = L4Re::chkcap(L4Re::Util::make_unique_cap<L4::Semaphore>());
```



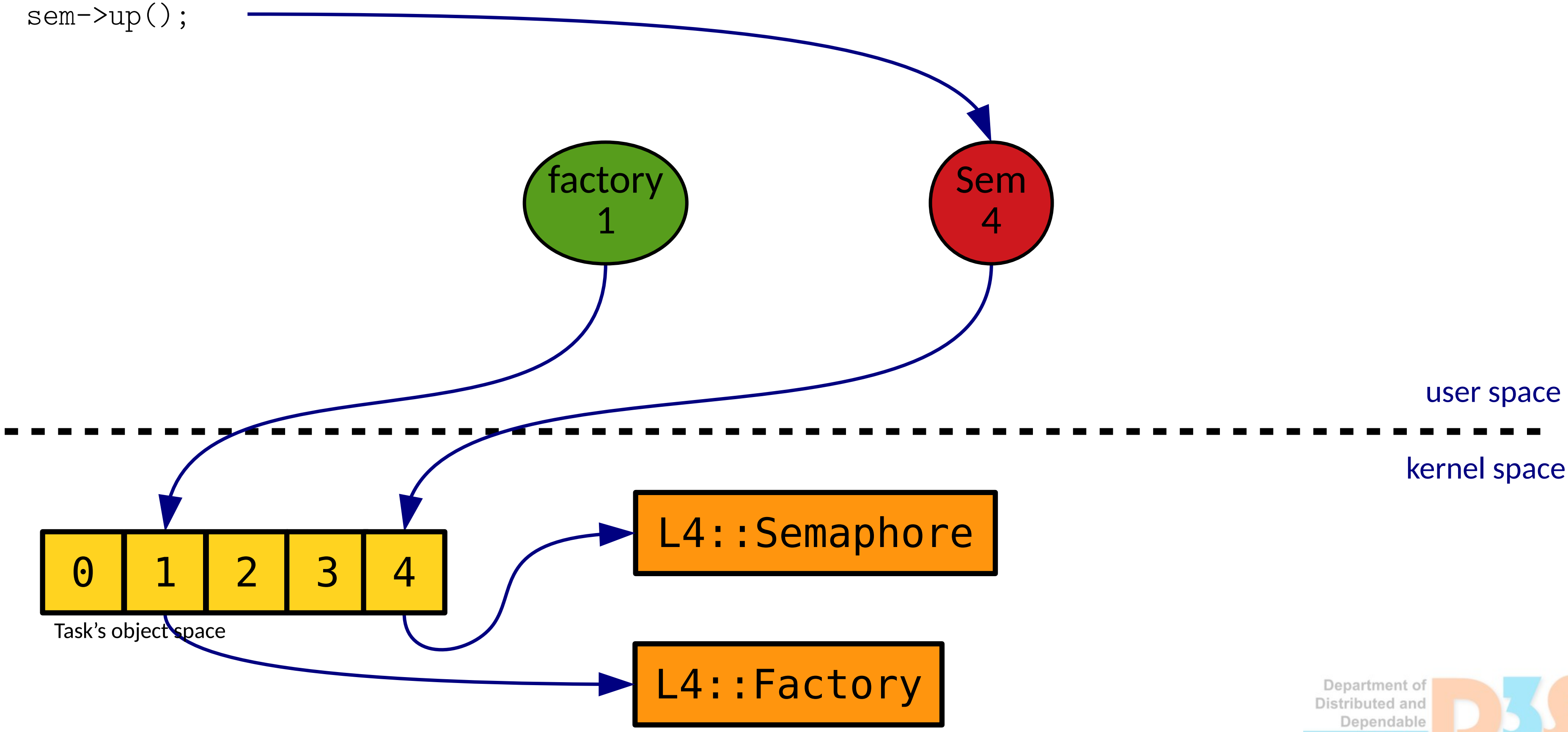
New Object Creation in L4Re / Fiasco.OC

```
auto sem = L4Re::chkcap(L4Re::Util::make_unique_cap<L4::Semaphore>());

L4Re::chksys(L4Re::Env::env()->factory()->create(sem.get()));
```



New Object Creation in L4Re / Fiasco.OC

```
auto sem = L4Re::chkcap(L4Re::Util::make_unique_cap<L4::Semaphore>());

L4Re::chksys(L4Re::Env::env()->factory()->create(sem.get()));

sem->up();
```



Department of Distributed and Dependable Systems

Q&A

References

- Madhavapeddy, A., Scott, D., J.: Unikernels: Rise of the Virtual Library Operating System, ACM Queue, 2013
- Porter, D., E., et al.: Rethinking the library OS from the top down, ASPLOS, 2011
- Kantee, A.: The Rise and Fall of the Operating System, ;login:, October 2015, Vol. 40, No. 5
- Baumann, A., et al.: The Multikernel: A new OS architecture for scalable multicore systems, SOSP '09
- L4hq.org: Kernel APIs, http://l4hq.org/kernels/
- Kuz, I.: L4 User Manual, API Version X.2, NICTA 2004
- L4Re Documentation: http://l4re.org/doc/
- Herder, J., N., et al.: Countering IPC Threats in Multiserver Operating Systems, IEEE PRDC 2008
- Heiser, G.,: From L3 to seL4: What have we learnt in 20 years of L4 microkernels?, Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 2013
- Feske, N.,: A Case Study on the Cost and Benefit of Dynamic RPC Marshalling for Low-Level System Components, SIGOPS OSR Special Issue on Secure Small-Kernel Systems, July 2007
- Hartig, H., Hohmuth, M., Liedtke, J., Schoenberg, S., Wolter, J.,: The Performance of µ-Kernel-Based Systems. ACM SIGOPS Operating Systems Review. 31. 10.1145/269005.266660, 1997
- Golub, D., et. al.: Unix as an Application Program, USENIX 1990 Summer Conference