M3 – Microkernel for Minimalist Manycores

Nils Asmussen

MKC, 06/19/2014

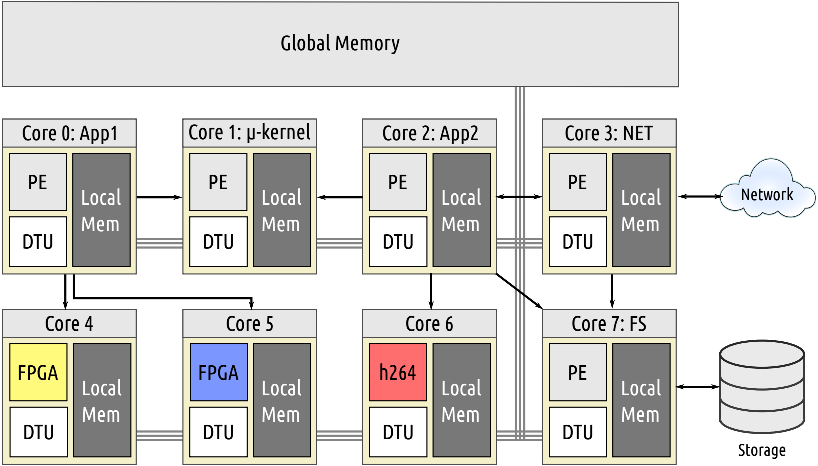
Outline

Outline

Motivation

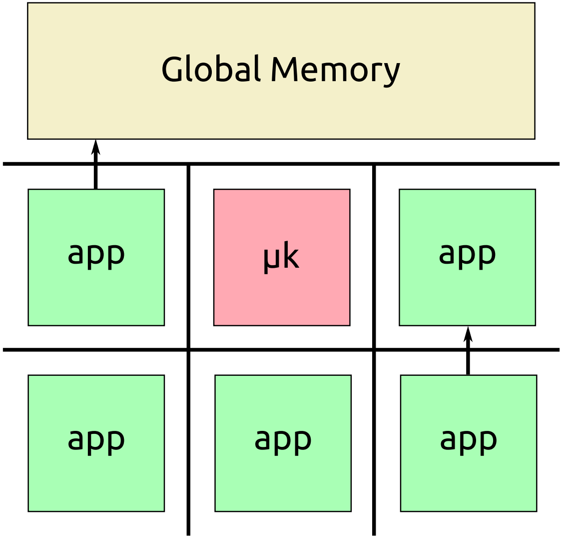
- We are moving towards accelerator architectures
- But how to really integrate heterogeneous cores and provide isolation?
- Energy consumption becomes more and more important
- Experiments show, that caches and MMUs consume lots of energy
- We try to address these issues by supporting heterogeneous cores, provide isolation, but be less energy demanding

Vision

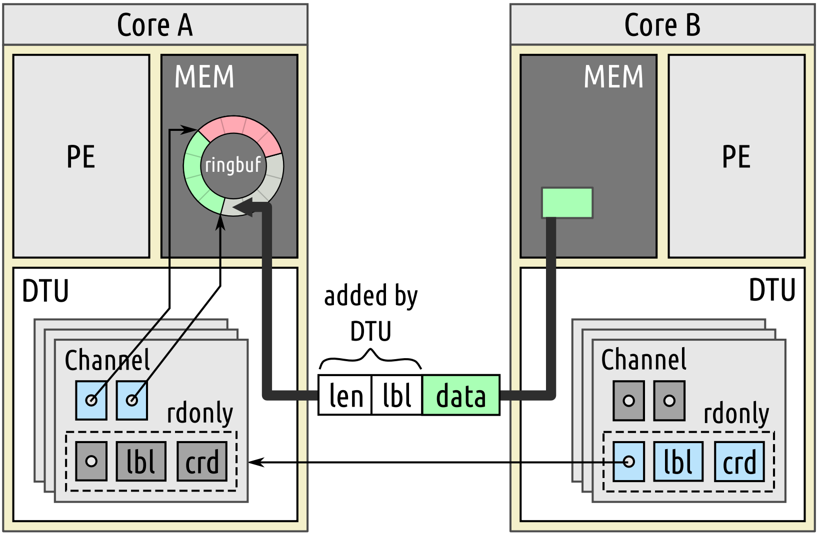

Global Memory
Core 0: App1
PE
DTU
Local Mem
Core 1: µ-kernel
PE
DTU
Local Mem
Core 2: App2
PE
DTU
Local Mem
Core 3: NET
PE
DTU
Local Mem
Network
Core 4
FPGA
DTU
Local Mem
Core 5
FPGA
DTU
Local Mem
Core 6
h264
DTU
Local Mem
Core 7: FS
PE
DTU
Local Mem
Storage

Outline

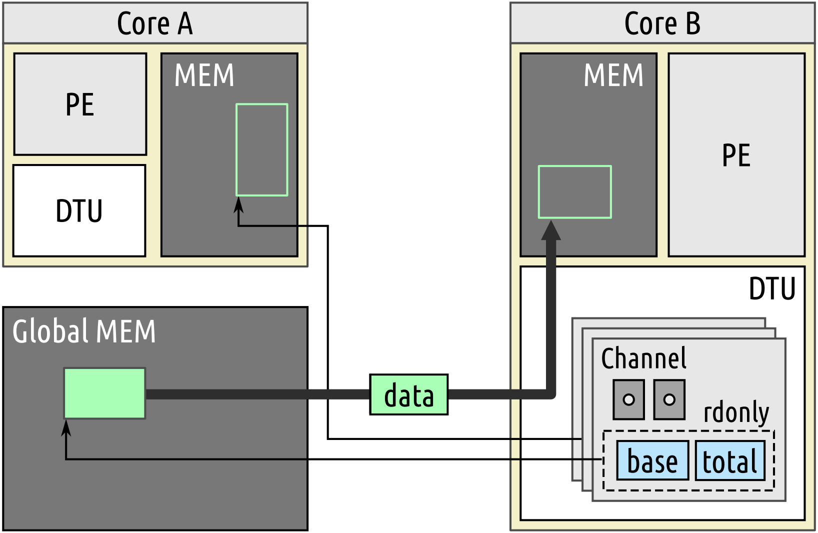
Isolation Principle



Data Transfer between Cores



Memory Access

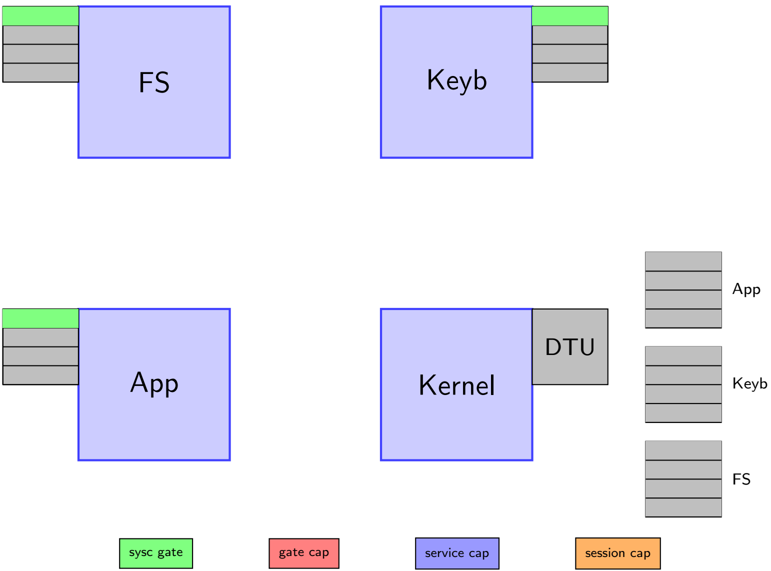


Outline

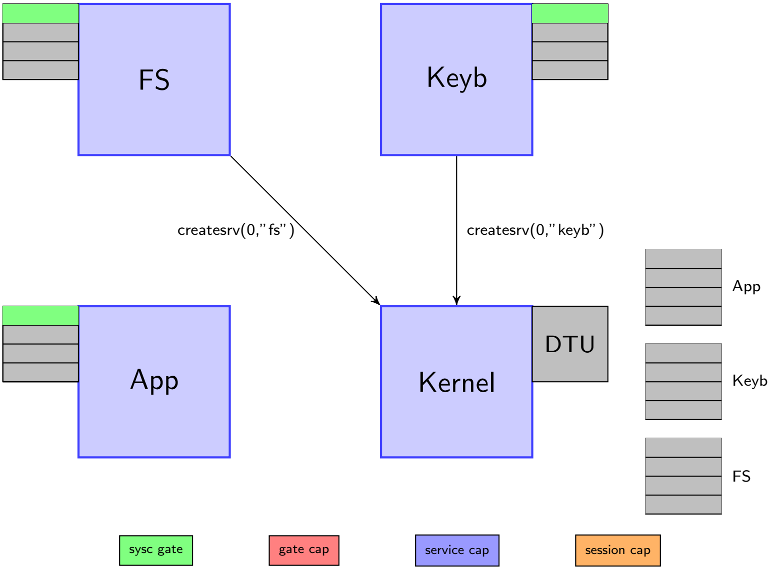
Capabilities

- The μ-kernel manages the capabilities of all tasks
- Like in L4, capabilties are protected by the kernel, selectors are managed by the applications
- Normal IPC can't be used because the kernel is not involved
- Thus, a special protocol is used to exchange capabilities

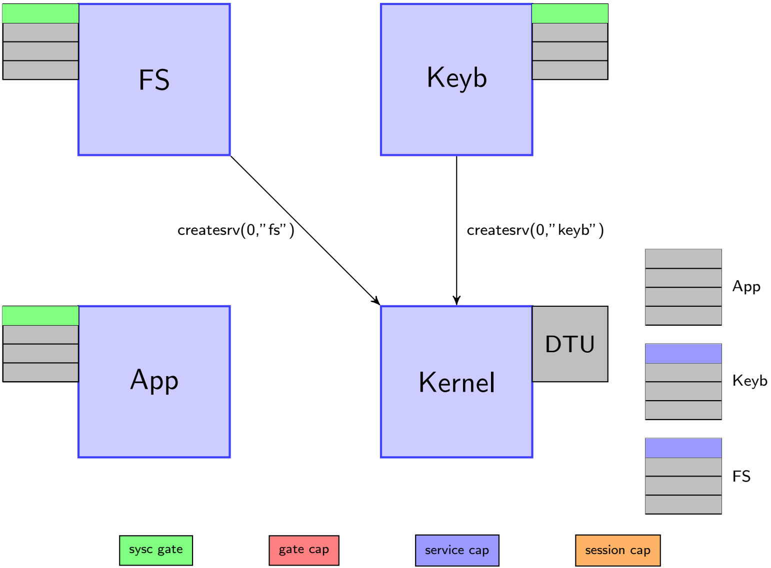
Sessions


FS
Keyb
App
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

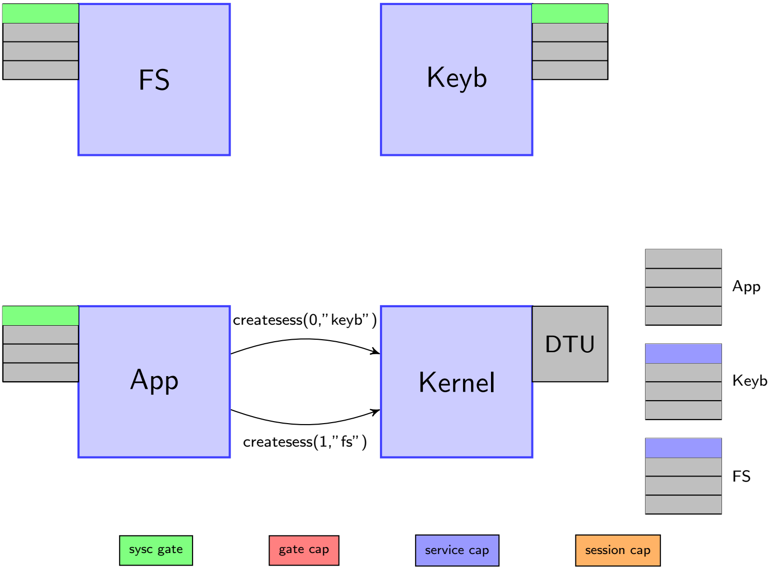
Sessions


FS
Keyb
createsrv(0,"fs")
createsrv(0,"keyb")
App
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

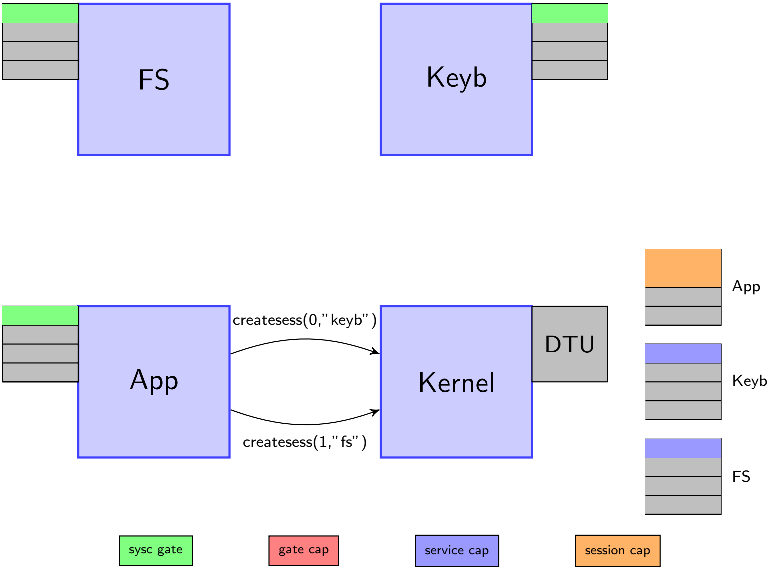
Sessions


FS
Keyb
createsrv(0,"fs")
createsrv(0,"keyb")
App
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

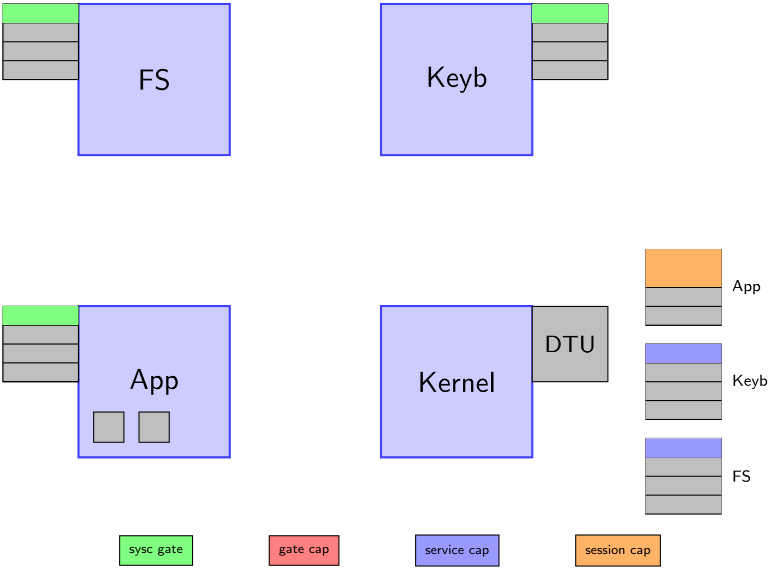
Sessions


FS
Keyb
App
createsess(0,"keyb")
createsess(1,"fs")
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

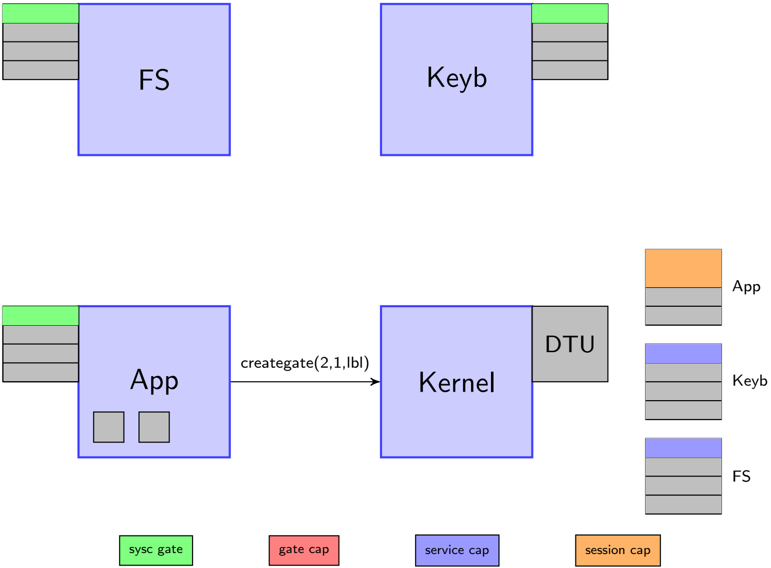
Sessions


FS
Keyb
App
createsess(0,"keyb")
createsess(1,"fs")
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

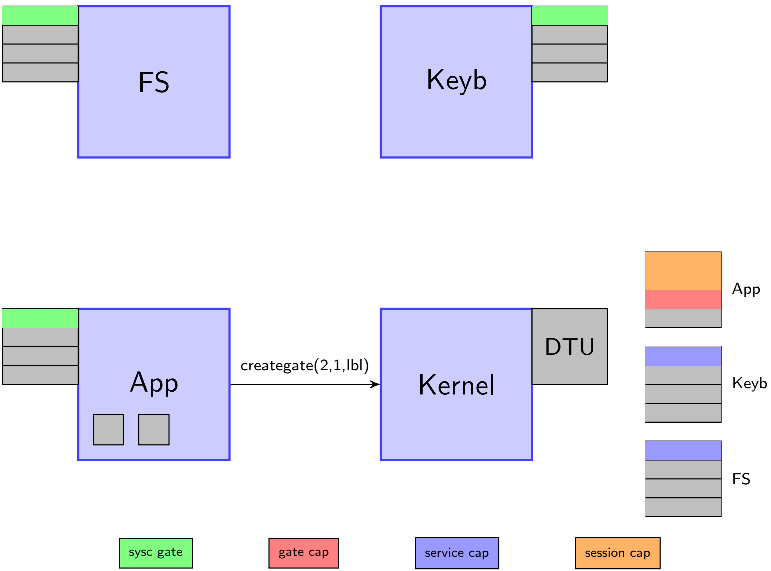
Sessions


FS
Keyb
App
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

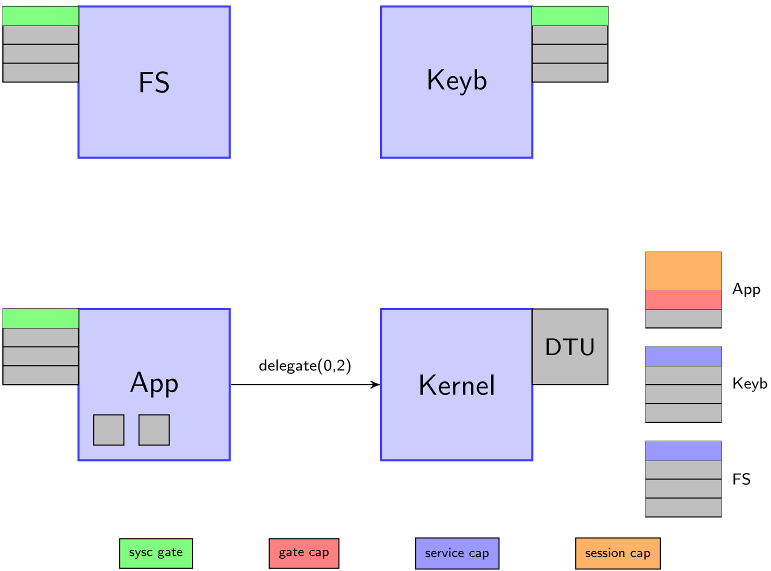
Sessions


FS
Keyb
App
creategate(2,1,lbl)
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

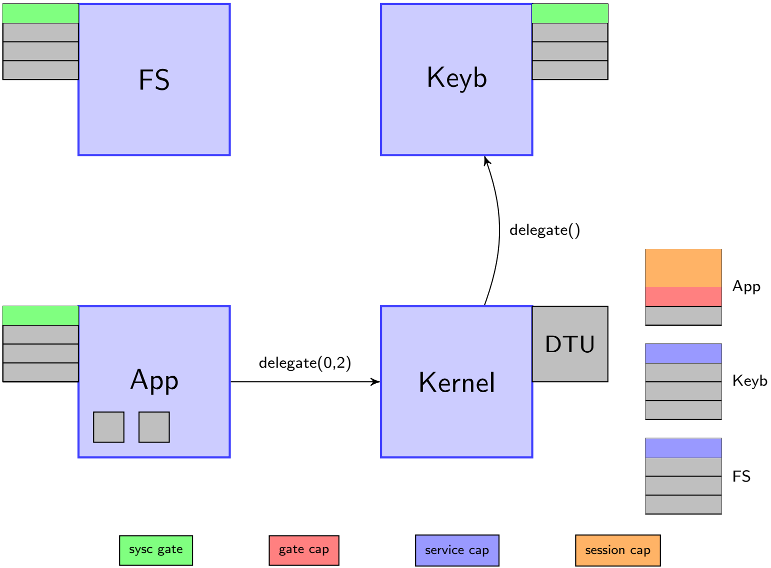
Sessions


FS
Keyb
App
creategate(2,1,lbl)
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

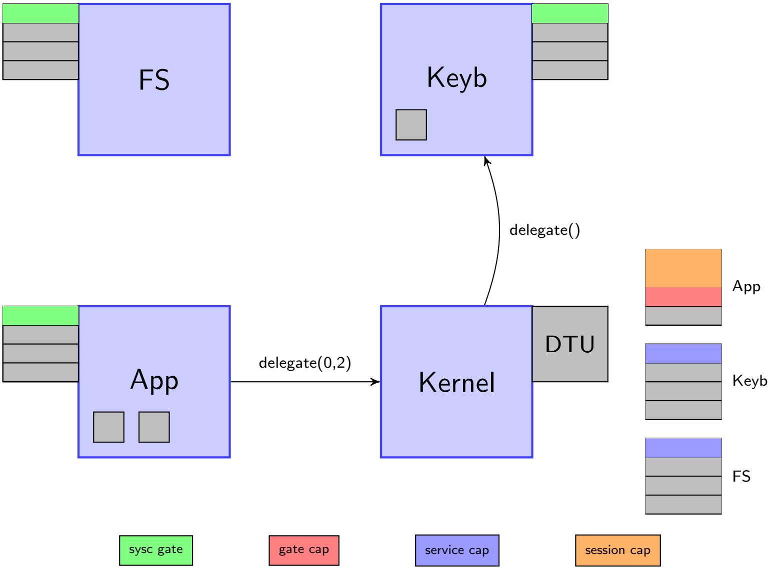
Sessions


FS
Keyb
App
delegate(0,2)
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

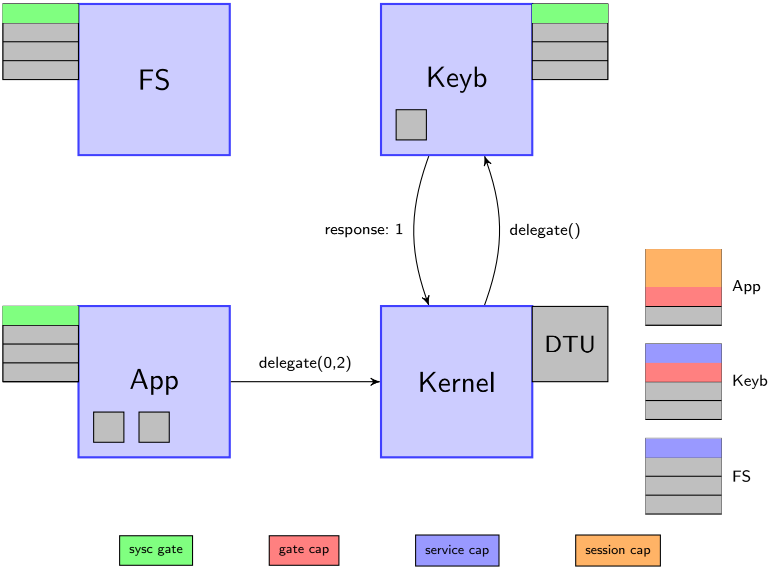
Sessions


FS
Keyb
delegate()
App
delegate(0,2)
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

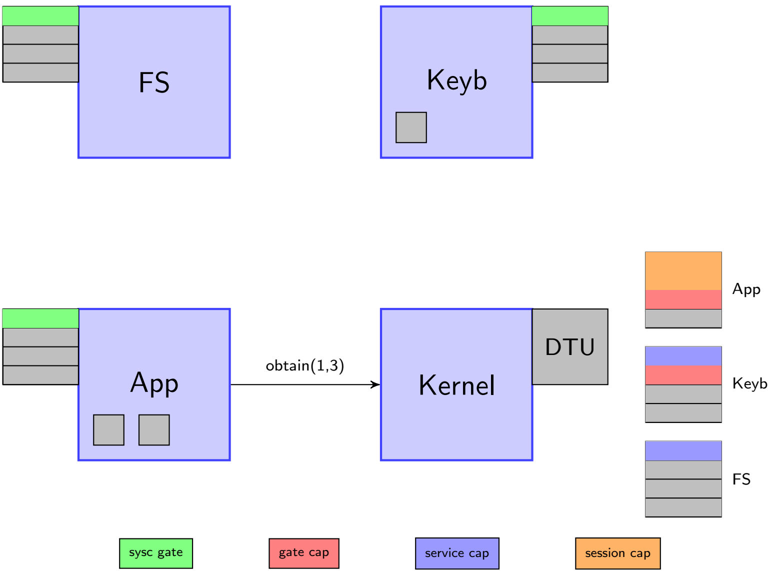
Sessions


FS
Keyb
delegate()
App
Keyb
FS
App
delegate(0,2)
Kernel
DTU
sysc gate
gate cap
service cap
session cap

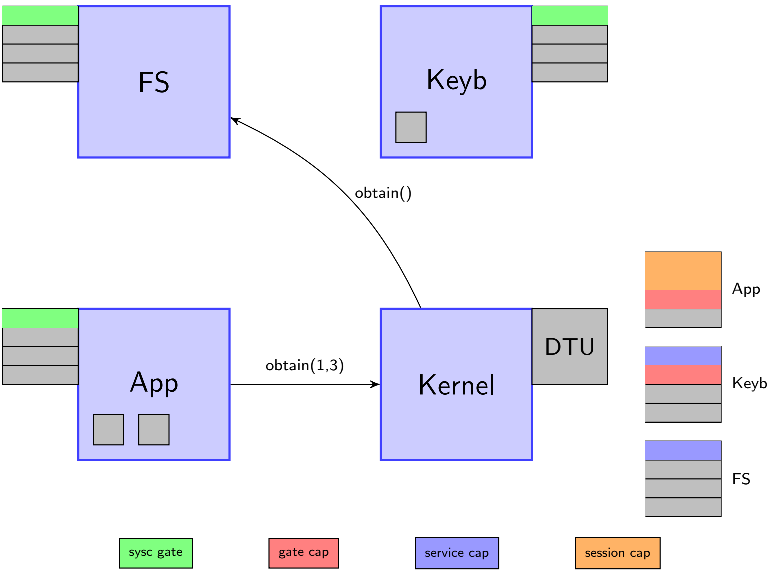
Sessions


FS
Keyb
response: 1
delegate()
App
Kernel
DTU
delegate(0,2)
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

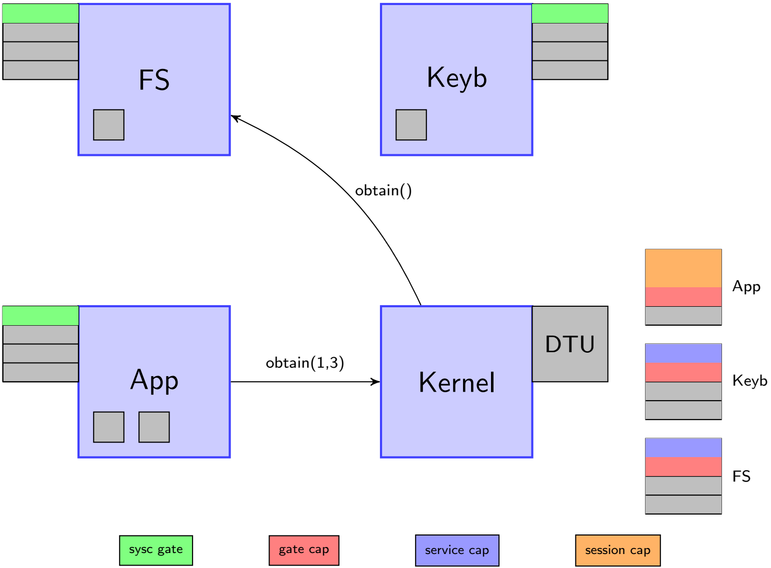
Sessions


FS
Keyb
App
obtain(1,3)
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

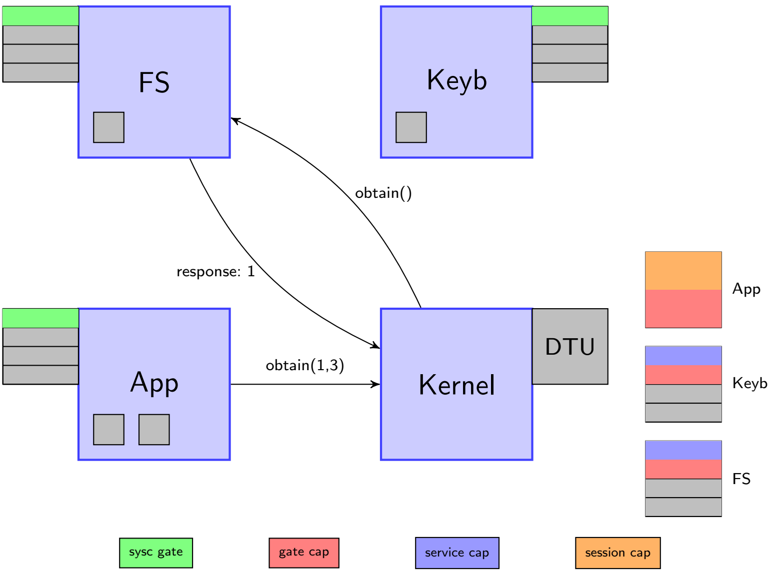
Sessions


FS
Keyb
obtain()
App
App
obtain(1,3)
Kernel
DTU
Keyb
FS
sysc gate
gate cap
service cap
session cap

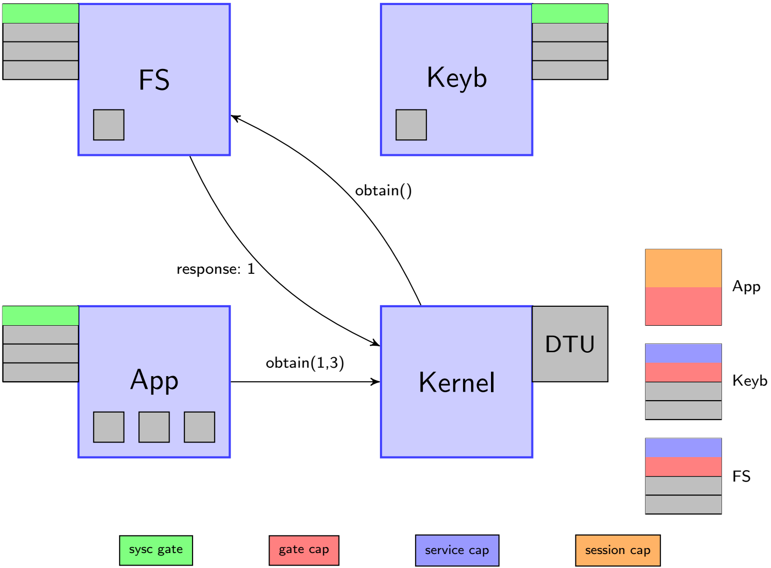
Sessions


FS
Keyb
obtain()
App
Keyb
FS
DTU
obtain(1,3)
App
Kernel
sysc gate
gate cap
service cap
session cap

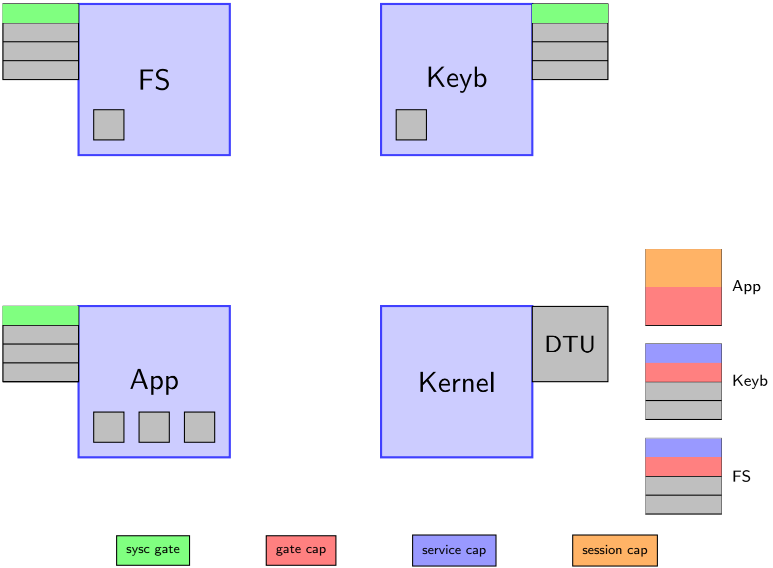
Sessions


FS
Keyb
obtain()
response: 1
App
Kernel
DTU
obtain(1,3)
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

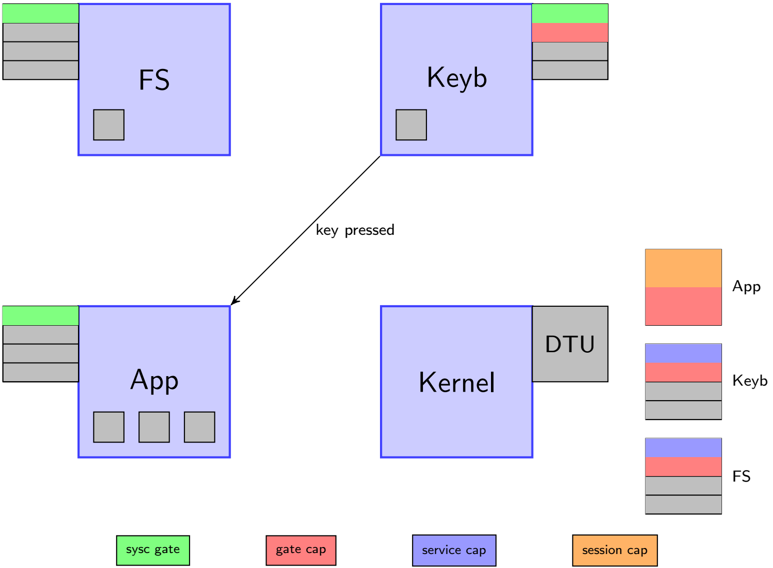
Sessions


FS
Keyb
obtain()
response: 1
App
obtain(1,3)
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

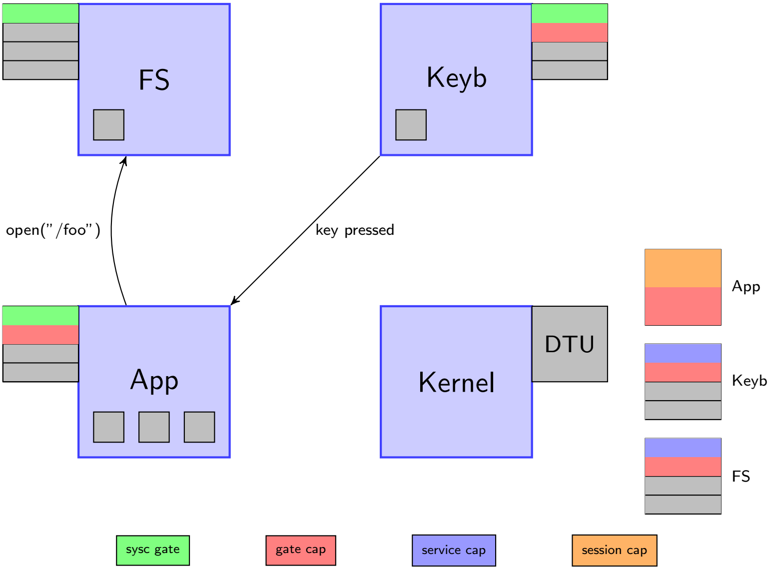
Sessions


FS
Keyb
App
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

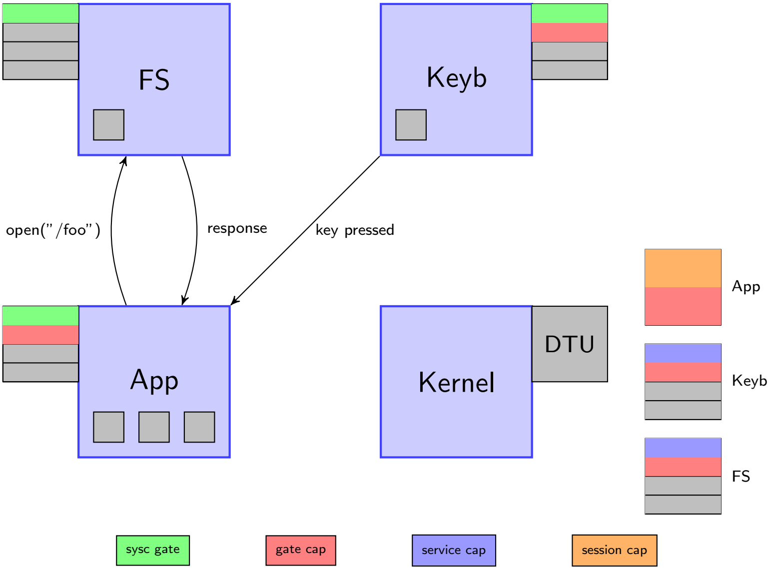
Sessions


FS
Keyb
key pressed
App
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

Sessions


FS
Keyb
open("/foo")
key pressed
App
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

Sessions


FS
Keyb
open("/foo")
response
key pressed
App
Kernel
DTU
App
Keyb
FS
sysc gate
gate cap
service cap
session cap

Memory

- The syscall `reqmem` can be used to request global memory
- Global memory is managed by the kernel
- This memory capability can be put on a channel to access it
- For that reason, memory capabilities are the same as gate capabilities
- I.e. unlike L4, there are no different kinds of capabilities
- Syscall `derivemem` to create a memory capability for a subset of a given one
- Syscall `createmem` to create a memory capability for the own local memory

Filesystem

- There is a filesystem service running on some core
- The FS uses global memory as the inode and buffer cache
- App goes to FS in order to know where file fragments are in global memory
- FS hands out memory capabilities to the application
- App reads/writes directly from/to global memory afterwards

Filesystem – API

- Like in L4Re, the mount-tree is application local
- The File abstraction is the low level abstraction to read/write files
- The BufferedFile abstraction sits on top and provides a buffer in between

Filesystem – API

- Like in L4Re, the mount-tree is application local
- The File abstraction is the low level abstraction to read/write files
- The BufferedFile abstraction sits on top and provides a buffer in between

Example

```
VFS::mount("/", "m3fs");

BufferedFile file;
VFS::open("/myfile", VFS::R, file);
while(file.read(buf, 512) > 0) {
    // ...
}
```

Tasks

- Syscall `createtask` to create a new task
- The kernel assigns the task to a core, if there is a free one
- The application gets the task capability and a memory capability for the whole local memory of that core
- Thus, it has complete control over that core
- Syscall `start` to start the task
- It can exchange capabilities with the task before starting it
- Syscall `wait` to wait until a task is finished (called `exit`)

Tasks – Examples

Executing ELF-Binaries

```
Task task("test");
char *args[] = {"/bin/hello", "foo", "bar"};
task.exec(3, args);
```

Tasks – Examples

Executing ELF-Binaries

```
Task task("test");
char *args[] = {"/bin/hello", "foo", "bar"};
task.exec(3, args);
```

Asynchronous Lambdas

```
Task task("test");
int a = 10;
task.delegate(CapRngDesc(1, 2));
task.run([a]() {
    Serial::get() << "a=" << a << "\n";
});
```

Outline

Summary

- M3 is a different kind of microkernel-based system
- The μ-kernel manages remote resources instead of local resources
- The μ-kernel and applications run alone on the cores
- Applications are linked against a library to get abstractions
- Capabilities are used to manage permissions in the system
- Similar operations with caps, but still different