barkhausen institut

Microkernel Construction

M³ Tutorial

Viktor Reusch, Nils Asmussen

07/02/2026

Preparation

- Install *qemu* or *VirtualBox* (or anything else that supports *qcow2*/*VDI*)
- Download VM image matching your setup from the course website
- Extract the image (≈15 GiB): `tar xzf m3-tutorial.….tar.gz`

Preparation

- Install *qemu* or *VirtualBox* (or anything else that supports *qcow2*/*VDI*)
- Download VM image matching your setup from the course website
- Extract the image (≈15 GiB): `tar xzf m3-tutorial.….tar.gz`
- Create a virtual machine (with 4 cores and 8 GiB of memory):
  - `qemu-system-x86_64 -enable-kvm -smp 4 -m 8G -drive if=virtio,format=qcow2,file=m3-tutorial.qcow2`
  - `qemu-system-aarch64 -enable-kvm -smp 4 -m 8G -drive if=virtio,format=qcow2,file=m3-tutorial.qcow2 -bios /usr/share/qemu-efi-aarch64/QEMU_EFI.fd -cpu max -M virt -device virtio-gpu-pci -device qemu-xhci -device usb-tablet -device usb-kbd`
  - Create new appliance in VirtualBox with VDI

Preparation

- Install *qemu* or *VirtualBox* (or anything else that supports *qcow2*/*VDI*)
- Download VM image matching your setup from the course website
- Extract the image (≈15 GiB): `tar xzf m3-tutorial.….tar.gz`
- Create a virtual machine (with 4 cores and 8 GiB of memory):
  - `qemu-system-x86_64 -enable-kvm -smp 4 -m 8G -drive if=virtio,format=qcow2,file=m3-tutorial.qcow2`
  - `qemu-system-aarch64 -enable-kvm -smp 4 -m 8G -drive if=virtio,format=qcow2,file=m3-tutorial.qcow2 -bios /usr/share/qemu-efi-aarch64/QEMU_EFI.fd -cpu max -M virt -device virtio-gpu-pci -device qemu-xhci -device usb-tablet -device usb-kbd`
  - Create new appliance in VirtualBox with VDI
- Open terminal, go to `$HOME/M3`
- If required, user name: `m3`

SSH Access

- You can access the VM via SSH for remote developing / *SFTP* / *SSHFS*
- Add to the QEMU command line: `-nic user,hostfwd=:127.0.0.1:2222-:22`
- Connect via: `ssh -p 2222 m3@127.0.0.1`

The Hands-on Part

- What you have to expect:
- Develop M³ applications
- Learn how to …
  - Send messages via the TCU
  - Use system services
  - Communicate with accelerators
- Inside the ready-made VM

- Introduction
- **Assignments**
  - **Hello World**
  - IPC
  - Filesystem Service
  - Accelerators
  - Sessions

Build System

- *ninjapie*
  - Originally created for M³
  - Primarily required for C/C++, but Rust is integrated as well
  - Open source: https://github.com/Barkhausen-Institut/Ninjapie
  - Simple Python scripts

Build System

- *ninjapie*
  - Originally created for M³
  - Primarily required for C/C++, but Rust is integrated as well
  - Open source: https://github.com/Barkhausen-Institut/Ninjapie
  - Simple Python scripts
- Heavy lifting done by *ninja*
  - Very fast
  - Build files not designed for humans
  - Generated from Python scripts by ninjapie

Build System

- *ninjapie*
  - Originally created for M³
  - Primarily required for C/C++, but Rust is integrated as well
  - Open source: https://github.com/Barkhausen-Institut/Ninjapie
  - Simple Python scripts
- Heavy lifting done by *ninja*
  - Very fast
  - Build files not designed for humans
  - Generated from Python scripts by ninjapie
- Convenience script: `b`
  - Builds everything
  - Provides commands for running, debugging, analyzing, …

Working with M³

- Environment variables
  - **M3_ISA**: x86_64, riscv
  - **M3_BUILD**: release, debug, bench
  - **M3_GEM5_CPU**: DerivO3CPU, TimingSimpleCPU
  - Already set for you in `.envrc`

Working with M³

- Environment variables
  - `M3_ISA`: `x86_64`, `riscv`
  - `M3_BUILD`: `release`, `debug`, `bench`
  - `M3_GEM5_CPU`: `DerivO3CPU`, `TimingSimpleCPU`
  - Already set for you in `.envrc`
- Commands
  - `./b`: build everything (no command)
  - `./b run <script>`: run given boot script (e.g., `boot/hello.xml`)
  - `./b -n clippy [path]`: Rust linting
  - `./b -n fmt`: format source code
  - `./b -n doc`: generate documentation
  - `./b -h`: detailed help

Build Scripts

- `/build.py`
  - Main file of Ninjapie
  - Generates ninja build file
  - Called by `b`

Build Scripts

- `/build.py`
  - Main file of Ninjapie
  - Generates ninja build file
  - Called by `b`
- `…/app/build.py`
  - Called by `/build.py`
  - Describes how to build a specific program/library/…
  - Example for Rust application:

```python
def build(gen, env):
    env.m3_rust_exe(gen, out = 'hello')
```

Building Rust With Cargo

.../app/Cargo.toml:

```toml
[package]
name = "hello"
version = "0.1.0"
edition = "2021"

# The M3 build system will link the final binary.
[lib]
path = "src/hello.rs"
crate-type = ["staticlib"]

[dependencies]
m3 = { path = "../../libs/rust/m3" }
```

Boot Scripts

/boot/….xml:

XML

```xml
<config>
    <kernel args="kernel"/> <!-- Start kernel -->
    <dom> <!-- On another tile -->
        <app args="root"> <!-- Start init process -->
            <dom>
                <app args="hello"/> <!-- Start hello application -->
            </dom>
            <dom>
                <app args="hello"/>
            </dom>
        </app>
    </dom>
</config>
```

Rust Primer

Rust

```rust
use m3::errors::{Code, Error}; // Imports
use m3::println;
pub fn my_func(param: &bool) -> Result<bool, Error> { // Function head
    let mut my_var: isize = 128; // Declare and define mutable integer
    other_func(&mut my_var); // Call function with mutable reference
    let my_other = calc_func(&my_var, 12 as i32); // Integer conversion
    if my_other.value < 128 { // Condition
        println!("Got integer {}!", my_other.info()); // Print macro
        return Ok(true); // Return enum variant
    }
    let ok_val = fallible_func().unwrap(); // Assert Result is Ok
    Err(Error::new(Code::Abort)) // Construct error object and return
}
```

Application Code

/src/apps/…/src/….rs:

Rust

```rust
#![no_std]

use m3::errors::Error;
use m3::println;

#[no_mangle]
pub fn main() -> Result<(), Error> {
    println!("Hello world!");
    Ok(())
}
```

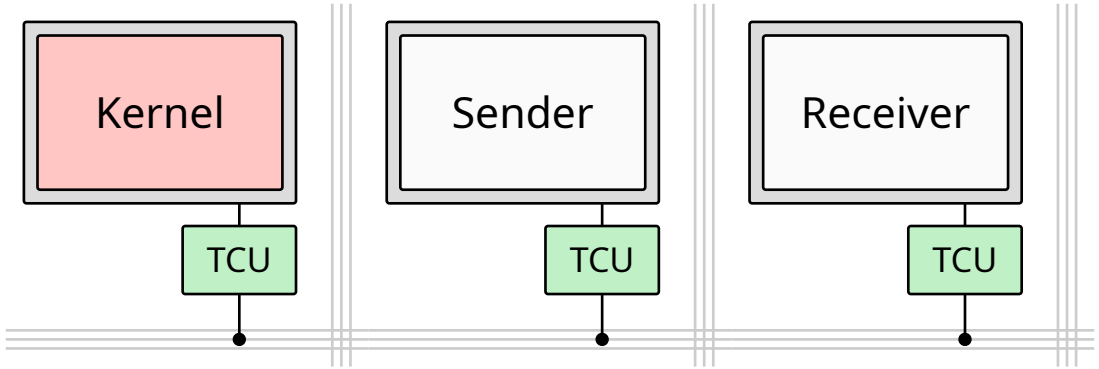
Execution Demo

Assignment #1: Hello World

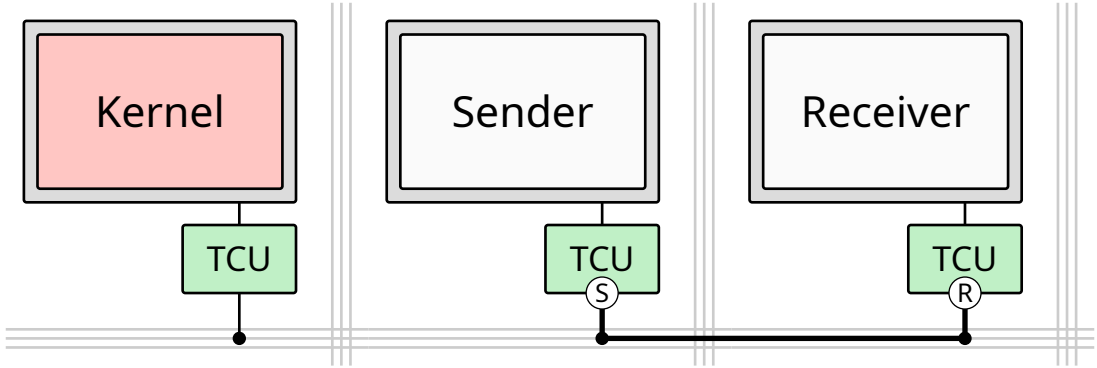
- Run the `rust-hello.xml` boot script
- It should print `Hello world!` in the terminal
- Modify the boot script to start two instances of the hello world application
- Modify the source code to print `Hello tutorial!` instead
- Run it again

- Introduction
- **Assignments**
  - Hello World
  - **IPC**
  - Filesystem Service
  - Accelerators
  - Sessions

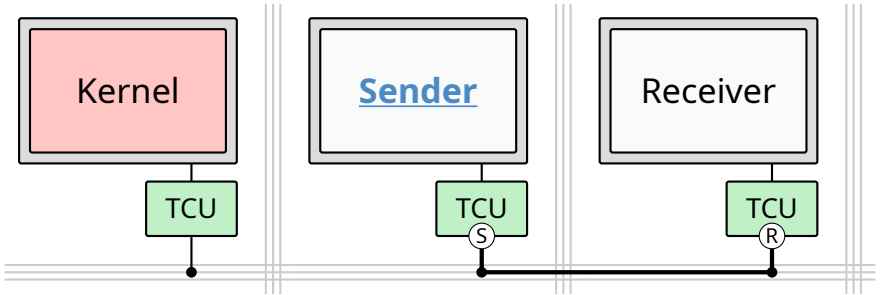
IPC on M³



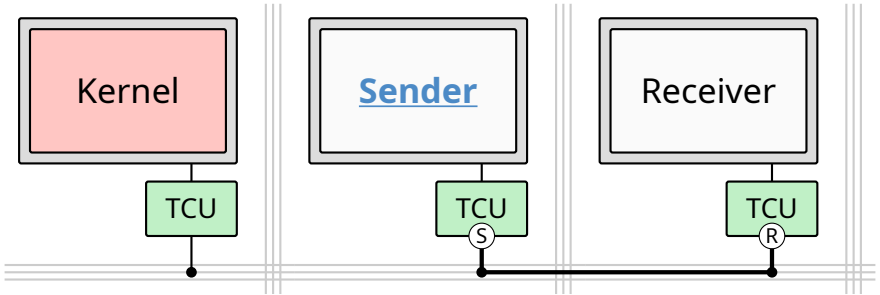
IPC on M³



Message Passing



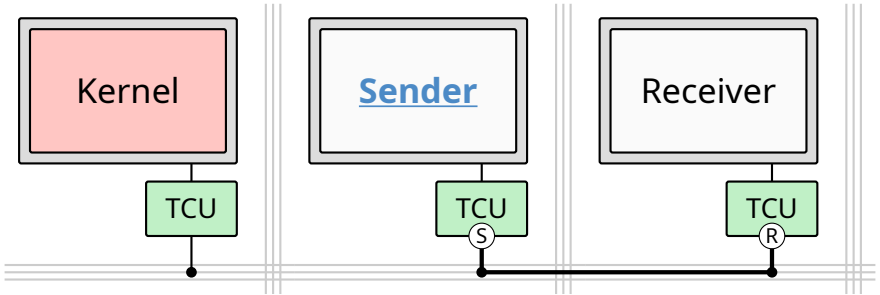
Message Passing



`SendGate`

- Allows to **send messages** to the connected `RecvGate`
- If sender specifies a “reply gate”, receiver can reply to each message

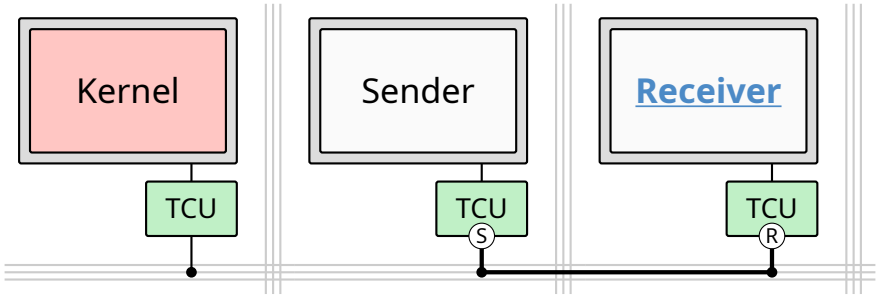
Message Passing



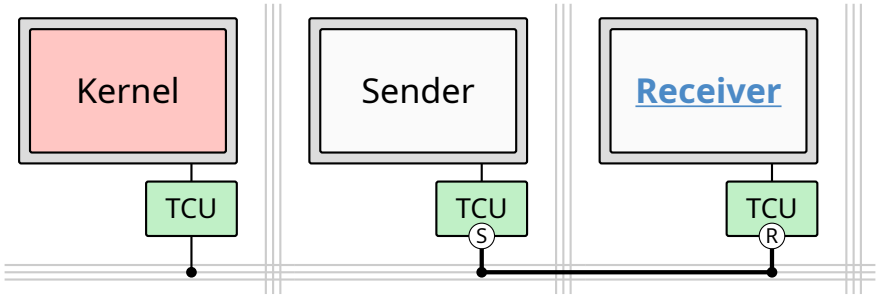
`SendGate`

- Allows to **send messages** to the connected `RecvGate`
- If sender specifies a “reply gate”, receiver can reply to each message
- Has **credits**: each credit allows to send one message
- Sending removes one credit, receiving a reply adds one credit

Message Passing



Message Passing



`RecvGate`

- Allows to **receive messages** from multiple `SendGate`s
- Has a receive buffer of `N` **slots** with `M` **bytes each**
- Sends fail if the message is larger than `M` or no slot is free
- If the credit system is used, the latter case can be avoided

Message Passing: ACKing messages

- Each received message is stored in a slot of the receive buffer
- The TCU needs to know **when slots can be reused**

Message Passing: ACKing messages

- Each received message is stored in a slot of the receive buffer
- The TCU needs to know **when slots can be reused**
- Replying on a message implicitly **acknowledges** this message
- If no reply is used, messages need to be acknowledged explicitly:

```rust
impl RecvGate {
    fn ack_msg(&self, msg: &tcu::Message) -> Result<(), Error> {}
}
```

... or automatically when using `GateIStream`

Message Passing: Marshalling

Unmarshalling: `GateIStream`

```rust
let mut gate_stream = recv_msg(&recv_gate).expect("receive failed");
let val = gate_stream.pop::<u32>().expect("unable to get value");
let text = gate_stream.pop::<String>().unwrap();
```

Message Passing: Marshalling

Unmarshalling: `GateIStream`

```rust
let mut gate_stream = recv_msg(&recv_gate).expect("receive failed");
let val = gate_stream.pop::<u32>().expect("unable to get value");
let text = gate_stream.pop::<String>().unwrap();
```

Marshalling: send macros

```rust
send_vmsg!(send_gate, &reply_gate, arg1, /*…*/);
reply_vmsg!(gate_stream, arg1, /*…*/);
let gate_stream = send_recv!(send_gate, &reply_gate, arg1, /*…*/);
```

Message Passing: Creation

- Programmatically:
  - `RecvGate::new(order, msg_order)`
  - `SendGate::new(&recv_gate)`
  - Receiver needs to delegate `SendGate` to sender

Message Passing: Creation

- Programmatically:
  - `RecvGate::new(order, msg_order)`
  - `SendGate::new(&recv_gate)`
  - Receiver needs to delegate `SendGate` to sender
- Configuration-based:
  - `SendGate::new_named(&name)`
  - `RecvGate::new_named(&name)`
  - Names and parameters defined in boot script

Message Passing: Creation

- Programmatically:
  - `RecvGate::new(order, msg_order)`
  - `SendGate::new(&recv_gate)`
  - Receiver needs to delegate `SendGate` to sender
- Configuration-based:
  - `SendGate::new_named(&name)`
  - `RecvGate::new_named(&name)`
  - Names and parameters defined in boot script

‼ The sender also needs a receive gate for replies. ‼

Create this receive gate programmatically!

Boot Script Example

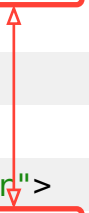
XML

```xml
<!-- … -->
<app args="root">
  <dom>
    <app args="receiver">
      <rgate name="chan" msgsize="64" slots="2"/>
    </app>
  </dom>
  <dom>
    <app args="sender">
      <sgate name="chan" credits="2"/>
    </app>
  </dom>
</app>
<!-- … -->
```

Boot Script Example

XML

```xml
<!-- … -->
<app args="root">
  <dom>
    <app args="receiver">
      <rgate name="chan" msgsize="64" slots="2"/>
    </app>
  </dom>
  <dom>
    <app args="sender">
      <sgate name="chan" credits="2"/>
    </app>
  </dom>
</app>
<!-- … -->
```



Boot Script Example

XML

```xml
<!-- … -->
<app args="root">
  <dom>
    <app args="receiver">
      <rgate name="chan" msgsize="64" slots="2"/>
    </app>
  </dom>
  <dom>
    <app args="sender">
      <sgate name="chan" credits="2"/>
    </app>
  </dom>
</app>
<!-- … -->
```

RecvGate::new_named("chan")

SendGate::new_named("chan")

Wait! Debugging

- Logging:
  - Pass flags via `M3_LOG=...` when starting the system
  - Example: `M3_LOG=Info,Error,KernSysc,LibServ`
  - Hardcoded for `M3_BUILD=bench` to `Info,Error`
  - All flags are defined in `src/libs/rust/base/src/io/logflags.rs`

More Debugging

- Gem5 flags:
  - Can be specified via `M3_GEM5_LOG=…`
  - `Tcu`: show send, recv, …
  - `TcuCmd`: show TCU command executions
  - …

More Debugging

- Gem5 flags:
  - Can be specified via `M3_GEM5_LOG=…`
  - `Tcu`: show send, recv, …
  - `TcuCmd`: show TCU command executions
  - …
- Instruction trace:
  - Set `M3_GEM5_LOG=Exec`
  - Run your scenario in gem5
  - `grep C0T04 run/gem5.log | ./b trace tilemux,<program>`
    shows you a beautified version of the instructions executed on `C0T04`

Assignment #2: IPC

- Build a program that sends items
- Build a program that receives items
- Use the credit system
- Simple version: communicate synchronously
- Fill in implementation at `src/apps/ipc` and `boot/ipc.xml`

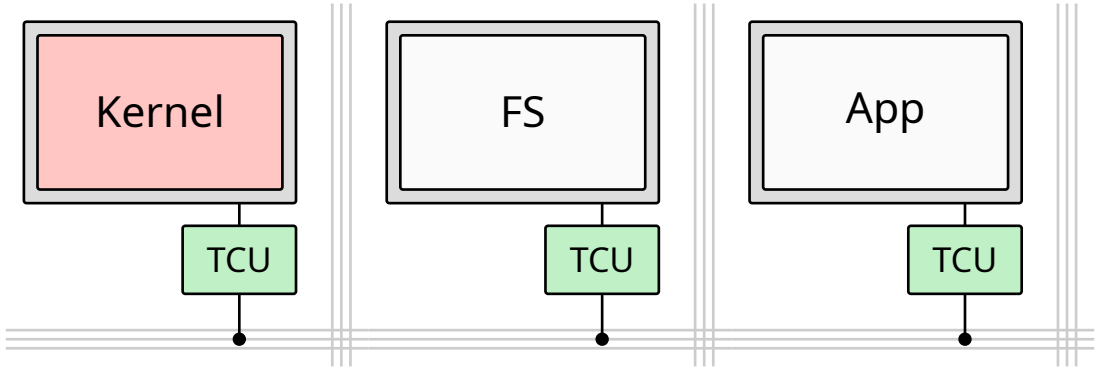
Assignment #2: IPC

- Build a program that sends items
- Build a program that receives items
- Use the credit system
- Simple version: communicate synchronously
- Fill in implementation at `src/apps/ipc` and `boot/ipc.xml`

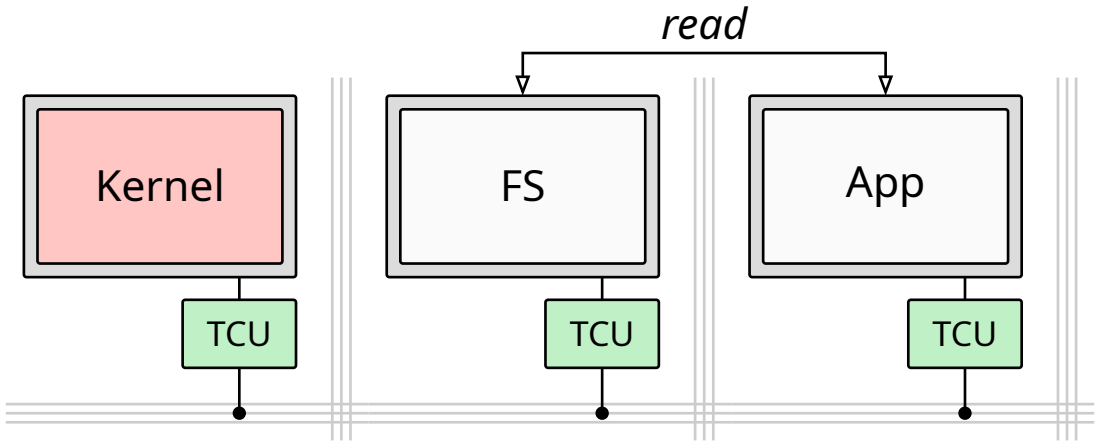
- Advanced version: communicate asynchronously
- Let the sender keep multiple messages in flight
- You must wait for a reply when credits/receive buffers depleted

- Introduction
- **Assignments**
  - Hello World
  - IPC
  - **Filesystem Service**
  - Accelerators
  - Sessions

Accessing Files



Accessing Files



Services

- A *server* offers a *service* for clients
- A **connection between client and server** is represented as a *session*
- Who can offer and use which service is defined via boot script

Boot Script Example (simplified)

XML

```xml
<mods><mod name="fs" file="default.img"/></mods>
<!-- … -->
<app args="m3fs mem" daemon="1">
  <serv name="m3fs"/>
  <mod name="fs"/>
</app>
<!-- … -->
<app args="pager">
  <sess name="m3fs"/>
  <mod name="fs" perm="r"/>
  <app args="/bin/myapp">
    <mount fs="m3fs" path="/"/>
  </app>
</app>
```

File API

Virtual File System

Rust

```rust
mod VFS {
  // Open `path` with given permissions
  fn open(path: &str, flags: OpenFlags)
    -> Result<FileRef<dyn File>, Error> {}
  // Create directory at `path` with given mode
  fn mkdir(path: &str, mode: FileMode) -> Result<(), Error> {}
}
```

FileRef and File

```rust
// Read into `buffer` (supports non-blocking mode)
fn read(&mut self, buf: &mut [u8]) -> Result<usize, Error> {}
// Write from `buffer` (supports non-blocking mode)
fn write(&mut self, buf: &[u8]) -> Result<usize, Error> {}
```

Command-Line Arguments

Analogous to Rust's Standard Library

```Rust
let first_arg: &str = m3::env::args().skip(1).next().unwrap();
```

Look at the generated documentation for type information.

Assignment #3: Read a file

- **Write a program** that reads the contents of a given file
- Print the contents to stdout (`m3::println`)
- Modify `src/apps/rdfile`

Assignment #3: Read a file

- **Write a program** that reads the contents of a given file
- Print the contents to stdout (`m3::println`)
- Modify `src/apps/rdfile`

- You can **test the program** in the shell via `boot/shell.xml`
- All programs are already on the “PATH”.
- Use `ls` in the shell to find a suitable file
- You can exit the shell via `Ctrl+D` (takes a while)
- Or write a boot script with a hard-coded path

Assignment #3: Read a file

- **Write a program** that reads the contents of a given file
- Print the contents to stdout (`m3::println`)
- Modify `src/apps/rdfile`

- You can **test the program** in the shell via `boot/shell.xml`
- All programs are already on the “PATH”.
- Use `ls` in the shell to find a suitable file
- You can exit the shell via `Ctrl+D` (takes a while)
- Or write a boot script with a hard-coded path

- **Advanced version**: use the Rust standard library
- See `src/apps/ruststdtest` for an example

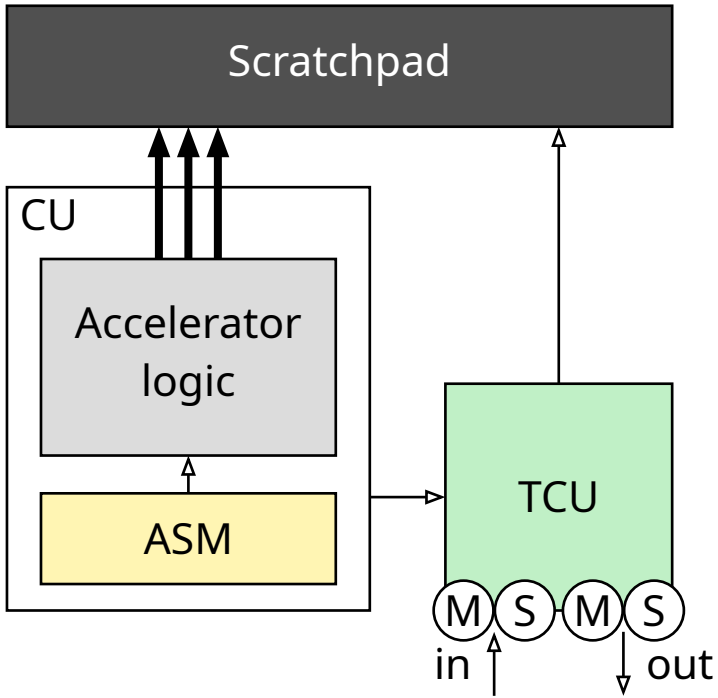
- Introduction
- **Assignments**
  - Hello World
  - IPC
  - Filesystem Service
  - **Accelerators**
  - Sessions

Accelerator Tile

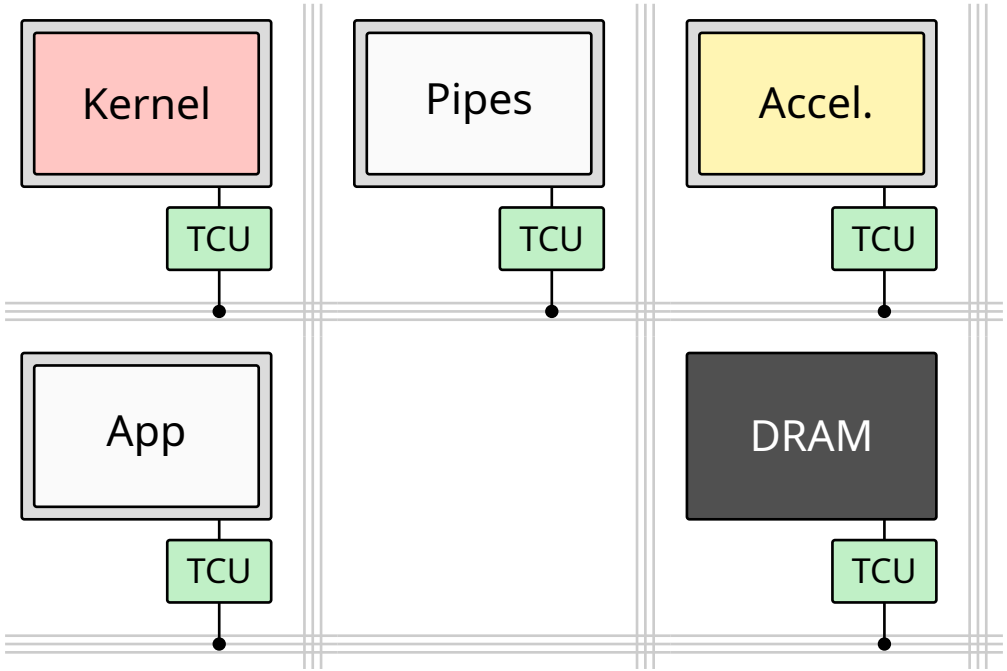
- Accelerators can sit on their own tiles
- Activities on other tiles **stream data** in/out
- Communication via **file protocol**
- We use the ***pipes* service** to offload buffer & notification management

Accelerator Tile

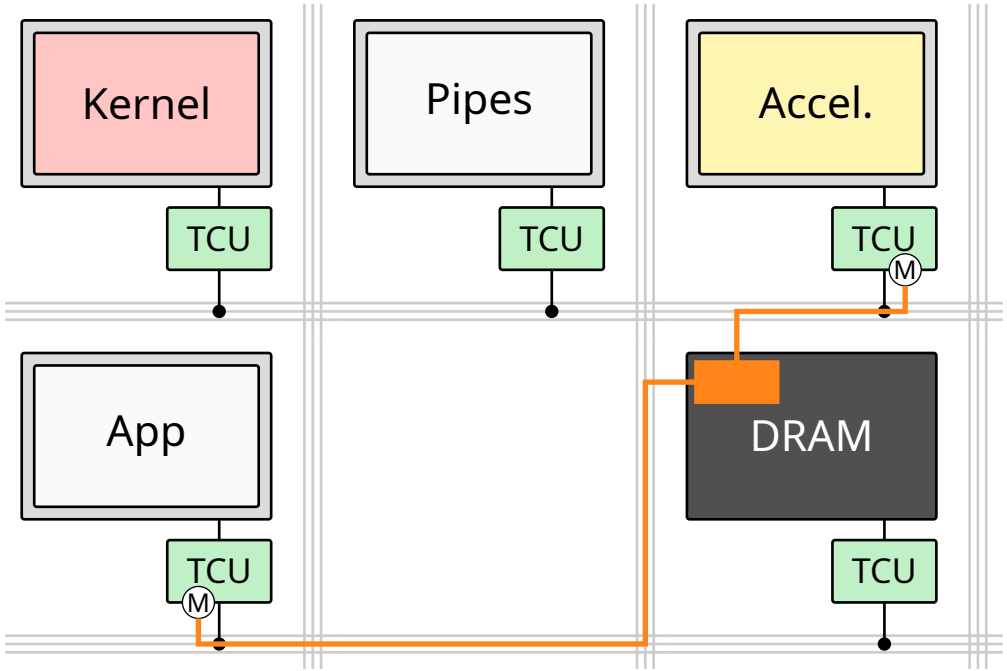
- Accelerators can sit on their own tiles
- Activities on other tiles **stream data** in/out
- Communication via **file protocol**
- We use the ***pipes* service** to offload buffer & notification management



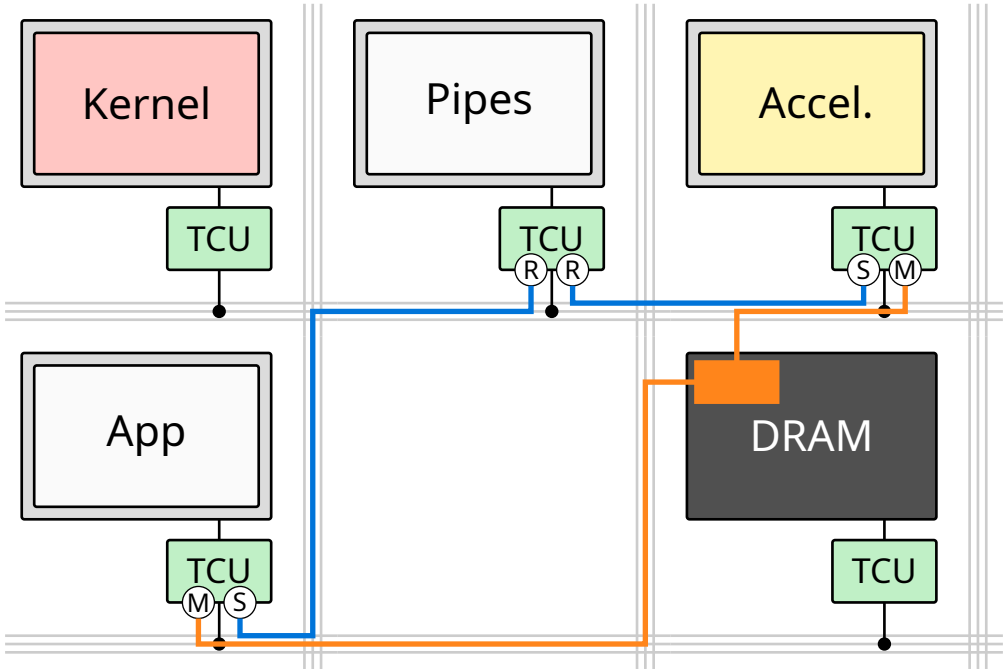
System Setup With Accelerators



System Setup With Accelerators



System Setup With Accelerators



Boot Script Example (Simplified)

XML

```xml
<!-- … -->
<app args="pipes" daemon="1">
  <serv name="pipes"/>
</app>
<!-- … -->
<app args="app">
  <sess name="pipes"/>
  <!-- Give the app control over the accelerator tile: -->
  <tiles type="accel"/>
</app>
<!-- … -->
```

Required Rust Bits

Abstractions

`Pipes`, `IndirectPipe`, `Tile`, `ChildActivity`, `StreamAccel`

And of course the file abstractions you already know.

Required Rust Bits

Abstractions

`Pipes`, `IndirectPipe`, `Tile`, `ChildActivity`, `StreamAccel`

And of course the file abstractions you already know.

Rust Slices

```Rust
let input_slice = &input[from_this_index..];
let output_slice = &mut output[from_this_index..];
```

Assignment #4: Use an accelerator

- Write a program that uses the *rot13* accelerator
- Use the program arguments as input
- Print the accelerator output to stdout
- Modify `boot/acceldemo.xml` and `src/apps/acceldemo`
- Use M^3 's Rust documentation for learning how to use the abstractions

- Introduction
- **Assignments**
  - Hello World
  - IPC
  - Filesystem Service
  - Accelerators
  - **Sessions**

L4-style microkernels:
Delegate access rights to memory, files, etc. via IPC

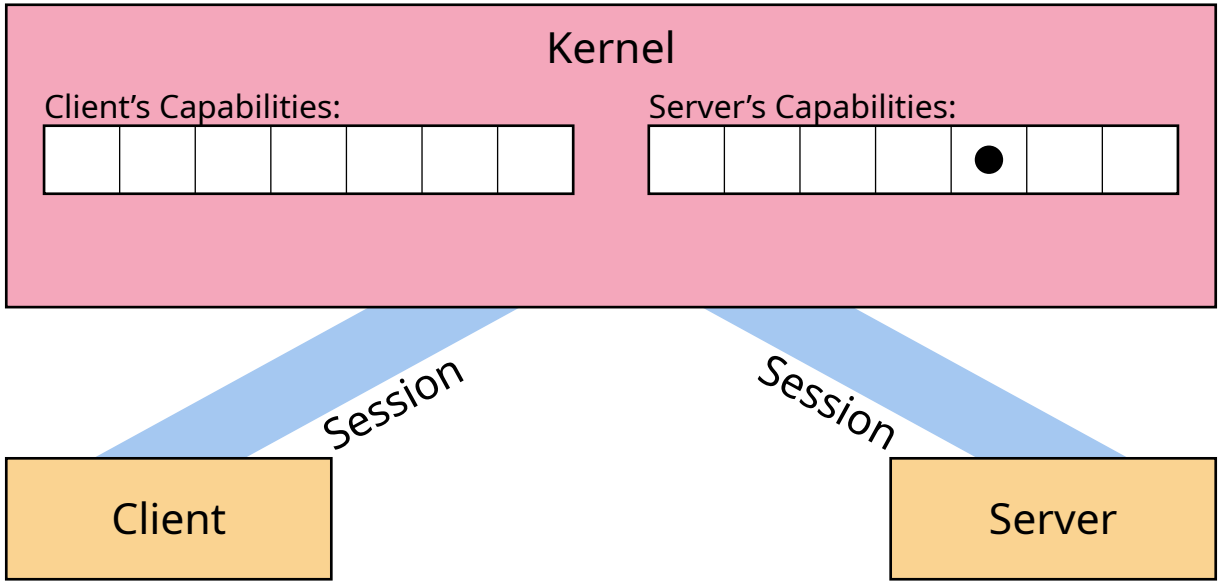
L4-style microkernels:
Delegate access rights to memory, files, etc. via IPC

M³ ’s IPC bypasses the kernel …

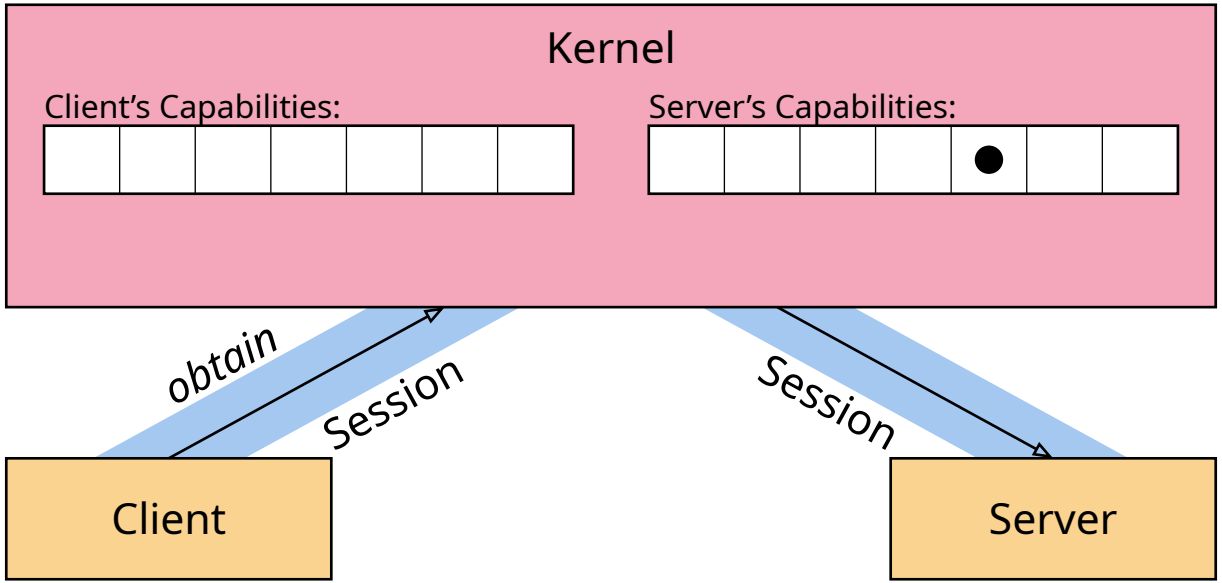
L4-style microkernels:
Delegate access rights to memory, files, etc. via IPC

M³ ’s IPC bypasses the kernel …
How to exchange capabilities?

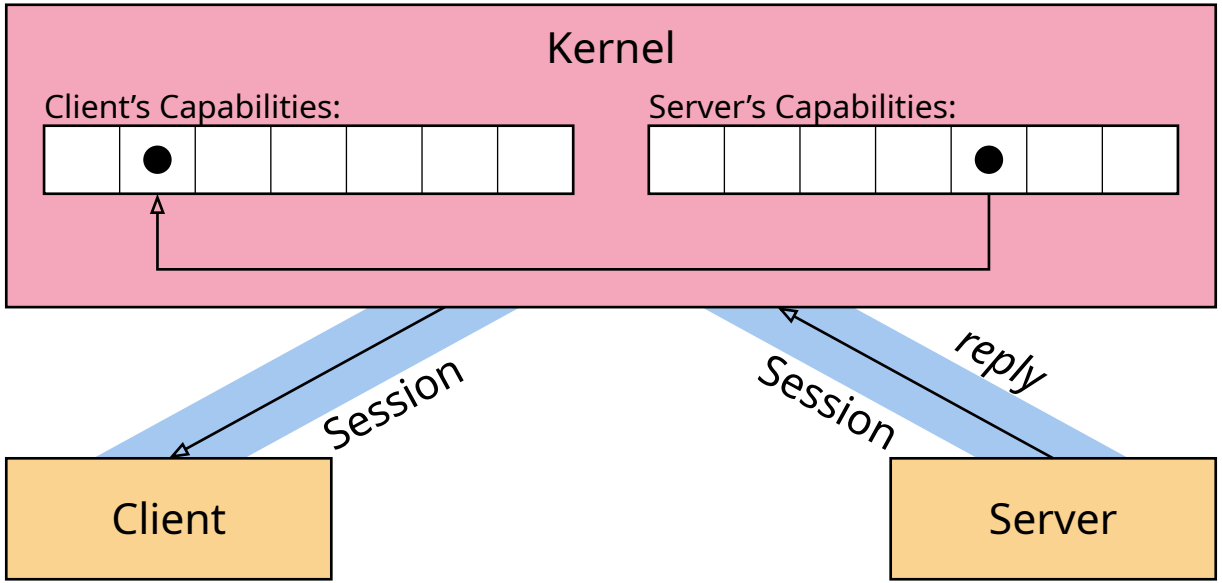
Servers and Sessions via the Kernel



Servers and Sessions via the Kernel



Servers and Sessions via the Kernel



Client Side of Obtain

Rust

```
let crd = session.obtain(
  1, // Number of capabilities to obtain
  |os| { // Prepare arguments for the server
    // First argument is the opcode:
    os.push(42usize);
    os.push("my argument");
  },
  |is| { // Handle reply from the server
    let some_reply_arg = is.pop::<bool>();
    Ok(())
  },
)
```

Server Side of Obtain (Simplified)

```rust
// Register the handler my_handler for opcode 42:
hdl.reg_cap_handler(42usize, ExcType::Obt(1), Session::my_handler);
```

Server Side of Obtain (Simplified)

```rust
// Register the handler my_handler for opcode 42:
hdl.reg_cap_handler(42usize, ExcType::Obt(1), Session::my_handler);
```

```rust
impl Session {
  fn my_handler(xchg: &mut CapExchange<'_>) {
    /* … */
    // Instruct the kernel to transfer the capability my_cap:
    xchg.out_caps(CapRngDesc::new_single(
      CapType::Object,
      my_cap.sel(),
    ));
  }
}
```

Assignment #5: Servers and sessions

- Server allocates memory capability containing message
- The server provides access to this memory **capability via sessions**
- The **client obtains** copy of capability via a session
- The client prints the message stored behind the memory capability
- Modify `src/apps/sessions`
- **Fill in** the handler registration, capability exchange, and obtain logic

Conclusion

- M³ is available as **open source** (both hardware and software): https://github.com/Barkhausen-Institut/M3

- `README.md` explains how to set it up

Conclusion

- M³ is available as **open source** (both hardware and software): https://github.com/Barkhausen-Institut/M3

- `README.md` explains how to set it up
- **If you think M³ is interesting, contact us for thesis topics!** nils.asmussen or viktor.reusch @barkhauseninstitut.org