TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science, Institute for System Architecture, Operating Systems Group

EXERCISE 2: IPC

Inter-Process Communication

Nils Asmussen

Dresden, 11/13/2018

Announcement

- Starting next week, we'll switch exercise and lecture
- Lecture will be at 2:50 PM
- Exercise will be at 4:40 PM

Recap

- Inter-Process Communication
  - Send
  - Wait (open/closed)
  - Call: Send + closed Wait
  - Reply and Wait
- Message Payload
  - Plain data : copy
  - Capabilities (memory, kernel objects) : map
- Sync / Async
  - Sync: Rendezvous, direct copy sender → receiver
  - Async: Queues, buffers, fire and forget

The Plan

- Echo server a.k.a. Log server
  - Send a string to a local echo server
  - Return the message to the caller
- Capability Delegation
  - Create your own echo server (Endpoint + Thread) and send a capability for this server to another thread
  - Use your new echo server for logging
- Client-Server
  - Connection and Session management, server registers a service, client looks it up

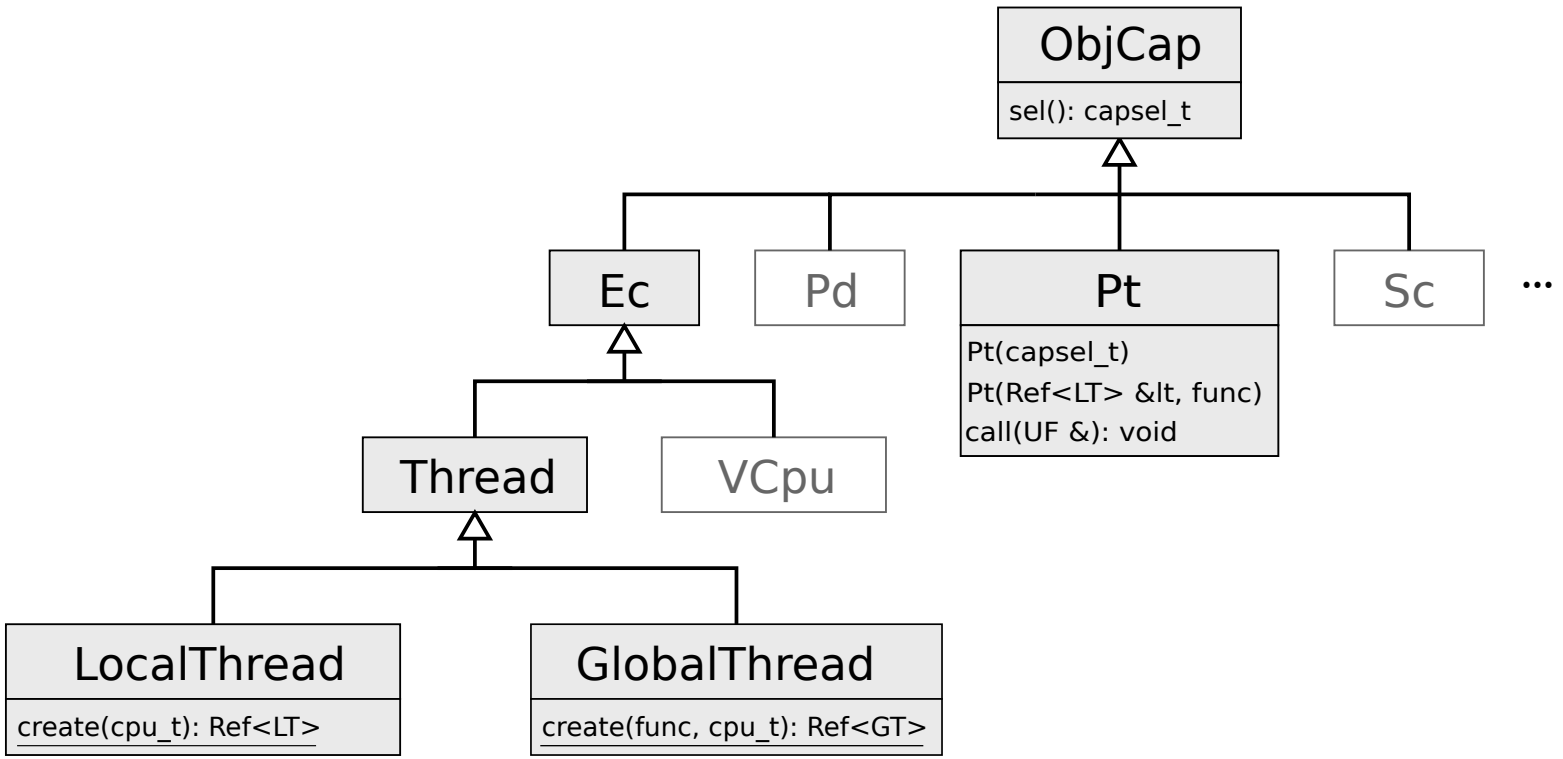
Setup

- Download the source archive from `http://os.inf.tu-dresden.de/~nils/nova-nre.tar.gz`
- `tar xfz nova-nre.tar.gz`
- `cd cross && ./download.sh x86_64` (XOR `./build.sh`)
- Choose host platform (Ubuntu 18.04 x86_64)
- `cd ../nre` (your working directory)
- Build via: `./b`
- Build and run via: `./b qemu boot/echo`
- `<DIR>/nre/apps/ipc`: echo, delegate, client, server
- `<DIR>/nre/include`: headers, e.g. UtcbFrame.h

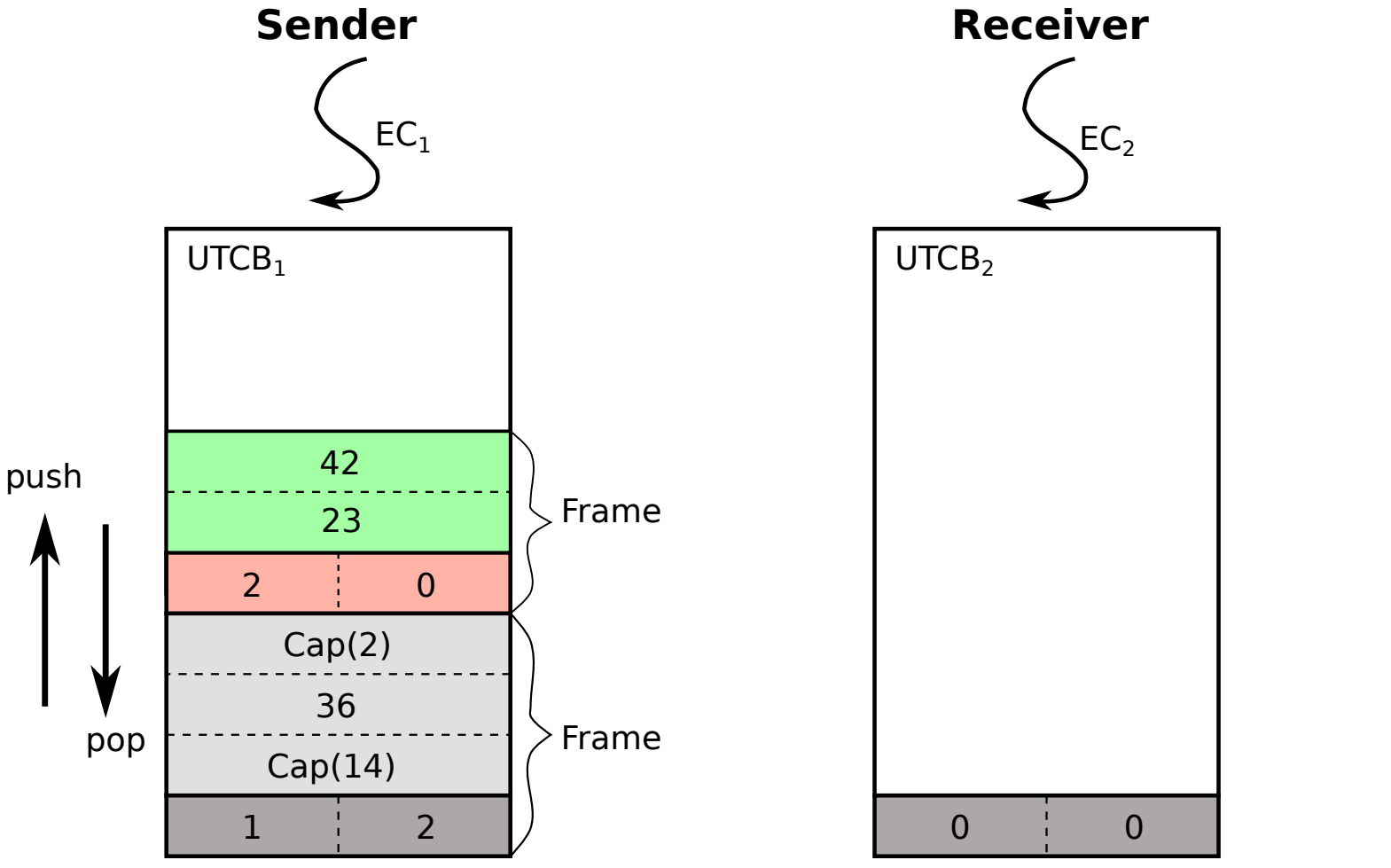
NOVA Abstractions

- Thread / Execution Context (EC)
  - Unit of execution, thread or virtual machine
- Portal (PT)
  - Communication endpoint
  - Messages (data, capabilities) are sent to portals
  - Bound thread receives the messages
- Local Threads (services)
  - Wait/block in portals until get invoked/called
  - Return to the portal after request completion
- Global Threads
  - Run on their own

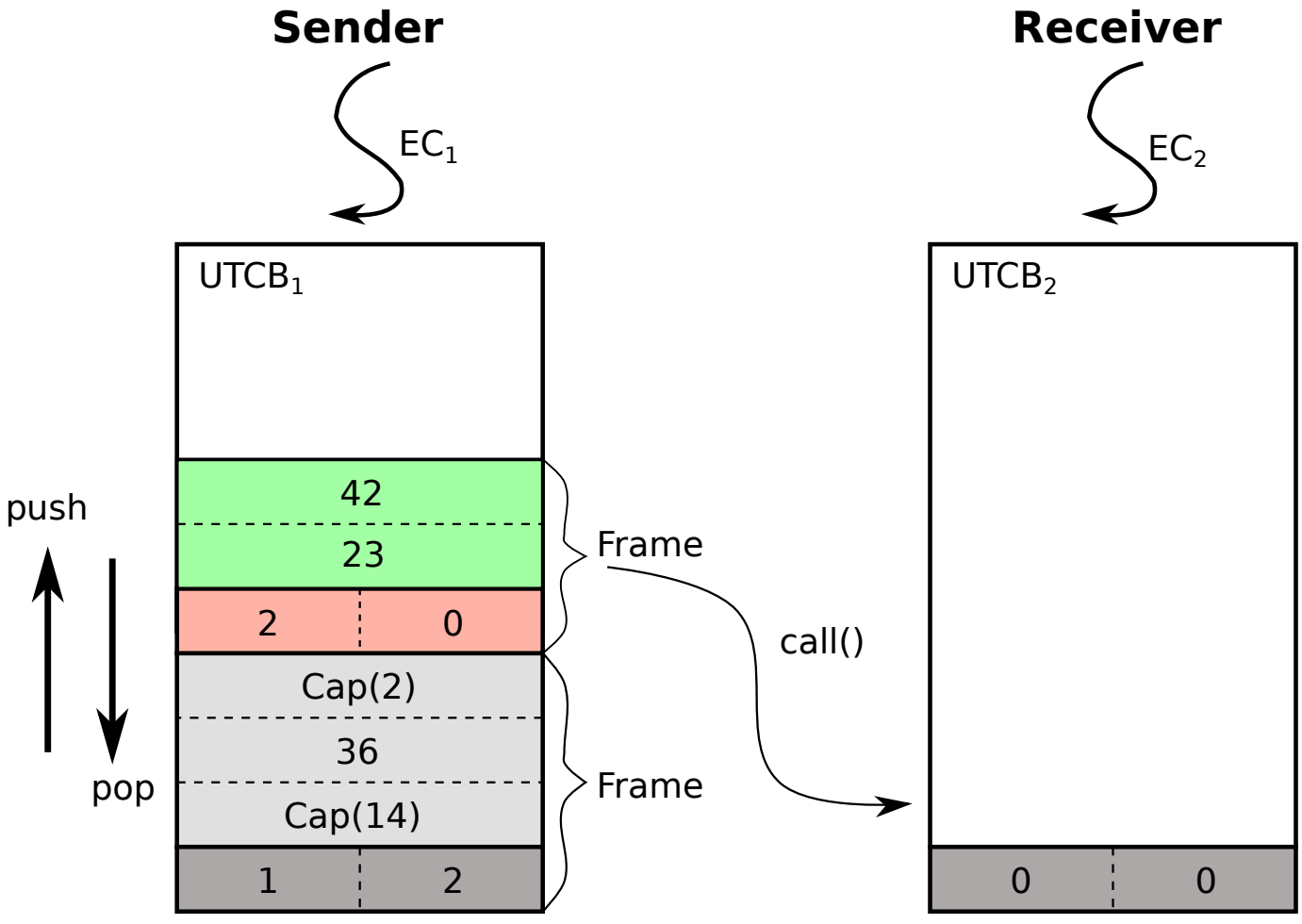
NRE Capability Hierarchy



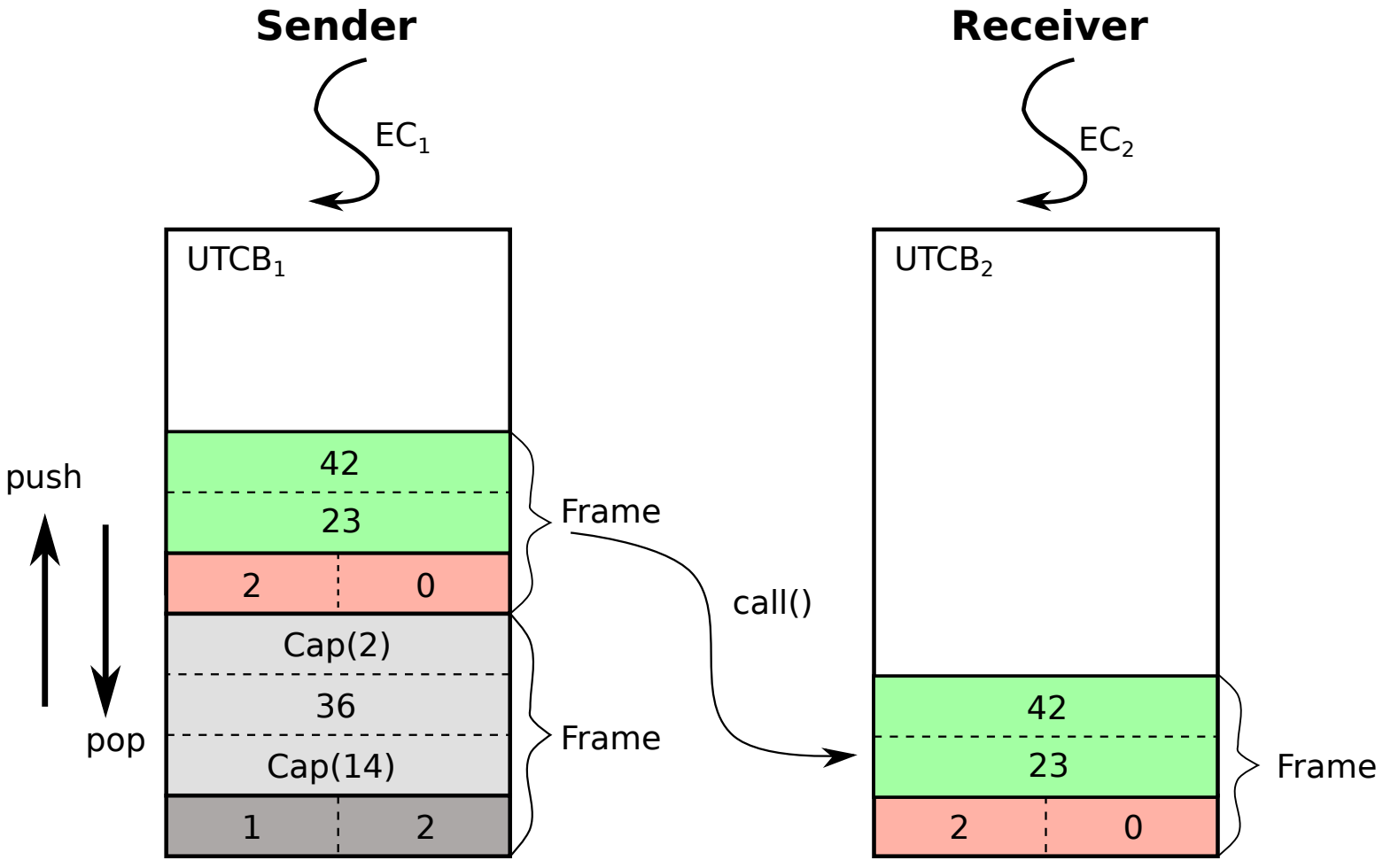
NRE UTCB Frames



NRE UTCB Frames



NRE UTCB Frames



NRE UTCB Usage

- Data accessed via stream operators:
  - read: `UtcbFrame >> value;`
  - write: `UtcbFrame << value;`
- `UtcbFrame::finish_input()`
  - Only required when reading AND writing
  - Needs to be done if reading is finished
- `UtcbFrame` constructor pushes a new frame on the UTCB, the destructor pops it again

Usage example

```
UtcbFrameRef uf;     // UTCB frame
uf >> x;             // read argument
uf.finish_input();   // end of input
uf << 2 * x;         // write result back
```

Service "Loop"

Fiasco.OC/L4Re Style

```
for(; receive(&utcb); reply(utcb)) {
  process_message(utcb);
  generate_reply(&utcb);
}
```

- Receive: waits for incoming message which is copied into receiver's UTCB
- Handle message and generate reply
- Reply: copy reply into caller's UTCB and await next message

Service "Loop"

NOVA/NRE Style

```
PORTAL void portal_function() {
  process_message(utcb);
  generate_reply(&utcb);
}
```

- Receive: waits for incoming message which is copied into receiver's UTCB
- Handle message and generate reply
- Reply: copy reply into caller's UTCB and await next message

1. Echo server (apps/echo)

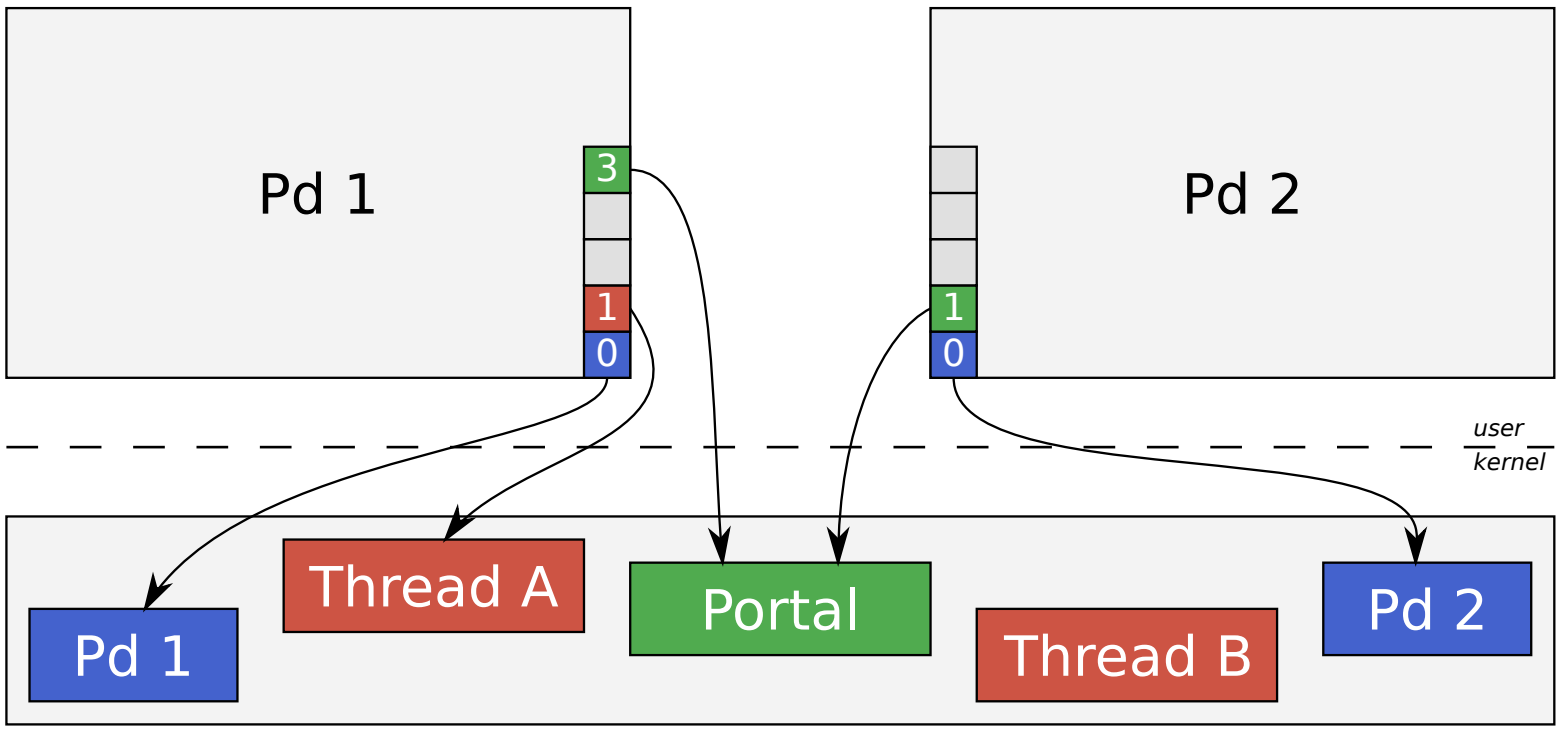
```
// create a new thread
Reference<LocalThread> lt = LocalThread::create(
  CPU::current().log_id());
// create a new portal for this thread
// Pt::Pt(const Reference<LocalThread>&, func_ptr);
Pt echo(lt, portal_echo);

{
  UtcbFrame uf;    // new UTCB frame
  uf << 42;        // fill in argument
  echo.call(uf);   // call server
  int res;
  uf >> res;       // read result
}
```

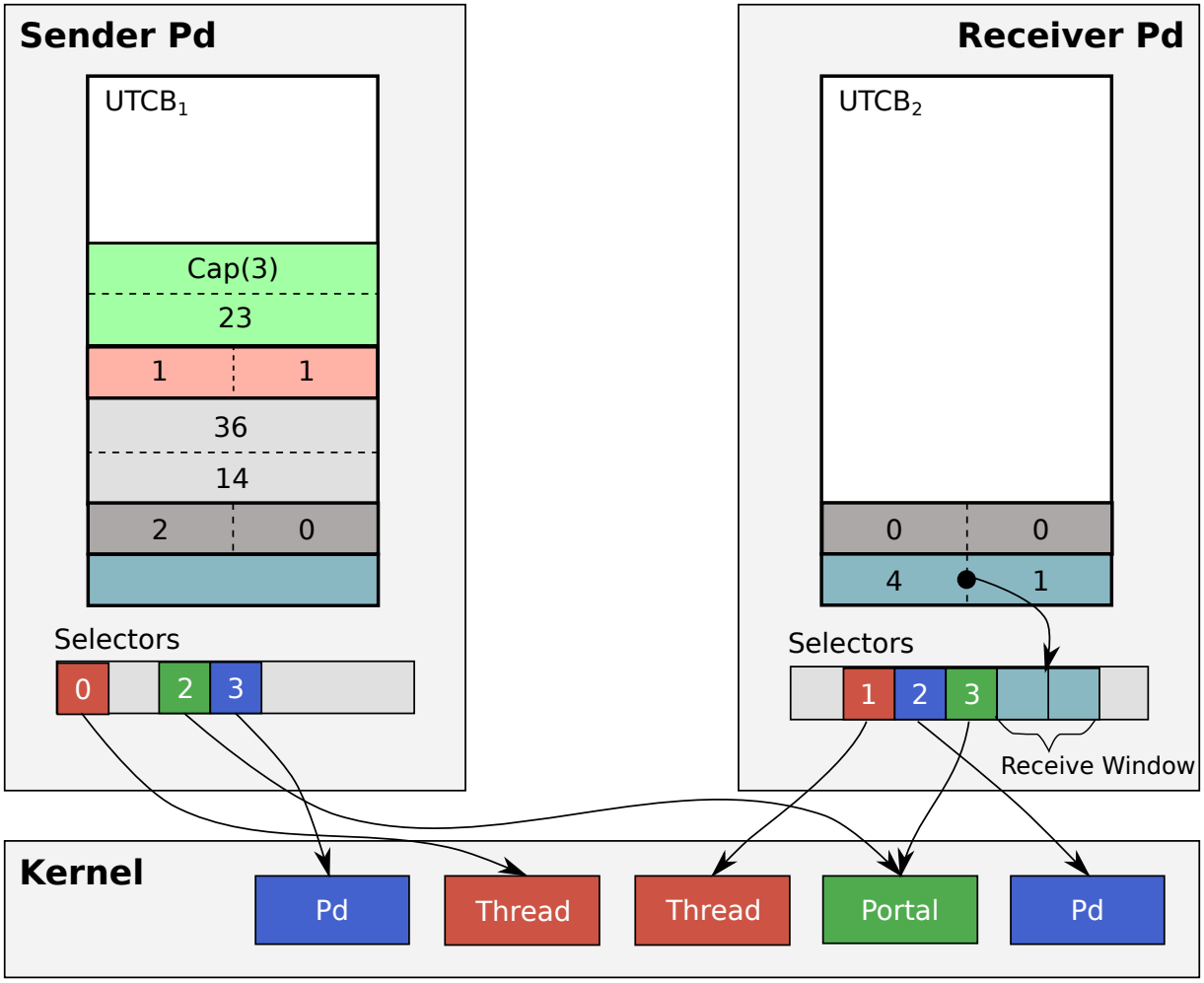
1. Echo server (apps/echo)

- Implement service functionality
- Read argument, send it back to the caller
- Improvement: send two numbers a and b, send back a+b / a-b / a*b / a/b ...

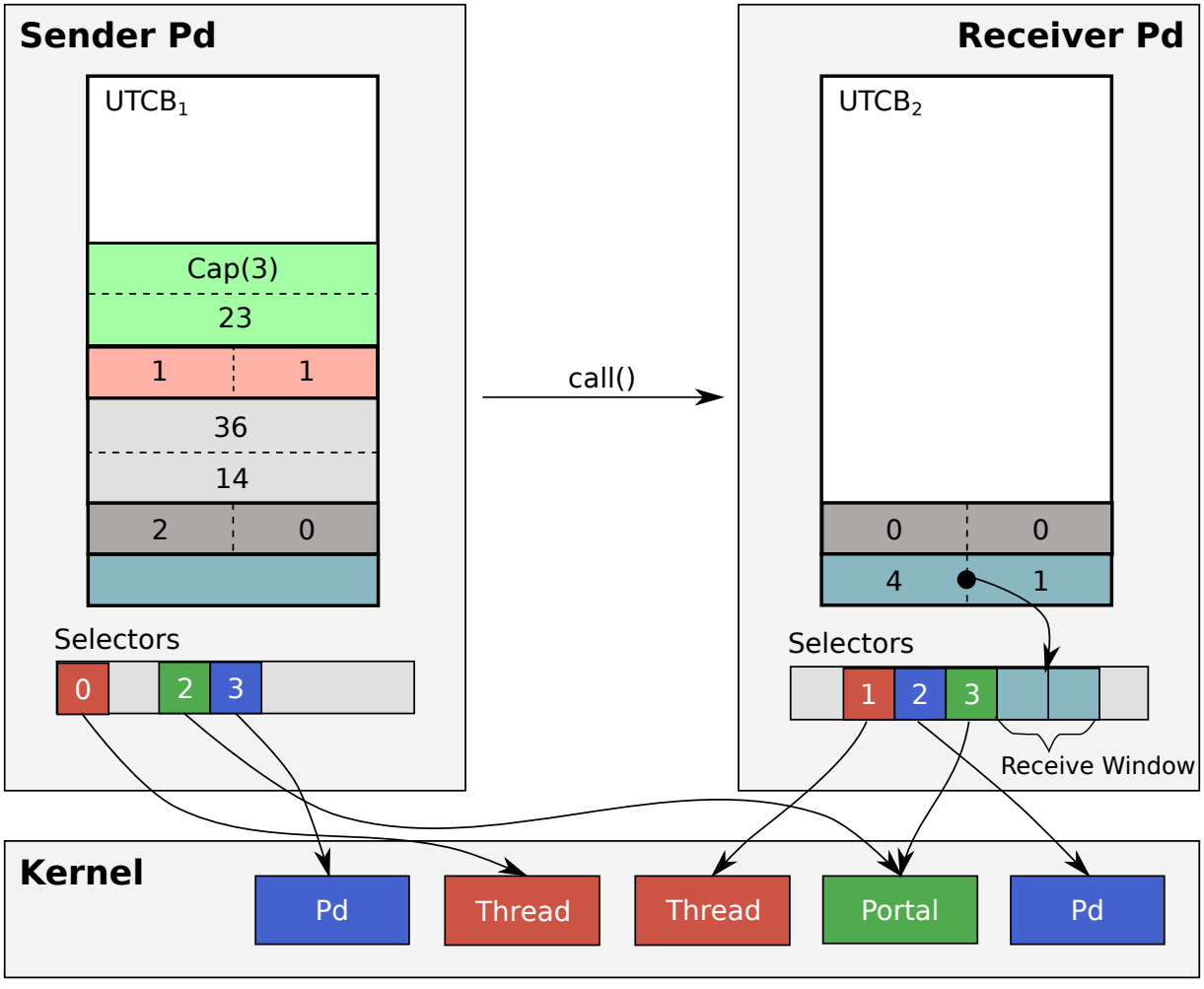
Capability Space



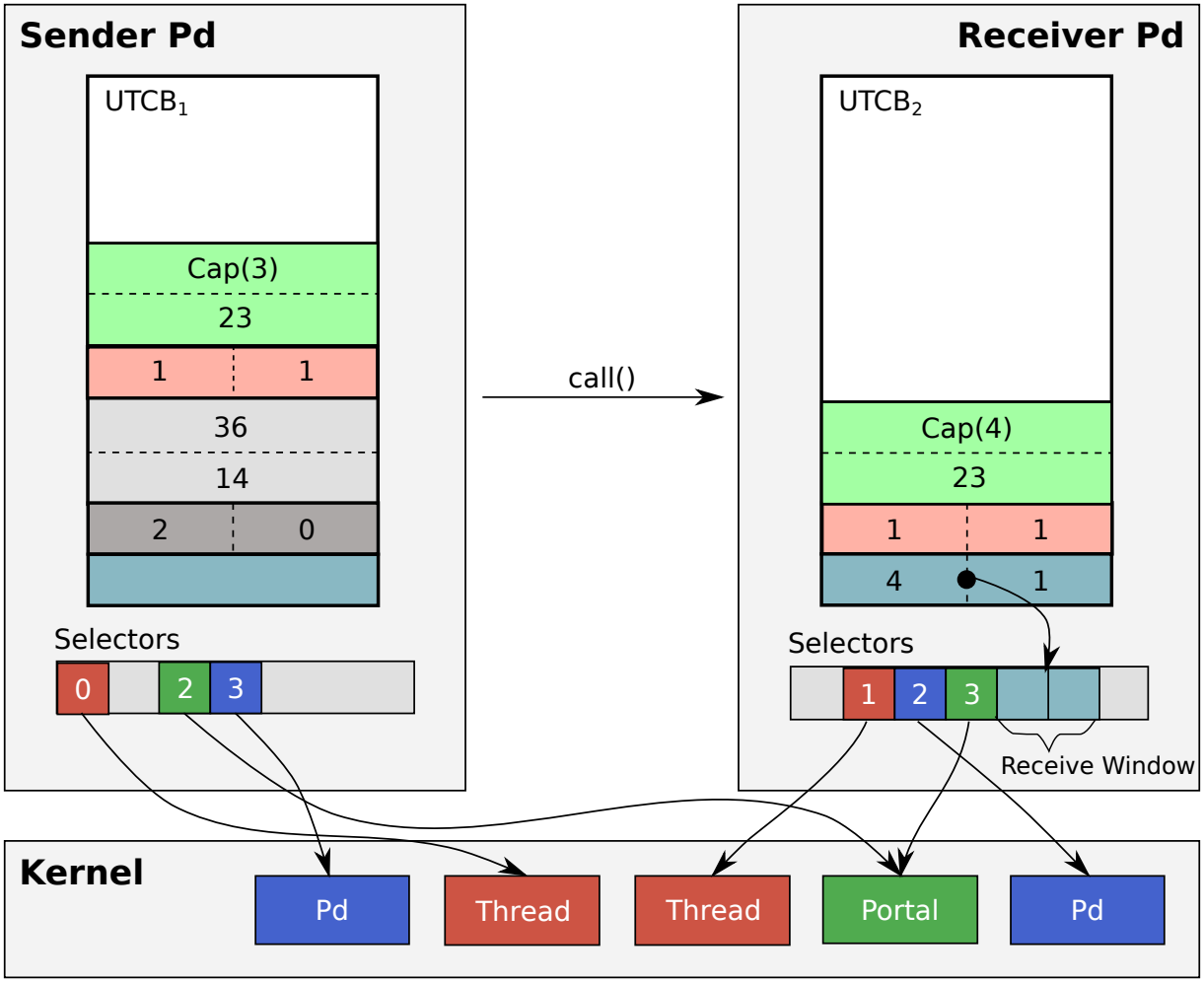
Capability Delegation over IPC



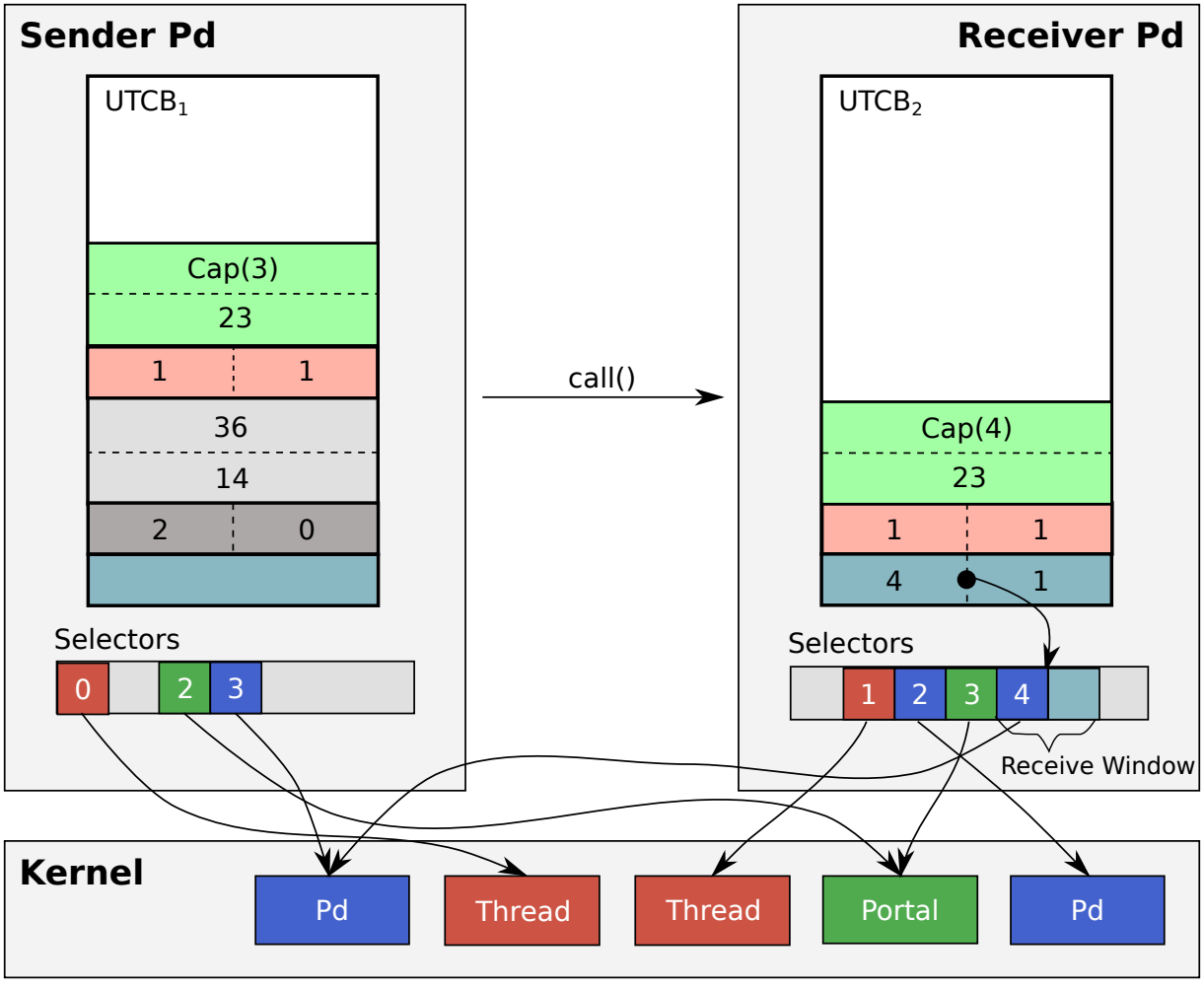
Capability Delegation over IPC



Capability Delegation over IPC



Capability Delegation over IPC



Capability Range Descriptor (Crd)

- Capabilities exchanged via:
  - send: `UtcbFrame::delegate(cap);`
  - receive: `UtcbFrame::delegation_window(Crd);`
- Denote caps via Crd: type, offset and size
- power-of-two sized, size-aligned, e.g:
  - Offset 0, size = 1 → cap index 0
  - Offset 0, size = 2 → cap index 0-1
  - Offset 8, size = 4 → cap index 8-11
  - Offset 16, size = 32 → illegal, misaligned

2. Delegation (apps/delegate)

```
// allocate/reserve num cap space indices
capsel_t CapSelSpace::get().allocate(num);

// open receive window for new caps
// see include/Desc.h for Crd details
Crd crd(start, order, Crd::OBJ_ALL);
UtcbFrame::delegation_window (crd);

// cap delegation: add cap to UTCB for sending
UtcbFrame::delegate(capsel_t sel);

// invoke portal, thereby receiving cap(s)
Pt::call(UtcbFrame&);

// create portal, bound to <sel>
Pt::Pt(capsel_t sel);
```

3. Client/Server

```
// SERVER
PORTAL static void echo_func(void*) {
  // ... service implementation
}
int main() {
  Service *s = new Service("echo",
    CPUSet(CPUSet::ALL), echo_func);
  s->start();
}

// CLIENT
int main() {
  PtClientSession sess("echo");
  UtcbFrame uf;
  sess.pt().call(uf);
}
```