HelenOS IPC and Behavior Protocols

Martin Děcký

DISTRIBUTED SYSTEMS RESEARCH GROUP
http://dsrg.mff.cuni.cz/

CHARLES UNIVERSITY IN PRAGUE
FACULTY OF MATHEMATICS AND PHYSICS

Motivation

- HelenOS components[1] use complicated communication protocols
  - Asynchronous messages
  - Forwarding of messages
  - Dynamic connection establishment
  - Memory sharing
- What is the most suitable formalism for ensuring communication correctness?
  - Static checking, run-time checking

[1] No formal definition of components yet, just separate tasks.

Components

- Unformal
  - Component = task
    - Physically flat, but logically nested (forwarding)
  - Provided interface = accepted IPC methods
    - Implicit
  - Required interface = called IPC methods
    - Implicit
  - Binding = IPC connection
    - dynamic

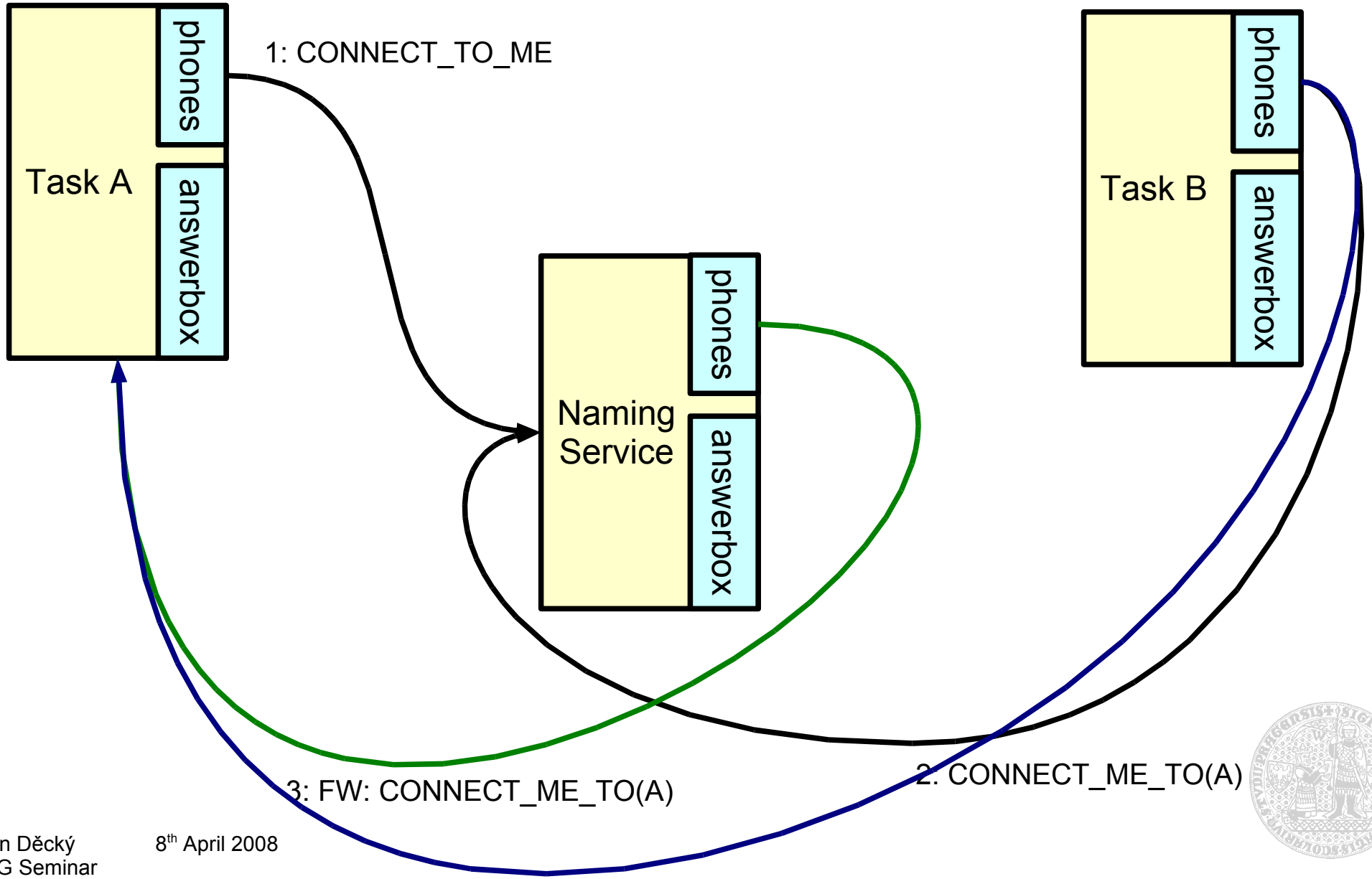
HelenOS IPC

- Key features
  - Simple kernel interface
    - Direct payload, sharing address space areas, copying memory
    - Mandatory answers, forwarding of messages
    - Bounded buffers
    - Suitable for interrupt notifications
  - Standard user space interface
    - Synchronous and asynchronous variants
    - Fast and slow variants
    - Blocking and two-way semantics (except interrupts)

HelenOS IPC

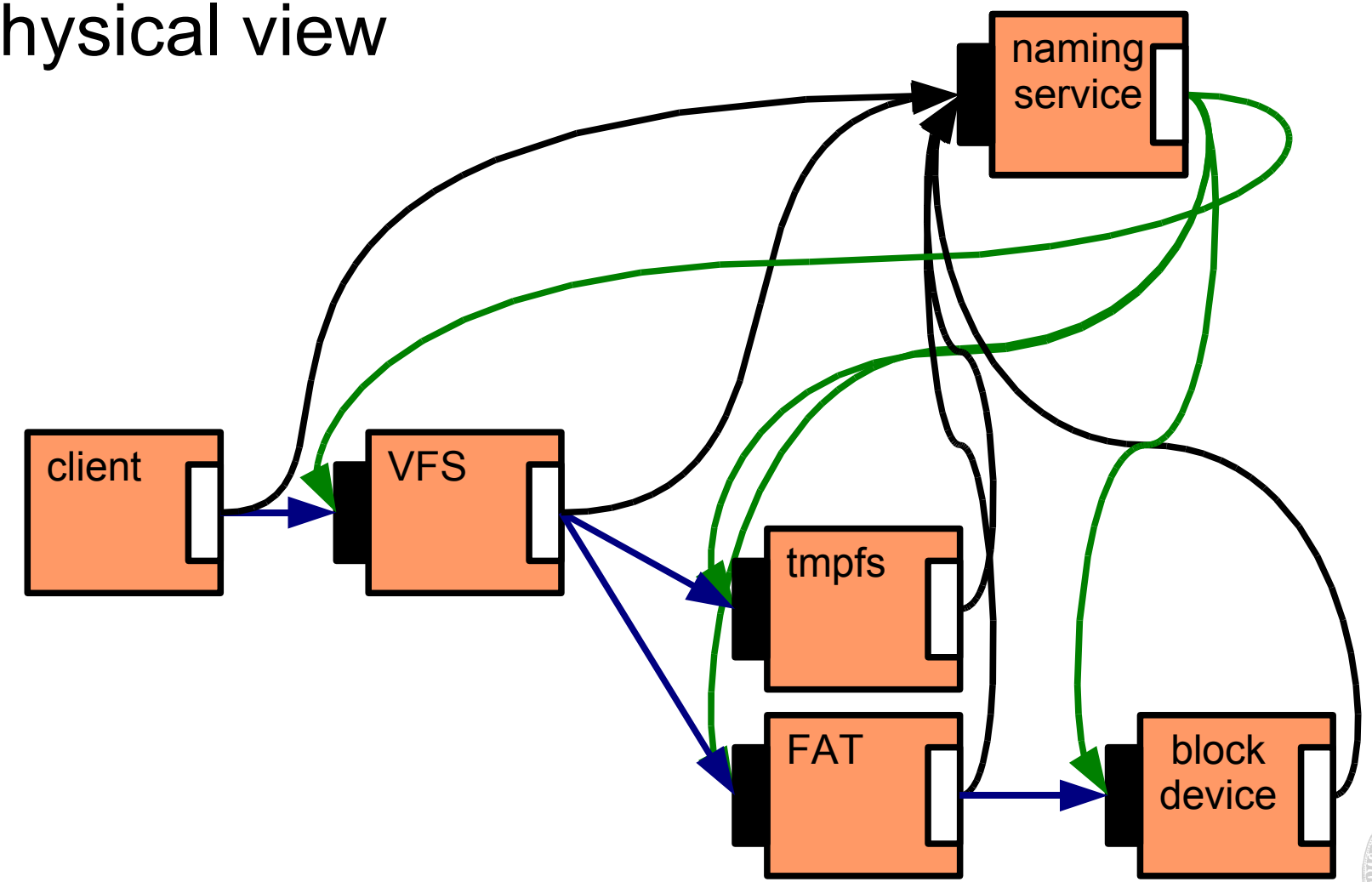
- Key features
  - Advanced interface
    - Extensive use of fibrils
      - User space managed pseudo threads
      - Cooperative context switching
    - Avoiding classical message loop
    - Avoiding (most) synchronization issues in multithreaded components
    - More straightforward way of programming

HelenOS IPC overview



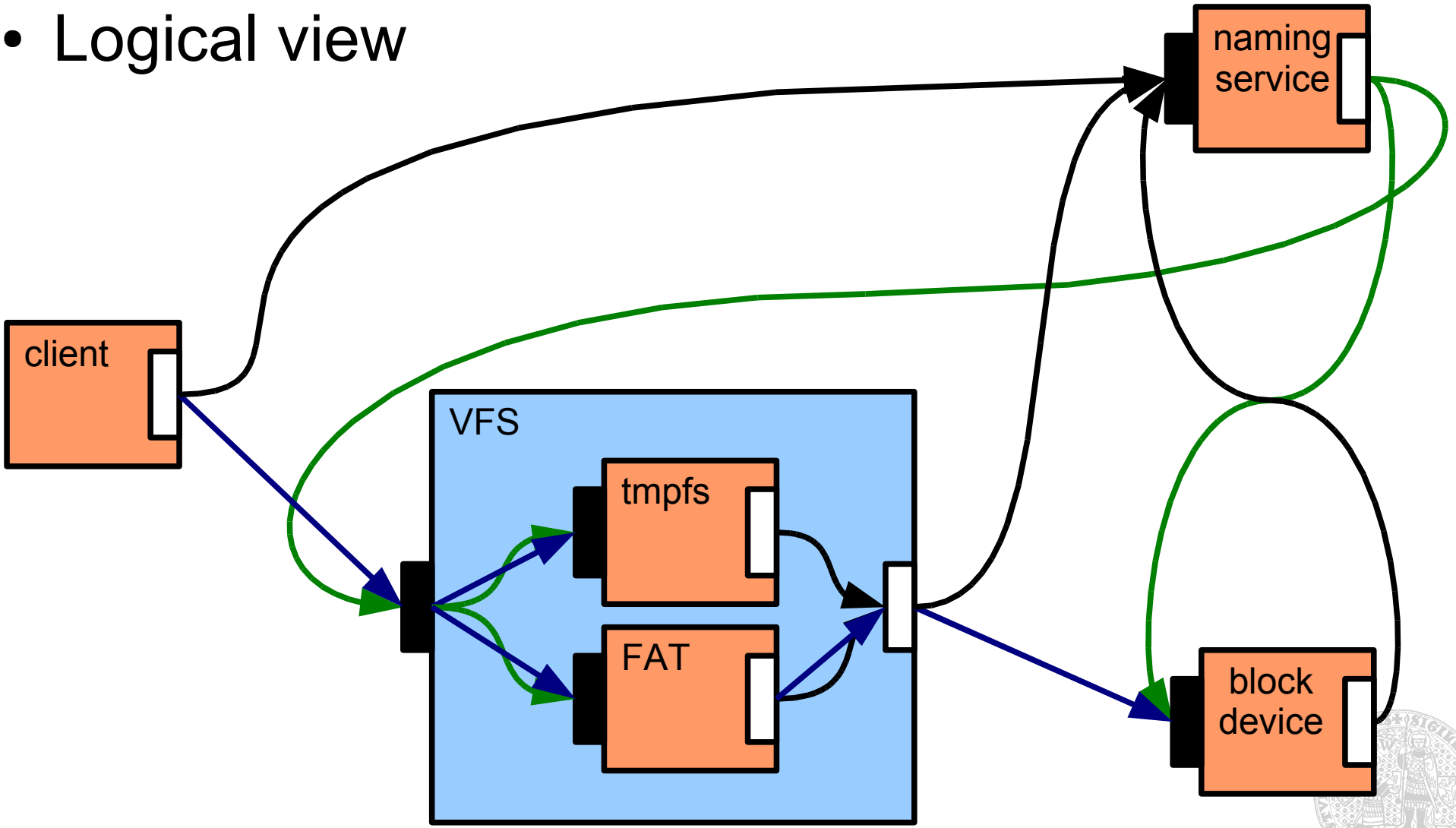
Components

- Physical view



Components

- Logical view



Kernel IPC layer

- Synchronous messages
  - `SYS_IPC_CALL_SYNC_FAST`
    `SYS_IPC_CALL_SYNC_SLOW`
    - Caller is blocked for answer
- Asynchronous messages
  - `SYS_IPC_CALL_ASYNC_FAST`
    `SYS_IPC_CALL_ASYNC_SLOW`
    - Kernel has fixed buffer for delivered messages
  - `SYS_IPC_ANSWER_FAST`
    `SYS_IPC_ANSWER_SLOW`
    - Answer and delivered message

Kernel IPC layer

  - SYS_IPC_FORWARD_FAST
    - Forward delivered message
- Receiving messages
  - SYS_IPC_WAIT
    - Wait for a message or answer
- Close phone
  - SYS_IPC_HANGUP

Standard user interface

- Synchronous calls
  - Trivial syscall wrappers
- Asynchronous calls
  - If kernel buffer is full ➔ block and resend later
    - In case of interrupt notification ➔ discard
- New connections
  - `IPC_M_CONNECT_TO_ME`
    - Callback connection
  - `IPC_M_CONNECT_ME_TO`

Standard user interface

- Sharing address space areas
  - `IPC_M_SHARE_IN`
    - Request sharing from party
  - `IPC_M_SHARE_OUT`
    - Send address space area
- Copying data
  - `IPC_M_DATA_READ`
  - `IPC_M_DATA_WRITE`

Message loop

```
while (true) {
    callid = ipc_wait_for_call(&call);
    switch (IPC_GET_METHOD(call)) {
    case IPC_M_SHARE_IN:
        switch (IPC_GET_ARG3(CALL)) {
        case SERVICE_CLOCK:
            ipc_answer_2(callid, EOK, (ipcarg_t) *addr, AS_AREA_READ);
            break;
        default:
            ipc_answer_0(callid, ENOENT);
        }
        continue;
    default:
        ipc_answer_0(callid, ENOENT);
    }
}
```

Fibrils

- User space managed cooperative threads
  - Each kernel-managed thread contains one
  - Fibril preemption
    - Explicit context switch: fibril_switch()
    - In certain functions of advanced IPC interface
  - Autospawning of manager fibrils
    - Waiting for incoming messages or answers
    - Switching to fibrils which can either process a message or is waiting for an answer
  - Synchronization and atomicity
    - Serialization of messages (disabling fibril preemption)

Multithreaded server

```
void client_connection(ipc_callid_t callid, ipc_call_t *call) {
    // Accept or refuse message
    if (i_want_to_refuse) {
        ipc_answer_0(icallid, ELIMIT);
        return;
    }
    ipc_answer_0(icallid, EOK);

    // Get next message in the protocol
    callid = async_get_call(&call);
    handle_call(callid, call);
    ipc_answer_2(callid, 2, 4, 6);

    // Get next message in the protocol
    callid = async_get_call(&call);
    ...
}

int main() {
    async_manager();
}
```

Using Behavior Protocols

- Explicit definition of interface
  - Invocation ➔ required method on party interface
  - Receiving ➔ provided method on own interface
- Run-time behavior checking
  - Checking of communication protocol
  - Major problem
    - `?open + ?read + ?write + ?close`
    - Entities/sessions as suggested by Ondřej Šerý

Q&A