Kernel Support for User-Level Threads on L4

Diplomverteidigung

Dietrich Clauß

dc2@os.inf.tu-dresden.de

Operating Systems Group

Institute for System Architecture

TU Dresden

Outline

- Motivation
- Design Goals
- Achievements
- Details
  - Activation Messages
  - Deschedule Blocking Threads
- Conclusions
- Outlook

Motivation

Multiple Concurrent Activities – Threads:

- independent execution
- efficient communication

Motivation

Multiple Concurrent Activities – Threads:

- independent execution
- efficient communication

Kernel-Level Threads:

- Preemptive Scheduling
- Blocking Operating-System Services

User-Level Threads:

- User-Controlled Per-Thread Resources
- Efficient Communication and Thread Management

Motivation (2)

Combine the Benefits

- Optimize kernel-level threads, or
- enhance user-level threads.

Enhance User-Level Threads

- First-Class User-Level Threads
- Add Kernel Support
- Use Kernel Thread(s) as Virtual (Multi-)Processor

Design Goals

- High Performance for User-Level Threads
- User-Level Threads on Pageable User-Controlled Resources
- Preserve Performance of Kernel-Level Threads
- Preserve Minimality of the Kernel
- L4 API Compatibility: Relaxed
  - for applications using user-level threads
  - for legacy applications

Achievements

Which Functionality is Needed?

- Running Threads
  - forward progress
  - cooperative thread switch
  - preemptive thread switch
- Blocking Threads: IPC
  - atomic send-to-receive switch
  - zero timeout
  - fairness on open wait

Achievements

Which Functionality is Needed?

- Running Threads
  - forward progress
  - cooperative thread switch
  - preemptive thread switch
- Blocking Threads: IPC
  - atomic send-to-receive switch
  - zero timeout
  - fairness on open wait

What is needed for user-level threads?

Achievements

Which Functionality is Needed?	for UL Threads
- Running Threads	
- forward progress	yes
- cooperative thread switch	efficient
- preemptive thread switch	possible
- Blocking Threads: IPC	
- atomic send-to-receive switch	
- zero timeout	
- fairness on open wait	

What is needed for user-level threads?

Achievements

Which Functionality is Needed?	for UL Threads
- Running Threads	
- forward progress	yes
- cooperative thread switch	efficient
- preemptive thread switch	possible
- Blocking Threads: IPC	Global or Local
- atomic send-to-receive switch	relaxed
- zero timeout	relaxed
- fairness on open wait	yes

What is needed for user-level threads?

Achievements (2)

Per-Thread Data in L4:

- Thread Control Block (TCB)
- Kernel Memory

Achievements (2)

Per-Thread Data in L4:

- Thread Control Block (TCB)
- Kernel Memory

Moving Per-Thread Data to User Level:

- data becomes untrusted
- user-level dispatcher code

Achievements (2)

Per-Thread Data in L4:

- Thread Control Block (TCB)
- Kernel Memory

Moving Per-Thread Data to User Level:

- data becomes untrusted
- user-level dispatcher code

Interaction Between Kernel and Dispatcher:

- on preemptive thread switch, and
- to deliver IPC messages

Achievements (3)

Dispatcher Code

Dedicated Dispatcher Thread:

- + no kernel extensions
- – costly preemptive thread switch
- – extra kernel thread

Dispatcher Code on Virtual Processor:

- + inexpensive preemptive thread switch
- + no extra kernel thread
- – requires kernel extensions

Achievements (4)

Occupied Virtual Processors

Achievements (4)

Occupied Virtual Processors

When?

- on blocking global communication
- on long-running system calls

Achievements (4)

Occupied Virtual Processors

When?

- on blocking global communication
- on long-running system calls

Issues:

- forward progress

Achievements (4)

Occupied Virtual Processors

When?

- on blocking global communication
- on long-running system calls

Issues:

- forward progress

Solutions:

- use multiple virtual processors
- deschedule the invoker

Outline

- Motivation
- Design Goals
- Achievements
- Details
  - Activation Messages
  - Deschedule Blocking Threads
- Conclusions
- Outlook

Activation Messages

Move Ready-to-Run Threads to Another VP

- moveable user-level threads
- local thread IDs
- blocking user threads $\leq$ number of VPs
- local communication at user level

Activation Messages

Move Ready-to-Run Threads to Another VP

- moveable user-level threads
- local thread IDs
- blocking user threads $\leq$ number of VPs
- local communication at user level
- notification about blocking/unblocking of VPs

Activation Messages

Activation Messages (2)

When to Send?

- on switch to receiver, or
- on timer interrupt, or
- on ready list enqueue/dequeue operations

How to Send?

- nested IPC, or
- extra sender role

Where to Send?

- to a virtual processor, or
- to a dedicated dispatcher thread

Activation Messages (2)

When to Send?

- on switch to receiver, or
- on timer interrupt, or
- on ready list enqueue/dequeue operations

performance

How to Send?

- nested IPC, or
- extra sender role

Where to Send?

- to a virtual processor, or
- to a dedicated dispatcher thread

Activation Messages (2)

When to Send?

- on switch to receiver, or — performance
- on timer interrupt, or — IPC chains, ex_regs
- on ready list enqueue/dequeue operations

How to Send?

- nested IPC, or
- extra sender role

Where to Send?

- to a virtual processor, or
- to a dedicated dispatcher thread

Activation Messages (2)

When to Send?

- on switch to receiver, or — performance
- on timer interrupt, or — IPC chains, ex_regs
- on ready list enqueue/dequeue operations

How to Send?

- nested IPC, or — influences regular IPC
- extra sender role

Where to Send?

- to a virtual processor, or
- to a dedicated dispatcher thread

Activation Messages (2)

When to Send?

- on switch to receiver, or — performance
- on timer interrupt, or — IPC chains, ex_regs
- on ready list enqueue/dequeue operations

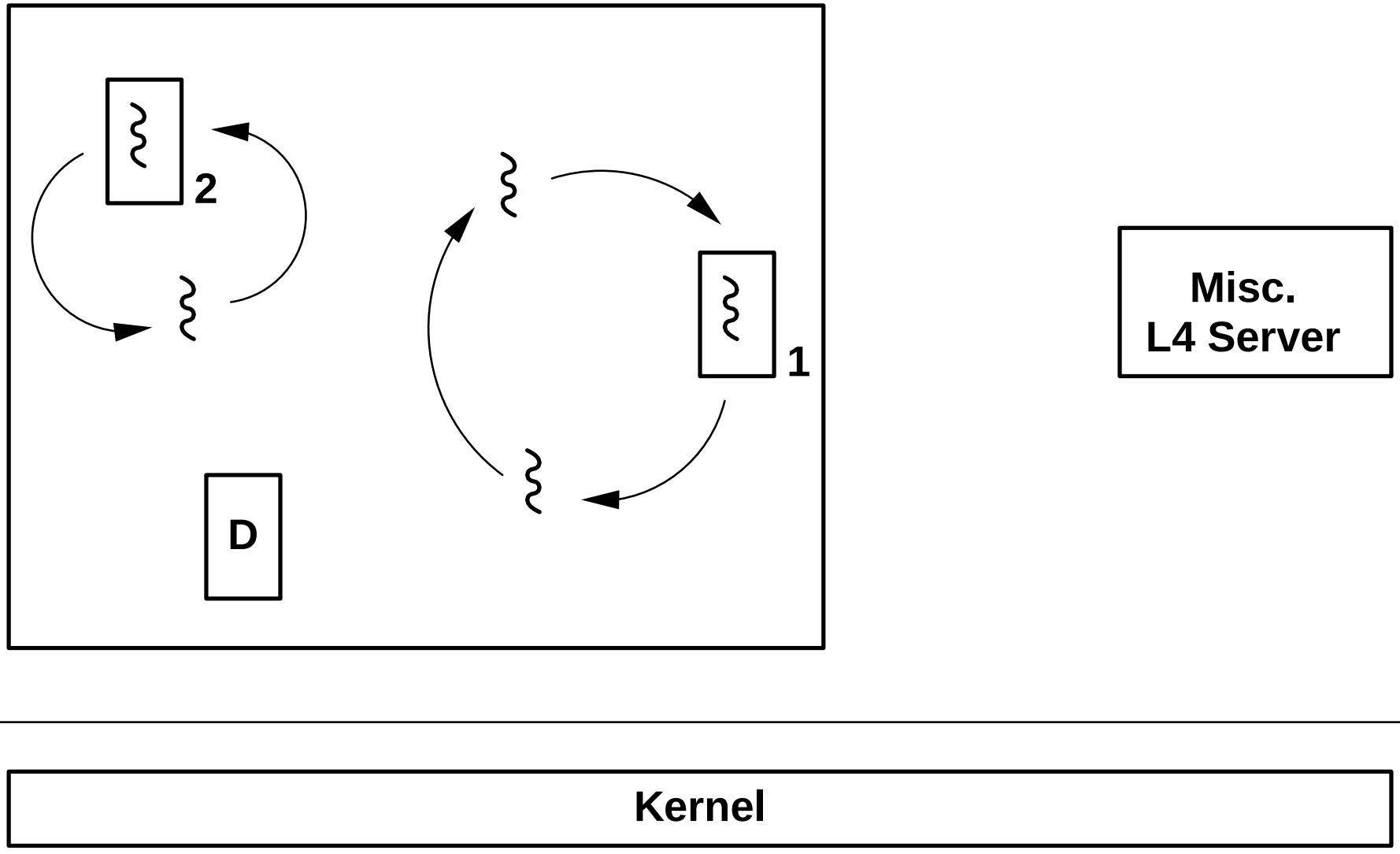
How to Send?

- nested IPC, or — influences regular IPC
- extra sender role

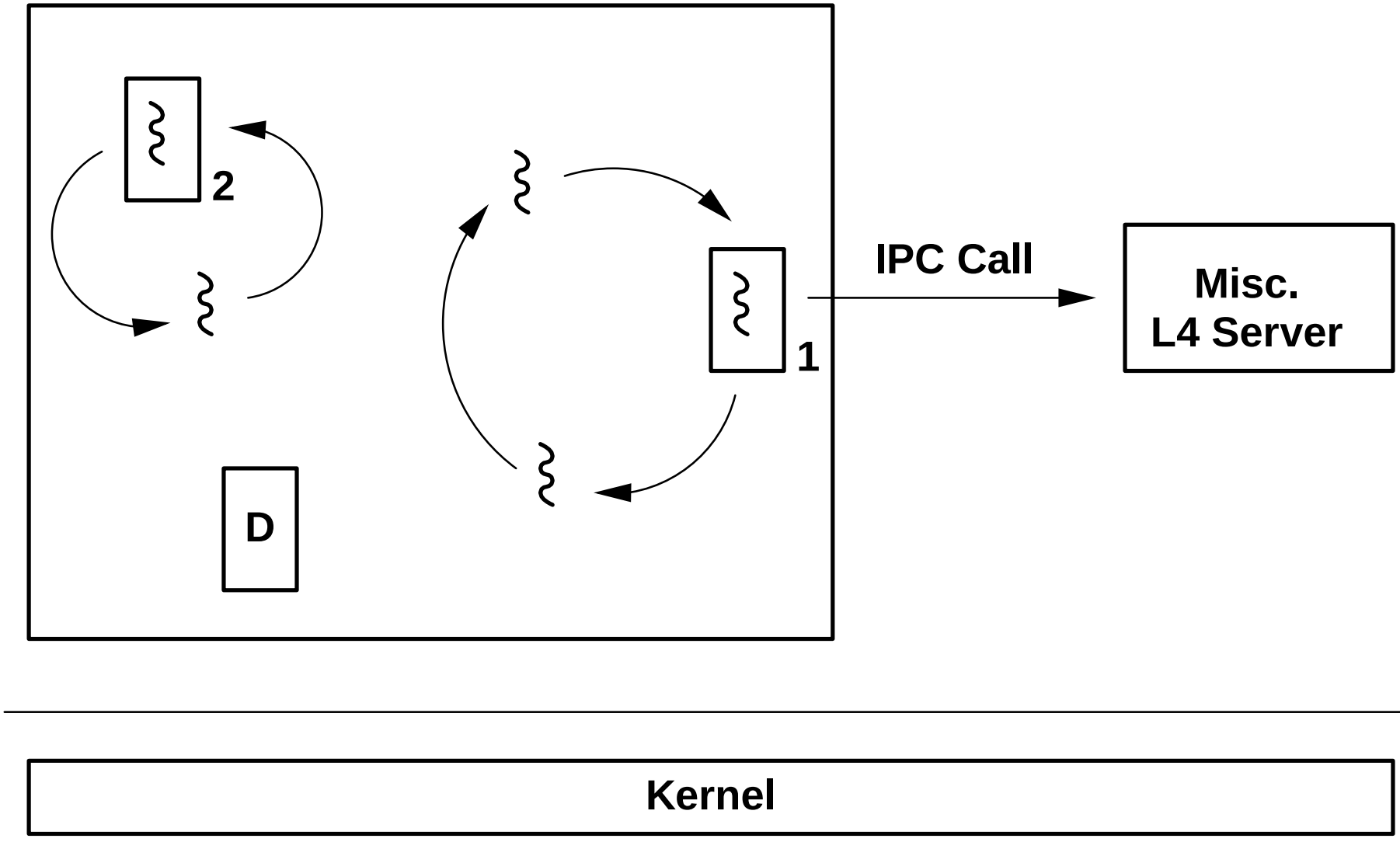
Where to Send?

- to a virtual processor, or — (minimality)
- to a dedicated dispatcher thread

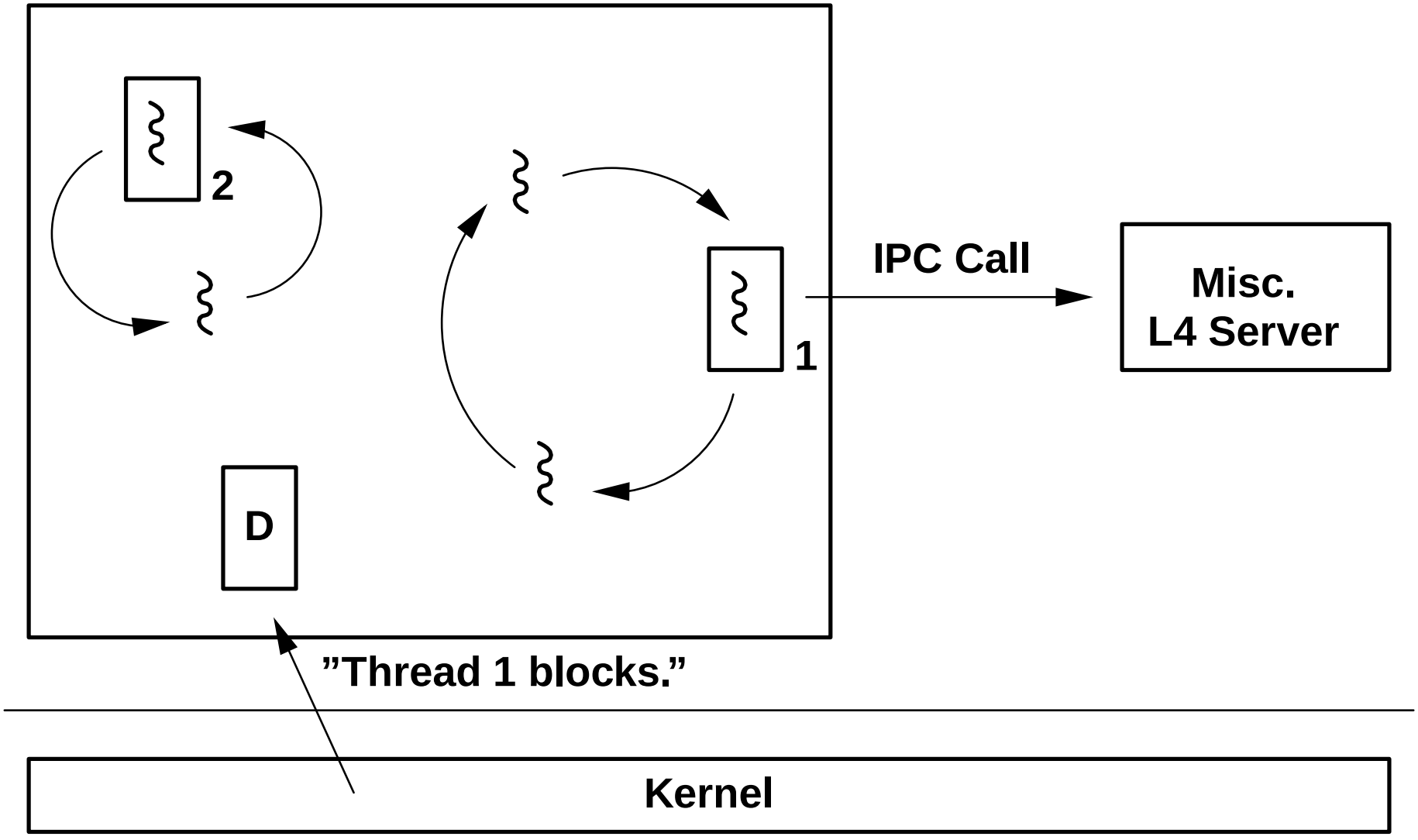
Activation Messages (3): Invoking IPC



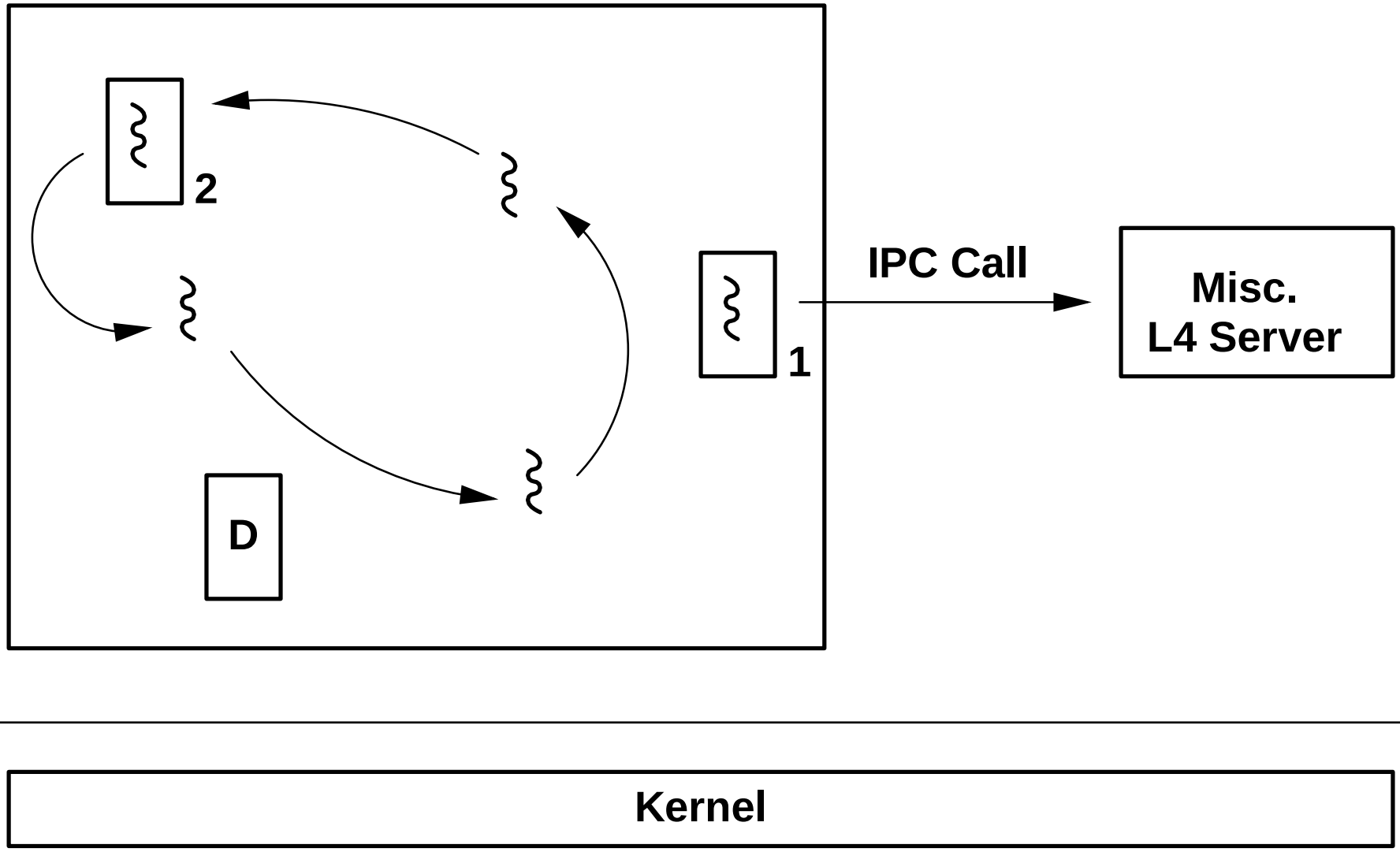
Activation Messages (3): Invoking IPC



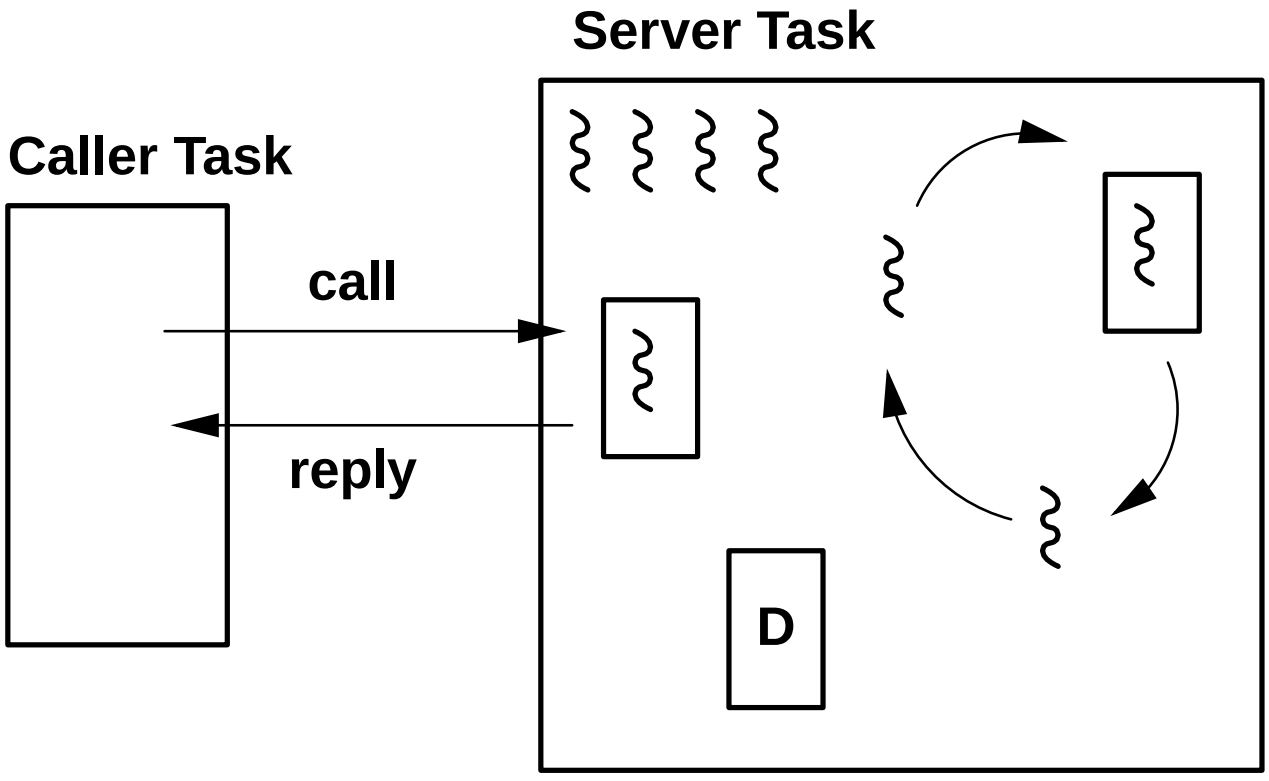
Activation Messages (3): Invoking IPC



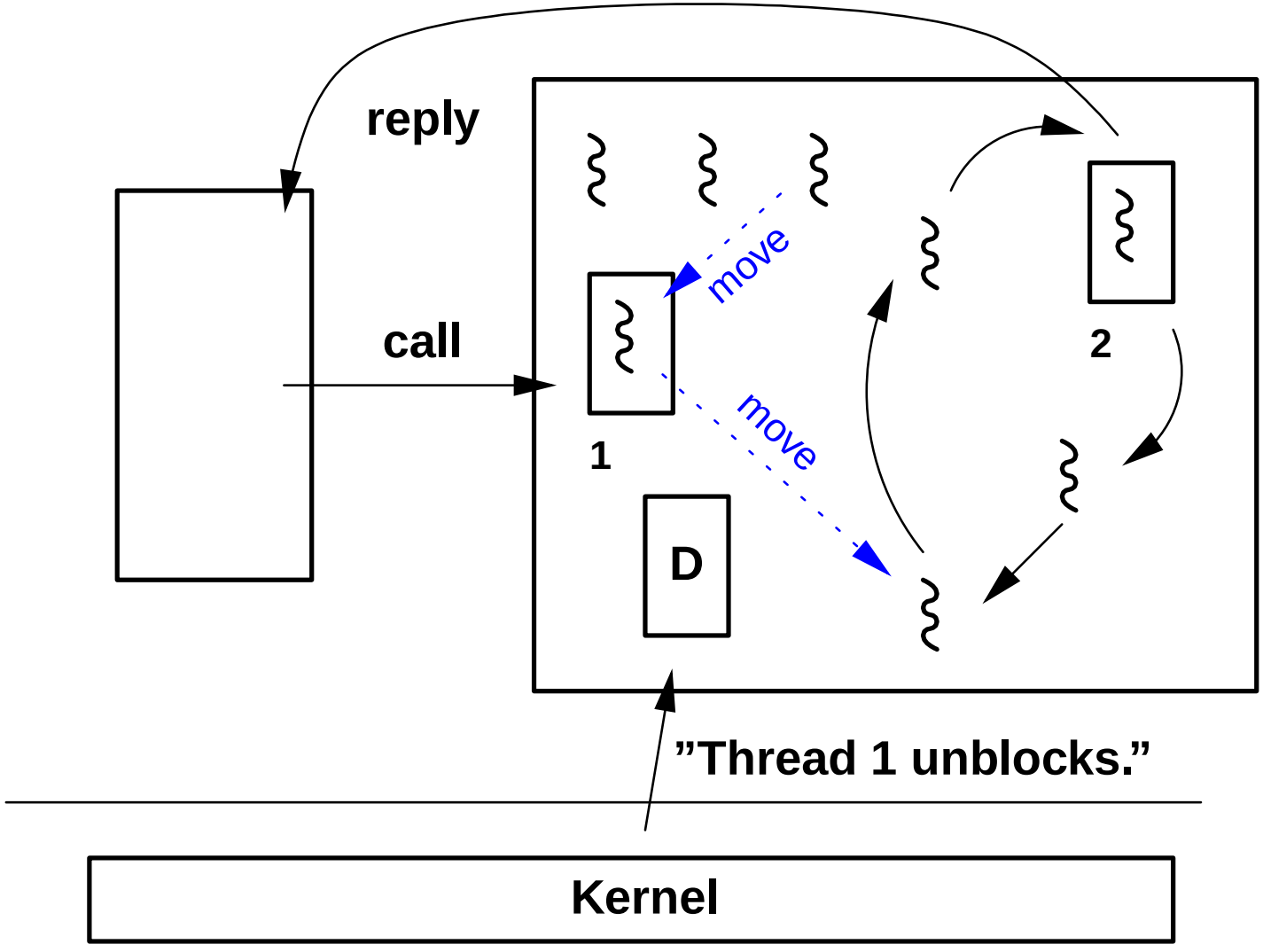
Activation Messages (3): Invoking IPC



Activation Messages (4): Receiving IPC



Activation Messages (4): Receiving IPC (2)



Activation Messages (5): Achievements

- Benefits of User-Level Threading Fully Operative
  - Efficient Local Communication
  - Flexible Per-Thread Resources
- User Threads can use L4 Functionality
  - Invoking IPC Calls
  - Receiving IPC Calls
- Preserved Performance
- Preserved Minimality

Activation Messages (6): Limitations

- Separated Local/Global Thread IDs
- Separated Local/Global Communication
- Limited Number of Concurrent Global IPCs
- No Different Priorities
- Restricted Scheduling Fairness

Outline

- Motivation
- Design Goals
- Achievements
- Details
  - Activation Messages
  - Deschedule Blocking Threads
- Conclusions
- Outlook

Deschedule Blocking User-Level Threads

1. User-Level Thread Invokes IPC
   - deschedule invoking thread
   - save blocking state at user level
2. Partner Replies
   - check if reply is valid
   - deliver the message

Deschedule Blocking User-Level Threads (2)

Zero Timeouts:

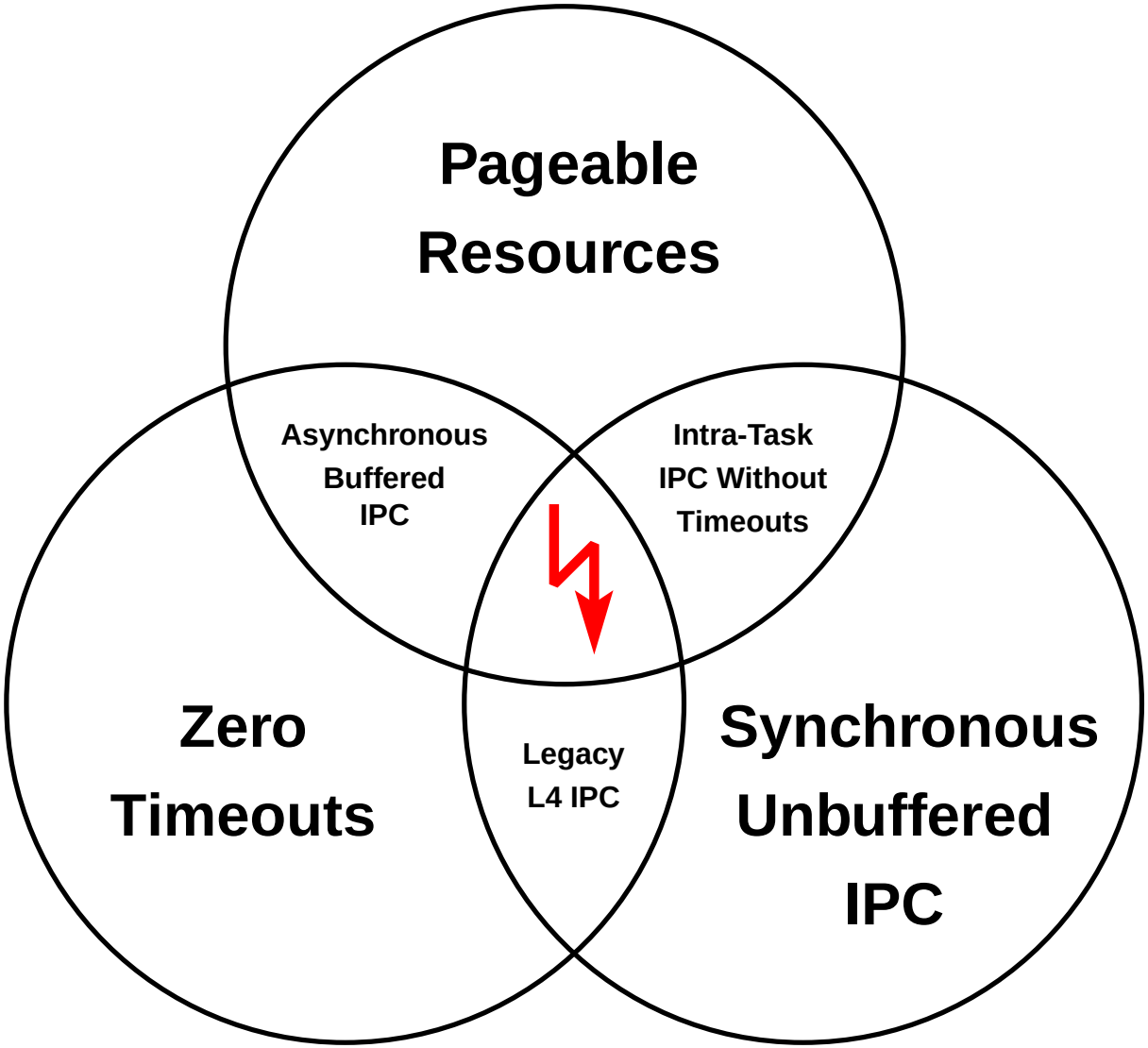
- dispatcher needs time
- partner thread does not trust the dispatcher

Fairness on Open Wait:

- Sender Queue
  - entirely in trusted memory
  - self-dequeueing
- Sender Information in the Receiver's Memory
- Wakeup Signals

Not Applicable.

Conclusions



Outlook

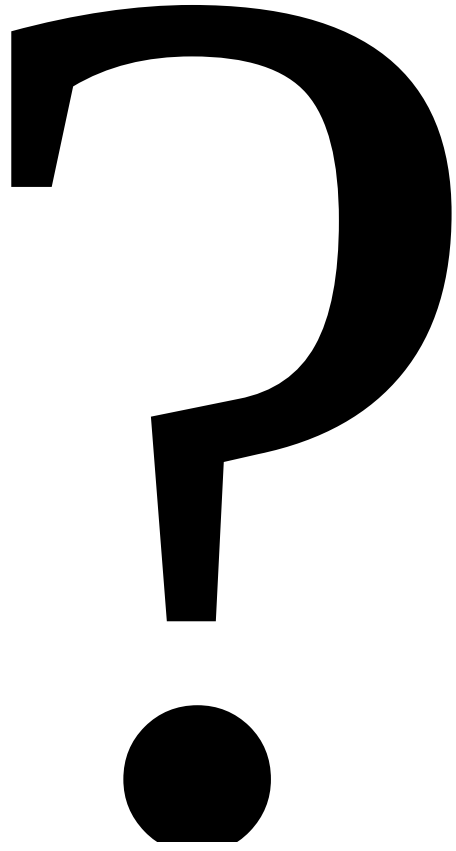
Summary

- investigated methods for user-level threading on L4
- proposed an $N : M$ threading model
- kernel part implemented and working

Future Work

- User-Level Thread Package
- Delayed Preemptions
- Applications

Questions



L4 Functionality

Inter-Process Communication (IPC):

- Synchronous and Unbuffered
  - No Buffers in the Kernel
- Zero Timeouts
  - Enforce Trust Relations
- Atomic Send-to-Receive Switch
  - Guarantee Functionality

L4 Functionality (2)

Client-Server Trust Relations:

- Client Trusts the Server
- Server Guarantees Functionality

Achieved by:

- IPC Call with Atomic Send-to-Receive Switch, and
- Reply with Zero Timeout.

L4 Functionality (3)

Fairness:

- Multiple Blocking Senders
- Receive from Any Thread
- Which Sender Comes First?

Progress:

- Thread States: Running or Blocking
- No Starvation of Running Threads

Dispatcher Code on VP

Entry Points, Asynchronous Events

1. set asynchronous timeout
2. timeout triggers
3. save (part of) the VP state
4. resume execution at entry point
5. perform thread switch

Issues:

- save state area
- lost preemption signals
- kernel cannot wait for user-level code

L4 Trust Relations