

## THE L4 MICROKERNEL

- Developed and implemented on ix86 by Jochen Liedtke, GMD (Germany)  $\approx$  1992–95 (“Version 2”).
- Successor of Eumel (1979) and L3 (1987).
- Ongoing development by Liedtke at IBM T.J. Watson Research Center (1997–99), University of Karlsruhe (since 1999): Versions “4”, “X”, “5”.
- Implementations at Dresden University of Technology and UNSW.

## THE L4 MICROKERNEL

- Developed and implemented on ix86 by Jochen Liedtke, GMD (Germany)  $\approx$  1992–95 (“Version 2”).
- Successor of Eumel (1979) and L3 (1987).
- Ongoing development by Liedtke at IBM T.J. Watson Research Center (1997–99), University of Karlsruhe (since 1999): Versions “4”, “X”, “5”.
- Implementations at Dresden University of Technology and UNSW.

### FEATURES:

- 7(!) system calls (Versions 2–4)
- recursive address spaces
- user-level page fault handlers
- user-level device drivers
- user-level scheduling
- real-time capable

## L4 IMPLEMENTATIONS

- ix86:
  - Liedtke's kernel, 100% assembler, Versions 1–4.x
  - Hohmut, Dresden, called *Fiasco*, C++, Version 2 $\epsilon$
  - Dannowski, new Karlsruhe kernel, C++, Version X, in progress
- MIPS R4x00:
  - Elphinstone, UNSW, 64-bit, assembler&C, Version 2 $\epsilon$
- Alpha:
  - Schönberg (Dresden), Potts (UNSW), SMP, 64-bit, PALcode&C, Version 2 $\epsilon$
- StrongARM:
  - Dannowski, Karlsruhe C++ (same as ix86), in progress
  - Wiggins (UNSW) assembler&C, in progress
- UltraSPARC:
  - Zadarnowski (UNSW), 64-bit, assembler&C, early days...
- IA-64:
  - Liedtke (Karlsruhe), 64-bit, ???, early days...

## L4 IMPLEMENTATION EXAMPLE: MIPS

### HISTORY:

- Written by Kevin Elphinstone, then a PhD student at UNSW, 1995–7
- First 64-bit version
- Essentially complete since February 1998
- Used in OS research projects at UNSW since 1996
- Used in teaching at UNSW since 1997
- Support for multiple page sizes in progress (Szmajda)

## L4 IMPLEMENTATION EXAMPLE: MIPS

### HISTORY:

- Written by Kevin Elphinstone, then a PhD student at UNSW, 1995–7
- First 64-bit version
- Essentially complete since February 1998
- Used in OS research projects at UNSW since 1996
- Used in teaching at UNSW since 1997
- Support for multiple page sizes in progress (Szmajda)

### STATISTICS (KERNEL VERSION 79):

- 6k lines assembler source (\* .s)
- 5k lines C source (\* .c)
- 1.7k lines C and assembler header files (\* .h)
- 80kB kernel text and static data
- 1MB kernel data (mostly TCBs and page tables)
- fast (details later)

## MAIN L4 ABSTRACTIONS

**threads:** execution abstraction and UIDs

**tasks:** address spaces and resources

**IPC:** message-based communication, including VM mappings

**flexpages:** VM page abstraction, including multiple page sizes

**clans and chiefs:** task hierarchy for (arbitrary) security models

**paggers, excepters, preempters, interrupt handlers:** exceptions

- Generally strict separation of:
  - *mechanisms* (provided by kernel) and
  - policy (implemented by user-level servers).
- Minimality achieved by orthogonality of mechanisms.

## L4 THREADS

- A thread is the basic active entity (unit of execution and scheduling).
- Threads communicate via message-passing IPC.
- Each thread has
  - a register set (IP, SP, user-visible registers, processor state)
  - an associated task/address space
  - a page fault handler (pager)  
this is a thread which receives page faults (via IPC)
  - an exception handler (MIPS only)  
this is a thread which receives exceptions (via IPC)
  - preemptors (not implemented)  
thread which receives *preemption messages*
  - scheduling parameters (priority, time slice)

## TASKS

- A task essentially provides an address space (plus a *clan boundary*).
- An (active) task contains one or more (active) threads.
- The number of threads in a task is fixed (128 on R4k).
  - Upon task creation, the full set of (128) threads is created with the task,
  - all but one are *inactive* (i.e., they do nothing and consume no resources).
  - further threads can be activated via a system call (`lthread_ex_regs()`)
- Similarly, upon system initialisation, the full set of tasks (2048 on R4k) is created, but in an *inactive* state.
- A task has a *chief* (parent/owner).

# IPC

- Message-passing IPC provides communication between threads.
- All IPC is
  - synchronous (i.e., blocking), and
  - unbuffered

This is key to high IPC performance

- IPC requires an *agreement* between sender and receiver (i.e., receiver must be expecting IPC, must provide buffers, etc.).
  - IPC supports in-line and out-of-line *by-value* data.
  - IPC supports *map* and *grant* VM operations for *by-reference* data.
- Blocking can be limited by *timeouts*.

## FLEXPAGES

- Describe virtual memory regions for use in *mapping* operations.
- Generalisation of pages by abstracting over page size.
- Usually called *fpages*.

### PROPERTIES:

- size  $2^s$ , ( $\geq$  hardware page size),
- aligned to  $2^s$
- kernel should try to map whole fpage as a single super-page
- partially populated fpages: an fpage refers to all mapped pages within the region designated by it.

## ADDRESS SPACES

Address spaces are recursively constructed from mappings into other address spaces:

- A “magic” initial address space  $\sigma_0$  maps physical memory one-on-one;
- each address space (pager) can map portions of its own address space into another (cooperating) address space.

## ADDRESS SPACES

Address spaces are recursively constructed from mappings into other address spaces:

- A “magic” initial address space  $\sigma_0$  maps physical memory one-on-one;
- each address space (pager) can map portions of its own address space into another (cooperating) address space.

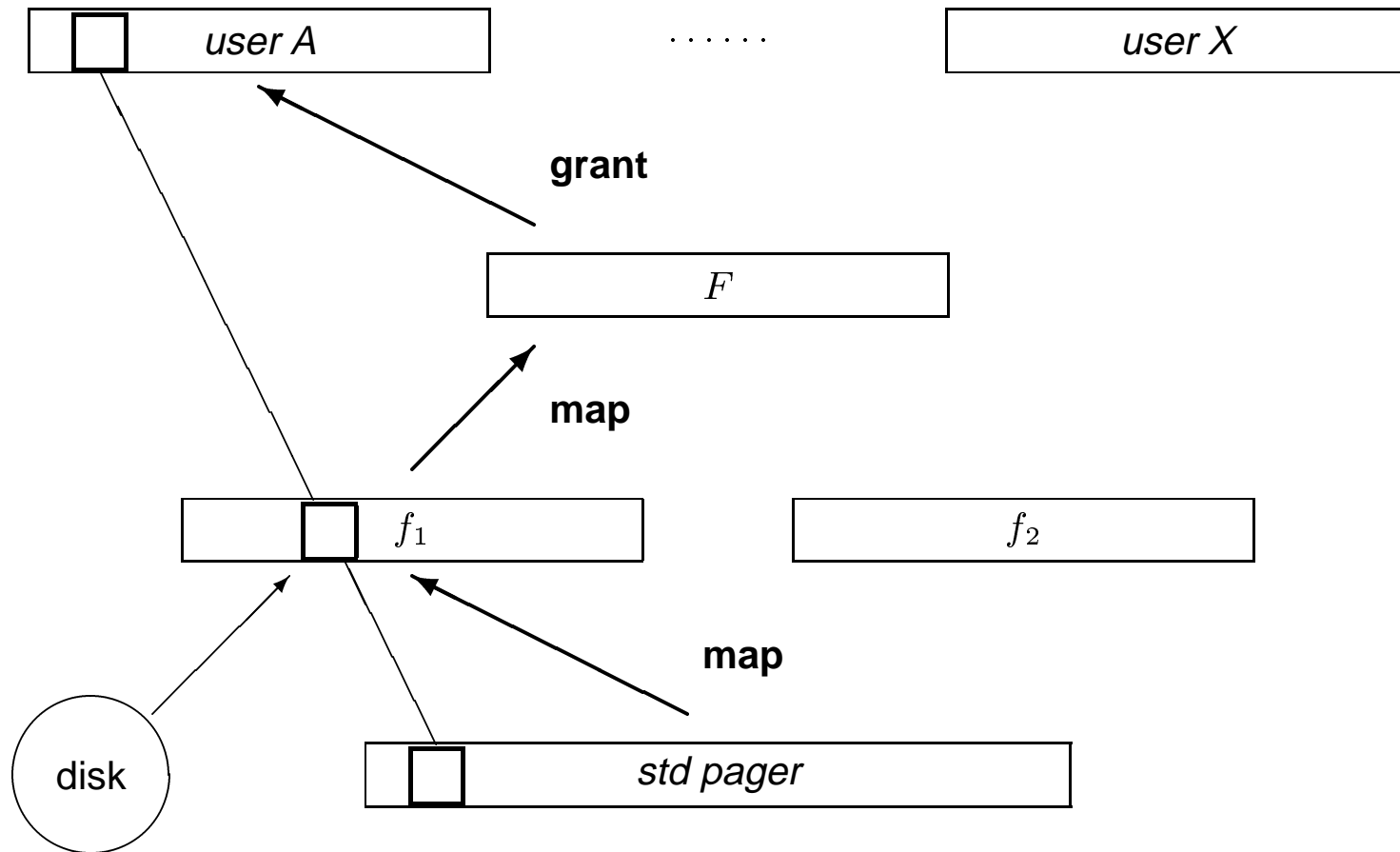
### THIS IS DONE WITH THE HELP OF THREE MAPPING PRIMITIVES:

**Map:** map a page from sender's to receiver's AS  
caller retains page

**Grant:** map a page from sender's to receiver's AS  
page is removed from caller

**Flush:** undoes a Map (removes page from receiver's address space)

# GRANTING



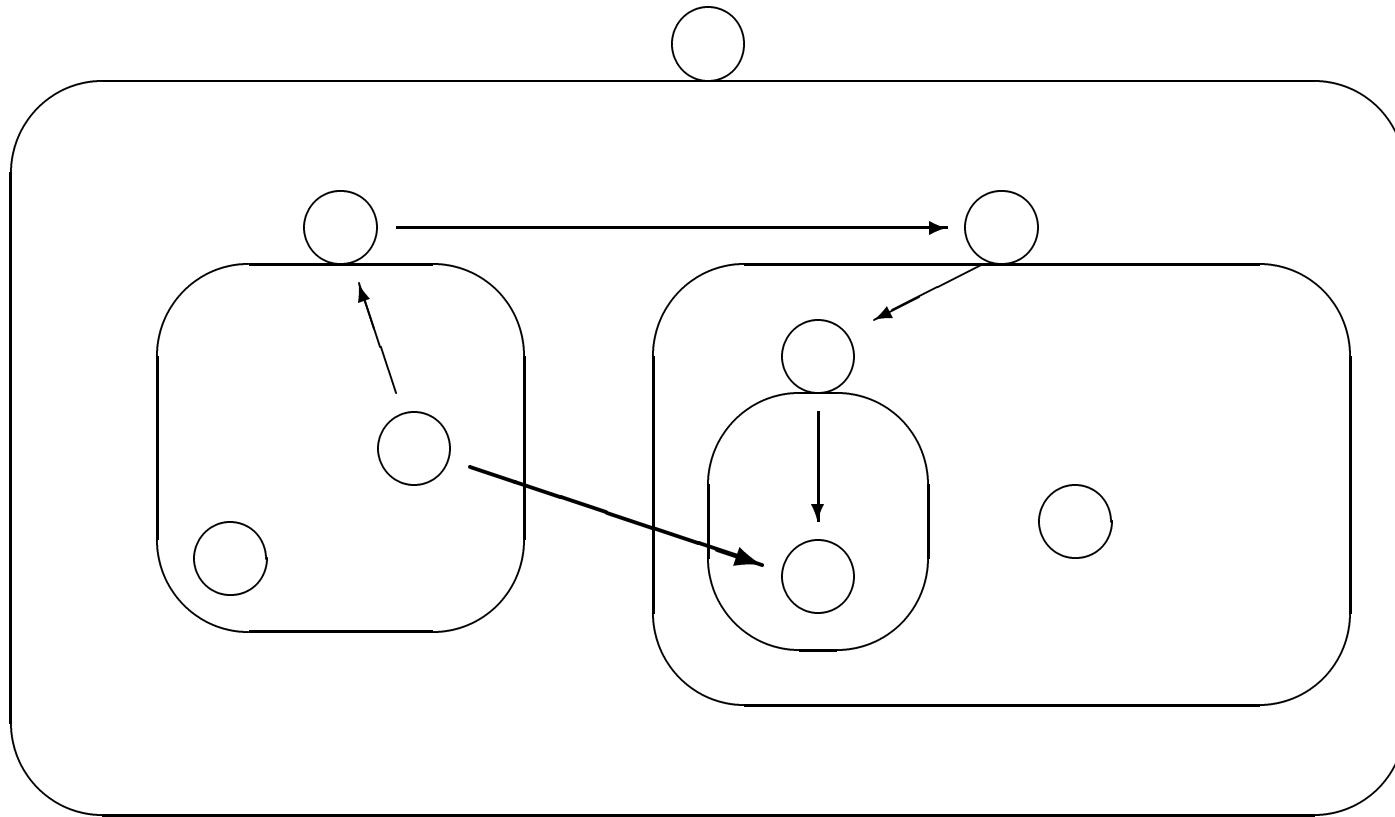
## CLANS & CHIEFS

Clans & Chiefs are a *security mechanism* for:

- **task control:** define ownership of tasks
  - a task creating another task becomes the *chief* of that task
  - the set of tasks created by a chief is that chief's *clan*
  - task can be killed only
    - ① directly by its chief
    - ② indirectly when its chief is killed
- **communication control:** restrict the flow of messages
  - intra-clan messages are delivered directly
  - inter-clan messages are redirected to chief
  - chief can forward the message transparently

Depth of hierarchy is limited (16 on MIPS).

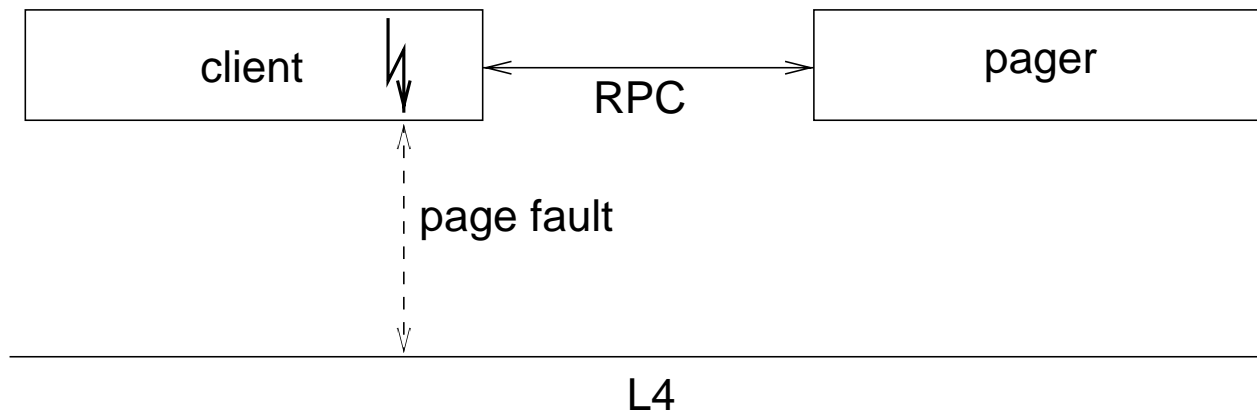
## INTER-CLAN IPC:



## PAGE FAULTS AND PAGERS

- L4 maintains kernel page tables containing mappings *explicitly established* by user threads (via IPC).
- If a thread triggers a page fault, the kernel invokes the thread's pager by
  - ① sending an IPC message to the pager on the faulter's behalf
  - ② catching the pager's reply and continue the faulter

The pager's reply is expected to contain a mapping for the missing page.



## EXCEPTIONS AND EXCEPTERS

**L4/MIPS EXCEPTION HANDLING:** Totally analogous to page faults:

- Each thread has an *excepter*
- If a thread triggers an exception, the kernel invokes the thread's excepter by:
  - ① sending an IPC message to the pager on the faulters' behalf,
  - ② catching the pager's reply and continue the faulter.
- The excepter may chose not to reply, leaving the excepting thread blocked forever.

**L4/IX86 EXCEPTION HANDLING:** Virtualisation of hardware:

- A thread installs its own interrupt vector, using (kernel-emulated) processor features.

The kernel handles some exceptions internally (TLB miss, system call).

## INTERRUPTS AND INTERRUPT HANDLERS

- Each hardware interrupt is modelled as a virtual (hardware) thread.
- (At most) one user-level interrupt-handler thread is associated with each hardware interrupt.
- If an interrupt occurs, the kernel generates an IPC from the interrupt thread to the interrupt handler.
- The interrupt handler is in general (part of) a *device driver*. *device registers*.
  - It will need access to *device registers*.
  - This is done via a special mapping protocol of the root pager (covered later).

The kernel handles some interrupts internally (timer interrupt).

## DEVICE INTERFACES

- Devices are controlled via special *device registers*, typically:
  - **status register**, to obtain device status,
  - **control register**, to send commands to device  
status and control registers are at the same address,
  - **data register(s)** to pass data/command parameters.
- Number of registers is normally small,  
data and parameter buffers are passed in memory, address specified in  
data registers
- Device registers are either *memory mapped* or accessed via *I/O instructions*
- Devices access only *physical memory*, i.e. bypass the MMU.

## DEVICE DRIVER

- Interface between hardware (device controller) and OS.
- Processes OS device requests and controls device by writing to data & command registers.
- Monitors device by reading status & data registers and handling device interrupts.
- Transfers data between OS buffers and device.

## L4 DEVICE DRIVER

- Runs at user level.
- Has mappings for device registers (MIPS) and physical memory.
- Typically consists of **top half** and **bottom half**.

## L4 DEVICE DRIVER

- Runs at user level.
- Has mappings for device registers (MIPS) and physical memory.
- Typically consists of **top half** and **bottom half**.

**BOTTOM HALF HANDLER:** processes device interrupts:

- receive L4 interrupt IPC,
- check success (status register),
- make data available (copying or mapping),
- initiate next request (if one),
- notify/reply to user (IPC).

## L4 DEVICE DRIVER

- Runs at user level.
- Has mappings for device registers (MIPS) and physical memory.
- Typically consists of **top half** and **bottom half**.

**BOTTOM HALF HANDLER:** processes device interrupts:

- receive L4 interrupt IPC,
- check success (status register),
- make data available (copying or mapping),
- initiate next request (if one),
- notify/reply to user (IPC).

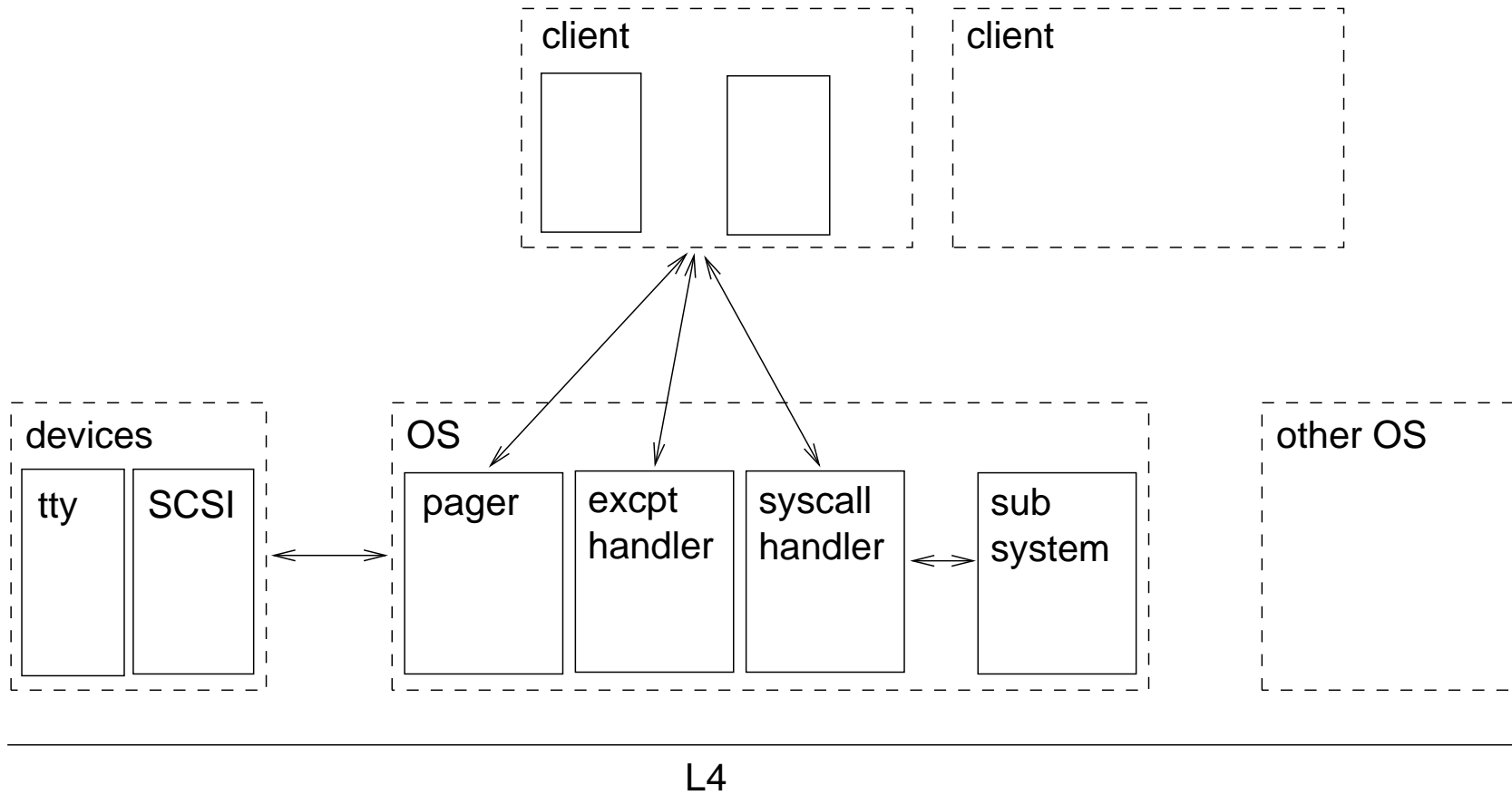
**TOP HALF HANDLER:** processes user requests:

- initiate I/O (set up parameter buffer, write device registers)  
or queue request if device busy,
- reply to user (if asynchronous)

## L4 DEVICE DRIVER...

- Bottom half must be fast (to avoid missing interrupts):
  - runs at high priority,
  - generally runs with interrupts disabled,
  - does minimal amount of work,
  - longer tasks left to top half  
(copying buffers, replying to user).
- Concurrency control required between top and bottom half.
- Bottom half must not be blocked by top half.

# L4-BASED OS DESIGN



## OS STRUCTURE

- OS server is chief of client processes.
- Client's "system calls" are library stubs performing RPC to OS server.
- "OS" may consist of many server threads in same or separate tasks.
- OS server may redirect client requests to other servers
  - within OS server task,
  - outside OS server task.
- clans & chiefs mechanism:
  - prevents direct client access to servers running as separate tasks,
  - supports easy redirection of "system call" RPC.