FIASCO.OC

CAPABILITY BASED DESIGN

- Virtual resource naming

MULTI-PROCESSOR SUPPORT

- Transparent migration

BROAD SUPPORT OF ARM PLATFORMS

RESOURCE QUOTAS

FIASCO.OC

VIRTUALIZATION

- Support for hardware virtualization features
- Improved para-virtualization

FLEXIBLE SYSTEM CONFIGURATION

- Script based

FIASCO.OC / BASIC CONCEPTS

API BASICS

TASKS (PROTECTION DOMAINS)

THREADS

IRQS

CAPABILITIES

MULTI-CORE MODEL

CAPABILITY TABLES

API CONCEPTS

EVERYTHING IS AN OBJECT

- all system calls run on an object
- 'one system call' — send message to object

CAPABILITIES — THE OBJECT REFERENCES

UNIFORM INTERFACES

- kernel and user-level objects with uniform interfaces

FACTORY FOR OBJECT CREATION

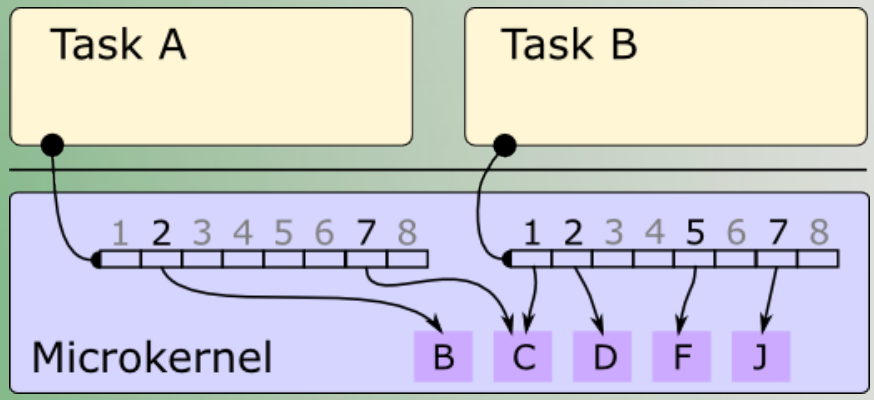
CAPABILITIES

REFERENCES TO KERNEL OBJECTS

- local naming (per protection domain)
- access control

KEPT IN PER TASK CAPABILITY TABLE

- comparable to file descriptors, file-descriptor table



OVERVIEW: KERNEL OBJECTS

TASK	protection domain
THREAD	thread of execution in a task
FACTORY	creation of (kernel) objects
IPC GATE	communication channel / object
IRQ	async. signalling (software / hardware)
ICU	hardware IRQ controller / manager
SCHEDULER	manage CPUs and scheduling

TASK

(class Task : Kobject)

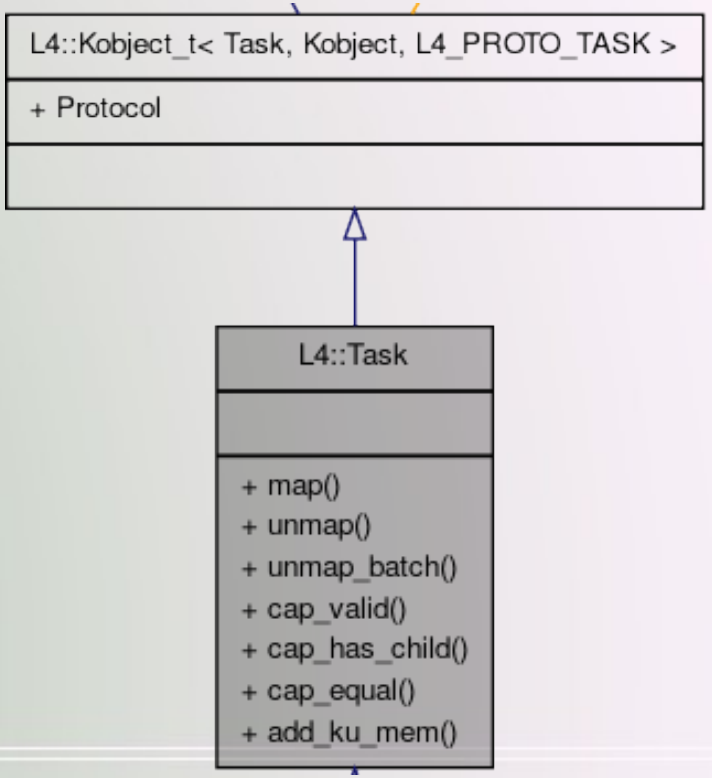
PASSIVE PROTECTION DOMAIN (NO THREADS)

Memory protection (incl. IA32 IO-ports)

- virtual memory space (MMU)
- kernel-user memory (KU memory)

Access control (kernel objects, IPC)

- object space (capability table)



OBJECT SPACE

CONTAINS CAPABILITIES TO KERNEL OBJECTS

SIMILAR TO VIRTUAL MEMORY...

- completely user-level managed
- pass capabilities directly or via IPC (MapItem, Flexpage)

THREAD

(class Thread : Kobject)

EXECUTES IN A TASK

access to virtual memory and capabilities of that task

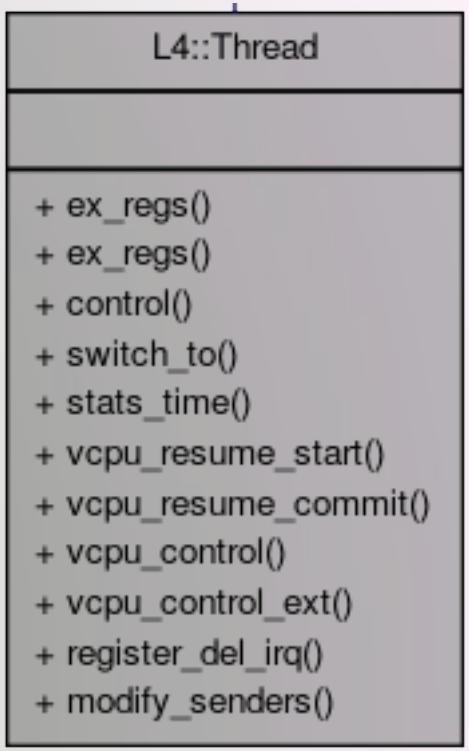
STATES: ready, running, blocked

UTCB: for message contents (sys-call parameters)

ACTIVE ENDPOINT IN SYNCHRONOUS IPC

NEED SCHEDULER OBJECT TO SETUP SCHEDULING

EXTENDED FEATURES: vCPU mode (later)



FACTORY

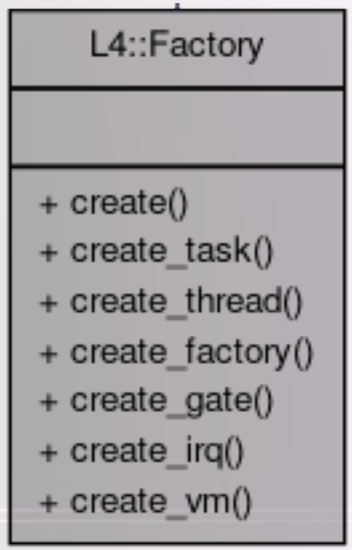
(class Factory : Kobject)

CREATE (KERNEL) OBJECTS

- limited by kernel-memory quota
- secondary kernel memory also accounted
  page tables, KU memory, FPU state buffers, mapping nodes

GENERIC INTERFACE

- used for kernel and user-level objects

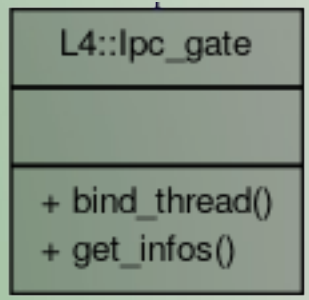


IPC GATE

(class Ipc_gate : Kobject)

COMMUNICATION CHANNEL

- messages forwarded to a thread
- incl. protected label for identification
- fundamental primitive for user-level objects

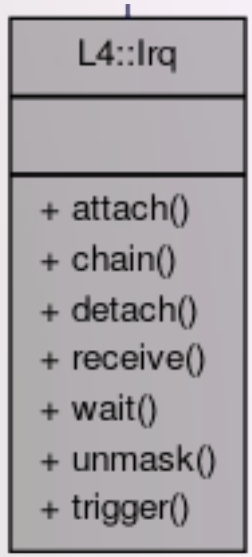


IRQ

(class Irq : Kobject)

ASYNCHRONOUS SIGNALING

- signal forwarded as message to thread
- no payload
- incl. protected label for identification
- fundamental primitive for hardware IRQs and software signaling



ICU

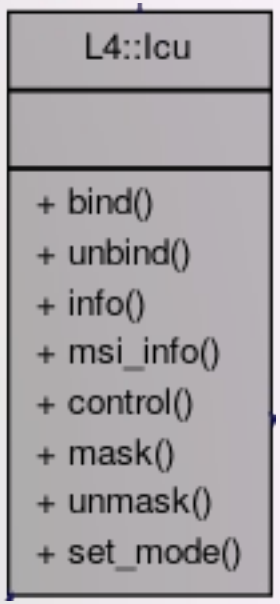
(class Icu : Kobject)

INTERRUPT CONTROLLER ABSTRACTION

- bind IRQ object to a hw IRQ pin / source
  IRQ object gets triggered by hardware interrupt
- control parameters of IRQ pin / source

GENERIC INTERFACE

- used also for virtual IRQ sources

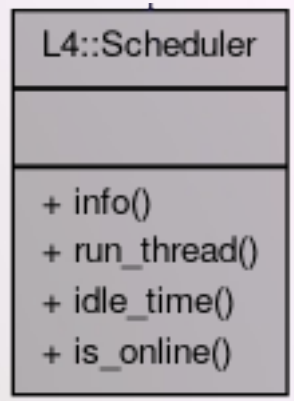


SCHEDULER

(class Scheduler : Kobject)

MANAGE CPUs and CPU TIME

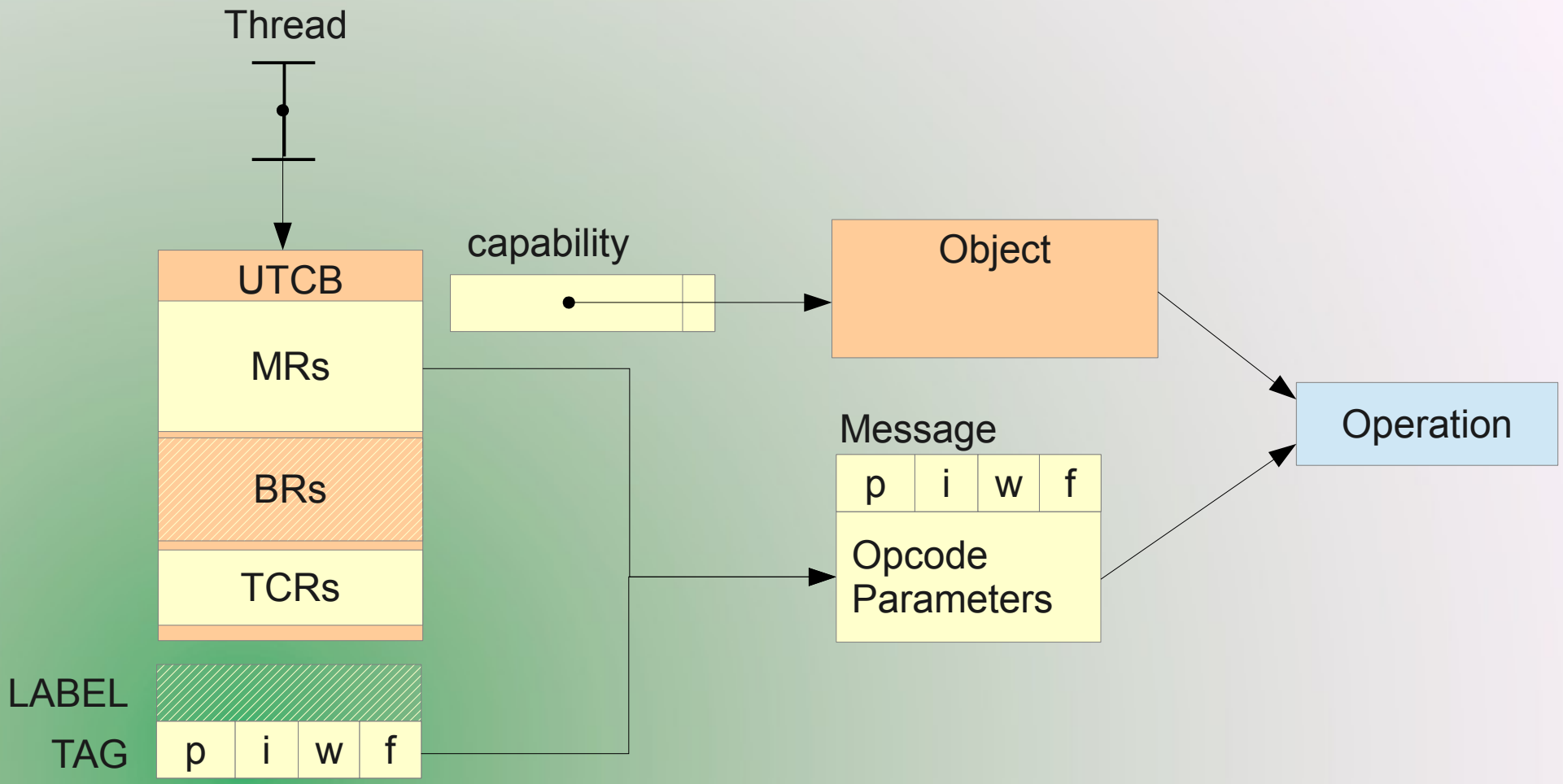
- bind thread to CPU
- control scheduling parameters
- gather statistics



GENERIC INTERFACE

- used to refine resource management policies

KERNEL INTERFACE



MESSAGES

TAG message descriptor

- number of words
- number of items
- flags
- protocol id (payload)

LABEL protected message payload

- secure identification of a specific capability a message was sent through

MESSAGES: UTCB user-level thread control block

MR message registers

- untyped message data
- message items (capabilities, memory pages, IO ports)

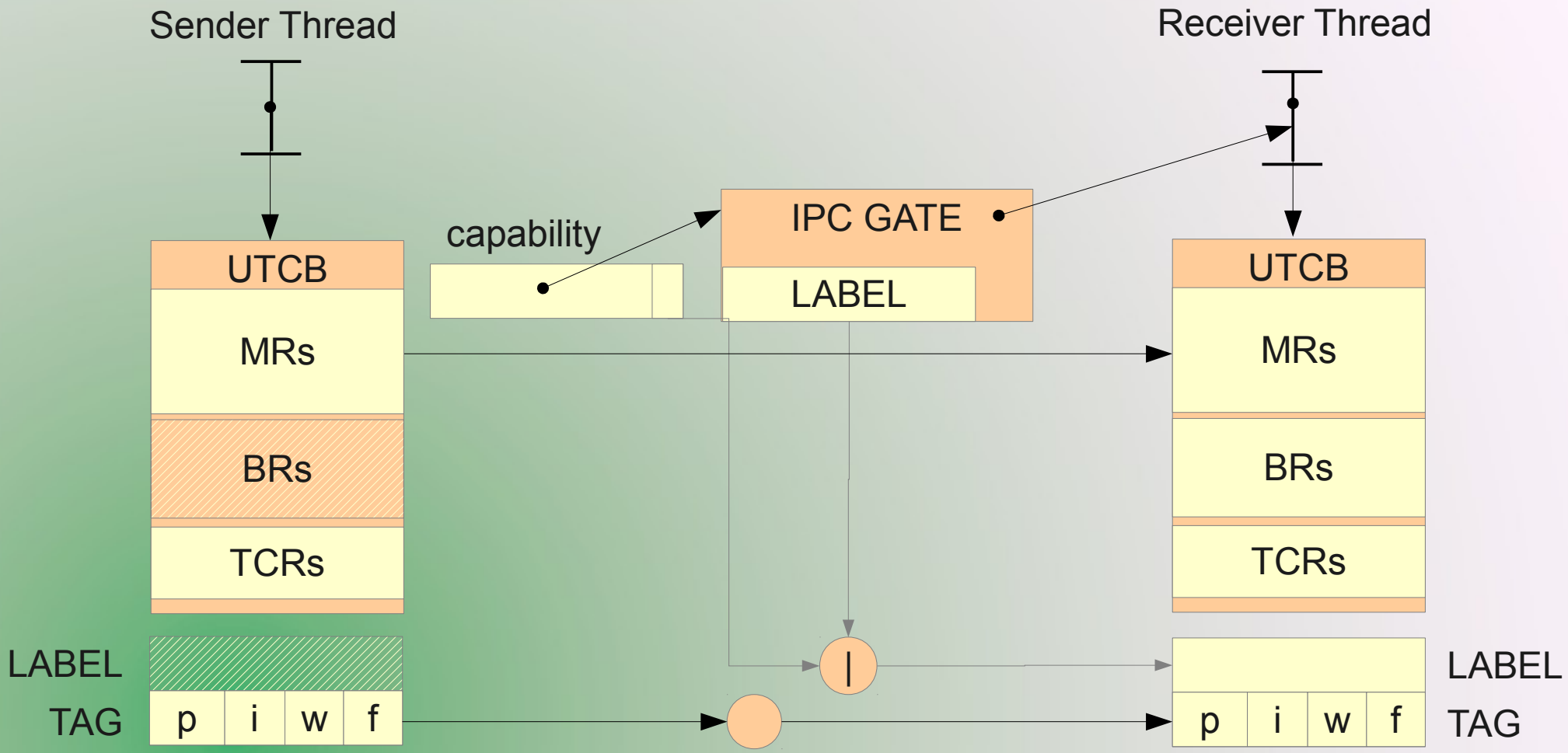
BR buffer registers

- receive buffers for capabilities, memory, IO-ports
- absolute timeouts

TCR thread control registers

- error code
- user values

COMMUNICATION (IPC)



IPC II

SYNCHRONOUS

- sender waits until the receiver is ready to receive
- blocking can be limited by a timeout

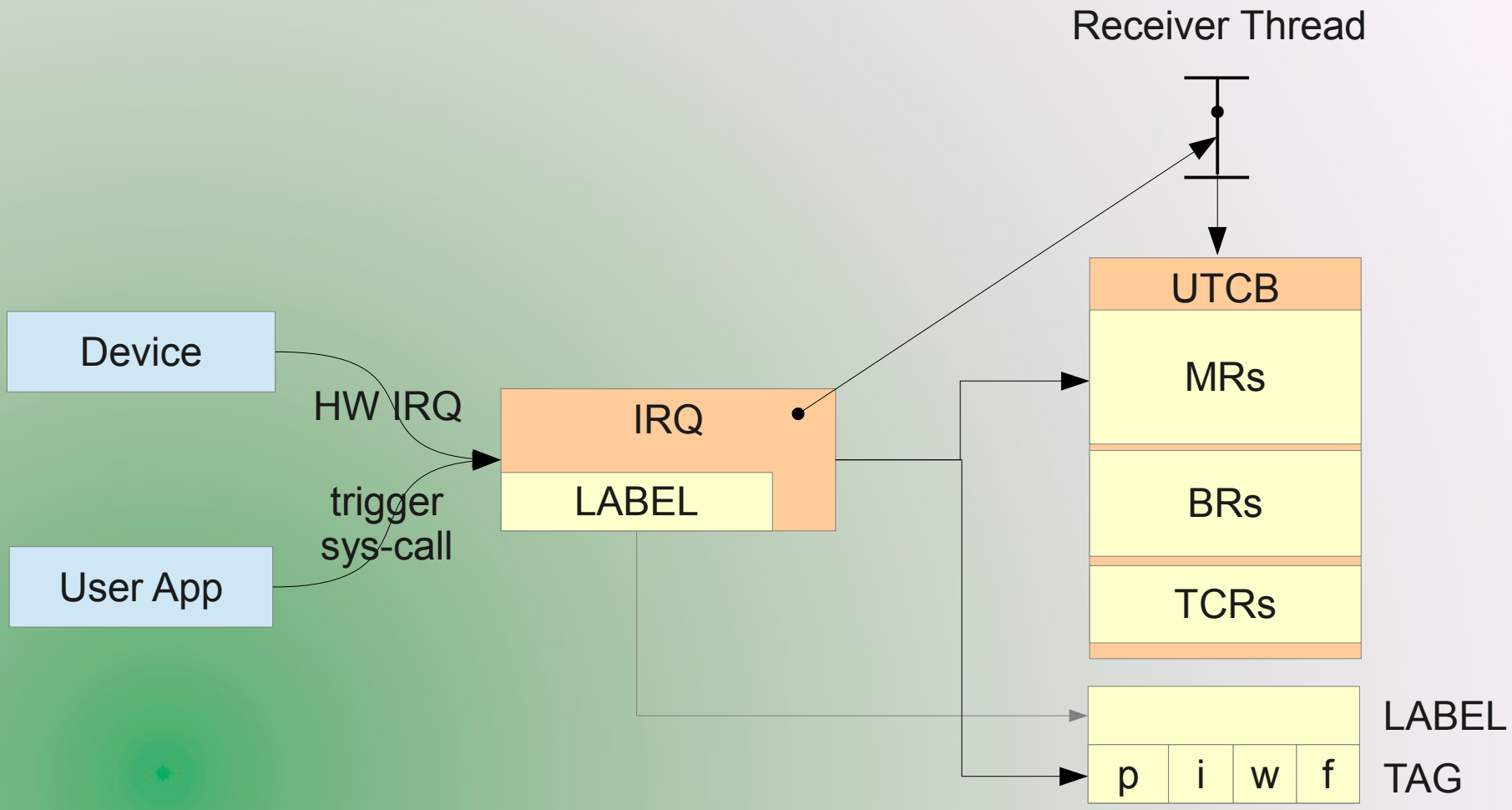
ATOMIC DATA ONLY IPC

- map IPC is not atomic (may block in map)

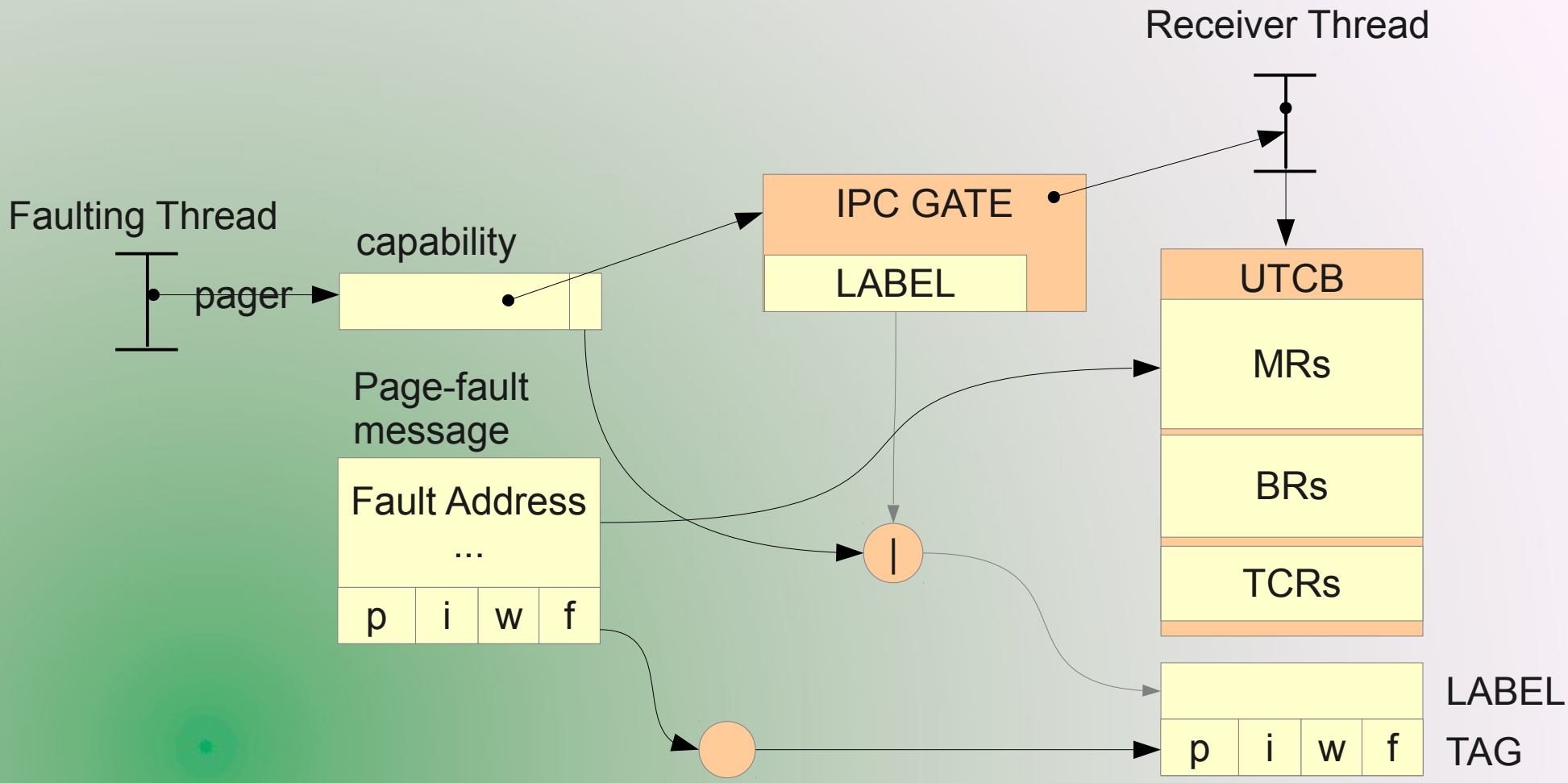
ATOMIC SEND-RECEIVE TRANSITION IN CALL

- reply can have a zero timeout

ASYNC. IRQ MESSAGE



PAGE-FAULT MESSAGE



THE FIASCO.OC MULTI-CORE MODEL

MULTI-CORE MODEL I

OPTIMIZE FOR CPU LOCAL OPERATIONS & DATA STRUCTURES

- CPU local ready queues
- CPU local timeout queues
- Thread Control Blocks (TCBs) are CPU local
  - CPU number can be compared for equality to the current CPU
- No implicit thread migration
- CPU-local IPC without any global data structures, locks

MULTI-CORE MODEL II

CROSS-CPU IPC & TASKS

- Tasks spawn multiple CPUs
- IPC can be cross-CPU
- IRQs can be cross-CPU
- User applications can migrate threads among CPUs

CAPABILITIES

- can be access without locks
- use Read Copy Update (RCU) for kernel object deletion

IMPLEMENTATION DETAILS

CAPABILITY TABLES

IMPLEMENTING CAP-TABLES

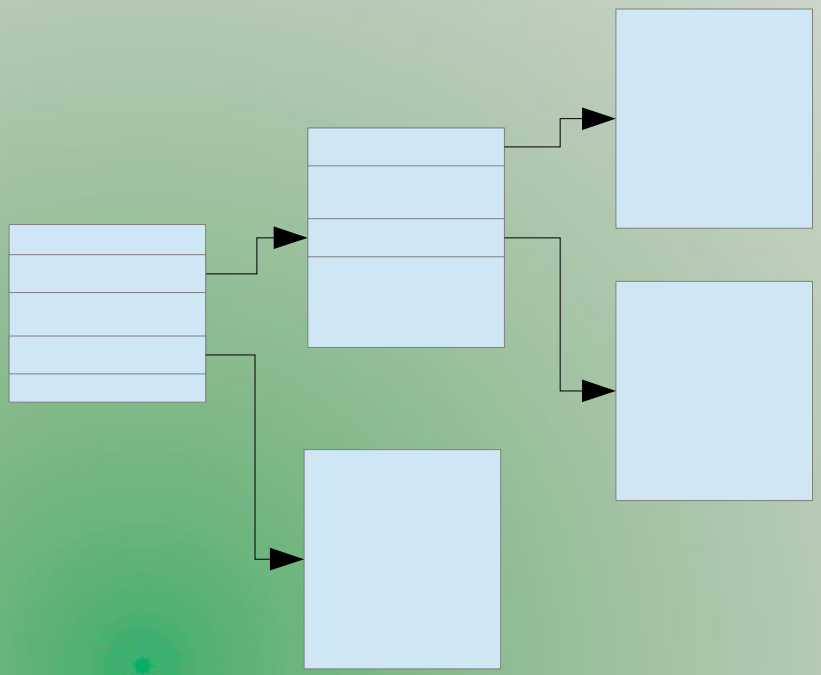
PARAMETERS

- fully user-controlled
- lookup / dereference is performance critical
- 32bit systems 2^{20} capabilitiy slots per task
- kernel structure 16byte per capability + mapping node
- sparsely populated
- $2^{20} \cdot$ 16byte = 16.777.216 = 16MB (array per task)
  this is too big...

CAPABILITY TABLES

SOLUTION 1: multi-level array

- lookup needs software lookup in hierarchical structure



256 caps per 4k page

1024 pointers per 4k page

→ 3 levels (16, 1024, 256)

CAPABILITY TABLES

SOLUTION 2: virtually mapped array

- direct lookup, index with cap index
- page faults in non mapped areas
  - access capabilities with special asm code
  - special case in page-fault path that returns 0 for a faulting read

WHEN TO USE WHAT?

KERNEL IN PHYSICAL MEMORY

- PPC, FIASCO-UX
- multi-level array

KERNEL IN VIRTUAL MEMORY

- IA32, ARM ...
- virtually mapped sparse array