TECHNISCHE UNIVERSITÄT DRESDEN

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

Memory

Björn Döbel

Dresden, 2010-10-26

So far ...

- Introduction
  - monolithic vs. microkernels
  - microkernel history / L4 family
  - L4 concepts: tasks, threads and IPC
  - Fiasco.OC/TUDOS introduction
- Threads & Synchronization
  - address spaces / tasks
  - threads
    - TCB, kernel entry
  - scheduling
- **Memory**

Outline

- Task creation
- Page-fault handling
- Flexpages
- Hierarchical pagers
- Region manager
- Dataspaces

TECHNISCHE UNIVERSITÄT DRESDEN

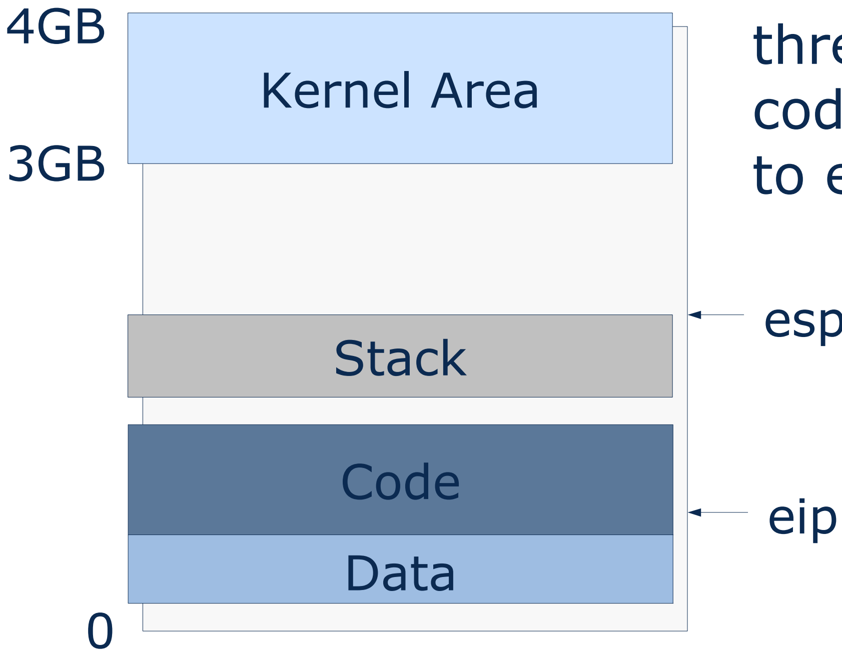
Task Creation

Booting the system

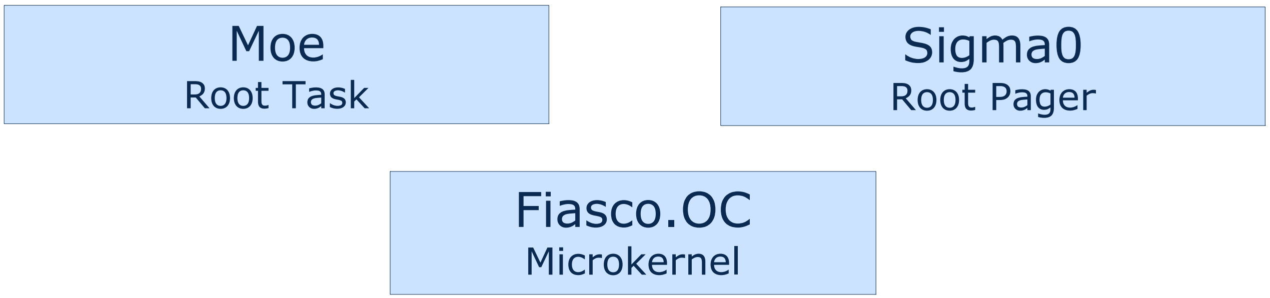
- **BIOS** starts, then loads and executes boot sector
- **boot loader** (i.e. GRUB) loads kernel and multiboot modules
- **bootstrap** interprets binary modules and sets up kernel structures (Kernel Info Page)
- Fiasco.OC **kernel** started by bootstrap
- moe (**roottask**) and Sigma0 (initial address space) started by kernel

→ next step: start application tasks

Address space layout



thread needs to access code/data/stack/... to execute



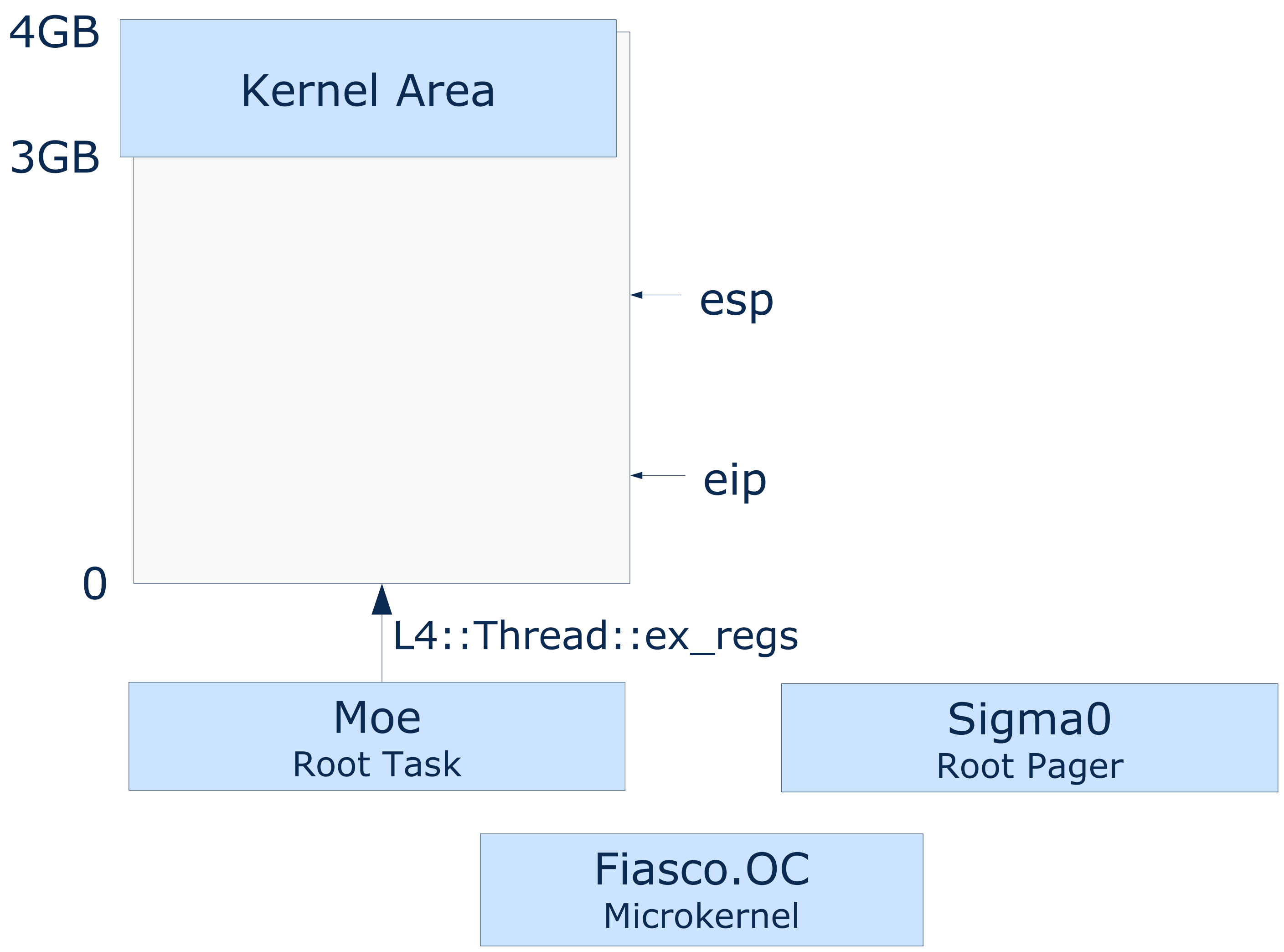
create Task and Thread

```
/* Create a new task. */
l4_msgtag_t
L4::Factory::create_task (Cap< Task > const &  task_cap,
                          l4_fpage_t const &    utcb_area,
                          l4_utcb_t             *utcb = l4_utcb()
)


/* Create a new thread. */
l4_msgtag_t
L4::Factory::create_thread (Cap< Thread > const &  target_cap,
                            l4_utcb_t              *utcb = l4_utcb()
)
```

start new Thread

```
/* Commit the given thread-attributes object. */
l4_msgtag_t
L4::Thread::control (Attr const &  attr)


/* Exchange basic thread registers. */
l4_msgtag_t
L4::Thread::ex_regs (l4_addr_t      ip,         /* instruction pointer */
                     l4_addr_t      sp,         /* stack pointer */
                     l4_umword_t  flags,
                     l4_utcb_t      *utcb = l4_utcb()
)
```

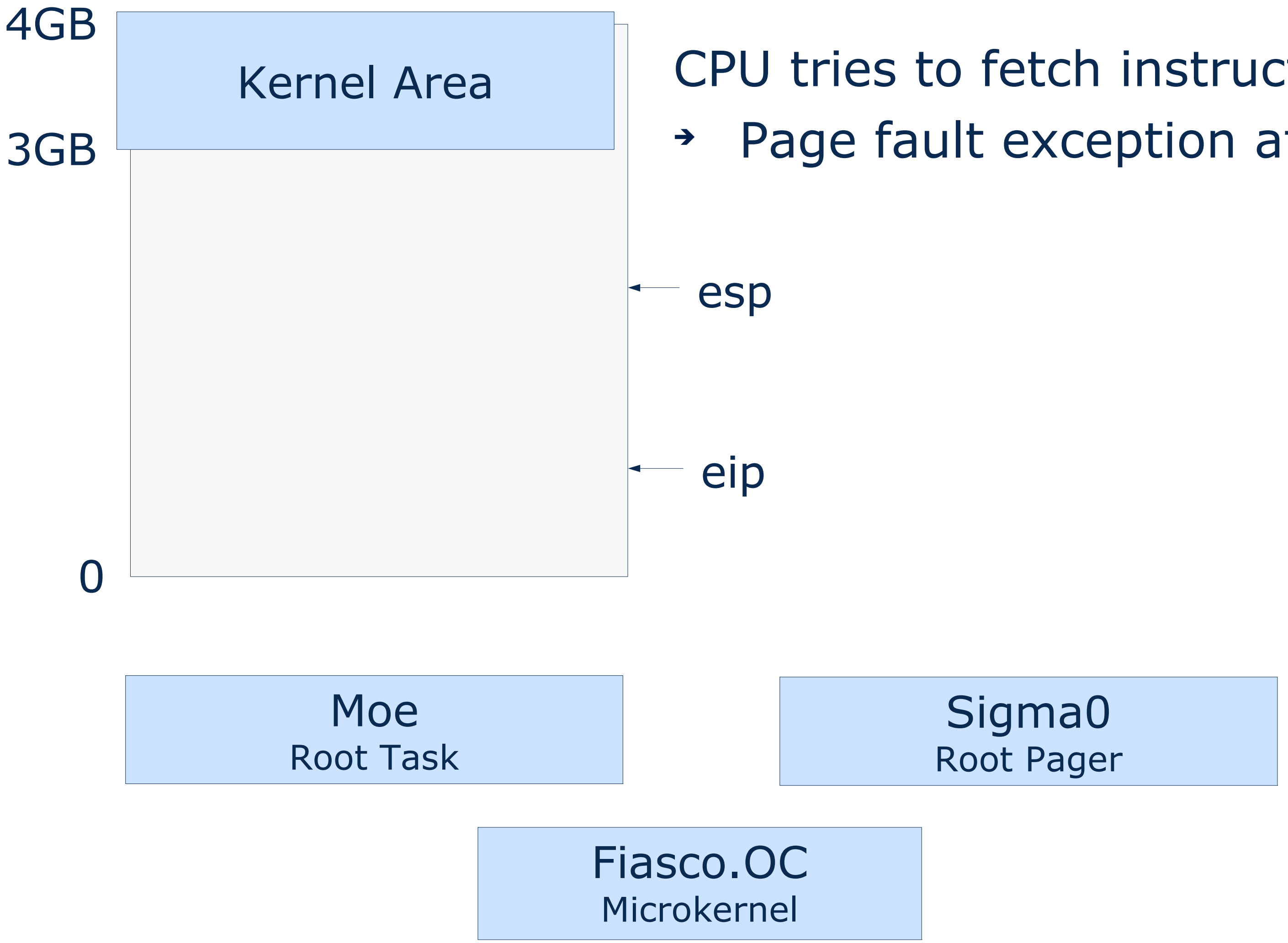
L4::Thread::ex_regs



First thread executes ...

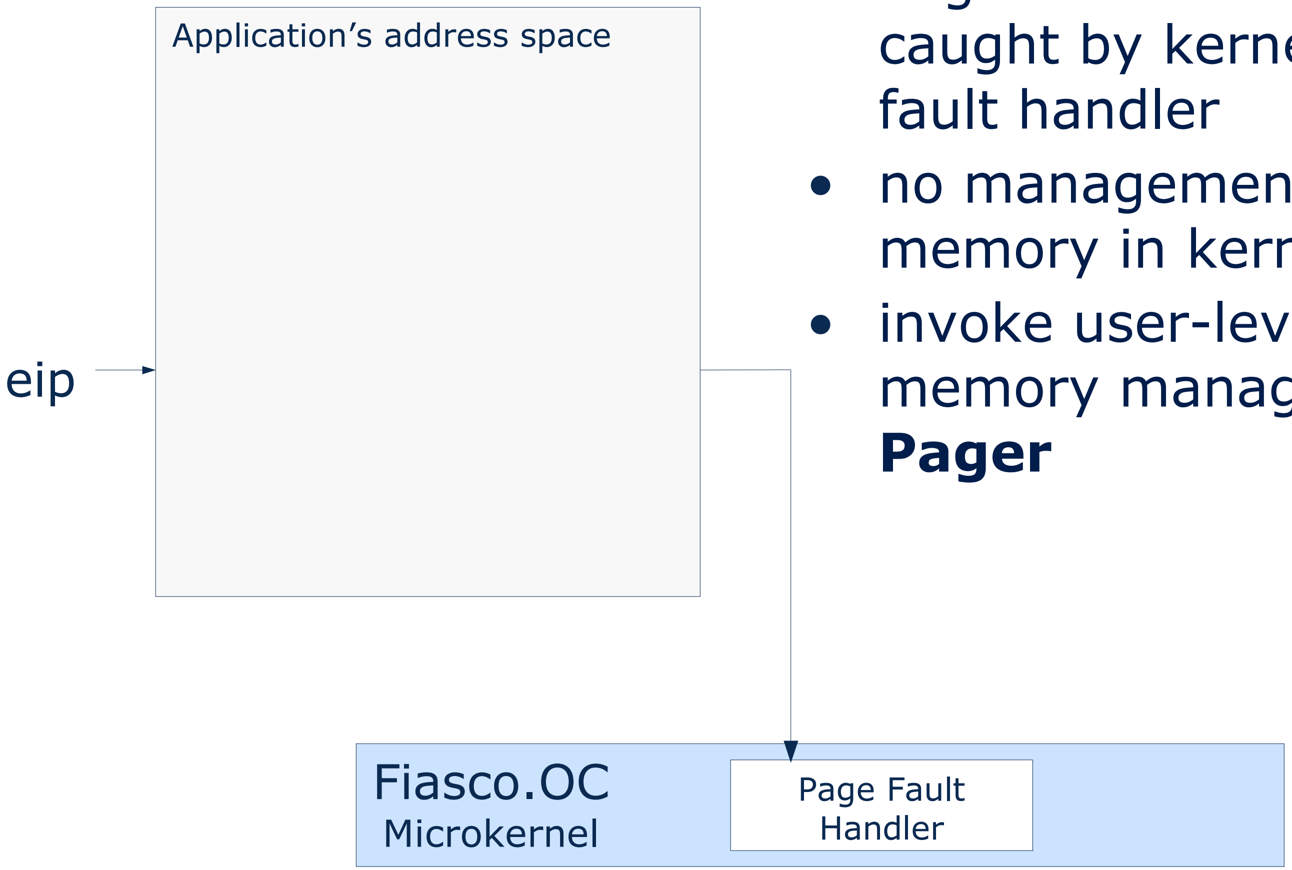
CPU tries to fetch instruction

- Page fault exception at eip



Page Fault Handling

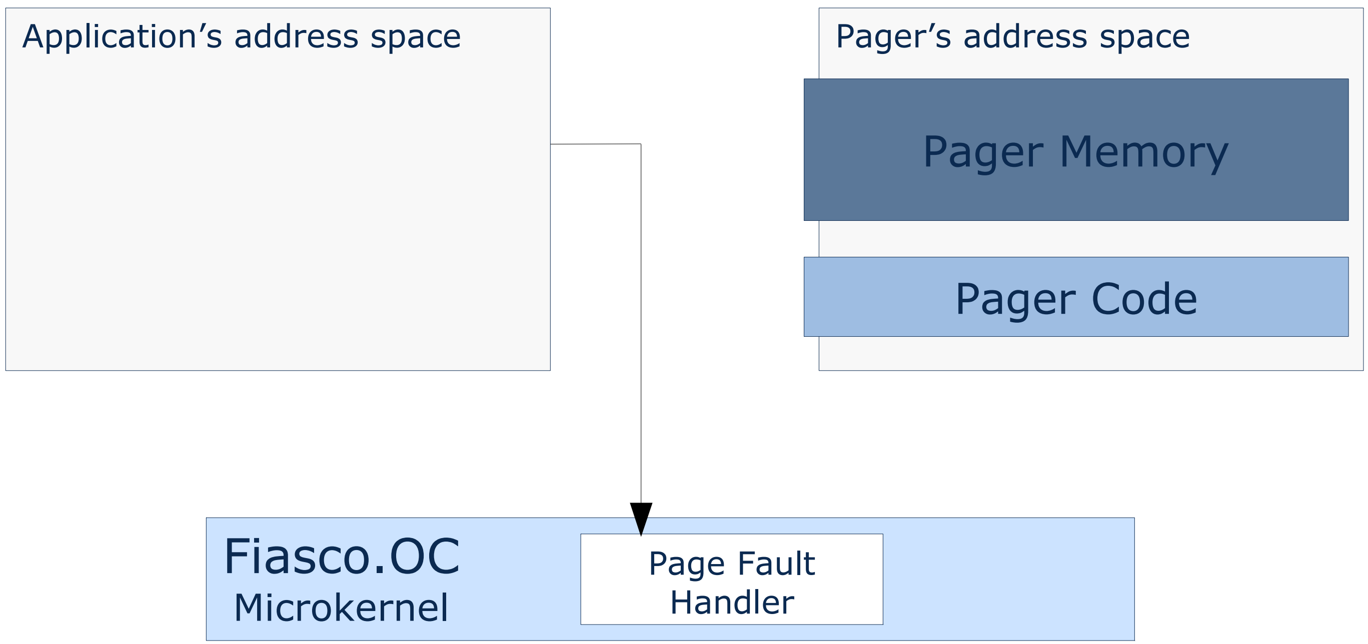
Page-fault handling



- Page fault exception is caught by kernel page-fault handler
- no management of user memory in kernel
- invoke user-level memory management ⇨ **Pager**

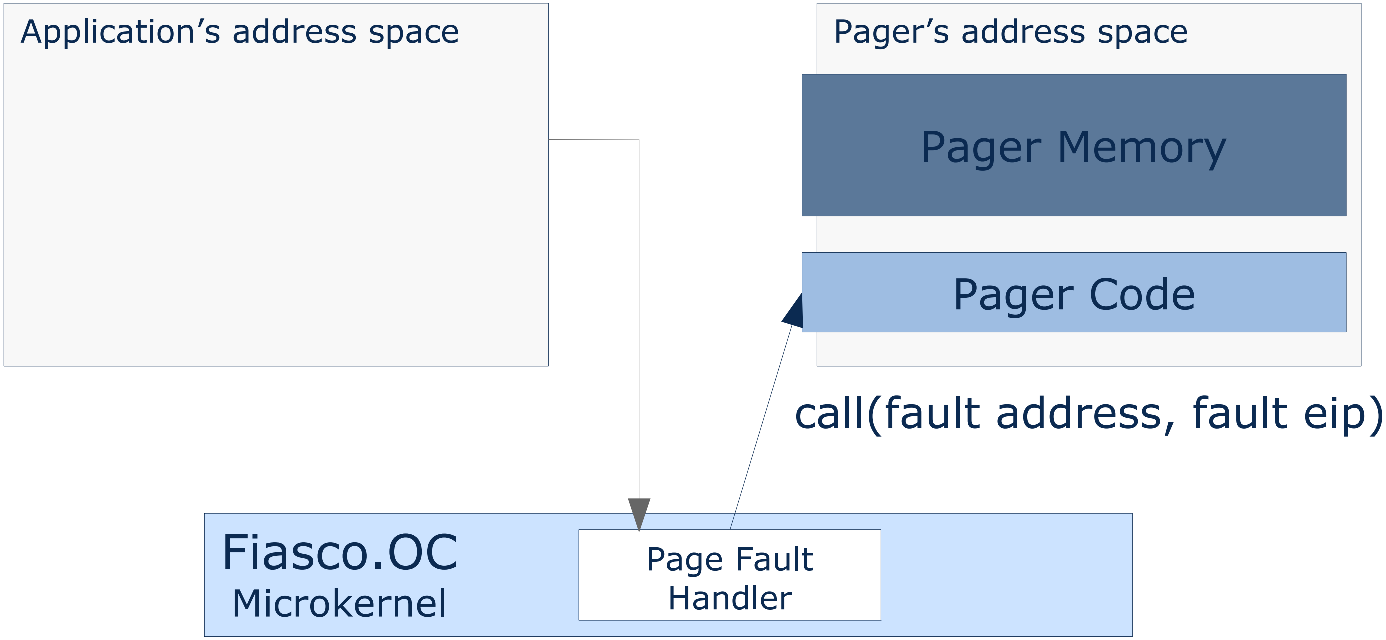
Pager

- thread which is invoked on page fault
- each thread has a (potentially different) pager assigned

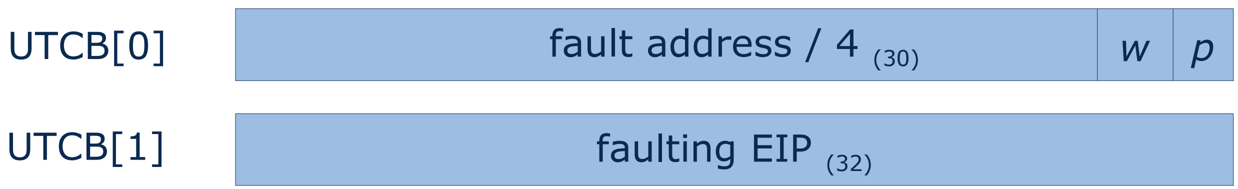


Pager invocation

- communication with pager thread using IPC
- kernel page fault handler sets up IPC to pager
- pager sees faulting thread as sender of IPC



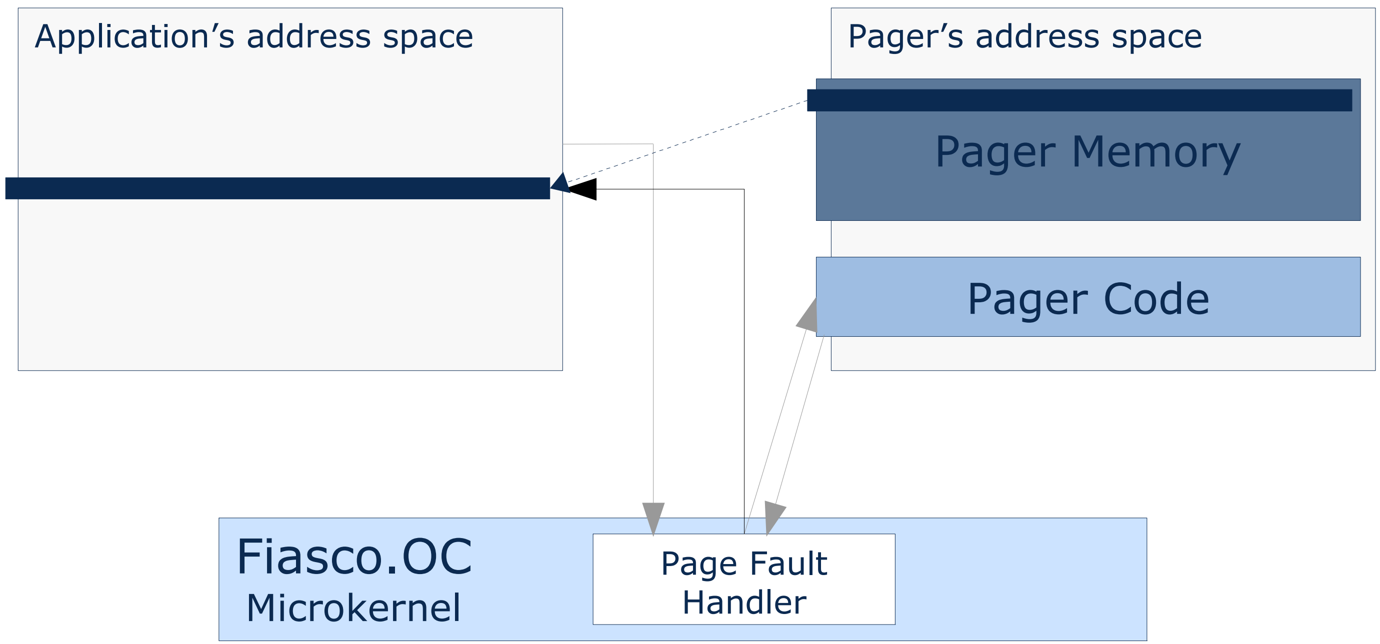
Page-fault IPC



w = 0	read page fault
w = 1	write page fault
p = 0	no page present
p = 1	page present

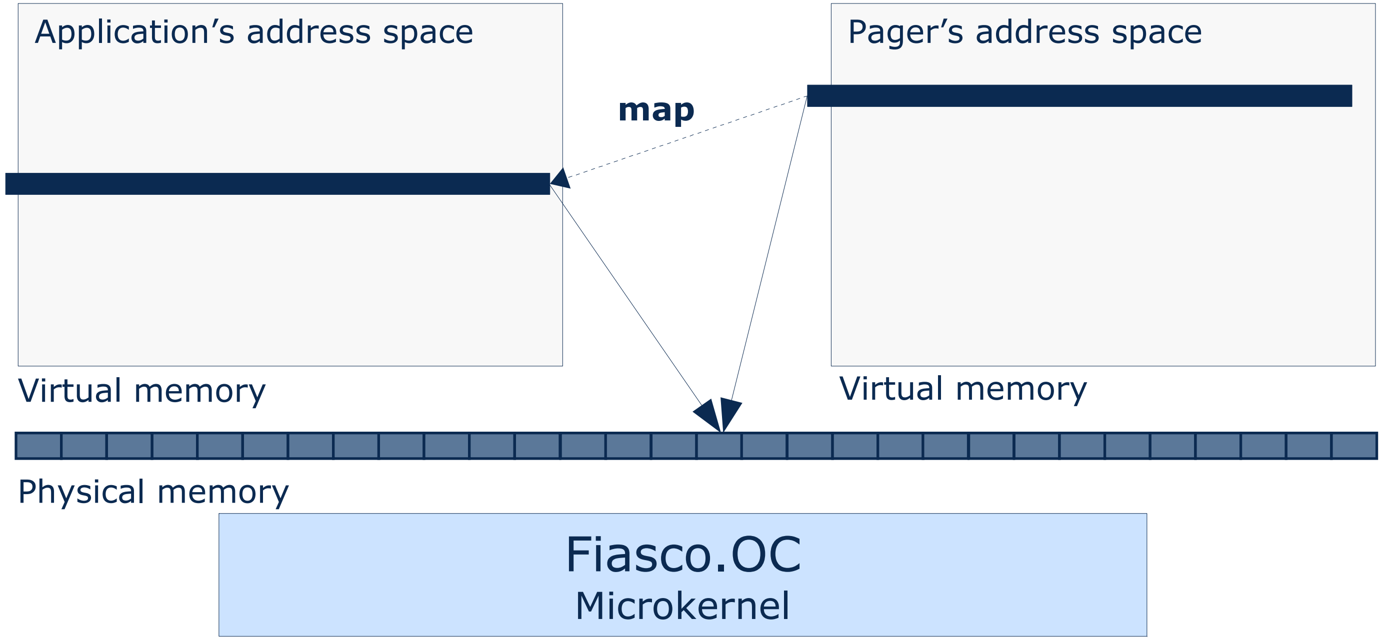
Pager's reply

- pager maps pages of it's own address space to the application's address space
- flexpage IPC enables these mappings



Page mapping

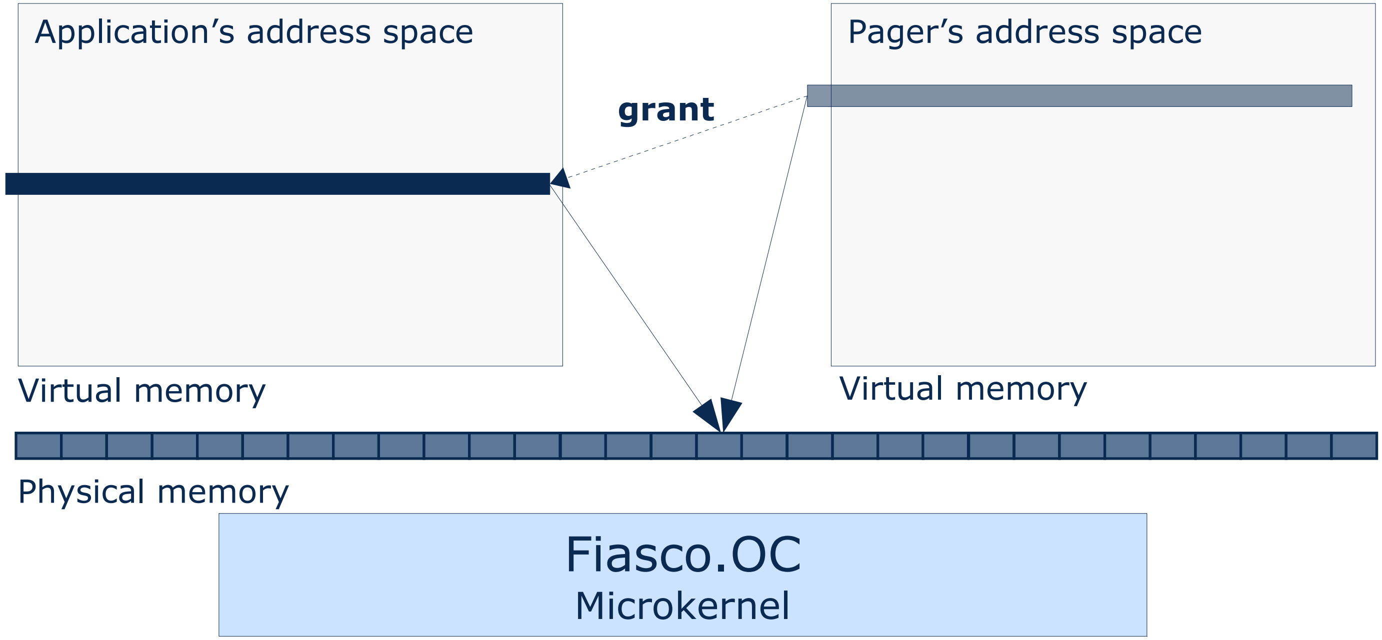
- map creates an entry in the receiver's address space pointing to the same page frame
- only valid pager address space entries can be mapped



Page granting

- Special case: grant pages (flag: L4_FPAGE_GRANT)

➔ Removes mapping from sender's address space

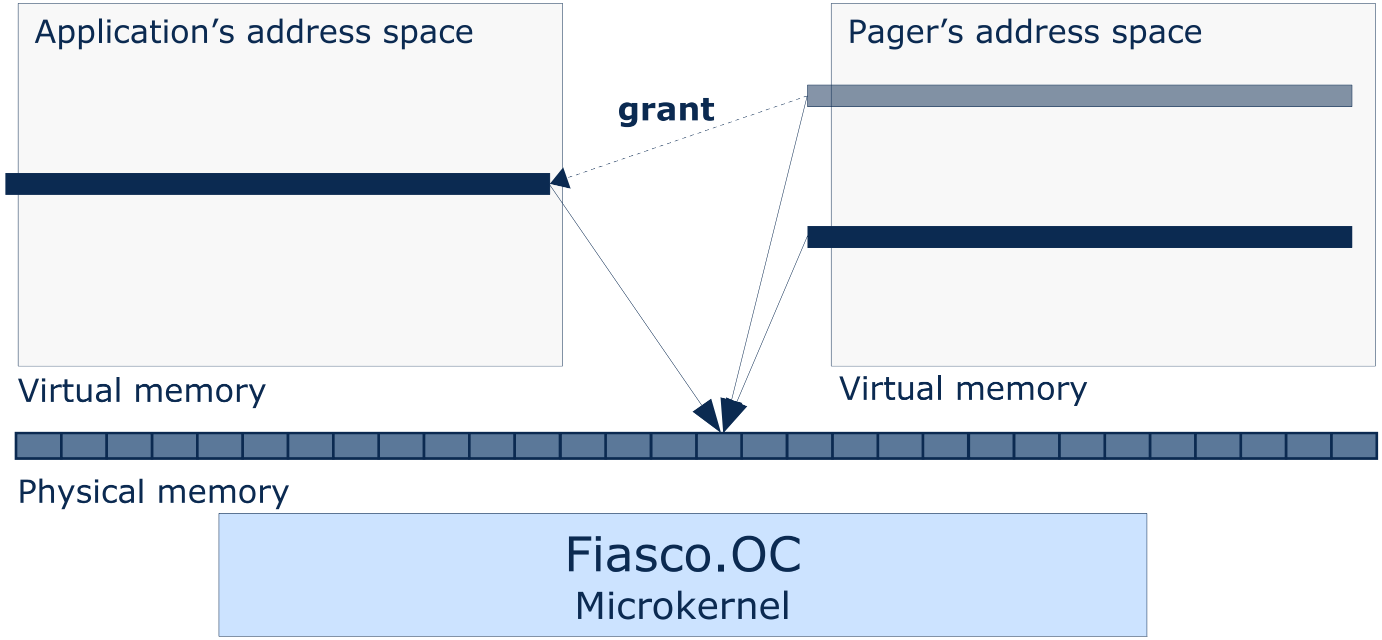


Page granting

- Special case: grant pages (flag: L4_FPAGE_GRANT)

➔ Removes mapping from sender's address space

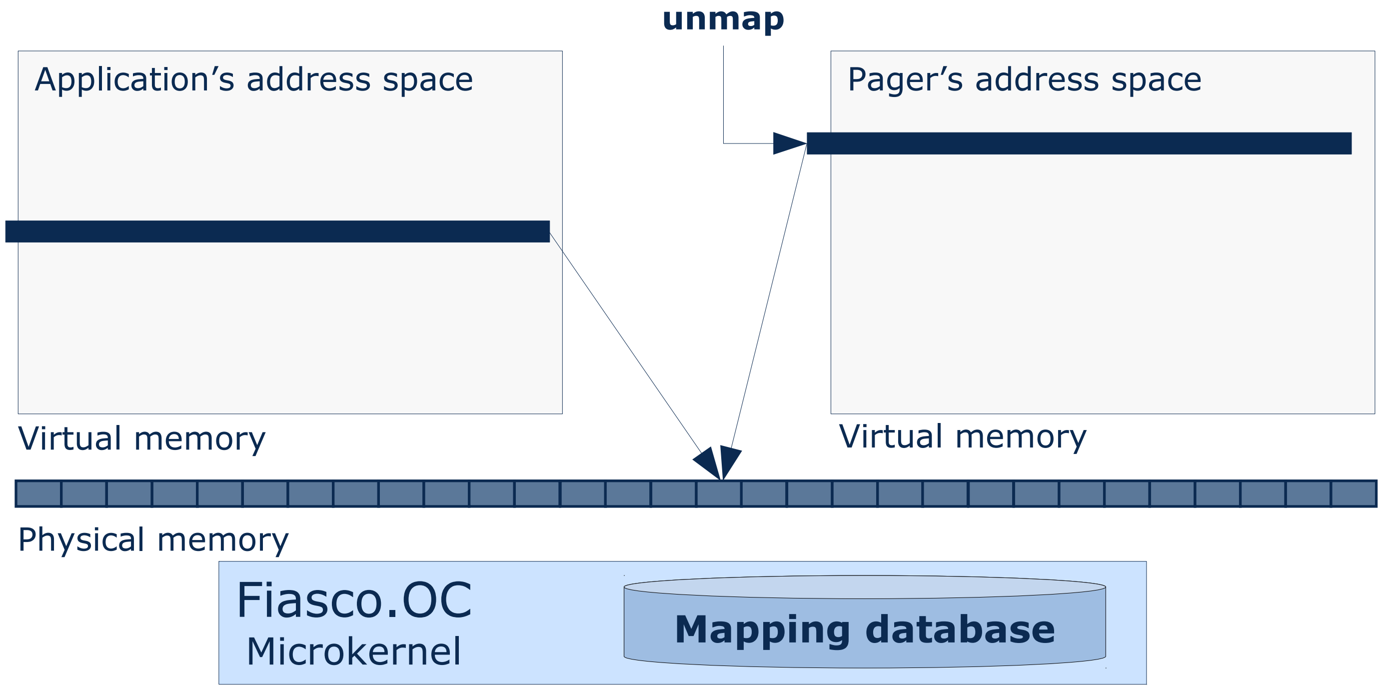
  ➔ ATTENTION: aliases remain



Page unmap

- Removes entries to a page frame (fpage is specified in invoker's address space)

→ Kernel tracks mappings in a database



Flexpages

Flexpage types

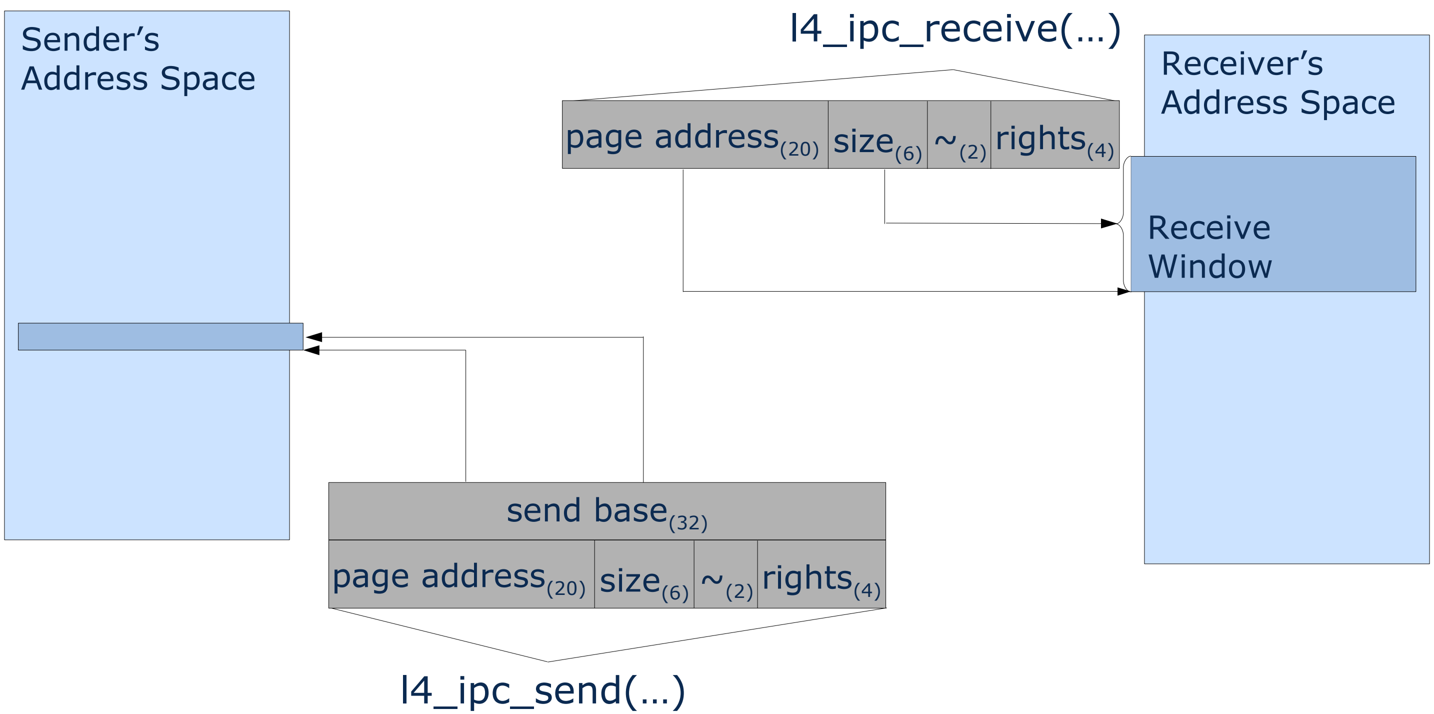
- In general, flexpages represent areas within an address space

- Flexpages in Fiasco.OC are used to describe:
  - Memory pages
  - I/O ports
  - Capabilities

- Today only flexpages for memory pages are described.

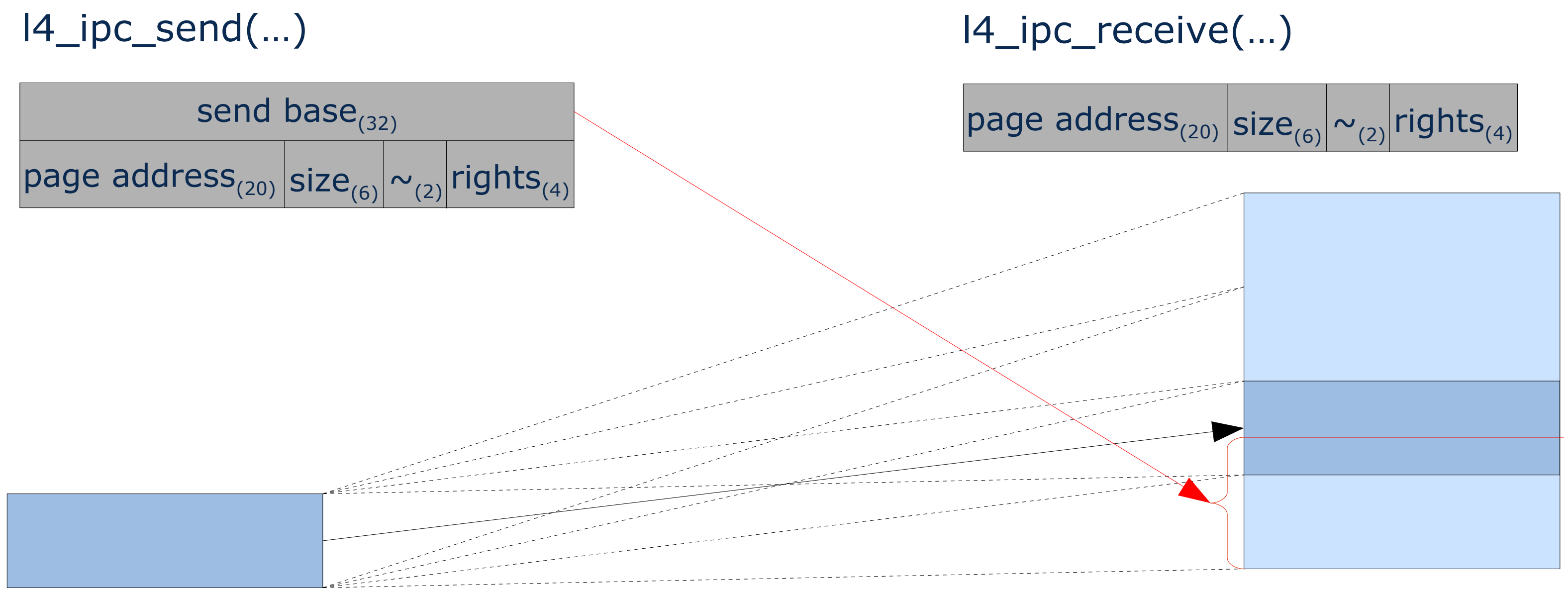
Flexpage IPC in detail

- Size-aligned
- Size 2^{size}, smallest is hardware page
- Source and target area of a map IPC are described by flexpages



Flexpage offset

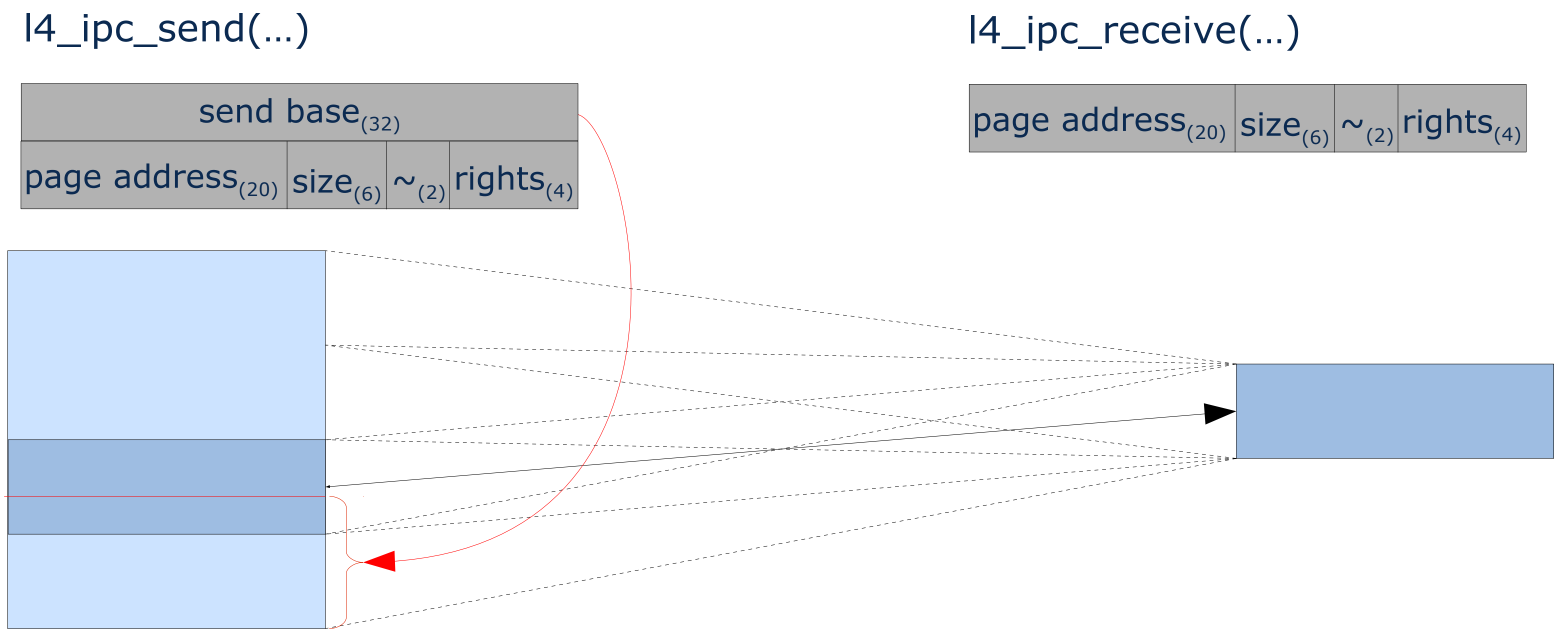
- send flexpage is smaller than the receive window
  - target position is derived from send flexpage alignment and send base



Flexpage offset

- send flexpage is larger than receive window
  - target position is derived from receive flexpage alignment and send base

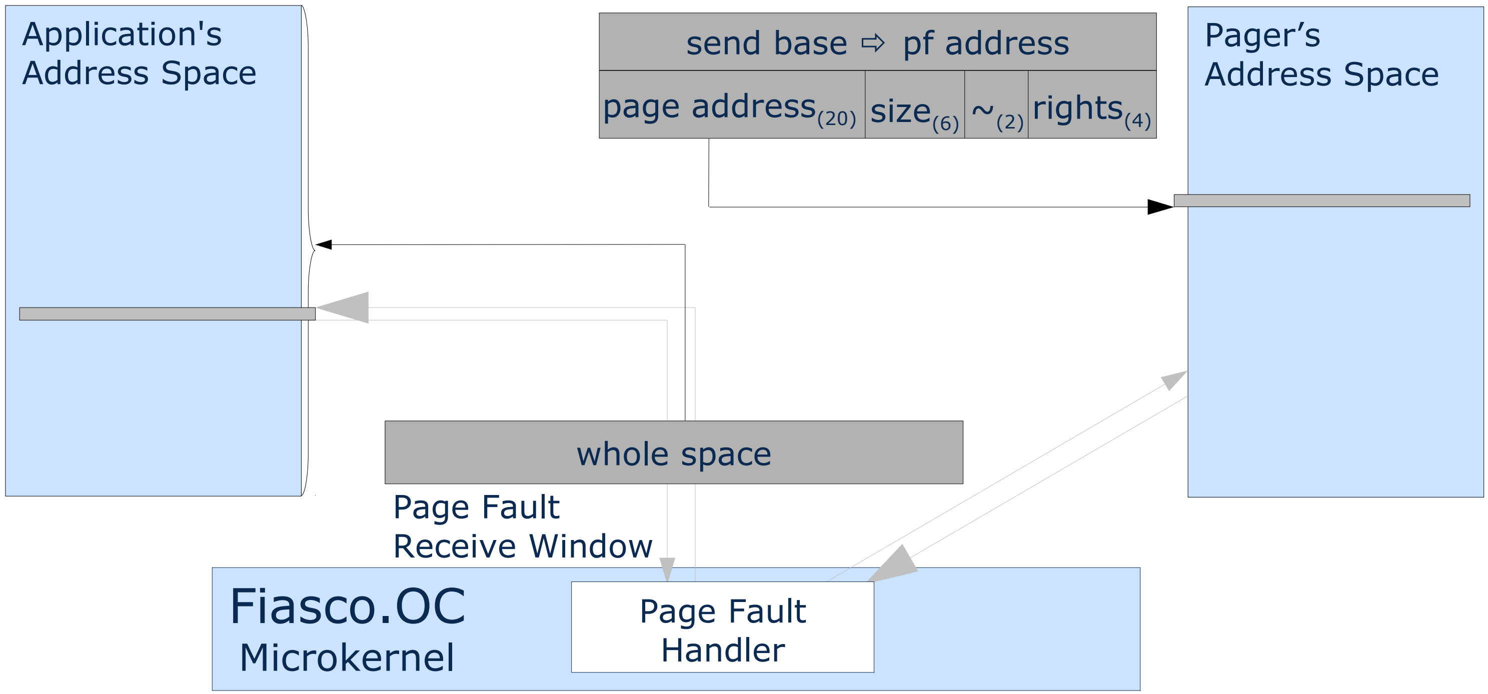
➔ send base depends on information about the receiver



Page fault in detail

- kernel page fault handler sets receive window to whole address space

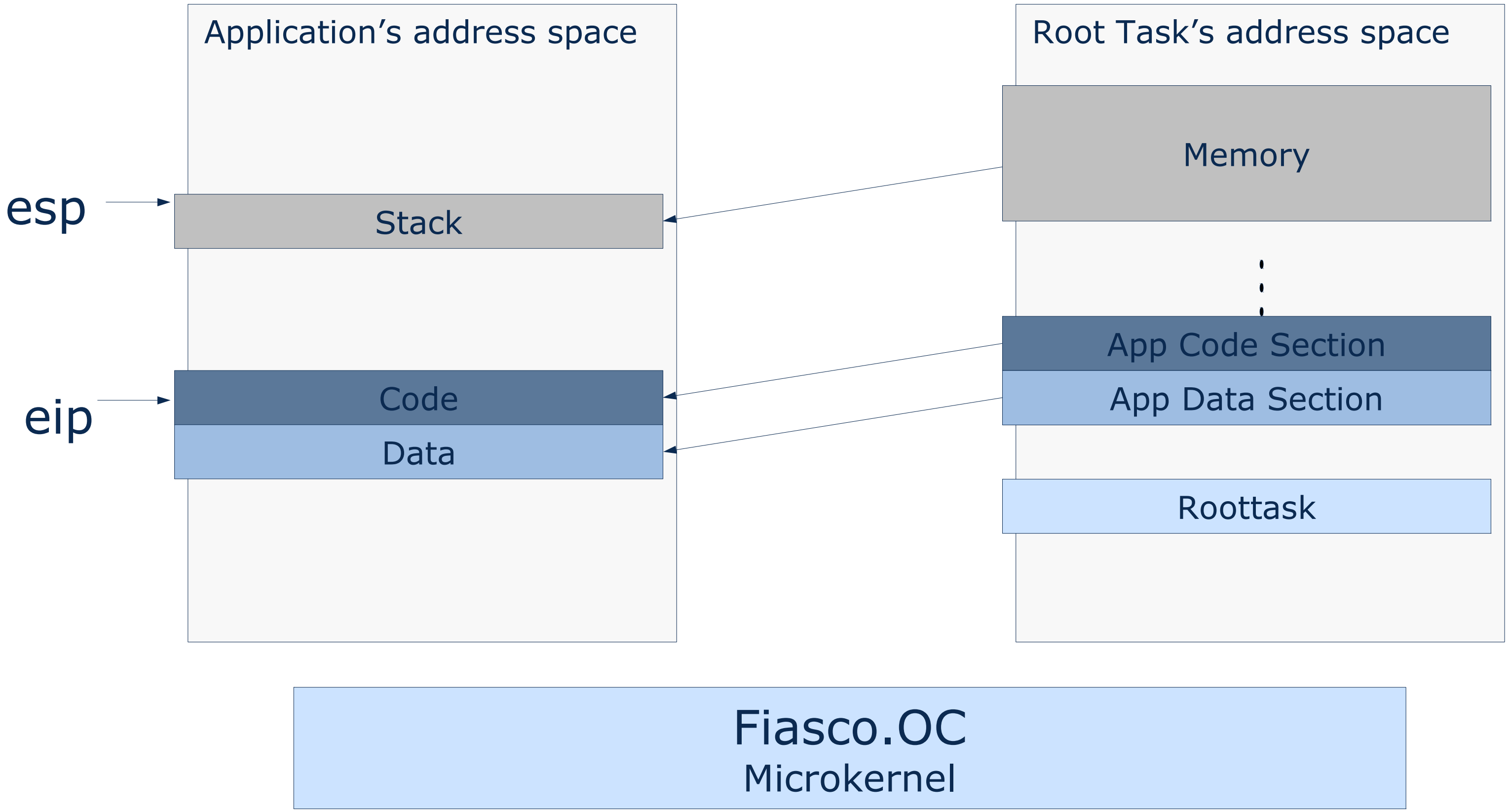
➔ pager can map more than just one page, where the page fault happened to the client



Root task's pager

- pages are mapped as they are needed

➔ *demand paging*



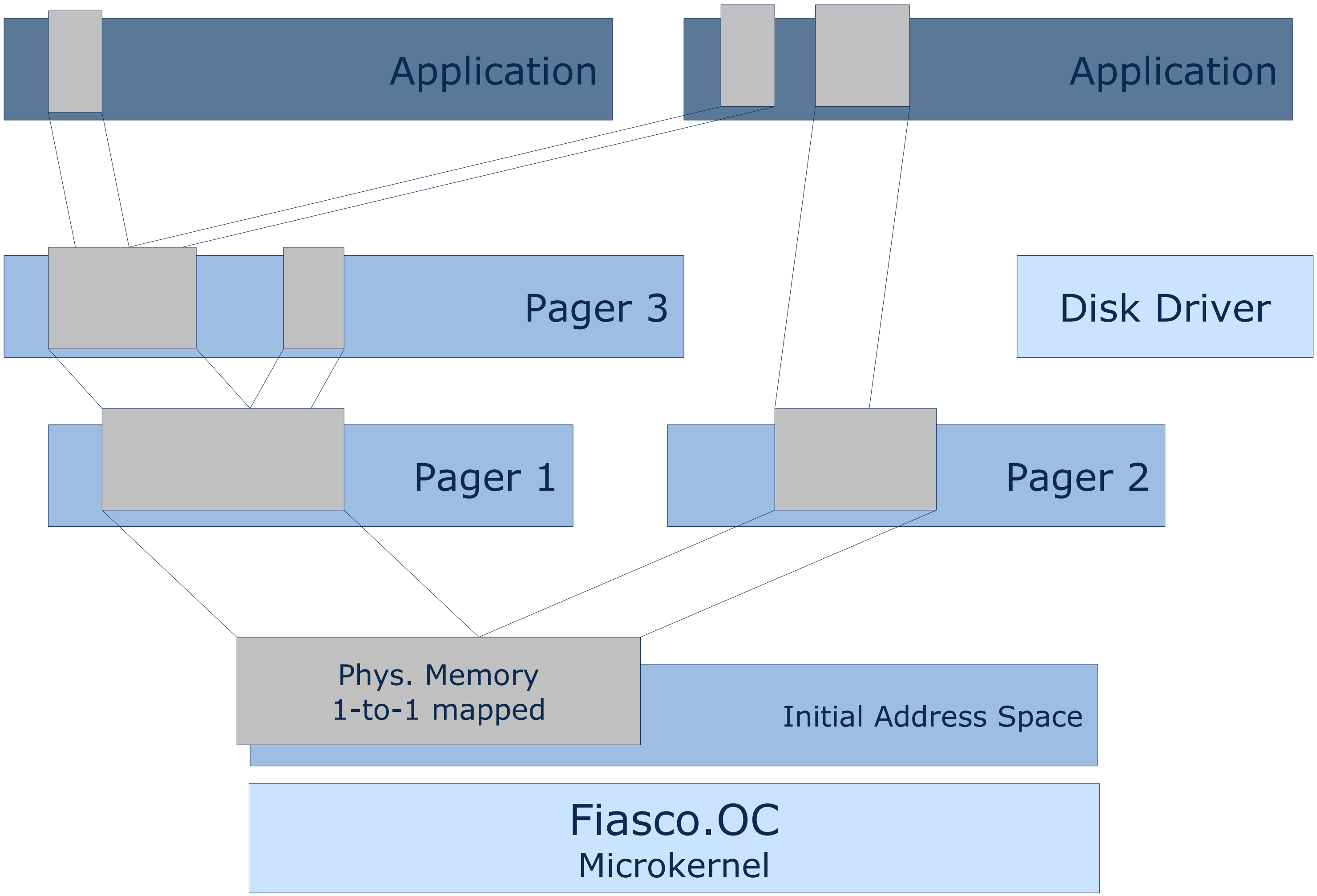
Hierarchical Pagers

Motivation

- initial pager can only implement basic memory management
- no knowlegde about application requirements
  - different requirements at the same time
- missing services for advanced memory management
  - e.g. no disk driver for swapping
- build more advanced pagers on top of the initial one

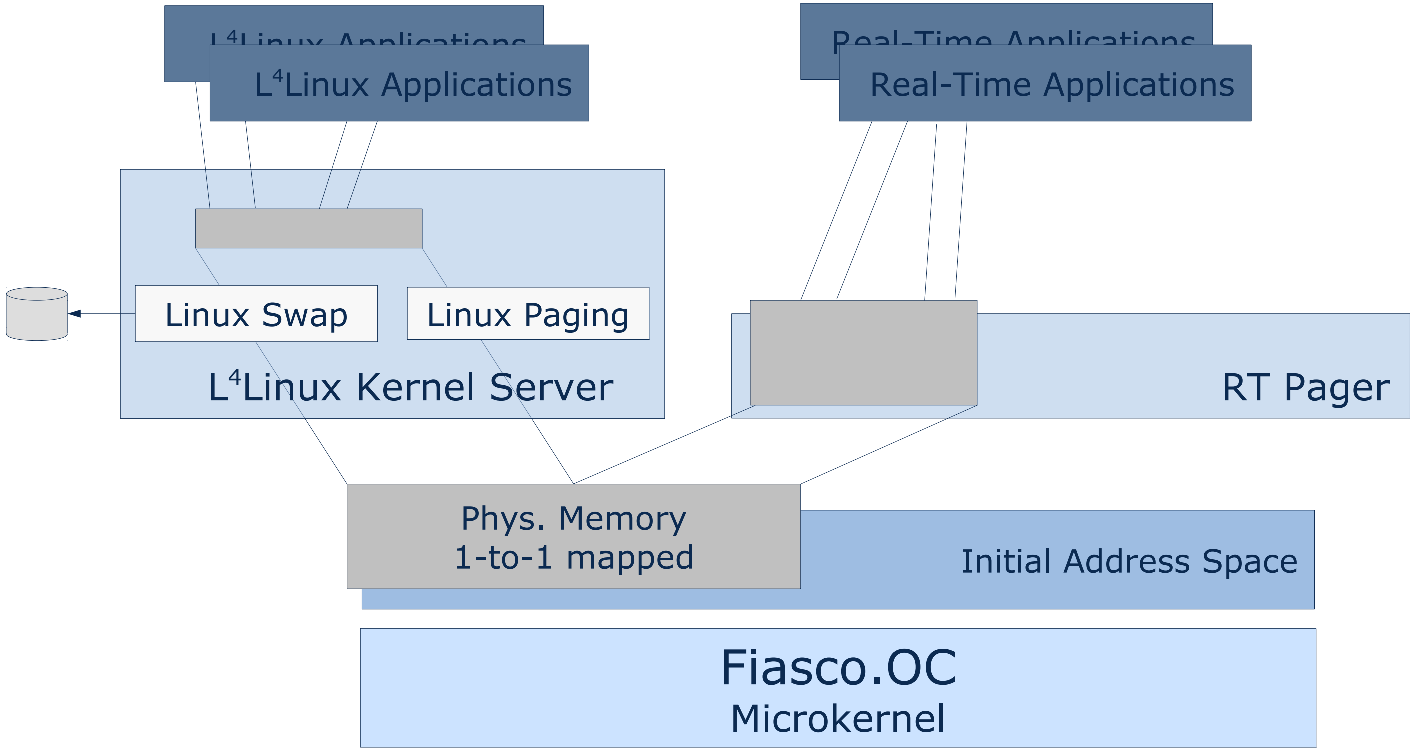
➔ pager hierarchy

Pager hierarchy



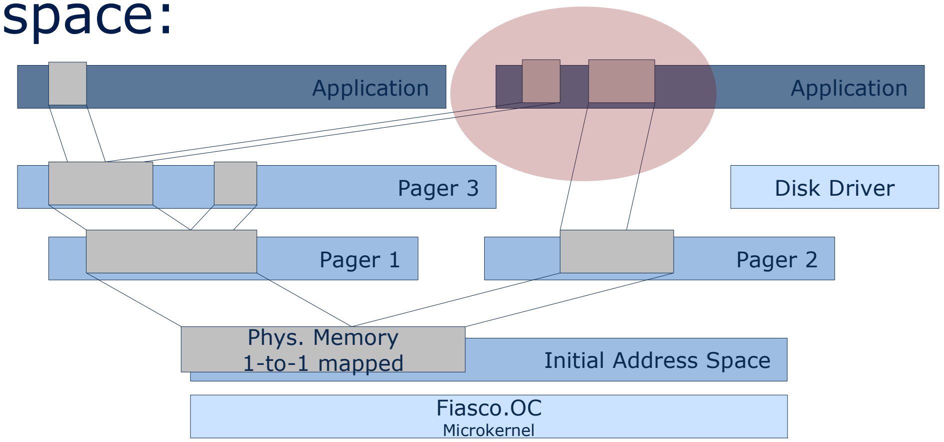
Real world example

- L^4Linux implements Linux paging policy
- RT pager implements real-time paging policy (e.g. no swapping)



Virtual memory management

- pager has to specify send base
- pager needs to know client's address space layout
  - no problems with only one pager (e.g. L4Linux)
- possible conflicts if more than one pager manages an address space:

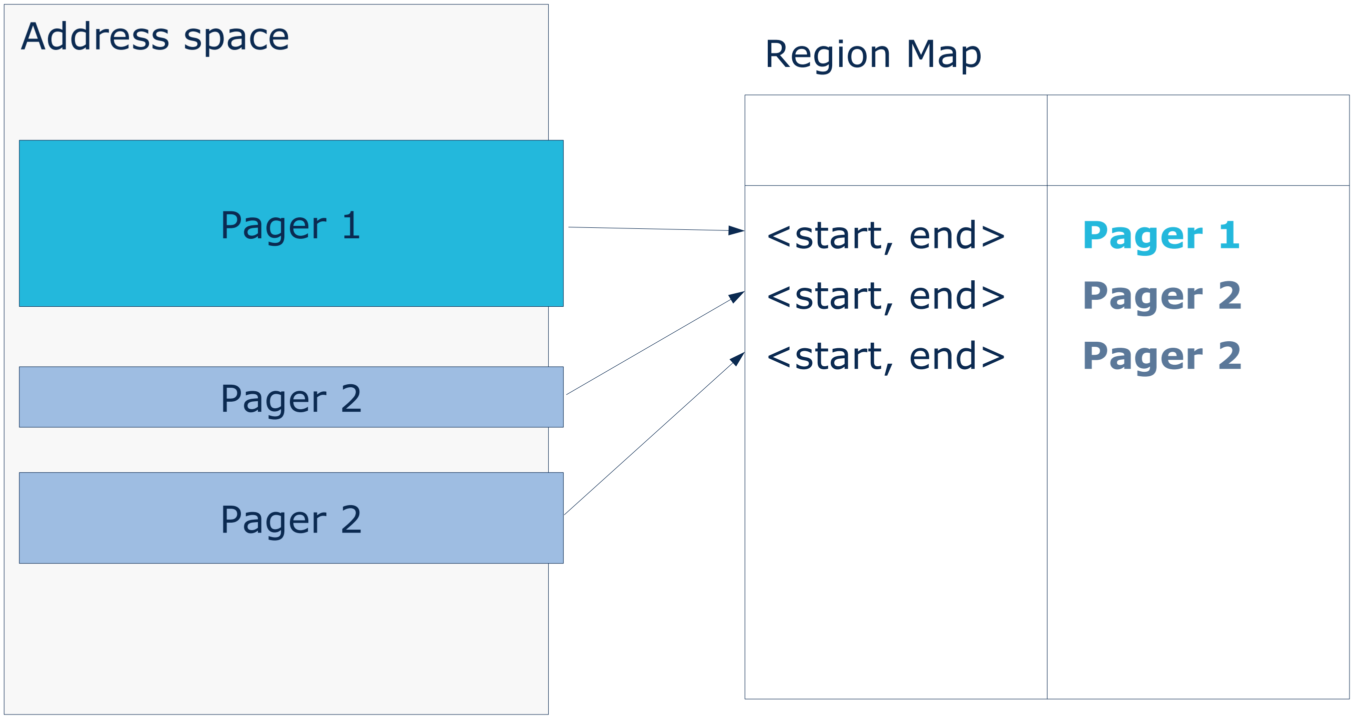


➔ virtual memory must be managed independent of pagers

Region Mapper

Region Map

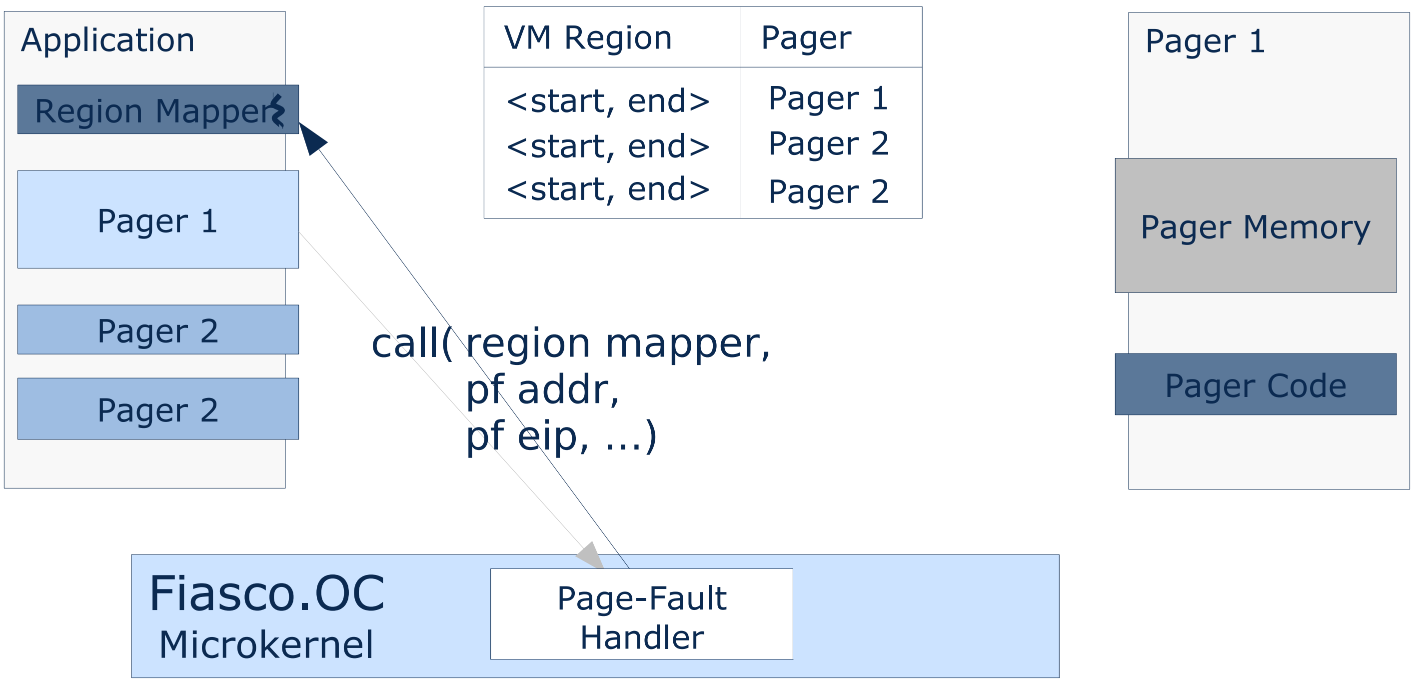
- per address space map that keeps track which part of the address space is managed by which pager



Region mapper

- intermediate pager that identifies which pager should handle a page fault
- can reside in the application's address space

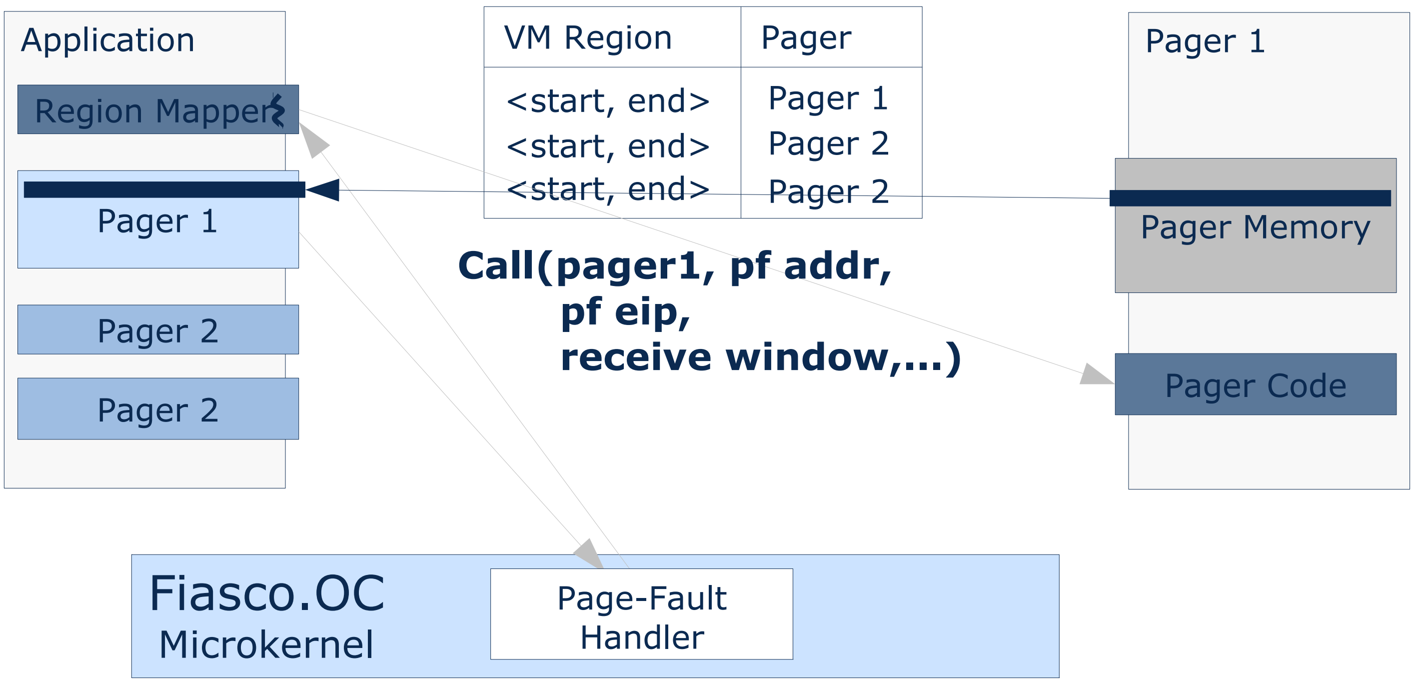
➔ region mapper has to be pager of all thread of a task



Region mapper

- region mapper calls the pager that is responsible
- receive window gets restricted to the area managed by that pager

➔ no interference between different pagers



High-level memory abstraction

- memory management in terms of pages so far
- application's view to memory
  - code / data sections
  - memory mapped files
  - anonymous memory (heaps, stacks, …)
  - network / file system buffers
  - …

➔ abstraction to map this view to low-level memory management

Dataspaces

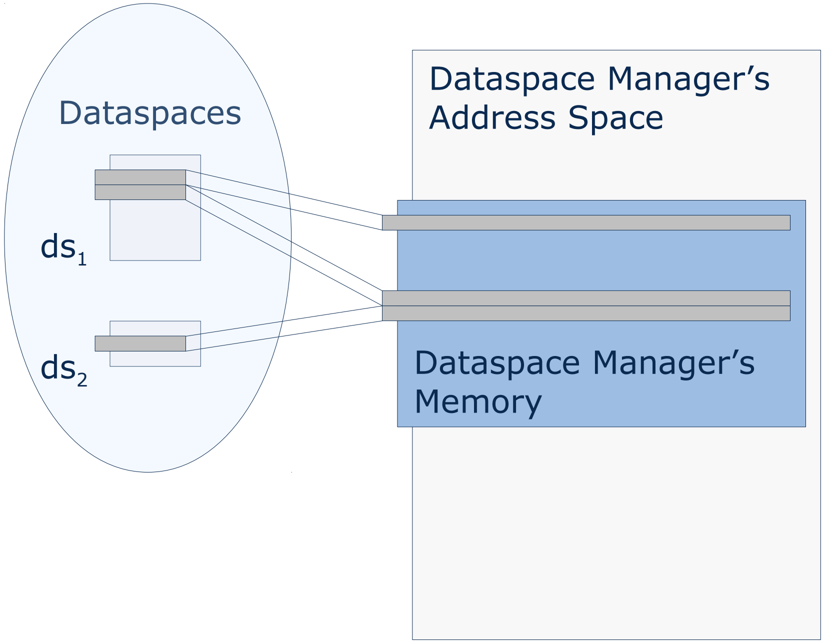
Dataspaces

- Dataspace: *unstructured data container*
- abstraction for anything that contains data:
  - Files
  - Anonymous memory
  - I/O adapter memory
  - …
- Dataspaces are implemented by *Dataspace Managers*
- Dataspaces can be attached to regions of an address space

Dataspace manager

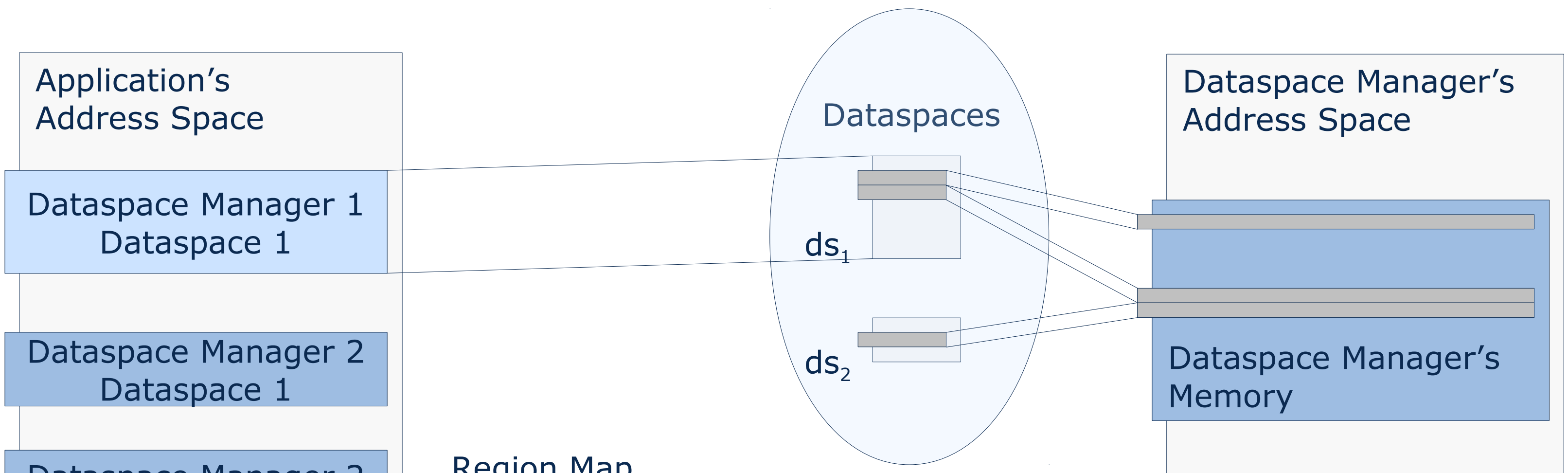
- a Dataspace Manager determines the semantic of a dataspace
- each Dataspace Manager is the pager for its dataspaces

➔ implements the paging policy (page replacement etc.)



Dataspaces & region mapper

- region map keeps track which dataspaces are attached to which virtual memory regions
- region mapper translates page faults to dataspace offsets



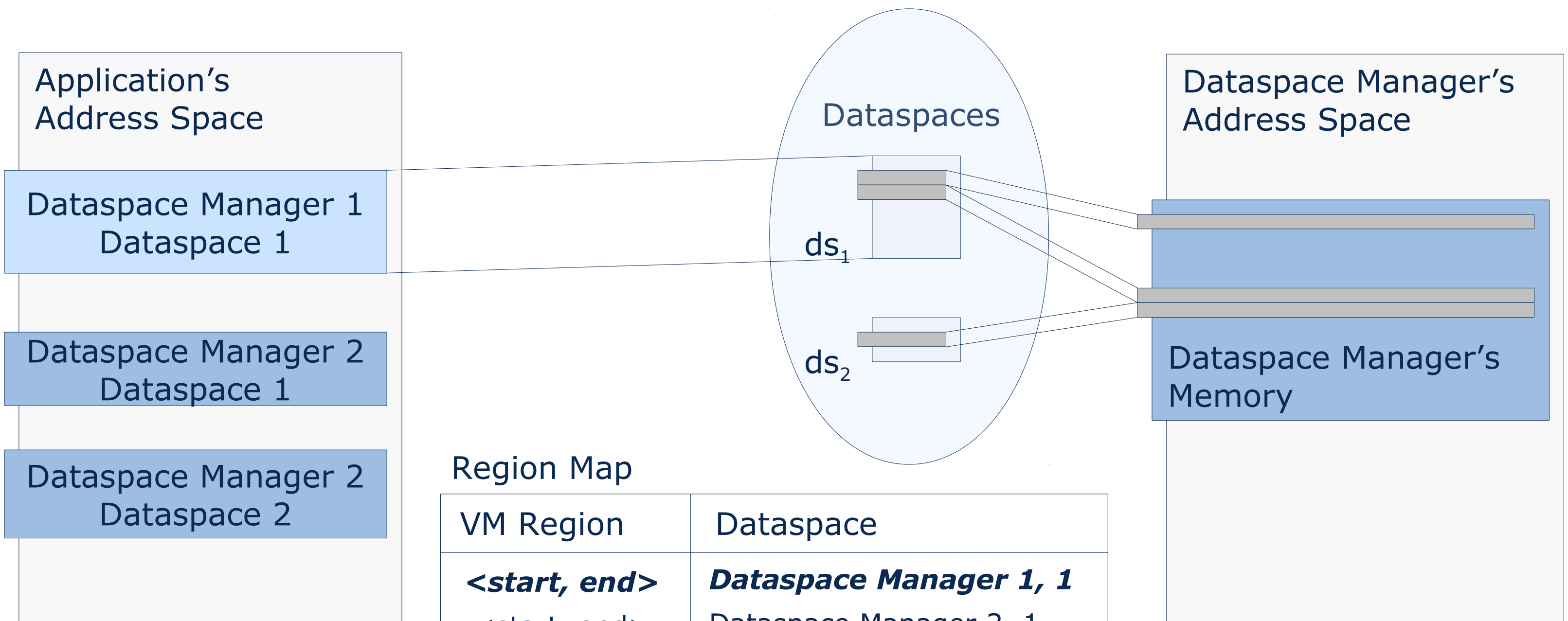
Region Map

VM Region	Dataspace
<start, end>	***Dataspace Manager 1, 1***
<start, end>	Dataspace Manager 2, 1
<start, end>	Dataspace Manager 2, 2

Dataspaces & region mapper

- region mapper propagates fault to dataspace manager's fault handler

➔ dataspace fault (ds_manager_id, ds_id, offset)



Region Map

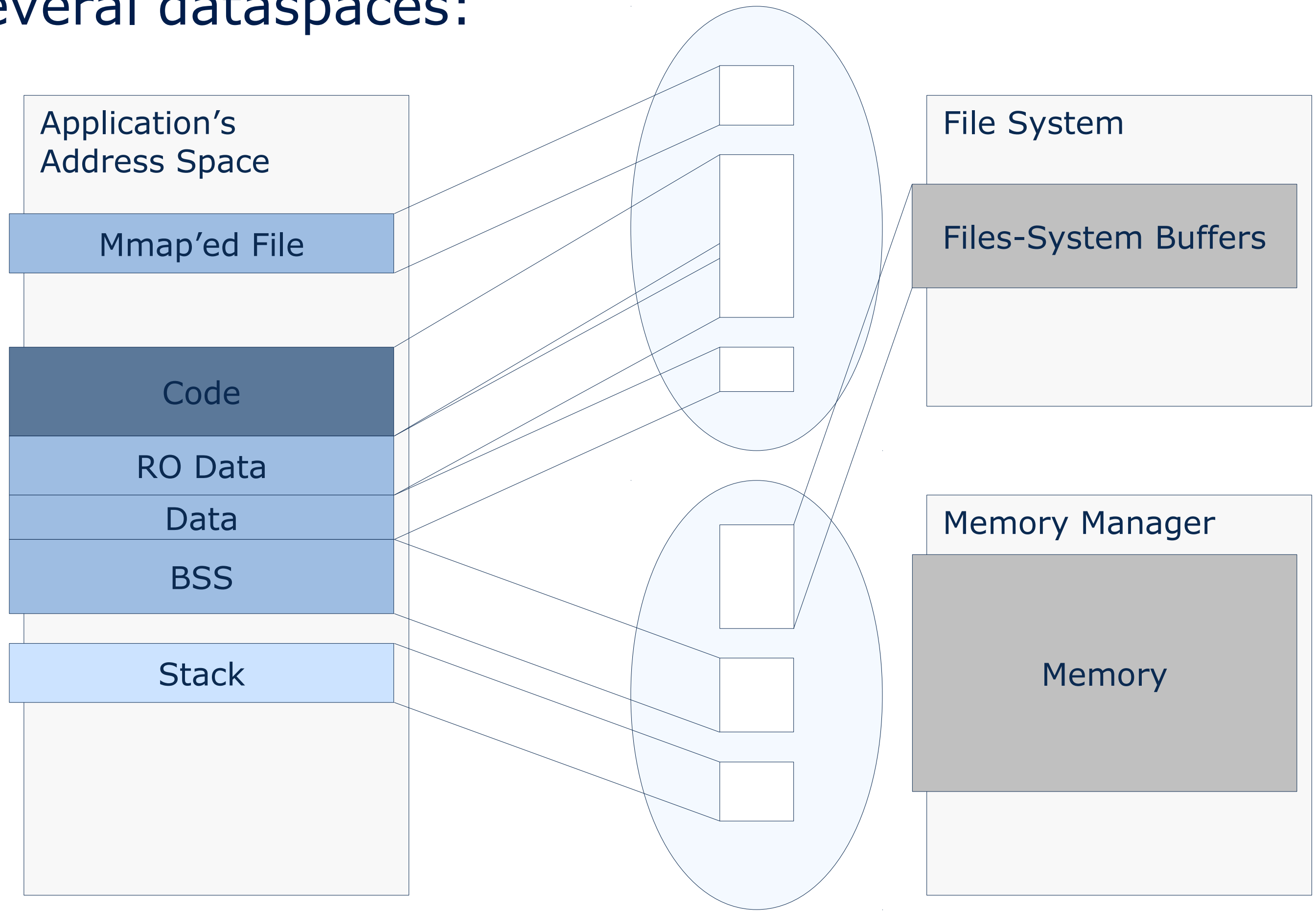
VM Region	Dataspace
<start, end>	***Dataspace Manager 1, 1***
<start, end>	Dataspace Manager 2, 1
<start, end>	Dataspace Manager 2, 2

Application's view to dataspaces

- allocate / free dataspaces
  - create / destroy dataspace
  - semantic depends on dataspace type:
    - anonymous memory: open (size)
    - file: open (filename, mode, …)
    - ...
- attach / detach dataspace
  - create / remove entry in region map
  - ➔ Makes dataspace contents accessible to application
- propagate capability
  - grant access rights to other applications
  - ➔ very easy shared memory implementation

Application address spaces

- application address spaces are constructed from several dataspaces:



Other issues in memory management

- page Allocation Algorithms
  - list-based algorithms, bitmaps, trees, …
- page Replacement Algorithms
  - Least-Recently-Used (LRU)
  - Working Sets
  - Clock
  - …

➔ both page allocation and page replacement are implemented by dataspace managers

➔ can have different strategies for the dataspaces of an application

Memory sharing

- memory sharing important for
  - shared libraries
  - data transfer between system components
  - …
- different types of sharing
  - full sharing, all clients see modifications
  - ➔ easy to implement, pager / dataspace manager grants access rights to pages / dataspaces
  - lazy copying of dataspaces
  - ➔ copy-on-write

Summary

- closer look on tasks/threads:
  - creation
  - page-fault handling
- flexpages
  - memory pages, I/O ports, Capabilities
  - structure
  - offset computation
- pager hierarchy
- region mapper & dataspaces

More information

- Flexpages

  H.Härtig, J.Wolter, J.Liedtke: "*Flexible sized page objects*" , http://os.inf.tu-dresden.de/papers_ps/flexpages.pdf

- Dataspaces

  Mohit Aron, Yoonho Park, Trent Jaeger, Jochen Liedtke, Kevin Elphinstone, Luke Deller: "*The SawMill Framework for VM Diversity*", ftp://ftp.cse.unsw.edu.au/pub/users/disy/papers/Aron_PJLED_01.ps.gz

Next Lesson

- next lecture (2.11.) on 'Communication':
  - IPC flavors
  - communication control
- next exercise (2.11.) on:
  - Practical exercise, computer pool