TECHNISCHE UNIVERSITÄT DRESDEN

**Faculty of Computer Science** Institute of Systems Architecture, Operating Systems Group

# MEMORY

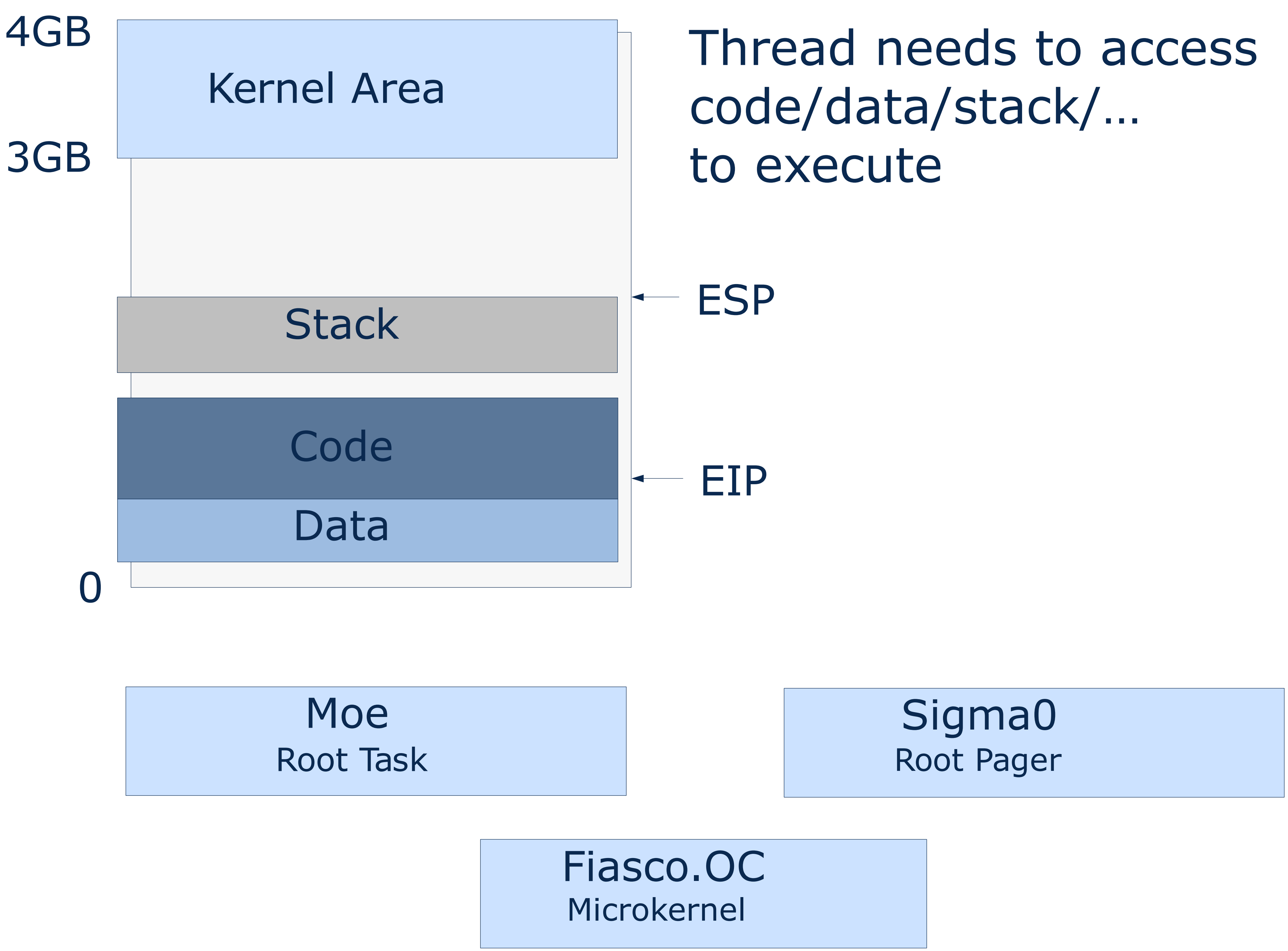
**MICHAEL ROITZSCH**

# AGENDA

- Introduction
  - Monolithic vs. microkernels
  - L4 concepts: Threads and IPC
  - Fiasco.OC/TUDOS introduction
- **Today: Memory Management**
  - Task creation
  - Page-fault handling
  - Flexpages
  - Hierarchical pagers
  - Region manager
  - Dataspaces

# TASK CREATION

# ADDRESS SPACE




Thread needs to access code/data/stack/… to execute

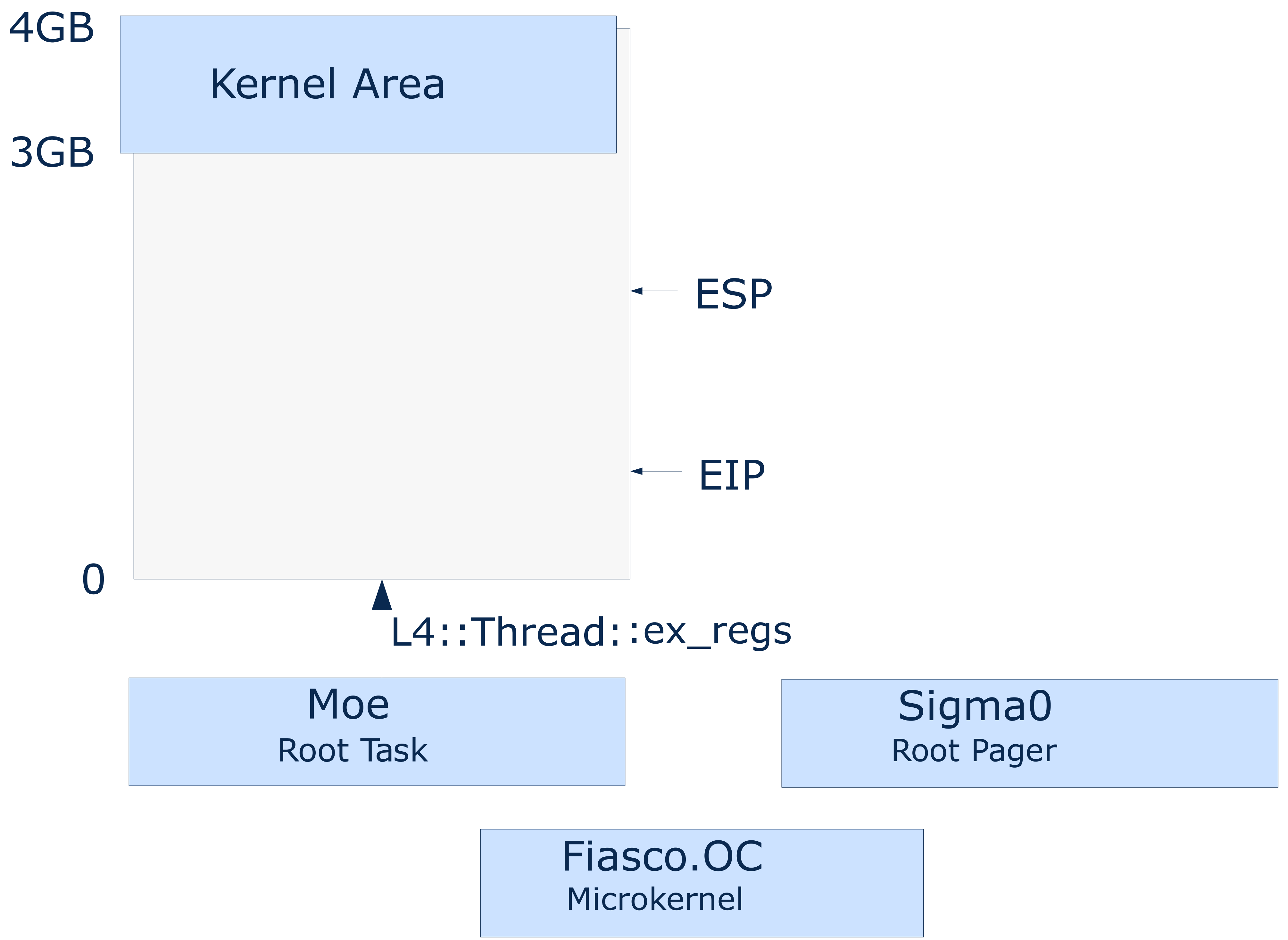
# CREATION

```
/* 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()
)
```

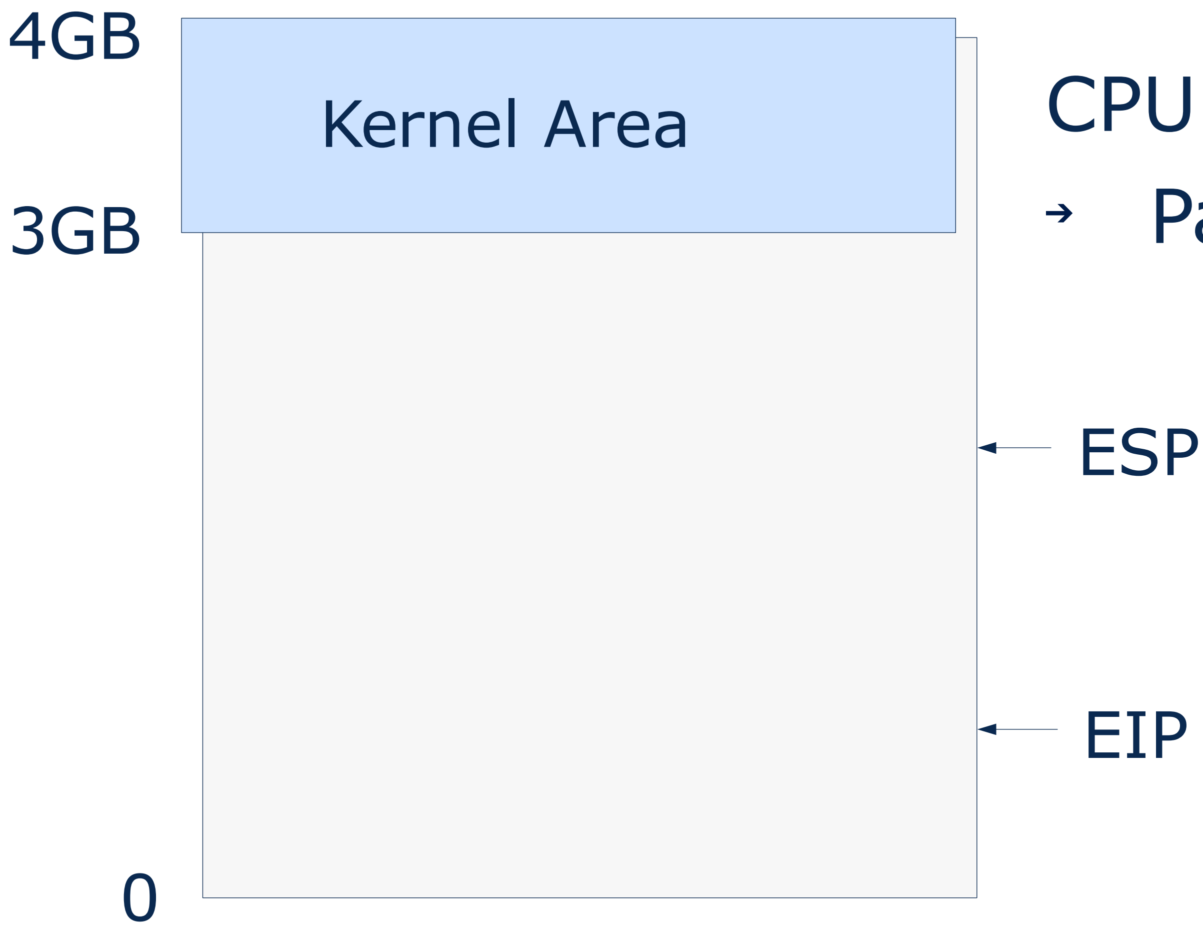
# CREATION

```
/* 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()
)
```

# EX_REGS

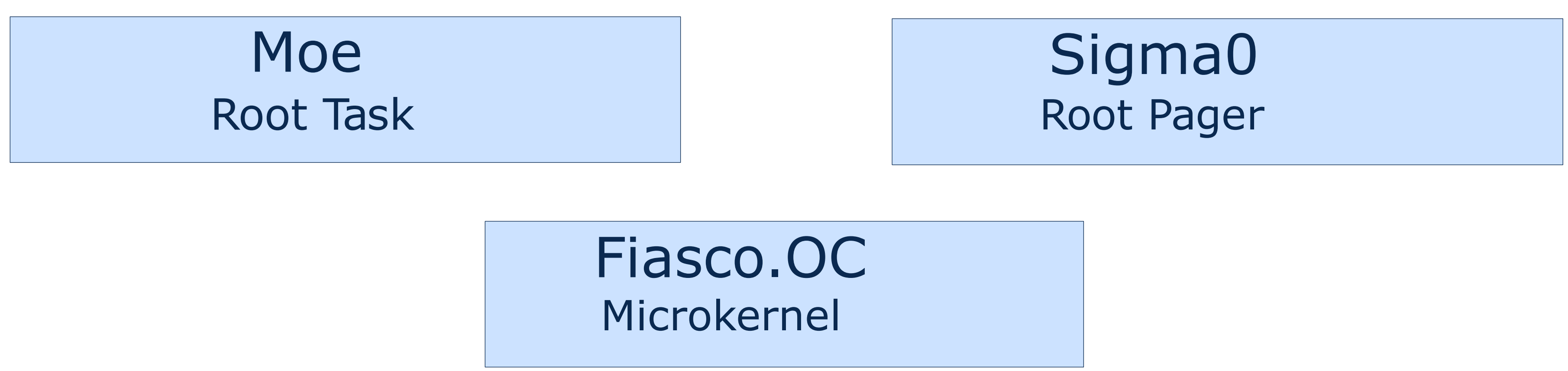

4GB
Kernel Area
3GB
ESP
EIP
0
L4::Thread::ex_regs
Moe
Root Task
Sigma0
Root Pager
Fiasco.OC
Microkernel

# EX_REGS



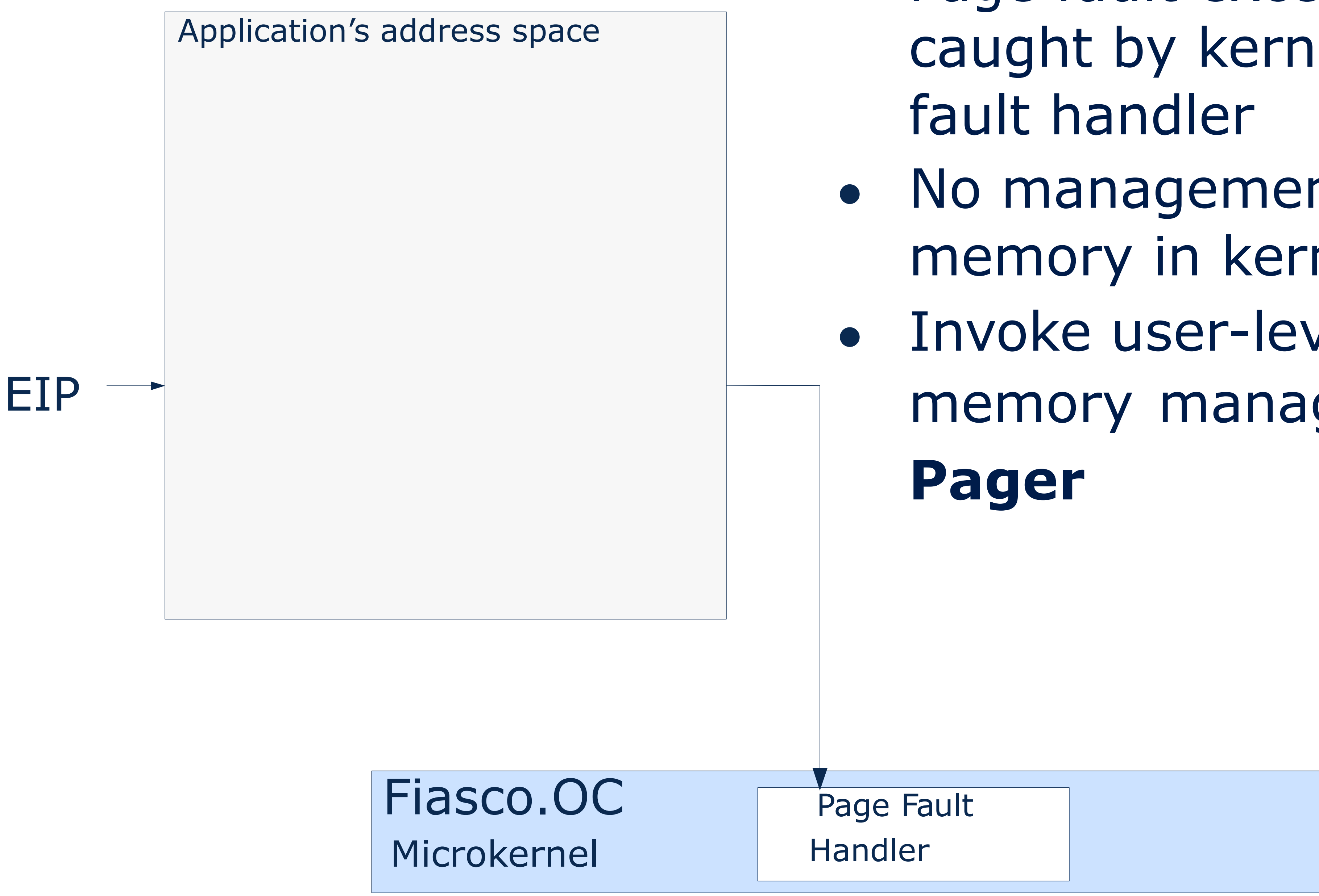

CPU tries to fetch instruction

→ Page fault exception at EIP



# PAGE FAULT HANDLING

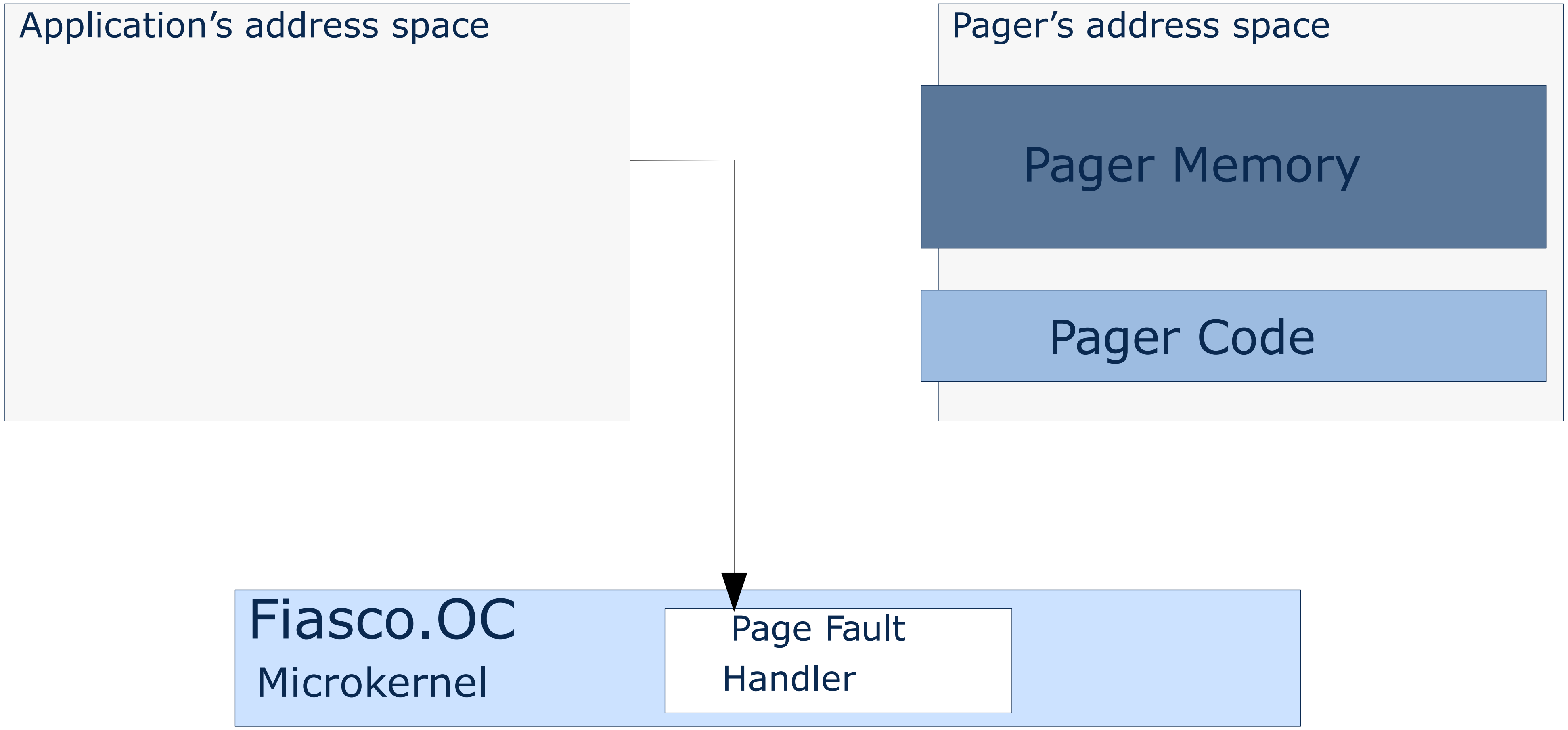
# PAGE FAULT




- Page fault exception is caught by kernel page-fault handler
- No management of user memory in kernel
- Invoke user-level memory management → **Pager**

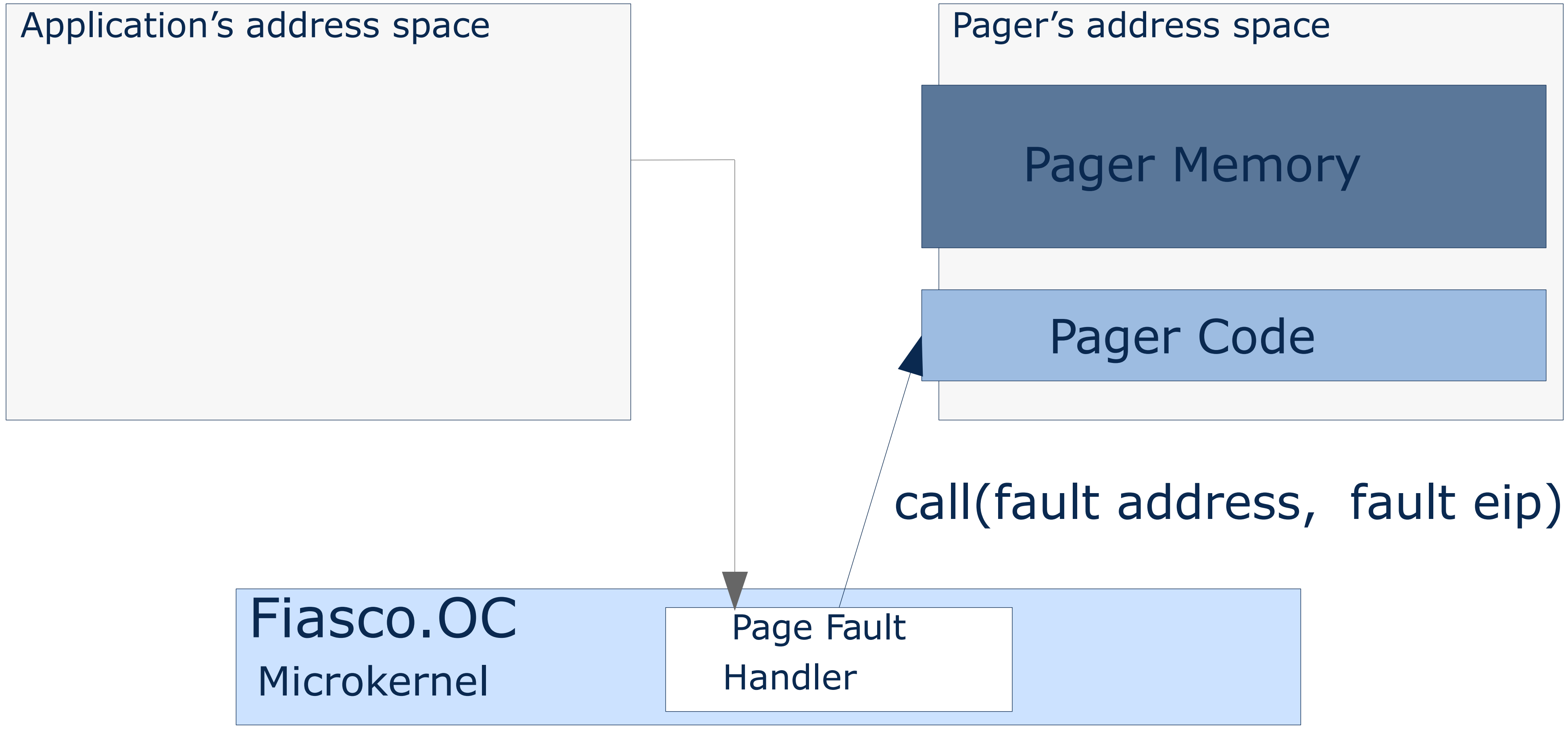
# PAGER

- Thread which is invoked on page fault
- Fiasco.OC: 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



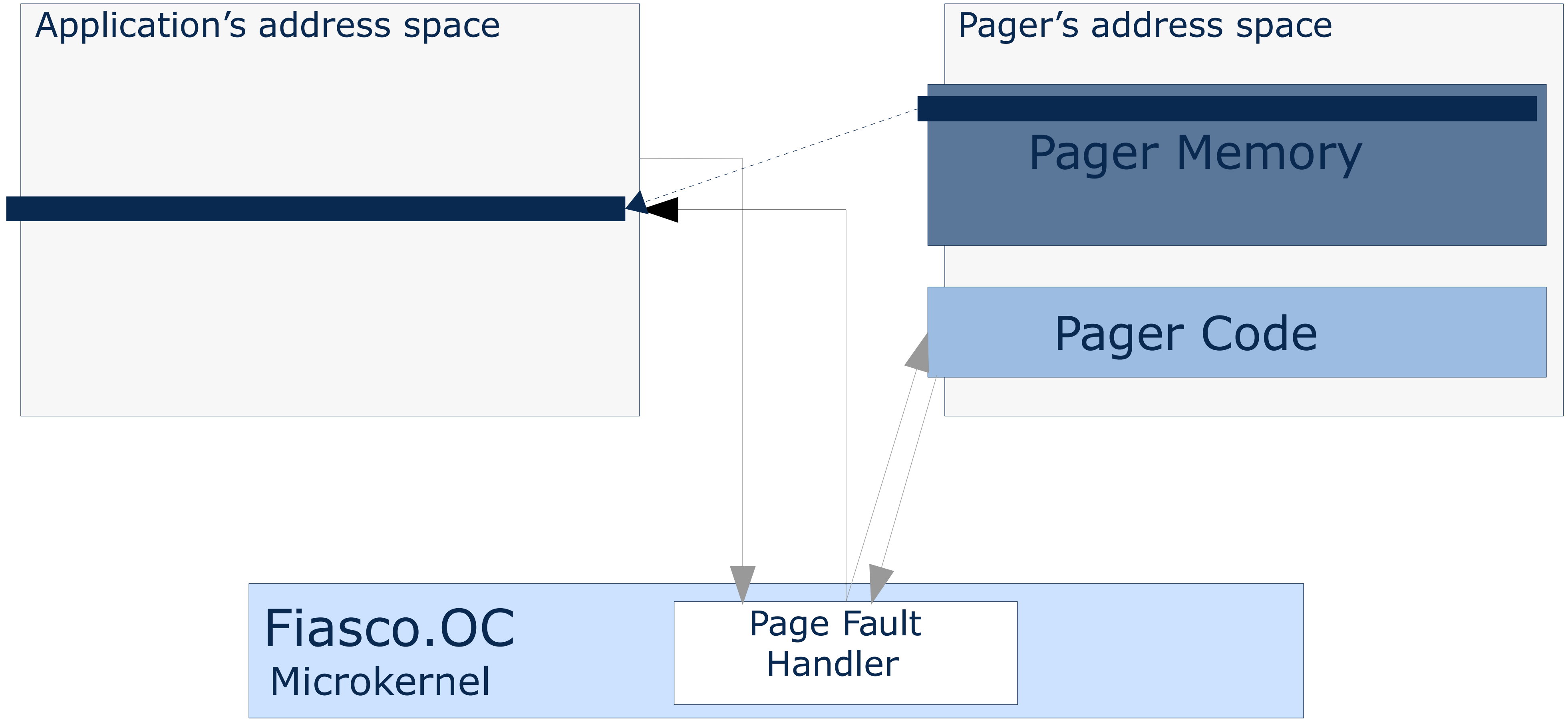
# PAGE-FAULT IPC

| | | | |
|---|---|---|---|
| UTCB[0] | fault address / 4 (30) | *w* | *p* |
| UTCB[1] | faulting EIP (32) | | |

| | |
|---|---|
| *w* = 0 | read page fault |
| *w* = 1 | write page fault |
| *p* = 0 | no page present |
| *p* = 1 | page present |

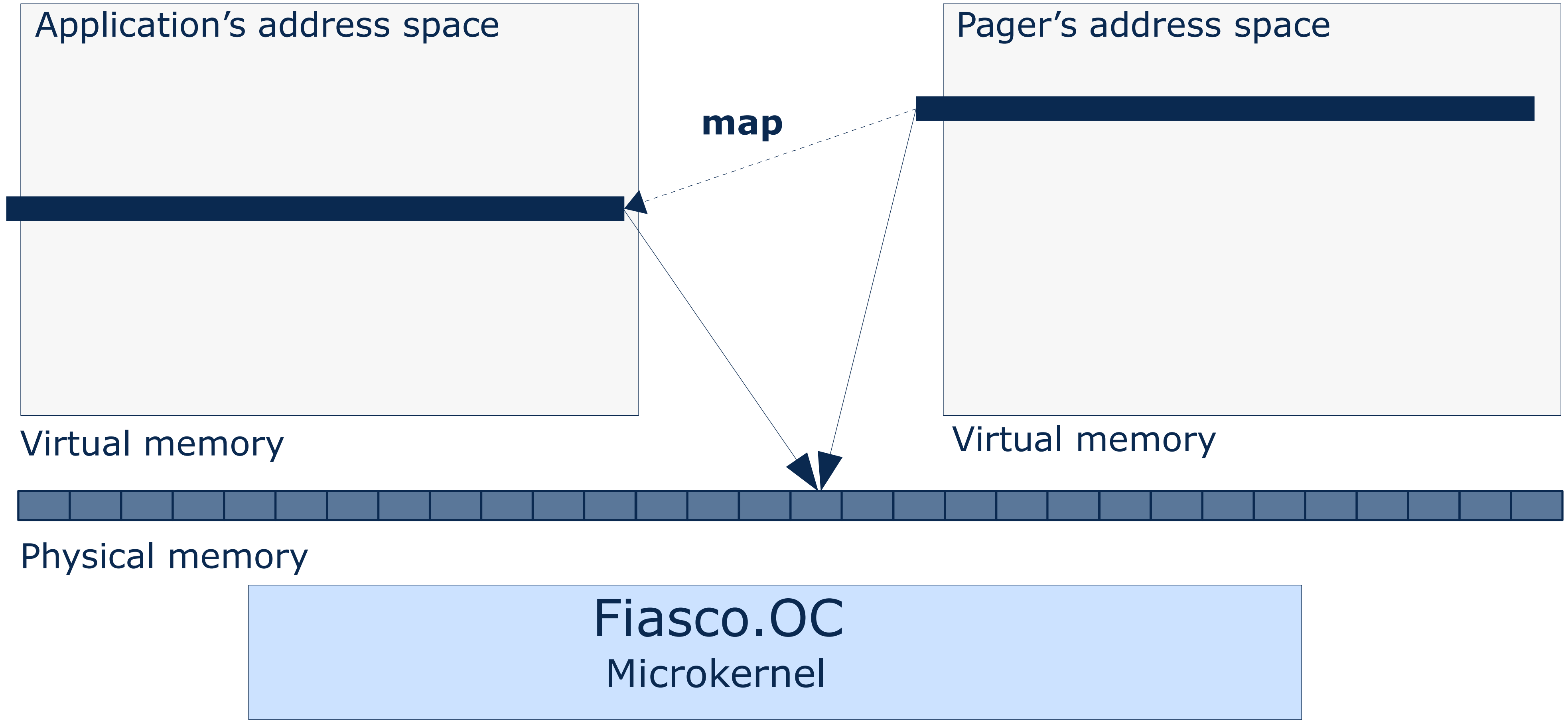
# PAGER REPLY

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



# PAGE MAPPING

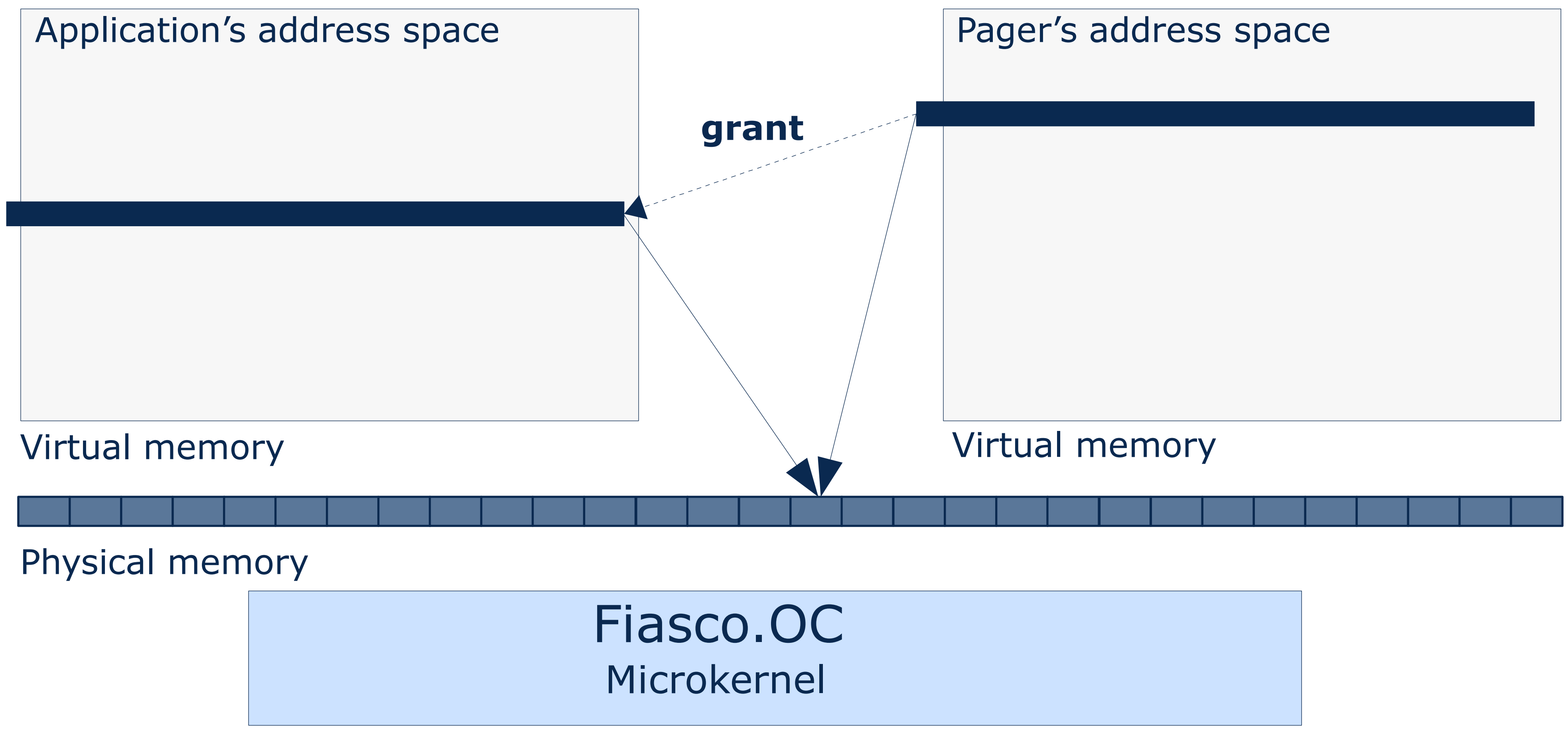
- map() creates an entry in the receiver's address space pointing to the same page frame
  - In hardware: page table entry
- Only valid pager address space entries can be mapped



# PAGE GRANTING

- Special case: grant pages (flag: L4_FPAGE_GRANT)

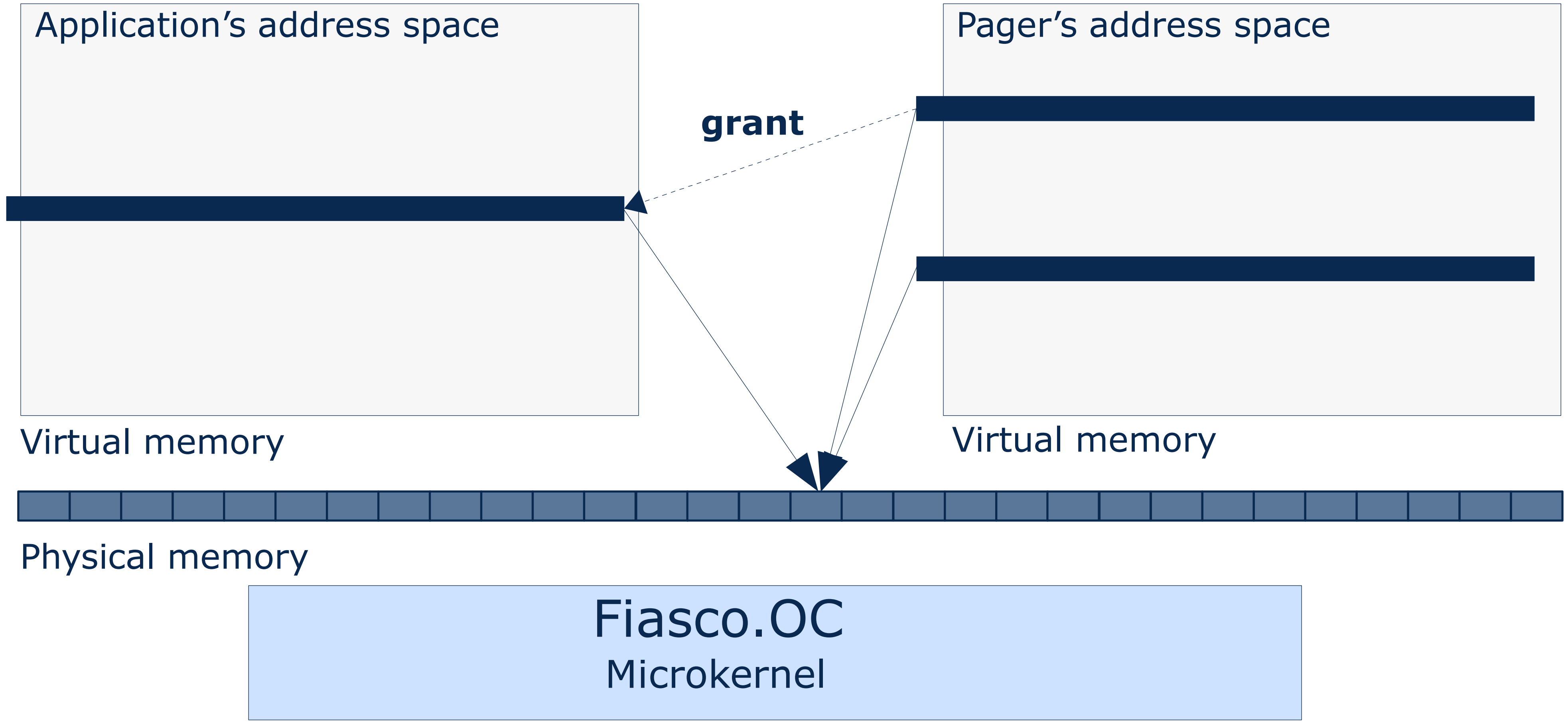
→ Removes mapping from sender's address space



# PAGE GRANTING

- Special case: grant pages (flag: L4_FPAGE_GRANT)

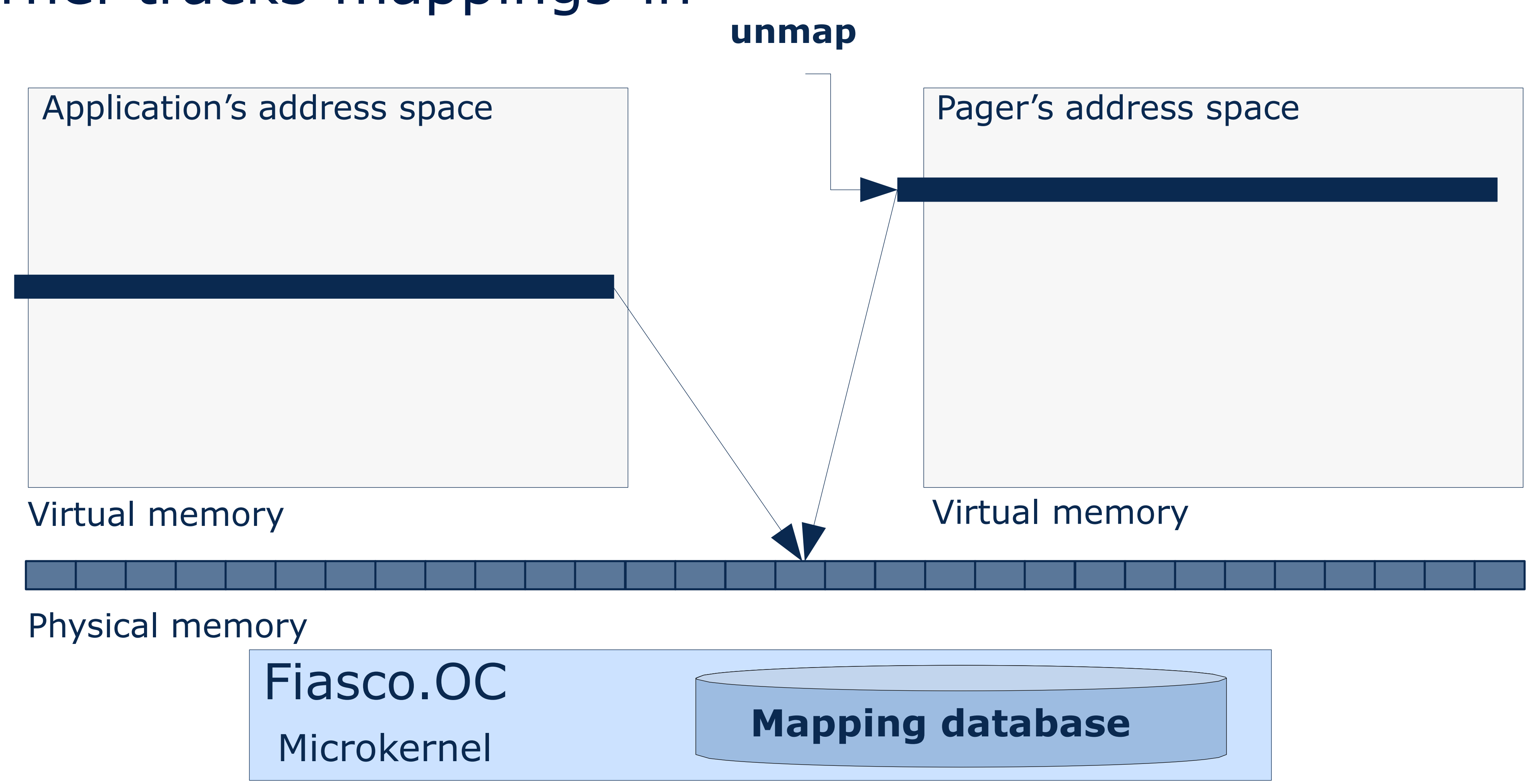
→ Removes mapping from sender's address space
  - → ATTENTION: aliases remain



# PAGE UN-MAPPING

- Removes entries to a page frame (fpage is specified in invoker's address space)
- Dedicated system call: do not need partner's consent

→ Kernel tracks mappings in a database



# FLEXPAGES

# FLEXPAGE TYPES

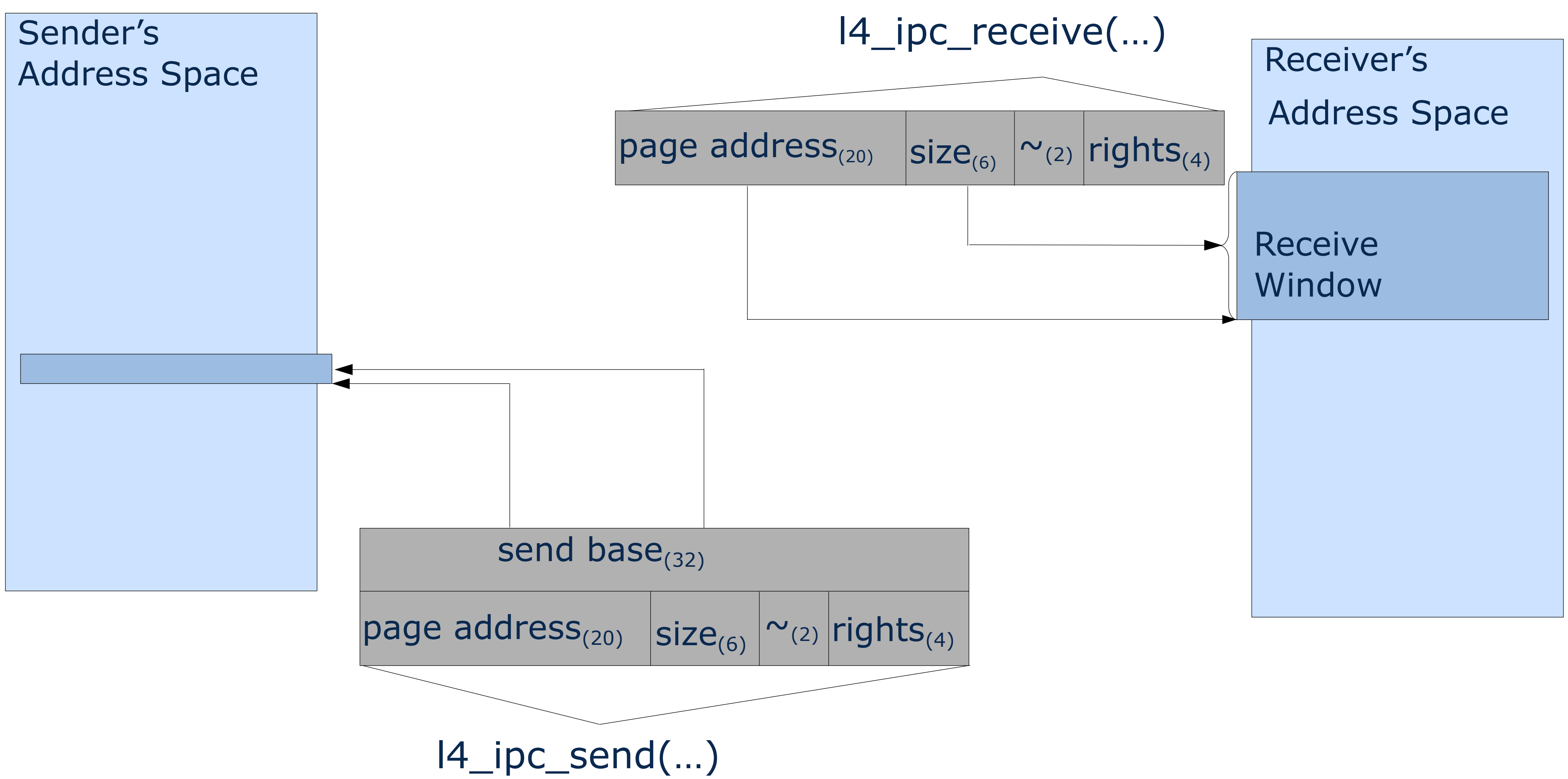
- Flexpages represent resources attached to an address space

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

- Today: only flexpages for memory

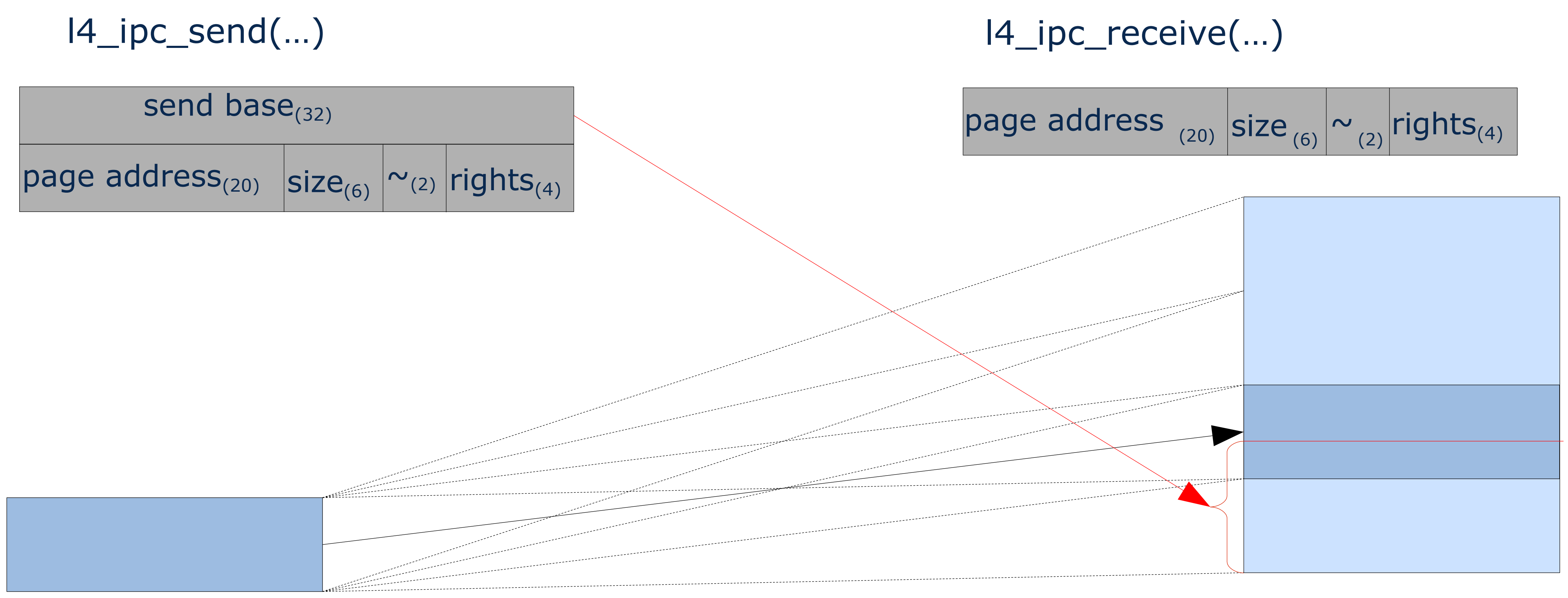
# FLEXPAGE IPC

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



# FLEXPAGE OFFSET

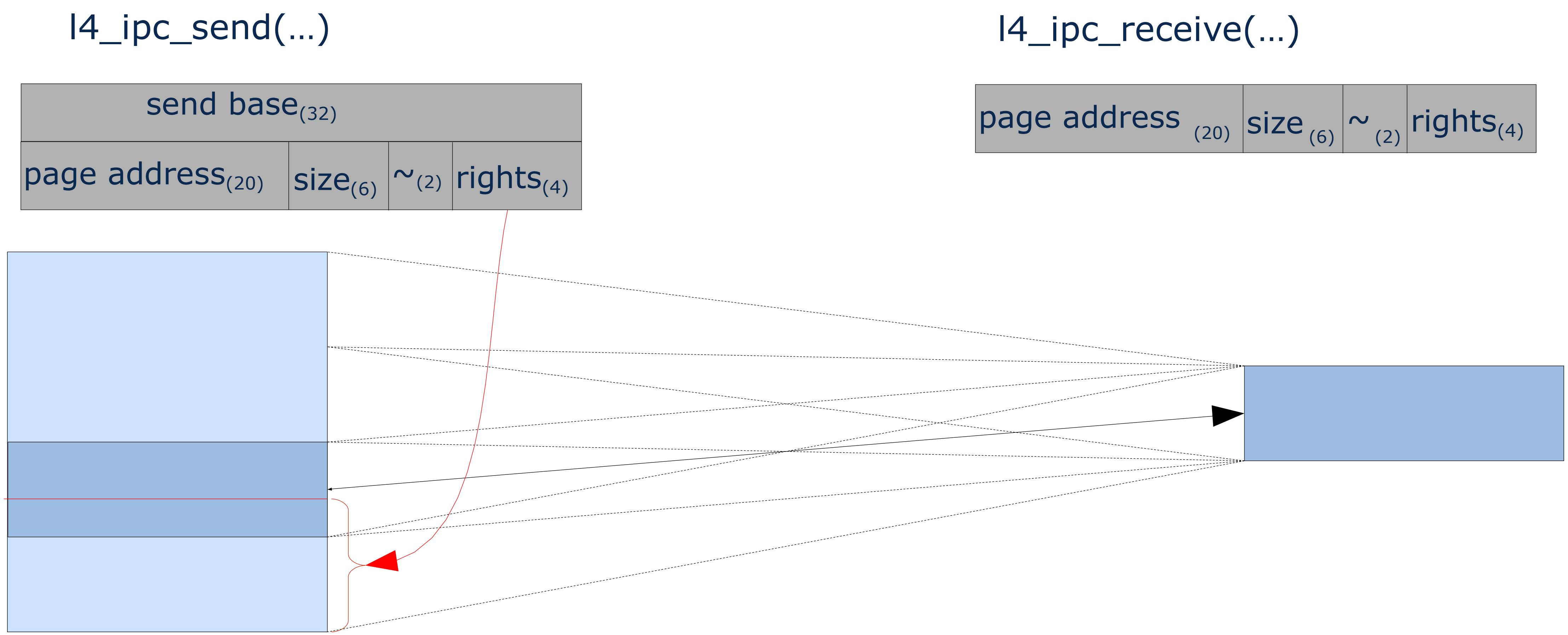
- Send flexpage is smaller than the receive window
  - Target position is derived from send flexpage alignment and send base



# FLEXPAGE OFFSET

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

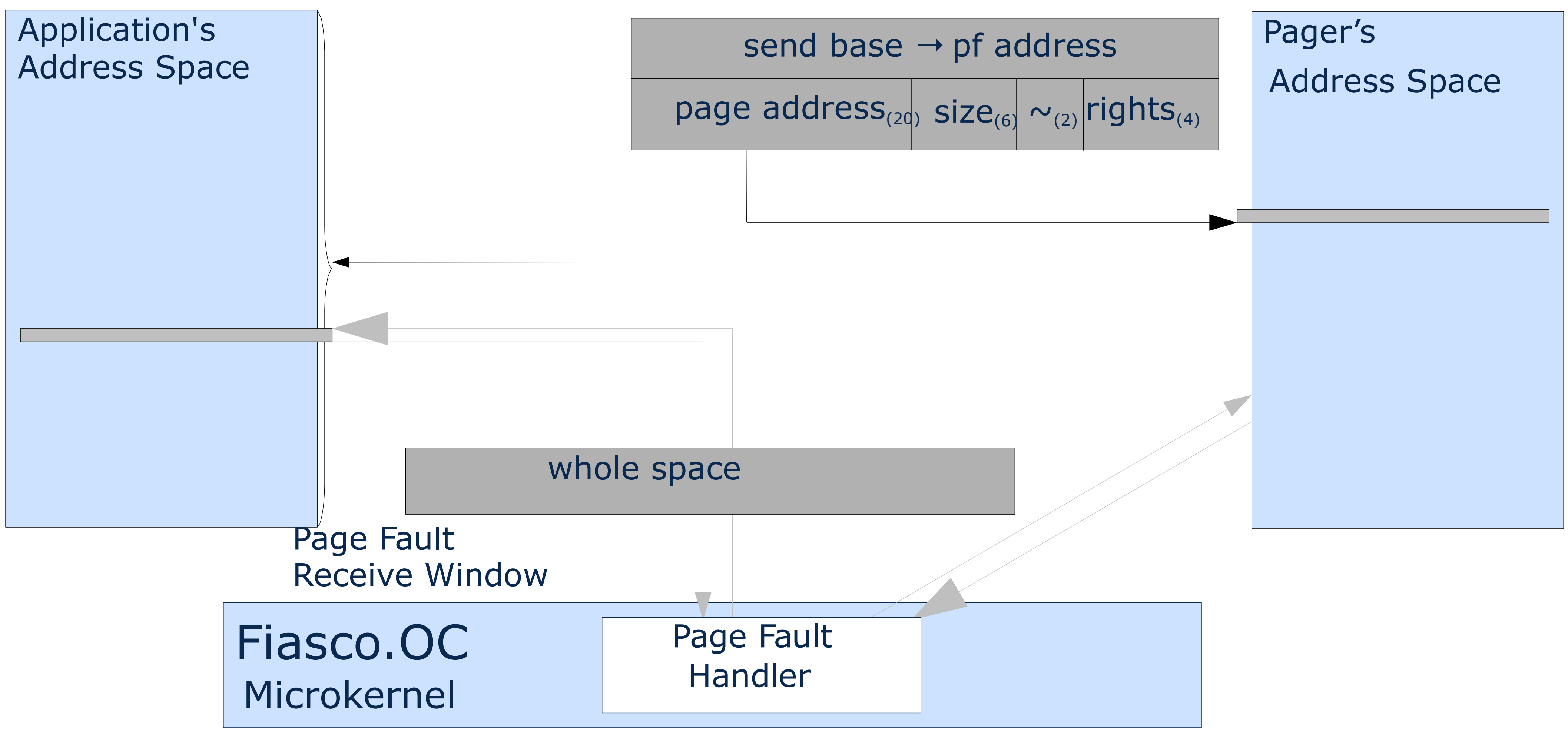
→ Send base depends on information about the receiver



# PAGE FAULT

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

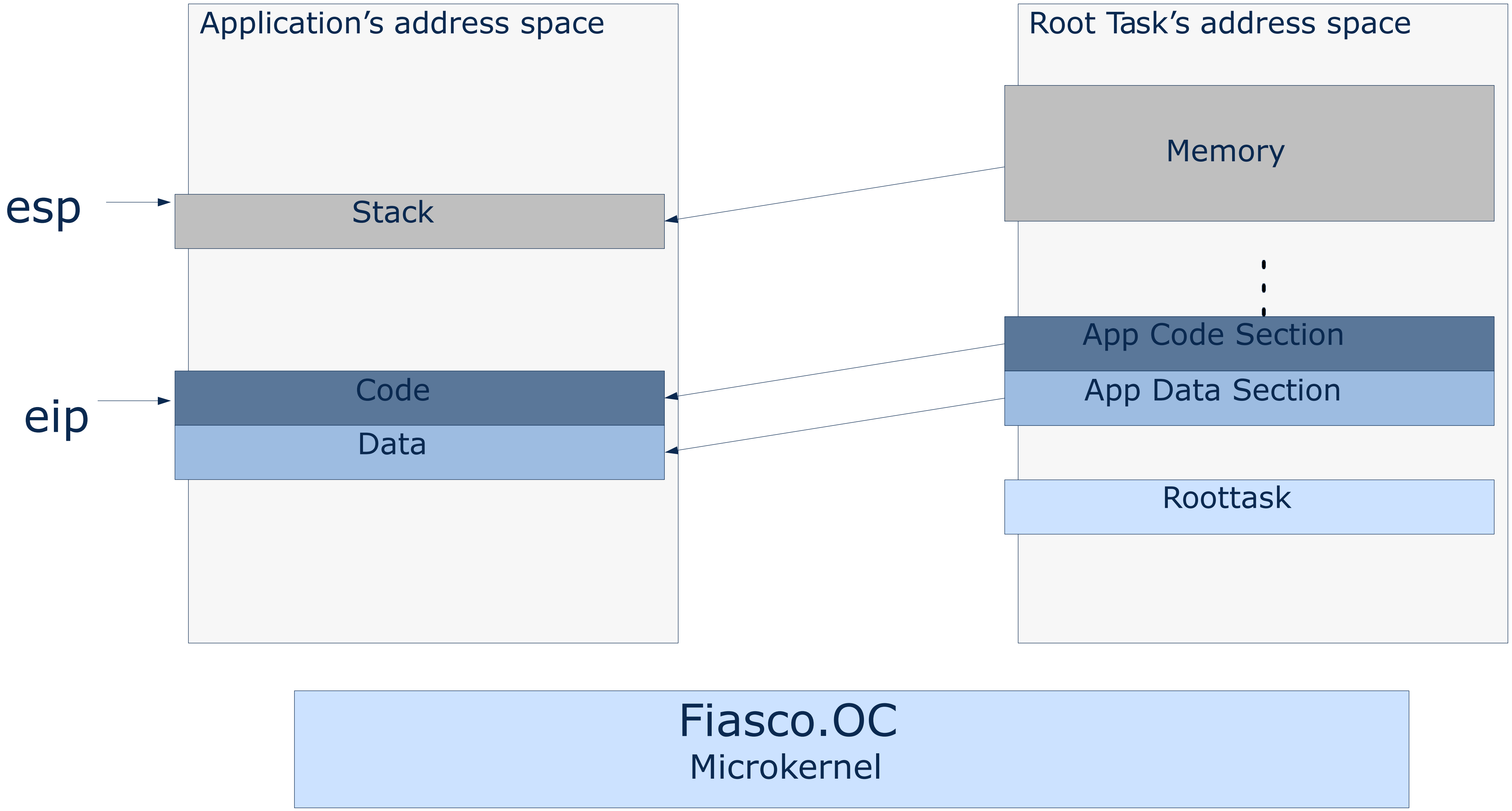
→ Pager can map more than just one page, where the page fault happened to the client



# ROOT TASK

- Pages are mapped as they are needed

→ *demand paging*



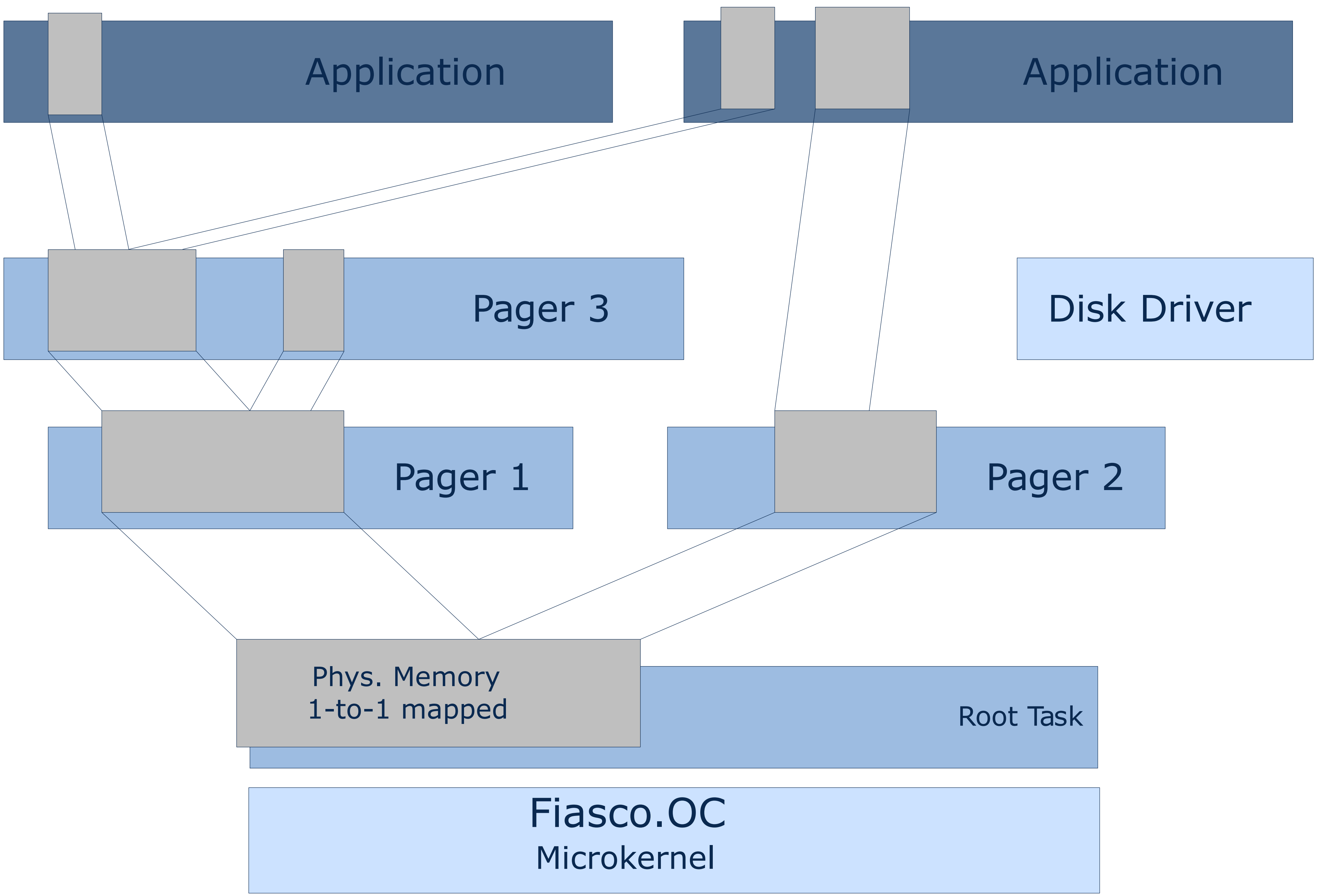
# MANAGEMENT

- Initial pager can only implement basic memory management
- No knowledge 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

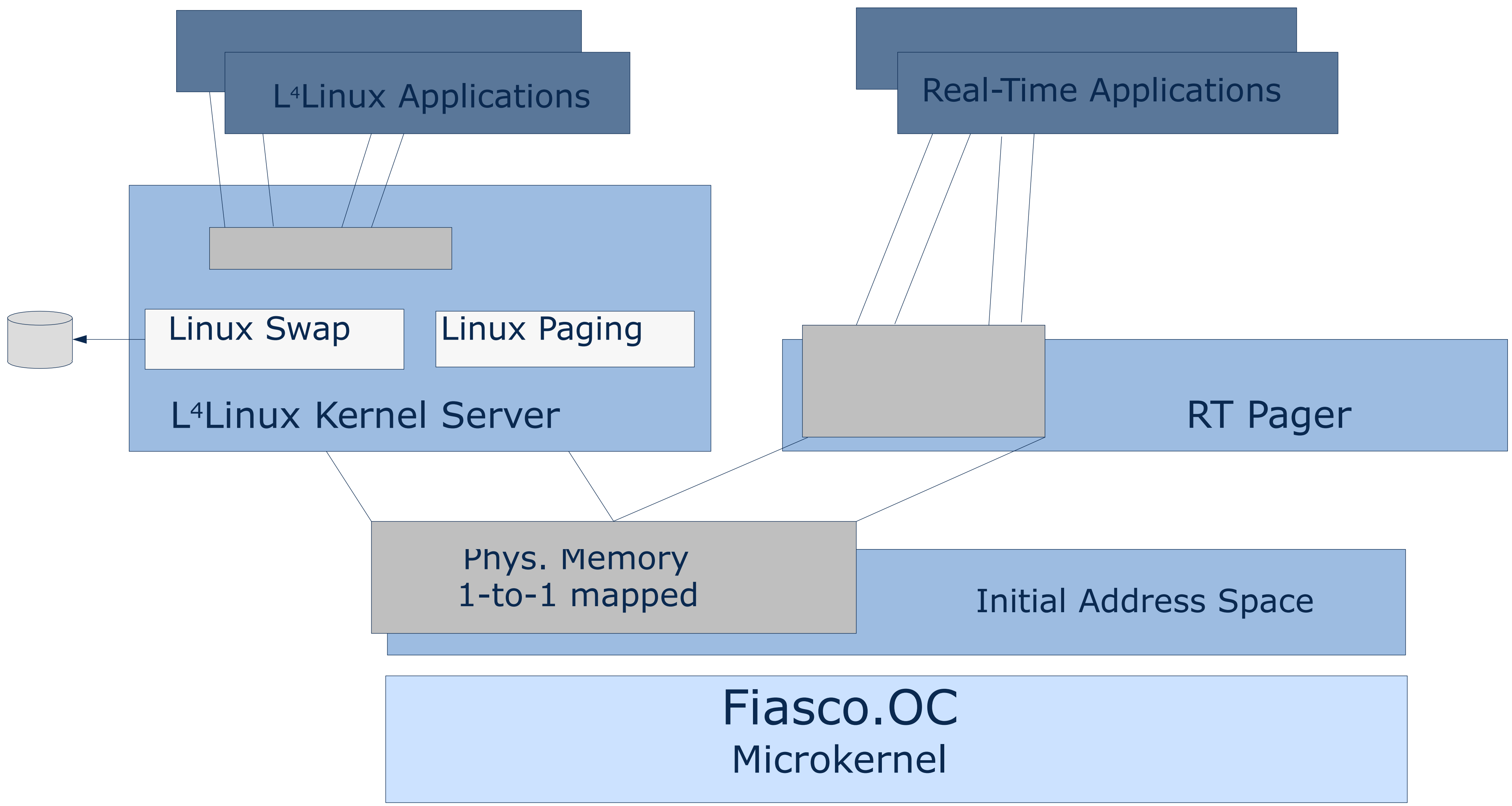
# HIERARCHICAL PAGERS

# PAGER HIERARCHY



# EXAMPLE

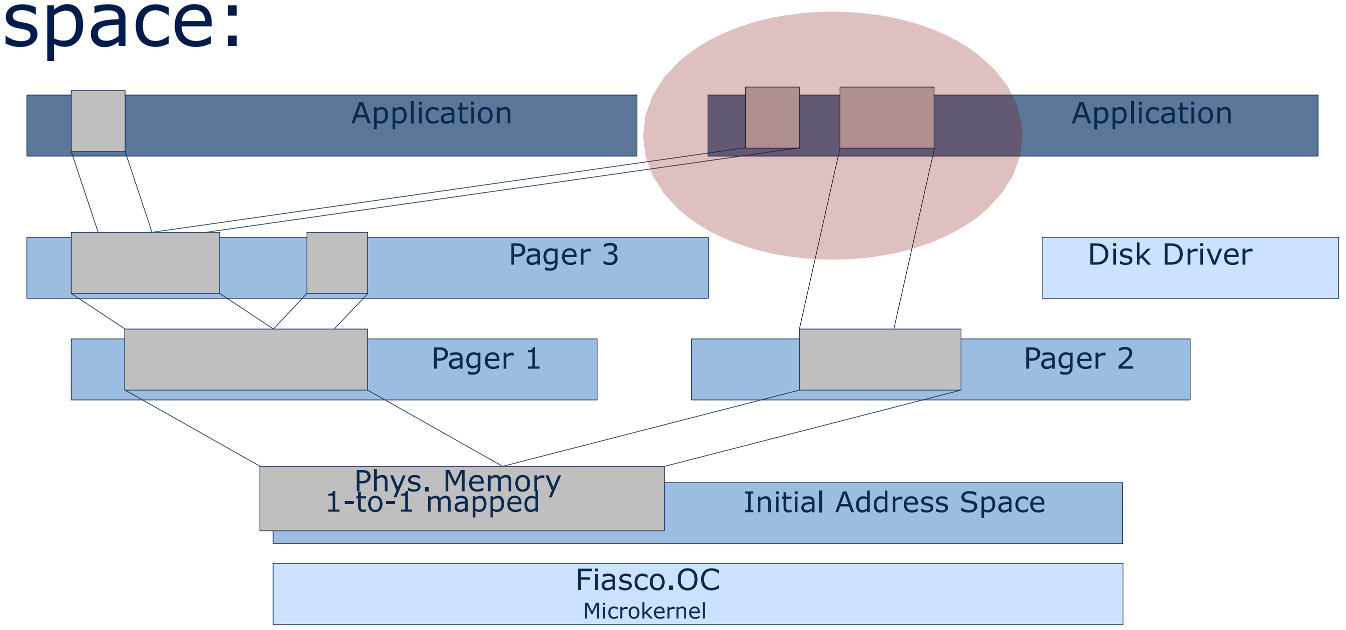
- $L^4$Linux implements Linux paging policy
- RT pager implements real-time paging policy (e.g. no swapping)



# REGION MANAGER

# VIRTUAL MEMORY

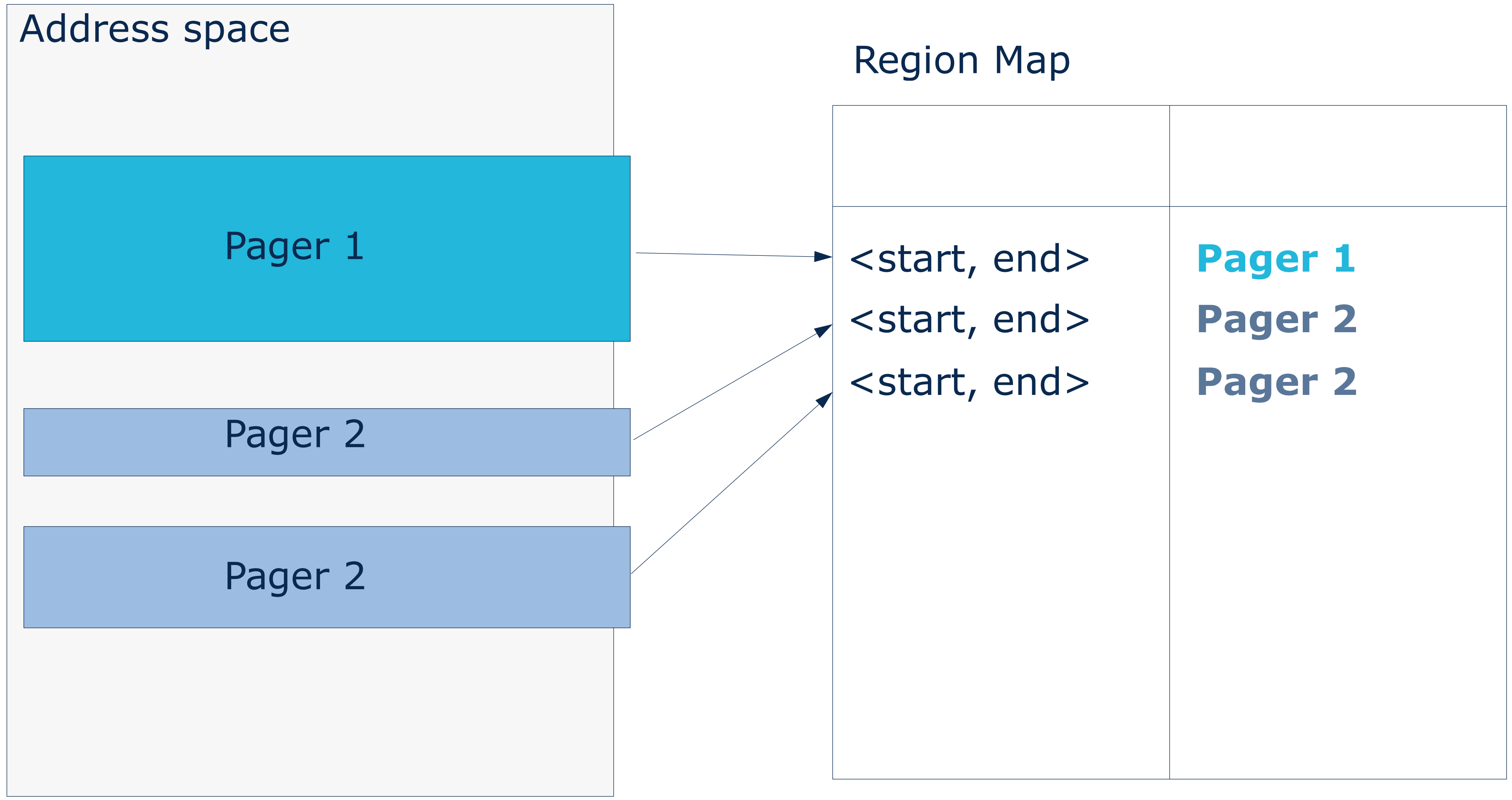
- Pager has to specify send base
- Pager needs to know client's address space layout
  - No problems with only one pager (e.g. L4Linux)
- Possible conflicts if more than one pager manages an address space:




→ Virtual memory must be managed independent of pagers

# REGION MAP

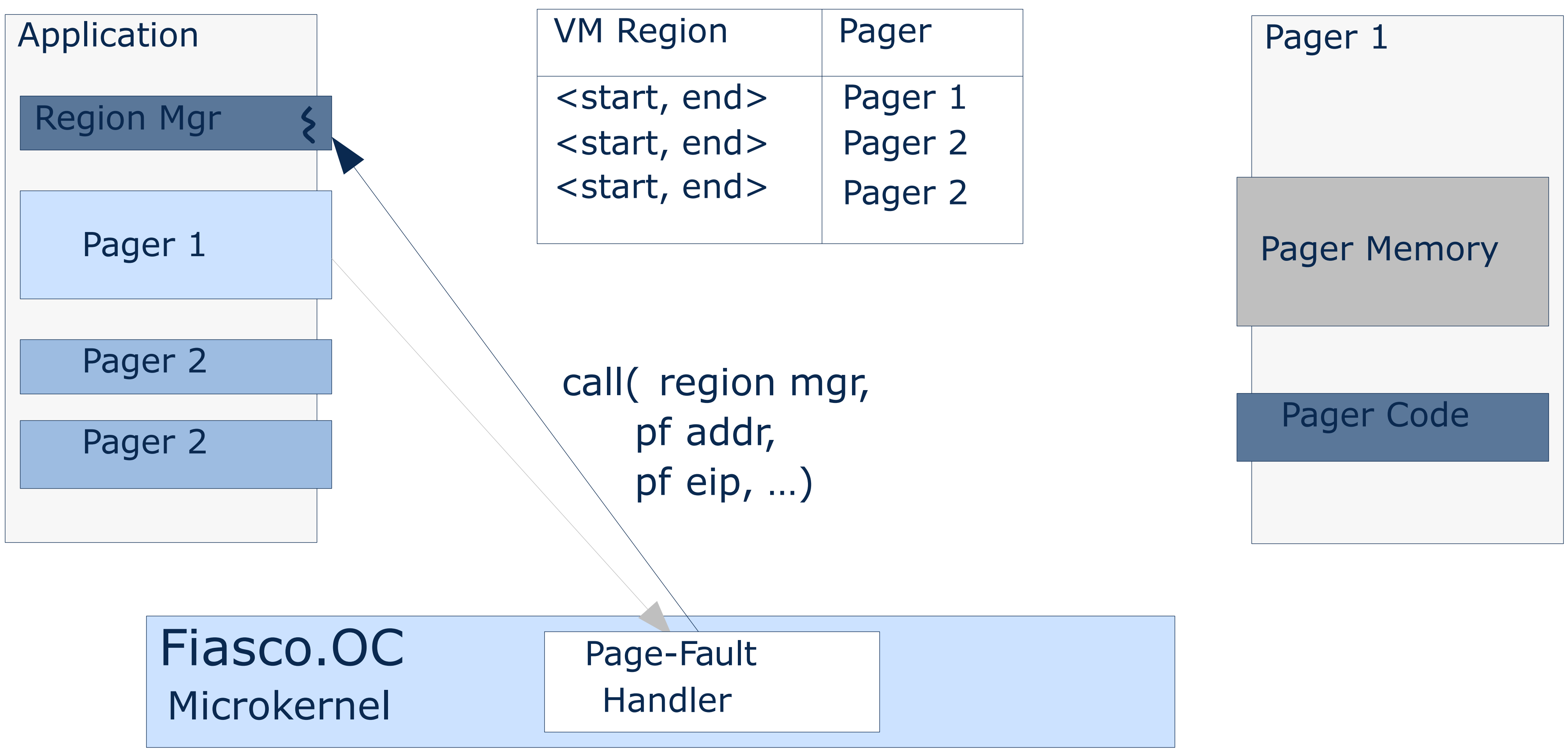
- Per address space map that keeps track which part of the address space is managed by which pager



# REGION MANAGER

- Intermediate pager that identifies which pager should handle a page fault
- Resides in the application's address space

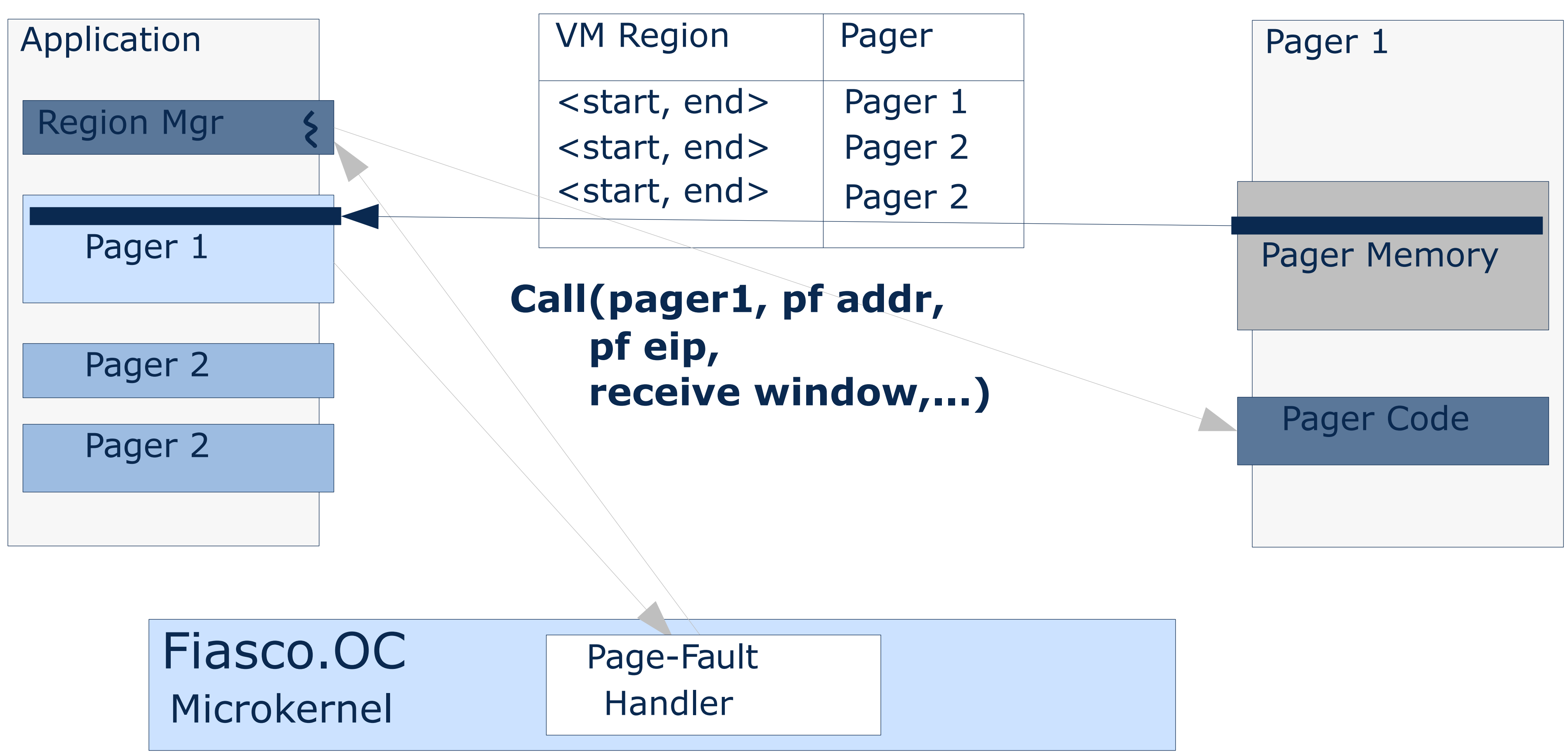
→ Region manager is the pager of all threads of a task



# REGION MANAGER

- Region manager calls the pager that is responsible
- Receive window gets restricted to the area managed by that pager

→ No interference between different pagers



# 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

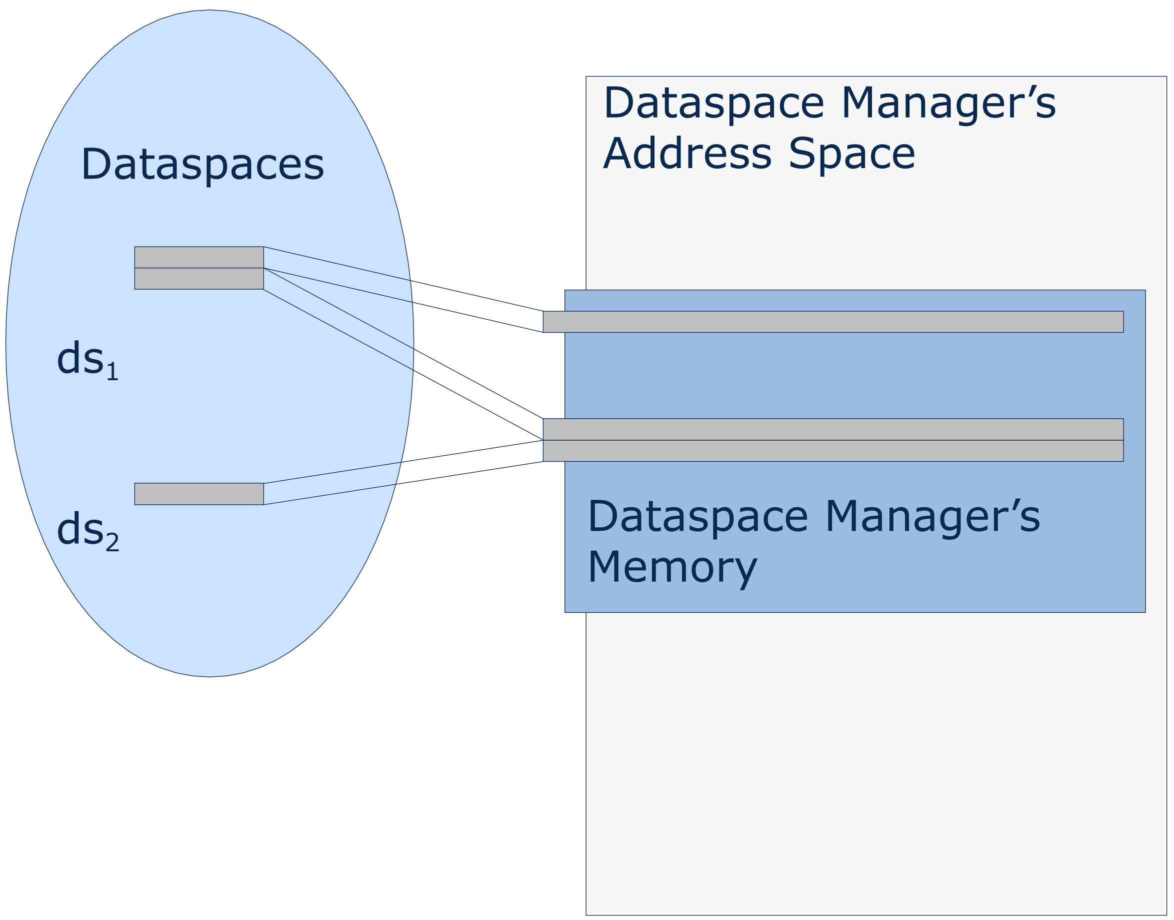
# DATASPACES

- 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

# DS MANAGER

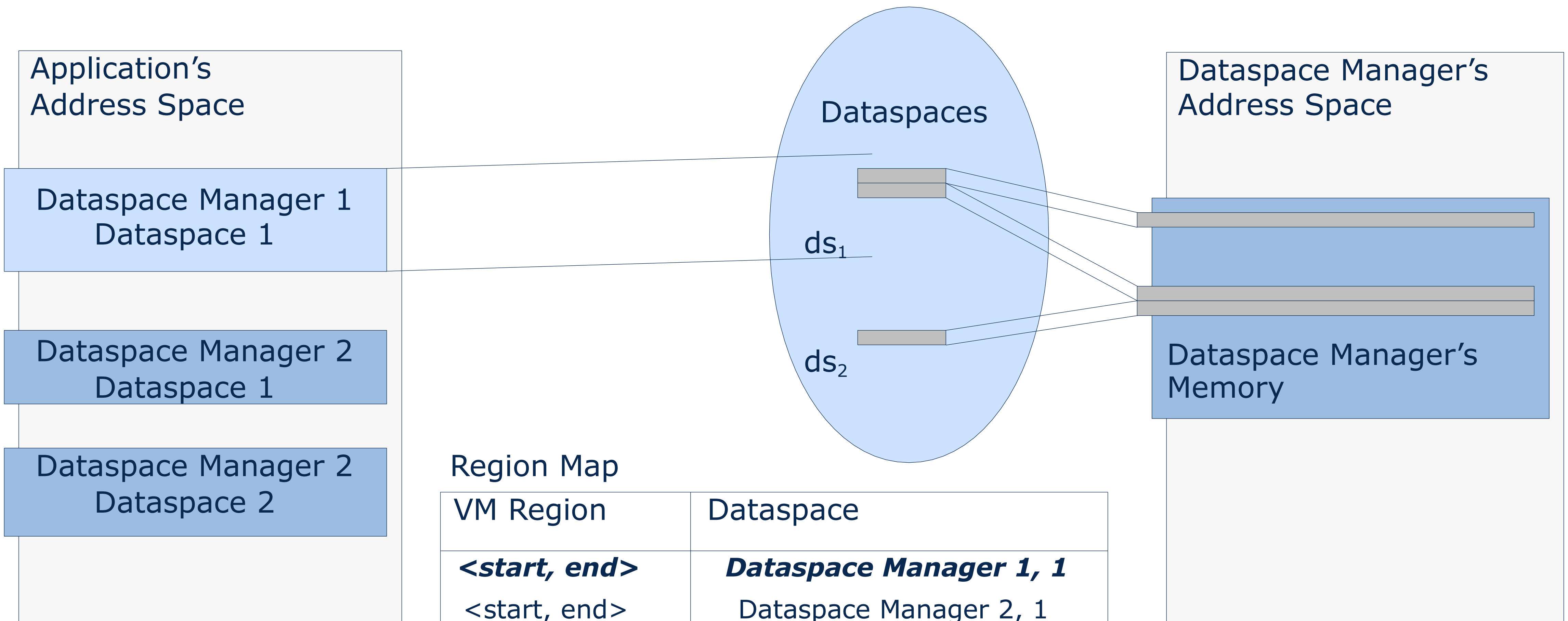
- DS Manager determines the semantic of a dataspace
- Each DSM is the pager for its dataspaces

→ Implements the paging policy (page replacement etc.)



# DS & REGIONS

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



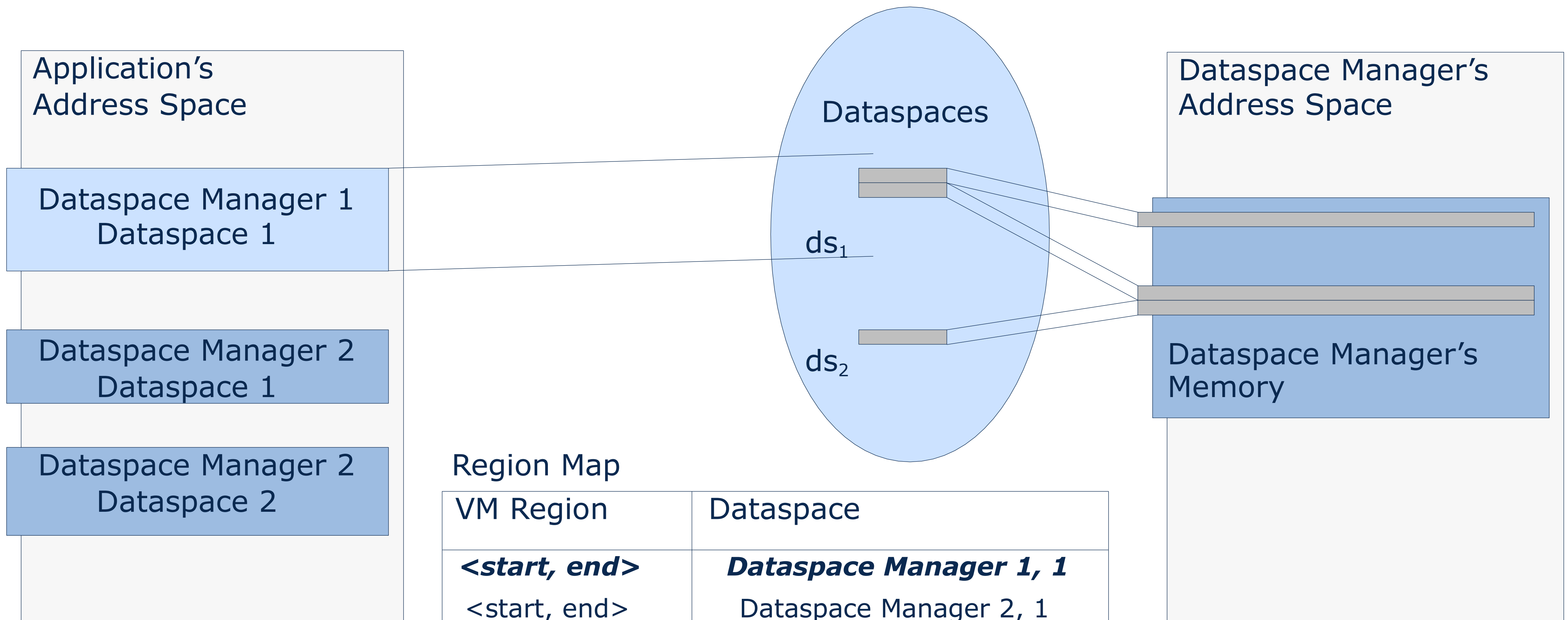

Region Map

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

# DS & REGIONS

- Region manager propagates fault to dataspace manager's fault handler

➔ Dataspace fault (ds_manager_id, ds_id, offset)




Region Map

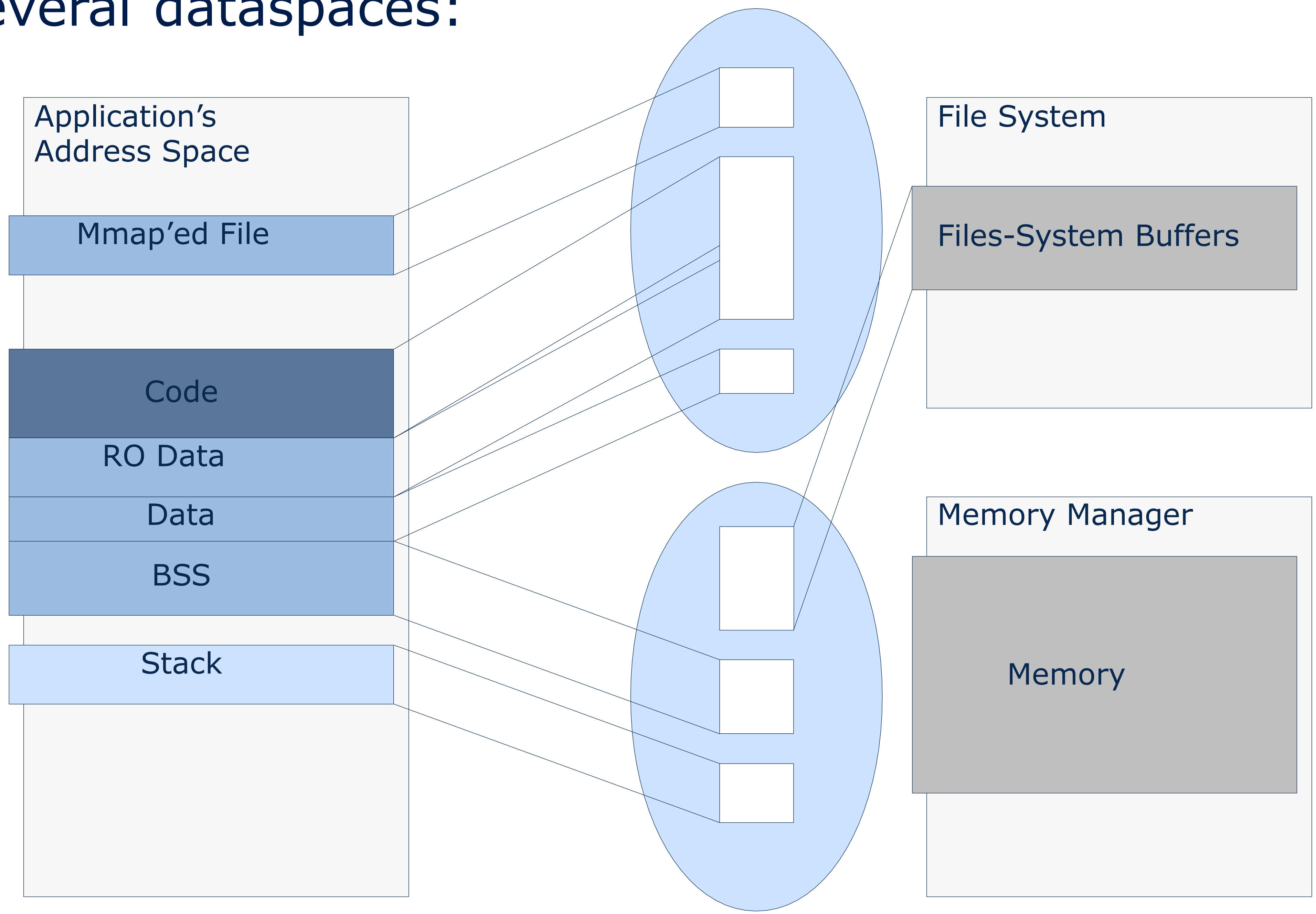
| VM Region | Dataspace |
|---|---|
| ***<start, end>*** | ***Dataspace Manager 1, 1*** |
| <start, end> | Dataspace Manager 2, 1 |
| <start, end> | Dataspace Manager 2, 2 |

# APPLICATION VIEW

- 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

# ADDRESS SPACE

- Application address spaces are constructed from several dataspaces:



# FURTHER ISSUES

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

→ Page allocation and 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 manager & dataspaces

# MORE INFO

- 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