



**TECHNISCHE  
UNIVERSITÄT  
DRESDEN**

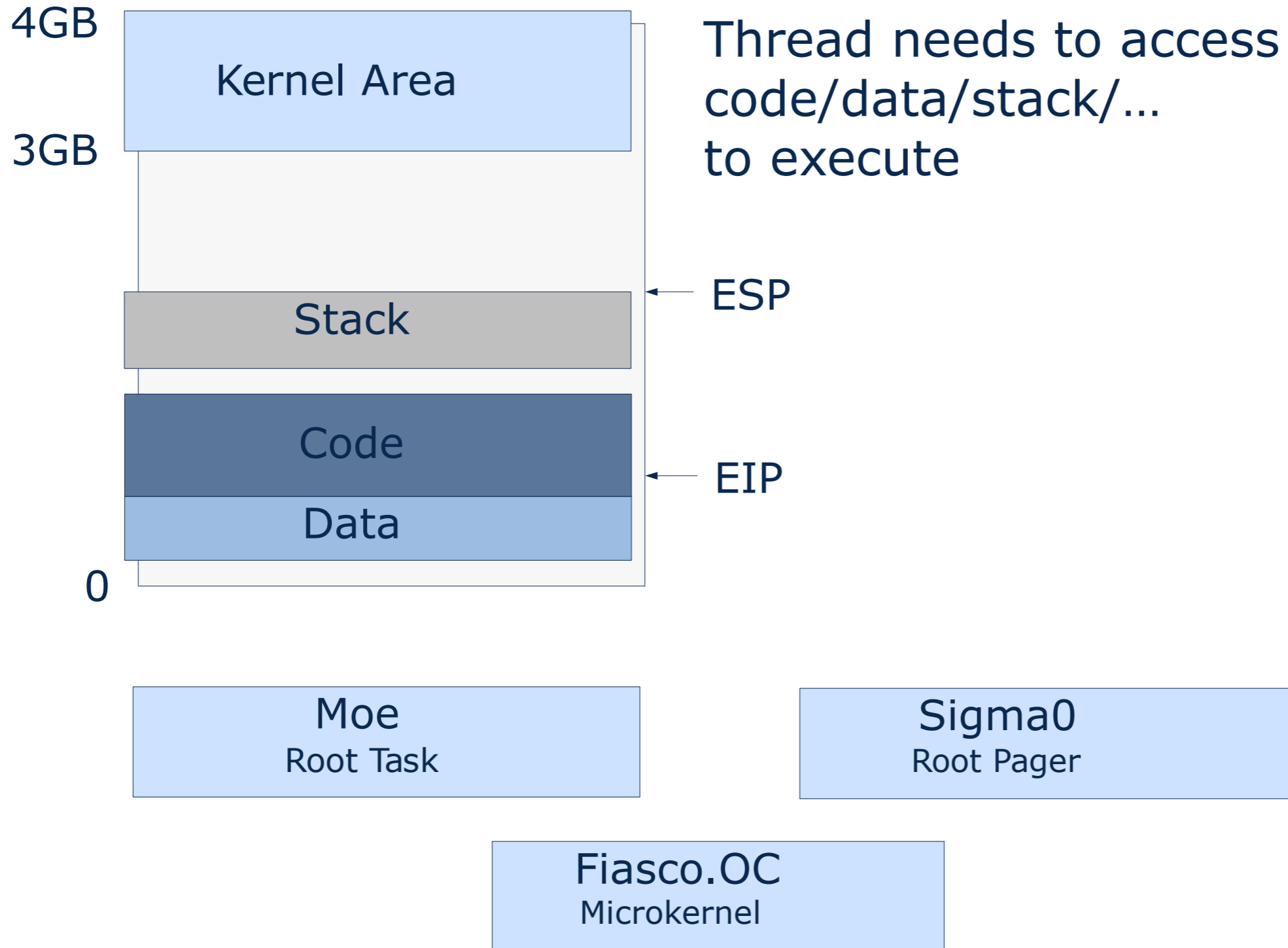
**Faculty of Computer Science** Institute of Systems Architecture, Operating Systems Group

# MEMORY

**MICHAEL ROITZSCH**

- 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



```
/* 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()  
)
```

```
/* Commit the given thread-attributes object. */
```

```
l4_msgtag_t
```

```
L4::Thread::control (Attr const & attr)
```

```
/* Exchange basic thread registers. */
```

```
l4_msgtag_t
```

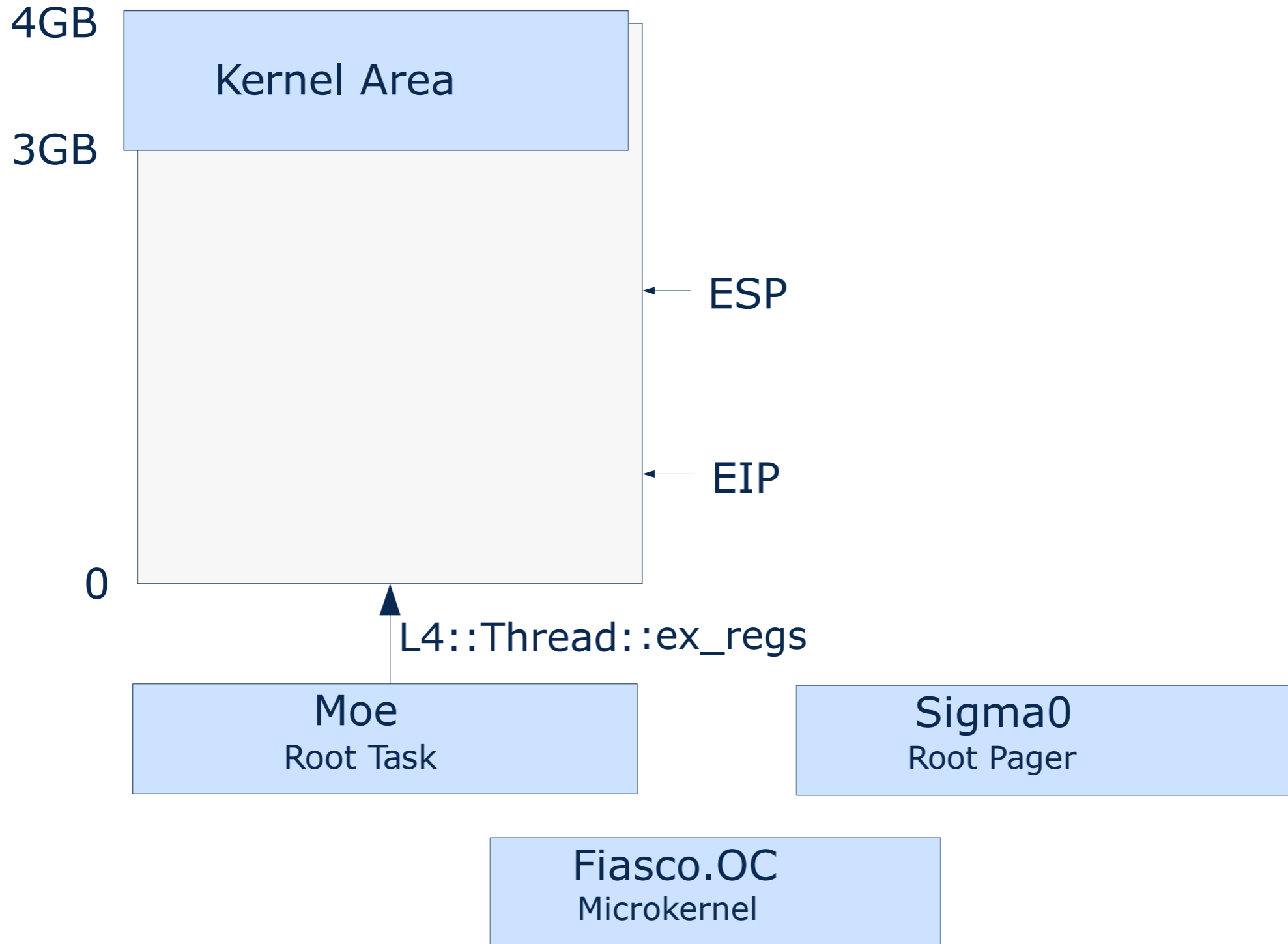
```
L4::Thread::ex_regs (l4_addr_t ip, /* instruction pointer */
```

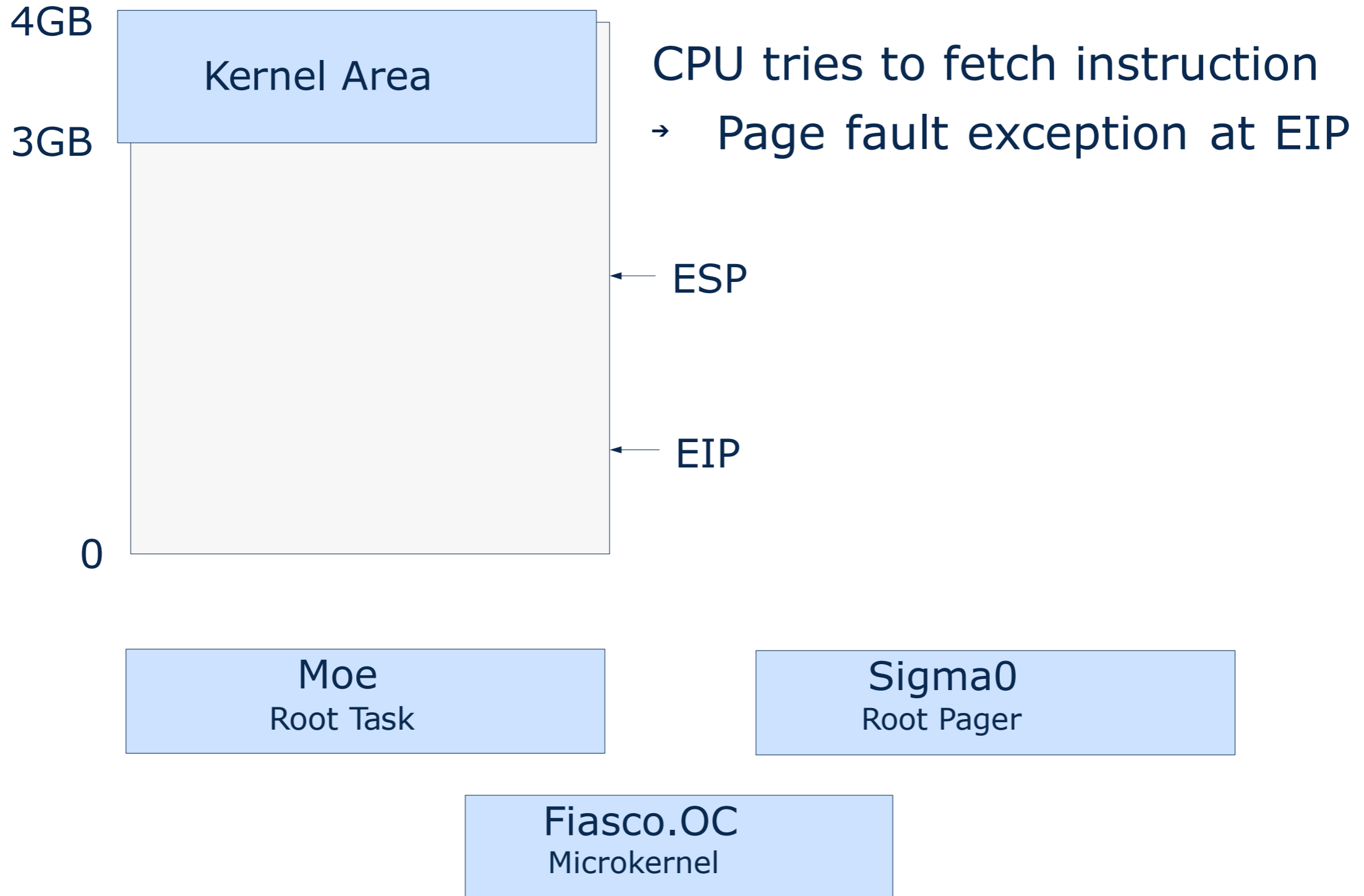
```
l4_addr_t sp, /* stack pointer */
```

```
l4_umword_t flags,
```

```
l4_utcb_t *utcb = l4_utcb()
```

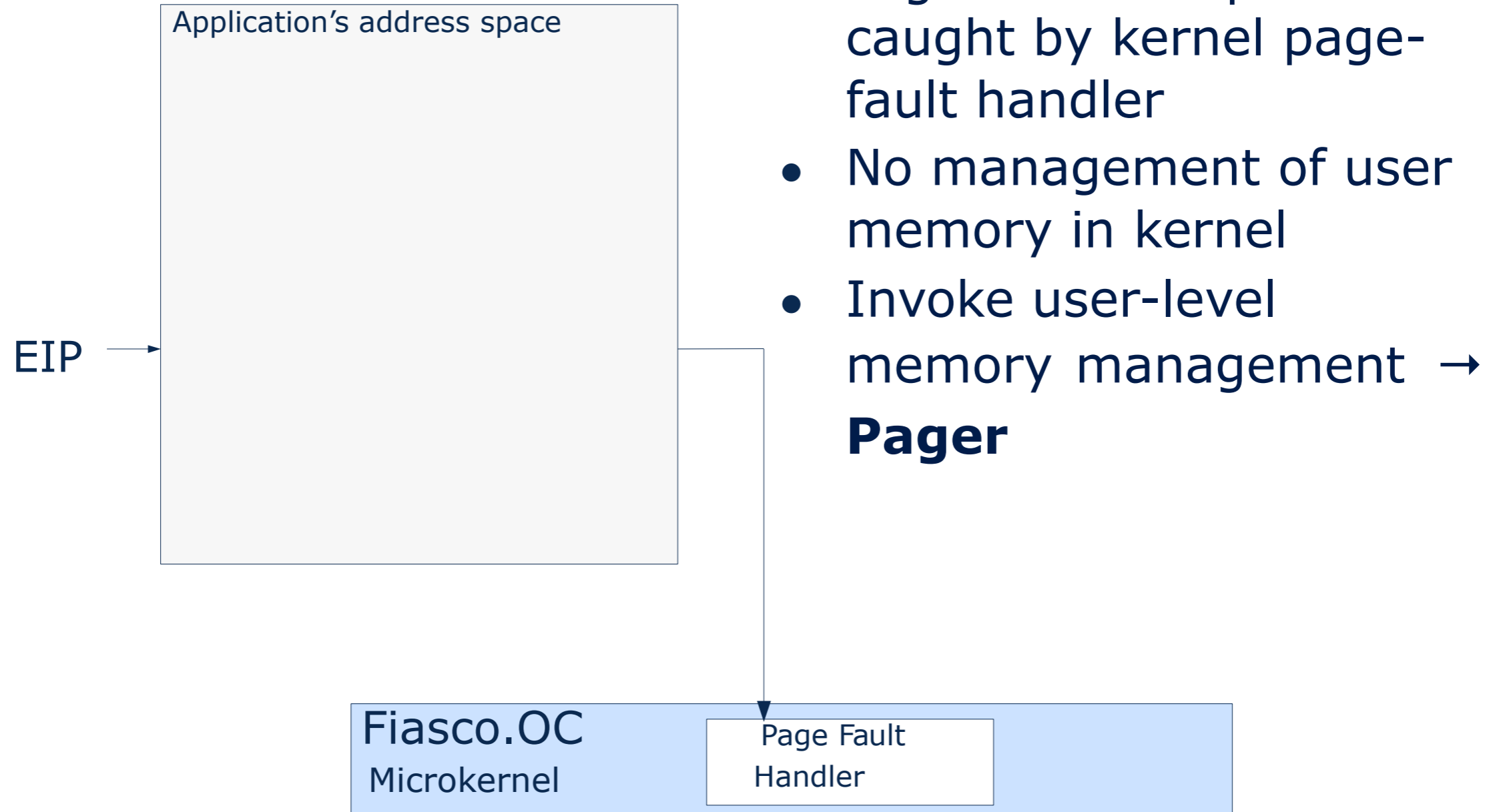
```
)
```



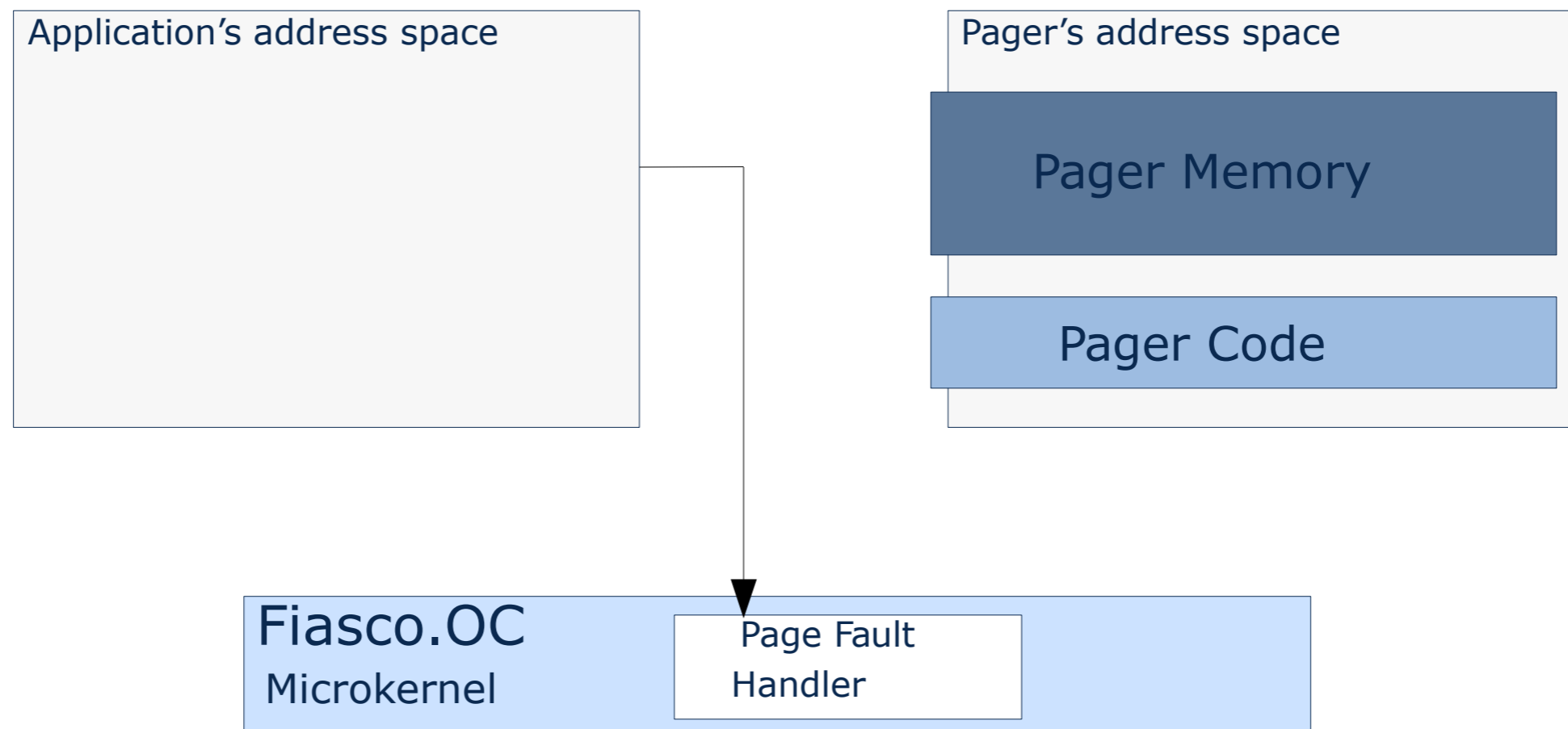




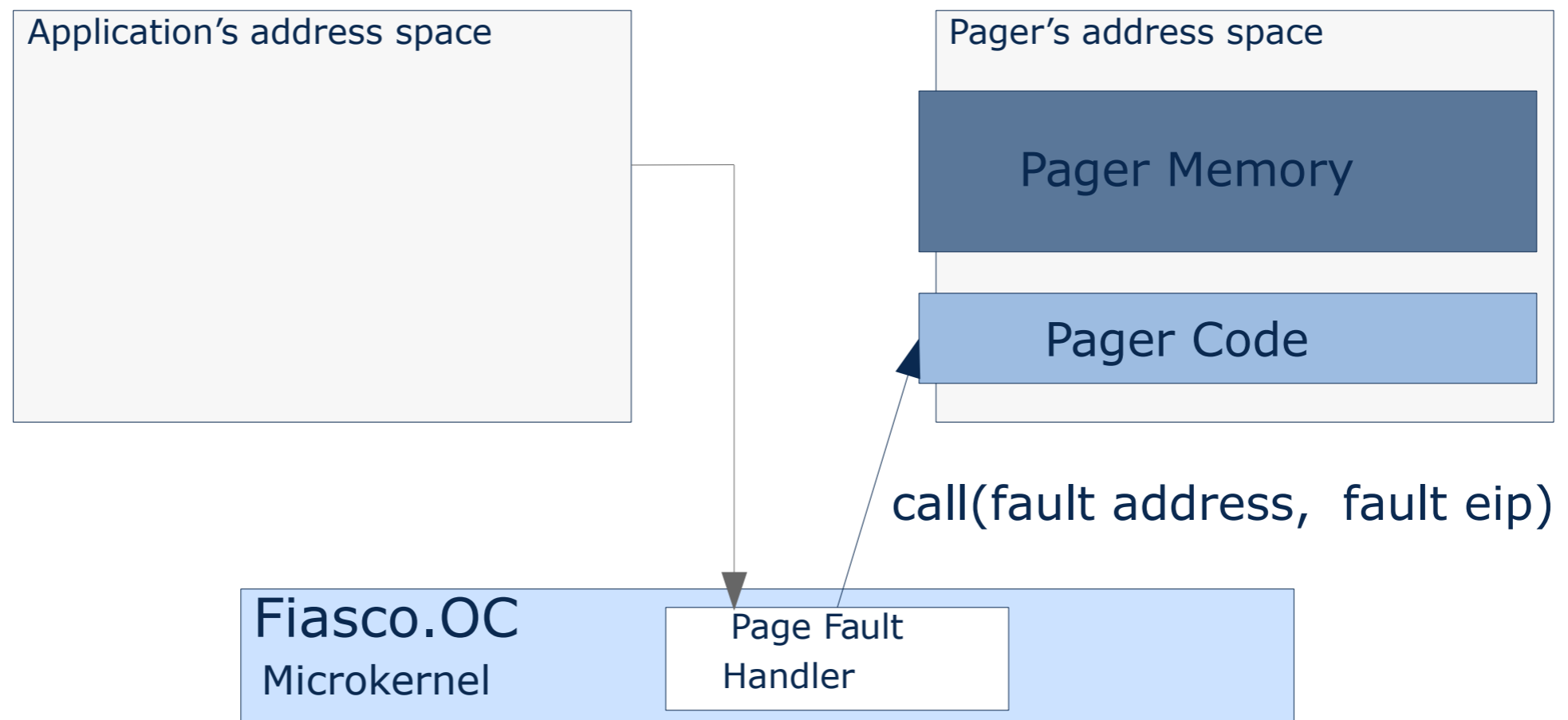
# PAGE FAULT HANDLING

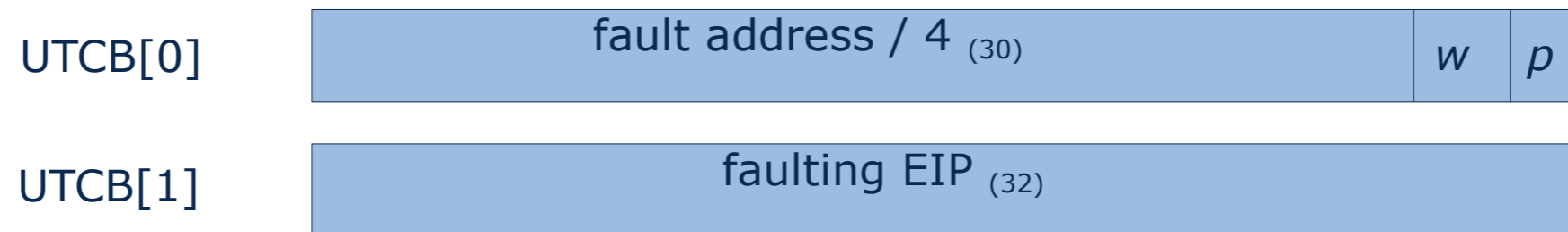


- Thread which is invoked on page fault
- Fiasco.OC: each thread has a (potentially different) pager assigned



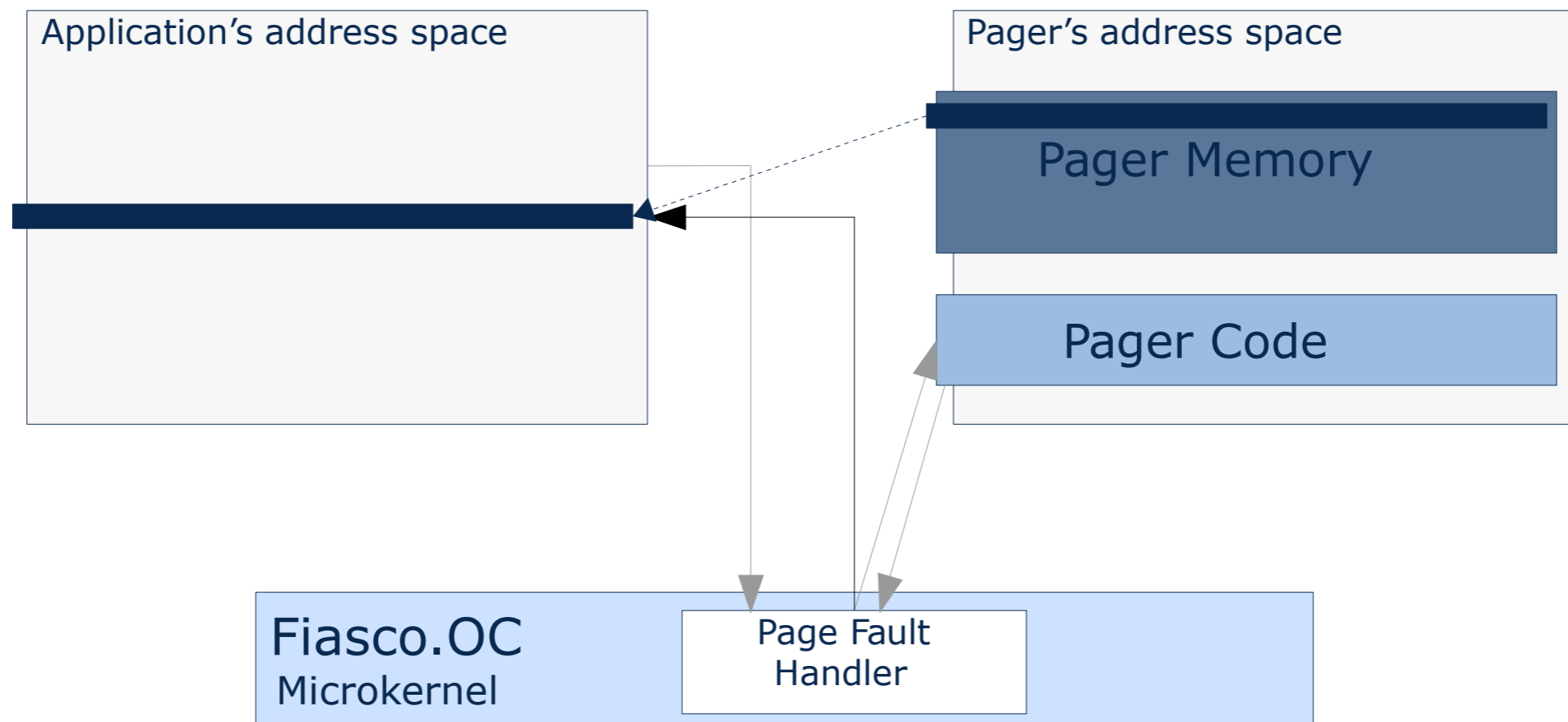
- Communication with pager thread using IPC
- Kernel page fault handler sets up IPC to pager
- Pager sees faulting thread as sender of IPC



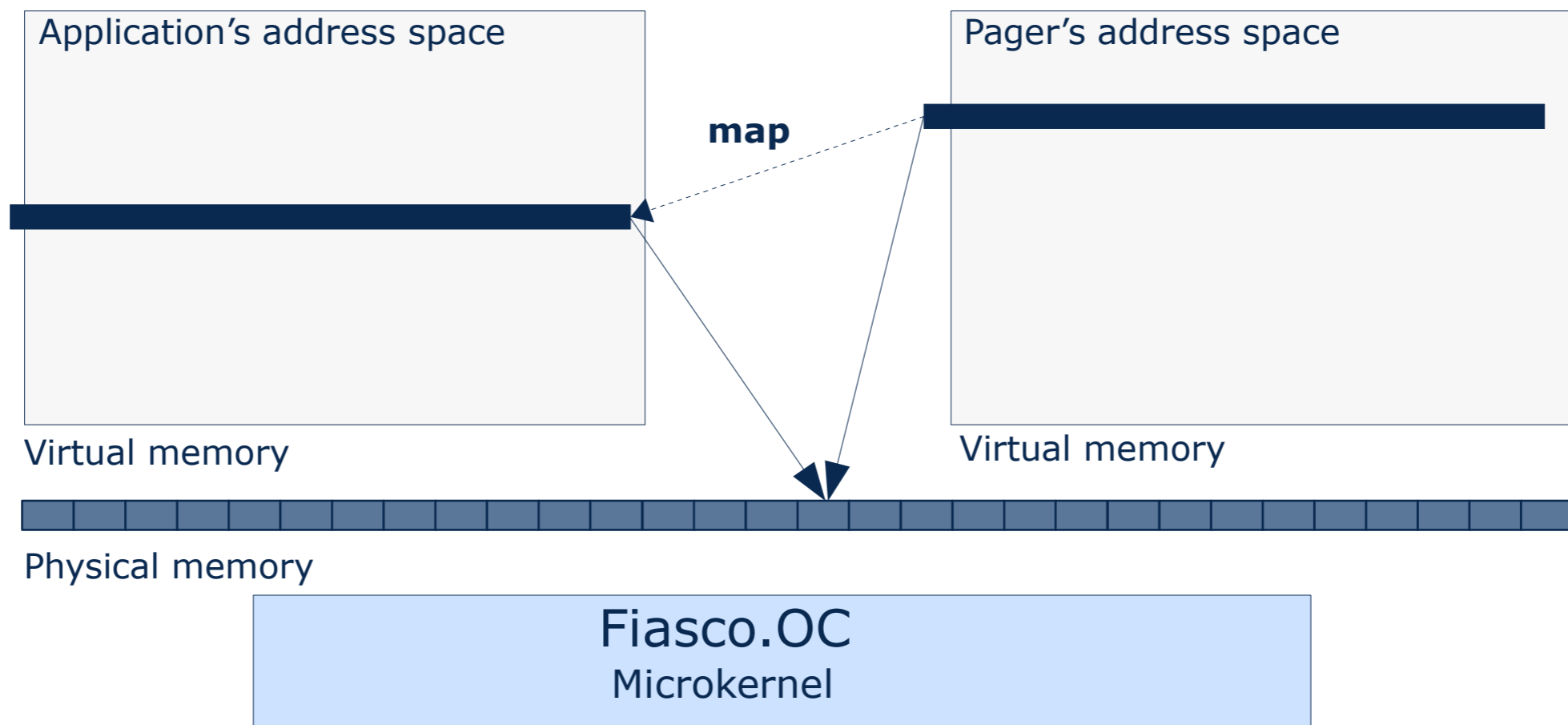


*w* = 0    read page fault  
*w* = 1    write page fault  
*p* = 0    no page present  
*p* = 1    page present

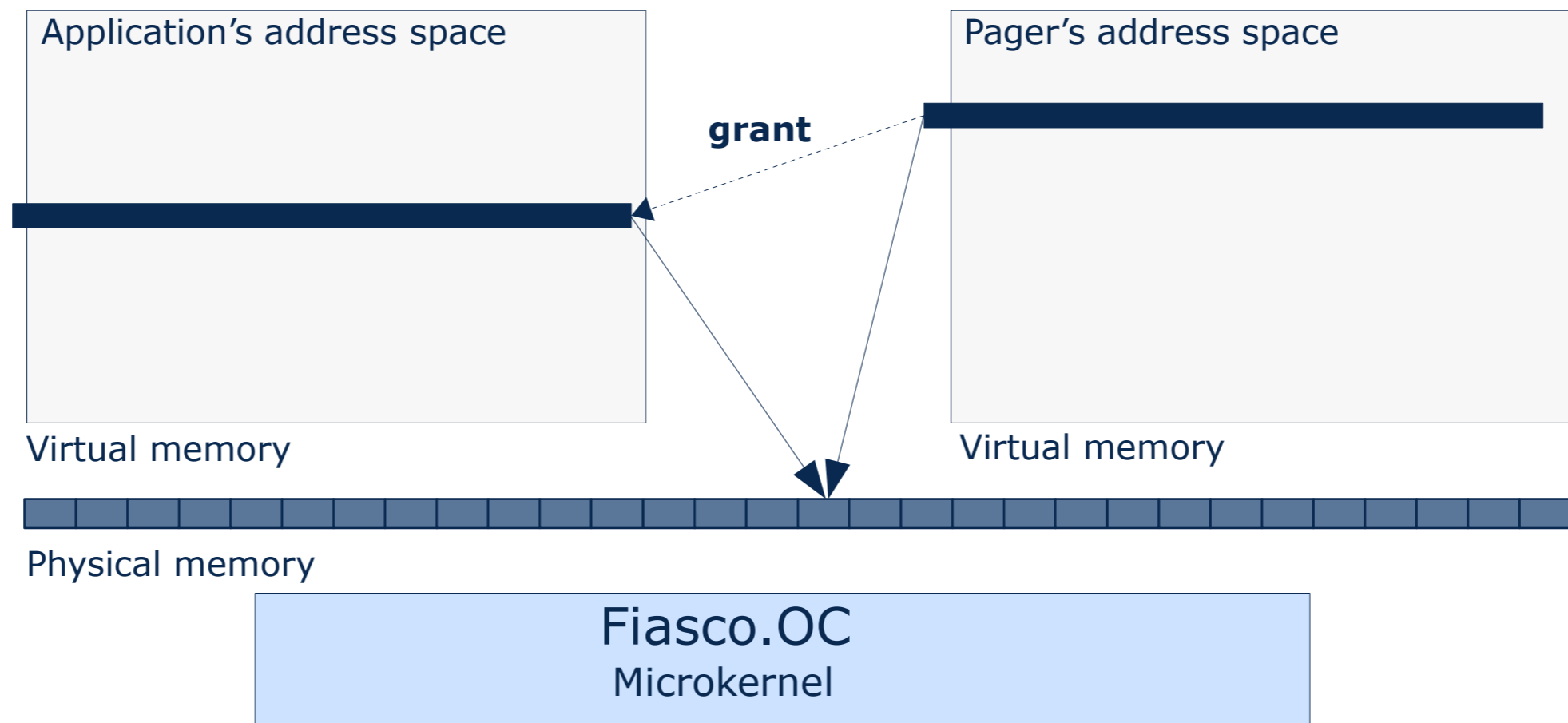
- Pager maps pages of it's own address space to the application's address space
- Flexpage IPC enables these mappings



- `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

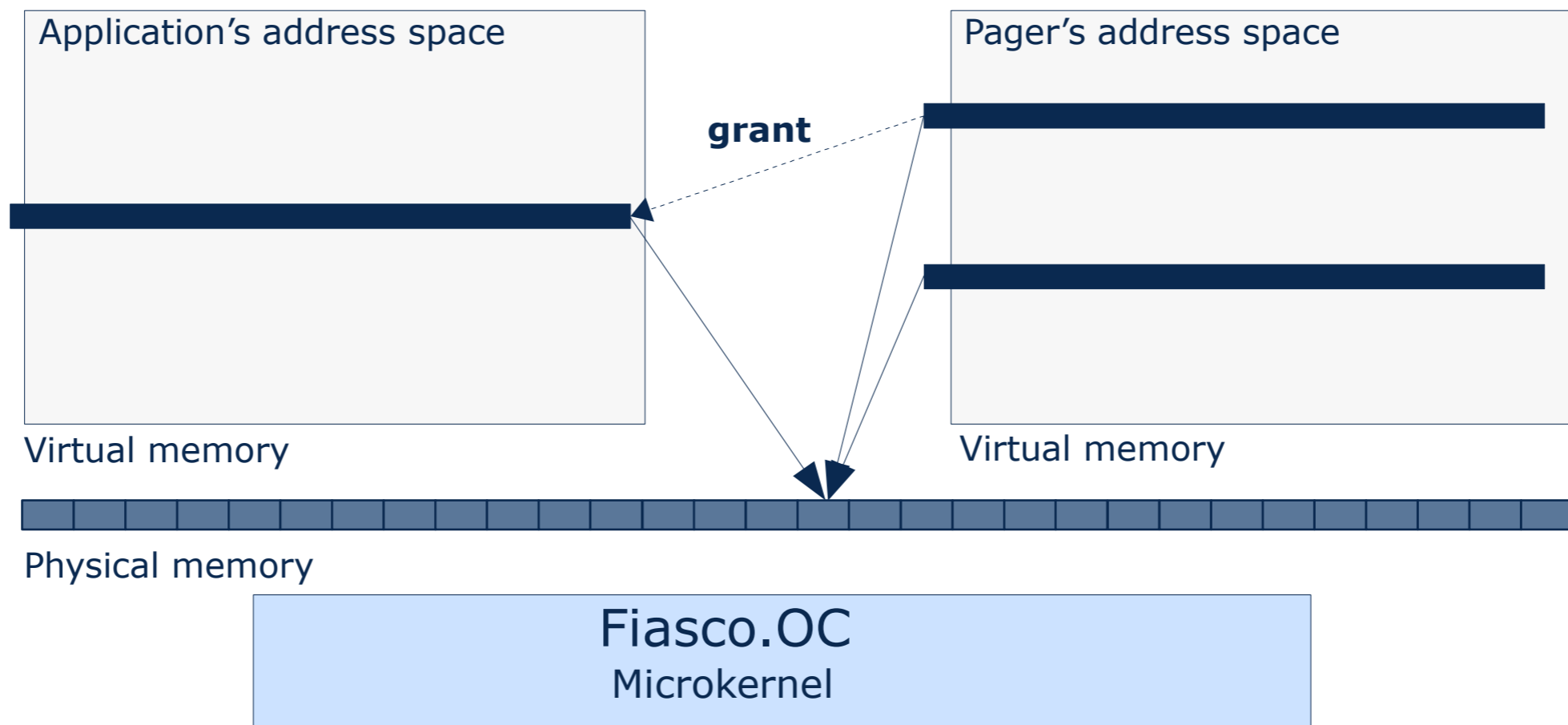


- Special case: grant pages (flag: L4\_FPAGE\_GRANT)
- Removes mapping from sender's address space

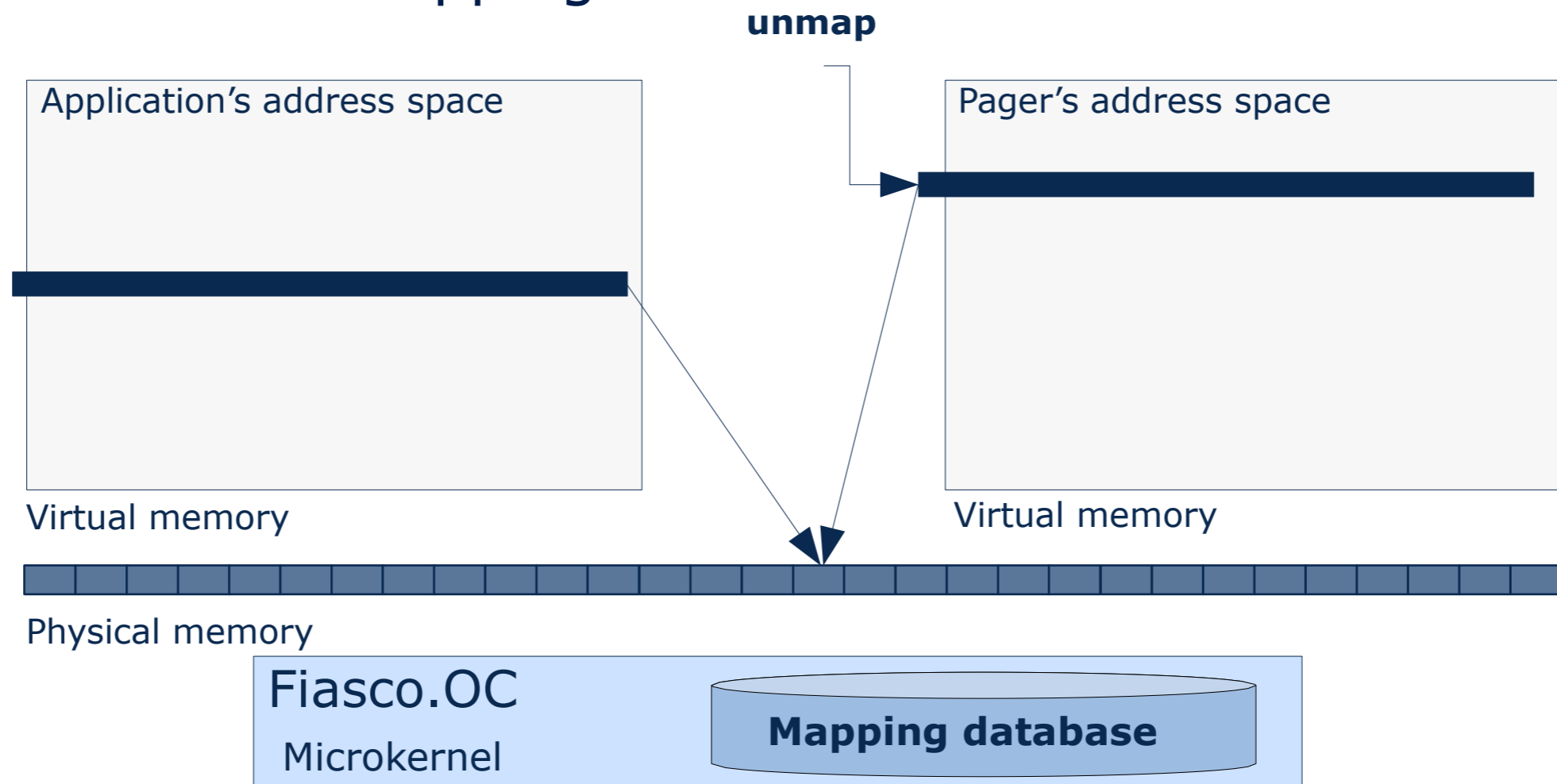




- Special case: grant pages (flag: L4\_FPAGE\_GRANT)
- Removes mapping from sender's address space
  - **ATTENTION: aliases remain**



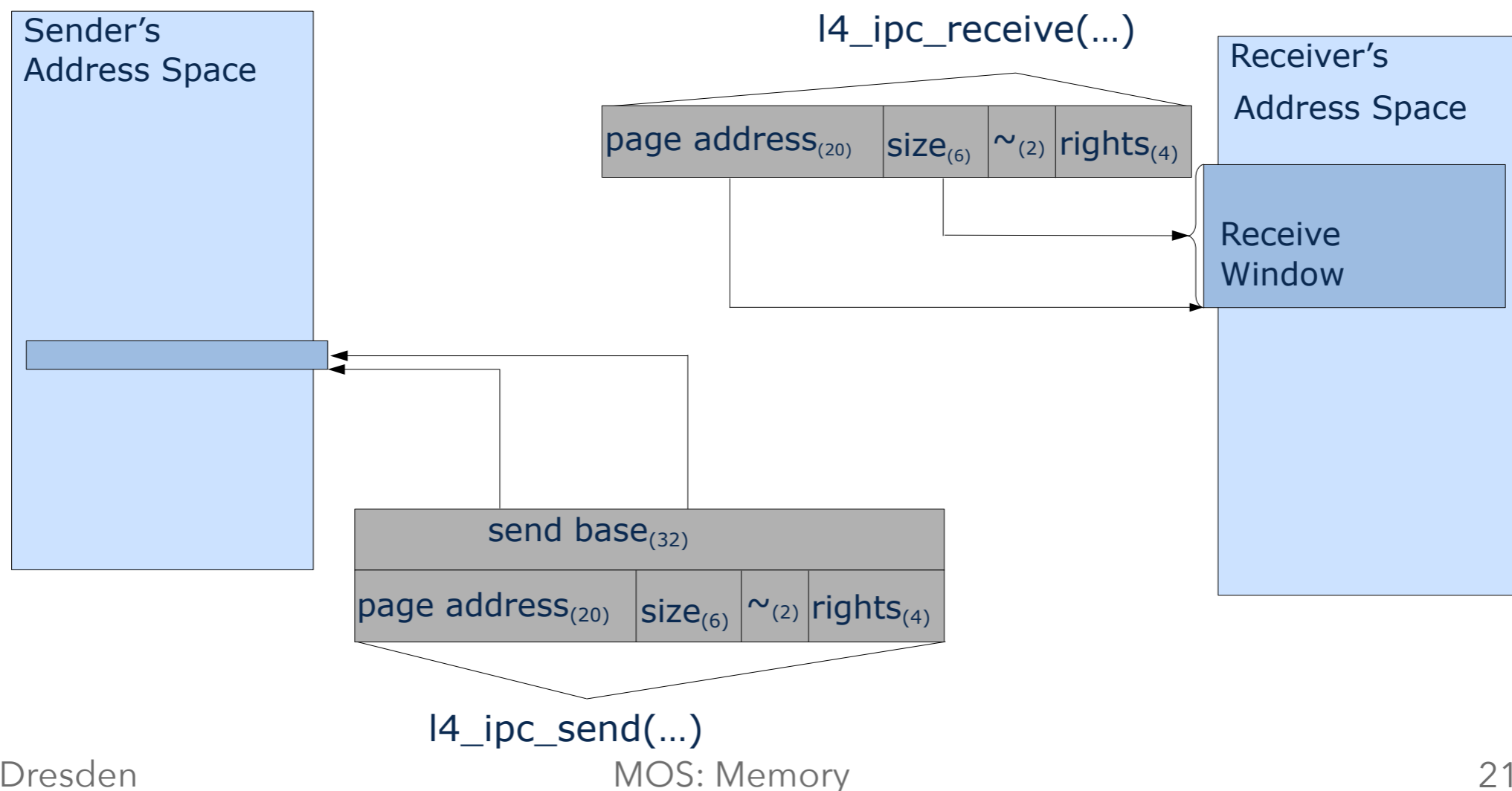
- Removes entries to a page frame (fpage is specified in invoker's address space)
- Dedicated system call: do not need partner's consent
- Kernel tracks mappings in a database



# FLEXPAGES

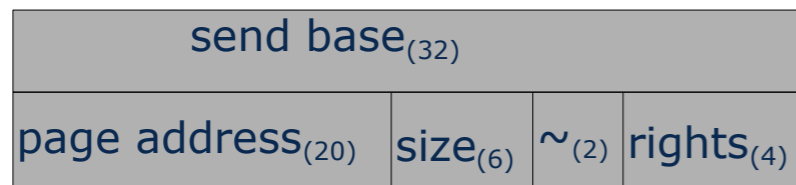
- Flexpages represent resources attached to an address space
- Flexpages in Fiasco.OC are used to describe:
  - Memory pages
  - I/O ports
  - Capabilities
- Today: only flexpages for memory

- 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

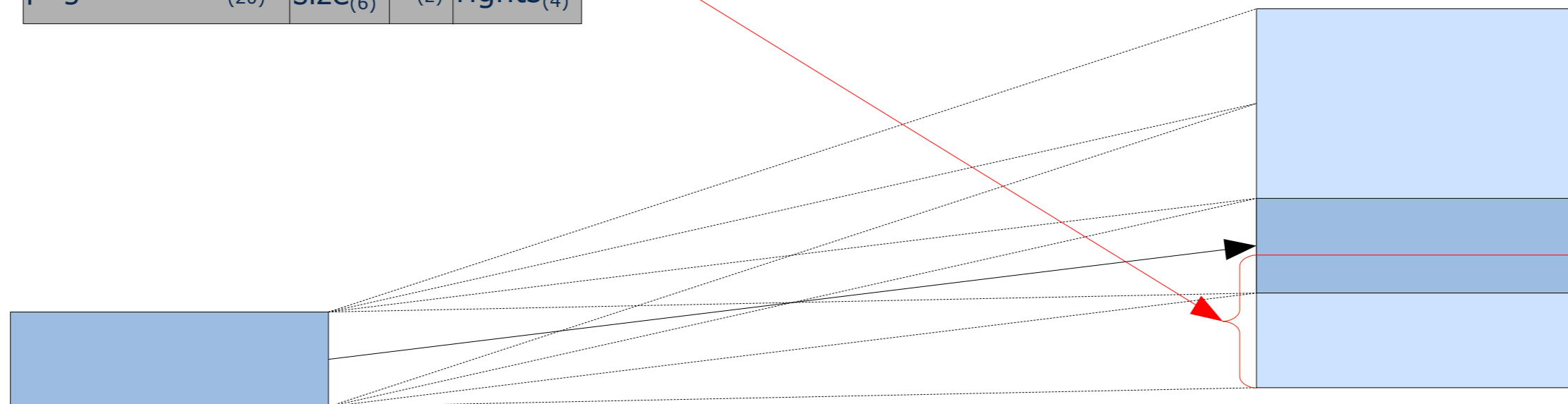
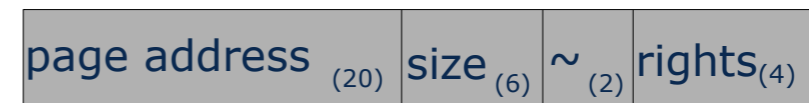


- Send flexpage is smaller than the receive window
  - Target position is derived from send flexpage alignment and send base

`l4_ipc_send(...)`

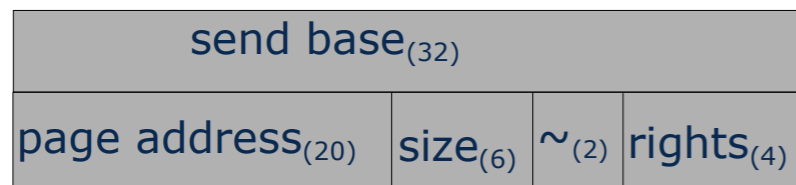


`l4_ipc_receive(...)`

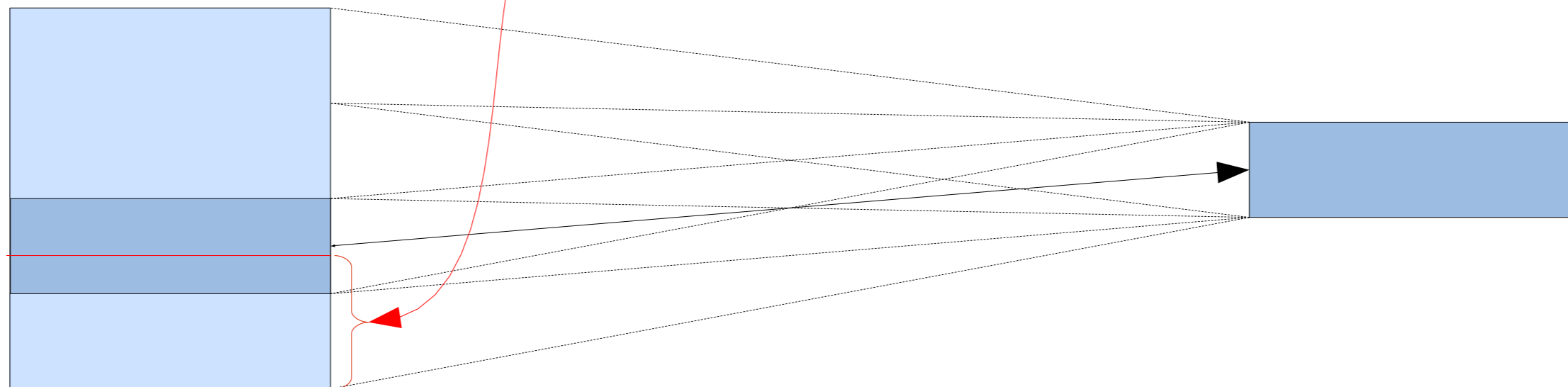
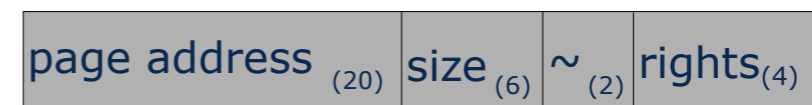


- Send flexpage is larger than receive window
  - Target position is derived from receive flexpage alignment and send base
- Send base depends on information about the receiver

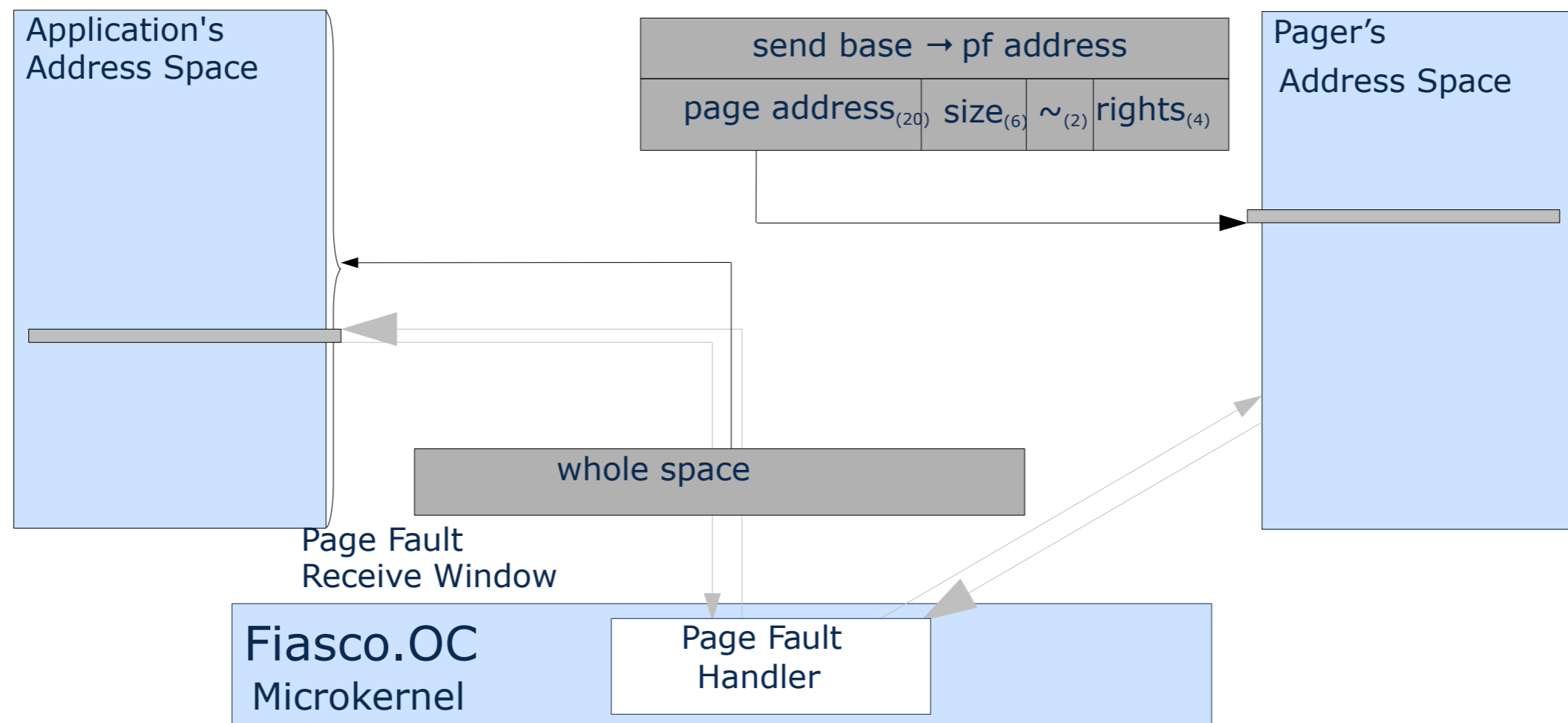
`l4_ipc_send(...)`



`l4_ipc_receive(...)`

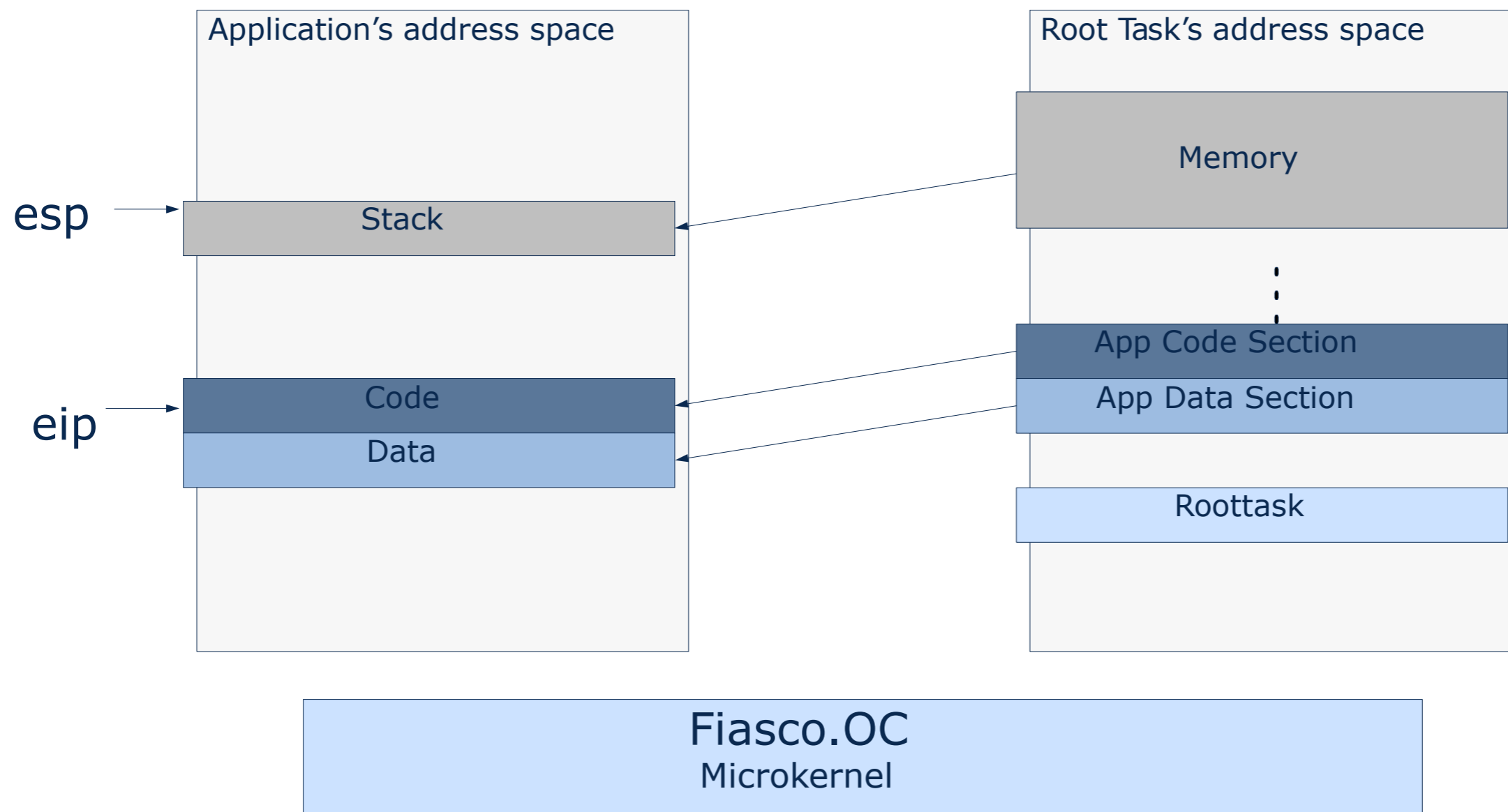


- Kernel page fault handler sets receive window to whole address space
- Pager can map more than just one page, where the page fault happened to the client

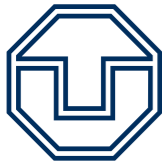




- Pages are mapped as they are needed  
→ *demand paging*

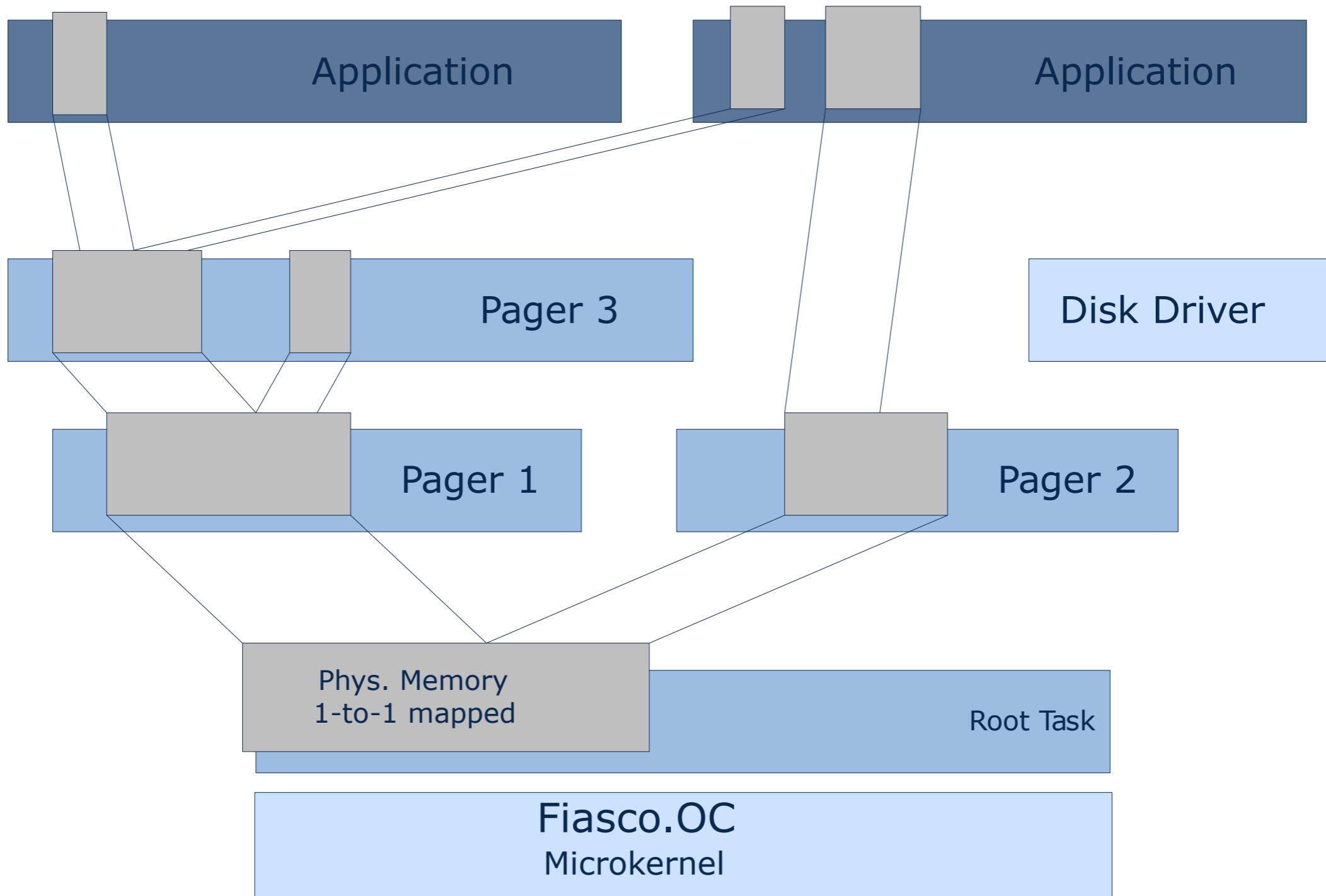


- Initial pager can only implement basic memory management
  - No knowledge about application requirements
    - Different requirements at the same time
  - Missing services for advanced memory management
    - e.g. no disk driver for swapping
  - Build more advanced pagers on top of the initial one
- Pager hierarchy

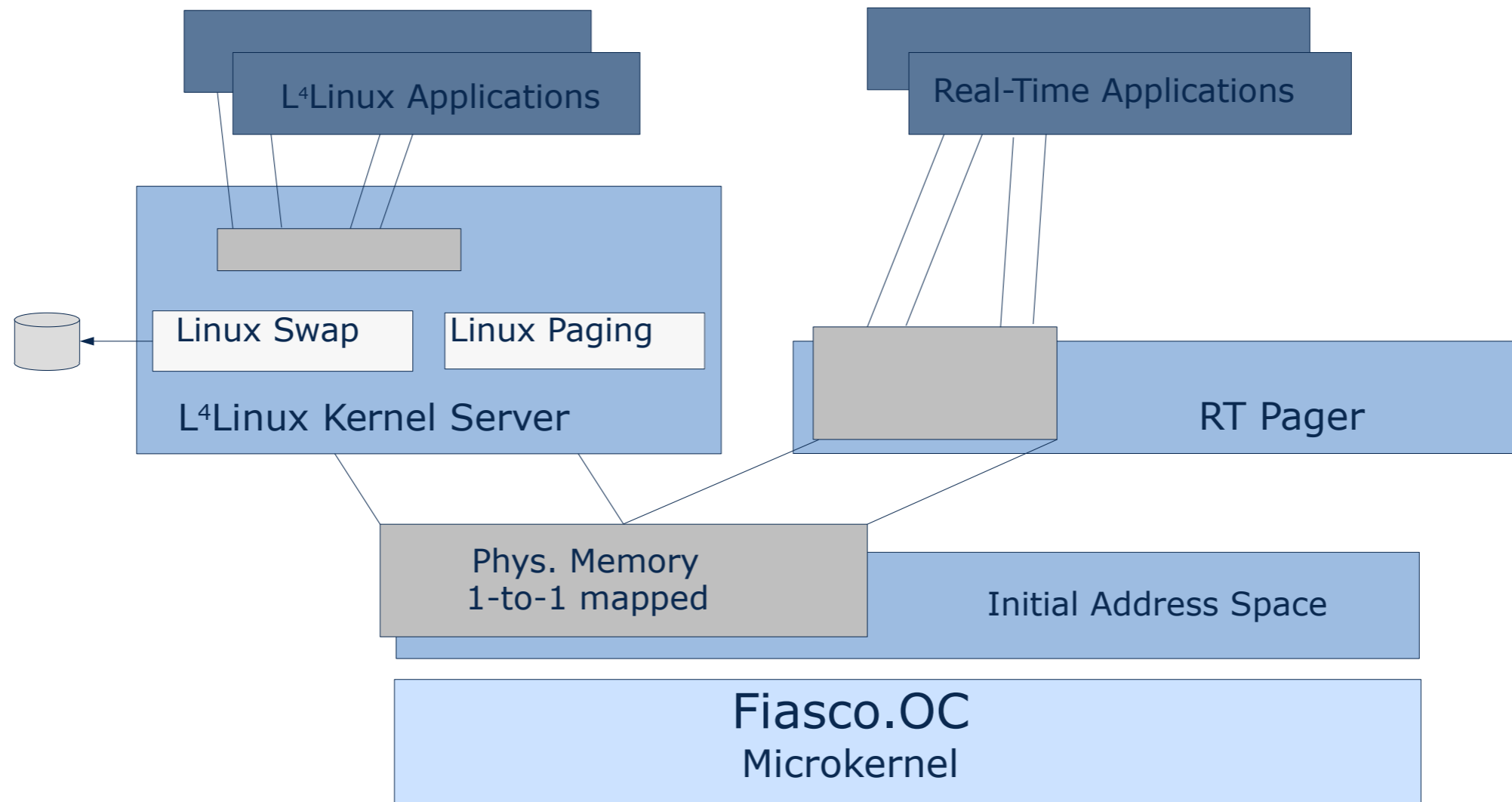


# HIERARCHICAL PAGERS

# PAGER HIERARCHY

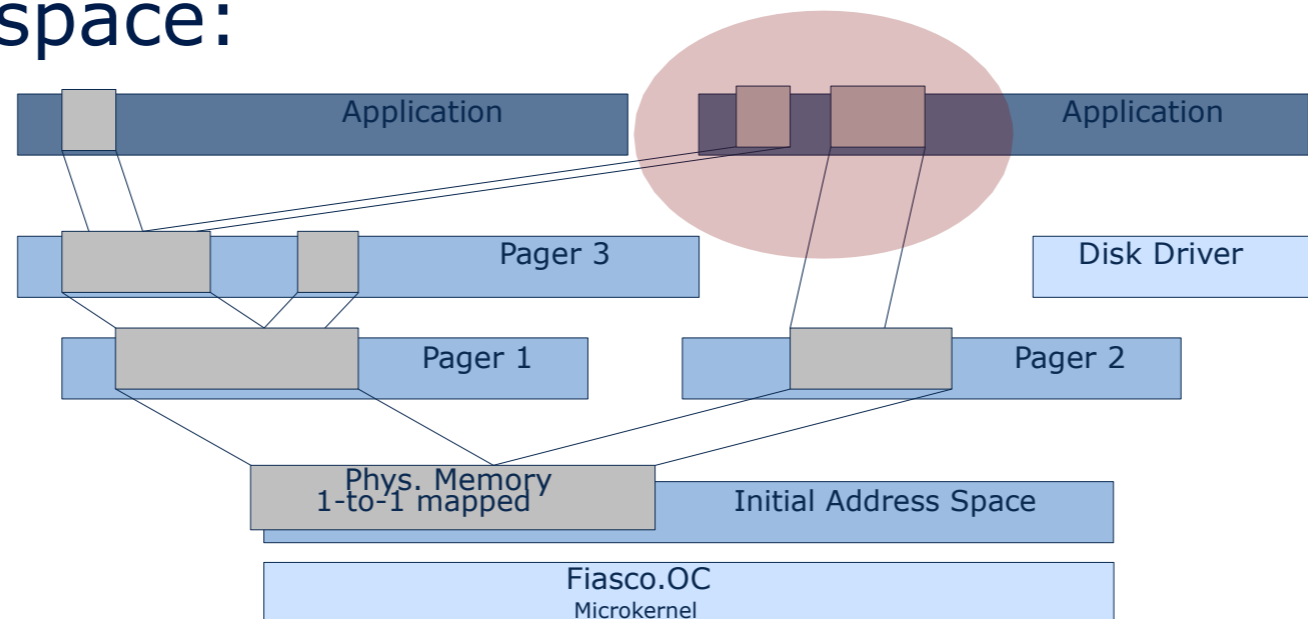


- L<sup>4</sup>Linux implements Linux paging policy
- RT pager implements real-time paging policy (e.g. no swapping)



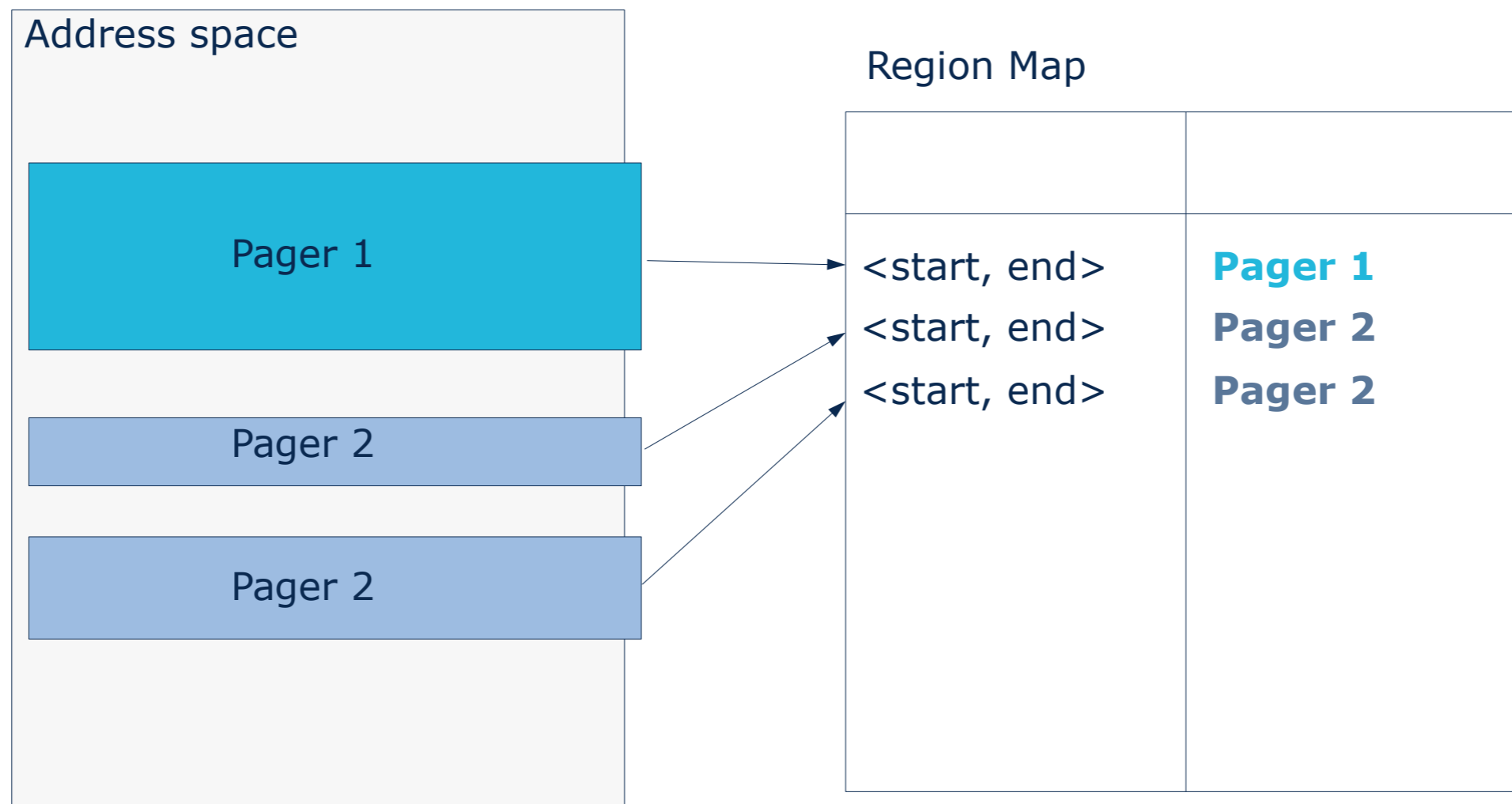
# REGION MANAGER

- Pager has to specify send base
- Pager needs to know client's address space layout
  - No problems with only one pager (e.g. L<sup>4</sup>Linux)
- Possible conflicts if more than one pager manages an address space:



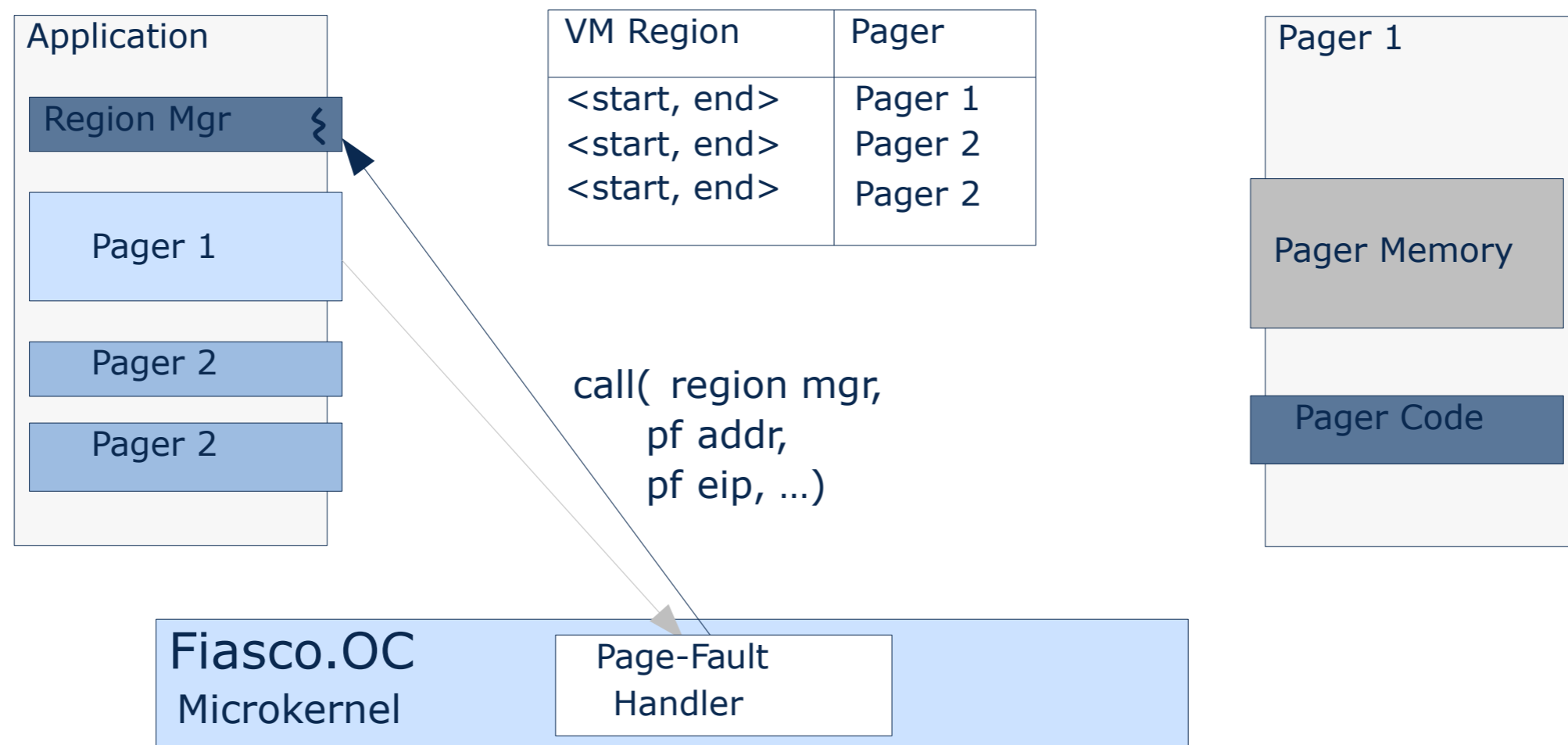
→ Virtual memory must be managed independent of pagers

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

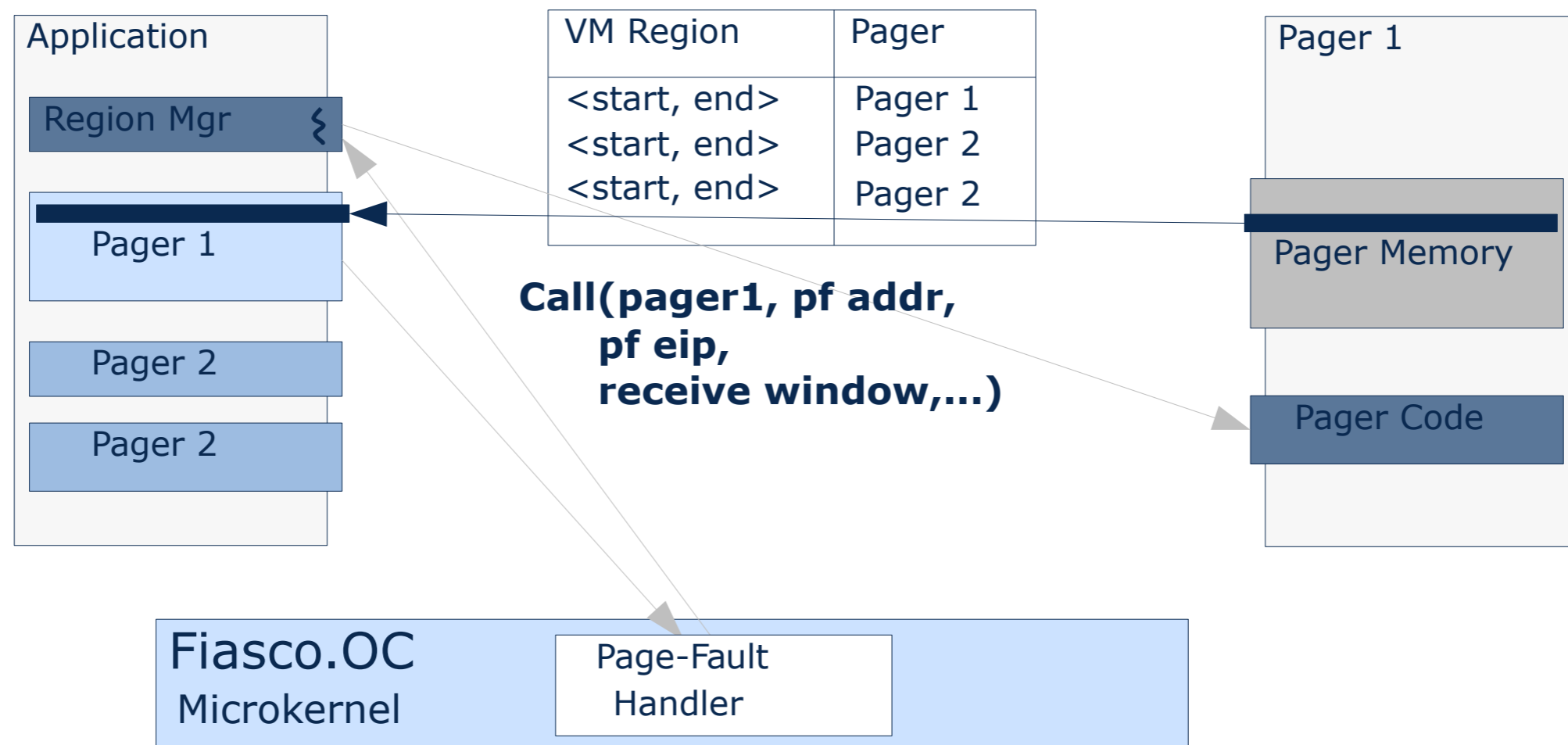




- Intermediate pager that identifies which pager should handle a page fault
- Resides in the application's address space
- Region manager is the pager of all threads of a task



- Region manager calls the pager that is responsible
  - Receive window gets restricted to the area managed by that pager
- No interference between different pagers

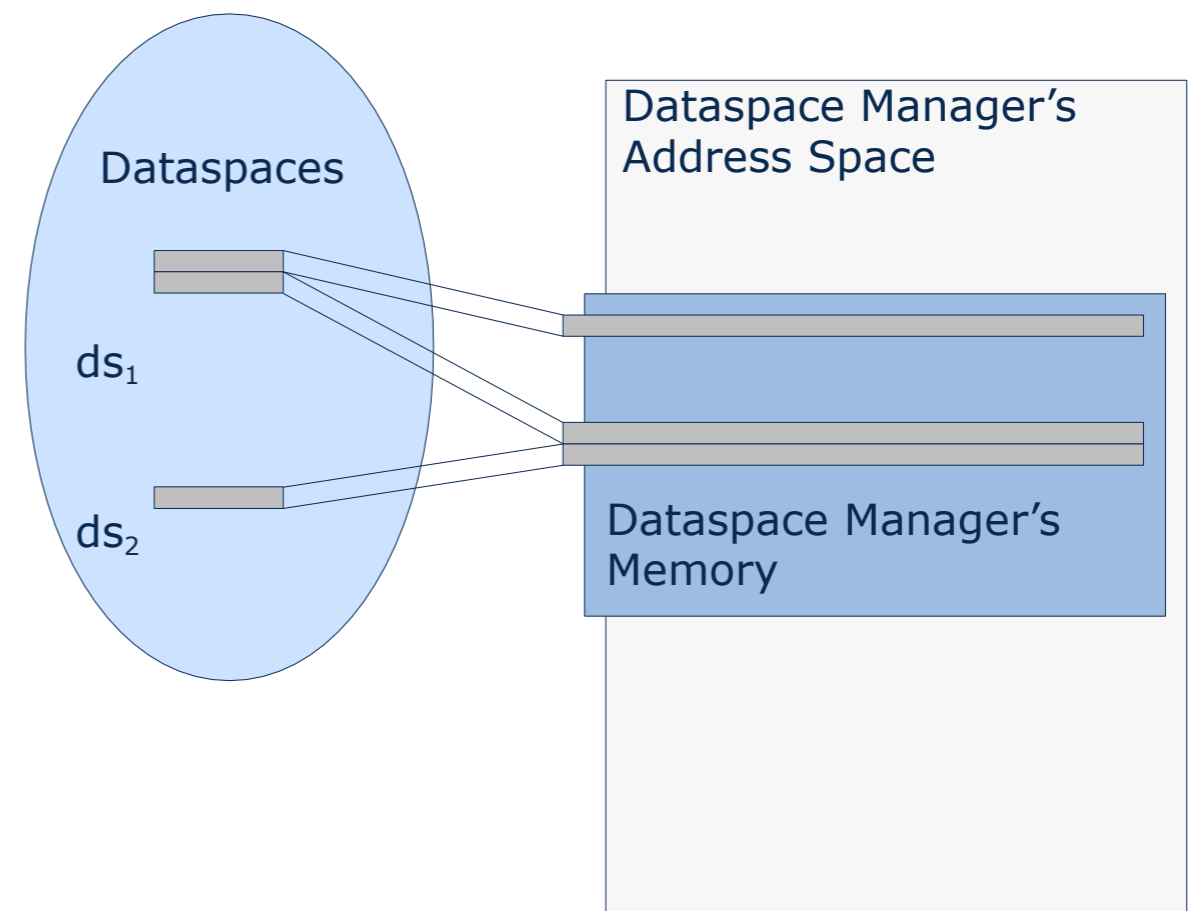


- Memory management in terms of pages so far
  - Application's view to memory:
    - code / data sections
    - memory mapped files
    - anonymous memory (heaps, stacks, ...)
    - network / file system buffers
    - ...
- Abstraction to map this view to low-level memory management

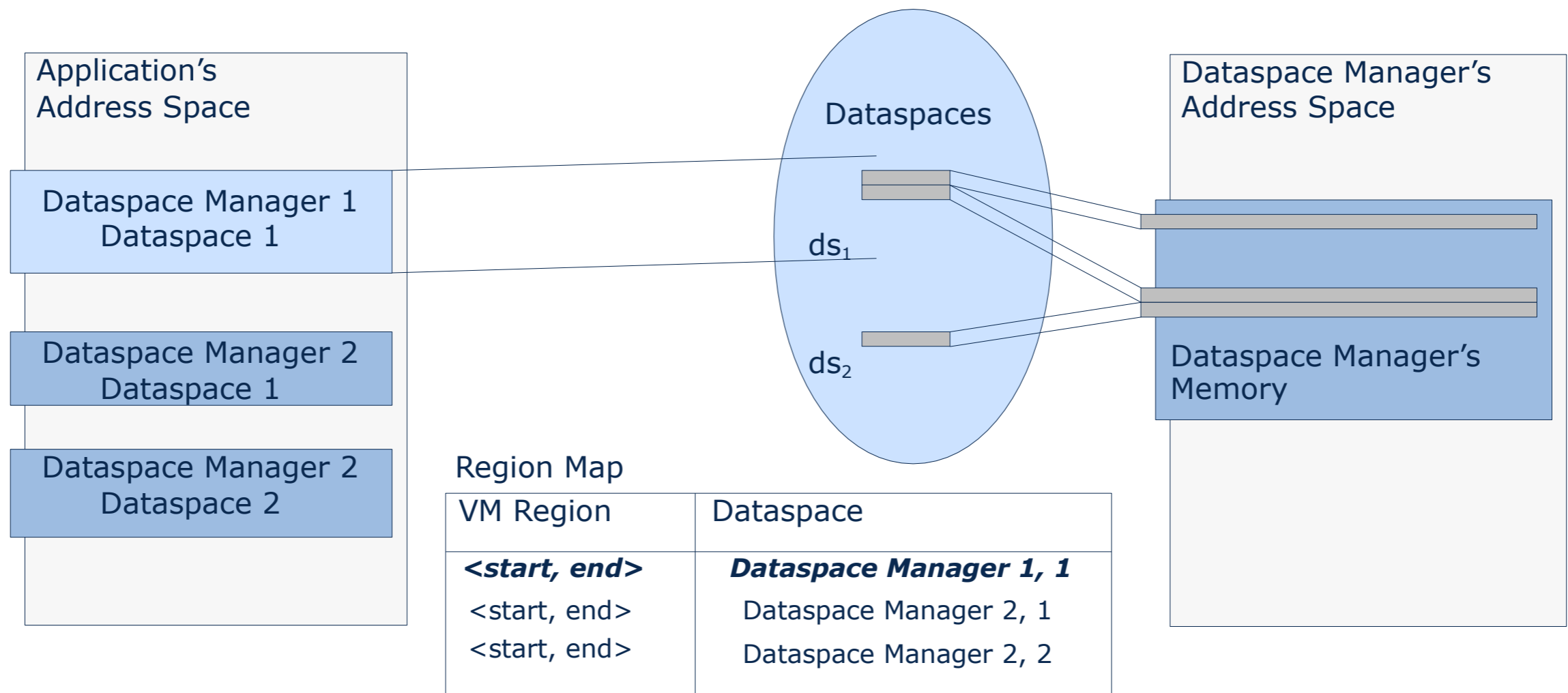
# DATASPACE

- 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

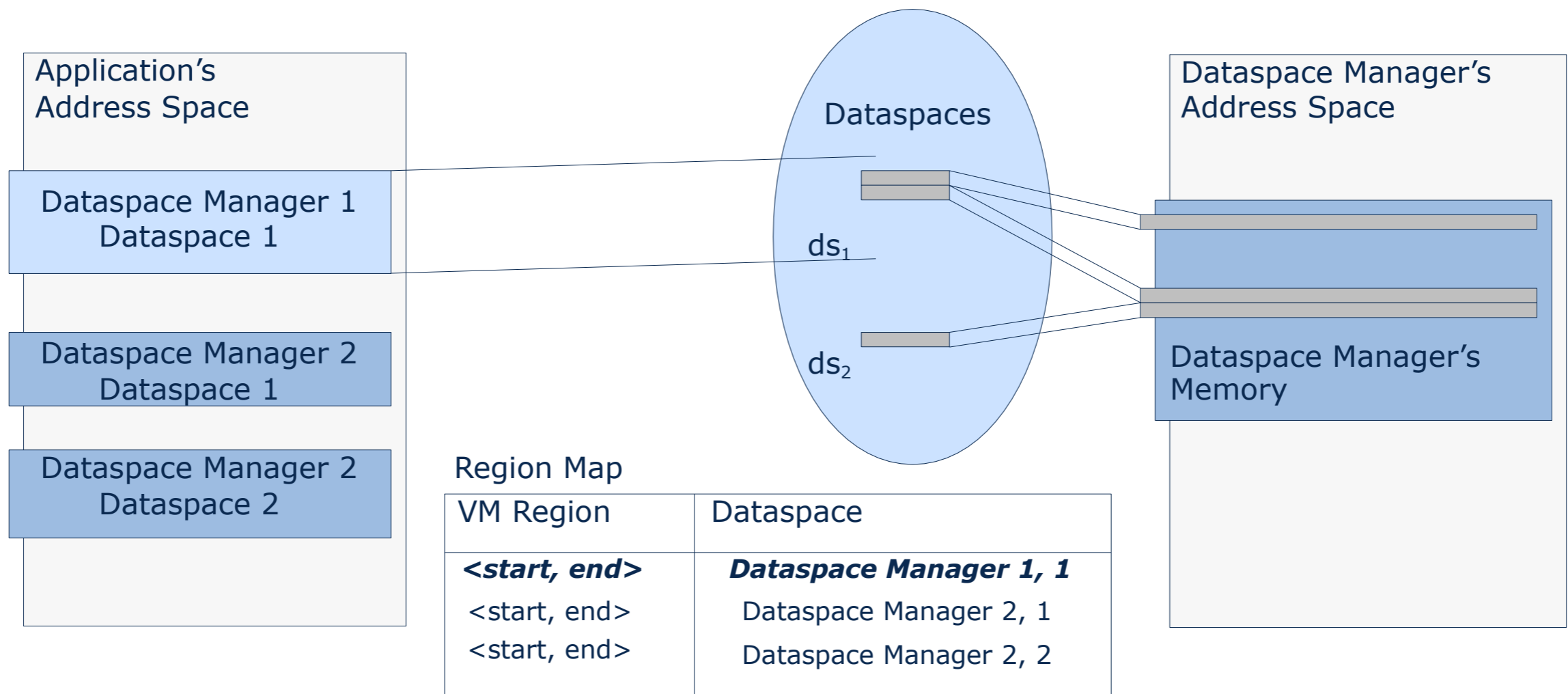
- DS Manager determines the semantic of a dataspace
- Each DSM is the pager for its dataspace
- Implements the paging policy (page replacement etc.)



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



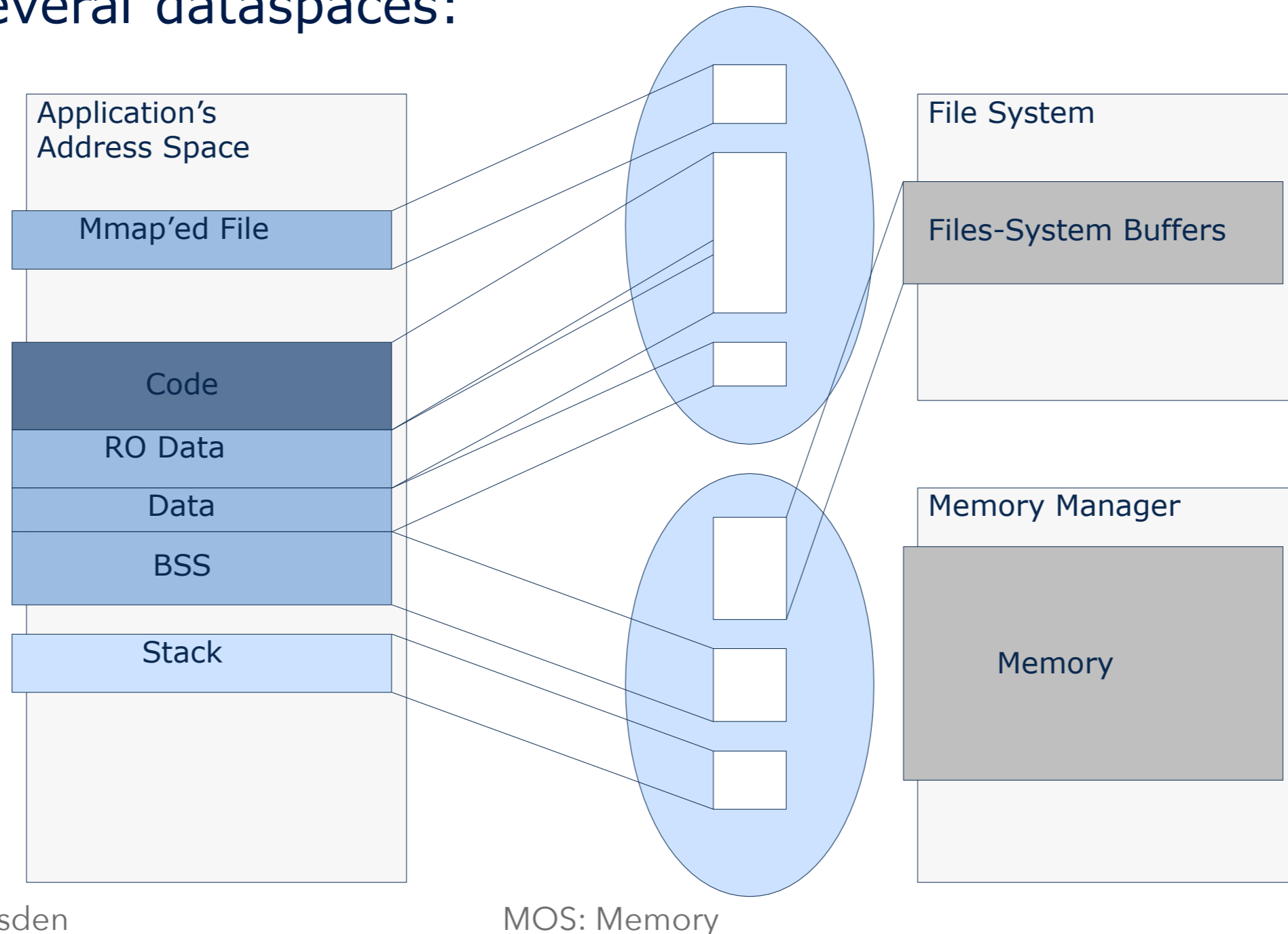
- Region manager propagates fault to dataspaces manager's fault handler
- Dataspace fault (ds\_manager\_id, ds\_id, offset)





- allocate / free dataspace
  - 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 are constructed from several dataspaces:



- 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 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

- Closer look on tasks/threads:
  - Creation
  - Page-fault handling
- Flexpages
  - Memory pages, I/O ports, Capabilities
  - Structure
  - Offset computation
- Pager hierarchy
- Region manager & dataspaces

- Flexpages

H.Härtig, J.Wolter, J.Liedtke: "*Flexible sized page objects*",  
[http://os.inf.tu-dresden.de/papers\\_ps/flexpages.pdf](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](ftp://ftp.cse.unsw.edu.au/pub/users/disy/papers/Aron_PJLED_01.ps.gz)