

The B-Method for the Construction of Microkernel-Based Systems

Sarah Hoffmann (STMicroelectronics)

sarah.hoffmann@st.com



International Workshop on System Verification, Sydney, 2006

Secure Embedded Systems

Advanced System Technology Security Research Group

- research on prevention of attacks on applications in embedded systems
- software research focused on microkernel-based systems
 - allow slim general-purpose systems
 - good confinement with minimal performance loss
- additional requirement: systems with high-level of confidence

The B4L4 Project

1. Application-private information must stay private unless the application explicitly allows to share it.
2. The establishment of all communication channels is controlled by the system.

- started as case study to formally define API of L4 microkernel
- extended to formally prove fundamental security properties of a complete L4-based system

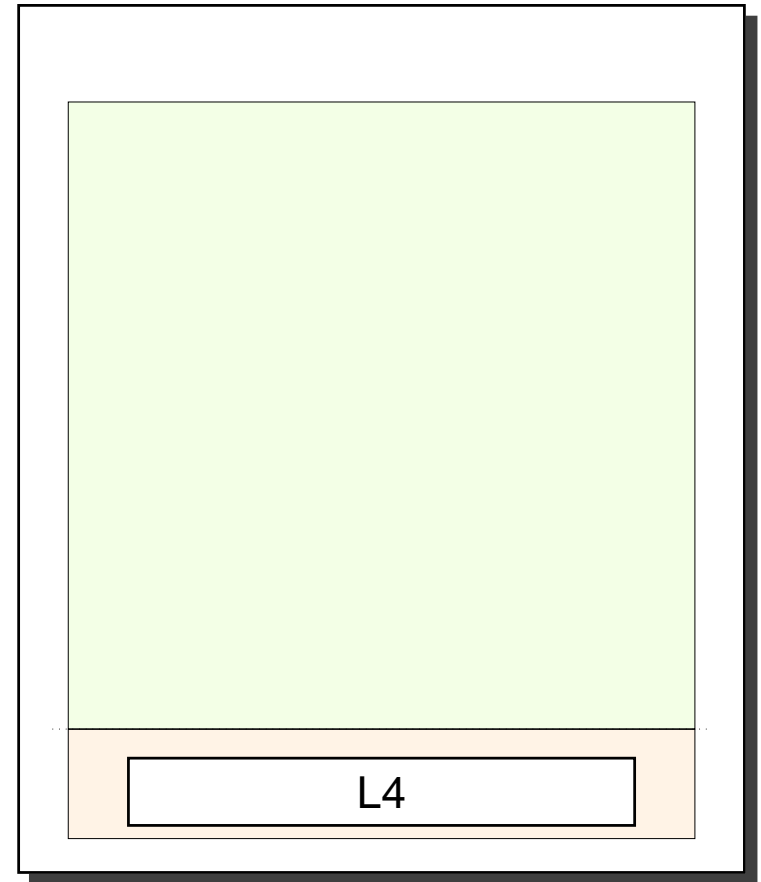
- Introduction to L4 and B
- Model of the L4 Kernel API
- Model of an L4-based secure system (L4 Core)
- B Animation for test case generation

L4 Microkernel

- minimum of code in privileged mode
- abstraction of hardware
- no policy

ST-modified version of L4 X.2:

- *11 system calls*
+ protocols for page faults,
interrupts, ...
- *15.000 LOC*



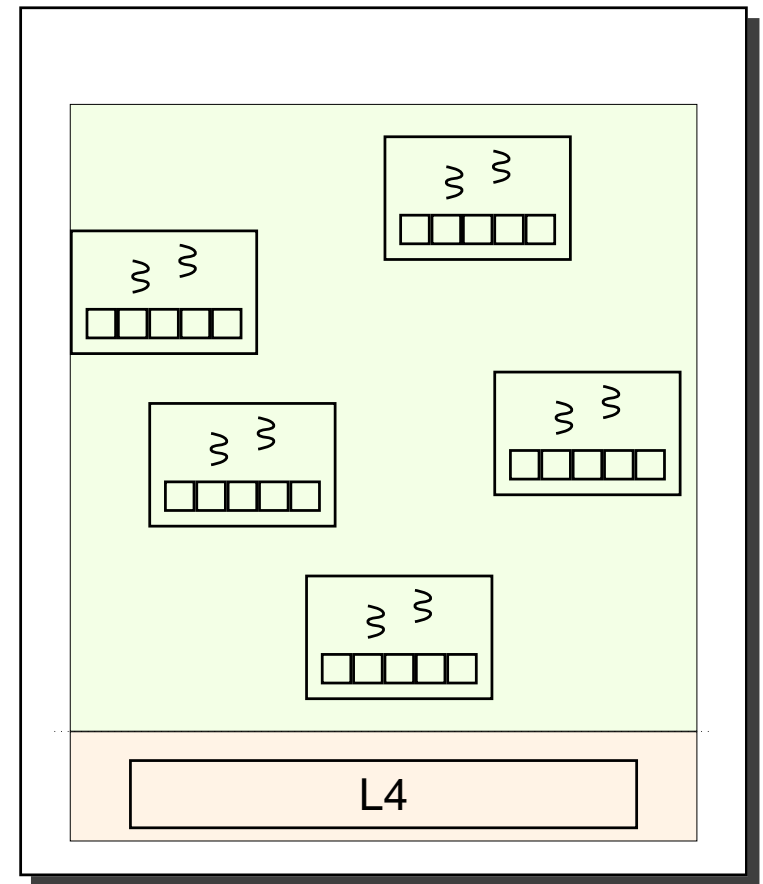
L4 Kernel API(I)

Tasks - address spaces

- virtual memory implementation using MMU
- container for all other objects (threads)

Threads - execution context

- priority-based preemptive round-robin scheduling



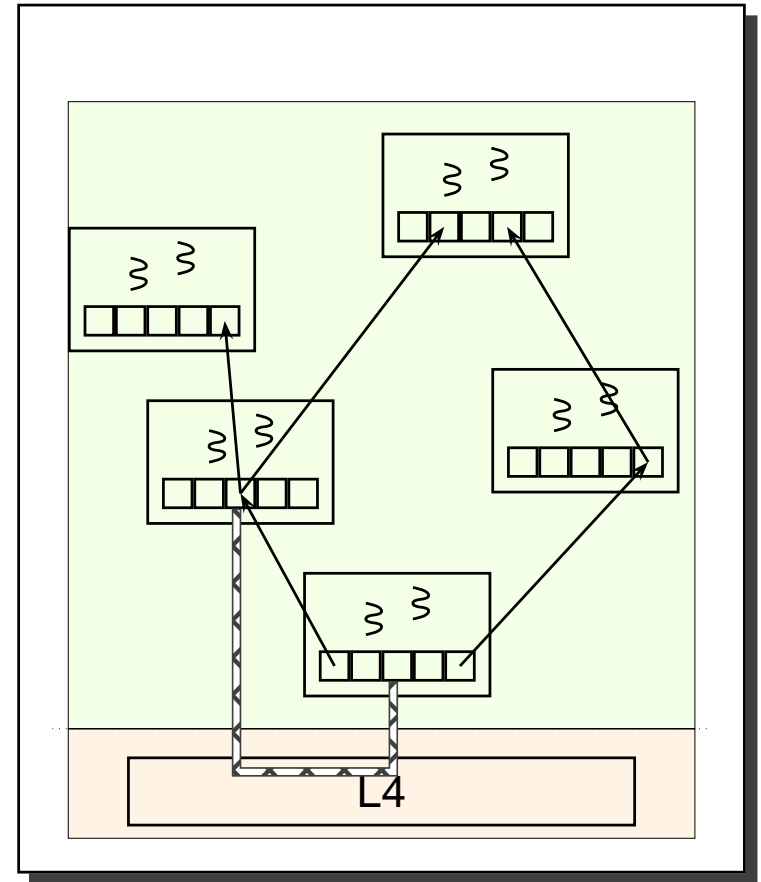
L4 Kernel API(II)

Mapping - memory management

- decentralized memory transfer
- results in mapping trees for each page
- *added: control of mapping rights between tasks*

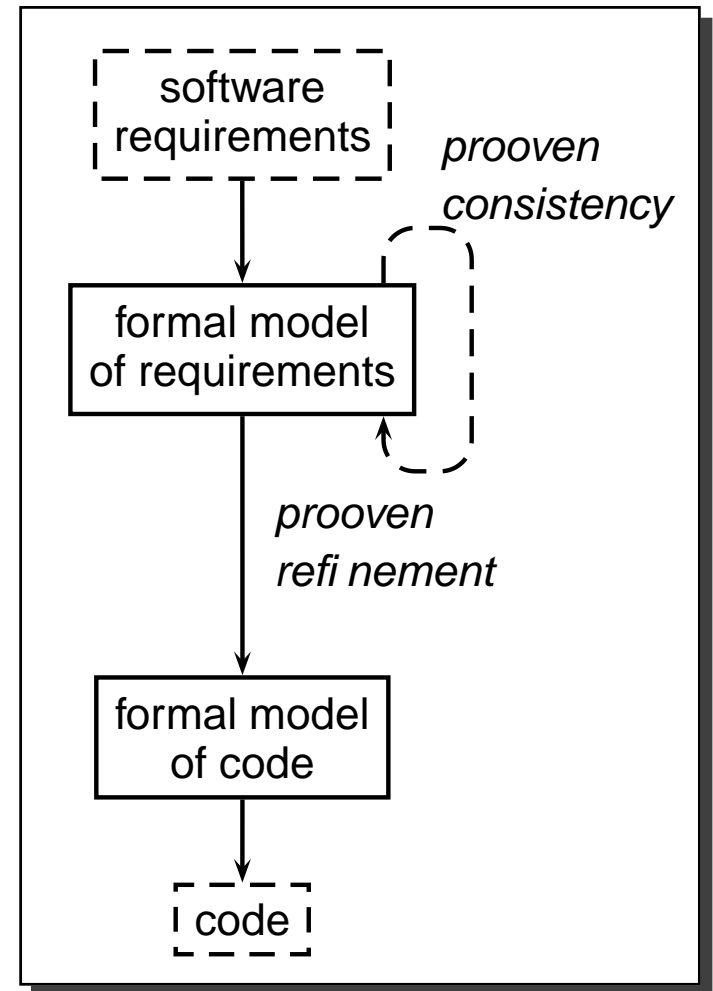
IPC - thread communication

- untyped, synchronous
- generalizes interrupts, page faults, exceptions
- *added: communication control via explicit channel creation*



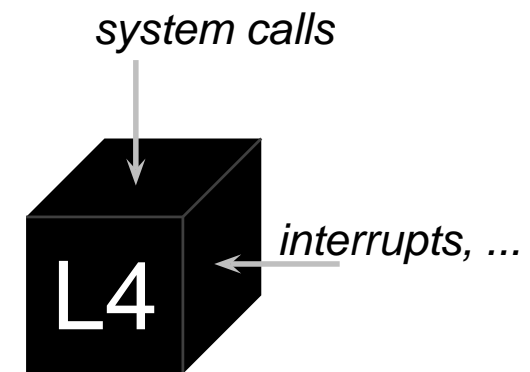
- formal method for system modelization
- consistent refinements to descent to code level
- based on abstract machines
 - state with its invariants
 - events
 - guard (precondition)
 - state transition

formal language based on set theory and first-order logic



Kernel API Model - Goals

- constraint: pre-existing API
 - written with specific implementation in mind
 - three kinds of definition: general, architecture-dependent, implementation-dependent
- pure API Model without reference to existing implementations
- events: defined by system calls and external events
- state: added as required
- no B model refinements



Kernel API Model - Events

```
word ThreadControl(ThreadId dest, ThreadId SpaceSpecifier,  
                  ThreadId scheduler, ThreadId pager, void* UtcbLocation)
```

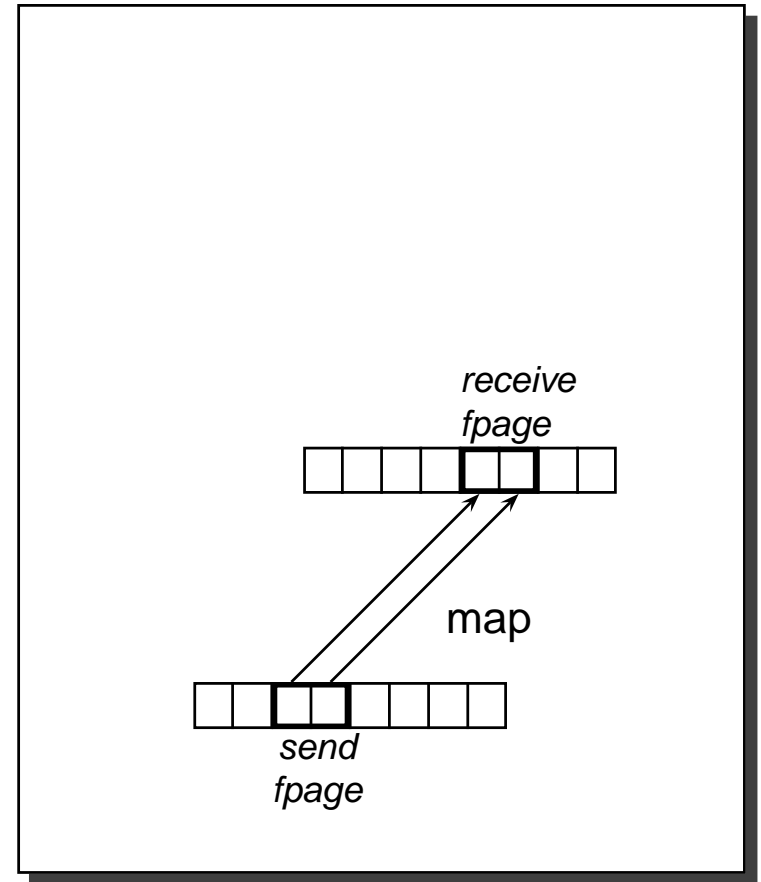
```
L4_ThreadControl =  
  ANY dest, space, scheduler, pager, utcb  
  WHERE [...]  
  THEN  
    SELECT dest  $\notin$  existing  
    THEN  
      SELECT dest = space  
      THEN  
        ThreadCreation_NewAS  
      WHEN dest  $\neq$  space  
      THEN  
        ThreadCreation_ExistAS  
    WHEN dest  $\in$  existing  
    THEN  
      ...
```

```
L4_ThreadControl_Create_NewAS =  
  ANY [...]  
  WHERE  
    dest  $\notin$  existing  $\wedge$  space = dest  $\wedge$  [...]  
  THEN  
    ThreadCreation_NewAS  
  END
```

```
L4_ThreadControl_Create_ExistAS =  
  ANY [...]  
  WHERE  
    dest  $\notin$  existing  $\wedge$  space  $\neq$  dest  $\wedge$  [...]  
  THEN  
    ThreadCreation_ExistAS  
  END  
...
```

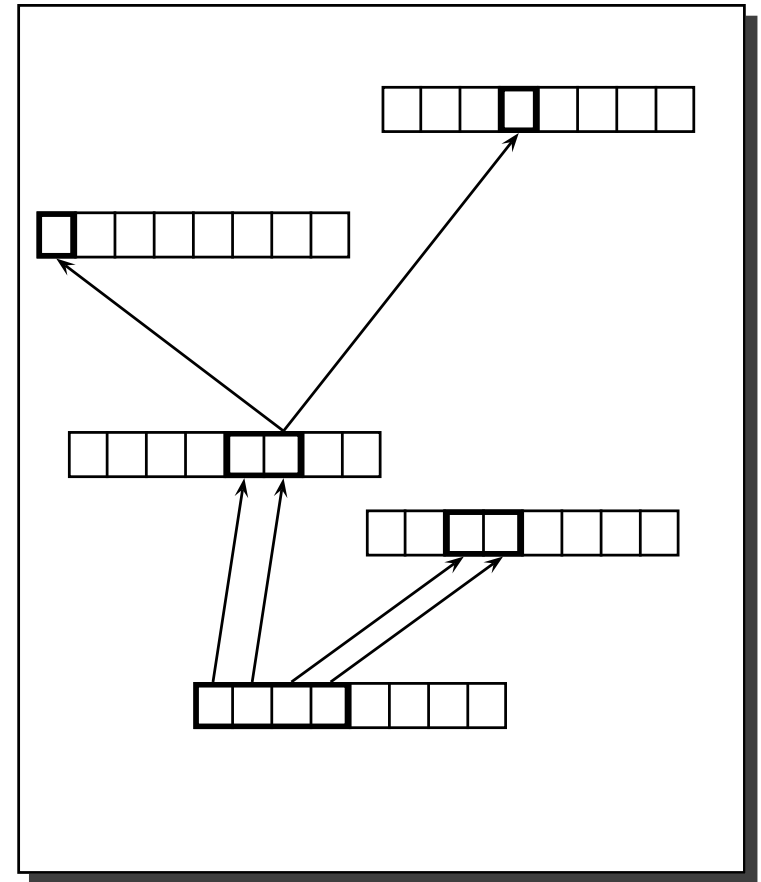
Kernel API Model - Mapping(I)

- memory organized in *fpages*
 - size of 2^n
 - aligned to multiple of size



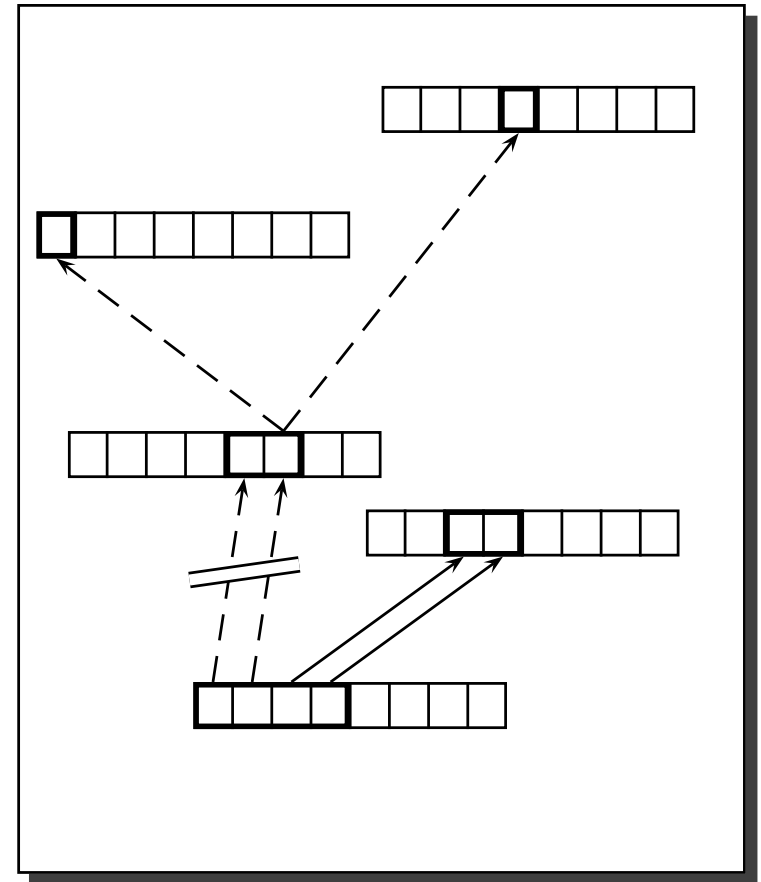
Kernel API Model - Mapping(I)

- memory organized in *fpages*
 - size of 2^n
 - aligned to multiple of size
- root space contains physical memory



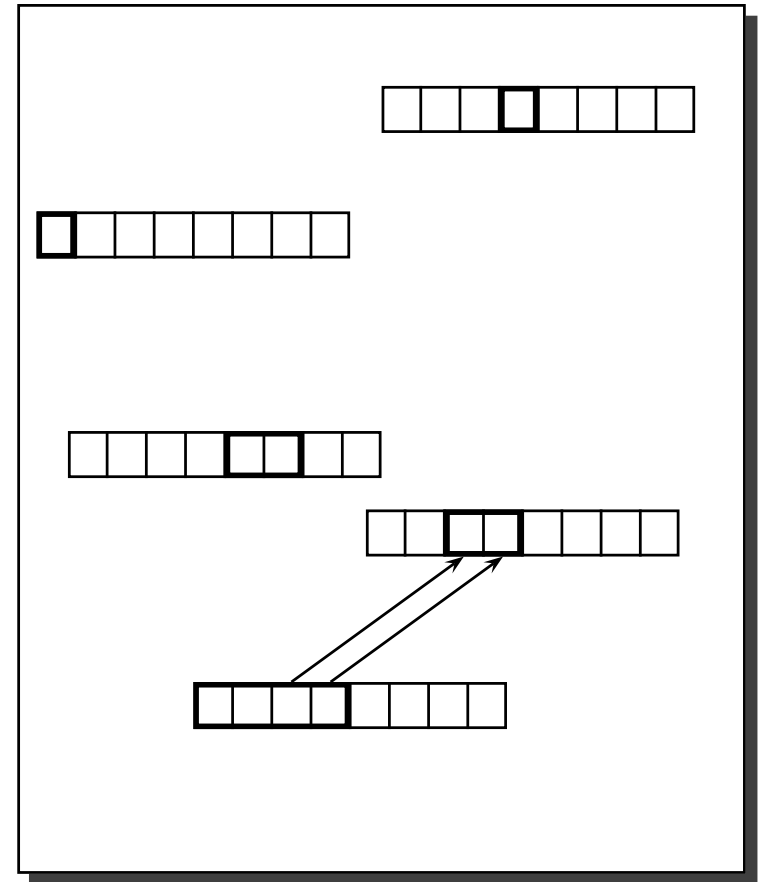
Kernel API Model - Mapping(I)

- memory organized in *fpages*
 - size of 2^n
 - aligned to multiple of size
- root space contains physical memory
- *unmap* removes complete subtree



Kernel API Model - Mapping(I)

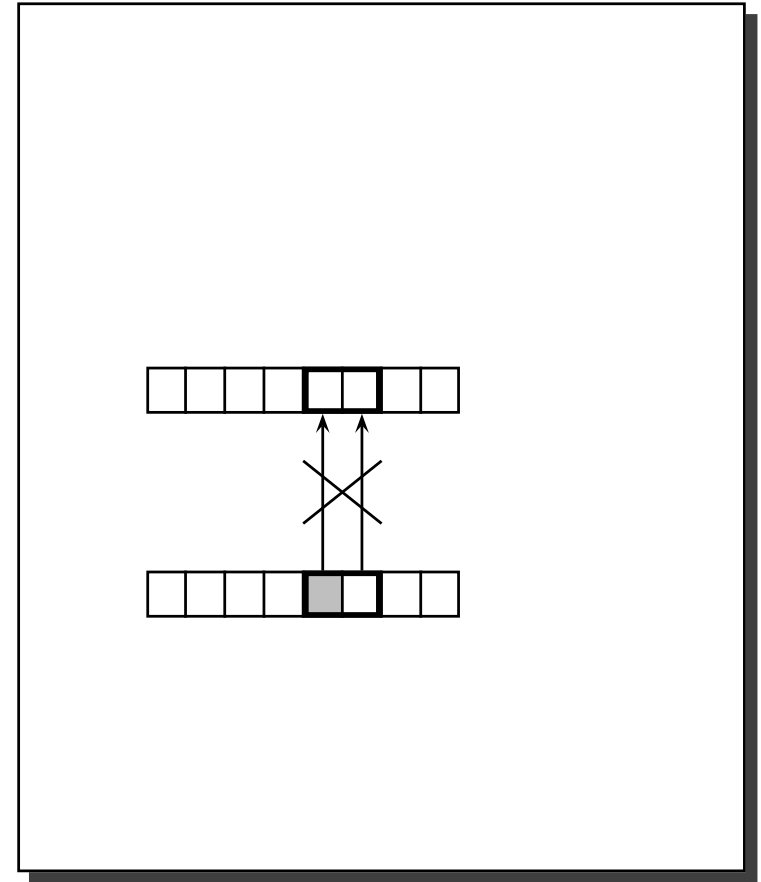
- memory organized in *fpages*
 - size of 2^n
 - aligned to multiple of size
- root space contains physical memory
- *unmap* removes complete subtree
- API only defines nominal use-cases, all others are implementation-defined



Kernel API Model - Mapping(II)

model behaviour

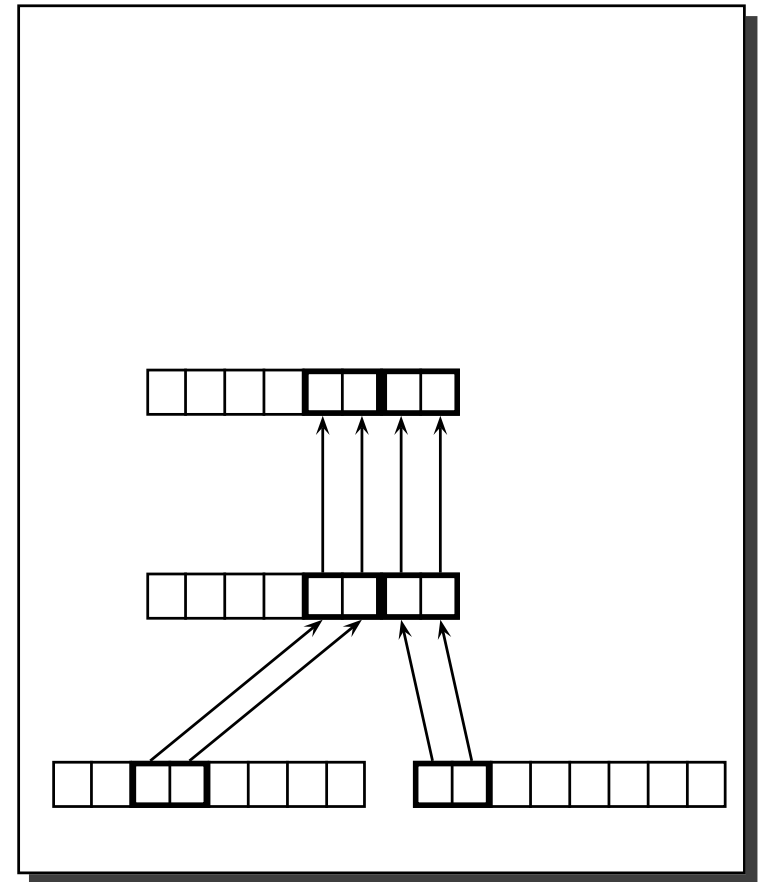
- existing mappings are not split



Kernel API Model - Mapping(II)

model behaviour

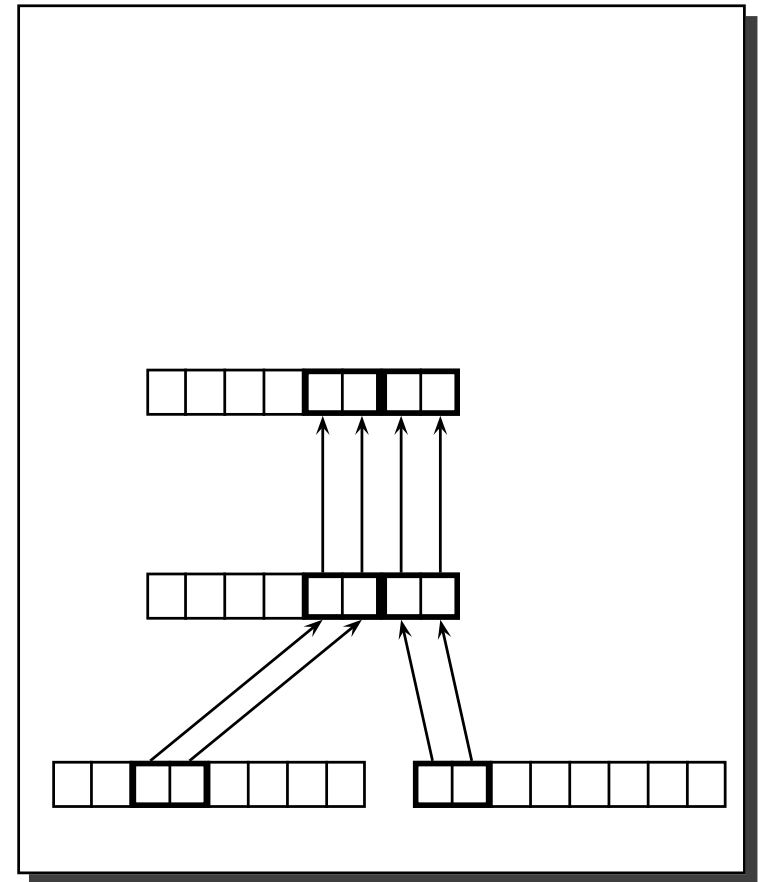
- existing mappings are not split
- new mappings are not joined



Kernel API Model - Mapping(II)

model behaviour

- existing mappings are not split
- new mappings are not joined
- map behaves like unmap followed by map



Kernel API Model - Mapping Model(I)

- **t_page**: set of virtual pages

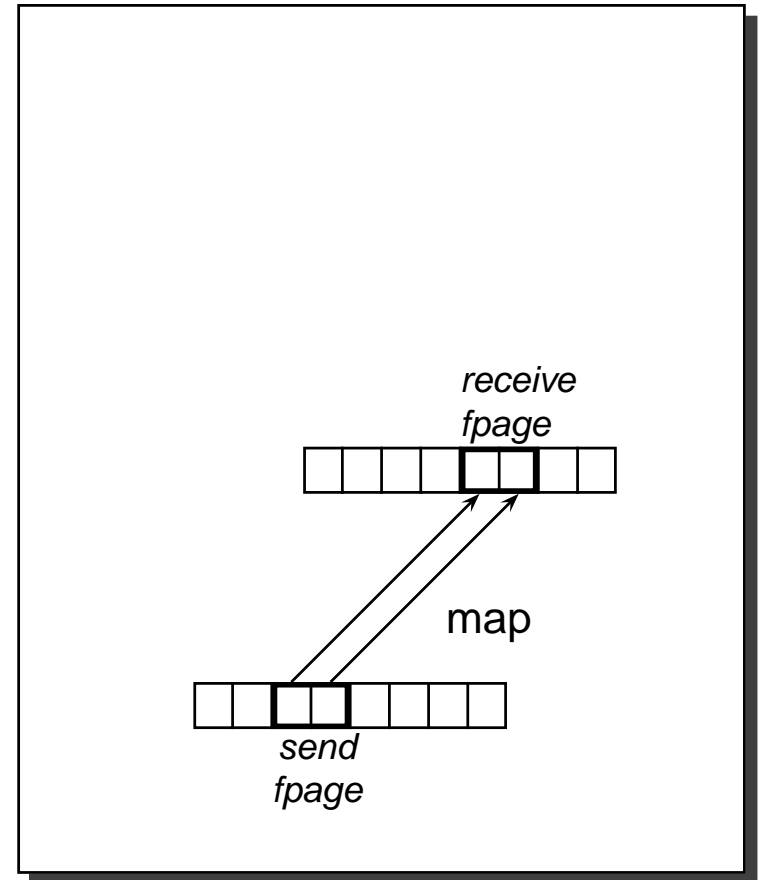
$\text{spaceMem} : \text{t_space} \rightarrow (\mathcal{N} \rightarrow \text{t_page})$

- **mapping**: set of relations between two t_pages

$\text{mapping} \subset \text{t_pages} \times \text{t_pages}$

- **mapping database**: set of mappings

$\text{mappingDB} \subset \mathcal{P}(\text{t_pages} \times \text{t_pages})$



Kernel API Model - Mapping Model(II)

map event: send fpage (b_s, s) to receive fpage (b_r, s)

1. find intervals in send page containing mappings

$$I = \{i_s, i_e | [\dots] \wedge \exists m \in \text{mappingDB} : \\ [i_s, i_e] = [i_s - 1, i_e + 1] \cap \text{mapRange}(m, \text{sender}, b_s, s)\}$$

with

$$\text{mapRange}(m, sp, b, s) = \{y | (0 < y < s) \wedge \text{spaceMem}(sp)(b + y) \in \text{ran}(m)\}$$

2. construct set of new mappings

$$M_m = \lambda(m_s, m_e).(\lambda y. \text{spaceMem}(\text{sender})(b_s + y) \mapsto \text{spaceMem}(\text{recvr})(b_r + y)) \\ [m_s, m_e][I]$$

3. construct set of mappings to remove

$$M_u = \{m \in \text{mappingDB} \mid \text{ran}(m) \subseteq (\bigcup \text{mappingDB})^*[\text{ran}(\bigcup M_m)]\}$$

4. new state

$$\text{mappingDB} := (\text{mappingDB} - M_u) \cup M_m$$

Kernel API Model - Status

- model complete including error cases
 - 49 state variables, 152 events, 150 lines of invariant properties
- corner cases and small omission uncovered
- successfully used to ensure consistent API extension

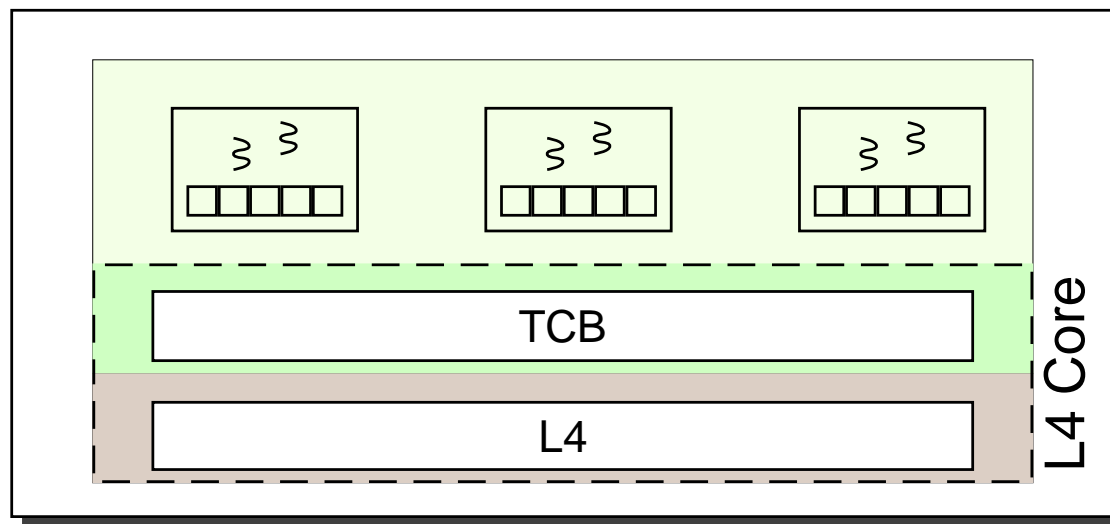
however

- invariant only describes kernel consistency
- properties about user tasks require a policy

Extension of the L4 Kernel

L4 Core

- add layer introducing policy and basic resource management
- flatten task structure to avoid uncontrollable dependencies between tasks



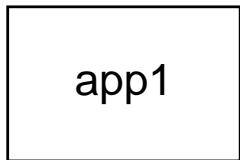
Reminder: Security Properties

- starting-point: system model fulfilling basic system properties

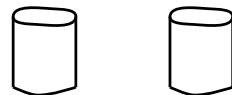
1. Application-private information must stay private unless the application explicitly allows to share it.
2. The establishment of all communication channels is controlled by the Core.

- refinement towards an L4-based multi-task system

Core Security Model - State



Applications \subseteq *APPLICATIONS*

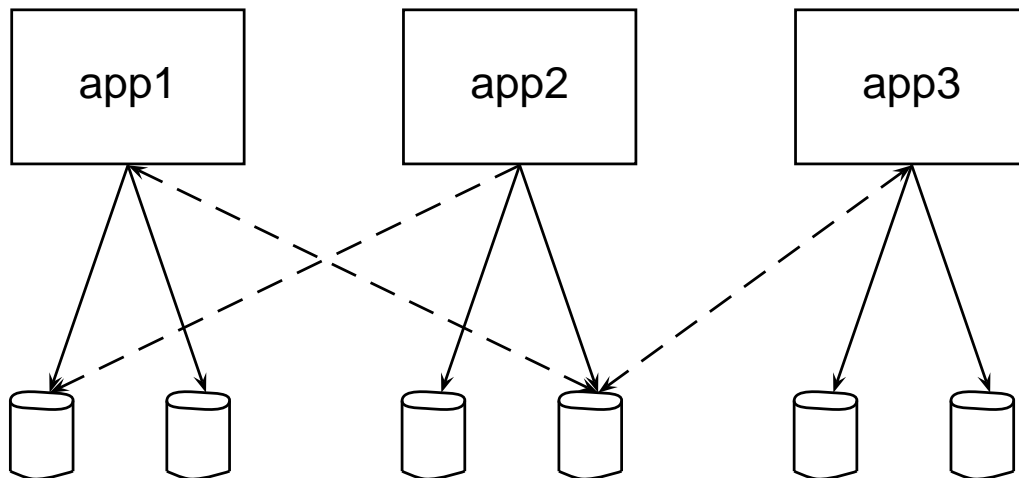


Infos \subseteq *INFORMATION*

InfoValue \in *Infos* \rightarrow *INFOVALUES*

Application-private information must stay private unless the application explicitly allows to share it.

Core Security Model - State



$Applications \subseteq APPLICATIONS$

$InfoOwner \in Infos \rightarrow Applications$

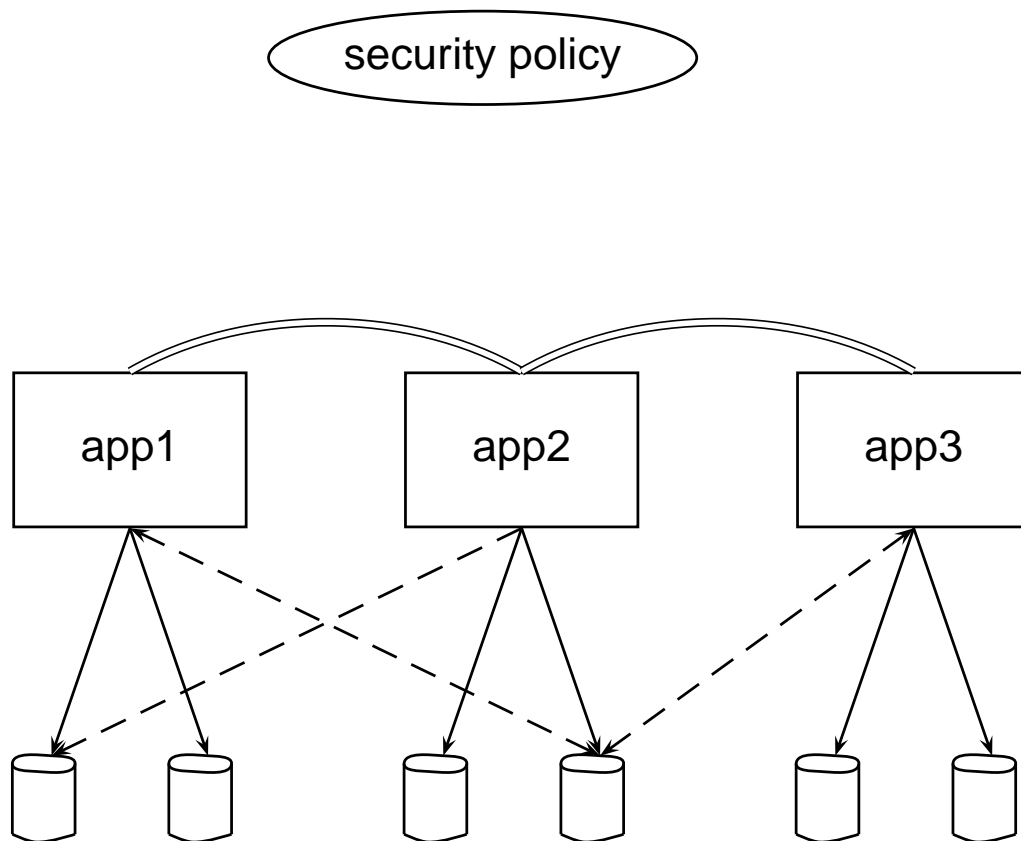
$InfoAccessRights \in Infos \leftrightarrow Applications$

$Infos \subseteq INFORMATION$

$InfoValue \in Infos \rightarrow INFOVALUES$

Application-private information must stay private unless the application explicitly allows to share it.

Core Security Model - State



$SecurityPolicyCommunication \in APPLICATIONS \leftrightarrow APPLICATIONS$

$Communication \in Applications \leftrightarrow Applications$

$Applications \subseteq APPLICATIONS$

$InfoOwner \in Infos \rightarrow Applications$

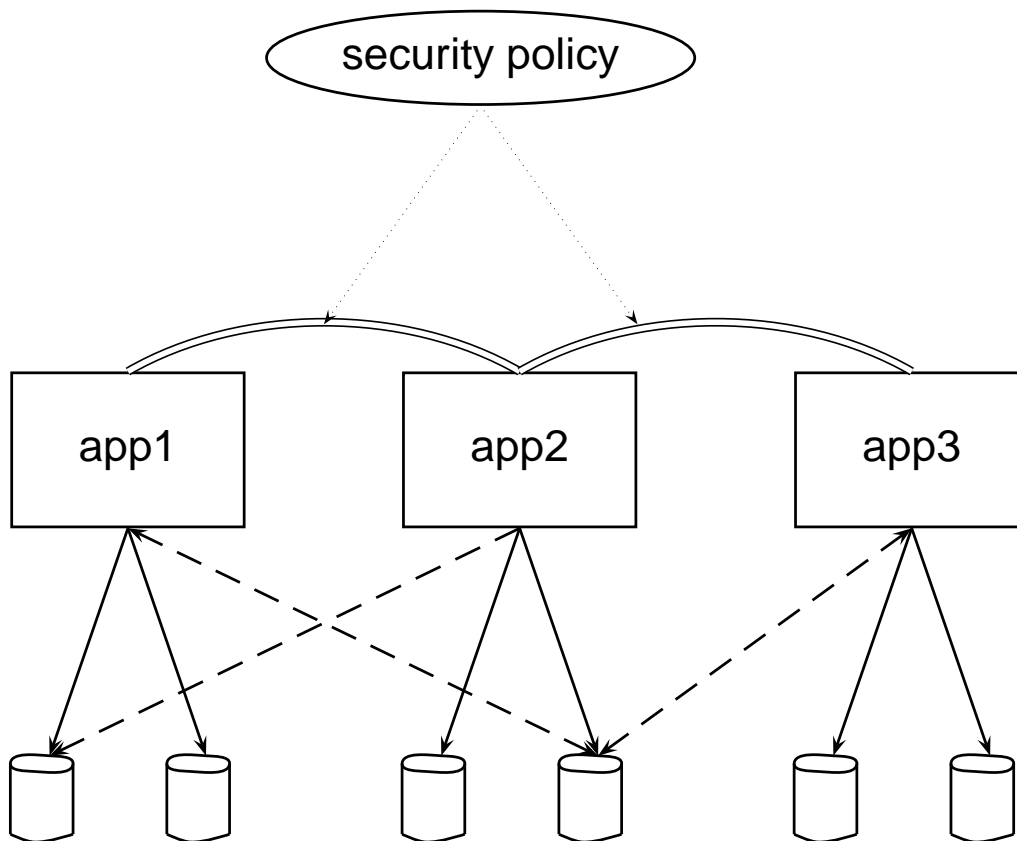
$InfoAccessRights \in Infos \leftrightarrow Applications$

$Infos \subseteq INFORMATION$

$InfoValue \in Infos \rightarrow INFOVALUES$

The establishment of all communication channels is controlled by the Core.

Core Security Model - State



$SecurityPolicyCommunication \in$
 $APPLICATIONS \leftrightarrow APPLICATIONS$

$Communication \subseteq$
 $SecurityPolicyCommunication$

$Communication \in$
 $Applications \leftrightarrow Applications$

$Applications \subseteq APPLICATIONS$

$InfoOwner \in Infos \rightarrow Applications$

$InfoAccessRights \in Infos \leftrightarrow Applications$

$Infos \subseteq INFORMATION$

$InfoValue \in Infos \rightarrow INFOVALUES$

The establishment of all communication channels is controlled by the Core.

Core Security Model - Events

- modify security policy
 - load/terminate application
 - create/destroy communication channel
 - create/destroy "information containers"
 - read/write information
 - transfer/copy information
-
- guards: maintain non-static properties
 - state transitions: maintain invariants

Creating Communication

- communication channel can be created explicitly
- *or* through common access to information

$\forall \text{app1,app2} \in \text{Application} :$

$(\exists \text{info} \in \text{Infos} :$

$\{\text{app1,app2}\} \subseteq \text{InfoAccessRights}(\text{info})) \Rightarrow (\text{app1,app2}) \in \text{Communication}$

- thus, createCommunication does
 - add a set of Communications
 - add a (possibly empty) set of InfoAccessRights

Example: Refinement Memory(I)

1. Refinement: physical memory

- physical memory is the only information container

$\text{MemPhys} \in \text{ADDRPHYS} \rightarrow \text{VALS}$

$\text{MemPhysOwner} \in \text{dom}(\text{MemPhys}) \rightarrow \text{Applications}$

$\text{MemPhysRights} \in \text{dom}(\text{MemPhys}) \leftrightarrow \text{Applications}$

- refine creation of information

createInfo =

ANY app, addphys

WHERE

app \in Applications \wedge

addphys $\neq \emptyset \wedge$ addphys \cap dom(MemPhys) = \emptyset

THEN

MemPhys := MemPhys \Leftarrow (addphys \times {VALSINIT}) ||

MemPhysOwner := MemPhysOwner \Leftarrow (addphys \times {app})

END

Example: Refinement Memory(II)

2. Refinement: virtual memory

- adds virtual address spaces and address translation

$\text{MemVirtS} \subseteq \text{MEMVIRTS}$

$\text{MemVirtOwner} \in \text{MemVirtS} \rightarrow \text{Applications}$

$\text{Mapping} \in \text{MemVirts} \rightarrow (\text{ADDRVirt} \rightarrow \text{dom}(\text{MemPhys}))$

- new function: mapping

Map =

ANY app, memvirt, newmapping

WHERE

$[\dots] \wedge \text{ran}(\text{newmapping}) \times \text{app} \subseteq \text{PhysAccessRights}$

THEN

$\text{Mapping} := \text{Mapping} \triangleleft \{ \text{memvirt} \mapsto (\text{Mapping}(\text{memvirt}) \triangleleft \text{newmapping}) \}$

END

Example: Refinement Memory(III)

3. Refinement: Dataspaces and Handles

- **dataspace: block of memory, accessed via handle**

$\text{MemHandle} \subseteq \text{MEMHANDLE}$

$\text{MemPhysHandle} \in \text{dom}(\text{MemPhys}) \twoheadrightarrow \text{MemHandle}$

$\text{MemHandleOwner} \in \text{MemHandle} \rightarrow \text{Application}$

- **creating a communication channel through sharing**

`shareDataspace` **ref** `createCommunication` =

ANY `app1`, `app2`, `handle`

WHERE

$\text{handle} \in \text{MemHandleOwner}^{-1}(\text{app1}) \wedge$

$\text{Mappable}(\text{handle}) \times \{\text{app2}\} \subseteq \text{SecurityPolicyCommunication} \wedge [\dots]$

THEN

$\text{Communication} := \text{Communication} \cup \{\text{app2}\} \times \text{Mappable}(\text{handle})$

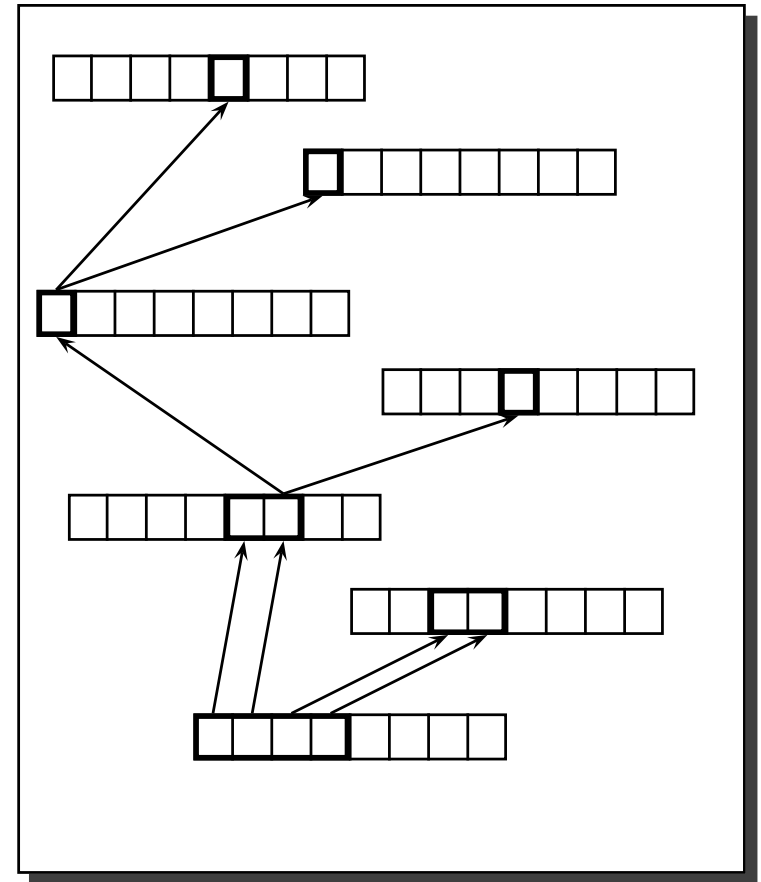
$\cup \text{Mappable}(\text{handle}) \times \{\text{app2}\}$

$\text{MemPhysAccess} := \text{MemPhysAccess} \triangleleft (\text{MemPhysHandle}^{-1}(\text{handle}) \times \{\text{app2}\})$

END

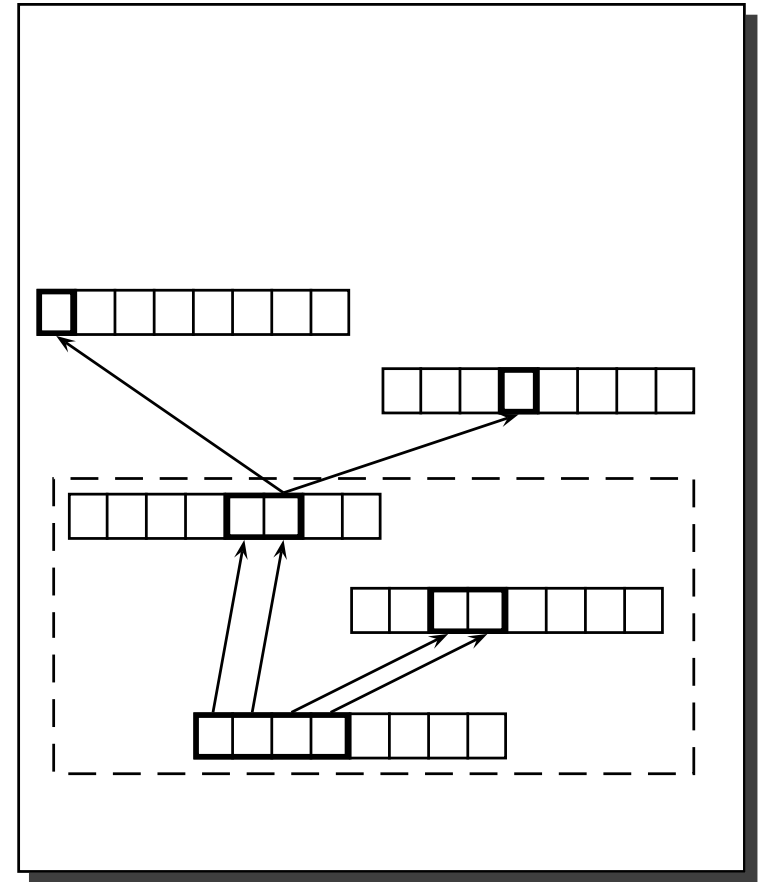
Kernel Mapping vs. Core Memory

- Core uses restricted Kernel model



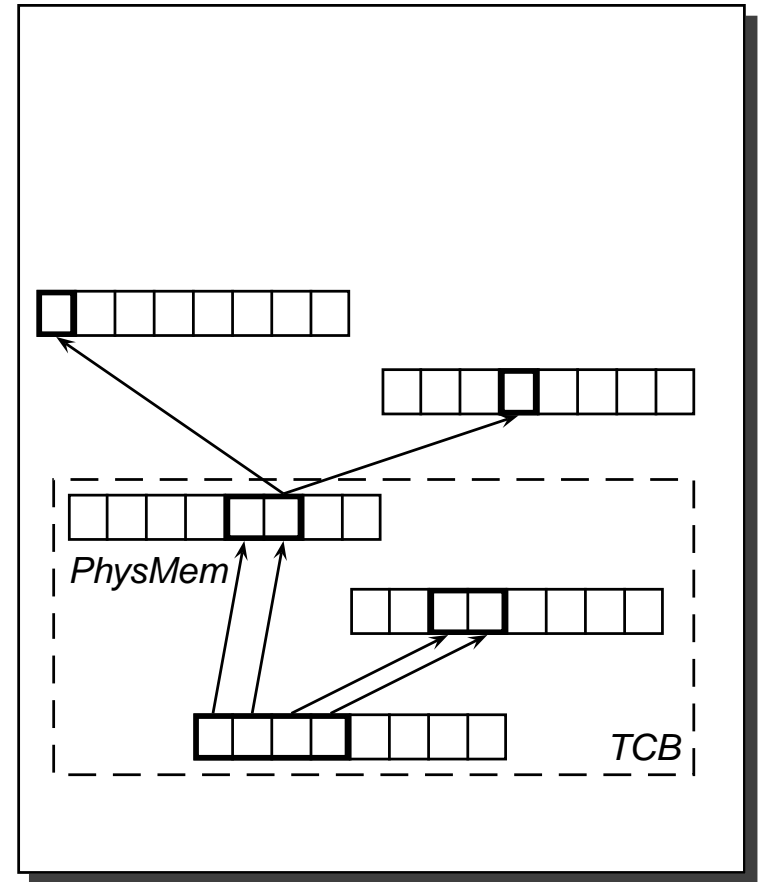
Kernel Mapping vs. Core Memory

- Core uses restricted Kernel model
 - disallow mapping outside TCB



Kernel Mapping vs. Core Memory

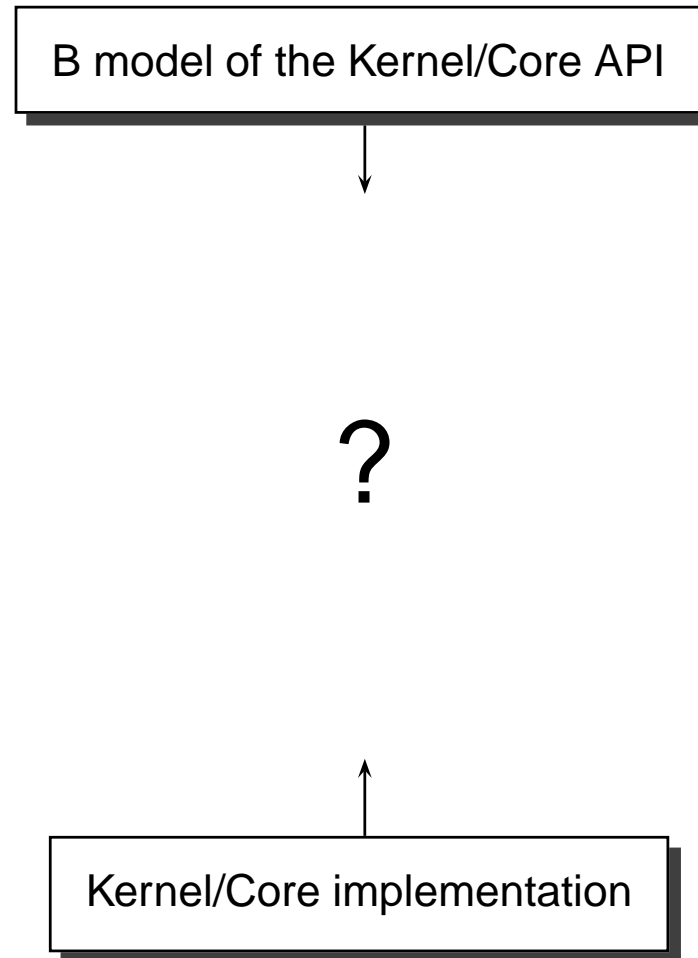
- Core uses restricted Kernel model
 - disallow mapping outside TCB
- PhysMem \approx virtual memory of memory manager
- fpages \approx dataspaces



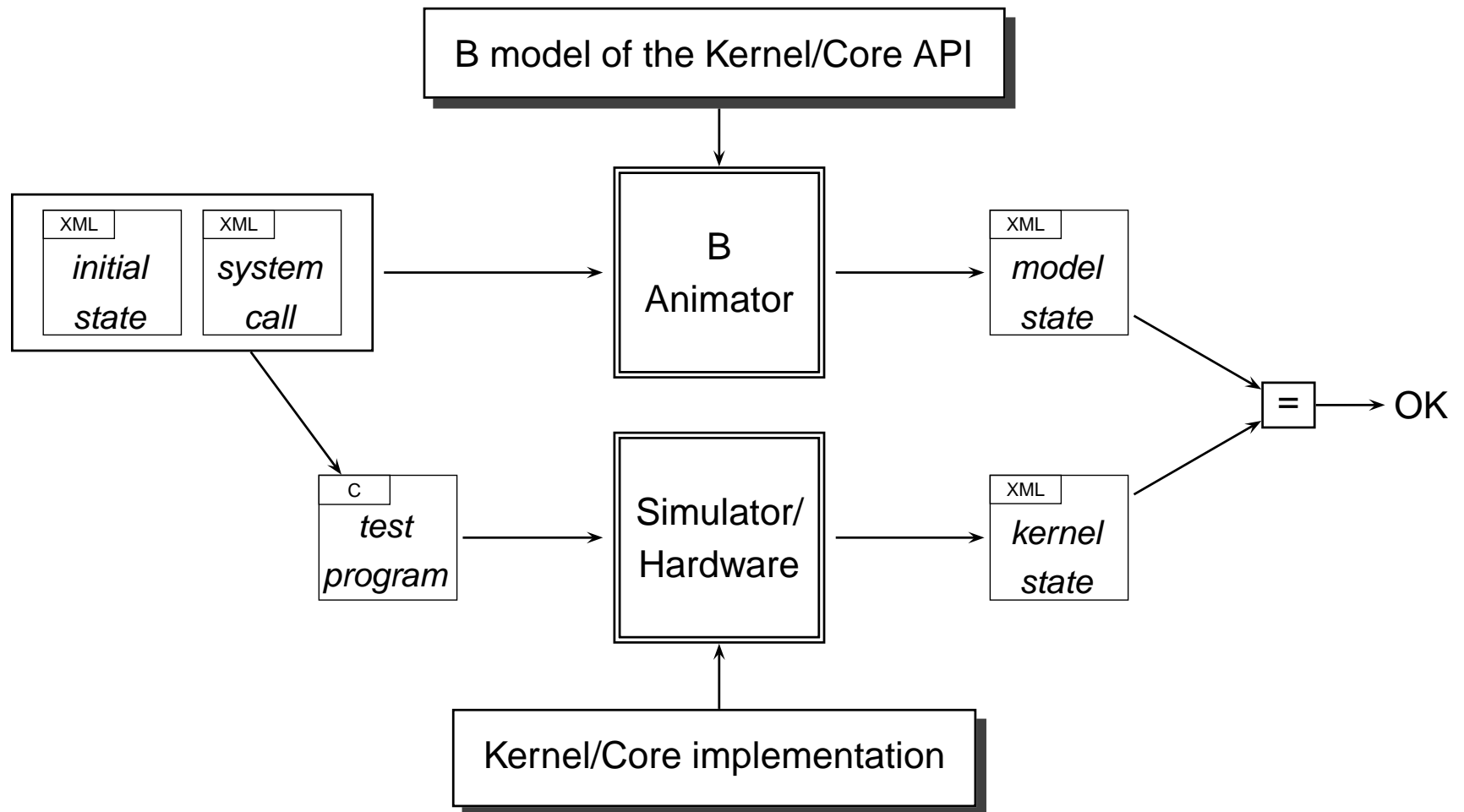
Core Model: Status

- modeled down to API level
 - 12 levels of refinement
 - 25 state variables
 - 60 events
 - final refinement: 2500 lines of code
- to be done
 - multiple servers (refinement to code level)
 - inclusion of Kernel API model

Animation of the B Model



Animation of the B Model



- API-level formalization of micro-kernel systems possible
 - verification of completeness
 - help for construction of test cases
 - allowing higher level of certification
- application of event B in complex real-life model

