



# VMM in L4/Hurd

- The concepts



## History of GNU/Hurd

- Founded at CMU in the late 1980s
  - Multi-process kernel
  - Thought as GNU's OS
  - Redeemed by Linux
- 
- Most constant GNU project, ever!



## GNU/Hurd

- Based on MACH
- Paradigm:  
*The Operating System should empower users while maintaining strict system security.*
- Capability based
- Component based



## General Information about L4/Hurd

- Started in 2002
- Identified Components:
  - Loader (laden)
  - Booter (wortel)
  - Taskserver (task)
  - VMM (phymem)
  - DDF (fabrica/deva)



## VMM Goals

- Less centralitization
- Eviction Policy
  - Flexible
  - Under application control
- Efficient
  - Avoid copying
- Accounting
  - Client should provide resources to server



## Learning from MACH

- Reduced knowlege
  - Subsystems moved into servers
- Unable to make smart paging decisions
  - Page fault statistics
  - Access pattern detecion

Too much mechanism in kernel !!



## Conclusion for L4/Hurd

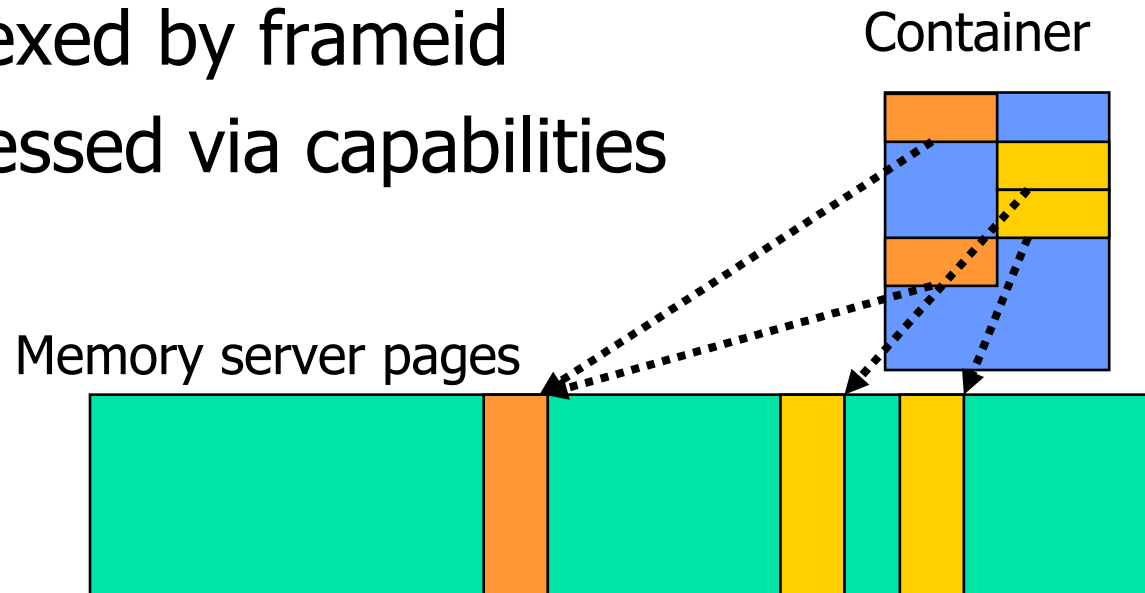
- Self Paging of Applications
  - Multiplexing
- Different frametypes
  - Guaranteed frames (gf)
    - Medium term agreement
    - $\Sigma gf < \text{Available frames in Box}$
  - Extra frames (ef)
    - Short term contract
    - Must returned immediately



## Conclusion (2)

### ■ Use Containers

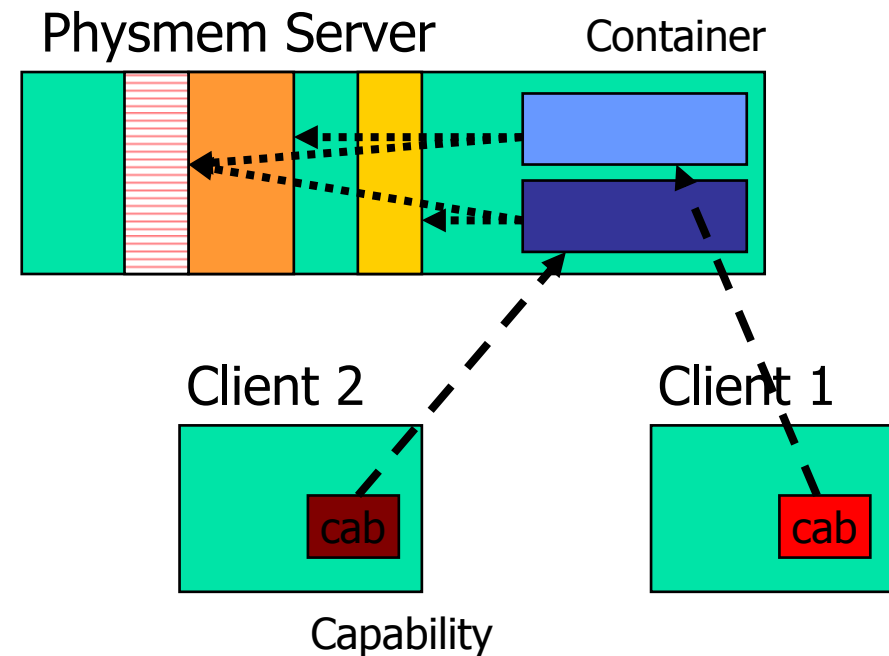
- Logical object held by memory server
- Contain existent frames or logical frames
- Indexed by frameid
- Accessed via capabilities





## Design

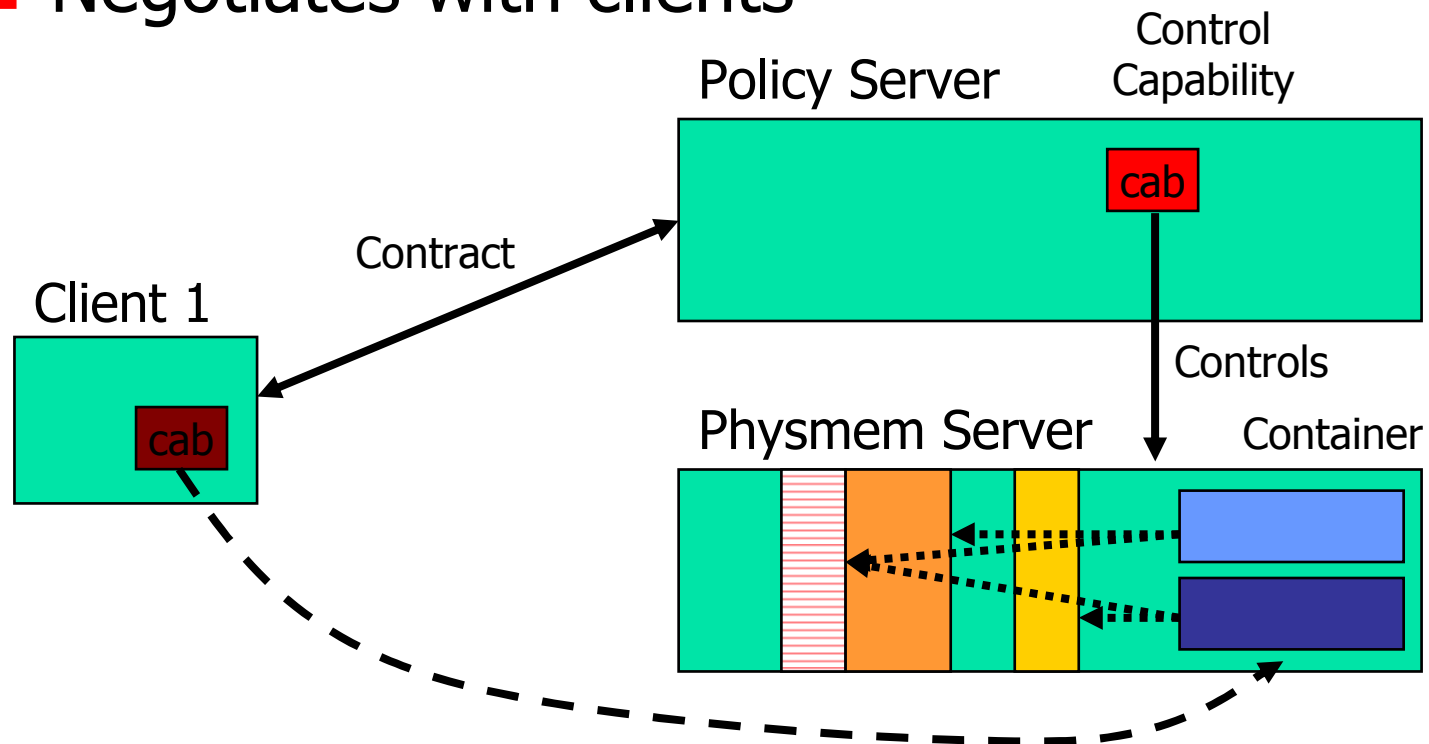
- Containers managed by a central server (physmem)
- Manages Containers
- Forces Limitation
  - Guaranteed/ Extra Frames





## Policy of Guaranteed and Extra Frames

- Imposed by the policy server
  - Holds control capability of phymem
  - Negotiates with clients





## ADT Container: Operations (1)

- Create (out container)
  - Creates a new Container
- Destroy (in container)
  - Destroys a container and release its internal datastructures

### Shortcuts

- Create\_With (out container, in count, out weakref)
  - Create with *count* allocated frames
- Create\_From (out container, in src\_cont, in start, in count, out weakref)
  - Create new container with *src\_cont* frames from *src\_cont*



## ADT Container: Operations (1)

- Allocate (in container, in src, in start, in cnt, in flags)
  - Allocate *cnt* frames in a container beginning at frameid *start*
- Deallocate (in container, in start, in cnt, in flags)
  - Deallocate *cnt* frames beginning at frameid *start*



## ADT Container: Operations (2)

- Map (in container, in start, in cnt, in flags)
  - Map *cnt* frames (beginning at *start*) of a container into the senders space
- Grant\_Access (in container, in task, out fullref)
  - Delegate capability of a container to other task
- Share (in container, in task, out weakref)
  - Create a remote capability for other task



## ADT Container: Operations (3)

- Lock\_Map (in container, in unmap)
  - Pin/Lock frames of that *container*
  - Unmap them in all other tasks
- Lock\_Physical (in container, in unmap, out pframes[])
  - Pin/Lock frames of that *container*
  - Return their addresses
  - Unmap them in all other tasks
- Unlock (in container)



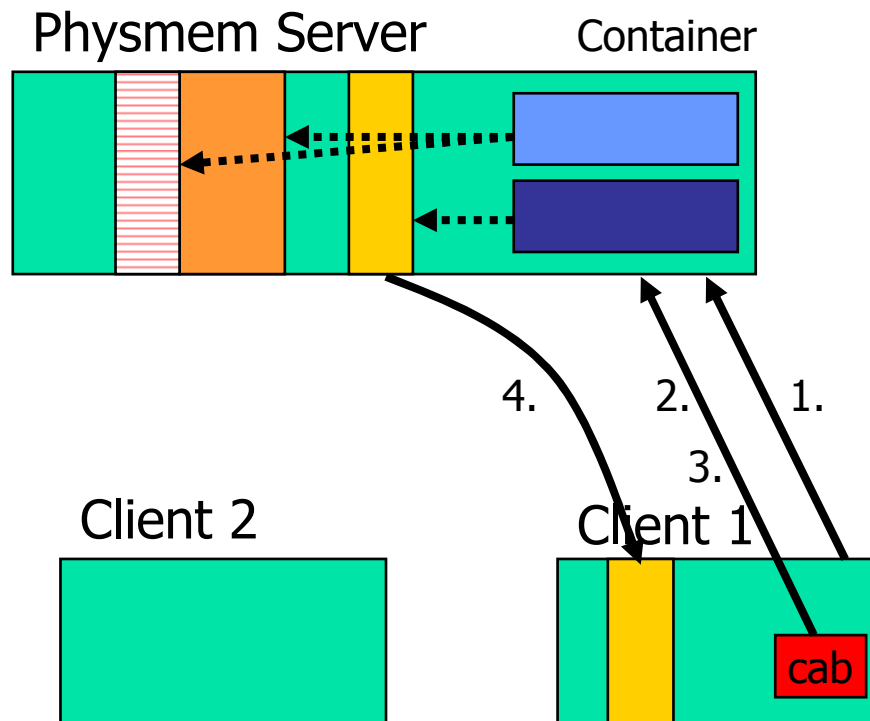
## ADT Container: Operations (4)

- Copy (in src\_container, in src\_start, in dest\_container, in dest\_start, in cnt, out error\_frame)
  - Copies logically from a relative position in one container to another
- Copy\_Gather (in container, in srcframes[], in dest\_container, in dest\_start, out error\_frame)
  - Copies logically data from arbitrary frames of the source container to the destination container



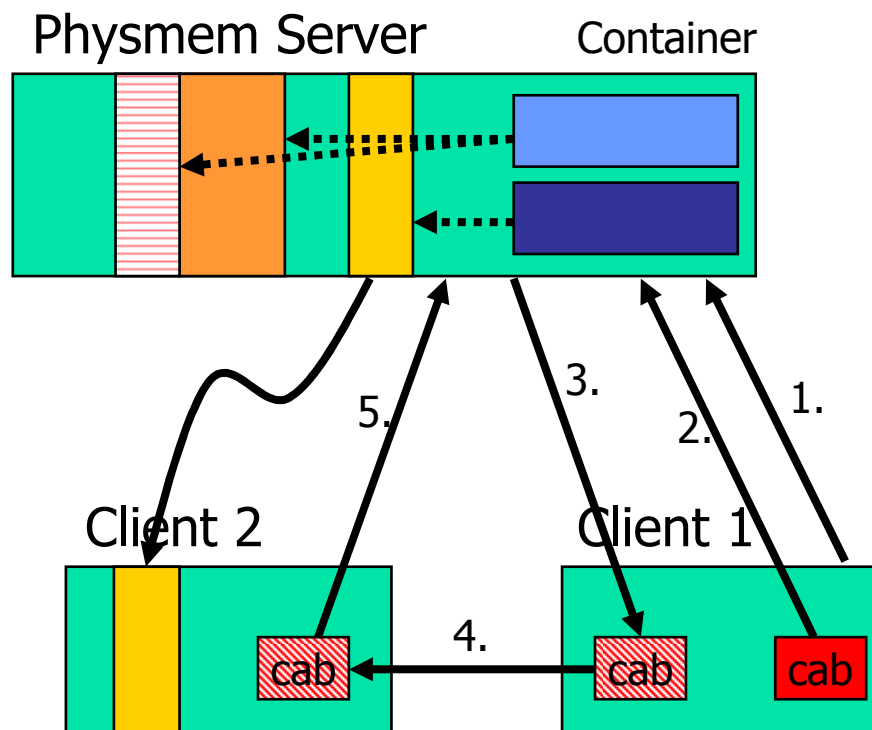
## Example: Mapping a Page

1. Client creates Container
2. Client allocates frames
3. Client requests mapping
4. Physmem returns mapping





## Example: Sharing a Container

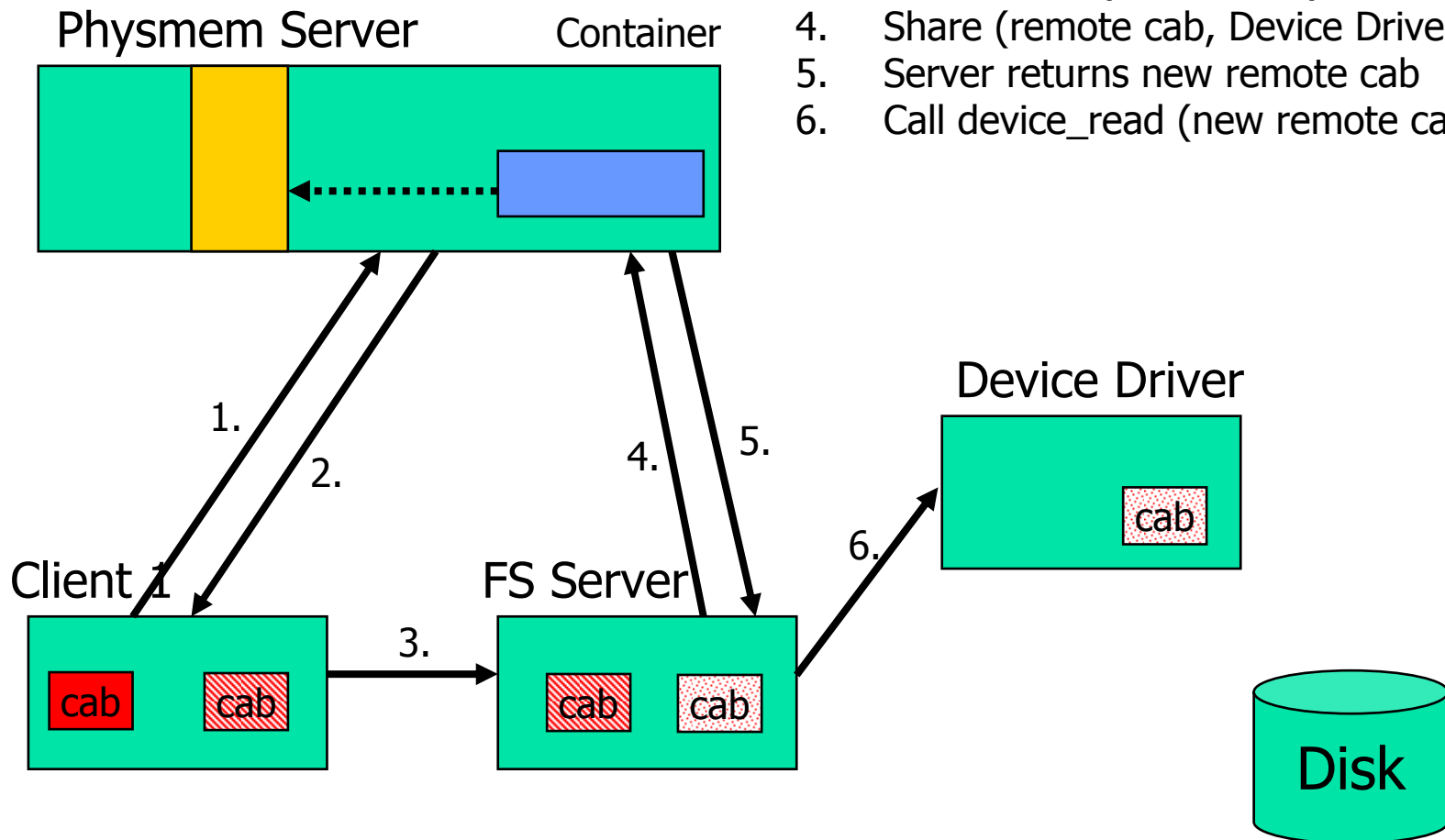


1. Client 1 creates Container
2. Client 1 requests Share
3. Phymem returns weak reference
4. Client 1 transfers reference to Client 2
5. Client 2 gets memory mapped



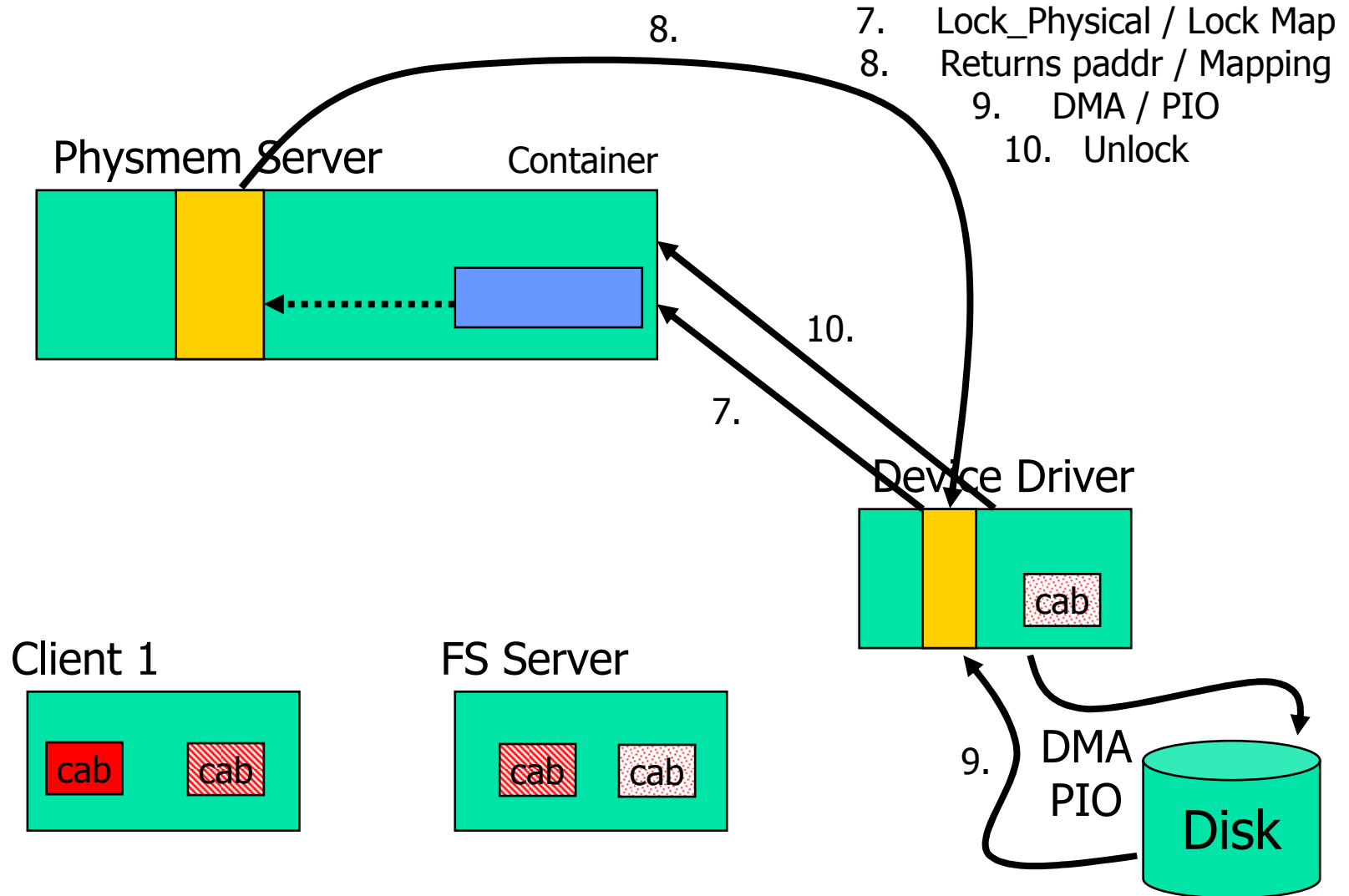
## Example: Reading from a file (1)

1. Creates a new container with frames
2. Server returns container & remote cab
3. Call fs\_read (remote cab)
4. Share (remote cab, Device Driver)
5. Server returns new remote cab
6. Call device\_read (new remote cab)



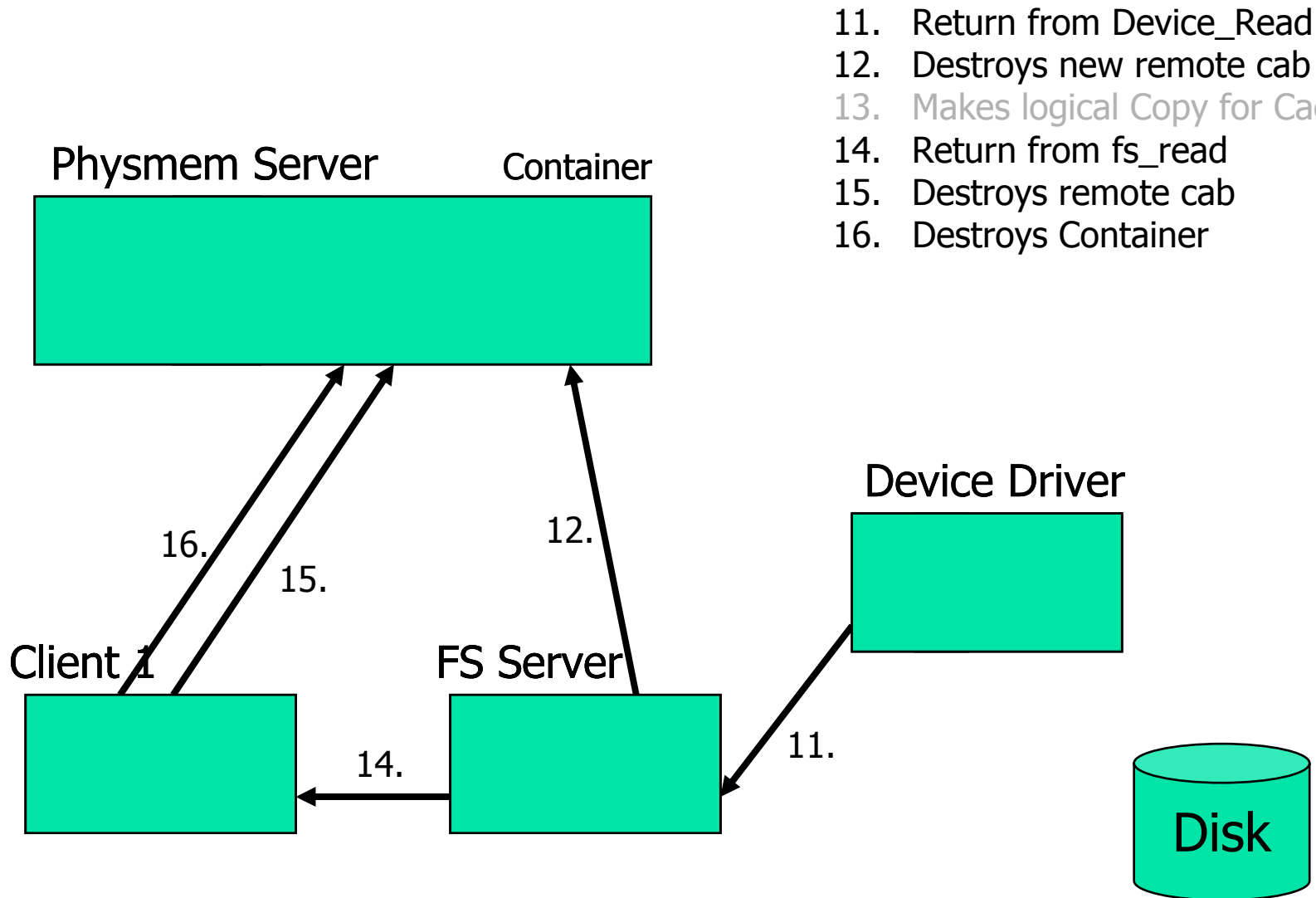


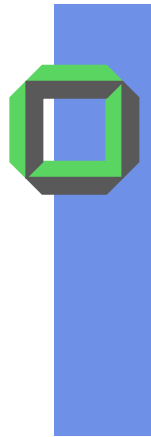
# Example: Reading from a file (2)





## Example: Reading from a file (3)





Neal says:

Code needs testing  
Code needs implementation



## References

- Porting the GNU Hurd to the L4 Microkernel, Markus Brinkmann (Aug 2003)
- Virtual Memory Management, A new Approach for the Hurd on the L4 Microkernel, Neal Walfield (Jul 2002, <http://web.walfield.org/>)