# Project 7 Discussion

# FileManager . Init

**Initialize FCB table**
**and FCB free list**

**Initialize OpenFile table**
**and OpenFile free list**

**Allocate a frame**

**Reads sector zero into that frame**

# FileManager . Open

**Call “findFCB”**

**Consult the directory (in memory)**
**Allocate a new FCB (or return existing FCB)**
**May wait, if necessary**

**Allocate an OpenFile**

**May wait, if necessary**

**Initialize the OpenFile object**

**openFile.fcb = ...**
**openFile.currentPos = 0**
**openFile.numberOfUsers = 1**

# Handle_Sys_Open

*First, deal with arguments.*

*Next, locate an empty slot in the fileDescriptors table.*

*No need to acqurie lock, since we assume that this process has only one thread, this thread is the current thread, and that only the process's thread can modify the fileDescriptors array for this PCB.*

*Set "i" to the free entry.*

```
for i = 0 to ...
    if pcb.fileDescriptor [i] == null
        ...
```

*If there are no free entries, return -1.*

*(continued)*

# Handle_Sys_Open

*Open the file. This may block.*

**open = fileManager.Open (filename)**

*If this fails, return -1.*

**pcb.fileDescriptor [i] = open**

**return i**

# Handle_Sys_Read

```
-- Check the file descriptor...
if fileDesc < 0 || fileDesc >= MAX_FILES_PER_PROCESS
    return -1

-- Get the OpenFile...
open = currentThread.myProcess.fileDescriptor [fileDesc]
if null, then return -1

-- Check for a bad argument...
If sizeInBytes == zero, return ...
If sizeInBytes < zero, return ...
```

*(continued)*

# Handle_Sys_Read

**Acquire “fileManagerLock”.**
**Using the current size of the file and current position...**
**Figure out exactly which sectors we are going to read**
**and check the arguments.**
**If any problems...**
**(Nothing is read)**
**Release “fileManagerLock”**
**Return -1**
**If everything is OK, continue...**
**Update the new current position.**
**Release “fileManagerLock”.**
**Do the reading.**
**Invoke fileManager.SynchRead to do the read**

# Handle_Sys_Read

**Must move data into virtual address space.**

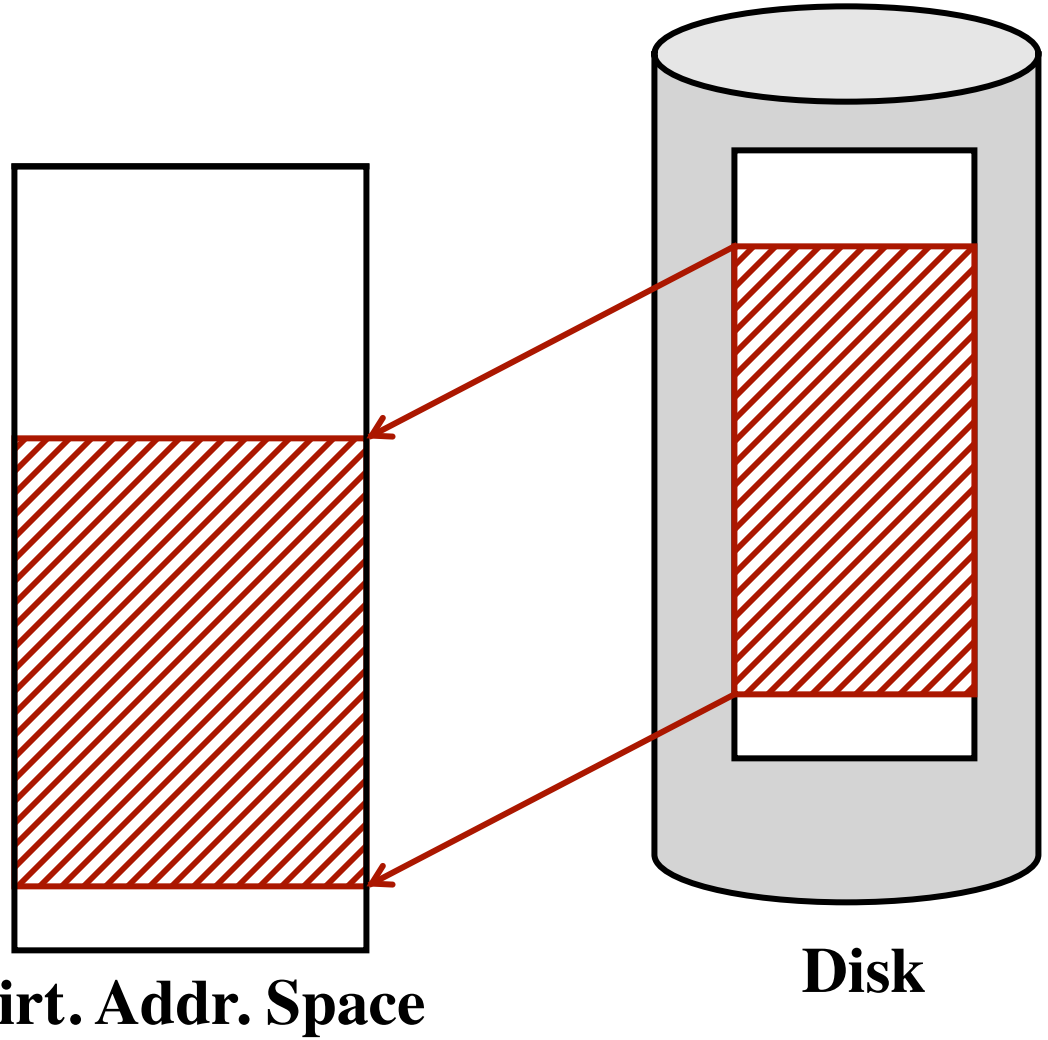
**The target area may be spread over several pages.**
**But frames are not contiguous!**

**Must divide the reading into several read operations.**

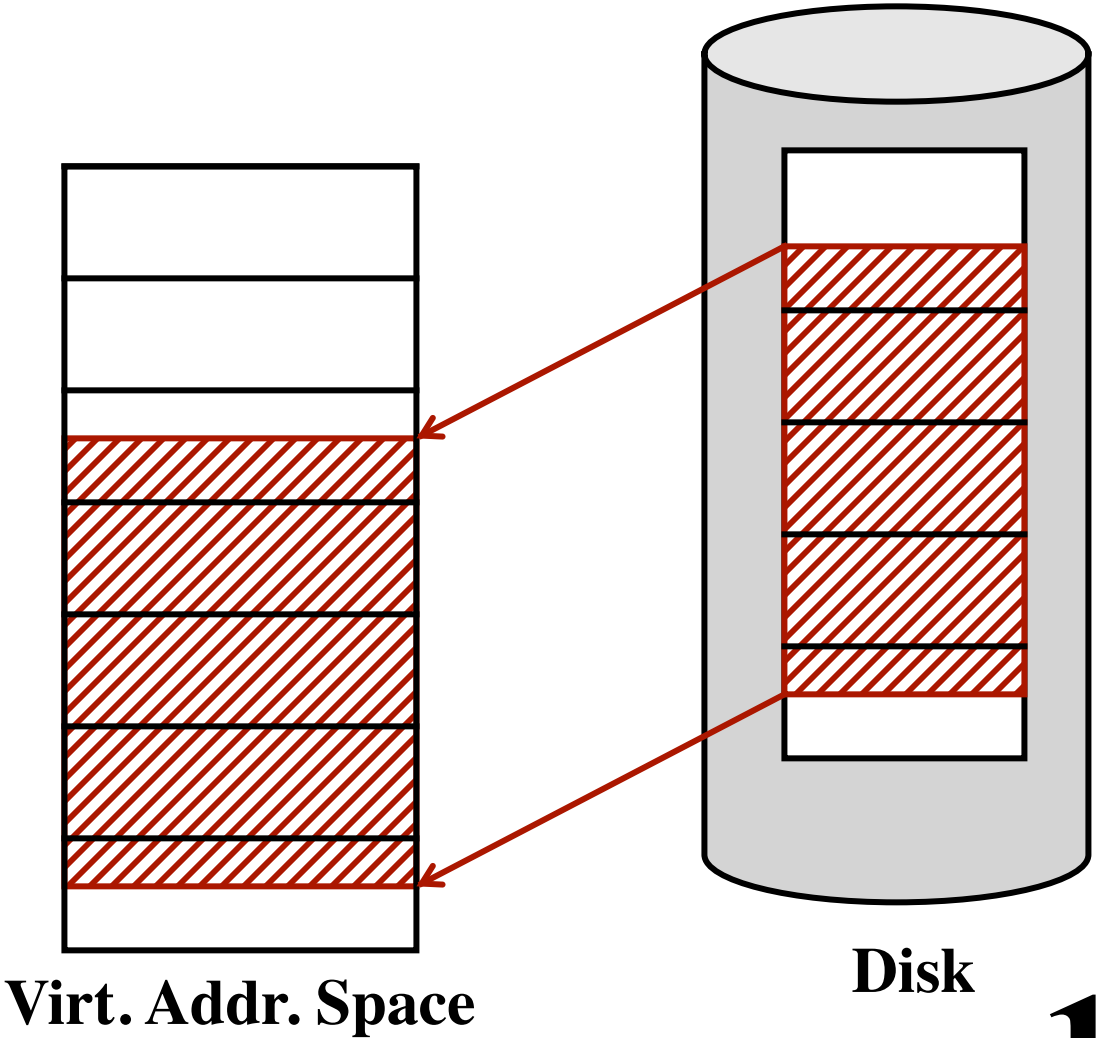
**Identify "chunks"**
**Each chunk is a separate call to fileManager.SynchRead**

***And don't forget to mark each frame as "dirty"!***

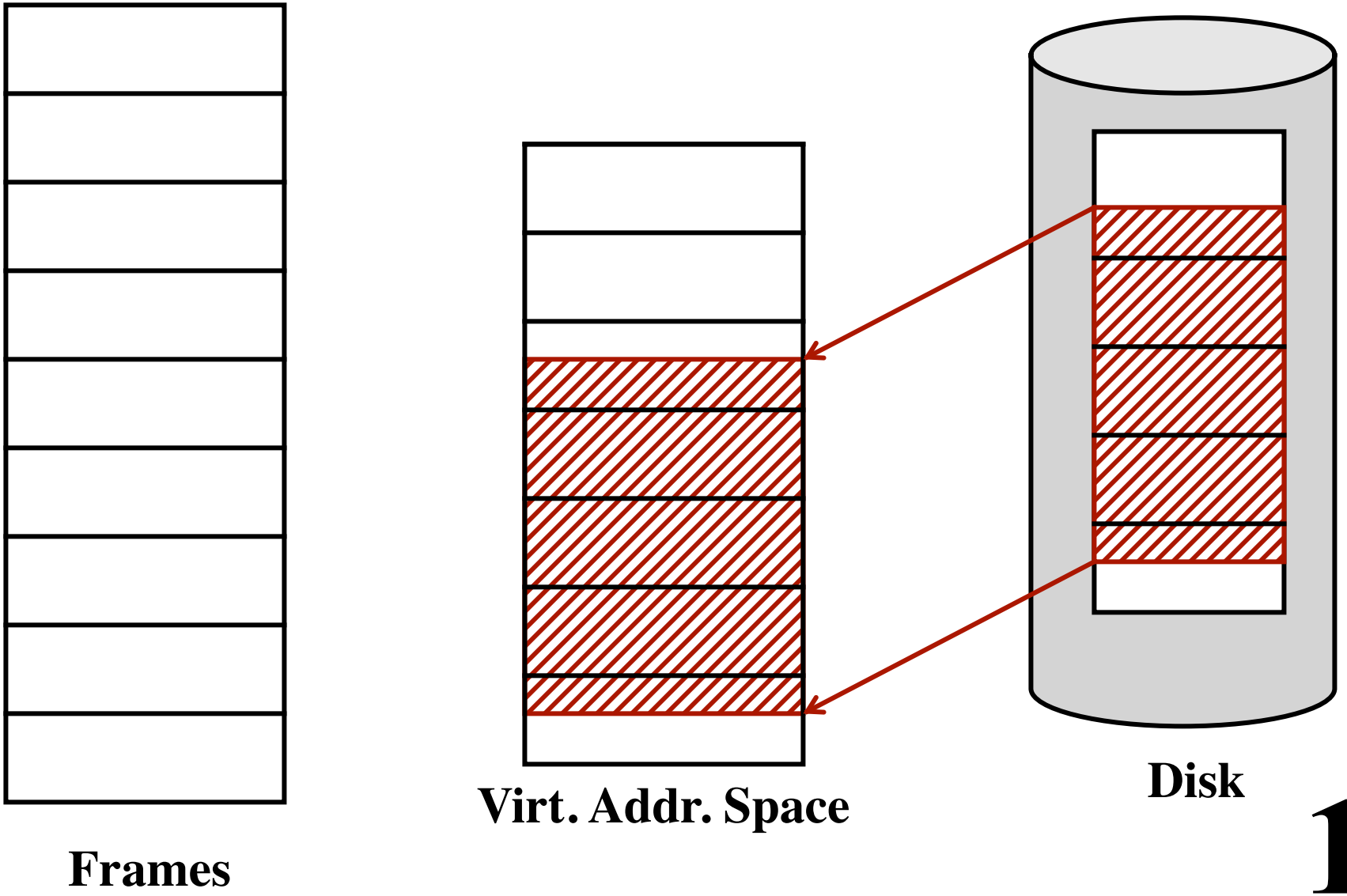
# Handle_Sys_Read



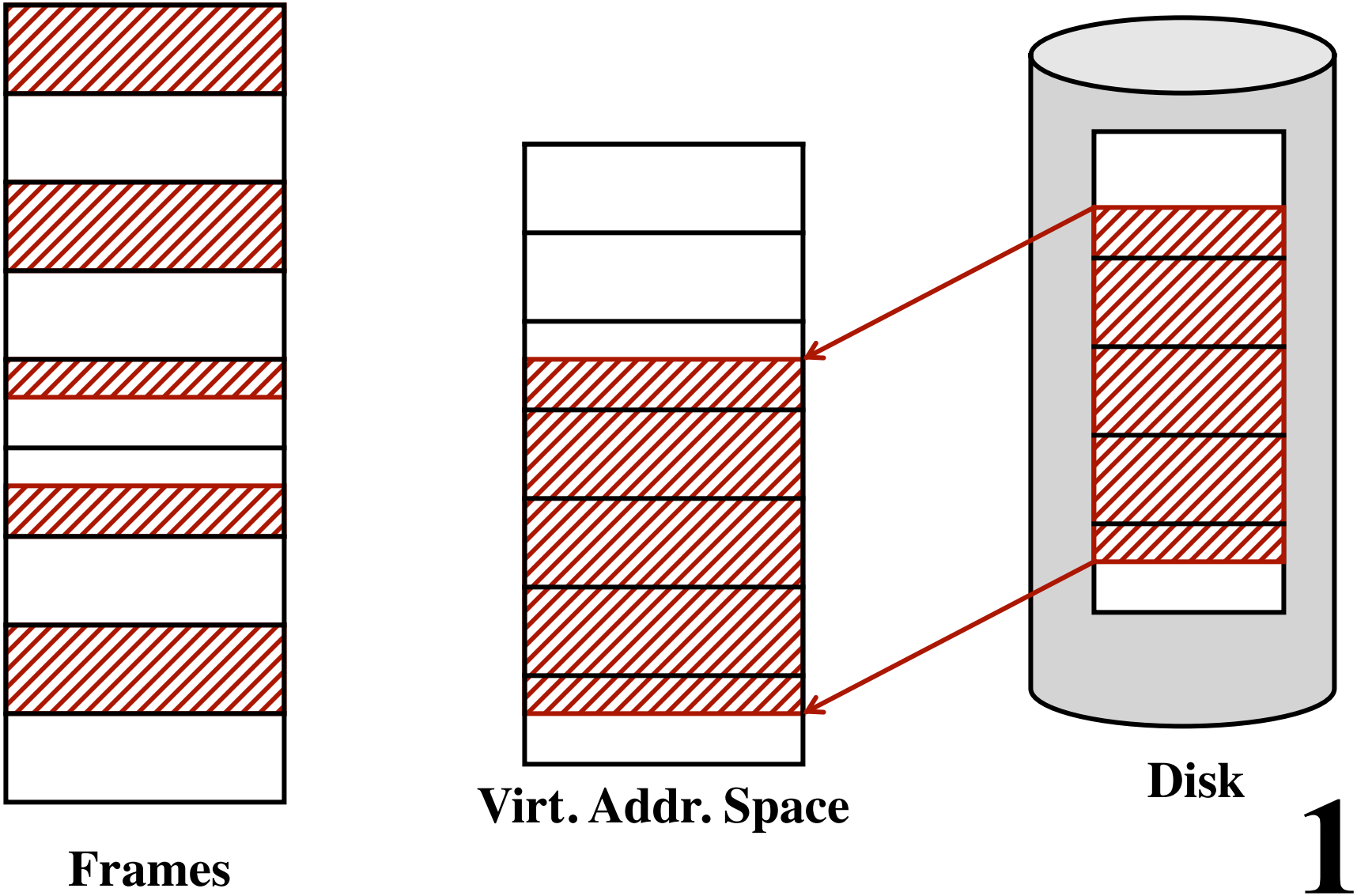
# Handle_Sys_Read



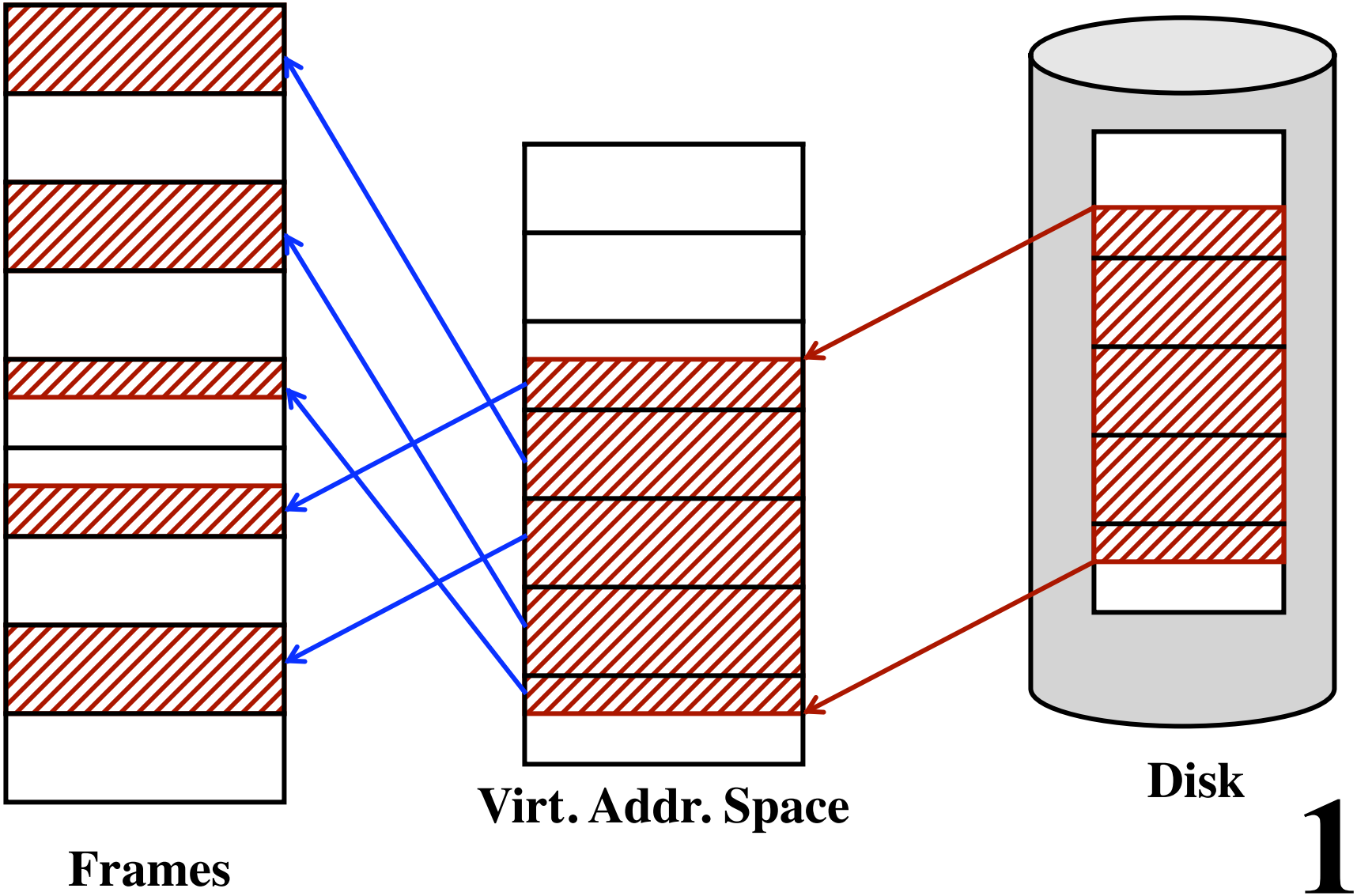
# Handle_Sys_Read



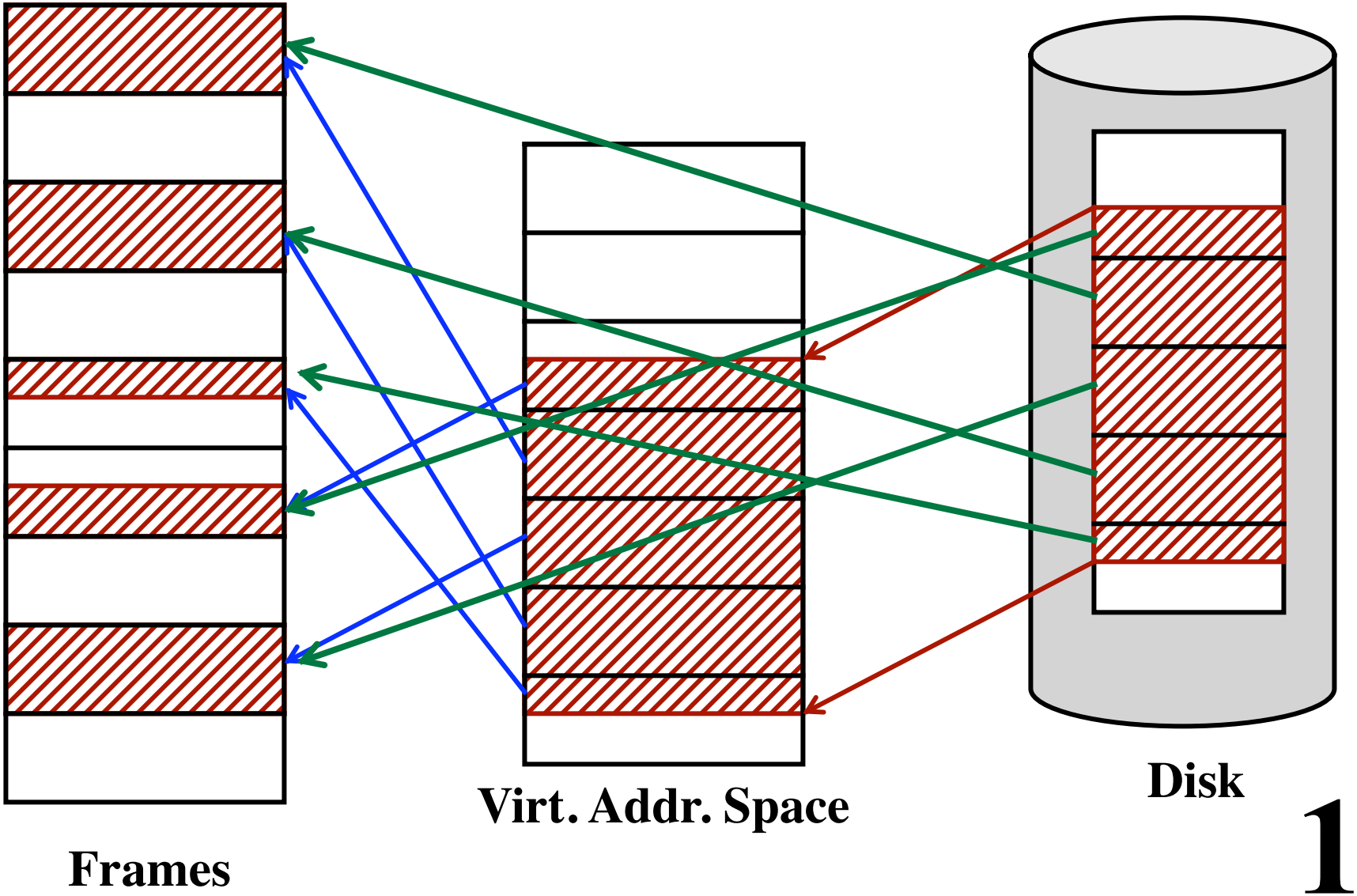
# Handle_Sys_Read



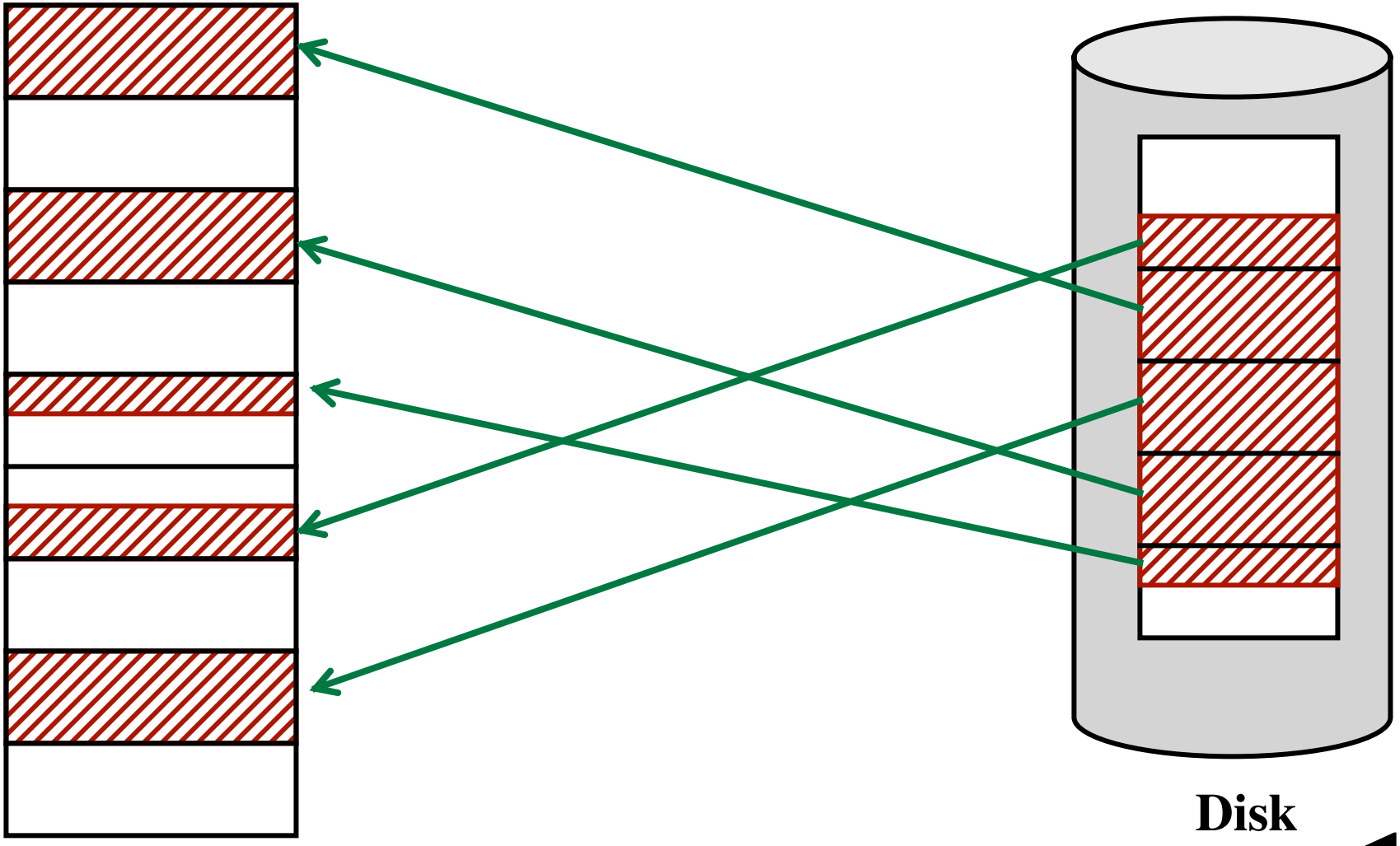
# Handle_Sys_Read



# Handle_Sys_Read



# Handle_Sys_Read



# Handle_Sys_Read

***Problem:***

**Must divide the reading into several "chunks"**
**Some pages may be invalid or not-writable.**
**This is an error...**
**Specs say: Must return -1 *without doing any I/O***

***Approach:***

**Go through the chunks twice.**

**Pass 1:** **Compute chunk locations**
**Check for errors, and return if any**

**Pass 2:** **Re-compute all chunk locations**
**Perform the reading**
**Update dirty bits**

# Handle_Sys_Seek

```
Acquire FileManagerLock

-- Check the fileDescriptor.  Is it in the array?
-- Does it point to an "open" OpenFile?
if problems...
    release FileManagerLock
    return -1
endIf

-- Check for bad argument
if newCurrentPos < -1 or greater than file size...
    release FileManagerLock
    return -1
endIf
```

*(continued)*

# Handle_Sys_Seek

```
-- If newCurrentPos is -1, set newCurrentPos to
-- current size of file...
 if newCurrentPos == -1
    newCurrentPos = open.fcb.sizeOfFileInBytes
endIf

-- Set the currentPos...
... = newCurrentPos

-- Release lock and return it...
Release FileManagerLock
return ...
```

# FileManager.Close

**method Close (open: ptr to OpenFile)**

***Write out the buffer, if dirty.***

***The "numberOfUsers" for the OpenFile is decremented and, if zero, the OpenFile is freed.***

***If the OpenFile is freed, then the "numberOfUsers" for the FCB is decremented. If it too is zero, the FCB is freed.***

*(continued)*

# FileManager.Close

```
acquire “fileManagerLock”
fileManager.Flush (open)
fcb = open.fcb
decrement open.numberOfUsers
if it became zero...
    openFileFreeList.AddToEnd (open)
    anOpenFileBecameFree.Signal (...)
    decrement fcb.numberOfUsers
    if it became zero...
       fcbFreeList.AddToEnd (fcb)
       anFCBBecameFree.Signal (...)
release “fileManagerLock”
```

# FileManager.Flush

**method Flush (open: ptr to OpenFile)**

***Write out the buffer, if dirty.***
***Assumption: caller already holds the fileManagerLock.***

# FileManager.Flush

```
method Flush (open: ptr to OpenFile)

Write out the buffer, if dirty.
Assumption: caller already holds the fileManagerLock.

if open.fcb.bufferIsDirty
    ...bufferIsDirty = false
    diskDriver.SynchWriteSector (
        ....relativeSectorInBuffer + ...startingSectorOfFile,
        1,
        ...bufferPtr...)
endIf
```

# Handle_Sys_Close

*Check the argument*

*Is it a legal array index?*

*Does it point to an open file?*

**open = currentThread.myProcess.fileDescriptor [fileDesc]**
**currentThread.myProcess.fileDescriptor [fileDesc] = null**

*-- Make sure this file was really open...*
**if open == null...**

**fileManager.Close (open)**