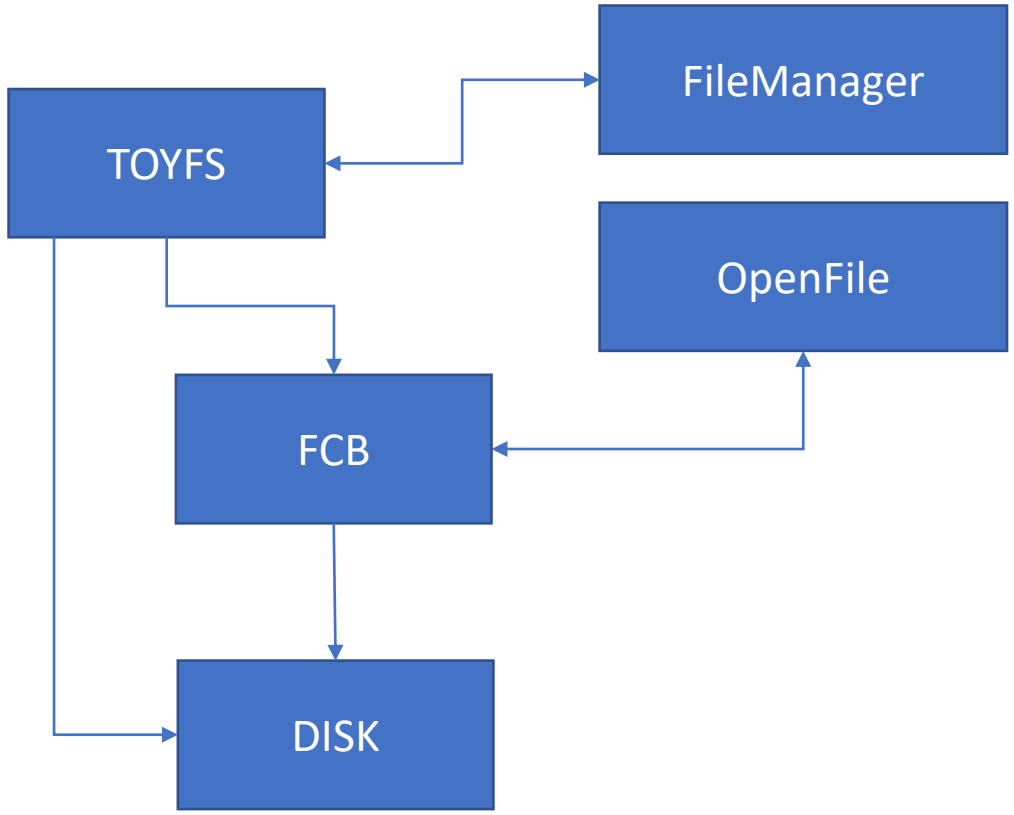
A6 Hints

ToyFS



There are other classes as well

FileManager

- Manages OpenFile and Pipe Objects
- Methods include
  - GetAnOpenFile
  - Open, Close
  - GetAPipe
  - …
- Includes data structures
  - OpenFileTable
  - openFileFreeList
  - …

FCB

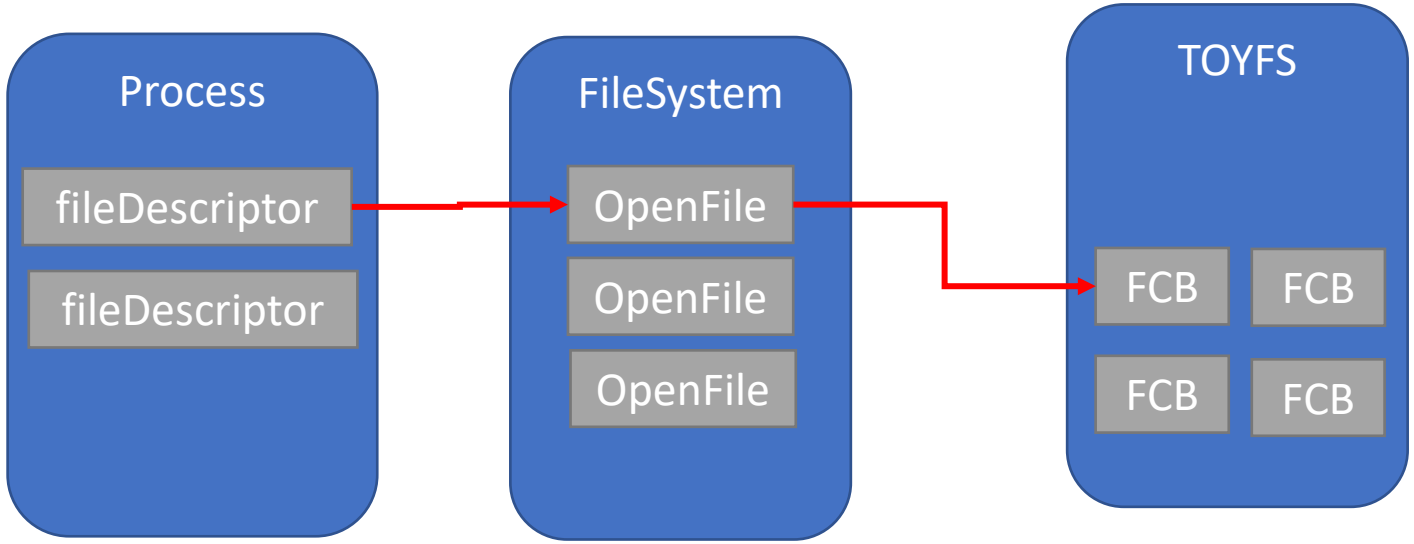
- Records all data associated with a single ToyFS file
- Has an InodeData object that can be used to get file info, like the actual sectors on disk.
- Has methods to read and write to disk

TOYFS

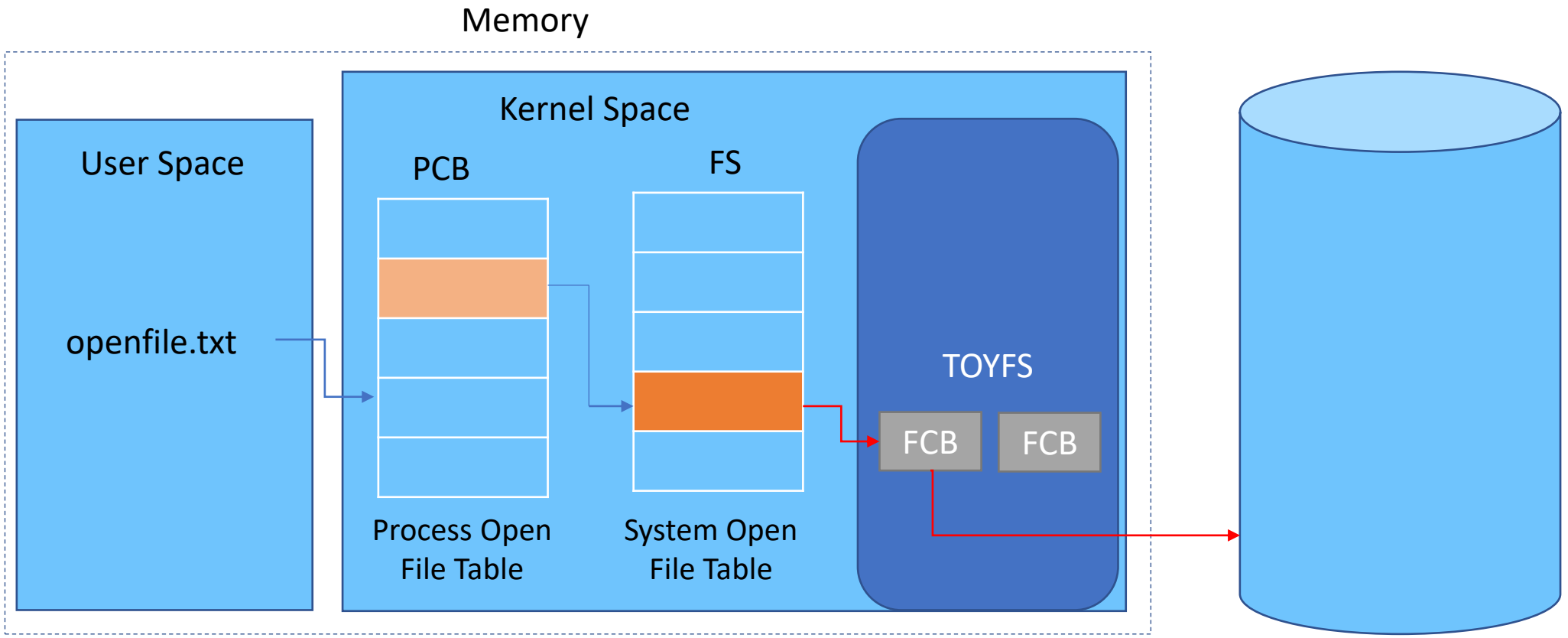
- Contains the superblock (in memory)
  - This include data and inode bitmaps
- Has a handle to root directory (root always in inode 1 on disk)
- FCB Table, + List of FREE FCB
- Can allocate/free inodes or data blocks
- Methods to look up inode give a file name or an FCB

FileSystem has limited OpenFile objects. It maintain a free list of OpenFile objects

TOYFS has limited FCBs. It maintain a free list of FCBs



In Memory File System Implementation



- A File Descriptor, is an integer. It's the index of the entry in the FileDescriptor table.

A6

- Open (30 points)
- Close (15 points)
- Read (40 points)
- Seek (20 points)
- Write (30 points)
- Stat (25 points)
- ChDir (10 points)
- Write/extend (15 points)
- Open/create (20 points extra)
- Open and Exec/permissions (10 points extra)

FileManager.Open

- Get a new fd from process using Get_Open_FD
  - Get_Open_FD: you should implement this. Simply looks up a null entry in the file descriptor table for the process
- Call FileSystem.Open to get file
- Check for file kind and permissions
- If all checks, set process file descriptor table to point to the open file at the fd index
- Return fd

ToyFS.Open

1. Use NameToInodeNum to get the inode number of the file.
2. GetFCB() to get file FCB.
3. Get an openfile handler using fileManager.GetAnOpenFile(false) for not waiting. Connect the open file to the FCB and initialize it either to a FILE or a directory using: OpenFile.Init
4. To check if it's a file type, use fcb.inode.mode & TYPE_FILE == TYPE_FILE
5. Return the openfile

Check for these methods failing or returning nulls. Set error and release the fcb before returning!

ToyFS.ReadFile

- Calculate actual size of byte so that you don’t overflow the file size.
- Use file.fcb.SynchRead to read from the disk.
  - However this is not very straight forward, as SynchRead writes into a physical address while the buffer handed as an argument is a virtual address.
- Calculate buffer pages
- For every page
  - Calculate physical address
  - Copy only up to page size using file.fcb.SynchRead and pass physical address
  - Remember, you might need to start somewhere in the middle of the frame,
  - Set current page to dirty and referenced using aaddrSpace.SetDirty and SetReferenced
- Update current position to final position reached.
- Return total number of bytes read.

ToyFS.WriteFile

- Very similar to read, but you're reading from a physical address instead of writing to it.
- Use file.fcb.SynchWrite
- What if you are increasing the size of the file?
  - Update inode file size to new size
  - Set inode to dirty and write inode back.

ToyFS.WriteFile

- Use file.fcb.SynchWrite
  - SynchWrite might call SynchRead, which is already implemented for you.
    - We can only write a "sector" to a disc, that's why partial writes need to read the sector first.
  - SynchWrite might allocate a new sector (using SynchRead)
  - AllocateNewSector is NOT implemented for you, you have to implement it.

InodeData.AllocateNewSector

- Takes as an argument the new logical sector number.
- Allocate new data block using AllocDataBlock()
  - Check the code for it.
- If sucessful, increase the number of blocks for the inode, balloc
- If the new sector number is less than 10, just use direct pointers by updating the direct pointers array using something like:
  - direct[new sector number] = new block allocated

AllocateNewSector: Indirect allocation

Is it the fist time we use indirect? (check if indirect == 0)

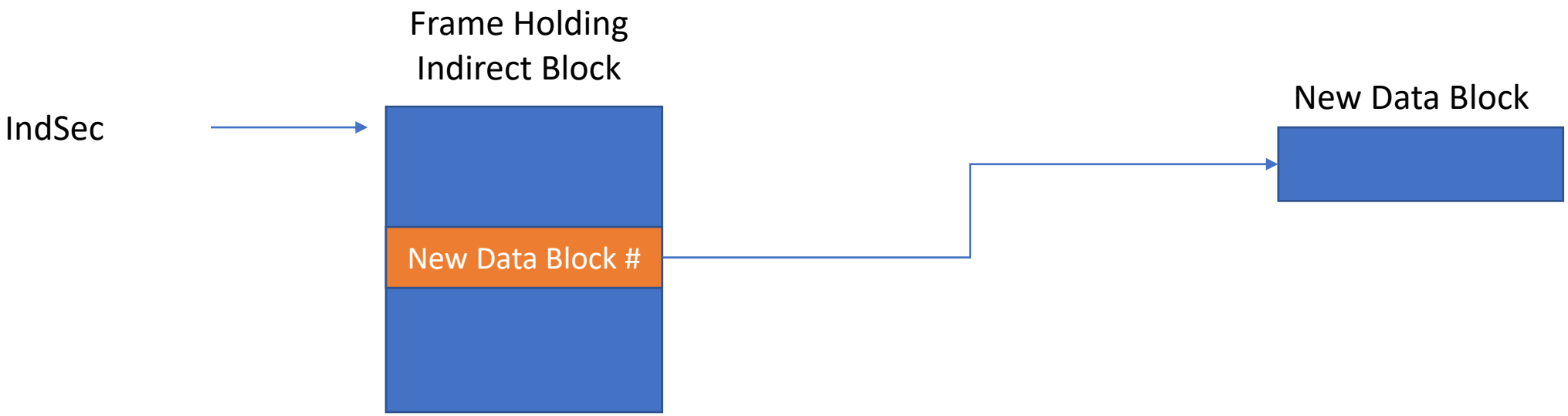
- If so we need to allocate another new block to hold the pointers. You save that block number in inode.indir1
- Now, we allocate it on disk, but where to store it in memory when we need to read it?
- There is a field in inode data, indSec, that points to that frame.
  - Get a new frame, using GetANewFrame and let indSec point to it.
  - MemoryZero the frame.

AllocateNewSector: Indirect allocation

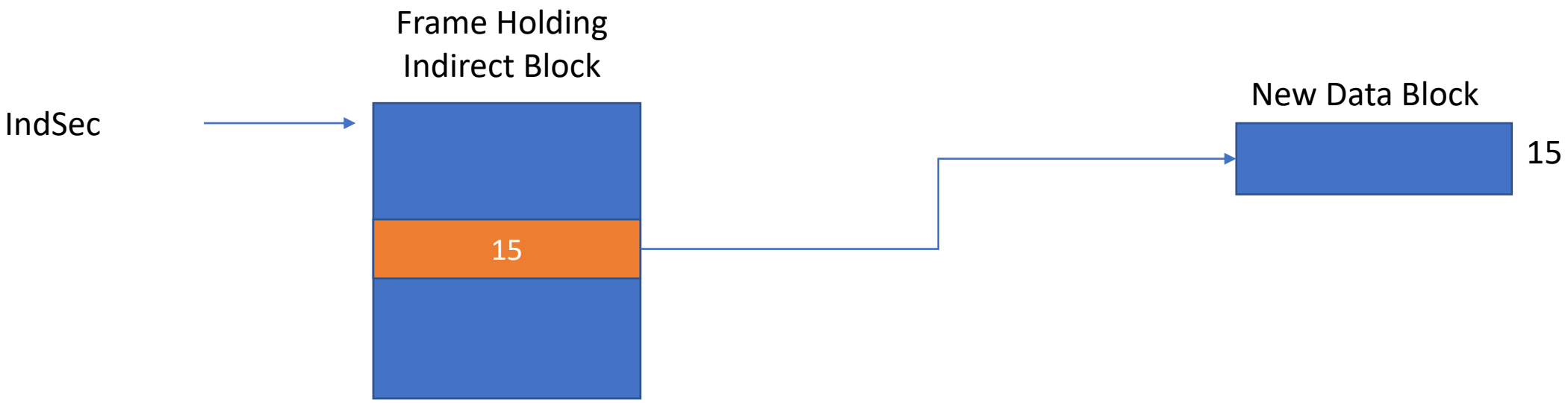
Now if indirect has already been used before:

- IndSec must be pointing to the frame that holds the indirect block infomration.
- However, this is not always the case, maybe the indirect block has not been loaded from the disk.
- How to check? Check the value of IndSec, if it's <= 0 that means we need to load the block.
- This is easily done using self.GetIndirect

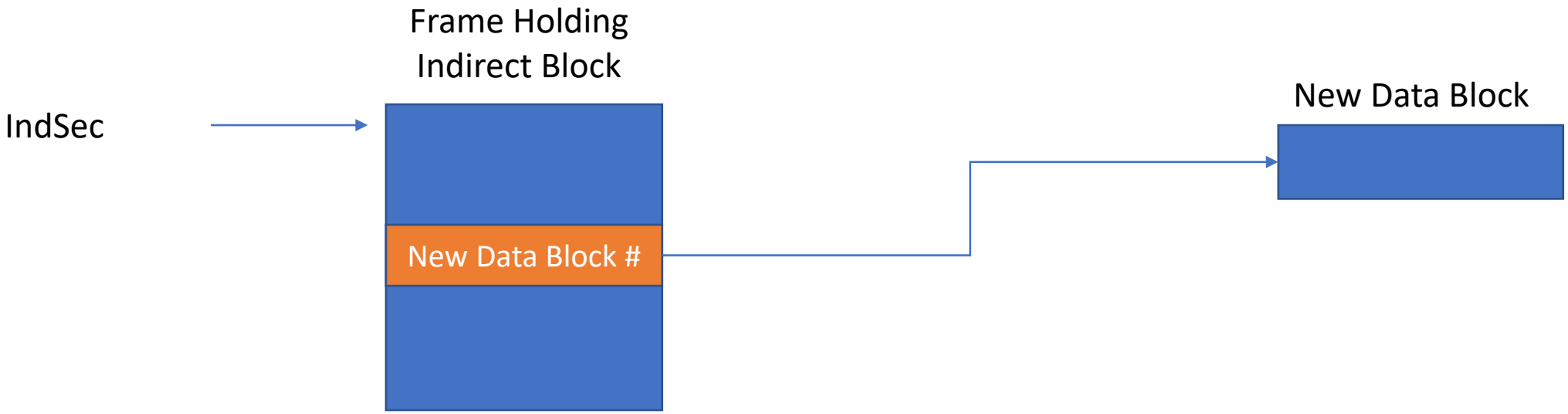
AllocateNewSector



AllocateNewSector

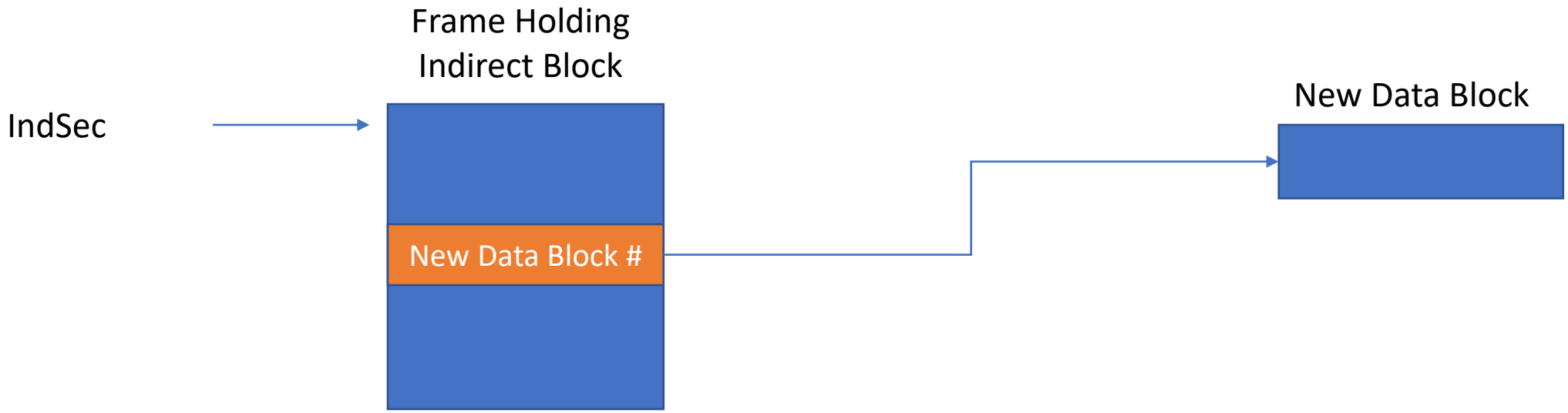


AllocateNewSector



How to write the new Data Block number in the frame? That's for you to figure out!

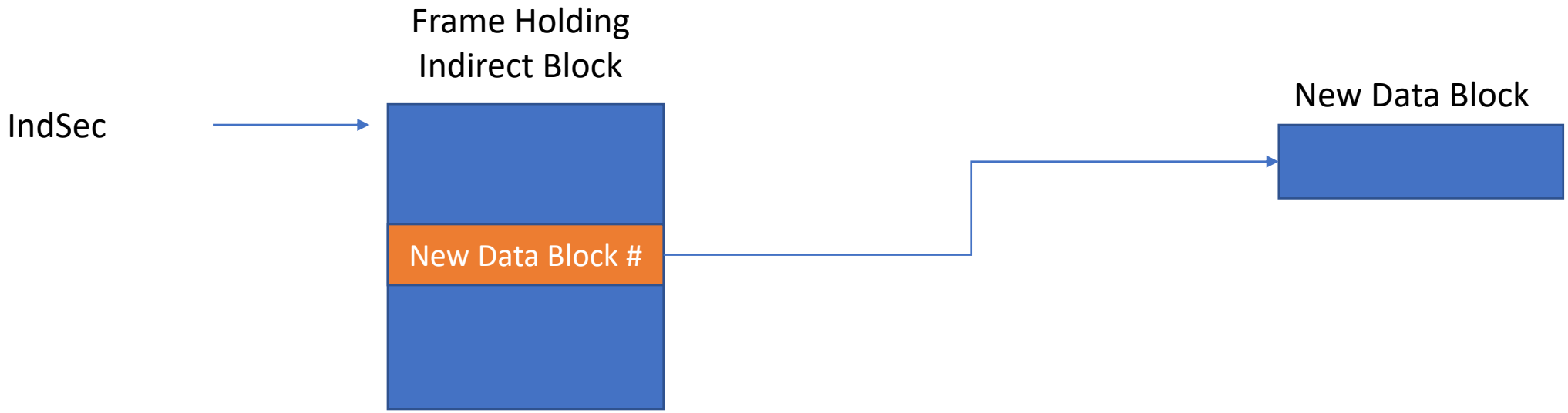
AllocateNewSector



How to write the new Data Block number in the frame? That's for you to figure out!

Hint: You need to calculate the address inside of the frame by calculating the offset inside of the frame. The offset will depend on the logical sector number of the file.

AllocateNewSector



How to write the new Data Block number in the frame? That's for you to figure out!

Hint: You need to calculate the address inside of the frame by calculating the offset inside of the frame. The offset will depend on the logical sector number of the file.

- All of what we have done, is just in memory.
- To actually save the inode and the indirect block on disk, you have to use WriteInode and SaveIndirect methods.
- Make sure you call them at the right time.

Sys_Stat

- After checking arguments, calls filesystem.Stat

```
method Stat (localName: String, statBuf: ptr to statInfo) returns int
```

- StatInfo is a record that is defined in Syscall.h. We have to populate its fields in this method.
- One populated you have to copy the local statInfo record to the user address space (one provided in the argument, the statBuf)
  - Use CopyByteToVirtual for that. Each record entry needs 4 bytes.