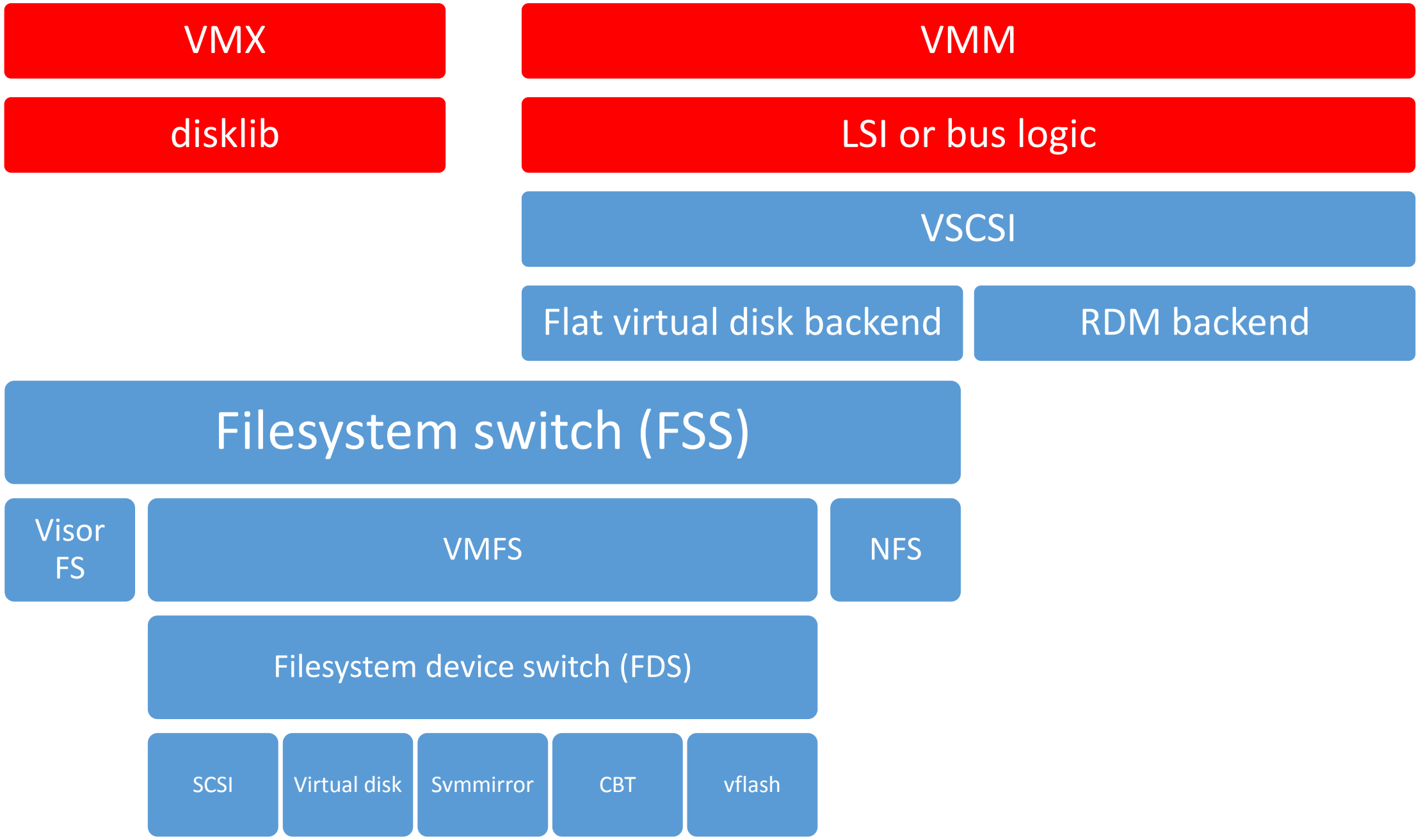
ESX storage intro


VMX
VMM
disklib
LSI or bus logic
VSCSI
Flat virtual disk backend
RDM backend
Filesystem switch (FSS)
Visor FS
VMFS
NFS
Filesystem device switch (FDS)
SCSI
Virtual disk
Svmmirror
CBT
vflash

Monitor to vscsi

BusLogicCmd_ExecuteSCSICommand

BusLogicSendCommand(adapter, targetID, SCSI_Command *cmd, SGPinArrType *lPtr)

VMKCall_VSCSIExecuteCommand(adapter->vscsiIDs[targetID], cmd, lPtr)

```
struct  SGPinArrType {
   uint32 sgLen;
   sgPinType sg[0];
};
```

```
struct SCSI_Command {
   SCSI_CommandType          type;
   uint64                    startTC;
   uint8                     cdb[SCSI_MAX_CMD_LEN];
   uint32                    cdbLength;
   uint32                    dataLength;
   xferDir                   direction;
   uint8                     lun;
   uint64                    sectorPos;
   SG_Array                  sgArr;
};
```

```
struct VSCSI_Handle {
   319    * Total size, number of blocks and blocksize on the given target
   321    uint64            length;
   322    uint64            numBlocks;
   323    uint32            blockSize;

   329    * SCSI handle for RDM backends.
   331    SCSI_Handle       *scsiHandle;

   336    * List of filters attached to this device, protected by VSCSI_Handle's
   341    List_Links        filterList;

   428    List_Links        cmdQueue;     // command queue

   312    uint8             virtualAdapterID;  // XXX Revisit
   313    uint8             virtualTargetID;
}
```

VSCSI dispatch

VSCSI_VmkExecuteCommand(VSCSI_HandleID..)
handle = VSCSI_HandleFind(handleID) // vscsiHandleArray[index] = handle; // at VSCSI_CreateDevice
VSCSI_HandleCommand
completionFrame = Async_PushCallbackFrameSafe(token, VSCSIAsyncIODone);
VSCSI_IssueCommandBE(handle, cmd, token, extCmd, 0, errStatus);
handle->devOps->VSCSI_VirtCommand(handle, cmd, token)

flatBE.c VSCSI_VirtCommand = VSCSI_FSCommand,
cbrc_filter.c VSCSI_VirtCommand = CBRCFilterIssueCommand,

VSCSI_IssueFSAsync
status = FSS_AsyncFileIO(handleID, &cmd->sgArr, token, flags);

VSCSI completion

```
VSCSI_VmkExecuteCommand(VSCSI_Handle..)
        VSCSI_HandleCommand
                completionFrame = Async_PushCallbackFrameSafe(token, VSCSIAsyncIODone);
                VSCSI_IssueCommandBE(handle, cmd, token, extCmd, 0, errStatus);
                        handle->devOps->VSCSI_VirtCommand(handle, cmd, token)


    VSCSIAsyncIODone(Async_Token *token, void *data)
    {
      SCSI_Result *result;
      ASSERT_FINAL_CALLBACK(token);
      result = (SCSI_Result *)token->result;
      VSCSICompleteIOCommand(token, // last callback
                    ((VSCSICompletionFrame *)data)->handle->handleID,
                    result->bytesXferred);
    }
```

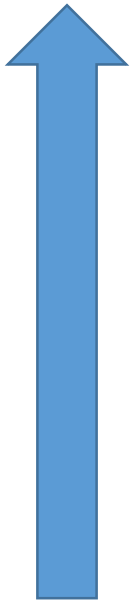
VSCSI completion

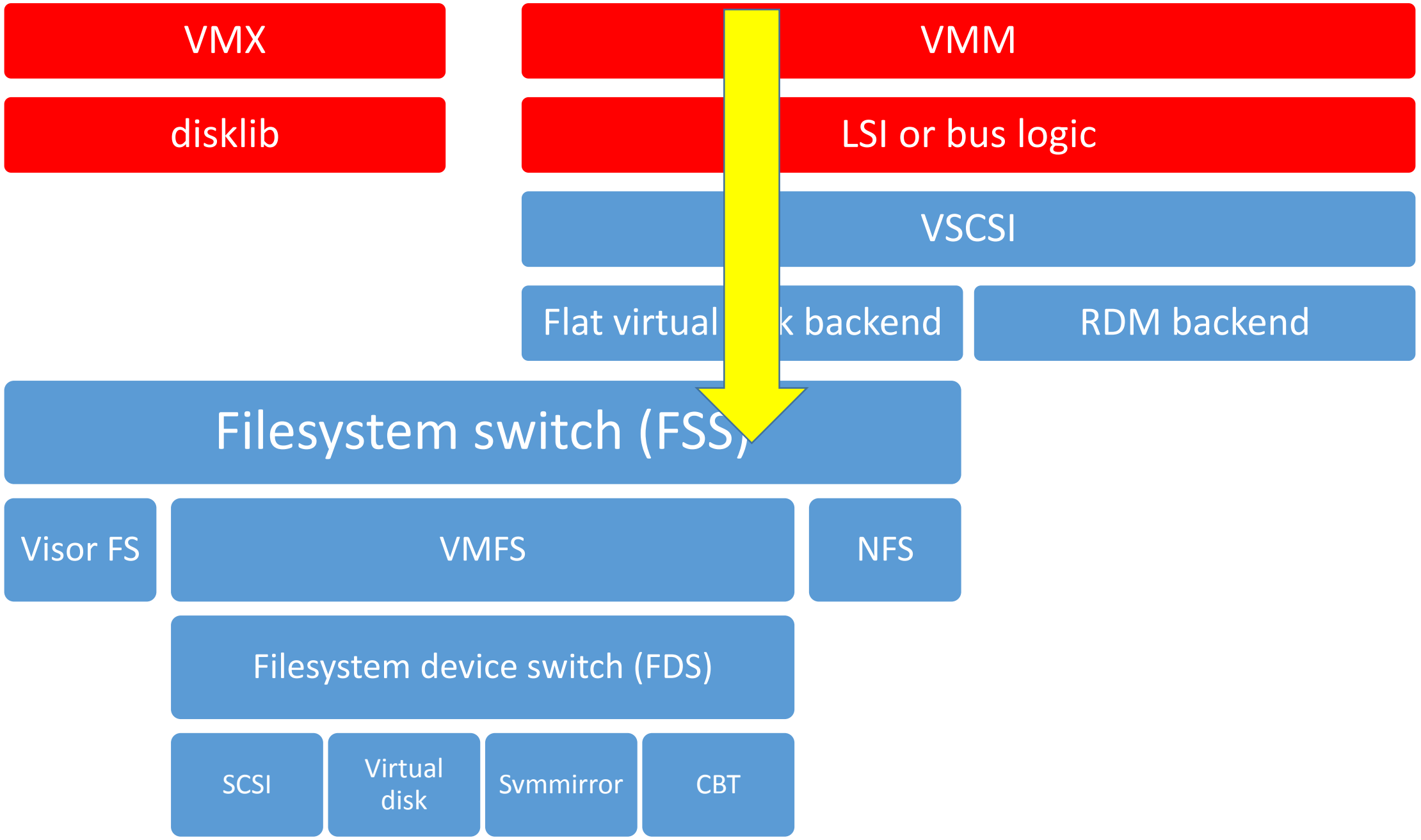
```
 ioFrame = Async_PushCallbackFrameSafe(token, SVMAsyncGuestIODone,
                                       sizeof(SVM_GuestIOFrame));

static void SVMAsyncGuestIODone(Async_Token *token, void *data)
{

   SVMFree(ioFrame->fileSGA);

   Async_TokenCallback(ioFrame->parentToken);
   Async_ReleaseToken(ioFrame->parentToken); // matches alloc in SVM_AsyncIO
}

Async_TokenCallback(Async_Token *token)
{
   if (LIKELY(token->frameList))
      Async_PopCallbackFrame(token);
  else
      Async_IODone(token); // last one
}
```

VSCSI flat backend to FSS

```
VSCSI_VmkExecuteCommand(VSCSI_HandleID..)
        handle = VSCSI_HandleFind(handleID) // vscsiHandleArray[index] = handle; // at VSCSI_CreateDevice
        VSCSI_HandleCommand
                completionFrame = Async_PushCallbackFrameSafe(token, VSCSIAsyncIODone);
                VSCSI_IssueCommandBE(handle, cmd, token, extCmd, 0, errStatus);
                        handle->devOps->VSCSI_VirtCommand(handle, cmd, token)

flatBE.c        VSCSI_VirtCommand = VSCSI_FSCommand,
cbrc_filter.c   VSCSI_VirtCommand = CBRCFilterIssueCommand,

VSCSI_IssueFSAsync
        status = FSS_AsyncFileIO(handleID, &cmd->sgArr, token, flags);
```

FSS_AsyncFileIO
VSCSI_FSCommand
VSCSI_IssueCommandBE
VSCSI_HandleCommand
VSCSI_VmkExecuteCommand
PVSCSIVmkProcessCmd
PVSCSIVmkProcessRequestRing
PVSCSI_ProcessRing
VMMVMKCall_Call
VMKVMM_EnterVMKernel




VMX
VMM
disklib
LSI or bus logic
VSCSI
Flat virtual
k backend
RDM backend
Filesystem switch (FSS)
Visor FS
VMFS
NFS
Filesystem device switch (FDS)
SCSI
Virtual disk
Svmmirror
CBT

Disklib

- Disk operations in ESX
  - Create, clone virtual disks
  - Query sparse disks and chains
- Disk operations and IO in hosted

- All disks are stacks of extents (chains)
  - Same interface even if it’s a single flat extent
  - Writes go to the bottom and reads start at the bottom
  - Both disklib and the kernel (deltaDisk)

Disklib

```
/*
 * We have 3 kind of objects: Chain, Link and Extent
 * where Extent can be one of Flat, Sparse, Zero, VMFS or more.
 *
 * We have 4 layers of functional responsibility: Lib, Chain, Link, Extent.
 * o The Lib layer is responsible for library "glue" issues and interfacing
 *   with the user.
 * o The Chain layer is responsible for operations that operate on multiple
 *   links, such as Copy-on-write functionality, re-parenting, combine etc.
 * o The link layer is responsible for managing a collection of extents
 * o The extent layer is responsible for the backing store of data, be it
 *   flat file, zero backed, sparse file backed or VMFS file backed.
 *
*/
```

```
* This is the layout of the objects in memory:
*
*         /----------------------\
*         | /------------------\ |   LIB LAYER
*         | | DiskLibHandleInt | |
* --------| |------------------| |--------------
*         | |      Chain       | |
*         | \------------------/ |
*         \----------------------/
*                                   CHAIN LAYER
*         /-----------------\
*         | /-------------\ |
*         | |  ChainLink  | |
* --------| |-------------| |--------------
*         | |    Link     | |
*         | \-------------/ |
*         \-----------------/
*                               LINK LAYER
*         /-----------------\
*         | /-------------\ |
*         | | LinkExtent  | |
* --------| |-------------| |--------------
*         | |   Extent    | |
*         | \-------------/ |
*         \-----------------/   EXTENT LAYER
*
*
* LIB LAYER    is only visible from diskLib.c
* CHAIN LAYER  is only visible from chain.c
* LINK LAYER   is only visible from link.c
* EXTENT LAYER is only visible from flat.c, sparse.c, zero.c and vmfs.c
*
```

FlatCommonExtentCreate // a type of extents

DiskLibCreateObj

DiskLibCreateObjPosix

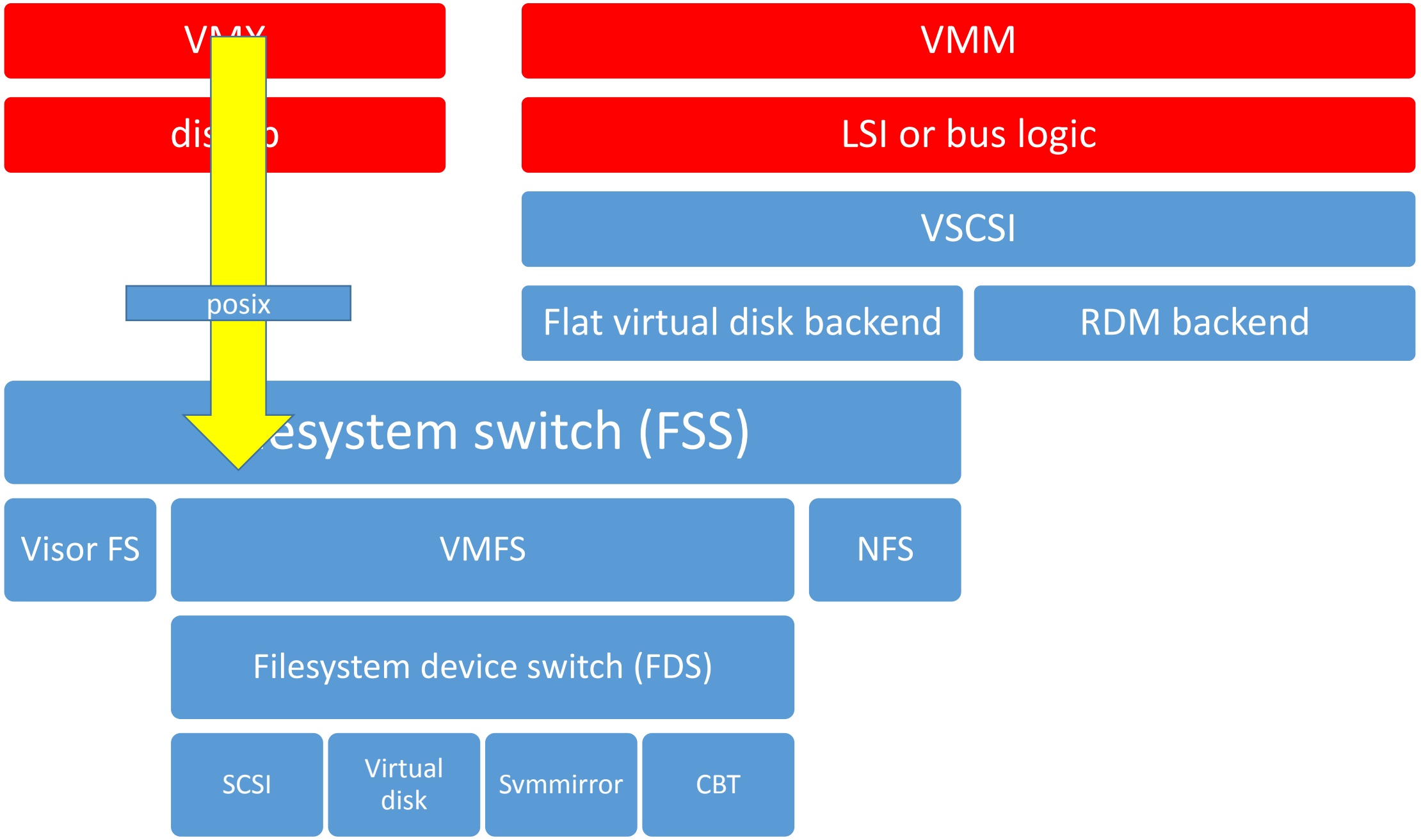
ObjLib_Create

```
374 extern const ObjLibBackend fileBE; // a type of object
375 extern const ObjLibBackend vblobBE;
377 extern const ObjLibBackend vsanObjBE;
379 extern const ObjLibBackend vvolObjBE;
380 extern const ObjLibBackend encFileBE;
```

```
71 const ObjLibBEInterface fileBEInterface = {
72    NULL, /* Init */
73    NULL, /* Exit */
74    FileBECreate,
75    FileBEOpen,
76    FileBEClose,
```

FileBECreate

FileIO_Create // posix


VMX
VMM
dis p
LSI or bus logic
VSCSI
posix
Flat virtual disk backend
RDM backend
esystem switch (FSS)
Visor FS
VMFS
NFS
Filesystem device switch (FDS)
SCSI
Virtual disk
Svmmirror
CBT

Filesystem switch (FSS)

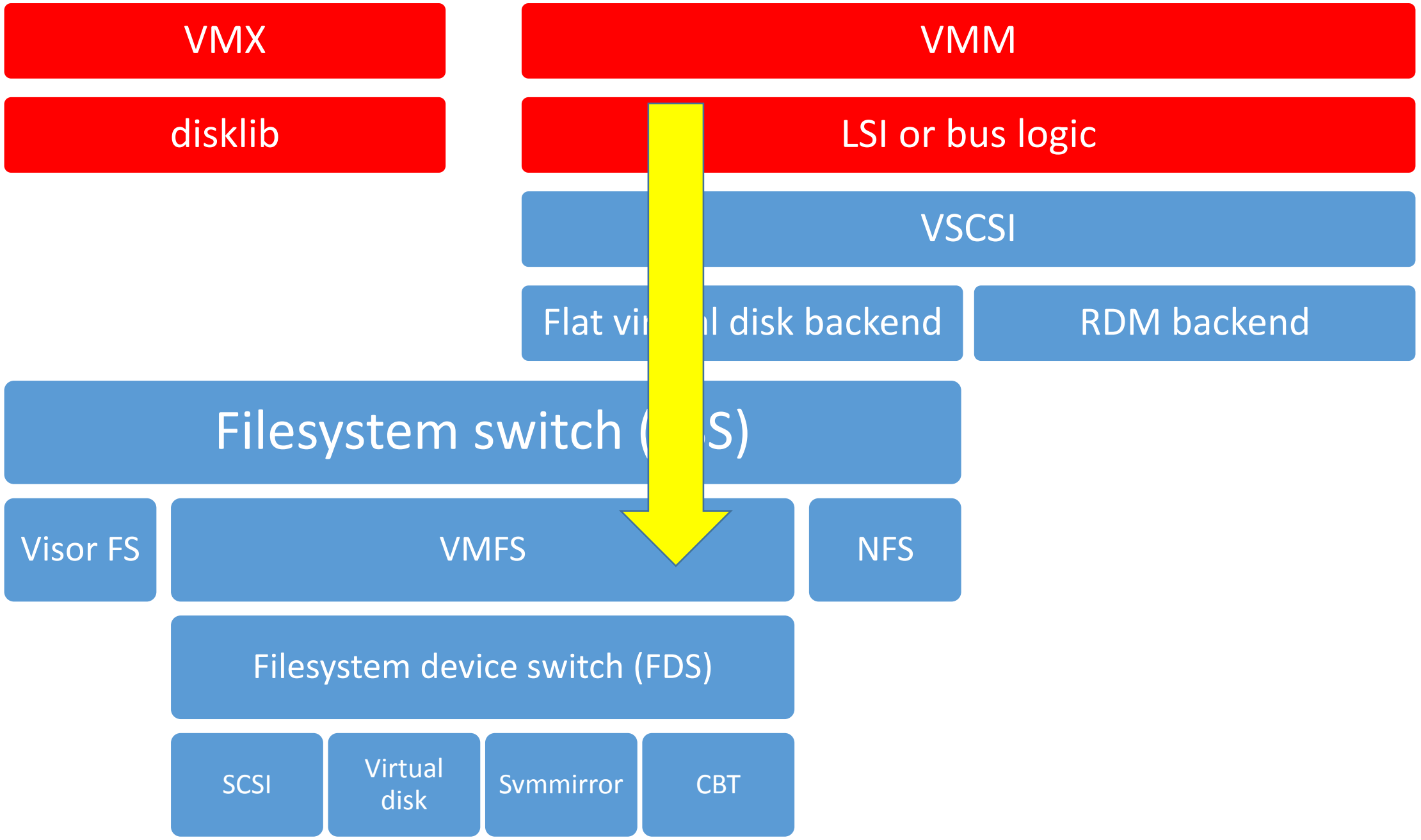
bora/vmkernel/virtscsi/flatBE.c:
VSCSI_IssueFSAsync

```
FS_FileHandleID handleID = handle->devDesc.fid; // stored when creating the virtual disk
ioDoneFrame = Async_PushCallbackFrameSafe(token, VSCSI_FSVirtAsyncDone)
status = FSS_AsyncFileIO(handleID, &cmd->sgArr, token, flags);
```

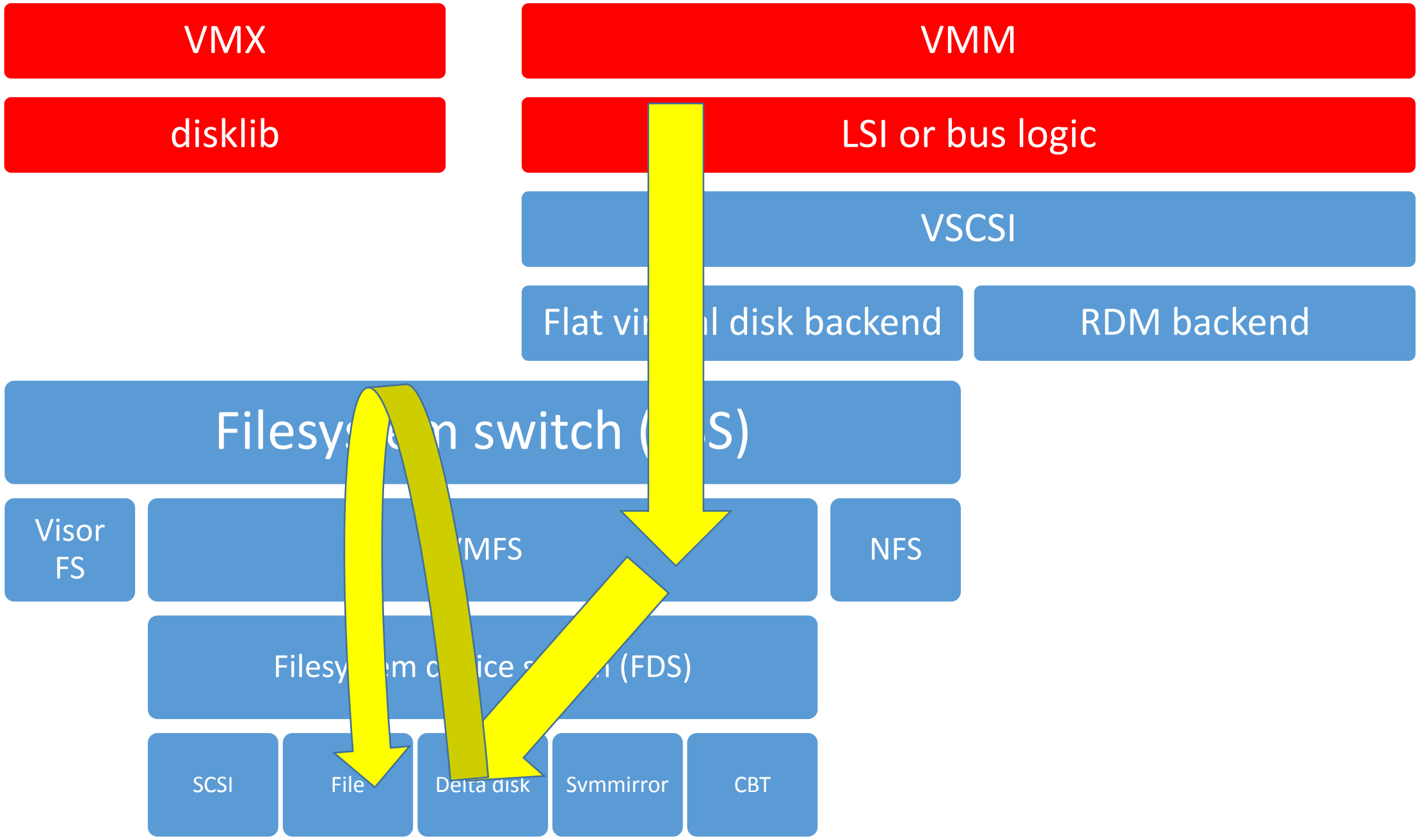
```
4303 FSS_AsyncFileIO(FS_FileHandleID fileHandleID, const SG_Array *sgArr,
4304             Async_Token *token, IO_Flags ioFlags)
```

```
FSS_FD2FILEOPS(desc)->FSS_FileIO(identity,
                                 desc,
                                 fhID,
                                 sgArr,
                                 token,
                                 ioFlags,
                                 bytesRead);
```

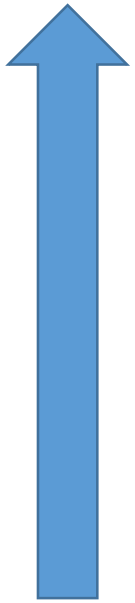
VMFS: .FSS_FileIO = Fil3_FileIOWithRetry // Fil3_FileIO // another talk on its own: lots of locking and weird things
NFS: .FSS_FileIO = NFSOpFileIO,
VisorFS: .FSS_FileIO = VisorFSFileIO,

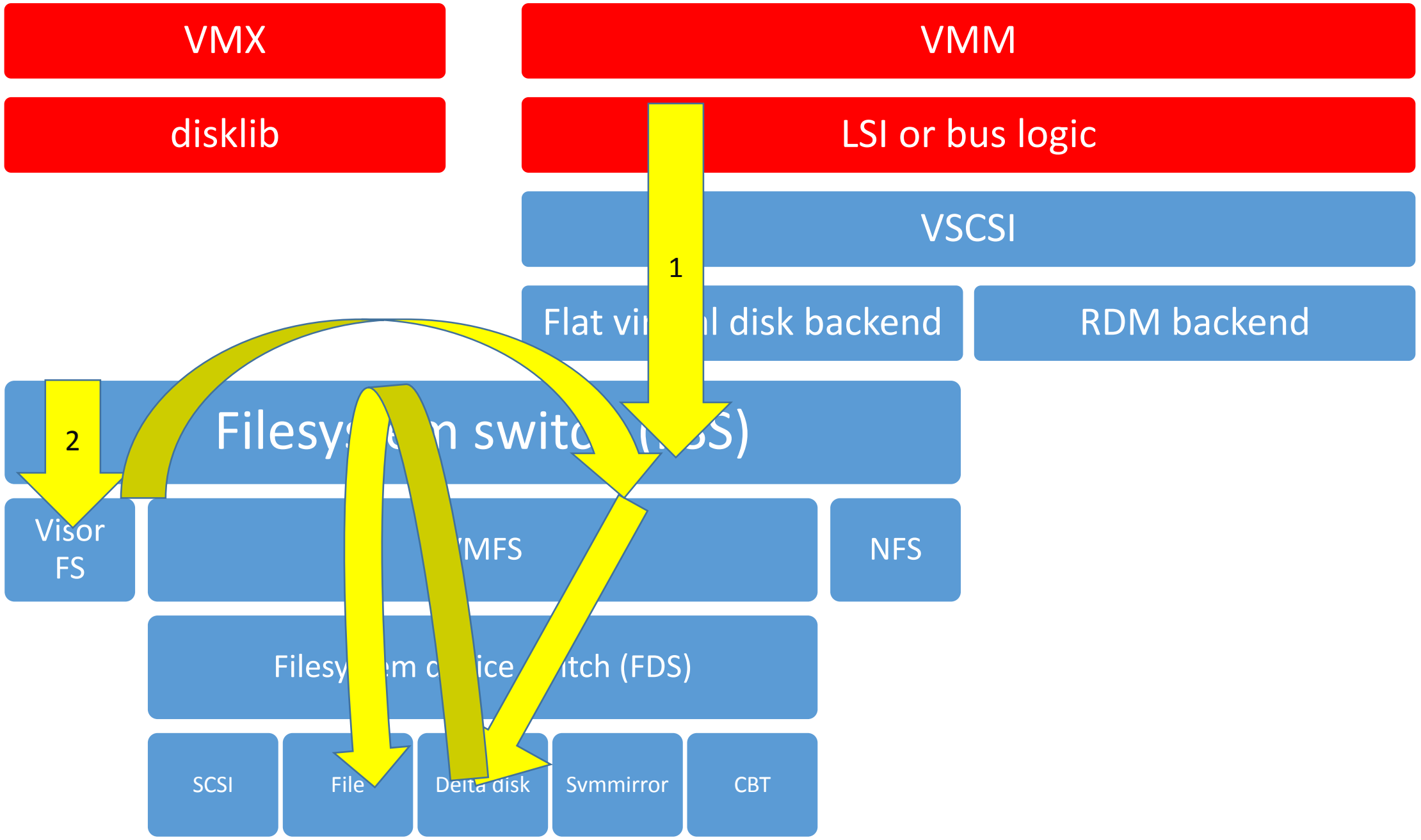

VMX
VMM
disklib
LSI or bus logic
VSCSI
Flat vi l disk backend
RDM backend
Filesystem switch (S)
Visor FS
VMFS
NFS
Filesystem device switch (FDS)
SCSI
Virtual disk
Svmmirror
CBT

```
Fil3_FileIO
        FDS_AsyncIO(fdsHandleArray, sg, ioFlags, token)
                FDSAutoAlignAndIssue
                        FDS_FHTODEVOPS(fdsHandle)->FDS_AsyncIO(fdsHandle->hid, sgArr);

Delta Disk:     .FDS_AsyncIO = COW_AsyncIO,
CBT:            .FDS_AsyncIO = CBT_AsyncIO,
file driver:    .FDS_AsyncIO = FSFile_AsyncIO,

FSFile_AsyncIO,
        FSS_AsyncFileIO(driverData->fhid, tmpSG, token)

COW_AsyncIO
        COW_FileIO
                fsInfo->extentOps->COW_FileIO(chi, level, sgArr, token, ioFlags);

SESPARSE:       .COW_FileIO = SESparseAsyncFileIO
VMFSSPARSE:     .COW_FileIO = VMFSSparseAsyncFileIO
```

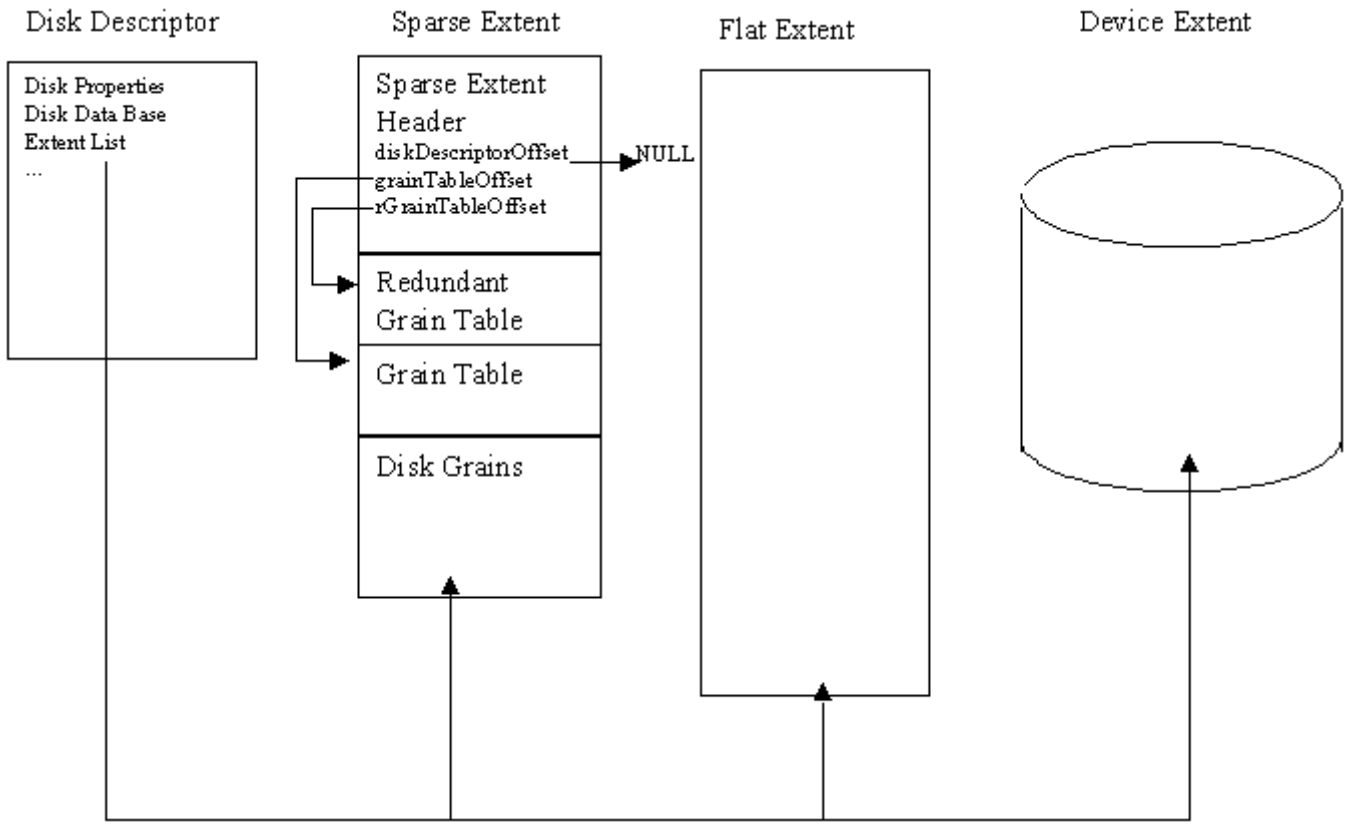

VMX
VMM
disklib
LSI or bus logic
VSCSI
Flat vi l disk backend
RDM backend
Filesy m switch (S)
Visor FS
MFS
NFS
Filesy m d ice (FDS)
SCSI
File
Delta disk
Svmmirror
CBT

...
Fil3_FileIO
Fil3_FileIOWithRetry
FSSFileIO
FSS_AsyncFileIO
SESparseAsyncFileIO // one type of virtual disk
FDS_AsyncIO
DevFSFileIO // the “/” file system
FSSFileIO
FSS_AsyncFileIO

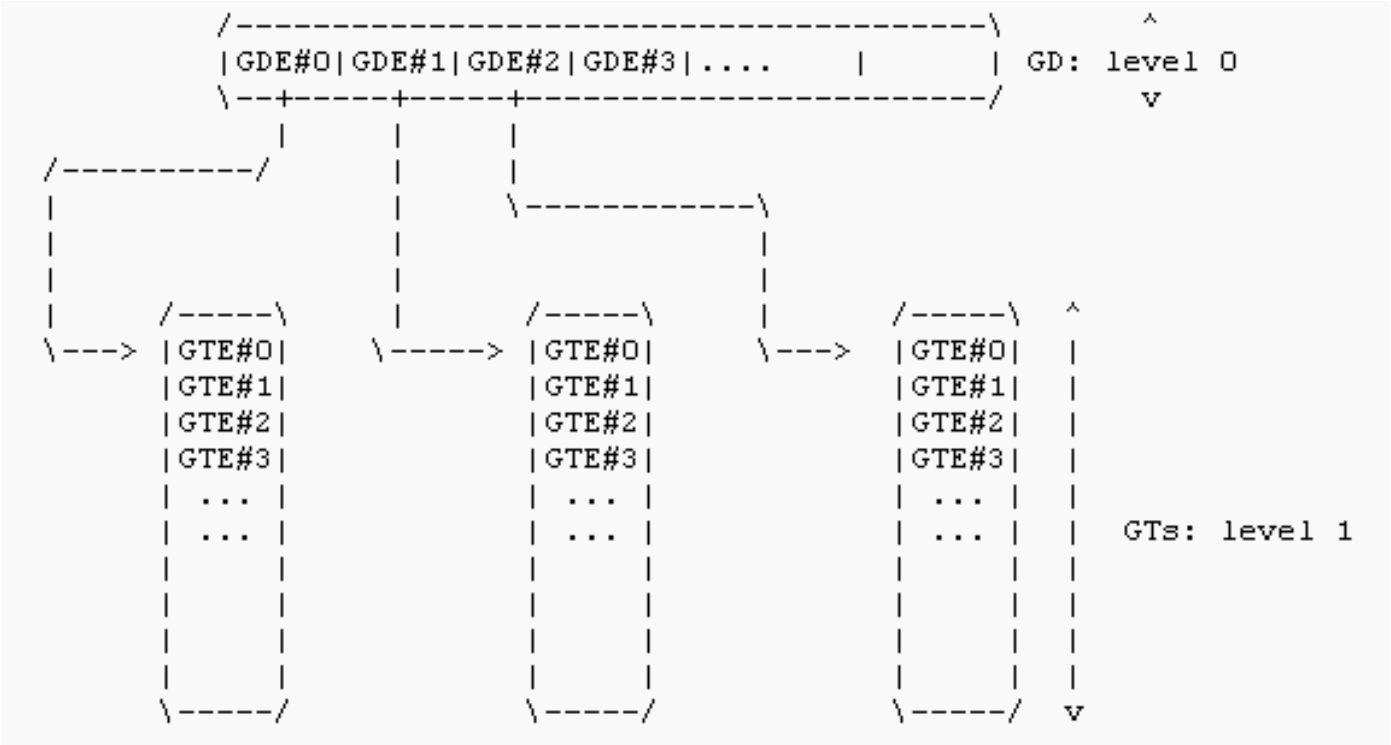



VMX
VMM
disklib
LSI or bus logic
VSCSI
1
Flat vi l disk backend
RDM backend
2
Filesy m switc (FSS)
Visor FS
MFS
NFS
Filesy m d ice itch (FDS)
SCSI
File
Delta disk
Svmmirror
CBT

Sparse extent

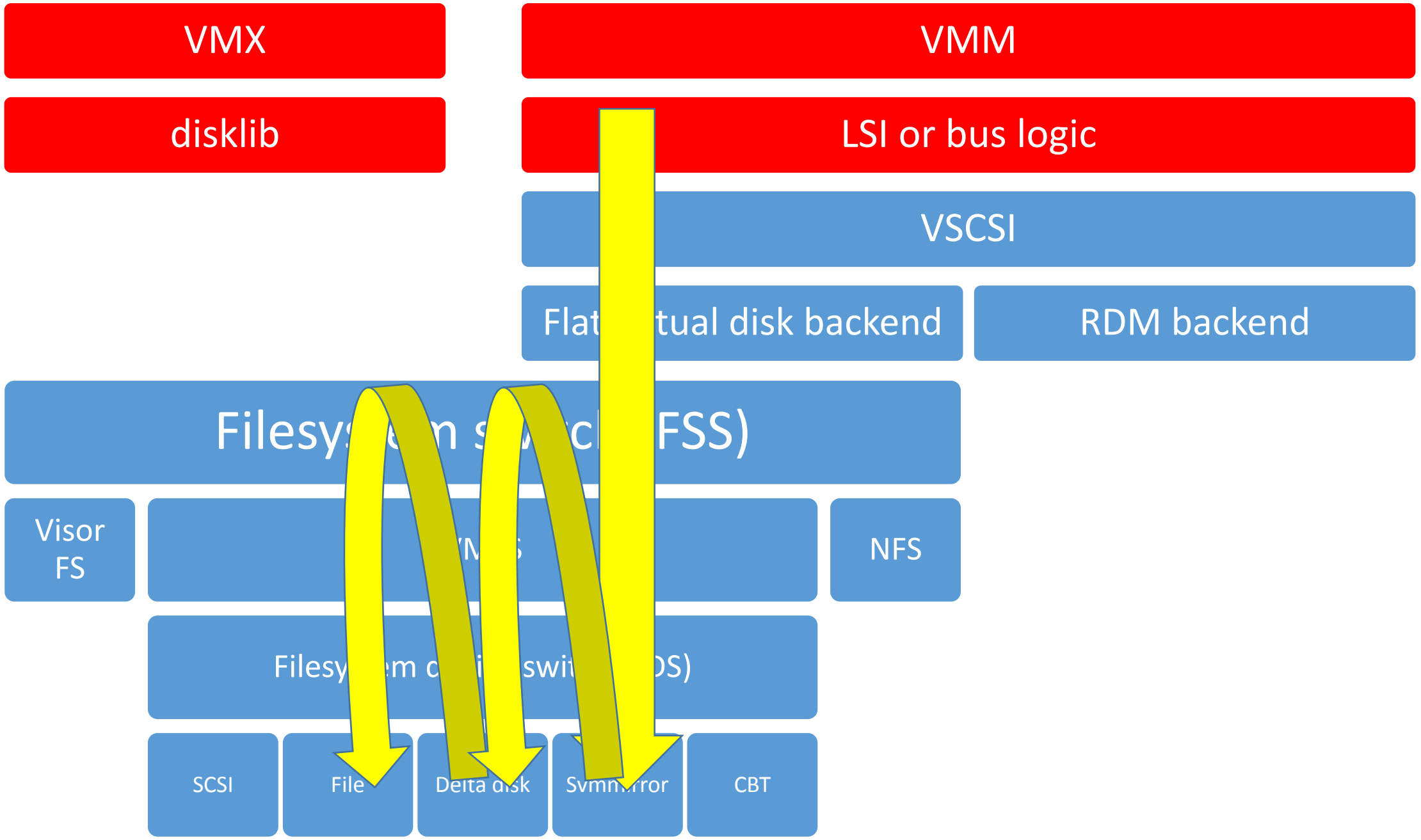


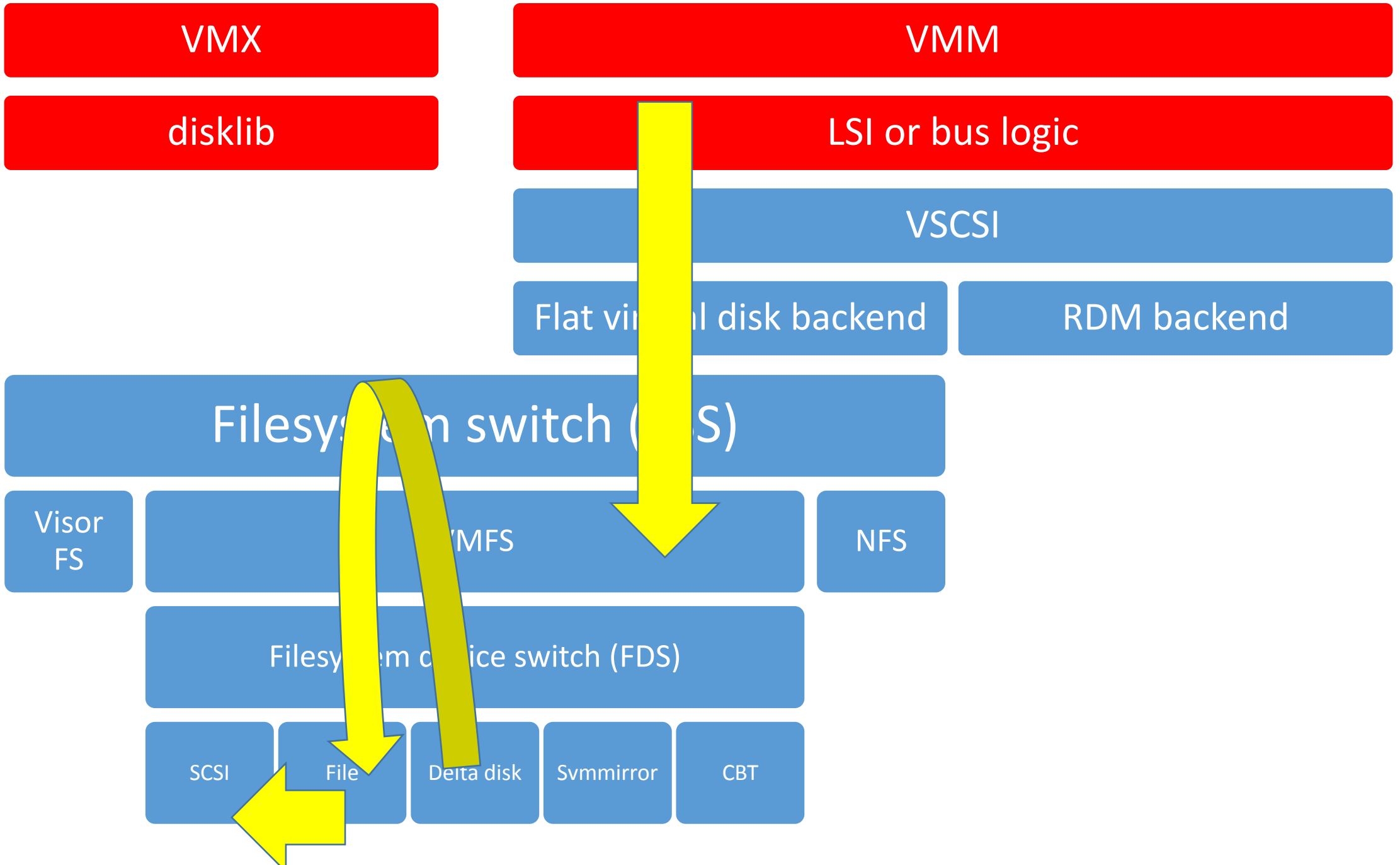
Sparse extent



```
static FDS_DeviceOps svmOps = {
   SVM_OpenDevice,
   SVM_CloseDevice,
   SVM_AsyncIO,
   SVM_Ioctl,
   SVM_RescanDevices,
   SVM_MakeDev,
};
```

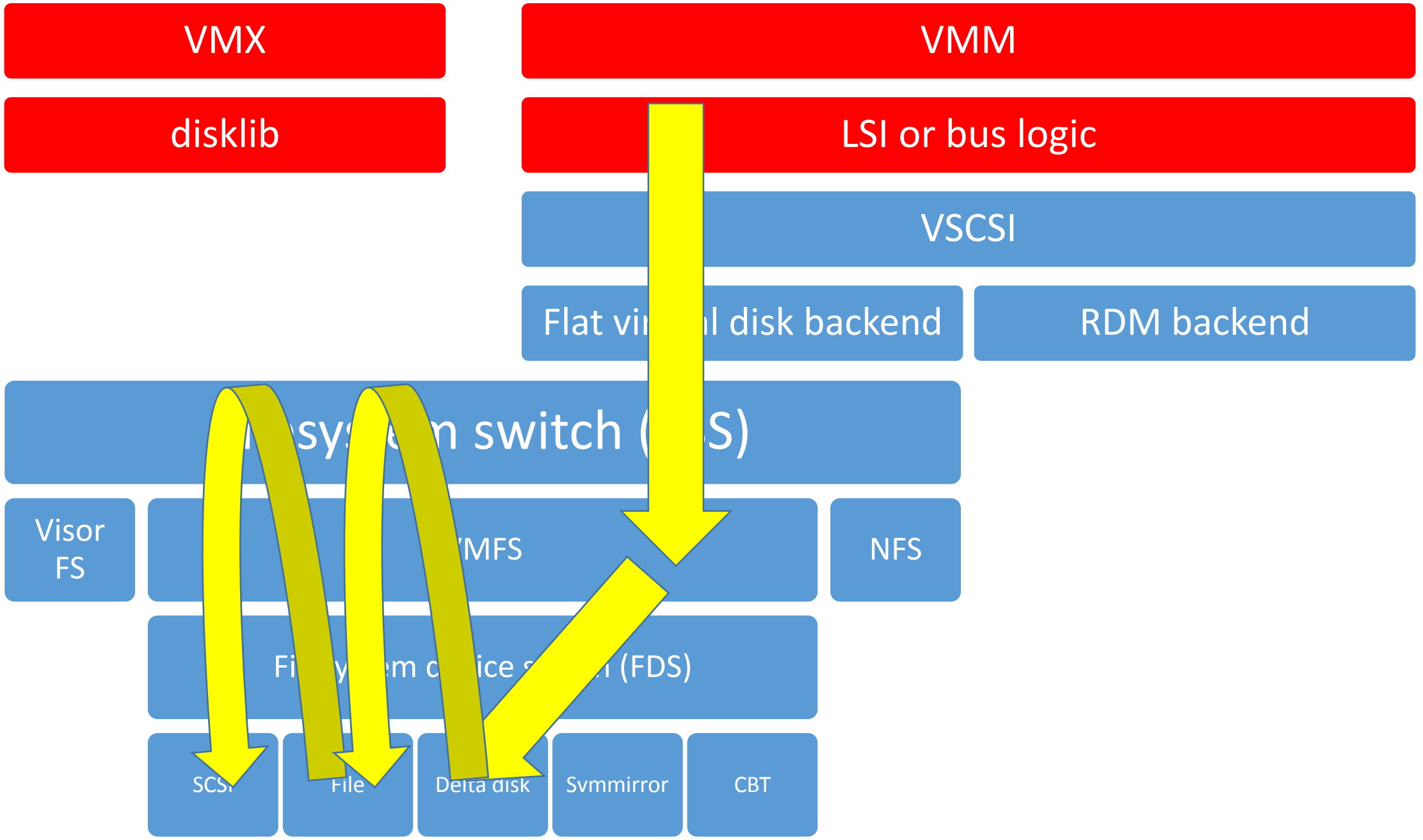
VMX:
VMKernel_FDSMakeDev
    SVM_MakeDev // kernel
        driverData->fhids[0] = args->da.mirror.fhids[0];
        driverData->fhids[1] = args->da.mirror.fhids[1];
        DevLib_CreateDevice // create the fds device
Mirror_InstallDiskMirror
    DiskLib_InstallMirrorDriver
      ioctlParams.request = IOCTLCMD_VMFS_GET_FILE_HANDLE,
     ioctlParams.argp = newFid;
     objErr = ObjLib_Ioctl(*newMetaObjHandle,
            &ioctlParams);

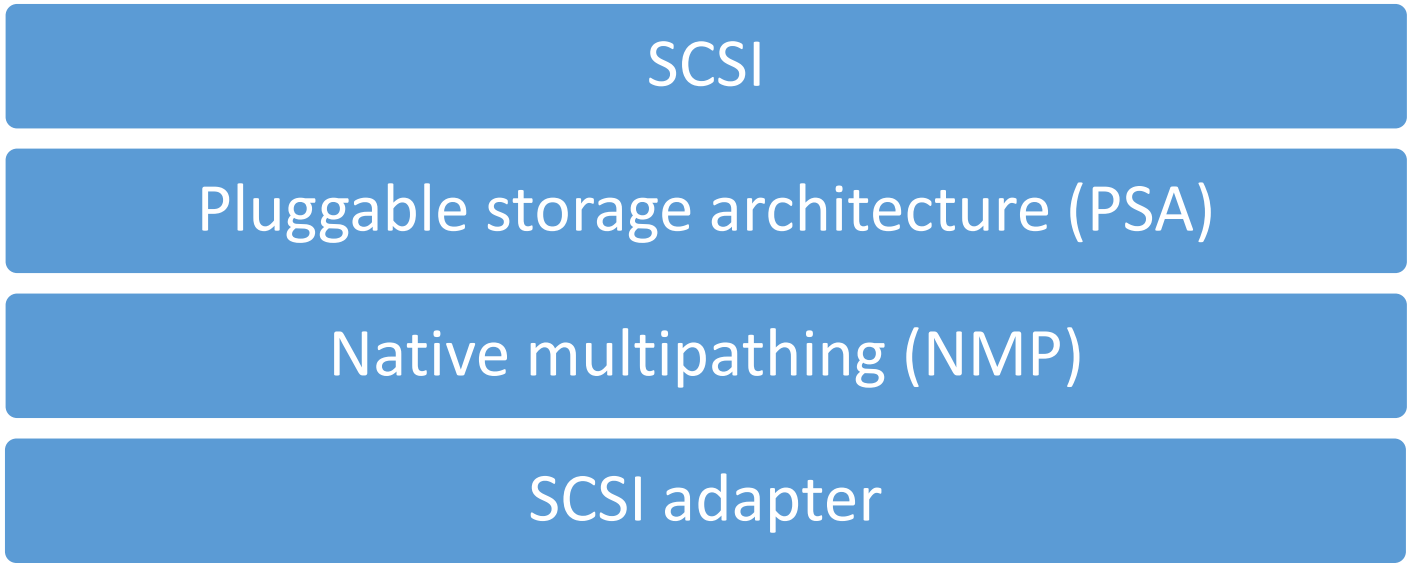

VMX
VMM
disklib
LSI or bus logic
VSCSI
RDM backend
NFS
SCSI
CBT


VMX
VMM
disklib
LSI or bus logic
VSCSI
Flat vi l disk backend
RDM backend
Filesy m switch (S)
Visor FS
MFS
NFS
Filesy m d ice switch (FDS)
SCSI
File
Delta disk
Svmmirror
CBT

FSFile_AsyncIO
FSS_AsyncFileIO
fdsHandle)->FDS_AsyncIO

Delta Disk: .FDS_AsyncIO = COW_AsyncIO,
CBT: .FDS_AsyncIO = CBT_AsyncIO,
file driver: .FDS_AsyncIO = FSFile_AsyncIO,
genericScsiDriver.c 88 .FDS_AsyncIO = (FDS_AsyncIOOp)FDS_NotSupported,
cdromDriver.c 336 .FDS_AsyncIO = FDS_CommonAsyncIO,
diskDriver.c 442 .FDS_AsyncIO = FDS_CommonAsyncIO,
charDriver.c 111 .FDS_AsyncIO = CharDriverAsyncIO,


VMX
VMM
disklib
LSI or bus logic
VSCSI
Flat vi l disk backend
RDM backend
sy m switch (S)
Visor FS
MFS
NFS
Fi em d ice (FDS)
SCSI
File
Delta disk
Svmmirror
CBT



A SCSI path:

adapter, channel, target, LUN

SCSI_IssueAsyncDeviceToken
SCSIAsyncDeviceCommand
SCSIStartDevice // takes care of limiting the queue of IO
(other plug-ins in PSA framework)
nmp_LogicalDeviceStart
nmp_IssueCommandToLogicalDevice
nmp_SelectPhysicalPathAndIssueCommand
nmp_PspSelectPhysicalPath
PSP->selectPath
nmp_IssueCommandToPhysicalPath
SCSIIssueAsyncPathCommand

TODO:

- I/O emulation in monitor
- vmfs asyncIO !!!
- Disklib
- user worlds and buffer cache
- Disk scheduling
- Linux compatibility layer
- Physical device driver
- SAN and LUNs
- objects, VSAN
- what happens with native snapshots?
- FDS retries (FSS retries?) and libAIO
- VAAI
- Mount operations
- SAN discovery