Examples of exercises
File System

ARCOS

Operating Systems Design
Degree in Computer Engineering
University Carlos III of Madrid

Exercise
statement (1/3)

We have a single processor machine and need to implement a file system for an UNIX operating system with a non-preemptive monolithic kernel. The file system requires the following:

- Reserve the initial block for a future partition table, which will be filled with zeroes.
- The superblock will be stored in a complete disk block.
- The free resource management will be done through a bitmap, using a byte with value 0 to indicate a free resource and 1 for a used resource.
- The number of elements in the file system (files and directories) will be, as maximum, **`numInodes`**. The i-node related to each entry will be allocated in a complete disk block.

Exercise
statement (2/3)

- The file names will have a maximum length of 200 characters.
- Each file will only have a disk block.
- The maximum number of data blocks in disk will be **`numDataBlocks`**.
- There is only one root directory, and no subdirectories. However, the file system will be designed for having a maximum of 200 entries in each directory, which will be stored in the corresponding i-node.
- Each file has a read and writer pointers (not shared), and will not be possible to unmount the file system if there is, at least, an opened file.

Exercise
statement (3/3)

You are asked to:

a) Design the data structures in disk that are required for the exercise in a simple way.

b) Design the data structures in memory that are required for the exercise.

c) Design the low level functions for handling disk blocks (alloc, free, bmap) and i-nodes (ialloc, ifree y namei).

d) Design the file system interface mount, umount, open, close, creat, unlink, read, write and the mkfs utility.

Exercise
solution

1. Initial approach:
    1. Draw a diagram of initial system state
    2. Modify the diagram to incorporate the exercise requirements
2. Answer the proposed questions
3. Review the answers

Exercise
solution

1. Initial approach:
    1. Draw a diagram of initial system state
    2. Modify the diagram to incorporate the exercise requirements
2. Answer the proposed questions
3. Review the answers

General approach

Memory

File system syscalls

Descriptors	namei usage		i-node alloc.	File Attr.	I/O	File Sys.	View
open creat dup pipe close	open creat chdir chroot chown chmod	stat link unlink mknod mount umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

Low level algorithms of the file system

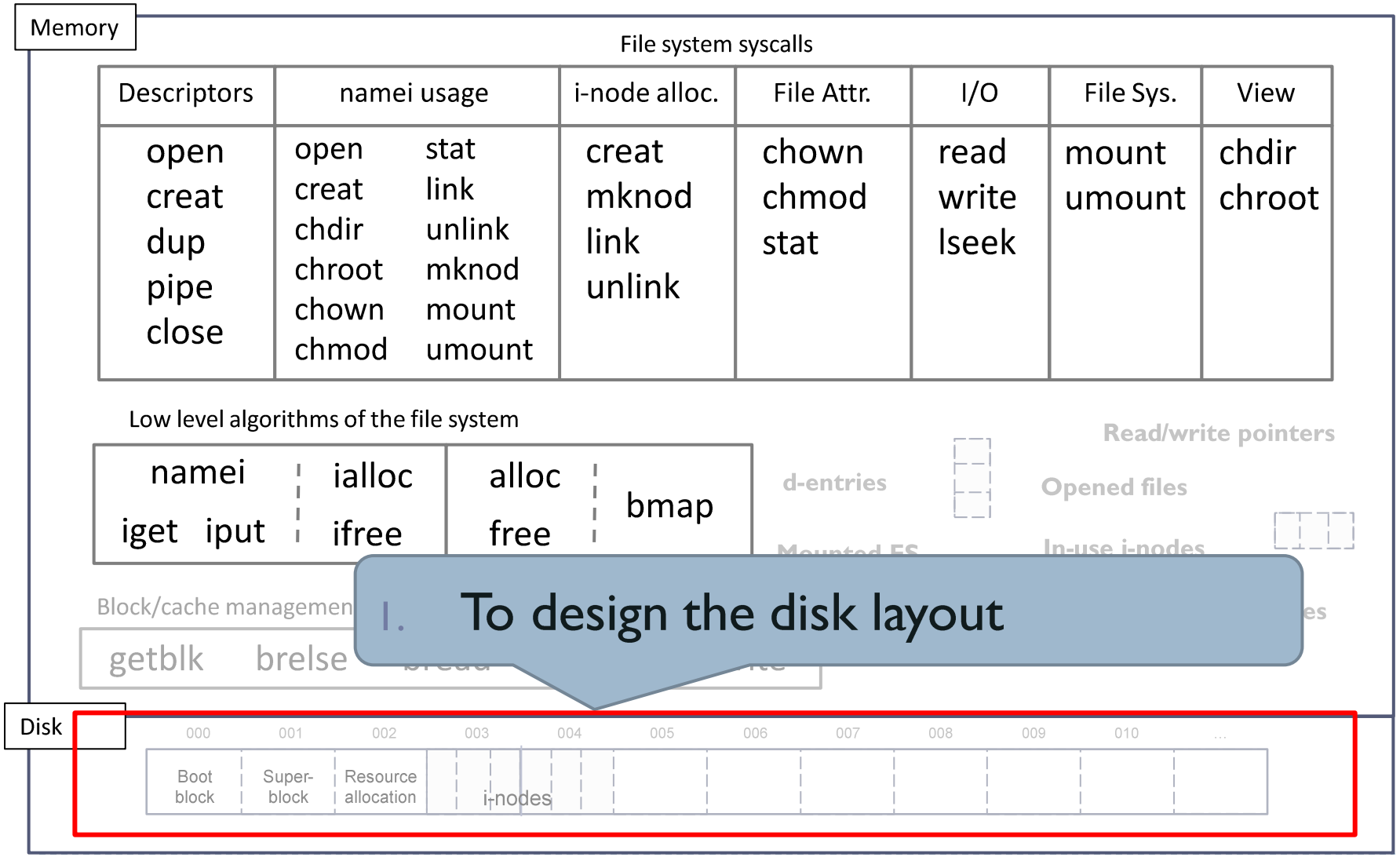
namei		ialloc	alloc		bmap
iget	iput	ifree	free		

Block/cache management algorithms

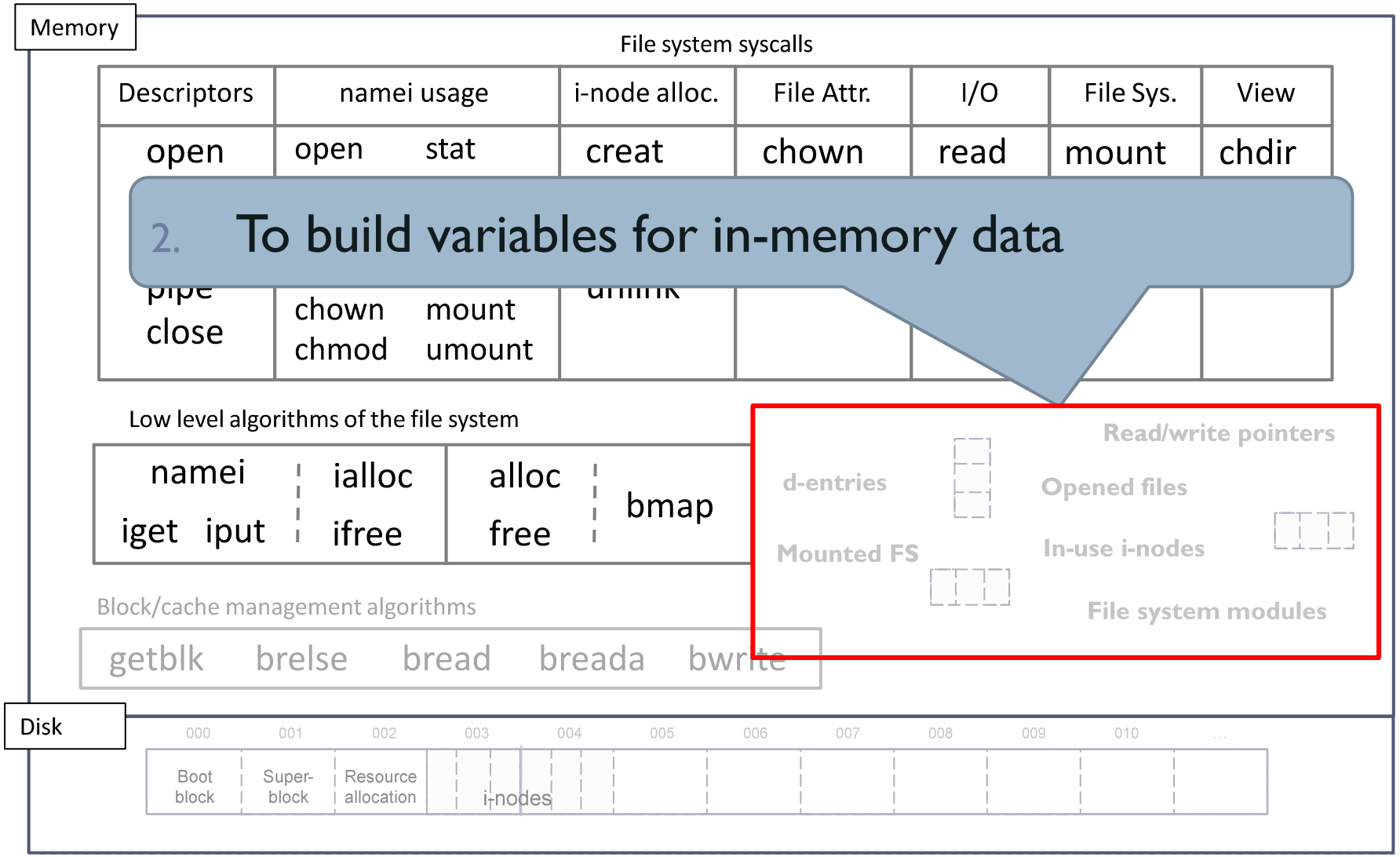
getblk	brelse	bread	breada	bwrite

Disk

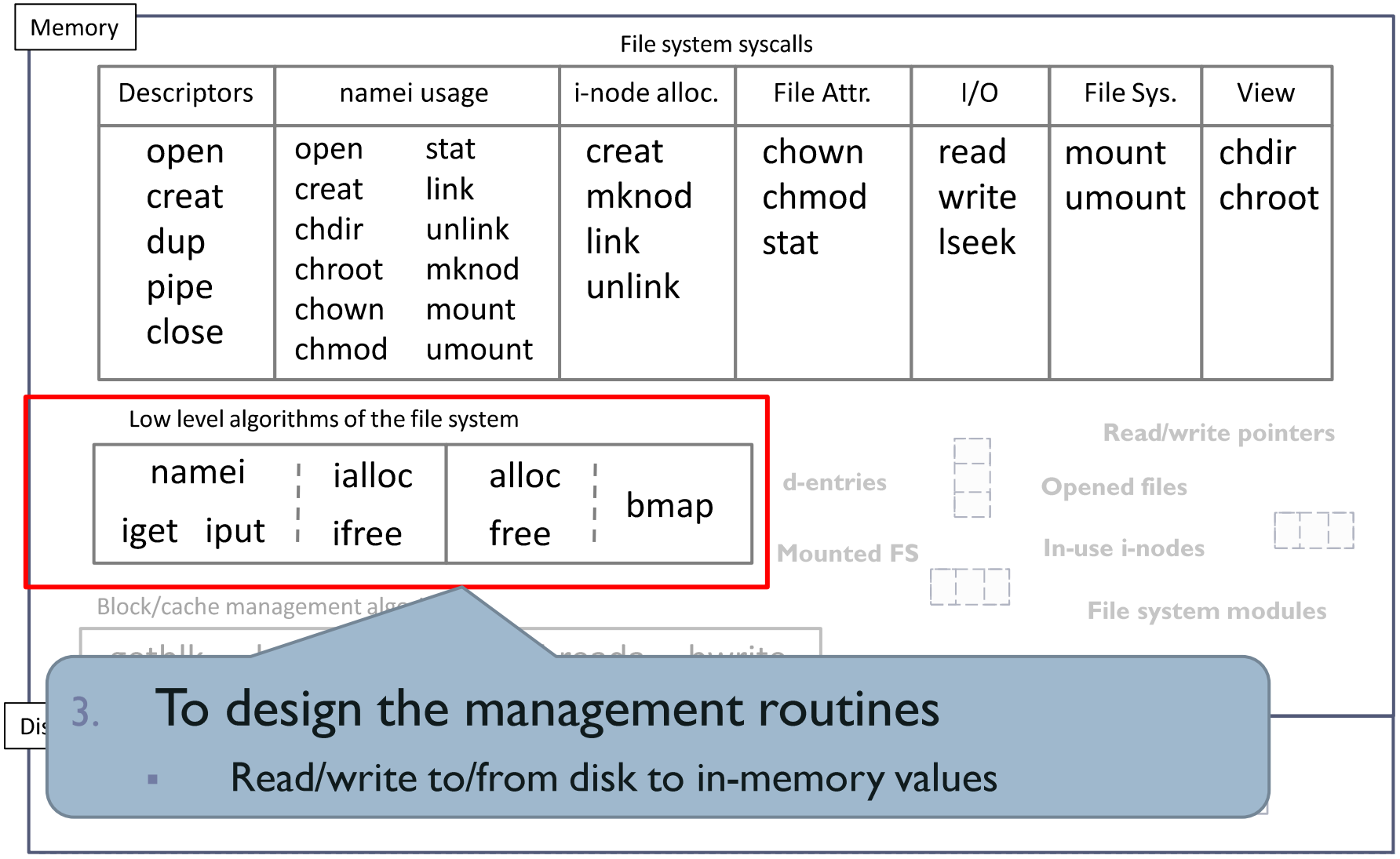
General approach



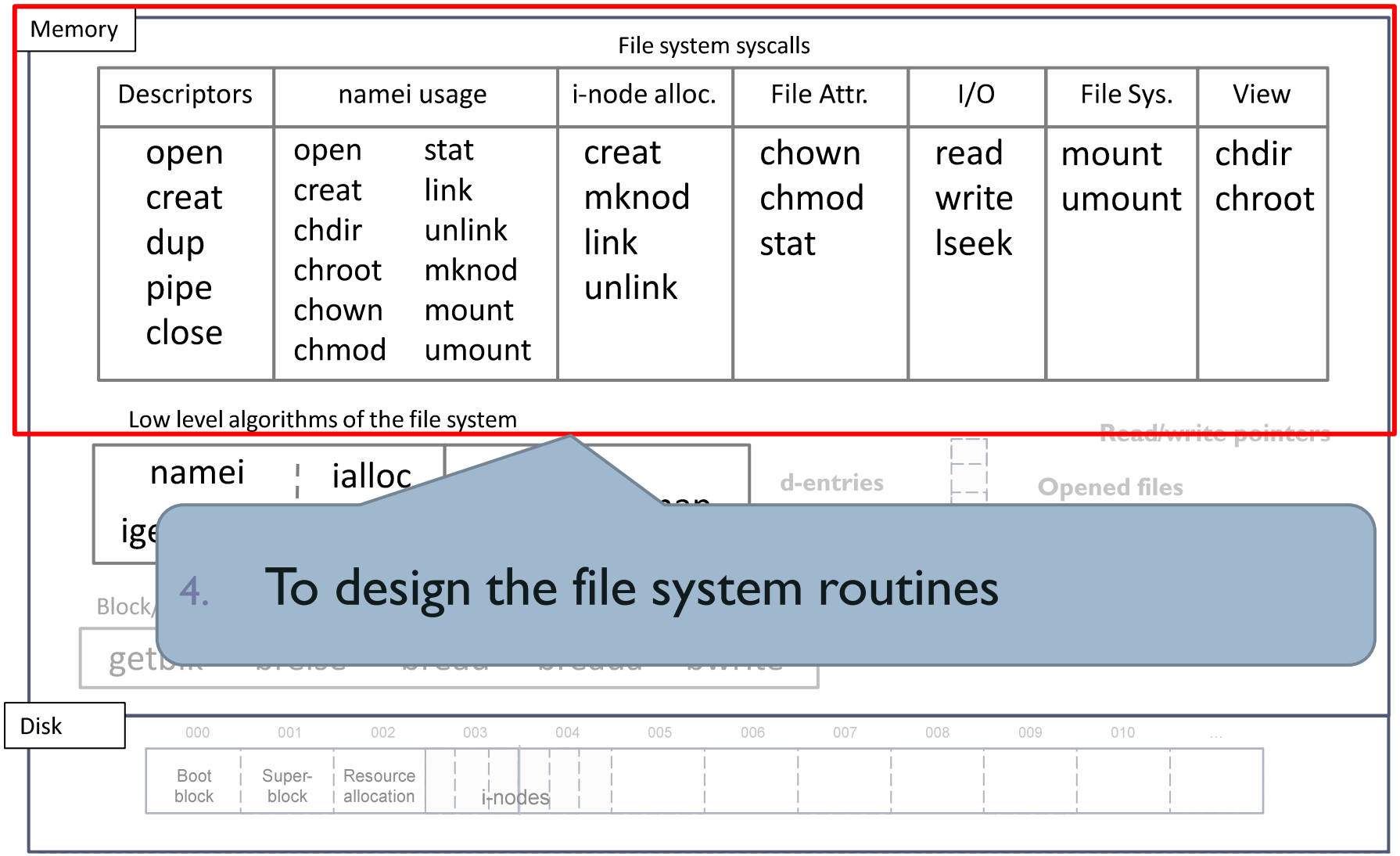
General approach



General approach



General approach



Exercise
solution

1. Initial approach:
    1. Draw a diagram of initial system state
    2. Modify the diagram to incorporate the exercise requirements
2. Answer the proposed questions
3. Review the answers

General approach

Memory

File system syscalls

Descriptors	namei usage		i-node alloc.	File Attr.	I/O	File Sys.	View
open creat dup pipe close	open creat chdir chroot chown chmod	stat link unlink mknod mount umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

Low level algorithms of the file system

namei		ialloc	alloc		bmap
iget	iput	ifree	free		

Read/write pointers

d-entries

Opened files

In-use i-nodes

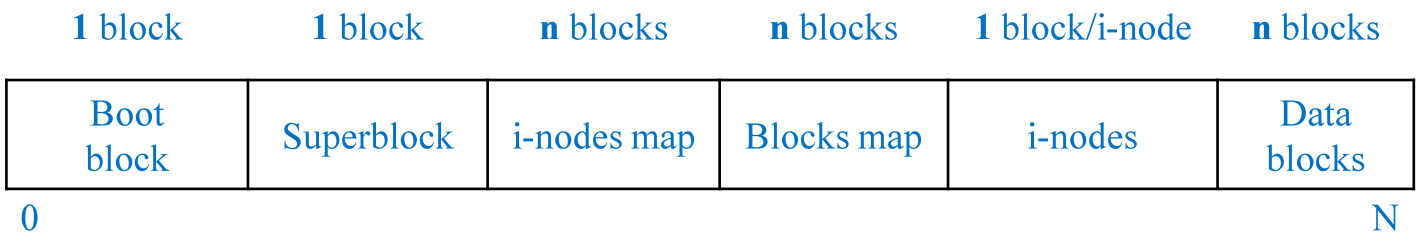
Block/cache managemen

getblk brelse

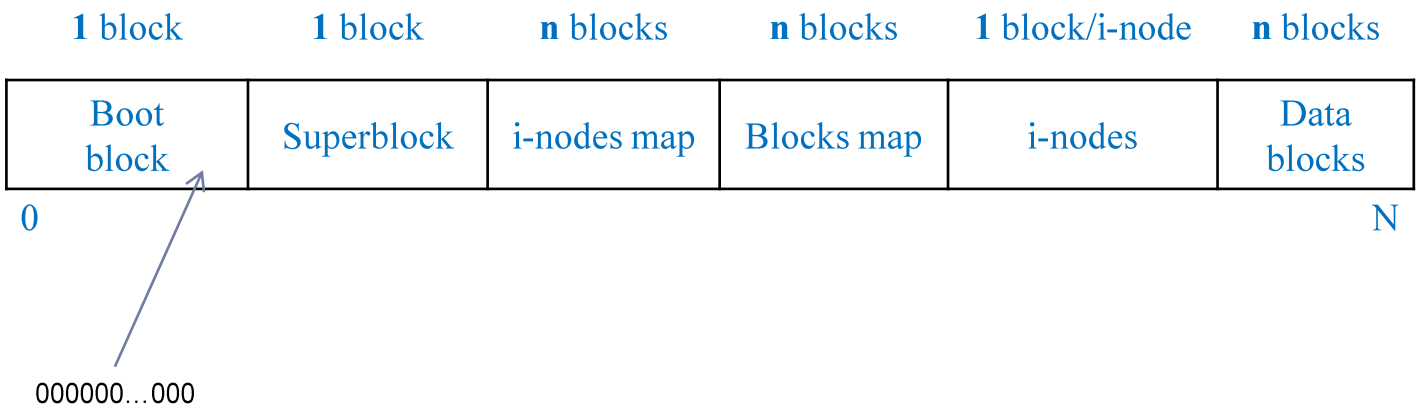
1. To design the disk layout

Disk

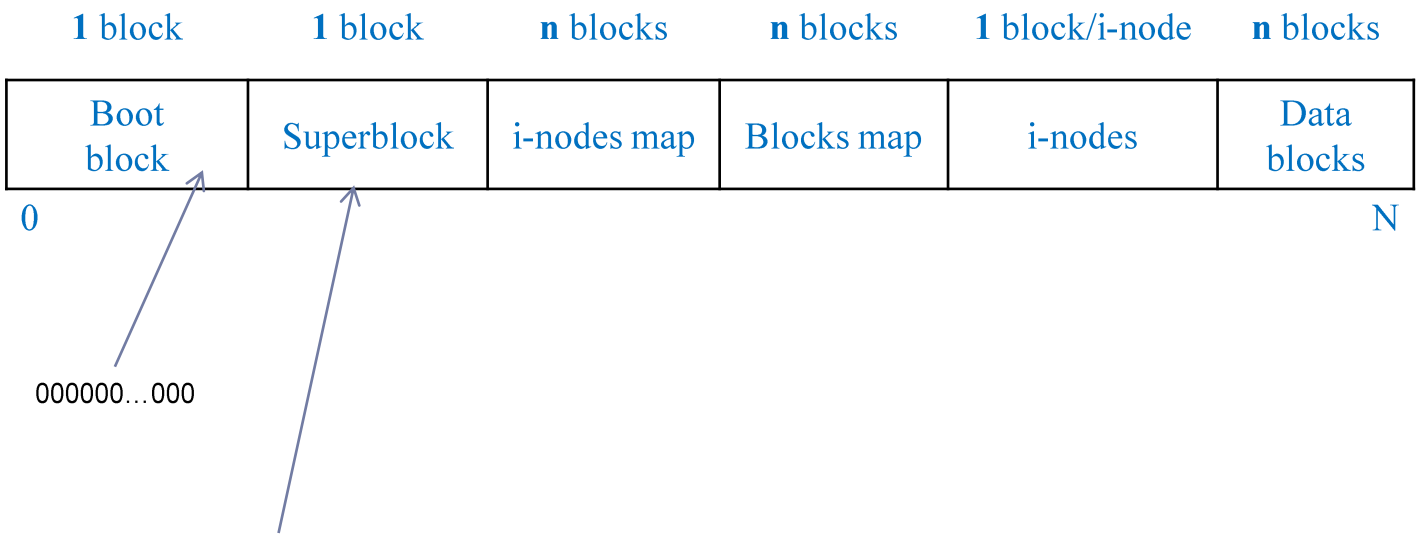
Example of disk organization



Example of disk organization

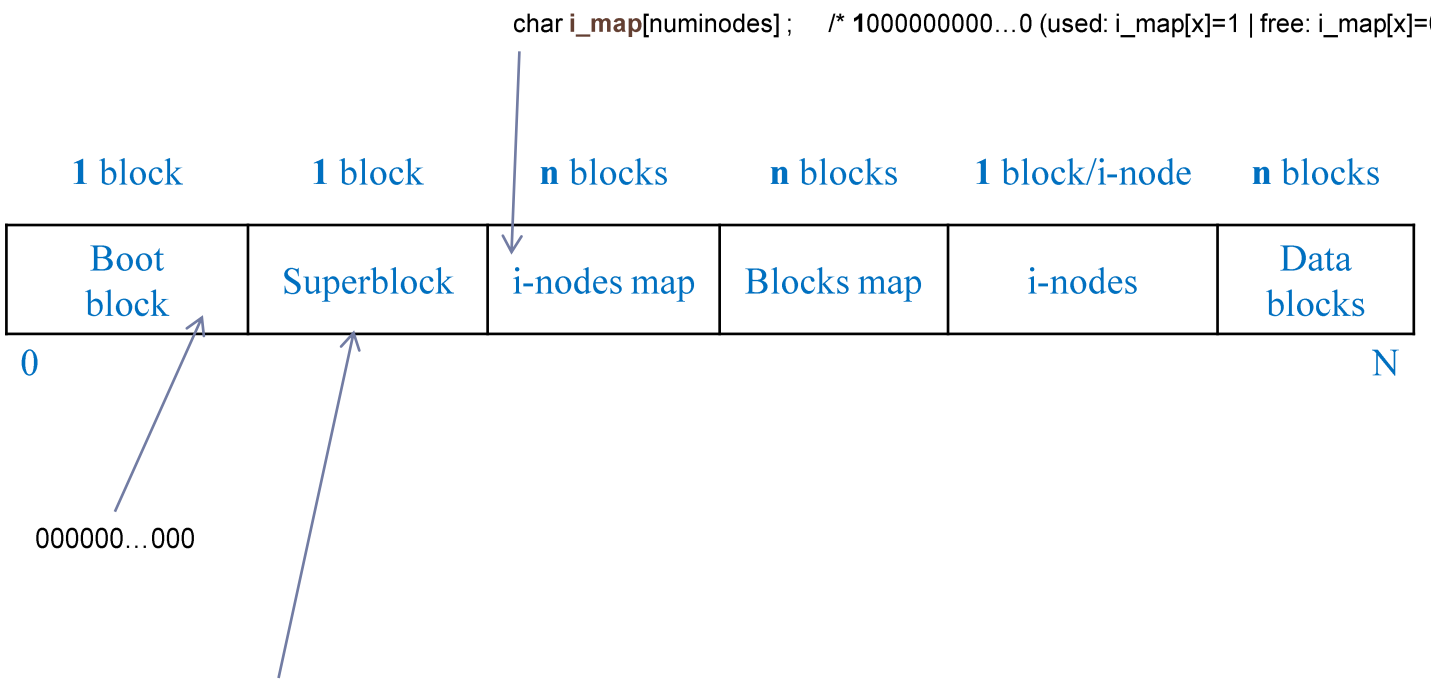


Example of disk organization



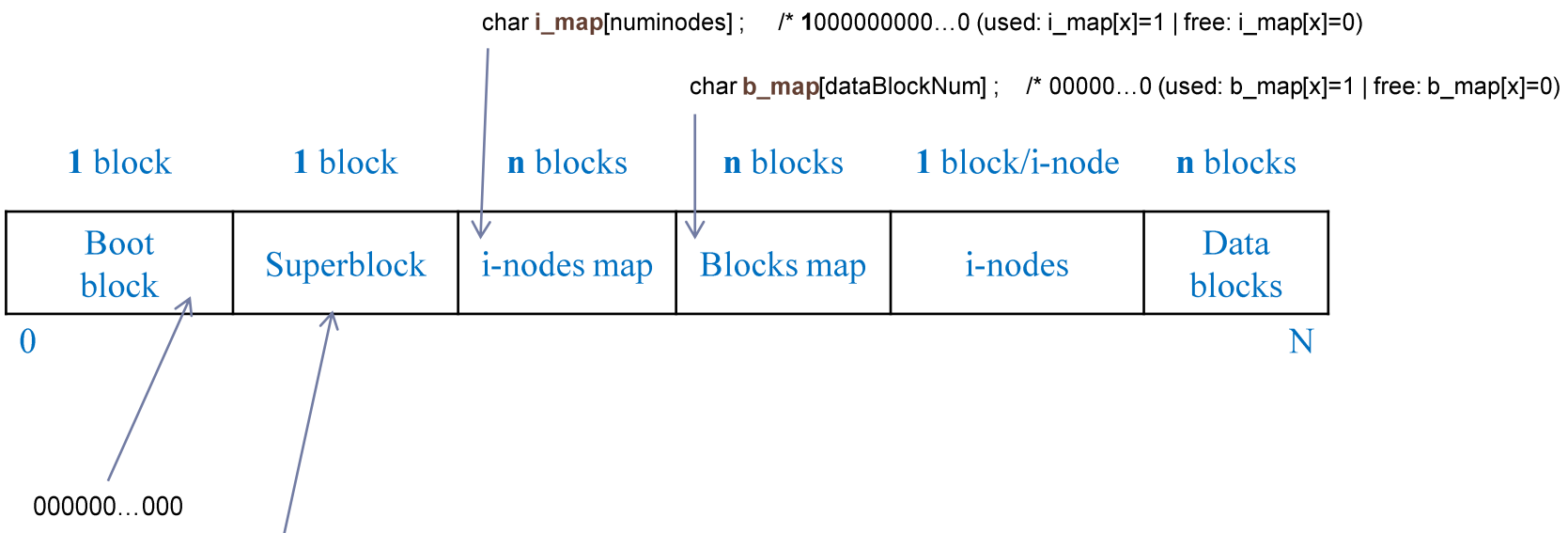

```
typedef struct {
    unsigned int magicNum;          /* Magic number of the superblock: 0x000D5500 */
    unsigned int InodeMapNumBlocks; /* Number of blocks of the i-node map */
    unsigned int DataMapNumBlock;   /* Number of blocks of the data map */
    unsigned int numinodes;         /* Number of i-nodes in the device */
    unsigned int firstinode;        /* Number of the 1st i-node in the device. (root inode) */
    unsigned int dataBlockNum;      /* Number of data blocks in the device. */
    unsigned int firstDataBlock;    /* Number of the 1st data block */
    unsigned int deviceSize;        /* Total disk space. (in bytes) */
    char padding[992];              /* Padding field (to complete a block) */
} superblock_t;
```

Example of disk organization



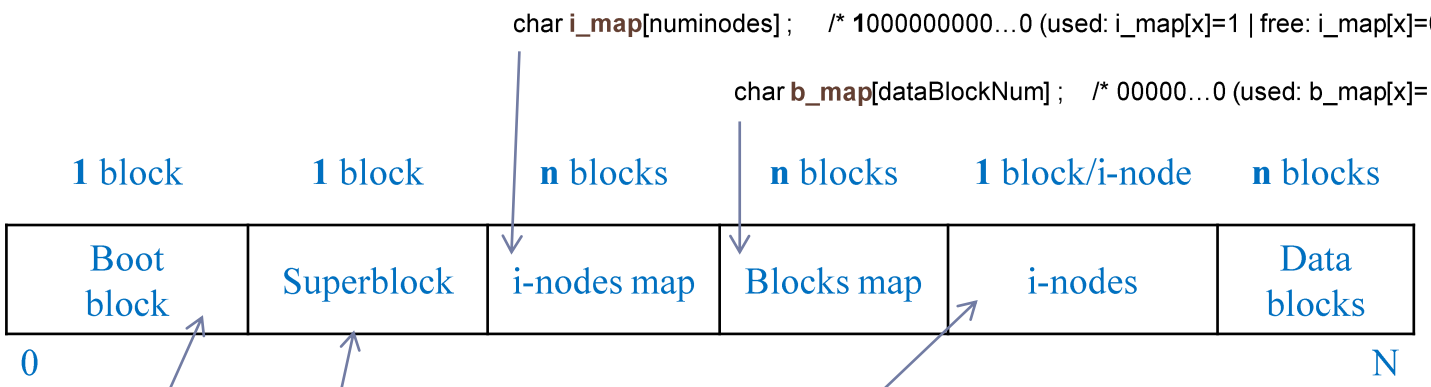

```
typedef struct {
    unsigned int magicNum;              /* Magic number of the superblock: 0x000D5500 */
    unsigned int InodeMapNumBlocks;     /* Number of blocks of the i-node map */
    unsigned int DataMapNumBlock;       /* Number of blocks of the data map */
    unsigned int numinodes;             /* Number of i-nodes in the device */
    unsigned int firstinode;            /* Number of the 1st i-node in the device. (root inode) */
    unsigned int dataBlockNum;          /* Number of data blocks in the device. */
    unsigned int firstDataBlock;        /* Number of the 1st data block */
    unsigned int deviceSize;            /* Total disk space. (in bytes) */
    char padding[992];                  /* Padding field (to complete a block) */
} superblock_t;
```

Example of disk organization




```
typedef struct {
   unsigned int magicNum;          /* Magic number of the superblock: 0x000D5500 */
   unsigned int InodeMapNumBlocks; /* Number of blocks of the i-node map */
   unsigned int DataMapNumBlock;   /* Number of blocks of the data map */
   unsigned int numinodes;         /* Number of i-nodes in the device */
   unsigned int firstinode;        /* Number of the 1st i-node in the device. (root inode) */
   unsigned int dataBlockNum;      /* Number of data blocks in the device. */
   unsigned int firstDataBlock;    /* Number of the 1st data block */
   unsigned int deviceSize;        /* Total disk space. (in bytes) */
   char padding[992];              /* Padding field (to complete a block) */
} superblock_t;
```

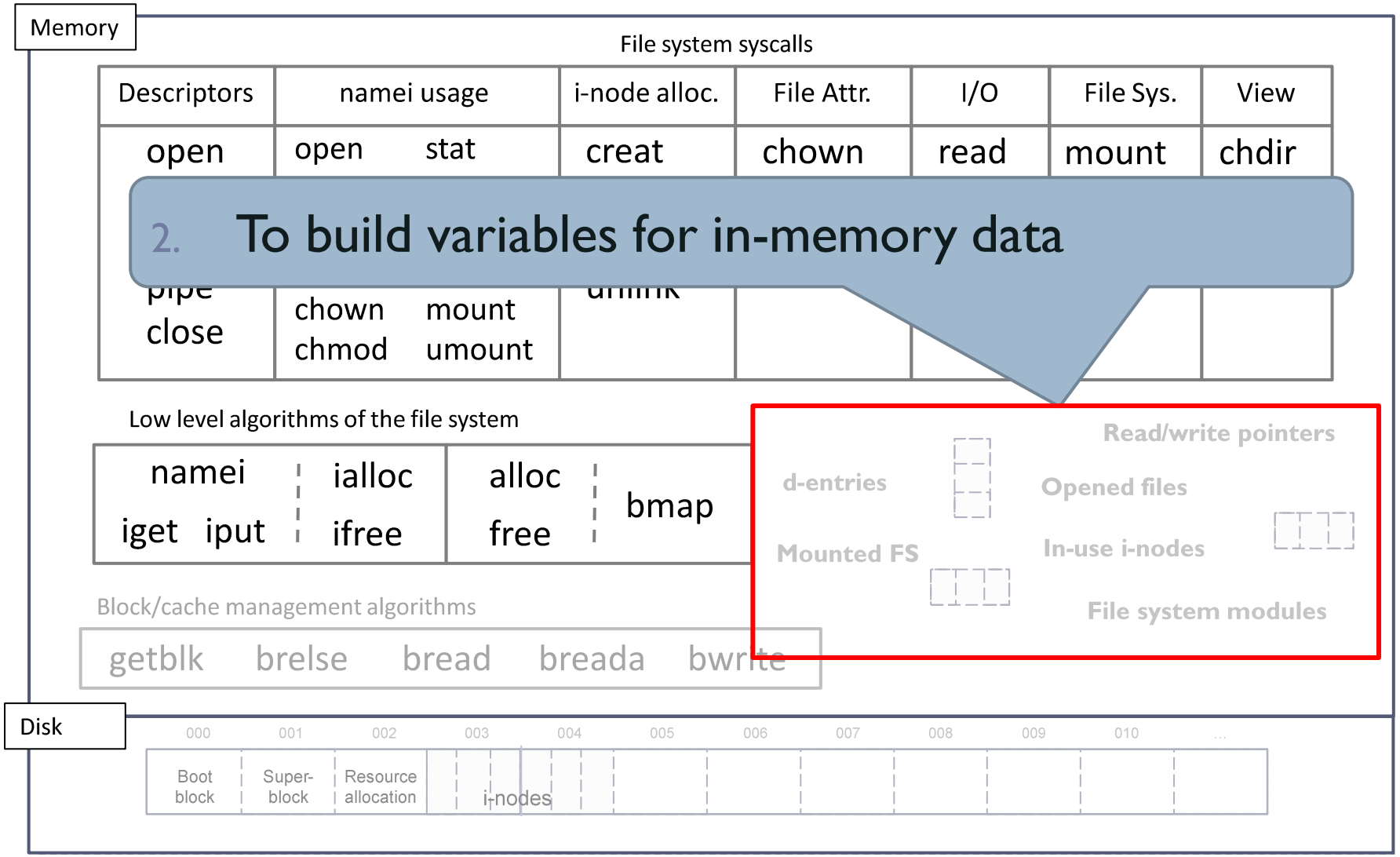
Example of disk organization




```
typedef struct {
    unsigned int type;                  /*  FILE_T o DIRECTORY_T */
    char name[200];                     /* file/directory name */
    unsigned int inodesContent[200];    /* type==dir: directory i-node list */
    unsigned int size;                  /* Current file size in bytes */
    unsigned int directBlock;           /* Direct block number */
    char padding[PADDING_inode];        /* Padding field to fill a block*/
} inode_t;
```

```
typedef struct {
    unsigned int magicNum;              /* Magic number of the superblock: 0x000D5500 */
    unsigned int InodeMapNumBlocks;     /* Number of blocks of the i-node map */
    unsigned int DataMapNumBlock;       /* Number of blocks of the data map */
    unsigned int numinodes;             /* Number of i-nodes in the device */
    unsigned int firstinode;            /* Number of the 1st i-node in the device. (root inode) */
    unsigned int dataBlockNum;          /* Number of data blocks in the device. */
    unsigned int firstDataBlock;        /* Number of the 1st data block */
    unsigned int deviceSize;            /* Total disk space. (in bytes) */
    char padding[992];                  /* Padding field (para completar un bloque) */
} superblock_t;
```

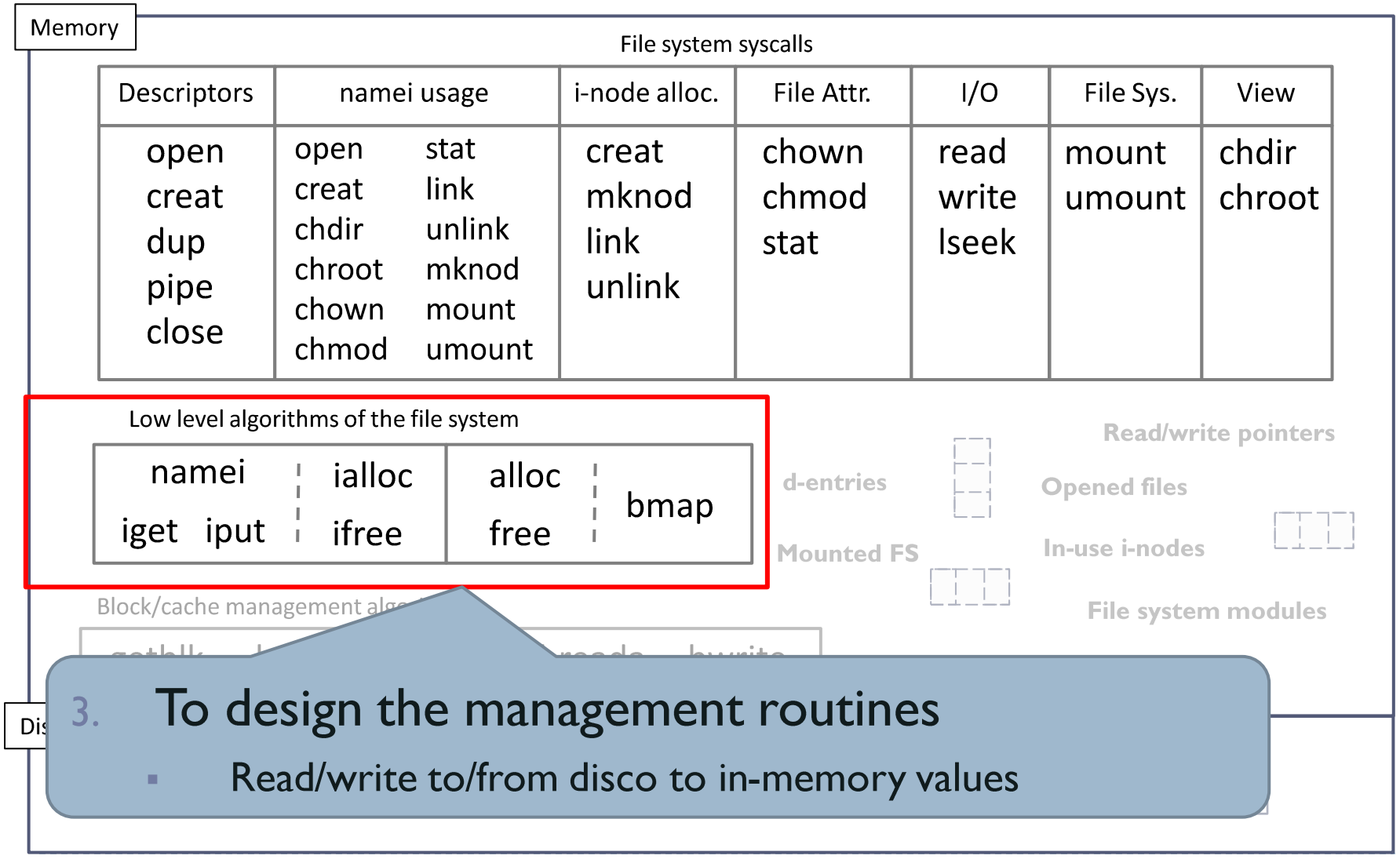
General approach



Example of variables…

```
// Information read from disk
superblock_t  sblocks [1] ;
char i_map [numinode] ;
char b_map [dataBlockNum] ;
indode_t inodes [numinode] ;

// Additional in-memory Information
struct {
    int position;
    int opened;
} inodes_x [numinode] ;

…
```

General approach



3. To design the management routines
   - Read/write to/from disco to in-memory values

Example: ialloc y alloc

```
int ialloc ( void )
{
   // to search for a free i-node
   for (int=0; i<sblocks[0].numinodes; i++)
   {
        if (i_map[i] == 0) {
           // i-node busy right now
           i_map[i] = 1;
           // default values for the i-node
           memset(&(inodes[i]),0,
                      sizeof(indode_t));
           // return the i-node indentification
           return i;
        }
   }

   return -1;
}
```

```
int alloc ( void )
{
   char b[BLOCK_SIZE];

   for (int=0; i<sblocks[0].dataBlockNum; i++)
   {
        if (b_map[i] == 0) {
           // busy block right now
           b_map[i] = 1;
           // default values for the block
           memset(b, 0, BLOCK_SIZE);
           bwrite(DISK, i+sblocks[0].firstDataBlock, b);
           // it returns the block id
           return i;
        }
   }
   return -1;
}
```

Example: ifree y free

```
int ifree ( int inode_id )
{
    // to check the inode_id vality
    if (inode_id > sblocks[0].numinodes)
        return -1;

    // free i-node
    i_map[inode_id] = 0;

    return 0;
}
```

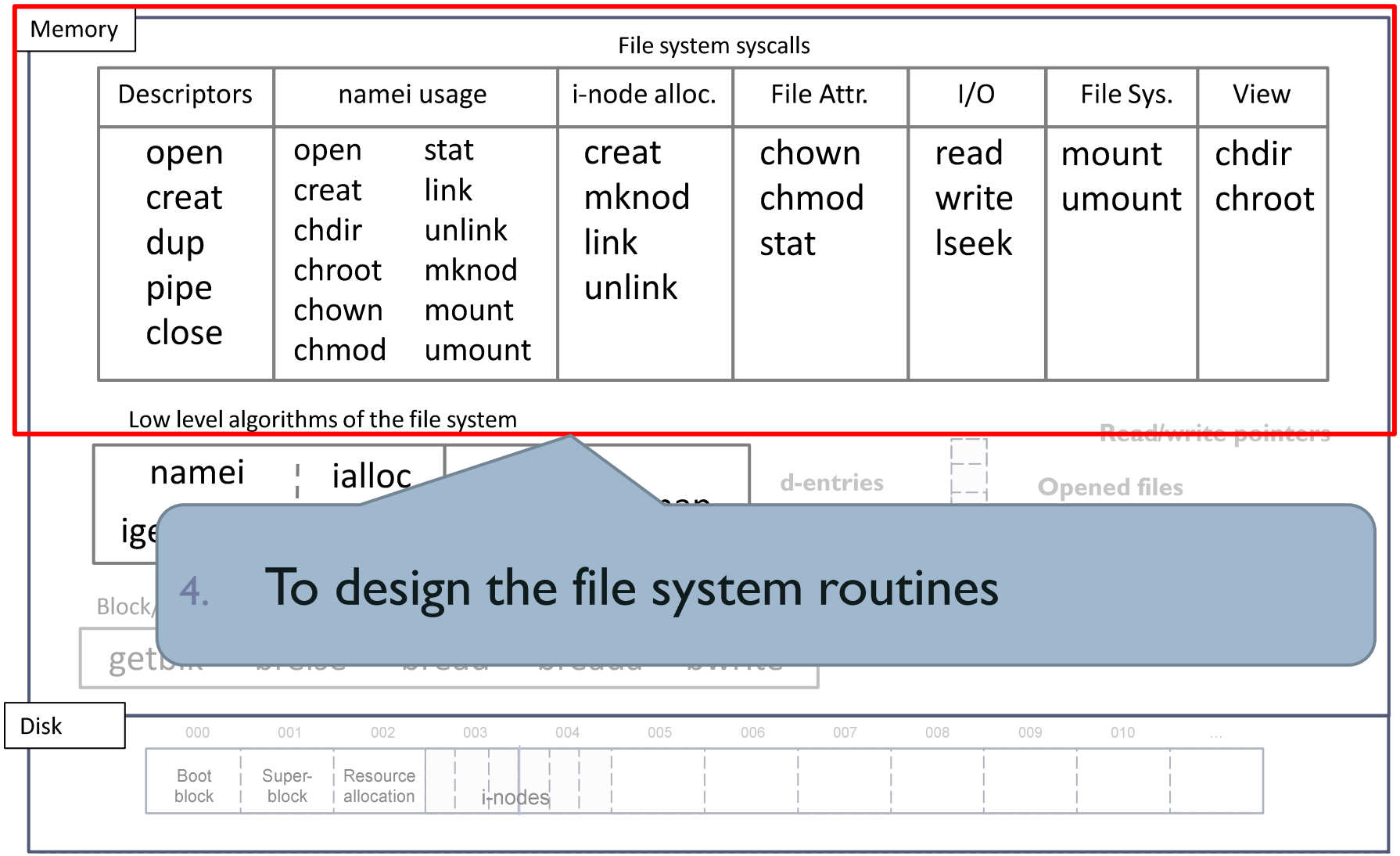
```
int free ( int block_id )
{
    // to check inode_id the vality
    if (block_id > sblocks[0].dataBlockNum)
        return -1;

    // free block
    b_map[block_id] = 0;

    return 0;
}
```

Example: namei y bmap

```
int namei ( char *fname )
{
   // seek for the i-node with name <fname>
   for (int=0; i<sblocks[0].numinodes; i++)
   {
        if (! strcmp(inodes[i].name, fname))
             return i;
   }

    return -1;
}
```

```
int bmap ( int inode_id, int offset )
{
    // check for if it is a valid i-node ID
    if (inode_id > sblocks[0].numinodes)
        return -1;

    // return the i-node block
    if (offset < BLOCK_SIZE)
        return inodes[inode_id].directBlock;

    return -1;
}
```

General approach



mount

```
typedef struct {
    unsigned int magicNum;              /* Magic number of the superblock: 0x000D5500 */
    unsigned int InodeMapNumBlocks;     /* Number of blocks of the i-node map */
    unsigned int DataMapNumBlock;       /* Number of blocks of the data map */
    unsigned int numinodes;             /* Number of i-nodes in the device */
    unsigned int firstinode;            /* Number of the 1st i-node in the device. (root inode) */
    unsigned int dataBlockNum;          /* Number of data blocks in the device. */
    unsigned int firstDataBlock;        /* Number of the 1st data block */
    unsigned int deviceSize;            /* Total disk space. (in bytes) */
    char padding[992];                  /* Padding field (to complete a block) */
} superblock_t;
```

```
int mount ( void )
{
    // Read disk block 1 and store it into sblocks[0]
    bread(DISK, 1, &(sblocks[0]) );

    // Read from disk i-node map
    for (int i=0; i<sblocks[0].InodeMapNumBlocks; i++)
          bread(DISK, 2+i, ((char *)i_map + i*BLOCK_SIZE) ;

    // Read from disk block map
    for (int i=0; i<sblocks[0].DataMapNumBlock; i++)
         bread(DISK, 2+i+sblocks[0].InodeMapNumBlocks, ((char *)b_map + i*BLOCK_SIZE);

    // Read i-nodes from disk
    for (int i=0; i<(sblocks[0].numinodes*sizeof(inode_t)/BLOCK_SIZE); i++)
         bread(DISK, i+sblocks[0].firstinode, ((char *)inodes + i*BLOCK_SIZE);

    return 1;
}
```

sync

```
typedef struct {
    unsigned int magicNum;              /* Magic number of the superblock: 0x000D5500 */
    unsigned int InodeMapNumBlocks;     /* Number of blocks of the i-node map */
    unsigned int DataMapNumBlock;       /* Number of blocks of the data map */
    unsigned int numinodes;             /* Number of i-nodes in the device */
    unsigned int firstinode;            /* Number of the 1st i-node in the device. (root inode) */
    unsigned int dataBlockNum;          /* Number of data blocks in the device. */
    unsigned int firstDataBlock;        /* Number of the 1st data block */
    unsigned int deviceSize;            /* Total disk space. (in bytes) */
    char padding[992];                  /* Padding field (to complete a block) */
} superblock_t;
```

```
int sync ( void )
{
    // Write block 1 from sblocks[0] into disk
    bwrite(DISK, 1, &(sblocks[0]) );

    // Write i-node map to disk
    for (int i=0; i<sblocks[0].InodeMapNumBlocks; i++)
          bwrite(DISK, 2+i, ((char *)i_map + i*BLOCK_SIZE) ;

    // Write block map to disk
    for (int i=0; i<sblocks[0].DataMapNumBlock; i++)
         bwrite(DISK, 2+i+sblocks[0].InodeMapNumBlocks, ((char *)b_map + i*BLOCK_SIZE);

    // Write i-nodes to disk
    for (int i=0; i<(sblocks[0].numinodes*sizeof(diskInodeType)/BLOCK_SIZE); i++)
         bwrite(DISK, i+sblocks[0].firstinode, ((char *)inodes + i*BLOCK_SIZE);

    return 1;
}
```

umount

```
typedef struct {
    unsigned int magicNum;              /* Magic number of the superblock: 0x000D5500 */
    unsigned int InodeMapNumBlocks;     /* Number of blocks of the i-node map */
    unsigned int DataMapNumBlock;       /* Number of blocks of the data map */
    unsigned int numinodes;             /* Number of i-nodes in the device */
    unsigned int firstinode;            /* Number of the 1st i-node in the device. (root inode) */
    unsigned int dataBlockNum;          /* Number of data blocks in the device. */
    unsigned int firstDataBlock;        /* Number of the 1st data block */
    unsigned int deviceSize;            /* Total disk space. (in bytes) */
    char padding[992];                  /* Padding field (to complete a block) */
} superblock_t;
```

```
int umount ( void )
{
    // make sure that all files are closed
    for (int=0; i<sblocks[0].numinodes; i++) {
        if (inodes_x[i].opened == 1) {
            return -1;
        }
    }

    // flush metadata on disk
    sync();
    return 0;
}
```

mkfs

```
typedef struct {
    unsigned int magicNum;              /* Magic number of the superblock: 0x000D5500 */
    unsigned int InodeMapNumBlocks;     /* Number of blocks of the i-node map */
    unsigned int DataMapNumBlock;       /* Number of blocks of the data map */
    unsigned int numinodes;             /* Number of i-nodes in the device */
    unsigned int firstinode;            /* Number of the 1st i-node in the device. (root inode) */
    unsigned int dataBlockNum;          /* Number of data blocks in the device. */
    unsigned int firstDataBlock;        /* Number of the 1st data block */
    unsigned int deviceSize;            /* Total disk space. (in bytes) */
    char padding[992];                  /* Padding field (to complete a block) */
} superblock_t;
```

```
int mkfs ( void )
{
    // setup with default values the superblock, maps, and i-nodes
    sblocks[0].magicNum = 1234;
    sblocks[0].numinodes = 50;
    …
    for (int=0; i<sblocks[0].numinodes; i++)
          i_map[i] = 0; // free
    for (int=0; i<sblocks[0].dataBlockNum; i++)
          b_map[i] = 0; // free
    for (int=0; i<sblocks[0].numinodes; i++)
         memset(&(inodes[i]), 0, sizeof(diskInodeType) );

    // to write the default file system into disk
    umount();

    return 0;
}
```

open and close

```
int open ( char *name )
{
    int inode_id ;

    inode_id = namei(name) ;
    if (inode_id < 0)
        return inode_id ;

    inodes_x[inode_id].position = 0;
    inodes_x[inode_id].opened   = 1;

    return inode_id;
}
```

```
int close ( int fd )
{

    if (fd < 0)
        return -1;

    inodes_x[fd].position = 0;
    inodes_x[fd].opened   = 0;

    return 0;
}
```

creat and unlink

```
int creat ( char *name )
{
    int b_id, inode_id ;

    inode_id = ialloc() ;
    if (inode_id < 0) { return inode_id ; }
    b_id = alloc();
    if (b_id < 0) { ifree(inode_id); return b_id ; }

    inodes[inode_id].type= 1 ; // FILE
    strcpy(inodes[inode_id].name, name);
    inodes[inode_id].directBlock = b_id ;
    inodes_x[inode_id].position = 0;
    inodes_x[inode_id].opened   = 1;

    return 0;
}
```

```
int unlink ( char * name )
{
    int inode_id ;

    inode_id = namei(name) ;
    if (inode_id < 0)
        return -1;

    free(inodes[inode_id].directBlock);
    memset(&(inodes[inode_id]),
                0,
                sizeof(diskInodeType));
    ifree(inode_id) ;

   return 0;
}
```

read and write

```
int read ( int fd, char *buffer, int size )
{
   char b[BLOCK_SIZE] ;
   int b_id ;

   if (inodes_x[fd].position+size > inodes[fd].size)
      size = inodes[fd].size - inodes_x[fd].position;
   if (size =< 0)
      return -1;

   b_id = bmap(fd, inodes_x[fd].position);
   bread(DISK, b_id, b);
   memmove(buffer,
                b+inodes_x[fd].position,
                size);
   inodes_x[fd].position += size;

   return size;
}
```

```
int write ( int fd, char *buffer, int size )
{
   char b[BLOCK_SIZE] ;
   int b_id ;

   if (inodes_x[fd].position+size > BLOCK_SIZE)
      size = BLOCK_SIZE - inodes_x[fd].position;
   if (size =< 0)
      return -1;

   b_id = bmap(fd, inodes_x[fd].position);
   bread(DISK, b_id, b);
   memmove(b+inodes_x[fd].position,
                buffer, size);
   bwrite(DISK, b_id, b);
   inodes_x[fd].position += size;

   return size;
}
```

Exercise
solution

1. Initial approach:
    1. Draw a diagram of initial system state
    2. Modify the diagram to incorporate the exercise requirements
2. Answer the proposed questions
3. Review the answers

Examples of exercises
File System

ARCOS

Operating Systems Design
Degree in Computer Engineering
University Carlos III of Madrid