Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Part 3: File systems and I/O

Seminar 8

Minix file system

fSO

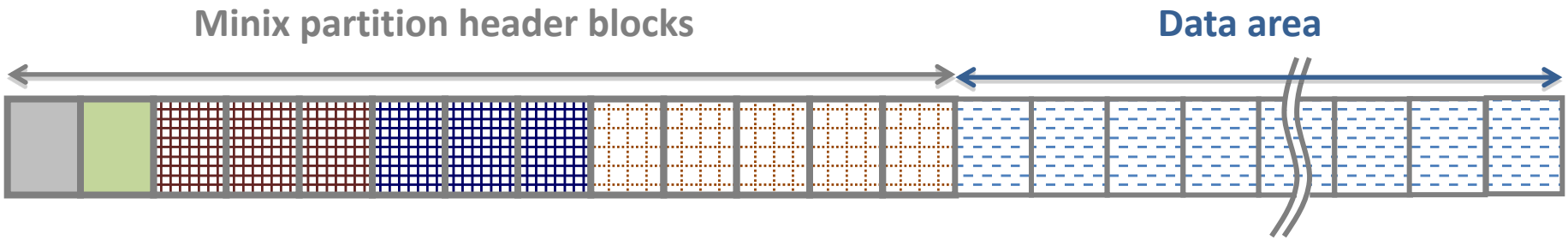
Minix file system

- **Goals**
  - To know the structure of a Minix partition
  - To know how the OS manage i-nodes to keep file information
  - To understand the bit map concept to manage free and busy space
  - To be able to locate a particular file within a directory structure from its absolute path
- **Bibliography**
  - "Operating Systems Design and Implementation" (3rd Edition), Andrew S. Tanenbaum, Prentice Hall 2006 Section 5.6

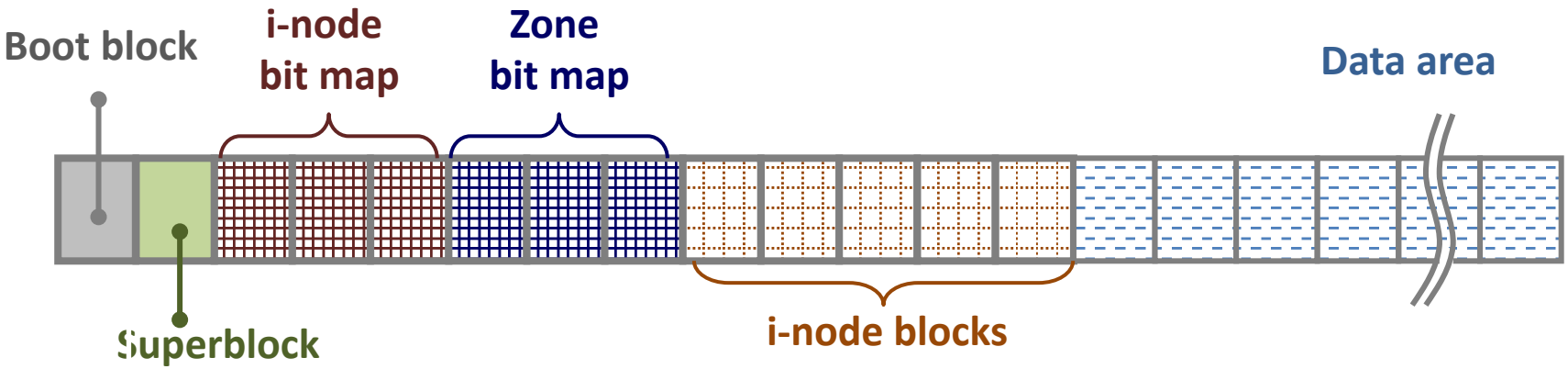
- **Partition structure**
- i-node structure
- Directory entry
- Standard sizes
- Exercises

Partition structure

- **A Minix partition** is built upon a set of fixed size blocks (i.e. 1KByte)
  - A partition structure contains:
    - A **header** made up of block groups intended to store the data structures that sustain the file system
    - **Data area** made up of blocks intended to store file data

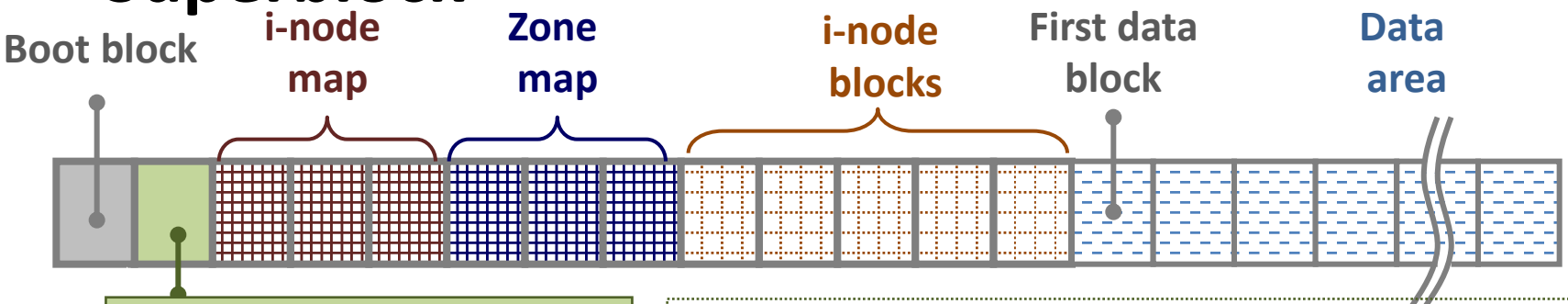


- **Header blocks**



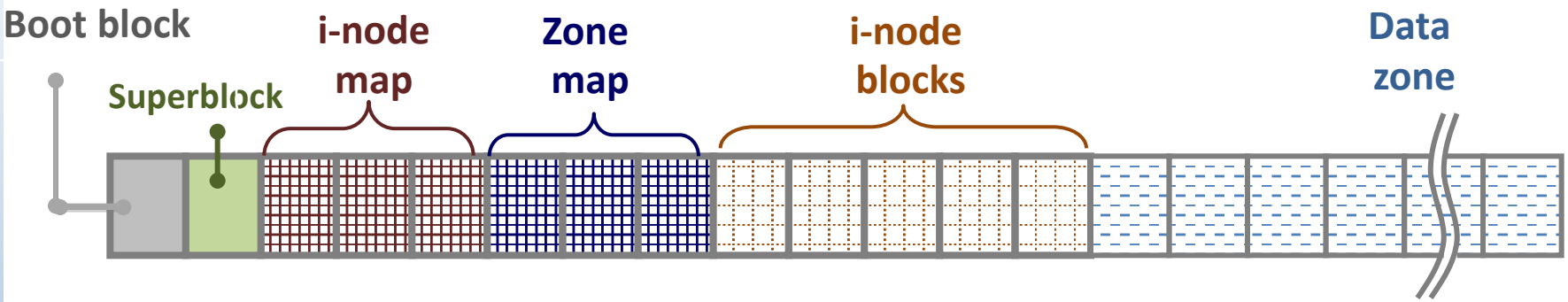
  - **Boot block:** contains the boot program that loads the operating system and transfers the control to it (Master Boot Record).
  - **Superblock:** is a data structure with the file system description that indicates the size and location of every element
  - **i-node bit map**: bit vector to manage free and allocated i-nodes. It contains one bit per i-node
  - **Zone bit map**: bit vector to manage free and allocated zones. It contains one bit per zone
  - **i-node blocks:** contains the i-node data structures. The i-node number depends on the partition size. i-node 0 is not used

- **Superblock**



	Superblock	
00	Number of i-nodes	It is fixed by the user at partition creation or it takes default values
02	Number of data zones	It is fixed at partition creation, it can be less than partition size
04	Number of i-node map blocks	[i-nodes number / Block bits number]
06	Number of zone map blocks	[Data zones number / Block bits number]
08	Block number of the first data zone	2 (boot and superblock) + number of i-node map blocks + number of zone map blocks + number of i-node blocks
0A	N (1 zone= 2^N * 1024)	1 zone = 2^N blocks N value is stored into the superblock
0C	Maximum file size	Maximum file size in bytes
10	Magic number	Numerical value that guaranties that the partition contains a MINIX file system

- **i-node/zone bit maps**



i-node map

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Zone map

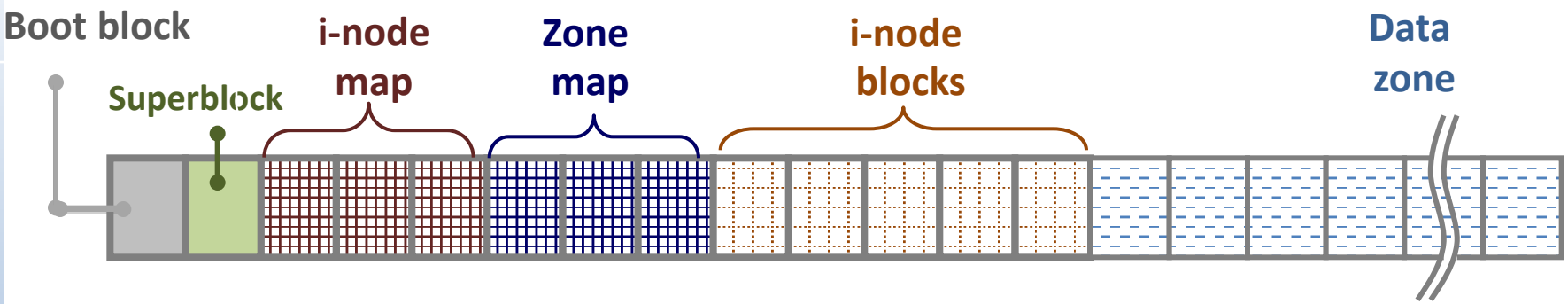
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0
1	0	1	1	1	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

0 bit -> allocated
1 bit -> free

Every **i-node** is **represented** by **one bit** equal to 0 or 1 if the i-node is allocated or free, respectively

Every **data zone** is **represented** by **one bit** equal to 0 or 1 it the zone is allocated or free, respectively

- **Data zone**



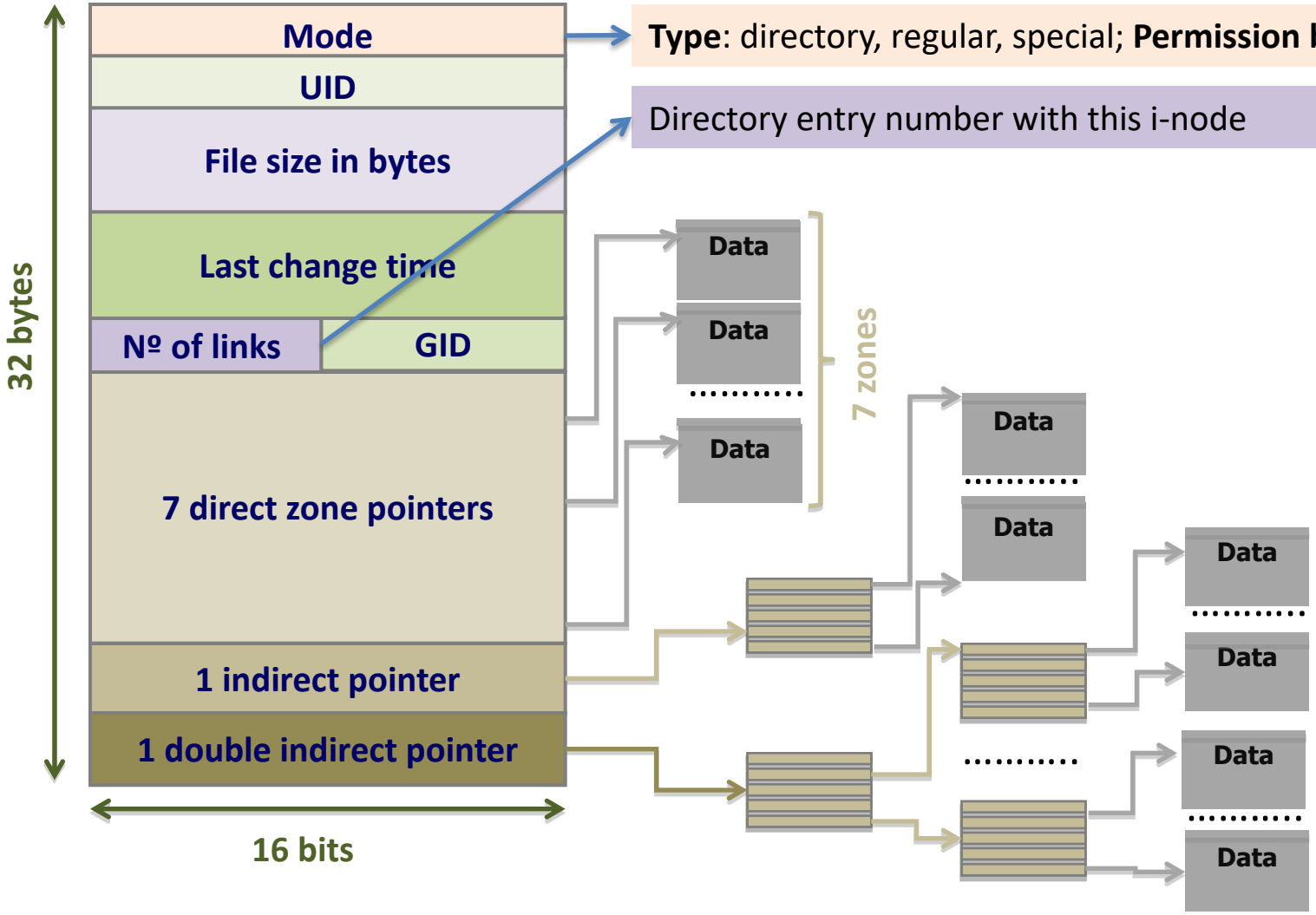
- Block set used to store regular file content, directory entries and block references
  - To be able to address big partitions, MINIX file system allows grouping blocks into zones
  - Zone is the file allocation unit
    - **1 zone = 2^N blocks** → by default 1 zone = 2^0 blocks = 1block
    - The first data block (referenced inside the superblock) is adjusted to a zone starting block

- Partition structure
- **i-node structure**
- Directory entry
- Standard sizes
- Exercises

i-node structure

– Data structure that contains all file attributes except its name

- Every file has an associated i-node
- It controls indexed allocation by means of direct, indirect and double indirect pointers



i-node structure

- Performance analysis
  - **Efficient random access:** The maximum number of disk accesses is 4 (worst case)
    - The indirect pointers are only used with big or very big files (commonly few)
    - Small file (common case) access is very efficient
  - **Reliable and elegant design**: every file has its own separated data structure

i-node structure

- Performance analysis
  - **Efficient random access:** The maximum number of disk accesses is 4 (worst case)
    - The indirect pointers are only used with big or very big files (commonly few)
    - Small file (common case) access is very efficient
  - **Reliable and elegant design**: every file has its own separated data structure

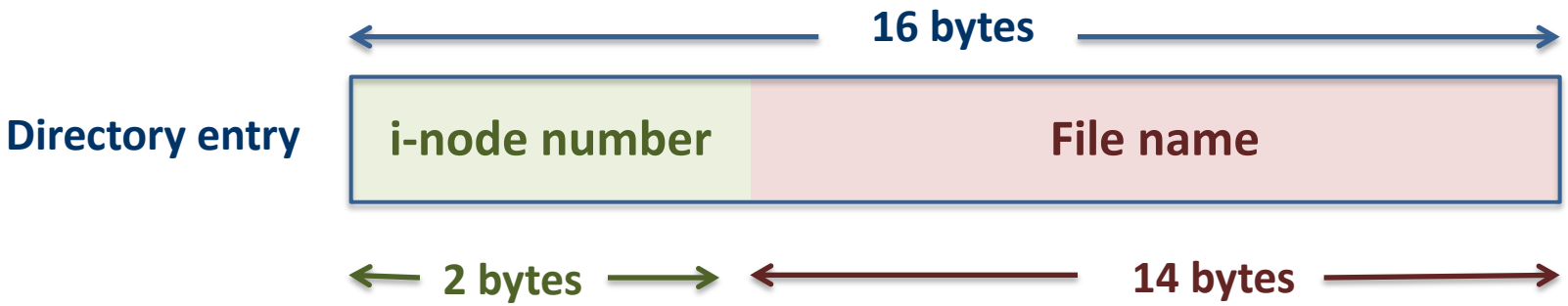
Starting point : i-node number.

1. **Read the block were the i-node is located**
2. **Read the block pointed by 1 indirect double pointer**
3. **Read the block pointed by the 2.**
4. **Read the data block**

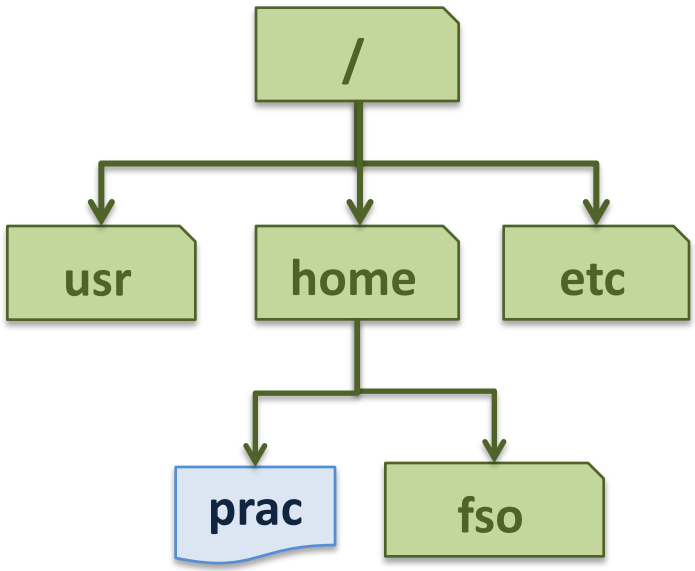
- Partition structure
- i-node structure
- **Directory entry**
- Standard sizes
- Exercises

Directory entry

- **Minix directories**
  - Directory structure as a directed acyclic graph (DAG)
  - Directories are files which content is interpreted as registers → directory entries (also named links)
- **Minix directory entry or link**
  - It has a 16 byte size
    - 2 bytes for the i-node
    - 14 bytes for the file name



- **Directory entry**
  - When a directory is created, the **entries '.' and '..'** are automatically created
  - **i-node 1 describes the root directory**
  - When a directory entry is removed it is marked with i-node 0



1	.
1	..
3	usr
4	home
5	etc

4	.
1	..
35	prac
84	fso

84	.
4	..

3	.
1	..

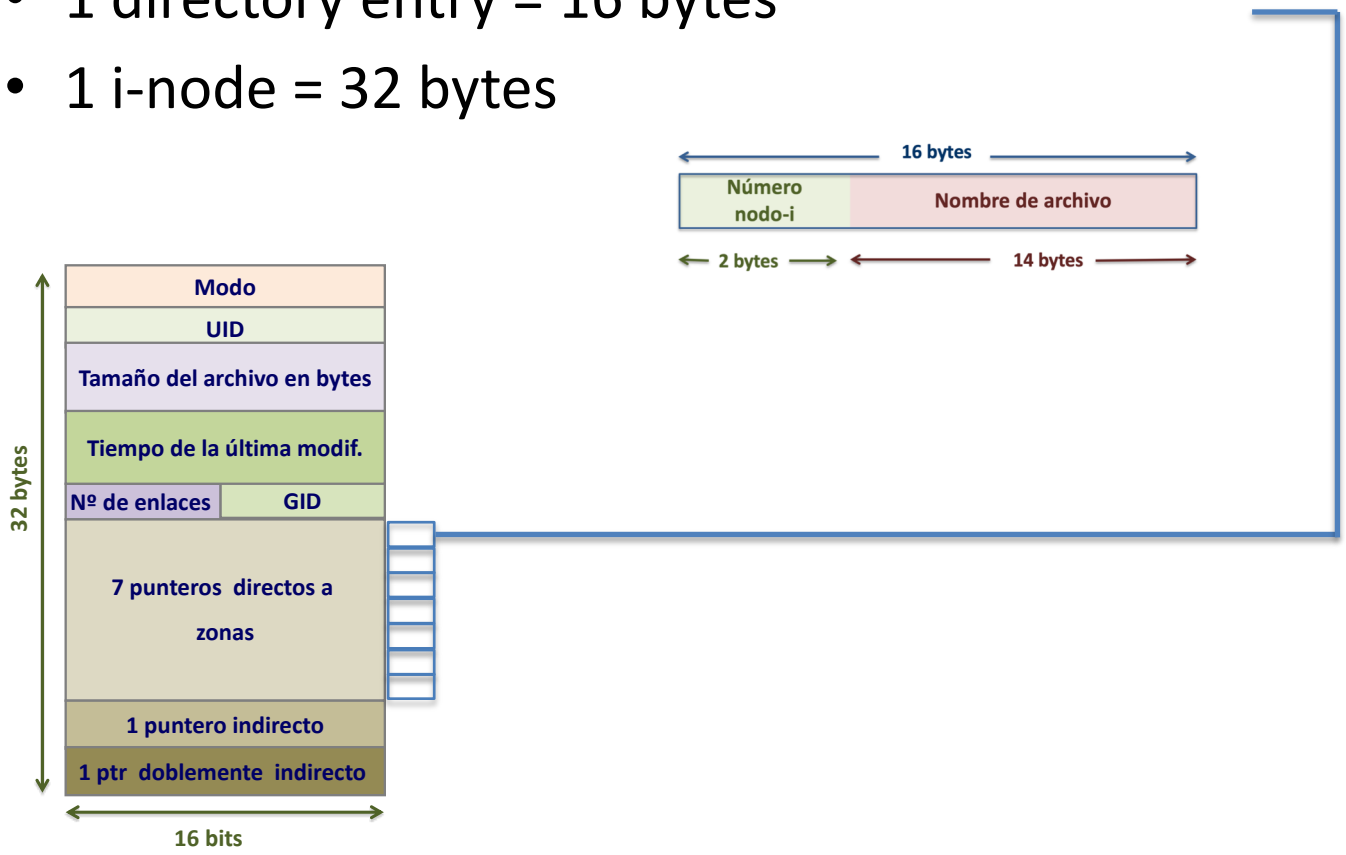
5	.
1	..

Content

- Partition structure
- i-node structure
- Directory entry
- **Standard sizes**
- Exercises

Standard Sizes

- Default sizes for Minix elements
  - N = 0; 1 Zone =2^0 blocks = 1024 bytes
  - 1 pointer to zone or block = 2 bytes = 16 bits
  - 1 directory entry = 16 bytes
  - 1 i-node = 32 bytes

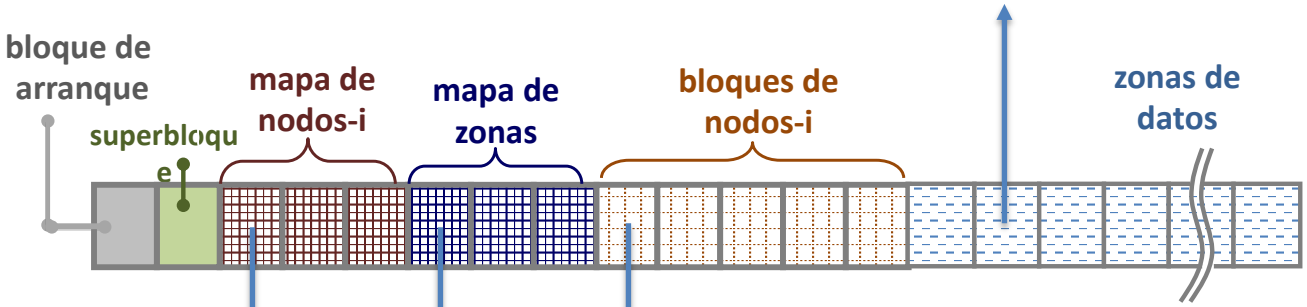


Standard sizes

- Some infered values

Number of references to block in 1 block

Number of directory entries in 1 block



Number of i-nodes in 1 block

Number of zones mapped in 1 block

Number of inode mapped in 1 block

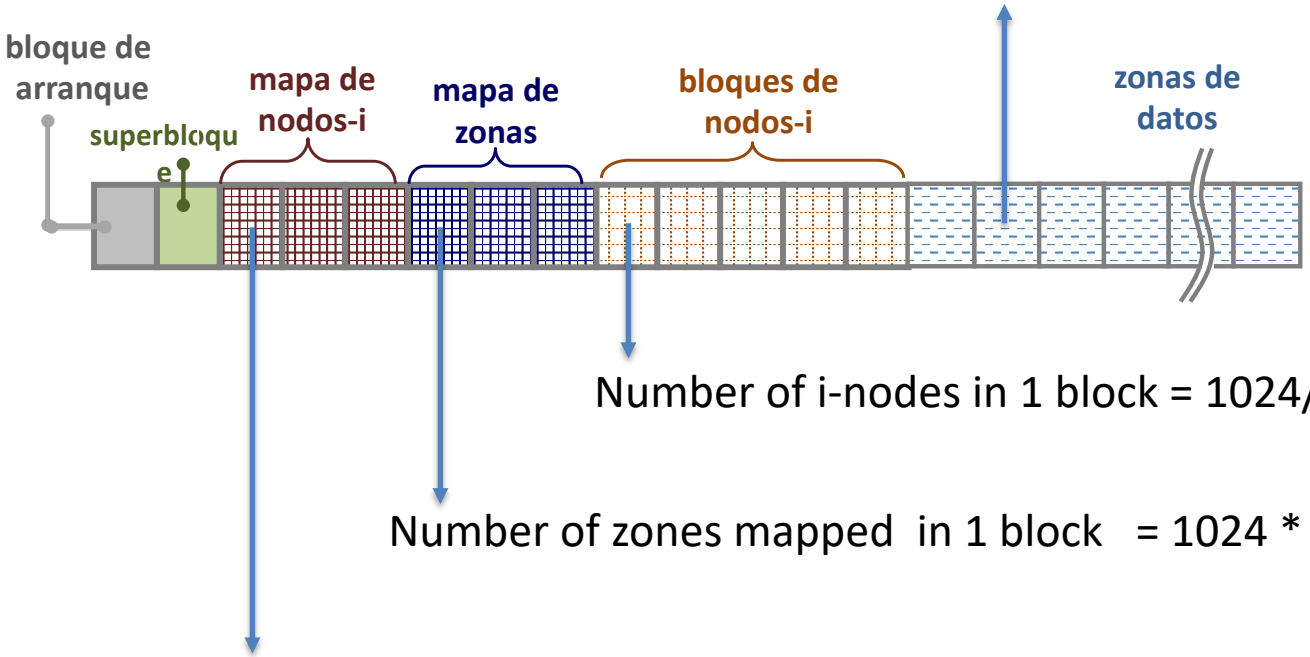
In which block is located i-node 213?
Which position in the block?

Standard sizes

- Some infered values

Number of references to block in 1 block = 1024/2 = $2^{10}/2^1 = 2^9 = 512$

Number of directory entries in 1 block = 1024/16 = $2^{10}/2^4 = 2^6 = 64$



Number of i-nodes in 1 block = 1024/32 = $2^{10}/2^5 = 2^5 = 32$

Number of zones mapped in 1 block = 1024 * 8 = $2^{10}\,2^3 = 2^{13} = 8192$

Number of inode mapped in 1 block = 1024 * 8 = $2^{10}\,2^3 = 2^{13} = 8192$

In which block is located i-node 213? **Floor**(213/32) = 6 + (1 + NBMN + NBMZ)

Which position in the block? 213 % 32 = 21

- Partition structure
- i-node structure
- Directory entry
- Standard sizes
- **Exercises**

- **Exercise 1: Maximum size of a Minix file**

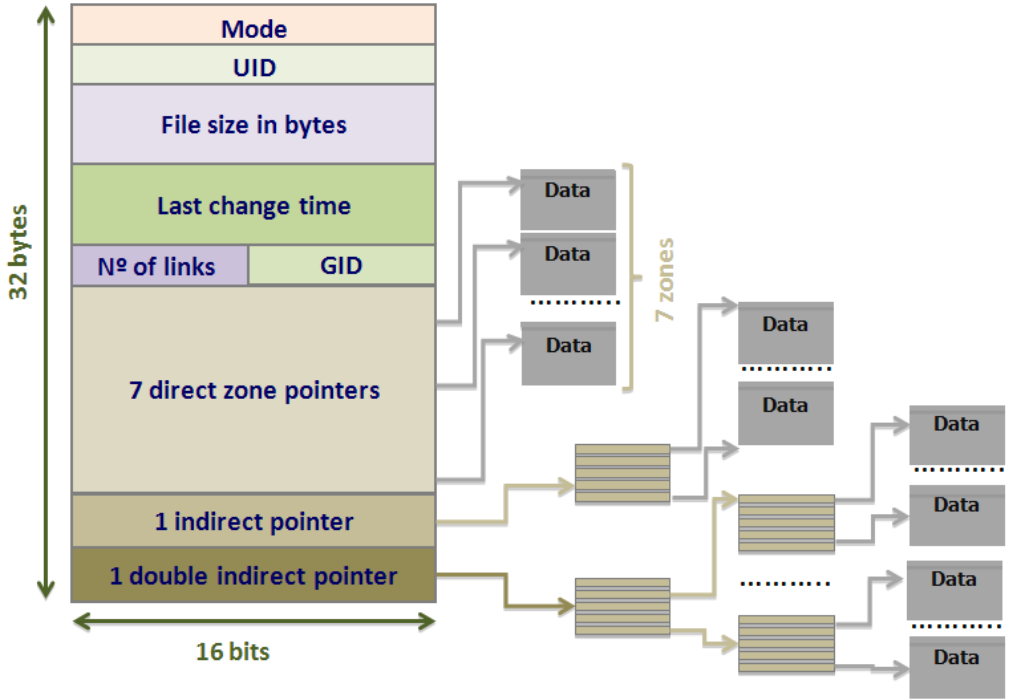
  Obtain the maximum theoretical size of a Minix file. Specify the maximum size taking into account the device size (64 MB). Specify the addressed blocks by every pointer type. The Minix parameters considered are:

  16 bit data zone pointers

  1Kbyte block size

  1 zone = 1 block

  i-node structure: 7 direct, 1 indirect and 1 double indirect pointers



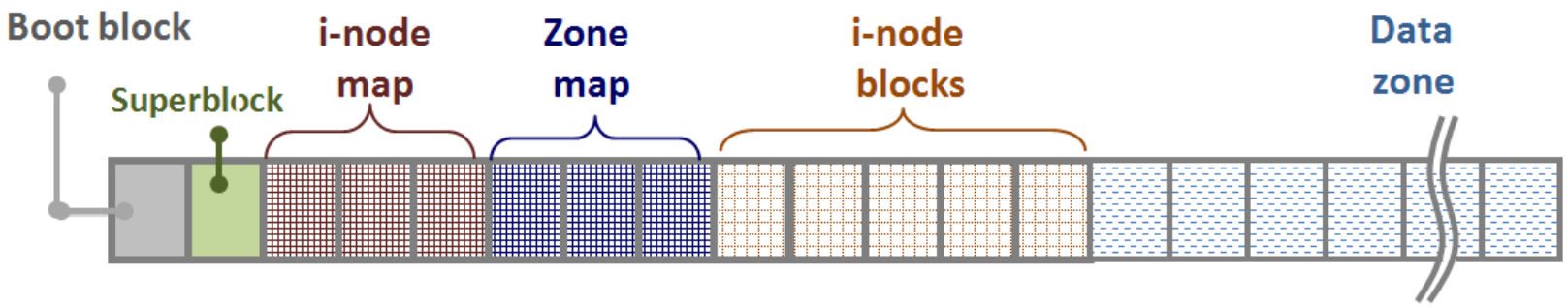
- **Exercise 2: Minix partition structure**

  Given a Minix partition in a 20 Mbyte Disk with the following parameters:

  - 16 bit data zone pointers
  - 1 Kbyte blocks and 1 zone = 1 block
  - 32 byte i-node size
  - Maximum number of i-nodes: 512

  a) Specify all the file system data structures and the blocks number that every one requires

  b) In case of zone bit map fault think about how to reconstruct it from the information available in the other (error free) file system structures



- **Exercise 2: Minix partition structure**

  Given a Minix partition in a 20 Mbyte Disk with the following parameters:

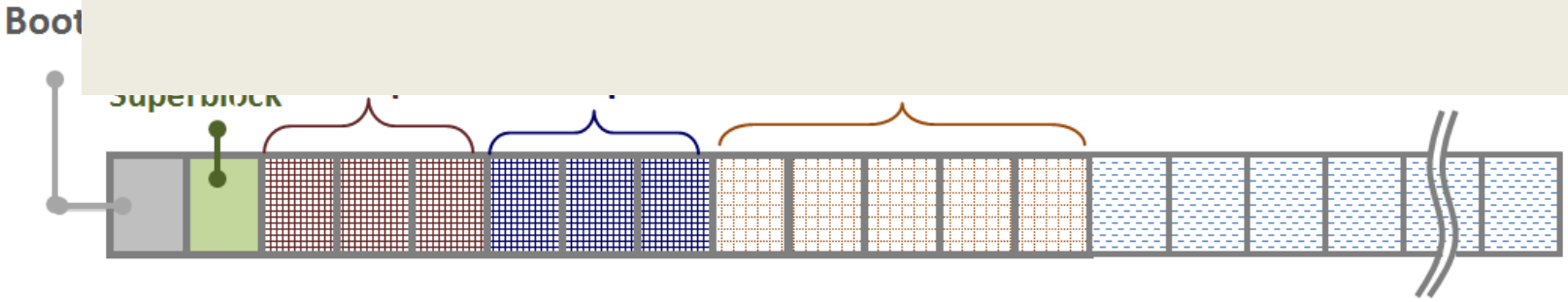
  - 16 bit data zone pointers

a)

- 1 Boot block
- 1 super block
- Ceiling(512 bits i-nodes, 1 block = 1024 * 8) = 1 block
- Disk data Zone = 20MB => 20*1024*1024/1024 =
  Ceiling(20* 1024 bits, 8*1024 bits) = 3

b)

- Zones i-nodes, we need 512 i-nodes
  - Ceiling(512 * 32 Bytes , 1024 * 1 byte) = 16
- Data zones: 20*1024*1024 - (1 + 1 + 1 + 3 + 16) = 20971501 zones



- **Exercise 3: Looking for a file inside a directory**

  Consider a Minix file system created with standard sizes which actual directory entries are the following:

1	.
1	..
3	usr
4	home
5	etc

4	.
1	..
35	prac
84	fso

3	.
1	..
60	ut12
61	sut12

84	.
4	..
90	map
91	listado

5	.
1	..

90	.
84	..

a) Draw the directory and file tree that corresponds to this system

b) Describe **what i-node numbers and how many blocks** are accessed to read the first 128 bytes in file **/home/fso/listado**

c) What information should we get when reading the first 32 bytes in file **/home/fso/map**

- **Exercise 3: Looking for a file inside a directory**

  Consider a Minix file system created with standard sizes which actual directory entries are the following:

1	.
1	..
3	usr
4	home
5	etc

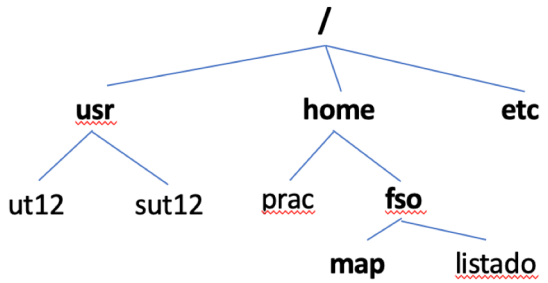
4	.
1	..
35	prac
84	fso

3	.
1	..
60	ut12
61	sut12

84	.
4	..
90	map
91	listado

5	.
1	..

90	.
84	..



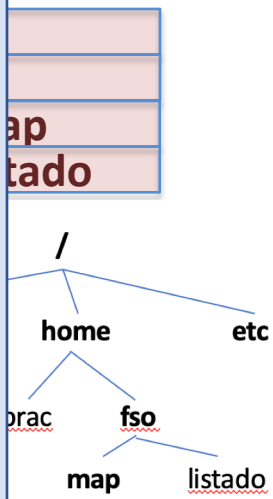
a) Draw the directory and file tree that corresp

b) Describe **what i-node numbers and how ma**
read the first 128 bytes in file **/home/fso/listado**

c) What information should we get when reading the first 32 bytes in file **/home/fso/map**

- ory

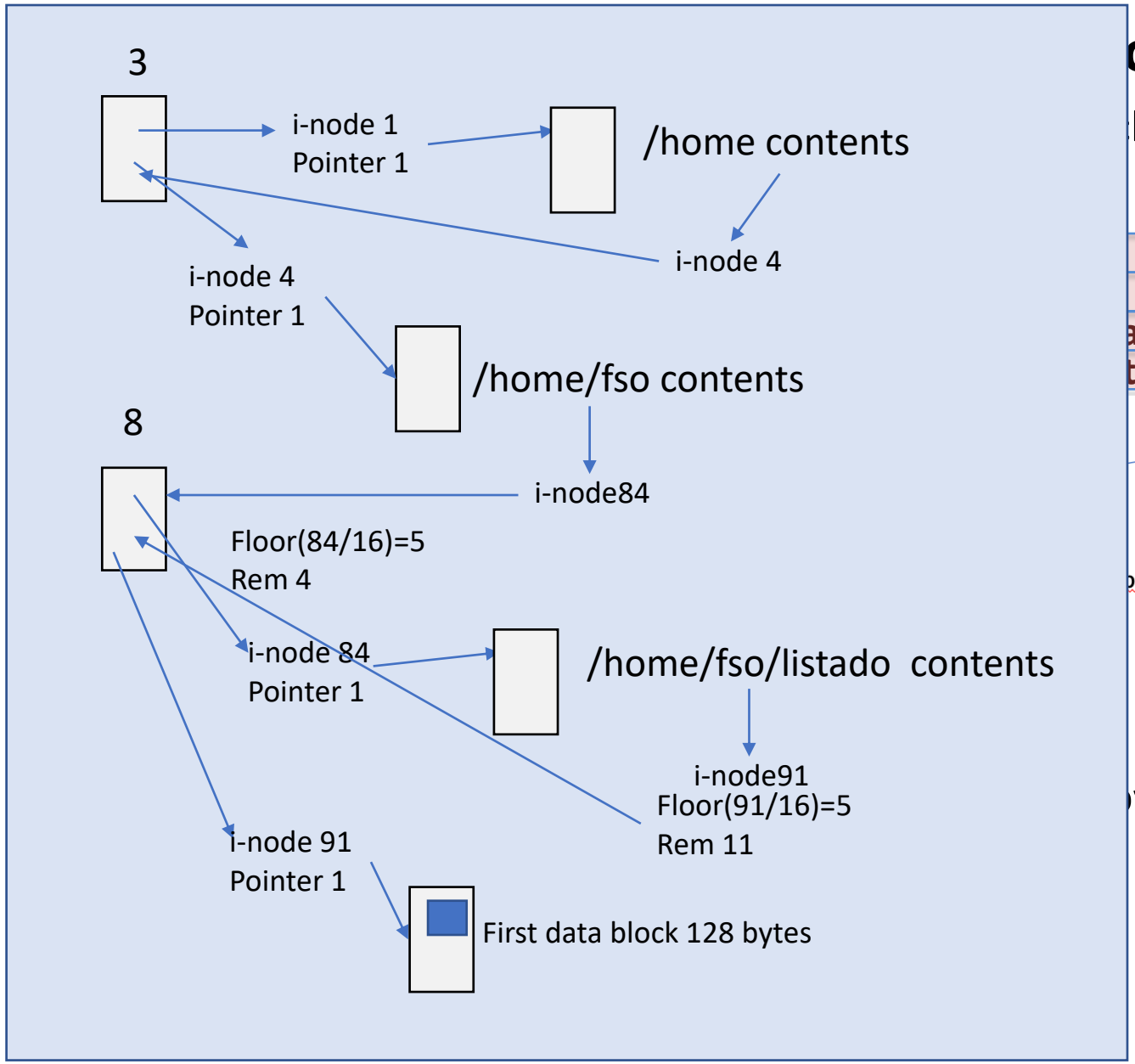
h actual



a)

b)

c) ytes in file



- **Exercise 3: Looking for a file inside a directory**

  Consider a Minix file system created with standard sizes which actual directory entries are the following:

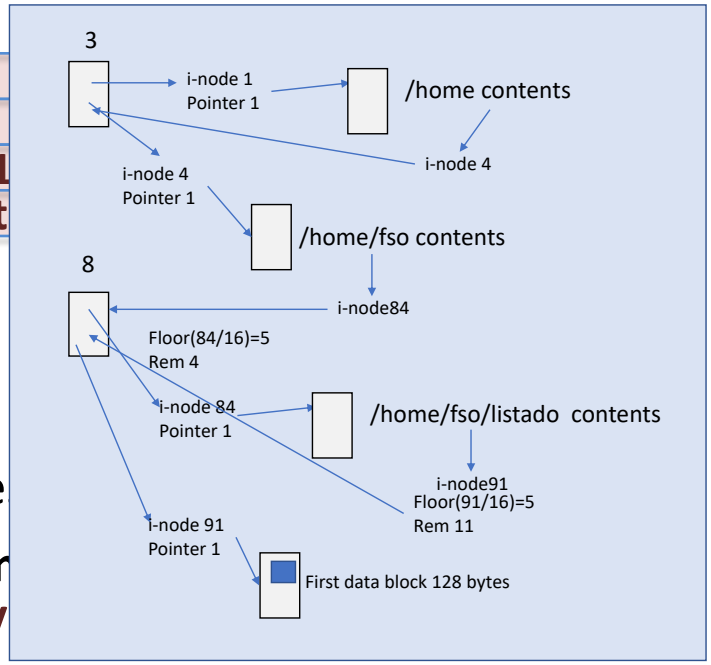
1	.
1	..
3	usr
4	home
5	etc

4	.
1	..
35	prac
84	fso

3	.
1	..
60	ut1
61	sut

5	.
1	..

90	.
84	..



a) Draw the directory and file tree that corre

b) Describe **what i-node numbers and how r** read the first 128 bytes in file **/home/fso/**

c) What information should we get when reading the first 32 bytes in file **/home/fso/map**

90	.
84	..

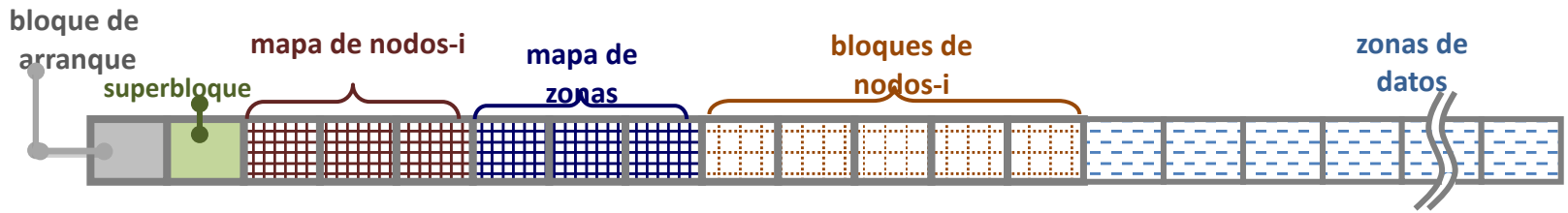
Ejercicio 4:

- Un disco con una capacidad de 8GB, se formatea con una versión de Minix. Los tamaños usados en el formateo son:
  - Tamaño de bloque = 2KBytes . Tamaño de Zona = 2^0 bloques = 1 Bloque
  - Los punteros a Zona son de 32bits=4Bytes
  - El tamaño del nodo-i es de 64 Bytes (7 punteros directos, 1 indirecto, 1 doble indirecto) .
  - Cada entrada de directorio ocupa 32 Bytes.
  - Al formatear se ha reservado espacio en la cabecera para 4.096 nodos-i
- Se pide:

  a) Calcule el número de bloques que ocupa cada uno de los elementos de la cabecera: Mapa de bits nodos-i, Mapa de bits Zonas y Nodos-i.

  b) Calcule el bloque que corresponde a la primera Zona de datos y el número de Zonas de datos.

  c) Suponga que este disco almacena un único directorio, el directorio raíz que contiene 10 archivos regulares,
  - c1) Indique el número de zonas de datos que ocupa el directorio raíz
  - c2) Suponga además que cada uno de los archivos regulares contiene una información que ocupa 50KBytes e indique de forma justificada el número de zonas de datos ocupadas para este caso, tenga en cuenta tanto los datos como los metadatos del archivo.



Ejercicio 4 (solución):

- Un disco con una capacidad de 8GB, se formatea con una versión de Minix. Los tamaños usados en el formateo son:
  - Tamaño de bloque = 2KBytes . Tamaño de Zona = 2^0 bloques = 1 Bloque
  - Los punteros a Zona son de 32bits=4Bytes
  - El tamaño del nodo-i es de 64 Bytes (7 punteros directos, 1 indirecto, 1 doble indirecto) .
  - Cada entrada de directorio ocupa 32 Bytes.
  - Al formatear se ha reservado espacio en la cabecera para 4.096 nodos-i
- Se pide:

  a) Calcule el número de bloques que ocupa cada uno de los elementos de la cabecera: Mapa de bits nodos-i, Mapa de bits Zonas y Nodos-i.

a)

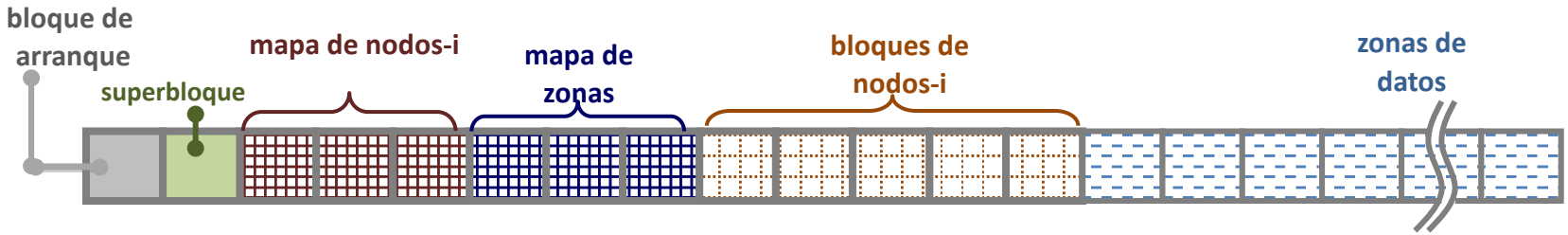
Tamaño de Bloque=2KBytes → 2*8* 1024 bits

4096 nodos-i →Mapa de nodos-i = 4096 bits → 1 bloque

Nodos-i → 4096 nodos-i * 64Bytes = 2^{12} * 2^6 =2^{18} Bytes → 2^{18} /2KBytes =2^7=128 bloques

Disco 8GBytes y Zonas = 2KBytes → 8GBytes/2KBytes= 2^{22} zonas

Mapa de zonas con 2^{22} bits → 2^{22}/2*8*1024 =2^8 bloques= 256 bloques

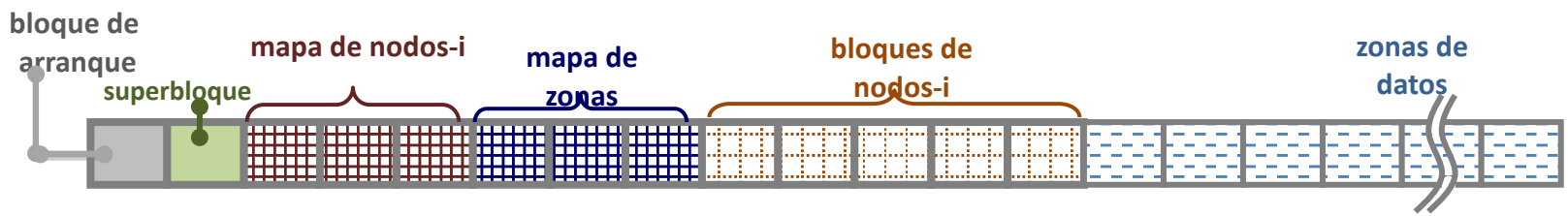


Ejercicio 4:

- Un disco con una capacidad de 8GB, se formatea con una versión de Minix. Los tamaños usados en el formateo son:
  - Tamaño de bloque = 2KBytes . Tamaño de Zona = 2^0 bloques = 1 Bloque
  - Los punteros a Zona son de 32bits=4Bytes
  - El tamaño del nodo-i es de 64 Bytes (7 punteros directos, 1 indirecto, 1 doble indirecto) .
  - Cada entrada de directorio ocupa 32 Bytes.
  - Al formatear se ha reservado espacio en la cabecera para 4.096 nodos-i
- Se pide:

  b) Calcule el bloque que corresponde a la primera Zona de datos y el número de Zonas de datos.

b)
1 Arranque + 1 SuperBloque + 1 Mapa nodos-i + 256 Mapa de Zonas +128 Nodos-i = 387
Bloque 387 será el primer bloque de datos = Primera zona de datos

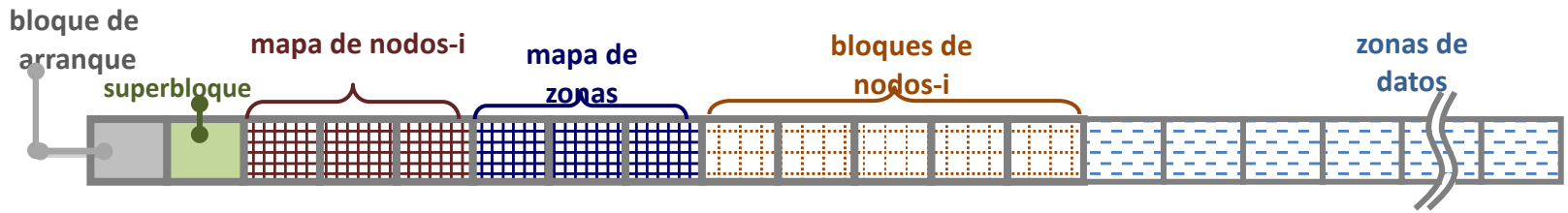


Ejercicio 4:

- Un disco con una capacidad de 8GB, se formatea con una versión de Minix. Los tamaños usados en el formateo son:
  - Tamaño de bloque = 2KBytes . Tamaño de Zona = 2^0 bloques = 1 Bloque
  - Los punteros a Zona son de 32bits = 4Bytes
  - El tamaño del nodo-i es de 64 Bytes (7 punteros directos, 1 indirecto, 1 doble indirecto) .
  - Cada entrada de directorio ocupa 32 Bytes.
  - Al formatear se ha reservado espacio en la cabecera para 4.096 nodos-i
- Se pide:

  c) Suponga que este disco almacena un único directorio, el directorio raíz que contiene 10 archivos regulares,
  - c1) Indique el número de zonas de datos que ocupa el directorio raíz

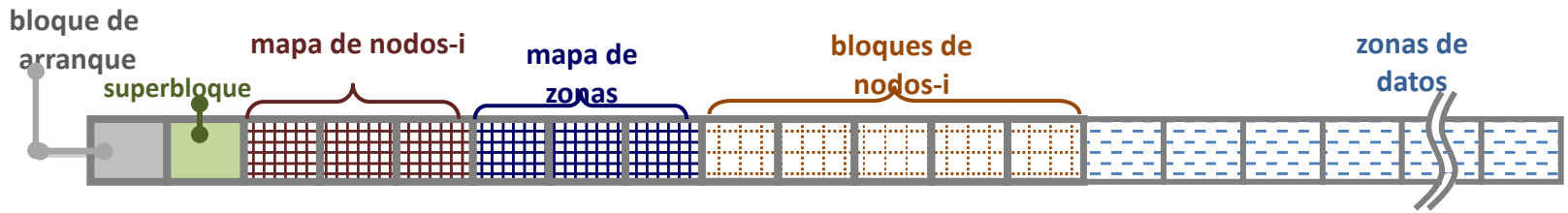
c1)
Directorio raíz contiene 10 entradas de archivos regulares + . y .. = 12 entradas → 12 entradas *64 Bytes =1280 Bytes → 1 zona



Ejercicio 4:

- Un disco con una capacidad de 8GB, se formatea con una versión de Minix. Los tamaños usados en el formateo son:
  - Tamaño de bloque = 2KBytes . Tamaño de Zona = 2^0 bloques = 1 Bloque
  - Los punteros a Zona son de 32bits=4Bytes
  - El tamaño del nodo-i es de 64 Bytes (7 punteros directos, 1 indirecto, 1 doble indirecto) .
  - Cada entrada de directorio ocupa 32 Bytes.
  - Al formatear se ha reservado espacio en la cabecera para 4.096 nodos-i
- c) Suponga que este disco almacena un único directorio, el directorio raíz que contiene 10 archivos regulares,
  - c1) Indique el número de zonas de datos que ocupa el directorio raíz
  - c2) Suponga además que cada uno de los archivos regulares contiene una información que ocupa 50KBytes e indique de forma justificada el número de zonas de datos ocupadas para este caso, tenga en cuenta tanto los datos como los metadatos del archivo.

50KBytes → 25 zonas de datos → Necesita 25 punteros a zona → 7 punteros directos + 1 zona que contiene 18 punteros → Total 26 zonas para cada archivos
26 zonas *10 archivos =260 zonas para archivos regular
260 zonas + 1 zona del raíz = 261 zonas



Ejercicio-5

Dado el siguiente código en el cual se generan al menos tres procesos P1, P2 y P3:

```
...
pipe(fd);/*pipe1*/
pipe(fd2);/*pipe2*/
if(fork() != 0){
  /***Proceso P1 ***/
  dup2(fd[1],STDOUT_FILENO);
  close(fd[0]); close(fd[1]);
  dup2(fd2[0],STDIN_FILENO);
  close(fd2[0]);
close(fd2[1]);
  /*tabla proceso P1*/
}else{
  /***Proceso P2 ***/
  dup2(fd[0],STDIN_FILENO);
  close(fd[0]); close(fd[1]);
```

```
  pipe(fd); /*pipe3*/
  if(fork() != 0){
    close(fd2[0]);
    close(fd2[1]);
    dup2(fd[1],STDOUT_FILENO);
    close(fd[0]);
    close(fd[1]);
    /*tabla proceso P2*/
  }else{
     /***Proceso P3 ***/
     dup2(fd[0],STDIN_FILENO);
     close(fd[0]);
     close(fd[1]);
     dup2(fd2[1],STDOUT_FILENO);
     close(fd2[0]);
     close(fd2[1]);
    /*tabla proceso P3*/
  }
 }
...
```

a) Indique el contenido de las tablas de descriptores de archivo para los procesos P1, P2 y P3, en los puntos del código marcados como /*tabla proceso Pi*/.

b) Determine el parentesco que existe entre P1, P2 y P3 así como el esquema de redirecciones resultante de ejecutar el código.

Ejercicio-5 (solución)

Dado el siguiente código en el cual se generan al menos tres procesos P1, P2 y P3:

```
...
pipe(fd);/*pipe1*/
pipe(fd2);/*pipe2*/
if(fork() != 0){
  /***Proceso P1 ***/
  dup2(fd[1],STDOUT_FILENO);
  close(fd[0]); close(fd[1]);
  dup2(fd2[0],STDIN_FILENO);
  close(fd2[0]);
close(fd2[1]);
  /*tabla proceso P1*/
}else{
  /***Proceso P2 ***/
  dup2(fd[0],STDIN_FILENO);
  close(fd[0]); close(fd[1]);
```

```
  pipe(fd); /*pipe3*/
  if(fork() != 0){
    close(fd2[0]);
    close(fd2[1]);
    dup2(fd[1],STDOUT_FILENO);
    close(fd[0]);
    close(fd[1]);
    /*tabla proceso P2*/
  }else{
     /***Proceso P3 ***/
     dup2(fd[0],STDIN_FILENO);
     close(fd[0]);
     close(fd[1]);
     dup2(fd2[1],STDOUT_FILENO);
     close(fd2[0]);
     close(fd2[1]);
    /*tabla proceso P3*/
  }
}
```

Proceso P1		
0	fd2[0]	/*pipe2*/
1	fd[1]	/*pipe1*/
2	stderr	

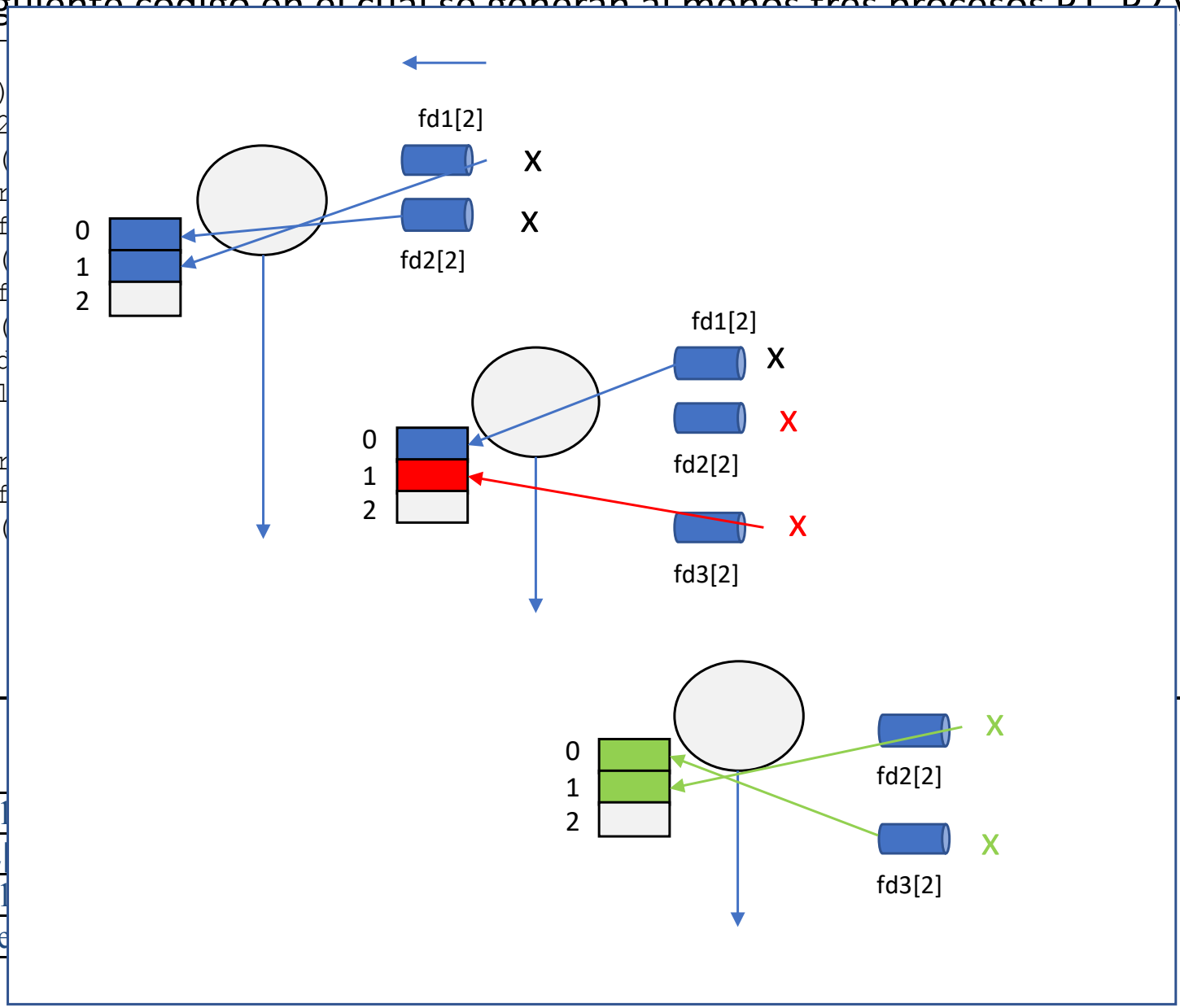
Proceso P2		
0	fd[0]	/*pipe1*/
1	fd[1]	/*pipe3*/
2	stderr	

Proceso P3		
0	fd[0]	/*pipe3*/
1	fd2[1]	/*pipe2*/
2	stderr	

Ejercicio-5 (solución)

Dado el sig

```
...
pipe(fd)
pipe(fd2
if(fork(
   /***Pr
   dup2(f
   close(
   dup2(f
   close(
close(fd
   /*tabl
}else{
   /***Pr
   dup2(f
   close(
```



Proceso P1	
0	fd2[
1	fd[1
2	stde

Ejercicio-5 (solución)

Dado el siguiente código en el cual se generan al menos tres procesos P1, P2 y P3:

```
...
pipe(fd);/*pipe1*/
pipe(fd2);/*pipe2*/
if(fork() != 0){
  /***Proceso P1 ***/
  dup2(fd[1],STDOUT_FILENO);
  close(fd[0]); close(fd[1]);
  dup2(fd2[0],STDIN_FILENO);
  close(fd2[0]);
close(fd2[1]);
  /*tabla proceso P1*/
}else{
  /***Proceso P2 ***/
  dup2(fd[0],STDIN_FILENO);
  close(fd[0]); close(fd[1]);
  pipe(fd); /*pipe3*/
  if(fork() != 0){
    close(fd2[0]);
    close(fd2[1]);
    dup2(fd[1],STDOUT_FILENO);
    close(fd[0]);
    close(fd[1]);
    /*tabla proceso P2*/
  }else{
     /***Proceso P3 ***/
     dup2(fd[0],STDIN_FILENO);
     close(fd[0]);
     close(fd[1]);
     dup2(fd2[1],STDOUT_FILENO);
     close(fd2[0]);
     close(fd2[1]);
    /*tabla proceso P3*/
  }
 }
...
```

El proceso P1 es el padre de P2, P2 es el padre de P3
Se establece un anillo de comunicación entre los tres procesos utilizando tres tubos