



# CS 1550

Week 13

—

Lab 5

Teaching Assistant

Henrique Potter

# Lab 5 – Bigger Files for xv6

---

- Reuse xv6 qemu
- Changing how xv6 organizes file blocks in the **inode**

# Lab 5 – Bigger Files for xv6

---

- Current implementation limit files to 72 KB

Disk Sectors

0	
1	
2	
...	
139	

xv6 Can only map 140 sectors

# Lab 5 – Bigger Files for xv6

---

- Current implementation limit files to 72 KB

Disk Sectors

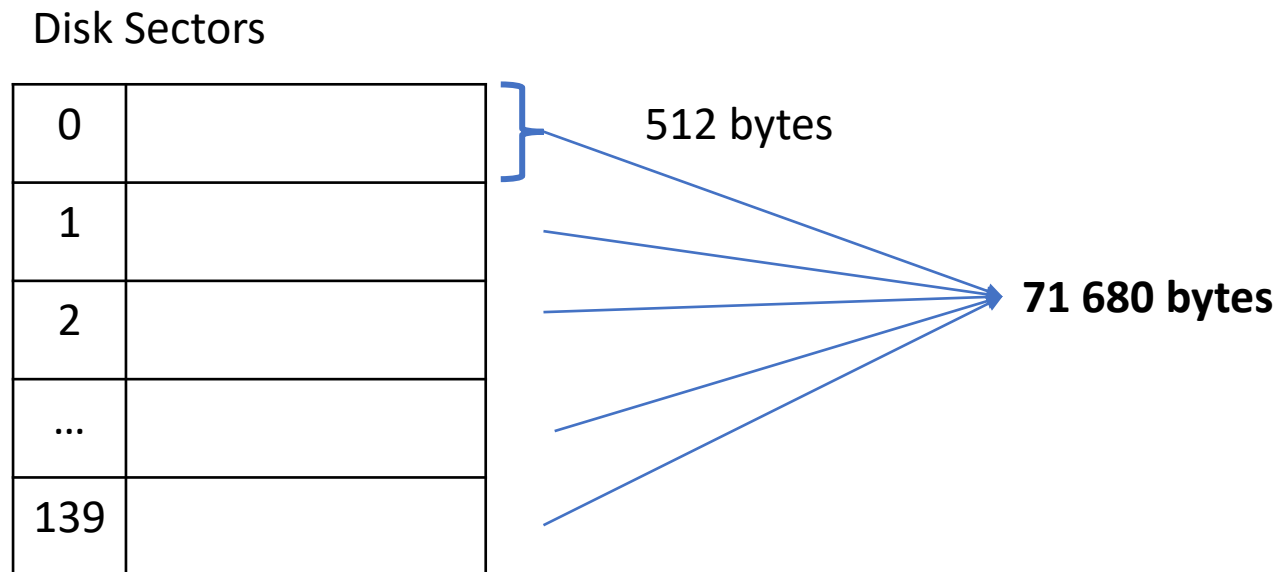
0	
1	
2	
...	
139	

} 512 bytes

# Lab 5 – Bigger Files for xv6

---

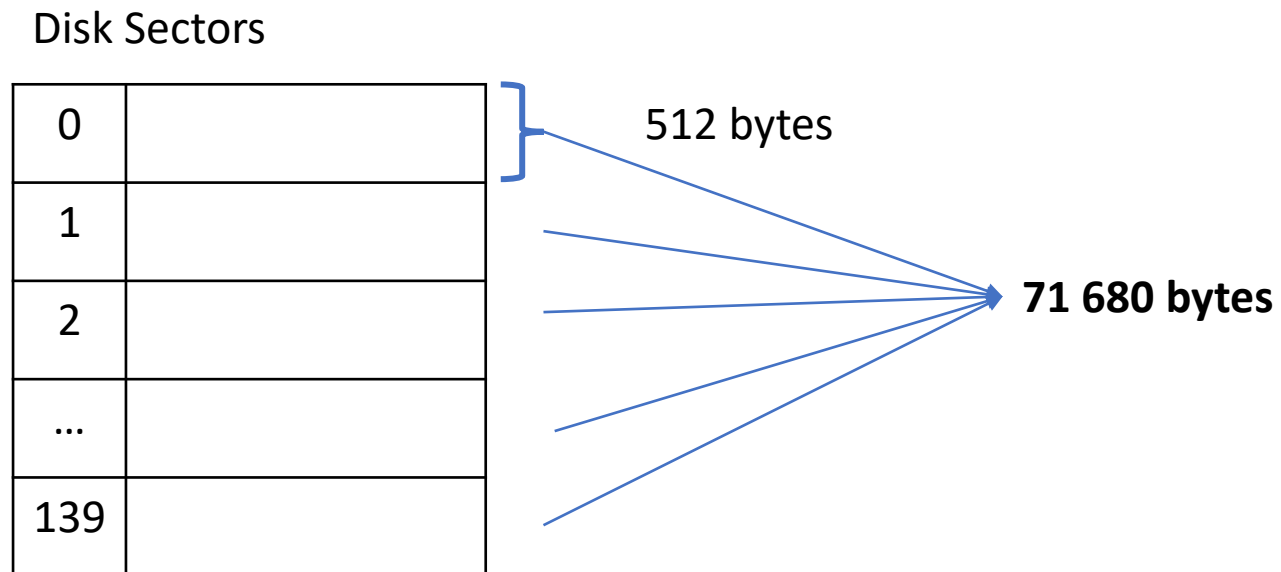
- Current implementation limit files to 72 KB



# Lab 5 – Bigger Files for xv6

---

- Current implementation limit files to 72 KB



**We want it to be able to map more than 140 sectors/data blocks.**

# Lab 5 – Bigger Files for xv6

---

- This limit is based in the current implementation of direct and indirect references you can use in the **inode**.

# Lab 5 – Bigger Files for xv6

---

- This limit is based in the current implementation of direct and indirect references you can use in the **inode**.
- We can modify xv6 to be able to **map** bigger files by adding a **double indirect** reference in its inode.



# Lab 5 – Bigger Files for xv6

---

- This limit is based in the current implementation of direct and indirect references you can use in the **inode**.
- We can modify xv6 to be able to **map** bigger files by adding a **double indirect** reference in its inode.
- xv6 file system **is similar** to **Unix Fast File System (FFS)** or (UFS)

# Lab 5 – xv6 Unix Fast File System

---

**inode**

metadata

Every file have an inode  
that holds its metadata



# Lab 5 – xv6 Unix Fast File System

---

**inode**

metadata

Every file have an inode  
that holds its metadata

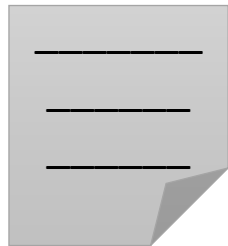
It also maps where to find in  
disk the **blocks of data** that  
compose that file

# Lab 5 – xv6 Unix Fast File System

---

**inode**

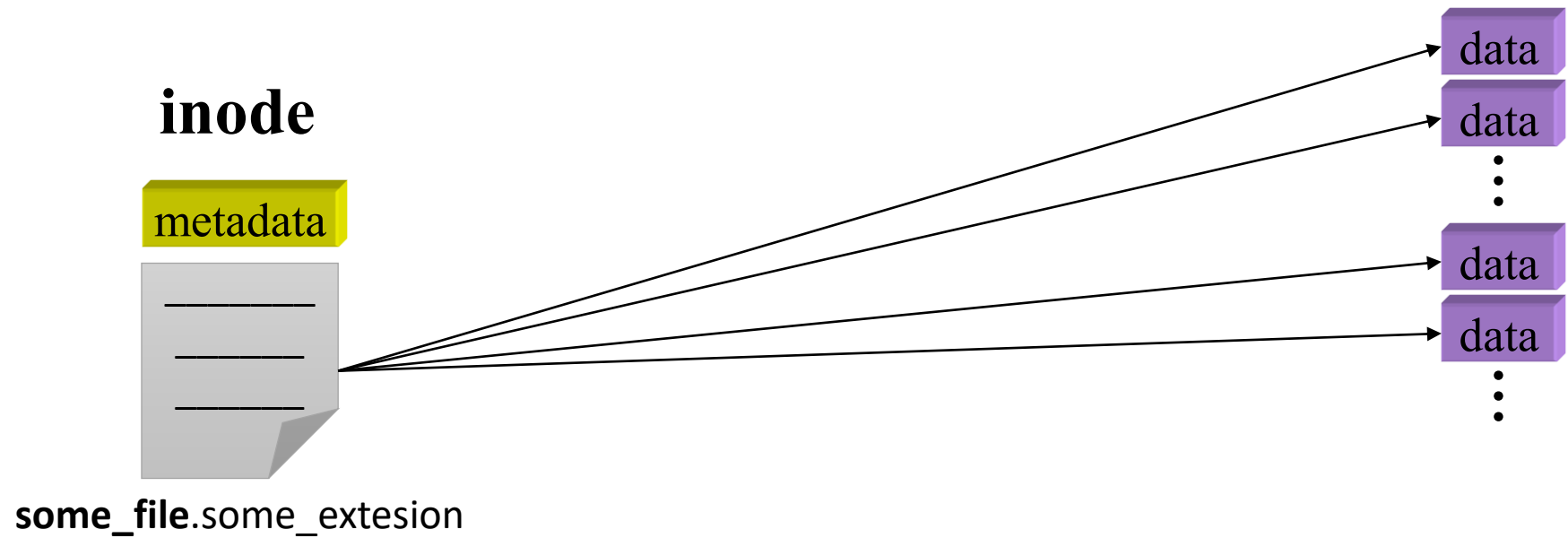
metadata



**some\_file**.some\_extesion

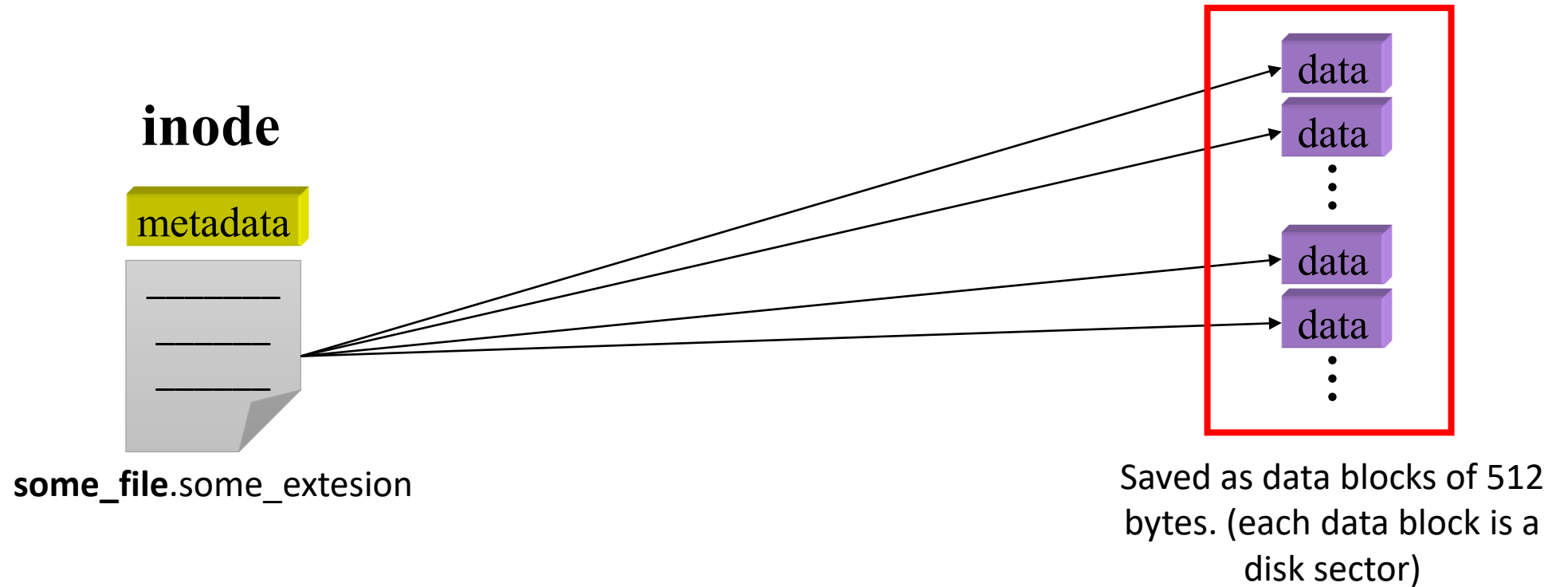
# Lab 5 – xv6 Unix Fast File System

---



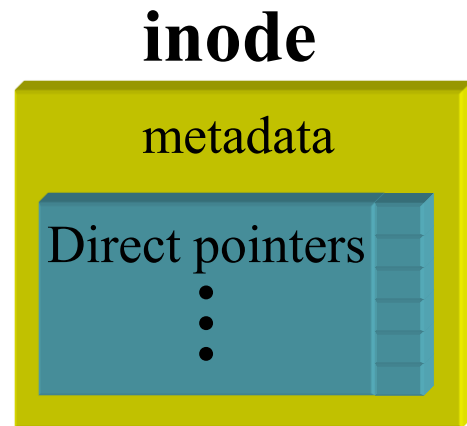
# Lab 5 – xv6 Unix Fast File System

---



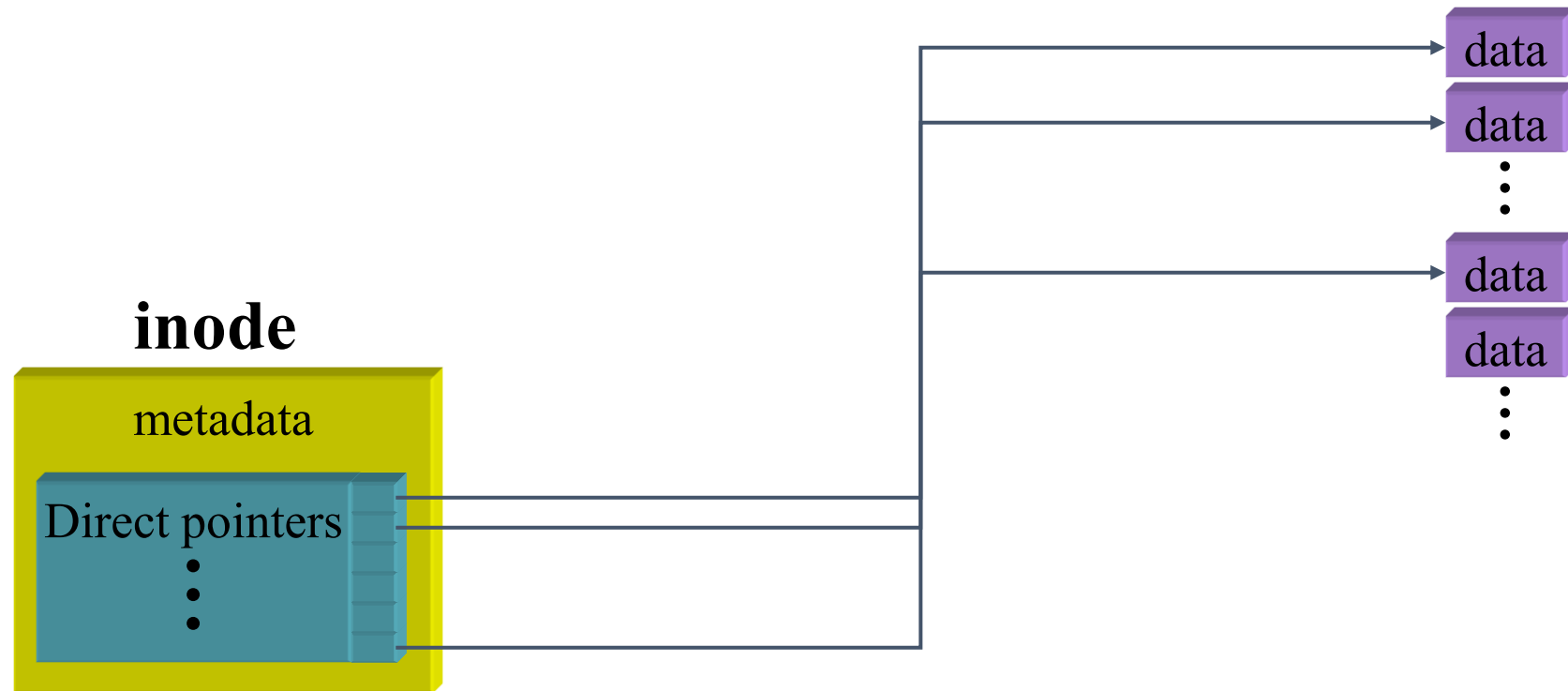
# Lab 5 – xv6 Unix Fast File System

---



# Lab 5 – xv6 Unix Fast File System

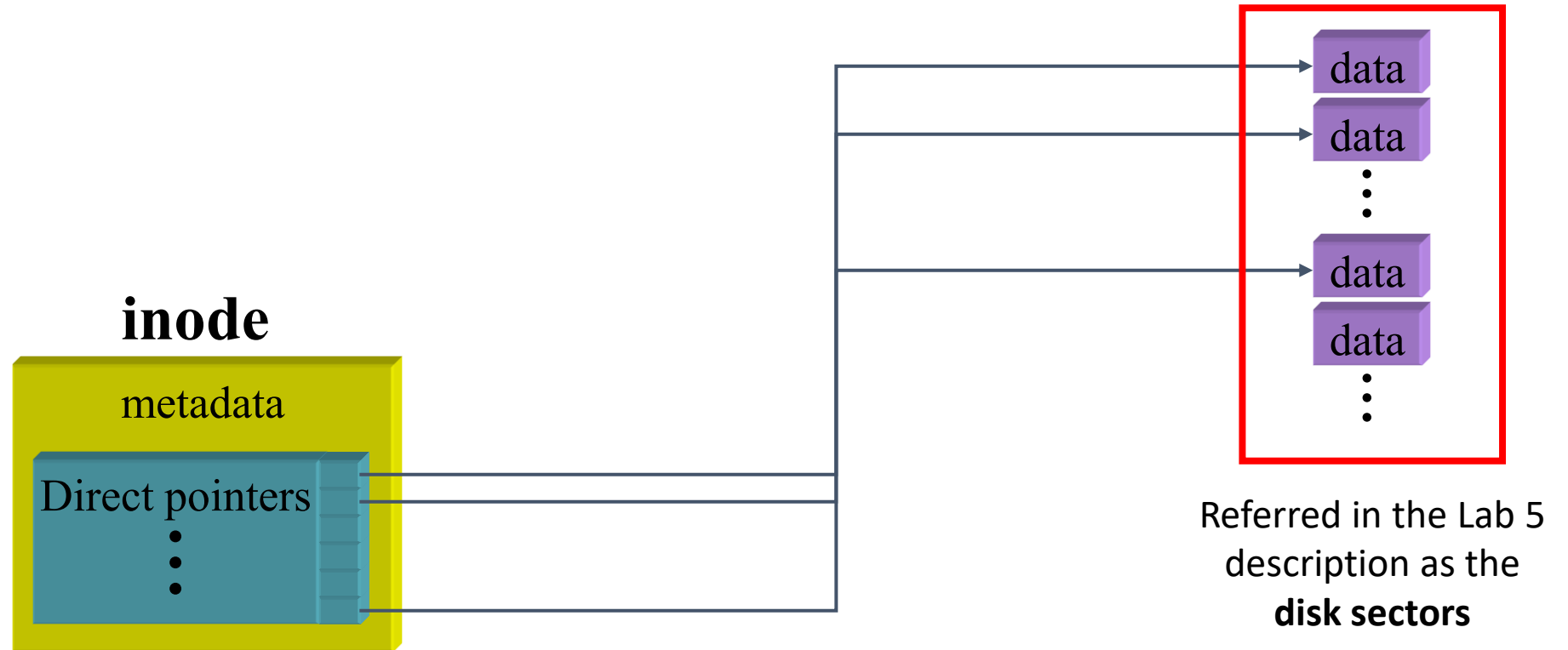
---





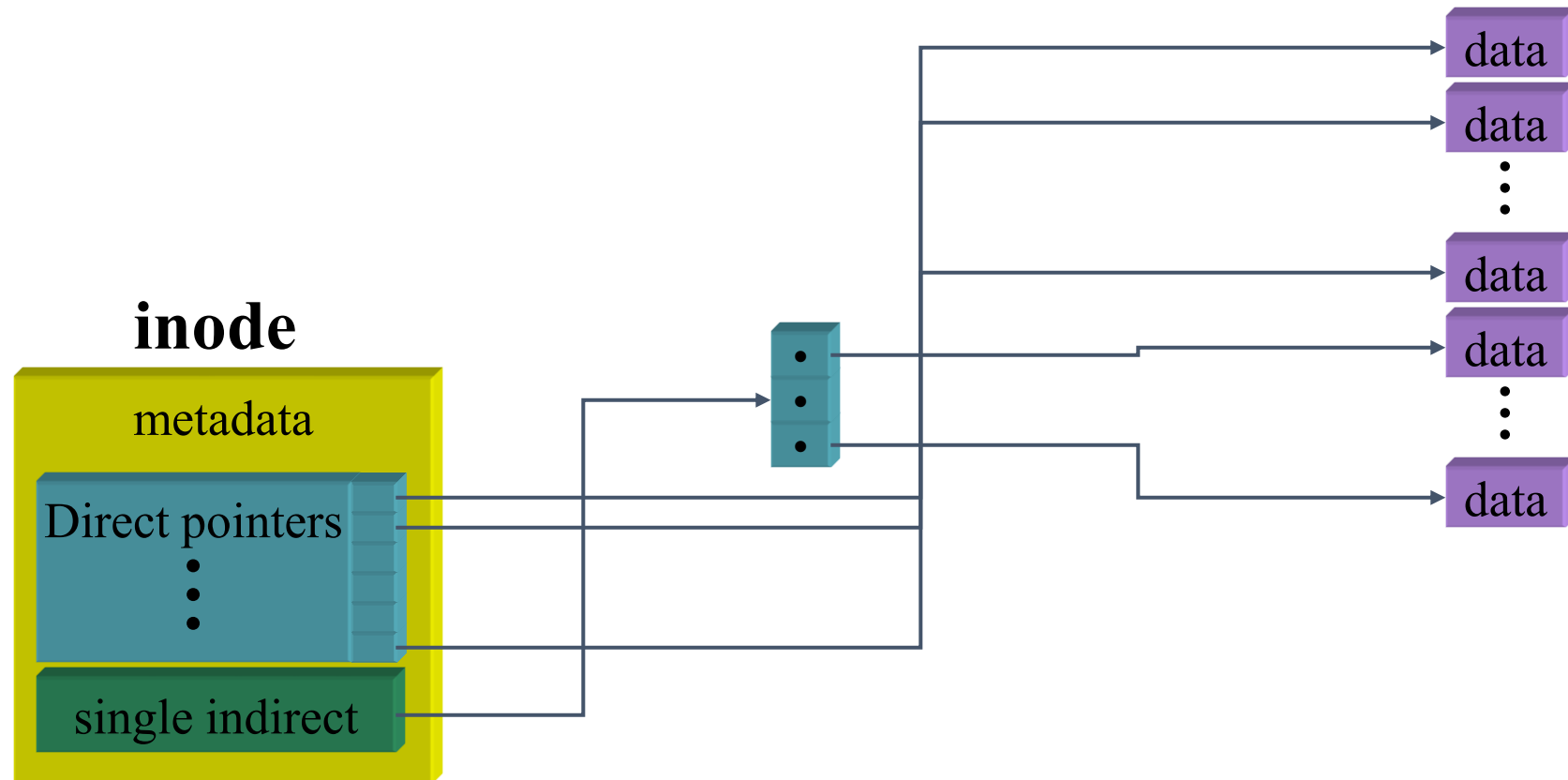
# Lab 5 – xv6 Unix Fast File System

---

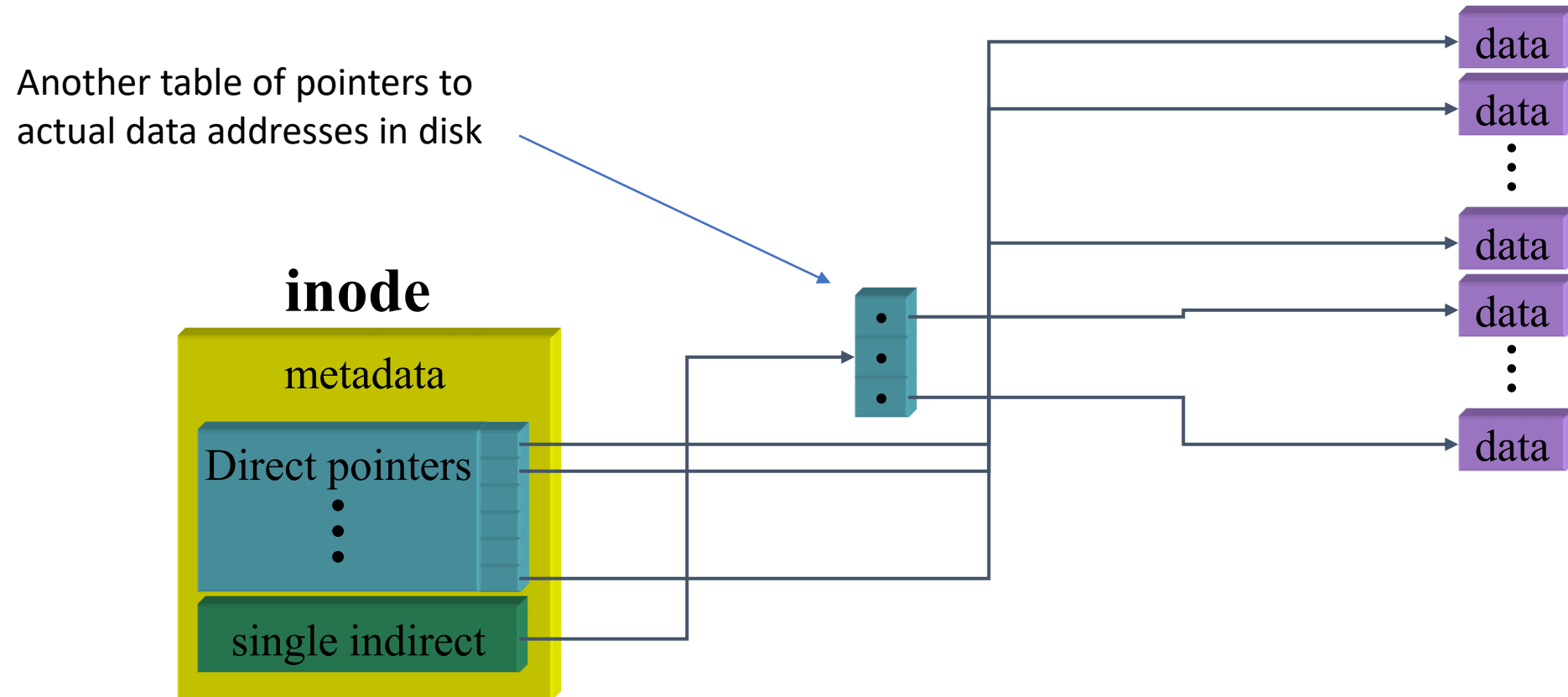


# Lab 5 – xv6 Unix Fast File System

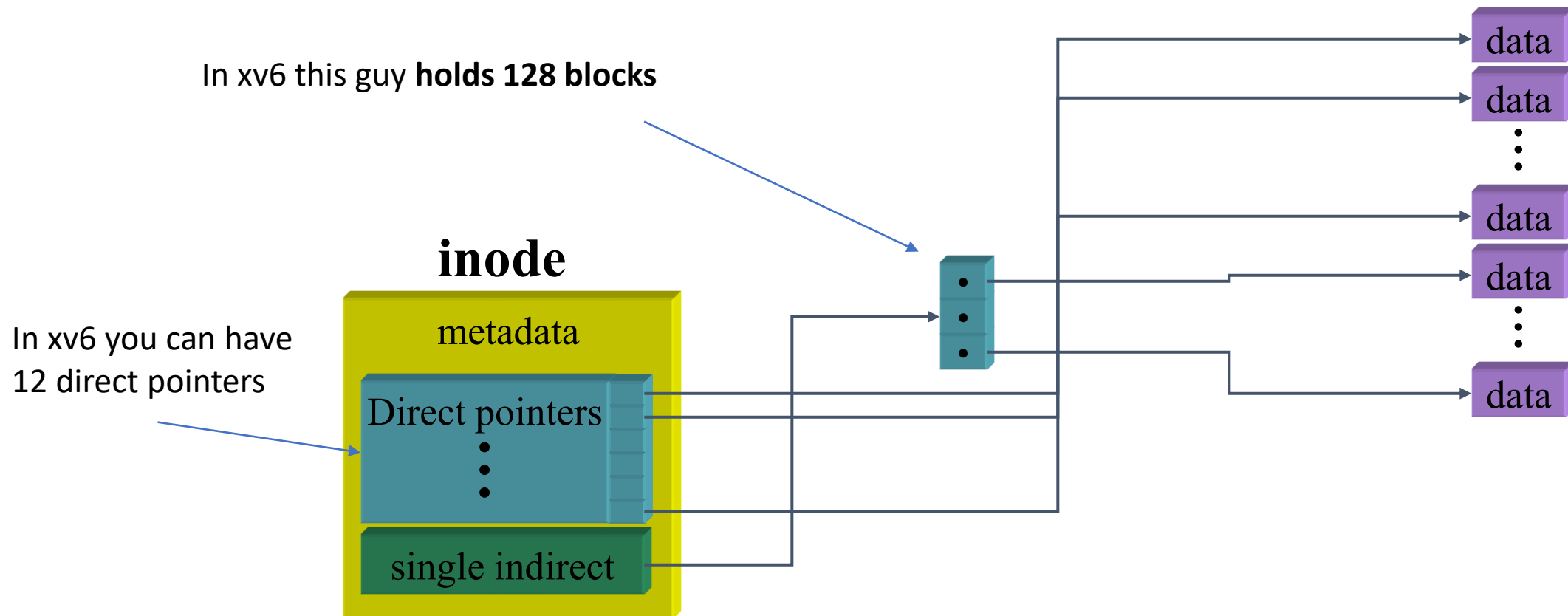
---



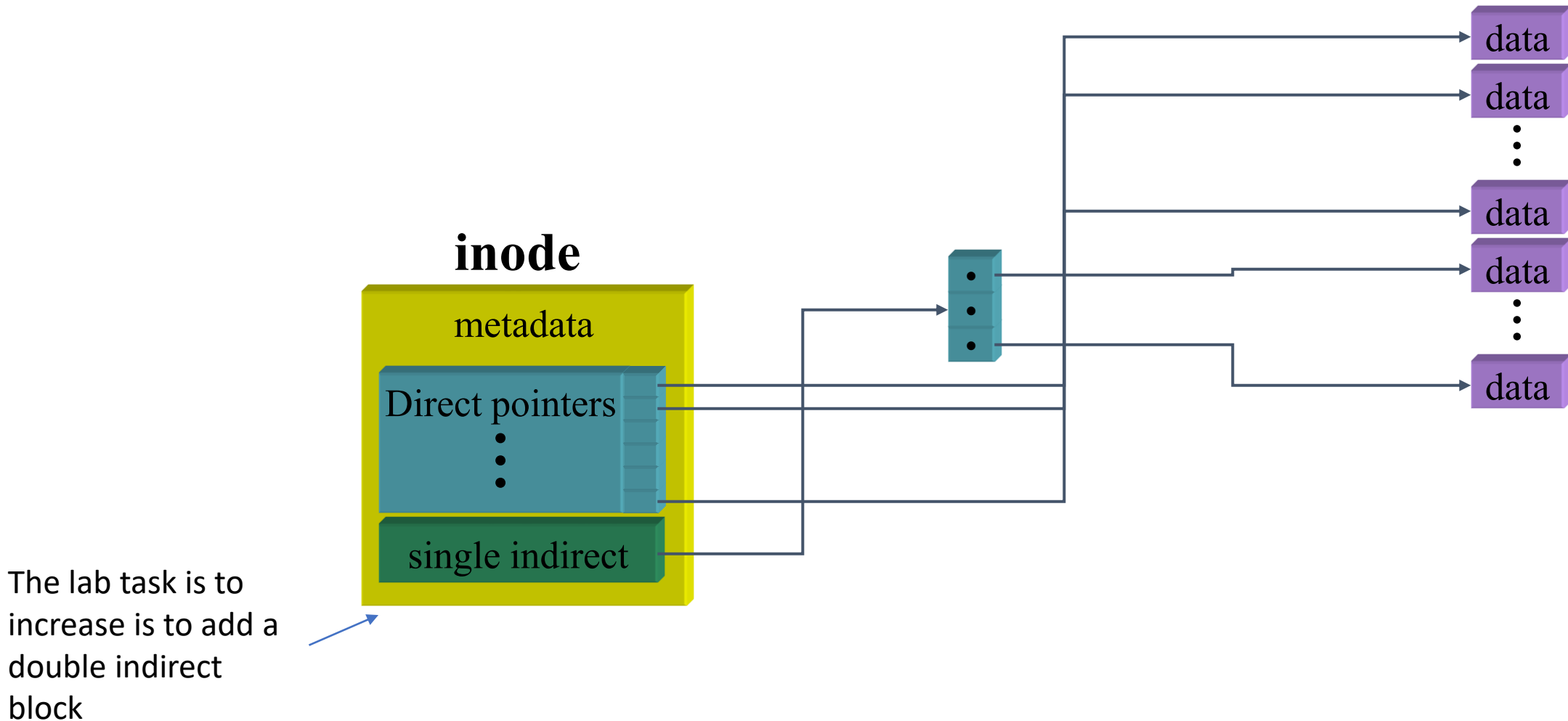
# Lab 5 – xv6 Unix Fast File System



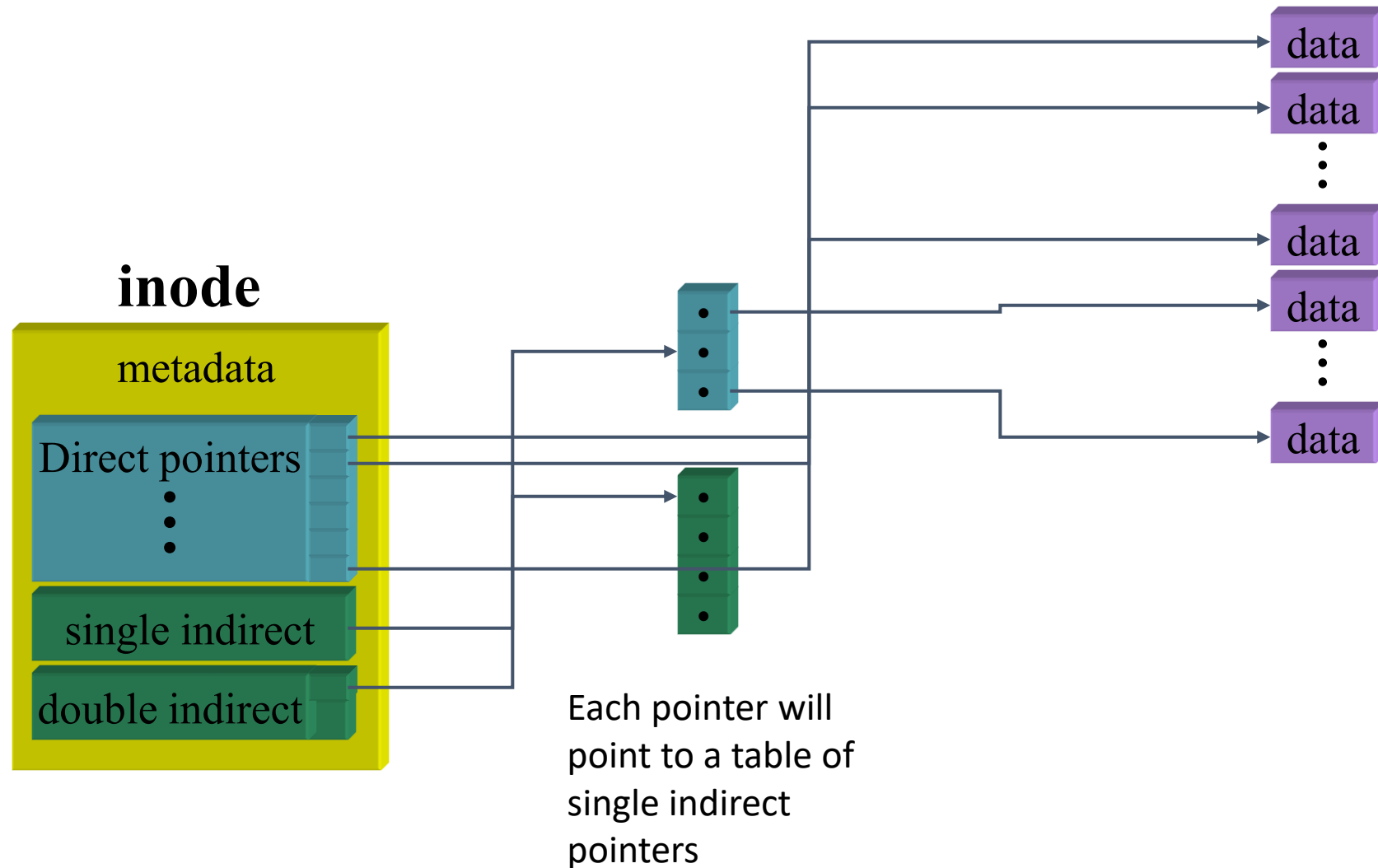
# Lab 5 – xv6 Unix Fast File System



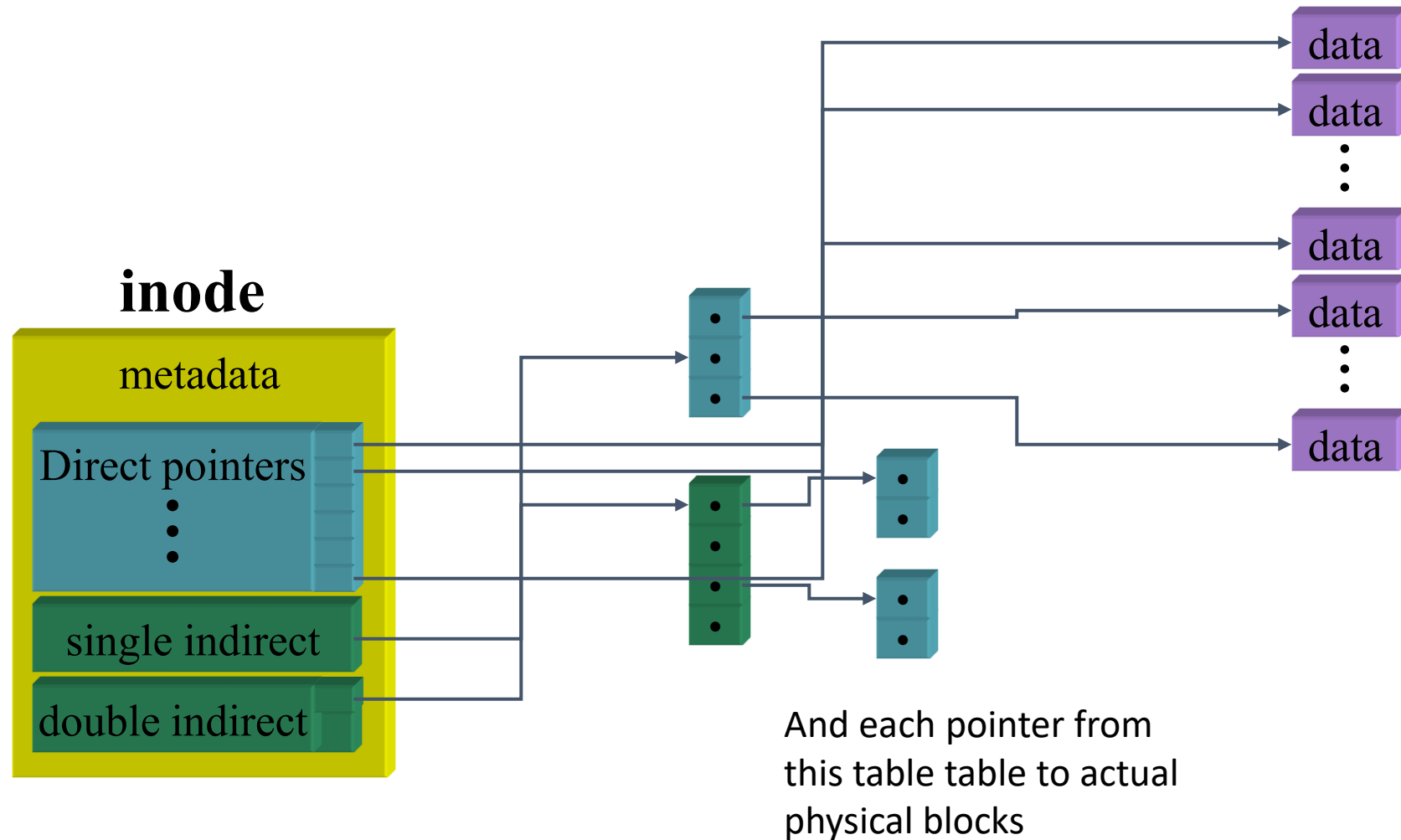
# Lab 5 – xv6 Unix Fast File System



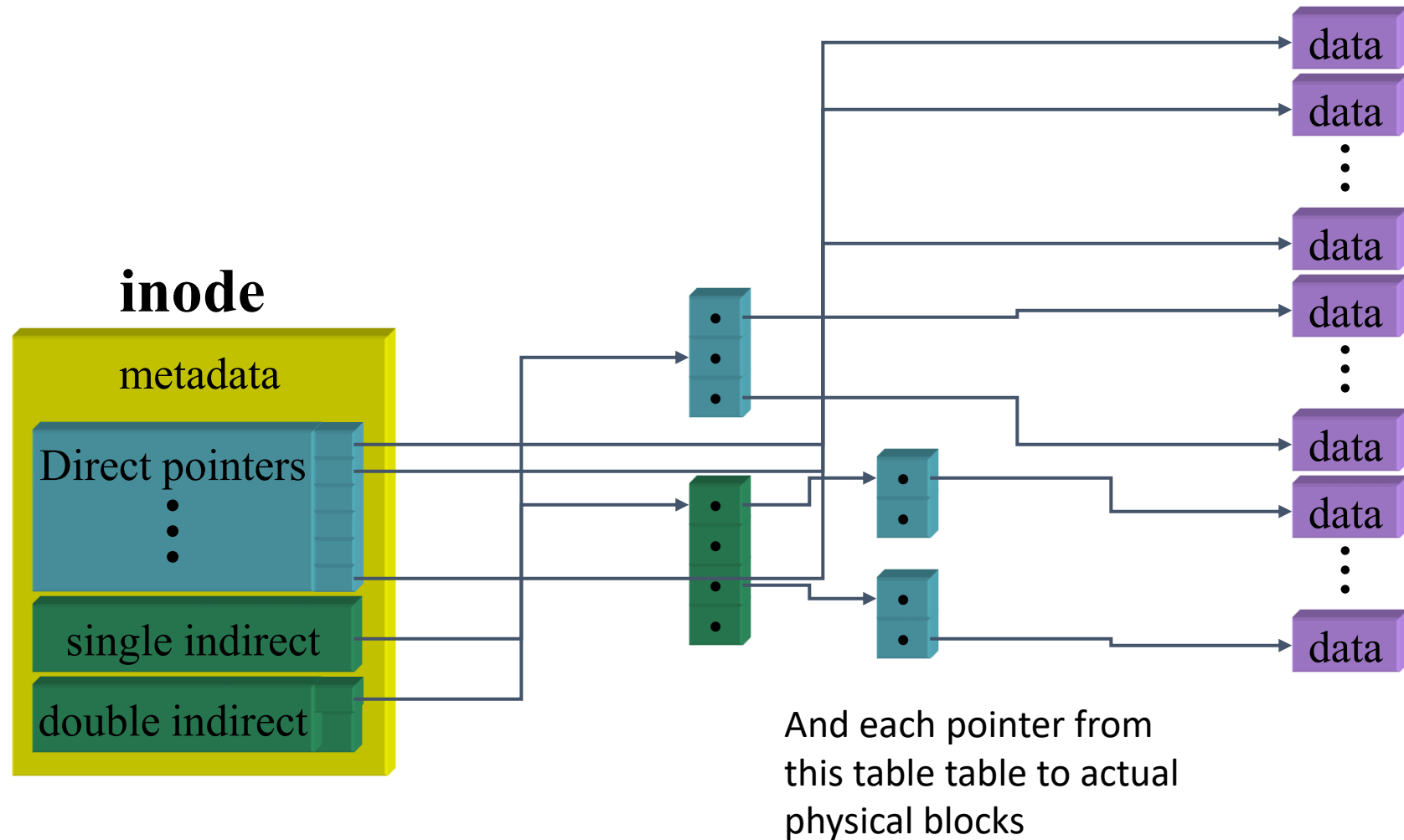
# Lab 5 – xv6 Unix Fast File System



# Lab 5 – xv6 Unix Fast File System

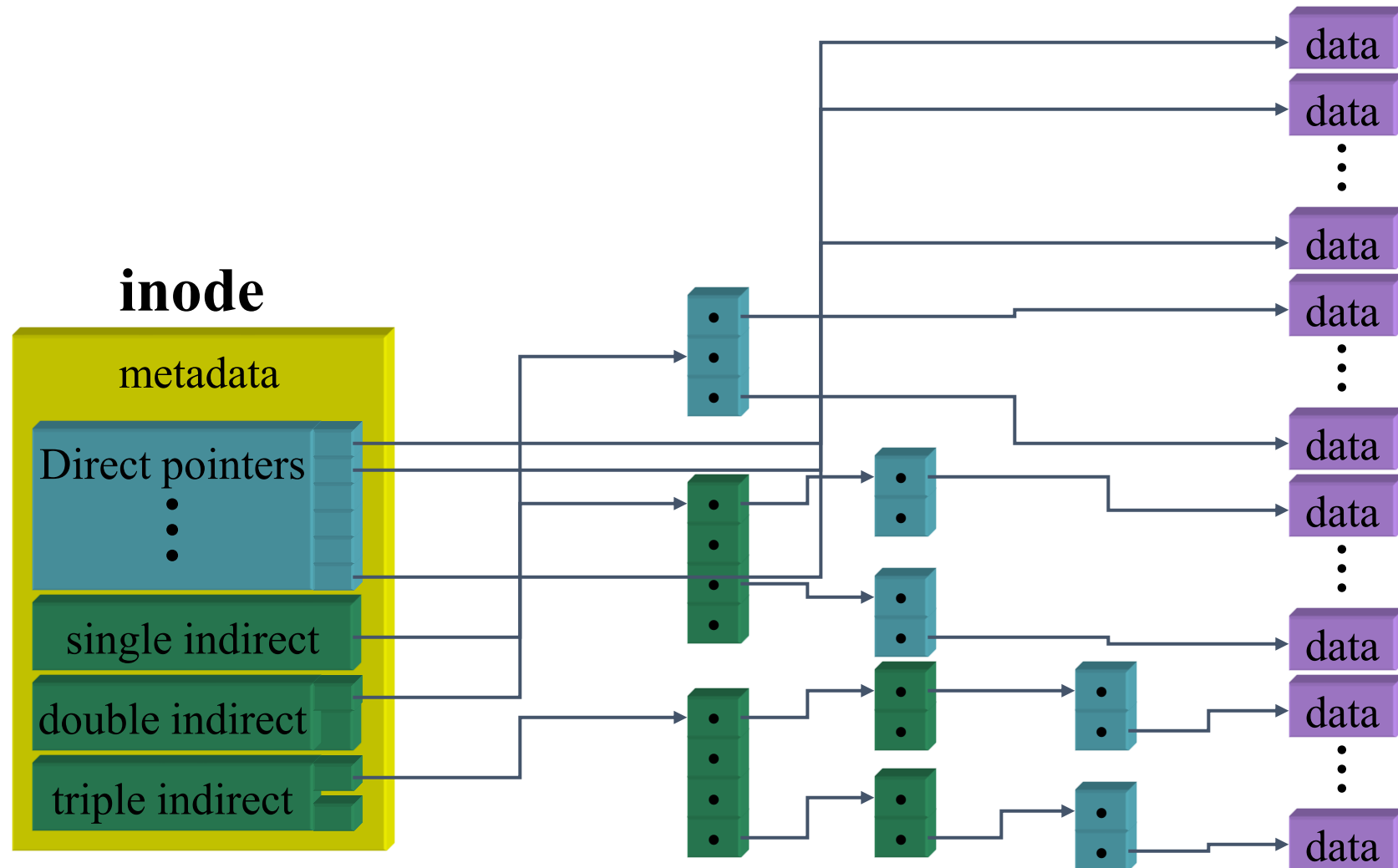


# Lab 5 – xv6 Unix Fast File System

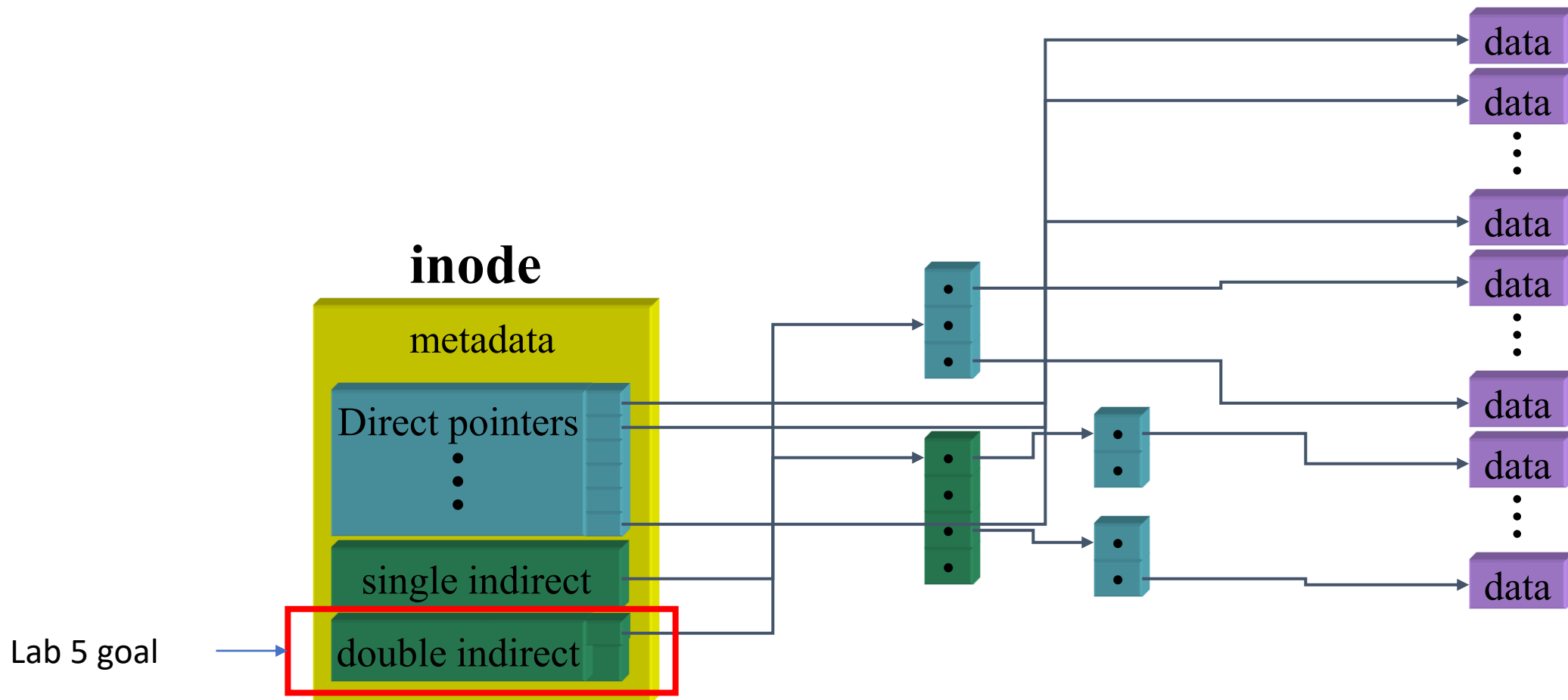




# Lab 5 – xv6 Unix Fast File System



# Lab 5 – xv6 Unix Fast File System



It will allow us to map files up to 8.5 MB

# Lab 5 – Bigger Files for xv6

---

- Important xv6 files for this lab are:
  - **fs.h** – defines the number of direct blocks
  - **fs.c** – bmap()
  - **file.h** – inode struct

# Lab 5 – Bigger Files for xv6

---

- qemu optimizations
  - Modify the xv6 makefile
    - CPUS := 1
    - QEMUESTRA = -snapshot

# Lab 5 – Bigger Files for xv6

---

- qemu optimizations
  - Modify the xv6 makefile
    - CPUS := 1
    - QEMUESTRA = -snapshot
- Param.h
  - FSSIZE = 20000

# Lab 5 – Bigger Files for xv6

---

- qemu optimizations
  - Modify the xv6 makefile
    - CPUS := 1
    - QEMUESTRA = -snapshot
- Param.h
  - FSSIZE = 20000
- Copy **big.c**
  - Will attempt to create a big file and will use as many sectors it could use to create it

# Lab 5 – Bigger Files for xv6

---

- Writes to disk are delayed and will be written in batch once disk operations are done

# Lab 5 – Bigger Files for xv6

---

- Writes to disk are delayed and will be written in batch once disk operations are done
  - In the **bmap()** from the **fs.c** file

```
...
if((addr = a[bn]) == 0){
    a[bn] = addr = balloc(ip->dev);
    log_write(bp);
}
...
```

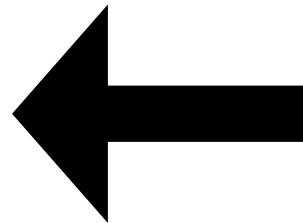


# Lab 5 – Bigger Files for xv6

---

- Writes to disk are delayed and will be written in batch once disk operations are done
  - In the **bmap()** from the **fs.c** file

```
...  
if((addr = a[bn]) == 0){  
    a[bn] = addr = balloc(ip->dev);  
    log_write(bp);  
}  
...
```



**Blocks are not immediately  
written to disk**

# Lab 5 – Bigger Files for xv6

---

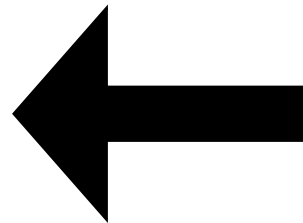
- Writes to disk are delayed and will be written in batch once disk operations are done
  - In the **end\_op(void)** from the **log.c** file

# Lab 5 – Bigger Files for xv6

---

- Writes to disk are delayed and will be written in batch once disk operations are done
  - In the **end\_op(void)** from the **log.c** file

```
...  
commit();  
...
```



At the end of all FS system calls  
end\_op is called and **commits**  
**all pending disk writes**

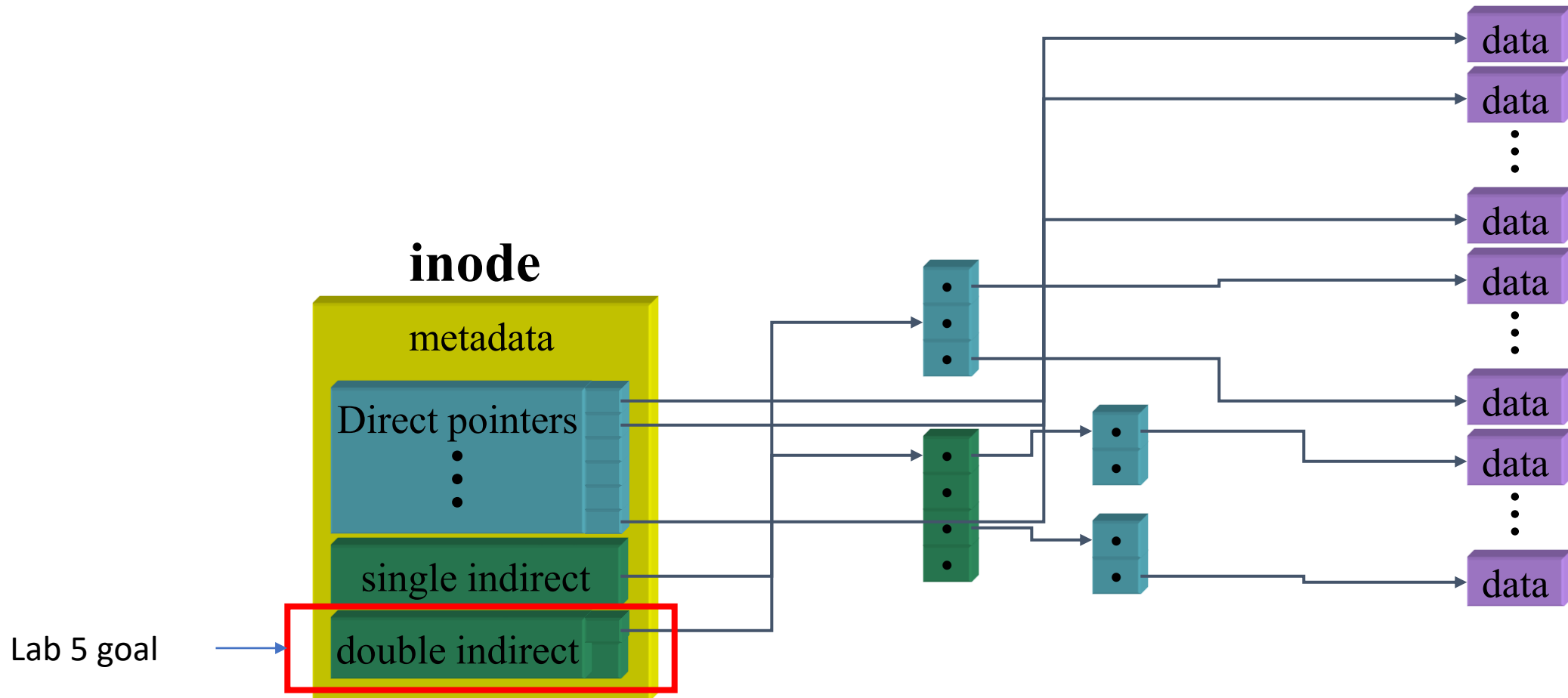
# Lab 5 – Bigger Files for xv6

---

- Writes to disk are delayed and will be written in batch once disk operations are done
  - In the **commit(void)** from the **log.c** file

```
commit()
{
    if (log.lh.n > 0)
        ...
        write_head(); // Write header to disk -- the real commit
        ...
}
```

# Lab 5 – xv6 Unix Fast File System



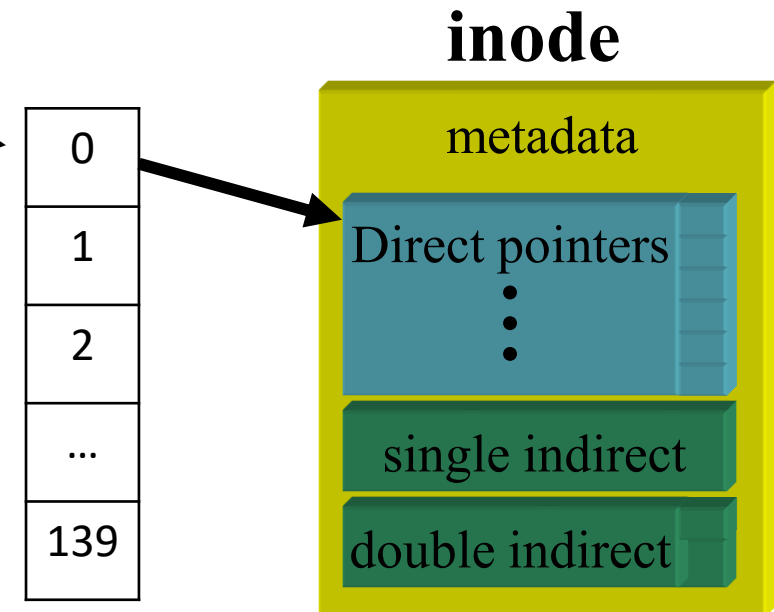
It will allow us to map files up to 8.5 MB

# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

```
bmap(struct inode *ip, uint bn)
uint addr, *a;
struct buf *bp;

if(bn < NDIRECT){
    if((addr = ip->addrs[bn]) == 0)
        ip->addrs[bn] = addr = balloc(ip->dev);
    return addr;
}
...
```



# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

```
bmap(struct inode *ip, uint bn)
```

```
uint addr, *a;
```

```
struct buf *bp;
```

```
if(bn < NDIRECT){
```

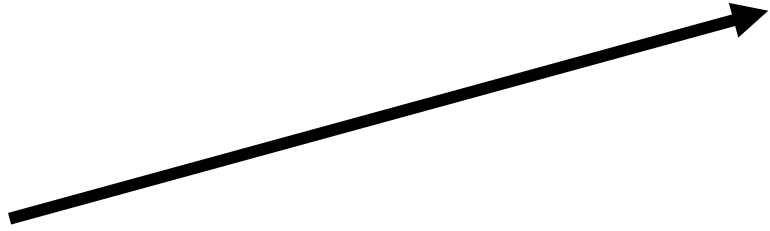
```
    if((addr = ip->addrs[bn]) == 0)
```

```
        ip->addrs[bn] = addr = balloc(ip->dev);
```

```
    return addr;
```

```
}
```

```
...
```



0
1
2
...
139

## inode

metadata

Direct pointers

⋮

single indirect

double indirect

# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

```
bmap(struct inode *ip, uint bn)
```

```
uint addr, *a;
```

```
struct buf *bp;
```

```
if(bn < NDIRECT){
```

```
    if((addr = ip->addrs[bn]) == 0)
```

Is the pointer  
empty?

```
        ip->addrs[bn] = addr = balloc(ip->dev);
```

```
    return addr;
```

```
}
```

```
...
```

0
1
2
...
139

## inode

metadata

Direct pointers

⋮

single indirect

double indirect



# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

```
bmap(struct inode *ip, uint bn)
```

```
uint addr, *a;
```

```
struct buf *bp;
```

```
if(bn < NDIRECT){
```

```
    if((addr = ip->addrs[bn]) == 0)
```

```
        ip->addrs[bn] = addr = balloc(ip->dev);
```

```
    return addr;
```

```
}
```

```
...
```

If it is empty  
create a new  
block



0
1
2
...
139

## inode

metadata

Direct pointers

⋮

single indirect

double indirect

# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

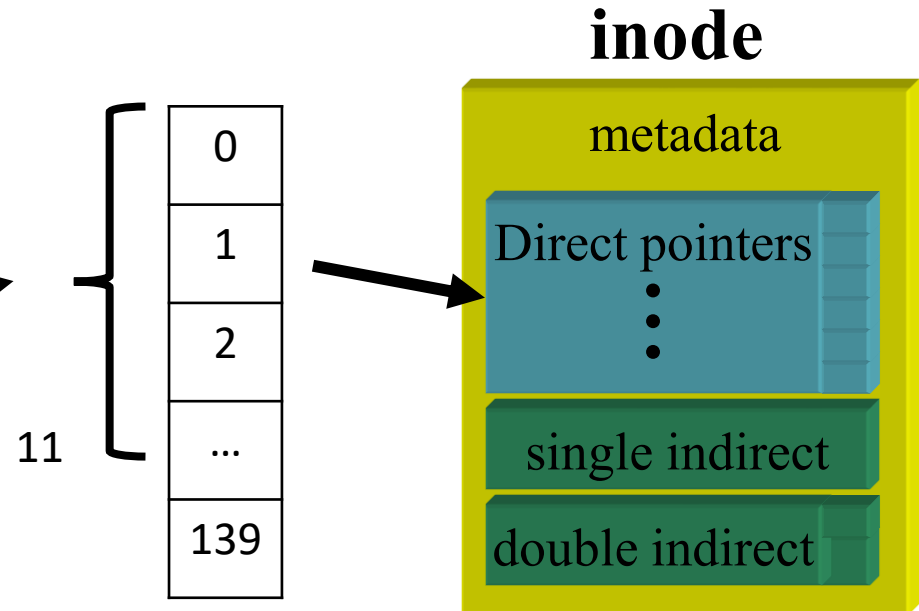
```
bmap(struct inode *ip, uint bn)
```

```
uint addr, *a;
```

```
struct buf *bp;
```

```
if(bn < NDIRECT){
```

```
...
```



# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

```
bmap(struct inode *ip, uint bn)
```

```
uint addr, *a;
```

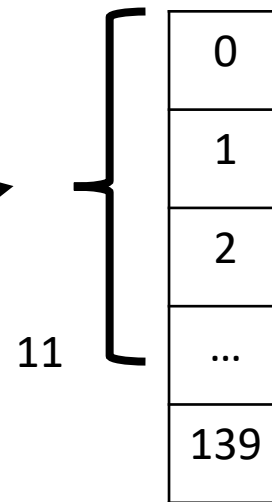
```
struct buf *bp;
```

```
if(bn < NDIRECT){
```

```
...
```

```
bn -= NDIRECT;
```

```
if(bn < NDINDIRECT){
```



**inode**

metadata

Direct pointers

⋮

single indirect

double indirect

# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

```
bmap(struct inode *ip, uint bn)
```

```
uint addr, *a;
```

```
struct buf *bp;
```

```
if(bn < NDIRECT){
```

```
...
```

```
bn -= NDIRECT;
```

```
if(bn < NDINDIRECT){
```

12



0
1
2
...
139

## inode

metadata

Direct pointers

⋮

single indirect

double indirect

# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

```
bmap(struct inode *ip, uint bn)
```

```
uint addr, *a;
```

```
struct buf *bp;
```

```
if(bn < NDIRECT){
```

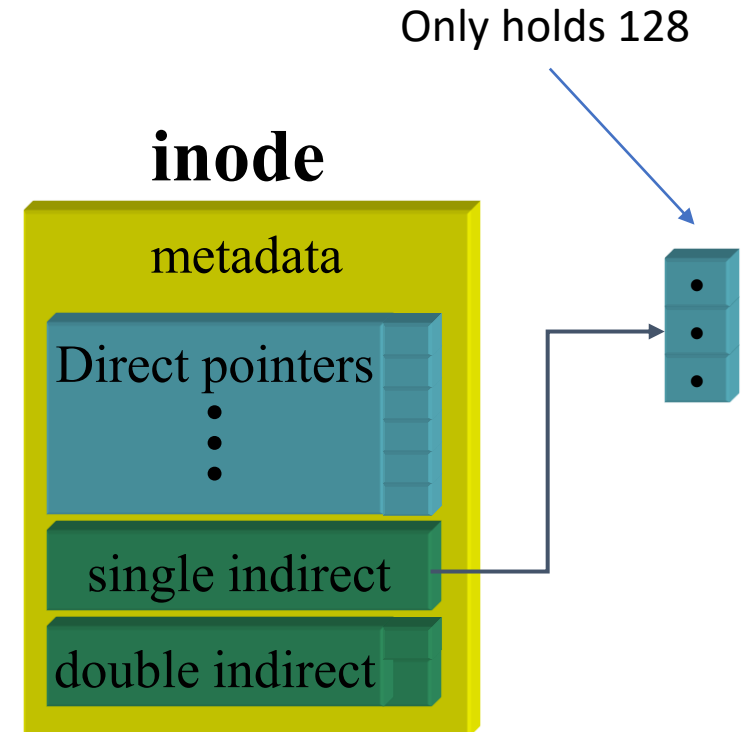
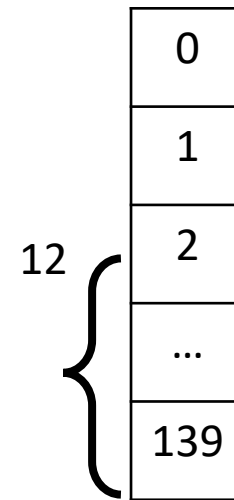
```
...
```

```
bn -= NDIRECT;
```

```
if(bn < NDINDIRECT){
```



We map 12 - 139  
to 0 - 127



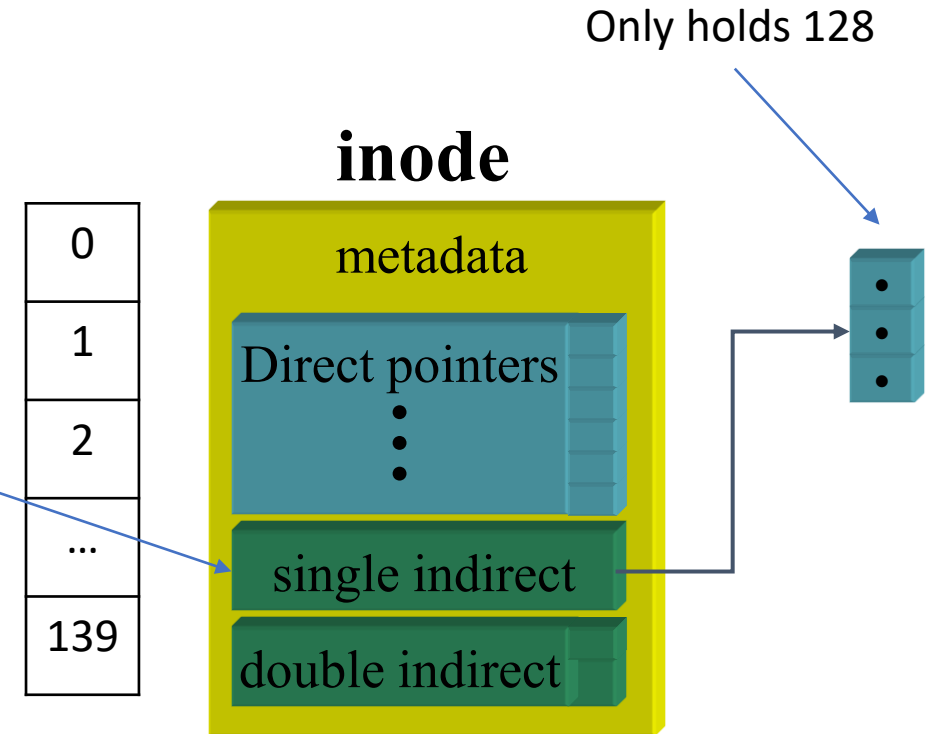
# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

// Load indirect block, allocating if necessary.

```
if((addr = ip->addrs[NDIRECT+1]) == 0)
    ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
bp = bread(ip->dev, addr);
a = (uint*)bp->data;

if((addr = a[bn]) == 0){
    a[bn] = addr = balloc(ip->dev);
    log_write(bp);
}
```



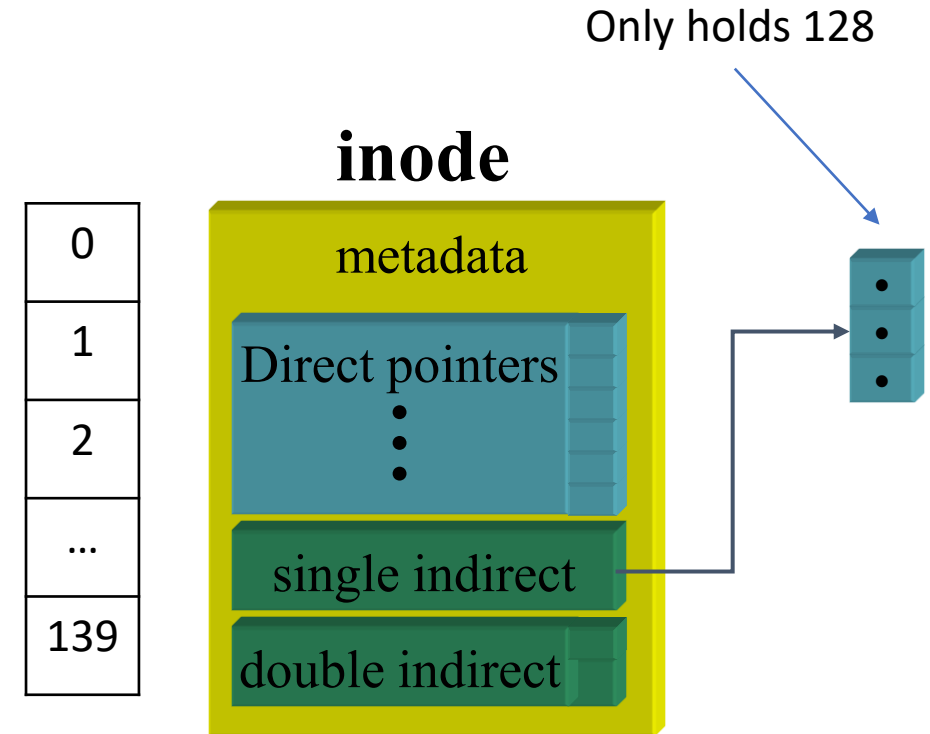
# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

// Load indirect block, allocating if necessary.

```
if((addr = ip->addrs[NDIRECT+1]) == 0)
    ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
bp = bread(ip->dev, addr);
a = (uint*)bp->data;

if((addr = a[bn]) == 0){
    a[bn] = addr = balloc(ip->dev);
    log_write(bp);
}
```



# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

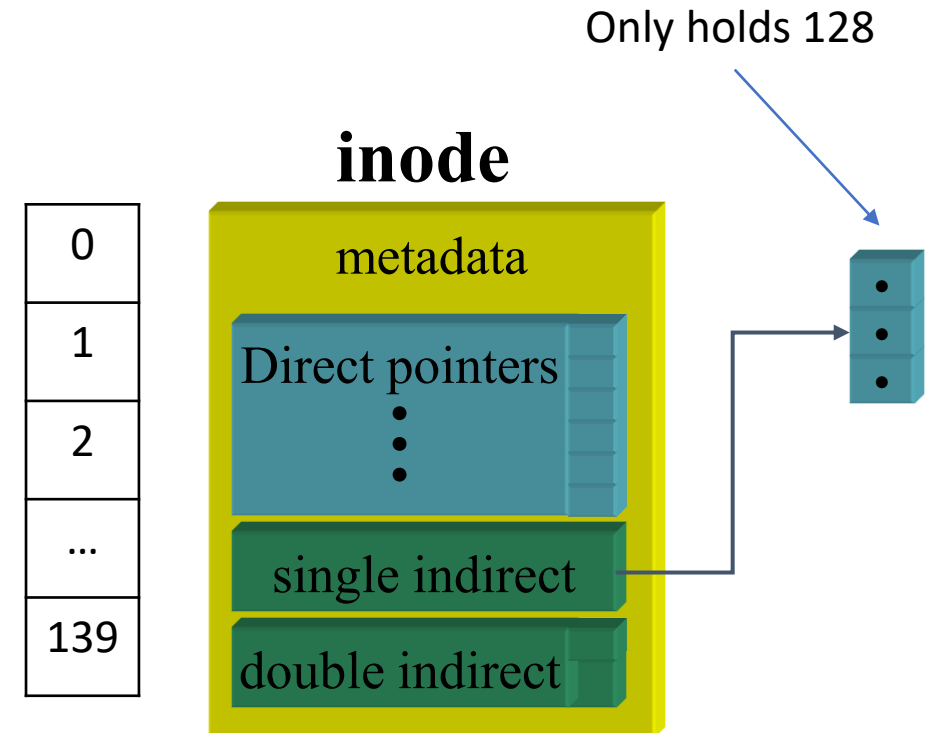
// Load indirect block, allocating if necessary.

```
if((addr = ip->addrs[NDIRECT+1]) == 0)
    ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
```

```
bp = bread(ip->dev, addr);
```

```
a = (uint*)bp->data;
```

```
if((addr = a[bn]) == 0){
    a[bn] = addr = balloc(ip->dev);
    log_write(bp);
}
```





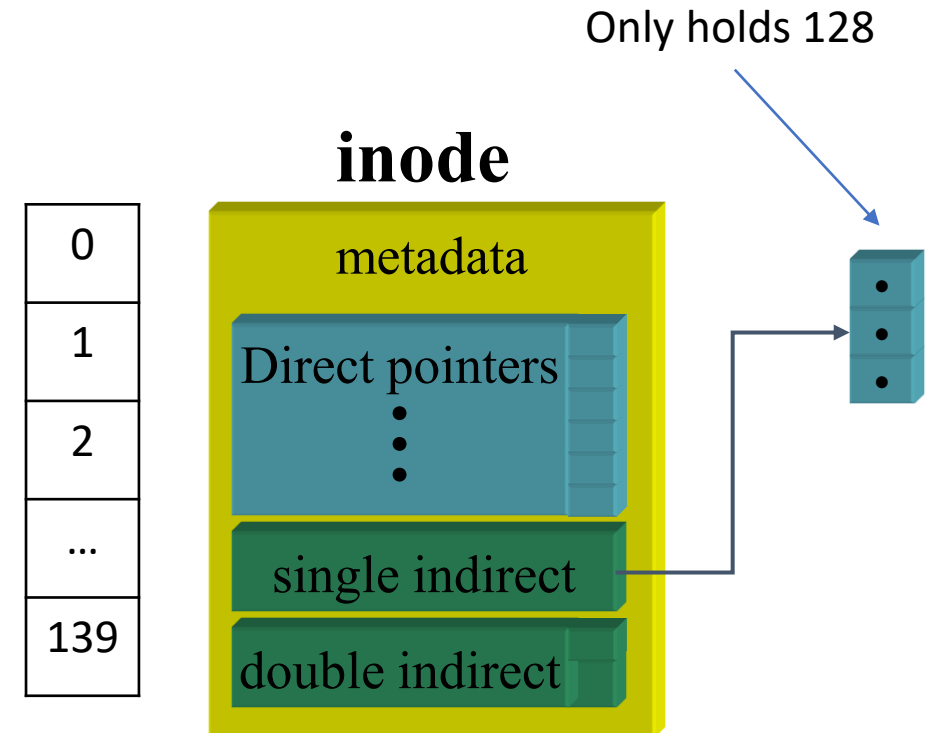
# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

// Load indirect block, allocating if necessary.

```
if((addr = ip->addrs[NDIRECT+1]) == 0)
    ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
bp = bread(ip->dev, addr);
a = (uint*)bp->data;

if((addr = a[bn]) == 0){
    a[bn] = addr = balloc(ip->dev);
    log_write(bp);
}
```



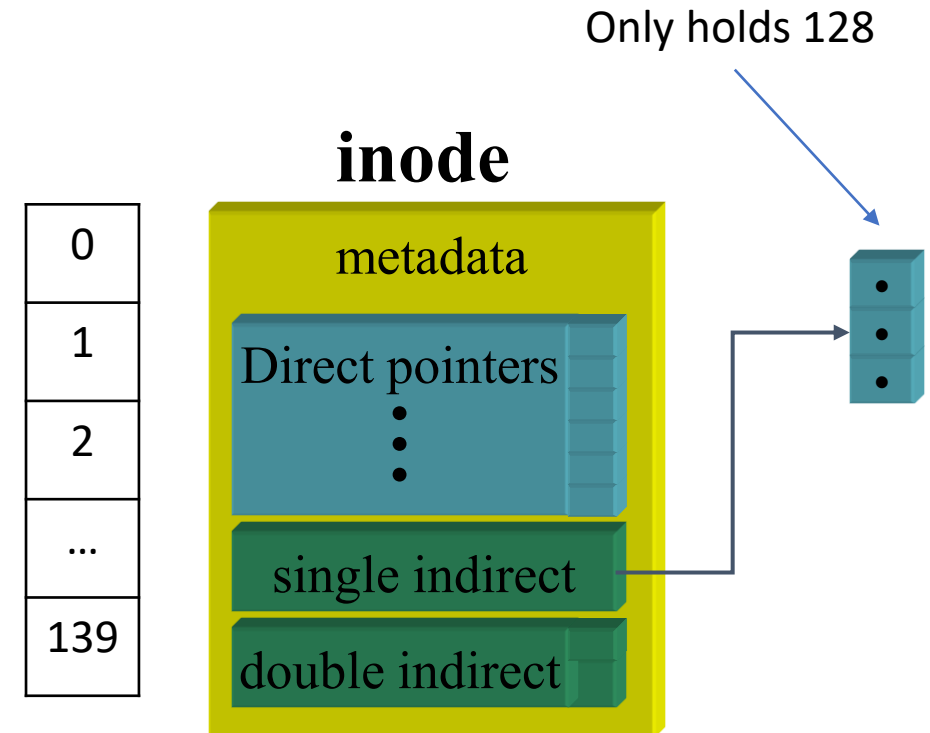
# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

// Load indirect block, allocating if necessary.

```
if((addr = ip->addrs[NDIRECT+1]) == 0)
    ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
bp = bread(ip->dev, addr);
a = (uint*)bp->data;

if((addr = a[bn]) == 0){
    a[bn] = addr = balloc(ip->dev);
    log_write(bp);
}
```



# Lab 5 – Bigger Files for xv6

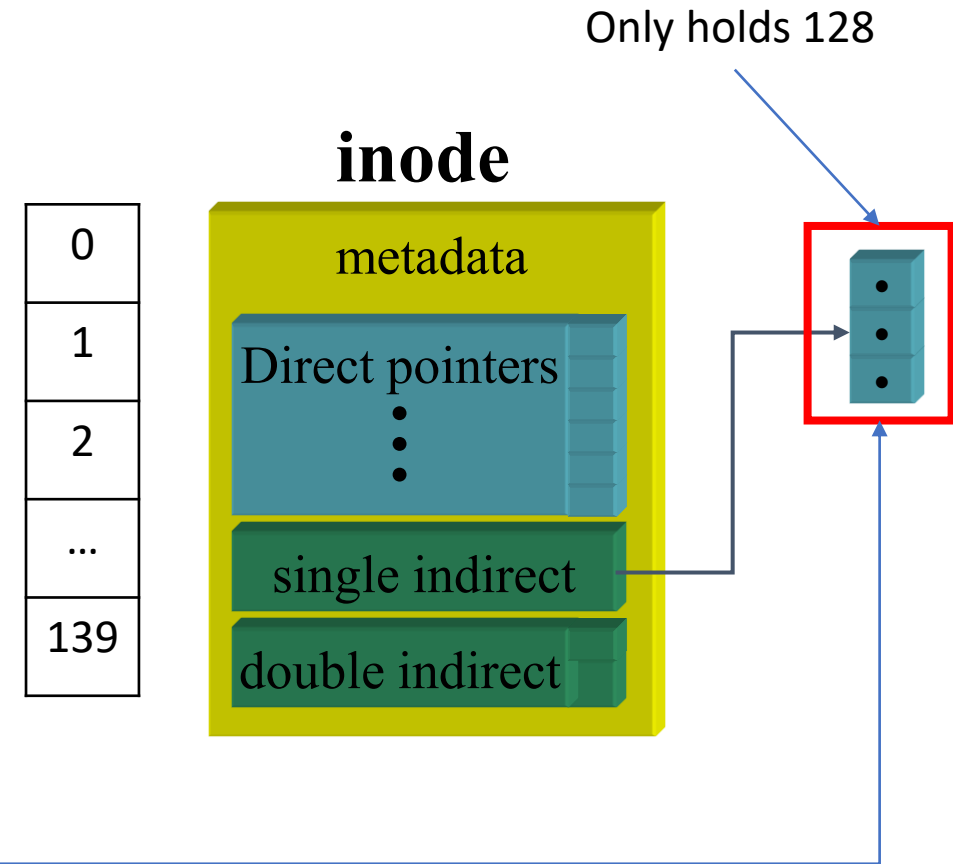
- In the **bmap()** from the **fs.c** file

// Load indirect block, allocating if necessary.

```
if((addr = ip->addrs[NDIRECT+1]) == 0)
    ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);
bp = bread(ip->dev, addr);
a = (uint*)bp->data;
```

```
if((addr = a[bn]) == 0){
    a[bn] = addr = balloc(ip->dev);
    log_write(bp);
}
```

We are checking if  
the index in the  
block from the  
table we just read  
is empty

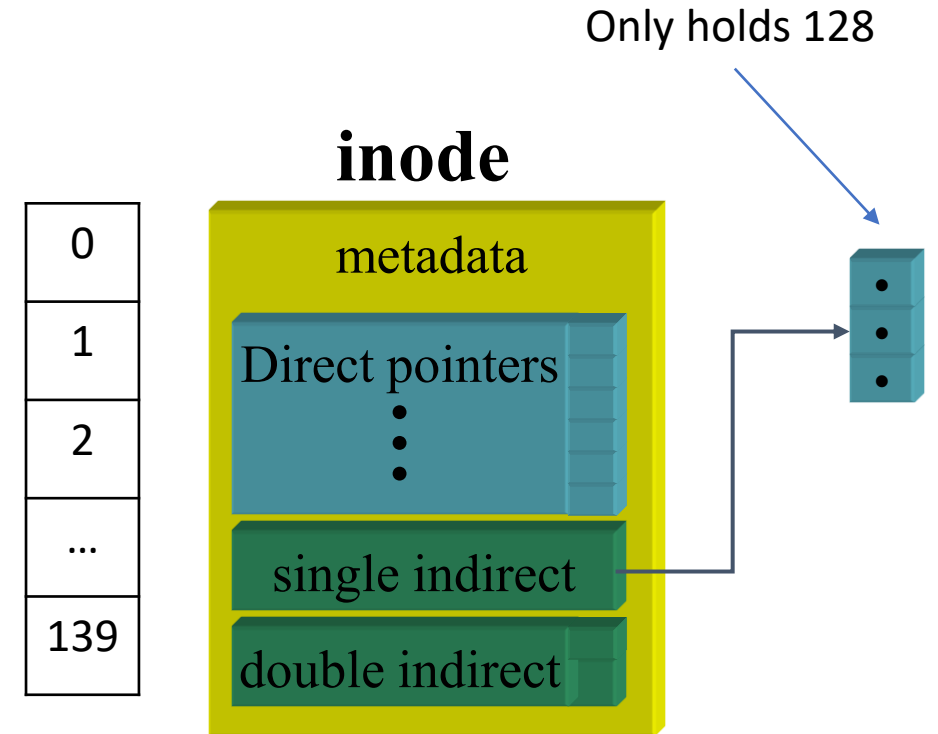


# Lab 5 – Bigger Files for xv6

- In the **bmap()** from the **fs.c** file

// Load indirect block, allocating if necessary.

```
...  
if((addr = a[bn]) == 0){  
    a[bn] = addr = balloc(ip->dev);  
    log_write(bp);  
}  
brelse(bp);  
return addr;
```



# Lab 5 – Bigger Files for xv6 – hints

---

- If you change the definition of NDIRECT, you'll probably have to change the size of `addrs[]` in `struct inode` in `file.h`. Make sure that `struct inode` and `struct dinode` have the same number of elements in their `addrs[]` arrays.

```
// in-memory copy of an inode
struct inode {
    uint dev;           // Device number
    uint inum;          // Inode number
    int ref;            // Reference count
    struct sleeplock lock; // protects everything below here
    int valid;          // inode has been read from disk?

    short type;         // copy of disk inode
    short major;
    short minor;
    short nlink;
    uint size;
    uint addrs[NDIRECT+1];
};
```

## Lab 5 – Bigger Files for xv6 – hints

---

- If you change the definition of NDIRECT, make sure to create a new fs.img, since mkfs uses NDIRECT too to build the initial file systems. If you delete fs.img, make on Unix (not xv6) will build a new one for you.
- If your file system gets into a bad state, perhaps by crashing, delete fs.img (do this from Unix, not xv6). make will build a new clean file system image for you.
- Don't forget to brelse() each block that you bread(). brelse() releases the buffer cache for the block (check bio.c).

# Lab 5 – Bigger Files for xv6 – bmap()

---

- If all goes well, big will now report that it can write **16523** sectors. It will take big a few dozen seconds to finish.
  - $11 + 128 + 128 * 128$

# Project 4 – Discussion

---

- Use **FUSE** to create our own file system.
- A **file** will represent our disk device.



# Project 4 – Discussion

---

- FUSE Setup
  - `cd /u/OSLab/USERNAME`
  - `cp /u/OSLab/original/fuse-2.7.0.tar.gz .`
  - `tar xvfz fuse-2.7.0.tar.gz`
  - `cd fuse-2.7.0`
  - `./configure`
  - `make`

# Project 4 – Discussion

---

- FUSE Setup
  - `cd /u/OSLab/USERNAME`
  - `cp /u/OSLab/original/fuse-2.7.0.tar.gz .`
  - `tar xvfz fuse-2.7.0.tar.gz`
  - `cd fuse-2.7.0`
  - **`./configure`**
  - **`make`**

# Project 4 – Discussion

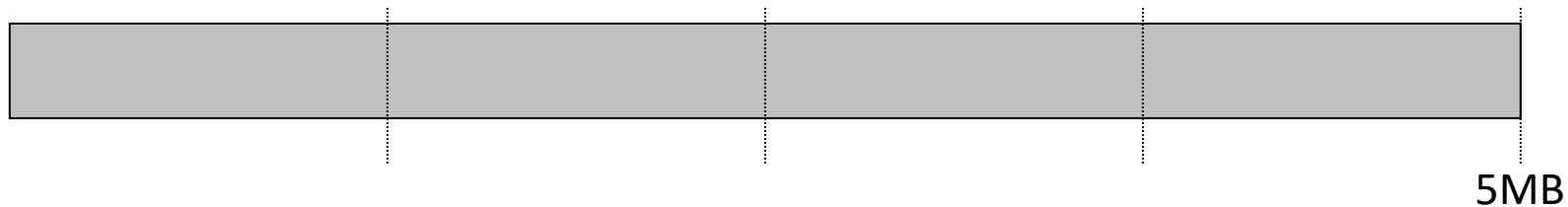
---

- Keep track of free / allocated memory regions with a **bitmap**
  - One bit in map corresponds to a fixed-size region of memory
  - **Bitmap** is a constant size for a given amount of memory regardless of how much is allocated at a particular time

# Project 4 – Discussion

---

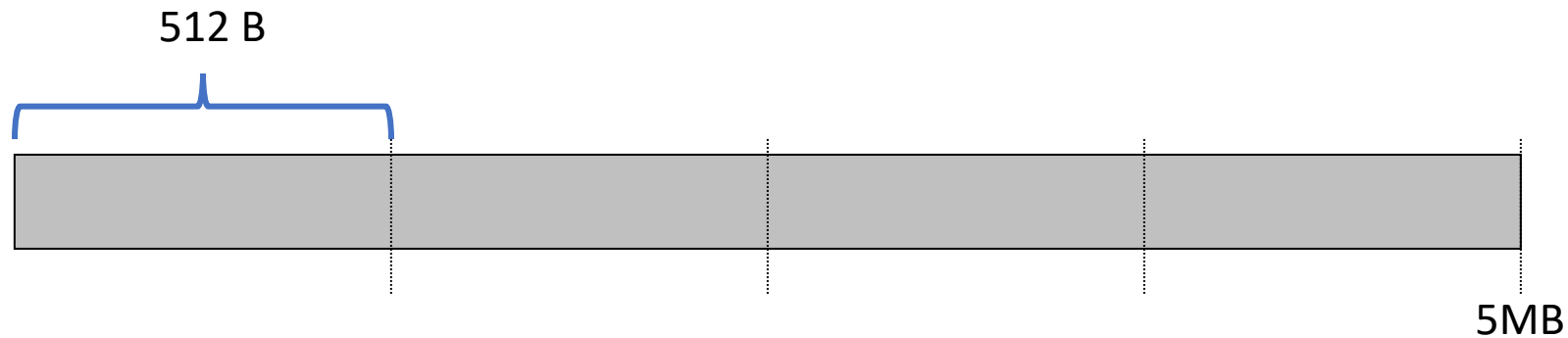
- Keep track of free / allocated memory regions with a **bitmap**
  - One bit in map corresponds to a fixed-size region of memory
  - **Bitmap** is a constant size for a given amount of memory regardless of how much is allocated at a particular time



# Project 4 – Discussion

---

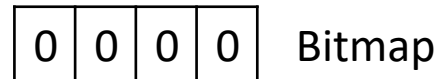
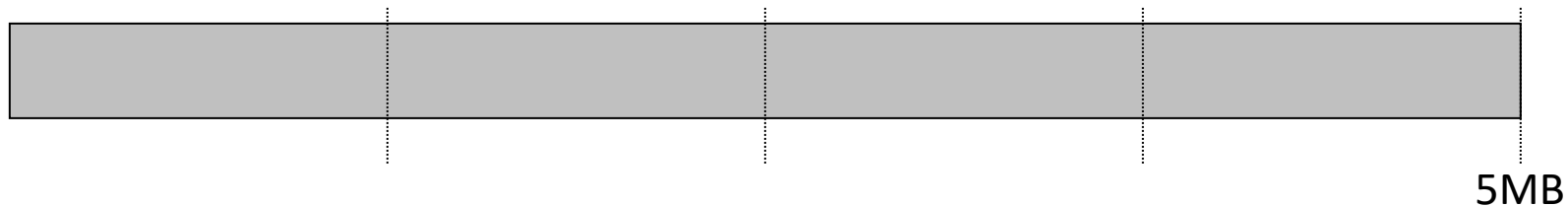
- Keep track of free / allocated memory regions with a **bitmap**
  - One bit in map corresponds to a fixed-size region of memory
  - **Bitmap** is a constant size for a given amount of memory regardless of how much is allocated at a particular time



# Project 4 – Discussion

---

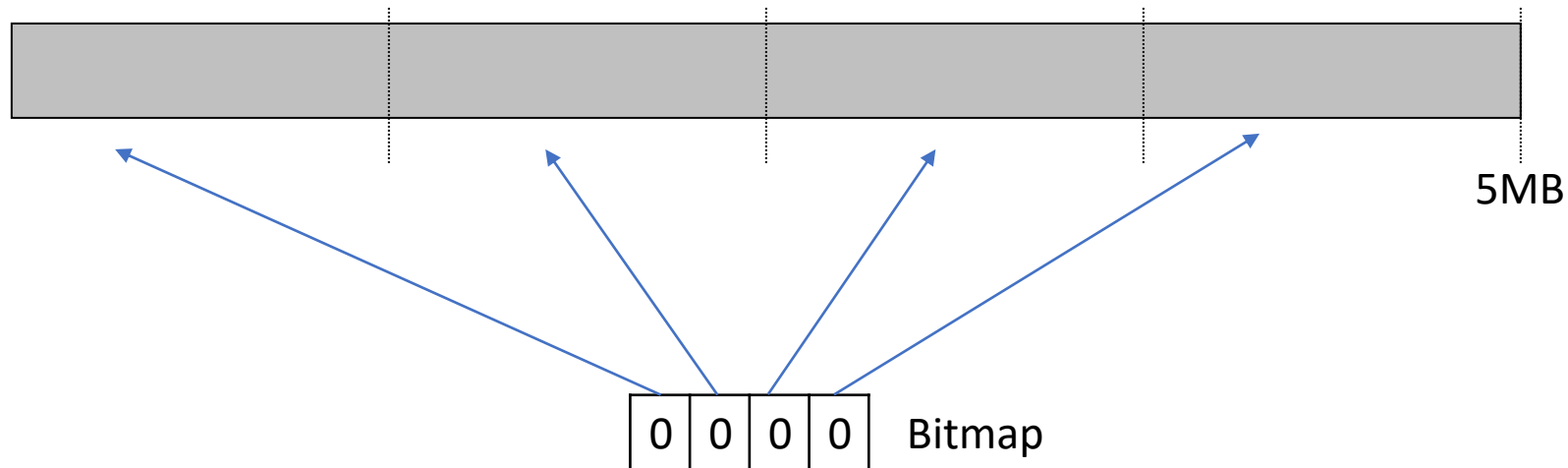
- Keep track of free / allocated memory regions with a **bitmap**
  - One bit in map corresponds to a fixed-size region of memory
  - **Bitmap** is a constant size for a given amount of memory regardless of how much is allocated at a particular time



# Project 4 – Discussion

---

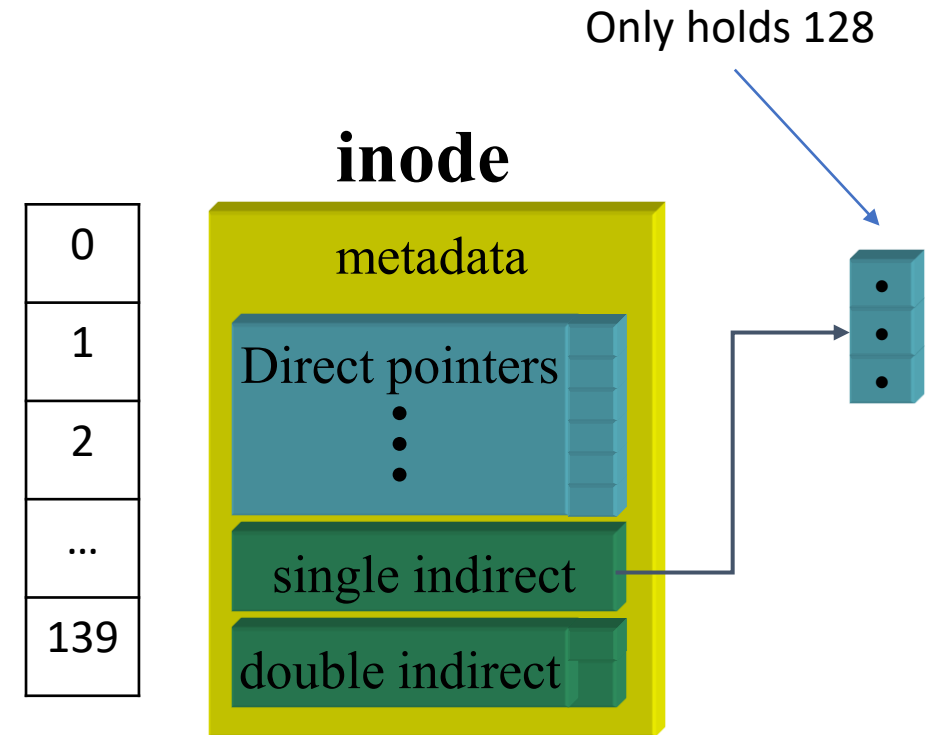
- Keep track of free / allocated memory regions with a **bitmap**
  - One bit in map corresponds to a fixed-size region of memory
  - **Bitmap** is a constant size for a given amount of memory regardless of how much is allocated at a particular time



# Lab 5 – Discussion

- Allocating blocks

```
...  
if((addr = a[bn]) == 0){  
    a[bn] = addr = balloc(ip->dev);  
    log_write(bp);  
}  
brelse(bp);  
return addr;
```

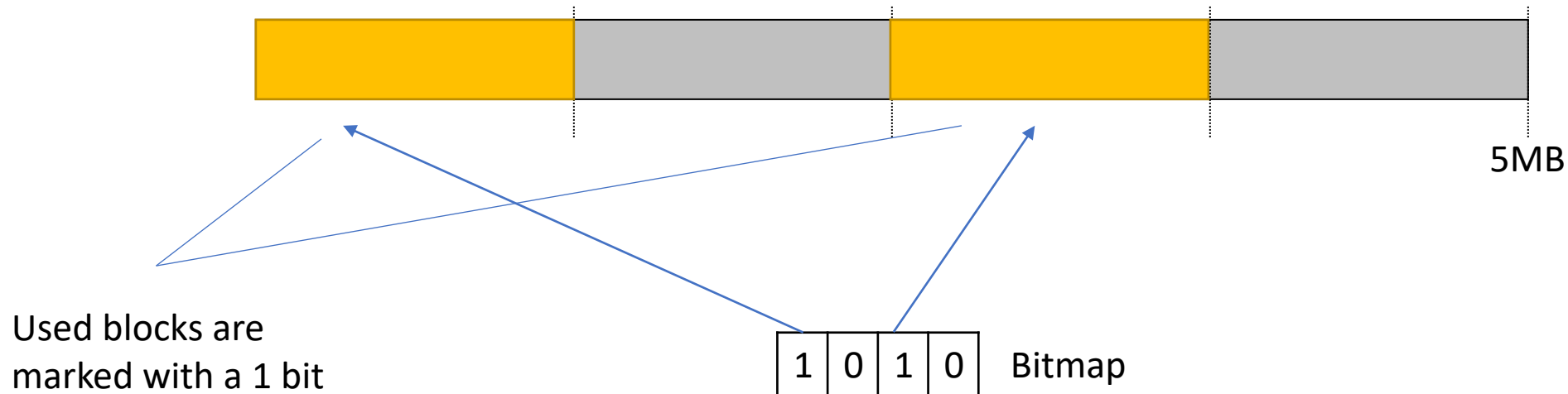




# Project 4 – Discussion

---

- Keep track of free / allocated memory regions with a **bitmap**
  - One bit in map corresponds to a fixed-size region of memory
  - **Bitmap** is a constant size for a given amount of memory regardless of how much is allocated at a particular time





# CS 1550

Week 13

—

Lab 5

Teaching Assistant

Henrique Potter