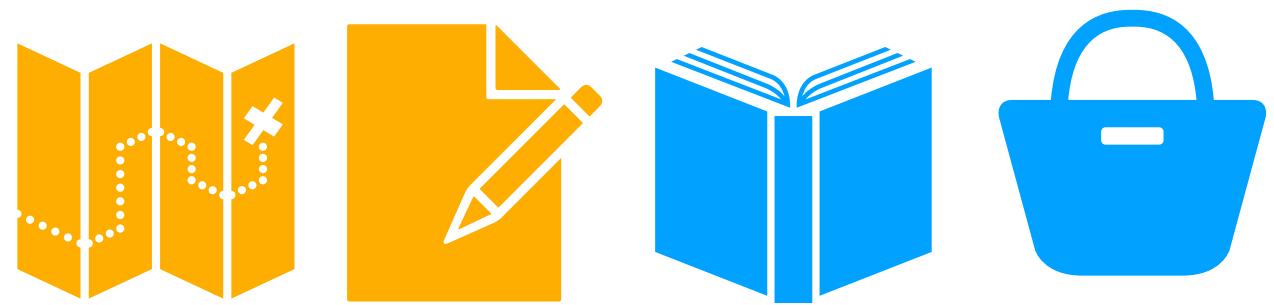
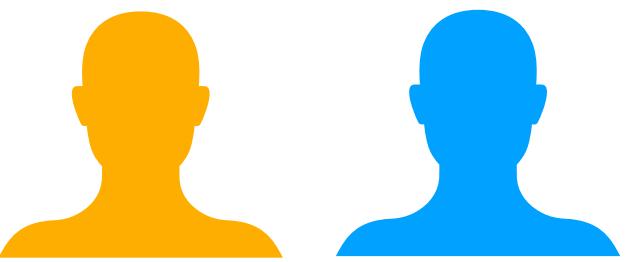


File system

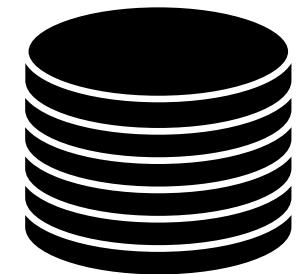
Abhilash Jindal

File system



Operating system

Computer hardware



1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

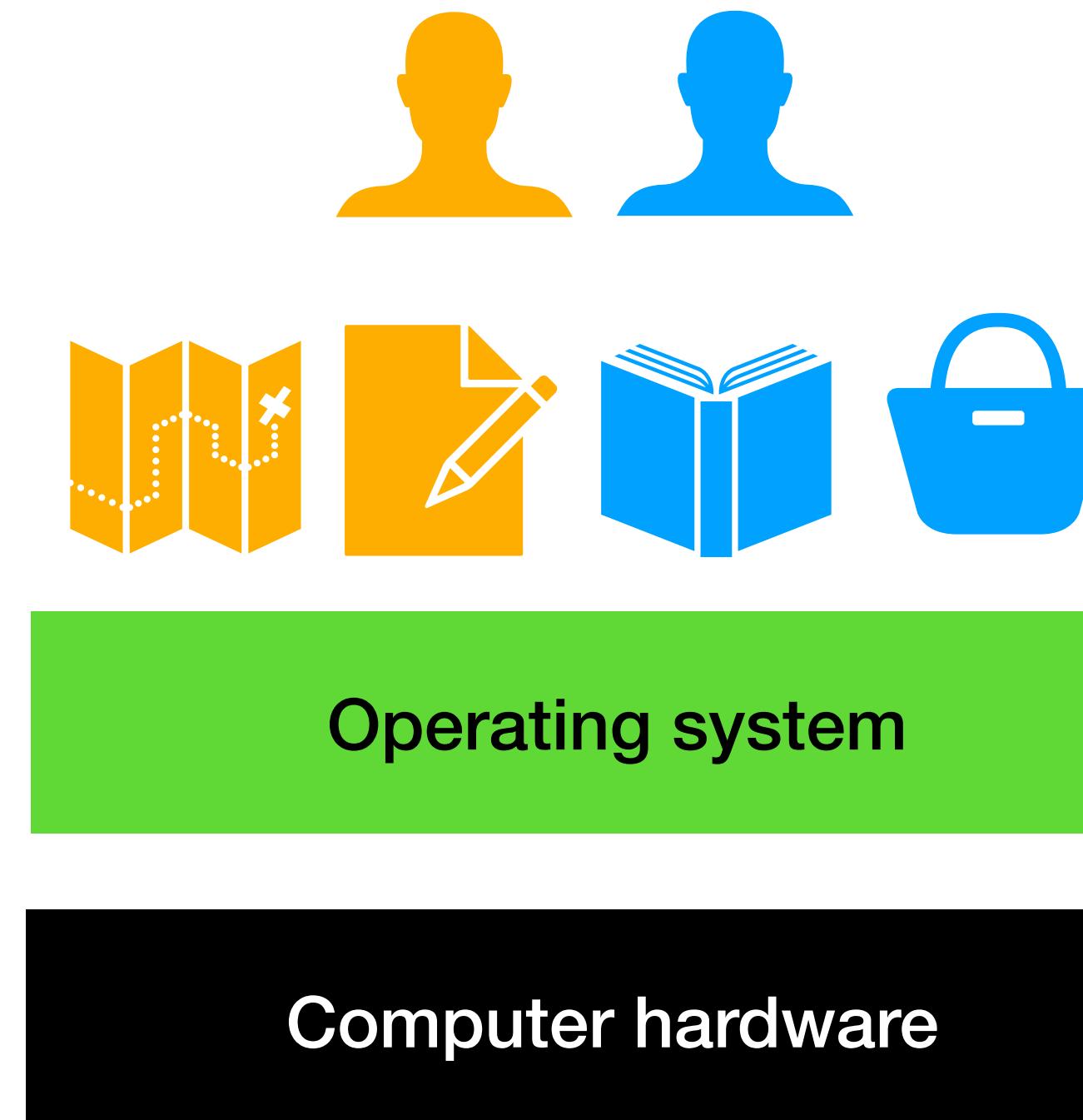
File system

Example: io.c

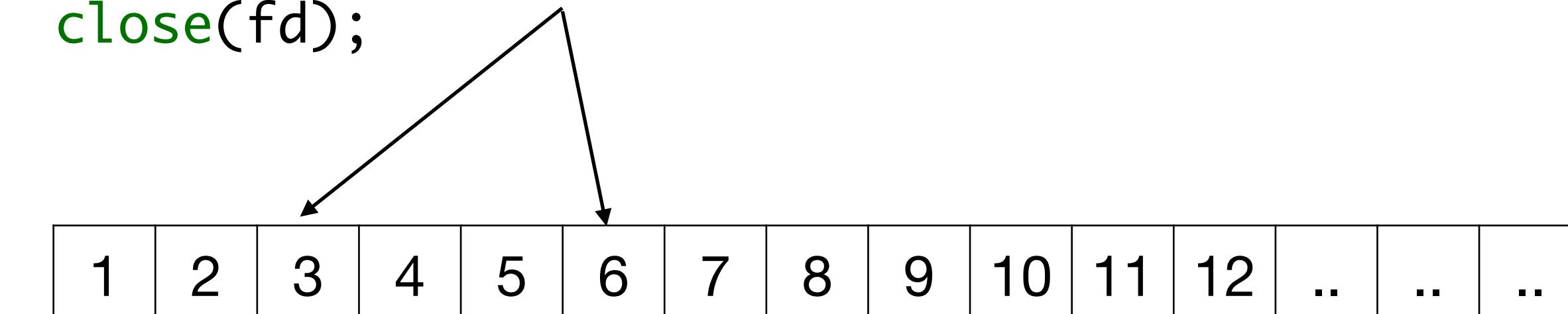
Disk interface: List of blocks

File system OS interface: Folders and files.

open, write, read, lseek, close, append,
permissions, truncation, file descriptor offset

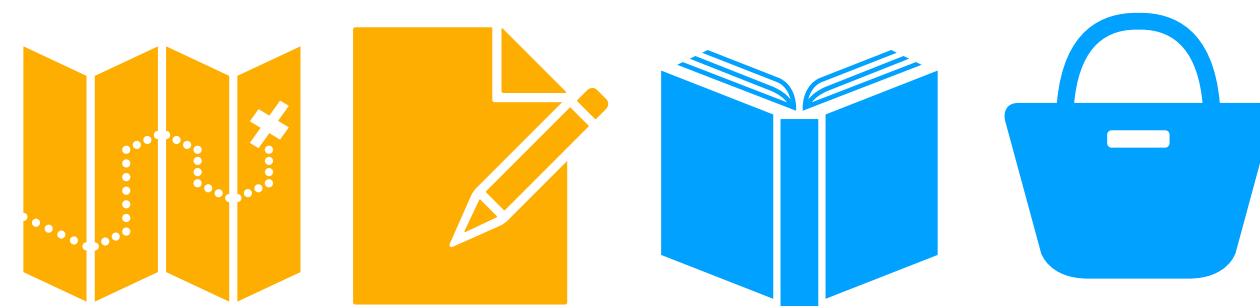
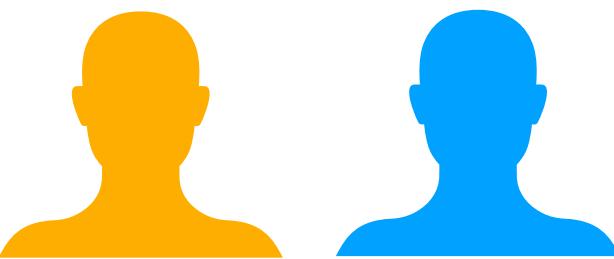


```
int fd = open("/tmp/file", O_WRONLY | O_CREAT);  
int rc = write(fd, "hello world\n", 12);  
close(fd);
```



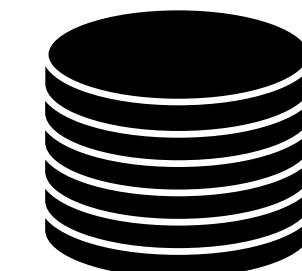
Why file system?

Why not just multiplex disk blocks like memory?



Operating system

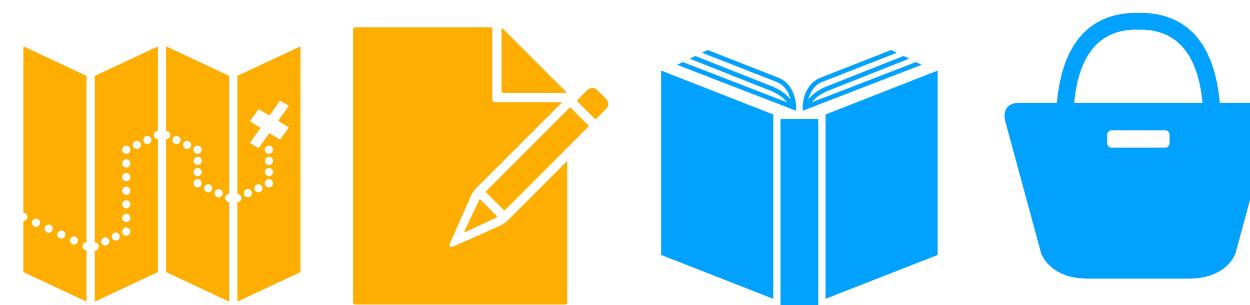
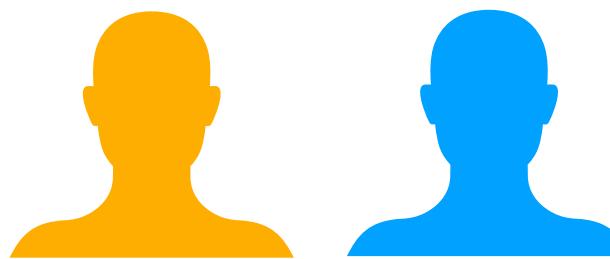
Computer hardware



Why file system?

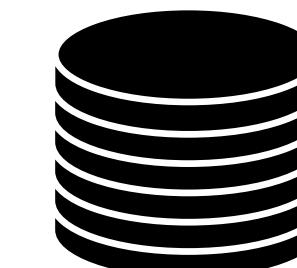
Why not just multiplex disk blocks like memory?

- Disk blocks live after programs exits, computer restarts



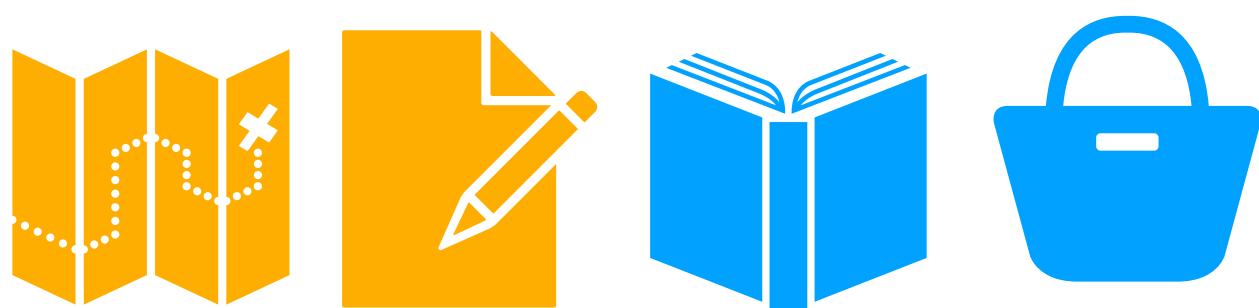
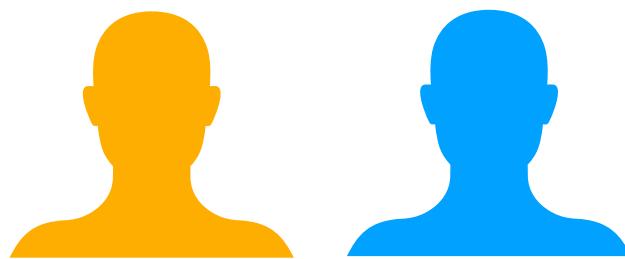
Operating system

Computer hardware



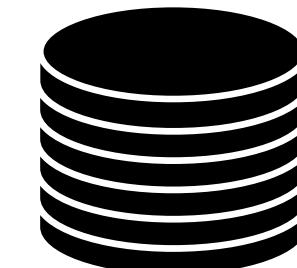
Why file system?

Why not just multiplex disk blocks like memory?



Operating system

Computer hardware



- Disk blocks live after programs exits, computer restarts
- Different programs read / write same file
 - vim writes io.c
 - gcc reads io.c, write io
 - We finally run io

Files as sequence of bytes

Files as sequence of bytes

- Other options: Files have structured records

Files as sequence of bytes

- Other options: Files have structured records
 - Can build structure on top

Files as sequence of bytes

- Other options: Files have structured records
 - Can build structure on top
 - But may not optimise disk accesses

Files as sequence of bytes

- Other options: Files have structured records
 - Can build structure on top
 - But may not optimise disk accesses
- Also expose raw disk blocks

Files as sequence of bytes

- Other options: Files have structured records
 - Can build structure on top
 - But may not optimise disk accesses
- Also expose raw disk blocks
 - Databases

Files as sequence of bytes

- Other options: Files have structured records
 - Can build structure on top
 - But may not optimise disk accesses
- Also expose raw disk blocks
 - Databases
 - File system checker (fsck)

Files as sequence of bytes

- Other options: Files have structured records
 - Can build structure on top
 - But may not optimise disk accesses
- Also expose raw disk blocks
 - Databases
 - File system checker (fsck)
 - Disk defragmenter

Files as sequence of bytes

- Other options: Files have structured records
 - Can build structure on top
 - But may not optimise disk accesses
- Also expose raw disk blocks
 - Databases
 - File system checker (fsck)
 - Disk defragmenter

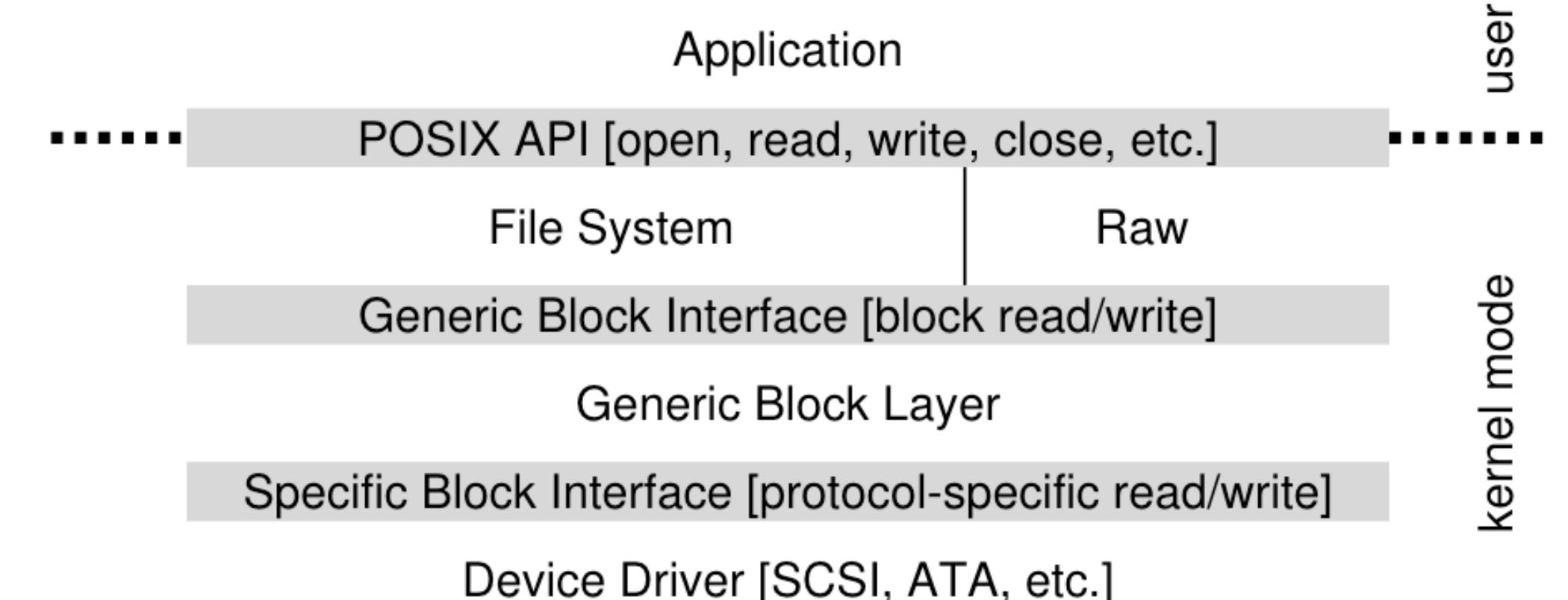


Figure 36.4: The File System Stack

Flexible abstraction

- Stitch multiple file systems into a common directory tree
 - `mount -t ext3 /dev/sda1 /home/abhilash/photos/`
 - `mount -t ext2 /dev/sdb1 /home/abhilash/docs/`
- `/proc`
- Run `tty. cat <filename>`
- `/sys`

Agenda

Agenda

- Build a file system (OSTEP Ch. 40, xv6 Ch. 6)
 - On-disk data structure. Organize disk blocks to expose files and directories

Agenda

- Build a file system (OSTEP Ch. 40, xv6 Ch. 6)
 - On-disk data structure. Organize disk blocks to expose files and directories
- Optimizations (OSTEP Ch. 41)

Agenda

- Build a file system (OSTEP Ch. 40, xv6 Ch. 6)
 - On-disk data structure. Organize disk blocks to expose files and directories
- Optimizations (OSTEP Ch. 41)
- Crash consistency: Don't lose data when computer restarts (OSTEP Ch. 42)

File system characteristics

File system characteristics

- File system contains lots of files ~100K

File system characteristics

- File system contains lots of files ~100K
- Most files are small ~2KB

File system characteristics

- File system contains lots of files ~100K
- Most files are small ~2KB
- A few big files use most of the disk space

File system characteristics

- File system contains lots of files ~100K
- Most files are small ~2KB
- A few big files use most of the disk space
- Directories have typically < 20 files and directories

xv6 file system

File system implementation , OSTEP Ch.40, xv6 Ch. 6

How to store files?

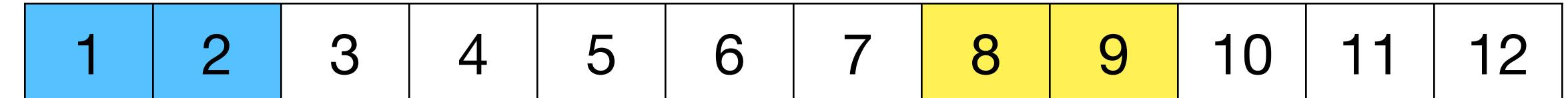
Contiguous allocation



- “a.txt” -> (base = 1, size = 2)
- “b.txt” -> (base = 8, size = 2)

How to store files?

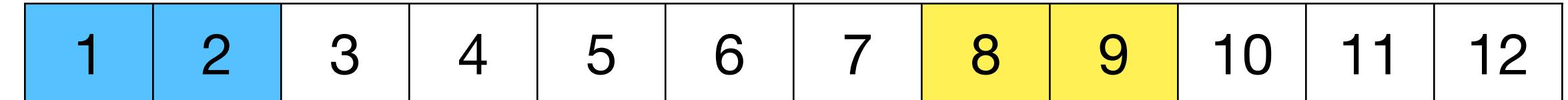
Contiguous allocation



- “a.txt” -> (base = 1, size = 2)
 - “b.txt” -> (base = 8, size = 2)
- Growth. “b.txt” wants to use 6 blocks. Need to copy to a new location.

How to store files?

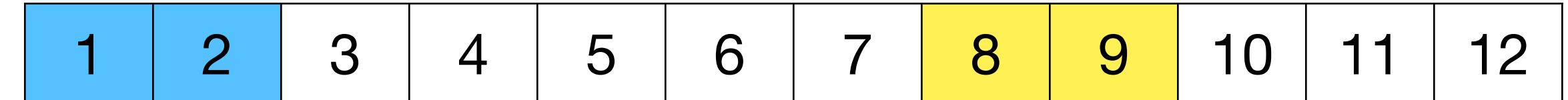
Contiguous allocation



- “a.txt” -> (base = 1, size = 2)
 - “b.txt” -> (base = 8, size = 2)
-
- Growth. “b.txt” wants to use 6 blocks. Need to copy to a new location.
 - Fragmentation. Want to create a file “c.txt” with 6 blocks.

How to store files?

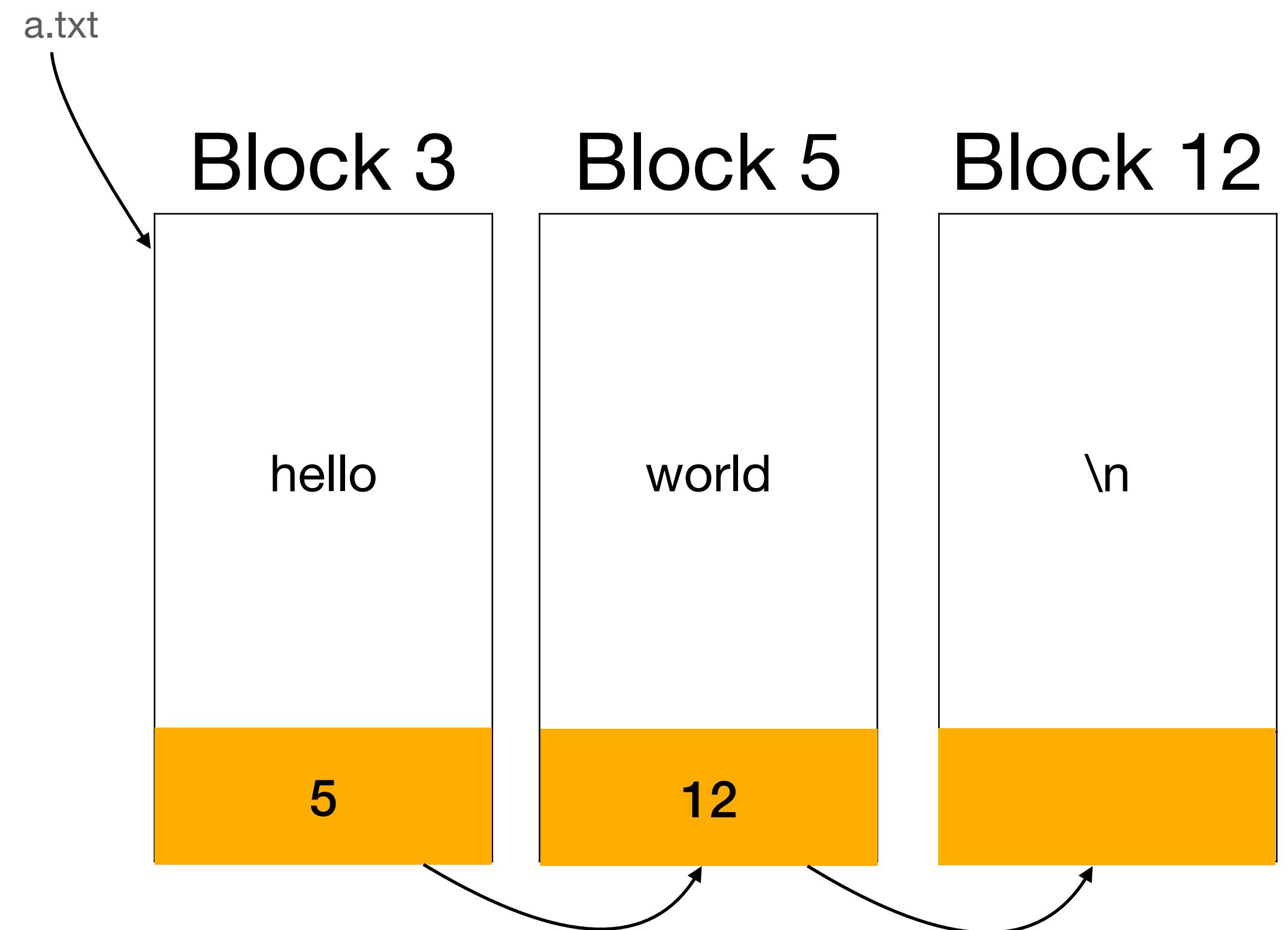
Contiguous allocation



- “a.txt” -> (base = 1, size = 2)
 - “b.txt” -> (base = 8, size = 2)
-
- Growth. “b.txt” wants to use 6 blocks. Need to copy to a new location.
 - Fragmentation. Want to create a file “c.txt” with 6 blocks.
- ✓ Sequential file rw is sequential disk rw

How to store files?

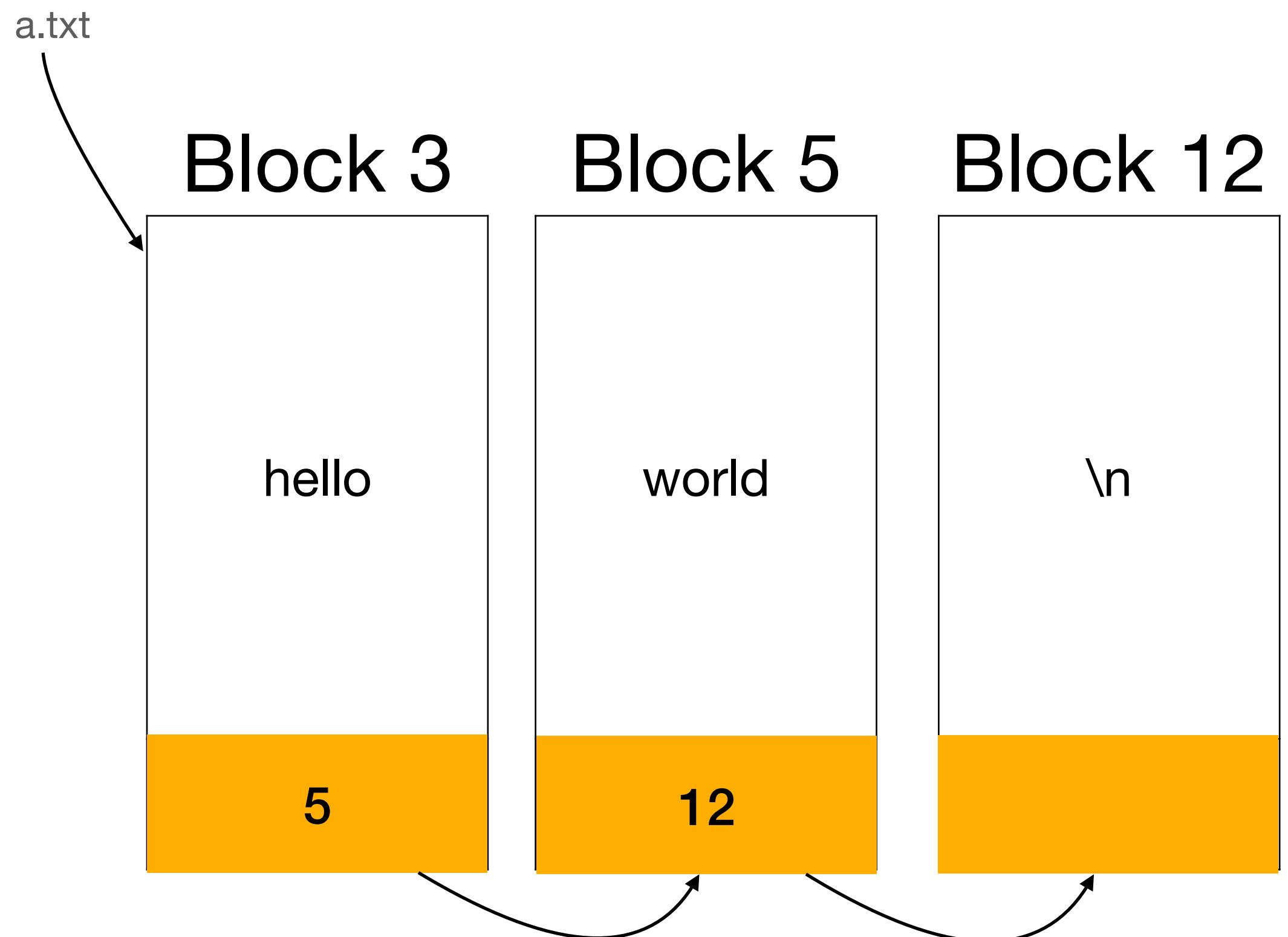
Linked list of blocks



How to store files?

Linked list of blocks

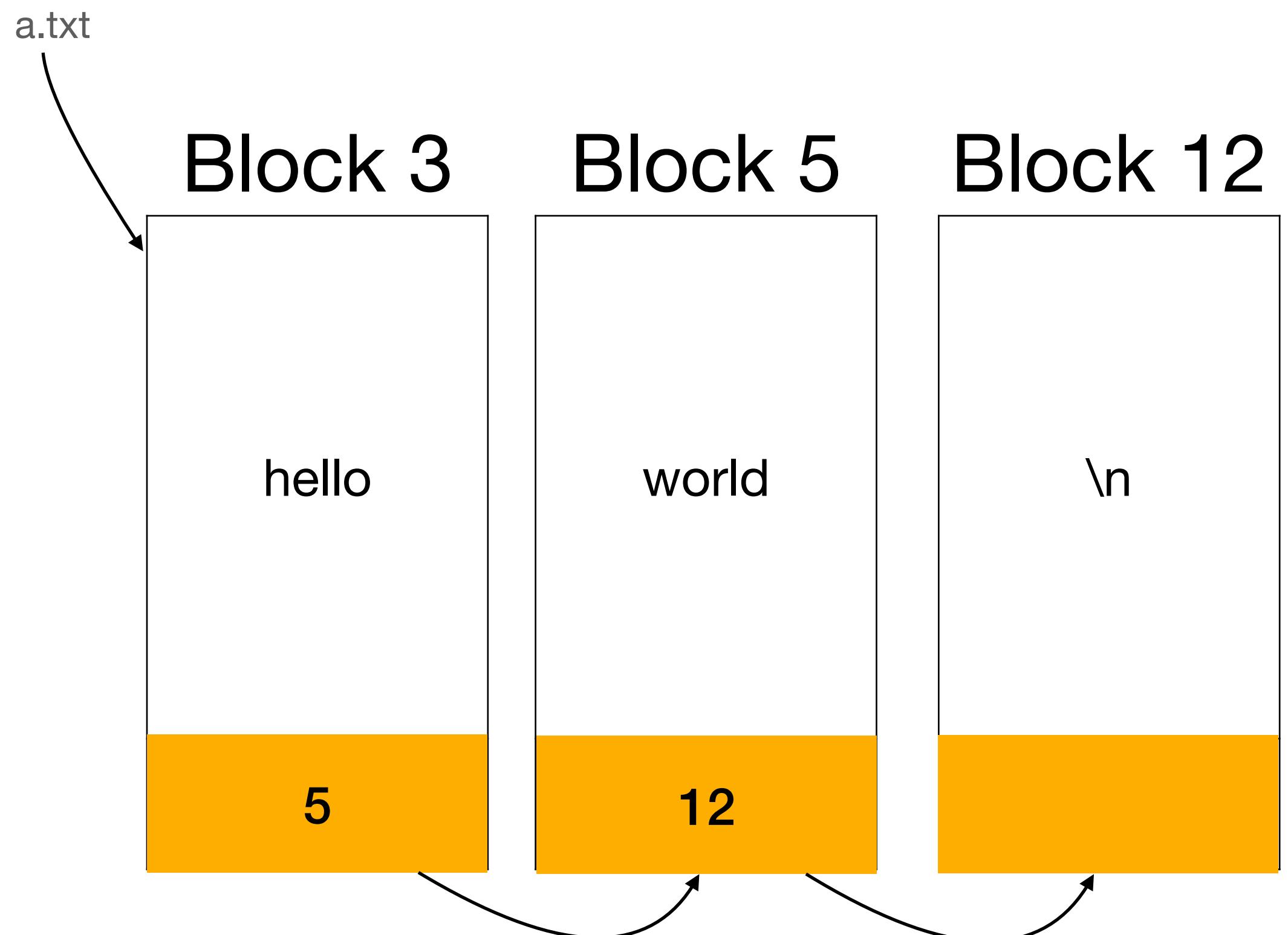
✓ Files can grow easily



How to store files?

Linked list of blocks

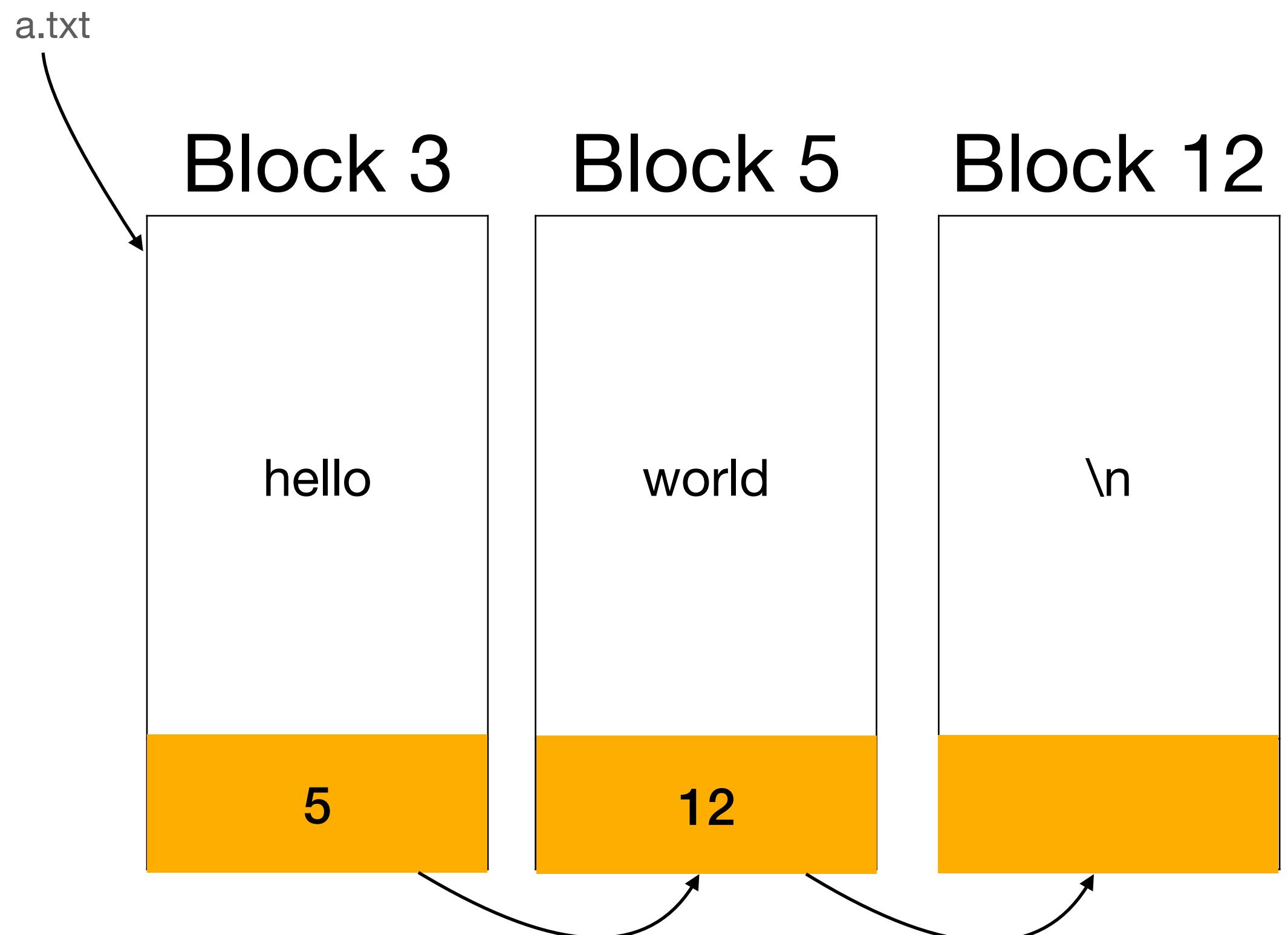
- ✓ Files can grow easily
- Seek / appends are terrible:



How to store files?

Linked list of blocks

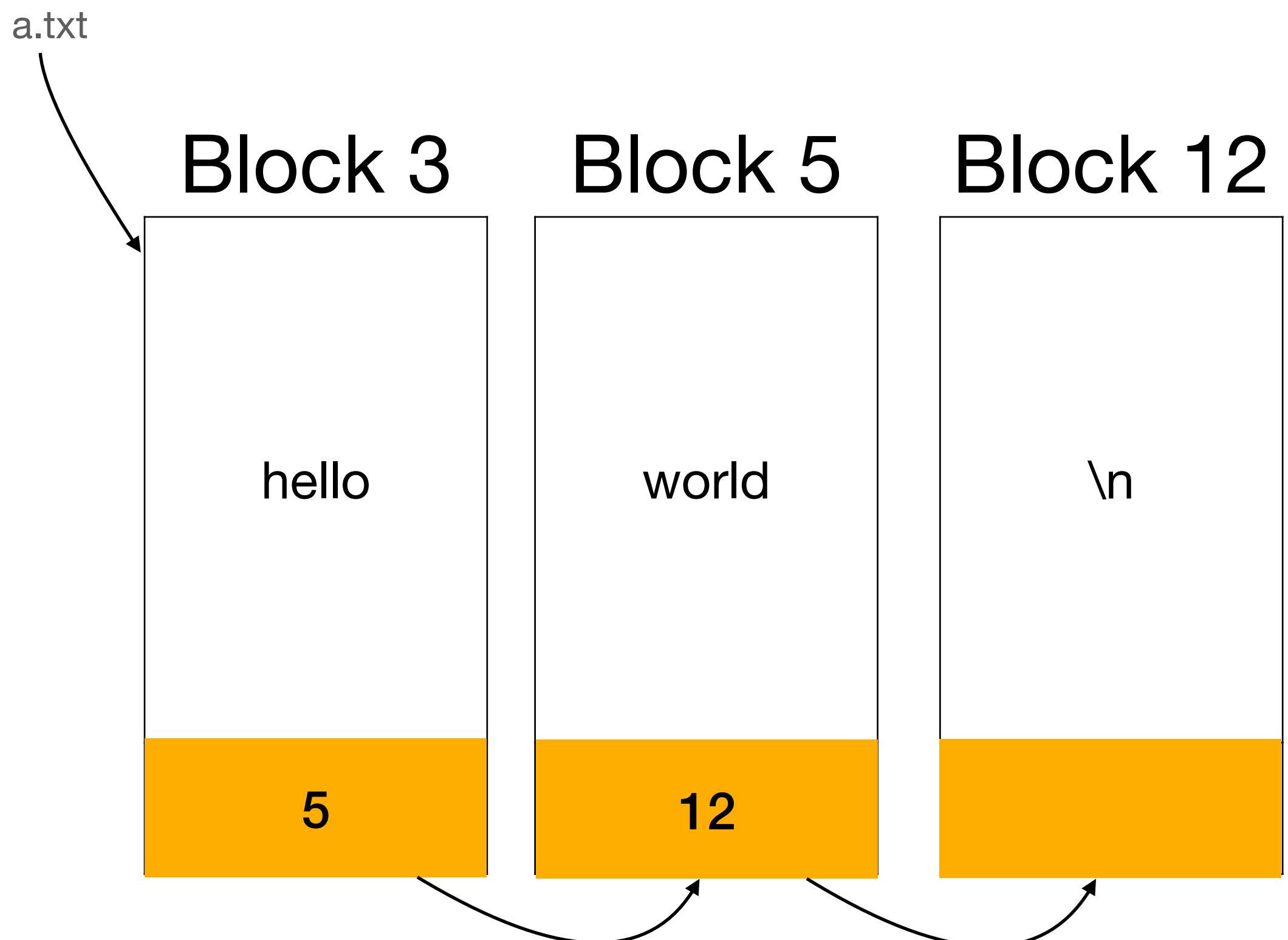
- ✓ Files can grow easily
- Seek / appends are terrible:
 - Need to read the whole file



How to store files?

Linked list of blocks

- ✓ Files can grow easily
 - Seek / appends are terrible:
 - Need to read the whole file
 - Sequential rws become random disk rws

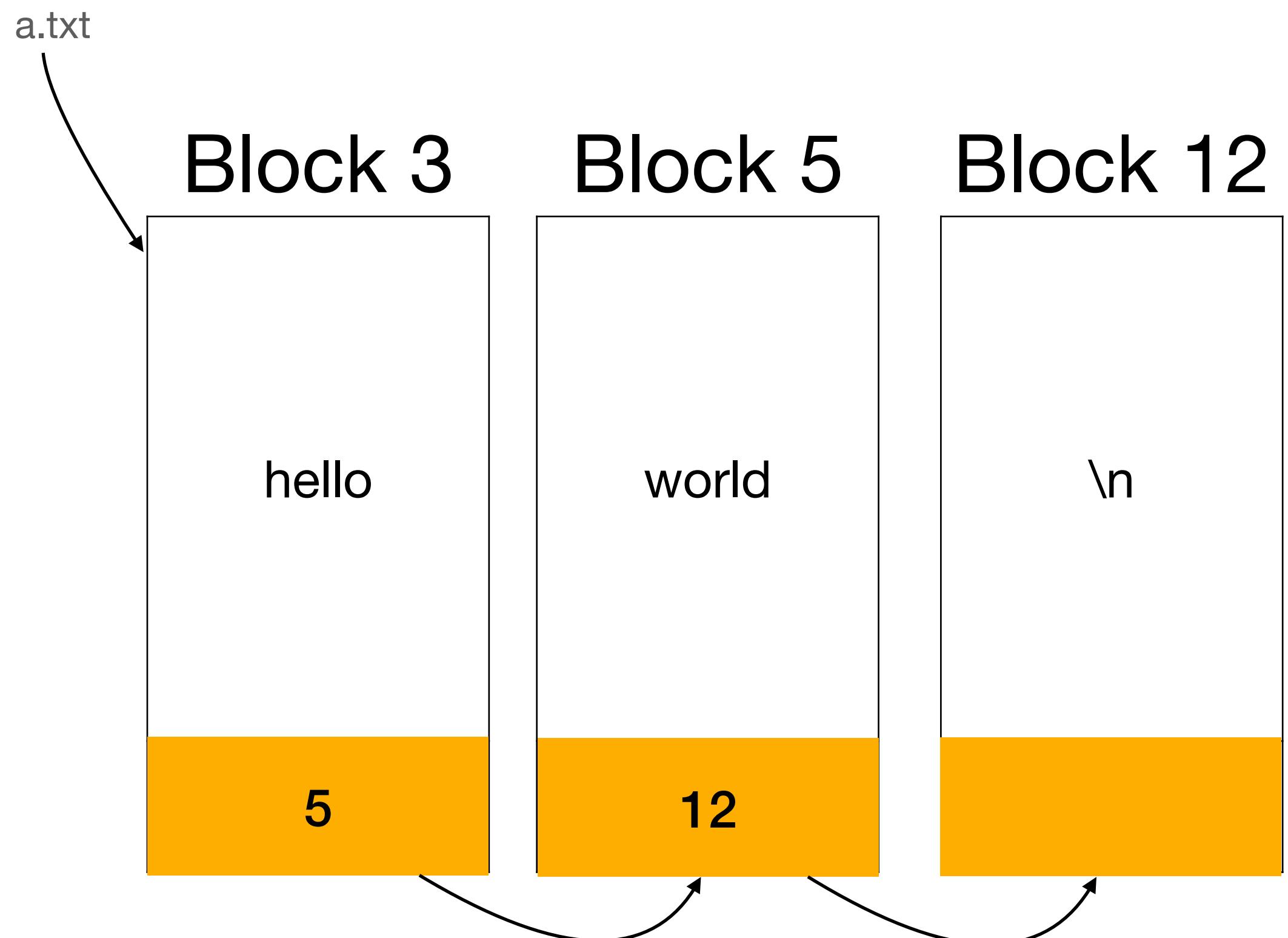


How to store files?

Linked list of blocks

✓ Files can grow easily

- Seek / appends are terrible:
 - Need to read the whole file
- Sequential rws become random disk rws
 - Cannot send >1 in-flight IO requests.
Lose disk scheduling potential.

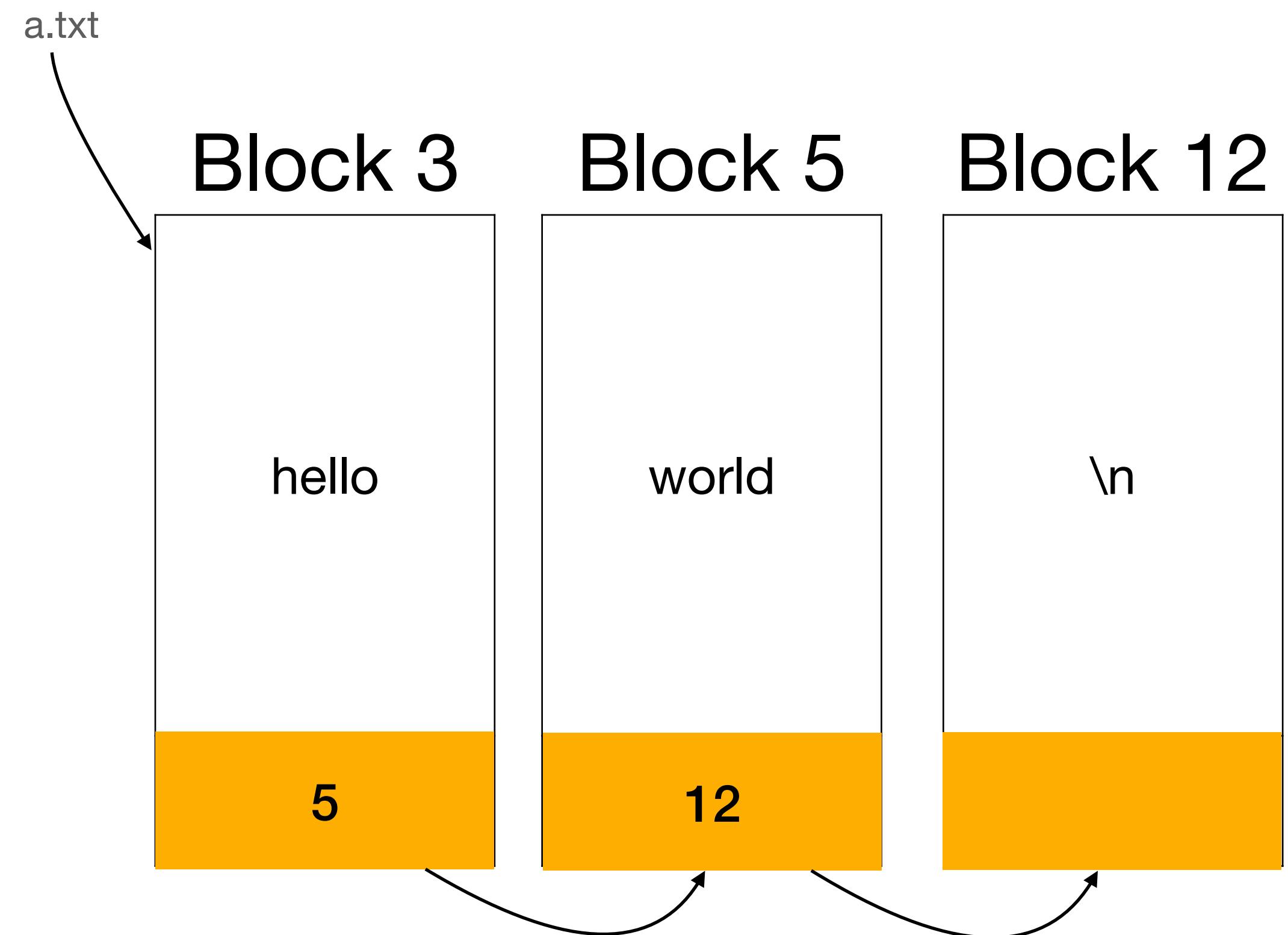


How to store files?

Linked list of blocks

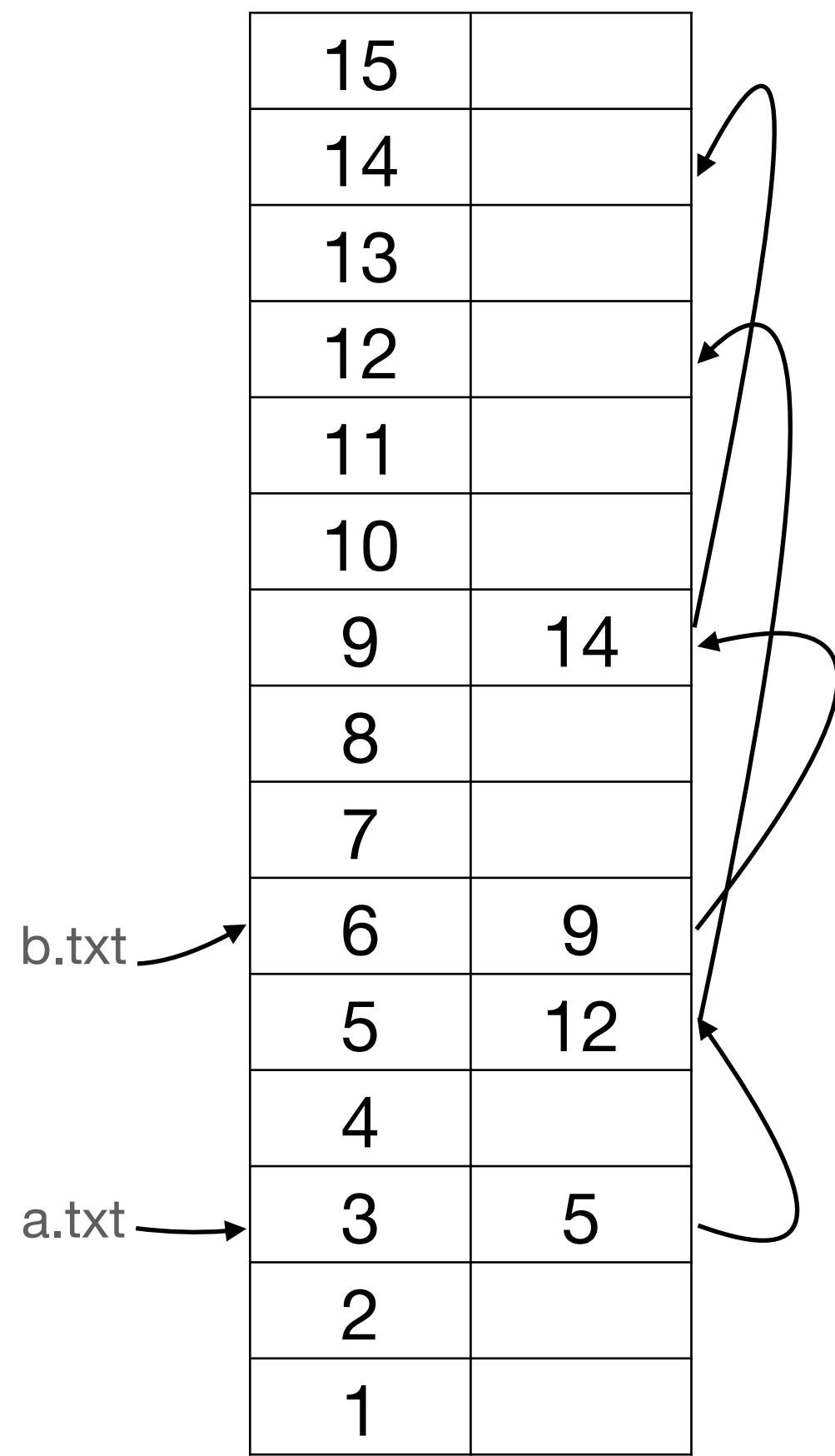
✓ Files can grow easily

- Seek / appends are terrible:
 - Need to read the whole file
- Sequential rws become random disk rws
 - Cannot send >1 in-flight IO requests.
Lose disk scheduling potential.
- If one block gets corrupted, parts of the file is lost



How to store files?

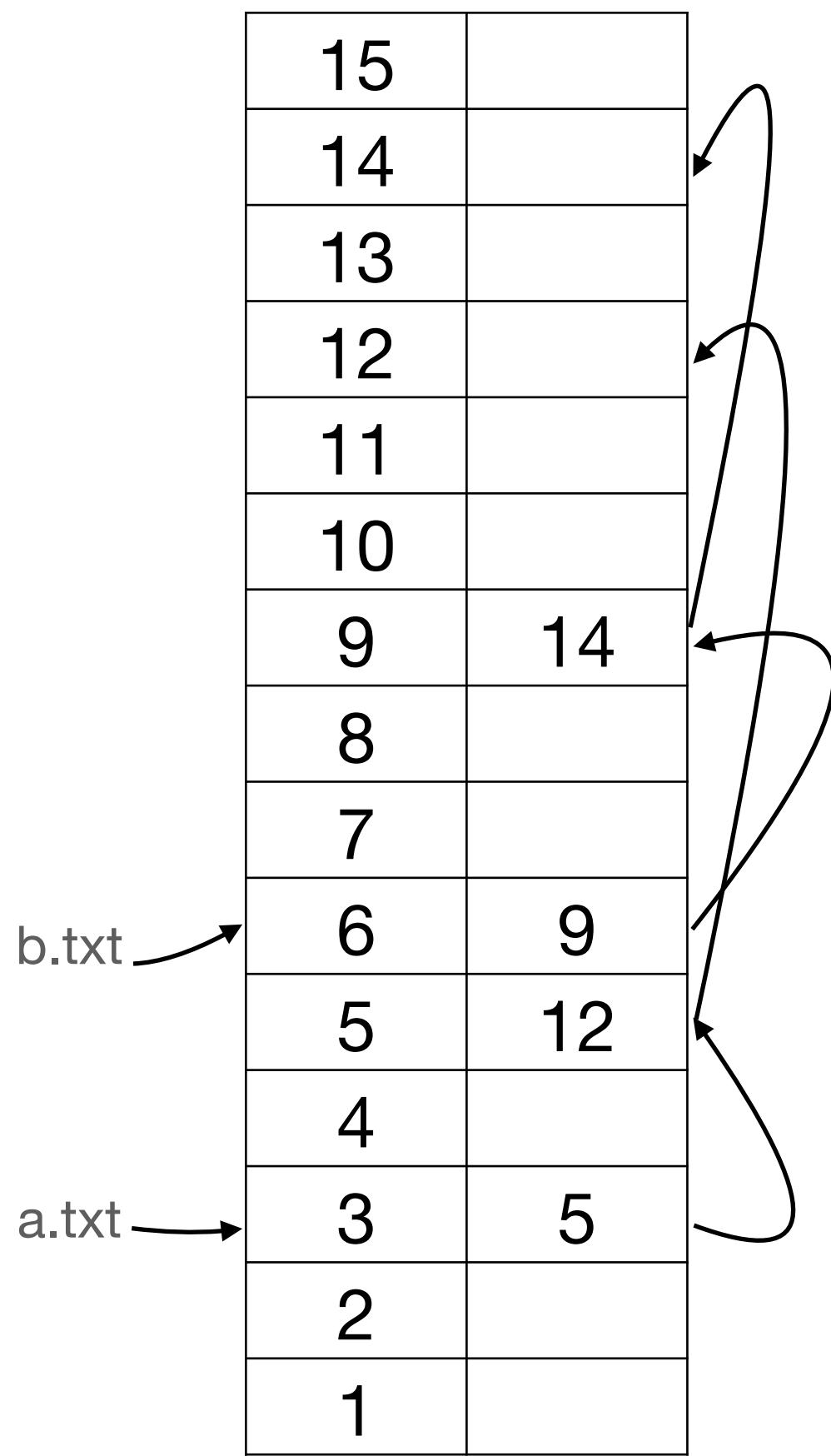
File Allocation Table (FAT filesystem)



How to store files?

File Allocation Table (FAT filesystem)

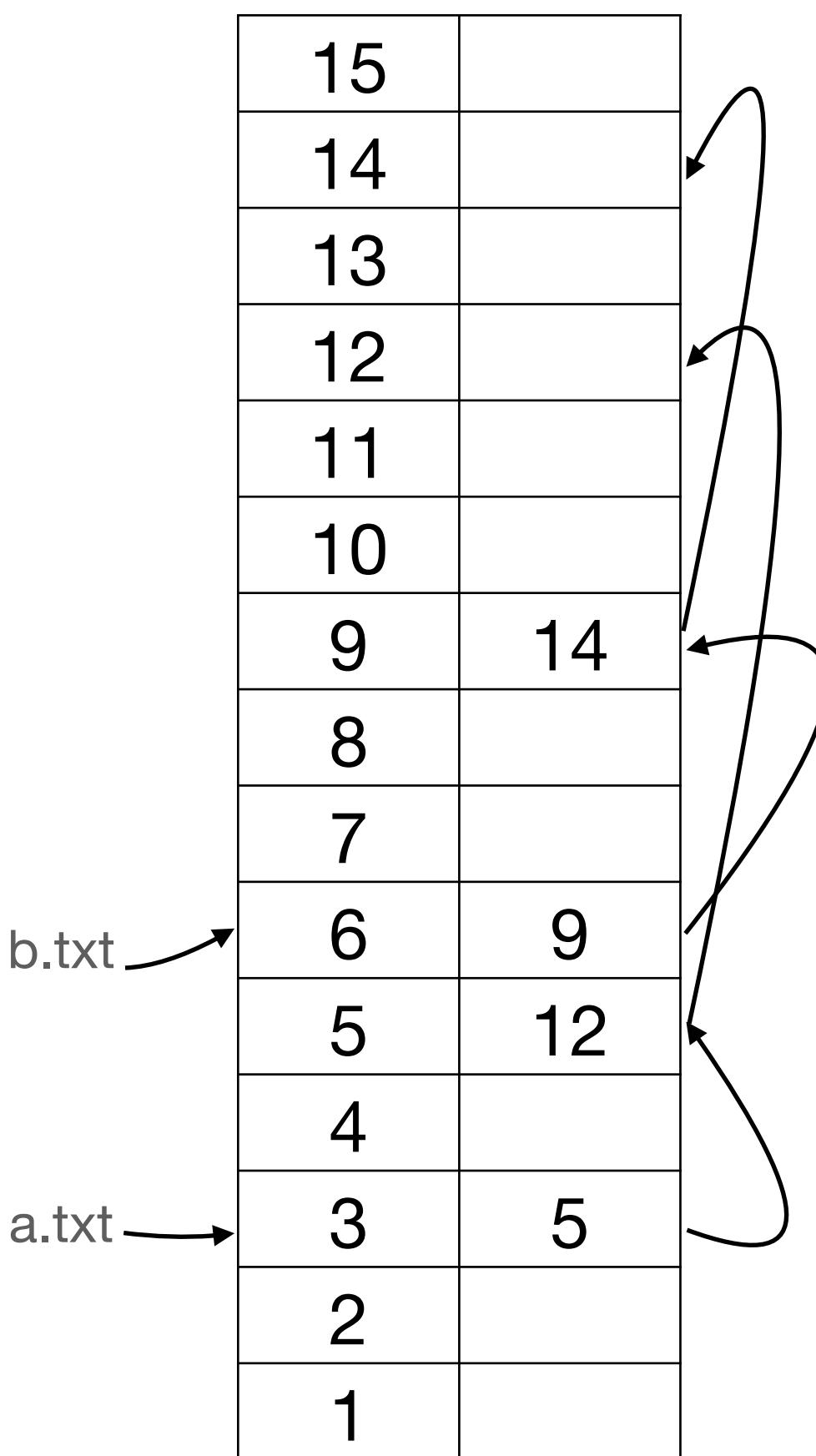
- Fast seeks/appends



How to store files?

File Allocation Table (FAT filesystem)

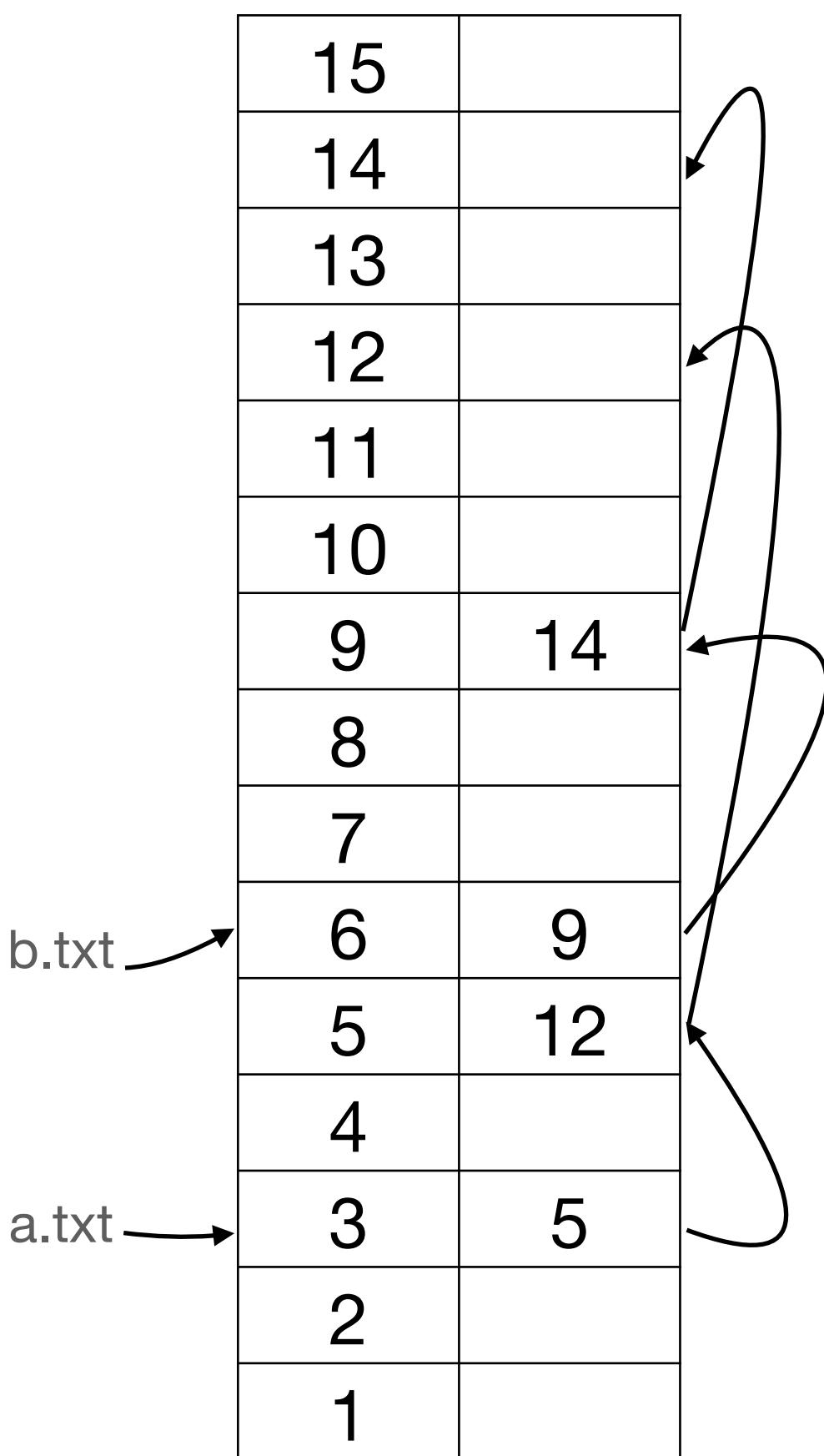
- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory



How to store files?

File Allocation Table (FAT filesystem)

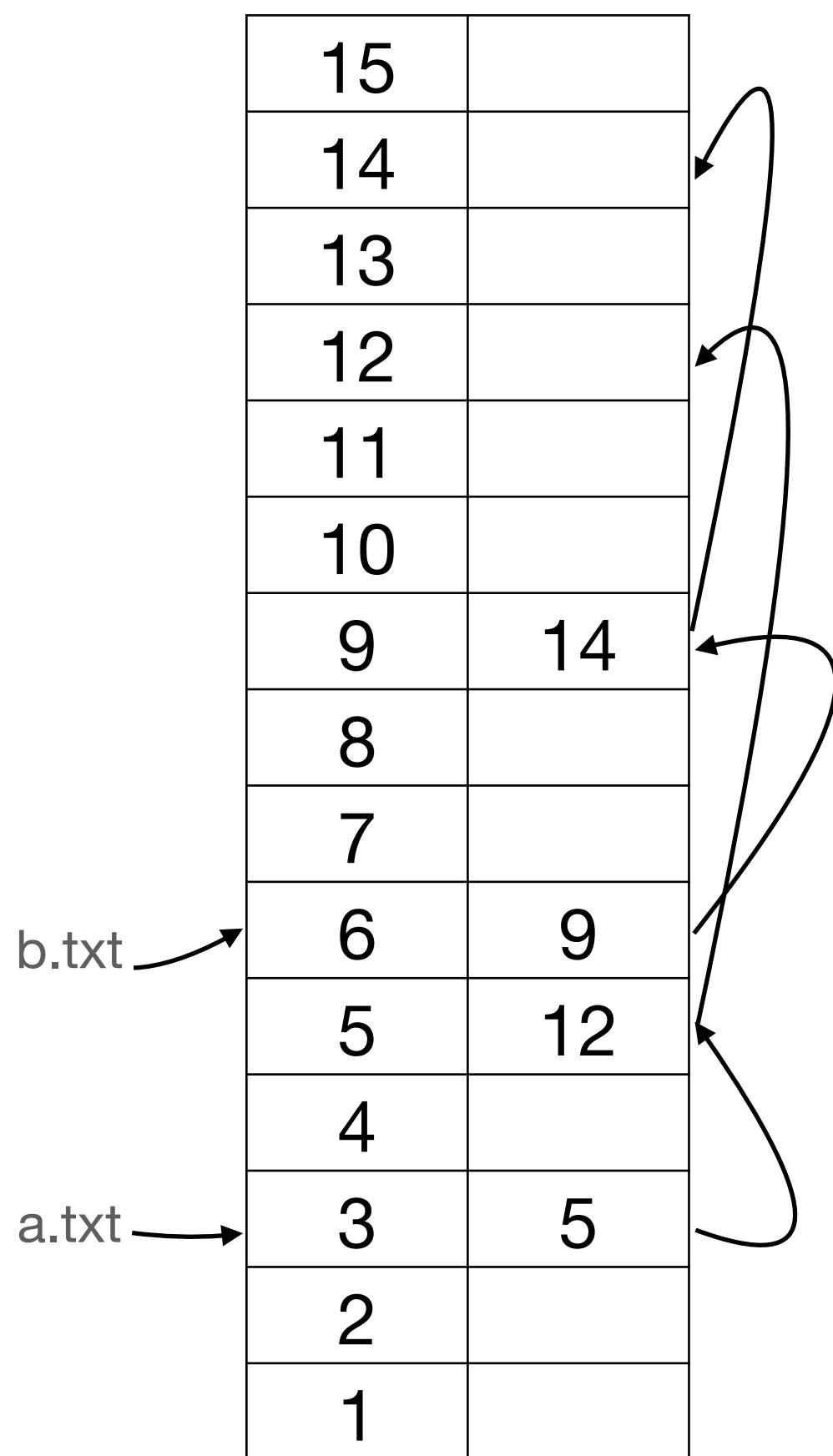
- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory
- Size of block: 2KB to 32KB.



How to store files?

File Allocation Table (FAT filesystem)

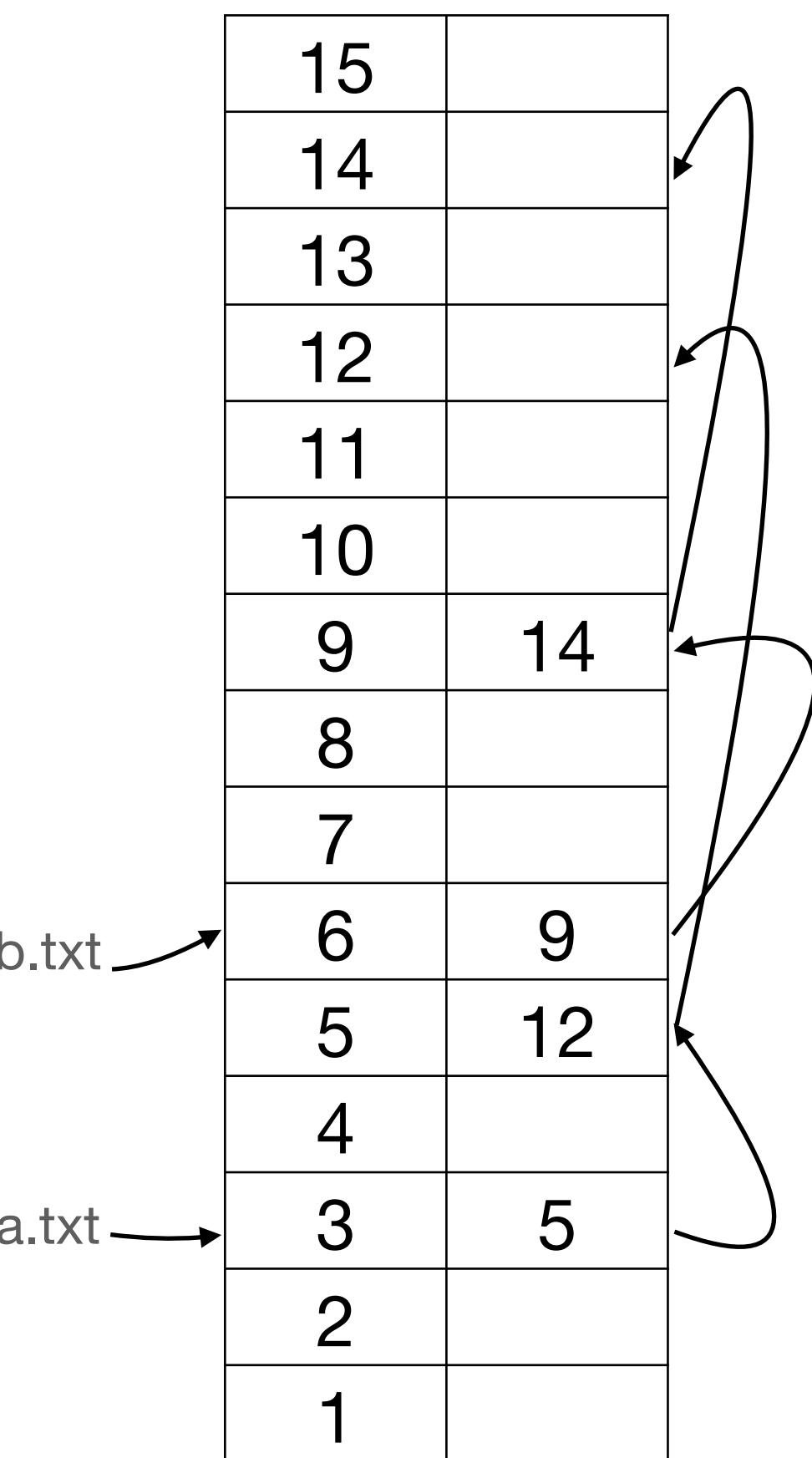
- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory
- Size of block: 2KB to 32KB.
- FAT16



How to store files?

File Allocation Table (FAT filesystem)

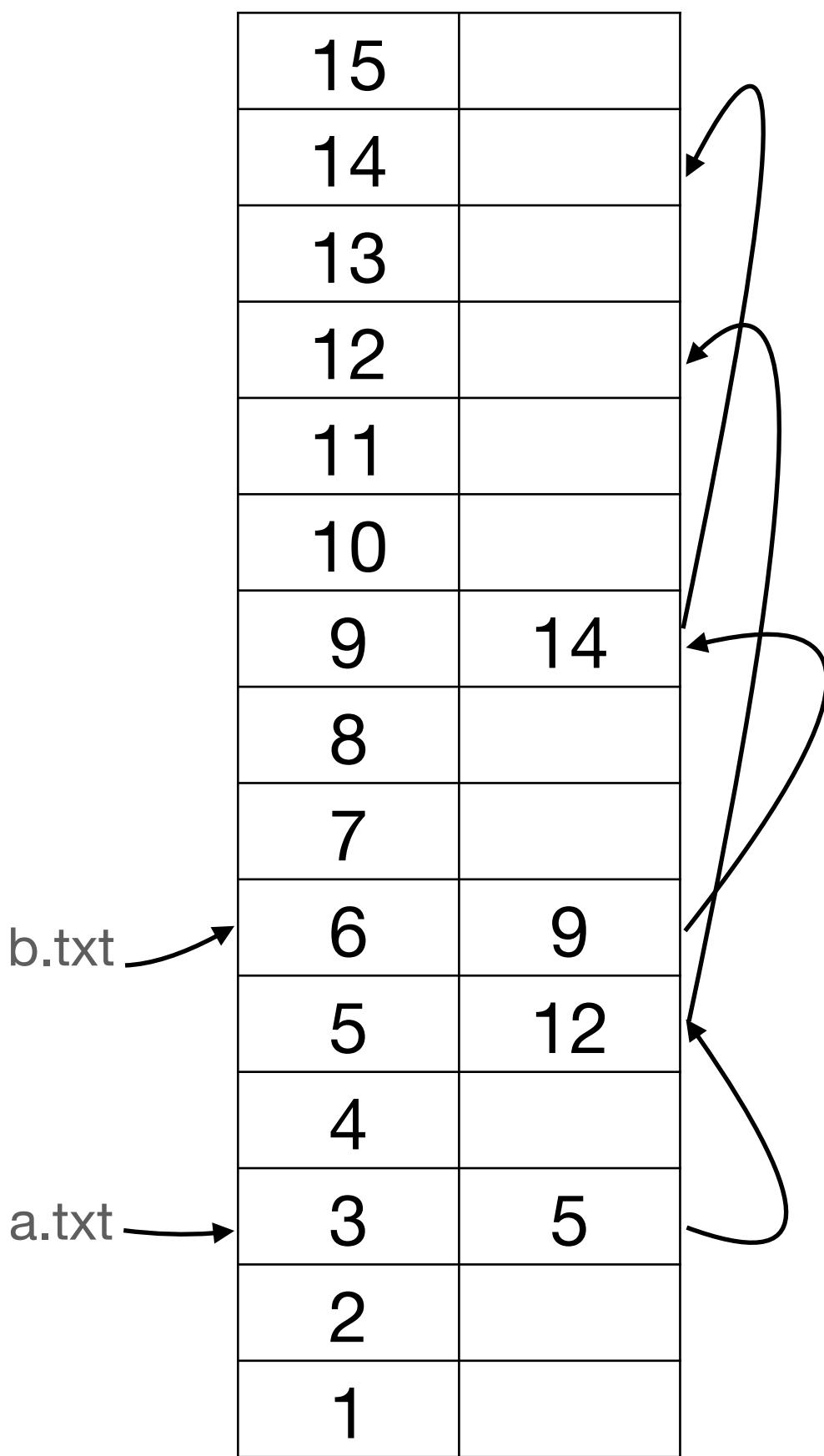
- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory
- Size of block: 2KB to 32KB.
 - FAT16
 - 2^{16} entries. Maximum disk size: $2^{16} * 2\text{KB} = 128 \text{ MB}$



How to store files?

File Allocation Table (FAT filesystem)

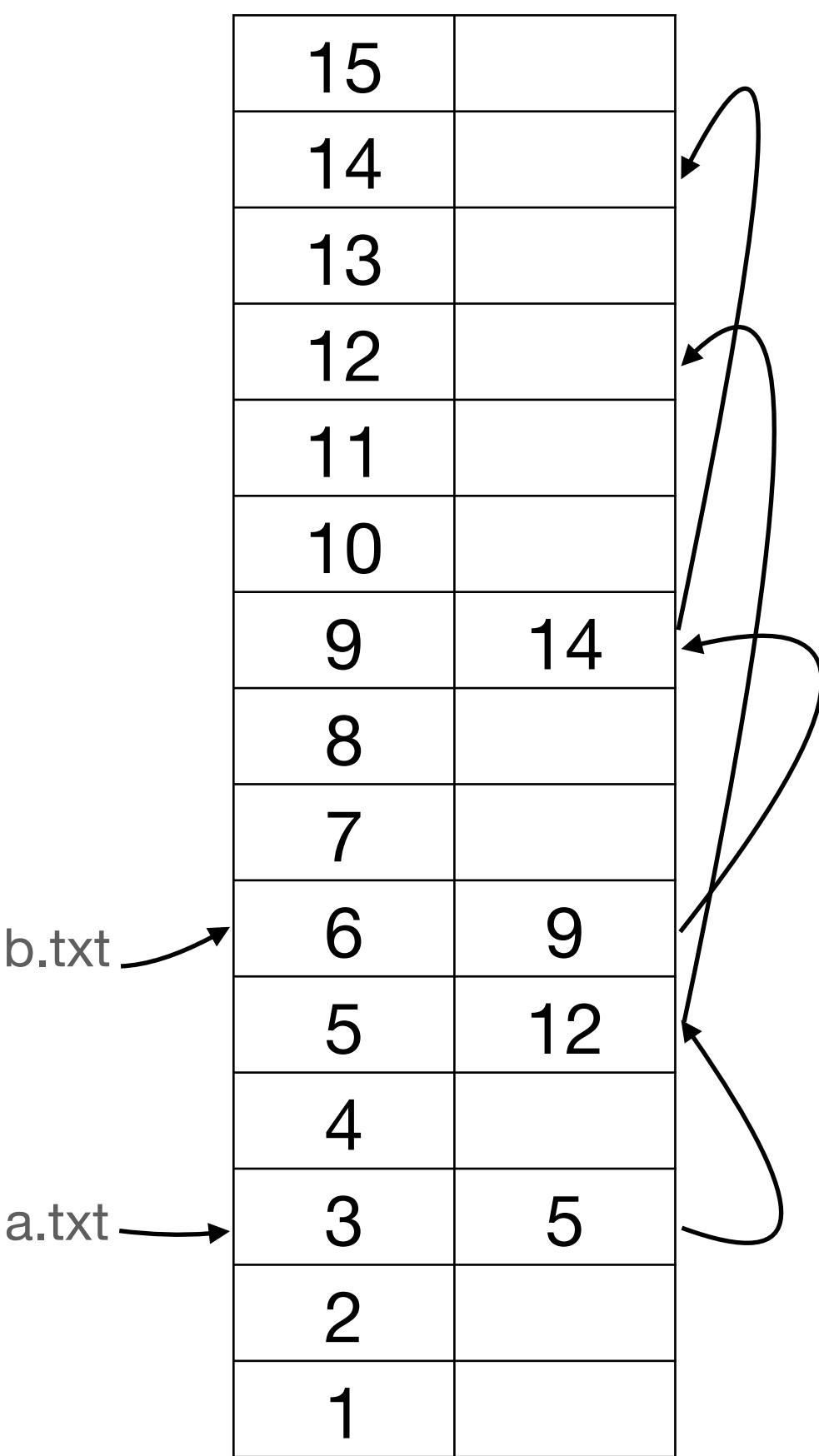
- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory
- Size of block: 2KB to 32KB.
 - FAT16
 - 2^{16} entries. Maximum disk size: $2^{16} * 2\text{KB} = 128 \text{ MB}$
 - Size of table = $2^{16} * (2 \text{ bytes}) = 128\text{KB}$



How to store files?

File Allocation Table (FAT filesystem)

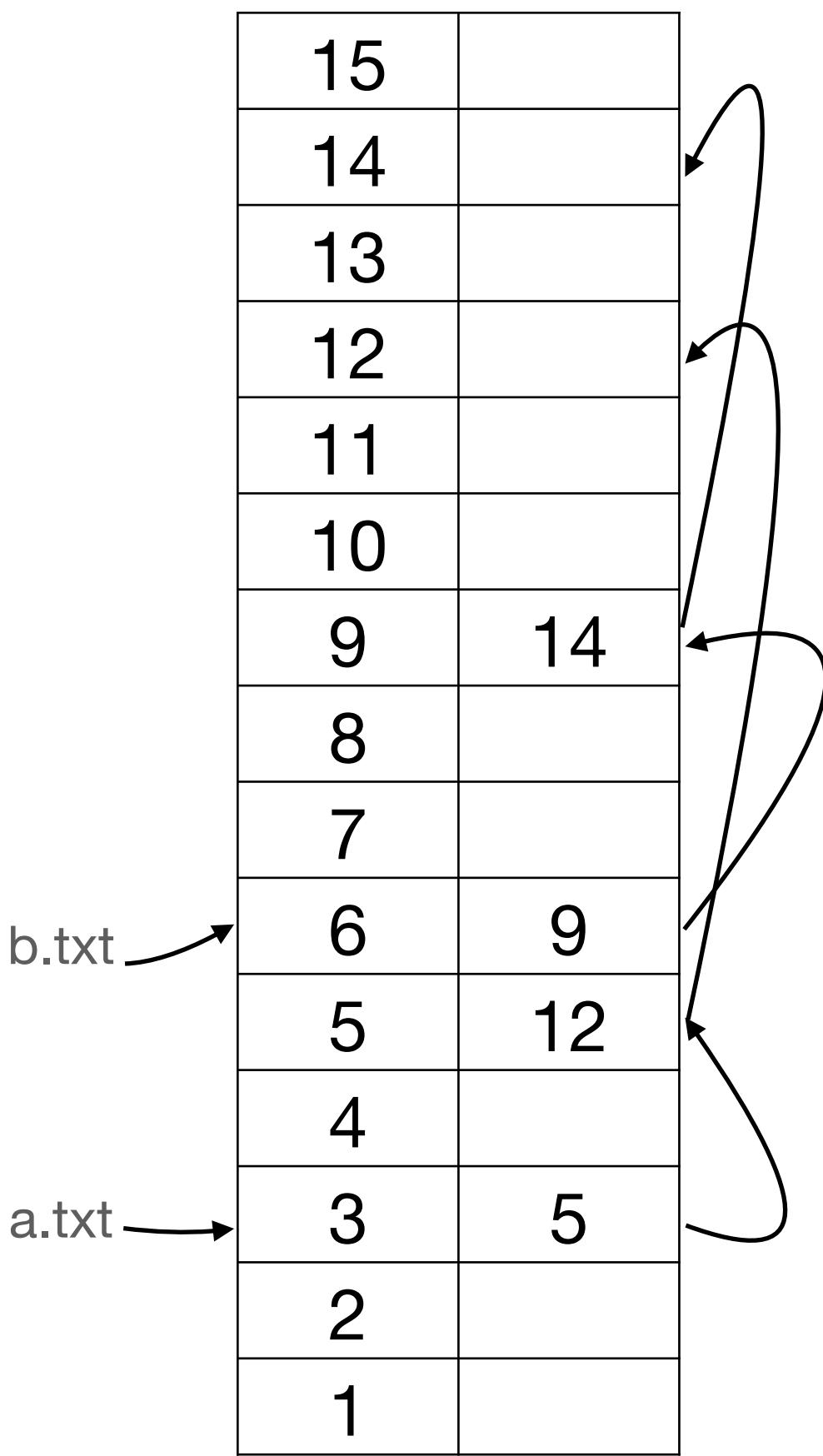
- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory
- Size of block: 2KB to 32KB.
 - FAT16
 - 2^{16} entries. Maximum disk size: $2^{16} * 2\text{KB} = 128 \text{ MB}$
 - Size of table = $2^{16} * (2 \text{ bytes}) = 128\text{KB}$
 - FAT32



How to store files?

File Allocation Table (FAT filesystem)

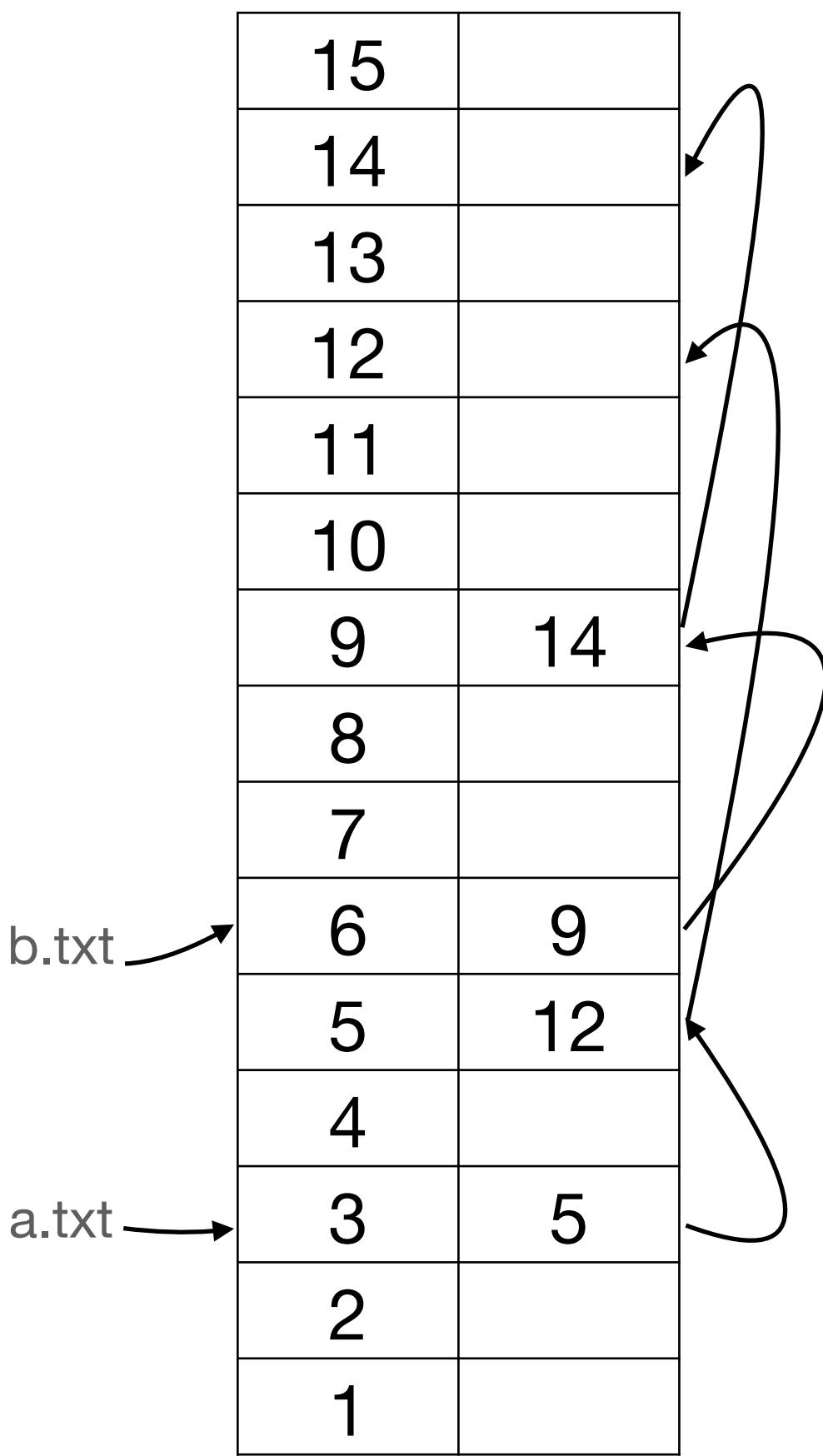
- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory
- Size of block: 2KB to 32KB.
 - FAT16
 - 2^{16} entries. Maximum disk size: $2^{16} * 2\text{KB} = 128 \text{ MB}$
 - Size of table = $2^{16} * (2 \text{ bytes}) = 128\text{KB}$
 - FAT32
 - 2^{28} entries. Maximum disk size: $2^{28} * 2\text{KB} = 512 \text{ GB}$



How to store files?

File Allocation Table (FAT filesystem)

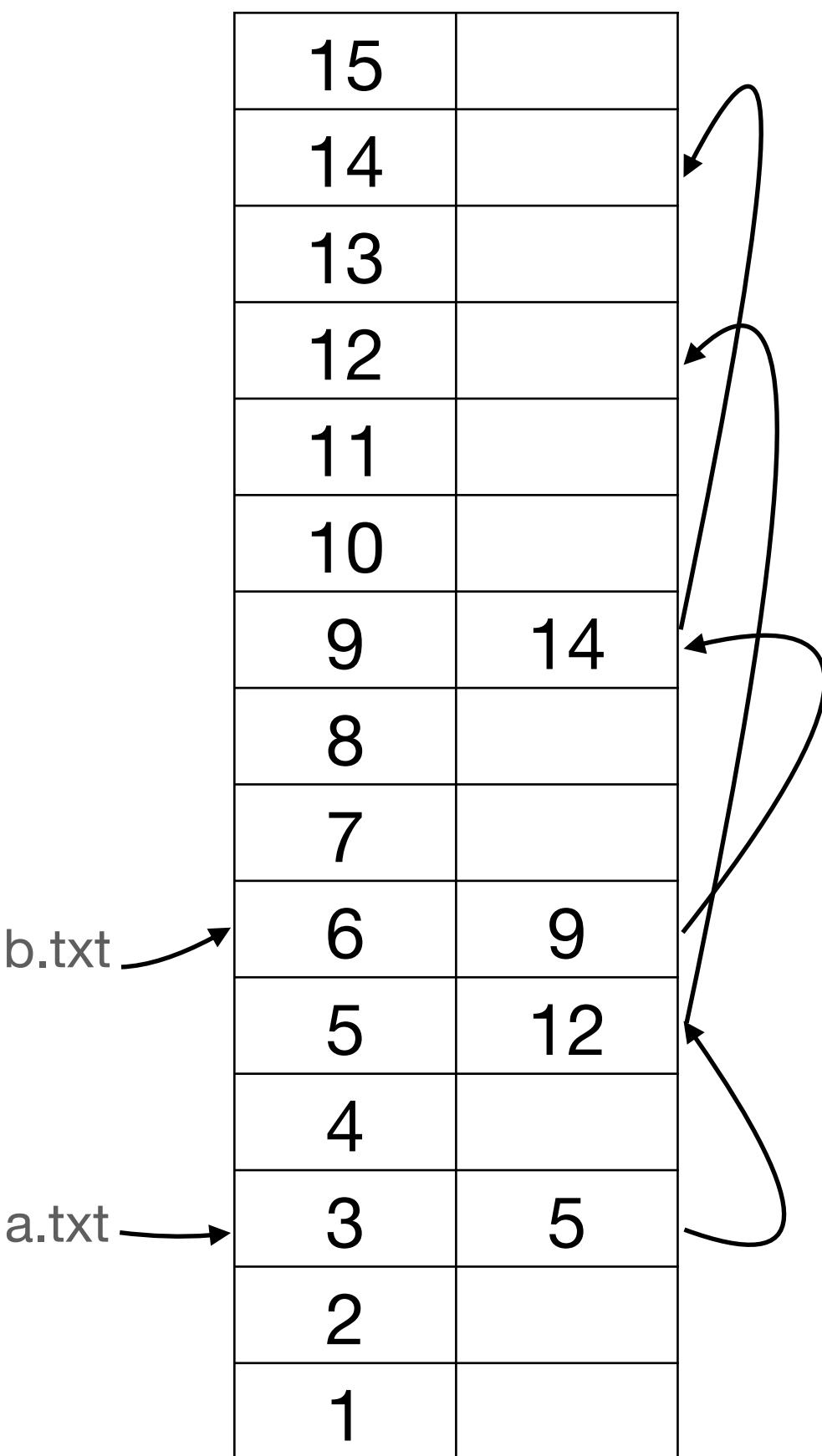
- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory
- Size of block: 2KB to 32KB.
 - FAT16
 - 2^{16} entries. Maximum disk size: $2^{16} * 2\text{KB} = 128 \text{ MB}$
 - Size of table = $2^{16} * (2 \text{ bytes}) = 128\text{KB}$
 - FAT32
 - 2^{28} entries. Maximum disk size: $2^{28} * 2\text{KB} = 512 \text{ GB}$
 - Size of table = $2^{28} * (4 \text{ bytes}) = 1\text{GB}$



How to store files?

File Allocation Table (FAT filesystem)

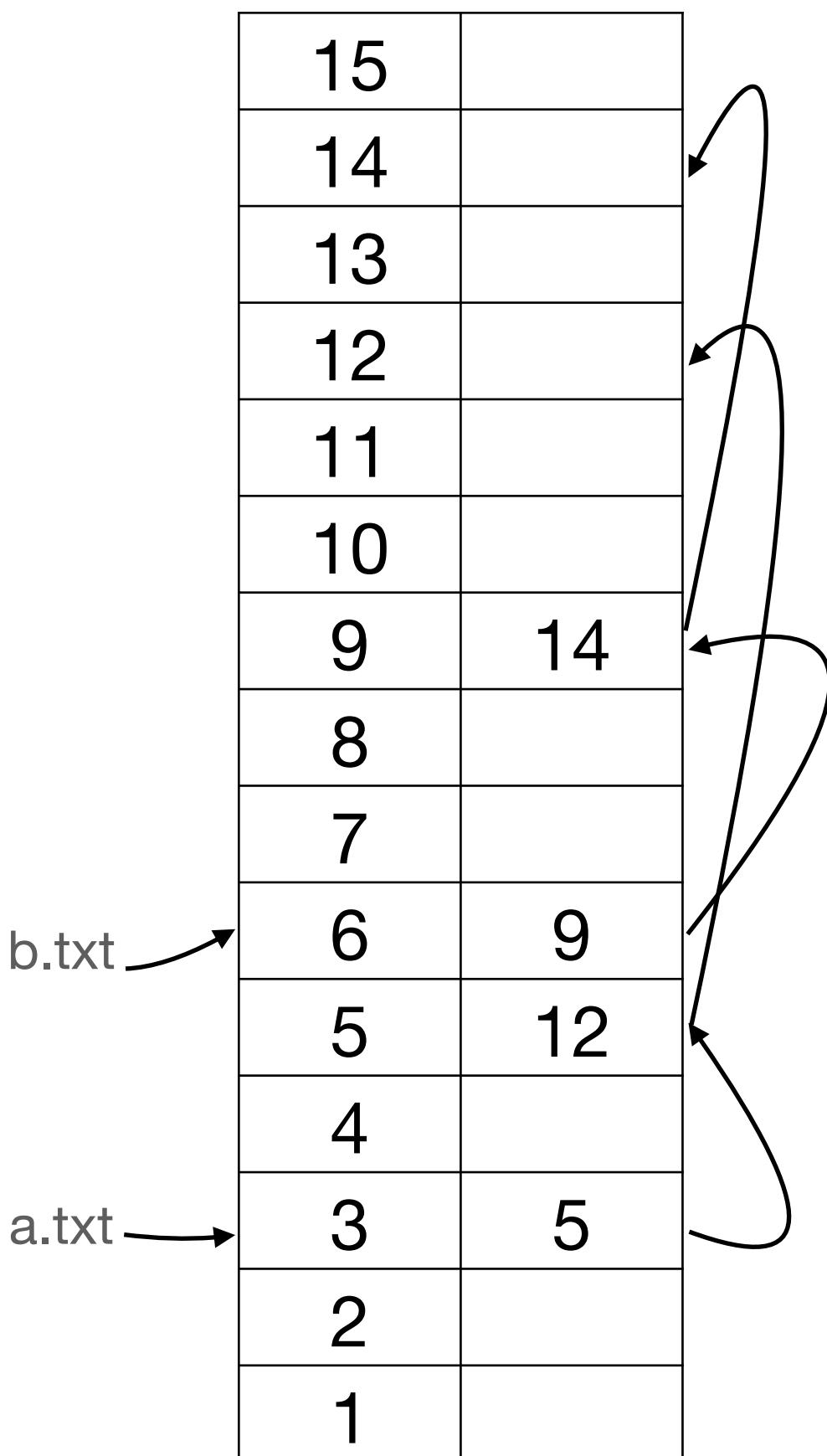
- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory
- Size of block: 2KB to 32KB.
 - FAT16
 - 2^{16} entries. Maximum disk size: $2^{16} * 2\text{KB} = 128 \text{ MB}$
 - Size of table = $2^{16} * (2 \text{ bytes}) = 128\text{KB}$
 - FAT32
 - 2^{28} entries. Maximum disk size: $2^{28} * 2\text{KB} = 512 \text{ GB}$
 - Size of table = $2^{28} * (4 \text{ bytes}) = 1\text{GB}$
- Reliability:



How to store files?

File Allocation Table (FAT filesystem)

- Fast seeks/appends
 - Bring table into memory, do pointer chasing in memory
- Size of block: 2KB to 32KB.
 - FAT16
 - 2^{16} entries. Maximum disk size: $2^{16} * 2\text{KB} = 128 \text{ MB}$
 - Size of table = $2^{16} * (2 \text{ bytes}) = 128\text{KB}$
 - FAT32
 - 2^{28} entries. Maximum disk size: $2^{28} * 2\text{KB} = 512 \text{ GB}$
 - Size of table = $2^{28} * (4 \text{ bytes}) = 1\text{GB}$
- Reliability:
 - Lose file system if we lose FAT table. Keep two copies.



Block size in FAT

Block size in FAT

- Large block size:

Block size in FAT

- Large block size:
 - ✓ Support larger disks

Block size in FAT

- Large block size:
 - ✓ Support larger disks
 - ✓ Reduced random IO

Block size in FAT

- Large block size:
 - ✓ Support larger disks
 - ✓ Reduced random IO
 - ✓ Reduced Metadata overhead. FAT32 overhead: 4 bytes / 2KB ~ 0.2%;
4 bytes / 32KB ~ 0.01%

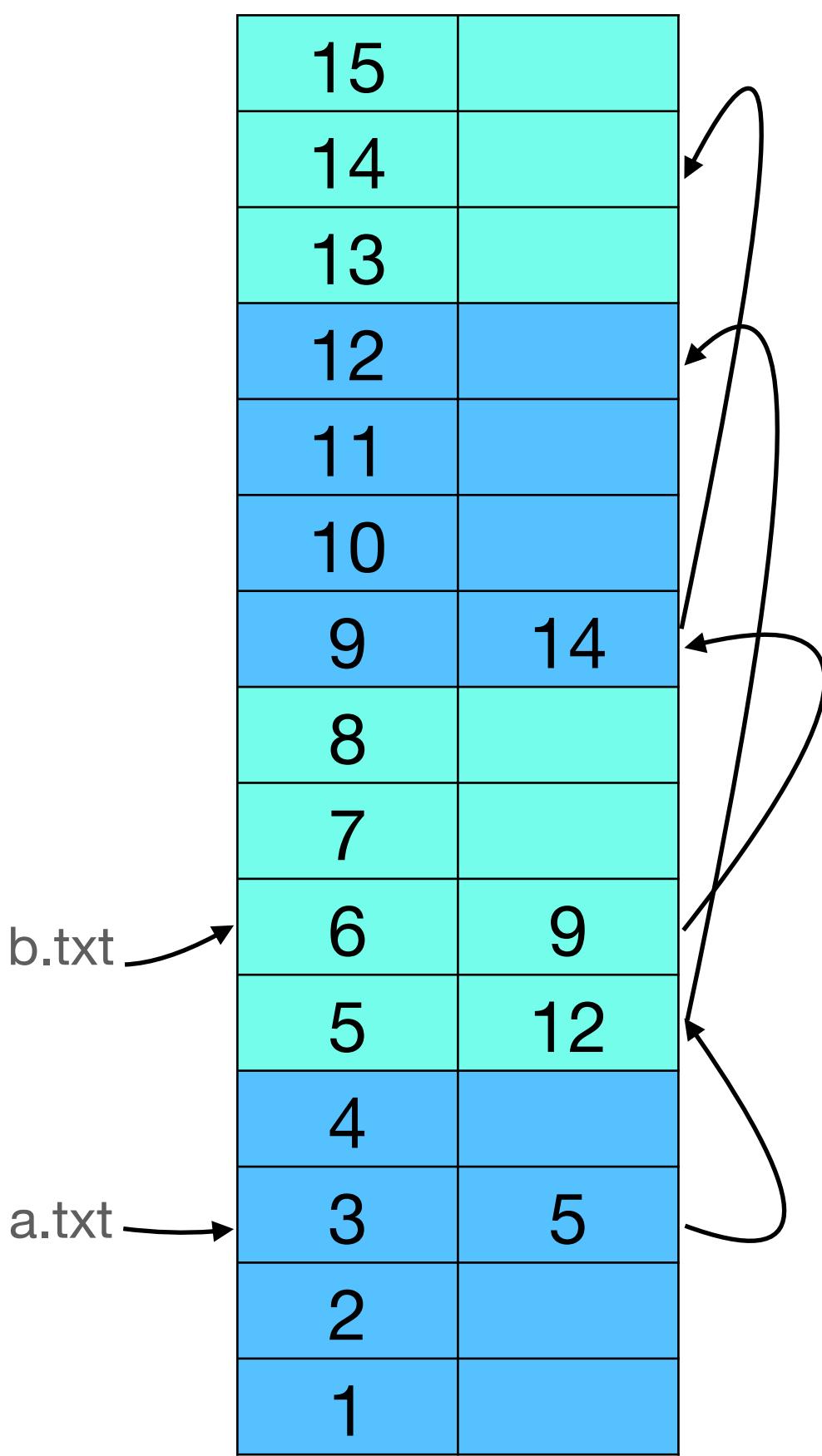
Block size in FAT

- Large block size:
 - ✓ Support larger disks
 - ✓ Reduced random IO
 - ✓ Reduced Metadata overhead. FAT32 overhead: 4 bytes / 2KB ~ 0.2%;
4 bytes / 32KB ~ 0.01%
- Increased internal fragmentation: minimum file size is block size

Block size in FAT

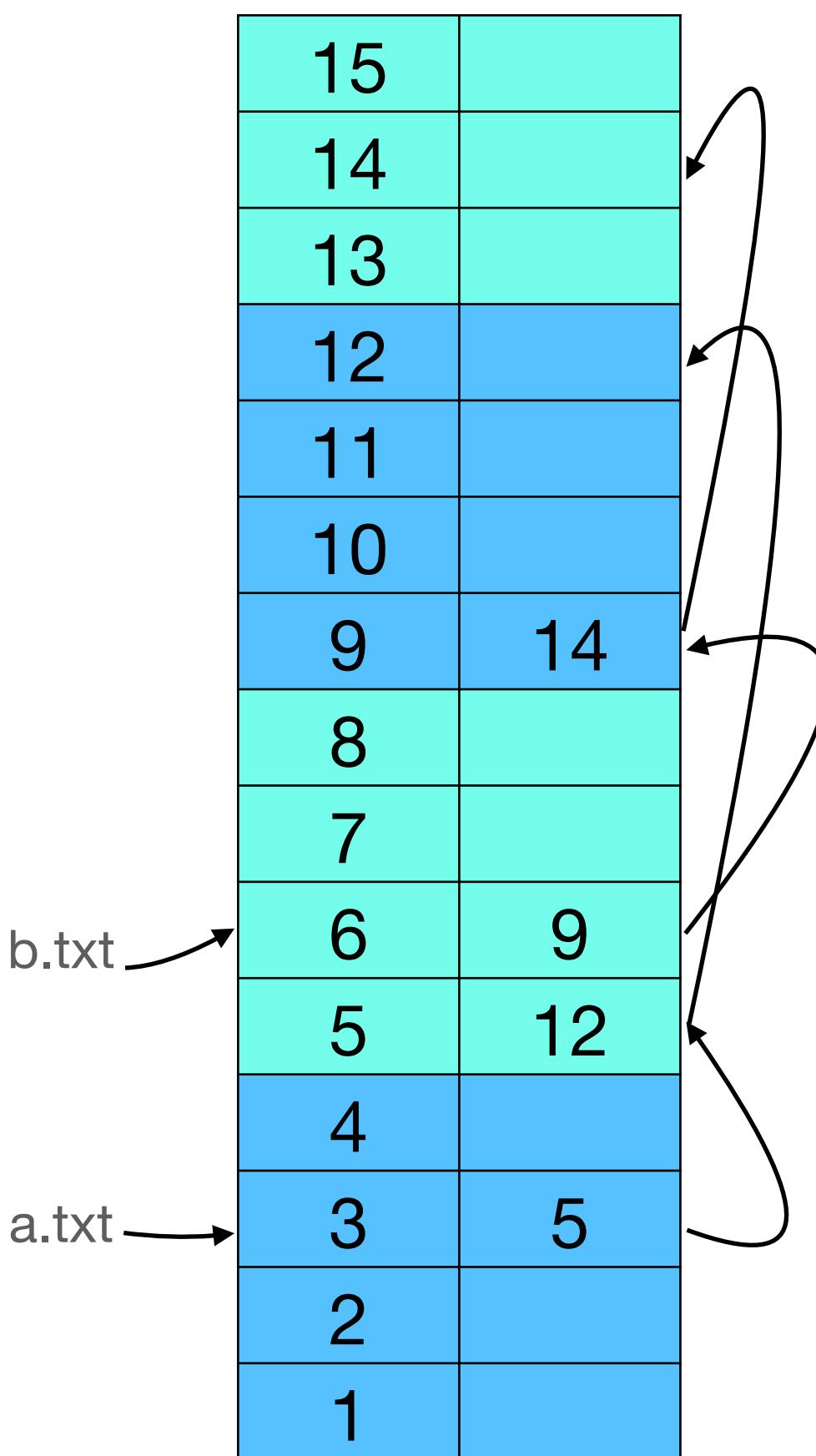
- Large block size:
 - ✓ Support larger disks
 - ✓ Reduced random IO
 - ✓ Reduced Metadata overhead. FAT32 overhead: 4 bytes / 2KB ~ 0.2%;
4 bytes / 32KB ~ 0.01%
- Increased internal fragmentation: minimum file size is block size
- Increase buffer cache pressure: lesser number of blocks can be cached

Performance



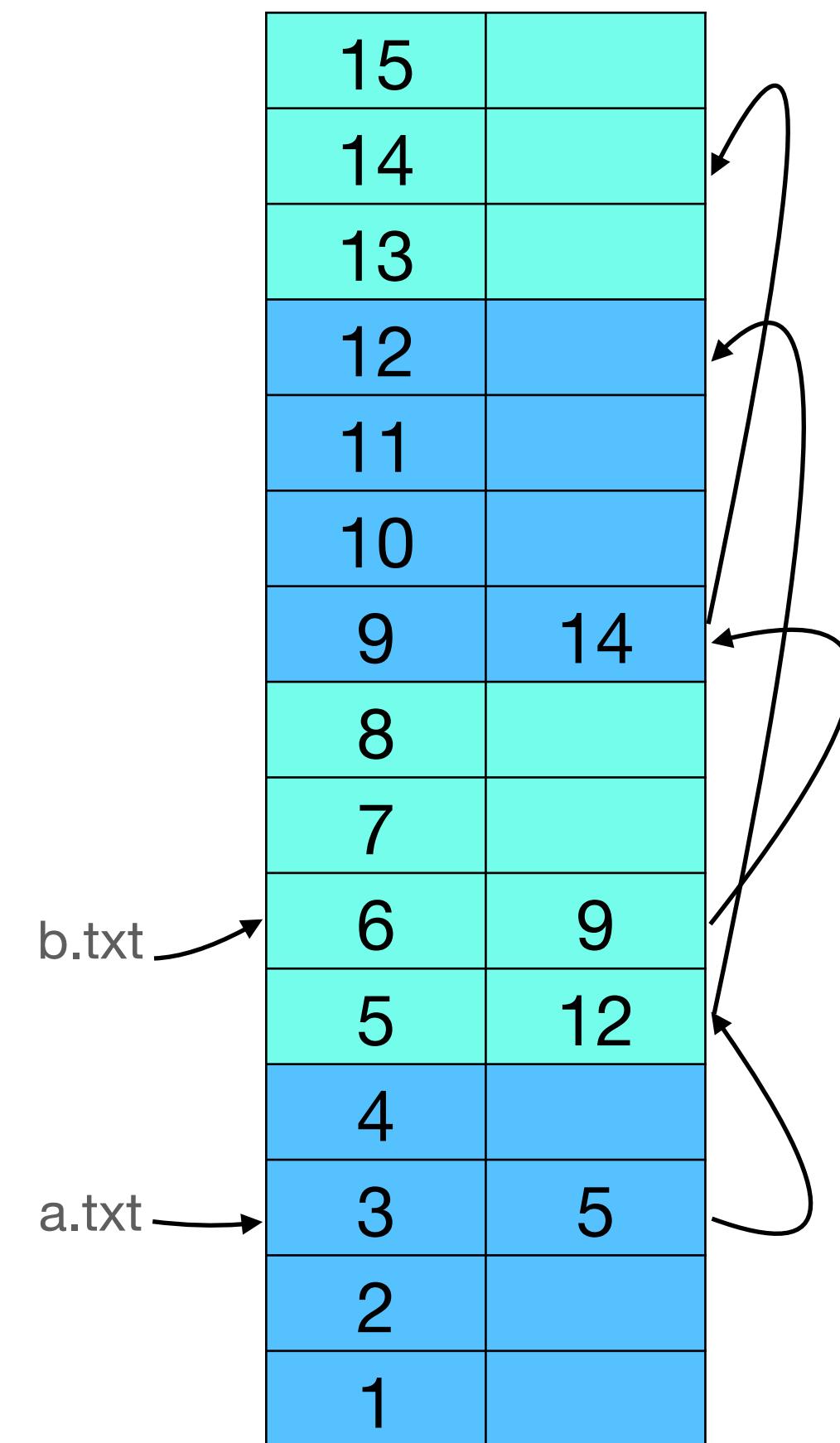
Performance

- Sequential IO



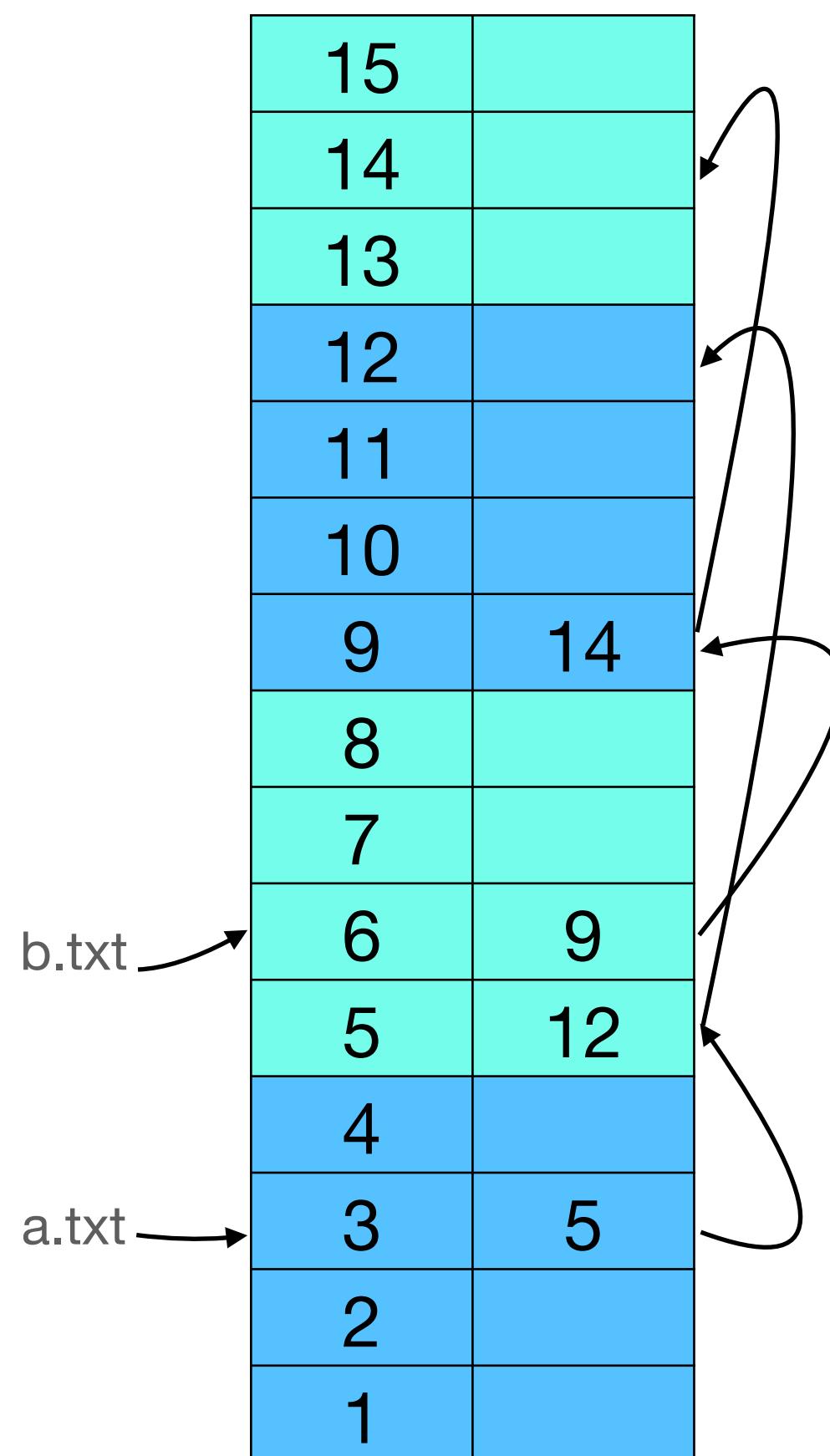
Performance

- Sequential IO
 - Better than linked list. Can find the list of blocks apriori and send requests. Disk controller can schedule them.



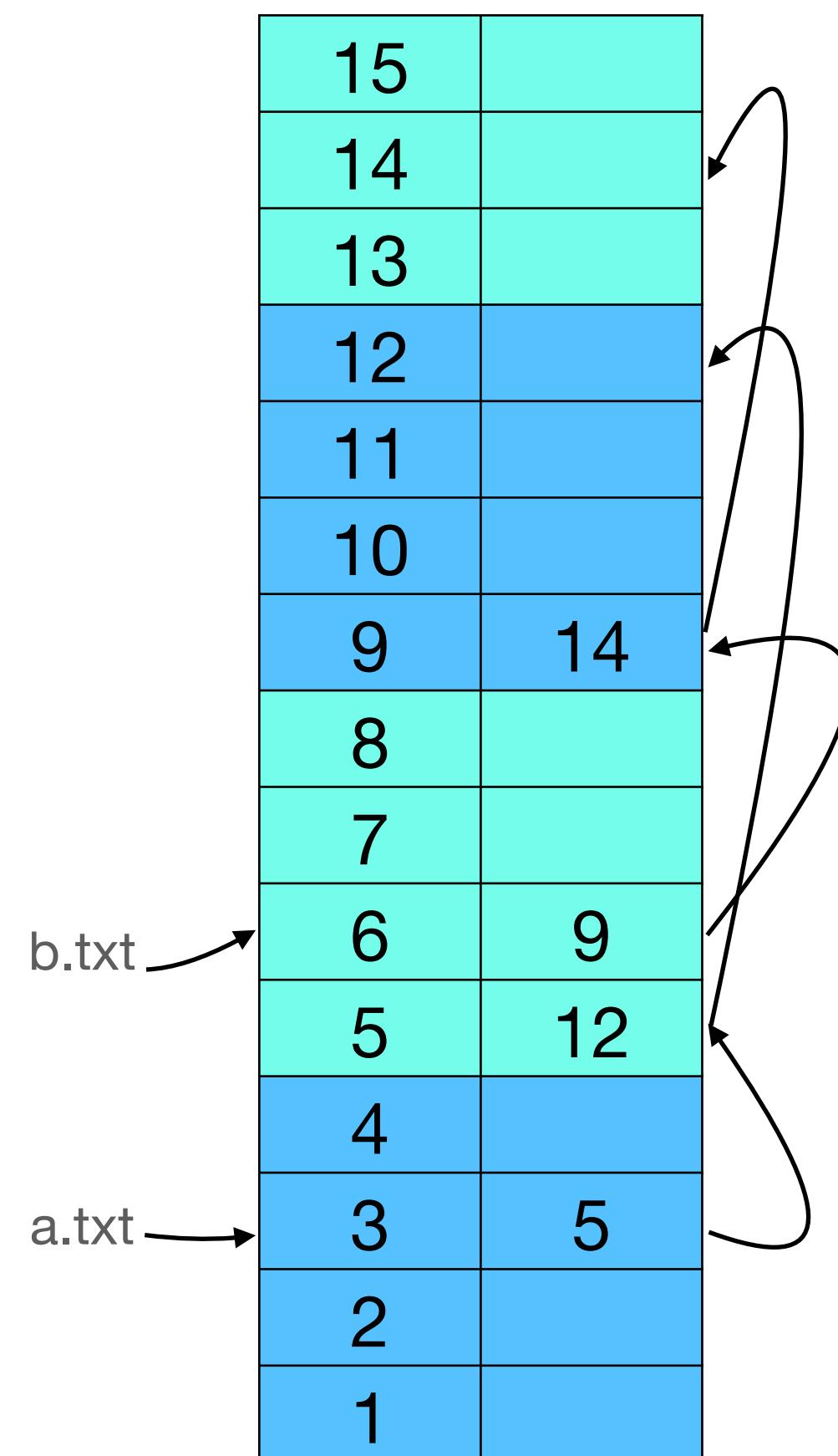
Performance

- Sequential IO
 - Better than linked list. Can find the list of blocks apriori and send requests. Disk controller can schedule them.
 - Worse than contiguous allocation since it did only 1 seek.



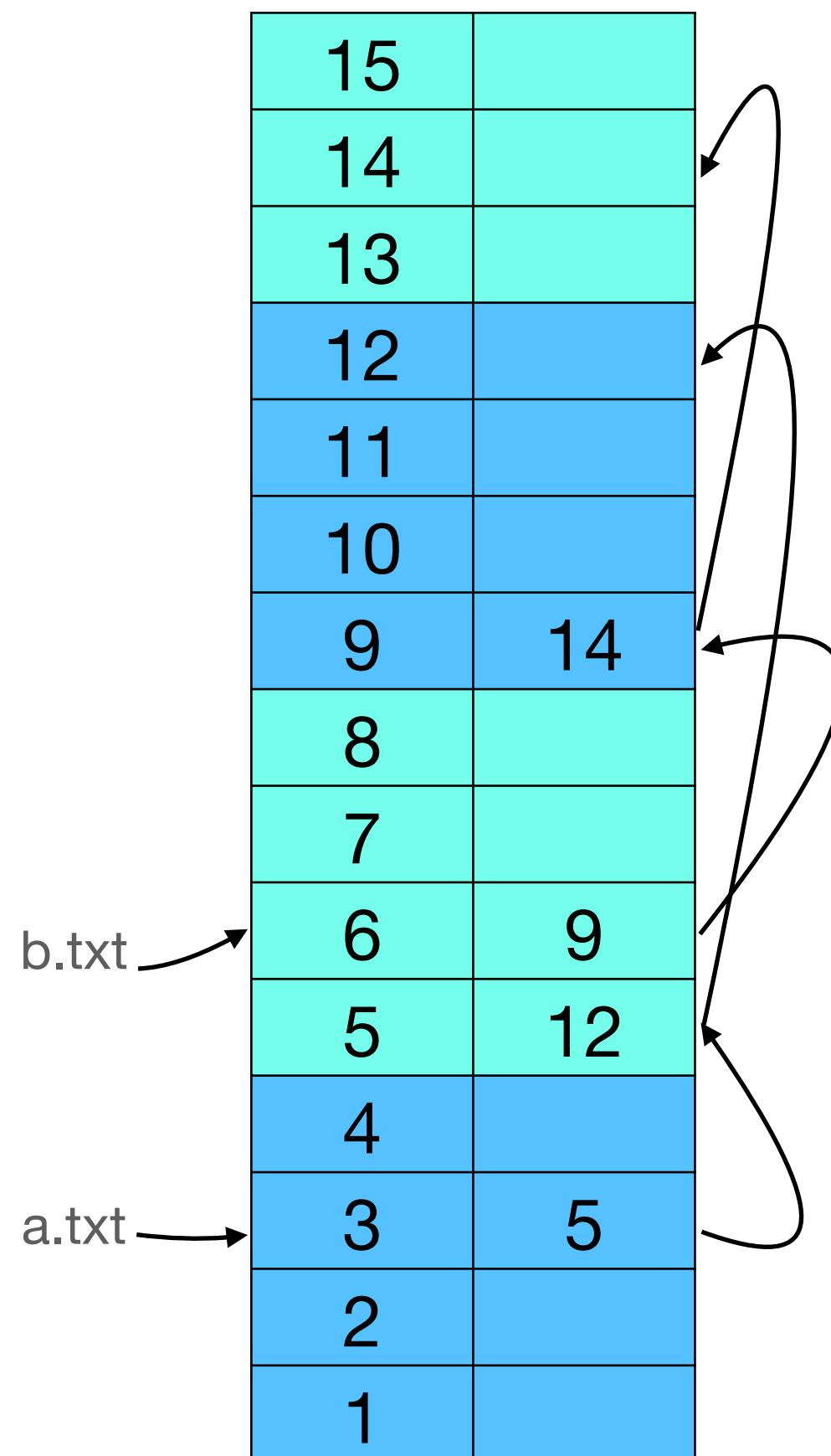
Performance

- Sequential IO
 - Better than linked list. Can find the list of blocks apriori and send requests. Disk controller can schedule them.
 - Worse than contiguous allocation since it did only 1 seek.
- Random IO



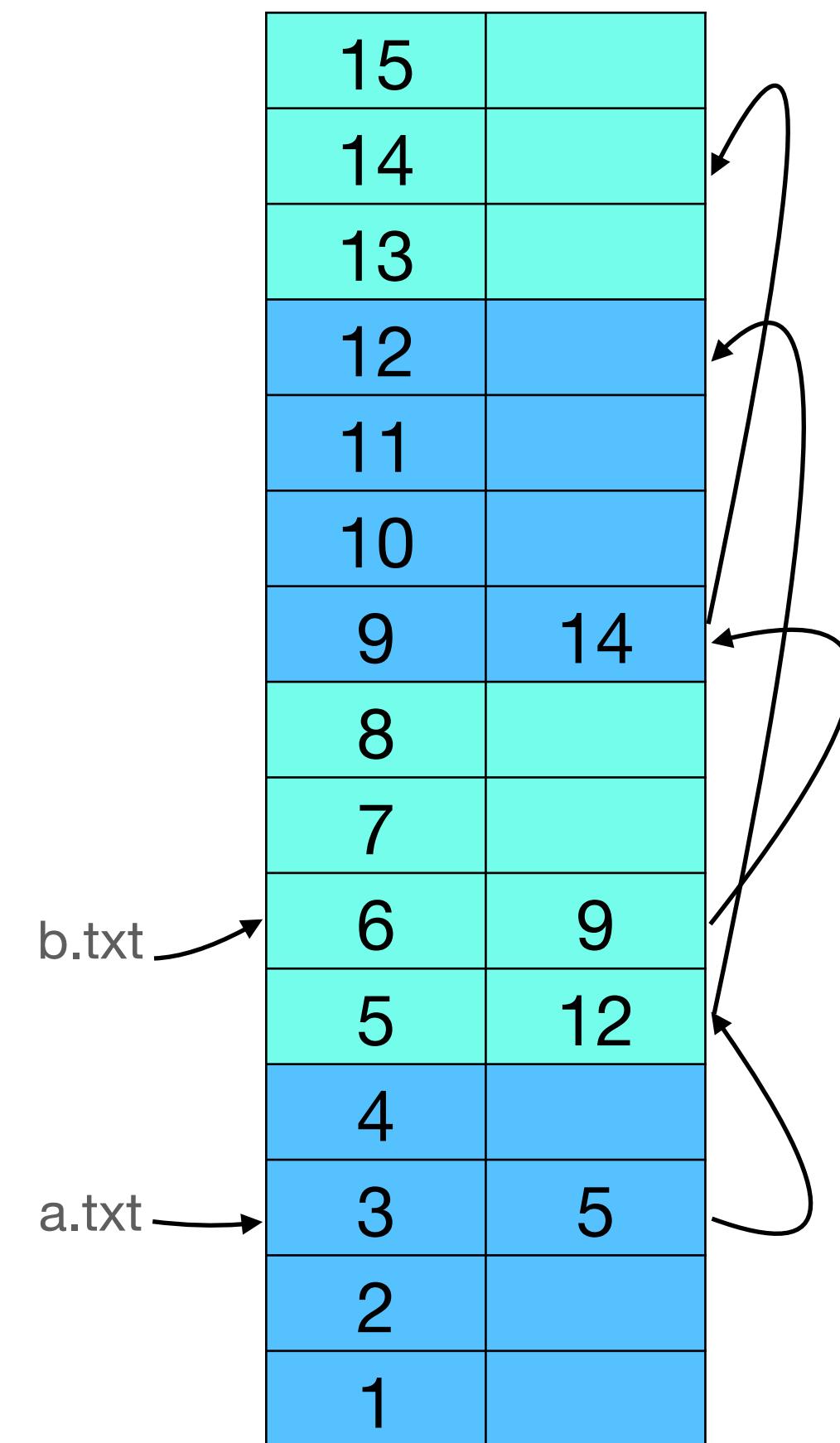
Performance

- Sequential IO
 - Better than linked list. Can find the list of blocks apriori and send requests. Disk controller can schedule them.
 - Worse than contiguous allocation since it did only 1 seek.
- Random IO
 - As fast as it can be. Find the block in memory and send disk request



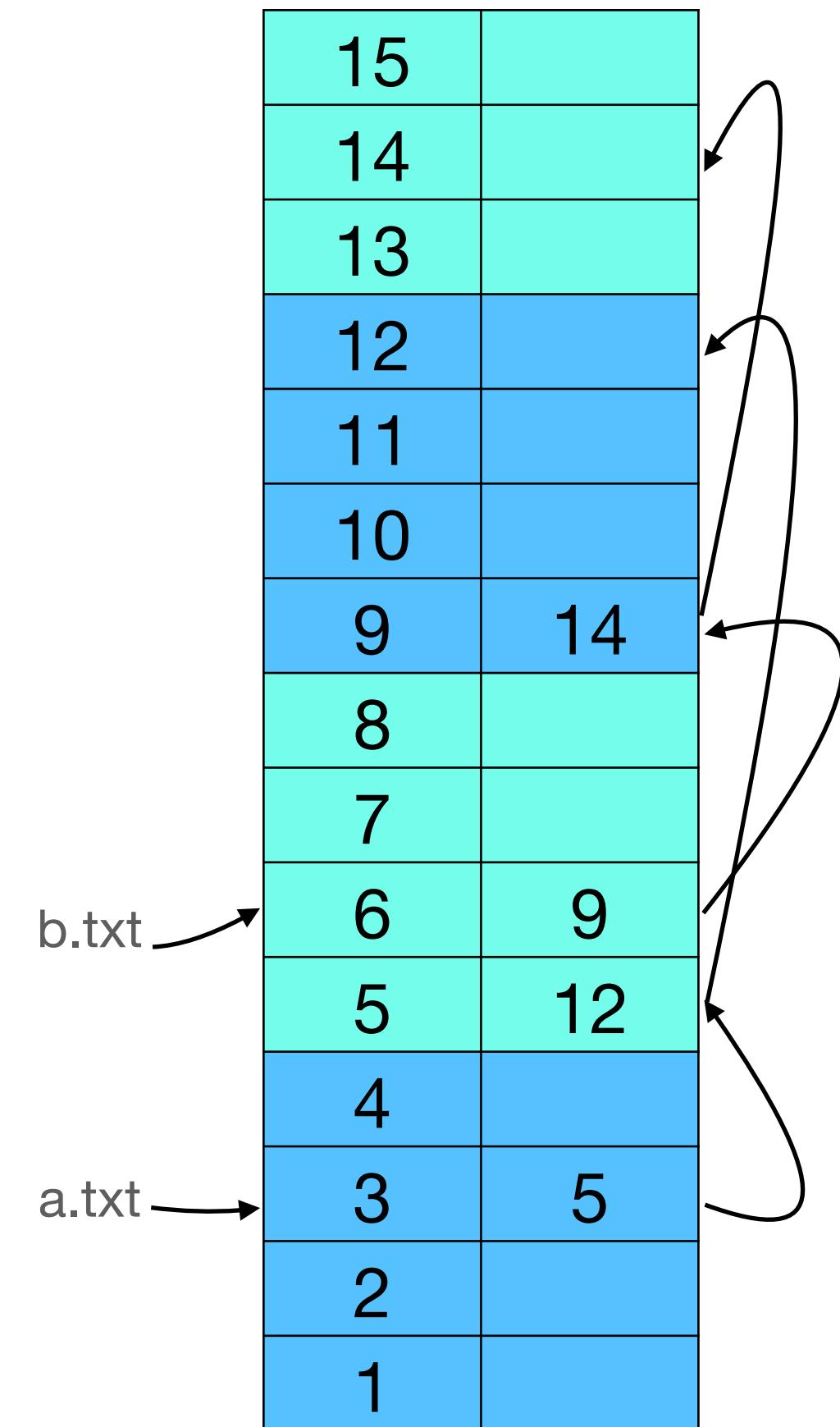
Performance

- Sequential IO
 - Better than linked list. Can find the list of blocks apriori and send requests. Disk controller can schedule them.
 - Worse than contiguous allocation since it did only 1 seek.
- Random IO
 - As fast as it can be. Find the block in memory and send disk request
 - Use buffer cache for FAT table when it does not fit in memory (1GB for FAT-32)



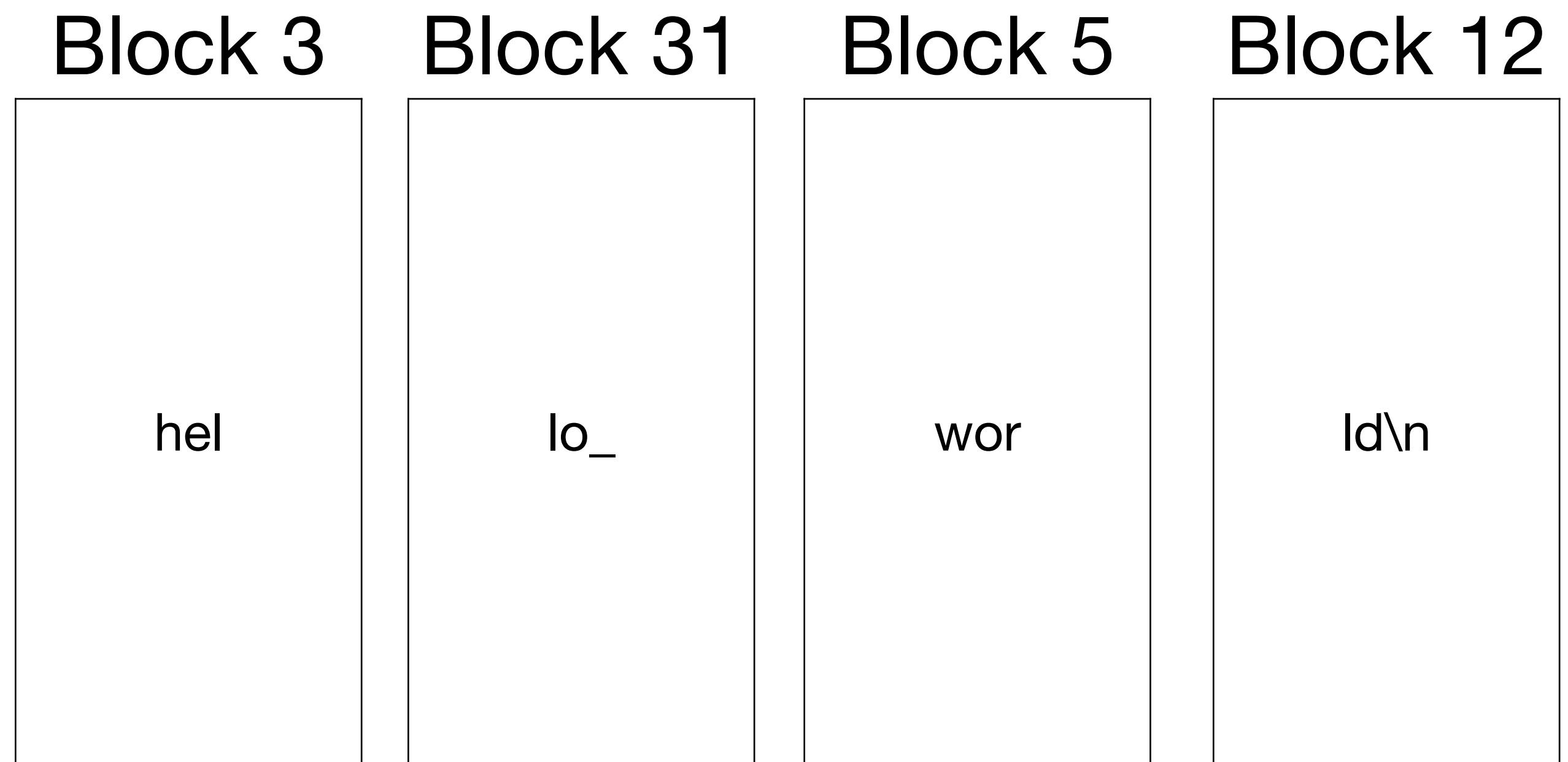
Performance

- Sequential IO
 - Better than linked list. Can find the list of blocks apriori and send requests. Disk controller can schedule them.
 - Worse than contiguous allocation since it did only 1 seek.
- Random IO
 - As fast as it can be. Find the block in memory and send disk request
- Use buffer cache for FAT table when it does not fit in memory (1GB for FAT-32)
 - To locate file's blocks, we might have to read many metadata blocks



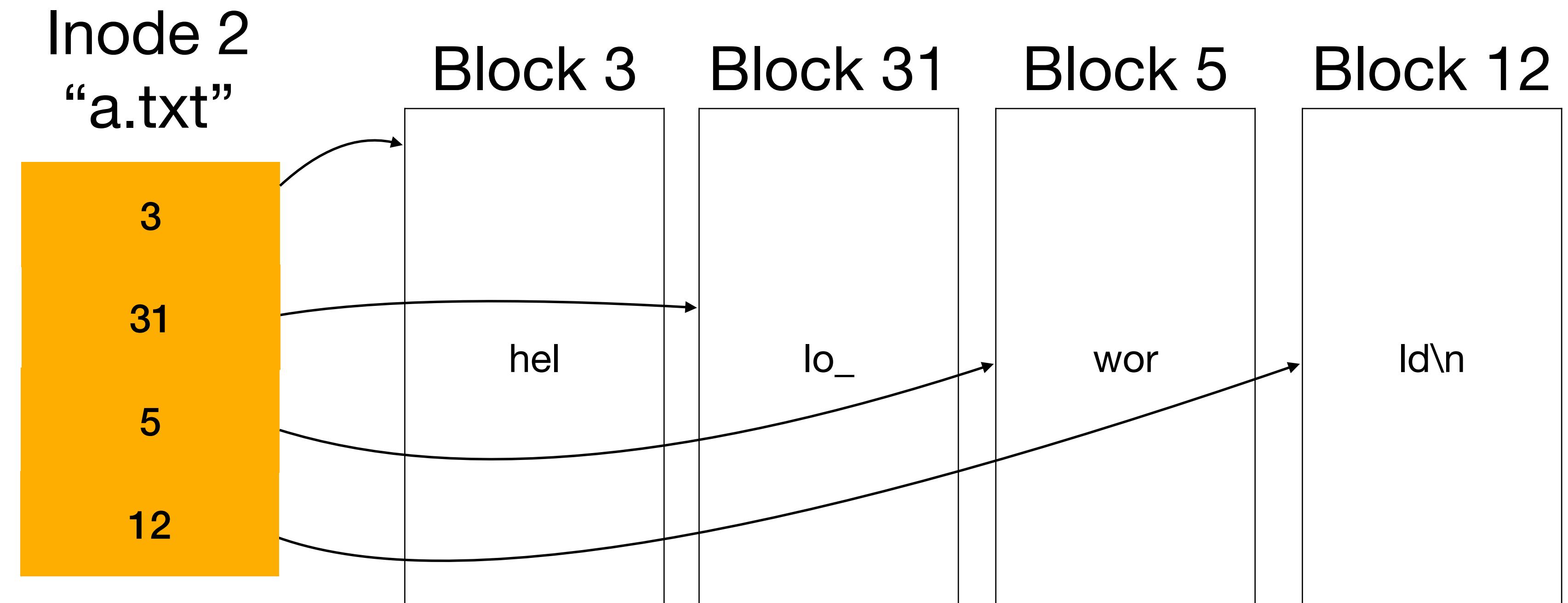
How to store files?

Index and data blocks



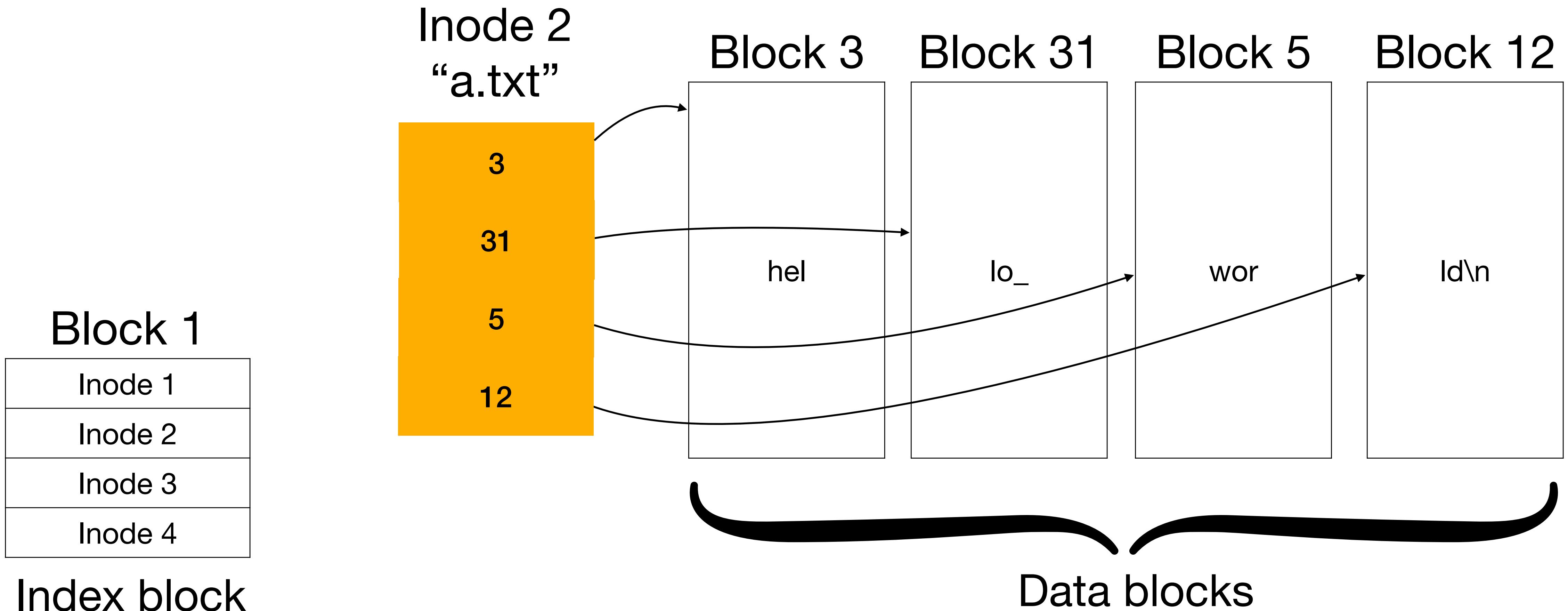
How to store files?

Index and data blocks



How to store files?

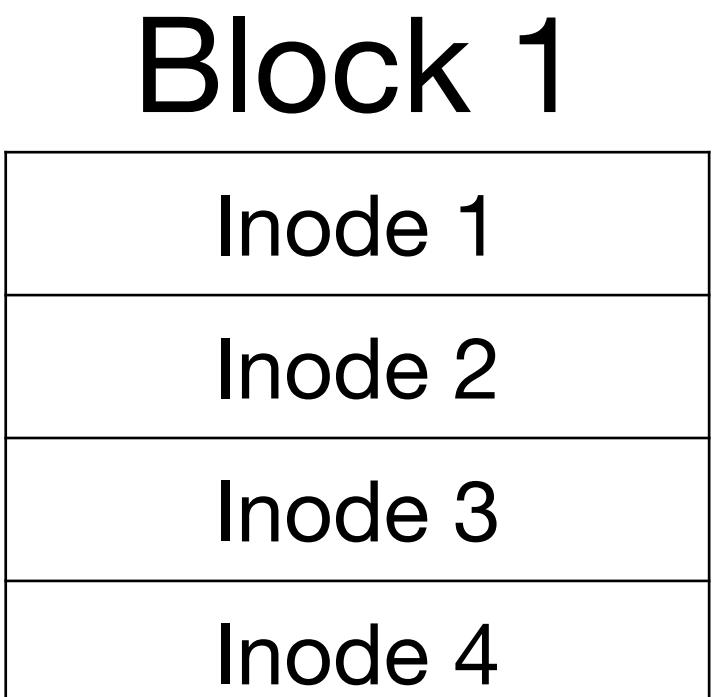
Index and data blocks



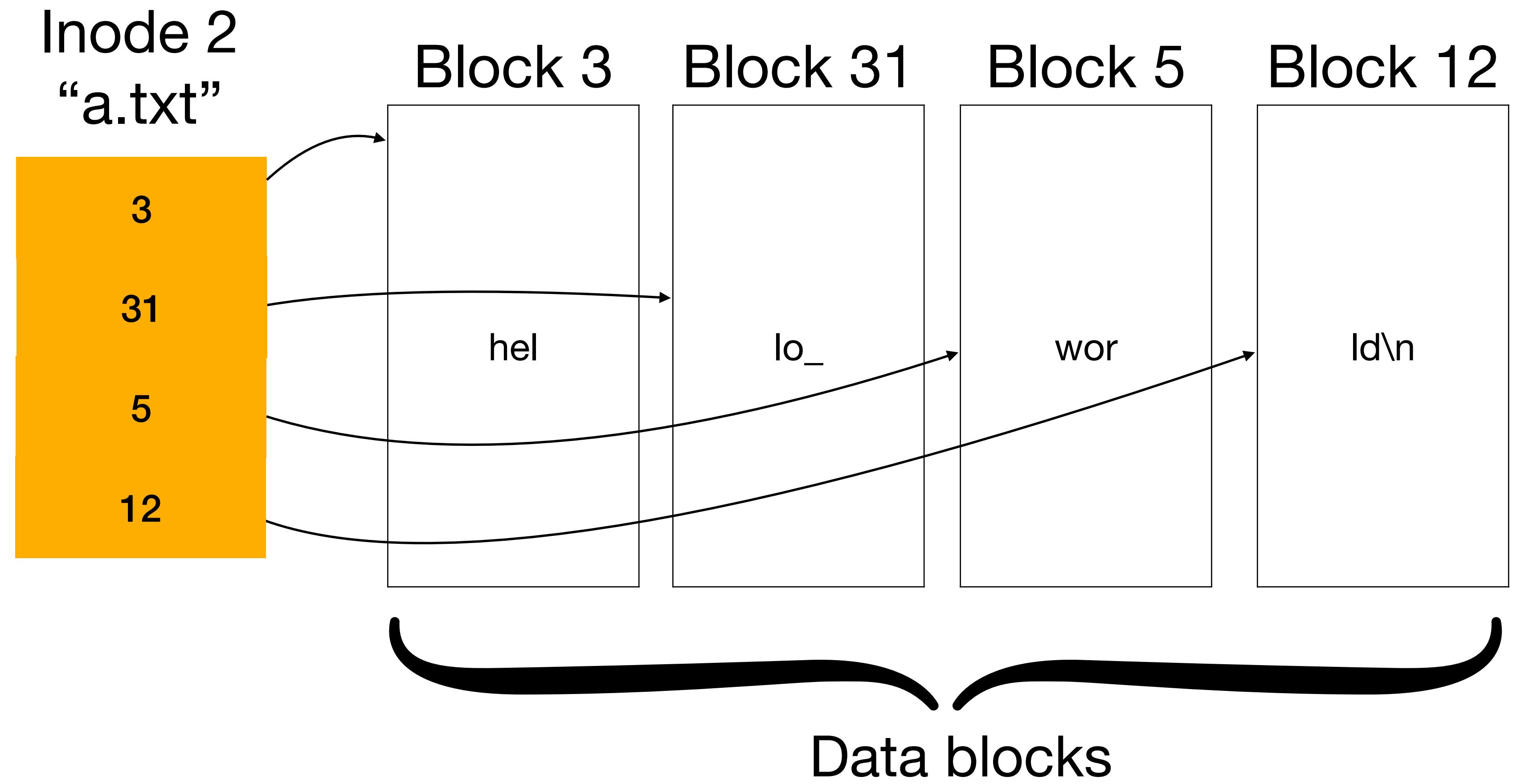
How to store files?

Index and data blocks

- One metadata block overhead for locating file's data blocks

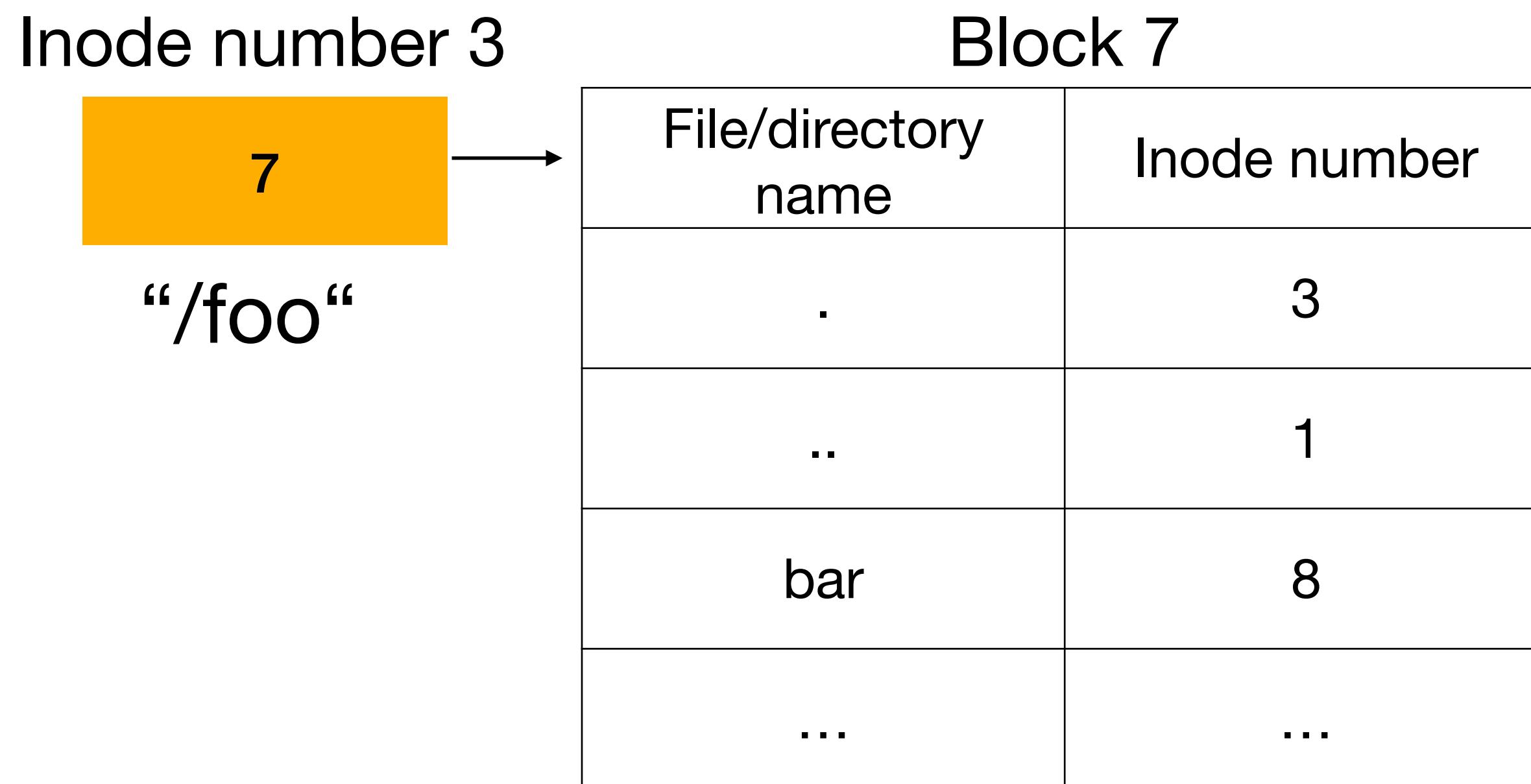


Index block



How to store directories?

/foo/bar



How to store directories?

/foo/bar

Inode number 3

7

“/foo“

Block 7

File/directory name	Inode number
.	3
..	1
bar	8
...	...

Inode number 8

16

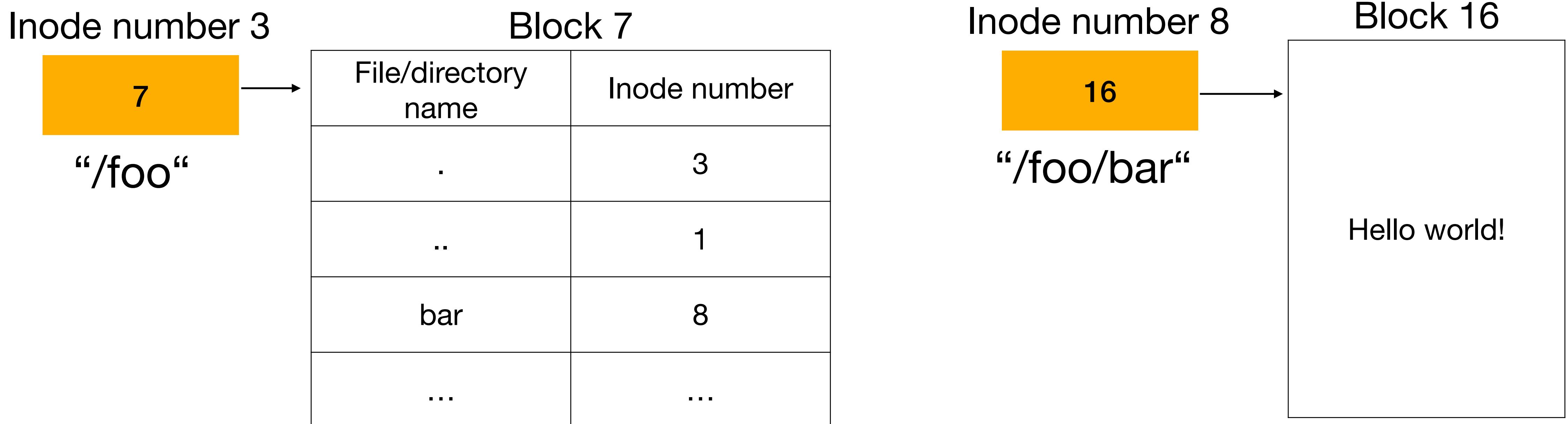
“/foo/bar“

Block 16

Hello world!

How to store directories?

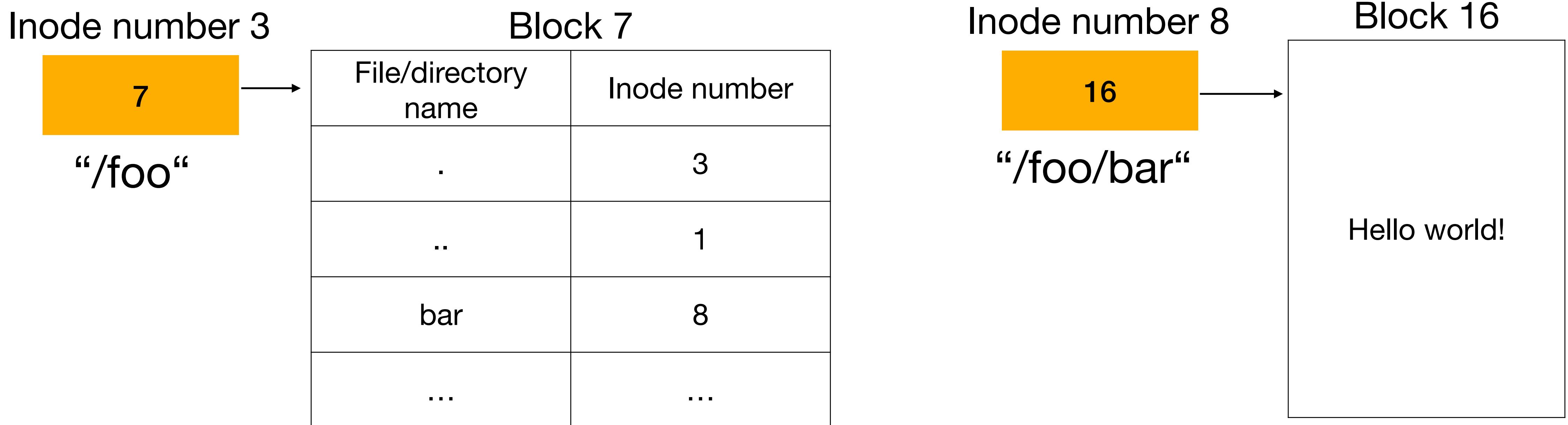
/foo/bar



- In xv6, each directory entry is 16 bytes. 32 (=512/16) directory entries in one data block

How to store directories?

/foo/bar



- In xv6, each directory entry is 16 bytes. 32 (=512/16) directory entries in one data block
- Directories have typically < 20 files and directories

Other things in inode

```
stat /tmp/file
```

```
File: /tmp/file
Size: 14          Blocks: 8          IO Block: 4096   regular file
Device: 803h/2051d Inode: 22414820    Links: 1
Access: (0600/-rw-----)  Uid: ( 1000/    dell)  Gid: ( 1000/    dell)
Access: 2024-01-24 06:29:51.395609006 +0530
Modify: 2024-01-24 06:29:51.395609006 +0530
Change: 2024-01-24 06:29:51.395609006 +0530
Birth: -
```

Modify time: last time when data nodes were changed
Change time: last time when inode was changed

Type = directory
Size
Accessed Time
Created time
Modified time
Owner user ID
Owner group ID
rwx mode
nlinks
3

File system layout

Example: /foo/bar

File system layout

Example: /foo/bar

Inode = 1 “/“

Type = directory
Size
2

Block 2

File/directory name	Inode number
.	1
foo	8

File system layout

Example: /foo/bar

Inode = 1 “/“

Type = directory
Size
2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo“

Type = directory
Size
16

Block 16

File/directory name	Inode number
.	8
..	1
bar	9

File system layout

Example: /foo/bar

Inode = 1 “/“

Type = directory
Size
2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo“

Type = directory
Size
16

Block 16

File/directory name	Inode number
.	8
..	1
bar	9

Inode = 9 “/foo/bar”

Type = file
Size
3
31
5
12

Block 3

hel

Block 31

lo

Block 5

wor

Block 12

ld

Reading a file

Example: /foo/bar

Inode = 1 “/“

Type = directory
Size
Access time
2

Inode = 8 “/foo“

Type = directory
Size
Access time
16

Block 2

File/directory name	Inode number
.	1
foo	8

Block 16

File/directory name	Inode number
.	8
..	1
bar	9

Inode = 9 “/foo/bar”

Type = file
Size
Access time
3
31
5
12

Block 3

hel

Block 31

lo

Block 5

wor

Block 12

ld

Reading a file

Example: /foo/bar

Inode = 1 “/”

Type = directory
Size
Access time
2

Inode = 8 “/foo”

Type = directory
Size
Access time
16

Block 2

File/directory name	Inode number
.	1
foo	8

Block 16

File/directory name	Inode number
.	8
..	1
bar	9

Inode = 9 “/foo/bar”

Type = file
Size
Access time
3
31
5
12

Block 3

hel

Block 31

lo

Block 5

wor

Block 12

ld

```
char buf[10];
fd = open("/foo/bar", O_RDONLY)
while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}
close(fd);
```

Reading a file

Example: /foo/bar

Inode = 1 “/”

Type = directory
Size
Access time
2

Inode = 8 “/foo”

Type = directory
Size
Access time
16

Block 2

File/directory name	Inode number
.	1
foo	8

Block 16

File/directory name	Inode number
.	8
..	1
bar	9

Inode = 9 “/foo/bar”

Type = file
Size
Access time
3
31
5
12

```
char buf[10];
fd = open("/foo/bar", O_RDONLY)

while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}

close(fd);
```

Block 3

hel

Block 31

lo

Block 5

wor

Block 12

ld

Reading a file

Example: /foo/bar

Inode = 1 “/”

Type = directory
Size
Access time
2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”

Type = directory
Size
Access time
16

Block 16

File/directory name	Inode number
.	8
..	1
bar	9

Inode = 9 “/foo/bar”

Type = file
Size
Access time
3
31
5
12

```
char buf[10];
fd = open("/foo/bar", O_RDONLY)

while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}

close(fd);
```

Block 3

hel

Block 31

lo

Block 5

wor

Block 12

ld

Reading a file

Example: /foo/bar

Inode = 1 “/”

Type = directory
Size
Access time
2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”

Type = directory
Size
Access time
16

Block 16

File/directory name	Inode number
.	8
..	1
bar	9

Inode = 9 “/foo/bar”

Type = file
Size
Access time
3
31
5
12

```
char buf[10];
fd = open("/foo/bar", O_RDONLY)

while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}

close(fd);
```

Block 3

hel

Block 31

lo

Block 5

wor

Block 12

ld

Reading a file

Example: /foo/bar

Inode = 1 “/”

Type = directory
Size
Access time
2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”

Type = directory
Size
Access time
16

Block 16

File/directory name	Inode number
.	8
..	1

Inode = 9 “/foo/bar”

Type = file
Size
Access time
3
31
5
12

```
char buf[10];
fd = open("/foo/bar", O_RDONLY)

while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}

close(fd);
```

Block 3

hel

Block 31

lo

Block 5

wor

Block 12

ld

Reading a file

Example: /foo/bar

Inode = 1 “/”

Type = directory
Size
Access time
2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”

Type = directory
Size
Access time
16

Block 16

File/directory name	Inode number
.	8
..	1

Inode = 9 “/foo/bar”

Type = file
Size
Access time
5

```
char buf[10];
fd = open("/foo/bar", O_RDONLY)

while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}

close(fd);
```

Block 3

hel

Block 31

lo

Block 5

wor

Block 12

ld

Reading a file

Example: /foo/bar

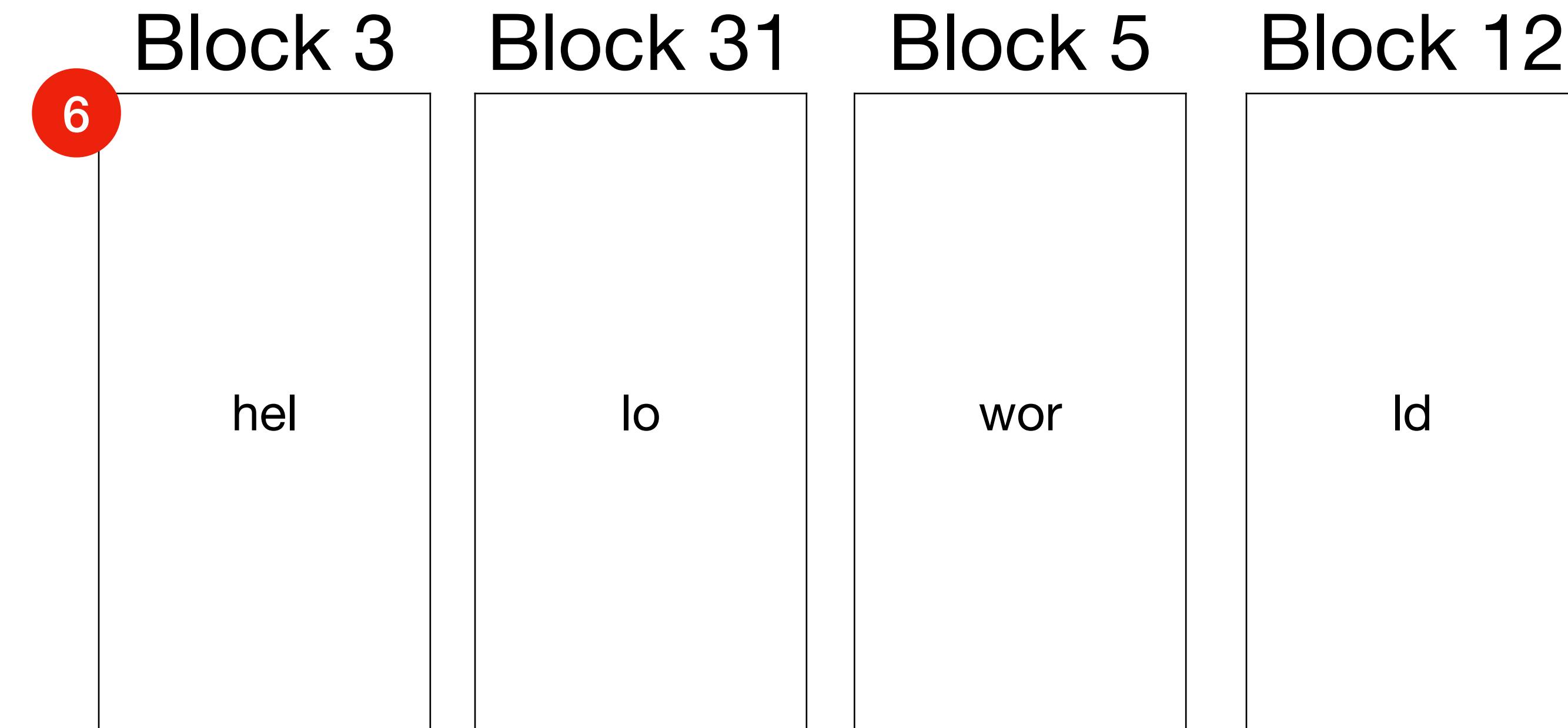
Inode = 1 “/”
Type = directory
Size
Access time

Inode = 8 “/foo”
Type = directory
Size
Access time

Block 2	
File/directory name	Inode number
.	1
foo	8

Block 16	
File/directory name	Inode number
.	8
..	1

Inode = 9 “/foo/bar”
Type = file
Size
Access time
5
3
31
5
12



Reading a file

Example: /foo/bar

Inode = 1 “/”
Type = directory
Size
Access time

1 2

Block 2	
File/directory name	Inode number
.	1
foo	8

2

Inode = 8 “/foo”
Type = directory
Size
Access time

3 16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	9

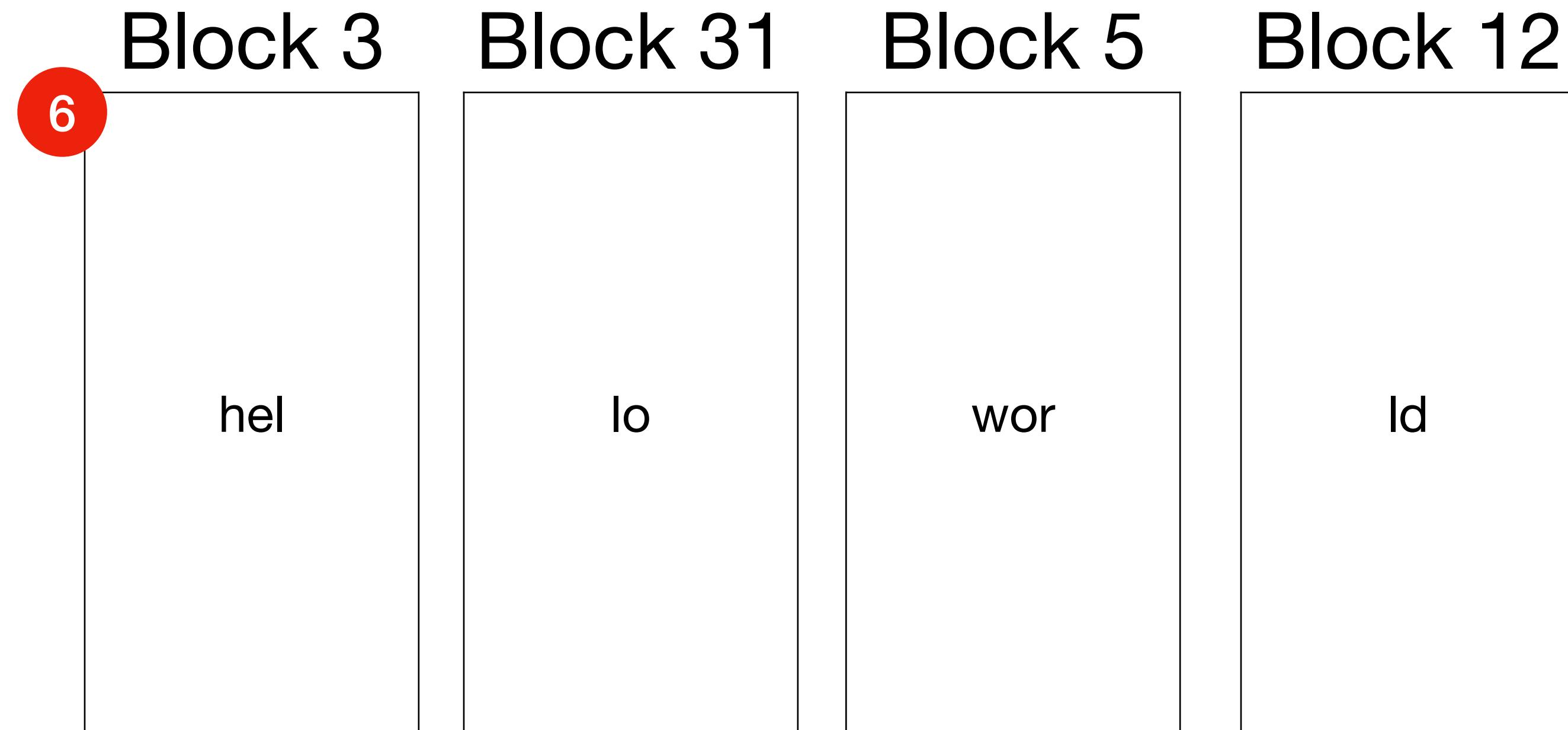
4

Inode = 9 “/foo/bar”
Type = file
Size
Access time
5 3 31 7

```
char buf[10];
fd = open("/foo/bar", O_RDONLY)

while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}

close(fd);
```



Reading a file

Example: /foo/bar

Inode = 1 “/”
Type = directory
Size
Access time

1 2

Block 2	
File/directory name	Inode number
.	1
foo	8

2

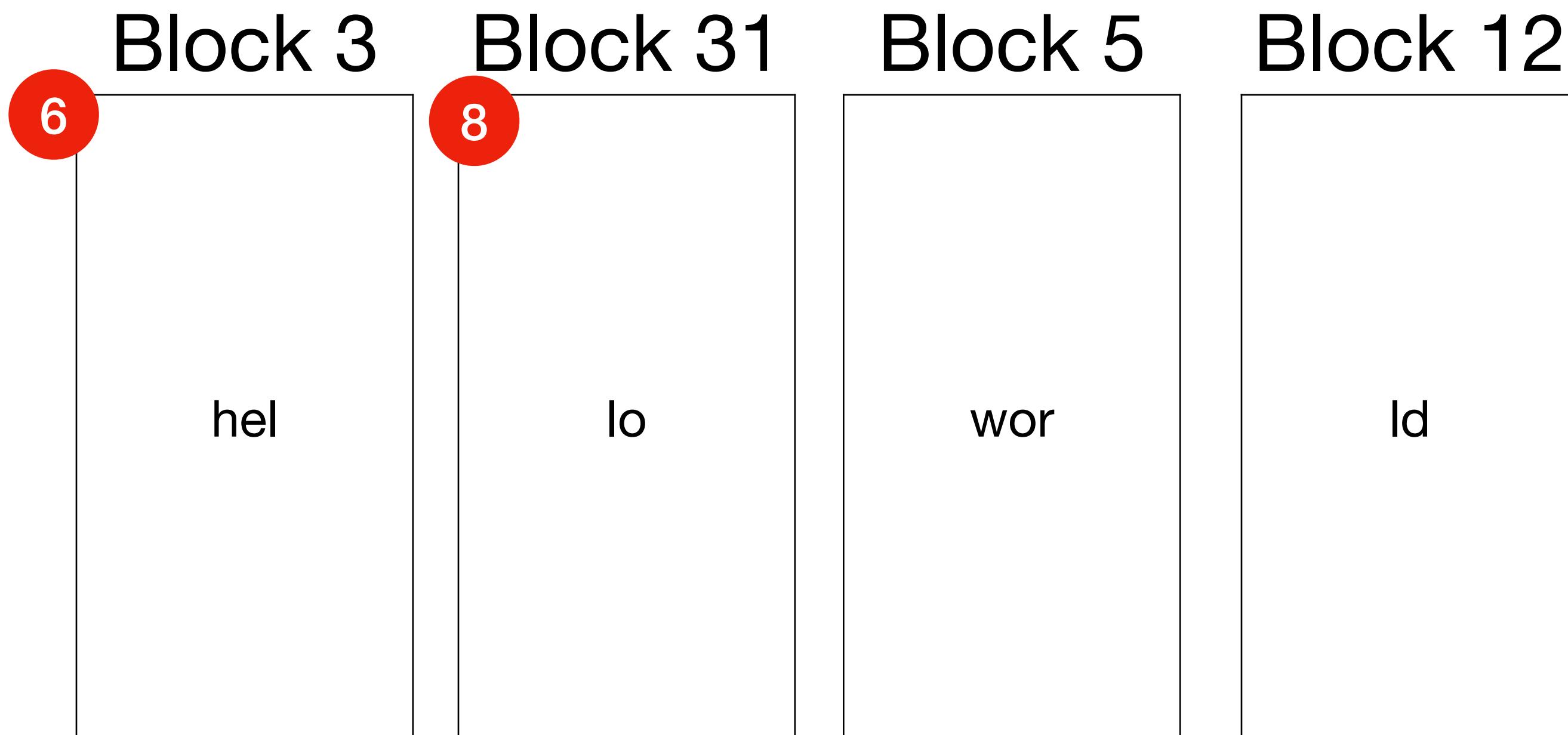
Inode = 8 “/foo”
Type = directory
Size
Access time

3 16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	9

4

Inode = 9 “/foo/bar”
Type = file
Size
Access time
5 3 31 7



```
char buf[10];
fd = open("/foo/bar", O_RDONLY)
while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}
close(fd);
```

Reading a file

Example: /foo/bar

Inode = 1 “/”
Type = directory
Size
Access time

1 2

Block 2	
File/directory name	Inode number
.	1
foo	8

2

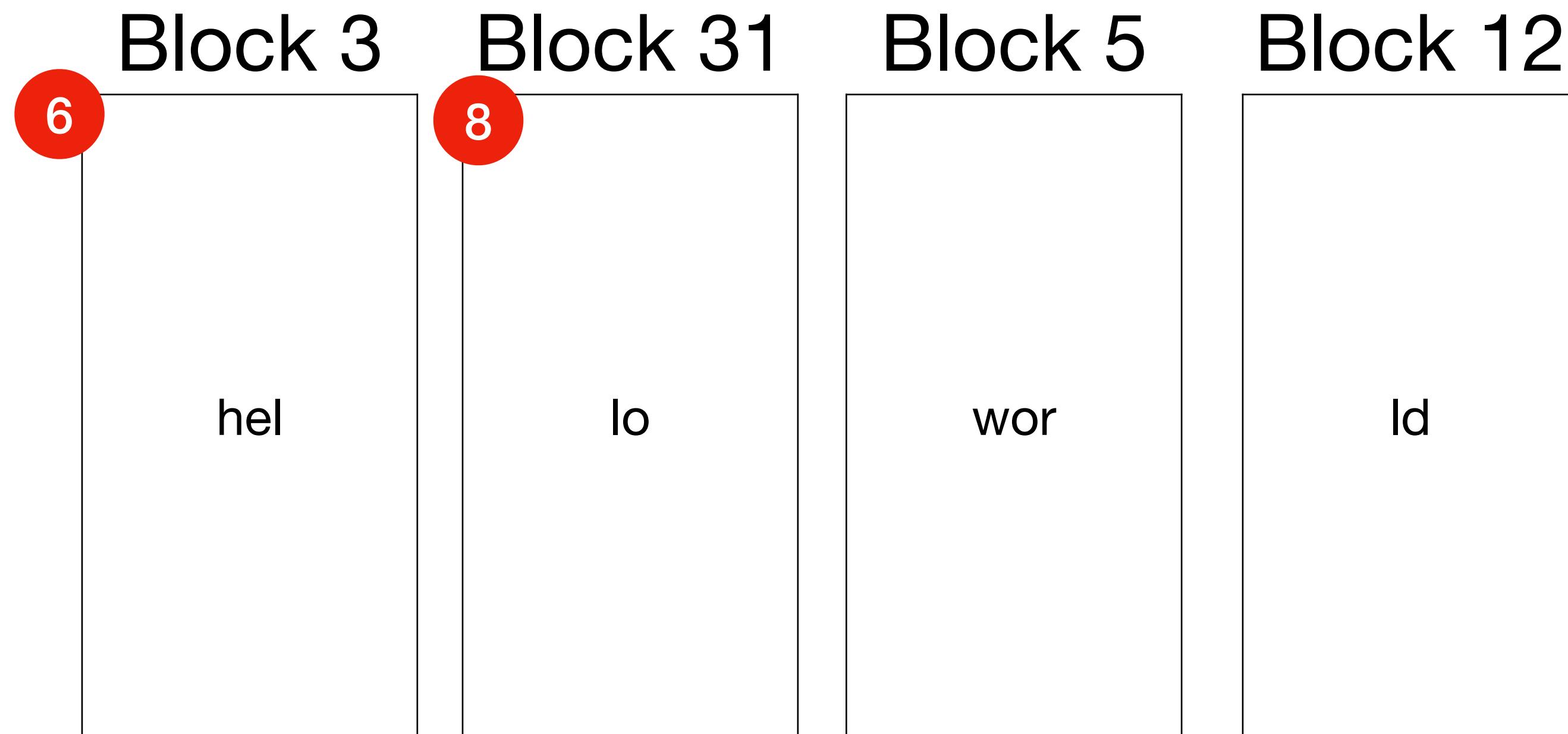
Inode = 8 “/foo”
Type = directory
Size
Access time

3 16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	9

4

Inode = 9 “/foo/bar”
Type = file
Size
Access time
5 3 31 7



Reading a file

Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
Access time	2

Block 2	
File/directory name	Inode number
.	1

2	foo	8
---	-----	---

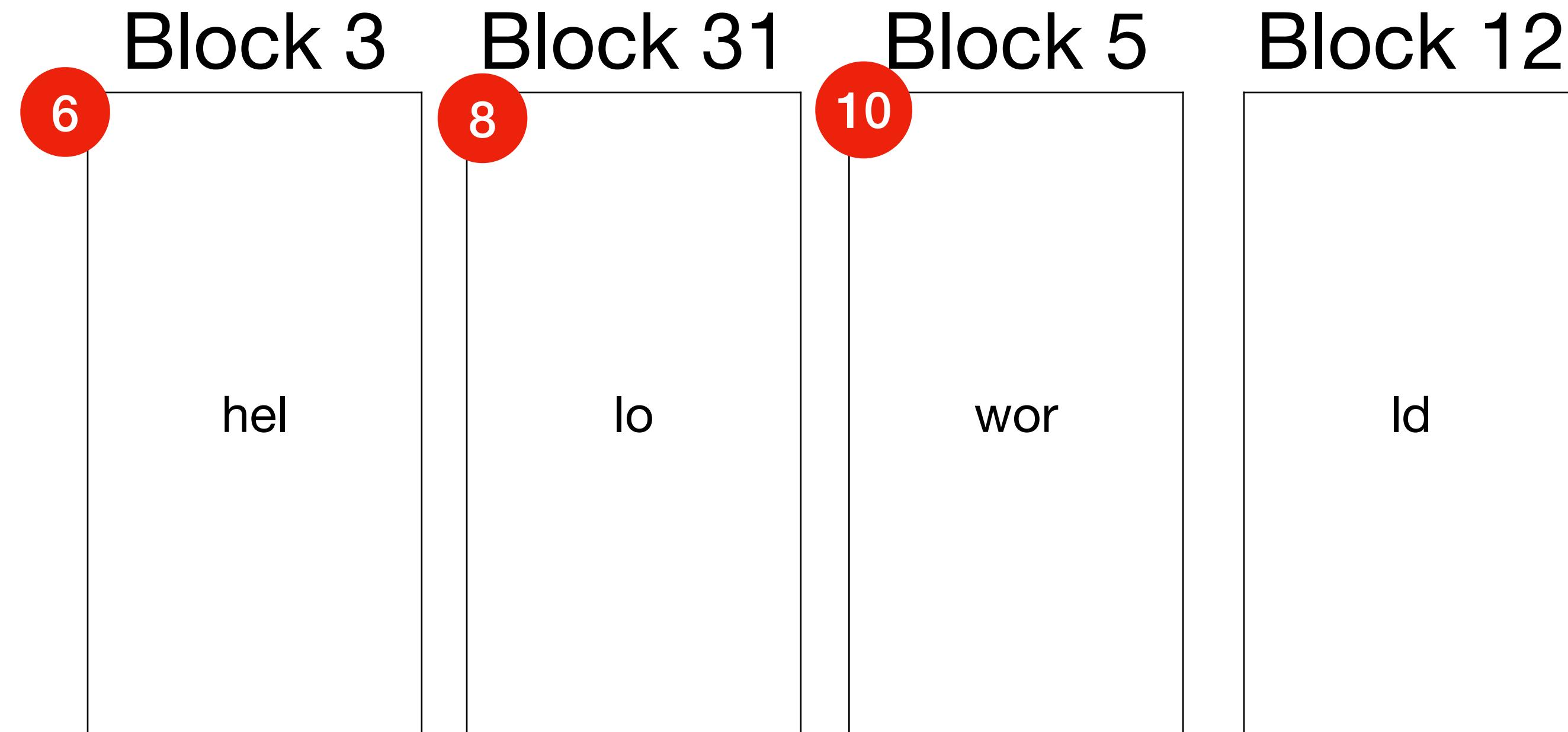
Inode = 8 “/foo”	
Type = directory	
Size	
Access time	16

Block 16	
File/directory name	Inode number
.	8

4	bar	9
---	-----	---

Inode = 9 “/foo/bar”	
Type = file	
Size	
Access time	

5	3
9	31
6	5
8	12



```
char buf[10];
fd = open("/foo/bar", O_RDONLY)
while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}
close(fd);
```

Reading a file

Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
Access time	2

Block 2	
File/directory name	Inode number
.	1

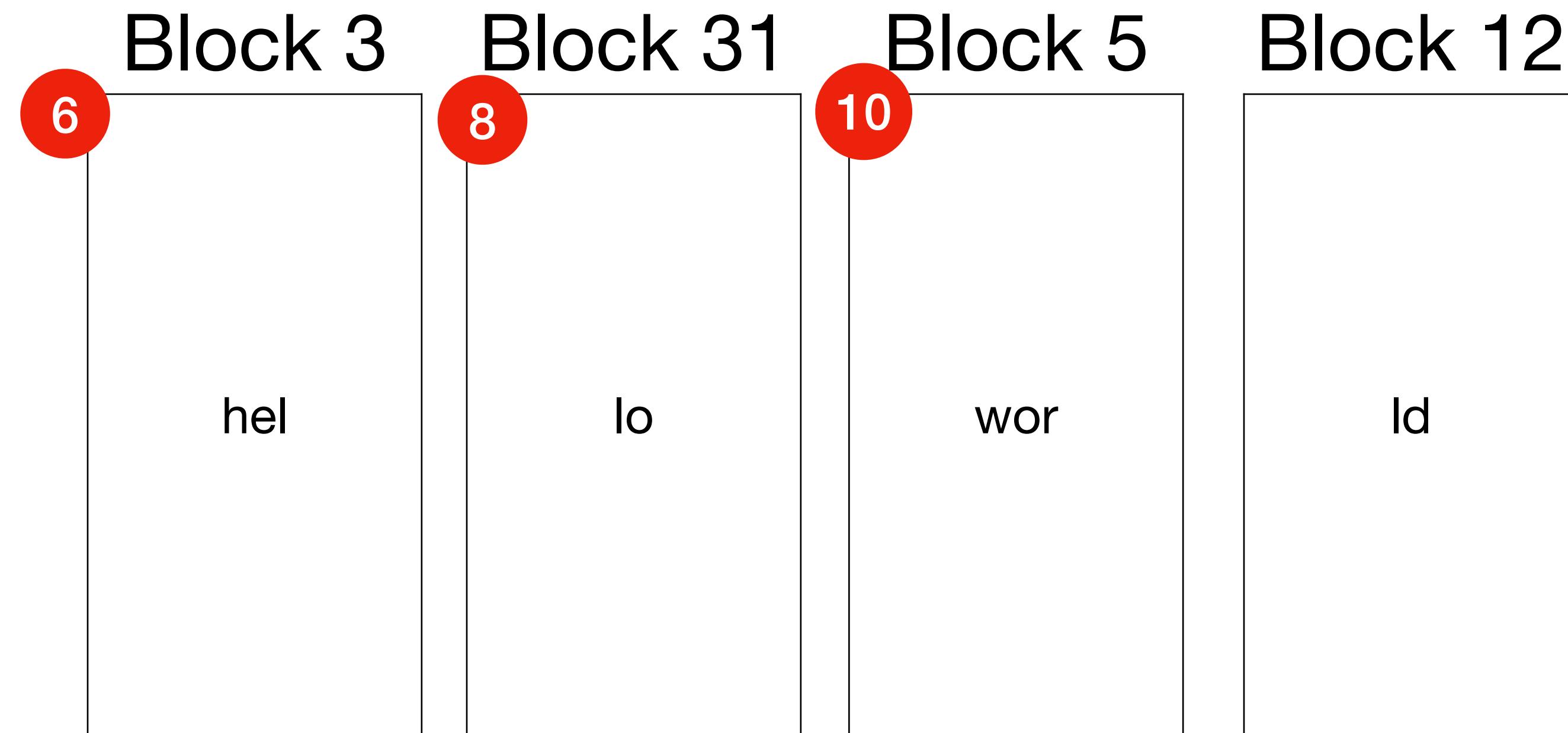
2	foo	8
---	-----	---

Inode = 8 “/foo”	
Type = directory	
Size	
Access time	16

Block 16	
File/directory name	Inode number
.	8

4	bar	9
---	-----	---

Inode = 9 “/foo/bar”	
Type = file	
Size	
Access time	
5	3
31	7
9	5
12	11



```
char buf[10];
fd = open("/foo/bar", O_RDONLY)

while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}

close(fd);
```

Reading a file

Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
Access time	2

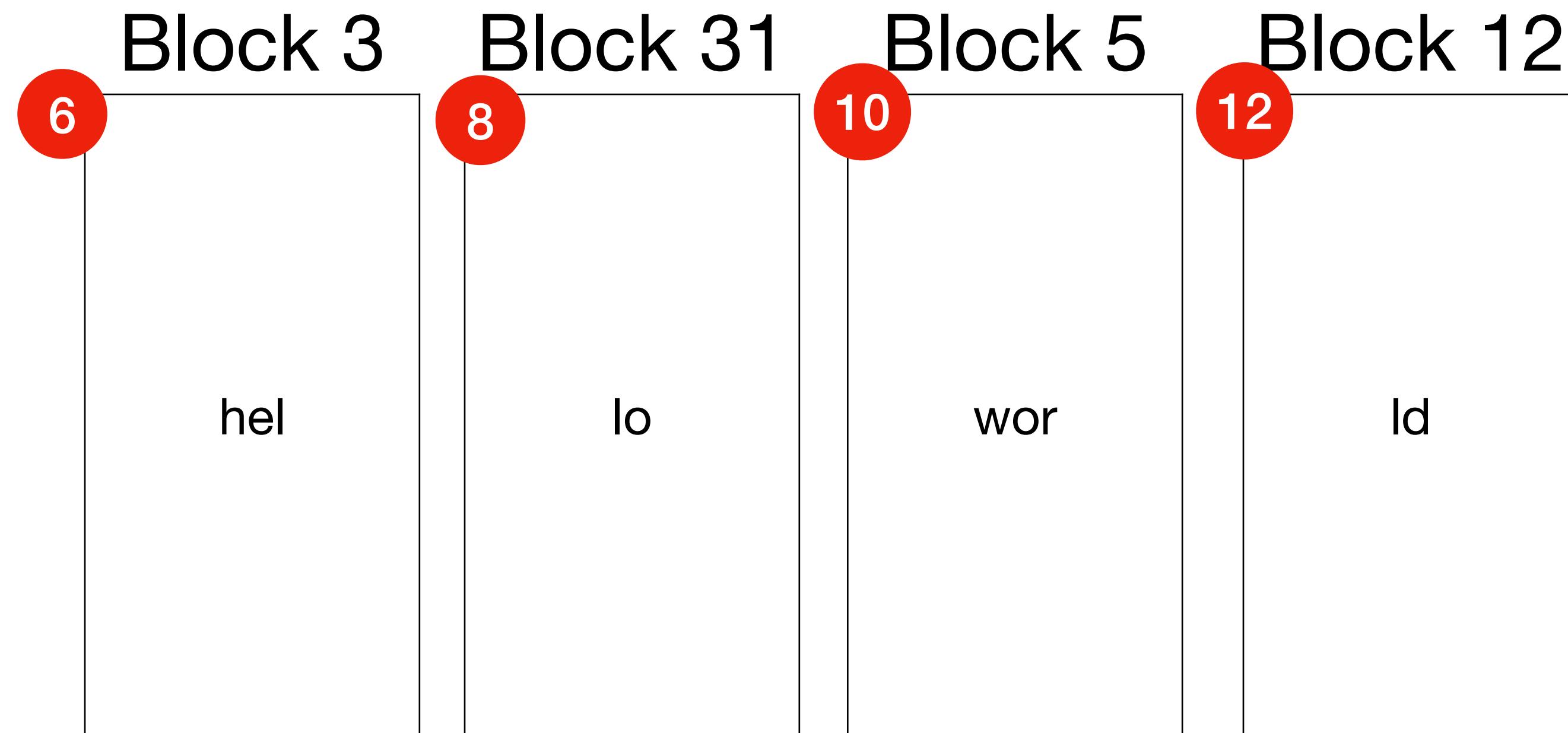
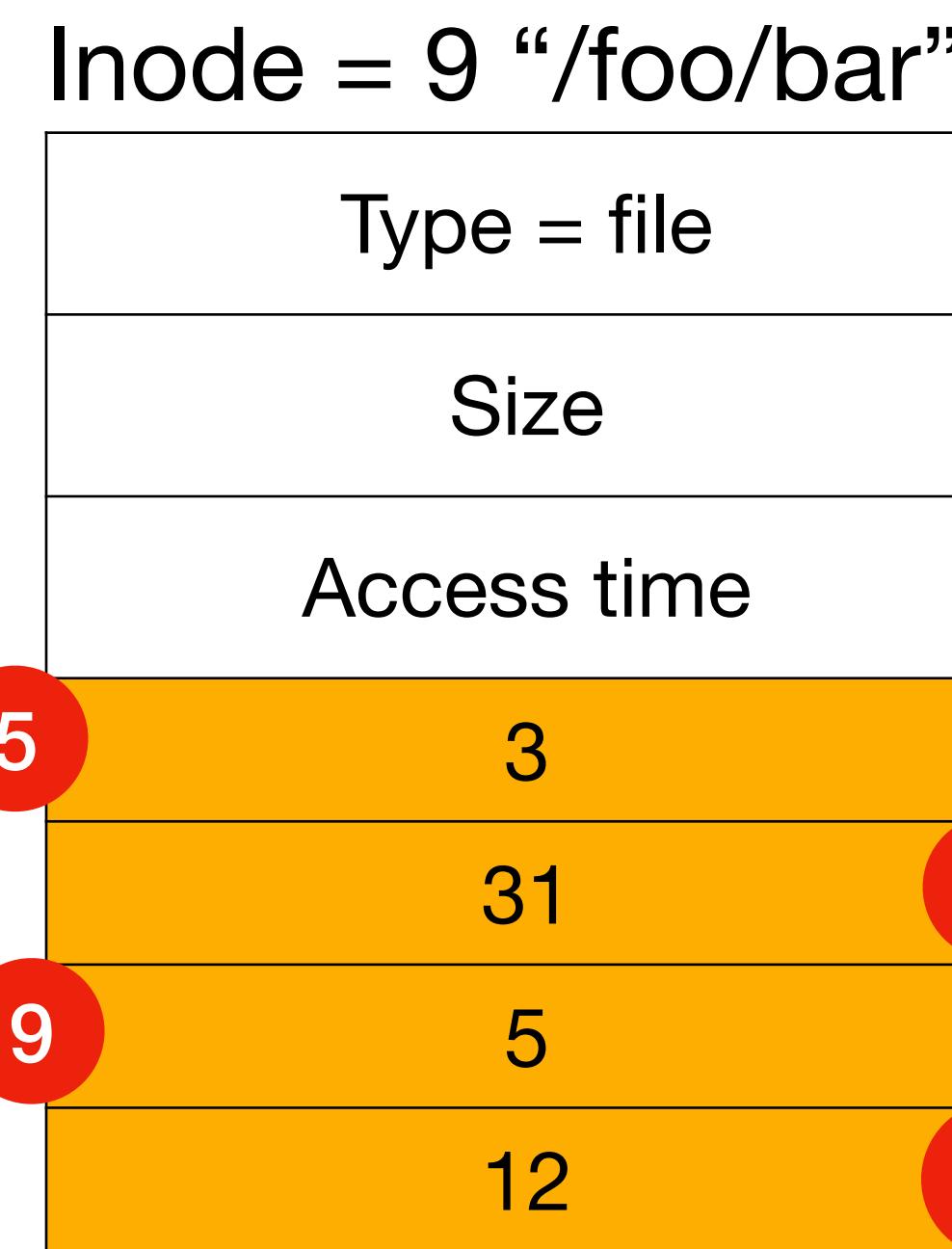
Block 2	
File/directory name	Inode number
.	1

2	foo	8
---	-----	---

Inode = 8 “/foo”	
Type = directory	
Size	
Access time	16

Block 16	
File/directory name	Inode number
.	8

4	bar	9
---	-----	---



Reading a file

Example: /foo/bar

Inode = 1 “/”
Type = directory
Size
Access time

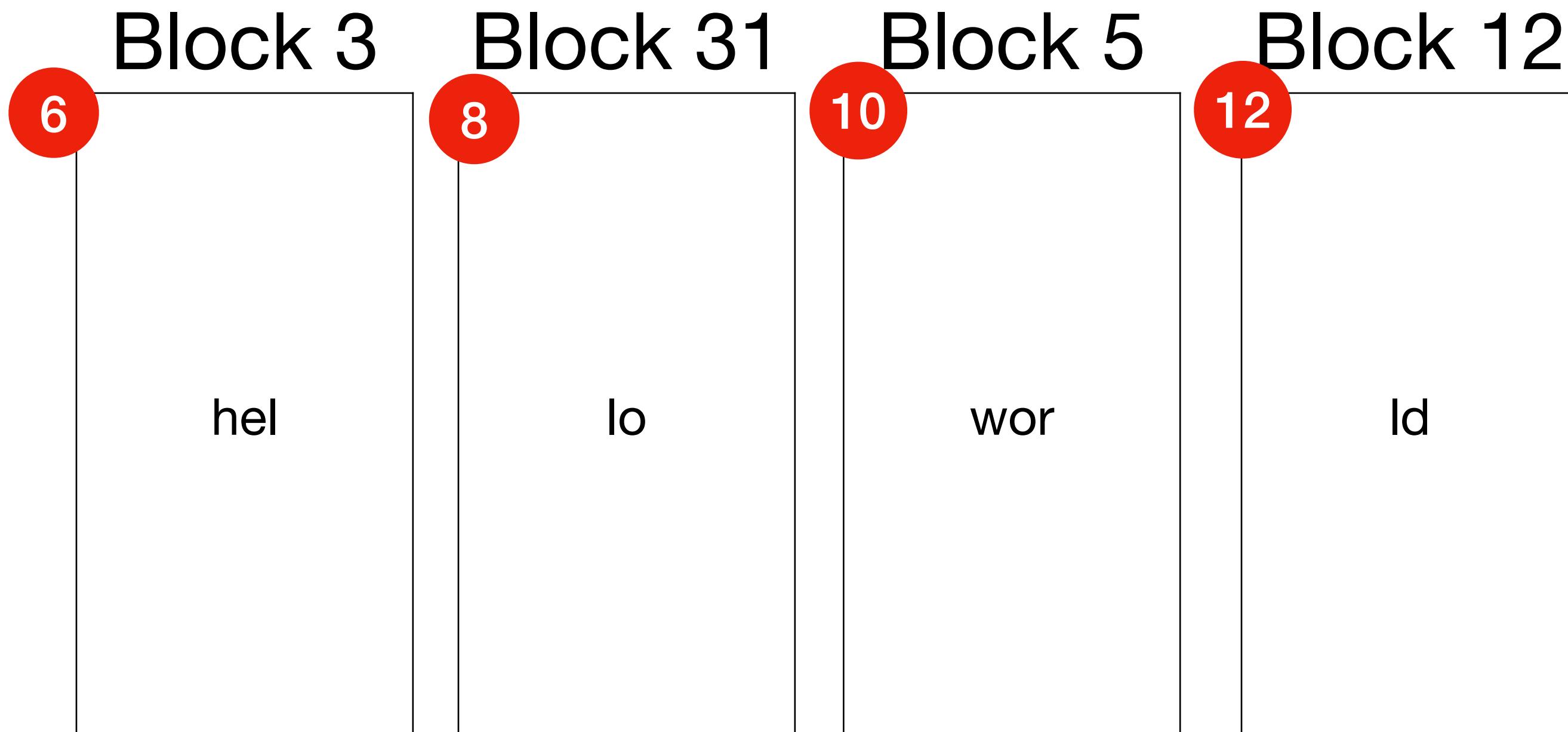
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”
Type = directory
Size
Access time

Block 16	
File/directory name	Inode number
.	8
..	1
bar	9

Inode = 9 “/foo/bar”

Type = file
Size
Access time



```
char buf[10];
fd = open("/foo/bar", O_RDONLY)

while(read(fd, &buf, 10) > 0) {
    // print buf etc.
}

close(fd);
```

Storing large files

Storing large files

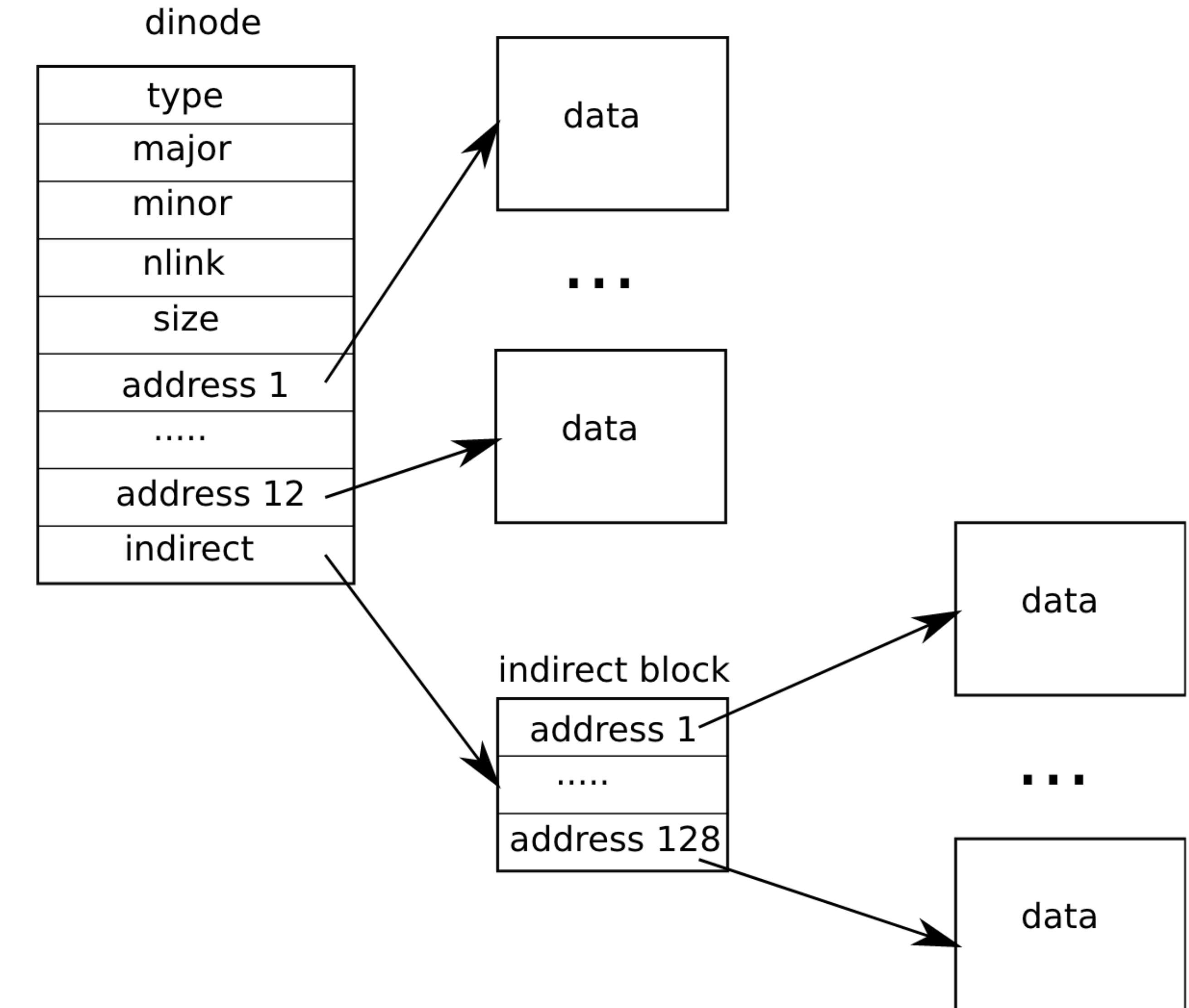
- Keep inodes of fixed size (64 bytes) for simplicity

Storing large files

- Keep inodes of fixed size (64 bytes) for simplicity
- 8 inodes in a 512 byte block

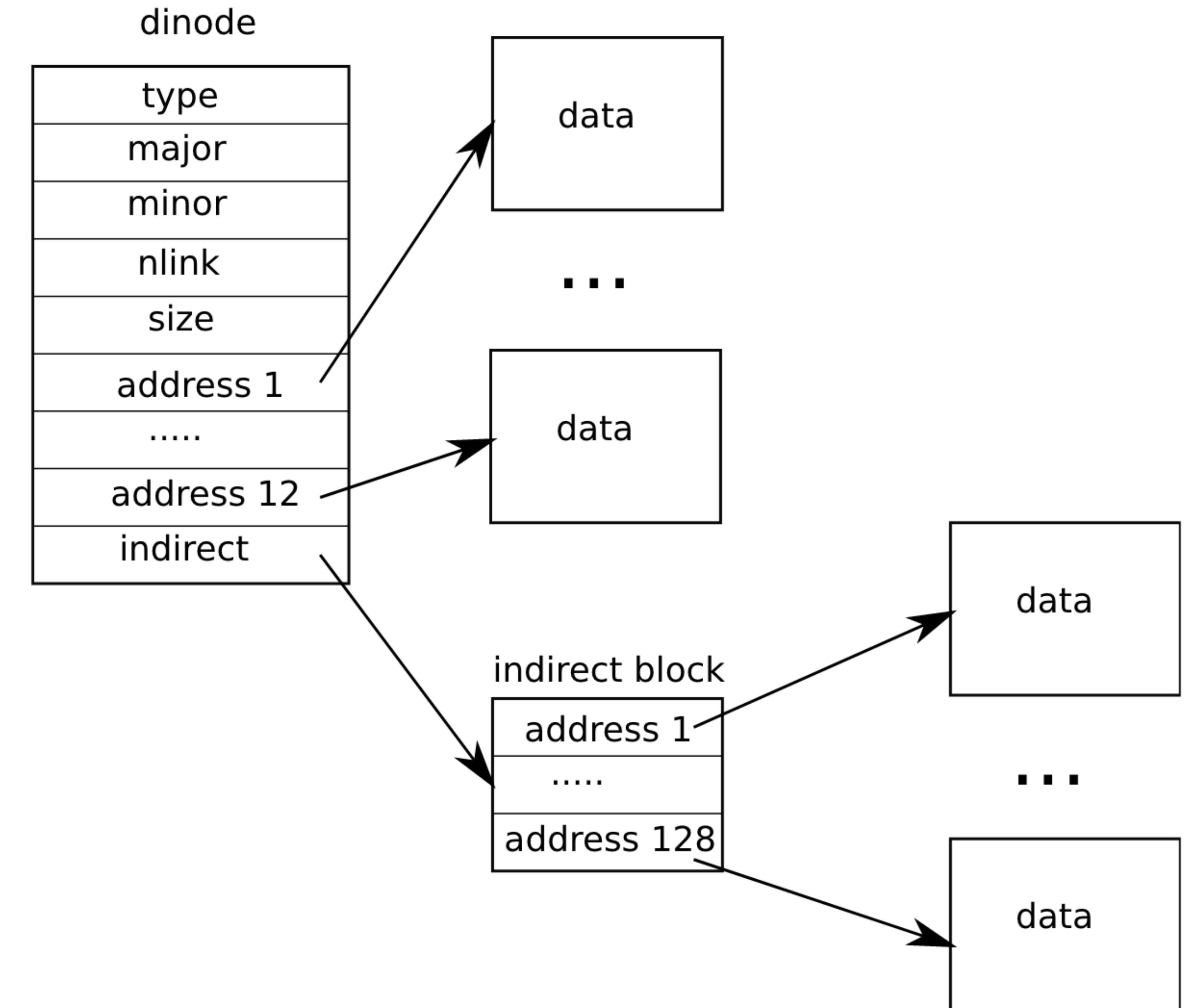
Storing large files

- Keep inodes of fixed size (64 bytes) for simplicity
- 8 inodes in a 512 byte block



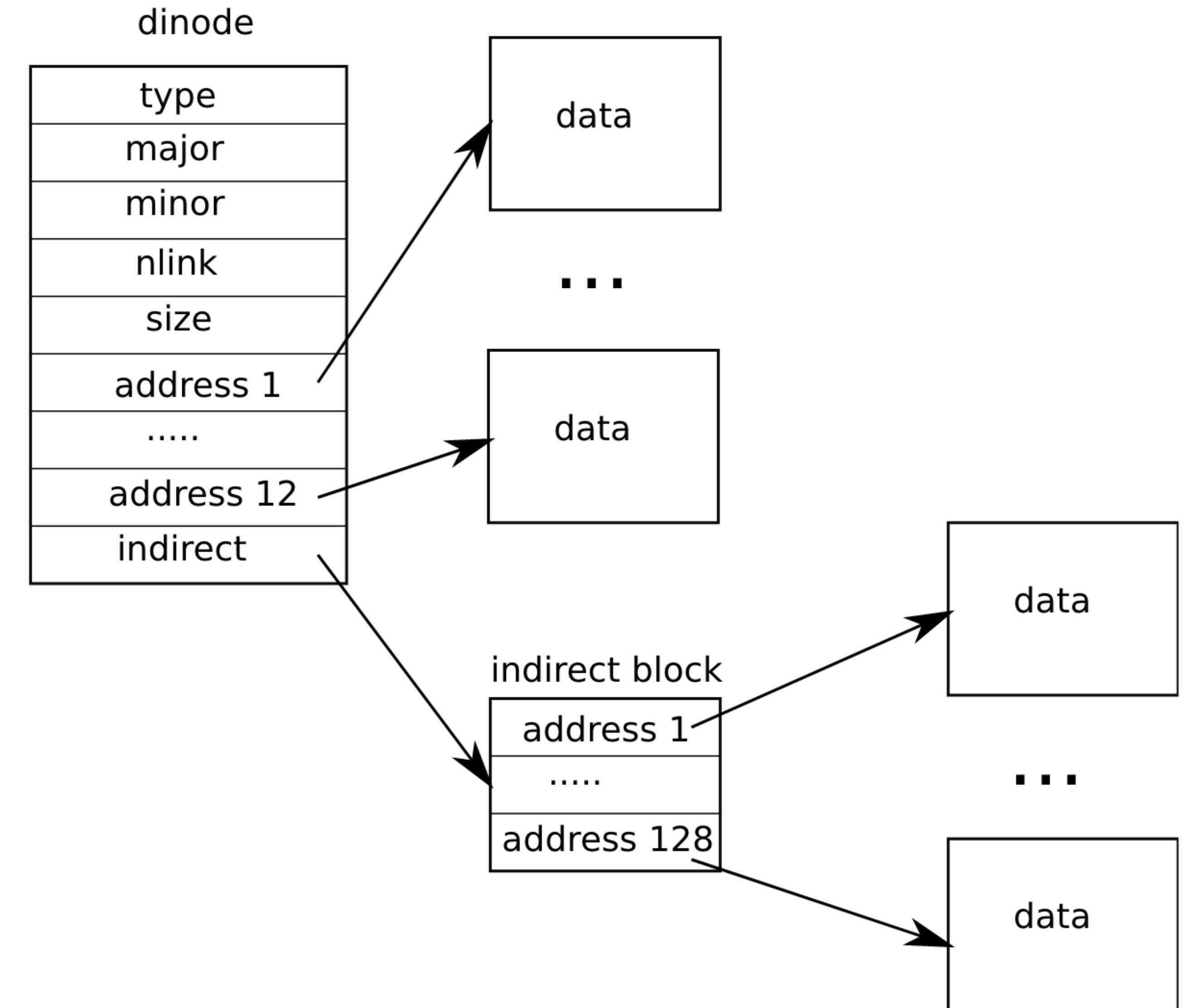
Storing large files

- Keep inodes of fixed size (64 bytes) for simplicity
- 8 inodes in a 512 byte block
- Most files are < 2KB



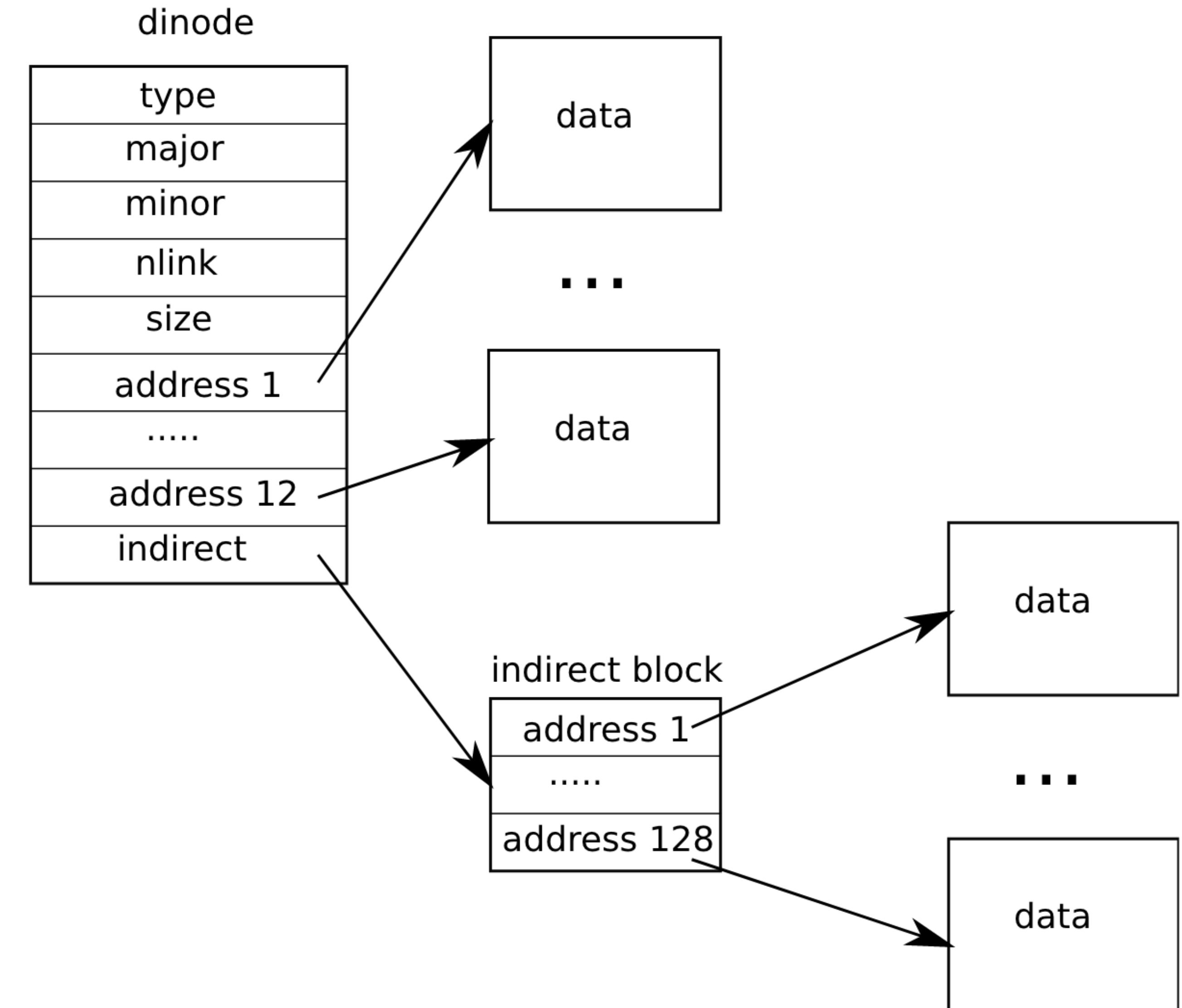
Storing large files

- Keep inodes of fixed size (64 bytes) for simplicity
- 8 inodes in a 512 byte block
- Most files are < 2KB
- $12 \times 512 \text{ bytes} = 6\text{KB}$



Storing large files

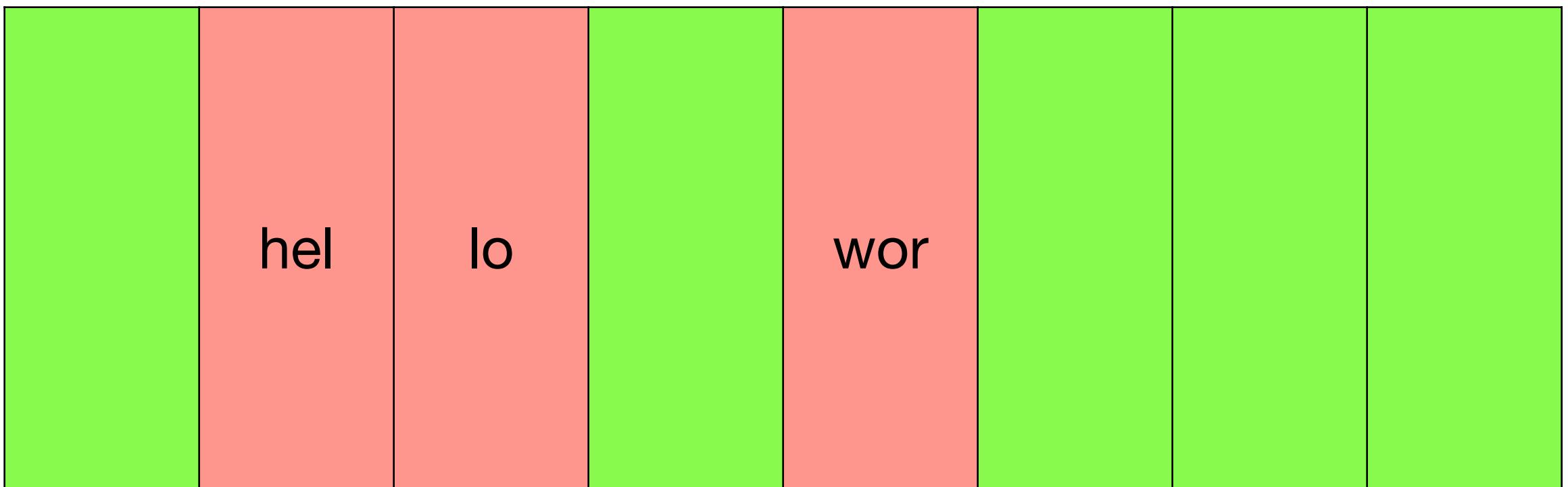
- Keep inodes of fixed size (64 bytes) for simplicity
- 8 inodes in a 512 byte block
- Most files are < 2KB
- $12 \times 512 \text{ bytes} = 6\text{KB}$
- Most files do not need indirect block



How to track free blocks?

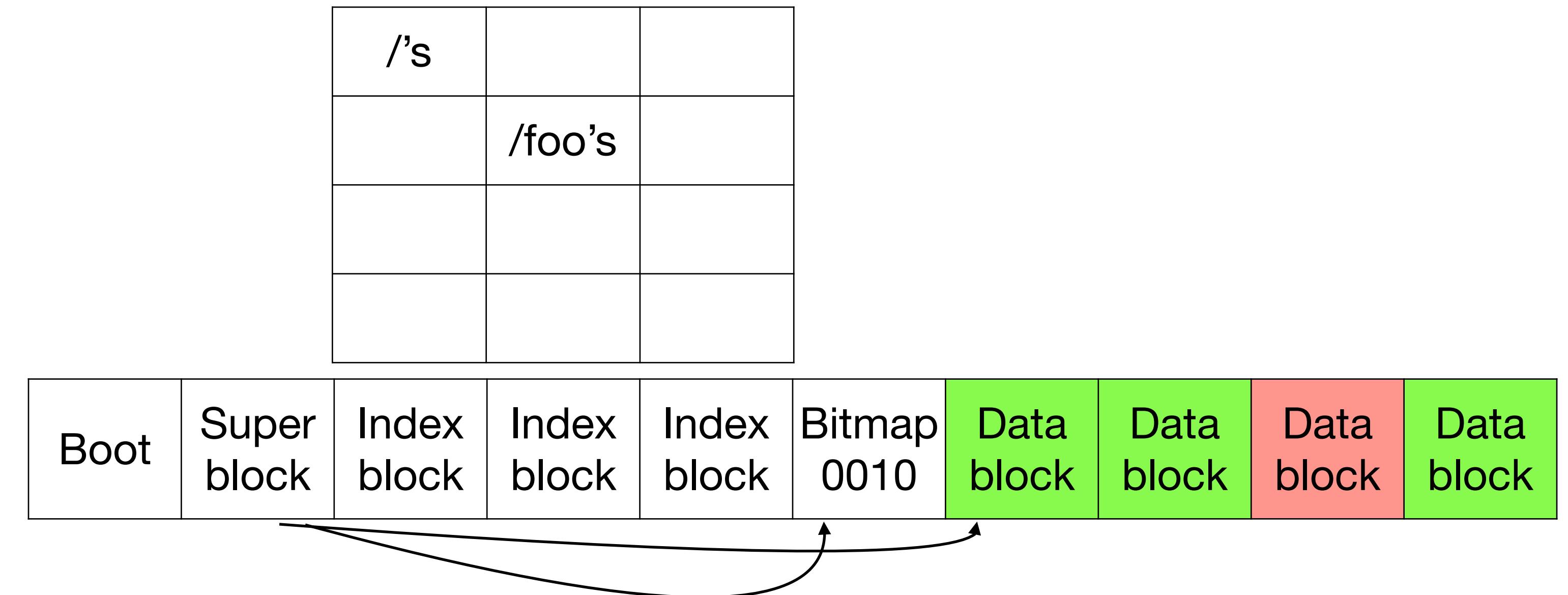
- Keep bitmap in another block

10010111



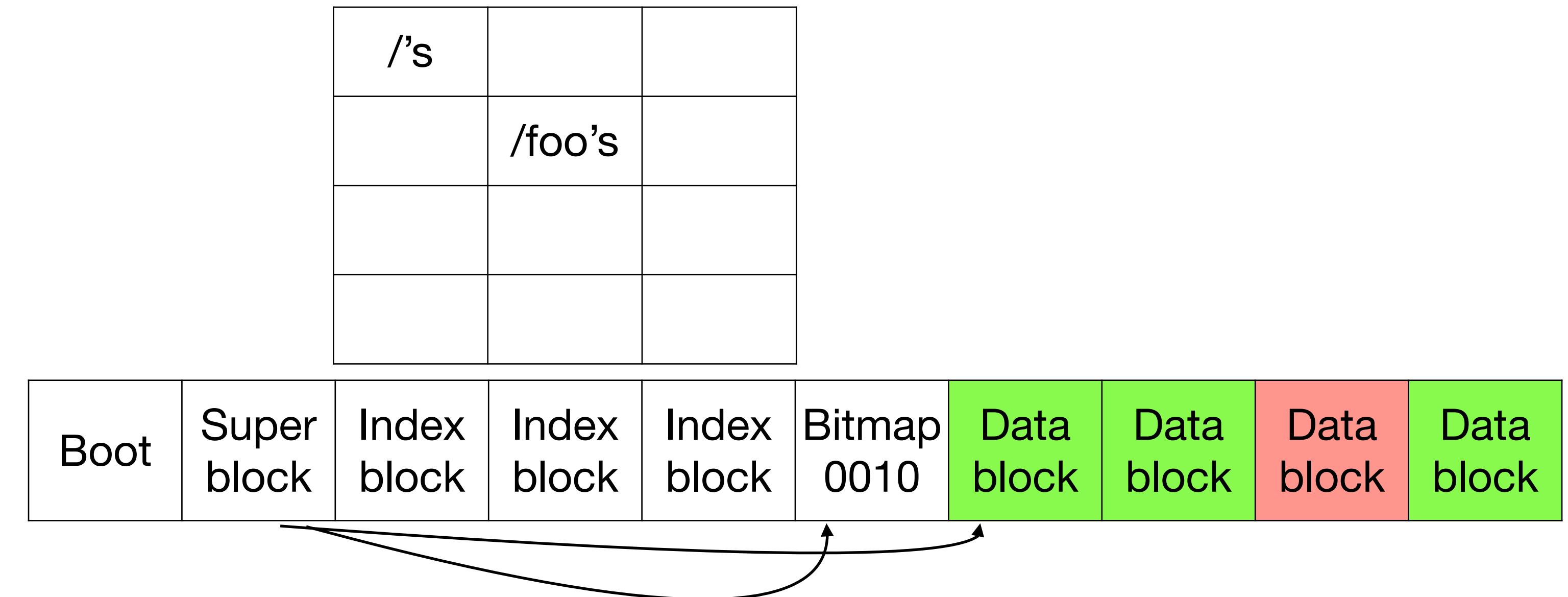
Putting it all together: xv6 FS organisation

- Data region contains actual file and directory data
- File system structure is maintained via nodes stored in index blocks



Putting it all together: xv6 FS organisation

- Data region contains actual file and directory data
- File system structure is maintained via nodes stored in index blocks
- Superblock contains file system metadata:
 - how many inodes are in system, etc



Writing a file

Example: /foo/bar



Inode = 1 “/“

Type = directory
Size
2

Block 2

File/directory name	Inode number
.	1
foo	8

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Inode = 8 “/foo“

Type = directory
Size
16

Block 16

File/directory name	Inode number
.	8
..	1

```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar

Inode = 1 “/“

Type = directory
Size
2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo“

Type = directory
Size
16

Block 16

File/directory name	Inode number
.	8
..	1

Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar

Inode = 1 “/“

Type = directory	
Size	
1	2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo“

Type = directory	
Size	
16	

Block 16

File/directory name	Inode number
.	8
..	1

Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar

Inode = 1 “/“

Type = directory	
Size	
1	2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo“

Type = directory	
Size	
16	

Block 16

File/directory name	Inode number
.	8
..	1

Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

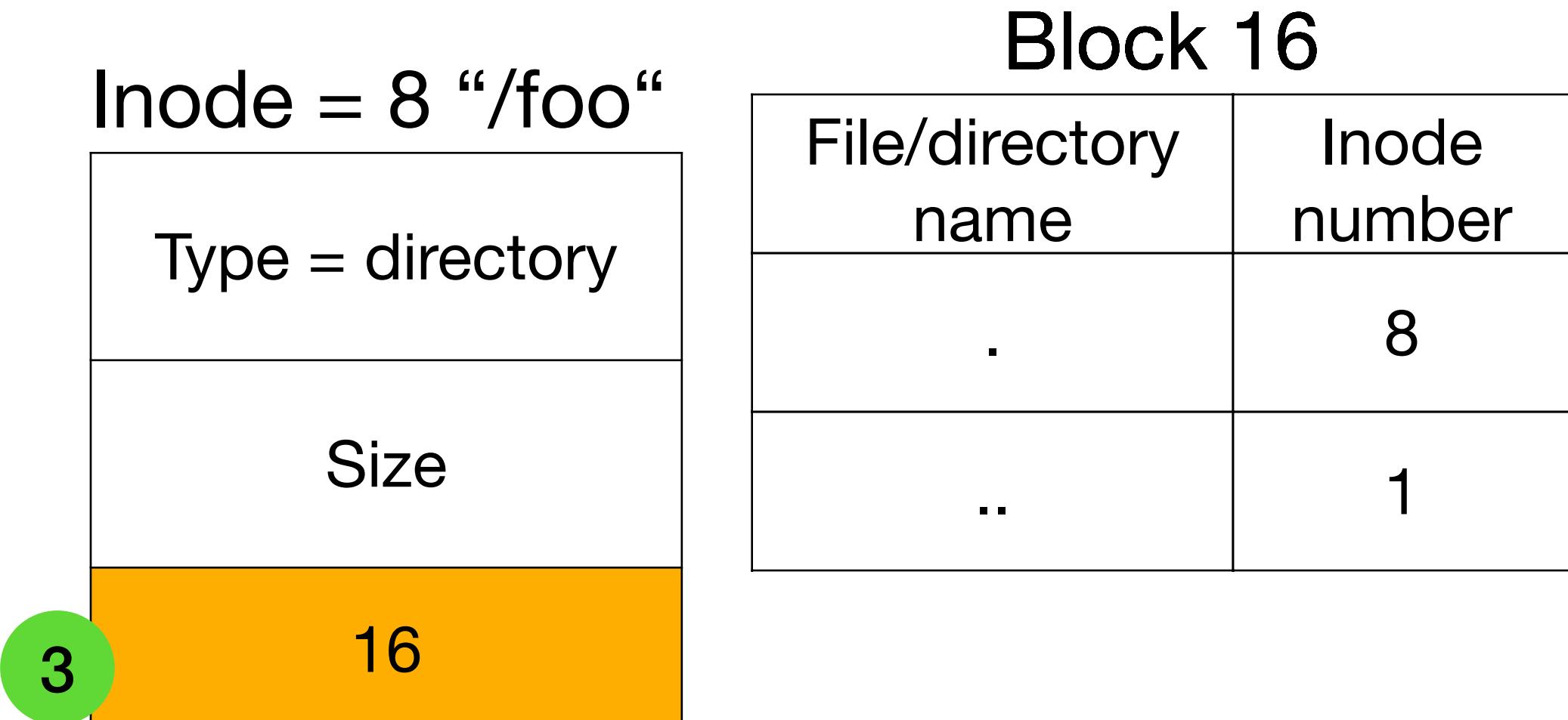
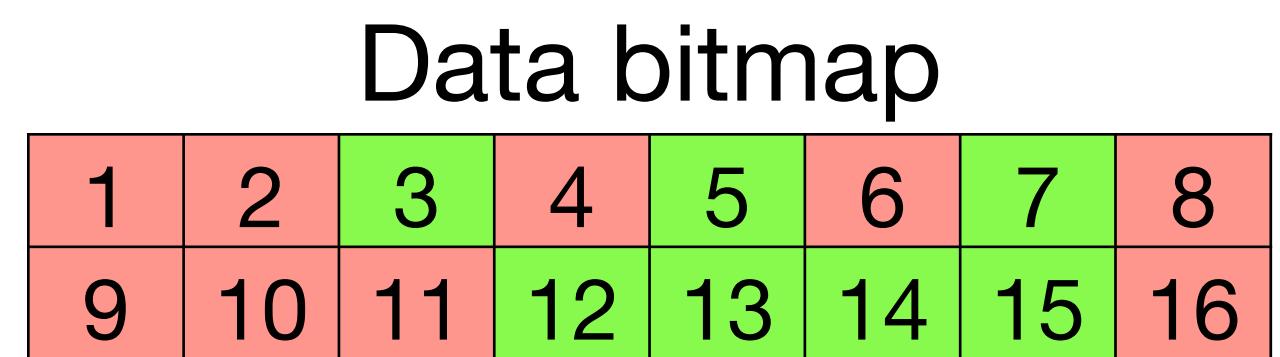
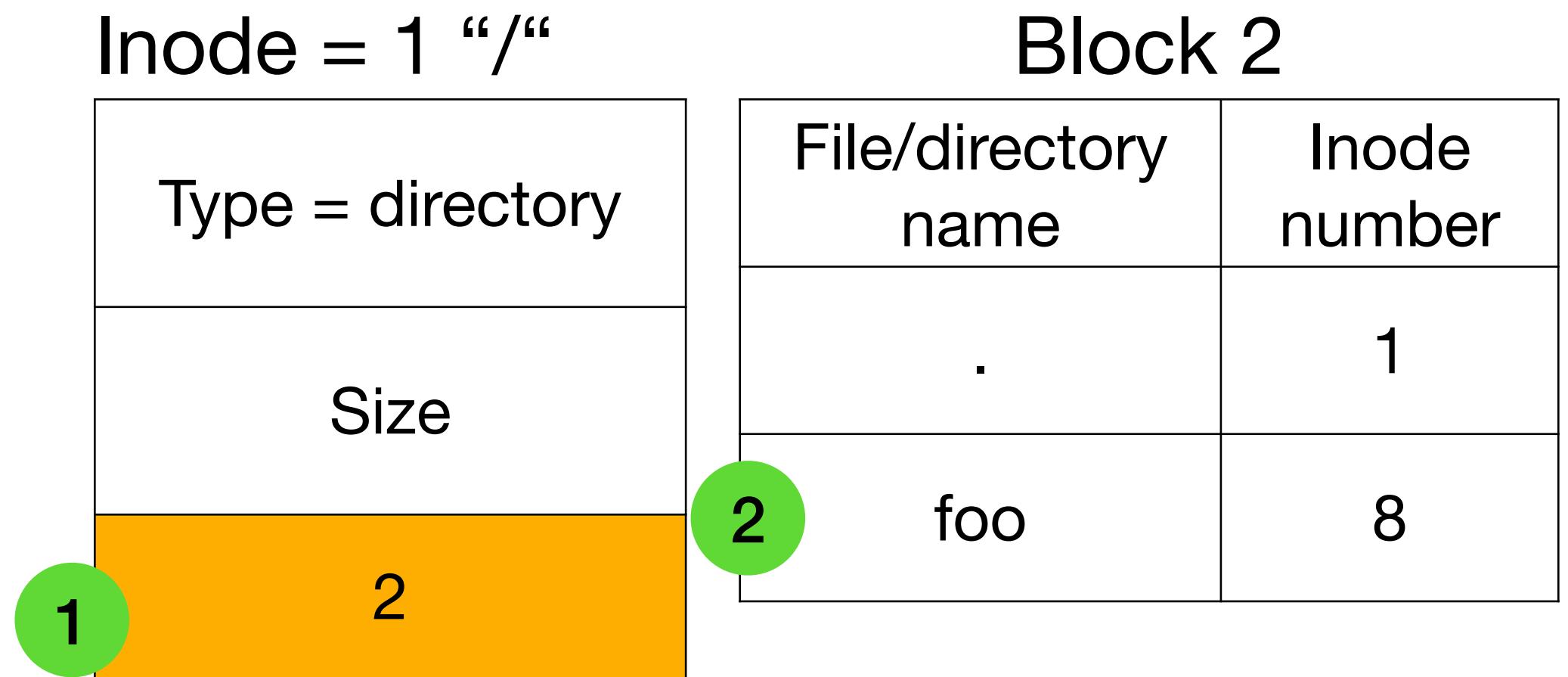
```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar



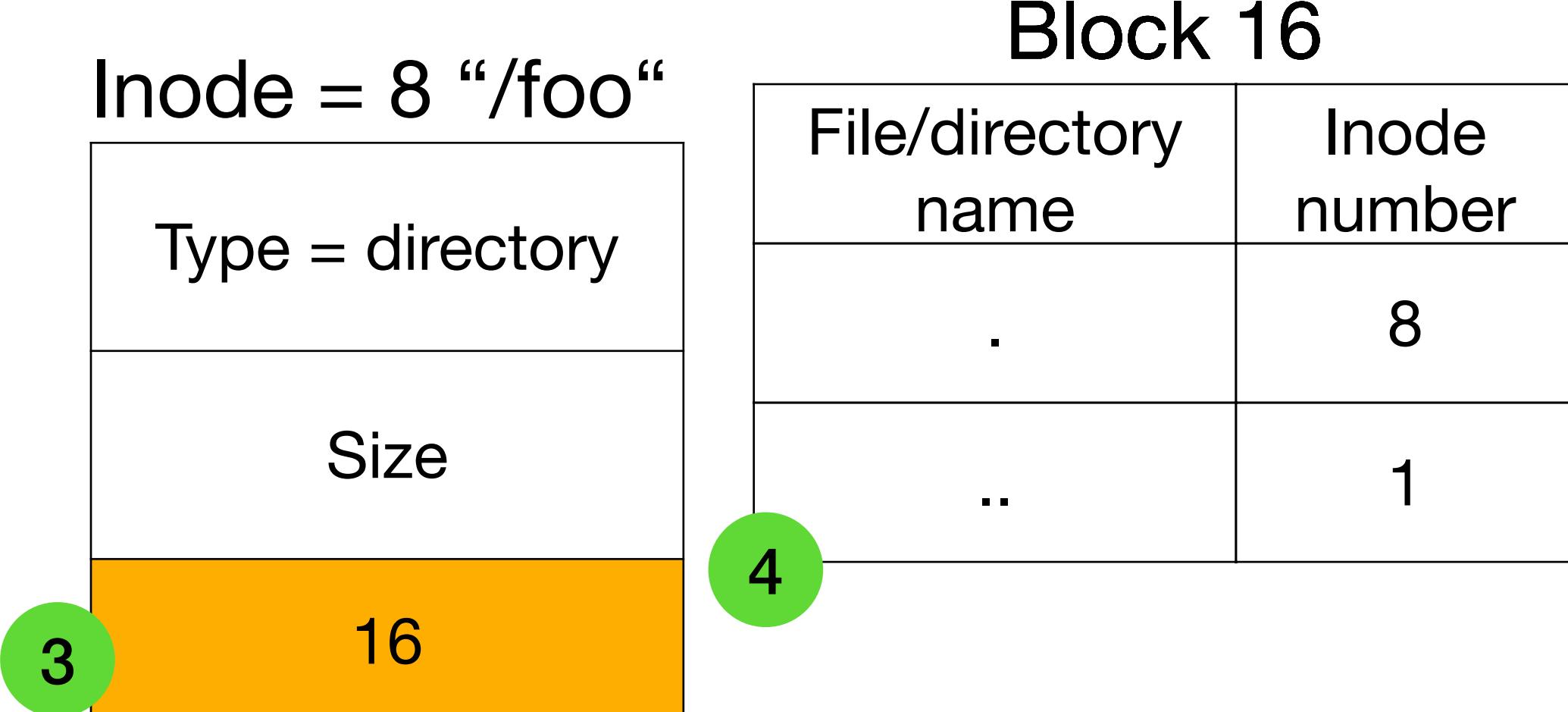
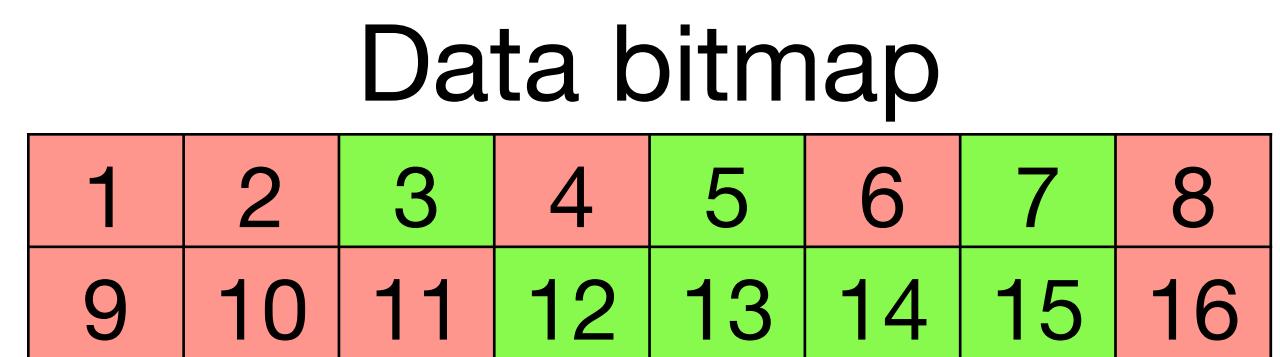
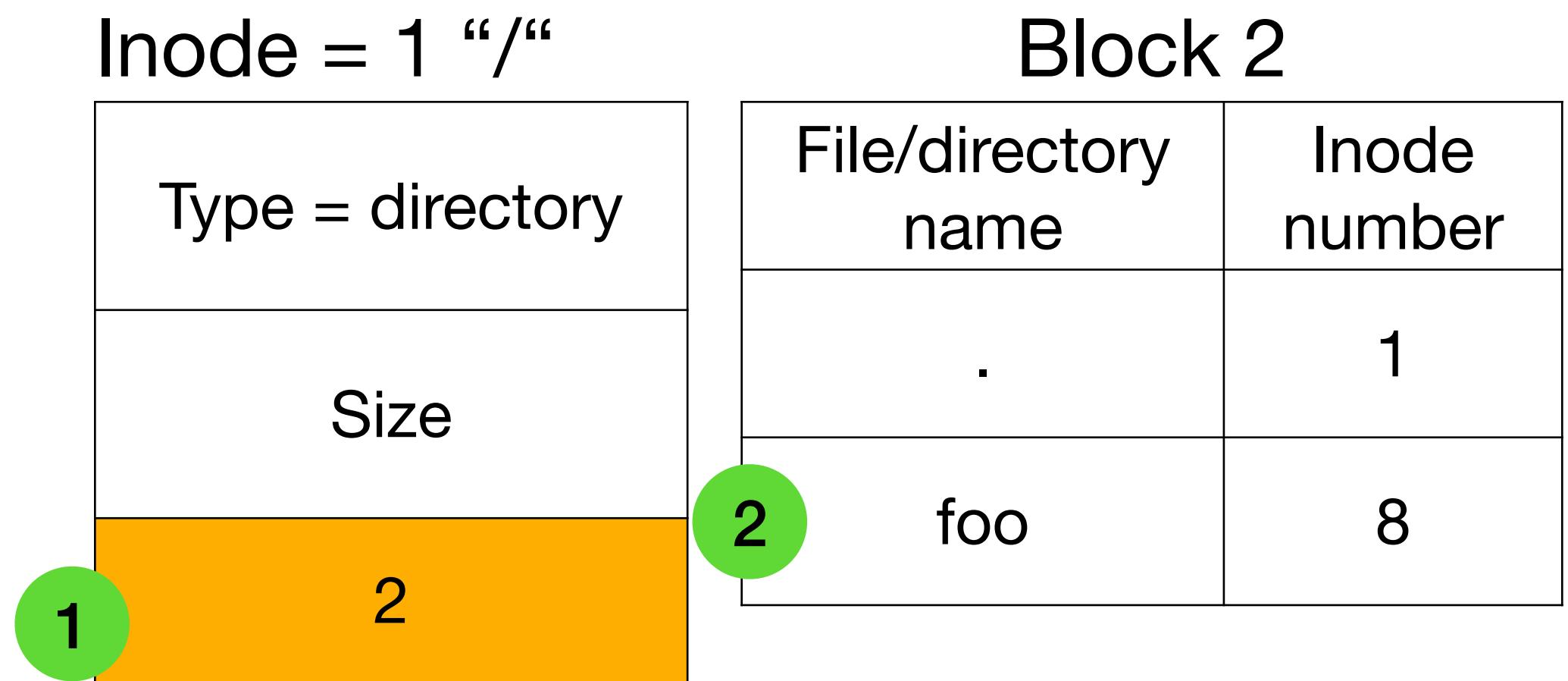
```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar



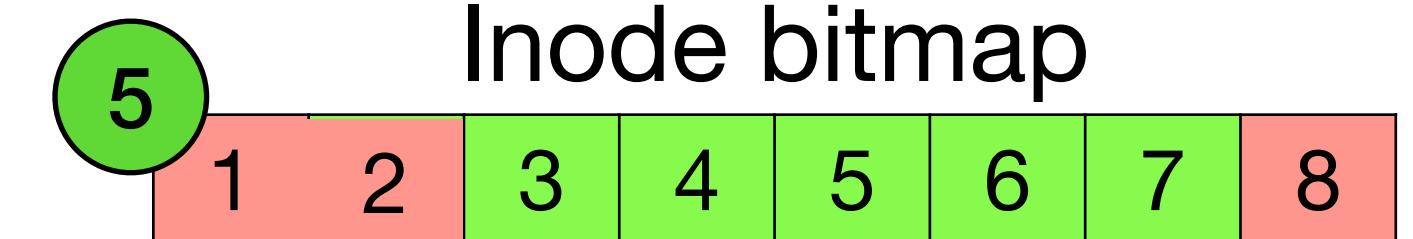
```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar



Inode = 1 “/“

Type = directory	
Size	
2	2

Block 2

File/directory name	Inode number
.	1
foo	8

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Inode = 8 “/foo“

Type = directory	
Size	
16	16

Block 16

File/directory name	Inode number
.	8
..	1

Writing a file

Example: /foo/bar

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

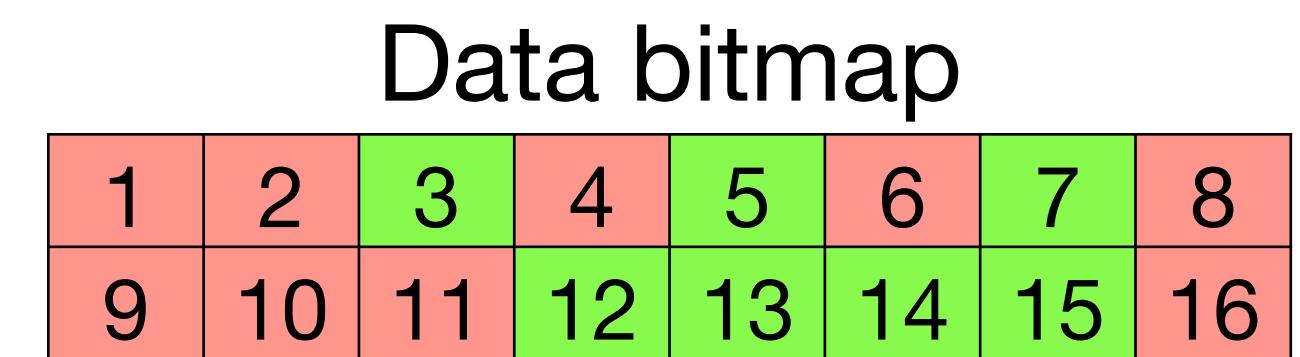


1	Inode = 1 “/“
	Type = directory
	Size
2	2

Block 2	
File/directory name	Inode number
.	1
foo	8

3	Inode = 8 “/foo“
	Type = directory
	Size
4	16

Block 16	
File/directory name	Inode number
.	8
..	1



Writing a file

Example: /foo/bar

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time

```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Inode = 1 “/”

Type = directory	2
Size	
1	2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”

Type = directory	4
Size	
3	16

Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Data bitmap

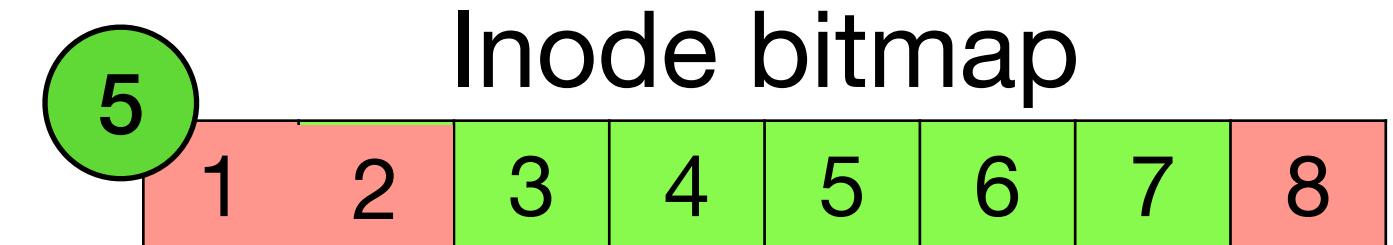
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Writing a file

Example: /foo/bar

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time

```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Inode = 1 “/“

Type = directory	2
Size	
1	2

Block 2

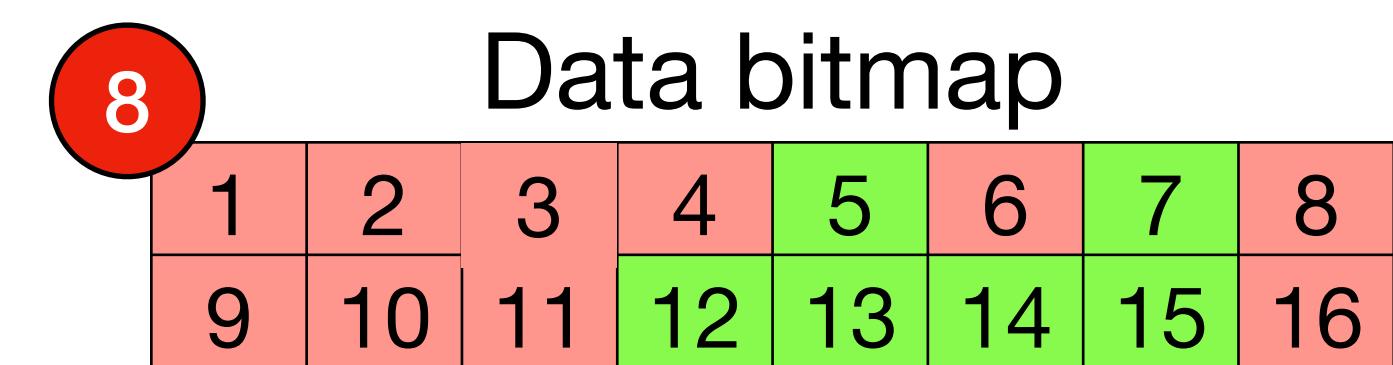
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo“

Type = directory	4
Size	
3	16

Block 16

File/directory name	Inode number
.	8
..	1
bar	2



Writing a file

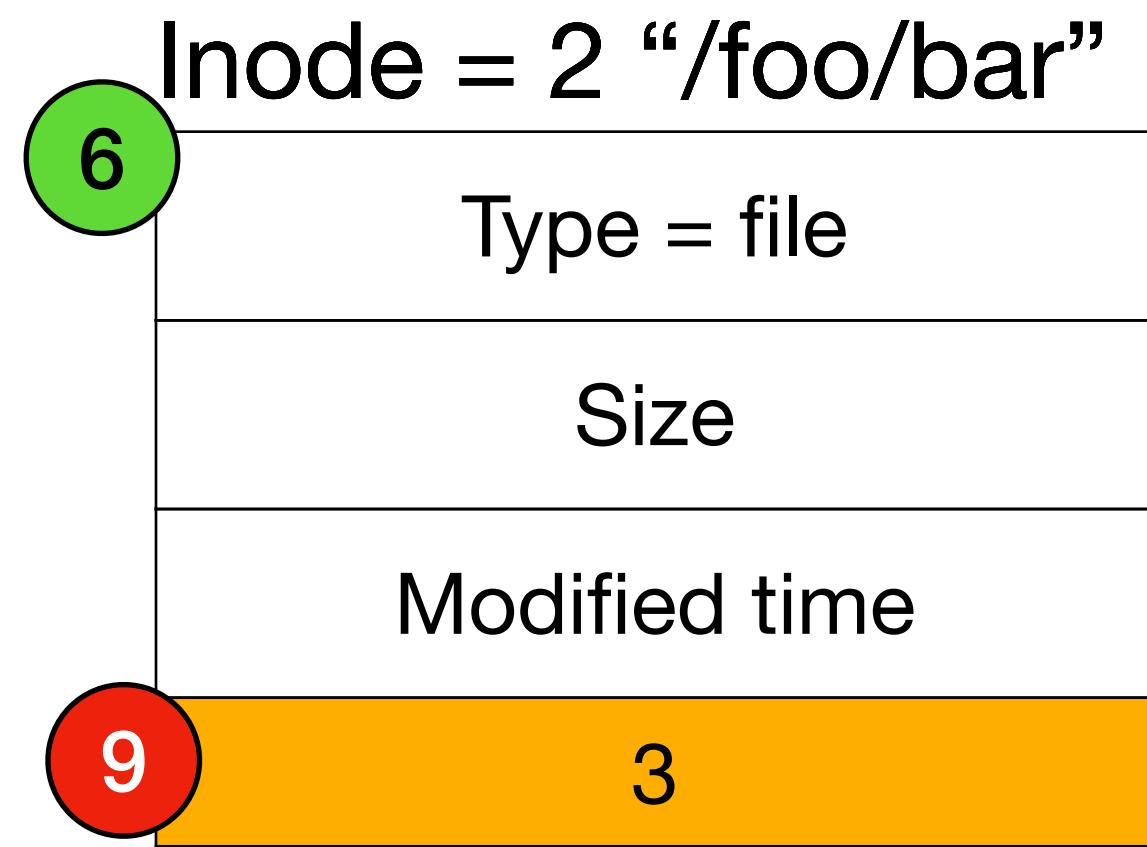
Example: /foo/bar

Inode = 1 “/”
Type = directory
Size
2

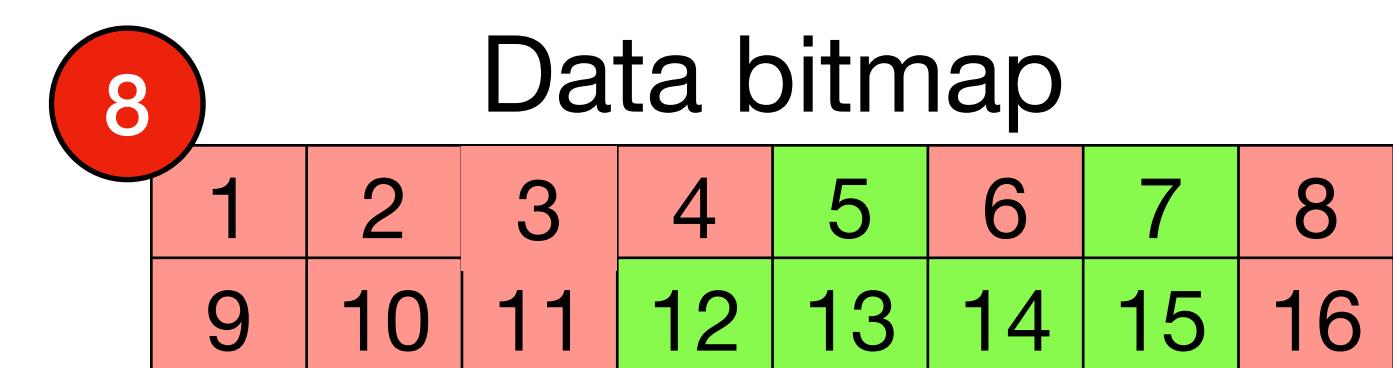
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”
Type = directory
Size
16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```



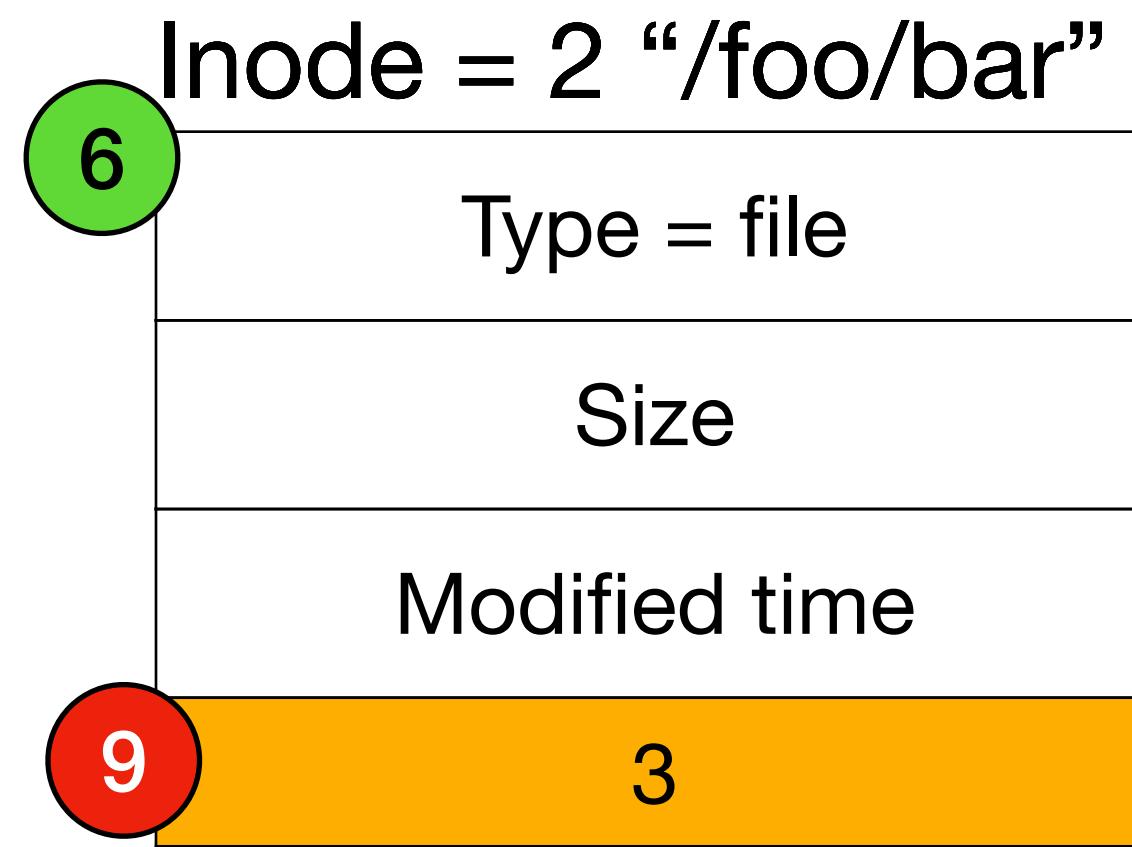
Writing a file

Example: /foo/bar

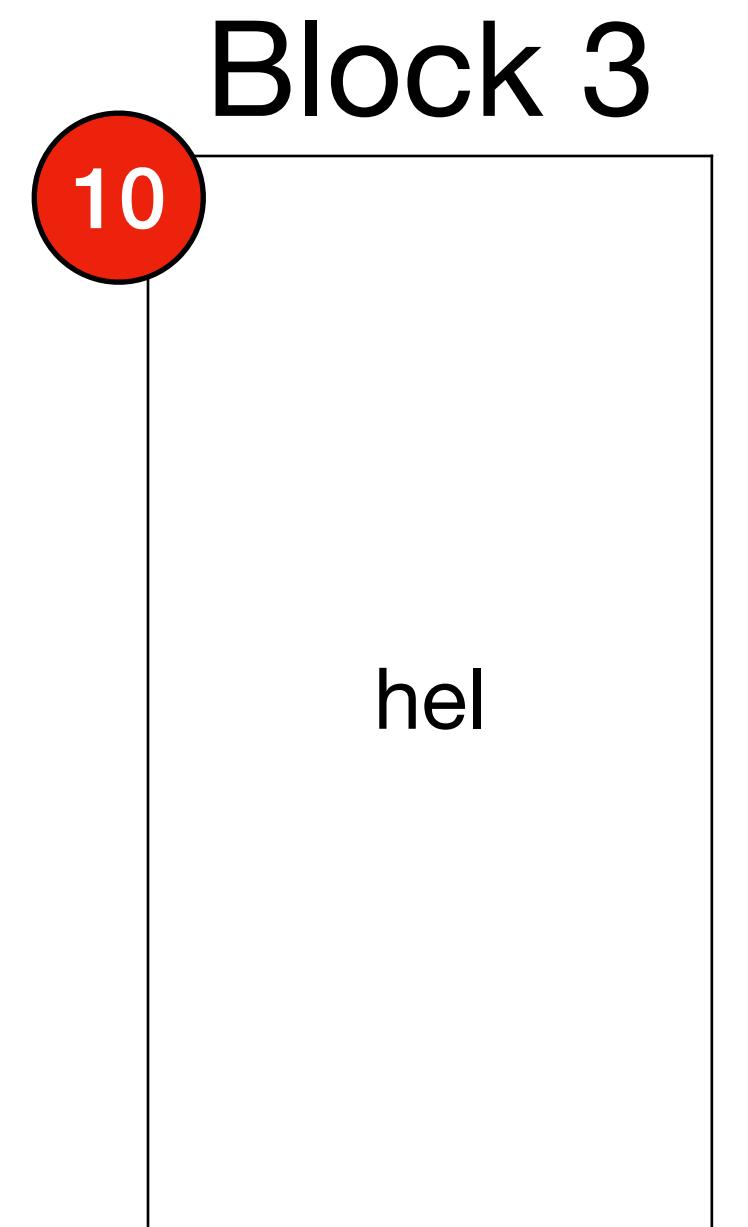
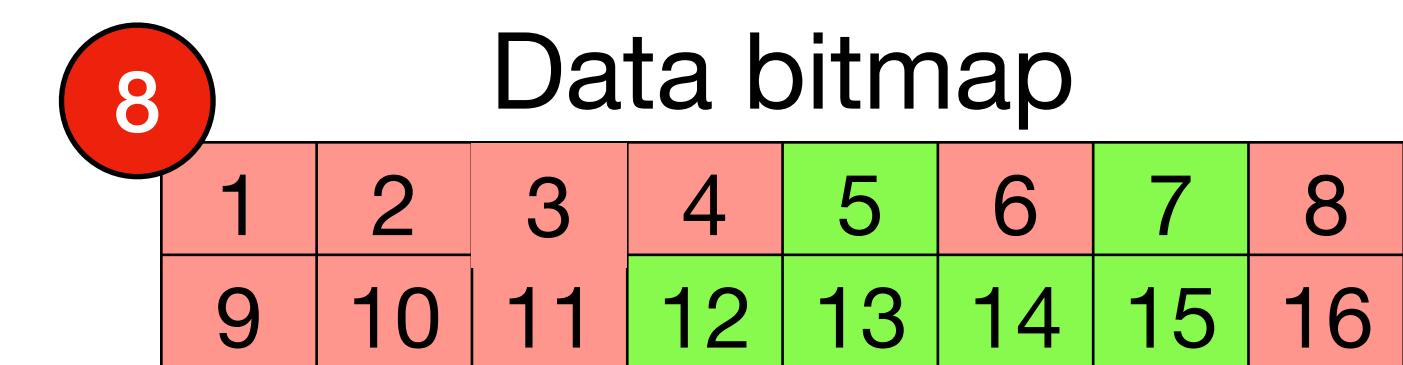
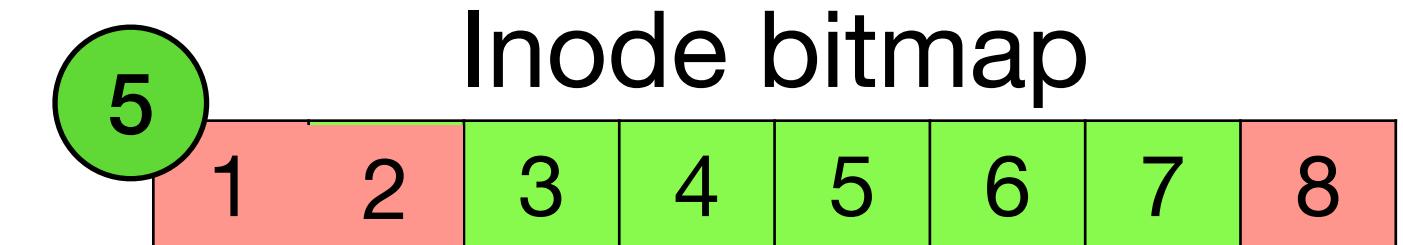
Inode = 1 “/”	
Type = directory	
Size	
2	

Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

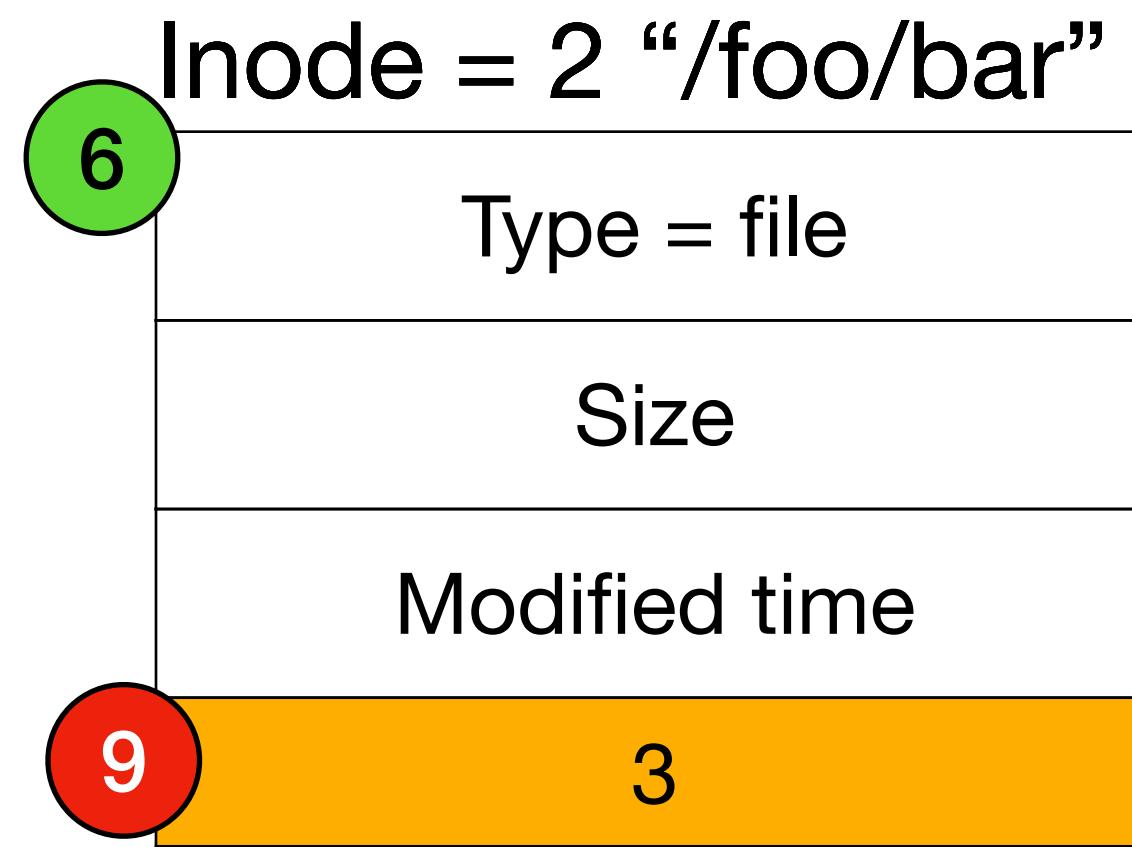
Writing a file

Example: /foo/bar

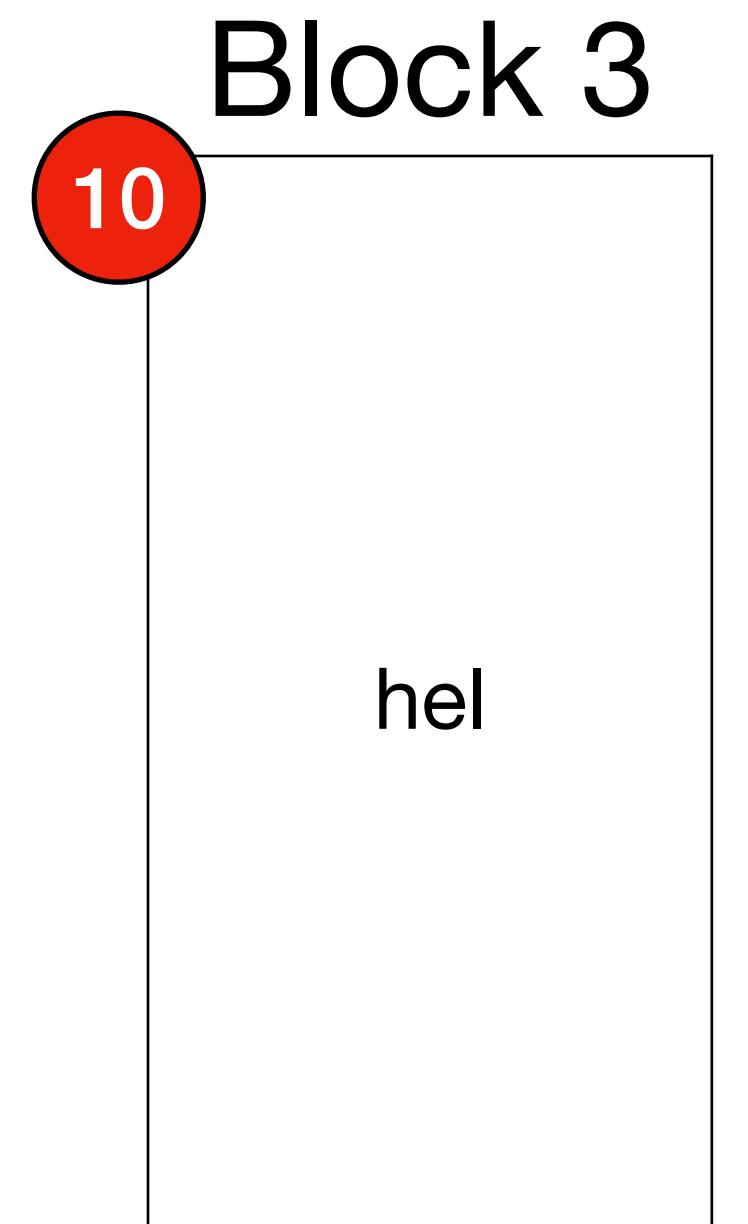
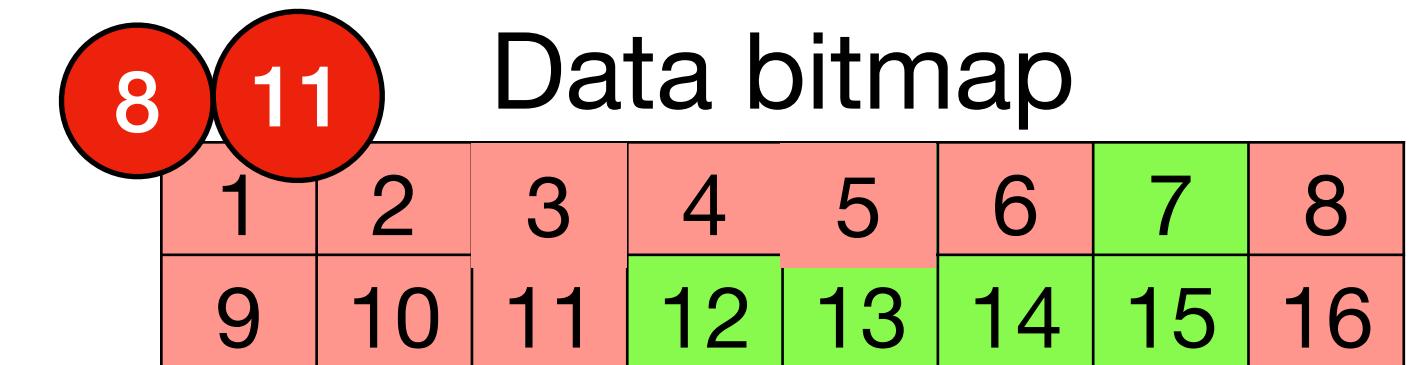
Inode = 1 “/”	
Type = directory	
Size	
2	

Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

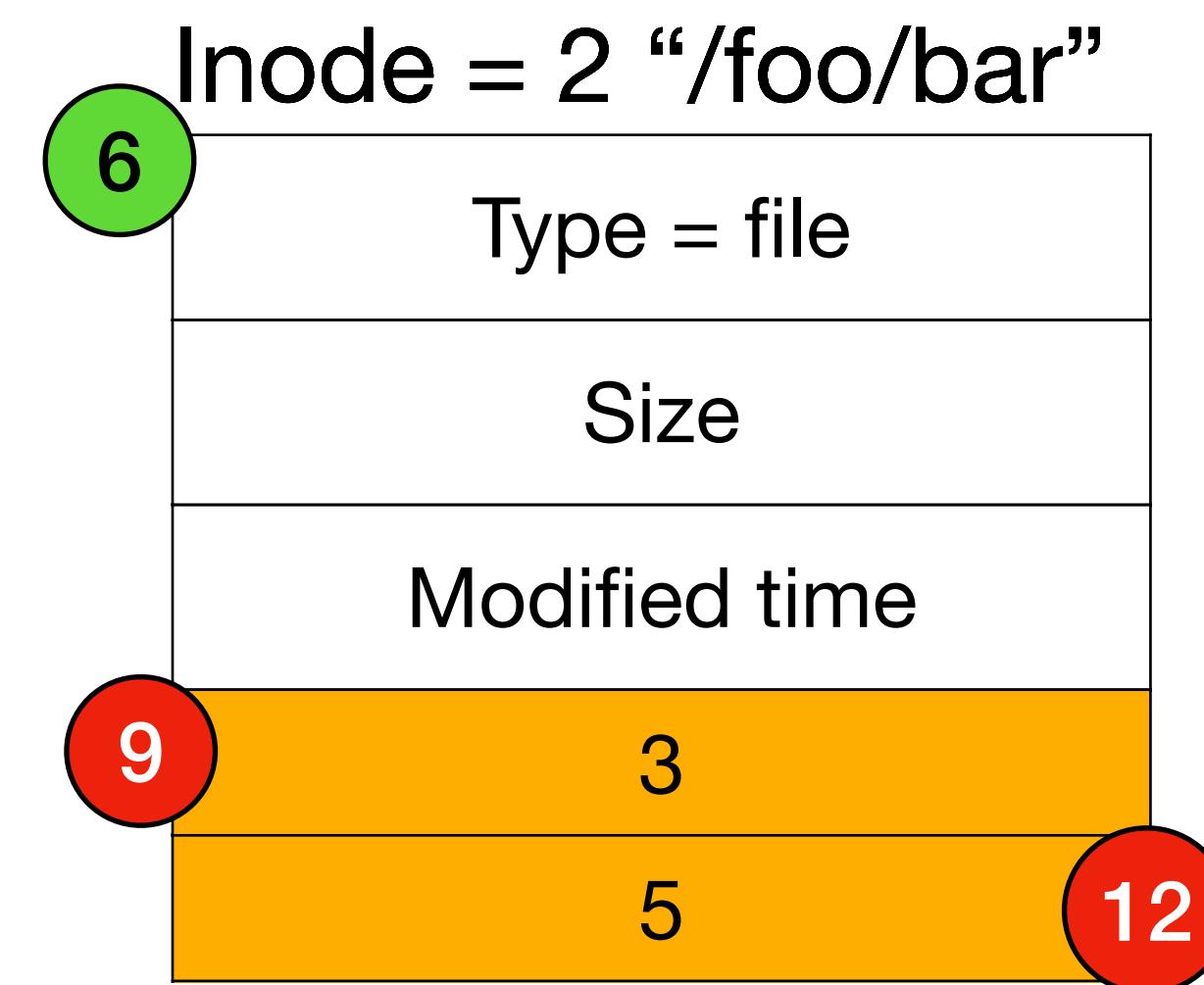
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

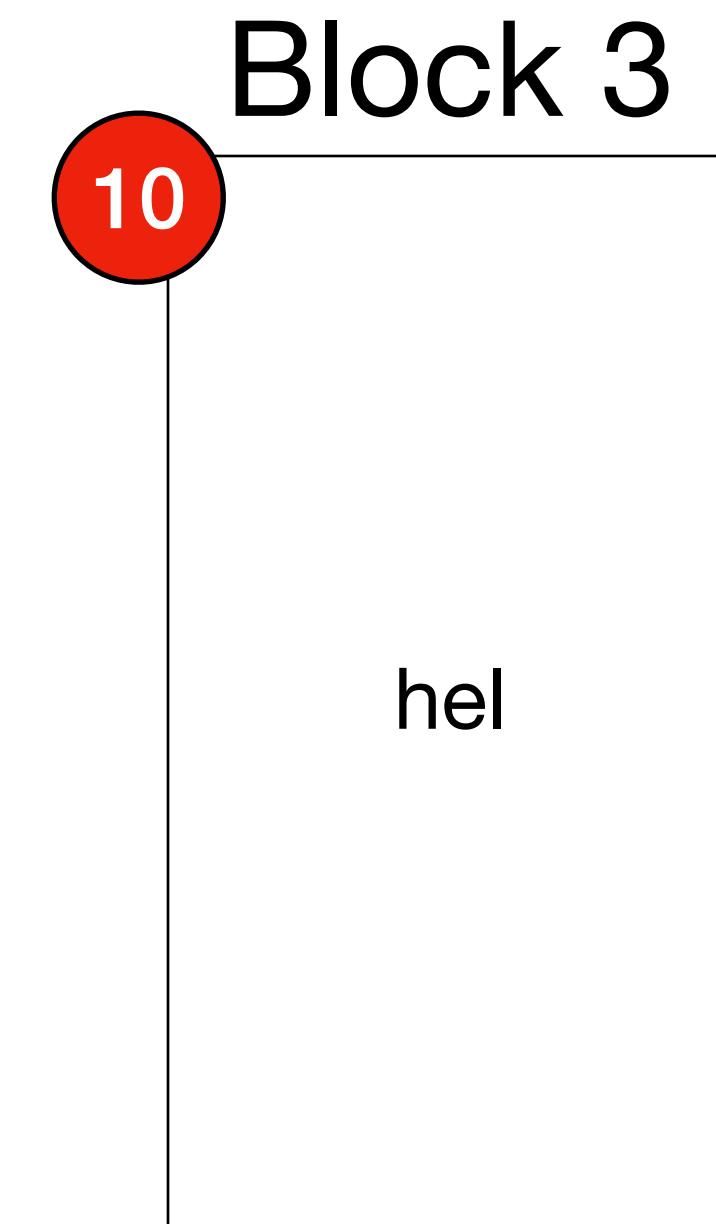
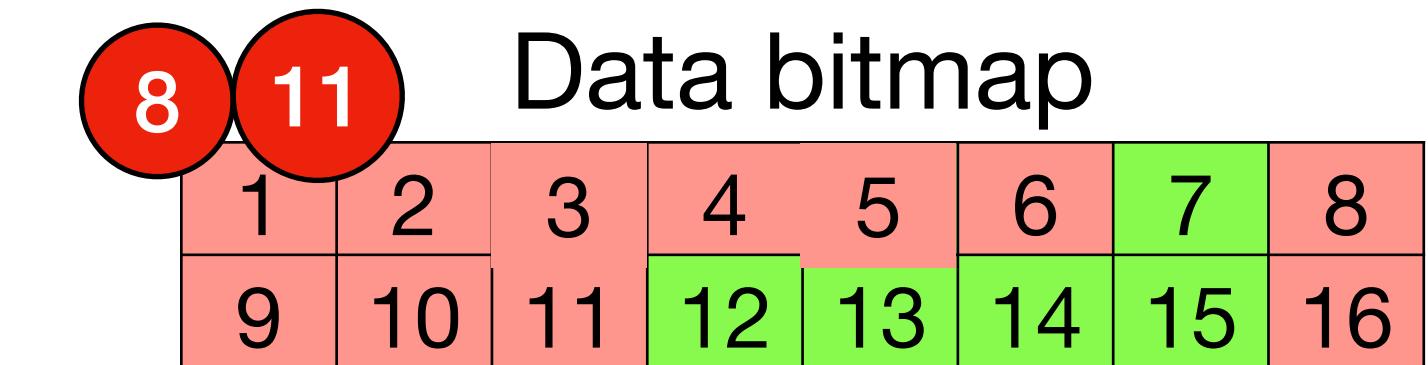
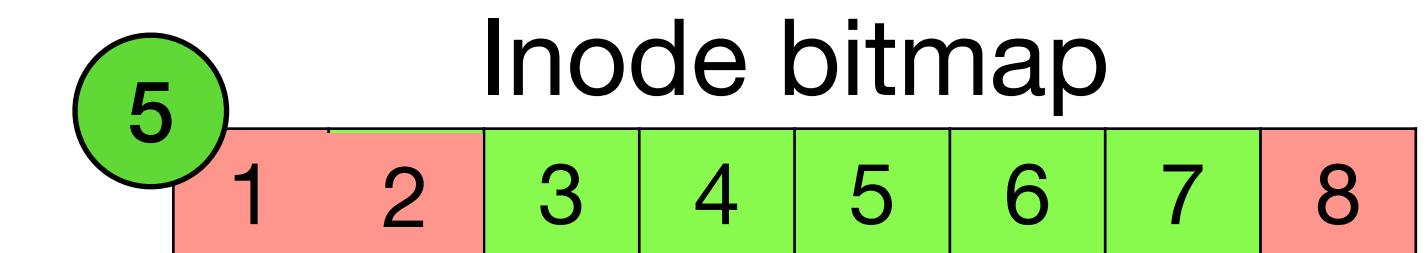
Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```



Writing a file

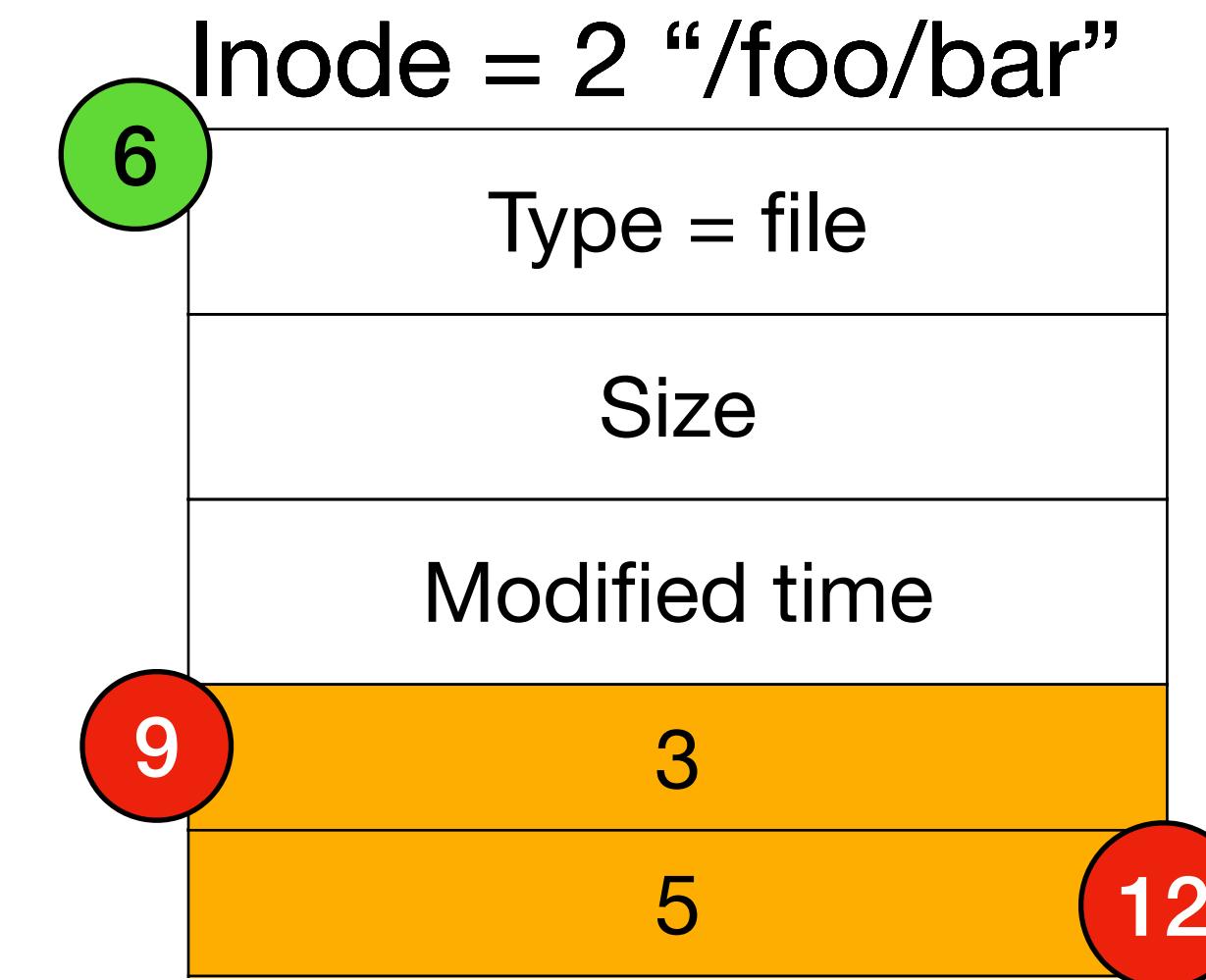
Example: /foo/bar

Inode = 1 “/”
Type = directory
Size 2

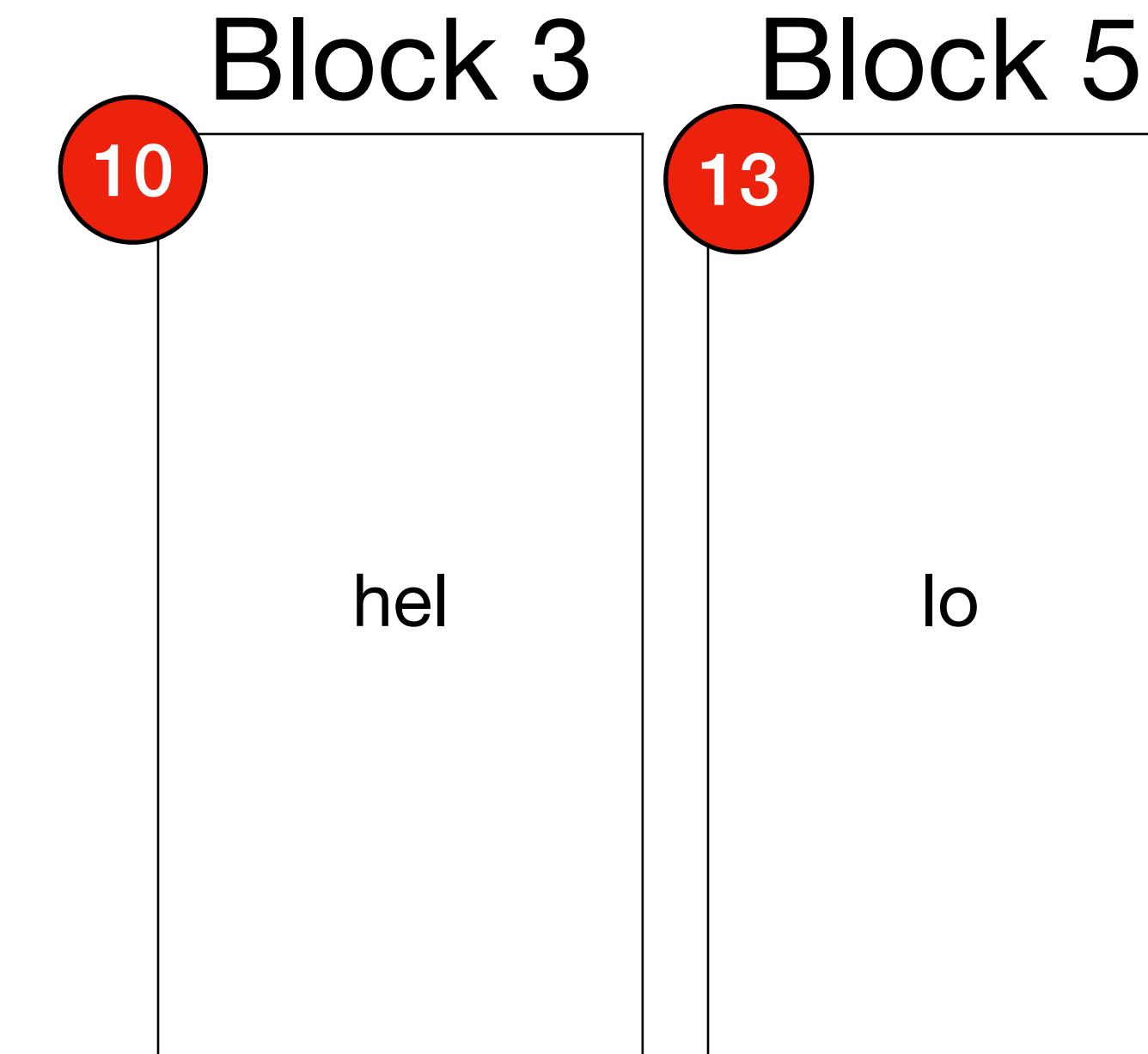
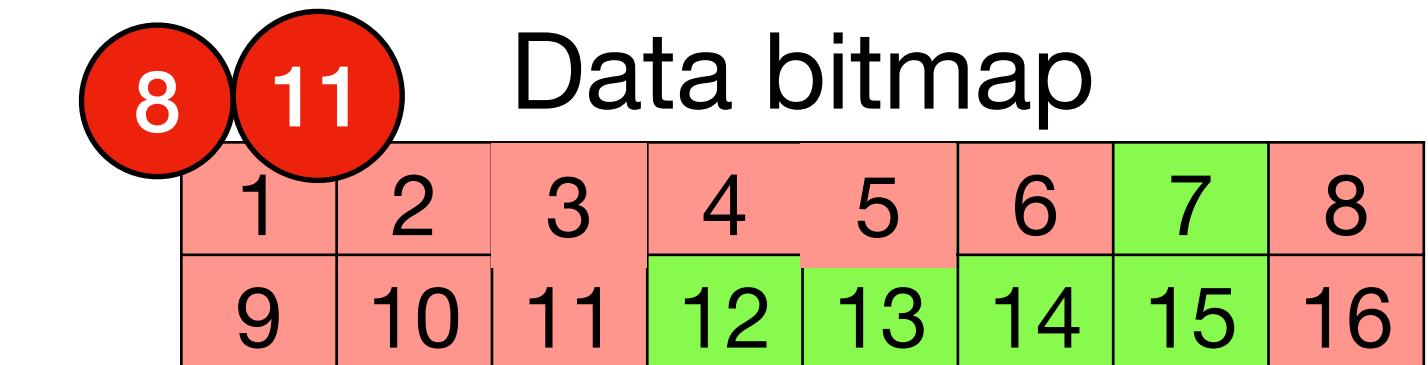
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”
Type = directory
Size 16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

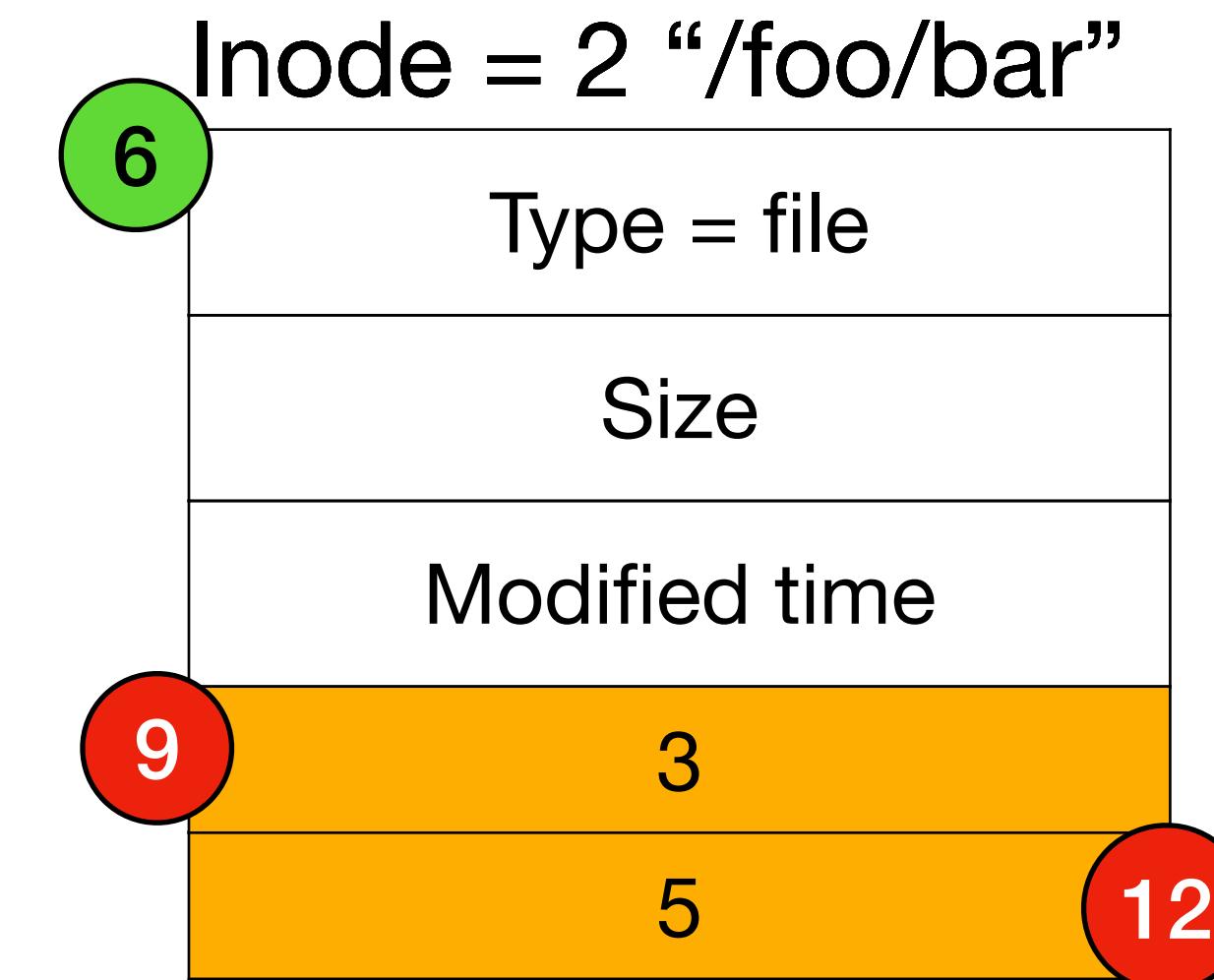
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

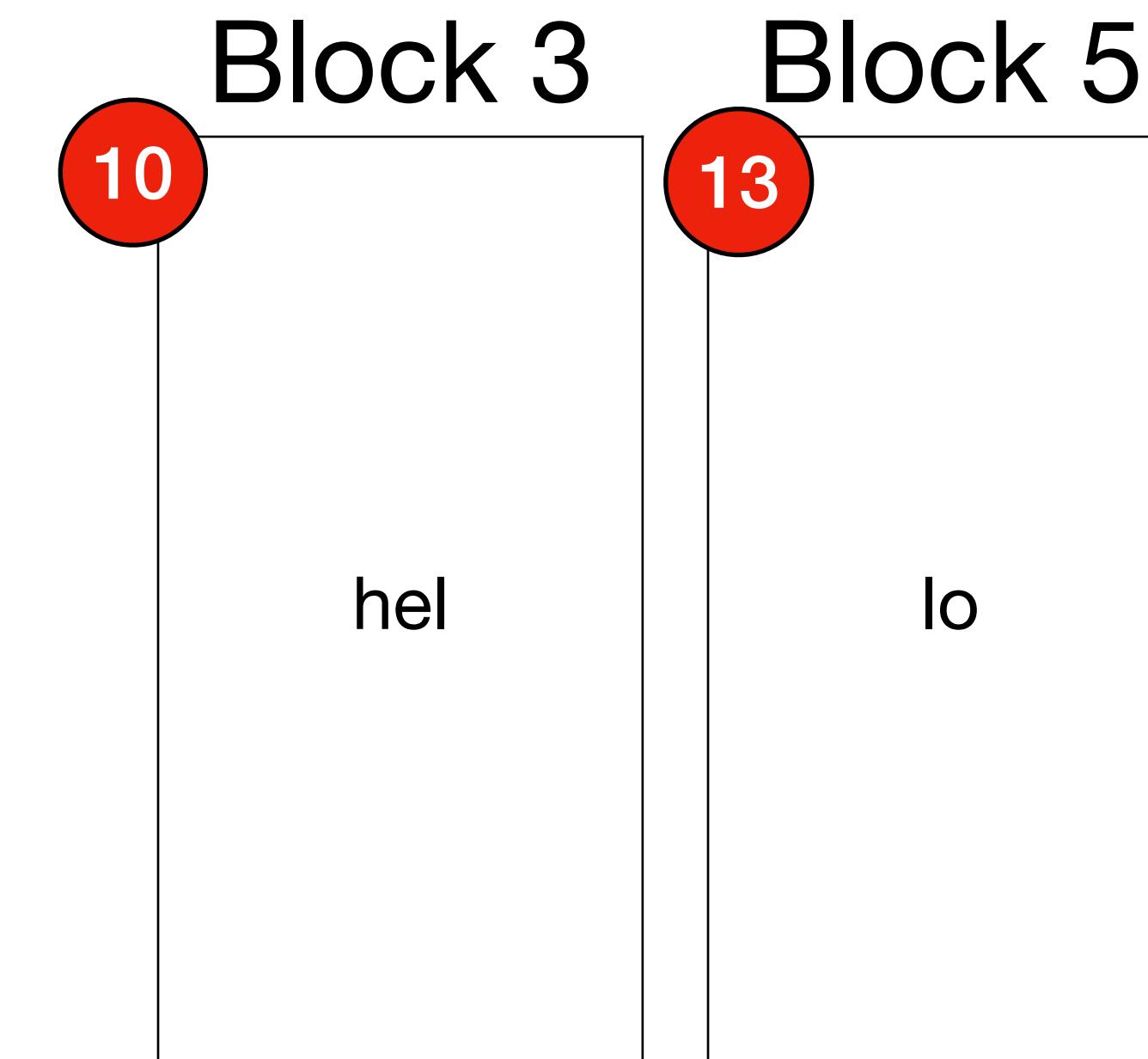
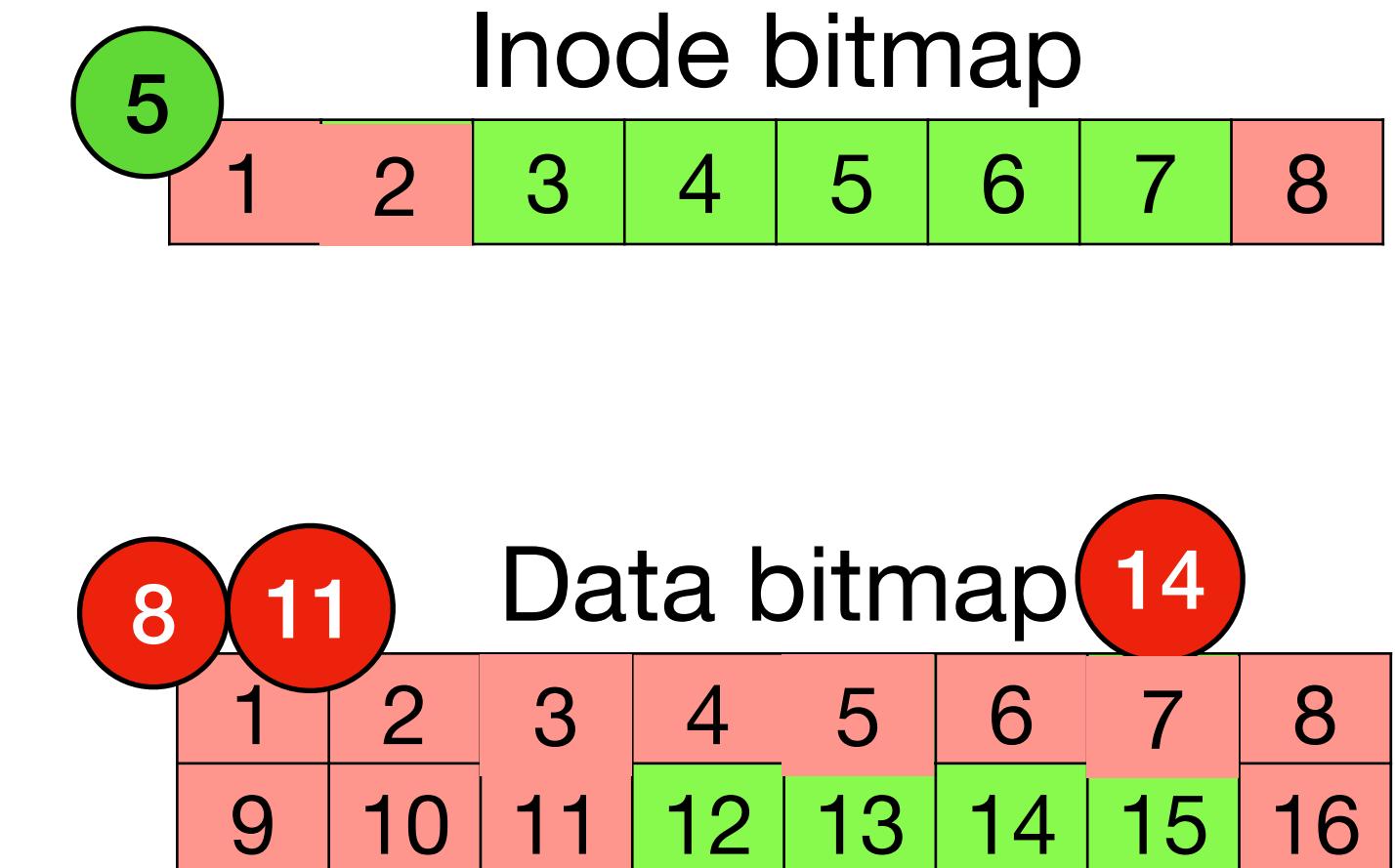
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```



Writing a file

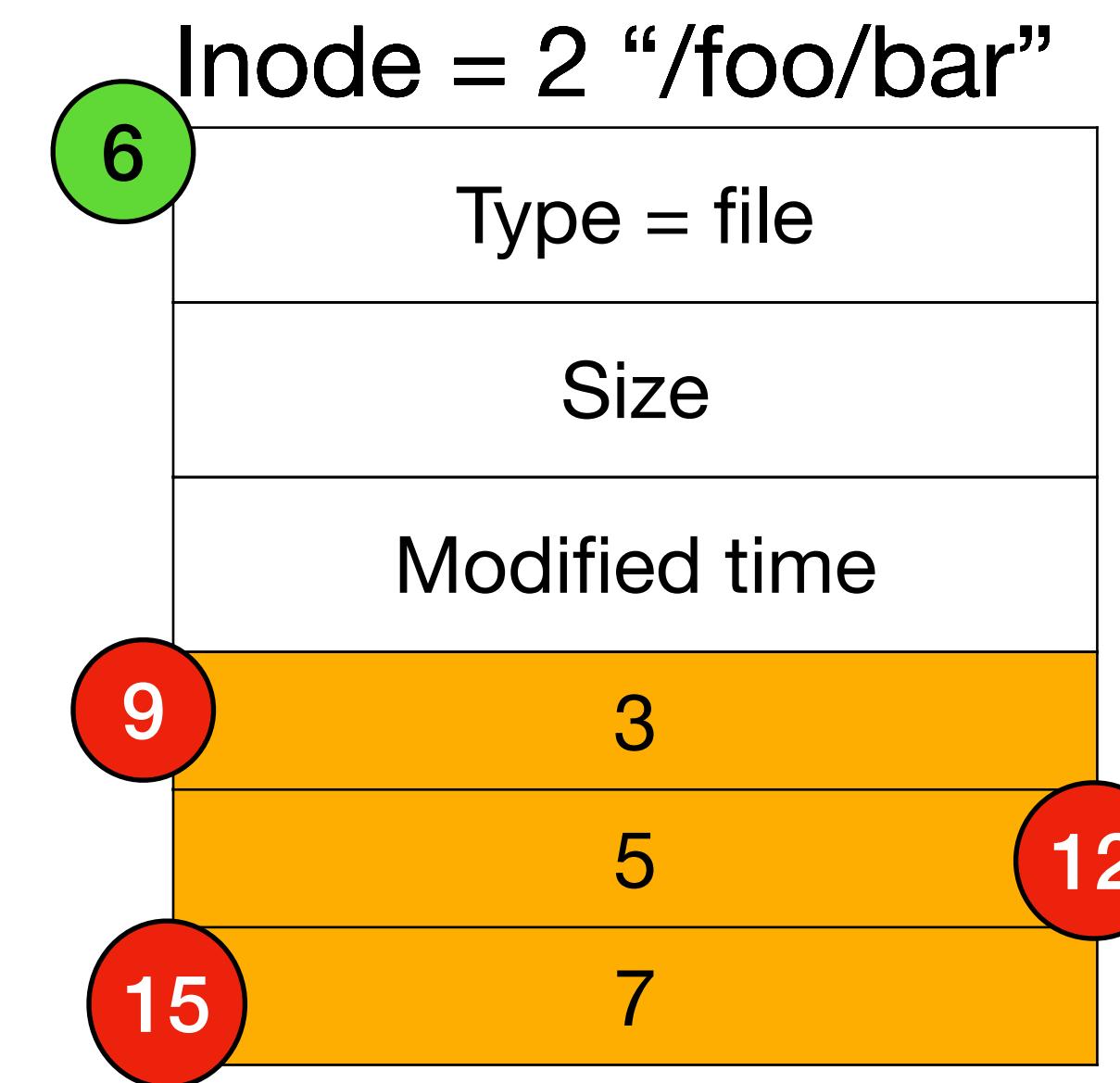
Example: /foo/bar

Inode = 1 “/”
Type = directory
Size
2

Block 2	
File/directory name	Inode number
.	1
foo	8

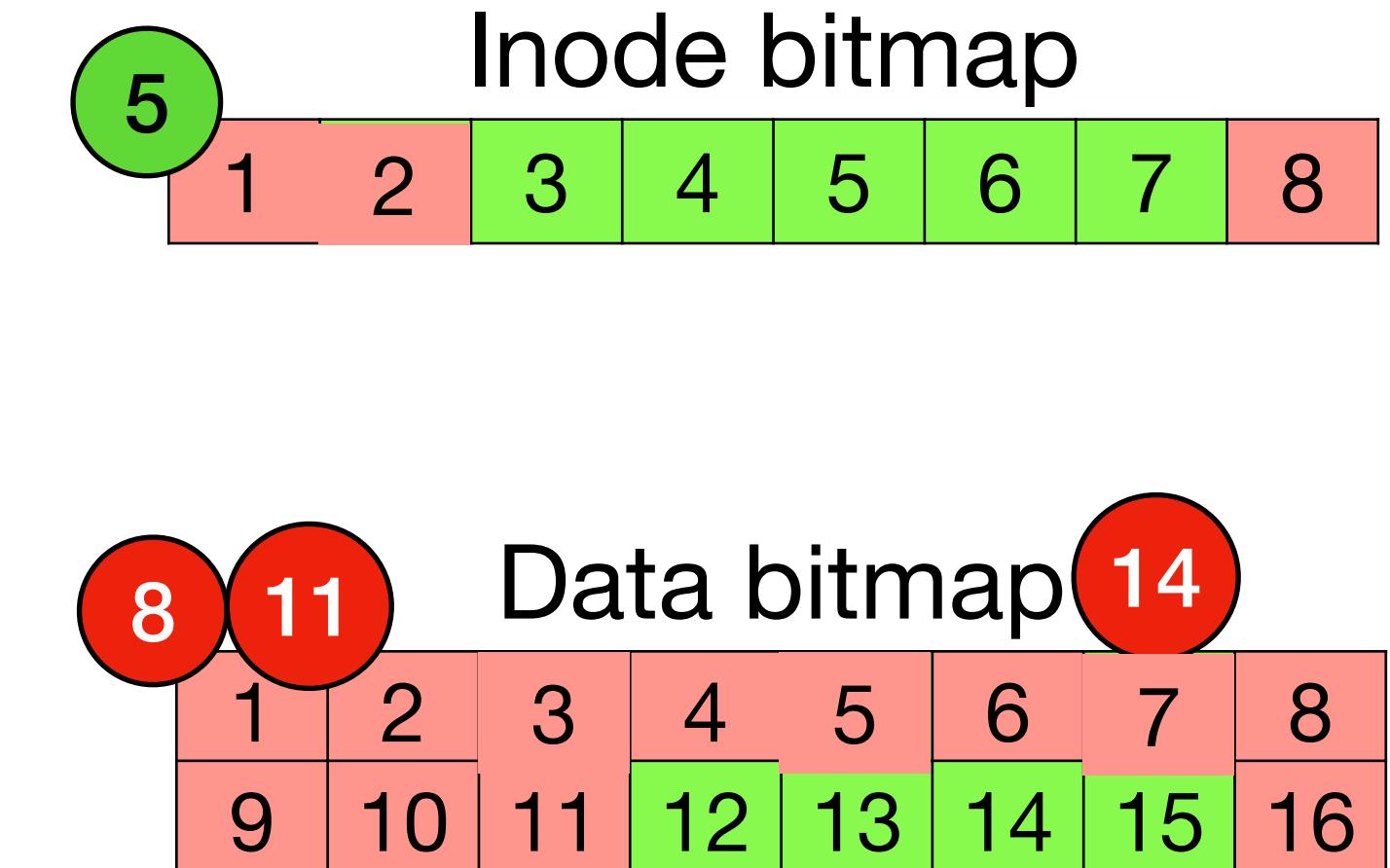
Inode = 8 “/foo”
Type = directory
Size
16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
  
```



Writing a file

Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

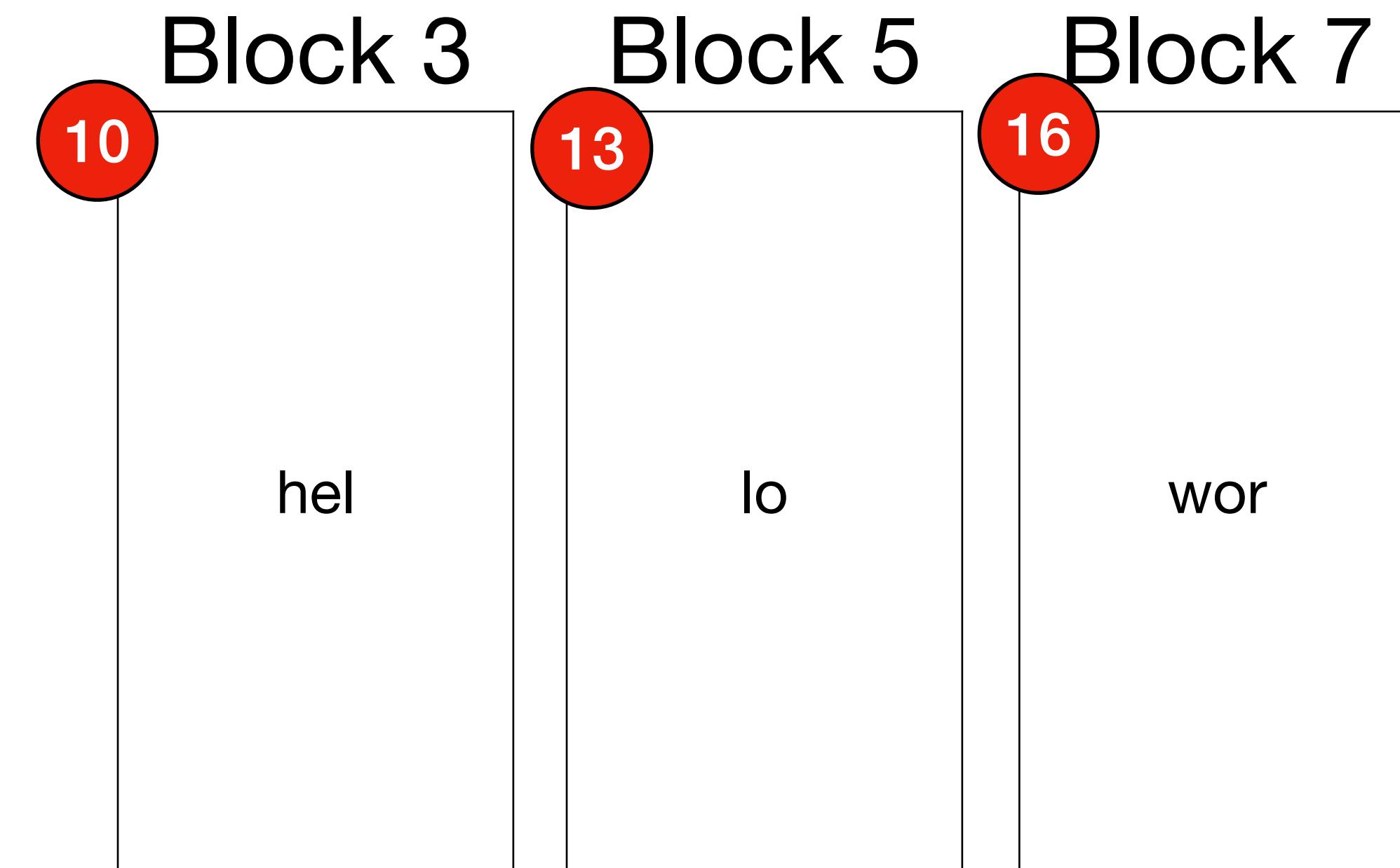
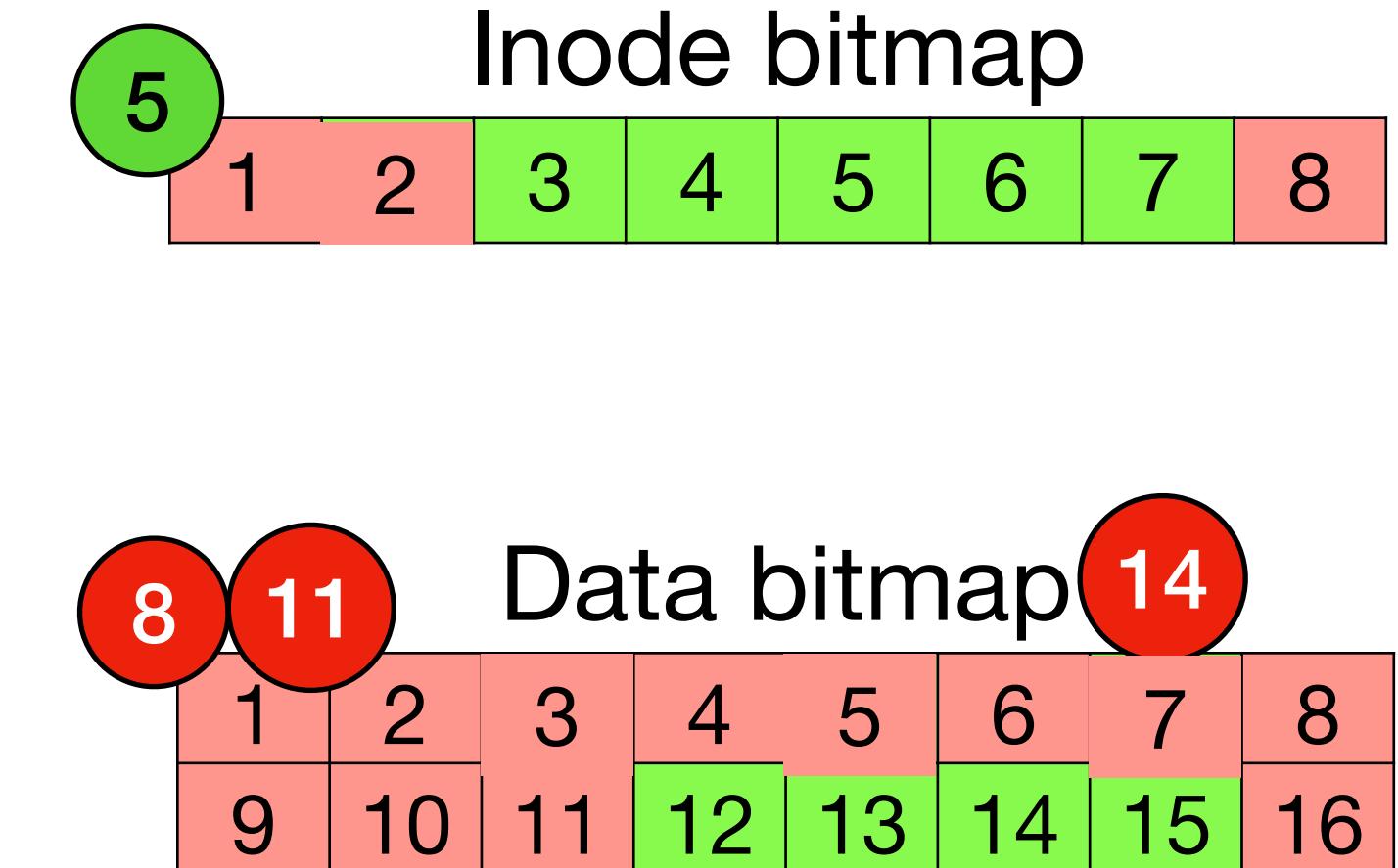
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time
9	3
15	7
12	

```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



QUESTION

Writing a file

Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

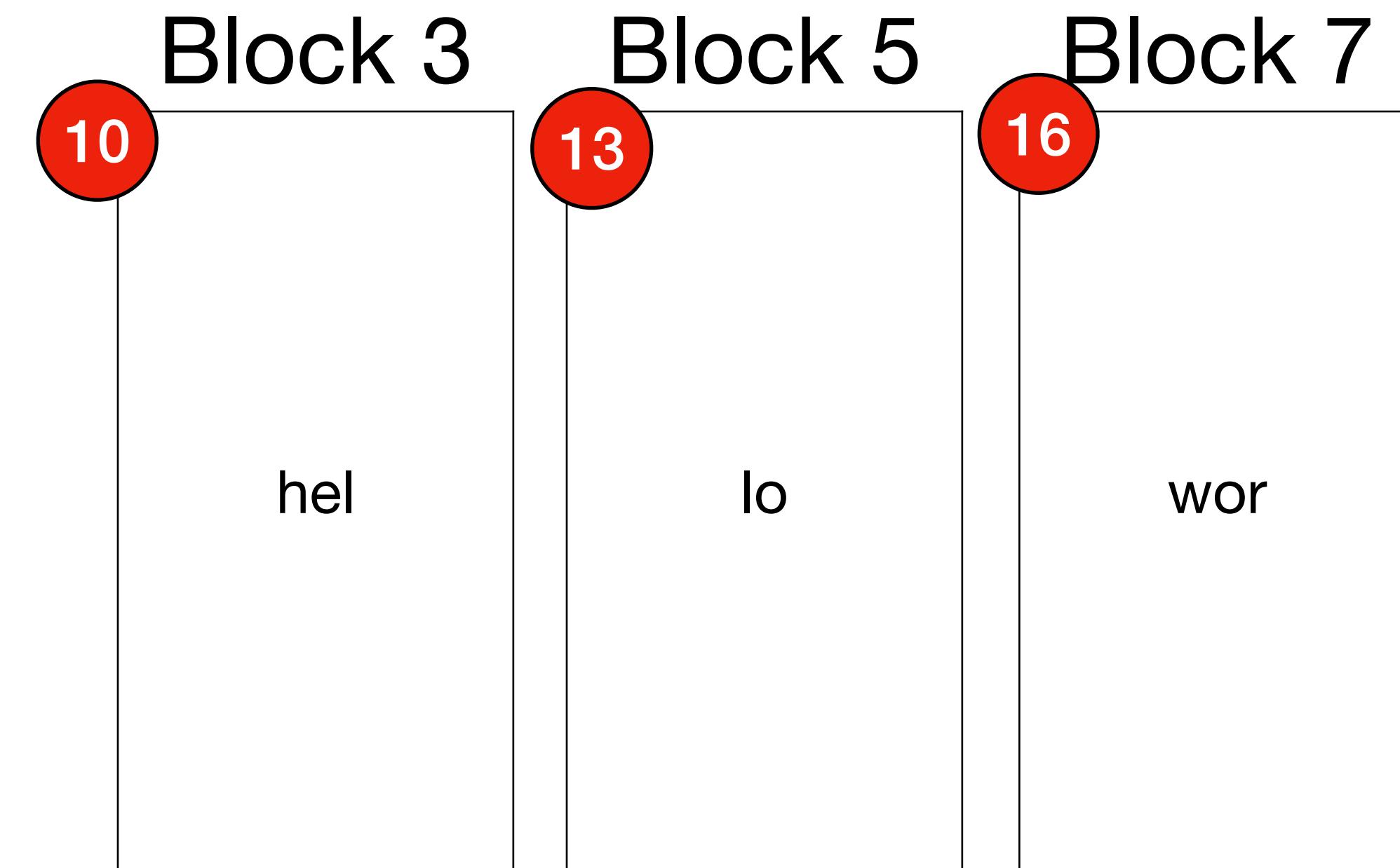
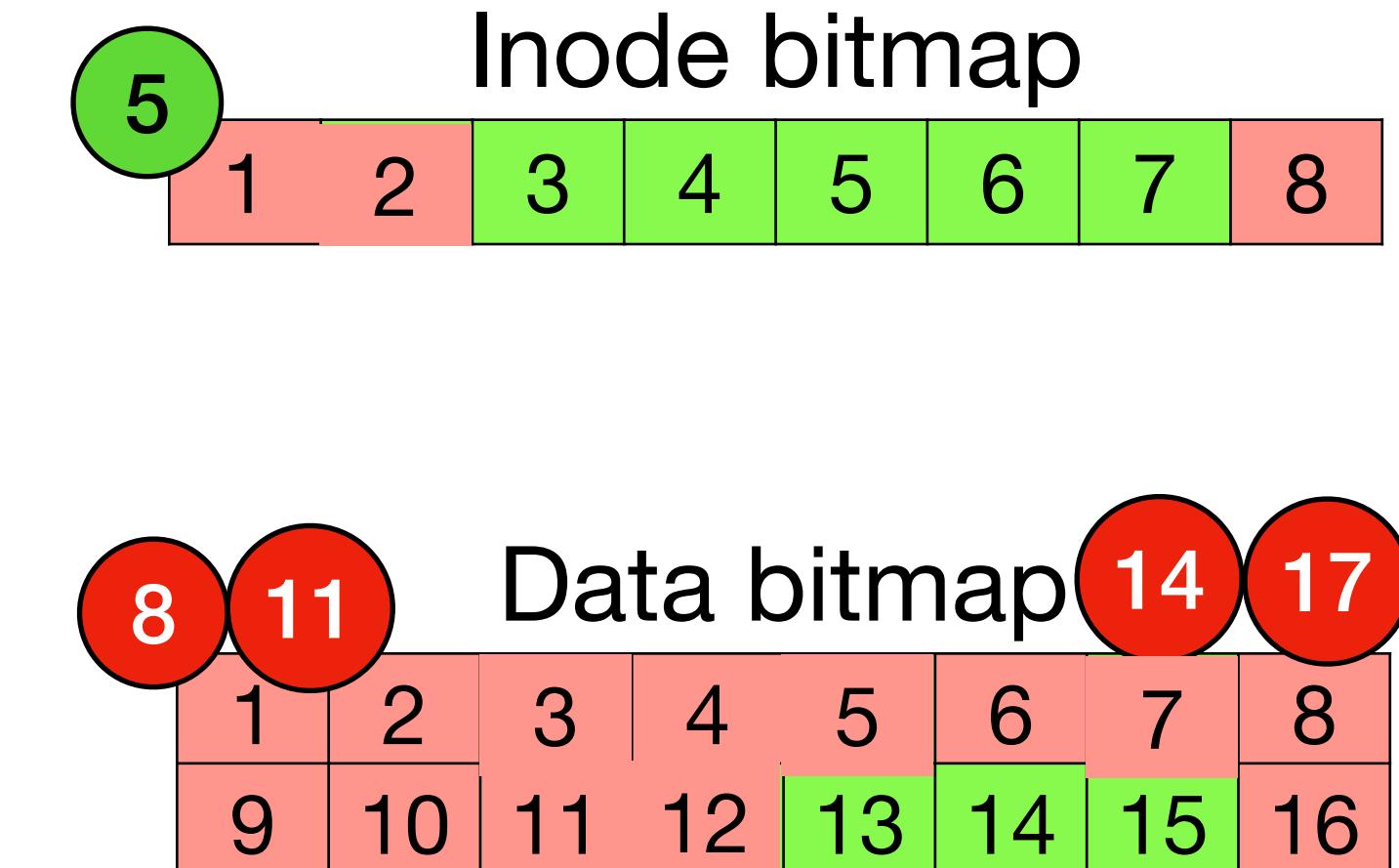
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time
9	3
15	5
12	7

```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

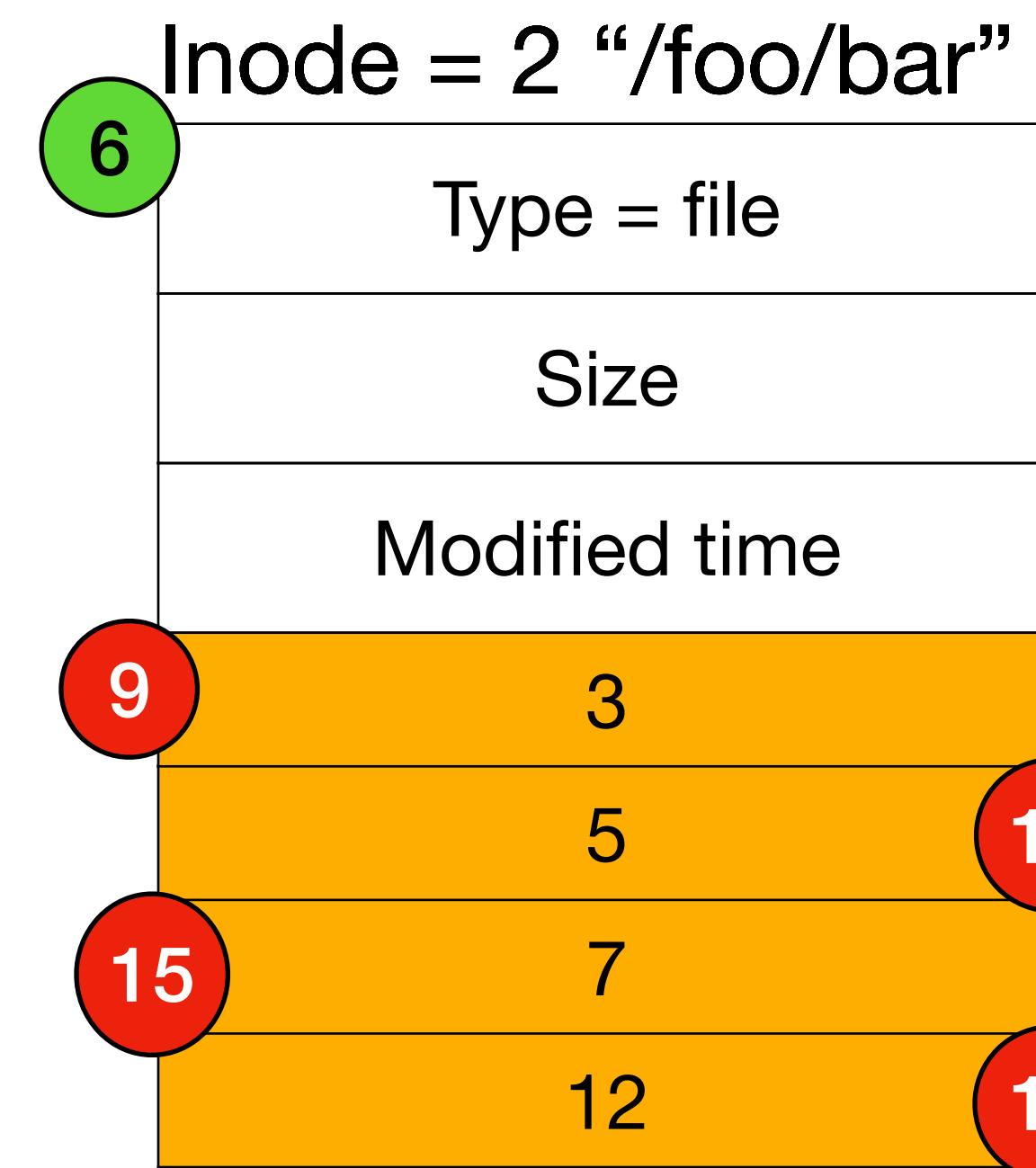
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

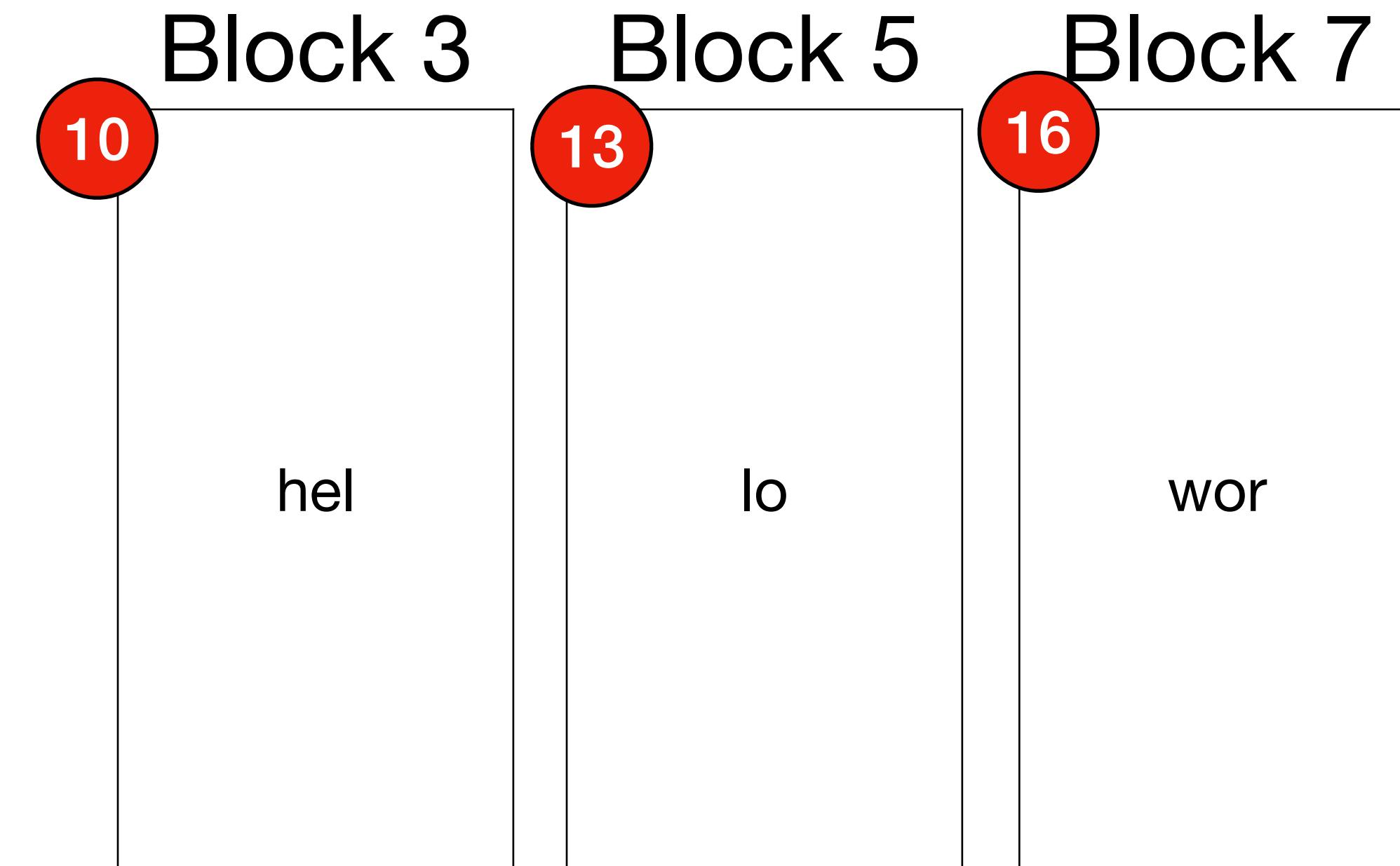
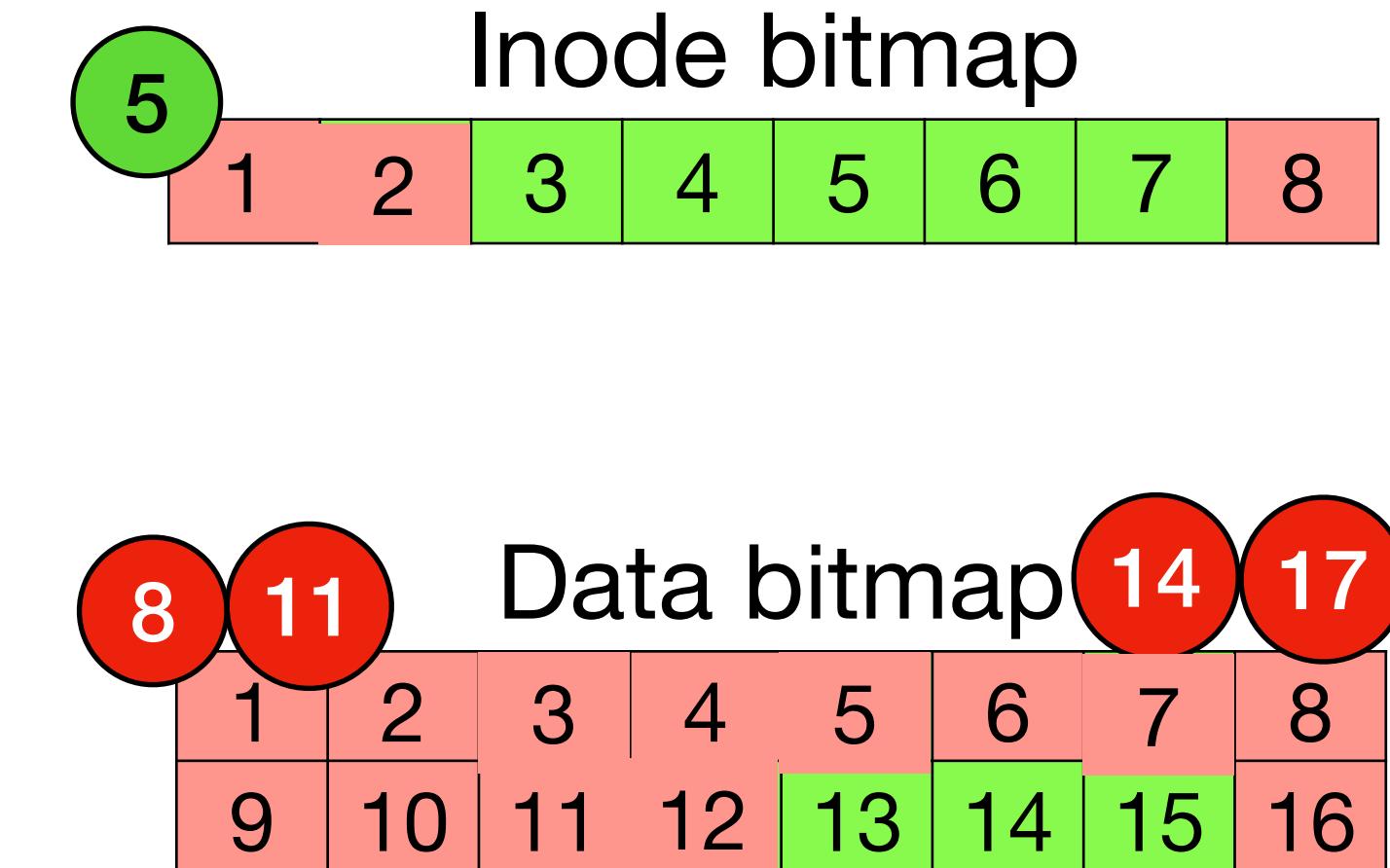
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

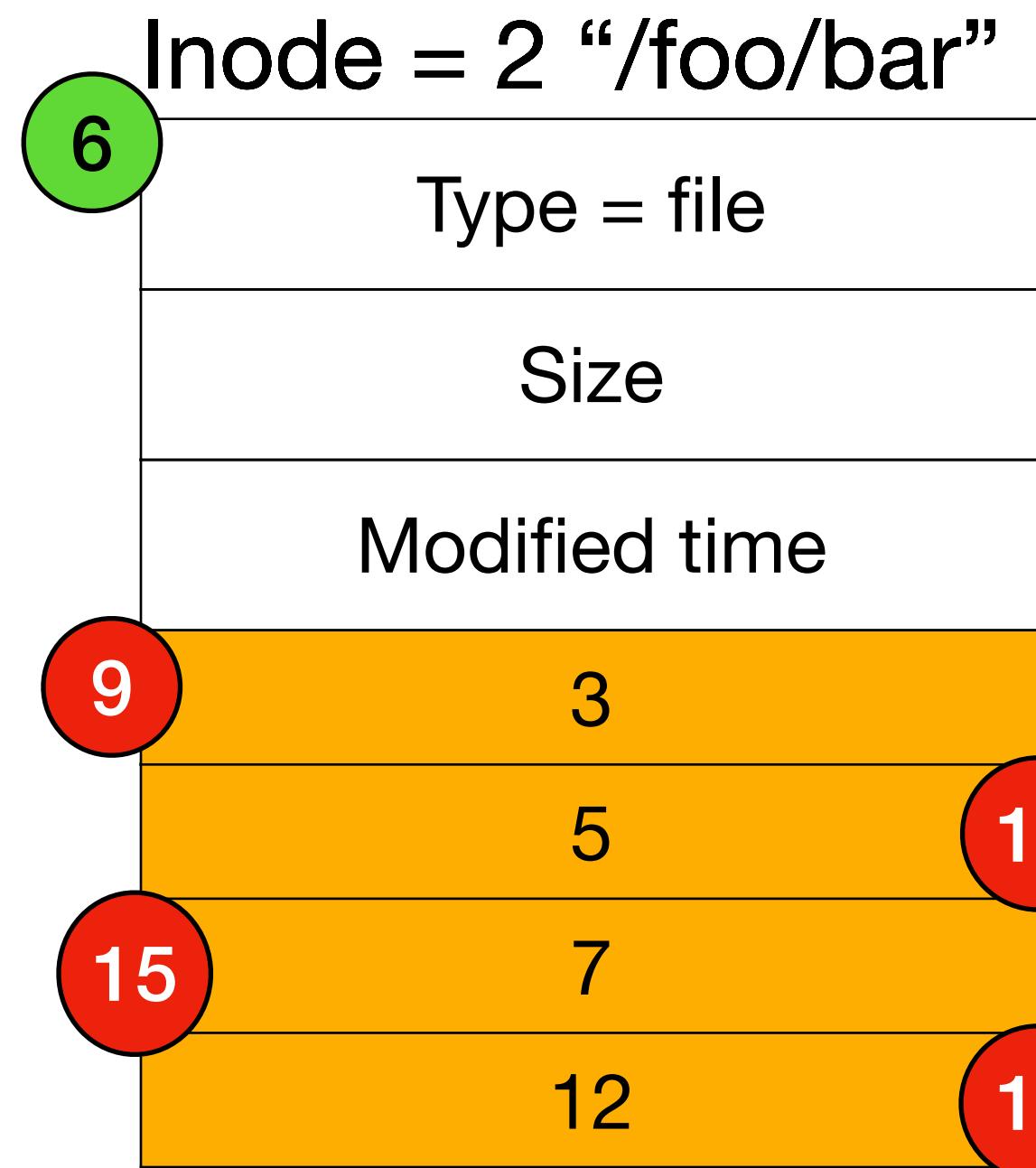
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	2
	1

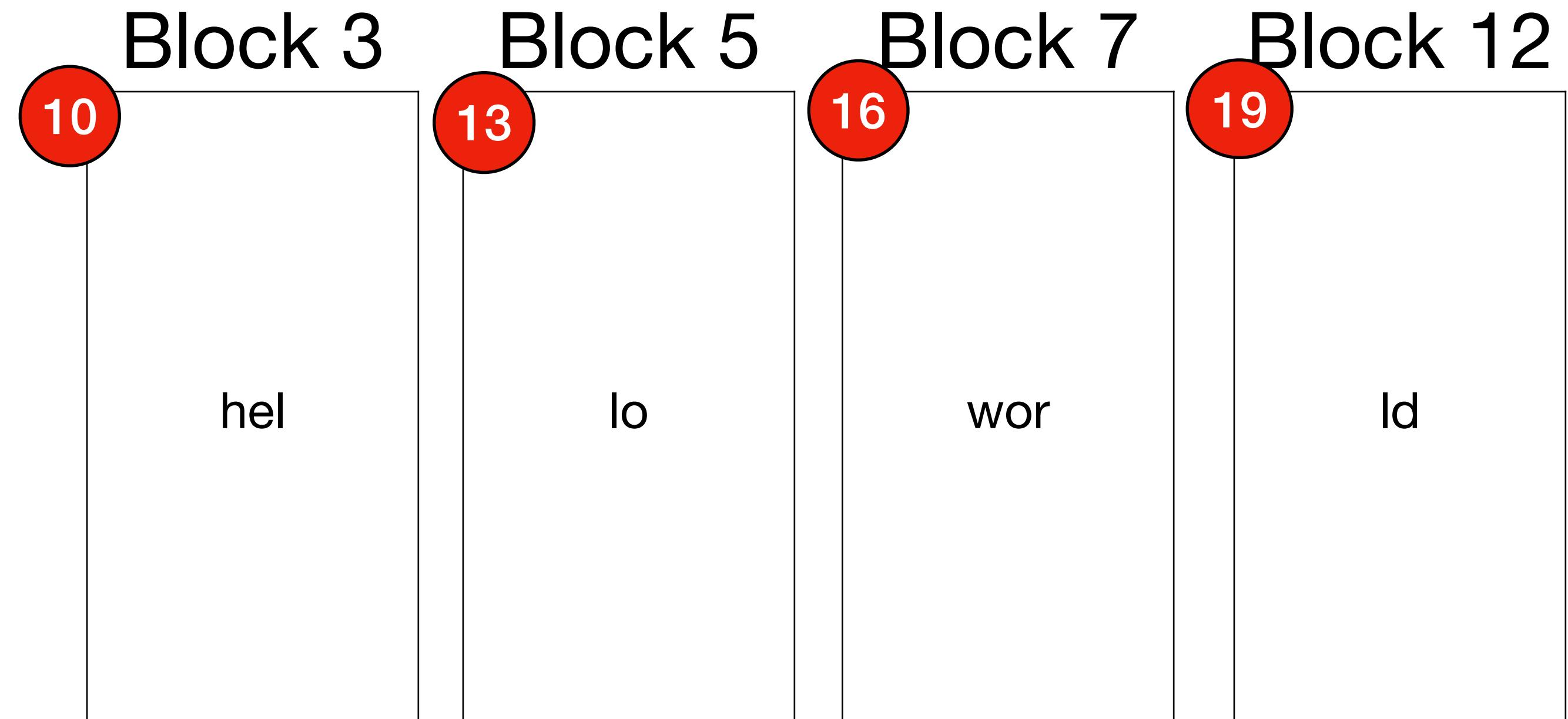
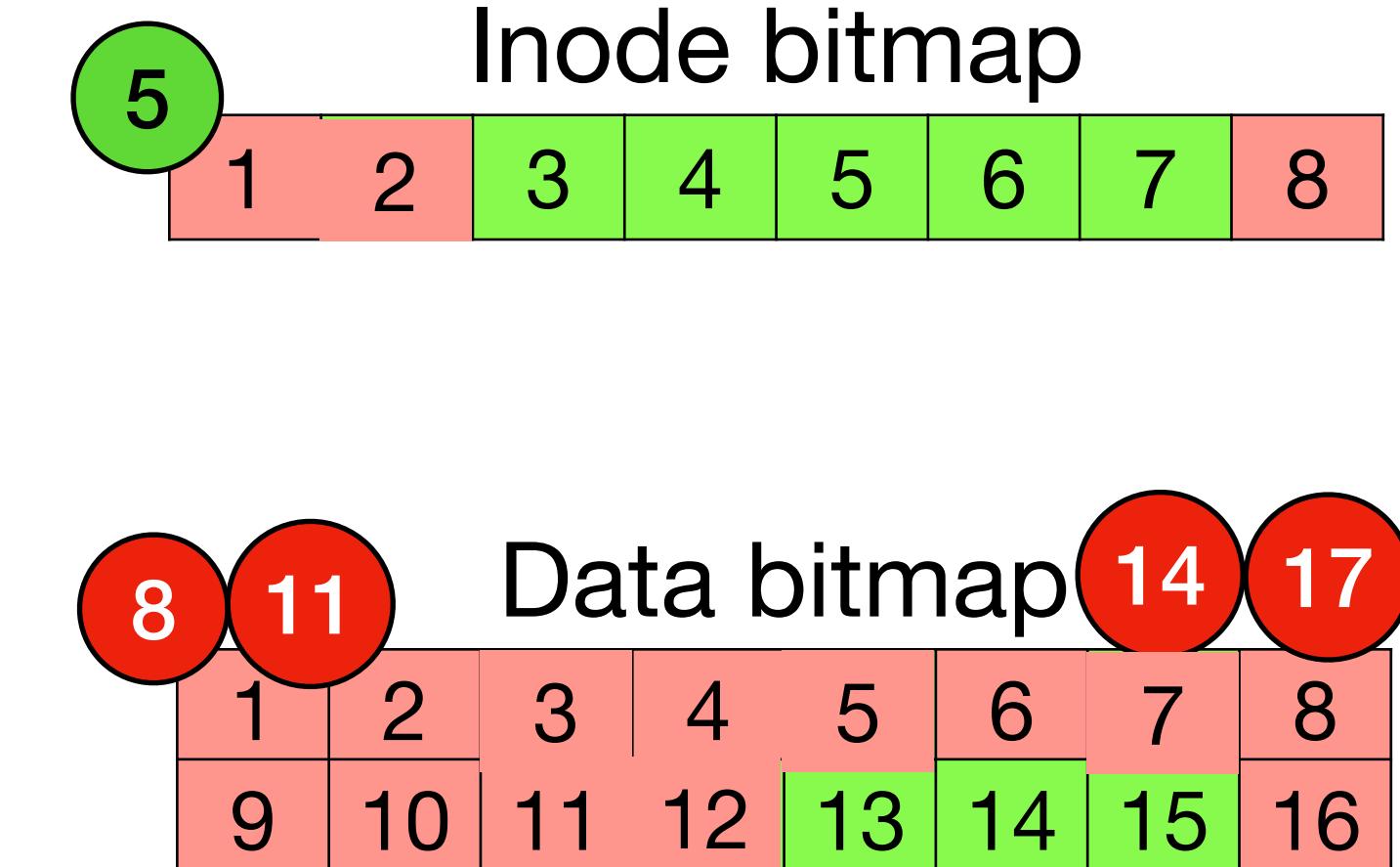
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	16
	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

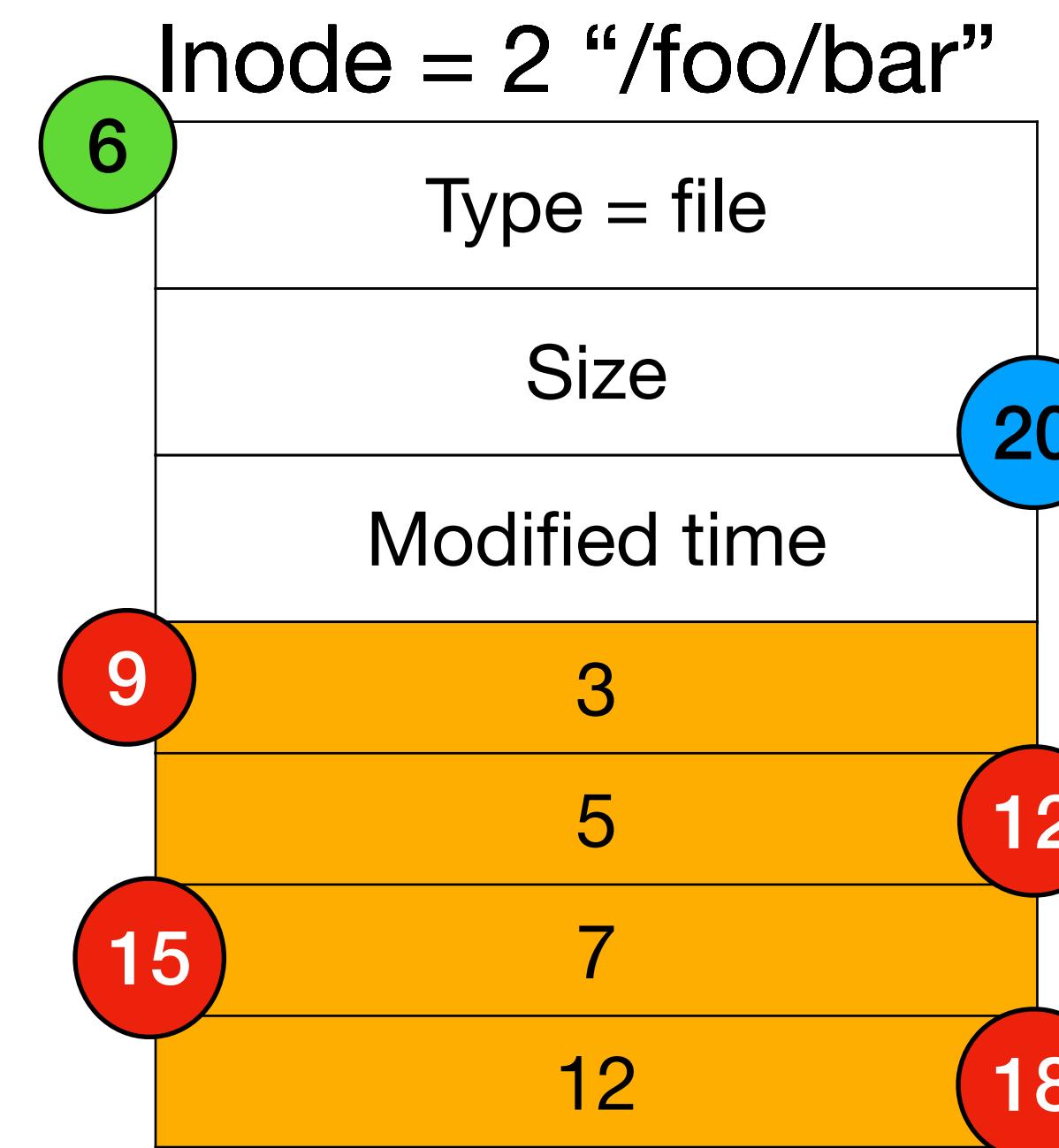
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	2

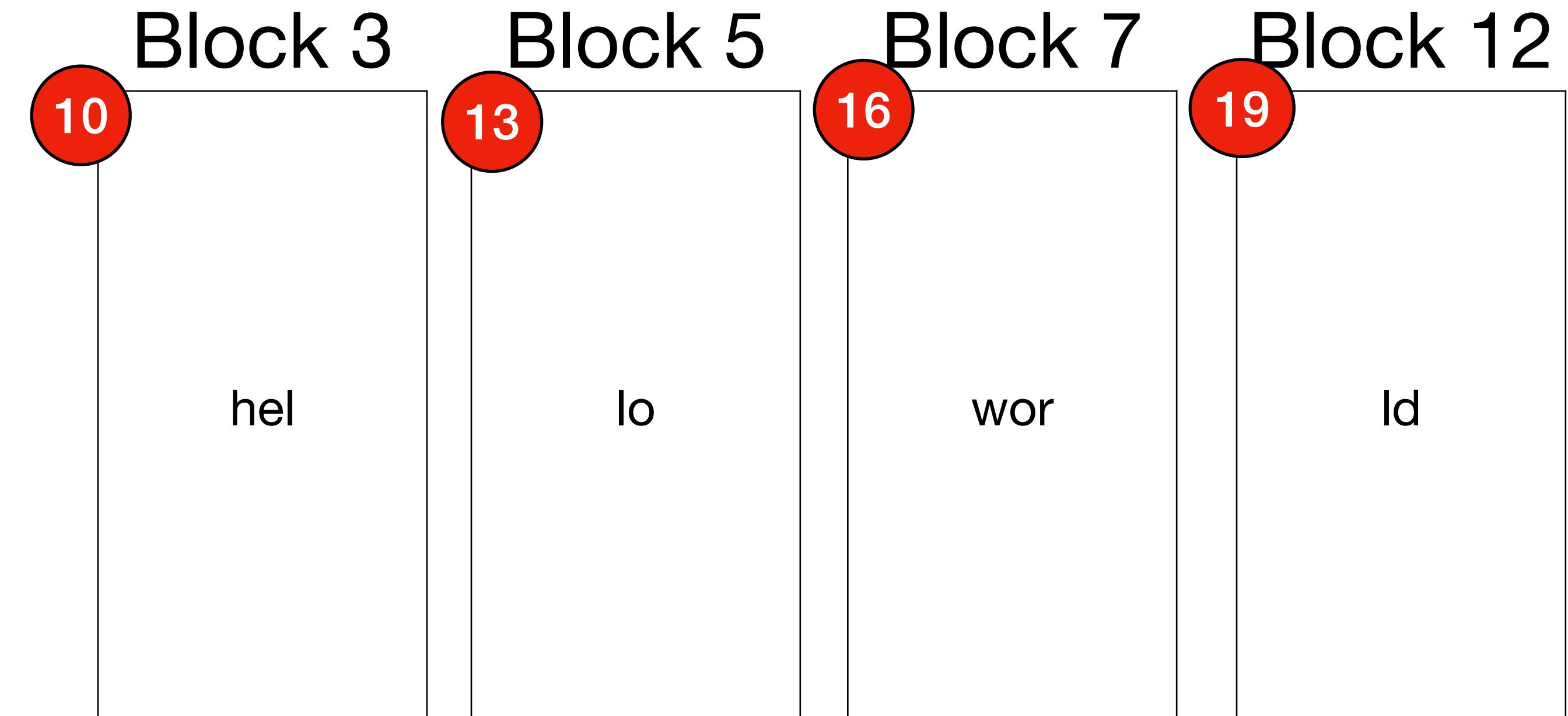
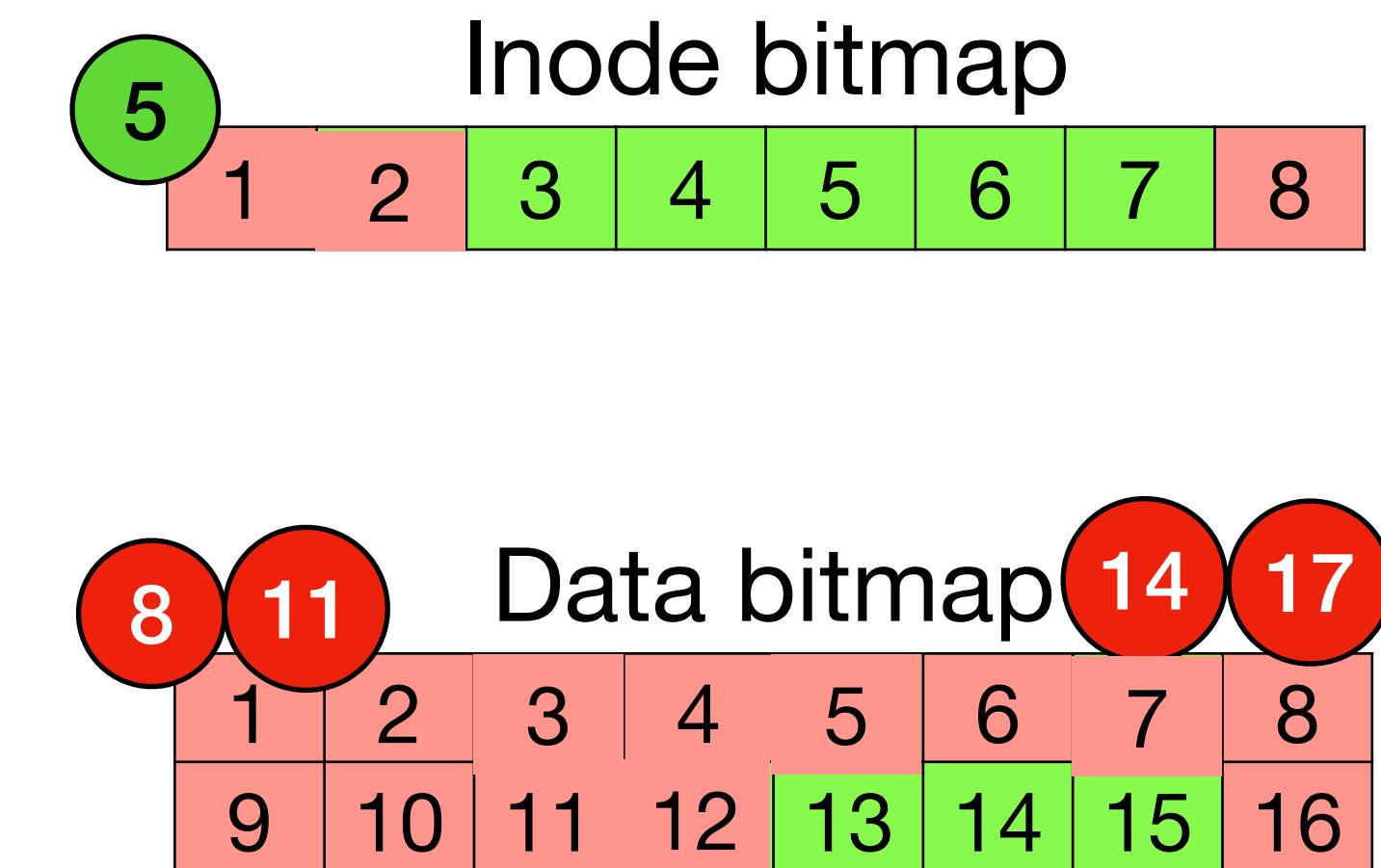
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Xv6 code walkthrough

p8-read-fs

- fs.h
 - ROOTINO=1: root folder is at the first inode
 - struct superblock
 - NDIRECT: 12 direct pointers. NINDIRECT: Number of pointers that can fit in the second-level pointer node (128).
 - MAXFILE: maximum number of data blocks (140). Max file size is 70 KB
 - struct dinode (64 bytes). IPB = 8.
 - struct dirent. 16 bytes. 32 directory entries in one data block of directory
- Makefile, mkfs.c creates a disk image with the file system containing one “/welcome.txt” file
- main.c reads and prints contents from welcome.txt

File System Optimizations

OSTEP Ch.41

Performance problems

- Fragmentation
- Poor locality
- Poor use of the buffer cache
- Minimal disk scheduling opportunities

Fragmentation problem

A1	A2	B1	B2	C1	C2	C3	D1				
----	----	----	----	----	----	----	----	--	--	--	--

Fragmentation problem

A1	A2	B1	B2	C1	C2	C3	D1				
----	----	----	----	----	----	----	----	--	--	--	--

A1	A2	B1	B2	C1	C2	C3	D1	A3			
----	----	----	----	----	----	----	----	----	--	--	--

Fragmentation problem

A1	A2	B1	B2	C1	C2	C3	D1				
----	----	----	----	----	----	----	----	--	--	--	--

A1	A2	B1	B2	C1	C2	C3	D1	A3			
----	----	----	----	----	----	----	----	----	--	--	--

A1	A2	B1	B2	C1	C2	C3		A3			
----	----	----	----	----	----	----	--	----	--	--	--

Fragmentation problem

A1	A2	B1	B2	C1	C2	C3	D1				
----	----	----	----	----	----	----	----	--	--	--	--

A1	A2	B1	B2	C1	C2	C3	D1	A3			
----	----	----	----	----	----	----	----	----	--	--	--

A1	A2	B1	B2	C1	C2	C3		A3			
----	----	----	----	----	----	----	--	----	--	--	--

A1	A2	B1	B2	C1	C2	C3	E1	A3			
----	----	----	----	----	----	----	----	----	--	--	--

Fragmentation problem

A1	A2	B1	B2	C1	C2	C3	D1				
----	----	----	----	----	----	----	----	--	--	--	--

A1	A2	B1	B2	C1	C2	C3	D1	A3			
----	----	----	----	----	----	----	----	----	--	--	--

A1	A2	B1	B2	C1	C2	C3		A3			
----	----	----	----	----	----	----	--	----	--	--	--

A1	A2	B1	B2	C1	C2	C3	E1	A3			
----	----	----	----	----	----	----	----	----	--	--	--

A1	A2			C1	C2	C3	E1	A3			
----	----	--	--	----	----	----	----	----	--	--	--

Fragmentation problem

A1	A2	B1	B2	C1	C2	C3	D1				
----	----	----	----	----	----	----	----	--	--	--	--

A1	A2	B1	B2	C1	C2	C3	D1	A3			
----	----	----	----	----	----	----	----	----	--	--	--

A1	A2	B1	B2	C1	C2	C3		A3			
----	----	----	----	----	----	----	--	----	--	--	--

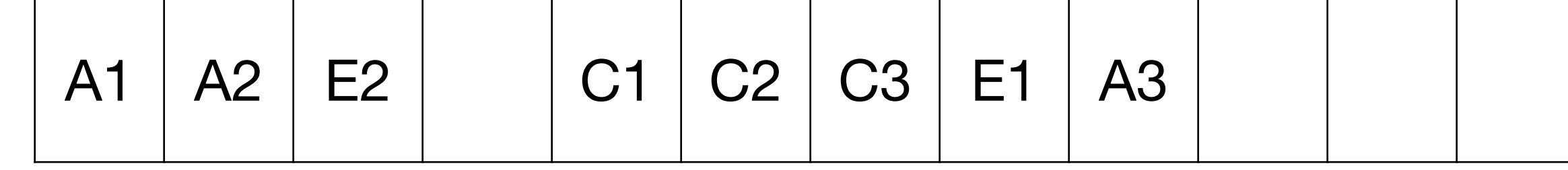
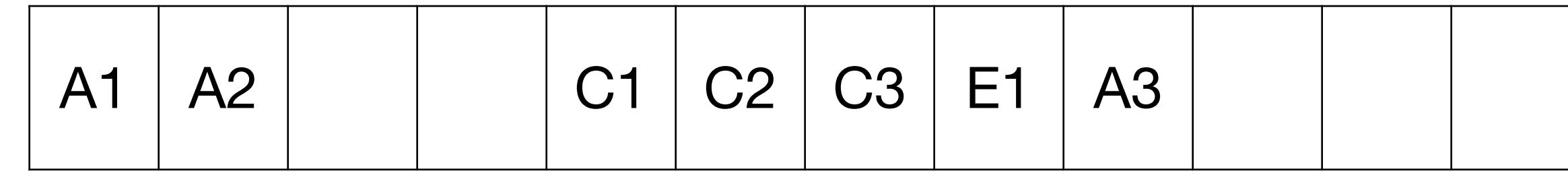
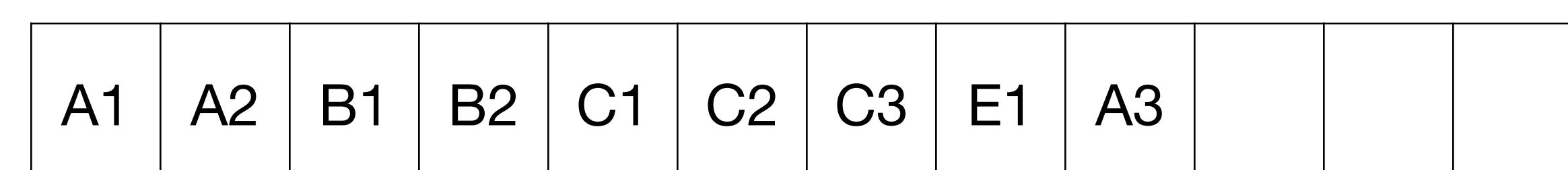
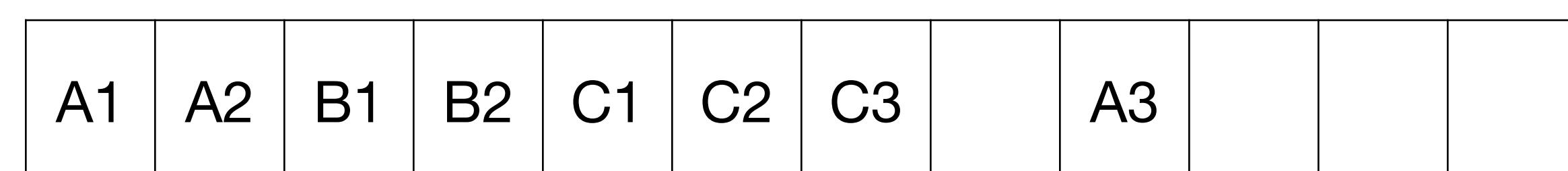
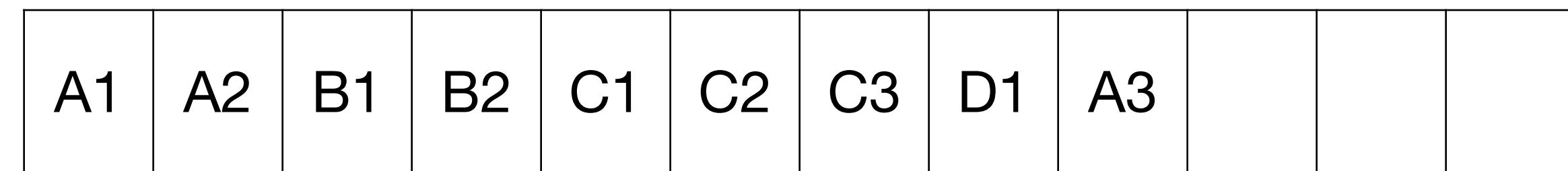
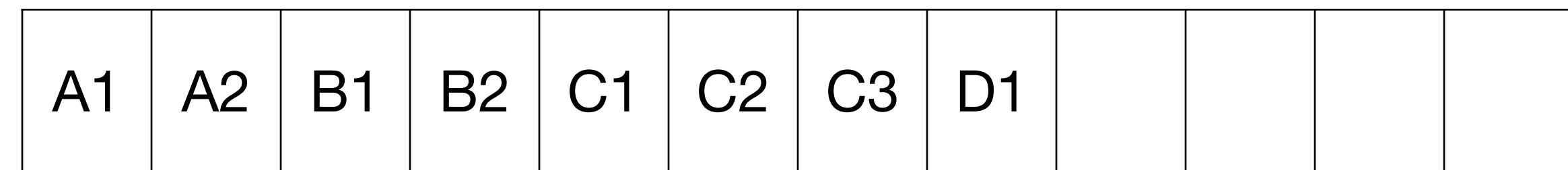
A1	A2	B1	B2	C1	C2	C3	E1	A3			
----	----	----	----	----	----	----	----	----	--	--	--

A1	A2			C1	C2	C3	E1	A3			
----	----	--	--	----	----	----	----	----	--	--	--

A1	A2	E2		C1	C2	C3	E1	A3			
----	----	----	--	----	----	----	----	----	--	--	--

Fragmentation problem

- Over time, a file's data blocks get spread all over the disk
 - Disk head(s) need to go back and forth to read files sequentially



Fragmentation problem

Defragmentation

- Defragmenter rearranges data blocks

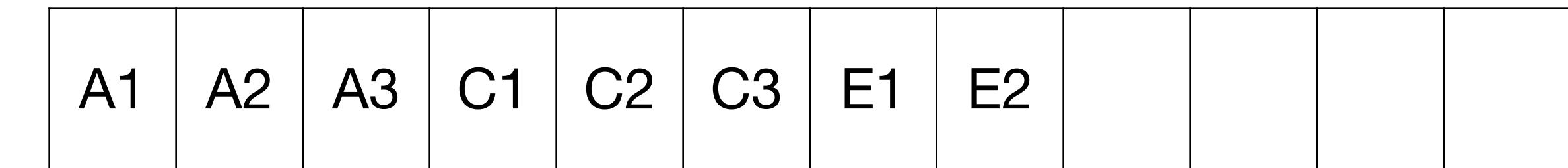
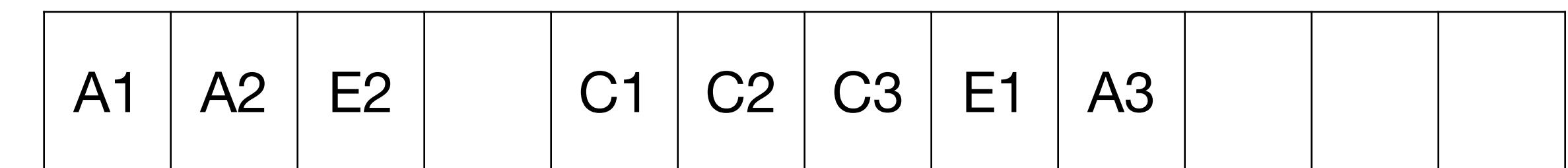
A1	A2	E2		C1	C2	C3	E1	A3				
----	----	----	--	----	----	----	----	----	--	--	--	--

A1	A2	A3	C1	C2	C3	E1	E2					
----	----	----	----	----	----	----	----	--	--	--	--	--

Fragmentation problem

Defragmentation

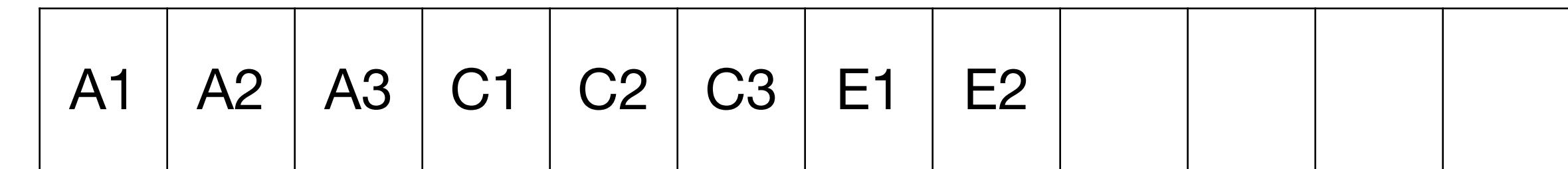
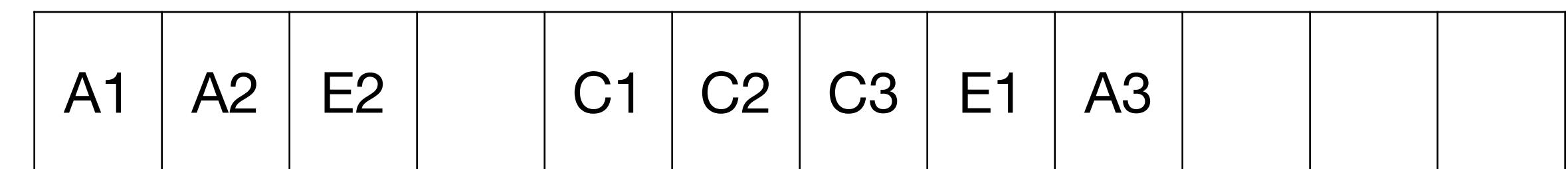
- Defragmenter rearranges data blocks
 - Also updates data block pointers in file's inode



Fragmentation problem

Defragmentation

- Defragmenter rearranges data blocks
 - Also updates data block pointers in file's inode
- Modern FS such as ext4 do defragmentation in background: without making FS unavailable



Fragmentation problem

Pre-allocate blocks

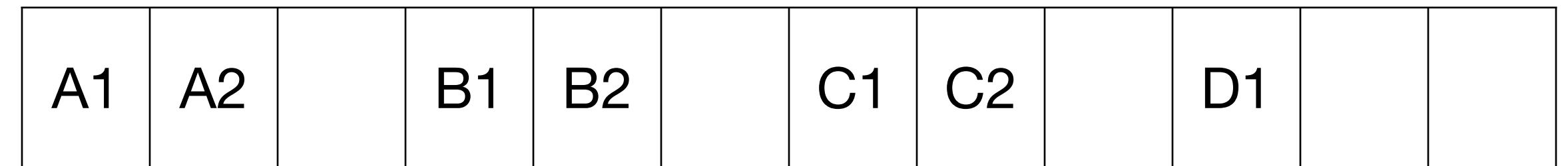
Fragmentation problem

Pre-allocate blocks

- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation

Fragmentation problem

Pre-allocate blocks

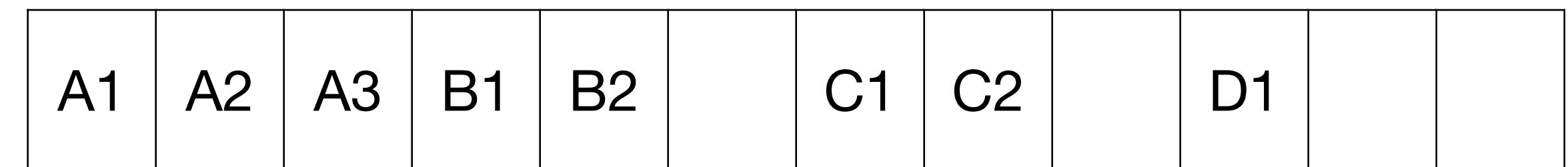
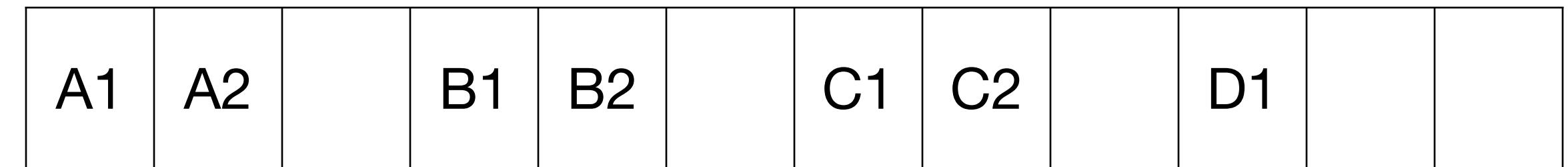


- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation

Fragmentation problem

Pre-allocate blocks

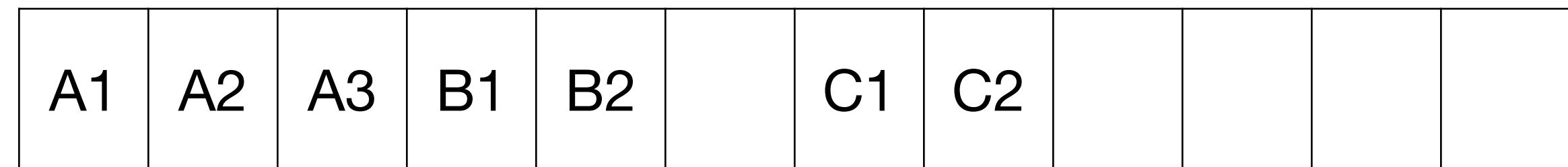
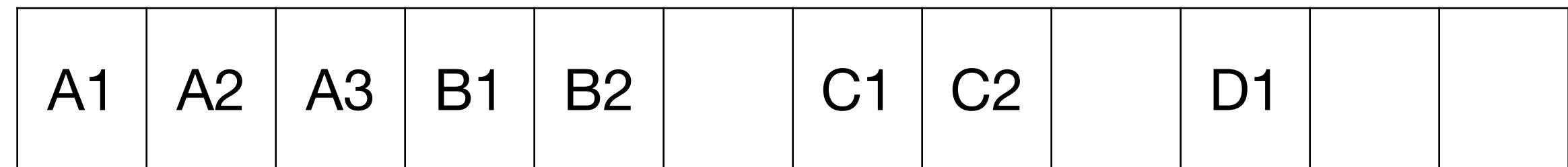
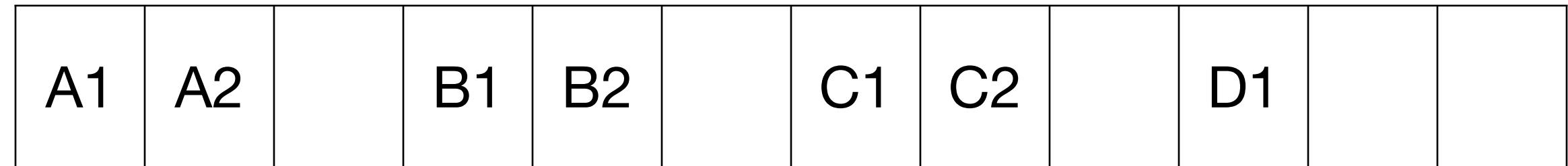
- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation



Fragmentation problem

Pre-allocate blocks

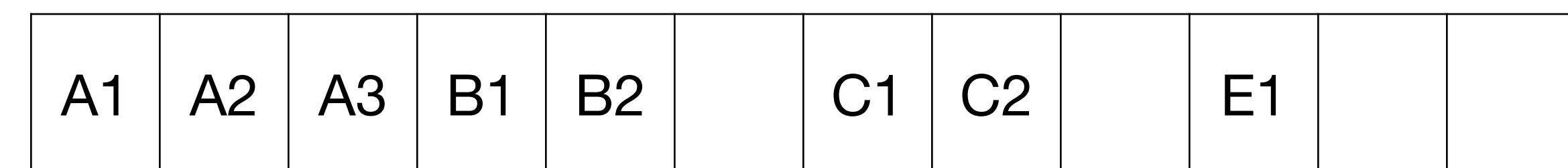
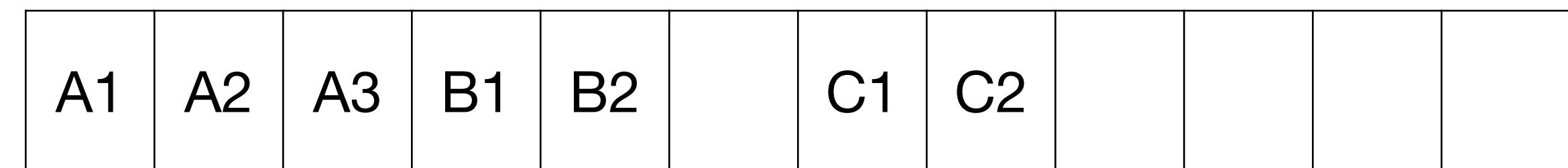
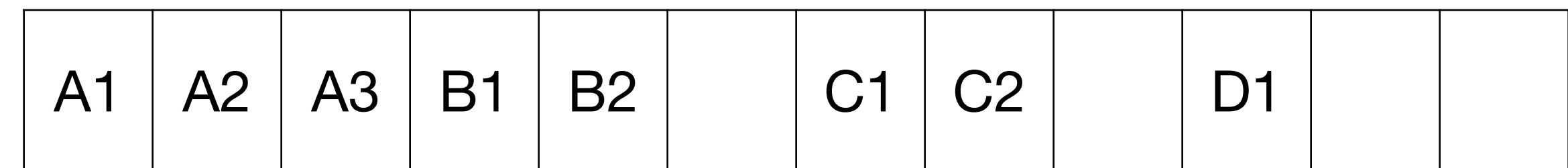
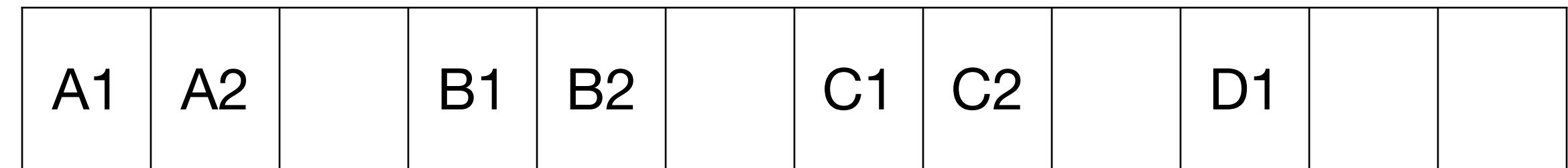
- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation



Fragmentation problem

Pre-allocate blocks

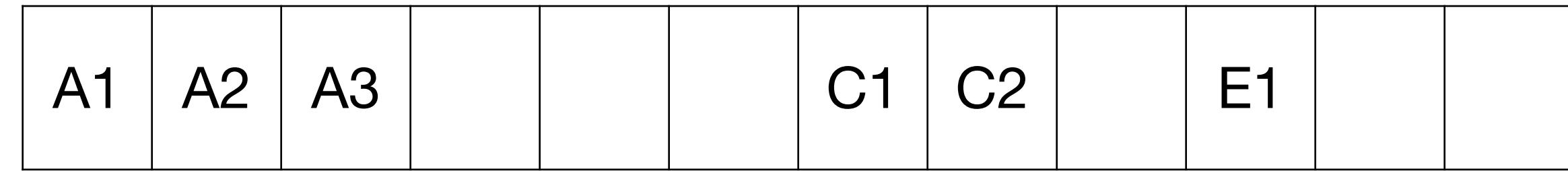
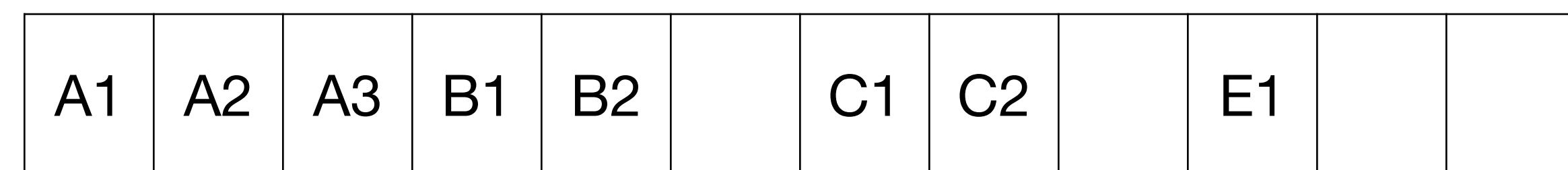
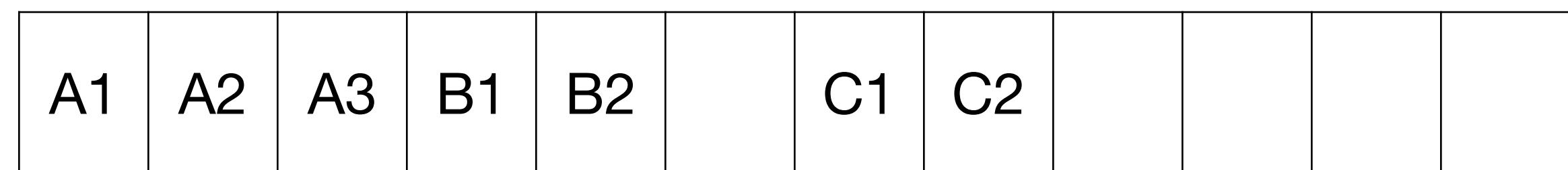
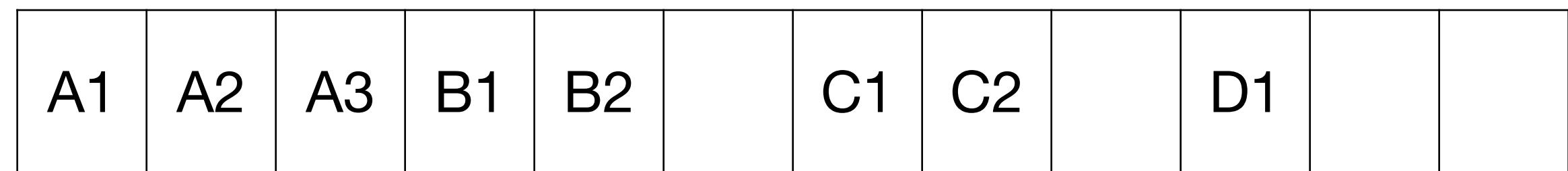
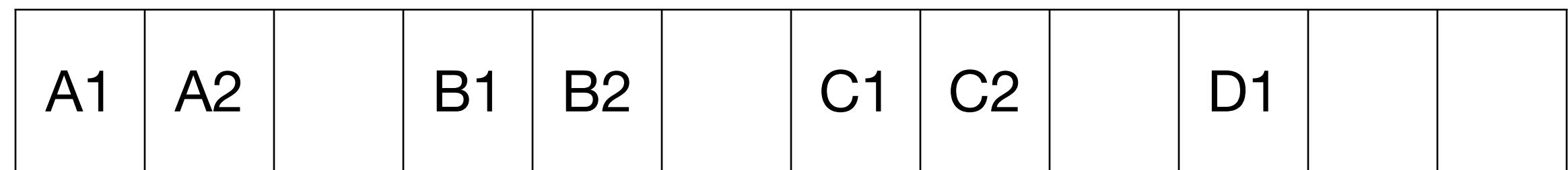
- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation



Fragmentation problem

Pre-allocate blocks

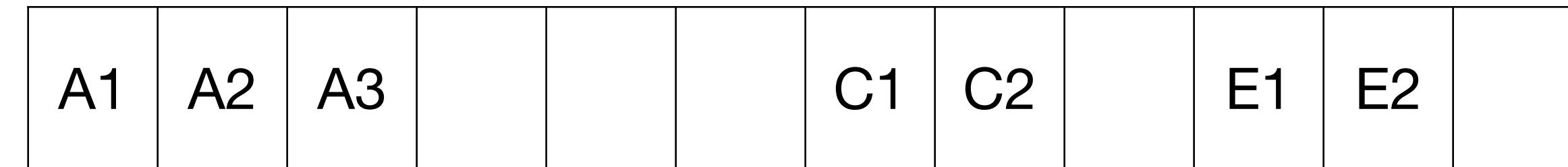
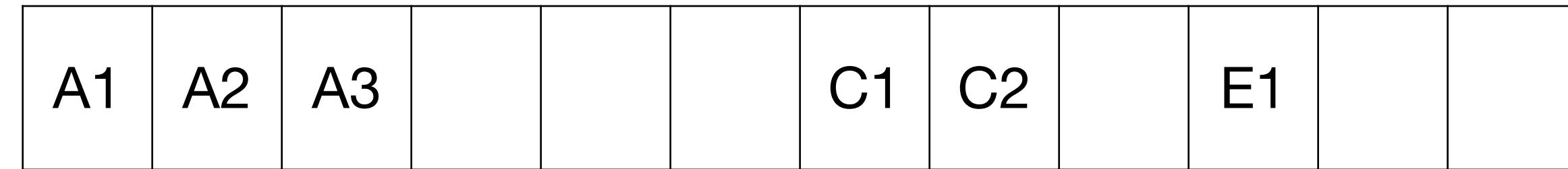
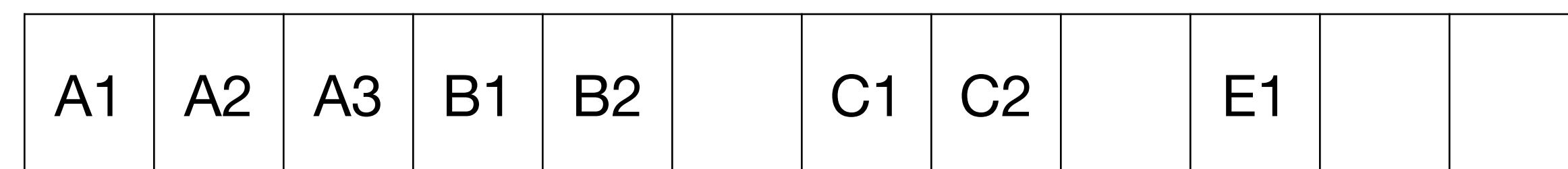
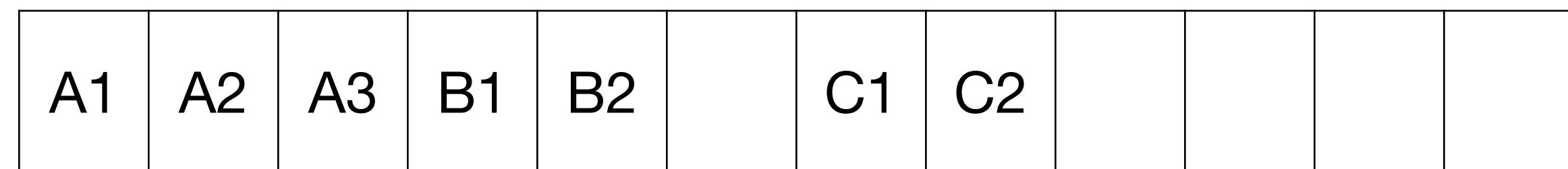
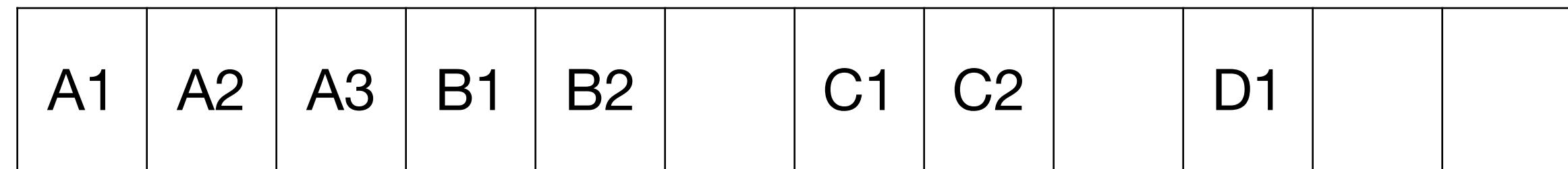
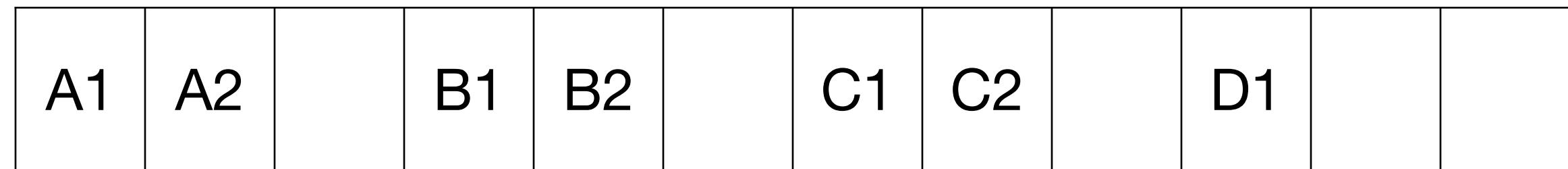
- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation



Fragmentation problem

Pre-allocate blocks

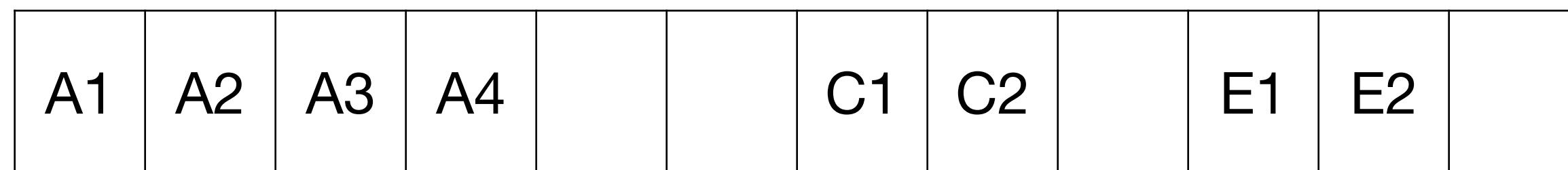
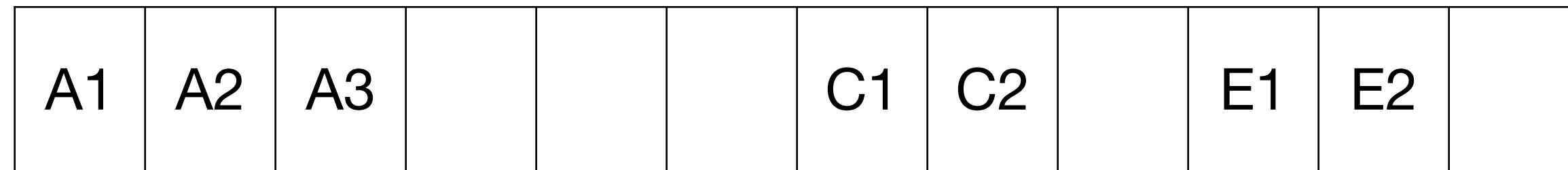
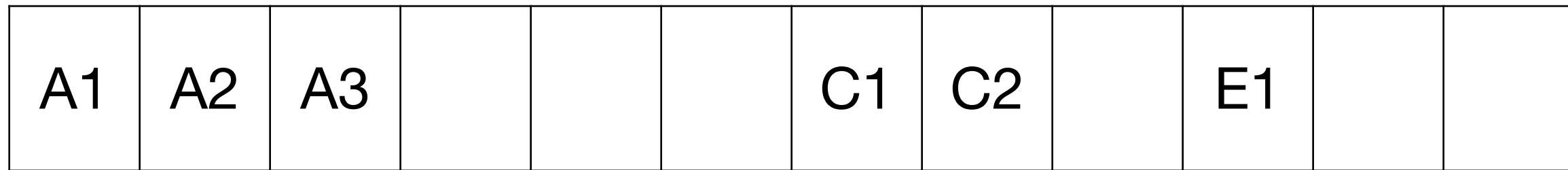
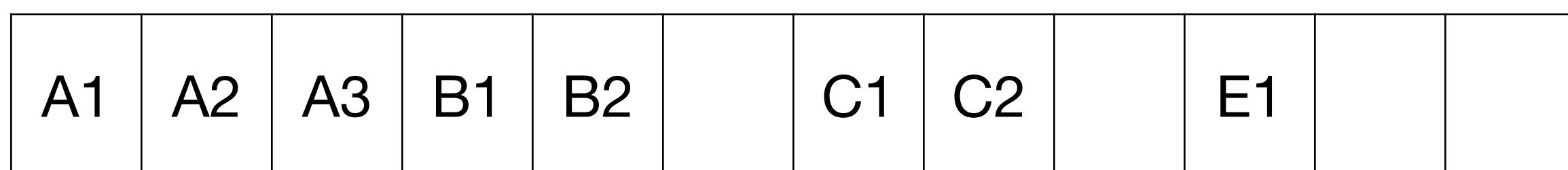
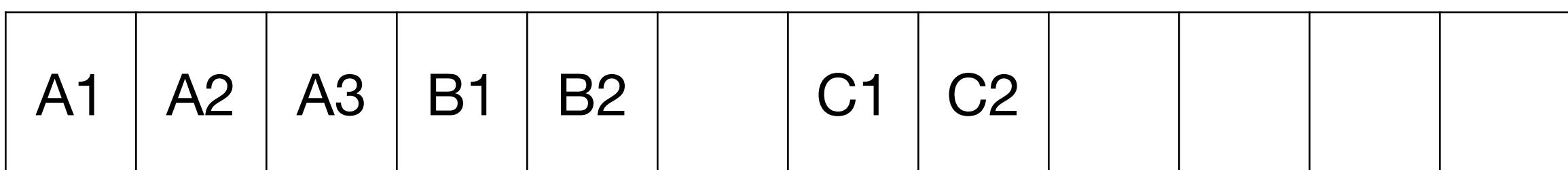
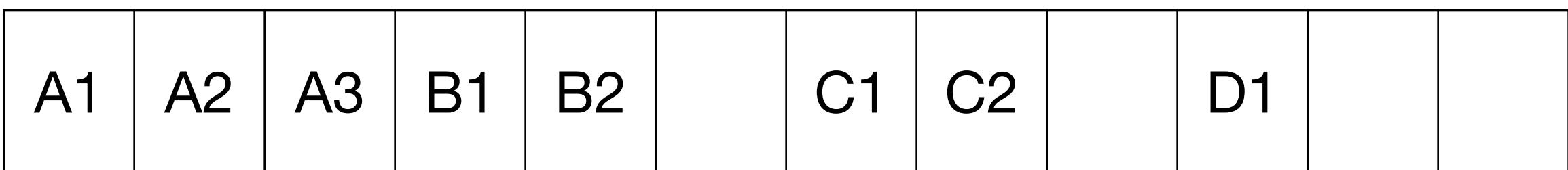
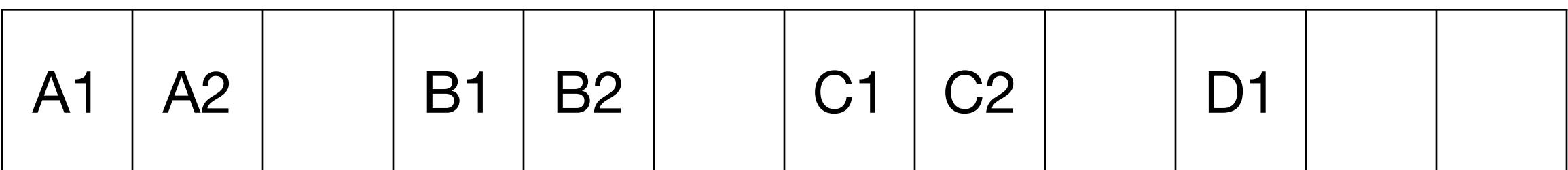
- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation



Fragmentation problem

Pre-allocate blocks

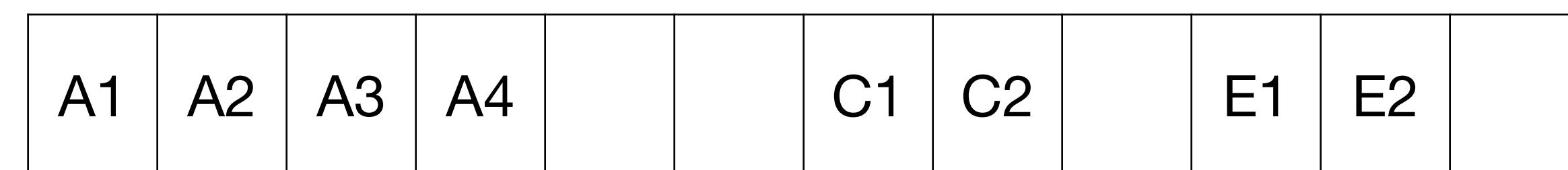
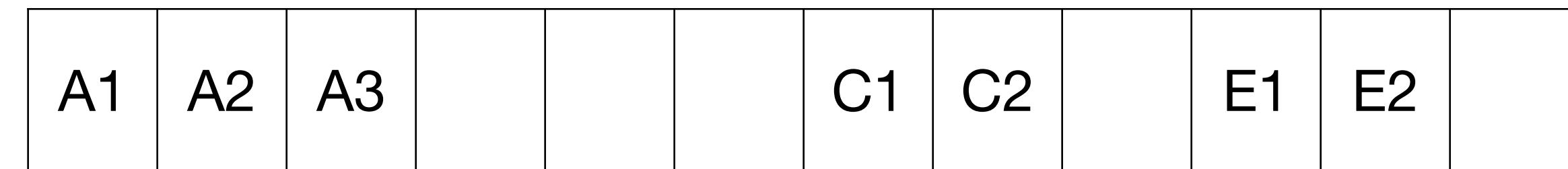
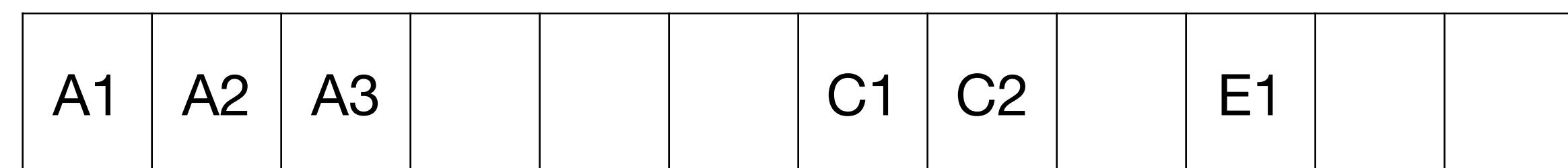
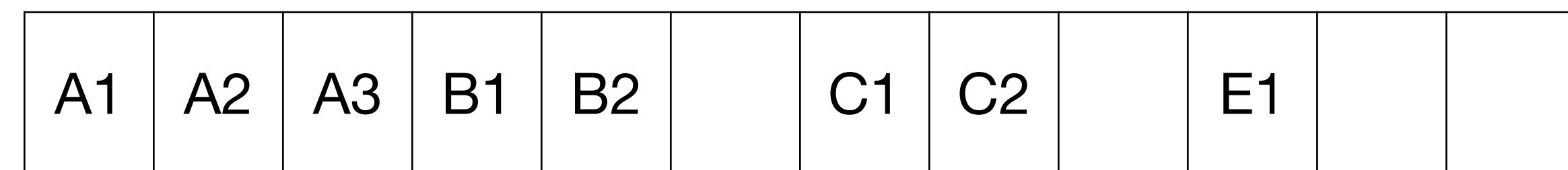
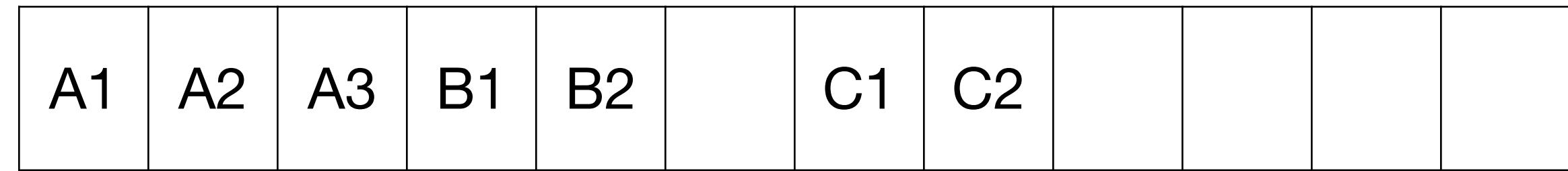
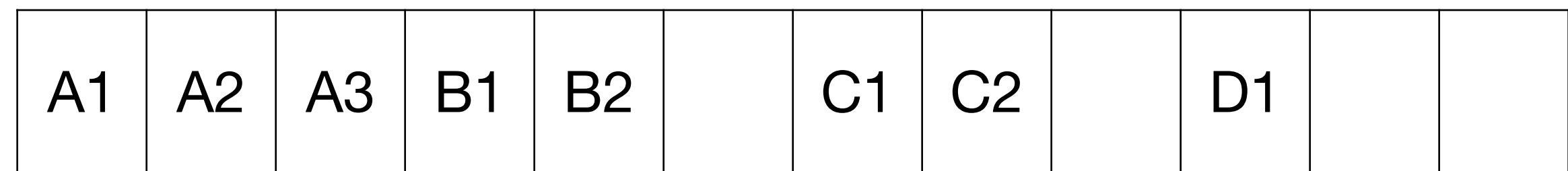
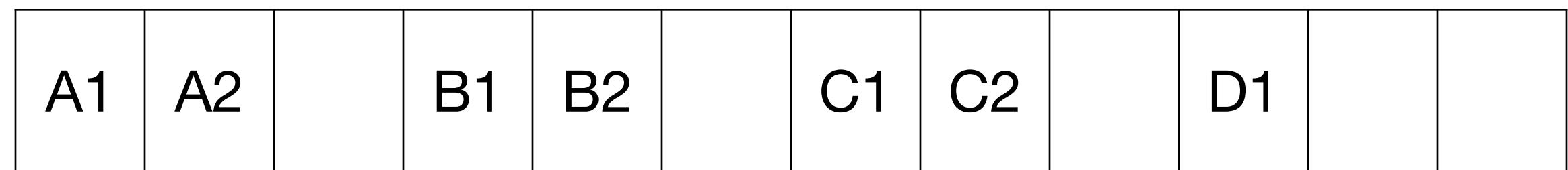
- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation



Fragmentation problem

Pre-allocate blocks

- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation
 - Reduce metadata lookup overhead by keeping extents

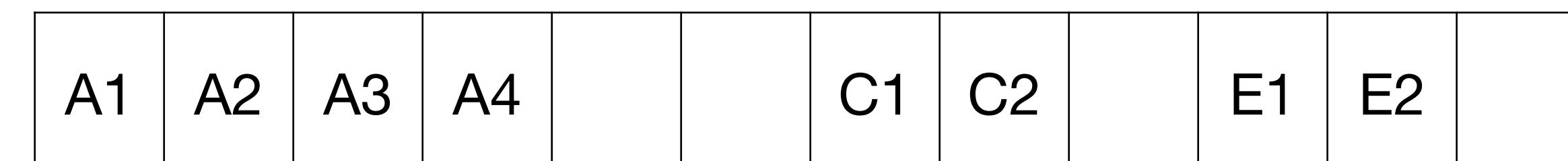
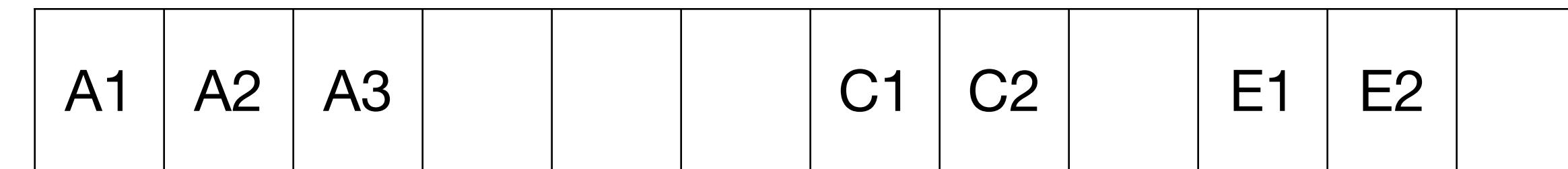
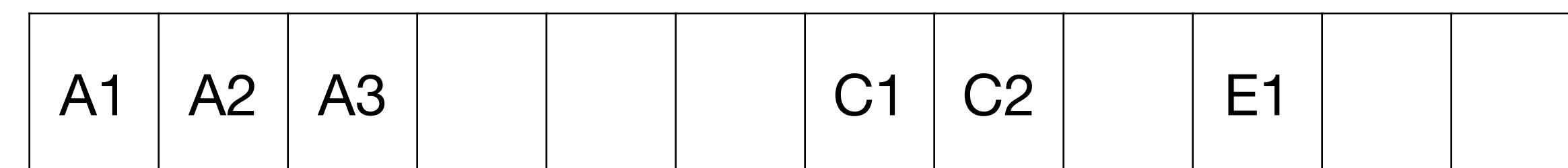
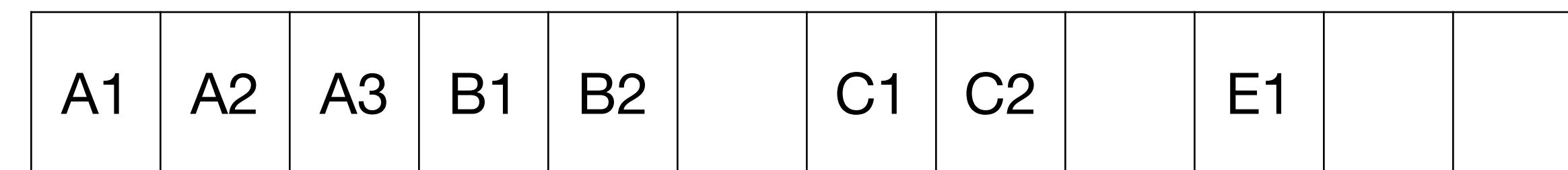
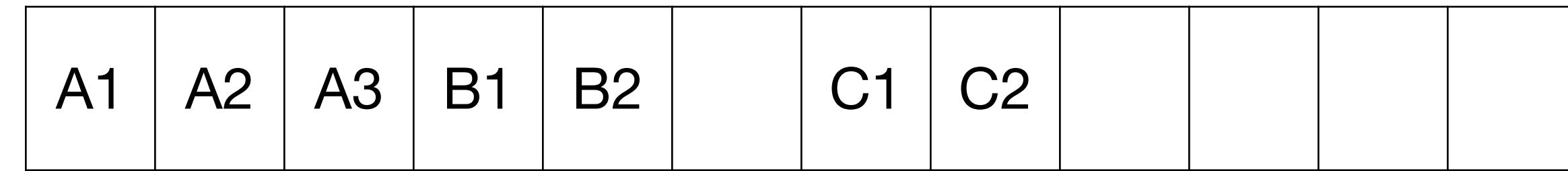
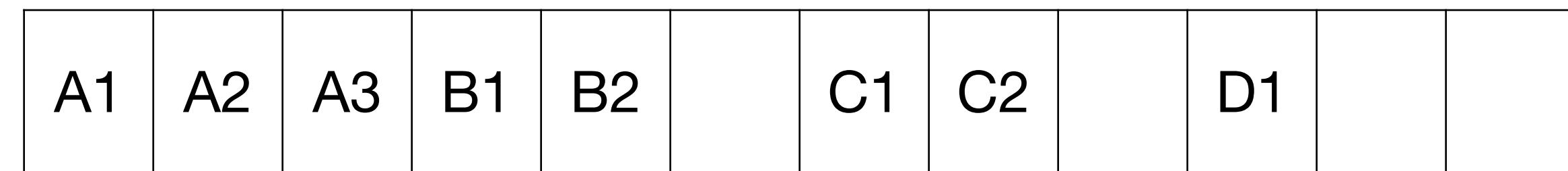
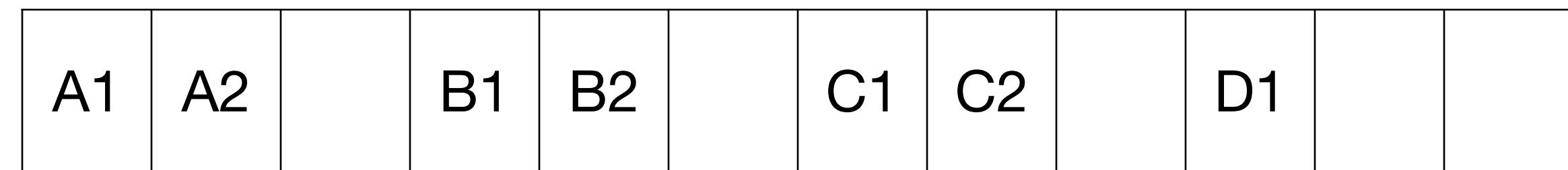


Fragmentation problem

Pre-allocate blocks

- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation
 - Reduce metadata lookup overhead by keeping extents

A
1
2
3
4



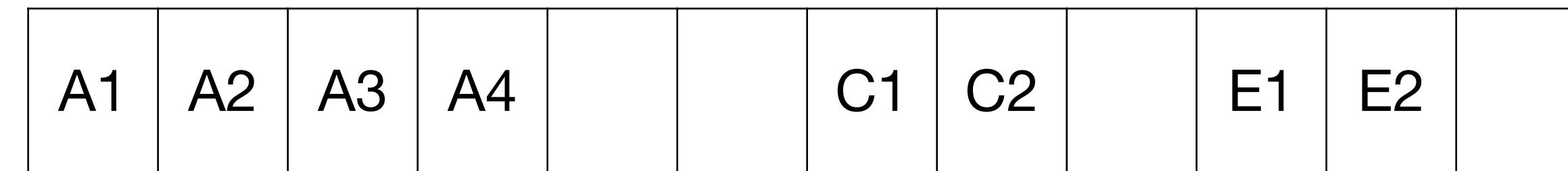
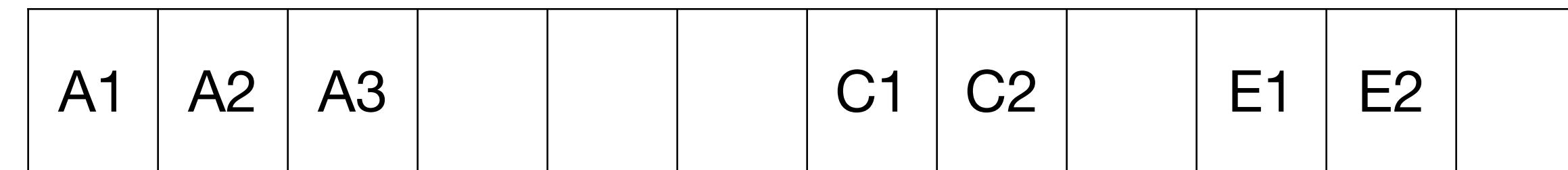
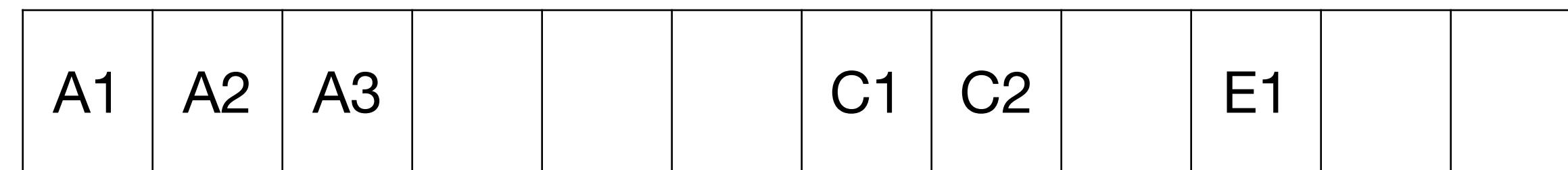
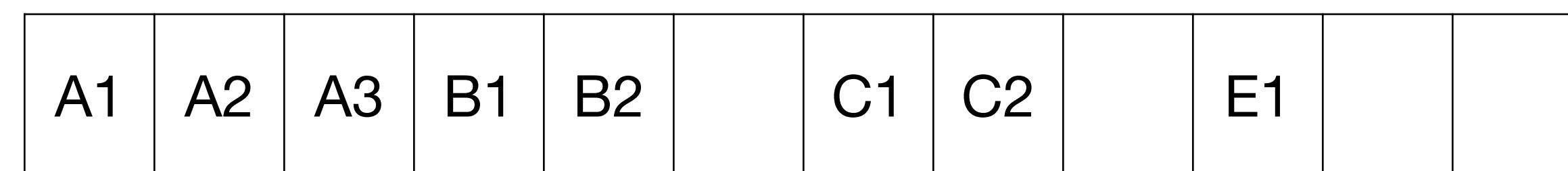
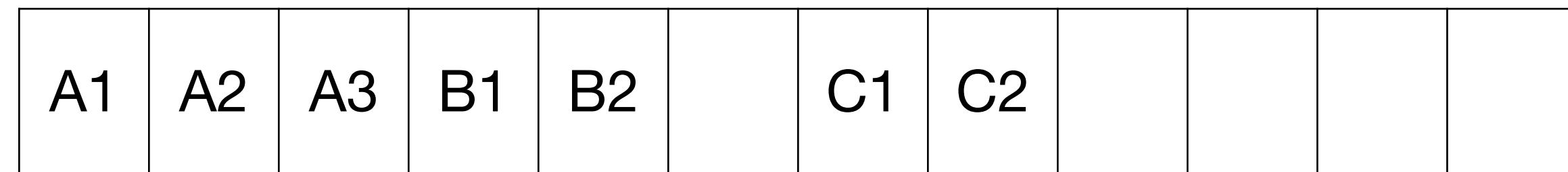
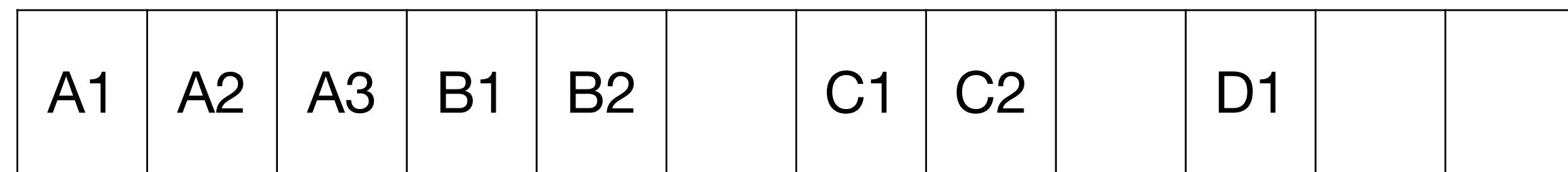
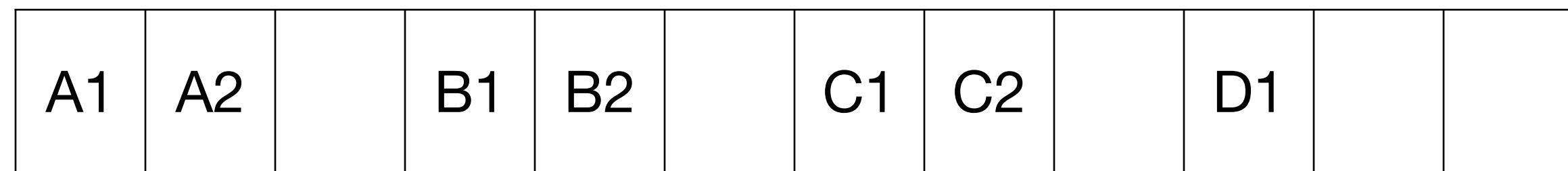
Fragmentation problem

Pre-allocate blocks

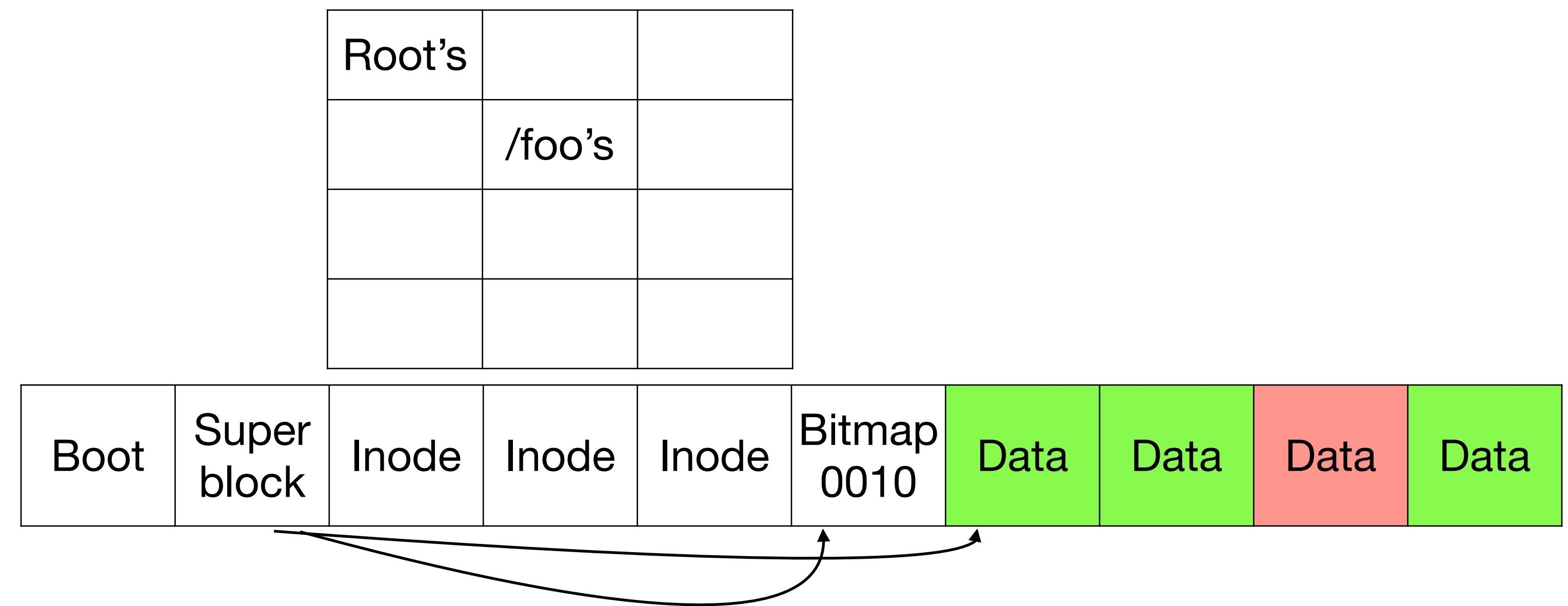
- Disks have grown bigger
 - Ext3 pre-allocates 8 blocks at file creation
 - Reduce metadata lookup overhead by keeping extents

A
1
2
3
4

A
1, 4

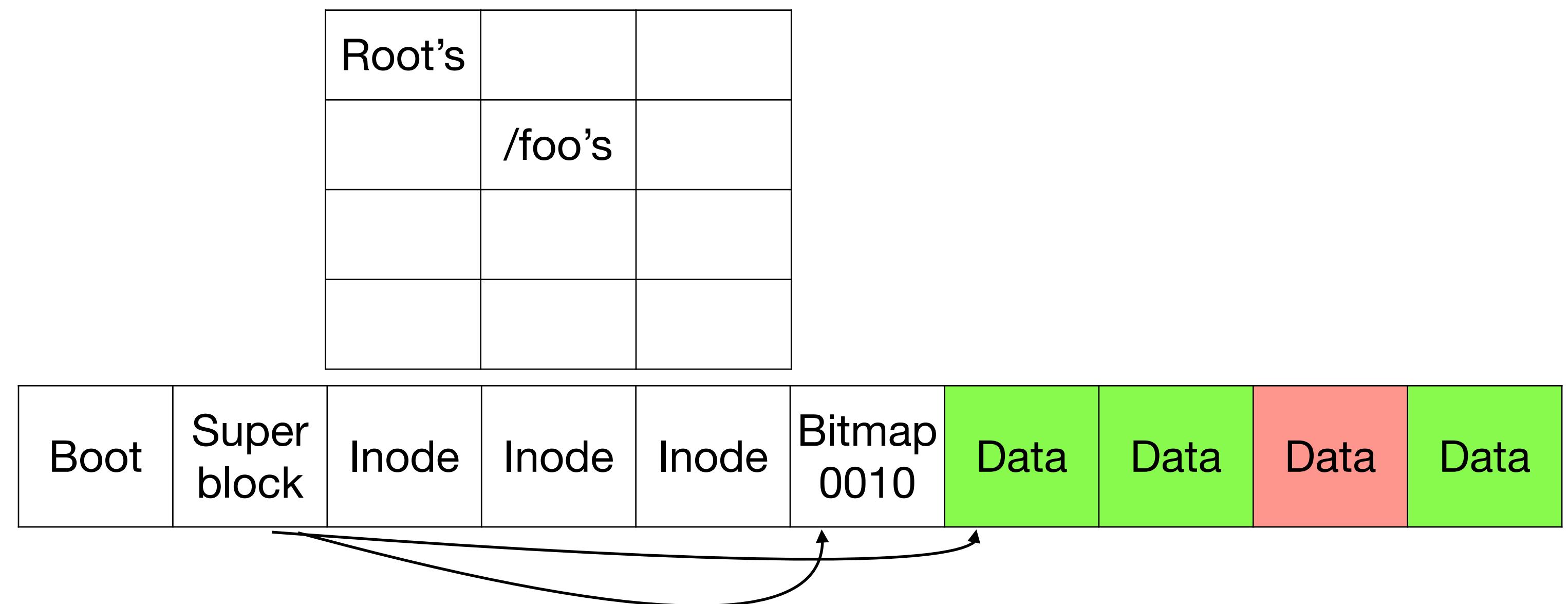


Locality problem



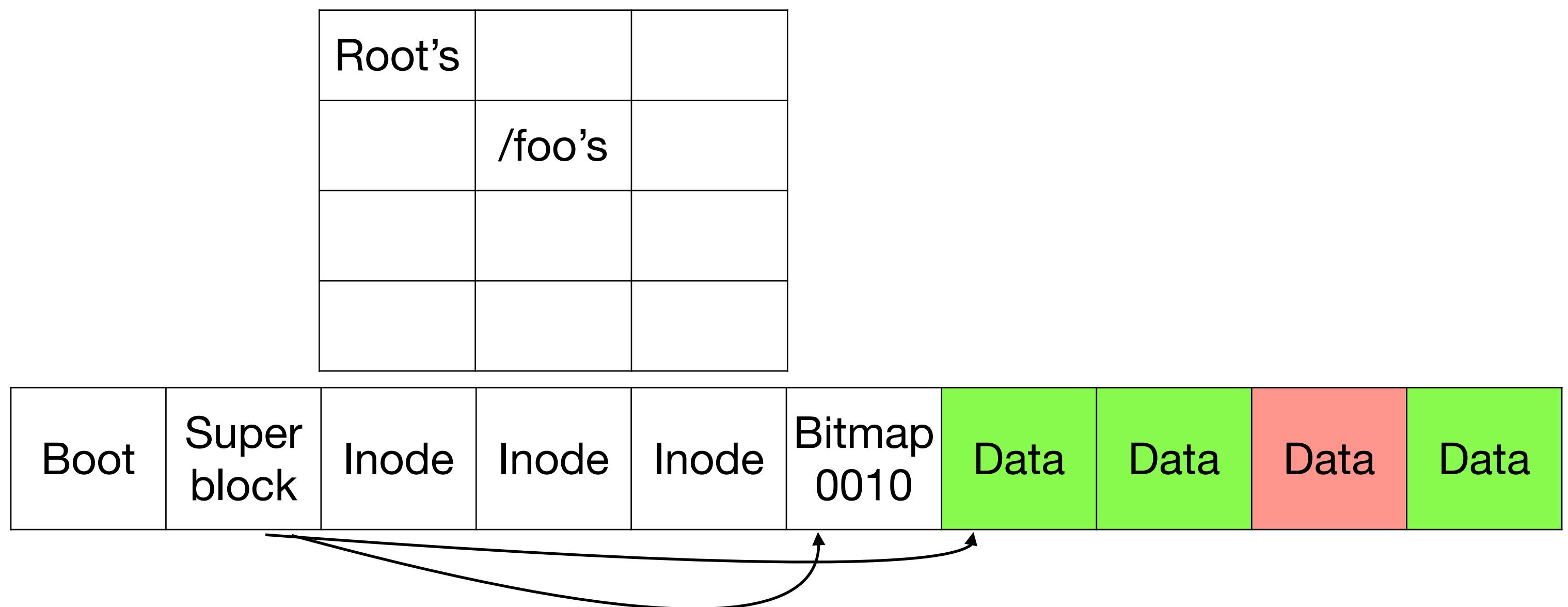
Locality problem

- Fast `ls`: just lookup inodes



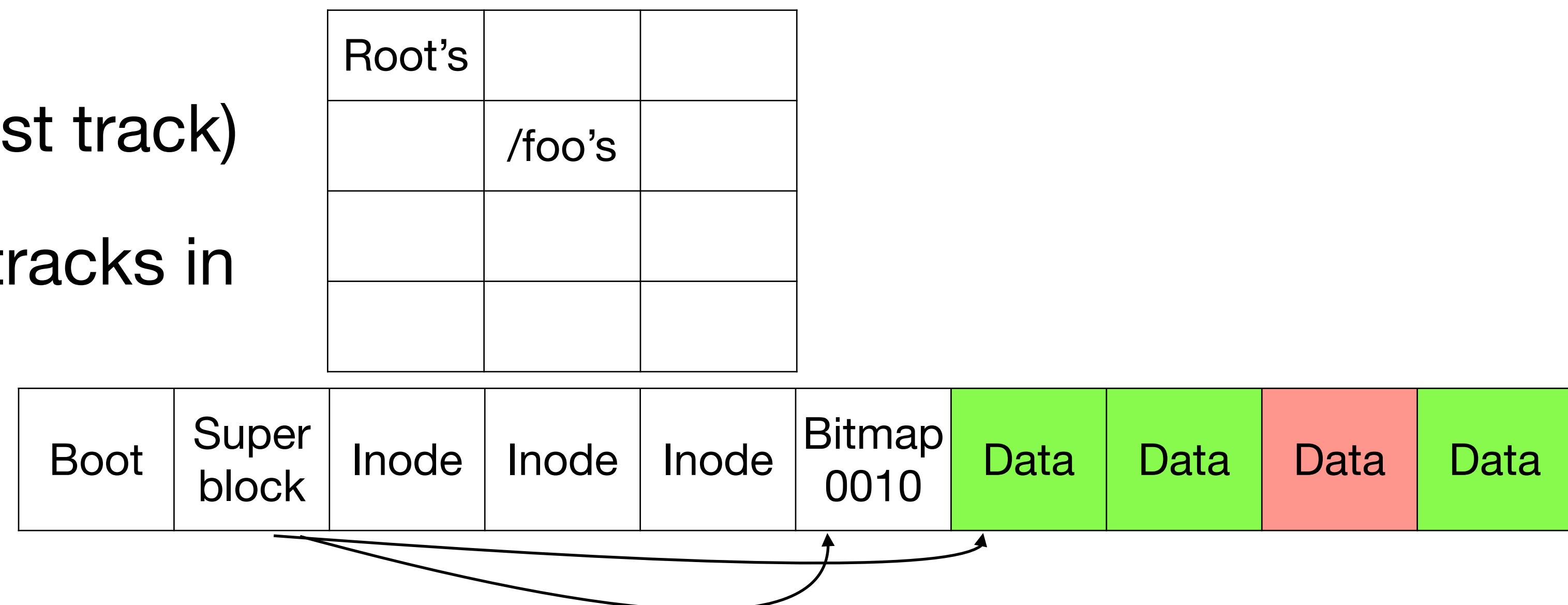
Locality problem

- Fast `ls`: just lookup inodes
- inodes and data blocks are far apart:
slow file reads

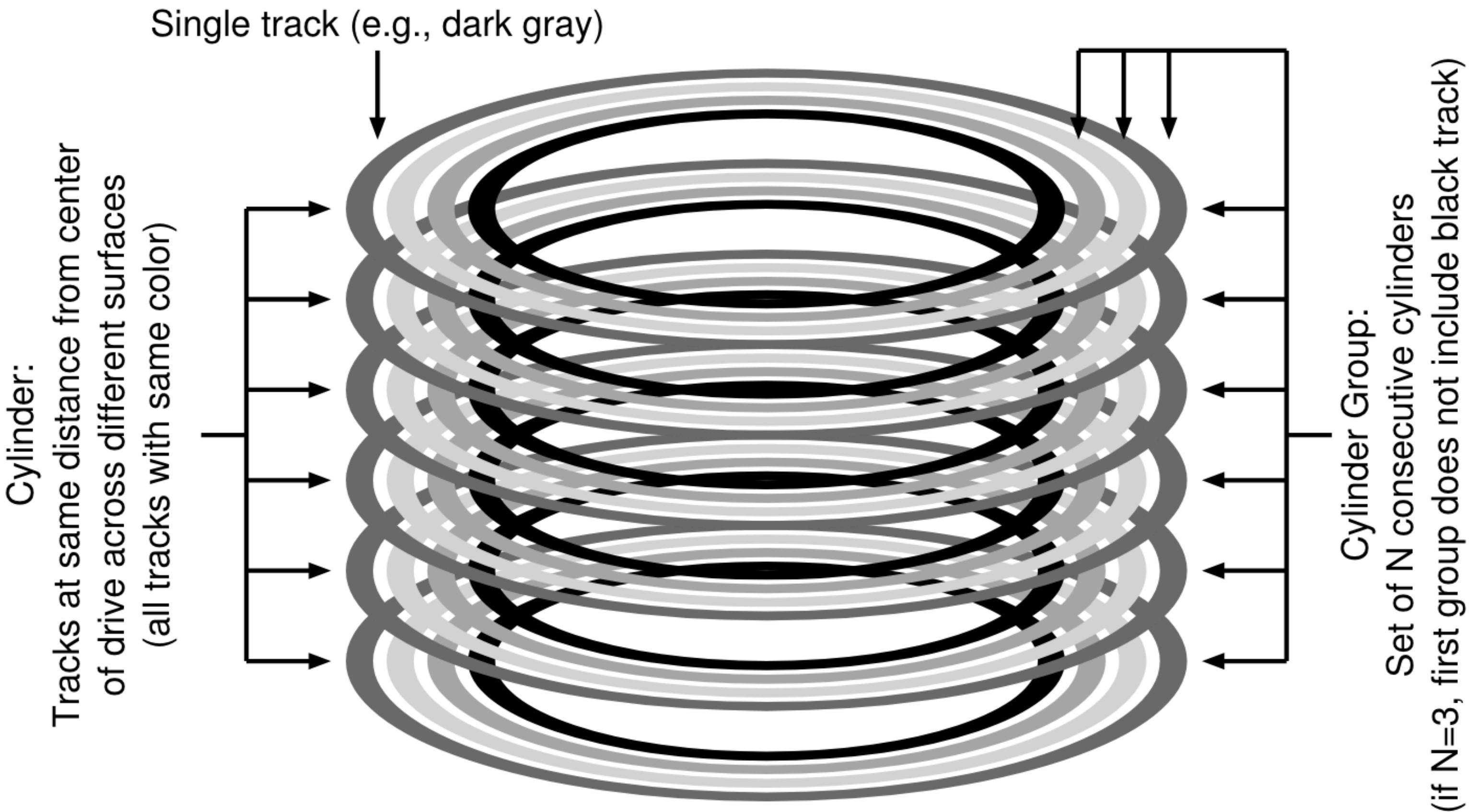


Locality problem

- Fast `ls`: just lookup inodes
- inodes and data blocks are far apart: slow file reads
- Back and forth disk head movement at write time:
 - inodes, bitmap (inner most track)
 - data blocks (outer most tracks in worst case)

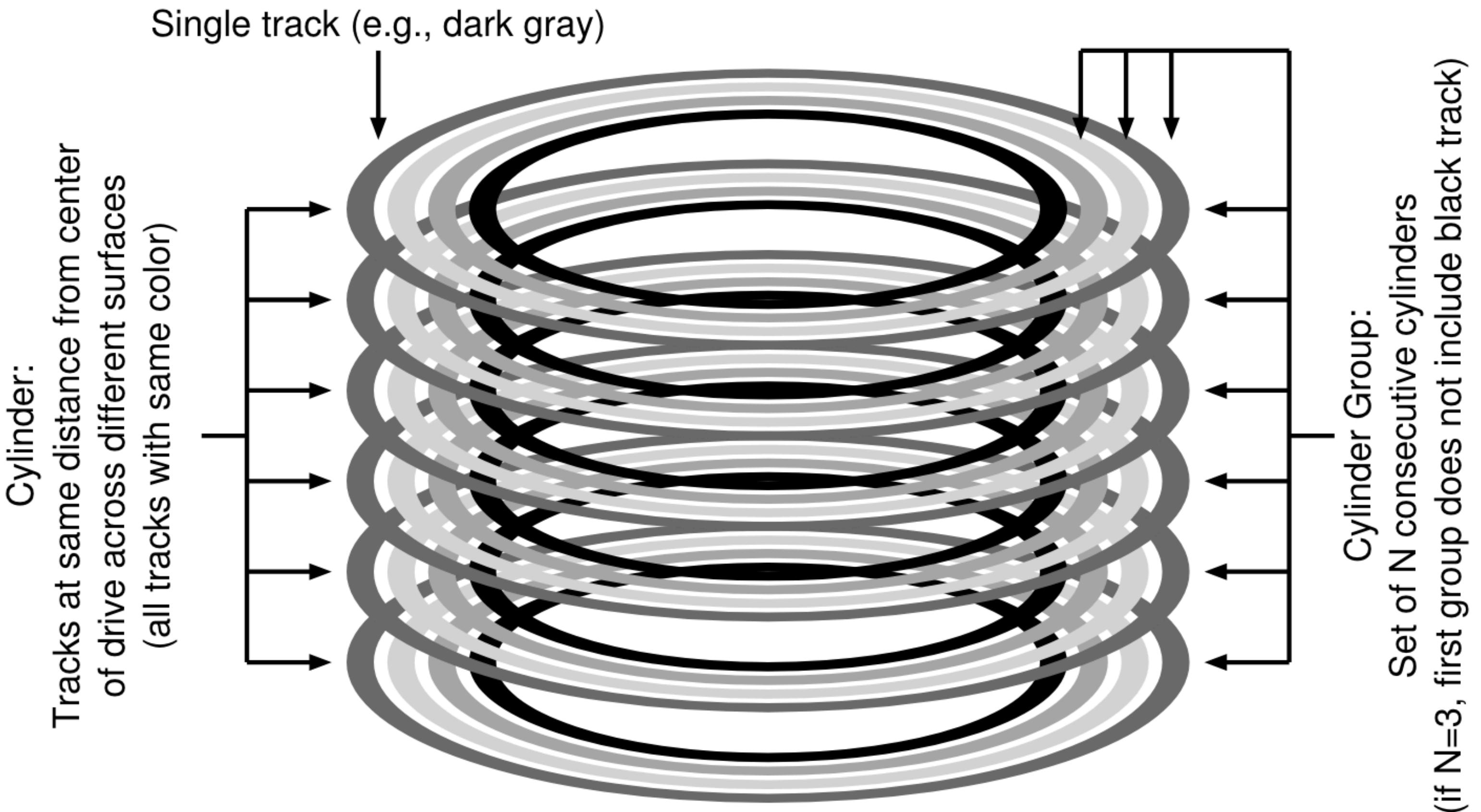


Break disk into locality groups



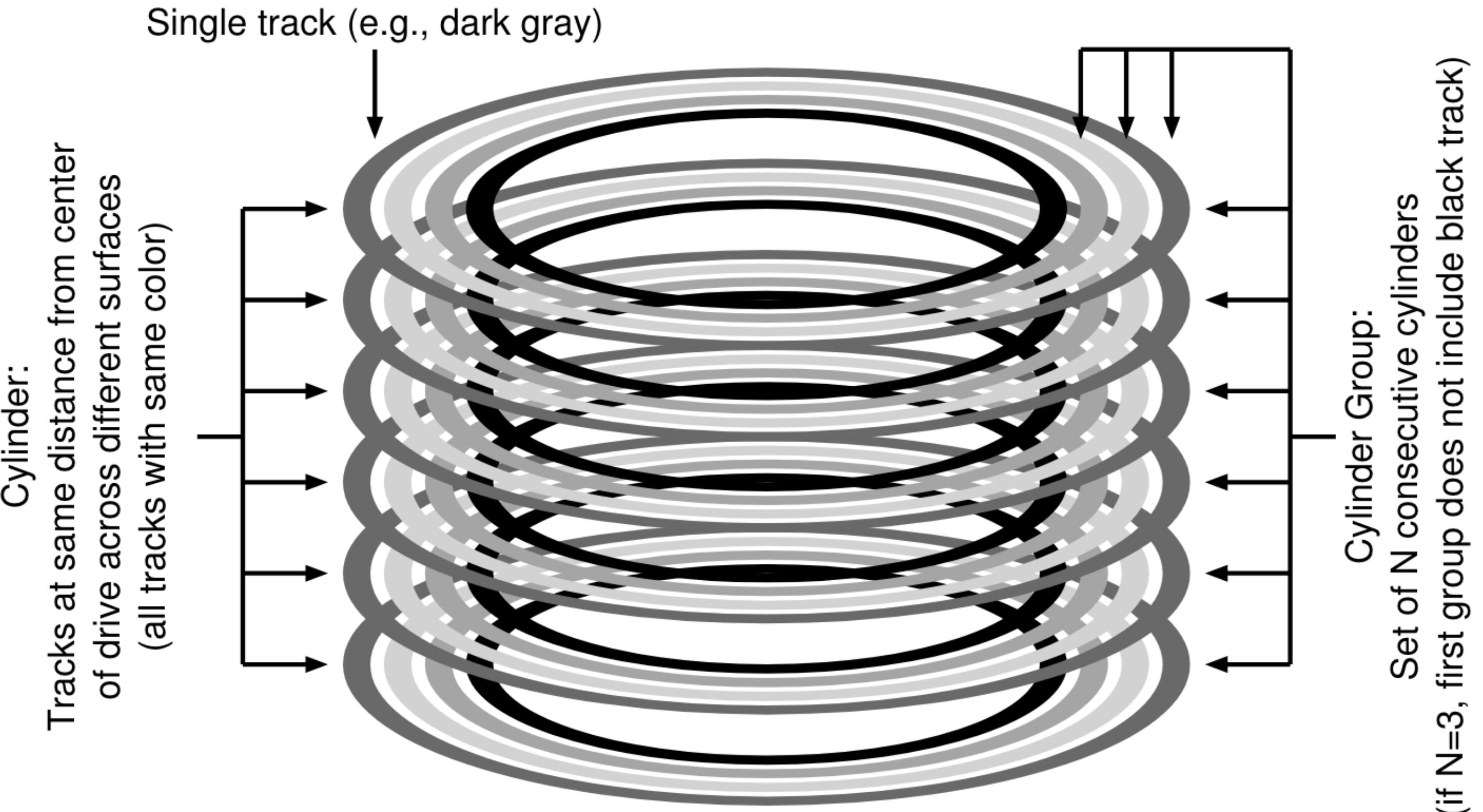
Break disk into locality groups

- Data on the same cylinder require no seek



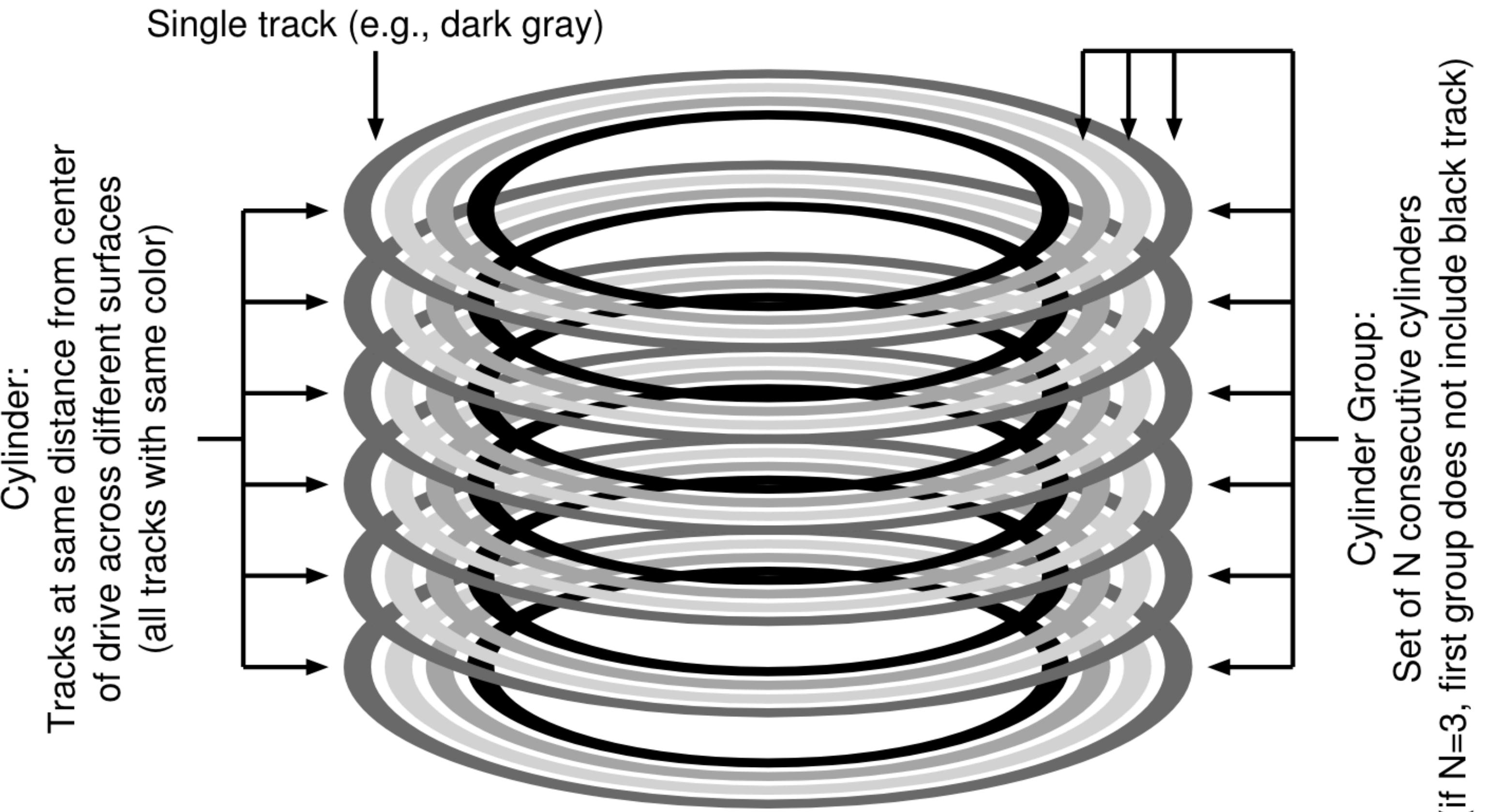
Break disk into locality groups

- Data on the same cylinder require no seek
- Break disk into “cylinder groups”



Break disk into locality groups

- Data on the same cylinder require no seek
- Break disk into “cylinder groups”
- Try to keep accesses limited within a group

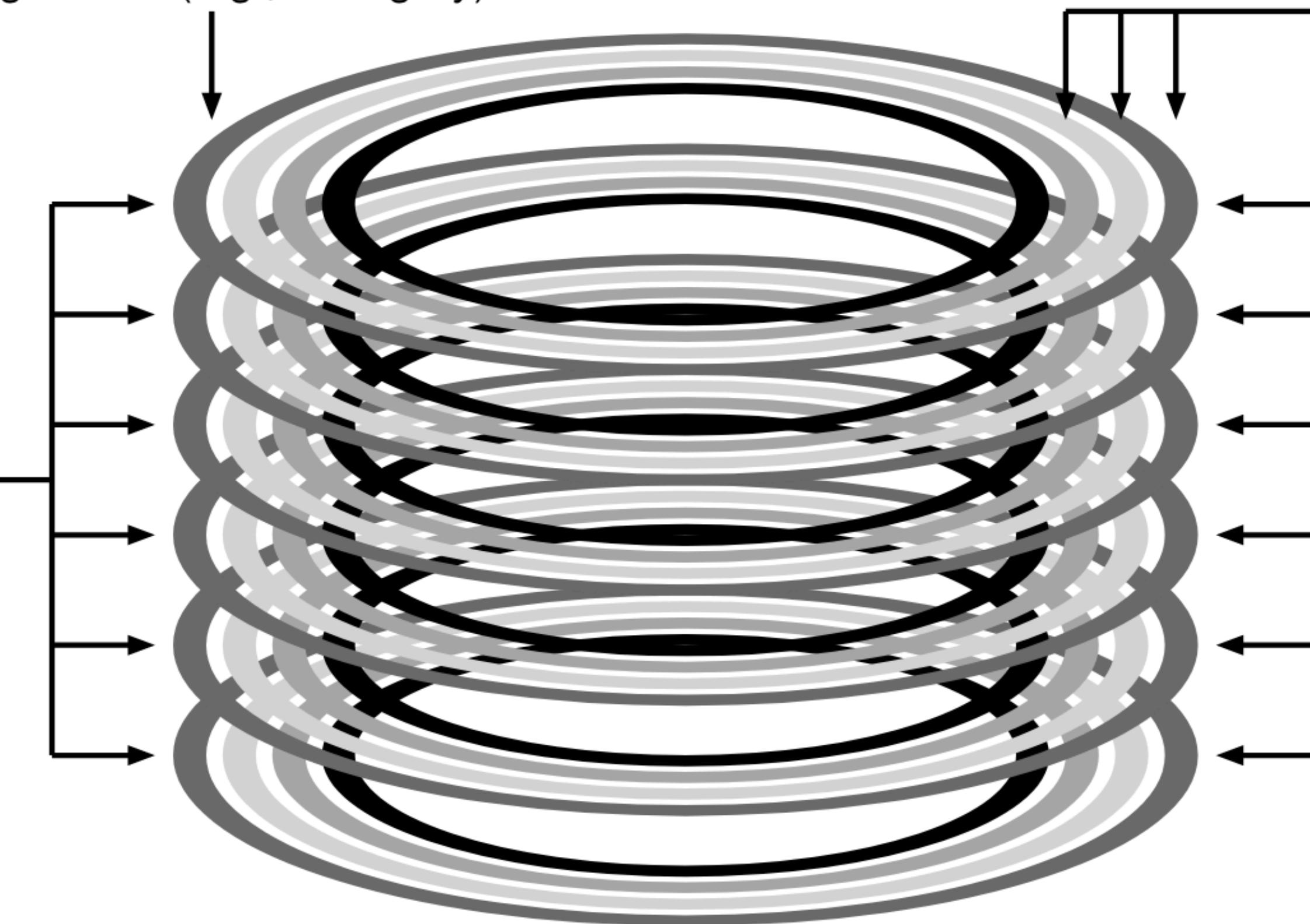


Break disk into locality groups

- Data on the same cylinder require no seek
 - Break disk into “cylinder groups”
 - Try to keep accesses limited within a group

Cylinder:
Tracks at same distance from center
of drive across different surfaces
(all tracks with same color)

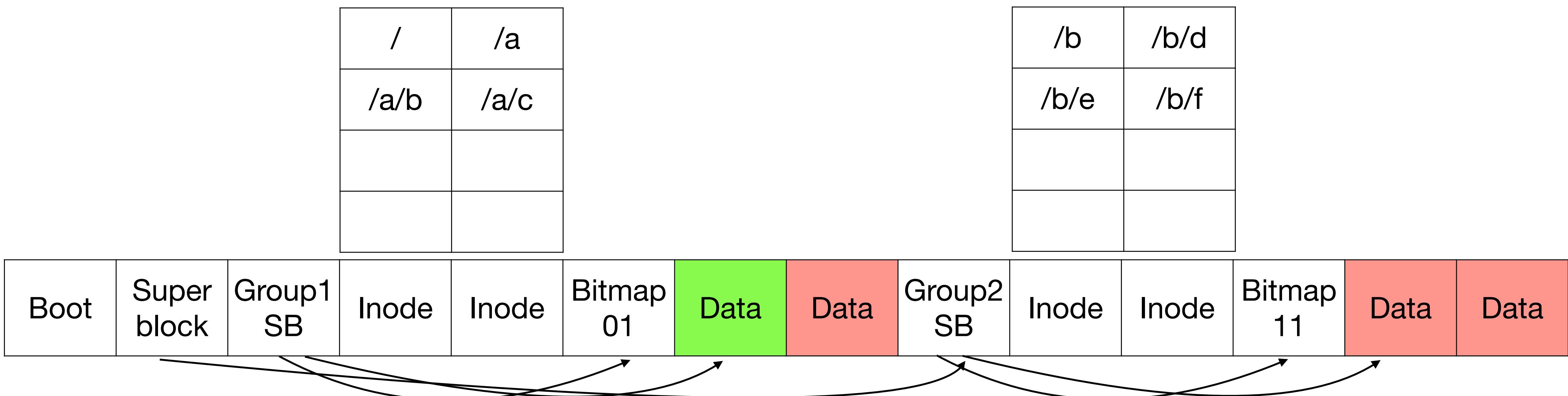
Single track (e.g., dark gray)



Cylinder Group:
Set of N consecutive cylinders
(if N=3, first group does not include black track)

Locality-aware file system

- Principle: Keep related stuff together



Greedy allocation

- Keep inode and data blocks of each file/directory in the same group.
- Allocate new files and directories on the most empty group

Greedy allocation

- Keep inode and data blocks of each file/directory in the same group.
- Allocate new files and directories on the most empty group
- /a, /a/c, /a/d, /a/e, /b, /b/f

Greedy allocation

- Keep inode and data blocks of each file/directory in the same group.
- Allocate new files and directories on the most empty group
- /a, /a/c, /a/d, /a/e, /b, /b/f

Group	inodes	Data blocks
1	/_____	/_____
2	a_____	a_____
3	b_____	b_____
4	c_____	cc_____
5	d_____	dd_____
6	e_____	ee_____
7	f_____	ff_____
8	_____	_____

Greedy allocation

- Keep inode and data blocks of each file/directory in the same group.
- Allocate new files and directories on the most empty group
- /a, /a/c, /a/d, /a/e, /b, /b/f
- Very slow
 - ls -l /a
 - Linking /a/c, /a/d, /a/e to create an executable

Group	inodes	Data blocks
1	/_____	/_____
2	a_____	a_____
3	b_____	b_____
4	c_____	cc_____
5	d_____	dd_____
6	e_____	ee_____
7	f_____	ff_____
8	_____	_____

Keeping related stuff together

- /a, /a/c, /a/d, /a/e
- /b, /b/f

Group	inodes	Data blocks
1	/_____	/_____
2	acde_____	accddee_____
3	bf_____	bff_____
4	_____	_____
5	_____	_____
6	_____	_____
7	_____	_____
8	_____	_____

What about large files?

- /a, /a/c, /a/d, /a/e
- /b, /b/f

Group	inodes	Data blocks
1	/_____	/_____
2	ac_____	acccccccccc
3	bf_____	bff_____
4	_____	ccccccccc_____
5	de_____	ddee_____
6	_____	_____
7	_____	_____
8	_____	_____

What about large files?

- /a, /a/c, /a/d, /a/e
- /b, /b/f
- /a/d, /a/e got separated from /a

Group	inodes	Data blocks
1	/_____	/_____
2	ac_____	acccccccccc
3	bf_____	bff_____
4	_____	ccccccccc_____
5	de_____	ddee_____
6	_____	_____
7	_____	_____
8	_____	_____

What about large files?

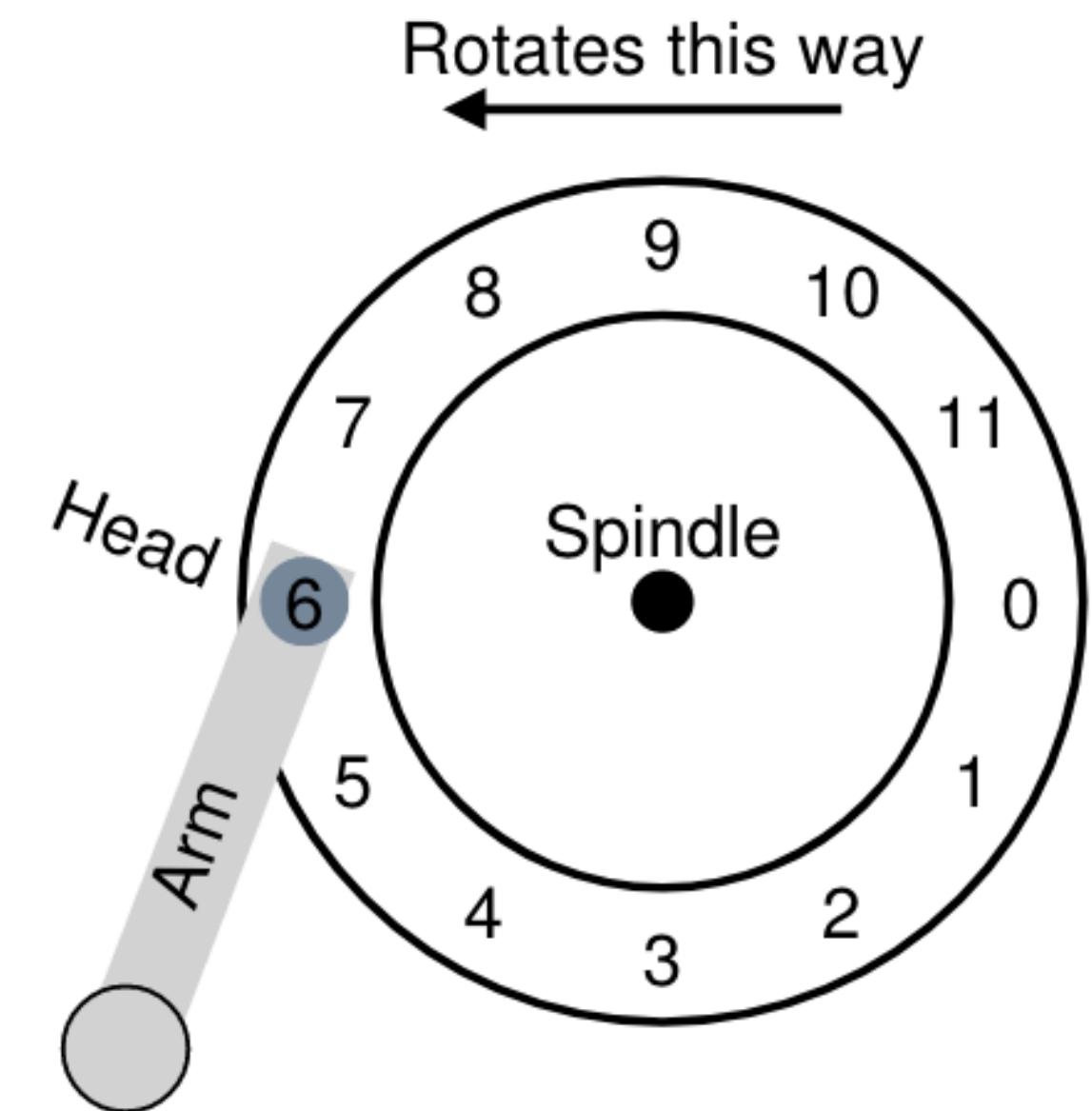
- /a, /a/c, /a/d, /a/e
- /b, /b/f
- Keep the data block of direct pointers within the group

Group	inodes	Data blocks
1	/_____	/_____
2	acde_____	accddee_____
3	bf_____	bff_____
4	_____	cccccccccccc
5	_____	ccccc_____
6	_____	_____
7	_____	_____
8	_____	_____

Disk cache

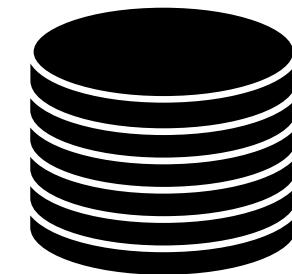
Disk rotation

- By the time, FS could give command to read block 7, disk has already rotated
 - Disk controller caches sectors on the entire track



Buffer cache

Write-through cache

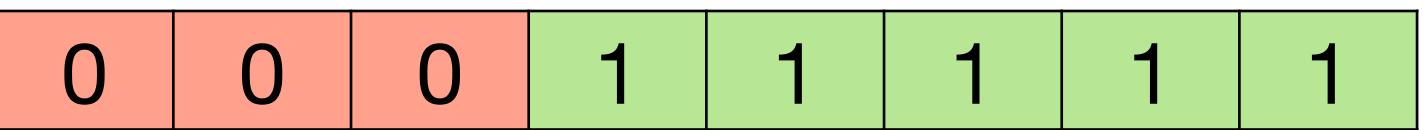


- Example: untar create and write 100 files

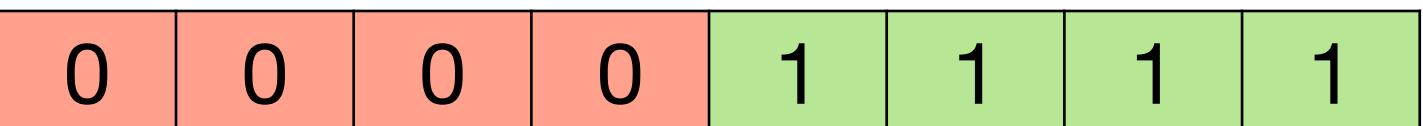
Directory data block

.	3

Inode bitmap

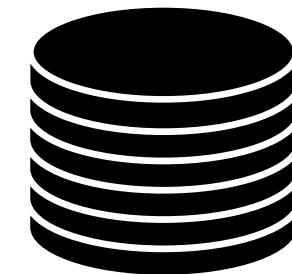


Data bitmap



Buffer cache

Write-through cache

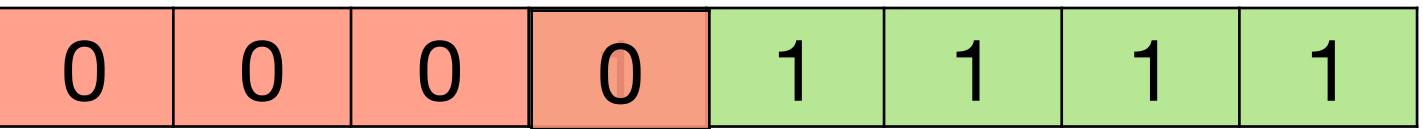


- Example: untar create and write 100 files

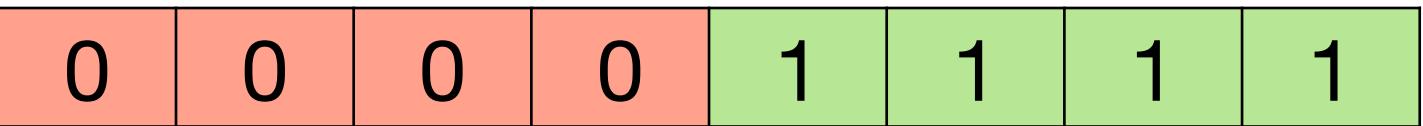
Directory data block

.	3
a.txt	4

Inode bitmap

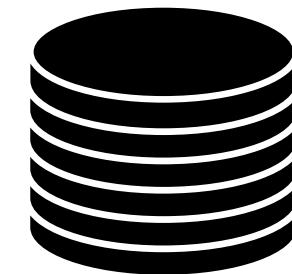


Data bitmap



Buffer cache

Write-through cache

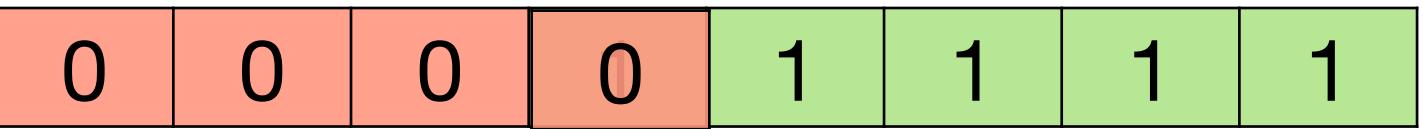


- Example: untar create and write 100 files

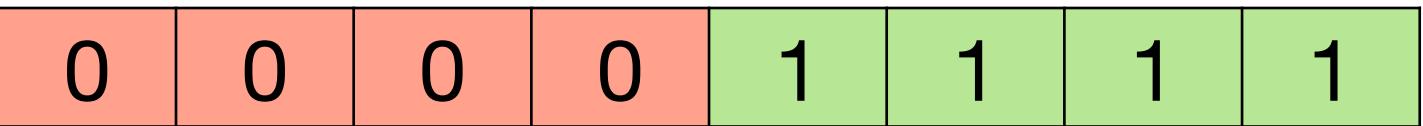
Directory data block

.	3
a.txt	4

Inode bitmap

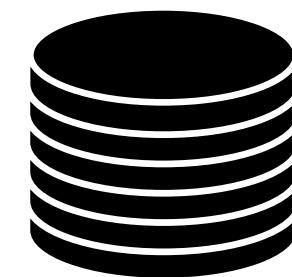


Data bitmap



Buffer cache

Write-through cache

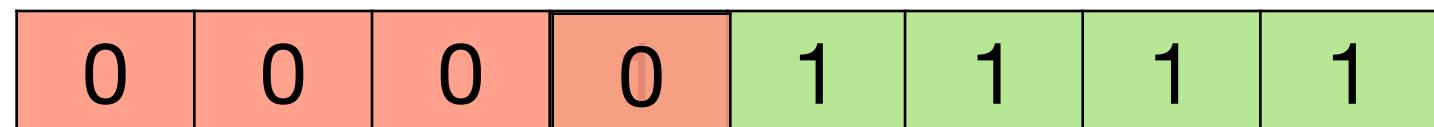


- Example: untar create and write 100 files

Directory data block

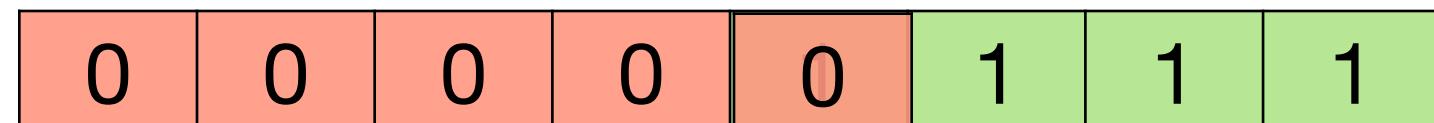
.	3
a.txt	4

Inode bitmap



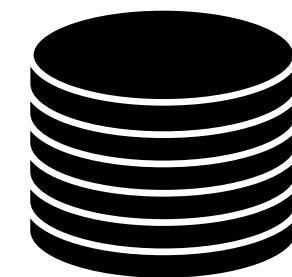
aaaa

Data bitmap



Buffer cache

Write-through cache

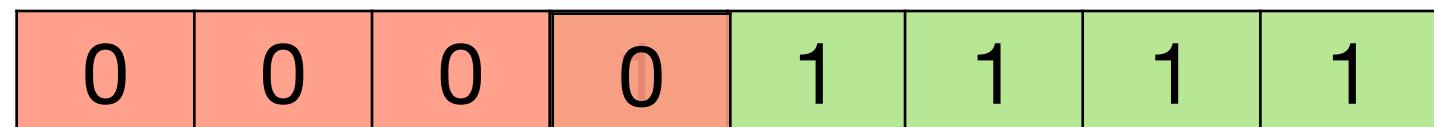


- Example: untar create and write 100 files

Directory data block

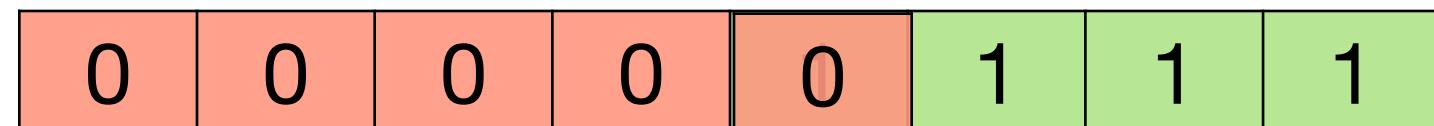
.	3
a.txt	4

Inode bitmap



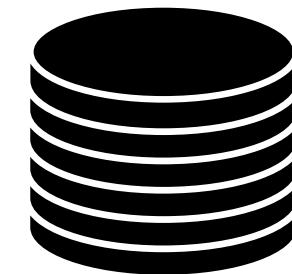
aaaa

Data bitmap



Buffer cache

Write-through cache

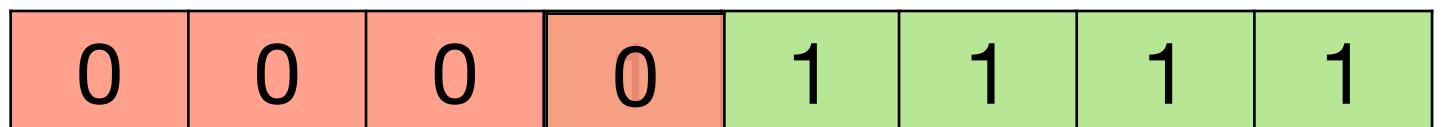


- Example: untar create and write 100 files

Directory data block

.	3
a.txt	4

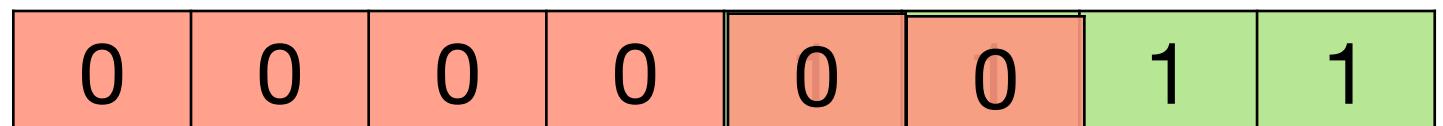
Inode bitmap



aaaa

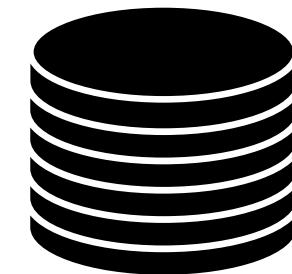
aaaa

Data bitmap



Buffer cache

Write-through cache

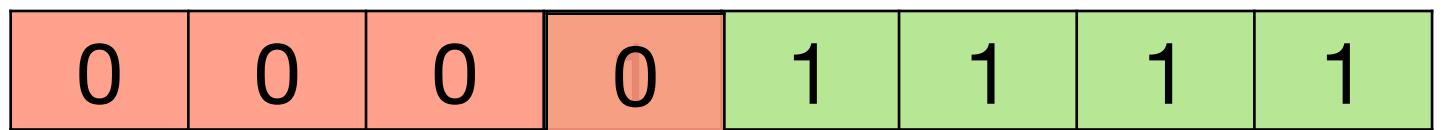


- Example: untar create and write 100 files

Directory data block

.	3
a.txt	4

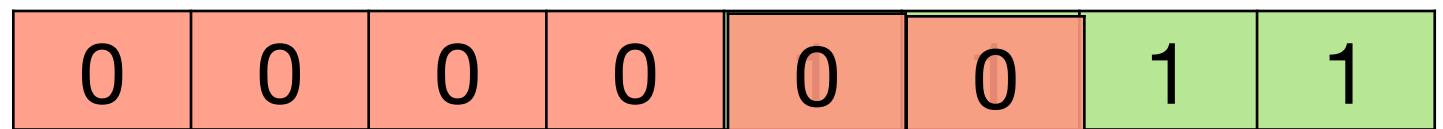
Inode bitmap



aaaa

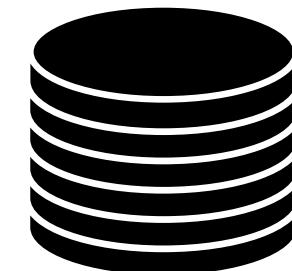
aaaa

Data bitmap



Buffer cache

Write-through cache

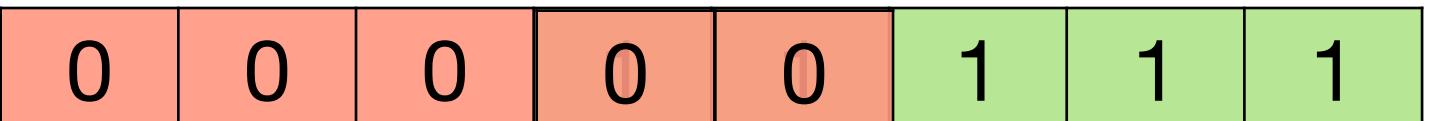


- Example: untar create and write 100 files

Directory data block

.	3
a.txt	4
b.txt	5

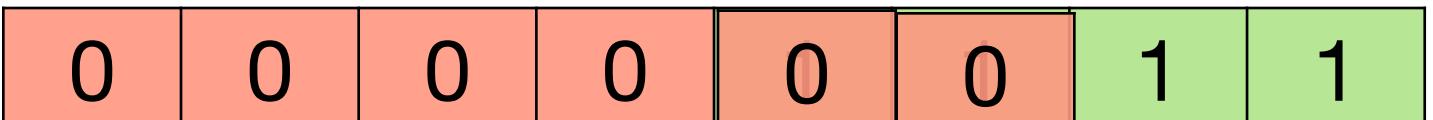
Inode bitmap



aaaa

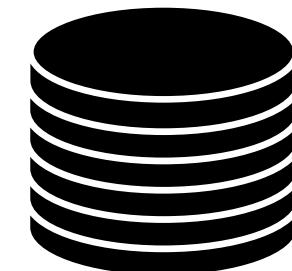
aaaa

Data bitmap



Buffer cache

Write-through cache

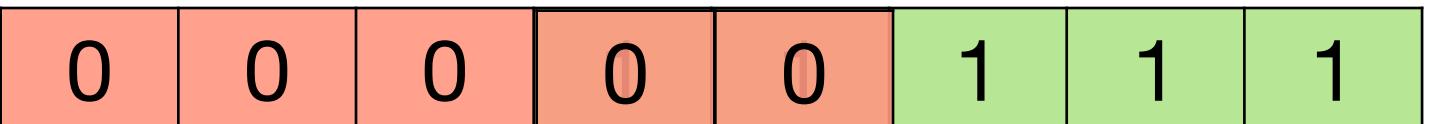


- Example: untar create and write 100 files

Directory data block

.	3
a.txt	4
b.txt	5

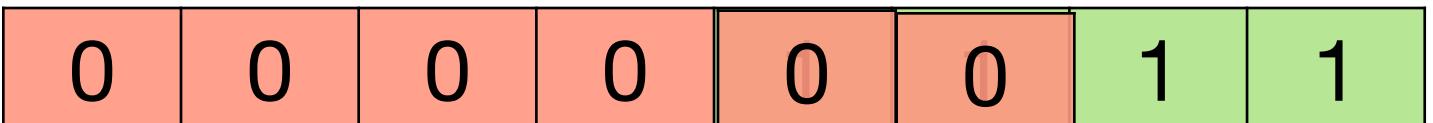
Inode bitmap



aaaa

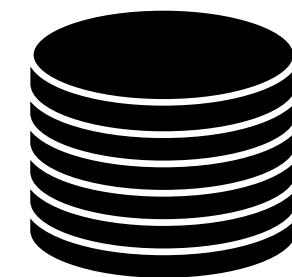
aaaa

Data bitmap



Buffer cache

Write-through cache

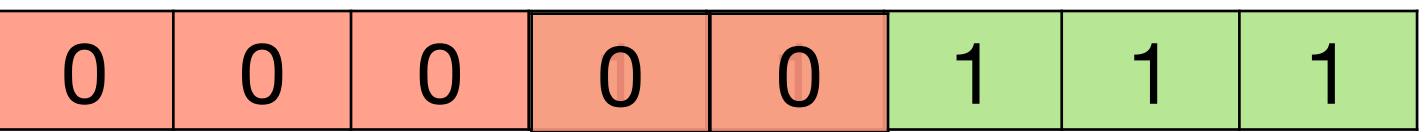


- Example: untar create and write 100 files

Directory data block

.	3
a.txt	4
b.txt	5

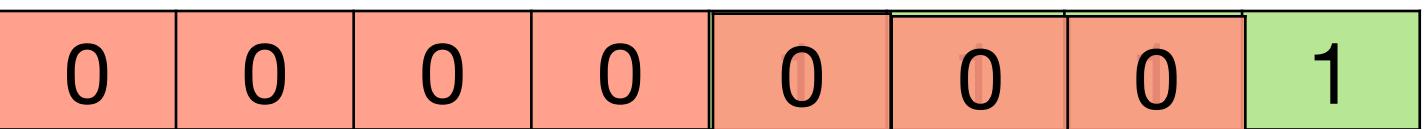
Inode bitmap



aaaa

aaaa

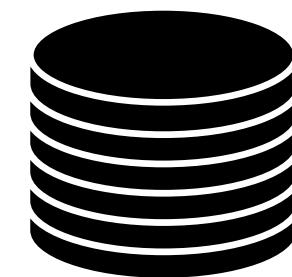
Data bitmap



bbbb

Buffer cache

Write-through cache

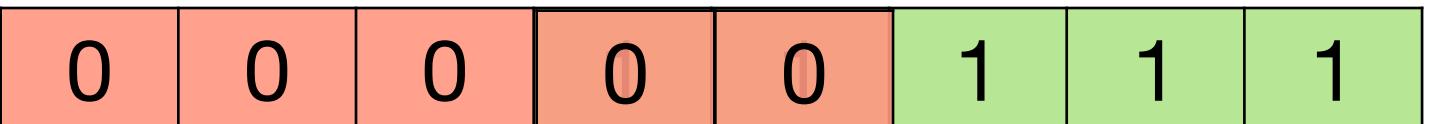


- Example: untar create and write 100 files

Directory data block

.	3
a.txt	4
b.txt	5

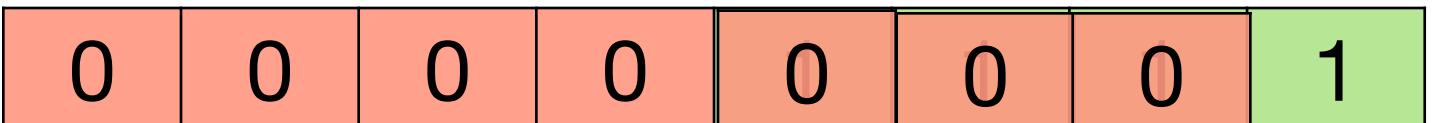
Inode bitmap



aaaa

aaaa

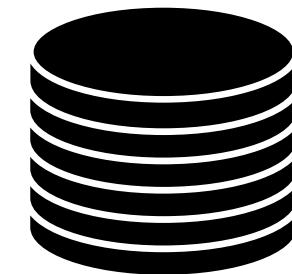
Data bitmap



bbbb

Buffer cache

Write-through cache

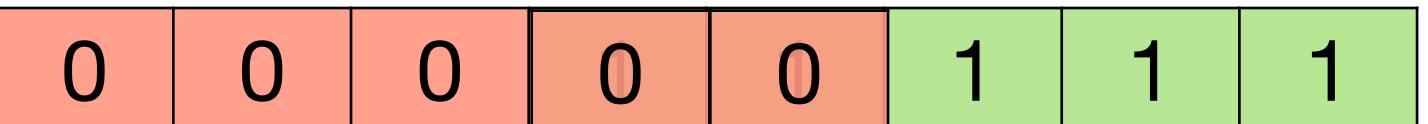


- Example: untar create and write 100 files

Directory data block

.	3
a.txt	4
b.txt	5

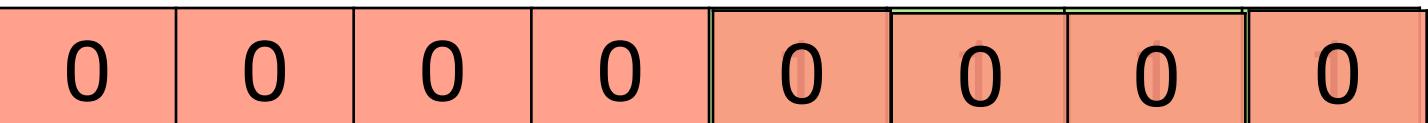
Inode bitmap



aaaa

aaaa

Data bitmap

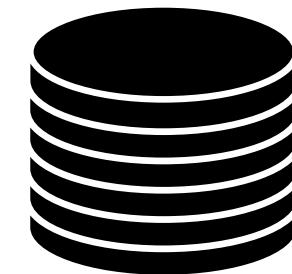


bbbb

bbbb

Buffer cache

Write-through cache

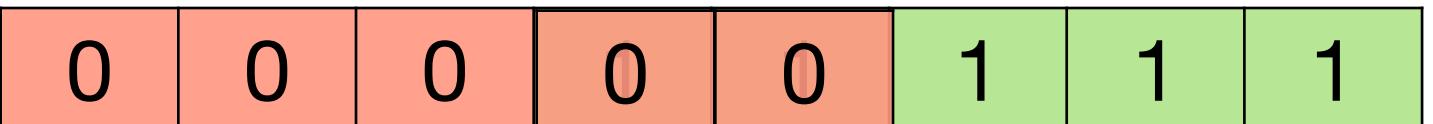


- Example: untar create and write 100 files

Directory data block

.	3
a.txt	4
b.txt	5

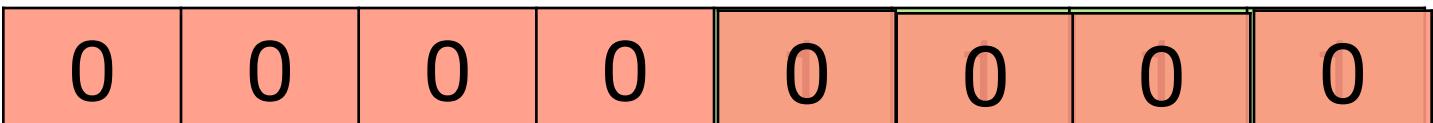
Inode bitmap



aaaa

aaaa

Data bitmap

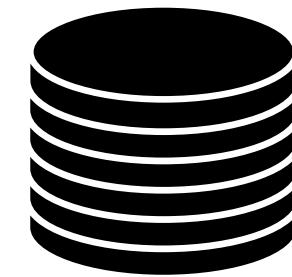


bbbb

bbbb

Buffer cache

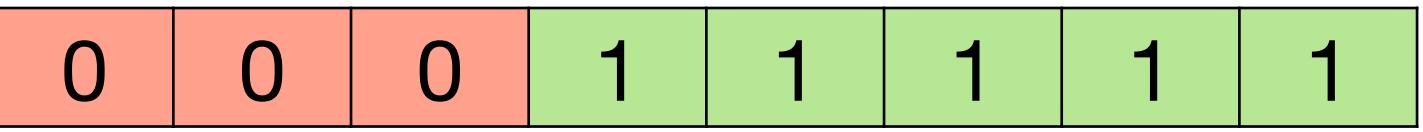
Write-back cache



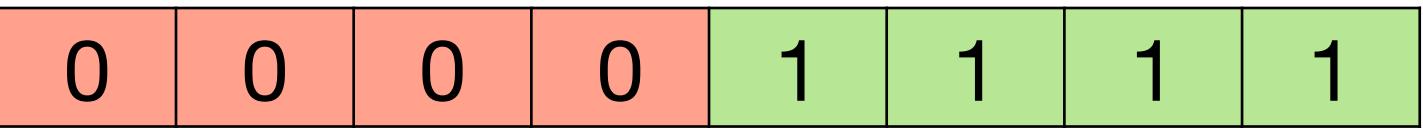
Directory data block

.	3

Inode bitmap

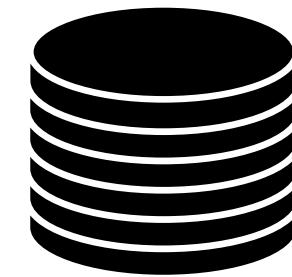


Data bitmap



Buffer cache

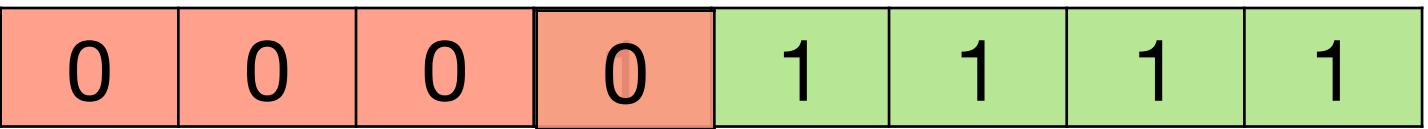
Write-back cache



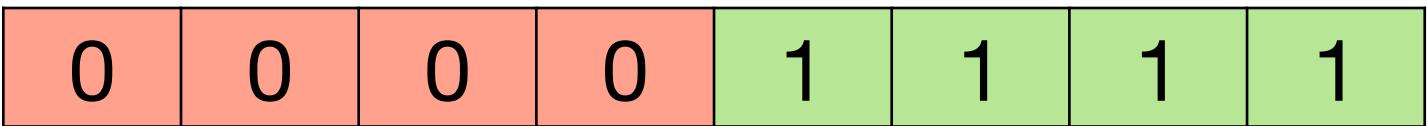
Directory data block

.	3
a.txt	4

Inode bitmap

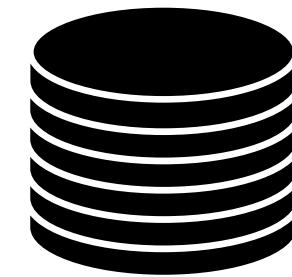


Data bitmap



Buffer cache

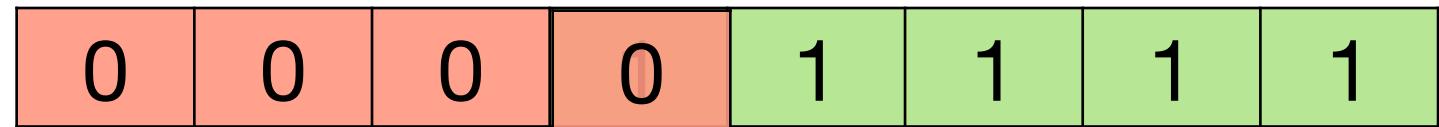
Write-back cache



Directory data block

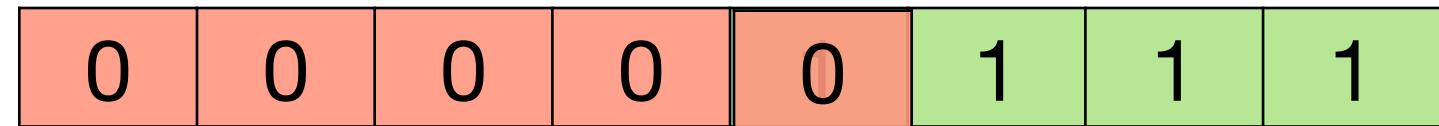
.	3
a.txt	4

Inode bitmap



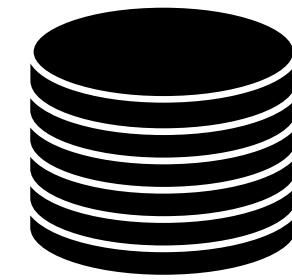
aaaa

Data bitmap



Buffer cache

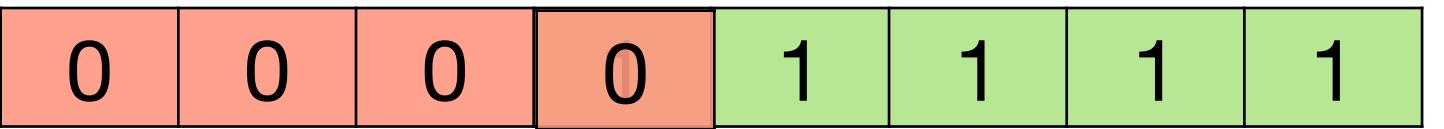
Write-back cache



Directory data block

.	3
a.txt	4

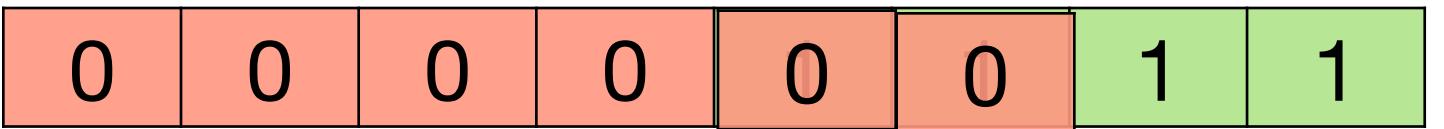
Inode bitmap



aaaa

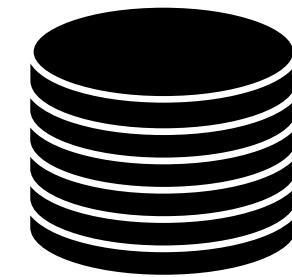
aaaa

Data bitmap



Buffer cache

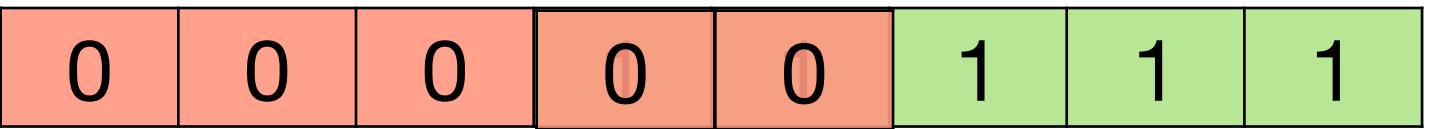
Write-back cache



Directory data block

.	3
a.txt	4
b.txt	5

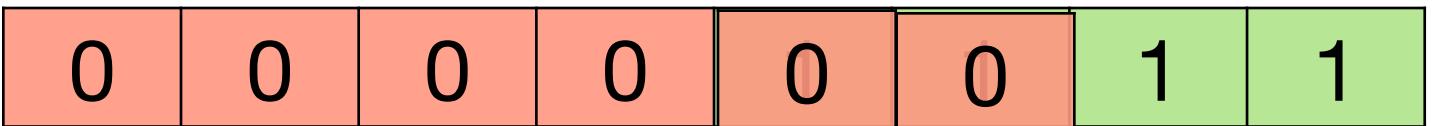
Inode bitmap



aaaa

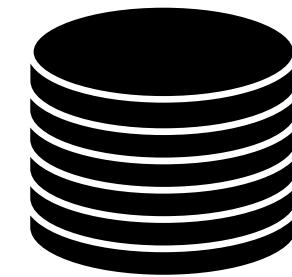
aaaa

Data bitmap



Buffer cache

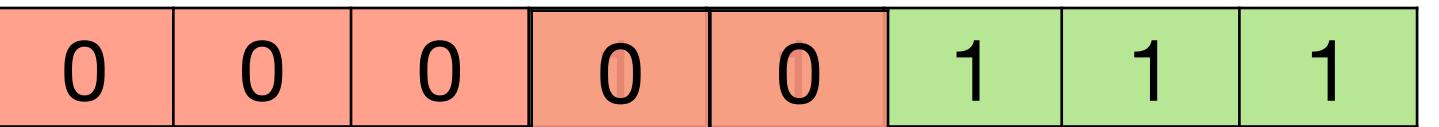
Write-back cache



Directory data block

.	3
a.txt	4
b.txt	5

Inode bitmap



aaaa

aaaa

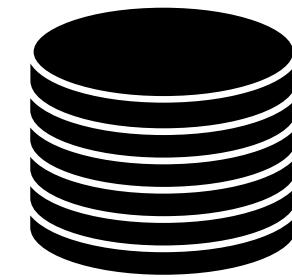
Data bitmap



bbbb

Buffer cache

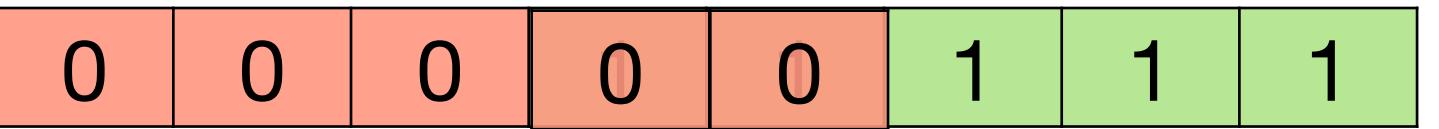
Write-back cache



Directory data block

.	3
a.txt	4
b.txt	5

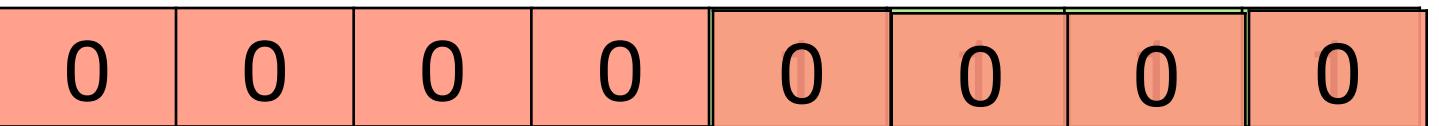
Inode bitmap



aaaa

aaaa

Data bitmap

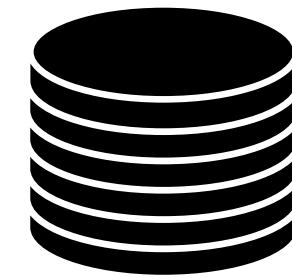


bbbb

bbbb

Buffer cache

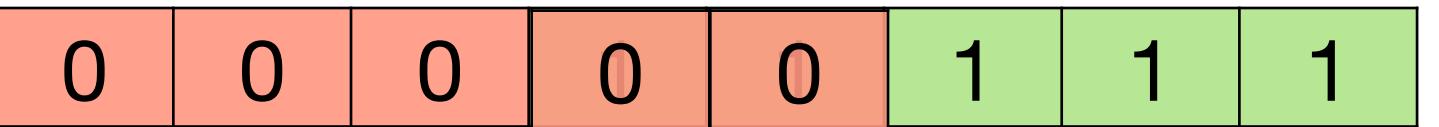
Write-back cache



Directory data block

.	3
a.txt	4
b.txt	5

Inode bitmap



aaaa

aaaa

Data bitmap



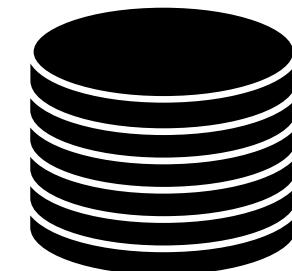
bbbb

bbbb

Buffer cache

Write-back cache

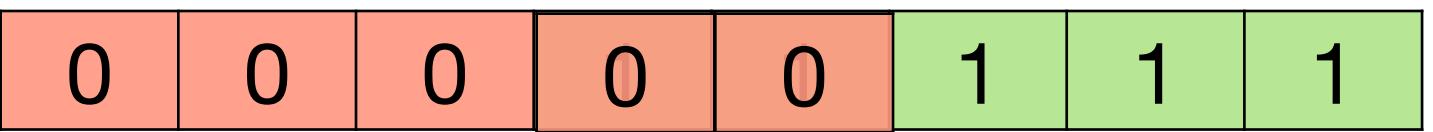
- Absorb multiple writes into single write
- Better disk scheduling opportunity



Directory data block

.	3
a.txt	4
b.txt	5

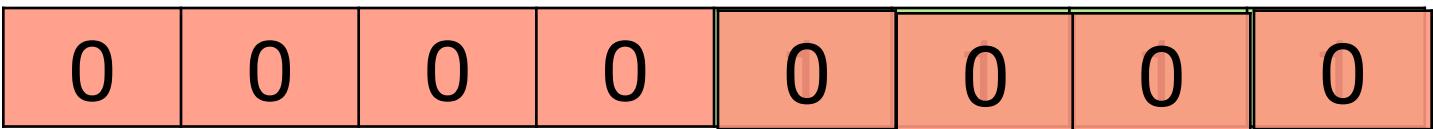
Inode bitmap



aaaa

aaaa

Data bitmap



bbbb

bbbb

Crash consistency

OSTEP Ch.42

Writing a file

Example: /foo/bar



Inode = 1 “/“

Type = directory
Size
2

Block 2

File/directory name	Inode number
.	1
foo	8

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Inode = 8 “/foo“

Type = directory
Size
16

Block 16

File/directory name	Inode number
.	8
..	1

```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar

Inode = 1 “/“

Type = directory
Size
2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo“

Type = directory
Size
16

Block 16

File/directory name	Inode number
.	8
..	1

Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar



Inode = 1 “/“

Type = directory	
Size	
1	2

Block 2

File/directory name	Inode number
.	1
foo	8

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Inode = 8 “/foo“

Type = directory	
Size	
16	

Block 16

File/directory name	Inode number
.	8
..	1

```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar

Inode = 1 “/“

Type = directory	
Size	
1	2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo“

Type = directory	
Size	
16	

Block 16

File/directory name	Inode number
.	8
..	1

Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar



Inode = 1 “/“

Block 2

File/directory name	Inode number
.	1
foo	8

Type = directory	
Size	
2	2

1 2

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Inode = 8 “/foo“

Block 16

File/directory name	Inode number
.	8
..	1

Type = directory	
Size	
16	16

3 16

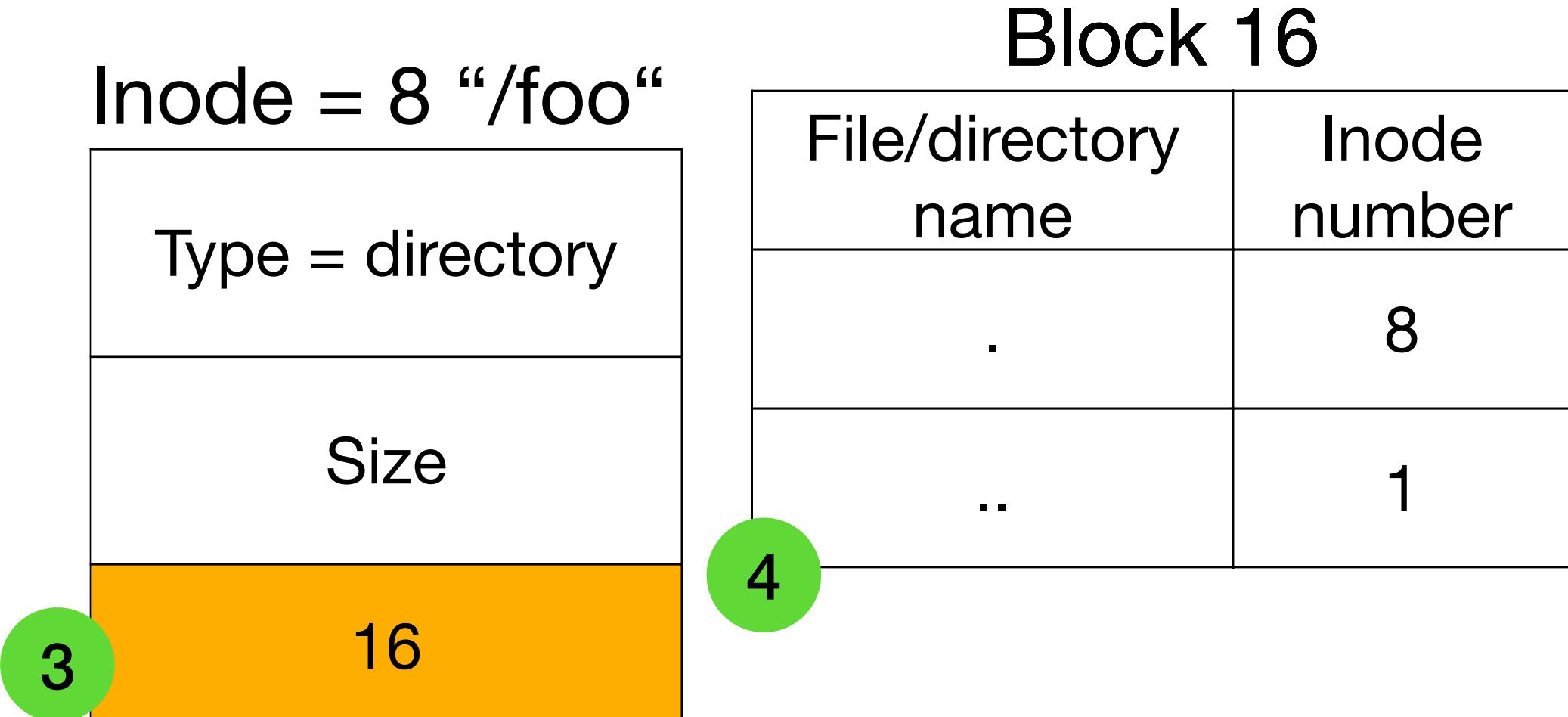
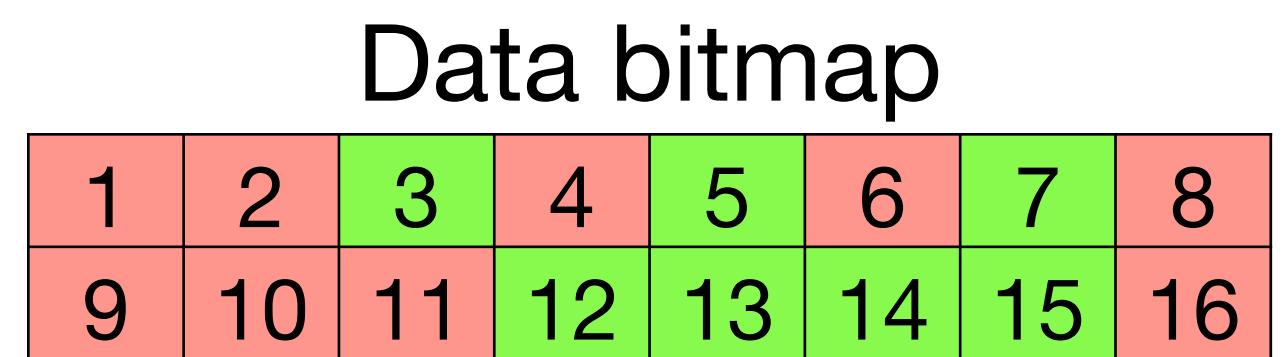
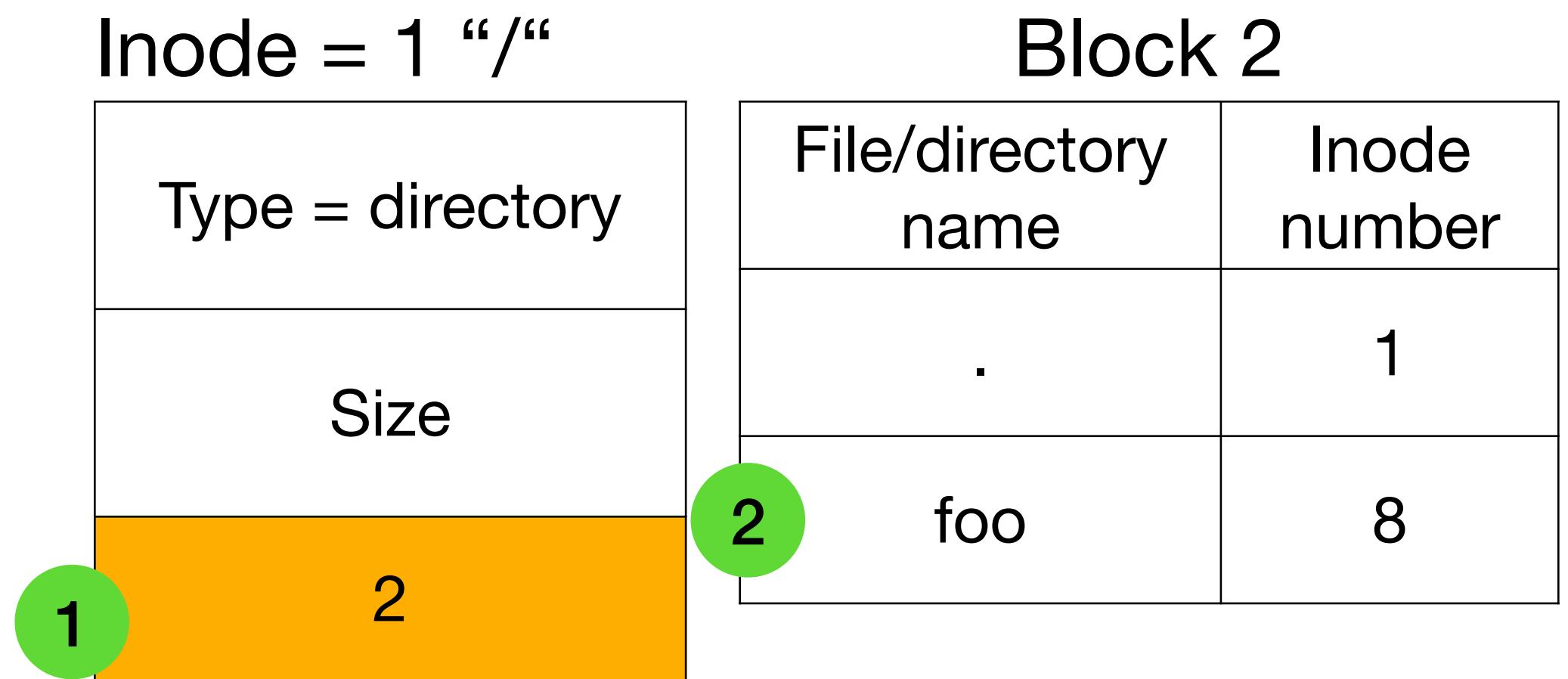
```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar



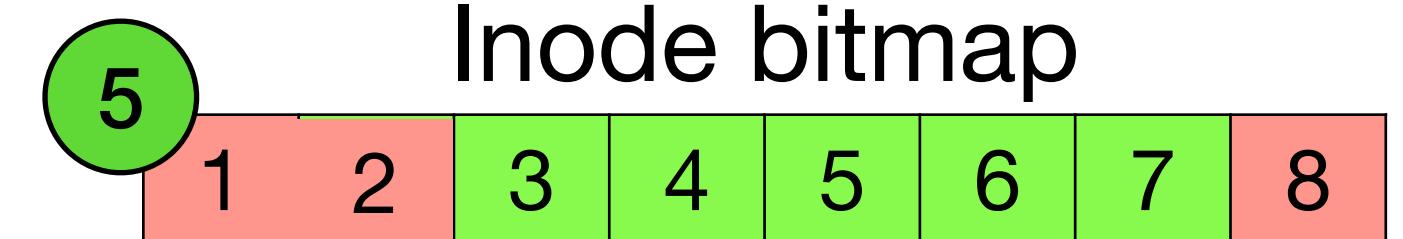
```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```

Writing a file

Example: /foo/bar



Inode = 1 “/“

Type = directory	
Size	
2	2

Block 2

File/directory name	Inode number
.	1
foo	8

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Inode = 8 “/foo“

Type = directory	
Size	
16	16

Block 16

File/directory name	Inode number
.	8
..	1

Writing a file

Example: /foo/bar

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

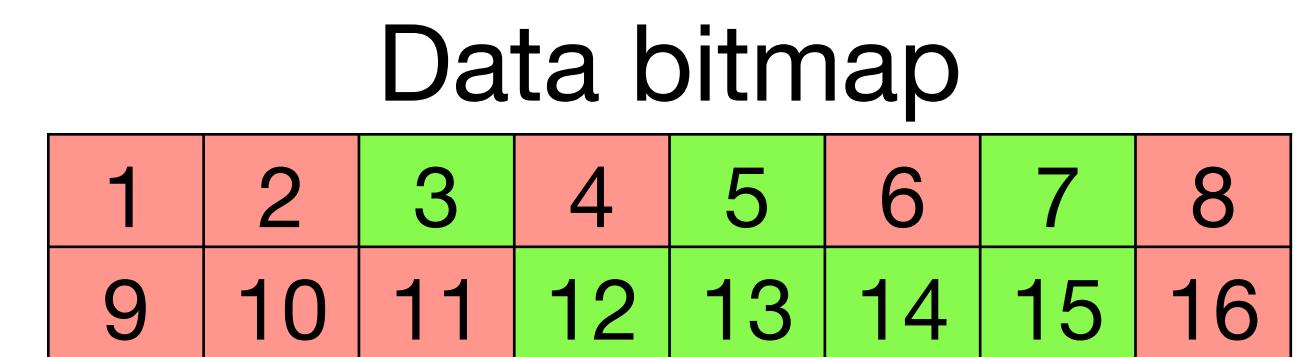


1	Inode = 1 “/“
	Type = directory
	Size
2	2

Block 2	
File/directory name	Inode number
.	1
foo	8

3	Inode = 8 “/foo“
	Type = directory
	Size
4	16

Block 16	
File/directory name	Inode number
.	8
..	1



Writing a file

Example: /foo/bar

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time

```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Inode = 1 “/”

Type = directory	2
Size	
1	2

Block 2

File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”

Type = directory	4
Size	
3	16

Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Writing a file

Example: /foo/bar

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time

```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Inode = 1 “/“

Type = directory	2
Size	
1	2

Block 2

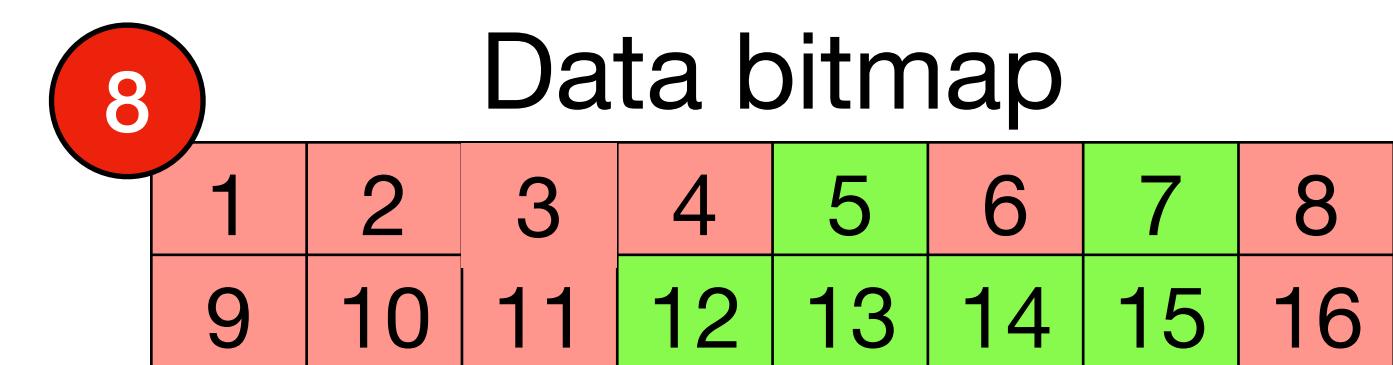
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo“

Type = directory	4
Size	
3	16

Block 16

File/directory name	Inode number
.	8
..	1
bar	2



Writing a file

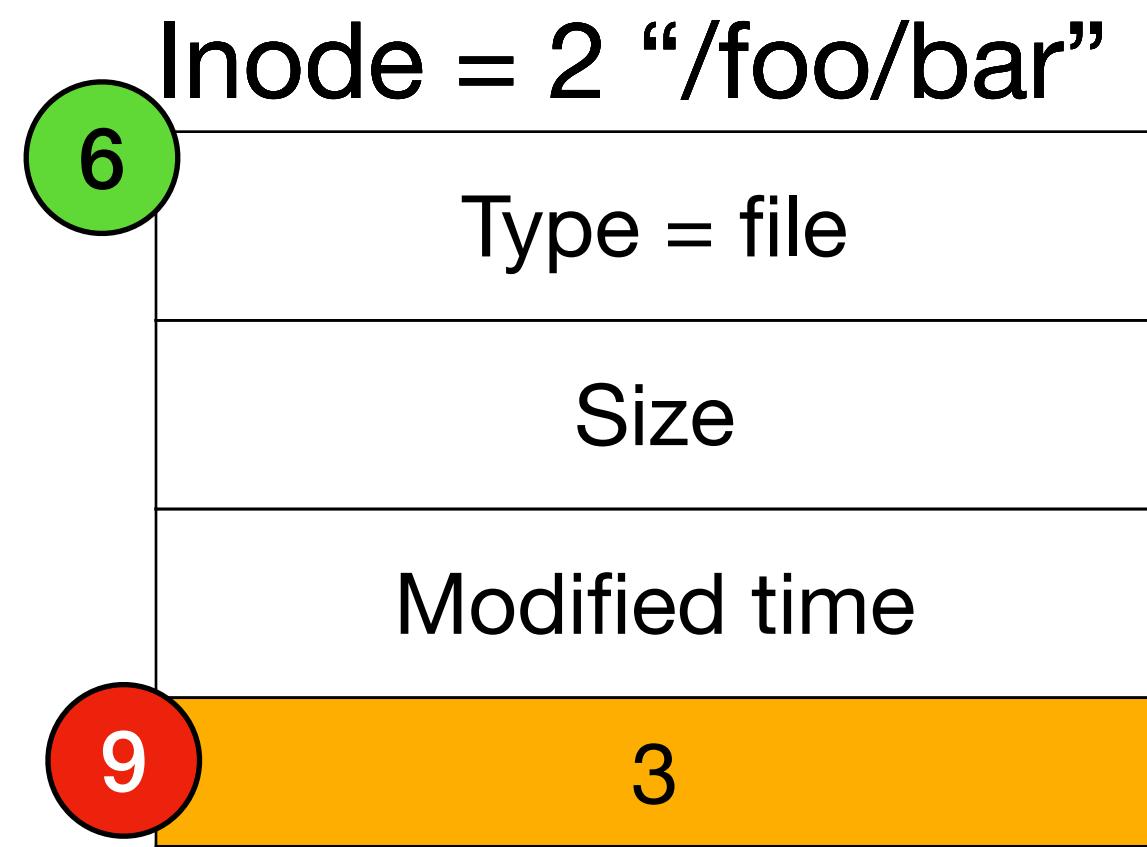
Example: /foo/bar

Inode = 1 “/”
Type = directory
Size
2

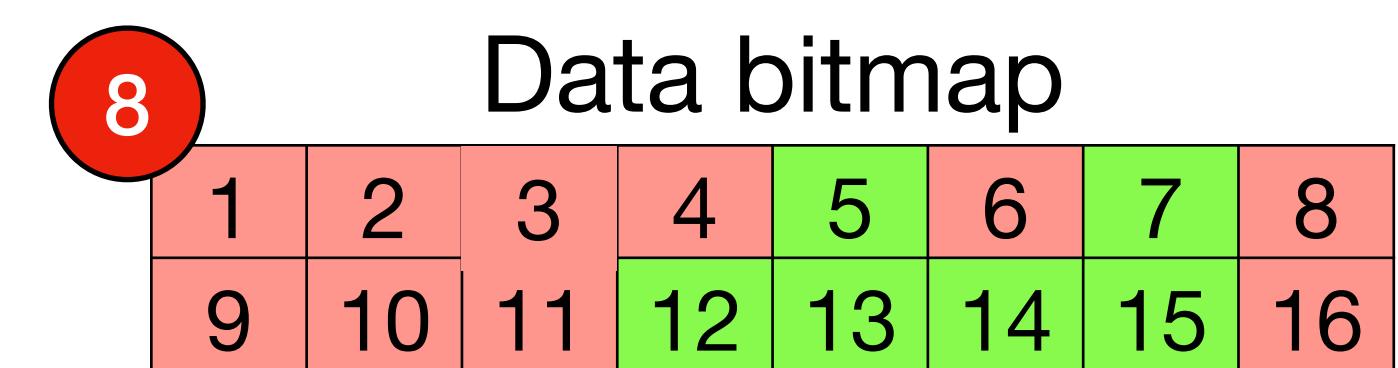
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”
Type = directory
Size
16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```



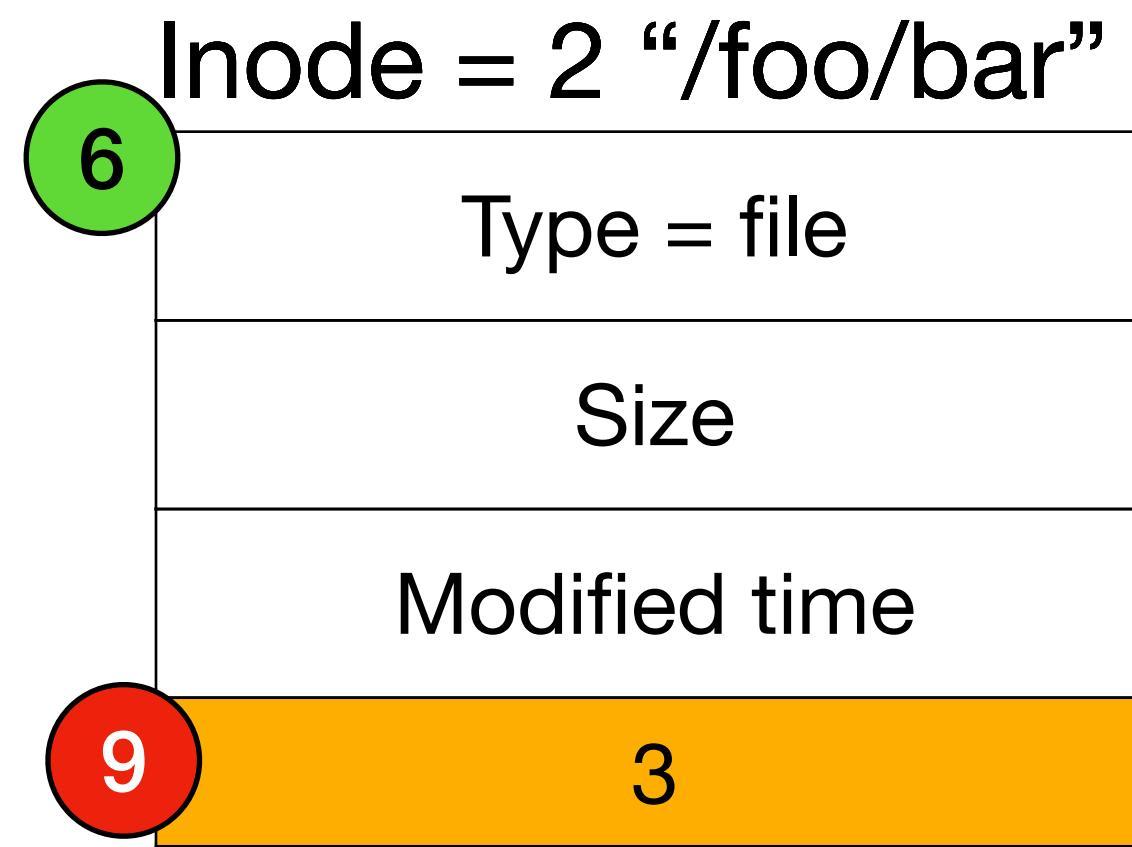
Writing a file

Example: /foo/bar

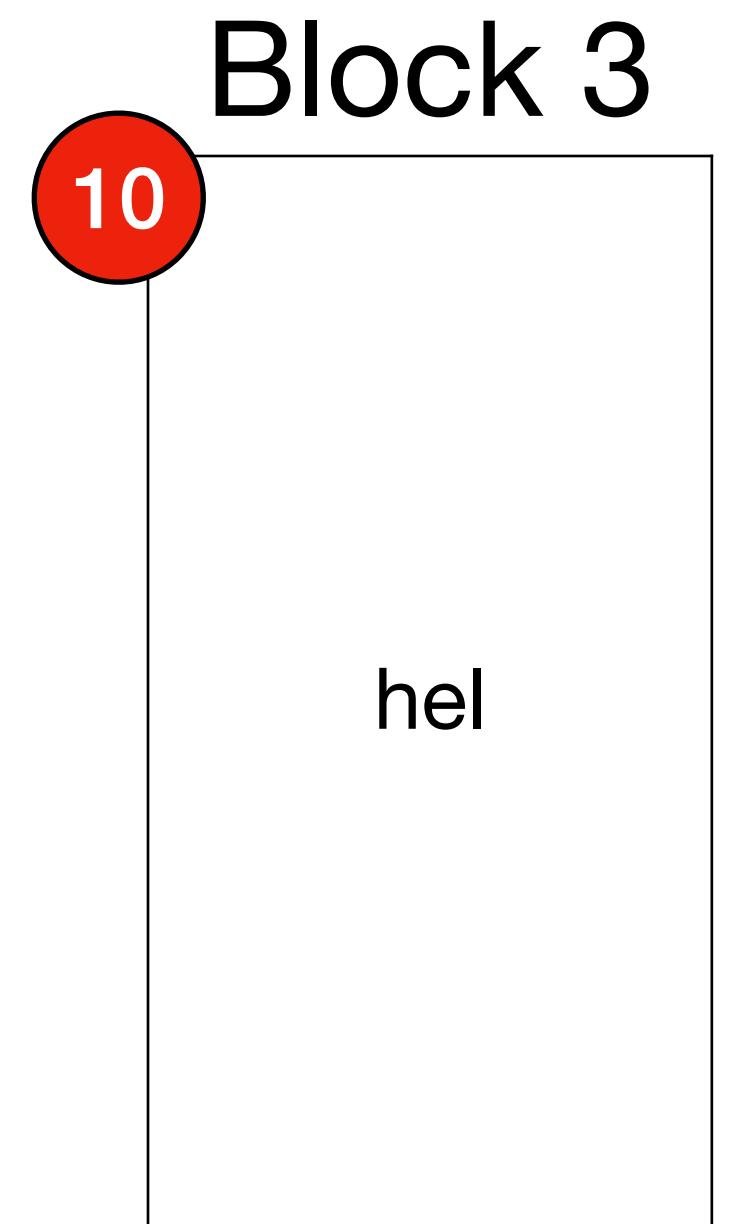
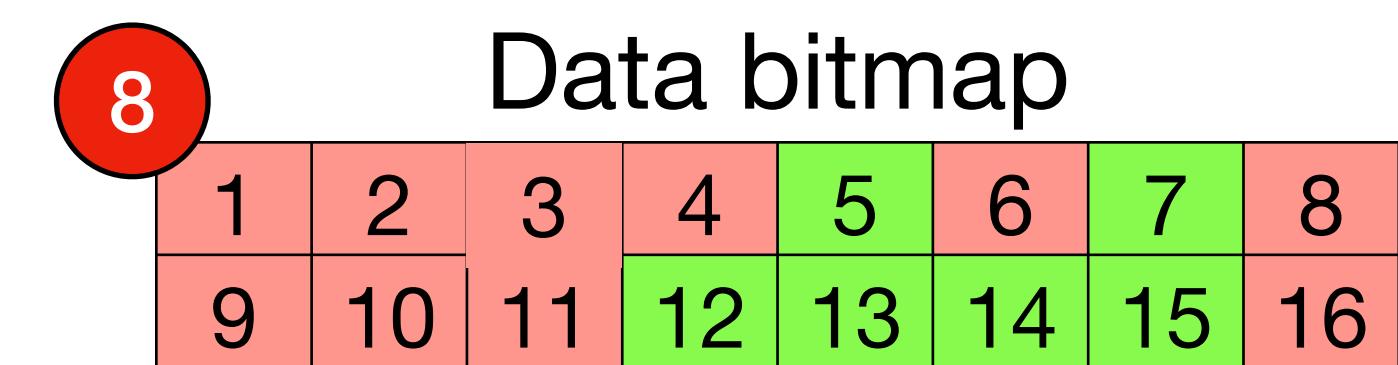
Inode = 1 “/”	
Type = directory	
Size	
2	

Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

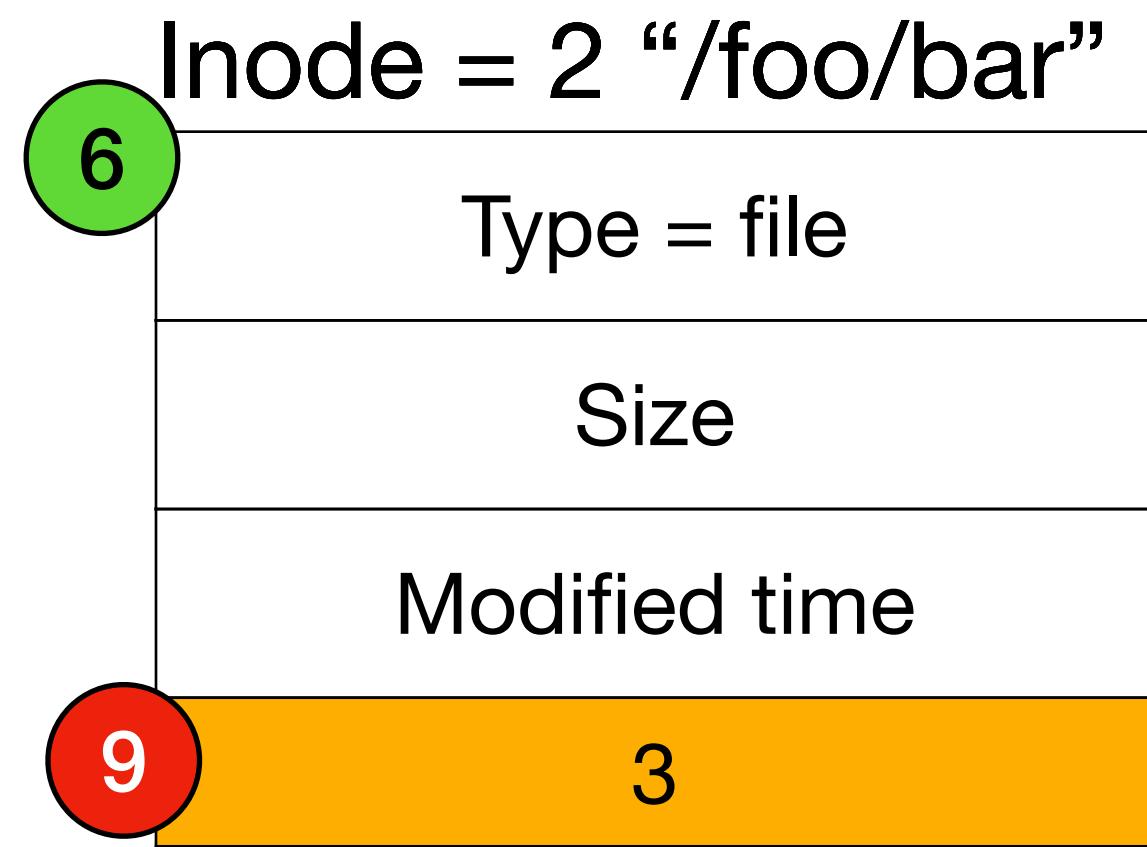
Writing a file

Example: /foo/bar

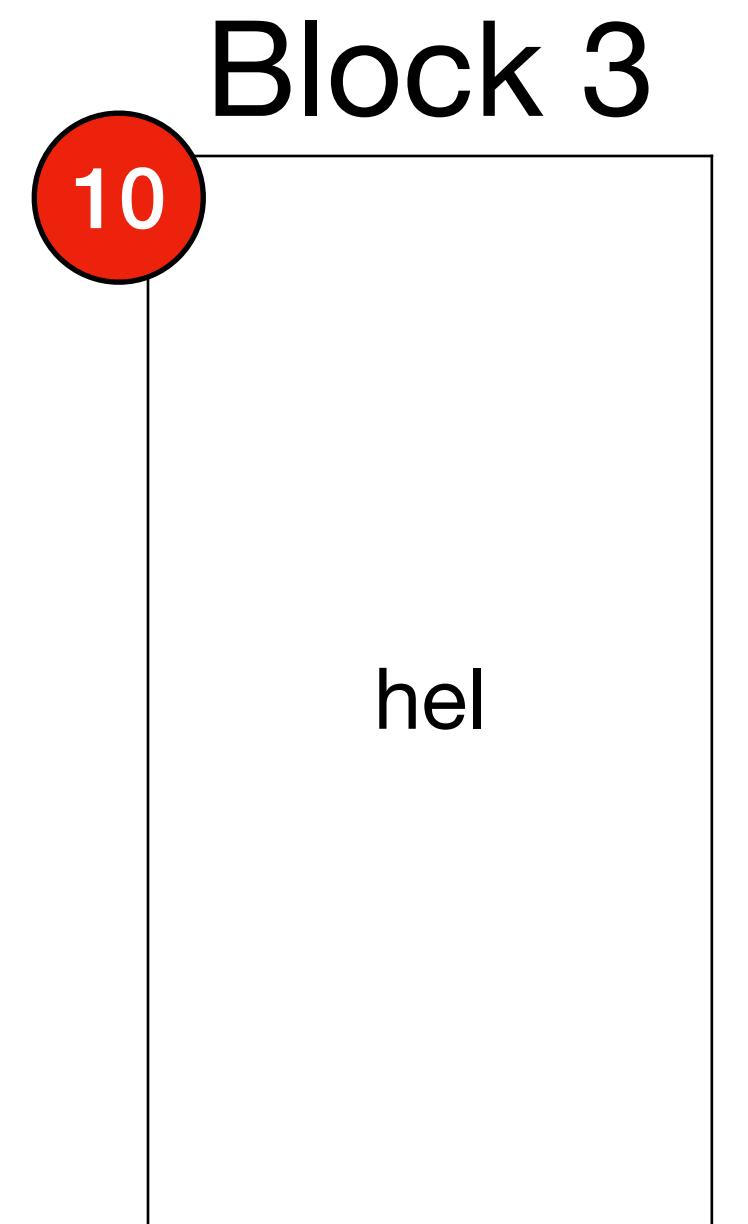
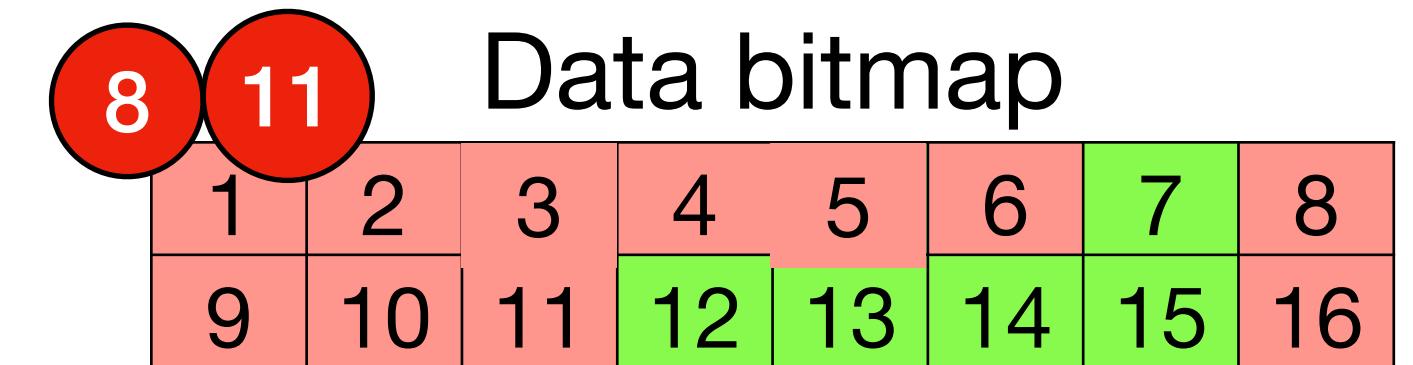
Inode = 1 “/”	
Type = directory	
Size	
2	

Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

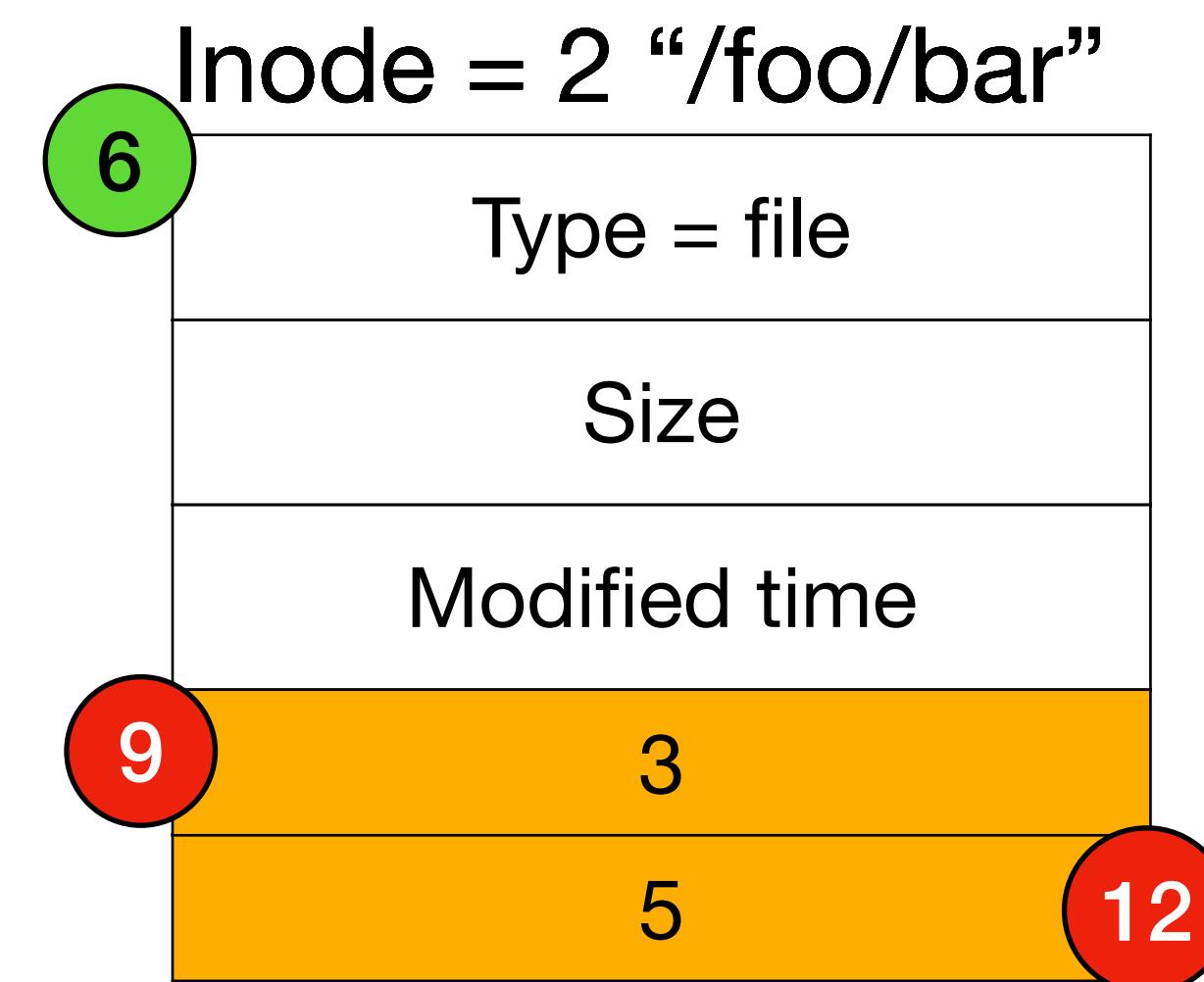
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

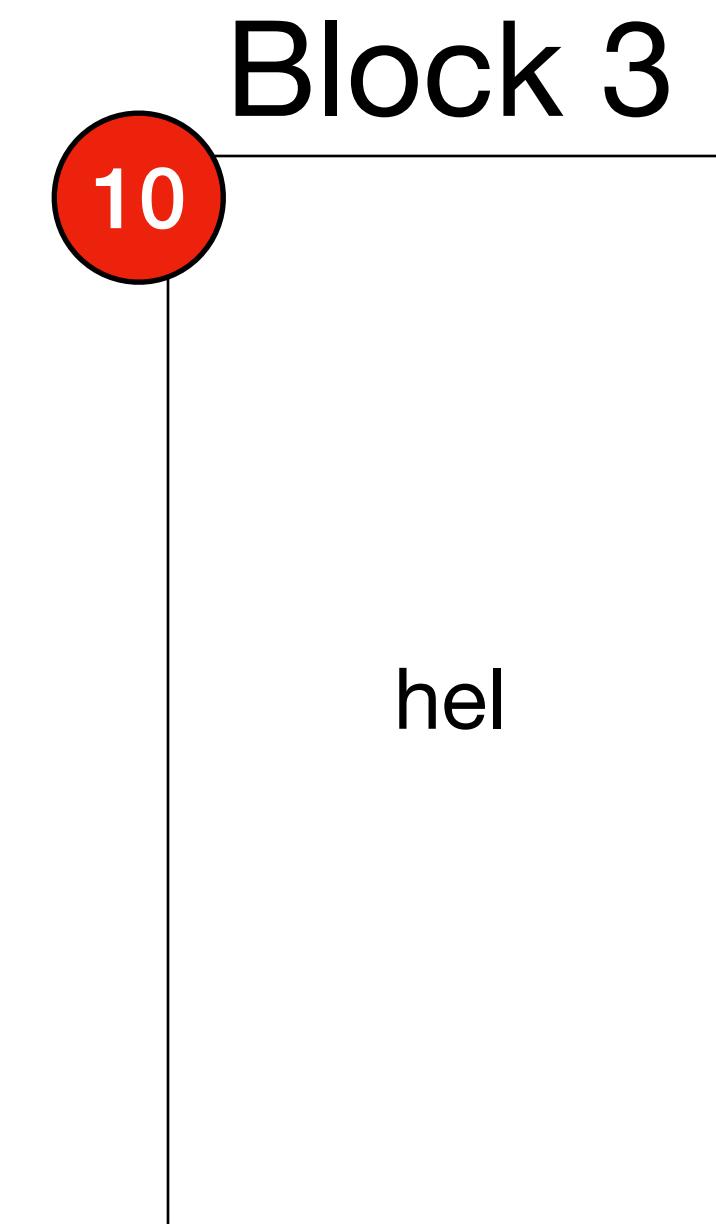
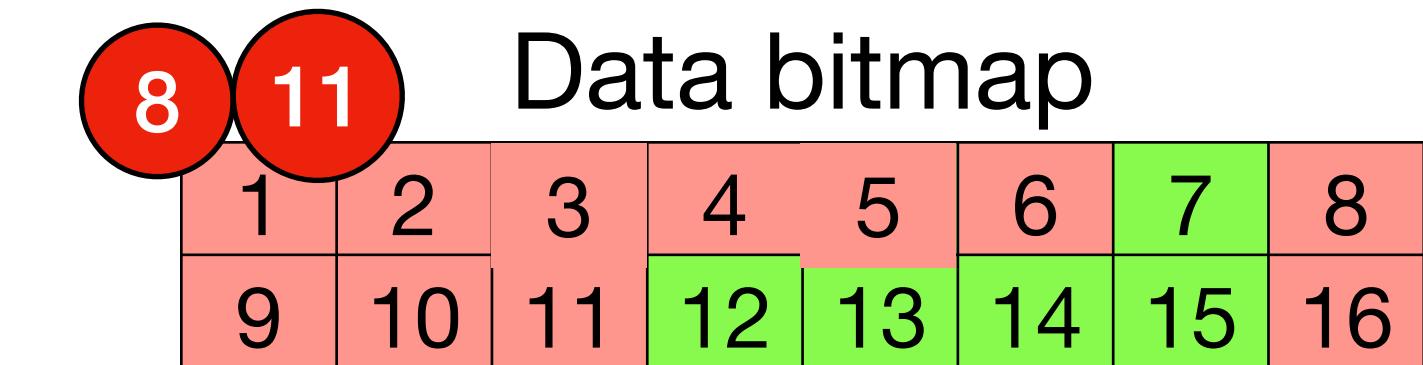
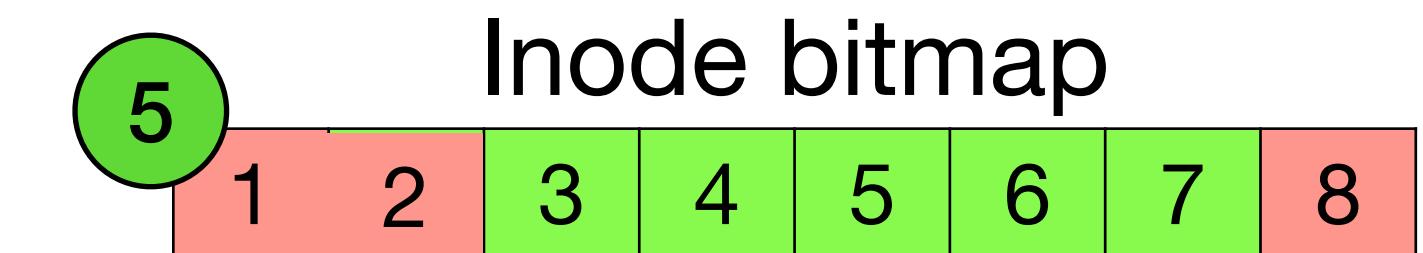
Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);

```



Writing a file

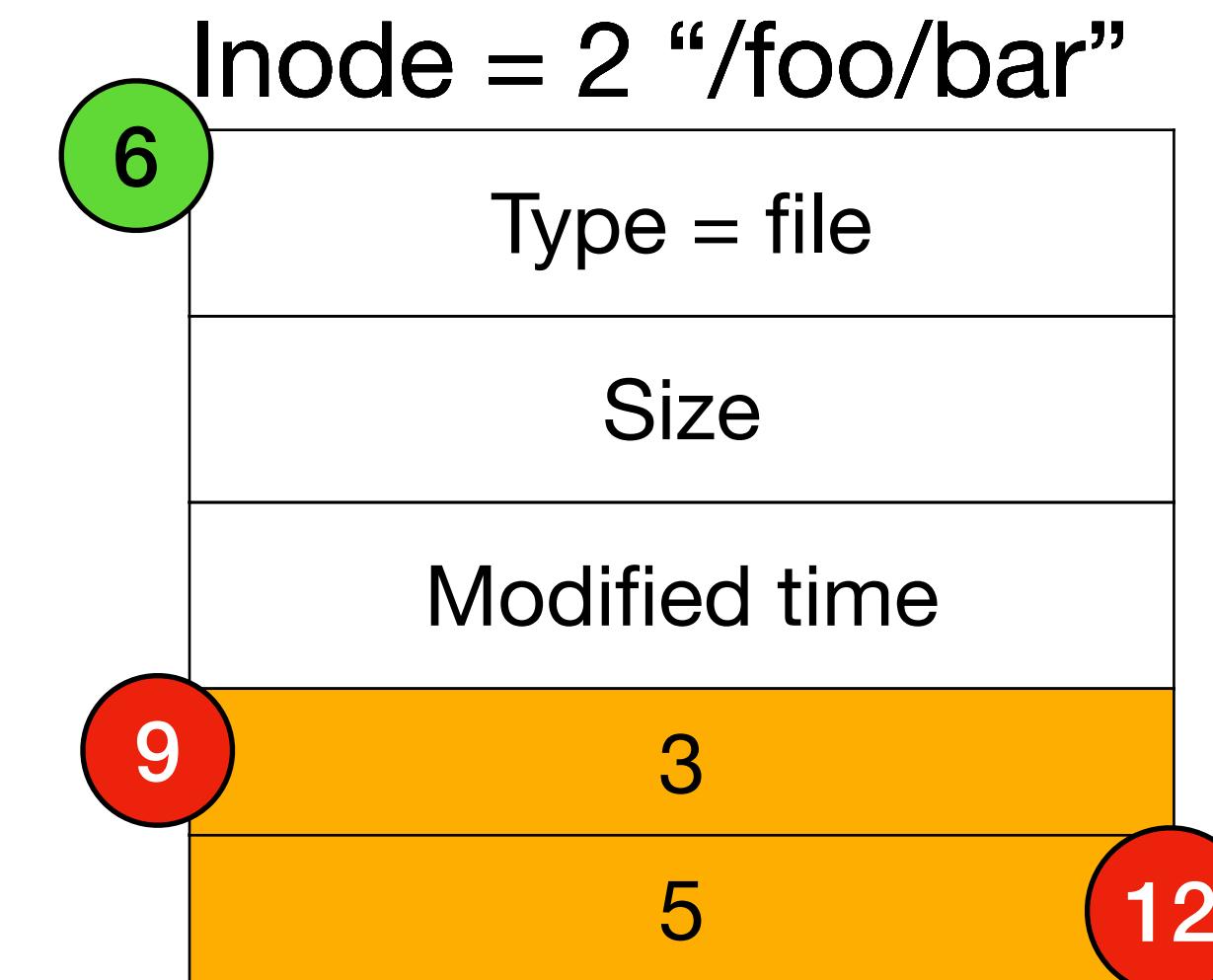
Example: /foo/bar

Inode = 1 “/”
Type = directory
Size
2

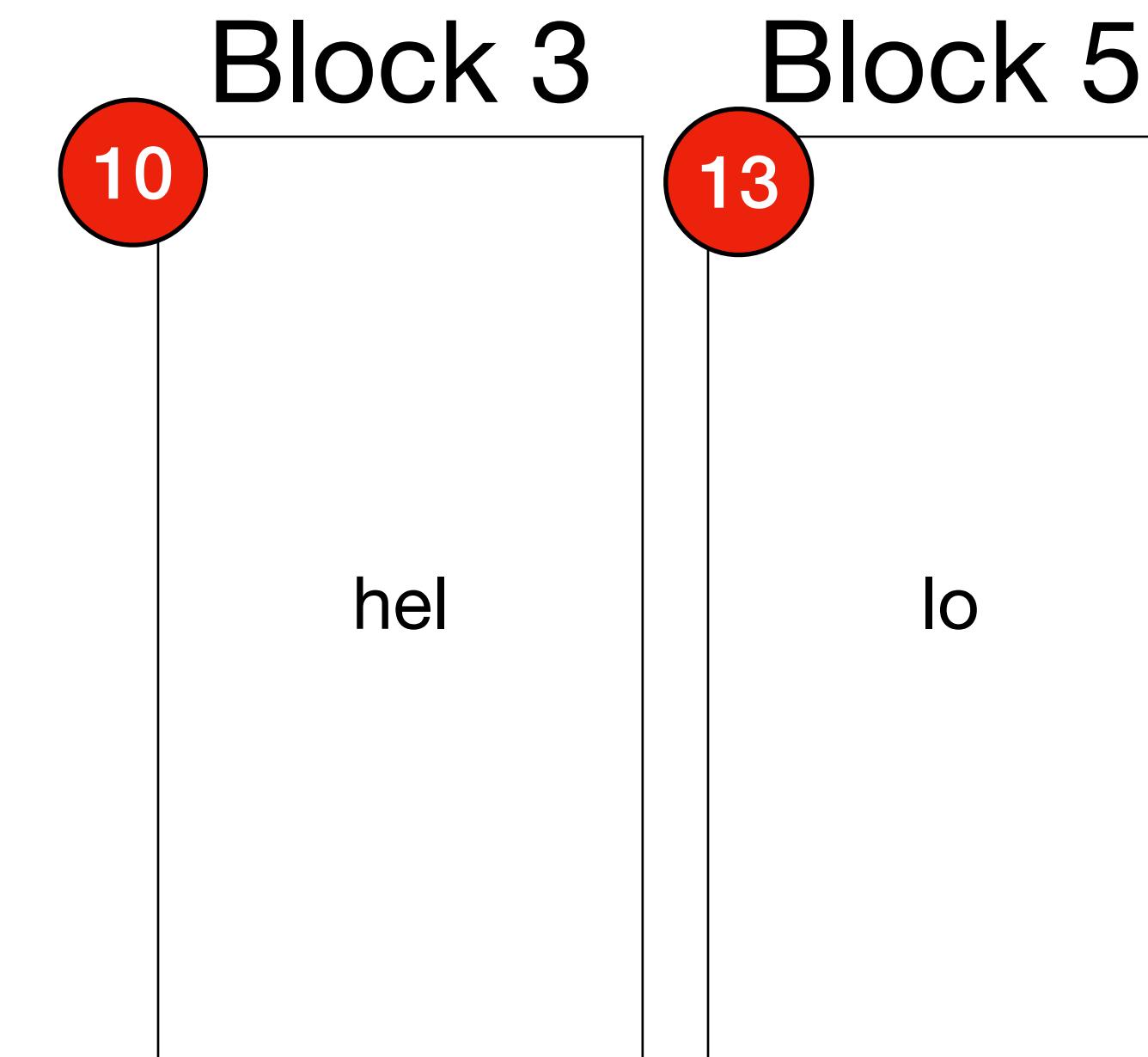
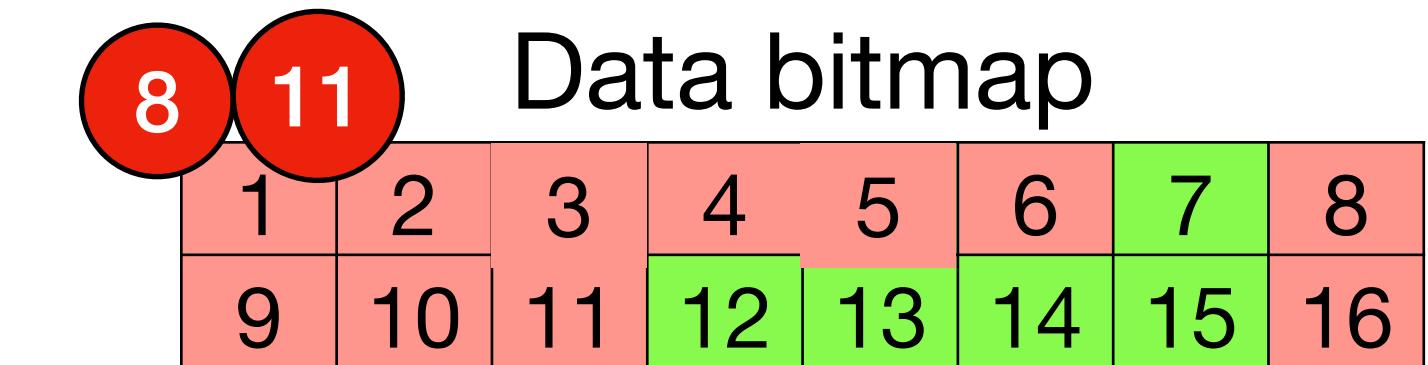
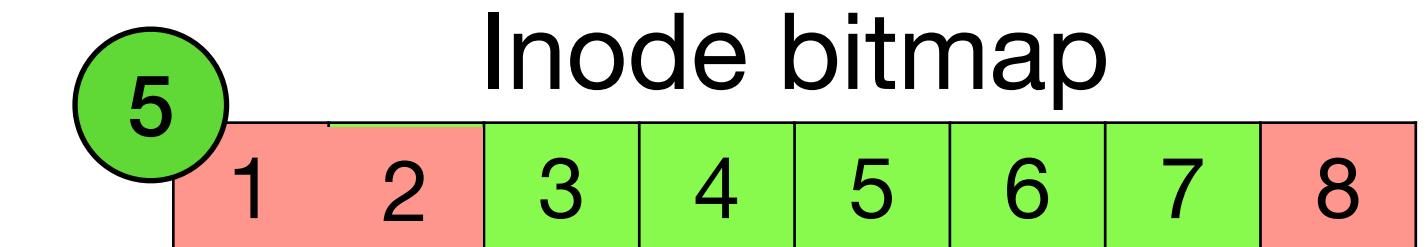
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”
Type = directory
Size
16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

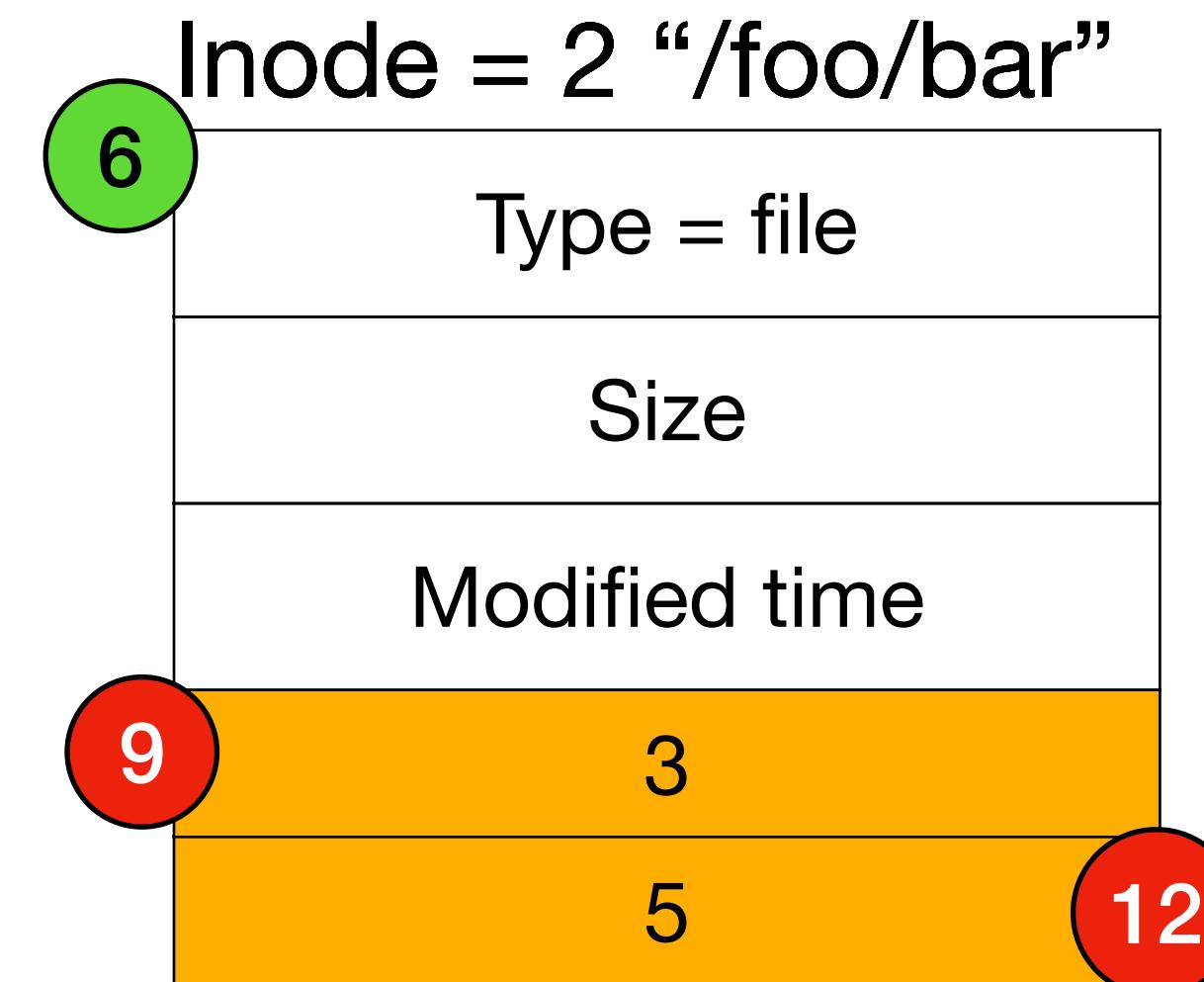
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

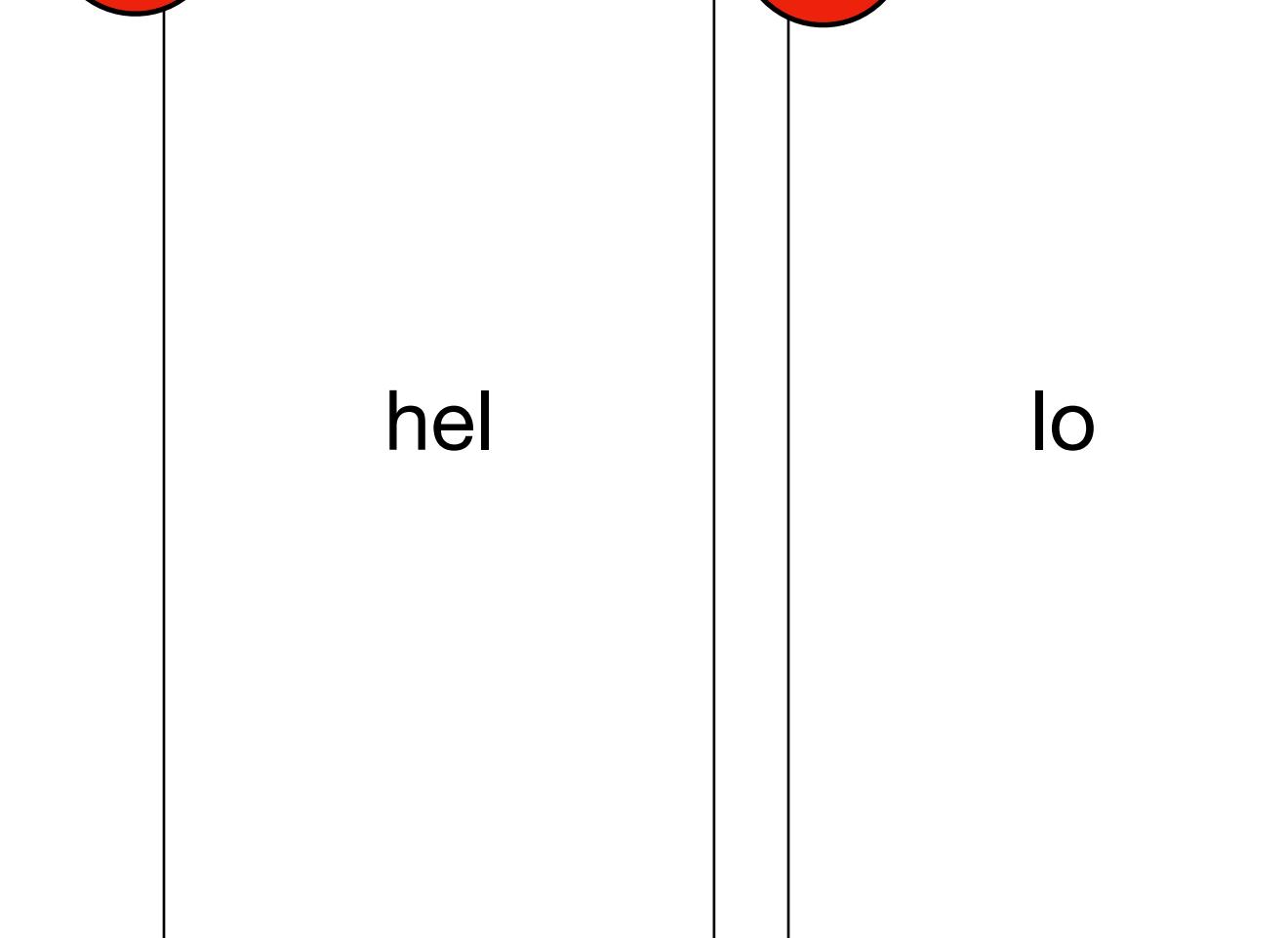
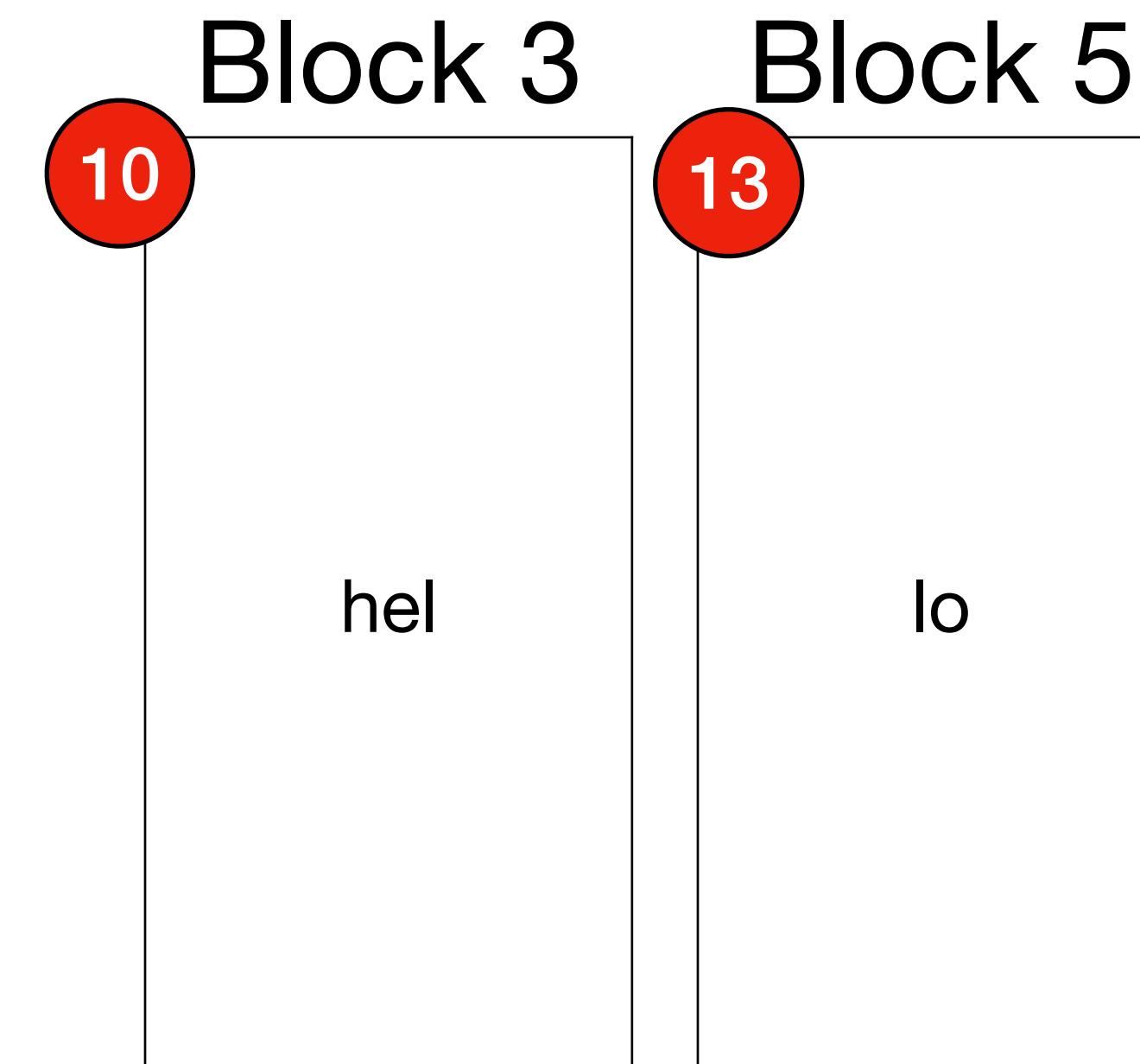
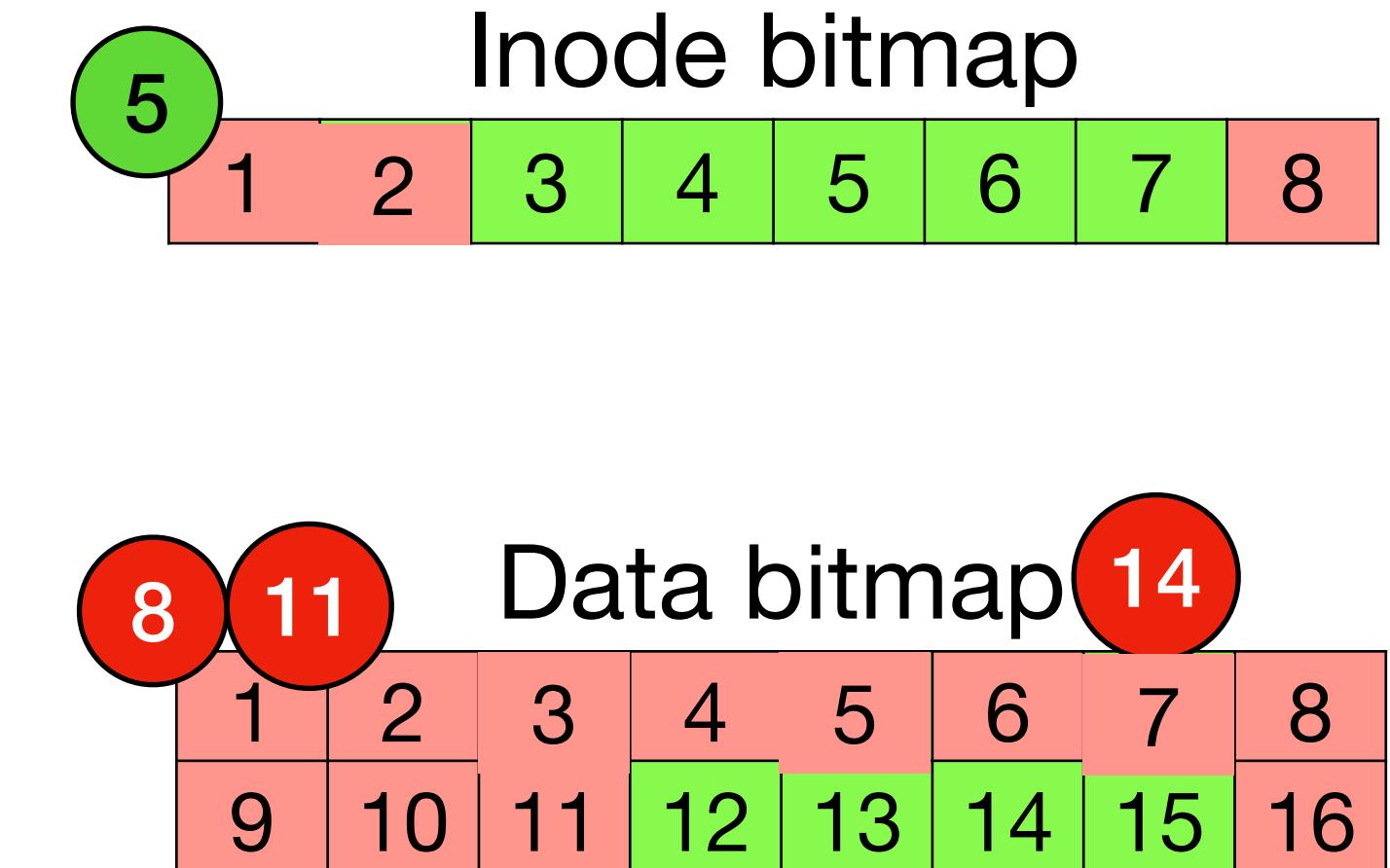
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```



Writing a file

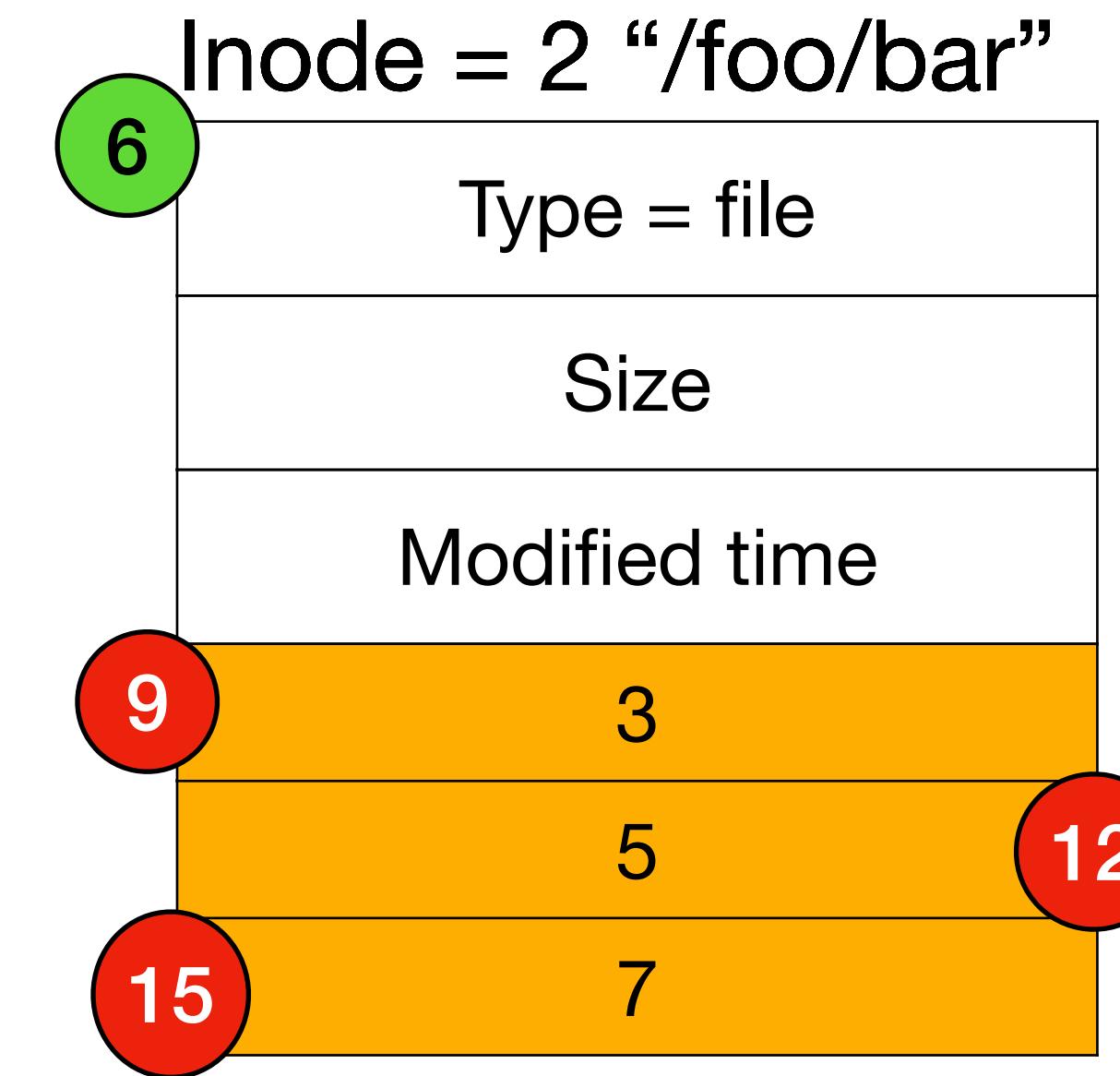
Example: /foo/bar

Inode = 1 “/”
Type = directory
Size
2

Block 2	
File/directory name	Inode number
.	1
foo	8

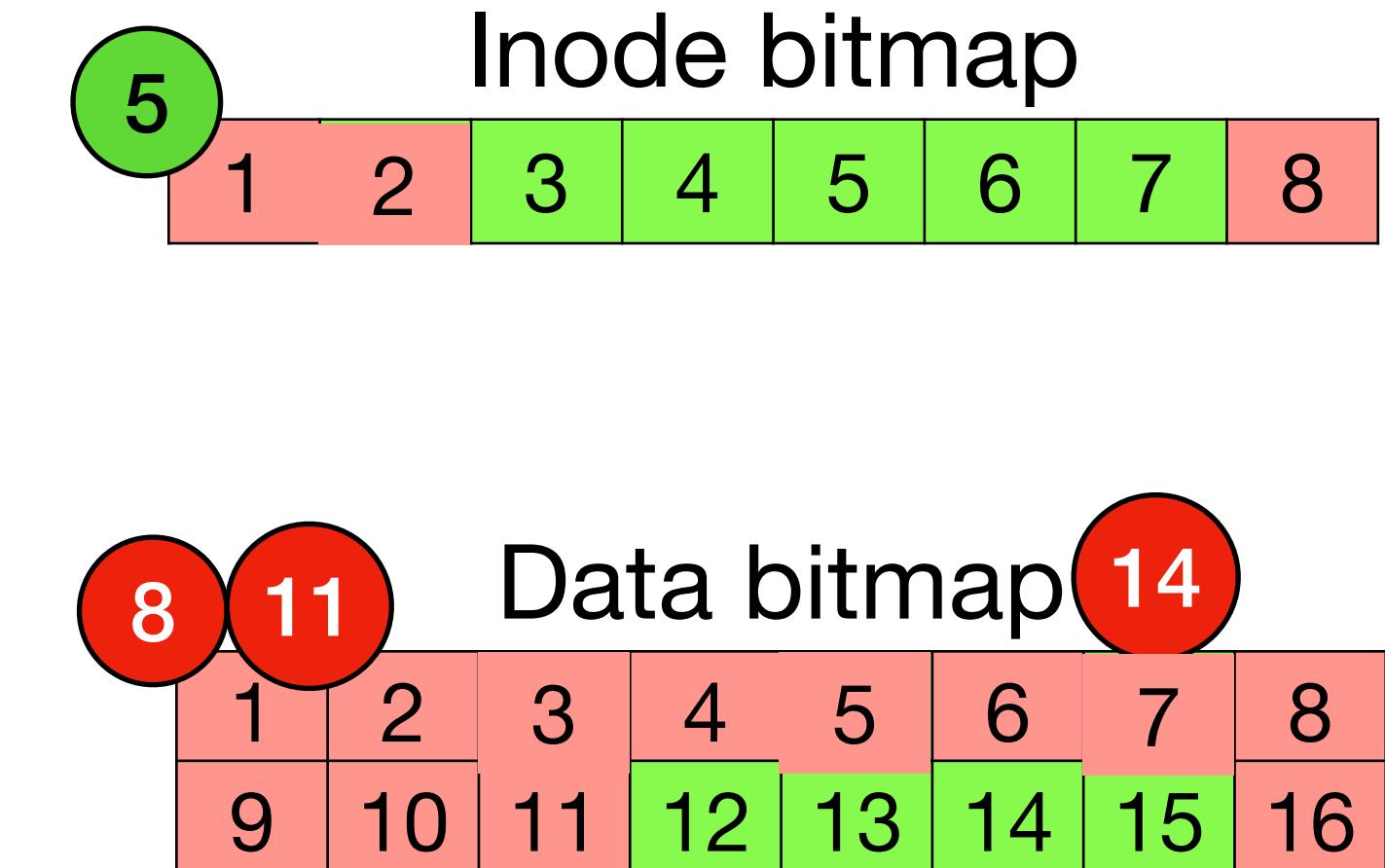
Inode = 8 “/foo”
Type = directory
Size
16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```

fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
  
```



Writing a file

Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

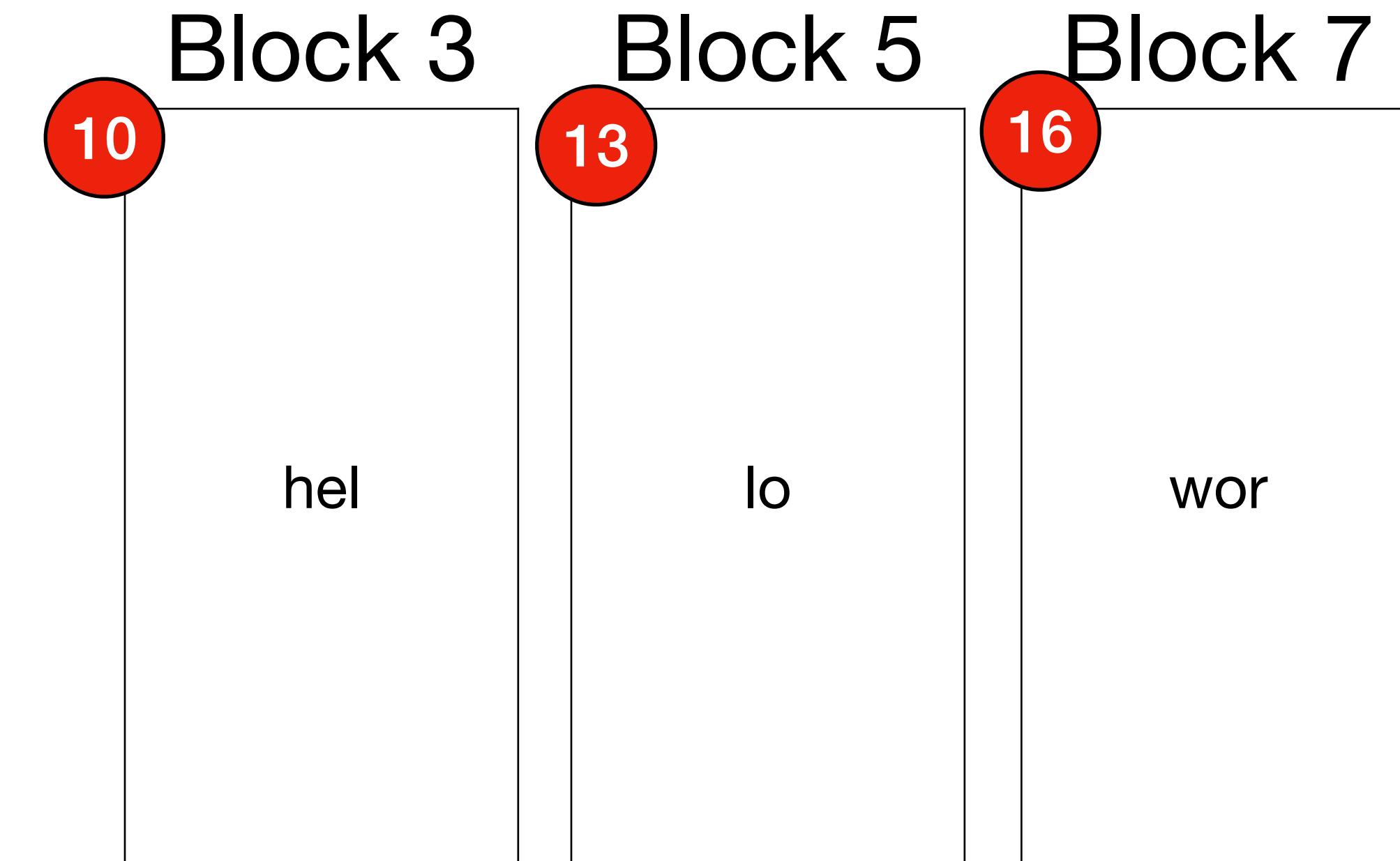
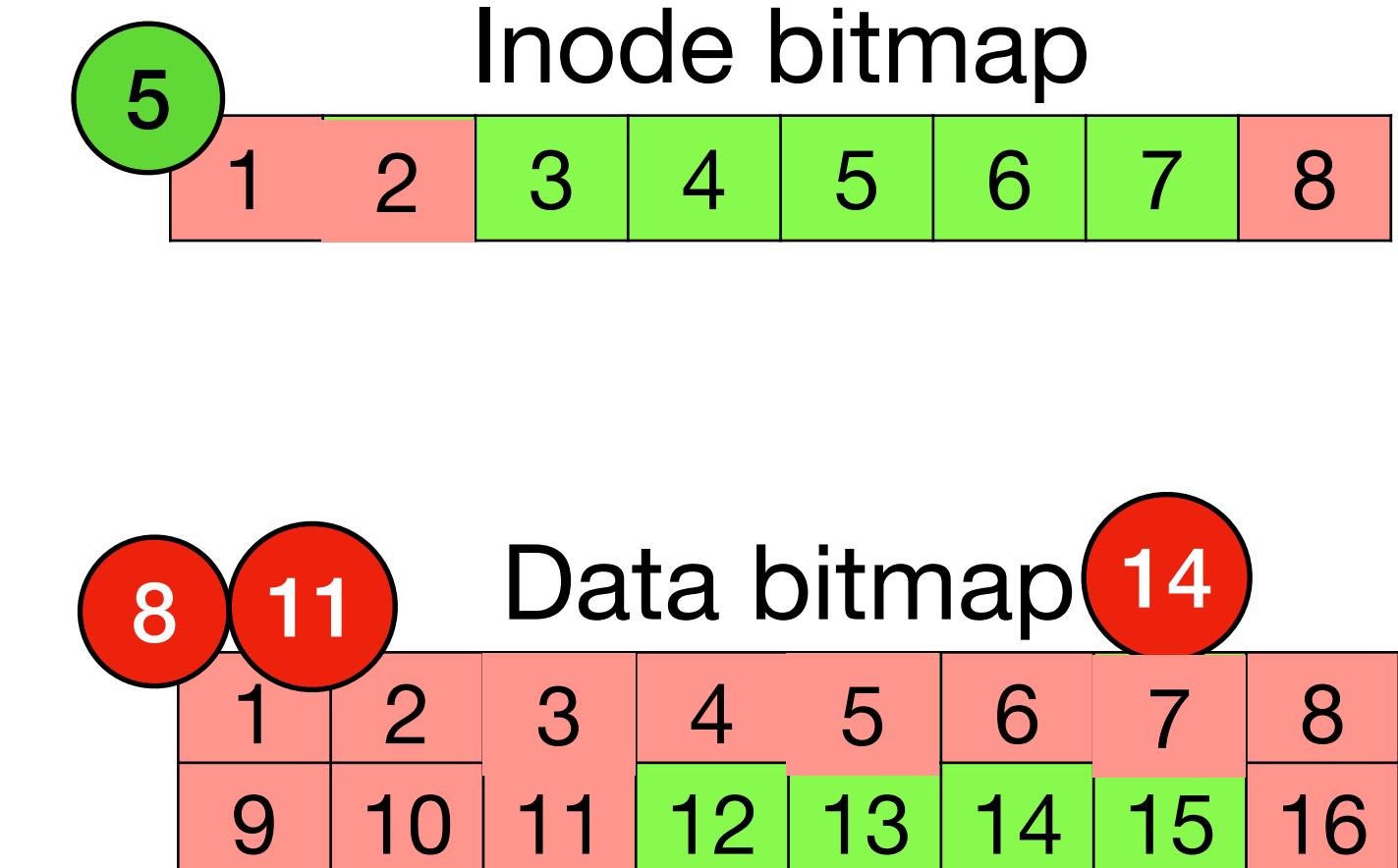
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time
9	3
15	5
	7

```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```

Writing a file

Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

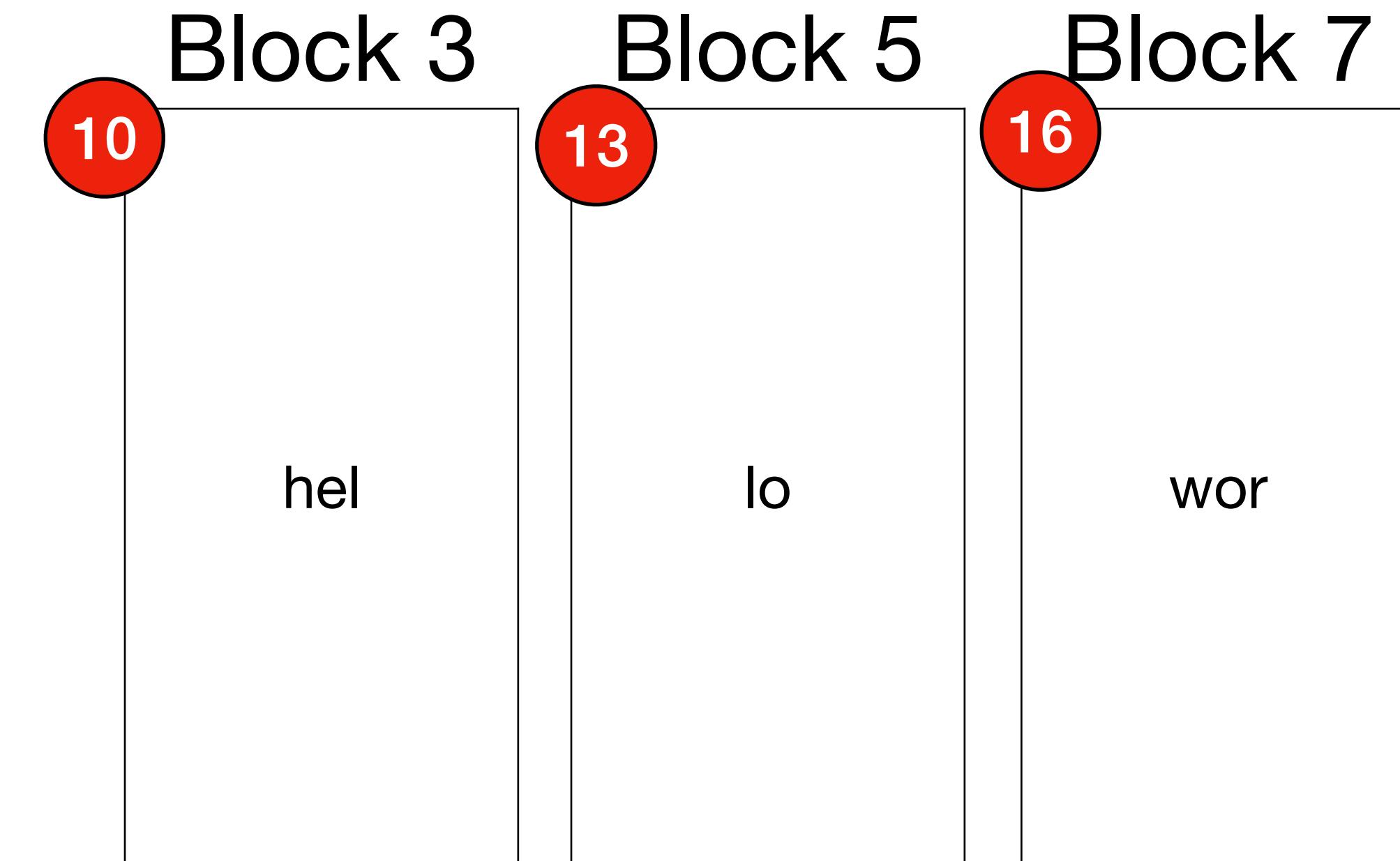
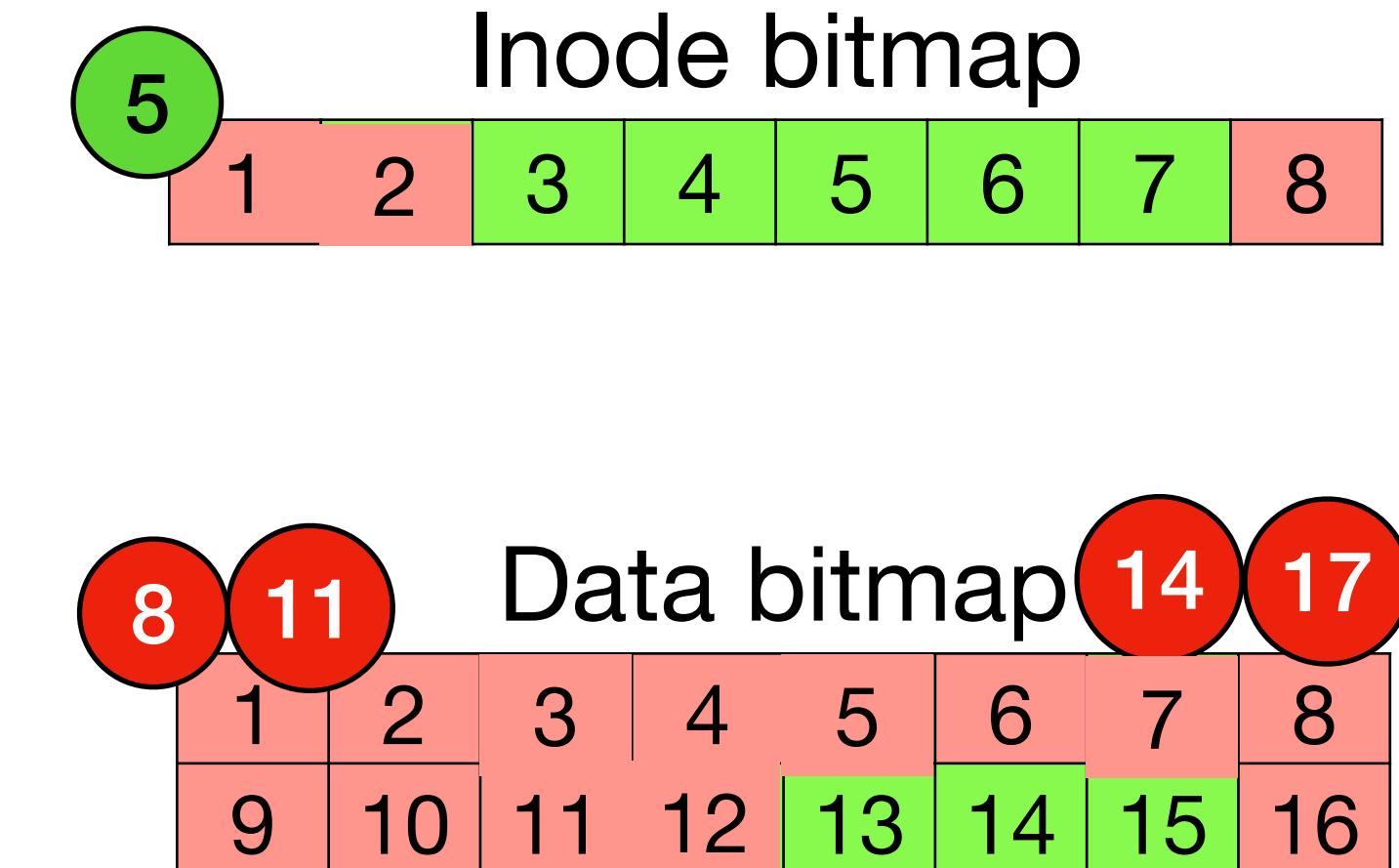
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

6	Inode = 2 “/foo/bar”
	Type = file
	Size
	Modified time
9	3
15	5
12	7

```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

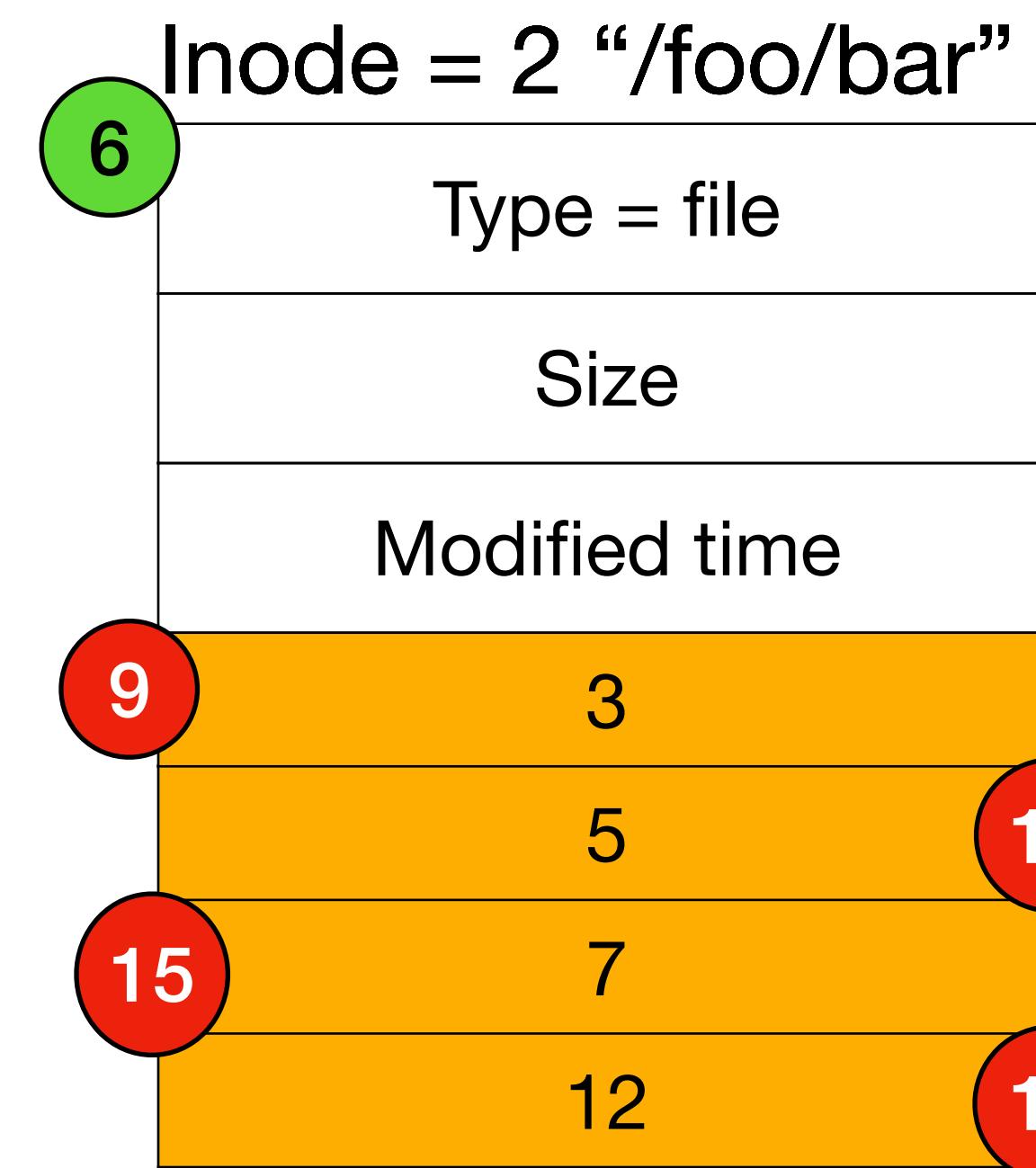
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	
2	1

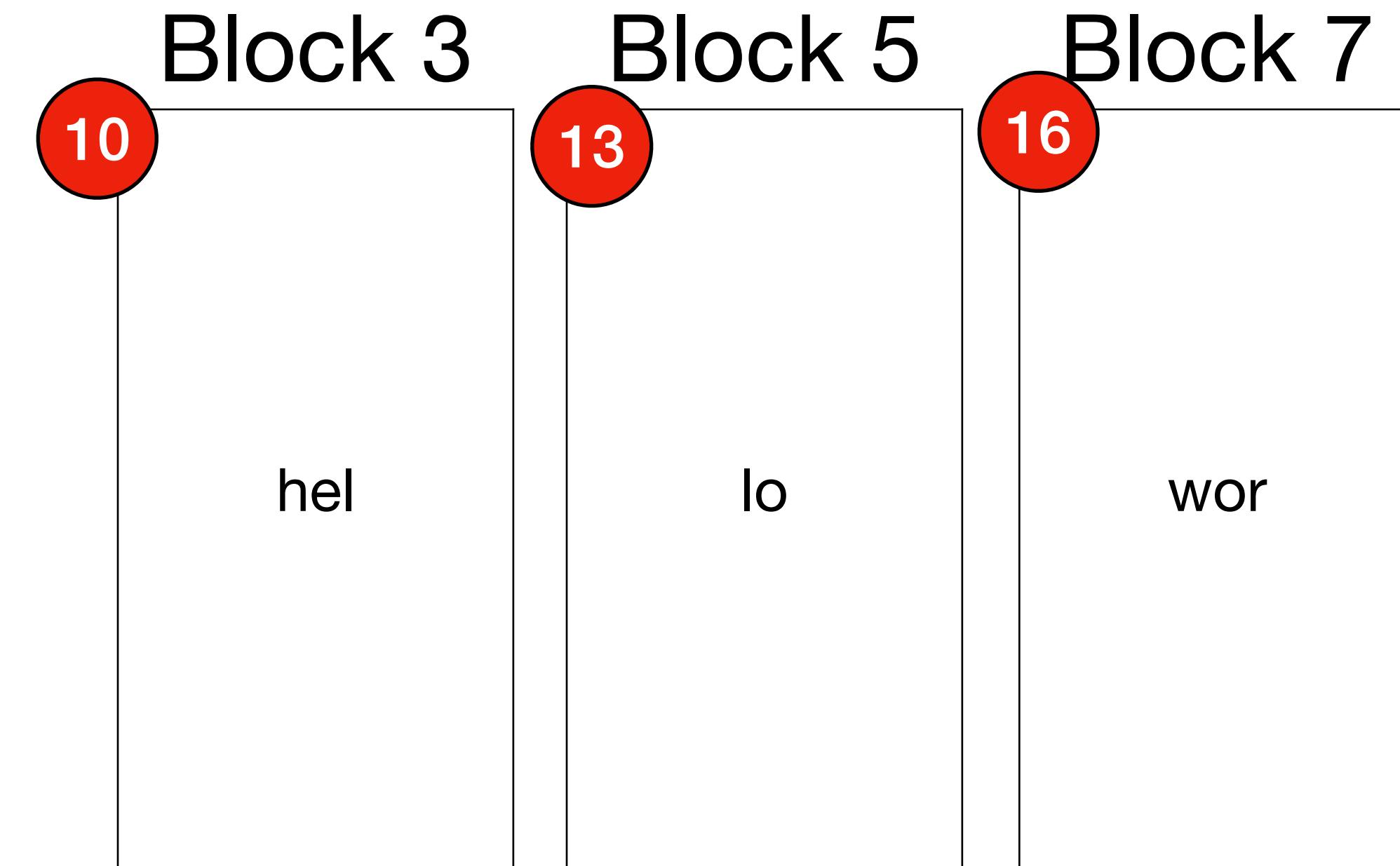
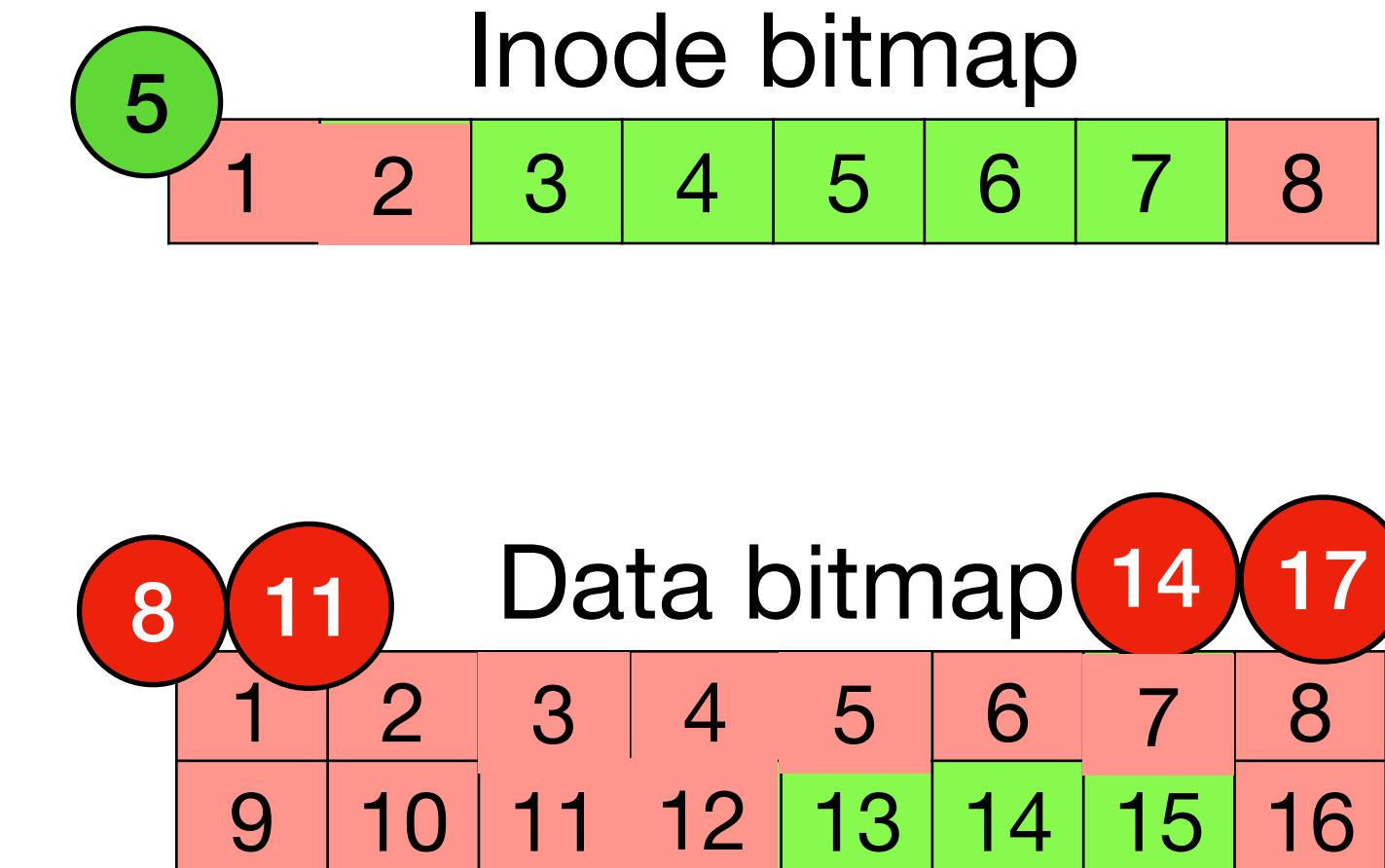
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	
16	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

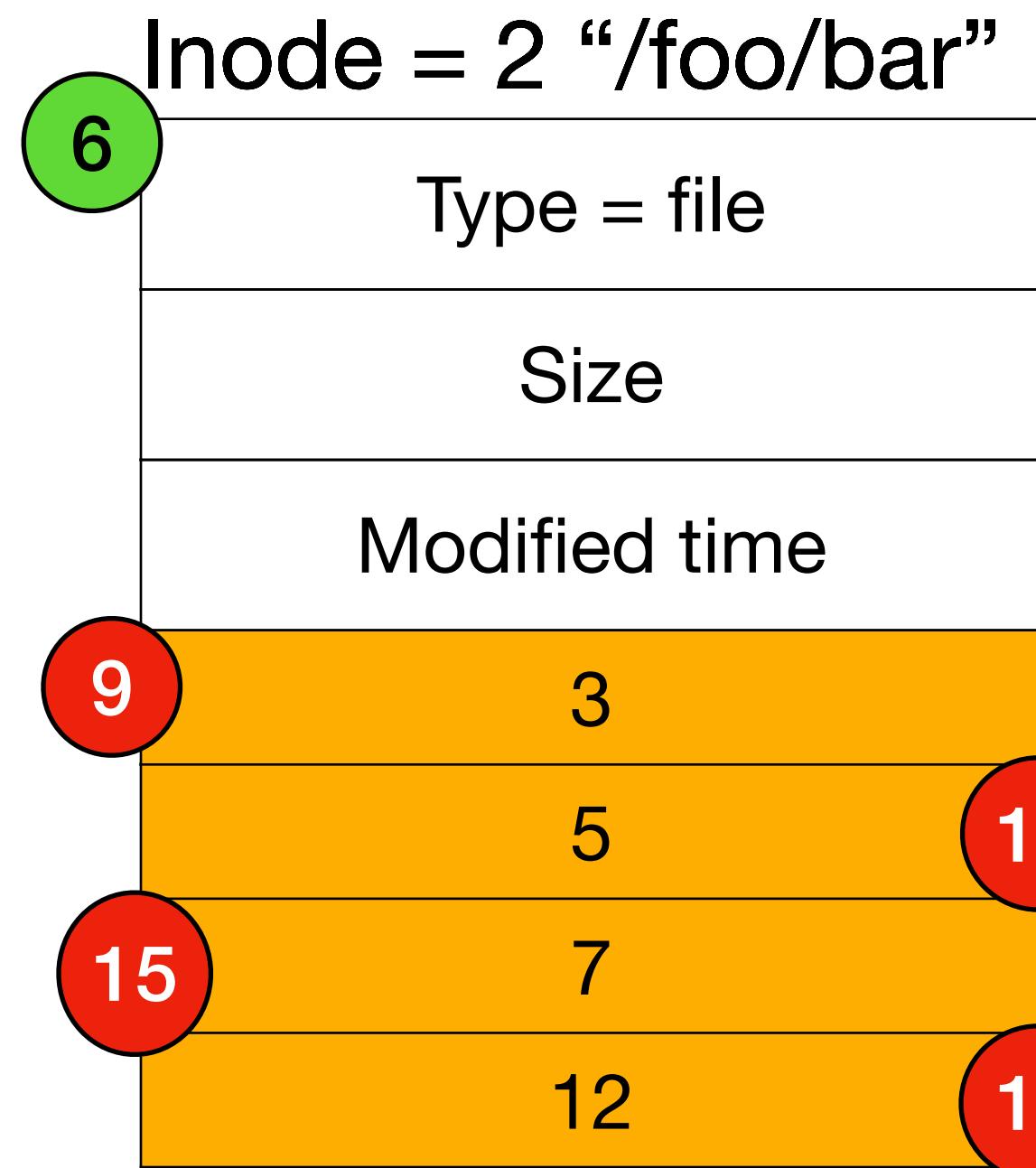
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	2
	1

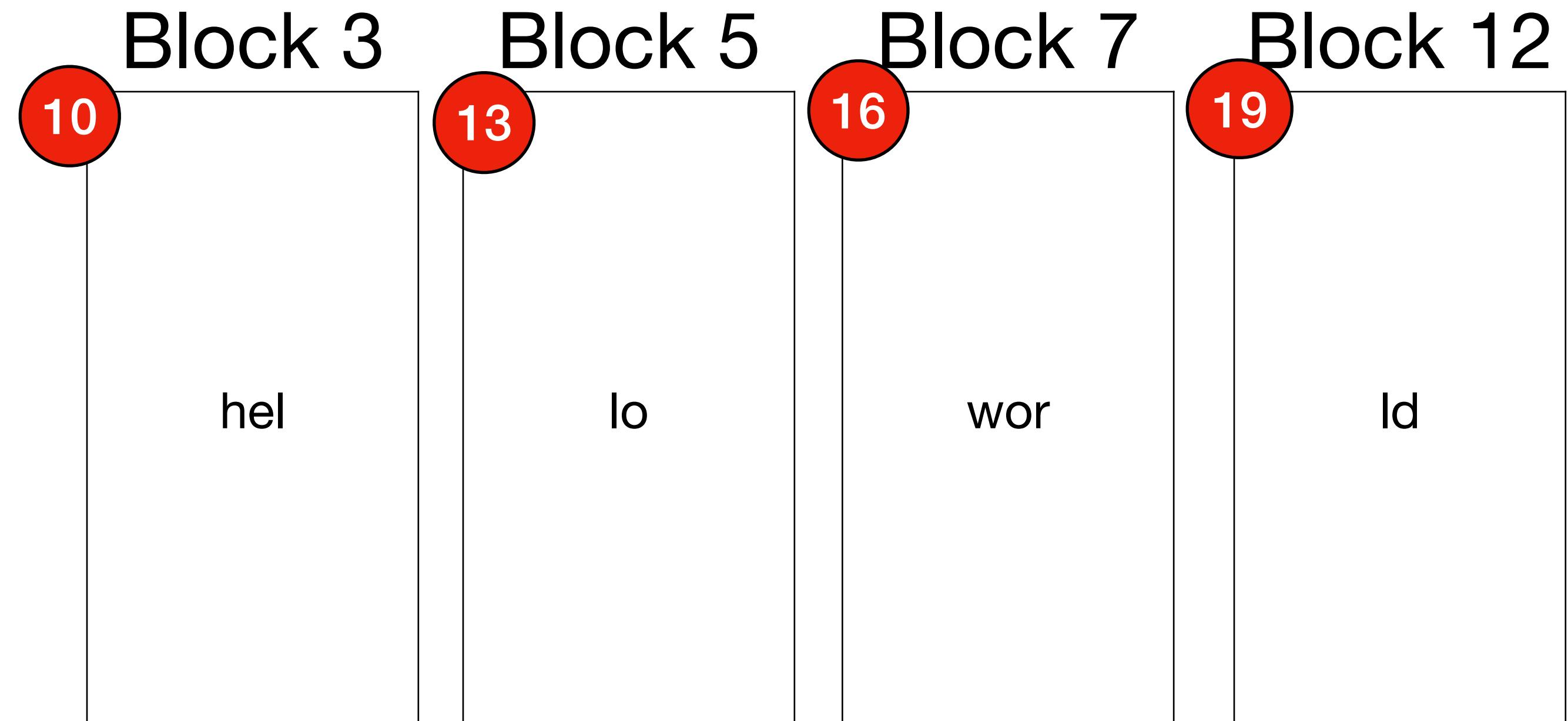
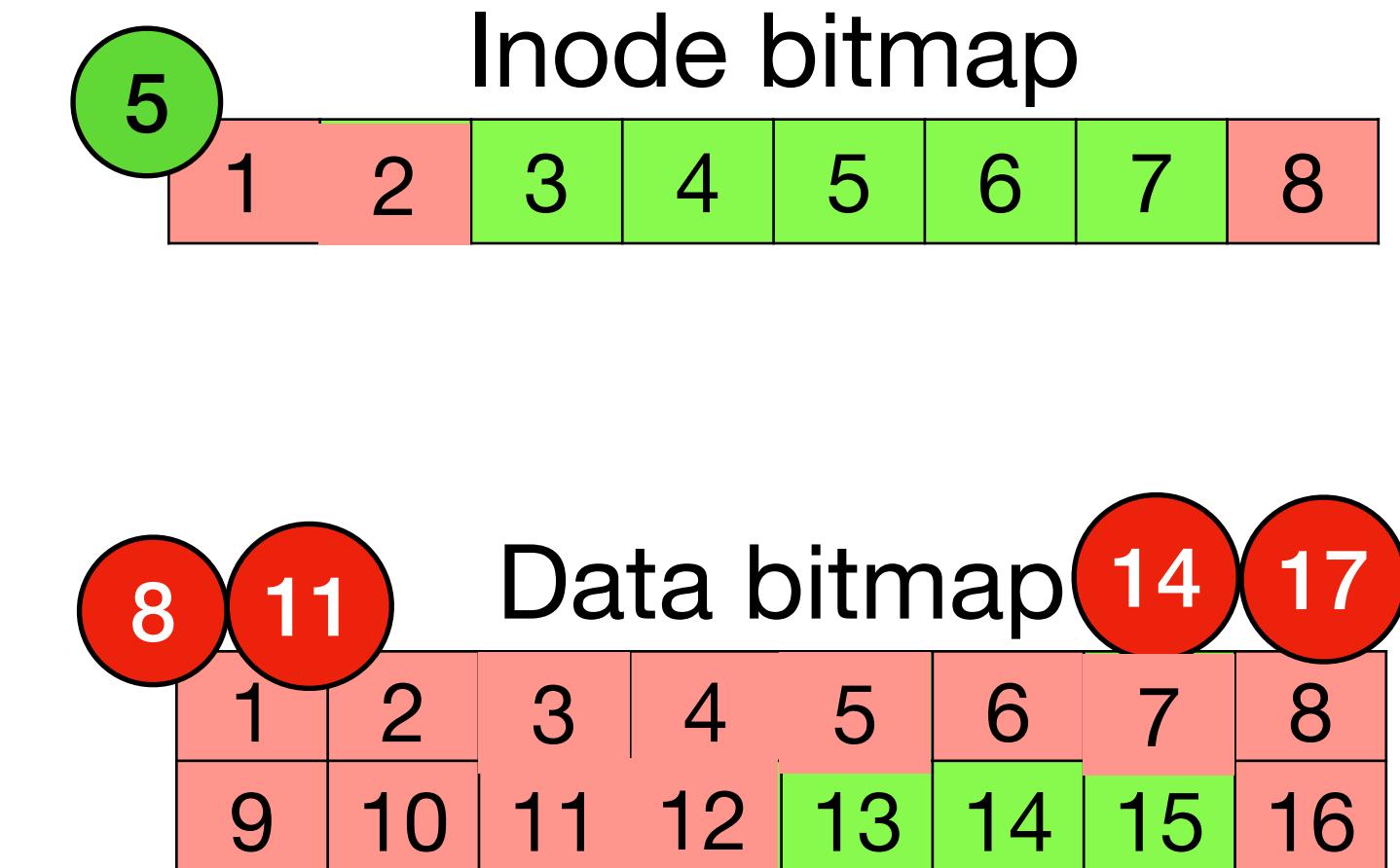
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	16
	3

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Writing a file

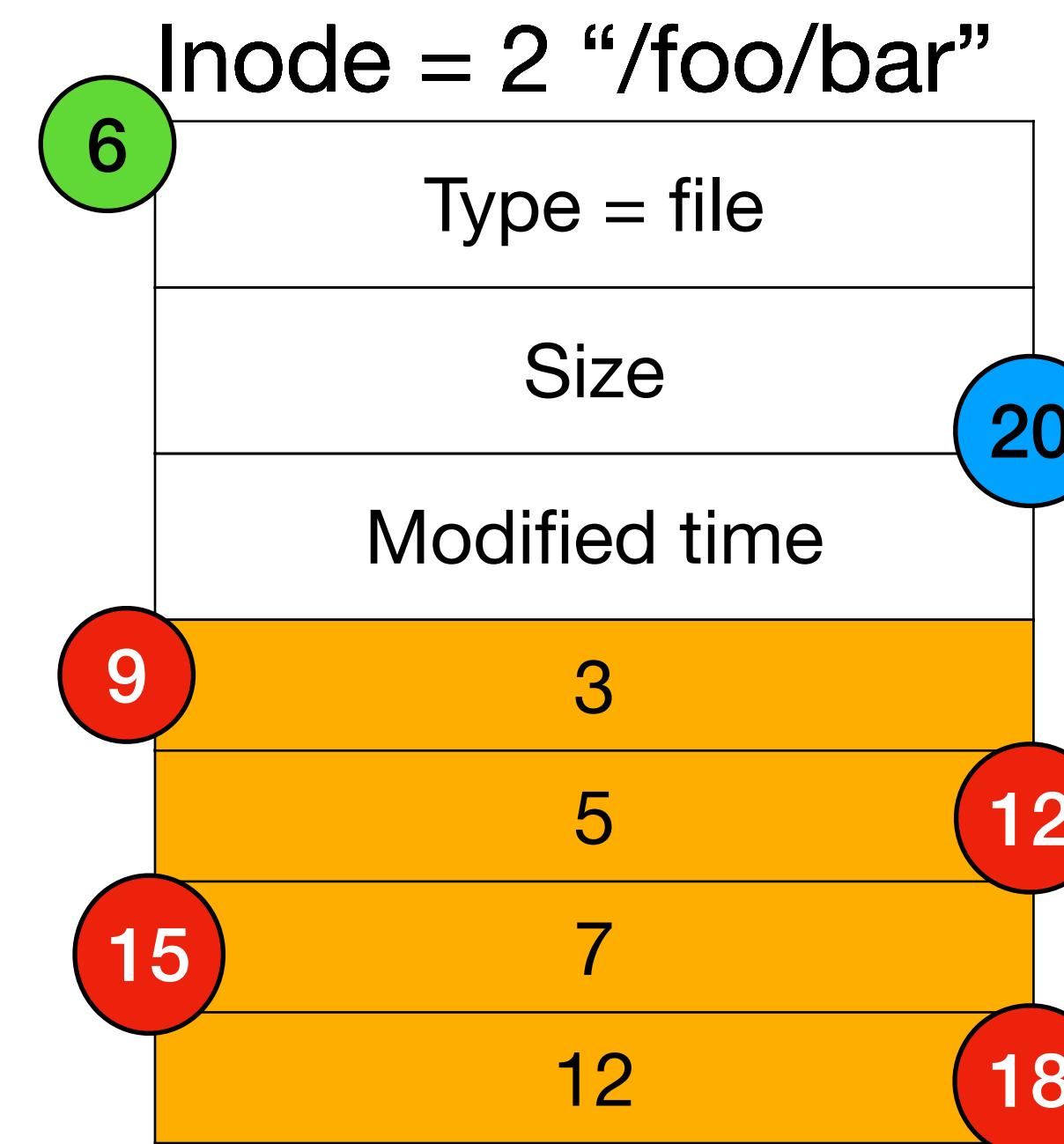
Example: /foo/bar

Inode = 1 “/”	
Type = directory	
Size	2

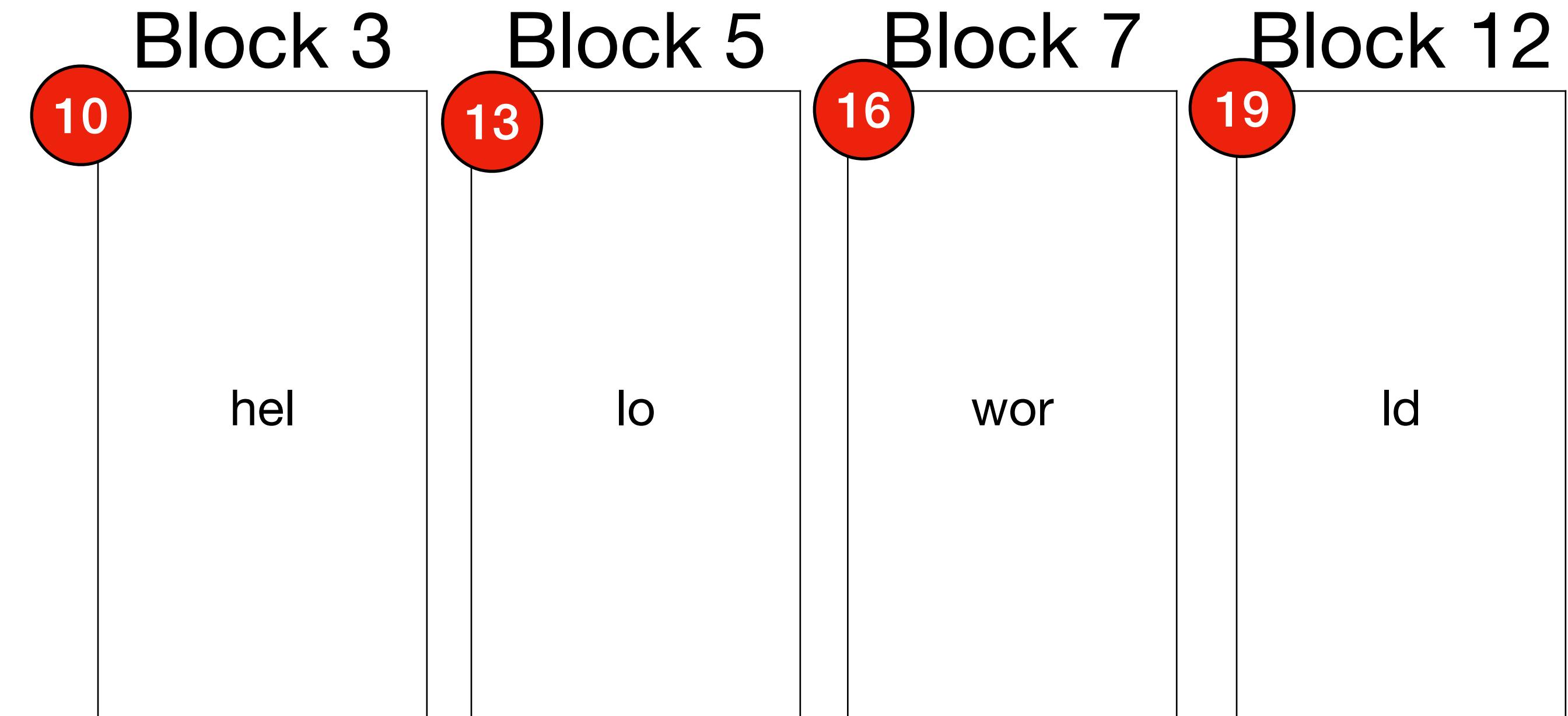
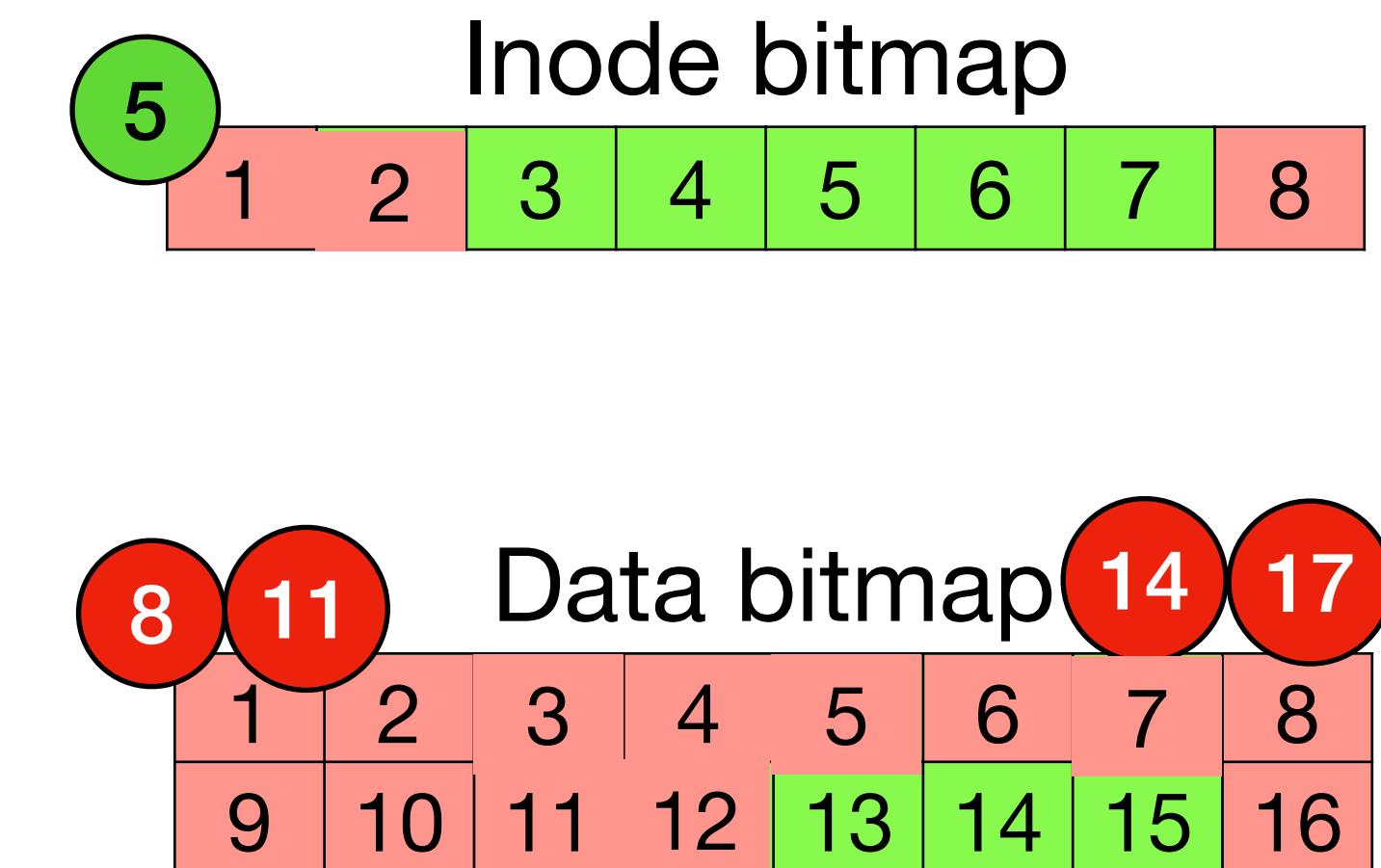
Block 2	
File/directory name	Inode number
.	1
foo	8

Inode = 8 “/foo”	
Type = directory	
Size	16

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2



```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```



Crash problem

- Sending all the requests in parallel for better write throughput (disk scheduling)
- Crash can happen at any time => only a subset of blocks may get written

Inode = 2 “/foo/bar”

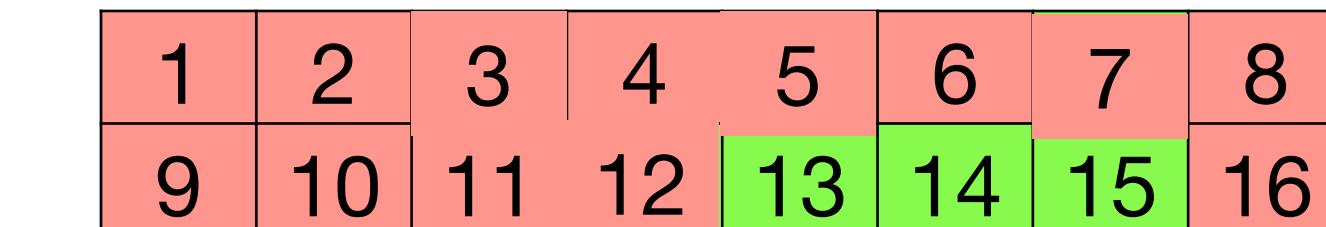
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE);
write(fd, "hello world\n", 12);
close(fd);
```

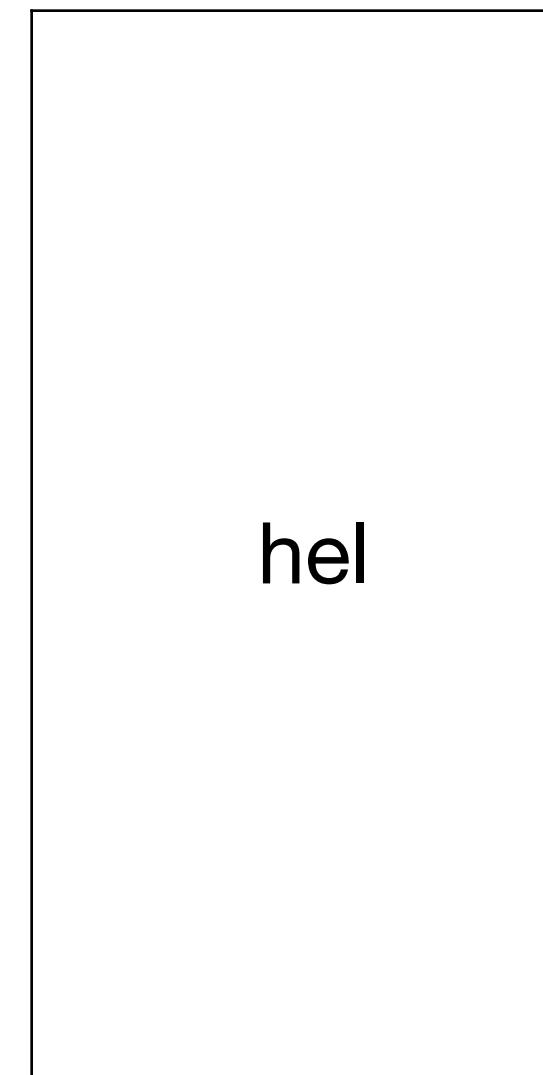
Inode bitmap



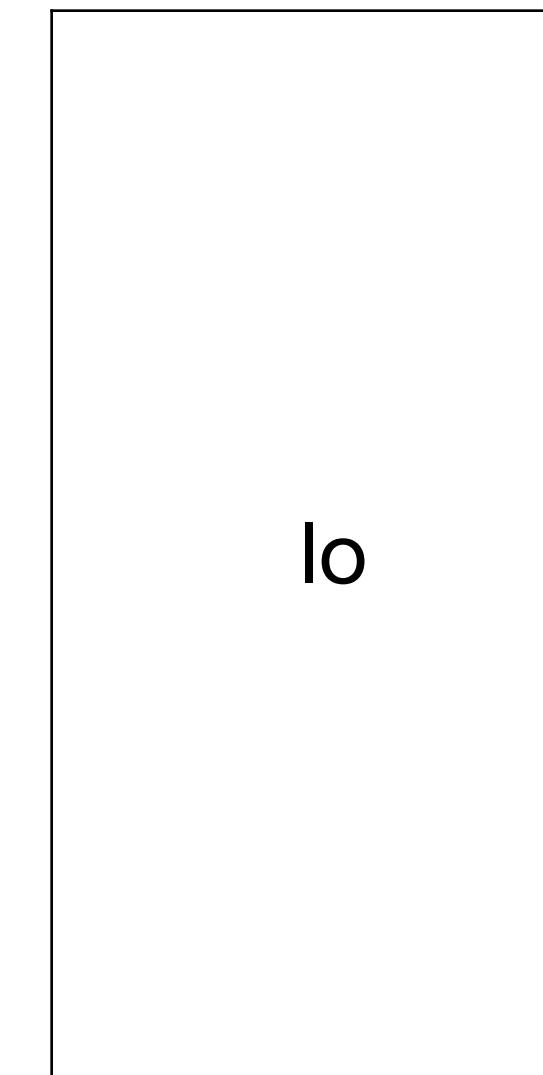
Data bitmap



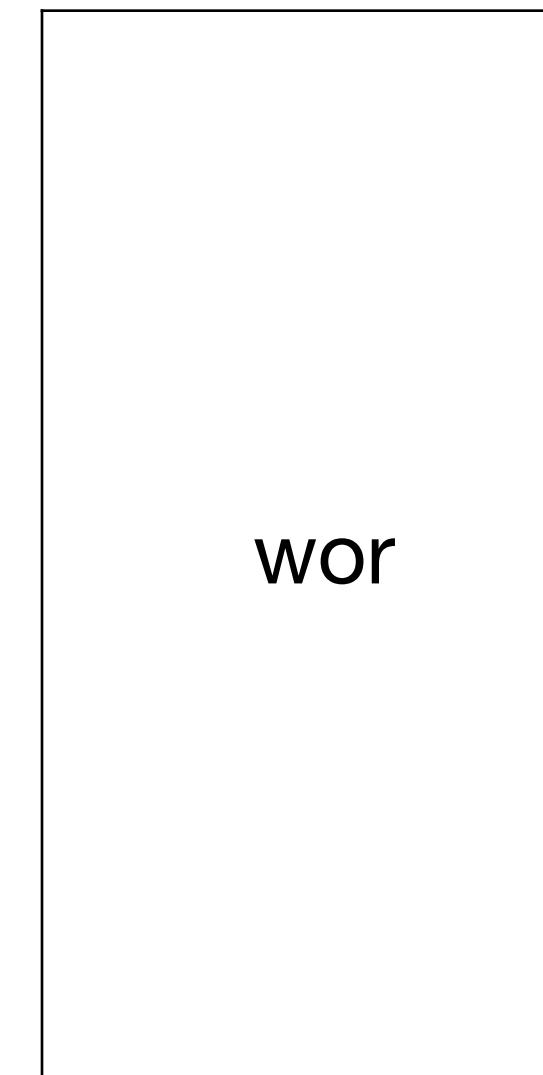
Block 3



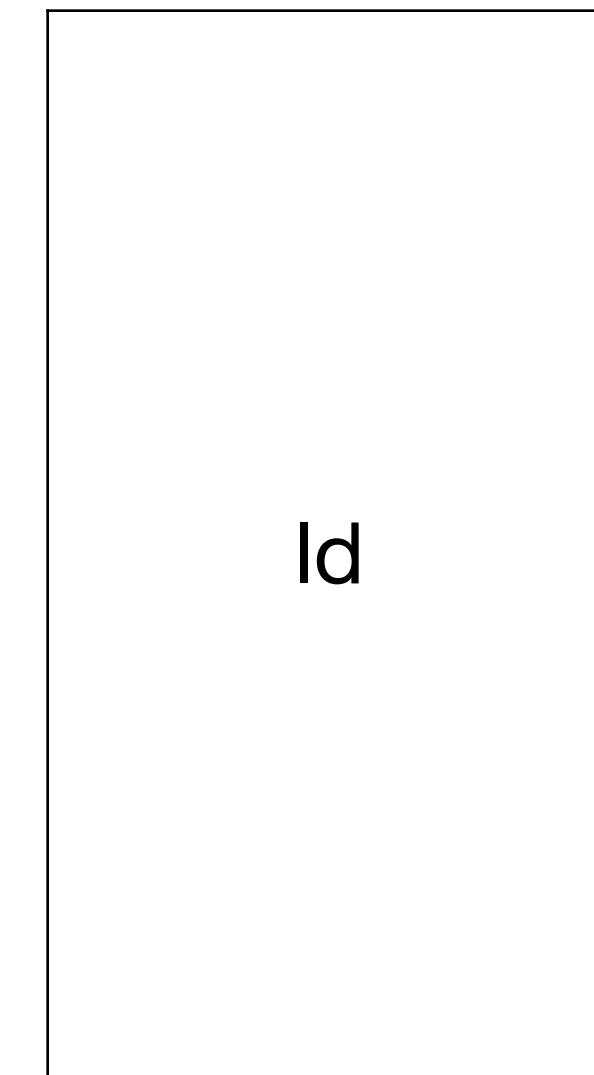
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

What can go wrong?

Missed data blocks

- File now contains garbage data

Inode = 2 “/foo/bar”

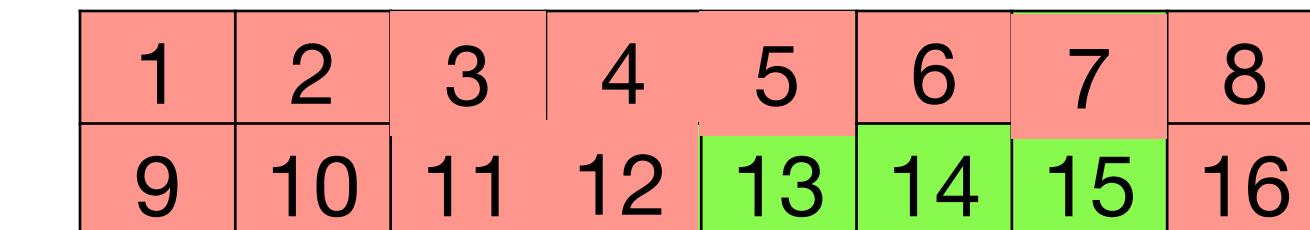
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE);
write(fd, "hello world\n", 12);
close(fd);
```

Inode bitmap



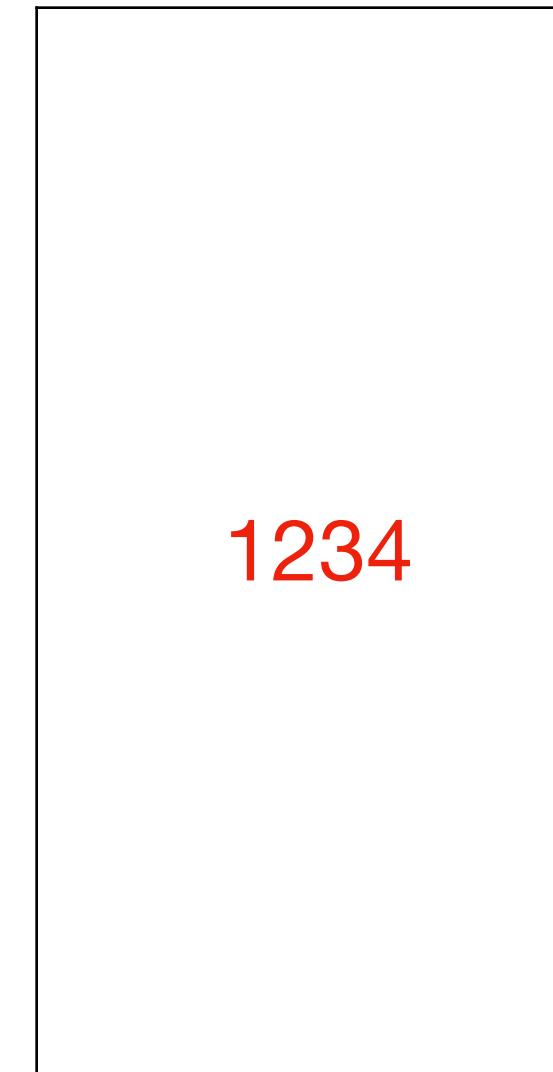
Data bitmap



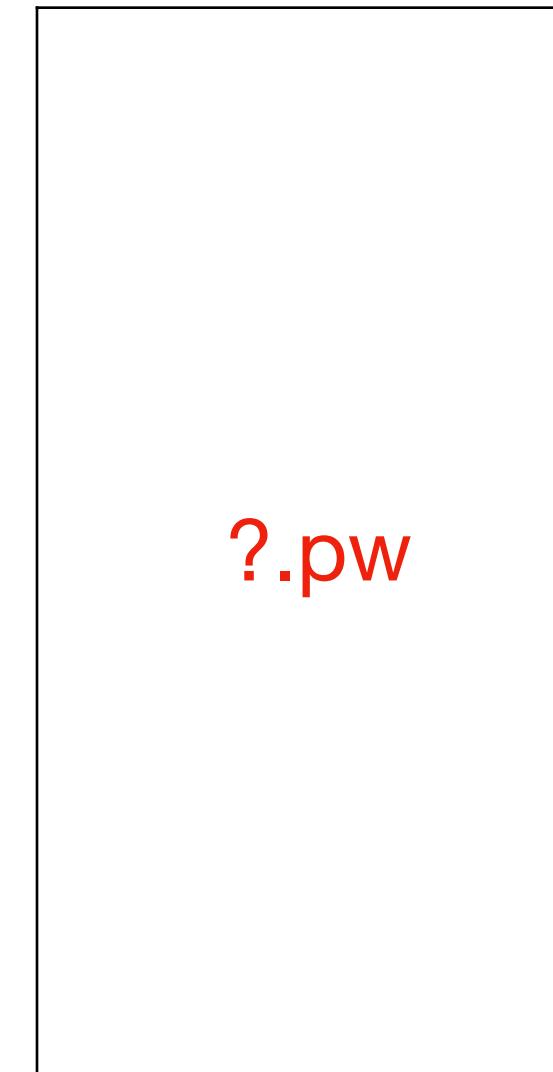
Block 3



Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

What can go wrong?

Missed data bitmap

- File initially looks ok
- Data blocks can get overwritten later by contents of another file

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

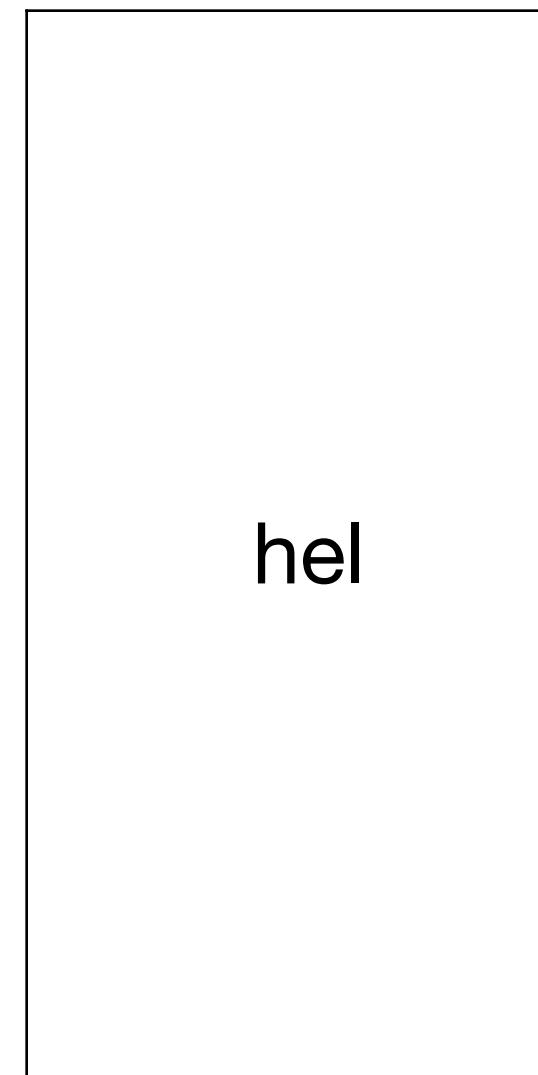
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

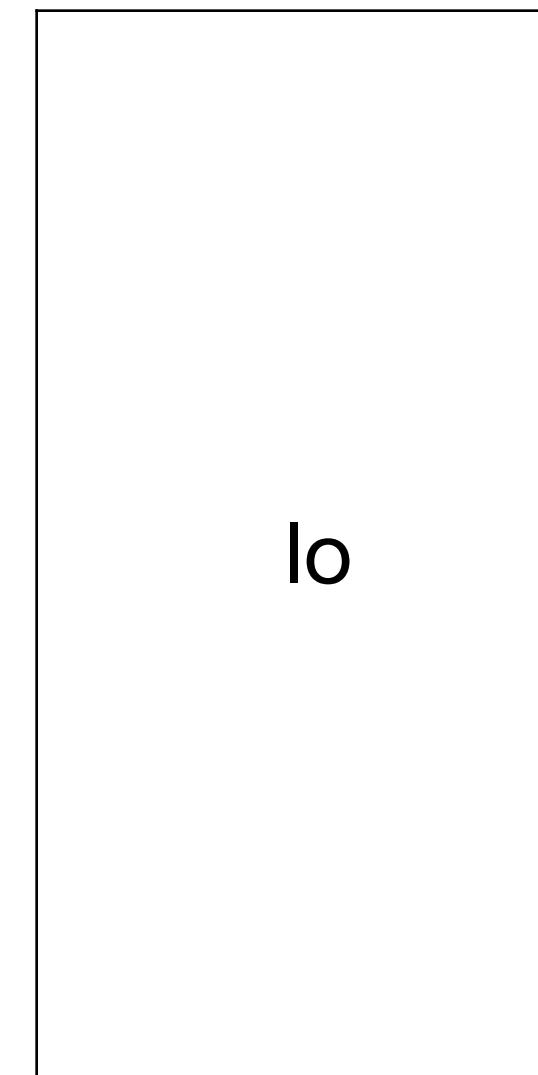
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

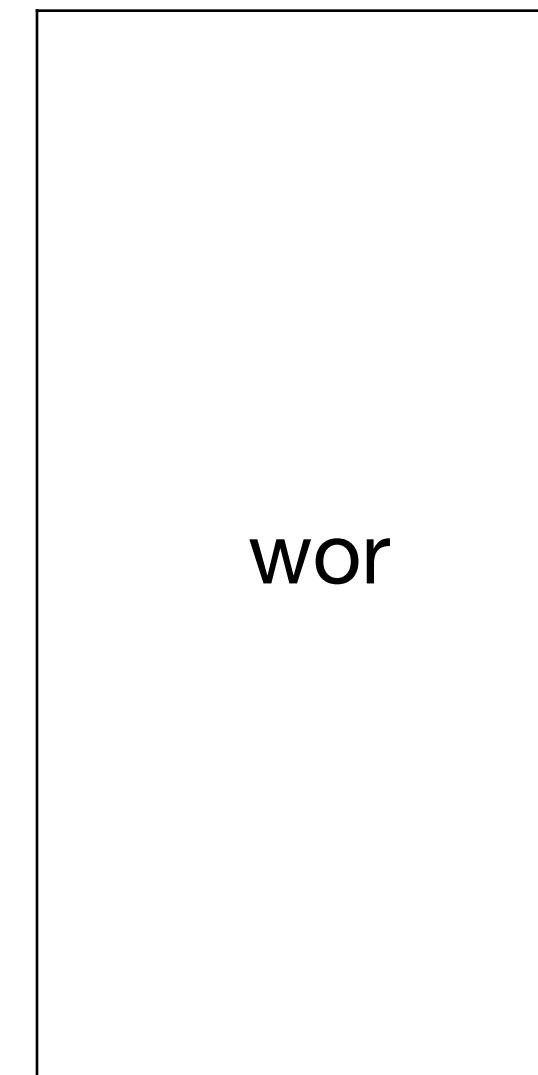
Block 3



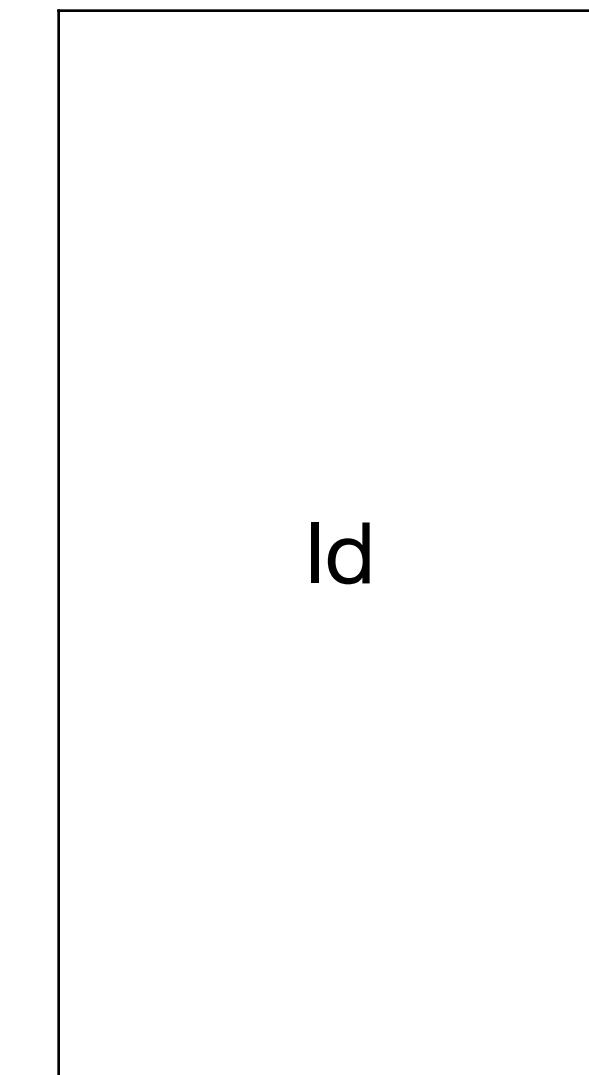
Block 5



Block 7



Block 12



Block 16

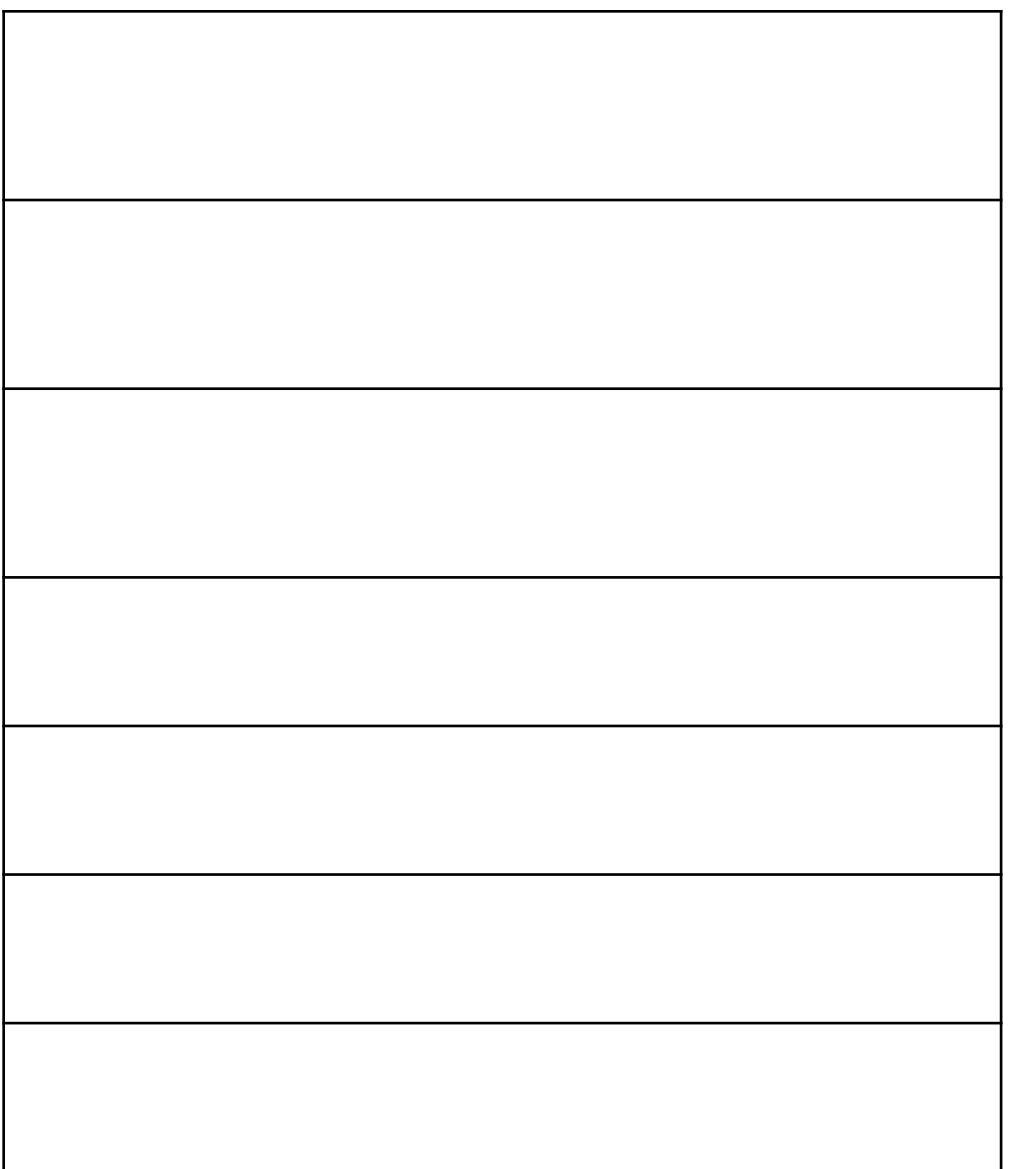
File/directory name	Inode number
.	8
..	1
bar	2

What can go wrong?

Missed file's inode block

- File data is present but not accessible via any file
- Leaked data blocks

Inode = 2

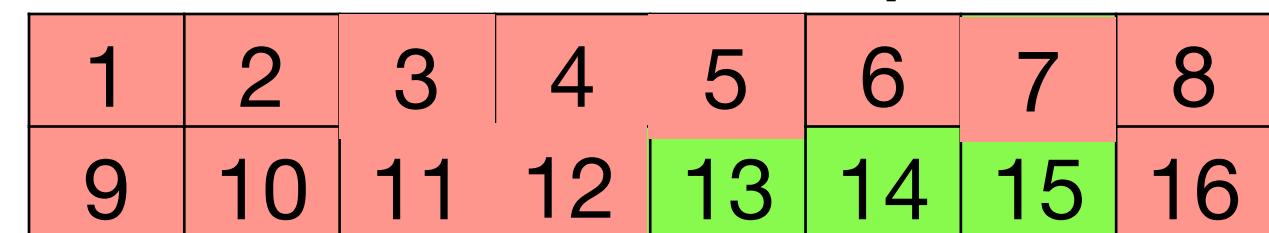


```
fd = open("/foo/bar", O_CREATE);
write(fd, "hello world\n", 12);
close(fd);
```

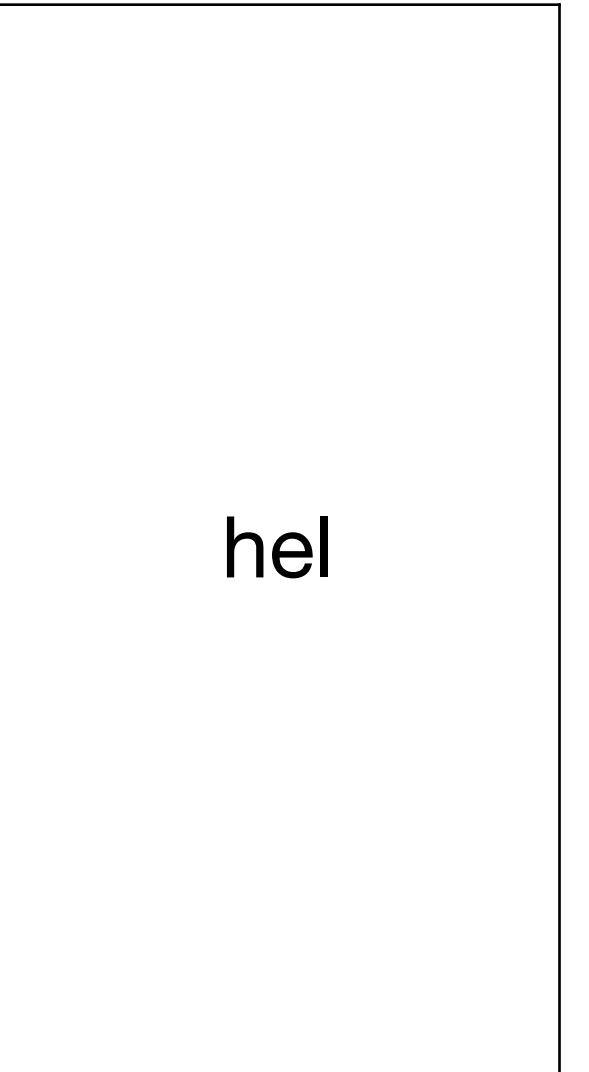
Inode bitmap



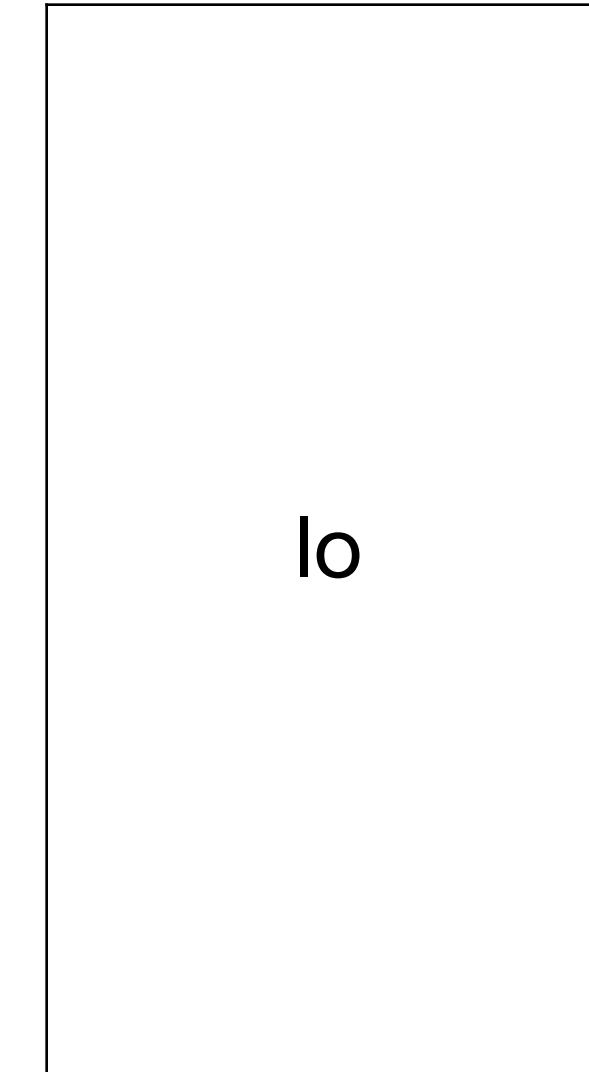
Data bitmap



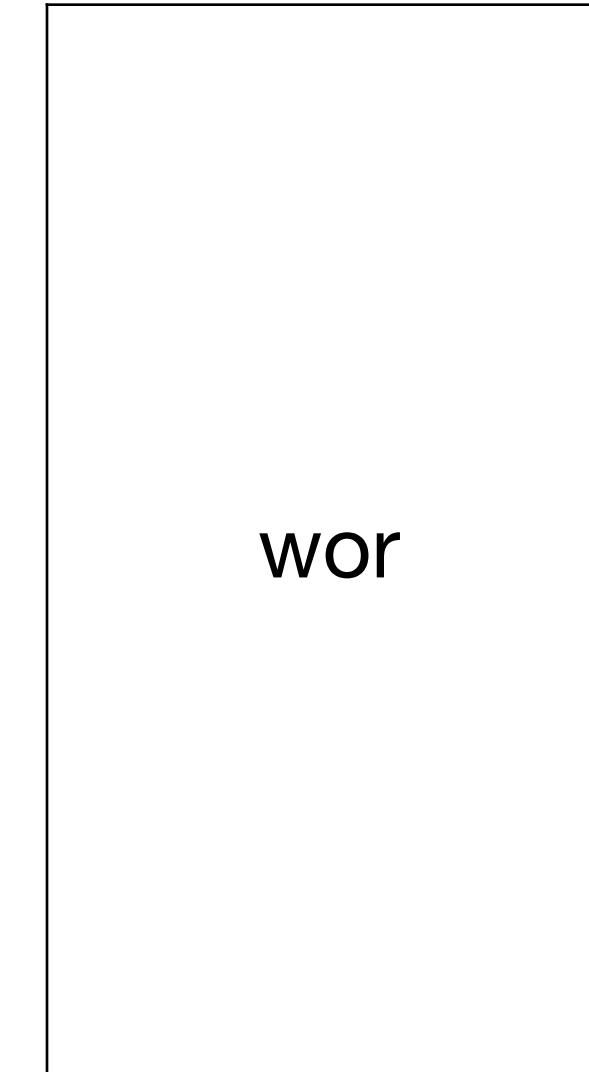
Block 3



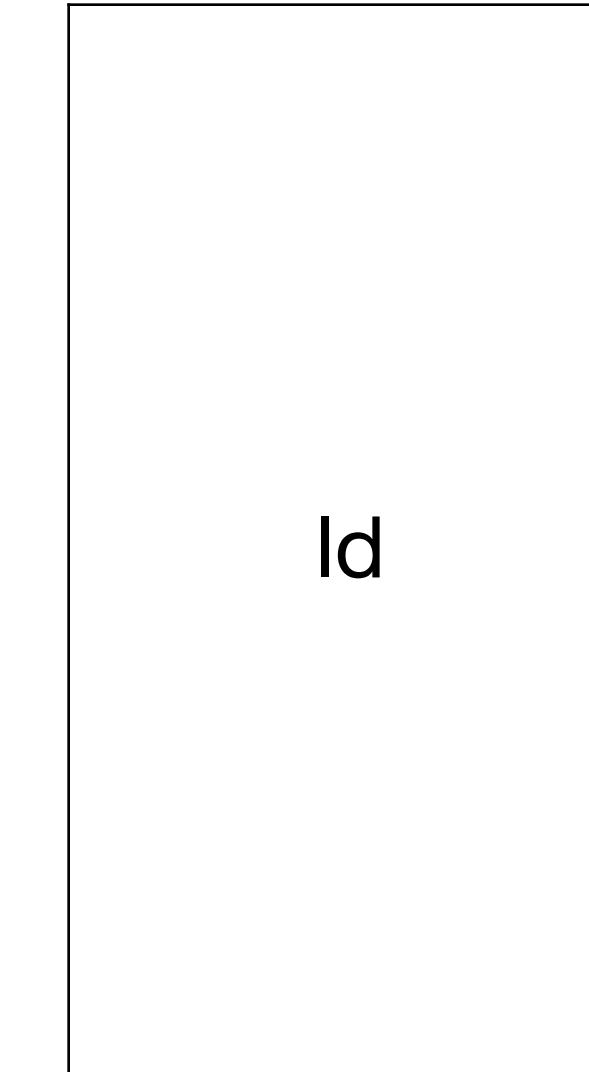
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

What can go wrong?

Missed file's inode block

- Directory points to a deleted file. Leaked sensitive information

Inode = 2 “/pass”

Type = file
Size
Modified time
14

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

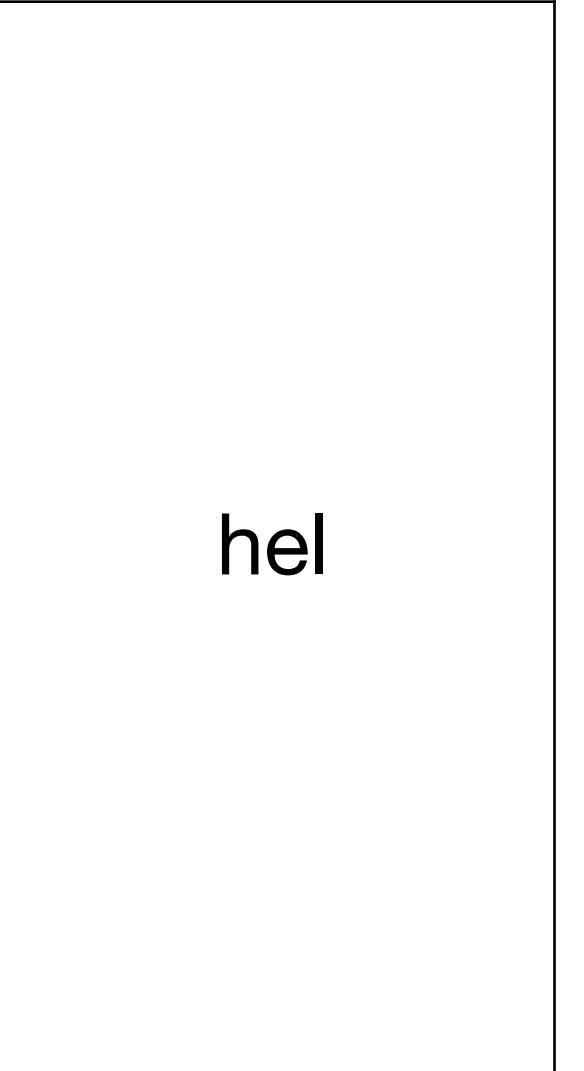
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

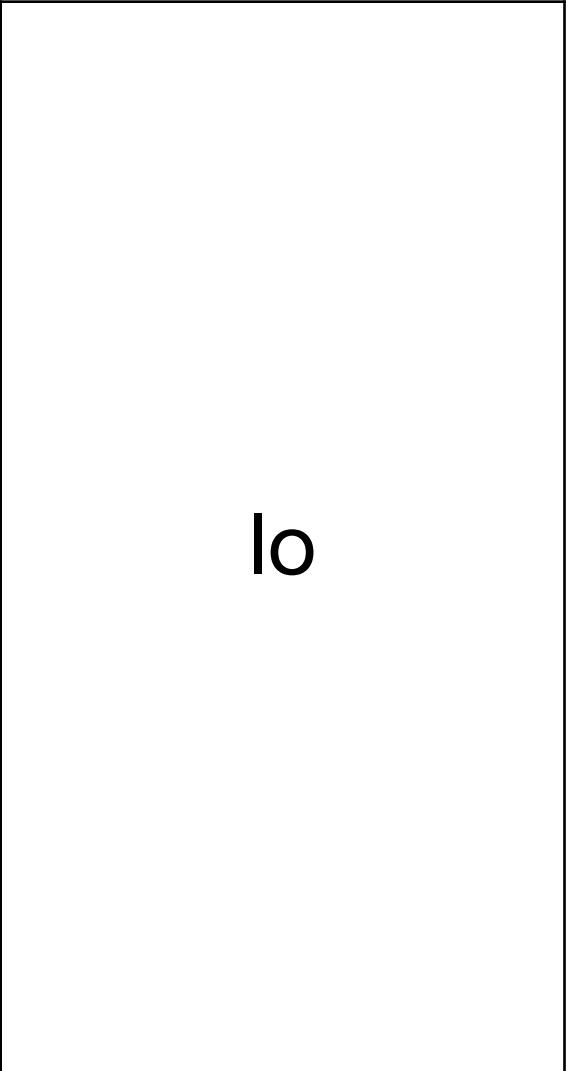
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

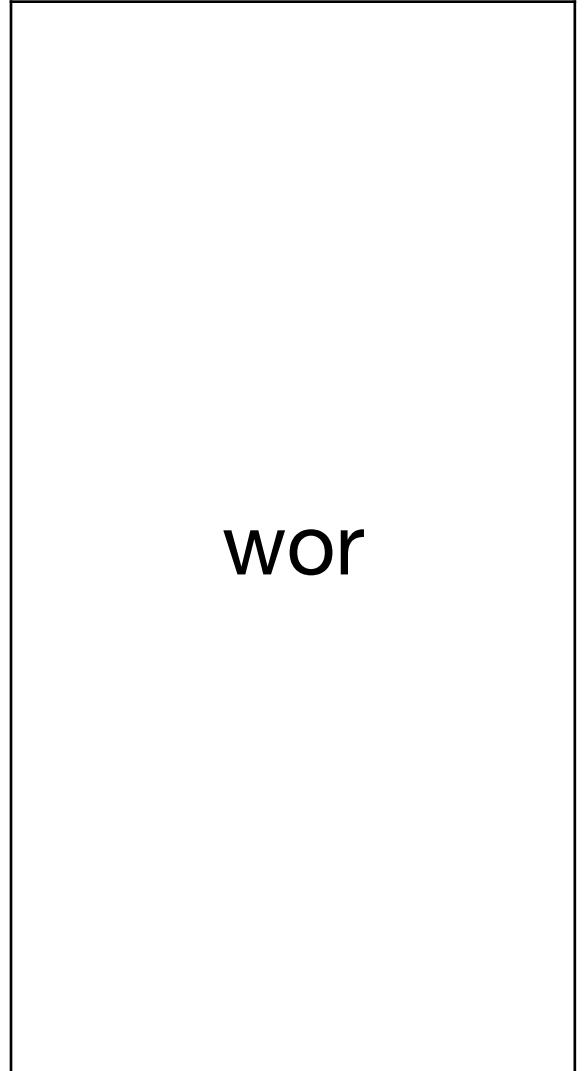
Block 3



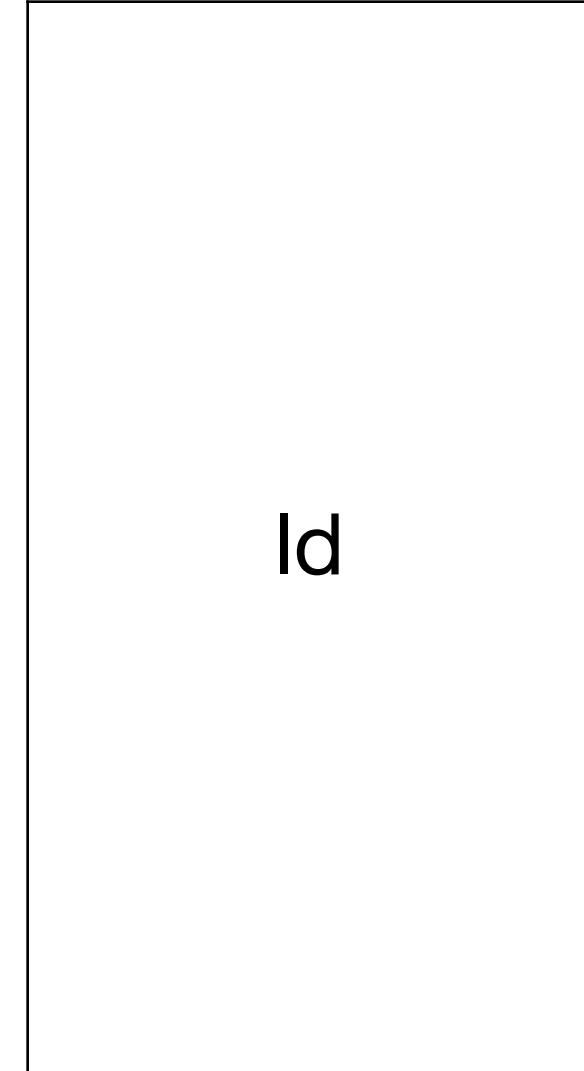
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

What can go wrong?

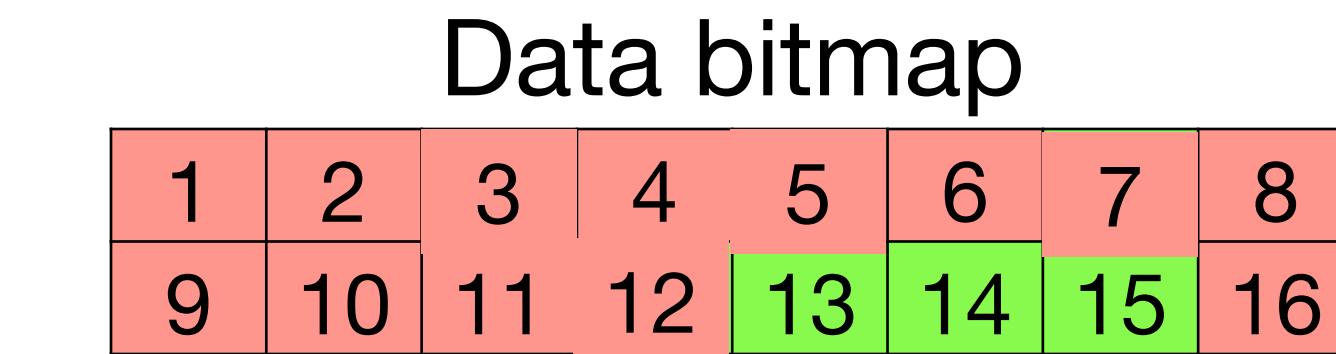
Missed parent's inode block

- File exists but cannot be reached
- Leaks file inode and data blocks

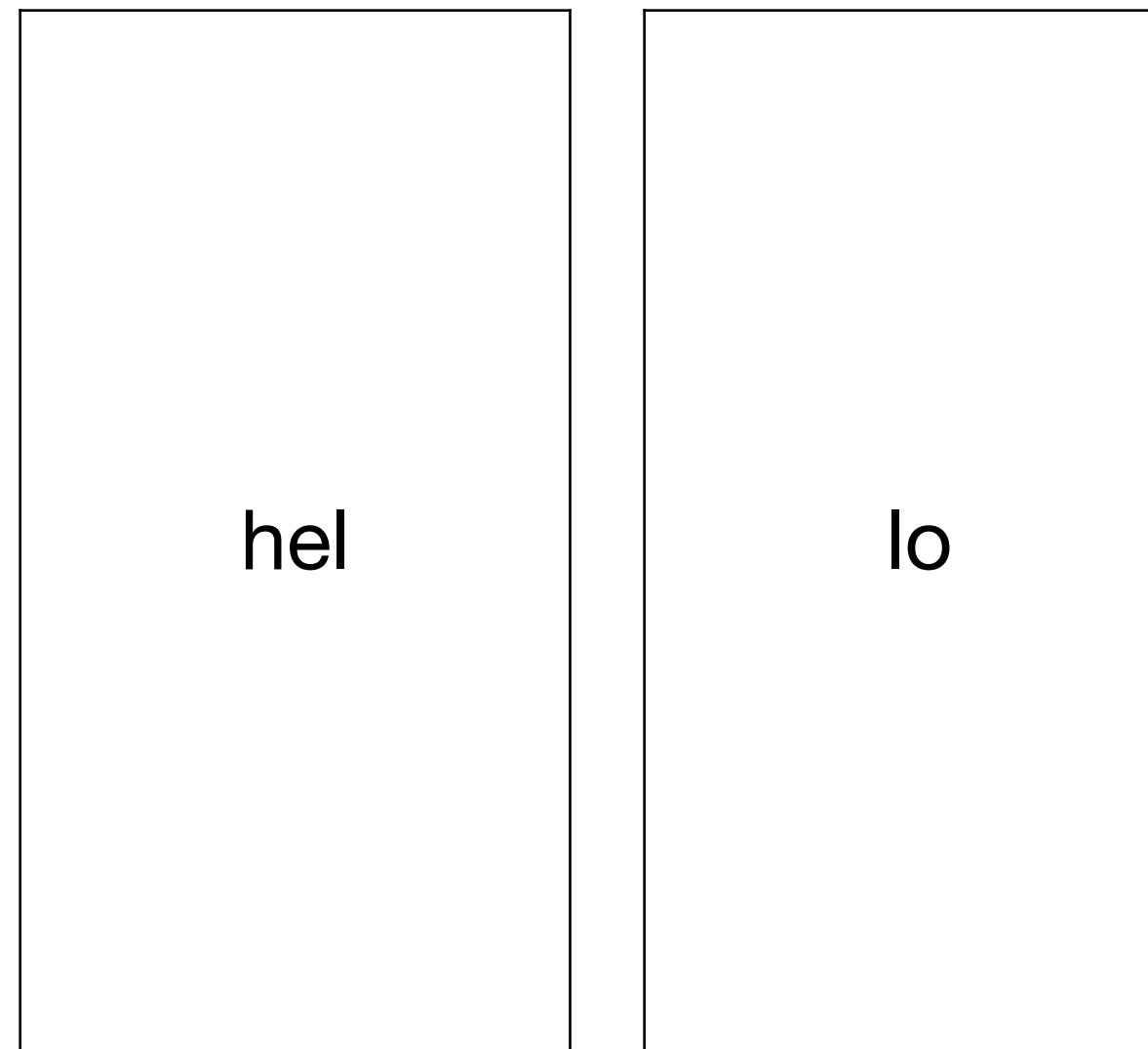
Block 16	
File/directory name	Inode number
.	8
..	1

Inode = 2 "/foo/bar"
Type = file
Size
Modified time
3
5
7
12

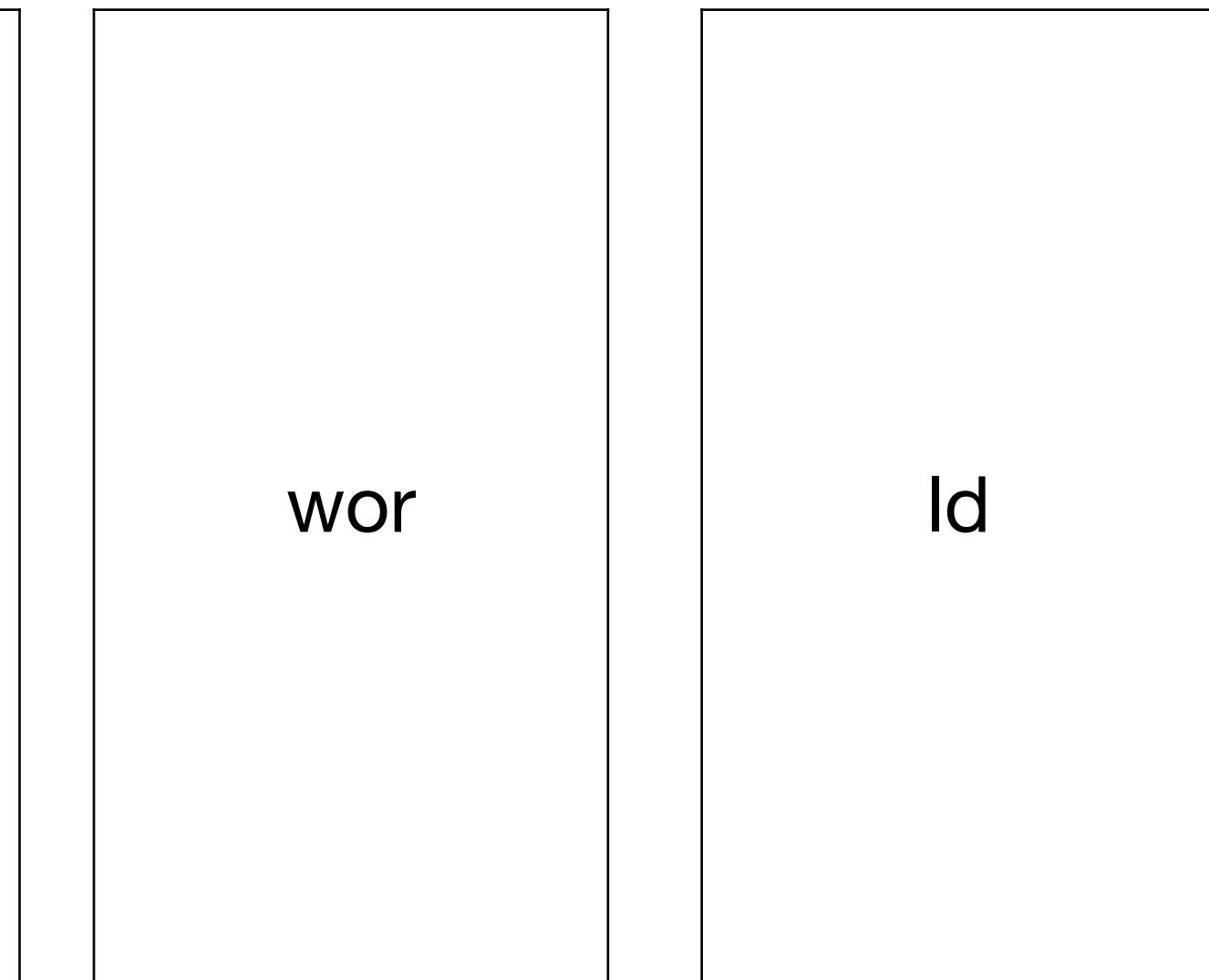
```
fd = open("/foo/bar", O_CREATE);
write(fd, "hello world\n", 12);
close(fd);
```



Block 3 Block 5



Block 7 Block 12



What can go wrong?

Missing inode bitmap

- Inode gets overwritten by another file => Lose file.
- Parent points to another file.
- Leaked data blocks.

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

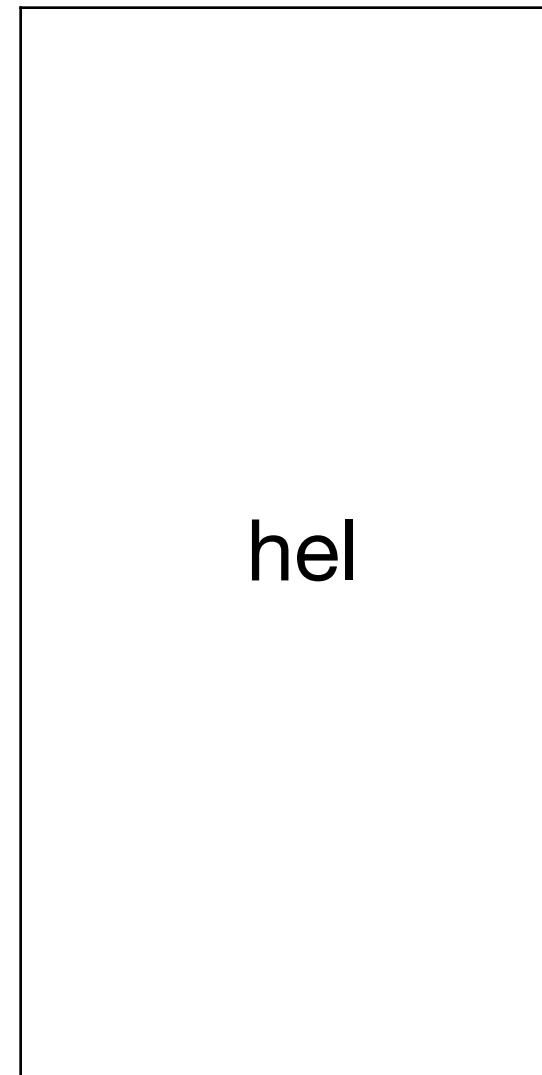
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

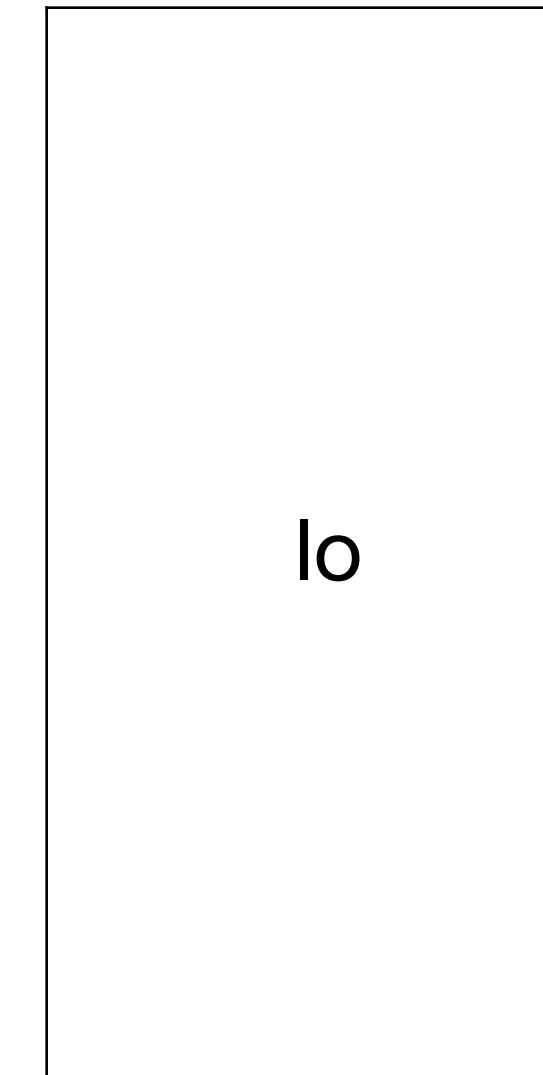
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Block 3



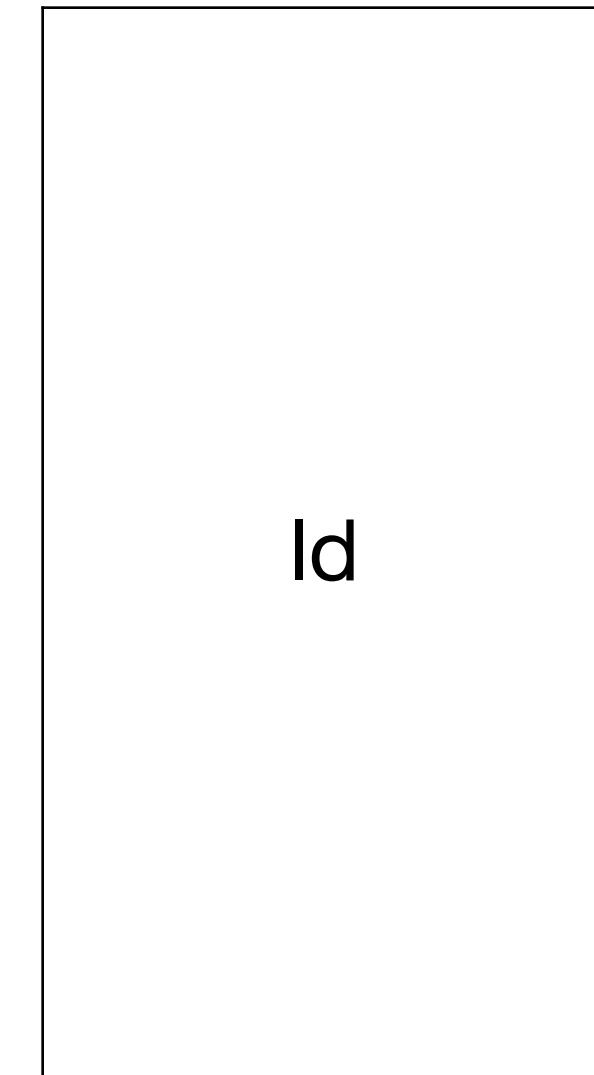
Block 5



Block 7



Block 12



Crash consistency

- File system stays “consistent” across power failures/kernel crashes

Inode = 2 “/foo/bar”

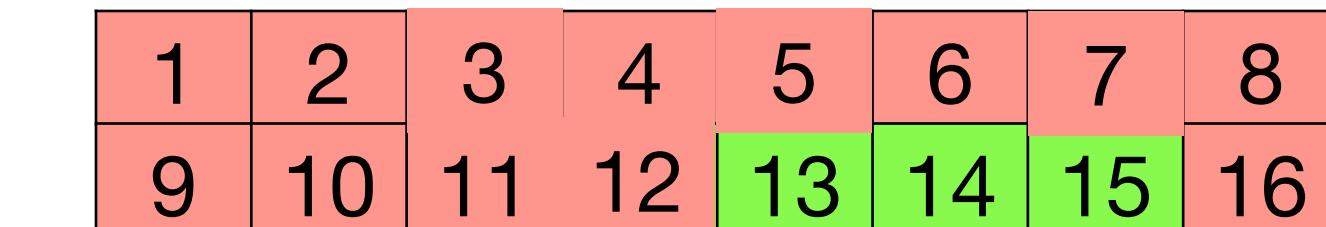
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE);
write(fd, "hello world\n", 12);
close(fd);
```

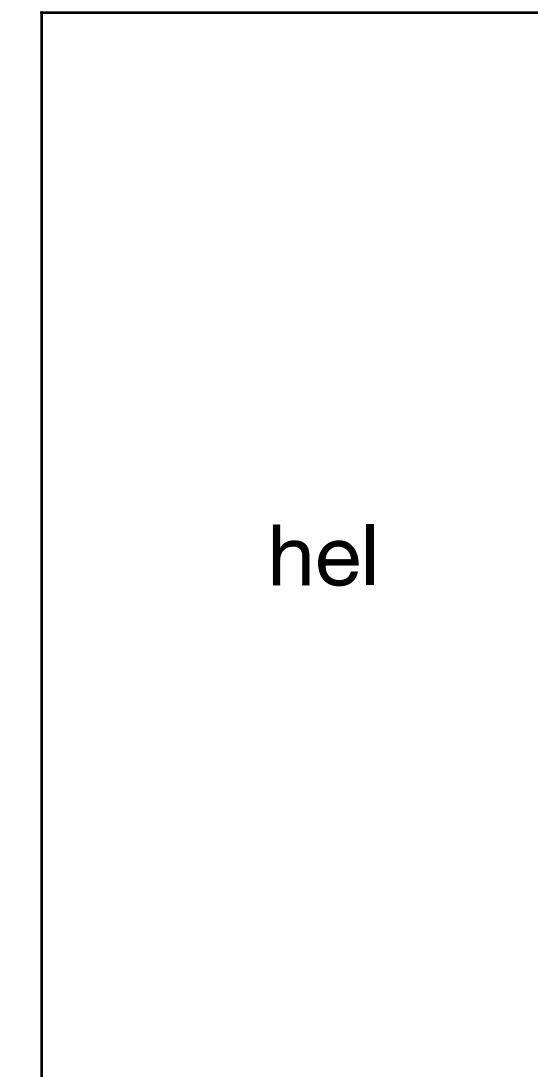
Inode bitmap



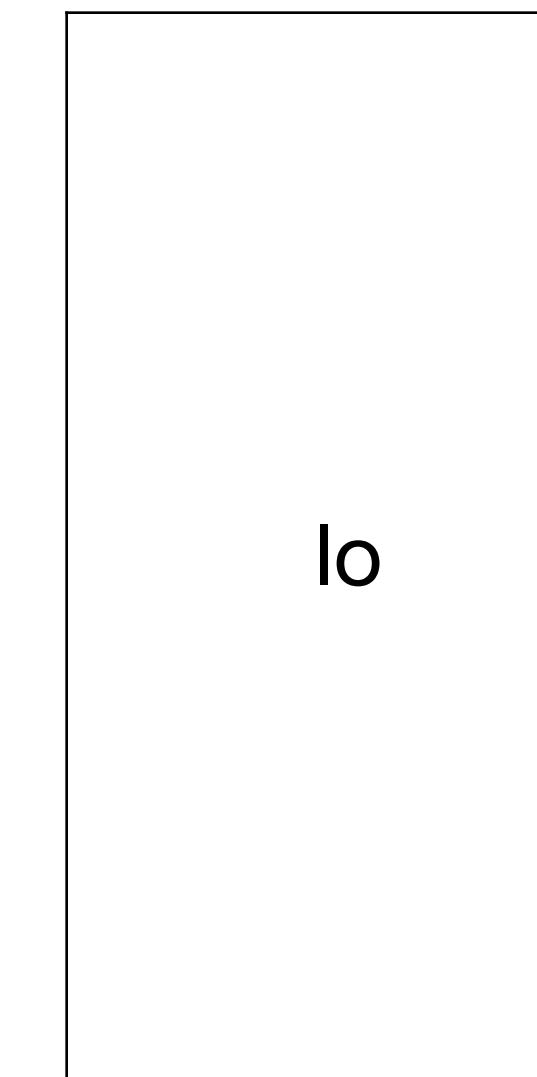
Data bitmap



Block 3



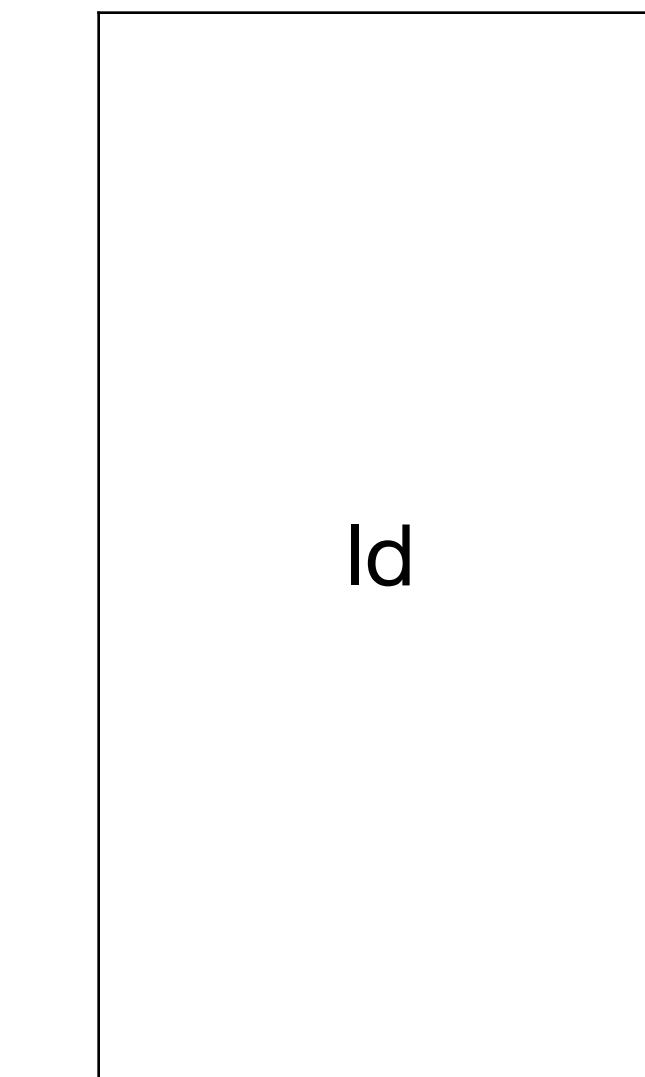
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Crash consistency

- File system stays “consistent” across power failures/kernel crashes
- Challenge: Disk only writes one block at a time

Inode = 2 “/foo/bar”

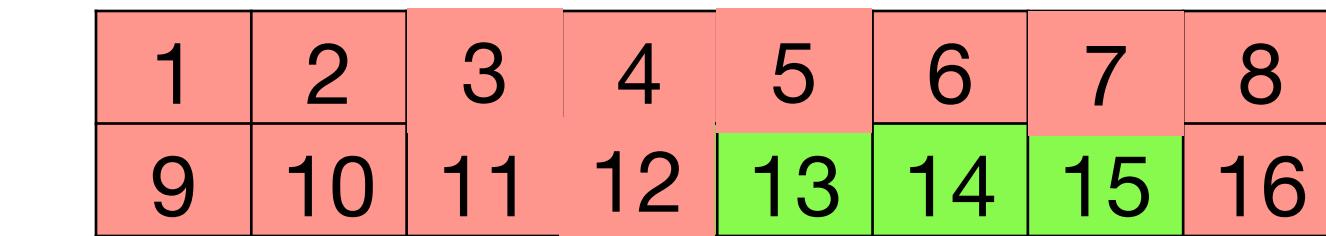
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

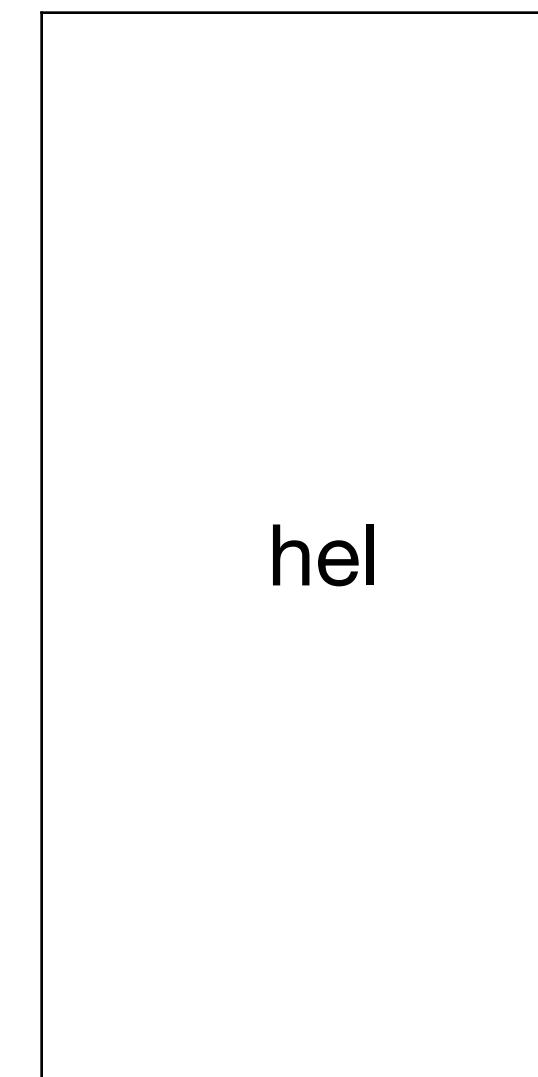
Inode bitmap



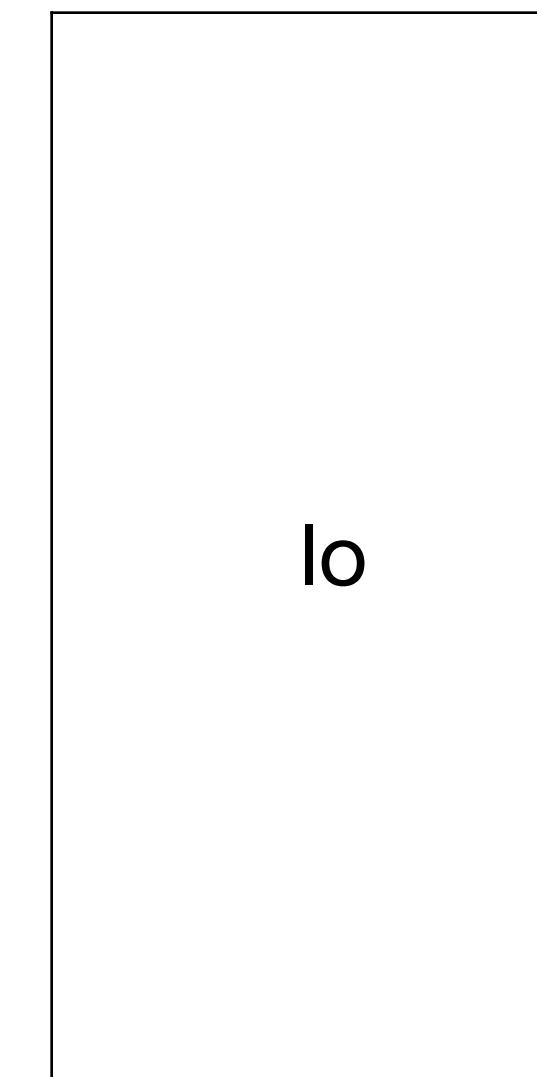
Data bitmap



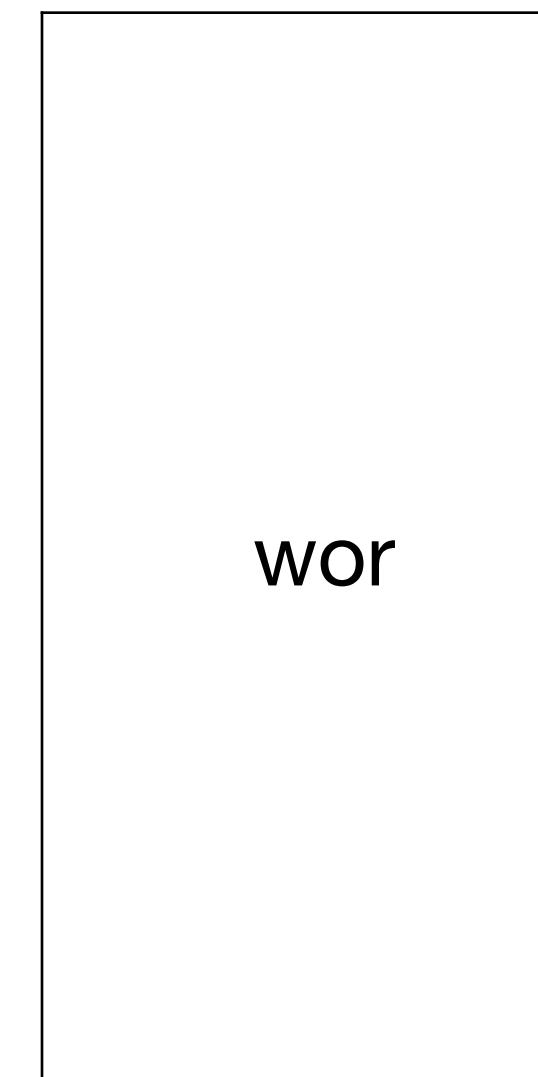
Block 3



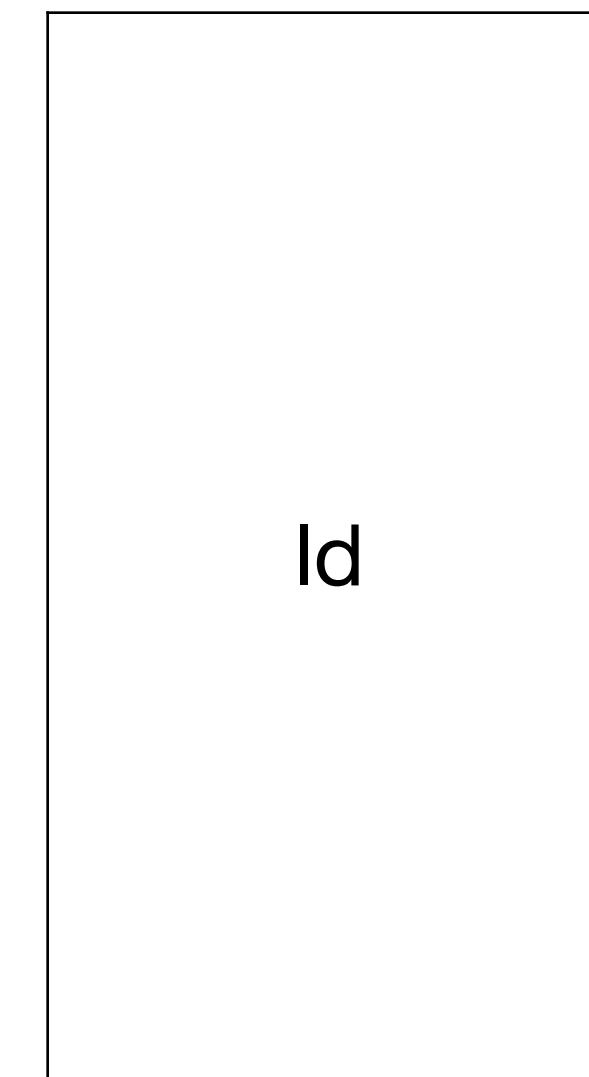
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Durability guarantees

```
fd = open("/foo/bar", O_CREATE)
write(fd, ...);
write(fd, ...);
write(fd, ...);
close(fd);
```

Durability guarantees

- At every write call. Most conservative.

```
write(fd, ...) {  
    begin_txn(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_txn(..);  
}
```

```
fd = open("/foo/bar", O_CREATE)  
write(fd, ...);  
write(fd, ...);  
write(fd, ...);  
close(fd);
```

Durability guarantees

- At every write call. Most conservative.
 - Buffer cache is write-through. Terrible performance

```
write(fd, ...) {  
    begin_txn(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_txn(..);  
}
```

```
fd = open("/foo/bar", O_CREATE)  
write(fd, ...);  
write(fd, ...);  
write(fd, ...);  
close(fd);
```

Durability guarantees

- At every write call. Most conservative.
 - Buffer cache is write-through. Terrible performance
- At close. Example: network file systems.

```
write(fd, ...) {  
    begin_txn(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_txn(..);  
}
```

```
fd = open("/foo/bar", O_CREATE)  
write(fd, ...);  
write(fd, ...);  
write(fd, ...);  
close(fd);
```

Durability guarantees

- At every write call. Most conservative.
 - Buffer cache is write-through. Terrible performance
- At close. Example: network file systems.
 - Long file operations increase risk of loss. Writing many small files is slow.

```
write(fd, ...) {  
    begin_txn(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_txn(..);  
}
```

```
fd = open("/foo/bar", O_CREATE)  
write(fd, ...);  
write(fd, ...);  
write(fd, ...);  
close(fd);
```

Durability guarantees

- At every write call. Most conservative.
 - Buffer cache is write-through. Terrible performance
- At close. Example: network file systems.
 - Long file operations increase risk of loss. Writing many small files is slow.
- At some point in the future (within 5 to 30 seconds)

```
write(fd, ...) {  
    begin_txn(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_txn(..);  
}
```

```
fd = open("/foo/bar", O_CREATE)  
write(fd, ...);  
write(fd, ...);  
write(fd, ...);  
close(fd);
```

Durability guarantees

- At every write call. Most conservative.
 - Buffer cache is write-through. Terrible performance
- At close. Example: network file systems.
 - Long file operations increase risk of loss. Writing many small files is slow.
- At some point in the future (within 5 to 30 seconds)
 - Most performant. Buffer cache is write-back.

```
write(fd, ...) {  
    begin_txn(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_txn(..);  
}
```

```
fd = open("/foo/bar", O_CREATE)  
write(fd, ...);  
write(fd, ...);  
write(fd, ...);  
close(fd);
```

Durability guarantees

- At every write call. Most conservative.
 - Buffer cache is write-through. Terrible performance
- At close. Example: network file systems.
 - Long file operations increase risk of loss. Writing many small files is slow.
- At some point in the future (within 5 to 30 seconds)
 - Most performant. Buffer cache is write-back.
 - One transaction contains several operations

```
write(fd, ...) {  
    begin_txn(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_txn(..);  
}
```

```
fd = open("/foo/bar", O_CREATE)  
write(fd, ...);  
write(fd, ...);  
write(fd, ...);  
close(fd);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

Durability guarantees

- At every write call. Most conservative.
 - Buffer cache is write-through. Terrible performance
 - At close. Example: network file systems.
 - Long file operations increase risk of loss. Writing many small files is slow.
 - At some point in the future (within 5 to 30 seconds)
 - Most performant. Buffer cache is write-back.
 - One transaction contains several operations
 - `fsync` to ensure that write is on disk. time `./fsync`

```
write(fd, ...) {  
    begin_txn(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_txn(..);  
}
```

```
fd = open("/foo/bar", O_CREATE)  
write(fd, ...);  
write(fd, ...);  
write(fd, ...);  
close(fd);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

Order writes

- Dangling pointers are very bad!
Can live with space leaks.

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

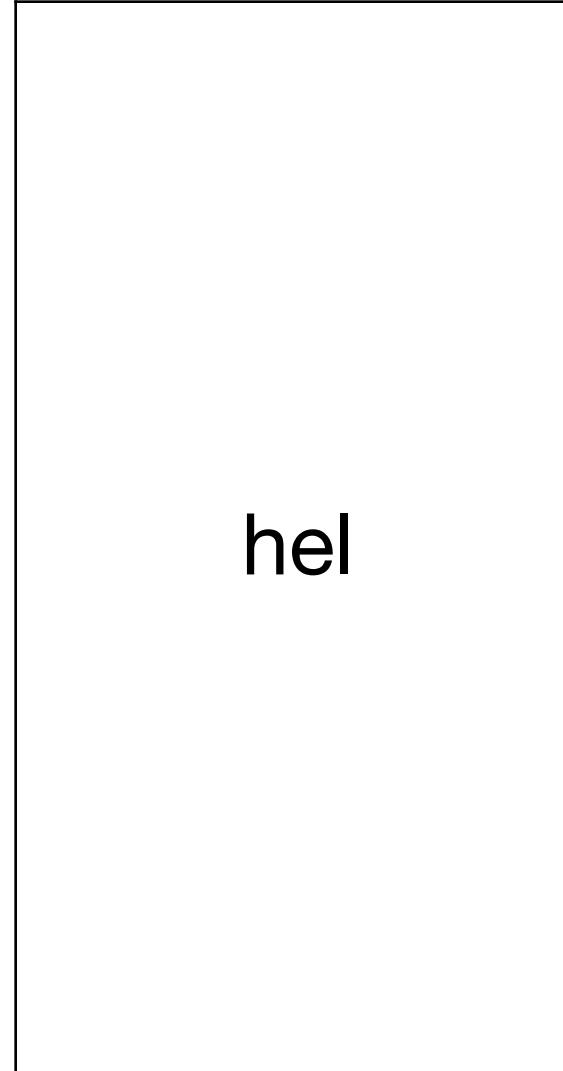
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

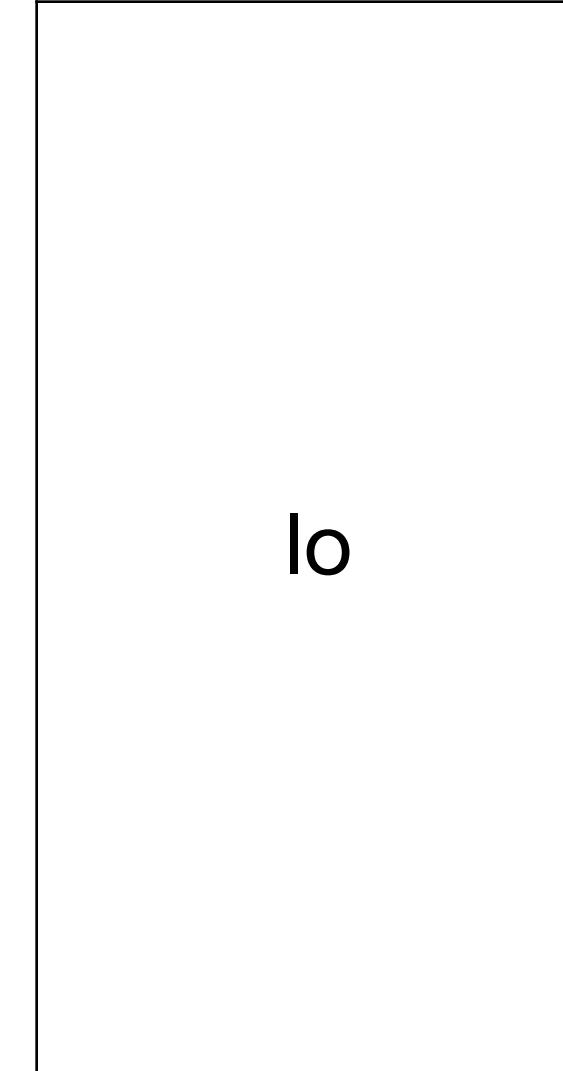
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

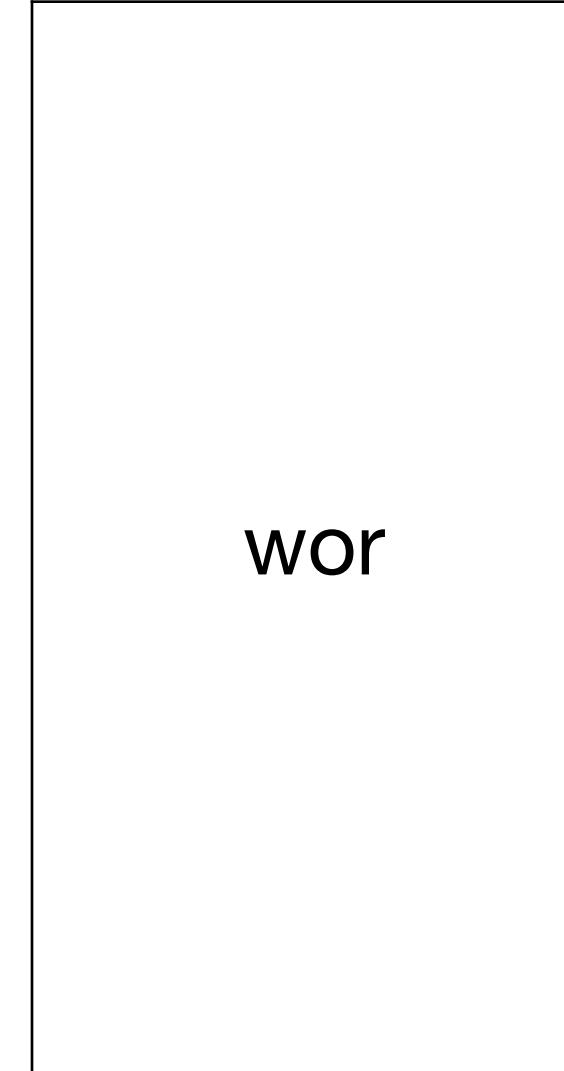
Block 3



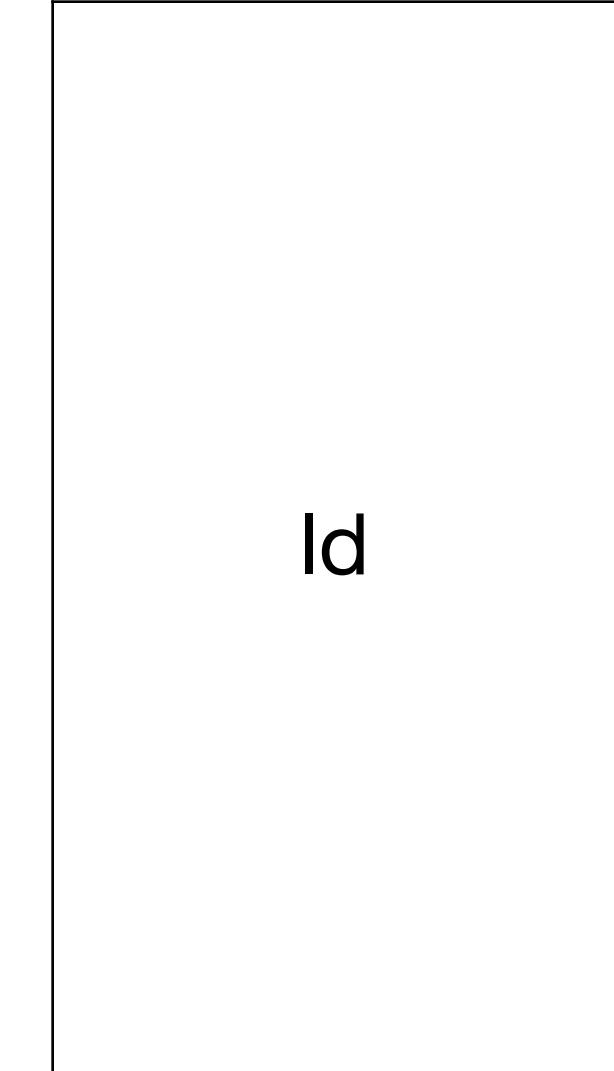
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Order writes

- Dangling pointers are very bad!
Can live with space leaks.
- Avoid dangling pointers by ordering writes. First write child, then parent

Inode = 2 “/foo/bar”

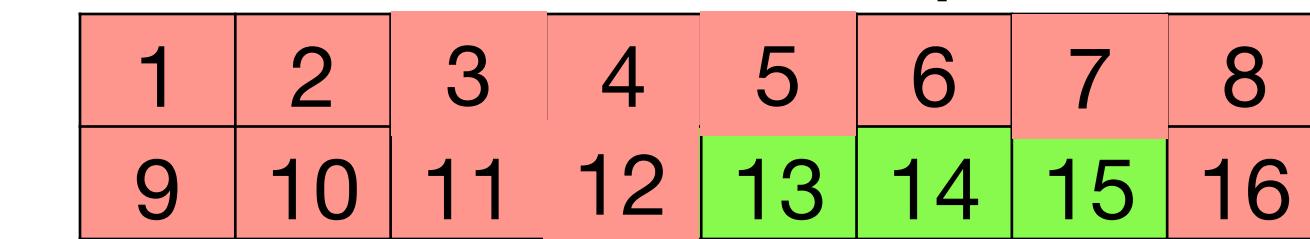
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

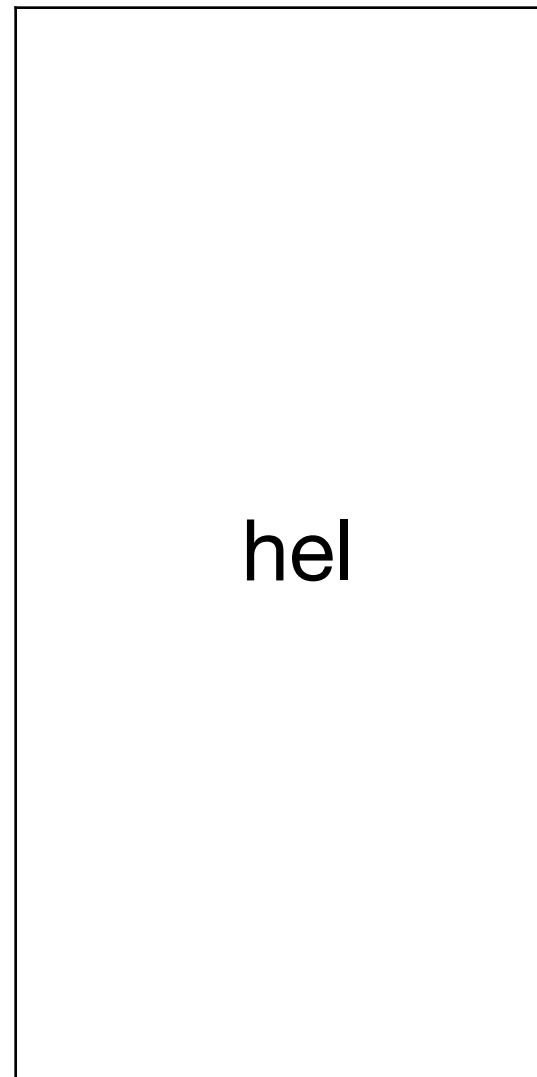
Inode bitmap



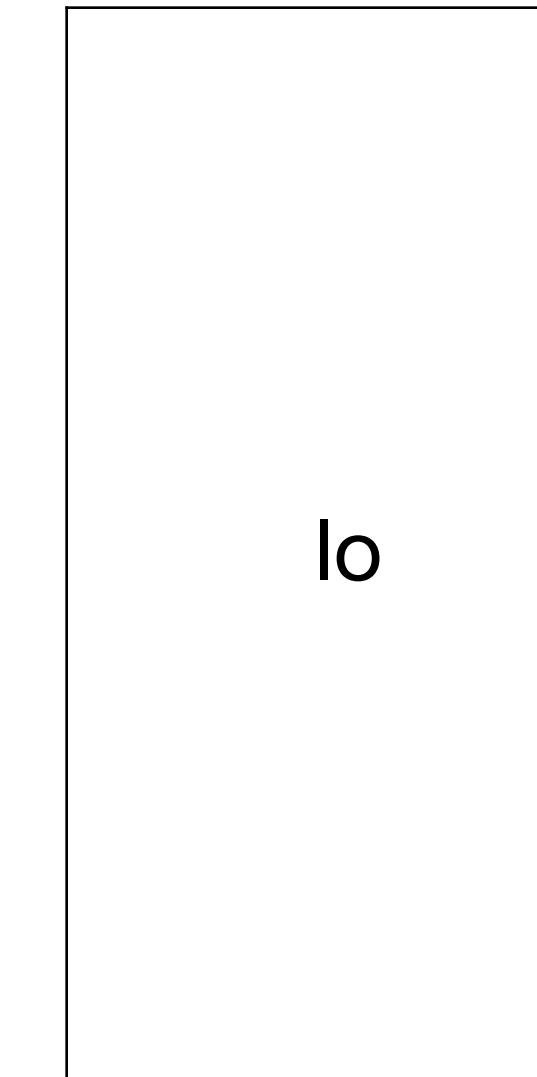
Data bitmap



Block 3



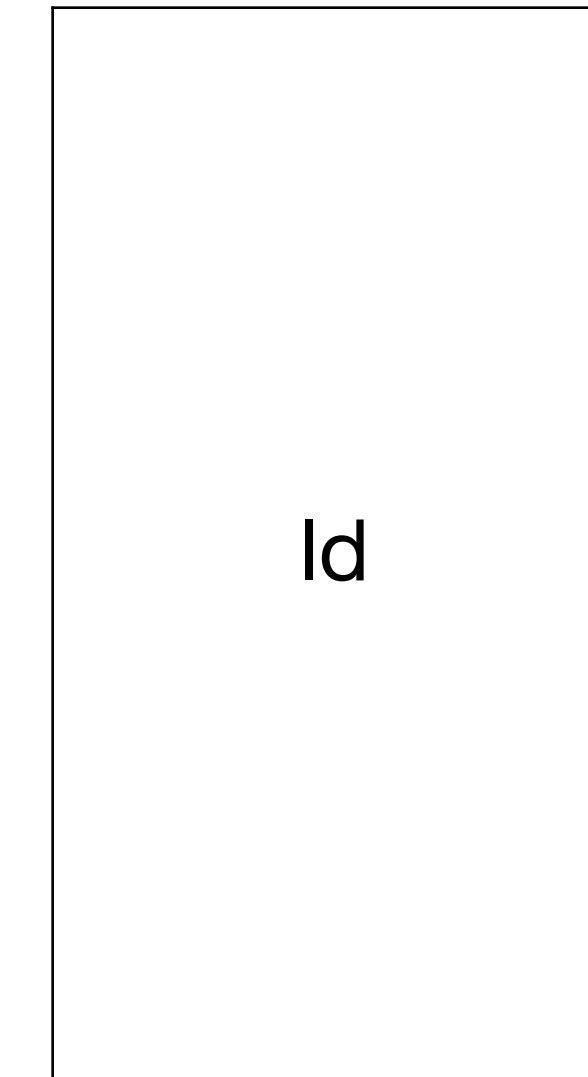
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Order writes

- Dangling pointers are very bad!
Can live with space leaks.
- Avoid dangling pointers by ordering writes. First write child, then parent

Inode = 2 “/foo/bar”

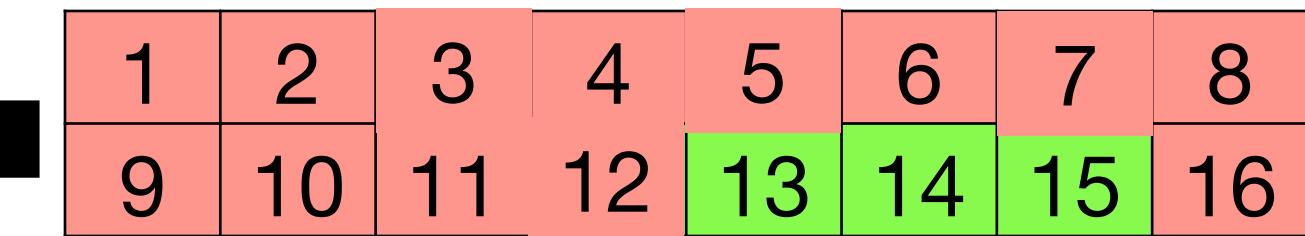
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

Inode bitmap



Data bitmap



Block 3

Block 5

Block 7

Block 12

Block 16

File/directory name	Inode number
.	8
..	1
bar	2

hel

lo

wor

ld

Order writes

- Dangling pointers are very bad!
Can live with space leaks.
- Avoid dangling pointers by ordering writes. First write child, then parent

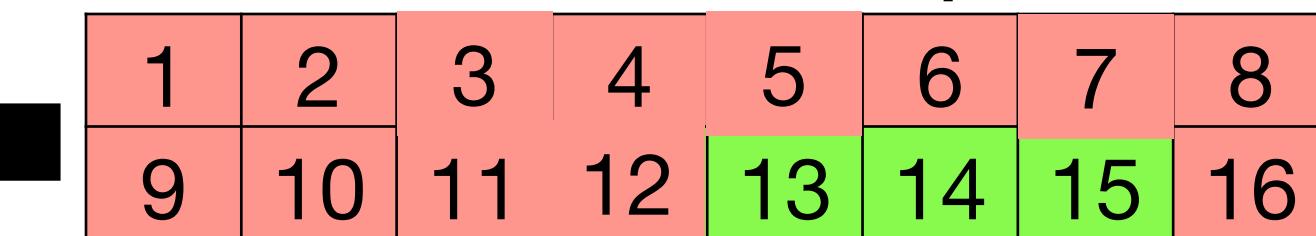
Inode = 2 “/foo/bar”
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

Inode bitmap



Data bitmap

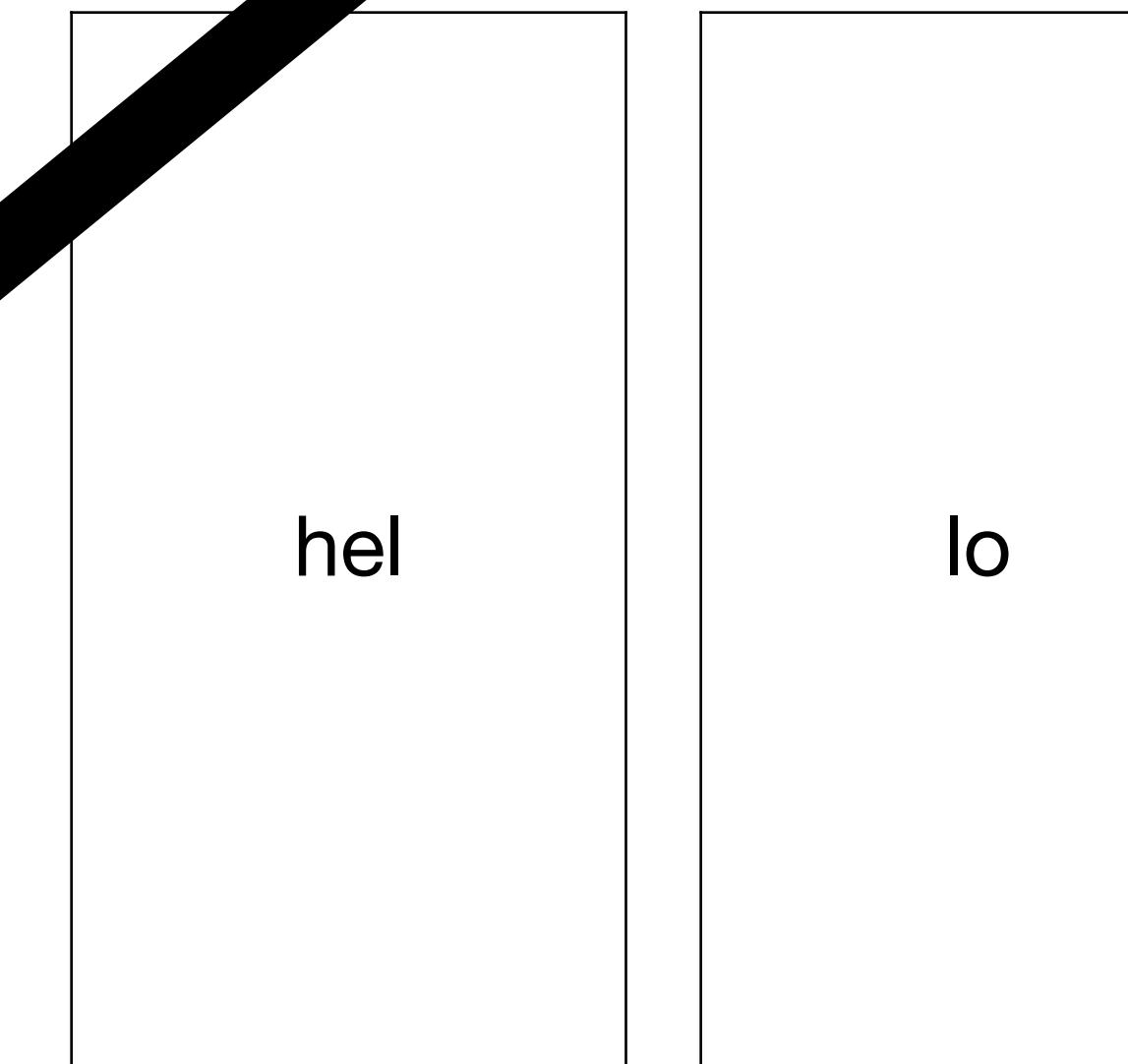


Block 3

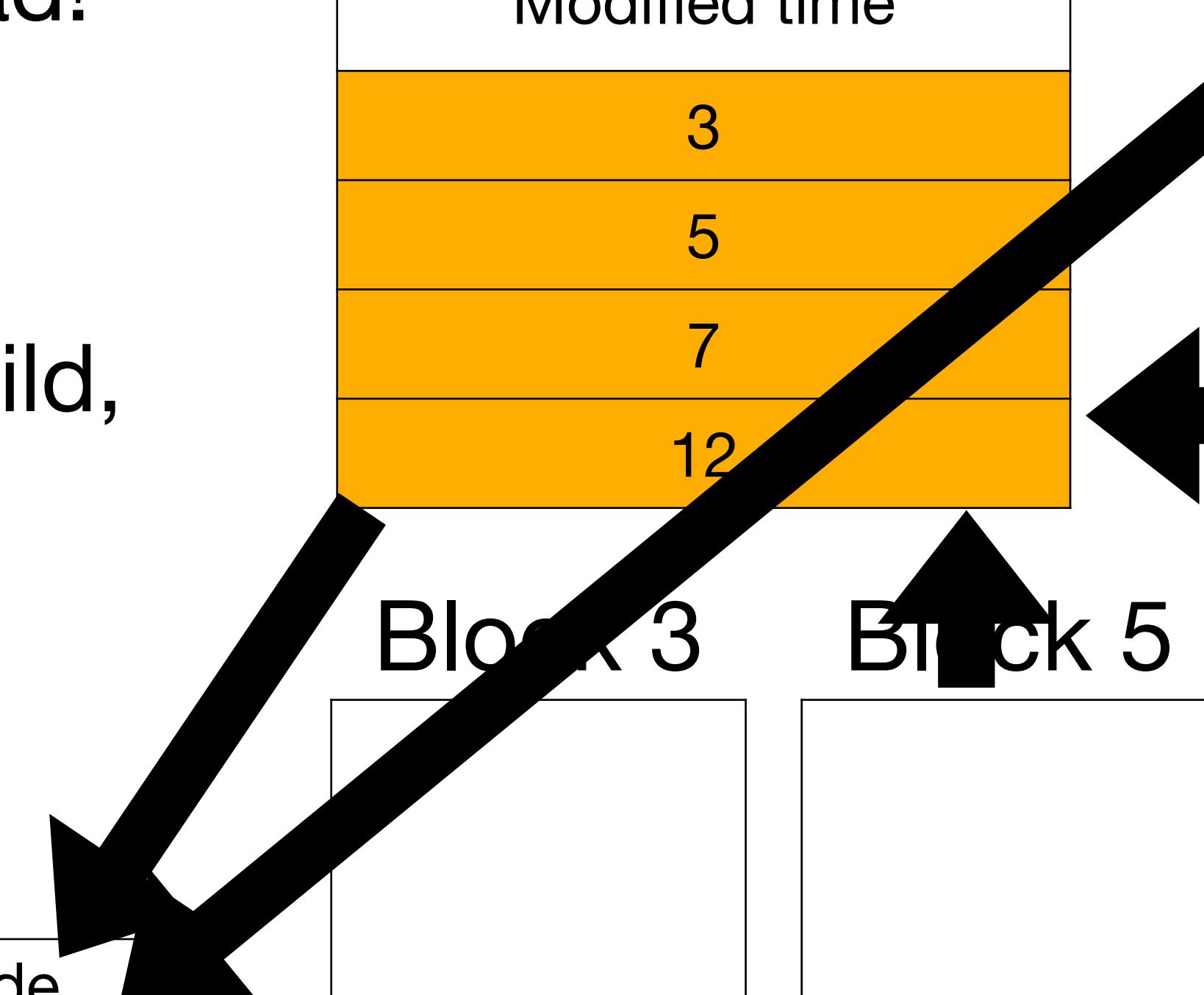
Block 5

Block 7

Block 12



File/directory name	Inode number
.	8
..	1
bar	2



Order writes

- Dangling pointers are very bad!
Can live with space leaks.
- Avoid dangling pointers by ordering writes. First write child, then parent
- Modern disks allow specifying ordering.

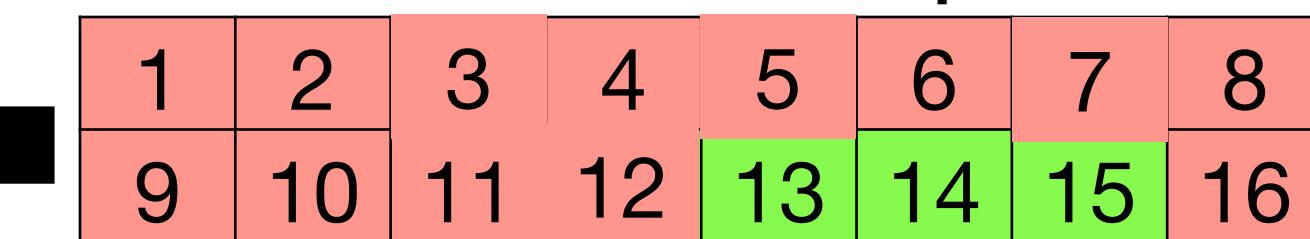
Inode = 2 “/foo/bar”
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

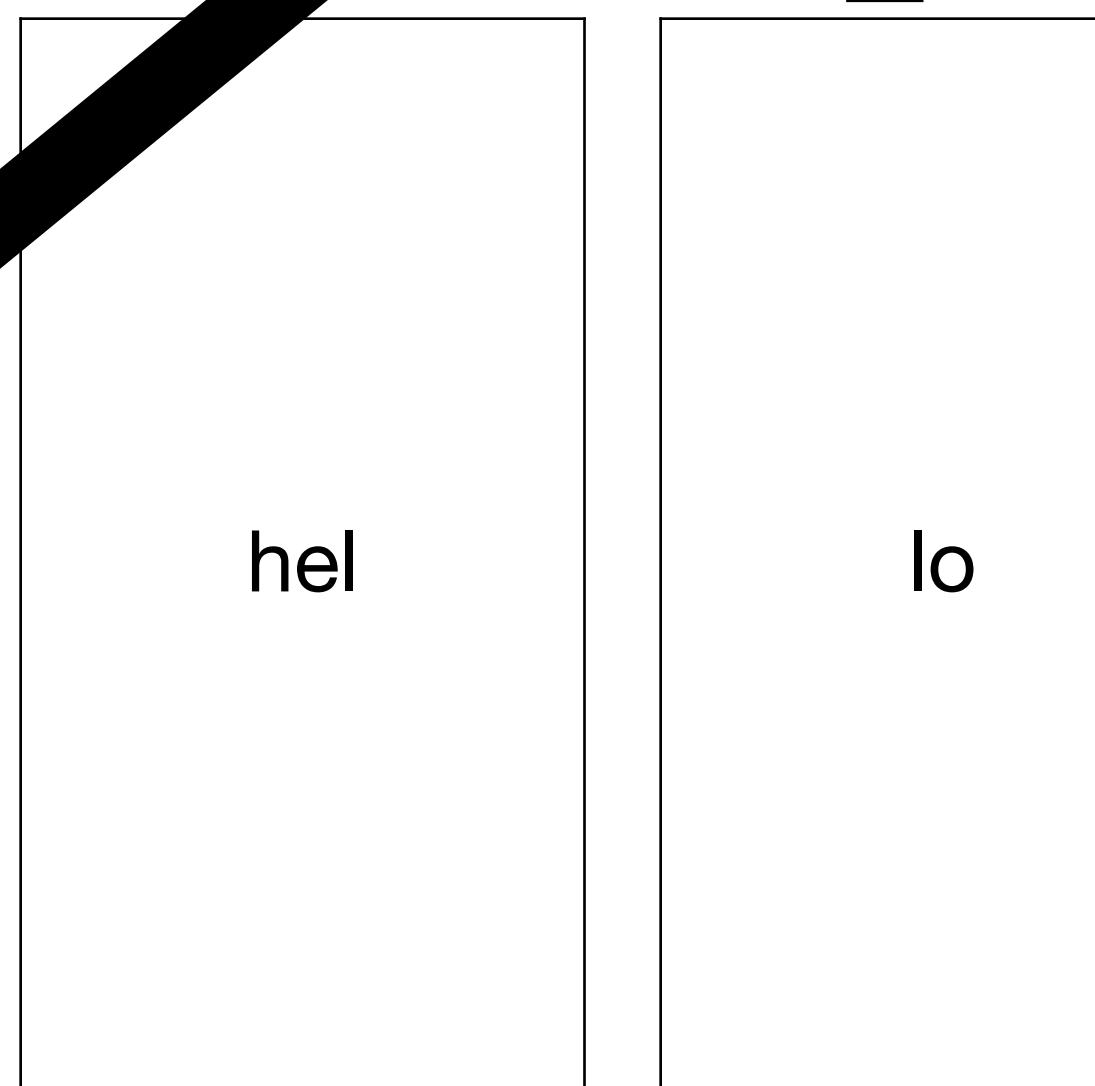
Inode bitmap



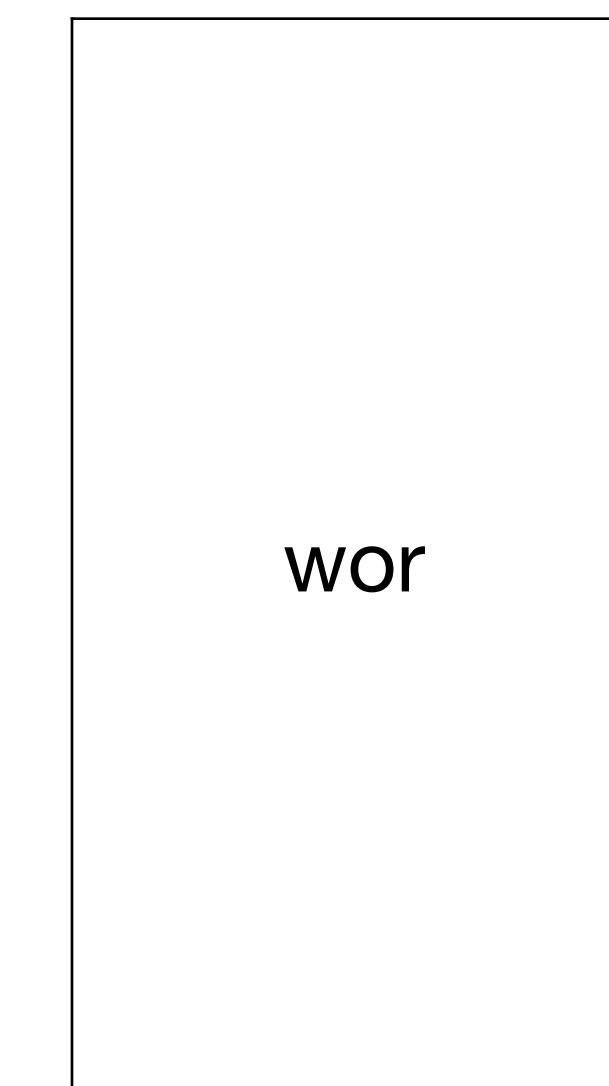
Data bitmap



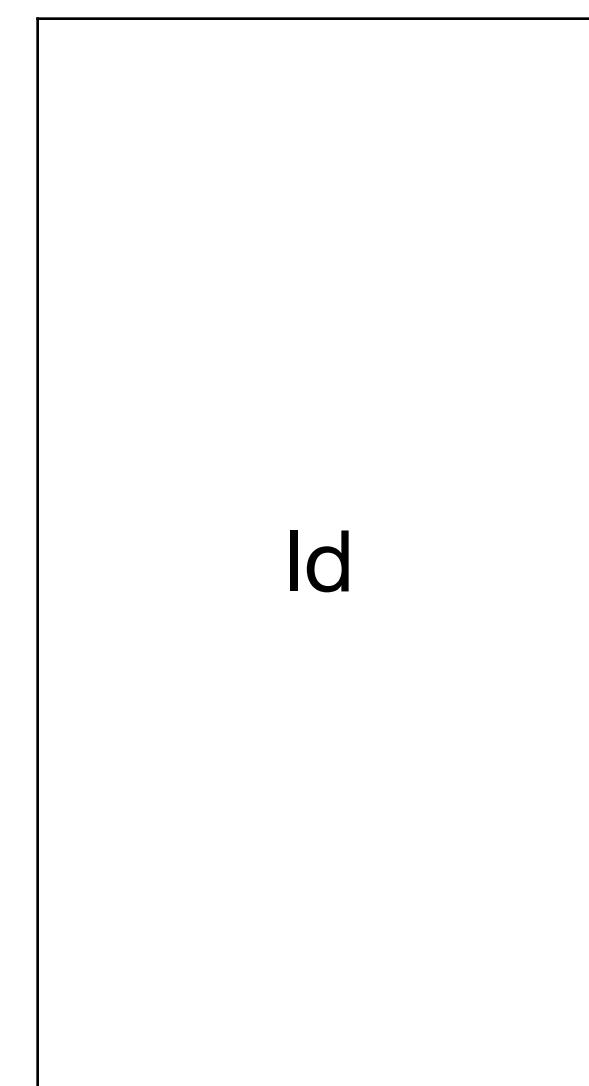
Block 3 Block 5



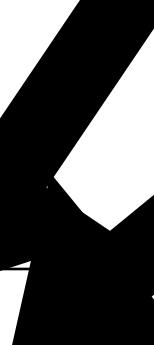
Block 7



Block 12



File/directory name	Inode number
.	8
..	1
bar	2



What about deletes?

- Truncate

Inode = 2 “/foo/bar”

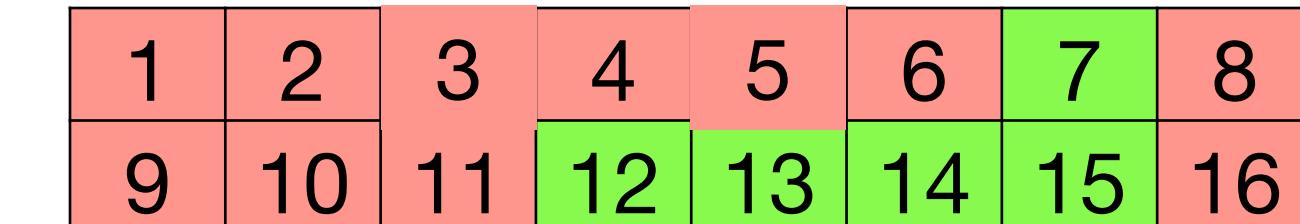
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

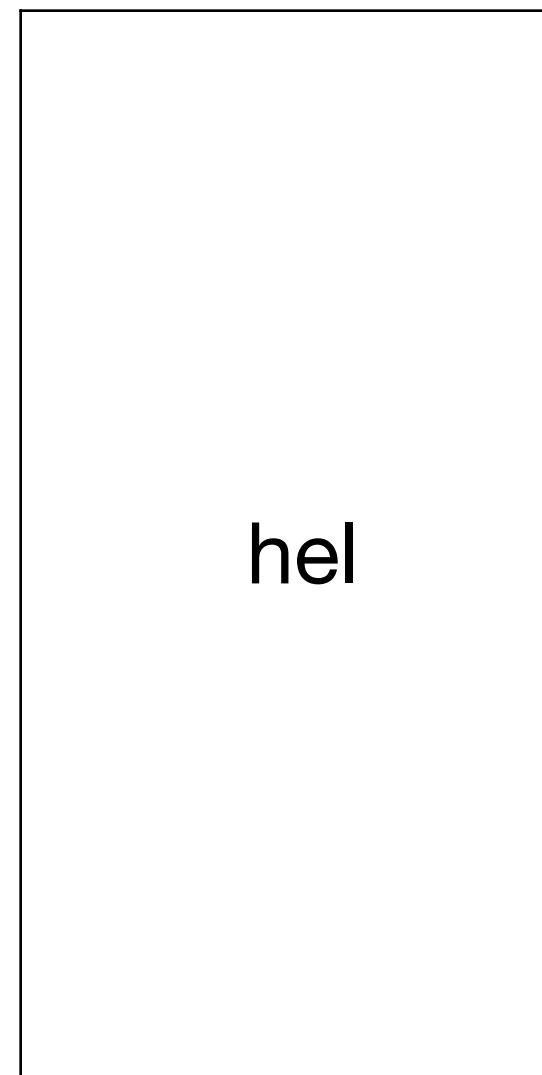
Inode bitmap



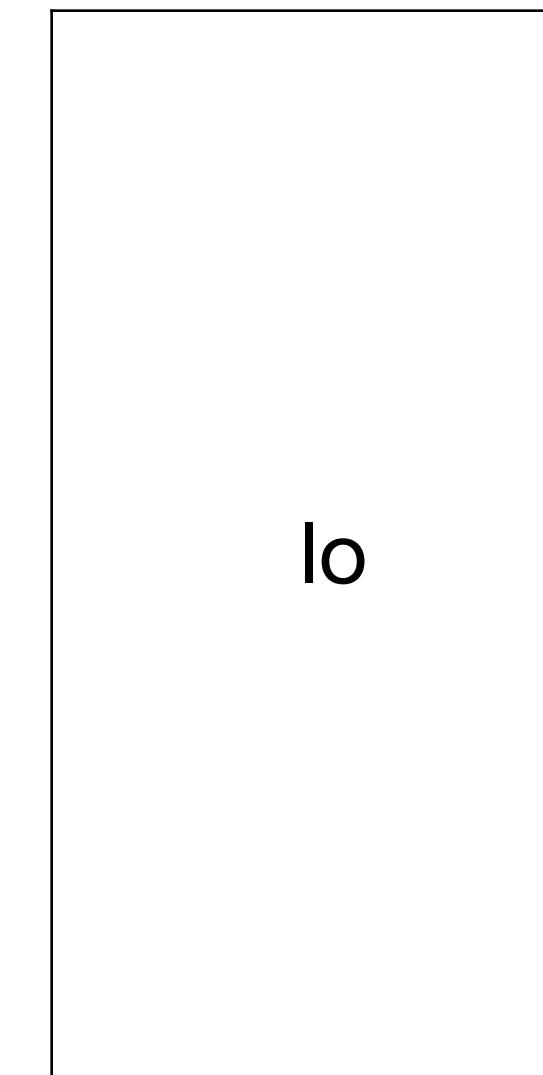
Data bitmap



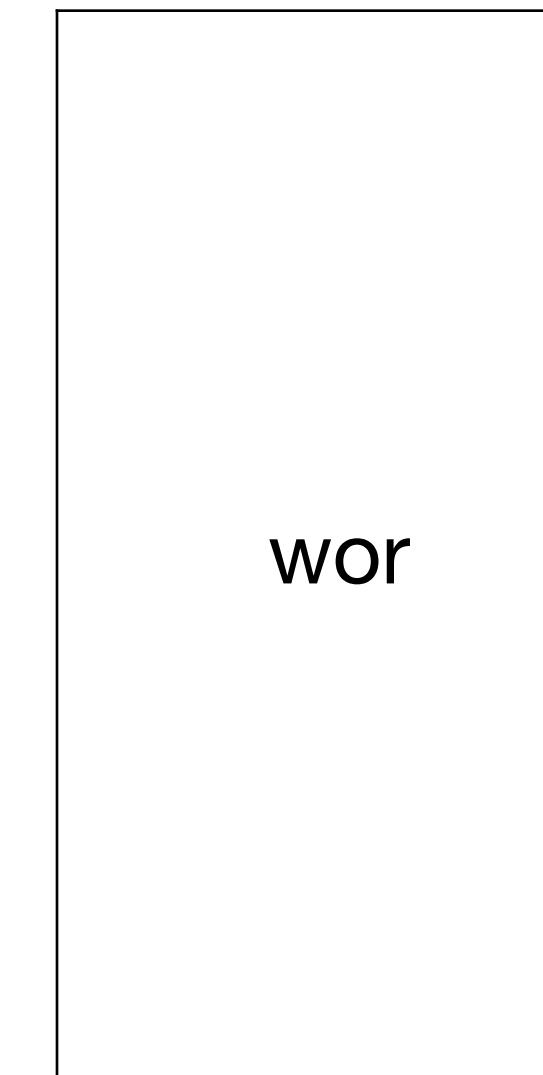
Block 3



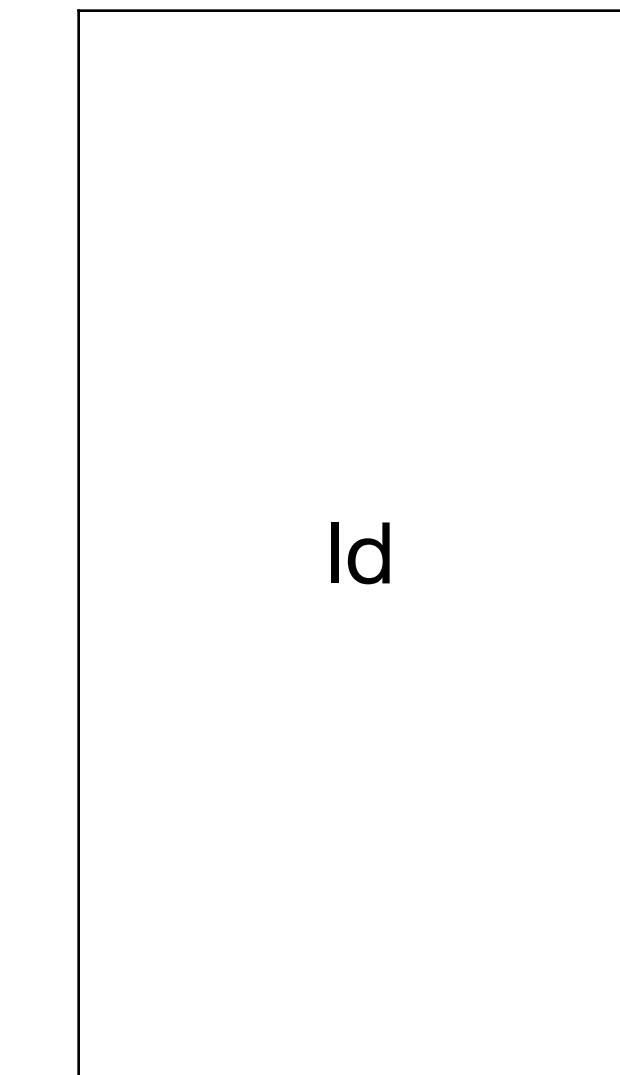
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

What about deletes?

- Truncate

Inode = 2 “/foo/bar”

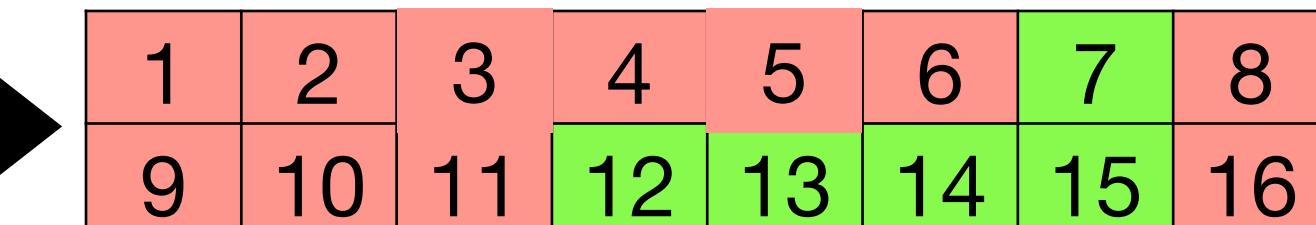
Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

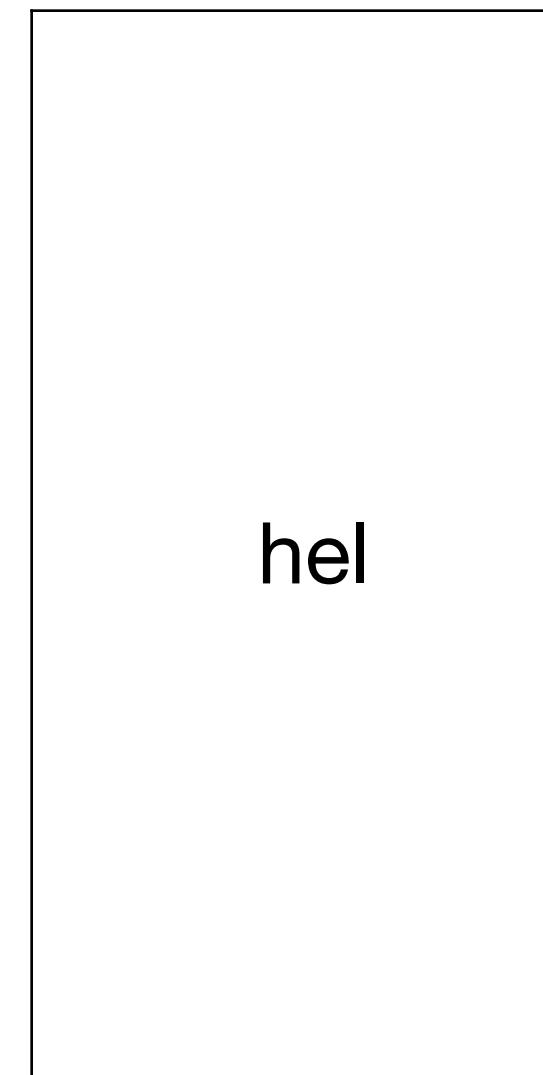
Inode bitmap



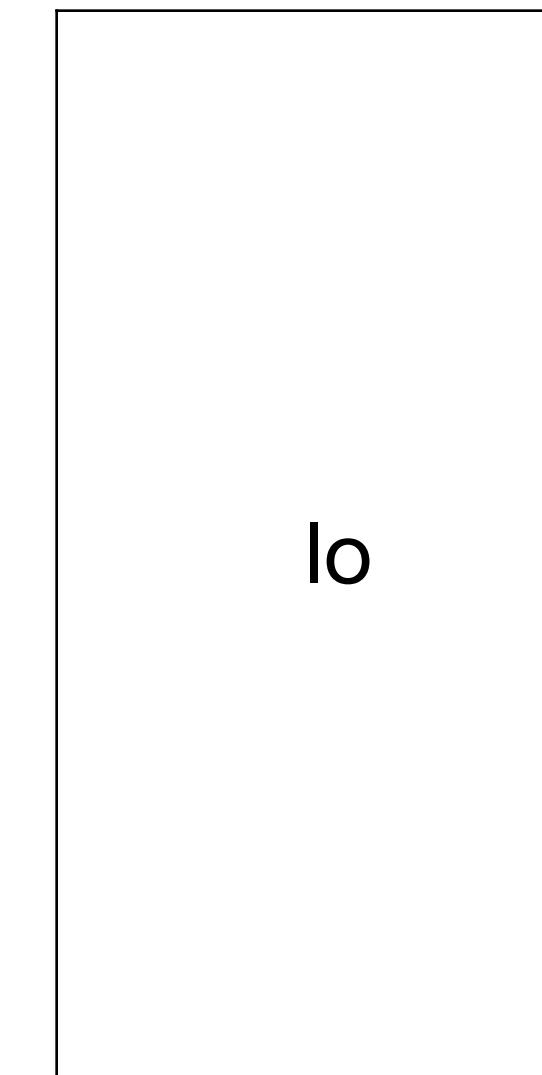
Data bitmap



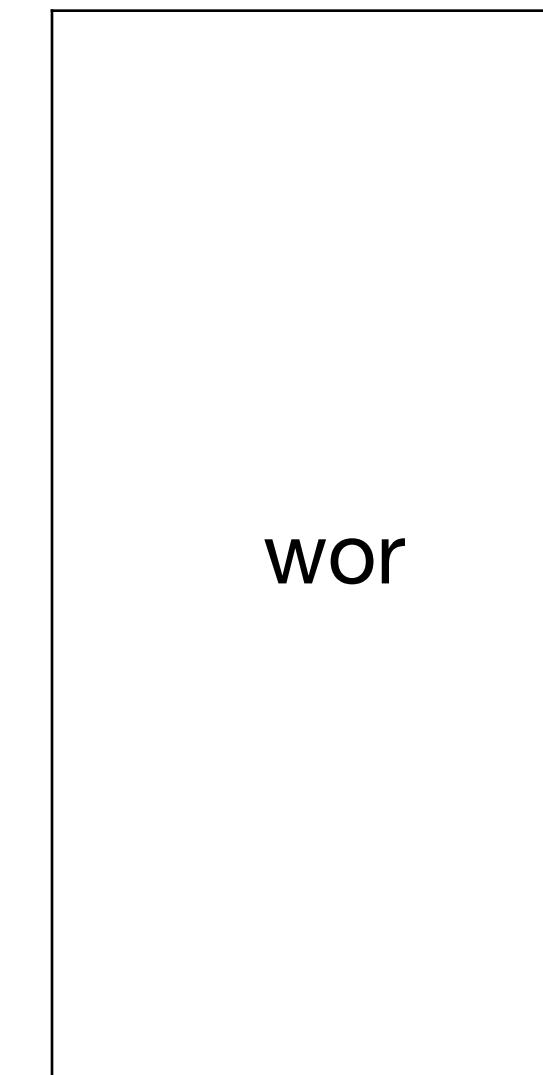
Block 3



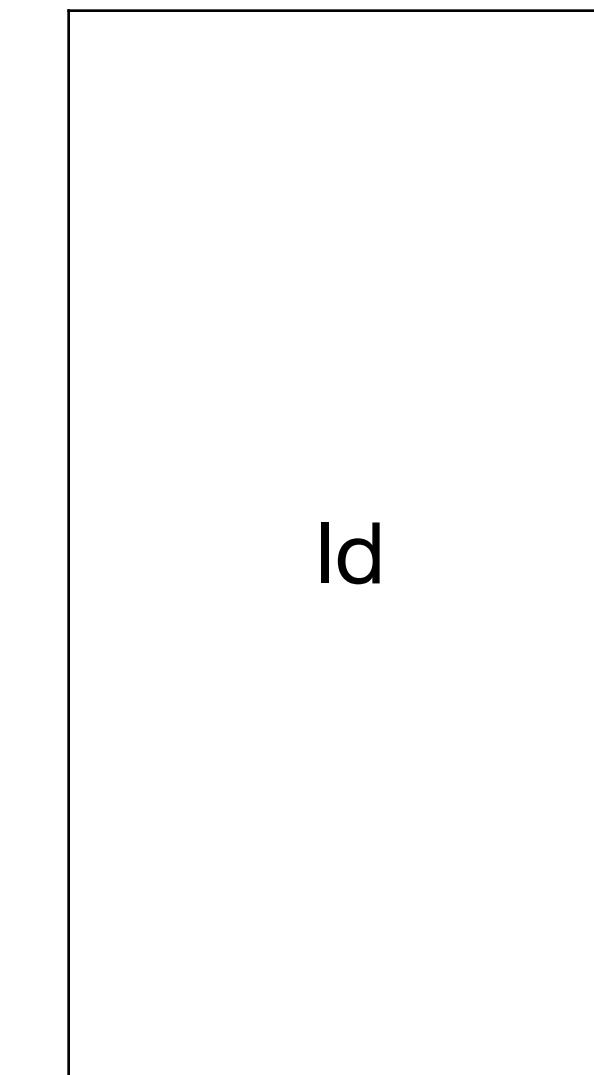
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

What about deletes?

- Unlink

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

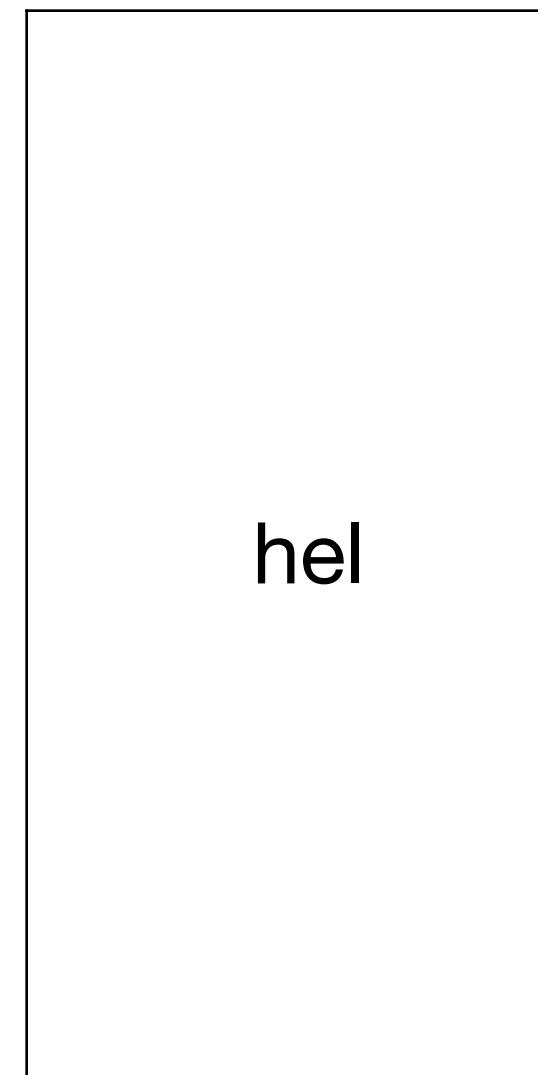
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

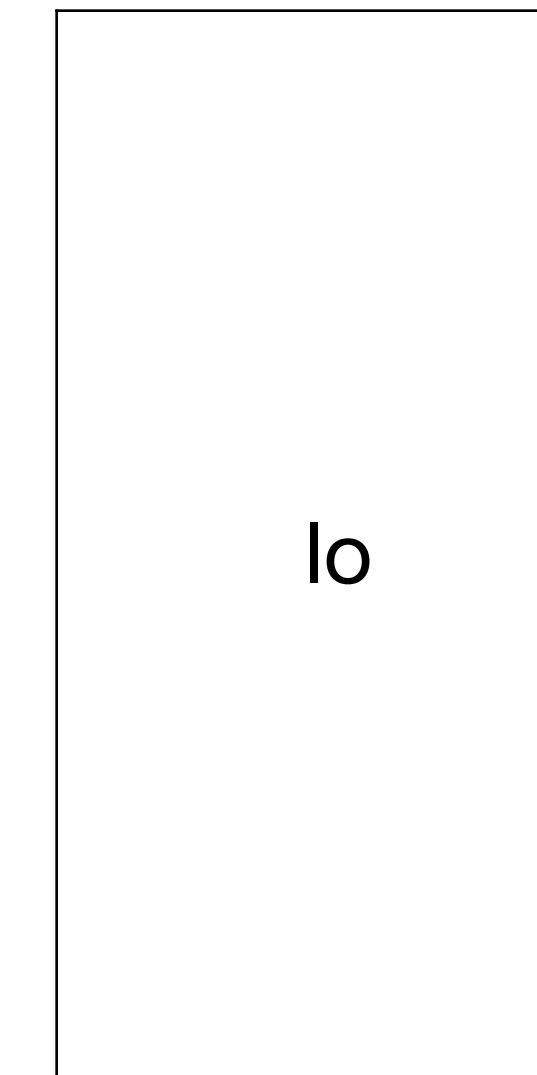
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

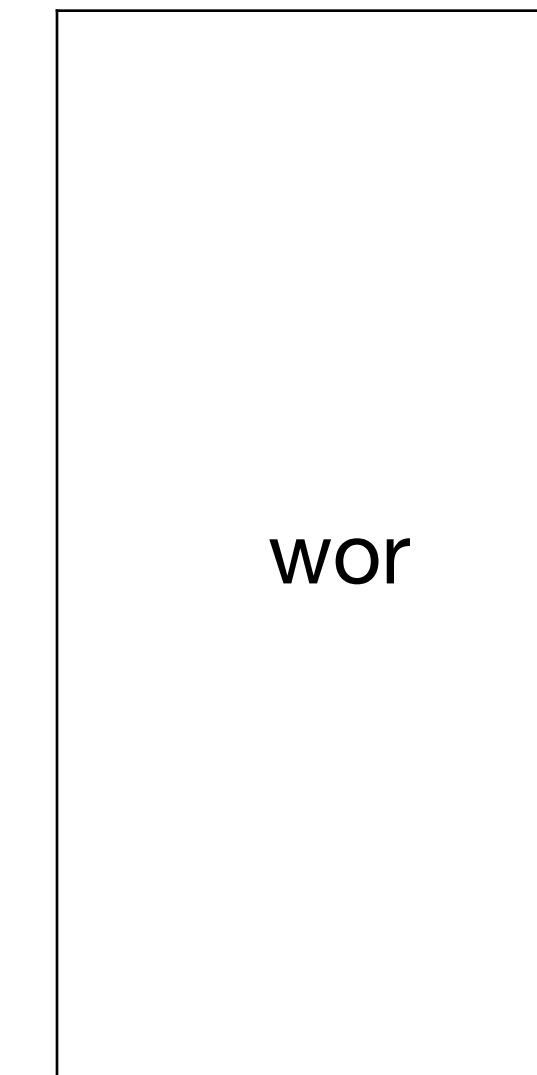
Block 3



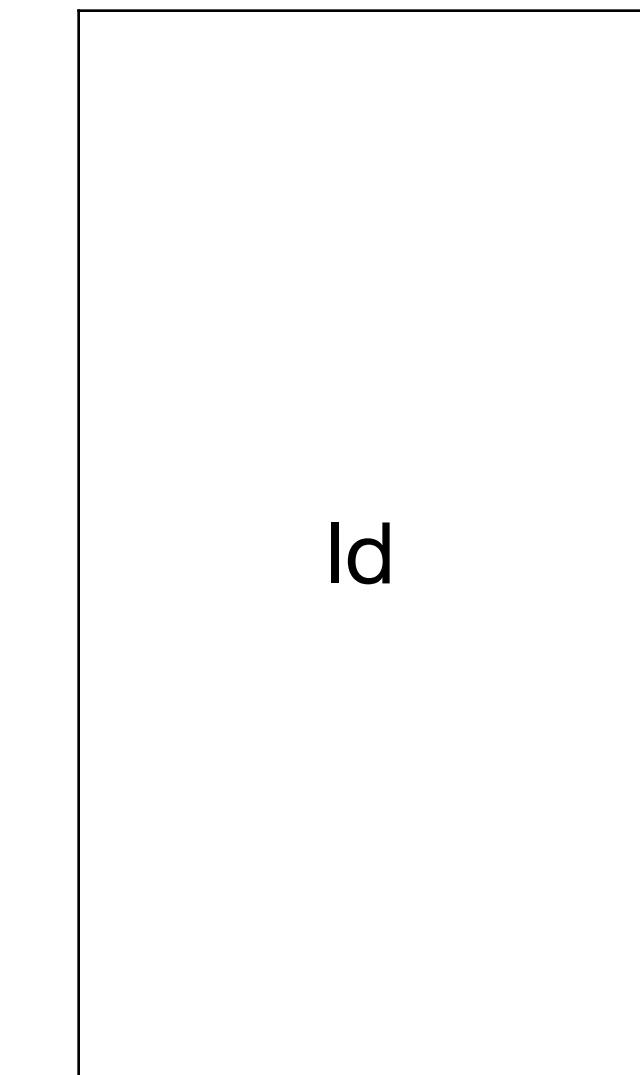
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

What about deletes?

- Unlink

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

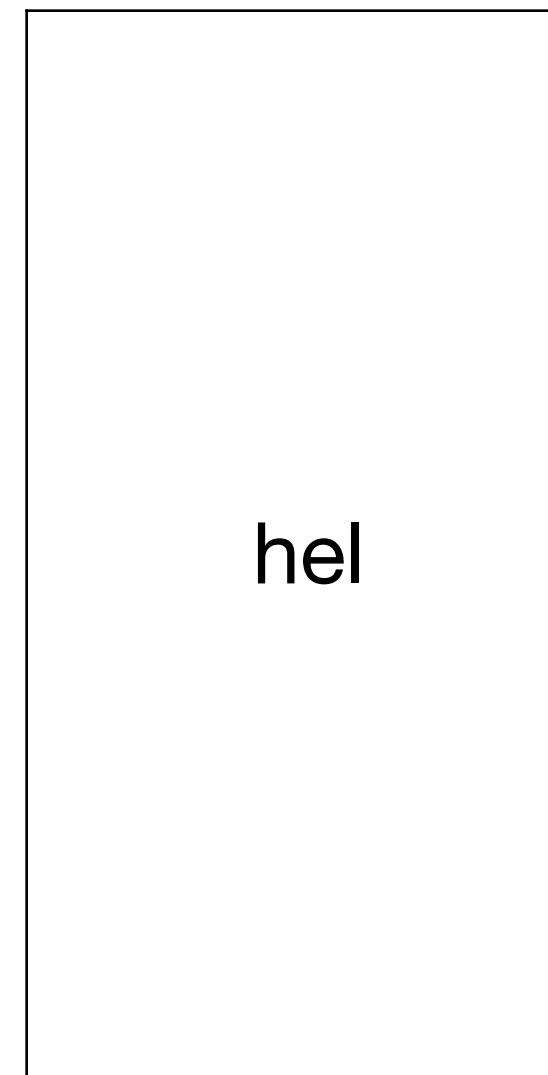
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

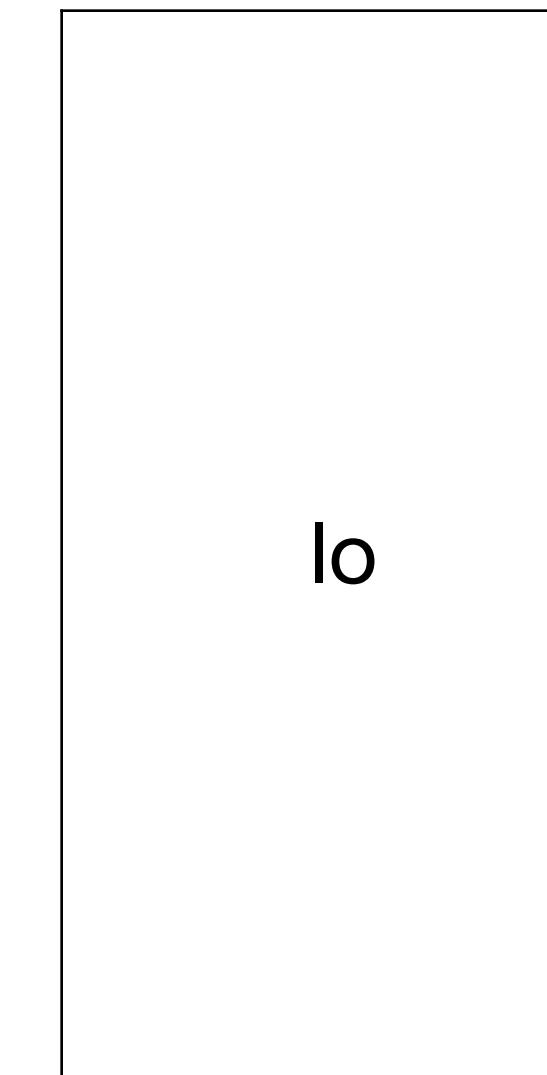
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Block 3



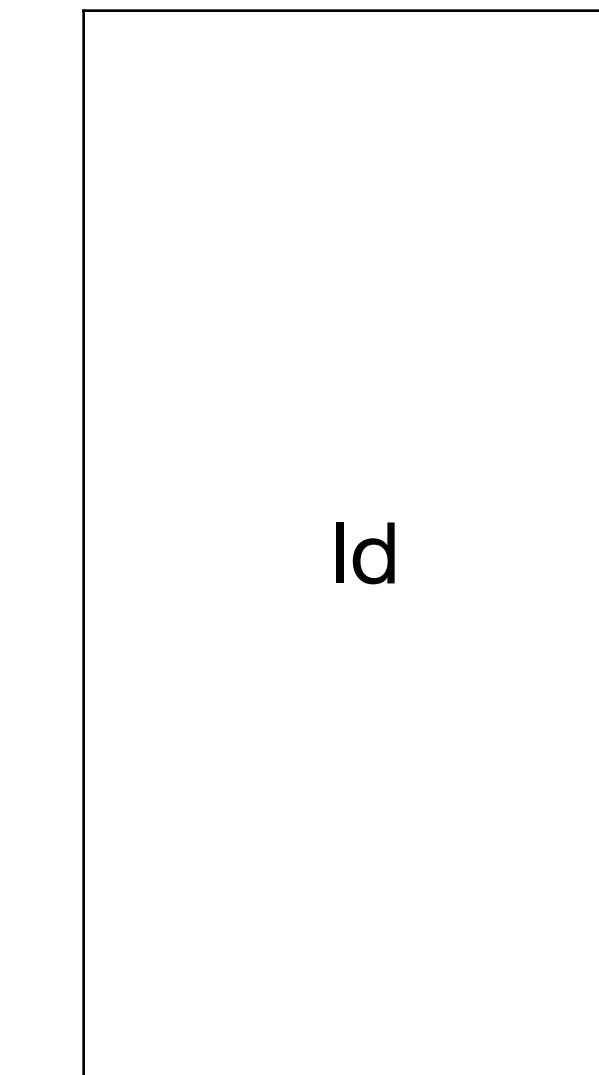
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

What about moves?

`mv a/foo b/`

“a” data block-1

File/directory name	Inode number
foo	2

“b” data block-1

File/directory name	Inode number
bar	36
foo	2

What about moves?

`mv a/foo b/`

“a” data block-1

File/directory name	Inode number
foo	2

“a” data block

File/directory name	Inode number

1

“b” data block-1

File/directory name	Inode number
bar	36
foo	2

“b” data block

File/directory name	Inode number
bar	36

2

What about moves?

`mv a/foo b/`

“a” data block-1

File/directory name	Inode number
foo	2

“a” data block



File/directory name	Inode number

“b” data block-1

File/directory name	Inode number
bar	36
foo	2

“b” data block



File/directory name	Inode number
bar	36

Might lose file ‘foo’

What about moves?

`mv a/foo b/`

“a” data block-1

File/directory name	Inode number
foo	2

“a” data block



File/directory name	Inode number

“a” data block-2



File/directory name	Inode number
foo	2

Might lose file ‘foo’

“b” data block-1

File/directory name	Inode number
bar	36
foo	2

“b” data block



File/directory name	Inode number
bar	36

“b” data block-2



File/directory name	Inode number
bar	36
foo	2

What about moves?

`mv a/foo b/`

“a” data block-1

File/directory name	Inode number
foo	2

“a” data block



File/directory name	Inode number

“a” data block-2



File/directory name	Inode number
foo	2

Might lose file ‘foo’

Have two pointers to foo

“b” data block-1

File/directory name	Inode number
bar	36
foo	2

“b” data block



File/directory name	Inode number
bar	36

“b” data block-2



File/directory name	Inode number
bar	36
foo	2

What about moves?

`mv a/foo b/`

“a” data block-1

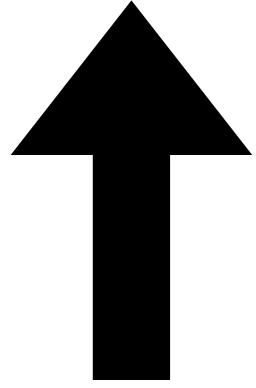
File/directory name	Inode number
foo	2

“a” data block

1	
File/directory name	Inode number

“a” data block-2

2	
File/directory name	Inode number
foo	2



“b” data block-1

File/directory name	Inode number
bar	36
foo	2

Might lose file ‘foo’

“b” data block

2	
File/directory name	Inode number
bar	36

Have two pointers to foo

“b” data block-2

1	
File/directory name	Inode number
bar	36
foo	2

Operations in a transaction can create circular dependencies

mv a/foo b/; mv b/bar a/;

“a” data block

File/directory name	Inode number
foo	2

“b” data block

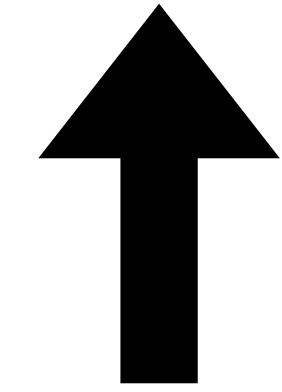
File/directory name	Inode number
bar	36

Operations in a transaction can create circular dependencies

`mv a/foo b/; mv b/bar a/;`

“a” data block

File/directory name	Inode number
foo	2



“b” data block

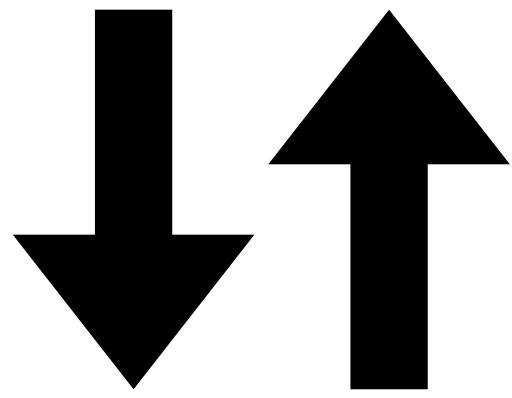
File/directory name	Inode number
bar	36

Operations in a transaction can create circular dependencies

`mv a/foo b/; mv b/bar a/;`

“a” data block

File/directory name	Inode number
foo	2



“b” data block

File/directory name	Inode number
bar	36

Operations in a transaction can create circular dependencies

`mv a/foo b/; mv b/bar a/;`

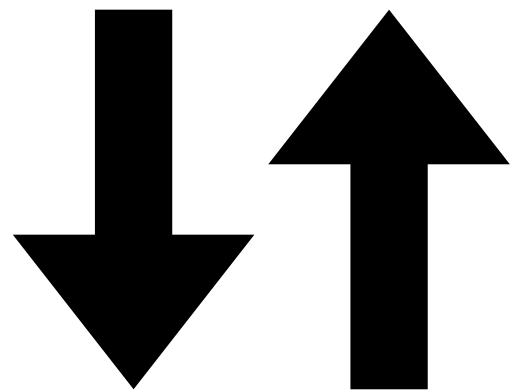
- Detect cycles before happening.
- Close the transaction. Flush to disk.

“a” data block

File/directory name	Inode number
foo	2

“b” data block

File/directory name	Inode number
bar	36



Fixing space leaks

fsck scans entire disk

Operations	Order	Problem	Fix

Fixing space leaks

fsck scans entire disk

Operations	Order	Problem	Fix
Writing to file	Data bitmaps, data blocks, inode block	Leak data blocks	Mark data blocks free in bitmap

Fixing space leaks

fsck scans entire disk

Operations	Order	Problem	Fix
Writing to file	Data bitmaps, data blocks, inode block	Leak data blocks	Mark data blocks free in bitmap
Creating a file	Inode bitmap, file's inode, parent's inode	Leak inodes	Move to lost and found

Fixing space leaks

fsck scans entire disk

Operations	Order	Problem	Fix
Writing to file	Data bitmaps, data blocks, inode block	Leak data blocks	Mark data blocks free in bitmap
Creating a file	Inode bitmap, file's inode, parent's inode	Leak inodes	Move to lost and found
Truncate	inode, data bitmap	Leak data blocks	Mark data blocks free in bitmap

Fixing space leaks

fsck scans entire disk

Operations	Order	Problem	Fix
Writing to file	Data bitmaps, data blocks, inode block	Leak data blocks	Mark data blocks free in bitmap
Creating a file	Inode bitmap, file's inode, parent's inode	Leak inodes	Move to lost and found
Truncate	inode, data bitmap	Leak data blocks	Mark data blocks free in bitmap
Unlink	Parent's inode, inode bitmap	Leak inodes	Move to lost and found

Fixing space leaks

fsck scans entire disk

Operations	Order	Problem	Fix
Writing to file	Data bitmaps, data blocks, inode block	Leak data blocks	Mark data blocks free in bitmap
Creating a file	Inode bitmap, file's inode, parent's inode	Leak inodes	Move to lost and found
Truncate	inode, data bitmap	Leak data blocks	Mark data blocks free in bitmap
Unlink	Parent's inode, inode bitmap	Leak inodes	Move to lost and found
mv a/foo b/	b's inode, a's inode	Multiple links to foo	Set foo's nlinks=2

Problems

- Need to carefully decide ordering

Problems

- Need to carefully decide ordering
- Proactively detect cycles. Cycles in ordering force a commit

Problems

- Need to carefully decide ordering
- Proactively detect cycles. Cycles in ordering force a commit
- Ordering reduces write throughput (reduced disk scheduling opportunities)

Problems

- Need to carefully decide ordering
- Proactively detect cycles. Cycles in ordering force a commit
- Ordering reduces write throughput (reduced disk scheduling opportunities)
- File system code needs to co-evolve with fsck. Wrong assumptions, bugs in fsck can destroy the file system

Problems

- Need to carefully decide ordering
- Proactively detect cycles. Cycles in ordering force a commit
- Ordering reduces write throughput (reduced disk scheduling opportunities)
- File system code needs to co-evolve with fsck. Wrong assumptions, bugs in fsck can destroy the file system
- Fsck scans the entire disk. 70GB disk with 2 million inodes takes 10 minutes. Impractical for large disks.

Crash consistency

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

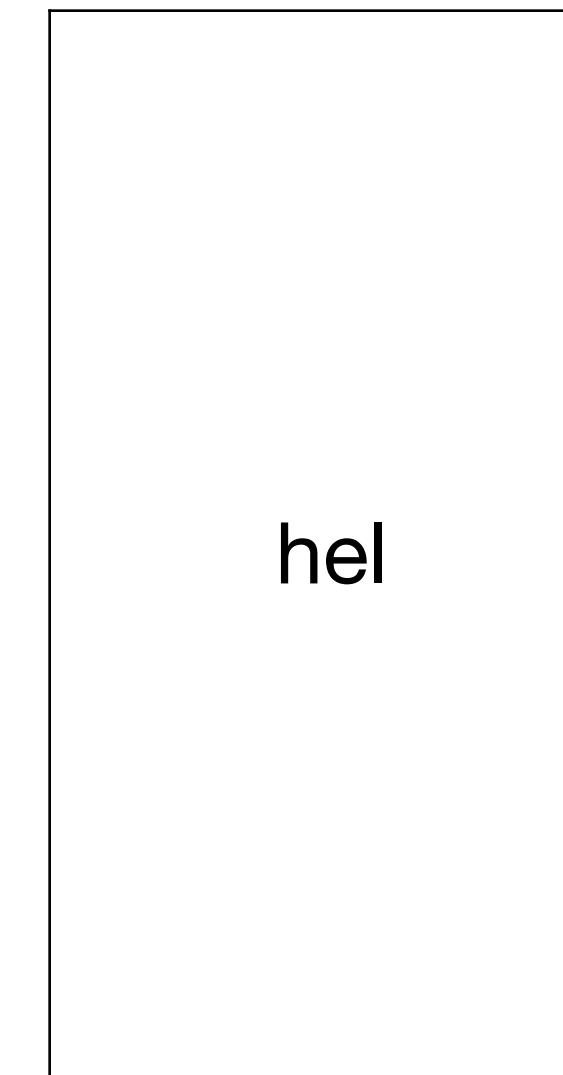
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

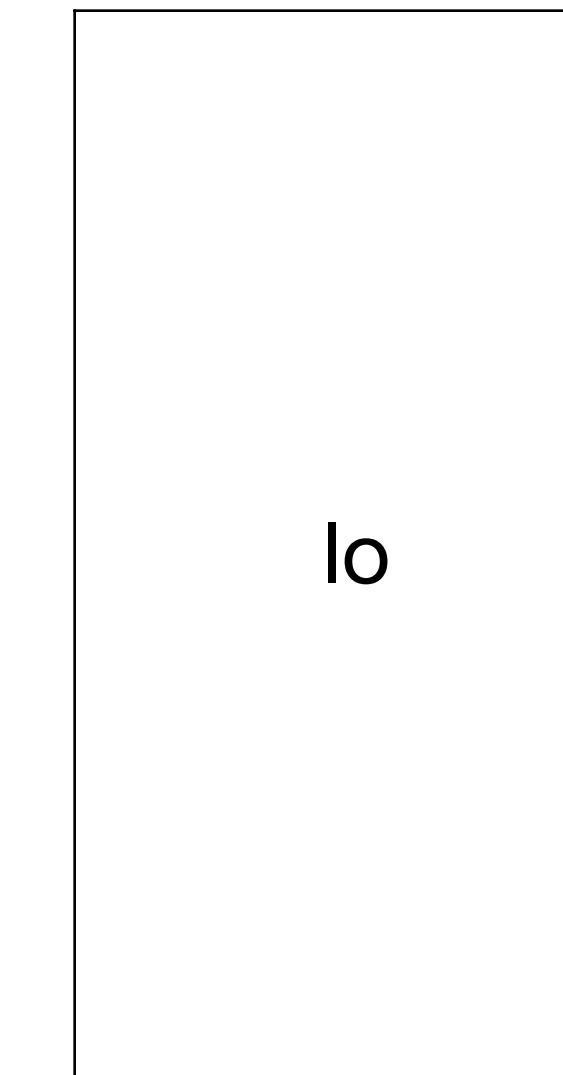
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Block 3



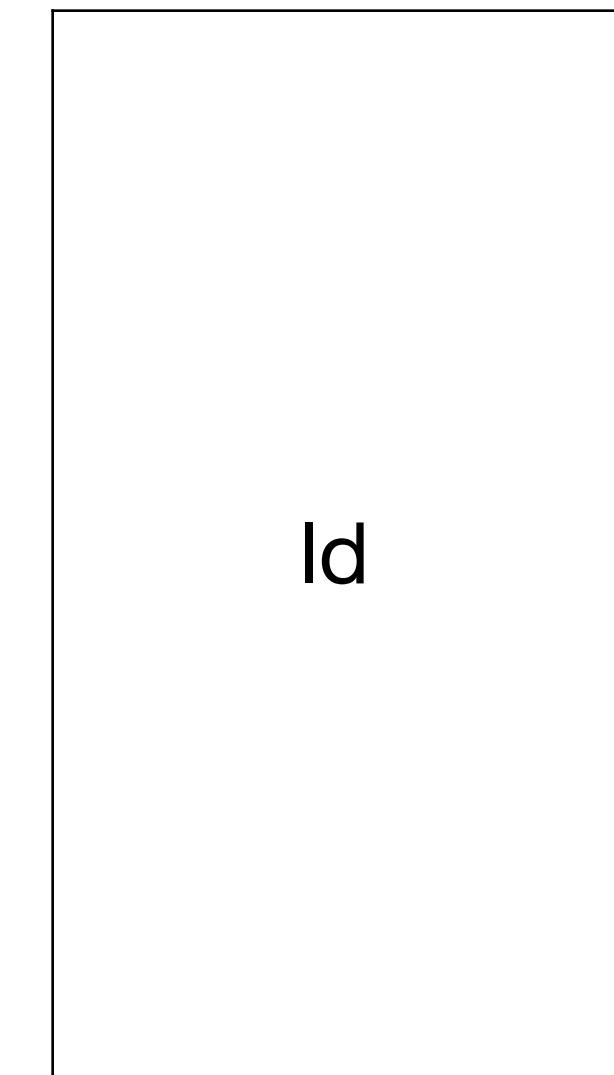
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Crash consistency

- Durability guarantee: at some point in the future (5 to 30 seconds)
 - Most performant. Buffer cache is write-back.
 - One transaction contains several operations

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE);
write(fd, "hello world\n", 12);
close(fd);
```

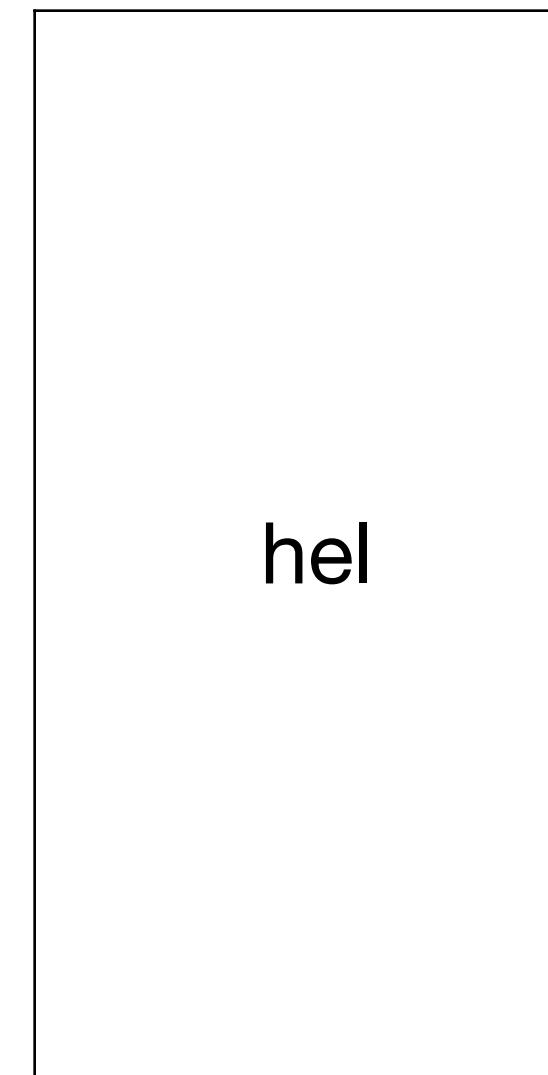
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

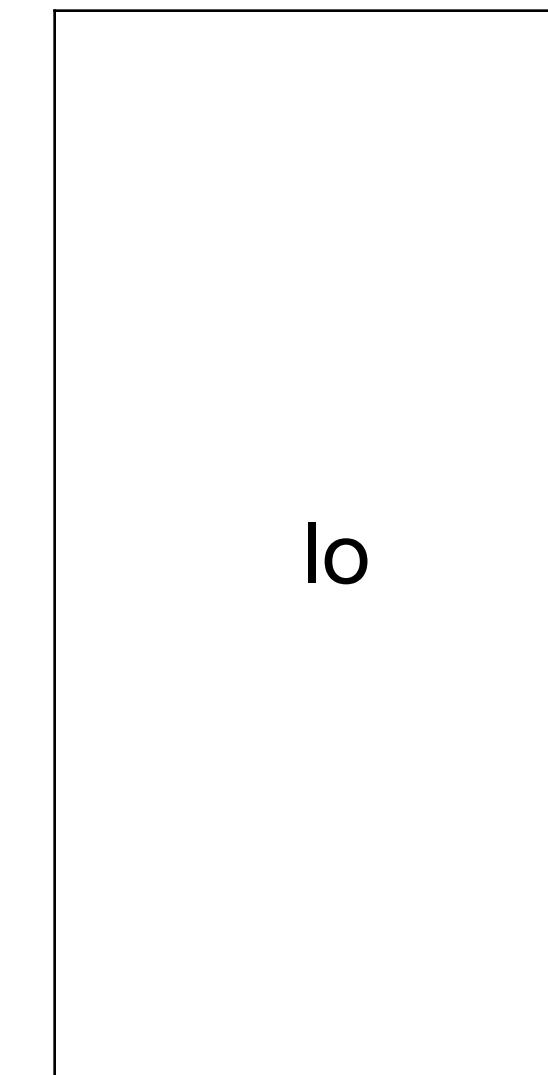
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Block 3



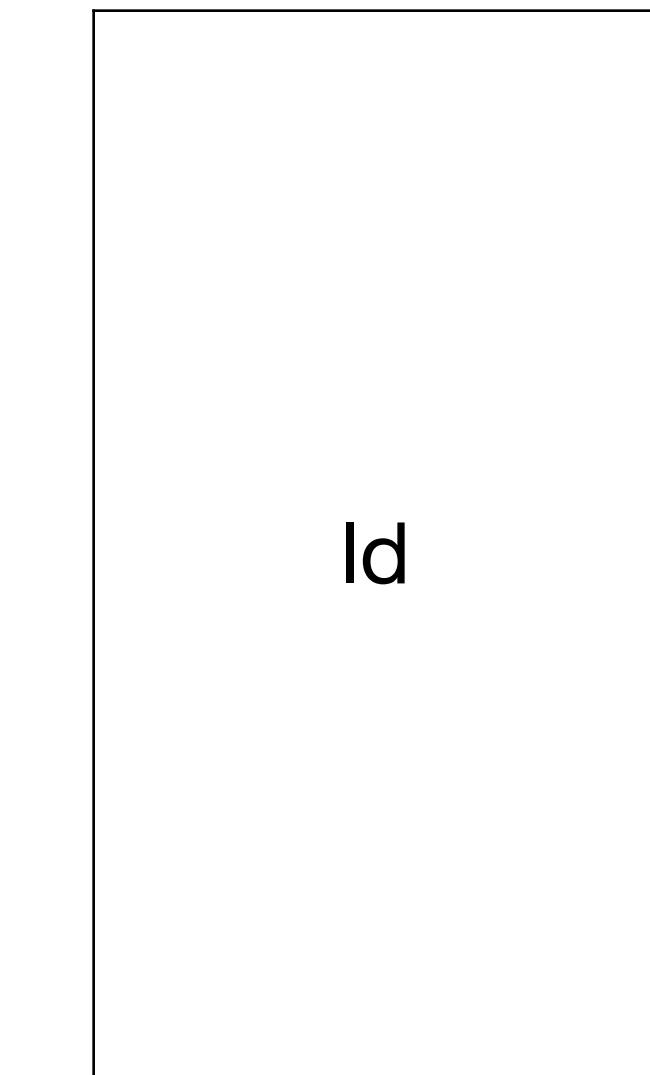
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Crash consistency

- Durability guarantee: at some point in the future (5 to 30 seconds)
 - Most performant. Buffer cache is write-back.
 - One transaction contains several operations
- Make writes atomic with respect to power failures / kernel crashes: either all the blocks are written or none of the blocks are written

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE);
write(fd, "hello world\n", 12);
close(fd);
```

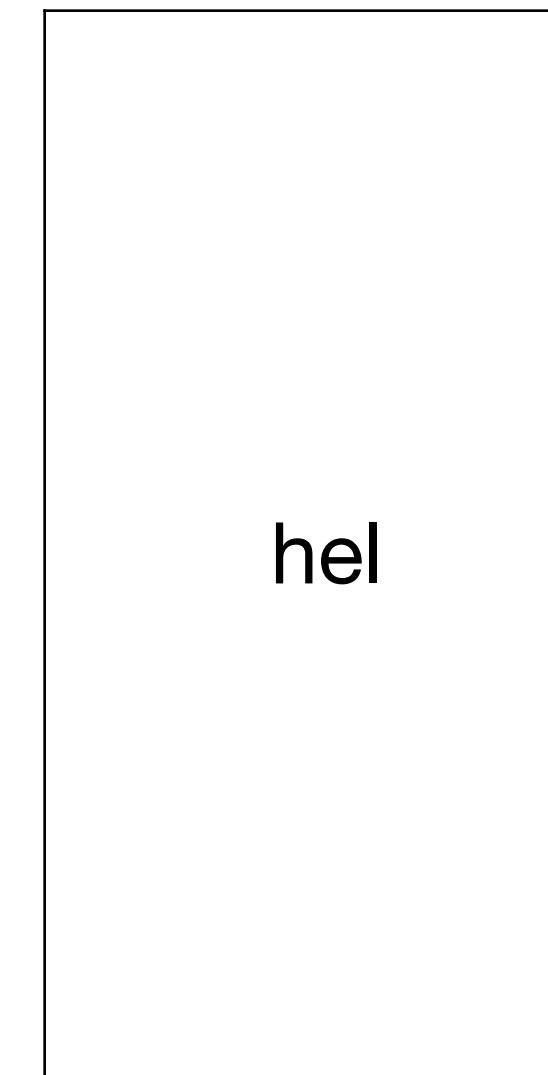
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

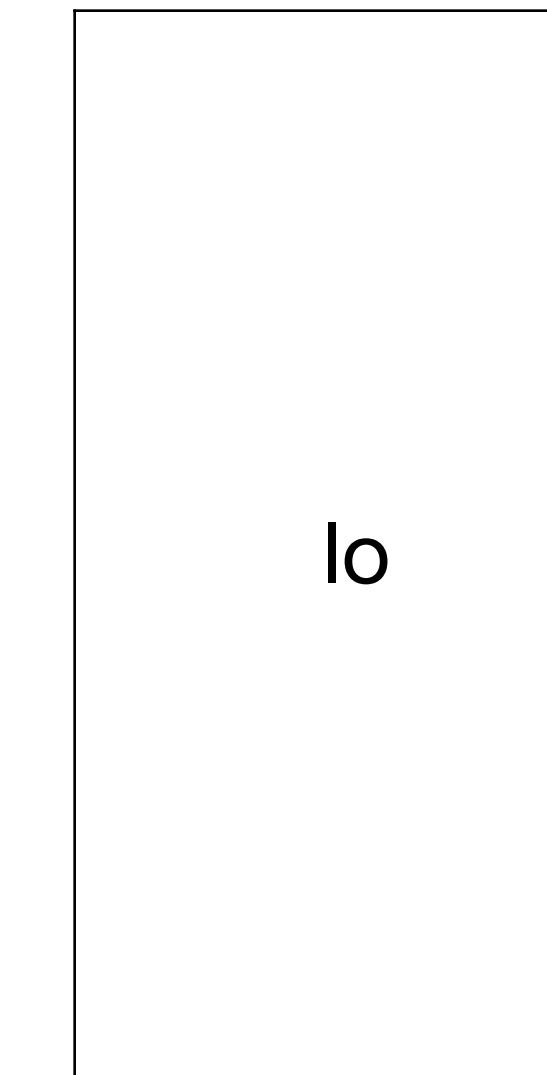
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

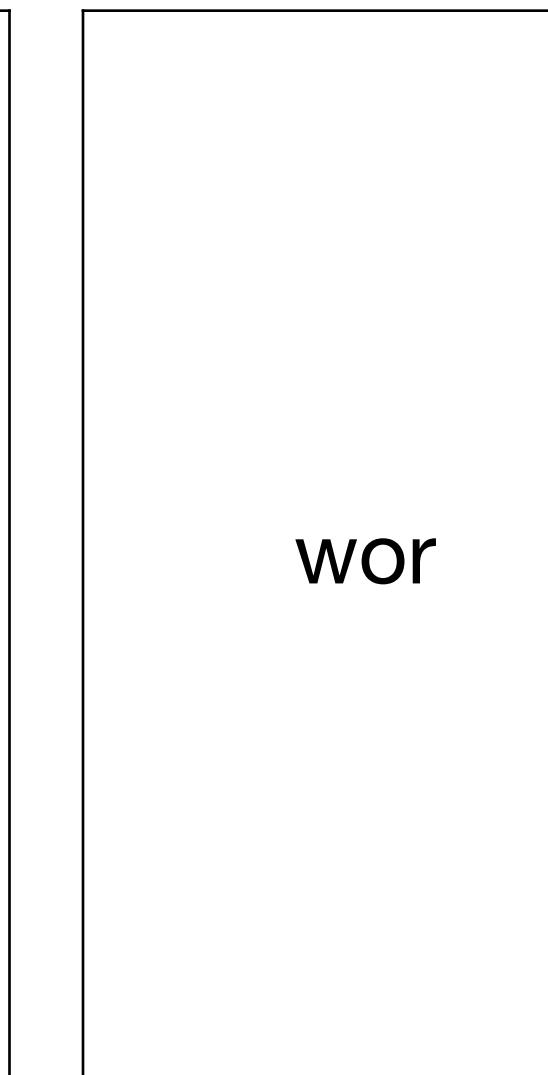
Block 3



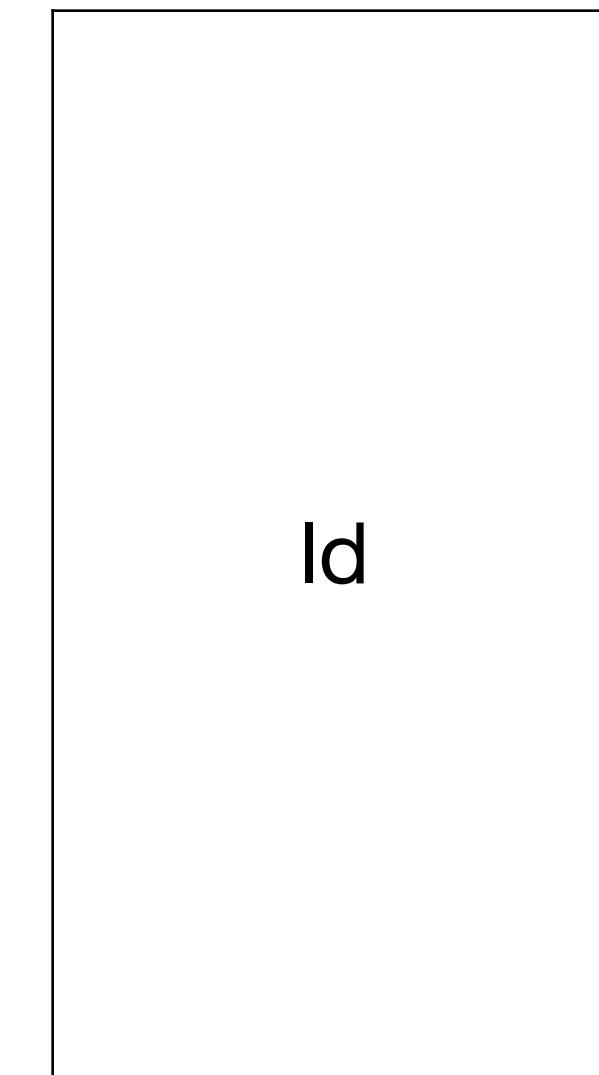
Block 5



Block 7



Block 12



Crash consistency

- Durability guarantee: at some point in the future (5 to 30 seconds)
 - Most performant. Buffer cache is write-back.
 - One transaction contains several operations
- Make writes atomic with respect to power failures / kernel crashes: either all the blocks are written or none of the blocks are written
- Challenge: Disk only writes one block at a time

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)
write(fd, "hello world\n", 12);
close(fd);
```

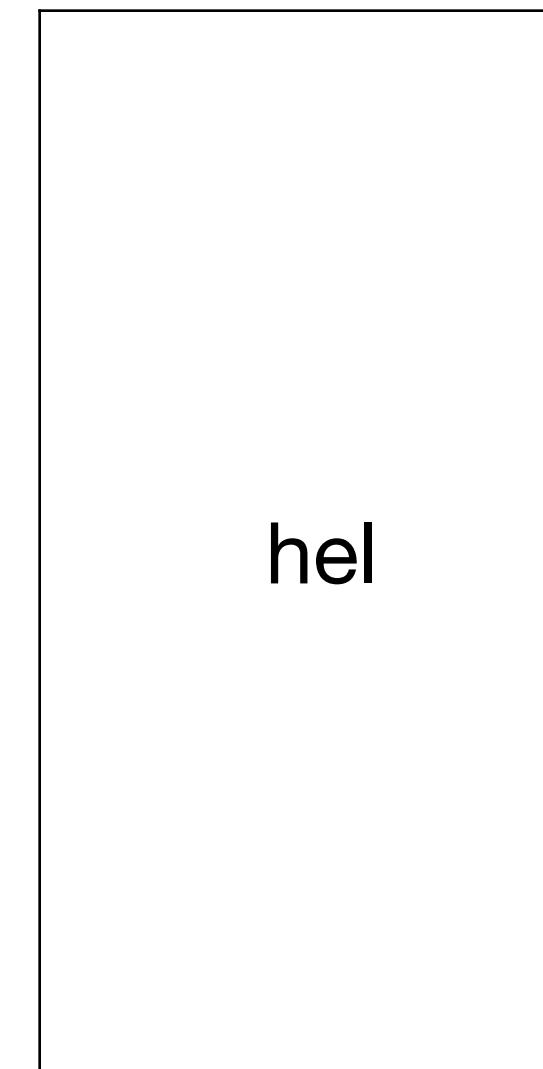
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

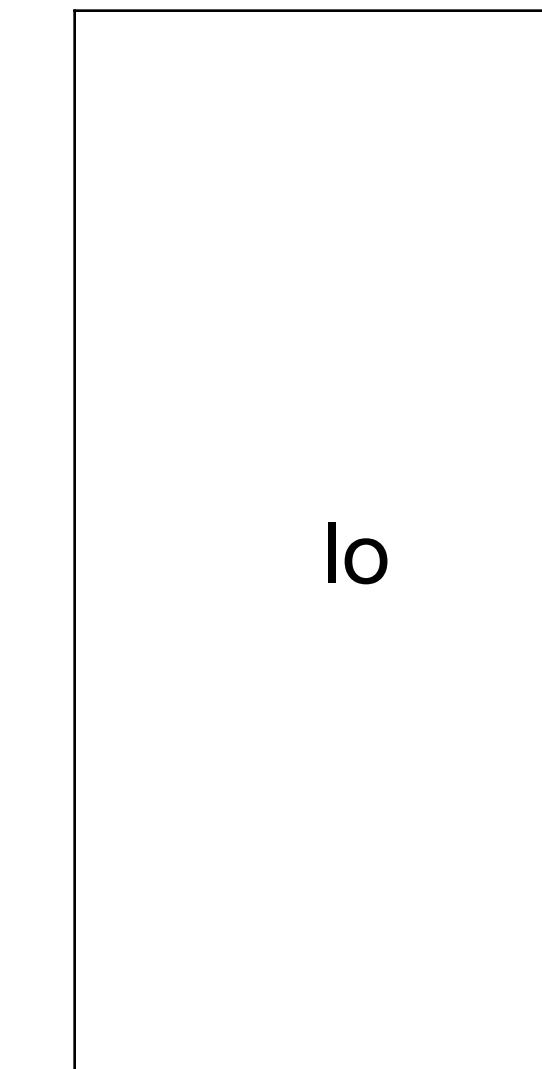
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

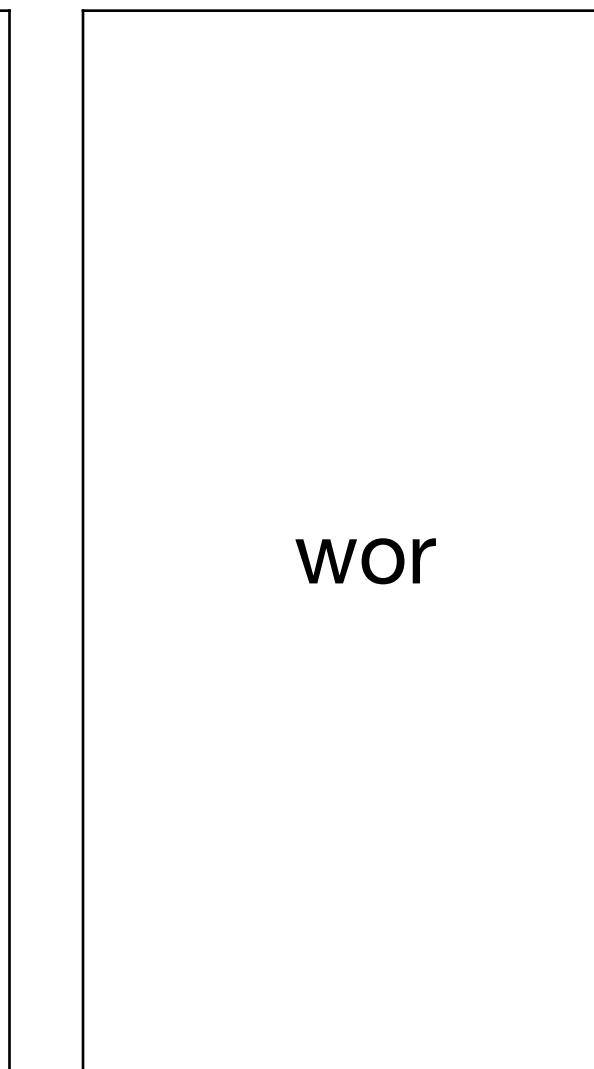
Block 3



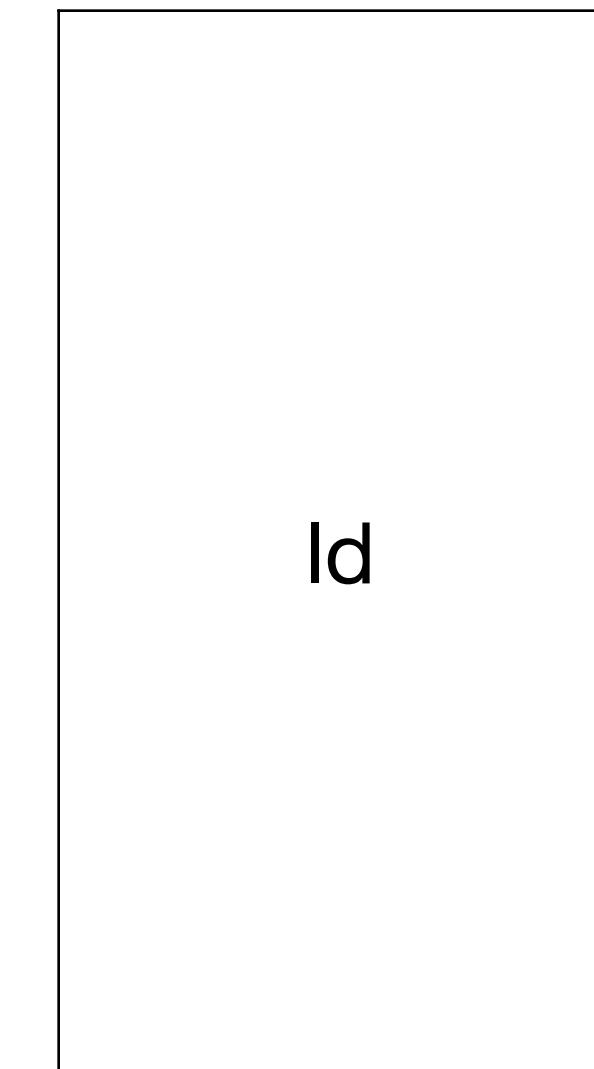
Block 5



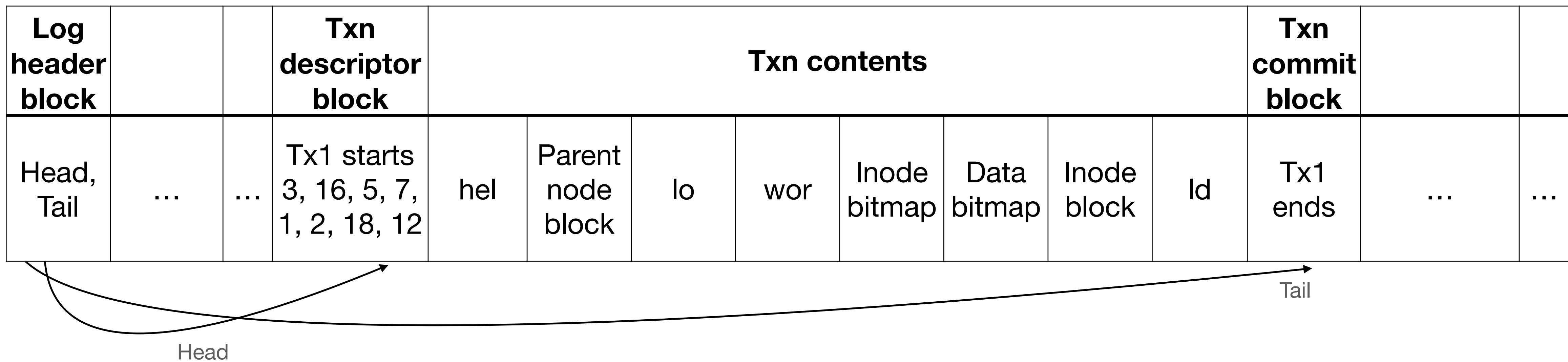
Block 7



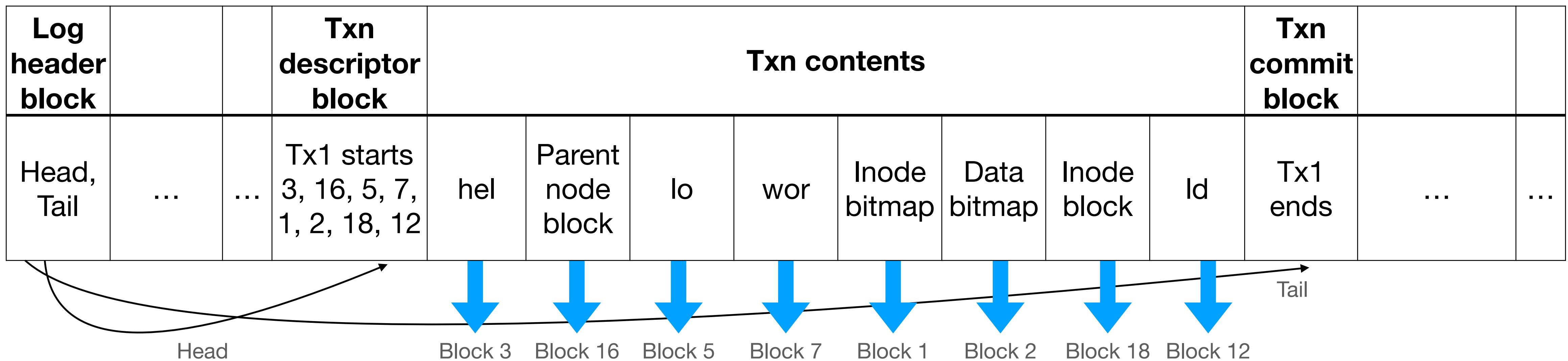
Block 12



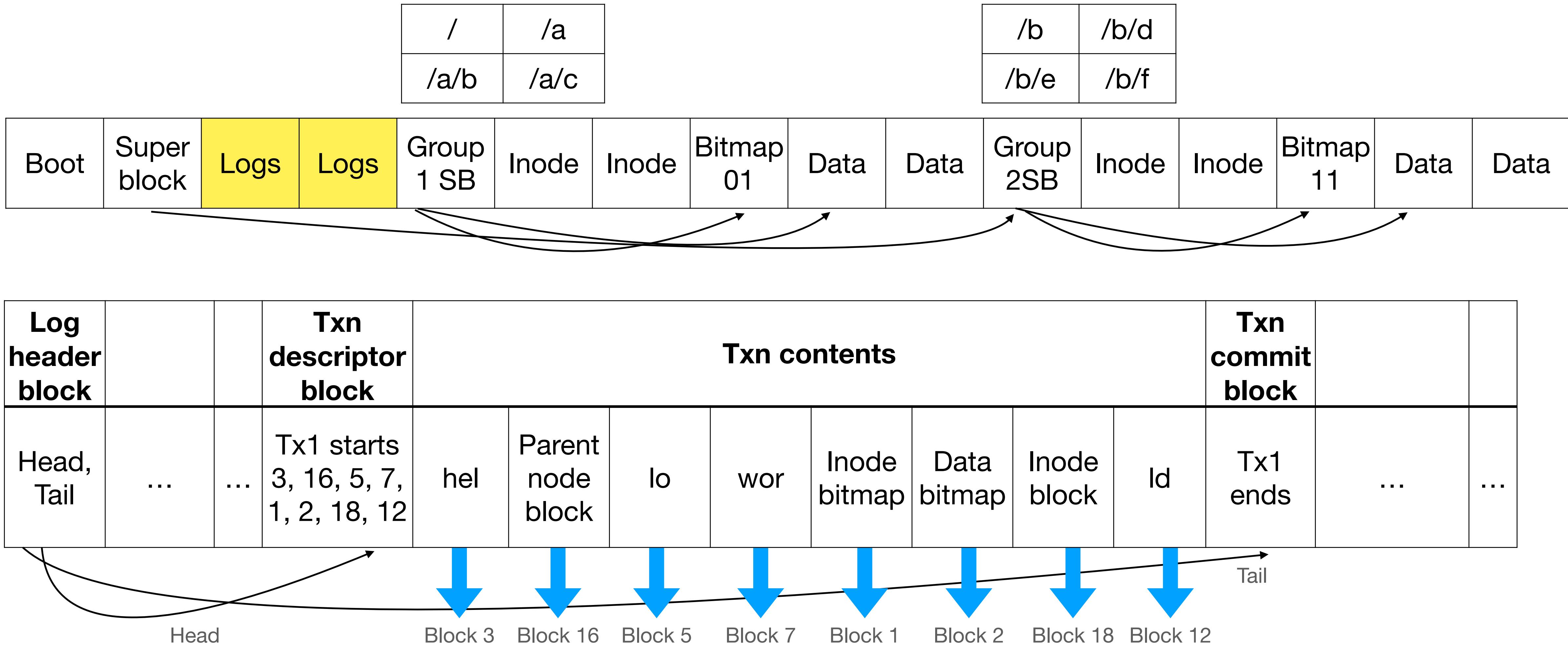
Write-ahead log



Write-ahead log

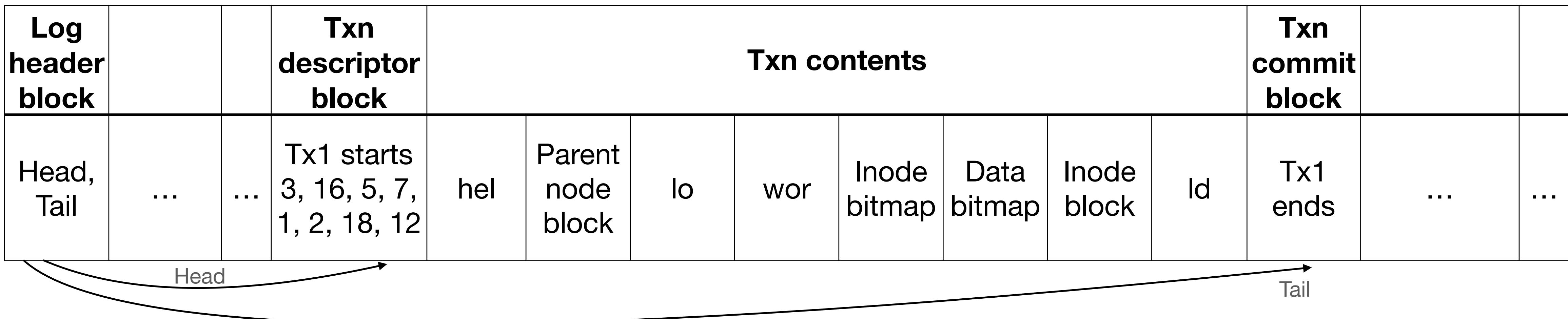


Write-ahead log



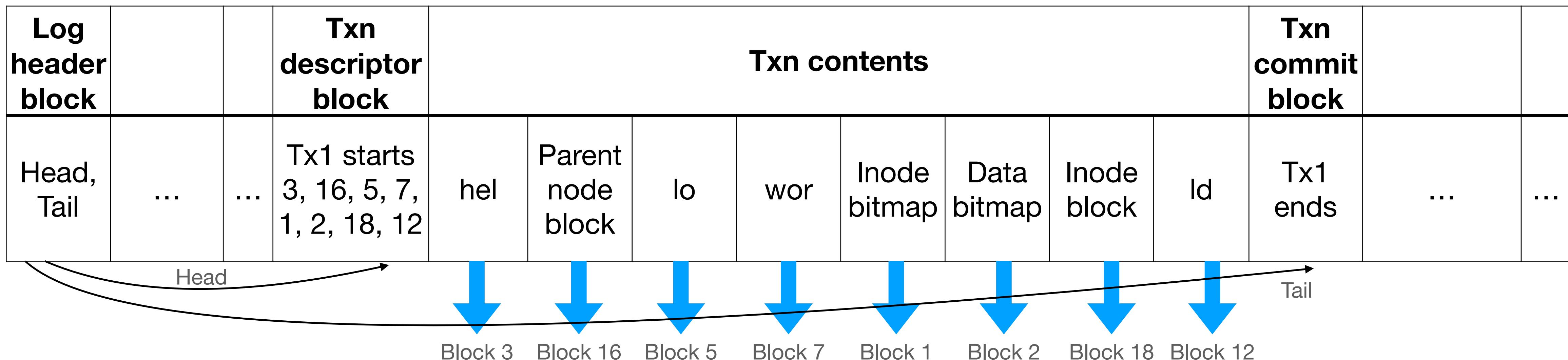
Recovery procedure

- At reboot:
 - read from head to tail
 - “Redo” transaction contents to home locations
 - Update head of log header block
 - Atomic: All blocks are written for committed transactions. None for uncommitted transactions.



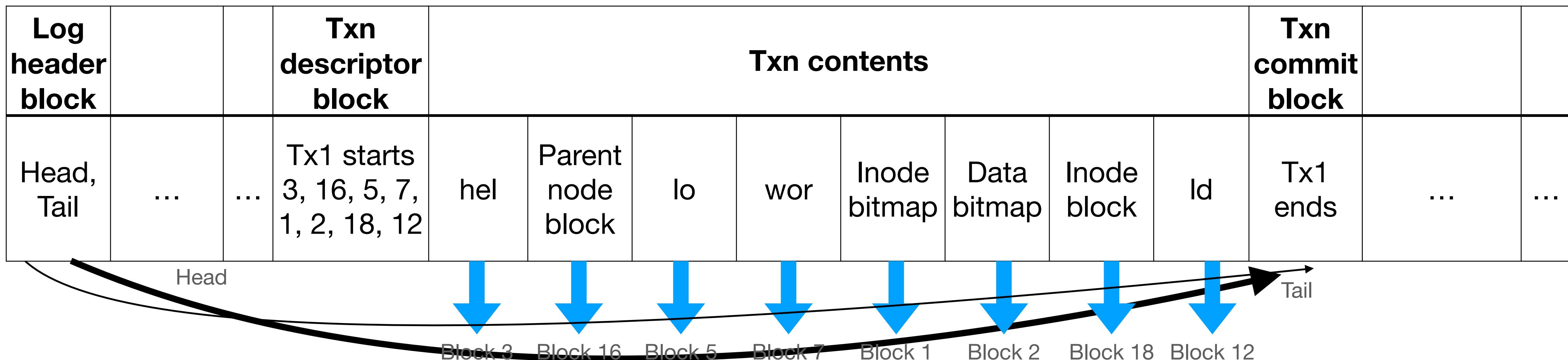
Recovery procedure

- At reboot:
 - read from head to tail
 - “Redo” transaction contents to home locations
 - Update head of log header block
 - Atomic: All blocks are written for committed transactions. None for uncommitted transactions.



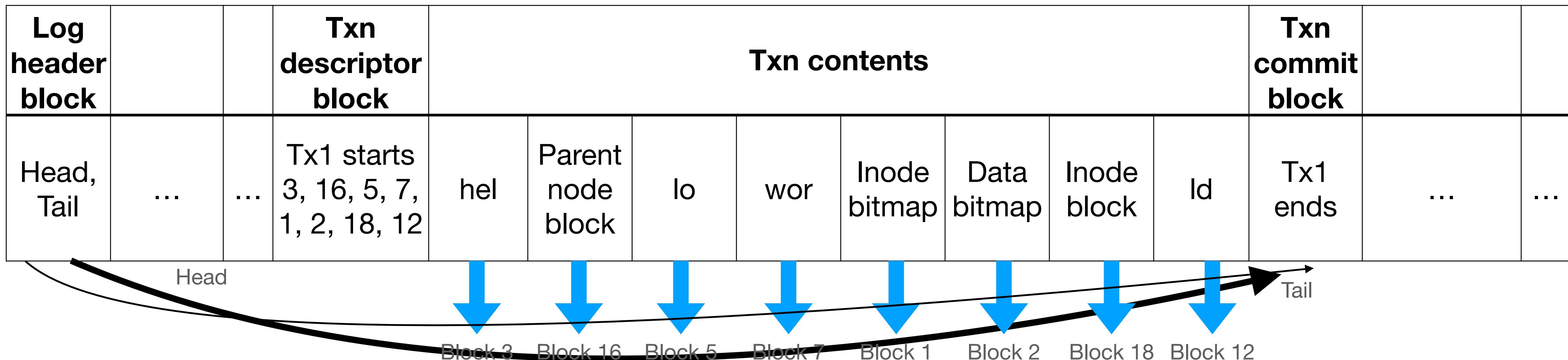
Recovery procedure

- At reboot:
 - read from head to tail
 - “Redo” transaction contents to home locations
 - Update head of log header block
 - Atomic: All blocks are written for committed transactions. None for uncommitted transactions.



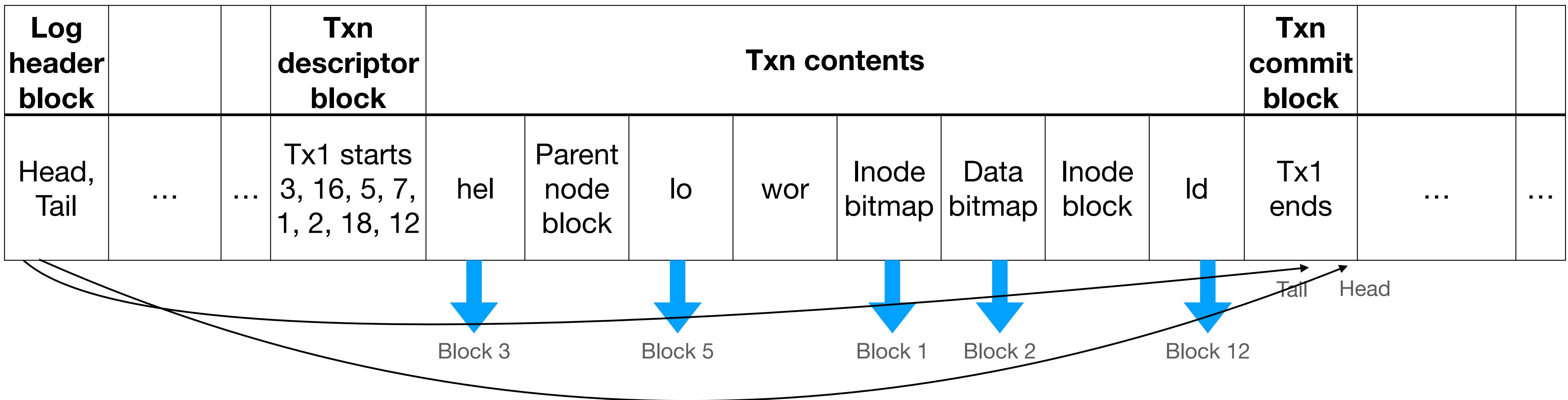
Recovery procedure

- At reboot:
 - read from head to tail
 - “Redo” transaction contents to home locations
 - Update head of log header block
 - Atomic: All blocks are written for committed transactions. None for uncommitted transactions.

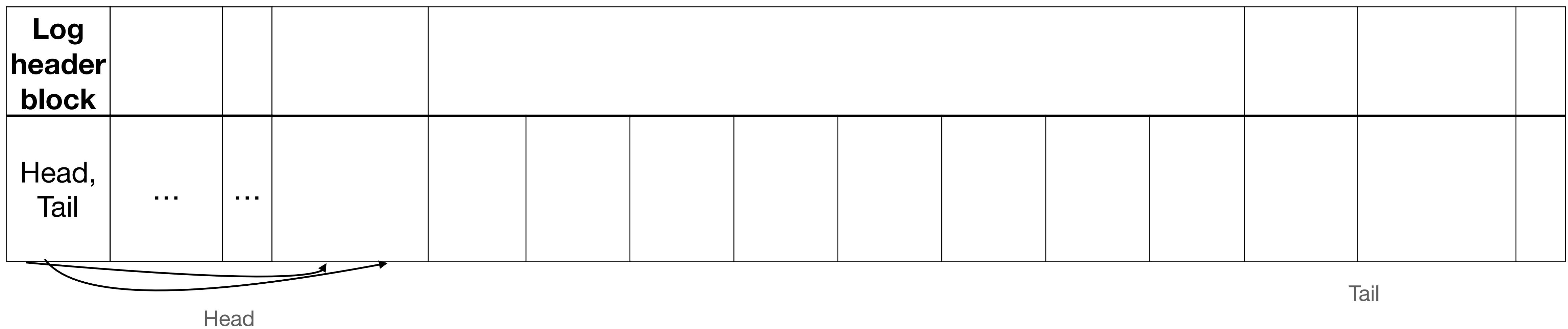


Why do we need ordering?

- At reboot:
 - read from head to tail
 - “Redo” transaction contents to home locations
 - Update head of log header block
 - Atomic: All blocks are written for committed transactions. None for uncommitted transactions.

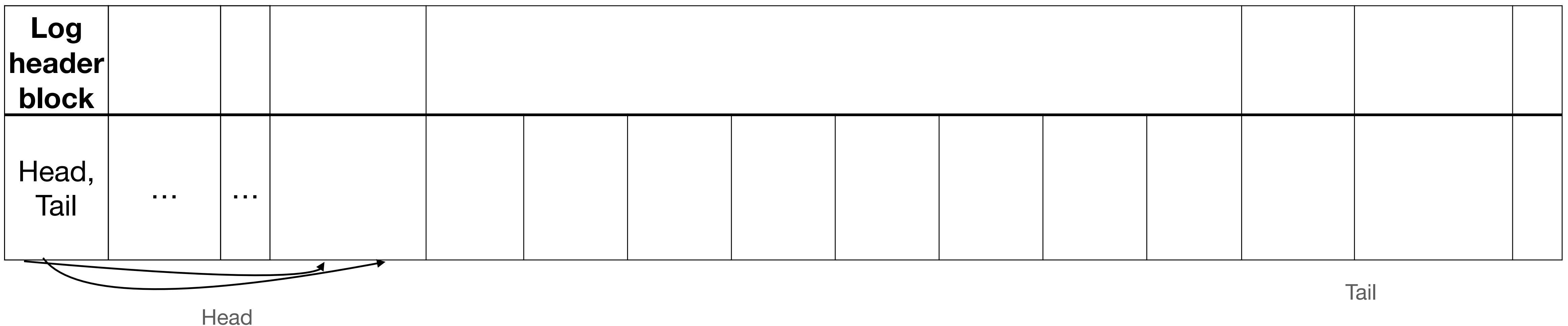


Transaction end



Transaction end

- Update tail in log header block



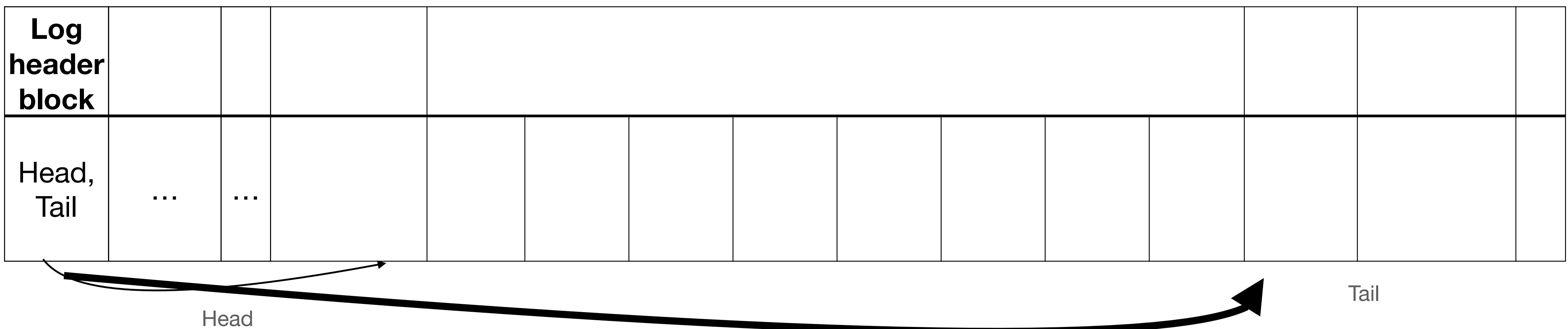
Transaction end

- Update tail in log header block

The diagram illustrates a log header block and a circular buffer. The log header block is a vertical column of four rows. The top row is labeled "Log header block". The second row contains the text "Head, Tail" with three ellipsis ("...") symbols positioned between the first and second columns. The third and fourth rows are empty. Below the log header block is a horizontal line. A large, thick black arrow originates from the bottom-left, points right along the line, then curves back up to point to the "Tail" label in the fourth row of the log header block. The "Head" label is located at the bottom-left, below the arrow's starting point.

Transaction end

- Update tail in log header block
 - Write transaction descriptor block



Transaction end

- Update tail in log header block
 - Write transaction descriptor block

Head

Tail

Transaction end

- Update tail in log header block
 - Write transaction descriptor block
 - Write transaction contents

Head

Tail

Transaction end

- Update tail in log header block
 - Write transaction descriptor block
 - Write transaction contents

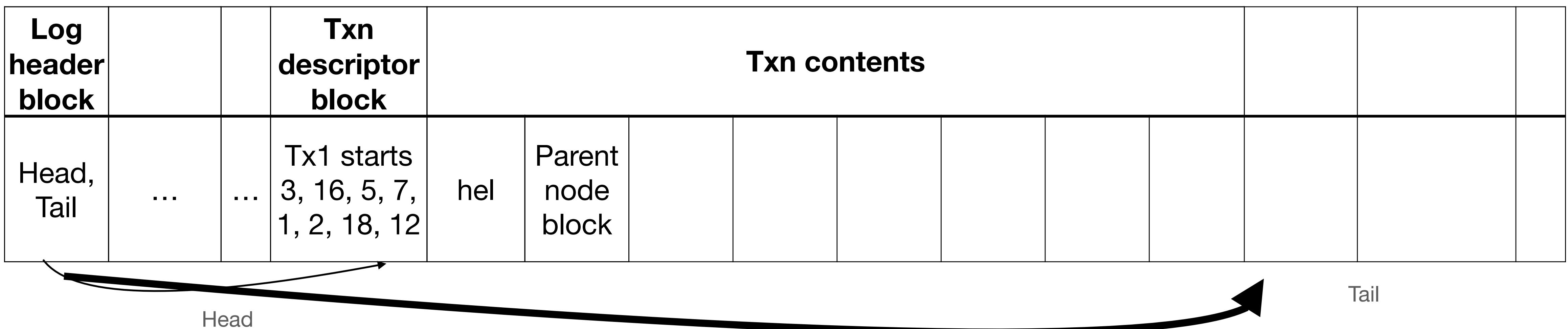
Log			Txn							
header			descriptor							
block			block							
Head,		Tx1 starts						
Tail		3, 16, 5, 7,	hel					
,		...	1, 2, 18, 12							

Head

Tail

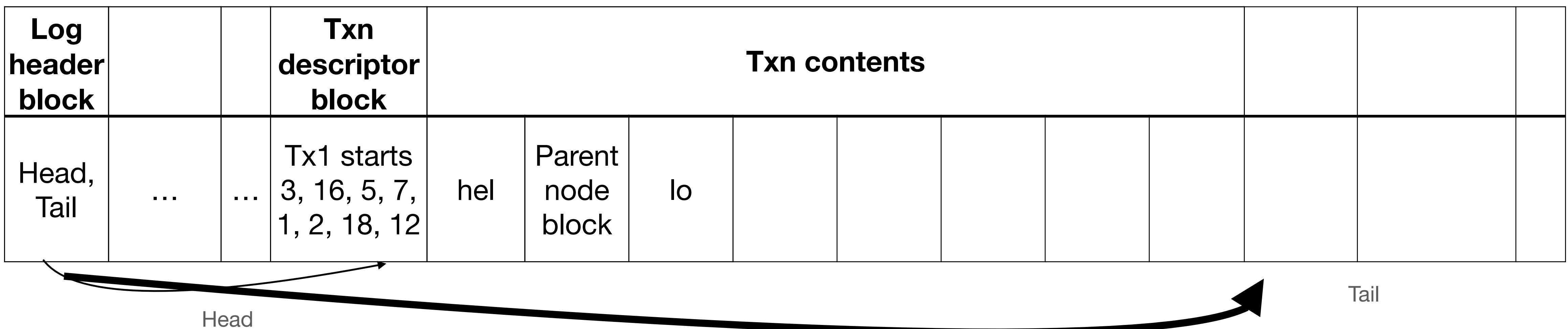
Transaction end

- Update tail in log header block
 - Write transaction descriptor block
 - Write transaction contents



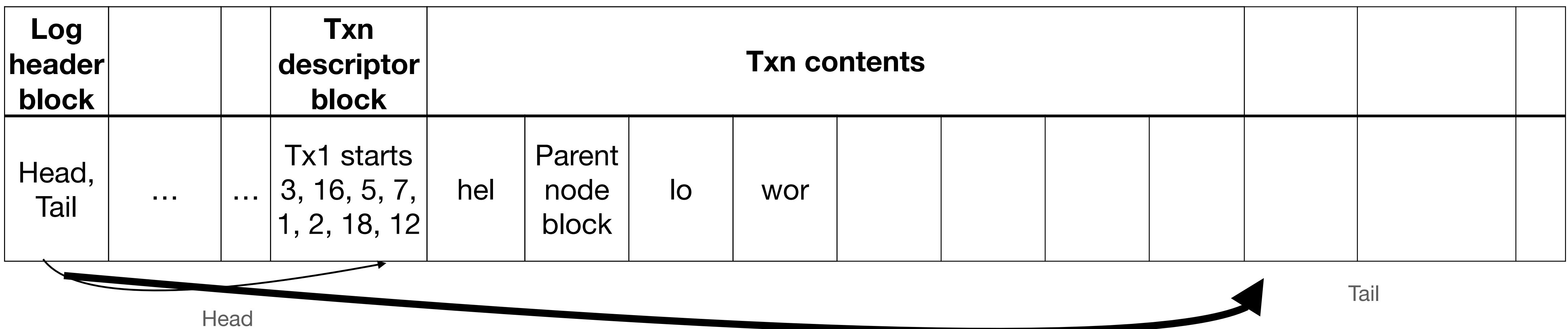
Transaction end

- Update tail in log header block
 - Write transaction descriptor block
 - Write transaction contents



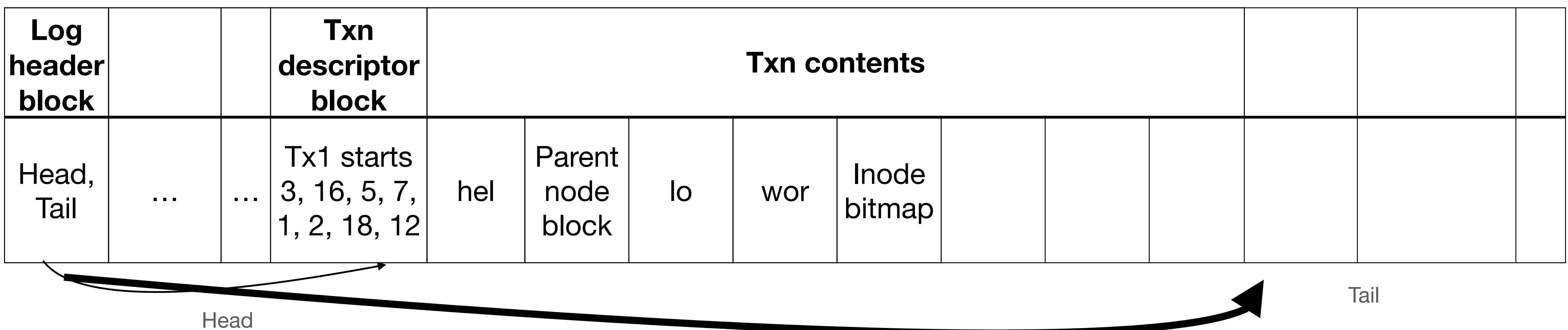
Transaction end

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents



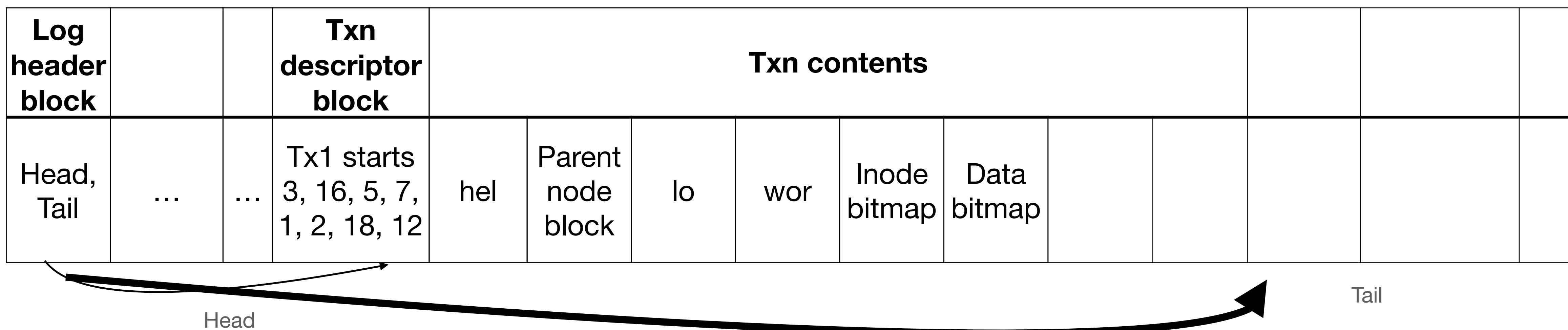
Transaction end

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents



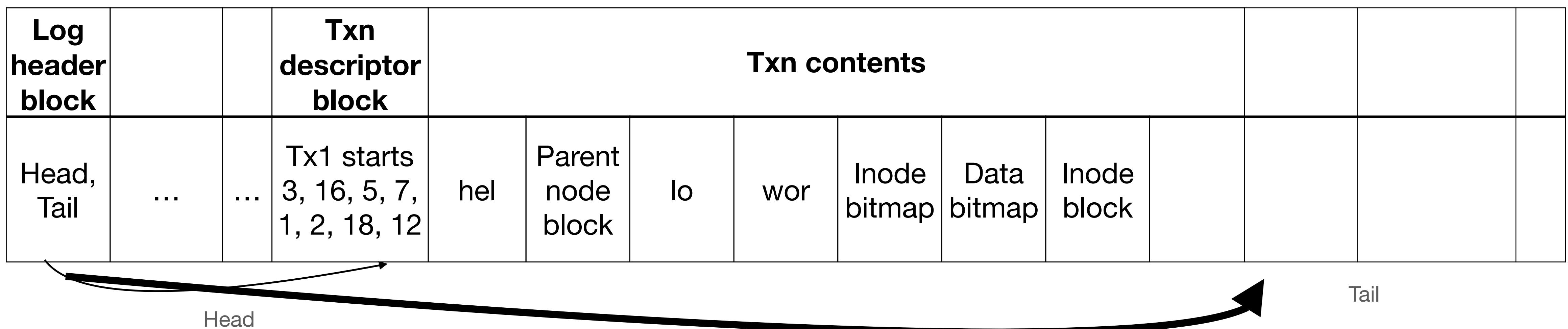
Transaction end

- Update tail in log header block
 - Write transaction descriptor block
 - Write transaction contents



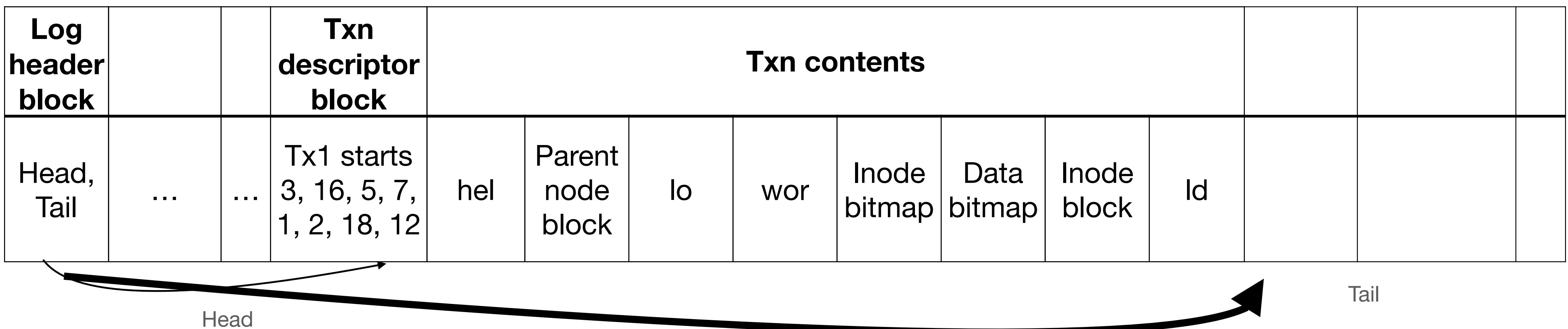
Transaction end

- Update tail in log header block
 - Write transaction descriptor block
 - Write transaction contents



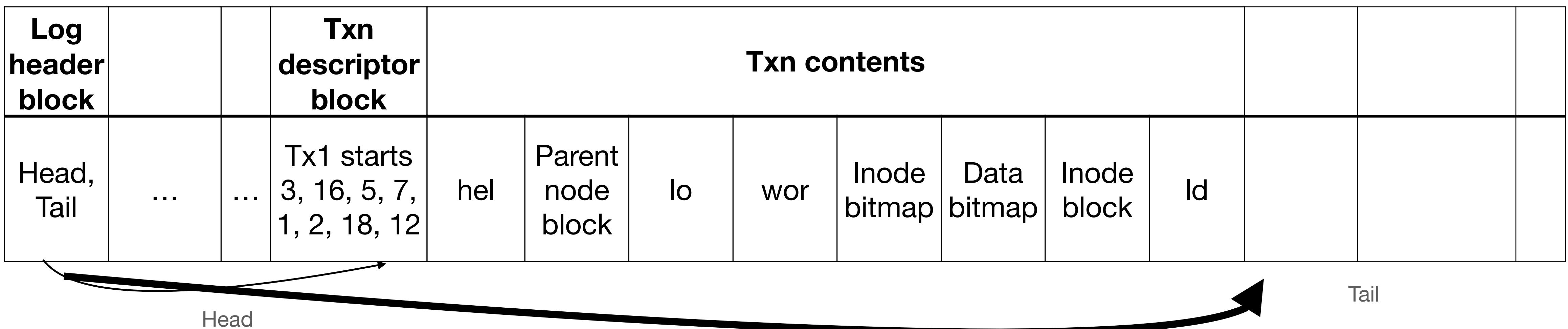
Transaction end

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents



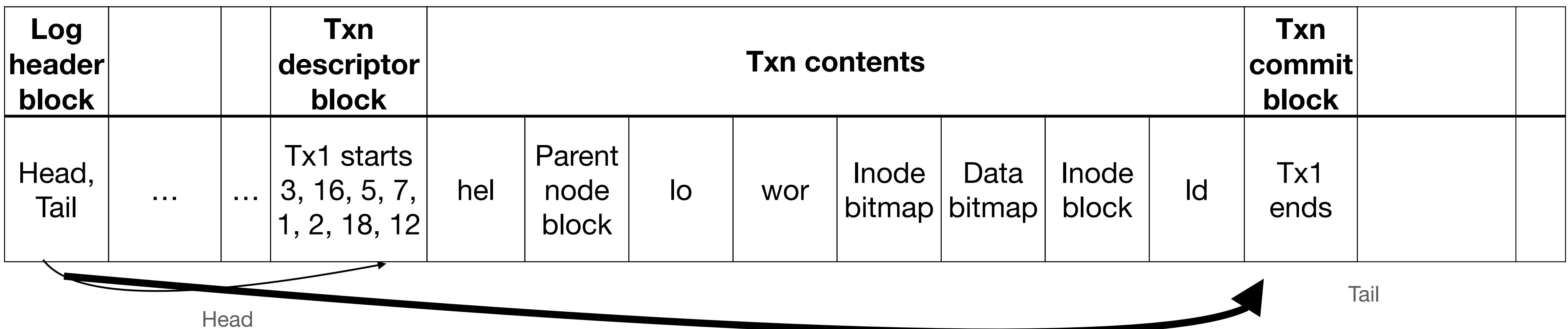
Transaction end

- Update tail in log header block
- Write transaction commit block
- Write transaction descriptor block
- Write transaction contents



Transaction end

- Update tail in log header block
- Write transaction commit block
- Write transaction descriptor block
- Write transaction contents



Transaction end

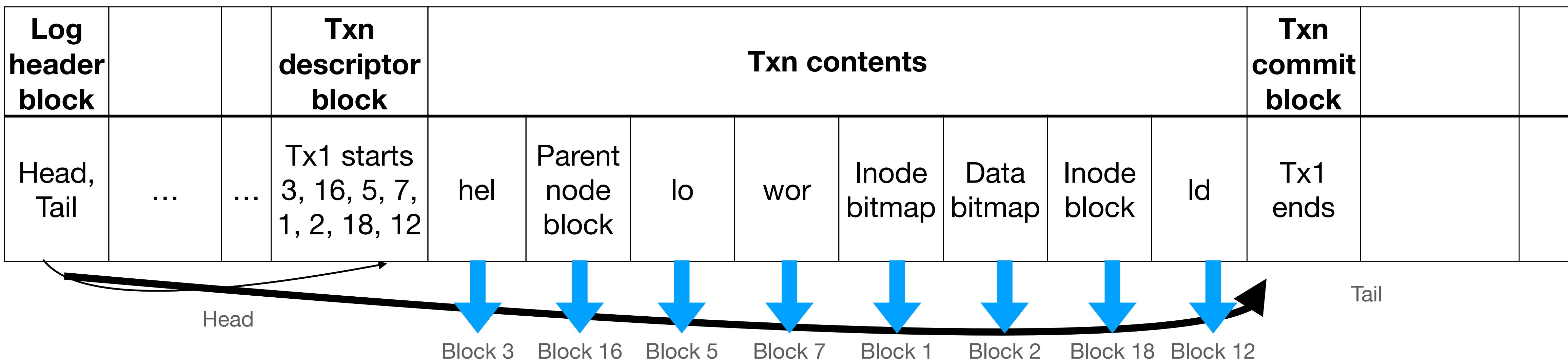
- Update tail in log header block
 - Write transaction commit block
 - Write transaction descriptor block
 - Write transaction contents to home locations
 - Write transaction contents

Log header block			Txn descriptor block	Txn contents						Txn commit block		
Head, Tail	Tx1 starts 3, 16, 5, 7, 1, 2, 18, 12	hel	Parent node block	lo	wor	Inode bitmap	Data bitmap	Inode block	ld	Tx1 ends



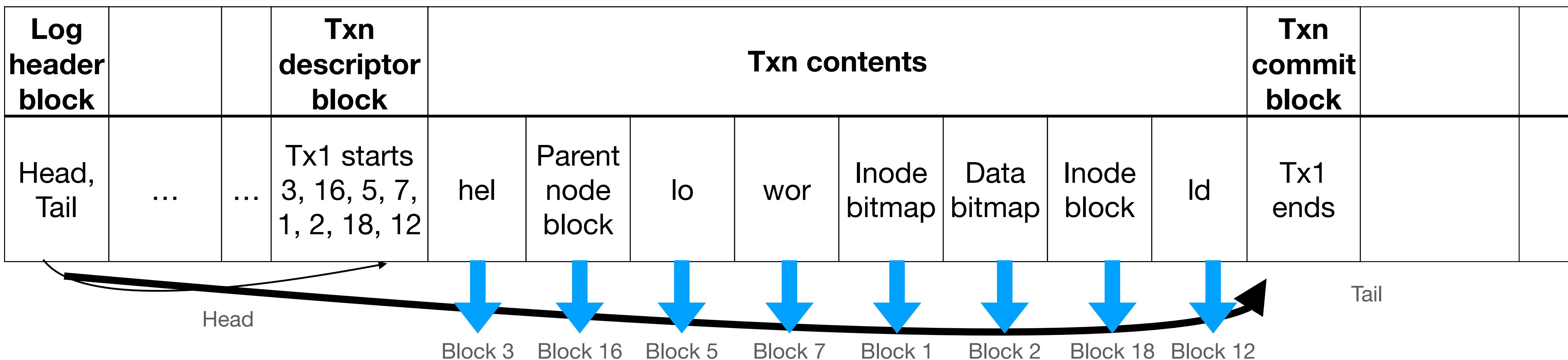
Transaction end

- Update tail in log header block
- Write transaction commit block
- Write transaction descriptor block
- Write transaction contents to home locations
- Write transaction contents



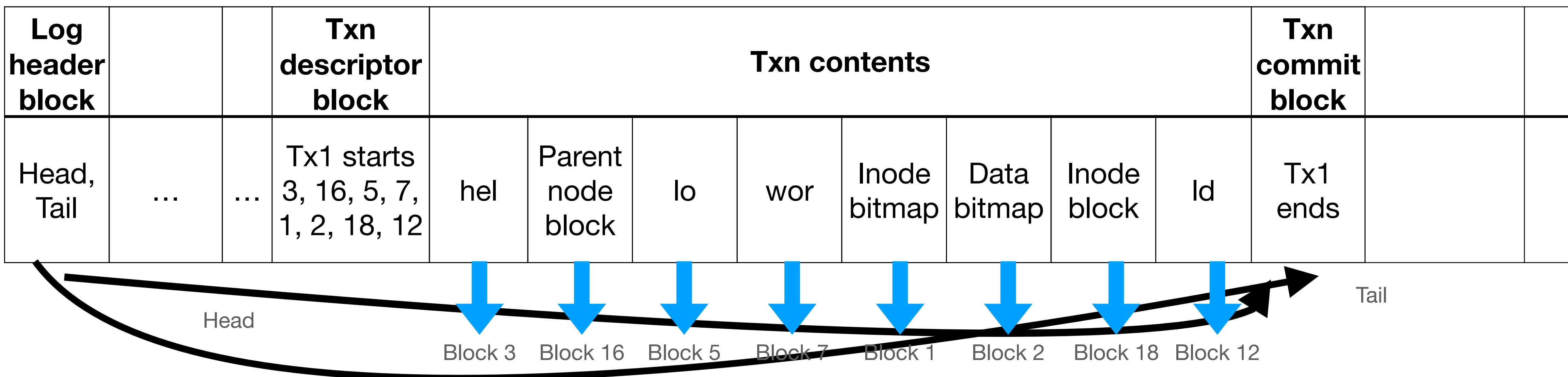
Transaction end

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents
- Write transaction commit block
- Write transaction contents to home locations
- Update head in log header block



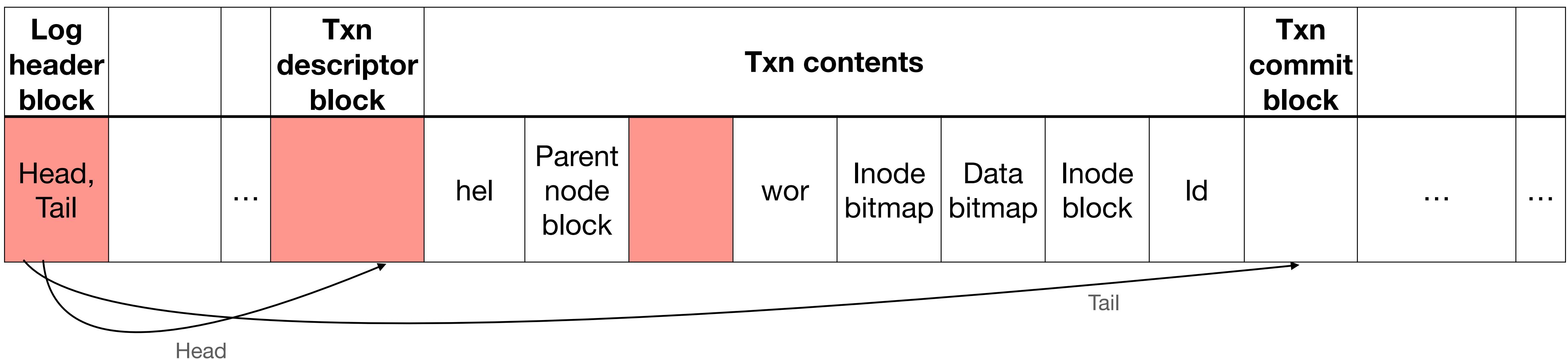
Transaction end

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents
- Write transaction commit block
- Write transaction contents to home locations
- Update head in log header block



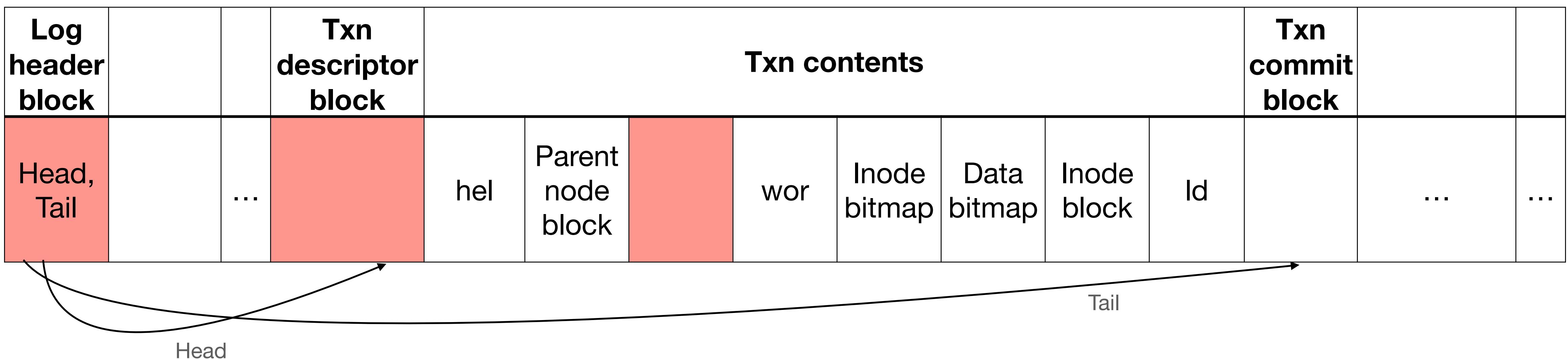
Three writes can be unordered

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents



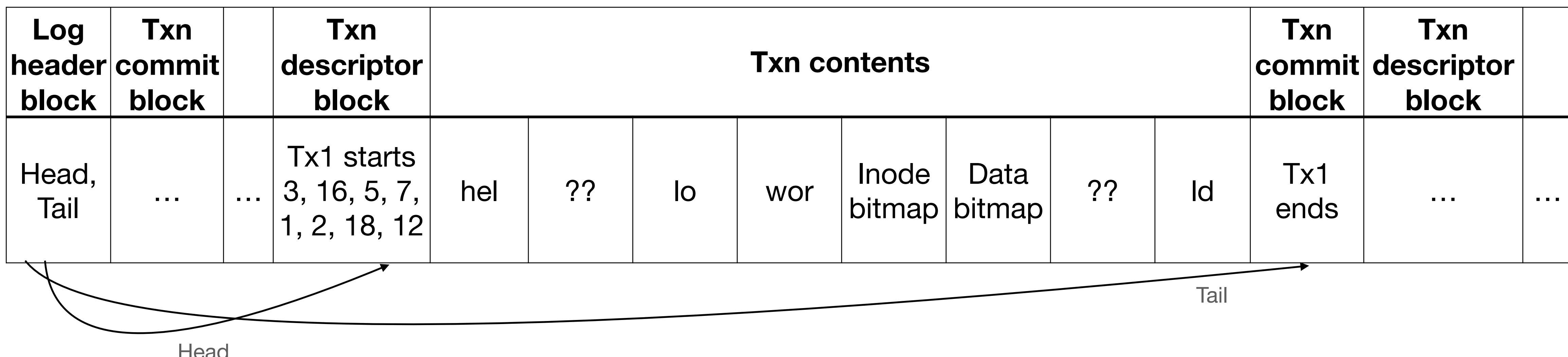
Three writes can be unordered

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents
- Recovery scans for “Tx end” backwards



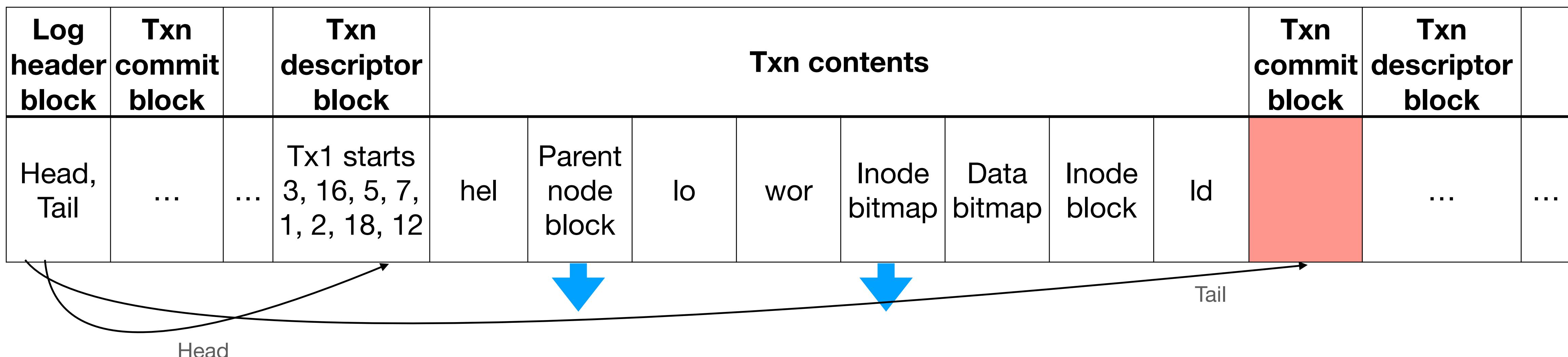
Why transaction commit needs to be ordered?

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents
- Write transaction commit block
- Write transaction contents to home locations
- Update head in log header block



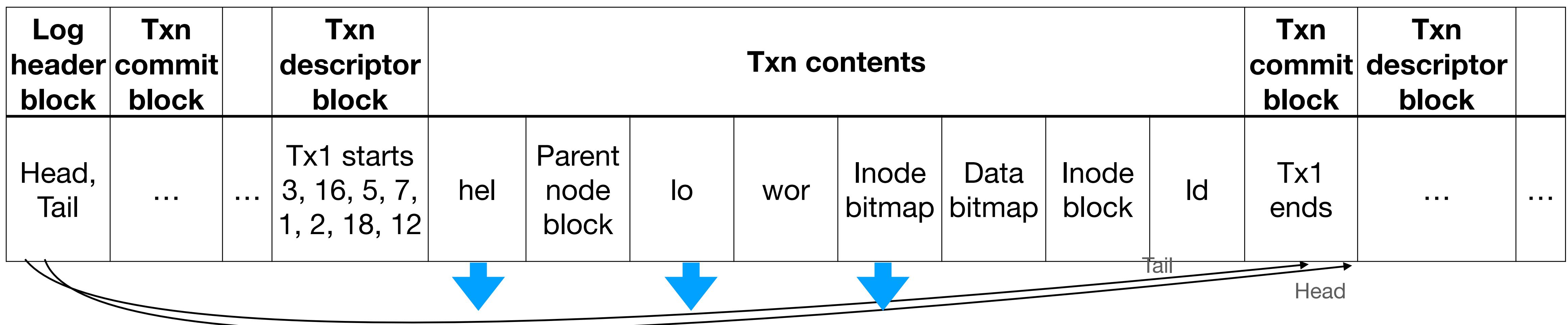
Writing to home locations need to be ordered

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents
- Write transaction commit block
- Write transaction contents to home locations
- Update head in log header block



Transaction end

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents
- Write transaction commit block
- Write transaction contents to home locations
- Update head in log header block



Log can have multiple transactions

- Close the transaction after 30 seconds

```
begin_txn(26);
```

Log can have multiple transactions

- Close the transaction after 30 seconds

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Log can have multiple transactions

- Close the transaction after 30 seconds
- At close, send disk write requests to write the transaction's dirty buffers

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Log can have multiple transactions

- Close the transaction after 30 seconds
- At close, send disk write requests to write the transaction's dirty buffers
- Need not wait for the writes to be on disk! Start a new transaction.

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Log can have multiple transactions

- Close the transaction after 30 seconds
- At close, send disk write requests to write the transaction's dirty buffers
- Need not wait for the writes to be on disk! Start a new transaction.

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
}
```

Log can have multiple transactions

- Close the transaction after 30 seconds
- At close, send disk write requests to write the transaction's dirty buffers
- Need not wait for the writes to be on disk! Start a new transaction.
- When old transaction's dirty buffers are written to their home locations, mark them as valid. They may now be evicted.

```
begin_txn(26);
```

```
    write(fd, ...){  
        begin_op(..);  
        bwrite( .. );  
        bwrite( .. );  
        bwrite( .. );  
        end_op(..);  
    }
```

```
    write(fd, ...){  
        begin_op(..);  
        bwrite( .. );  
        bwrite( .. );  
        bwrite( .. );  
        end_op(..);  
    }
```

```
begin_txn(27);
```

```
    write(fd, ...){  
        begin_op(..);  
        bwrite( .. );  
        bwrite( .. );  
        bwrite( .. );  
        end_op(..);  
    }
```

begin_txn(26);

Log can have multiple transactions!

Buffer cache

Log

begin_txn(26);

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

Log can have multiple transactions!

Buffer cache

Log

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
---------	--------	-------

Log

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log

```
Txn 26  
1, 3, 5
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log

Txn 26 1, 3, 5	111111
-------------------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log

Txn 26 1, 3, 5	111111	333333
-------------------	--------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log

Txn 26 1, 3, 5	111111	333333
-------------------	--------	--------

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log

Txn 26 1, 3, 5	111111	333333
-------------------	--------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY
Block 2	222222	DIRTY

Log

Txn 26 1, 3, 5	111111	333333
-------------------	--------	--------

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);  
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY
Block 2	222222	DIRTY

Log

Txn 26 1, 3, 5	111111	333333	555555
-------------------	--------	--------	--------

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);  
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY
Block 2	222222	DIRTY

Log

Txn 26 1, 3, 5	111111	333333	555555	Txn 26 commit
-------------------	--------	--------	--------	------------------

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

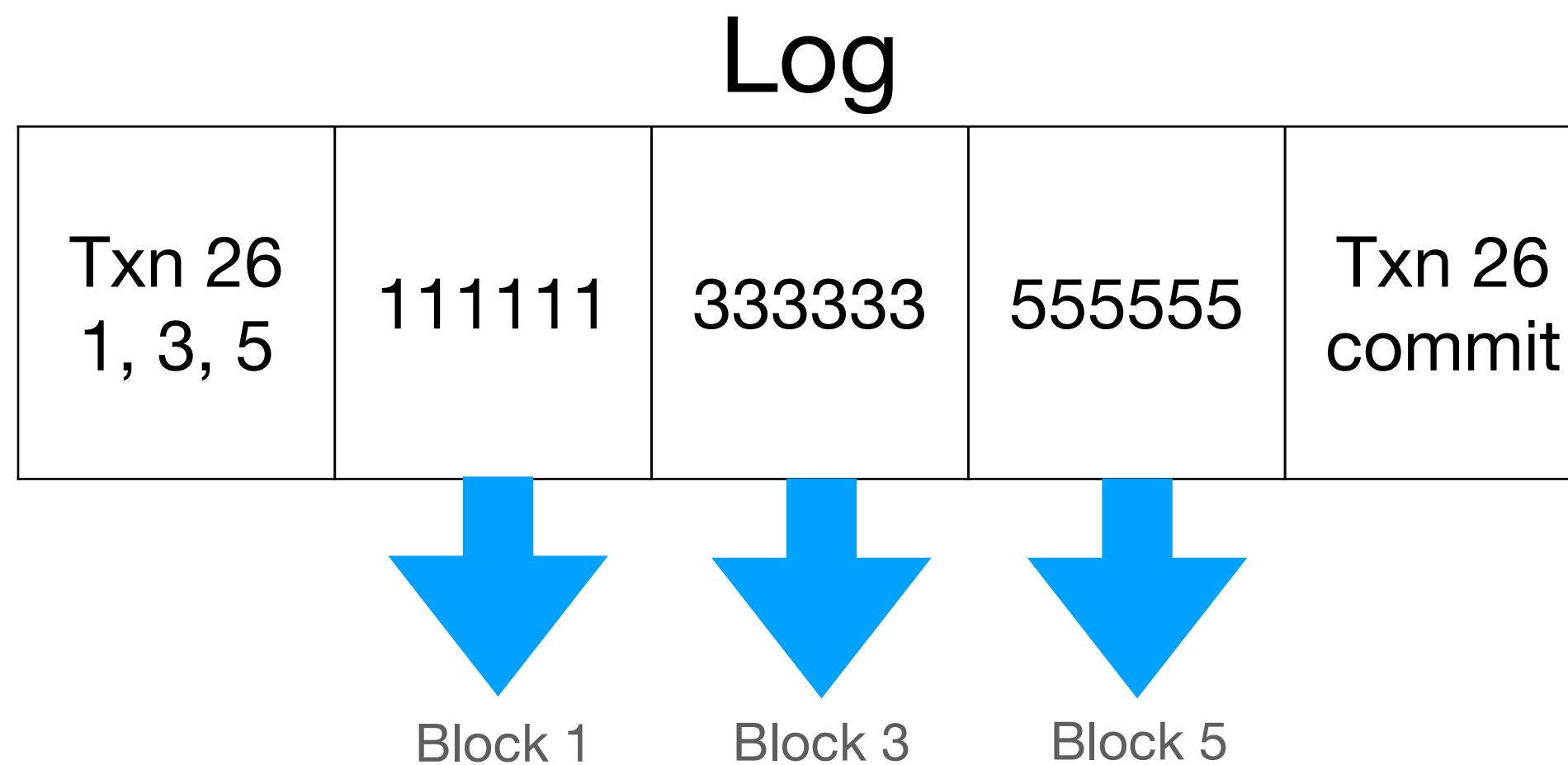
```
begin_txn(27);  
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY
Block 2	222222	DIRTY



```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

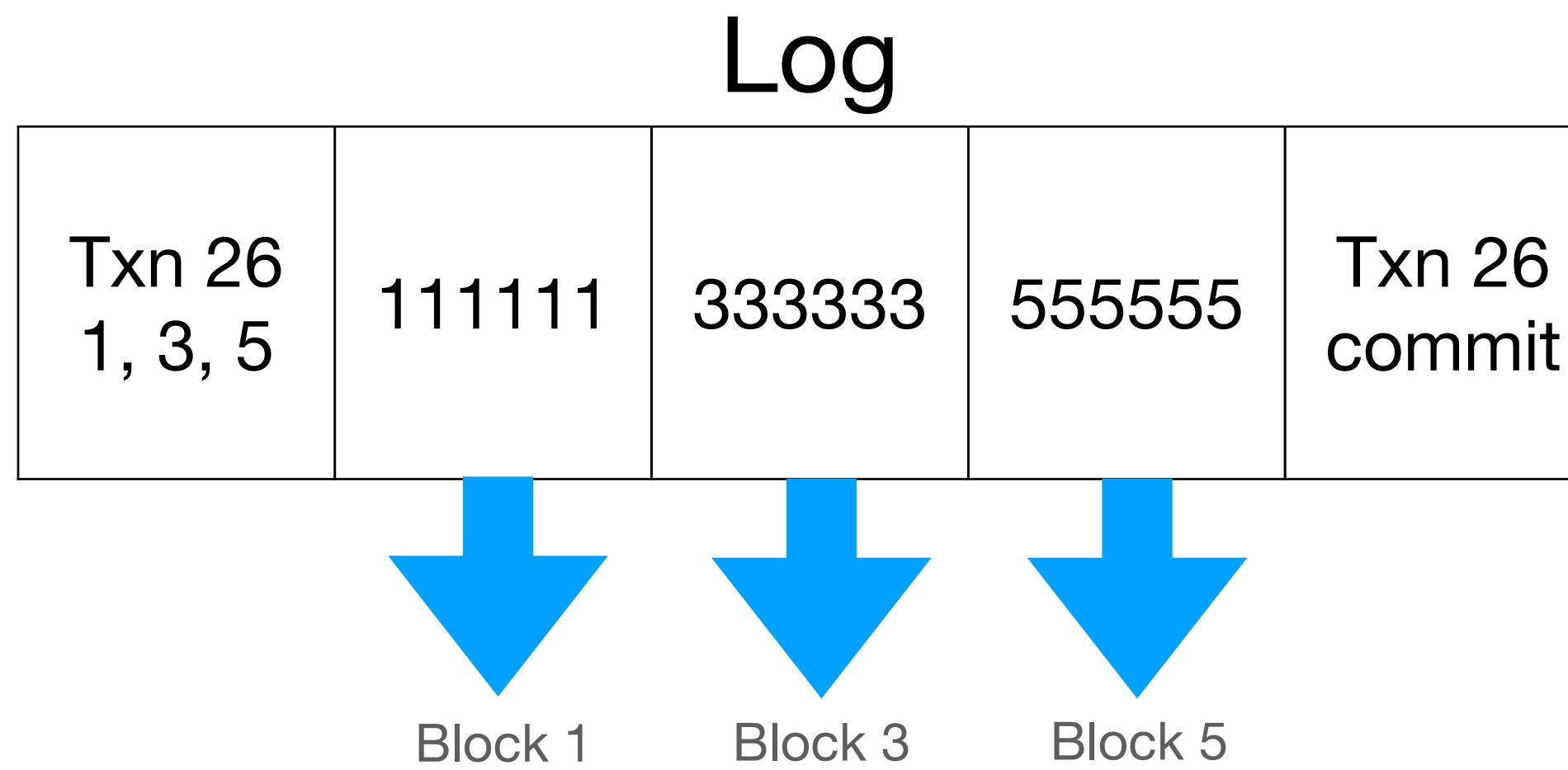
```
begin_txn(27);  
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY
Block 2	222222	DIRTY
Block 4	444444	DIRTY



```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

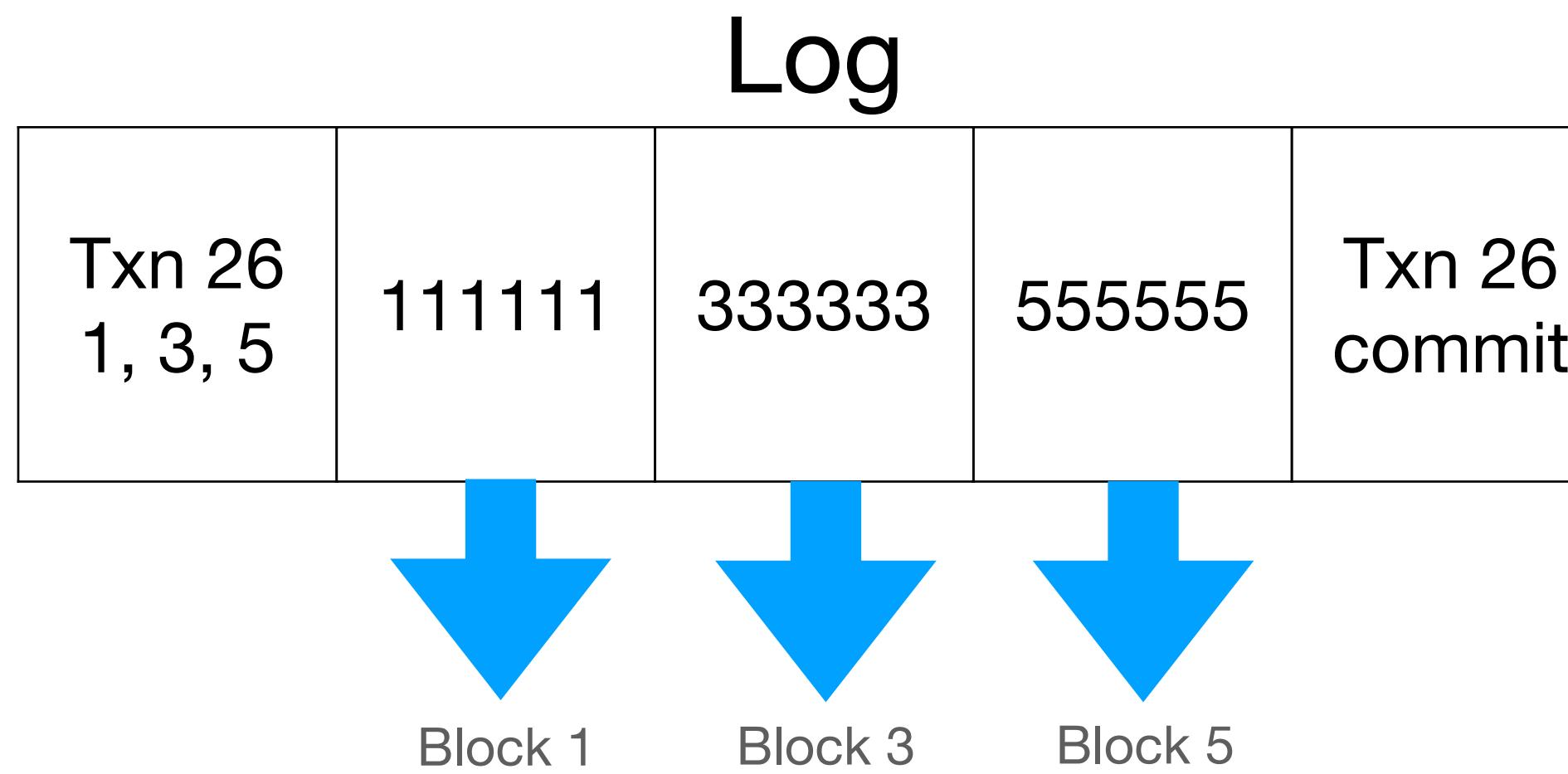
```
begin_txn(27);  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Log can have multiple transactions!

Buffer cache

Block 1	111111	VALID
Block 3	333333	VALID
Block 5	555555	VALID
Block 2	222222	DIRTY
Block 4	444444	DIRTY



```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);  
write(fd, ...){  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
}
```

begin_txn(26);

Buffer conflicts

Buffer cache (memory)

Log (disk)

begin_txn(26);

Buffer conflicts

Buffer cache (memory)

Log (disk)

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

begin_txn(26);

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
---------	--------	-------

Log (disk)

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	333333	DIRTY

Log (disk)

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	333333	DIRTY

Log (disk)

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log (disk)

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log (disk)

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log (disk)

```
Txn 26  
1, 3, 5
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log (disk)

Txn 26 1, 3, 5	111111
-------------------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log (disk)

Txn 26 1, 3, 5	111111
-------------------	--------

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log (disk)

Txn 26 1, 3, 5	111111
-------------------	--------

```
begin_txn(26);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	999999	DIRTY
Block 5	555555	DIRTY

Log (disk)

Txn 26 1, 3, 5	111111
-------------------	--------

```
begin_txn(26);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	999999	DIRTY
Block 5	555555	DIRTY

Log (disk)

Txn 26 1, 3, 5	111111	999999
-------------------	--------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	999999	DIRTY
Block 5	555555	DIRTY

Log (disk)

Txn 26 1, 3, 5	111111	999999	555555
-------------------	--------	--------	--------

```
begin_txn(26);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	999999	DIRTY
Block 5	555555	DIRTY

Log (disk)

Txn 26 1, 3, 5	111111	999999	555555	Txn 26 commit
-------------------	--------	--------	--------	------------------

```
begin_txn(26);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	999999	DIRTY
Block 5	555555	DIRTY
Block 2	222222	DIRTY

Log (disk)

Txn 26 1, 3, 5	111111	999999	555555	Txn 26 commit
-------------------	--------	--------	--------	------------------

```
begin_txn(26);
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}
```

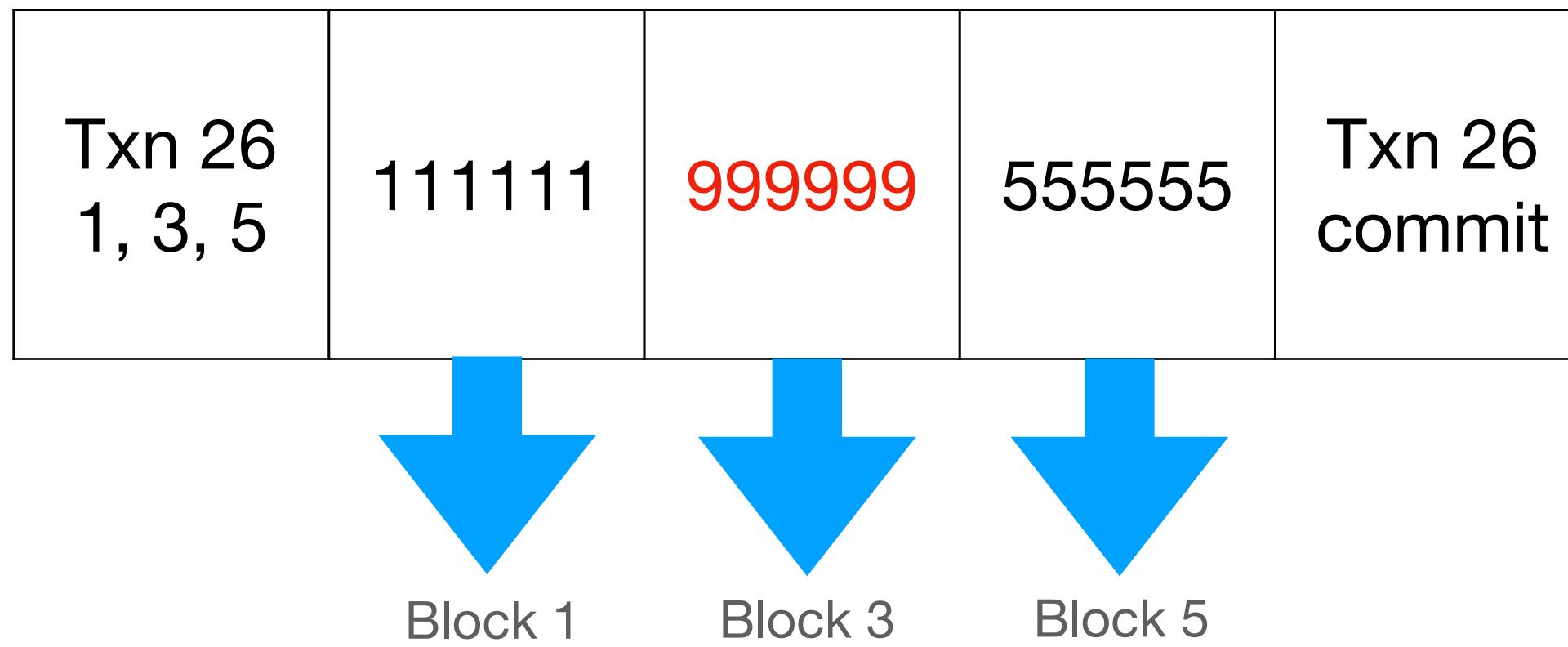
```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	999999	DIRTY
Block 5	555555	DIRTY
Block 2	222222	DIRTY

Log (disk)



```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

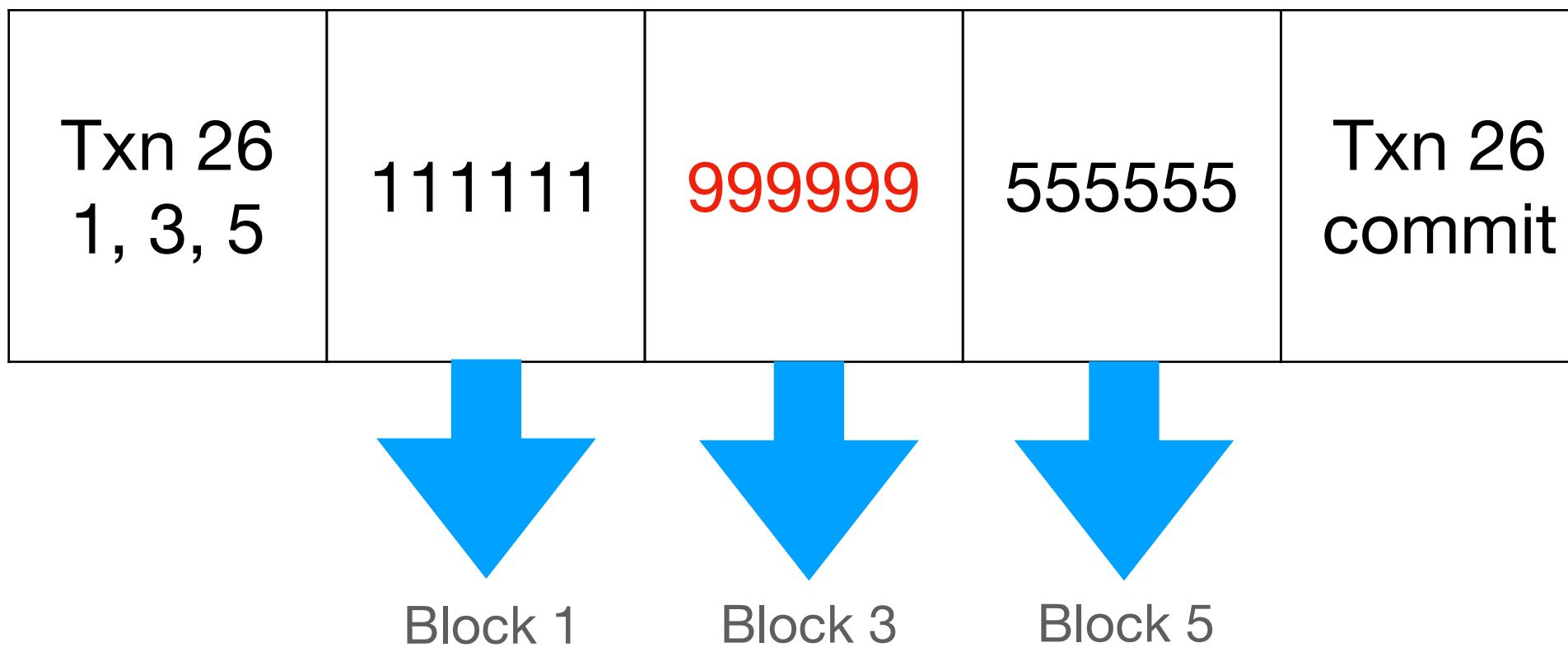
```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	DIRTY
Block 3	999999	DIRTY
Block 5	555555	DIRTY
Block 2	222222	DIRTY
Block 4	444444	DIRTY

Log (disk)



```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn();
```

```
begin_txn(27);  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

}

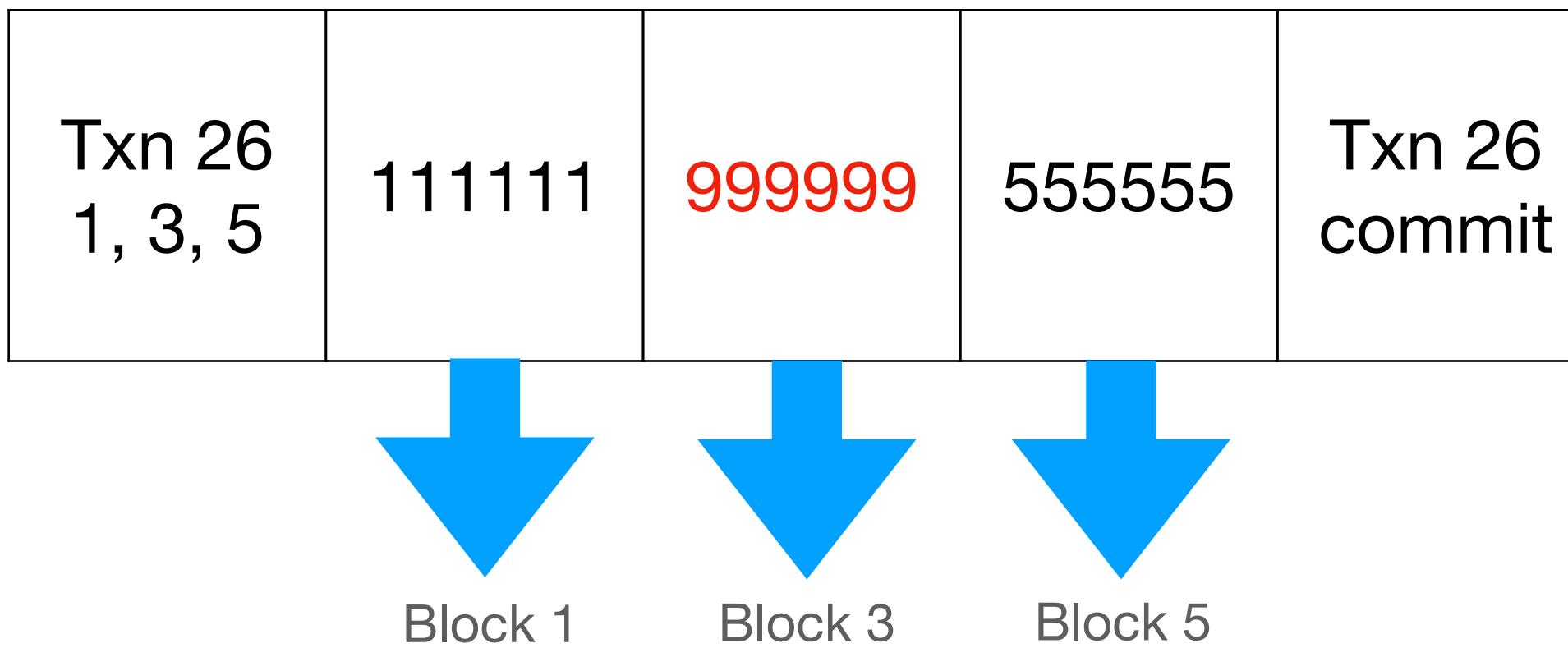
```
begin_txn(26);
```

Buffer conflicts

Buffer cache (memory)

Block 1	111111	VALID
Block 3	999999	VALID
Block 5	555555	VALID
Block 2	222222	DIRTY
Block 4	444444	DIRTY

Log (disk)



```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

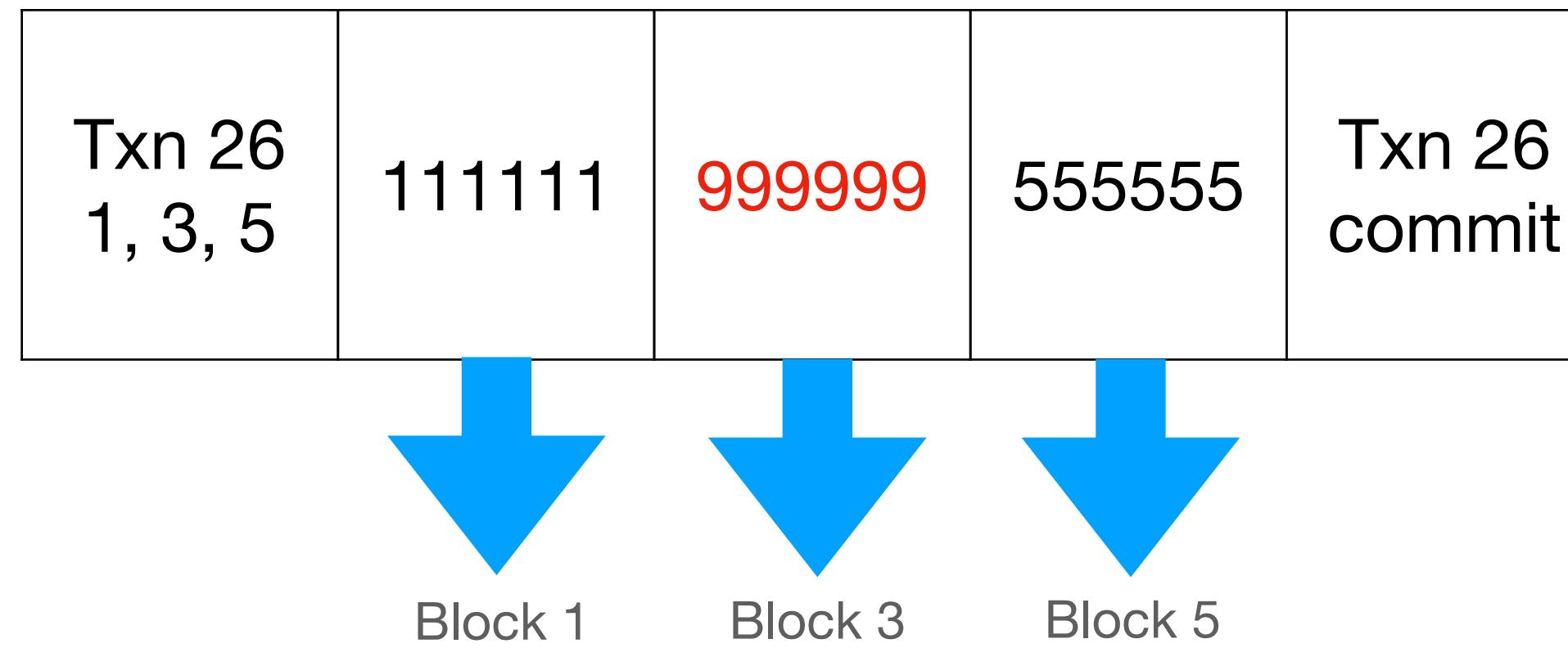
}

Buffer conflicts

Buffer cache (memory)

Block 1	111111	VALID
Block 3	999999	VALID
Block 5	555555	VALID
Block 2	222222	DIRTY
Block 4	444444	DIRTY

Log (disk)



- Uncommitted txn 27 wrote a block!

```
begin_txn(26);
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}  
end_txn();
```

```
begin_txn(27);
```

```
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}
```

```
}
```

begin_txn(26);

Buffer conflicts: Copy on write

Buffer cache

Log

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY
---------	--------	-------

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY
Block 3	333333	DIRTY
Block 5	555555	DIRTY

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO

Log

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO

Log

```
Txn 26  
1, 3, 5
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO

Log

Txn 26 1, 3, 5	111111
-------------------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO

Log

Txn 26 1, 3, 5	111111
-------------------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
end_txn(..);
```

```
begin_txn(27);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO

Log

Txn 26 1, 3, 5	111111
-------------------	--------

```
begin_txn(26);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO
Block 3	999999	DIRTY

Log

Txn 26 1, 3, 5	111111
-------------------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
end_txn();
```

```
begin_txn(27);  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO
Block 3	999999	DIRTY
Block 2	222222	DIRTY

Log

Txn 26 1, 3, 5	111111
-------------------	--------

```
begin_txn(26);  
  
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}  
  
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}  
}  
  
end_txn();  


---

  
begin_txn(27);  
write(fd, ...){  
    begin_op(..);  
    bwrite(..);  
    bwrite(..);  
    bwrite(..);  
    end_op(..);  
}  
}
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO
Block 3	999999	DIRTY
Block 2	222222	DIRTY

Log

Txn 26 1, 3, 5	111111	333333
-------------------	--------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
end_txn();
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
}
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO
Block 3	999999	DIRTY
Block 2	222222	DIRTY

Log

Txn 26 1, 3, 5	111111	333333	555555
-------------------	--------	--------	--------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}  
end_txn(..);
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO
Block 3	999999	DIRTY
Block 2	222222	DIRTY

Log

Txn 26 1, 3, 5	111111	333333	555555	Txn 26 commit
-------------------	--------	--------	--------	------------------

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
end_txn();
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
}
```

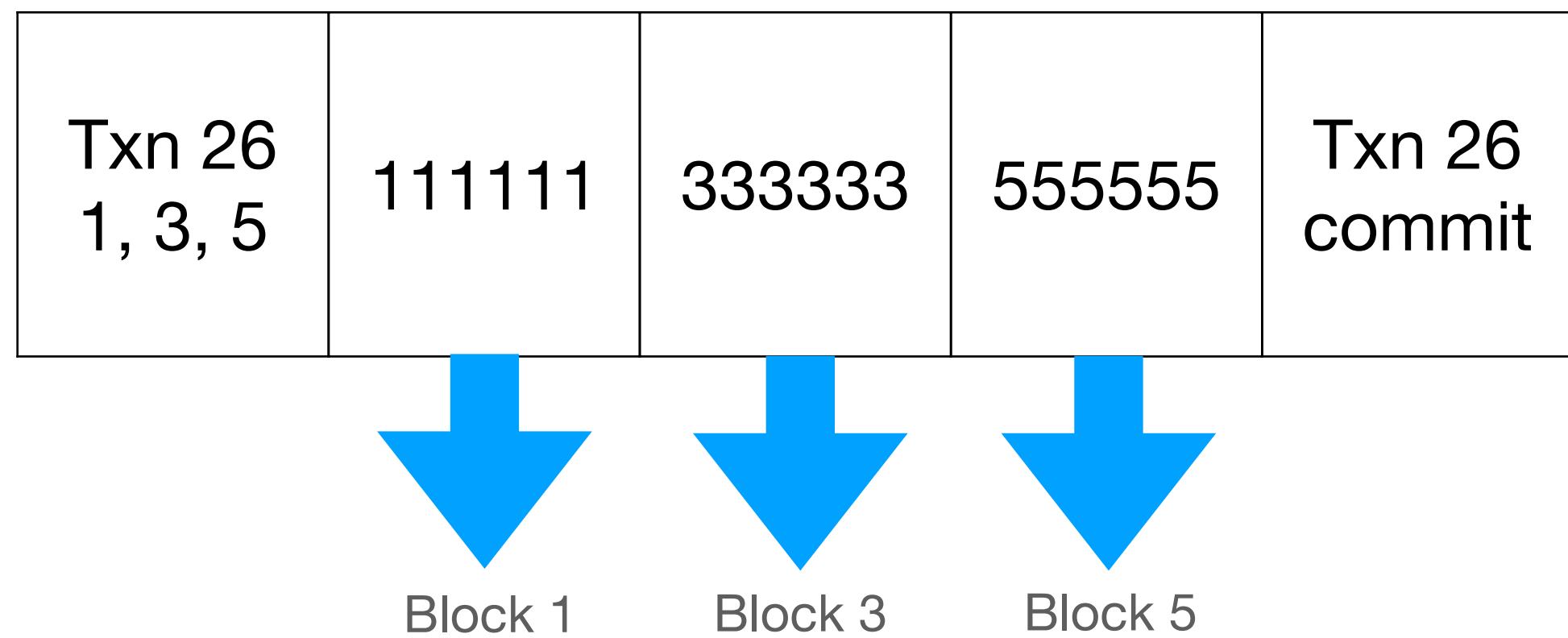
```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	DIRTY, RO
Block 3	333333	DIRTY, RO
Block 5	555555	DIRTY, RO
Block 3	999999	DIRTY
Block 2	222222	DIRTY

Log



```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
end_txn();
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
}
```

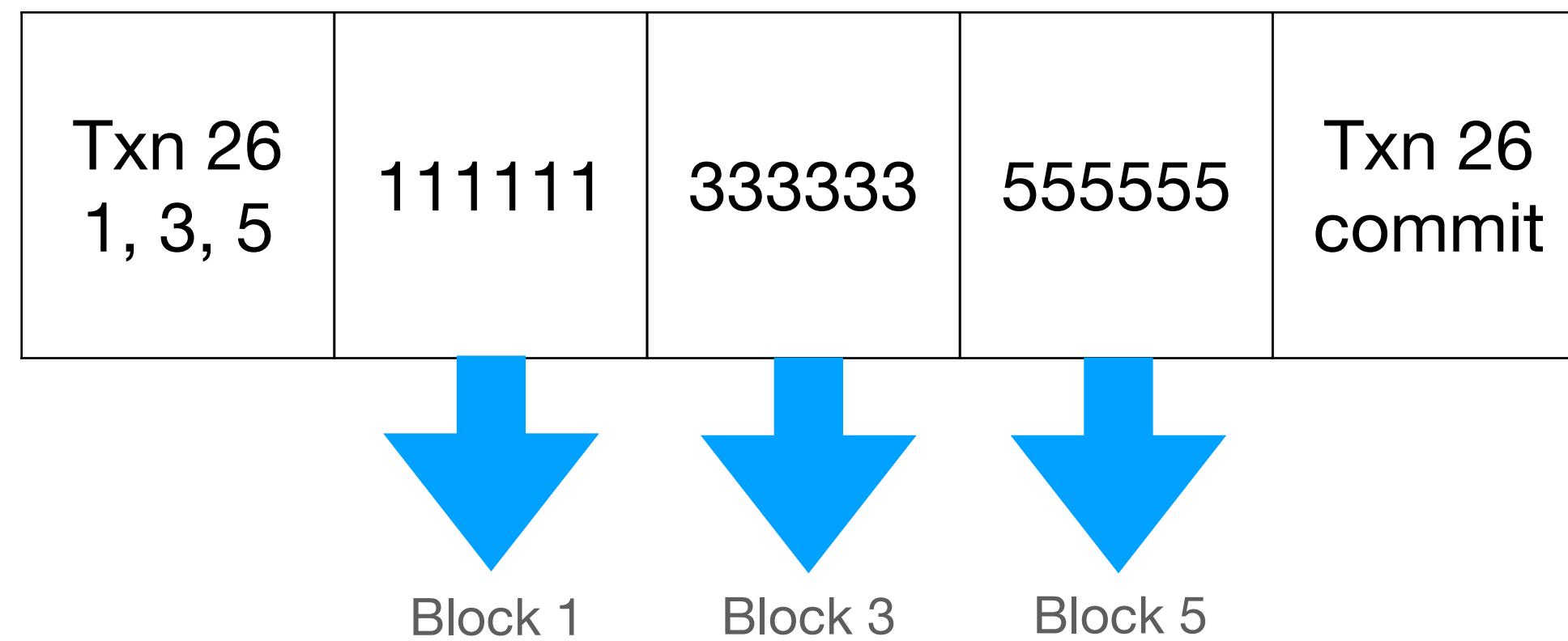
```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	VALID
Block 3	333333	DELETE
Block 5	555555	VALID
Block 3	999999	DIRTY
Block 2	222222	DIRTY

Log



```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
end_txn();
```

```
begin_txn(27);  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

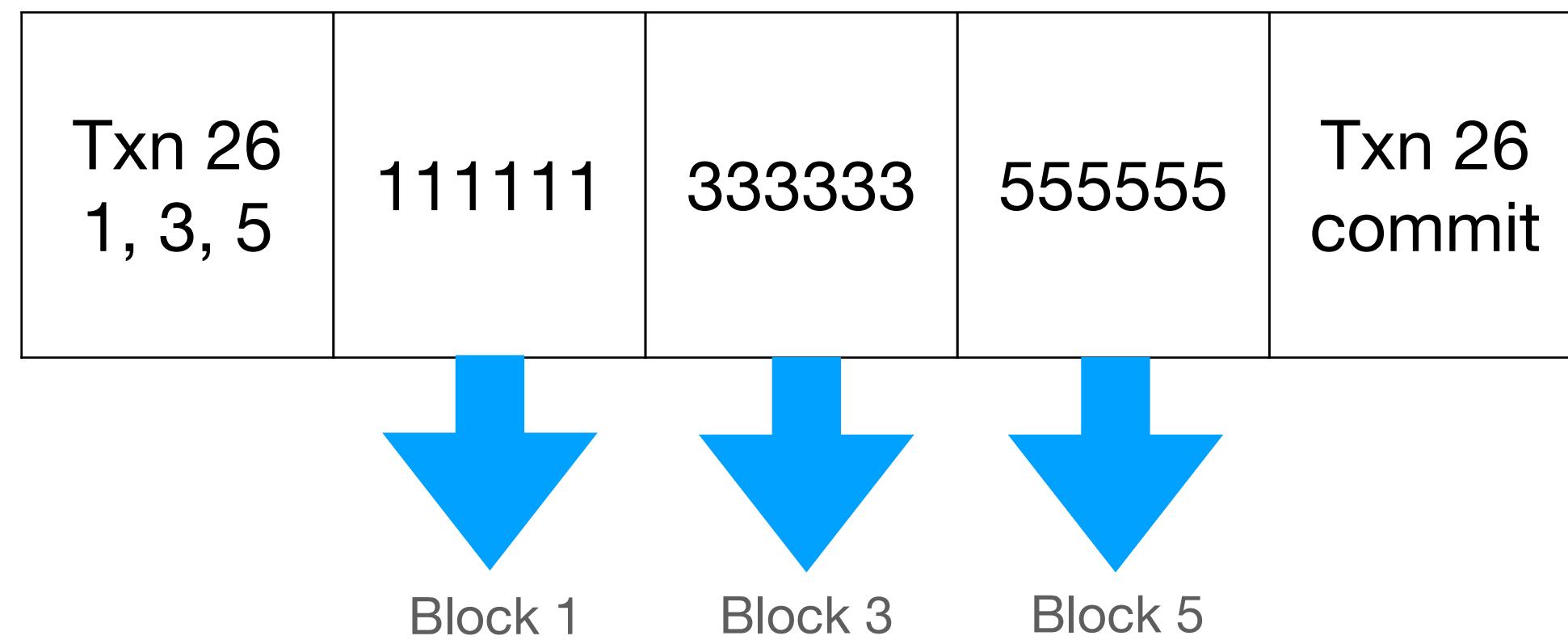
```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	VALID
Block 3	333333	DELETE
Block 5	555555	VALID
Block 3	999999	DIRTY
Block 2	222222	DIRTY
Block 4	444444	DIRTY

Log



```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
end_txn();
```

```
begin_txn(27);
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
}
```

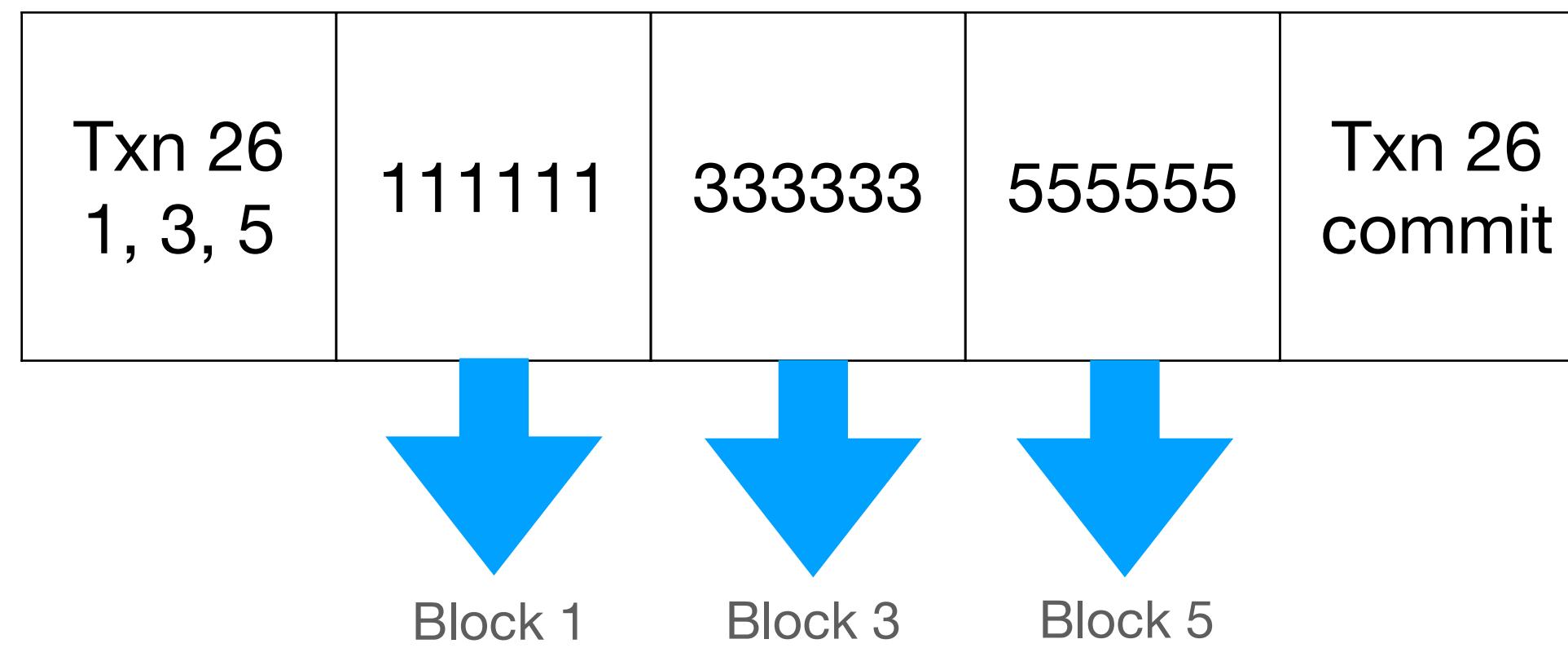
```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	VALID
Block 3	333333	DELETE
Block 5	555555	VALID
Block 3	999999	DIRTY
Block 2	222222	DIRTY
Block 4	444444	DIRTY

Log



- If new transaction writes to an old transaction's buffer, write to a new copy!

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
end_txn();
```

```
begin_txn(27);  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
}
```

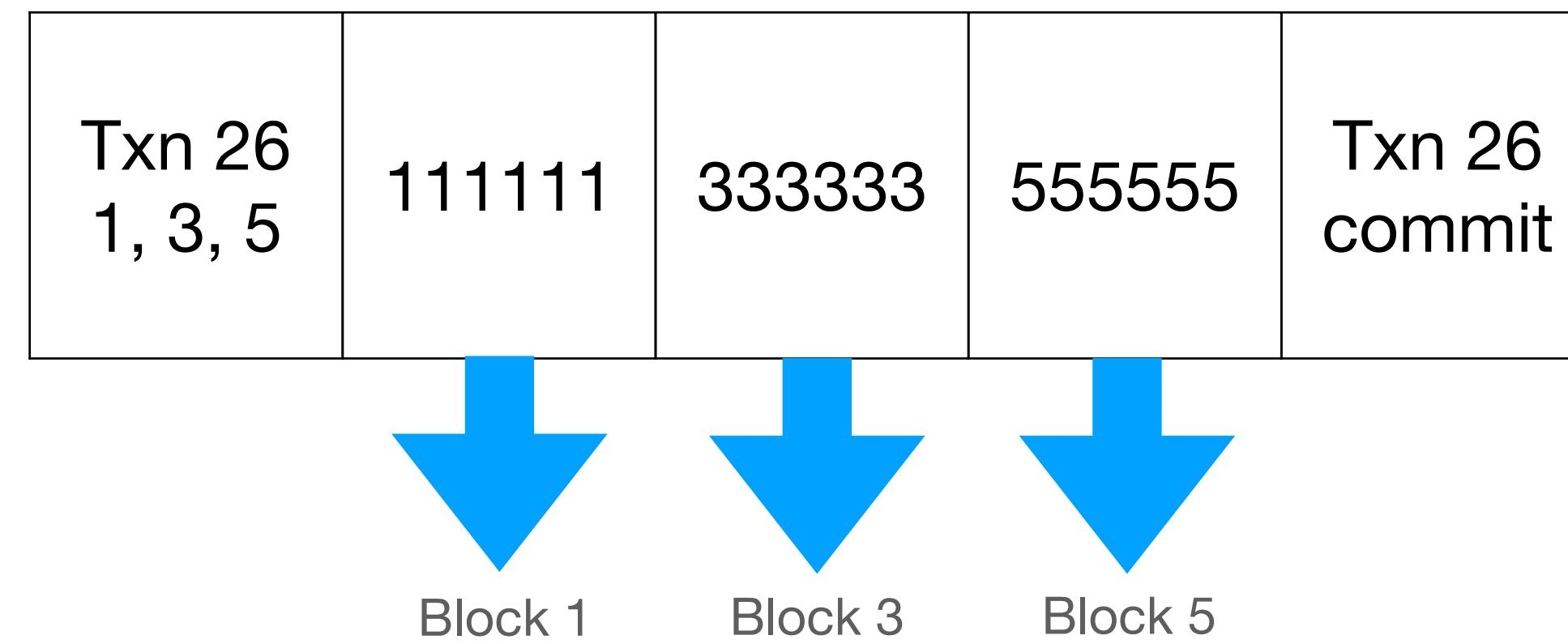
```
begin_txn(26);
```

Buffer conflicts: Copy on write

Buffer cache

Block 1	111111	VALID
Block 3	333333	DELETE
Block 5	555555	VALID
Block 3	999999	DIRTY
Block 2	222222	DIRTY
Block 4	444444	DIRTY

Log



- If new transaction writes to an old transaction's buffer, write to a new copy!
- Old copy is written to the log and disk. After it is written to home location, delete.

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

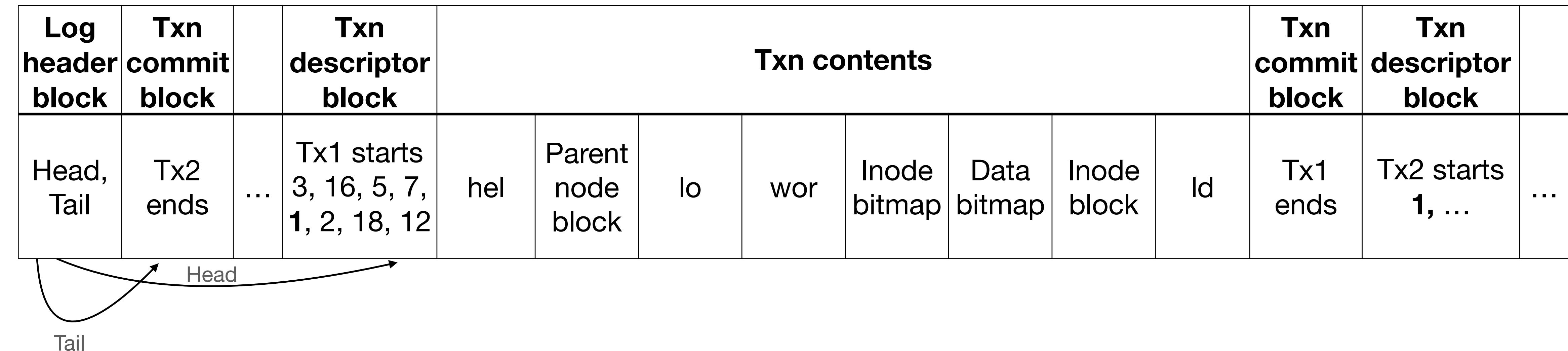
```
end_txn();
```

```
begin_txn(27);  
write(fd, ...) {  
    begin_op(..);  
    bwrite( .. );  
    bwrite( .. );  
    bwrite( .. );  
    end_op(..);  
}
```

```
}
```

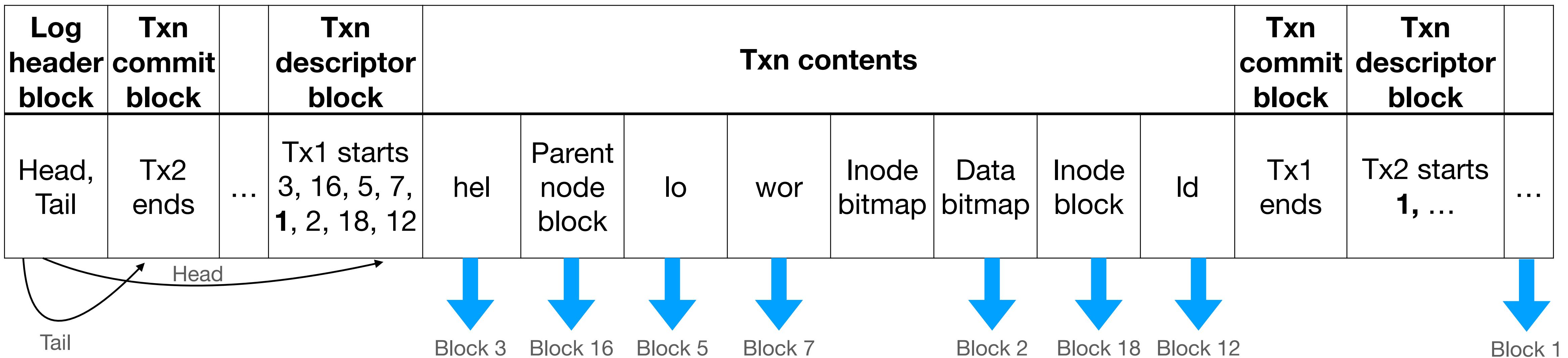
Recovery procedure with multiple transactions

- At reboot:
 - read from head to tail
 - “Redo” Txn contents to home locations
 - Update head of log header block
 - Don’t write overwritten blocks



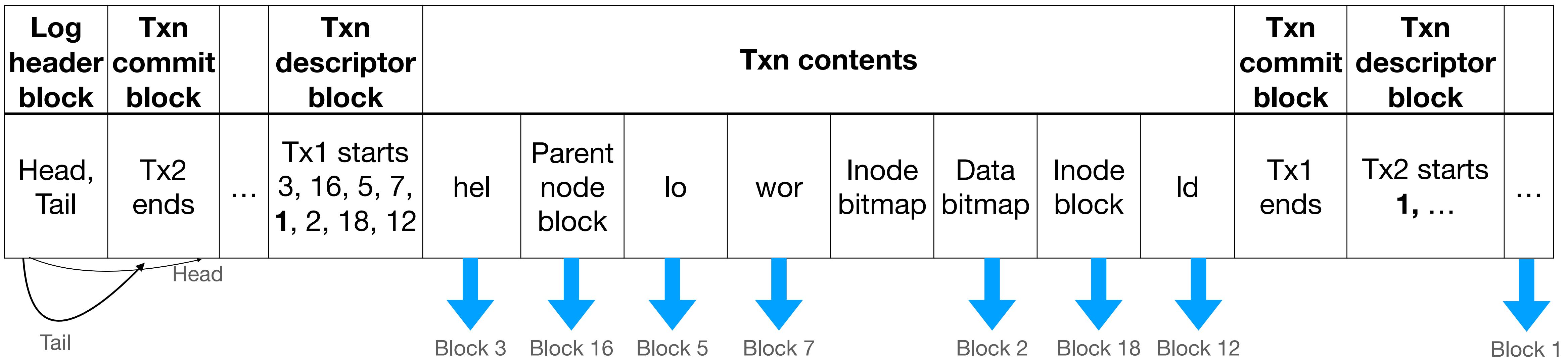
Recovery procedure with multiple transactions

- At reboot:
 - read from head to tail
 - “Redo” Txn contents to home locations
 - Update head of log header block
 - Don’t write overwritten blocks

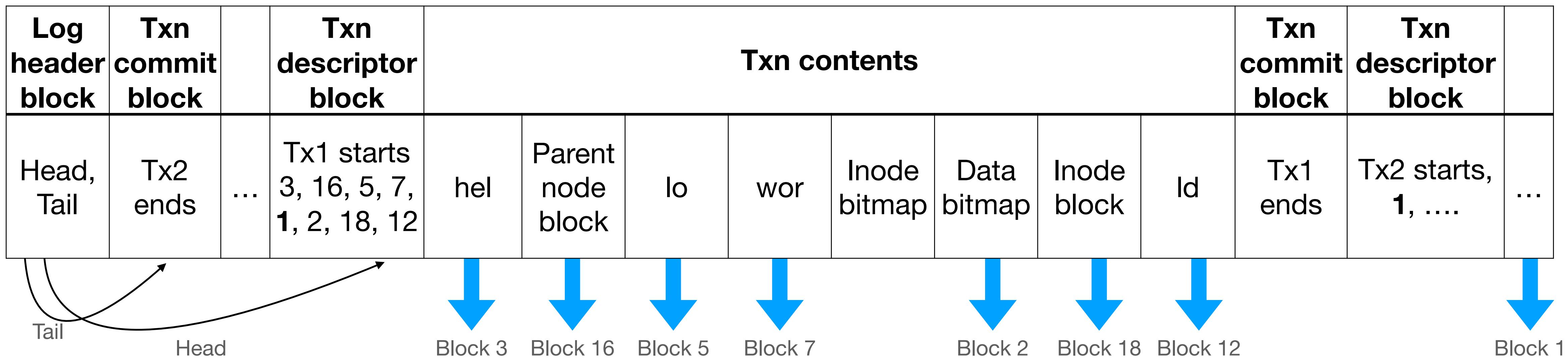


Recovery procedure with multiple transactions

- At reboot:
 - read from head to tail
 - “Redo” Txn contents to home locations
 - Update head of log header block
 - Don’t write overwritten blocks

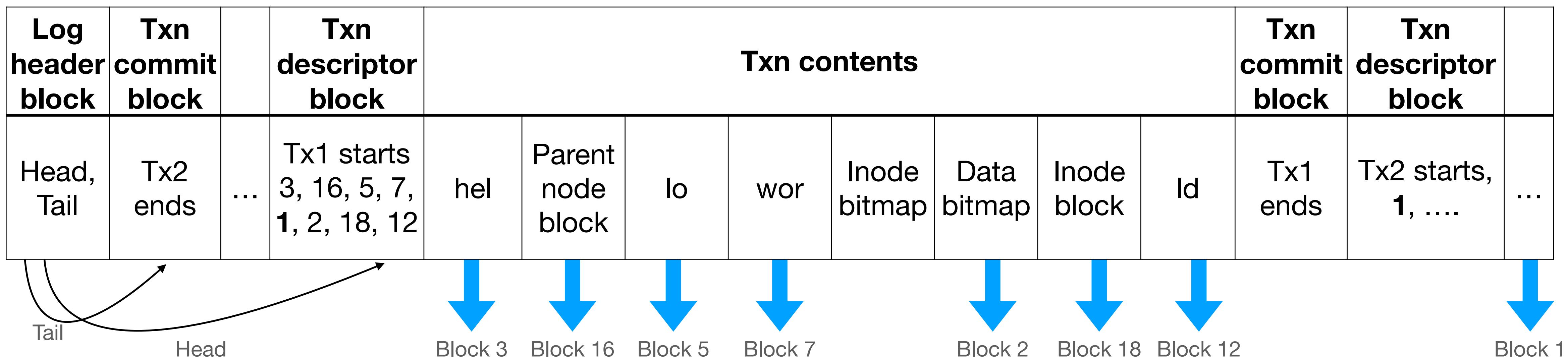


Performance



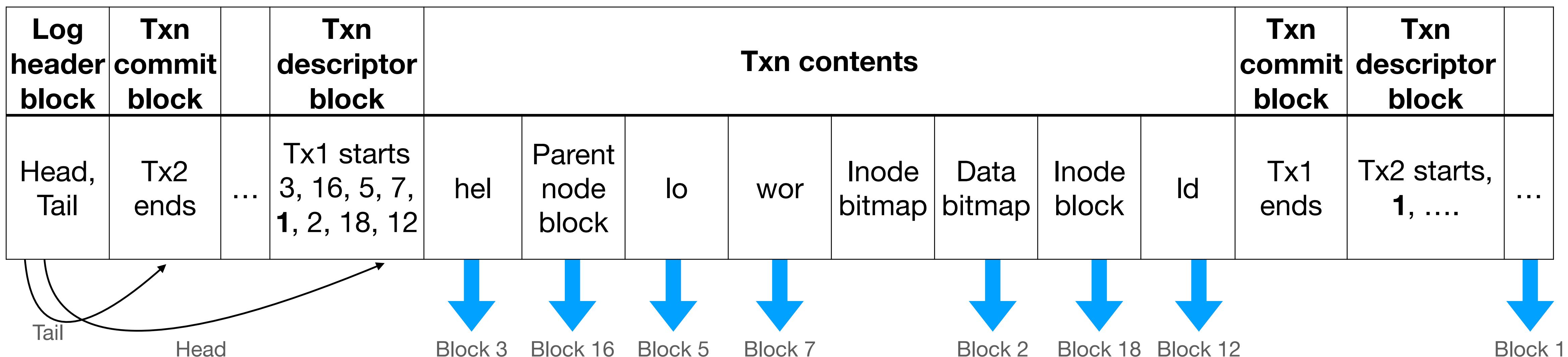
Performance

- Writing to log is one large sequential write. Durable as soon as commit block is written.



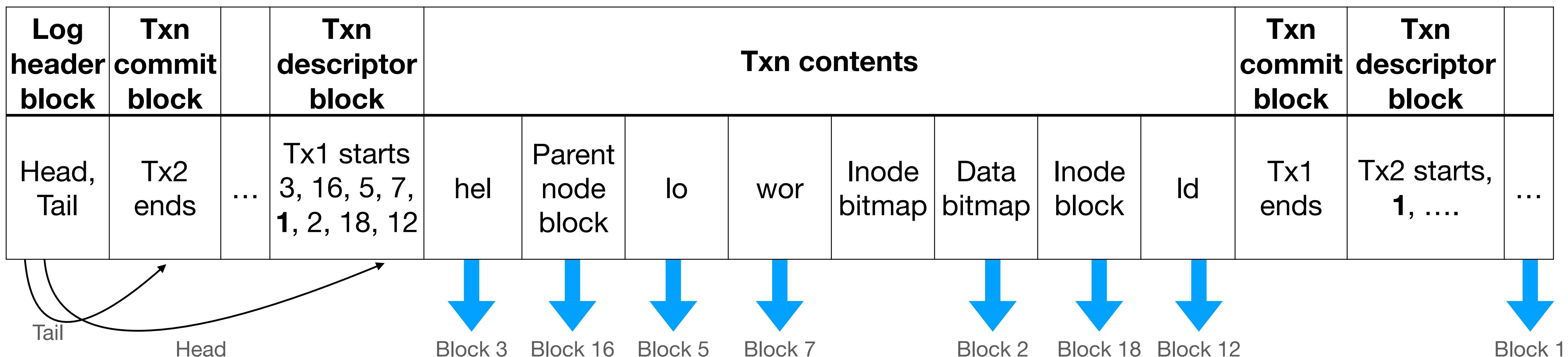
Performance

- Writing to log is one large sequential write. Durable as soon as commit block is written.
- Send bulk writes to home locations => good disk scheduling opportunity.



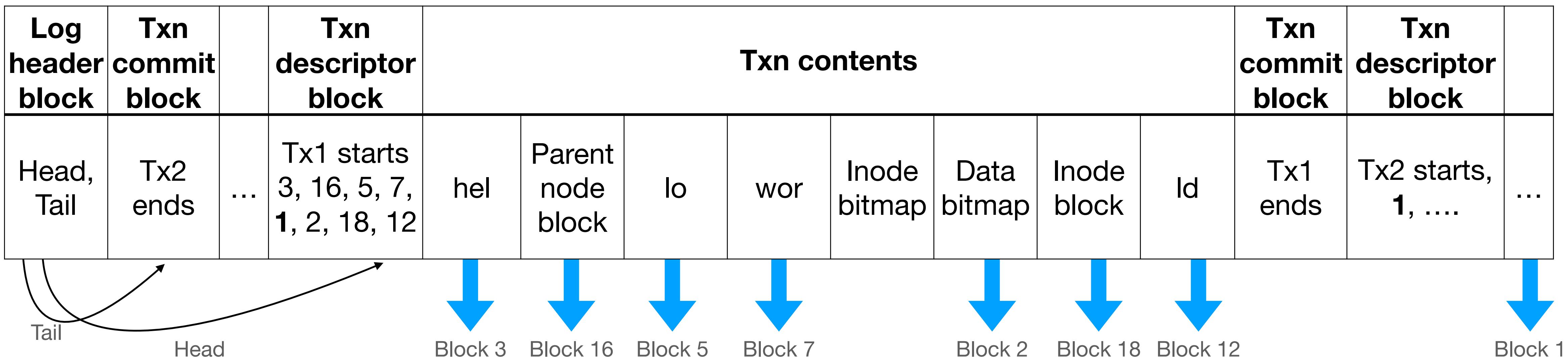
Performance

- Writing to log is one large sequential write. Durable as soon as commit block is written.
- Send bulk writes to home locations => good disk scheduling opportunity.
- All writes are off the “critical path”. New transactions are started while old one is applying writes.



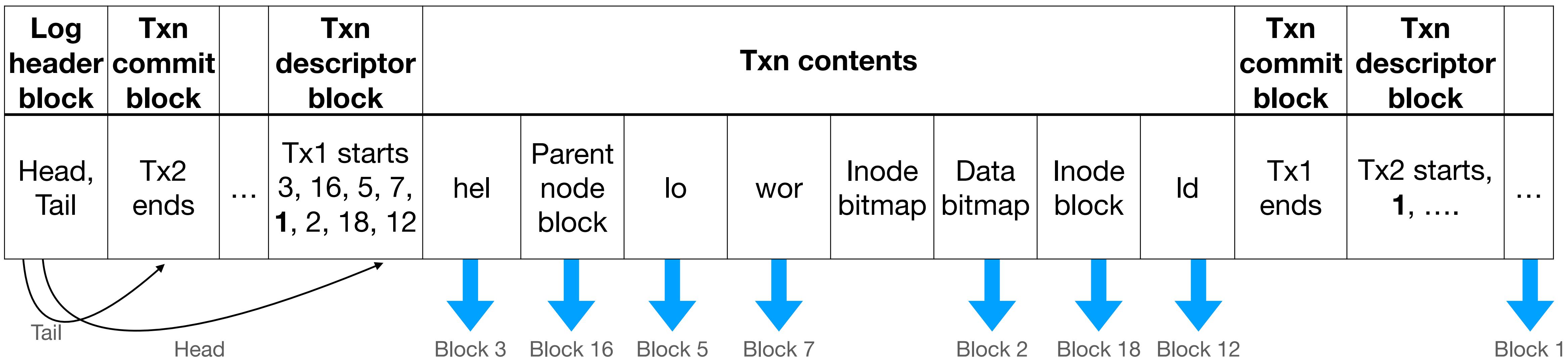
Performance

- Writing to log is one large sequential write. Durable as soon as commit block is written.
- Send bulk writes to home locations => good disk scheduling opportunity.
- All writes are off the “critical path”. New transactions are started while old one is applying writes.
- Recovery only reads the log and not the whole disk. Scales well with disk sizes.



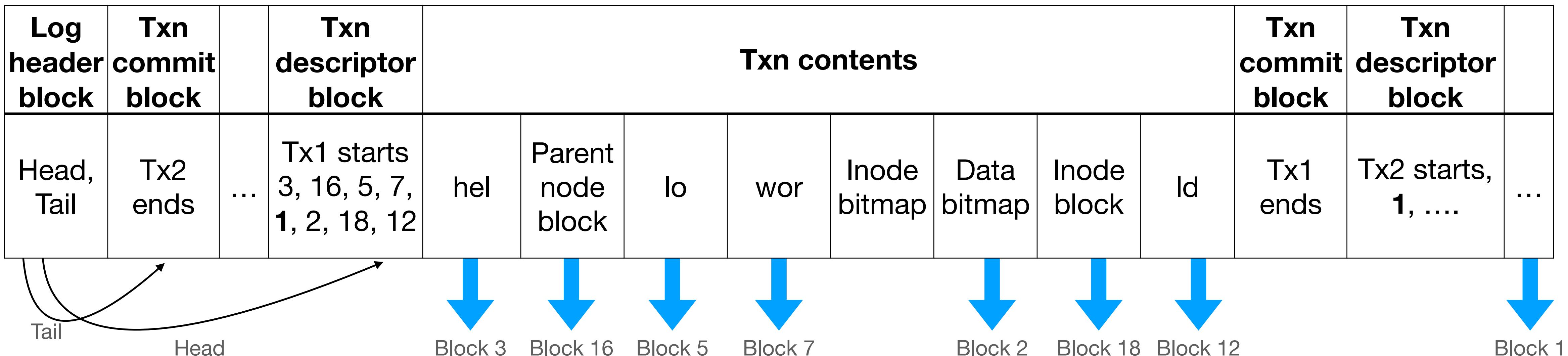
Performance

- Writing to log is one large sequential write. Durable as soon as commit block is written.
- Send bulk writes to home locations => good disk scheduling opportunity.
- All writes are off the “critical path”. New transactions are started while old one is applying writes.
- Recovery only reads the log and not the whole disk. Scales well with disk sizes.
- Need not carefully reason about ordering! Can evolve file system data structures easily!



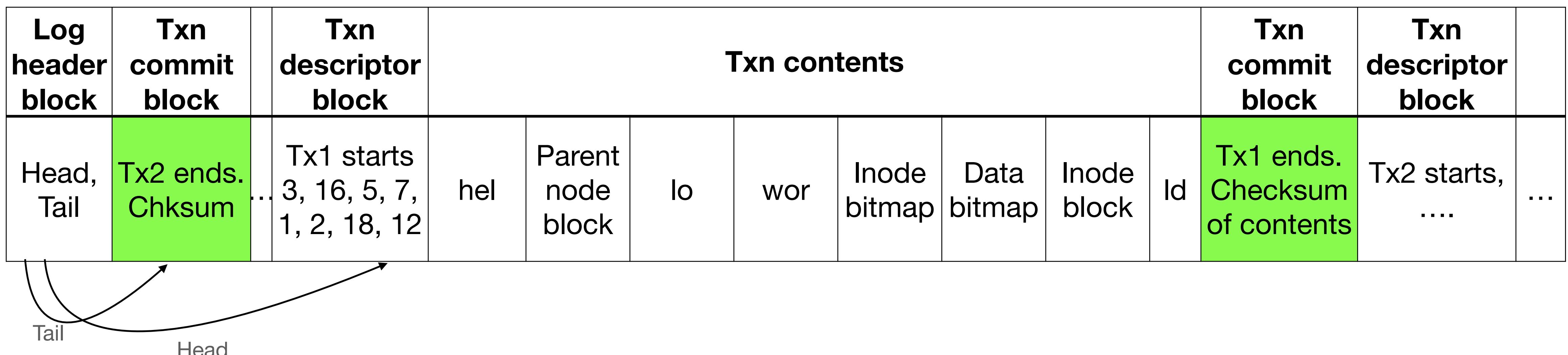
Performance

- Writing to log is one large sequential write. Durable as soon as commit block is written.
- Send bulk writes to home locations => good disk scheduling opportunity.
- All writes are off the “critical path”. New transactions are started while old one is applying writes.
- Recovery only reads the log and not the whole disk. Scales well with disk sizes.
- Need not carefully reason about ordering! Can evolve file system data structures easily!
- Writes each block twice



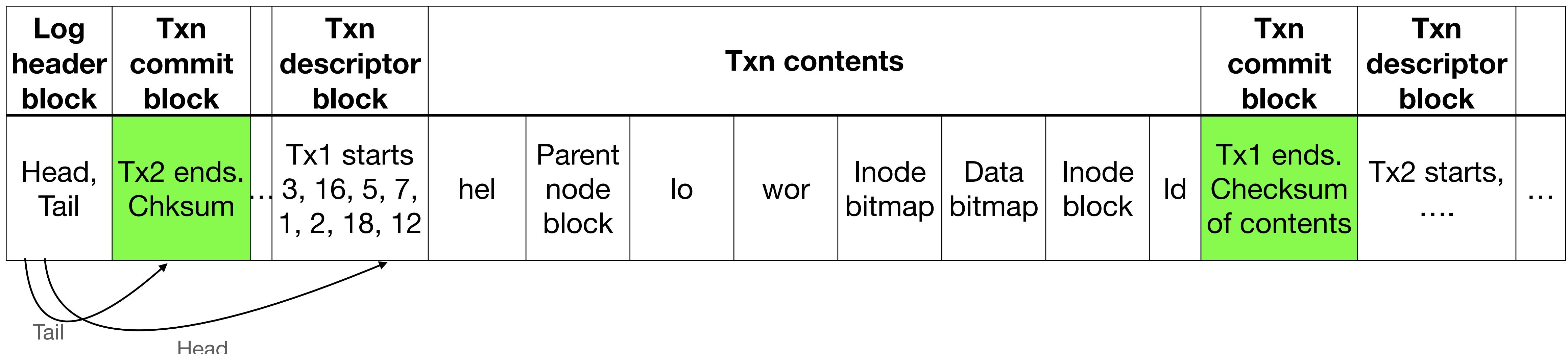
Transaction end (ext4)

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents
- Write transaction commit block
- Write transaction contents to home locations
- Update head in log header block



Transaction end (ext4)

- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents
- Durable after commit block is written. Get to commit as quickly as possible. Recovery verifies checksum before applying.
- Write transaction commit block
- Write transaction contents to home locations
- Update head in log header block



xv6 code walkthrough

p12-log

- Only single transaction in the log. Only single operation in transaction.
- file.c: begin_op, end_op
- log.c:
 - struct logheader is the “descriptor block” containing number of blocks and their home locations
 - end_op calls commit
 - commit calls
 - write_log to write all dirty blocks to log
 - write_head writes the transaction descriptor block. Same as commit block for xv6.
 - install_trans copies from log to disk
 - write_head marks transaction as applied by change number of blocks in log header to zero
- main.c calls initlog which calls recover_from_log.
 - recover_from_log does last two steps of commit

Log size constraints

Log size constraints

- Transactions cannot write more blocks than the space available in the log

Log size constraints

- Transactions cannot write more blocks than the space available in the log
 - When the log is full => File system operations need to wait for old transactions to free the log

Log size constraints

- Transactions cannot write more blocks than the space available in the log
 - When the log is full => File system operations need to wait for old transactions to free the log
- A large file write might write lots of data blocks

Log size constraints

- Transactions cannot write more blocks than the space available in the log
 - When the log is full => File system operations need to wait for old transactions to free the log
- A large file write might write lots of data blocks
 - Only partially make the write atomic

Log size constraints

- Transactions cannot write more blocks than the space available in the log
 - When the log is full => File system operations need to wait for old transactions to free the log
- A large file write might write lots of data blocks
 - Only partially make the write atomic
 - Do not write data blocks

Reducing log size

Metadata journaling

- Only write metadata to log:

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

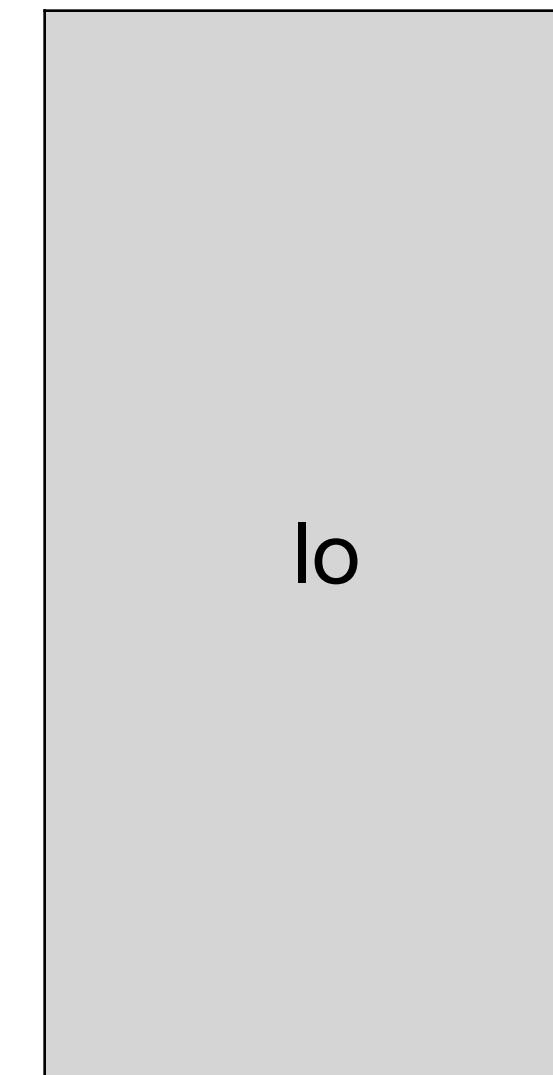
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Block 3



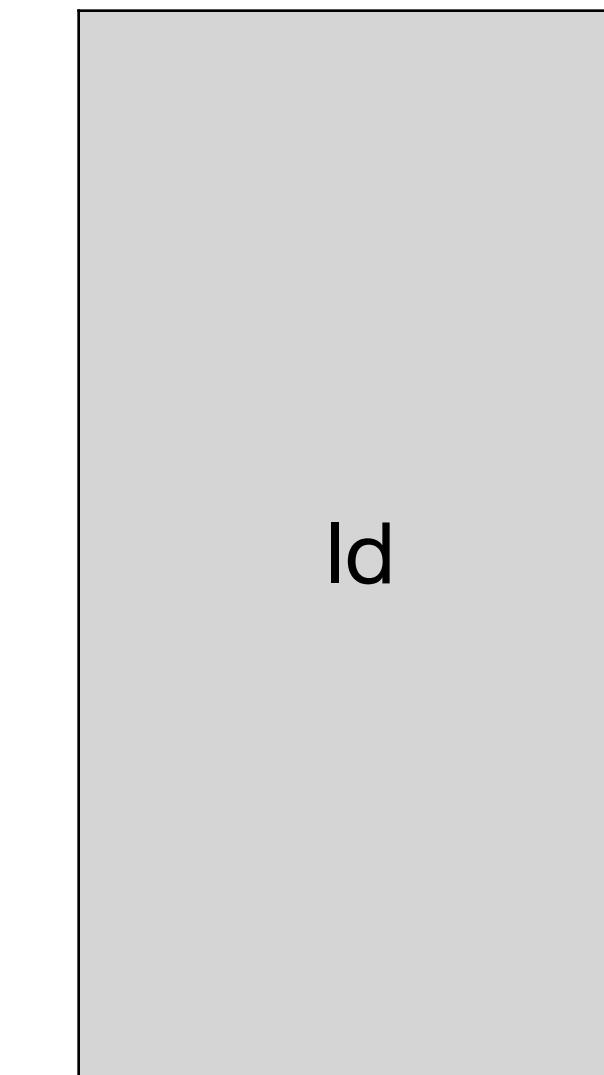
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Reducing log size

Metadata journaling

- Only write metadata to log:
 - Inode blocks

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

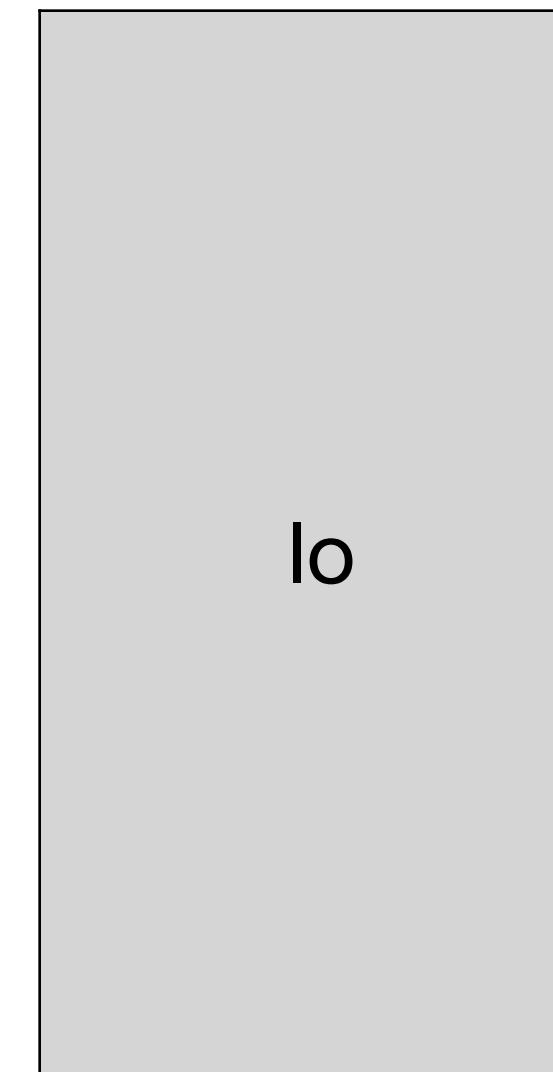
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Block 3



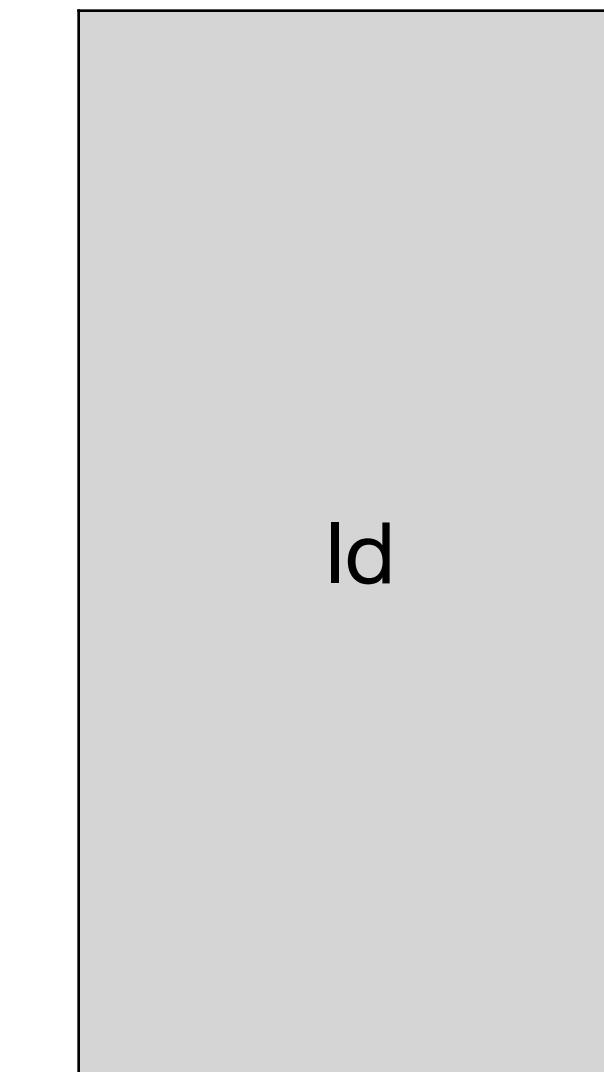
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Reducing log size

Metadata journaling

- Only write metadata to log:
 - Inode blocks
 - Data and inode bitmaps

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

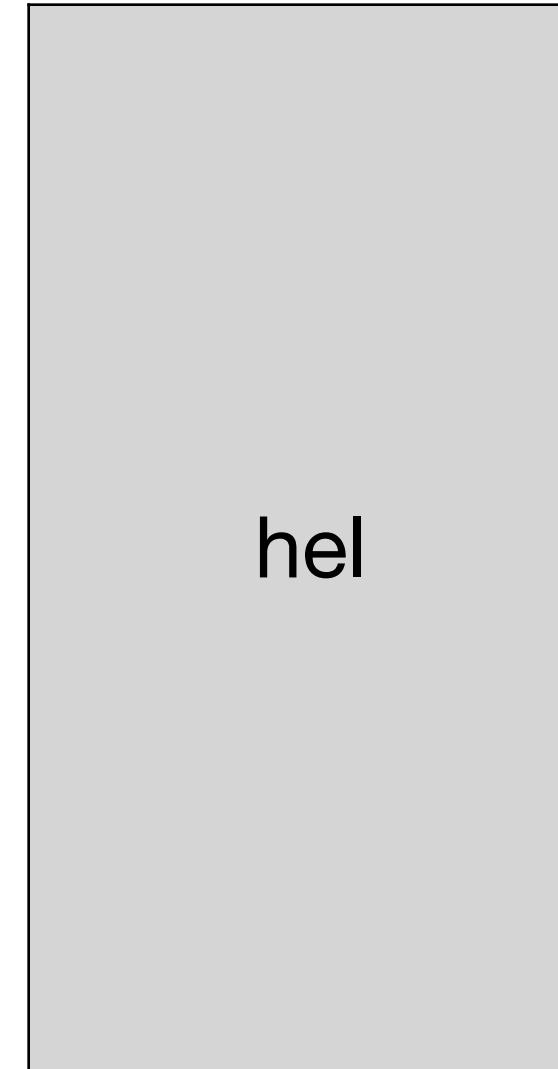
Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

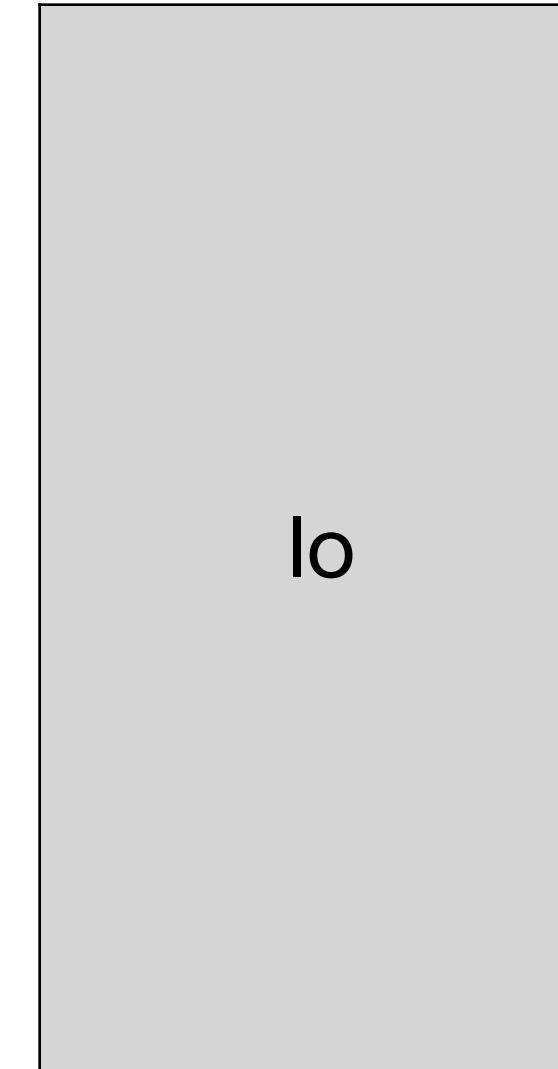
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Block 3



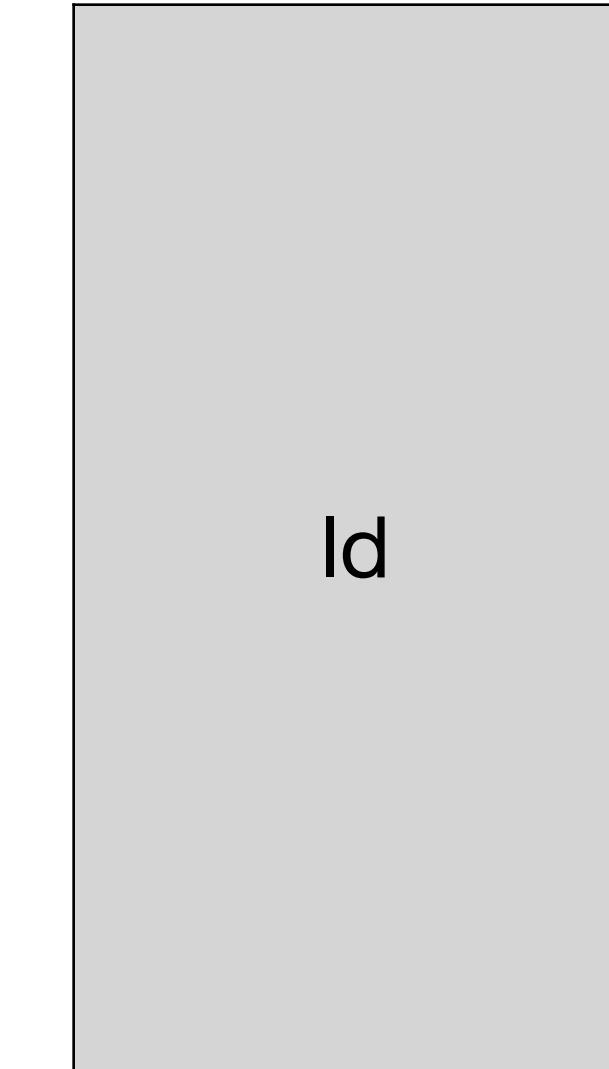
Block 5



Block 7



Block 12



Block 16

File/directory name	Inode number
.	8
..	1
bar	2

Reducing log size

Metadata journaling

- Only write metadata to log:
 - Inode blocks
 - Data and inode bitmaps
 - Directory's data blocks containing directory entries

Block 16	
File/directory name	Inode number
.	8
..	1
bar	2

Inode = 2 “/foo/bar”

Type = file
Size
Modified time
3
5
7
12

```
fd = open("/foo/bar", O_CREATE)  
write(fd, "hello world\n", 12);  
close(fd);
```

Inode bitmap

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

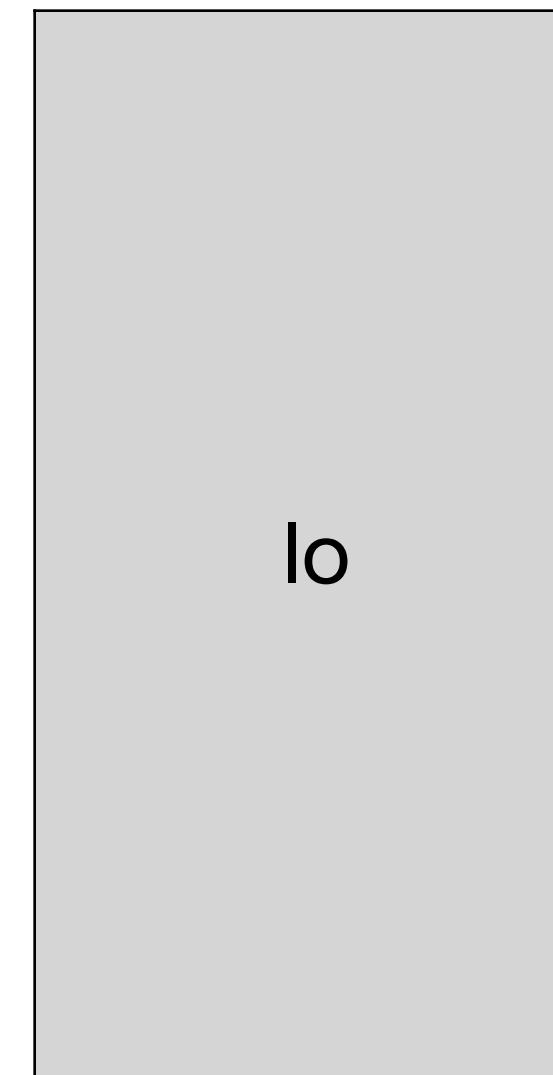
Data bitmap

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Block 3



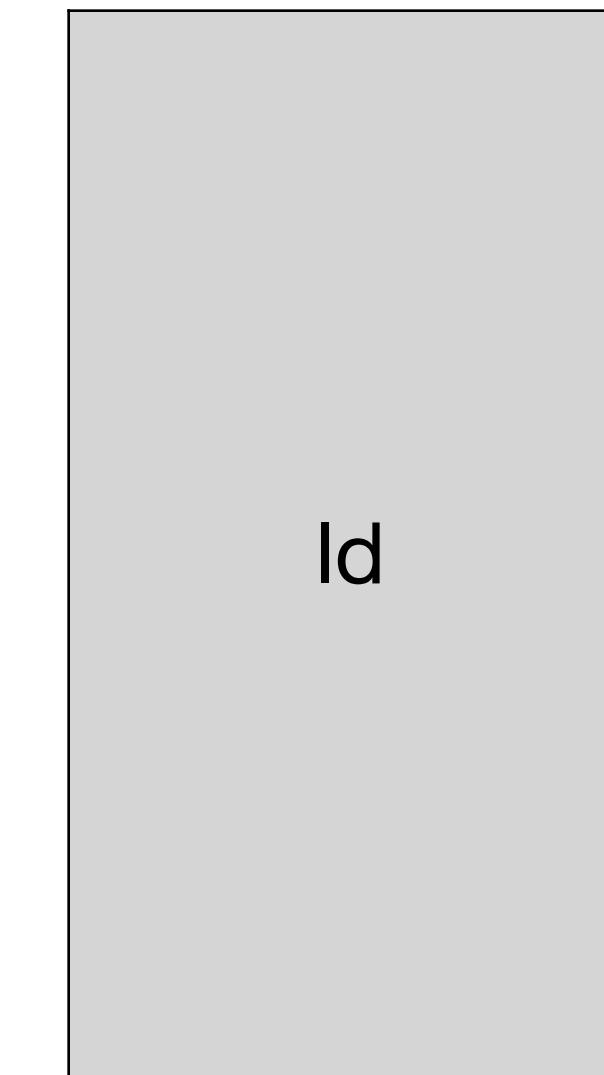
Block 5



Block 7

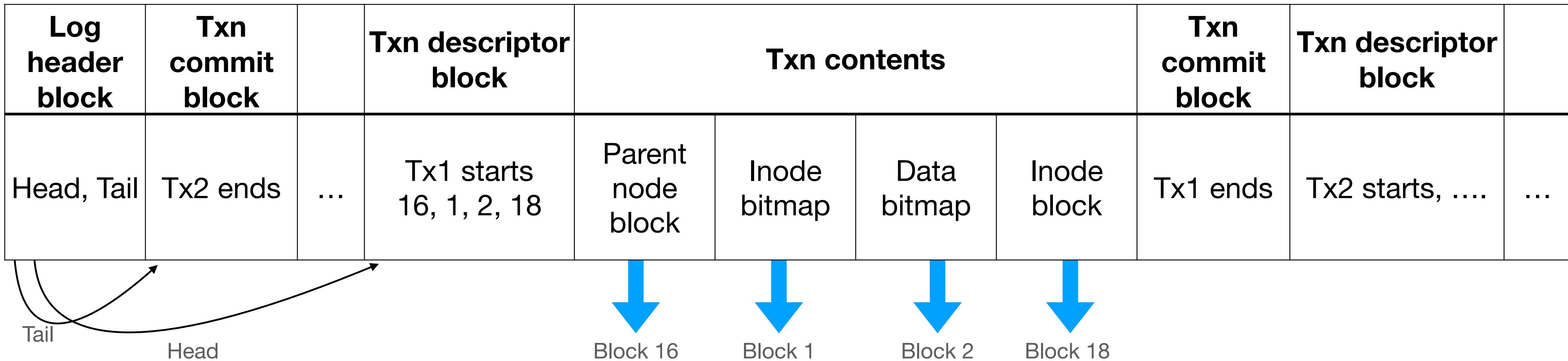


Block 12



Metadata journalling

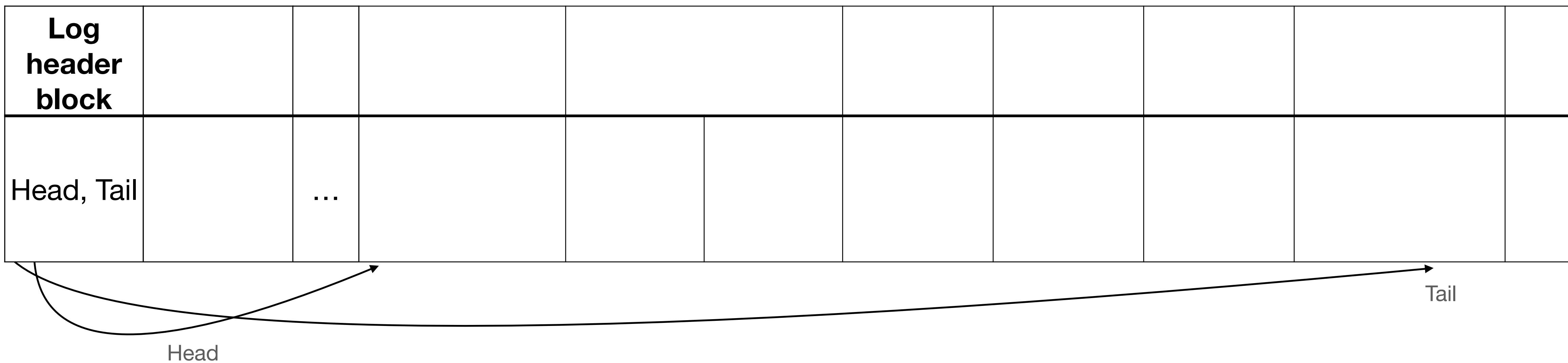
- Write data blocks
- Update tail in log header block
- Write transaction descriptor block
- Write transaction contents
- Write transaction commit block
- Write transaction contents to home locations
- Update head in log header block



Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz



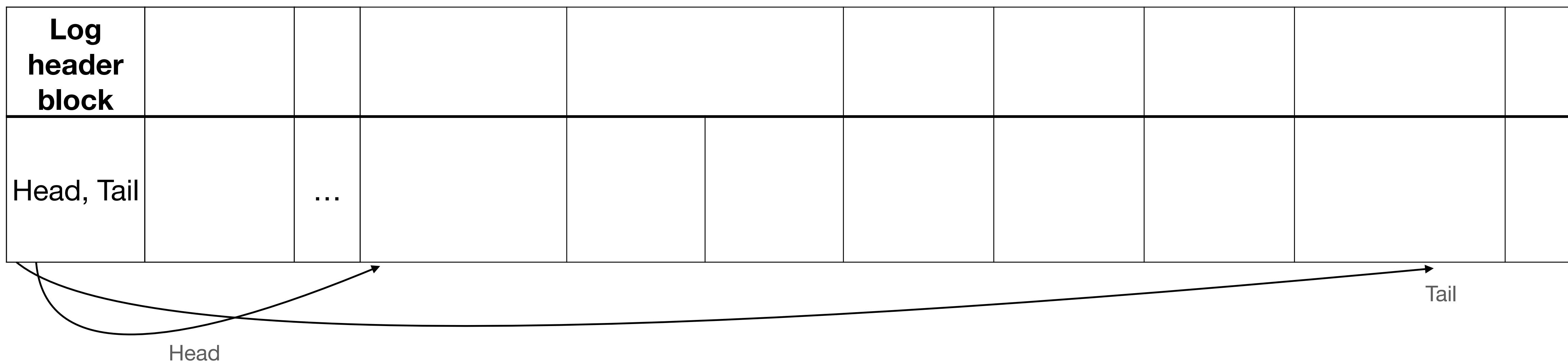
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

File/directory name	Inode number
.	8
..	1



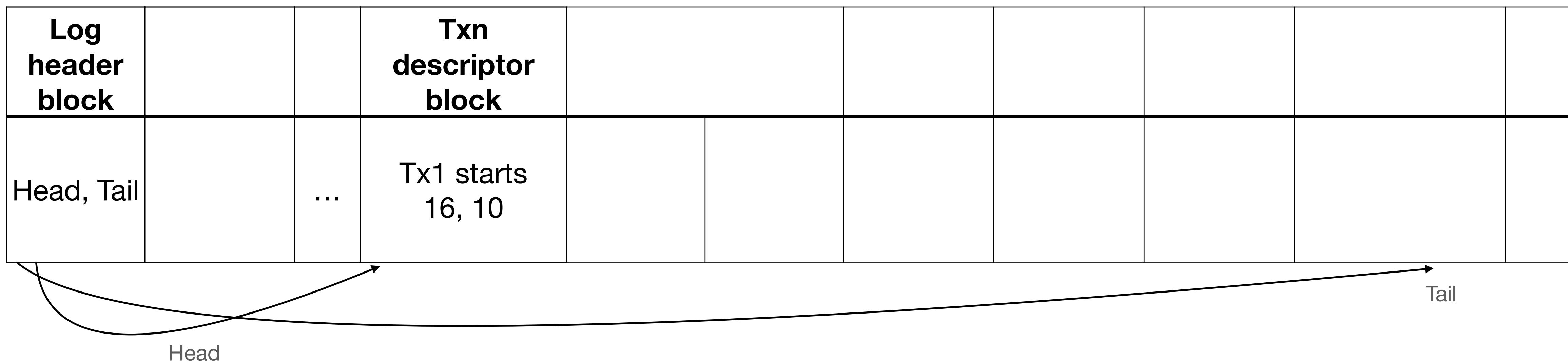
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

File/directory name	Inode number
.	8
..	1



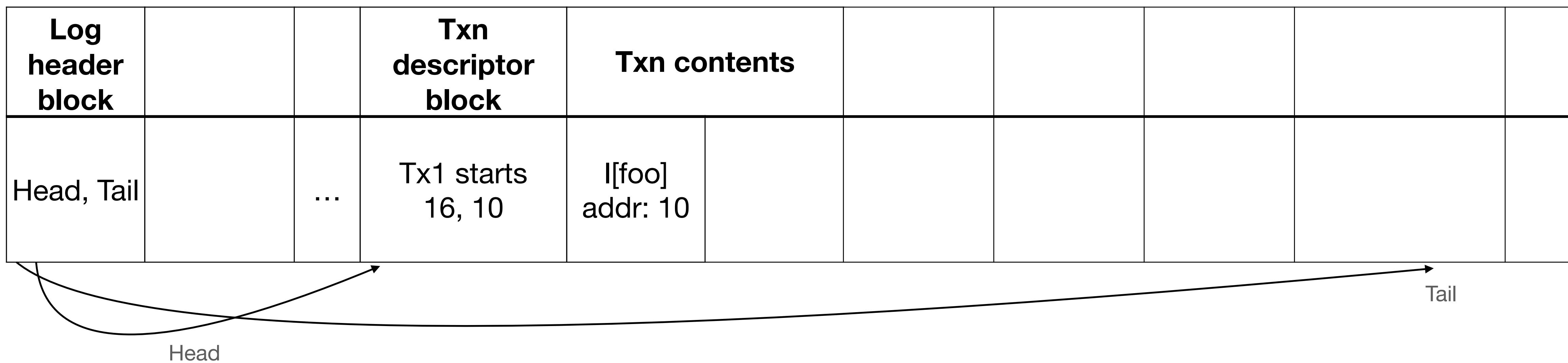
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

File/directory name	Inode number
.	8
..	1



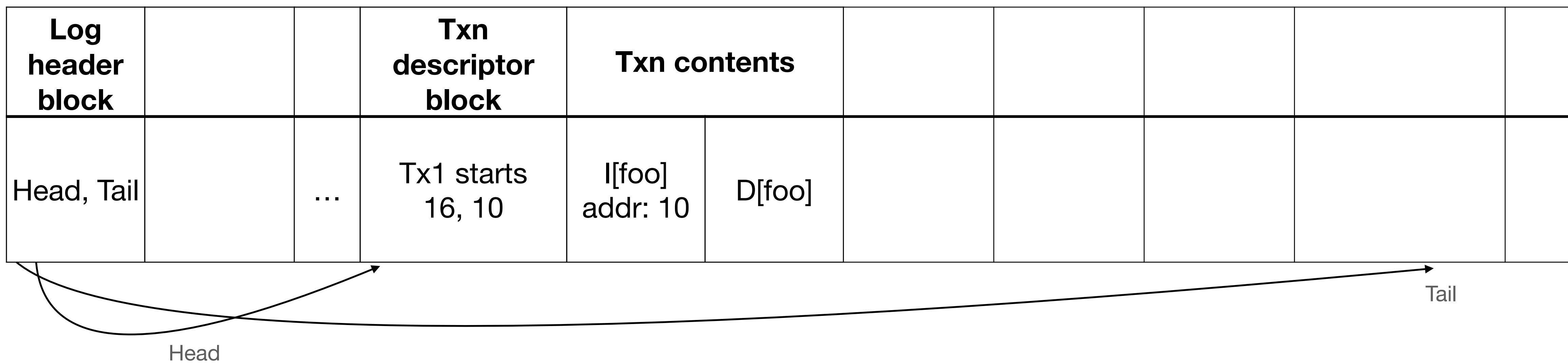
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

File/directory name	Inode number
.	8
..	1



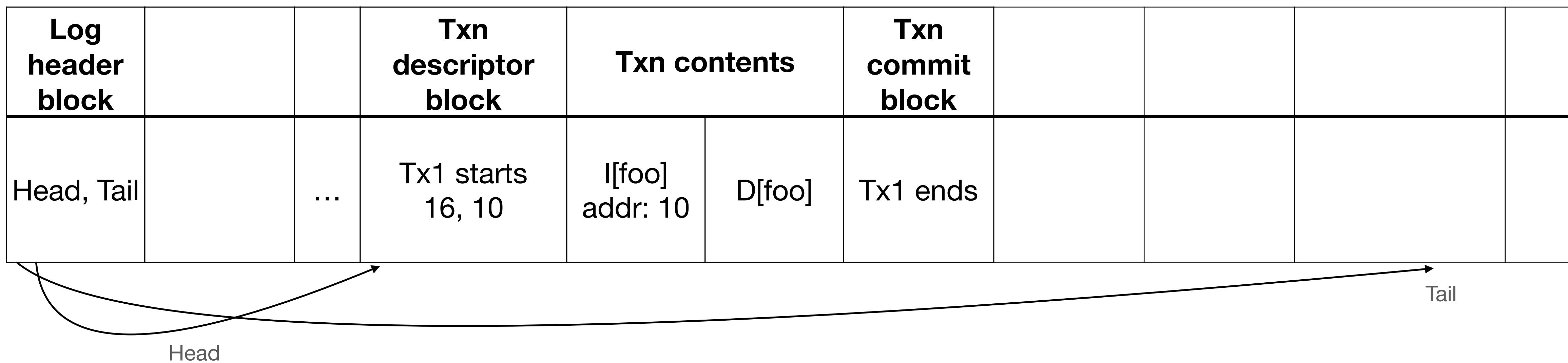
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

File/directory name	Inode number
.	8
..	1



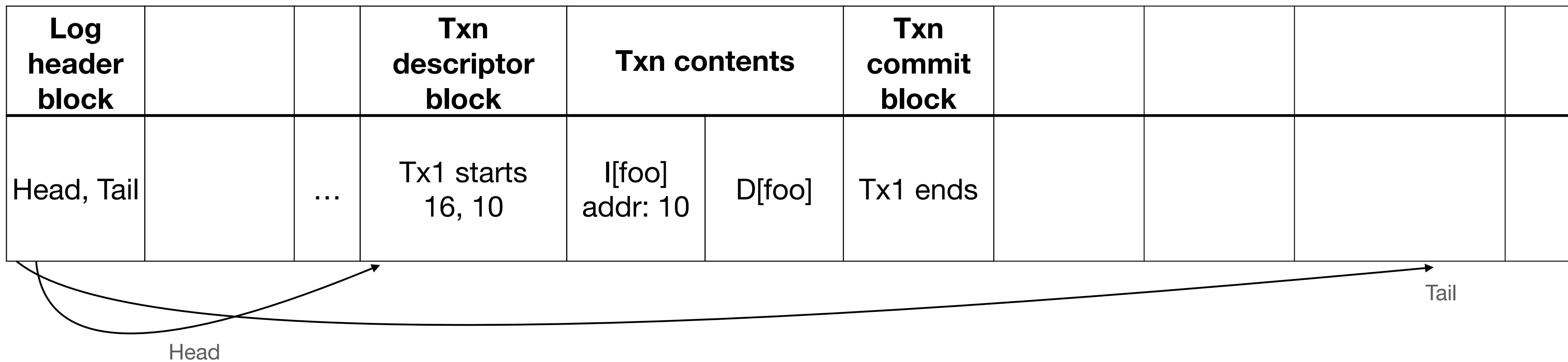
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

hello



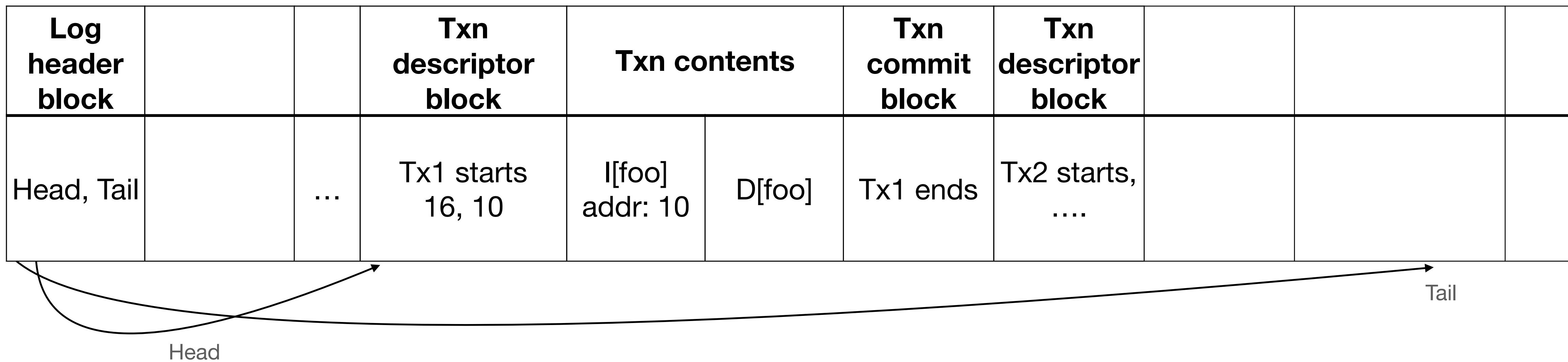
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

hello



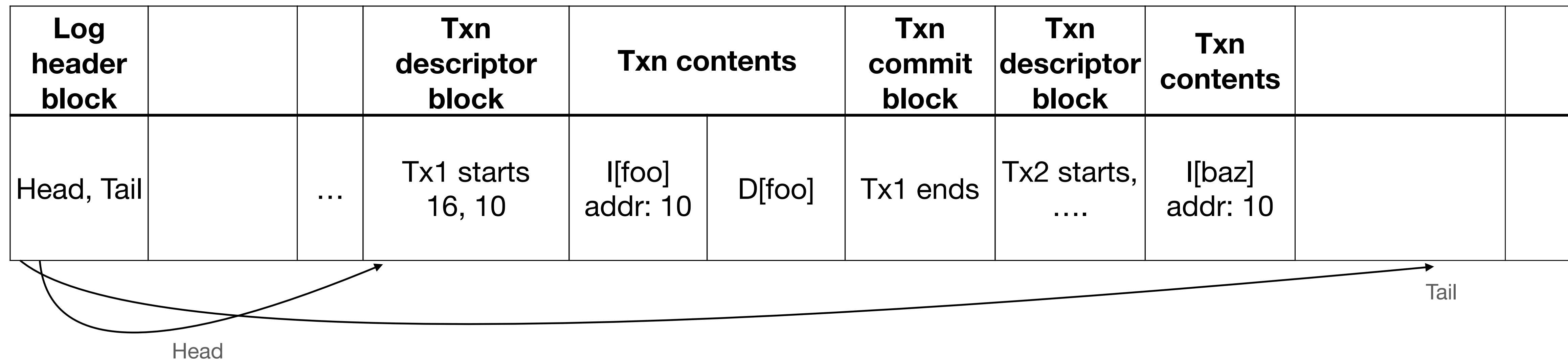
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

hello



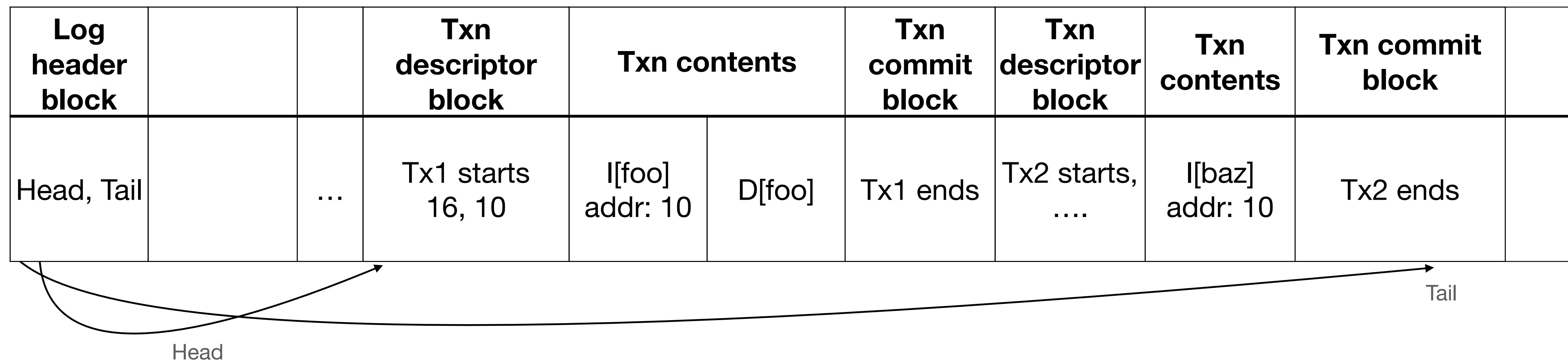
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

hello



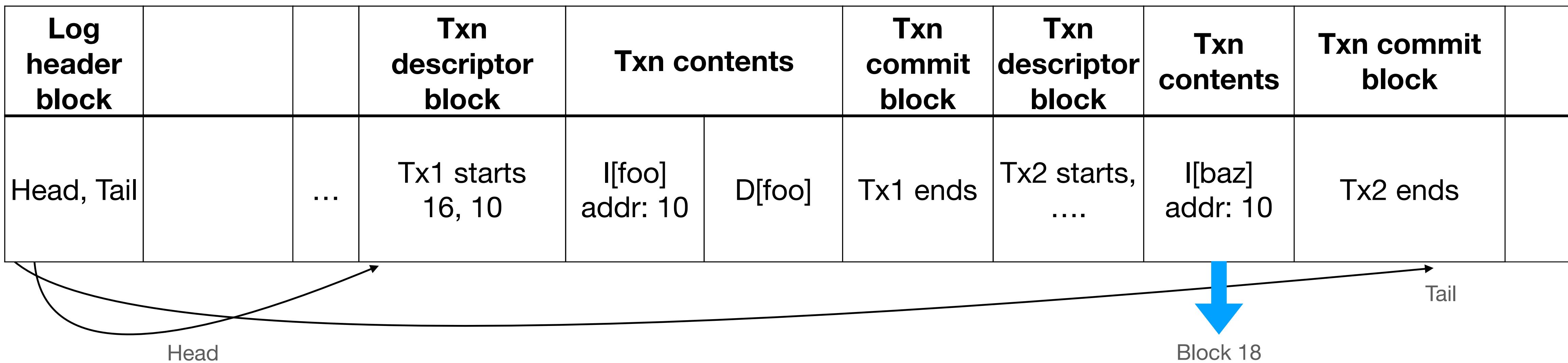
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

hello



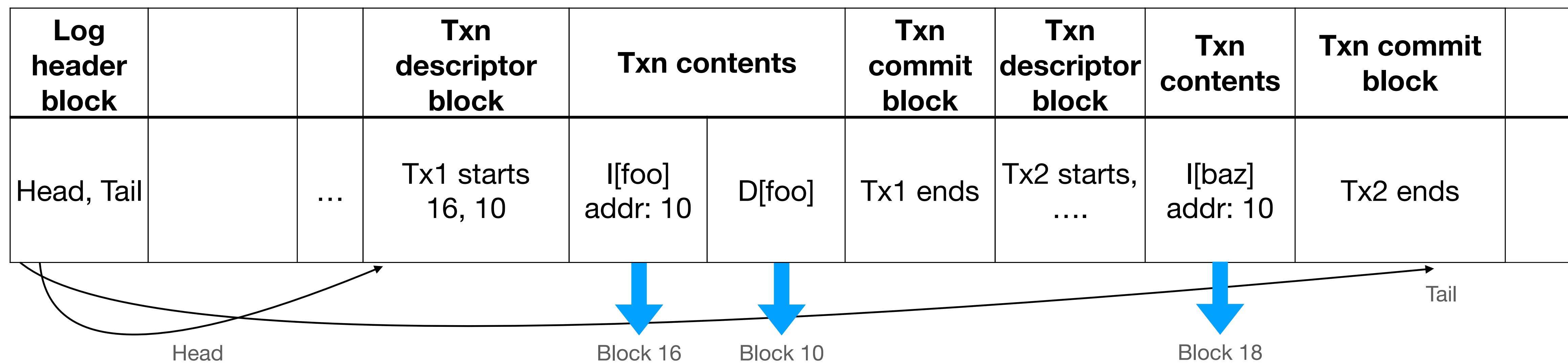
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

hello



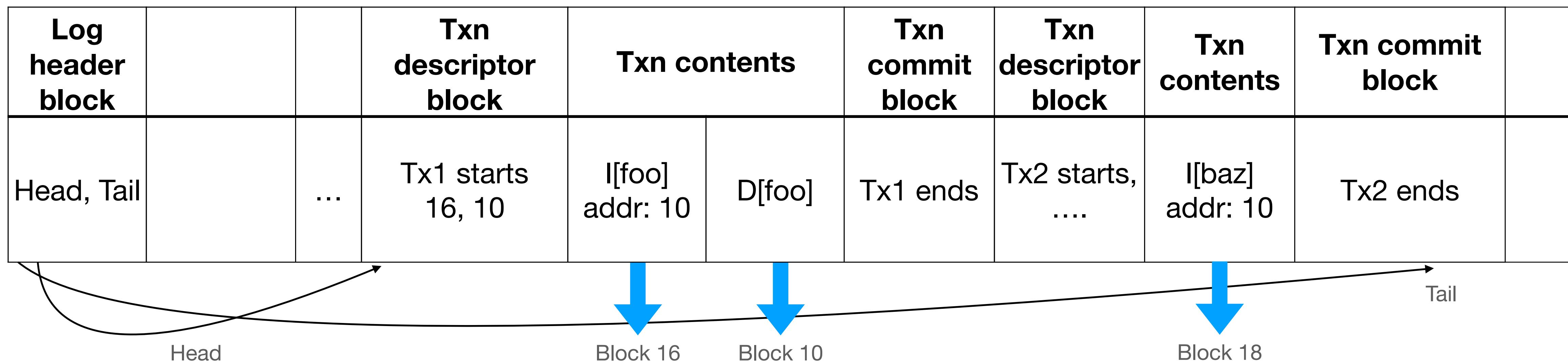
Block 10

Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

File/directory name	Inode number
.	8
..	1



Block 10

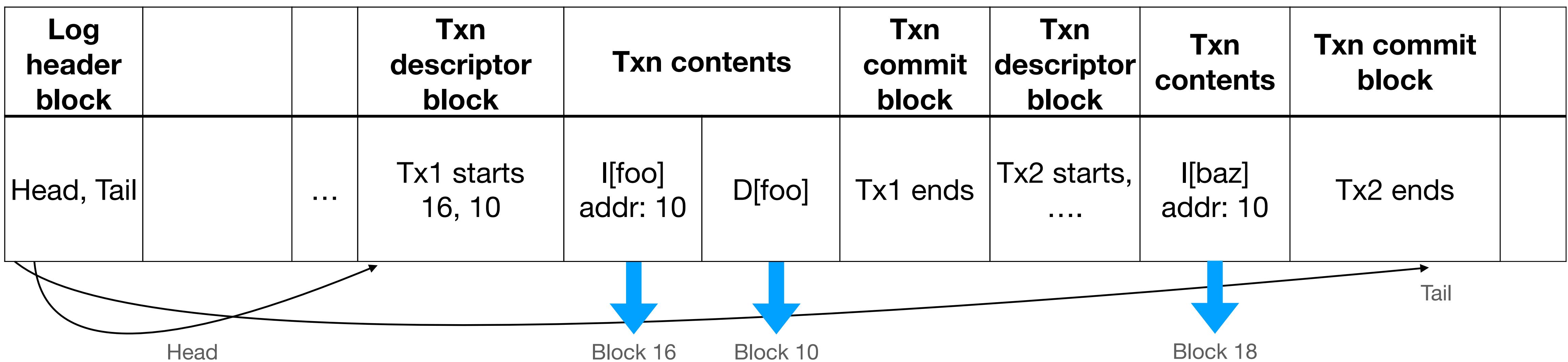
Data block reuse

T1: mkdir /foo

T2: rmdir /foo; echo hello > /baz

- Directory data block is “FS metadata” and is in the log
- File data block is not in the log

File/directory name	Inode number
.	8
..	1



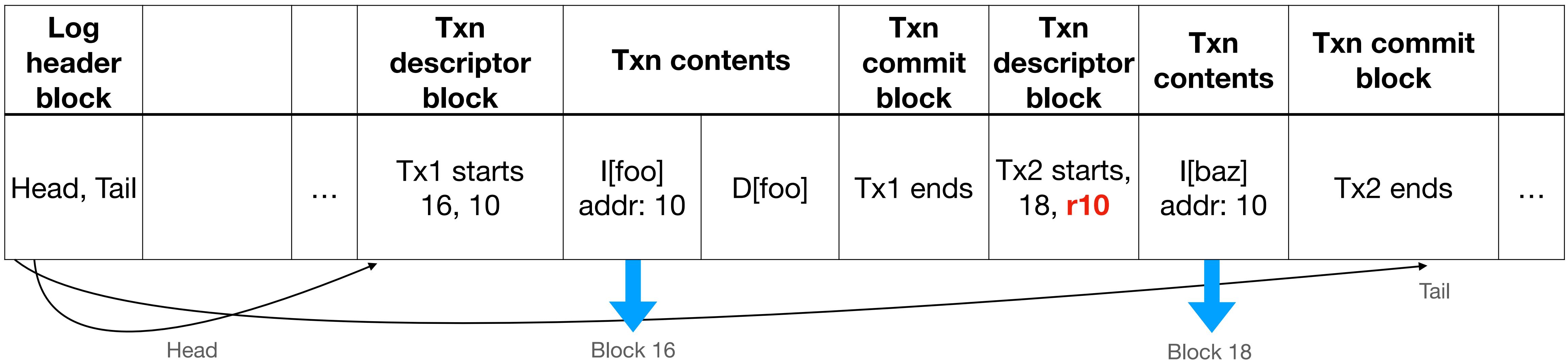
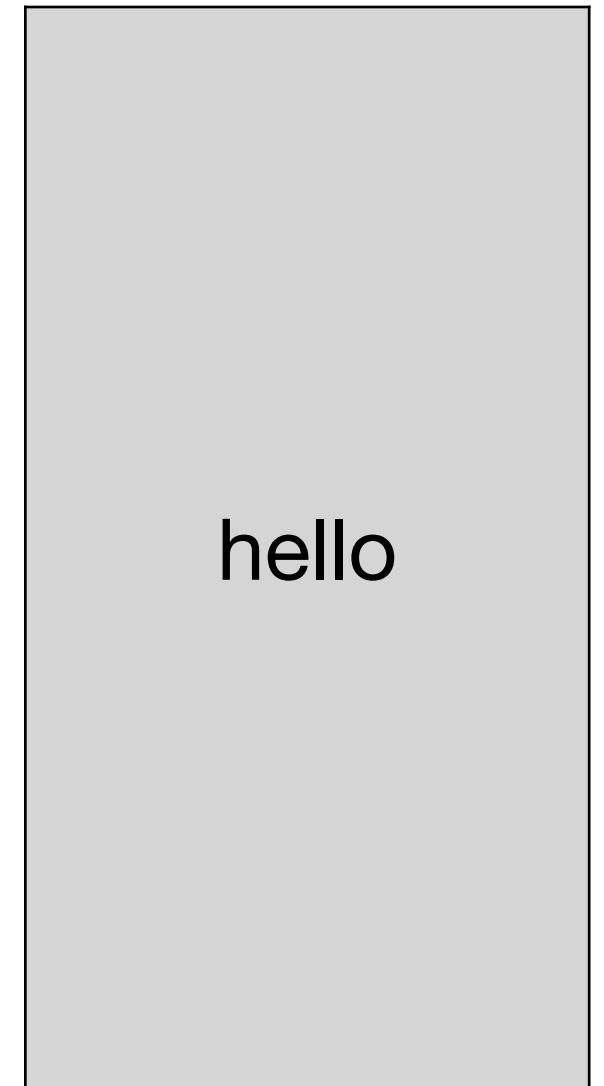
Block 10

Data block reuse

T1: mkdir /foo

T2: rm /foo; echo hello > /baz

- Directory data block is “FS metadata” and is in the log
- File data block is not in the log



Conclusion

Conclusion

- File system is an on-disk data structure to expose file and directories over disk blocks

Conclusion

- File system is an on-disk data structure to expose file and directories over disk blocks
 - Contiguous allocation, linked list, FAT. Indexed FS: Inodes, direct and indirect pointers to data blocks. Directory entries in directory data blocks.

Conclusion

- File system is an on-disk data structure to expose file and directories over disk blocks
 - Contiguous allocation, linked list, FAT. Indexed FS: Inodes, direct and indirect pointers to data blocks. Directory entries in directory data blocks.
 - Optimizations: break disk into groups (locality awareness), defragmentation, extents, disk cache, buffer cache

Conclusion

- File system is an on-disk data structure to expose file and directories over disk blocks
 - Contiguous allocation, linked list, FAT. Indexed FS: Inodes, direct and indirect pointers to data blocks. Directory entries in directory data blocks.
- Optimizations: break disk into groups (locality awareness), defragmentation, extents, disk cache, buffer cache
- Crash consistency: disk can only write one block at a time. Ensure FS is consistent across power failures/ kernel crashes. Have garbage data, loss of files, loss of security, block leaks

Conclusion

- File system is an on-disk data structure to expose file and directories over disk blocks
 - Contiguous allocation, linked list, FAT. Indexed FS: Inodes, direct and indirect pointers to data blocks. Directory entries in directory data blocks.
 - Optimizations: break disk into groups (locality awareness), defragmentation, extents, disk cache, buffer cache
 - Crash consistency: disk can only write one block at a time. Ensure FS is consistent across power failures/ kernel crashes. Have garbage data, loss of files, loss of security, block leaks
 - Ordering: No dangling pointers. Cycles force commit. Fix leaks with fsck. Does not scale to large disks.

Conclusion

- File system is an on-disk data structure to expose file and directories over disk blocks
 - Contiguous allocation, linked list, FAT. Indexed FS: Inodes, direct and indirect pointers to data blocks. Directory entries in directory data blocks.
 - Optimizations: break disk into groups (locality awareness), defragmentation, extents, disk cache, buffer cache
 - Crash consistency: disk can only write one block at a time. Ensure FS is consistent across power failures/ kernel crashes. Have garbage data, loss of files, loss of security, block leaks
 - Ordering: No dangling pointers. Cycles force commit. Fix leaks with fsck. Does not scale to large disks.
 - Write ahead logging: No dangling pointers, no leaks. Recovery will “redo log”. 5-30s durability guarantees. Overlap computation with disk writes (new Txn starts immediately). CoW for buffer conflicts.

Conclusion

- File system is an on-disk data structure to expose file and directories over disk blocks
 - Contiguous allocation, linked list, FAT. Indexed FS: Inodes, direct and indirect pointers to data blocks. Directory entries in directory data blocks.
 - Optimizations: break disk into groups (locality awareness), defragmentation, extents, disk cache, buffer cache
 - Crash consistency: disk can only write one block at a time. Ensure FS is consistent across power failures/ kernel crashes. Have garbage data, loss of files, loss of security, block leaks
 - Ordering: No dangling pointers. Cycles force commit. Fix leaks with fsck. Does not scale to large disks.
 - Write ahead logging: No dangling pointers, no leaks. Recovery will “redo log”. 5-30s durability guarantees. Overlap computation with disk writes (new Txn starts immediately). CoW for buffer conflicts.
 - Txn needs to block if log is full. Metadata WAL: data block reuse problem.

Conclusion

- File system is an on-disk data structure to expose file and directories over disk blocks
 - Contiguous allocation, linked list, FAT. Indexed FS: Inodes, direct and indirect pointers to data blocks. Directory entries in directory data blocks.
 - Optimizations: break disk into groups (locality awareness), defragmentation, extents, disk cache, buffer cache
 - Crash consistency: disk can only write one block at a time. Ensure FS is consistent across power failures/ kernel crashes. Have garbage data, loss of files, loss of security, block leaks
 - Ordering: No dangling pointers. Cycles force commit. Fix leaks with fsck. Does not scale to large disks.
 - Write ahead logging: No dangling pointers, no leaks. Recovery will “redo log”. 5-30s durability guarantees. Overlap computation with disk writes (new Txn starts immediately). CoW for buffer conflicts.
 - Txn needs to block if log is full. Metadata WAL: data block reuse problem.
 - WAL: need not carefully reason about ordering. Durable as soon as Txn commit is written to log. Log write is sequential. Writing to home locations provide good disk scheduling opportunity.