

The inode Layer II

xv6-rev7

Carmi Merimovich

CS School, Tel Aviv Academic College

January 2, 2017

(memory) inode

```
3671 #define NDIRECT 12

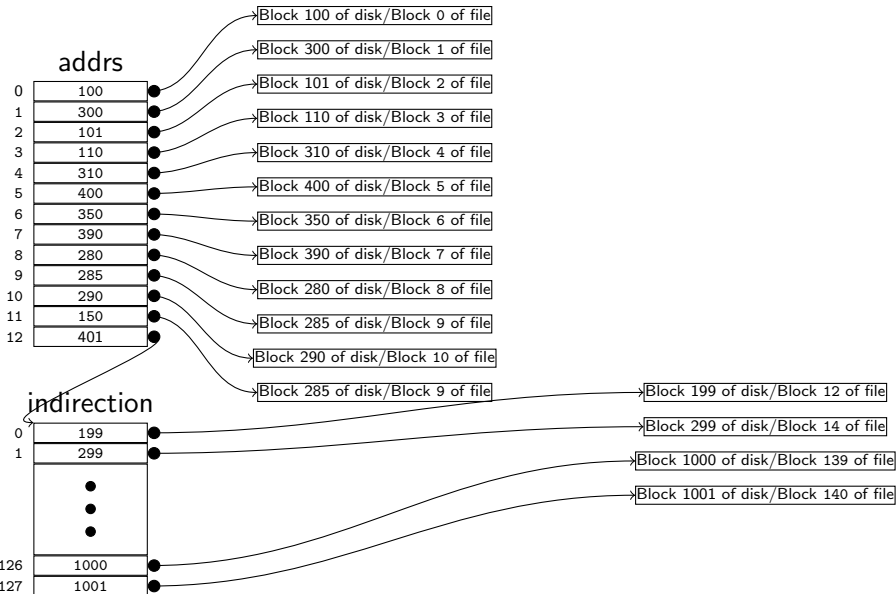
3762 struct inode {
    uint dev; // Device number
    uint inum; // Inode number
    int ref; // Reference count
    int flags; // I_BUSY, I_VALID

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

#define I_BUSY 0x1
#define I_VALID 0x2
```

We aim at readi.
Beforehand we need bmap for which we need addrs.

addrs



bmap(ip,bn)

- Return the physical block # which block #bn of the file ip uses.
- If asked for a non existent block, allocates it.
- Allocation uses the buffer layer function balloc().
- If the inode is updated it is the caller responsibility to call iupdate.

bmap() (1)

```
4810 bmap(struct inode *ip, uint bn) {  
    uint addr, *a;  
    struct buf *bp;  
  
    if (bn < NDIRECT) {  
        if ((addr = ip->addrs[bn]) == 0)  
            ip->addrs[bn] = addr = balloc(ip->dev);  
        return addr;  
    }  
}
```

bmap() (2)

```
bn -= NDIRECT;
```

```
if (bn < NINDIRECT) {  
    if ((addr = ip->addrs[NDIRECT]) == 0)  
        ip->addrs[NDIRECT] = addr = balloc(ip->dev);  
    bp = bread(ip->dev, addr);  
    a = (uint*)bp->data;  
    if ((addr = a[bn]) == 0) {  
        a[bn] = addr = balloc(ip->dev);  
        log_write(bp);  
    }  
    brelse(bp);  
    return addr;  
}  
panic("bmap: out of range");  
}
```

readi

```
4902 readi(struct inode *ip, char *dst, uint off, uint n) {
    uint tot, m;
    struct buf *bp;
    if (ip->type == T_DEV) {
        if (ip->major < 0 || ip->major >= NDEV ||
            !devsw[ip->major].read)
            return -1;
        return devsw[ip->major].read(ip, dst, n);
    }
    if (off > ip->size || off + n < off) return -1;
    if (off + n > ip->size)
        n = ip->size - off;
    for (tot=0; tot<n; tot+=m, off+=m, dst+=m) {
        bp = bread(ip->dev, bmap(ip, off/BSIZE));
        m = min(n - tot, BSIZE - off%BSIZE);
        memmove(dst, bp->data + off%BSIZE, m);
        brelse(bp);
    }
    return n;
}
```


Filling in addr

addr	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0

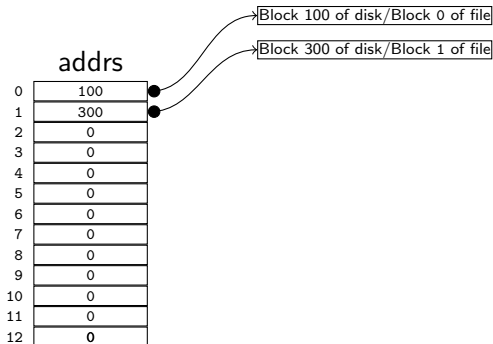
Filling in addr

Block 100 of disk/Block 0 of file

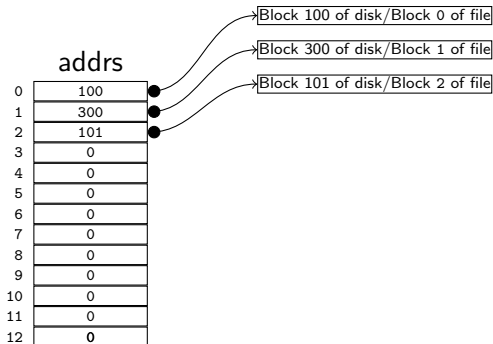
addr

0	100
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0

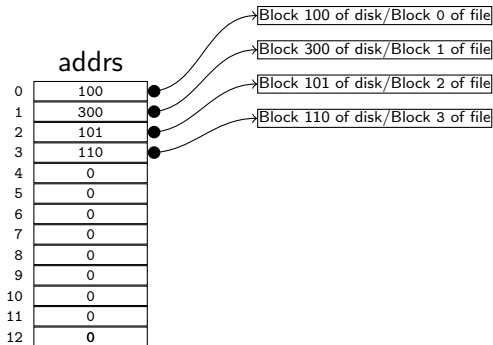
Filling in addr



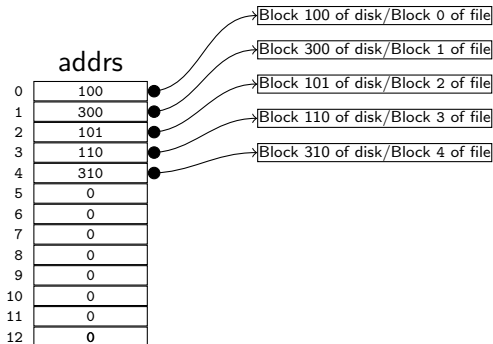
Filling in addr



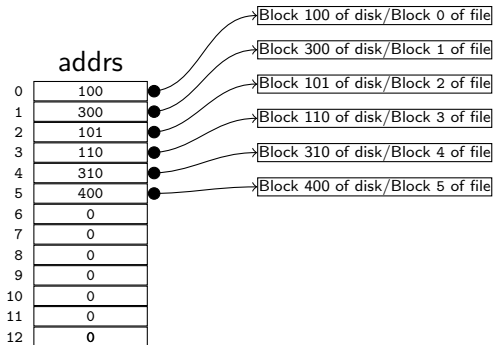
Filling in addr



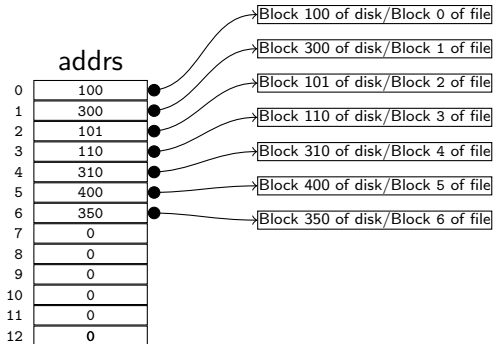
Filling in addr



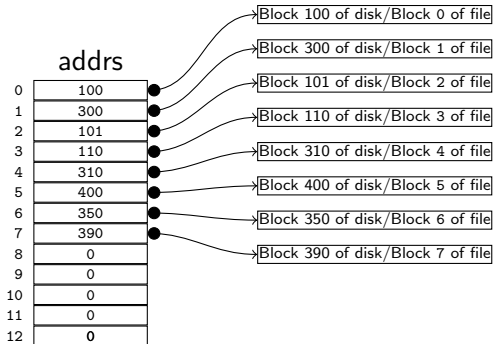
Filling in addr



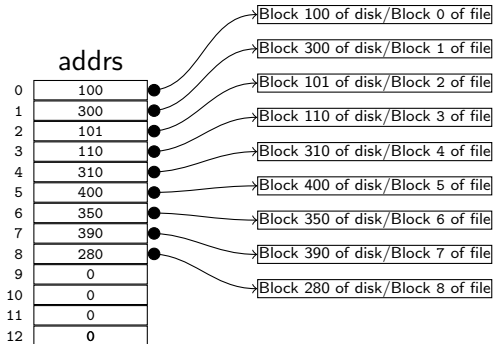
Filling in addr



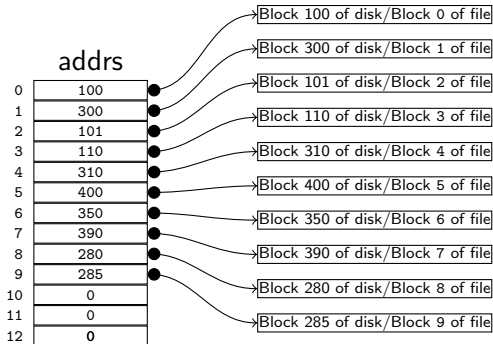
Filling in addr



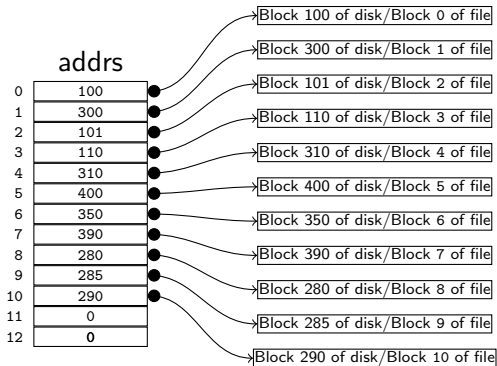
Filling in addr



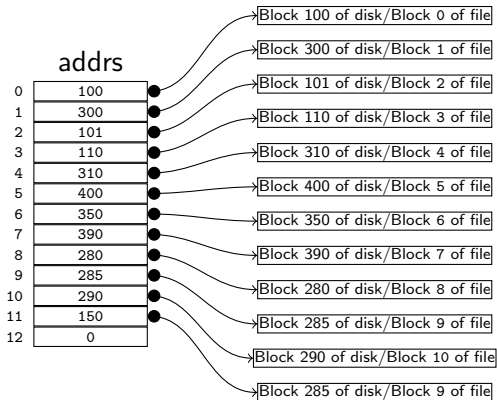
Filling in addr



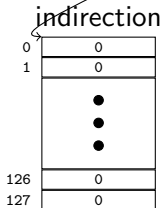
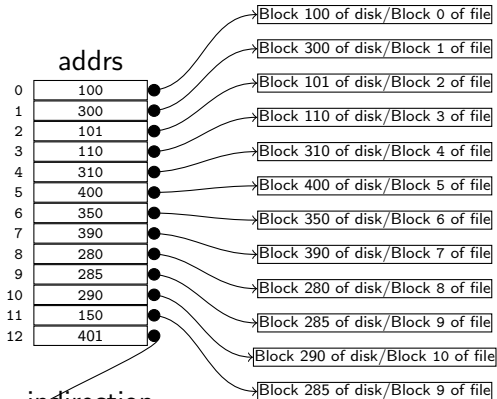
Filling in addr



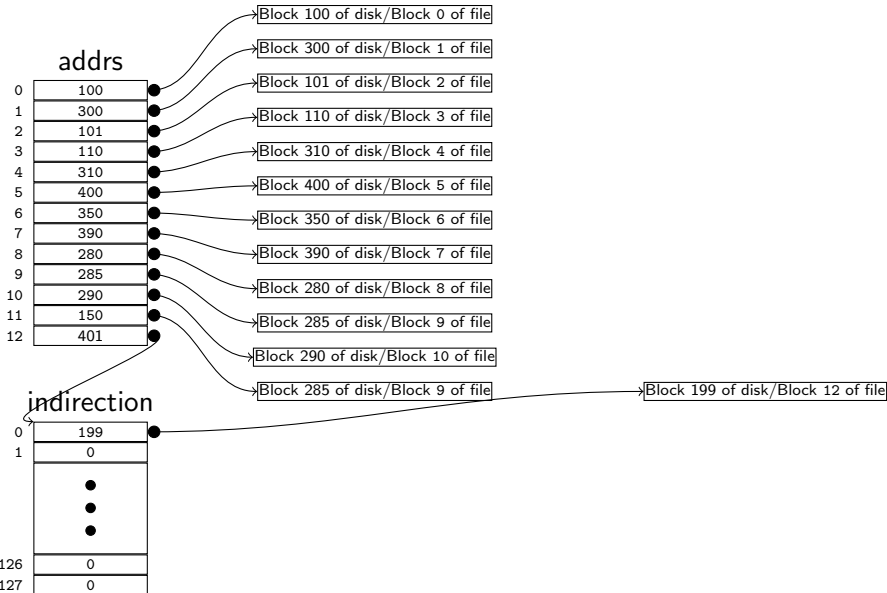
Filling in addr



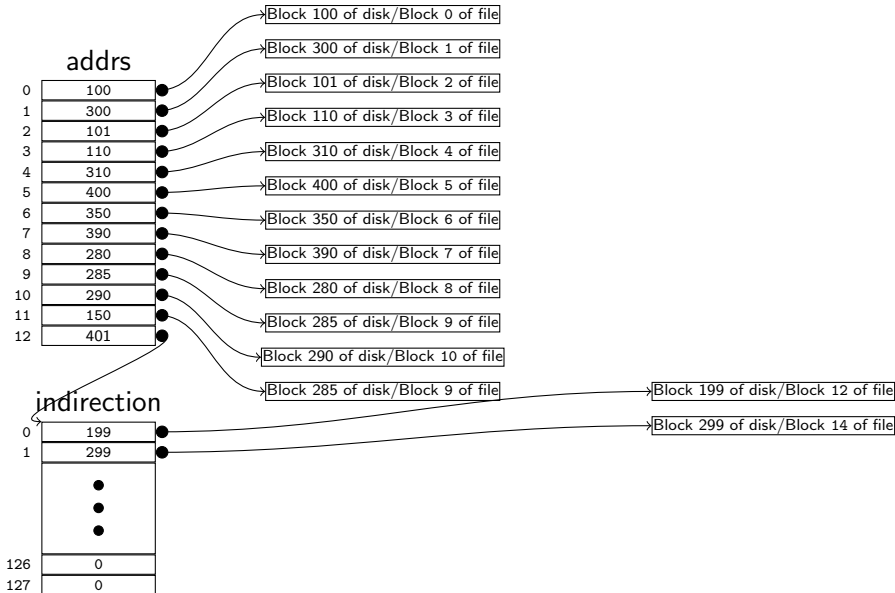
Filling in addr



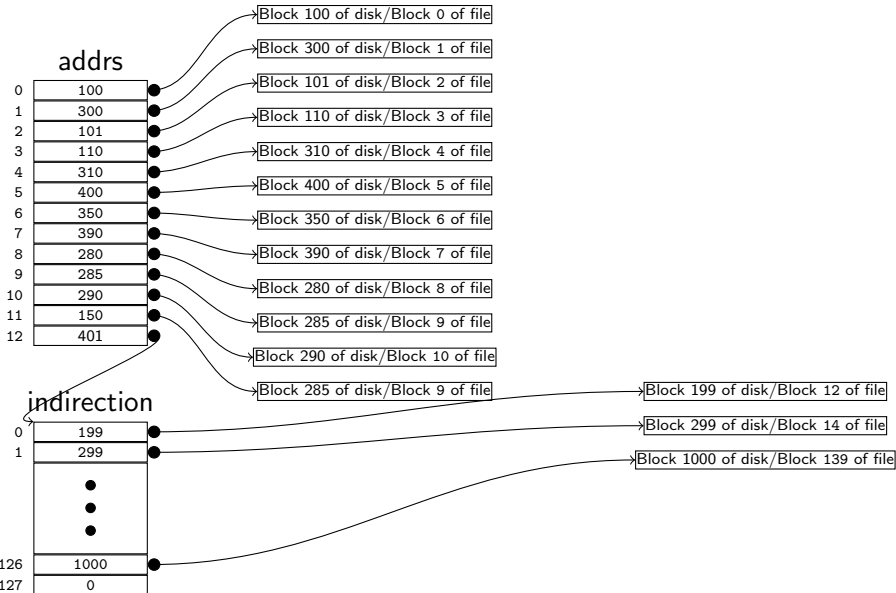
Filling in addr



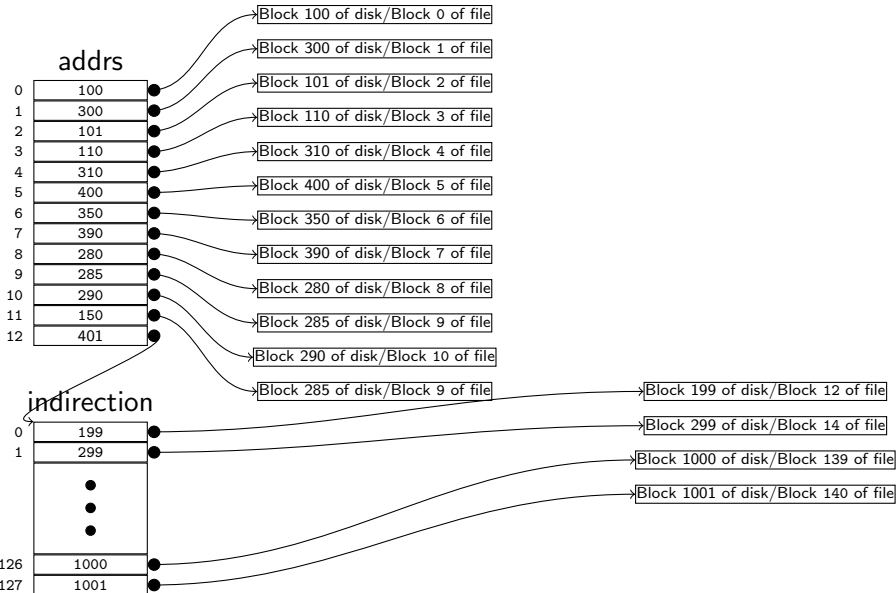
Filling in addr



Filling in addr



Filling in addr



writei (1)

```
4950 int writei(struct inode *ip, char*src, uint off, uint n,
          uint tot, m;
          struct buf *bp;

if (ip->type == T_DEV) {
    if (ip->major < 0 || ip->major >= NDEV || !devsw[ip->major])
        return -1;
    return devsw[ip->major].write(ip, src, n);
}

if (off > ip->size || off + n < off) return -1;
if (off + n > MAXFILE*BSIZE) return -1;
```

writei (2)

```
for (tot=0; tot<n; tot+=m, off+=m, src+=m) {  
    bp = bread(ip->dev, bmap(ip, off/BSIZE));  
    m = min(n - tot, BSIZE - off%BSIZE);  
    memmove(bp->data + off%BSIZE, src, m);  
    log_write(bp);  
    brelse(bp);  
}  
  
if (n > 0 && off > ip->size) {  
    ip->size = off;  
    iupdate(ip);  
}  
return n;  
}
```

itrunc (1)

```
4855 static void itrunc(struct inode *ip) {  
    int i, j;  
    struct buf *bp;  
    uint *a;  
  
    for (i = 0; i < NDIRECT; i++){  
        if (ip->addrs[i]){  
            bfree(ip->dev, ip->addrs[i]);  
            ip->addrs[i] = 0;  
        }  
    }  
}
```

itrunc (2)

```
if (ip->addrs[NDIRECT]) {
    bp = bread(ip->dev, ip->addrs[NDIRECT]);
    a = (uint*)bp->data;
    for(j = 0; j < NINDIRECT; j++) {
        if(a[j])
            bfree(ip->dev, a[j]);
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT]);
    ip->addrs[NDIRECT] = 0;
}

ip->size = 0;
iupdate(ip);
}
```

XV6 on disk structure: The big picture

