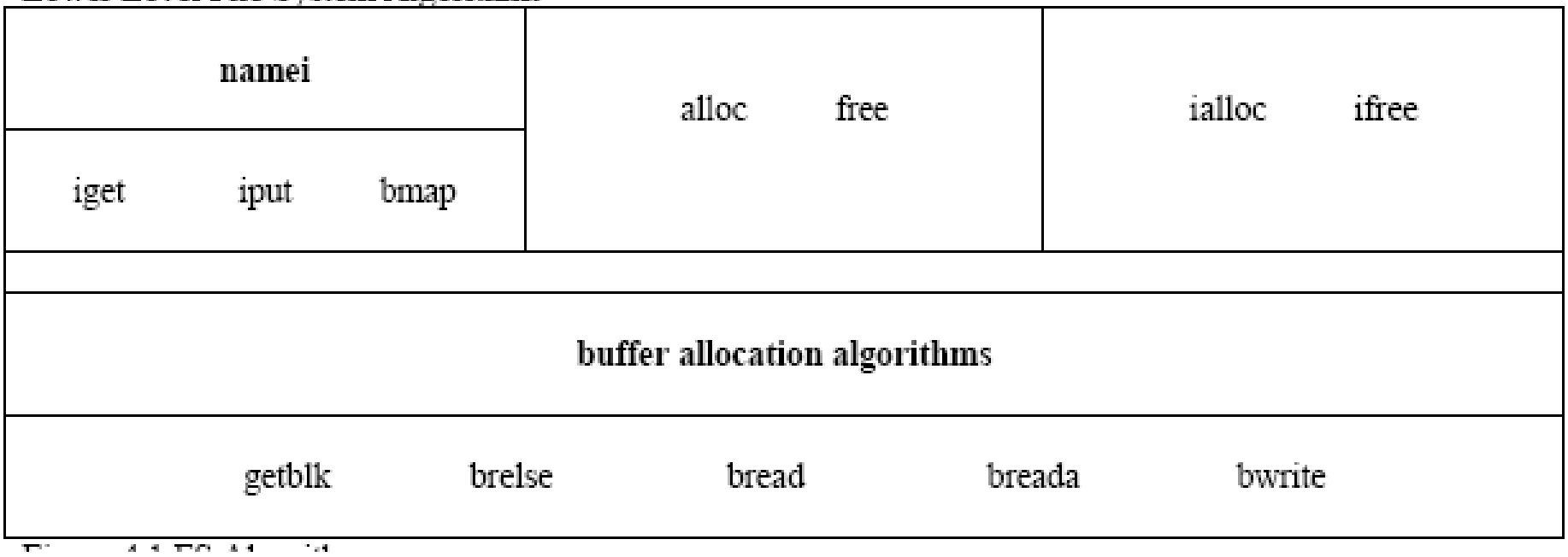
Lesson IV: Internal representation of files

- Svaka datoteka na UNIX sistemu ima **jedinstveni inode**,
- **inode** sadrži **sve informacije** potrebne da **odrede prava pristupa datoteci**, kao što su **vlasništvo, prava pristupa, veličina datoteke, lokacija datoteke** na disku.
- **Procesi pristupaju datoteci** preko jasno definisanog skupa **sistemskih poziva**.
- **Algoritmi** koje obrađujemo se nalaze **na nivou iznad baferskog keša**.
- To su:
  - **iget:** vraća vrednost prethodno identifikovanog inoda, (moguće je čitanje iz inode tabele preko baferskog keša)
  - **iput:** otpušta inode
  - **bmap:** postavlja kernelske parametre za pristupanje datoteci
  - **namei: konvertuje ime datoteke** u **inode** koristeći algoritme iget, iput i bmap.
  - **alloc** i **free** alociraju i oslobađaju slobodne disk blokove za datoteku
  - **ialloc** i **ifree** alociraju i oslobađaju slobodne inode-ove za datoteke

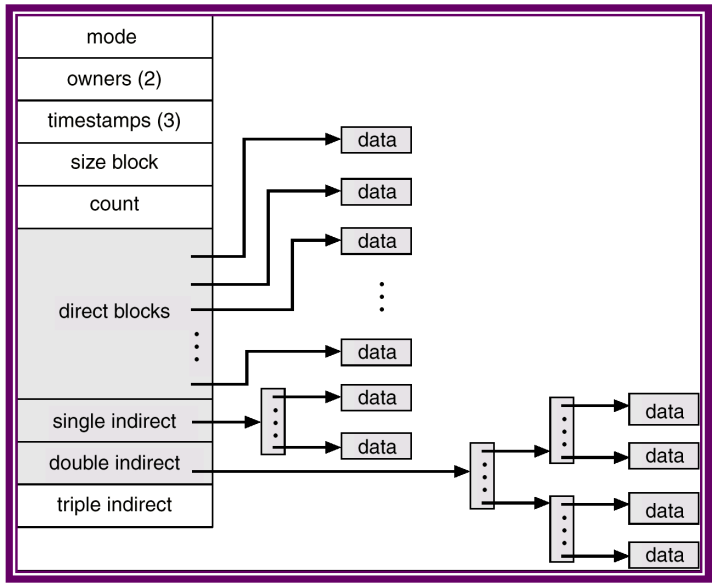
FS Algorithms

Lower Level File System Algorithms



INODES

- **Definicija:**
  - struktura koja potpuno opisuje datoteku
  - sve osim imena
- Postoje 2 strukture:
- **disk inode**
  - to su inodes u statičkoj formi na disku (in inode table)
  - pasive files
- **in-core inode**.
  - disk inode kernel čita u memorijsku strukturu koja se naziva nazvati in-core inode
  - active files



disk-inode

- **Disk inode** se sastoji od **sledećih polja**:
  - **vlasništvo** se deli na 2 komponente, na korisničko vlasništvo i grupno vlasništvo i ta dva identifikatora određuju vlasničke odnose grupe prema datoteci
    - **vlasnik datoteke** (**UID identifikator**).
    - **grupa** kojoj pripada datoteka (**GID identifikator**)
  - **tip datoteke**: datoteka može biti regularna, direktorijum, karakter ili blok specijalna datoteka, FIFO (pipe)
  - **prava pristupa za datoteku**: definišu se preko
    - 3 vlasničke kategorije (**owner, group, other**)
    - 3 prava pristupa za njih **(read, write, execute**) koja imaju precizno značenje, a nezavisno se deklarišu
  - **vremena pristupa datoteke**: ima 3 karakteristična vremena (vreme poslednje modifikacije, vreme poslednjeg pristupa, i vreme kad je inode poslednji put modifikovan)
  - **broj linkova na datoteku**: predstavlja broj različitih imena za isti prostor na disku
  - **tabela koja opisuje alokaciju** datoteke **na disku**
  - **veličina** datoteke
- Inode ne opisuje path imena već svaka path komponenta ima poseban inode.

Disk inode example

- **U ovom primeru** imamo regularnu datoteku veličine 6030 bajtova čiji je vlasnik user mjb sa pravima pristupa rwx, datoteka pripada grupi os i svi članovi grupe imaju r i x pravo bez w prava, dok svi ostali useri imaju r i x pravo bez w prava. Poslednji put je neko pristupao i to samo čitao datoteku 23. oktobra 2004 u 1:45 PM, a posledni put je neko upisao nešto u datoteku 22. oktobra 2004 u 10:30 AM. Inode je promenjen zadnji put 23. oktobra u 1:30 PM.
- **Postoji razlika** između **upisa u datoteku** i **upisa u inode**, **svaka promena** u **datoteci se reflektuje u inode**, dok postoje promene koje idu samo u inode a nemaju veze sa upisom datoteku, kao što je promena vlasništva, grupe, prava pristupa ili link setovanje.

owner mjb
group os
type regular file
perms rwxr-xr-x
accessed Oct 23 2004 1:45 P.M.
modified Oct 22 2004 10:30 A.M.
inode Oct 23 2004 1:30 P.M.
size 6030 bytes
disk adresses

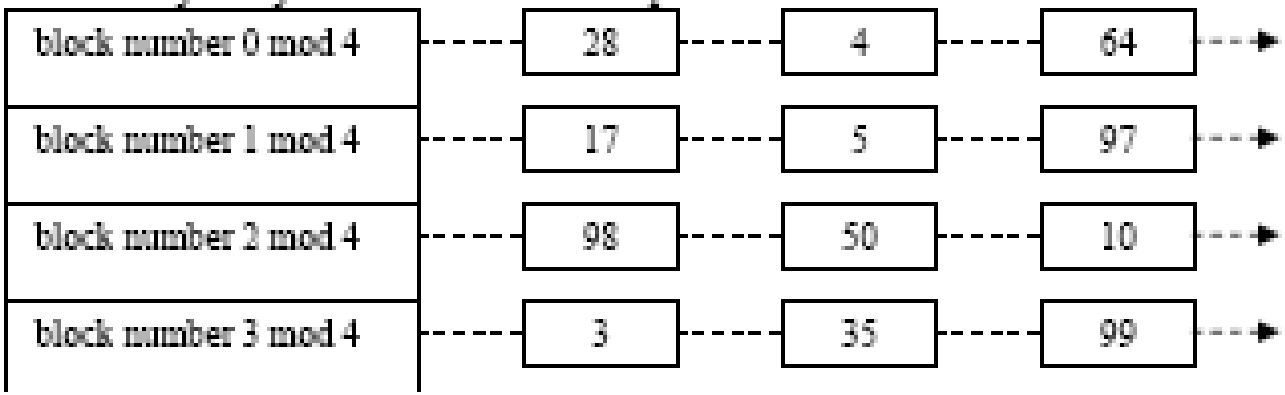
rwx datoteku veličine 6030 bajtova čiji je vl

in-core inode

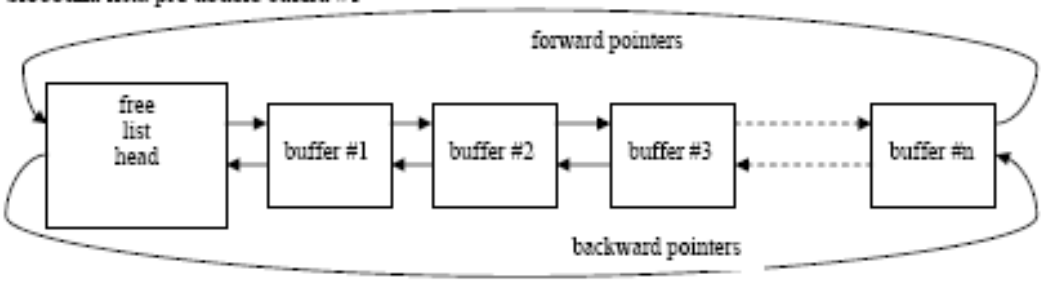
- **in-core inode** sadrži dodatne informacije u odnosu na **polja disk inode:**
- **Status in-core inoda** koji pokazuje:
  - da li inode zaključan (**locked**)
  - da li proces čeka da inode postane otključan (**unlocked**)
  - da li se **in-core inode razlikuje od svoje disk kopije** kao rezultat **promene polja u inode**
  - da li se **in-core inode** razlikuje od svoje disk kopije **kao rezultat promene** podataka **u datoteci**
  - da li je ta datoteka postala **mountpoint**
- **FS descriptor**(opis FS): u vidu logičkog broja koji sadrži tu datoteku, odnosno iz kog FS je taj inode
- **inode broj: in-core** inode poseduje i ovo polje zato što **na disku se pozicija inode** određuje u polju **fiksnog formata**, zna se offset, a **u memoriji polje nije fiksno** i mora da se zna koji je inode u **in-core tabeli**
- **ukazivači** na druge **in-core inodove**. Kernel povezuje **in-core inode-ove** u **hash queue liste** i **slobodne liste** na sličan način kao kod baferskog keširanja. **Hash queue se idetifikuje** n**a osnovu FS** i **inoda broja**. Kernel sadrži najviše jednu kopiju disk inoda, a ona može biti ili u hash queue ili u slobodnoj listi.
- **broj referenci**, pokazuje broj aktivnih instanci na tu datoteku, broj procesa koji su otvorili datoteku, odnosno taj inode

inode cache

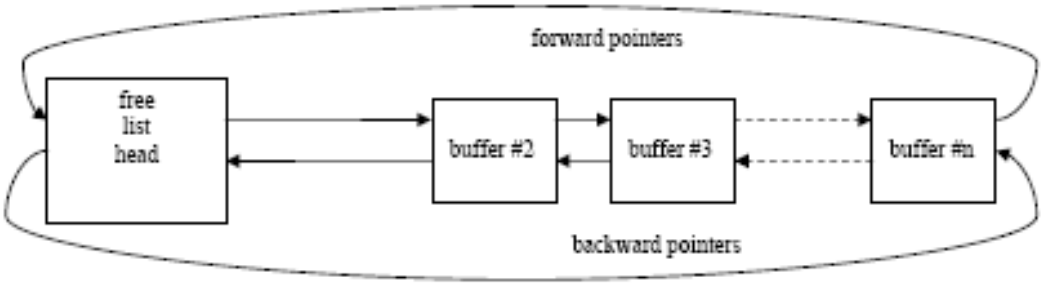
Na sledećoj slici je dat sistem sa 4 hash queue:



Slobodna lista pre dodele bafera #1



Slobodna lista posle dodele bafera #1



in-core inode v buffer cache

- **Mnoga polja** u in-core(inode) su **analogna poljima** u baferskom zaglavlju
- **upravljanje incore inodovima** je slično kao i kod baferskog keša.
- Kada se **inode zaključa**, ostali procesi ne mogu da ga otvore, a ostali procesi postave svoje flagove da su zainteresovani za taj inode i da ih treba probuditi kada se inode otključa.
- Kernel postavlja i **druge flagove** koji ukazuju na razliku između disk inode i njegove in-core kopije, kako bi na bazi tih flagova sravnjuje(destage) stanje inoda na disku.
- **Najveća razlika** između incore inoda i bafer zaglavlja je u **broju referenci** za **in-core inode** koga **uopšte nema kod baferskog keš zaglavlja**.
- **Svaki proces** koji pristupa **in-core inodu** povećava mu **reference count**
- **inode može biti u free listi samo ako je RC=0**.
  - Samo takav inode može napustiti in-core,
  - dok kod **baferskog keša**, **bafer je free listi samo ako je otključan** (unlocked)

iget algorithm (in-core creator)

- **algorithm iget**
- input: **file system and inode number**
- output: **locked inode**
- {
- while(not done)
- {
- **if(inode in inode cache**) /* if #1*/ /***in-core hit***/
- {
- **if(inode locked**) /* if #2*/
- {
- **sleep** (event inode becomes unlocked);
- continue; /* loop back to while*/
- }/* if #2*/
- }/*if #1*/
- **/*special processing for mount point directory*/**
- **if(inode on inode free list) remove from free list;** #if inode on the free list, incore inode **used by no processes**
- **increment inode reference count**;
- return (inode);

iget algorithm (inode not in inode cache)

- **/* inode not in inode cache*/ /*in-core miss*/**
- if(no inodes on free list) **return(error)** #table is full, no chance

#creation of new incore inode

- **remove new inode** from **free list**; # in inode cache
- **reset inode number** in **free list**; # in inode cache
- **remove inode** from **old hash queue, place** on **new one**;
- **read inode** from **disk(algorithm bread)**;
- **initialize inode** (eg. reference count to 1) (RC=1)
- return(inode)
- }/*while*/
- }/*main*/

Pristupanje inodovima

- Kernel identifikuje partikularni inode preko:
  - **FS number**
  - **inode number**
- a alocira incore inode preko **iget** algoritma koji veoma podseća na **getblk** algoritam.
- **Kernel mapira FS broj i inode broj** u **hash queue** i traži inode u odgovarajući hash queue.
- Moguće su 2 situacije:
- **in-core hit**: RC++; lock of inode;
- **in-core miss**
  - Ako ne može da ga nađe, **alocira slobodan inode** iz **free liste**, **lockuje** ga i počinje **čitanje disk inode** u **svoju in-core kopiju**.

Pristupanje disk-inode-ovima

- **disk inode** se određuje prema formuli:
- **block num=**
- **((inode number - 1)/ number of inodes per block)**
- **+**
- **start block of inode list)**
- taj blok se čita pomoću algoritma **bread**.

- Kako je inode tipična struktura od **64 ili 128 bajtova**, offset unutar bloka se određuje:
- **offset =**
- **((inode number - 1) % number of inodes per block))**
- *** size of disk inode**

RC v lock/unlock of in-core inode

- Kernel manipulira sa **inode lock** i RC **nezavisno**.
- Kernel lock inode svaki put kad treba da zaštiti inode od drugih procesa,
- Reference Count (RC)
  - svaki **open SC** povećava RC,
  - svaki **close SC** dekrementira RC.
- Nikada se inode ne lockuje sa posebnim SC,
- lock i unlock of inode se obavljaju u toku SC
- (iget: lock by iget, unlock by status field in incore),
- (iput: lock and unlock by iput)

Oslobađanje inode iput

- Kada kernel oslobađa inode,
- **prvo se dekrementira incore RC za taj inode**.
- Ako **RC padne na 0**,
  - kernel **upisuje incore inode** na **disk** inode **samo ako ima promena.**
  - **incore inode ide u free listu.**
  - **ako i LC padne na 0,** kernel takođe može oslobadja blokove podataka datoteke (brisanje)
- **Incore inode** je i dalje tu (u kešu) ali se može izbaciti ako nema više mesta u inode cache memoriji.

iput

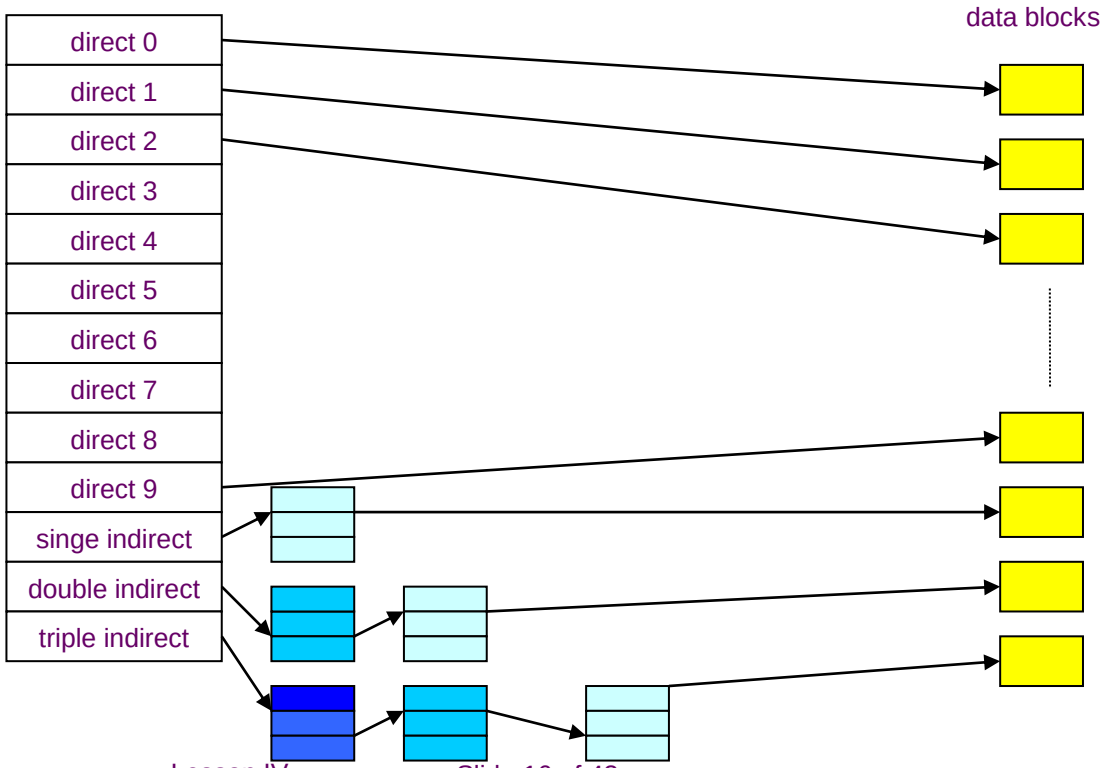
```
algorithm iput          /* access to in core inode*/
input: pointer to incore inode
output: none
{
        lock inode if not already locked;
        decrement inode reference count; RC- -;
        if(reference count == 0)
        {
                if(link count == 0) {
                free disk blocks for file (algorithm free);
                set file type to 0;
                free inode (algorithm ifree) }
    if (file accessed or inode changed or file changed)  update disk inode;
                put inode on free list;
        } /*RC is not 0*/
release inode lock;
}/*main*/
```

no more process

file removing

Struktura regularne datoteke

- Svaki blok na disku ima unikatnu adresu, i može pripadati samo jednoj datoteci.
- Svi blokovi koji pripadaju datoteci moraju se naći u inodu te datoteke.
- Da bi **inode imao malu veličinu**, ali da bi **mogao opisivati efikasno i male ali i veoma velike datoteke**, koriste se inode šema uvedena na UNIX System V na primer sa 13 ulaza.
- **Broj ulaza može biti proizvoljan** kao i **broj stepena indirekcije**.

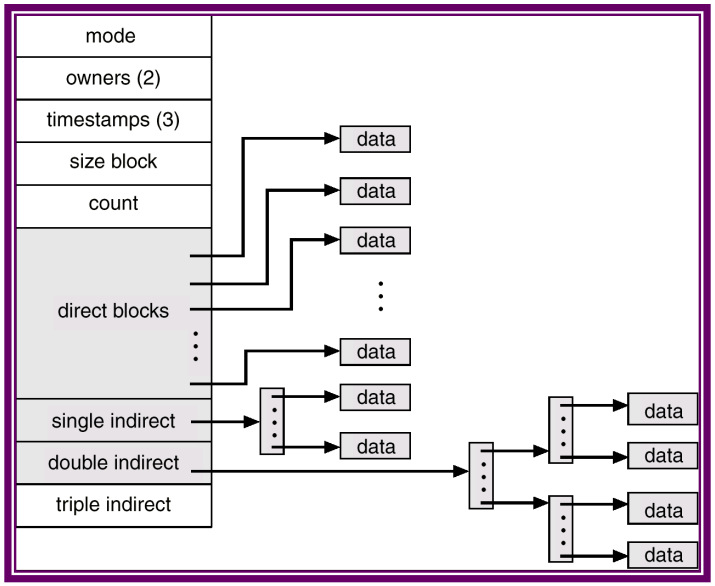


File max

- Sa 32 bitnim pointerima, 1K blokovima = 256 pointers per block
- 10 direct blocks with 1K bytes each = 10K bytes
- 1indirect block with 256 direct blocks = 256K bytes
- 1double indirect block with 256 indirect blocks = 64M bytes
- 1triple indirect block with 256 double indirect blocks= 16G bytes

algorithm bmap

- **/*block map of logical file byte offset to file system block*/**
- **byte offset ->FS block**
- **algorithm bmap**
- input:
  - (1) inode
  - (2) byte offset
- output:
  - (1) block number in file system
  - (2) byte offset into block
  - (3) bytes of I/O in block
  - (4) read ahead block number



- {
- **calculate logical block number in file** from **byte offset**; (**byte_offset/1K)**
-
- **calculate start byte** in **block** for I/O; (**byte_offset%1K)** /* output 2*/
- **calculate number** of **bytes** to **copy** to **user**; /* output 3*/
- **check** if **read – ahead** applicable, mark inode; /*output 4/*

algorithm bmap

- **determine level of indirection;**
- while (**not at necessary level of indirection**)
- {
- calculate index into inode or indirect block from logical block number in file;
- **get disk block number** from **inode** or **indirect block**;
- **release buffer** from **previous disk read**, if any (algorithm brelse);
-
- if (no more levels of indirections) **return(block number);**
- **read indirect disk block** (algorithm bread)
- **adjust** logical block in file **according** to **level** of **indirection**;
- }
- }

algorithm bmap

- Moguće su **0 u pointerima**,
  - to znači da proces nije nikada ništa upisao na tom offsetu i
  - **ti blokovi ostaju na 0**, tj sadrže 0-le
  - a **ne troše prostor** na disku.

- **BSD** uvodi **veće sistemske blokove** 4K, 8K i
  - uvodi pojam **block/fragment,**
  - gde jedan sistemski blok može sadržavati fragmente
  - koji pripadaju različitim datotekama.
  - To zahteva modifikaciju inode strukture.

Direktorijumi

- To su **specijalne datoteke** koje imaju značajne uloge za UNIX stablo.
  - Sastoje se od specijalnh struktura FCB,
  - od kojih svaka odgovara jednoj datoteci, a
  - sadrže ime datoteke i inode koji joj pripada.
- UNIX System V ima maksimum 14 bajtova za ime i 2 bajta za inode.
- Evo jednog isečka etc direktorijuma:
- **prvi ulaz . predstavlja sam taj direktorijum**, **drugi ulaz .. odgovara roditelskoj grani**, ostalo su grane i datoteke

Byte offset in directory	Inode number (2 bytes)	File Names
0	83	.
16	2	..
32	1798	init
48	1276	fsck
64	85	clri

- Mada se direktorijumi tretiraju kao datoteke,
  - kernel ne dozvoljava direktan upis u direktorijume
  - upis u diride preko SC poziva, creat, mknod, link i unlink SC.
- Program **mkfs** kreira **root direktorijum koji ima 2 ulaza . i .. koji dele isti inode i to je root direktorijum.**

Konverzija path imena u inode - namei

- Inicijalni pristup datoteci se odvija preko njenog path imena u SC kao što su **open**, **chdir** ili **link**.

- **Kernel interno radi** sa **inodovima**, radije nego sa **path imenima**, (ovo su brojevi, a ovo su imena različitih veličina).

- Algoritam **namei** konvertuje **path imena** u **inodove** preko kojih se pristupa datoteci.

- Algoritam **namei** razbija **pathname**
  - na **jednu pojedinačnu komponentu** iz imena
  - u **jednom trenutku**,
  - **konvertujući to ime** u njegov **inode**,
  - **tako što nalazi FCB tog imena** u **roditeljskom direktorijumu**.

Konverzija path imena u inode namei

- **Svaki proces** ima svoj **tekući direktorijum** čiji se inode upisuje u **u-area** za taj proces.
- Proces dobija tekući direktorijum od svog procesa roditelja, a može ga promeniti preko **chdir SC**.
- Sve path komponente startuju od tekućeg direktorijuma, osim ako nemaju **leading /,** koji ukazuju da sve počinje od root direktorijuma.
- Kernel to zna da identifikuje, a **u-area** sadrži **tekući direktorijum**
- dok se **root inode** čuva u globalnoj varijabli.
- Algoritam **namei** koristi **među-inodove** ili srednje inodove ili working inodove, a ima **iterativnu petlju** u **čijem svakom prolazu se analizira jedna grana**, za koju **user mora imati r** i **x pravo**.

algorithm namei
/* convert path name to inode*/

- **algorithm namei** /* convert path name to inode*/
- input: **path name**
- output: **locked inode = locked incore inode)**
- {
- if (path name **starts** from root)
- 	**working inode** = **root inode** (algorithm **iget**);
- else
- 	**working inode** = **current directory inode** (algorithm **iget**);

algorithm namei
/*convert path name to inode*/

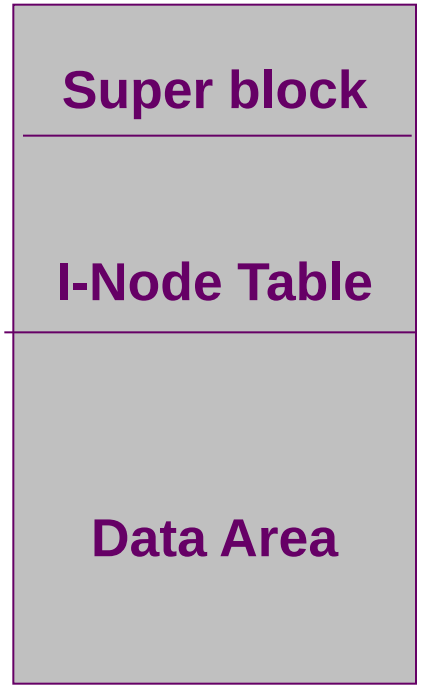
- while (**there is more path name**)
- {
  - read next path name component from input;
  - verify that working inode is of directory, access permissions OK;
- **if (working inode is of root and component is “..”) continue;**
  - /*loop back to while*/
- **read directory (working inode)**
  - by repeated use of algorithms bmap, bread and brelse
- if (**component matches an entry in directory** (working inode))
- {
  - **get inode number** for **matched component**;
  - release working inode (algorithm iput);
  - **working inode** = **inode** of **matched component** (**algorithm iget**)
- }
- **else**
- return (**no inode**);
- }/* while */
- **return (working inode);**
- }

namei

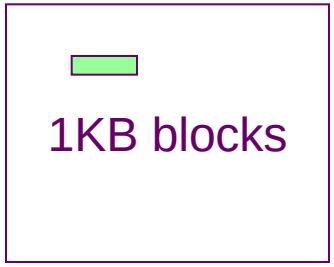
- Kernel obavlja **linearno pretraživanje** datoteka
  - koje se nalaze u working inodu,
  - pokušavajući da nađu sledeću path komponentu,
  - startujući u offsetu 0,
- Preko **bmap** se određuje blok na disku
  - koji se čita preko **bread**,
  - a kada se blok dobije on se otpušta preko **brelse**.
- Potom se direktorijumski blok pretražuje za path komponentu i
  - ako je nađe iz FCB se dobija njen inode,
  - a oslobađa se stari working inode,
  - uzima novi sa **iget**.
- Ako ne nađe podudarenje,
  - mora pročitati sve direktorijumske blokove dok ne nađe FCB,
  - a može se dogoditi slučaj da ga uopšte ne nađe.

Superblock

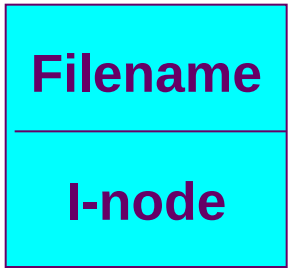
- FS Layout



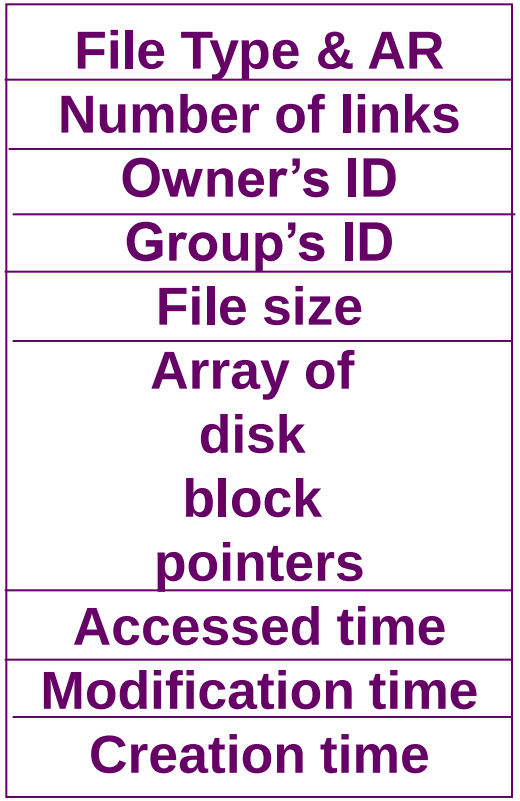
DATA area



File-info



I-node



Super Block

- Superblock se sastoji od sledećih polja:
  - **veličina FS**
  - **broj slobodnih blokova** u FS
  - **lista slobodnih blokova raspoloživih** u FS
  - **Index/ukazivač** na **sledeći slobodan blok** u **free block listi**
  - **veličina inode tabele**
  - **broj slobodnih inodova** u **FS**
  - **lista slobodnih inodova raspoloživih** u **FS**
  - **index/ukazivač na sledeći slobodan inode** u **free inode listi**
  - **lock polja** za **liste slobodnih blokova** i **za liste slobodnih inodova**
  - **dirty flag** koji ukazuju da je **superblock modifikovan** i da mora da se sravni sa **disk super-blokom**

Dodeljivanje inoda za novu datoteku

- Algoritam **ialloc** obavlja **dodeljivanje disk inoda** za **novo kreiranu datoteku**.
- FS sadrži **linearnu listu inodova**, a **inode je slobodan** ako je njegov sadržaj **nula**.
- **Kada** se **kreira** nova datoteka mora se **naći slobodan inode**,
- a to **zahteva intenzivno pretraživanje inode tabele** što može dugo da traje zbog **višestrukih disk čitanja**.
- **Da bi se to ubrzalo**, **superblock sadrži keširano polje** koje sadrži listu slobodnih inodova.

ialloc

- algorithm ialloc /* allocate inode */
  - input: **FS**
  - output: **locked inode**
- {
- while (not done) {
- **if (super block locked)**
- { sleep (event super block becomes free); continue; /*loop back to while*/ }
- **if (inode list in superblock is empty) /*empty SB list*/**
- { lock super block;
- get remembered inode for free inode search;
- **search disk for free inodes** until super block full, or no more free inodes
  - (algorithms bread i brelse)
- **unlock superblock**; wake up (event superblock becomes free);
- **if(no free inodes found on disk) return(no inode);**
- set remembered inode for **next free inode search**; }

ialloc

- **/* there are inode in superblock inode list*/**
  - **get inode number** from **super block list**;
  - **get inode** (algorithm **iget**);
- **if(inode not free after all) /* !!! */**
- {
  - write inode to disk;
  - release inode (algorithm iput);
  - continue /* while loop*/
- }
- **/* inode is free*/**
- initialize inode;
- write inode to disk;
- decrement FS free inode count;
- return(inode);
- }/* while */
- }

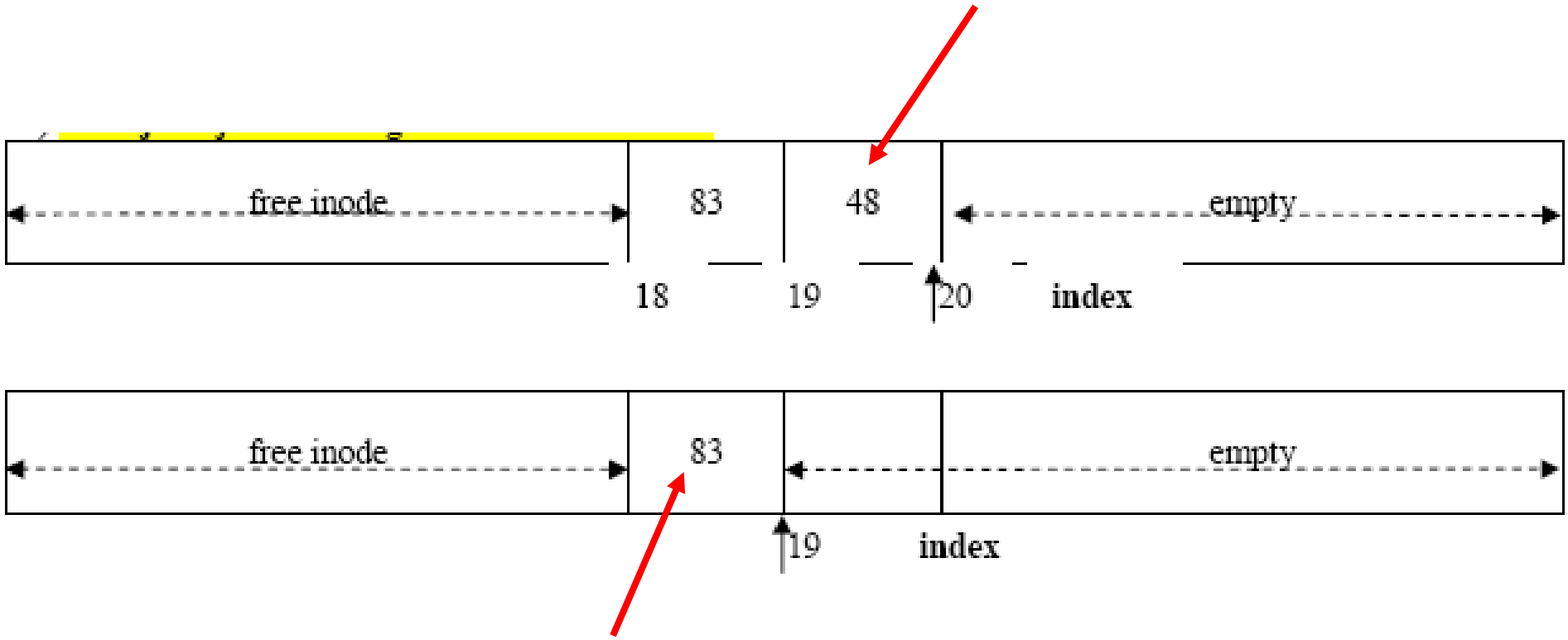
ialloc

- Kernel prvo mora da proveri da li je superblock locked,
- jer se u njemu nalazi free inode lista.

- Ako **lista nije prazna**, kernel uzima sledeći inode,
  - alocira novi in-core inode (algoritam iget),
  - **popunjava oba inodes (disk inode i in-core inode) i**
  - **vraća locked inode**.

- Ako je superblock **lista prazna**,
  - kernel mora čitati inode tabelu sa diska i
  - popuniti superblock listu,
  - naravno pamteći koji je zadnji inode detektovao i
  - koji je to blok tabele zadnji čitao,
  - da bi sledeći put ponovo čitao
  - (remembered inode, remembered block).

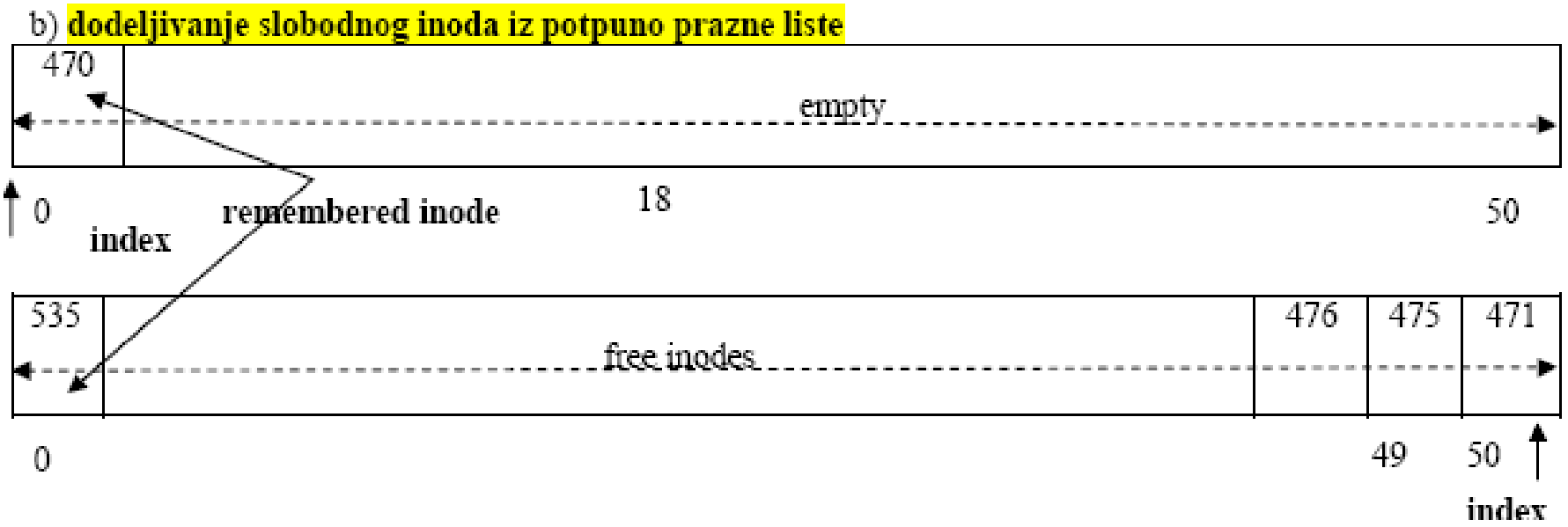
dodeljivanje slobodnog inoda iz sredine liste



- U ovom slučaju dodeljuje se inode=48 koji na indexu 19 u listi slobodnih inodova.

dodeljivanje slobodnog inoda iz potpuno prazne liste

- kreće se od zapamćenog inoda 470,
- sa diska se čita **inode tabela** i popunjavaju se svih 50 ulaza u superbloku.
- Index se pomera na zadnju poziciju, a zadnji zapamćeni nod se ažurira na 535, koji će služiti za sledeće disk pretraživanje.



ifree

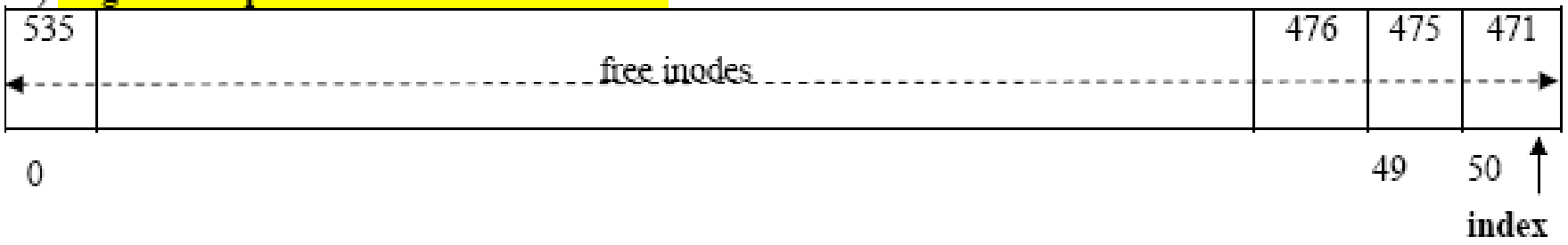
- Algoritam za oslobađanje inode je mnogo jednostavniji.
- **algorithm ifree** /* inode free */
  - input: FS, inode number
  - output: none
- {
- increment FS free inode count;
- if (super block locked) return;
- if (inode list full)
- {
- if(inode number **less than** remebered inode for search)
- set remebered inode for search = input inode number;
- }
- else
- **store inode number in inode list;**
- return;
- }

ifree

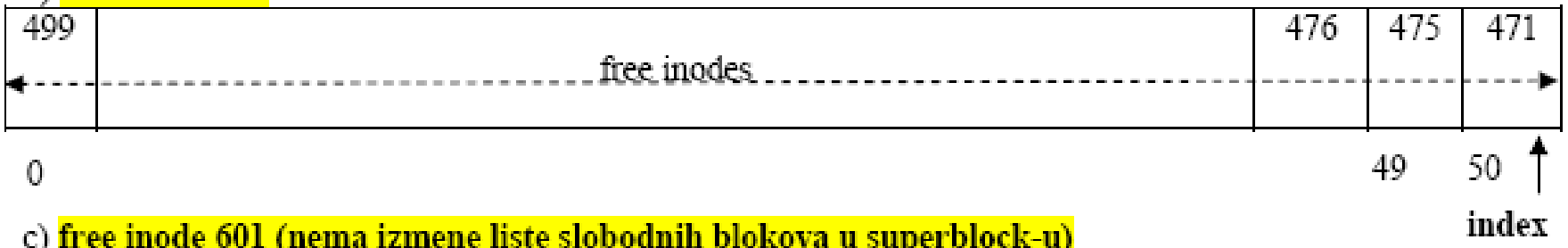
- Pravila:
- lista je sređena po opadajućim brojevima, osetno je manja nego inode tabela.
- sadrži svoje ulaze koji se indeksiraju od najvišeg do najnižeg
- poslednji inode (**index 1**) uvek predstavlja zapamćeni inode (**remembered**)
- u slučaju potpuno prazne liste (**svi inodes free**) ako se pojavi novi free inode, doći će do promene na **head** tabele samo ako je novi inode manji od zapamćenog u protivnom se ne ubacuje u listu.

ifree

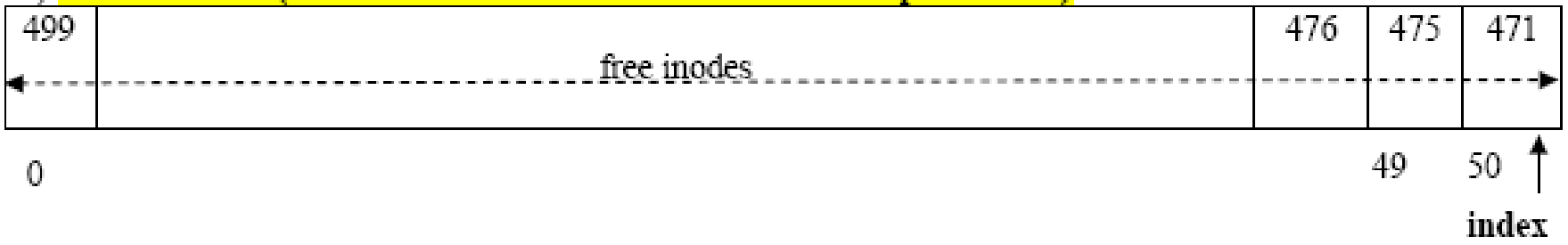
a) **originalni super block i lista free inodova**



b) **free inode 499**



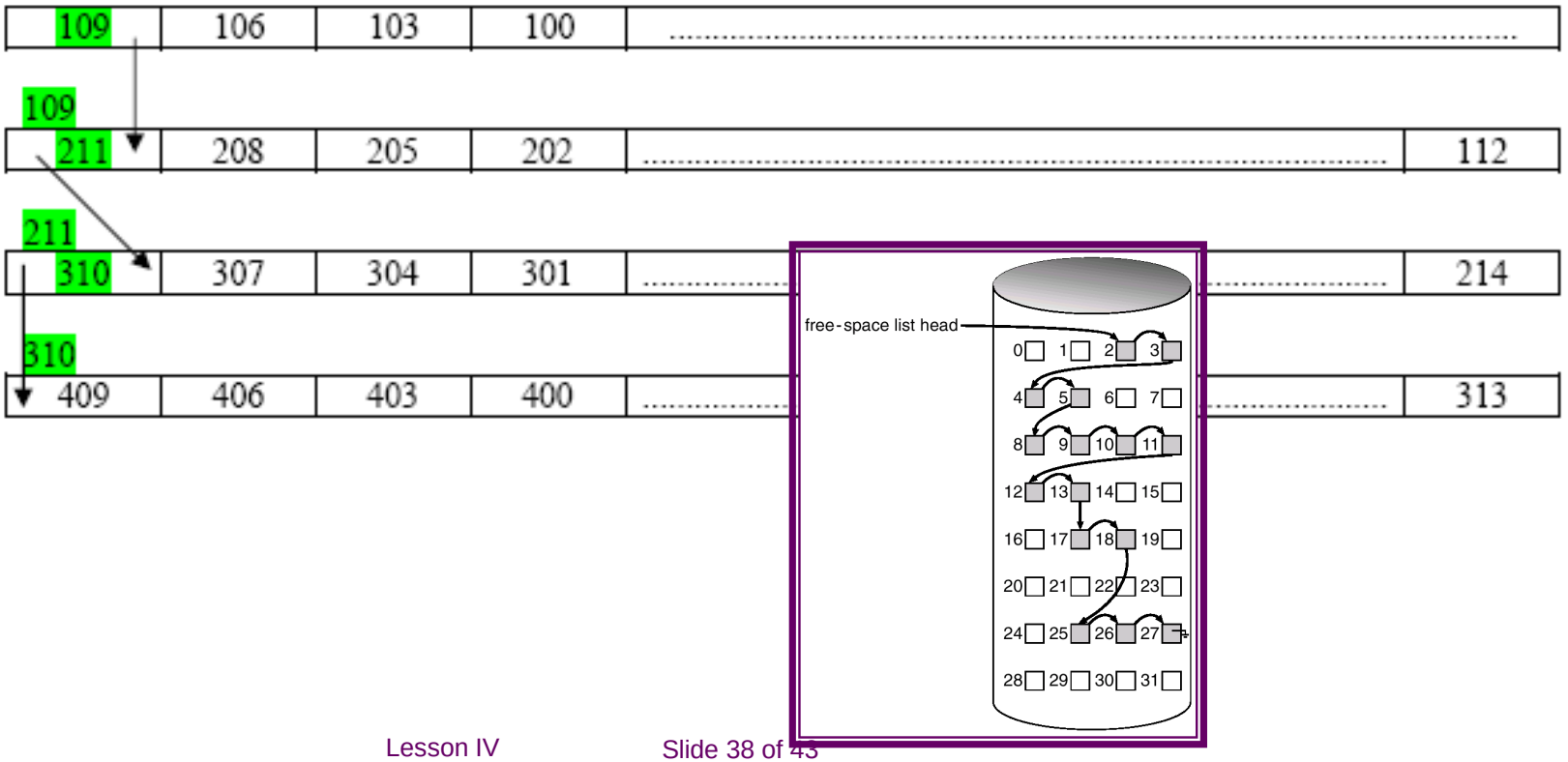
c) **free inode 601 (nema izmene liste slobodnih blokova u superblock-u)**



Naravno u svi ovim situacijama moguće su race situacije, kad kernel dodeljuje inode, kada kreira in-core inode itd. Mada se performanse smanjuju zbog locking superblocka i inodova, ipak regulišu race conditions.

Alokacije disk blokova

- Svakoj datoteci mora biti prvo dodeljen blok iz **liste slobodnih blokova**,
- a svi njeni dodeljeni blokovi se **upisuju** u **inode** u **direktne** i **indirektne ukazivače**.
- **In-core superblok** sadrži **ograničenu listu slobodnih blokova** sa **ukazivačem** na **blokove na disku koje sadrže listu sledećih blokova**.
- Program **mkfs** kreira ovu **linkovanu listu** of **free blocks čiji je početak** u **superblocku**.



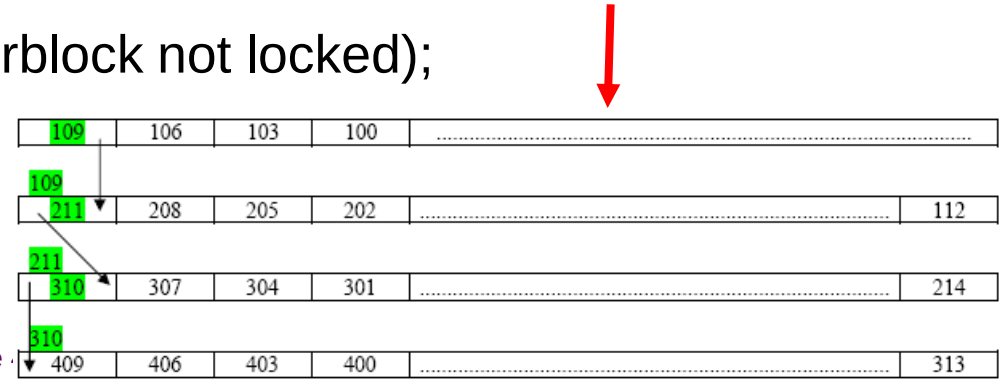
Alokacije disk blokova

- Kada kernel želi da dodeli blok iz FS,
  - kernel prvo analizira keširanu listu iz superbloka i
  - **uzima prvi slobodan blok**.
  - Samo ako je to zadnji slobodan blok iz superblok keša,
  - tada se prelazi na **listu sa diska**,
  - čita se blok iz linkovane liste i popunjava **superblok free list keš**
  - iz koga se uzima prvi slobodan blok,
  - kome se dodeljuje mesto u kešu (**getblk**) i puni se nulama.

- Program **mkfs** i se trudi da liste slobodnih blokova sadži blokove **sa sličnim adresama**, naravno iz razloga performansi.
- Međutim, ta harmonija se brzo narušava jer se **free blokovi** često izbacuju i vraćaju u listu sasvim slučajno.
- **Kernel pokušava povremeno** da **sortira free liste**.

alloc

- algorithm alloc /* FS block allocation*/
  - **input: FS number**
  - **output: buffer for new block** {
- while (**super block locked**) sleep (event super block not locked);
- remove block from super block free list;
- if (removed **last block** from **free list**)
- {
- lock superblock;
- **read block just taken** from **free list** (algorithm **bread**);
- **copy block numbers** in block **into super block**;
- **release block buffer** (algorithm **brelse**);
- unlock superblock;
- wakeup processes (event superblock not locked);
- }



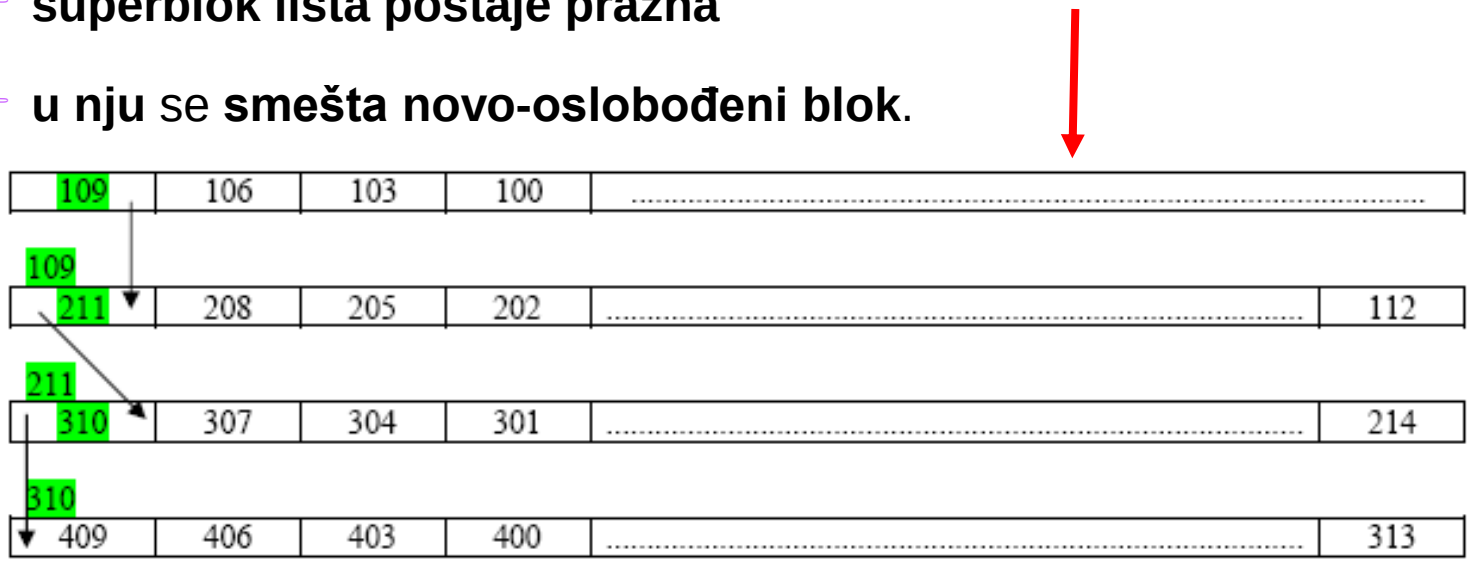
alloc

- **/* cache buffer allocation*/**
- **get buffer** for **block removed** from **superblock list** (algorithm getblk);
- **zero buffer contents**;
- decrement total count of free blocks;
- mark **superblock modified**;
- **return buffer**;
- }

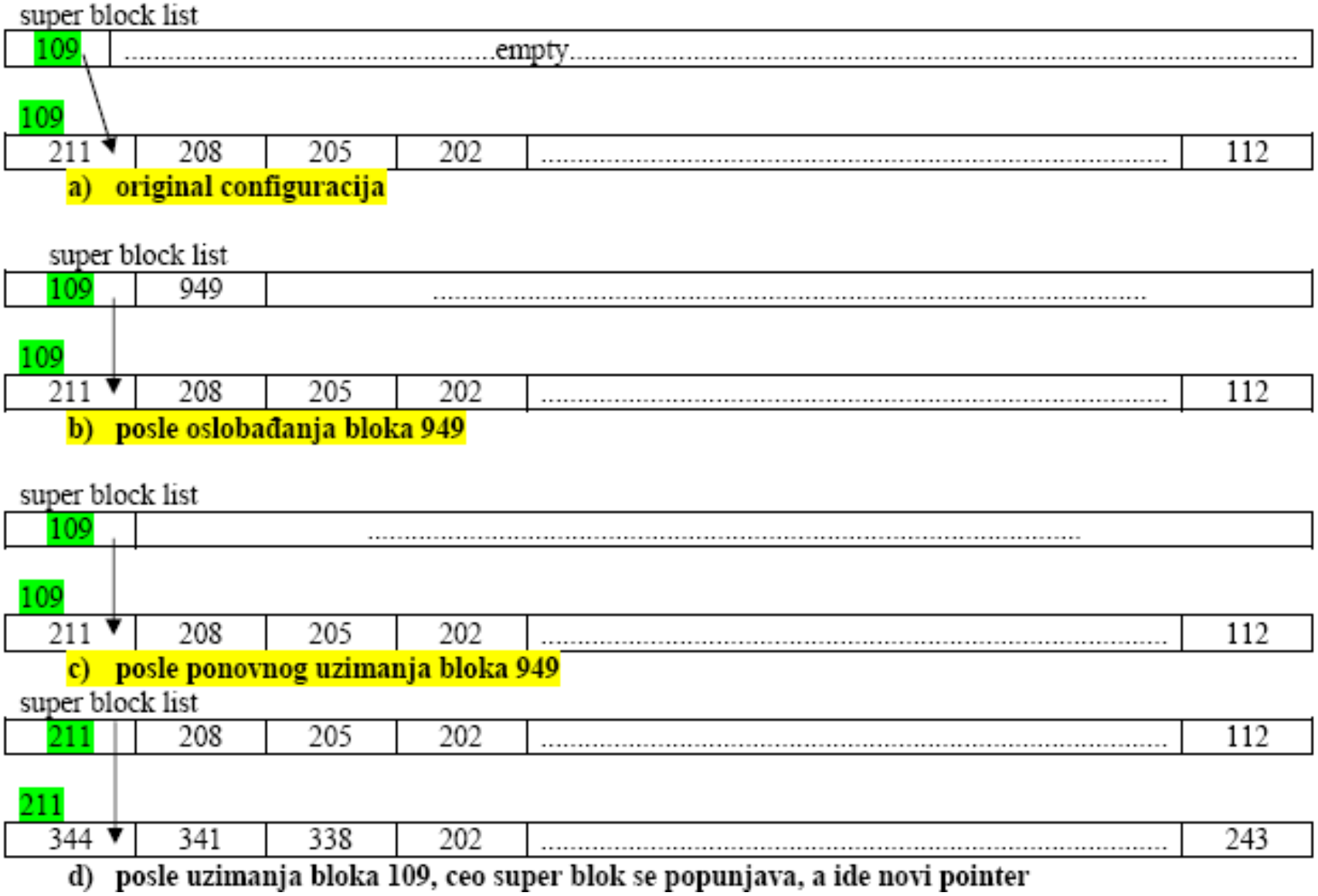
free

- Algoritam **free** radi **povratak bloka u free listu.**

- Ako **keširana superblok** lista **nije puna**,
  - **blok** se **umeće u keširanu superblok listu**.
- **Ako je puna**,
  - **lista** se **upisuje** u **neki blok na disku**
  - **blok** se **stavlja** u **linkovanu listu**,
  - **superblok lista postaje prazna**
  - **u nju** se **smešta novo-oslobođeni blok**.



example


super block list
109
empty
109
211
208
205
202
112
a) original configuracija
super block list
109
949
109
211
208
205
202
112
b) posle oslobađanja bloka 949
super block list
109
109
211
208
205
202
112
c) posle ponovnog uzimanja bloka 949
super block list
211
208
205
202
112
211
344
341
338
202
243
d) posle uzimanja bloka 109, ceo super blok se popunjava, a ide novi pointer

No linked list for free inodes: reasons

- Veoma je zanimljivo primetiti da se **linkovane liste** ne primenju za slobodne inodove.
- Ima **3 glavna razloga** za drukčiji tretman blok free listi i inode free listi:
  - Kernel može odrediti da li je **inode** slobodan ispitivanjem njegovog sadržaja, ako **type** polje slobodno i **inode** je slobodan. Međutim, nema nikakvog načina da se odredi da li je blok slobodan ili nije na osnovu njegovog sadržaja. Zato, podrška cele **free liste** za blokove neophodna.
  - **Disk blokovi su veoma povoljni za povezane** liste zato što jedan disk blok može **sadržati veliki broj pointera na slobodne blokove**. Na drugoj strani **mnogo** je **veći broj blokova nego broj inodova.**
  - Procesi mnogo više **teže da koriste slobodne blokove** nego **slobodne inodove**, tako da je **mnogo bitnije optimizovati performanse** za **liste slobodnih blokova** nego za **inode liste**.

Drugi tipovi datoteke

- UNIX sistem podržava još 2 tipa datoteka:
- **pipe**
- **specijalne** datoteke.
- **Pipe** datoteka koja se zove još i **FIFO** razlikuje se od obične datoteke po **tranzijentnim podacima**:
  - kada se podaci jedanput pročitaju iz **pipe** datoteke, ne mogu se ponovo pročitati,
  - takođe podaci se čitaju onim redosledom kojim u upisivani,
  - nema promene poretka.
- Kernel upisuje pipe datoteku na disk na isti način kao i obične s tim što korisiti isključivo **direktne pointere** a ne indirektne.
- **Specijalne datoteke** su
  - blok specijalne
  - karakter specijalne datoteke
- koji **specificraju uređaje** i njihov inode ne ukazuje na nikakve podatke na disku, već njihov **inode** sadrži 2 broja: **major** i **minor** number.
  - major broj **selektuje klasu uređaja** kao što je disk
  - minor broj ukazuje uređaj unutar klase.