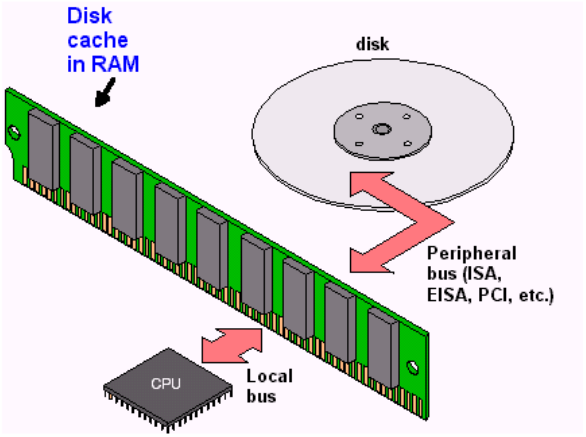
Lesson III: Buffer cache

- **Kernel pokušava** da **smanji broj disk pristupa** preko **baferskog keša,**
  - a to je **skup internih bafera podataka**
  - koji sadrži podatke **nedavno korišćenih disk blokova**.
- **(read)** Kada se proces obraća datoteci za čitanje,
  - kernel pokušava da je locira u kešu,
  - ako je u kešu ne čita je sa diska (**read hit**),
  - a ako nije prvo se dovede u keš (**read miss**),
  - čuvajući u kešu one podatke za koje algoritam smatra da su najkorisniji.
- **(write)** Po pitanju upisa, keš je vrlo beneficijalan,
  - može da **kompenzuje višestruka upisivanje** ili
  - da **potpuno elimiše upis** na **disk**, (ako usledi proces brisanja datoteke)
  - takođe više odloženih upisa se mogu kombinovati u optimalnom redosledu.
- Keš algoritam izdaje instrukcije za
  - pre-cache (read-ahead) i
  - delayed-write da bi se maksimizovao keš efekat.

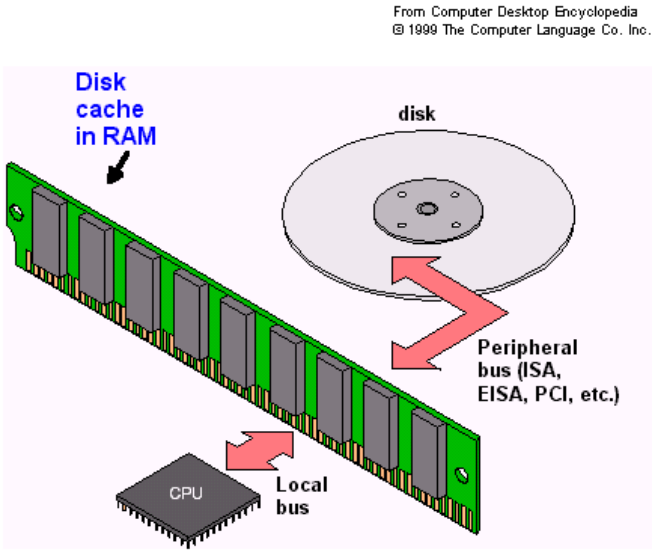


Baferi (Buffers)

- Za vreme **sistemske inicijalizacije**,
  - kernel alocira prostor za **određeni broj bafera (total cache size)**,
  - a taj prostor zavisi od **memorijske veličine** i **sistemskih performansi**.
- Bafer se sastoji od **2 dela-komponente**:
  - **bafer:** memorijsko polje koje sadrži podatke, na nivou logičkog bloka FS
  - **bafersko zaglavlje (header)** koje identifikuje bafer
- **Kernel ispituju zaglavlja** kako bi **analizirao sadržaj keš bafera**.

tag memory

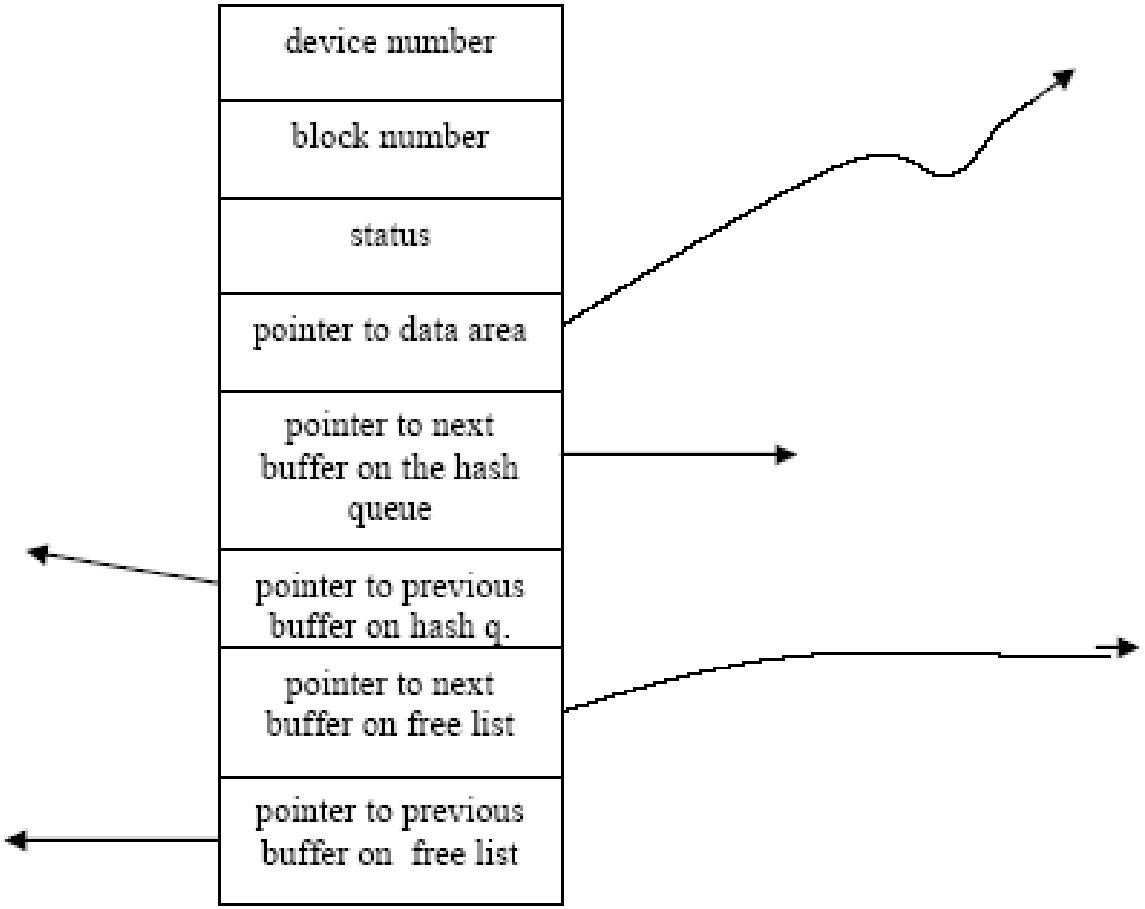
data cache memory



Baferska zaglavlja (Buffer headers)

- **Bafer zaglavlje** sadrži **sledeća polja**:
- **device number**
- **block number**
- **data pointer:** ukazivač na blok iz data-area (mora da pokrije sve blokove iz data area)
- **status:**
  - **locked**: bafer je trenutno locked što znači da je zauzet (puni se ..)
  - **valid**: bafer sadži validne podatke
  - **delayed write**: kernel mora upisati sadržaj bafera na disk pre nego što obavi ponovnu dodelu bafera
  - **in reading/writing:** kernel trenutno čita ili piše po baferu
  - **wait for be free:** proces čeka da bafer postane slobodan
- **set pointera** za alokaciju bafera koje koristi keš algoritam

Buffer header



Struktura bafera (Buffer pool)

- **Baferi se opslužuju po LRU algoritmu**:
  - kada se bafer dodeli disk bloku,
  - taj blok će ostati tu
  - sve dok ne postane **najstariji** u baferu po pitanju obraćanja.

- **Kernel održava listu slobodnih** bafera u **LRU redosledu**.

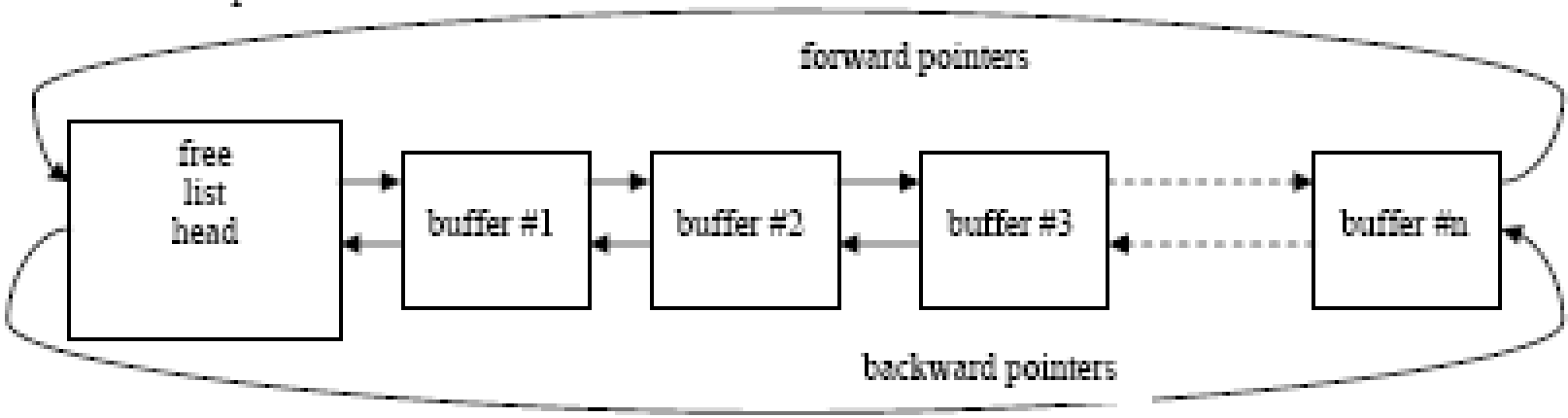
- **Slobodna lista** je **duplo linkovana cirkularna lista** sa **header-om na početku.**

- U **početku**, **svi su baferi slobodni**:
- **Buffer is free?**

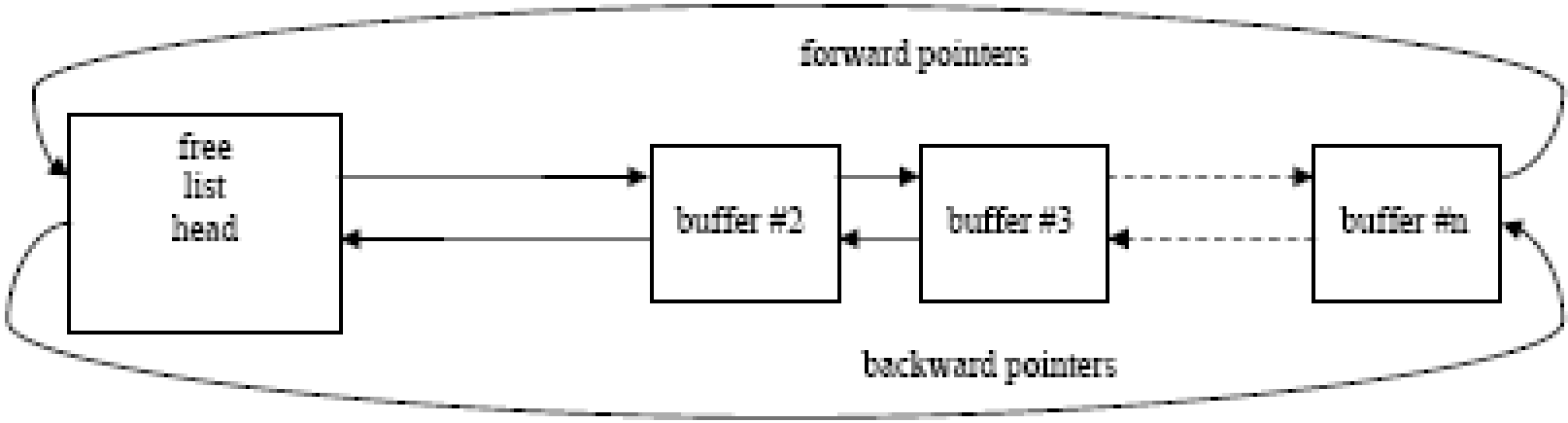
- kernel uzima bafer koji je na početku free liste
  - tada ga izbacuje iz slobodne liste,
  - a po povratku ga stavlja na kraj free liste.

Struktura bafera (Buffer pool)

Slobodna lista pre dodele bafera #1



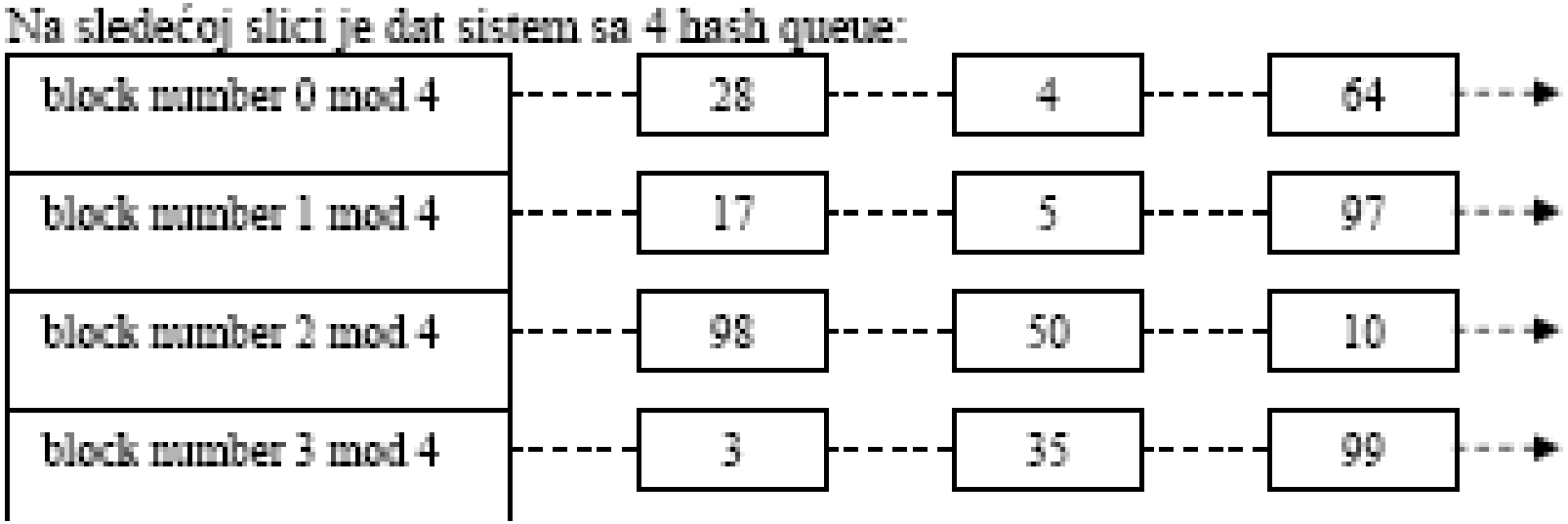
Slobodna lista posle dodele bafera #1



Struktura bafera (Buffer pool)

- Kada kernel prisitupa disk bloku,
- **on pretražuje sve headers** za odgovarajuću kombinaciju
- <**device, block number**>
- **Da ne bi pretraživao sva zaglavlja**,
  - kernel organizuje cache-bafer u odvojene redove (**hash queue**)
  - na bazi hash funkcije od **device-block** broja
- Kernel povezuje bafere na duplo linkovani hash redove čekanja, u sličnu strukturu kao za listu slobodnih blokova
- **Broj bafera** za jedan **hash queue** je **promenljiv** i zavisno od obraćanja disku
- Kernel korsiti **hash funkciju** za raspodelu baferu između hash redova, koja mora biti dovoljna brza
- **Broj hash redova** se **određuje prilikom podizanja sistema** i to može po default-u ili sistem administrator određuje

hash queue example



- Na levoj strani su headeri, blokovi se raspoređuju u **hash redove** po funkciji LBA mod 4.
- Isprekidane linije ukazuju na hash pointere u oba smera.
- Svaki bafer je u nekom od hash queue, ali može se dogoditi da bude i u slobodnoj listi (never allocated).

Tehnike za dobijanje podataka iz bafera

- **Keš algoritam** upravlja baferskim kešom. **Za svako čitanje sa diska kernel** mora prvo da odredi da li je blok u kešu (cache buffer pool) i ako nije tamo mora da mu **dodeli slobodan bafer**. To se **isto** dešava i **prilikom upisa**, oba algoritma i za čitanje i za upis koriste čuveni UNIX **getblk** algoritam koji fukcioniše na sledeći način
- Algoritam **getblk** ima **5 mogućih scenarija** prilikom dodele baferu u kešu:
  - #1 **Kernel pronalazi** bafer u njegovom **hash queue** i bafer je slobodan **(hit&free)**
  - #2 Kernel **ne nalazi** bafer u njegovom **hash queue**, zato može da alocira bafer iz free liste **(miss&there is free)**
  - #3 Kernel **ne nalazi bafer** u njegovom hash queue, zato može da alocira bafer iz free liste ali taj bafer ima atribut delayed write, što znači da mora prvo da se upiše na disk **(miss&there is free, dirty)**
  - #4Kernel **ne nalazi bafer** u njegovom hash queue, ali i **free lista je prazna (miss&there is no free)**
  - #5 Kernel nalazi bafer u njegovom hash queue, **ali je taj bafer locked (busy),** neko drugi ga je vec da ga čita **(hit&busy)**
- Pažnja: blok je free, ako je u free listi, koja znači da ga niko ne koristi pri čemu može biti u hash listi ili ne..

getblk (block is in the cache)

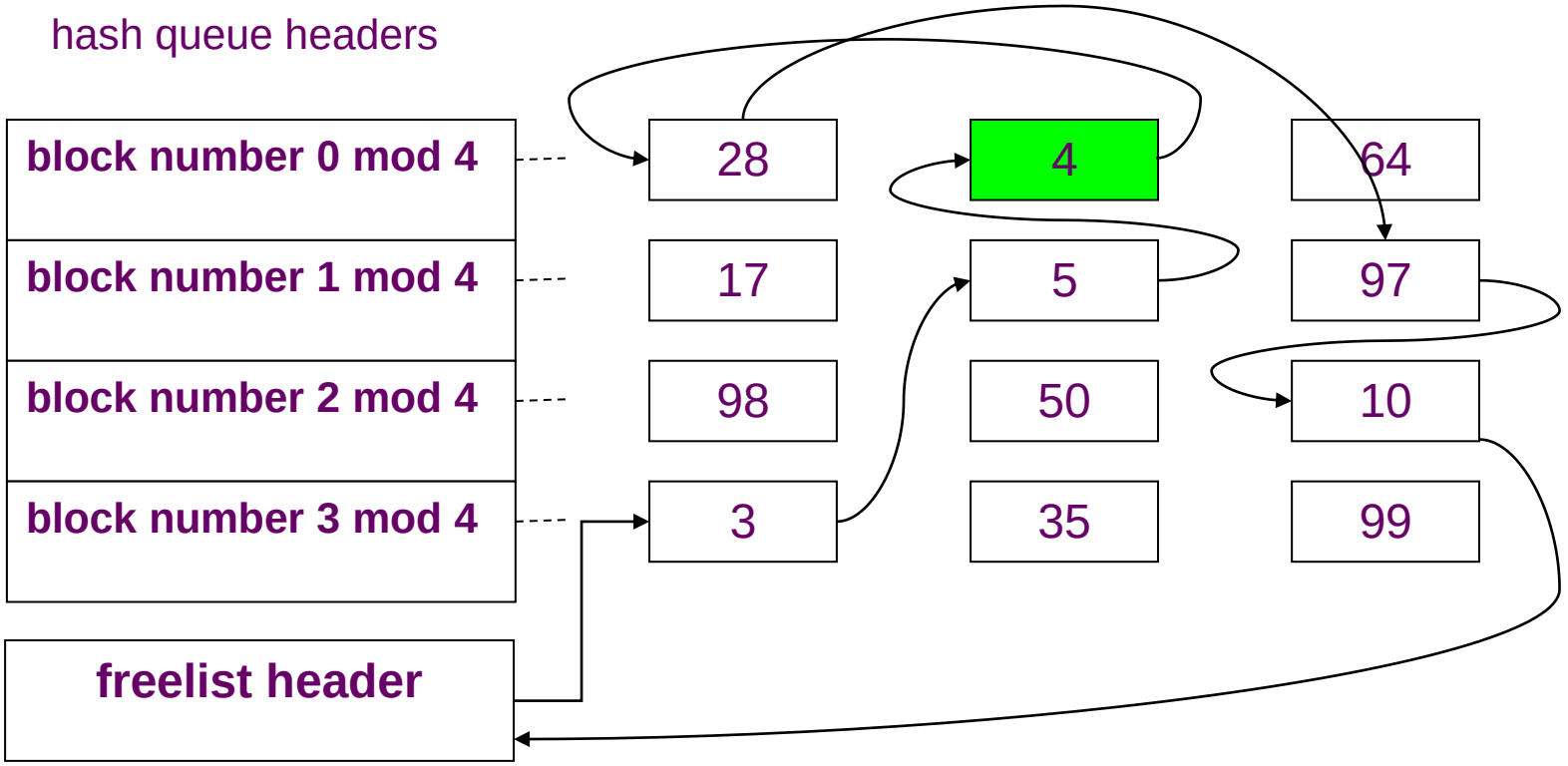
- **algoritam getblk**
- input: **FS number, block number**
- output: **locked buffer** that can now be used for block
- {
- while (buffer not found)
- {
- if (**buffer in hash queue**)
- {
- if **(buffer busy**) /***scenario 5***/
- {
- sleep (**event buffer becomes free**);
- continue; /* back to while lop*/
- }
- **mark buffer busy**; /***scenario 1***/
- **remove buffer** from **free list**; /*lock*/
- return buffer;
- }

getblk (block is not in the cache)

- else /***block not on hash queue***/
- {
- if (**there are no buffers on free list**) /***scenario 4***/
- {
- sleep (**event any buffer becomes free**);
- continue; /* back to while lop*/
- }
- remove buffer from free list; /*lock*/
- if (**buffer marked for delayed write**) /***scenario 3***/
- {
- asynchronous write buffer to disk;
- continue; /* back to while loop*/
- }
- /* **scenario 2 – found a free buffer***/
- **remove buffer** from **old hash queue**;
- **put buffer** onto **new hash queue**;
- return buffer;
- }/*else*/ }/*while*/ }/*main*/.

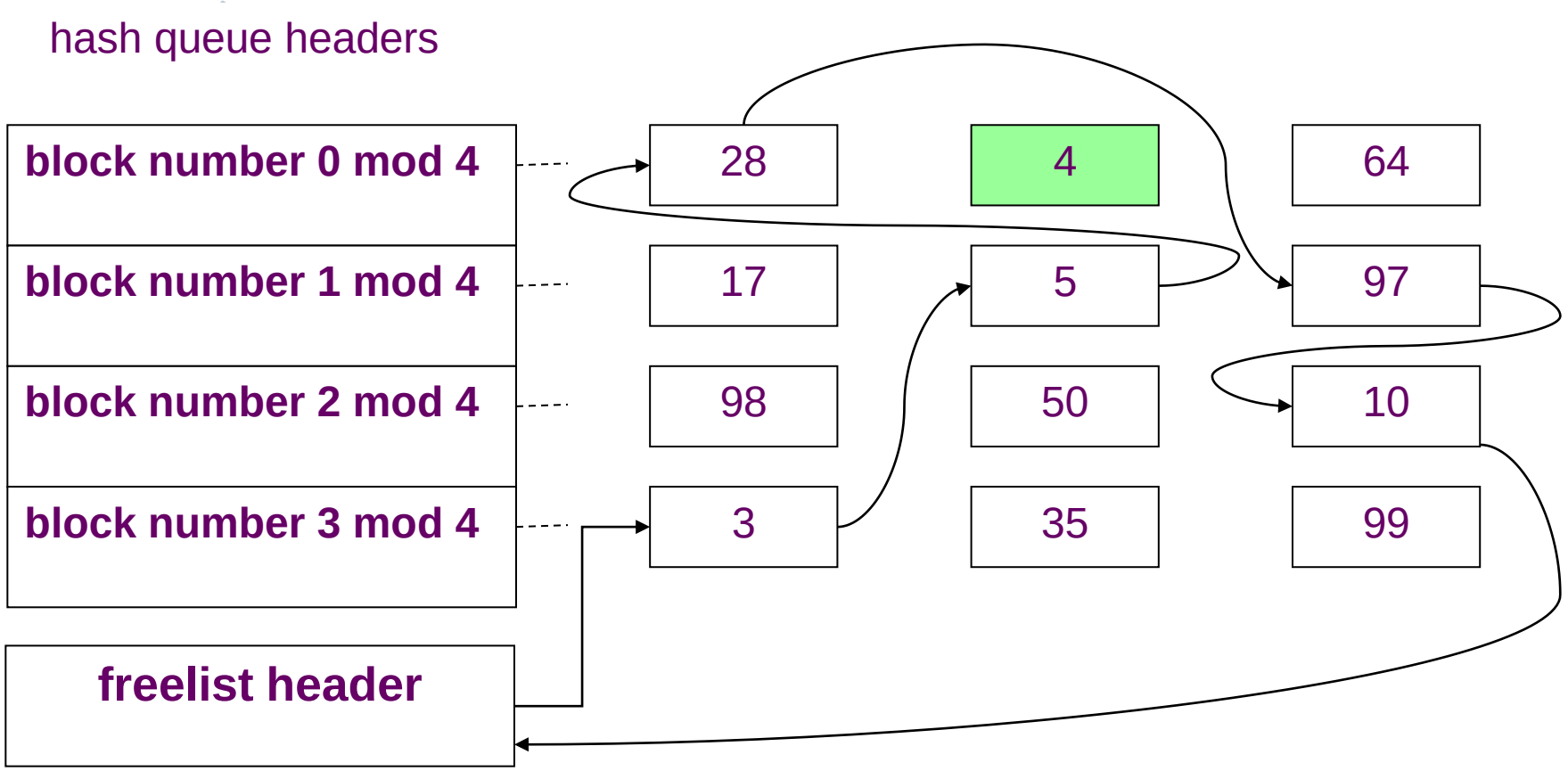
scenario #1 (searching)

- Tražimo blok 4, blok 4 se nalazi u hash queue 0 i slobodan je



scenario #1 (allocating)

- **Blok 4 je nađen, izbacuje se iz free liste**



after allocating

- Pre nego što objasnimo ostale scenarije, demonstrirajmo šta se dešava kada se bafer dodeli.

- Svaki proces posle **getblk** dobija bafer **koga kernel je locked** samo za njega.
- Iza toga blok može da se čita da se upisuje i

- **sve dok mu kernel drži lock**, nijedan proces **ne može pristupiti tom baferu**.

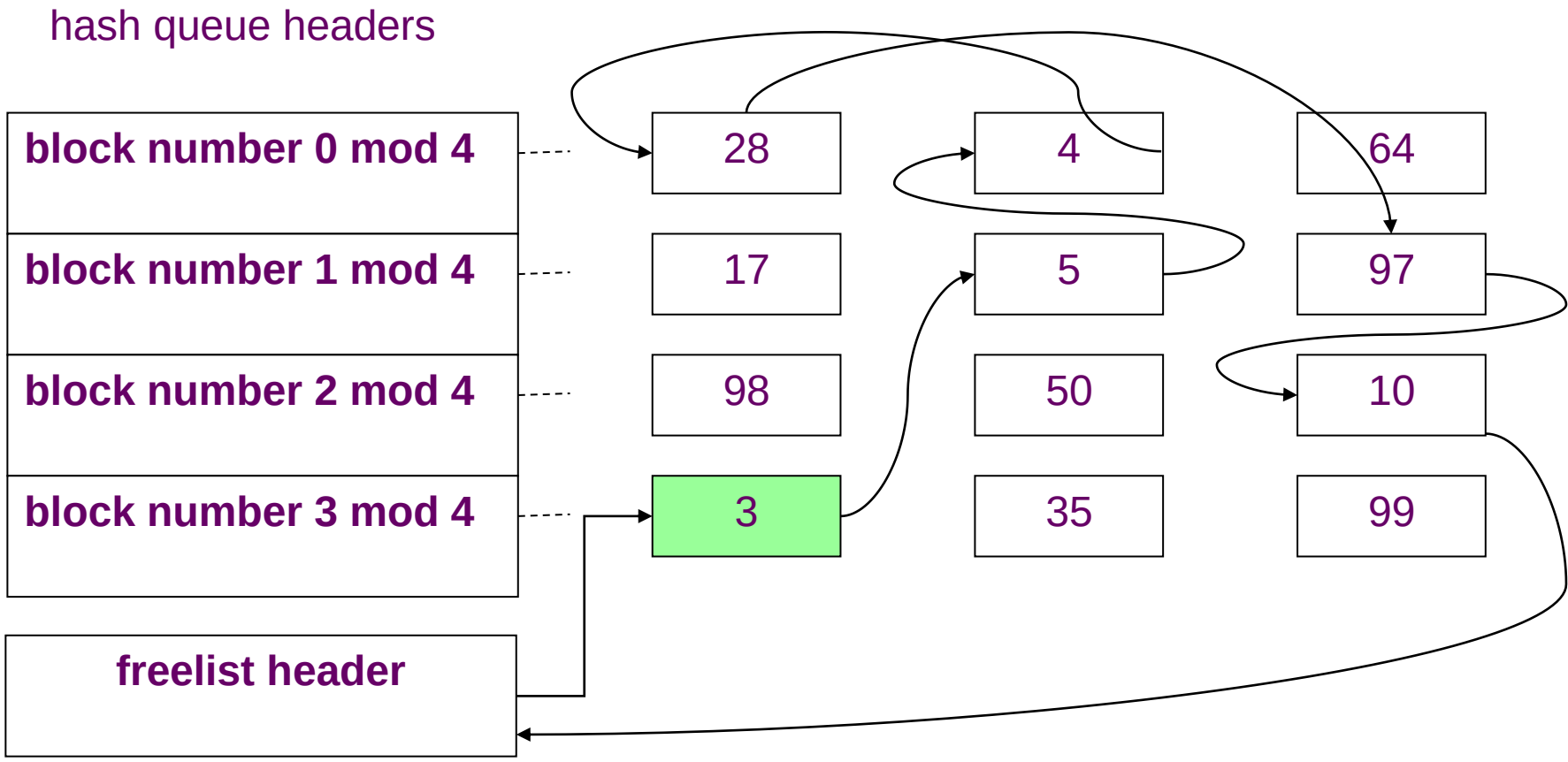
- Kada proces obavi svoje, **bafer se mora osloboditi** preko algoritma **brelse**.

- Uvek **kada se ažurira free lista**, prekidi **moraju biti blokirani**.

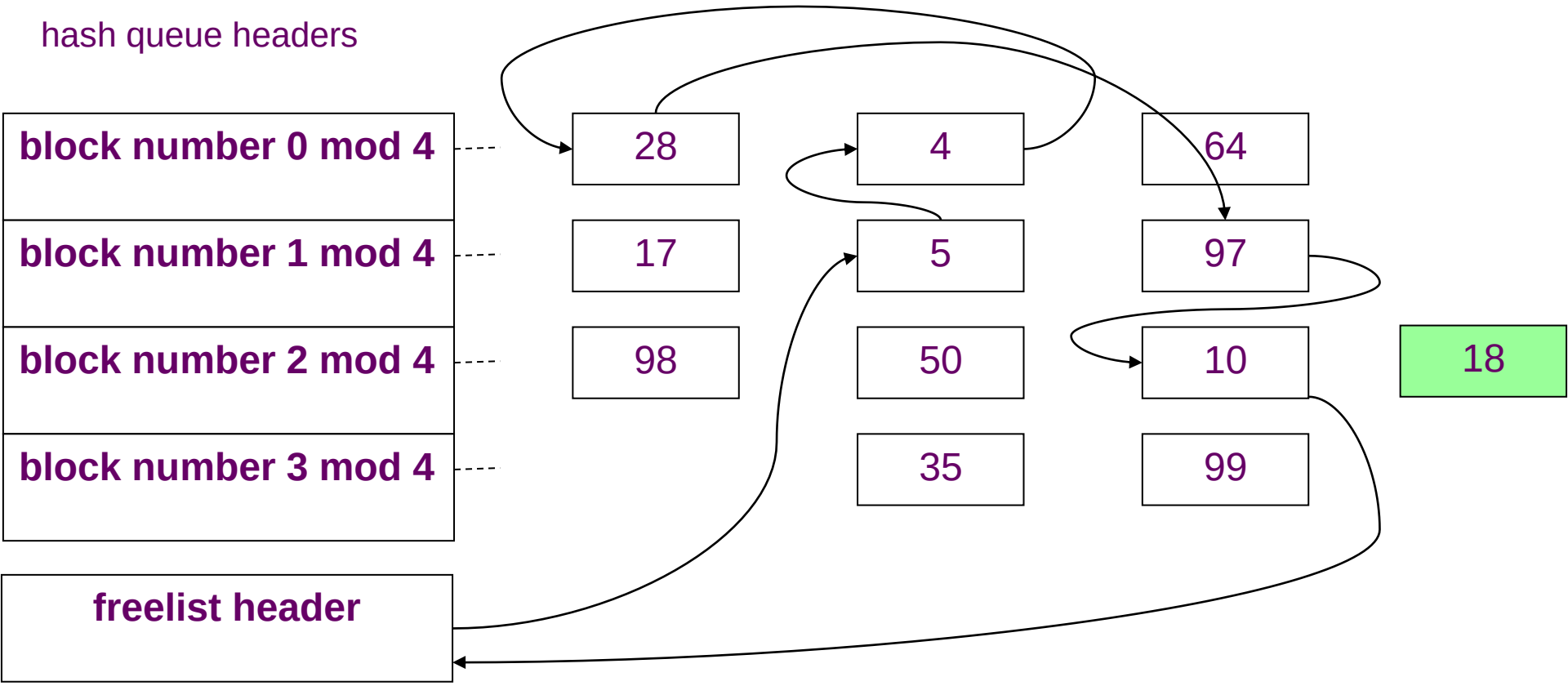
algoritam brelse

- **algoritam brelse**
- **input: locked buffer**
- **output: none**
- {
- **wakeup all procs**: event, **waiting for any buffer** to become free;
- **wakeup all procs**: event, waiting **for this buffer** to become free;
- **raise CPU execution** level **to block interrupts**;
- **if (buffer contents valid and buffer not old)**
- enqueue buffer at the **end of free list**
- else
- enqueue buffer **at** the **beginning of free list**
- lower CPU execution level to **allow interrupts**;
- **unlock(buffer);**
- }/*main**/**

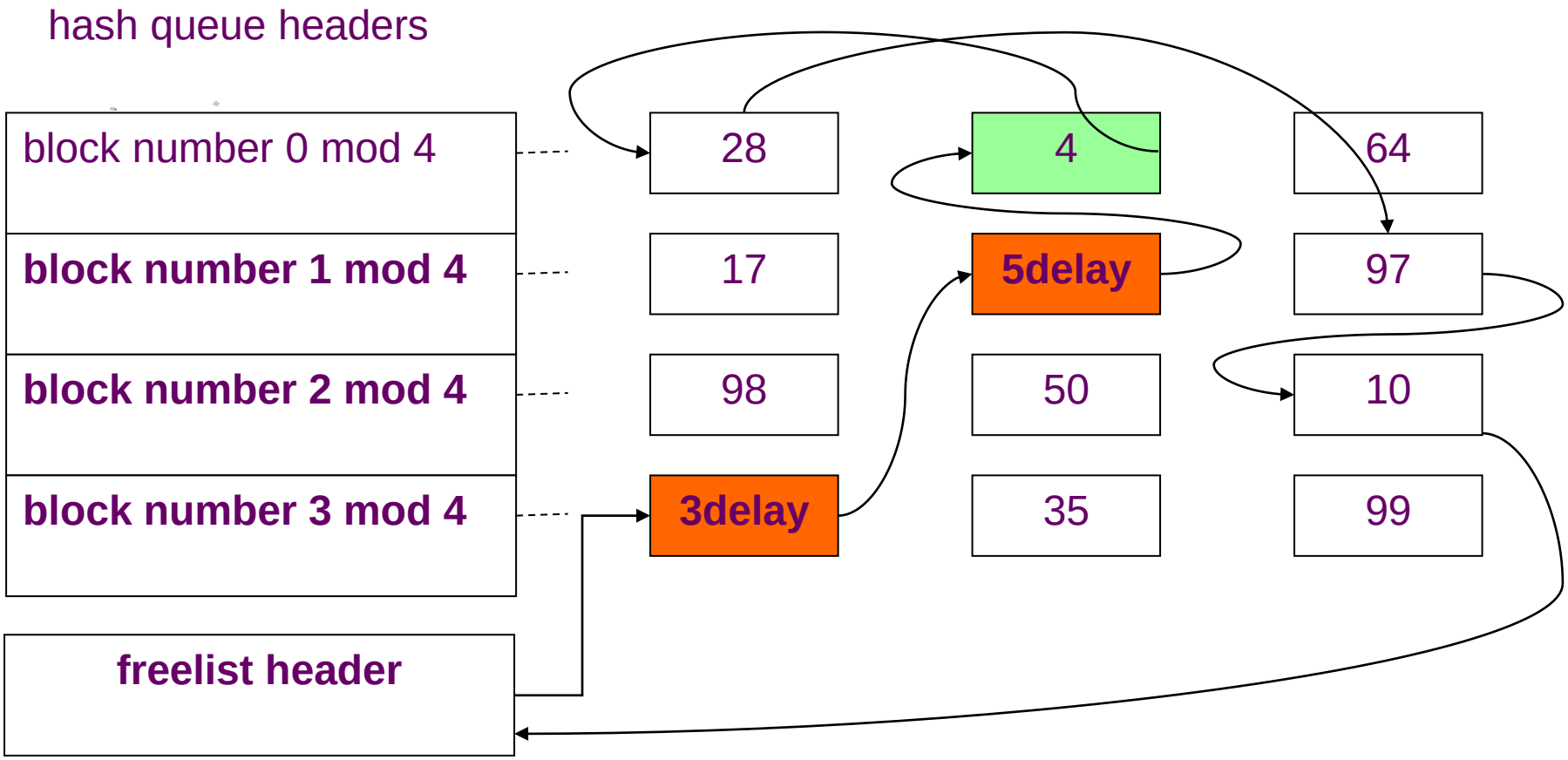
scenario 2. a) Tražimo blok 18, nije u kešu



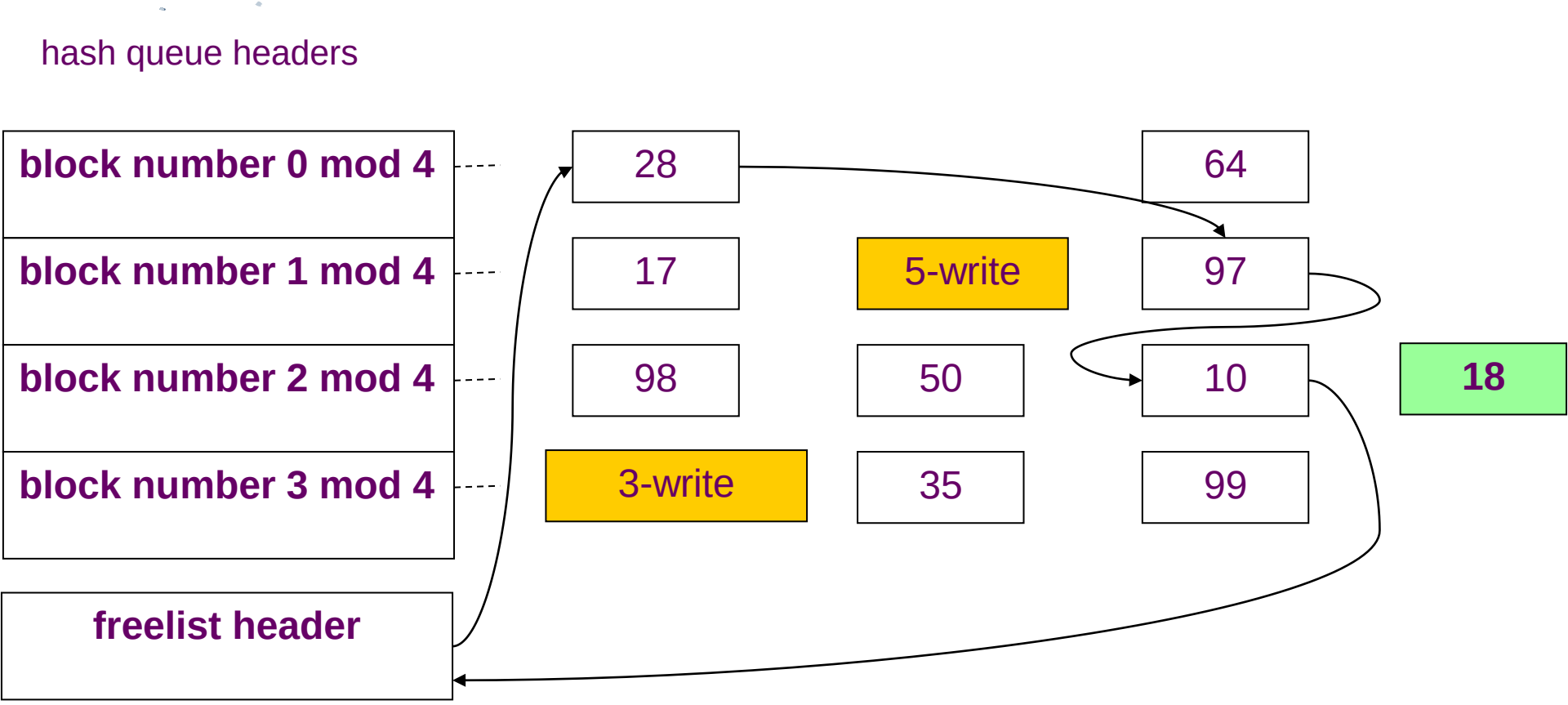
scenario 2. b) Uklanja se prvi blok iz free liste i dodeljuje novom bloku 18



scenario 3. a) Tražimo blok 18, nije u kešu, ali 3 i 5 moraju da se upišu na disk

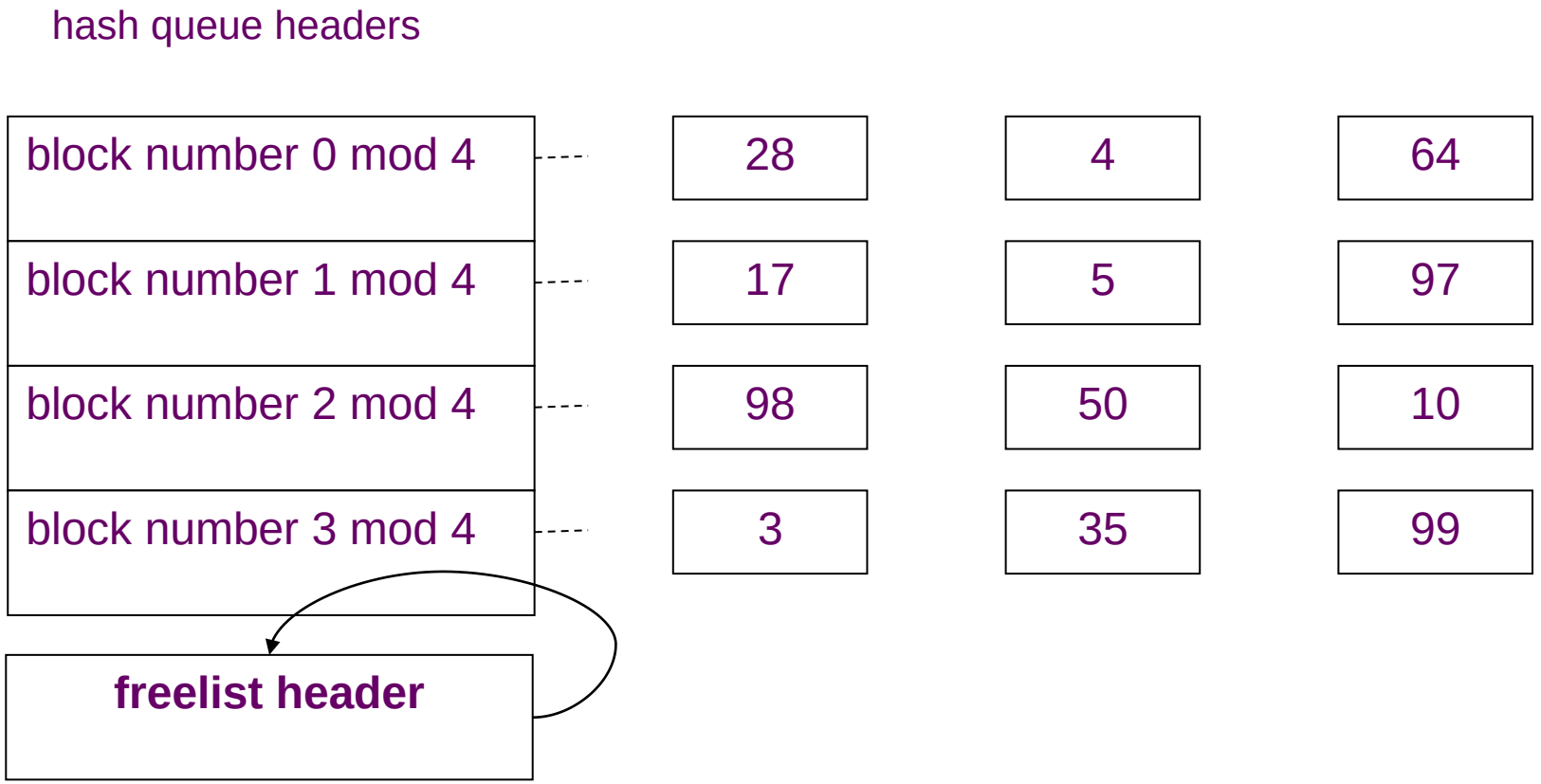


cenario 3. b) Blokovi 3 i 5 se upisuju a prvi slobodni se dodeljuje za 18, a to je 4



scenario 4. a) Tražimo blok 18, nije u kešu, nema ni jednog slobodnog bloka

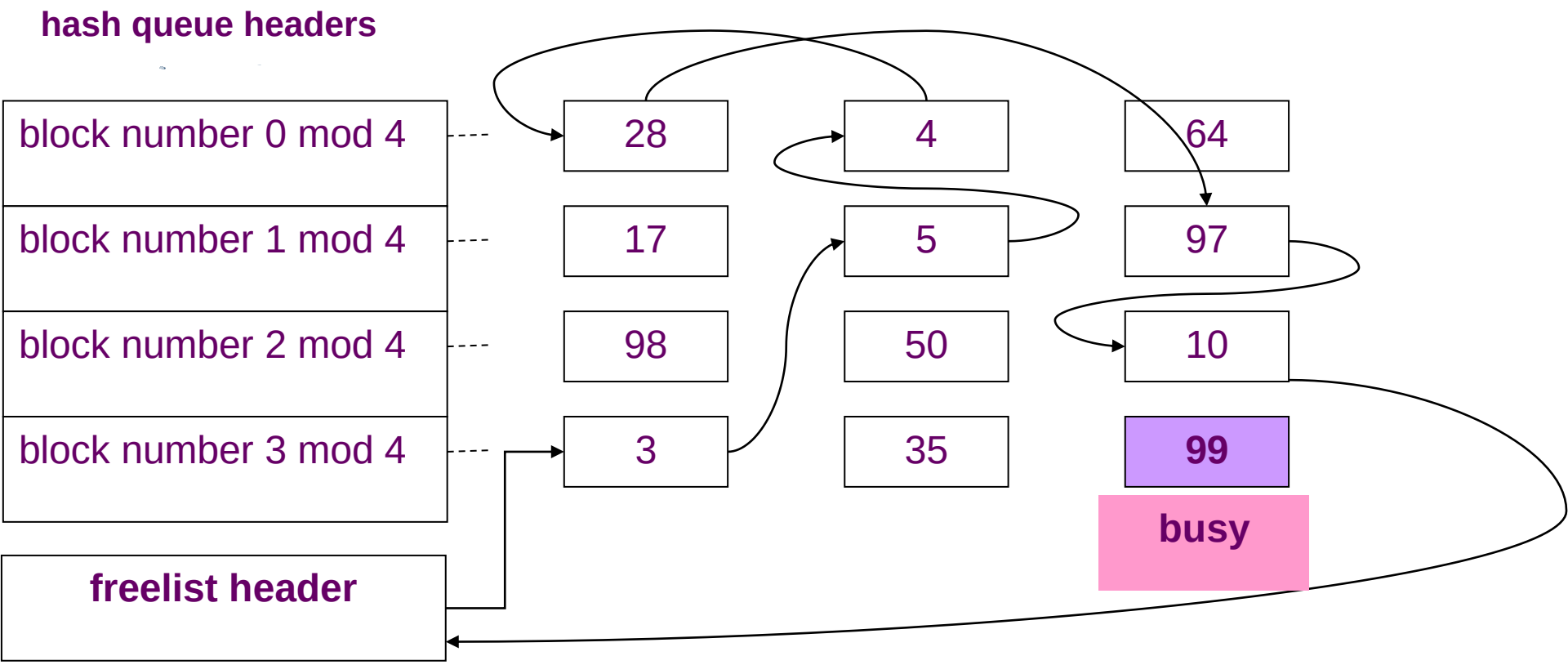
- free list je prazna



- Proces mora otići na spavanje sve dok neko ne obavi algoritam brelse koji će osloboditi bar jedan bafer.
- Čak i da je u kešu hit, proces mora na spavanje.

scenario 5. a) Tražimo blok 99, u kešu je, ali je trenutno zauzet (locked)

- Proces mora da čeka da blok bude oslobođen.



Čitanje i upis kroz bafer bread

- Algoritam za čitanje je bread.
- Kod njega su 2 glavne karaktersitike, a **cache hit** i **cache miss**.

- **algorthm bread()** /* block */
- input: file system block number
- output: buffer containing data
- {
- get buffer for block (algorithm **getblk**);
- if (**buffer data valid**) return (buffer); **/*hit*/**
- else
- **initiate disk read**; **/*miss*/**
- sleep (event disk read complete);
- return (buffer);
- }

read-ahead (breada)

- **Za povećanje performansi koristi se read-ahead tehnika:**

- **algoritam breada** /* block read and read ahead*/
- input:
  - (1) file system block number for immediate read
  - (2) file system block number for asynchronous read
- output: buffer containing data for immediate read
- {
- if (**first block not in** the **cache**) /*cache miss*/
- {
- **get buffer for first block** (algorithm **getblk**);
- if (buffer data not valid) **initiate disk read**;
- }
- if (**second block not in** the **cache**)
- {
- **get buffer for second block** (algorithm **getblk**);
- if (buffer data valid) release buffer (algorithm **brelse**)
- else **initiate disk read**;
- }
-

read-ahead (breada)

- if (first block **was originally** in the cache)
- {
- read first block (algorithm bread);
- return buffer;
- }
- sleep (event first buffer contains valid data)
- return buffer
- }/*main*/
- **miss (2 read=read normal + read ahead)**
- Ako **prvi blok nije u kešu** odvija se **sinhrono disk čitanje** sa sleep-om, a ako je u kešu zadaje je neposredno prosleđivanje bafera.
- Odmah se obavlja **asinhroni read-ahead**, proveri se da li je sledeći sekvencijalni blok u kešu.
- Ako jeste nikom ništa, ali ako nije nađe se blok u kešu (getblk) i ako je dodeljeni bafer prazan, inicira se asinhrono disk čitanje.
- **hit (no disk reading)**
- Ako dodeljeni bafer već ima validne podatke read-ahead se ne izvršava, već se bafer otpušta (to je hit za read-ahead).

upis kroz bafer (bwrite)

- Algoritam za upis je **bwrite**. 2 glavne karaktersitike, cache hit i cache miss.
- **algorthm bwrite()** /* block */
- input: **buffer to be written**
- output: none
- {
- **initiate disk write; /* ovaj I/O možda se obavlja ili sihrono ili delayed */**
- **if(I/O synchronous)**
- {
- sleep (event I/O complete)
- release buffer (algorithm **brelse**);
- }
- else if (**buffer marked** for **delayed write**);
- **mark buffer** to **put** at **head** of **free list**;
- }

upis kroz bafer (bwrite)

- Pretpostavimo da je upis **hit(in_cache)** i kernel daje nalog za upis na disk.
- Taj nalog može biti sinhroni i DW.
- Ako je **sinhroni write**, kernel upisuje blok na disk i čeka da se I/O završi.
- Ako **delayed write**, to se markira za taj bafer i ne vrši se I/O, a bafer se oslobođa.
- Upis se inicira u **getblk** algoritmu po scenariju 3 kada je DW blok u free listi, a dodeljen je novi blok iz free liste.
- Ti DW baferi se iniciraju za write, odnosno **pražnjenje**.
- DW blokovi se potiskuju na početak free liste kako bi se praznili na svaki keš miss.

Prednosti i nedostaci baferskog keša

- **Prednosti:**
  - **Smanjuje disk trafic**
  - DW je izrazita prednost
  - **keš je kernelska memorija** koja je zaštićena i dobro sinhronizovana, **nije podložna programerskim** ili **korisničkim greškama**

- **Nedostaci:**
  - **međutransfer**, sa **diska u keš**, pa u **user buffer**
  - **DW** može dovesti do **gubitka podataka**, što se ublažava sa journaling tehnikom