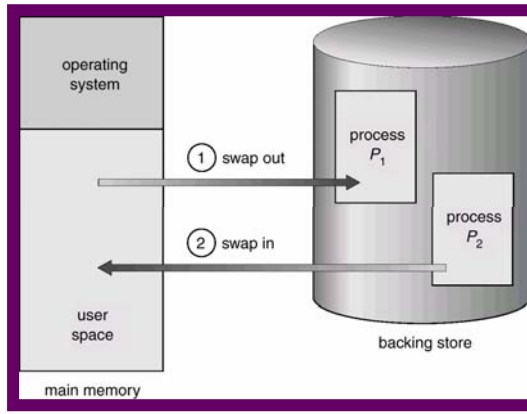
Memory management policies

- **CPU sheduling algorutmi** imaju **snažan uticaj na MMU policije**.
- **Barem deo procesa** se **mora nalaziti u memoriji** da bi **mogao da počne izvršavanje**, CPU ne može izvršavati proces koji je potpuno na swapu.
- Zahvaljući **swap konceptu**,
  - uvek ima **više aktivnih procesa** nego što bi se oni **mogli naći u memoriji ustovremeno**
  - to je jedan od glavnih zadatak MMU
  - da odluči koji će proces da bude **delimično ili potpuno u memoriji a koji na** swap.
- **MMU monitoriše zauzeće memorije**
  - povremeno **neke procese** upisuje na **swap** uređaj
  - **povremeno neke vraća sa swap**-a **u glavnu memoriju**.

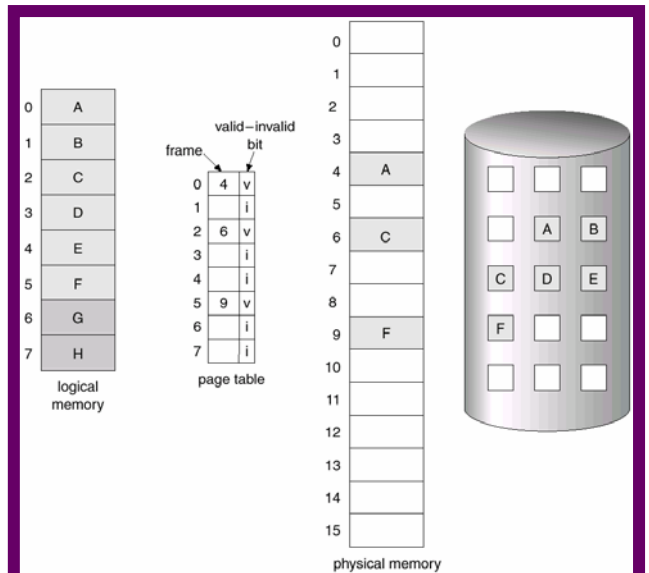
Memory management policies



- **Prvi UNIX sistemi**
  - **transferišu cele procese** između **swap i glavne memorije,**
  - a jedini delimičan transfer može biti **shared text region**,
  - a takva MMU se naziva **swaping**.

- **Prvi UNIX sistemi** imali malu maksimalnu **veličinu procesa 64K**, što je uslovljavala **mala količina RAMa**.

- BSD UNIX počevši od release 4.0 je uveo **demand paging MMU**,
  - gde se **transferišu stranice** memorije **između memorije i swapa**,
  - a **ne celi procesi**,
  - a to je kasnije uveo i UNIX System V.



Memory management policies

- **Paging** je uveo **puno poboljšanja**,
  - **dovoljno** je da **jedna stranica procesa bude u memoriji da on počne da se izvršava**,
  - a sve ostale stranice se dobijaju preko **DP**.

- Prednost koji donosi **DP** je u tome što **omogućava**
  - veliku **fleksibilnost u mapiranju** između **virtuelne i fizičke memorije**
  - dozvoljavajući da **veličina procesa bude veća od fizičke memorije**
  - a i **više procesa** mogu da **stanu** u **ready queue** simultano u memoriji.

- **Swaping policija** je **lakša za realizaciju** i ima **manji sistemski overhead**.

- U ovoj lekciji objasnićemo obe policije.

Swapping

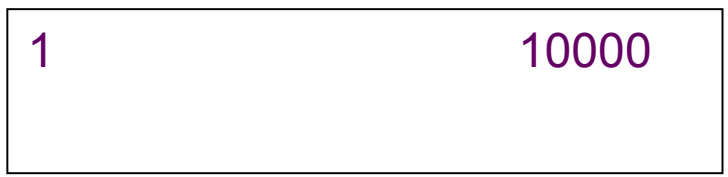
- Swapping algoritam ima **3 funkcionalna dela**:
- **I upravljanje swap uređajem**
- **II swap out** = prebacivanje **celog procesa** iz **memorije** na **swap**
- **III swap in** = **vraćanja celog procesa** sa **swapa** u **memoriju**

Allocation of swap space

- **Swap uređaj** je **block uređaj** u konfigurabilnoj **sekciji diska**.
  - **Dok kernel datotekama,** dodeljuje **jedan blok** u **vremenu**,
  - **swap space** se alocira **u grupama kontinualnih blokova**.
- **Prostor alociran za datoteke** koristi se **statički**, pošto **blok može pripadati datoteci veoma dugo**, pa alokaciona politika mora da bude optimizovana da **smanji fragmentaciju**.
- **Alokacija prostora** u **swapu** je **tranzijenta**, **zavisno** od **trenutnog stanja izvršavanja procesa.**
- Proces koji je na swapu će napustiti swap i osloboditi prostor koji je zauzeo.
- Zbog sporosti disk I/O poželjeno da transferi sa **diska budu veći**, zato je swap prostor za proces alokacija **uvek kontinualna bez obzira na fragmentaciju**, tako da **ceo proces može** da se **transferiše** u swapin u jednom **disk IO**.
- Zato što je **swap alokacija drukčija** u **odnosu na alokaciju** u FS, **data stuktura** za **vođenje free prostora je takođe različita**.
- Kernel podržava **free listu** za FS kao linkovanu listu **free blokova** kojoj se pristupa **preko FS superblocka**.
- Za **swap** se kreira **posebna struktura** u memoriji koja se naziva **mapa**
  - (**incore swapmap**),
  - koja radi na principu **first-fit** alokacije.

Allocation of swap space

- Mapa je polje sa ulazima gde se **svaki ulaz** sastoji od:
  - **adrese alokatibilnog resursa**
  - **broja jedinica** koji su tu **raspoloživi** (free)
- kernel interpretira adresu i jedinicu prema vrhu mape.
- Inicijalno, mapa sadrži jedan ulaz koji ukazuje na adresu i **totalni broj resursa**.
- Na primer, kernel tretira svaku jedinicu swap mape kao grupu disk blokova, i **tretira** adresu kao **blok offset** od početka **swap area**.
- Slika ilustruje inicijalnu **swap mapu**, koja se sastoji od **10.000** blokova = **10MB swap**, koji počinju na adresi 1.



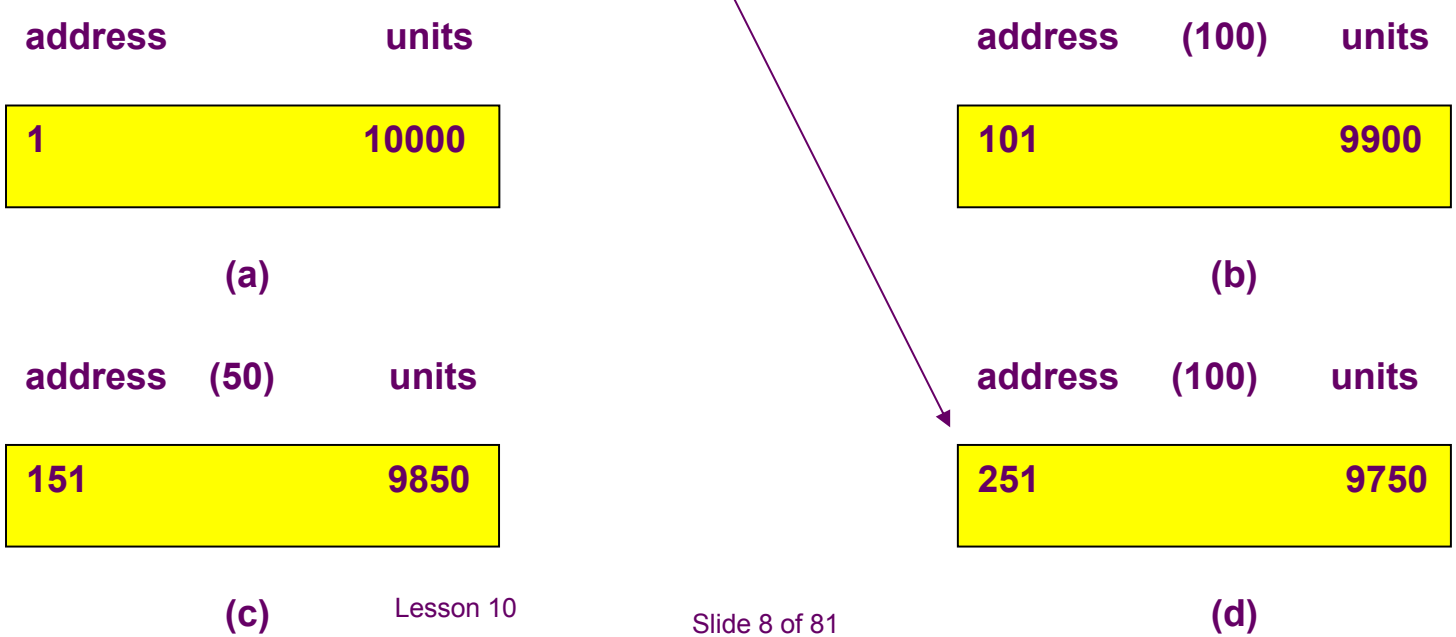
- Kada kernel dodelju i oslobađa swap resurse, ažurira se mapa da reflektuju novi slobodni prostor.

algorithm malloc

- Na slici je dat algoritam malloc za alokaciju prostora iz mape za swap.
- **algorithm malloc** **/* algorithm to allocate map space*/**
- input:
  - (1) map address /*indicate which map to use*/
  - (2) requested number of units
- output: **address if successful**, **0 otherwise**
- {
- **for(every map entry**)
- {
- if(current map **can it fit** requested units)
- {
- **if(requested units == number of units in entry**) **delete entry from map**;
- else
- **adjust start address** of **entry;**
- **return(original adress** of **entry**);
- }
- }
- return(0);
- }

algorithm malloc

- Kernel pretražuje mapu za prvi ulaz koji sadrži dovoljno slobodnog prostora da zadovolji zahtev. Ako zahtev **uzme sve blokove** sa tog ulaza, ulaz se briše iz **mape** i mapa se komprimuje (jedan ulaz manje). Ako je ulaz veći od onoga što se traži, ulaz se modifikuje
- **start address = start address + requested_size**
- **unit = unit – requested**
  - Slika sadži sliku swap mape (a),
  - kada se dodeljuju 100 jedinica (b),
  - 50 (c) i 100 (d).
- Kernel prilagođava swap mapu da prikazuje da je prvih 250 jedinica alocirano i da sada sadrži 9750 jedinica na početnoj adresi 251.



algorithm malloc

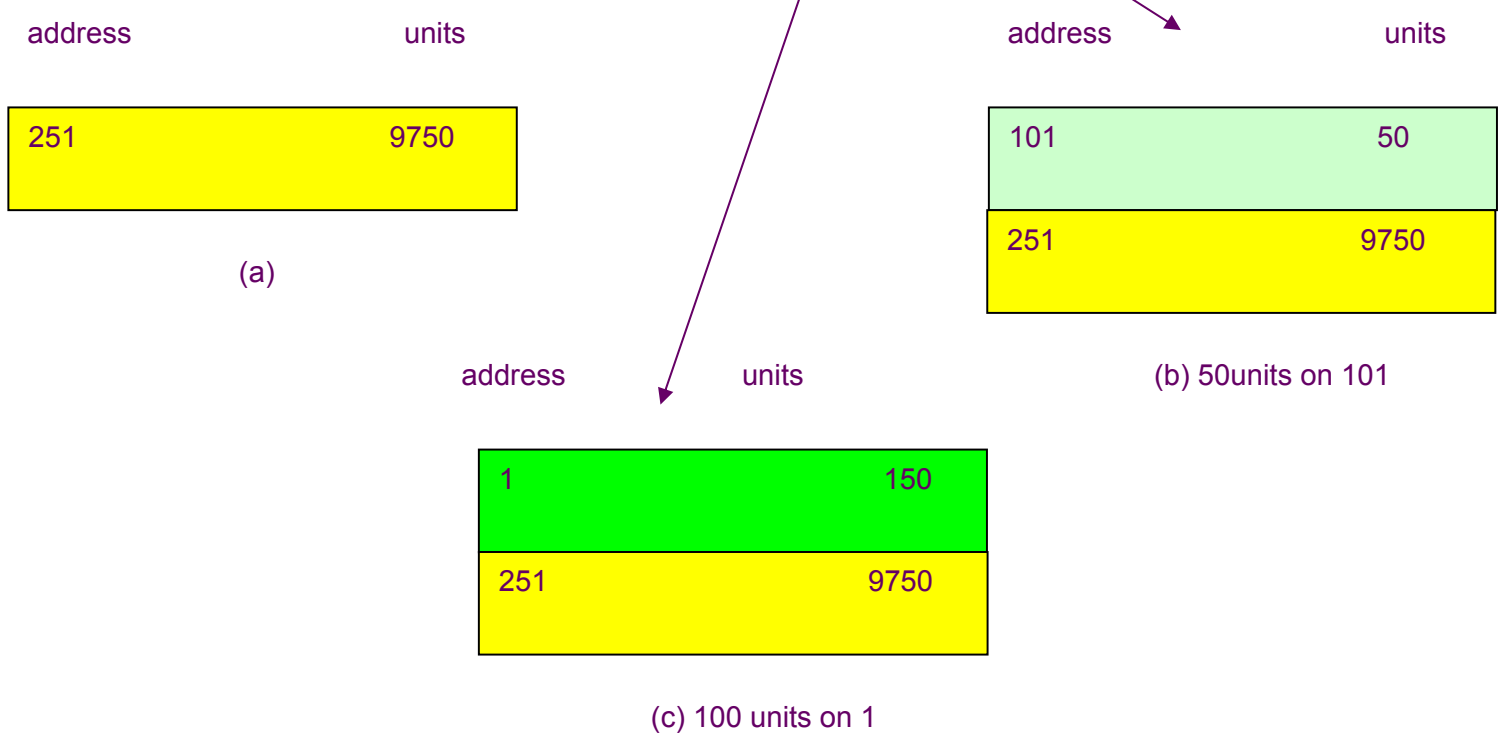
- **Kada se oslobađaju swap resursi**, kernel nalazi njihove pozicije u mapi i tada su sledeće 3 **situacije moguće**:
- 1. **Oslobođeni resursi** kompletno **prave potpunu full šupljinu u mapi** (**sa obe strane**),
  - ona je kontinualna sa ulazima čiji adrese neposredno predhode novoj šupljini i prate je dalje u mapi.
  - U ovom slučaju, kernel kombinuje novu šupljinu sa 2 susedna ulaza i
  - kreira jednu **veliku šupljinu** sa 1-im ulazom u mapi
  - **(broj ulaza u mapi se smanjuje za 1)**

- 2. Oslobođeni resursi **delimično prave šupljinu u mapi** (samo sa jedne strane).
  - To je slučaj kada se nova šupljina naslanja na šupljinu **sa donje strane** (adrese joj prethode)
  - ili šupljina **naslanja na šupljinu sa gornje strane** (adrese joj slede).
  - Podešava se samo taj ulaz sa kim se spaja, a **broj ulaza** u mapi se **ne menja**

- 3. **Oslobađanje resursa** dobija se **šupljina**, **ali ona nije susedna sa nikim**. Kernel ubacuje **novi ulaz u mapu**, koji opisuje novu šupljinu

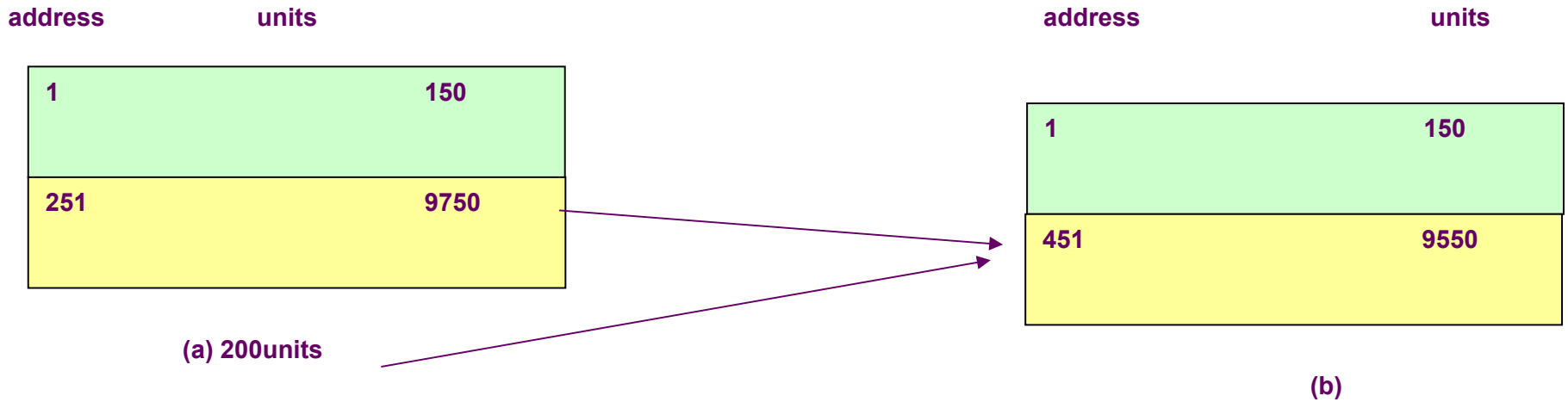
algorithm malloc

- Demonstrirajmo to na prethodnom primeru.
- Ako kenel oslobodi **50 swap jedinica na adresi 101 swapa** , nema dodirivanja šupljina, mapa uvodi novi ulaz **kao** pod (b).
- Ako kernel **oslobodi 100 jedinica na adresi 1**, onda će se **šupljine spojiti sa gornje strane**, pa imamo situaciju kao pod c.



algorithm malloc

- Prepostavimo da sada kernel zahteva **200 units swap space**. Zato što prvi ulaz, sadrži samo 150 jedinica, kernel će zadovoljiti zahtev iz drugog ulaza mape (b)



- Na kraju, ako kernel oslobodi 350 jedinica, koji startuje na 151, to su resursi koji su se dodeljivali separatno, ali kernel će ih osloboditi odjedanput, pa će prvih 300 resursa da spoje šupljinu između prva ulaza, koji će se spojiti u jedan ulaz.

algorithm malloc

- Tradicionalni UNIX koristi **samo jedan swap**, ali na
- **novi UNIX System V dozvoljavaju više swap uređaja**.

- Kernel **bira swap uređaj po RR šemi**,
- tako da obezbeđuje uvek **dovoljno freeswap prostora**,
- administrator može kreirati ili izbaciti swap prostor dinamički.

- Kada se **swap izbacuje**, njegovi **podaci moraju u memoriju**, pa eventualno **na neki drugi swap**, pa se tek onda izbacuju iz swap područje

Swaping process out

- Kernel swapuje (swapout) proces **kada mu nedostaje memorija**, a to može da se **dogodi u sledećim situacijama**:
  - 1**. SC fork** mora alocirati prostor za proces dete
  - 2. **SC brk** povećava veličinu procesa
  - 3. proces postaje veći jer mu je **stack region** porastao
  - kernel želi **da oslobodi prostor** u memoriji **za procese koji su prethodno u swap-out** stanju, da bi ih vratio u memoriju (**swapin**)
- Prvi slučaj je unikatan (fork), jer to je **jedini slučaj** kada se prethodno zauzeta memorija **ne oslobađa** (roditelj drži svoje resurse a traži kompletno to isto za dete)
- Kada **kernel izabere proces** za **swapout**, **kernel dekrementira RC** za svaki **region procesa** i obavlja **swap out** za **sve regione čiji je RC pao na 0**.
- **Kernel alocira prostor** na **swap uređaju** i **lockuje proces** u memoriji (za slučajeve od 1-3), **štiteći proces** od **swapper-**a dok se ne obavi tekuća swap operacija.
- Kernel čuva **swap adresu regiona** u **RT ulazu.**

swap-out

- **Kernel swapuje maksimalnu Qpodataka**
  - **koliko je moguće po jednom I/O direktno** između **user adresnog prostora i diska**
  - preskačući **bafer cache** (**bypass**).
- Ako hadrver ne može da prebaci više stranica **odjedanput,** kernel u **iteracijama** prebacuje stranicu po stranicu. Sama tehnika transfera zavisi od disk kontrolera, od memorijske organizacije itd.
- Na primer kod **paging sistema**, **virtuelne adrese su kontinualne**, ali **fizičke** adrese su **diskontinualne**, **što će uzrokovati više disk zahteva**.
- **Swapper** čeka da se **kompletira disk I/O** pre nego što otpočne **novi**.
- **Nije neophodno** da **kernel prebaci kompletan virtuleni adresi prostor** procesa na swap.
- Kopiraju se samo **dodeljene fizičke adrese**, a **nedodeljene virtuelne adrese** se **ne kopiraju**.
- Kada kernel **vraća proces u memoriju**,
  - kernel zna **virtulenu adresnu mapu procesa**,
  - tako da može da ponovo dodeli proces na **korektne virtualne adrese**.

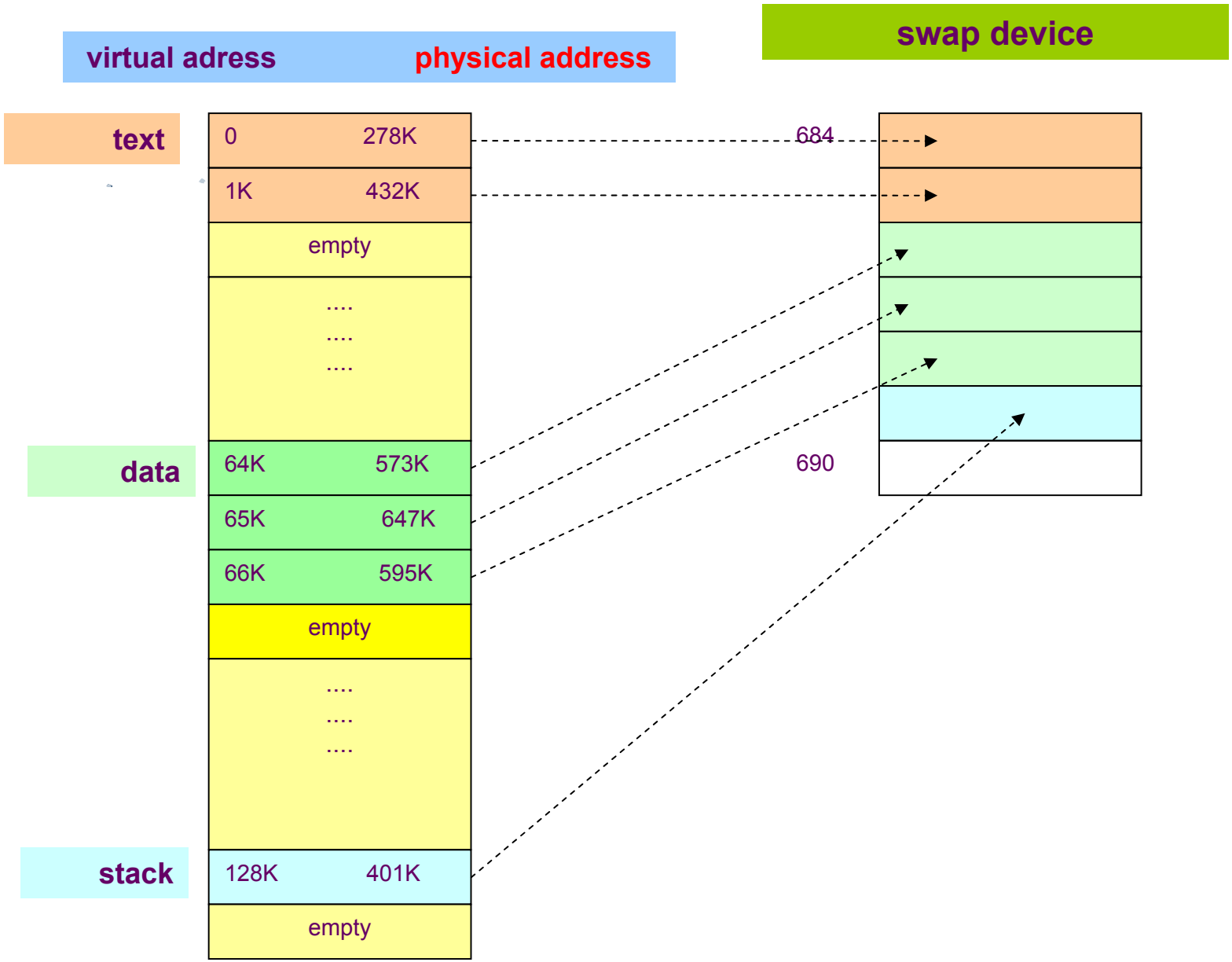
swap out example

- **Na slici je dat primer** mapiranja **incore image procesa** na **swap uređaj**.

- Proces sadrži **3 regiona za text, data i stack**.
- **Text region** se završava na **virtulenoj adresi 2K**, a
- **data region** startuje na **virtulenoj adresi 64K**,
- ostavljajući **gap od 62K** u virtuelnom adresnom prostoru.

- Kada kernel swapuje proces
  - on swapuje stranice na adresi **0, 1K, 64K, 65K, 66K i 128K**,
  - **gapove u virtuelnom adresnom prostoru ne swapuje**
    - (62K između end(text) i start(data) i
    - 61K između end(data) i start(stack)),
  - ali se **swap prostor puni kontinualno**.

layout of virtual address



swap in example

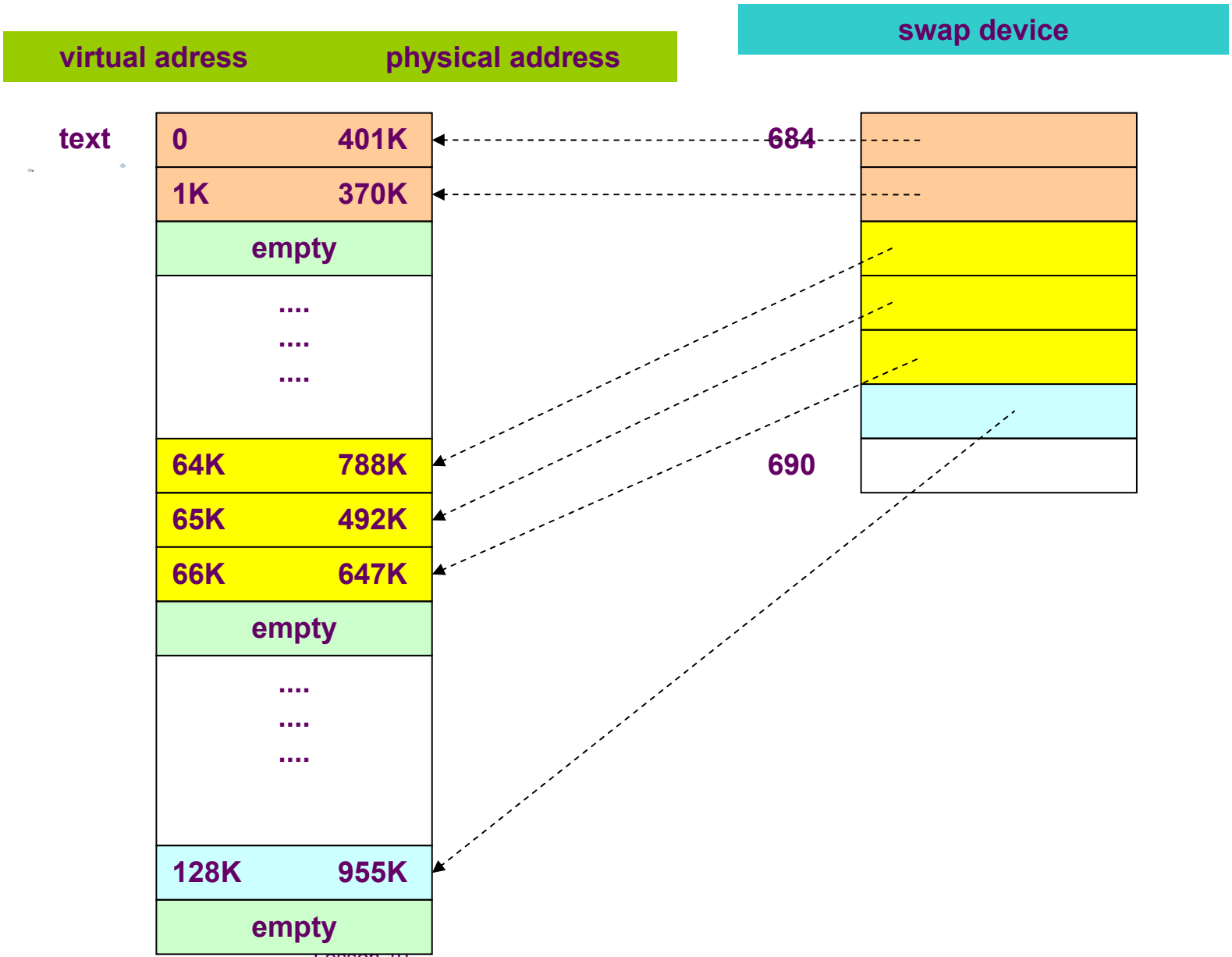
- Kada kernel obavlja **swapin**, **zna da proces ima 62K šupljinu** preko procesove memorijske mape i na taj način dodeljuje fizičku memoriju u saglasnosti sa mapom.
- Taj slučaj je prikazan na slici.
- Zapažamo da **fizičke adrese pre i posle swapinga nisu iste**, ali proces se **bavi virtuelnim adresama**, tako da se **user-level kontekst ništa nije promenio**, jer je **virtuleni prostor ostao isti**.

- Teorijski, i ostatak memorijskog prostora okupiranog procesom kao što je **u-area, kernelski stack** je **poželjno da se swapuju takođe**, mada kernel može privremeno da lockuje region u memoriji dok se obavlja senzitivna operacija.

- Praktično, kernel **ne swapuje u-area** ako **u-area** sadrži **translacione adrese za procese**.

- Različite implementacije takođe diktiraju da li proces može **swapovati samog sebe** ili mora da **zahteva od drugog procesa da ga swapuje**

layout of virtual address



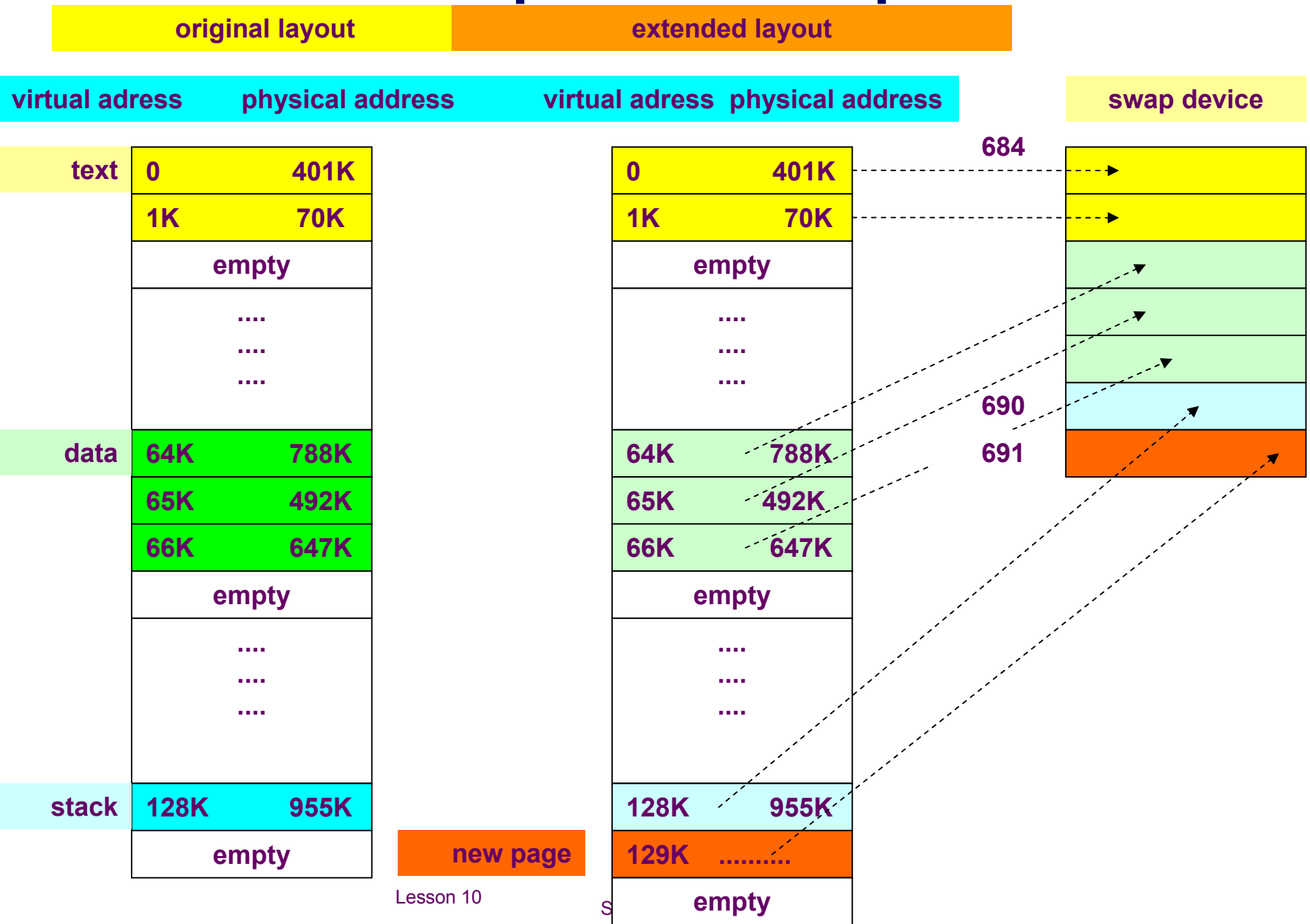
fork swap

- **Opis forka podrazumeva** da **roditeljski proces ima dovoljno memorije** da **napravi dete** (**child context**).
- U suprotnom,
- **kernel swapuje ceo proces-roditelj, ali……**
- **bez oslobađanja memorije** koju **proces zauzima**
- (to je kopija roditelja na swapu, a dete nije dobilo svoju memoriju).
- Kada je **swap kompletan**, **dete proces postoji na swapu**,
- roditelj ga postavlja u **ready to run in swap**,
- a vraća se **user mod**.
- **Swaper** može pokupiti dete (**swapin**) u stanje **3<ready to run>**,
- pa kad ga **scheduler izabere**,
- dete će **okončati svoj deo fork**a i
- **vratiti** se u **user mod.**

expansion swap

- Ako **proces zahteva više fizičke memorije**
  - od one koja mu je **trenutno alocirana**
  - (rezultat **rasta user stacka** ili obavi **brk SC**),
  - **a** to **nije trenutno raspoloživo** u **fizičkoj memoriji**,
  - **kernel obavlja proširenje swapa za proces**.
- **Kernel rezerviše mesto u swapu da sadrži**
  - **kompletan prostor procesa**
  - **uključujući** i **proširenje** koje se zahteva,
  - tako što se **uspostavi korespodencija između nove virtuelne adrese** i **swapa**,
  - a **novoj virtuelnoj adresi se ne dodeljuje fizička adresa** jer **nije trenutno raspoloživa.**
- Na kraju, **ceo se proces swapuje**, a **novi prostor se napuni nulama**.
- **Swapin** procesa će kasnije **dodeliti procesu fizičku adresu za novi prostor**, što je prikazano na slici.

expansion swap



swapping process in

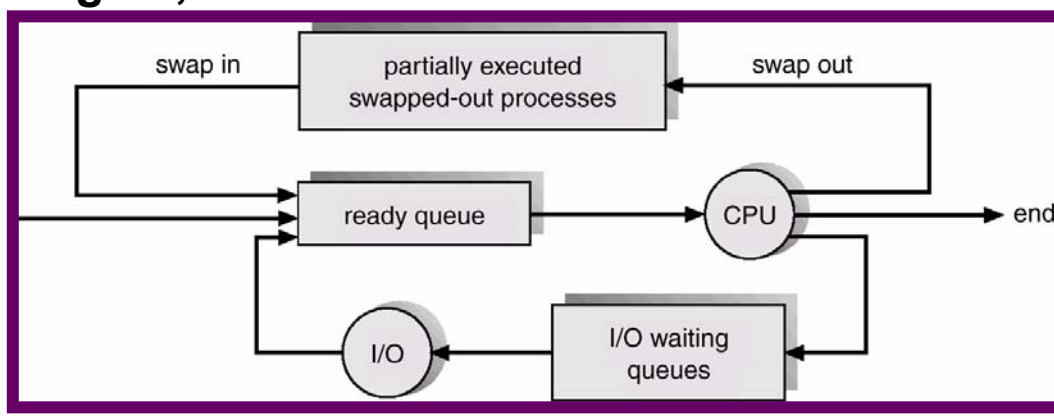
- **Proces 0**, **swapper**, je **jedini proces** koji obavlja **swapin** procesa sa swap uređaja u memoriju.
- Kada se **UNIX podigne**,
  - **swapper odlazi u beskonačnu petlju** i
  - njegov jedini posao je da swapuje (out, in).
- **swapper ide na spavanje** ako **nema za njega posla**
  - (nema procesa za swapin ili swapout).
- **kernel** bira **swapper** kao i **sve druge procese**,
  - **ali to je proces koji se izvršava isključivo u kernelskom modu**.
  - **swapper** ne koristi SC, ali koristi **interne kernelske finkcije**.
- **clock handler** meri **vreme koje svaki proces provodi**
  - **u memoriju**
  - **na swapu.**
- Kada se **swapper probudi**,
  - on **proverava sve procese** koji su **stanju (ready to run, swapped**)
  - **bira jedan** koji je **najduže swapovan**.

swapping process in

- **Ako ima dovoljno memorije** za proces, obavi se **swapin**:
  - **alocira fizičku memoriju**
  - **čita proces sa swap** uređajai
  - **oslobađa swap**.
- Posle uspešnog swapin, swapper **ide na ostale procese stanju** (ready to run, swapped) i obavlja swapin.
- **Jedna** od **sledećih sitacija može da mu se dogodi**:
  - **nema procesa u stanju** (**ready to run, swapped**).
    - **Swapper odlazi na spavanje**, dok se ne pojavi proces na swap uređaju
  - **swapper nalazi proces u stanju** (ready to run, swapped) **ali nema dovoljno fizičke memorije za swapin.**
    - Tada **swapper** pokušava da obavi **swapout** za neki drugi proces **iz stanja 3**,
    - **ako uspe**, pokušava **failed swapin proces**

algorithm swapper

- **algorithm swapper /* swap in, swap out process, swap out other processes to make room */**
  - input: none
  - output: none
- {
- **loop**:
- **for(all swapped out process that** are **ready to run**)
- **pick a process swapped out longest;**
- **if(no such process**)
- {
- **sleep(event must swap in);**
- goto loop;
- }
- if**(enough room** in **main memory for process**)
- {
- **swapin process**;
- goto **loop**;
- }



algorithm swapper

- **/* loop 2: revised algorithm */**
- **loop2:**
- **for(all processes loaded in main memory**, **not zombie** and **not locked in memory)**
- {
- **if(there is** a **sleeping process**)
- **choose process** such that **priority + residence time** is **numerically highest**;

- else **/*no sleeping process*/**
- **choose process** such that **priority + residence time is numerically highest;**
- }

- **if(chosen proces not sleeping** or **residency requirements not satisfied**) **sleep (event must swap process in)**
- else
- **swap out process**;
- goto **loop2**;
- }

algorithm swapper

- **Ako swapper mora** da **obavi swapout**, **on ispituje sve procese u memoriji**,
  - **Zombie procese se ne swapuju**, zato što **ne zauzimaju fizičku memoriji**,
  - **procesi locked u memoriji**,
  - procesi koji obavljaju **region operacije** se takođe ne swapuju.
- **Kernel radi radi swapout za uspavane** procese radije nego (**ready to run**), zato što oni imaju veliku šansu da budu izabrani od schedullera.
  - **Izbor uspavanog procesa** za swapout se obavlja na **bazi prioriteta i vremena koje je proces proveo u memoriji.**
  - **Ako nema uspavanih procesa** u **memoriji**, proces se bira između (ready to run) procesa na bazi **nice vrednosti** i **vremena koje je proces proveo u memoriji**.
- Uvodi se **UNIX 2-second-rule**:
- Proces <**ready to run> mora biti u memoriji barem 2 sekunde**, **pre nego što doživi swap out**,
- a takođe **nema ni swapin za proces** koji **nije na swapu bar 2 sekunde**.
- **Ako nijedan proces ne ispunjava uslove** gornje za **swapout**, **swapper odlazi na spavanje na događaj** (**want a swap process into memory but cannot find room for it**).
  - **Clock će buditi swappper** na **svaku sekundu**.
  - **Pri svakom uspavljivanju drugog procesa kernel će probuditi swapper**.

2-second rule

- Na **slici je dato 5 procesa**, i **vreme provedeno u memoriji i na swapu**.
- **Pretpostavimo** da su svi procesi **CPUbound** i da **ne obavljaju ni jedan SC**, pa se CSw dešavaju po **RR=1sec**, ali tada se uvek izvršava i visoko prioritetni swapper.
- **Pretpostavimo** da su **procesi iste veličine** i da **sistem ima mesta za samo 2 procesa** u memorji.
- Inicijalno procesi **A i B su** u memoriji, dok su **C, D i E na swapu**.
- Swapper **ne može swapovati** nijedan proces za prve 2 sekunde jer nema memorije i nijedan proces nije zadovoljio 2sec pravilo.
- Ali posle 2 sec, swapper obavi swap out za A i B, a obavi swap in za C i D (E ne može nema mesta) i počne da izvršava C.
- Posle 3će sekunde, E bi mogao da ima swapin, ali C i D nisu proveli 2sec u memoriji.
- Tek u 4toj sekundi,
  - **swapper obavi swapout za C i D**,
  - a obavi **swapin za E i A.**

time	Proc A	Proc B	Proc C	Proc D	Proc E
0	0	0	swap out	swap out	swap out
	runs		0	0	0
1	1	1	1	1	1
		runs			
2	2	2	2	2	2
	swap out	swap out	swap in	swap in	
	0	0	0	0	
			runs		
3	1	1	1	1	3
				runs	
4	2	2	2	2	4
	swap in		swap out	swap out	swap in
	0		0	0	0
					runs
5	1	3	1	1	1
	runs				
6	2	4	2	2	2
	swap out	swap in	swap in		swap out
	0	0	0		0
		runs			

swapper discussion

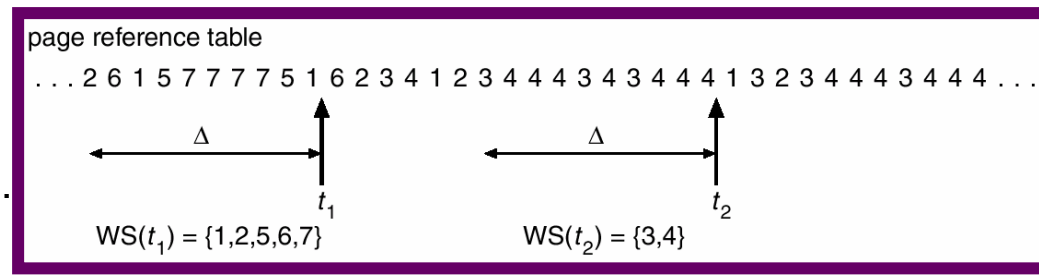
- Za **swapin**, s**wapper** bira procese
  - **na bazi vremena provedenog od trenutka kad je proces doživeo swapout**,
  - a **drugi kriterijum je prioritet**, jer će takvi procesi imati bolju šansu da se izvrše uskoro.
- Algoritam za izbor **swapout** procesa za (**make a room**) je kompleksniji.
- **Prvo**, **swapper** će **obaviti swapout** procesa na **bazi prioriteta**, **memory residence time**, i **nice vrednosti**.
  - **Mada swapper obavlja swapout procesa** da bi **napravio mesto za swapin proces**,
  - može da obavi swapout procesa ali koji ne obezbeđje dovoljno memorije za **dolazeće incomming procese.**
  - Na primer, swapper pokušava da obavi swapin za proces koji **okupira 1MB** a sistem nema dovoljno free memory, a **procesi u memoriji su mali**, tada je **alternativa** da se obavi **swap out za grupu procesa** dok se ne obavi zahtev.
  - U nekim implementacijama, swapper pokušava da obavi swapout više manjih procesa da bi napravio mesto za veći proces, a tek onda radi izbor za **swapin**
- **Drugo**, **ako swapper spava**
  - zato **što nije našao dovoljno memorije** da obavi **swapin** procesa,
  - **kad se probudi, on će ponovo pretraživati** swap **za izbor swapin** procesa
  - iako je u **prethodnoj iteraciji izabrao jedan**.
  - **Razlog je** što drugi procesi u swapu mogu da se **probude** i da **budu bolji za swapin** nego **prethodno izabrani proces**.
  - Neka mala šansa da da prethodno izabrani proces bude ponovo izabran **ipak postoji**.

Demand Paging

- Mašine čije su
  - memorijske **arhitekture bazirane na stranicama**
  - čiji CPU imaju **restartable instrukcije**
    - (u delu instrukcije se dogodi PF, a CPU restartuje instrukciju posle PF handlera)
- **mogu podržavati kernel DP algoritam**, koji swapuje **pages** između memorije i swap uređaja.
- **DP oslobađa procese limitacija** koje nameće **veličina fizičke memorije**.
  - Na primer mašina može sadržavati 1 ili 2MB fizičke memorije a da izvršava procese veličine 4 ili 5MB.
- Kernel ispoljava ograničenje u pogledu virtualne veličine procesa, zavisno od veličine virtuelne memorije koju sistem podržava. Pošto **proces na primer ne može** da **stane u fizičku memoriju**, kernel će **puniti dinamički delove procesa** u memoriju i izvršavati proces bez obzira što **neki njegovi delovi** nisu u **memoriji nego na swap uređaju**.
- DP je **transparentno** za korisnika, osim za **maksimalnu virtuelnu veličinu** procesa.

Princip lokalnosti

- Procesi teže da **izvršavaju instrukcije** u **malim porcijama njihovog text prostora** kao što je **programska petlja** i često pozivani **potprogrami** i
- njihove **data reference teže** da se **grupišu** u **male podskupove data prostora** procesa,
- a to je poznato pod nazivom **princip lokalnosti**.
- **Denning** je uveo pojam **working set procesa,**
  - **koji obuhvata skup svih stranica** kojima je proces **pristupao**
  - **u poslednjih n memorijskih referenci**,
  - **pri čemu** se **n zove window working seta**.
- Kako je **working set deo procesa**, **više procesa mogu simultano biti u memoriji** nego u swaping sistemu, što povećava performanse.
- Kada se pojavi adresa za page koja nije u memoriji,
  - **PF** se dogodi,
  - PF **handler reguliše PF**,
  - working set se obnovi
  - a stranica se pročita u memoriju.



Princip lokalnosti

- **Slika pokazuje sekvencu page referenci** koju bi proces mogao napraviti, prikazuju se working set za različite **window**, a primenjuje se LRU za page replacement..
- Kako se proces izvršava, menja se **njegov working set**, zavisno od **memorijskih referenci**, širi window obuhvata veći working set, što znači da neće praviti PF greške tako često.

- Nije praktično koristiti **pure working set model**,
  - jer je preskupo pratiti **poredak page referenci.**

- Umesto toga, **sistem aproksimira working set model,**
  - **setujući reference bit**,
  - **svaki** put **kada se sistem obrati stranici**
  - **sample-ujući sistem periodično**.

- Ako je page bila **skoro referencirana**
  - **ona je u working setu**,
- u protivnom ona je **zastarela "ages"** i
  - **može se po potrebi izbaciti iz memorije**

setu, u protivnom ona je zastarela "ages" i moze sa po potrebi izbaciti iz memorije.

	working sets/window sizes			
page references	2	3	4	5
24	24	24	24	24
15	15 24	15 24	15 24	15 24
18	18 15	18 15 24	18 15 24	18 15 24
23	23 18	23 18 15	23 18 15	23 18 15
24	24 23	24 23 18	23 18 15 24	23 18 15 24
17	17 24	17 24 23	17 24 23 18	17 24 23 18 15
18	18 17	18 17 24	....	...
24	24 18	....	....	...
18	18 24	....	....	...
17	17 18	....	....	...
17	17	....	....	...
15	15 17	15 17 18	15 17 18 24	...
24	24 15	24 15 17	....	...
17	17 24	....	....	...
24	24 17	....	....	...
18	18 24	18 24 17	....	...

PF

- Kada proces pristupa stranici koja nije iz **working seta**, to će izazvati **validity PF.**
- Kernel **suspenduje proces** sve dok se stranica **ne dovede u memoriju**.

- Kada se page dovede u memoriju,
- proces **restartuje instrukciju koja je izazvala PF**,
- tako da **implementacija paginga ima 2 dela**:
  - **swapout retko korišćenih stranica** i
  - **rukovanje PF (swapin faulted page)**.

- Ovo uglavnom važi za sve paging sisteme,
- a sada ćemo opisati **specifičnosti**
- **za UNIX System V.**

Data structure for demand paging

- Kernel sadrži **4 glavne data strukture** za podršku **low-level memory management** funkcija i DP:
  - **PageTable ulazi**
  - **disk blok deskriptor**
  - **page frame data tabela (pfdata)**
  - **swap use tabela**
- **Kernel alocira** prostor za **pfdata samo jedanput** kad se podigne sistem, ali druge strukture se kreiraju dinamički.

- Podsetimo se da **region sadrži page tabele** za pristup fizičkoj memoriji.
- Svaki **ulaz PT** sadrži:
  - **fizičku adresu stranice**
  - **zaštitne bite** koji pokazuju da li proces može da čita, piše ili izvršava stranicu i sledeća polja koja podržavaju **DP**:
    - **valid**
    - **reference**
    - **modify**
    - **copy on write**
    - **age**

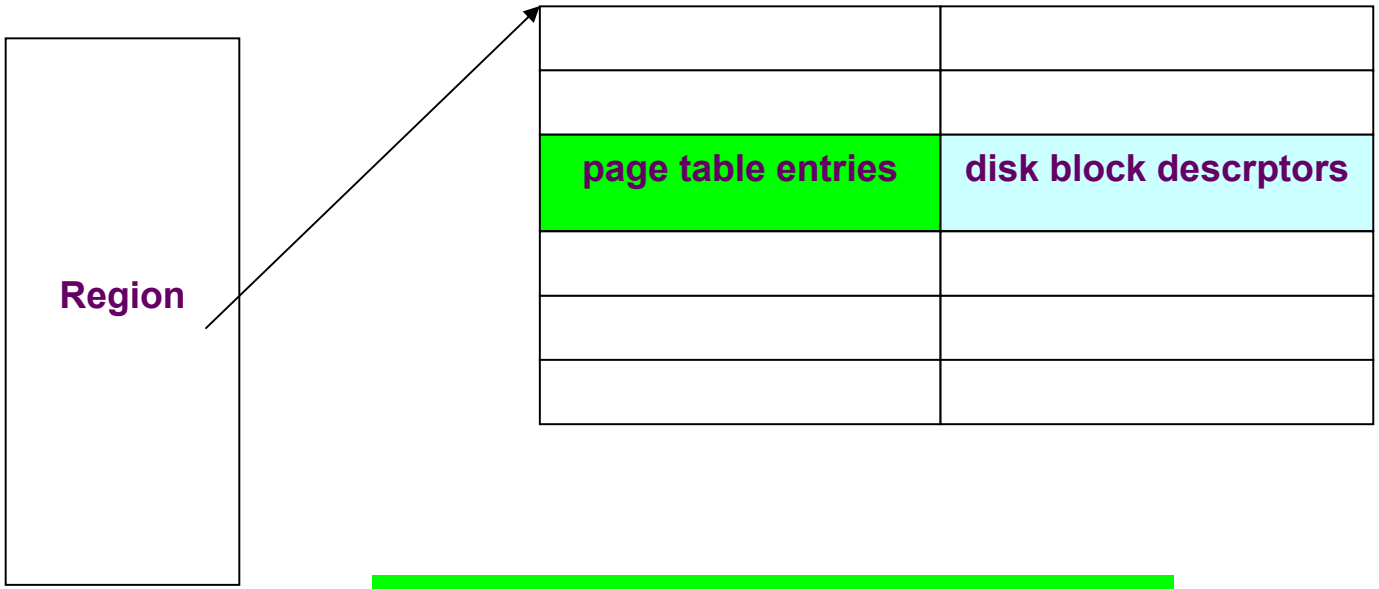
data structure for demand paging

- **valid bit**
  - Kernel postavlja **valid bit** kada je sadržina stranice legalna,
  - ali ako je **valid bit =0**, to ne znači da je referenca na stranicu obavezno ilegalna.
- **reference bit**
  - pokazuje da li je proces **skoro referencirao stranicu**
- **modify bit**
  - ukazuje da je proces **modifikovao sadržaj stranice.**
- **copy on write bit**
  - se koristi u fork SC, i **pokazuje da kernel mora da formira novu kopiju stranice**, kada proces **pokušava da je promeni**.
- **age bits**
  - koji pokazuje **koliko dugo** je **stranica bila član working set procesa**.

- **kernel** manipuliše **bitom validnosti, copy on write, i age bitima**,

- **hardver setuje reference** i **modify bitove u PT ulazu**.

data structure for demand paging



page table entry

page (physical) address	age	Cp/W	mod	ref	val	prot

disk block descriptor

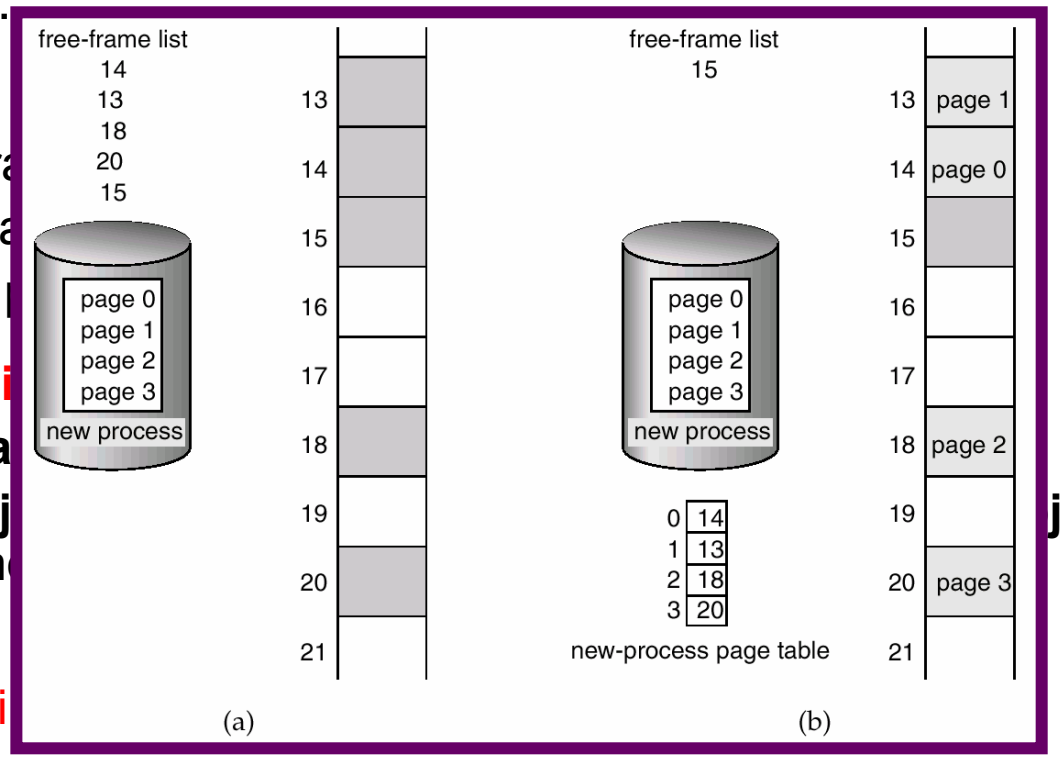
swap device	block number	type(swap, file, fill 0, demand fill)

PT entry + disk block descriptor

- Svaki **ulaz u PT** ima **pridruženi disk blok deskriptor**,
- **dbd** opisuje **swap kopiju virtuelne stranice**.
- Procesi **koji dele** region **pristupaju zajedničkim ulazima u PT i disk blok deskriptorima.**
- **Sadržaj virtuelne stranice** je ili u **na swapu** ili u **egzekutibilnoj datoteci** ili **nije u swapu-disku**.
  - Ako je **stranica na swapu**, **disk blok deskriptor** sadrži
    - **device number** koji **identifikuje swap na disku**
    - **blok number** na swapu koji sadrži stranicu.
  - Ako je **stranica u exe datoteci**, disk blok deskriptor sadrži
    - **logički blok datoteke** koji sadrži stranicu,
    - a **kernel može** lako da **konvertuje taj broj** u **disk adresu**.
- **Disk blok deskriptor** sadrži **setovanje za 2 specijalna uslova** za vreme **exec SC**:
  - **"demand fill"**
  - **"demand zero"**

pfdata (page-frame) table

- **pfdata** tabela opisuje **fizičku memoriju** svake stranice, a tabela se indeksira po broju stranice:
- **polja u pfdata ulazu su**:
  - **page state**,
    - pokazuje da li je str
    - koji je DMA kanal za
    - da li stranica može
  - **RC = broj procesa koji**
    - **RC jednak** broju **va**
    - **Mada se ne očekuj** **dele page**, što ćem **sistemu.**
  - **logički uređaj** (**swap** ili
  - **pointere** na **druge ulaze** u **pfdata tabeli**
    - na **listu slobodnih stranica** i
    - **hash queue stranica**

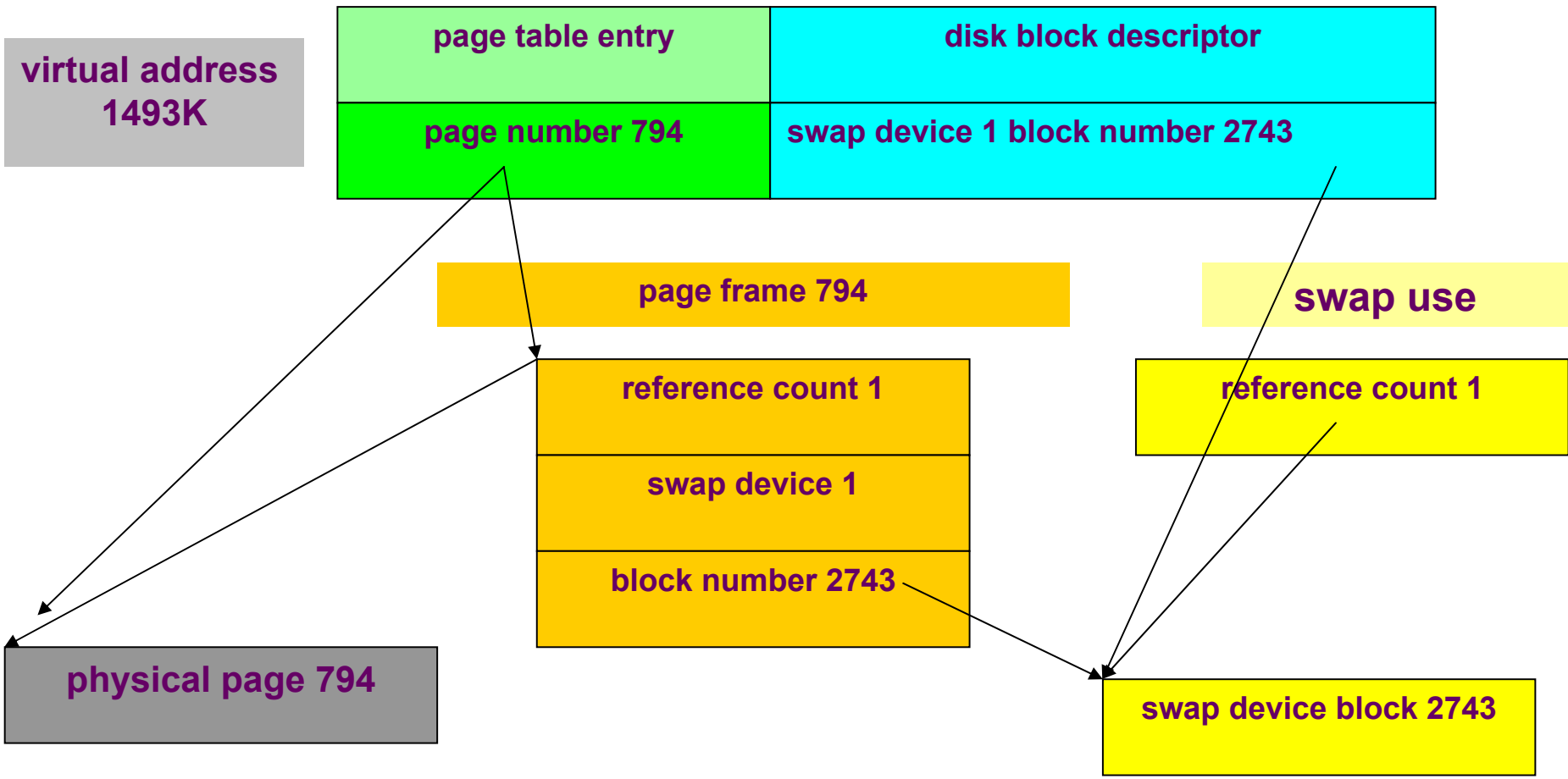


pfdata table

- Kernel linkuje **ulaze pfdata tabele** na:
  - **free listu**
  - **hashed listu**
  - što je analogno **linkovanim listama** kod bafer keša.
- **Free lista** (nisu prazne stranice, već kao cache bafer su **trenutno slobodne** za korišćenje) je
  - **cache of pages** koje se **raspoložive za dodeljivanje**
  - free lista olakšava pretraživanje.
- Kernel **alocira nove stranice** iz liste na bazi **LRU algoritma**.
- **Kernel** obavlja **hashing za pfdata ulaz**
  - **u saglasnosti** sa njegovom **swap adresom**
  - **na bazi swap adrese**, **kernel** može lako da **locira stranicu** ako je u **memoriji.**
- da bi **dodelio fizičku stranicu regionu**
  - kernel ukljanja stranicu sa head-a **free liste**,
  - ažurira device i blok number za swap
  - postavlja stranicu u odgovarajući **hash queue**.

swap-use table

- **Swap-use tabela sadrži ulaze** za **svaku stranicu** na **swap uređaju**.
- Ulaz sadrži **RC** o tome koliko ulaza iz PT ukazuju na **page u swapu**.
- **----------primer----------**
- **Na slici** je **prikazana veza između**
  - **PT ulaza,**
  - **disk blok deskriptora,**
  - **ulaza pfdata tabele i**
  - **swap-use table.**
- Virtuelna adresa **1493K** procesa se mapira na PT ulaz
  - koji pokazuje na **fizičku page 794**;
- **disk blok deskriptor** za taj PT ulaz ukazuje da kopija stranice postoji
  - na swap uređaju 1 u disk-bloku 2743.
- Ulaz **pfdata tabele za page 794**, takođe ukazuje da
  - kopija stranice postoji na swap uređaju 1 u disk-bloku 2743 i
  - da je njegov in-core RC=1.
  - Videćemo kasnije, zašto se swap adresa duplicira u pfdata tabeli i u disk blok deskriptoru.
- **Swap-use count** za **virtuelne stranice je 1**, što znači da **jedan ulaz iz PT ukazuje na swap kopiju**.


virtual address
1493K
page table entry
disk block descriptor
page number 794
swap device 1 block number 2743
page frame 794
swap use
reference count 1
reference count 1
swap device 1
block number 2743
physical page 794
swap device block 2743

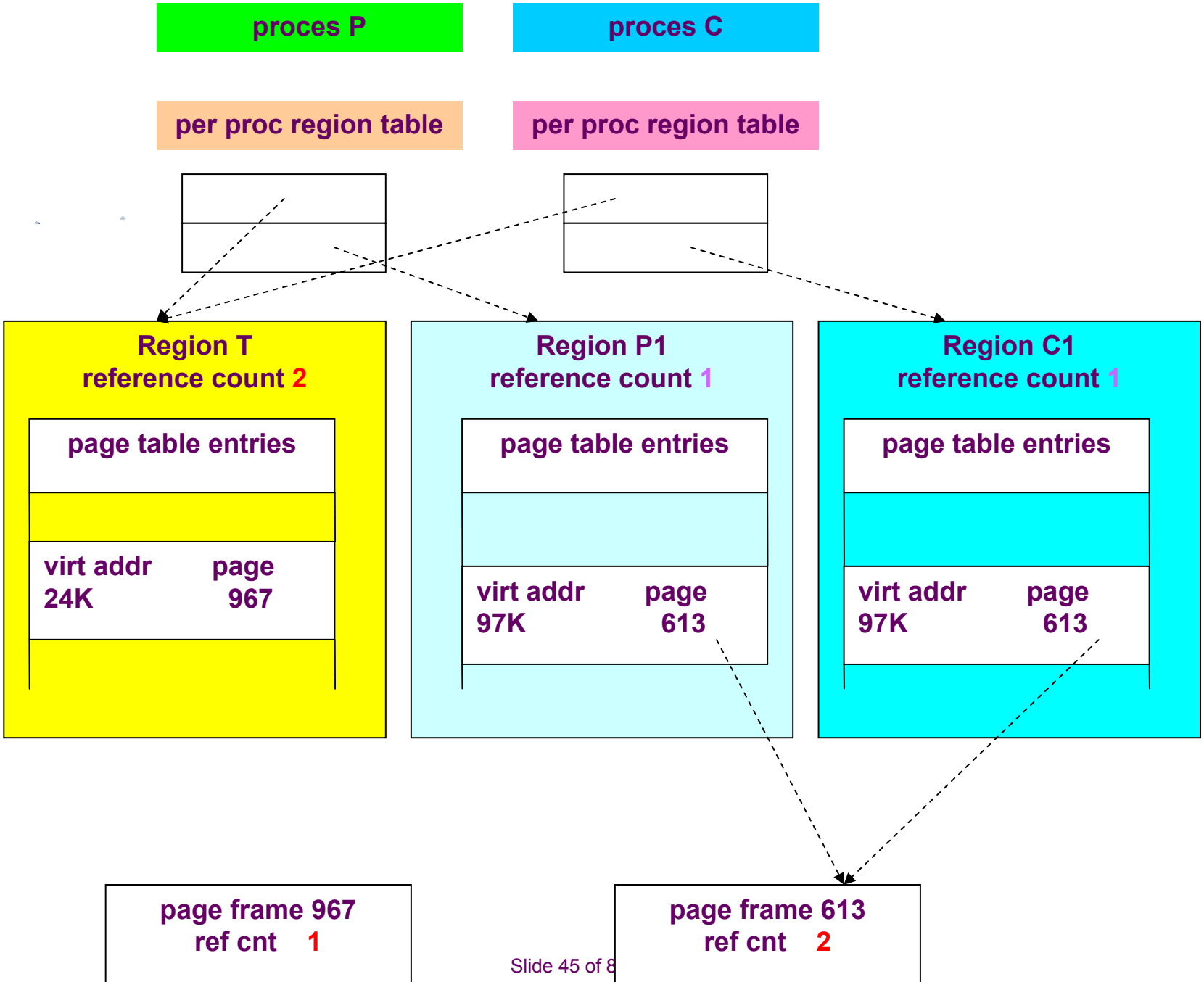
Fork in paging system

- **Kernel duplicira svaki region roditeljskog procesa** za **vreme fork SC** i **dodeljuje ga procesu detetu**.
- Tradicionalno, **za swaping sisteme**, kernel **pravi fizičku kopiju roditeljskog adresnog prostora**, a to je **nekorisno jer procesi zovu exec** odmah posle forka, a **exec oslobodi svu memoriju od svoje kopije pa traži novu**.
- Na UNIX system V sa paging, kernel **izbegava kopiranje stranica** preko **manipulacije** sa region tabelama, PT ulazima, i ulazima pfdata tabele.
- **Za deljive regione**, kernel prosto inkrementira **region RC**.
- Za **privatne regione** (**data i stack**), kernel će inicirati
- **novi ulaz** u **region tabeli**, i
- novu **PageTable**,
- ali tada se se ispituju svaki ulaz roditeljske PT:
  - **ako je stranica (page) validna**,
    - inkrementira se RC u ulazu za pfdata tabeli,
    - koji ukazuje na broj procesa koji **dele stranicu preko različitih regiona**
    - (za razliku od broja koji dele page preko deljenja regiona).
  - **Ako stranica postoji na swapu**, inkrementira se RC za stranicu u swap-use tabeli.

Fork in paging system

- **Na ovaj način stranici može da se pristupa iz oba regiona (roditelj, dete) sve dok neko** od procesa **ne poželi upis u nju**.
- **Kernel tada kopira stranicu** tako da svaki region **dobije svoju privatnu kopiju**. **Da bi se ovo dešavalo, kernel postavlja**
  - "**copy on write**" bit za svaku page privatnog regiona roditelja i deteta
  - **za vreme forka**.
  - **Ako drugi proces zatraži upis**, dogodiće se **protection fault** i kernel će napraviti kopiju **faulted procesu**.
- **Fizičko kopiranje** stranica **se odlaže do realne potrebe**.

- **Slika** daje izgled struktura kada **proces forkuje**.
- Procesi dele pristup PT za **shared text region T**,
  - tako da je njegov region RC=2
  - pfdata RC za pages u T =1.
- Kernel alocira **novi child data region (C1),** koji je **kopija regiona P1** procesa roditelja.
  - PT ulazi za oba regiona su i**dentični**,
  - a prikazan je ulaz sa **VA=97K**.
  - Ovaj PT ukazuje na pfdata ulaz 613, čiji je RC=2, što govori da **2 regiona** ukazuju na **istu stranicu-frame**.


proces P
proces C
per proc region table
per proc region table
Region T
reference count 2
page table entries
virt addr
24K
page
967
Region P1
reference count 1
page table entries
virt addr
97K
page
613
Region C1
reference count 1
page table entries
virt addr
97K
page
613
page frame 967
ref cnt 1
page frame 613
ref cnt 2

exec in paging system

- Kada **proces pozove exec SC**, kernel čita **celu exe-file iz FS u memoriju**.
- **Na DP**, **exe-file može biti prevelika** da **stane u raspoloživu memoriju**.
- Kernel ne obavlja neku reorganizaciju memorije,
  - on **jednostavno pozove punjenje file** a
  - **memorija se dodelju prema potrebi**.
- Prvo se **dodeli PT** i **disk blok desktiptori** na **exe-file**,
  - PT ulazi za **non-bss** se markiraju kao "**demand fill**",
  - a PT ulazi za **bss data je markiraju kao "demand zero**".
- **Preko read algoritma**, **proces pravi PF prilikom čitanja svake stranice**.
- **Fault handler** kada vidi "**demand fill**"
  - to **ima značenje** da stranicu neposredno treba **napuniti sa exe-file** tako da je ne mora brisati,
- a ako je markirana sa "**demand zero**",
  - to **znači da sadržina stranice mora** da se **popuni nulama.**
- U opisu **validity fault handlera**, biće objašnjeno kako se to radi.
- **Ako proces ne može da stane u memoriju**,
  - **page-steale**r proces periodično swap-out stranice u swap,
  - da bi se oslobodila memorija za **incomming exe-file**.

exec in paging system

- **Postoje ozbiljni nedostaci ove šeme**.
- **Prvo**,
- **proces za svaku stranicu izaziva PF**,
- bez obzira da li će exe-file ikada da koristi tu stranicu ili ne.

- **Drugo,**
- **page-stealer** proces može
- **swap-ovati** stranice pre **nego što se obavi exec**
  - što izaziva **2 extra swap operacije po stranici**,
  - ako proces **prerano potraži stranicu** (call or jump).

exec in paging system

- Da bi učinio **exec efikasnijim**,
  - kernel može da **obavlja DP direktno** iz **exe-file**
  - ako su data propisno dodeljeni,
  - na šta **ukazuje magic number**.

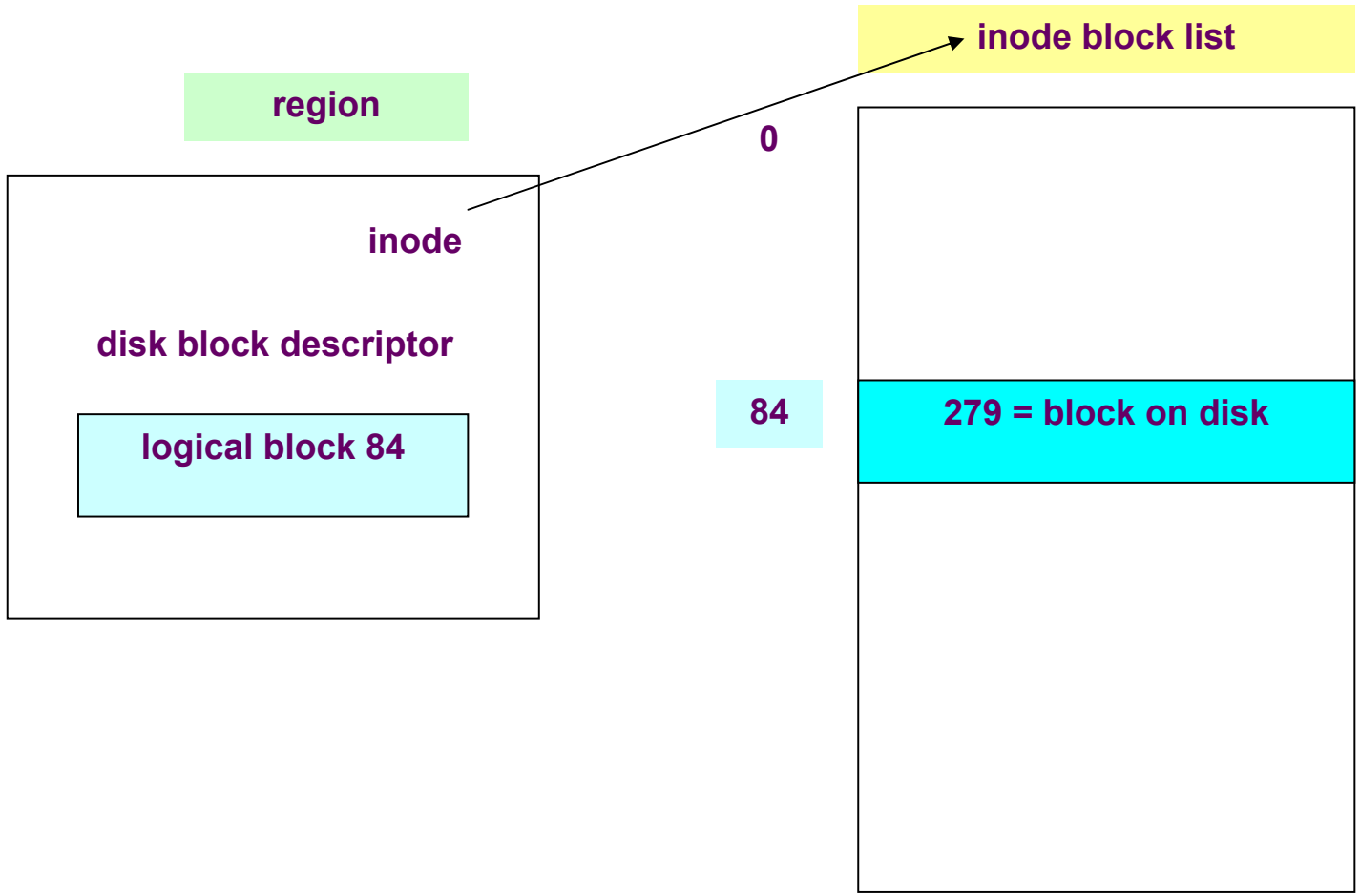
- **korišćenje** standarnih algoritama **kroz keš bafer nije baš pogodno za DP.**

- Za **paging direktno iz exe file**,
  - kernel pronalazi **sve brojeve disk blokova** za **exe-file**
  - kada se obavlja exec
  - **attachuje** ih u **listu** u **file-inodu**.

- **Kada se setuju PT** za **takvu exe-file**,
  - kernel markira **disk blok deskriptor** sa **logičkim blok brojem**
  - (počevši od bloka 0 u exe-file) koji se **sadrži u stranici**,
  - a **validity fault handler** posle toga **koriste ove informacije**
  - **kada puni stranicu iz datoteke**.

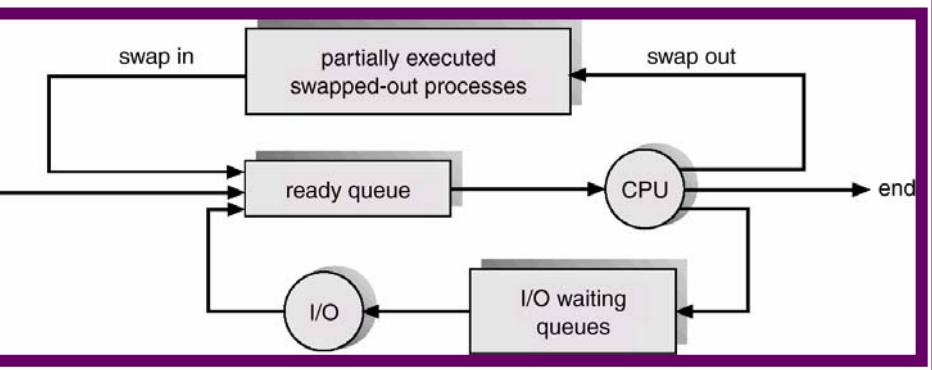
example

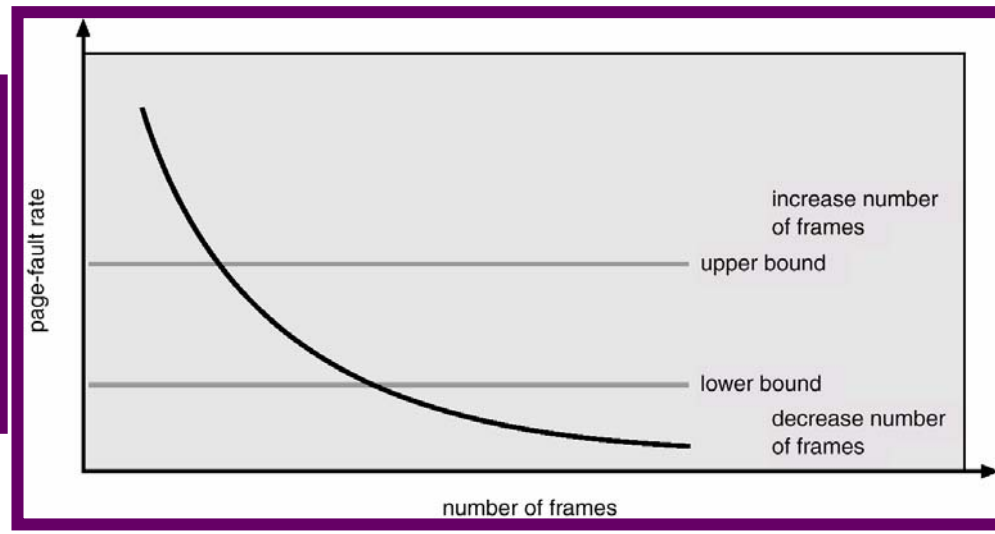
- Na slici imamo tipično aranžiranje, gde **disk blok deskriptor** ukazuje na stranicu koja je **84th blok datoteke**. Kernel prati pointer iz regiona u inode u kome traži odgovarujuće blokove diska.



Page stealer process

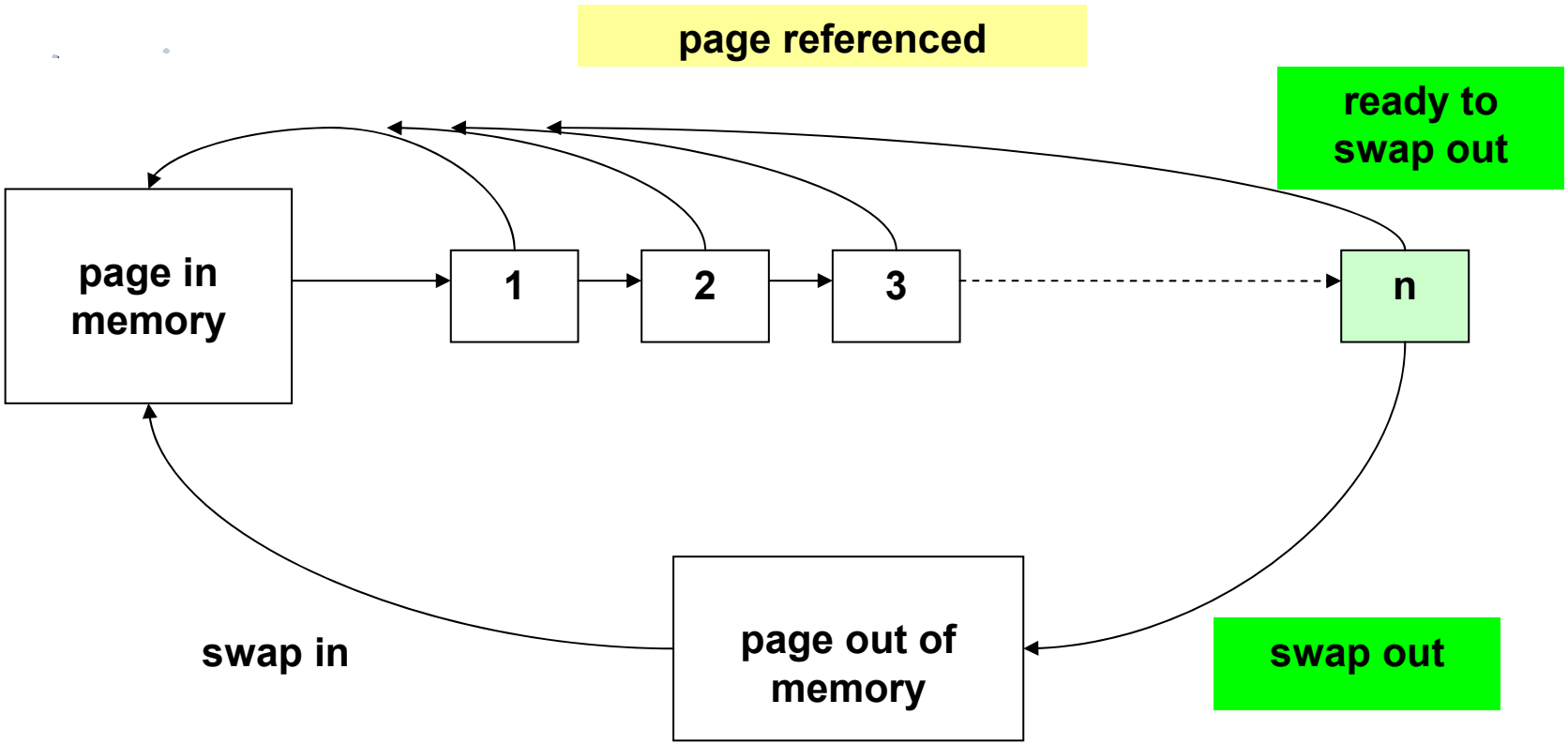
- **Page stealer** proces obavlja **swap-out stranica** koji **nisu više deo working seta procesa.**
- **Kernel kreira page stealer** za vreme **sistemske inicijalizacije** i poziva sve dok je UNIX podignut, uvek kada je **malo slobodnih stranica**.
- **PS ispituje svaki aktivni**, **unlocked region**, preskačući locked regione, i **inkrementira age polje svih validnih stranica**.
- Kernel lockuje region kada se dogodi **PF u regionu**, tako **PS ne može ukrasti stranicu iz takvog regiona.**





Page stealer process

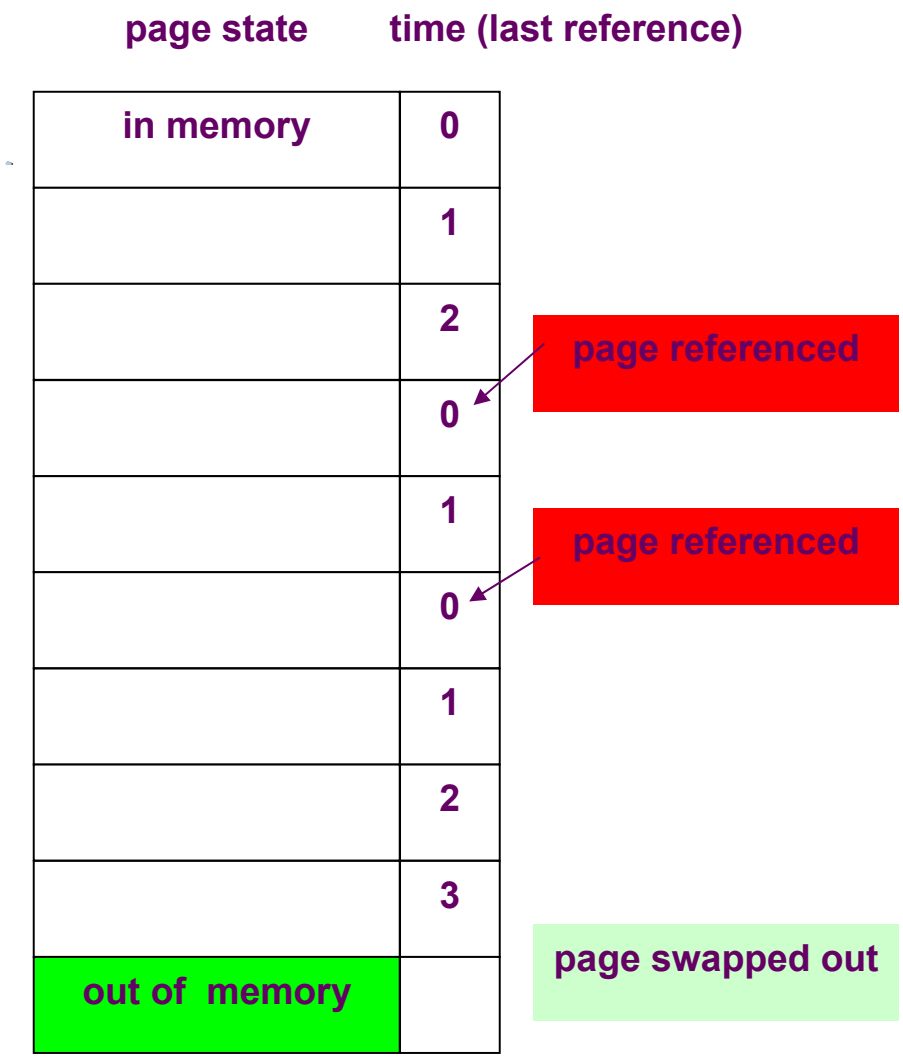
- Postoje **2 stanja** za stranice u memoriji:
  - 1. **page is aging i nije još poželjna za swapping**
  - 2. **page je poželjna za swapping** i raspoloživa za **reassingment** drugim virtulenim stranicama.
- Prvo stanje je kada proces **nedavno pristupao stranici**, pa je ona u njegovom **workung setu**. Neke mašine **setuju reference bit** kada se proces obraća stranici, a ako to ne podržava hardver, **to se može simulirati softverski**.
- **PS gasi reference bit** za **takve stranice**, ali pamti koliko je ispitivanja prošlo od poslednje reference za stranicu.
- Prvo stanje se sastoji od **više nižih stanja**, koje odgovaraju broju **prolazaka PS pre nego što stranica postane poželja za swapping**.
- **Kada broj probije threshold** vrednost, **kernel stavlja stranicu** u drugo stanje (**ready to swapped**).
- **Maksimalni period** za koji **stranica postane stara (age),** zavisi od implementacije, a ograničena je brojem **raspoloživih stranica**.


page referenced
ready to swap out
page in memory
1
2
3
n
swap in
page out of memory
swap out

PS example

- Na slici-dole je prikazana interakcija između procesa koji pristupaju stranici i ispitivanja koje obavlja PS.
- **Stranica počinje u memoriji** i **slika prikazuje broj ispitivanja PS uzmeđu memorijskih referenci.**
- **Proces** referencira stranicu posle **drugog ispitivanja**, obarajući njen **age=0**.
- Slično, proces referncira stranicu ponovo posle još jednog PS ispititivanja.
- Na kraju, PS ispituje stranicu 3 puta bez promene njene reference i obavlja **swap-ou**t.

PS example


page state
time (last reference)
in memory
0
1
2
0
1
0
1
2
3
out of memory
page referenced
page referenced
page swapped out

PS calling

- **Ako 2 ili više procesa dele region**, oni **ažuriraju reference** bite **za iste ulaze** u **PT**. Stranice mogu biti deo working seta za više procesa, **ali to ne zanima PS**.
- **Ako je stranica** deo working seta **bilo kog procesa ona ostaje** u **memoriji**,
- ako **ne pripada ni jednom working** setu, **poželjna je za izbacivanje.**
- **Nije bitno da li region ima više stranica u memoriji od drugih**, PS ne pokušava da obavi swap-out za **jednak broj stranica iz svih aktivnih regiona**.
- **Kernel će probuditi PS**, kada je količina slobodnih stranica ispod low-water mark i PS tada **swap-uje stranice** sve dok količina slobodne memorije ne dostigne **high-water mark**.
- Korišćenje **low-water mark** i **high water mark** smanjuje **trashing:**
  - **ako kernel koristi samo jedan threshold**, on će **prazniti stranice iznad potrebnih** vrednosti, pa je moguće da se one opet vrate u memoriju i da opet naprave trashing.
  - Zato se swaping-out obavlja do **high-water** mark i to traje duže sve dok broj free pages na padne ispod low-water mark, tako da **PS se ne poziva tako često**.
- **Administratori mogu da podese low** i **high water mark**.

PS

- Ako PS odluči da obavi **swap-out stranice**, analizira se da li **kopija stranica postoji na swapu.**
- **Postoje 3 mogućnosti:**
- 1. ako **nema kopije stranice na swap uređaju**,
  - kernel bira stranicu za swapping:
  - **PS postavlja stranicu na listu stranica koje će imati swap-out** i nastavlja, **swap-out je logički završen**.
  - Kada lista stranica koje treba da se **swapuju dostigne makismum**, kernel upisuje **stranice na swap uređaj**.
- 2. ako **kopija stranice već postoji na swapu**, i **nijedan proces joj nije modifikovao sadržaj (in-core**), a to znači da je modify bit 0,
  - kernel obriše **valid bit za stranicu**,
  - **dekrementira RC** u **pfdata ulazu**,
  - **gura ulaz stranice** na **free listu za buduće alokcije**
- 3. ako je **kopija stranice na swapu, ali in-core sadržaj je modifikovan**,
  - kernel bira stranicu za swapping,
  - **kao pod 1,**
  - a **oslobađa njen prostor na swapu**.

case 1, 2 i 3

- **PS kopira stranicu** na **swap samo u slučaju 1 i 3.**
- Objasnimo razliku između ovih slučajeva.

- Pretpostavimo da je stranica **na swapu** i da je ušla u memoriju nakon PF, a pretpostavimo da kernel automatski ne ukljanja disk copy.

- Eventualno, PS odluči da swap-out stranicu opet.

- Ako **nijedan proces nije modifikovao stranicu**, memorijska kopija je identična disk kopiji **pa nema potrebe da se ponovo upisuje na swap**. (**case 2**)

- Ako je **proces modifikovao stranicu** (case 3)
  - kernel mora da upiše stranicu na swap,
  - **to se ne radi overwrite**,
  - nego se oslbodi prostor na swapu koji je prethodno zauzimala,
  - a stranica se ponovo upiše negde **drugde na swap**,
  - **tako da se swap prostor održava kontinualnim, a to daje bolje** performanse.

case 1 i 3

- **PS puni listu stanica** koje će **biti swap-ovane,**
  - **moguće** iz **različitih regiona**,
  - **swapuje ih kad je lista puna**.

- **Svaka stranica procesa ne mora biti swapovana**: neke stranice nemaju dovoljno starosti (age) i
- to je **glavna razlika** između **paging i swapping sistema** kod kojih ide **ceo proces na swap**.

- Ako swap **uređaj nema dovoljno mesta**, **tada kernel swapuje po jednu stranicu u vremenu.**

- **Postoji veća fragmentacija** na **swapu**
  - **kod paging sistema** u **odnosu na swaping sistem,**
  - **zato što kernel swapuje stranicu po stranicu.**

page swap-out

- Kada kernel upisuje stranicu na swap,
  - on **isključuje valid bit u PT** ulazu i
  - dekrementira **use count** u **pfdata table ulazu**.
- **Ako RC (use-count) padne na 0**,
  - **plasira** se **ulaz pfdata tabele** na **free listu**,
  - **keširajući** ga **dok se ne dodeli**.
- Ako **use-count nije 0**
  - **drugi procesi dele stranicu** kao rezultat **fork SC**,
  - **ali kernel će svakako obaviti swap-out za stranicu**.
- Na kraju,
  - **kernel alocira swap space**,
  - **čuva swap adresu** u **disk blok deskriptoru** i
  - i**nkrementira swap-use table count** za stranicu.

- **Ako proces napravi PF** dok je **stranica na free listi**,
  - **kernel će vratiti stranicu iz memorije**
  - nego da je obnavlja sa swap-a,
  - a ako je **kernel ne izbaci** iz **free liste** ona će ipak biti kasnije **swap-ovana**.

PS example

- Na primer,
- ako PS **swapuje 30, 40, 50 i 20 stranica** od procesa A, B, C i D,
- a zatim upiše **64 stranice u jednoj disk operaciji**.

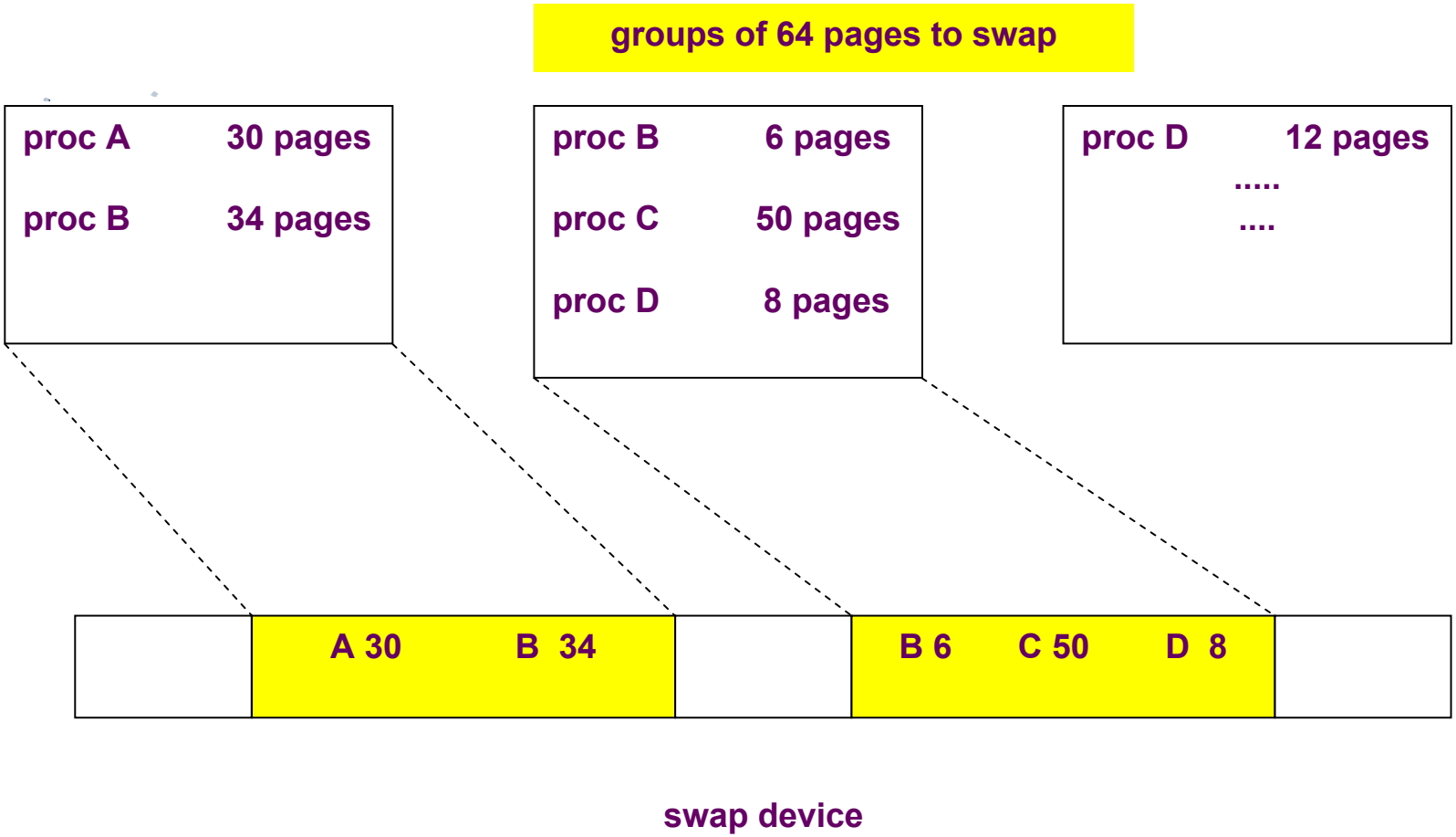
- Slika prikazuje sekvencu page-swaping operacija koje se dešavaju kada PS **ispituju redom procese A, B, C i D**.
- PS dodeljuje prostor na na **swapu za 64 stanice**, pa zatim upisuje 30 stranica procesa A i 34 stranice procesa B.

- Zatim alocira prostor za još 64 stranice na swapu, i njoj swapuje 6 stranica procesa B, 50 stranica procesa C i 8 stranica za proces D.

- Dva područja na swapu za vreme 2 swap write operacije ne moraju da budu **kontinualna**.
- **PS proces** drži **ostalih 12 stranica procesa D** u listi za swapping, ali **nema upisa** na **swap dok se ne popuni lista.**

- Kada procesi uzimaju svoje stranice sa swap-a kada se dogodi PF i kada obave exit, tako se ažurira **free space na swapu.**

PS example: PS swapuje 30, 40, 50 i 20 stranica od procesa A, B, C i D



Page faults

- Sistem može obuhvatiti **2 tipa PF**:
- **validity faults**
- **protection faults**

- Zato što fault handleri mogu čitati stranicu sa diska u memoriju
- i **spavati za vreme disk I/O operacije,**
  - **fault handler je jedini izuzetak interrupt handlera,**
  - **koji po pravilu ne mogu na spavanje.**

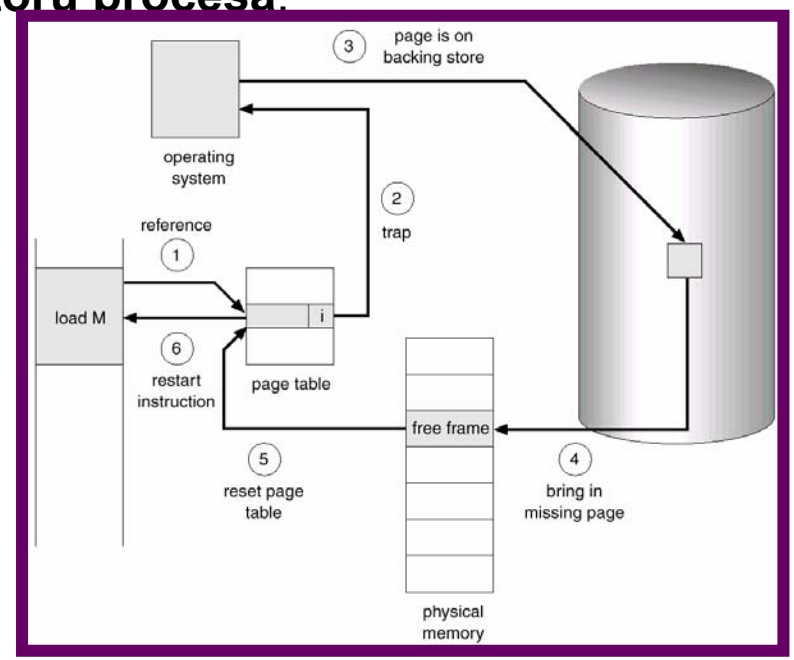
- Međutim **FH spava** u **kontekstu procesa koji je izazvao PF**,
- **FH je povezan** sa **tekućim procesom**, **zajedno spavaju**. .

Validity fault handler

- Ako **proces pokušava da priđe stranici** čiji **valid bit nije setovan**,
  - **dogodiće se validity fault** i
  - **kernel će prozvati VFH vfault**.

- **Valid bit nije setovan** za:
- 1. **stranice izvan virtuelnog adresnog prostora procesa**

- 2. **stranice koje jesu u virtuelnom prostoru procesa**,
- **ali još nema dodeljenu fizičku stranicu.**



algorithm vfault

- **algorithm vfault /*handler for validity faults*/**
- input: address where proces faulted
- output: none
- {
- find region, PT entry, **disk block descriptor** corresponding to faulted address, **lock region**;
- **if(address outside virtual address space)**
- {
- **send signal(SIGSEGV: segementation violation) to process**;
- goto out;
- }
- **if(address now valid)** /* process may have slept above*/
- {
- **goto out**; /*next page*/
- }
- **if(page in cache)**
- {
- **remove page from cache;**
- **adjust page table entry**;
- while(page contents not **valid)** /* another process faulted first*/
- sleep(event contents become **valid**)
- }

algorithm vfault

- else /* **page not in cache***/
- {
- **assign new page to region**;
- put new page in cache, update pfdata entry;
- if(page not previously loaded and page "**demand zero**")
  - clear assigned page to 0;
- else
- {
- **read virtual page** from **swap device** or **exec file**;
- **sleep**(event I/O done);
- }
- **awaken processes (event page contents valid);**
- }
- **set page valid bit**;
- **clear page modify bit, page age**;
- **recalculate** process **priority**;
- **out: unlock region**;
- }

algortithm vfault

- **Hardver obaveštava kernel** koja je **virtuelna adresa izazvala PF**,
- kernel nalazi
  - **region i PT ulaz**
  - **disk blok deskriptor** za tu stranicu.
- **Kernel lockuje region** koji **sadrži PT ulaz**, tako da spreči **race condition** koji bi **mogao** da se **napravi** ako **PS pokuša** da obavi **swap-out** za **stranicu**.
- **Ako disk blok deskriptor nema zapis za faulted stranicu**,
  - pokušana memorijska referenca je invalidna
  - kernel šalje "segmentation violation" signal procesu koji je emitovao tu referencu.
- **Ako je** memorijska **referenca bila legalna**, kernel **alocira stranicu memorije** u koju će se pročitati **sadržina swap uređaja ili iz exe-datoteke**.
- **Stranica koja je izazvala PF nalazi se** u **jednom** od **5 stanja**:
- **1. samo swap uređaju a ne u memoriji**
- **2. u free page listi u memoriji**
- **3. u executabilnoj datoteci**
- **4. označena kao "demand zero"**
- **5. označena kao "demand fill"**

case 1

- U slučaju 1, **stranica je**
  - **samo na swapu**
  - **nije** u **memoriji**,
  - **jednaput** je bila u **memoriji** ali ju je **PS** prebacio u swap.

- Iz **disk blok deskriptora**,
- **kernel nalazi swap uređaj** i **blok number**
- verifikuje da li je page u **page kešu**.

- **Kernel ažurira PT ulaz** tako da ukazuju na stranicu koju treba pročitati,
  - podešavaju pfdata ulaz na hash listu
  - kako bi ubrzali operaciju za fault handler
  - čitaju stranicu sa **swap uređaja**.

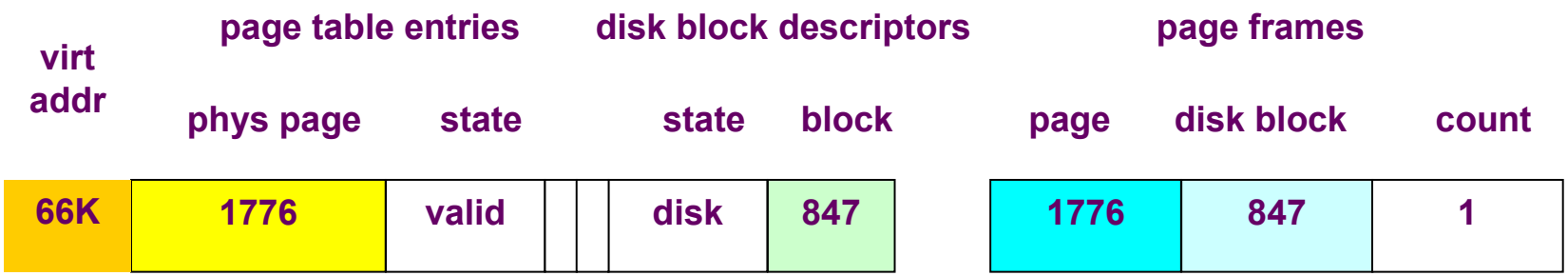
- **Faulted proces spava** sve dok se **ne završi I/O**, kada **kernel** budi sve procese koji su čekali na tu stranicu.

- **Uzimo za primer** PT **ulaz** za **virtulenu** adresu **66K** na sledećoj slici.


page table entries
disk block descriptors
page frames
virt addr
phys page
state
state
block
page
disk block
count
0
1K
2K
3K
4K
64K
65K
66K
67K
1468
Inv
None
Inv
1861
Inv
none
Inv
1036
Inv
file
3
DF
5
Disk
1206
DZ
Disk
847
1036
387
0
1648
1618
1
1861
1206
0

case 1

- Ako proces izazove validity fault, fault handler ispituje disk blok desktriptor i vidi da se stranica nalazi na **swap bloku 847**.
- **Virtuelna adresa** je **legalna**, pa fault handler pretražuje **page cache** i tamo ne **nalazi ulaz za disk blok 847**.
- Ako nema kopije u fizičkoj memoriji ili u page kešu, fault handler je mora pročitati sa diska (swap).
- **Kernel dodeljuje stranicu 1776**
  - **čita sadržaj sa swapa** u **nju**
  - **ažurira PT** ulaz da **ukazuje** na **stranicu 1776**
  - na kraju **ažurira disk blok deskriptor** da **ukazuje** na **stranicu koja je još uvek i na swapu**
  - **pfdata ulaz za stranicu 1776** da pokazuje na **blok 847** na swap uređaju koji sadrži **duplikat virtulene stranice**



case2

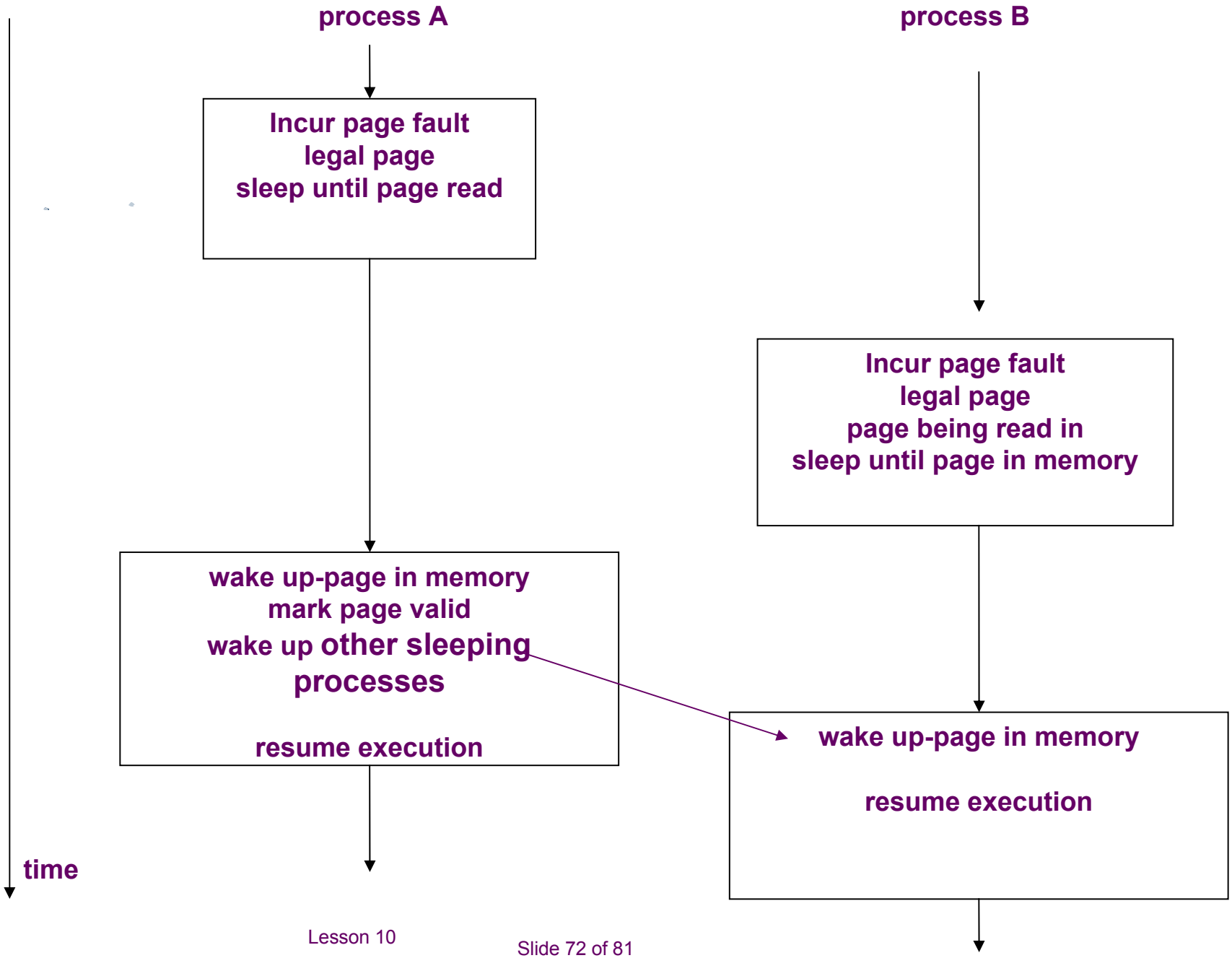
- (**case2) Kernel ne obavlja uvek I/O operaciju** kada se dogodi **validity fault**, bez obzira što disk blok deskriptor ukazuje na stranicu koja je na swapu.
- Moguće je da kernel
  - odradio **swap-out za** tu stranicu a ona je jos **uvek** u **page cache**
  - **da drugi procec** uzme istu **fizičku stranicu**.
- U **oba slučaja**, **fault handler nalazi stranicu** u **page kešu,**
- a detektuje je po **bloku u disk desktriptor** bloku.
- Kernel ponovno
  - **dodeljuje PT** ulaz da ukazuje na **nađenu stranicu**,
  - inkremnetira njen **page RC**,
  - uklanja je iz **free liste**.
- **Na primer**: **proces pravi PF** kada pristupa **virtuelnoj adresi 64K**.
- Tražeći po page kešu (**po disk bloku 1206**)
  - kernel nalazi da page okvir 1861 ima **disk blok desktriptor 1206**.
  - Kernel postavlja ulaz **64K** u PT da ukazuje na **frame 1861**,
  - **postavlja valid bit**.
- Zato se **disk blok upisuje** u **obe tabele PT** i **pfdata**.

case1-another proces

- Takođe, **fault hanlder ne mora** da **čita stranicu u memoriju**
- ako je **drugi proces** već imao **PF za isti frame, tj isti disk blok**,
- ali je **nije još pročitao**.

- **Fault handler** nalazi **region koji sadrži koji sadrži PT** ulaz
- zaključan **drugom instancom fault handlera**.

- **On tada spava** dok **prethodna instanca ne pročita stranicu**
- i
- **oglasi je validnom**, **kao na slici**


process A
process B
Incur page fault
legal page
sleep until page read
Incur page fault
legal page
page being read in
sleep until page in memory
wake up-page in memory
mark page valid
wake up other sleeping processes
resume execution
wake up-page in memory
resume execution
time

case #3

- (**case 3**) ako **kopija stranice**
  - **ne postoji na swapu**
  - **nego u originalnom exe-file**,
  - **kernel čita stranicu iz originalne datoteke**.
- **Fault handler** ispituje **disk blok deskriptor**,
  - **nalazi logički blok datoteke** koji **sadrži page**
  - **nalazi inode pridružen** sa **ulazom u region tabeli**.
- Koristi se **logički blok** kao **offset** za polje **disk blokova** attach-ed u **inodu** za vreme **exec SC**.
- Poznajući **disk blok number** iz inode, **stranica** se **čita sa diska u memoriju**.
- **Na primer**
  - **disk blok deskrtiptor**
  - za **virtuelne adresu 1K prikazuje**
  - da **page sadrži logički broj 3** u exe-file.

case #4,5

- (case 4, 5)
- Ako **proces izazove PF** za **stranicu označenu kao:**
- **"demand fill"**
- **"demand zero"**

- kernel
  - **alocira free page u memoriji**, i ažurira odgovarujući PT ulaz.
  - na zahtev "**demand zero**" upisuju se **0-le u stranicu**.
  - na kraju se brišu flagovi "**demand zero**" ili "**demand fill**".

- **Page je validna** i njen sadržaj se **ne duplicira** na **swap ili u FS**.

- Ovo će se dogoditi za virtulene adrese **3K i 65K**.

- **Nijedan proces** nije pristupao ovim stranicama nakon što se **obavio exec SC**.

end of vfault

- **Validity fault** handler **završava se:**
  - **setovanjem valid bita**
  - **brisanjem modify bita**
  - zatim se **rekalkuliše prioritet procesa**,
  - pošto proces **može biti uspavan u fault handleru** (**kernel priority**) dajući mu non-fair prednost kad se vrati u user mod.
- **Na kraju**,
- **po povratku** u **user mod**,
- **proverava** da li je bilo **nekih signala dok se obrađivao PF**.

Protection fault handler

- **Druga vrsta memory fault-a** koji **proces može izazvati** je **protection fault**,
- koji znači
  - da **proces pristupa validnoj stranici**,
  - ali **zaštitini biti pridruženi stranici ne dozvojavaju pristup**.
- Proces može takođe izazvati **protection fault**,
  - kada pokušava **upis u stranicu**
  - koja ima setovan **copy-on-write**, za vreme fork SC.
- Kernel mora **odrediti** da li su **prava zabranjena** zato što
  - page zahteva **copy-on-write (to je legalno)**
  - ili se nešto **zaista ilegalno** dogodilo,
- **Hardver će obezbediti** PFH **virtulenu adresu** gde se dogodila greška,
- a PFH će odrediti odgovarajući **region i ulaz u PT**.

algorithm pfault

- **algorithm pfault /*handler for protection faults*/**
- input: **address where proces faulted**
- output: none
- {
- **find region**, PT entry, disk block descriptor corresponding to faulted address, **lock region**;

- **if(page not valid in memory)** goto **out**;

- if(**copy on write** bit not set) /* real program error-signal*/ goto **out;**

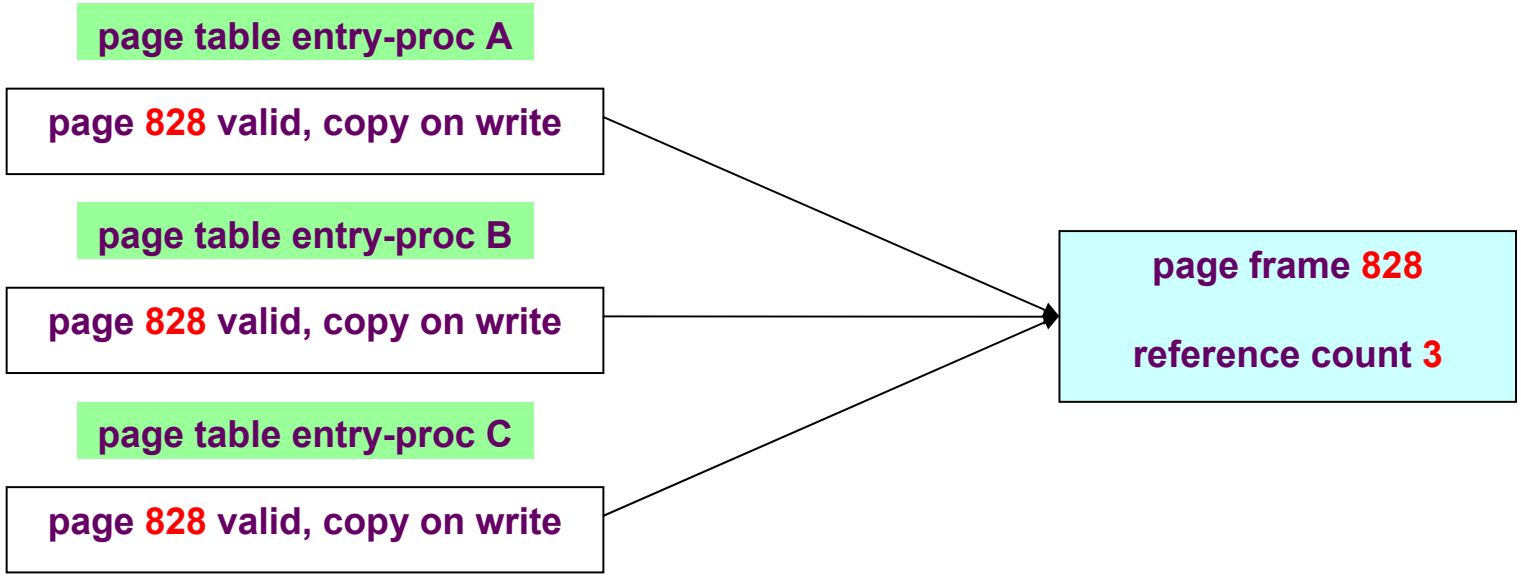
- **if(page frame reference count > 1) /*copy on write*/**
- {
- **allocate a new physical page**;
- **copy contents** of **old page to new page;**
- **decrement old page frame RC;**
- **update page table entry** to **point** to **new physical page**;
- }

algorithm pfault

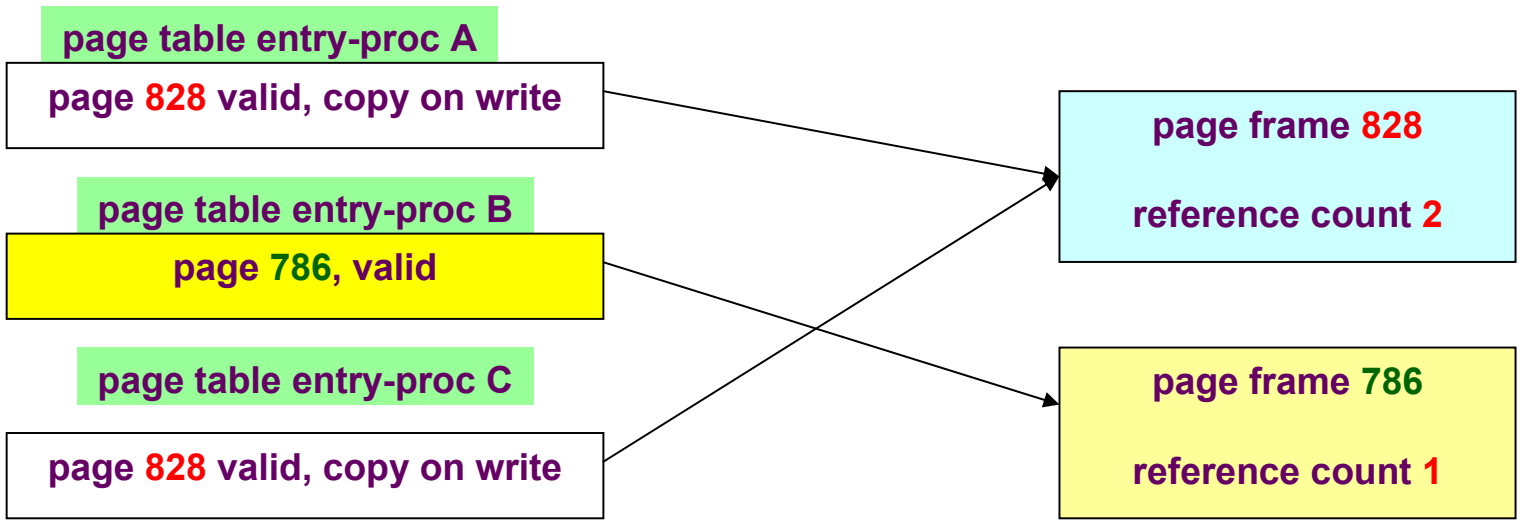
- **else /* steal page, since nobady else is using it */**
- {
- **if(copy of page exists on swap device)**
- free space on swap device, break page association;
- **if(page is on page hash queue)**
- remove from hash queue;
- }
- **set modify bit**, **clear copy-on-write** bit in PT entry;
- **recalculate proces priority**;
- **check for signals**;
- **out**: unlock region;
- }
- **On će zaključati region**, tako da PS ne može ukrasti stranicu dok nju obrađuje protection fault handler radi sa njom.
- Ako **PFH** odredi da je PF izazvao **copy-on-write bit**, **kernel alocira novu stranicu** i kopira sadržaj stare stranice u nju.
- **Drugi procesi** zadržaće reference na **staru stranicu**.
- Posle kopiranja stranice i ažuriranja PT ulaza sa novom stranicom, kernel dekrementira **RC starog pfdata ulaza**.

copy-on-write bit example

- **Slika pokazuje scenario:**
- **tri procesa dele fizičku stranicu 828.**
- **Proces B pokušava upis u stranicu**,
  - ali **to izaziva protection fault** zbog **copy-on-write bita**.
  - PFH **alocira stranicu 786**,
  - **kopira sadržaj stranice 828**,
  - dekrementira RC za page 828 i
  - ažurira PT ulaz za proces B da ukazuje na page 768.
- ne
- Ako je copy-on-write bit setovan ali nijedan drugi proces ne deli stranicu,
- kernel će dozvoliti procesu da obavi **ponovno korišćenje fizičke stranice**.
- **Ukida** se **copy-on-write bit**, razdvaja se stranica od svoje disk kopije (ako postoji) jer **drugi procesi mogu** da **dele disk kopiju**.
- Zatim se ukljanja pfdata ulaz iz page queue, zato što nova kopija virtuelne stranice nije na swapu.
- Zatim, se dekremntira swap-use count, i ako count padne na 0, oslobađa se swap prostor.



a) before process B incurs protection fault



b) after protection fault runs for process B

Hybrid system with swaping and demand paging

- Mada **DP tretira sistem mnogo fleksibilnije od swaping sistema**, ali su moguće situacije kada **PS i VFH trash**-uju **zvog nedostatka memorije**.
- Ako je zbir working setova svih procesa veća od fizičke memorije, **fault handler obično spava**, zato što ne može da **alocira stranice za procese**.
- **PS ne može da krade stranice dovoljno brzo**, zato što su sve stranice u working setu, tako je performanse sistema **drastično padaju**, jer kernel provodi mnogo vremena u overheadu rearanžirajući memoriju.

- **System V kernel** izvršava i **swapping i paging algoritme** da bi **spečio trashing.**
- **Kada kernel više ne može da dodeljuje stranice za procese**,
  - **budi se proces swapper**
  - **gura ceo novi proces** kao "**ready to run but swappped**".
  - više procesa mogu biti u takvom stanju simultano.
  - **swapper uradi swap-out** za cele procese sve dok **slobodna memorija** ne pređe **high-level mark**.
  - **Svaki proces** koji se **ceo swapuje postaje "ready to run but swappped"**.
  - Iza toga se radi **normalan DP**, a postoje razne **hibridne kombinacije.**