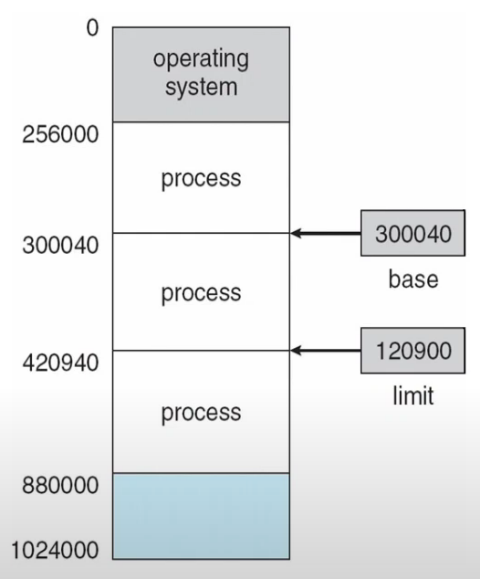
**Menaxhimi i memories**

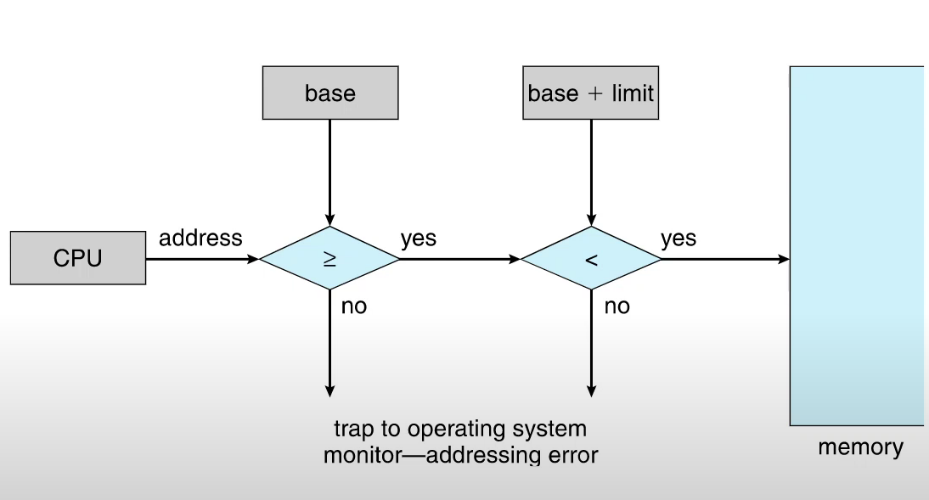
*Segment* – I referohemi një pjese të memories të dedikuar për një proces.

*Adresa fizike* – Është adresa reale e memories.

*Adresa virtuale/logjike* – Adres e gjeneruar nga cpu për një proces.

*Memory Menagment Unit* ( MMU ) – Është pjes harduerike e cila bën menagjimin e memories.

Në memorie gjinden disa *regjistra kufitar* të cilët e përcaktojn hapsirën e një prcoesi në memorie, kta regjistra kontrollohen nga OS, ndërsa CPU kontrollon kërkesat që i vin për qasje në memorie se a janë brenda kufive.



*MMU* është pjes harduerike e cila e bën përkthimin e adresave logjike në ato fizike.

Lidhja e adresave logjike me ato fizike ( instruksioneve me të dhëna ) mund të bëhet në tri forma :  *Compile Time* – Kompajlleri ja gjeneron adresën fizike direkt, dhe në atë adress vendoset procesi, OS nuk e menaxhon. *Load Time* – Kompajlleri e gjeneron një adres logjike për procesin, pas përkthimit OS e mer dhe e vendos në memorie në pozitën fillestare të adresës së përkthyer dhe më nuk e lëkund nga aty. *Execute Time* – Kompajlleri ja gjeneron adresën logjike pastaj OS e vendos në memorie pra e njëjta si *load time* mirpo këtu lejohet edhe lëvizja e procesit në memorie.

Kur bëhet ndarja e memories (dmth venodsja e proceseve në memorien e njëjtë ) memory sharing duhet pas parasysh tri kushte : *Transparenca* – Në main memory të mund të vendosen shumë procese njëkohësisht, këto procese mos të dit se janë në memorie të përbashkët, proceset mos të përkujdesen për adresat fizike që iu kan përcaktuar. *Siguria* – Proceset nuk duhet të mund të ndërprejn punën e njëri tjetrit. Proceset nuk duhet të mund të ndërprejn punën e OS. *Efiqienca* – Memoria e përbashkët nuk dueht ta degradojë performancën e cpu.

*Rilokimi* është formë e menagjimit të memories ram. Kemi dy forma të *rilokimit* : *Rilokimi statik* - Në *load time* pasqyrohet adresa e procesit në adres fizike dhe OS e vendos procesin në atë adresë dhe më nuk mund ta lëviz nga aty. *Rilokimi dinamik* – Hardueri shton regjistra bazë në adresën logjike, pastaj i krahason me regjistrat kufitar të memories, nësë krahasimi del jo i suksesëshëm ndëpritet memoria virtuale dhe eleminohet përdorimi i adresës fizike.

Përparësit e përdorimit të *rilokimit dinamik* : Na lejon të bëjm lëvizjen e procesve në memorie, lejon zgjerimin e procesit, hardueri është i shpejtë dhe i thjesht dy regjistra shtesë, shtim dhe krahasim.

Mangësitë e përdorimit të *rilokimit dinamik* : Meqenëse proceset mund të lëvizin nuk mundësohet memory sharing, menagjimi i memories komplikohet, procesi është i limituar nga madhësia fizike e memories.

*Rilokimi dinamik e plotëson kushtin e* : Transparencës – Proceset nuk kan dijeni për memorien e përbashkët. Sigurisë : Secila instancë e memories është e kontrolluar. Efqiencës : Hardueri është i shpejtë në krahasimin e regjistrave kufitar të adresës logjike me të asaj fizike, mirpo lëvizjet në memorie dhe zgjerimi i proceseve e ngadalsojnë punën shkaku se i gjith procesi duhet kopjuar.

*Vrimat* janë pjesë të lira të memories të gatshme të ju alokohen proceseve.

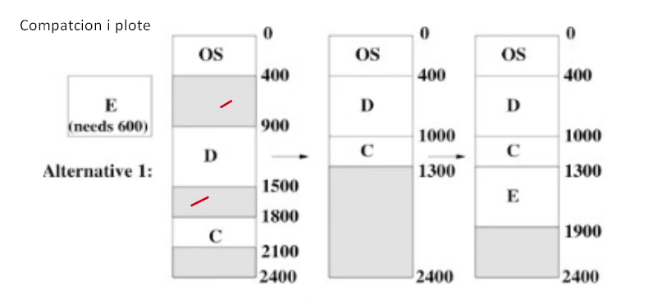
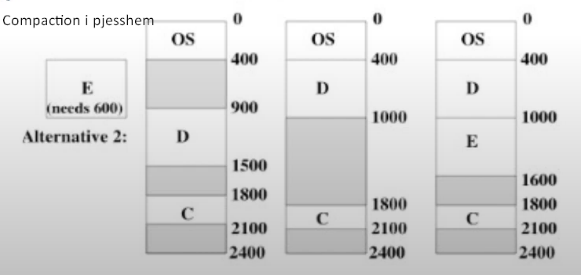
Për të zgjedhur se cila *vrim* të ju alokohet cilit proces përdorim këto rregulla : *First-Fit* : Procesi vendoset në vrimën e parë të lirë e cila ka hapsirë të mjaftueshme për at process, kërkimi fillon nga vrima e parë ose nga *first-fit* i fundit. *Best-Fit* : Procesi vendoset në vrimën që iu aforhet më së afërti me madhësi procesit. Për kët form OS duhet të kontrolloj të gjith listën. *Worst-Fit* : Procesi vendoset në vrimën me hapsirën më të madhe.

*FF* dhe *BF* janë më efiqiente se *WF*. Ndërsa *FF* është më e shpejtë se *BF*.

*Fragmentim* quajm *vrimat* të cilat gjenden në mes proceseve, ndahet në fragmentim të *brendshëm* dhe të *jashtëm*.

*Fragmentim* *i jashtëm* ndodhë shkaku i ngarikim dhe shkarkimit të proceseve nga memoria. Në memorie ka hapsira të lira mirpo nuk janë hapsira të vazhdueshme.

*Fragmentimi i brendshëm* ndodhë kur blloku i alokuar për një proces ka hapsirë më të madhe se që procesit ju duhet. Le të themi se kemi një proces me 8846 byte të cilit i është alokuar një bllok prej 8848 byte, në kso raste është më e udhës ti ipet i gjith blloku procesit dhe mos të mbahet evidencë për 2 byte.

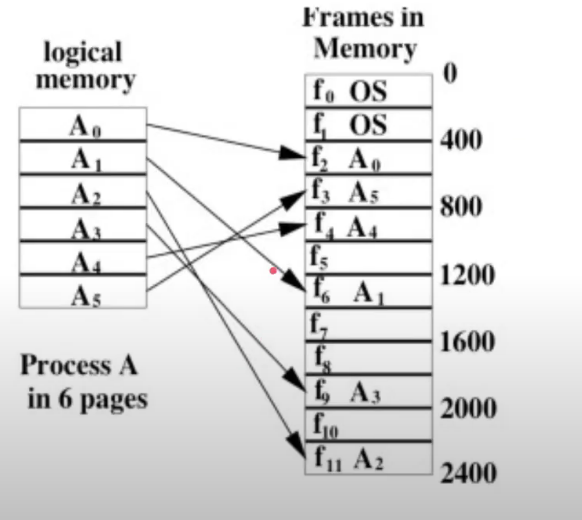
*Compaction* : Është defragmentimi i memories mund të jet i plotë ose i pjesëshëm.

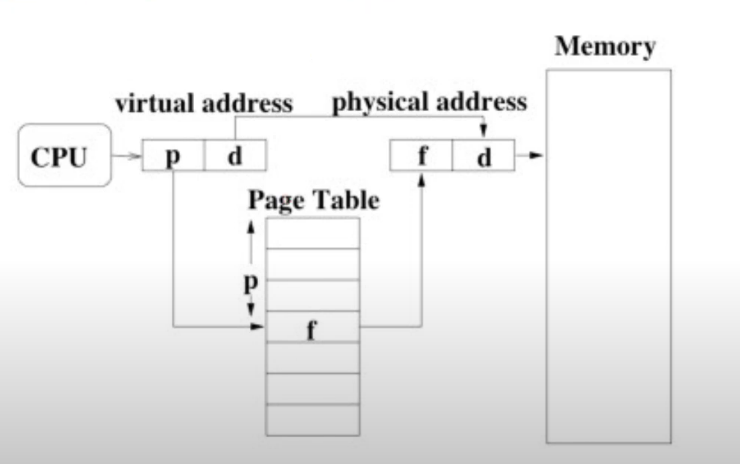
Problemet me menagjimin e memories me *rilokim* : *Fragmentimi* – Bëhen comapction të shpeshta që gjë shkaktojn vonesa. *Alokimi i vazhueshëm­* – Vështërsit në zgjerimin/ngushtimin e proceseve. *Nevoja që procesi të gjindet gjithmon në memorie* – Swapping e zgjedh kët problem por jo në mënyrë përfekte.

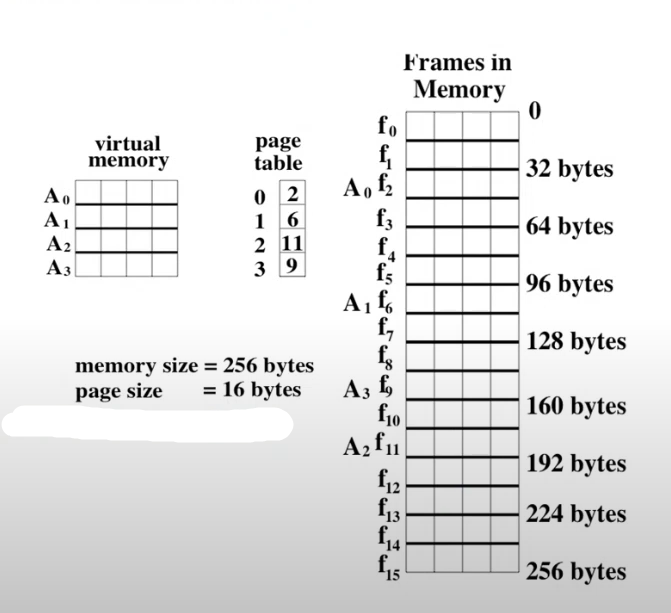
*Swaping* është kalimi i procesit nga memoria ram në disk, nëse përodrim *rilokim* statik për mengjimin e memories atëher duhet që *OS* ta kthej në të njëjtën adresë, procesin e lëvizur. Nëse jemi duke përdorur *rilokim dinamik* e vendos në ndonjë vrimë të lirë.

*Paging* është forma e menagjimit të memories e cila përdoret sot, bazohet në rregullin 90/10, që thotë se proceset 90% të kohës përdorin vetëm 10% të memories që i alokohet. Duke u bazuar në kët konstatojmë se nuk ka nevoj të vendoset i gjith procesi në memorie, pra të vendoset vetëm 10% i procesit i cili shfrytëzohet, ktë e bën *pagin* duke e eliminuar kështu *fragmentimin e jashtëm* shkaku se nuk është i vazhdueshëm. Por nuk e eleminon *fragmentimin e brendshëm*.

*Pagin realizohet në formë harduerike*.

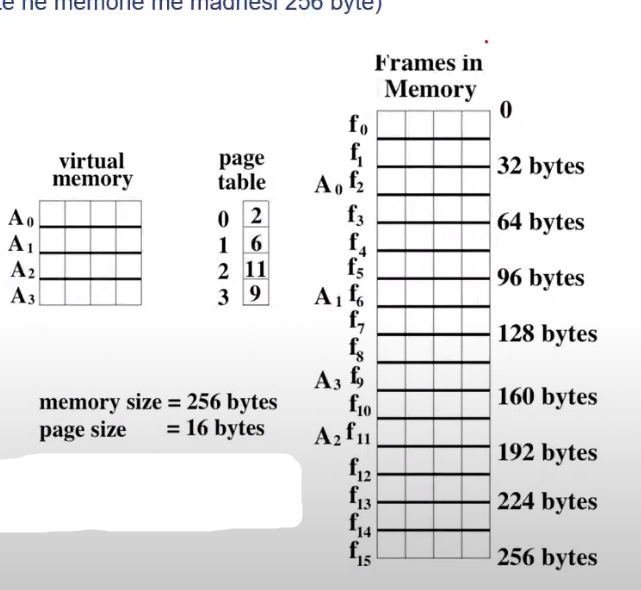


Problem i cili shfaqet kur përdorim *paging* është gjetja e adresave kur *page* nuk janë të alokuar në vazhdimësi në memorie. Ne e dimë se procesi përdorë adresat logjike për ti emërtuar lokacionet në memorie. CPU i gjeneron adresën logjike procesit nga 0 deri në sa ka nevoj. Pastaj OS e ndan procesin në *page*, hradueri më pastaj bën përkthimin e adresave logjike në ato fizike. Për ta zgjedhur kët problem duhet mbajtur një *Page Table* në të cilën mbahen shenime se cili *page* në cilin *frame* të memories gjendet.

Shembull : Memoria ram ka **256** **Bajt**, Ndërsa një frame i memories i ka **16 Bajt**. Numri i biteve për adresim është **8** **Bit** ngase **2^8 = 256**. Prej 8 biteve 4 bit janë për Page dhe 4 tjerë janë për offset. Është dhënë adresa virtuale **24** të kthehet në adres fizike.

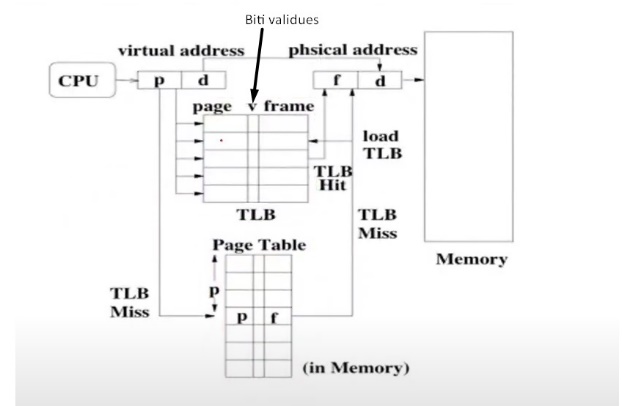
*Zgjidhje* : **p** =24/16 = 1; **d** = 24%16 = 8;

Në page table numrit **p**(1) i është asocuar frame **6**, kështu që në atë frame do vendoset page i 24.

Shembull : Kemi të bëjm me page të madhësisë **1 word** ose **4 byte**. Secili frame ka nga **4 words**. Për të adresuar 256 Bajt me words ne duhet të përdorim **6 Bit** sepse 256/4 = 64 words. 2^6 = 64. **p** = 4bit, **d** = 2bit. Adresa virtuale 13 të kthehet në adres fizike.

*Zgjidhje* : **p** = 13/4 = 3, **d** = 13%4 = 1. Frame = 9, d=1;

*Page Table* është një tabel që mban një sasi të konsiderueshme të të dhënave. Kemi mundësin e ruajties të kësaj tabelie në regjistra me qrastë do përfitojmë nga shpejtësia mirpo do të jemi të kufizuar në hapsirë, ose kemi mundësin e ruajties në memorie me qrastë shpejtësia është më e ulët por nuk kemi kufizim në memorie.

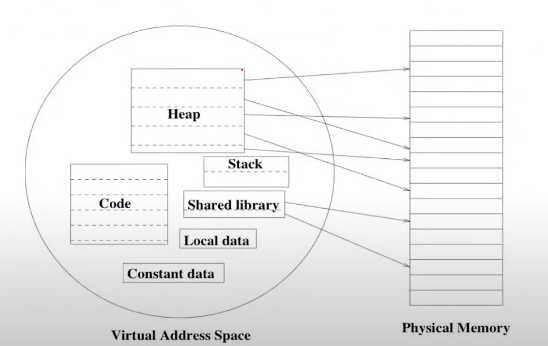
*TLB* është një tabel përkrahëse për *Page Table*, është më vogël me madhësi se *Page Table*. *TLB* ruhet në L1 cache dhe në të ruhen vetëm pasqyrimet më të përdorura. Madhësia tipike e *TLB* është 8-2042 Bajt. *TLB* i mban disa nga pasqyrimet e *Page Table* së bashku me nga një bit vildues, i cili tregon nëse pasqyrimi është i sakt ( a është i njëjti me pasqyrimin që gjendet në page table ). Nëse kyt bit është 1 kemi pasqyrim të saktë, por nëse është 0 kemi të bëjmë me *TLB MISS*. **EMA** (Effency Memory Access), sa më i madhë cache aq më i madhë EMA.

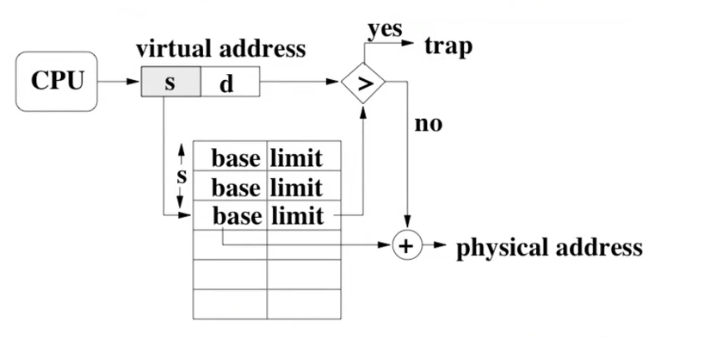
Hapat e inicimit të memories kur procesi starton : 1.Procesi ka kërkesën për n page. 2.Nëse n frame për page janë të lira i alokohen pagave, nësë nuk janë lirohen frame për të ju alokuar atyre page-s. 3.OS e pasqyron secilin page në frame dhe e bën vendosjen në *Page Table*.OS e pastron *TLB* (Flush TLB). 5.OS e starton procesin. 6.Gjatë kohës kur është duke u ekzekutuar procesi, OS e ngarkon TLB me vlerat e page-s.

*PCB* duhet të zgjerohet për të mbajtur *Page Table* dhe *TLB* kopje nëse është e mundur.

Gjatë *Cotnext Switching* fillimisht kopjohet *Page Table dhe TLB* në PCB, pastrohet TLB dhe rivendoset Page Table së bashku me TLB e procesit të ri.

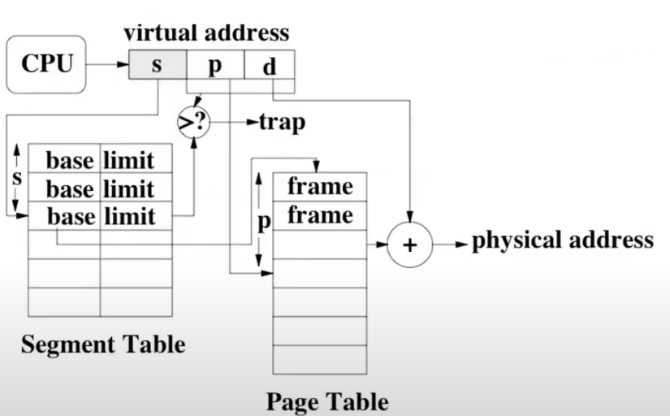
*Page vs Rilikomi* : Page është përmisim i madhë në krahasim me rilokimin sepse e menjanon fragmentimin e jashtëm pa pasur nevoj për kompaktim, lejon ekzekutimin e proceseve edhe nëse nuk gjendet të plota në memorie, lejojnë memory sharing. Mirpo edhe *Page* e ka koston e vetë, në përkthimin e adresave virtuale në fizike kemi kosto, kërkohet TLB për performanc më të mirë si dhe OS duhet të jetë më kompleks për ta kontrolluar dhe mbajtur *Page Table.*

*Segmentimi* mundëson që sistemi operativ të shfaqë memorien në pikpamjen e userit, userit memorien e she si të ndarë në *heap, data, code, stack...*

Kur përdoret segmentimi kompajlleri duhet të gjenerojë referencë për të identifikuar segmentin dhe offset-in. Përdoret një *Segment Table* ku ruhen pasqyrimet e segmenteve të memories virtuale në frame të memories fizike.

Nëse përdorim *Segmentimin* për menagjim të memories do të kemi problemin me fragmentimin e jashtëm, kjo përshkak se madhësia e segmenteve varion, nuk janë të madhësis së njëjtë si page, mirpo fragmentimin e brendshëm e menjanon. Për të implementuar segmentimin mund të përdoret rilokimi dinamik ose ai statik, ndërsa pasqyrimet realizohen njësoj si tek Page.

Për të përfituar nga të mirat e *Segmentimit* dhe *Page* mund ti kombinojmë së bashku. Memorien virtuale e ndajmë në segmente dhe secilin segment në të e ndajmë në page, të cilat pastaj si të tilla do të vendosen në frame të memories fizike. Në rastin e përdorimit të kombinimit Adresa virtuale duhet të zgjerohet për të mbajtur në vete numrin e segmentit, numrin e page brenda atij segmenti si dhe offseton e atj page.

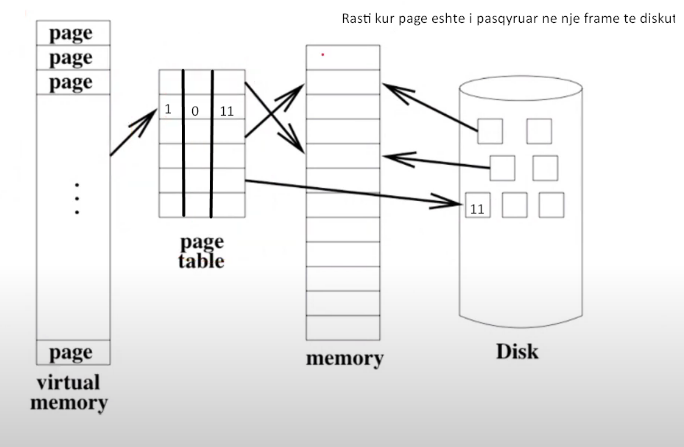


Nëse përdorim kombinim të segment dhe page, *segment table* mund të ruhet në regjistra ndërsa *Page Table* së bashku me *TLB* ti vëndosim në memorie në kët rast do të fitojmë nga shpejtësia mirpo hapsira do jet e kufizuar. Ose mund ti vendosim të dyat tabelat në memorie.

Përparësit e përdorimit të kombinit janë : fleksibiliteti, kombinimi i pikpamjes së kompajllerit me të OS, eleminon fragmentimin e jashtëm, lejon ndarjen e memories mes proceseve. Ka edhe kosoto në aspektin e context switching shkaku se duhet të bëjmë përkthim kompleks të adresave.

Duke përdorur diskun mund të krijojmë një memorie virtuale të “pa fundme” nga madhësia, në të cilën mund të vedosim proceset të cilat nuk kanë vend në memorien fizike, kjo gjë na mundëson të ekzekutojmë shumë procese në të njëjtën kohë, të ekzekutohen procese që kanë madhësi më të madhe se memoria fizike ose të ekzekutohen proceset pa u vendosur të plota në memorien fizike.

Nëse kërkohet një page i cili gjendet në disk ai së pari duhet kthyer në memorien fizike për të arritut të dërgohet për procesim, me qenë se page gjendet në disk me shumë gjasë ne memorien fizike nuk ka pasur hapsir të mjaftueshme, pra duhet lirohet ndonjë frame ashtu që të vendoset page i thirrur në kët rast na ndihmon *demand paging*.(Page të kërkuara)

Për të përdorur *demand paging* në *Page Table* duhet të vendoset edhe një bit validues i cili tregon nëse page gjendet në memorie ose në disk (1- gjendet në memorie, 0- gjendet në disk). Me rasti ne kërkimit të një page kontrollohet biti validues, nëse ai është 0 fillon page të vendoset në memorie derisa të bëhet vendosja nga disku në memorie zhvillohet context switching, kur page narkohet në memorie lëshohet një sinjal që tregon për ngarkimin e page. Pasqyrimi në *Page Table* ndryshon së bashku me bitin validues, (ndryshon frame nga ai i disku në një të memories ku është ngarkuar).

*Demand Paging* mund të bëhet në tri raster : 1.*Në kohën kur starton procesi :* Hasira virtuale nuk mund të jet më e madhe se ajo fizike. 2.*Mbikqyrje* : Programeri i aplikacinit duhet të vendos kur të thirret page e kur të largohet, lejon që madhesia virtuale të jet më e madhe se ajo fizike. Kërkon shumë punë nga programeri. 3.*Page të kërkuara* : Procesi e njofton OS kur i duhet një page ose kur ka përfunduar punën me një page.

Page e kërkuar *(Demand)* OS e bën load në momentin e parë që referencohet, nëse procesi nuk gjendet në memorie kemi të bëjmë me *page-fault* me qrast OS e largon procesin nga CPU deri sa procesi të bëhet load nga disku në memorie, nëse nuk ka vend e largon ndonjë page që gjendet në memorie.

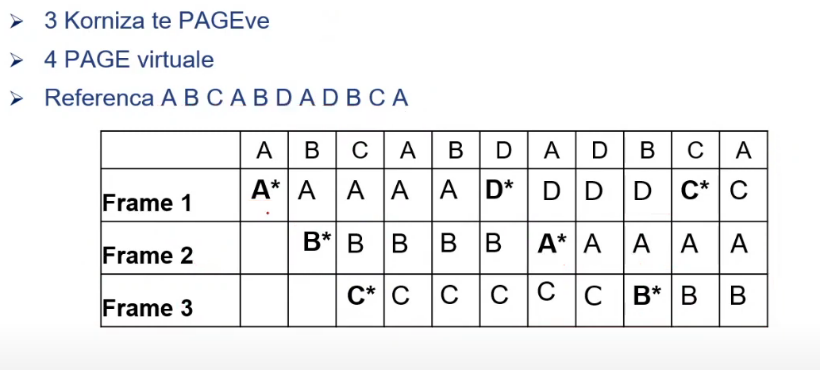
*Page e paracaktuar* : OS e parashikon se cili page do të përdoret ga procesi dhe e ngarkon në memorie, nëse parashikimi dështon hasim në page-fault, gjithashtu OS mund të largoj page të rëndësishëm nga memoria.

Por cka ndodhë me këto page të cilat largohen për me liru hapsirë në memorie?

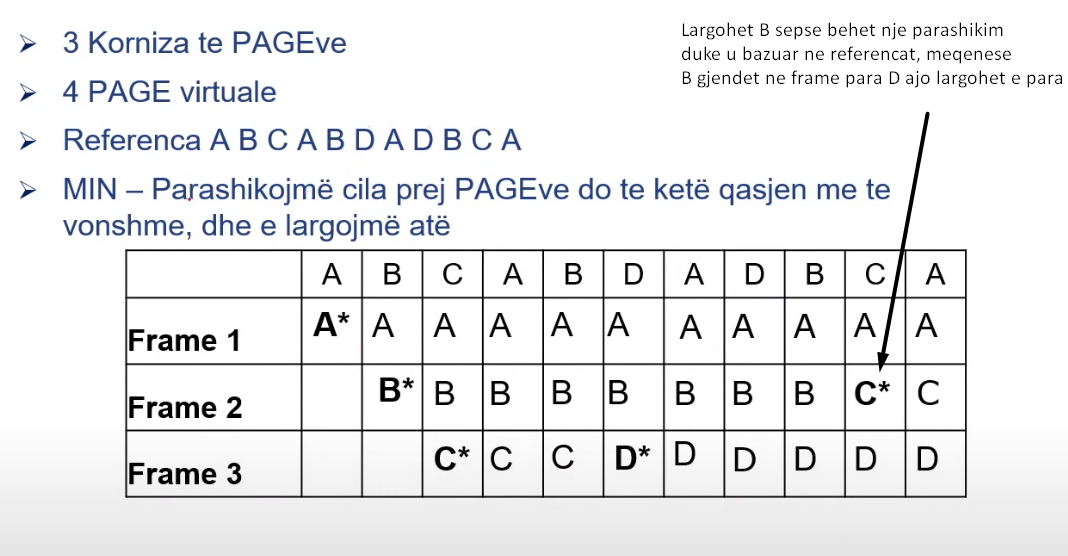
-OS i kontrollon se a kanë në përmbajtie vetëm kod ose përmbanë edhe të dhëna. Në rast se ka vetëm kod thjesht e largon nga memoria me qenëse kodi gjindet në disk dhe mund të mirret nga aty, por nëse mban të dhëna vendoset në *swap space* – është një hapsirë e rezervuar për page e zhvendosura.

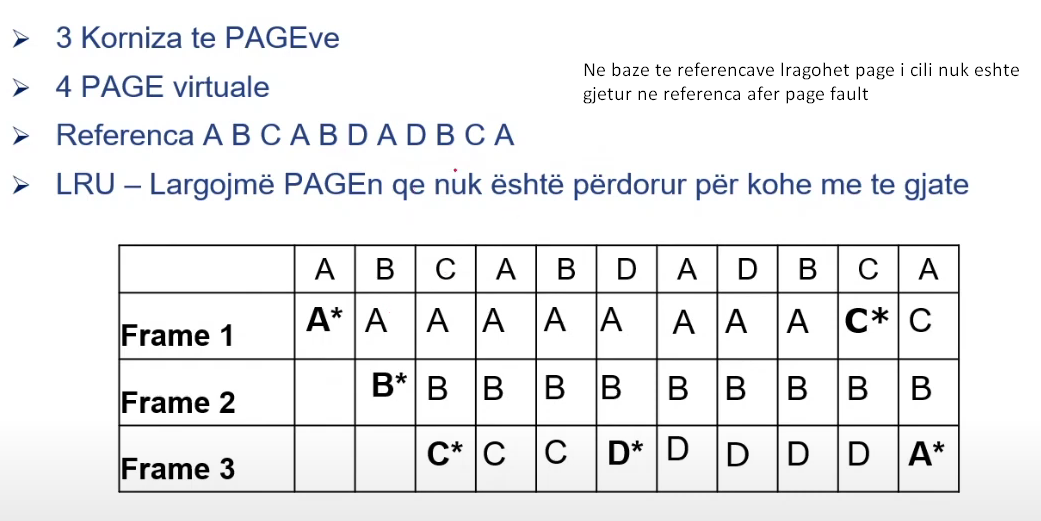
Për të pëzrgjedhur cilin page të largojmë përdorim disa algoritme të zhvendosjes të tilla si :

**1.FIFO** : E largon page më të vjetër, OS mund të largoj page të cilat ende përdoren por janë të vjetra.

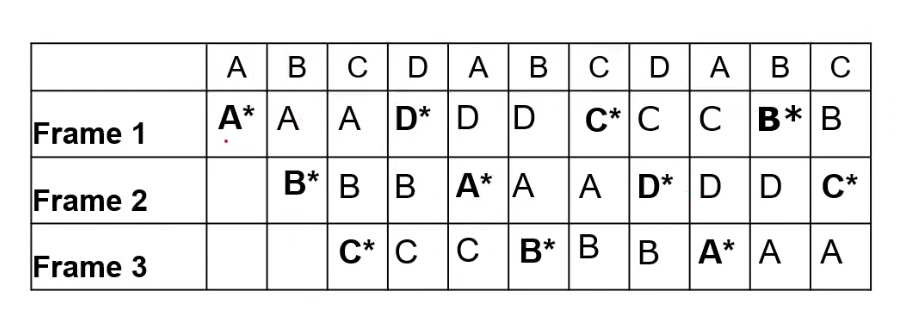


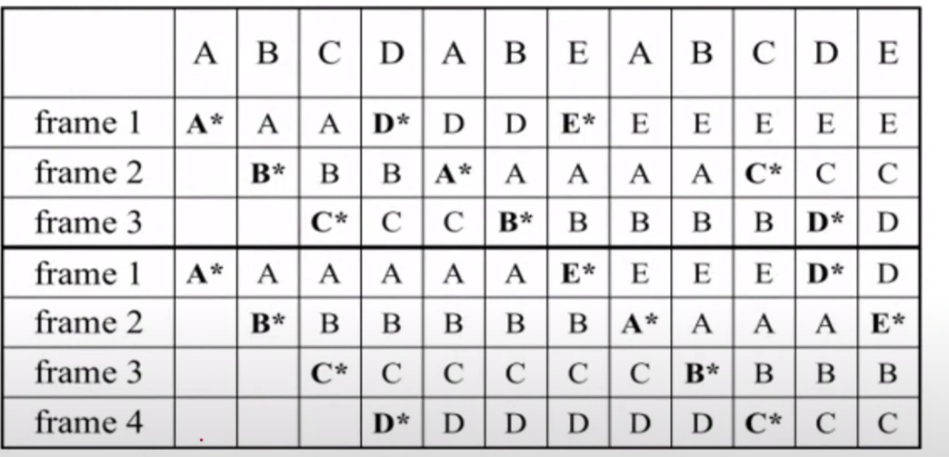
**2.MIN** : Largon page që përdoret më së paku.



**3.LRU** : Least Recently Used – Është e përafërt me MIN bazohet në idenë e tashmja është parashikues i sukseshëm i së ardhmës. E lagon page që ka kohën më të gjatë pa u përdorur.

**4.Random** : Me raste mund të funksionoj mirë.

*LRU në rastin më të keq :*

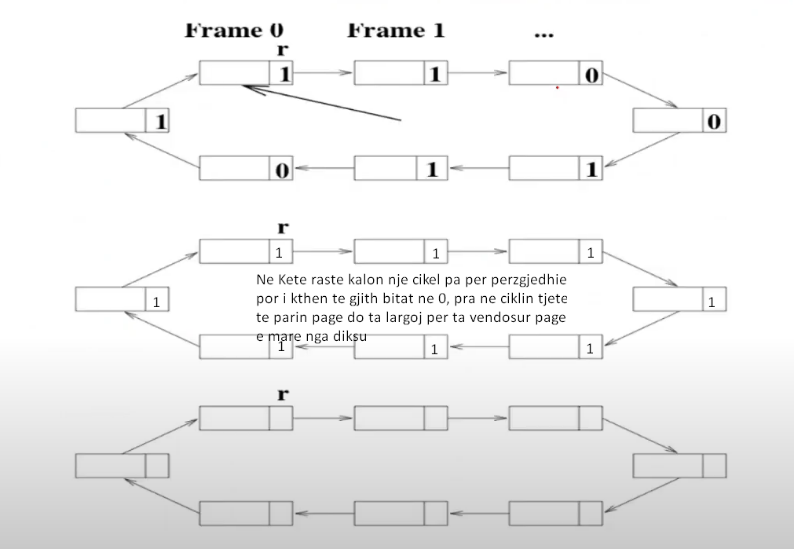
*Në FIFO nëse rritet numri i frame rritet edhe numri i page-fault:*

*Në LRU rritja e frame gjithmon e ulë numrin e page-fault*.

*LRU është algoritmi më i pështatshëm për zhvendosje të proceseve.*

*LRU si dhe të gjitha përafrimet e LRU kërkojnë mbështetje harduerike(bit referues)*.

*Përafrimi i LRU-s*  : mund ta përdorim me një bit referues ose me 8 bita refereues. Në rastin me 1b, 1- Tregon që në ciklin e kaluar ky page është përdorur, 0- Tregon që në ciklin e kaluar nuk është përdorur dhe me shumë gjasë mund të zgjedhet nga OS për tu larguar. Nëse përdorim 8b, pas gjdo cikli duke filluar nga ana e majt biti do bëhet 1-Nëse është përodrur në at cikël ose 0-Nëse nuk është përdorur në atë cikël. OS do ta zgjedh ta largoj page me më së paku 1.

Formë tjetër më e avancuar e algortimit *LRU* është *CLOCK*(mundësia e dytë). OS e organizonë memorien si në formë rrethore dhe kërkonë nëpër të nga ana e orës, i kontrollon page në bitin referues se a e kanë 1 ose 0, nëse page e ka bitin 1, OS e kalon nuk e prek por bitin ja kthen në 0. Nëse e ka 0 e zgjedh për ta largur. Nëse të gjitha page në memorie kanë bitin 1, LRU kthehet në FIFO ngase i pari do largohet.

LRU *CLOCK* i zgjedhuar(mundësia e dytë e zgjeruar) : Bazoehet në atë se largimi i page të cilët kanë pësuar ndryshim(dallojn nga ai që është në disk) kushton më shumë ngase duhet të rishkruhen në disk. Përdorë dy bita njëri tregon se a është përdorur ose jo tjt tregon se a është modifikuar ose jo.

-(0,0) – Page nuk është përdorur nuk është modifikuar.

-(0,1) – Page nuk është përdorur por është modifikuar.

-(1,0) – Page është përdorur por nuk është modifikuar.

-(1,1) – Page është përdorur dhe është modifikuar.

OS kërkon raste me (0,0) vetëm ato mund ti largoj, në rastin më të keq do i duhet ti bëj 3 cikle për të gjetur një page me (0,0).