Operacinės sistemos

N. Sarafinienė

2013m.

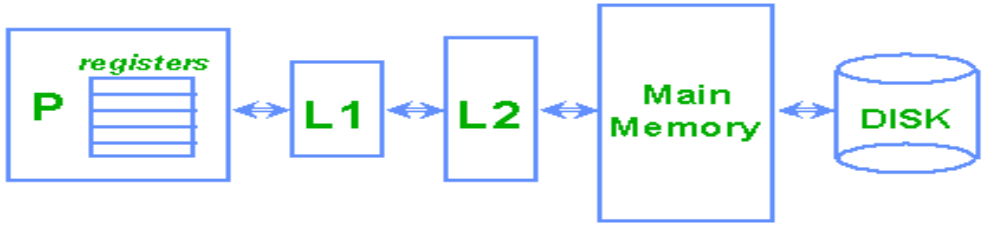
Kalbėsime

- Nagrinėsime atmintinės valdymo problemas
- Fiksuotų bei dinaminių skyrių sudarymą
- Virtualią atmintinę
- Puslapių lenteles

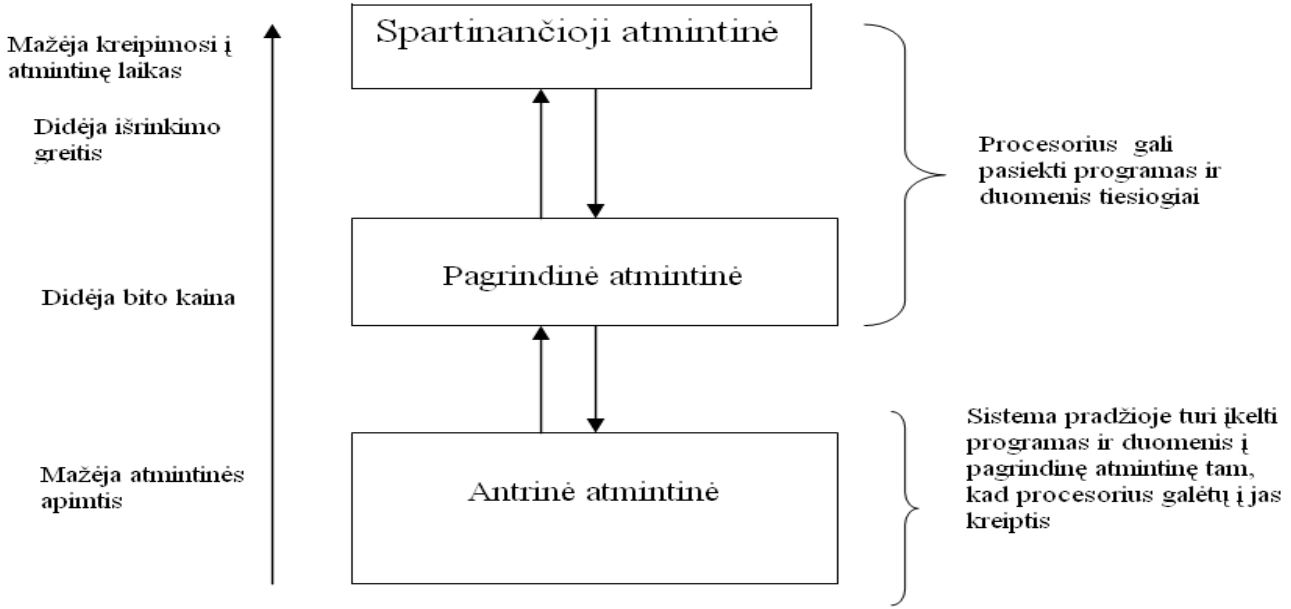
Atmintinės valdymas

- Operacinei sistemai reikia efektyviai išnaudoti pagrindinę atmintinę, sudedant į ją tiek procesų kiek įmanoma.
  - Daugumoje atvejų operacinės sistemos branduolys (kernel) užima tam tikrą fiksuotą pagrindinės atmintinės dalį.
  - Likusi atmintinės dalis yra padaloma daugybei procesų.
- Atmintinė:
  - ***pagrindinės atmintinės***
    - yra reliatyviai nedidelė savo apimtimi
    - pasižymi reliatyviai nedideliu informacijos išrinkimo greičiu,
  - ***išorinė, arba antrinė atmintinė***
    - pavyzdžiui diskai, magnetinės juostos,
    - pasižymi didele apimtimi,
    - yra reliatyviai pigi,
    - tačiau turi didesnius duomenų išrinkimo laikus.

Atmintinės hierarchija ir charakteristikos



Access time	0.5 clk	1 clk	5 clks	10-50 clks	10^5 clks
Capacity	1KB	16KB	1MB	1GB	10GB
Block Size	8B	64B	128B	4-16 KB	



Reikalavimai keliami pagrindinės atminties valdymui

- Patalpinimo vietos pakeitimo galimybė
- Apsauga
- Dalinimasis
- Loginė organizacija
- Fizinė organizacija

Efektyvus atmintinės valdymas

- Kurį iš procesų palikti pagrindinėje atmintinėje, kai jos pritrūksta naujiems procesams?
- Kokio dydžio atmintinės sritį paskirti kiekvienam procesui?
- Kur pagrindinėje atmintinėje patalpinti kiekvieną procesą?
- ...
- Kur saugoti informaciją apie užimtą, laisvą sritį?

Paprastas atmintinės valdymas

- Nagrinėsime paprastas realizacijas, kurios gali būti naudojamos atmintinės valdymui, kai nėra naudojama virtualios atmintinės sąvoka.
- Jei sistemoje nėra naudojamas perdengimo mechanizmas , tai vykdomo proceso visas programos kodas turi būti pagrindinėje atmintinėje.
- Pagrindinėje atmintinėje turi būti ir operacinės sistemos kodas.

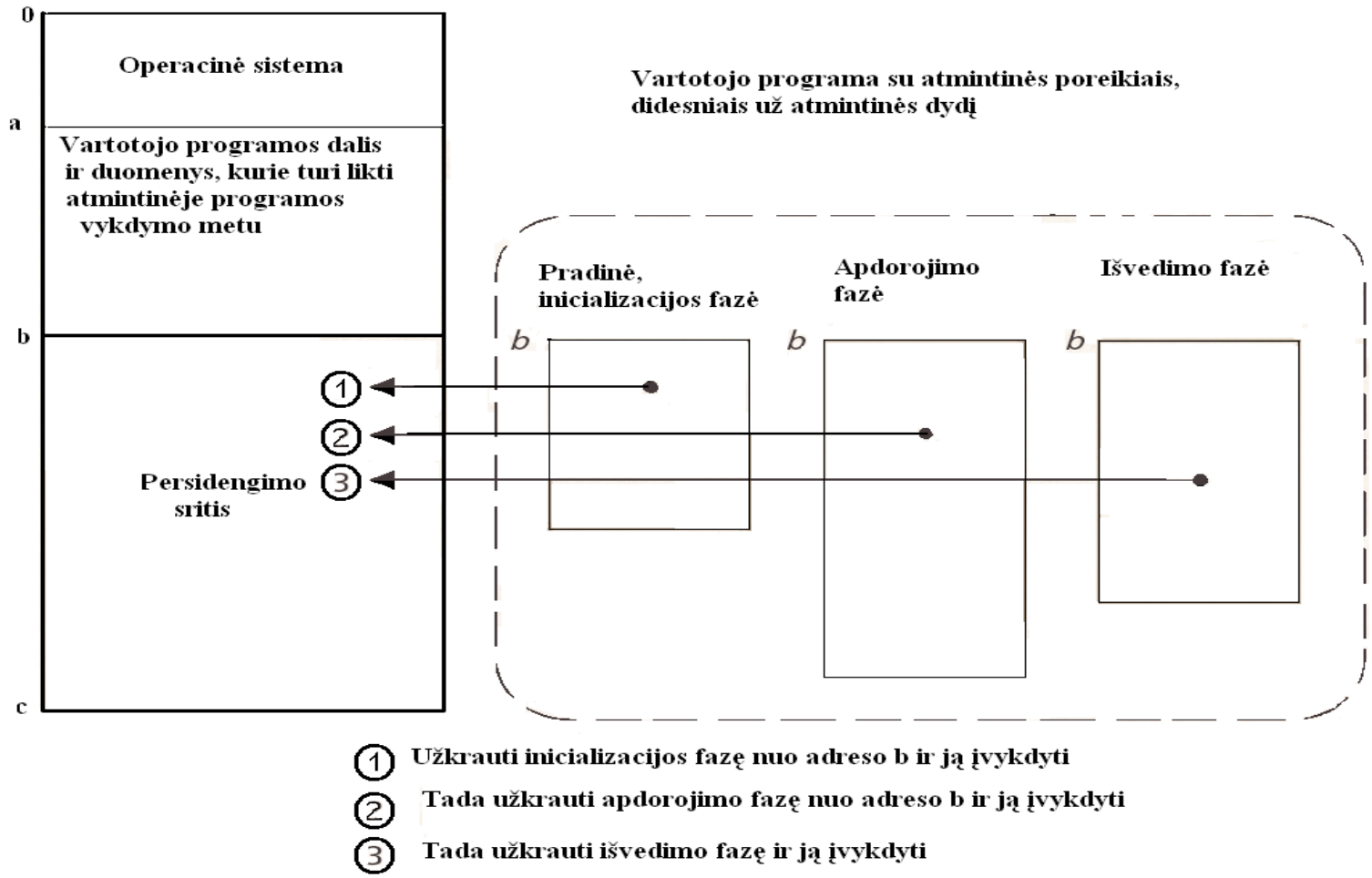
Windows operacinės sistemos poreikiai atmintinei

Operacinė sistema	Išleidimo metai	Minimalūs atmintinės reikalavimai	Rekomenduotinas atmintinės dydis
Windows 1.0	1985	256KB	
Windows 2.03	1987	320 KB	
Windows 3.0	1990	896 KB	1MB
Windows 3.1	1992	2.6 MB	4 MB
Windows 95	1995	8MB	16 MB
Windows NT 4.0	1996	32 MB	96 MB
Windows 98	1998	24 MB	64 MB
Windows ME	2000	32 MB	128 MB
Windows 2000 Professional	2000	64MB	128 MB
Windows XP Home	2001	64MB	128 MB
Windows XP Professional	2001	128 MB	256 MB

Vietos procesui skyrimas atmintinėje

- **Nuoseklių** (ištisinių adresų) zonos skyrimas
  - procesas egzistuotų kaip vientisas, nuoseklių adresų erdvėje esantis blokas.
  - tai labai paprastas atmintinės dalinimo būdas,
  - atsiranda problemos:
    - tinkamo dydžio laisvo bloko atradimas
    - netinkamas atmintinės panaudojimas.
- **Neištisinės** srities skyrimas
  - procesas, jo duomenys skaldomi tam tikro dydžio, atskirose atmintinės vietose talpinamais gabalais (puslapiais, segmentais).
  - lengviau atrasti tinkamas proceso patalpinimui vietas atmintinėje,
  - leidžia padidinti procesų, vienu metu esančių pagrindinėje atmintinėje kiekį,
  - realizacija yra sudėtingesnė.

Ištisinė sritis ir *perdengimo* mechanizmas



Daugiaprogramės, daugiavartotojiškos sistemos

- Prireikė algoritmų, kurie leistų paskirstyti atmintinę kelioms programoms (keliems procesams).
- Paprasti algoritmai, naudojami skirstant pagrindinę atmintinę yra šie:
  - Fiksuoto dydžio skyriai
  - Dinaminis skyrių formavimas
  - Paprastas puslapiavimas
  - Paprasta segmentacija

Fiksuoti skyriai

- Pagrindinė atmintis yra sudaloma į eilę nepersidengiančių skyrių (dalių). Šios dalys gali būti tiek vienodo tiek skirtingo dydžio.
- Procesas, kurio dydis yra mažesnis arba lygus skyriaus dydžiui, gali būti patalpinamas į šį skyrių.
- Procesorius gali greitai persijungti tarp procesų.
- Naudojami keli ribiniai registrai apsaugai nuo to, kad procesai negadintų vienas kito duomenų ar programos, kreipdamiesi į ne jam skirtą atmintinės bloką – tokie kreipiniai neleidžiami.
- Jei visi skyriai yra užimti, operacinė sistema gali iškelti (swap) procesą iš jo užimamo skyriaus.

Vienodo dydžio skyriai

Opeacinė sistema 8M	8M	8M	8M	8M

Skirtingo dydžio skyriai

Opeacinė sistema 8M	2M	4M	6M	8M	12M

Fiksuoti skyriai

- Atsiranda vidinės fragmentacijos problema:
  - nes nežiūrint kokia maža programa būtų – jai skiriamas visas skyrius ir jame gali būti daug nenaudojamos vietos.
  - Skirtingo fiksuoto ilgio skyriai kiek sumažina šią problemą, tačiau problema išlieka.

Skirtingo dydžio skyriai

Opeacinė sistema 8M	2M	4M	6M	8M	12M

Procesai	Poreikiai atmintinei
A	64 KB
B	2,48 MB
C	8,12 MB

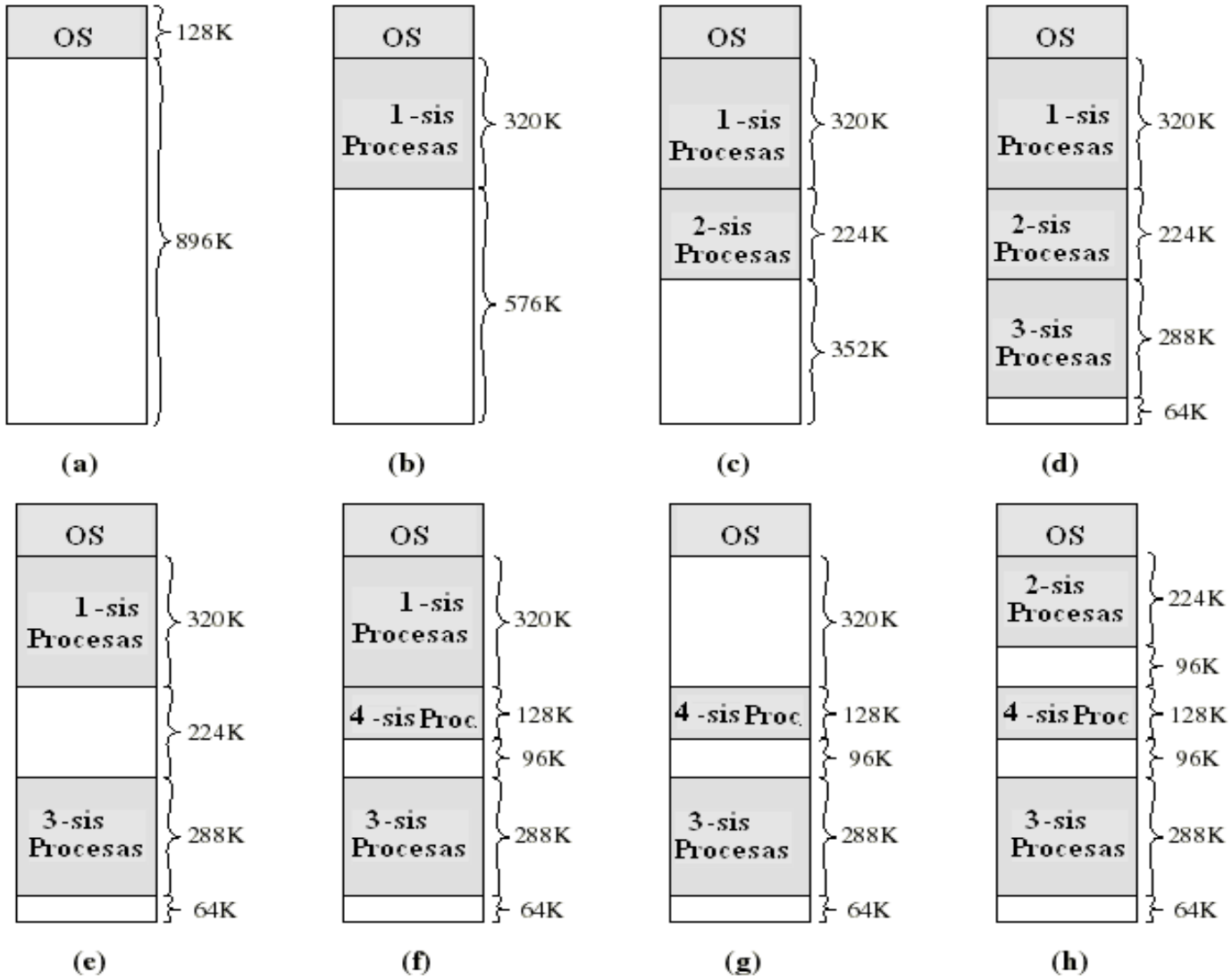
Fiksuoti skyriai ir proceso patalpinimas

- Esant laisvam (neužimtam) skyriui, procesas gali būti patalpinamas į jį.
- Kai visi skyriai yra vienodo dydžio, tai:
  - visai nėra svarbu, į kurį skyrių procesas bus patalpintas.
  - Jei visi skyriai yra *užimti blokuotais procesais*, tai galima parinkti vieną iš jų ir laikinai iškelti į antrinę atmintį (swap) atlaisvinant vietą kitam procesui.
- Kai skyriai yra skirtingo dydžio:
  - Gali būti naudojama daug eilių , kurių kiekviena yra surišta su atskiru skyriumi,
  - Problema gaunasi tame, kad eilė skyrių, o tuo pačiu ir eilių gali būti tuščios, jei neatsiranda atitinkamo dydžio procesų, kai tuo tarpu kitos eilės gali būti pakankamai didelės

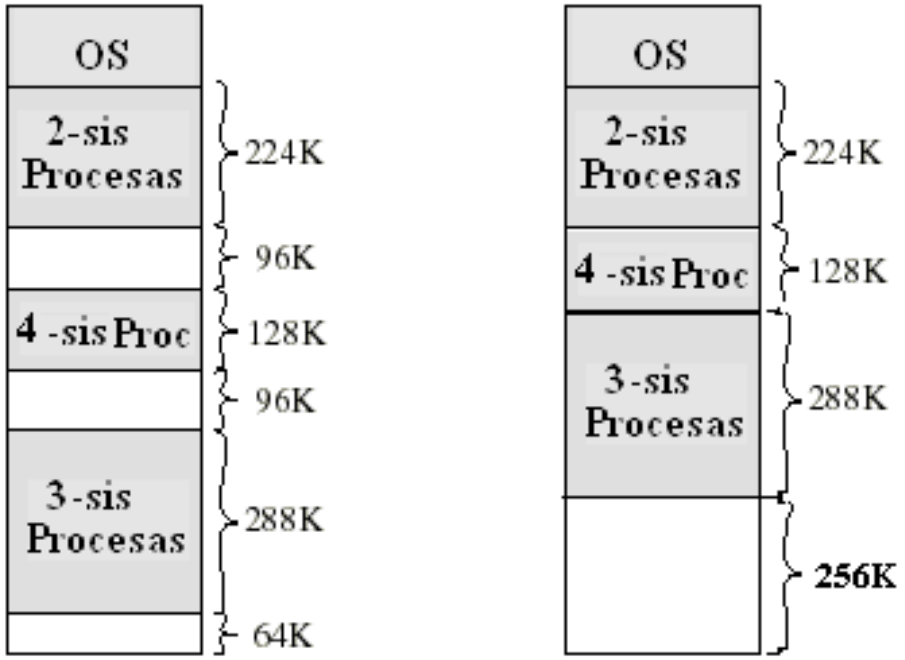
Dinaminis skyrių formavimas

- Taikant šį principą skyrių *kiekis*, jų *dydis* yra kintami.
- Kiekvienam procesui jį talpinant pagrindinėje atmintinėje yra išskiriamas *tokio dydžio skyrius, kokio jis prašo*.
- Kai procesas yra užbaigiamas, tai į jo vietą jau galima patalpinti arba tokio paties dydžio arba mažesnį procesą.
- Taip atsiranda „skylės“ pagrindinėje atmintinėje – tai laisvos sritys, kurios lieka didesnio skyriaus vietoje suformavus mažesnį skyrių. Tai yra vadinama **išorine fragmentacija**:
  - nors ir yra laisvos atminties, tačiau nauji procesai nebetelpa.
  - reikia panaudoti kokį nors **suspaudimo** algoritmą

Dinaminis skyrių formavimas



Suspaudimas



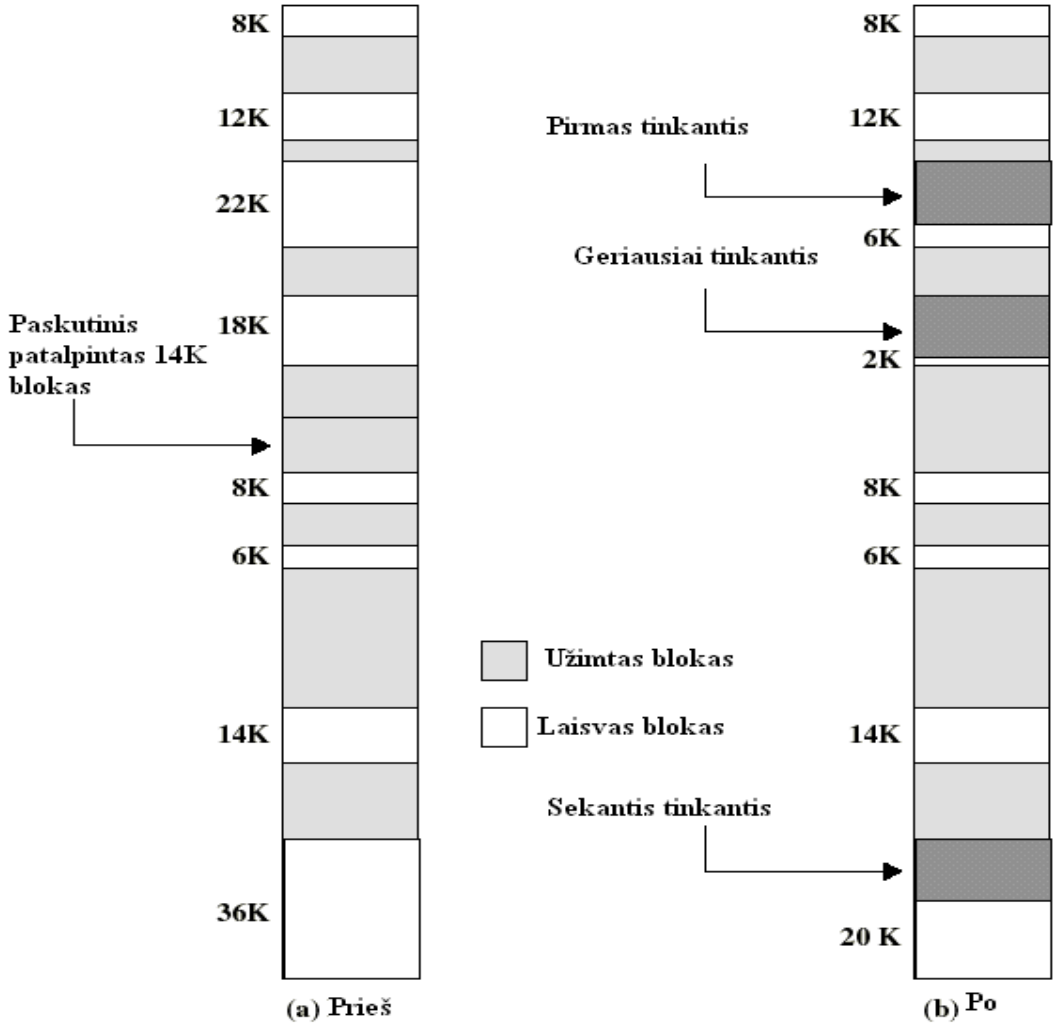
Talpinimo algoritmai dinaminių skyrių atveju

- Talpinimo algoritmas yra taikomas, siekiant nuspręsti, kurį laisvą atmintinės skyrių („skylę“) paskirti procesui, jį talpinant pagrindinėje atmintinėje.
- Algoritmu siekiama patalpinimą vykdyti taip, kad rečiau reikėtų naudoti suspaudimą, nes jis reikalauja laiko.
- Galimi ir naudojami šie algoritmai:
  - *Geriausiai tinkančio*
  - *Pirmo tinkamo*
  - *Sekančio tinkančio*.

Talpinimo algoritmai dinaminių skyrių atveju

- Taikant *geriausiai tinkančio* paiešką, yra randama mažiausias laisvas atmintinės blokas („skylė“), į kurį telpa talpinamas procesas:
  - lieka mažiausia neišnaudota erdvė,
  - algoritmas reikalauja daugiau laiko tinkamos „skylės“ paieškai.
- *Pirmo tinkamo* atveju yra surandamas (nuo atmintinės pradžios) pirmas laisvas, tinkantis savo dydžiu blokas, ir į jį talpinamas procesas.
  - Šis algoritmas realizuojamas gana paprastai,
  - Nereikalauja daug veiksmų.
- *Sekančio tinkančio* atveju yra randama pirma tinkama dydžiu skylė, einanti po paskutinio patalpinimo.

Talpinimo algoritmų pavyzdys, talpinant 16K procesą atmintinėje



Komentarai

- Taikant „*sekantis tinkamas*“ algoritmą dažnai naujai talpinamam procesui yra priskiriamas didžiausias blokas, esantis pagrindinės atmintinės pabaigoje.
- Taikant *pirmo tinkamo* paiešką, skylės radimas sukasi apie atmintinės pradžią, jį taikant gaunama mažesnė fragmentacija nei „sekančio tinkamo“ atveju.
- „*Geriausiai tinkančio*“ paieška yra susijusi su mažiausio neužimto bloko suradimu: likęs laisvas fragmentas bus gaunamas mažiausias.
- Kartais yra naudojamas algoritmas, kuris proceso patalpinimui ieško blogiausiai tinkančios savo dydžiu „skylės“. Randamas didžiausias savo apimtimi blokas, į kurį patalpinus procesą jame lieka didžiausia neišnaudota erdvė. Yra tikimasi, kad į šią neišnaudotą erdvę vėliau bus galima patalpinti kitą procesą.
- Pagrindinėje atmintinėje labai greitai gaunamos skylės, kurios yra per mažos bet kokio proceso patalpinimui, ir reikia atlikti suspaudimus.

Vietos procesui skyrimas atmintinėje

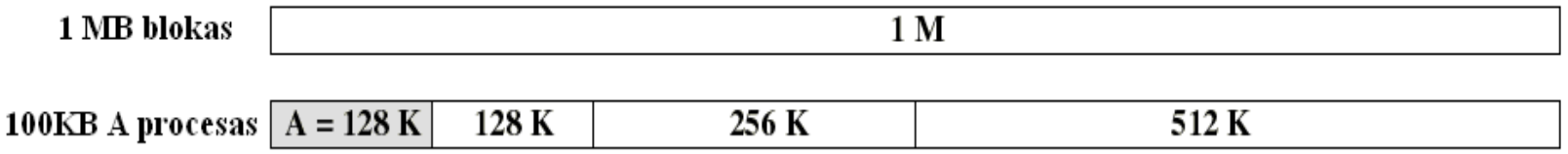
- **Nuoseklių** (ištisinių adresų) zonos skyrimas
  - procesas egzistuotų kaip vientisas, nuoseklių adresų erdvėje esantis blokas.
  - tai labai paprastas atmintinės dalinimo būdas,
  - atsiranda problemos:
    - tinkamo dydžio laisvo bloko atradimas
    - netinkamas atmintinės panaudojimas.
- **Neištisinės** srities skyrimas
  - procesas, jo duomenys skaldomi tam tikro dydžio, atskirose atmintinės vietose talpinamais gabalais (puslapiais, segmentais).
  - lengviau atrasti tinkamas proceso patalpinimui vietas atmintinėje,
  - leidžia padidinti procesų, vienu metu esančių pagrindinėje atmintinėje kiekį,
  - realizacija yra sudėtingesnė.

Bičiuliška sistema (Buddy System)

- Bičiuliška sistema, tai algoritmas, kuriuo bandoma apeiti tiek fiksuotų, tiek dinaminių skyrių problemas.
- Modifikuota šio algoritmo versija yra naudojama UNIX SVR4 .
- Atmintinės blokai, kurie yra išskiriami procesams yra **2^{K}** dydžio

Algoritmo žingsniai

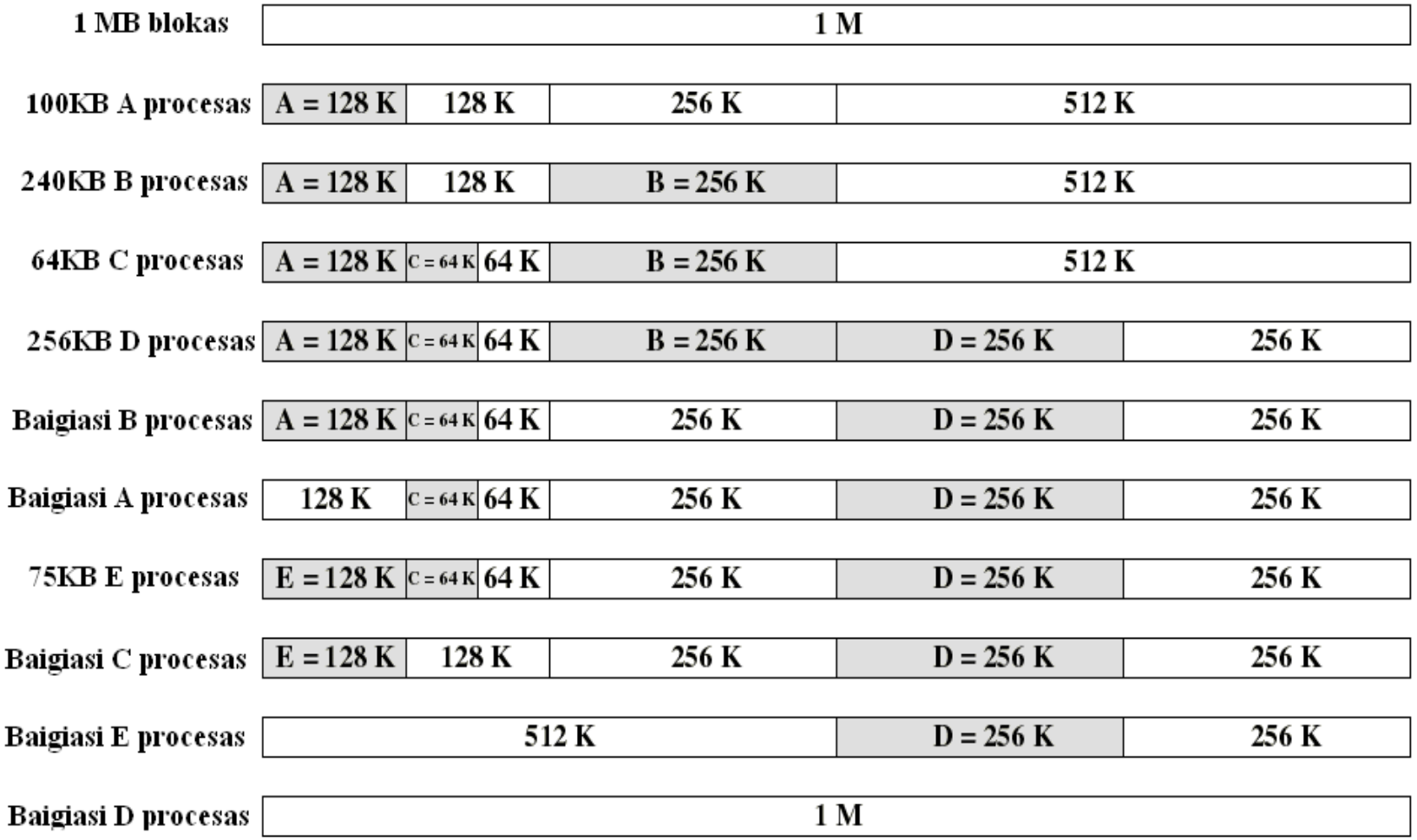
- Pradžioje visa atmintinė yra laisva, taigi pradedama turint **2^{U}** dydžio bloką. Tarkime, atsiranda pareikalavimas patalpinti **S** dydžio procesą.
  - Jei **2^{U-1} < S <= 2^{U}**, tai yra išskiriamas visas blokas **2^{U}**.
  - Priešingu atveju blokas sudalomas į dvi vienodo dydžio **2^{U-1}** dalis (bičiulius).
    - Jei **2^{U-2} < S <= 2^{U-1}**, tai procesui išskiriama viena iš dalių (vienas bičiulis), o jei ne, tai viena iš dalių vėl yra daloma į dvi dalis.
    - Šis procesas yra kartojamas tol, kol gaunamas mažiausias blokas, kuris yra lygus arba didesnis nei **S**.
    - Pavyzdys: Turim 1MB= 2^{10} KB=1024KB . Procesas prašo 100 KB
    - 2^9=512<100< 2^{10} =1024 – netinka
    - 2^8=256<100<2^9=512 –netinka
    - 2^7=128<100<2^8=256 –netinka
    - 2^6= 64<100<2^7=128- tinka, skiriam 128 KB



Bičiuliška sistema

- Jei du bičiuliai tampa laisvais , bičiuliai yra apjungiami.
- Operacinė sistema palaiko keletą sąrašų apie esančias skyles.
  - **i**-tas sąrašas apima skyles, kurių dydis yra **2^{i}**.
  - Kai tik pora bičiulių atsiranda **i**-tame sąraše, jie yra išmetami iš šio sąrašo ir apjungiami į vieną bendrą skylę **(i+1)** sąraše.
- Atsiradus naujai užklausai, t.y. norint patalpinti **k** dydžio procesą, tokį:
  - kuriam galioja: **2^{i-1} < k <= 2^{i}**
  - yra patikrinamas **i**-tas sąrašas. Jei šis sąrašas yra tuščias, tikrinamas **(i+1)** sąrašas.
  - Jame radus skylę ji bus sudaloma į du bičiulius, viena iš dalių bus priskirta procesui, o kita įtraukta į **i**-tą sąrašą.

Bičiulių sistemos taikymo pavyzdys



Virtuali atmintinė

- Virtuali atmintinė reiškia tai, kad kiekvienos programos požiūriu galim galvoti, kad galim kreiptis į visą atmintinės adresų sritį, naudodamiesi adresavimo schema.
- Vienintelis ribojimas yra bitų kiekis nusakant atmintinės adresą.
- Atmintinėje turi rastis tiek programos kodas tiek ir duomenys.
- Jei naudojama 32-bitų adresavimo schema, tai bus leistini adresai iki 2^{32}.
- Kiekvienam vartotojui leidžiant galvoti, kad realiai tiek vietos yra kompiuteryje, OS iš tikrųjų atmintinėje laiko tik tam tikras programų bei duomenų dalis. Likusi dalis yra saugoma papildomoje atmintinėje (diskuose).
- Virtuali atmintinė tai yra realios atmintinės ir papildomos atmintinės kombinacija.

Virtualios atmintinės sąvoka leidžia išspręsti eilę problemų:

- Programuotojai gali rašyti programos kodą *nesirūpindami* apie fizinę atminties struktūrą.
- Leidžia vykdyti procesus, kurių *pareikalavimai* pagrindinės atmintinės dydžiui yra didesni nei jos dydis.
- Procesai gali vykdyti programas, kurių kodas yra tik ***dalinai*** patalpintas pagrindinėje atmintinėje.
- Programos gali būti ***kilojamos*** (relocatable), t.y. jos gali būti patalpinamos bet kur fizinėje atmintinėje.

- Procesams leidžiama ***dalintis*** viena bibliotekos ar programos kopija, įkelta į pagrindinę atmintinę.

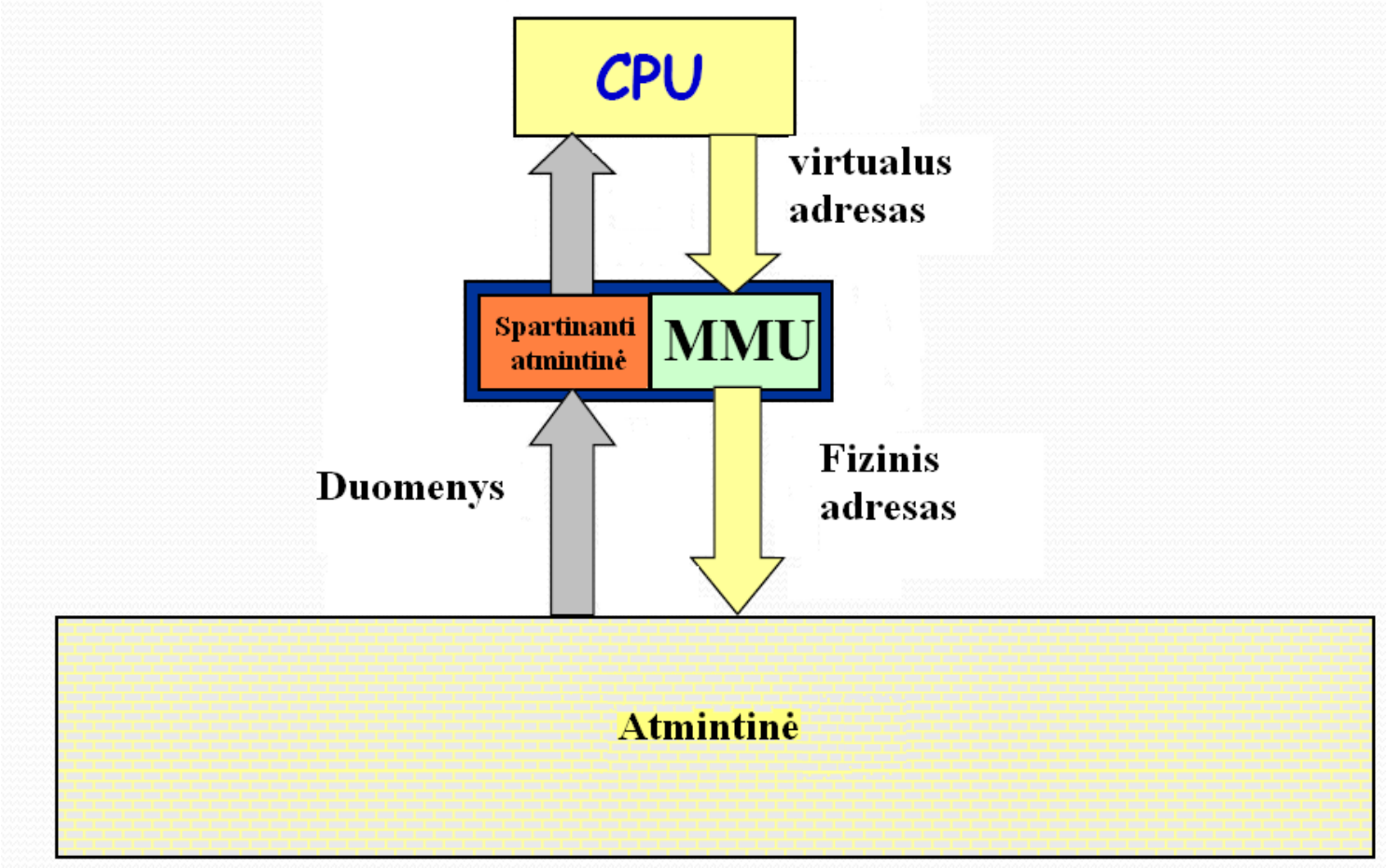
Adreso transliavimas

- **Fizinis adresas** (absoliutus adresas) nurodo objekto fizinę vietą pagrindinėje atmintinėje.
- **Virtualus adresas** tai nuoroda į objekto vietą, kuria operuoja procesas, nepriklausanti nuo to, kur atmintinėje jis yra patalpintas, kokia adresų forma yra naudojama kompiuteryje.

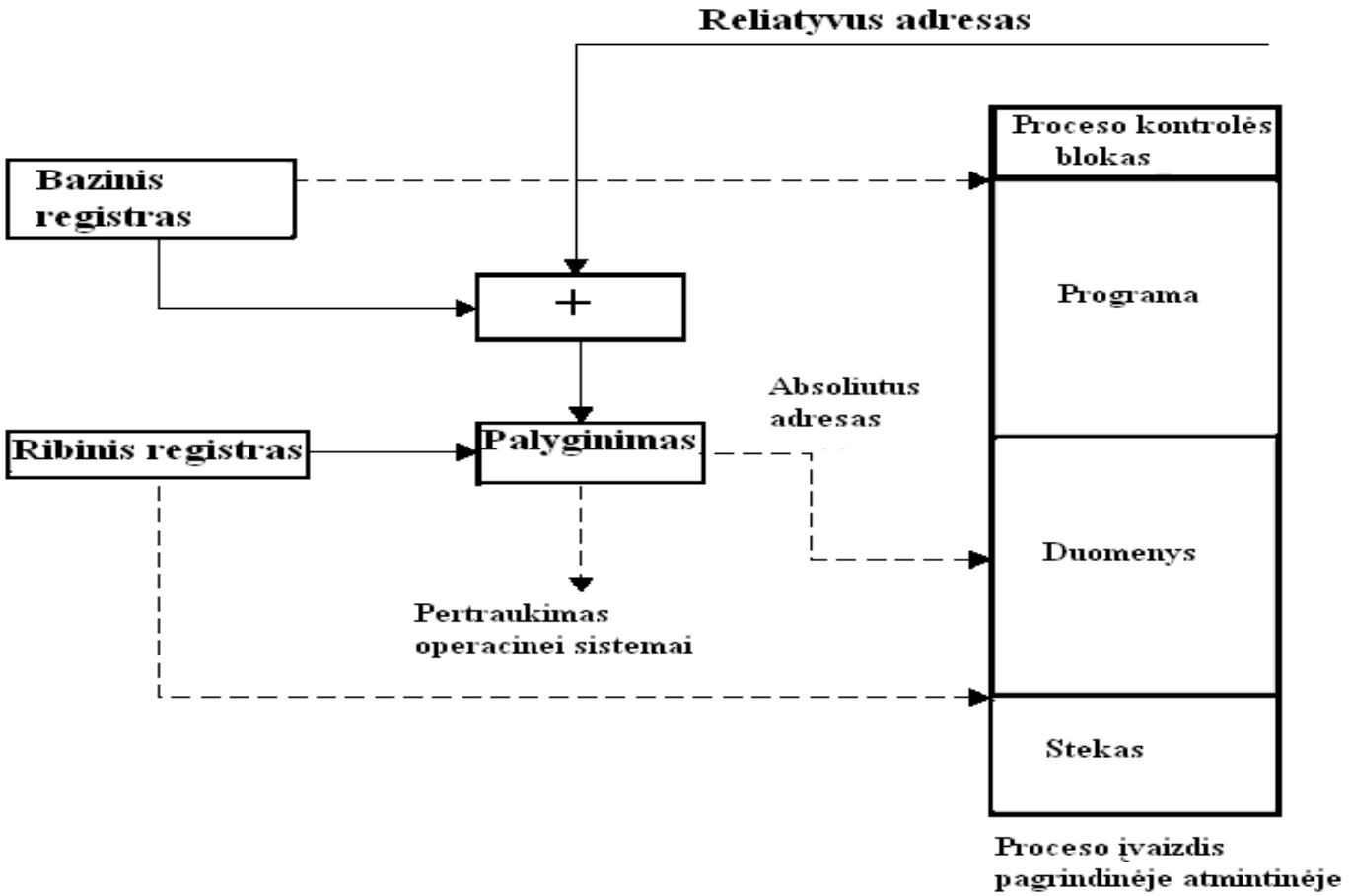
Bazinis, reliatyvus adresas

- Kompiliatoriai gamina kodą, kuriame visos nuorodos į atmintinę yra išreiškiamos **loginiais** adresais.
- **Reliatyvus adresas** yra loginio adreso pavyzdys, kai adresas yra išreiškiamas atžvilgiu kažkurio žinomo programos **taško-bazinio adreso** (pavyzdžiui, pradžios).
- **Fiziniai** (realūs) adresai yra išskaičiuojami „eigoje“, kai yra vykdomos komandos su reliatyviais adresais.

Adreso transliacija



Reliatyvaus adreso transliacija į fizinį adresą



Rezidentinė arba darbinė proceso dalis

- Į pagrindinę atmintinę įkelta proceso dalis dar yra vadinama **rezidentine** arba **darbine** dalimi.
- Kodėl gi proceso vykdymui pakanka mažesnės srities nei viso proceso dydis?
  - Tai surišta su *lokališkumo* principu, kuris pasireiškia tuo:
    - dažniausiai didelę proceso vykdymo laiko dalį procesas vykdo nedidelę komandų aibę (vyksta ciklas)
    - naudoja greta esančius duomenis,
    - dauguma skaičiavimų yra vykdomi nuosekliai.
  - Todėl bet kuriuo momentu procesui pakanka nedidelės rezidentinės srities pagrindinėje atmintinėje.

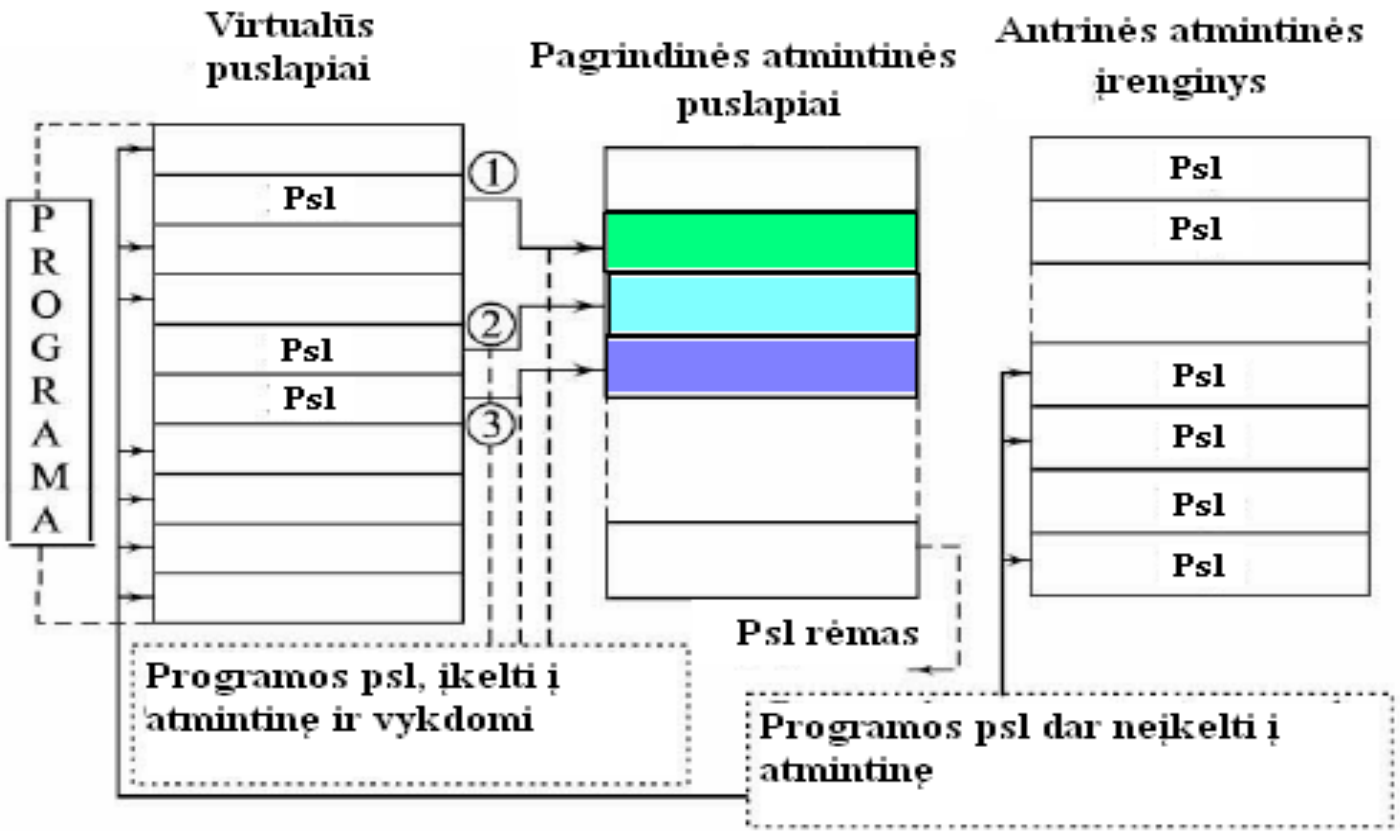
Skaidymas puslapiais

- Kiekvienas procesas yra suskaldomas į vienodo dydžio puslapius.
  - Kai kurios operacinės sistemos puslapio dydį sutapatina su bloko dydžiu ar sektoriaus dydžiu diske.
- Pagrindinė atmintinė taip pat yra sudaloma į fiksuoto, puslapio dydžio blokus, vadinamus rėmais (frames).
  - Jie yra reliatyviai mažo dydžio lyginant su visos atmintinės dydžiu.
- Taip skirstant viename disko sektoriuje bus vienas proceso puslapis, kuris tilps į vieną pagrindinės atmintinės rėmą.
- Dauguma sistemų naudoja 2 - 8 KB puslapio dydžius.

Fragmentacija ir puslapiai

- Nebelieka išorinės fragmentacijos :
  - Proceso puslapiai proceso talpinimo į pagrindinę atmintinę metu gali užimti laisvus rėmus (page frames).
  - Nauda:
    - procesui nebūtina užimti ištisinės adresų erdvės pagrindinėje atmintinėje
    - proceso puslapiai gali būti išbarstomi į egzistuojančius laisvus rėmus.
  - Procesui pasibaigus, tiesiog padaugėja laisvų rėmų.
- Kadangi puslapio (rėmo) dydis yra pakankamai nedidelis, tai sumažėja ir vidinės fragmentacija.

Programos įkėlimas



Puslapių lentelė

- Proceso puslapiai gali būti išdėstomi bet kur pagrindinėje atmintinėje.
- Valdant atmintinę tenka sekti, kur yra išdėstyti kiekvieno proceso puslapiai, kokie yra likę laisvi rėmai .
- Operacinė sistema priversta laikyti pagrindinėje atmintinėje *kiekvieno aktyvaus proceso puslapių lentelę*.
  - Kiekvienas *puslapių lentelės įrašas* - tai nuoroda į numerį rėmo, kuriame atitinkamas puslapis yra realiai patalpintas.
    - Tai vieno lygio puslapių lentelės- jose realių ir virtualių puslapių atitikmuo.
    - Kiekvienas procesas turi savo puslapių lentelę.
  - Į puslapių lentelę yra kreipiamasi nurodant proceso virtualaus puslapio NR, siekiant nustatyti, kokiame realiame rėme šis puslapis yra pagrindinėje atmintinėje.

Puslapių lentelė

- *Tai atmintinėje esanti duomenų struktūra, kurioje saugomas:*
  - Virtualių adresų atvaizdavimas į fizinius adresus.
- Puslapių lentelė leidžia nustatyti, ar virtualus puslapis, į kurį vyksta kreipinys yra atvaizduotas į fizinį puslapį (yra pagrindinėje atmintinėje).
- Viename iš CPU registrų visada yra tuo metu vykdomo proceso puslapių lentelės *pradžios fizinis adresas.*

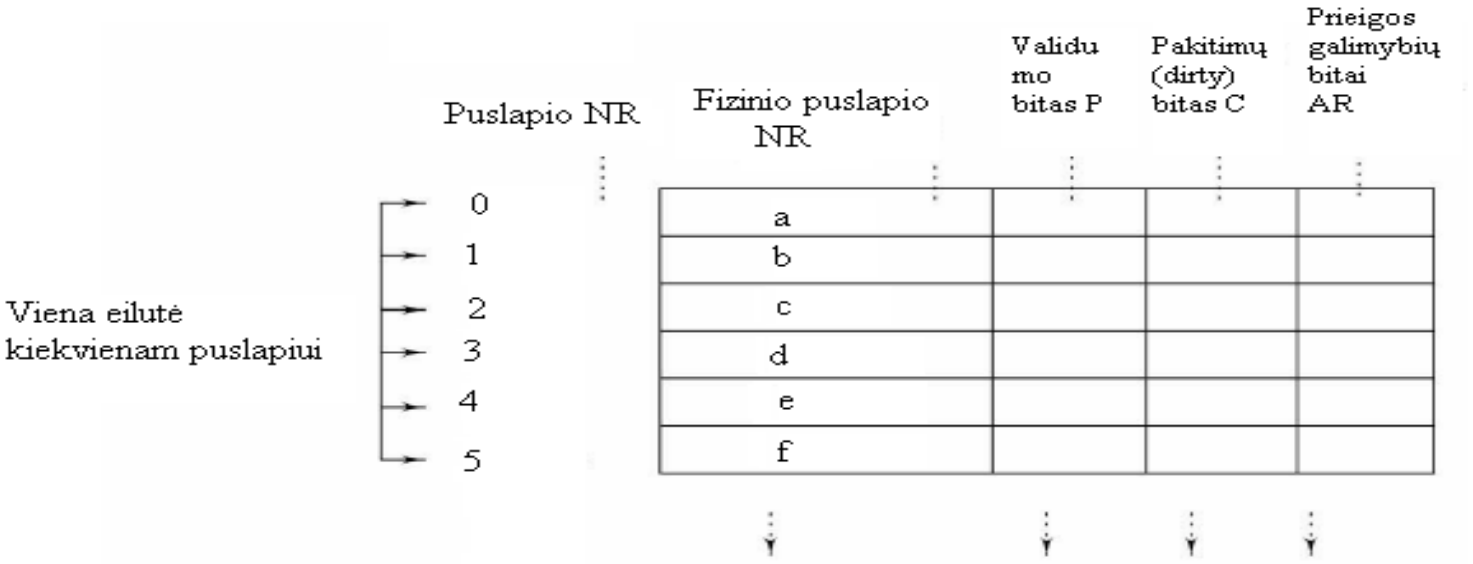
Puslapių lentelė

Sistemos, kurios naudoja vieno lygio puslapių lenteles, paprastai reikalauja, kad visa puslapių lentelė visą laiką būtų saugoma pagrindinėje atmintinėje.

OS gali pasiekti šią lentelę adreso transliacijai

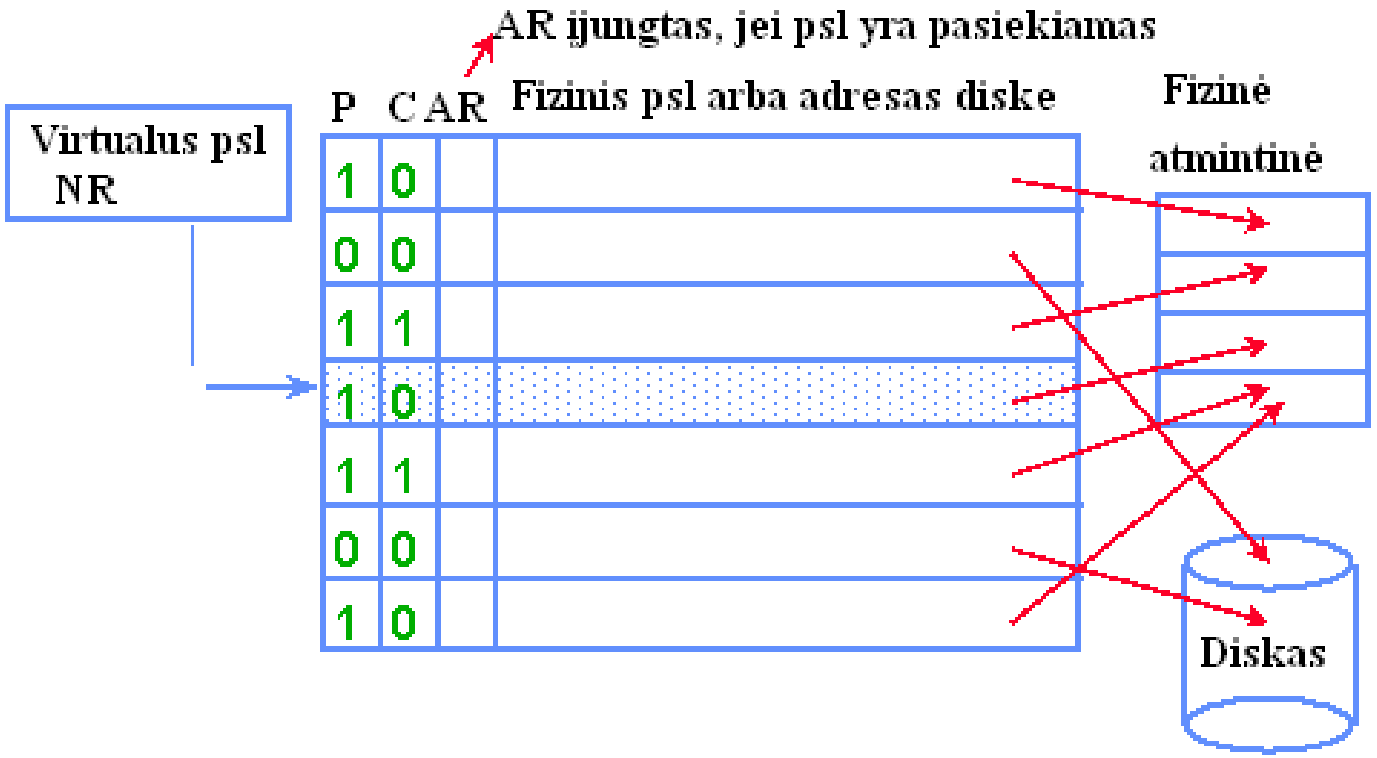
Kai programoje sutinkamas loginis adresas, yra kreipiamasi į puslapių lentelę ir yra nustatomas atitinkamo rėmo numeris.

Puslapio numeris tarnauja puslapių lentelės indeksu



a valid bit, P (can also be called presence bit), change bit, C (can also be called dirty bit), and access rights, AR bits

Puslapio išrinkimas



Puslapiai ir loginiai (virtualūs) adresai

- Dvejetainėje adreso išraiškoje dalis adreso bitų yra skiriama *puslapio numeriui* užduoti ir likusi dalis – *poslinkiui*.
- Kiekvienas loginis (virtualus) adresas yra išreikštas puslapio numeriu bei poslinkiu tame puslapyje.
  - objekto adresas yra nusakomas pora (P,R),
  - P - yra atitinkamo puslapio numeris,
  - R - poslinkis puslapio pradžios atžvilgiu.

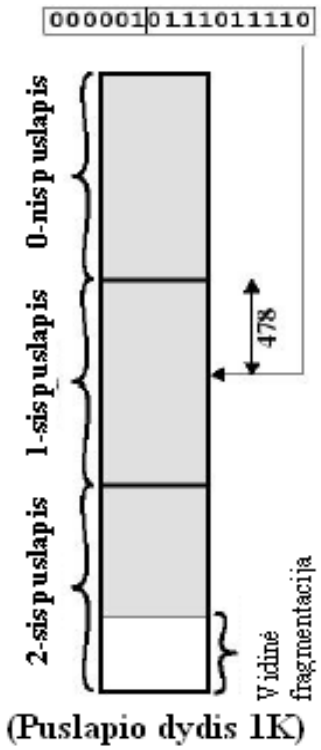
  - Poslinkio reikšmė nusako objekto vietą puslapio rėme.

Loginio adreso transliavimas į fizinį adresą

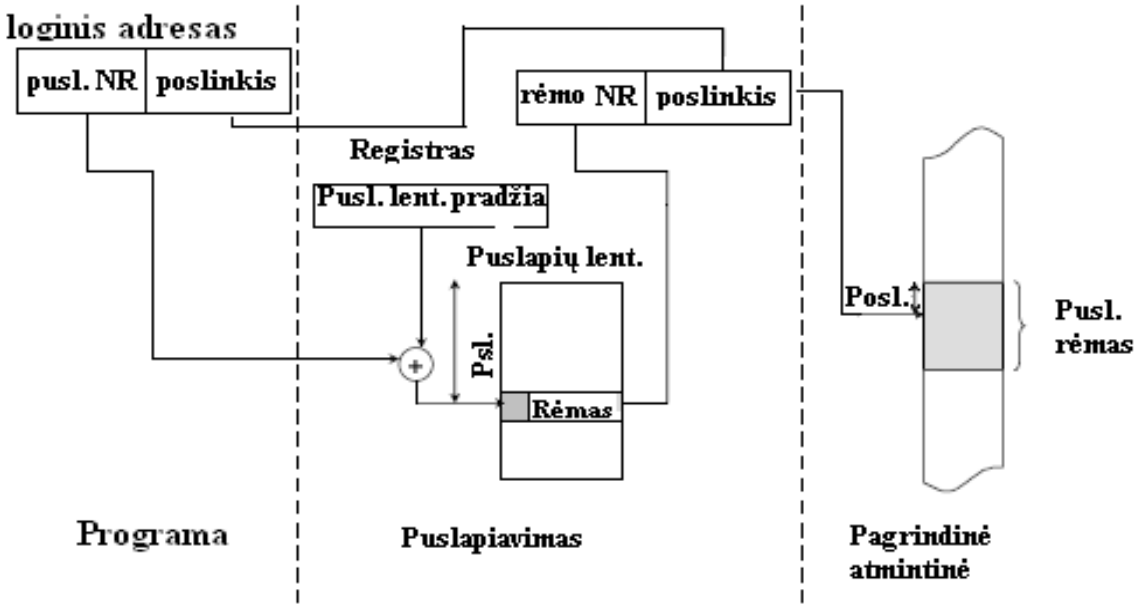
16 bitų adresas
0000010111011110

6 bitai –psl,10 bitų -poslinkis

Loginis adresas:
1-puslapis, 478-poslinkis



(Puslapio dydis 1K)



Puslapių lentelės ir reikalavimai atmintinei

- Tarkim, sistema naudoja 64-bitų adresus ir 4-kB puslapius.
- Viso gali būti 2^{52} puslapių lentelės įėjimai
- Jei kiekvieno įėjimo ilgis 7 baitai, tai,
- $7 * 2^{52}$ B atminties (apie 30,000,000 GB) reikės vienos programos puslapių lentelės saugojimui.

Puslapio dydis

- Jei puslapio dydis bus mažas, tai sumažėtų vidinė fragmentacija, tačiau tada išaugtų puslapių lentelės dydis.
- Imant didesnį puslapio dydį:
  - sumažinamas puslapių lentelės dydis,
  - darosi patogesni informacijos mainai su diskiniu įrenginiu.
  - reikia mažiau puslapių mainų.
  - Paprastai naudojami 4K, 8K arba 16K puslapių dydžiai.
- Jei puslapių lentelės yra labai didelės, tai galima jos visos nelaikyti pagrindinėje atmintinėje.

Rezidentinis proceso dydis

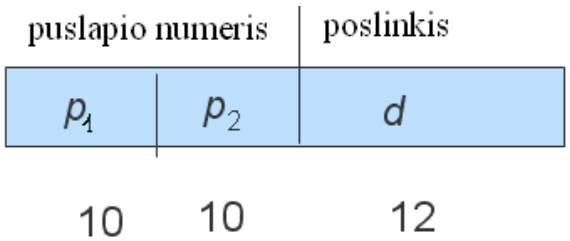
- ***Kiek*** pagrindinės atmintinės rėmų skirti kiekvienam iš aktyvių procesų, t.y. kiek puslapių įkelti į pagrindinę atmintinę.
  - Jei į pagrindinę atmintinę bus įkelta nedaug puslapių, tai bus reikalinga dažnai juos keisti.
  - Jei kiekvienam procesui skiriama daug puslapių – tai ribotas pagrindinės atmintinės rėmų kiekis gali apriboti multiprogramavimo lygį.
- Nusakant rezidentinį procesui skirtų rėmų kiekį gali būti :
  - atsižvelgiama į proceso dydį ir rėmų kiekis skiriamas proporcingai procesų dydžiui,
  - gali būti skiriami ir vienodi rėmų kiekiai.

Puslapių lentelių tipai

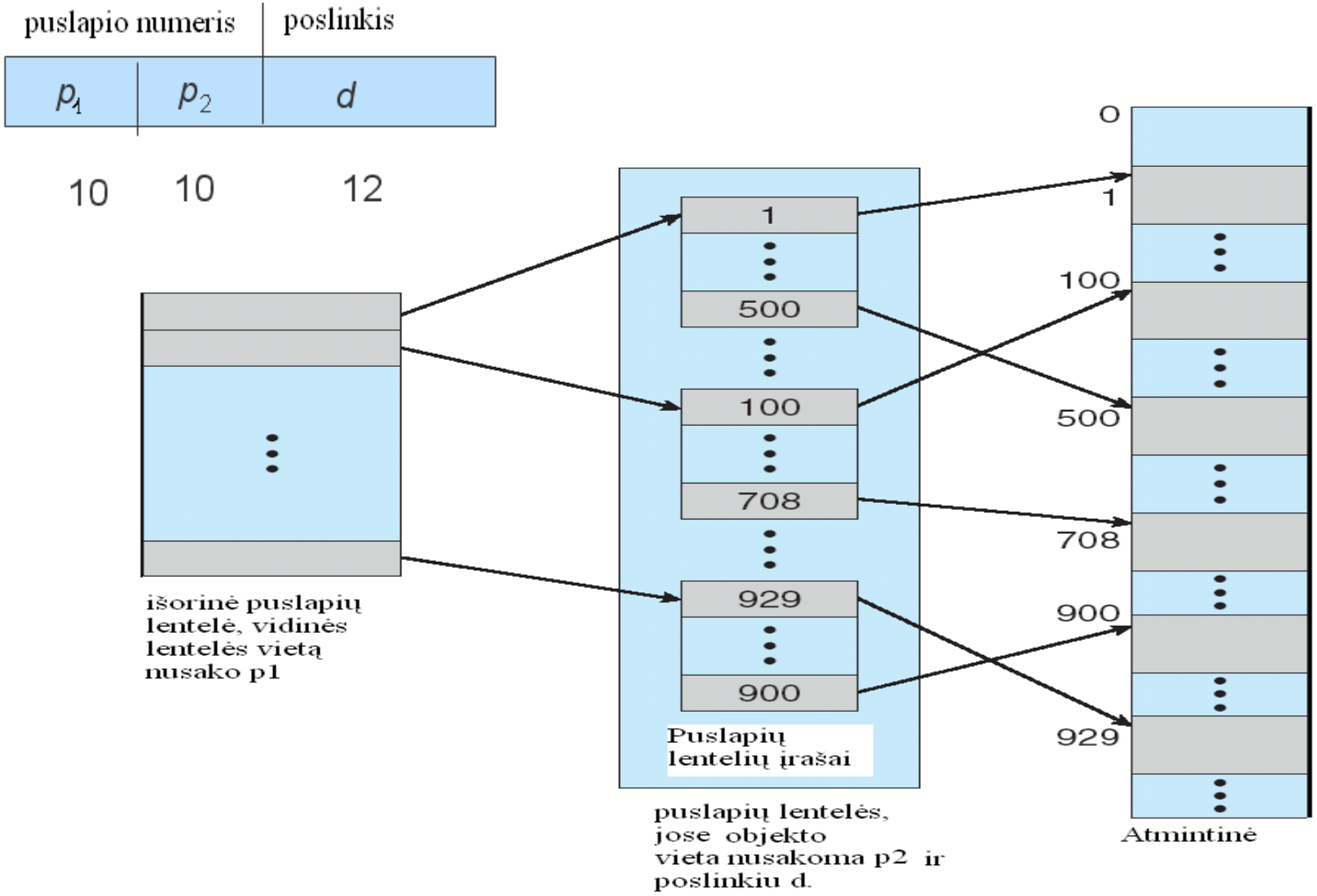
- Vieno lygio,
- hierarchinės,
- invertuotos santraukos (hash) tipo, puslapių lentelės

Hierarchinė puslapių lentelė

- Hierarchinių lentelių atveju visa loginė adresų erdvė yra sudaloma į kelias puslapių lenteles.
  - Dažnai yra naudojamos dviejų lygių hierarchinės puslapių lentelės.
  - Pavyzdžiui, loginis adresas *32 bitų adrese su 4 KB* puslapiais yra skaldomas:
    - į puslapio numerį, kuriam skiriama 20 bitų
    - ir 12 bitų poslinkį.
  - Kadangi pati puslapių lentelė taip pat yra puslapiuojama, tai puslapio numerio bitai yra sudalomi :
    - į 10 bitų puslapių lentelės numerį
    - bei 10 bitų puslapio poslinkį .
  - Puslapių lentelės numeris (p1) yra indeksu išorinėje puslapių lentelėje, jis nurodo vidinę puslapių lentelę o puslapio poslinkis (p2) rodo ieškomo puslapio vietą vidinėje lentelėje.



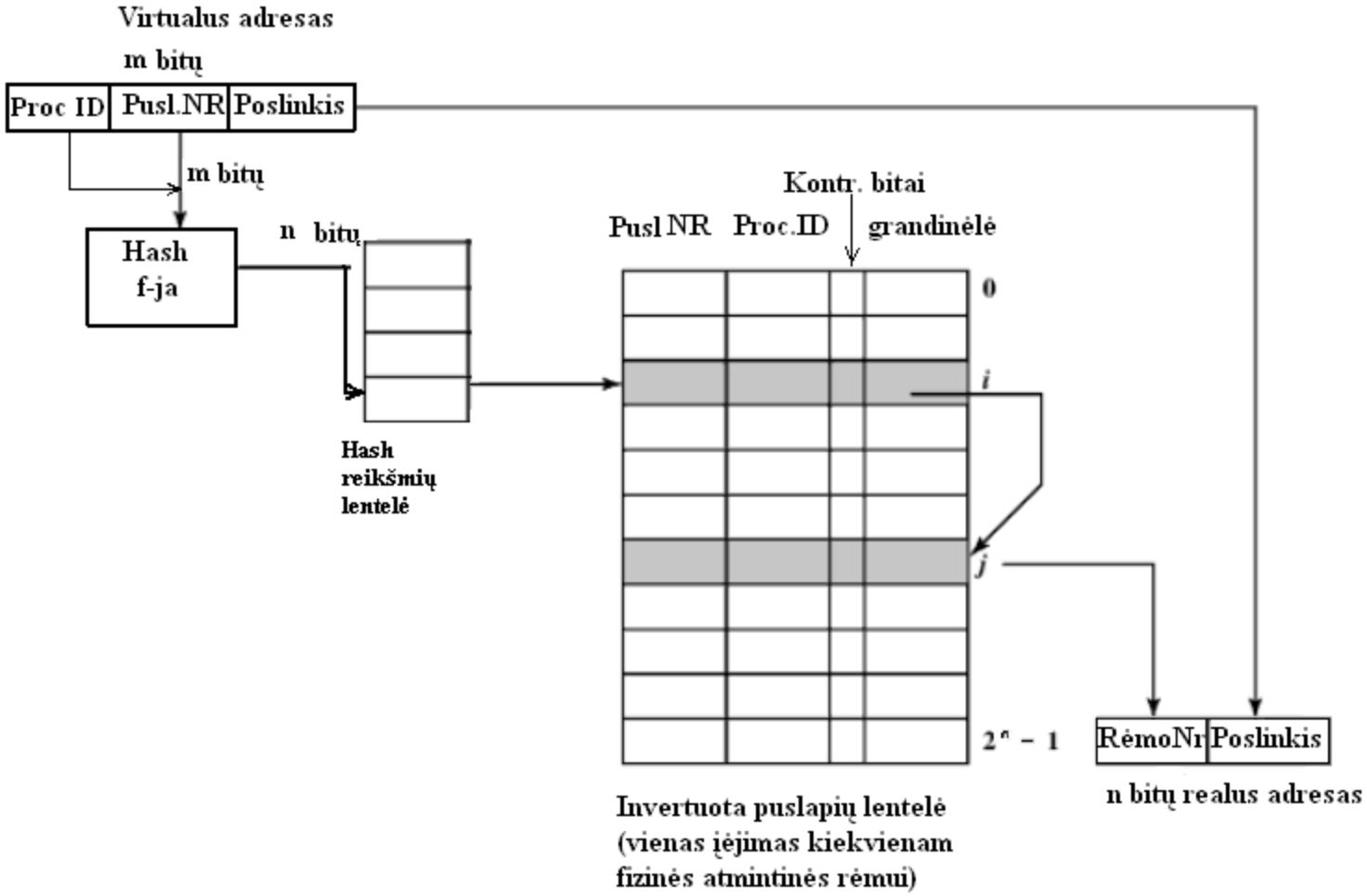
Dviejų lygių hierarchinė puslapių lentelė



Invertuoto (inverted) tipo puslapių lentelės

- Taikant šio tipo lenteles, yra ryškiai sumažinamas puslapių lentelėms reikalingas atmintinės dydis.
- Šiuo atveju puslapių lentelėje yra tiek įėjimų, kiek yra fizinių pagrindinės atmintinės rėmų.
- Kiekvienas įėjimas saugo numerį virtualaus puslapio, kuris patalpintas į šį rėmą. Taigi tokios lentelės dydis priklauso nuo pagrindinės atmintinės dydžio.
- Virtualaus puslapio numeriui taikoma santraukos (hash) funkcija.
  - Tai trumpina paieškos laiką – nereikia lygint su visais lentelės elementais
  - Santraukos funkcijos reikšmė yra naudojama kaip lentelės indeksas.
  - Kadangi daugiau nei vienai įėjimo reikšmei gali būti gaunama ta pati išėjimo (santraukos) reikšmė, tai invertuotoje santraukos lentelėje yra saugoma grandinėlė įrašų, kurias tenka peržiūrėti ieškant tinkamo puslapio.

Invertuoto (inverted) tipo puslapių lentelės



TLB (Translation Look-Aside Buffer) lentelės

- TLB lentelės yra laikomos CPU spartinančioje atmintinėje ir jas naudoja atmintinės valdymo įranga virtualaus adreso transliacijai pagreitinti.
- Kai virtualus adresas sutinkamas pirmą kartą, tai to adreso transliacijai į fizinį adresą yra pasinaudojama puslapių lentele, o atitikimas tarp virtualaus ir fizinio adreso yra išsaugomas TLB lentelėse.
- TLB lentelės saugo atitikimus tarp virtualių puslapių bei jų fizinių adresų.

Adreso transliavimas naudojant TLB įrašus