Atmiņas hierarhija un keši

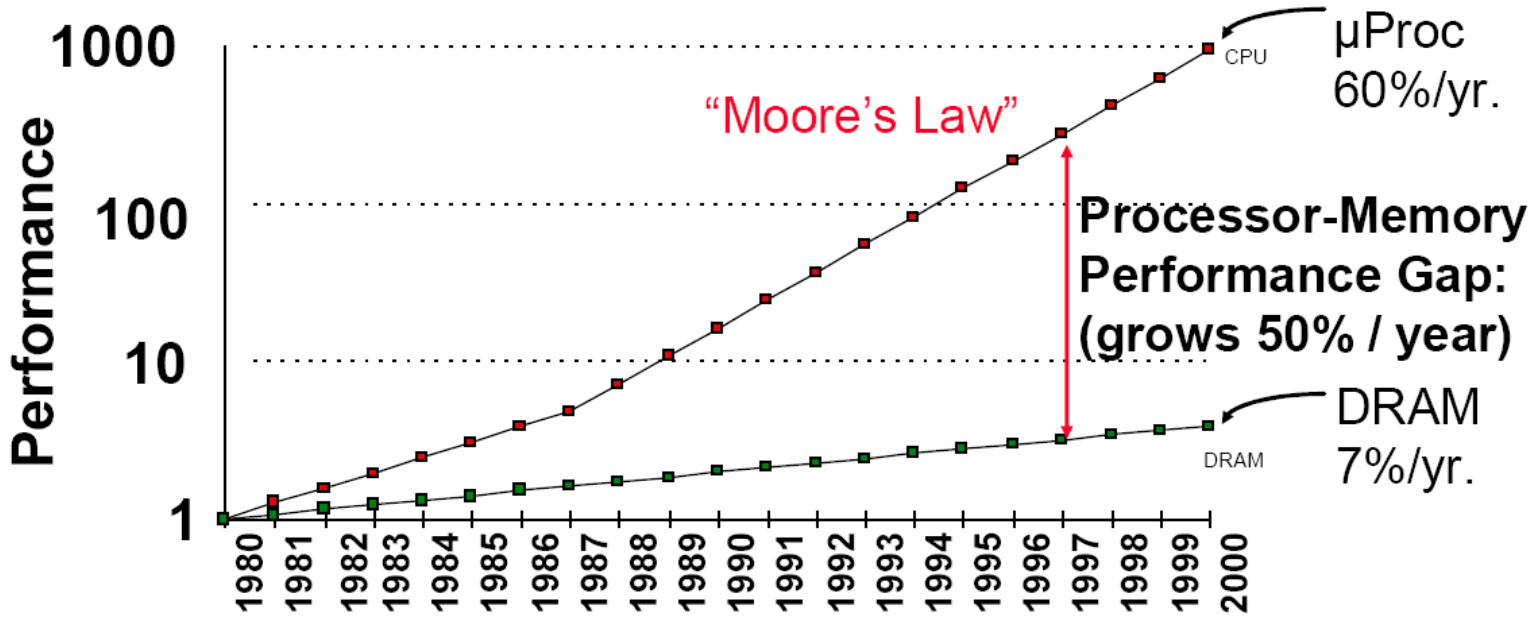
Atmiņas struktūra

Ideāla atmiņa

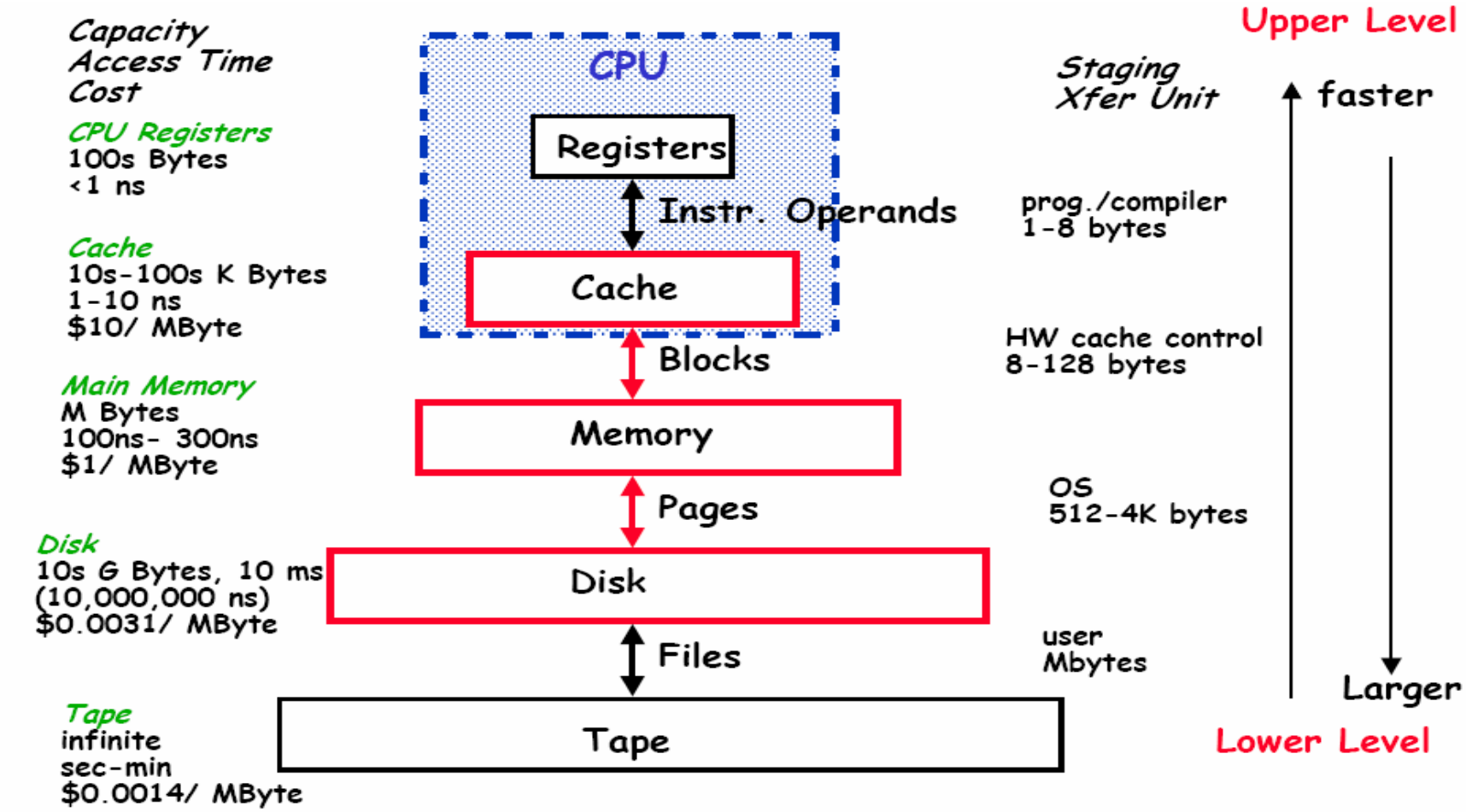
- Bezgalīga apjoma ar ļoti mazu piekļuves laiku
  - Diemžēl tad tā būtu ļoti dārga
  - Praktiski tas ir tehniski neiespējami
- Risinājums ir atmiņas hierarhija kurā ir:
  - ***Lieli*** pēc uzglabājamo datu apjoma bet ***lēni*** mezgli
  - ***Mazi*** pēc uzglabājamo datu apjoma bet ***ātri*** mezgli
- Hierarhijas līmeņus raksturo:
  - Piekļuves laiks - ***latentums***
  - Viena baita izmaksas
  - Kopējā ietilpība
  - Datu pārraides ātrums – ***caurlaides spēja***
  - Pārraides vienība

Kāpēc atmiņa ir tik nozīmīga?

- CPU veiktspēja ir uzlabojusies daudz ātrāk nekā atmiņas veiktspēja. Šodien atmiņas piekļuve ir šaurā vieta.
  - 1980. gadā nebija nekādu kešu CPU
  - 2001. gadā jau bija 2. līmeņa kešs CPU un tas izņēma 90% tranzistoru procesorā.



Atmiņas hierarhija



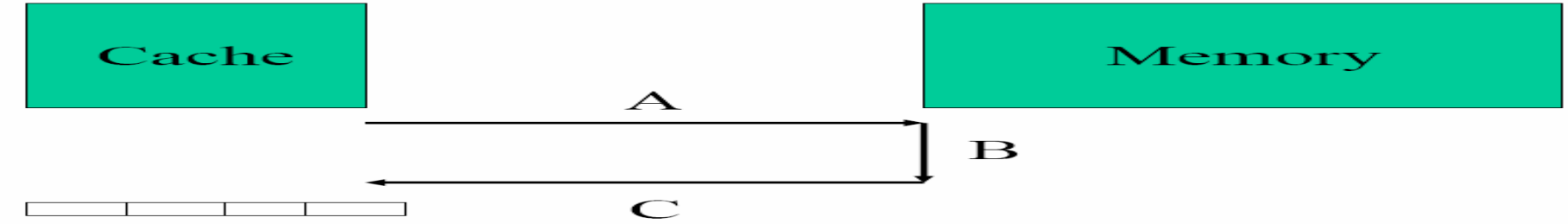
Atmiņas pamatprincipi

- Lokalitāte
  - *Lokalitāte laikā (Temporal Locality)*: atmiņas adreses kurām tika izdarīta piekļuve visdrīzāk tiks izmantotas atkal pēc neilga laika (piemēram cikla kods)
  - *Lokalitāte telpā (Spatial Locality)*: atmiņas apgabali kas pieguļ kādai jau iepriekš apmeklētai adresei visdrīzāk būs vajadzīgi tuvākajā laikā (piemēram piekļuve datu masīviem)
- Lokalitāte + mazāk bet ātrākas aparatūra = atmiņas hierarhija
- *Līmeņi*: katrs augstākais līmenis ir mazāks pēc apjoma, ātrāks, dārgāks/bitu
- *Ietveroši*: dati kas ir augstākā līmenī būs arī zemākos līmeņos
- *Koherenti*: dati kas ir dažādos līmeņos ***ir jāsaskaņo***

Atmiņas pamatprincipi

- Definīcijas
  - *Augstāk*: tuvāk procesoram
  - *Bloks*: minimālais datu apjoms kas var būt (nebūt) augstākā līmenī
  - *Bloka adrese*: bloka izvietojums atmiņā
  - *Trāpījuma laiks (Hit time)*: laiks kas vajadzīgs lai piekļūtu augstākā līmeņa datiem ieskaitot noteikšanu vai tie tur ir
  - *Zaudējumi kļūdas gadījumā (Miss penalty)*: piekļuves laiks + pārraides laiks
  - *Piekļuves laiks (Access time)*: laiks kas vajadzīgs lai atmiņa saņemtu un apstrādātu pieprasījumu (nosaka atmiņas latentumu)
  - *Pārraides laiks (Transfer time)*: laiks kas vajadzīgs lai saņemtu visu bloku no atmiņas (nosaka atmiņas caurlaides spēja)

Atmiņas pamatprincipi



- *Zaudējumi kļūdas gadījumā (Miss penalty)* = A + B + C
- *Piekļuves laiks (Access time)* = A + B
- *Pārraides laiks (Transfer time)* = C
- Joslas platuma problēmas var atrisināt ar naudas palīdzību bet latentumu....
- Konveijerizācijā bija līdzīgi: ieviešanas rezultātā ***uzlabojās*** CPU ***caurlaides spēja*** bet ***"sabojājās" latentums***
- Atmiņas gadījumā lai iegūtu joslas platumu var:
    - Veidot platākas kopnes, lielākus datu blokus, izgudrot jaunus atmiņas šūnu izvietojuma organizācijas veidus
- Latentuma ziņā ir grūtāk:
    - Pieprasījumam no keša jānonāk atmiņā (ārpus kodola)
    - Jāizdara atmiņas apsekošana (lookup)
    - Bitiem jādodas atpakaļ pa tiem pašiem vadiem

Kešs

- Keša definīcija:
  - Vēsturiski droša vieta kur noslēpt vai uzglabāt lietas
- Keša definīcija CS kontekstā:
  - Pirmais atmiņas hierarhijas līmenis kas nāk pēc CPU
  - Pamatā šodien gan ar vārdu kešs saprot visus mehānismus un vietas kas veidotas ar domu veikt atkārtoti izmantojamo lietu uzglabāšanu

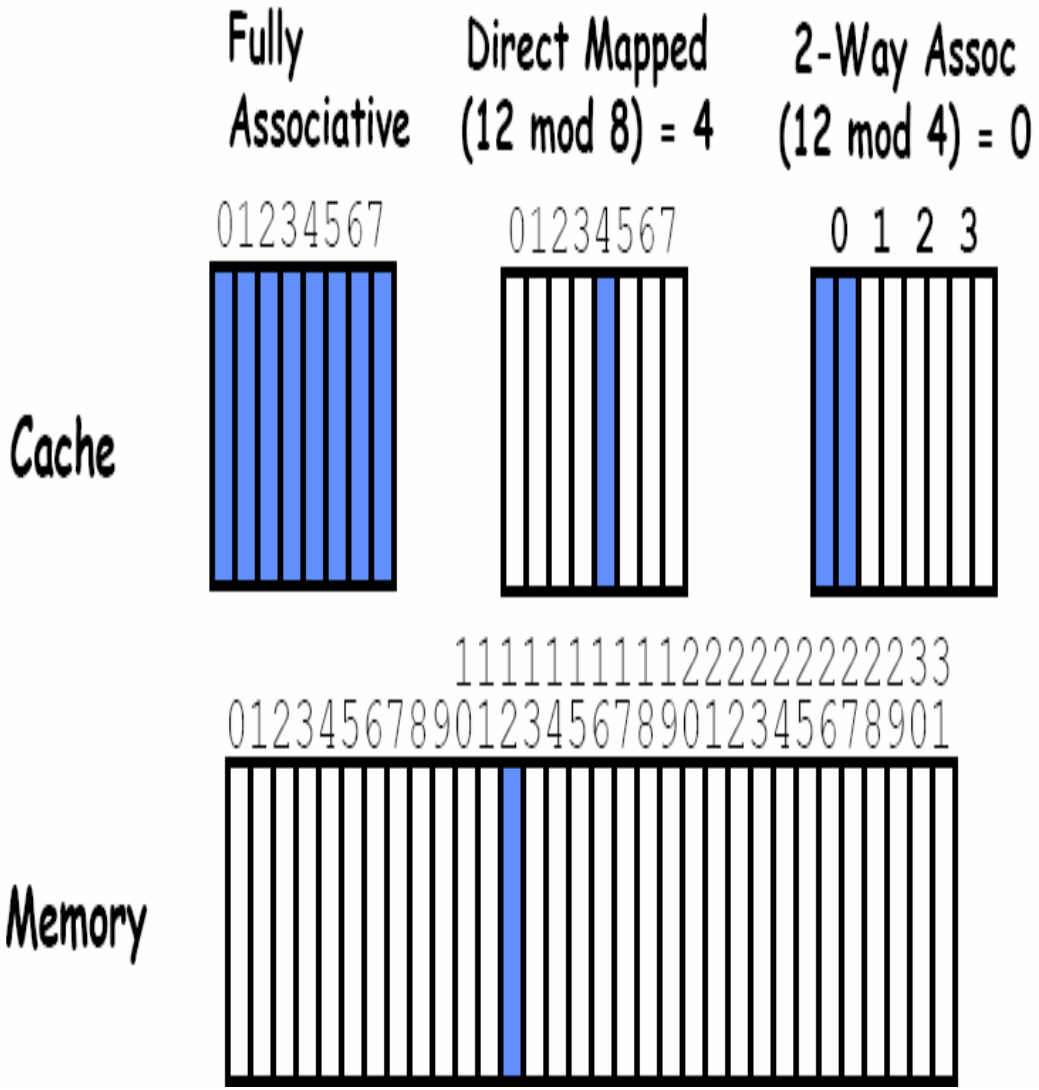
Četri pamata jautājumi

- Q1: Kurā vietā novietot nolasīto bloku augstākā līmenī? *(Block placement)*
- Q2: Kā noteikt vai bloks jau ir augstākā līmenī? *(Block identification)*
- Q3: Kuru bloku vajag aizvietot ja nav trāpījums? *(Block replacement)*
- Q4: Kas notiek ieraksta laikā? *(Write strategy)*

Kurā vietā novietot nolasīto bloku augstākā līmenī?

- Direct Mapped: Katram blokam ir tikai viena vieta kešā kurā tas var atrasties.
- Fully associative: Jebkurš bloks var tik izvietots jebkurā keša vietā.
- Set associative: Katram blokam ir noteikta vietu kopa kurā to var ievietot.
  - Ja kopā ir *n* bloki tad tādu novietojumu sauc par “n-way set associative”
- Kāda ir “direct mapped” keša asociativitāte?

Asociativitātes piemēri



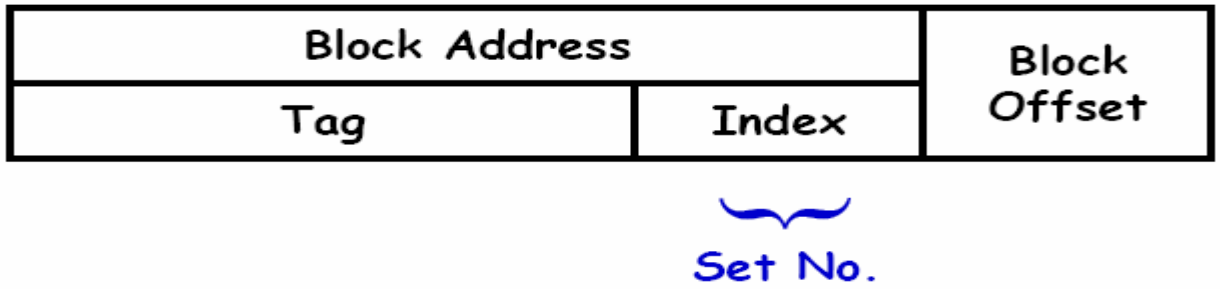
Fully associative:
12 bloks var būt ievietots jebkurā vietā

Direct mapped:
Block no. = (Block address) mod (No. of blocks in cache)
12 bloks var tikt ievietots tikai 4. vietā (12 mod 8)

Set associative:
Set no. = (Block address) mod (No. of sets in cache)
12 bloks var tikt ievietots jebkurā 0 kopas vietā (12 mod 4)

Kā noteikt vai bloks jau ir augstākā līmenī?

- Adresi sadala divās pamatdaļās
  - Bloka nobīde (Block offset): nosaka datu vietu blokā
  - Bloka adrese (Block address): sastāv no taga (tag) un indeksa
    - index: norāda keša kopu
  - Tagi (tag): salīdzinot tos var noteikt vai dati ir vai nav kešā (hit/miss)
    - tag size = address size - index size - offset size
    - Asociativitātes palielināšana samazina indeksu bet palielina tagu



Kuru bloku vajag aizvietot ja nav trāpījums (miss)?

- Keša kļūdas pieprasa bloku aizstāšanu (replacement)
  - direct mapped gadījumā nav jādomā
  - set associative gadījumā ir jādomā un jāpieņem lēmums
- Aizvietošanas stratēģijas:
  - Optimālā – aizvieto to bloku kuru ***neizmantos visilgāk***
  - Least Recently Used (LRU) – izmet to kam ***nav piekļūts visilgāko laiku*** (optimāli lokalitātei laikā)
  - Gadījuma (Random) – gandrīz tikpat labs kā LRU (to arī ir viegli realizēt aparatūrā)
  - FIFO izmet to kas kešā ir ***turēts visilgāk*** (viegli realizēt)
  - Least frequently used (LFU) – izmet to kam ir bijušas ***vismazāk piekļuves***

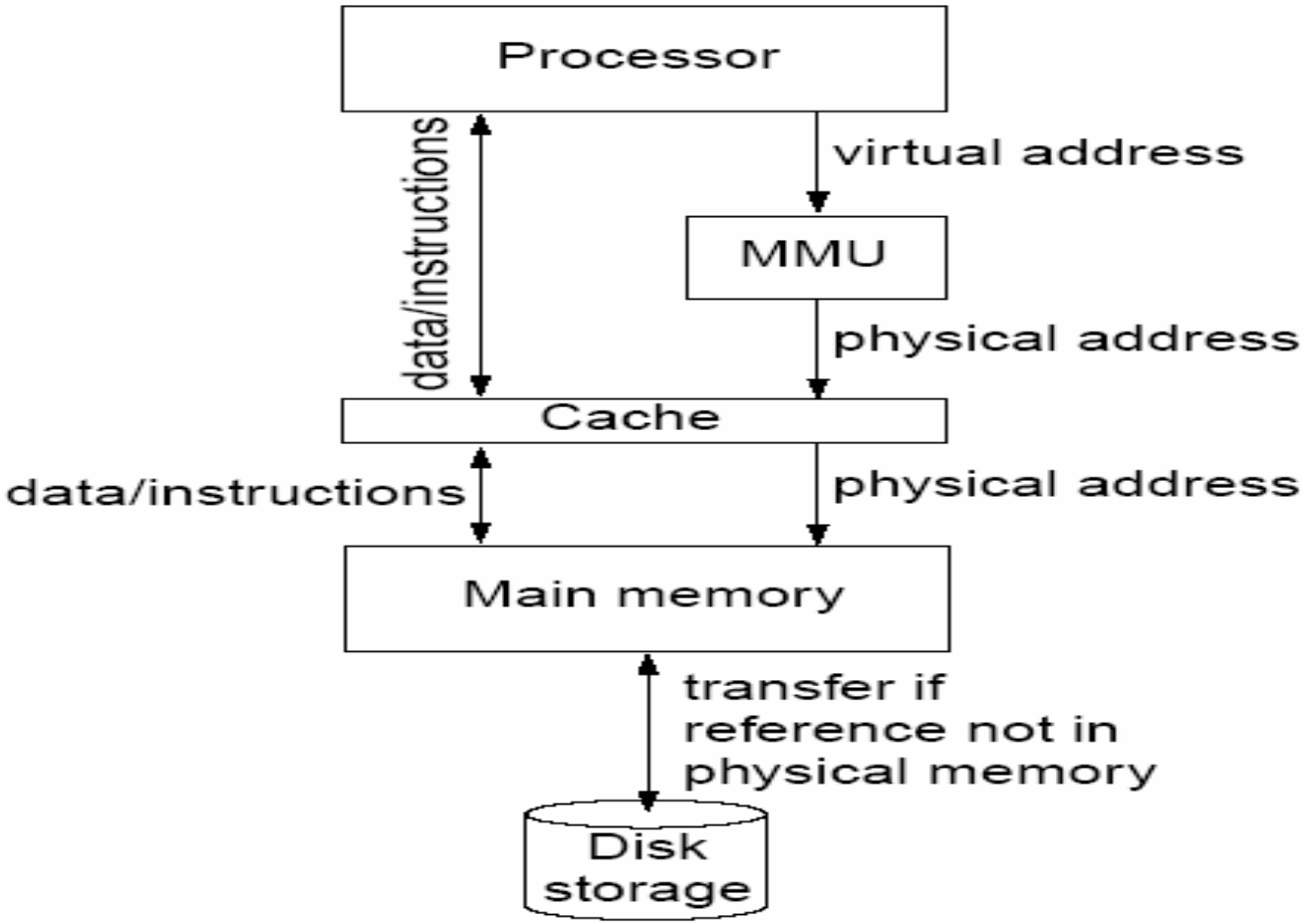
Kas notiek ieraksta laikā?

- Ieraksts cauri (Write through): Informācija tiek ierakstīta abās vietās (keša blokā un zemāka līmeņa blokā).
- Ieraksts atpakaļ (Write back): Informācija tiek ierakstīta tikai keša blokā. Modificētais keša bloks tiek ierakstīts atmiņā tikai tad kad tas tiek aizvietots.
  - Tīrs / netīrs? (jāpievieno ”dirty bit” katram blokam)
- Ieraksts cauri
  - Vieglāk izveidot
  - Pamatatmiņa ir vienmēr konsistenta
  - Jālieto ieraksta buferi jo citādi rodas ieraksta aizkaves (write stalls)
- Ieraksts atpakaļ
  - Mazāka atmiņas plūsma
  - Ieraksti notiek ar keša ātrumu
  - Pamatatmiņa ***ne vienmēr ir konsistenta*** ar kešu saturu
  - Aizvietošanas (Evictions) darbības ir ilgākas jo tad ir jāveic ieraksts atmiņā pirms var aizvietot bloku

Virtuālā atmiņa

- Adrešu lauks kas ir nepieciešams programmas darbam parasti ir daudzkārt lielāks nekā pieejamais pamat atmiņas apjoms.
- Tikai neliela programmas daļa ietilpst pamatatmiņā bet pārējais tiek glabāts sekundārajā atmiņā (diskos)
- Lai programmu varētu izpildīt tai ir jāatrodas pamatatmiņā. Tapēc kādam tās segmentam vispirms ir jātop ielādētam pamatatmiņā (potenciāli aizvietojot kādu citu tur jau esošu segmentu)
- Datu un programmu pārvietošanu no un uz pamatatmiņu notiek automātiski (OS)
- Tas ***kā to dara*** tiek saukts par VM tehnoloģiju
- CPU binārā adrese ir virtuāla (loģiskā) adrese kas ir daudzkārt lielāka nekā RAM apjoms.
- Ja virtuālā adrese attiecas uz to programmas daļu kas jau atrodas RAM (kešā) tad piekļuve tai notiek tieši. Ja tā nav tad tā vispirms ir jāielādē pamatatmiņā.
- Virtuālo adrešu translāciju uz fiziskām veic speciāls mezgls - Memory Management Unit (MMU).

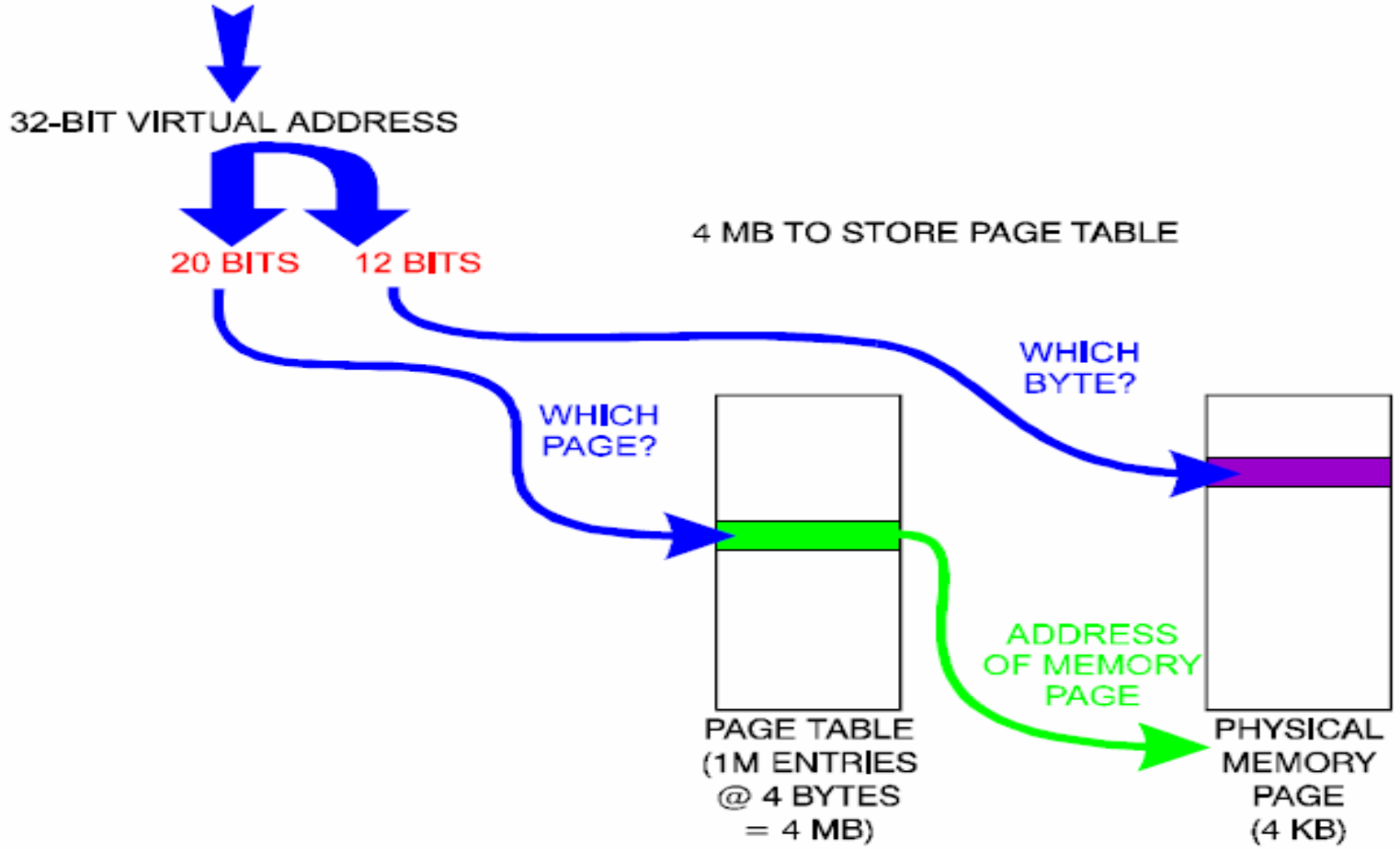
Virtuālā atmiņa



Virtuālā atmiņa un pieprasījumlapošana

- Virtuālā atmiņa nodrošina arī uzdevumu aizsardzību vienam no otra.
- Pamatā šodien virtuālais atmiņas lauks (kods un dati) ir sadalīts vienāda izmēra lapaspusēs (2 -16 KB) “pages”
- Fiziskā atmiņa ir sadalīta kadros “frames” kas pēc izmēra atbilst lapaspusei
- Kādas vēl organizācijas var būt +/- ?
- LPP ir pamata informācijas elements kas ar VM sistēmas palīdzību var tikt pārvietots starp pamatatmiņu un disku
- OS izlemj kuras dotās lietotnes lpp ievietot RAM un kuras aizvietot tā lai minimizētu lpp kļūdas “page faults”
- Lpp kļūda ir situācija kurā CPU atsaucas uz adresi kas atrodas lapā kas savukārt neatrodas RAM.

Adrešu translācija



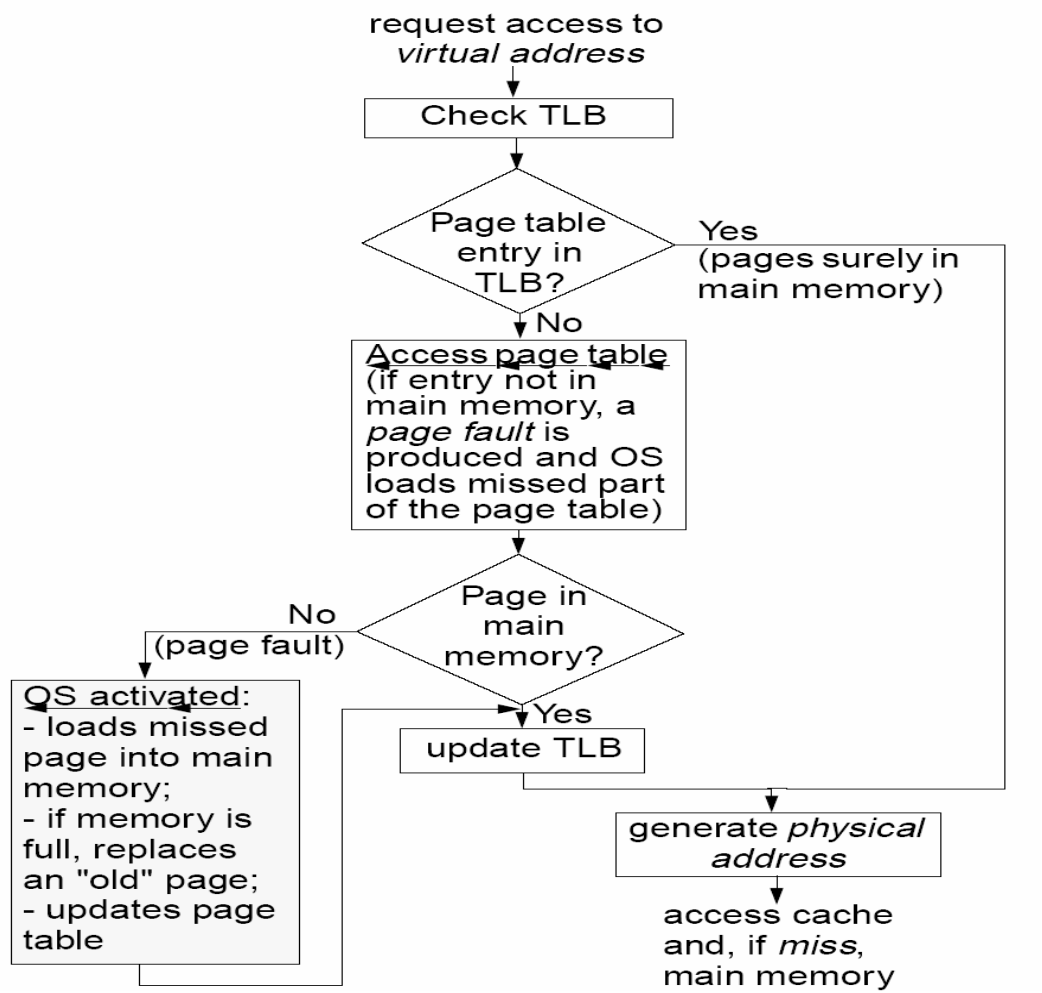
Adrešu translācija LPP tabula

- Katra piekļuve atmiņā izvietotam vārdam izsauc virtuālās adreses translāciju fiziskajās adresēs:
  - Virtuālā adrese = lpp. Nr. + nobīde
  - Fiziskā adrese = kadra nr. + nobīde
- Translāciju veic MMU ar lpp tabulas palīdzību
- Lpp tabula satur ***vienu ierakstu katrai*** virtuālās atmiņas lpp
- Katrs Lpp tabulas ieraksts satur atmiņas kadra adresi kurā atrodas vajadzīgā lpp (ja tā ir ievietot pamatatmiņā)
- Katrs Lpp tabulas ieraksts satur arī papildus informāciju:
  - Ir/nav ielādēta RAM
  - Ir/nav izmainīta
  - Piekļuves tiesību informācija
- Diemžēl lpp tabula ir ***ļoti liela*** un piekļuvei tai ir jānotiek ***ļoti ātri***.

LPP tabula/TLB

- Lai atrisinātu šo problēmu lpp tabulas ierakstiem lieto speciālu kešu “translation lookaside buffer (TLB)”
- Tā darbība ir līdzīga atmiņas kešu darbībai un tā satur tikai nesen lietotos ierakstus
- Lpp tabula ir pat pārāk liela lai to glabātu pamatatmiņā tāpēc to ar VM palīdzību sadala pa atmiņas hierarhiju:
  - TLB kešu
  - Pamatatmiņu
  - Disku

Virtuālā atmiņa



Kopumā

- CPU – atmiņa veiktspējas atšķirība ir pamata ierobežojošais faktors kas neļauj kāpināt kopējo datora veiktspēju
- Atmiņas hierarhija:
  - Izmanto lokalitātes principus
  - Tuvāk CPU => mazāk, ātrāk, dārgāk
  - Tālāk no CPU => lielāks, lēnāks, lētāks
- 4 pamata jautājumi
- Programmas kas nepielieto lokalitāti neiegūst uzlabojumu no atmiņas hierarhijas
- Keši ir aparatūra bet VM vairāk tomēr ir programmatūras pārziņā.
- Kešu parametri
  - Kopējais izmērs, bloka izmērs, asociativitāte
  - Adresācija ar virtuālo vai fizisko adresi?
- Nākotnē - Intelligent RAM (“IRAM”)

Mājās

- http://en.wikipedia.org/wiki/Computer_memory