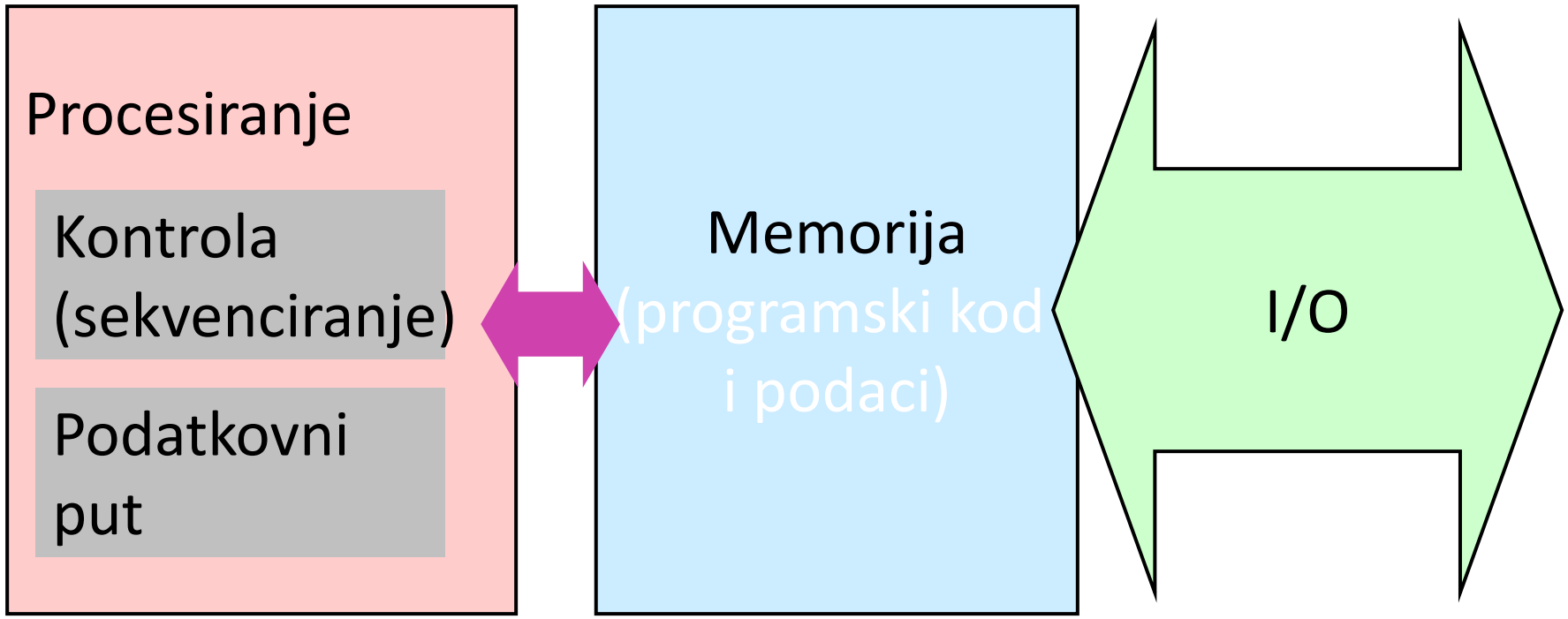
# Građa računala

## Memorijska hijerarhija

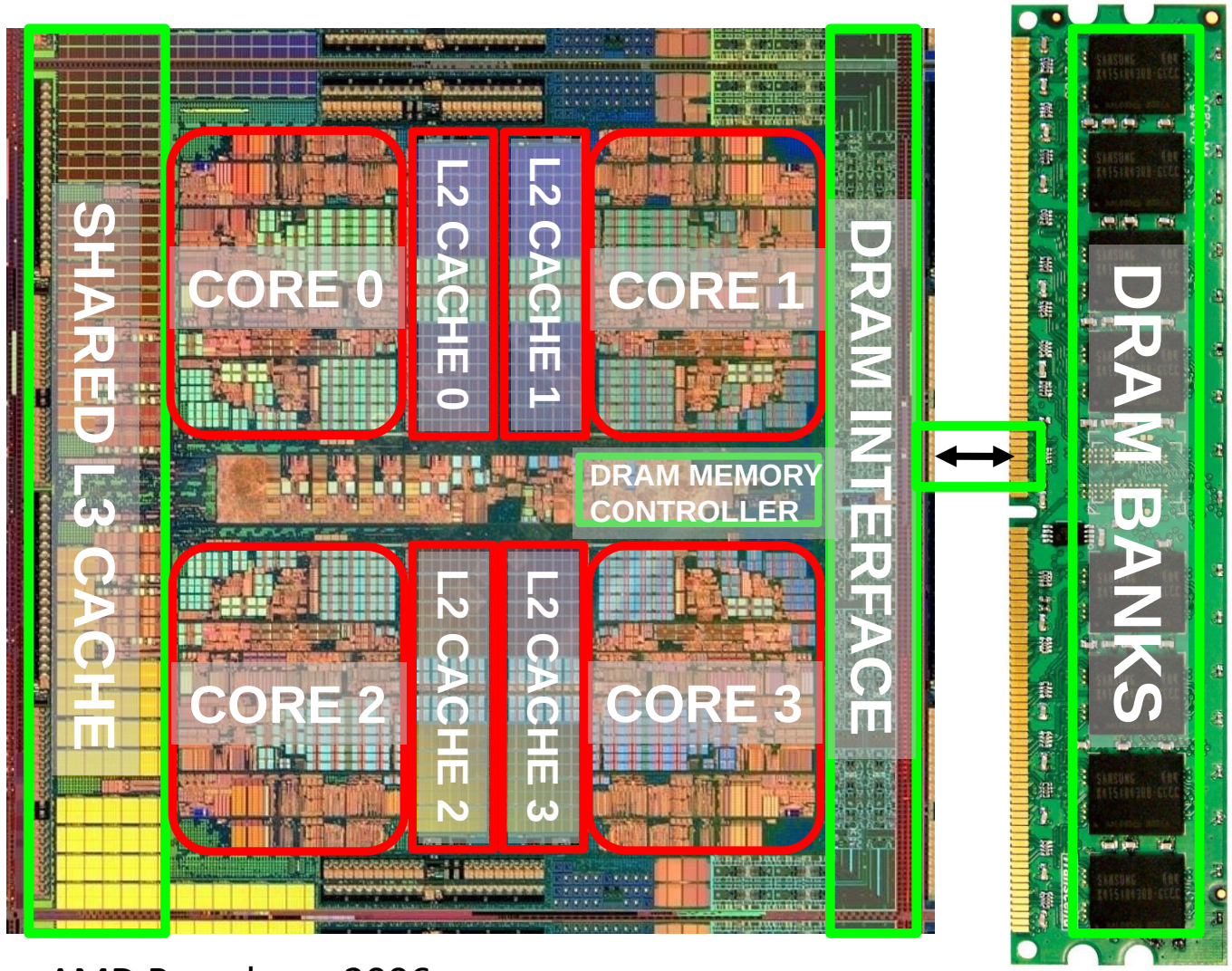
# Ponavljanje: Što je računalo?

- Pokrili smo dvije od slijedeće tri komponente.



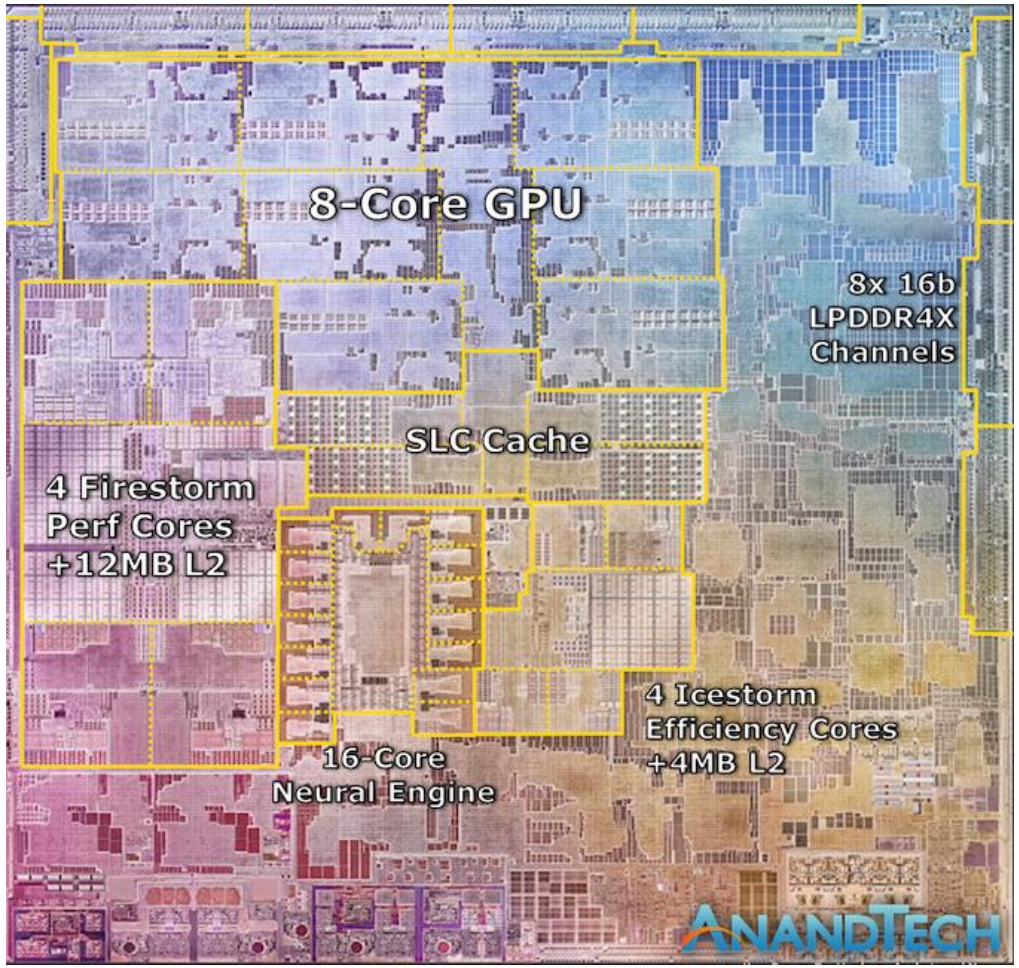
# Memorija je od ključne važnosti

# Memorija u modernom sustavu



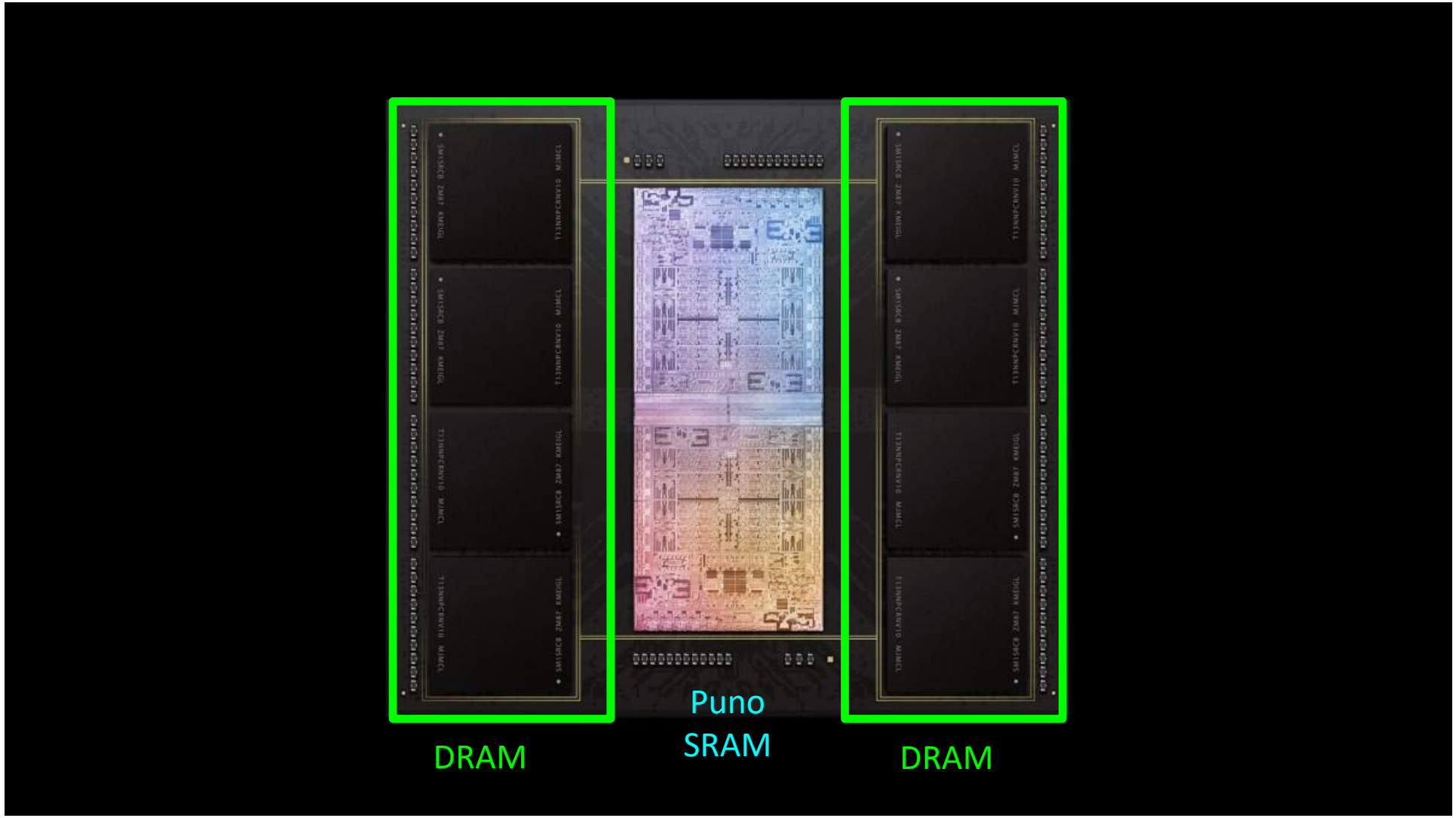

AMD Barcelona, 2006

# Veliki dio modernih čipova je memorija

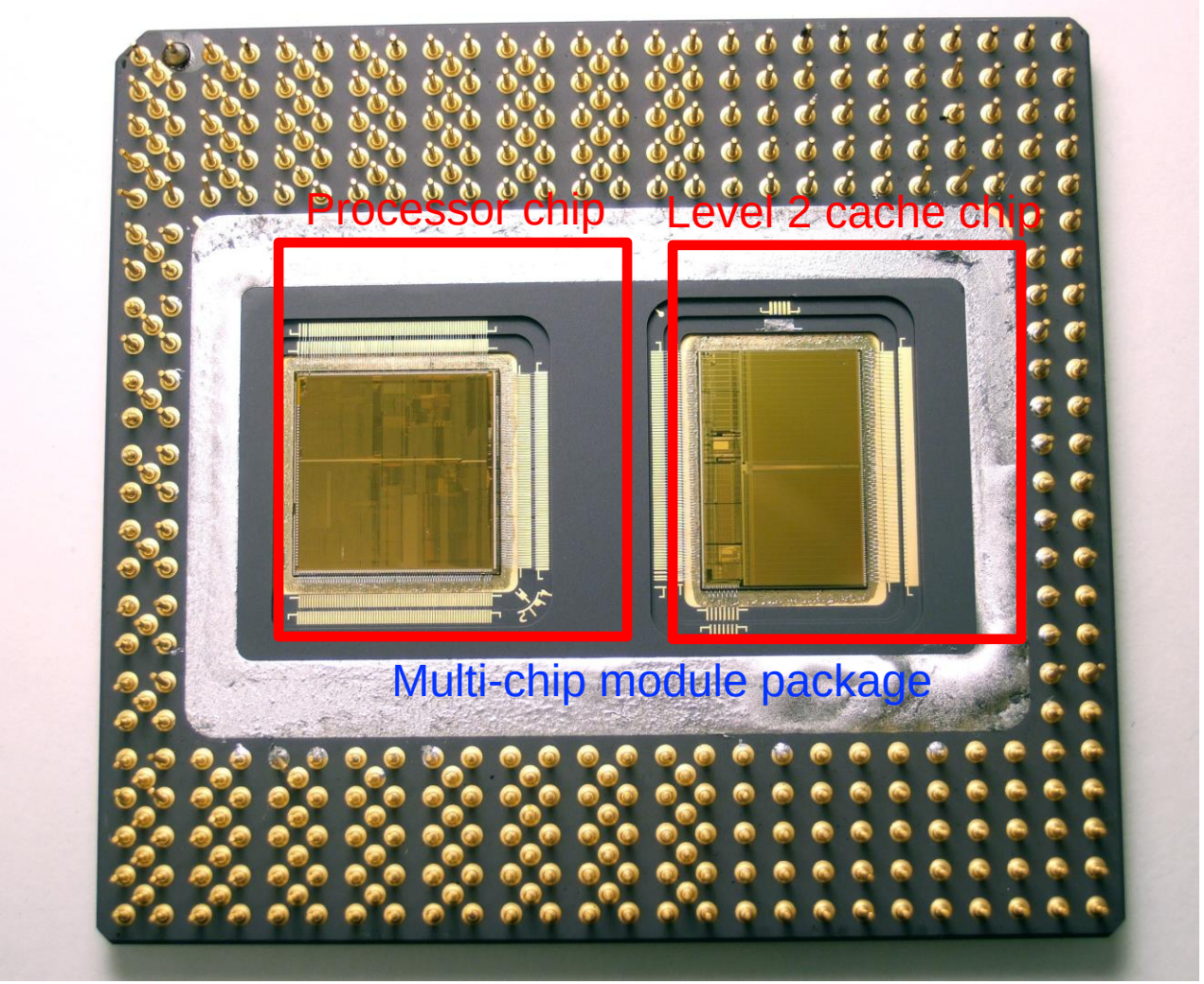



Apple M1, 2021

# Veliki dio modernih sustava je memorija

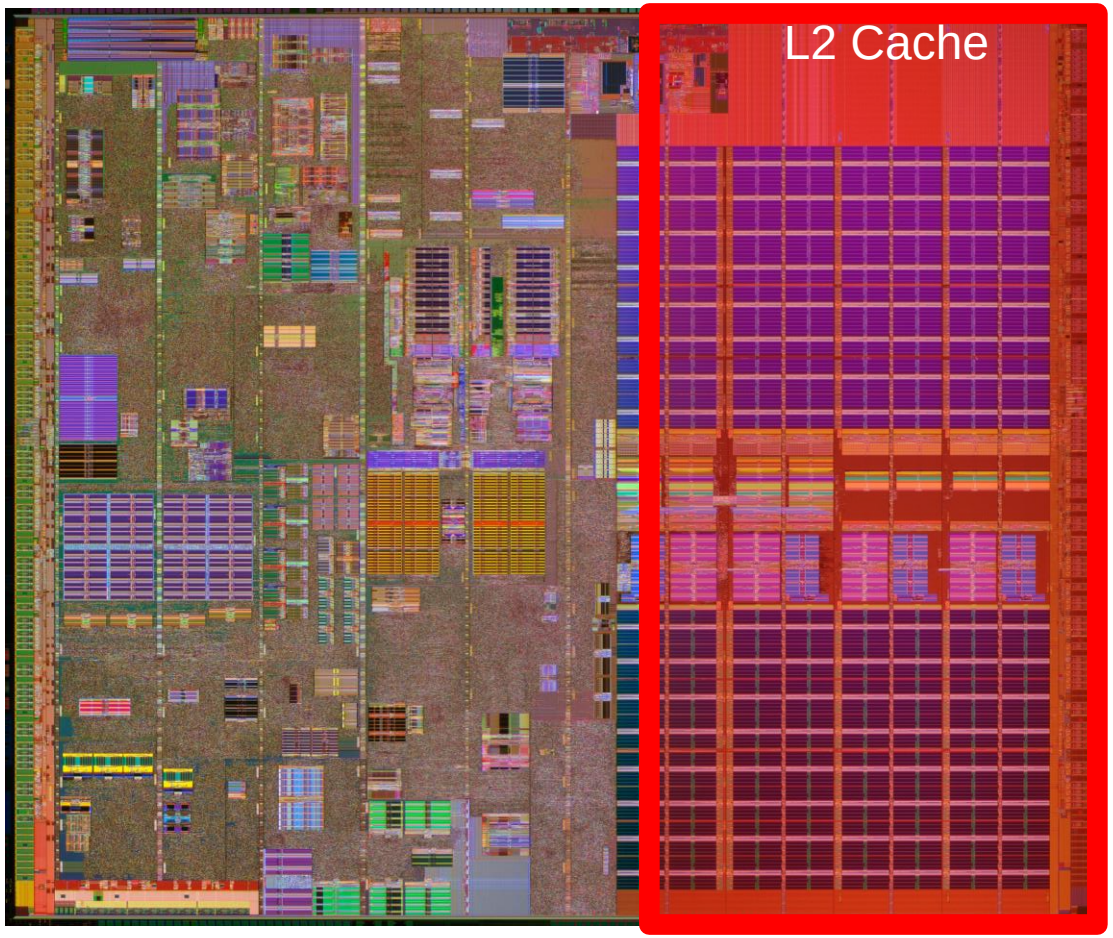


# Veliki dio modernih sustava je memorija



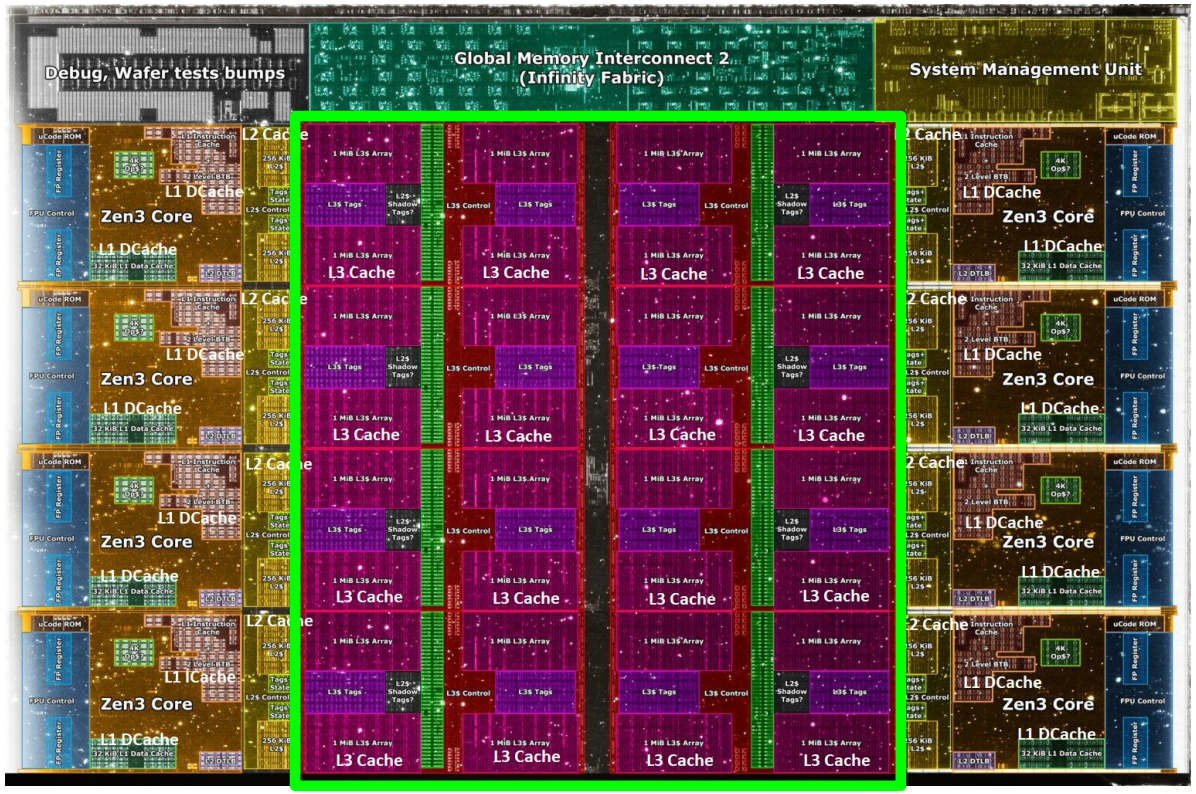

Intel Pentium Pro, 1995

# Veliki dio modernih sustava je memorija




Intel Pentium 4, 2000

# Veliki dio modernih sustava je memorija




AMD Ryzen 5000, 2020

## Broj jezgri:

8 jezgri (*cores*)/16 dretvi (*threads*)

## L1 Caches (predmemorija):

32 KB per core

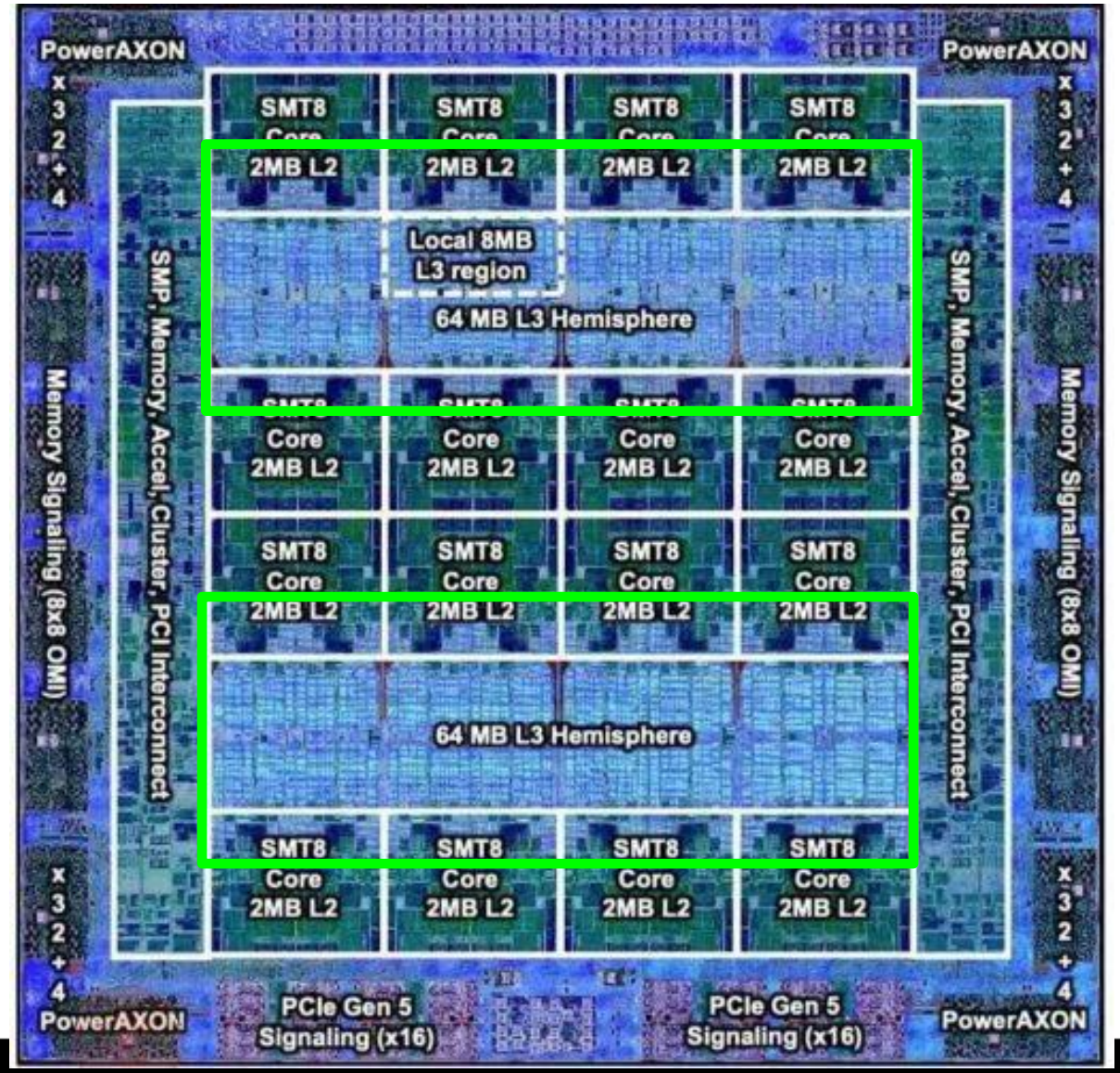
## L2 Caches:

512 KB per core

## L3 Cache:

32 MB shared

# Veliki dio modernih sustava je memorija




IBM POWER10, 2020

## Cores:

15-16 cores,
8 threads/core

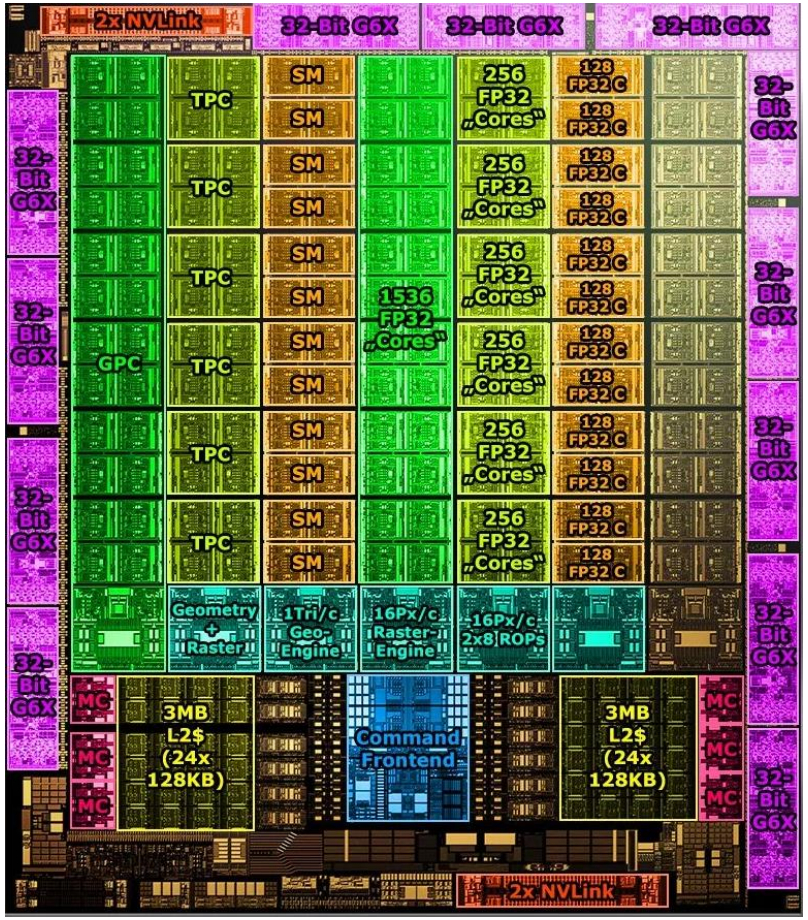
## L2 Caches:

2 MB per core

## L3 Cache:

120 MB shared

# Veliki dio modernih sustava je memorija




Nvidia Ampere, 2020

## Cores:

128 Streaming Multiprocessors

## L1 Cache or Scratchpad:

192KB per SM

Može se koristiti kao L1 Cache i/ili Scratchpad

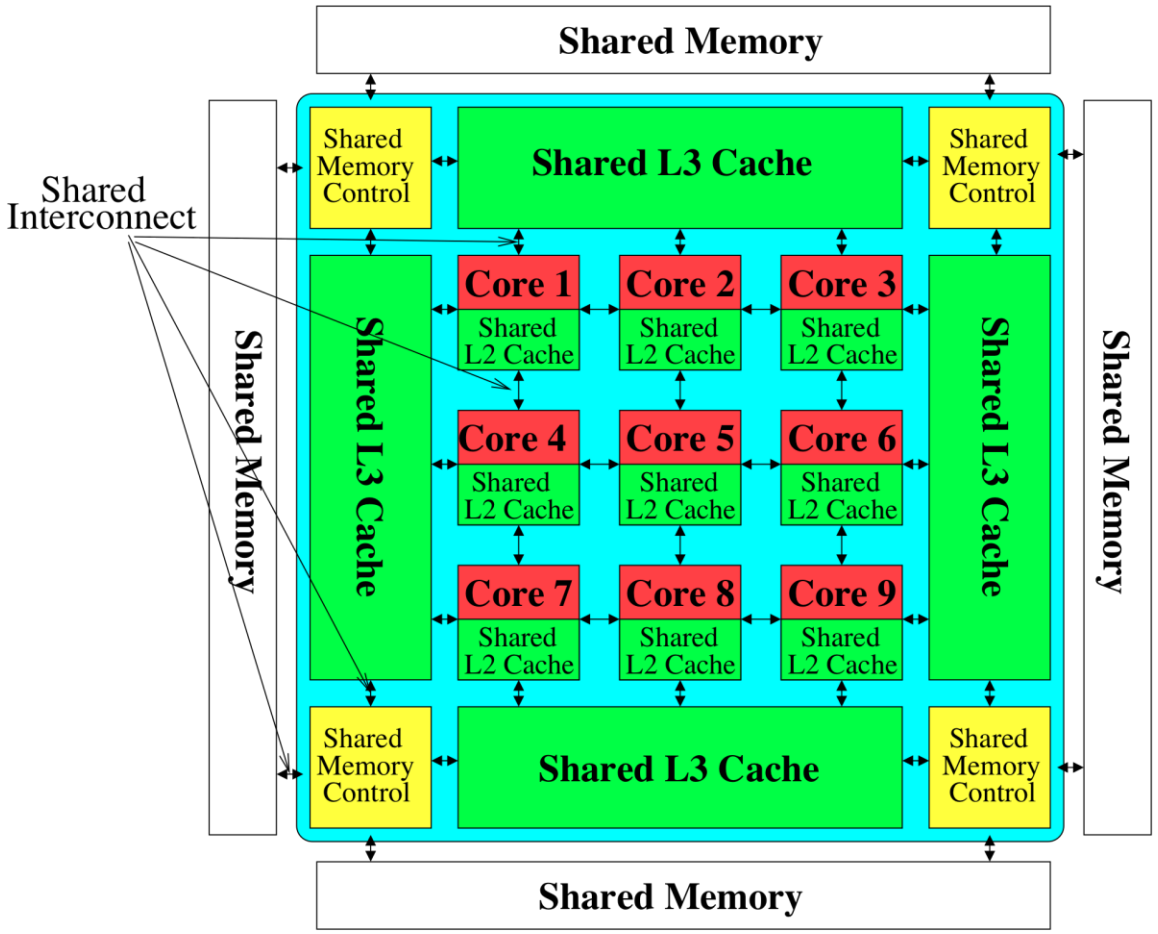
## L2 Cache:

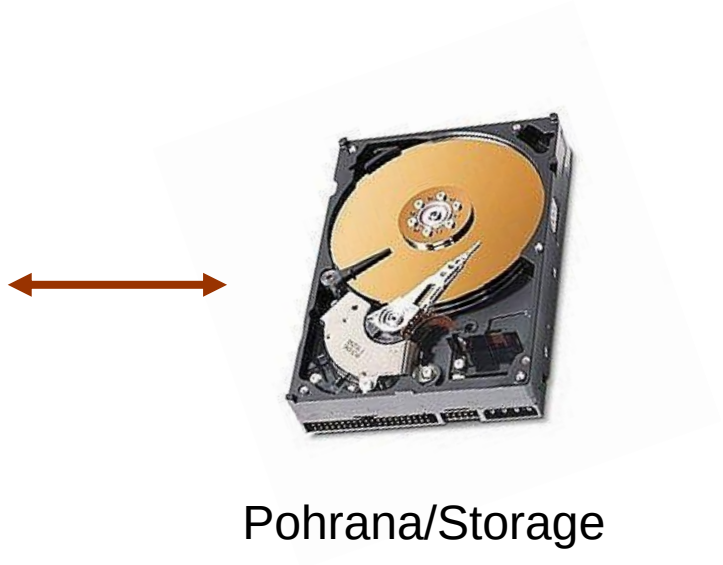
40 MB shared



https://www.tomshardware.com/news/infrared-photographer-photos-nvidia-ga102-ampere-silicon

# Memorijski sustav: Veći dio platforme



Pohrana/Storage

**Većini dio sustava je posvećen je pohranjivanju i premještanju podataka**

**Ipak, sustav je još uvijek usporen („*usko grlo*") u memoriji**

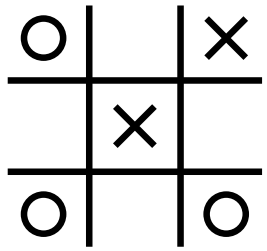
# Memorija je kritična za performanse

Računarstvo je limitirano od strane podataka

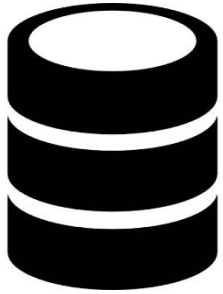
# Usko grlo - memorija

- Računamo sa puno+puno više „faktora” (podataka)
  - ML/AI, Genomics, Data Analytics, Databases, Graph Analytics, …
- Obrada podataka zahtjeva brzo i efikasno procesiranje velike količine podataka
- Podaci se povećavaju
  - Možemo generirati mnogo podataka više nego što možemo obraditi

# Perspektiva aplikacije

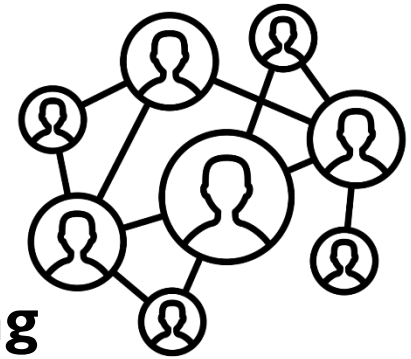


# Memorija je ključna za performanse



**In-memory Databases**

[Mao+, EuroSys'12;
Clapp+ (**Intel**), IISWC'15]

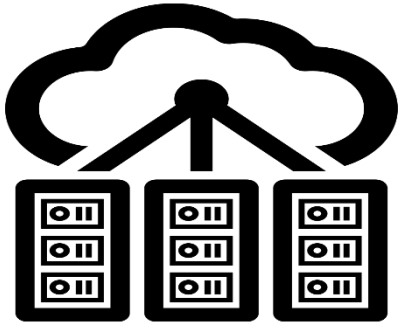


**Graph/Tree Processing**

[Xu+, IISWC'12; Umuroglu+, FPL'15]

**In-Memory Data Analytics**

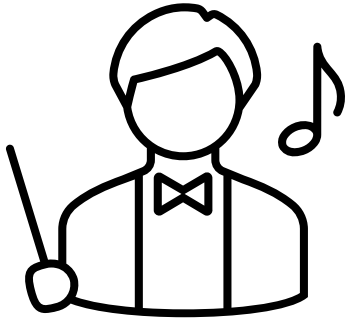
[Clapp+ (**Intel**), IISWC'15;
Awan+, BDCloud'15]



**Datacenter Workloads**

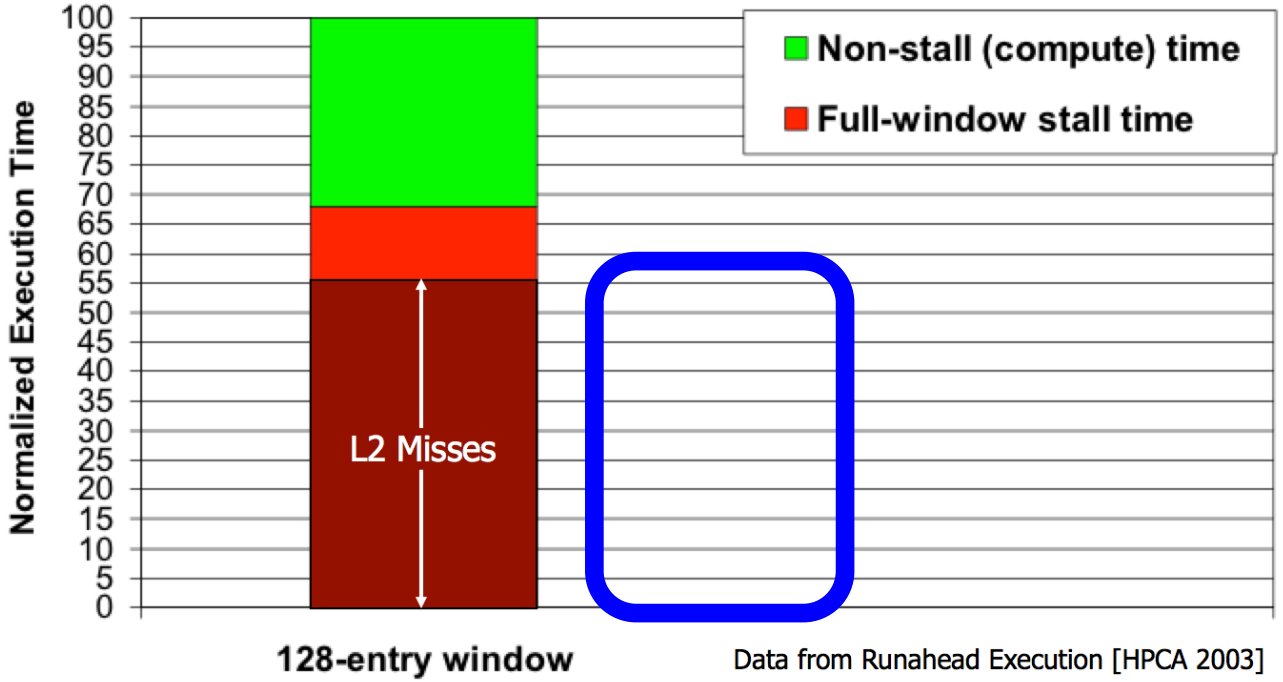
[Kanev+ (**Google**), ISCA'15]

# Perspektiva izvedbe



# Usko grlo memorije

- "**It's the Memory, Stupid!**" (Richard Sites, MPR, 1996)
  - „Problem je u memoriji (ne u CPU-u), *pametnjakoviću!*"

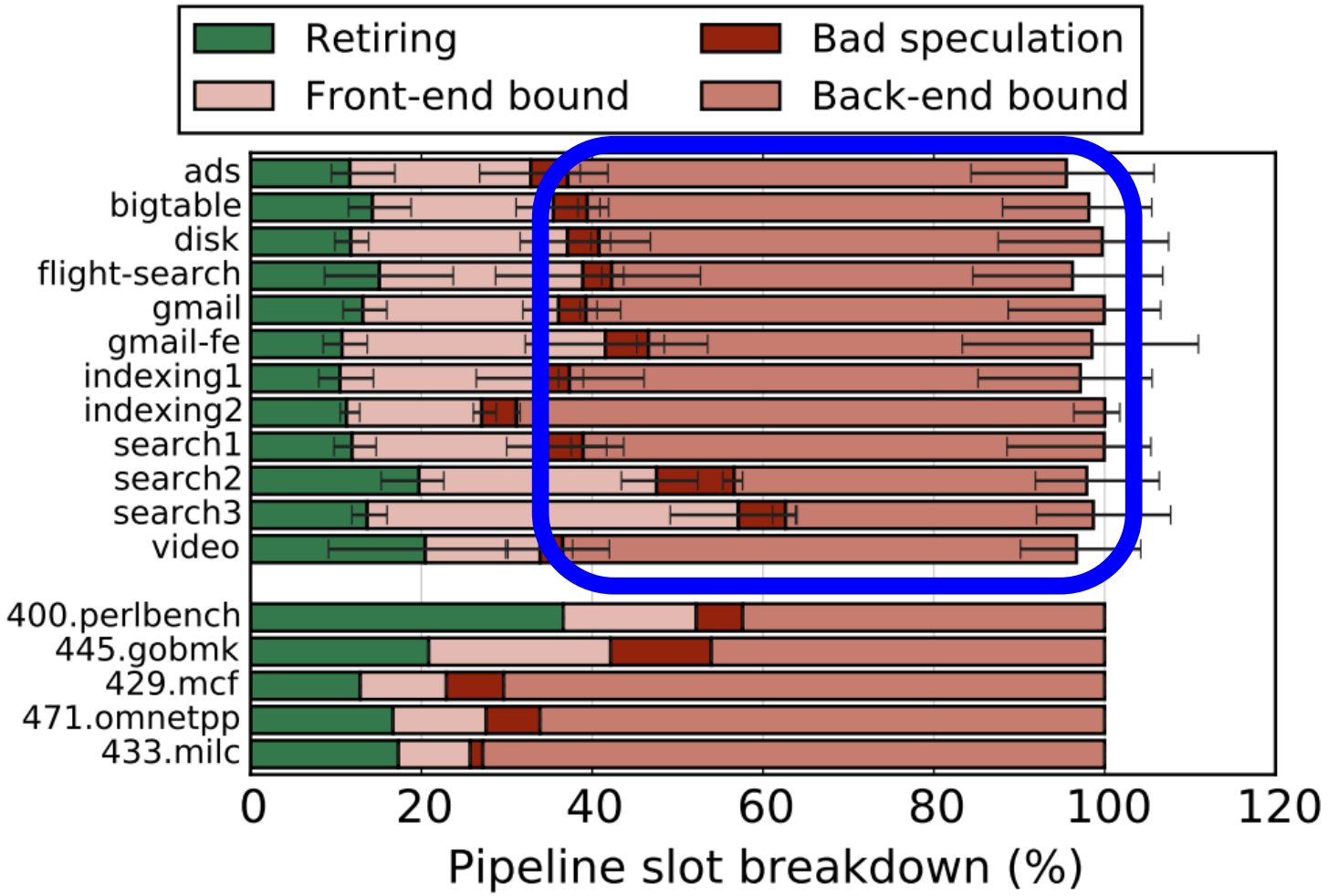



Data from Runahead Execution [HPCA 2003]

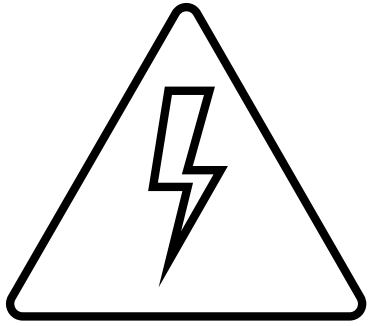
I expect that over the coming decade memory subsystem design will be the *only* important design issue for microprocessors.

# Usko grlo memorije

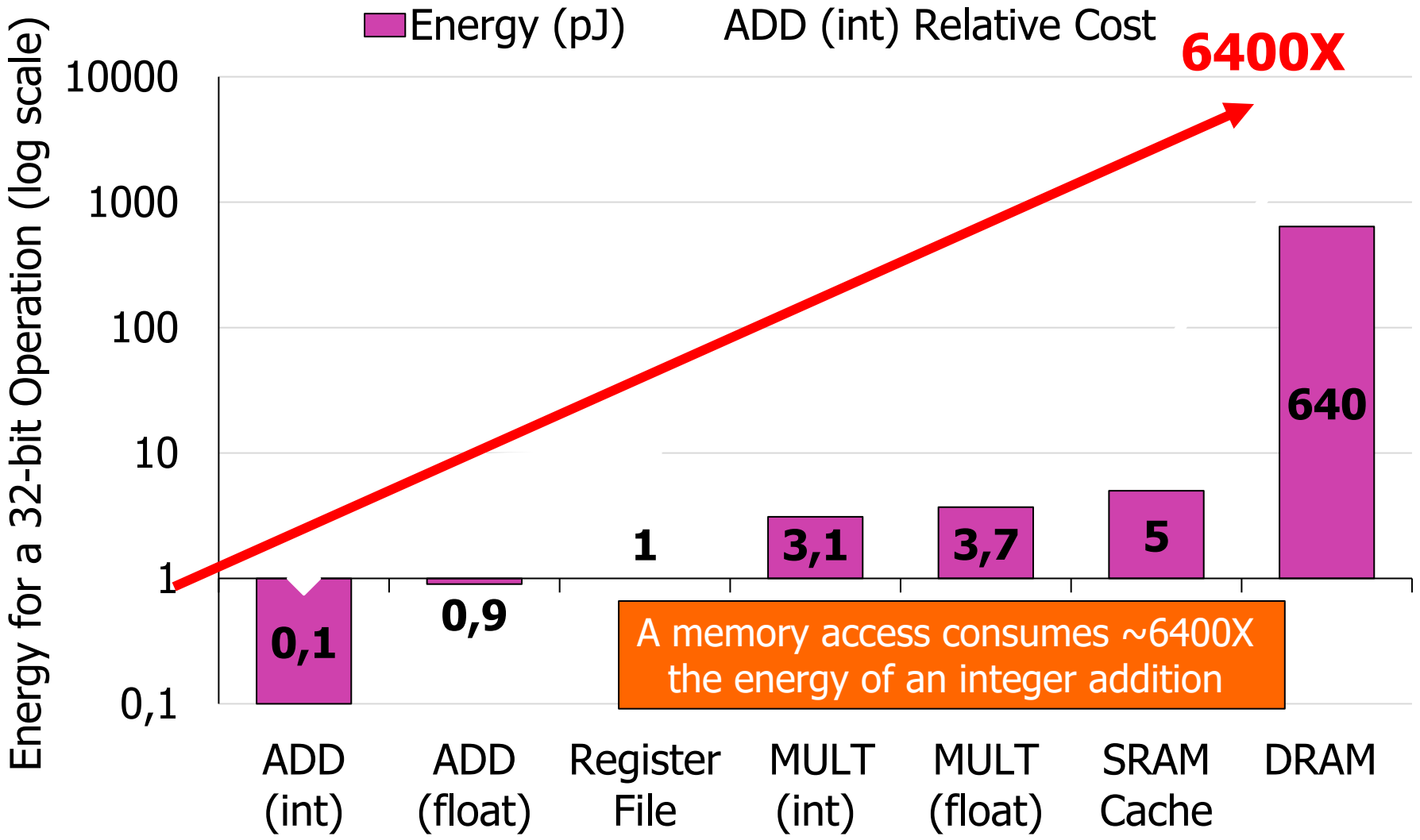
- Sva radna opterećenja *Googleovog* podatkovnog centra (2015):



# Energetska perspektiva



# Kretanje podataka u odnosu na računalnu energiju

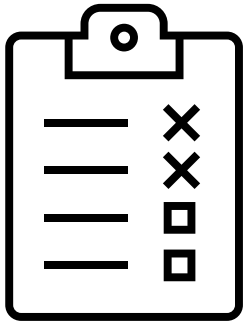


# Kretanje podataka u odnosu na računalnu energiju

| 32-bit Operacija | Energija (pJ) | DODATI (int) Relativni trošak |
|---|---|---|
| ADD (int) | 0.1 | 1 |
| ADD (float) | 0.9 | 9 |
| Register File | 1 | 10 |
| MULT (int) | 3.1 | 31 |
| MULT (float) | 3.7 | 37 |
| SRAM Cache | 5 | 50 |
| DRAM | 640 | 6400 |

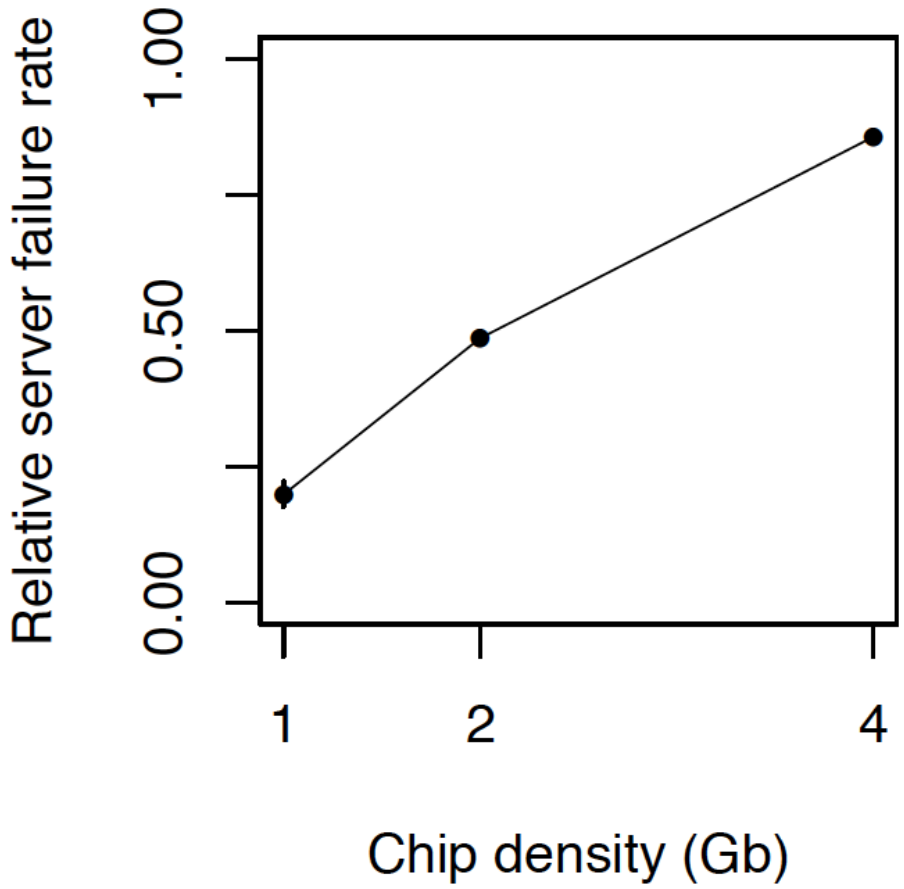
Pristup memoriji troši ~6400X
Energije cjelobrojnog dodatka

# Perspektiva pouzdanosti i sigurnosti



# Memorija je ključna za pouzdanost

- Podaci sa svih Facebookovih poslužitelja širom svijeta (2015)



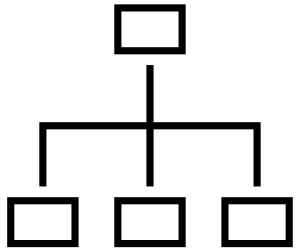

Kako se povećava memorija, tako pada pouzdanost sustava

# Memorija je ključna za računanje

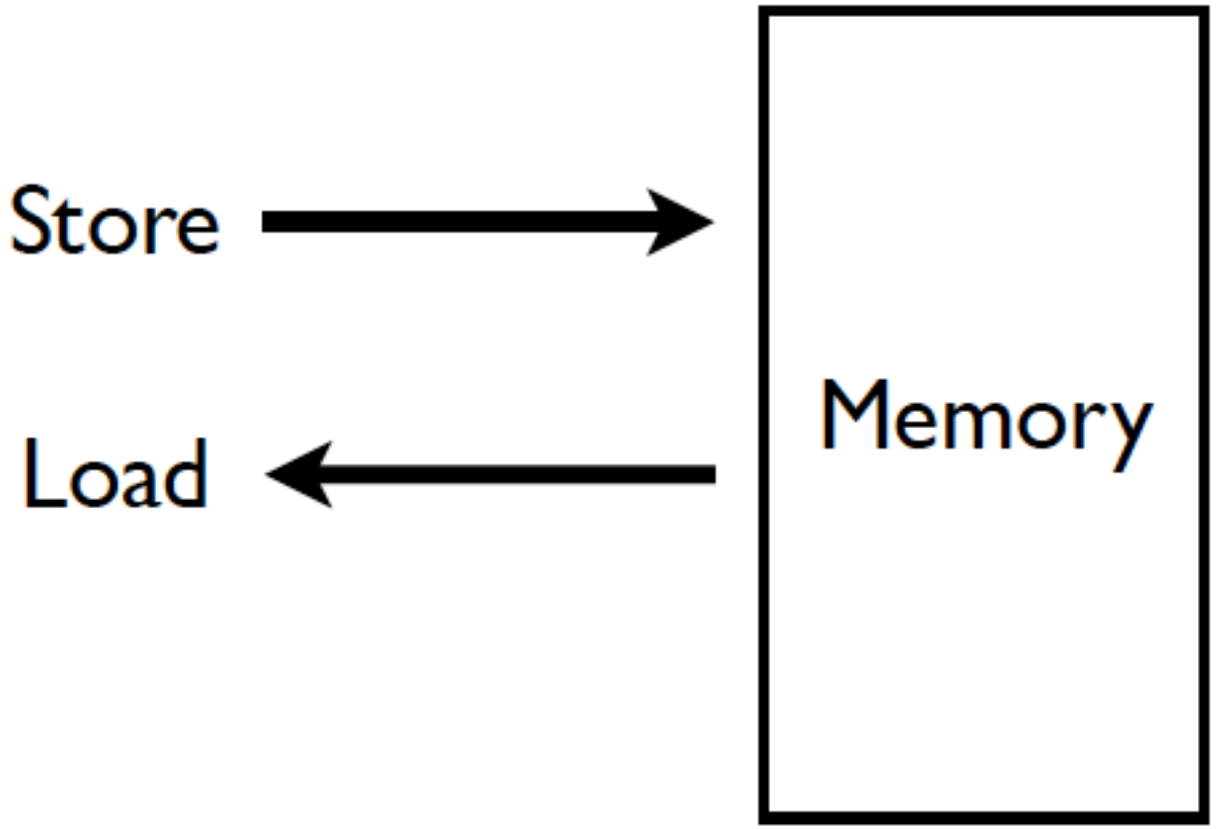
# Memorija je ključna za računanje

- Izvedbe/Performanse
- Energije
- Pouzdanosti
- Sigurnosti
- Troškova
- Faktora obrasca
- Kvalitete usluge & Predvidljivosti
- …

# Organizacija i tehnologija memorije



# Memorija (*Što vidi programer*)



# Apstrakcija: Virtualna vs. Fizička memorija

- Programer vidi virtualnu memoriju
  - Pretpostavlja da je beskonačna
- Stvarnost: Fizička memorija je mnogo manja od one za koju programer pretpostavlja
- Sustav (system software + hardware, OS) mapira adrese virtualne memorije u fizičke
  - Sustav automatski upravlja fizičkim memorijskim prostorom transparentno prema programeru

\+ Programer ne mora znati fizičku veličinu memorije niti njome upravljati → Mala fizička memorija može se pojaviti kao ogromna programeru → Život je olakšan programeru

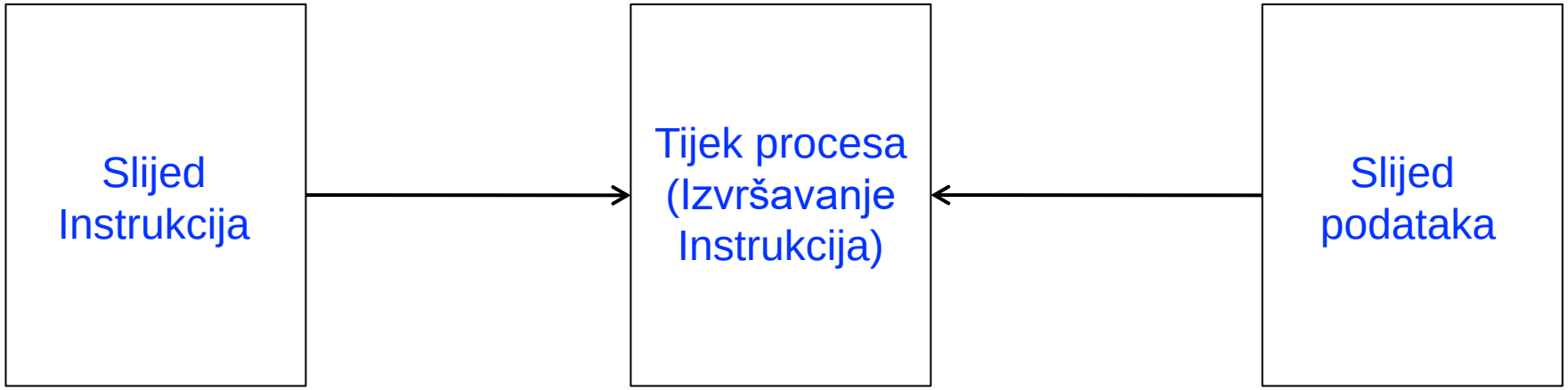
\-- Složeniji sistemski softver i arhitektura

Klasičan primjer kompromisa programera i (mikro)arhitekta

# (Fizički) Memorijski sustav

- Potrebna vam je veća razina prostora za pohranu da biste automatski upravljali malom količinom fizičke memorije
  → Fizička memorija ima spremište sigurnosnih kopija: disk
- Prvo ćemo početi s fizičkim memorijskim sustavom
- Za sada, ignorirajte virtualno → fizičku indirektnost

# Idealni sustav




- Pristup s nultom latencijom
- Beskonačan kapacitet
- Nulta cijena
- Savršen kontrolni tok

- Nema zastoja na putu
- Savršen protok podataka (nema zavisnosti reg/memorije)
- Super međusobna povezanost (operand komunikacija)
- Dovoljno funkcionalnih jedinica
- Računanje je **jako brzo** i bez kašnjenja

- Nema kašnjenja u pristupu
- Beskonačni kapacitet
- Beskonačna brzina/dolaznost
- Besplatno

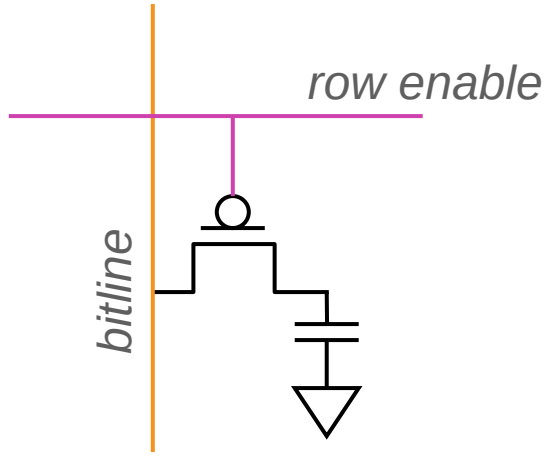
# Kako možemo pohraniti podatke?

- Flip-Flops (ili Latches)
  - Vrlo brz, paralelan pristup
  - Vrlo skupo (jedan bit košta desetke tranzistora)
- Statički RAM (*opisat ćemo ih kasnije*)
  - Relativno brzo, samo jedna po jedna podatkovna riječ (stranica)
  - Skupo (jedan bit košta 6+ tranzistora)
- Dinamički RAM (*opisat ćemo ih kasnije*)
  - Sporije, jedna po jedna podatkovna riječ, čitanje uništava sadržaj (refresh), potrebno periodičko „osvježenje” električnog naboja kondenzatora (gubitak podataka)
  - Jeftin (jedan bit košta samo jedan tranzistor plus jedan kondenzator)
- Ostale tehnologije pohrane (flash memorija, tvrdi disk, traka)
  - Mnogo sporije, pristup traje dugo vremena, non-volatile (nepostojana)
  - Vrlo jeftino (jedan tranzistor pohranjuje mnogo bitova ili bez tranzistora koji su uključeni)

# Tehnologija memorije: DRAM i SRAM

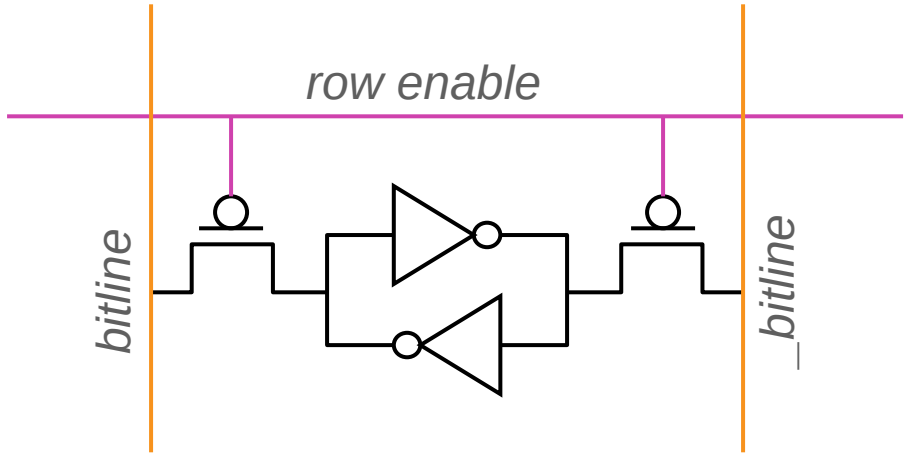
# Tehnologija memorije: DRAM

- Dynamic random access memory
- Kondenzatorski naboj označava pohranjenu vrijednost
  - Hoće li kondenzator biti napunjen ili ispražnjen, označava skladištenje od 1 ili 0
  - 1 kondenzator
  - 1 pristupni tranzistor
- Kondenzator „curi” kroz RC put
  - DRAM ćelija s vremenom gubi naboj
  - DRAM ćeliju treba osvježiti



# Tehnologija memorije: SRAM

- Static random access memory
- Dva križna pretvarača pohranjuju jedan bit informacije
  - Povratni put omogućuje zadržavanje pohranjene vrijednosti u "ćeliji"
  - 4 tranzistora za skladištenje
  - 2 tranzistora za pristup



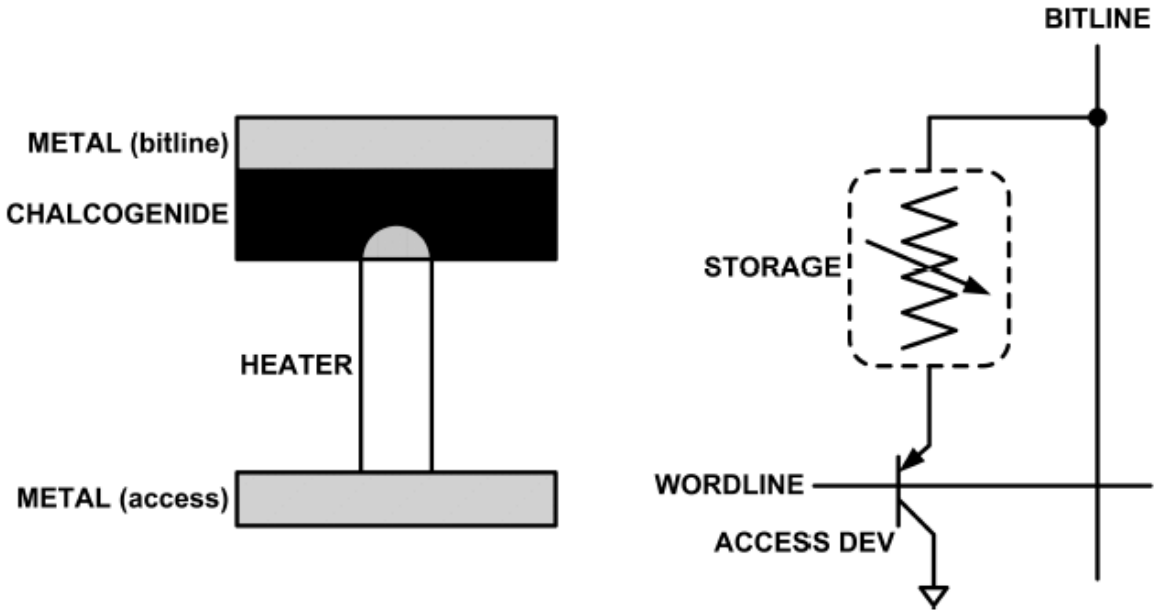
# DRAM vs. SRAM

- DRAM
  - Sporiji pristup (kondenzator)
  - Veća gustoća (1T 1C cell)
  - Manja cijena
  - Zahtijeva osvježavanje (struja, performanse, sklopovi)
  - Proizvodnja zahtijeva spajanje kondenzatora i logike zajedno

- SRAM
  - Brži pristup (nema kondenzatora)
  - Niža gustoća (6T cell)
  - Veća cijena
  - Nema potrebe za osvježavanjem
  - Proizvodnja kompatibilna s logičkim postupkom (bez kondenzatora)

# Za razmišljanje: Promjena faze/strane u memoriji

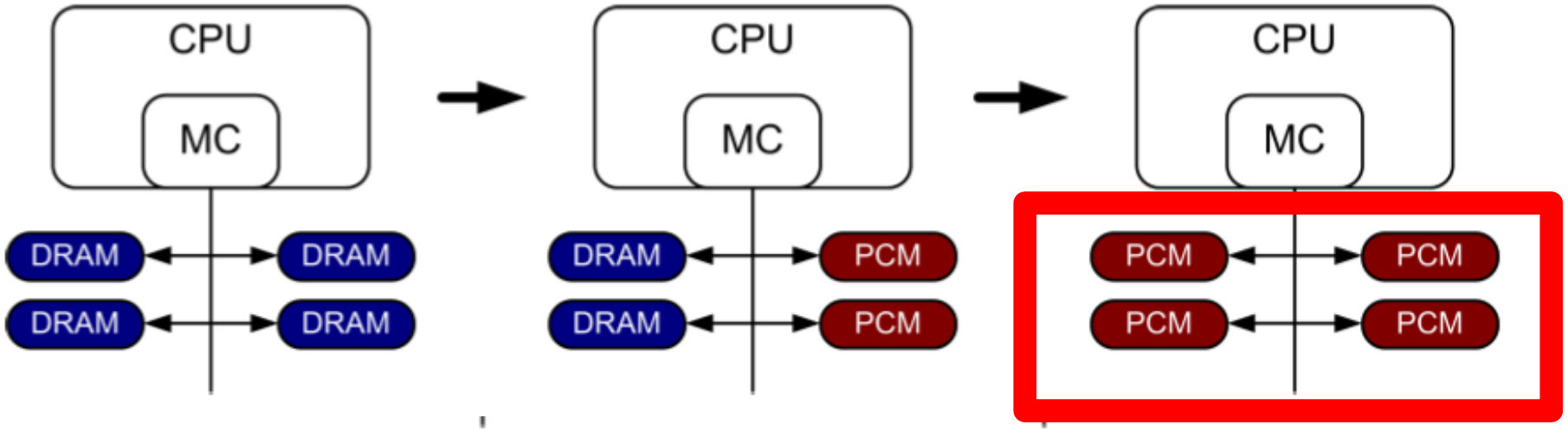
- Materijal za promjenu faza (čaša kalkogenida) postoji u dva stanja:
  - Amorfni: Niska optička refleksivnost i visoka električna otpornost
  - Kristalni: Visoka optička refleksivnost i niska električna otpornost




PCM je otporna memorija: Visoka otpornost (0), Nizak otpor (1)

# PCM bazirana glavna memorija

- Kako bi se trebala organizirati PCM (glavna) memorija?




- Čista PCM glavna memorija [Lee et al., ISCA'09, Top Picks'10]
  - Kako redizajnirati sustav koji bi tolerirao nedostatke PCM-a
- Hibridna PCM+DRAM [Qureshi+ ISCA'09, Dhiman+ DAC'09]
  - Kako particionirati/migrirati podatke između PCM-a i DRAM-a

# Intel Optane Persistent Memory (2019)

- Non-volatile glavna memorija
  - memorija koja zadržava vrijednost podatka kada nema struje
- Bazirana na 3D-XPoint tehnologiji




https://www.storagereview.com/intel_optane_dc_persistent_memory_module_pmm

# DRAM vs. PCM

- DRAM
  - Brži pristup (kondenzator)
  - Niža gustoća (kondenzatori su manje skalabilani) → veća cijena
  - Zahtijeva osvježavanje (struja, performanse, sklopovi)
  - Proizvodnja zahtijeva spajanje kondenzatora i logike
  - Nepostojan (gubitkom struje – gube se podaci)
  - Nema problema s izdržljivošću
  - Manje zahtjeva za energijom
- PCM
  - Sporiji pristup (“promjena faze” temeljena na grijanju i hlađenju)
  - Veća gustoća (materijal za promjenu faza skalabilniji) →niža cijena
  - Nema potrebe za osvježavanjem
  - Proizvodnja zahtijeva manje konvencionalne procese
  - Non-volatile/Nehlapljivo (**ne gubi** podatke pri gubitku energije)
  - Problemi s izdržljivošću (ćelija se ne može koristiti nakon što N piše u nju)
  - Veći zahtjevi za energijom

# Naboj u odnosu na Otporno pamćenje

- Memorija punjenja (e.g., DRAM, Flash)
  - Pisanje podataka hvatanjem naboja Q
  - Čitanje podataka otkrivanjem napona V
- Memorija sa otporom (e.g., PCM, STT-MRAM, memristors)
  - Pisanje podataka pulsirajućom strujom dQ/dt
  - Čitanje podataka otkrivanjem otpora R

# Obećavajuće tehnologije memorije sa otporom

- PCM
  - Ubrizgaj struju za promjenu faze materijala
  - Otpor određen fazom
  -
- STT-MRAM
  - Ubrizgaj struju za promjenu polariteta magneta
  - Otpor određen polaritetom
- Memristors/RRAM/ReRAM
  - Ubrizgaj struju za promjenu atomske strukture
  - Otpor određen udaljenosti atoma

# Zanimljivost o Flash Memoriji & SSD

- Flash memorija je bila vrlo "sumnjiva" tehnologija u nastajanju
  - najmanje dva desetljeća
- Koristi se u SSD diskovima
  - SSD nema diskove – ali se tako prevodi na .hr „Solid State Drive”

# SSD kontroler flash memorije

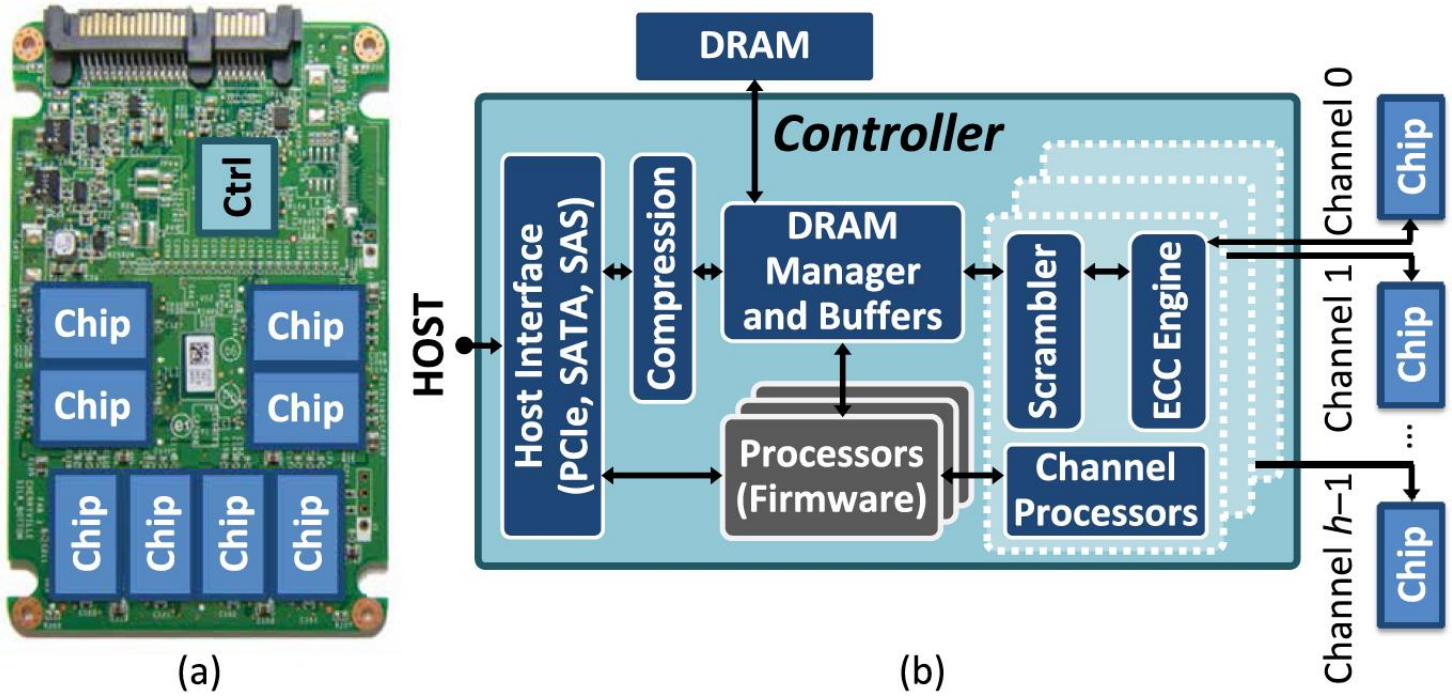
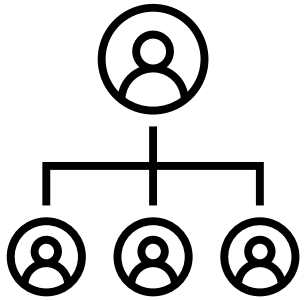



**Fig. 1.** ***(a) SSD system architecture, showing controller (Ctrl) and chips. (b) Detailed view of connections between controller components and chips.***

# Hijerarhija memorije



# Idealna memorija

- Nulto vrijeme pristupa (latencija)
- Beskonačan kapacitet
- Besplatna
- Beskonačna propusnost/brzina (za paralelnu podršku višestrukim pristupima)

# Problem

- Zahtjevi idealne memorije suprotstavljaju se jedni drugima
- Veće je sporije
  - Veća → Potrebno je više vremena za određivanje lokacije
- Brže je skuplje
  - Tehnologija memorije: SRAM vs. DRAM vs. SSD vs. Disk vs. Traka
- Viša propusnost/brzina je skuplja
  - Trebate više banaka, više sučelja, više kanala, višu frekvenciju ili bržu tehnologiju

# Problem

- Veće je sporije
- SRAM, < 1KByte, sub-nanosec
  - SRAM, KByte~MByte, ~nanosec
  - DRAM, Gigabyte, ~50 nanosec
  - PCM-DIMM (Intel Optane DC DIMM), Gigabyte, ~300 nanosec
  - PCM-SSD (Intel Optane SSD), Gigabyte ~Terabyte, ~6-10 µs
  - Flash memory, Gigabyte~Terabyte, ~50-100 µs
  - Hard Disk, Terabyte, ~10 millisec
- Brže je skuplje (područje novčanih troškova i čipova)
  - SRAM, < 0.3$ per Megabyte
  - DRAM,  < 0.03$ per Megabyte
  - PCM-DIMM (Intel Optane DC DIMM), < 0.004$ per Megabyte
  - PCM-SSD, < 0.001$ per Megabyte
  - Flash memory, < 0.00008$ per Megabyte
  - Hard Disk, < 0.00003$ per Megabyte
  - Ove ogledne vrijednosti (circa ~2022) skalirajte sa vremenom
- I druge tehnologije imaju svoje mjesto
  - MRAM, RRAM, STT-MRAM, memristors, … (još nisu sazrile/odrasle)

# Problem (tablica)

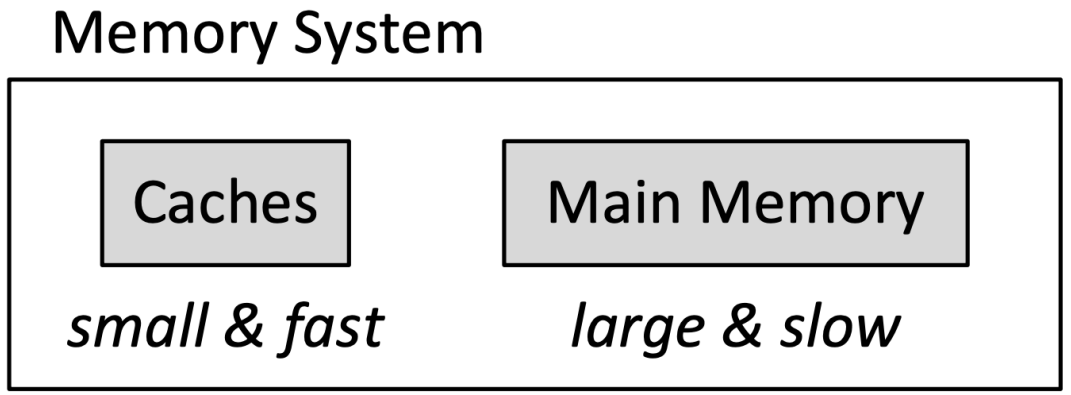
| Memorijski uređaj | Kapacitet | Kašnjenje | Cijena po megabajtu (MB) |
|---|---|---|---|
| SRAM | < 1 KByte | sub-nanosec | |
| SRAM | KByte~MByte | ~nanosec | < 0.3$ |
| DRAM | Gigabyte | ~50 nanosec | < 0.03$ |
| PCM-DIMM<br>(Intel Optane DC DIMM) | Gigabyte | ~300 nanosec | < 0.004$ |
| PCM-SSD<br>(Intel Optane SSD) | Gigabyte<br>~Terabyte | ~6-10 µs | < 0.001$ |
| Flash memory | Gigabyte<br>~Terabyte | ~50-100 µs | < 0.00008$ |
| Hard Disk | Terabyte | ~10 millisec | < 0.00003$ |

Veće je sporije

Brže je skuplje
($$$ i područje čipa)

# Zašto hijerarhija memorije?

- Želimo **i brzo i veliko**
- Ali, ne možemo postići oboje s jednom razinom pamćenja
- Ideja: Imati više razina prostora za pohranu (progresivno veće i sporije jer su razine dalje od procesora) i osigurati da se većina podataka koji su potrebni procesoru čuva na brzim/bližim razinama

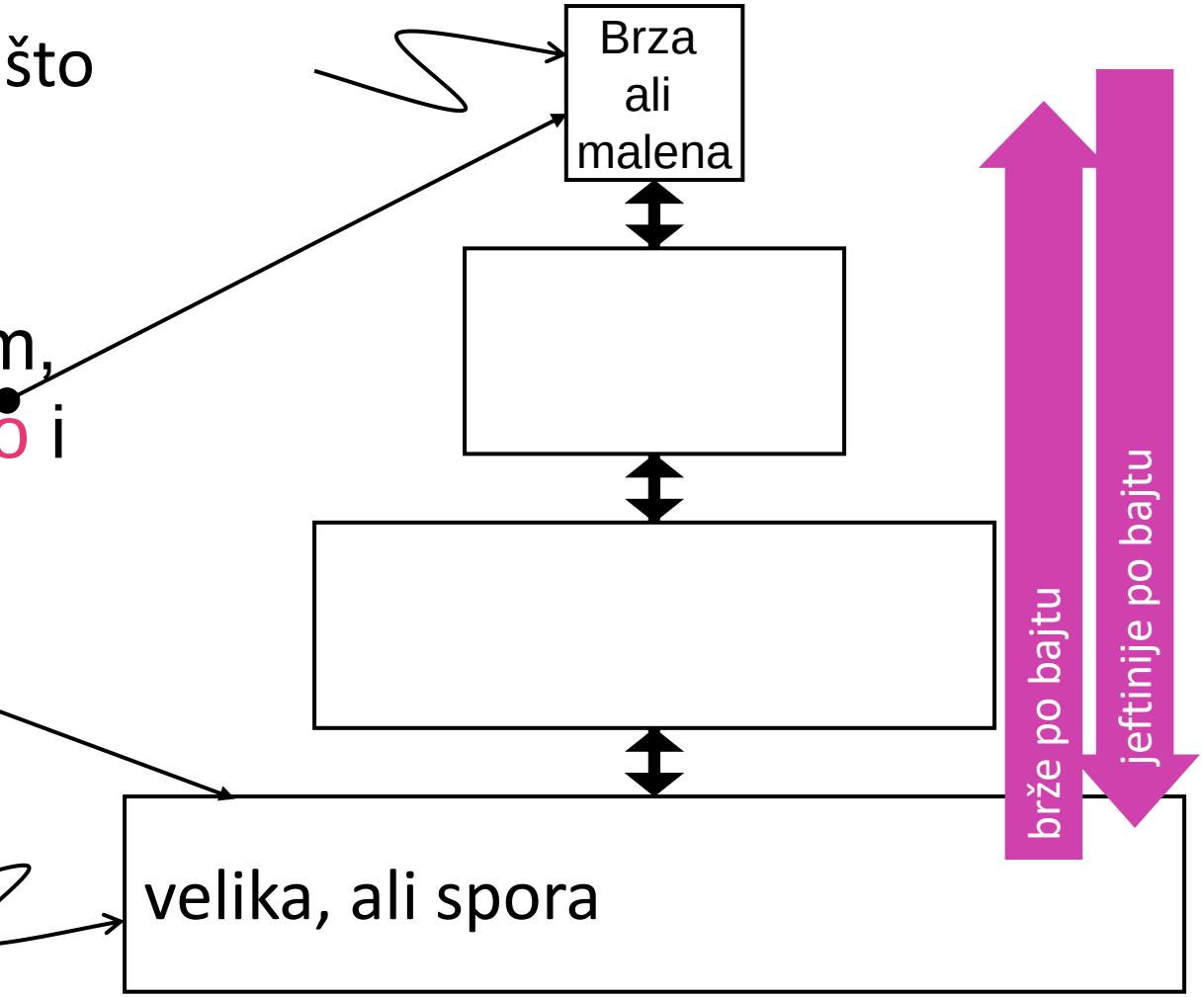


# Hijerarhija memorije

premjestite ono što ovdje koristite

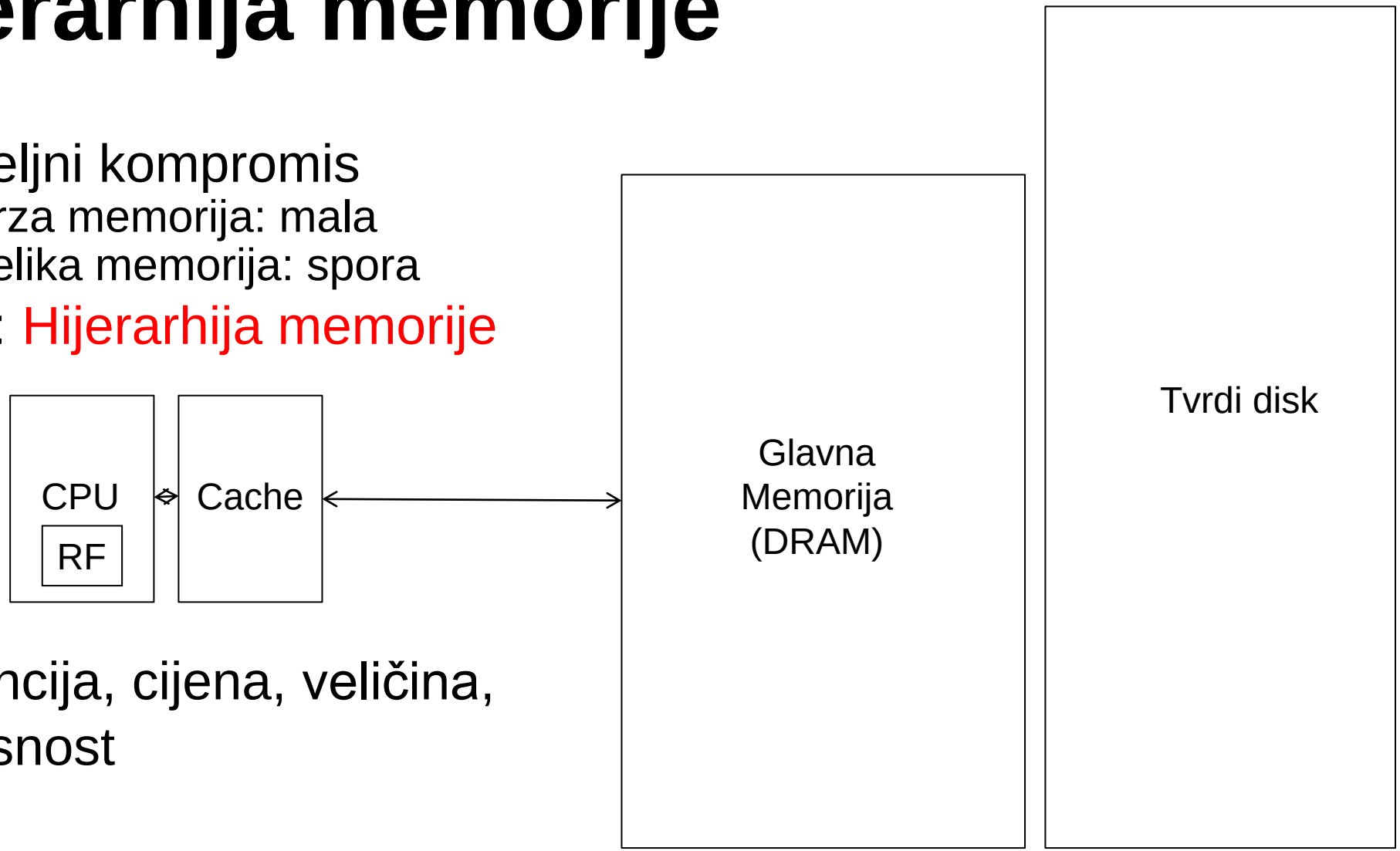
Sa dobrim smještajem, memorija izgleda brzo i veliko

Pohratite ostalo ovdje



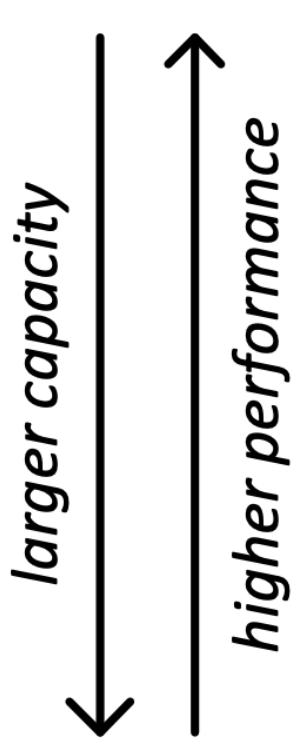
# Hijerarhija memorije

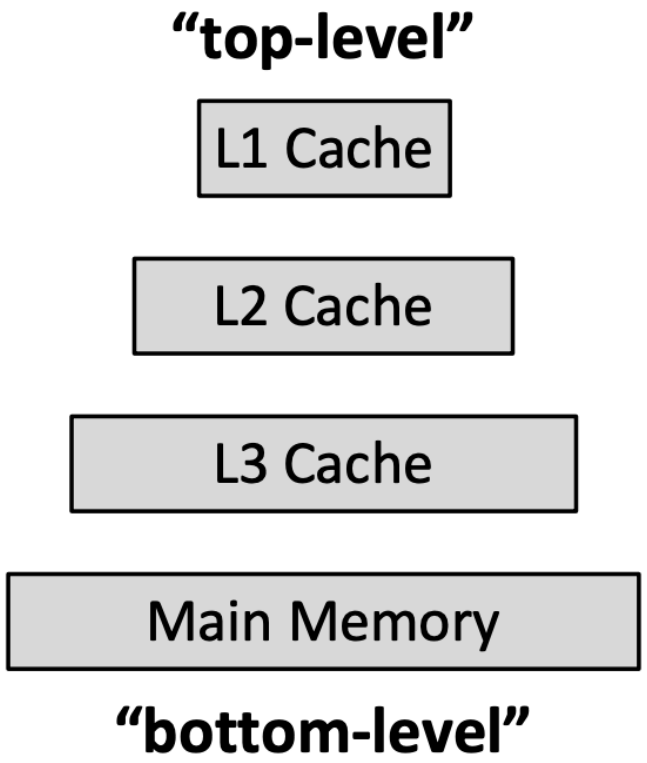
- Temeljni kompromis
  - Brza memorija: mala
  - Velika memorija: spora
- Idea: Hijerarhija memorije




- Latencija, cijena, veličina, propusnost

# Primjer hijerarhije memorije




| capacity | latency |
|---|---|
| 10's of KB | ≈ 1ns |
| 100's of KB | < 5ns |
| several MB | ≈ 10ns |
| several GB | ≈ 100ns |

# Lokacija, lokalitet, lokalnost

- Nečija nedavna prošlost je vrlo dobar prediktor njihove bliske budućnosti.

- Vremenski lokalitet: *„Ako ste upravo nešto učinili, vrlo je vjerojatno da ćete uskoro ponoviti istu stvar”*
  - Budući da ste danas ovdje, postoji velika šansa da ćete biti ovdje iznova i iznova redovito (*i na sljedećem predavanju*)

- Prostorni lokalitet: *„Ako ste nešto učinili, vrlo je vjerojatno da ćete učiniti nešto slično / povezano (u prostoru)”*
  - Svaki put kad te nađem u ovoj učionici, vjerojatno sjediš blizu istih ljudi.

# Lokalitet memorije

- "Tipičan" program ima puno lokaliteta u referencama memorije
  - tipični programi sastoje se od "petlji"

- Vremenski: Program ima tendenciju da se odnosi na istu lokaciju memorije mnogo puta i sve u malom vremenskom okviru

- Prostornim: Program ima tendenciju upućivanja na lokacije memorije u blizini, unutar vremenskog okvira
  - najznačajniji primjeri:
    1. Referenca memorijske lokacije/straince → uglavnom sekvencijalni/streaming
    2. Referenca u polja/vektore → često strujanje/koračanje

# Osnove predmemoriranja (Caching): Iskoristite vremenski lokalitet

- Idea: Spremi nedavno pristupljene podatke u automatski upravljanu brzu memoriju (nazvanu predmemorija)
- Iščekivanje: isti mem. lokaciji će se uskoro ponovno pristupiti

- Načelo vremenskog lokaliteta
  - Nedavno pristupljenim podacima ponovno će se pristupiti u bliskoj budućnosti
  - Ovo je Maurice Wilkes imao na umu:
    - Wilkes, "Slave Memories and Dynamic Storage Allocation," IEEE Trans. On Electronic Computers, 1965.
    - "The use is discussed of a fast core memory of, say 32000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is nearer that of the fast memory than that of the slow memory."

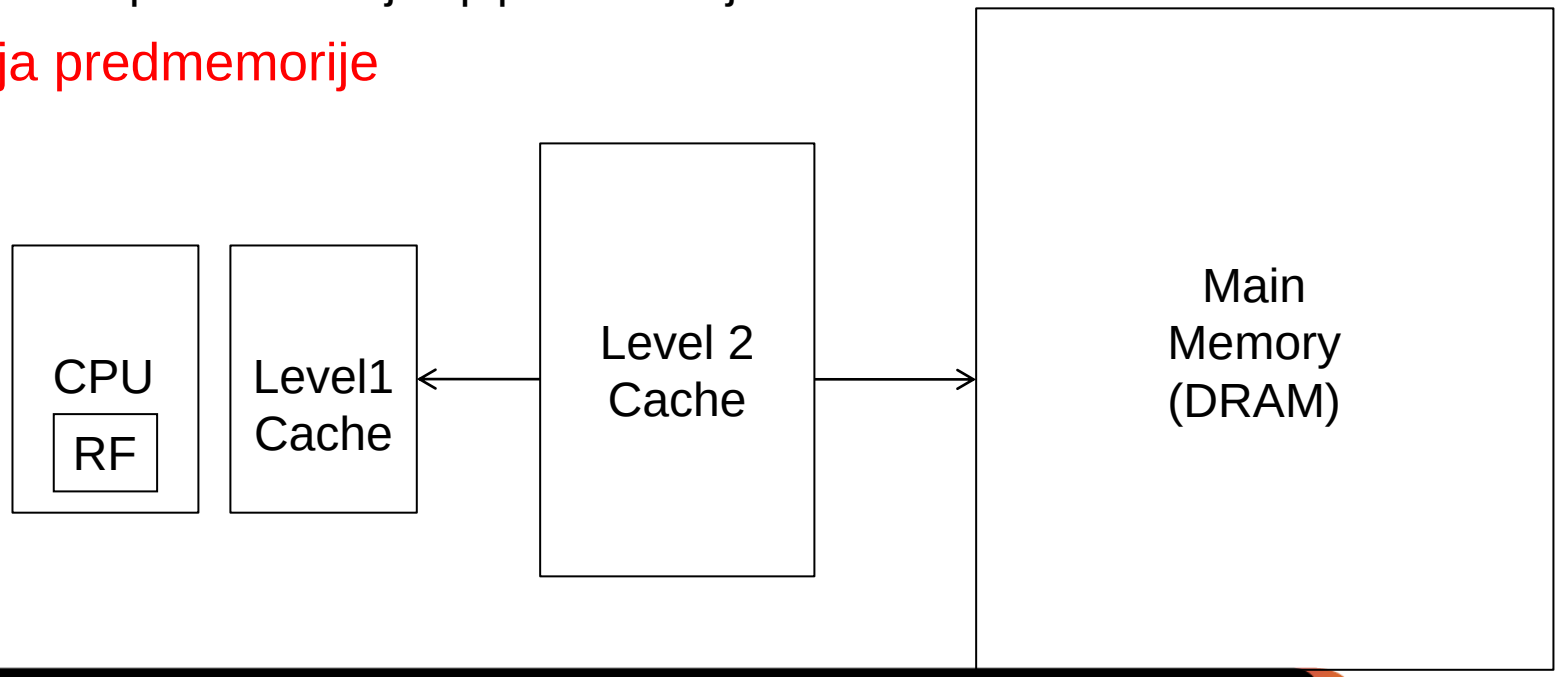
# Osnove predmemoriranja: Iskoristite prostorni lokalitet

- Idea: Pohrana podataka u adrese uz nedavno pristupljenu u brzoj memoriji kojom se automatski upravlja
  - Logički podijelite memoriju na blokove jednake veličine
  - Dohvatite predmemoriranje pristupljenog bloka u cijelosti
- Iščekivanje: uskoro će se pristupiti obližnjim memorijskim lokacijama

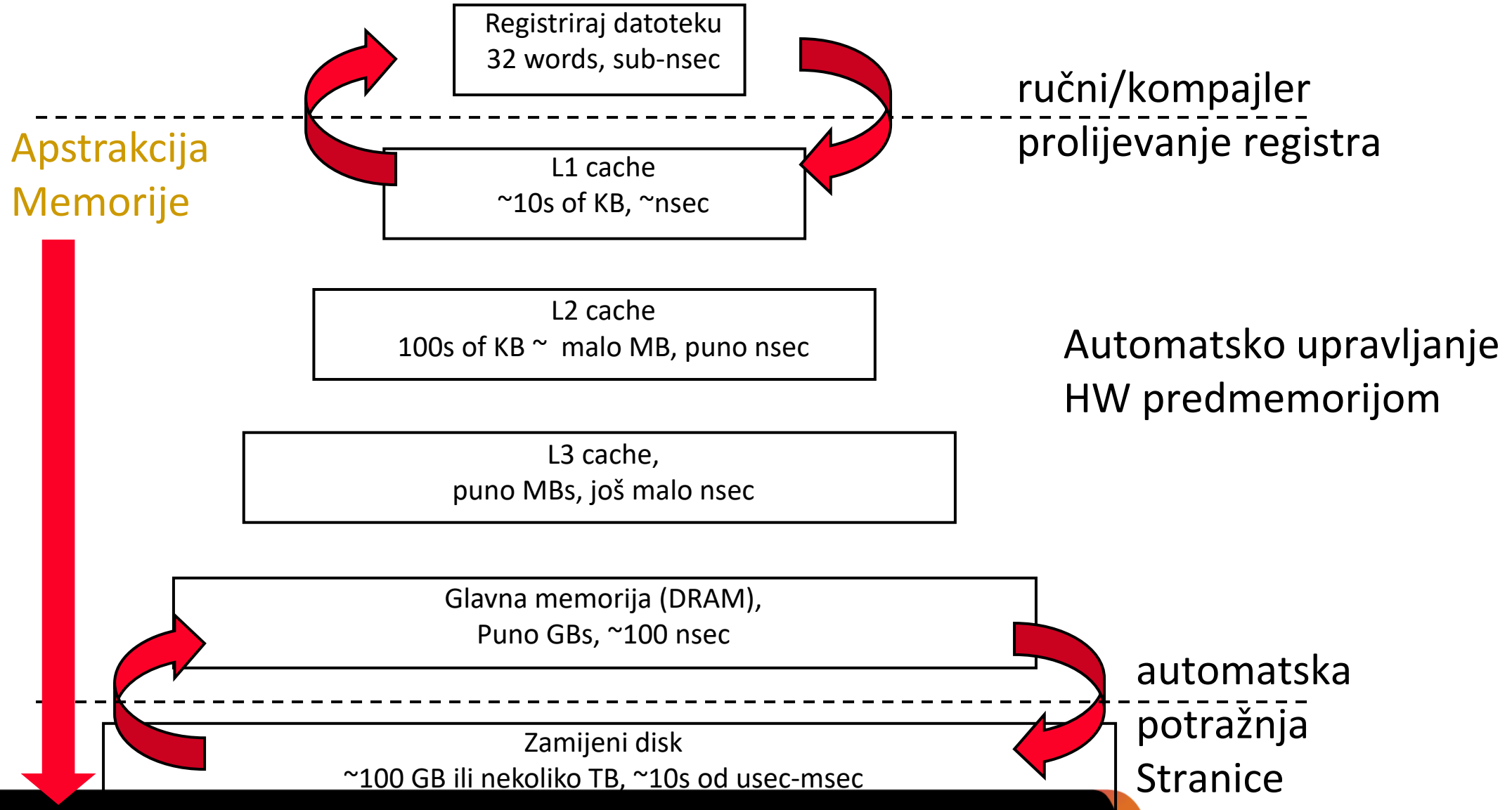
- Princip prostornog lokaliteta
  - Podacima u blizini u memoriji bit će pristupljeno u bliskoj budućnosti
    - E.g., sequential instruction access, array traversal
  - Evo što je IBM 360/85 implementirao:
    - 16 Kbyte cache with 64 byte blocks
    - Liptay, "Structural aspects of the System/360 Model 85 II: the cache," IBM Systems Journal, 1968.

# Predmemoriranje u (pipeline) dizajnu

- Predmemorija mora biti čvrsto integrirana u tijek procesa
  - Idealno, pristup u jednom ciklusu, tako da se operacije ovisne o opterećenju ne odugovlače
- Visokofrekventni tijek podatka → Nije moguće povećati predmemoriju
  - Ali, želimo veliku predmemoriju i pipeline dizajn.
- Idea: Hijerarhija predmemorije



# Moderna hijerarhija memorije



# Osnove predmemorije i operacija

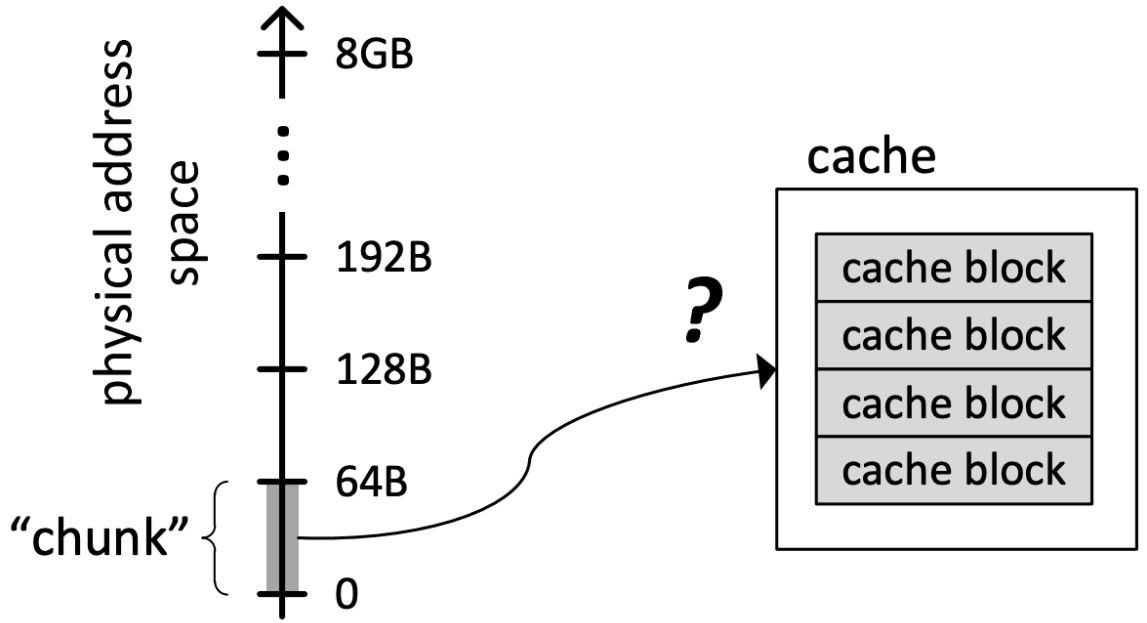
# Cache - predmemorija

- Svaka struktura koja "pamti" korištene (ili proizvedene) podatke
  - kako bi se izbjeglo ponavljanje operacija s dugom latencijom potrebnih za reprodukciju/dohvaćanje podataka od nule
  - Npr.: web cache

- Najčešće u kontekstu dizajna procesora: automatski upravljana struktura memorije
  - npr. memoirajte u brzom SRAM-u najčešće ili nedavno pristupljene LOKACIJE DRAM memorije kako biste izbjegli ponovno plaćanje latencije DRAM pristupa

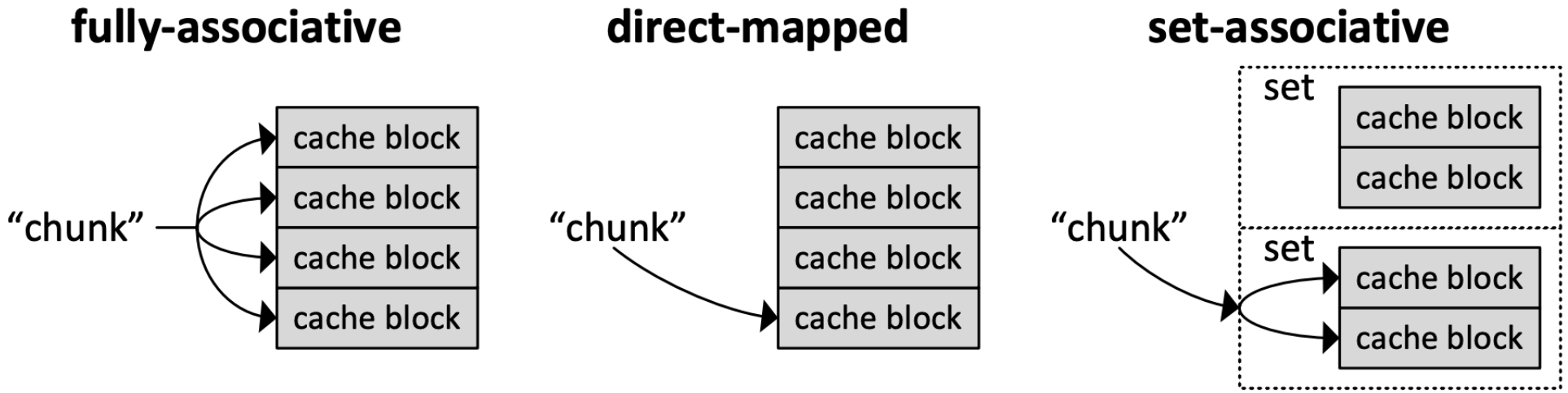
# Logička organizacija predmemorije (I)

- Ključno pitanje: Kako mapirati dijelove glavnog memorijskog adresnog prostora u blokove u predmemoriji?
  - Na koje se mjesto u predmemoriji može smjestiti određeni "glavni dio memorije"?



# Logička organizacija predmemorije (II)

- Ključno pitanje: Kako mapirati dijelove glavnog memorijskog adresnog prostora u blokove u predmemoriji?
  - Na koje se mjesto u predmemoriji može smjestiti određeni "glavni dio memorije"?



# Dizajni predmemorije

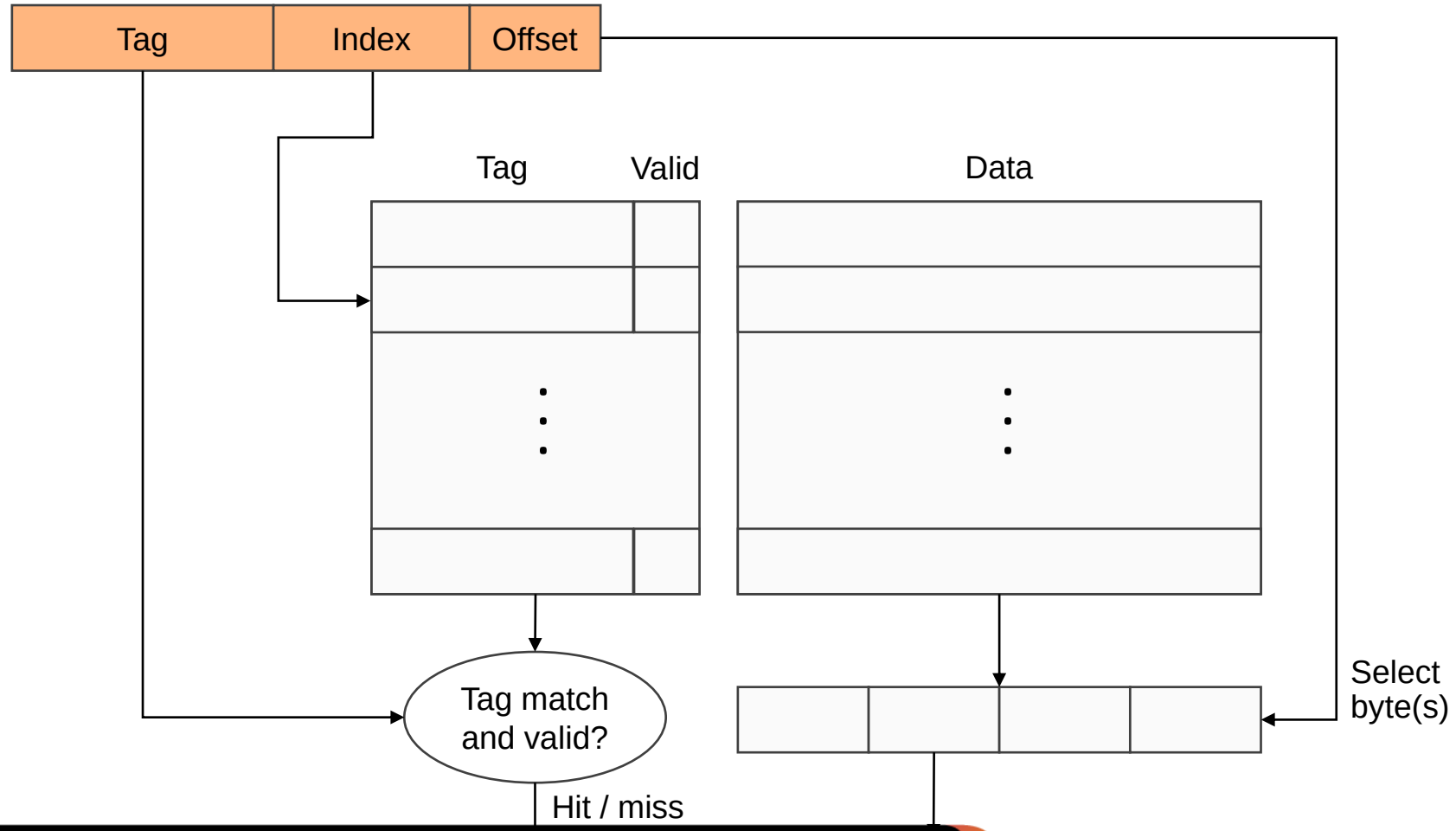
# Izravno preslikana predmemorija

- Jednostavan dizajn jer se svako mjesto memorije mapira samo u jedan blok predmemorije
- Međutim, to često dovodi do sukoba.
  - Kada se više puta pristupi nekoliko lokacija s istim indeksom predmemorije

- Prednost:
  - Jednostavan dizajn, stoga jeftin
  - Brzo pretraživanje
- Nedostatak:
  - Niska stopa pogodak (hit) kada postoji sukob

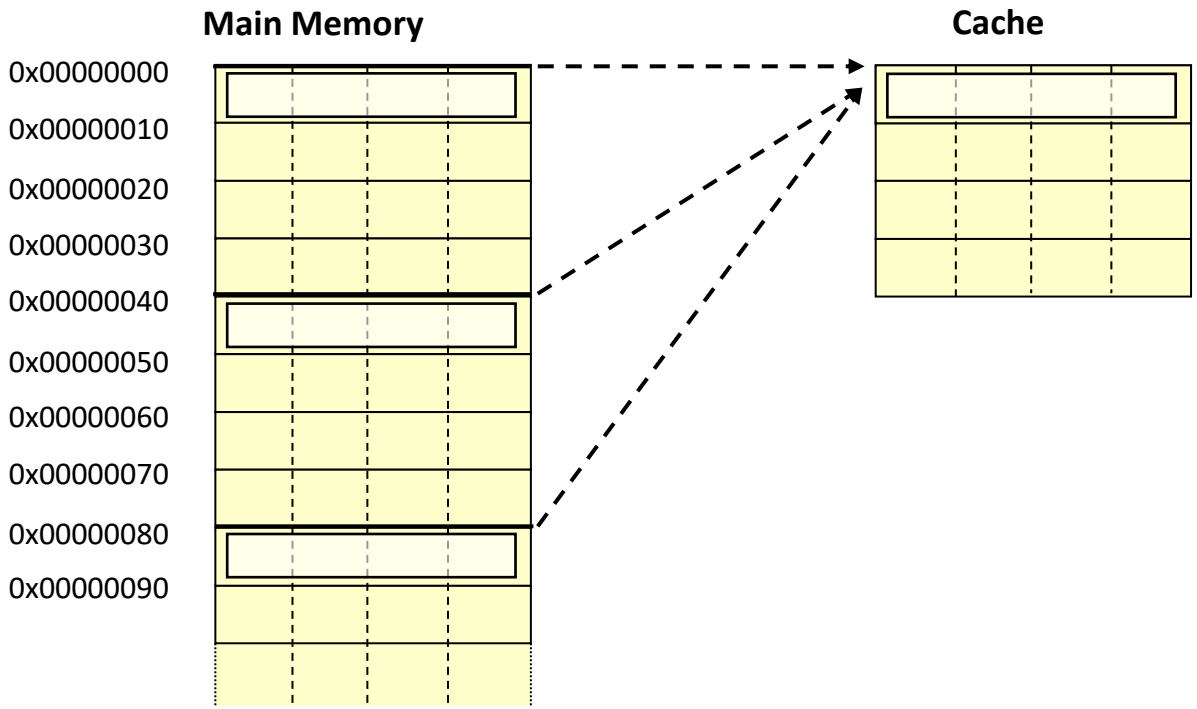
# Izravno preslikana predmemorija

- Indeks koji se koristi za odabir jednog bloka predmemorije
- Uspoređivanje oznaka
  - Ako je valjano i oznake se podudaraju, pogodak (HIT)
- Kada je pogodak, *Offset* odabire početni bajt iz podatkovnog polja.



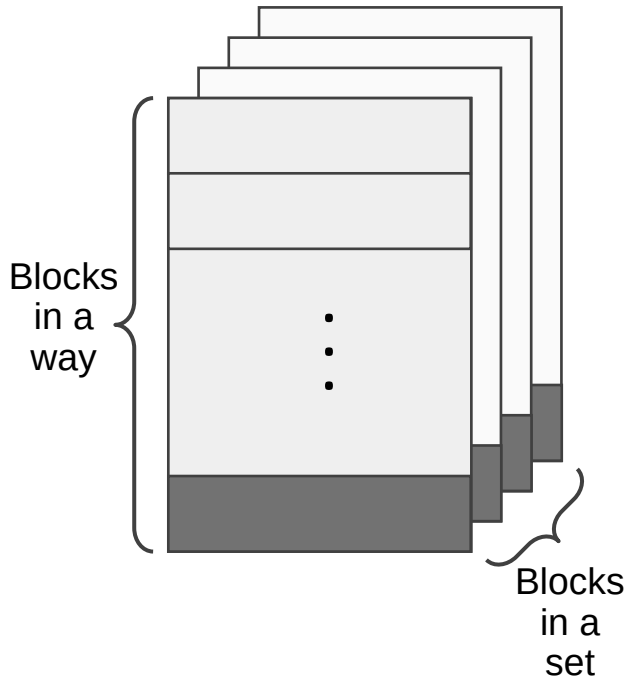
# Izravno preslikana predmemorija

- Loša strana je što svaka adresa u memoriji ima samo jedno mjesto u predmemoriji na koje mapira.
- Više memorijskih mjesta moglo bi se boriti za **isti** redak predmemorije.



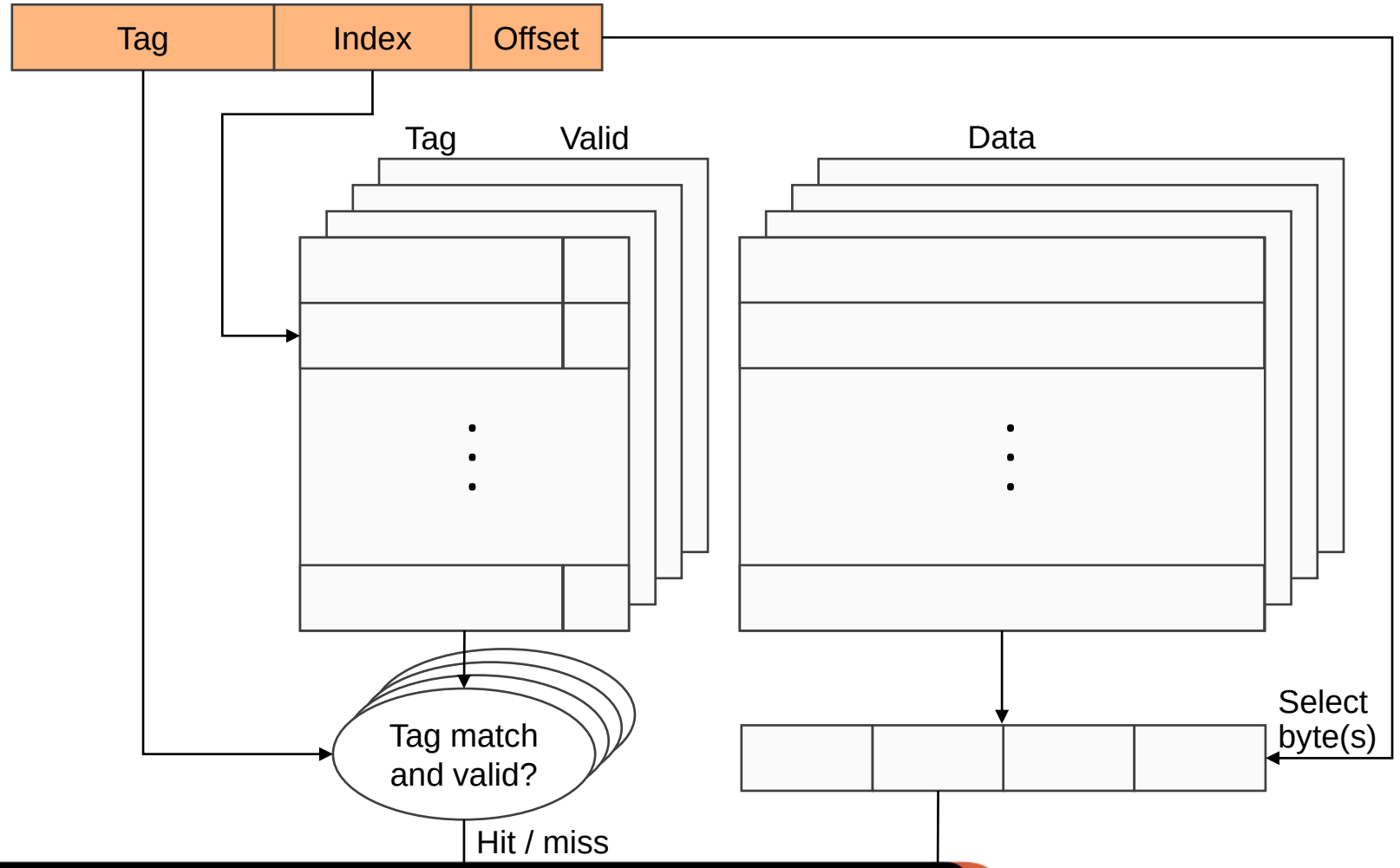
# Postavljena asocijativna predmemorija

- Svako mjesto memorije mapira se u blokove predmemorije N.
  - Svaka grupa blokova N predmemorije naziva se skup - dakle, asocijativni N-skup/grupa.
  - Grupa predmemorija blokova u istom polju, ali s različitim indeksima naziva se stranicom.
- Poboljšana stopa pogodaka u usporedbi s izravno mapiranom predmemorijom
  - Manje sukoba kada postoji nekoliko memorijskih mjesta s istim indeksom predmemorije
- Prednost:
  - Kombinira brzinu izravno mapirane predmemorije s poboljšanom fleksibilnošću u postavljanju
- Nedostatak:
  - Za pronalaženje blokova unutar skupa potreban je složeniji hardver.



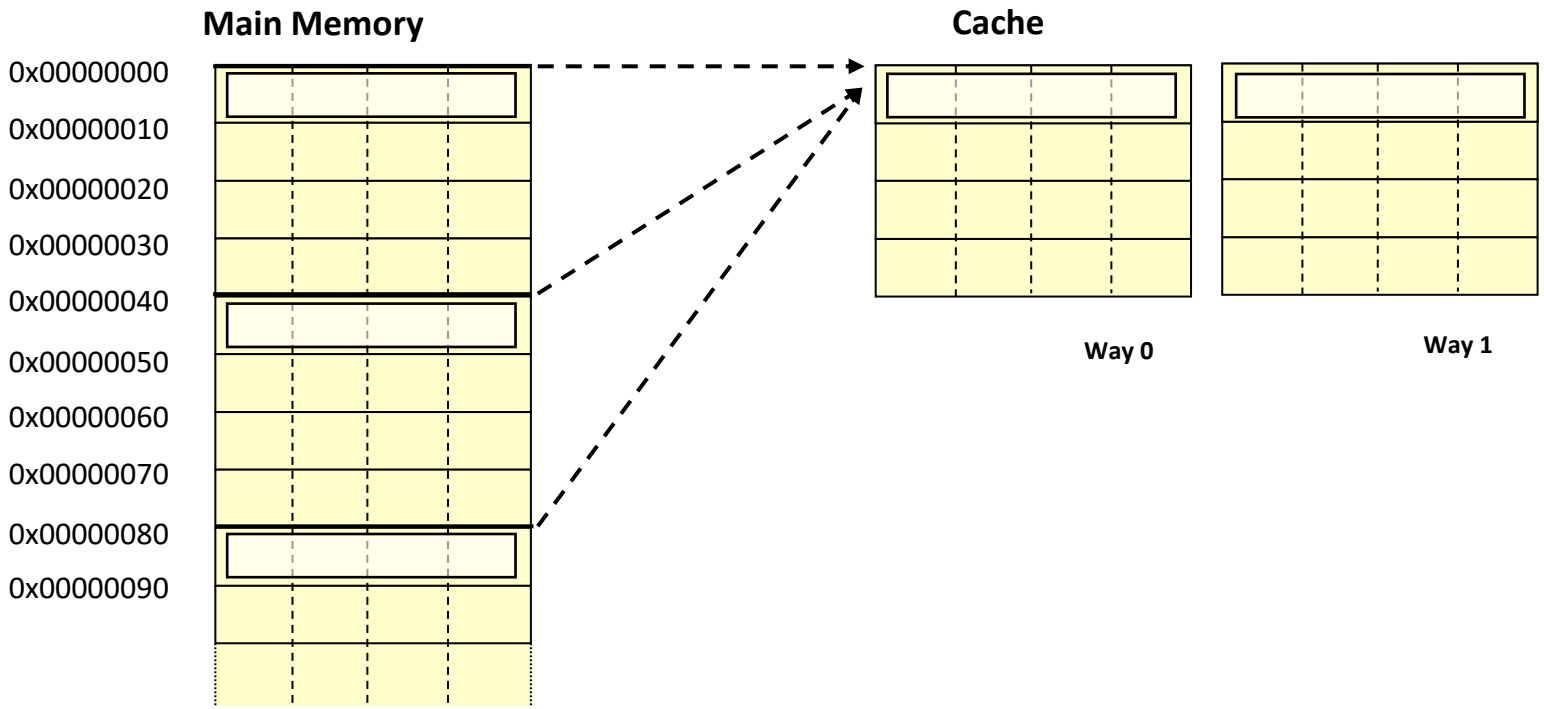
# Postavljena asocijativna predmemorija

- Umjesto samo jedne oznake i podatkovnog polja, postoji više.
- Adresa je podijeljena kao i prije, ali se paralelno traži više polja i provjeravaju njihove oznake.



# Postavljena asocijativna predmemorija

- Sada svaka adresa u memoriji mapira na više lokacija predmemorije (načina), smanjujući sukob.

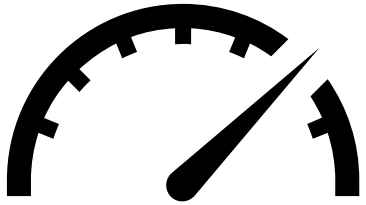


# Potpuno asocijativna predmemorija

- U potpuno asocijativnoj predmemoriji blok se može smjestiti na bilo koje dostupno mjesto u predmemoriji.
- To čini predmemoriju fleksibilnijom, povećavajući brzinu učitavanja.
- Ali to je također složenije, jer traženje bloka uključuje usporedbu sa svim blokovima.

- Prednost:
  - Dobra fleksibilnost: veća stopa pogodaka
- Nedostatak:
  - Traženje podudaranja može biti skupo, zahtjevno za energijom i sporo.

# Performanse predmemorije



# Parametri predmemorije u odnosu na brzinu pogodaka (HIT) / pogreški(MISS)

- Veličina predmemorije
- Veličina bloka
- Asocijativnost
- Pravila zamjene
- Pravilo umetanja/smještaja
- Pravila promocije

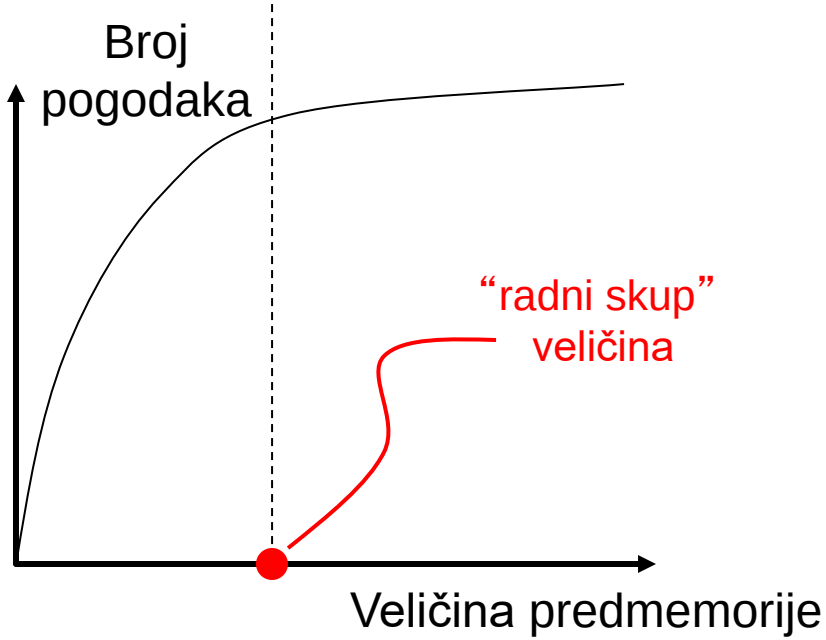
# Veličina predmemorije

- Veličina predmemorije: ukupni kapacitet podataka (ne uključujući oznaku)
- veći može bolje iskoristiti vremenski lokalitet

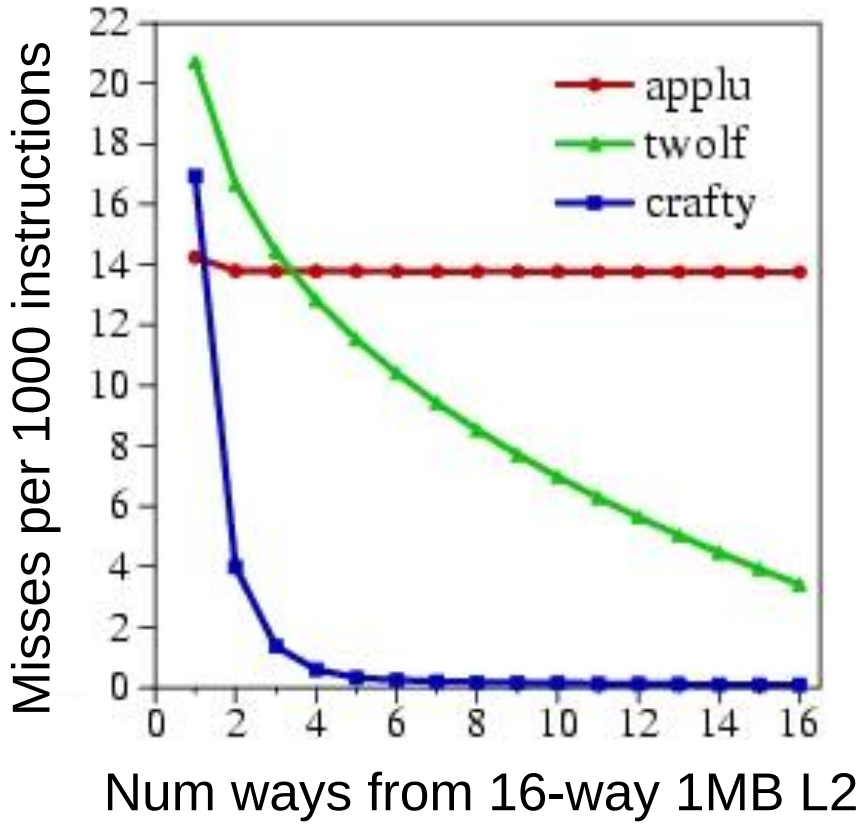
- Prevelika predmemorija negativno utječe na latenciju učitavanja i promašaja
  - veće je sporije

- Premalena predmemorija
  - ne iskorištava dobro vremenski lokalitet
  - korisni podaci koji se često zamjenjuju

- Radni skup: cijeli skup podataka na koje se odnosi izvršna aplikacija
  - Unutar vremenskog intervala



# Prednosti većih predmemorija uvelike variraju




- Prednosti veličine predmemorije uvelike se razlikuju među aplikacijama

Aplikacija s niskim uslužnim programima

Aplikacija visokog uslužnog programa

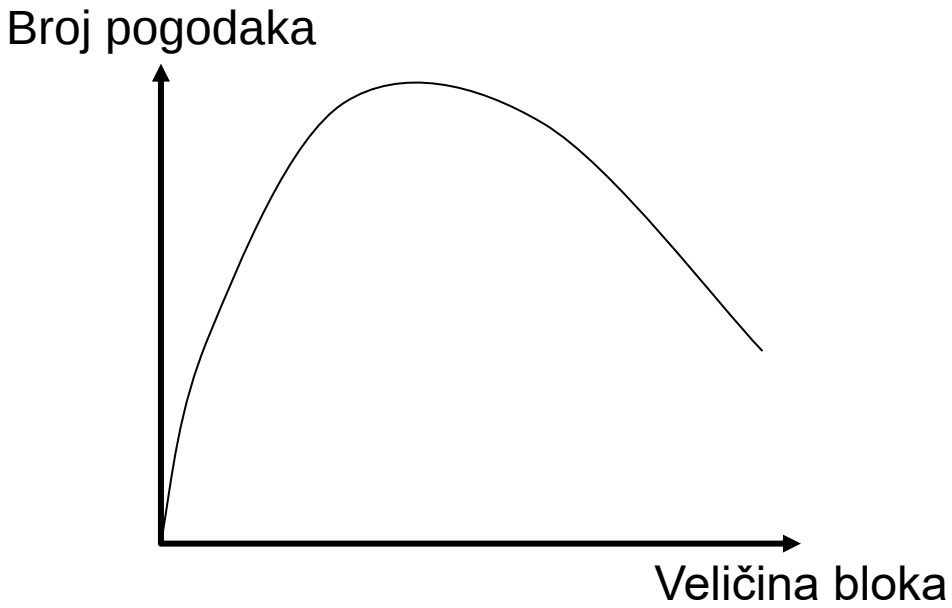
Zasićenje uslužne aplikacije

# Veličina bloka

- Veličina bloka su podaci pridruženi adresnoj oznaci
  - nije nužno jedinica prijenosa između hijerarhija
    - Podblokiranje: Blok podijeljen na više komada (svaki w / V / D bitovi)

- Premaleni blok podataka
  - ne iskorištavaju prostorni lokalitet dobro
  - imaju veću oznaku iznad „glave”

- Preveliki blok podataka
  - premalo ukupnih blokova → iskorištavati vremenski lokalitet nije dobro
  - uzaludno gubljenje prostora, vremena, propusnosti, energije ako prostor nije stvarno jako+jako velik

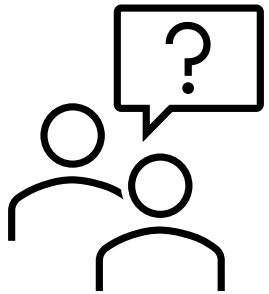


# Veliki blokovi: kritična riječ i podblokiranje

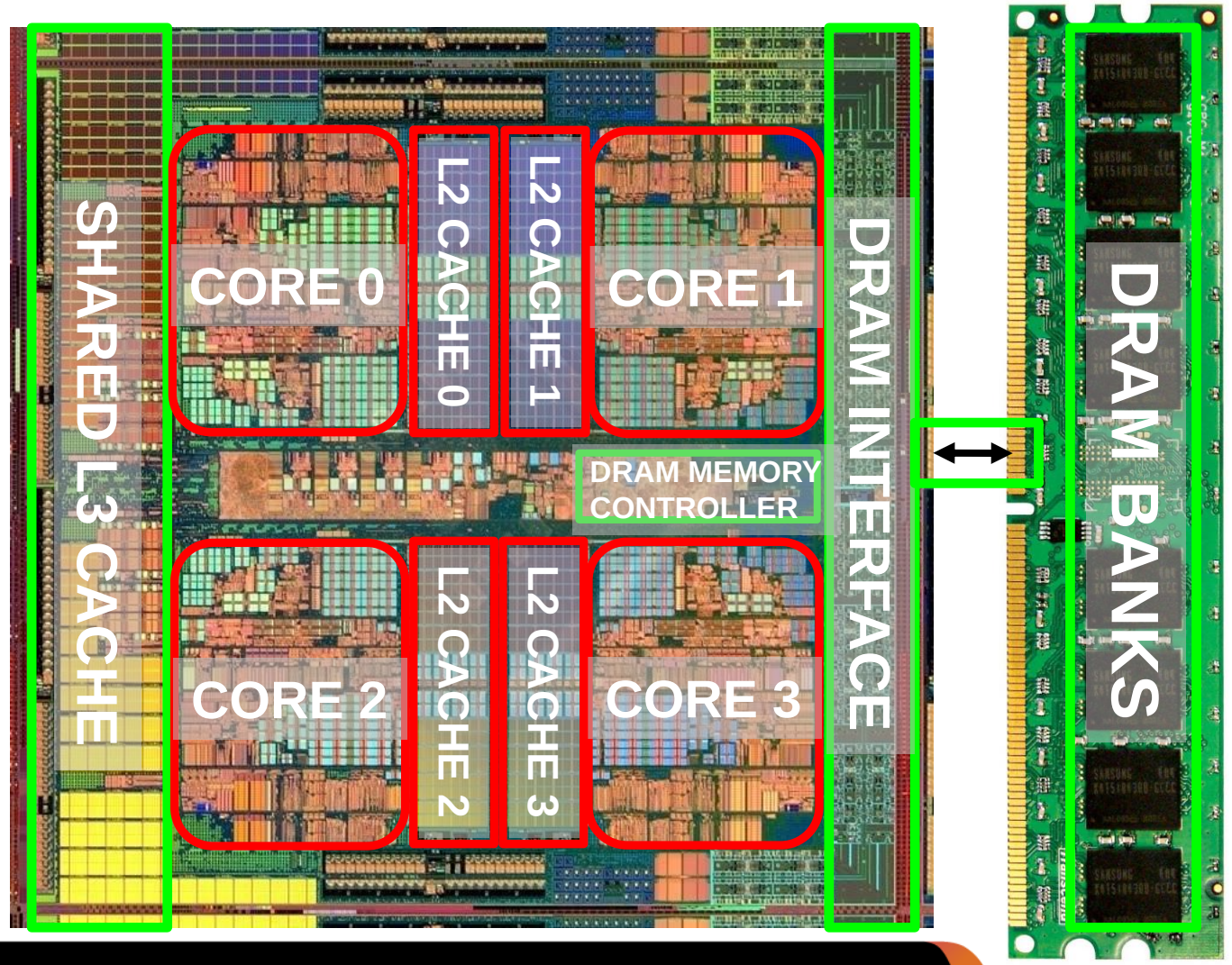
- Ispunjavanje velikih predmemorijskih blokova može potrajati dugo vremena u predmemoriji
  - Idea: Blok predmemorije ispuni sa „critical-word first” (kritični podatak)
  - Odmah dostavite kritične podatke procesoru

- Veliki predmemorski blokovi mogu trošiti propusnost sabirnice
  - Idea: Podjela bloka na podblokove
  - Pridruživanje zasebnih „valjanih i prljavih” bitova za svaki podblok
  - Ponavljanje: Kada je ovo korisno?

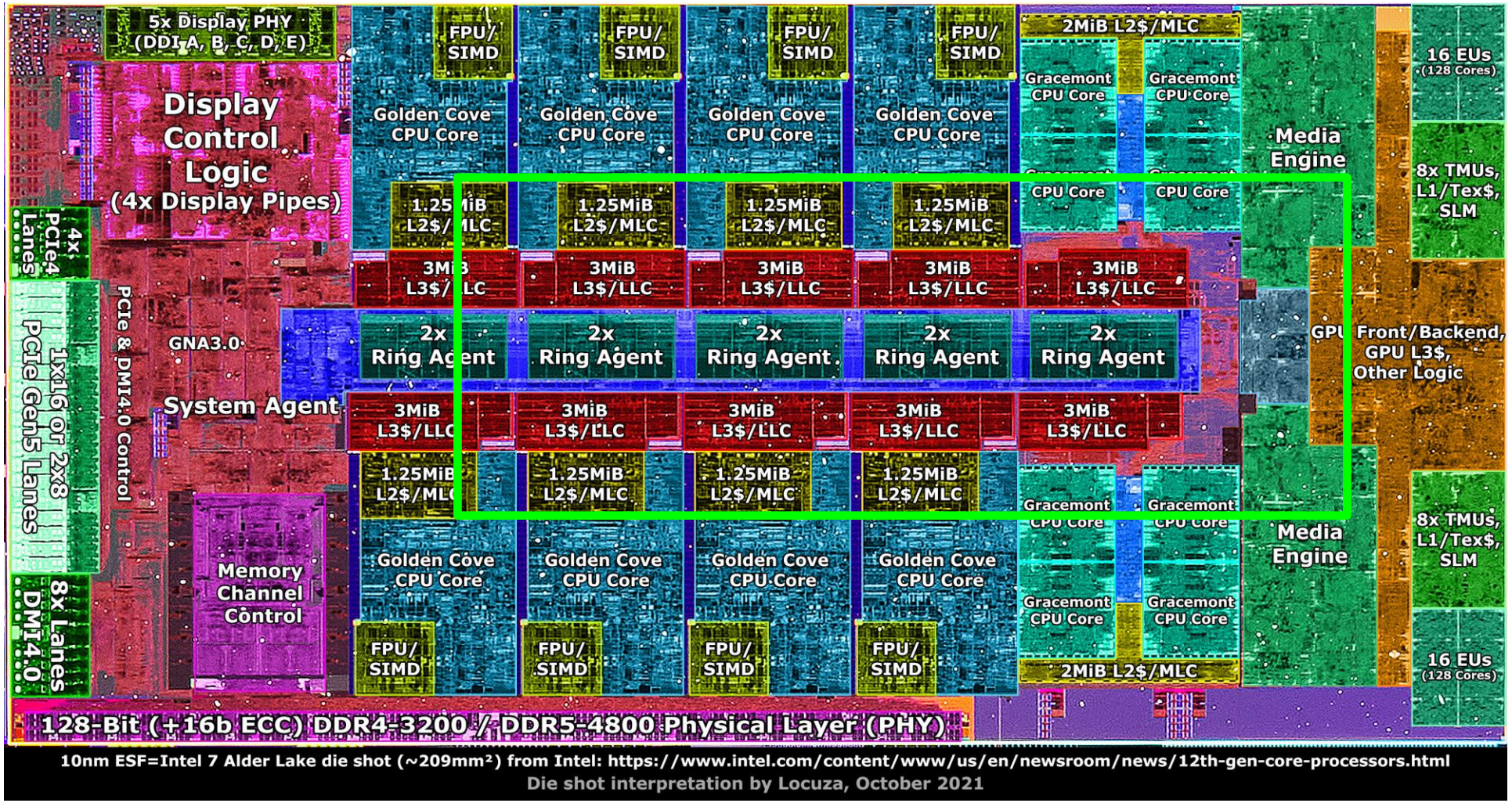
# Pitanja u predmemoriranju višejezgrenih procesora



# Predmemorije u višejezgrenom sustavu

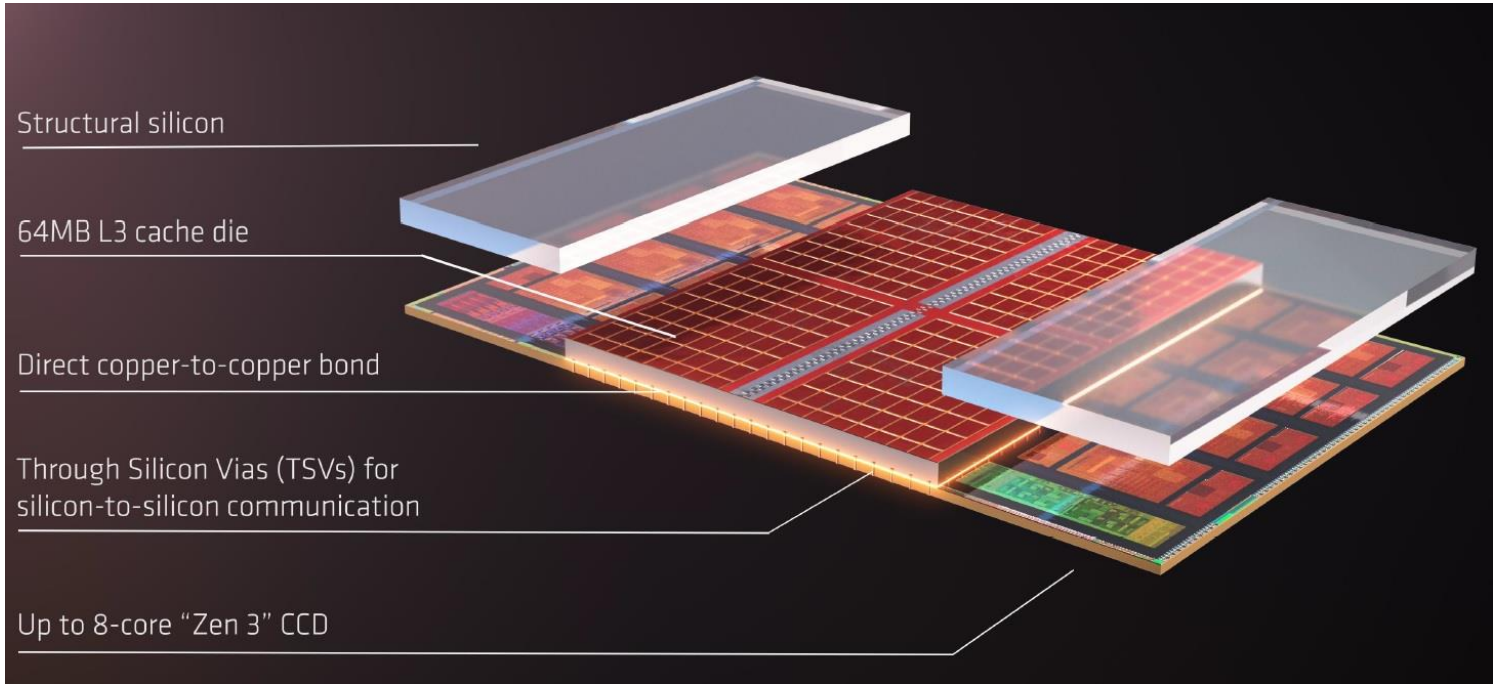


# Predmemorije u višejezgrenom sustavu




Intel Alder Lake, 2021

# Predmemorije u višejezgrenom sustavu

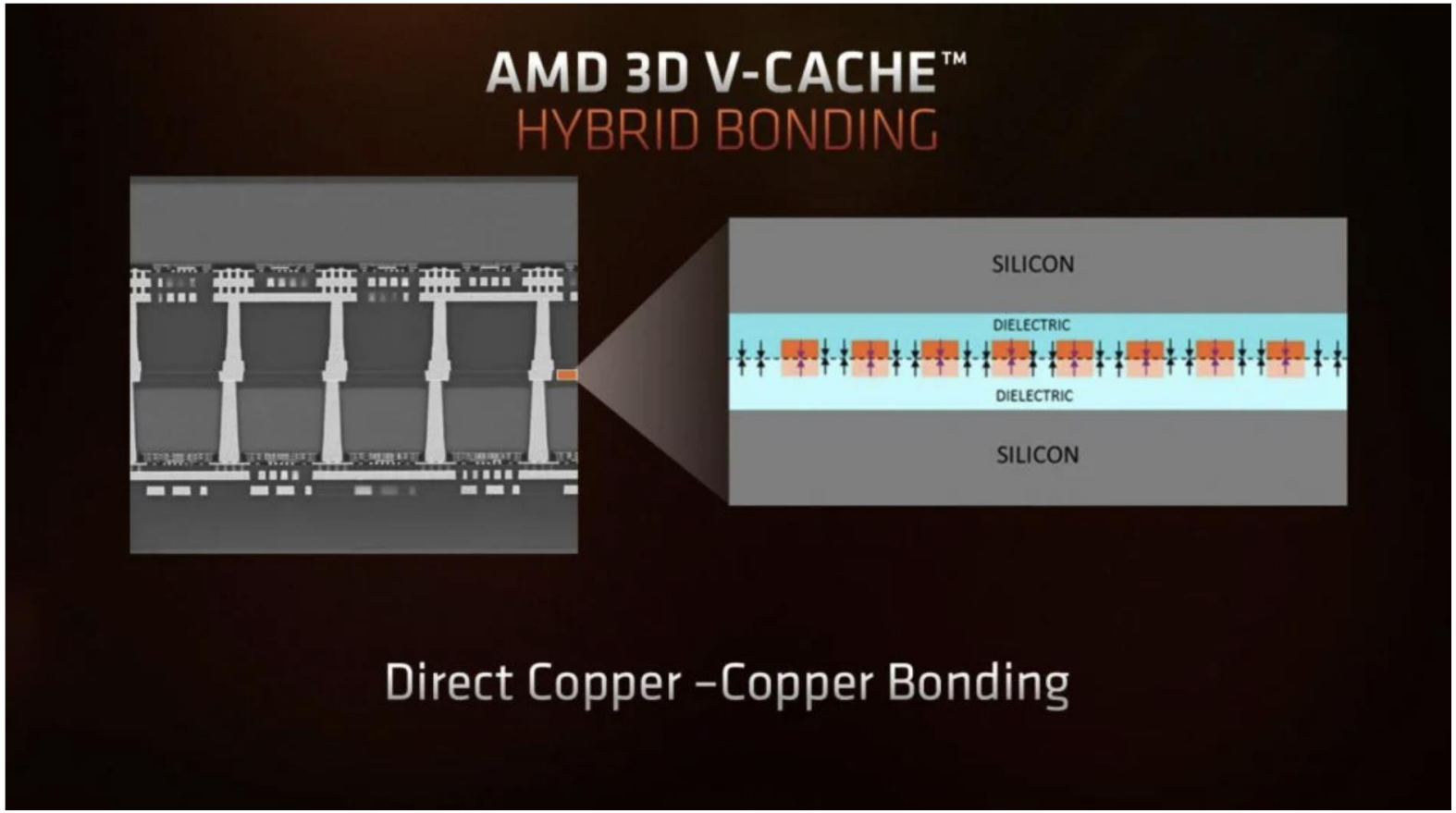



AMD povećava L3 veličinu svojih 8-jezgrenih Zen 3 procesora s 32 MB na 96 MB

Dodatnih 64 MB L3 predmemorije

Složeno na vrhu procesora
- Povezano pomoću Through Silicon Vias (TSVs)
- Ukupno 96 MB L3 predmemorije

# Tehnologija 3D slaganja (*3D Stacking*): primjer




AMD Ryzen 7 5800X3D: The 3D V-Cache in detail (4)

Source: AMD

# Predmemorije u višejezgrenim sustavima

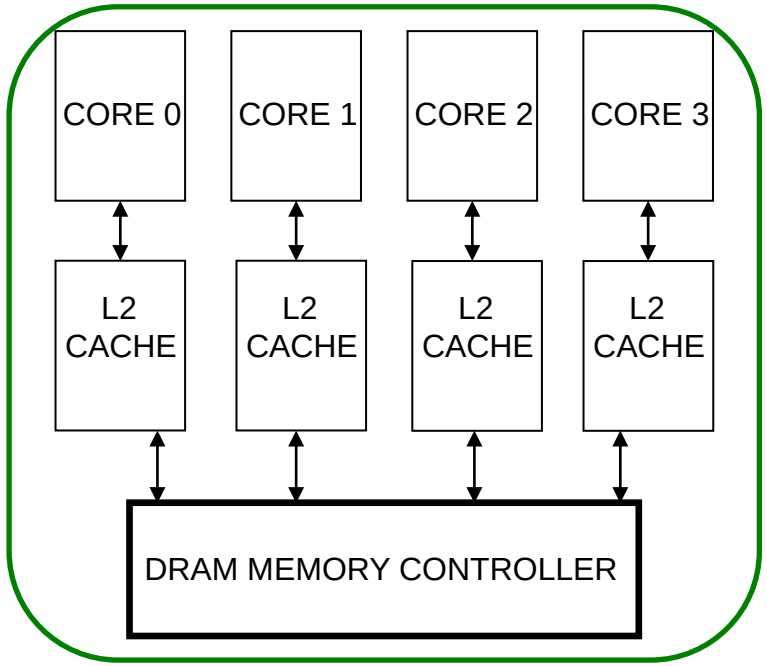
- Učinkovitost predmemorije postaje još važnija u višejezgrenom/višenitnom sustavu
  - Propusnost memorije je na višem stupnju
  - Prostor predmemorije ograničen je resurs u jezgrama/dretvama

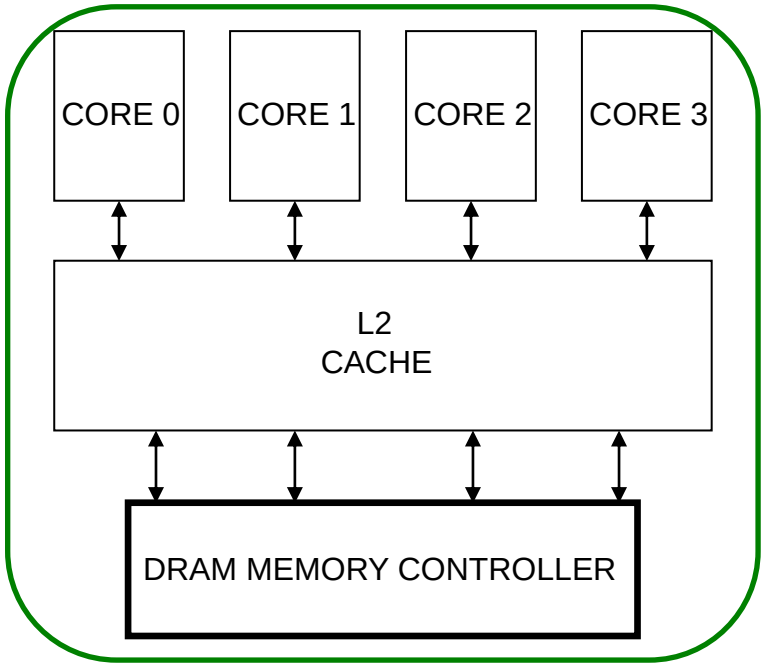
- Kako dizajniramo predmemorije u višejezgrenom sustavu?

- Mnogo odluka i pitanja:
  - Djeljena ili privatna predmemorija?
  - Kako da maksimiziramo performanse cijelog sustava?
  - Kako da pružimo QoS i prediktivne performanse za različite dretve (threads) u djeljenoj predmemoriji?
  - Trebaju li algoritmi za upravljanje predmemorijom biti svjesni dretvi?
  - Koliko prostora treba dodjeliti dretvama u djeljenoj predmemoriji?
  - Trebamo li kompresirati podatke u predmemoriji?
  - Kako da bolje iskoristimo predikciju i upravljanje predmemorijom?

# Privatna vs. dijeljena predmemorija

- Privatna predmemorija: Predmemorija pripada jednoj jezgri (zajednički blok može biti u više predmemorija)
- Dijeljena predmemorija: Predmemoriju zajednički koristi više jezgri



# Koncept i prednosti dijeljenja resursa

- Idea: Umjesto posvećivanja hardverskog resursa hardverskom kontekstu, dopustite da ga koristi više konteksta (zašto ne bi svi dijelili HW)
  - Primjeri resursa: funkcionalne jedinice, sabirnica, predmemorije, memorija…
- Zašto?

+ Zajedničko korištenje resursa poboljšava iskorištenost/učinkovitost → propusnost
  - Kada resurs ostane u stanju mirovanja za jednu dretvu, druga dretva ga može koristiti; nema potrebe za repliciranjem dijeljenih podataka
+ Smanjuje latenciju komunikacije
  - Na primjer, podaci koji se dijele između više dretvi mogu se čuvati u istoj predmemoriji u višeslojnim procesorima
+ Kompatibilno s modelom programiranja zajedničke memorije

# Nedostaci dijeljenja resursa

- Dijeljenje resursa stvara utrku/natjecanje za resursima
  - Kada resurs nije u stanju mirovanja, druga dretva ga ne može koristiti
  - Ako prostor zauzima jedna dretva, druga dretva ga želi ponovno zauzeti

- Ponekad smanjuje performanse svake ili neke dretve
  - Performanse niti mogu biti gore nego kad se izvode samostalno
- Uklanja izolaciju performansi → nedosljedne performanse u svim ciklusima
  - Izvedba dretvi ovisi o nasljednim dretvama (trebam podatak od druge dretve)
- Nekontrolirano (free-for-all) djeljenje degradira QoS
  - Uzrokuje nepravednost, gladovanje (starvation)

Potreba za učinkovitim i pravednim korištenjem zajedničkih resursa (Algoritam u OSu?!)

# Zajedničke predmemorije između jezgri

- Prednosti:
  - Visok efektivni kapacitet
  - Dinamičko particioniranje raspoloživog prostora predmemorije
    - Nema fragmentacije zbog statičkog particioniranja
    - Ako jedna jezgra ne koristi neki prostor, druga jezgra može
  - Lakše održavati koherentnost (predmemorski blok nalazi se na jednom mjestu)

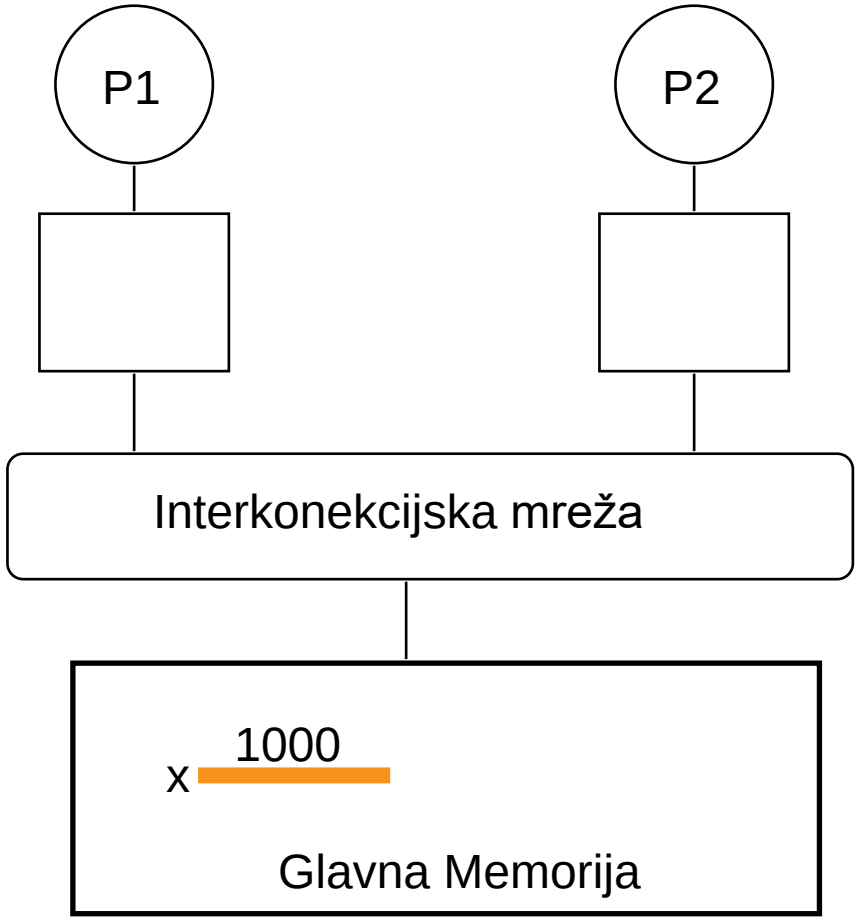
- Nedostaci:
  - Sporiji pristup (predmemorija nije čvrsto povezana s jezgrom)
  - U Jezgrama nastaju promašaji u sukobu zbog pristupa drugih jezgri
    - Promašaji zbog međujezgrenih smetnji
    - Neke jezgre mogu uništiti proj pogodaka drugih jezgri
  - Teže je jamčiti minimalnu razinu usluge (ili pravednosti) svakoj jezgri (koliko prostora, koliko propusnosti?)

# Koherentnost predmemorije (međusobno usklađene predmemorije)

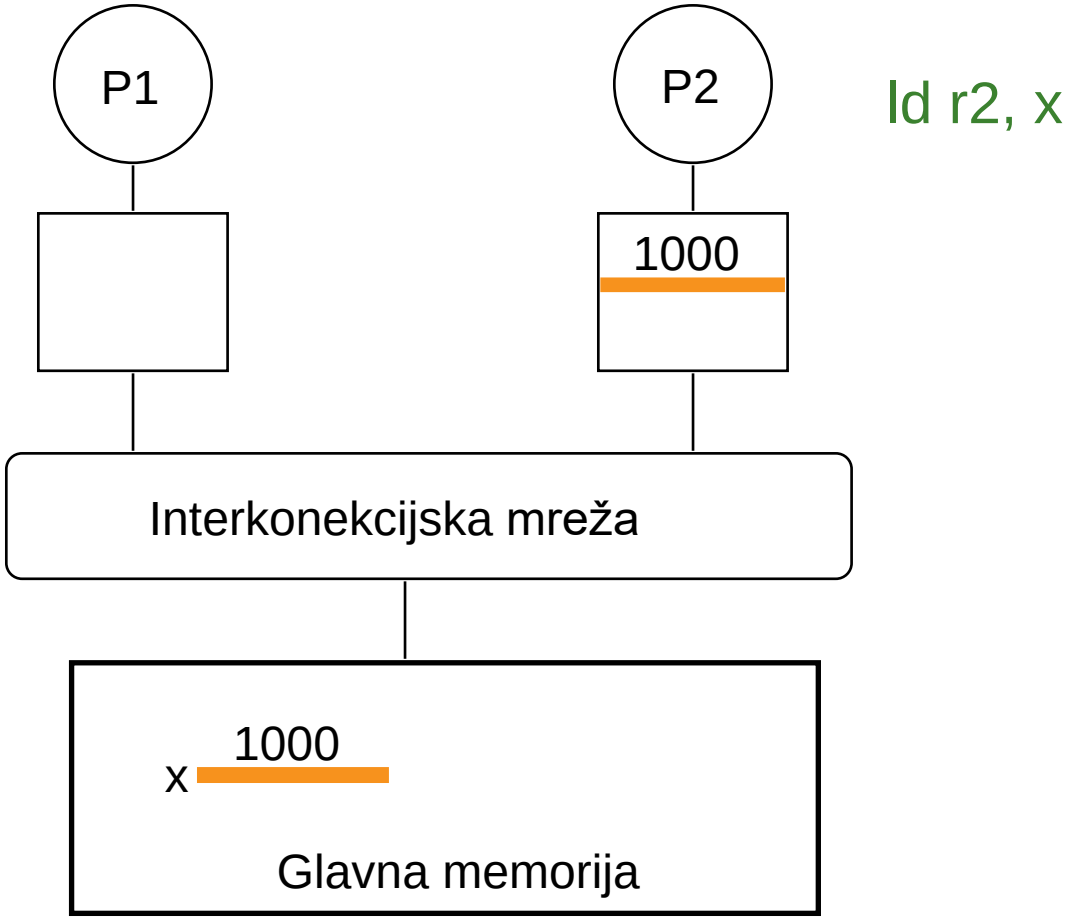
# Koherentnost predmemorije

- Osnovno pitanje:

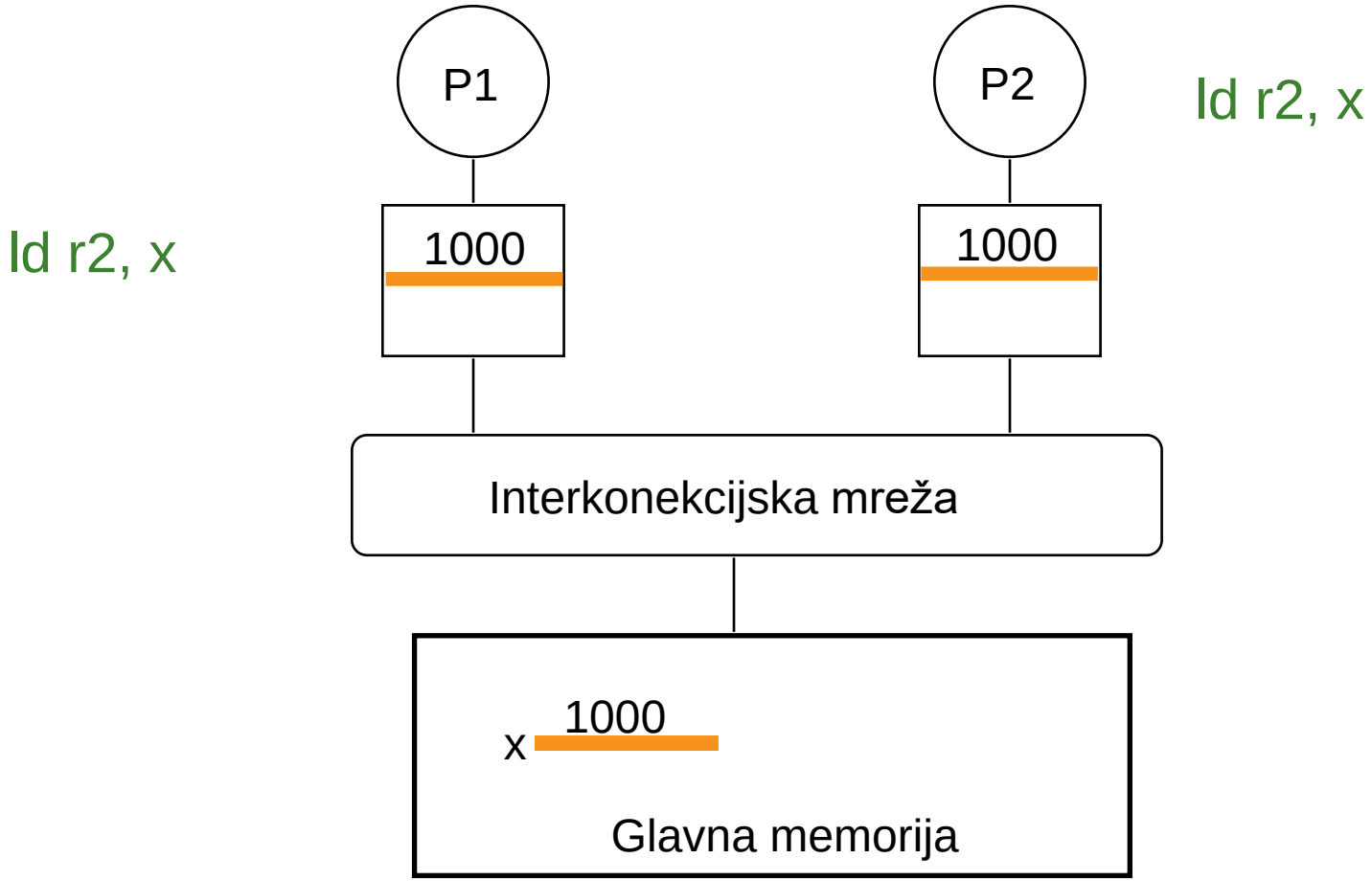
Ako više procesora predmemorira isti blok, kako osigurati da svi vide dosljedno stanje?



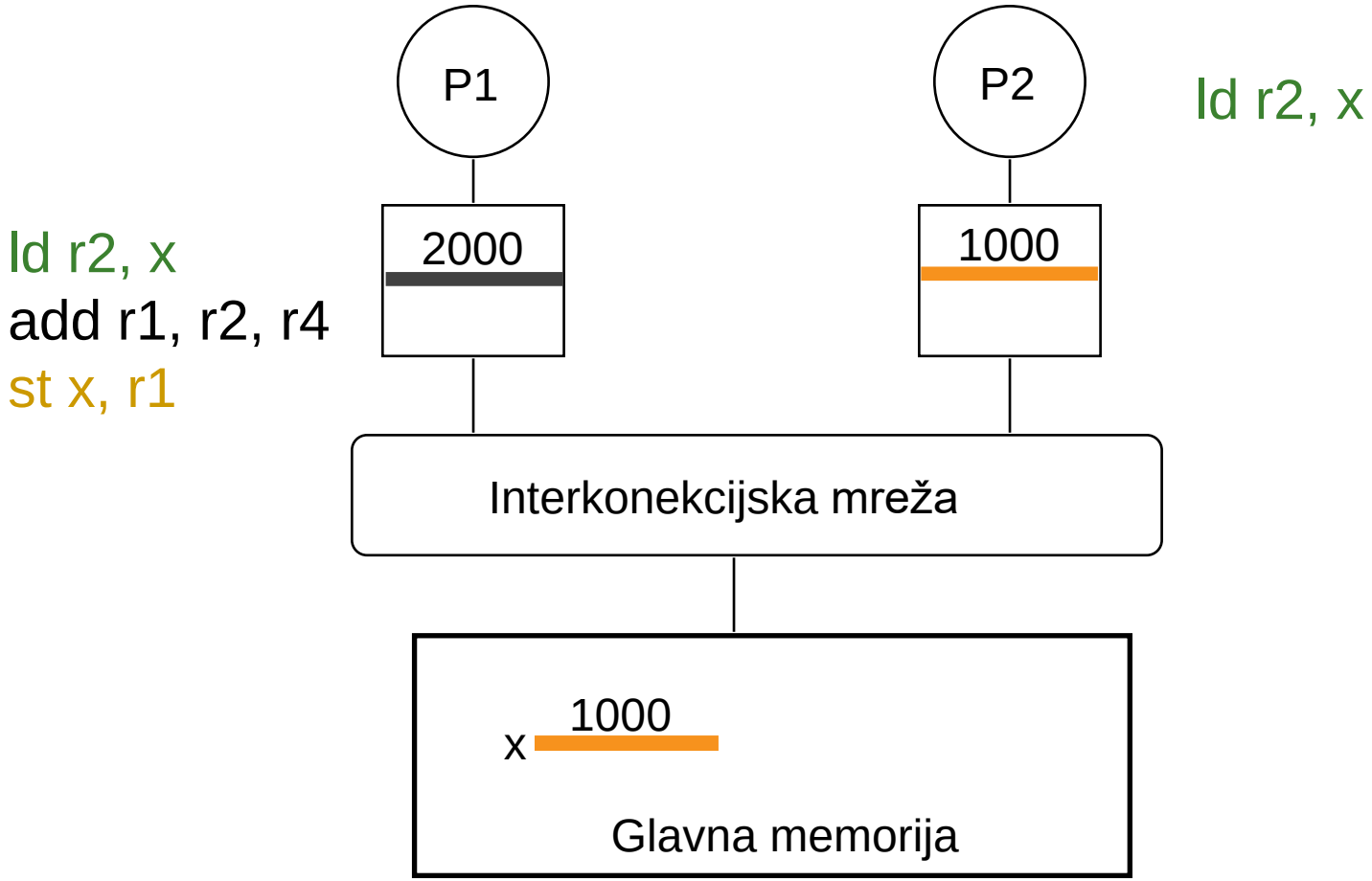
# Problem koherentnosti predmemorije



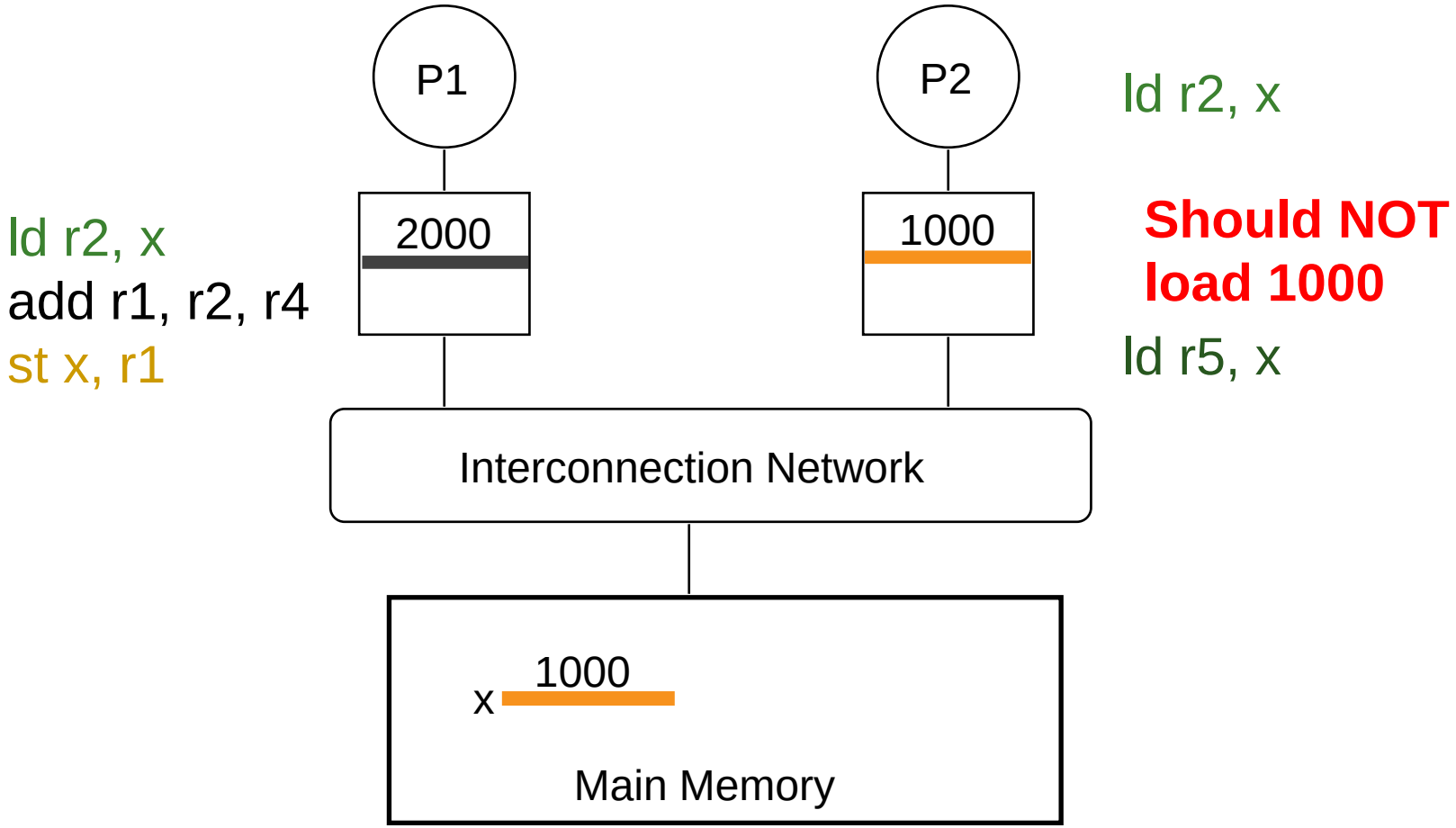
# Problem koherentnosti predmemorije



# Problem koherentnosti predmemorije

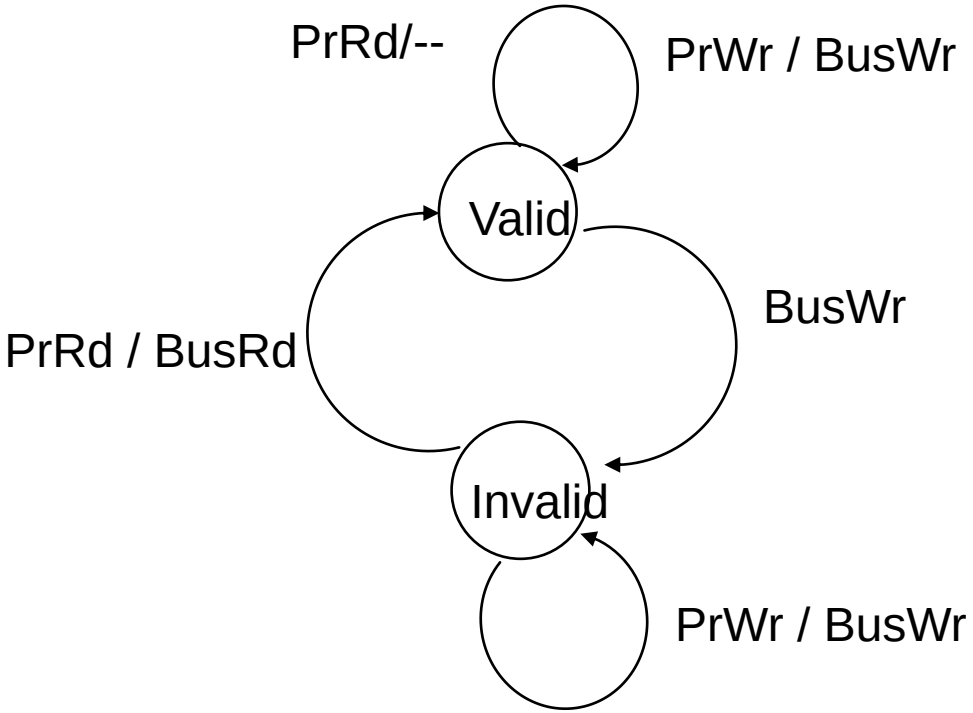


# Problem koherentnosti predmemorije



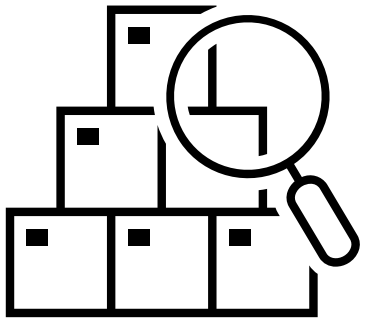
# Vrlo jednostavna shema koherentnosti

- Idea: Sve predmemorije „prisluškuju" (promatraju) međusobne operacije pisanja /čitanja. Ako procesor piše u blok, svi ostali poništavaju blok.
- Jednostavan protokol:




- Predmemorija za pisanje bez pisanja
- Akcije lokalnog procesora u predmemorijskom bloku: PrRd, PrWr,
- Akcije koje se emitiraju u sabirnici za blok: BusRd, BusWr

# Vrste memorije



# Apstrakcija: Virtualna u odnosu na fizičku memoriju

- Programer vidi virtualnu memoriju
  - Pretpostavlja da je memorija "beskonačna"
- Stvarnost: Fizička memorija je mnogo manje veličine od one za koju programer pretpostavlja
- Sustav (system software + hardware, cooperatively) mapira adrese virtualne memorije u fizičku memoriju
  - Sustav automatski upravlja fizičkim memorijskim prostorom transparentno prema programeru

+ Programer ne mora znati fizičku veličinu memorije niti njome upravljati → Mala fizička memorija može se pojaviti kao ogromna programeru → Život je olakšan programeru

-- Složeniji sistemski softver i arhitektura

Klasičan primjer kompromisa programera i (mikro)arhitekta



Zahtijeva zaobilazak i mapiranje između virtualnih i fizičkih prostora

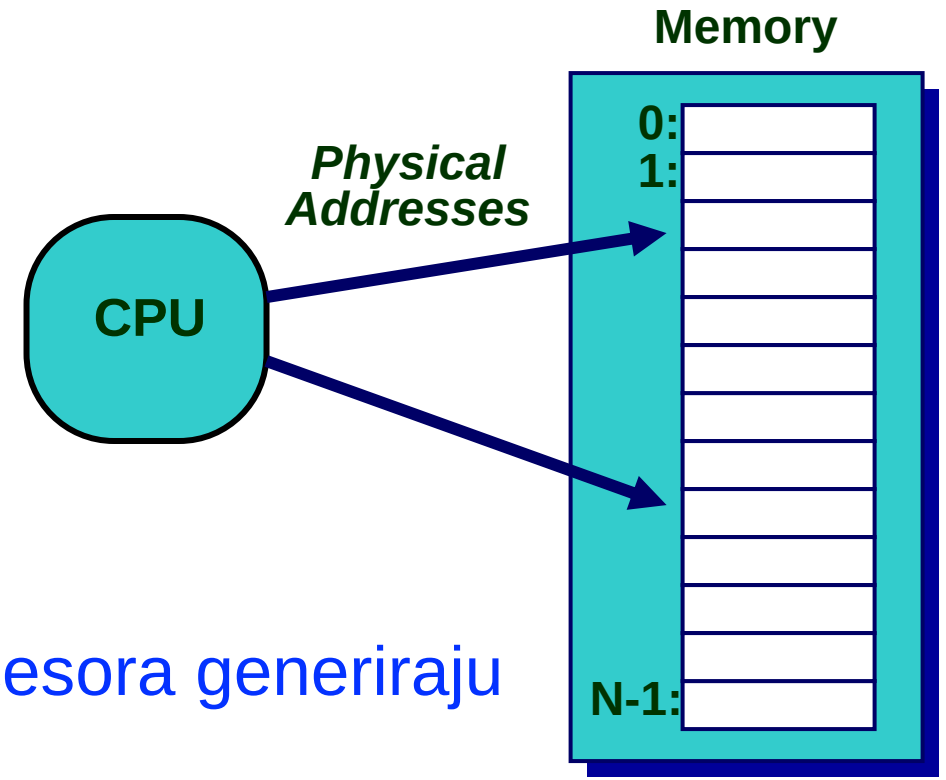
# Prednosti automatskog upravljanja memorijom

- Programer se ne bavi fizičkim adresama
- Svaki proces ima svoj:
  - Virtualni adresni prostor (jako veliki)
  - Neovisno mapiranje virtualnih → fizičke adrese

- Omogućuje
  - Kod i podaci koji će se nalaziti bilo gdje u fizičkoj memoriji (premještanje i fleksibilno mjesto podataka)
  - Izolacija/odvajanje koda i podataka različitih procesa u fizičkoj memoriji (zaštita i izolacija)
  - Dijeljenje koda i podataka između više procesa (sharing)

# Sustav samo s fizičkom memorijom

- Primjeri:
  - većina Cray super-računala
  - rana osobna računala (PCs)
  - mnogi stariji ugrađeni sustavi




Upute za čitanje ili pohranu procesora generiraju fizičke memorijske adrese

# Problem

- Fizička memorija je ograničene veličine (cijena)
  - Što ako vam treba više?
  - Treba li programer biti zabrinut zbog veličine kodnih/podatkovnih blokova koji odgovaraju fizičkoj memoriji?
  - Treba li programer upravljati kretanjem podataka s diska na fizičku memoriju?
- Više programa možda će trebati fizičku memoriju
  - Treba li programer osigurati da svi procesi (različiti programi) mogu stati u fizičku memoriju?
  - Treba li programer osigurati da dva procesa ne koriste nenamjerno ili pogrešno ne koriste isti dio fizičke memorije?
- ISA može imati adresni prostor veći od fizičke veličine memorije
  - Npr. 64-bitni adresni prostor s mogućnošću adresiranja bajtova
  - Što ako nemate dovoljno fizičkog pamćenja?

# Poteškoće s izravnim fizičkim rješavanjem

- Programer mora upravljati fizičkim memorijskim prostorom
  - Nezgodno i teško
  - Teže kada imate **više procesa**
- Teško je podržati premještanje koda i podataka
  - Adrese su izravno navedene u programu
- Teško je podržati više procesa
  - Zaštita i izolacija između više procesa
  - Dijeljenje fizičkog memorijskog prostora bez problema
- Teško je podržati razmjenu podataka/kodova u svim procesima
  - Različiti procesi moraju (ili mogu) se pozivati na istu fizičku adresu

# Virtualna memorija

- Idea: Dajte svakom programu iluziju velikog adresnog prostora dok imate malu fizičku memoriju
  - Tako da programer ne brine o upravljanju fizičkom memorijom (unutar procesa ili kroz procese)
- Programer može pretpostaviti da ima "**beskonačnu**" količinu fizičke memorije
- Hardver i softver kooperativno i automatski upravljaju fizičkim memorijskim prostorom kako bi pružili iluziju
  - Iluzija se održava za svaki neovisni proces

# Osnovni mehanizam

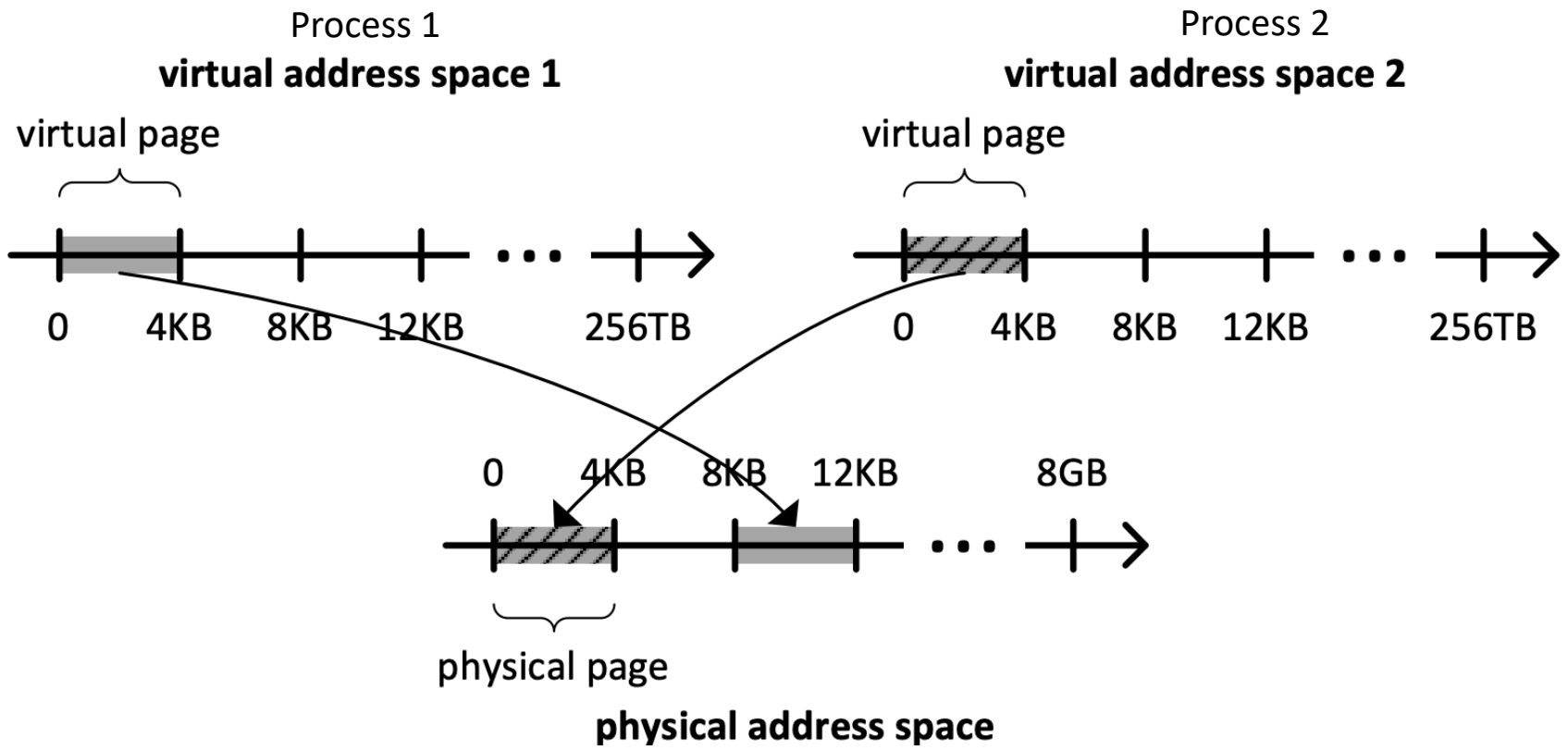
- Zaobilaženje (u adresiranju) i mapiranju

- Adresa generirana svakom instrukcijom u programu je "virtualna adresa"
  - tj. to nije fizička adresa koja se koristi za adresiranje glavne memorije
  - tzv. “linear address” u x86

- Mehanizam „translacije adrese" mapira ovu adresu na "fizičku adresu"
  - tzv. “real address” u x86
  - Mehanizam za prevođenje adresa (translacija) može se zajedno implementirati u hardver i softver

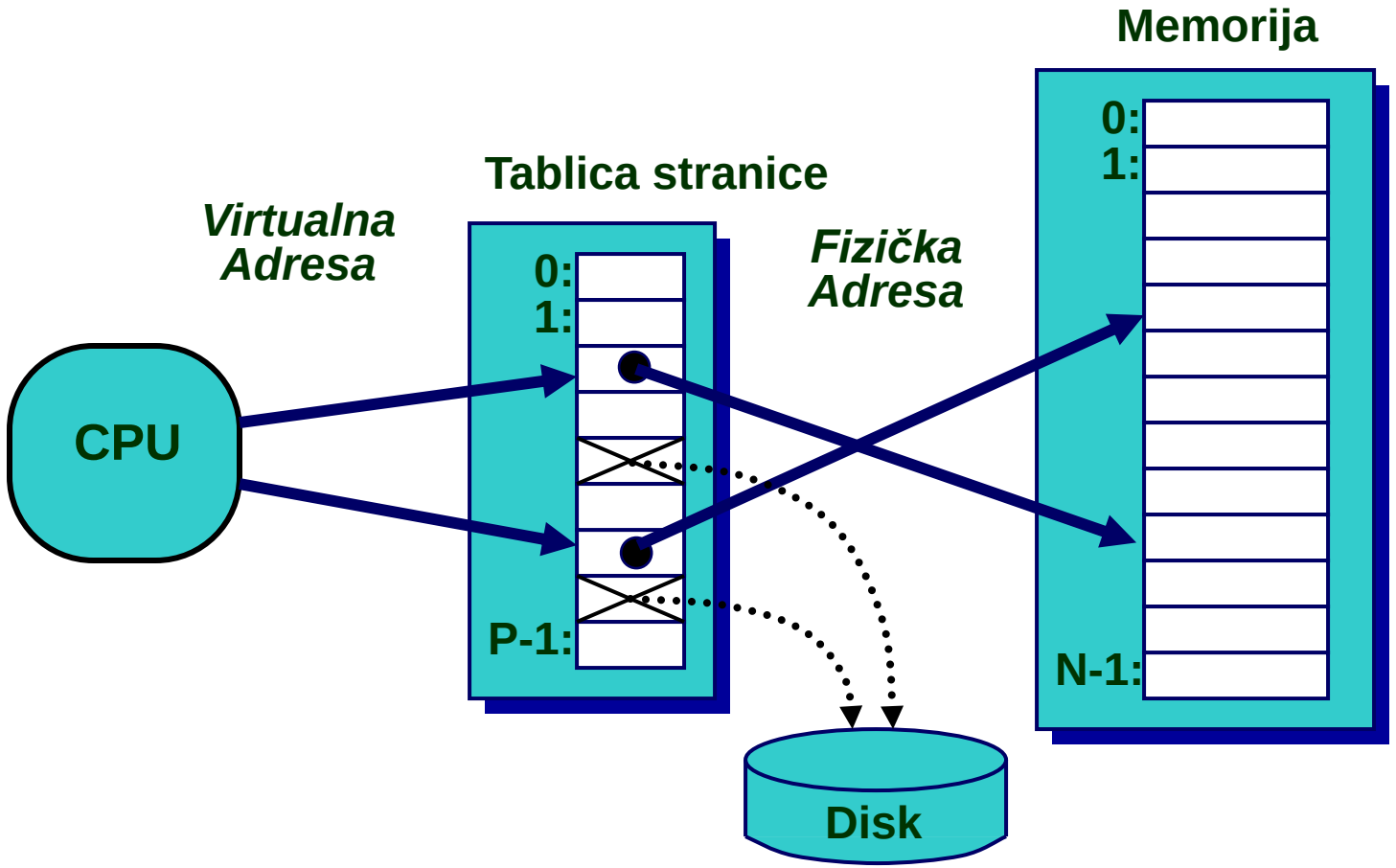
# Virtualna memorija: konceptualni prikaz

- **Iluzija velikog, zasebnog adresnog prostora po procesu**




Zahtijeva **neizravnost (zaobilazak) i mapiranje** između virtualnih i fizičkih prostora

# Sustav s virtualnom memorijom (temeljen na stranici)




- Adresna translacija: Hardver pretvara virtualne adrese u fizičke adrese putem tablice pretraživanja kojom upravlja OS (tablica stranice)

# Četiri problema u zaobilaženju i mapiranju

- Kada preslikati virtualnu adresu na fizičku adresu?
  - Kada program prvi put referencira virtualnu adresu
- Koja je granularnost mapiranja?
  - Byte? Kilo-byte? Mega-byte? …
  - Višestruke granularnosti?
- Gdje i kako pohraniti virtualno → fizičko mapiranje?
  - U bazu podataka Operativnog sustava? Hardware? Negdje drugdje?
- Što učiniti kada je fizički adresni prostor pun?
  - Izbaci malo vjerojatnu virtualnu adresu iz fizičke memorije

# Virtualne stranice, fizički okviri

- Virtualni adresni prostor podijeljen na stranice
- Fizički adresni prostor podijeljen u okvire

- Virtualna stranica mapirana je na:
  - Fizički okvir, ako je stranica u fizičkoj memoriji
  - U suprotno na mjesto na disku (sektor)

- Ako virtualna stranica kojoj se pristupa nije u memoriji, već na disku
  - Virtualni memorijski sustav unosi stranicu u fizički okvir i prilagođava mapiranje → to se naziva straničenje

- Tablica stranica je tablica koja pohranjuje mapiranje virtualnih stranica u fizičke okvire

# Fizička memorija kao predmemorija

- Drugim riječima...

- Fizička memorija je predmemorija za stranice pohranjene na disku
  - Zapravo, to je potpuno asocijativna predmemorija u modernim sustavima (virtualna stranica potencijalno se može mapirati u bilo koji fizički okvir)

- Slični problemi s predmemoriranjem postoje kao što smo ih ranije obradili:
  - Smještaj: gdje i kako smjestiti/pronaći stranicu u predmemoriji?
  - Zamjena: koju stranicu ukloniti da biste napravili mjesta u predmemoriji?
  - Granularnost upravljanja: velike, male, jednolične stranice?
  - Politika pisanja: Što ćemo s pisanjem? Pisati natrag?

# Hvala vam na pažnji!