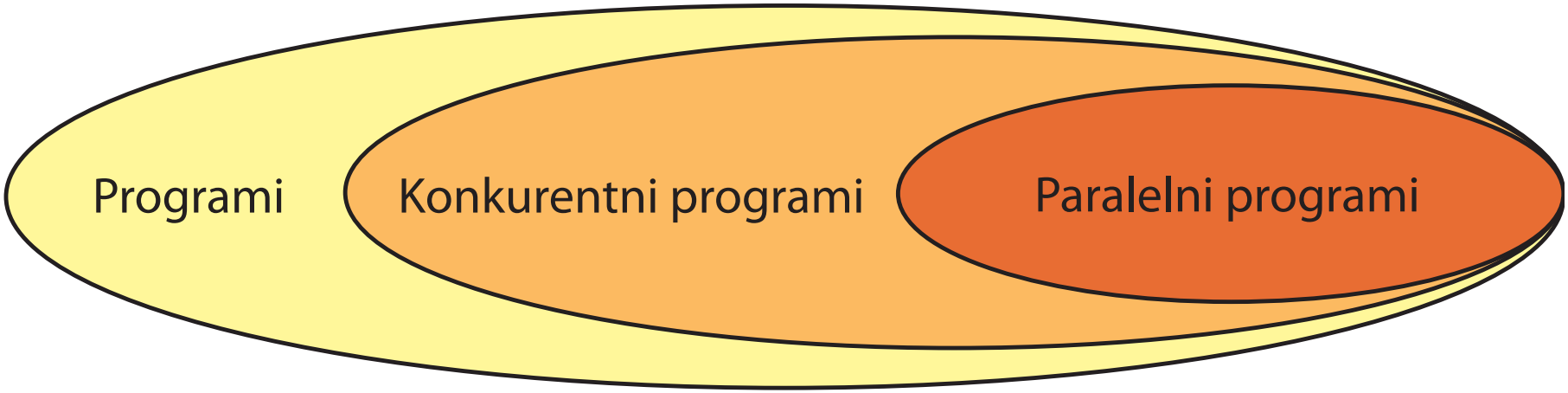
Uvod u paralelne i distribuirane sisteme

Konkurentna, paralelna i distribuirana obrada

- Odnos svih programa, konkurentnih programa i paralelnih programa:



Konkurentna, paralelna i distribuirana obrada

- **Konkurentno** – dešava se **tokom istog vremenskog intervala**
- **Paralelno** – dešava se **u isto vreme**. Paralelna obrada po definiciji zahteva veći broj procesora ili jezgara. U ovakvoj situaciji, više od jednog konkurentnog procesa može se **istovremeno** izvršavati. Kao i kod konkurentne obrade, ne mora biti komunikacije niti koordinacije između procesa
- **Distribuirano** – **više programa** izvršava se **konkurentno** i **međusobno komunicira** kako bi zajedno izvršili neko izračunavanje. Suština distribuirane obrade je u tome da se **rezultujuće izračunavanje distribuira između više procesa** koji **međusobno komuniciraju**

Tipovi programa i sistema

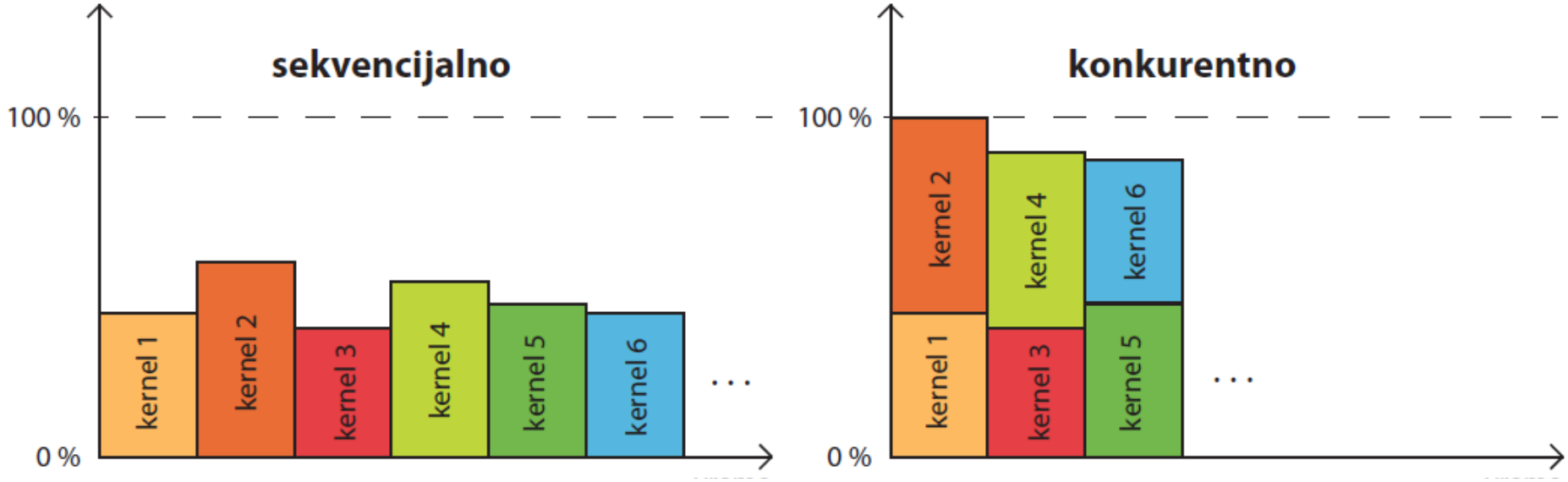
- **Obrada podataka**, tj. **računarski programi** koji implementiraju odgovarajuće algoritme, po svojoj prirodi mogu biti **sekvencijalni** ili **konkurentni**

- **Okruženje**, tj. **računarski sistem** na kome se neki program izvršava, zavisno od svoje arhitekture i dostupnih resursa može nuditi mogućnost **serijskog** ili **paralelnog izvršavanja** računarskih programa

- Da bi **računarski programi** mogli da iskoriste **mogućnosti paralelnih sistema** neophodno je da imaju **konkurentnu prirodu**

Sekvencijalni i konkurentni programi

- Izvršavanje sekvencijalnih i konkurentnih programa:



Paralelni i distribuirani algoritmi

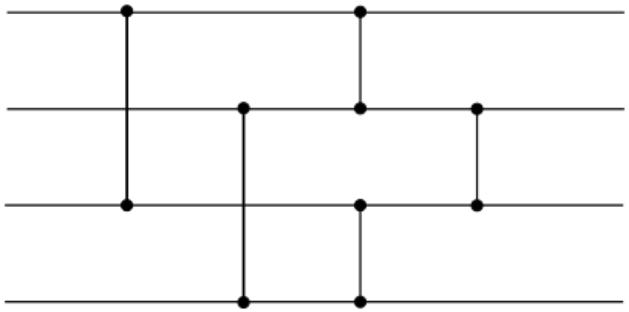
- Model **paralelnih algoritama** sa **deljenom memorijom**
  - Model je **PRAM** (engl. *parallel random access machine*)
- Model **paralelnih algoritama** sa **slanjem poruka**
  - Modeli su **Bulova logička kola** (engl. *Boolean circuits*) i **mreže za sortiranje** (engl. *sorting networks*)
- Model **distribuiranih algoritama** sa **slanjem poruka**
  - Model je **graf** u kome je **svaki čvor konačni automat** (engl. *finite-state machines*)

Paralelni i distribuirani algoritmi

- **Paralelni algoritmi sa deljenom memorijom**
  - Svi procesori imaju pristup deljenoj memoriji. Projektant algoritma bira program koji izvršava svaki od procesora
  - Teoretski model: **paralelna mašina sa slučajnim pristupom** (engl. *parallel random access machine* – **PRAM**)
  - PRAM je **apstraktna mašina sa deljenom memorijom**, analogna RAM, **zanemaruje sinhronizaciju i komunikaciju**, cena algoritma se iskazuje kroz dve mere: $O(vreme)$ i $O(vreme \times brojProcesora)$
  - Programi u deljenoj memoriji mogu da se **prošire na distribuirane** sisteme ako **operativni sistem enkapsulira komunikaciju** između čvorova i **virtuelno objedinjuje memoriju** pojedinačnih sistema
  - Model sa asinhronom deljenom memorijom je bliži realnim sistemima pošto uzima u obzir mašinske instrukcije kao što su uporedi i zameni (engl. *compare-and-swap* – CAS)

Paralelni i distribuirani algoritmi

- **Paralelni algoritmi sa slanjem poruka**
  - Projektant algoritma bira strukturu mreže, kao i programe koje izvršava svaki od računara
  - Koriste se modeli kao što su **Bulova logička kola** (engl. *Boolean circuits*) i **mreže za sortiranje** (engl. *sorting networks*). Bulova kola mogu da se posmatraju kao računarska mreža: svaki gejt odgovara računaru koji izvršava jako jednostavan program. Mreže za sortiranje takođe mogu da se posmatraju kao računarske mreže: svaki komparator je računar



Izvor: https://en.wikipedia.org/wiki/Sorting_network

Paralelni i distribuirani algoritmi

- **Distribuirani algoritmi sa slanjem poruka**
  - Projektant algoritma bira samo program. Svi računari izvršavaju isti program. Sistem mora da radi ispravno nezavisno od strukture mreže
  - Model **grafa** u kome se svaki od **čvorova** predstavlja **konačni automat**, **potezi** odgovaraju **komunikacionim kanalima**
  - Algoritmi specifični za distribuirano okruženje:
    - **Slika sistema** (engl. *snapshot*) – Chandy-Lamport algoritam beleži konzistentno globalno stanje svih procesa i poruka u asinhronom distribuiranom sistemu
    - **Konsenzus** (engl. *consensus*) – Bitcoin proof-of-work algoritam
    - **Samo-stabilizujući** (engl. *self-stabilizing*) – Dijksta Token Ring algoritam za međusobno isključivanje (Distribuirani sistem je samo-stabilizujući ako uvek završava u ispravnom stanju, nezavisno od stanja u kome je inicijalizovan i to ispravno stanje se dostiže u konačnom broju koraka)
  - Asinhrona priroda distribuiranih sistema:
    - **Sinhronizatori** se koriste kako bi se sinhroni algoritmi izvršavali u asinhronim sistemima
    - **Logički satovi** pružaju kauzalno uređenje događaja (šta se desilo pre čega)
    - **Algoritmi za sinhronizaciju satova** pružaju globalno konzistentne fizičke vremenske otiske (engl. *time stamps*)

Složenost izračunavanja algoritama

- Kod **paralelnih algoritama**, još jedan resurs koji se posmatra prilikom analize složenosti, pored vremena i prostora, je i **broj računara**
- Ako problem odlučivanja može da se reši u **polilogaritamskom vremenu** primenom **polinomnog broja procesora** kaže se da je problem u **klasi NC** ("*Nick's Class*" – Stephen Cook dao ime po Niku Pipengeru). U teoriji kompleksnosti, klasa **NC** je **skup problema odlučivanja** koji su odlučivi u **polilogaritamskom vremenu** na **paralelnom računaru** sa **polinomnim brojem procesora**. Problem je u klasi **NC** ako postoje konstante c i k takve da problem može da se reši u vremenu $\mathrm{O}(\log^c n)$ primenom $O(n^k)$ paralelnih procesora
- Klasa NC može se jednako dobro definisati primenom formalizama PRAM-a ili Bulovih kola
- Nerešen problem:

  P = NC?

Složenost izračunavanja algoritama

- Prilikom analize **distribuiranih algoritama**, **više pažnje** se obično posvećuje **operacijama komunikacije** nego izračunavanja
- Možda najprostiji model distribuiranog izračunavanja je **sinhroni sistem** u kome **svi čvorovi rade sa istim korakom** (engl. *lockstep*)
- Ovaj model je poznat kao **LOCAL**
  - Tokom svake runde komunikacije, svi čvorovi paralelno:
    1) Prime poslednju poruku od svojih suseda
    2) Izvrše odgovarajuće lokalno izračunavanje
    3) Pošalju novu poruku svojim susedima
  - Kod ovakvih sistema, **centralna mera složenosti** je **broj sinhronih komunikacionih rundi neophodnih za kompletiranje zadatka**

Hijerarhija apstrakcija

Nivo 5 — Nivo programskih jezika

Prevođenje (kompajler)

Nivo 4 — Nivo asemblerskog jezika

Prevođenje (asembler)

Nivo 3 — Nivo mašine operativnog sistema

Delimična interpretacija (operativni sistem)

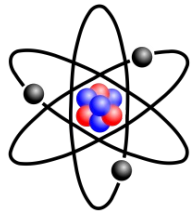
Nivo 2 — Nivo arhitekture skupa instrukcija (ISA)

Interpretacija (mikroprogram)

Nivo 1 — Nivo mikroarhitekture

Hardver

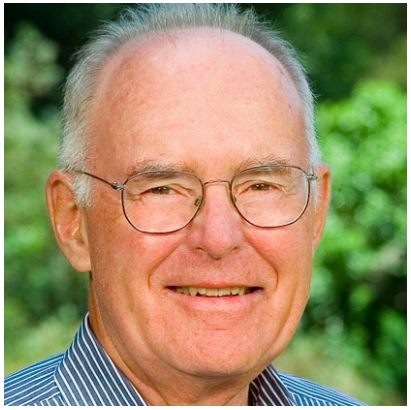
Nivo 0 — Nivo digitalne logike



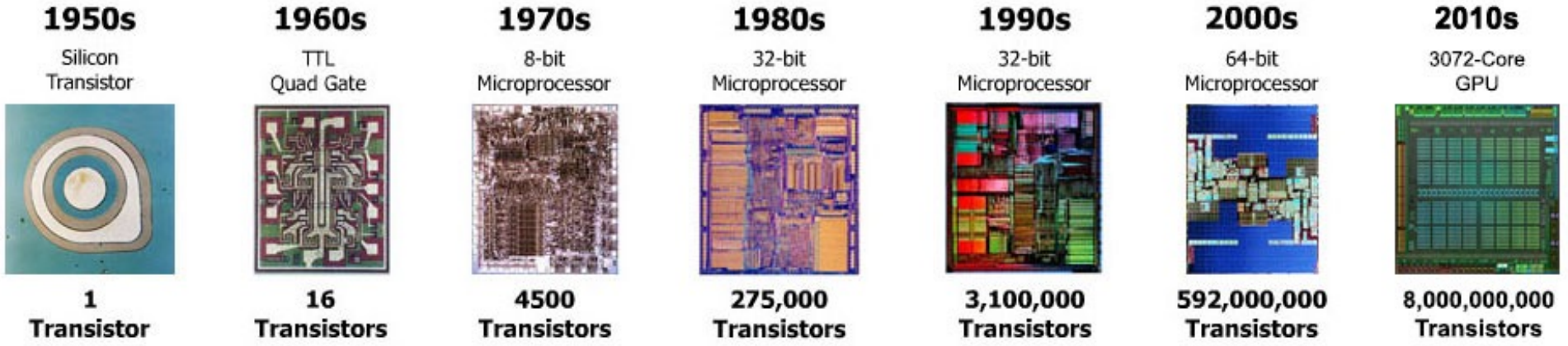
Paralelni sistemi

Porast performansi računara

- **Murov zakon** je zapažanje da se broj tranzistora u integrisanim duplira približno svake 2 godine.
- **Evolucija arhitekture elektronskih računara (1946-danas)**

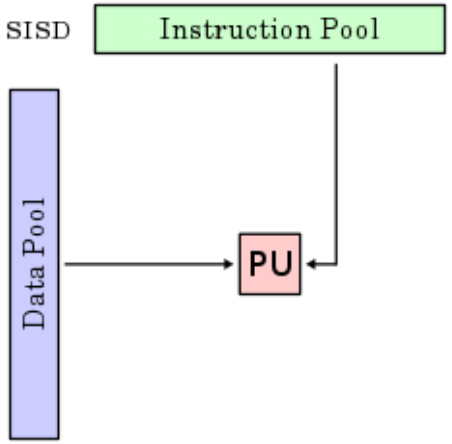


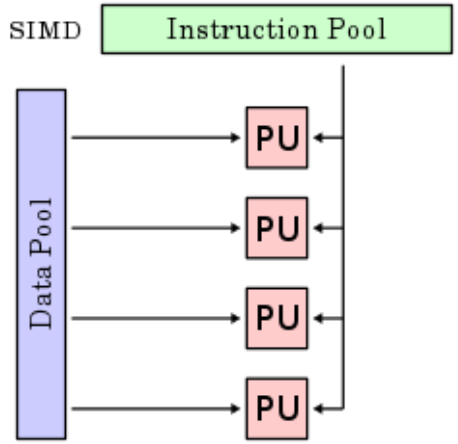
Gordon Moore
(1929–2023)



Flinova taksonomija

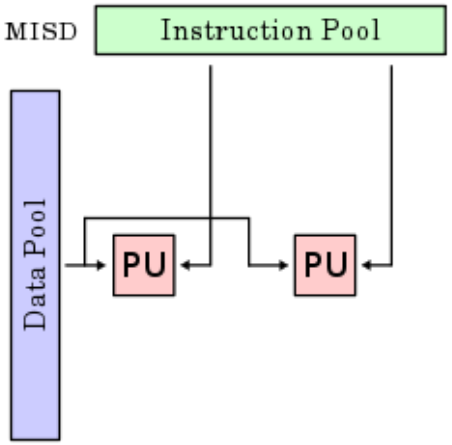
Single Instruction, Single Data (SISD)

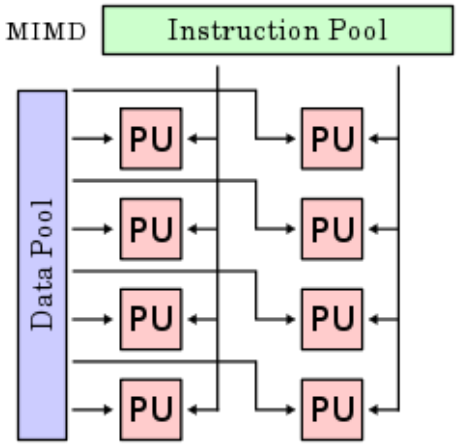




Single Instruction, Multiple Data (SIMD)

Multiple Instruction, Single Data (MISD)

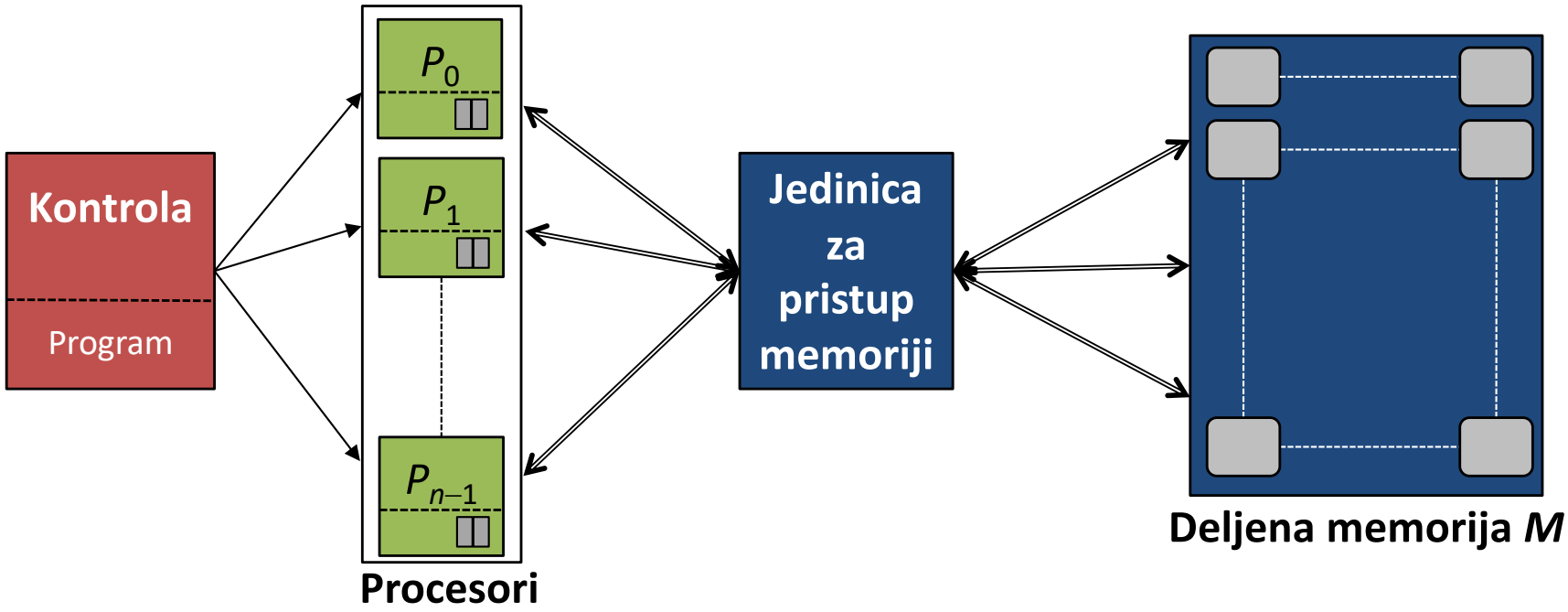




Multiple Instruction, Multiple Data (MIMD)

Izvor: https://en.wikipedia.org/wiki/Flynn%27s_taxonomy

Parallel Random Access Machine (PRAM)



- Model sa n procesora $P_0,\ldots,P_{n-1}$ povezanih sa globalnom deljenom memorijom M
- Svakoj memorijskoj lokaciji se može uniformno pristupiti od strane bilo kog procesora u konstantnom vremenu
- Komunikacija između procesora se realizuje čitanjem i pisanjem lokacija u M

Izvor: https://parallelprogrammingbook.org/

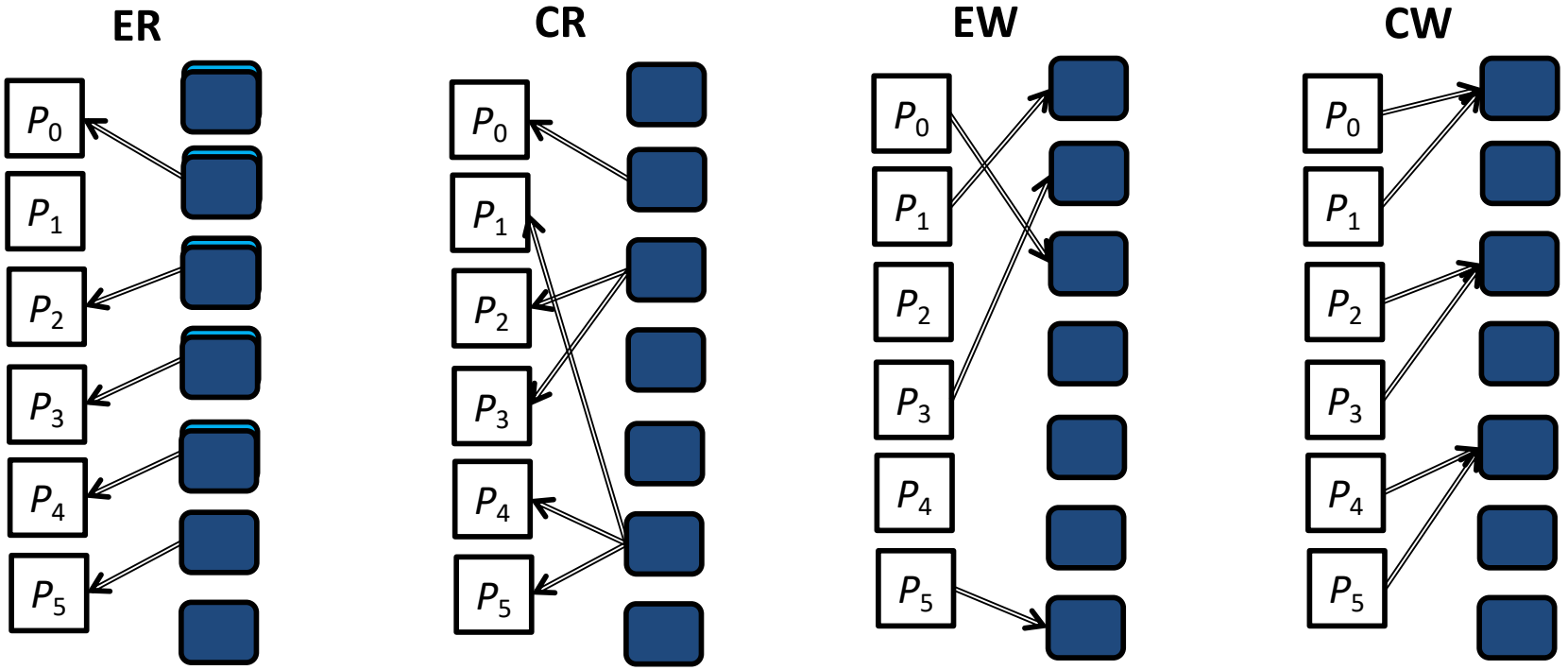
PRAM model

- Ima n identičnih procesora P_i, $i = 0,\ldots,n-1$ koji rade u istom koraku (engl. *lock-step*)
- U svakom koraku, svaki od procesora izvršava ciklus instrukcije u tri faze:

1. **Faza čitanja**: svaki od procesora može istovremeno da pročita jedan podatak iz (različite) lokacije deljene memorije i sačuva ga u svom lokalnom registru.
2. **Faza izračunavanja:** svaki od procesora može da izvrši neku od fundamentalnih operacija nad svojim lokalnim podacima i potom sačuva rezultat u registru.
3. **Faza upisa:** svaki od procesora može istovremeno da upiše jedan podatak u lokaciju deljene memorije, pri čemu **PRAM tip sa ekskluzivnim upisom** omogućava samo upis u različite lokacije, dok **PRAM tip sa konkurentnim upisom** omogućava da procesori upisuju na istu lokaciju (uslovi trke).

Tipovi PRAM

- **Četiri kombinacije**, tri najpopularnije varijante EREW, CREW, CRCW
- **Exclusive Read Exclusive Write (EREW):** Samo jedan procesor sme da čita i piše u jednu deljenu memorijsku lokaciju tokom bilo kog ciklusa
- **Concurrent Read Exclusive Write (CREW):** Više procesora može čitati podatke iz iste deljene memorijske lokacije, ali upis je zabranjen



Izvor: https://parallelprogrammingbook.org/

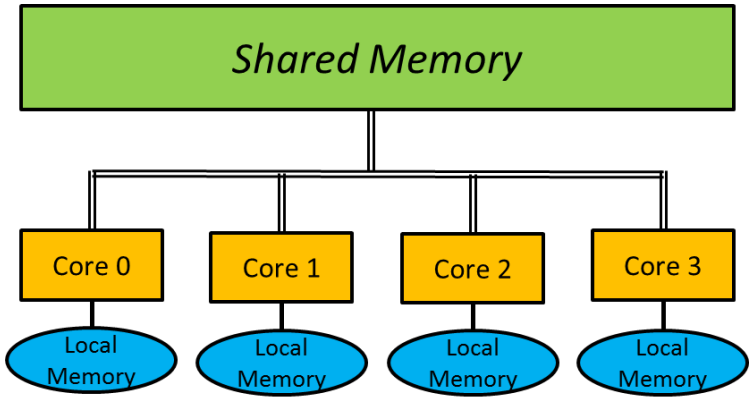
Tipovi PRAM

- **Concurrent Read Concurrent Write (CRCW):** dozvoljeno je istovremeno čitanje i upis u istu lokaciju u deljenoj memoriji
- Prilikom istovremenih upisa usled uslova trke neophodno je definisati koja će vrednost biti sačuvana:

1. **Prioritetni CW:** procesori imaju različiti prioritet, onaj sa najvišim uspeva da upiše vrednost
2. **Proizvoljni CW:** slučajno izabrani procesor upisuje vrednost
3. **Zajednički CW:** ako su sve vrednosti jednake, onda se vrši upis, u suprotnom lokacija ostaje nepromenjena
4. **Kombinovani CW:** sve vrednosti koje treba upisati se kombinuju u jedinstvenu vrednost primenom asocijativnih binarnih operacija (npr. sabiranje, množenje, maksimum/minimum, logičko I/ILI, …)

Izvor: https://parallelprogrammingbook.org/

Sistemi sa deljenom memorijom



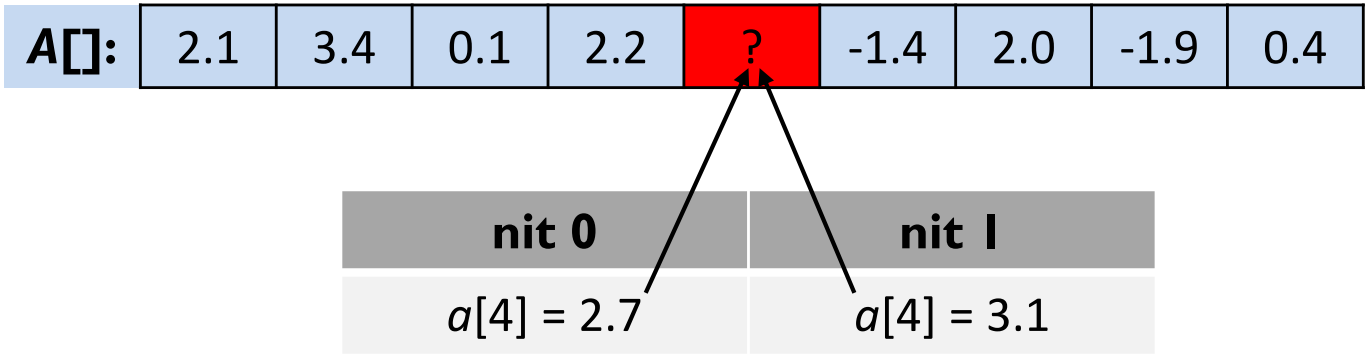
- **Sva jezgra imaju pristup deljenoj memoriji** preko **zajedničke magistrale** (engl. *shared memory*) – primer višejezgarni CPU
- Pored deljene memorije, svako od jezgara može imati svoju **manju lokalnu memoriju** (npr. keš) kako bi se **smanjio broj skupih memorijskih operacija** (tzv. *von-Neumann bottleneck*)
  - Savremeni višejezgarni CPU sistemi imaju podršku za ostavarivanje **koherentnosti keša** (engl. *cache cocherence*) – engl. *ccNUMA*: *cache coherent non-uniform access architectures*
- Najvažniji programski modeli: **višenitni C++11, OpenMP, CUDA**

Izvor: https://parallelprogrammingbook.org/

Projektovanje paralelnih programa

- **Podela** (engl. *partitioning*)**:**
  - Dati problem treba razložiti na delove primenom paralelizma na nivou podataka, zadataka ili upletenog paralelizma
- **Komunikacija** (engl. *communication*)**:**
  - Izabrana šema podele određuje tip i količinu neophodne komunikacije
- **Sinhronizacija** (engl. *synchronization*)**:**
  - Može se javiti potreba da se niti i/ili procesi sinhronizuju kako bi sarađivali na odgovarajući način
- **Balansiranje opterećenja** (engl. *load balancing*)**:**
  - Posao treba ravnomerno raspodeliti između niti ili procesa kako bi opterećenje sistema bilo balansirano i kako bi se minimizovao prazan hod
- Jedan od pristupa je i **Fosterova metodologija za projektovanje paralelnih algoritama**

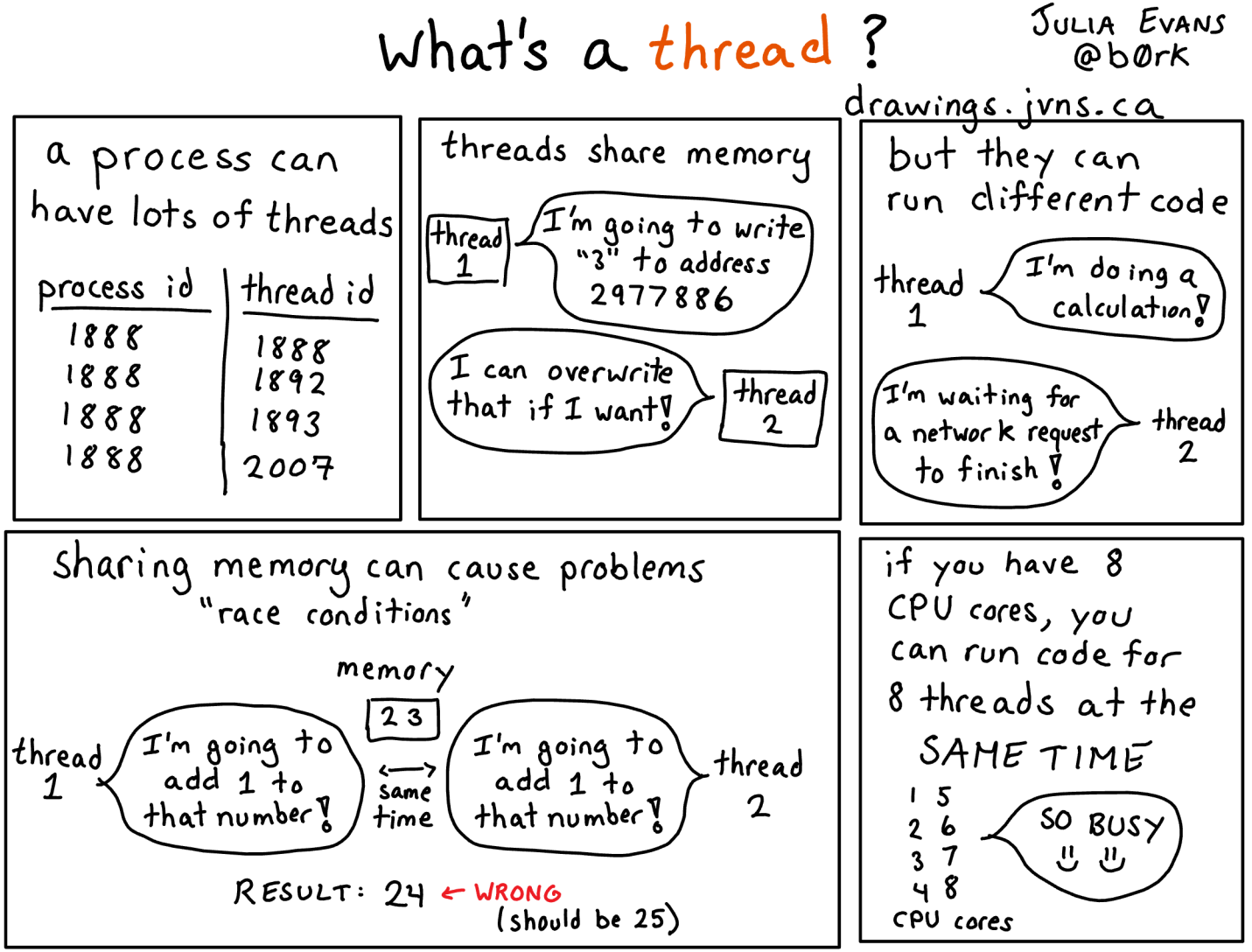
Sistemi sa deljenom memorijom



- Paralelizam se stvara kreiranjem niti koje se konkurentno izvršavaju u sistemu
- Razmena podataka se realizuje tako što niti čitaju sa i pišu u deljene memorijske lokacije
- **Uslov trke** (engl. *race condition*) se javlja kada dve niti istovremeno pristupe deljenoj promenljivoj
  - Odgovarajuće programske tehnike: muteksi, uslovne promenljive, atomične operacije
- Kreiranje niti lakše i brže od procesa – primer iz knjige *Parallel Programming*:
  - CreateProcess(): 12.76 ms
  - CreateThread(): 0.038 ms

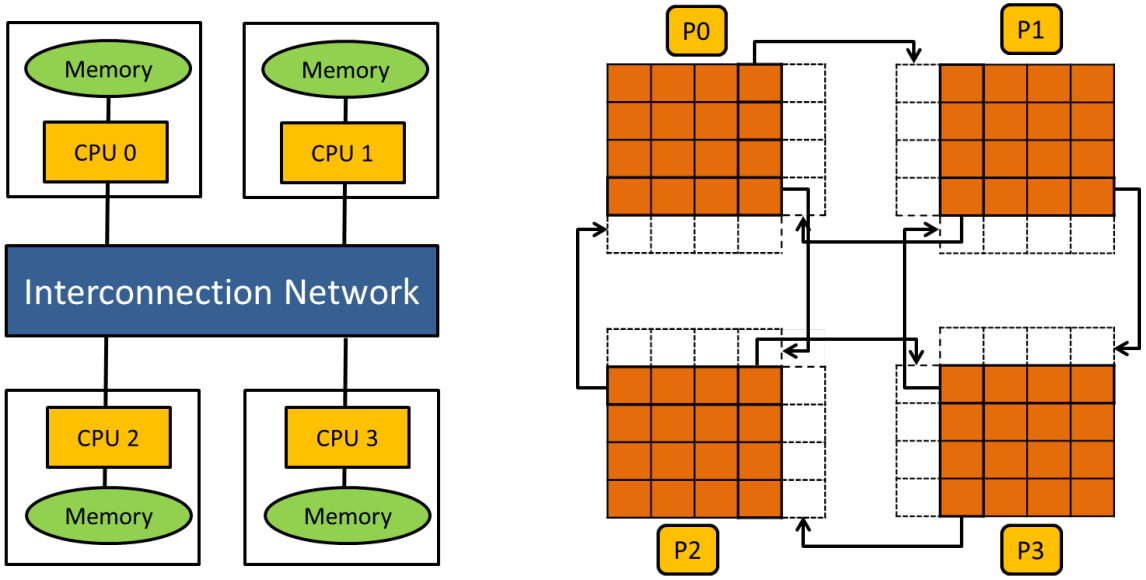
Izvor: https://parallelprogrammingbook.org/

Procesi i niti



Izvor: https://drawings.jvns.ca/drawings/threads.svg

Sistemi sa slanjem poruka

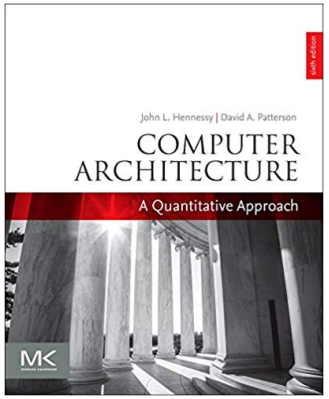


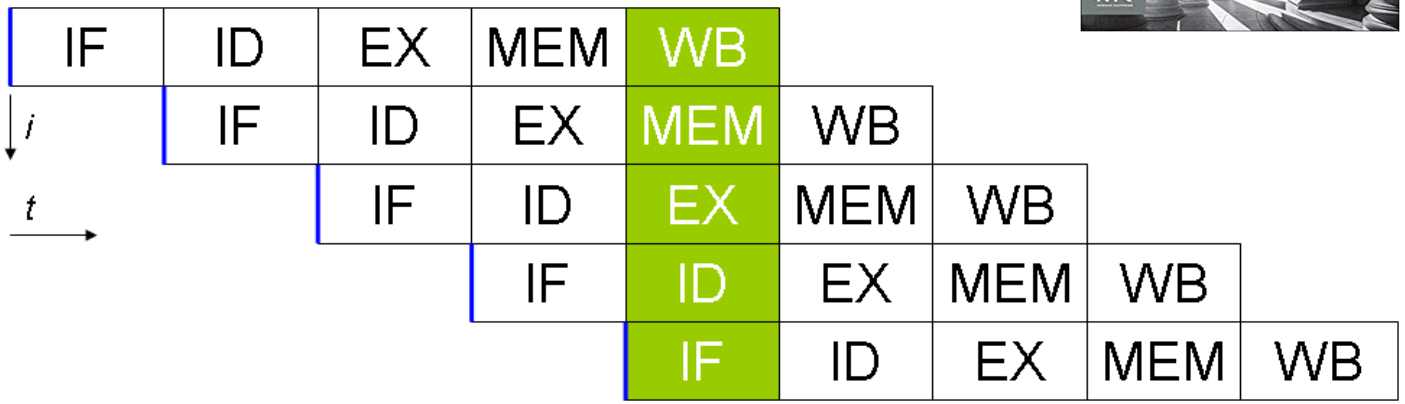
- **Svaki čvor** ima **sopstvenu privatnu memoriju**. Procesori eksplicitno komuniciraju slanjem poruka putem mreže
  - Najpopularniji standard: **MPI** (npr. MPI_Send, MPI_Recv, MPI_Bcast, MPI_Reduce)
- Primer su **računarski** (Beowulf) **klasteri**
  - Skup klasičnih računara povezanih sprežnom mrežom (engl. *interconnection network*) (npr. Ethernet ili Infiniband)

Izvor: https://parallelprogrammingbook.org/

Tipovi paralelizma

- **Paralelizam na nivou bitova** (engl. *bit-level parallelism*)
- **Paralelizam na nivou instrukcija** (engl. *instruction level parallelism* – ILP)
  - Protočna obrada (engl. *pipelining*)
  - Problem zavisnosti po podacima
  - RISC arhitektura
    - prvi primer MIPS – Hennessy, Patterson

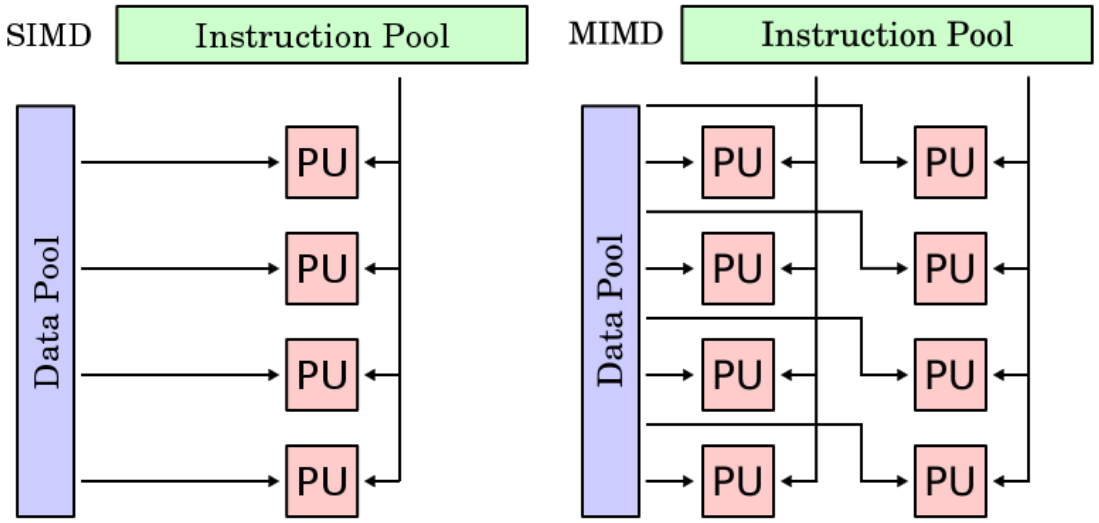




Izvor: https://en.wikipedia.org/wiki/Parallel_computing

Paralelizam na nivou podataka i zadataka

- **Paralelizam na nivou podataka** (engl. *data parallelism*) – tipično za SIMD arhitekture
- **Paralelizam na nivou zadataka** (engl. *task parallelism*)
- **Upleteni paralelizam** (engl. *braided parallelism*) – kombinovani paralelizam na nivoima podataka i zadataka

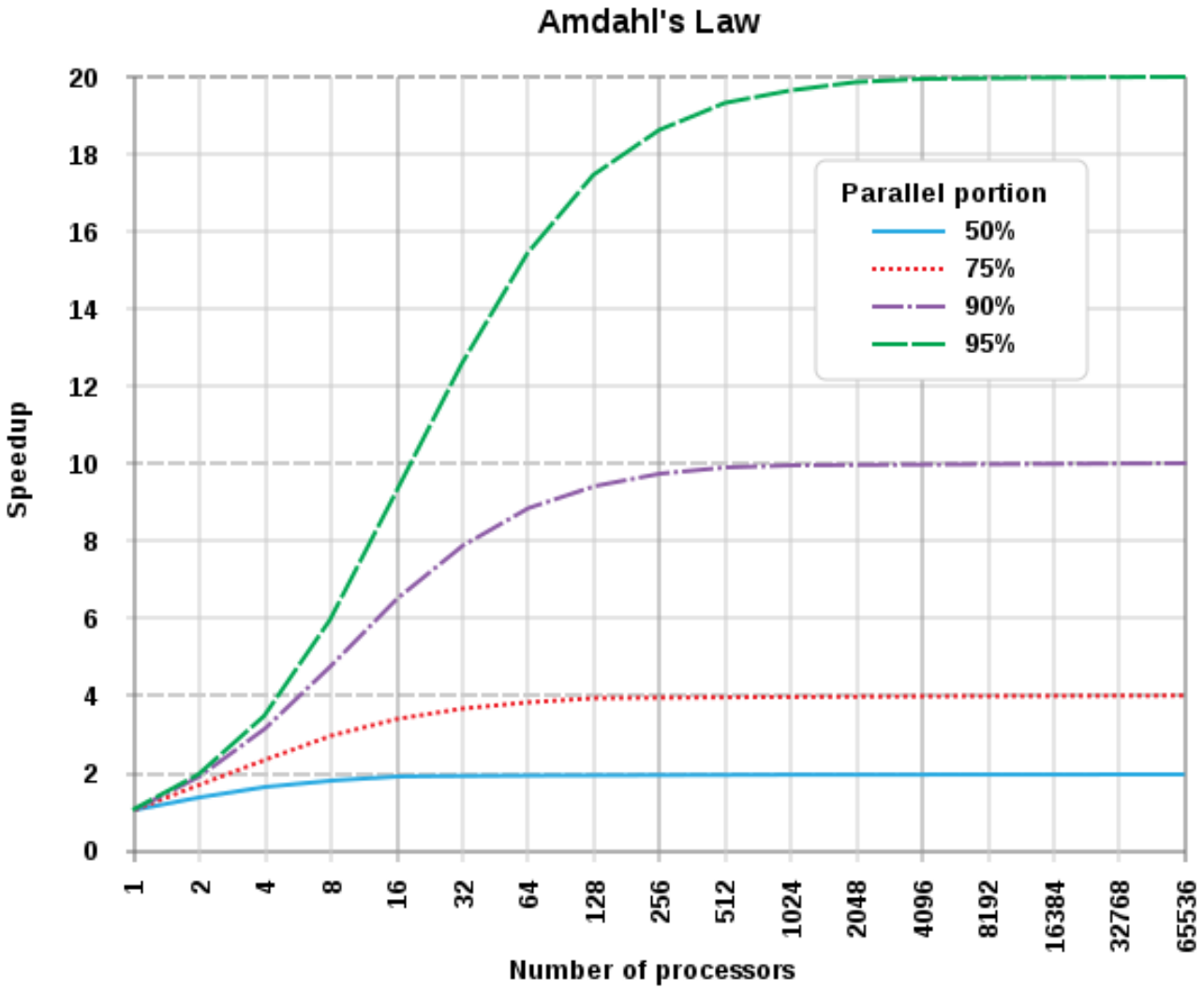


Amdalov zakon

- Dometi primene paralelizma:
  - **Amdahlov zakon** (G. M. Amdahl, 1967)
  - **Odnos serijskog i paralelnog dela obrade**
- **Paralelna obrada ima smisla** ako je:
  - **kratak sekvencijalni deo algoritma**
  - **visok stepen paralelizma**
- **Potencijalno ubrzanje programa primenom paralelizma dominatno je ograničeno delom programa koji ne može da se paralelizuje**
- Daje **gornju granicu** za ubrzanje koje se može postići paralelizacijom

Gene Amdahl
(1922–2015)

Amdalov zakon



Izvor: https://en.wikipedia.org/wiki/Parallel_computing

Gustafson-Barsisov zakon

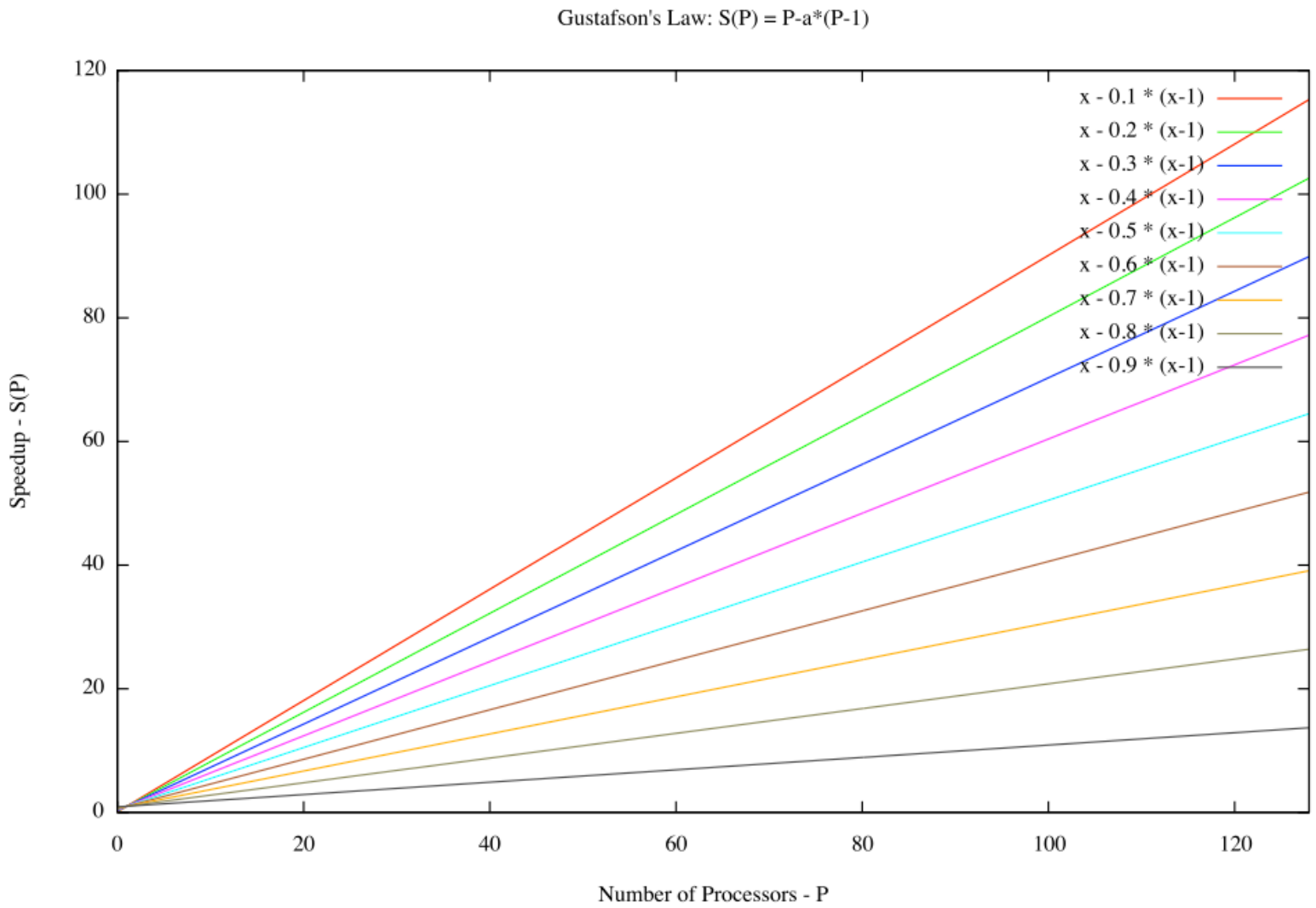
- Gustafson i Barsis, za razliku od Amdala, posmatraju **porast moći obrade** koji dovodi do toga da se **više podataka kompletnije analizira**, a **vreme obrade je fiksirano** (http://www.johngustafson.net/pubs/pub13/amdahl.htm)

- **Gustafson-Barsisov zakon** je specijalni slučaj kojim se može predvideti teoretski moguće ubrzanje primenom više procesora kada **deo programa koji se može paralelizovati skalira linearno sa veličinom problema**, dok **serijski deo ostaje konstantan**

- Ako su dostupni brži ili veći resursi, onda se i **veće instance problema** mogu rešiti **za isto vreme**

- Ograničenja postavljen usled nužno sekvencijalnog dela programa, mogu se delom prevazići povećanjem ukupne količine izračunavanja

Gustafson-Barsisov zakon

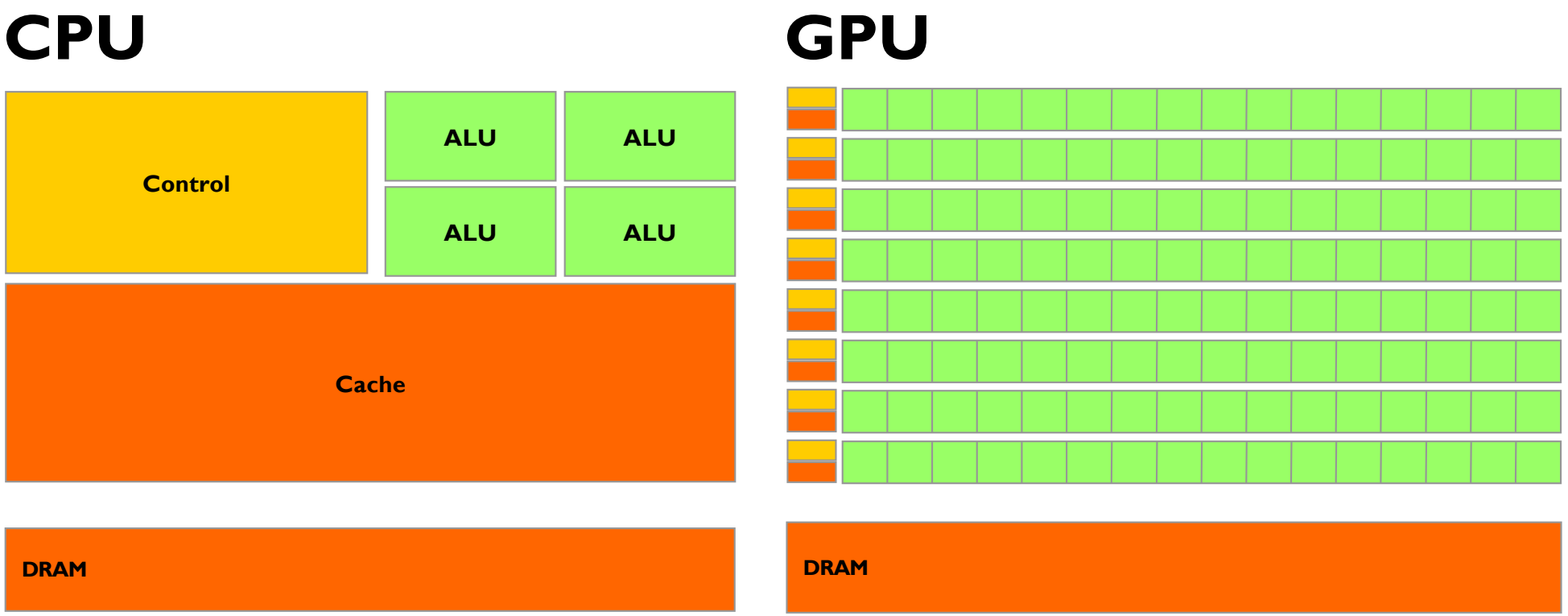


Izvor: https://en.wikipedia.org/wiki/Gustafson%27s_law

Heterogeni paralelni računarski sistemi

- Prelazak na **heterogene računarske procesore** je velika promena u **arhitekturama računara** i **računarstvu uopšte**
- **Homogeni računarski sistemi** – jedan ili više procesora iste arhitekture koriste se za izvršavanje programa
- **Heterogeni računarski sistemi** – skup procesora zasnovanih na različitih arhitekturama (CPU, GPU, FPGA, DSP) koristi se za izvršavanje programa
- **Svaki procesor** namenjen je za **različite zadatke** i s toga je njegova arhitektura zasnovana na **različitoj projektnoj filozofiji**
- Izvršavanje zadataka na arhitekturama koje su im najbolje prilagođene vodi do **unapređenih performansi** u smislu vremena i energije**,** ali zahteva **nove tehnike u programiranju** (primer – GPGPU programiranje)

Paralelna obrada na CPU i GPU



von Neumann, višejezgarna

SIMD, mnogojezgarna

Izvor: https://commons.wikimedia.org/wiki/File:Cpu-gpu.svg