Veebiteenuste ja hajussüsteemide arendus

Loeng 2 - Hajussüsteemide omadused. Lõimed

Pelle Jakovits

jakovits@ut.ee

Veebruar 2025

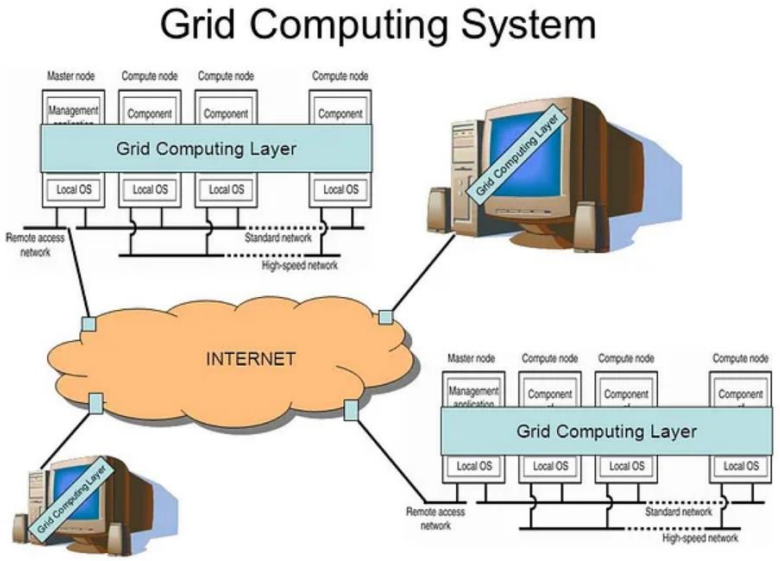
Sisukord

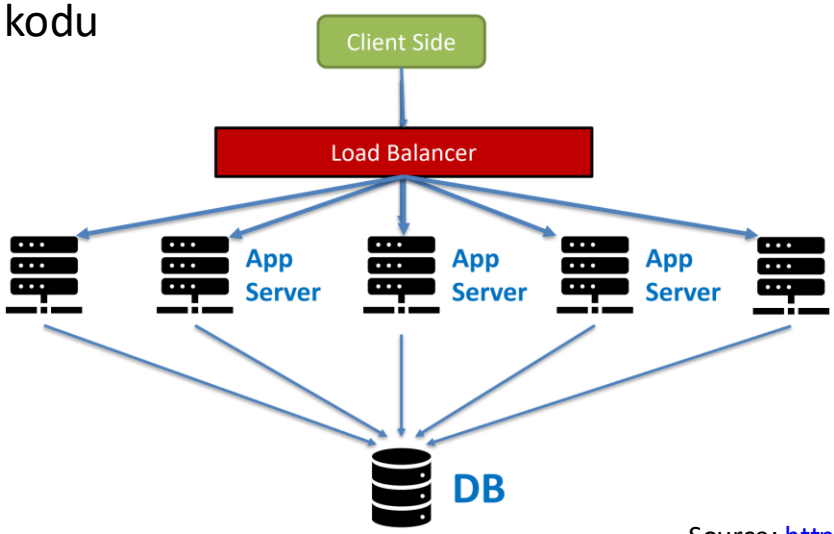
- Hajussüsteemide tüübid ja näited
- Hajussüsteemide omadused
  - Ressursside jagamine
  - Avatus
  - Paralleelsus
  - Skaleeritavus
  - Tõrkekindlus
  - Transparentsus
- Mitmelõimelised protsessid

Allikas: https://medium.com/@fiberoptics/cloud-computing-vs-grid-computing-fd87b632c8dd

Hajussüsteemide tüübid

- Kõrgjõudlusega hajusandmetöötlus
  - Arvutusklastrid, Grid'id
  - **Pilveplatvormid**
  - P2P võrgud
- Hajusad infosüsteemid
  - **Skaleeritavad infosüsteemid**
- Hajusad sardsüsteemid, sensorvõrgud
  - CAN (Car Area Network)
  - BAN (Body Area Network)
  - Tark kodu





Source: https://www.planet.com.tw/en/solutions/info/home-automation

Miks on vaja hajussüsteeme?

- Arvutusressursside koondamine ja effektiivsus
- Andmete ja arvutusvõimsuse laiali jagamine
- Detsentraliseeritus
- Ressursside kaugkasutus
- Tõrketaluvus
- Jõudlus

Loeng 2

HAJUSSÜSTEEMIDE OMADUSED

Definitsioon

Hajussüsteem on **autonoomsete arvutuselementide kogum**, mis paistab kasutajatele **ühe sidusa süsteemina**

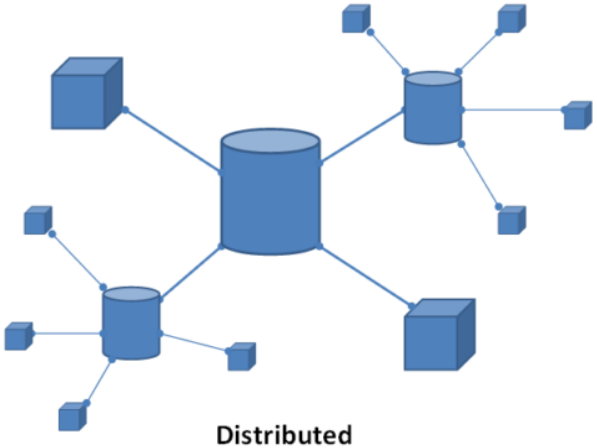
- **Autonoomsed arvutuselemendid**, sõlmed (nodes)
  - Serverid, arvutid
  - Protsessid, teenused
  - Nutitelefonid, sensorid
- **Üks sidus süsteem**
  - Sõlmed peavad omavahel koostööd tegema
  - Kasutaja ei pruugi aru saada, et tegemist on autonoomsete sõlmede koguga

Autonoomsusete sõlmede kogum

- Iga arvutuselement, sõlm (*ik: Node*) on **iseseisev üksus**
  - Sellel on oma aja mõõtmine.
    - Ei ole ühte globaaset kella
  - See toob sünkroniseerimise ja kordineerimise probleemid
- **Arvutussõlmede kogum**
  - Kuidas hallata liikmete nimekirja ja liitumist
  - Kuidas teda, et suheldakse õige (autentse) osapoolega

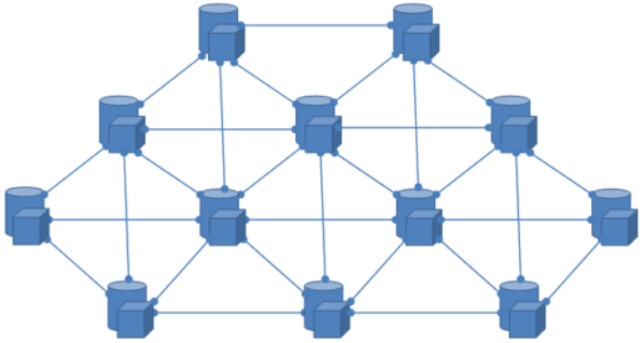
Centralized

one node does everything



Distributed

nodes distribute work to sub-nodes



Decentralized

nodes are only connected to peers

Allikas: https://www.delphitools.info/DWSH/

Ühtne ja sidus hajussüsteem

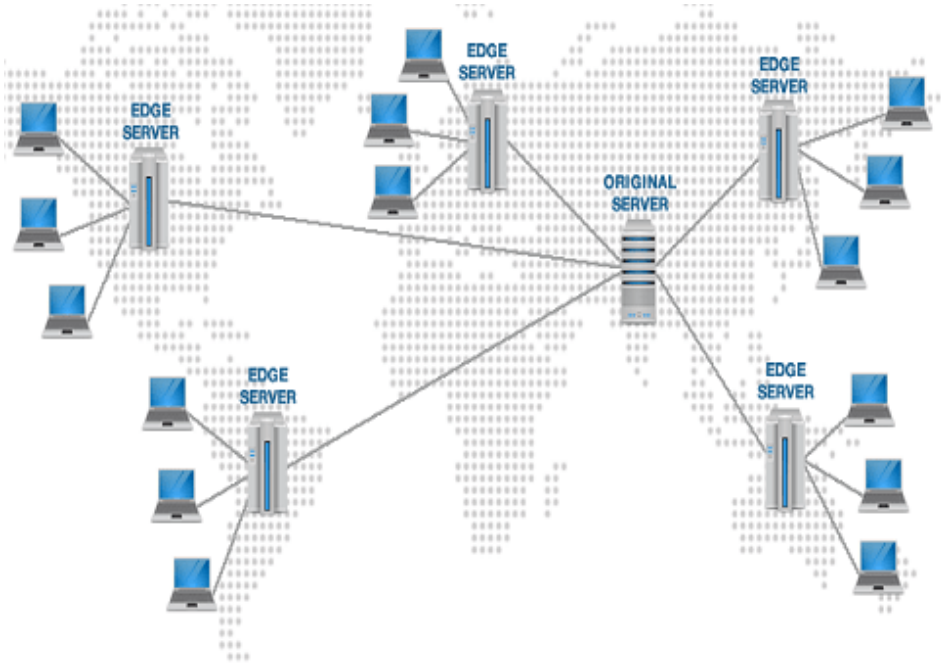
- Sõlmede kogu tervikuna toimib samamoodi, olenemata sellest, kus, millal ja kuidas toimub kasutaja ja süsteemi vaheline suhtlus.
- Hajutamise läbipaistvus
  - Lõppkasutaja ei tohiks näha, kus täpselt arvutused toimuvad
  - Rakenduse jaoks ei tohiks olla oluline, kus andmeid täpselt salvestatakse
  - On peidetud, kas andmetest on mitu koopiat (replitseerimine), või mitte
- Probleemid:
  - On paratamatu, et igal ajal võib mõni osa hajutatud süsteemist "kokku jooksta"
  - Osaliste rikete ja nende taastumise peitmine on sageli väga keeruline ja võib väga tihti olla võimatu

Hajussüsteemide põhiomadused

1. Ressursside jagamine
2. Läbipaistvus, transparentsus
3. Avatus
4. Tõrkekindlus
5. Skaleeritavus, paralleelsus

Ressursside jagamine

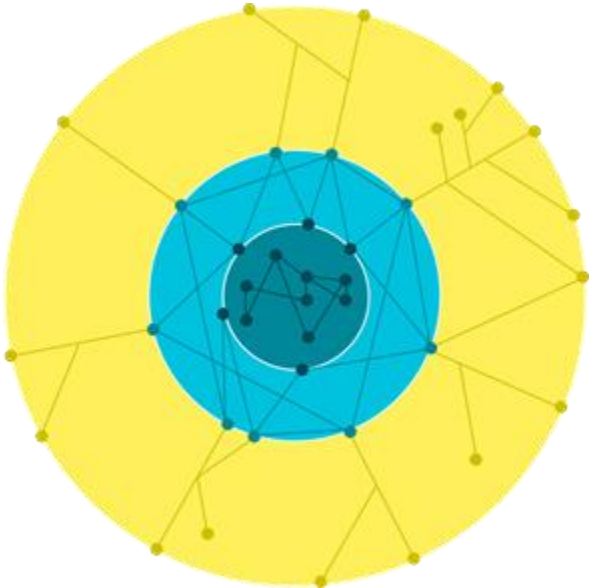
- Mis on ressursid?
  - Riistvara
    - Arvutusvõimsus:
    - paralleelarvutused
    - Salvestusruum:
      - Pilvepõhine failide haldus
  - Andmed
    - Andmete jagamine
    - Sisu levitamise võrgud (Content delivery networks)
  - Teenused
    - Jagatud sisselogimise teenus
    - Rakenduse väline tõlke API



https://www.belugacdn.com/how-content-delivery-networks-work/

Google Edge Network

- Globaalne andmete vahendamise võrk
- Koosneb kolmest kihist:
  a. Andmekeskused
     - USA, EU, Aasia
  b. Kohalikud ühedus-keskused
  c. Google vahemälu serverid



Läbipaistvus (transparentsus)

- Kasutaja peab nägema süsteemi kui tervikut, tema eest "peidetakse" hajussüsteemi komponendid
  - **Juurdepääsu transparentsus** — peidab andmete salvestus formaatide ja juurdepääsumeetodite erinevused
  - **Asukoha transparentsus** — peidab ressursi/objekti asukoha
  - **Migratsiooni transparentsus** — peidab selle, et ressurss võib kolida sele kasutamise ajal
  - **Replikatsiooni transparentsus** — peidab selle, et ressurst on mitu koopiat, ja sellele pääseb ligi mitmest asukohast
  - **Tõrke transparentsus** — peidab lihtsamad tõrked kasutaja eest
  - **Samaaegsuse transparentsus** — peidab selle, et sama ressurssi kasutavad sama aegselt mitu osapoolt
  - **Skaleerimise transparentsus** — peidab skaleeruvusprobleemid kasutaja eest

Täielik läbipaistvus

- Täieliku läbipaistvust võib olla võimatu saavutada
  - Ressursside ligipääsu latentsust on raske või ebamõistlik peita
  - Tõrgete täielik peitmine võib olla võimatu
  - Täielik läbipaitvus nõuab lisa jõudlust ja ressursse
    - Replitseeritud andmete 100% sünkroniseerimine võtab aega
    - Tõrgetega toimetulek võtab aega
- Hajussüteemi info avaldamine kasutajale võib olla kasulika
  - Asukohapõhiste teenuste kasutamine ja leidmine
  - Kasutaja saab paremini aru, miks süsteem on aeglane, kui näeb vigasid

Avatus

- Avatud süsteemi saab kokku panna heterogeensest riistvarast ja tarkvarast
- Riistvaraline avatus
  - Lisaseadmete lisamine, komponentide sobivus
  - Riistvara programmeeritus
- Tarkvaraline avatus
  - Lisaomaduste tekitamine, teenuste lisamine
  - Avalikud programmeerimisliidesed (API) ja protokollid
  - Suhtlus ja andmete vahetus teiste teenuste ja süsteemidega
- Süsteemid peaksid:
  - Vastama paika pandud liidestele (**interfaces**, APIs)
  - Olema võimelised koos töötama (**interoperability**)
  - Toetama rakenduste liigutamist teistele platvormidele (**portability**)
  - Olema kergesti laiendatavad (**extendable**)

Paralleelsus

- Tsentraliseeritud süsteem N protsessoriga annab kuni N kordse jõudluse kasvu, võrreldes ühe protsessoriga süsteemiga
- Hajussüsteem M ühe protsessoriga arvutiga annab meile kuni M kordse jõudluse kasvu, kui iga protsess töötab eri arvutis
- Kasutajad käivitavad programme või suhtlevad programmidega samaaegselt
- Paljud serveri protsessid töötavad samaaegselt, igaüks reageerides klientprotsessidelt saabuvatele erinevatele päringutele
- Paralleelsusest tulenevad omad probleemid (peamiselt sünkroniseerimine)

Skaleeritavus

- Hajussüsteem peab olema koostatud selliselt, et süsteemi arvutusressursse oleks võimalik vajadusel suuredada/vähendada
  - Kui kasutajate arv suureneb
  - Kui vaja salvestada rohkem andmeid
- Süsteemi kasvamine ei tohiks kaasa tuua tarkvara muutmise vajadust
- Süsteem peab olema efektiivselt kasutatav ka komponentide suure arvu juures
- Skaleeritavust mõõdetakse selle järgi, missugune on süsteemi jõudlus keskmiselt ühe protsessori kohta
- Süsteemi disainimisel ei tohiks eeldada ühegi ressursi piiratust

Skaleerimist takistavad tegurid

- Tsentraalsed teenused — üks server kõigi jaoks
- Tsentraalsed andmed — üks keskne andmekogu
- Tsentraliseeritud algoritmid — otsuste tegemine täieliku informatsiooni järgi

- Vajalik:
  - Üski masin ei sõltu täielikust infost kogu hajusa süsteemi kohta
  - Masinad teevad otsuseid ainult kohaliku informatsiooni alusel
  - Ühe masina tõrge ei blokeeri algoritmi tööd
  - Ei eeldata globaalse kella olemasolu

Skaleerimismeetodeid

- **Vertikaalne skaleermine**
  - Eraldame eraldi sõlmed süsteemi erinevate teenuste jaoks
  - Muudame sõlme võimsust (CPU tuumade arv, mälu maht)
- **Horisontaalne skaleerimine**
  - eraldame palju sõlmi ühe kihi teenuste realiseerimiseks
  - Suurendame sõlmede arvu
- Replitseerimine, puhverdamine (caching)
  - Mitme andmeobjekti koopia salvestamine erinevates serverites
  - Mitme koopia kasutamine viib sünkroniseerimisprobleemideni
    - **Global synchronization** vs **Eventual consistency**
    - Kui suudame taluda ebakõlasid, väheneb vajadus globaalse järele sünkroonimise järele

Skaleerimismeetodeid

- Asünkroomne suhtlus geograafilise skaleeruvuse saa vutamiseks
  - Tegeleme teiste tegevustega kuni vastus saabub kohale
  - Kõik rakendused seda ei toeta
- Arvutuste osaline läbi viimine kliendi seadmes
  - nt. JavaScript veebirakendustes
- Järjekordade kasutamine (e.g. MQTT, RabbitMQ)

Tõrkekindlus

- Tõrkekindluse tagamine
  - Riistvara dubleeritus: dubleeritud riistvarakomponentide kasutamine
  - Tarkvaraline taastumine: programmid peavad olema vigade korral suutelised taastama stabiilset seisundit
- Süsteemi käideldavuse (availability) mõiste: töövalmis oleku aeg jagatud kogu ajaga

Valed eeldused keskkonna kohta

- Peter Deutsch [Sun Microsystems]: Ei tohi eeldada, et:
  - Võrk on töökindel
  - Võrk on turvaline
  - Võrk on homogeenne
  - Topoloogia ei muutu
  - Latentsus puudub
  - Võrgu läbilaskevõime on lõpmatu
  - Andmete transportimise hind on null
  - Kogu süsteemi administreerib üksainus administraator

Loeng 2

LÕIMED JA SÜNKRONISEERIMINE

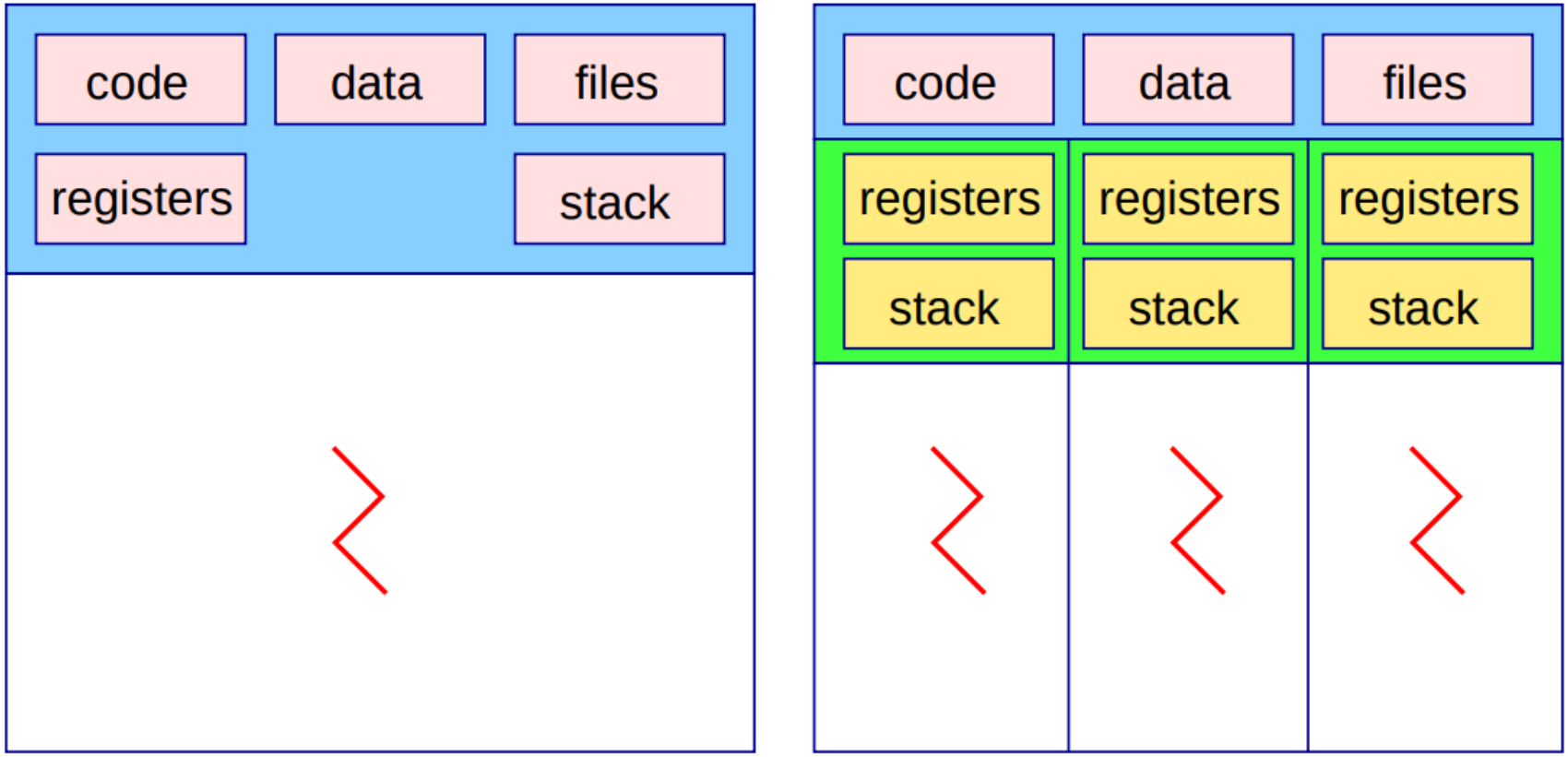
Lõimed

- Protsessi ja lõime mõiste
- Lõimede mudelid
- Sünkroniseerimine
- Lõimede realisatsioon

Protsessi mõiste

- Protsess on täitmisel olev programm
  - Protsesi olek (registrid)
    - Näiteks käsuloendur (program counter, instruction pointer)
  - Mälu sisu
    - Koodi sektsioon
    - Andmete sektsioon
    - Magasin (Pinu, Stack)
- Samast programmist võib olla käivitatud mitu protsessi
- Protsesside paralleelne täitmine
  - Ühes protsessori tuumas jookseb korraga üks protsess
  - Kontekstivahetus protsesside vahel võtab aega

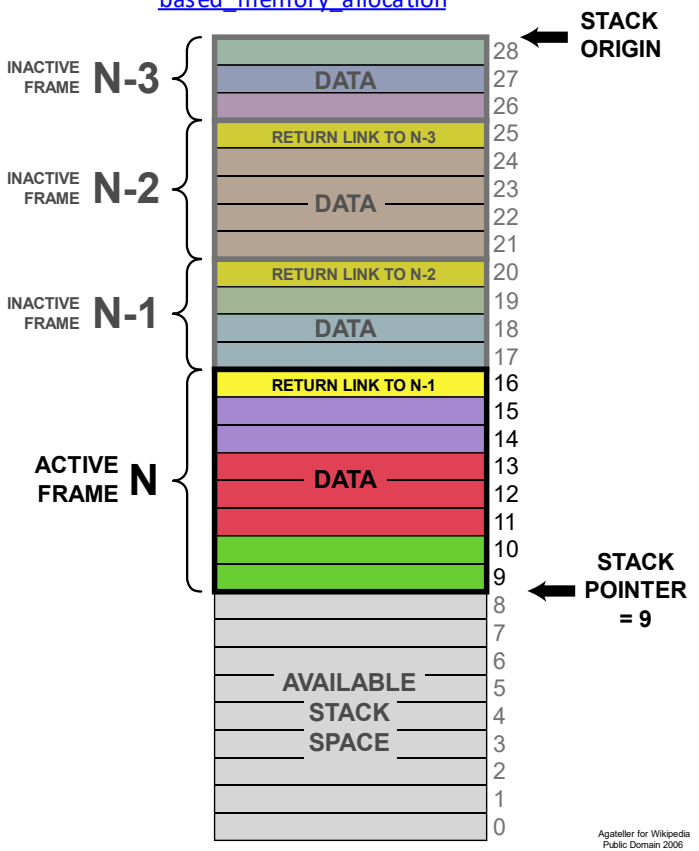
Ühe- ja mitmelõimelised protsessid



Ühe- ja mitmelõimelised protsessid

- Igal lõimel on oma täitmisjärg ja pinu (stack)
- Iga lõime täidetakse järjestikuliselt
- I/O ressursid (failipidemed, . . .) on jagatud
- Mälu on kõigi lõimede vahel jagatud (kood, andmed)
  - Lõimede vahel mälukaitset pole, lõimed teevad koostööd
  - Laiendusena on tihti võimalik kasutada lõimedes isiklikku mäluala **(TLS – thread-local storage)**

Allikas: https://en.wikipedia.org/wiki/Stack-based_memory_allocation

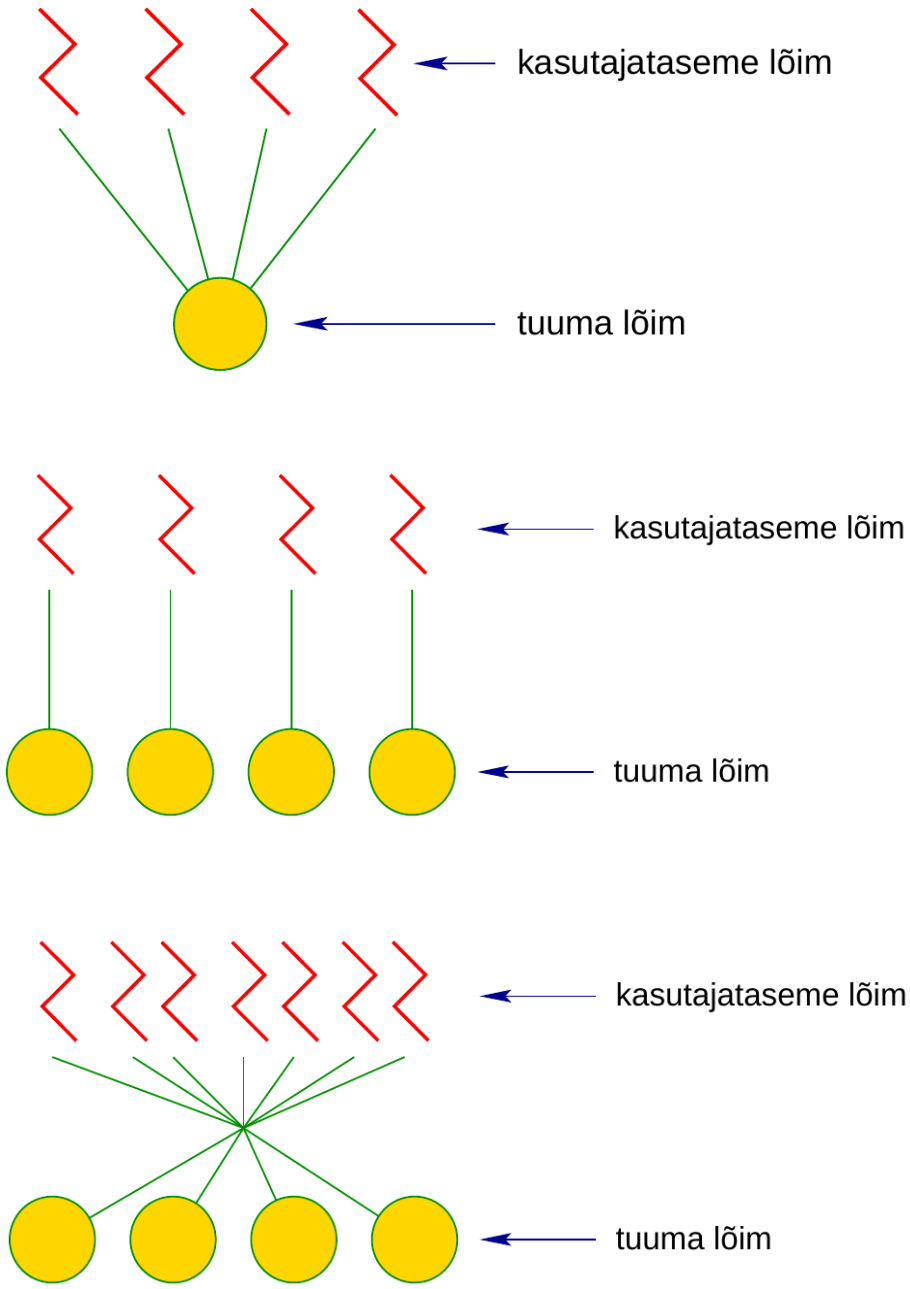


Lõimede mudelid

Mitu ühele
- Haldus kiire
- Korraga saab töötada kuni üks lõim
- Blokeeruva süsteemifunktsiooni kasutamisel blokeerub kogu protsess

Üks ühele
- Haldus kulukam
- Saab korraga teha mitut blokeeruvat operatsiooni
- Ei maksa liiga palju lõimi teha

Mitu mitmele
- Kombineerib eelmiste head küljed
- Keerukam



Lõimede head ja vead

- Abiks:
  - **Reageerimiskiirus** — parema interaktiivsuse saamiseks võib kasutajaliidese ja muu töö eraldi lõimedesse panna
  - **Paralleelprogrammeerimise** mugavamaks tegemine
  - **Ressursside jagamine** — aadressipiirkond on programmi mitme täitja poolt kasutatav
  - **Kokkuhoid** — hoiame kokku jagatud ressursside, uute protsesside loomise ning kontekstivahetuste pealt
  - **Mitme protsessori ära kasutamine** ühe protsessi poolt
- Aga:
  - Lõimede vahel lülitumine on endiselt **lisakulu**
  - Lõimede **silumine (debugging) on keeruline**
  - **Kaob isolatsioon**

Lõimede kasutamise mudelid

- Kolm levinumat mudelit:
  - **Jagaja-töötajad** — üks lõim võtab vastu päringuid ja edastab need töötlemiseks teistele lõimedele
  - **Meeskonnamudel** — kõik lõimed on võrdsed ja töötlevad päringuid vastavalt nende saabumisele
  - **Konveiermudel** — lõim viib läbi mingi osa operatsioonist ja annab päringu seejärel üle järgmisele lõimele
- Lõimekogu (thread pool)
  - Optimeering lõimede pideva tekitamise ja hävitamise vastu
  - Saab piirata aktiivsete lõimede arvu

Sünkroniseerimine

- Paralleelne juurdepääs mälule võib ühiseid andmeid sodida
  - counter++ ja counter-- tuleb täita atomaarselt
- Atomaarsed operatsioonid ja andmetüübid
  - Aitavad ainult väga lihtsatel juhtudel
- Võidujooks (race condition) — situatsioon, kus andmete kasutamisel mitme lõime/protsessi poolt korraga sõltub tulemus ajalistest teguritest
  - Kriitilised sektsioonid
  - Vastastikune välistamine kriitilistes sektsioonides
  - Realiseerimiseks on vaja mingeid sünkroniseerimisprimitiive

Sünkroniseerimisviisid

- **Mutex** – lukk – ainult üks lõim saab lukust "edasi"
- **Semafor** – lubab kuni x lõime lukust "edasi"
- **Lugemis-kirjutamislukk (rwlock)** - Mitu võivad lugeda, ainult üks saab kirjutada
- **Monitor** - Ainult üks lõim saab mingi jagatud objekti (meetodit) kasutada
  - **Condition variable** – ootame Monitori taga, kuni mingi tingimus on täidetud. Näiteks puhvris on piisaval arvul elemente.
- **Kriitiline regioon keele tasemel** – ainult üks lõim saab kriitilises regioonis asuvat koodi korraga täita
- **Lõimede lõpetamine (thread join)** - Ootame, kuni lõim töö lõpetab
- **Ilma lukkudeta läbi ajamine** — lukkude taga oodatud aeg on raisatud aeg
  - Atomaarsed operatsioonid
  - Read–modify–write
    - test-and-set, fetch-and-add, compare-and-swap

Selle nädala praktikum

- Lõimede programmeerimine Pythonis
  - Lõimede loomine
  - Lõimede grupi haldus (Thread Pool)
  - Sünkroniseerimine

Järgmine loeng

- Andmetevahetus hajussüsteemides:
  - Teadete edastus ja voogsuhtlus

Allikad ja viited

- Van Steen, Maarten, Tanenbaum, Andrew. Distributed Systems: Principles and Paradigms (Third edition). Published by Maarten van Steen, 2023.
  - Tasuta versioon: https://www.distributed-systems.net/
- Hajussüsteemide aine materjalid, Meelis Roos, Tartu Ülikool