LUT
University

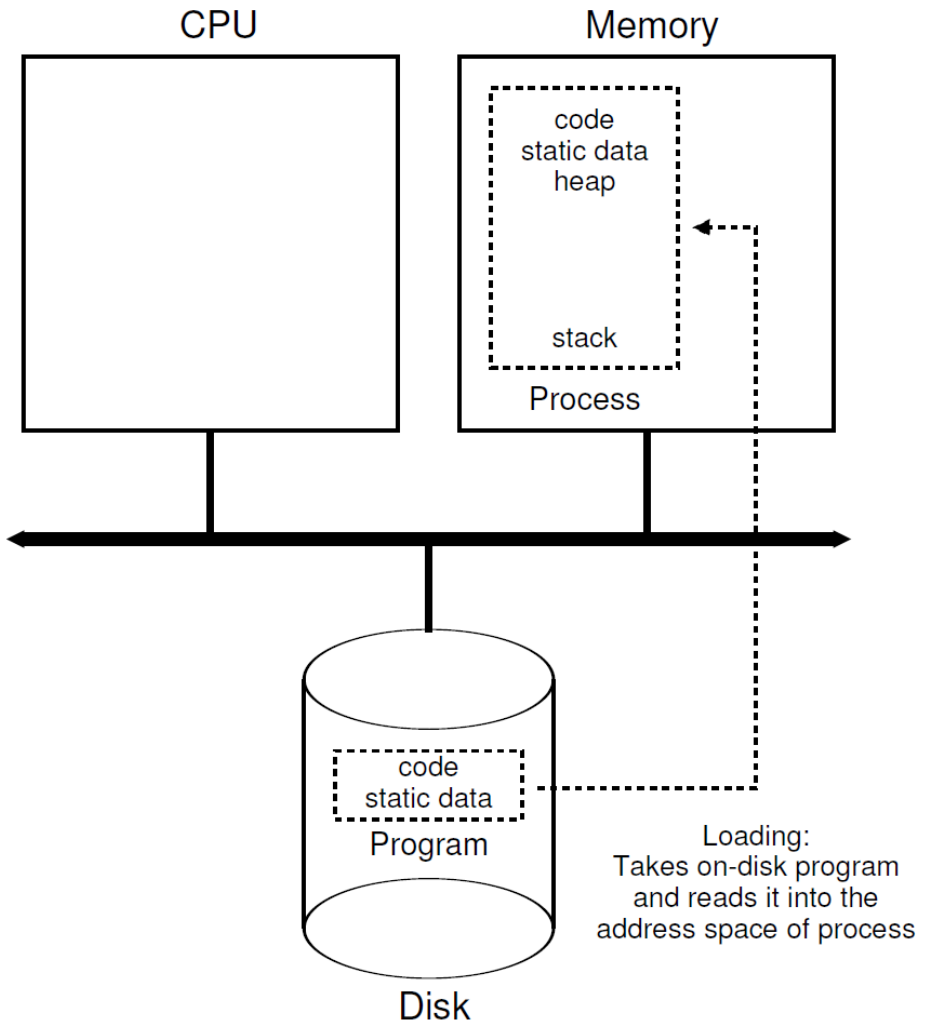
CT30A3370 - KÄYTTÖJÄRJESTELMÄT JA SYSTEEMIOHJELMOINTI 6 OP

Jussi Kasurinen (etu.suku@lut.fi)

Osa kalvoista Timo Hynnisen 2016 materiaaleista

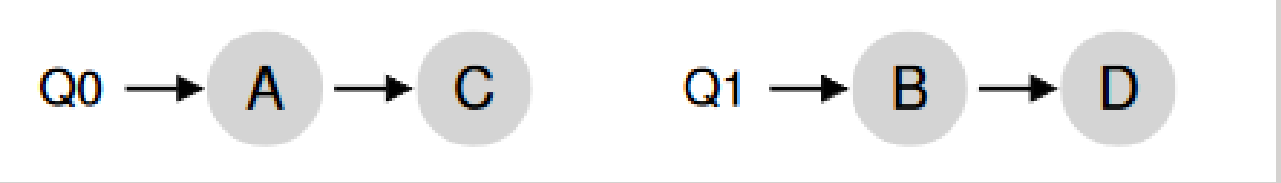
EDELLISILTÄ LUENNOILTA:

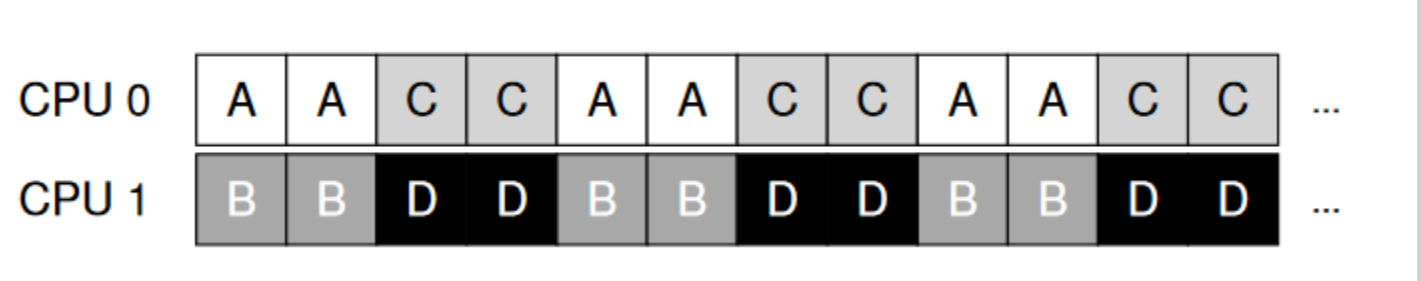
OHJELMA VS. PROSESSI



VUORONNUS MONELLE PROSESSORILLE?

- Jos suorittimia on useita, päättää vuorontaja siitä, mihin suoritinjonoon mikäkin prosessi menee.





ASSEMBLY-KUVAUS

- Symbolinen kuvaus konekielestä jolla rauta oikeasti laskee.

```
                .file   "cpu.c"
                .text
                .section        .rodata
.LC0:
                .string "common.h"
.LC1:
                .string "rc == 0"
                .text
                .globl  GetTime
                .type   GetTime, @function
GetTime:
.LFB5:
                .cfi_startproc
                pushq   %rbp
                .cfi_def_cfa_offset 16
                .cfi_offset 6, -16
                movq    %rsp, %rbp
                .cfi_def_cfa_register 6
                subq    $48, %rsp
                movq    %fs:40, %rax
                movq    %rax, -8(%rbp)
                xorl    %eax, %eax
                leaq    -32(%rbp), %rax
                movl    $0, %esi
                movq    %rax, %rdi
                call    gettimeofday@PLT
                movl    %eax, -36(%rbp)
                cmpl    $0, -36(%rbp)
                je      .L2
                leaq
__PRETTY_FUNCTION__.2816(%rip), %rcx
                movl    $10, %edx
                leaq    .LC0(%rip), %rsi
                leaq    .LC1(%rip), %rdi
                call    __assert_fail@PLT
.L2:
                movq    -32(%rbp), %rax
                cvtsi2sdq       %rax, %xmm1
```

MUISTINHALLINTA JA MUISTIAVARUUS

CT30A3370 - Käyttöjärjestelmät ja systeemiohjelmointi

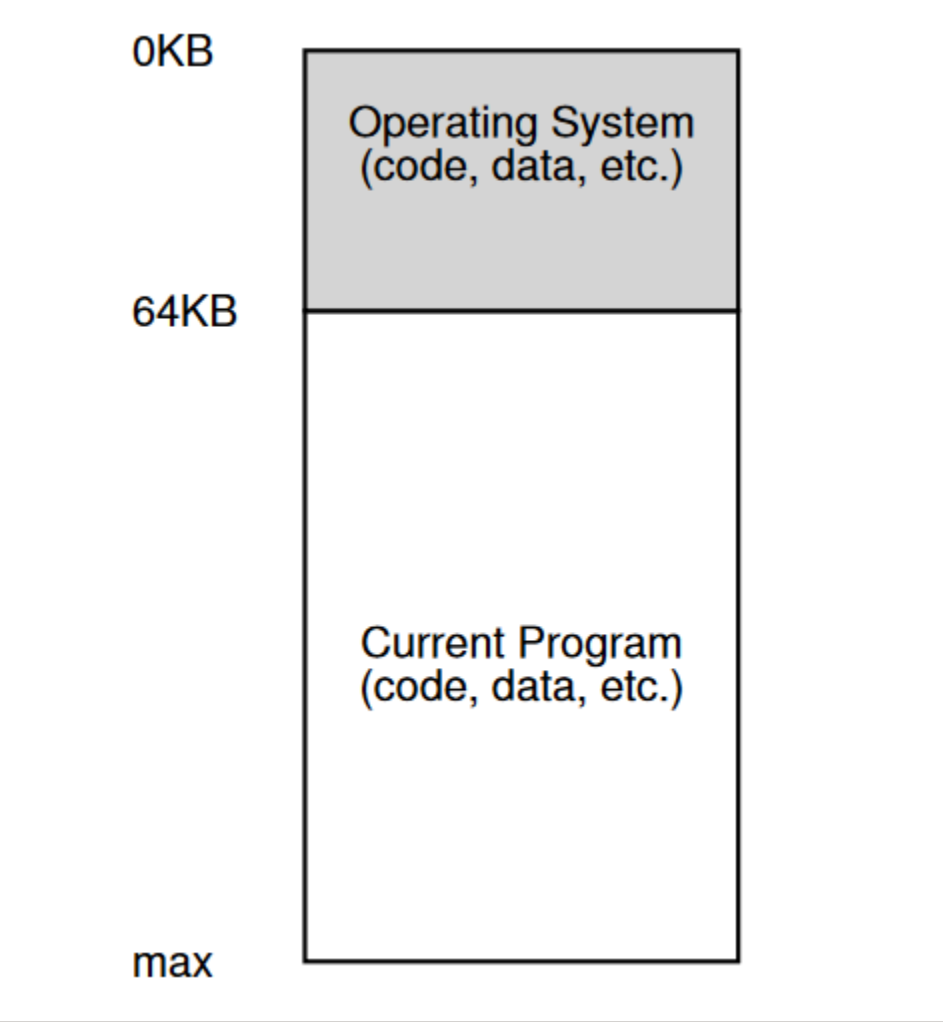
MUISTINHALLINTA

- ...Eli tässä vaiheessa meillä on
  - Joku idea siitä miten tietokone suorittaa asioita.
  - Joku idea siitä miten kone valitsee sen operaation mitä seuraavaksi tehdään.
  - Tiedämme millaisia käskyjä suorittimella tapahtuu.
- Seuraavaksi varmaankin pitäisi miettiä miten prosessit saadaan muistiin, ja vielä siten, että ne kiltisti leikkii keskenään.

YLEISESTI

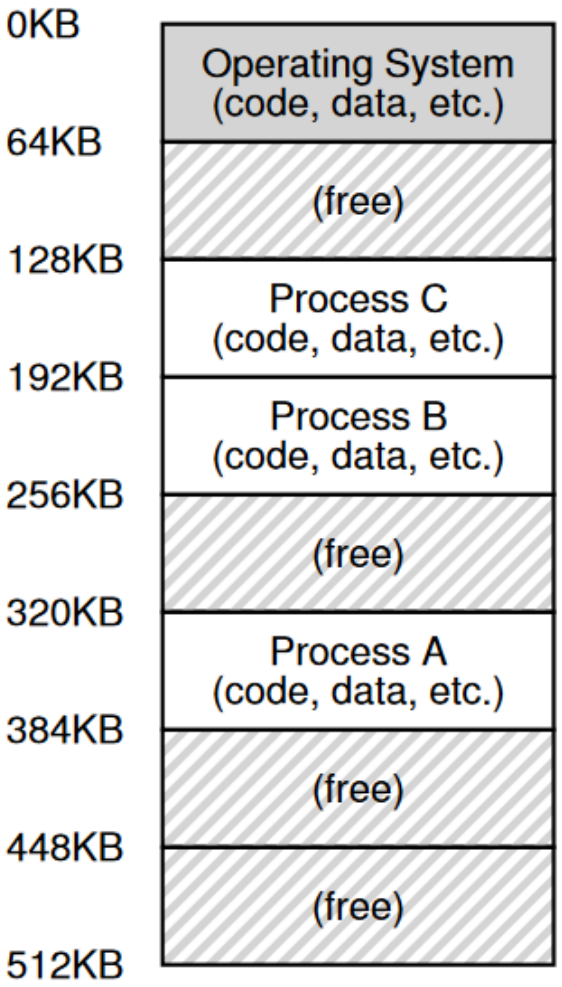
- Koneen keskusmuisti on periaatteessa ainoastaan jono tavuja paikasta 0 eteenpäin niin pitkälle kuin tilaa löytyy.
  - Eli jos koneessa on 8 gigatavua keskusmuistia, viimeinen paikka olisi 8589934591.
  - Kurssin ja kirjan esimerkeissä puhutaan naurettavan pienistä määristä, mutta käytännössä idea on edelleen sama, mutta paikkoja/tilaa on merkittävästi enemmän.
- Miten tämä saadaan jaettua järkevästi siten, että
  - A) Prosessit saa omat alueensa?
  - B) Käyttöjärjestelmä saa oman alueensa?
- Oletetaan selvyyden vuoksi, että meillä on yksi suoritin ja yksi muistialue käytössä.

PERUSTEET MUISTISTA



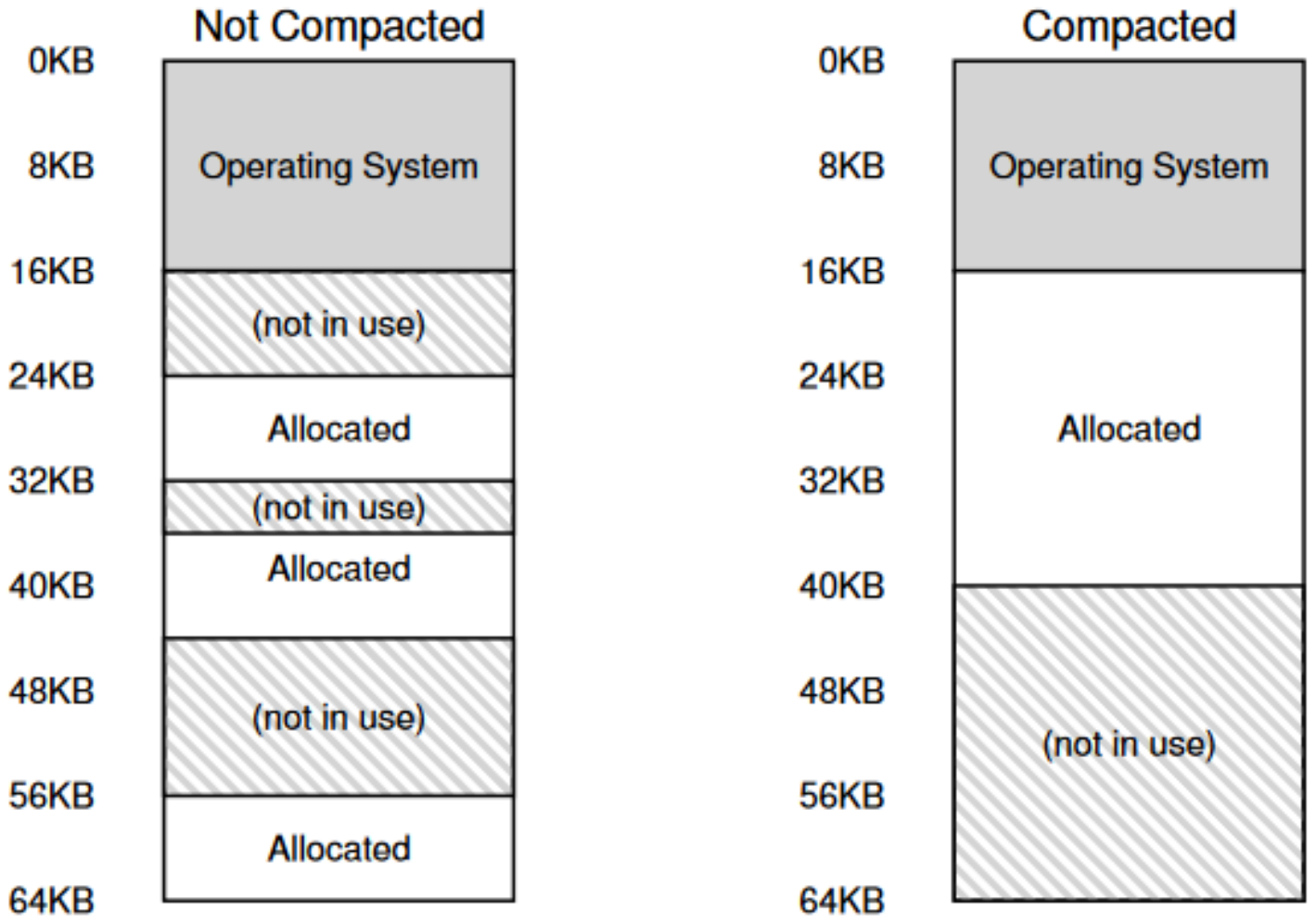
- Käyttöjärjestelmä lataa omat tietonsa alkusektorille, ja jakaa muuta aluetta prosessien käyttöön.

PERUSTEET MUISTISTA



- Vapaa muisti jaetaan lohkoiksi, joita annetaan prosessien käyttöön.
  - Käyttöjärjestelmä pitää kirjaa siitä, mikä prosessi on missäkin fyysisessä sijainnissa.
- Prosessien on tarkoitus työskennellä vain ja ainoastaan omalla alueellaan.
  - Mitä tapahtuisi jos prosessi pääsisi vaikkapa kirjoittamaan käyttöjärjestelmän muistiin?

PERUSTEET MUISTISTA

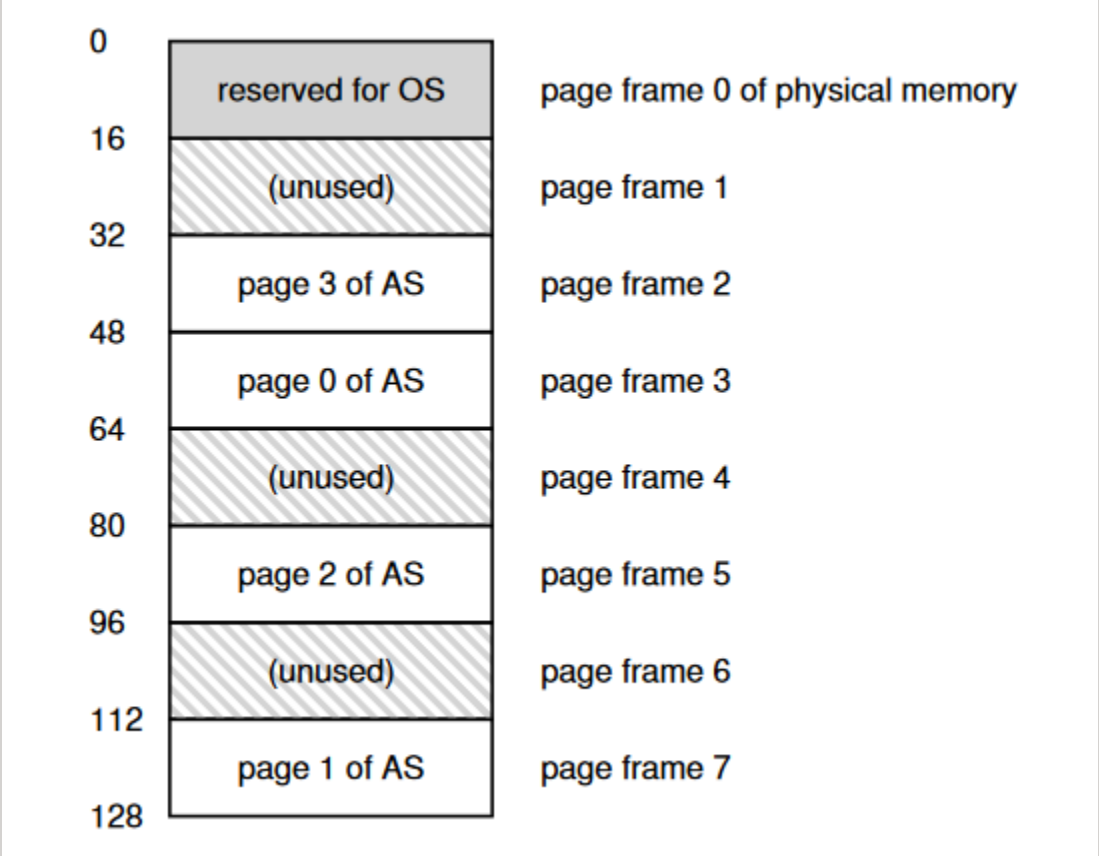


PERUSTEET MUISTISTA

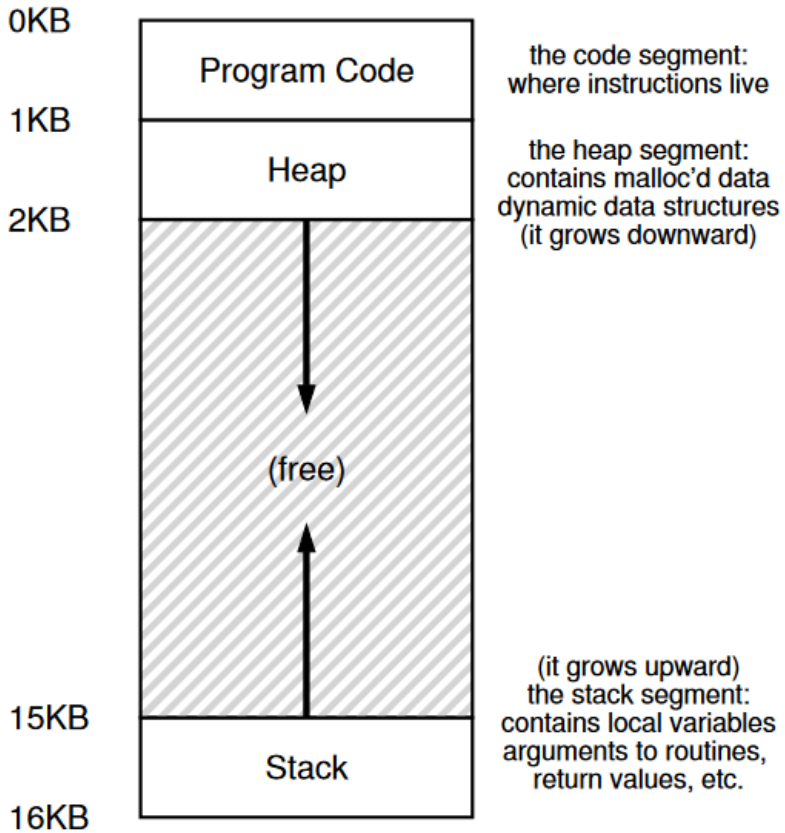
- Käyttöjärjestelmä myös huolehtii siitä, että muisti ei pääse pirstaloitumaan.
  - Best fit, worst fit, first fit…
- Muistin siivoaminen on kallis operaatio, mutta sitä pitää välillä tehdä.
- Lasketaan uudet optimoidut sijainnit, siirretään tiedot ja päivitetään rekisterit
- …Tai jos muisti on oikeasti täynnä, siirretään osa muistissa olevista tiedoista levylle siksi aikaa kun niitä ei tarvita. Tämä on hidasta ja tehotonta, mutta mahdollistaa isojen asioiden pyörittelemisen.

PERUSTEET MUISTISTA: SIVUTUS

- Idea: jaetaan muisti viipaleisiin.
  - Helppo pitää kirjaa mikä on käytössä ja mikä ei.
  - Helppo siirrellä paloja paikasta toiseen.
  - Helppo kirjata mitkä sivut on annettu minkäkin prosessin muistiavaruuteen.
- Käyttöjärjestelmällä on sivutustaulu (page table), jossa on tieto siitä missä mikäkin prosessin osa tai asia sijaitsee.

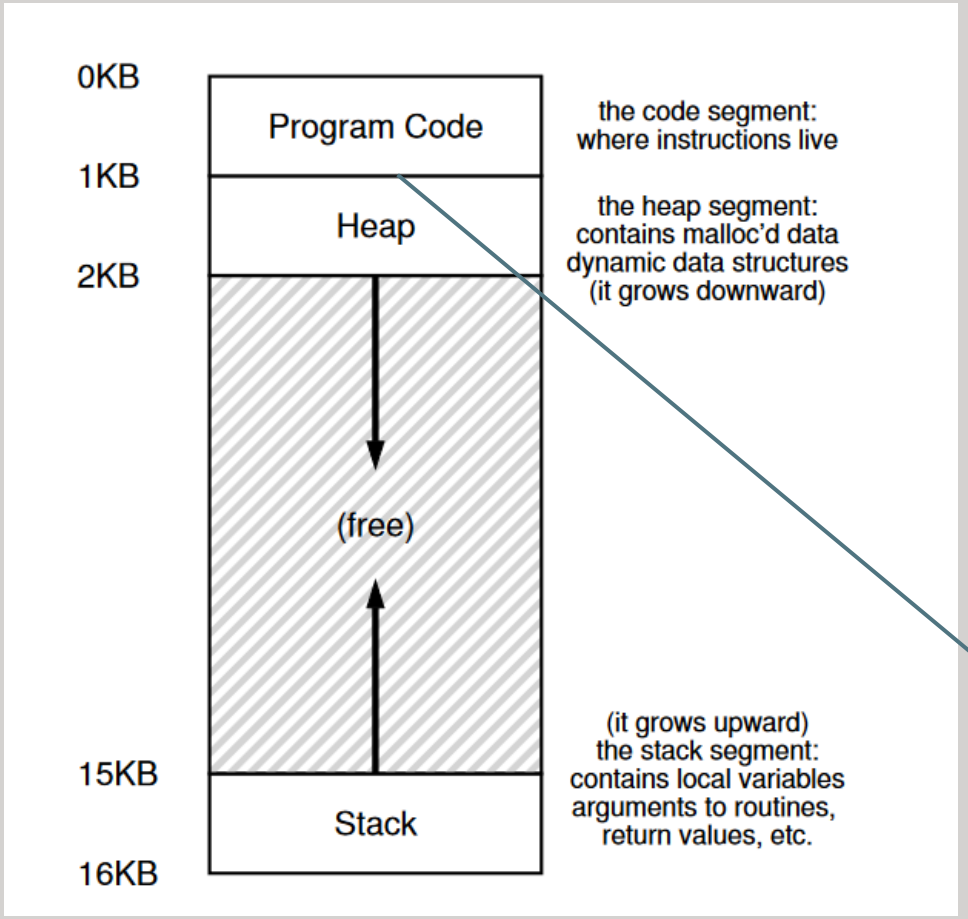


MITÄ PROSESSI LATAA MUISTIIN?



- Suoritettava tavukoodi
- Keko; ohjelman omat dynaamiset tietorakenteet (mallocilla luodut)
- Pino; ohjelman suorituksenaikainen työmuisti

MITÄ PROSESSI LATAA MUISTIIN?



- Suoritettava tavukoodi
- Keko; ohjelman omat dynaamiset tietorakenteet (mallocilla luodut)
- Pino; ohjelman suorituksenaikainen työmuisti

- Data (ei kuvassa) –alueella tallessa mm. erilaisten ympäristömuuttujien ja ohjelman omien muuttujien arvot. Normaalisti kuvattuna koodin ja keon väliin jos mainittu erikseen.

PROSESSIEN JA MUISTIALUEIDEN SUOJAUS

CT30A3370 - Käyttöjärjestelmät ja systeemiohjelmointi

PROSESSIEN SUOJAUS (TOISILTA PROSESSEILTA)

- Lähtökohta: samalla laitteella pitäisi pyörittää kahta eri prosessia
  - Mikä on prosessi…?
    - Eli ohjelman ajonaikainen suoritus, ohjelma on eri asia kuin prosessi
  - Prosesseja **pitää** suojella sekä toisiltaan..
  - Ja käyttöjärjestelmää pitää suojella prosesseilta
  - Yhden ohjelman suoritus, prosessi ei saanut sotkea toista!

PROSESSIEN SUOJAUS (TOISILTA PROSESSEILTA)

- Tätä varten pitää olla **mekanismeja** ja mekanismien kaverina pitää olla **protokollia**
  - Mekanismi tekee
  - Protokolla kertoo, miten mekanismeja käytetään
- Tavoite on siis
  - Estää sovellusohjelmia kaatamasta käyttistä
  - Estää sovellusohjelmia kaatamasta toisiaan
  - Estää käyttiksen osia kaatamasta toisiaan!

PROSESSIEN SUOJAUS (TOISILTA PROSESSEILTA)

- (Muutama) mekanismi millä tämä voidaan saavuttaa
  - Osoitteenmuunnos, Address translation
  - Käyttäjätila erillään etuoikeutetusta käyttöjärjestelmän tilasta
- (Yksinkertainen) protokolla
  - Sovellusohjelmat eivät saa lukea/kirjoittaa toisten ohjelmien tai käyttöjärjestelmän muistiin / muistiavaruuteen!

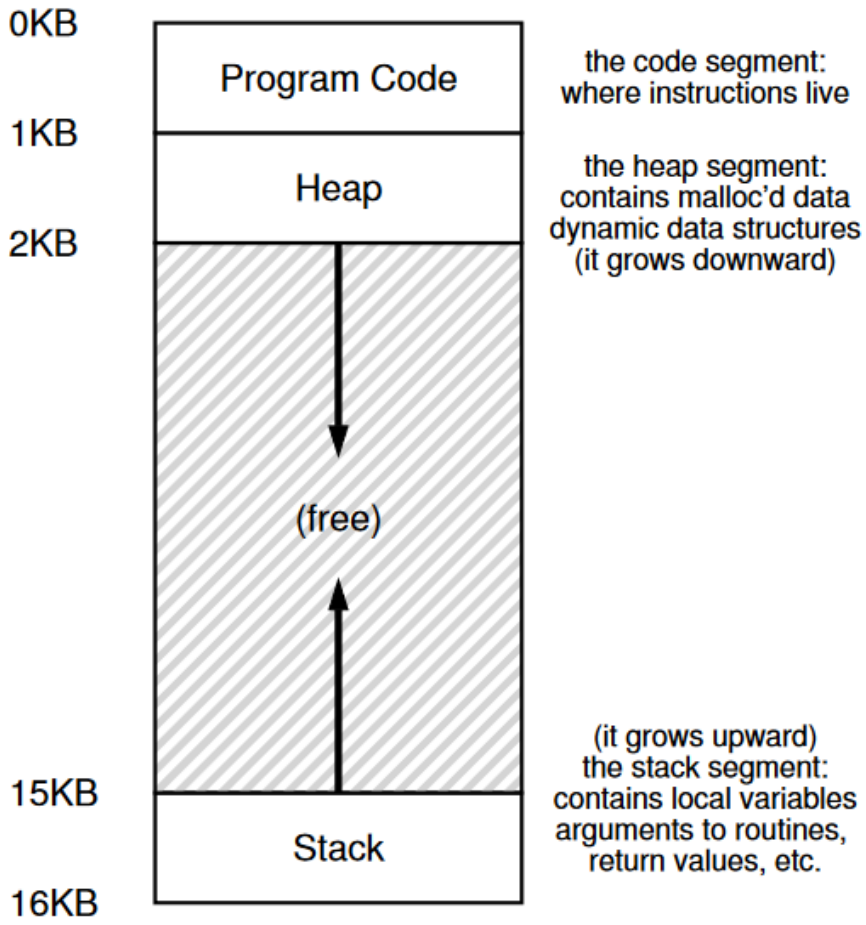
OSOITTEENMUUTOS

CT30A3370 - Käyttöjärjestelmät ja systeemiohjelmointi

OSOITTEENMUUTOS

- Osoitteenmuunnos on mekanismi, mikä virtualisoi keskusmuistin sovellusohjelmille
  - Sovellukset eivät näe oikeaa, fyysistä muistia, vaan ainoastaan käyttöjärjestelmän tarjoaman “virtuaalimuistin”

VIRTUAALIMUISTI



- ...Eli ohjelma luulee aina olevansa ainoa muistissa oleva asia ja muistialueella 0-N on hänen lähdekoodinsa ja loput on työmuistia.
- Samaan tapaan kuin ohjelma luulee olevansa ainoa suorittimella ajossa oleva asia.
- Käyttöjärjestelmän oma koodi (ja muut prosessit) ulottumattomissa

OSOITTEENMUUTOS

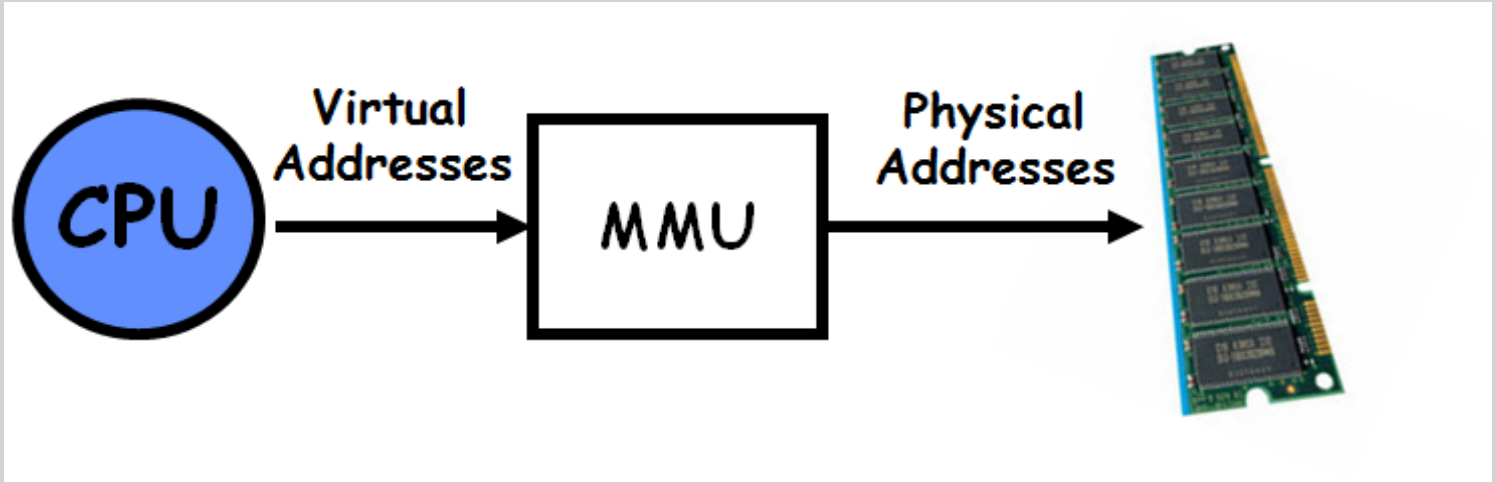
- Käytännössä ohjelma näkee muistiosoitteita vaikkapa nollasta 2^32:een asti
  - Sovellusohjelma kuvittelee käyttävänsä näitä osoitteita, mutta fyysisessä muistissa tilanne voi olla aivan toinen, koska käyttöjärjestelmän muistinhallintayksikkö, MMU (memory management unit) kääntää näitä virtuaalisia osoitteita fyysisiksi osoitteiksi.
    - …Eli MMU tietää mitkä muistin sivut, ja missä järjestyksessä, ovat prosessin käytössä.
    - Eli oikeat, fyysiset muistiosoitteet, eroavat sovelluksen näkemistä virtuaalisista osoitteista!

OSOITTEENMUUTOS

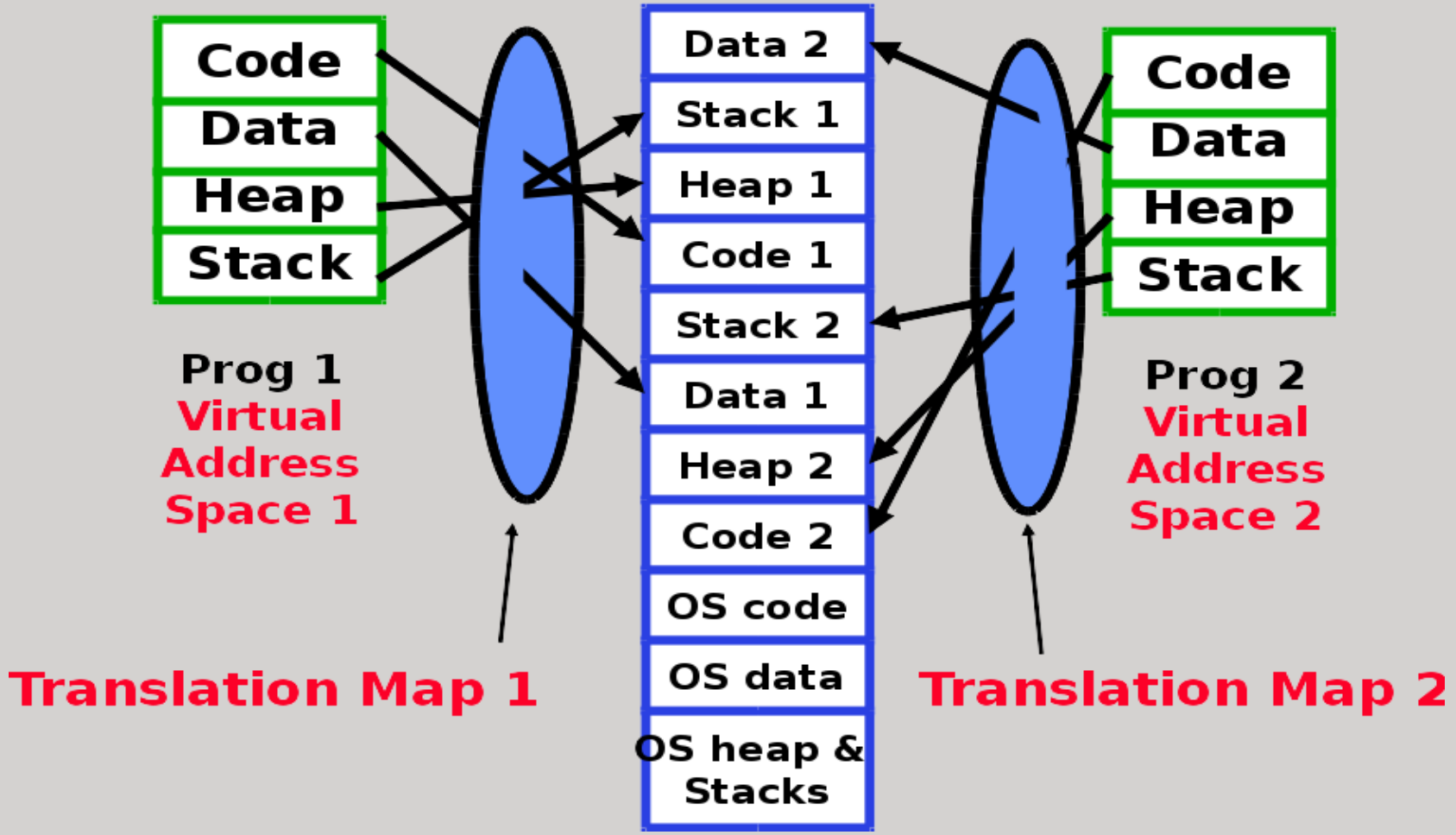
- Aina kun ohjelma kirjoittaa muistiin, muistiosoite, mihin kirjoitetaan laitetaan osoitteenmuunnostaulun läpi, eli etsitään osoitteista kirjaa pitävästä taulusta mikä virtuaalinen osoite vastaa mitäkin fyysistä osoitetta.
- Tässä huomio: osoitteenmuunnos pitää toimia tehokkaasti, se ei saa olla kohtuuttomasti resursseja hukkaava operaatio

OSOITTEENMUUTOS

- Suorittimelta tulevat ja sinne menevät muistiosoitteet menevät osoitteenmuunnoslaatikon läpi
- Laatikko tuottaa oikean, fyysisen muistin osoitteen, mikä menee keskusmuistiin



OSOITTEENMUUTOS



OSOITTEENMUUTOS

- Ohjelmilla on oma muistiavaruutensa
  - Osoitteenmuunnoksen jälkeen suoritin tietää, missä nämä “asiat” sijaitsevat oikeasti keskusmuistissa.
- Ohjelmien muistiavaruudet eivät mene päällekkäin.
  - …Paitsi silloin kun ne erikseen laitetaan menemään jaettuna muistina, mutta tämä on optimointiratkaisu joka ei ole tässä vaiheessa keskeinen asia.
- Ohjelmille tarjottavat muistiosoitteet eivät voi mennä päällekkäin fyysisessä muistissa
  - Tällä estetään se, että edes teoriassa yksi ohjelma ei voi sotkea toisen suoritusta menemällä sen muistialueelle.

OSOITTEENMUUTOS

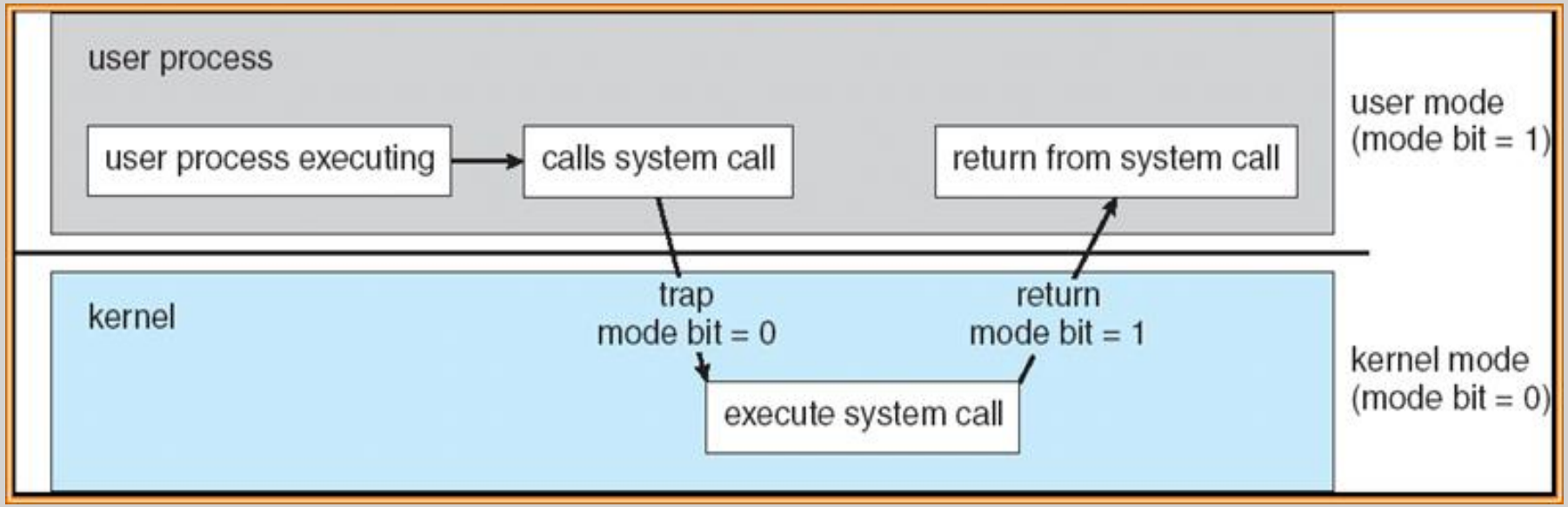
- Miksi pelkkä osoitteenmuunnos ei ole riittävä mekanismi turvaamaan ohjelmien / prosessien suoritusta?
- **Kuka** pääsee osoitteenmuunnostauluun käsiksi?
  - Käyttöjärjestelmä sinänsä on vain ohjelmisto siinä missä muutkin
    - Jos KJ pääsee osoitetauluun käsiksi, niin kyllähän sinne pääsee kuka tahansa muukin? Mistä muuten tiedettäisiin, mihin yhden ohjelman virtuaalimuisti osoittaa?
  - .. ja jos ohjelma pystyy itse muuttamaan osoitetaulua, se pystyy lukemaan / kirjoittamaan minne vain

OSOITTEENMUUTOS

- => Tarvitaan siis joku mekanismi, millä varmistetaan, että tiettyjä osia muistista ei pysty muokkaamaan kukaan muu kuin KJ!
  - Käyttöjärjestelmän täytyy siis pystyä pyörimään eri oikeuksilla kuin sovellusohjelmien?
- Intuitiivisesti tämä lie aika selvää? Olette käytännössä törmänneet eri käyttäjiin joilla on eri prosesseja ja eri oikeuksia tehdä.. Mitä nyt sitten ikinä ne tekevätkin.

KÄYTTÄJÄTILA JA ETUOIKEUTETTU TILA (KERNEL MODE)

- Jotta tällaisessa muistinhallinnassa olisi mitään järkeä, muistiosoitteiden muunnoksen olisi parempi olla pelkästään KJ:n hallittavissa.



KÄYTTÄJÄTILA JA ETUOIKEUTETTU TILA (KERNEL MODE)

- Useimmat tietokonelaitteistot tarjoavat (ainakin) kaksi suorituksen tilaa ohjelmille
  - Kernelitila (etuoikeutettu, suojattu)
  - Käyttäjätila
- Itse asiassa, esimerkiksi x86 - prosessoriarkkitehtuurissa on 4 moodia, millä ohjelmia voidaan ajaa...
  - Kaksi tarvitaan ainakin. Mutta, määrä voi vaihdella.

KÄYTTÄJÄTILA JA ETUOIKEUTETTU TILA (KERNEL MODE)

- Eli, osa laitteiston resursseista voidaan rajoittaa pelkästään käyttöjärjestelmän käyttöön ja osa pelkästään sovellusohjelmien käyttöön.
  - Osoitteenmuunnos tapahtuu pelkästään suojatussa tilassa
  - Osoitteenmuunnostauluun pääsee käsiksi vain käyttöjärjestelmän muistinhallintaosa (MMU)
- Joitain laitteiston käskyjä (konekieli), operaatioita, ei pysty suorittamaan käyttäjätilassa
  - Esimerkiksi: Osoitetaulua ei pysty muuttaamaan
    - Jos yrität, poikkeus, errori syntyy.
    - Game over, page fault, segmentation fault jne.

KÄYTTÄJÄTILA JA ETUOIKEUTETTU TILA (KERNEL MODE)

- Ongelma: Miten käyttäjätilasta vaihdetaan suojattuun (kerneli) tilaan?
  - Jos meillä on suorituksessa ohjelma käyttäjätilassa… miten päästä takaisin kernelitilaan?
  - “Suoritus pyörii käyttäjätilassa, tarvitsen jotain laitteistoresurssia, miten päästä kernelitilaan”?
- Tätä varten meillä on Käyttöjärjestelmän tarjoama rajapinta ja järjestelmäkutsut
  - Tämä on se juttu, minkä takia tämän kurssin nimessä on ‘systeemiohjelmointi’
    - System programming, järjestelmäkutsu = system call
  - Toisen periodin asia on käyttää tätä järjestelmän kutsurajapintaa, jotta voidaan toteuttaa suhteellisen matalalla tasolla ohjelmia

KÄYTTÄJÄTILA JA ETUOIKEUTETTU TILA (KERNEL MODE)

- Eli siis, järjestelmäkutsurajapinta.
  - Sovellusohjelmahan ei voi vain todeta, että “käynnistetäänpä kernelitila ja toimitaan seuraavaksi etuoikeutetusti”, se olisi vähän turhaa
    - Ja koko hommasta menisi pohja pois, kernelitilassa ohjelma voisi taas vaikkapa mennä muokkaamaan sitä osoitetaulua…
- Järjestelmäkutsu aiheuttaa nyt sen, että ohjelman suoritus siirtyy “hyvin tarkasti rajatulle käyttöjärjestelmän alueelle”, joka tekee sen palvelupyynnön, minkä sovellusohjelma pyysi.
  - Ohjelma itsessään ei edellenkään pääse toimimaan kernelitilassa etuoikeutetusti.
  - Tässä välissä anneta ohjelmalle mitään ylimääräisiä oikeuksia

KÄYTTÄJÄTILA JA ETUOIKEUTETTU TILA (KERNEL MODE)

- “Like warden in an insane asylum. Put patients in locked padded cell – patient can bang head against wall, but otherwise can’t cause any problems.
- Pascal – like putting them in a straight jacket. C is like giving them knives to play with.”

PINO JA SEN KÄYTTÖ MUISTINHALLINNASSA

PINO-OPERAATIOT

- Pino (stack) on abstrakti datatyyppi, mitä hallitaan käytännössä push- ja pop -operaatioilla
- Pino kun on tyypillinen LIFO (last-in-first-out) tietorakenne
- Eli siis
  - Push lisää alkion pinon päälle
  - Pop hakee pinosta päällimmäisen elementin ja samalla poistaa sen pinon päältä. Samalla toisiksi ylimmäinen elementti nousee päällimmäiseksi

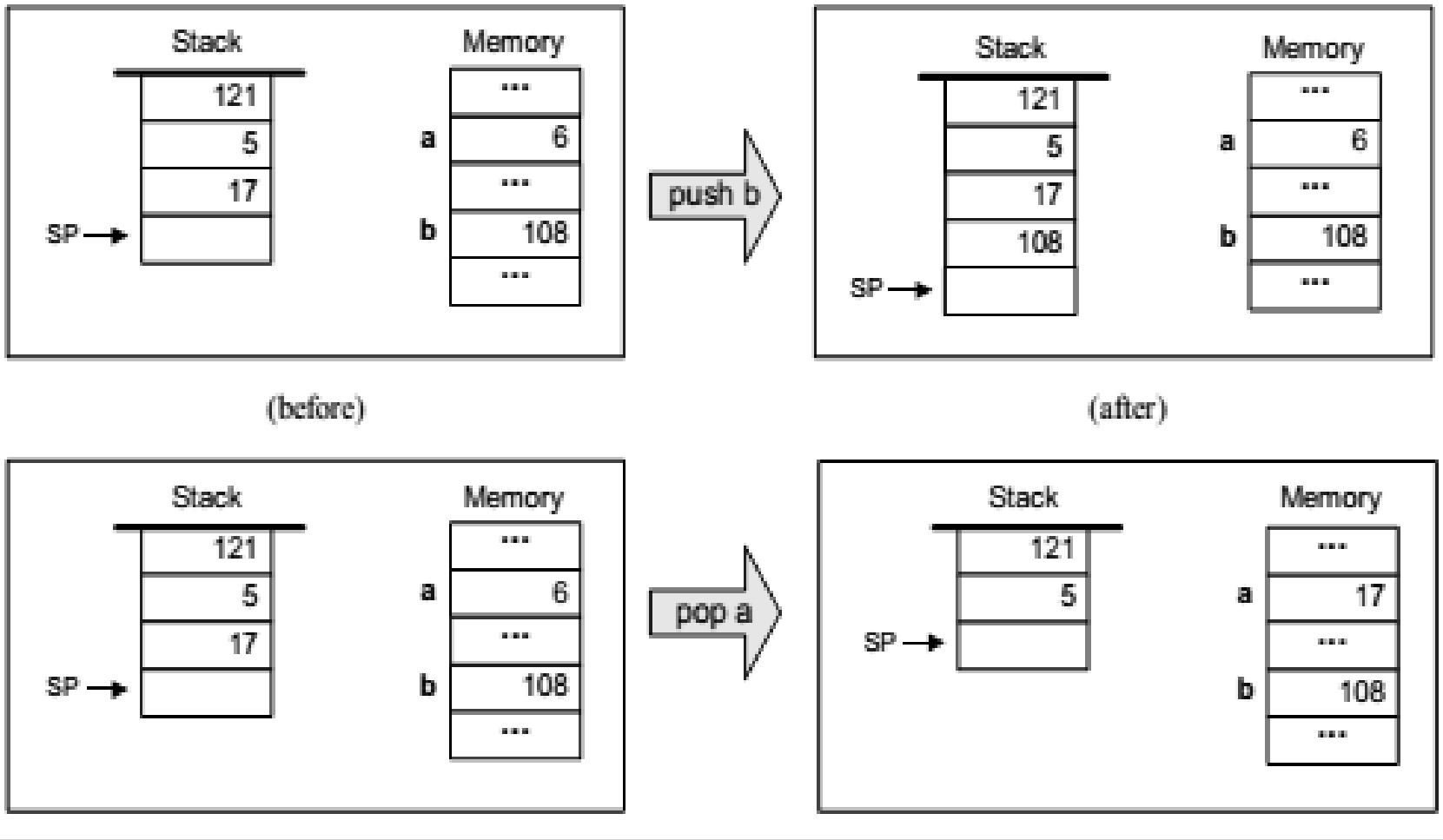
PINO-OPERAATIOT

- Pino siis eroaa perinteisestä muistinhallinnasta jokseenkin
  - Ainoastaan päällimmäiseen elementtiin pinossa on mahdollista päästä käsiksi
  - Ja elementteihin päästään käsiksi vain yksi kerrallaan.
  - Toisekseen: pinon käsittely on “dataa hukkaava” operaatio!
    - Ainoa tapa saada pinon päällimmäinen elementti käsiin on poistaa se pinosta kokonaan!
    - Verrattaen luvun lukeminen muistista ei vaikuta muistin tilaan mitenkään.
  - Ja vastaavasti pinoon lisääminen ei vaikuta jo pinossa olleisiin elementteihin mitenkään
    - Muistiin kirjoittaminen olisi puolestaan “dataa hukkaava” operaatio, sillä muistiin kirjoittaessa muistipaikan vanha arvo ylikirjoitetaan aina.

PINO-OPERAATIOT

- Pino voidaan toteuttaa muutamilla eri tavoilla
  - Yksinkertaisin on varmaan varata muistista määrämittainen taulukko, stackki, missä yhden elementin koko on VM-koko operointikoodin verran
  - sekä muuttuja, *stack pointer*, SP, käskyosoitin pitämään kirjaa päällimmäisestä elementistä

PINO-OPERAATIOT



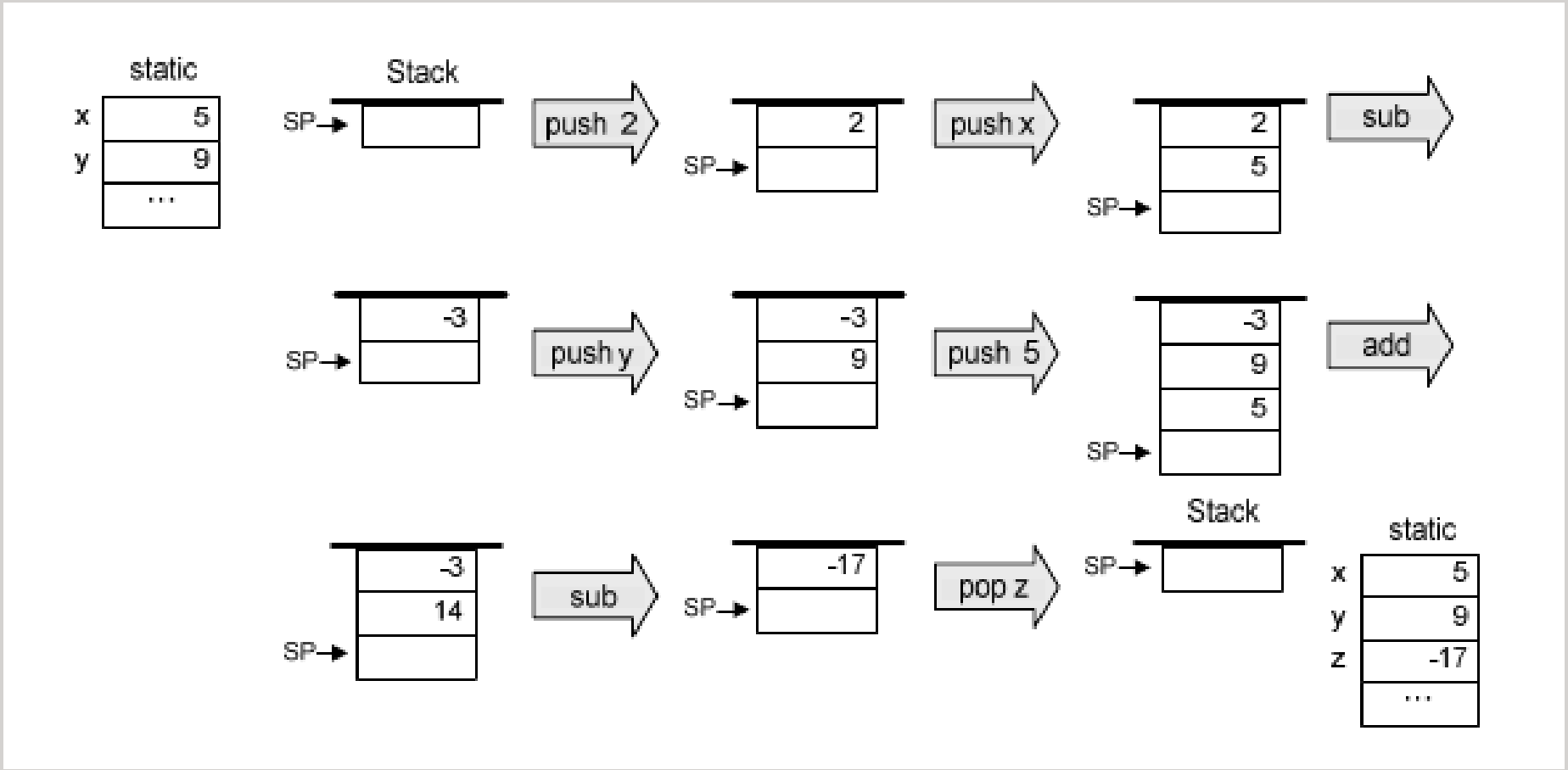
PINOKONEEN (STACK MACHINE) MALLI

- Mikä mielenkiintoisinta, pinon avulla pystytään toteuttamaan minkä tahansa aritmeettisen tai loogisen operaation suoritus (/ laskenta)
  - Vielä näppärämpää on se, että mikä tahansa ohjelmakoodi kirjoitettuna millä tahansa ohjelmointikielellä, voidaan kääntää pinokoneessa suoritettavaksi / toimivaksi ohjelmaksi!
- Java-virtuaalikone on hyvä oikean elämän esimerkki, se on tyypillinen pinokoneeseen perustuva virtuaalikonetoteutus.

PINOARITMETIIKKA: ESIMERKKI

```
//  z=(2-x)-(y+5)
push 2
push x
sub
push y
push 5
add
sub
pop z
```

PINOARITMETIIKKA: ESIMERKKI
Z = (2-X)-(Y+5)

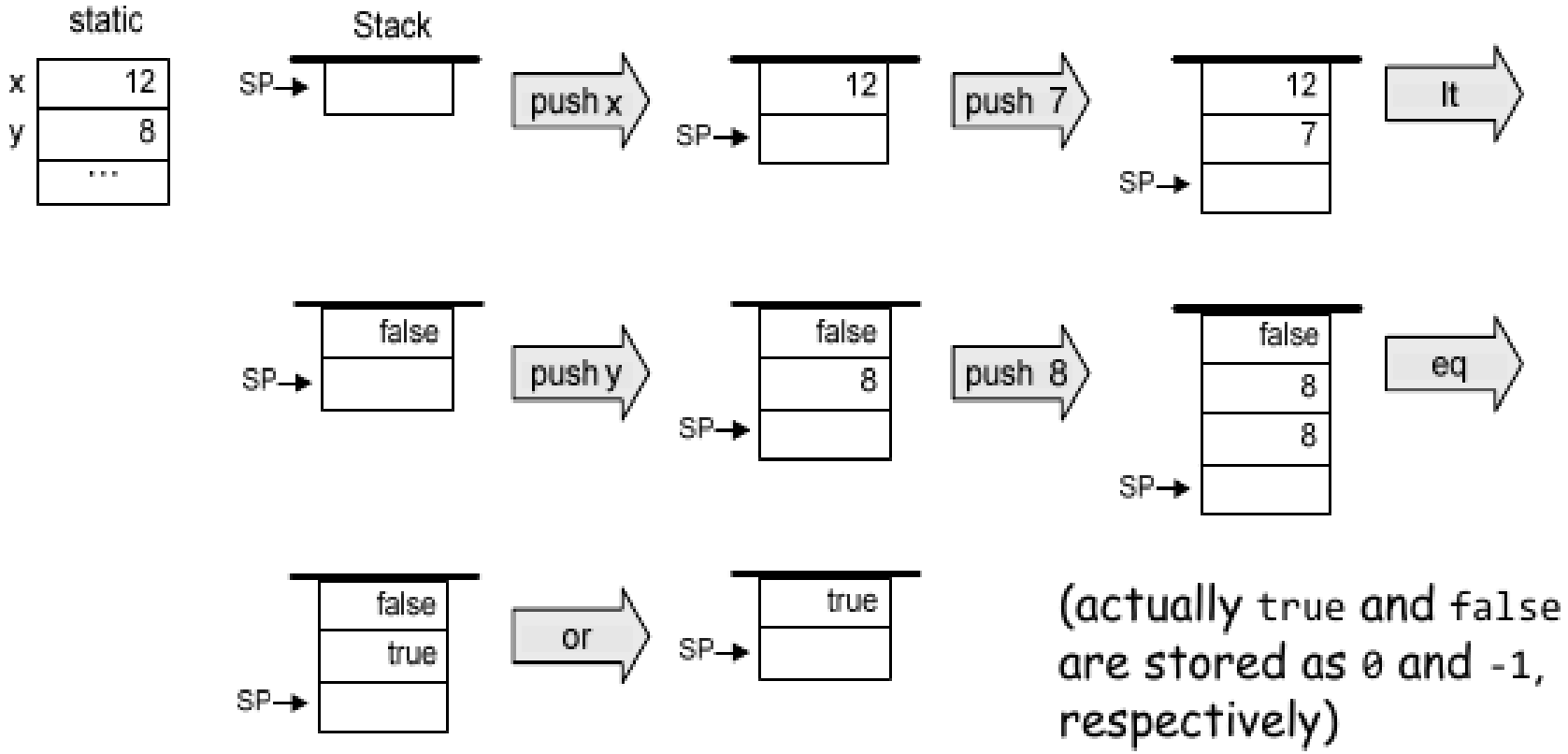


PINOPERUSTEINEN LOGIIKKA

- Loogisten operaatioiden suoritus toimii aivan samalla tavalla!

```
// if (x<7) or (y=8)
push x
push 7
lt
push y
push 8
eq
or
```

PINOPERUSTEINEN LOGIIKKA
IF (X<7) OR (Y=8)

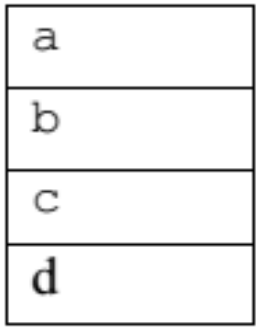


ALIOHJELMIEN KUTSUPINO

```
method a:
    call b
    call c

method b:
    call c
    call d

method c:
    call d

method d:
    ...
```

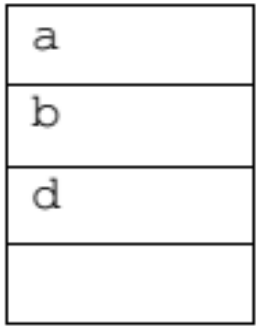
start a
start b
start c
start d

stack state→



end d
end c
start d

stack state→



end d
end b
start c
start d
end d
end c
end a

MITÄ TÄSTÄ LUENNOSTA PITÄÄ MUISTAA?

- Muistinhallinnan perusteet
- Pinokoneen perusteet

LUT
University