LUT
University

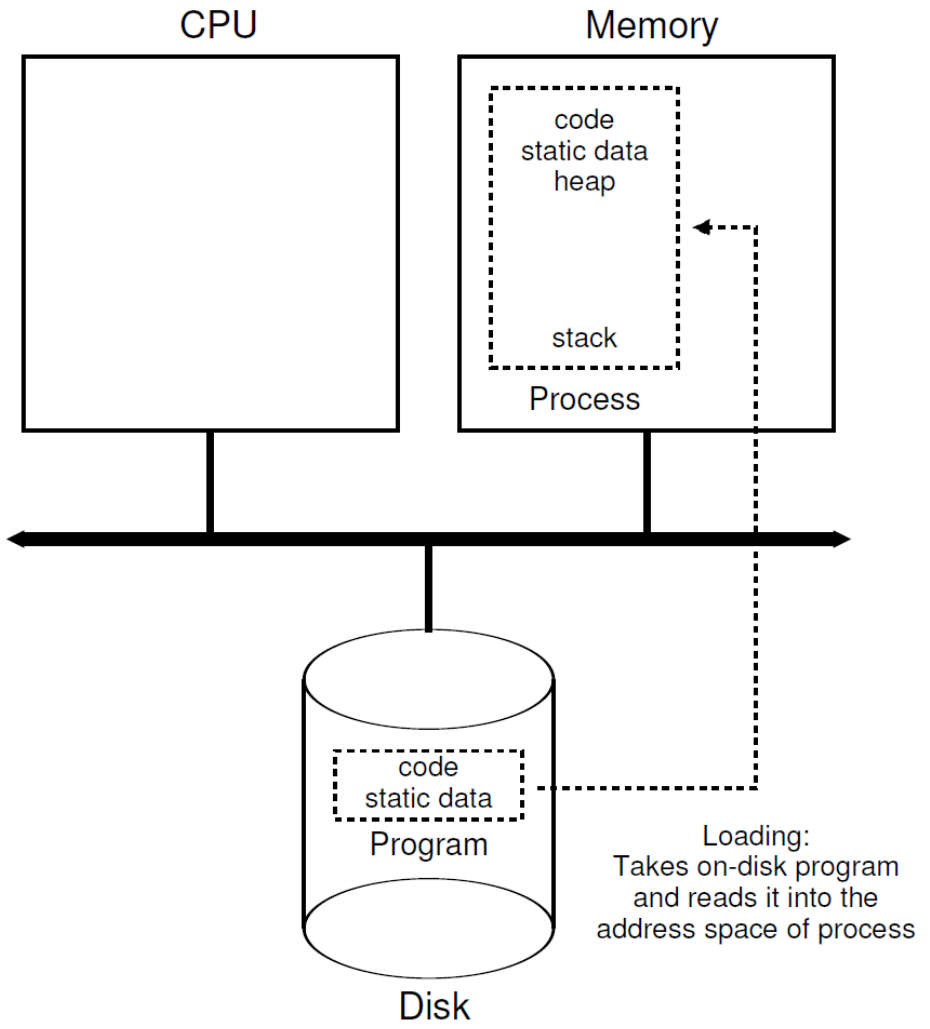
CT30A3370 - KÄYTTÖJÄRJESTELMÄT JA SYSTEEMIOHJELMOINTI 6 OP

Jussi Kasurinen (etu.suku@lut.fi)

Osa kalvoista Timo Hynnisen 2016 materiaaleista

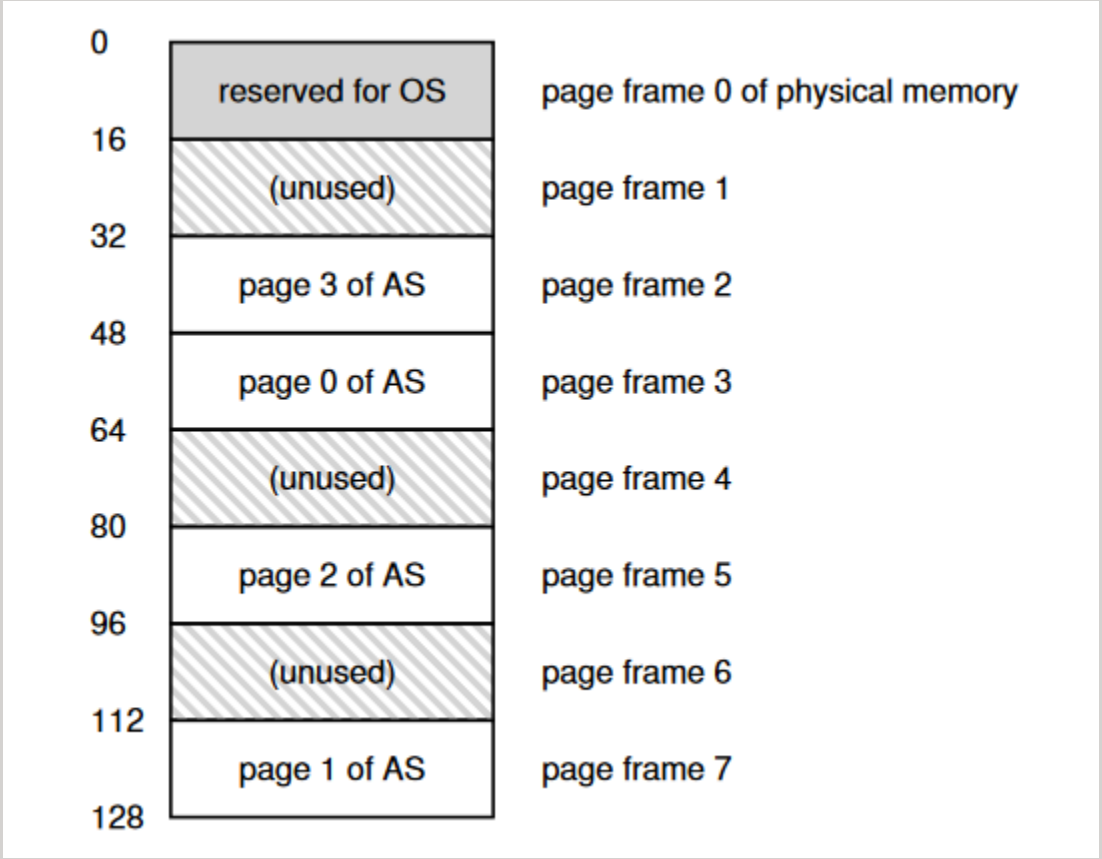
EDELLISILTÄ LUENNOILTA:

OHJELMA VS. PROSESSI

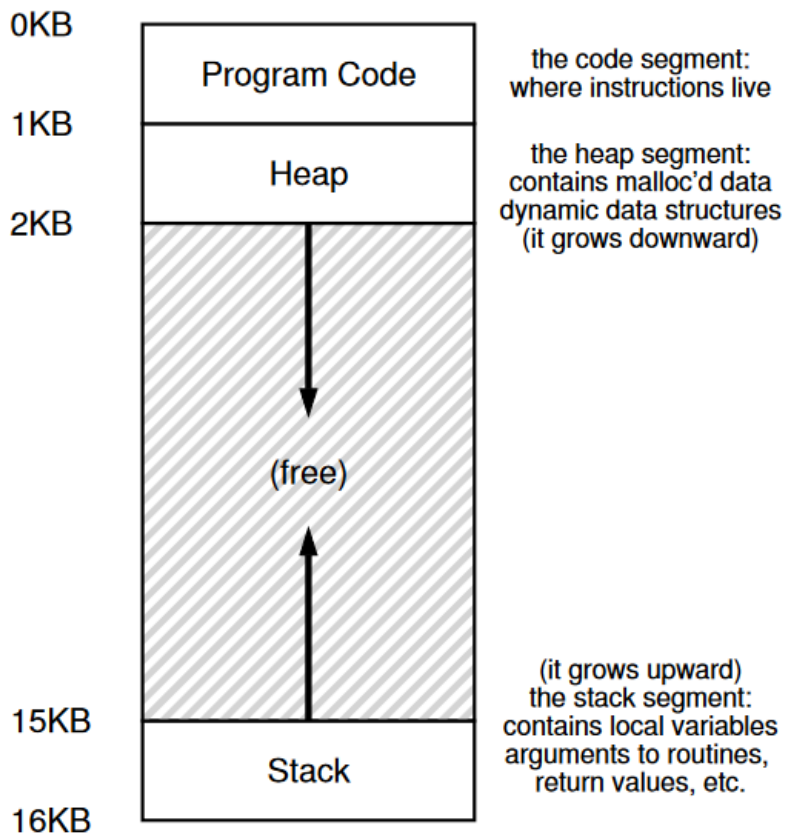


PERUSTEET MUISTISTA: SIVUTUS

- 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?



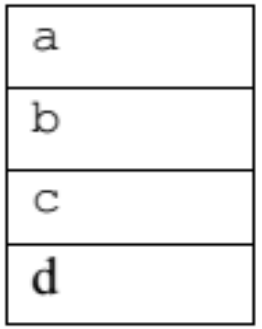
- Suoritettava tavukoodi
- Keko; ohjelman omat dynaamiset tietorakenteet (mallocilla luodut)
- Pino; ohjelman suorituksenaikainen työmuisti

ALIOHJELMIEN KUTSUPINO

```
method a:
    call b
    call c

method b:
    call c
    call d

method c:
    call d

method d:
    ...
```

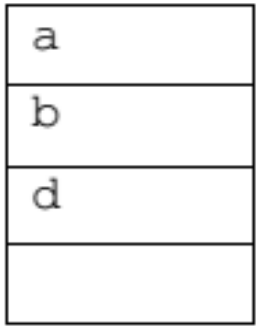
start a
start b
start c
start d

stack state→



end d
end c
start d

stack state→



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

KERNELIN JA KÄYTTÖJÄRJESTELMÄN RAKENNE

CT30A3370 - Käyttöjärjestelmät ja systeemiohjelmointi

MISSÄ MENNÄÄN?

- Meillä on suoritin, jossa ajetaan konekielisiä komentoja.
  - Useita eri prosesseja
  - Muistinhallinta, missä prosessit pääsee omille muistialueilleen.
  - …Ja joku idea siitä, miten koko kaaosta hallitaan.
- Seuraavaksi puhutaan siitä, miten tämä hallintakoneisto rakentuu, ja miten sitä ohjataan.

KÄYTTÖJÄRJESTELMÄN OSAT JA PALVELUT

- Ohjelmien suorituksen hallinta
  - Miten suoritetaan peräkkäistä ohjelmakoodia / käskyjä, jotta niistä muodostuu jokin mielekäs ohjelma?
- Tarvitaan lisäksi siirräntämekanismeja (input output, syöttö / tulostus)
  - Esimerkiksi tiedostoselaimella pystyy tarkastelemaan sekä tiedostoja kiintolevyllä että esimerkiksi verkon yli verkkolevyltä tai vaikkapa USB-muistilta!

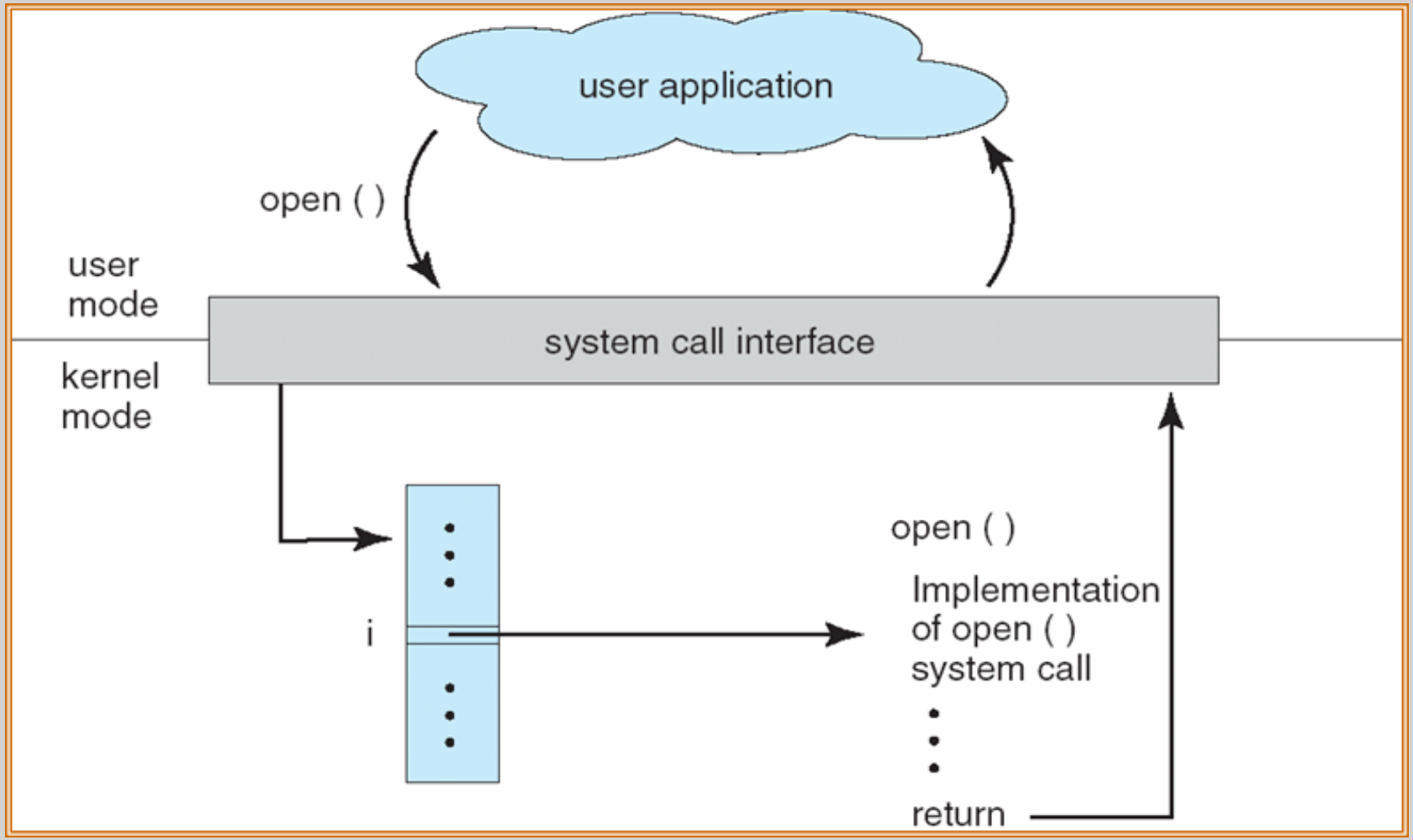
KÄYTTÖJÄRJESTELMÄN OSAT JA PALVELUT

- Tiedostojärjestelmän hallinta
  - Miten luetaan / kirjoitetaan / säilötään tiedostoja
  - Miten edes löydetään tiedostot muistista?
- Verkkoyhteydet
  - Näin herran vuonna 2018 kaikki on verkossa / pilvessä / missä lie
  - Voiko verkkopalvelusta nykyään puhua jo käyttöjärjestelmänä?!
- Lisäksi ehkä virheenetsintää ja korjausta, kirjanpitoa, resurssien jakoa…

JÄRJESTELMÄKUTSUT

- Sitten on järjestelmäkutsut ja rajapinnat
  - sovellusohjelmat tekevät järjestelmäkutsuja, jotka saavat KJ:n tekemään asioita puolestaan.
  - Käyttöjärjestelmä tarjoaa meille erilaisia rajapintoja, joita ohjelmoija pääsee käyttämään.

JÄRJESTELMÄKUTSUT



KÄYTTÖJÄRJESTELMÄN RAKENNE

- Minkälainen on käyttöjärjestelmän (ytimen) rakenne?
- Voi olla yksinkertainen
  - Muutama eri ohjelman suorituksen taso
  - Riippuu järjestelmän kompleksisuudesta
  - MS-DOS: yksi ohjelman suoritustaso (kaikki koodi samanarvoista)
  - Tai kaksikerroksinen (käyttäjä ja kerneli) kuten UNIX
    - Unixissa 2 suorituksen tasoa, KJ prosessit etuoikeutetussa tilassa

KÄYTTÖJÄRJESTELMÄN RAKENNE

- Monoliittinen
  - Käyttöjärjestelmä hoitaa sisäisesti kaikki palvelut ja toiminnot, jotka eivät ole käyttäjän suorittamia prosesseja.
  - Ei sisäistä hierarkiaa, kaikki osana yhtä isoa rakennelmaa.
- Kerroksittainen
  - Alemmat kerrokset erillään ylemmistä
- Mikrokerneli
  - Käyttöjärjestelmä koostuu useista käyttäjätason prosesseista / palveluista
  - Tehdään käyttöjärjestelmän osista itsenäisiä palveluja
    - Omat prosessinsa, jotka pyörivät omassa muistiavaruudessaan suojassa toisiltaan
    - Tällöin eri kernelin osat / käyttiksen palvelut eivät pääse sotkemaan toistensa suorituksia, koska ne eivät pääse toistensa muistialueisiin käsiksi
  - Esim. Windows

KÄYTTÖJÄRJESTELMÄN RAKENNE

- Modulaarinen
  - Pieni ydin, johon voidaan liittää dynaamisesti (==ajonaikaisesti) moduuleja
  - Tyypillisesti Linux
- Kaikki edellä mainitut ovat eri tapoja hallita kompleksisuutta
  - Eri rakenteissa on omat huonot ja hyvät puolensa
    - Eri tavoilla tehdä asioita on omat kannattajakuntansa: Esimerkiksi vaikkapa Torvaldsin näkemykset agressiivisen ehdottomia...
  - Joka tapauksessa, kompleksisuudesta pääsee eroon sitä jollain tavalla hallitsemalla
    - Voidaan jakaa kerroksiin, hajoittaa, abstrahoida...

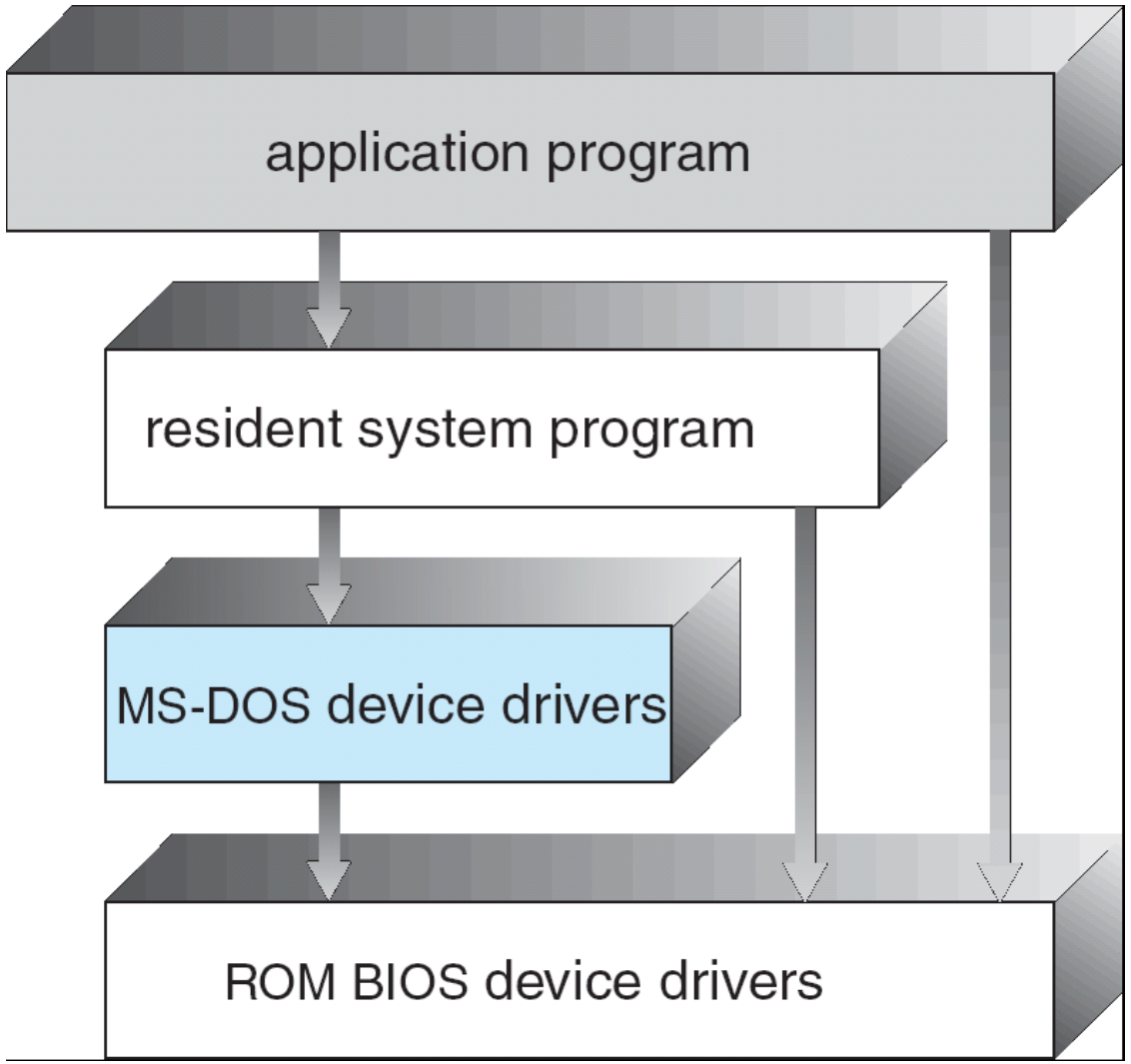
KÄYTTÖJÄRJESTELMIEN RAKENTEITA

CT30A3370 - Käyttöjärjestelmät ja systeemiohjelmointi

YKSINKERTAINEN RAKENNE

- MS-DOS
- “Mahdollisimman paljon toiminnallisuutta pienimmässä mahdollisessa tilassa”
  - Ei ole jaettu modulaariseksi
  - Rajapinnat ja toimintojen tasot huonosti erotettu toisistaan
  - Ei juuri suojausta (kernelin suojausta sovellukselta tai itseltään)
  - KJ on lähinnä kirjasto, joka pultataan sovellusohjelman kylkeen

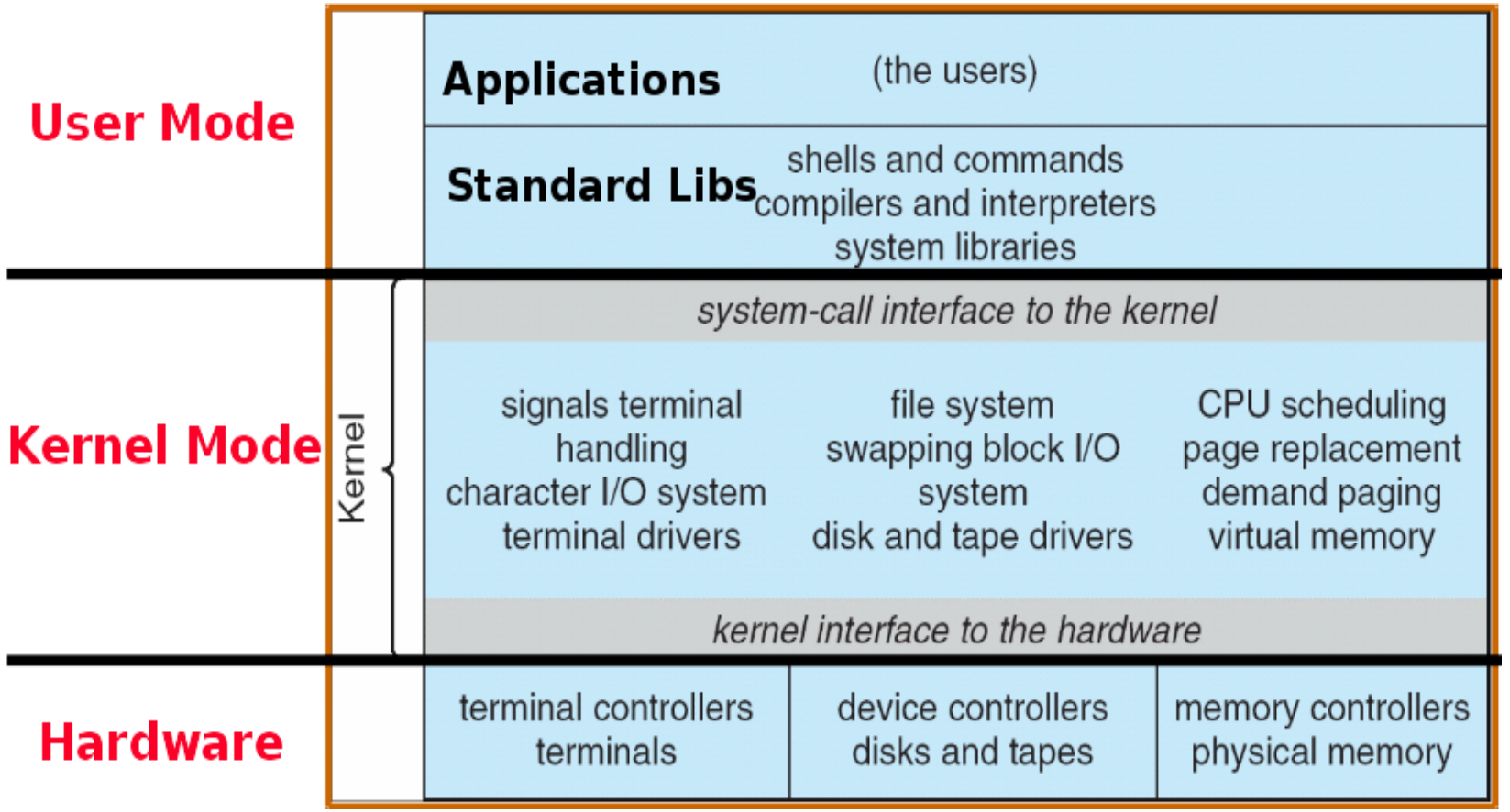
YKSINKERTAINEN RAKENNE



UNIX

- Myöskin yksinkertainen rakenne, jossa kaksi tasoa
  - Kerneli ja käyttäjä
- Kernelin sisällä tässä on kaikenlaista, mitä perinteisesti ollaan ymmärretty käyttöjärjestelmänä
  - Tiedostojärjestelmä, suorittimen vuorotus, virtuaalimuisti
  - Kaikki Kernelin sisällä on suojattua ulkopuolelta, toisin sanoen sovelluksilta.
  - Esimerkki monoliittisestä ytimestä
- Käyttäjätila on sitten kaikki tavalliset tietokoneohjelmat
- Ongelma tämän rakenteen kanssa: Jos jokin kernelin osa kaatuu, kaikki kaatuu
  - Ei olla suojauduttu siltä, että kerneli itse mokaa, vaikka kerneli onkin suojattu sovellusohjelmien spurdoilulta

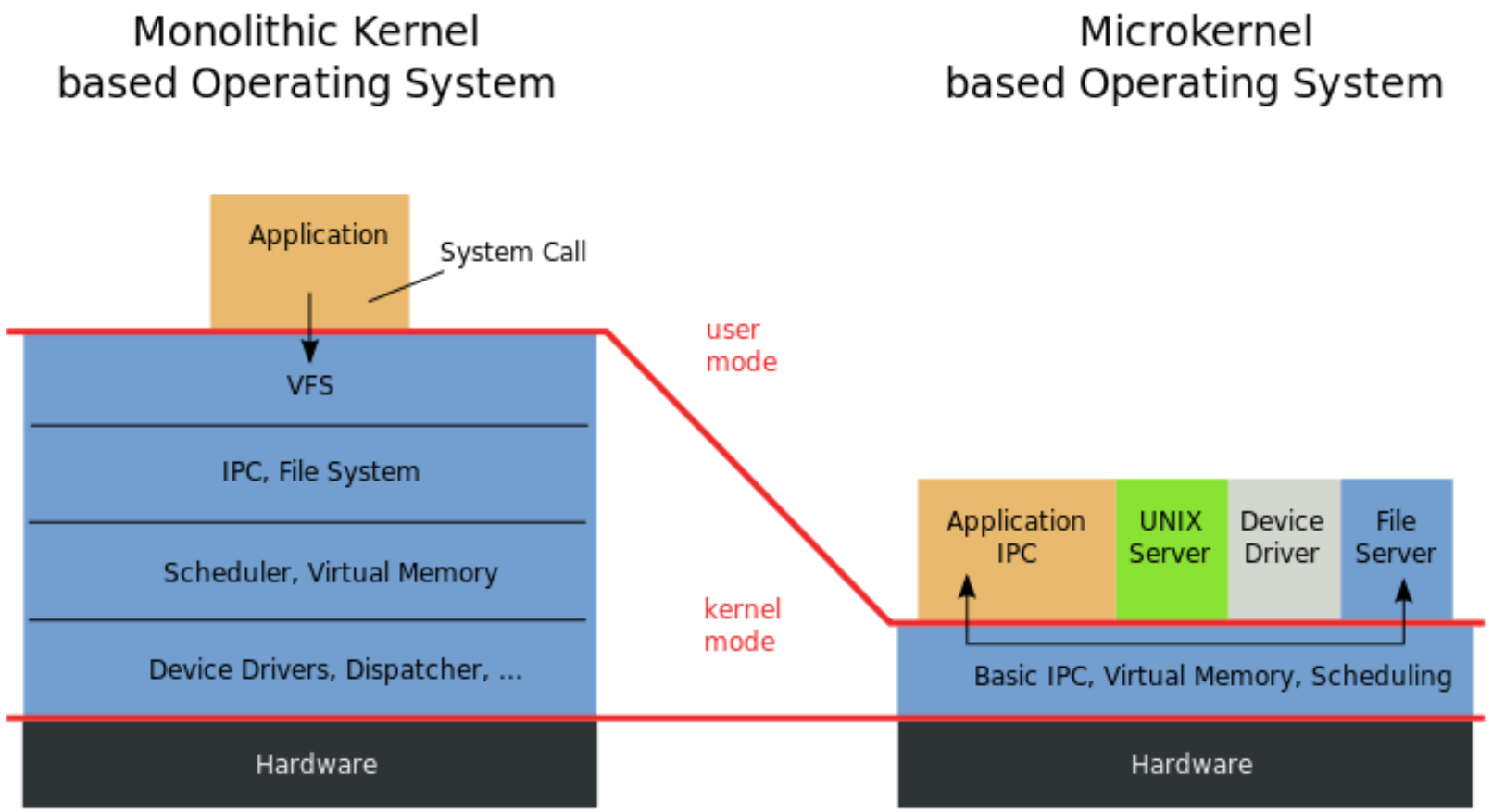
UNIXIN RAKENNE



KERROKSITTAINEN RAKENNE

- Rakennetaan enemmän kuin kaksi kerrosta
  - Otetaan hyvät puolet Unixin rakenteesta ja korjataan ongelmakohtia?
- Alin kerros on edelleen laitteistotaso
- Ylin kerros on käyttäjätaso
- Näiden välissä on useita kerroksia, jotka tarjoavat eri tasoisia palveluita
  - Ylemmän tason operaatiot (funktiot) voivat käyttää vain alemman tason palveluita
  - Etu: modulaarisuus, mikä helpottaa ylläpitoa ja vikojen korjausta
- Huono puoli: Kerroksiin jakaminen ei välttämättä ole helppoa
  - Pitäisi eksplisiittisesti päättää palveluiden tärkeysjärjestys!
  - Esim. pitäisikö vuorontajan sijaita virtuaalimuistin (osoitteenmuunnoksen) päällä vai alla?

MIKROKERNEL



MIKROKERNEL

- Mahdollisimman paljon käyttöjärjestelmän palveluja ulos suojatusta tilasta käyttäjätilaan omiksi prosesseikseen.
- MIkrokerneli-rakenteessa on ohut, nimensä mukaisesti, mikro-kerneli, joka tarjoaa vain muutamia palveluita
- Käytännössä, se osaa vuorottaa tehtäviä, hallinnoi ja tarjoaa virtuaalimuistia prosesseille ja mahdollistaa yksinkertaisen prosessien välisen kommunikoinnin

MIKROKERNEL

- Kaikki muut palvelut ovat mikrokernelin päällä, ja pyörivät käyttäjätilassa!
  - Ja omissa muistiavaruuksissaan, eli suojassa muilta
  - Eli nyt voikin yhtäkkiä olla erilaisia laiteajureita käyttäjätilassa.
  - Tai tiedostojärjestelmä käyttäjätilassa.
  - Keskeinen idea tässä siis: Kaikki, minkä ei tarvitse olla tekemisissä laitteistorajapinnan kanssa, on eriytetty mikrokernelin yläpuolelle

MIKROKERNEL

- Keskeinen ero ohjelmoijan kannalta se, miten järjestelmäkutsut toimivat:
  - Koska järjestelmäpalvelut ovat nyt käyttäjätilassa, ei enää tehdä järjestelmäkutsuja samaan tapaan kuin monoliittisen kernelin kanssa, koska ne järjestelmäpalvelut ei enää sijaitse kernelissä!
- Tämän sijaan, tehdään sovellustason prosessien välisen kommunikoinnin (eli application IPC) avulla palvelupyyntöjä näille käyttäjätilan palveluprosesseille!

MIKROKERNEL

- Mikrokernelin etuna on se, että käyttöjärjestelmän palveluita on helppo laajentaa
  - On helppo tehdä lisää ohjelmia, joita ajetaan käyttäjätilassa, jos tarvitaan jotain uusia palveluita…
  - Ja näitä palveluita voidaan rakentaa huoletta lisää, koska ei tarvitse murehtia siitä, että uusi palvelu voisi kaataa koko järjestelmän, sillä yksittäiset palvelut ovat omassa muistiavaruudessaan suojattuna.

MIKROKERNEL

- Toki myös käyttöjärjestelmän siirtäminen toiselle suorittimelle voi olla yksinkertaisempaa, koska tarvitsee portata vain tuo pieni mikrokerneliosa. (Ainoa osa, mikä on laitteistorajapinnan kanssa tekemisissä).
- Myös vikasietoisempi
  - Vianeristys, koska kernelin osat ovat suojattuna toisiltaan
- Huonoja puolia: Koska kaikki järjestelmäpalvelut vaativat prosessien välistä kommunikointia, yksi palvelupyyntö voi vaatia monta viestiä prosessien välillä => tehokkuus riippuu viestinnän nopeudesta.

MUISTINHALLINTA JA REKISTERIT

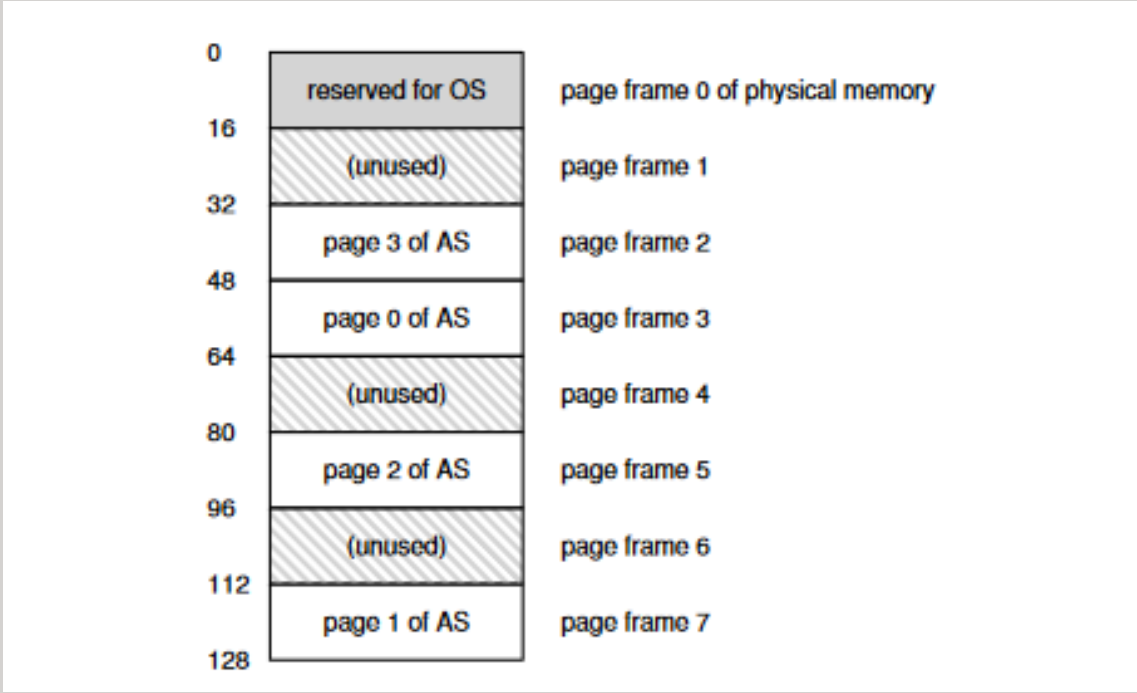
CT30A3370 - Käyttöjärjestelmät ja systeemiohjelmointi

MUISTINHALLINTA

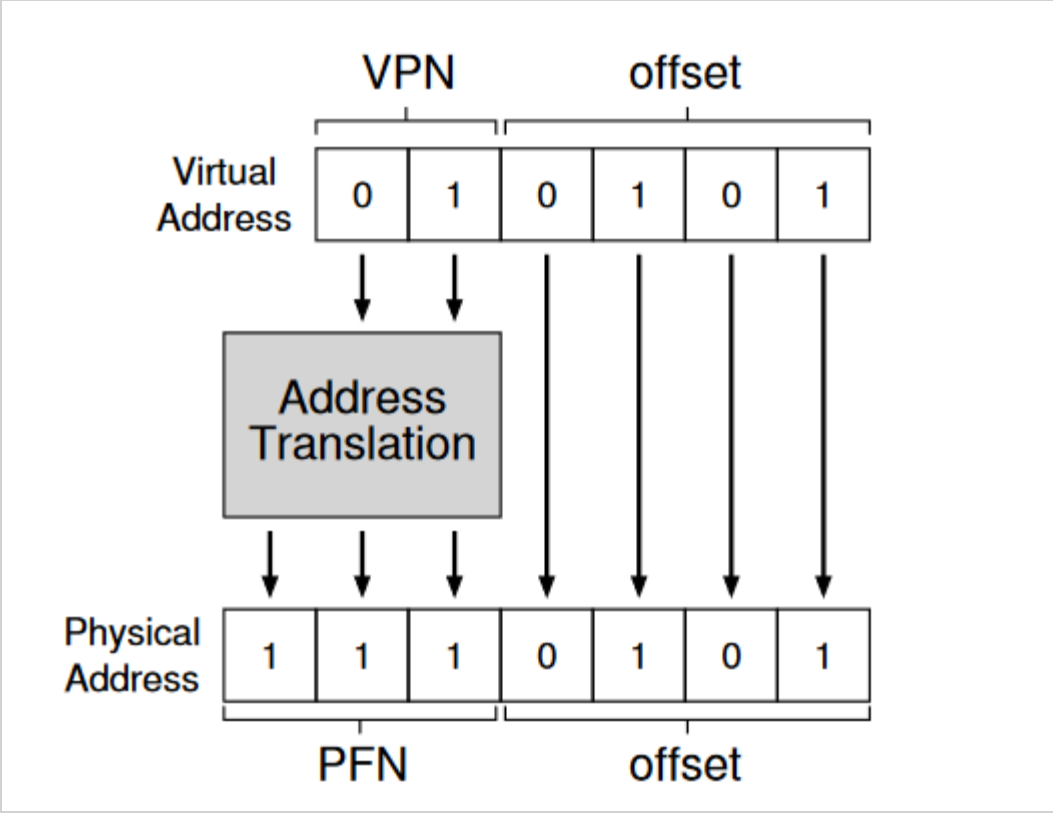
- Viime kerralla (ja äsken) mainittiin, että käyttöjärjestelmällä on muistiosoitteiden hallintaan MMU, ja sivutustaulu sekä muita asioita.
- Lisäksi, mistä kone esimerkiksi tietää, onko jokin asia muuttunut prosessin suorituksessa siten, että muutos pitäisi tallentaa jonnekin?
  - Aina kaiken kirjoittaminen ei ole tehokasta vaan päin vastoin: aina kun levyoperaatio voidaan jättää tekemättä se kannattaa jättää tekemättä.

MUISTIN SIVUTUS

- Eli ensimmäinen juttu oli muistin jakaminen tasakokoisiin paloihin, eli muistin sivutus.
- Käyttöjärjestelmä (ja vain käyttöjärjestelmä) tietää mikä sivu vastaa mitäkin fyysistä osoitetta.
- Prosessi näkee yhden jatkuvan muistialueen.



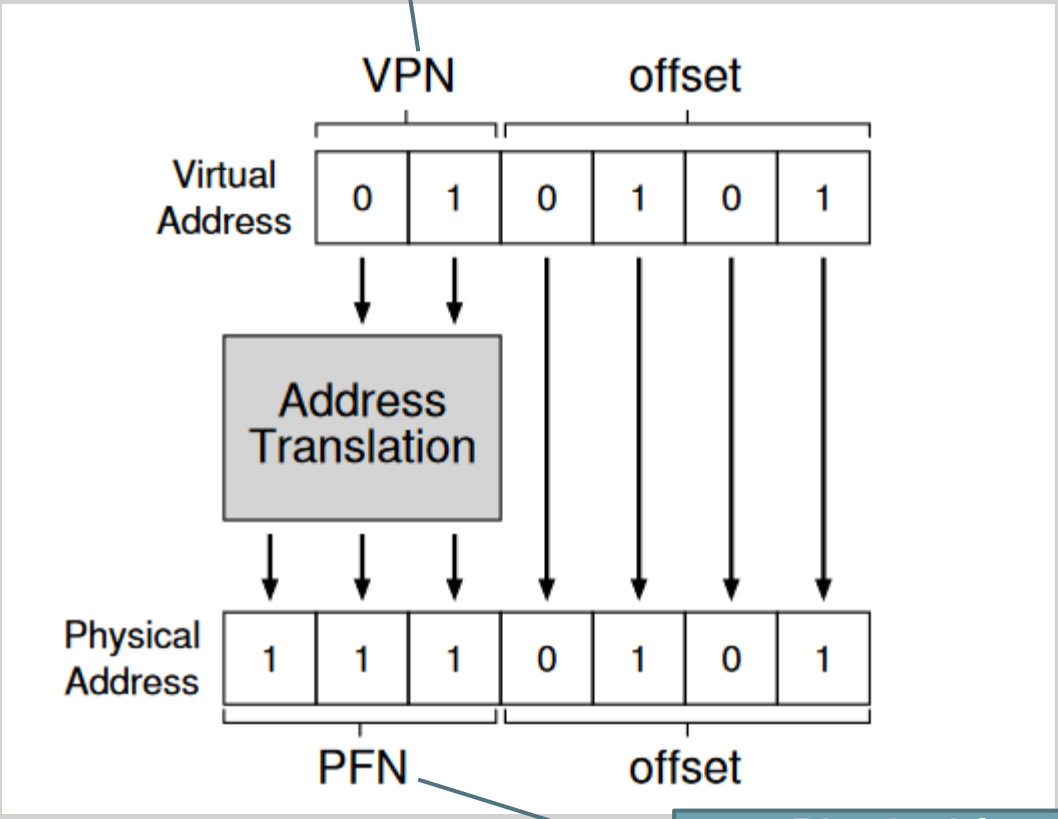
MMU JA OSOITEMUUNNOS



- Virtuaalisen osoitteen merkitsevät bitit menee muunnoksen läpi, tuloksena “oikea” osoite.
  - “Suuntanumero” vaihdetaan MMUn taulukon tietojen pohjalta.

MMU JA OSOITEMUUNNOS

Virtual page number, sivu jolla prosessi luulee olevansa.

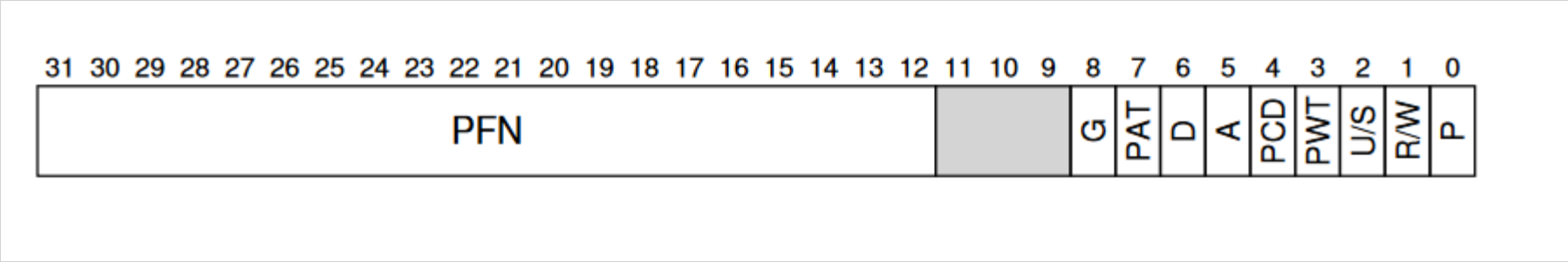


- Virtuaalisen osoitteen merkitsevät bitit menee muunnoksen läpi, tuloksena “oikea” osoite.
  - “Suuntanumero” vaihdetaan MMUn taulukon tietojen pohjalta.

Physical frame number, sivu/alue missä data oikeassa muistissa sijaitsee.

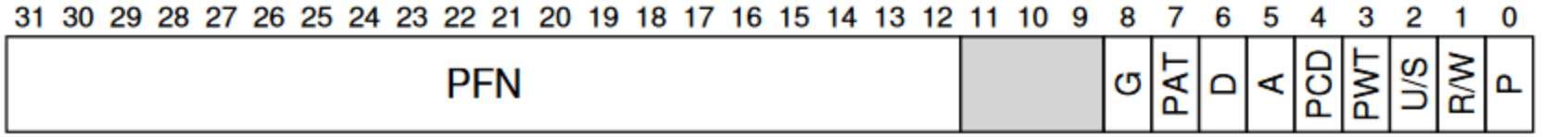
OSOITEMUUNNOSTAULUN SISÄLTÖ (X86-ESIMERKKI)

- Jos haetaan virtuaalisen sivunumeron (VPN, virtual page number) fyysistä sijaintia, mennään sivutauluun ko. sivunumeron kohdalle.
  - Tämäkin on yksinkertaistus; sivunumerointi ei välttämättä ole jono, se voi olla myös jotain muuta tehokkuuden lisäämiseksi. Sovitaan nyt että se on jono.
- Sieltä voi löytyä vaikkapa tässä formaatissa oleva tietue:



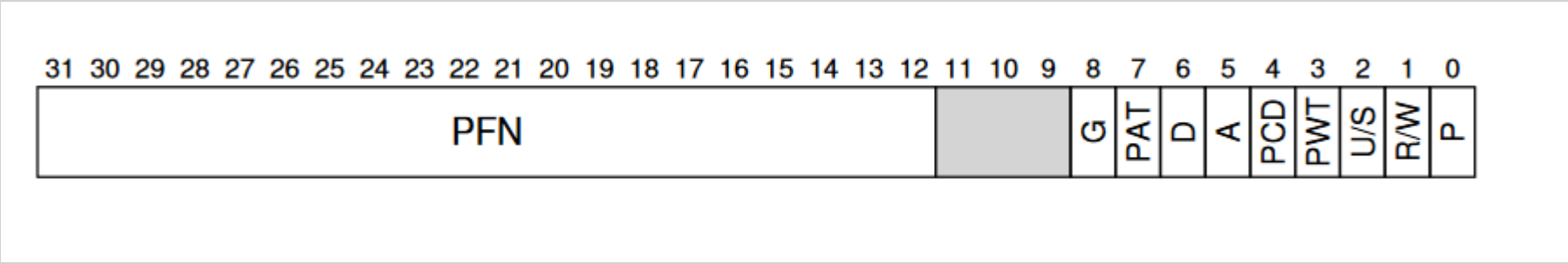
OSOITEMUUNNOSTAULUN SISÄLTÖ (X86-ESIMERKKI)

- PFN (physical frame number), eli fyysisen muistin osoite.
- Lisäksi aputietoa, josta voidaan päätellä, mitä tälle sivulle pitää tehdä silloin, kun prosessia vaihdetaan ja muistissa olevia asioita säädetään.



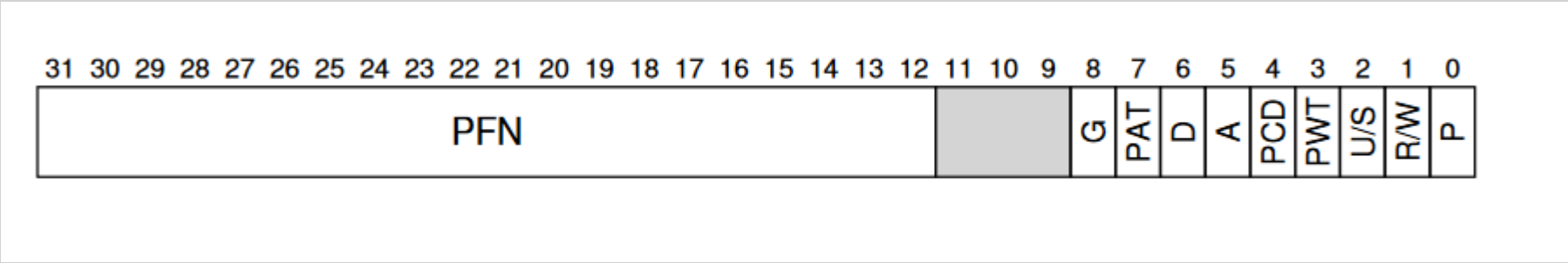
OSOITEMUUNNOSTAULUN SISÄLTÖ (X86-ESIMERKKI)

- P (present) kertoo onko tämä frame muistissa vai levylle swapattuna.
- R/W (read/write) saako tähän muistialueesen kirjoittaa.
- U/S (user/supervisor) saako tätä muistialuetta käyttää user-oikeuksilla.
- A (accessed) kertoo onko tähän muistialueeseen viitattu kertaakaan.
- D (dirty) onko muistialuetta muokattu (eli pitääkö se kirjoittaa takaisin levylle)
- G, PCD, PAT ja PWT –bitit säätelee kuinka tätä tietoa voidaan käyttää muissa rekistereissä ja raudan ohjauksessa.



OSOITEMUUNNOSTAULUN SISÄLTÖ (X86-ESIMERKKI)

- Ongelmaksi tietysti tulee se, että muistitaulusta tulee iso.
  - Ja ison taulun läpikäynti on hidasta.
  - …Eli otetaan useimmiten käytetyt muunnokset talteen erilliseen “pikavalintaan”!

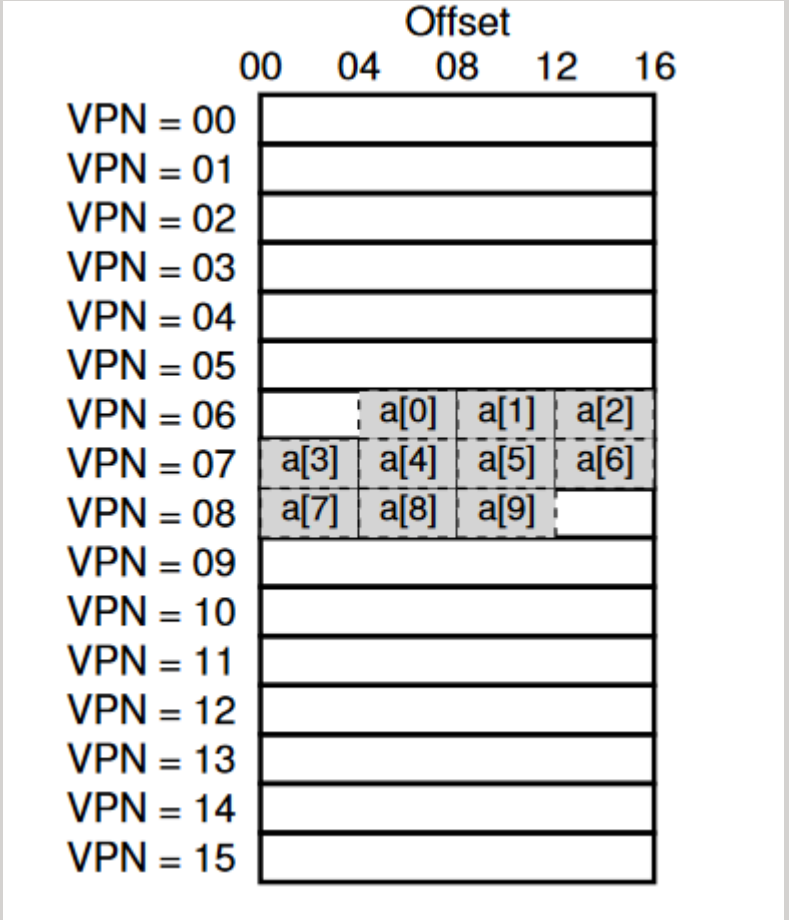


TRANSLATION-LOOKASIDE BUFFER (TLB)

- Assosiatiivinen välimuisti, “muistimuisti” jne…
- Taulukko johon on listattu viimeisimpiä käytettyjä muistialueita puskuriin, josta ne on nopeampi hakea kun isosta taulukosta.
  - Koko kymmenistä n. sataan osoitemuunnokseen.
  - Ei voi olla suuri, että on nopea käyttää; normaalisti prosessit käyttää toistuvasti samoja asioita joten on helpompi välillä korjata tätä puskuria kuin aina hakea uudelleen “isosta kirjasta”.
- Ensimmäinen prosessoriarkkitehtuurillinen merkittävä ero!
  - CISC: rautatason TLB, fyysinen piirisarja levyllä.
  - RISC: softatason (käyttiksen toteuttama) TLB

TRANSLATION-LOOKASIDE BUFFER (TLB)

- Oheista taulukkoa manipuloidessa riittää jos meillä on kolme pikavalintaa.
  - Kun lista käydään läpi, tehdään kolme hidasta hakua TLB “miss”:n takia.
  - Ja 7 nopeaa hakua TLB “hitin” ansiosta.



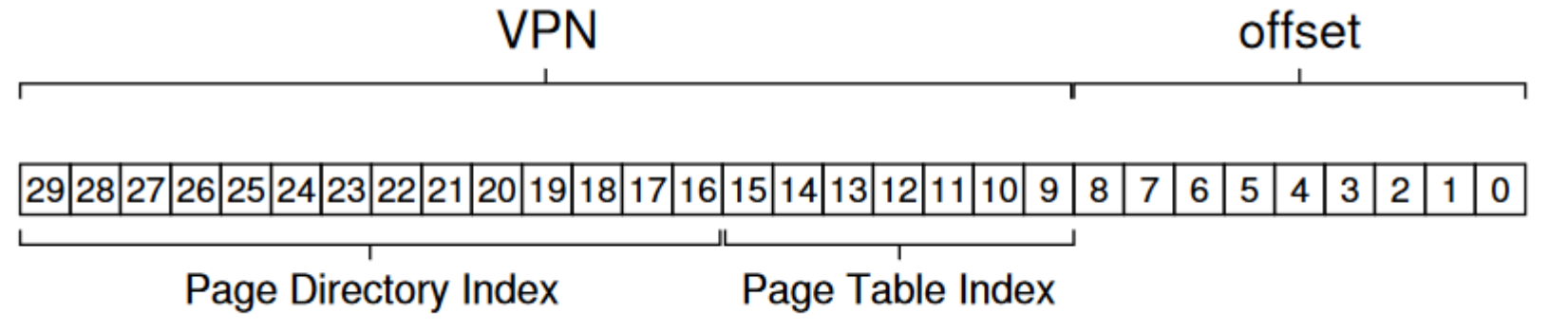
TRANSLATION-LOOKASIDE BUFFER (TLB)

- Periaatteessa lista jossa kerrotaan
  - Virtuaaliosoite ja sen vastinpari fyysinen osoite
  - Onko tämä käännös edelleen OK (nopeampi merkata “saa poistaa” kuin alkaa pyyhkiä yli, joku sen kuitenkin korvaa myöhemmin.)
  - Käyttöoikeudet
  - Omistava prosessi (ASID = address space ID)
    - Toinen vaihtoehto olisi tyhjentää puskuri aina kun suoritettava prosessi vaihtuu mutta tämä on nopeampi ratkaisu.

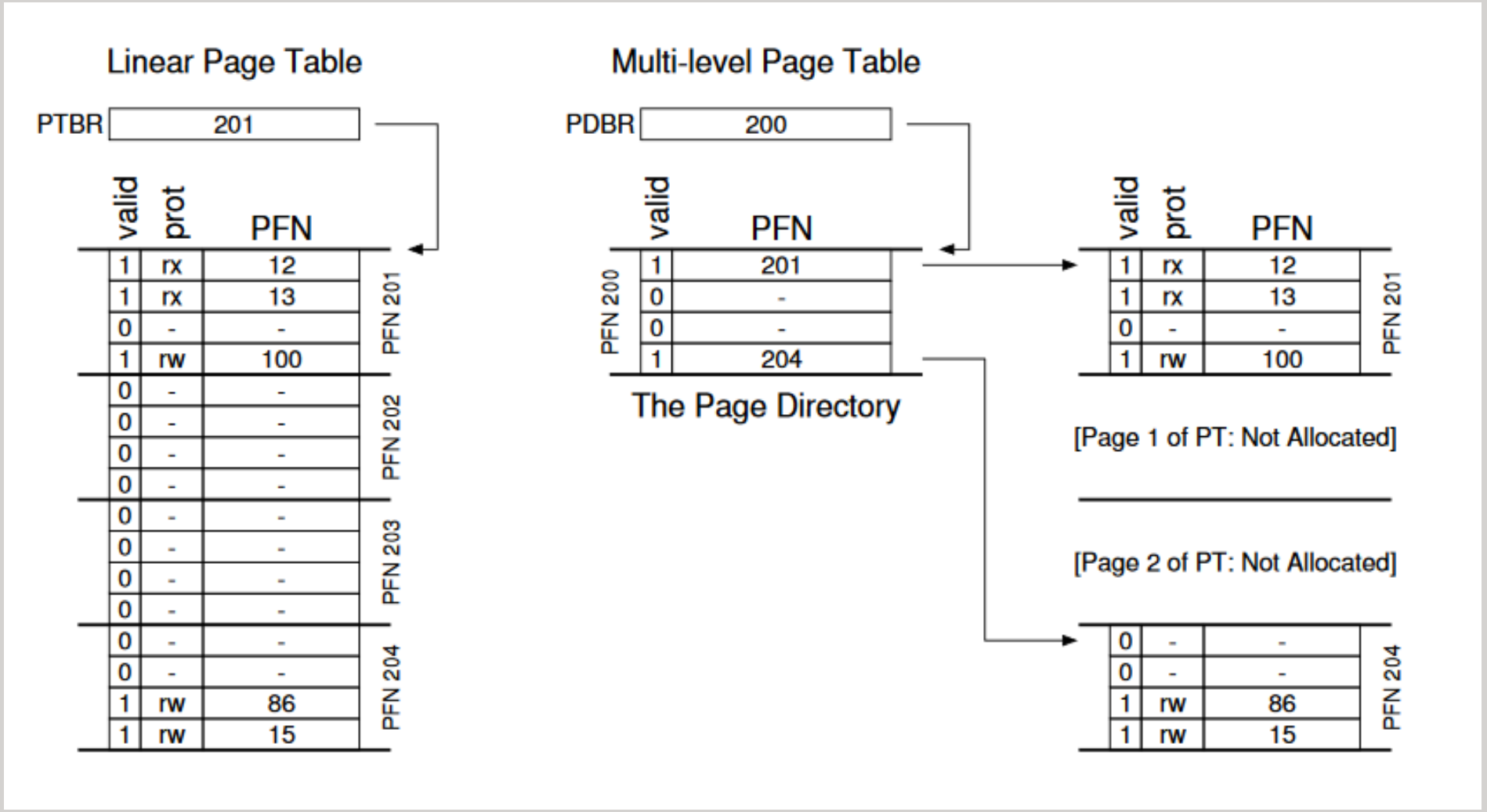
VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
—	—	0	—	—
10	170	1	rwx	2
—	—	0	—	—

MONITASOINEN SIVUTAULU

- TLBn kaveriksi on myös kehitelty muita tehostamistoimia; yksi esimerkiksi on monitasoinen sivutaulu.
  - Idea: ei käytetä kaikkia sivunumeroita kerralla, vaan jaetaan niitä ryppäissä.
  - Kaksi hakua lyhyestä indeksistä on nopeampi kuin yksi haku pitkästä.

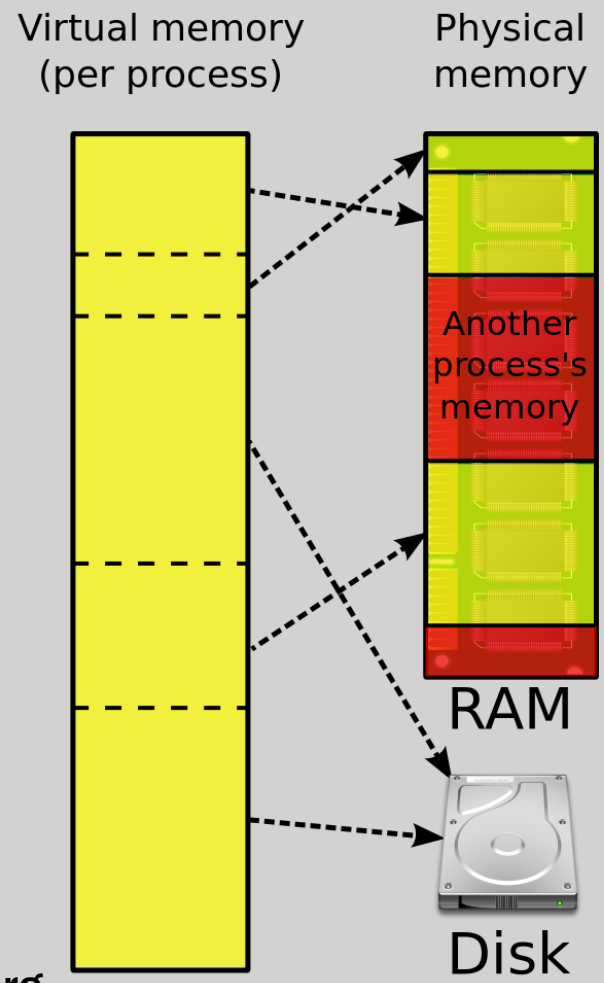


MONITASOINEN SIVUTAULU



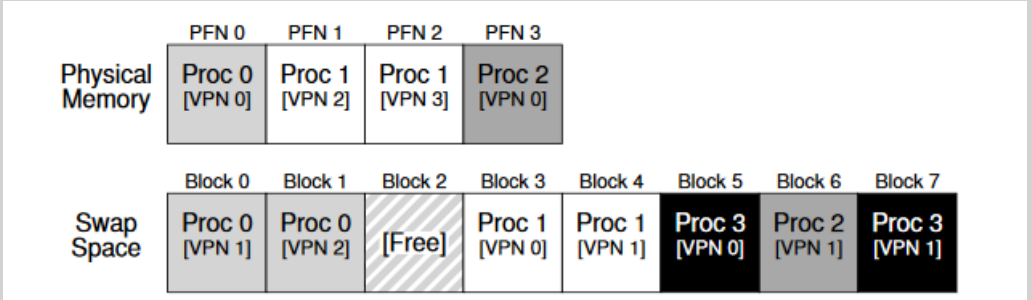
SWAPPING, HEITTOVAIHTO JA VIRTUAALIMUISTI

- Aiemmin jo mainittiin, joskus muistia pitää siirtää levylle ja sieltä pois.
  - Voidaan käsitellä muistin määrää suurempia asioita.
  - Kaiken ei tarvitse aina olla muistissa odottamassa oman prosessin ajovuoroa.
- Kokonaisuudesta puhutaan näennäismuistina, tai virtuaalimuistina.

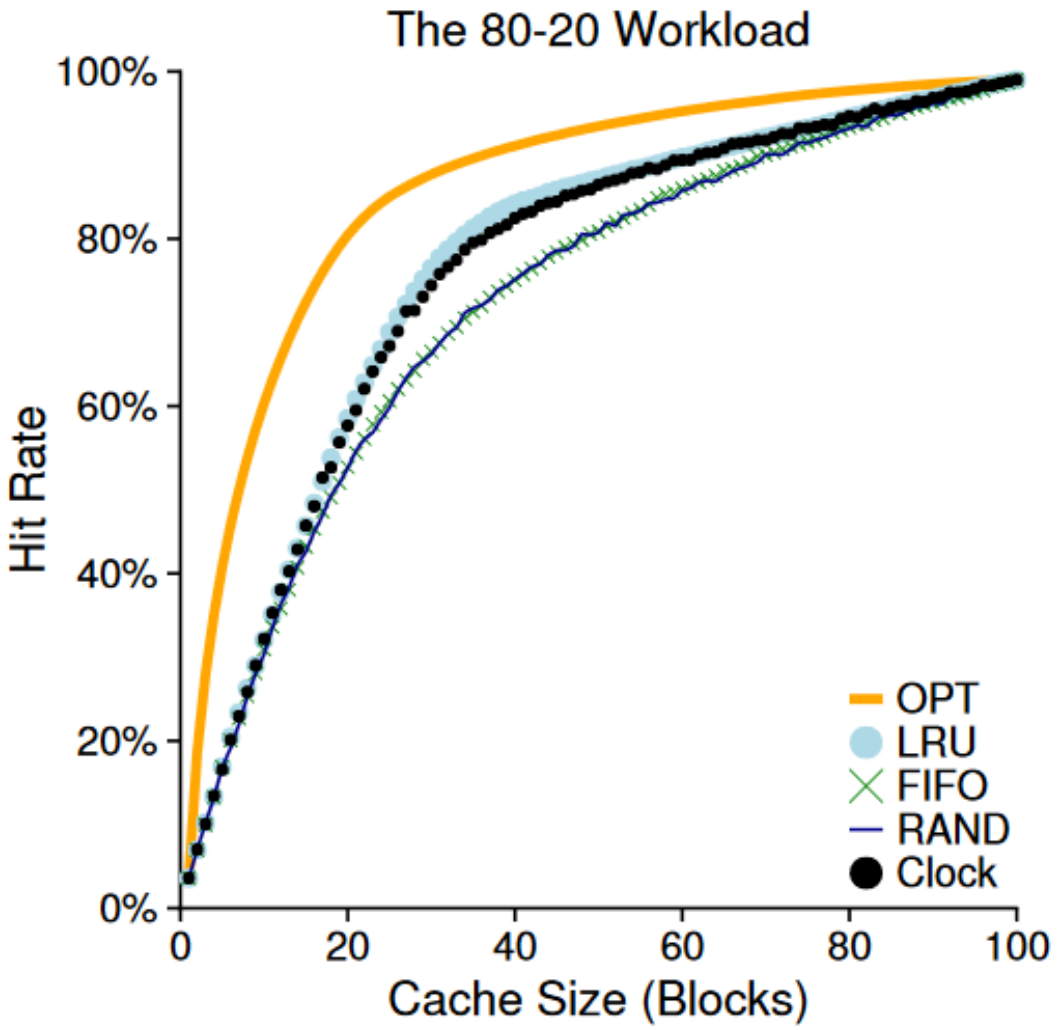


SWAPPING, HEITTOVAIHTO

- Jos tulee TLB miss (tämä osoite ei ole ollut käytössä), tai present bit on 0, tulee page fault.
- Siirretään blokki muistiin, päivitetään TLBtä, korjataan TLB osoittamaan että kaikki on OK ja jatketaan matkaa.
  - Ja kengitään TLBstä jotain pihalle sen pohjalta, mitä algoritmiä käytetään.
    - Ei-viitatut ensin, kauimmin sitten viitatut, siirtyvän osoittimen osoittama jne. Ei yhtä oikeaa tapaa.
    - Ja tietysti jos dirty bit on 1 niin joudutaan tekemään tarvittavat kirjoitukset ja muutokset että systeemi ei hukkaa työtä.



PAGE FAULT, KORVAUSTAPOJA



MITÄ TÄSTÄ LUENNOSTA PITÄÄ MUISTAA?

- Erilaiset kernel-ratkaisut
- Muistinhallinta
  - Perusperiaate sivutustaululle
  - Mikä on TLB

LUT
University