
NOTATER UKE 10 –

Multitasking, cache, hyperthreading

Multitasking foregår når flere enn et program bytter på samme CPU. Det er da SMP (simultanious multi processing). Da er det fair scheduling, som i at linux scheduleren (den som fordeler hvilke prosesser som kjører hvor), den forsøker å gi så lik tid til alle som mulig. Det koster en del å bytte fra en CPU til en annen, så de bytter en gang her og der i blant.

En regneenhet er en ALU, det kan være en ALU per core, hver enhet. Multithreading handler om å dele den ene ALU-en.

Kommandoen «lscpu» viser informasjon om CPU-arkitektur og maskinvarekonfigurasjon. Når du kjører kommandoen i terminalen, vil du få utskrift som viser detaljer om prosessorens arkitektur, antall prosessorer, antall kjerner per prosessor, hastighet, cache-størrelse, instruksjonssett og annen relatert informasjon.

Noen eksempler på informasjon som kan vises ved bruk av lscpu-kommandoen er:

CPU-arkitektur (f.eks. x86_64)

Antall prosessorer

Antall kjerner per prosessor

Prosessorens hastighet (f.eks. i GHz)

Cache-størrelser for L1, L2 og L3-cacher

Instruksjonssett som støttes av prosessoren (f.eks. SSE, AVX)

Internminne referer til en datamaskins primære lagringsplass som brukes til å kjøre applikasjoner og lagre data midlertidig mens de behandles. Dette inkluderer både RAM (random access memory) og ROM (read-only memory).

RAM – type flyktig minne som brukes til å lagre data og instruksjoner som brukes av datamaskinen mens den kjører. Den er raskere enn andre typer lagringsenheter, men den er også midlertidig og mister all informasjonen når strømmen slås av. Kapasiteten kan utvides ved å installere flere RAM-brikker.

ROM - er en type permanent lagring som brukes til å lagre data og instruksjoner som trengs av datamaskinen under oppstart. Den er saktere enn RAM, men er også permanent og beholder informasjonen selv når strømmen slås av. ROM inkluderer også bios (Basic I/O System) som brukes av datamaskinen for å starte opp og kommunisere med forskjellige enheter.

I tillegg til RAM og ROM kan en datamaskin også ha andre former for internminne, som for eksempel hurtigbuffer eller cacheminne som brukes til å akselerere dataoverføringer mellom prosessor og RAM.

CACHE

CPU utfører instruksjoner raskere enn data kan hentes fra RAM. Cache er hurtigere og mellomlagrer både instruksjoner og data. Ordet 'cache' er fransk og betyr hemmelig lager. Cache er hurtigRAM. Vi legger egentlig på et digert lager med enda flere registere. Men de er bare mellomlagring av data på vei inn til CPU-en, og på vei ut.

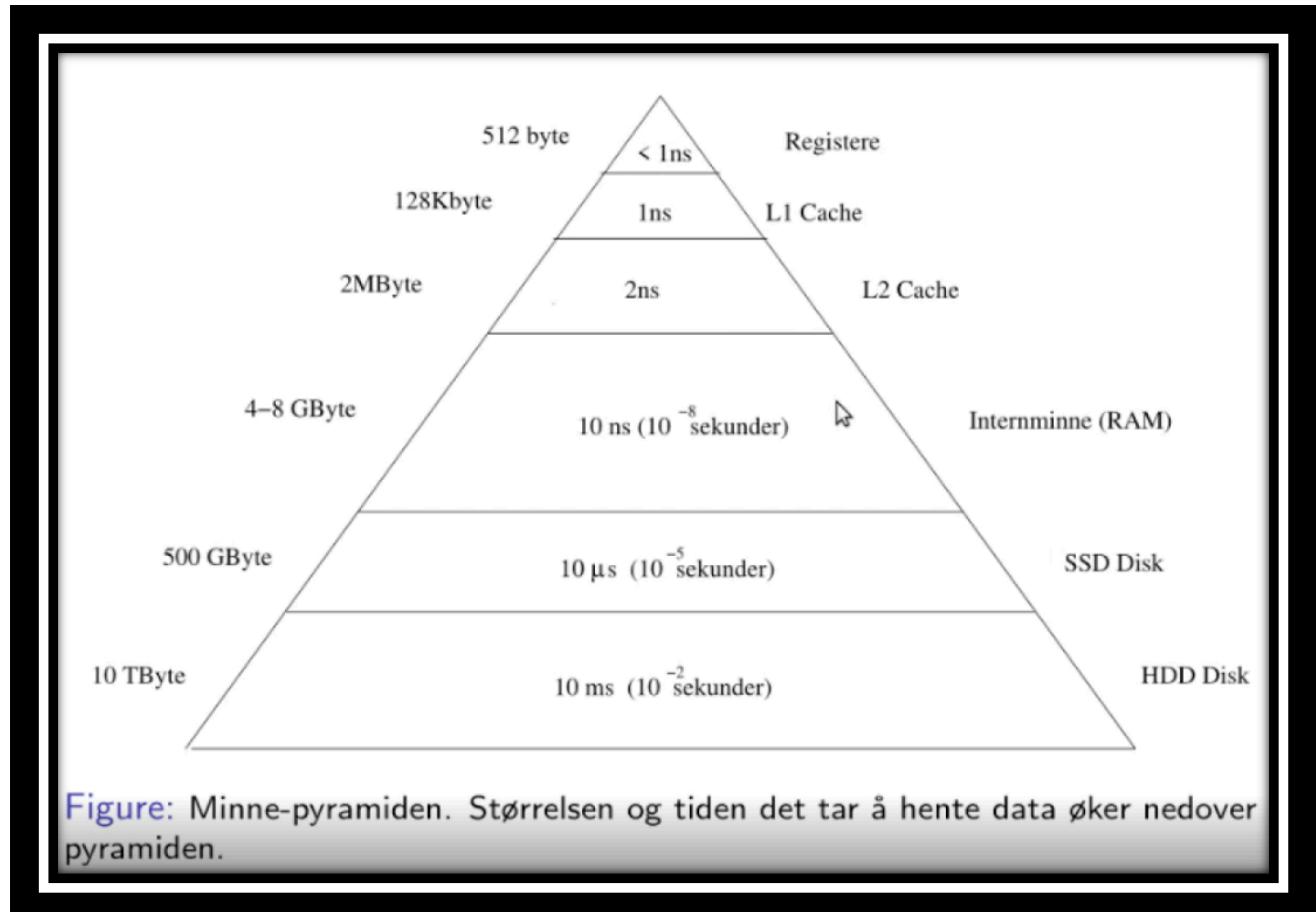
Data som brukes i RAM ligger ofte ved siden av hverandre (eks array). Vi sparer veldig mye tid på å hente mange byte av gangen og mellomlagre i cache.

MINNEPYRAMIDEN

Vi har forskjellige enheter som lagrer data, og er koblet opp til CPU, og her er det veldig stor forskjell på hvor lang tid det tar å hente data. Vi har registrene i toppen og det bruker kortest tid på å hente data til ALU-en, fordi det er jo registrene som er koblet med ALU-en, og her går det lynraskt (kan være under et nanosekund).

Også har man en del cache (L1, L2 og moderne datamaskiner har L3 cache også).

Teknologien er det samme, disse er alle SRAM.



HHD OG SSD

HHD (hard disk drive) er en mekanisk enhet som bruker roterende skiver og lese/skrivehoder til å lagre og hente data. HHD-enheter er vanligvis større og har høyere lagringskapasitet enn SSD-enheter, men er også tregere i hastighet og kan være mer utsatt for feil pga deres mekaniske natur.

SSD (solid state drive) er en enhet uten bevegelige deler, og bruker flashminne til å lagre data. SSD-enheter er vanligvis mindre og har lavere lagringskapasitet enn HHD-enheter, men tilbyr betydeligere raskere dataoverføringshastigheter og bedre pålitelighet på grunn av deres solid-state design. SSD er også vanligvis dyrere enn HHD på grunn av sin høyere ytelse og pålitelighet.

SRAM OG DRAM

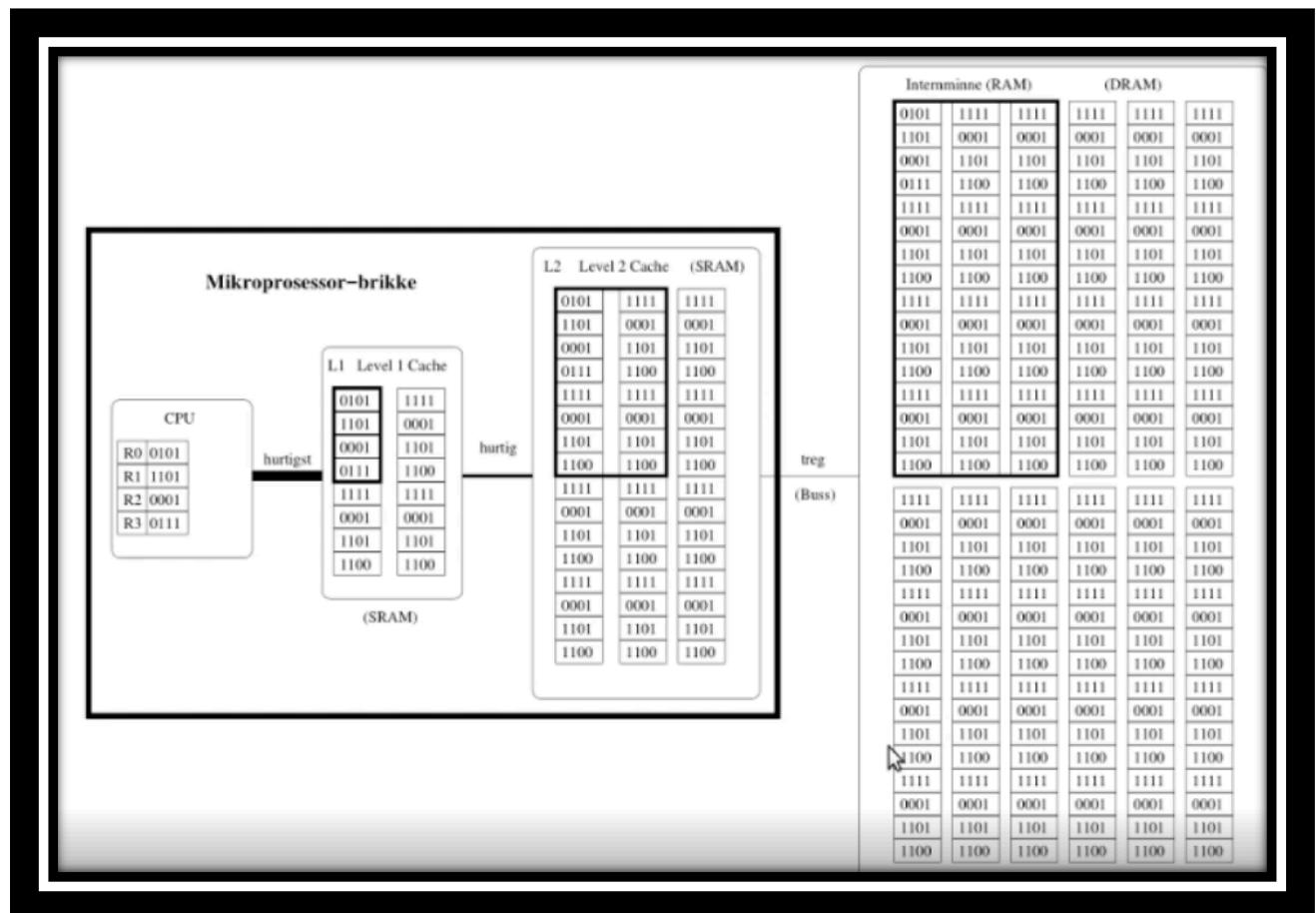
CPU-registere og cache er laget av SRAM (static RAM). SRAM består av 6 transistorer for hver bit som lagres. DRAM er en enklere teknologi og består kun av en transistor og en kapasitator (lagrer elektrisk ladning). Hvis kapasitatoren har ladning, er den 1, hvis ikke er

den 0. DRAM, er mindre og billigere, er ikke like hurtig og må lades på nytt 10 ganger i sekundet. Alt som er lagret i DRAM forsvinner når man skrur av strømmen. Så når du skrur på en datamaskin så står DRAM og lades opp hele tiden for at vi skal kunne beholde 0 eller 1, hvorav 0 er enklere å beholde, mens 1 må lades opp hele tiden.

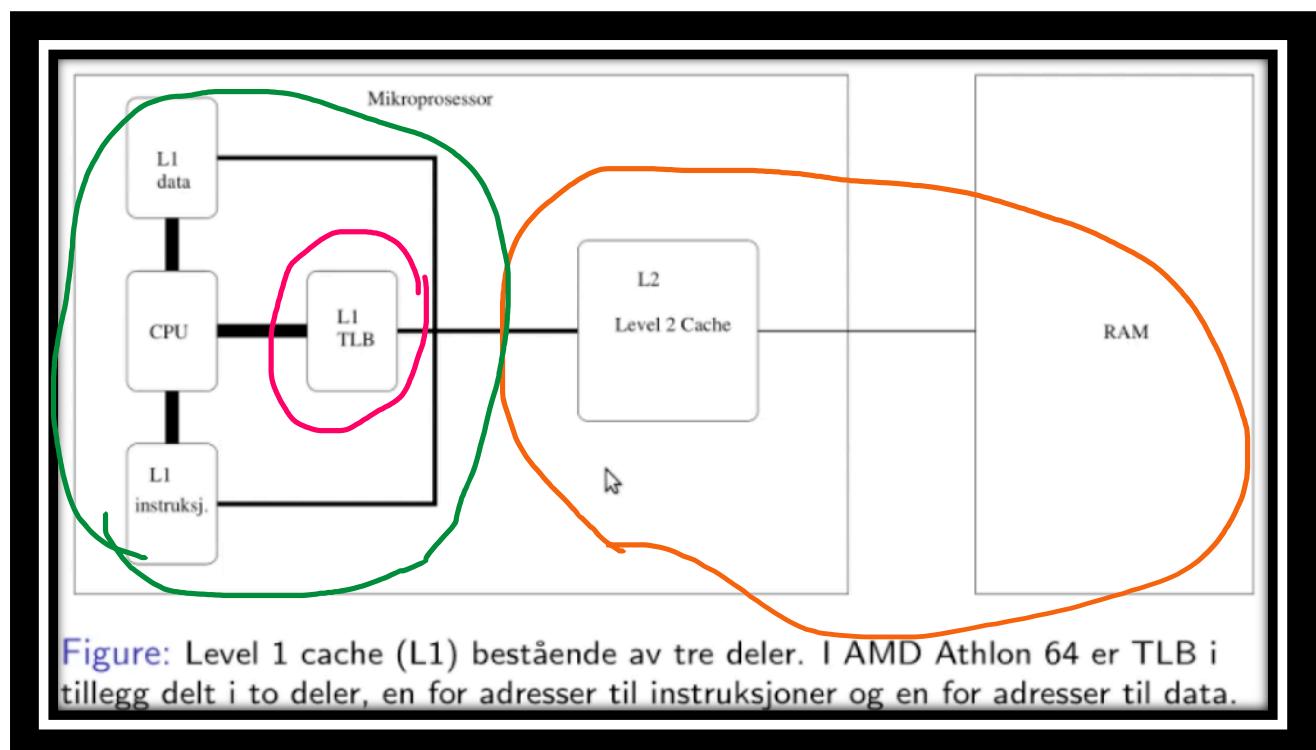
L1 OG L2 CACHE

Denne figuren viser prinsippene for cache og hvorfor man kan få til å kjøre programmet forttere ved å bruke cache. Helt til venstre ser vi CPU-en med registrene: R0, R1 osv. og vi kunne også ha satt inn ax, bx, cx osv. Disse brukes hele tiden for å mates inn i ALU-en. Denne CPU-en kan kverne instruksjoner ekstremt mye raskere enn RAM kan levere det. Derfor legger vi enn L1 og L2 cache (og evt annet).

Caching er jo noe som styres på hardware nivå, dette er ikke noe som operativsystemet går inn og styrer, eller håndterer.



Under vises mikroarkitekturen for L1 og L2 cache:



Man kan si at den oransje sirkelen følger Von Neuman arkitekturen hvor både datainstruksjoner foregår på bussen. Den siste biten følger Harward arkitekturen hvor man deler opp og instruksjoner kommer på en buss inn til CPU-en, og data ut av en annen. Vi har en tredje bit, rosa, **translation look aside buffer**, som er minneaddressering. Det er en type hurtigbuffer som brukes i datamaskiner og andre digitale enheter for å akselerere oversettelsen av virtuelle adresser til fysiske adresser i hovedminnet. Når er CPU leser eller skriver til en virtuell adresse, må oversettelsen fra den virtuelle adressen til den fysiske adressen finne sted. TBL lagrer en oversettelsestabell som inneholder oversikt over nylig brukte oversettelser for å unngå å måtte utføre denne oversettelsen hver gang. TBL brukes i moderne datamaskiner for å øke effektiviteten til virtuell minnehåndtering.

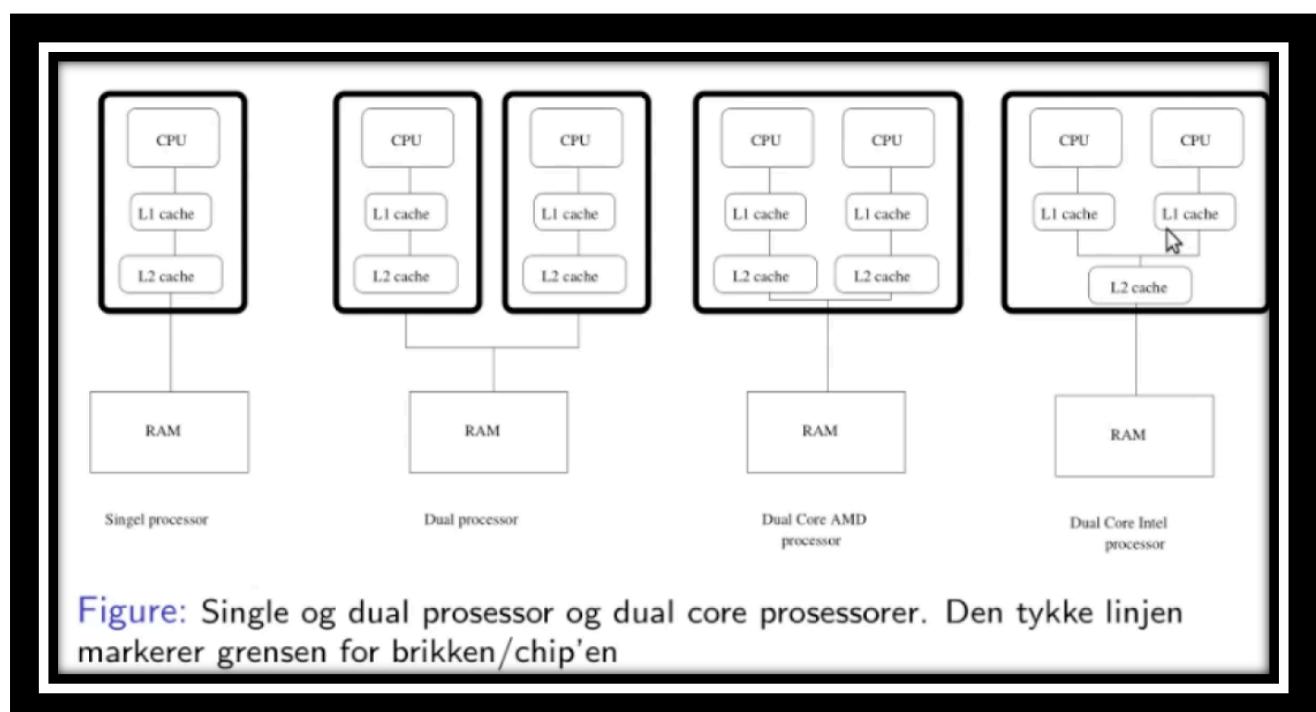
Hvorfor er ikke Real = User + System for time-kommandoen?

SVAR: det er riktig når CPU brukes 100% ikke mindre.

MULTITASKING OG MULTIPROCESSING

- **Multitasking** er det vi har sett nå når operativsystemet (Software) brukes til å fordele tid fra samme CPU mellom flere prosesser.

- **Multiprocessing** når to eller flere CPU'er i samme computersystem kjører flere prosesser virkelig samtidig, på samme tidspunkt, de må ikke dele samme internminne.
- **Symmetric Multiprocessing SMP**, to eller prosessorer deler samme internminne og kjører flere prosesser virkelig samtidig, på samme tidspunkt. Dette er det vi har jobbet med, alle registrene er koblet til samme internminne og jobber parallelt.
- **Multi Core Multiprocessing** to eller flere prosessorer på samme brikke (kjerne/enhet) deler cache og databus og kjører flere prosesser samtidig, regnes også som SMP.



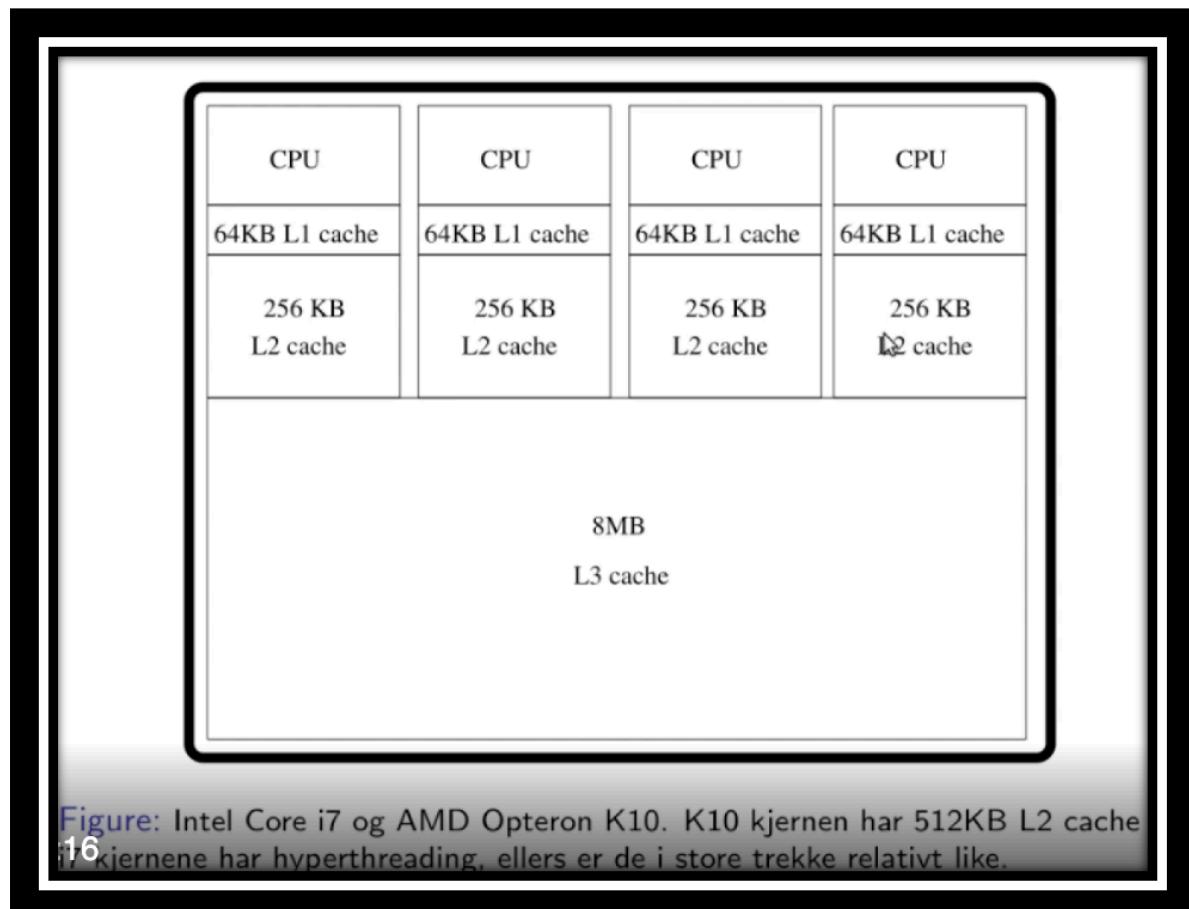
Hvilken av de som er best, kan variere, men i nr 3 (fra venstre) er det også vanlig å legge på en L3 som felles. Hvis det er mange prosesser som har mye til felles, da er det bedre med felles cache.

Under er det bildet av en 4 core CPU: L1 og L2 cache er begge små, raske minneenheter som er plassert nærmest prosessoren i datamaskinen. De har som hovedoppgave å redusere tiden det tar for prosessoren å få tilgang til data og instruksjoner som trengs for å utføre oppgaver.

L1-cache er den raskeste og minste typen cache, og ligger direkte på prosessoren. Den inneholder de mest kritiske dataene og instruksjonene som prosessoren trenger for å utføre oppgaver. L2 er større og ligger på prosessorens kjerne.

L3-cache er større enn både L1 og L2-cache og er vanligvis delt mellom flere prosessorkjerner. Det er ment å være en delt ressurs for hele systemet, og inneholder data og instruksjoner som ikke nødvendigvis er like kritiske som de som er lagret i L1 og L2. L3 fungerer som et mellomlagringssted for data og instruksjoner som brukes av flere prosessorkjerner samtidig, og bidrar til å redusere belastningen på hovedminnet og øke systemets ytelse.

Så, å ha L1 og L2 cache nær prosessoren bidrar til å redusere forsinkelsen i å hente data og instruksjoner fra hovedminnet, mens L3-cache fungerer som en delt ressurs for hele systemet, og bidrar til å øke effektiviteten og ytelsen. Samlet sett er bruk av flere nivåer av cache en viktig del av å optimalisere ytelsen til datamaskiner og systemer.



Mikroarkitektur

Vi har sett at hardware definerer et bestemt instruksjonssett og sett på noen av instruksjonene som er definert i X86-instruksjonssettet. Men i praksis finnes det mange måter å fysisk implementere et gitt instruksjonssett. Måten en produsent av

CPU-er, som Intel eller AMD, implementerer et instruksjonssett kalles en mikroarkitektur. Forskjeller i for eksempel hvordan mikro-operasjoner utføres og hvordan pipelining eller cache-nivåer er lagt opp utgjør forskjeller i mikroarkitekturen. Man kan si at datamaskinarkitekturen utgjør kombinasjonen av instruksjonssettet og mikroarkitekturen.

Serveren amdock er den serveren som Linux VM-ene våre kjører. Lister docker containere:

```
root@amdock:~/hh/pubnet# docker ps
```

LSCPU gir info om CPU-en.

```
root@amdock:~/hh/pubnet# lscpu
```

Under står modellnavnet, og instruksjonssettet er x86:

```
Core(s) per socket:      48
Socket(s):              1
NUMA node(s):           1
Vendor ID:              AuthenticAMD
CPU family:             23
Model:                  49
Model name:             AMD EPYC 7552 48-Core Processor
Stepping:                0
CPU MHz:                3260.517
BogoMIPS:               4391.65
Virtualization:         AMD-V
```

L1d er datacache og L1i er instruksjonscache:

```
BogoMIPS:               4391.65
Virtualization:          AMD-V
L1d cache:              1.5 MiB
L1i cache:              1.5 MiB
L2 cache:                24 MiB
```

HYPERTHEADING – Intel's varemerke threading

Hyperthreading (også kjent som simultaneous multithreading) er en teknologi som lar en enkelt fysisk prosessor kjerne behandler flere tråder samtidig ved å dele opp den fysiske prosessoren i virtuelle prosessorer, kalt «logiske prosessorer». Dette kan gi en betydelig

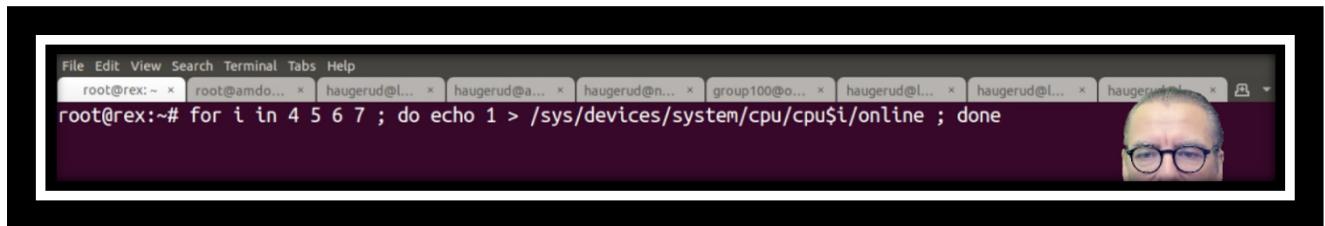
økning i systemets ytelse, spesielt for applikasjoner som bruker mye av prosessoren som video- og bildebehandling, databehandling, og virtuelle maskiner. Det er viktig å merke seg at hyperthreading ikke gir en dobling av prosessorens ytelse, men kan gi en økning i effektiviteten til systemet hvis det implementeres riktig og brukes med riktig programvare og maskinvare.

En single core CPU kan inneholde to prosessorer samtidig: (operativsystemet opplever dette som to selvstendige prosessorer). Hardware switcher selv mellom de to prosessene i løpet av nanosekunder.

Det som faktisk skjer når operativsystemet fordeler prosesser til denne prosessoren som er hyperthreading, så settes det i gang to prosessorer på samme CPU, men de to prosessoren deler ALU-en som er på denne CPU-en. Og da er det hardware som switcher selv mellom de to prosessene i løpet av nanosekunder, ekstremt hurtig, og dette er ikke i nærheten av det som skjer når man gjør context switch med operativsystemet, det tar veldig mye mer tid.

Hyperthreading på Linux-desktop rex med 8 CPU-er (eller 4?)

Kommandoen "echo 1 > /sys/devices/system/cpu/cpu\$1/online" vil sette den angitte CPU-kjernen (nummerert med \$1) på linjen til online tilstand i Linux-systemet, slik at den blir tilgjengelig for bruk. Dette kan være nyttig i tilfeller der en CPU-kjerne er blitt deaktivert og man ønsker å aktivere den igjen. Merk at denne kommandoen bør brukes med forsiktighet, og at det kan være potensielle risikoer og konsekvenser ved å endre systemets CPU-konfigurasjon.



```
File Edit View Search Terminal Tabs Help
root@rex: ~ | root@amdo... | haugerud@L... | haugerud@a... | haugerud@n... | group100@o... | haugerud@l... | haugerud@l... | haugerud@l...
root@rex:~# for i in 4 5 6 7 ; do echo 1 > /sys/devices/system/cpu/cpu$i/online ; done
```

HTOP

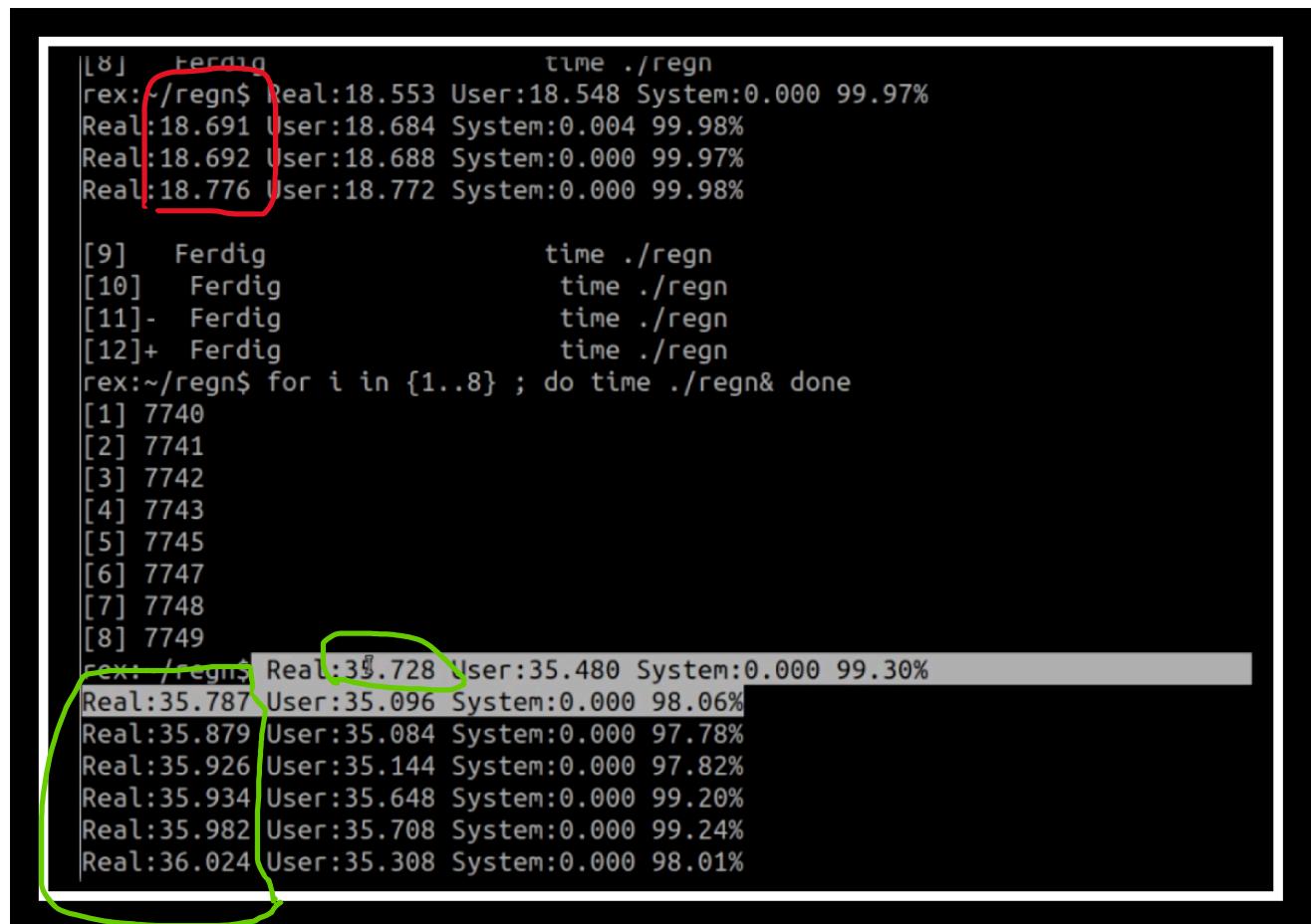
Er en interaktiv systemovervåkingsverktøy for Linux-operativsystemet som gir en oversikt over ressursbruk og systemytelse på en enkel og intuitiv måte. Htop gir en visuell fremstilling av systemets ressursbruk, inkludert CPU- og minnebruk, antall kjørende prosesser og deres tilhørende ressursbruk, og systemets belastning.

Noen av funksjonene til htop inkluderer:

- Visning av prosesser sortert etter ressursbruk, inkludert CPU-bruk og minnebruk.
- Interaktiv navigering, som gjør det mulig å endre sorteringsrekkefølgen på kolonnene ved å klikke på kolonnen.
- Mulighet til å drepe prosesser direkte fra grensesnittet.
- Rask og effektiv oppdatering av systeminformasjon, som gjør at man kan holde oversikt over systemets ytelse i sanntid.
- Mulighet til å bruke fargekoding for å gi en rask oversikt over ressursbruk.

8 CPU-ere i servere:

Under i rød kan man se at det tar 18,5 sek for 4 regn script. I grønn kan man se at det nesten tar dobbelt så lang tid med 8 regn script som vil si at det egentlig er fire CPU, med to core per socket.



```
[8] Ferdig          time ./regn
rex:~/regn$ Real:18.553 User:18.548 System:0.000 99.97%
Real:18.691 User:18.684 System:0.004 99.98%
Real:18.692 User:18.688 System:0.000 99.97%
Real:18.776 User:18.772 System:0.000 99.98%

[9] Ferdig          time ./regn
[10] Ferdig          time ./regn
[11]- Ferdig          time ./regn
[12]+ Ferdig          time ./regn
rex:~/regn$ for i in {1..8} ; do time ./regn& done
[1] 7740
[2] 7741
[3] 7742
[4] 7743
[5] 7745
[6] 7747
[7] 7748
[8] 7749
rex:~/regn$ Real:35.728 User:35.480 System:0.000 99.30%
Real:35.787 User:35.096 System:0.000 98.06%
Real:35.879 User:35.084 System:0.000 97.78%
Real:35.926 User:35.144 System:0.000 97.82%
Real:35.934 User:35.648 System:0.000 99.20%
Real:35.982 User:35.708 System:0.000 99.24%
Real:36.024 User:35.308 System:0.000 98.01%
```

I koden han kjører hvor masse data hentes fra RAM, kan man se hvordan hyperthreading er meget effektiv.

TASKSET

Kan vi fordele hvilken oppgave som skal kjøres på hvilken CPU. Under sier kommandoen at regn skal kjøres på CPU 0.

```
haugerud@rex:~/home/haugerud/regn 76x28
rex:~/regn$ time taskset -c 0 ./regn
```

grep «» /sys/devices/system/cpu/cpu*/topology/thread_siblings_list

```
haugerud@rex:~/home/haugerud/regn 86x28
rex:~/regn$ grep "" /sys/devices/system/cpu/cpu*/topology/thread_siblings_list
/sys/devices/system/cpu/cpu0/topology/thread_siblings_list:0,4
/sys/devices/system/cpu/cpu1/topology/thread_siblings_list:1,5
/sys/devices/system/cpu/cpu2/topology/thread_siblings_list:2,6
/sys/devices/system/cpu/cpu3/topology/thread_siblings_list:3,7
/sys/devices/system/cpu/cpu4/topology/thread_siblings_list:0,4
/sys/devices/system/cpu/cpu5/topology/thread_siblings_list:1,5
/sys/devices/system/cpu/cpu6/topology/thread_siblings_list:2,6
/sys/devices/system/cpu/cpu7/topology/thread_siblings_list:3,7
[1]- Avsluttet 255          time taskset -c $i ./a.out
[2]+ Avsluttet 255          time taskset -c $i ./a.out
```

LINUX

Docker er en plattform for å utvikle, levere og kjøre applikasjoner ved hjelp av containerteknologi. Docker er basert på en «containerization»-teknologi som lar utviklere pakke en applikasjon sammen med dens avhengigheter og kjøre den som en isolert enhet, kalt en «container». En container inneholder alt som trengs for at en applikasjon skal kjøre, inkludert kode, runtime, systemverktøy og biblioteker.

- Docker gjør det enklere å distribuere applikasjoner, kan kjøres med alt som støtter docker-teknologi

Hovedpoenget med virtuelle maskiner er å forenkle det å installere, driftse og kjøre servere. Tidligere kjørte alle servere (webservere og databaser osv.) på fysiske servere, gjerne rackservere som stod i store racks. Så da kjørte alt direkte på hardware altså feltet med **Infrastructure** under! Over hadde man et **Host Operating System** hvor alle applikasjonene kjørte rett på.

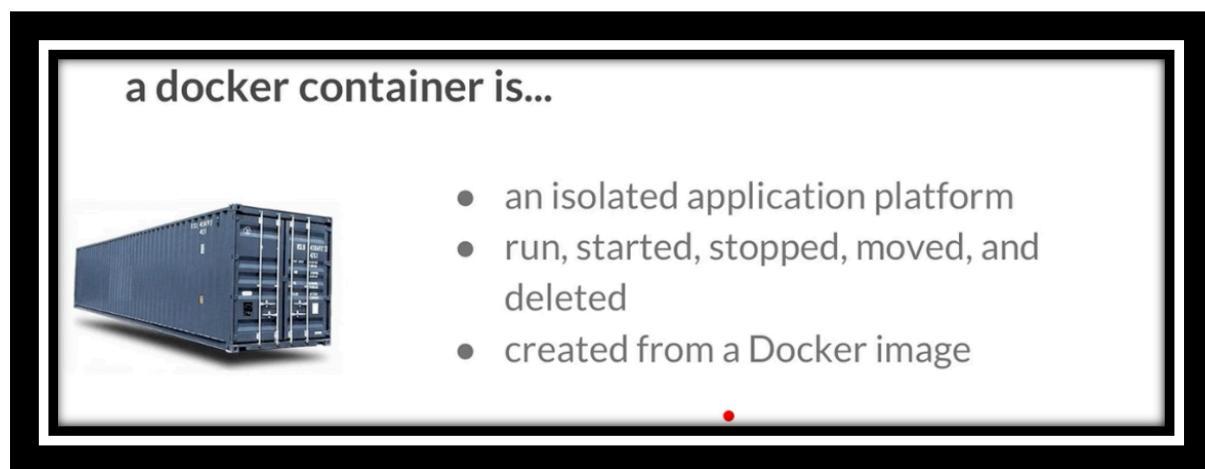
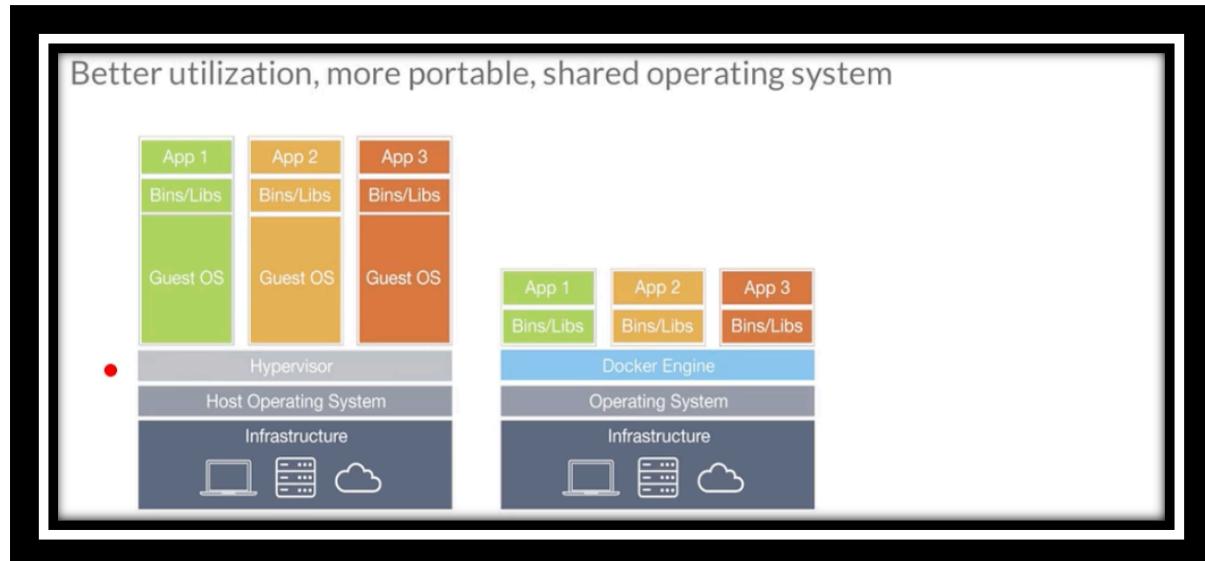
Til venstre ser vi **Hypervisor** som er infrastrukturen som gjør at man kan kjøre virtuelle maskiner. Oppå hypervisoren kjører det 3 virtuelle maskiner, og helt øverst så kjører applikasjonen. Oransje, gul og grønn kan være ulike typer av OS for eksempel: Ubuntu, Red Hat og Fedora, eller så kan de alle være det samme OS.

Hypervisoren, en virtualiseringsplattform, gjør det mulig å kjøre flere virtuelle maskiner (VM-er) på en enkelt fysisk datamaskin eller server. Ansvaret er å dele datamaskinenes fysiske ressurser mellom de virtuelle maskinene som kjører på den, og gjør det mulig å kjøre flere operativsystemer og applikasjoner på en enkelt datamaskin samtidig.

Hypervisoren gir for hver av de virtuelle maskinene over det samme API som fysisk hardware gir.

HØYRE SIDE: DOCKER

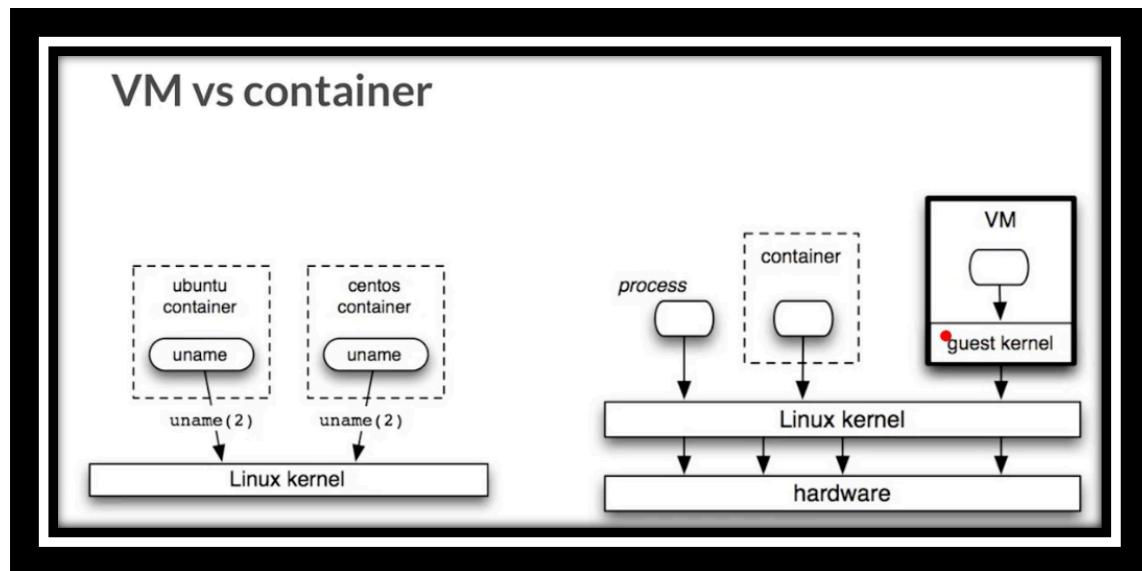
Under på høyre side er det samme prinsippet med like eller ulike os som har applikasjoner kjørende. Her er det en veldig forenkling med et underliggende operativsystem. Det er en docker engine som ligger mellom operativsystemet og docker containerne. Docker bruker mindre ressurser og en lynraske å starte. Operativsystemet er her allerede opp og kjører men ellers kan det ta rundt et halvt minutt.



VM VS CONTAINER

Under til høyre ser man en VM som kjører på en gjeste-OS. Man ser at en container er midt i mellom en vanlig VM og en prosess. I VM-en kjører prosessen på en gjeste-OS, mens containeren består av en vanlig prosess som på denne måten er isolert fra alle andre prosesser på en mye bedre måte enn standard.

Til venstre ser vi to ulike containere som begge kjører på Linux-kjernen med ulike os. Vi har også Windows containere, men de trenger en Windows kjerner under.



IMAGE

I docker og container-teknologien referer «image» til en ferdigpakket applikasjon og dens avhengigheter som kan kjøres i en container. Et image kan betraktes som en byggeblokk for en container, og består av alt som trengs for å kjøre applikasjonen, inkludert operativsystem, biblioteker, applikasjonskode og konfigurasjonsfiler.

DOCKER-IMAGE

Kan lages ved å skrive en Dockerfile, som er en oppskrift på hvordan image skal bygges, eller ved å laste ned et eksisterende image fra Docker Hub, som er et offentlig register for Docker-images.

Når et Docker-image er bygget eller lastet ned, kan det brukes til å starte en eller flere Docker-containere som kjører applikasjonen i et isolert miljø. Hver container kjører en instans av image, og de kan kommunisere med hverandre og med vertsoperativsystemet eller andre nettverkstjenester. Docker-imageet kan også deles med andre utviklere og brukere ved å laste opp til Docker Hub eller et annet image-registre. Dette gjør det enkelt å dele og distribuere applikasjoner og applikasjonsmiljøer, og det bidrar til en rask og effektiv utviklings- og distribusjonsprosess.

HELLO WORLD

Under logges det inn til group25 og det byttes til root:

```
Last login: Wed Mar  8 10:35:18 on ttys000
[amnadarstgir@Amnas-MacBook-Air ~ % ssh group25@os25.vlab.cs.oslomet.no
The authenticity of host 'os25.vlab.cs.oslomet.no (10.196.22.25)' can't be established.
ED25519 key fingerprint is SHA256:vc+gAXWBaRGtkbBtMTbdum5BhdaAuaiM3pGnnUkFr+s.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'os25.vlab.cs.oslomet.no' (ED25519) to the list of known hosts.
group25@os25.vlab.cs.oslomet.no's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-132-generic x86_64)

[ * Documentation: https://help.ubuntu.com
  * Management: https://landscape.canonical.com
  * Support: https://ubuntu.com/advantage
Last login: Mon Mar 13 20:36:07 2023 from 10.47.144.83
group25@os25:~$ sudo su
[sudo] password for group25:
[root@os25:/home/group25# ]
```

Kommandoer som gir info om docker info og versjon.

```
$ docker info
$ docker version
```

Men hva er det som egentlig skjer?:

1. Docker-verktøyet laster ned Docker-image «hello-world» fra Docker Hub, et offentlig register for Docker-images
2. Docker-verktøyet lager en Docker-container basert på dette image, og starter denne containeren
3. Når Docker-containeren starter, kjører den enkle «Hello World»-applikasjonen, som skrivet ut en melding i terminalen som bekrefter at Docker-containeren kjører.
4. Når «Hello World» applikasjonen er ferdig kjørt, stopper Docker-containeren automatisk.

Det som skjer under overflaten, er at Docker-verktøyet isolerer applikasjonen og dens avhengigheter i en virtuell kontainer. Containeren har sin egen filsystem og nettverksstabel, og kjører applikasjonen i en isolert og beskyttet miljø. Dette gjør det enkelt å distribuere og kjøre applikasjoner på tvers av forskjellige plattformer og systemer, og sikrer at applikasjonen fungerer likt uavhengig av hvor den kjører.

```
root@osG70:~# docker container run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
423ae2b273f4: Pull complete
de83a2304fa1: Pull complete
f9a83bce3af0: Pull complete
b6b53be908de: Pull complete
Digest: sha256:04d48df82c938587820d7b6006f5071dbbfceb7ca01d2814f81857c631d44df
Status: Downloaded newer image for ubuntu:latest
```

"Docker container run -it ubuntu bash" er en kommando som kjører en ny Docker-container basert på Ubuntu image, og lar deg samhandle med denne containeren via et interaktivt bash-skall.

Mer spesifikt betyr følgende:

- "docker container run" starter en ny Docker-container
- "-it" flaggene gjør at Docker-kommendolinjen blir interaktiv og knytter til stdin og stdout i containeren
- "ubuntu" angir at Docker skal bruke Ubuntu-image for å lage en ny container
- "bash" angir at Docker skal starte en bash-shell inne i containeren

Så når du kjører denne kommandoen, vil Docker-verktøyet laste ned Ubuntu-image fra Docker Hub hvis det ikke allerede er lastet ned, lage en ny container basert på dette image, og starte en bash-shell inne i containeren, slik at du kan samhandle med den på samme måte som om du skulle logget inn på en Ubuntu-server.

Merk at når du avslutter bash-shell-en inne i containeren, vil Docker-containeren stoppes og fjernes, med mindre du har angitt å beholde den ved å bruke "--rm" flagget ved kjøring av kommandoen.

```
root@osG70:~# docker container ps
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS
NAMES
root@osG70:~# docker container ps -a
```

«Docker container ps» vil vise alle docker som kjører.

«Docker container ps -a» eller “Docker container ps -as” vil liste alle docker containerer som har kjørt, inkludert de som er stoppet.

Under ser han hvor lang tid det tar å starte opp en container igjen.

```
root@osG70:~# time docker container run -it ubuntu bash
root@16dc01fedc0:/# exit

real    0m4,790s
user    0m0,316s
sys     0m0,136s
root@osG70:~#
```

Man kan laste ned images fra Docker Hub med kommandoen «docker pull». Når man kjører kommandoen vil verktøyet laste ned Linux-image fra Docker Hub, og lagre den i den lokale Docker-imagerepositoryen på vrtsovervingsystemet ditt. Dette gir muligheten til å kjøre docker containere basert på dette image senere ved å bruke kommandoen «docker container run» med «alpine» som argument for -it flagget i tilfellet under:

Downloading an image

```
$ docker pull alpine
```

Alpine er en liten linux versjon som er liten på kontrast med ubuntu som er veldig stor. I den første kommandoen vil man starte containeren, gjøre ls -l også stopper containeren.

Tilsvarende under starter man alpine skriver ut echo-meldingen og lukker.

Run a command

```
$ docker container run alpine ls -l
```

```
$ docker container run alpine echo "hello from
alpine"
```

ALPINE

Alpine er en lettvekts, enkel og sikker Linux-distribusjon som er spesielt populær blant Docker-brukere på grunn av sin lille størrelse og minimale ressursbruk. Alpine Linux inkluderer bare de mest grunnleggende komponentene som trengs for å kjøre et

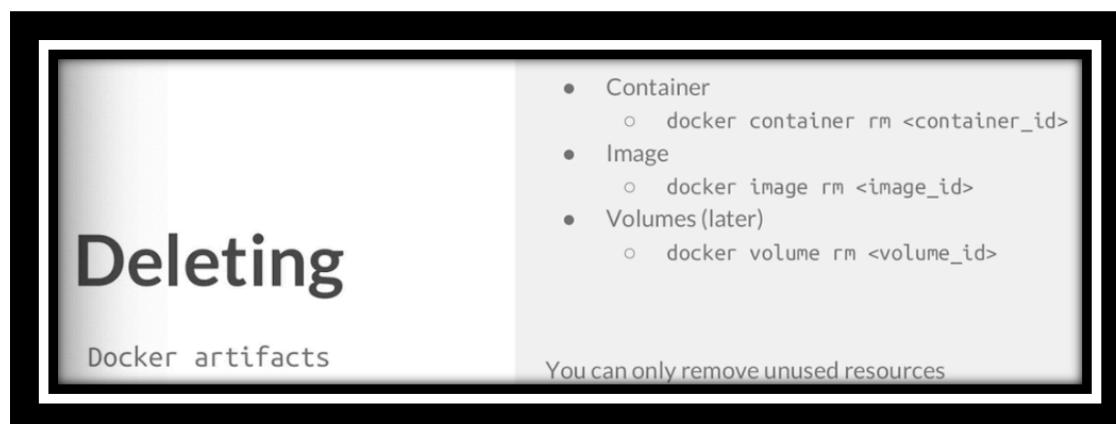
operativsystem, og den har derfor et lite fotavtrykk og krever minimalt med ressurser. Den støtter en rekke pakkebehandlingssystemer, inkludert apk, som gir en enkel måte å installere og administrere programvarepakker på. Alpine Linux er også kjent for å være sikker og stabil, og det er vanligvis raskt å patche kjente sårbarheter når de oppdages.

JVM

JVM står for Java Virtual Machine, og det er en virtuell maskin som gjør det mulig å kjøre Java-programmer på en rekke forskjellige plattformer, uavhengig av vertsoperativsystem og maskinvarearkitektur.

Annen kommando for å liste alle containere er «`docker container ls -a`».

SLETTING/FJERNING



Containerne tar ikke så mye plass som image-ene men det er likevel greit å rydde opp. «`docker image prune`» er en kommando som brukes til å fjerne ubrukede docker-images fra vertsoperativsystemets lokale docker- imagerepository. Man kan også bytte ut image med container, network, volume eller system.

```
root@osG70:~# docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
a45f01174649        alpine              "echo 'fra alpine'"   About a minute ago   Exited (0) 58 sec
ds_ago              alpine              "echo 'fra alpine'"   About a minute ago   Exited (0) About a
minute ago
8018167c0420        alpine              "echo 'fra alpine'"   About a minute ago   Exited (0) About a
16dc01fedc0         ubuntu              "bash"                28 minutes ago     Exited (0) 28 minut
es_ago              ubuntu              "bash"                29 minutes ago     Exited (0) 28 minut
cd7d74fcfa22        ubuntu              "bash"                32 minutes ago     Exited (0) 30 minut
es_ago              ubuntu              "bash"                40 minutes ago     Exited (0) 40 minut
e5b9634e3637        ubuntu              "bash"                40 minutes ago     Exited (0) 40 minut
es_ago              hello-world        "/hello"              40 minutes ago     Exited (0) 40 minut
9826bb2d54b1        hello-world        "/hello"              40 minutes ago     Exited (0) 40 minut
es_ago              hello-world        "/hello"              40 minutes ago     Exited (0) 40 minut
819988bc3c84e       hello-world        "/hello"              40 minutes ago     Exited (0) 40 minut
hour_ago            keen_davinci     "/bin/sh"              40 minutes ago     Exited (0) 40 minut
root@osG70:~# docker container rm a45f01174649
a45f01174649
root@osG70:~#
```

I bildet over slettes en container ved hjelp av id (også det samme som hostname). Man kan også bytte ut de id med feltet «names».

LISTE ALLE SOM KJØRER

«docker container ls»

LISTE ALLE INKLUDERT DE SOM IKKE KJØRER

«docker container ls -a»

LISTER ALLE IMAGES

«docker images ls»

/etc/lsb-release - Innholdet i filen er enkle variabeldefinisjoner som beskriver noen grunnleggende opplysninger om vertsoperativsystemet, som distribusjons-ID, distribusjonsversjon, distribusjonsbeskrivelse, distribusjonsfrigivelsesnummer og noen andre metadata.

POST FORWARDING

Refererer til videresending av nettverkstrafikk fra en port på vertsoperativsystemet til en port i en Docker-container som kjører på samme vertsoperativsystem. Dette kan være nyttig når du har en applikasjon som kjører i en Docker-container og som krever tilgang til en bestemt nettverksport på vertsoperativsystemet. Ved å sette opp post forwarding kan du gjøre denne porten tilgjengelig for Docker-containeren, slik at applikasjonen i containeren kan kommunisere med eksterne enheter som bruker den gitte porten.

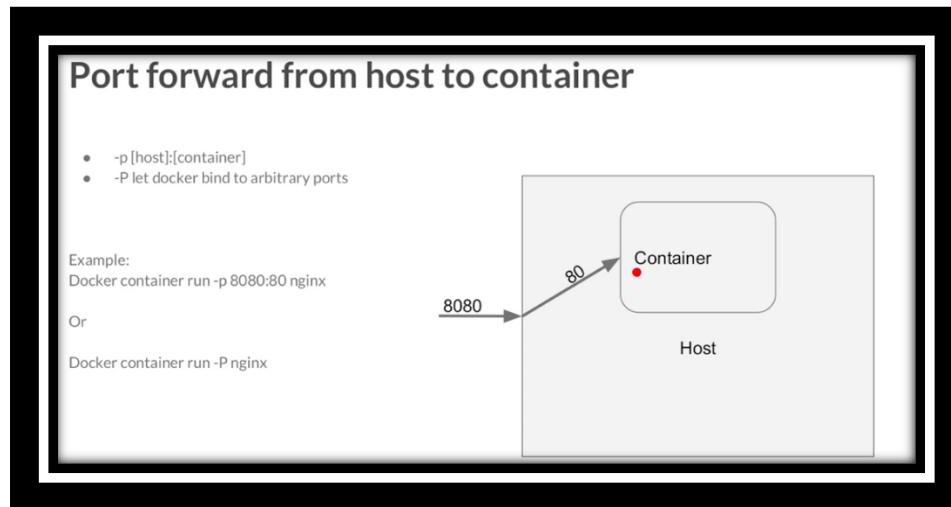
EKSEMPEL: viderefremidle port 80 fra vertsoperativsystemet til en container som kjører Apache webserver, kan du bruke følgende kommando:

```
docker run -d -p 80:80 httpd
```

«-d» til «detached mode» betyr at Docker-containeren kjører i bakgrunnen og ikke tar opp konsollvinduet. Dette er nyttig når du vil kjøre Docker-containeren som en daemon-prosess og ikke vil ha utdataene fra Docker-containeren til å vises i konsollvinduet.

Opsjonen "-p" i Docker-kommandoen står for "port forwarding" og brukes til å videresende trafikk fra en port på vertsoperativsystemet til en port i Docker-containeren.

PORT FORWARD FROM HOST TO CONTAINER

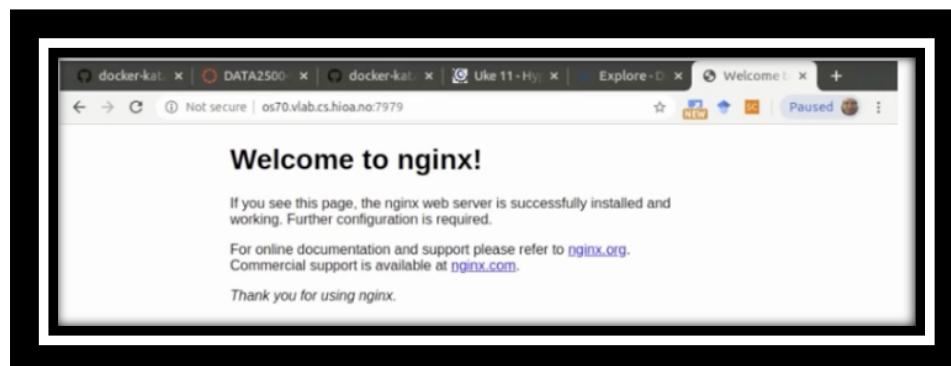


Her tar hårek port 7979 og sender den til port 80:

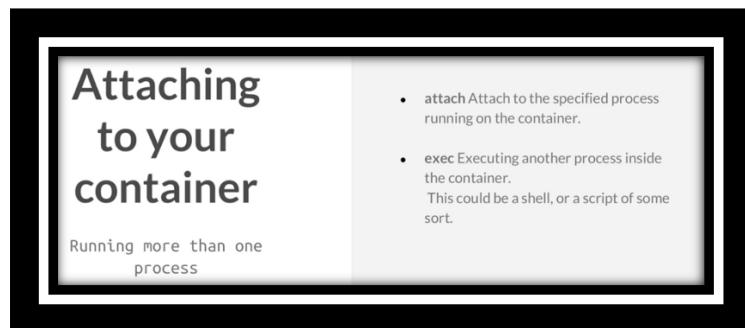
```
root@65fb40e639c4:/          root@e97bc379cae0:/  
root@osG70:~# docker container run -p 7979:80 nginx
```

Kommandoen over brukes til å kjøre en Docker-container som kjører en Nginx webserver, og videresende trafikk fra vertsoperativsystemets port 7979 til container port 80. nginx er navnet på Docker-imaget som skal kjører i Docker-containeren.

Se lenka han skrev:



Hvordan sette opp en container slik at den står og kjører i bakgrunnen, og hvordan gå inn og endre den? → attach og execute



Under er det en kommando som brukes til å starte en ny docker-container som kjører ubuntu-operativsystemet og starter en interaktiv bash-terminal inne i containeren. Så når jeg kjører denne kommandoen, vil Docker starte en ny container som kjører Ubuntu-operativsystemet og starter en interaktiv bash-terminal inne i containeren. Dette gjør det mulig å utføre interaktive oppgaver inne i Docker-containeren, for eksempel installere programvare eller utføre andre kommandoer.

```
root@osG70:~# docker container run -dit ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
423ae2b273f4: Pulling fs layer
de83a2304faf: Pulling fs layer
f9a83bce3af0: Pulling fs layer
b6b53be908de: Waiting
```

Under er det en måte å koble seg opp med containeren som er oppe og kjører. Dette kan gjøres ved exec -it id-nr

```
root@osG70:~# docker container ps -a
CONTAINER ID        IMAGE       COMMAND      CREATED          STATUS
f556566d27a2        ubuntu      "/bin/bash"   44 seconds ago   Up 41 seconds
8a8d6cae871        nginx      "nginx -g 'daemon off'"  5 minutes ago   Exited (0) 2 minu
tes ago             ecstatic_poincare
utes ago            unruffled_kare
7daabc49a632        ubuntu:14.04    "/bin/bash"   10 minutes ago  Exited (0) 10 min
utes ago            eager_gagarin
root@osG70:~# docker container exec -it f556566d27a2 bash
```

Under er kommandoen for å starte apache

```
Processing triggers for libc-bin (2.27-3ubuntu1) ...
root@f556566d27a2:/# /etc/init.d/apache2 start
 * Starting Apache httpd web server apache2
```

Under vises hvordan vi kan finne den som kjørende prosesser. Men hvis man tar exit vil alt stoppe. Man kan ta ctrl+p ctrl+q mens han holder control inn skal man trykke p og q. Nå kjører det en webserver på port 80 inni denne containeren.

```
root@f556566d27a2:/# ps aux | grep apa
root      966  1.0  0.4  73960  4720 ?        Ss   10:39   0:00 /usr/sbin/apache2 -k start
www-data  969  2.3  1.0  2067060 10444 ?        Sl   10:39   0:00 /usr/sbin/apache2 -k start
www-data  971  3.1  0.6  2067060 6364 ?        Sl   10:39   0:00 /usr/sbin/apache2 -k start
root     1026  0.0  0.1  11464  1104 pts/1    S+   10:40   0:00 grep --color=auto apa
```

LAGE KOPI AV EKSISTERENDE CONTAINER

Lister under kommandoene som viser alle containere inkludert det som kjører og er ferdige. Også er det listet alle docker image. Så skriver hårek kommdanoen som kopierer et image med ID(hostname) og kaller den nye kopien apacheubuntu. Hvis man lister i

```
root@osG70:~# docker container ps -a
CONTAINER ID        IMAGE           COMMAND          CREATED          STATUS
PORTS             NAMES
f556566d27a2        ubuntu          "/bin/bash"      4 minutes ago   Up 4 minutes
8a8d6caeae871      nginx           "nginx -g 'daemon of..." 9 minutes ago   Exited (0) 5 minu
tes ago
65fb40e639c4        ubuntu:14.04    "/bin/bash"      14 minutes ago  Exited (0) 14 min
utes ago
7daabc49a632        ubuntu:14.04    "/bin/bash"      15 minutes ago  Exited (0) 14 min
utes ago
root@osG70:~# container image ls
bash: container: command not found
root@osG70:~# docker image ls
REPOSITORY          TAG           IMAGE ID            CREATED          SIZE
nginx              latest        6678c7c2e56c    8 days ago       127MB
ubuntu             latest        72300a873c2c    2 weeks ago      64.2MB
alpine              latest        e7d92cdc71fe    7 weeks ago      5.59MB
ubuntu             14.04        6e4f1fe62ff1    2 months ago     197MB
hello-world         latest        fce289e99eb9    14 months ago    1.84kB
root@osG70:~# docker container commit f556566d27a2 apacheubuntu
sha256:c2a65a4cb29f3bbbefd2446096d428bdd8a9223b6aa9febdf31155c6c7f1ba5b
```

Hvis man lister igjen er det mulig å se at det har blitt opprettet en ny image som heter apacheubuntu:

```
root@osG70:~# docker container commit f556566d27a2 apacheubuntu
sha256:c2a65a4cb29f3bbbefd2446096d428bdd8a9223b6aa9febdf31155c6c7f1ba5b
root@osG70:~# docker image ls
REPOSITORY          TAG           IMAGE ID            CREATED          SIZE
apacheubuntu        latest        c2a65a4cb29f    7 seconds ago   189MB
nginx              latest        6678c7c2e56c    8 days ago       127MB
ubuntu             latest        72300a873c2c    2 weeks ago      64.2MB
alpine              latest        e7d92cdc71fe    7 weeks ago      5.59MB
ubuntu             14.04        6e4f1fe62ff1    2 months ago     197MB
hello-world         latest        fce289e99eb9    14 months ago    1.84kB
root@osG70:~#
```

Dette vil starte en ny docker container, og starte et interaktivt bash-terminal inne i containeren. Trafikken vil videresendes fra vertsoperativsystemets port 7878 til 80 i docker containeren slik at man kan få tilgang til Apache webserveren i containeren ved å besøke localhost:7878 i nettleseren».

```
root@osG70:~# docker container run -p 7878:80 -dit apacheubuntu /bin/bash
2ad42bba695564842e28af2d495693b832f58d7963f65f583ae6b1aecb2acc0d
root@osG70:~#
```

Under med docker container ps listet alle kjørende docker containere på vertsmaskinen. Man får listet:

- CONTAINER ID: En unik identifikator for containeren.
- IMAGE: Navnet på Docker-imaget som containeren ble startet fra.
- COMMAND: Kommandoen som ble brukt til å starte containeren.
- CREATED: Tidspunktet da containeren ble opprettet.
- STATUS: Statusen til containeren (kjører, stoppet, etc.).
- PORTS: Portene som er videresendt fra containeren til vertsmaskinen.
- NAMES: Navnet på containeren.

```
root@osG70:~# docker container ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
RTS
2ad42bba6955        apacheubuntu      "/bin/bash"        18 seconds ago   Up 14 seconds    0.0.0.0:7878->80/tcp
0.0.0.0:7878->80/tcp  eloquent_mahavira
f556566d27a2        ubuntu             "/bin/bash"        9 minutes ago    Up 9 minutes
eager_joliot
root@osG70:~# docker container exec -it 2ad42bba6955 bash
root@2ad42bba6955:/# /etc/init.d/apache2 start
 * Starting Apache httpd web server apache2
```

Under filtreres alle kjørende prosesser som har «apa» i navnet. Tar man control p og q, lister, så kan man se at den står og kjører.

```
root@2ad42bba6955:/# ps aux | grep apa
root      45  1.5  0.4  73960  4612 ?        Ss   10:46  0:00  /usr/sbin/apache2 -k start
www-data  49  1.2  1.2  2067060 12692 ?        Sl   10:46  0:00  /usr/sbin/apache2 -k start
www-data  76  1.3  1.0  2067060 10652 ?        Sl   10:46  0:00  /usr/sbin/apache2 -k start
root     111  0.0  0.1  11464  1012 pts/1      S+   10:46  0:00  grep --color=auto apa
root@2ad42bba6955:/#
```

Under lister Hårek opp containere for å logge seg inn og endre HTML fila med cat index.html

```
root@osG70:~# docker container ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
RTS                 apacheubuntu        "/bin/bash"        5 minutes ago     Up 5 minutes      0.0.0.0:7878->80/tcp
2ad42bba6955        eloquent_mahavira   "/bin/bash"        14 minutes ago    Up 14 minutes
f556566d27a2        ubuntu              "/bin/bash"        14 minutes ago    Up 14 minutes
eager_joliot

root@osG70:~# docker container exec -it 2ad42bba6955 bash
root@2ad42bba6955:/# cd /var/www/html/
root@2ad42bba6955:/var/www/html#
```

Sånn kan man legge til mer tekst, eller ved å åpne tekstedigingsverktøy:

```
root@2ad42bba6955:/var/www/html# echo "docker image fra OS70!" > index.html
root@2ad42bba6955:/var/www/html#
```

«apt install curl» er en kommando som brukes på Linux-baserte operativsystemer for å overføre data fra eller til en server, ved hjelp av ulike protokoller som HTTP, FTP, SCPs, og så videre. Curl kan brukes veldig ofte til å teste serverrespons og debugging.

Tester med curl:

```
root@2ad42bba6955:/var/www/html# curl http://127.0.0.1
docker image fra OS70!
root@2ad42bba6955:/var/www/html#
```

Kan også teste sånn:

```
root@2ad42bba6955:/var/www/html# curl http://localhost
docker image fra OS70!
```

DOCKER EKSTRA VIDEO!

Docker er en service man kan skru av og på:

```
group100@os100:~$ sudo su  
[sudo] password for group100:  
root@os100:/home/group100# cd  
root@os100:~/# service docker stop  
* Stopping Docker: docker
```

I utgangspunktet så vil ikke den kjøre før man angir «service docker start»:

```
root@os100:~/# service docker start  
* Starting Docker: docker
```

[OK]

Er også mulig å sjekke statusen til dockeren og om det er oppe og kjører eller ikke:

```
root@os100:~/# service docker status  
* Docker is running
```

[OK]

Docker ps og docker container ps er helt like kommandoer

```
root@os100:~/# docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
root@os100:~/# docker container ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```



