Operativ Systemer 4

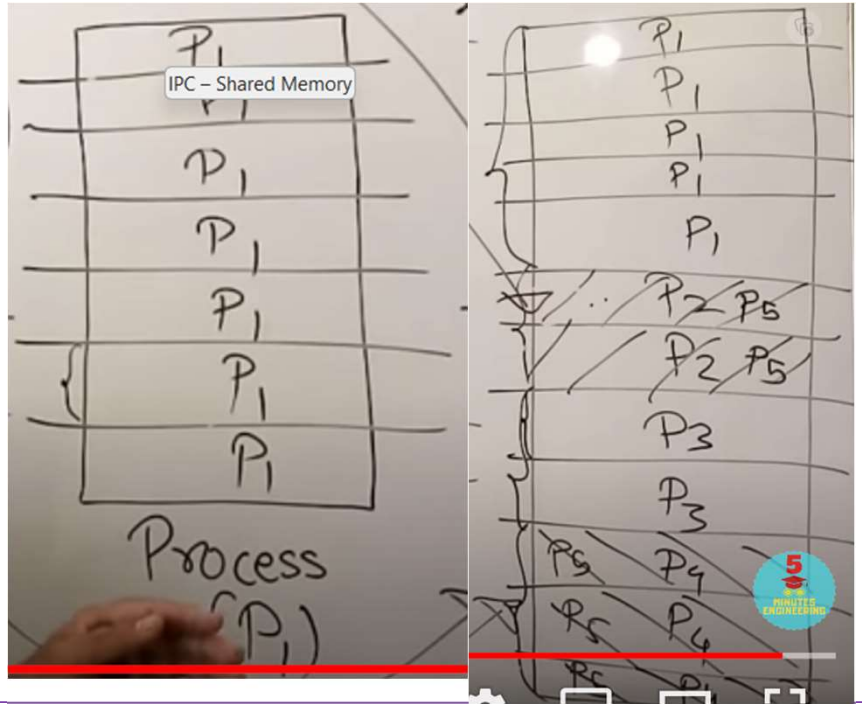
Lavet af: Vivek Misra

Memory and Paging

Her kommer vi til at snakke om Memory Unit, især omkring forskellen mellem Virtuel – og Fysisk Memory samt Page Tables.

Hvad er Paging?

- Vi kan se, at når processer bliver opdelt så er de kaldt for Pages.
- På den samme måde bliver Main Memory Unit opdelt i forskellige dele som er kaldt Frames.
  - Disse Pages bliver indsat i Main Memory Framet.



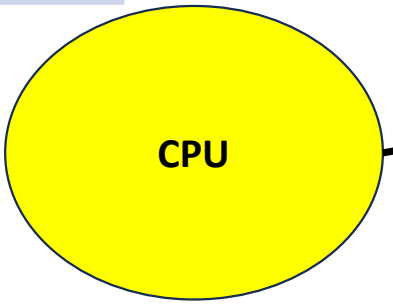
Hvad er Paging?

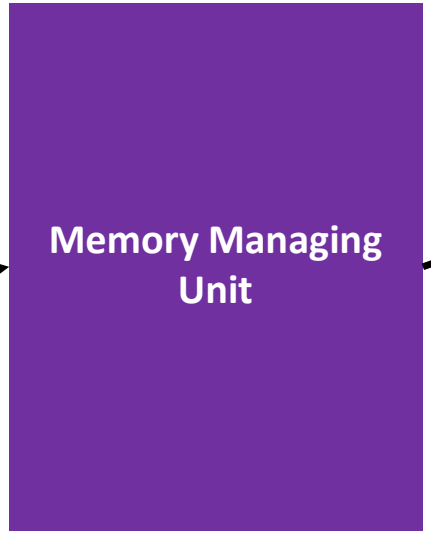
- Nu kommer vi til at snakke om Memory Managing Unit.
  - Vi kan se, at vi ahr forskellige apge af processer som er sat inde i Frames.
  - Men når CPU'en skal til at eksekvere noget arbejde, så er den med til at generere logiske addresser.
  - Ved Logiske Adresse, kan det ses at vi har Page Nummer og Page Offset. Page Nummer er teknisk set vores placering i Framen, men Page Offset er den udvalgte bit vi ønsker at have.
- Vi kan se, at Memory Managing Unit har arbejdet i at konvertere fra Logiske Adresser og om til Fysiske Adresser.
  - Den Fysiske Adresse er den aktuelle adresse, hvor det kan ses at det består af Frame Number og Offset.
- På næste side viser vi, hvordan det ser ud på en grafisk måde.

Hvad er Paging?

NR	Page 1
0	Data
1	Data
2	Data
3	Data
4	Data

NR	Page 2
0	Data
1	Data
2	Data
3	Data





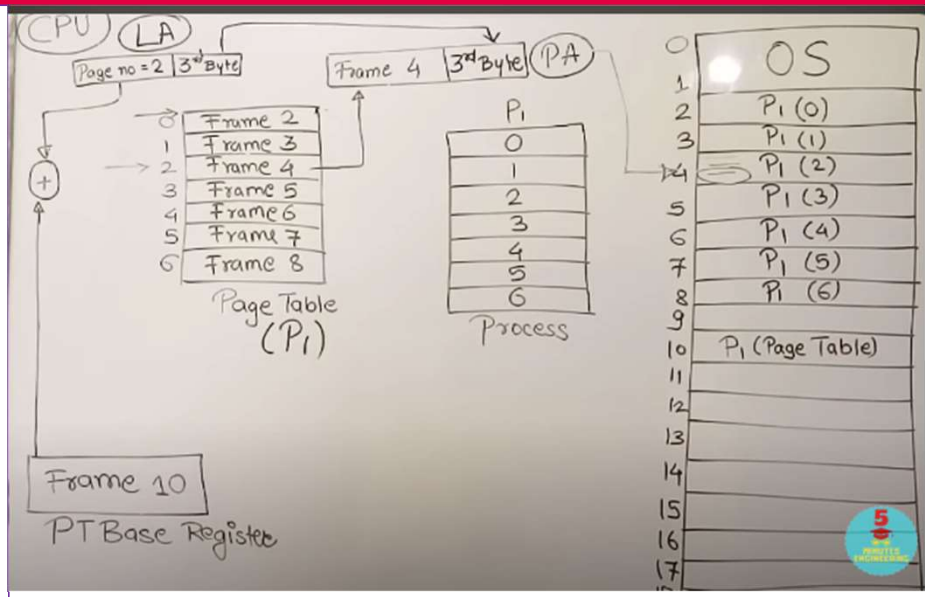
NR	MM
0	P1
1	P1
2	P1
3	P1
4	P1
5	P2
6	P2
7	P2

Logiske Addresser	
Page Frame	Page Offset

Fysiske Addresser	
Frame Nummer	Offset
0	Bits = 1111

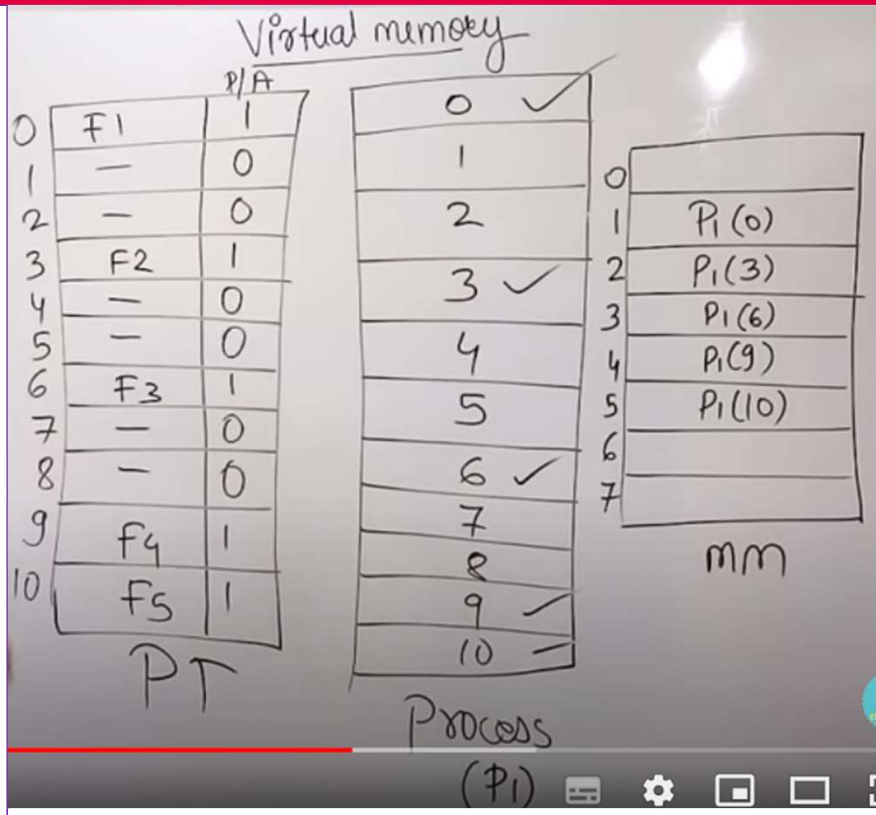
Hvad er Paging?

- Vi kan se, at når vi har noget CPU som eksekvere noget data så har vi noget Page Number fra det Logiske Adresse med Offset.
- Derefter har vi Page Table som indeholder alt informationen, hvor det kan ses at vi refereres til Frame 4 i den store Tabel.
- Derefter finder vi dataet inde i det.



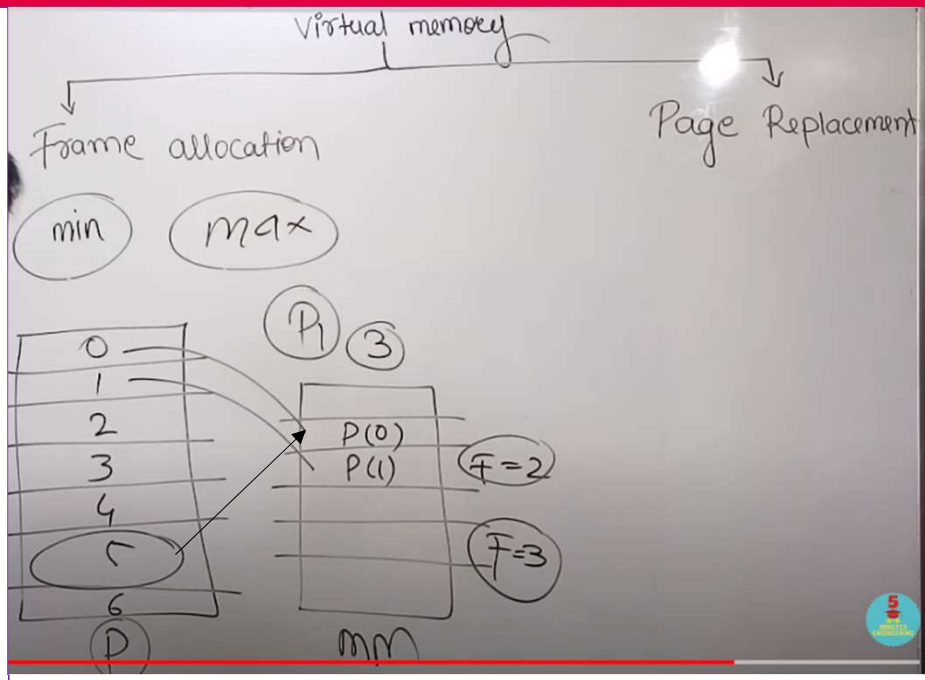
Hvad er Virtuel Memory?

- Idags memory kan nogengange være små på grund af Størrelserne af programmerne vi bruger.
- Dette kan være forårsaget af vores stor efterspørgsel af at køre flere programmer.
- Efter at have opdelt vores Process til Pages, så indsætter vi (udvalgte) Pages inde i Main Memory.
- Udefra de indsatte værdier, indskriver vi inde i Page Tabellen over hvilke værdier der er sat inde i Main Memory.
  - Vi indsætter binære værdier for, at betegne hvilke frames der er indsat i Main Memory.
  - Det betyder, at vi skriver 0 for inaktive (ikke findes i Main Memory) og 1 for aktiv i Main Memory.



Hvad er Virtuel Memory?

- Frame Allokation er der, hvor vi tilsætter pages i framen.
- Hvorimod Page Replacement er der, hvor vi erstatter indeholdet ud som befinder sig i Main Memory ude med noget andet.
  - Et eksempel på dette kan være FIFO = First In First Out, hvor værdien ændres engang den første værdi er eksekveret. Så kommer den næste værdi ind på den første plads.



Hvad er Fysisk Memory?

- Fysisk Memory er også kendt som RAM.
- RAM som vi allerede kender er også kaldt for Random Access Memory.
- RAM giver lagerplads for hurtigere hastighed og lagring for aktiv kørende processer, programmer og data.
  - Dette muliggøre CPU'en i at kunne få adgang til data, som gør at computeren fungerer mere hurtigere.

Software og Basic Koncepter

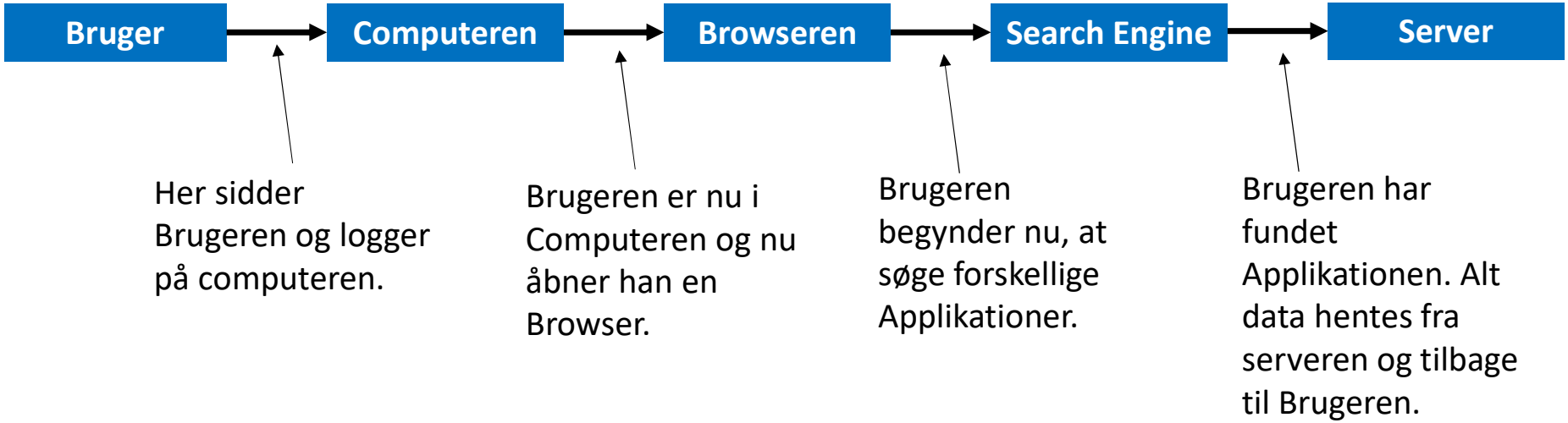
Nu skal vi lige lære nogle basic koncepter om hvad en Software er….

Hvad er Software?

- Når vi snakker om computer referer vi oftest til to termer.
  - Den ene term er kendt som Hardware, hvilket betegner infrastrukturen af computeren.
  - Hvorimod den anden term er Software, som betegner den visuelle system af computeren.
- Eksempel på Hardwaren fra vores daglig dag er:
  - Computeren sig selv, Tastatur, Skærm, CPU, Grafikkort osv.
  - Alle de fysiske genstande som befindes på computeren, og som kan røres (fysisk) ved er kendetegnet Hardware.
- Eksempel på Software fra vores daglig dag er:
  - Alt det visuelle som fremstår på skærmen.
  - Det kan være spil, office365-filerne, webbrowser osv.
- Operativ systemet er det som driver Softwaren.
  - Eksempler på dette kan være Windows, AppleIOS, Linux for Computere.
  - Eksempler på dette kan også være Android, AppleIOS, Microsoft(Lumia) for Telefon.

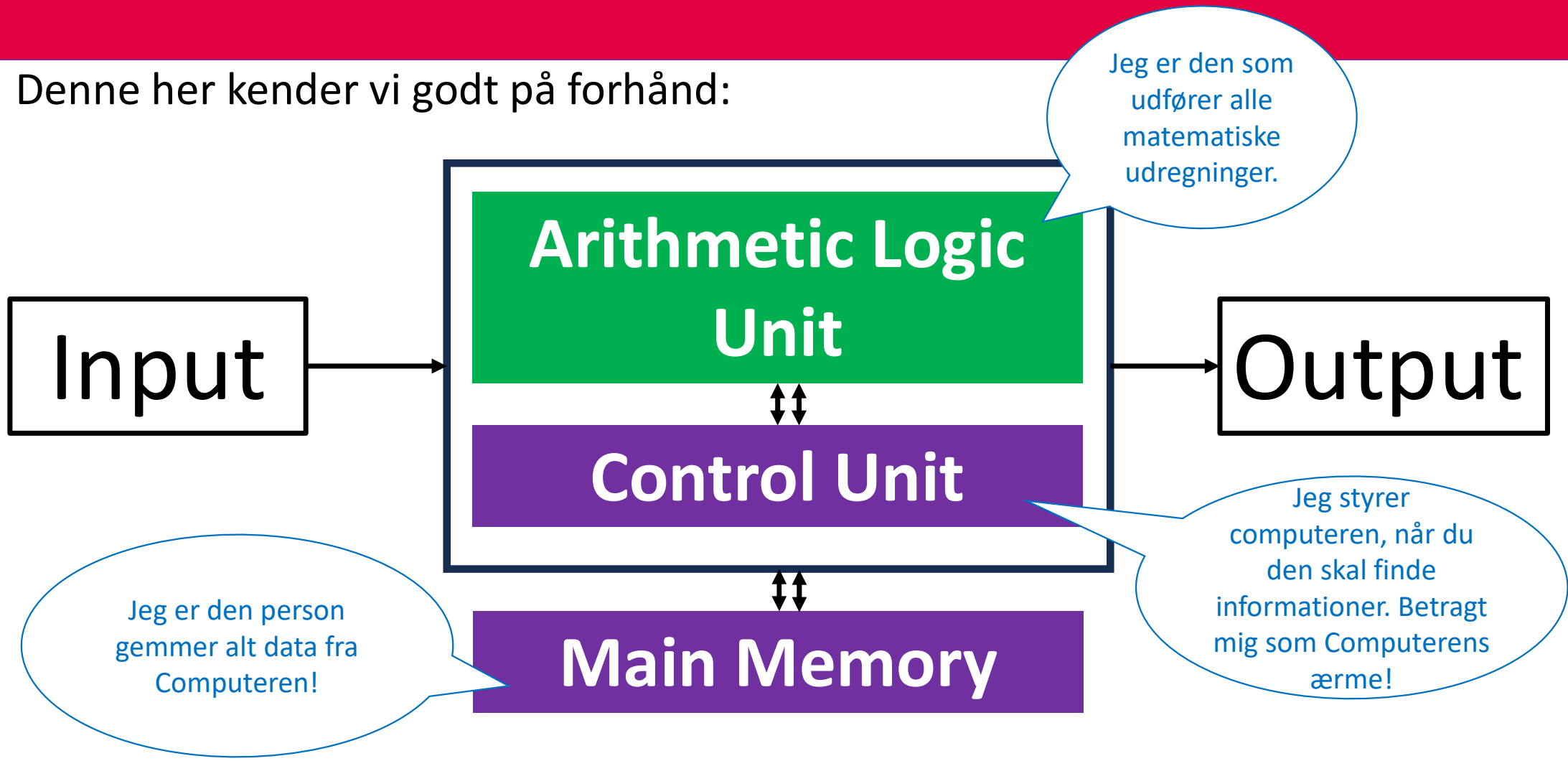
Hvad er Serveren?

- Når man snakker om Netværker plejer vi, at referere Søgning i Serveren.



Hvad er CPU'en?

- Denne her kender vi godt på forhånd:



Threading

Nu skal vi lære om hvad Core er, og hvilken relation de har til Threads.

Introduktion til Processer og Kerner

- Vi kan se, at den samlede CPU er i daglig tale kaldt for en Processor.
- Denne Processor skal udføre arbejdet på computeren.
- Men når Processoren bliver opdelt inde i, så er disse opdelte dele kaldt for Core (Kerne).
- Disse Core har forskellige arbejdsopgaver uddelt iblandt sig.
  - Eksempelvis en Core kan holde styr på Installering af Spil.
  - Den næste Core kan holde styr på samtale i Messenger.
  - Den tredje kan eksempelvis holde styr på at Synkronisere Data i OneDrive osv.
- På næste side viser vi typer af Core udefra opdeling med Grafisk Overblik.

Navngivning af Kerner

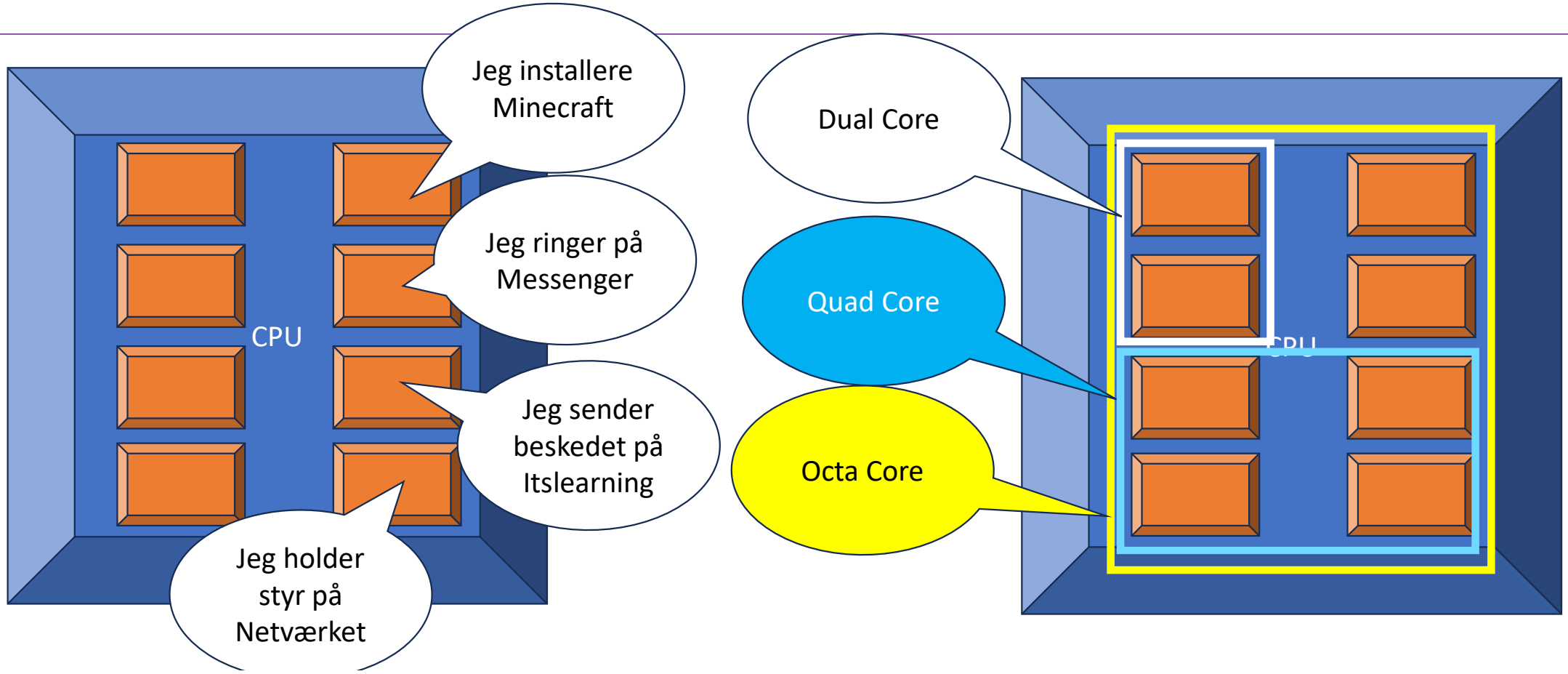
- Alt efter, hvordan Processoren bliver opdelt, så har det også betydning for navngivningen af Kernen (Cores).

Antal Cores	Navngivning af Cores
To Cores	Dual Code
Fire Cores	Quad Core
Otte Cores	Octa Core
Flere end 8 Cores	Multicore

- Man kan se, at eftersom teknologien blev avanceret og udnyttelse på plads hos hardware blev bedre så kan det ses tingene blev mindre i størrelsen.
  - Dette terminologi er kaldt for Nanoteknologi.
  - Basicially som betyder, at man har noget teknologi i små dele.
- Fordi der bliver dannet flere Core, kan vi derfor sige at der findes Multicore i Processoren.

Overblik af Kerner

- Billedet viser, hvordan Cores er indenfor i CPU'en.

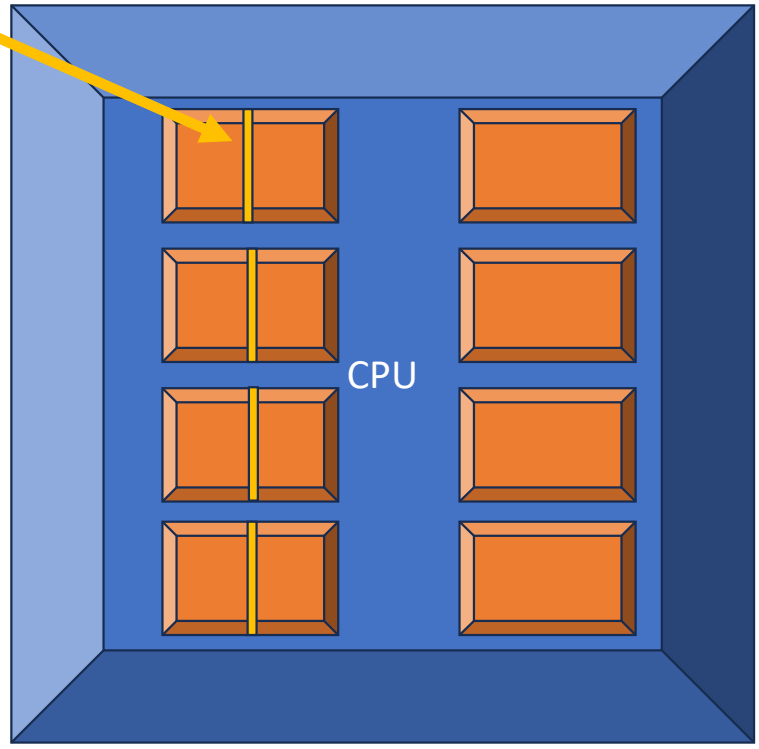


Threading i Kerner

- Som vi kender på forhånd, så er Kerner opdelt udefra en internt opdeling af CPU'en.
- Men når vi snakker om Threading, så kombinerer vi to kerner med hinanden.
  - Man kan sige, at to eller flere bliver syet sammen mellem hinanden, der virker effektivere.
  - Men hvordan skal det egentligt forstås hos Kerner?
- Når to Kerner bliver kombineret sammen, så resulterer det en 2x styrkelsen på deres arbejdsfunktionalitet.
  - EKS: Hvis der sker en Threading mellem to Core (Dual Core), så resulterer det at Dual Cores funktionalitet bliver opgraderet til en niveau svare Quad Cores.

Threading i Kerner

- Vi viser her et Eksempel på Threading i Cores.
  - Den gule plade liggende mellem Cores er Threading, som pilen peger hen imod.
- Det kan ses herhenne, at når vi laver en Threading så bliver Dual Core svarende til en Quad Core.
- Og når det sker ved en Quad Core, så bliver den også svarende til Octa Core ift. Funktionalitet.
- Det er vigtigt, at huske at Threading sker inde i den enkelte Core.
- Threading er IKKE en fysisk grænse mellem to core.

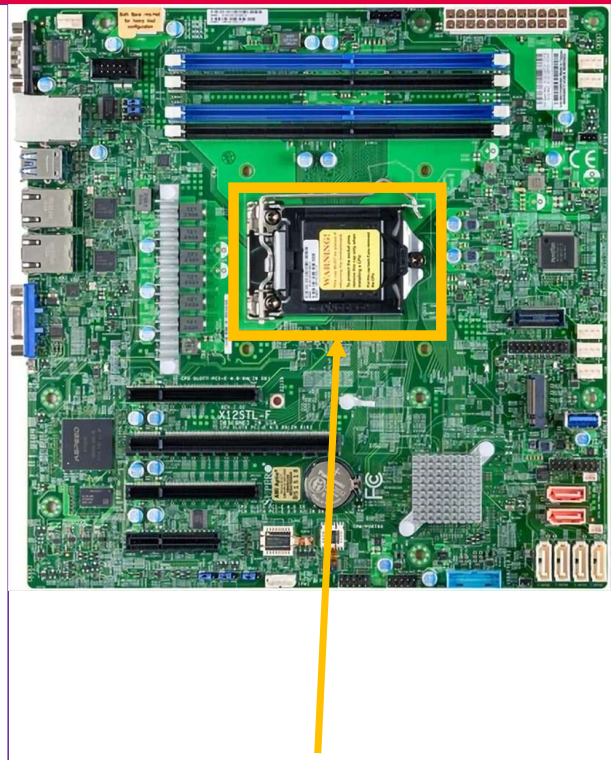


Kerner og Generationer

- Du kender måske Intel som producerer Core til Computeren.
- Du har måske også bidt mærke til at der står af Computerens core tilhører en specifik Generation.
- Denne Generationer fortæller, hvor mange Core er blev forbedre. Oftest er det såden, at virksomhederne forbedre/opdaterer Core 9 gange.
- Når virskomhederne skal lave en stor ændring, skifter de tallet ud ved i's plads og skriver eksempelvis i7 i stedet for i5 – 9th. Generation.

Input, Output og Motherboard

- Input Device er den enhed som indtager informationer.
- Output Device er den enhed som udgiver informationer.
- Nogen Devices er i stand til at have begge funktionaliteter.
- Men når man eksempelvis udskriver noget fra sin computer og til printeren, så undrer man sig over hvordan der dannes forbindelse mellem de to Devices??
  - Det er her, hvor vi snakker om Motherboard der holder Computersystemet sammen.
  - Motherboard kan betragtes som ”morens hjem”, som holder hele systemet sammen.
  - Det er her, hvor alle forbindelserne kommer til. Teknisk set kan man kalde det for Computer Hardwarens Hovedstad!



Dette er en Motherboard!

Multitasking

Vi kommer nu til at introducere til hvad Multitasking er i sammenhæng med Processer.

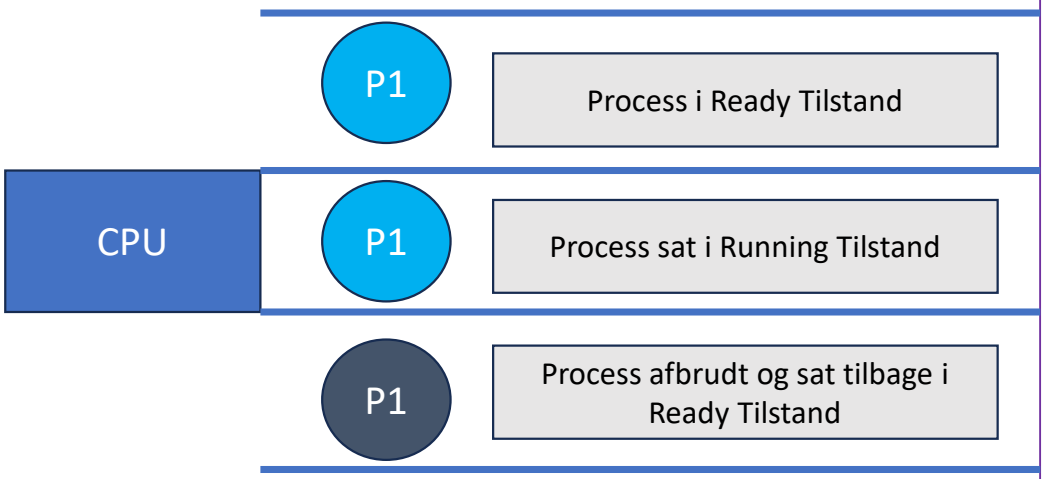
Scheduling Algoritmer

- Hensigten med Scheduling Algoritmer er, at kunne hjælpe CPU'en med følgende:
  - At kunne eksekvere processer inden for et specifik tidsramme.
  - Uddelegere en tidsramme, hvor processen kan eksekveres.
  - Udvælge en process, der er tilpassende for at være eksekveret i CPU'en.
- Der findes forskellige former for Scheduling Algoritmer:
  - Vi kan opdele de forskellige former mellem Preemptive og Non-Preemptive som tabellen:

Preemptive	Non-Preemptive
SRTF = Shortest Remaining Time First	FCFS (First Come First Serve)
LRTF = Longest Remaining Time First	SJF (Shortest Job First)
Round Robin	LJF (Longest Job First)
Priority Based	HRRN (Higest Responsive Ratio Next)
	Multilevel Queue

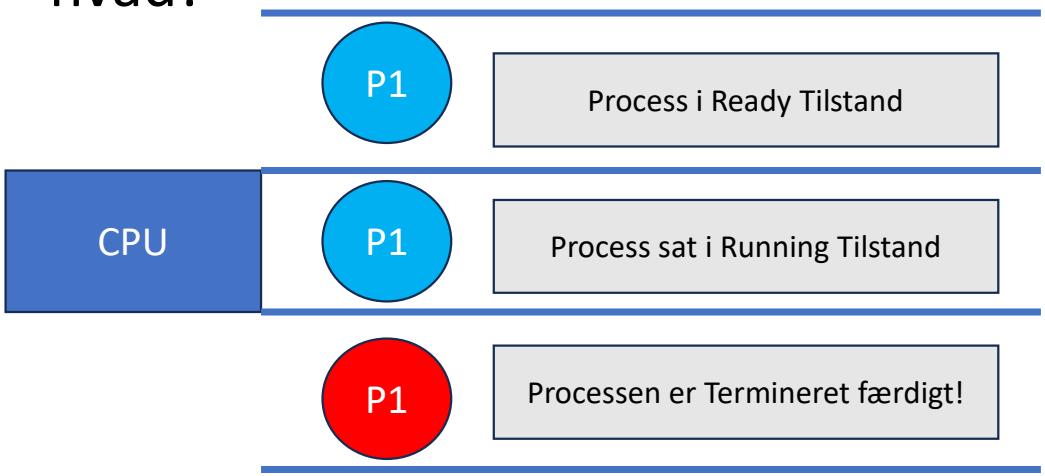
Preemptive

- Preemptive er der, hvor vi kan tage en process og afbryde det undervejs i eksekveringen og derved sætte den tilbage i Ready-Tilstand.



Non-Preemptive

- Non-Preemptive betyder, at den Process som allerede er i gang med at blive eksekveret af CPU'en vil til arbejdets sidste ende blive eksekveret uanset hvad!



Multiprogrammed	Multitasking
• Vi kan se, at det er en Non-Preemptive Process hvilket betyder processen er fuldt ud eksekveret af CPU'en. • Eksempelvis vi har 10 elever med 5 spørgsmål. Eleverne stiller deres spørgsmål foran læreren. • Læreren skal betragtes som CPU'en der vil eksekvere/løse alle de 5 opgaver for den enkelte og derefter gå videre til den næste.	• Vi kan se, at det er en Preemptive Process hvilket betyder at processen vil ikke eksekvere processen fuldt ud, men noget af det! • Eksempelvis vi har 10 elever igen med 5 spørgsmål. Eleven stiller deres spørgsmål foran læreren. • Læreren skal betragtes igen som CPU'en, hvor den bare indenfor et specifik tidsramme vil måske løse 3 spørgsmål eller dem alle og derved gå videre til den næste elev.

Process Synkronisering

- Process Synkronisering er der, hvor vi har utallige processer som kører i systemet.
  - Disse Processer kører i forskellige former som eksempelvis Serial eller Parallel Mode.
  - **Serial**: Det er der, hvor en process er eksekveret ad gangen.
  - **Parallel**: Det er der, hvor utallige processer er eksekveret på samme tid.
- Men når vi snakker om Processer, kan vi inddele dem i to kategorier:
  - **Coorperative**: Hvis adskillige processer har den samme ressource, som eksempelvis CPU, Skanner eller lignende. Så er disse kaldt for Coorporative Processer.
    - ***Coorperative Processer har indflydelse på hinanden, eftersom de deler den samme ressource.***
  - **Independent:** Hvis adskillige processer har forskellige ressourcer, som eksempelvis brug af forskellige server hos forskellige brancher. Så er disse processer kaldt for Independent Processer.
    - ***Independent Processer har IKKE indflydelse på hinanden, eftersom de IKKE deler den samme ressource.***

Race-Condition

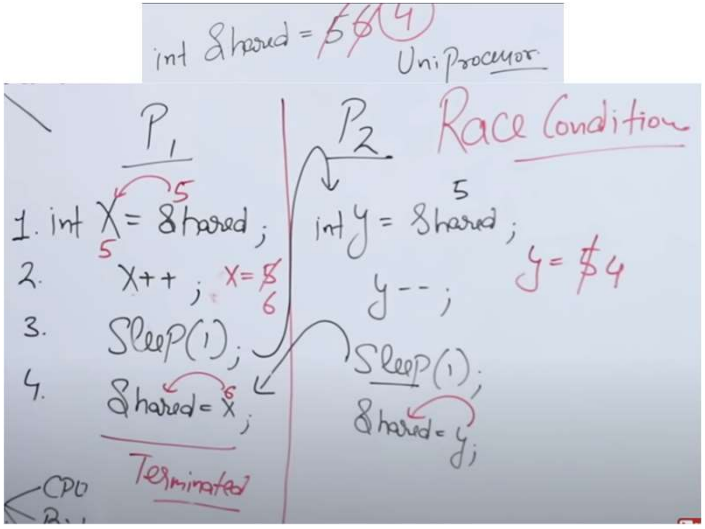
- ***Vi kommer til at fokusere lidt mere på Coorporative Processer.***
  - Grunden til at Coorporative Processer er vigtige, fordi hvis de ikke er synkroniseret ordentligt så kan dette resulterer en Deadlock eller problemer.
  - *EKS: Hvis en individ får udleveret en guitar i hånden, og har ingen forhåndsviden. Så vil personen skabe lyd og problemer og ikke noget synkronisering.*
- EKSEMPEL PÅ BILLEDET TIL HØJRE:

Vi kan se, at der er på forhånd defineret en nummeret variabel ved navn shared.

Når man kører igennem fra process 1, så kan vi se at vi får en værdi der er 6 efter x++. Efterfølgende bliver der lavet en context switching hvilket betyder at vi sætter en process til siden og derefter eksekvere process 2, da der er en sleep-statement i process 1.

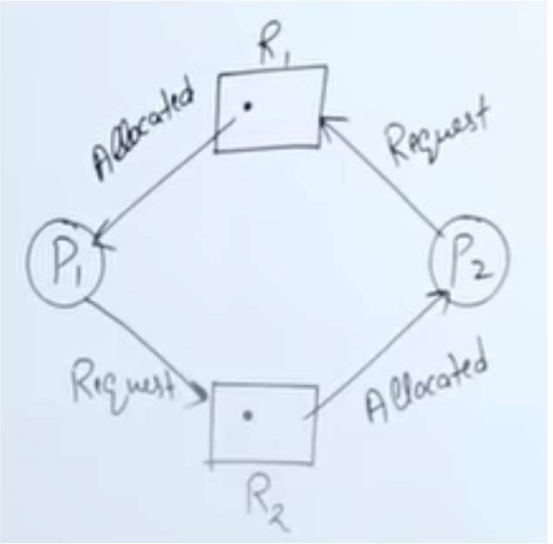
Vi kan se, at når process 2 er begyndt at blive eksekveret så får vi et tal på 4 ved y--, men fordi sleep statement kommer igen så laves der endnu en context switching, hvilket gør at vi skifter til process 1 ved shared = x som giver 6. Og dermed process 2 som giver resultatet 4.

Fordi vi kører to processer i omløb mod hinanden, så kan vi kalde det for en Race-Condition.



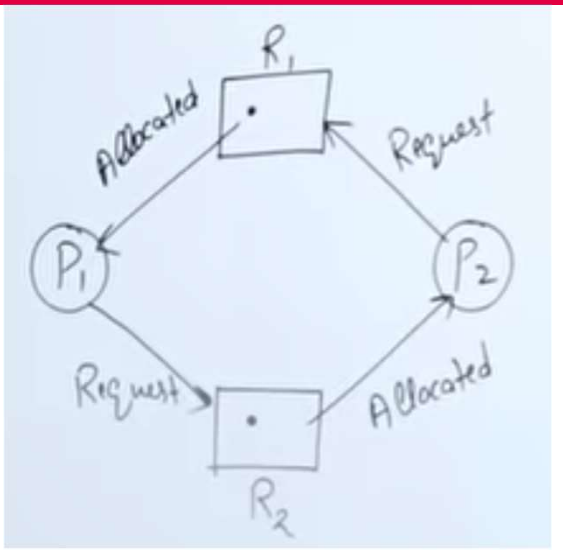
DeadLock

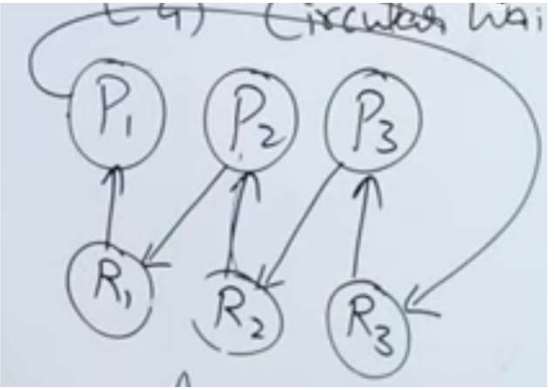
- Når to eller flere processer venter på at der sker noget ”event”, men der aldrig sker noget.
  - Dette process er kaldt for en Deadlock, som gør at to processer ikke kører videre.
- *Man kan se på eksemplet til højre, at vi starter ud med at vi har en Ressource 1 som er allokeret til Process 1, da det er den som bruger den lige nu. Men Process 2 anmoder om, at få Ressource 2, hvilket er ikke muligt. Fordi Ressource 2 er allokeret til Process 2. Det samme gør Process 2, hvor den anmoder om at bruge Ressource 1 men det er ikke muligt pga. Process 1.*
- ***Som det kan ses, så er alt gået i stå herhenne og det er kaldt for Deadlock Princippet.***



DeadLock

- Vi kan se, at for Deadlock er der nogle betingelser som skal opstå for at det virkeligt er en Deadlock:
  1. ***Mutual Exclusion:*** *Betyder, at alle ressourcer som der er, skal bruges af alle processer på en mutual eksklusiv måde. Det betyder, at processer må kun bruge ressourcen en efter en.*
  2. ***No-Preemption:*** *Betyder, at en process må holde fast til en ressource, indtil den er blevet fuldt ud eksekveret.*
  3. ***Hold & Wait:*** *Betyder, at en process må ikke opgive den ressource som de holder fast i. Kig på billedet til højre, det er et tydeligt eksempel.*
  4. ***Circular Wait:*** *Betyder, at når en en process holder fast i en ressource og opkræver en anden ressource, og hvor dette fortsætter nedad ved de andre processer. Kig på nederst billede til højre ->*

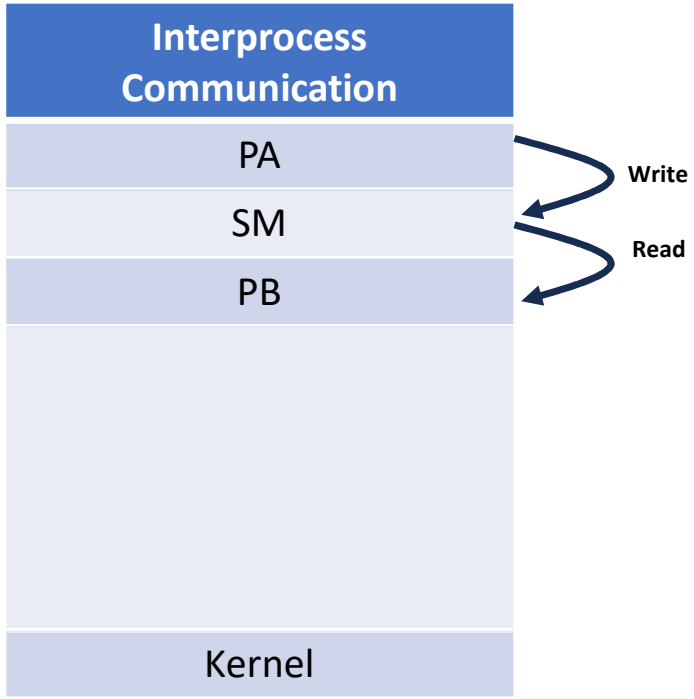




IPC – Shared Memory

- Vi kender allerede på forhånd fra sidste slide, at Interprocess Communication er der, hvor vi har en afsender og en modtager som veksler data mellem hinanden under et maskine.
- ***Kigget på billedet mod højre, kan det ses at vi har to koncepter som er nødvendige at forstå:***

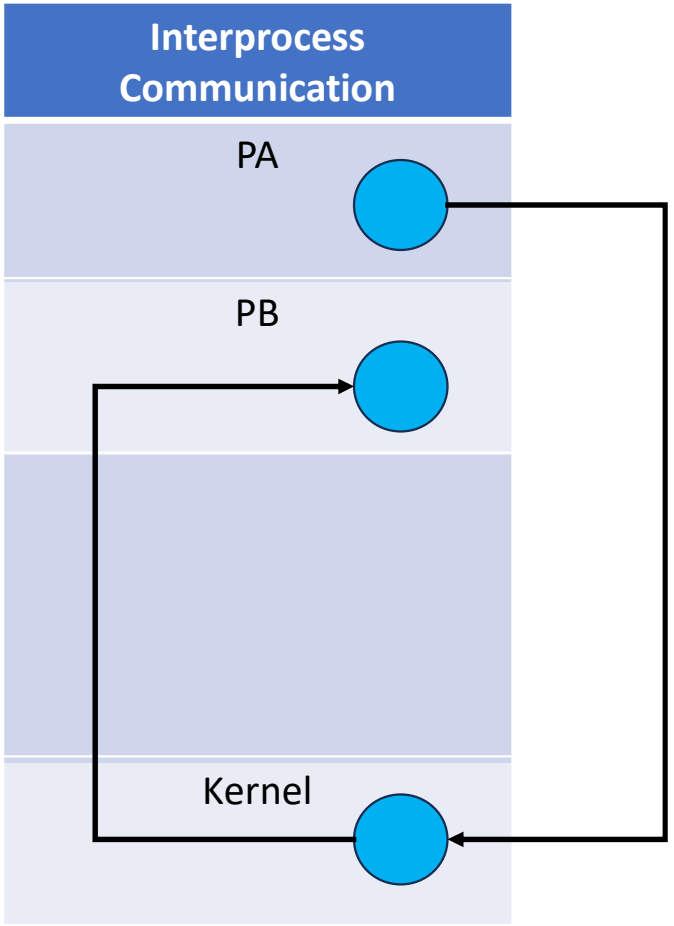
- Vi kan se, at vi har en kommunikation mellem to Processer, som er Process A og Process B. Derefter kan vi se, at vi har Shared Memory som er teknisk set vores resteplads for Memory.
- Her går Process B ind og finder hvilken som helst information, og derefter følgende eksekvere det.



IPC – Message Passing

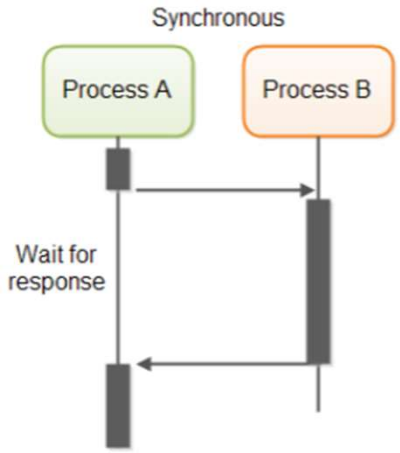
- Vi kender allerede på forhånd fra sidste slide, at Interprocess Communication er der, hvor vi har en afsender og en modtager som veksler data mellem hinanden under et maskine.
- ***Kigget på billedet mod højre, kan det ses at vi har to koncepter som er nødvendige at forstå:***

- Vi kan se, at Process A afsender noget besked til Process B. Men det er ikke en direkte process. Her kan det ses, at vi sender besked fra Process A til Kernel. Når Kernel modtager denne besked, så står det i en message queue som bagefter sendes til Process B.
- Betragt det bare som, at restepladsen er rykket fra Shared Memory til Kernel.



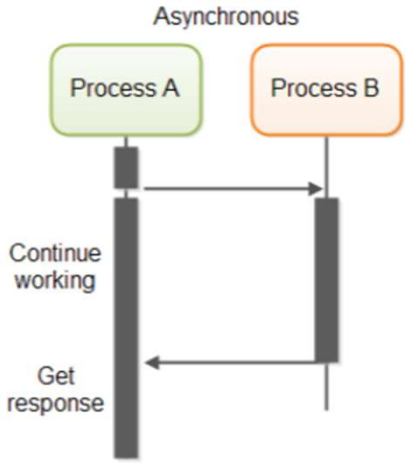
Synkrone Processer

- Synkrone Operation betyder, at en proces kører kun som en resultat af en anden proces som er i gang med at blive færdiggjort.



Asynkrone Processer

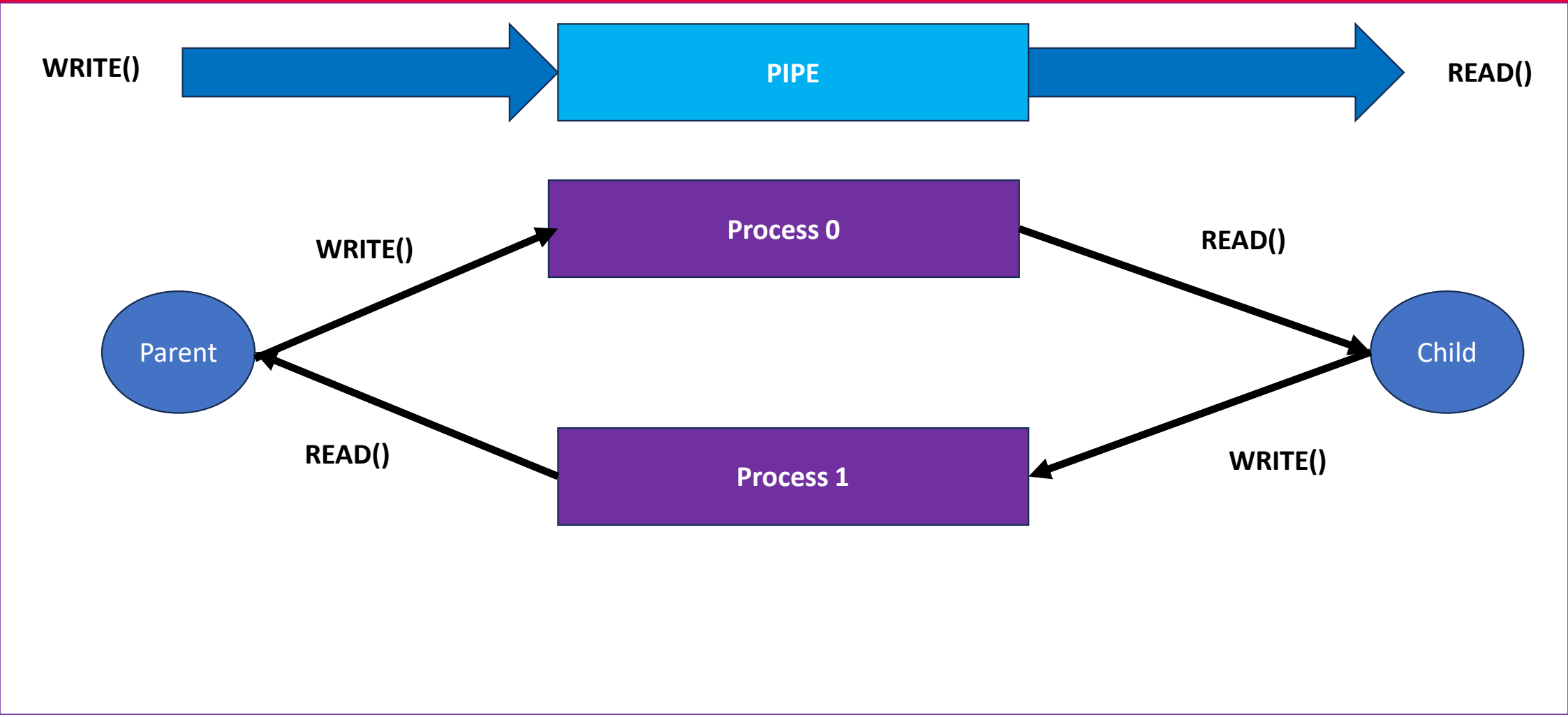
- Asynkrone Processer betyder, at det er en proces som kører uafhængigt af andre processer.



Hvad er Pipe?

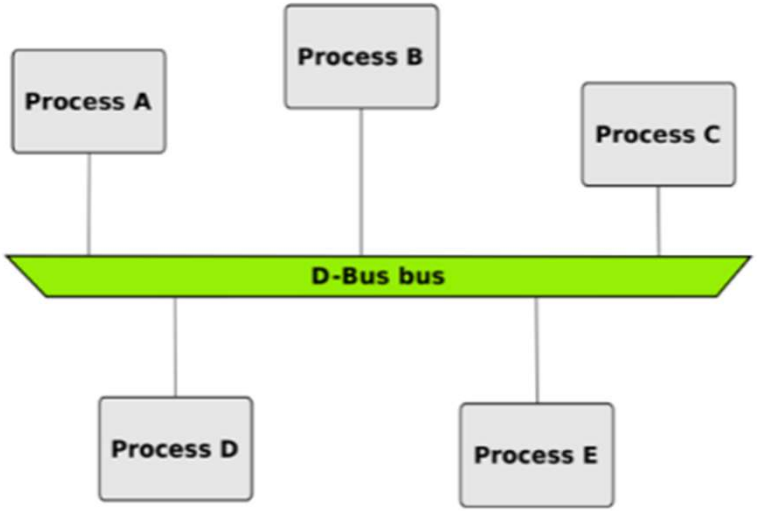
- Hensigten med Pipe er, at kunne illustrer forholdet mellem to processer i en Kommunikation.
  - ***Man kan eksempelvis tage udgangspunkt i FIFO-Princippet, hvor det kan ses at det er den første besked som er sendt inde og som kommer ud som output.***
  - Når man snakker om write() metoden, så er det metoden som illustrer hvor mange bits der er skrevet og read() illustrerer hvor mange bits der bliver læst.
  - Gennem Pipe() metode bliver data'et læst og vi får noget output i form af return.

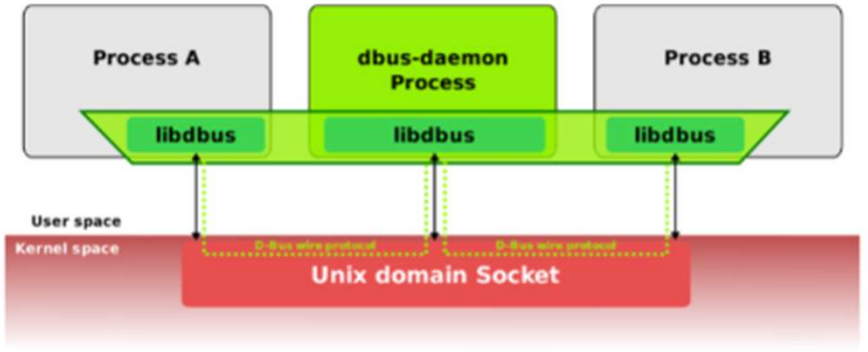
Hvad er Pipe?


WRITE()
PIPE
READ()
Process 0
WRITE()
READ()
Parent
Child
READ()
Process 1
WRITE()

UNIX og DBUS

- *Vi kan se, at UNIX Sockets er der, hvor man har en server som kan læse, skrive og få data fra.*
- *Hvorimod DBUS bruges til at repræsentere data på GUI 'en. Dette er også kaldt for en abstraktion.*





SLUT 4

Lavet af: Vivek Misra