7. Futures + Pipelines

1. Redegør for hvad et design pattern er? (slet dette punkt måske)
2. Futures
    a. Et resultat af en udregning som ikke er kendt endnu
        i. Bliver tilgængelig senere
    b. En future kan udregnes parallelt med andre udregninger
    c. En future er en Task der returnere en værdi
        i. Parallel Task – Asynkron Action
        ii. Future – Asynkron funktion
    d. Benyttes til at parallelisere kode med dataafhængigheder
    e. Eksempel, tegn/skriv eksempel fra disposition:
    f. Hvornår er en future klar?
        i. Hvis en task er færdiggjort, bliver resultatet kaldt som var det en variabel
        ii. Hvis en task er kørende, blokeres funktionen som skal bruge resultatet
        iii. Hvis en task ikke er påbegyndt, sættes den inline i den givne kontekst (hvis muligt)
3. Pipeline
    a. Ønsker at parallelisere en sekventiel proces
    b. Består af en serie af trin (se eksempel i disposition)
    c. Hvert trin er afhængigt af det forriges resultat
        i. Output i trin = input trin+1
    d. Eksempel, billedbehandling
        i. Sequential
        ii. Pipelined Sequential
        iii. Evt. Bottleneck pipeline
    e. Flere eksempler hvis tid!

7. Futures + Pipelines

Futures

En future er et resultat af en udregning som ikke kendes til start men bliver tilgængelig senere. Udregningen af en future kan ske parallelt med andre udregninger. En future er en Task der returnerer en værdi.

- Parallel task – asynkrone actions
- Futures – asynkrone funktioner.

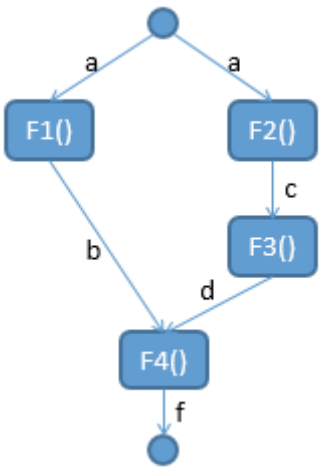
De bruges når vi ønsker at parallelisere kode med data afhængigheder.

```
static void Main(string [] args)
{
    var a = "A";
    var b = F1(a);
    var c = F2(a);
    var d = F3(c);
    var f = F4(b, d);
    System.Console.WriteLine(f);
}
```

Ved at køre ovenstående kode, bliver det hele kørt sekventielt. Dette resulterer i det lineære.

Hvis man skifter *var b* ud med en future, er man i stand til køre en parallel udregning af futureB, sådan at når man skal beregne *var f*, er man i stand til at udregne *f* hurtigere, eftersom at "var c" ikke skal vente på at "futureB" bliver færdig.

```
static void Main(string [] args)
{
    var = "A";
    Task<string> futureB = Task.Run(() => F1(a));
    var c = F2(a);
    var d = F3(c);
    var f = F4(futureB.Result, d);
    System.Console.WriteLine(f);
}
```



Hvornår er ens future klar?

- Hvis task'en er blevet kørt færdig, bliver resultatet kaldt, som om det er en variabel.
- Hvis task'en er i gang med at blive kørt, blokerer det funktionen som skal bruge resultatet.
- Hvis task'en endnu ikke er startet, bliver den sat inline i den givene kontext, hvis det er muligt.

> A task is something you want done.
>
> 203 A thread is one of the many possible workers which performs that task.
>
> In .NET 4.0 terms, a Task represents an asynchronous operation. Thread(s) are used to complete that operation by breaking the work up into chunks and assigning to separate threads.

Pipelines

Pipeline er et pattern…. (åbenbart)

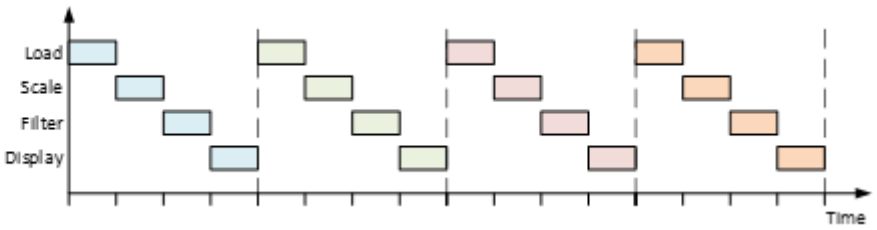
En pipeline paralleliserer processeringen af en sekvens af input værdier. En pipeline består af en serie af producer/forbruger trin. Hver af disse er afhængig af den forriges resultat. Kan bruges når der er for mange afhængigheder til at lave parallelle loops.

Processeringen inddeles i paralleliser-bare trin. Hvor output fra trin i er input til trin i+1. Ellers er trin uafhængige af hinanden.

For billede behandling kunne det være inddelt i fire trin:

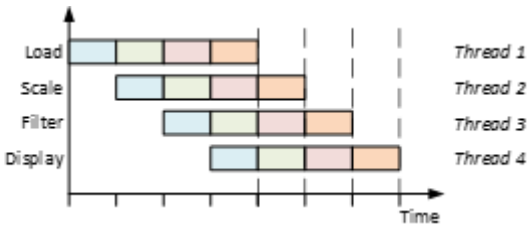
Her ses sekventiel behandling af billederne.

- Sequential (4 images)



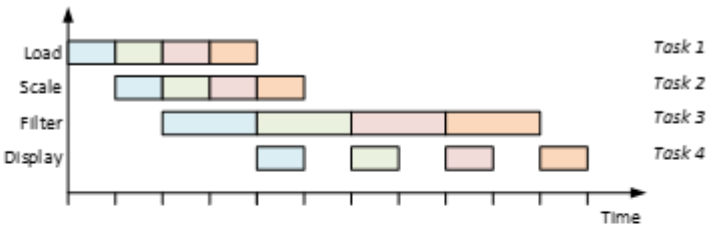
Her ses en pipelined udgave. Det blå billede skal stadig være færdig med at "Load" før den kan gå ned i "Scale" men når "Scale" for det blå billede bliver kaldt, så bliver "Load" for det grønne billede kaldt (parallelt)

- Pipelined – throughput > x 4

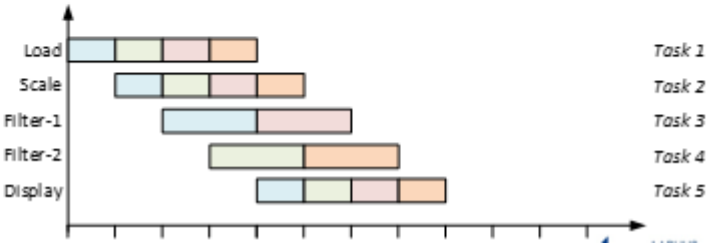


Hvad så hvis der kommer en flaskehals i pipelinen?

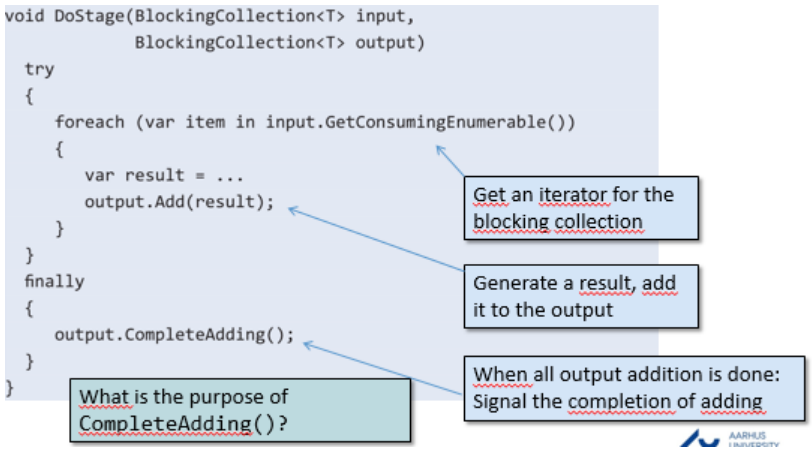
- Problem: Filter stage becomes bottleneck



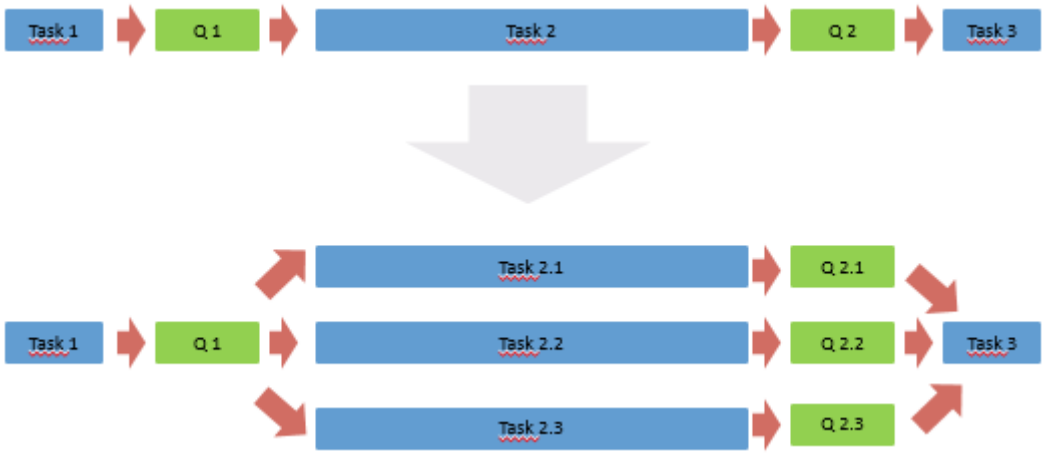
- Solution: Several Filter stages to feed Display



I c# laves det med tasks og concurrent queues (BlockingCollection<t>)



I C# vil det se sådan her ud hvis et trin har flere trin (tasks).

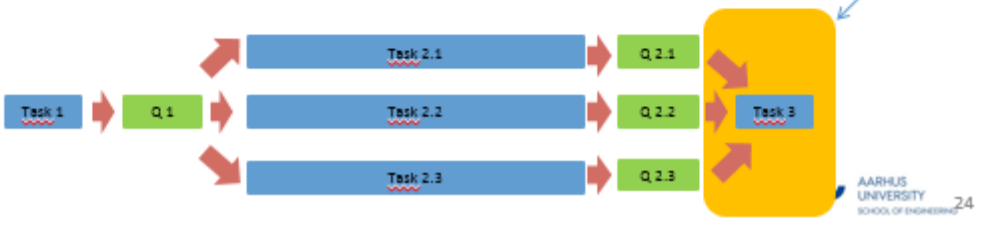


Her er det så forbrugeren der er ansvarlig for at tage ud af alle køerne.

```
private void ToLowerCase(BlockingCollection<string>[] inputs, BlockingCollection<string> output)
{
    var str = "";

    while(!inputs.All(bc=> bc.IsCompleted))
    {
        BlockingCollection<string>.TakeFromAny(inputs, out str);
        str = str.ToLower();
        output.Add(str);
    }
    output.CompleteAdding();
}
```

TakeFromAny() finds a "non-completed", data-bearing collection and retrieves data from it



AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

24