ISE

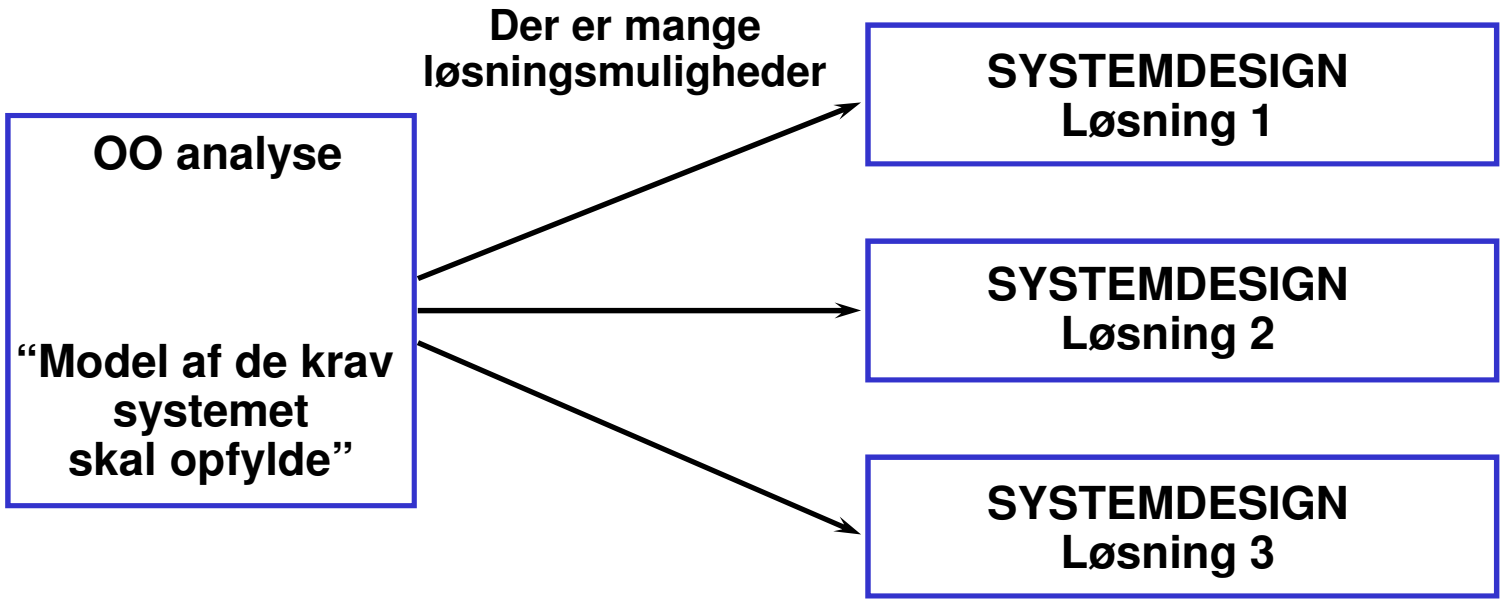
System Arkitektur Design

Disposition

- Fra analyse til design
- Arkitektur Design:
  - Bearbejdning af domæne- og applikations-model
  - Designprioriteringer
  - Opdeling i delsystemer og komponenter
  - Design principper – hvad er et godt design ?

Fra analyse til System Design (SD)

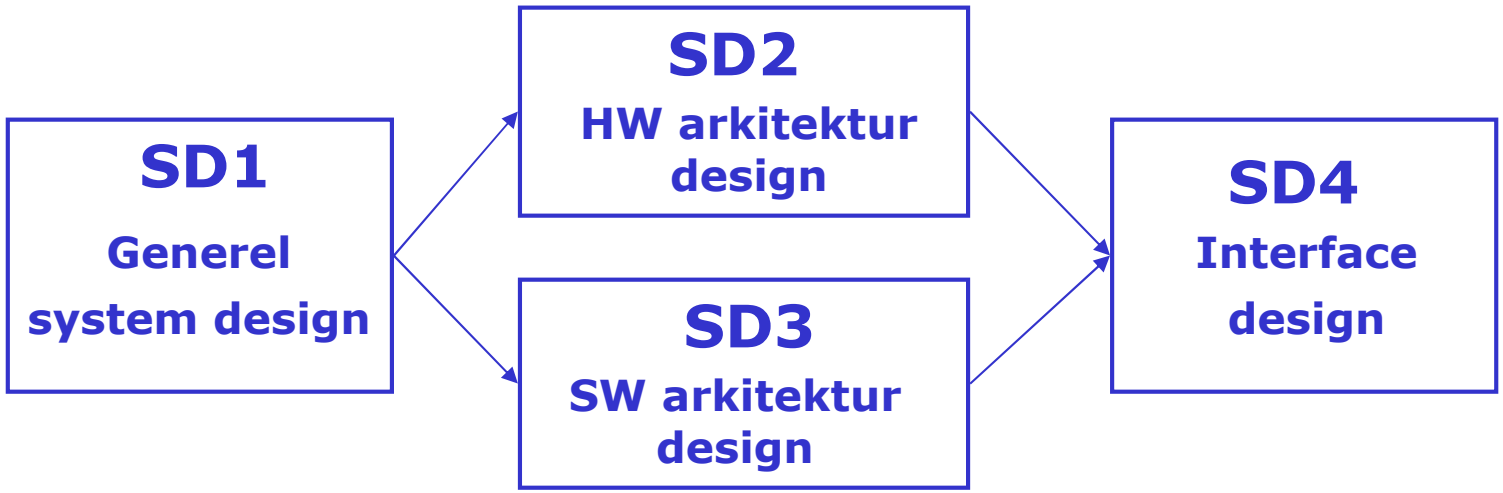
"HVAD" **"HVORDAN"**



Systemdesignerens opgave består i at skitsere de mulige løsninger og vælge den bedste af disse løsninger

System Design (SD) aktiviteter

- **SD1. Generel system design**
- **SD2. HW arkitektur design**
- **SD3. SW arkitektur design**
- **SD4. Interface design**
  - **specifikation af komponenternes interface**



SD1. Generel system design

Foretag følgende generelle systemdesign beslutninger:

A1. Fastlæg designprioriteringer

A2. Fastlæg strategi for arkitektur
(Opdeling i blokke – lagdelt – client/server)

A3. Fastlæg strategi for datalagring

A4. Fastlæg strategi for software kontrol

A5. Fastlæg strategi for opstart og nedlukning

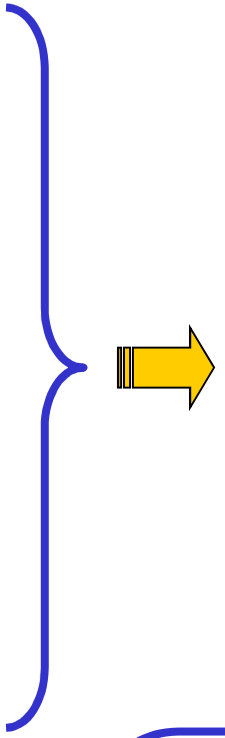
A6. Fastlæg strategi for fejlhåndtering

A7. Fastlæg selvtest- og backupfunktioner

SD1.A1: Fastlæg design prioriteringer

Prioritering af forskellige og ofte modstridende designkriterier

- **Optimering af performance**
- **Minimering af memory forbrug**
- **Minimering af strømforbrug**
- **Minimering af udviklingstid**
- **Minimering af pris**
- **Optimering af robusthed**
- **Optimering af sikkerhed**
- **Optimering af udvidelsesmulighed**
- **Optimering af flytbarhed**



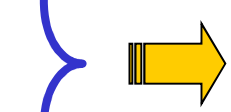
Strategi for:

- Arkitektur
- Design principper
- Infrastruktur

Teknologi til rådighed

Architecture Design Qualities (designkriterier)

- Performance (speed)
- Cost (price and time)
- Qualitiy
  - Usability
  - Efficeincy
  - Maintainability
  - Reusability
  - Reliability
  - Safety
  - Portability
  - Complexity
  - Adaptability
  - Testability
  - Understandability
  - Robustness
  - Learnability



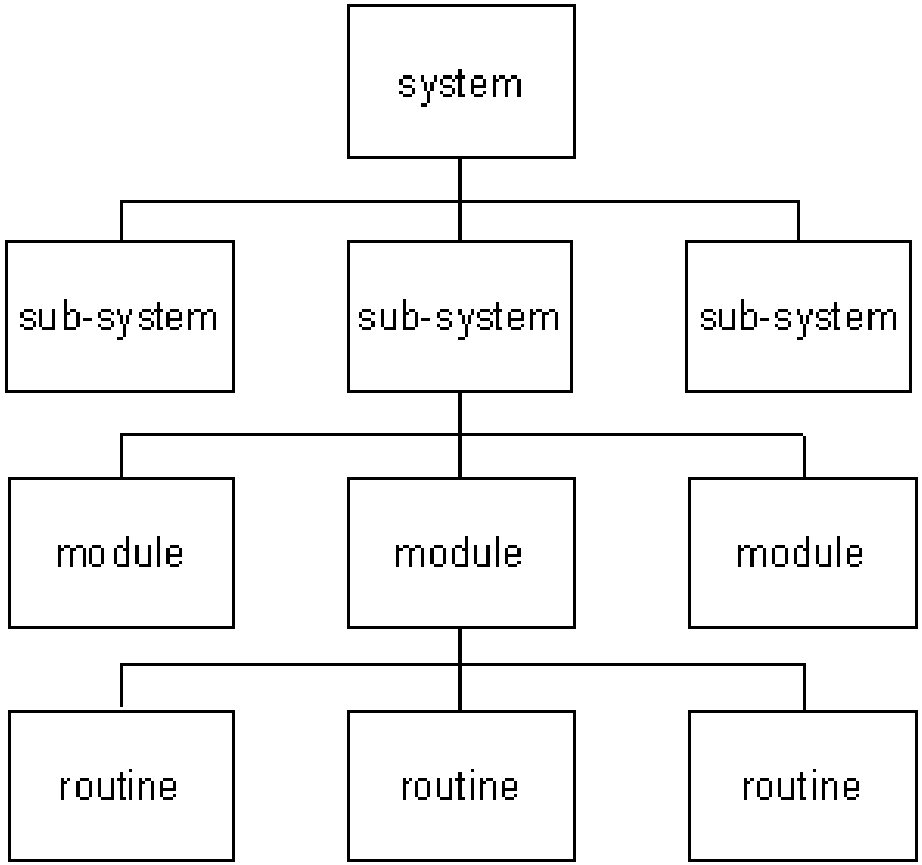
Vælg og prioriter

Architecture Design Principles (principper)

- Divide and conquer (Del og hersk - dekomponering)
- Increased cohension (Samhørighed i de enkelte komponenter)
- Reduce coupling (Lav kobling mellem komponenter)
- Abstraction (Gemmer information og kompleksitet)
- Reuse existing design (Mønstre)
- Ensure testability (Specificer og tænk på test)

Dekomponering

- Opdeling i delsystemer, komponenter, moduler, klasser, kredsløb og rutiner....



Cohesion (Samhørighed)

<< block >>

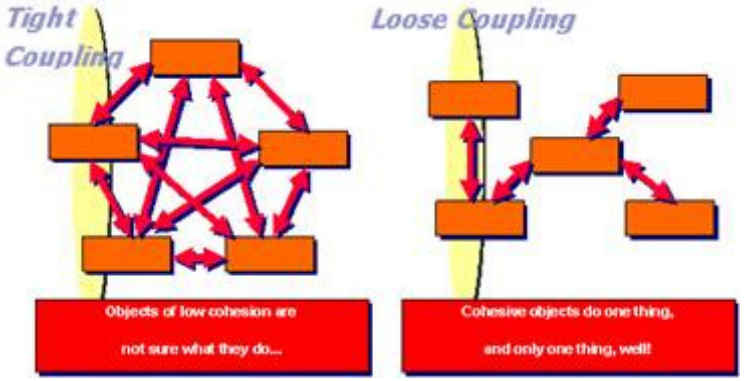
Internt i komponent

- Styrken hvormed funktioner relaterer til en given klasse eller komponent
- Vigtigt at der er stor samhørighed mellem de funktioner en given klasse eller komponent skal varetage
- Målet er at maksimere ”cohesion” og minimere ”coupling”

- Eksempel på forskellig slags ”cohesion”
  - ***Functional*** – all elements contributes to execution of a specific task
  - ***Sequential*** – output from one procedure becomes the input for the next
  - ***Communication*** – procedures that operates on the same set of input data
  - ***Temporal*** – unrelated activities that are sequential ordered in time
  - ***Logical*** – a number of procedures that are responsible for accomplishing a specific task

Coupling (Kobling/Binding)

- Estimat for hvor meget klasser/komponenter er afhængig af hinanden
- Tæt kobling betyder, der er kompleks og høj udveksling af information og kontrol imellem klasserne
- Desto laver kobling mellem klasserne, desto bedre er opdeling af designet
- Ulempen med høj kobling er:
  - Svært at debugge under udvikling
  - Svært at fejlfinde når systemet er sat i drift
  - Svært at vedligeholde
  - Svært at modificere designet med nye funktioner
- Råd
  - Fjern unødvendige afhængigheder
  - Minimer mængden af informations udvekslin
  - Brug ikke globale variable
  - Gør designet så simpelt som muligt



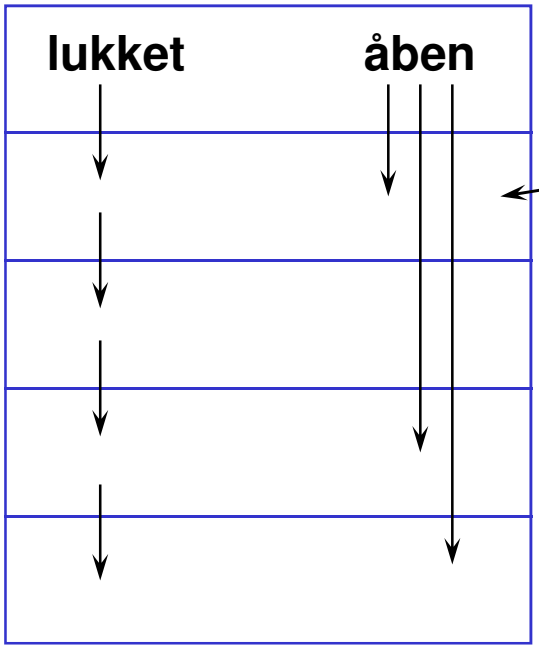
SD1.A2. Fastlæg strategi for arkitektur

- Fastlæg arkitekturprincipper ud fra prioriteringer og teknologi
  - Lagdeling
  - Udvikling af eller anvendelsen af Framework
- Fastlæg arkitekturmekanismer og etabler en domæneuafhængig infrastruktur
  - Eksempler: DataBase Management System (DBMS), Network
- Anvendelse af Arkitektur Patterns
  - Eksempler: Layers, Client/Server

Arkitekturens udformning og implementering afhænger af de valgte arkitekturprincipper og mekanismer

Arkitekturmodel: lag og partitioner

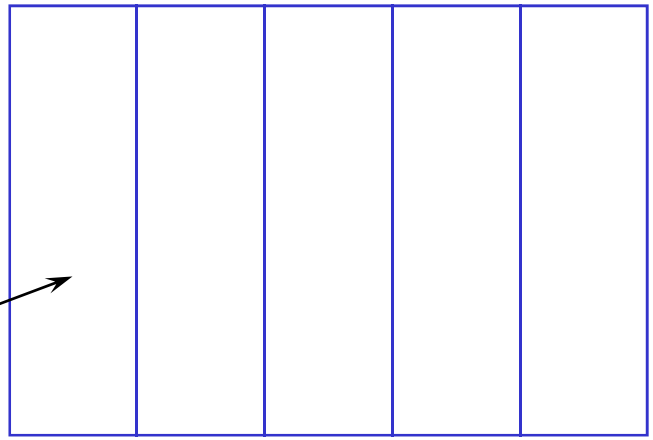
Vertikal opdeling i lag (*Layers*)



Virtual machines

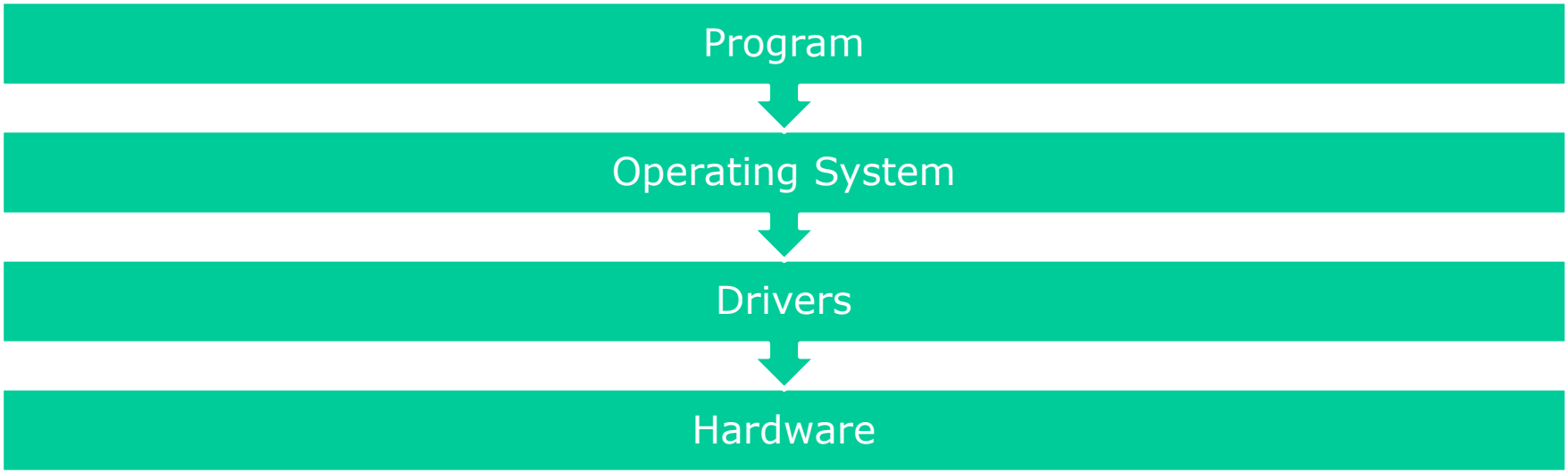
f.eks. OSI's 7 lags model

Horisontal opdeling i partioner (Partitions)

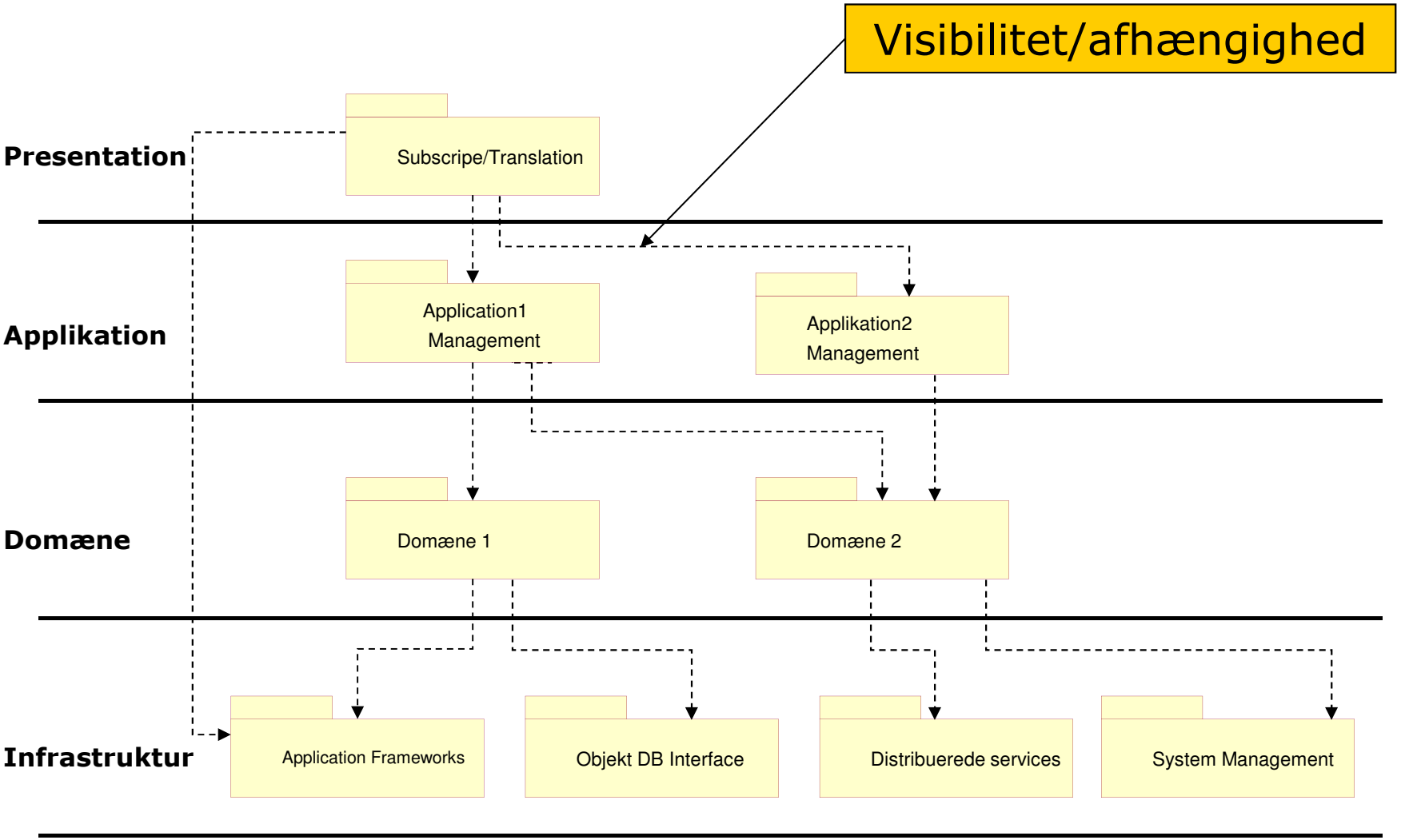


F.eks. et Operativ System lag opdelt i: *File System, Process control, Virtual memory management* og *device control*

Eksempel på et lagdelt system



Afhængigheder mellem lag

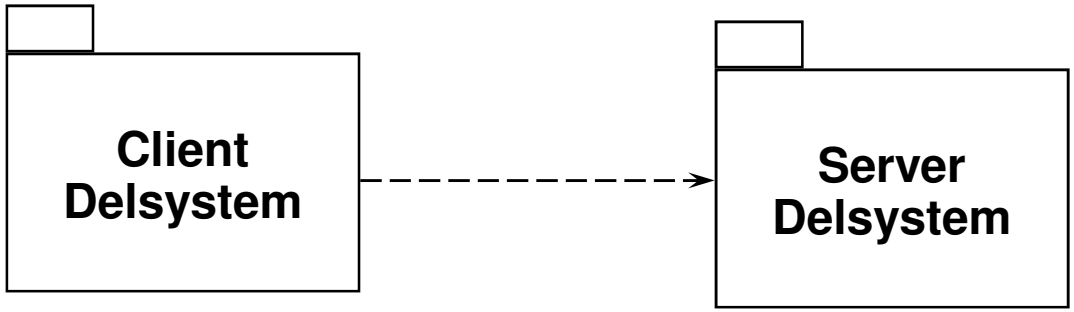


Opgave 1.

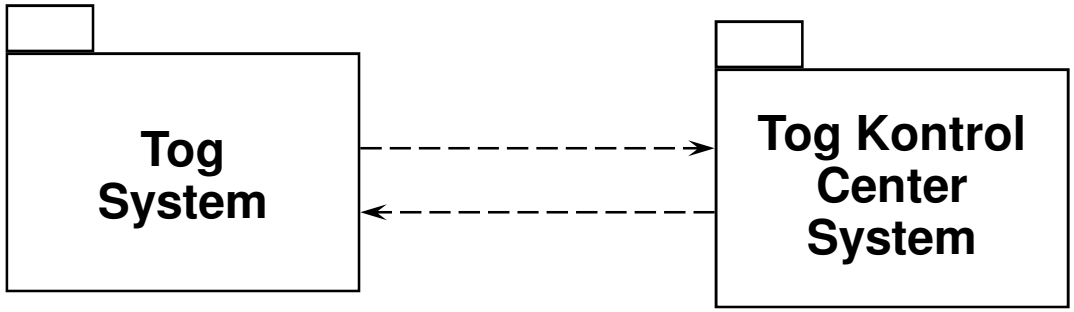
- Hvorfor er det vigtigt at minimere kobling mellem komponenter?
- Forklar begrebet logisk og sekventiel samhørighed?
- Hvordan forstår du ”abstraction” i forhold til den lagdelte arkitektur?
- Hvordan imødekommer en lagdelt arkitektur design principperne stor samhørighed ”cohesion” og lav kobling ”coupling”?

To typer af delsystemer

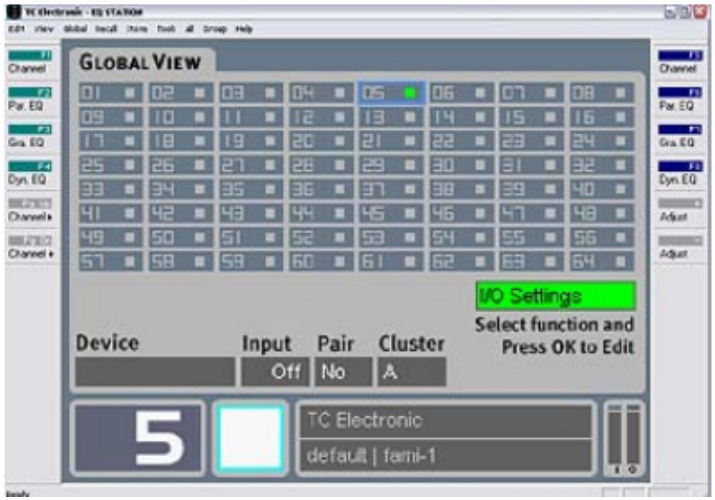
Client/server:

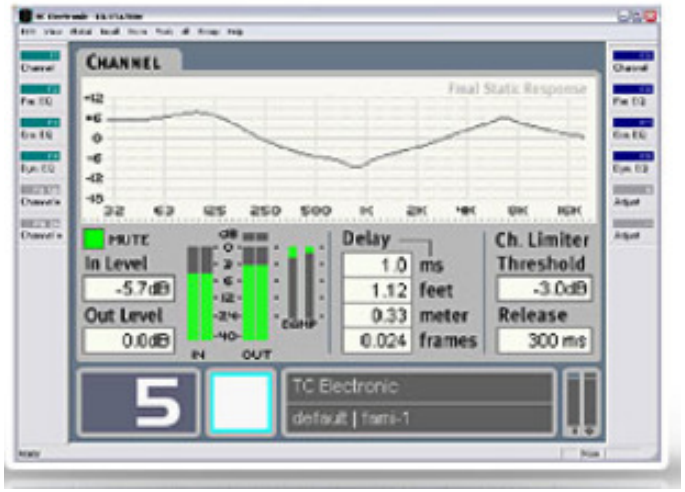


Peer to Peer:



Eksempel på Peer-to-Peer arkitektur

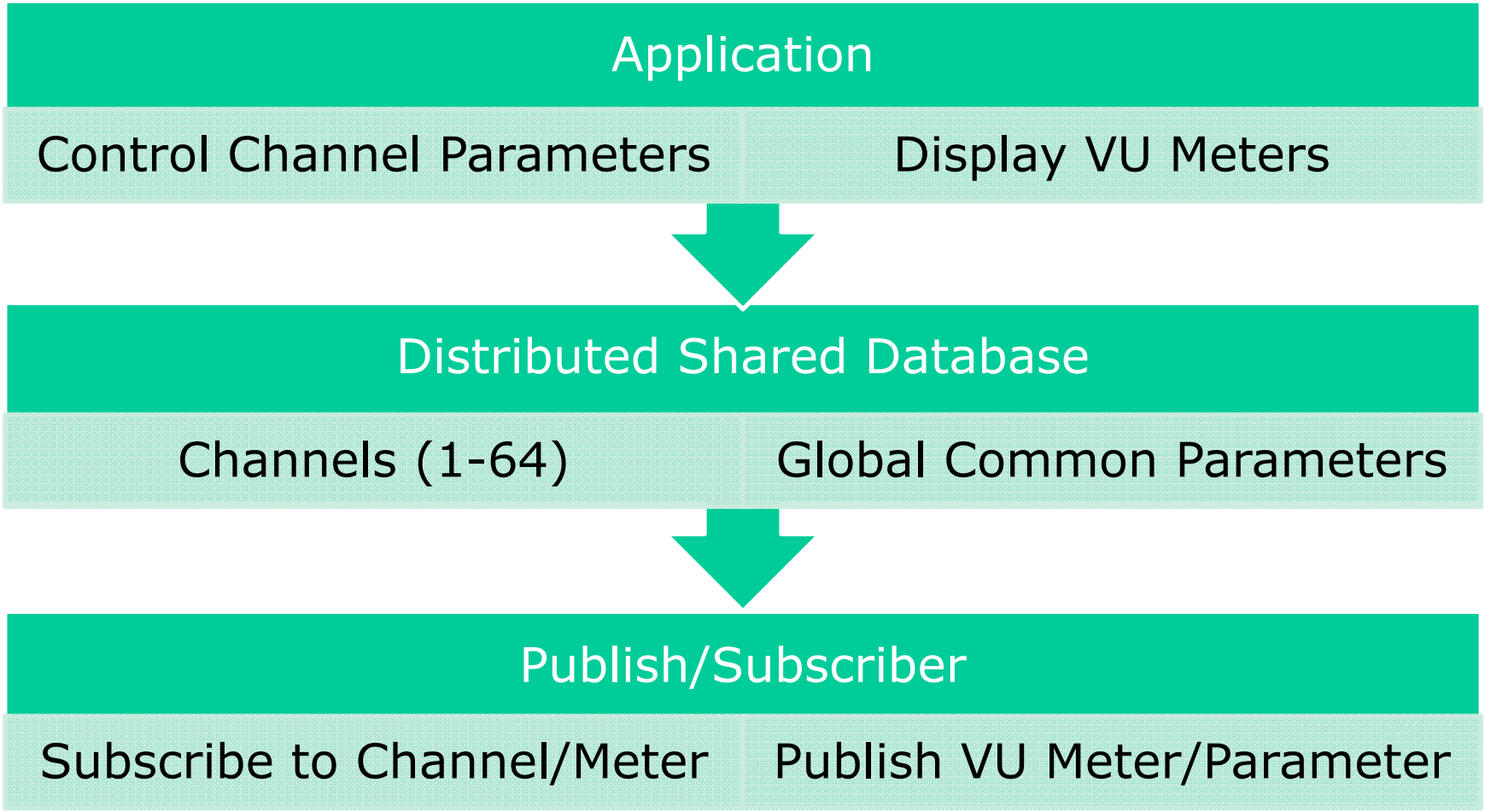




- Middlewaren benytter et publish/subscribe paradigm for peer kommunikation mellem EQ-Stations I et netværk
- Et netværk af EQ-Stations kan sammen håndtere op til 64 audio kanaler – styres også med tilsluttet PC

INGENIØRHØJSKOLEN I ÅRHUS

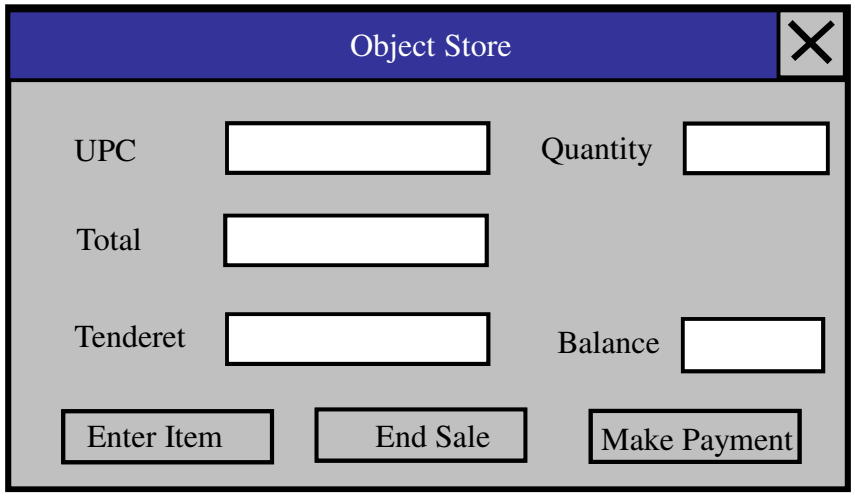
EQ-Station en lagdelt arkitektur



Three-Tier Arkitekturmodel

Presentation

<<boundary>>



Application

Logic

<<control>>

Record Sales

Authorize payments

Storage

<<domain>>

"Architectural Style" for apparatsystemer

Architectutral style eksempel for
Event kontrollerede kontinuerte processeringssystemer

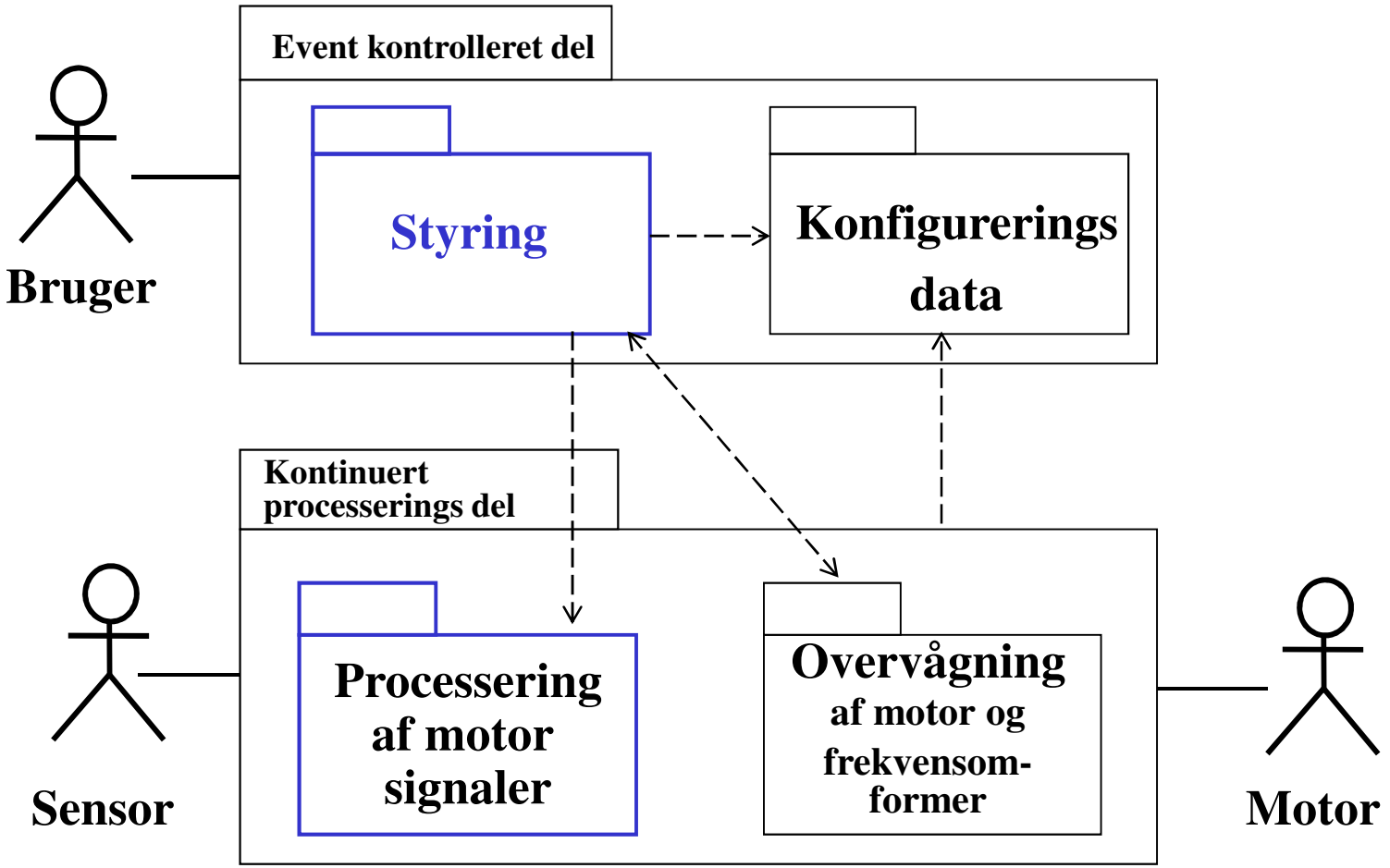
Eksempler på systemer:

- Et måleinstrument - f.eks. en støjmåler
- En frekvensomformer - f.eks. til styring af hastigheden for en motor
- En CD afspiller

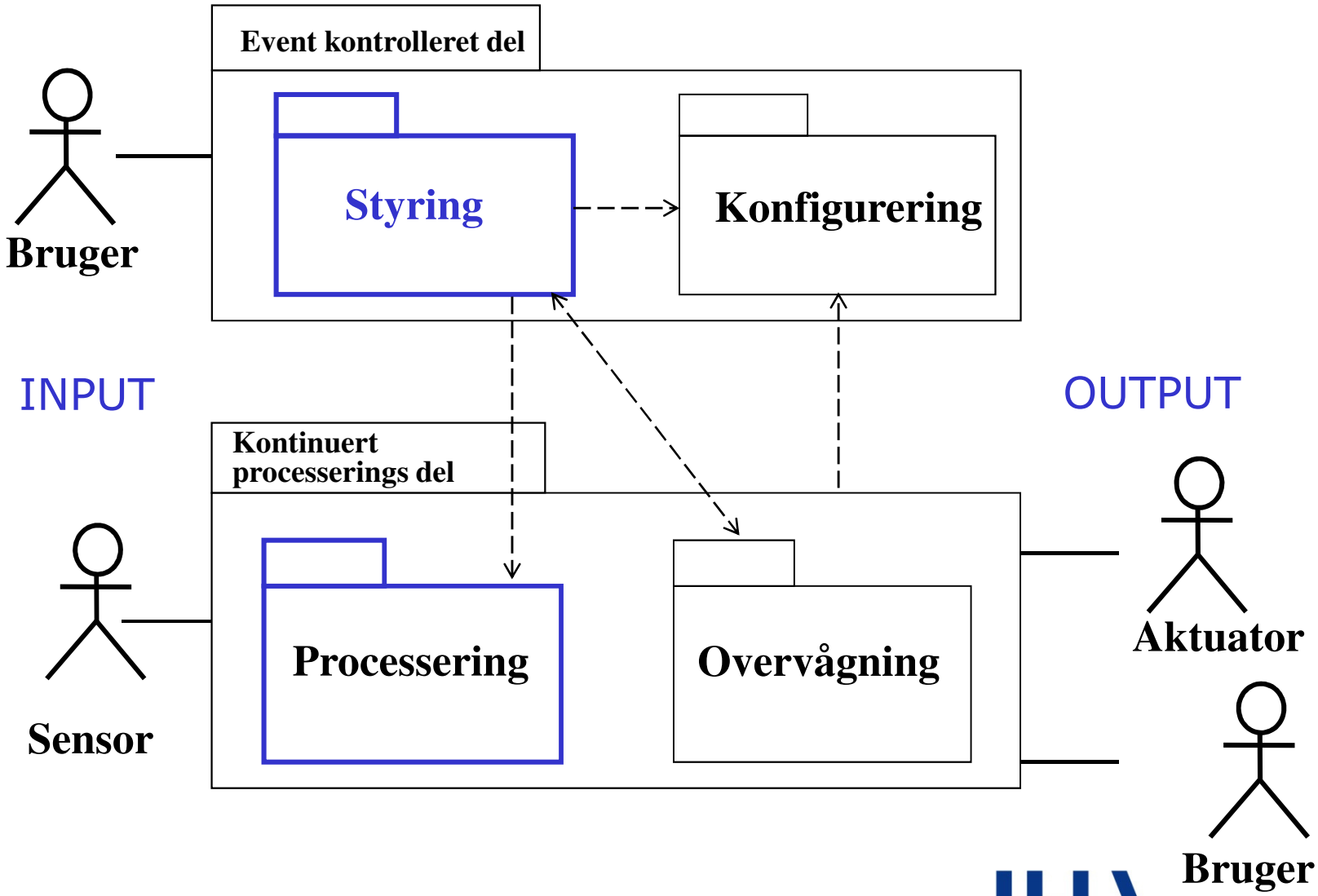
Løsningsmodel:

- Opdeling af applikationslaget i to adskilte dele: et diskret lag til håndtering af hændelserne (events) og et kontinuert lag til håndtering af den kontinuerete beregnings- og behandlingsdel

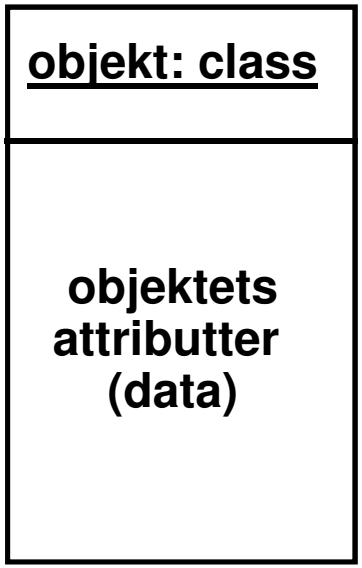
Eksempel på en frekvensomformer model



Architectural Style: todelt arkitekturmodel



SD1.A3: Fastlæg strategi for datalagring



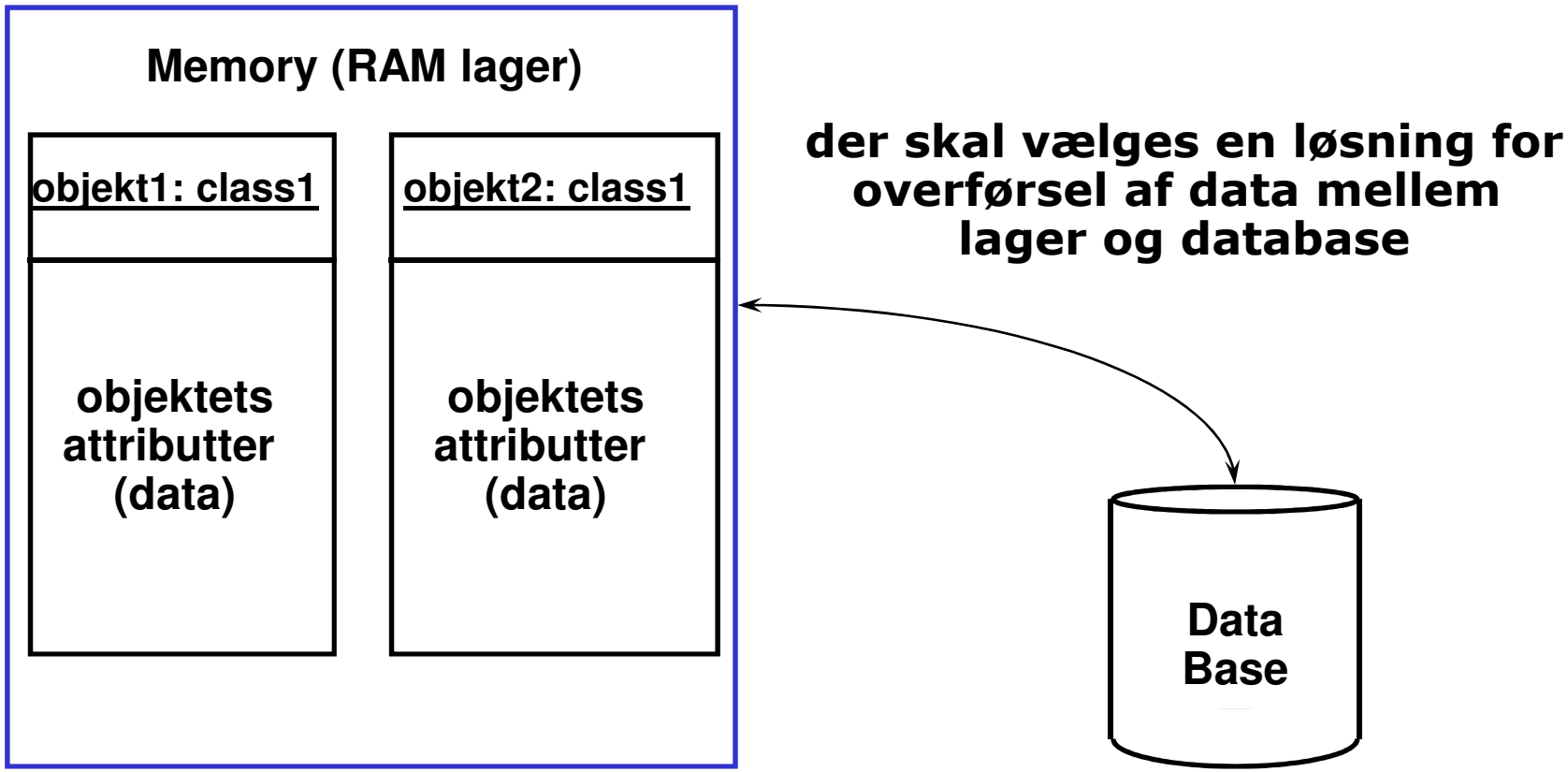
Et objekts data kan befinde sig i:

- Memory (ram)
- File
- Database
- Lagringskreds (eeprom)

Databasen kan være en:

- Simpel database (flad fil)
- Relationel database

Permanent datalagring (persistens)



Lagring i en file

Retningslinier for datalagring i en file:

- Data der er volumeniøse men svære at strukturere (f.eks. grafiske bitmaps)
- Data der volumeniøse med lav informations-tæthed (f.eks. logfile, debugging dumps)
- Rå data (f.eks. måleværdier fra en process)
- Flygtige data, der kun gemmes i kort tid (f.eks. alarmer)

SD1.A4: Fastlæg strategi for SW kontrol

SW kontrol kan opdeles i 3 forskellige typer:

- Procedure-drevet sekventielt system
  - Kontrollen er i programkoden
  - Programmet venter på et eksternt event og kalder koden
- Event-drevet system
  - Kontrollen er i operativsystemet
  - Operativsystemet kalder koden direkte for et givet event
- Concurrent system
  - Kontrollen er i et antal uafhængige task
  - Hvert task kan enten køre eller vente på et event og bliver gjort kørerklart af operativsystemet
  - Interrupt funktioner er en slags selvstændige tasks

SD1.A5: Fastlæg strategi for opstart og nedlukning

- System opstart
  - Hvorledes systemet initialiseres og konfigureres
  - Rækkefølgen programmer eller komponenter skal opstartes i
  - Hvem der har ansvar for oprettelse af fælles objekter
  - Etablering af kommunikation mellem processorer og task

- Systemet nedlukning
  - Allokerede resurser skal frigives
  - Rækkefølgen programmerne skal nedlukkes i
  - Skal systemets status og opsætning gemmes

SD1.A6: Fastlæg strategi for fejlhåndtering

- Design af systemets opførsel ved fejl ?
  - Skal systemet f.eks. køre videre med begrænset funktionalitet
  - Skal der indføres redundante processorer ?

- Design af en fælles fejlhåndteringsstrategi for hele systemet

SD1.A7: Fastlæg selvtest- og backupfunktioner

- Da vi har valgt HW resurser i systemdesignet har vi introduceret komponenter der kan fejle !
  - Citat fra en Unix manual:

    ”any hardware can fail at any time”

- Der skal derfor tages stilling til om der skal implementeres:
  - *Selvtest software*, der kan teste vigtige dele af systemets komponenter - ved opstart og løbende under drift
  - *Software til backup* af vigtige data (fra filer eller database)

Opgave 2.

- Fastlæg en strategi for arkitekturen af flaskeautomaten?
  - Hvordan kan arkitekturen understøtte designkriterieret: testbart?
- Fastlæg en strategi for fejlhåndtering af flaskeautomaten?
  - Er det muligt med en fælles fejlhåndteringsstrategi ?
- Fastlæg en strategi for selvtest af flaskeautomaten?
  - Hvad skulle en selvtest omfatte?