

Arkitektur og kommunikation

Når flere applikationer skal samarbejde til at ligne og opføre sig som én applikation.

Agenda

1

Det store billede

2

Udviklingen i opbygningen af applikationer (arkitektur)

3

Samarbejdet mellem applikationer

4

Udveksling af data mellem applikationer (XML, JSON)

5

Udfordringen hvis ikke vi taler samme sprog

6

Må vi have lov til at tale sammen?

7

Arkitektur set udefra (...og indefra)

8

Ordbog

Arkitektur og Kommunikation

- Jeg vil præsentere ord og begreber på en abstrakt måde
 - Relatere til “almindelige problemstillinger” – typisk udenfor sundhedsemnet
 - I starten taler jeg lidt omkring det helt store billede og en lille historiektion omkring hvordan virksomheder vokser samtidig med at teknologierne udvikler sig.
 - Vi kommer til at tale meget omkring “kommunikation” og vil sammenligne det med en gruppe (forskellige) mennesker skal tale sammen for at løse en opgave
- Spørg så snart der er noget som er uklart eller jeg ikke får forklaret det på en god måde

Arkitektur og kommunikation

- Jeg vil bruge nogle ord som betyder det samme:
 - **Program** (fx. Word, Faktureringsystem, Bookingsystem)
 - **Applikation** (en softwarevirksomhed har udviklet en applikation)
 - **App** (bare en forkortelse af applikation, men associeres ofte med mobiltelefoner)
 - **System** (bruges typisk til at signalere at flere applikationer kan arbejde sammen og levere indhold – fx. Hente data fra eksterne applikationer. Dvs. En applikation af afhængig af andre applikationer for at kunne levere)

Det store billede

Det store billede

- Vores arbejdsliv er i høj grad opbygget omkring et eller flere IT systemer.
- Uanset hvilken branche vi befinder os i, så er vi højere grad afhængig af vores IT systemer.
- Opbygningen af systemer tager ofte udgangspunkt i “domænet” (det vil sige, den del af virkeligheden vi prøver at modellere)

Det store billede

- Vi arbejder i højere grad sammen på tværs af fagligheder – det vil mange gange også sige at vi arbejder på tværs af systemer (hver afdeling/faggruppe/afdeling)
- Vi oplever også at der nogle gange opstår en “konflikt” i dette samarbejde på tværs.
 - Hvert system har en forskellig “holdning” til hvordan et specielt “domæneobjekt” ser ud
 - Fx. Kunde, borger, virksomhed, sag, kursus
 - ...når vi så, på tværs, refererer til en borger (eller andet) kan der opstå forvirring omkring hvad vi mener en borger omfatter i den kontekst (vi har vores eget udgangspunkt)

Det store billede

- Er der i virkeligheden en “konflikt” i betydningen af vores ord?
 - (Semantik)
- Måske er det systemerne som driller os – med deres begrænsninger?
- Husk: forskellige systemer, “samme opgave” – mange afdelinger.



Det store billede

- Hvis vi så drager en parallel til det at computersystemer skal kommunikere sammen
 - De taler forskellige sprog (fx. Dansk, engelsk, fransk, italiensk osv....)
 - De har ikke samme alfabet (latinske alfabet, kyrilisk, arabisk, kinesisk osv...)
 - Vi skal også stave korrekt (syntaks)



Det store billede

- Formålet er at opbygge et “Fælles sprog”, således vi kender bade betydning (semantikken) og hvordan det skrives (syntaksen)
- Det fællessprog skal være entydigt og have elementer fra begge sider
 - Ikke ren IT begreber og ikke rene sundhedsbegreber (Arabisk/Fransk)
- Menneskerne i dette skal bruge dette fælles sprog
- IT Systemerne bruger same principper omkring kommunikation – her har vi bare en masse tekniske begreber (som vi afdækker I dette forløb)
 - IT har klare regler for kommunikationen (indenfor IT kalder vi det protokoller og standarder)
 - Protokol = Hvordan taler vi (håndtryk, høflighed, ikke afbryde, give information)
 - Standarder = Hvad er reglerne for det vi leverer (fx. Altid et sagsnummer på 8 tal, en reference I et brev, et cpr nummer osv.)

Det store billede

- Opsummering
 - Kommunikation er komplekst
 - Vi mener det samme, men udtrykker os forskelligt
 - For at løfte vores opgave skal vi kunne kommunikere
 - Vi har nogle regler for hvordan vi kan/må kommunikere (det gør det nemmere?)
 - Vi skal have et fælles sprog for vores kommunikation
 - Menneskers problemer med kommunikation er ikke langt fra IT Systemernes problemer med kommunikation (her er der “bare” en masse tekniske begreber også)

Samarbejdet mellem applikationer - formålet

- Formålet med at have applikationer til at arbejde sammen er at fungere som eet sammenhængende system.
- Et sammenhængende system i én sammenhængende virksomhed
- Et system som samler det hele
- Ingen applikationer er udenfor, ingen ø'er



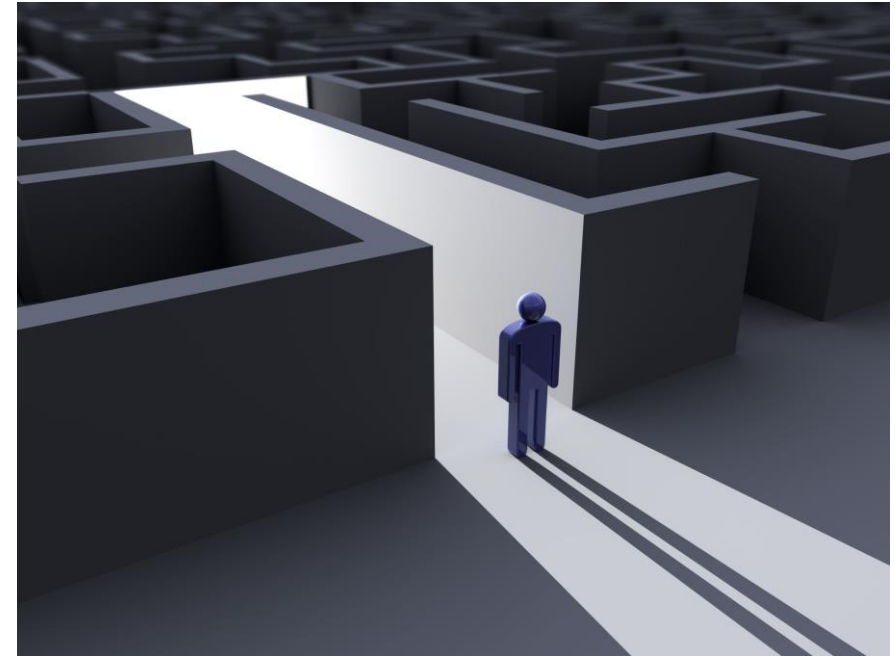


Er der spørgsmål omkring “Det store billede”?

Hvad er formålet med at kommunikere mellem systemer?

Formålet

- Man vil gerne lave virksomhedsarkitekturen "kundecentrisk" - således at ens kunder ikke opdager at der er 200 forskellige systemer
- Når man ringer til "kundesupport" den ene dag og til "bogholderiet" den anden dag, så kender begge til din sag.
 - De bruger forskellige systemer, men systemerne kan samarbejde.
- Kun én indgang...



Formålet

- Man vil også gerne opnå "strategy - IT alignment" - det vil sige at IT **aktivt** skal bidrage til og understøtte den strategi man har
 - Hvis man vil have verdensklasse support, skal man også prioritere dette
 - Hvis man vil have kundeorienterede systemer skal man også indkøbe (og bruge penge) på at købe systemer som kan samarbejde.
- Systemer, samarbejde, kommunikation skal alt sammen være en forlængelse af virksomhedens strategi.



Formålet

- Det gode arbejdsliv er også vigtigt, føle at systemerne ikke bremser ens arbejde.
- Kunder som er glade (...eller ikke sure) over at man har et overblik og rådgiver dem godt.
- Man arbejder med systemer som gør komplekse opgaver nemmere og har et sikkerhedsnet fra at lave fejl.



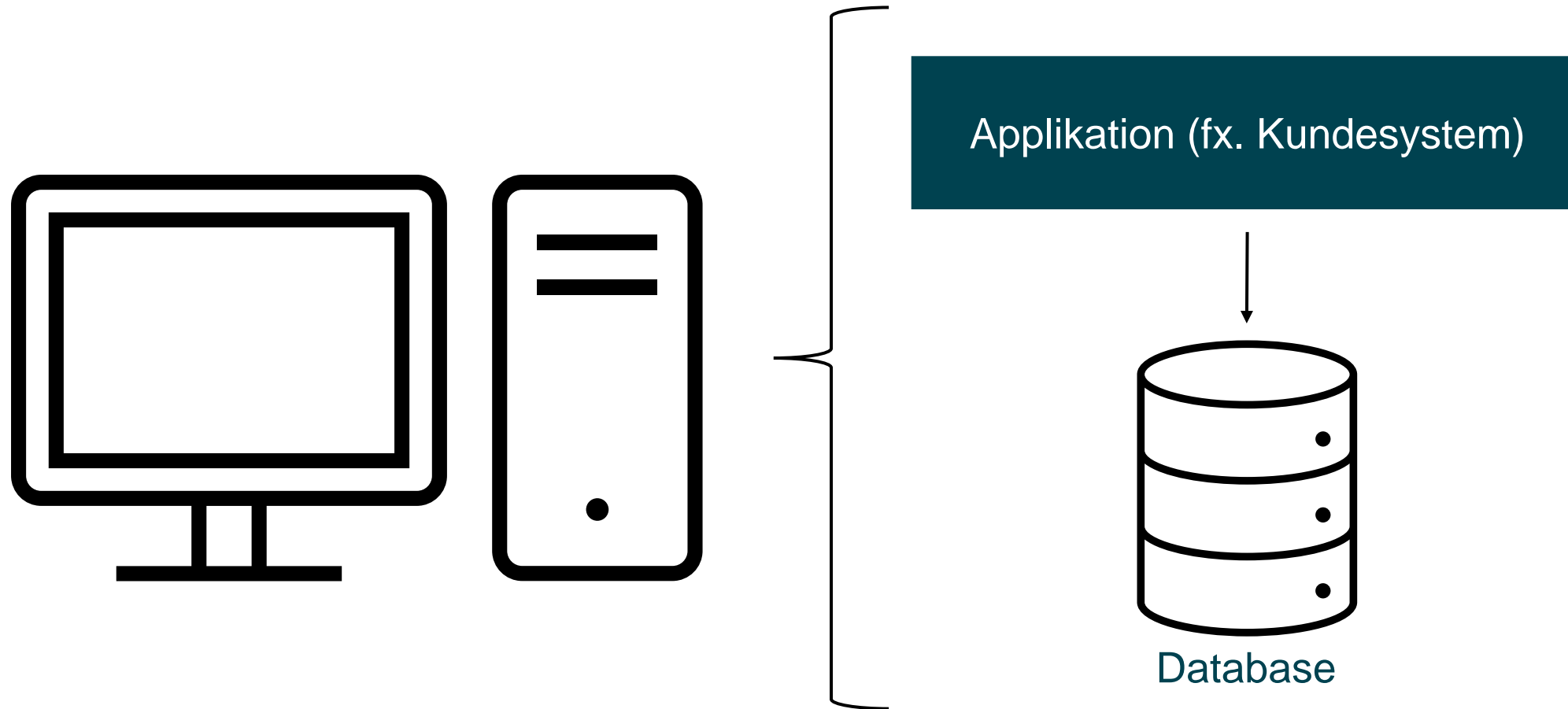
Udviklingen i opbygningen af applikationer

- en historielektion

Udviklingen i opbygningen af applikationer

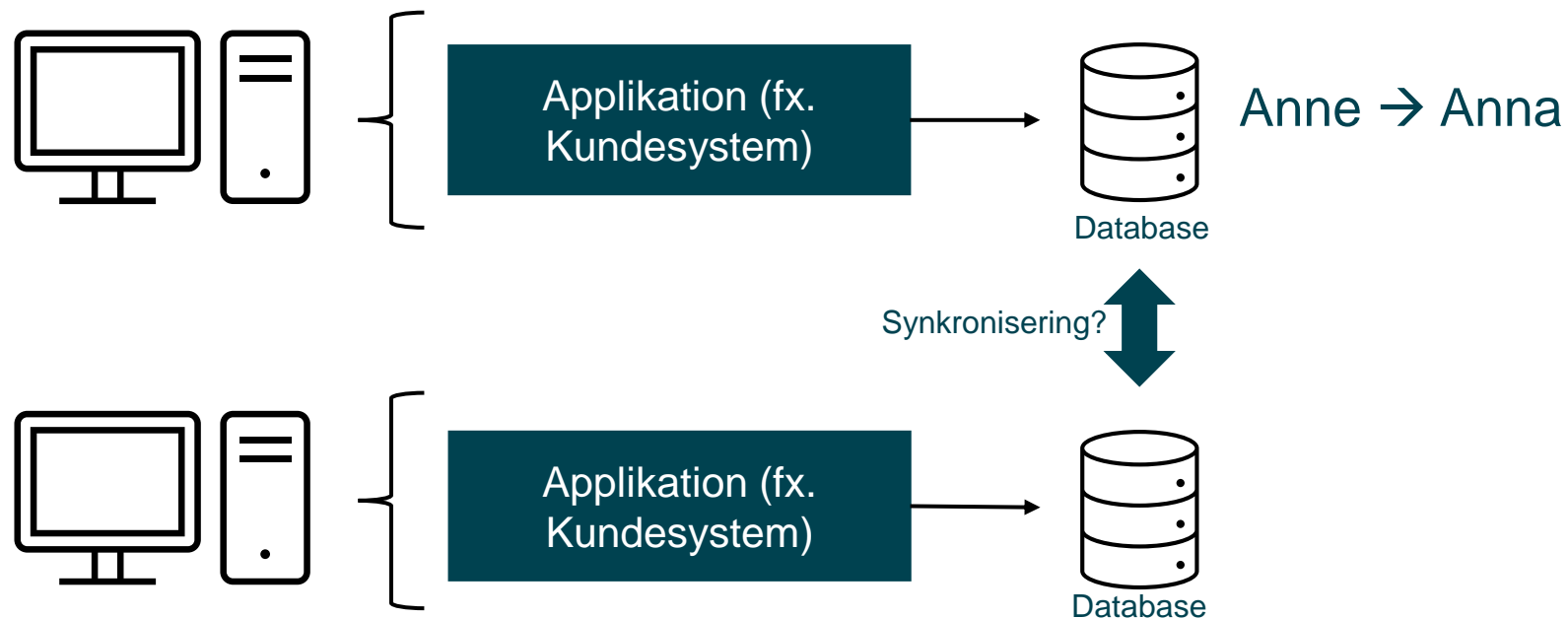
- De fleste applikationer er, historisk, opbygget til at være selvstændige, suveræne applikationer
 - Een opgave, der skal løses (fx. Fakturering, projektstyring, osv.)
 - Applikationen hører til i een afdeling, eller til een medarbejdergruppe (I mindre virksomheder er dette måske kun een person)
 - Måske kører denne application kun på din egen computer.
 - En application består typisk af 2 ting.
 - Selve programmet og så dataopbevaring (database)
 - En database er (typisk) kun for at du kan slukke din computer, når du går hjem. Når du åbner den om morgenen - så har du stadig data tilgængelig.

Udviklingen i opbygningen af applikationer



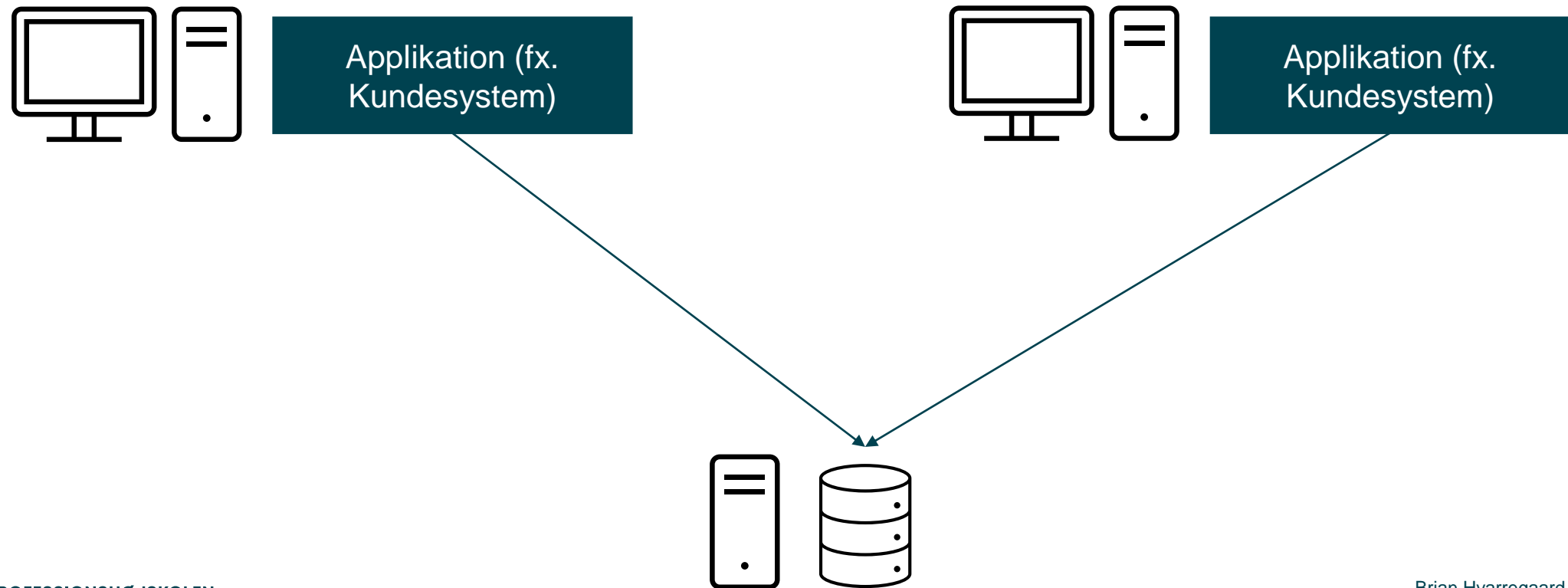
Udviklingen i opbygningen af applikationer

- Udviklingen vil så at flere personer skal bruge den samme application



Udviklingen i opbygningen af applikationer

- Ny model: “Distribuerede systemer” (vi arbejder på tværs af computere)

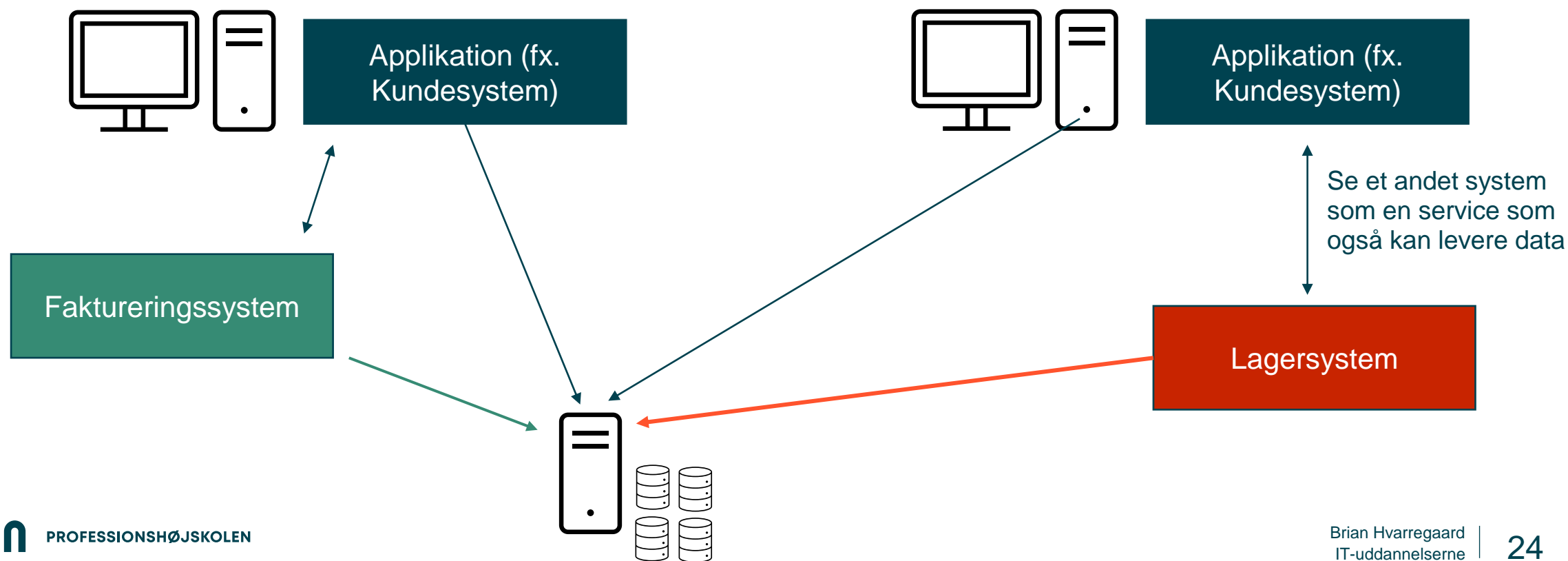


Udviklingen i opbygningen af applikationer

- Efterhånden som organisationer vokser og hver organisation får flere og flere forskellige applikationer til at understøtte opgaver – så stiger kompleksiteten også.
- Det tidligere eksempel tog udgangspunkt i at alle medarbejdere bruger “samme system”
- Hvad så når vi begynder at arbejde sammen på tværs af afdelinger og systemer?

Udviklingen i opbygningen af applikationer

- Ny model: “Serviceorienteret Arkitektur” (vi arbejder på tværs af systemer – systemer ser hinanden som noget, der kan levere en service til at udføre en opgave – til det fælles bedste)

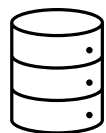


Udviklingen i opbygningen af applikationer

- Lad os forsimple tegningen en smule
- Vi ved at alle applikationer bruger en database for at kunne levere værdi
- Lad os fjerne det fra tegningen og kun se på hvad der er tilbage – bare for at få et enklere overblik.
- Samtidig fjerner vi også lige “server”, “computer” og “database”



Computer



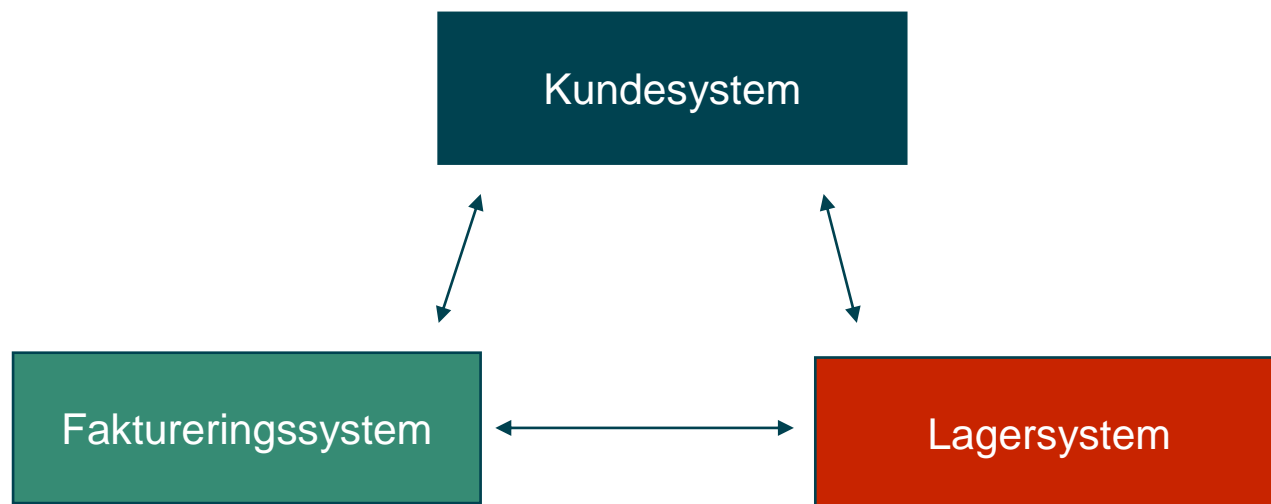
Database



Server

Udviklingen i opbygningen af applikationer

- “Serviceorienteret arkitektur”



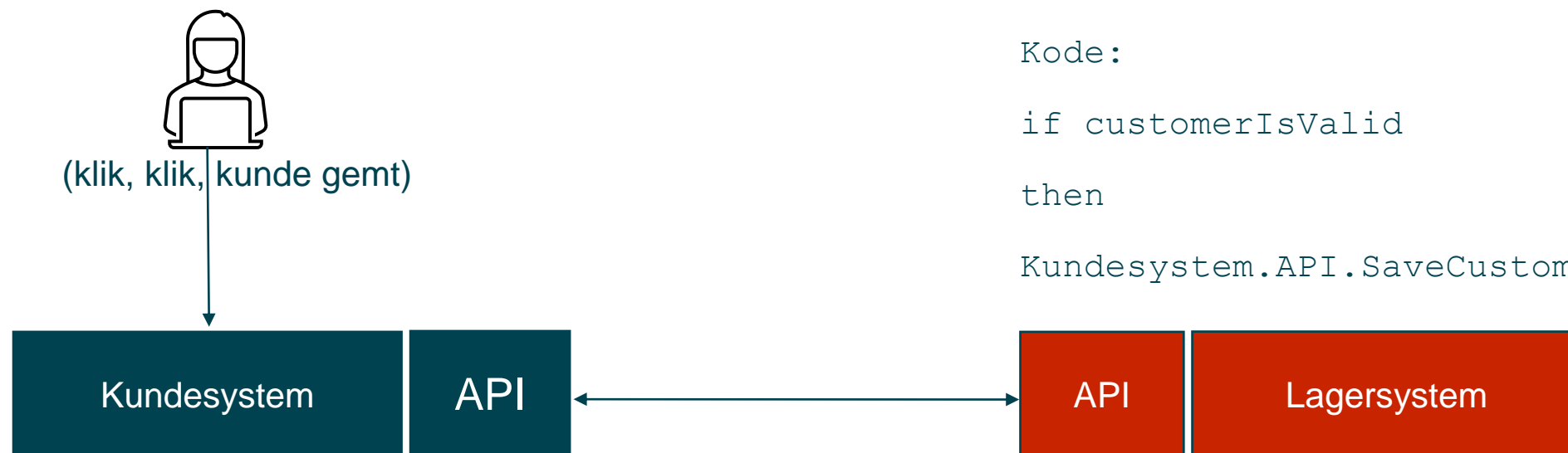
- Når vi “Tegner” diagrammer som dette kalder vi det ofte for en arkitektur.
- Det er en oversigt over hvordan vi har opbygget systemet
- Det er et resultat af en bevidst beslutning omkring sammenhæng for at understøtte opgaverne

Udviklingen i opbygningen af applikationer

- Vi vil gerne kunne kommunikere mellem alle vores applikationer.
- Ikke alle applikationer er bygget til at man kan kommunikere med det udefra, det kræver et helt særligt "håndtag"
- Det "håndtag" er et håndtag, hvor en anden applikation kan interagere med et system og bede det om at udføre handlinger – i stedet for en bruger.
- I de fleste moderne applikationer kommer dette med fra starten af og en stor del af værdien i et sådan system kommer fra dette "håndtag".
- Det er designet til at udføre de samme handlinger en en menneskelig bruger

Udviklingen i opbygningen af applikationer

- Rent teknisk kalder vi dette for et "Application Programmable Interface (**API**)" - det vil sige en snitflade hvor jeg kan "programmere" min applikation – dvs. Styre den.

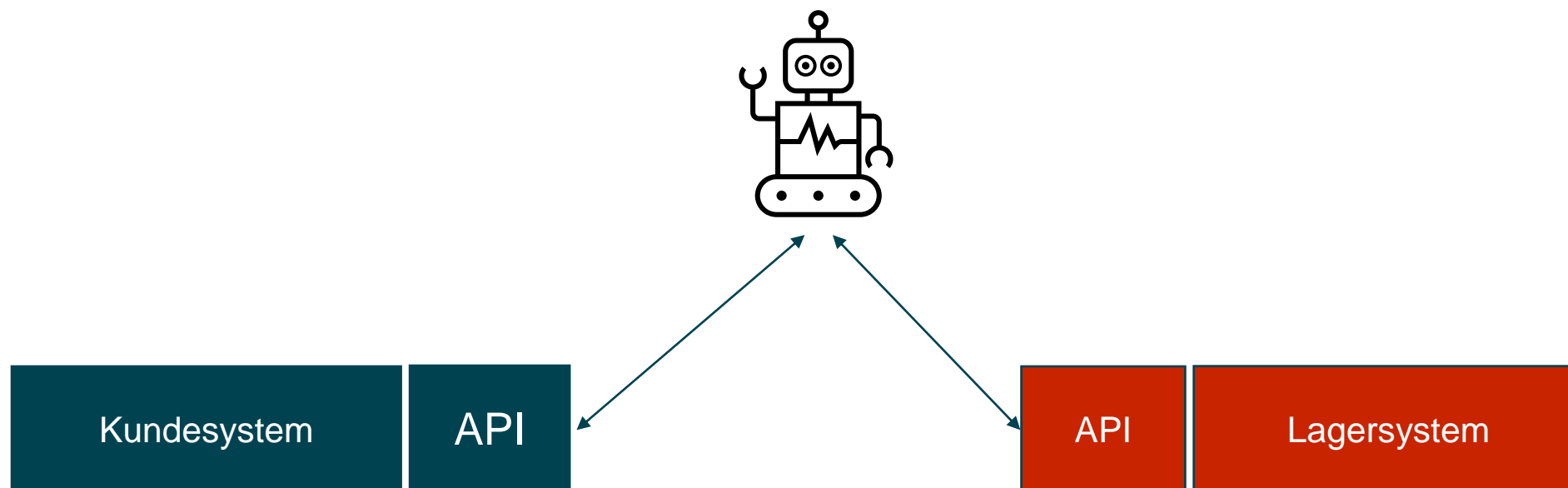


Kode:

```
if customerIsValid  
then  
Kundesystem.API.SaveCustomer (...);
```

Udviklingen i opbygningen af applikationer - sidenote

- Moderne "Robotic Process Automation (RPA)" bruger også disse API og kan programmeres uden at bruge kode. Her kan man ved peg-klik få systemer til at snakke sammen



Udviklingen i opbygningen af applikationer

- Et API kan typisk de samme ting (eller flere) end en bruger kan. Det vil også sige at det typiske API er stort og komplekst

Kundesystem

API

```
CreateCustomer();  
GetCustomer(id);  
UpdateCustomer();  
DeleteCustomer();  
ExistsCustomer();  
ArchiveCustomer();  
ValidateCustomer();  
GetCustomerCreditScore();
```

- I praksis kan det se således ud

UCN Demo API v1 OAS3

<https://localhost:7049/swagger/v1/swagger.json>

En demo af et API

[Terms of service](#)
[Example Contact - Website](#)
[Example License](#)

Customer

POST /Customer

Parameters

No parameters

Responses

Code	Description	Links
200	Success	No links

Try it out

GET

 /Customer

PUT

 /Customer

DELETE

 /Customer

GET

 /Customer/exists

PUT

 /Customer/archive

GET

 /Customer/validate

GET

 /Customer/creditscore

Udviklingen i opbygningen af applikationer

- Opsamling
 - Nye begreber: Arkitektur, Klienter, Server, Databaser, API
 - Den måde vi bygger applikationer på har ændret sig
 - Vi ser i højere grad et samspil mellem forskellige applikationer
 - Drømmen er at skabe mere værdi ved at samarbejde end hvert system kan for sig selv (synergi – $2+2=5$)
 - Mennesker bruger mus+tastatur til at udføre opgaver
 - Computere bruger et API til at udføre de samme opgaver som et menneske bruger mus+tastatur til



Er der spørgsmål omkring “Udviklingen i opbygningen af applikationer”?



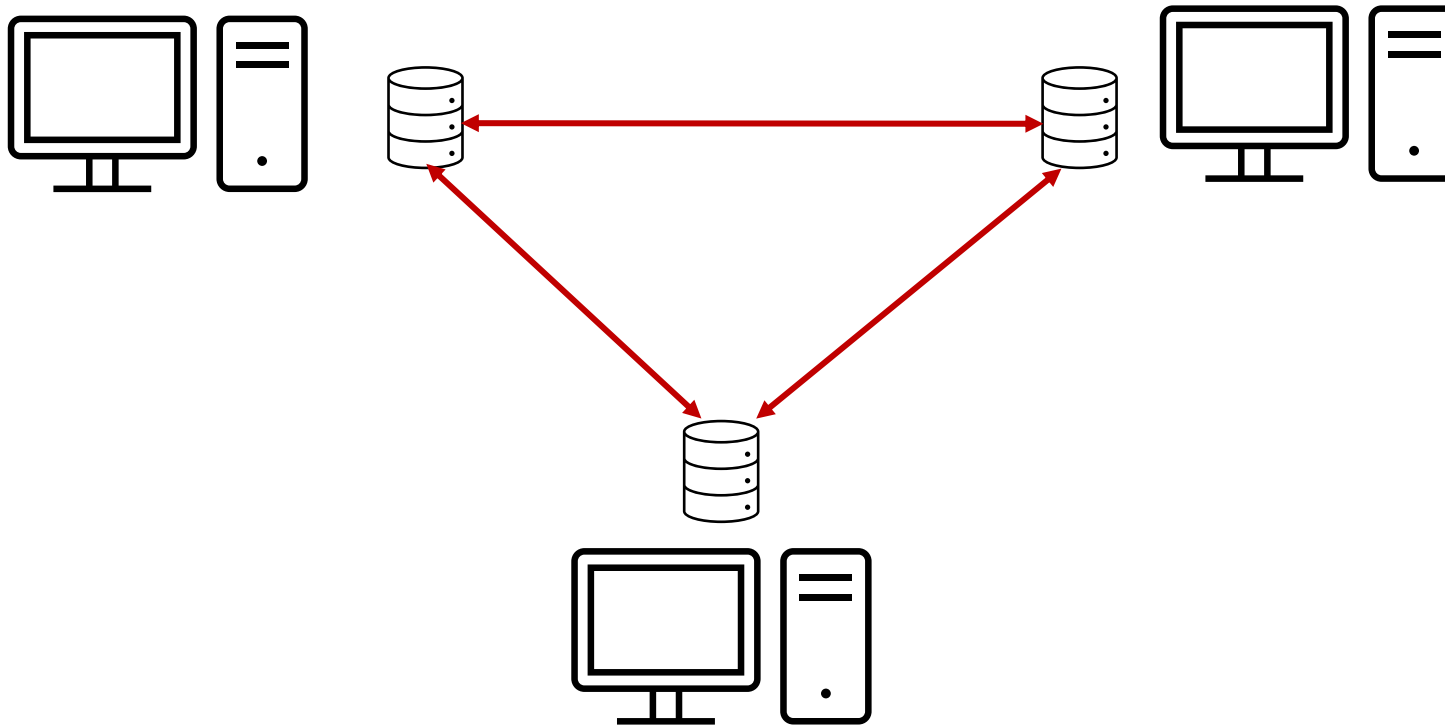
Samarbejdet mellem applikationer

Samarbejdet mellem applikationer

- Vi er nået til et punkt, hvor vi i vores daglige arbejde bruger mange forskellige applikationer
- Vi er forskellige professioner og bruger forskellige applikationer til at udføre arbejdet med
- Vi laver forskellige opgaver, mange gange involverer det dog de samme data.
- Vi oplever også at der kommer flere, nye systemer til
- Måske er der også behov fra eksterne interessenter i at "snakke sammen" med de applikationer og data vi har.

Samarbejdet mellem applikationer

- Den første tanke kan være at vi så "bare" skal hente data fra de andre systemers databaser:



Samarbejdet mellem applikationer

- I nogle tilfælde kan man bruge dette som sidste udvej – men det giver nogle problemer som vi ikke kan komme udenom, eller som vi ikke ønsker at komme udenom
- Kan vi "bare" pege en applikation ned i en anden applikations database og så kan vores applikation "bare forstå det"
 - Nej, desværre - det er ikke sikkert at "semantikken" mellem systemerne er ens og vi risikerer bare at ødelægge noget for den anden applikation.
 - Vi kan nok heller ikke rette i den del af applikationen – da det er en applikation vi har købt
 - Vi risikerer også at gå udenom alle reglerne som ligger i et API (fx regler om alder, køn, kundeforhold osv.) og så risikerer vi at gøre alt meget værre.
- Hvornår skal vi gøre det? Ved vi at der er nyt/ændret data at hente?
 - Nej, vi ser kun et øjebliksbillede af en database og ved ikke om noget er nyt, gammelt, ændret eller andet. Det ved kun applikationen og vi ser kun applikationens database.

Samarbejdet mellem applikationer

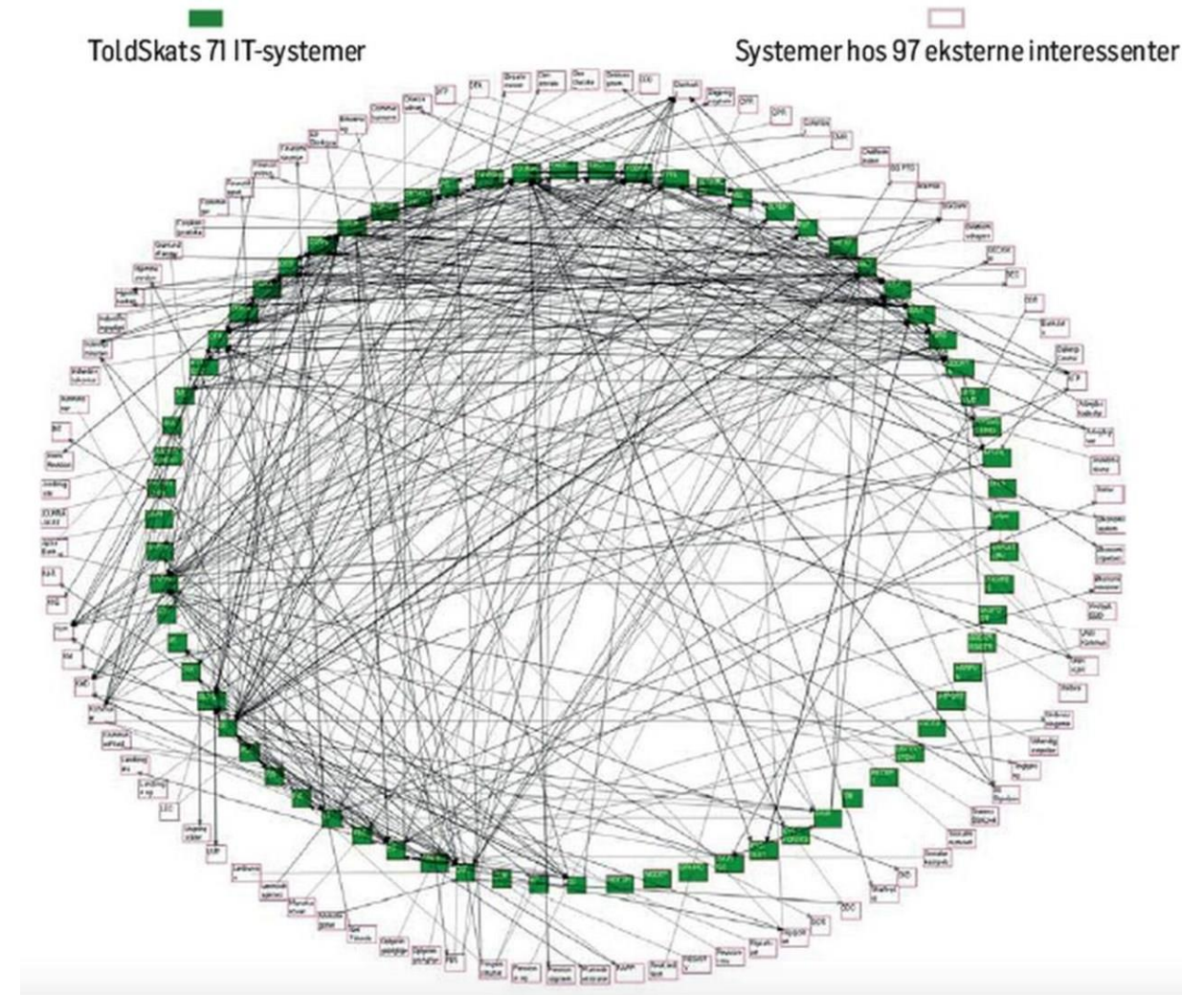
- Risikoen ved ikke at systemer taler sammen:
 - En kunde kontakter os ved at ringe på hovednummeret: "Jeg har skiftet adresse og flyttet fra København til Koldby"
 - Modtageren af denne telefonbesked opdaterer det i bogholderiet, således vi kan få den nye adresse på fakturaen.
 - 5 dage senere skal værers sælger på tur og besøge denne kunde, han slår op i kundesystemet og ser en adresse i København, hopper i metroen og tager på kundebesøg.
 - Der er bare ingen metroer i Thy
 - Vi har kun opdateret det ene system – hvem har overhovedet overblik over hvor mange steder man skal ændre adressen?

Samarbejdet mellem applikationer

- Et af problemerne her er overblik over data, over systemer samt over brugen af systemerne.

Samarbejdet mellem applikationer

- "Problemet" her er også kompleksitet
- Det er ret nemt at overskue når vi har få systemer, men når vi har 100 systemer, så skal hvert system kende til alle andre, kende deres data, hvordan den ser ud osv.
- Over tid vil det blive næsten umuligt at samarbejde mellem applikationer.



Samarbejdet mellem applikationer

- En applikation består groft sagt af 2 ting
 - En grænseflade til at læse og skrive data fra – det er nemmere for mennesker at bruge tastatur og mus til dette.
 - En database til at opbevare alle data i
- Når vi udfører vores arbejde, gemmer en kunde eller lignende, så afføder det en **handling**
- Handlingen indgår i en kontekst – fx "Gemme kunde".
 - Der er en handling, og så er der det data som er associeret med handlingen.

Samarbejdet mellem applikationer

- Denne handling er vigtig, da den "symboliserer" den interaktion der er med vores system og det job vi har udført.
- Den vil typisk være kendetegnet ved at indeholde et udsagnsord:
 - "Gemme", "Opdatere", "Læse", "Vise", "Slette", "Sende", "Flytte", "Arkivere"

Samarbejdet mellem applikationer

- Når en handling udføres vil vi gerne fortælle de andre systemer som er afhængig af dette om 2 ting:
 - Hvilken handling, der er udført (fx "gem kunde")
 - Hvordan den nye kunde ser ud (fx "Fornavn = Hans Hansen"...og resten af data)
- Dette vil vi gerne sende til det (de) andre systemer som der skal kommunikeres med.
 - Det vil sige at vi sender en "notifikation om hændelse og data" til den anden applikations API.
 - ...nu stoler vi på at den anden applikation kan modtage data, behandle det ud fra alle forretningsreglerne og gemme det i dens database
 - ...det var et system opdageret...så er det "bare" resten også!

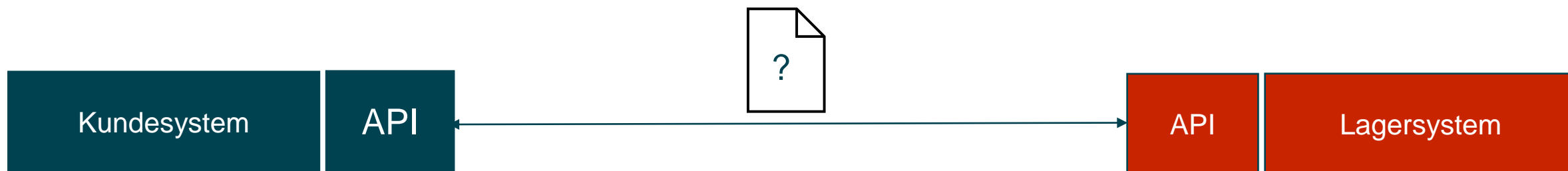
Samarbejdet mellem applikationer

- Opsummering
 - Kommunikation mellem applikationer sker gennem en række "håndtag" til at simulere en brugers interaktion med systemet – det kalder vi et API.
 - Ikke alle applikationer er født med et API – vi bør udvælge systemer ud fra om et "nyt" system har disse håndtag.
 - Kommunikation er komplekst, og antallet af samarbejdende applikationer forøger denne kompleksitet
 - Vi vil gerne kunne reagere på en "hændelse, således ved vi hvad der er sket og vi har data omkring denne hændelse" når vi skal sende det til andre applikationer.

Udveksling af data - formater

Udveksling af data - formater

- Når man kommunikerer mellem applikationer er det vigtigt at tale samme "sprog"
- Det sprog vi taler i kalder vi mange gange for "formatet"
- Vi skal forestille os at den "besked" som sendes mellem applikationer faktisk er meget lig noget vi kan læse:



Udveksling af data - formater

- Der er to formater som er de dominerende og som langt de fleste applikationer bruger til at udveksle data med (+99%)
 - XML
 - JSON
- Begge af disse formater kan nemt læses af et menneske
- De beskriver sig selv og er nemme for maskiner at læse også

XML

eXtensible Markup Language

XML

- Som navnet antyder er XML et markup language (det vil sige at det bruges til at beskrive samtidig med at man kommunikerer med det)
- Det kan nemt udvides og er meget fleksibelt
- Det er et sprog som er designet til at lave nye "sprog" med. I denne sammenhæng kan vi godt betragte de her nye sprog som "standarder" (noget vi har fastlagt)
- Det er nemt at læse for mennesker.
- Det består groft sagt af 2 ting
 - "tags/elementer" og "attributter"

XML

- Et "element" består af en (mindre end)
<
Efterfulgt af noget tekst, og så et (større end)
>
- Det kan fx se således ud: <navn>
- For at computeren kan forstå hvor langt navnet er, skal vi også fortælle hvornår <navn> slutter. Det vil sige hvor langt er navnet. Det gør vi med samme metode, blot tilføjer vi en / (skråstreg) til elementet

<navn>Hans Hansen</navn>

XML

- Formatet er også hierarkisk, det vil sige at vi kan bygge hierarkier i vores tekst:

```
<person>
  <navn>Hans Hansen</navn>
  <tlf>96727272</tlf>
  <email>hans@hansen.dk</email>
  <adresse>
    <gade>Hansensvej</gade>
    <nummer>7</nummer>
    <postnummer>9000</postnummer>
    <by>Aalborg</by>
  </adresse>
</person>
```

XML

- Attributter fortæller noget omkring det enkelte element

```
<person sprog="dansk">
  <navn>Hans Hansen</navn>
  <tlf type="privat">96727272</tlf>
  <email>hans@hansen.dk</email>
  <adresse>
    <gade>Hansensvej</gade>
    <nummer>7</nummer>
    <postnummer>9000</postnummer>
    <by>Aalborg</by>
  </adresse>
</person>
```

XML

- Vi har nu "skabt" vores eget nye sprog
- Lad os kalde det for "PersonML" (man putter typisk ML bagefter for at vise at vi har bygget det med XML)
- Det er et sprog man kan bruge til at repræsentere en person med en adresse med.
- Vi kan beslutte internt i vores organisation at dette er "standardmåden" at repræsentere en person med en adresse på, når vi kommunikerer internt.
 - Vi kan lave regler: fx "der skal ALTID være en email på en person"
 - Fordelen er at vi nu kan sige det til andre, og sige at vores format ser "således" her ud
 - Hvis andre applikationer kan give os dette format, så kan vi forstå det.
- Det er nu **en standard**

XML

- "Vi har to applikationer med hvert sit API, de bruger XML til at kommunikere med"



JSON

JavaScript Object Notation

JSON

- JSON er den mest brugte måde at kommunikere på
- Der er igen tale om det format som de to applikationer bruger til at sende til hinanden
- JSON er også læsbart for mennesker – men det er ikke så "tydeligt" som XML

JSON

- JSON formatet består af "elementer" - i JSON har de bare ikke <...> tegn, men bare teksten med anførelstegn " omkring.
- JSON består af "name/value pairs" - navn-og-værdi-par. Det vil sige et navn, fx `navn` og så en værdi, fx `Hans Hansen`.
- JSON starter med en tuborg parentes { og slutter med en tuborgparantes } - det svarer til den struktur vi kender fra XML

```
{ "navn" : "Hans Hansen" }
```

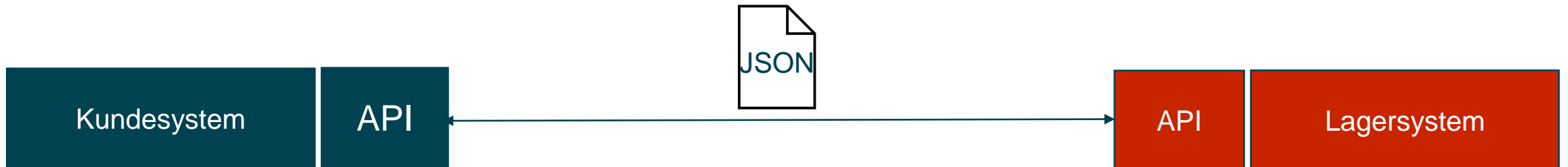
JSON

- Lige som XML er JSON formatet også hierarkisk:

```
{  
  "navn" : "Hans Hansen",  
  "tlf" : "96727272",  
  "email" : "hans@hansen.dk",  
  "adresse" : {  
    "gade" : "Hansensvej",  
    "nummer" : "7",  
    "postnummer" : "9000",  
    "by" : "Aalborg"  
  }  
}
```

JSON

- "Vi har to applikationer med hvert sit API, de bruger JSON til at kommunikere med"





Er der spørgsmål omkring XML og JSON?

Opgave – XML/JSON (30 min)

- I grupper skal i lave et nyt sprog
- Sproget skal kunne beskrive en "opskrift" (på mad)
 - (evt. tag udgangspunkt i en opskrift i finder online)
 - Navn, Beskrivelse, Ingredienser, trin
 - I bestemmer selv om det er XML eller JSON
- Hvad hedder jeres sprog? (*ML)

Kommunikation når vi ikke taler samme sprog

Vi taler ikke samme sprog

- Forskellige sprog – og hvad så?
- I de sidste eksempler har vi kun haft to applikationer som, heldigvis, talte samme sprog (XML eller JSON)
- Hvis vi øger kompleksiteten en smule og antager at vi har 4 applikationer som skal tale sammen, og alle formater er forskellige:

Vi taler ikke samme sprog

- Kundesystemet (XML)

```
<person sprog="dansk">
  <navn>Hans Hansen</navn>
  <tlf type="privat">96727272</tlf>
  <email>hans@hansen.dk</email>
  <adresse>
    <gade>Hansensvej</gade>
    <nummer>7</nummer>
    <postnummer>9000</postnummer>
    <by>Aalborg</by>
  </adresse>
</person>
```


Vi taler ikke samme sprog

- Supportsystemet (XML)

```
<person sprog="dansk">
  <fornavn>Hans</fornavn>
  <efternavn>Hansen</efternavn>
  <tlf type="privat">96727272</tlf>
  <email>hans@hansen.dk</email>
  <adresse>
    <gade>Hansensvej</gade>
    <område></område>
    <nummer>7</nummer>
    <postnummer>9000</postnummer>
    <by>Aalborg</by>
  </adresse>
</person>
```

Vi taler ikke samme sprog

- Lønssystem (JSON)

```
{  
  "navn" : "Hans Hansen",  
  "tlf" : "96727272",  
  "email" : "hans@hansen.dk",  
  "adresse" : {  
    "gade" : "Hansensvej",  
    "nummer" : "7",  
    "postnummer" : "9000",  
    "by" : "Aalborg"  
  }  
}
```

Vi taler ikke samme sprog

- Faktureringsystem (JSON)

```
{  
  "fuldenavn" : "Hans Hansen",  
  "mobil" : "96727272",  
  "email" : "hans@hansen.dk",  
  "gade" : "Hansensvej",  
  "nummer" : "7",  
  "postnummer" : "9000",  
  "by" : "Aalborg"  
}
```

Vi taler ikke samme sprog

- Udfordringen med at kommunikere sammen er nu blevet meget mere kompleks.
- Ikke kun skal vi kommunikere sammen med en masse forskellige applikationer
 - Alle bruger forskellige formater og har struktureret deres indhold forskelligt
- Læg dertil at formatet vi sender til hinanden kan ændre sig over tid, men kun for nogle.
- (En standard sikrer i meget højere grad at disse ændringer sker på en mere kontrolleret måde, fx ved at få en advarsel 6 md. i forvejen omkring hvad der sker af ændringer)

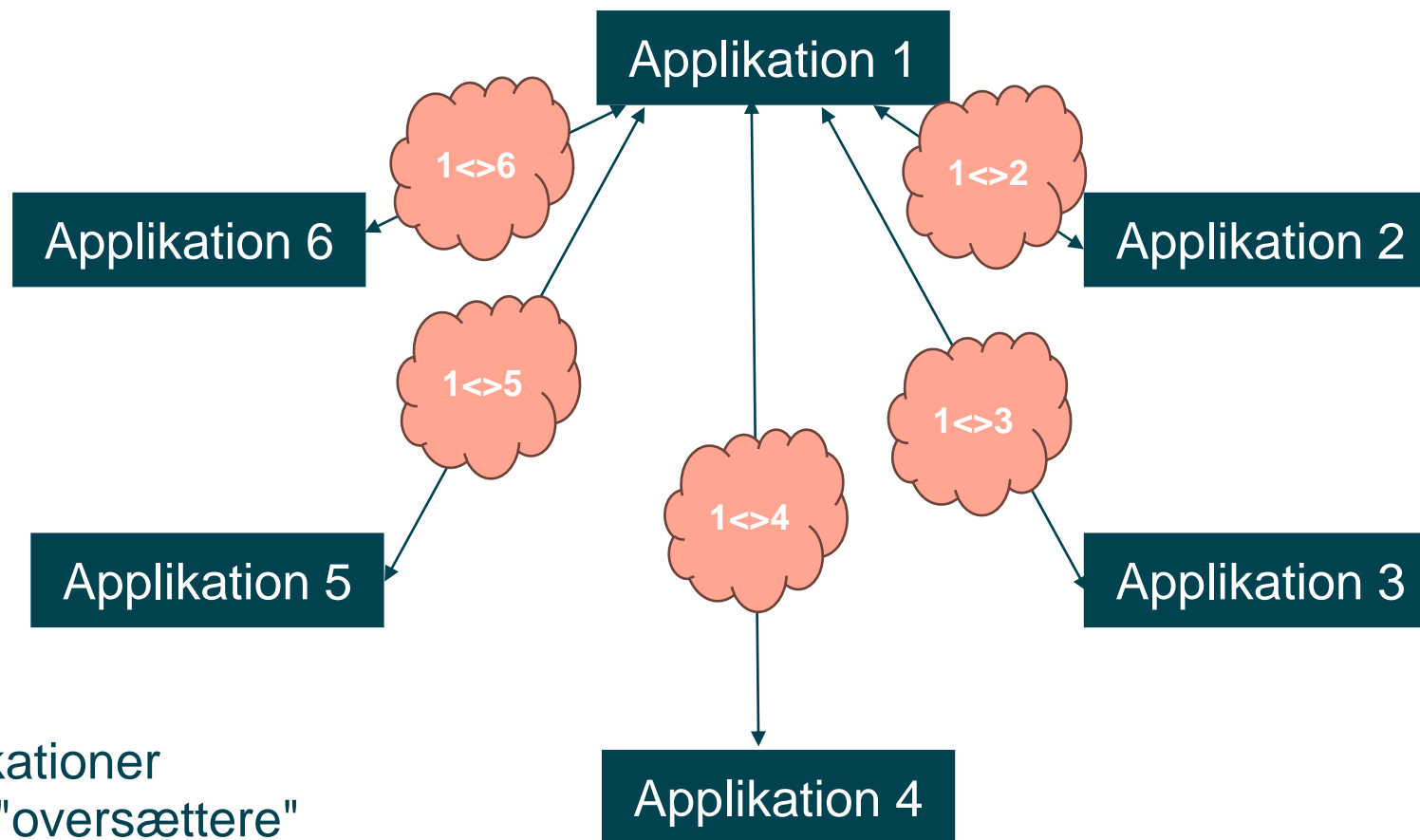
Vi taler ikke samme sprog

- Udfordringen med at kommunikere sammen er nu blevet meget mere kompleks.
- Ikke kun skal vi kommunikere sammen med en masse forskellige applikationer
 - Alle bruger forskellige formater og har struktureret deres indhold forskelligt
- Læg dertil at formatet vi sender til hinanden kan ændre sig over tid, men kun for nogle.
- (En standard sikrer i meget højere grad at disse ændringer sker på en mere kontrolleret måde, fx ved at få en advarsel 6 md. i forvejen omkring hvad der sker af ændringer)

Vi taler ikke samme sprog

- Der er generelt set to måder at løse "problemet" på
 - 1: Hvert system skal kende alle de andre systemer den skal kommunikere med (n-1) og så oversætte til og fra eget format og så destinationsprogrammet
 - Her sidder der typisk egne softwareudviklere og udvikler dette
 - 2: Man bygger en central applikation som indeholder sig eget (helt nye) format (i JSON eller XML) - det betyder at hver applikation så "kun" skal oversætte til/fra sit eget format til det "fælles format"
 - Her sidder der typisk egne softwareudviklere og udvikler dette.

Vi taler ikke samme sprog



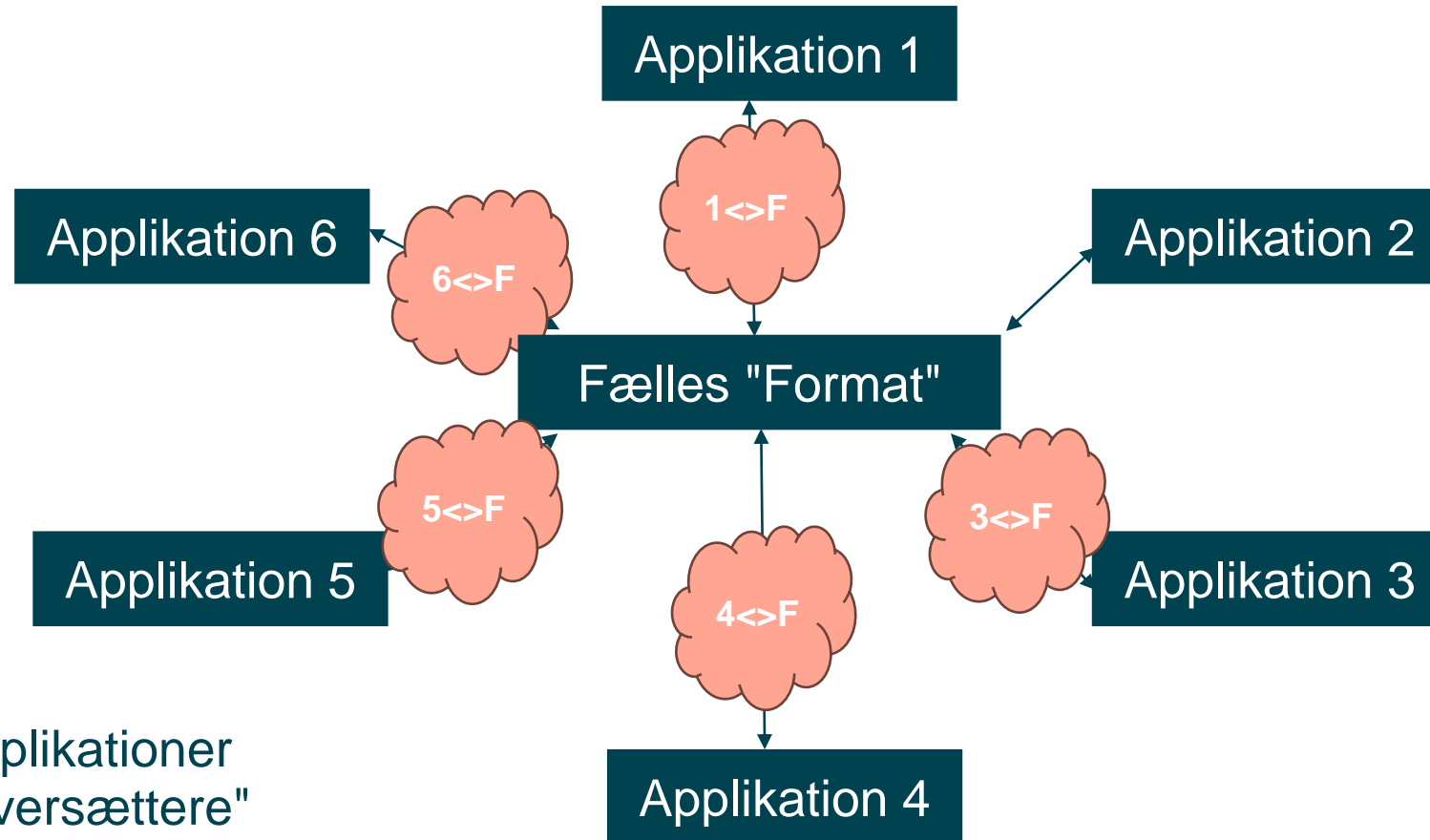
Kompleksitet:

n = antallet af applikationer

$n * n - 1$ = antallet af "oversættere"

(Her mangler alle de andres oversættere)

Vi taler ikke samme sprog



Kompleksitet:

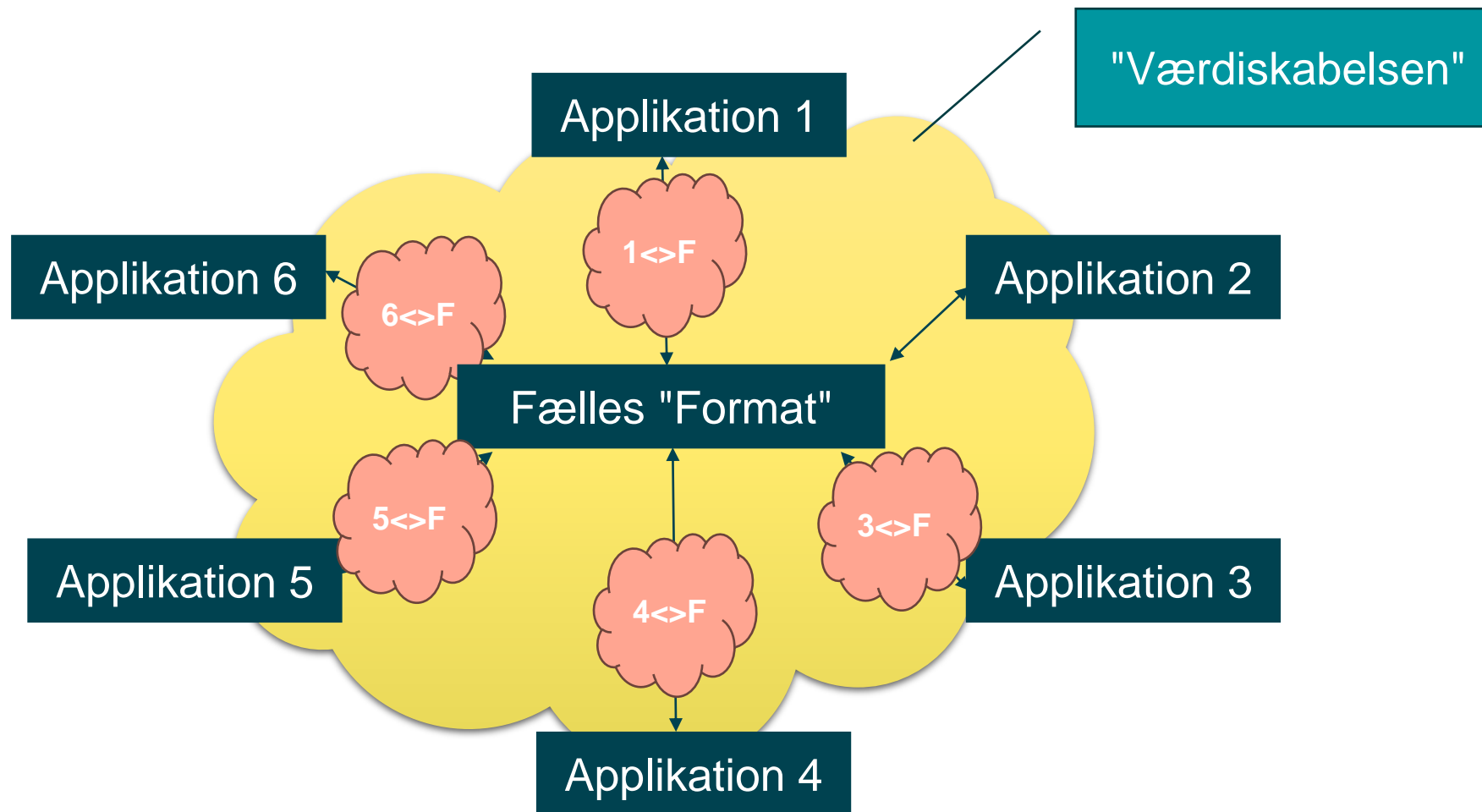
n = antallet af applikationer

n = antallet af "oversættere"

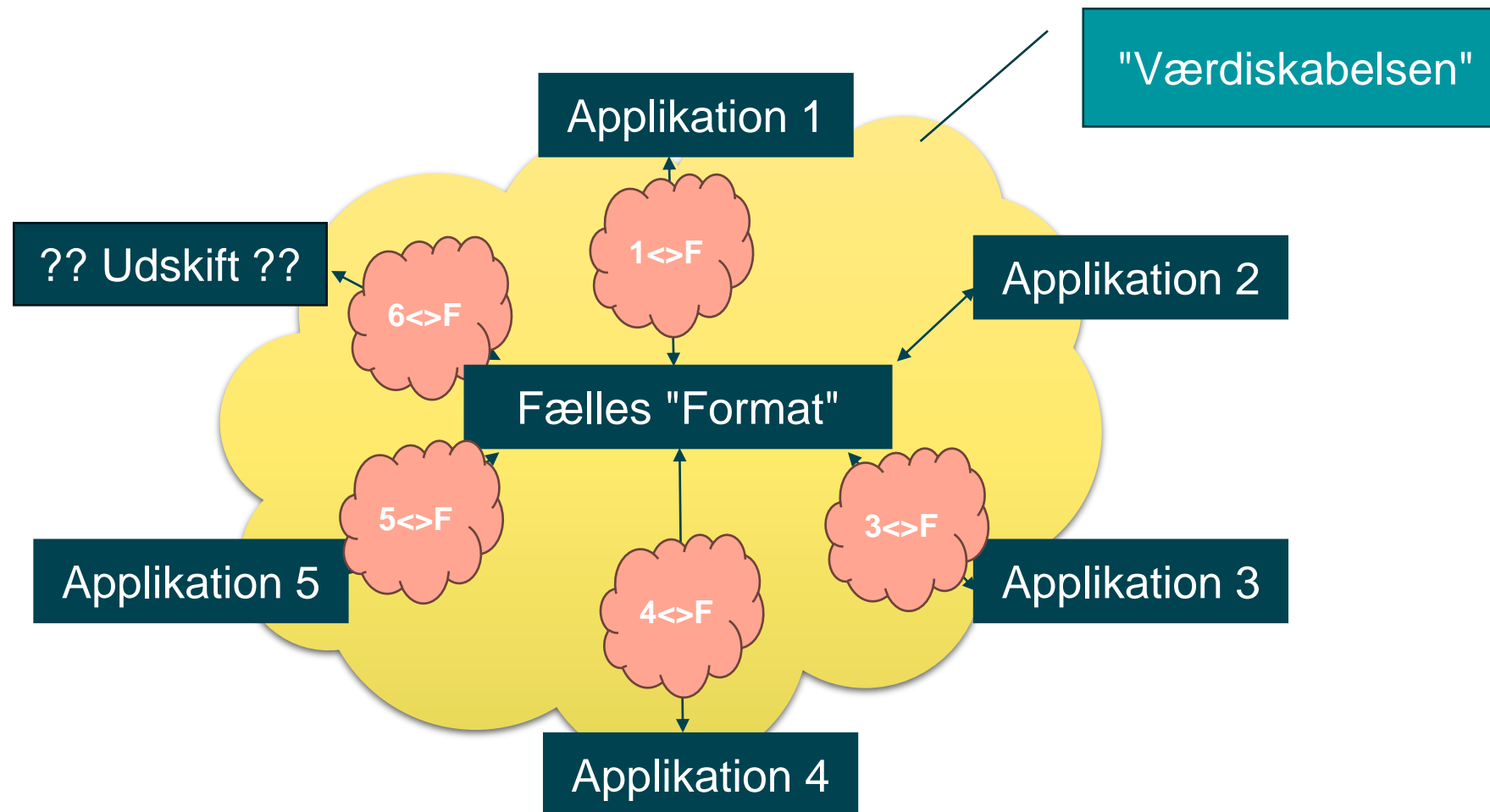
Vi taler ikke samme sprog

- Komplexiteten i dette er et enormt arbejde at vedligeholde.
- Der findes nogle platforme som kan dette på en måde, som ikke kræver at man skal skrive kode.
- At vedligeholde dette "spind" af oversættere er "fast arbejde" for en IT afdeling – det stiller et højt krav til at man dokumenterer dette og er meget systematisk.
- Man har behov for at der hele tiden er et vågent øje på applikationerne og deres "versioner"
 - Skal man opgradere til en nyere version? Hvad med formatet?
 - Hvor meget påvirkes

...sidespor



...sidespor



Vi taler ikke samme sprog

- Komplexiteten i kommunikationen mellem applikationer stiger markant hver gang en ny applikation skal med i "spindet"
- Vedligeholdet af oversætterne stiller store krav til dokumentation, systematik og viden omkring hvordan de forskellige systemer virker og til domænet.
- Værdien og synergien i systemet ligger i de regler og den logik som er indarbejdet i integrationerne mellem applikationerne
- Man udvælger (nye) systemer ud fra deres evne til at indgå i de eksisterende integrationer (at de har et API)



**Er der spørgsmål omkring kommunikation,
når vi ikke taler samme sprog ?**



Må vi have lov til at tale sammen? - noget omkring sikkerhed

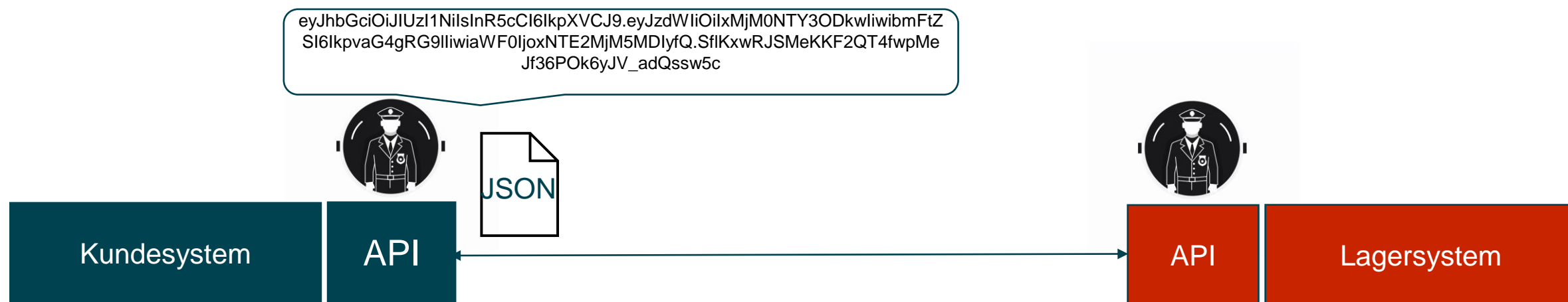
Må vi tale sammen?

- Alle applikationer har et ekstra lag af sikkerhed.
- Når vi logger ind som bruger skal vi typisk give brugernavn og adgangskode.
- Applikationer bruger ikke brugernavn og adgangskode – for det meste bruger de en "token"
- Formålet er det samme – at få adgang til data, enten at læse, eller at skrive i data.



Må vi tale sammen?

- Sikkerheden kan sammenlignes med en "vagt" i døren som lige skal tjekke dit armbånd inden du kan komme ind.
- Tokens "udstedes" af kilde-applikationen første gang man skal bruge den. Den applikation kan så også "tilbagekalde/slette" den - så er den ugyldig og man kan ikke "komme ind"



Arkitektur

Arkitektur

- Vi skal udvide vores "tegninger" og snakke lidt dybere omkring hvordan applikationer kommunikerer sammen.
- Vi vil tale omkring begreberne som bruges og hvordan der er flere ting som har samme betydning (afhængigt af afstanden man betragter dem fra)
 - WebService, REST, SOAP
 - Messaging, Queue, MQ...
- Det hele handler omkring hvordan vi kommunikerer (teknisk) og det format vi udveksler.
- Generaliseringen er at vi "bare kommunikerer" - hvis vi kigger dybt ind i teknikken taler vi om teknikker, hvis vi træder langt tilbage – sender vi bare data frem og tilbage for at skabe værdi.

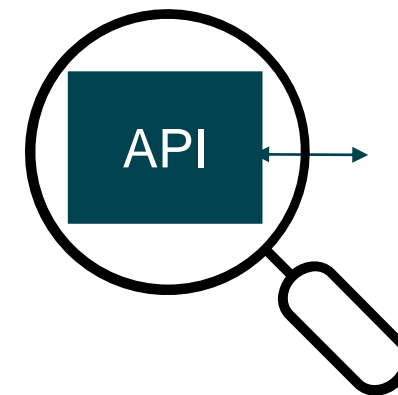
Arkitektur



- Webservice: Det er en service en applikation "udstiller" til web (internettet) således den service er let tilgængelig.
 - Det er ikke sikkert at vi i hverdagen skelner mellem API og Webservice
 - Det er mere "historisk" - nogle systemer har helt specifikt noget ekstra omkring API, mens i andre systemer er der webservice og et API sidestillet
- Vi har i sproget flere begreber som dækker det samme
 - "For at få udført opgave XXX vil jeg sende YYY data til applikation ZZZs Webservice|API"

Arkitektur

- Teknologi: Rent teknisk kan et API/Webservice være lavet på to måder:
 - **SOAP** – Et XML format som sender en information omkring hvad der skal gøre (handlingen) samt det data der skal sendes med til handlingen
 - **REST** – typisk et JSON format som er meget simplere (mere tilgængeligt) og som fortæller noget om en ressource



Arkitektur

- SOAP er en standard beskyttet af W3C (sidst ændret i 2007)
- SOAP: Vi kan sige at vi bruger SOAP formatet til at udveksle data med.
- Typisk er SOAP en ældre måde at kommunikere på. Meget populært for 10+ år siden.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <CreateRecipeRequest xmlns="http://example.com/recipes">
      <Recipe>
        <Name>Apple Pie</Name>
        <Ingredients>
          <Ingredient>Apples</Ingredient>
          <Ingredient>Sugar</Ingredient>
          <Ingredient>Flour</Ingredient>
          <!-- Additional ingredients here -->
        </Ingredients>
        <Instructions>
          <instruction>Peel and slice the apples.</instruction>
          <instruction>Mix the sliced apples, sugar, and flour together.</instruction>
          <instruction>Place the mixture into a pie crust.</instruction>
          <instruction>Bake in the oven at 350°F for 45 minutes.</instruction>
          <instruction>Serve and enjoy!</instruction>
        </Instructions>
      </Recipe>
    </CreateRecipeRequest>
  </soap:Body>
</soap:Envelope>
```

Arkitektur

- REST er en standard beskyttet af W3C – udvikles stadig, mere kompleks. Bygger på en simpel "arkitektur"
- Kombination af en URL (som i en browser), en "metode" (fx "GET, PUT, POST, DELETE") og selve data
- Simpel – kan læses og forstås af mennesker.

URL: <http://minapplikation.dk/opskrifter>

Metode: POST

Data:

```
{
  "Recipe": {
    "Name": "Apple Pie",
    "Ingredients": [
      "Apples",
      "Sugar",
      "Flour"
      /* Additional ingredients here */
    ],
    "Instructions": [
      "Peel and slice the apples.",
      "Mix the sliced apples, sugar, and flour together.",
      "Place the mixture into a pie crust.",
      "Bake in the oven at 350°F for 45 minutes.",
      "Serve and enjoy!"
    ]
  }
}
```

Arkitektur – lidt mere REST (se også slide 31)

URL	Metode	Kommentar
https://kundeapplikation.dk/kunde	POST	Opretter en ny kunde
https://kundeapplikation.dk/kunde/12	GET	Henter kunde med id 12
https://kundeapplikation.dk/kunde/12	PUT	Opdaterer kunde med id 12
https://kundeapplikation.dk/kunde/12	DELETE	Sletter kunde med id 12

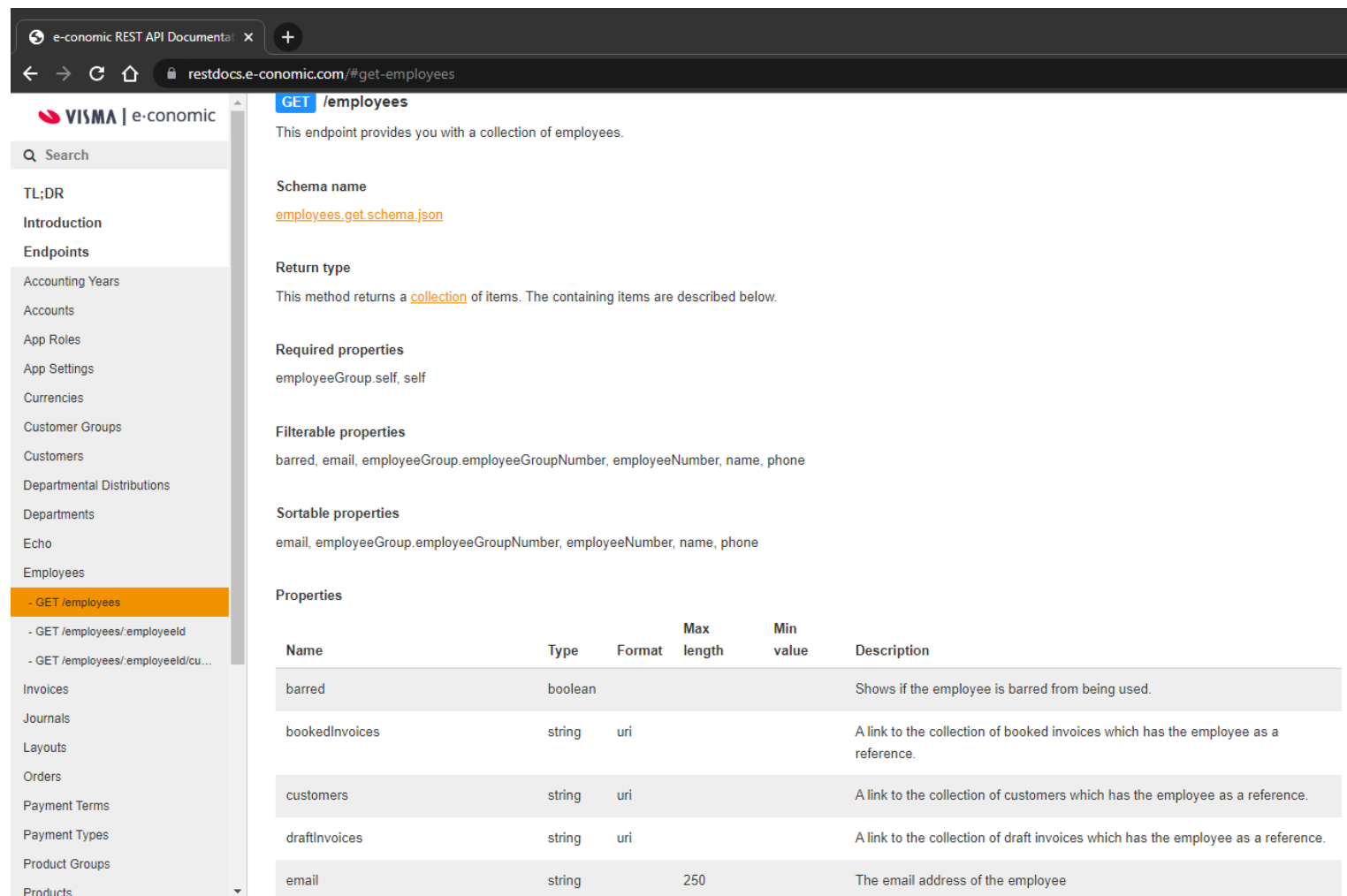
Arkitektur – lidt mere REST (se også slide 31)

URL	Metode	Kommentar
https://opskriftapplikation.dk/opskrift	POST	?
https://opskriftapplikation.dk/opskrift/12/ingredienser	GET	?
https://opskriftapplikation.dk/opskrift/12/ingredienser/5	PUT	?
https://opskriftapplikation.dk/opskrift/12/trin/8	DELETE	?

Arkitektur – Opgave (20 min)

- Åbn en browser og gå til: <https://swapi.dev/>
 - SWAPI tillader KUN get
- Svar på disse spørgsmål ved at kigge på formatet, indtaste en REST url og trykke på "request"
 - Hvor langt er det rumskib (starship) Luke Skywalker flyver i?
 - Hvor mange "films" er der lavet ? (hint: hent alle sammen)
 - Hvilket år der Anakin Skywalker født (hint: id=11)
 - Hvilke 2 køretøjer "kører" Anakin Skywalker i?
 - Navnet på dem

Arkitektur – andre systemer



The screenshot shows a web browser displaying the REST API documentation for the e-economic system. The left sidebar contains a navigation menu with various endpoints, and the main content area details the GET /employees endpoint.

GET /employees

This endpoint provides you with a collection of employees.

Schema name
[employees.get_schema.json](#)

Return type
This method returns a [collection](#) of items. The containing items are described below.

Required properties
employeeGroup.self, self

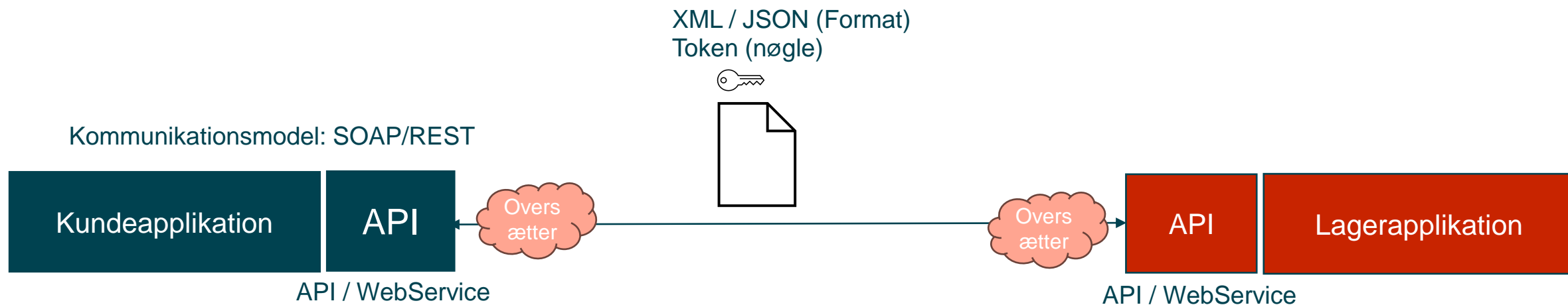
Filterable properties
barred, email, employeeGroup.employeeGroupNumber, employeeNumber, name, phone

Sortable properties
email, employeeGroup.employeeGroupNumber, employeeNumber, name, phone

Properties

Name	Type	Format	Max length	Min value	Description
barred	boolean				Shows if the employee is barred from being used.
bookedInvoices	string	uri			A link to the collection of booked invoices which has the employee as a reference.
customers	string	uri			A link to the collection of customers which has the employee as a reference.
draftInvoices	string	uri			A link to the collection of draft invoices which has the employee as a reference.
email	string		250		The email address of the employee

Arkitektur





**Er der spørgsmål til arkitekturen - og når to
applikationer taler sammen ?**

Messaging, Queues, MQ

Beskeddrevet kommunikation

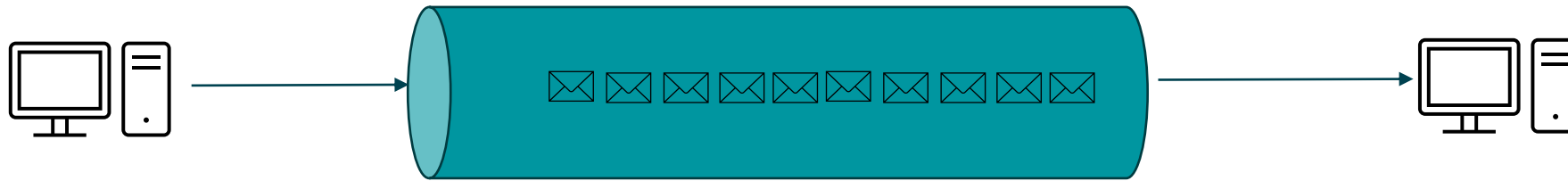
Messaging

- Kommunikation mellem systemer kan være komplekst – meget komplekst
- Et af de problemer man kan løbe ind i, når man forsøger at kommunikere mellem så mange systemer – er at de måske er "langsomme", "optagede", "nede"
- Hvis alle kalder hinanden direkte, og skal vente på at få udført en opgave, så er det noget som kan svække hastigheden på hele det samlede "system" af applikationer.

Messaging

- Typisk bruger man messaging til at løse dette problem
- Messaging er en af de ældste måder at kommunikere mellem applikationer (og internt i en applikation på)
- Tanken er at "klienten" kan placere en besked i en "kø" (deraf navnet – queues) - og så vende tilbage til sit arbejde.
 - Klienten (afsenderen) har nu glemt alt omkring den "uddelegerede" opgave og stoler nu på at "en eller anden" nok skal tage opgaven og udføre det.

Messaging



- Hvad er der indeni en besked?



Er der spørgsmål til messaging ?

Ordbog

Ordbog

Ord	Forklaring
Domæne	Et bestemt område af virkeligheden. Man observerer virkeligheden og beskriver det som et domæneområde. "Revisionsdomænet", "tømrerdomænet", "Sundhedsdomænet". Er et domæne komplekst kan man med fordel dele det op i mindre dele.
Domæneobjekt	Et bestemt element fra domænet. Det kan fx. være en kunde, borger, virksomhed – dette vil typisk være noget håndgribeligt i "virkeligheden"
Semantik	Betydningen af noget, det kan fx. Være betydningen af en kunde. To applikationer kan have forskellige betydning af det samme begreb. En kunde i supportsammenhæng er anderledes end en kunde i faktureringssammenhæng. Vi vil dog gerne kunne tale samme på tværs af systemer omkring samme begreb: en kunde. Dette kaldes semantisk interoperabilitet.
Syntaks	Grammatik, det vil sige om vi overholder vores format. I forbindelse med skrevet sprog kan man lave stavfejl osv. I forhold til fx. XML eller JSON er det typisk fordi man har lavet en fejl i fx. At glemme et tegn: < > " { - eller noget andet som gør formatet ugyldigt.

Ordbog

Ord	Forklaring
Protokol	En aftale omkring I hvilken rækkefølge vi gør tingene, begge parter kender aftalen. Man taler også om at følge protokollen når man er til galla-fester. Mellem applikationer kan protokollen være at man I en bestemt rækkefølge skal sende besked, få en token, sende en anden besked inkl. Den token osv.
Standard	En fastlagt aftale omkring fx. Et format. Det bruges som et sikkerhedsnet under vores kommunikation således vi har noget vi "alle" er enige om. Vi har også typisk en vis beskyttelse af formatet (governance) således at vi fx. Får en advarsel inden vi ændrer det, således alle kan nå at opdatere til det nye format. SOAP og RET er begge standarder, beskyttet af W3C.
Applikation / Program	En softwareudvikler eller et firma har udviklet et program til at løse en opgave. Der er typisk tale om en manuel opgave som nu er blevet digitaliseret.
Data	Meget bredt kan data være alt, et regneark kan indeholde data, data kan indsamles og gemmes. I forhold til applikationer er det and det data applikationerne indeholder. Data gemmes typisk I en database.

Ordbog

Ord	Forklaring
Database	En meget speciel applikation som er udviklet til det formål at være rigtig god til at gemme data på en struktureret måde. Næsten alle applikationer bruger en database på en eller anden måde. Databaser har permanent lager, det vil sige at data består selvom en applikation lukkes eller computeren genstartes.
Distribuerede Systemer	Når flere applikationer samarbejder på tværs af netværk/internet.
Server	En leverandør af en service. Indenfor applikationer snakker man ofte om at man har en server et sted. Der er mange betydninger af en server. Det kan være en fysisk server (computer i et skab) det kan også være en logisk server (fx. En database server, som indeholder alle vores data. Indenfor applikationer kan en applikation også være en "logisk" server. Hvis de applikationer taler sammen kan man den "initierende" part for klienten (klienten kommer og vil have en service) og den som bliver "kommunikeret til" er så en server. Eks: Klienten beder serveren om alle kunder som har udestående fakturaer. Dette sker fx. Via et API. Samme server og samme klient kan også have omvendte roller i en anden sammenhæng.
Klient	Se Server

Ordbog

Ord	Forklaring
Serviceorienteret Arkitektur (SOA)	Når alle vores applikationer samarbejder for at fremstå som eet enkelt system og værdien skabes i sammenhængen og logikken mellem systemerne. Vi betragter vores systemer, dels som nogle værktøjer til brugere, men også som serviceleverandører til andre systemer.
Arkitektur	Overordnet overblik/billede af hvordan alle vores applikationer hænger sammen.
API	Et andet ord for en WebService. En måde at tilbyde andre applikationer (se også "server") at løse en opgave på. API er for andre applikationer, hvad en brugergrænseflade er for mennesker.
Robotic Process Automation (RPA)	En måde at automatisere og integrere på. Oplær et program til at lave repetetive opgaver på, således man ikke behøver at gøre det manuelt.

Ordbog

Ord	Forklaring
Format	Den interne struktur på "noget" tekst. Det kan også dække over om man bruger XML eller JSON til at kommunikere med.
XML	XML er et format med <tag></tag> tags. Det er nemt at læse både for mennesker og computere og bruges til at udveksle data med. Man kan beslutte sig for at XML skal have en bestemt struktur, så kan vi kalde det for en standard. SOAP er XML som er lavet til en standard.
JSON	JSON er et format, ligesom XML. JSON bruger nogle andre tegn ":{ }" end XML. Princippet er næsten det samme. Typisk er det nyere API/Webservices som bruger JSON.
Integration	Når man binder to eller flere applikationer sammen og lader dem udveksle data.
Token	Sikkerhedsnøgle som en applikation skal vise til en anden applikation for at måtte læse eller skrive til dens API. Typisk vi en klient kalde et API på en server og vise den en token for at verificere at den må have lov til at udføre en handling. Lidt det samme som en sikkerhedsvagt/dørvagt skal tjekke en billet.

Ordbog

Ord	Forklaring
SOAP	XML baseret format (som også er en standard). SOAP beskriver hvordan en handling kan udføres på en server.
WebService	Se API. Det er to ord som dækker samme princip.
REST	Måde at kommunikere mellem applikationer på. Nyere tilgang som vil tage over for SOAP. REST består af en URL adresse, en metode (GET, PUT, POST, DELETE) og noget data. Data er typisk JSON.
Messaging	Alternativt til SOAP og JSON. En applikation kan sende en besked (XML eller JSON) til en kø og alle "abonnenter" kan så få en besked, når de er klar/har ressourcer til at tage flere opgaver. En "simpel" kommunikationsmekanisme som sikrer at alle kan "arbejde" og producere beskeder og at dem som skal udføre opgaverne kan gøre det i deres eget tempo.