IN1000-seminar med oblig 8

Fagutvalget ved informatikk arrangerer IN1000-seminar

- 3. november 16:**00** til 18:00 – servering av pizza
- Tips til oblig 8, orakler tilstede for spørsmål
- Mer info og påmelding i lenke fra semestersiden

Objektorientert programmering i Python

IN1000

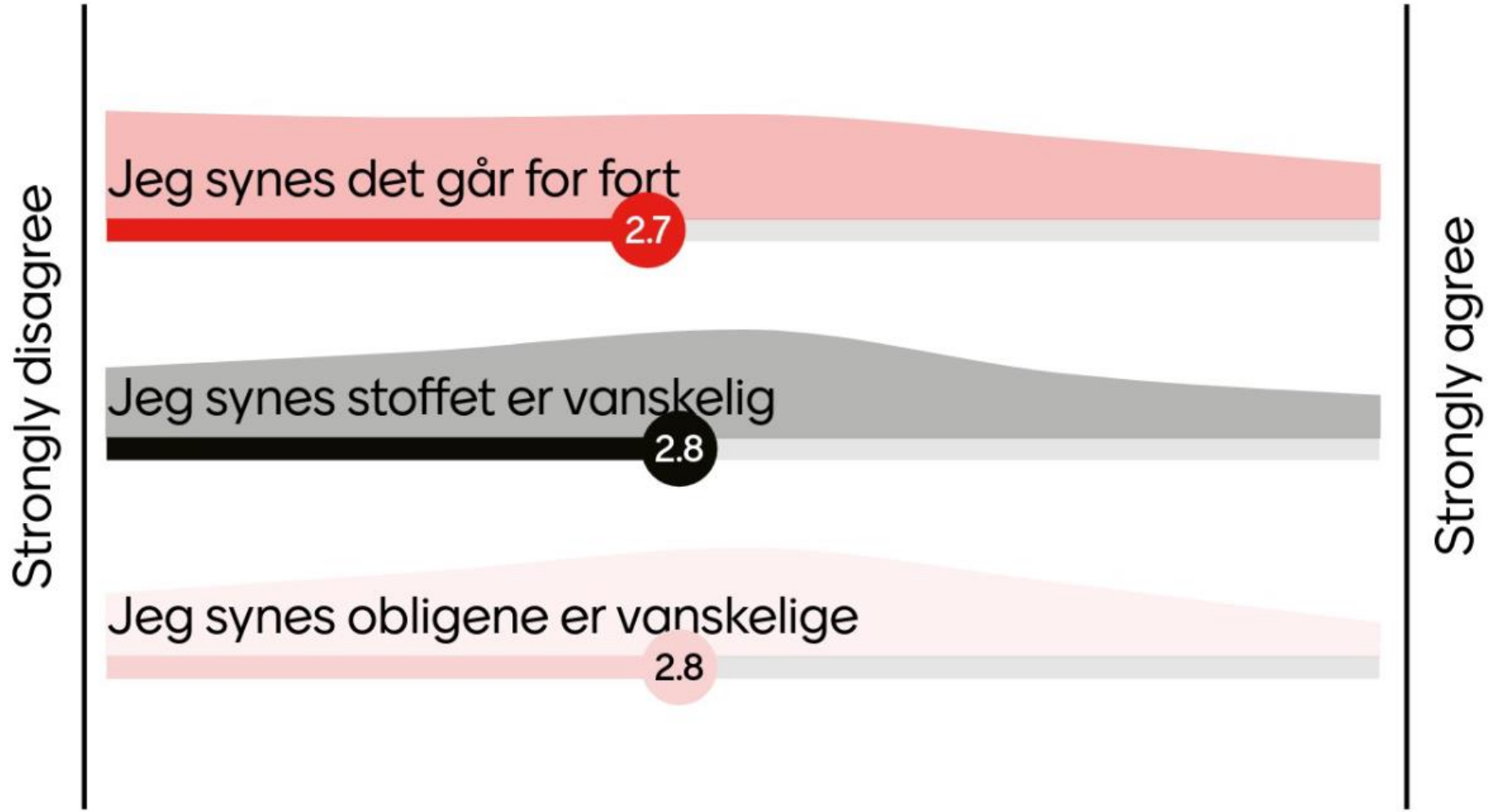
Høst 2021 – uke 10

Siri Moe Jensen og Geir Kjetil Sandve

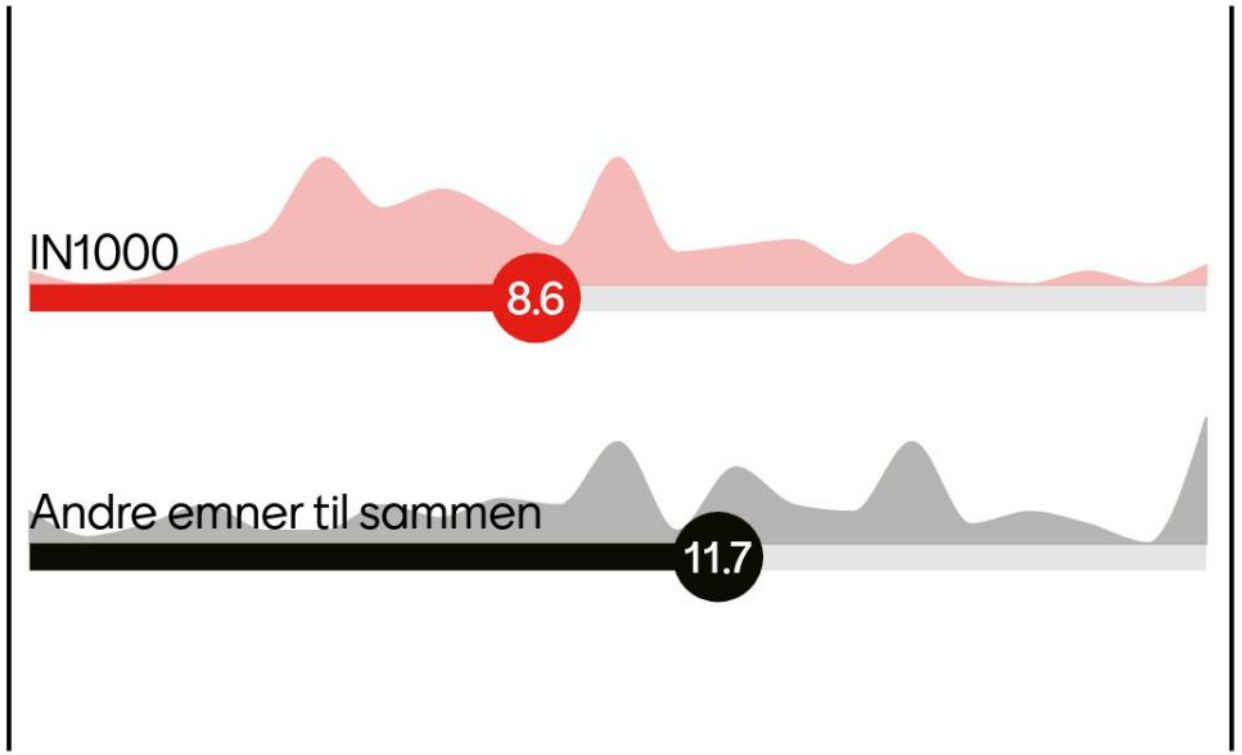
Innhold uke 10

- Underveisevaluering og studietips
- Noen sentrale datastrukturer for programmering
  - lenkede lister
  - trær
  - grafer
- Eksempler med livekoding: Offentlig transport som
  - lenket liste
  - graf
- Aggregerte klasser

Lett eller vanskelig?



Hvordan bruker du tiden din på studiet (antall timer per uke)?



Mer livekoding, mer eksempler, mer oppgaver

- Dette er det vi skal gjøre fremover!
- Ingen nye ting i Python, litt begreper men mest ulike anvendelser, større eksempler inkl livekoding – og større oppgaver
- Fokus på eksamen de siste ukene, inkl prøveeksamen

Vanskelig?

- Mer hjelp/ lab: Det holdes nå over 20 seminartimer og nesten like mange lab-timer per uke – gå gjerne på flere
- Gå på repetisjonstimen fredag kl 12:15, og skriv gjerne ønsket pensum på Mattermost (egen kanal for den timen)
- Fredagspython er åpen for alle, litt varierende kapasitet for hjelp
- Programmer oppgaver i det du ikke får til (men start med enkle – velg tag øving og passende tema i Trix).
- Finn ut hvorfor ting blir feil, krever nøtaktighet og analyse steg for steg – hva ligger i variabelen, hvilke type har det (og hva kan jeg gjøre med verdier av den typen), hva er resultatet av et uttrykk
- Bruk teori (forelesninger, lærebok, nettet om du finner ressurser på riktig nivå) for å forstå hva som skjer – og eksperimenter med enkle eksempler (bruk utskrifter eller kjør i Pythontutor)

Lett?

- Lag egne programmer, eller utvid og forbedre andre oppgaver (obliger, eksempler fra forelesninger, ...).
- Begynn evt å jobbe med annet stoff, som f eks IN1150 – dette ligger ute som nettkurs og holder de fleste i ånde en stund.

Diverse

- Semestersiden: Den inneholder mye informasjon, og er preget av utvikling over noen år + påtvunget ny mal i vår. MEN bruk 10-15 minutter så skjønner du hva som ligger hvor og kan få svar på mange spørsmål

- Planer fremover og ressurser: Plan ut semesteret ligger på timeplanen, og vi prøver å legge ut mest mulig på forhånd – så sjekk innimellom

- Samarbeid: Beste tips er å delta i gruppetimer og andre tilbud (inkludert seminaret 3.11). Inviter andre til å jobbe sammen. Skjønner at det ikke bare er lett – men bruk de arenaene som finnes, begynn i det små.

- NB: Du lærer minst like mye av å samarbeide med svakere studenter som må forklares ting. Vær litt large – og respekter avtaler ☺

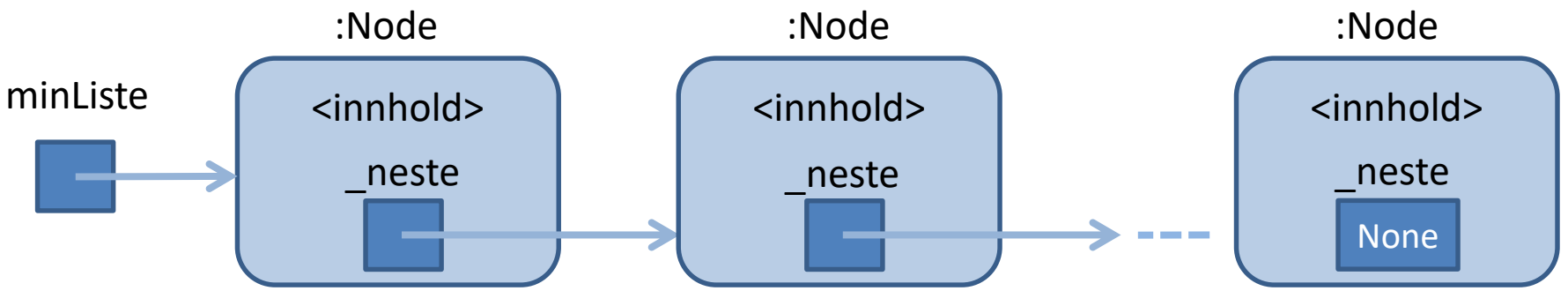
Å lage egne beholdere (containers)

- Trenger en datastruktur for å lagre ukjent antall (like) objekter. I Python har vi **lister**, **ordbøker** og **mengder** – hva om disse ikke fantes?
- Hvordan opprette en ny variabel hver gang vi trenger den – og få tak i den senere?
- Vi kan gå i løkke og opprette objekter etter behov – og vi har sett at et objekt deretter kan brukes fra ulike steder i programmet vårt, bare vi har en variabel med en referanse til objektet.
- Men hvordan sørge for nok referansevariable til et ukjent antall objekter, slik at vi kan finne dem igjen og bruke dem senere?

$\Rightarrow$ en klassisk datastruktur for et ukjent antall elementer er en *lenket liste*

Lenkede lister

- Poenget med denne strukturen er at for hvert nye objekt vi lager – så lager vi samtidig en referansevariabel som kan referere til et nytt objekt
- dvs hvert objekt må kunne referere til et annet objekt
- dermed får vi en lenket liste av objekter – og trenger bare ha én referanse til det første objektet fra der vi skal bruke listen

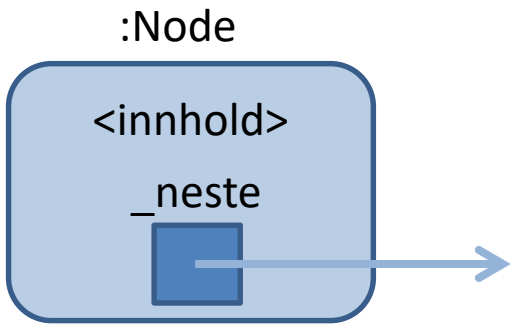


Lenket liste

Lager en generell klasse for objektene i lenkelisten

- Klassen Node

- Grensesnitt for klassen Node:
  - lese, endre, skrive ut, ... innhold
  - legg til ny etterfølger
  - hent etterfølger
  - (fjern etterfølger)



Klassen Node - implementasjon

- Datastruktur Node-klassen
  - en referansevariabel så vi får tak i neste objekt
  - innholdet objektene skal representere

```
class Node :
    def __init__(self, nytt) :
        self._innhold = nytt
        self._neste = None

    def nyEtterfolger (self, ny) :
        self._neste = ny

    def hentNeste (self) :
        return self._neste

# + metoder for manipulering av selve innholdet
```

Oppgave (litt nøttesmak)

```
class Node :
    def __init__(self, nytt) :
        self._innhold = nytt
        self._neste = None

    def nyEtterfolger (self, ny) :
        self._neste = ny

    def hentNeste (self) :
        return self._neste

    def hentInnhold(self):
        return self._innhold
```

Skriv kodelinjer som tar en ferdig lenket liste (minListe) og skriver ut innholdet i den bakerste noden (antar self._innhold er en string her)

Tips: Du trenger en ekstra referansevariabel som flytter seg gjennom objektene i listen

Live eksempel I - trikk

Et eksempel der lenkene representerer viktig informasjon om objektene våre (ikke bare hjelper oss å få tak i flere objekter):

Her er nodene *stasjoner på en trikkerute*, dvs. den lenkede listen består av objekter av klassen Stasjon som inneholder referanser til nabostasjoner.

<se egen presentasjon/ kildekode>

Innkapsling og generalisering

- Vi kan organisere et «uendelig» antall noder (f eks stasjoner) ved å opprette og lenke inn nye objekter av klassene Node (eller Stasjon) ved behov.

- Noen designmessige ulemper:
  - den som skal bruke klassen må kjenne til sammenlenkingen av objektene
  - for hver klasse vi ønsker å lage lister av, må vi skrive all koden som har med liste-håndtering å gjøre, om igjen – den ligger jo innbakt i klassen sammen med informasjon om innholdet

- Tiltak:
  - Skjule klassen Node (Stasjon) inne i en klasse Lenkeliste (LagRute) som håndterer (og skjuler) alle lenke-operasjoner, og kun tilbyr brukervennlige tjenester for å opprette og administrere noder (Stasjoner) i sitt grensesnitt
  - Ta emnet IN1010 til våren ☺

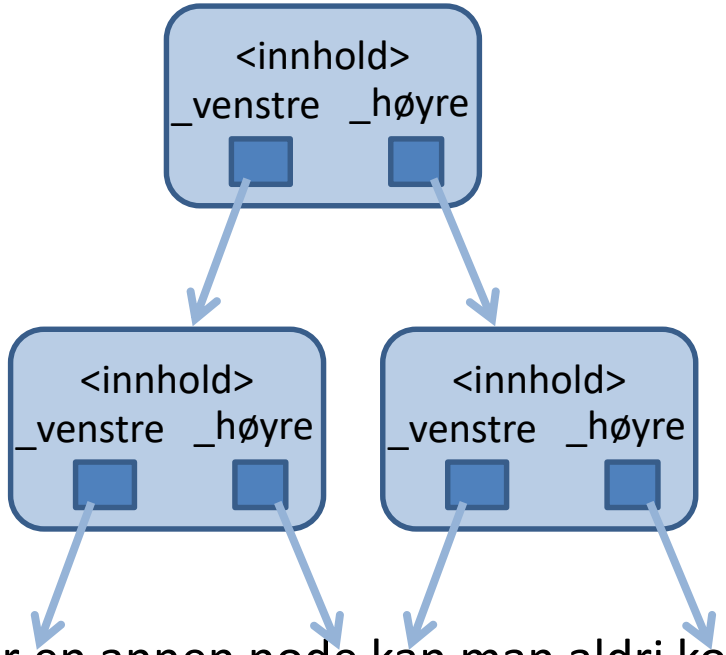
Andre sentrale datastrukturer

- Lenkede lister har flere ulemper
  1. Hvis man har mange noder, blir veien gjennom strukturen veldig lang, spesielt til siste node
  2. Hvis man skal representere noder med flere enn en relasjon til andre noder, holder det ikke med én neste-referanse

1. Punkt 1 kan avhjelpes noe vha endringer i implementasjon av nodene og listene. Mer om dette i IN1010.

2. Kan være nyttig med andre strukturer, for eksempel *trær.*

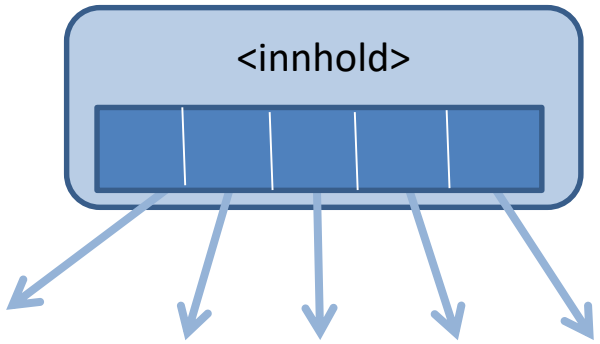
Trær



- Et tre har en node definert som **roten** i treet.
- Hver node kan ha flere **etterfølgere, barn**
- Trær har ikke sykler, dvs om man følger **kanter** fra roten eller en annen node kan man aldri komme tilbake til en man har vært i tidligere
- Spesielt binære trær (hver node har maks to etterfølgere) brukes ofte til å representere data for oppslag, sortering og traversering
- Antall etterfølgere kan ellers avhenge av hva datastrukturen skal representere

Trær

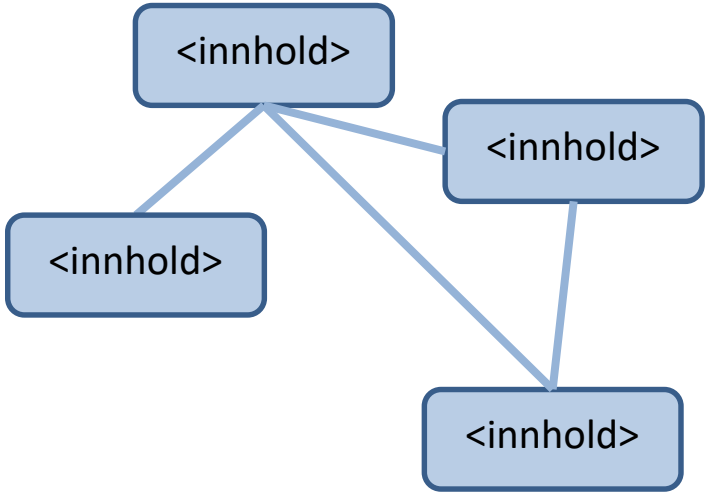
- Kan f eks modellere:
  - slektstrær
  - organisasjonskart



Generelt: Grafer

- En graf består av en mengde noder, og en mengde kanter, der hver kant forbinder to noder med hverandre.

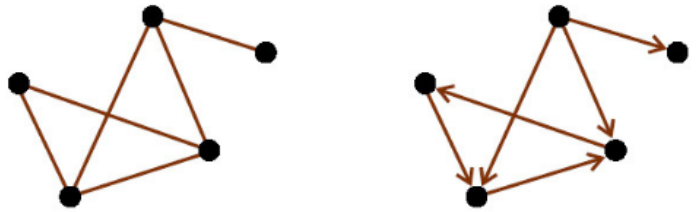
- En vei i en graf består av en mengde kanter i rekkefølge, slik at to på hverandre følgende kanter alltid har en node felles. En sti er en vei hvor hver node besøkes høyst én gang.



Formelt, hentet fra IN2010 (ikke IN1000 pensum)

Hva er en graf?

- En graf G =(V,E) har en mengde noder, V, og en mengde kanter, E
- |V| og |E| er henholdsvis antall noder og antall kanter i grafen
- Hver kant er et par av noder, dvs. (u, v) slik at $u, v \in V$
- En kant (u, v) modellerer at u er relatert til v
- Dersom nodeparet i kanten (u, v) er ordnet (dvs. at rekkefølgen har betydning), sier vi at grafen er rettet, i motsatt fall er den urettet



Urettet graf

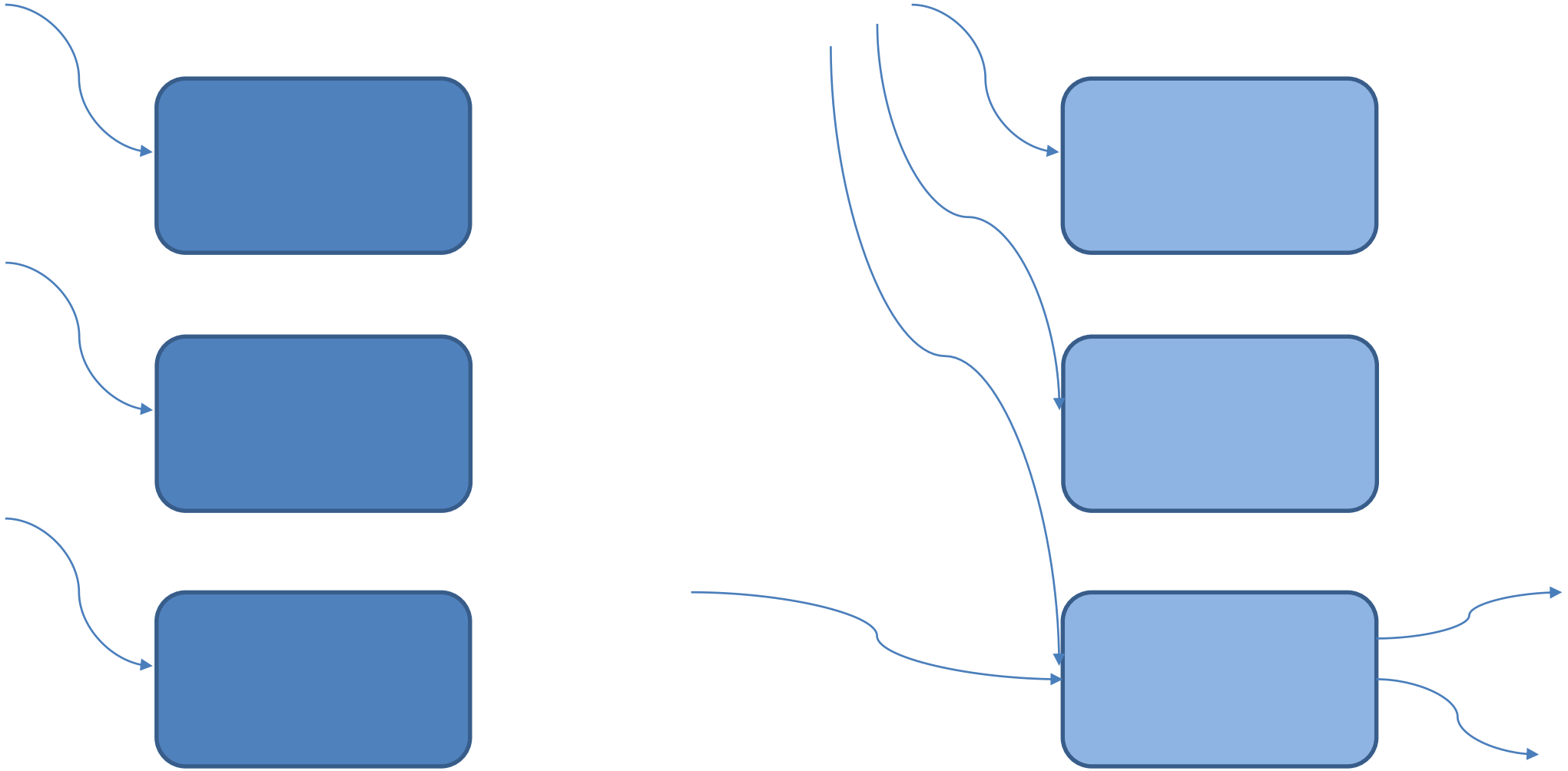
Rettet graf

- Grafer er den mest fleksible datastrukturen vi kjenner ("alt" kan modelleres med grafer)

Ingrid Chieh Yu (Ifi, UiO) INF2010 H2018, forelesning 4 8 / 54

Live eksempel II - kollektivnettet

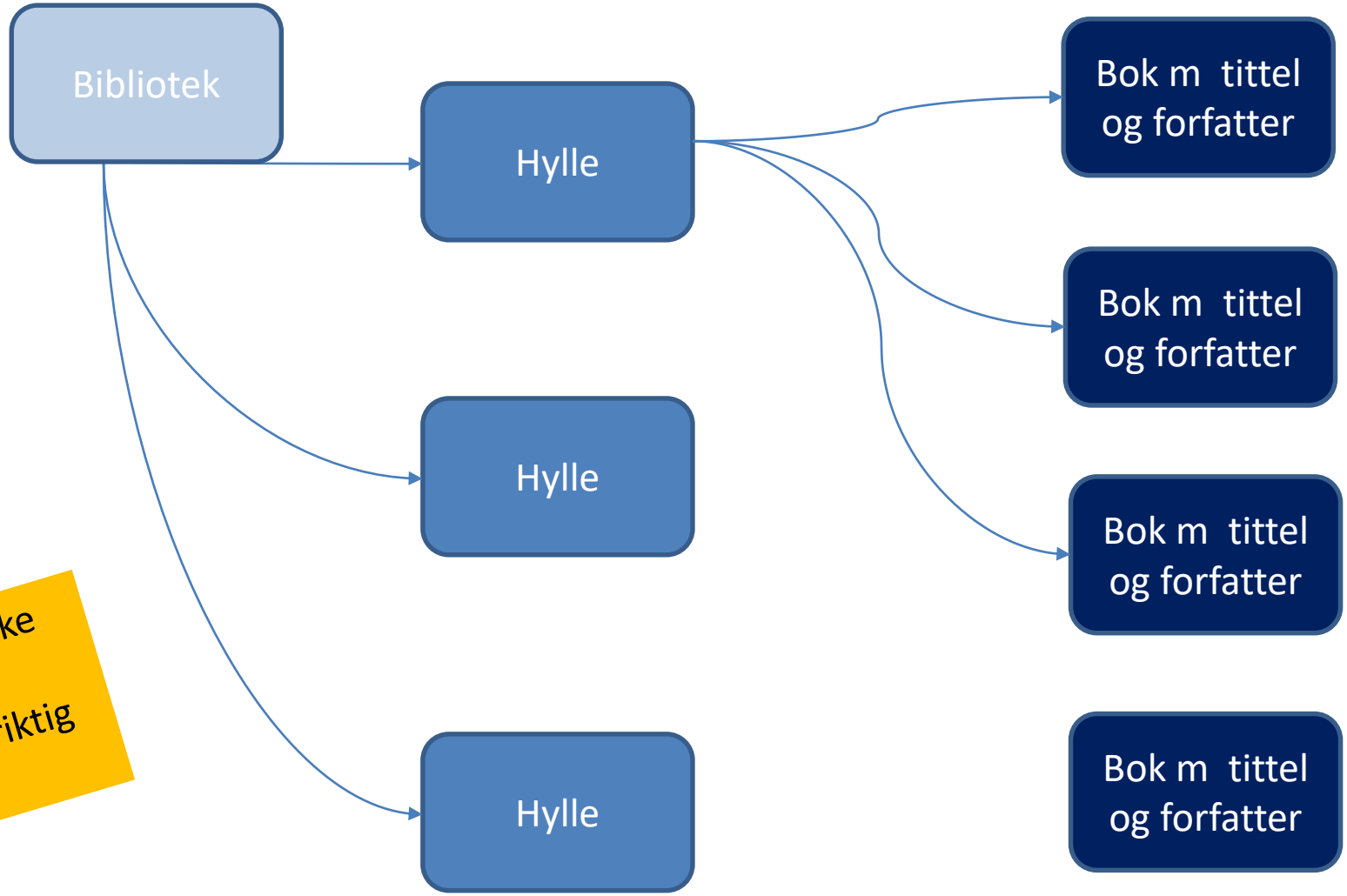
Objekter gir stor fleksibilitet og kan knyttes sammen på utallige måter



Vi kan f eks *splitte* sammensatt innhold og legge det i ulike klasser

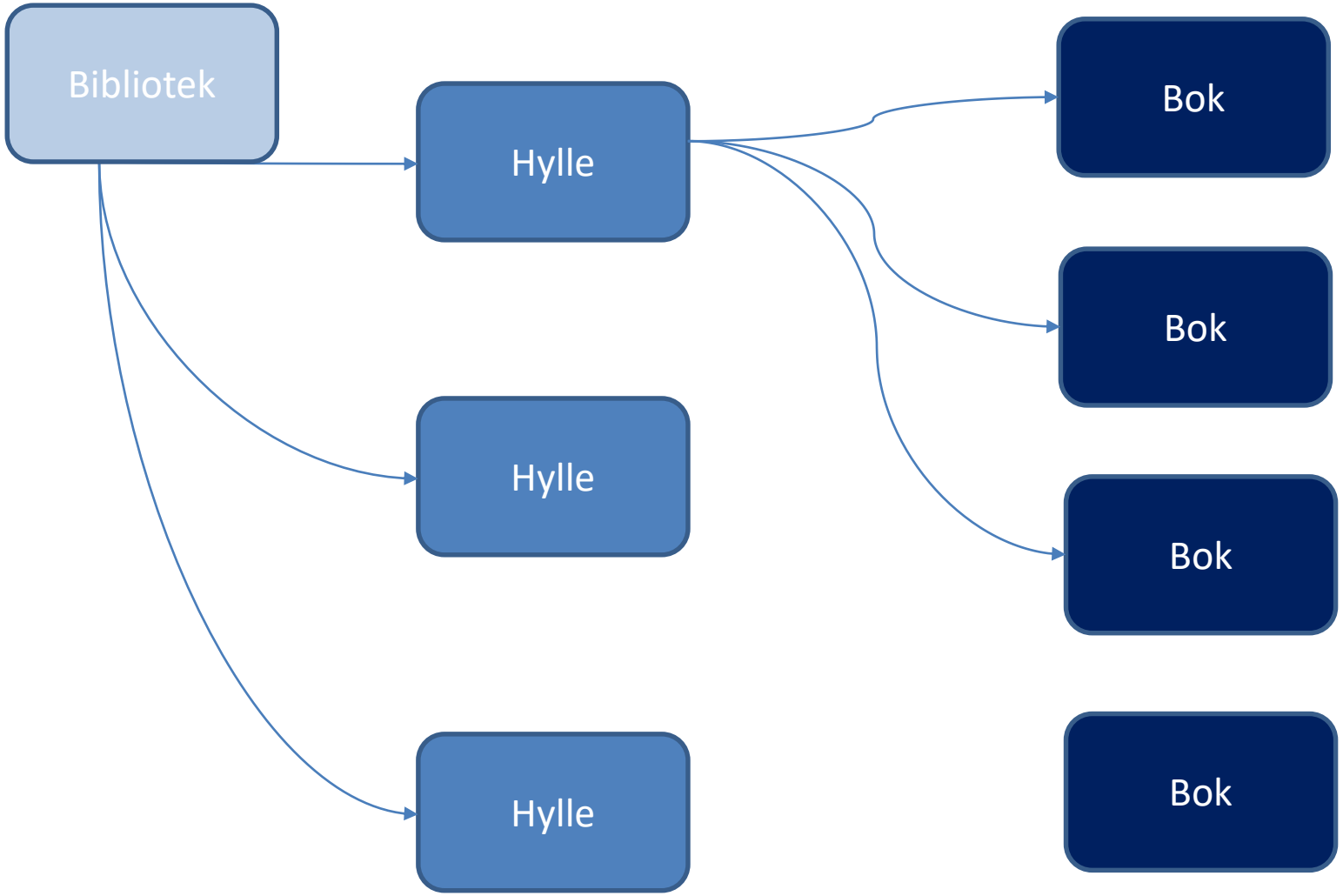
Bibliotek med hyller der hver hylle inneholder en eller flere bøker og hver bok har en tittel og en forfatter

Vi kan f eks *splitte* sammensatt innhold og legge det i ulike klasser

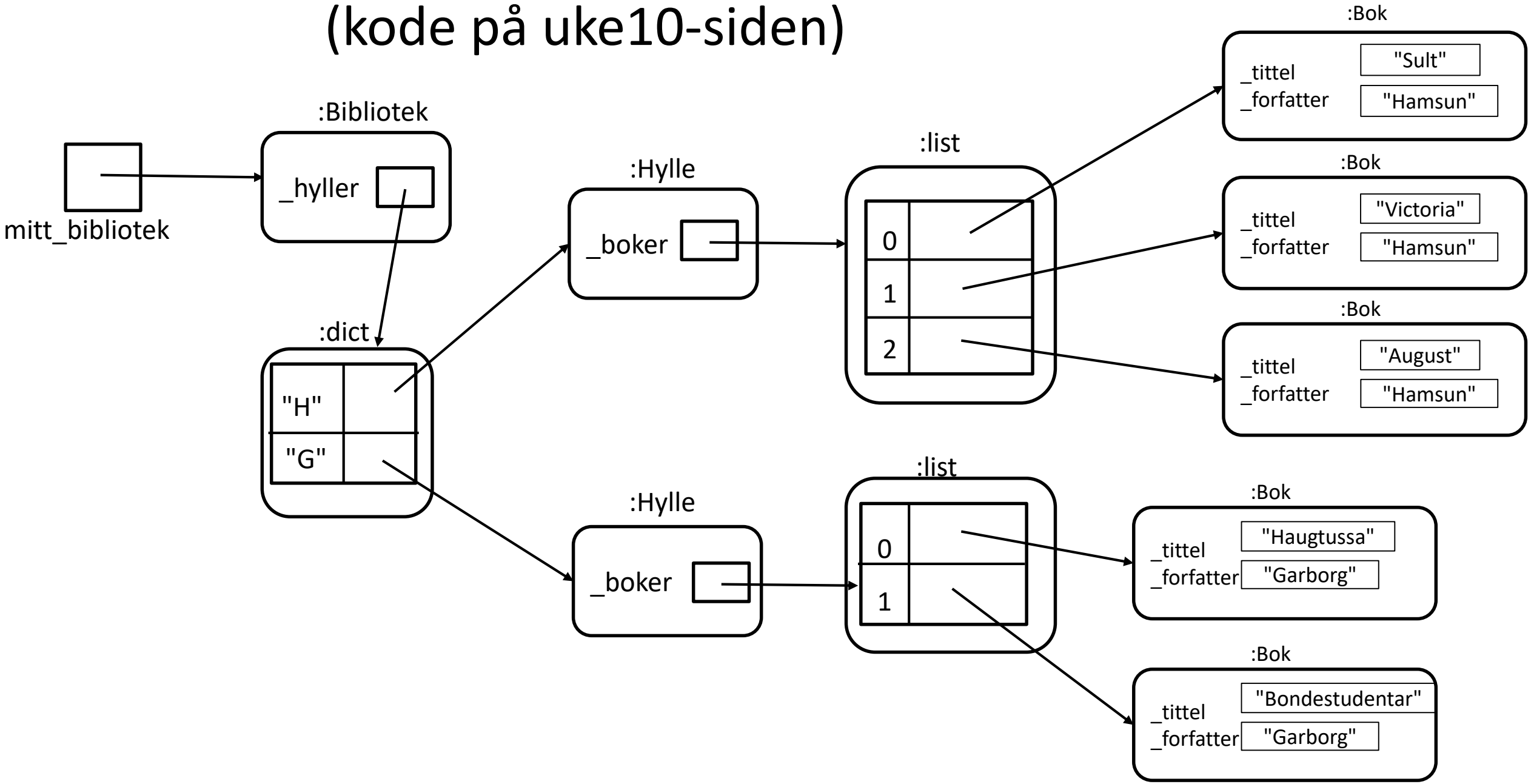


Forenklet tegning - ikke tegn slik i oblig!
Se senere slide for riktig eksempel

Kalles gjerne *aggregering* ("oppsamling") når man betrakter hele strukturen



Datastruktur-tegning i IN1000 oblig-format (kode på uke10-siden)

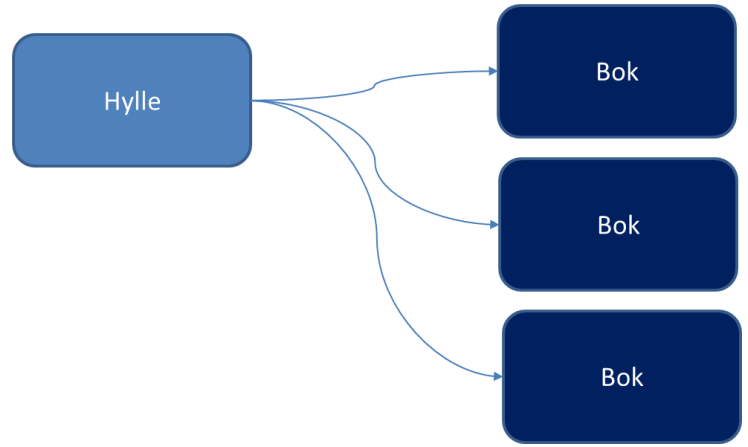


For å knytte sammen to objekter...

- Trenger vi en instansvariabel i klassen til objektet referansen går fra (eks instansvariabelen _navn i klassen Person)
- Opprette objektet vi skal referere til (av klassen Navn)
- Opprette objektet vi skal referere fra (av klassen Person)
- Knytte instansvariabelen i objektet vi skal referere fra (_navn), til objektet vi skal referere til

Hvis strukturen/ klassene er gitt, er det oftest enklest å jobbe "nedenfra og opp – starte med den klassen som ikke vet om (refererer til) andre klasser

- Hvis vi skal referere til mange objekter av samme type (eks av klassen Bok) fra et objekt (eks Hylle), er instansvariabelen i klassen Hylle typisk en liste eller ordbok som heter noe meningsfylt som f eks _boker
- Grovt: Hvis rekkefølgen er viktig velger vi en **liste**, hvis vi ønsker å slå opp fra en unik nøkkel (tittel? Ikke så entydig, kanskje ISBN?) velger vi en **ordbok**



Oblig 8

- To uker til leveringsfrist
- Start med å lese oppgaven nå
- Domenet og kravene som skal møtes er gitt i oppgaven – tar litt tid å sette seg inn i – akkurat det er 100% realistisk!

- Få en ekstra dytt på Fagutvalgets IN1000-seminar neste onsdag (ikke kast bort den tiden med å se på oppgaven for første gang)