Forelesning 13

NP er klassen av *ja-nei*-problemer der ethvert ja-svar har et bevis som kan sjekkes i polynomisk tid. Alt i NP kan i polynomisk tid reduseres til såkalt *komplette* problemer i NP. Dermed kan ikke disse løses i polynomisk tid, med mindre *alt* i NP kan det. Ingen har klart det så langt . . .

Pensum

- ☐ Kap. 34. NP-completeness
- ☐ Oppgave 34.1-4 med løsning (0-1 knapsack)

Læringsmål

- [M1] Forstå *optimering* vs *beslutning*
- [M2] Forstå *koding* av en instans
- [M3] Forstå at løsningen på 0-1-ryggsekkproblemet *ikke er polynomisk*
- [M4] Forstå *konkrete* vs *abstrakte* problemer
- [M5] Forstå repr. av beslutningsproblemer som *formelle språk*
- [M6] Forstå def. av P, NP og co-NP
- [M7] Forstå *redusibilitets-relasjonen*
- [M8] Forstå def. av NP-Hard og NPC
- [M9] Forstå den konvensjonelle hypotesen om P, NP og NPC
- [M10] Forstå hvordan NP-kompletthet kan bevises ved én reduksjon
- [M11] Kjenne noen NPC-problemer
- [M12] Forstå at *0-1-ryggsekk* er NPH
- [M13] Forstå at *lengste enkle vei* er NPH
- [M14] Kunne konstruere enkle NPC-bevis

Forelesningen filmes

Forelesning 13

NP-kompletthet

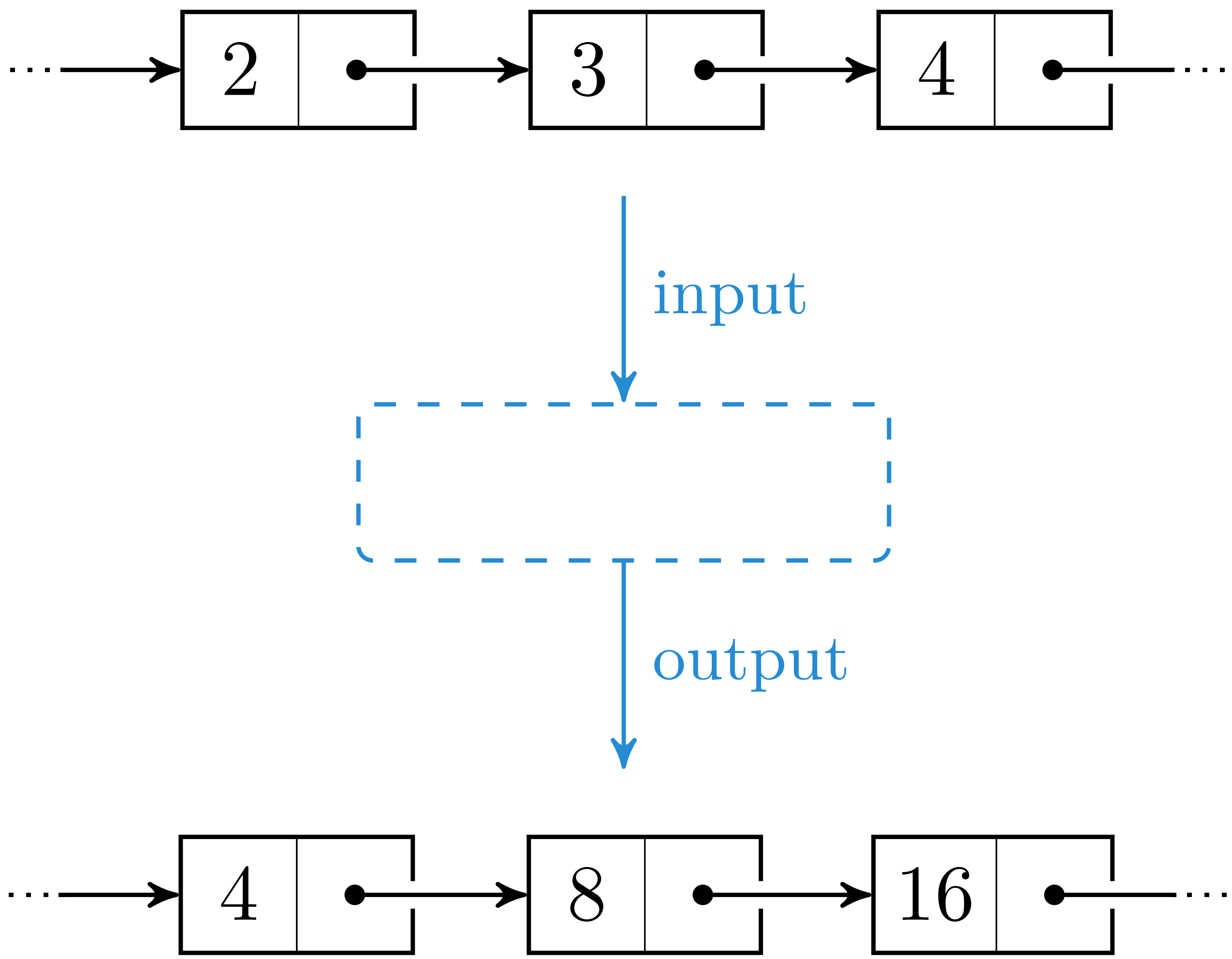
1. **Problemer**

2. **Reduksjoner**

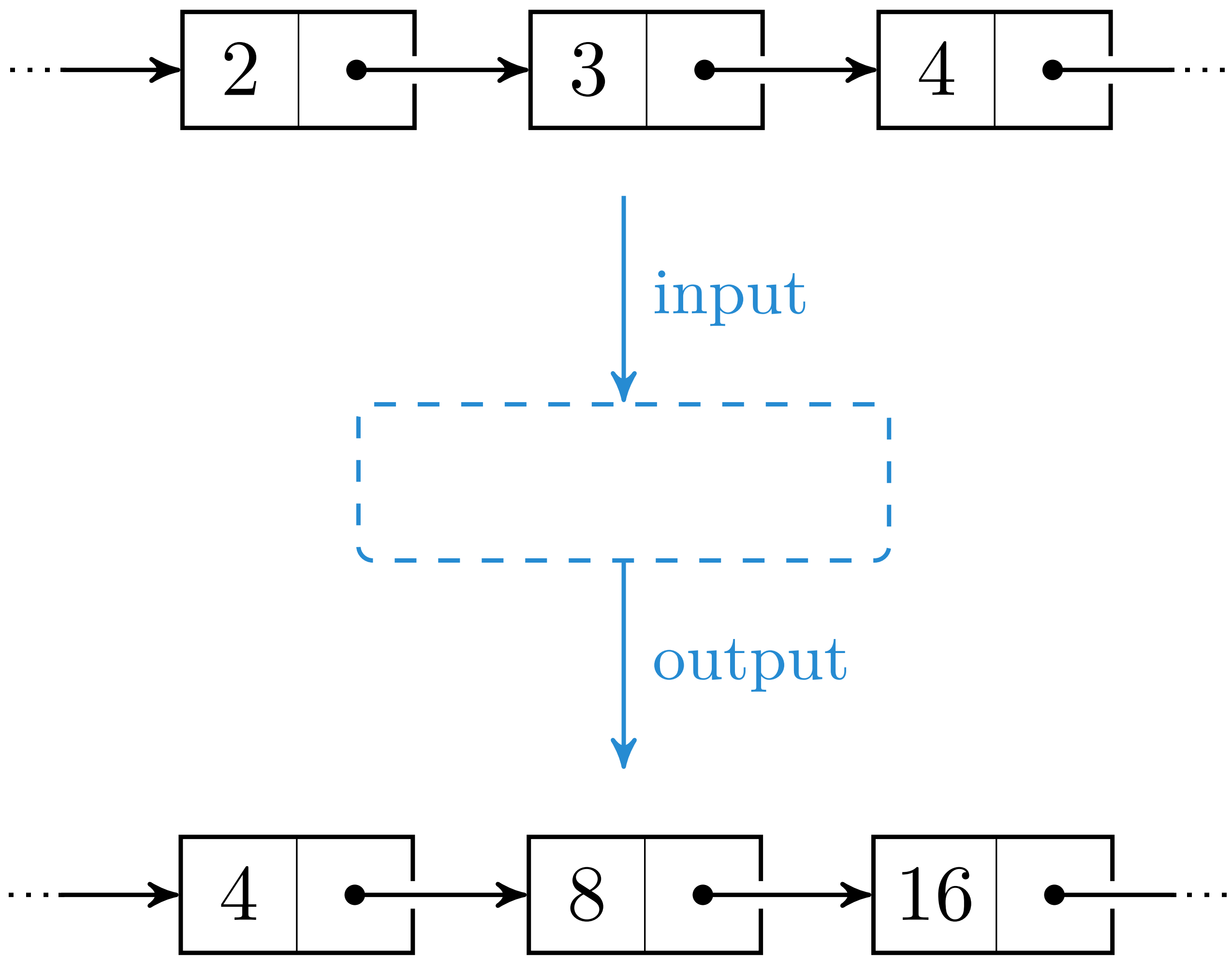
3. **Kompletthet**

1:4

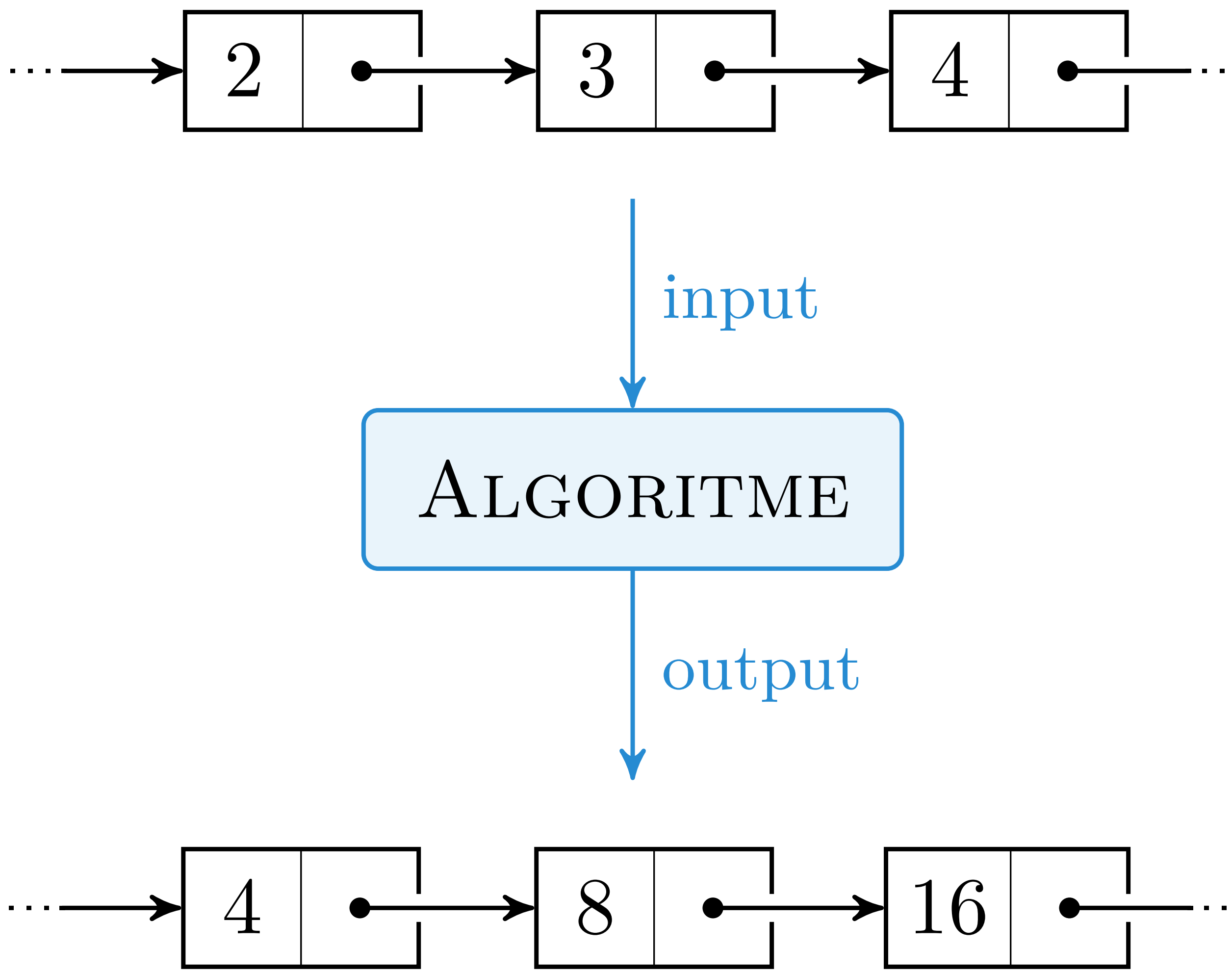
Problemer



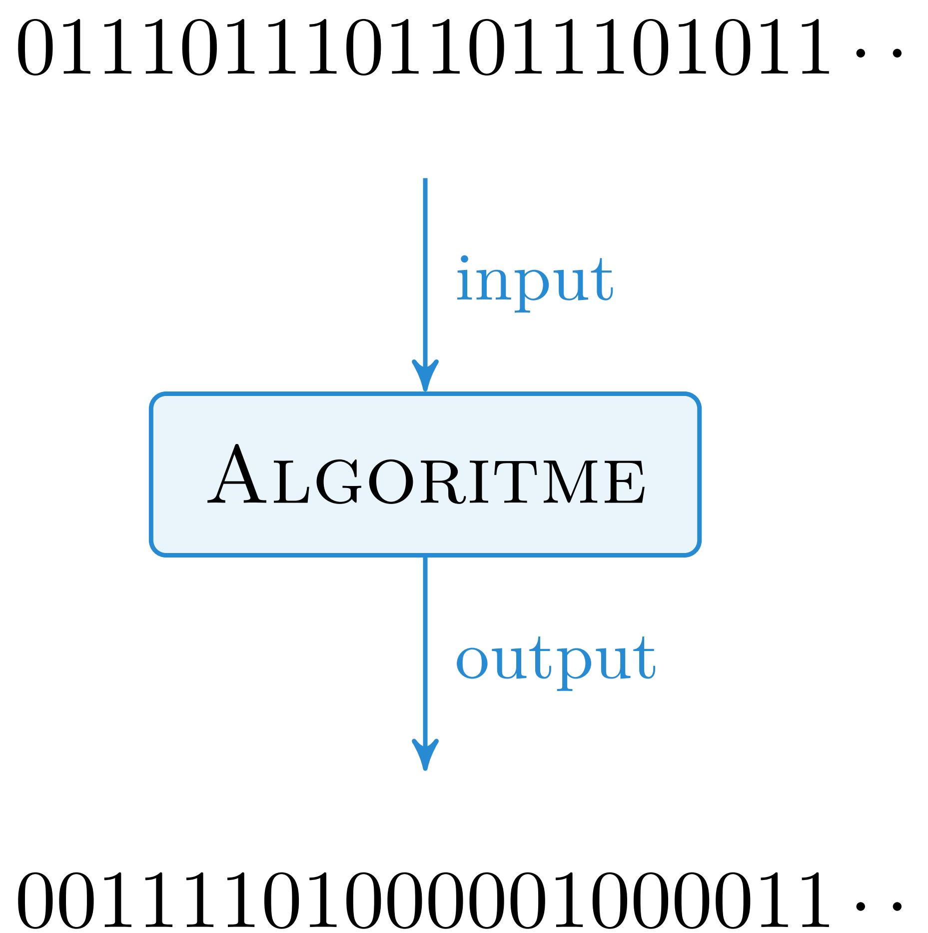
Et <u>problem</u> er en relasjon mellom input og output



Input og output kan være vilkårlige abstrakte objekter



Jobben vår er å produsere gyldig output



Et problem kalles <u>konkret</u> hvis input og output er bitstrenger

$0111011101101110101 1\cdots$

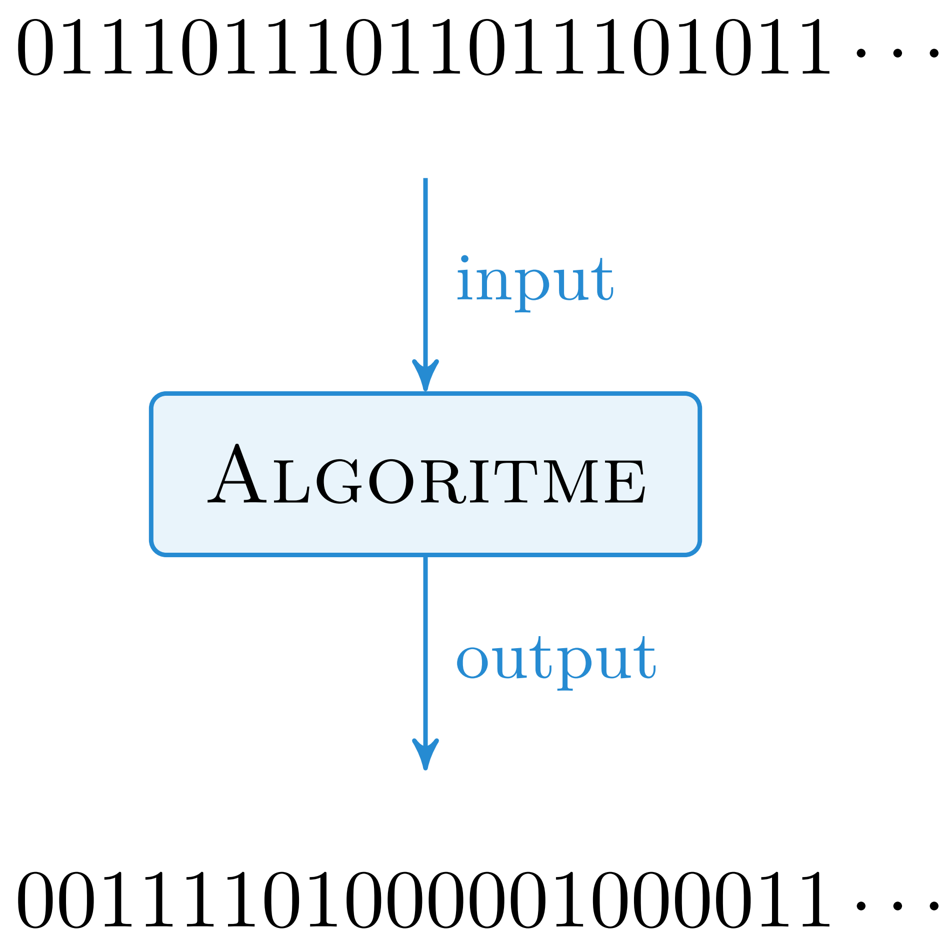
input

Algoritme

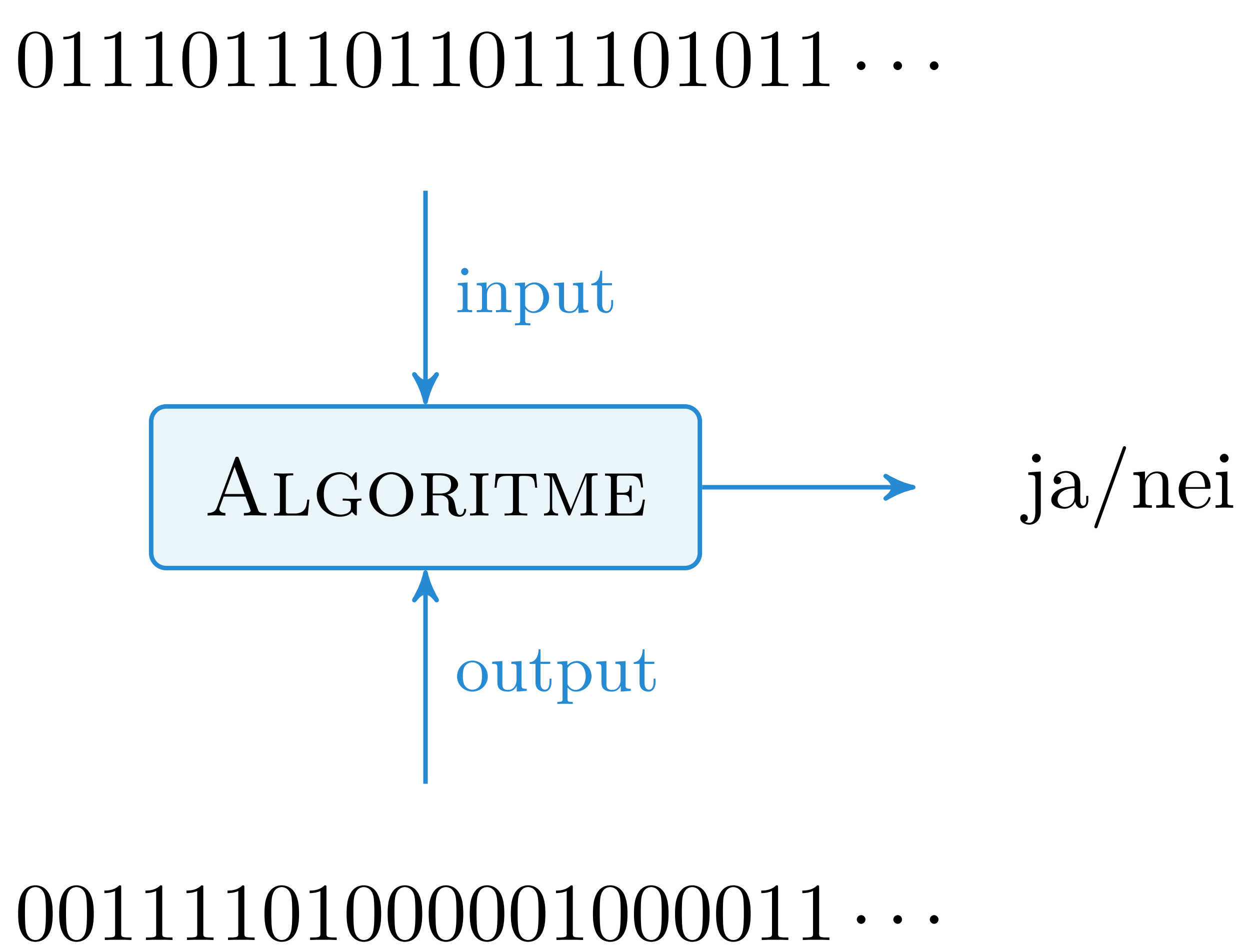
output

$0011110100000100001 1\cdots$

Vi koder instanser og resultater som bits



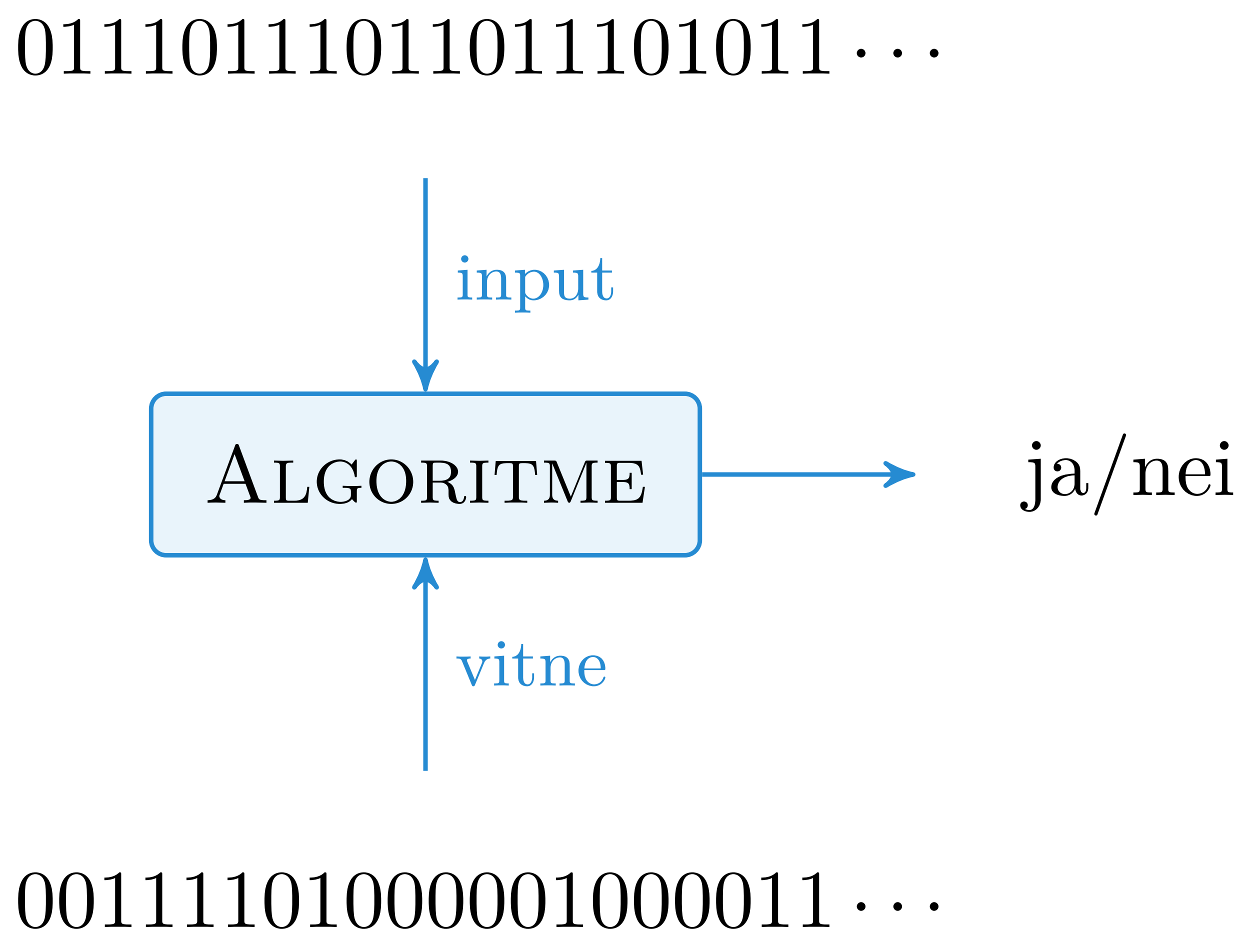
Egentlig ikke så viktig; vi vil bare forenkle «universet» vårt



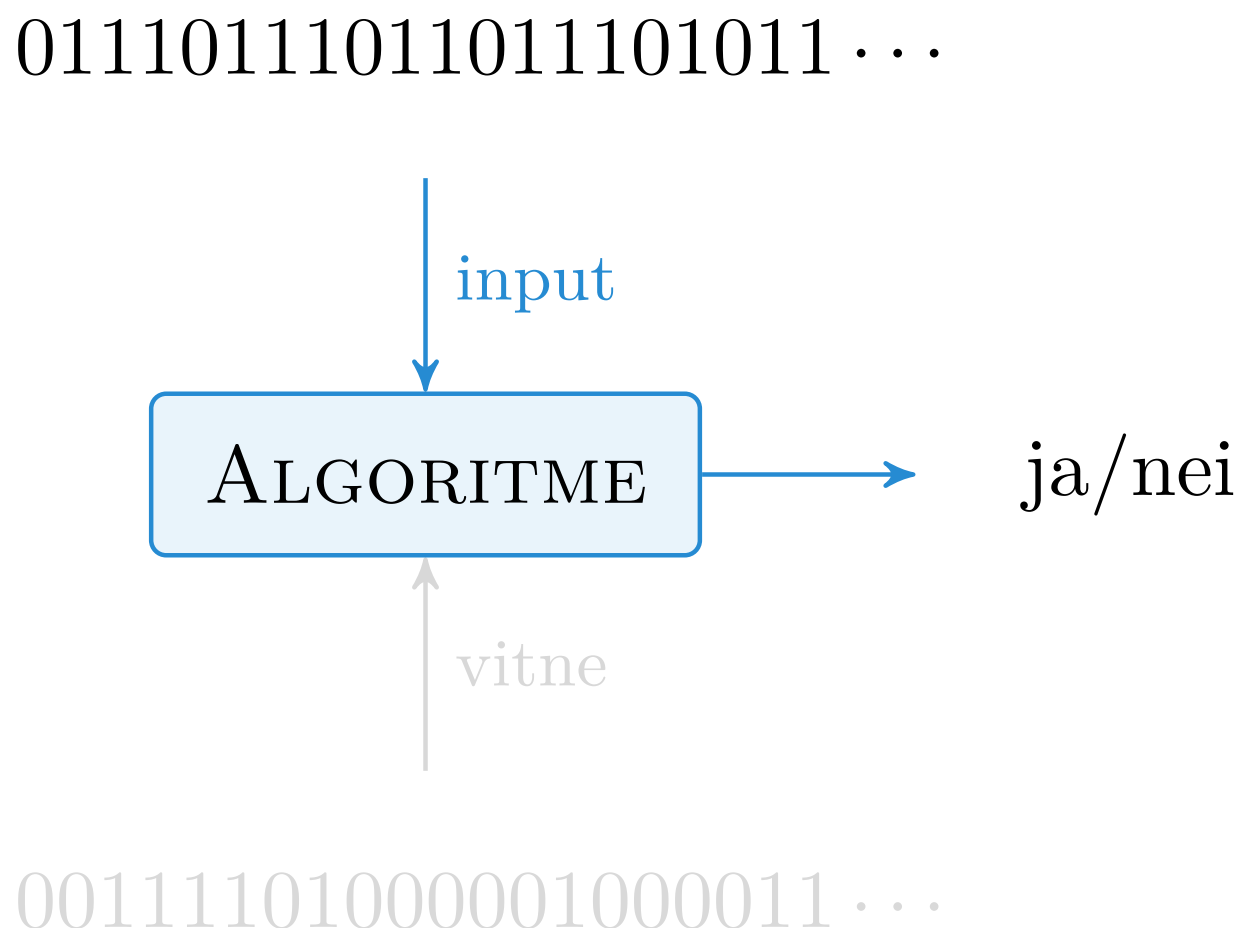
Og … vi kan da tenke oss at det kanskje ikke *finnes* noen gyldig løsning.

En verifikasjonsalgoritme sjekker om en løsning stemmer

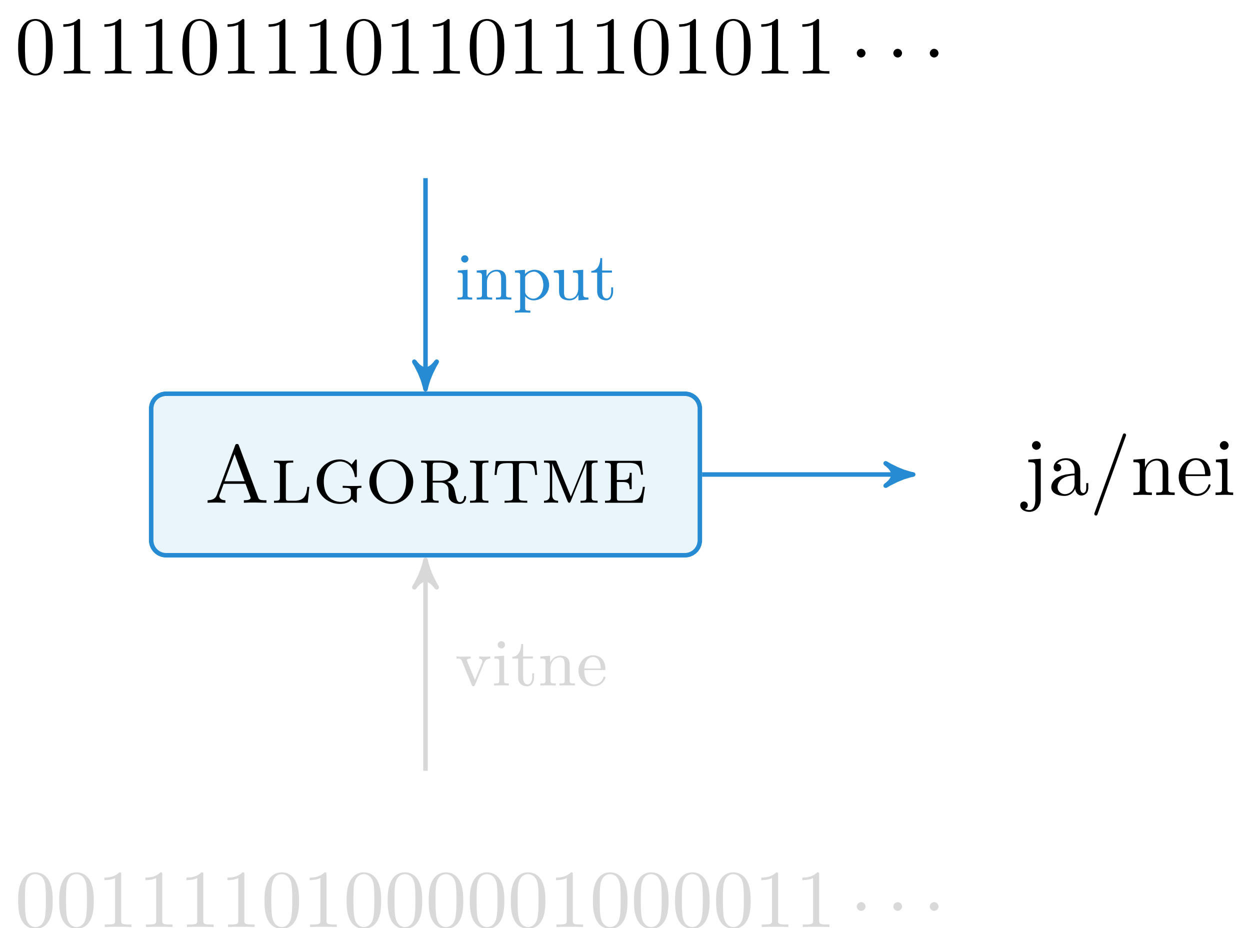
Kanskje vanligere perspektivt: Vi stiller et ja/nei-spørsmål, og hvis svaret er ja, så er sertifikatet et «bevis» for det. (Vi kan også ha sertifikat for nei, eller bare for én av delene.)



Vi kaller da gjerne løsningen et «sertifikat» eller «vitne»



Et beslutningsproblem kan vi tenke på som å stille spørsmålet . . .



«Finnes det et vitne?»

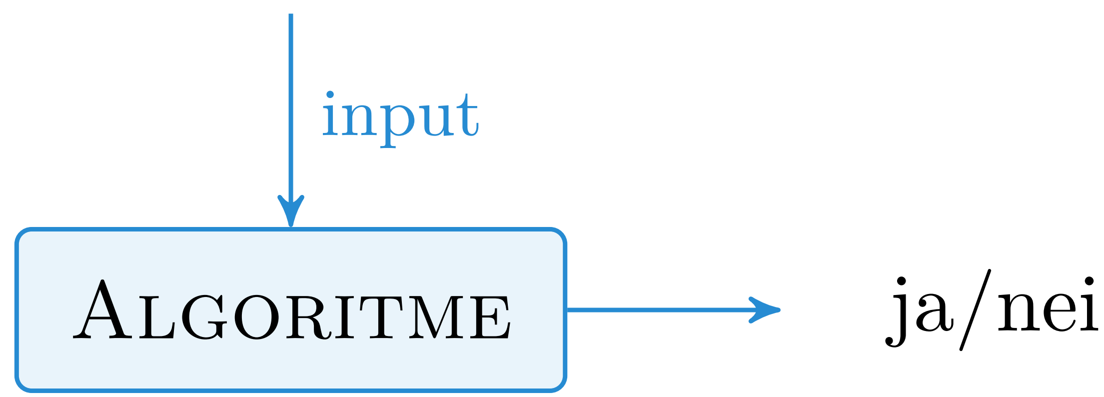
$0111011101101110101 1\cdots$

input ↓

ALGORITME → ja/nei

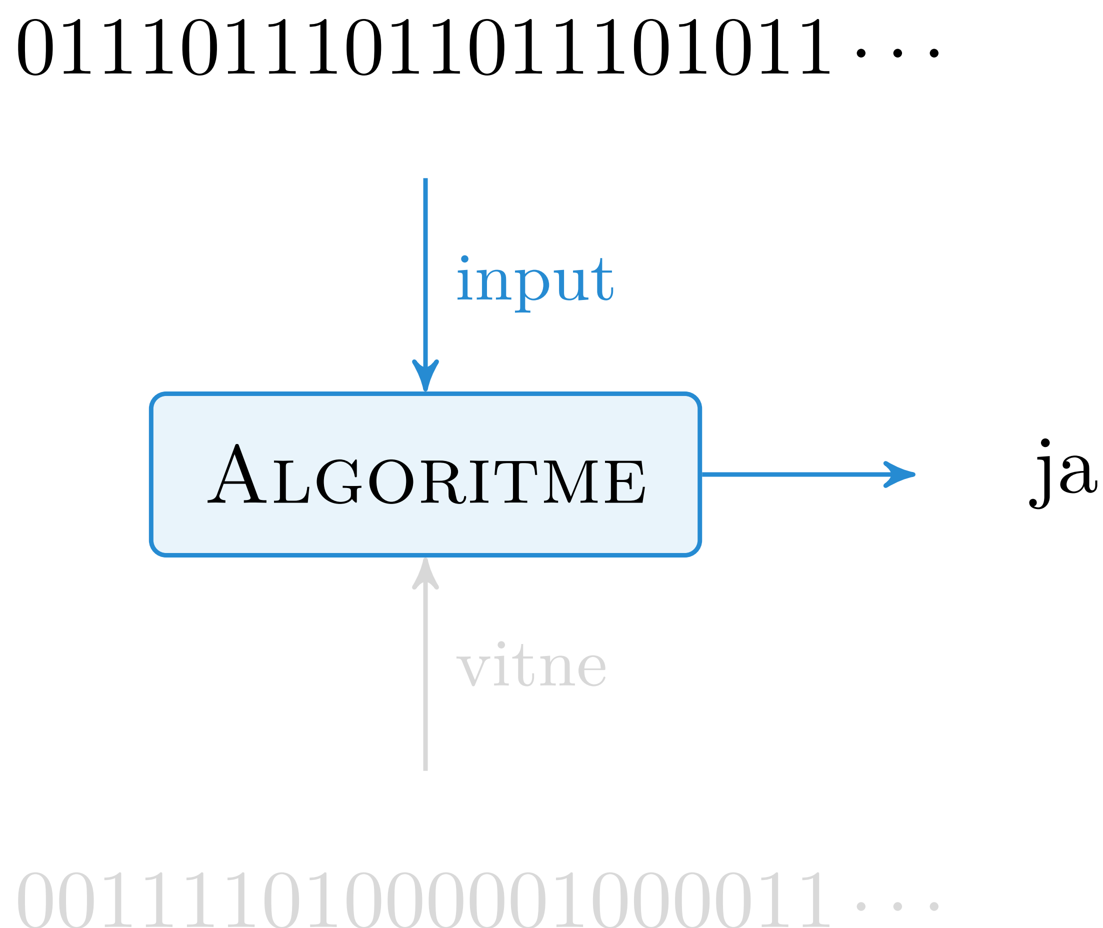
Mer generelt: Egentlig et r … som helst ja/nei-spørsmål

$$0111011101101110101 1\cdots$$



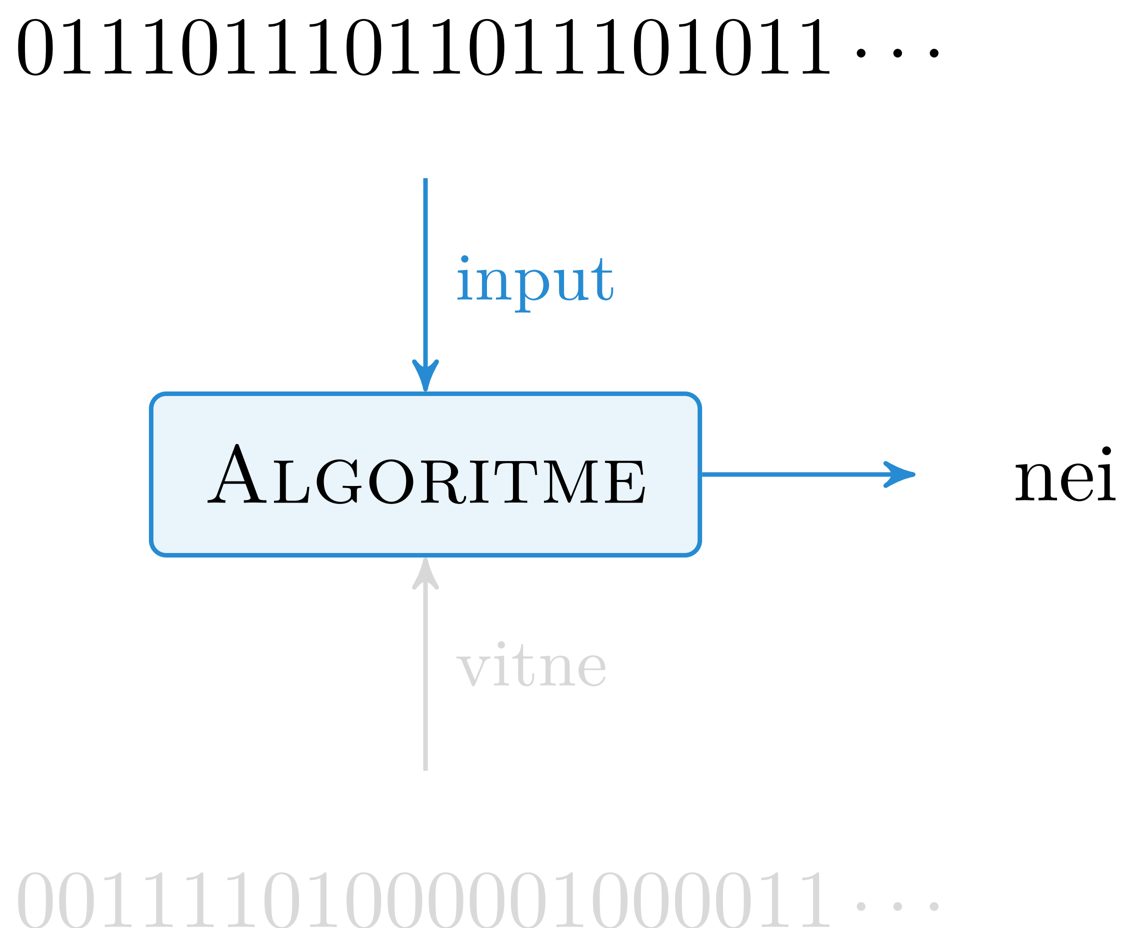
Pussig nok: Ekvivalent med å kunne løse bare ja-instanser i polynomisk tid. (Tilsvarer distinksjonen mellom decide og accept for språk; se teorem 34.2.)

Klassen **P** er slike problemer som kan løses i polynomisk tid



NP står for «non-deterministic polynomial time» – om vi på magisk vis kan gjette svaret (ikke-deterministisk), så kan vi altså løse problemet i polynomisk tid.

NP: Ja-svar har vitner som kan sjekkes i pol. tid



co-NP: Nei-svar har vitner som kan sjekkes i pol. tid

- **Optimering: Ikke nødvendigvis noe vitne**
- **Lag beslutningsproblem med terskling**
- **Avgjøres vha. optimering, som da er minst like vanskelig**
- **Hvis P = NP kan vi finne optimum vha. binærsøk med terskelen**

› **Problem som språk:**
Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke

- **Problem som språk:**
  Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke
- **Accept, reject, decide:**
  En algoritme A *aksepterer* x dersom $\mathrm{A}(x) = 1$.
  Den *avviser* x dersom $\mathrm{A}(x) = 0$.
  Den *avgjør* et språk L dersom …

- **Problem som språk:**
  Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke
- **Accept, reject, decide:**
  En algoritme A *aksepterer* x dersom $\mathrm{A}(x) = 1$.
  Den *avviser* x dersom $\mathrm{A}(x) = 0$.
  Den *avgjør* et språk L dersom …
  - $x \in \mathrm{L} \rightarrow \mathrm{A}(x) = 1$

- **Problem som språk:**
  Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke
- **Accept, reject, decide:**
  En algoritme A *aksepterer* x dersom $\mathrm{A}(x) = 1$.
  Den *avviser* x dersom $\mathrm{A}(x) = 0$.
  Den *avgjør* et språk L dersom ...
  - $x \in \mathrm{L} \to \mathrm{A}(x) = 1$
  - $x \notin \mathrm{L} \to \mathrm{A}(x) = 0$.

- **Problem som språk:**
  Konkrete beslutningsproblemer tilsvarer formelle språk (mengder av strenger). Ja-instanser er med, nei-instanser er ikke
- **Accept, reject, decide:**
  En algoritme A *aksepterer* x dersom $\mathrm{A}(x) = 1$.
  Den *avviser* x dersom $\mathrm{A}(x) = 0$.
  Den *avgjør* et språk L dersom . . .
  - $x \in \mathrm{L} \to \mathrm{A}(x) = 1$
  - $x \notin \mathrm{L} \to \mathrm{A}(x) = 0$.
- **Accept vs decide:**
  Selv om L er språket som *aksepteres* av A, så trenger ikke A *avgjøre* L, siden den kan la være å svare for nei-instanser (ved å aldri terminere)

› **Kompleksitetsklasse:** En mengde språk

- **Kompleksitetsklasse:** En mengde språk
- **P:** Språkene som kan avgjøres i polynomisk tid

- **Kompleksitetsklasse:** En mengde språk
- **P:** Språkene som kan avgjøres i polynomisk tid

Pussig nok, også språkene som aksepteres i pol. tid! (Thm. 34.2)

- **Kompleksitetsklasse:** En mengde språk
- **P:** Språkene som kan avgjøres i polynomisk tid
- **Cobham's tese:**
  Det er disse problemene vi kan løse i praksis

› **Sertifikat:**
En streng y som brukes som «bevis» for et ja-svar

- **Sertifikat:**
  En streng y som brukes som «bevis» for et ja-svar
- **Verifikasjonsalgoritme:**
  Tar inn sertifikat y i tillegg til instans x

- **Sertifikat:**
  En streng y som brukes som «bevis» for et ja-svar
- **Verifikasjonsalgoritme:**
  Tar inn sertifikat y i tillegg til instans x
- En algoritme A **verifiserer** x hvis det eksisterer et sertifikat y slik at $\mathrm{A}(x, y) = 1$

- **Sertifikat:**
  En streng y som brukes som «bevis» for et ja-svar
- **Verifikasjonsalgoritme:**
  Tar inn sertifikat y i tillegg til instans x
- En algoritme A **verifiserer** x hvis det eksisterer et sertifikat y slik at $\mathrm{A}(x, y) = 1$
- **Intuitivt:**
  Algoritmen «sjekker svaret». Om en graf har en Hamilton-sykel, kan sertifikatet være noderekkefølgen i sykelen.

- **Sertifikat:**
  En streng y som brukes som «bevis» for et ja-svar
- **Verifikasjonsalgoritme:**
  Tar inn sertifikat y i tillegg til instans x
- En algoritme A **verifiserer** x hvis det eksisterer et sertifikat y slik at $\mathrm{A}(x, y) = 1$
- **Intuitivt:**
  Algoritmen «sjekker svaret». Om en graf har en Hamilton-sykel, kan sertifikatet være noderekkefølgen i sykelen.
- **Asymmetrisk:**
  Det finnes ikke «motbevis» eller «anti-sertifikater»

› **NP:** Språkene som kan verifiseres i polynomisk tid

› **NP:** Språkene som kan verifiseres i polynomisk tid

N = Nondeterministic: Kan løses om vi klarer «gjette» sertifikater

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE ∈ **NP**
  Lett å verifisere i polynomisk tid

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*
- **co-NP:**
  Språkene som kan *falsifiseres* i polynomisk tid

$$\mathrm{L} \in \textbf{co-NP} \iff \bar{\mathrm{L}} \in \textbf{NP}$$

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*
- **co-NP:**
  Språkene som kan *falsifiseres* i polynomisk tid

$$\mathrm{L} \in \textbf{co-NP} \iff \bar{\mathrm{L}} \in \textbf{NP}$$

$\bar{\mathrm{L}}$ er *komplementet* til L: $x \in \bar{\mathrm{L}} \iff x \notin \mathrm{L}$

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*
- **co-NP:**
  Språkene som kan *falsifiseres* i polynomisk tid

$$\mathrm{L} \in \textbf{co-NP} \iff \bar{\mathrm{L}} \in \textbf{NP}$$

F.eks.: TAUTOLOGY

- **NP:** Språkene som kan verifiseres i polynomisk tid
- **HAM-CYCLE**
  Språket for Hamilton-sykel-problemet
- HAM-CYCLE $\in$ **NP**
  Lett å verifisere i polynomisk tid
  Merk: Ikke nødvendigvis lett å *falsifisere*
- **co-NP:**
  Språkene som kan *falsifiseres* i polynomisk tid

$$\mathrm{L} \in \textbf{co-NP} \iff \bar{\mathrm{L}} \in \textbf{NP}$$

F.eks.: TAUTOLOGY

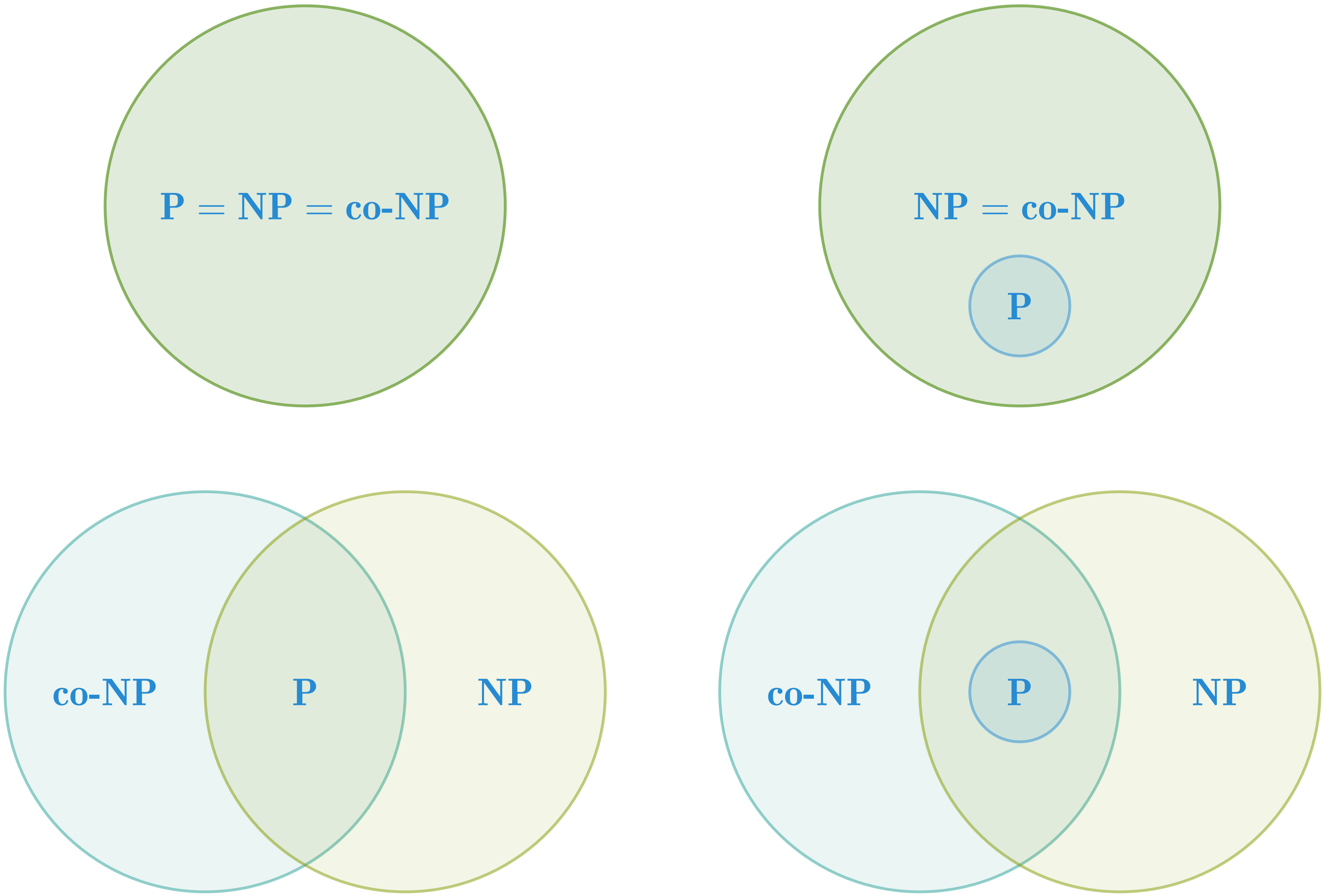
(Det er komplementet til negasjonen av SAT … som vi ser igjen siden)

› **P vs NP**

Om vi kan *løse* problemet, så kan vi *verifisere* det med samme algoritme, og bare ignorere sertifikatet

Dvs.: **P** $\subseteq$ **NP** og **P** $\subseteq$ **co-NP**

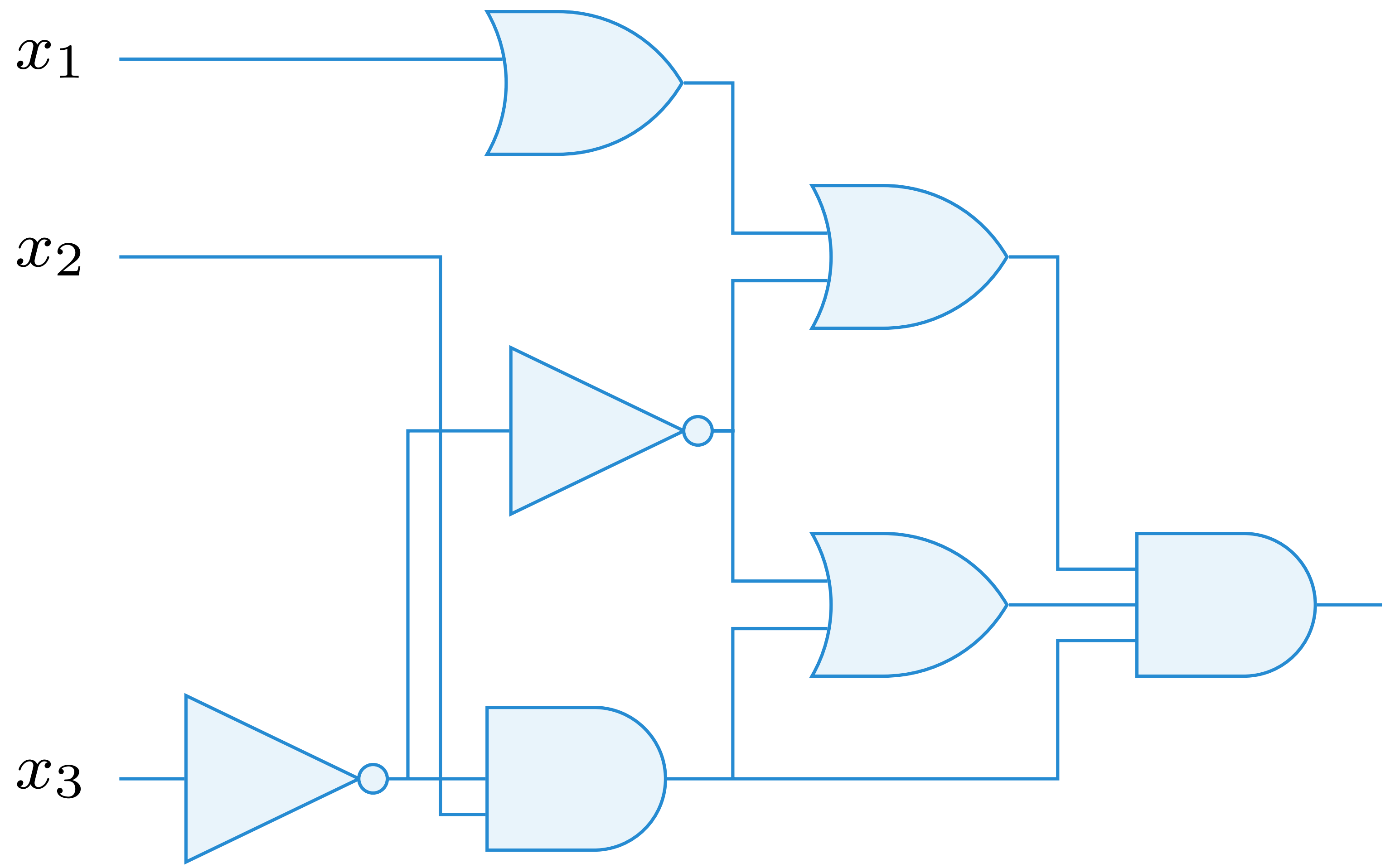
- **P vs NP**
  Om vi kan *løse* problemet, så kan vi *verifisere* det med samme algoritme, og bare ignorere sertifikatet
  Dvs.: $\mathbf{P} \subseteq \mathbf{NP}$ og $\mathbf{P} \subseteq \textbf{co-NP}$
- **Vi vet ikke** om $\mathbf{P} = \mathbf{NP} \cap \textbf{co-NP}$



Mulige scenarier; ingen vet hvilket som stemmer!

Disse problemene er NP-komplette; vi kommer til hva det betyr.

Noen (vanskelige) problemer



CIRCUIT-SAT

Instans: En krets med logiske porter og én utverdi

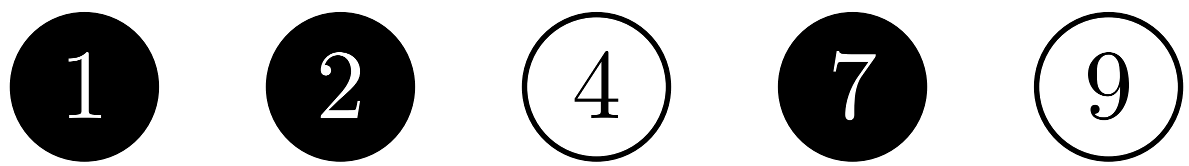
Spørsmål: Kan utverdien bli 1?

$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

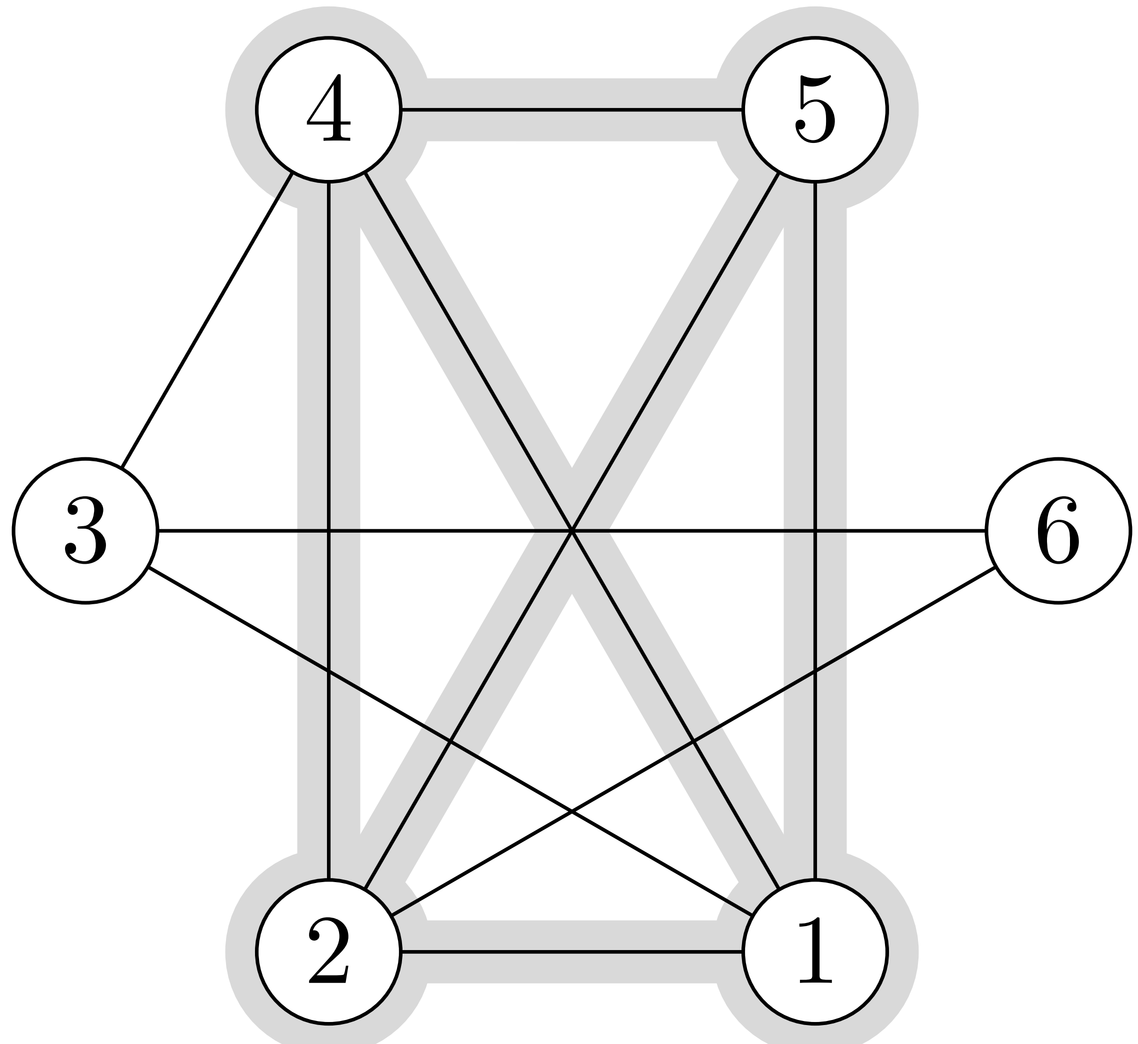
SAT
Instans: En logisk formel
Spørsmål: Kan formelen være sann?

$$\begin{aligned}\phi = {} & (x_1 \vee \neg x_2 \vee \neg x_3) \wedge \\ & (\neg x_1 \vee x_2 \vee x_3) \wedge \\ & (x_1 \vee x_2 \vee x_3)\end{aligned}$$

3-CNF-SAT
Instans: En logisk formel på 3-CNF-form
Spørsmål: Kan formelen være sann?



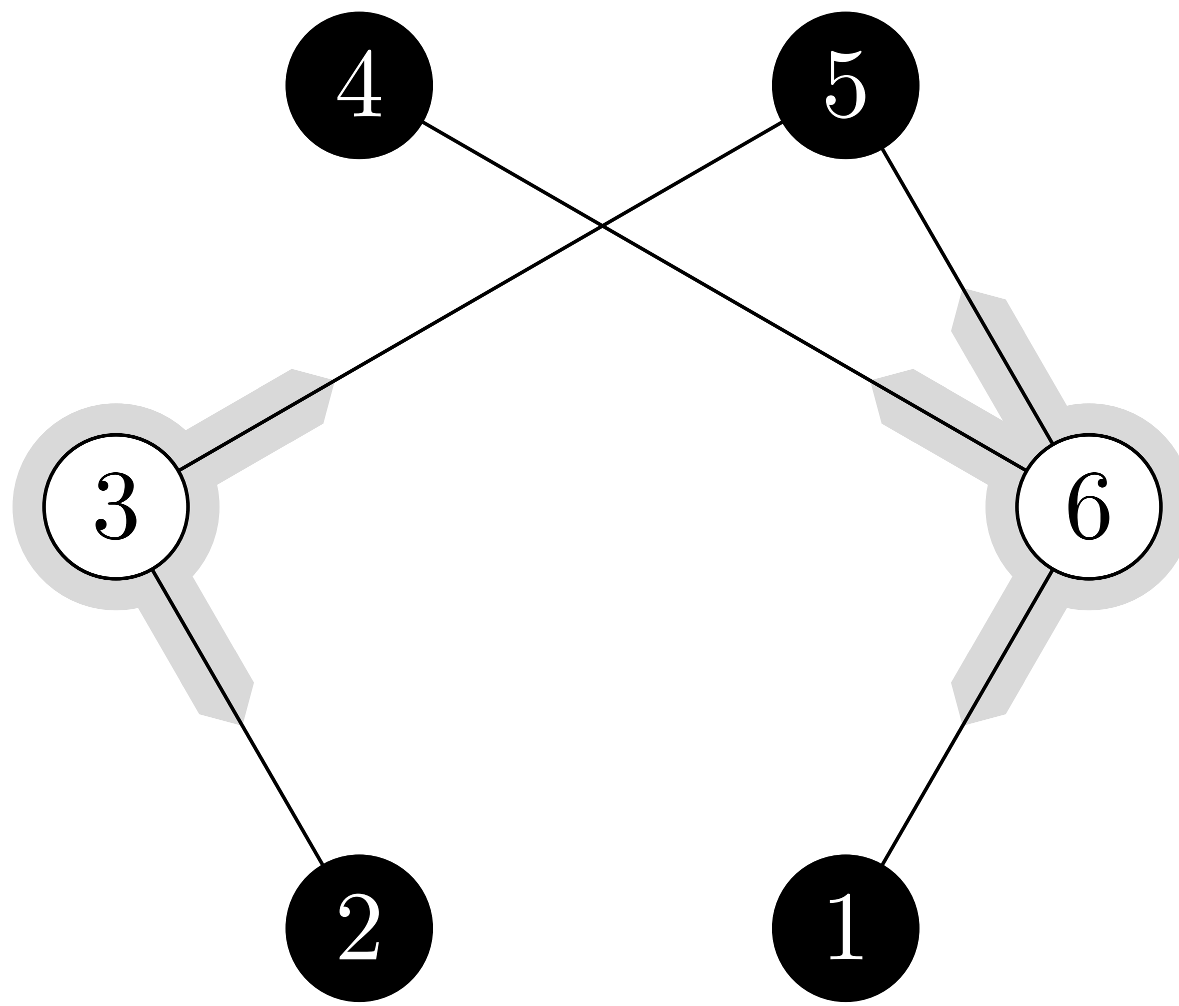
SUBSET-SUM
Instans: Mengde positive heltall S og positivt heltall t
Spørsmål: Finnes en delmengde $\mathrm{S}' \subseteq \mathrm{S}$ så $\sum_{s \in \mathrm{S}'} s = t$?



CLIQUE

Instans: En urettet graf G og et heltall k

Spørsmål: Har G en en komplett delgraf med k noder?

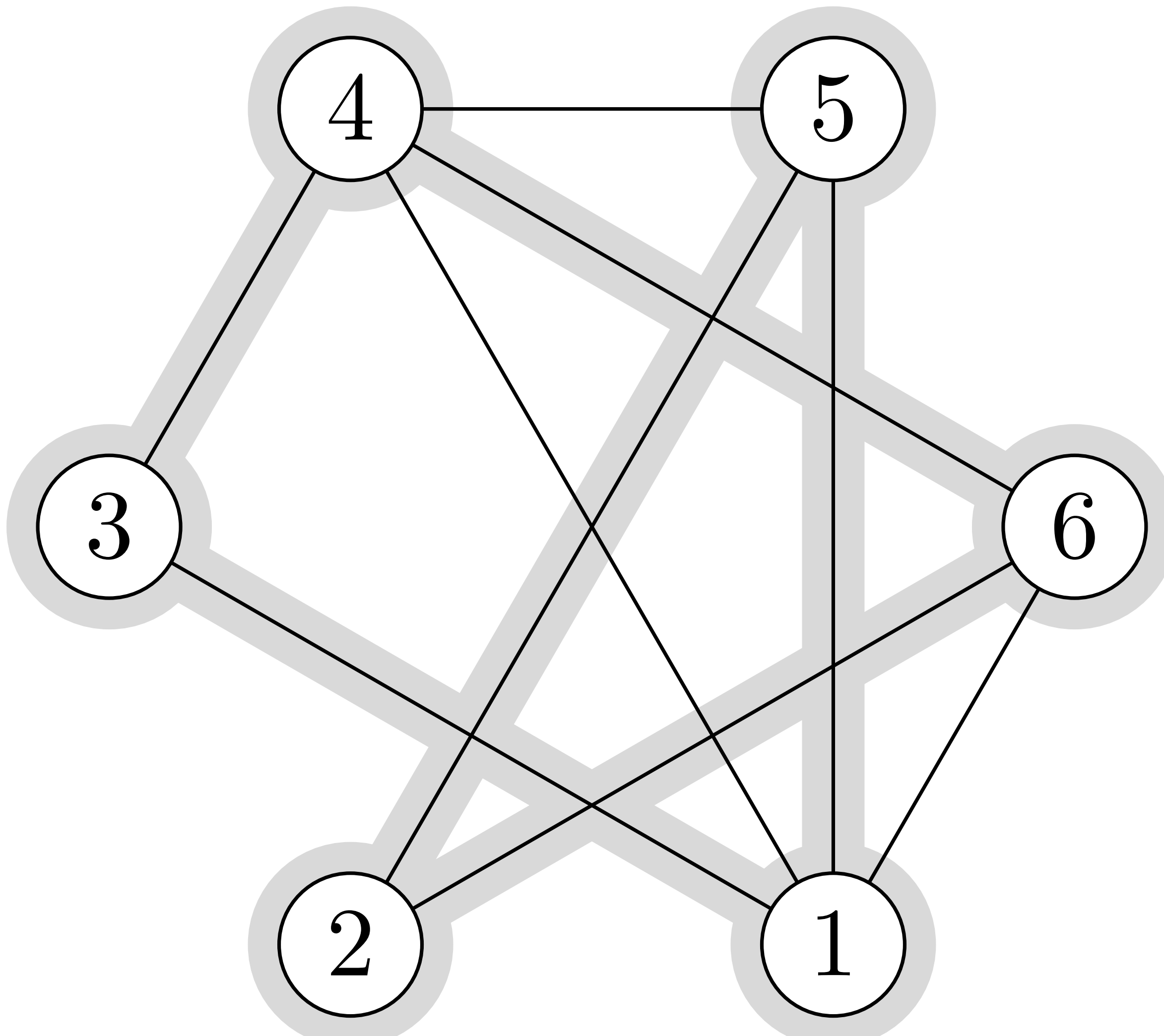


VERTEX-COVER

Instans: En urettet graf G og et heltall k

Spørsmål: Har G en et nodedekke med k noder?

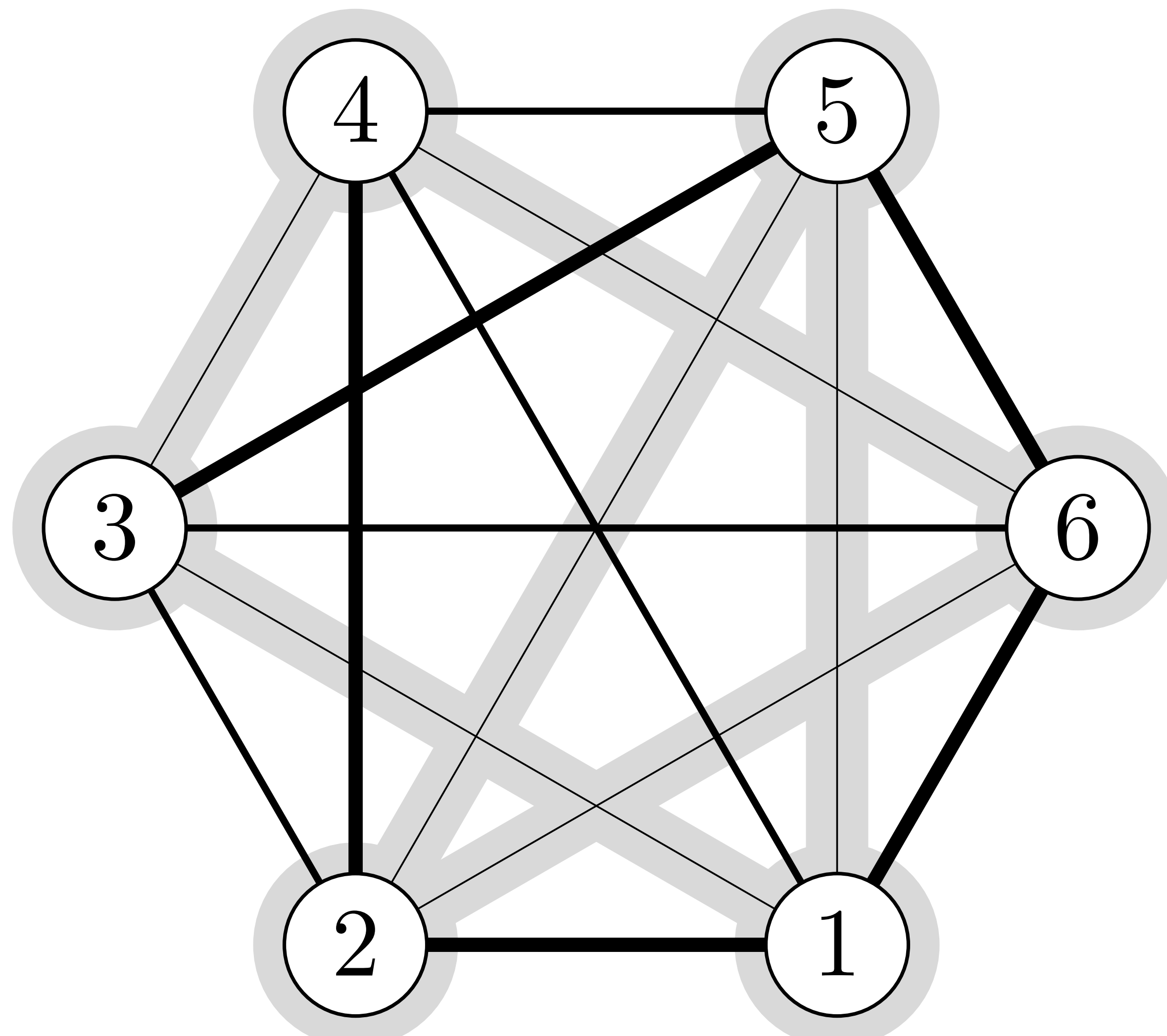
Dvs., k noder som tilsammen ligger inntil alle kantene



HAM-CYCLE

Instans: En urettet graf G

Spørsmål: Finnes det en sykel som inneholder alle nodene?



TSP

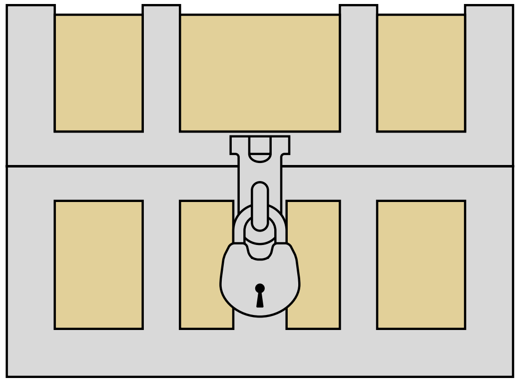
Instans: En komplett graf med heltallsvekter og et heltall k

Spørsmål: Finnes det en rundtur med kostnad $\leq k$?

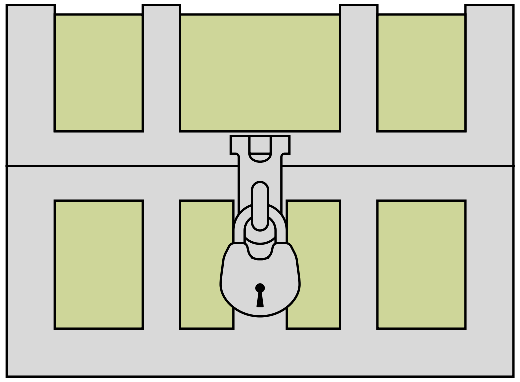
2:4

Reduksjoner

Spesifikt, såkalte Karp-reduksjoner: «Many-one»-reduksjoner som tar polynomisk lang tid.

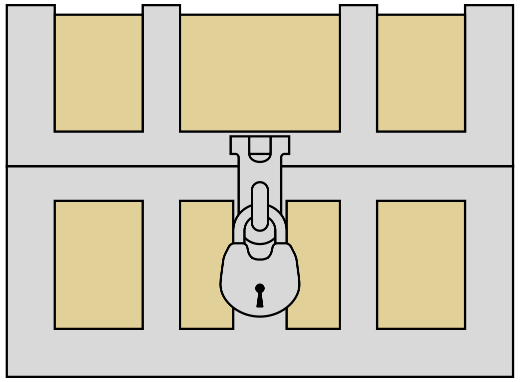


A

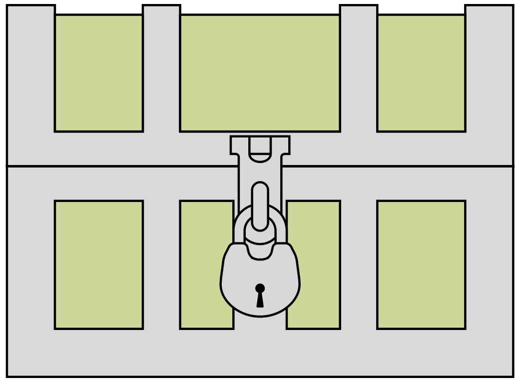


B

La oss si du har funnet to skattekister

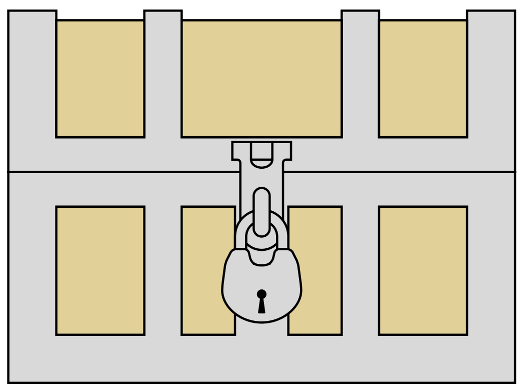


A

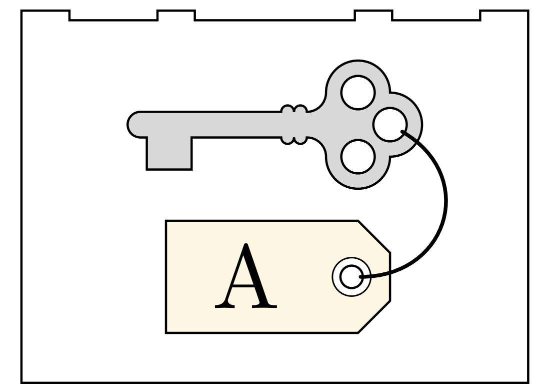


B

Du ønsker å si noe om hvor vanskelige de er å åpne

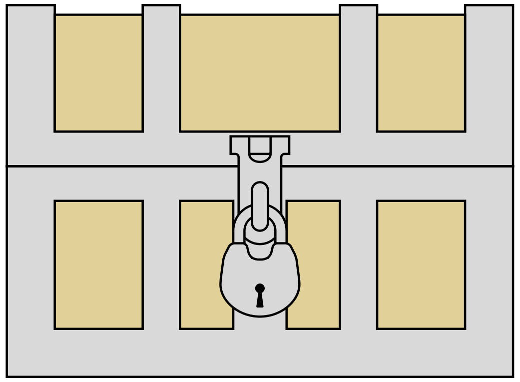


A

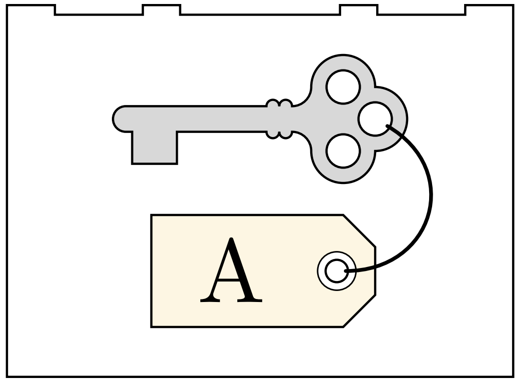


B

Anta at B inneholder nøkkelen til A

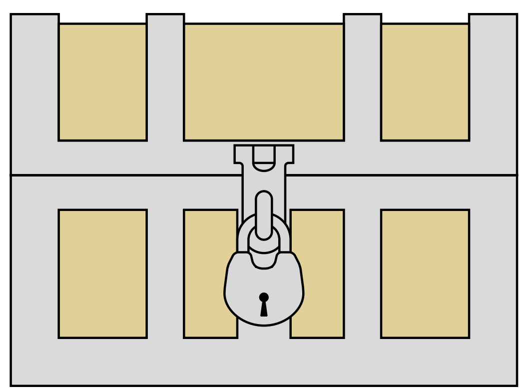


A

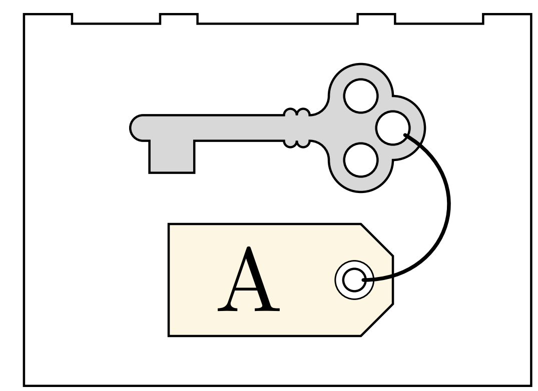


B

Kan det nå være vanskeligere å åpne A enn B?

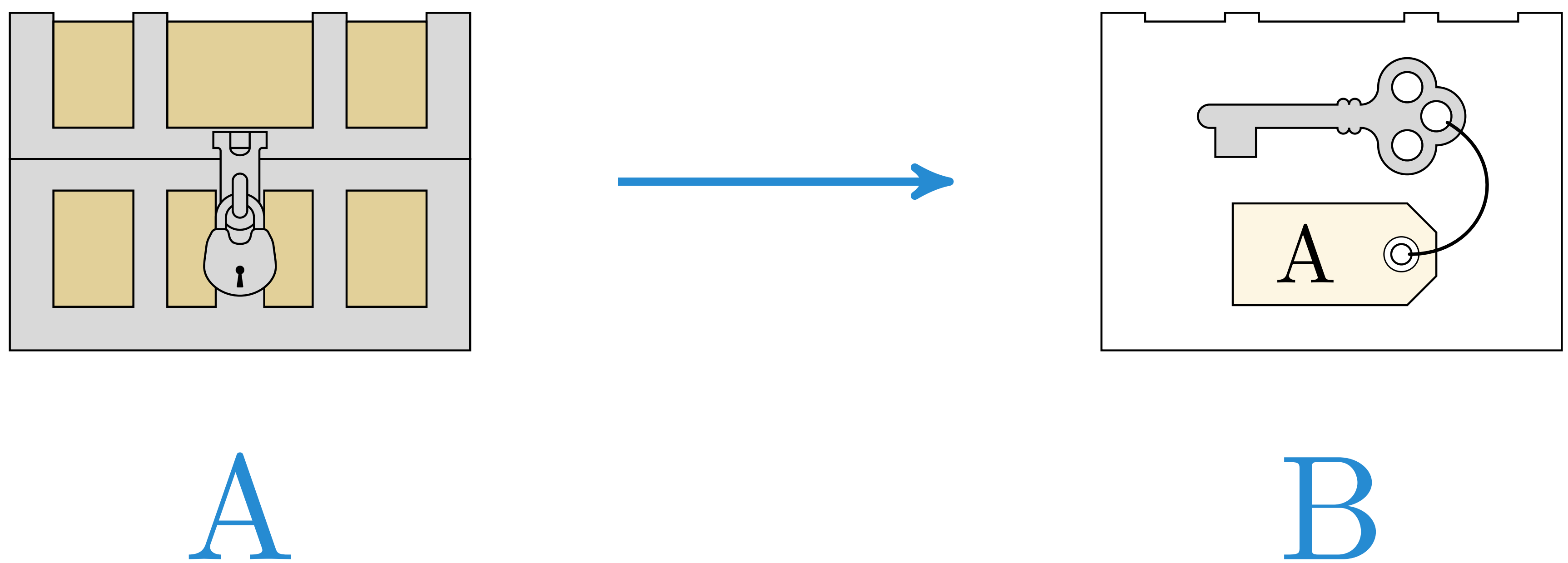


A

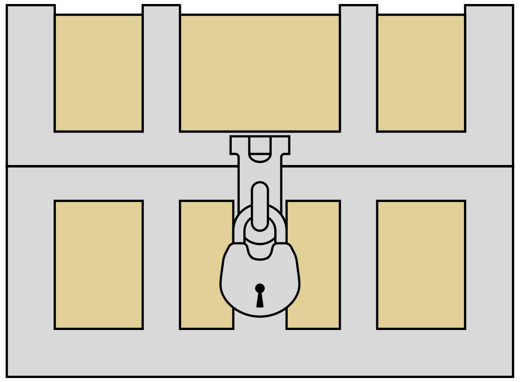


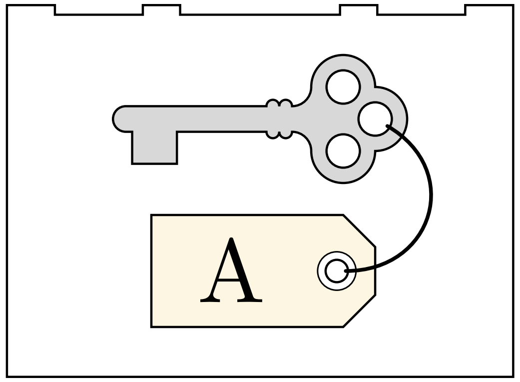
B

Nei! Om vi kan åpne B, så kan vi naturligvis åpne A



Vi har redusert problemet «åpne A» til problemet «åpne B»





A $\leqslant$ B

Da gir det ingen mening å si at A er vanskeligere enn B

Vi vil redusere fra beslutningsproblem Q til beslutningsproblem Q′

Vi formaliserer det som en reduksjon fra språk L_1 til språk L_2

Input: En bitstreng x.

Input: En bitstreng x.

Output: En bitstreng $f(x)$, der

$$x \in \mathrm{L}_1 \iff f(x) \in \mathrm{L}_2\,.$$

Input: En bitstreng x.

Output: En bitstreng $f(x)$, der

$$x \in \mathrm{L}_1 \iff f(x) \in \mathrm{L}_2 \,.$$

Om vi kan avgjøre om $f(x) \in \mathrm{L}_2$ kan vi også avgjøre om $x \in \mathrm{L}_1$

Input: En bitstreng x.

Output: En bitstreng $f(x)$, der

$$x \in \mathrm{L}_1 \iff f(x) \in \mathrm{L}_2 \,.$$

Men ikke omvendt!

Ikke omvendt … fordi det er ikke sikker f er invertibel. Om vi kan redusere fra L1 til L2 (med f), betyr ikke det at det finnes noen reduksjon fra L2 til L1.

Merk at det handler om retningen til f og ikke retningen til implikasjonen! Vi må ha "hvis og bare hvis" her, siden det bare betyr at de to svarene (ja eller nei) er de samme, og at reduksjonen er korrekt.

$$\alpha$$

$$\mathrm{A}(\alpha)$$

Et beslutningsproblem

α

β

$A(\alpha)$

$B(\beta)$

Enda et beslutningsproblem

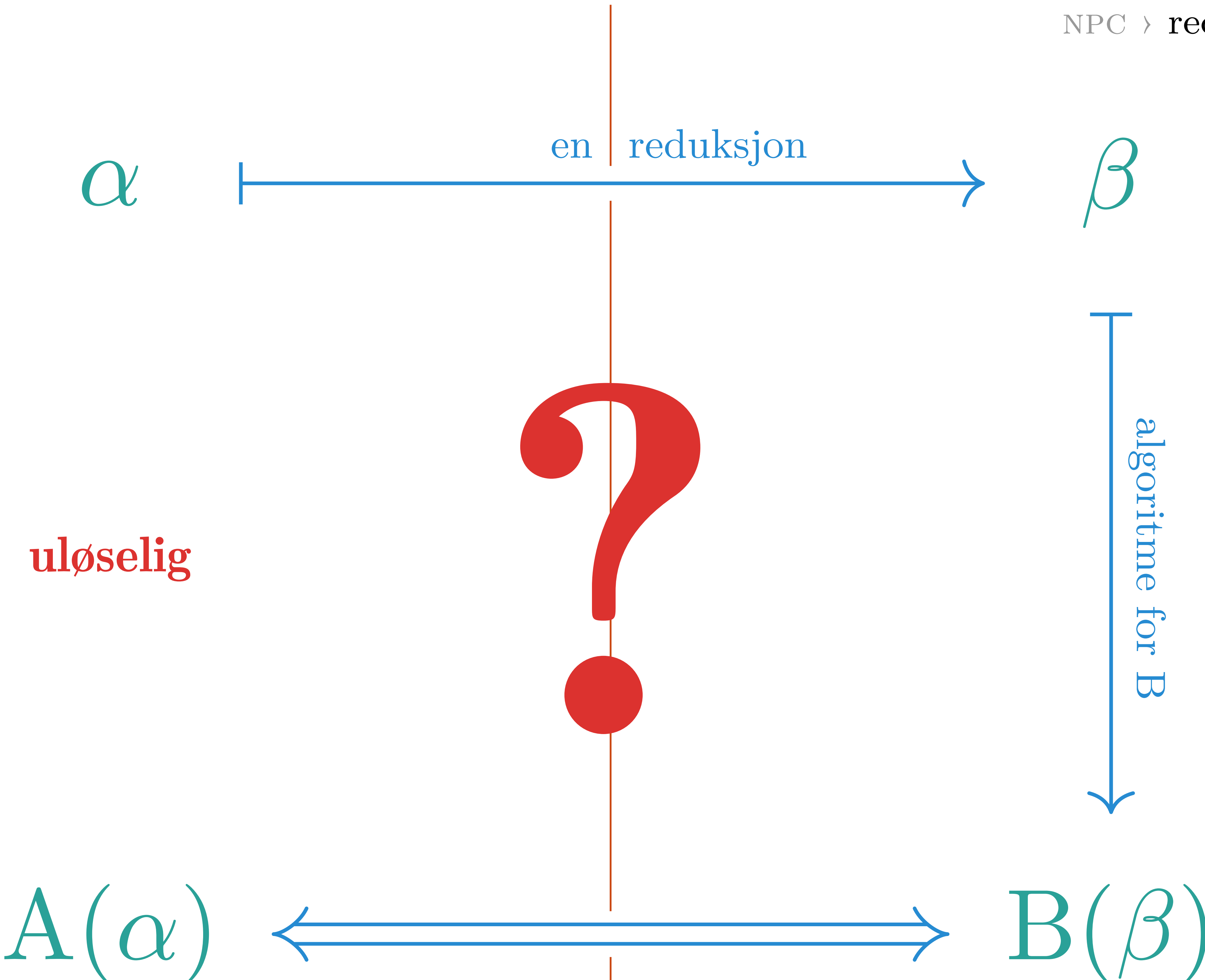
$$\alpha \xmapsto{\text{en reduksjon}} \beta$$

$$\mathrm{A}(\alpha) \iff \mathrm{B}(\beta)$$

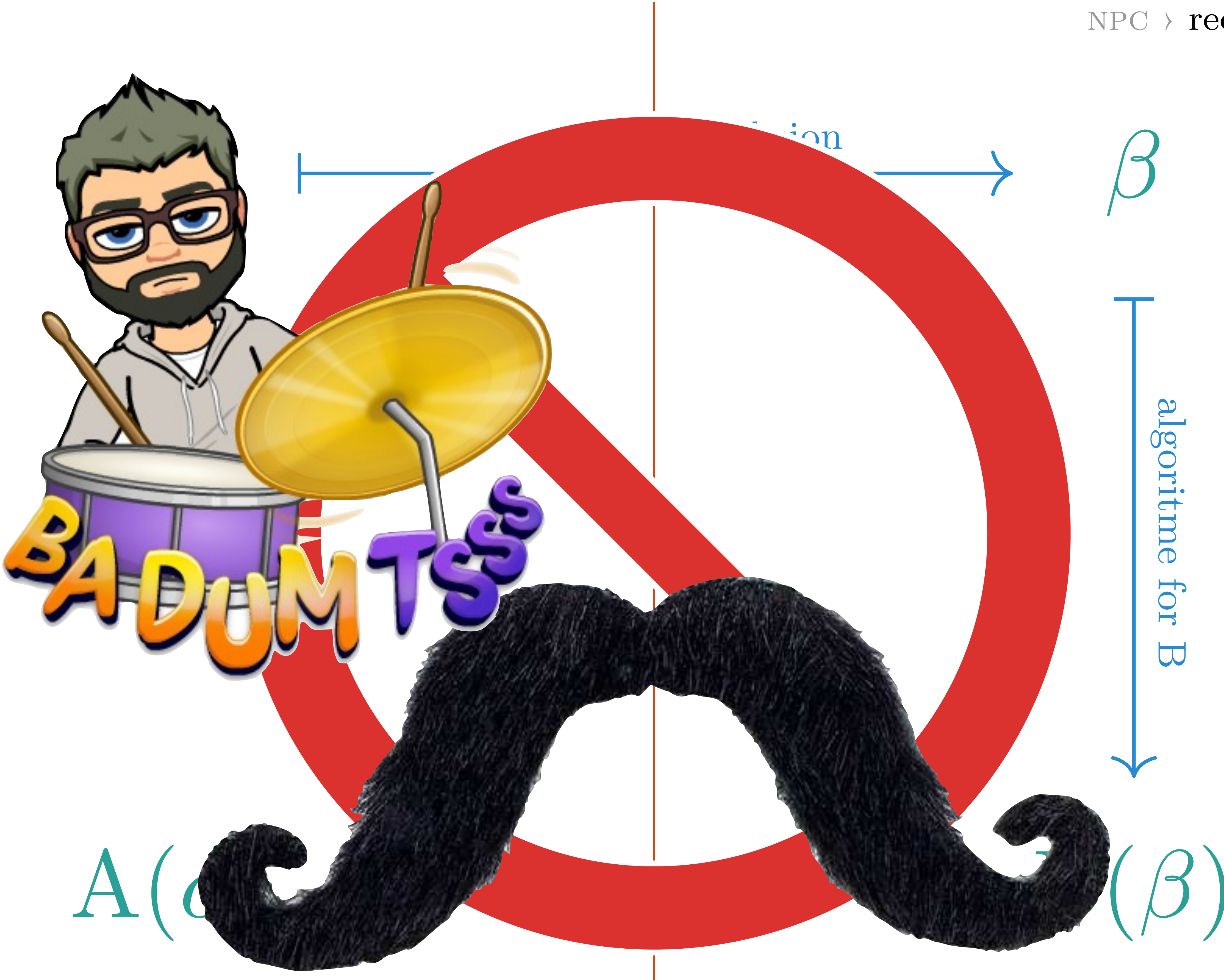
$$
\begin{tikzcd}
\alpha \arrow[r, mapsto, "\text{en reduksjon}"] & \beta \arrow[d, mapsto, "\text{algoritme for B}"] \\
\mathrm{A}(\alpha) \arrow[r, Leftrightarrow] & \mathrm{B}(\beta)
\end{tikzcd}
$$

$$\begin{tikzcd} \alpha \arrow[r, mapsto, "\text{en reduksjon}"] & \beta \arrow[d, mapsto, "\text{algoritme for B}"] \\ \mathrm{A}(\alpha) \arrow[r, Leftrightarrow] & \mathrm{B}(\beta) \end{tikzcd}$$

uløselig



α
en reduksjon
β
algoritme for B
uløselig
?
A(α)
B(β)



Nei! Om A reduserer til et løsbart problem, må A også være løsbart!

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\alpha \arrow[r, mapsto, "\text{en reduksjon}"] \arrow[d, mapsto, "\text{algoritme for A}"'] & \beta \arrow[d, mapsto, "\text{algoritme for B}"] \\
\mathrm{A}(\alpha) \arrow[r, Leftrightarrow] & \mathrm{B}(\beta)
\end{tikzcd}
$$

$$
\begin{tikzcd}
\alpha \arrow[rrr, mapsto, "\text{polynomisk reduksjon}"] \arrow[d, mapsto, "\text{beste mulige}"'] & & & \beta \arrow[d, mapsto, "\text{beste mulige}"] \\
\mathrm{A}(\alpha) \arrow[rrr, Leftrightarrow] & & & \mathrm{B}(\beta)
\end{tikzcd}
$$

$$
\begin{tikzcd}
\alpha \arrow[r, mapsto, "\text{polynomisk reduksjon}"] \arrow[d, mapsto, "\text{beste mulige}"'] & \beta \arrow[d, mapsto, "\text{beste mulige}"] \\
A(\alpha) \arrow[r, Leftrightarrow] & B(\beta)
\end{tikzcd}
$$

Hvis denne er polynomisk…

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\alpha \arrow[r, mapsto, "\text{polynomisk reduksjon}"] \arrow[d, mapsto, "\text{beste mulige}"'] & \beta \arrow[d, mapsto, "\text{beste mulige}"] \\
A(\alpha) \arrow[r, Leftrightarrow] & B(\beta)
\end{tikzcd}
$$

Hvis denne er polynomisk…

…så må denne også være det

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\alpha \arrow[r, mapsto, "\text{polynomisk reduksjon}"] \arrow[d, mapsto, "\text{beste mulige}"'] & \beta \arrow[d, mapsto, "\text{beste mulige}"] \\
\mathrm{A}(\alpha) \arrow[r, Leftrightarrow] & \mathrm{B}(\beta)
\end{tikzcd}$$

Hvis denne er eksponentiell . . .

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\alpha \arrow[r, mapsto, "\text{polynomisk reduksjon}"] \arrow[d, mapsto, "\text{beste mulige}"'] & \beta \arrow[d, mapsto, "\text{beste mulige}"] \\
A(\alpha) \arrow[r, Leftrightarrow] & B(\beta)
\end{tikzcd}$$

Hvis denne er eksponentiell…

…så må denne også være det

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\alpha \arrow[r, mapsto, "\text{«ubetydelig» reduksjon}"] \arrow[d, mapsto, "\text{beste mulige}"'] \arrow[dr, phantom, "\leqslant"] & \beta \arrow[d, mapsto, "\text{beste mulige}"] \\
\mathrm{A}(\alpha) \arrow[r, Leftrightarrow] & \mathrm{B}(\beta)
\end{tikzcd}$$

- **Redusibilitet:**
  Hvis A kan reduseres til B i polynomisk tid, skriver vi $A \leq_{\mathbf{P}} B$

- **Redusibilitet:**
  Hvis A kan reduseres til B i polynomisk tid, skriver vi $\mathrm{A} \leq_{\mathbf{P}} \mathrm{B}$
- **Ordning:**
  Relasjonen $\leq_{\mathbf{P}}$ utgjør en *preordning*

- **Redusibilitet:**
  Hvis A kan reduseres til B i polynomisk tid, skriver vi $A \leq_{\mathbf{P}} B$
- **Ordning:**
  Relasjonen $\leq_{\mathbf{P}}$ utgjør en *preordning*

Delvis ordning uten antisymmetri, dvs., vi tillater sykler

- **Redusibilitet:**
  Hvis A kan reduseres til B i polynomisk tid, skriver vi $A \leq_{\mathbf{P}} B$
- **Ordning:**
  Relasjonen $\leq_{\mathbf{P}}$ utgjør en *preordning*
- **Hardhetsbevis:**
  For å vise at B er vanskelig, redusér fra et vanskelig problem A, dvs., etablér at $A \leq_{\mathbf{P}} B$

Delvis ordning uten antisymmetri, dvs., vi tillater sykler

Et eksempel

› **CLIQUE**

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT
- Lag én node i G for hver literal i formelen

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT
- Lag én node i G for hver literal i formelen
- Ingen kanter mellom noder fra samme term

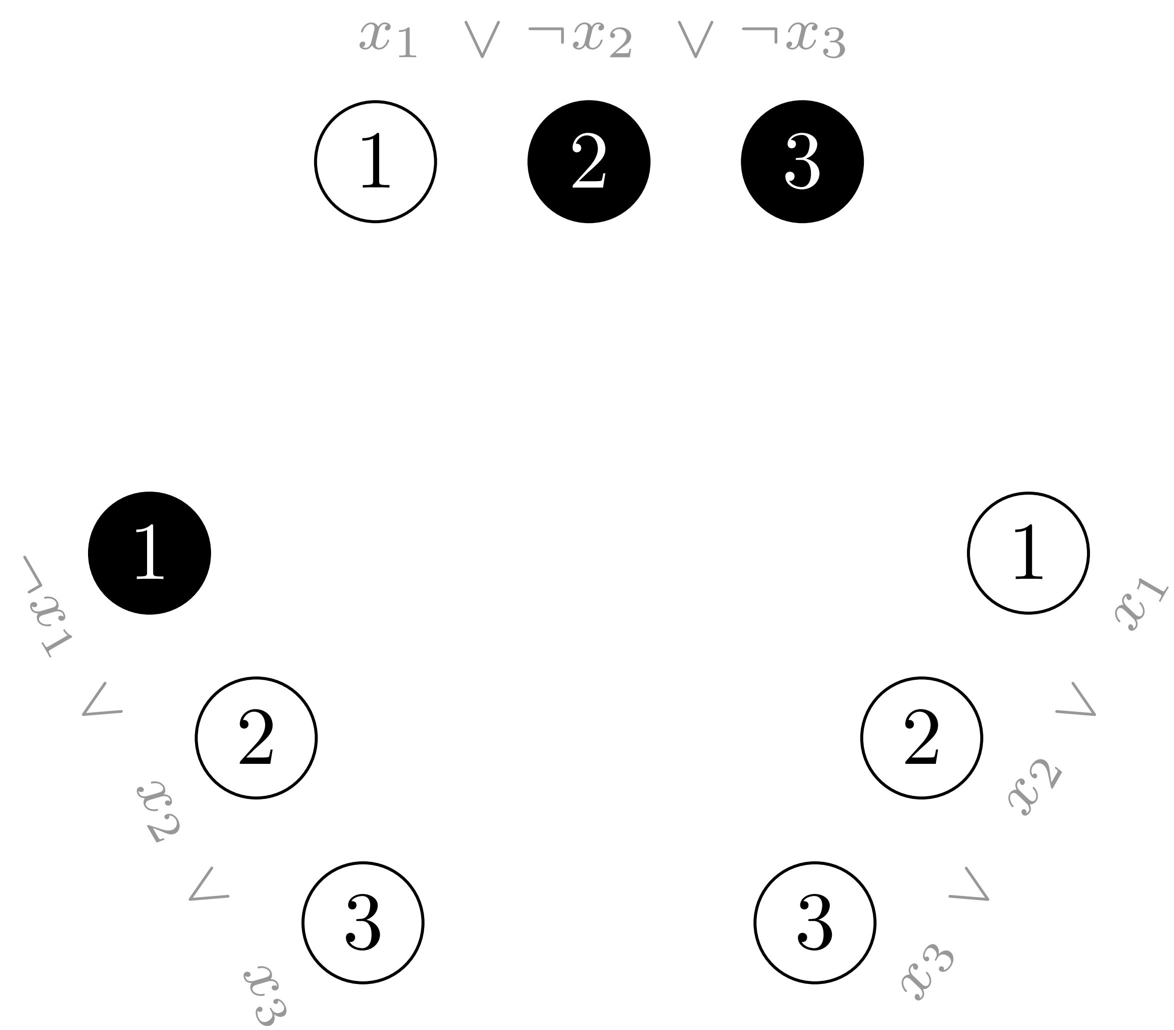
- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT
- Lag én node i G for hver literal i formelen
- Ingen kanter mellom noder fra samme term
- Ellers: Kanter mellom literaler som kan være sanne samtidig

- **CLIQUE**
  - **Instans:** En urettet graf G og et heltall k
  - **Spørsmål:** Har G en en komplett delgraf med k noder?
- Vi vil redusere fra 3-CNF-SAT
- Lag én node i G for hver literal i formelen
- Ingen kanter mellom noder fra samme term
- Ellers: Kanter mellom literaler som kan være sanne samtidig
- La k være antall termer

$$\mapsto$$

$$\begin{aligned}\phi = &(\; x_1 \vee \neg x_2 \vee \neg x_3) \wedge \\ &(\neg x_1 \vee \; x_2 \vee \; x_3) \wedge \\ &(\; x_1 \vee \; x_2 \vee \; x_3)\end{aligned}$$

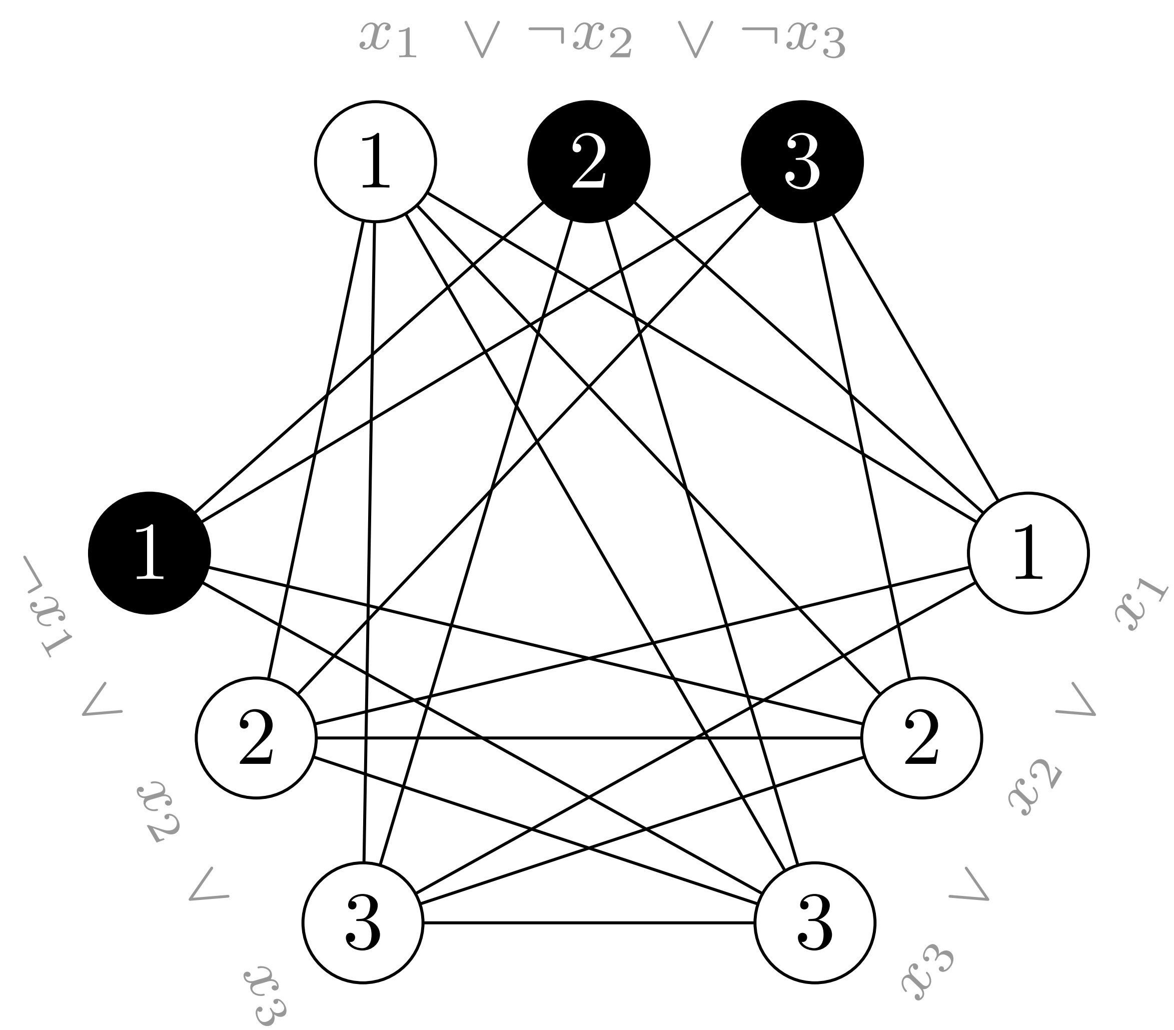
Kan ϕ være sann? $\iff$ Finnes en k-klikk?



$$\begin{aligned}\phi = &(\ \ x_1 \vee \neg x_2 \vee \neg x_3) \wedge\\ &(\neg x_1 \vee \ \ x_2 \vee \ \ x_3) \wedge\\ &(\ \ x_1 \vee \ \ x_2 \vee \ \ x_3)\end{aligned}$$

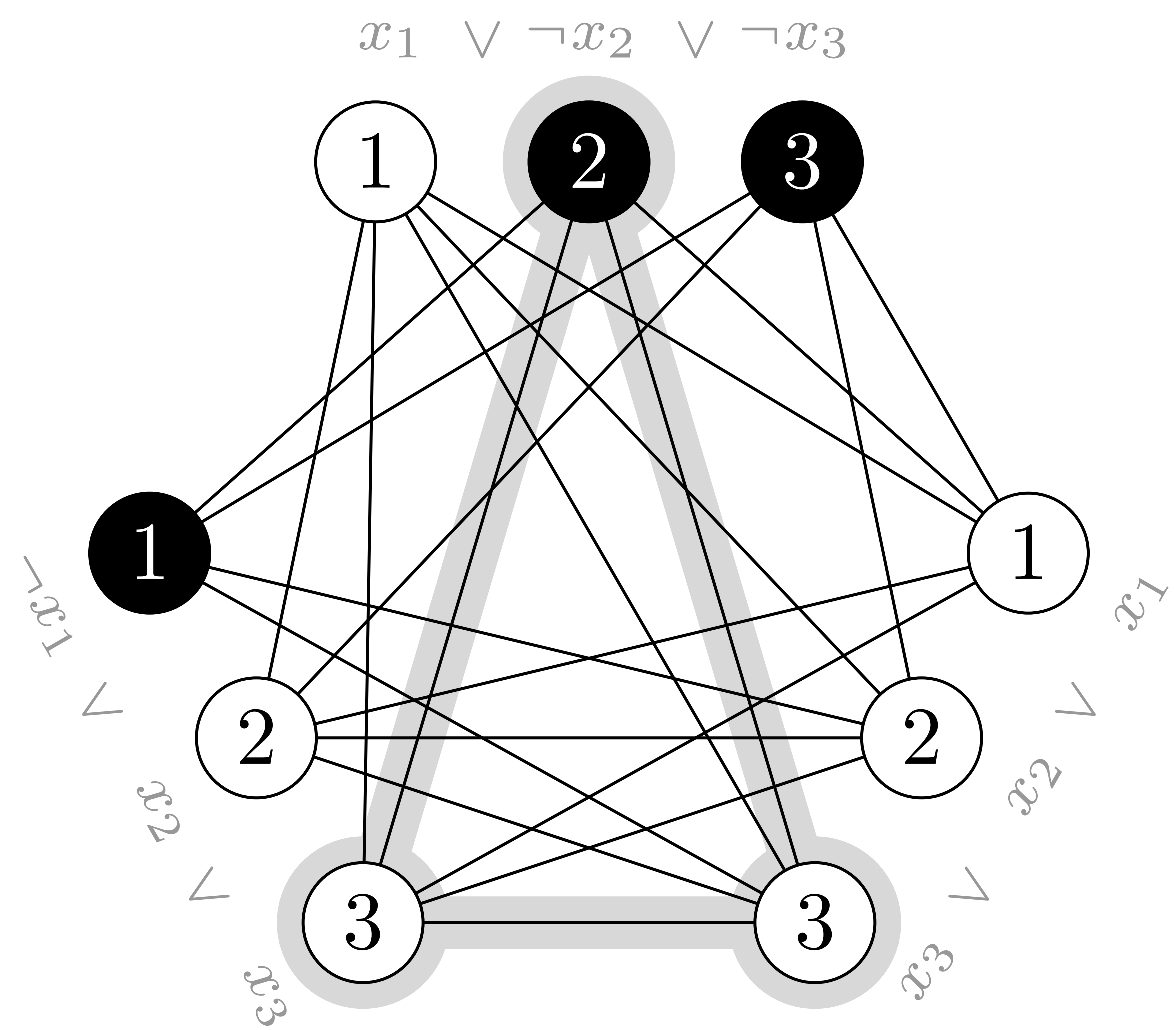
Kan ϕ være sann? $\Longleftrightarrow$ Finnes en k-klikk?

$$
\begin{aligned}
\phi = {} & (\ \ x_1 \vee \neg x_2 \vee \neg x_3\,) \wedge \\
& (\neg x_1 \vee \ \ x_2 \vee \ \ x_3\,) \wedge \\
& (\ \ x_1 \vee \ \ x_2 \vee \ \ x_3\,)
\end{aligned}
$$



Kan ϕ være sann? $\Longleftrightarrow$ Finnes en k-klikk?

$$\begin{aligned}\phi = (\ \ x_1 \vee \neg x_2 \vee \neg x_3\,) \wedge \\ (\neg x_1 \vee\ \ x_2 \vee\ \ x_3\,) \wedge \\ (\ \ x_1 \vee\ \ x_2 \vee\ \ x_3\,)\end{aligned}$$

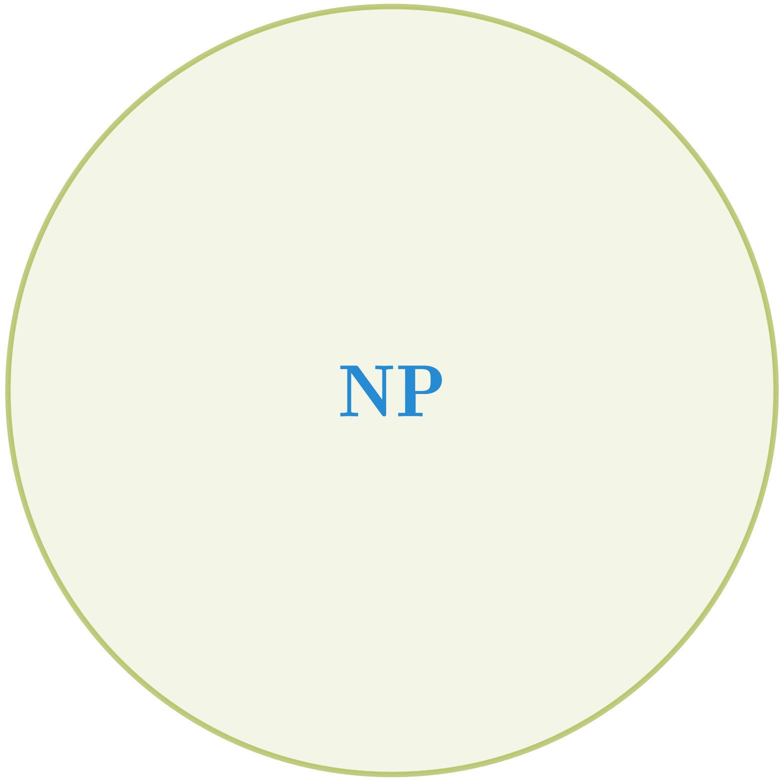


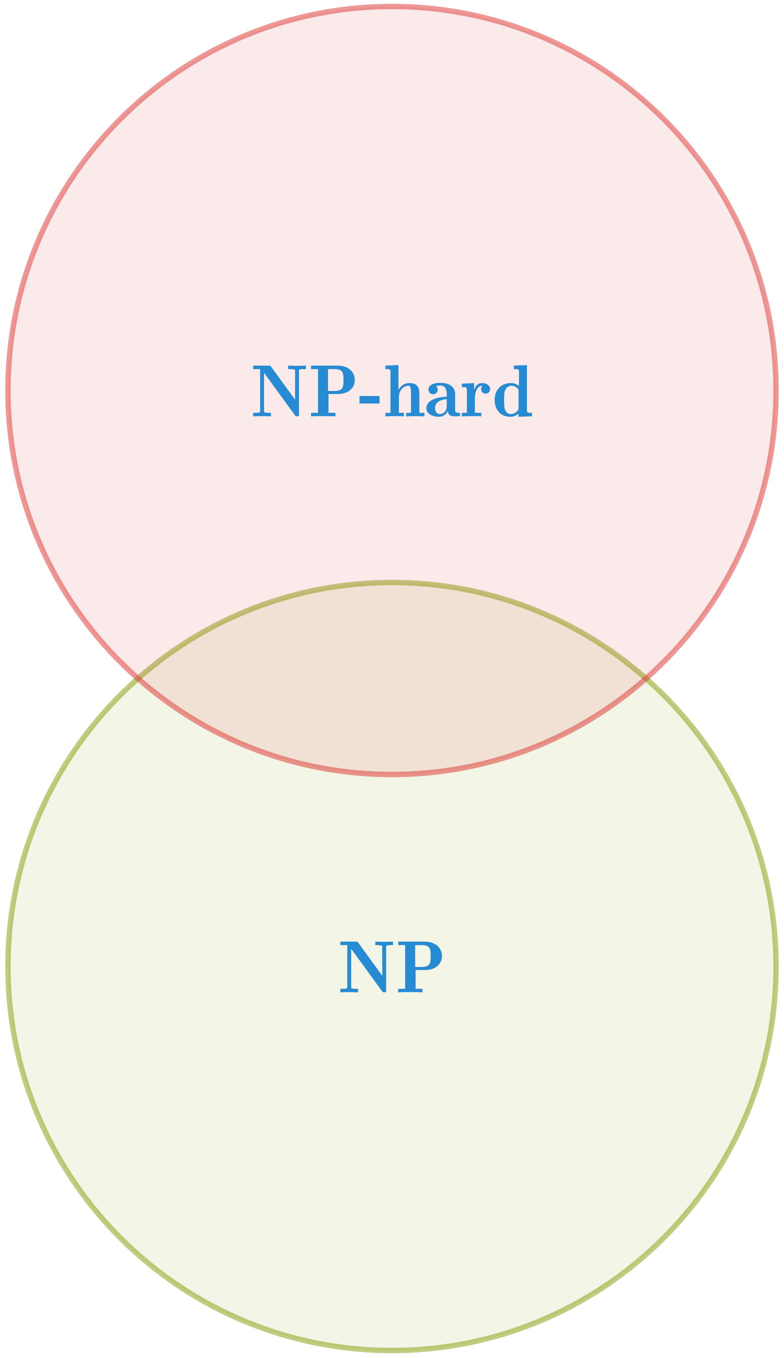
Tilsv. $x_1, x_2, x_3 = -, 0, 1$

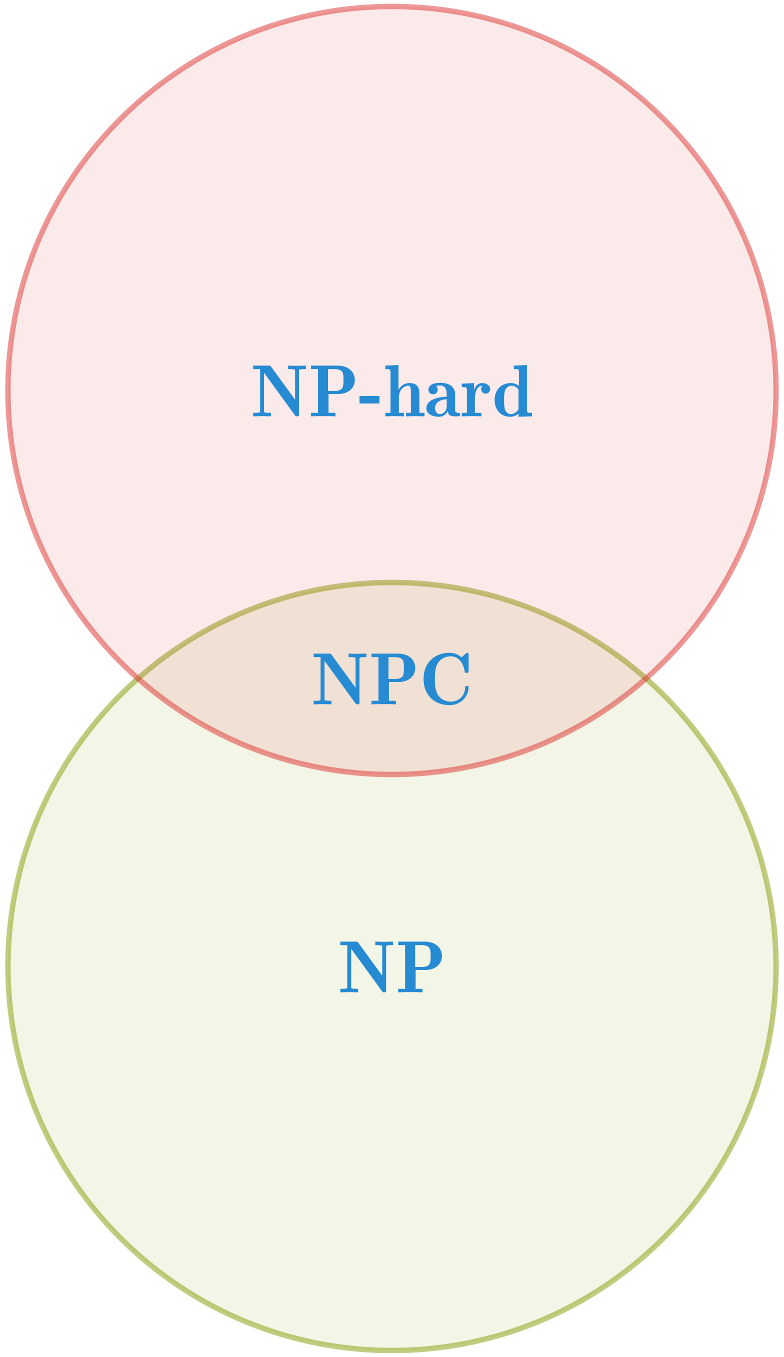
Kan ϕ være sann? $\Longleftrightarrow$ Finnes en k-klikk?

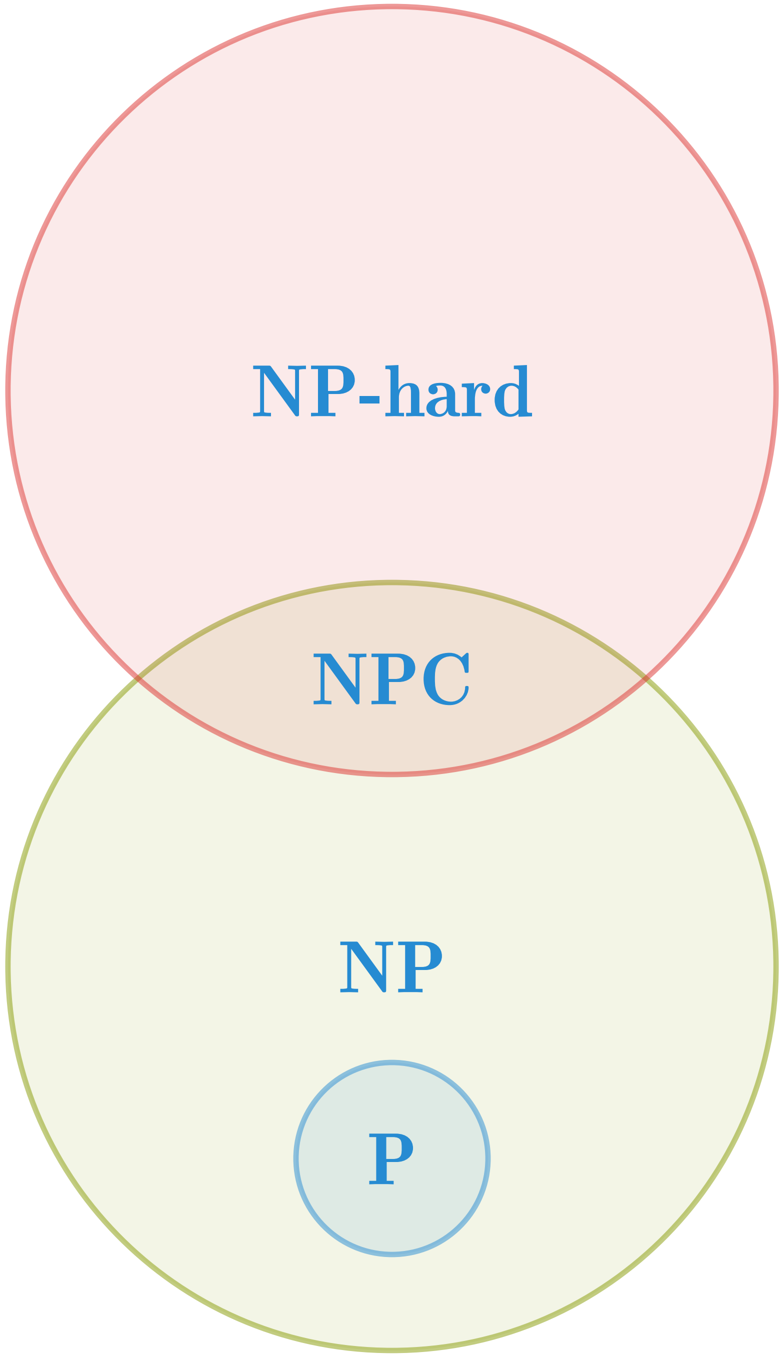
3:4

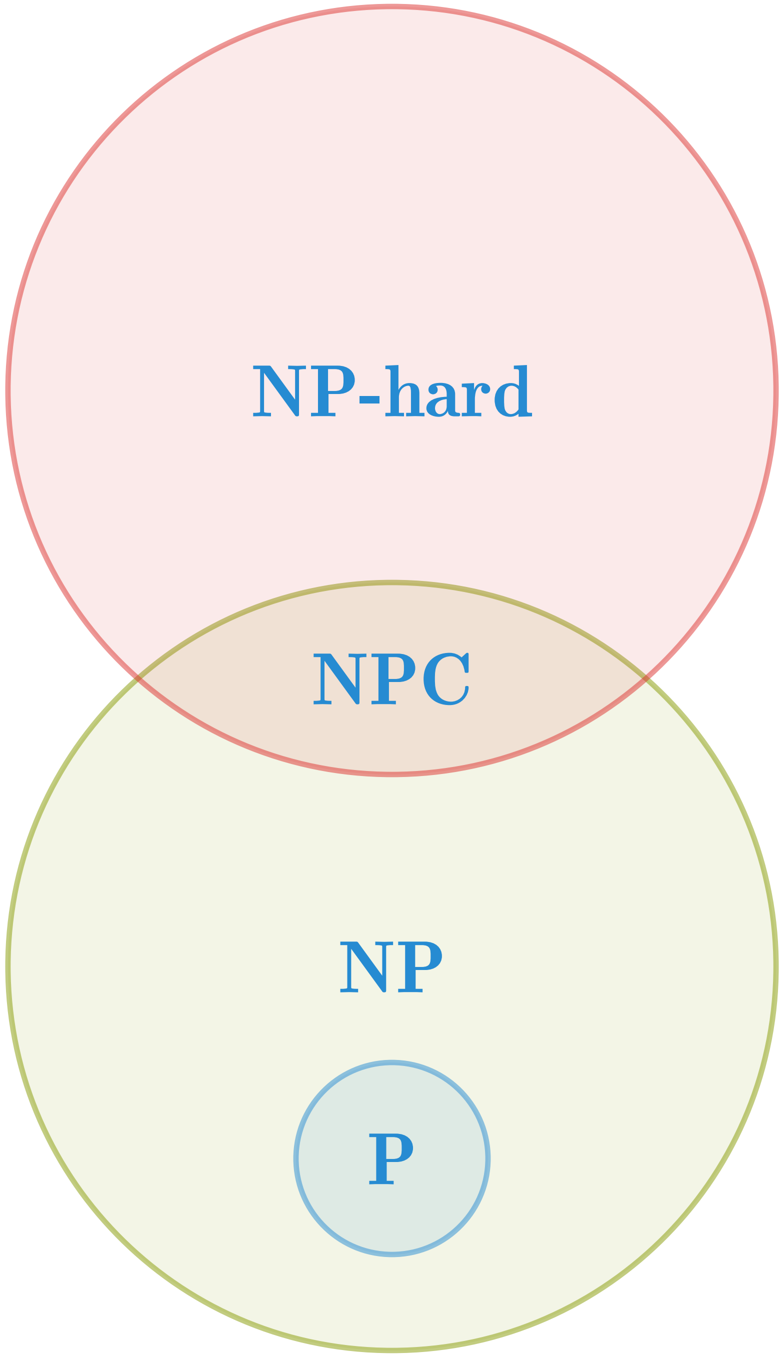
Kompletthet


NP

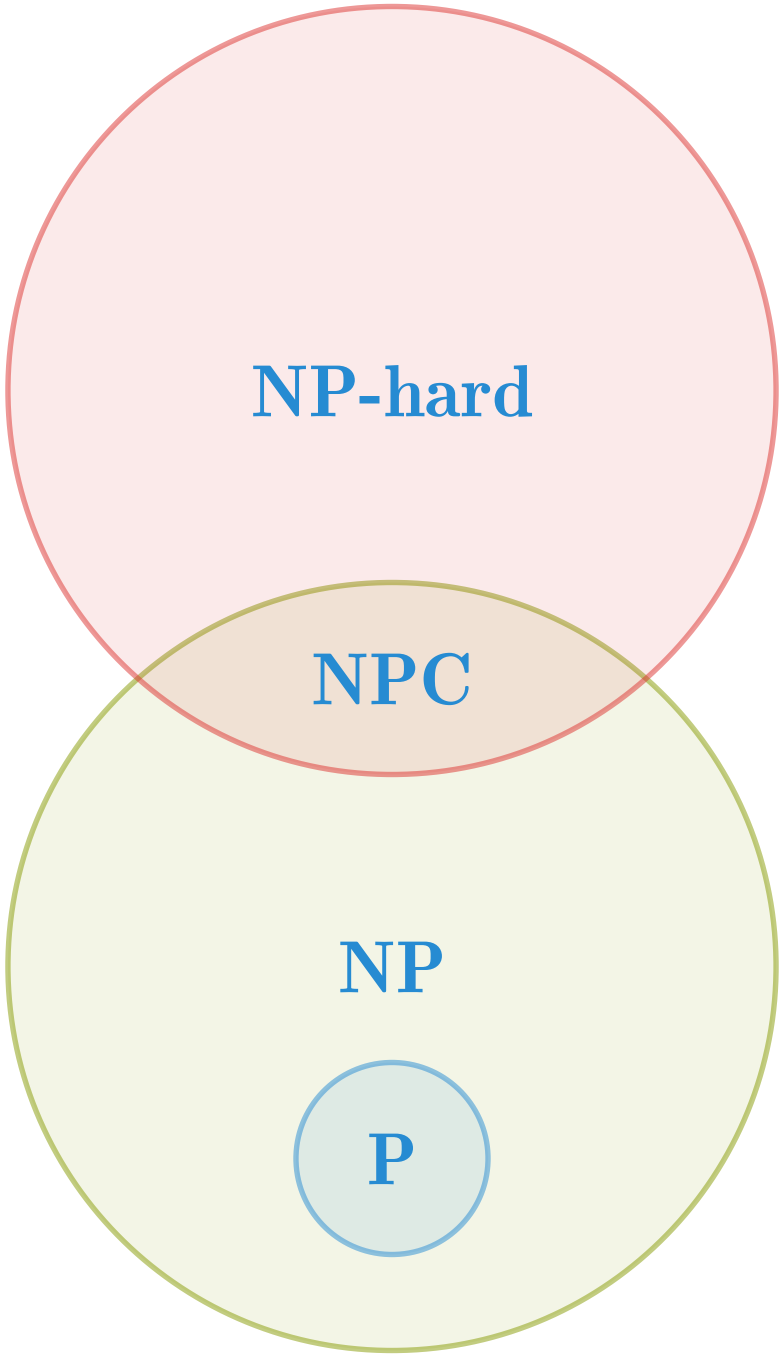

NP-hard
NP


NP-hard
NPC
NP


NP-hard
NPC
NP
P



Det er altså sånn de fleste *tror* ting er



(Og det finnes *mange* andre klasser . . .)

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.

› **Kompletthet:**
Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.

De komplette problemene er altså de vanskeligste i klassen

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.
- **Maksimalitet:**
  Et element er *maksimalt* dersom alle andre er mindre eller lik. For reduksjoner: Q er maksimalt dersom alle problemer i klassen kan reduseres til Q.

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.
- **Maksimalitet:**
  Et element er *maksimalt* dersom alle andre er mindre eller lik. For reduksjoner: Q er maksimalt dersom alle problemer i klassen kan reduseres til Q.
- **NPC:**
  De komplette språkene i **NP**, under polynomiske reduksjoner.

- **Kompletthet:**
  Et problem er *komplett* for en gitt klasse og en gitt type reduksjoner dersom det er *maksimalt* for redusibilitetsrelasjonen.
- **Maksimalitet:**
  Et element er *maksimalt* dersom alle andre er mindre eller lik. For reduksjoner: Q er maksimalt dersom alle problemer i klassen kan reduseres til Q.
- **NPC:**
  De komplette språkene i **NP**, under polynomiske reduksjoner.

Altså de vanskeligste problemene i **NP**

De vanskelige problemene fra tidligere er altså eksempler på NP-komplette problemer

- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.

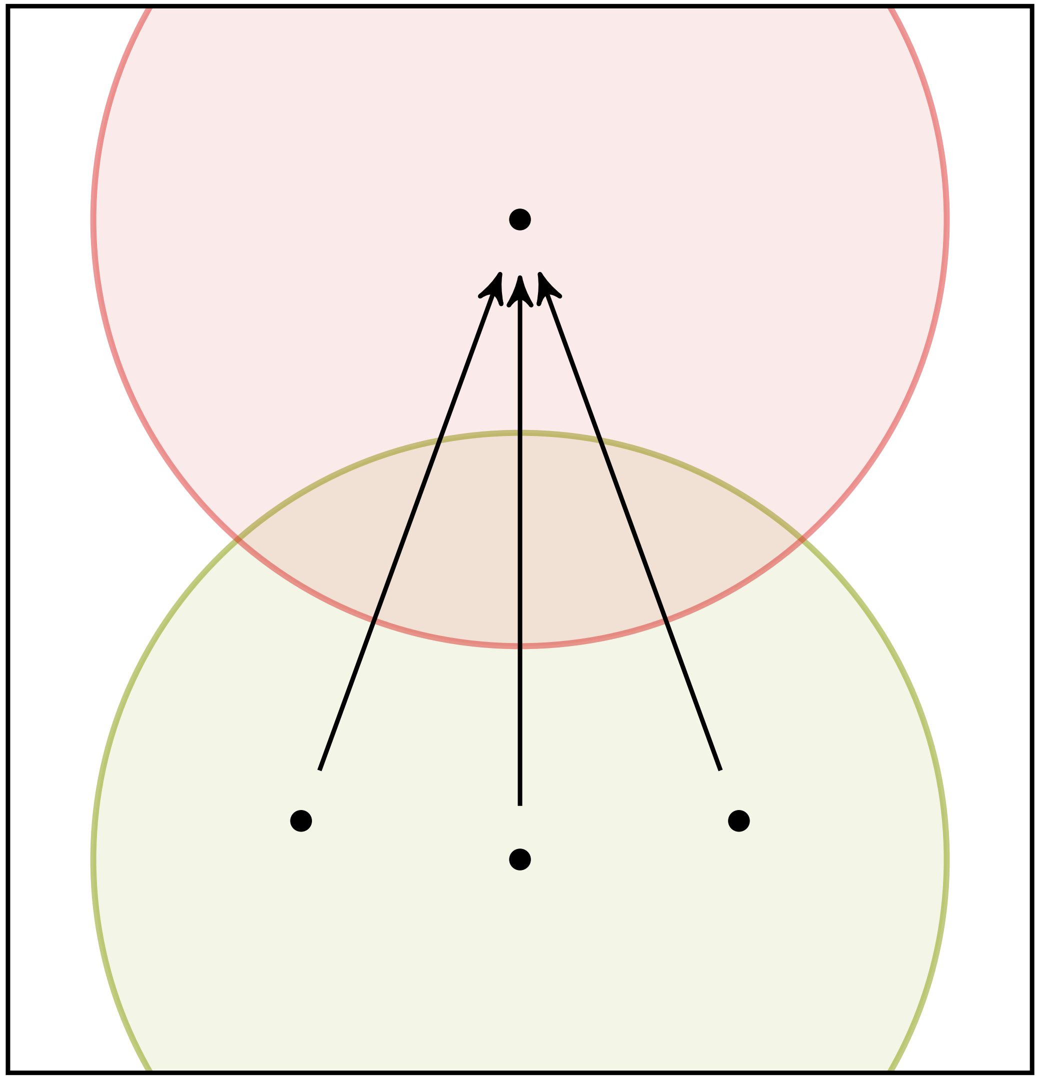
› **NP-hardhet:**
Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.

Vi kaller gjerne klassen **NP-hard**

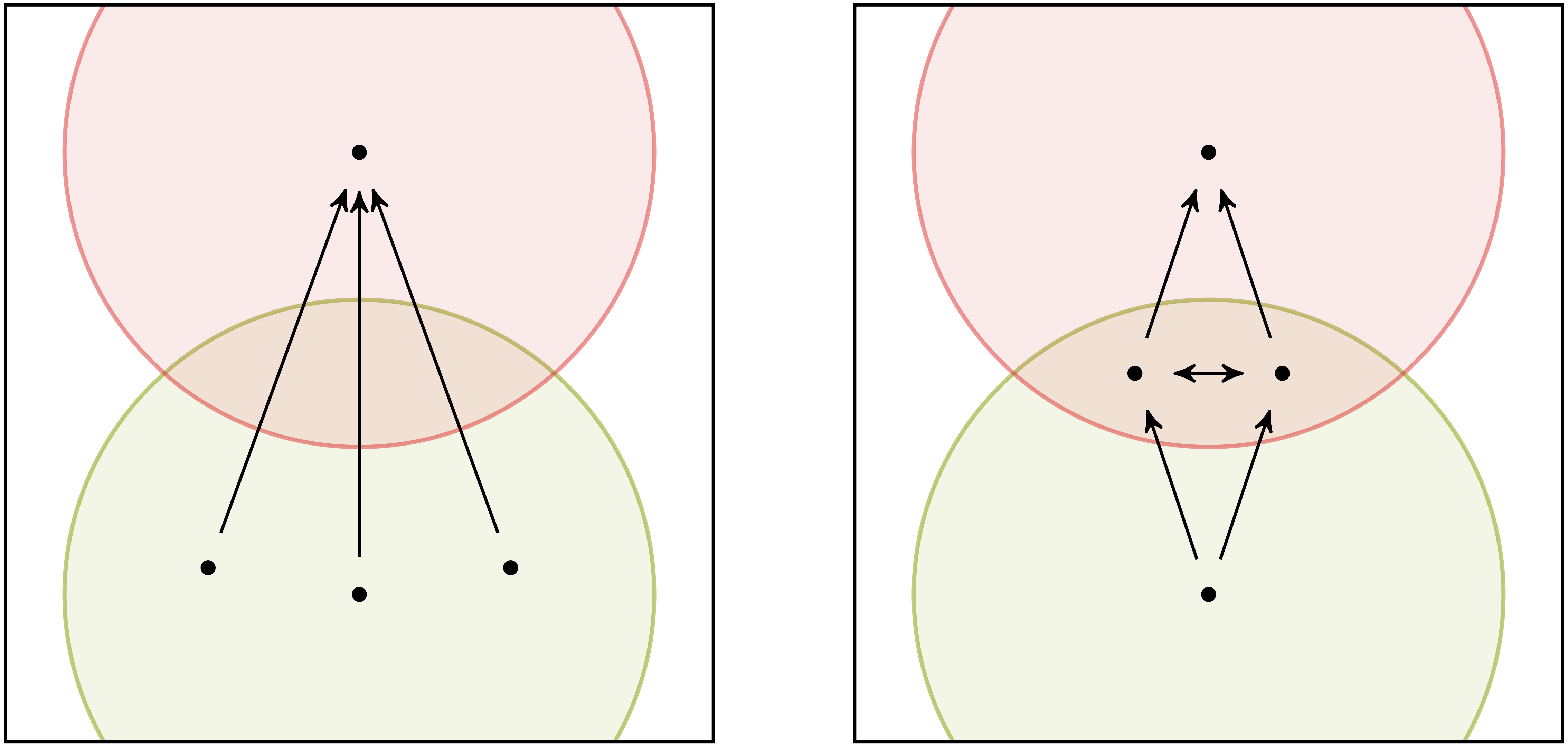
- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.
- Et problem er altså **NP**-komplett dersom det

- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.
- Et problem er altså **NP**-komplett dersom det
  - er **NP**-hardt, og

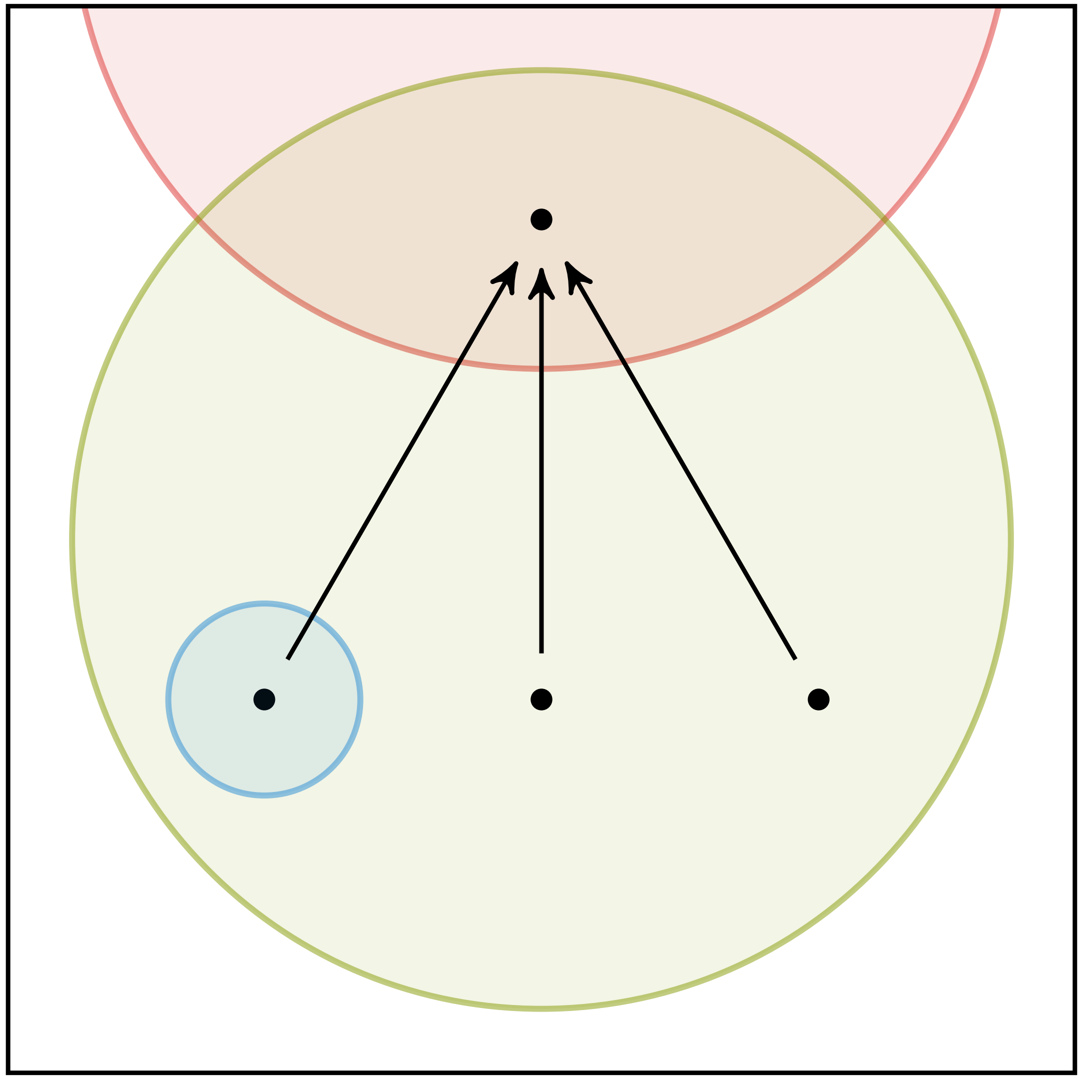
- **NP-hardhet:**
  Et problem Q er **NP**-hardt dersom alle problemer i NP kan reduseres til det.
- Et problem er altså **NP**-komplett dersom det
  - er **NP**-hardt, og
  - er i **NP**.



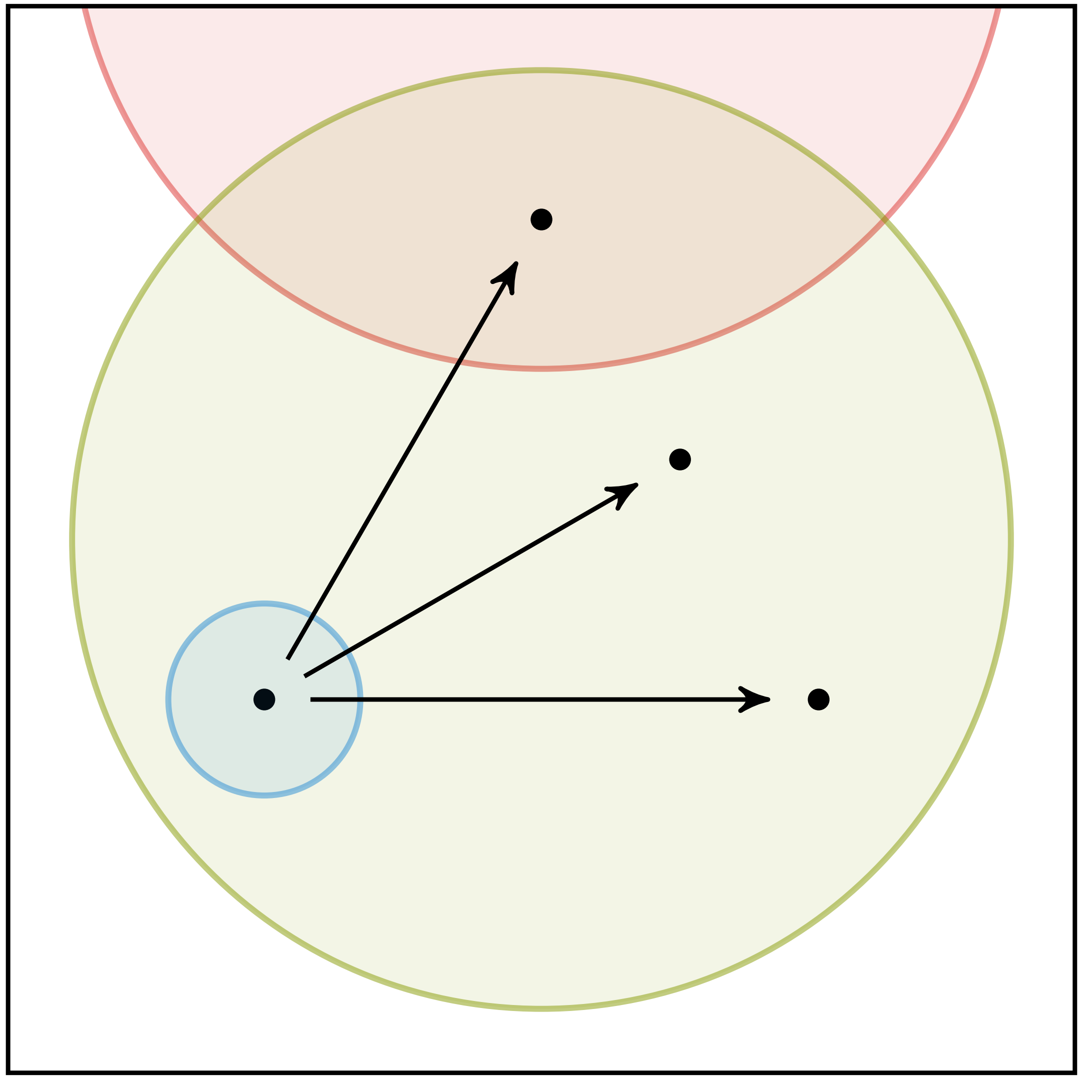
Alt i **NP** kan reduseres til alt i **NPH**



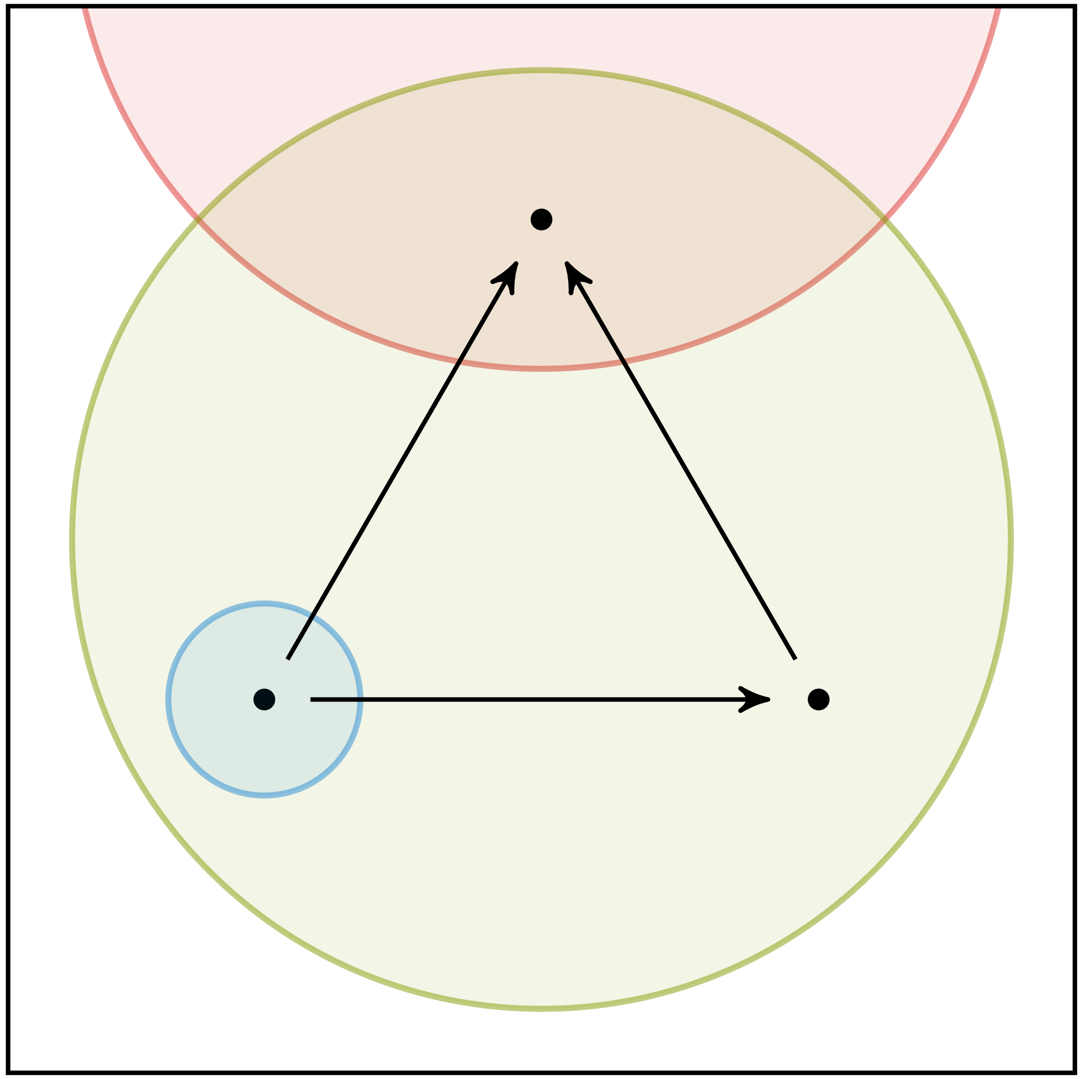
Dermed kan problemer i **NPC** reduseres til hverandre



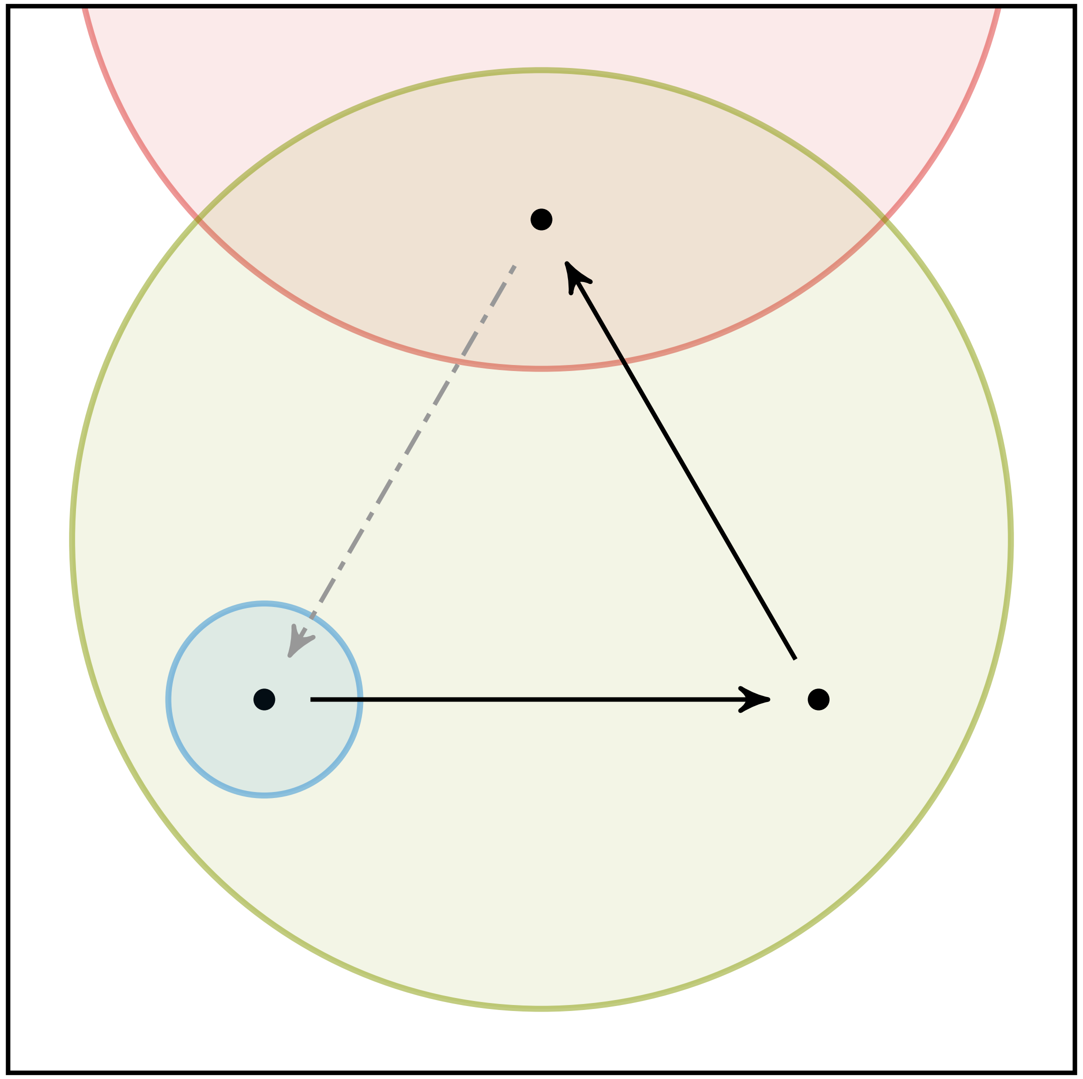
Alt i **NP** kan (per def.) reduseres til alt i **NPH**, og dermed **NPC**



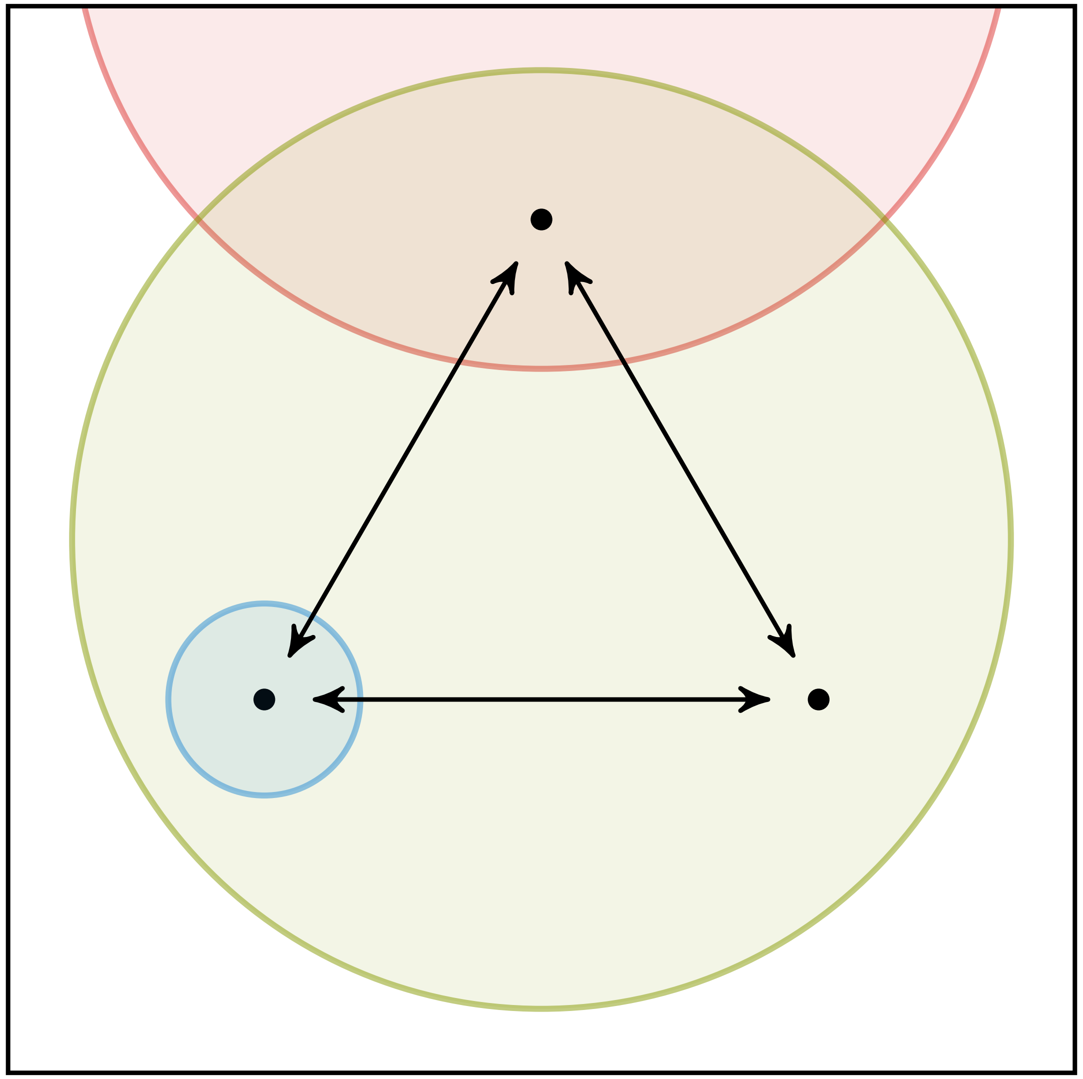
Problemer i **P** kan (trivielt, ved å løses) reduseres til alt i **NP**



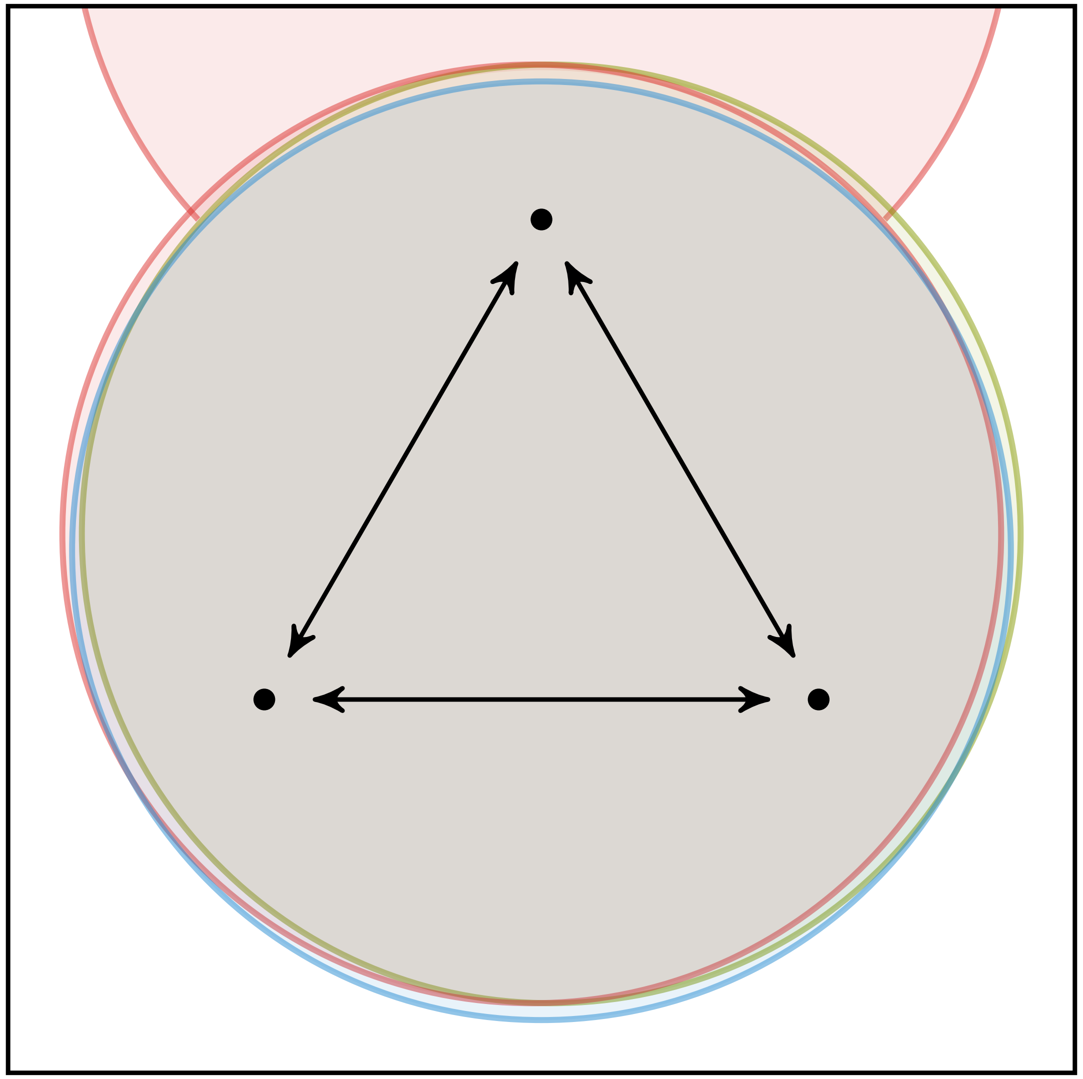
Dette er slik vi tror verden ser ut



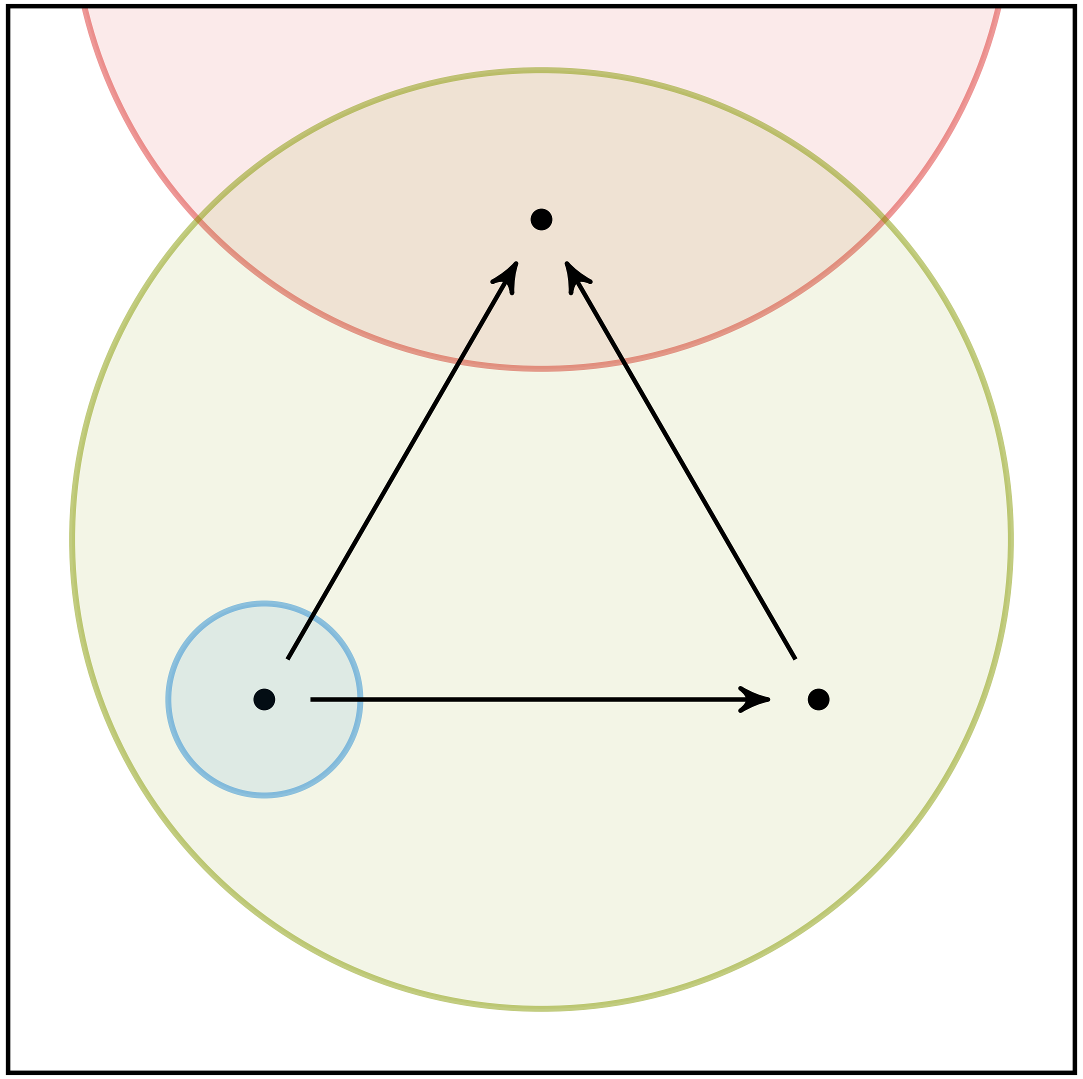
Hva om vi finner en reduksjon fra **NPC** til **P**?



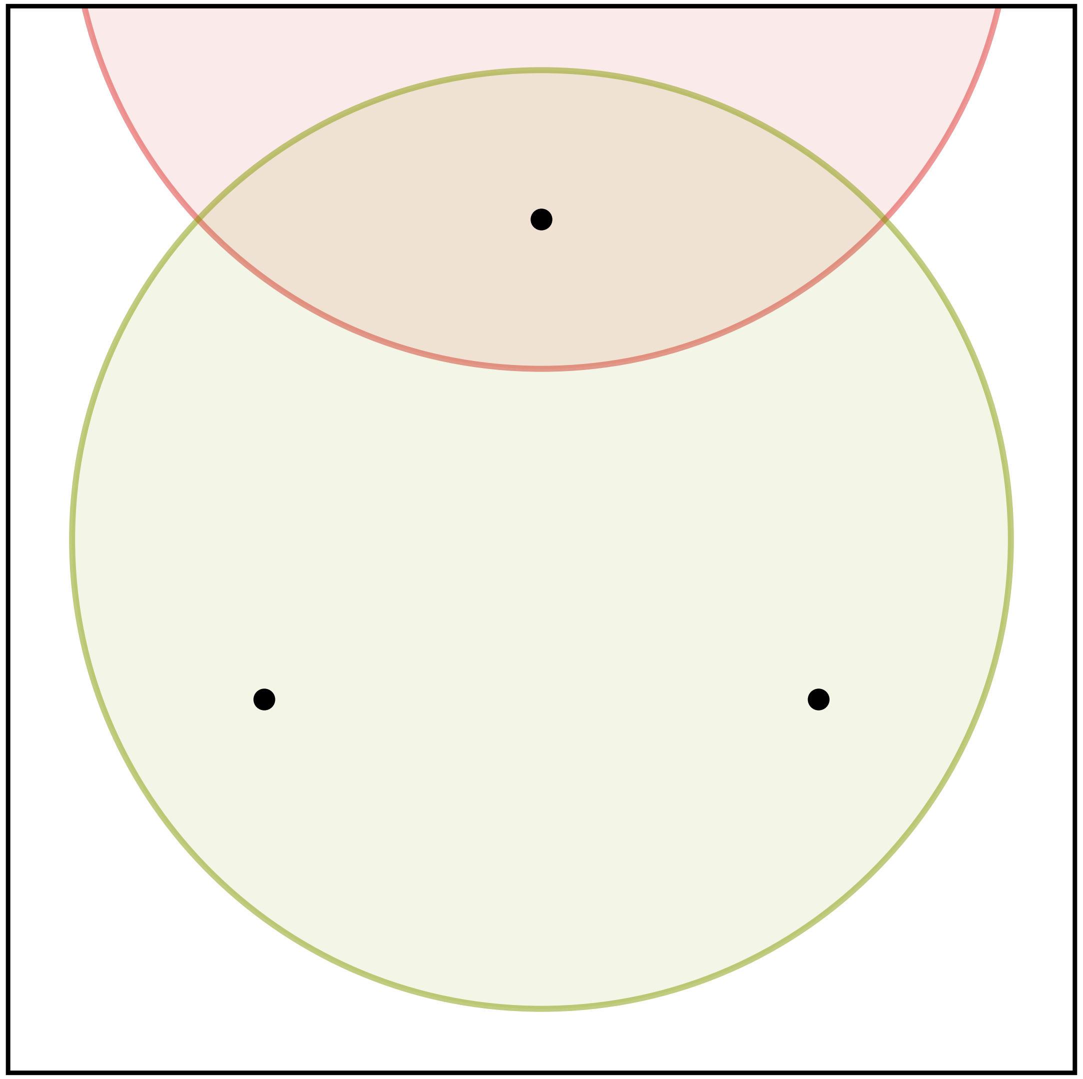
Da kan vi redusere alt i **NP** til alt annet i **NP** . . .



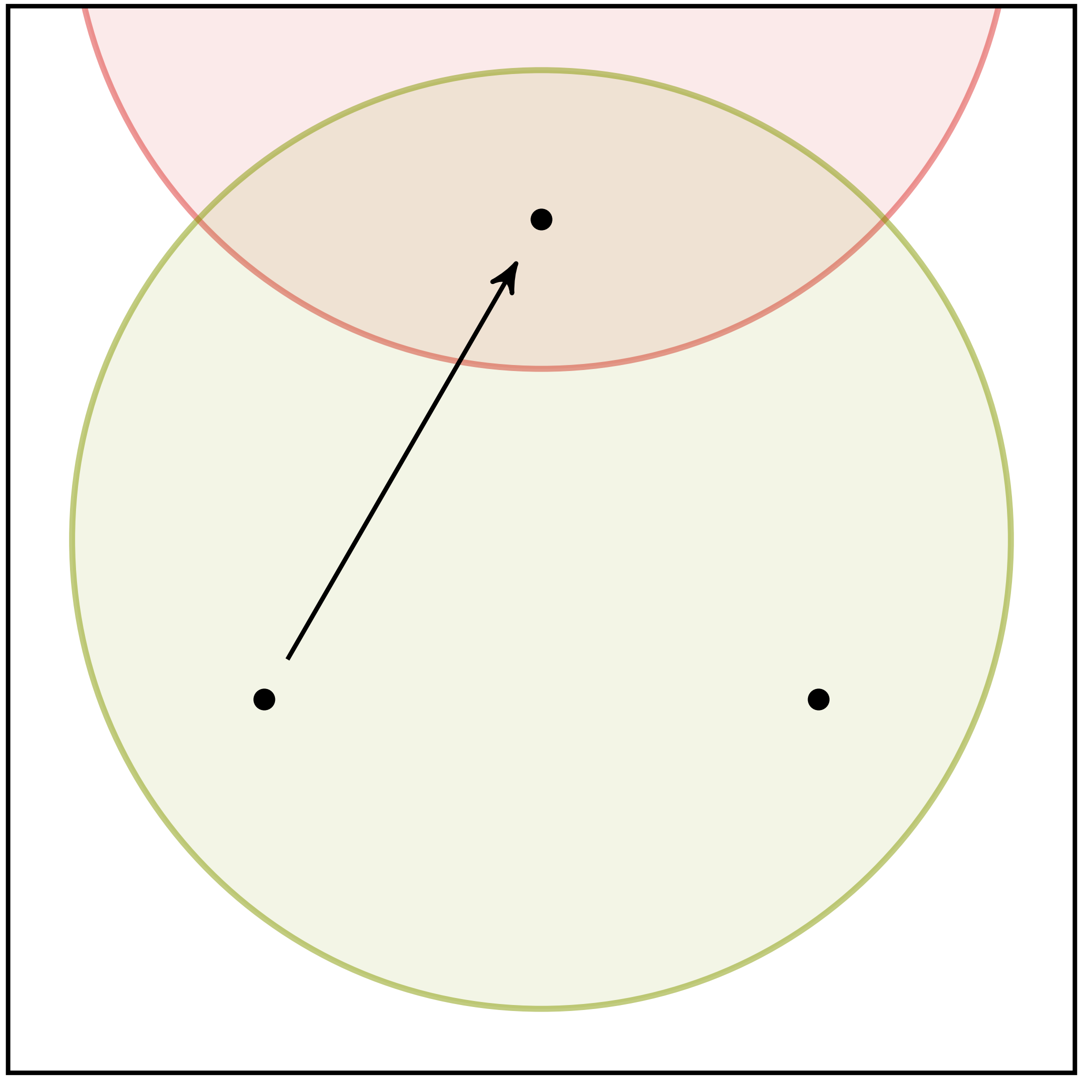
…som betyr at **P**, **NP** og **NPC** er samme klasse!



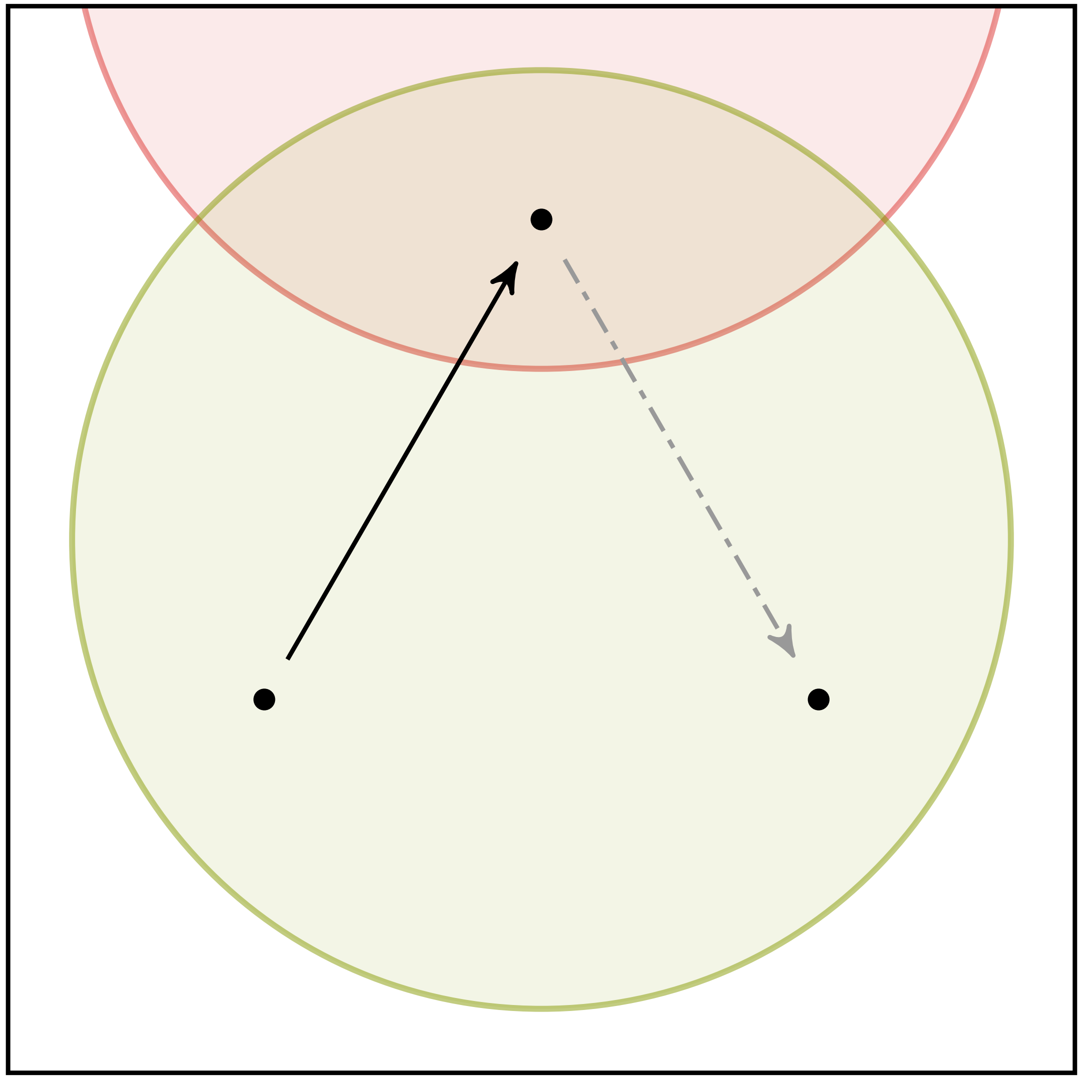
De fleste av oss tror ikke at $\mathbf{P} = \mathbf{NP}$



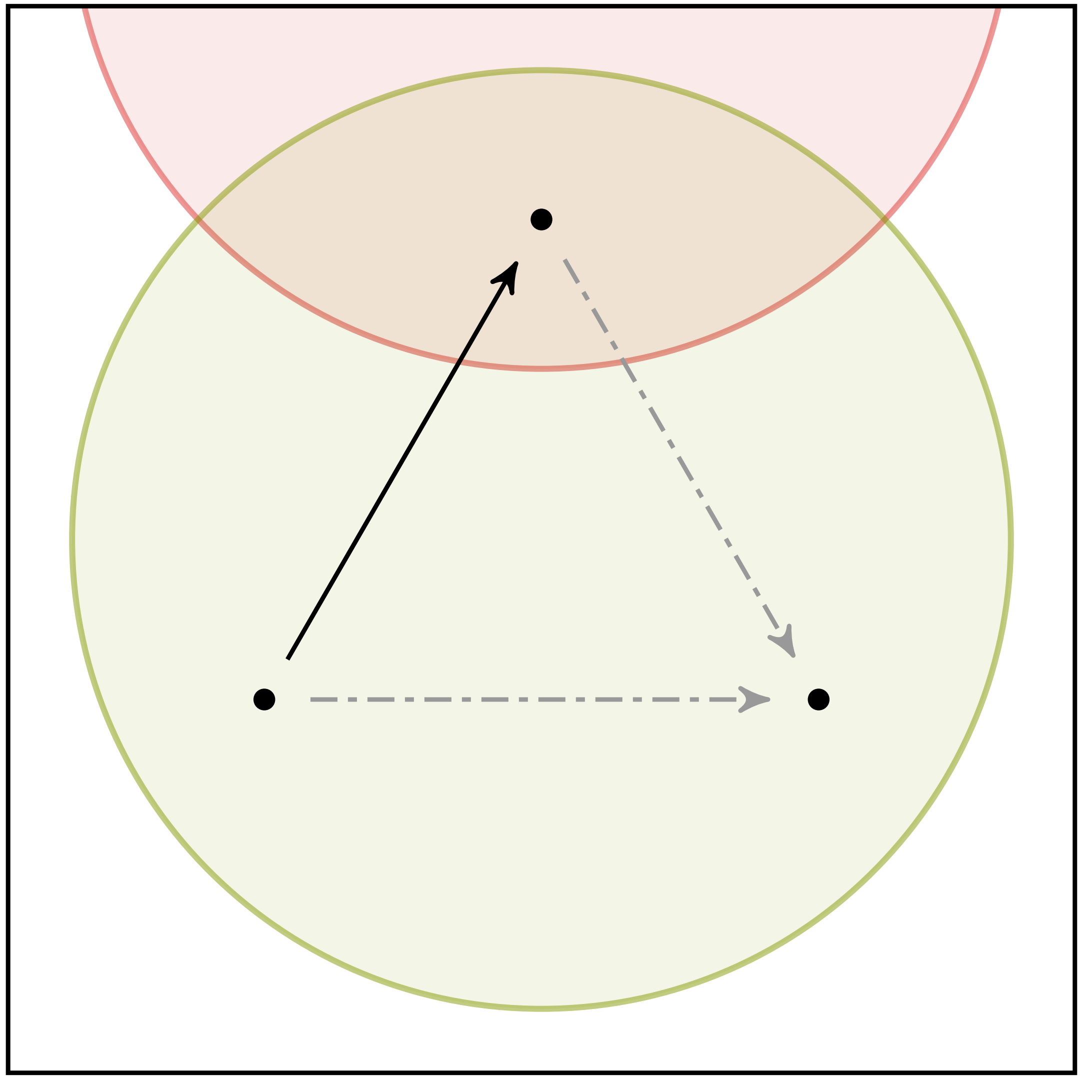
Vi kan bruke reduksjoner til å karakterisere problemer



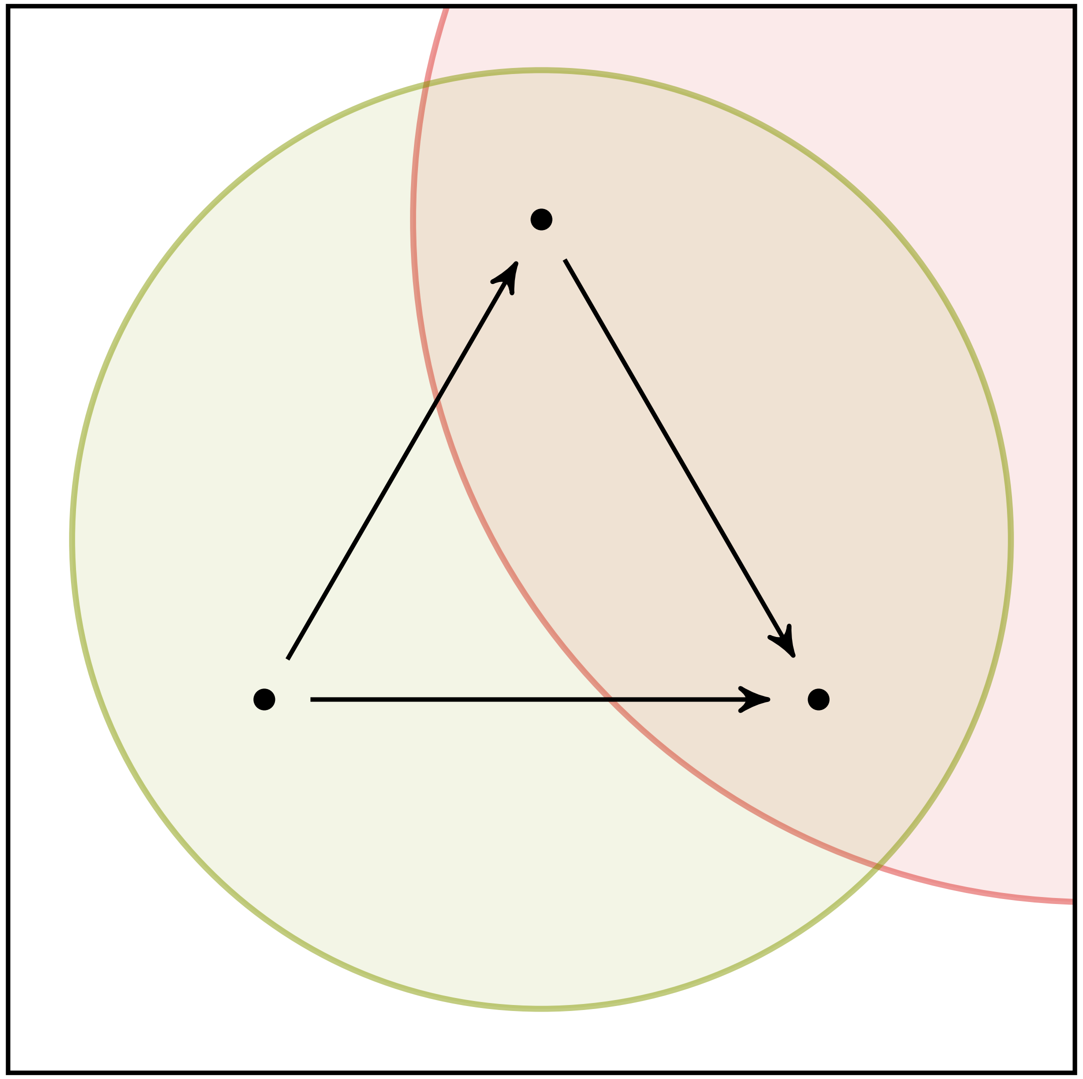
Om vi reduserer til **NPC** forteller det ingenting



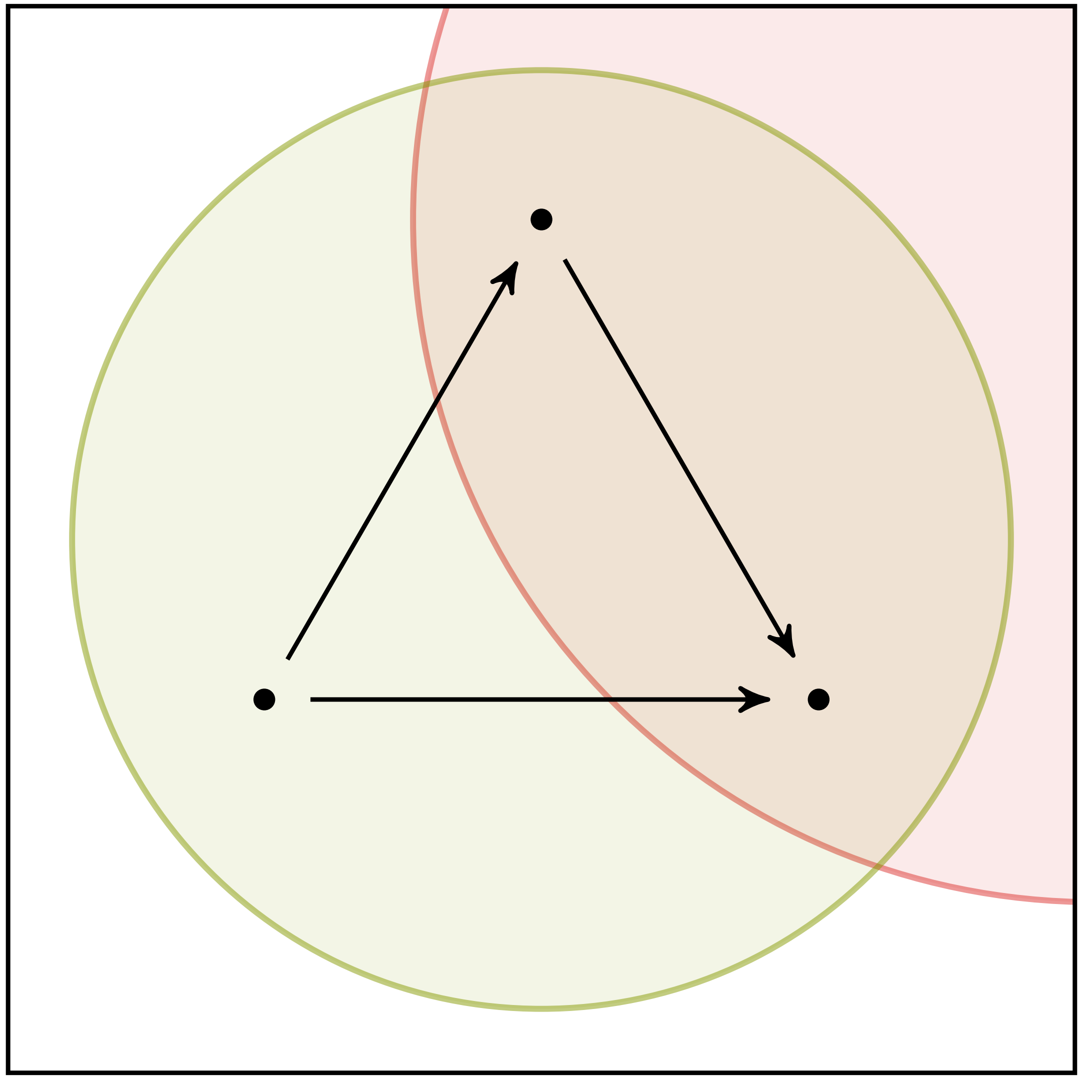
Hva med en reduksjon fra **NPC** til **NP**?



Alt i **NP** kan da reduseres til problemet vårt



Det er jo definisjonen på et **NP**-komplett problem!



Altså: For å vise at et problem er i **NPC** må vi redusere <u>fra</u> **NPC**

$$L \in \mathbf{NPC}$$

Hvordan viser vi at L er **NP**-komplett?

- Vis at $\mathrm{L} \in \mathbf{NP}$

At sertifikat for ja-svar kan verifiseres i pol. tid

- Vis at L $\in$ **NP**
- Velg et kjent **NP**-komplett språk L'

- Vis at $\mathrm{L} \in \mathbf{NP}$
- Velg et kjent **NP**-komplett språk L'
- Beskriv en algoritme som beregner en funksjon

$$f : \{0,1\}^* \to \{0,1\}^*$$

som mapper instanser av L' til instanser av L

Dette er altså reduksjonen fra L' til L, som viser $\mathrm{L}' \leq_{\mathbf{P}} \mathrm{L}$

- Vis at $\mathrm{L} \in \mathbf{NP}$
- Velg et kjent **NP**-komplett språk L'
- Beskriv en algoritme som beregner en funksjon

$$f : \{0,1\}^* \to \{0,1\}^*$$

som mapper instanser av L' til instanser av L

- Vis at

$$x \in \mathrm{L}' \iff f(x) \in \mathrm{L}\,,$$

for alle $x \in \{0,1\}^*$

Vi må sørge for at vi får samme svar for $f(x)$

- Vis at $\mathrm{L} \in \mathbf{NP}$
- Velg et kjent **NP**-komplett språk L'
- Beskriv en algoritme som beregner en funksjon

$$f : \{0,1\}^* \to \{0,1\}^*$$

som mapper instanser av L' til instanser av L
- Vis at

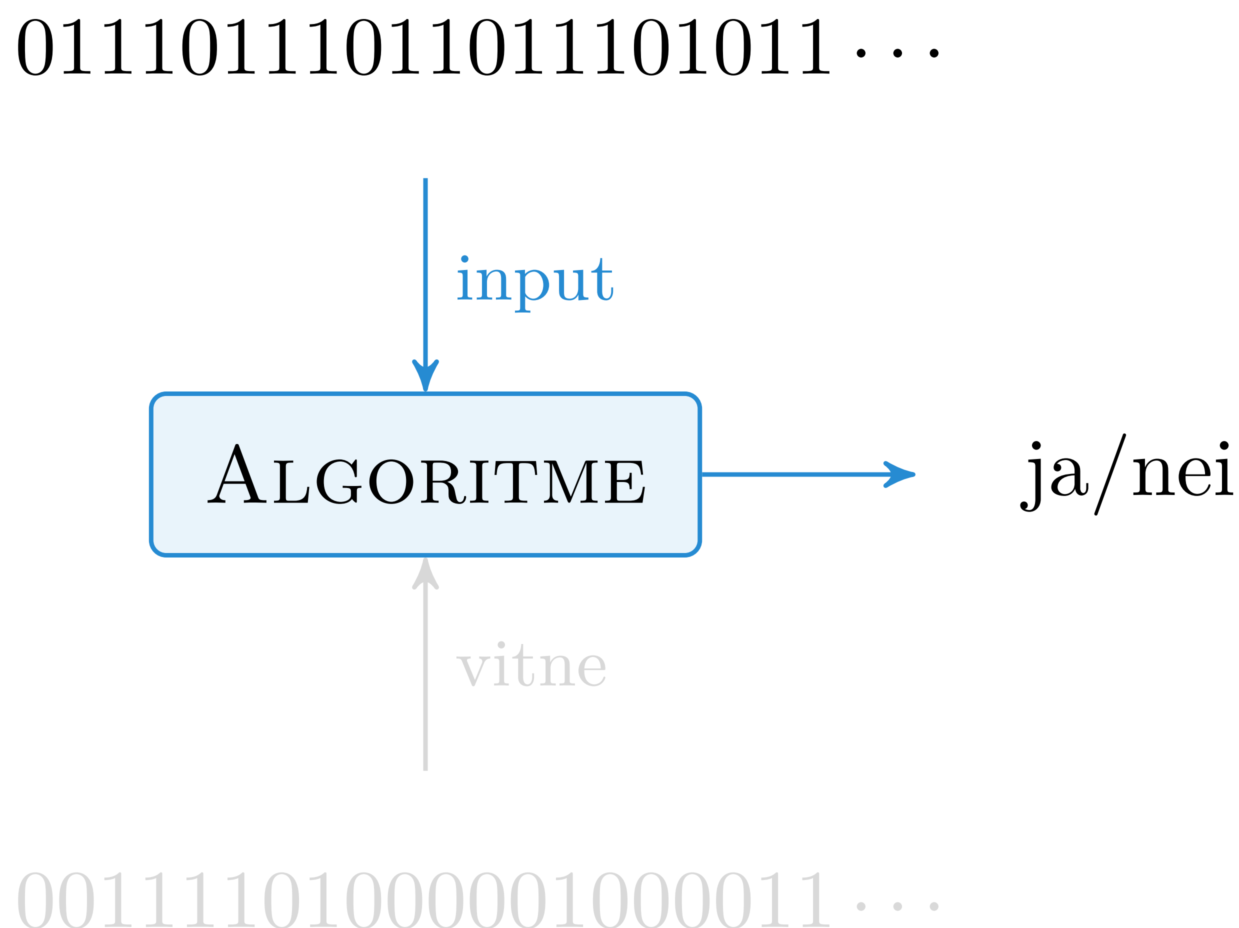
$$x \in \mathrm{L}' \iff f(x) \in \mathrm{L}\,,$$

for alle $x \in \{0,1\}^*$
- Vis at algoritimen som beregner f har polynomisk kjøretid

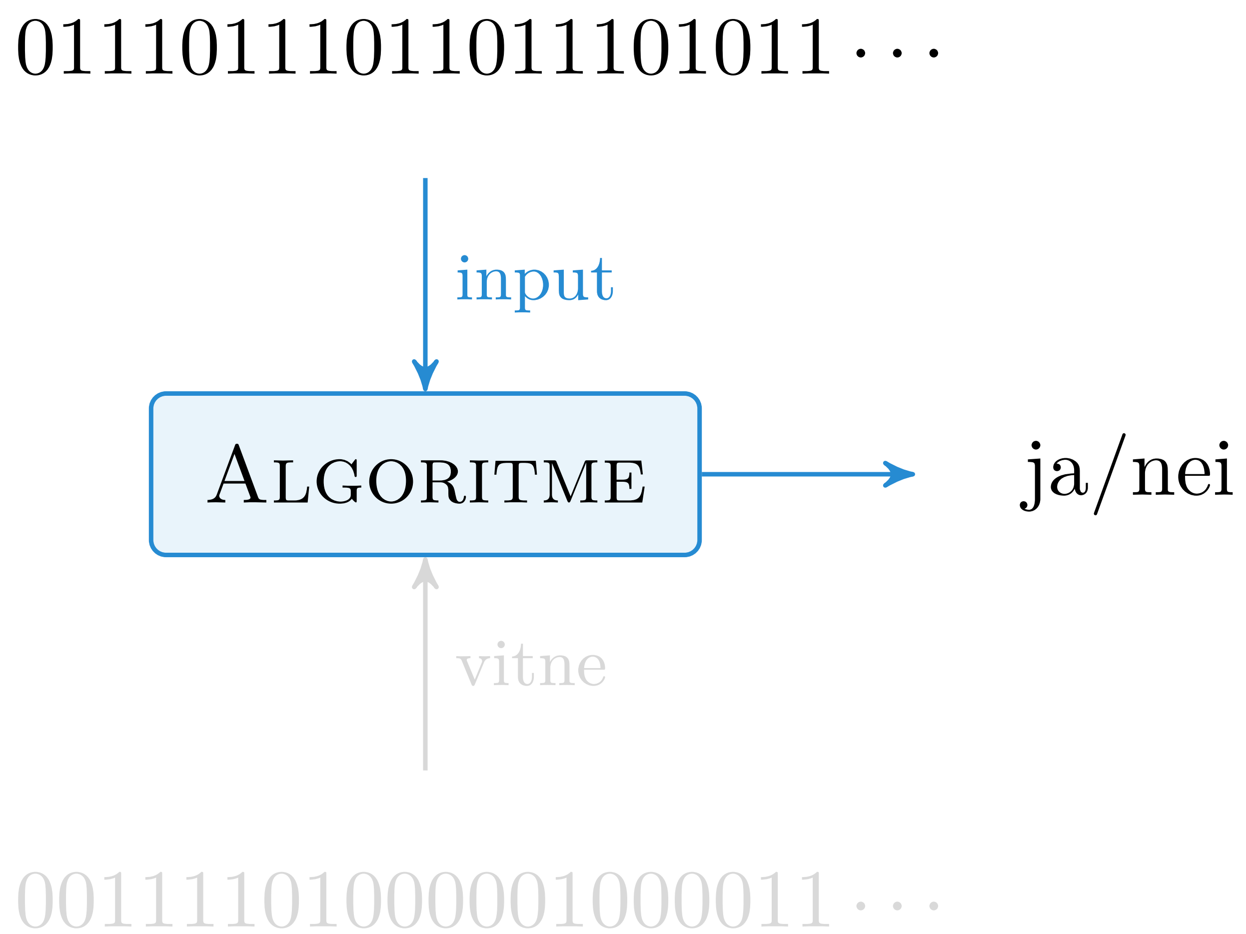
1. Problemer
2. Reduksjoner
3. Kompletthet

Bonusmateriale

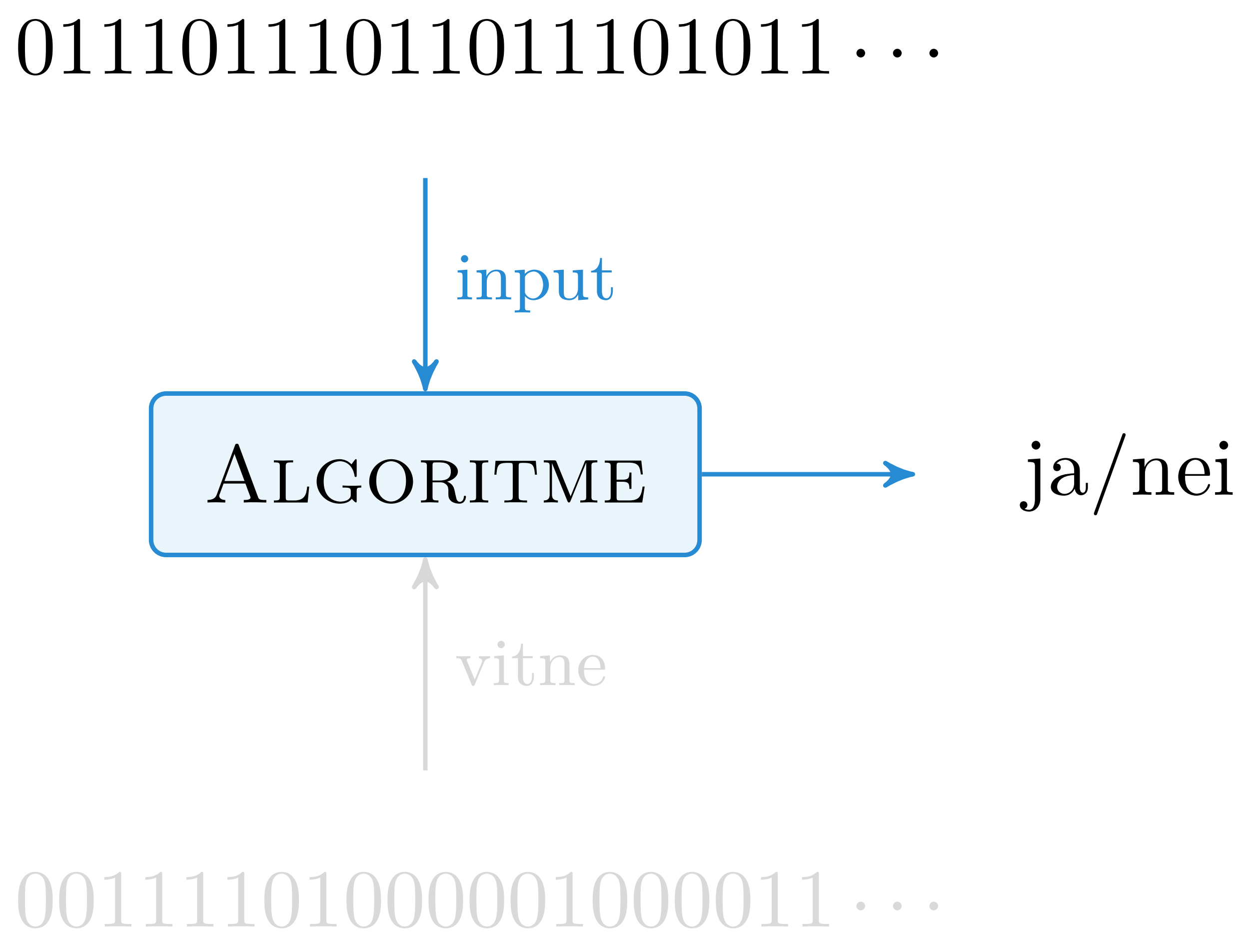
Rekonstruksjon av sertifikater



Hvis **P** = **NP** kan vi svare på om det finnes et vitne

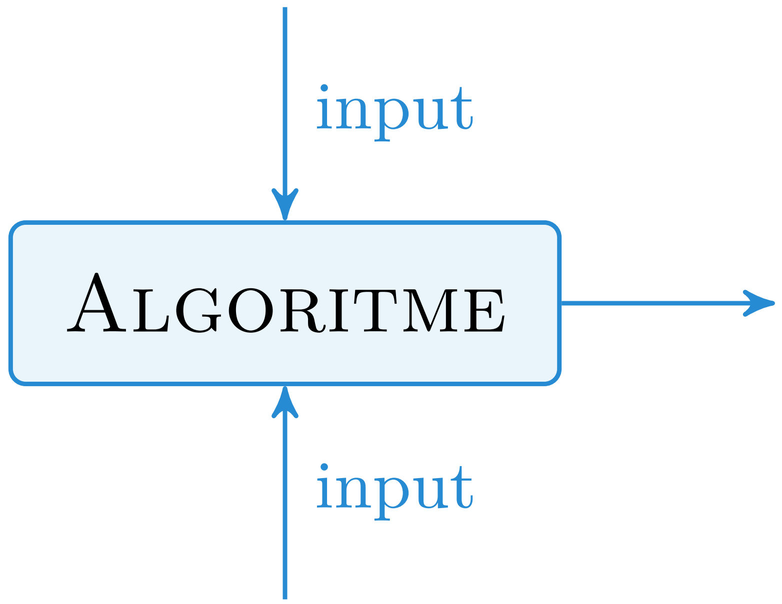


Ikke bare det: Vi kan rekonstruere vitnet!



Vi lager oss nye beslutningsproblemer, med deler av vitnet som input

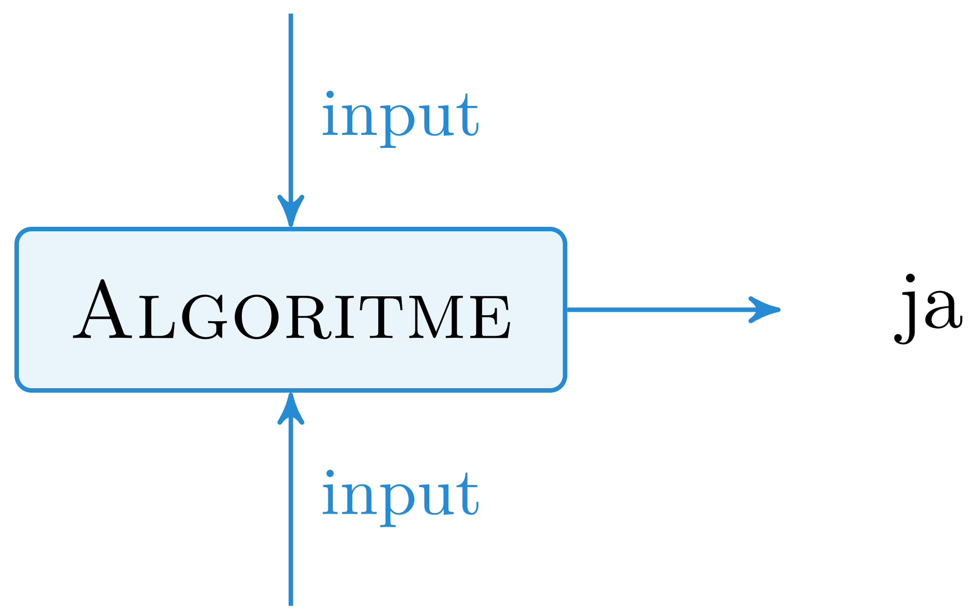
0111011101101110101 $\cdots$



0011110100000100011 $\cdots$

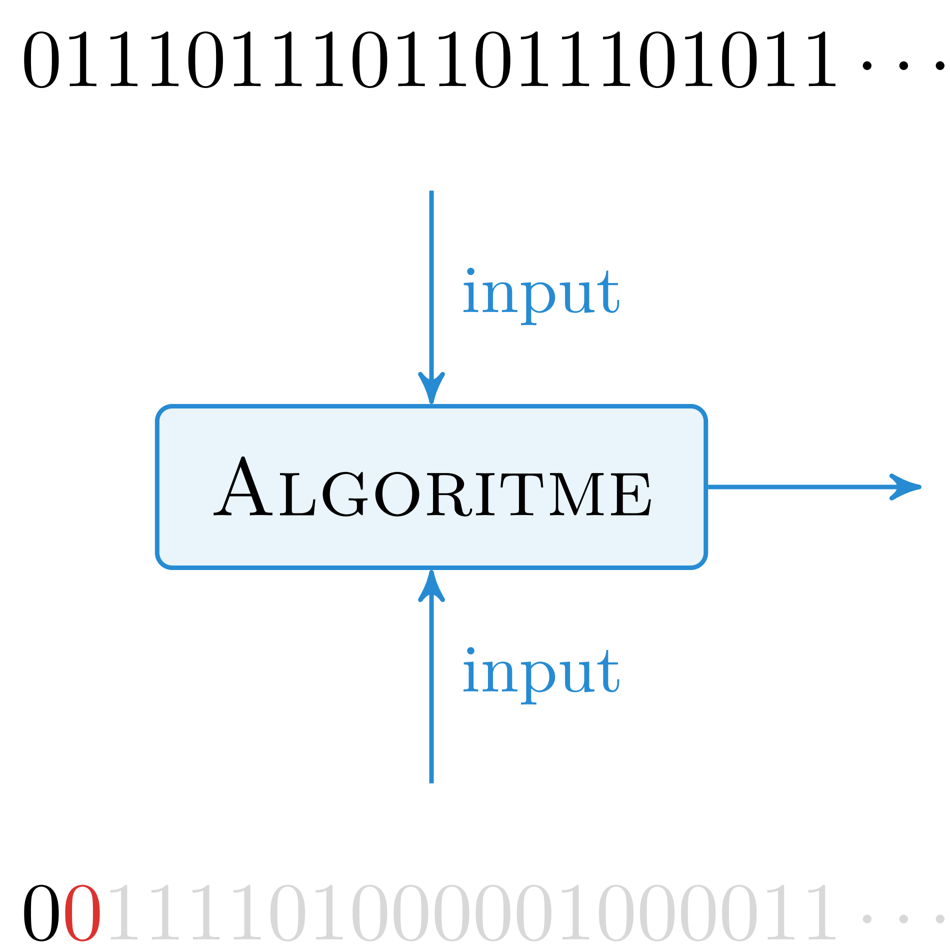
Problem 1: Finnes et vitne som starter med 0?

$0111011101101110101 1\cdots$

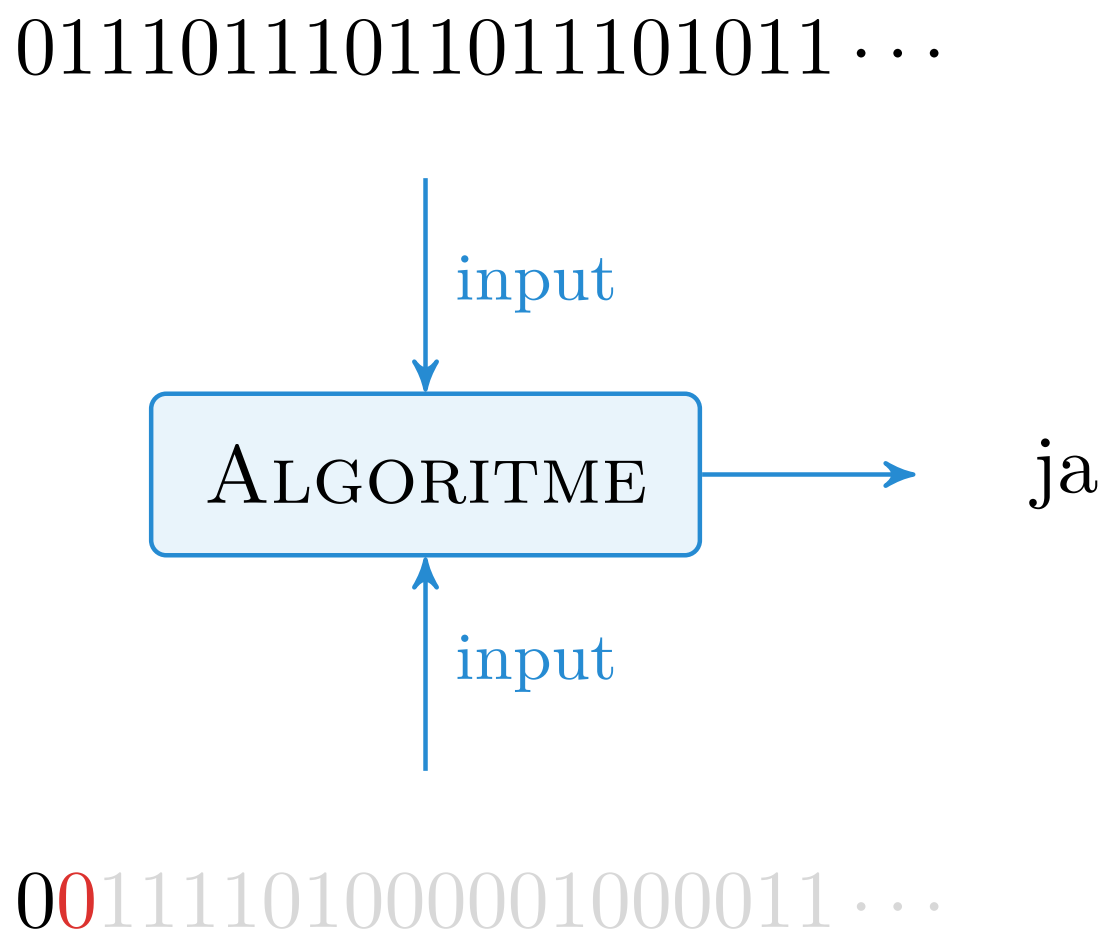


$0011110100000100001 1\cdots$

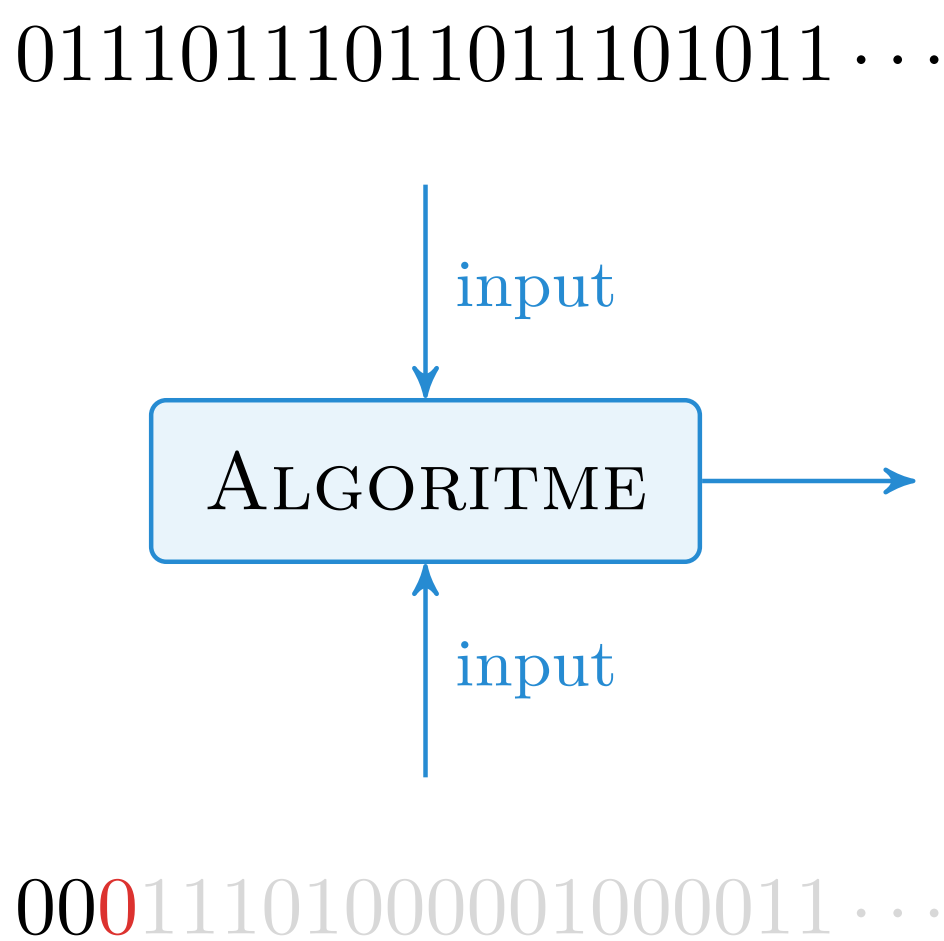
Ja, det gjør det. Vi setter første siffer til 0



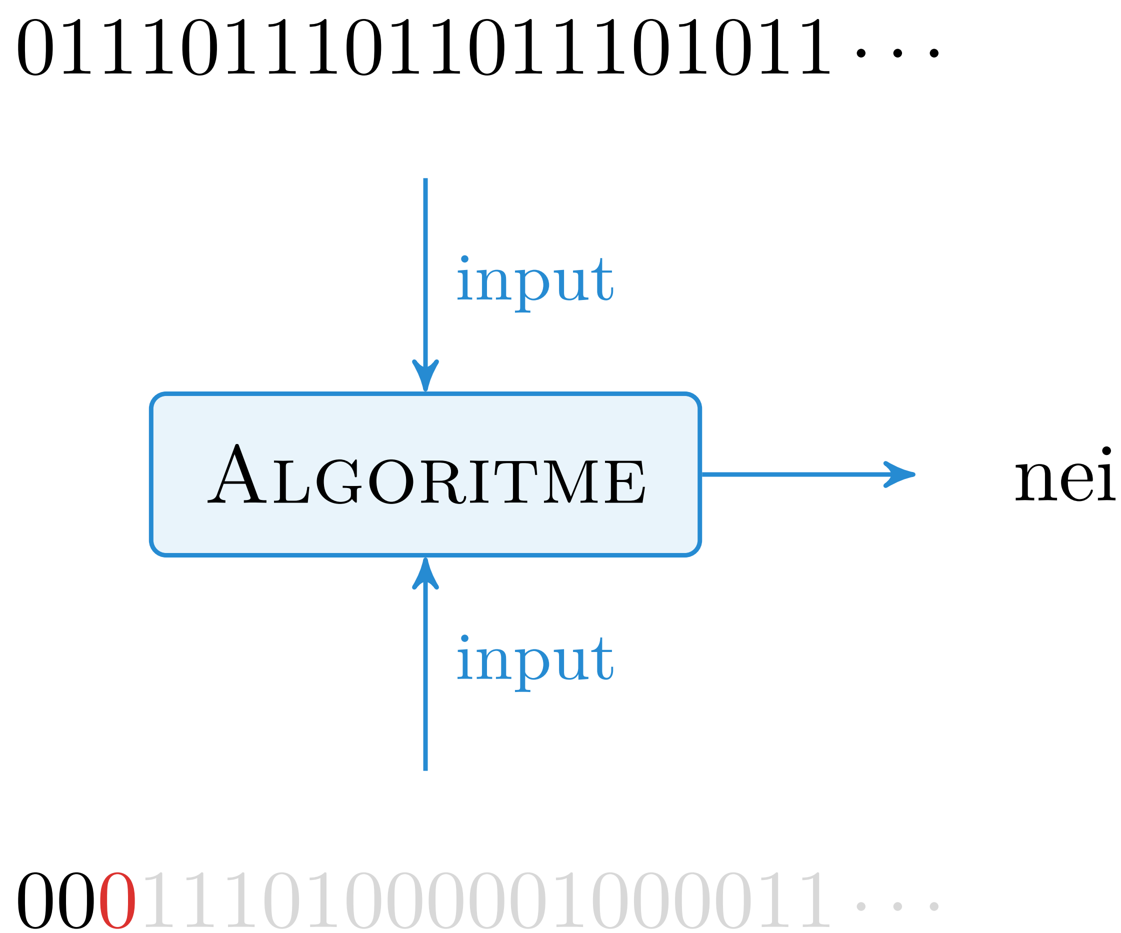
Problem 2: Finnes et vitne som starter med 00?



Ja, det gjør det. Vi setter andre siffer til 0

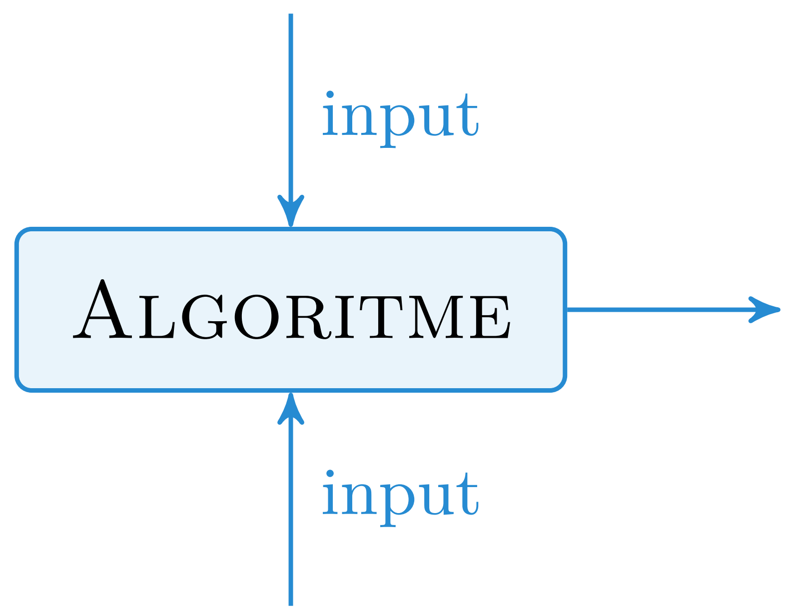


Problem 3: Finnes et vitne som starter med 000?



Nei, det gjør det ikke. Vi setter tredje siffer til 1

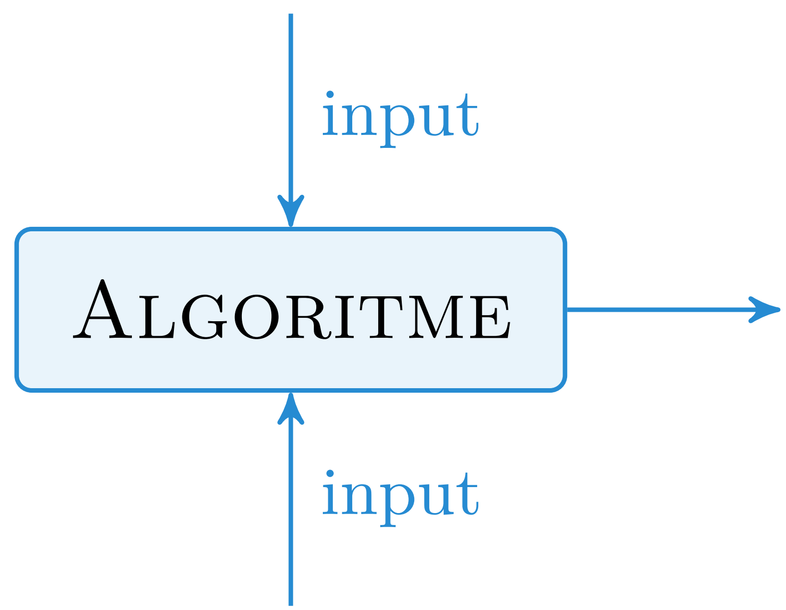
0111011101101110101 1 · · ·



0010110100000100001 1 · · ·

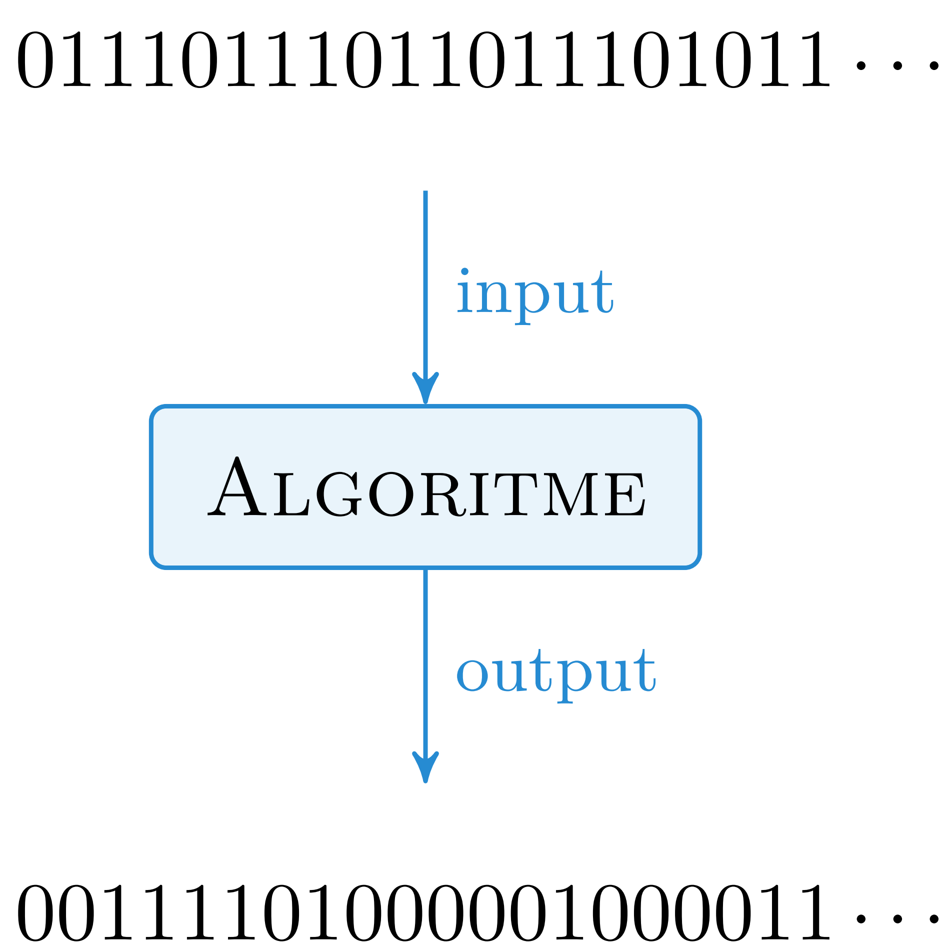
Problem 4: Finnes et vitne som starter med 0010?

$0111011101101110 1011\cdots$



$0011110100000 1000011\cdots$

Og slik fortsetter vi, til vi har konstruert et vitne



Med andre ord: Om vi kan løse beslutningsproblemer . . .

$0111011101101110101 1\cdots$

input

ALGORITME

output

$0011110100000100001 1\cdots$

…så kan vi løse «søkeproblemer» også, og finne gyldig output!

Litt mer om problemer

- **Abstrakt problem:**
  Binær relasjon $\mathrm{Q} \subseteq \mathrm{I} \times \mathrm{S}$
- **Encoding:**
  Funksjon til $\{0, 1\}^*$
- **Polynomisk kjøretid:**
  $\mathrm{T}(n) = \mathrm{O}(n^k)$, for en eller annen $k > 0$, der n er antall bits i encodingen til instansene.
- **Polynomisk relaterte encodings:**
  Kan du transformere mellom dem i pol. tid, spiller det ingen rolle hvilken du bruker.
- **Rimelige encodings:**
  Unngå spesielt éntallssystemet!

› **0-1 Knapsack:**
Vår DP-løsning har kjøretid $T(n, W) = \Theta(nW)$

Egentlig skal funksjonen ha ett argument; det lar seg gjøre

- **0-1 Knapsack:**
  Vår DP-løsning har kjøretid $T(n, W) = \Theta(nW)$
- **Encoding:**
  For enkelhets skyld, la oss si vi bruker $\Theta(n)$ bits på objektene. En rimelig encoding vil bruke $\Theta(m)$ bits på kapasiteten, der $m = \lg W$.

Objektene tar egentlig $\Theta(n \lg n)$ plass; ikke så viktig her

- **0-1 Knapsack:**
  Vår DP-løsning har kjøretid $T(n, W) = \Theta(nW)$
- **Encoding:**
  For enkelhets skyld, la oss si vi bruker $\Theta(n)$ bits på objektene. En rimelig encoding vil bruke $\Theta(m)$ bits på kapasiteten, der $m = \lg W$.
- **Polynomisk?**
  T er polynomisk som funksjon av n og W, men er den «polynomisk»?

Implisitt: Som funksjon av lengden på instans-encodingen vår

- **0-1 Knapsack:**
  Vår DP-løsning har kjøretid $T(n, W) = \Theta(nW)$
- **Encoding:**
  For enkelhets skyld, la oss si vi bruker $\Theta(n)$ bits på objektene. En rimelig encoding vil bruke $\Theta(m)$ bits på kapasiteten, der $m = \lg W$.
- **Polynomisk?**
  T er polynomisk som funksjon av n og W, men er den «polynomisk»?
- **Nei!**
  Vi må da skrive den som funksjon av n og m, og får $T(n, m) = \Theta(n2^m)$

Egentlig som funksjon av $n + m$, men konklusjonen blir den samme

- **Beslutningsproblem:**
  Ja-/nei-spørsmål. Output er 0 eller 1; ikke tvetydig (altså, en funksjon, ikke generell relasjon)
- **Konkret beslutningsproblem:**
  En funksjon fra $\{0,1\}^*$ til $\{0,1\}$.
  Ugyldige strenger mappes til 0.
- **Optimeringsproblem:**
  Vil maksimere eller minimere en verdi
- **Terskling:**
  Terskelversjonen av et optimeringsproblem er et beslutningsproblem. Kan løses vha. opt., og kan altså ikke være vanskeligere.

Beslutning vanskelig $\implies$ optimering vanskelig

Illustrasjon av P, NP og NPC

NP?

NP

NP

Nondeterministic Polynomial Time

Beslutningsproblemer* som kan verifiseres i polynomisk tid.†

* Uttrykt som formelle språk

† Gitt sertifikat av pol. størrelse

problemer som kan
løses i konstant tid

$O(1)$

$\omega(n^c)$

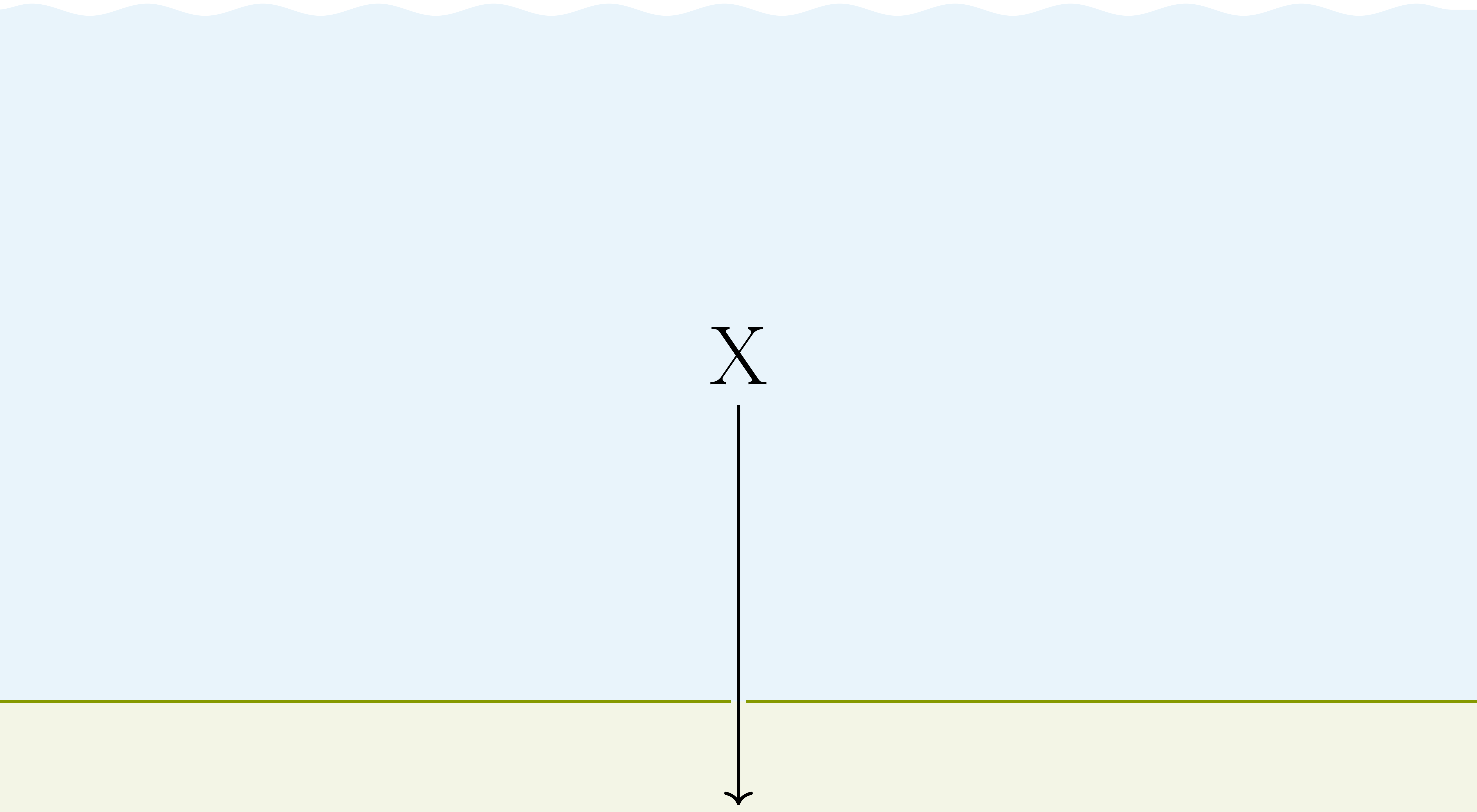
problemer som krever
superpolynomisk tid

$O(1)$

$$\omega(n^c)$$

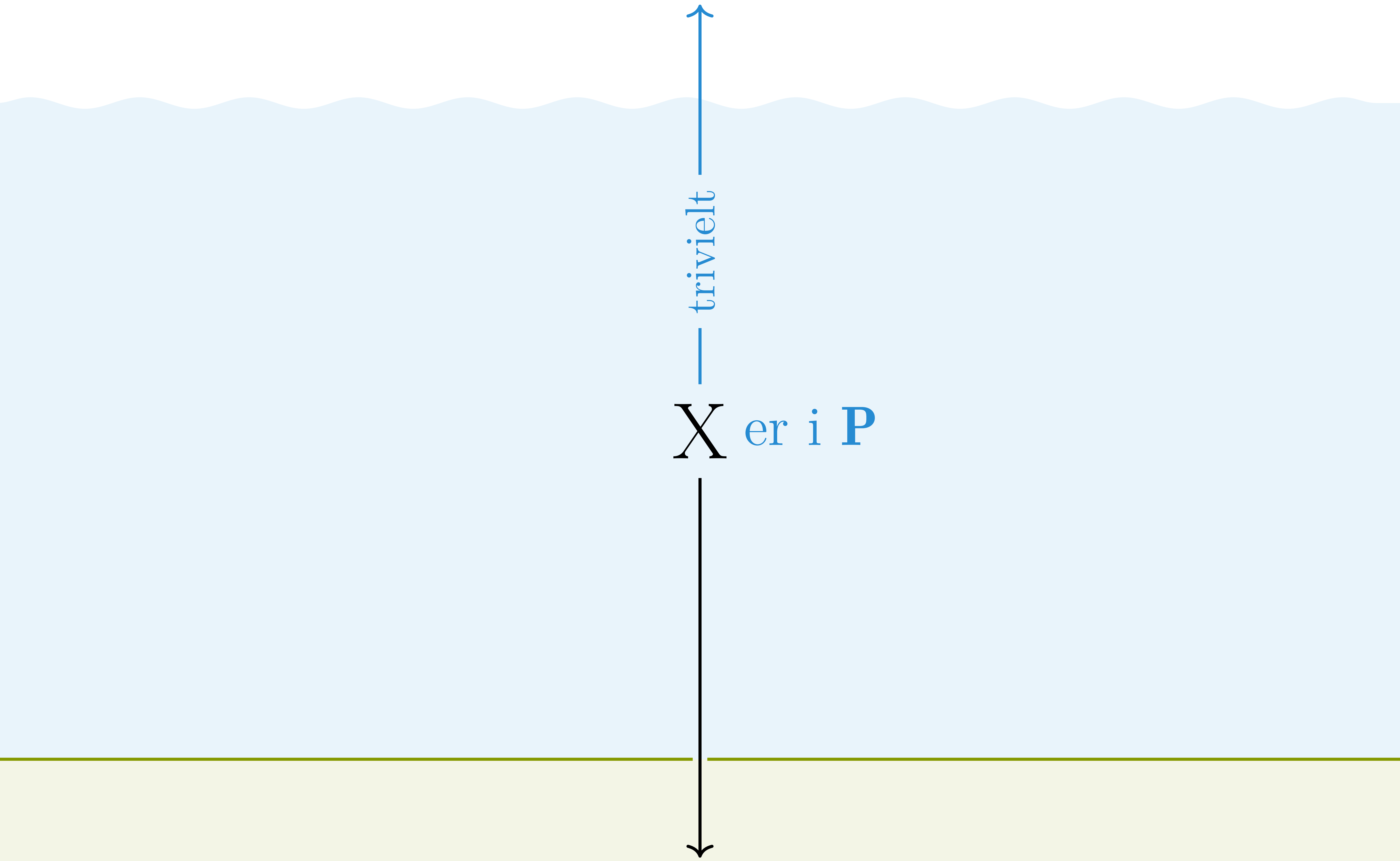
$$O(1)$$

$$\omega(n^c)$$



$\omega(n^c)$

X er i **P**

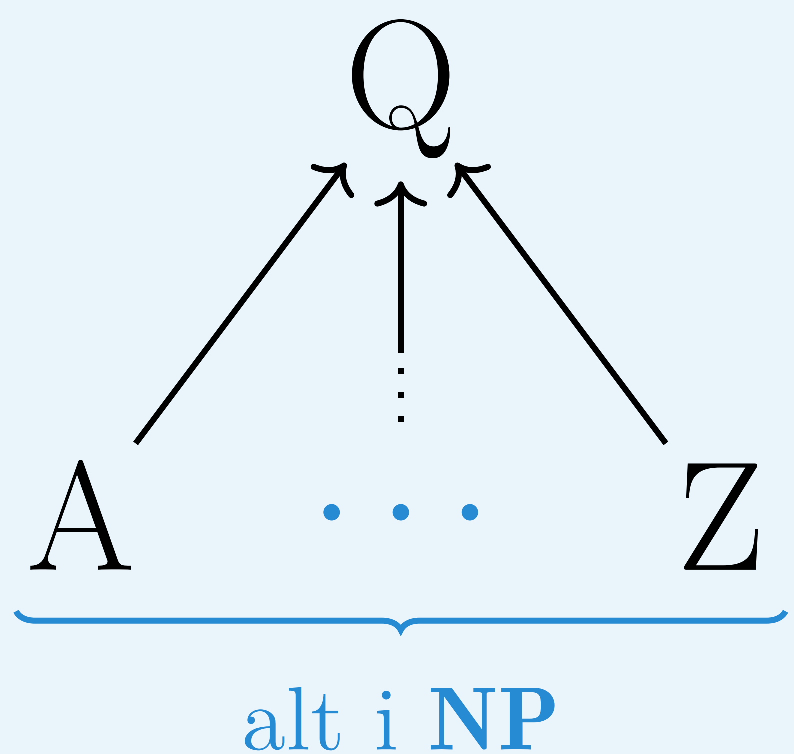

trivielt
X er i **P**

$$\omega(n^c)$$

$$\underbrace{\text{A} \quad \cdots \quad \text{Z}}_{\text{alt i } \mathbf{NP}}$$

$$\mathrm{O}(1)$$

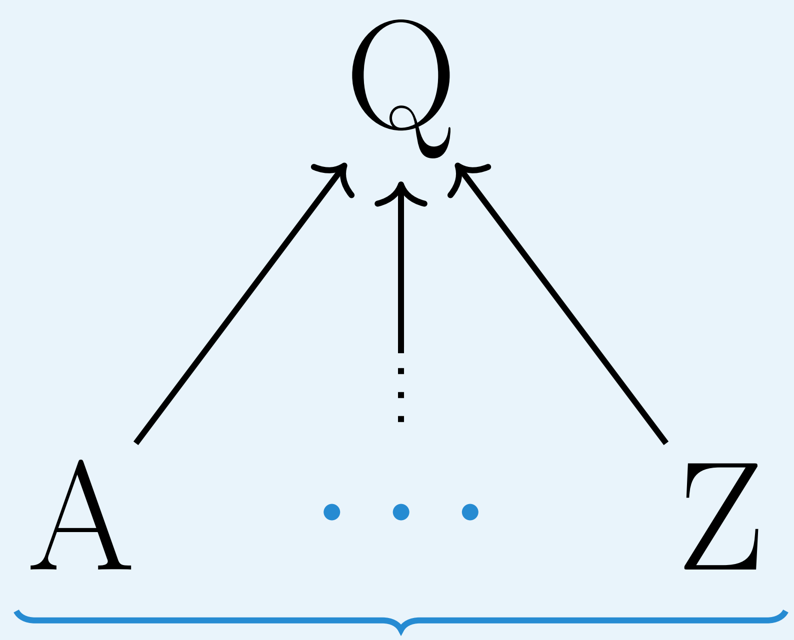
$$\omega(n^c)$$



$$O(1)$$

$\omega(n^c)$

i **NPC**, per def.

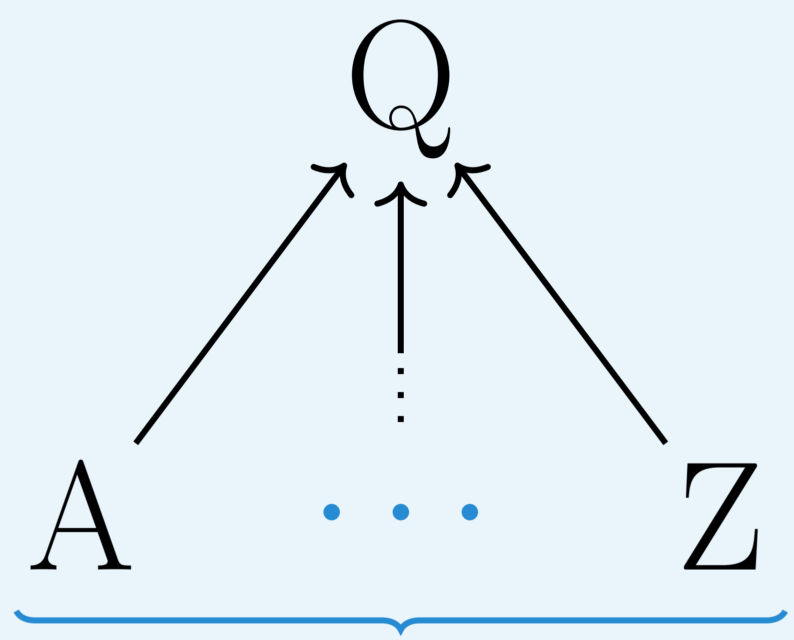


alt i **NP**

$O(1)$

$\omega(n^c)$

i **NPC**

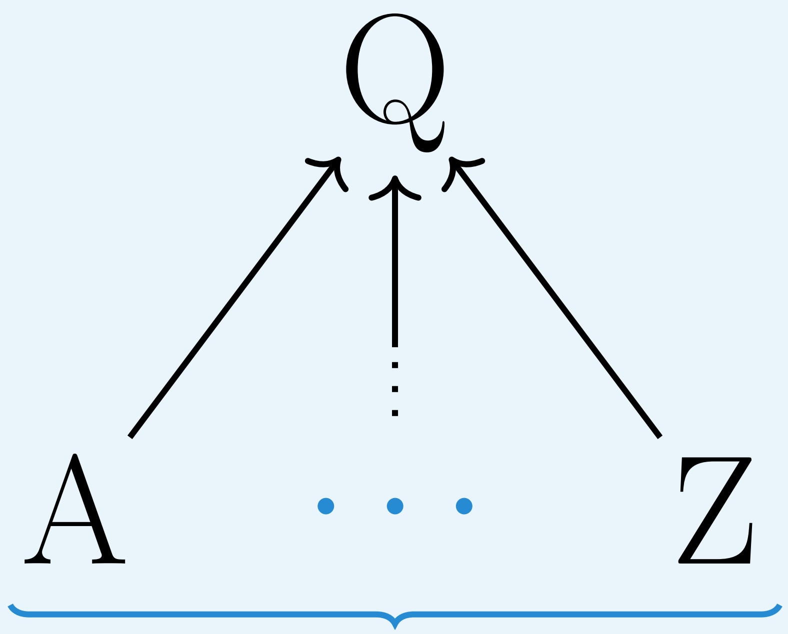


alt i **NP**, inkl. **P** og **NPC**

$O(1)$

$\omega(n^c)$

i **NPC**

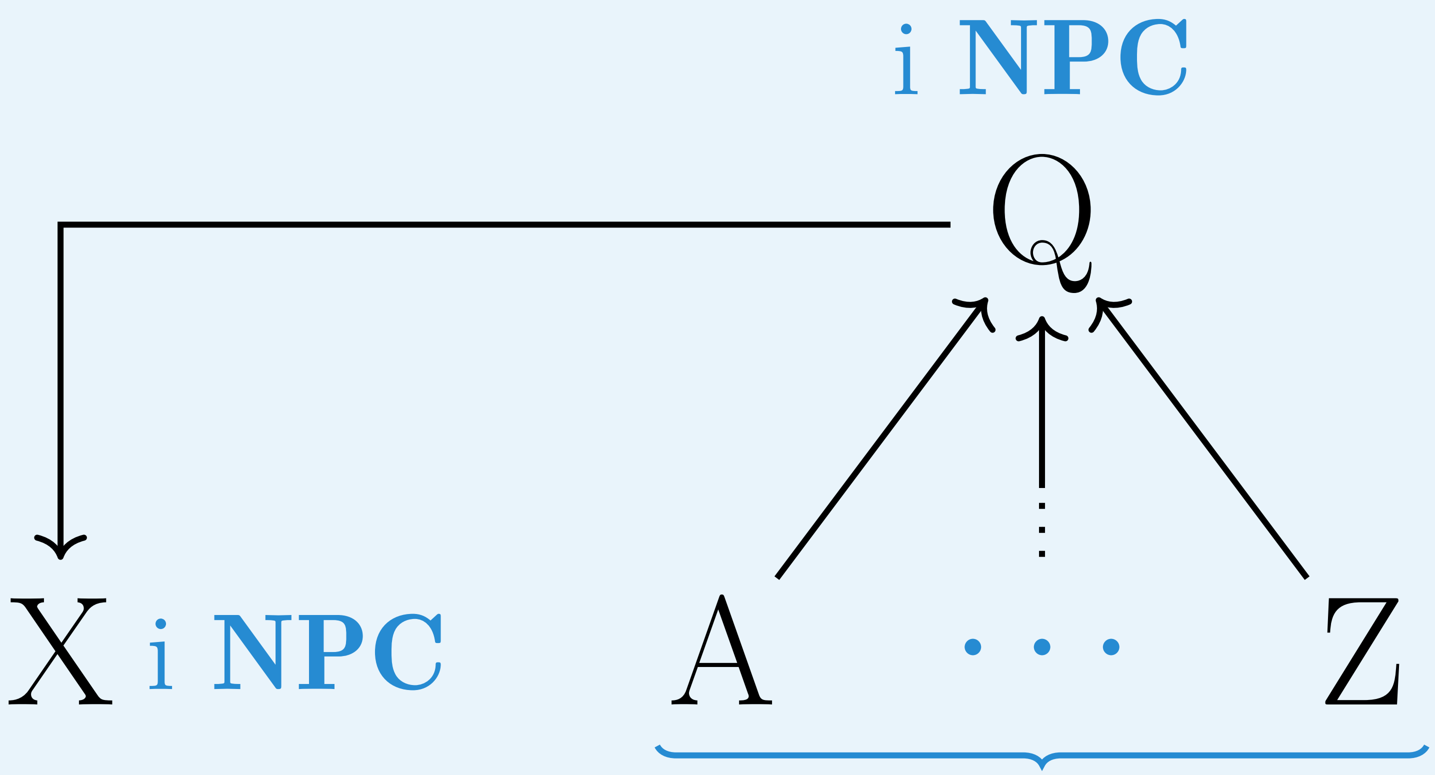


alt i **NP**, inkl. **P** og **NPC**

(så probl. i **NPC** reduserer til hverandre)

$O(1)$

$\omega(n^c)$

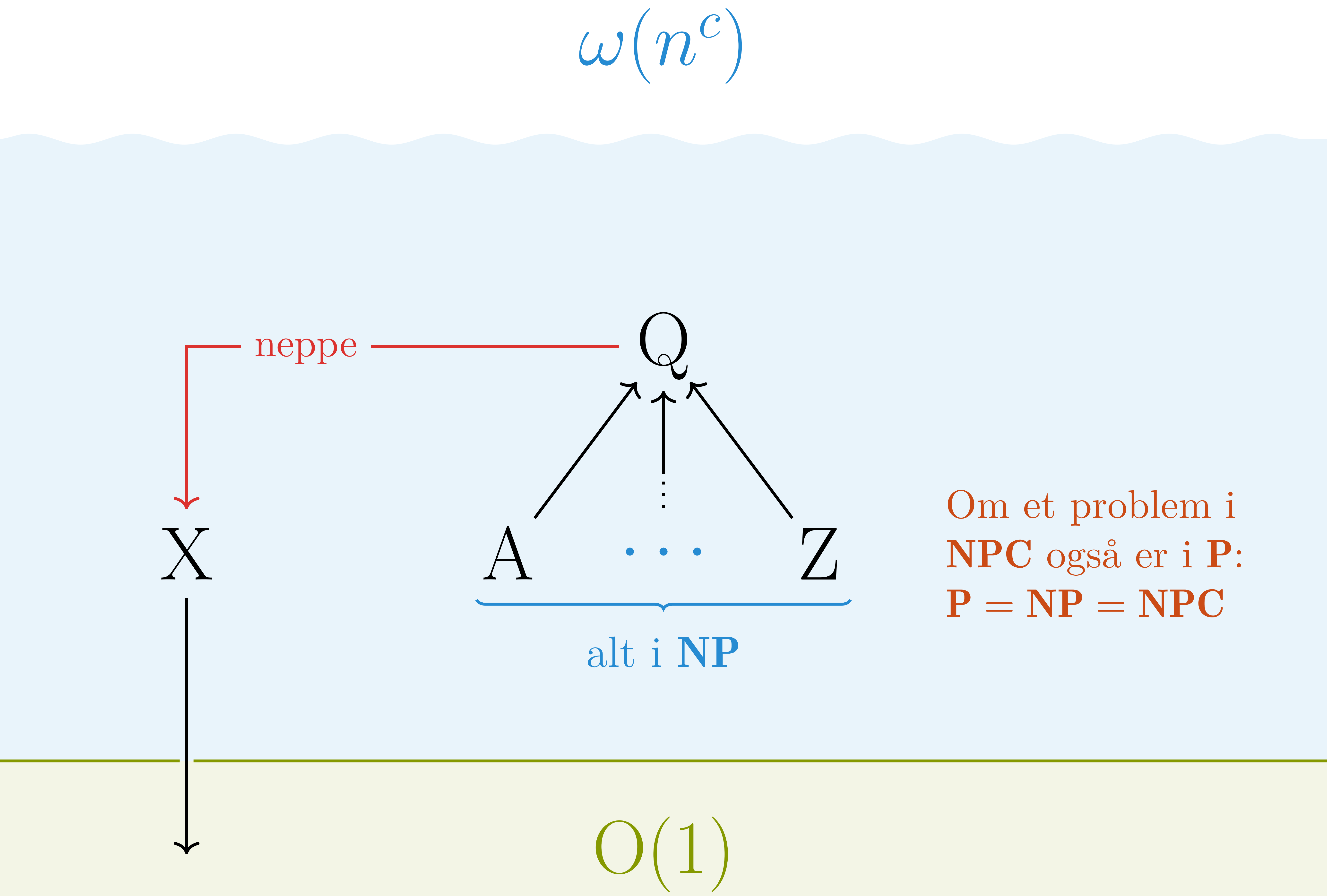


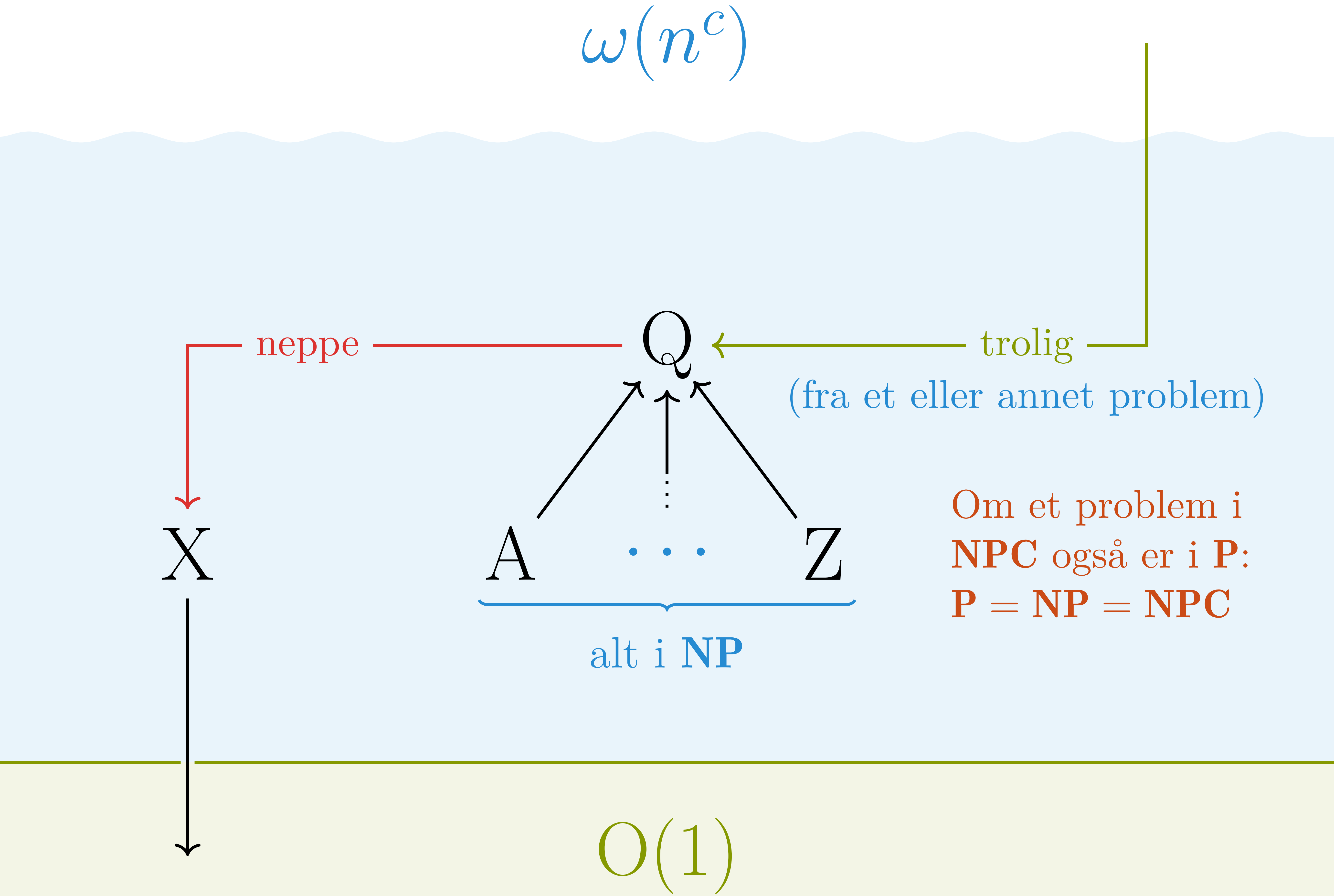
alt i **NP**, inkl. **P** og **NPC**

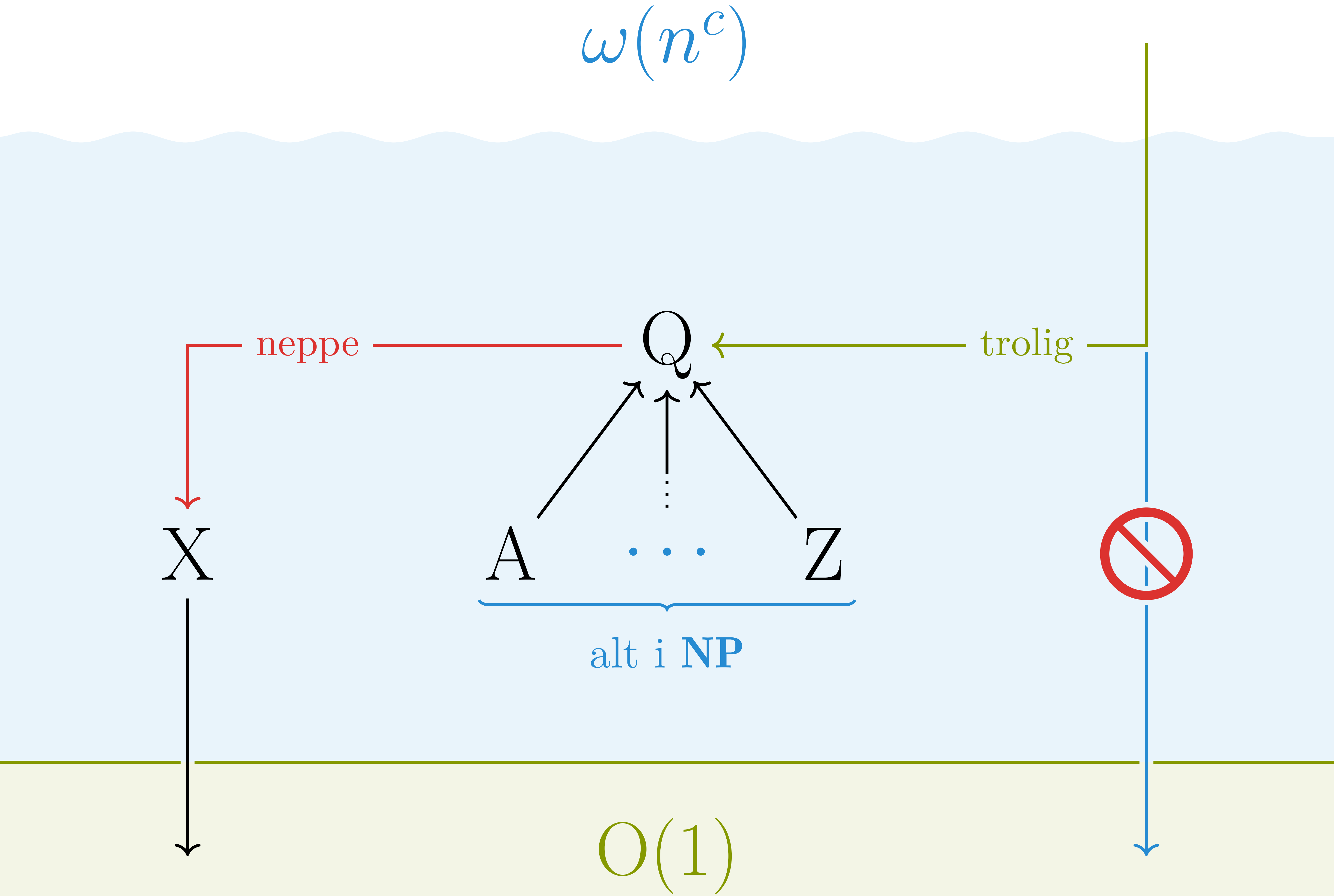
(så probl. i **NPC** reduserer til hverandre)

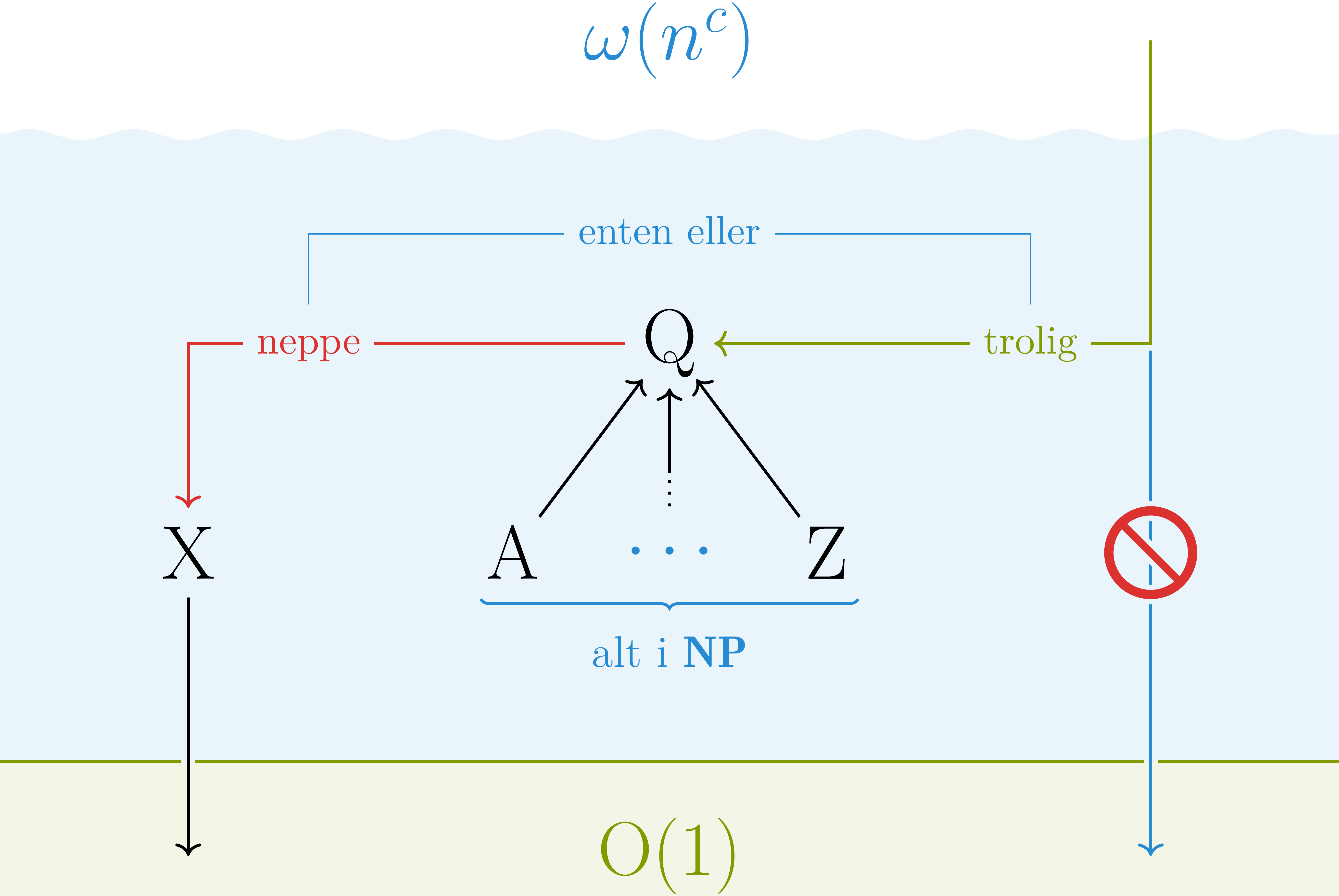
NPC-bevis:
Redusér *fra* et
problem i **NPC**

$O(1)$


$\omega(n^c)$
neppe
Q
X
A
. . .
Z
alt i **NP**
Om et problem i
NPC også er i **P**:
P = **NP** = **NPC**
O(1)


$\omega(n^c)$
neppe
Q
trolig
(fra et eller annet problem)
X
A
$\cdots$
Z
alt i **NP**
Om et problem i **NPC** også er i **P**:
P = **NP** = **NPC**
$O(1)$


$\omega(n^c)$
neppe
Q
trolig
X
A
. . .
Z
alt i **NP**
O(1)


ω(n^c)
enten eller
neppe
Q
trolig
X
A
. . .
Z
alt i NP
O(1)