Forelesning 2b

Vi jobber videre med grunnleggende byggesteiner, og skisserer et rammeverk for å tilegne seg resten av stoffet. Vi ser på hvordan enkle algoritmer kan transformere, eller *redusere*, mellom ulike problemer eller delproblemer, og bygge videre på løsninger eller delløsninger.

Pensum

- ☐ App. B i pensumheftet: B.1

Læringsmål

Fra pensumheftet

[B_6] Forstå definisjonene av *søkeproblemer*, *beslutningsproblemer* og *optimeringsproblemer*

[B_7] Forstå ulike typer reduksjoner (*Turing*, *Cook*, *Levin* og *Karp*), og forholdet mellom dem

Støtter også følgende mål:

[A_7] Forstå *løkkeinvarianter* og *induksjon*

[C_1] Forstå *rekursiv dekomponering* og *induksjon over delproblemer*

[M_6] Forstå *redusibilitets-relasjonen* som relativ vanskegrad.

mlh@ntnu.no

Forelesningen filmes

s.ntnu.no/video-opptak

Forelesning 2

Problemer og reduksjoner

1. **Problemtyper**
2. **Reduksjoner**
3. **Dekomponering**

1:3

Problemer

Et spesifikt problem svarer til en beskrivelse av akseptable løsninger.

Et spesifikt problem svarer til en beskrivelse av akseptable løsninger.

Et generelt problem er en familie med spesifikke problemer.

Dette er det vi normalt bare kaller et problem

mlh@ntnu.no

Et spesifikt problem svarer til en beskrivelse av akseptable løsninger.

Et generelt problem er en familie med spesifikke problemer.

Dette kaller vi en instans av problemet

Et spesifikt problem svarer til en beskrivelse av akseptable løsninger.

Et generelt problem er en familie med spesifikke problemer.

Vi kan se på det som en binær relasjon mellom input og output.

Input er det som identifiserer instansen

mlh@ntnu.no

Et spesifikt problem svarer til en beskrivelse av akseptable løsninger.

Et generelt problem er en familie med spesifikke problemer.

Vi kan se på det som en binær relasjon mellom input og output.

Vi bruker gjerne «input» og «instans» om hverandre

mlh@ntnu.no

For eksempel er sortering et generelt problem . . .

For eksempel er sortering et generelt problem . . .

Vi får inn sekvenser og skal produsere sorterte permutasjoner

mlh@ntnu.no

For eksempel er sortering et generelt problem . . .

. . . mens å skulle sortere $a = \langle 5, 2, 4, 6, 1, 3 \rangle$ er en bestemt instans av problemet.

For eksempel er sortering et generelt problem . . .

. . . mens å skulle sortere $a = \langle 5, 2, 4, 6, 1, 3 \rangle$ er en bestemt instans av problemet.

Vi omtaler da gjerne a i seg selv som en instans.

mlh@ntnu.no

I

Vi har altså en mengde inputs/instanser

mlh@ntnu.no

I O

Vi har også en mengde outputs/resultater

mlh@ntnu.no

$$\mathrm{I} \times \mathrm{O}$$

Det cartesiske produktet er mengden av par (x, y) der $x \in \mathrm{I}, y \in \mathrm{O}$

mlh@ntnu.no

$$R \subseteq I \times O$$

En binær relasjon mellom I og O er en delmengde av produktet

mlh@ntnu.no

$$\mathbf{R} \subseteq \mathbf{I} \times \mathbf{O}$$

R angir hvilke outputs som er riktige løsninger for hvilke inputs

mlh@ntnu.no

$$\mathbf{R} \subseteq \mathbf{I} \times \mathbf{O}$$

Merk: Noen instanser har kanskje ingen løsninger

mlh@ntnu.no

$$\mathrm{R} \subseteq \mathrm{I} \times \mathrm{O}$$

Vi kan tenke på disse som «ugyldige inputs»

Vi gir gjerne input og output en konkret representasjon

mlh@ntnu.no

$$\mathrm{R} \subseteq \{0,1\}^* \times \{0,1\}^*$$

Vi gir gjerne input og output en konkret representasjon

mlh@ntnu.no

$$\mathrm{R} \subseteq \{0,1\}^* \times \{0,1\}^*$$

Vi bruker bitstrenger til å kode de abstrakte objektene

mlh@ntnu.no

Gitt $\mathrm{R} \subseteq \{0,1\}^* \times \{0,1\}$, er det flere ting vi kan ønske å gjøre. For hver $x \in \{0,1\}^*$ kan vi f.eks.

Gitt $\mathrm{R} \subseteq \{0,1\}^* \times \{0,1\}$, er det flere ting vi kan ønske å gjøre. For hver $x \in \{0,1\}^*$ kan vi f.eks.

1. Prøve å finne en y slik at $(x,y) \in \mathrm{R}$;

Vi sier da at vi har et søkeproblem

Gitt $\mathrm{R} \subseteq \{0,1\}^* \times \{0,1\}$, er det flere ting vi kan ønske å gjøre. For hver $x \in \{0,1\}^*$ kan vi f.eks.

1. Prøve å finne en y slik at $(x,y) \in \mathrm{R}$;
2. For en gitt y, sjekke om $(x,y) \in \mathrm{R}$;

Vi kaller dette å verifisere y

Gitt $\mathrm{R} \subseteq \{0,1\}^* \times \{0,1\}$, er det flere ting vi kan ønske å gjøre. For hver $x \in \{0,1\}^*$ kan vi f.eks.

1. Prøve å finne en y slik at $(x,y) \in \mathrm{R}$;
2. For en gitt y, sjekke om $(x,y) \in \mathrm{R}$;
3. Finne ut om y finnes, slik at $(x,y) \in \mathrm{R}$.

Dette kaller vi et beslutningsproblem

mlh@ntnu.no

Gitt $\mathrm{R} \subseteq \{0,1\}^* \times \{0,1\}$, er det flere ting vi kan ønske å gjøre. For hver $x \in \{0,1\}^*$ kan vi f.eks.

1. Prøve å finne en y slik at $(x,y) \in \mathrm{R}$;
2. For en gitt y, sjekke om $(x,y) \in \mathrm{R}$;
3. Finne ut om y finnes, slik at $(x,y) \in \mathrm{R}$.

(Beslutningsproblemer defineres gjerne som en mengde instanser)

mlh@ntnu.no

Å verifisere en løsning kan være enklere enn å faktisk finne den (eller å løse beslutningsproblemet)

IS-SORTED(A, n)

```
IS-SORTED(A, n)
1  for i = 1 to n − 1
```

```
IS-SORTED(A, n)
1  for i = 1 to n − 1
2        if A[i] > A[i + 1]
```

```
IS-SORTED(A, n)
1  for i = 1 to n − 1
2        if A[i] > A[i + 1]
3              return FALSE
```

```
IS-SORTED(A, n)
1  for i = 1 to n − 1
2        if A[i] > A[i + 1]
3              return FALSE
4  return TRUE
```

For å verifisere, må vi ha både input og output. Enklest her om vi får permutasjonen som sorterer A

```
SORTS(A, π, n)
1  for i = 1 to n − 1
2        if A[π[i]] > A[π[i + 1]]
3              return FALSE
4  return TRUE
```

(Bør egentlig sjekke at π faktisk er en permutasjon . . .)

mlh@ntnu.no

```
SORTS(A, π, n)
1  for i = 1 to n − 1
2        if A[π[i]] > A[π[i + 1]]
3              return FALSE
4  return TRUE
```

Vi kan sjekke π i lineær tid . . .

mlh@ntnu.no

```
SORTS(A, π, n)
1  for i = 1 to n − 1
2        if A[π[i]] > A[π[i + 1]]
3              return FALSE
4  return TRUE
```

. . . men å finne π: $\Omega(n \lg n)$ i verste tilfelle!

mlh@ntnu.no

```
SORTS(A, π, n)
1  for i = 1 to n − 1
2        if A[π[i]] > A[π[i + 1]]
3              return FALSE
4  return TRUE
```

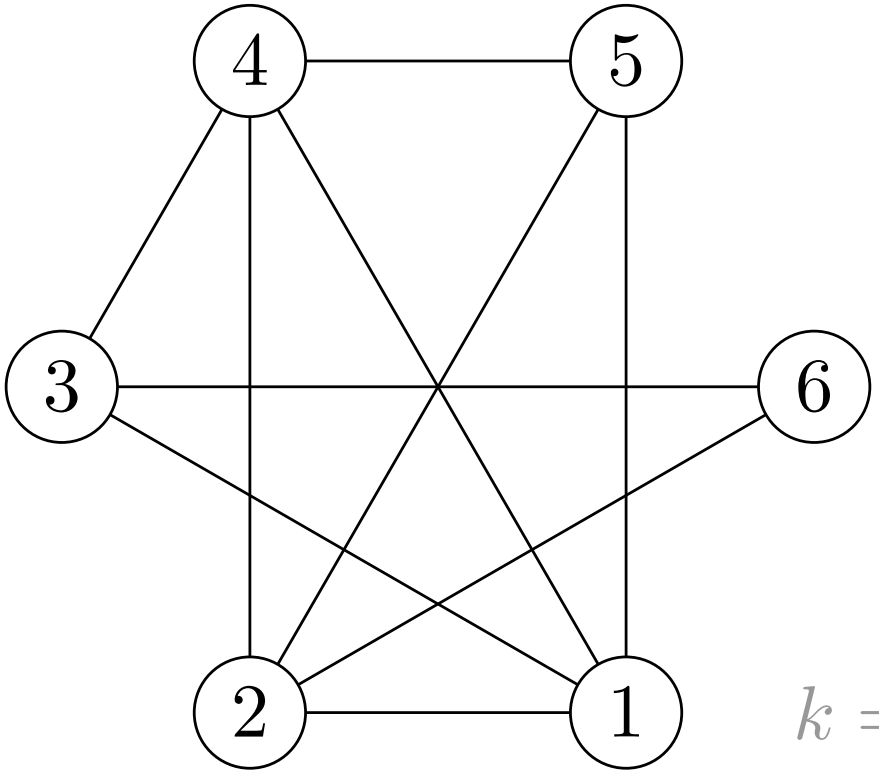
(Mer om det i forelesning 4)

mlh@ntnu.no

CLIQUE

CLIQUE · Relasjon

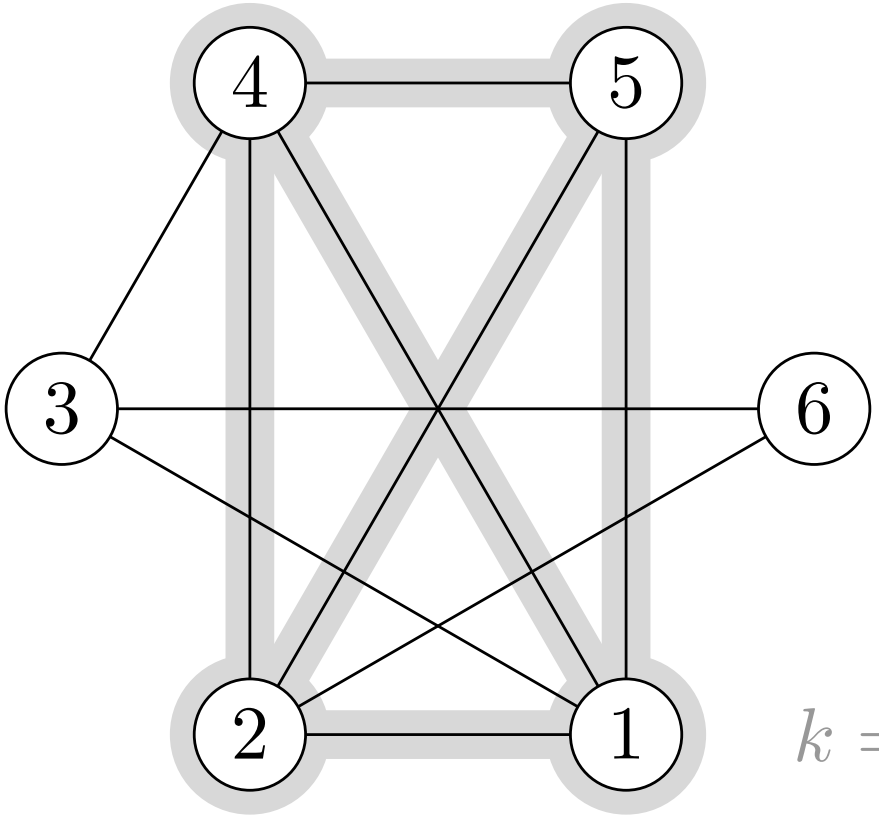
mlh@ntnu.no



CLIQUE · **Relasjon**

Instans: En urettet graf G og et positivt heltall k

mlh@ntnu.no

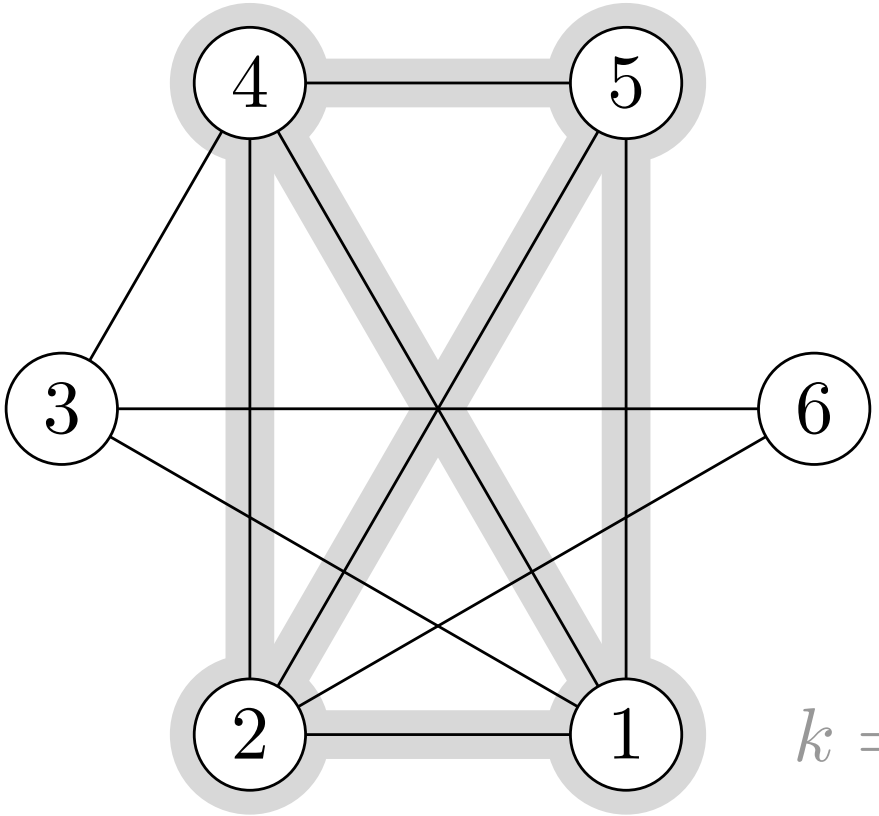


CLIQUE · **Relasjon**

Instans: En urettet graf G og et positivt heltall k

Løsning: En en komplett delgraf med k noder

For verifikasjon, kaller vi gjerne den påståtte løsningen et «sertifikat»



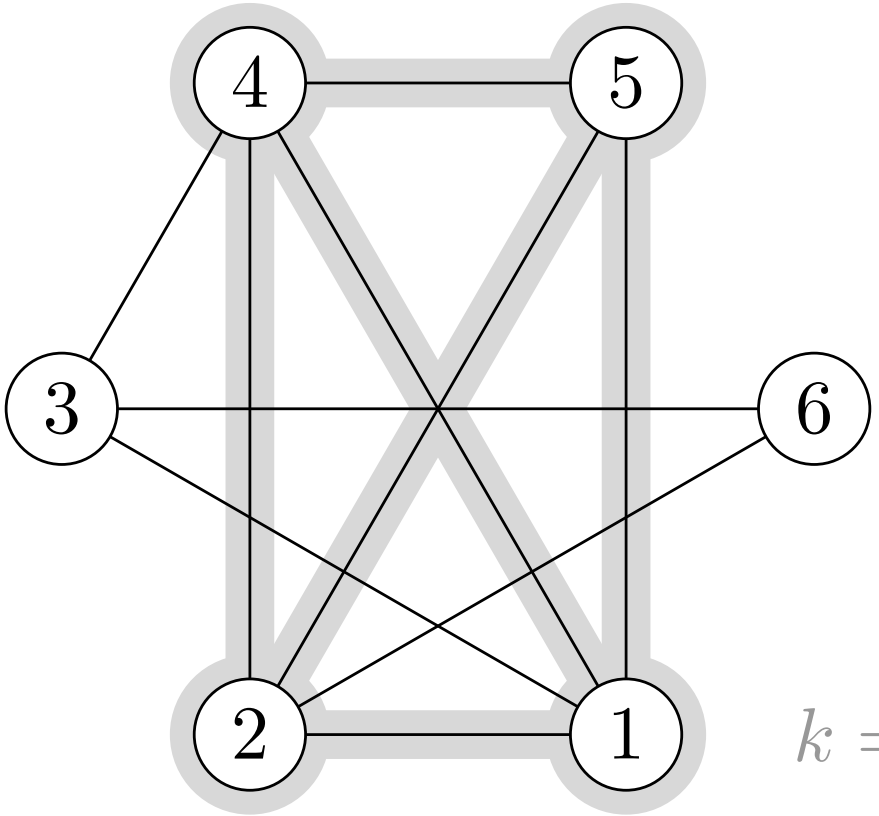
CLIQUE · **Verifikasjon**

Instans: En urettet graf G og et positivt heltall k

Sertifikat: En graf H

Sjekk: Er H en komplett delgraf med k noder?

mlh@ntnu.no

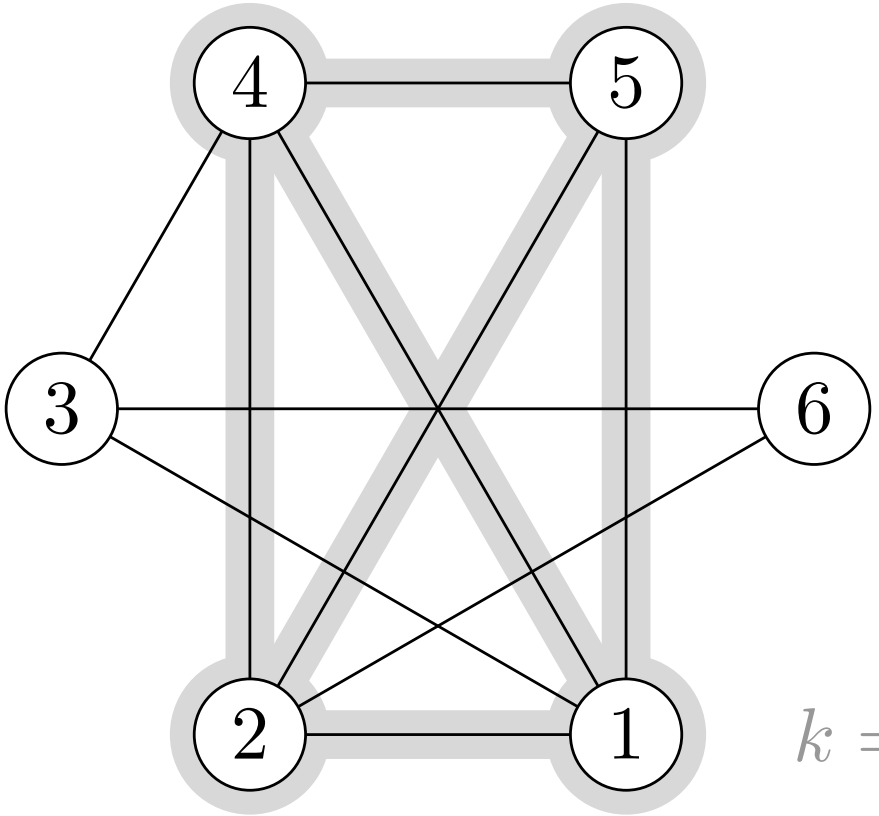


CLIQUE · **Søkeproblem**

Instans: En urettet graf G og et positivt heltall k

Oppgave: Finn en komplett delgraf med k noder

Vil feile hvis ingen løsning eksisterer!



CLIQUE · **Beslutningsproblem**

Instans: En urettet graf G og et positivt heltall k

Spørsmål: Har G en komplett delgraf med k noder?

Et «nei» tilsvarer at søkeproblemet feiler

Det er lett å verifisere at vi har funnet en k-klikk, men å faktisk finne en er svært vanskelig!

I hvert fall i praksis ... det er et åpent spørsmål om det faktisk er en forskjell!

2:3

Reduksjoner

Du skal velge ut flest mulig blant n personer slik at alle kjenner minst k av de andre.

Oppgave i, august 2010

mlh@ntnu.no

UNIQUE(A, n)

Vi vil avgjøre om elementene i A er forskjellige

UNIQUE(A, n)
1 INSERTION-SORT(A, n)

Vi reduserer ved å sortere . . .

```
UNIQUE(A, n)
1  INSERTION-SORT(A, n)
2  // TODO: Er naboverdier forskjellige?
```

. . . til å sjekke at alle naboer er forskjellige

```
UNIQUE(A, n)
1  INSERTION-SORT(A, n)
2  // TODO: Er naboverdier forskjellige?
```

Vi startet med hele unikhetsproblemet

mlh@ntnu.no

```
UNIQUE(A, n)
1  INSERTION-SORT(A, n)
2  // TODO: Er naboverdier forskjellige?
```

Vi har nå redusert til det som gjenstår på linje 2

mlh@ntnu.no

```
UNIQUE(A, n)
1  INSERTION-SORT(A, n)
2  // TODO: Er naboverdier forskjellige?
```

Om vi antar en algoritme for TODO-biten, så har vi løst problemet

mlh@ntnu.no

```
UNIQUE(A, n)
1  INSERTION-SORT(A, n)
2  // TODO: Er naboverdier forskjellige?
```

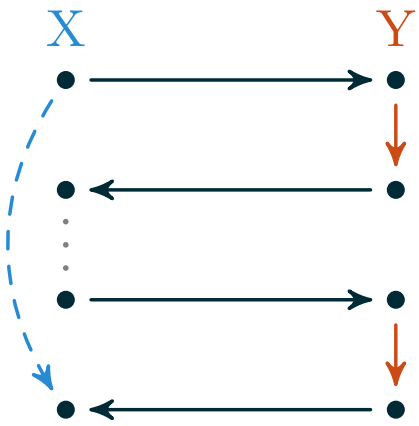
En slik hypotetisk subrutine kaller vi et orakel

SOLVE-X(*args*)

SOLVE-X(*args*)
1 do some work to reduce to Y

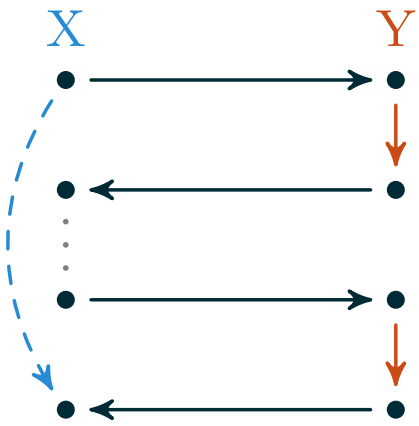
```
SOLVE-X(args)
1  do some work to reduce to Y
2  // TODO: Solve problem Y
```


mlh@ntnu.no



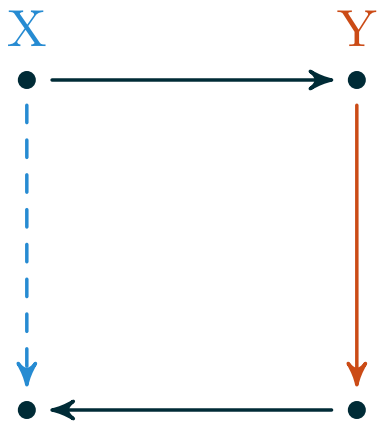
Cook

Turing, $O(n^k)$

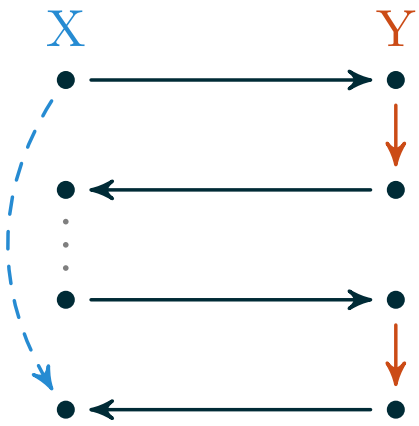


Cook

Turing, $O(n^k)$

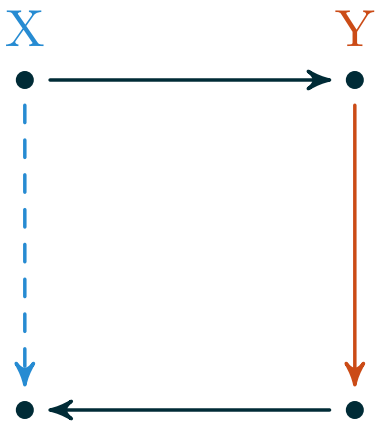


Levin

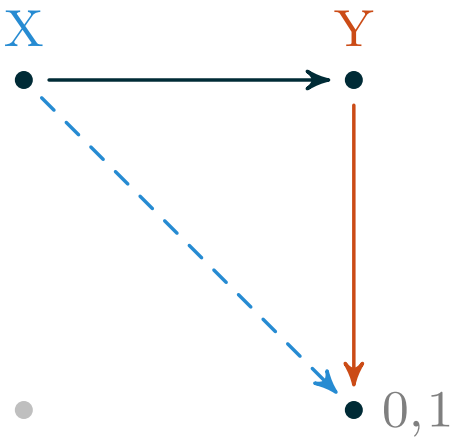


Cook

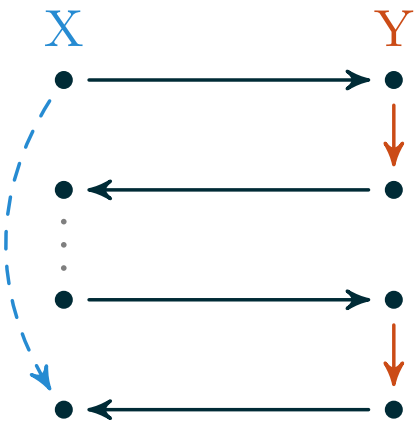
Turing, $O(n^k)$



Levin

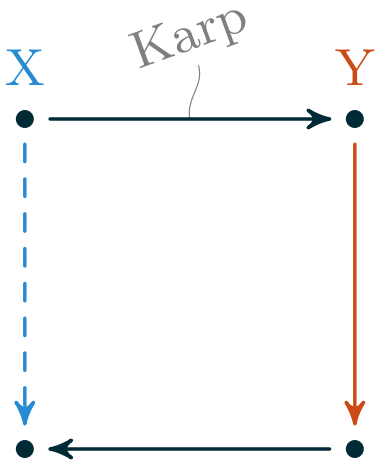


Karp

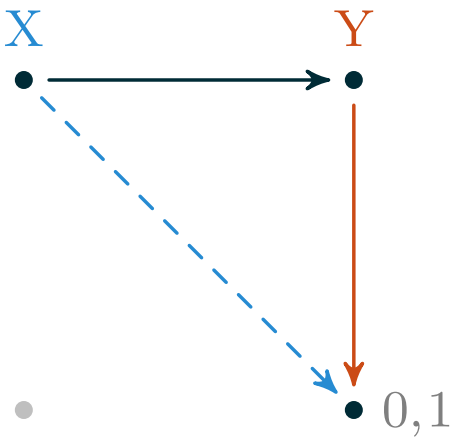


Cook

Turing, $O(n^k)$

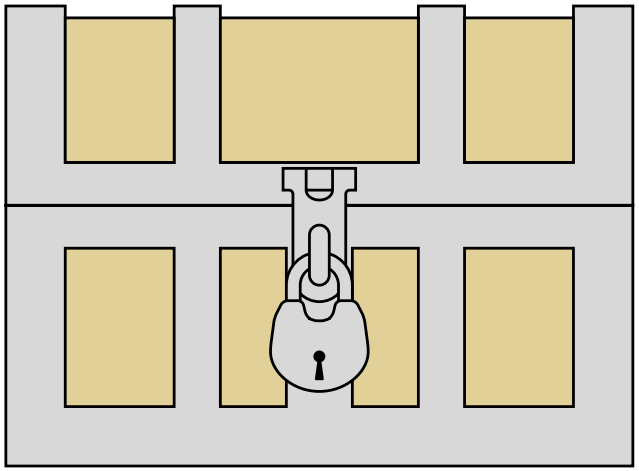


Levin



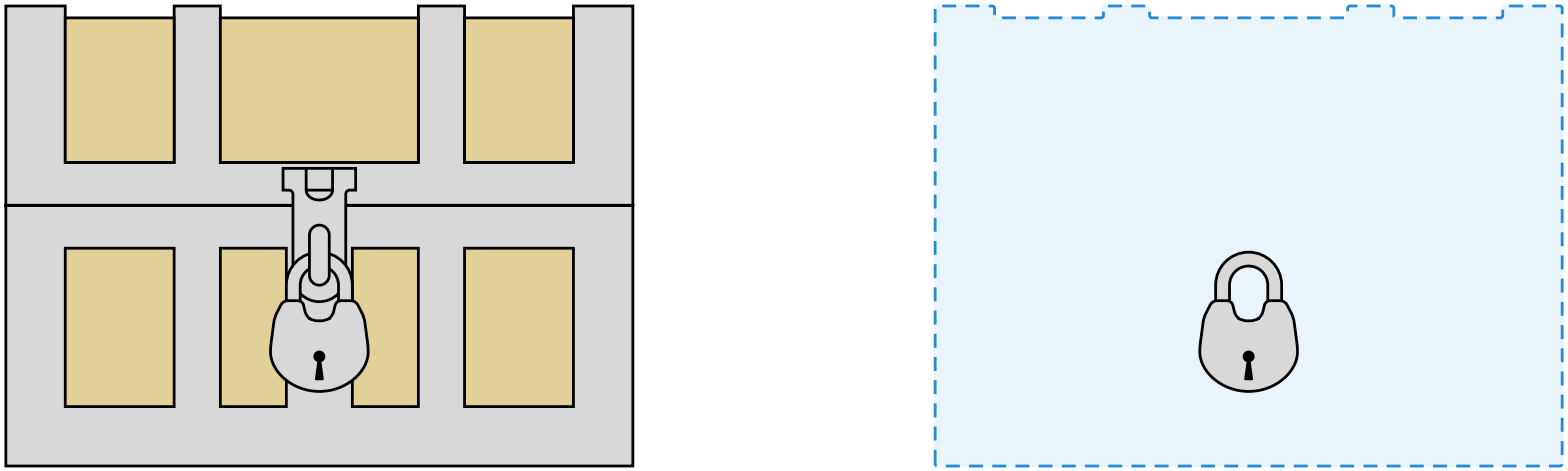
Karp

Reduksjoner og vanskegrad

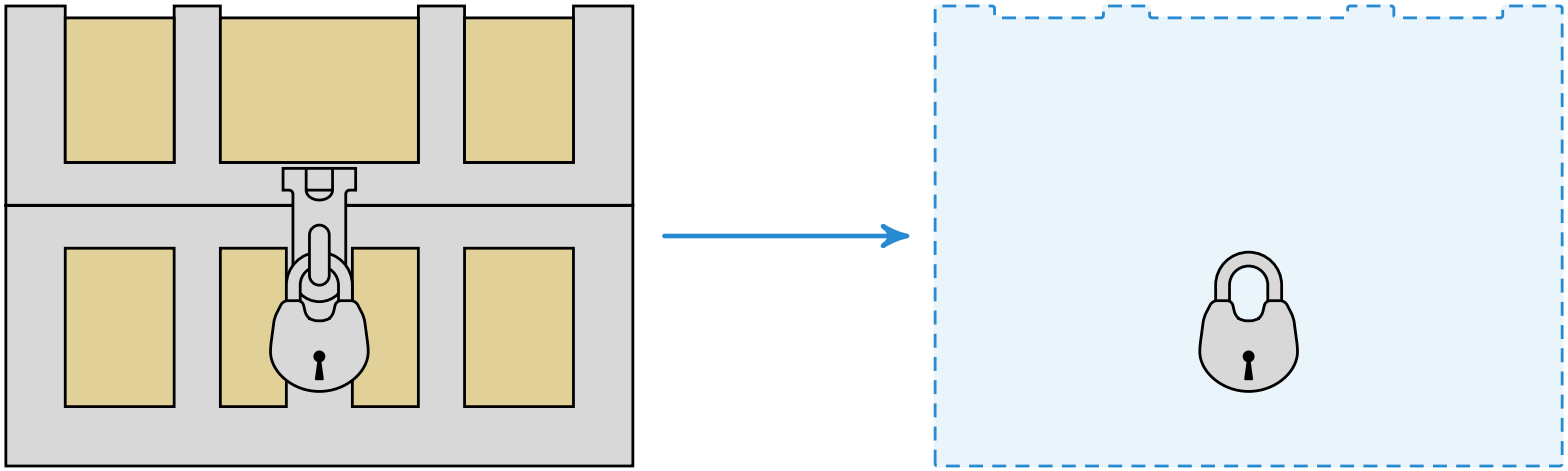


Vi vil åpne en skattekiste

mlh@ntnu.no

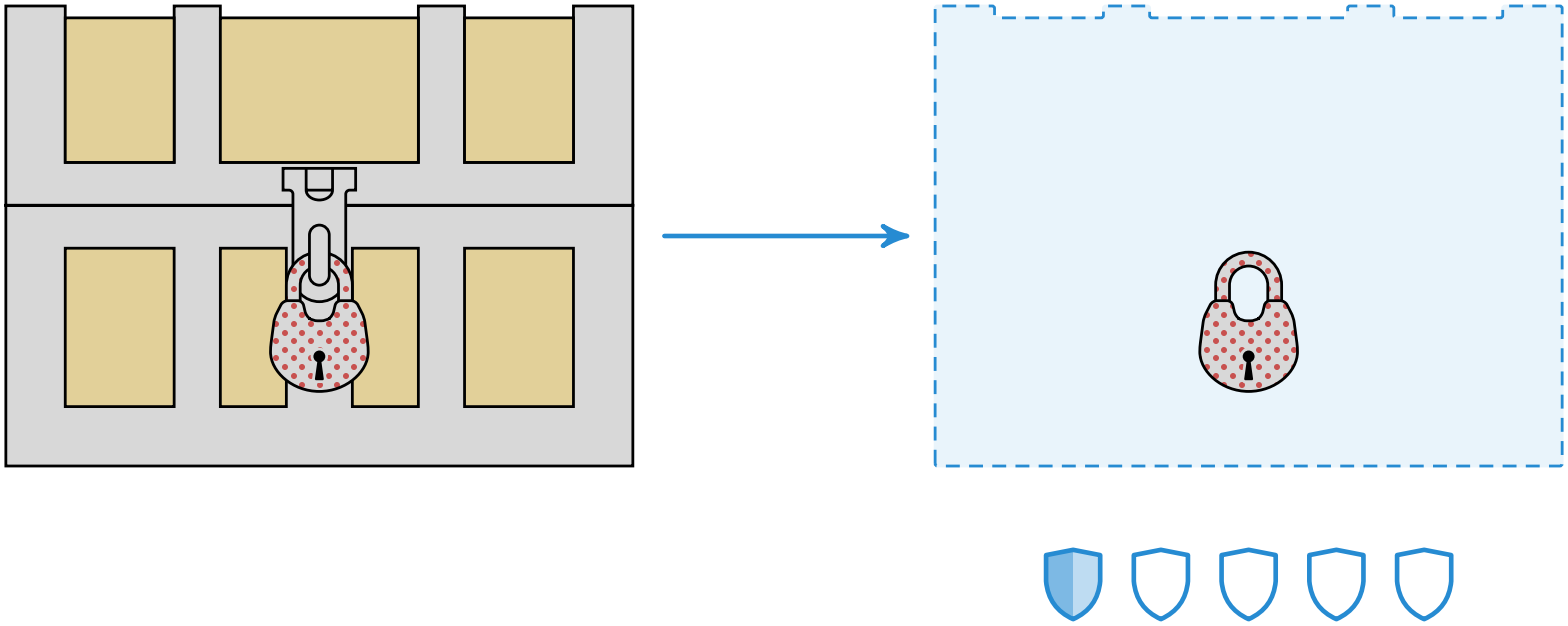


Én måte å gjøre det på: Få åpnet låsen

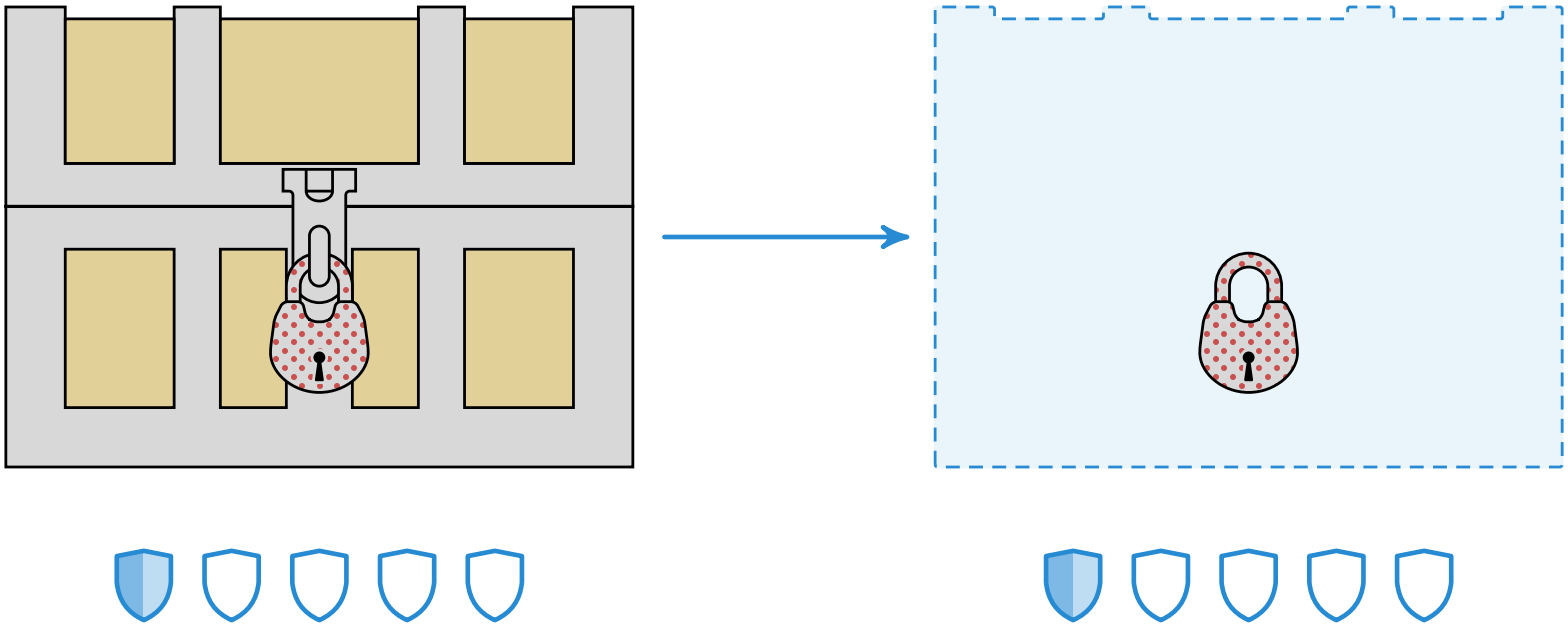


Å åpne kista reduserer til å åpne låsen

mlh@ntnu.no

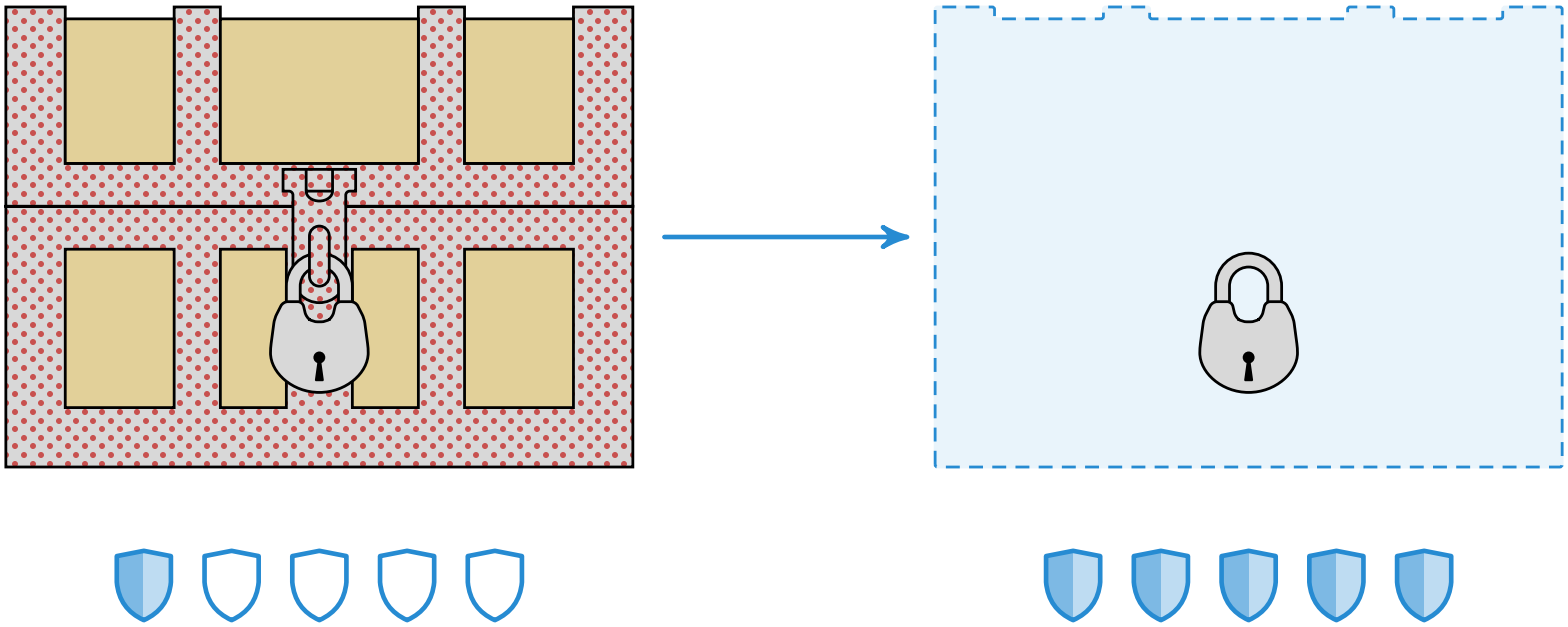


La oss si at låsen er rusten og lett å åpne

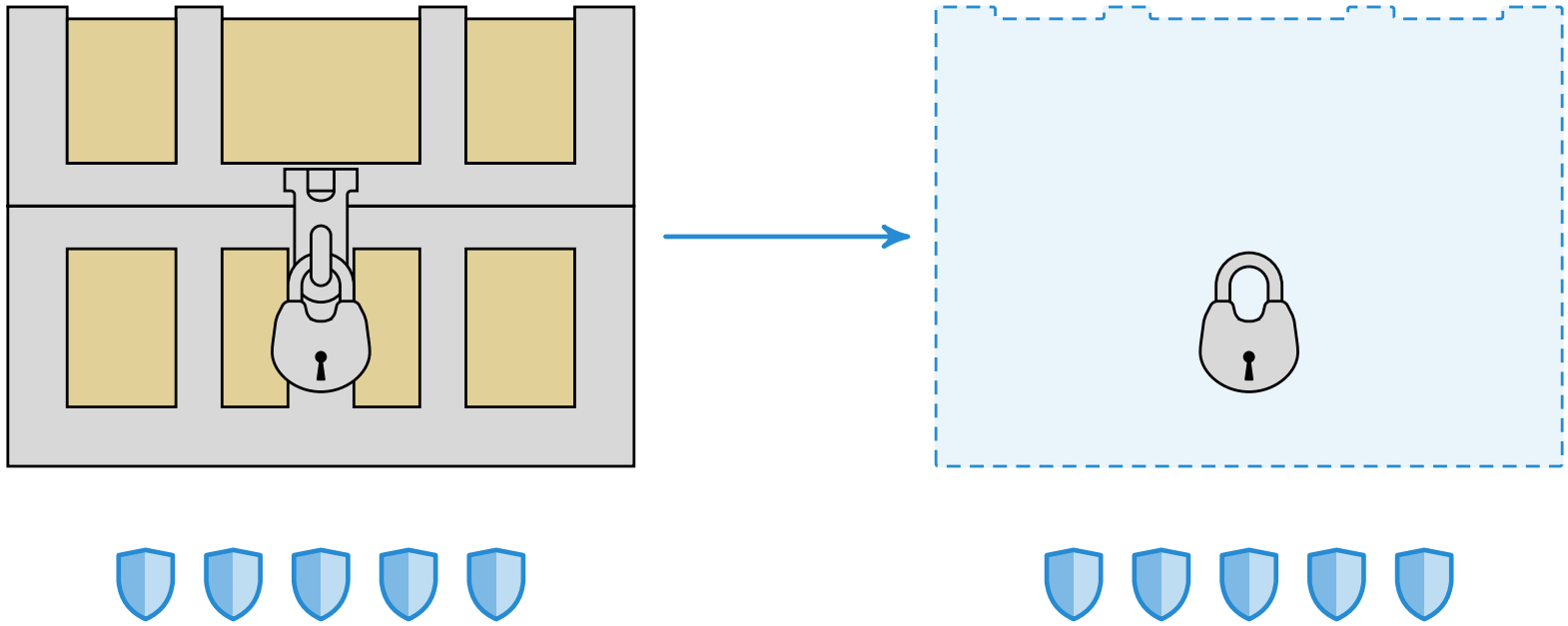


Da er det lett for oss å få opp kista

mlh@ntnu.no

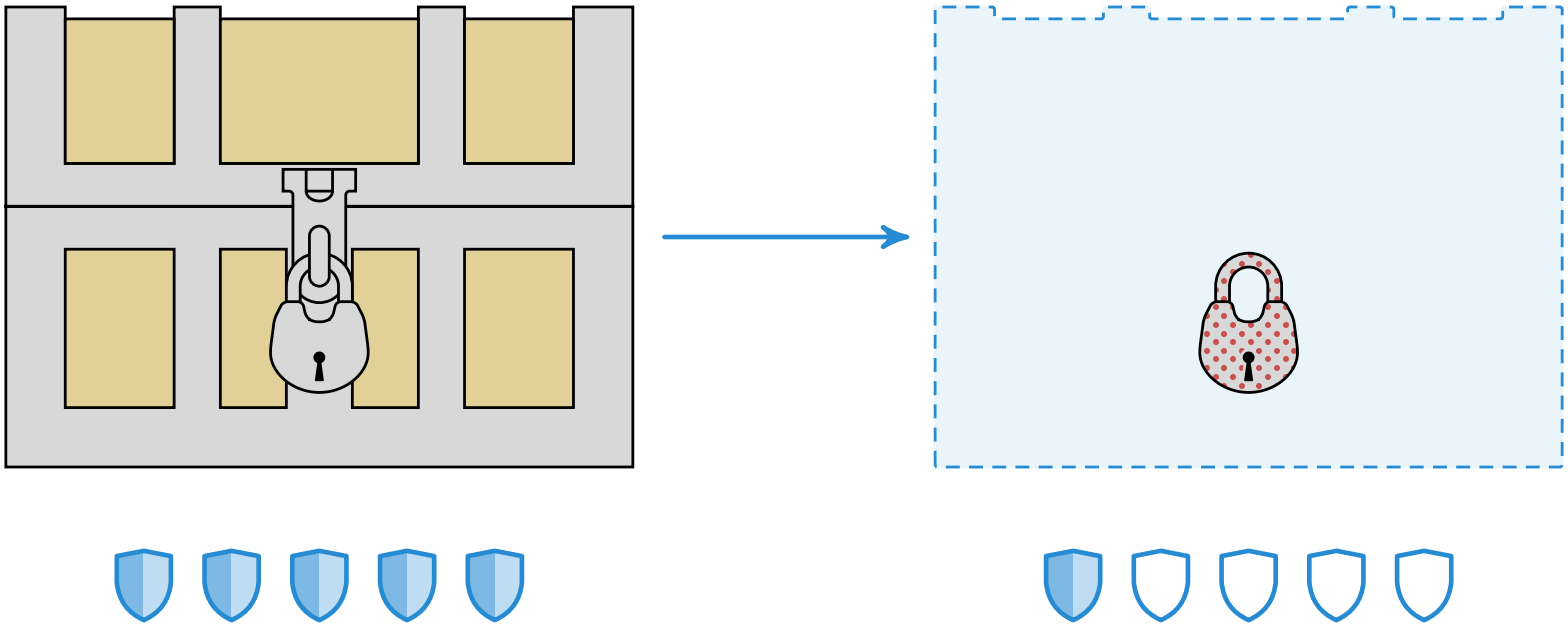


Men selv med en ny, solid lås kan det være lett å få opp kista!

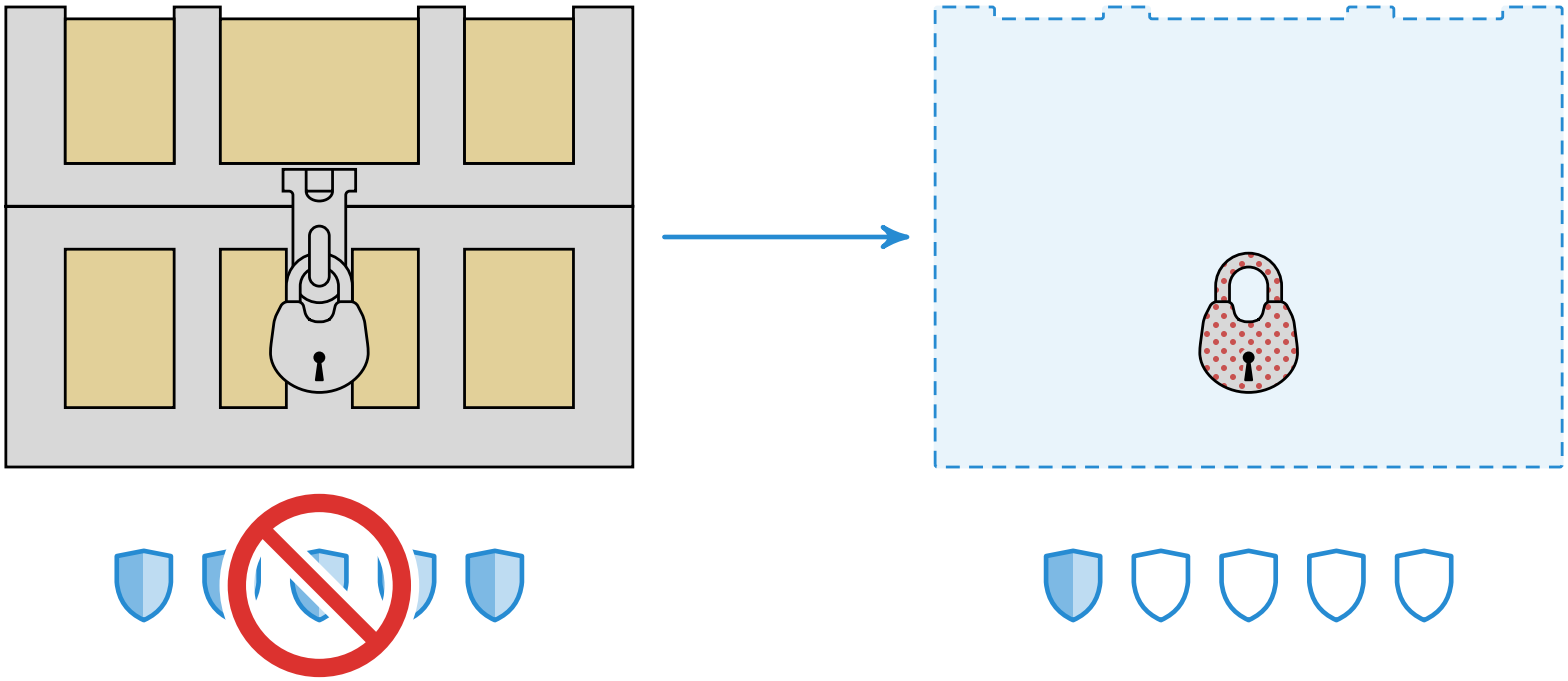


Dette er også et mulig scenario, naturligvis

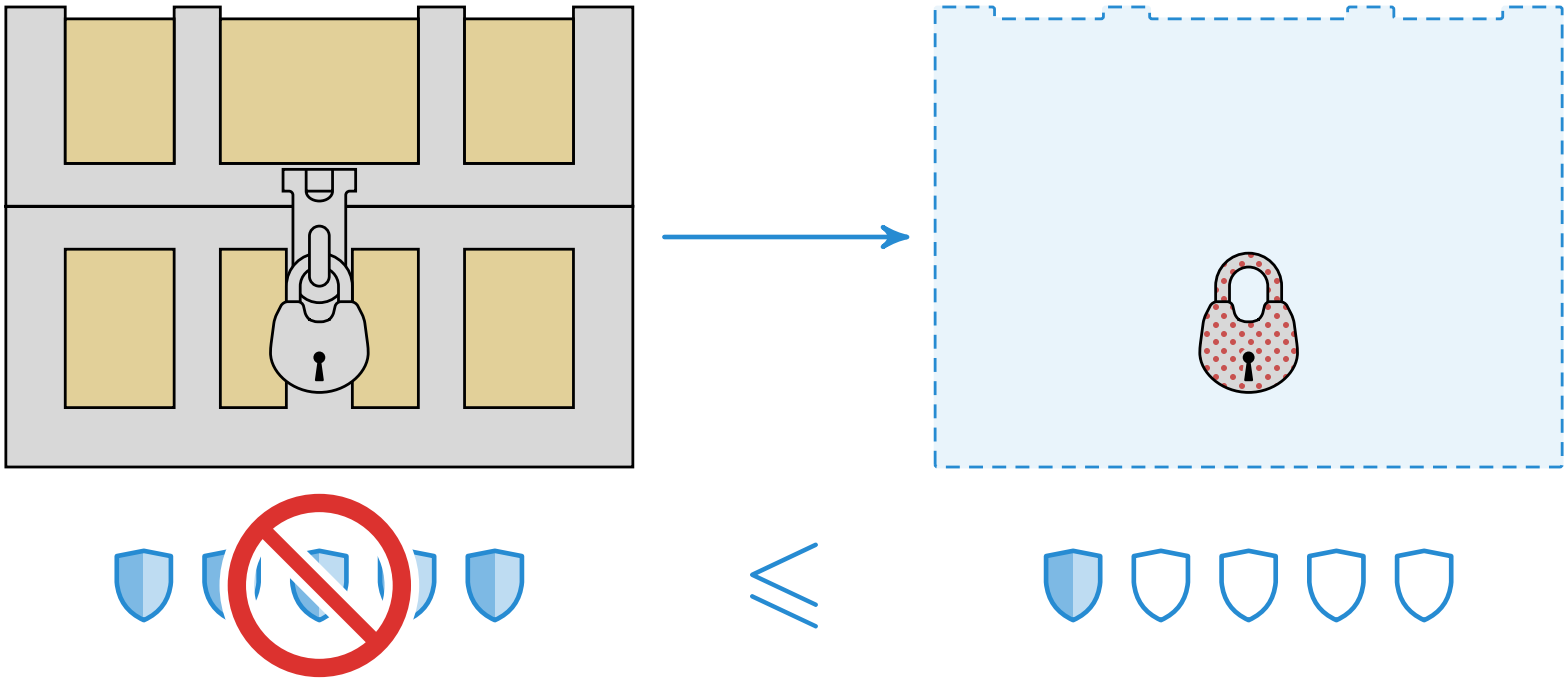
mlh@ntnu.no



Men hva med dette?

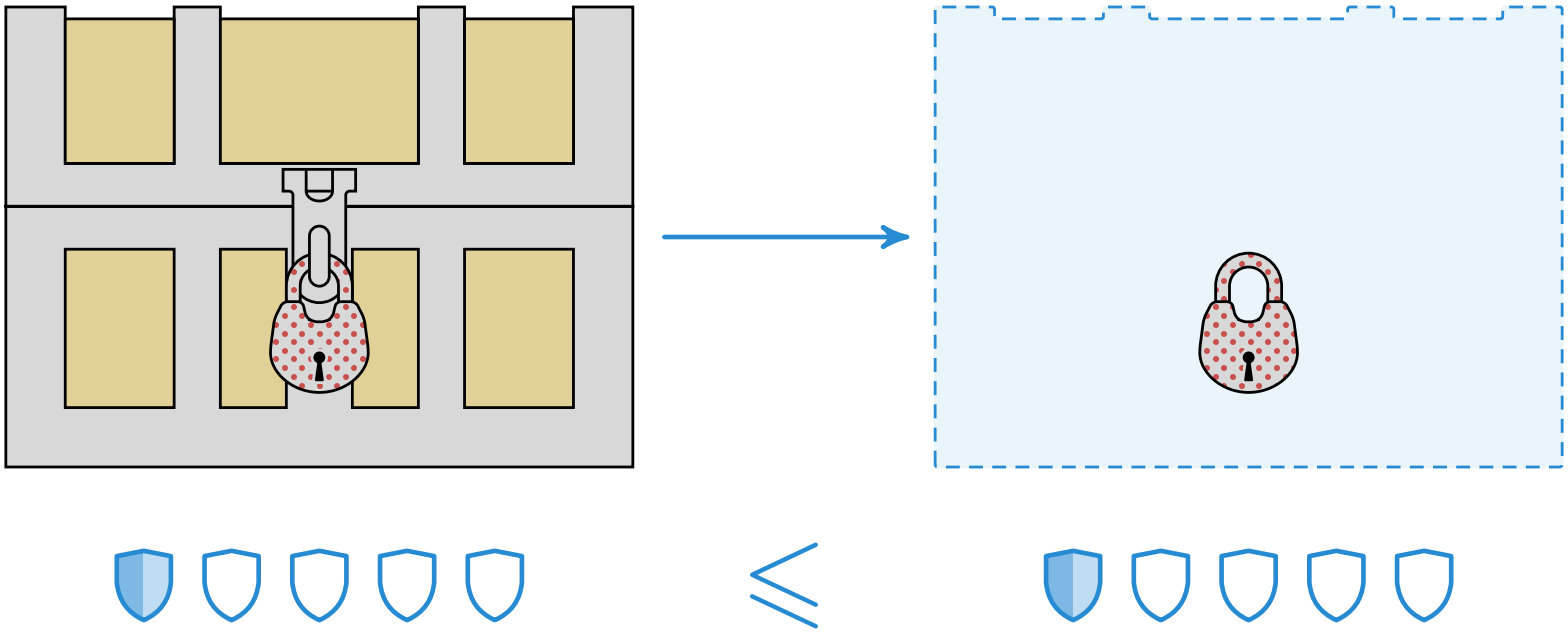


Det er naturligvis umulig!



Det kan ikke være lettere å få opp låsen enn kista

mlh@ntnu.no



Det kan ikke være lettere å få opp låsen enn kista

Poenget er: Man kan redusere til flere ulike problemer. Hvis ett av dem er enkelt, så har vi funnet en løsning

Hvis vi vet at det vi reduserer fra ikke kan ha en løsning, så kan ingen av dem vi reduserer til ha det heller

Tungvint

```
1  transformer instansen
2  // TODO: Noe vanskelig
```

Lettvint

```
1  transformer instansen
2  // TODO: Noe enkelt
```


mlh@ntnu.no

LØS-ENKELT-PROBLEM
1 enkel reduksjon
2 // TODO: Fritt valg

mlh@ntnu.no

Løs-Vanskelig-Problem

1 enkel reduksjon

2 // TODO: Alltid vanskelig!

mlh@ntnu.no

Du skal velge ut færrest mulig blant n personer slik at alle kjenner minst k av de andre.

Oppgave 4a, august 2011

Kan vi bruke reduksjons-tankegangen til å lage mer skreddersydde algoritmer, som Insertion-Sort?

«Andre regel er å dele opp alle kompliserte saksforhold i så mange deler som det er mulig og nødvendig.

Tredje regel er å tenke i en bestemt rekkefølge, slik at man begynner med de enkleste og lettest kjennelige objektene og fortsetter trinnvis til erkjennelse av de mer sammensatte.»

Thomas av Boueberge oppsummerer Descartes' metode

mlh@ntnu.no

«Andre regel er å dele opp alle kompliserte saksforhold i så mange deler som det er mulig og nødvendig.

Tredje regel er å tenke i en bestemt rekkefølge, slik at man begynner med de enkleste og lettest kjennelige objektene og fortsetter trinnvis til erkjennelse av de mer sammensatte.»

(*Vredens dag* av Kurt Aust, 1999)

3:3

Dekomponering

Kjerneprinsipp

- **Bryt ned problemet så det kan løses trinn for trinn**
- **Fokuser på ett (representativt) trinn**

Rekursjon … og

Matematisk induksjon

Altså ikke *filosofisk* induksjon, f.eks.

Vi sier stort sett bare «induksjon».

En rekursiv prosedyre kaller seg selv.

Evt.: Den er definert vha. seg selv.

Evt.: Den bruker seg selv som subrutine.

Om du er litt rusten på rekursjon:

Lurt å børste støv av kunnskapene!

- **Del opp i mindre problemer**
- **Induksjonshypotese: Anta at du kan løse de mindre problemene**
- **Induksjonssteg: Konstruer fullstendig løsning ut fra del-løsningene**

Kalles også «induktivt premiss». Ordet «premiss» passer godt til algoritmedesign, siden det også kan bety «betingelse» – og vi beskriver her betingelser for at trinnet vårt skal bli korrekt.

Også kalt induktsjonstrinn.

Vi må også sørge for at ting terminerer – at ting blir rett når vi kommer til grunntilfellet (base case) i rekursjonen/ induksjonen.

$$\frac{\mathrm{P}(a)}{\forall x\,\mathrm{P}(x)}$$

∀-introduksjon: Vis P for vilkårlig a

mlh@ntnu.no

$$\frac{\mathrm{P}(a)}{\forall x\, \mathrm{P}(x)}$$

$\forall$-introduksjon: Vis P for vilkårlig a

mlh@ntnu.no

$$\dfrac{\begin{array}{c}P\\ \vdots\\ Q\end{array}}{P \Rightarrow Q}\ \not{H}$$

H

⇒-introduksjon: Midlertidig anta P og vis deretter Q

mlh@ntnu.no

$$\frac{P \Rightarrow Q, \quad P}{Q}$$

⇒-eliminasjon: Modus ponens

mlh@ntnu.no

vilkårlig

$$\frac{\mathrm{P}(a)}{\forall x\,\mathrm{P}(x)}$$

$$\frac{\begin{array}{c}\mathrm{P}\qquad \mathrm{H}\\ \vdots\\ \mathrm{Q}\end{array}}{\mathrm{P}\Rightarrow\mathrm{Q}}\ \cancel{\mathrm{H}}$$

$$\frac{\mathrm{P}\Rightarrow\mathrm{Q},\ \ \mathrm{P}}{\mathrm{Q}}$$

Induksjon kombinerer disse

mlh@ntnu.no

$$\frac{\mathrm{P}(a)}{\forall x\,\mathrm{P}(x)}$$

vilkårlig

$$\frac{\begin{array}{c}\mathrm{P}\\ \vdots\\ \mathrm{Q}\end{array}}{\mathrm{P}\Rightarrow\mathrm{Q}}\;\cancel{\mathrm{H}}$$

H

$$\frac{\mathrm{P}\Rightarrow\mathrm{Q},\;\;\mathrm{P}}{\mathrm{Q}}$$

Vi introduserer og eliminerer mange implikasjoner

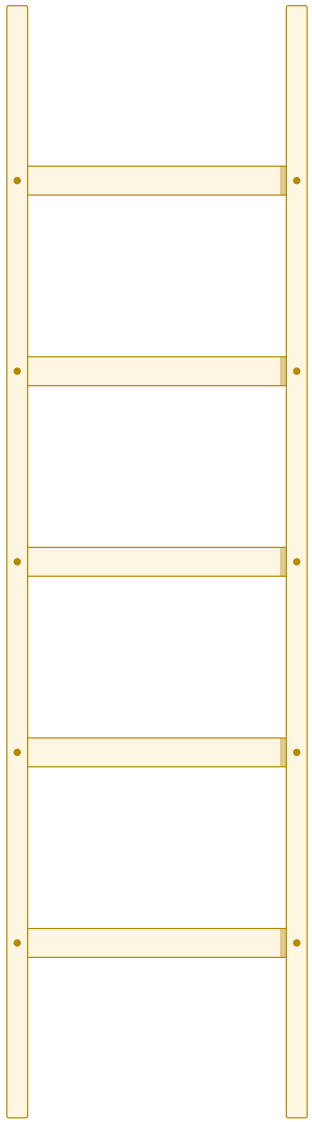
mlh@ntnu.no

Generell induksjon kan foregå i et nettverk av utsagn . . .

mlh@ntnu.no

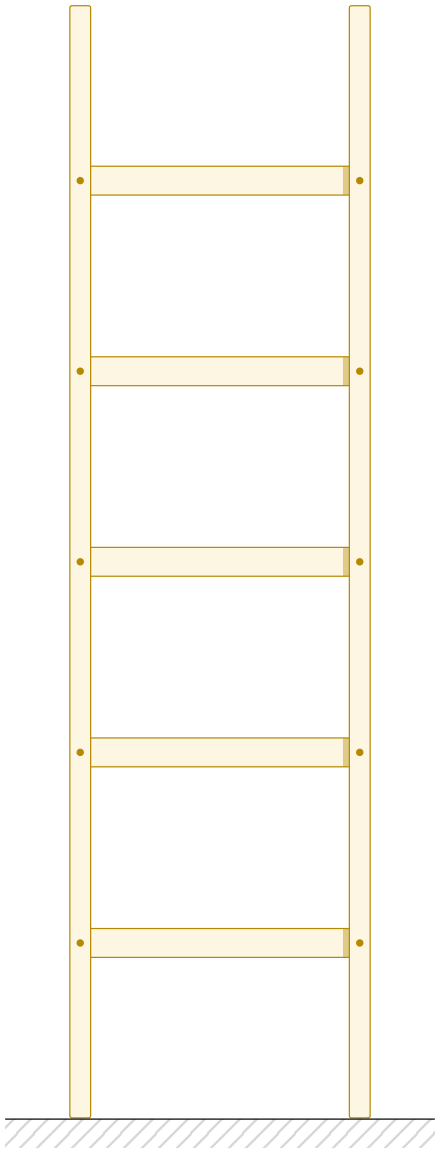
. . . men vi kan alltid ordne dem i en serie med trinn

mlh@ntnu.no



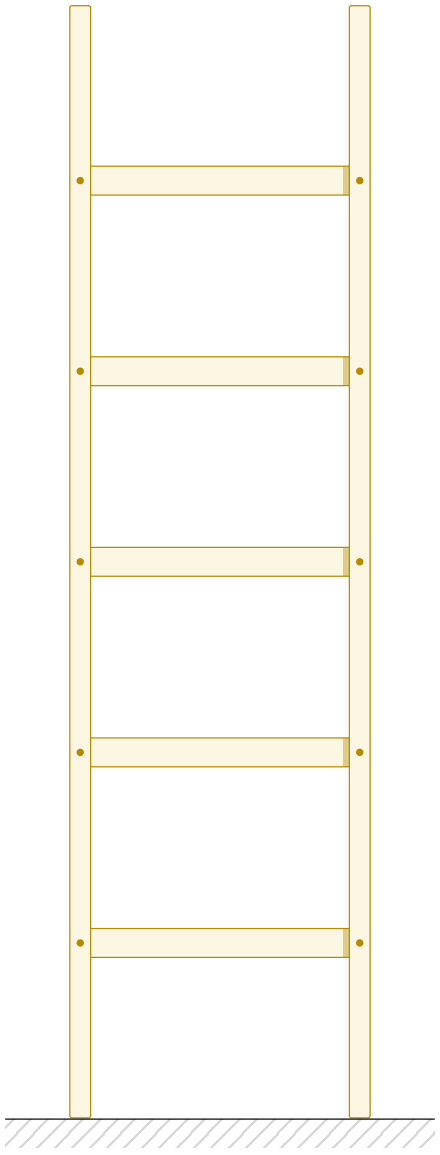
. . . men vi kan alltid ordne dem i en serie med trinn

mlh@ntnu.no



Ett eller flere tilfeller baserer seg ikke på noen andre

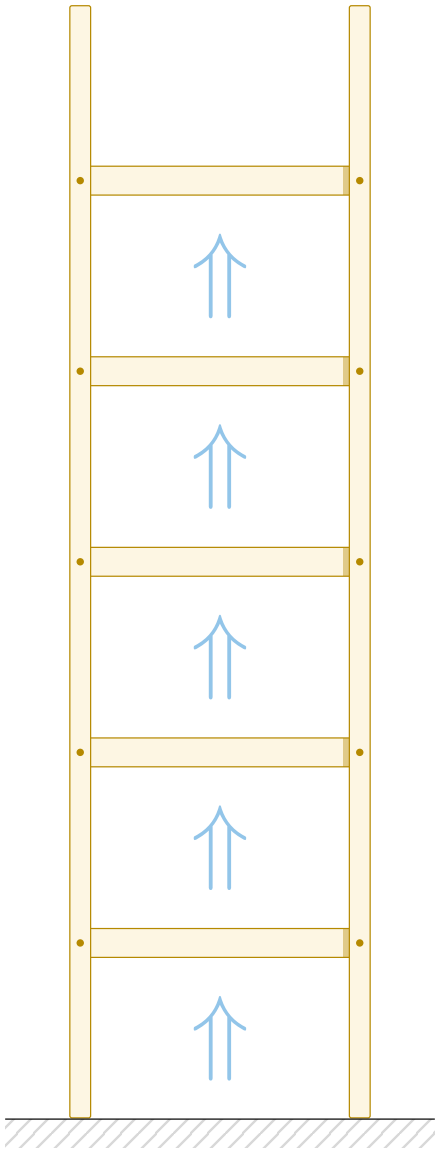
mlh@ntnu.no



Grunntilfelle

Ett eller flere tilfeller baserer seg ikke på noen andre

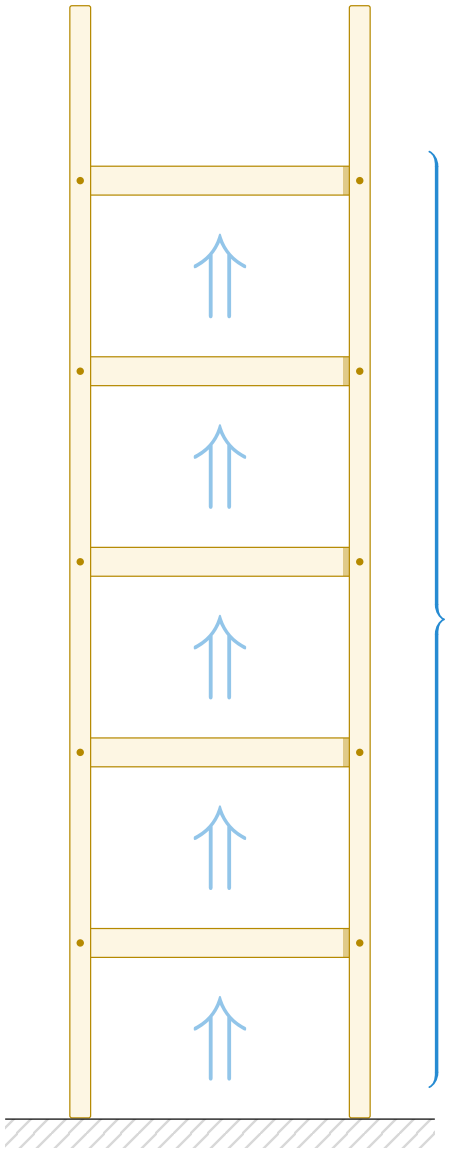
mlh@ntnu.no



Grunntilfelle

De andre er induktive: De følger av tidligere trinn

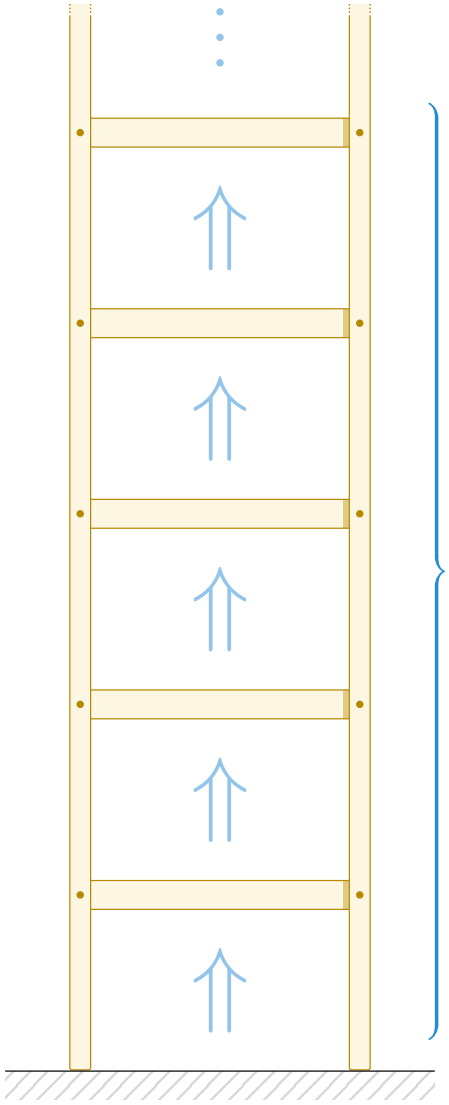
mlh@ntnu.no



Induktive trinn

Grunntilfelle

De andre er induktive: De følger av tidligere trinn



Induktive trinn

Grunntilfelle

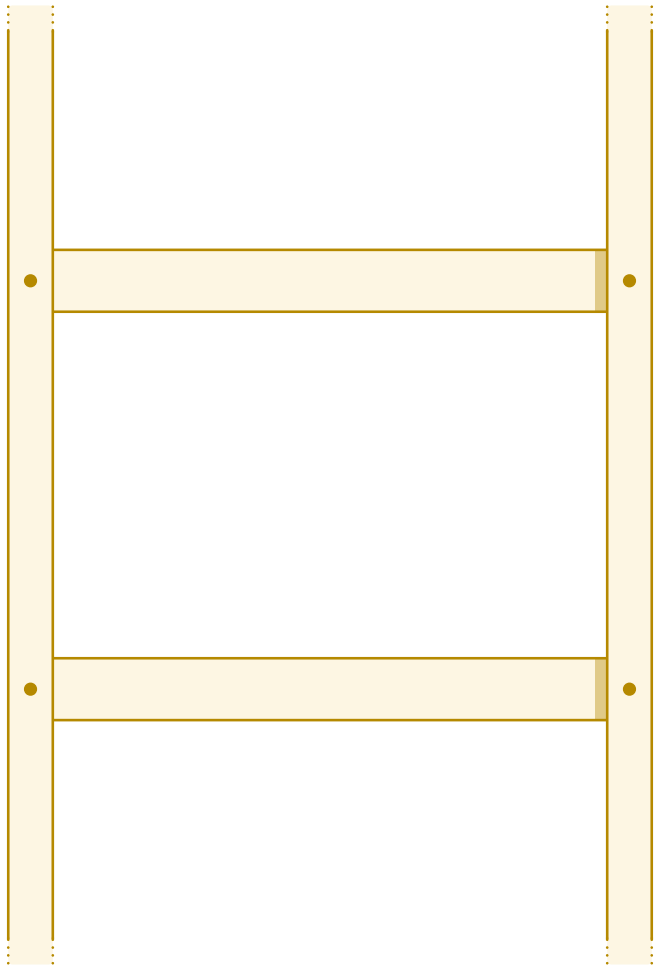
mlh@ntnu.no

Grunntilfellet (ev. grunntilfellene) viser vi for seg

mlh@ntnu.no

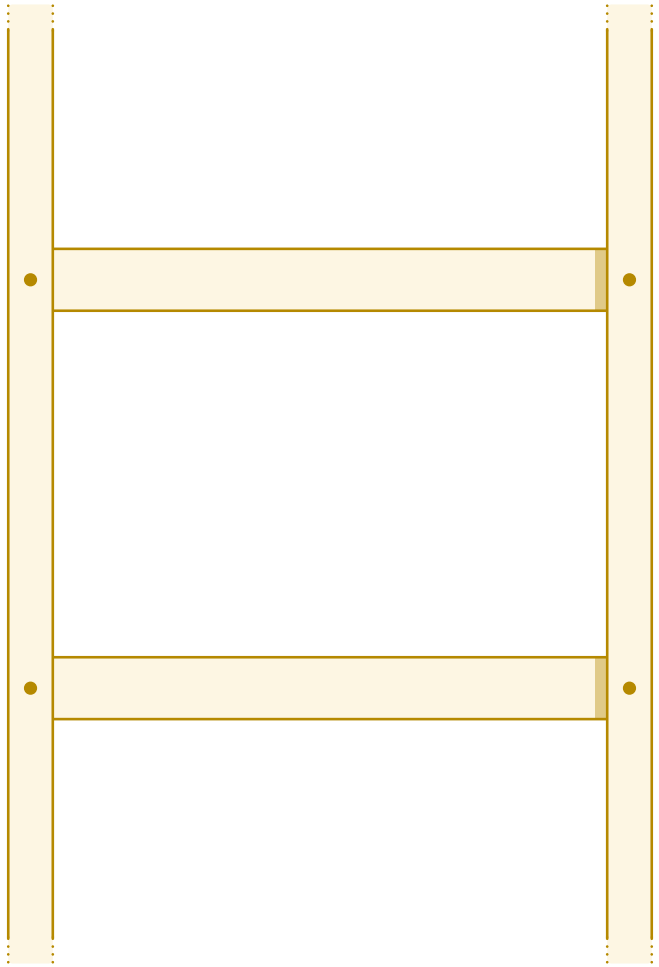
(Det er ofte svært enkelt)

mlh@ntnu.no



Vil beskrive alle induktive trinn

mlh@ntnu.no

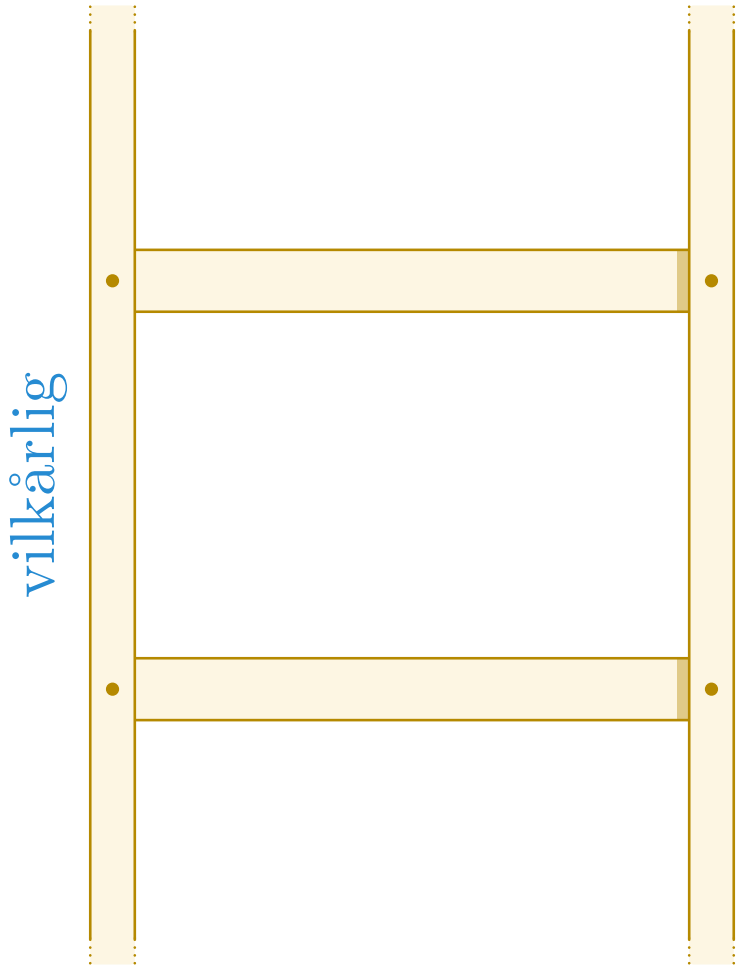


Vil beskrive alle induktive trinn

vilkårlig

$$\frac{\mathrm{P}(a)}{\forall x\,\mathrm{P}(x)}$$

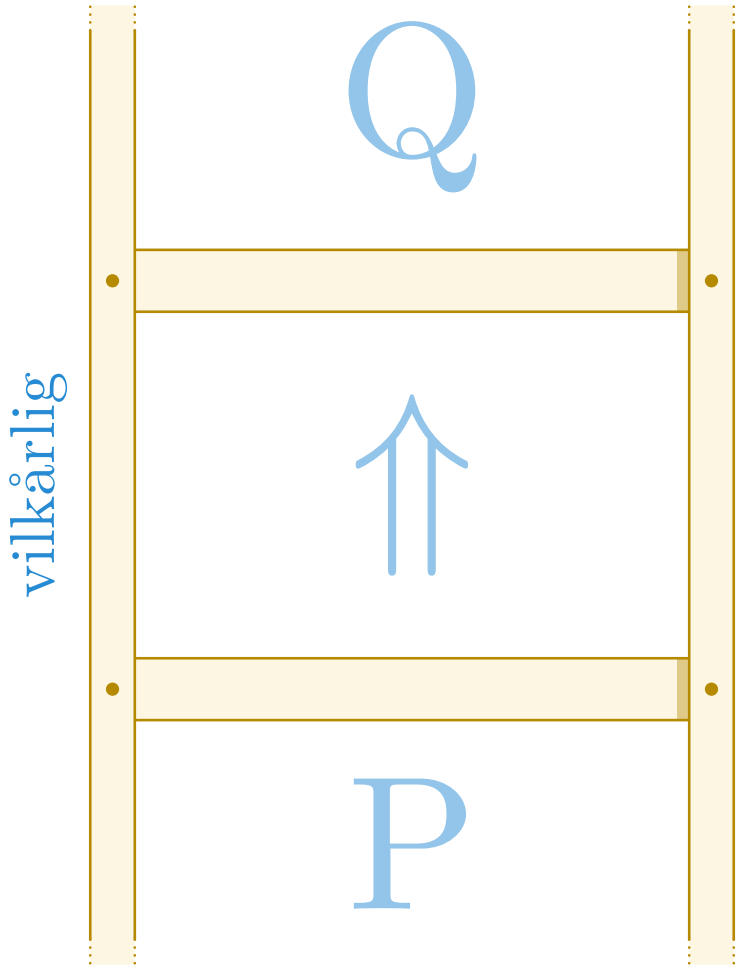
Så vi ser på et vilkårlig et



vilkårlig

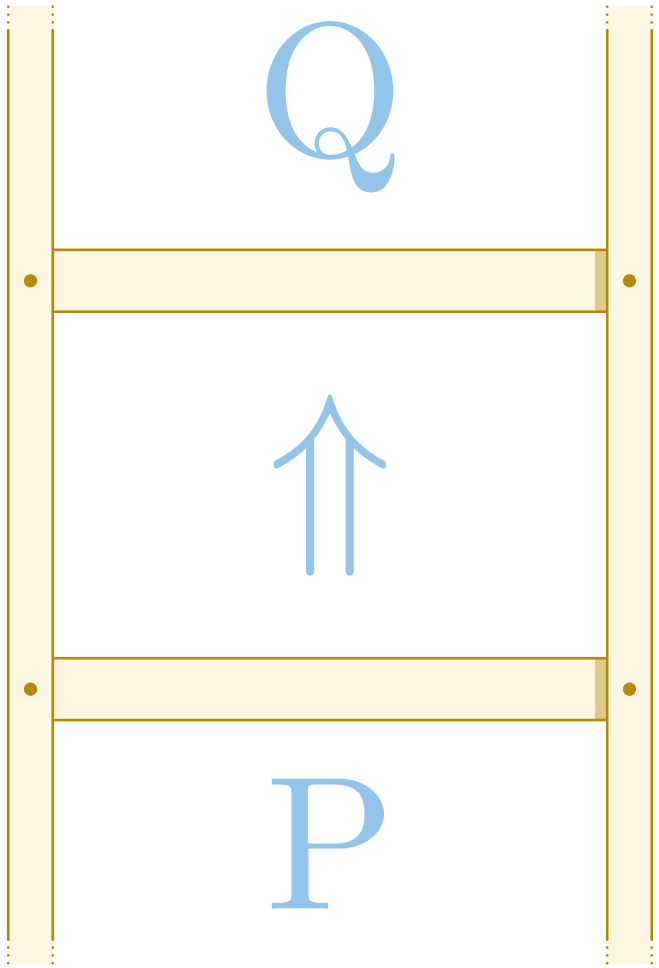
$$\frac{\mathrm{P}(a)}{\forall x\,\mathrm{P}(x)}$$

mlh@ntnu.no



Vil vise implikasjon

mlh@ntnu.no

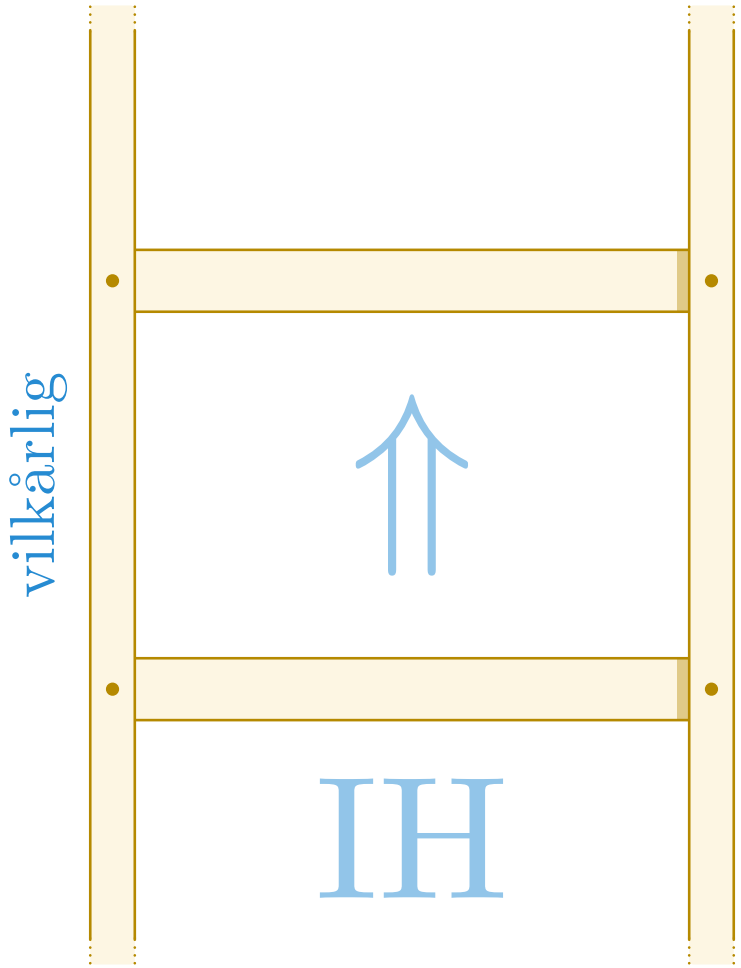


Vil vise implikasjon

$$\dfrac{\begin{array}{c}P \qquad H\\ \vdots\\ Q\end{array}}{P \Rightarrow Q}\ \cancel{H}$$

Anta forrige trinn

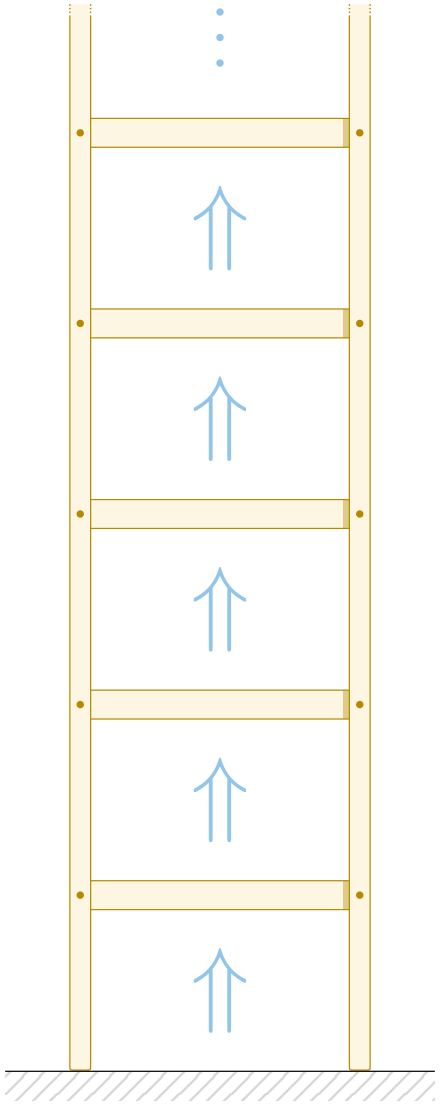
mlh@ntnu.no



$$\dfrac{\begin{array}{c}P\\ \vdots\\ Q\end{array}}{P \Rightarrow Q}\ \cancel{H}\qquad H$$

P = Induksjonshypotesen

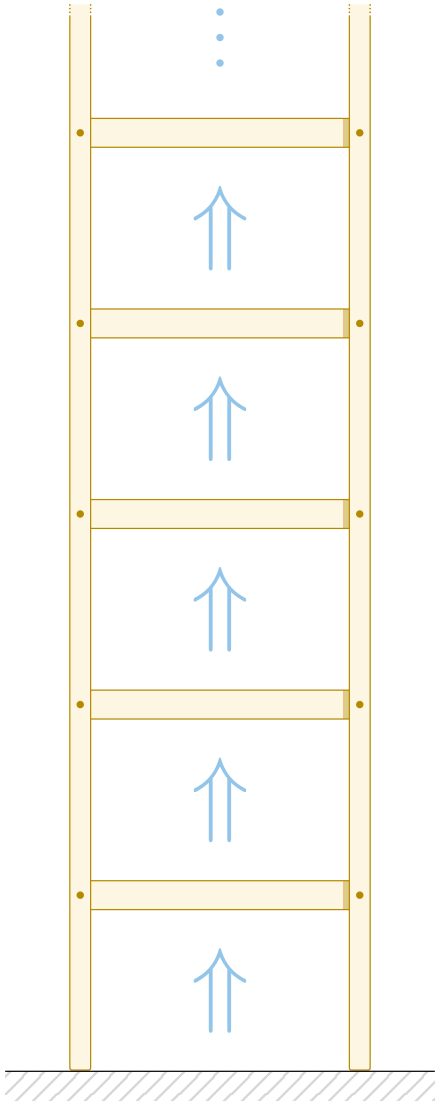
mlh@ntnu.no



rett

Vi vet at grunntilfellet stemmer

mlh@ntnu.no



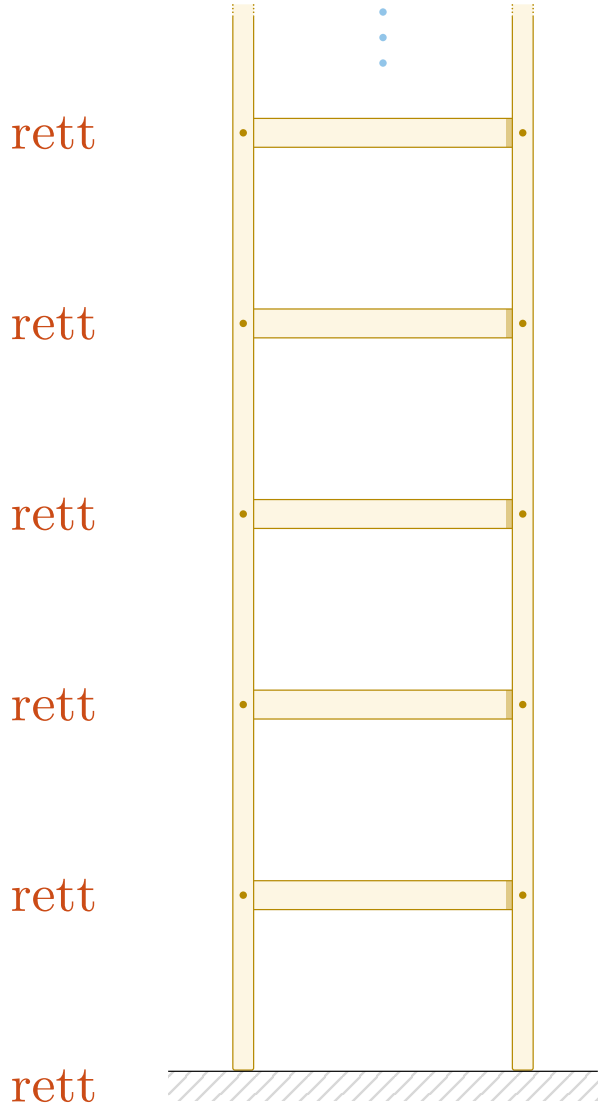
rett

Vi vet at grunntilfellet stemmer

$$\frac{P \Rightarrow Q, \quad P}{Q}$$

Dette «smitter» til alle de andre

mlh@ntnu.no



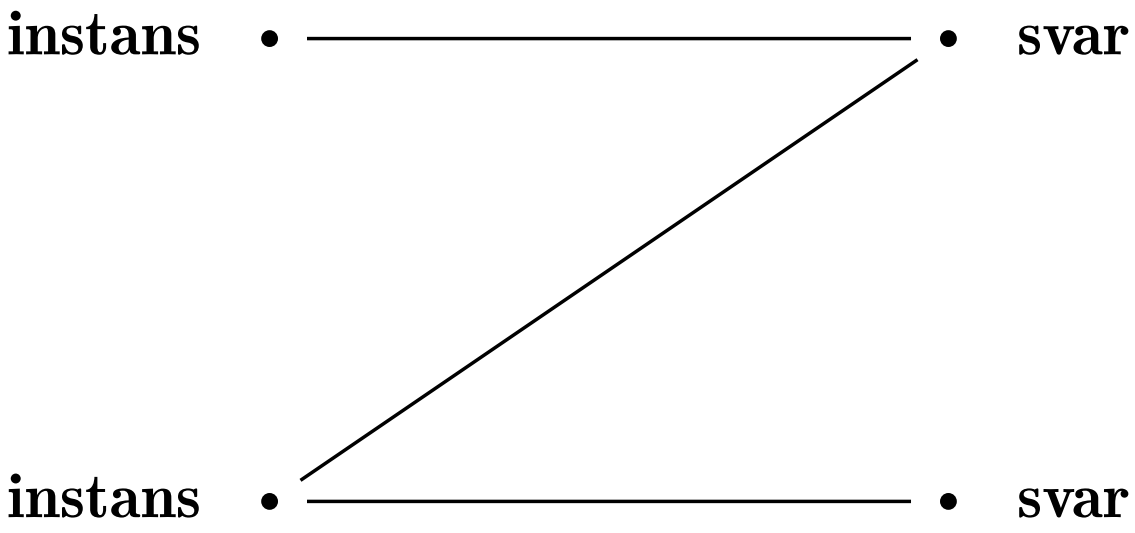
Vi vet at grunntilfellet stemmer

$$\frac{P \Rightarrow Q, \quad P}{Q}$$

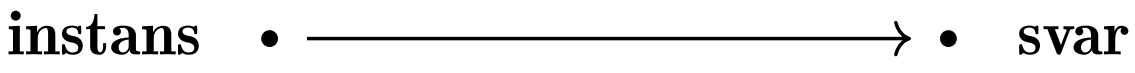
Dette «smitter» til alle de andre

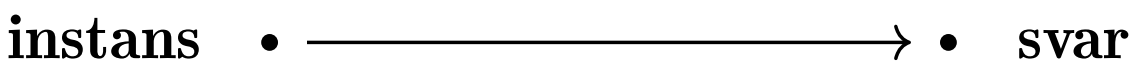
mlh@ntnu.no

Problem



Problem





En algoritme finner ett riktig svar for hver instans

Problem

Algoritmen må løse alle instanser. Vi ser på en vilkårlig instans

mlh@ntnu.no

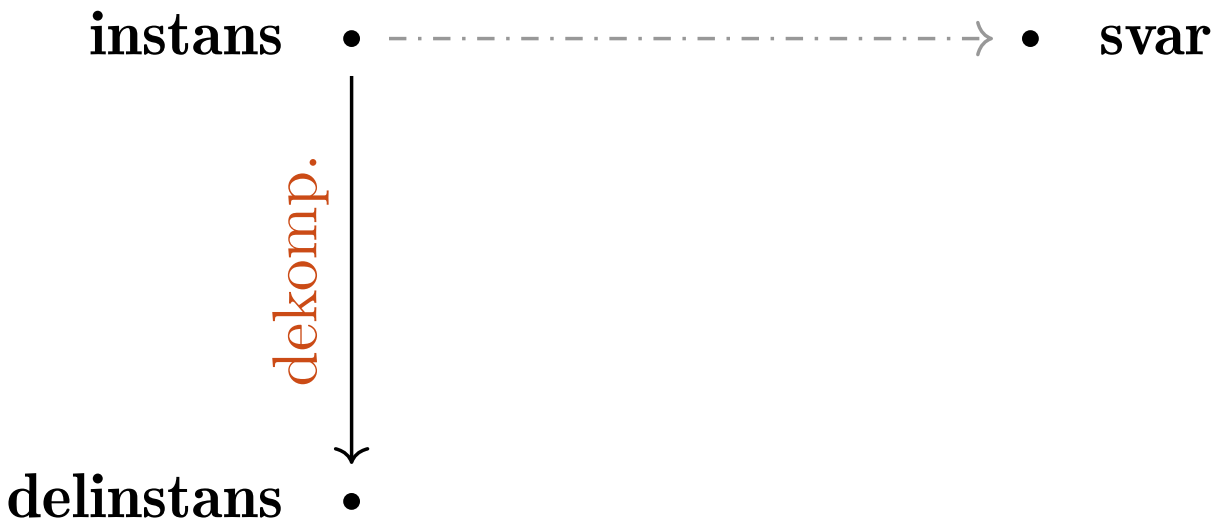
Problem

Vi vet ikke ennå hvordan vi løser instansen

Problem

Vi dekomponerer instansen i én eller flere delinstanser

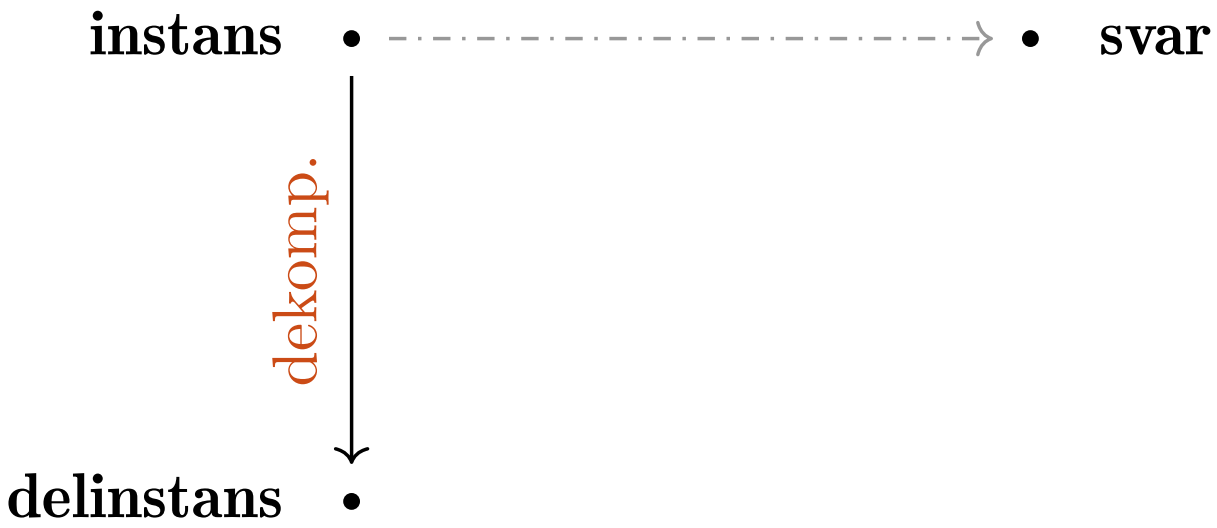
Problem



Vi dekomponerer instansen i én eller flere delinstanser

mlh@ntnu.no

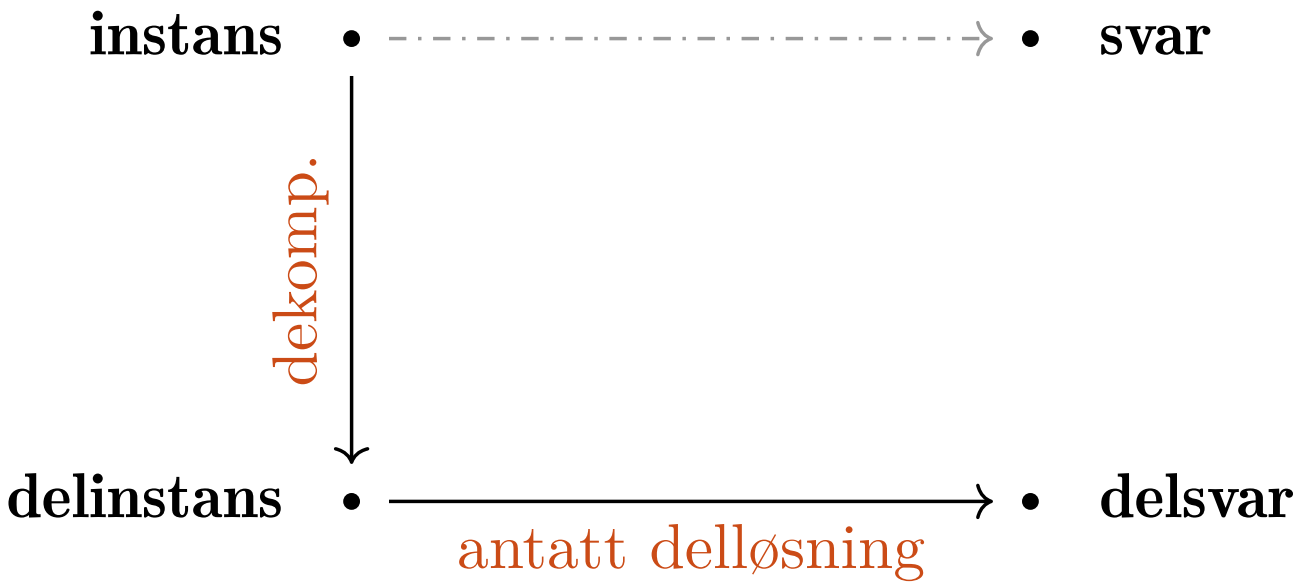
Problem



Vi antar at vi kan løse delinstansene

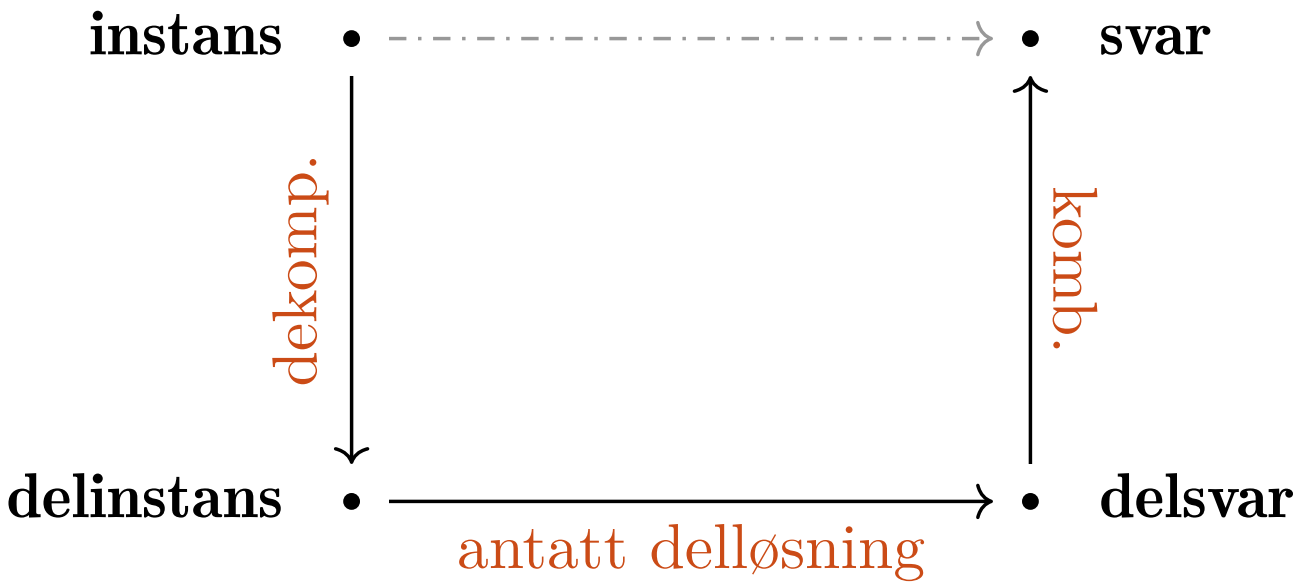
mlh@ntnu.no

Problem



Vi antar at vi kan løse delinstansene

Problem

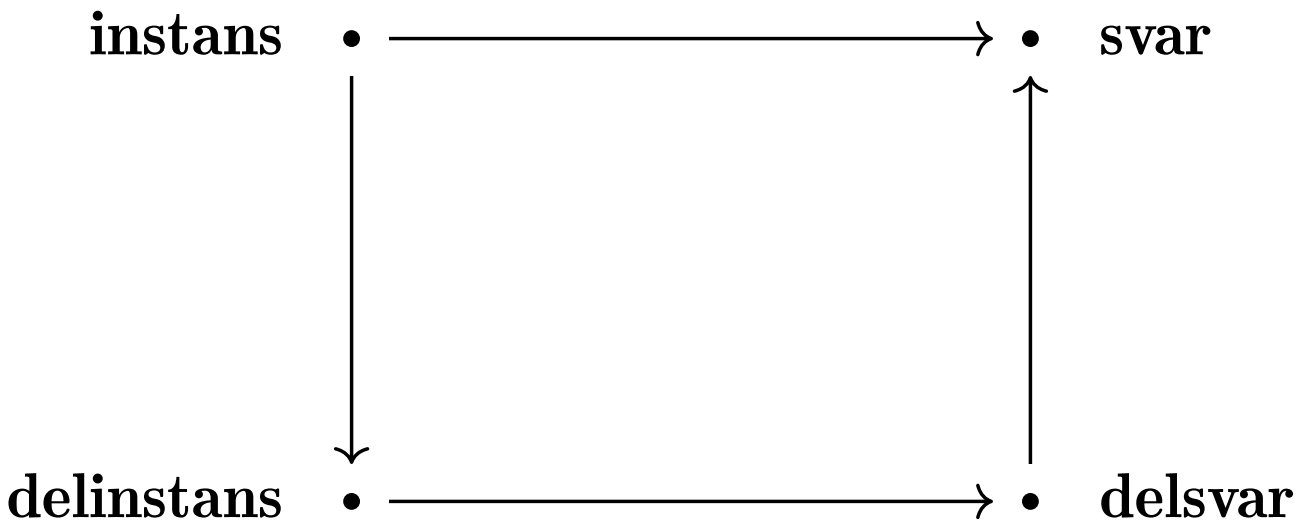


Vi kombinerer så delsvar til et endelig svar

Problem

Om vi setter sammen disse får vi en løsning!

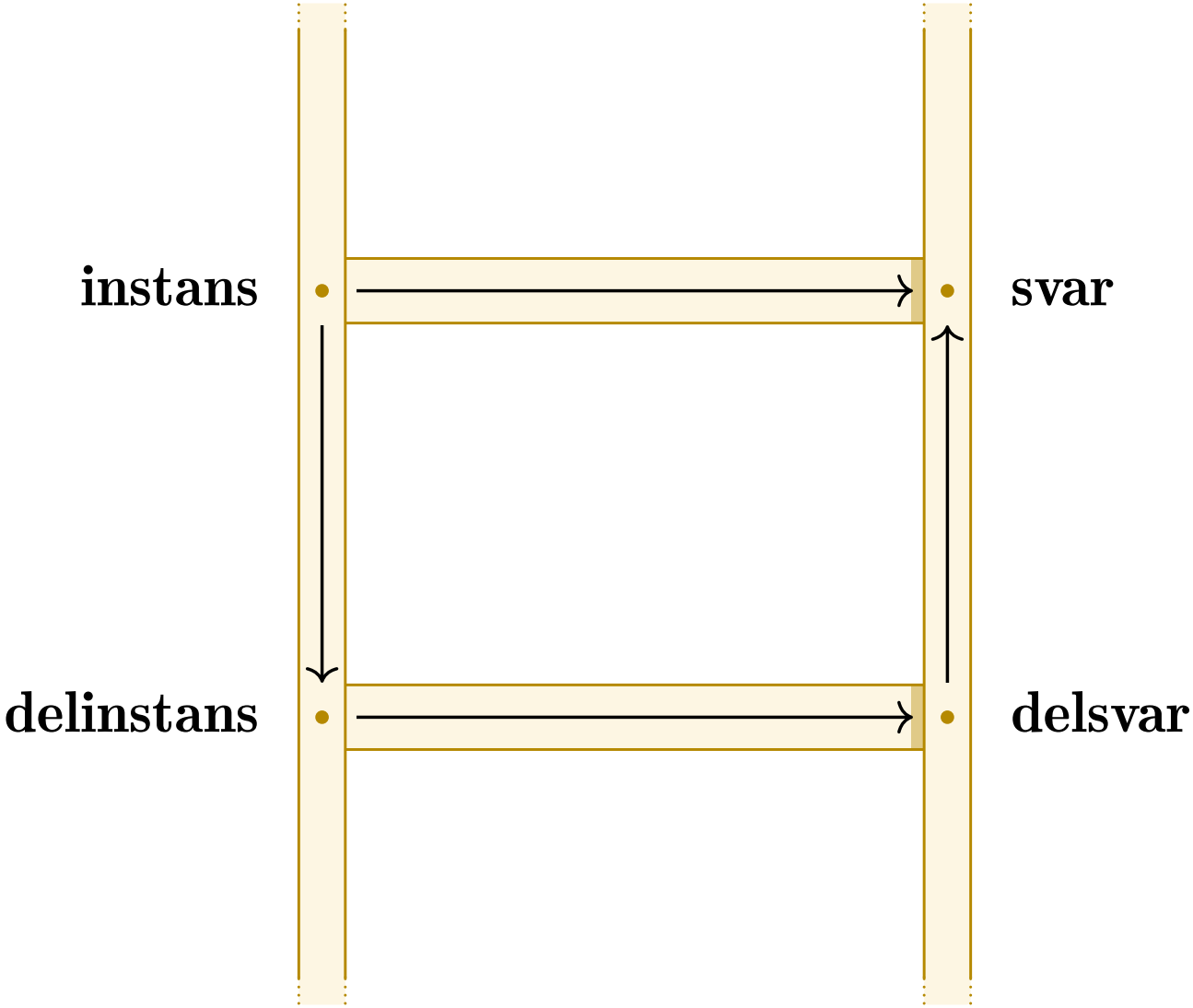
Problem



Hvorfor fungerer dette?

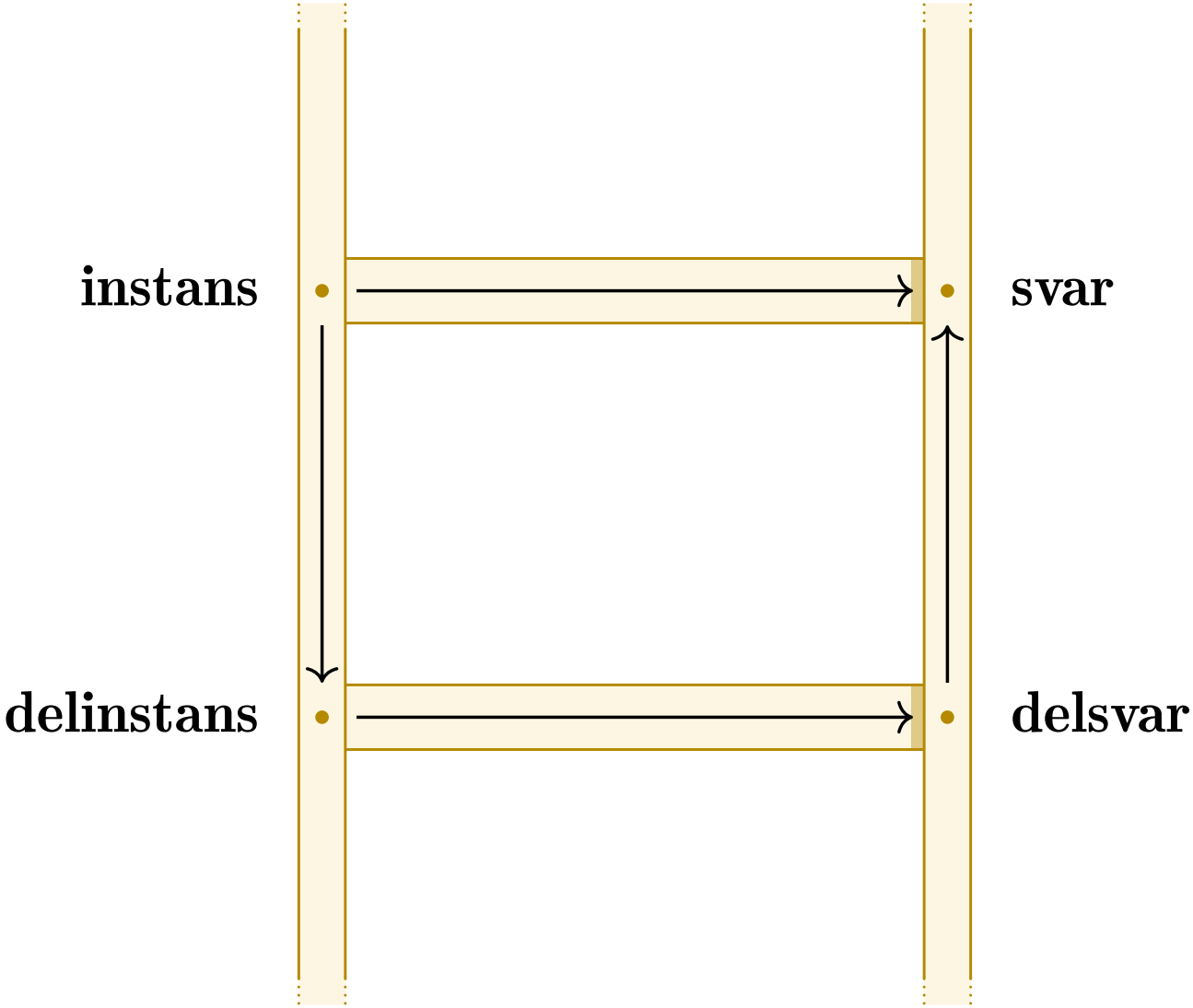
mlh@ntnu.no

Problem

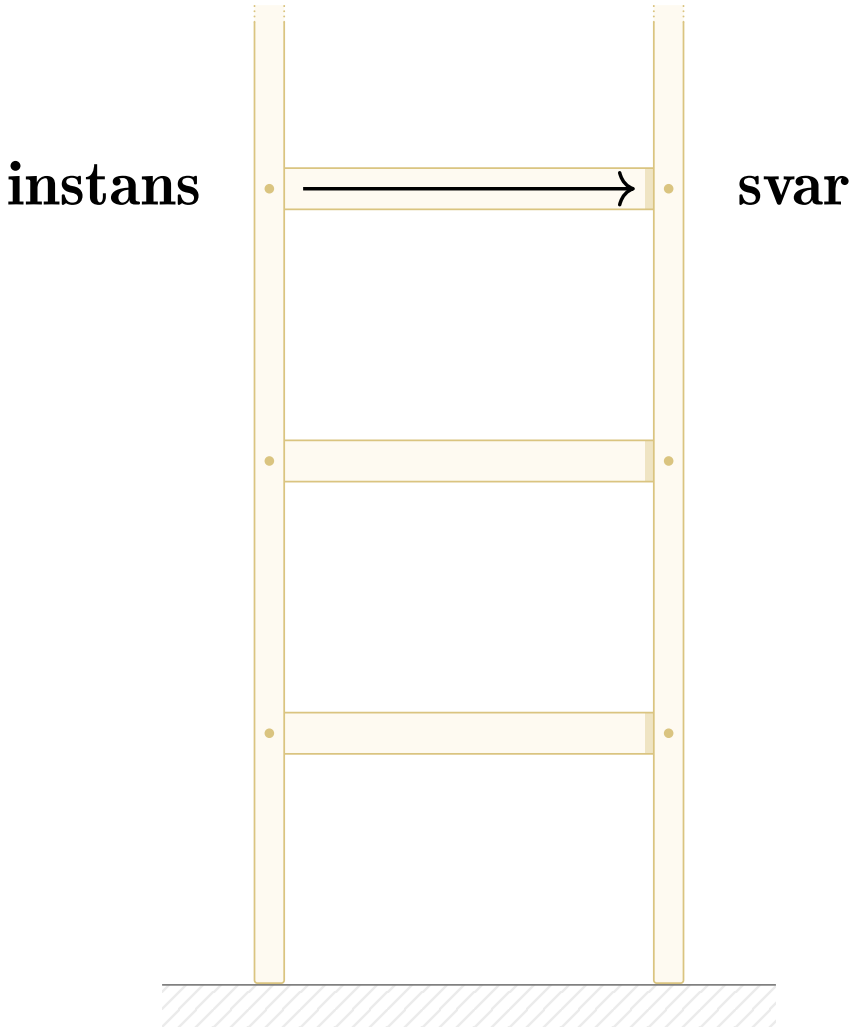


Det er samme prinsipp som før!

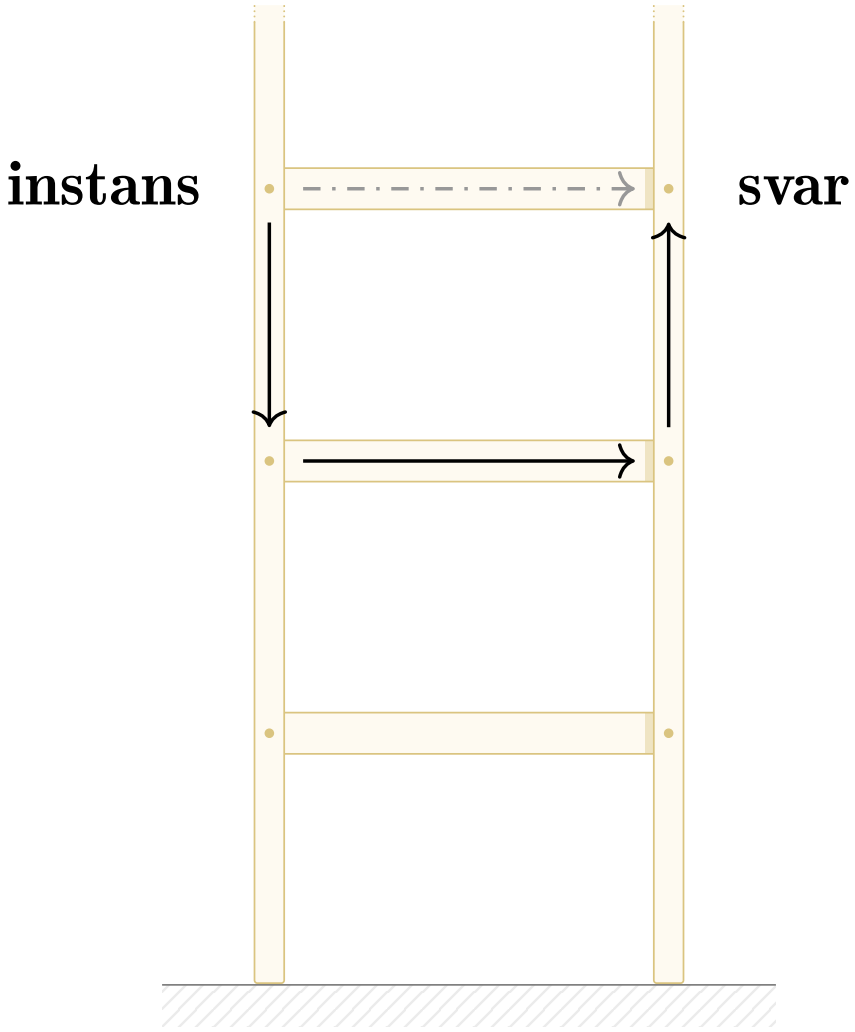
Problem



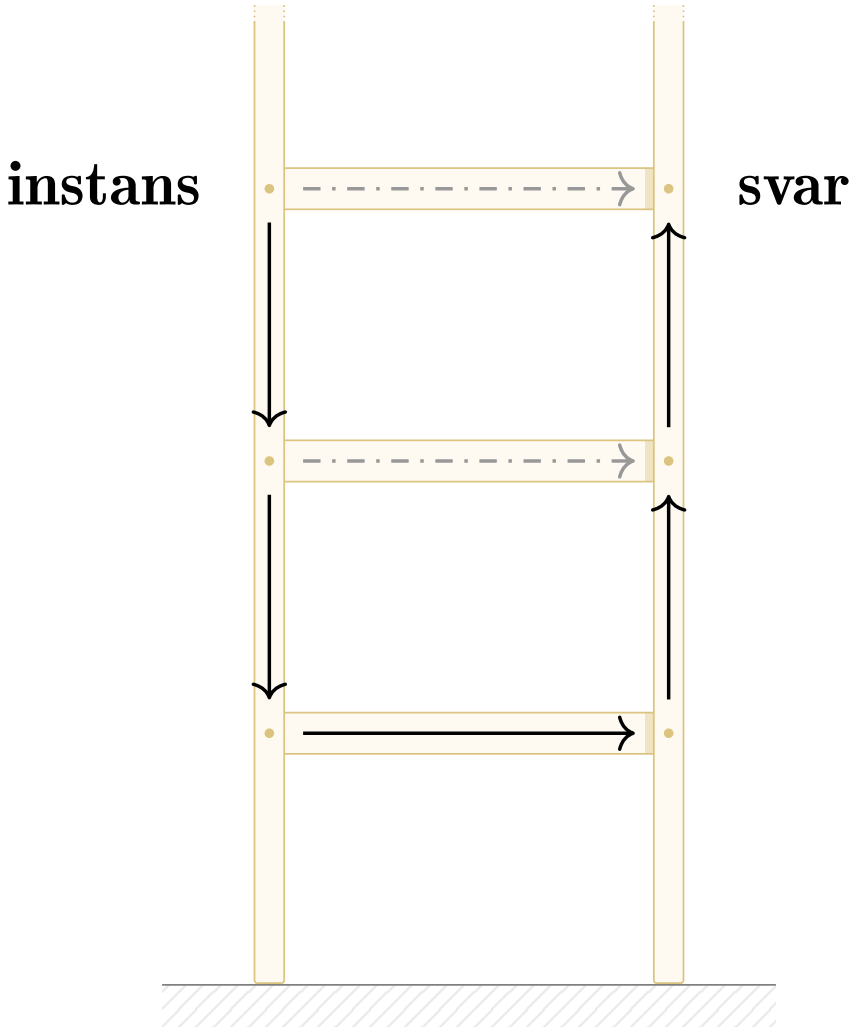
Vis grunntilfelle. Anta delløsning. Vis korrekt dekomp./kombinasjon



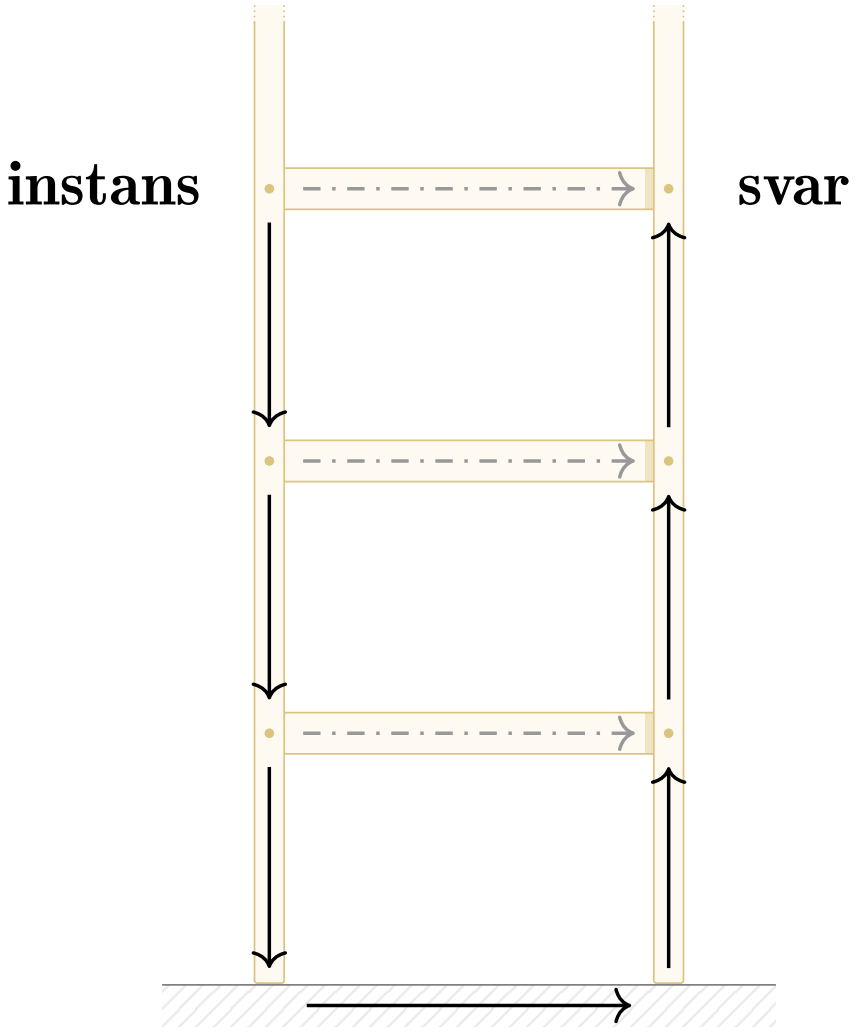
Poenget er at det samme skjer på hvert nivå



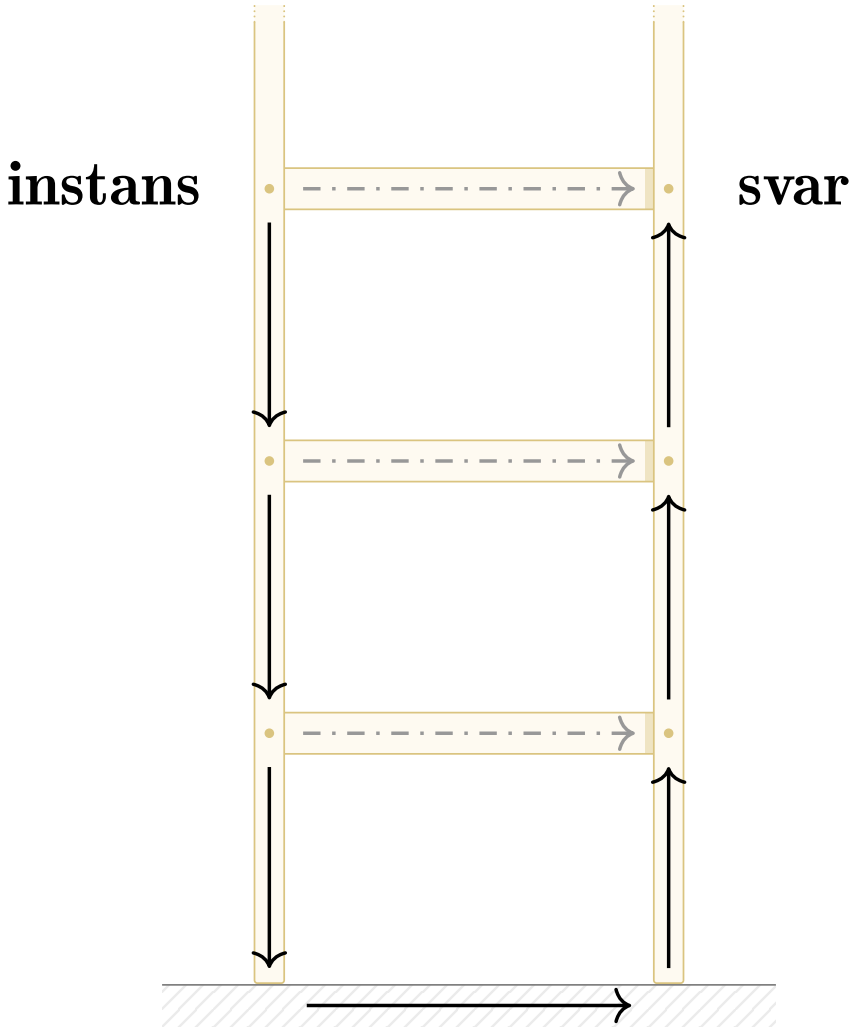
Poenget er at det samme skjer på hvert nivå



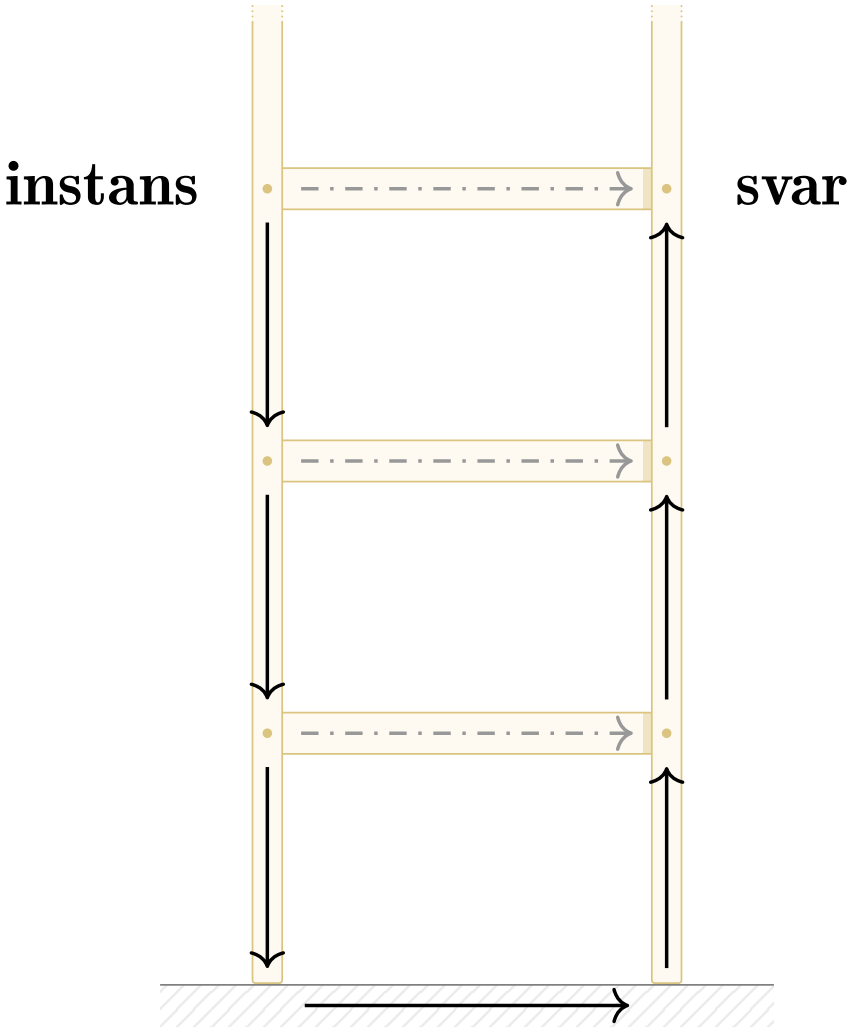
Poenget er at det samme skjer på hvert nivå



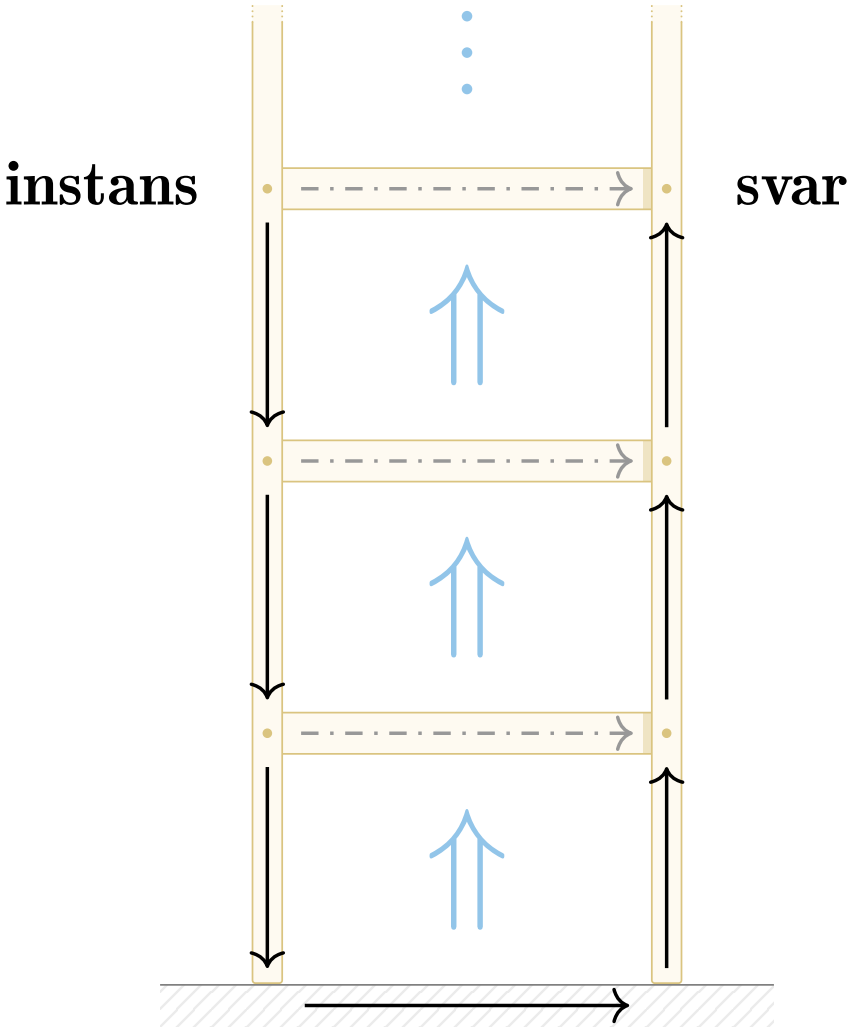
Poenget er at det samme skjer på hvert nivå



Om grunntilfelle, dekomponering og kombinasjon er rett . . .



. . . så er alle svarene riktige



. . . så er alle svarene riktige

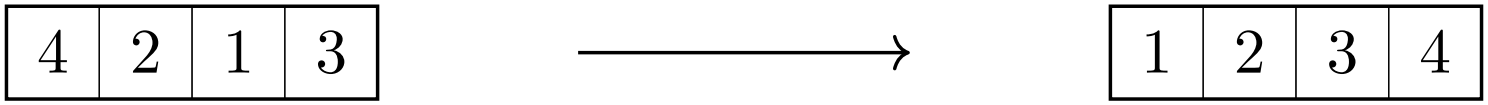
Annet perspektiv på induksjon: Løkkeinvarianter

Vis at den holder før løkka, og at løkka bevarer den. Da holder den også til slutt

Designmentode: Inkrementell.

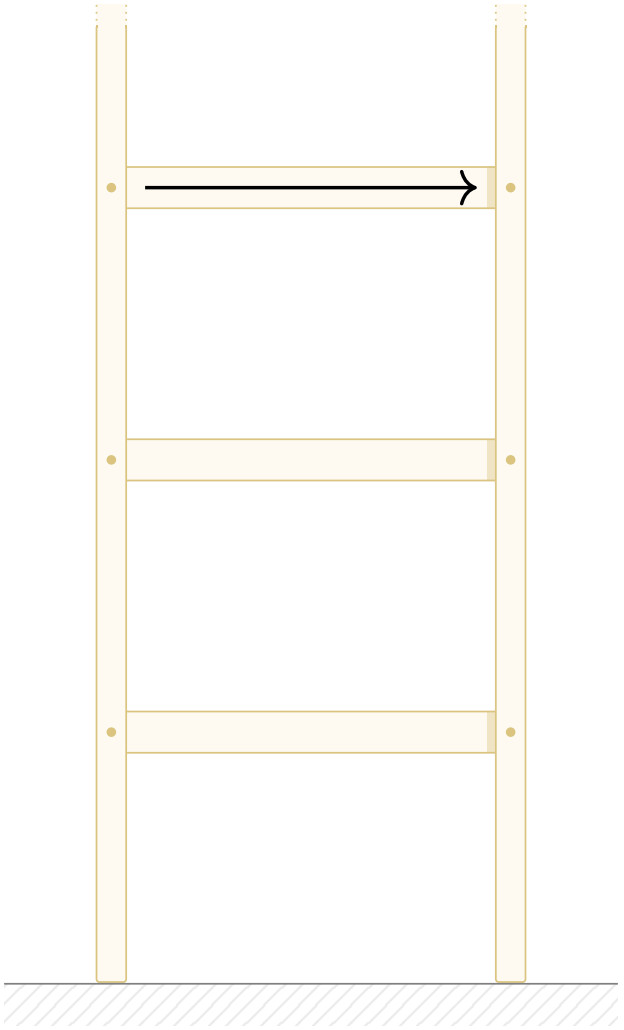
«Dekomponerer» i én delinstans.

Boka kaller dette «the incremental method». Vi skal se flere mer avanserte designmetoder – som i hovedsak baserer seg på samme dekomponeringsprinsipp.



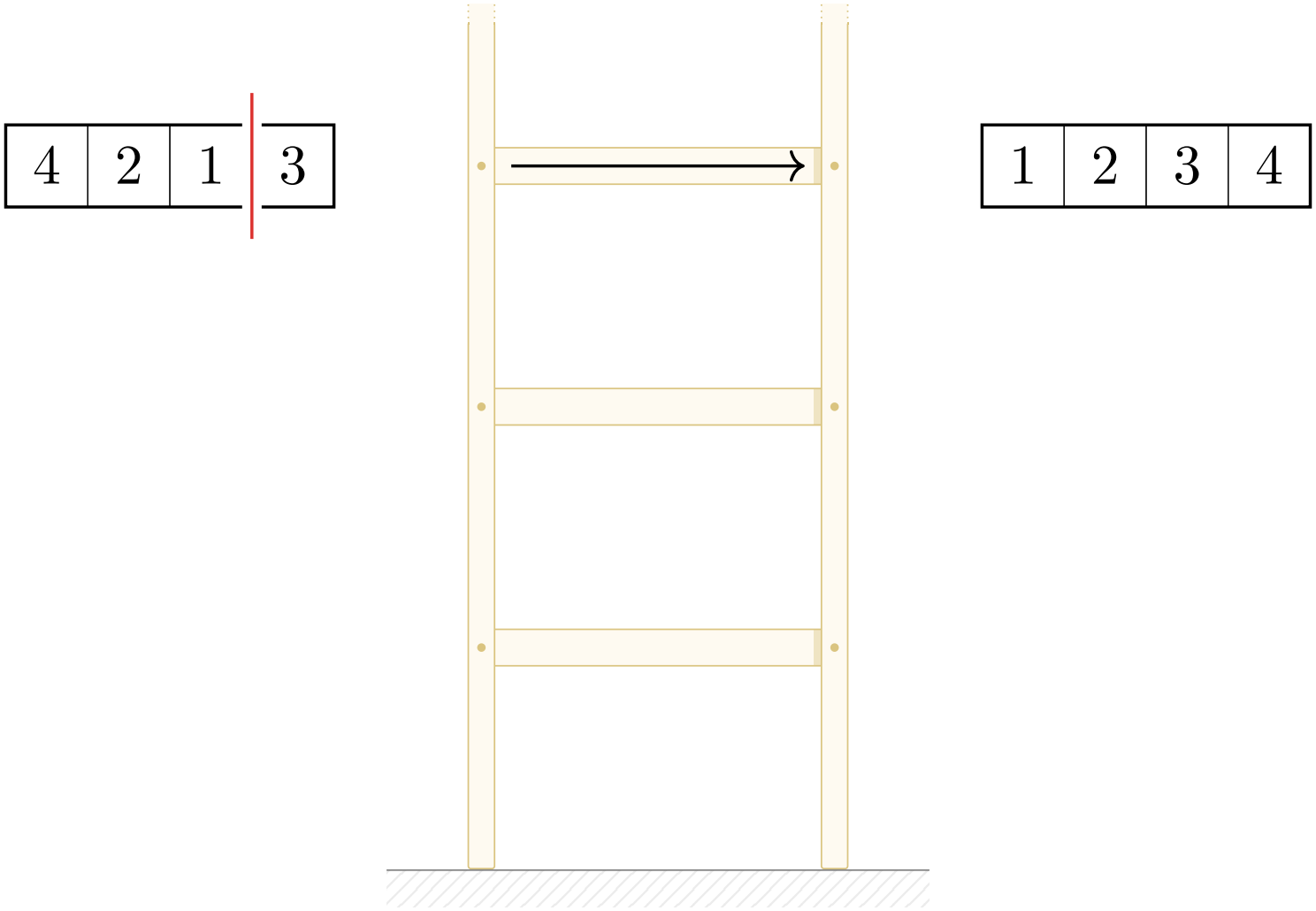
Vi vil sortere en sekvens

4	2	1	3

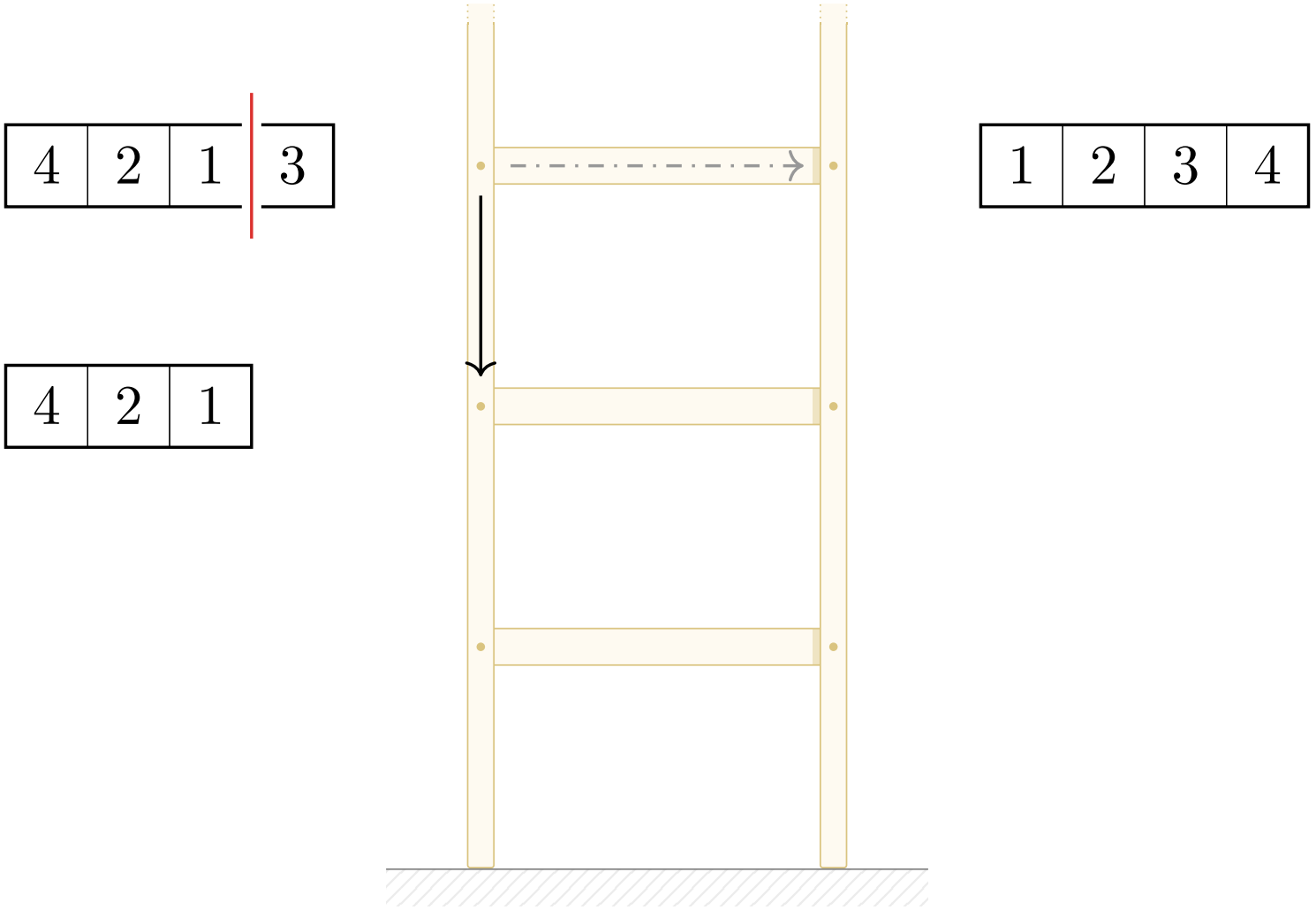


1	2	3	4

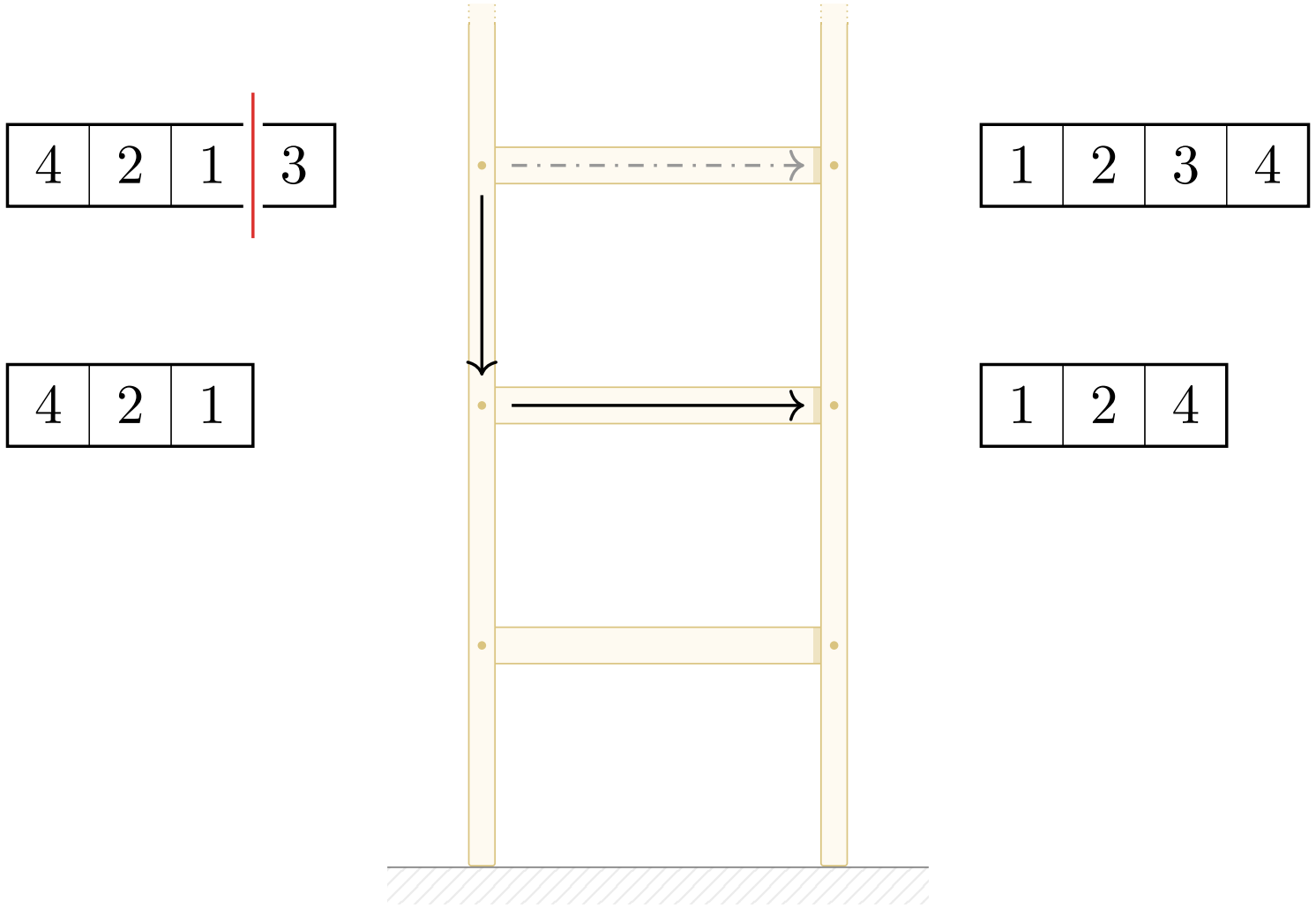
Vi bruker dekomponerings-/induksjonsstigen vår



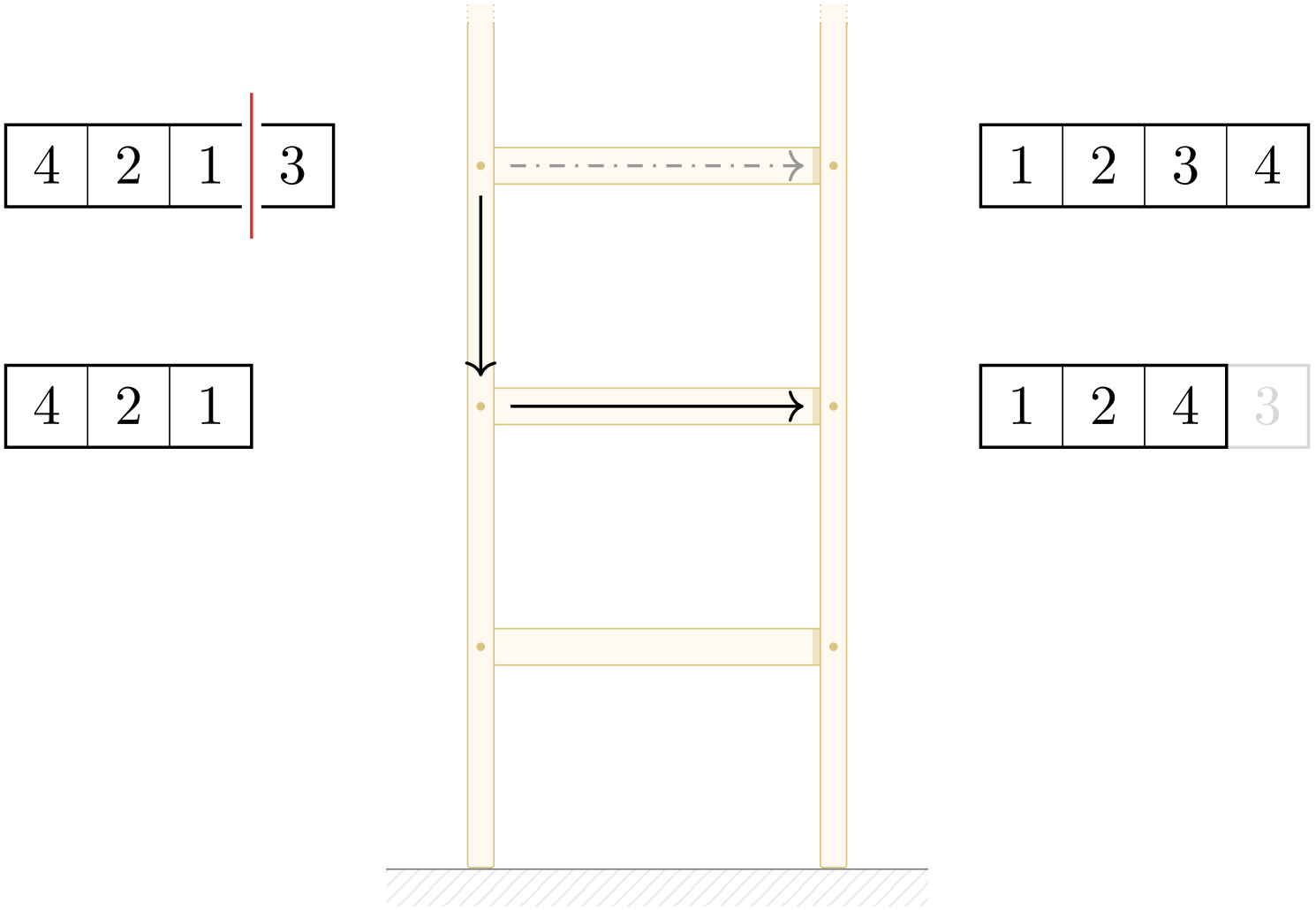
Enkleste dekomponering: Kutt av siste element



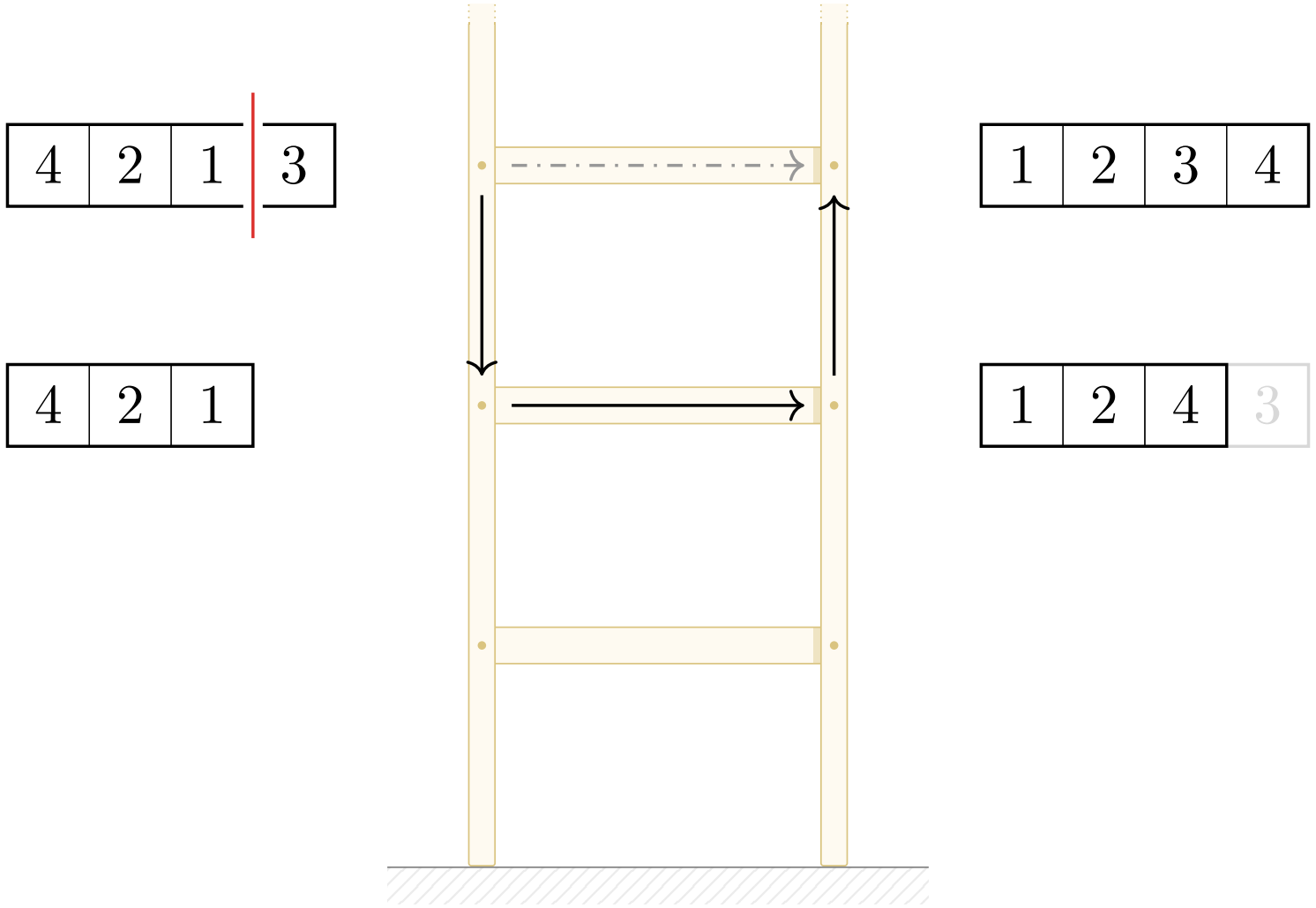
Har nå ny og mindre instans. IH: Vi kan løse denne



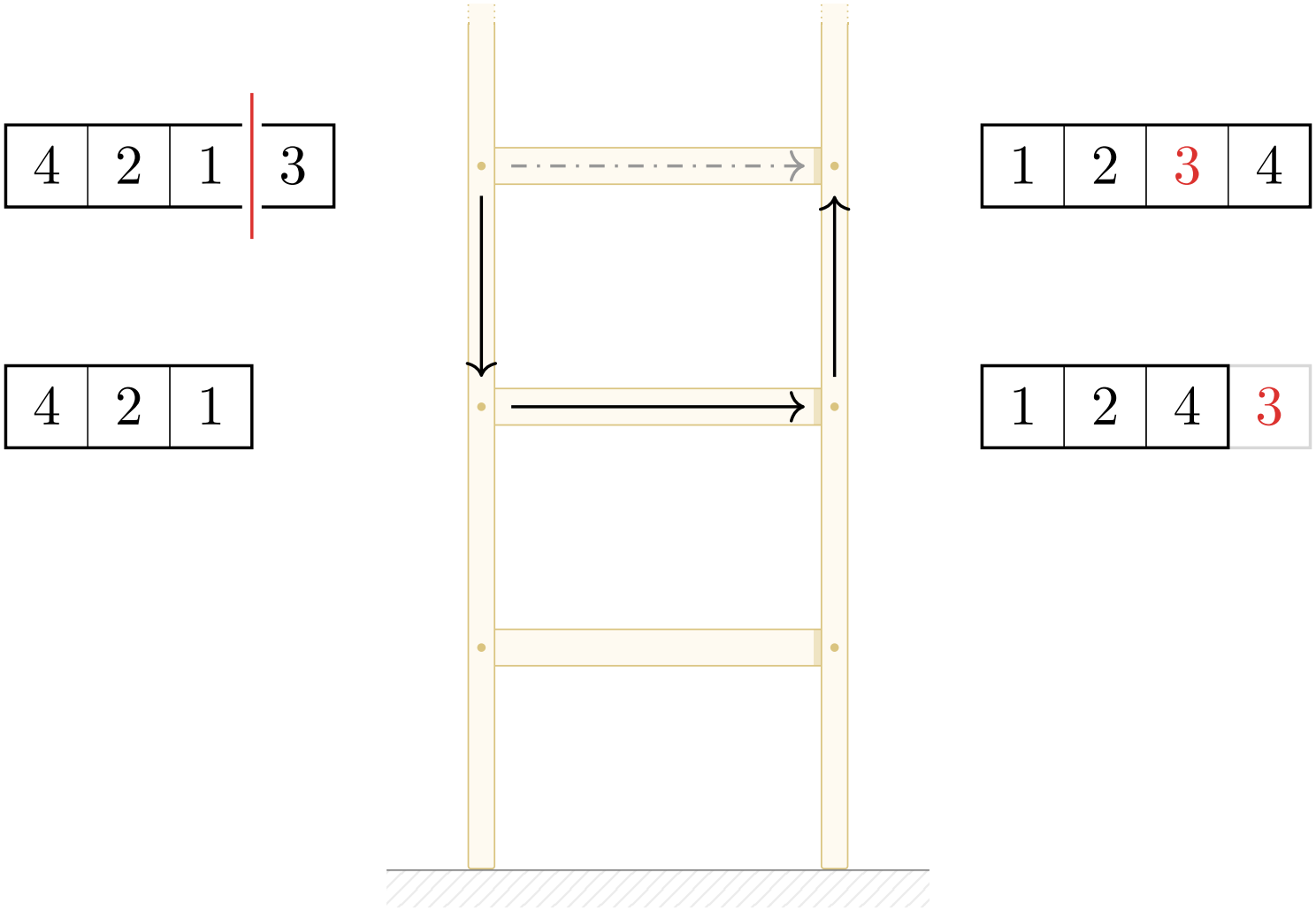
Har nå ny og mindre instans. IH: Vi kan løse denne



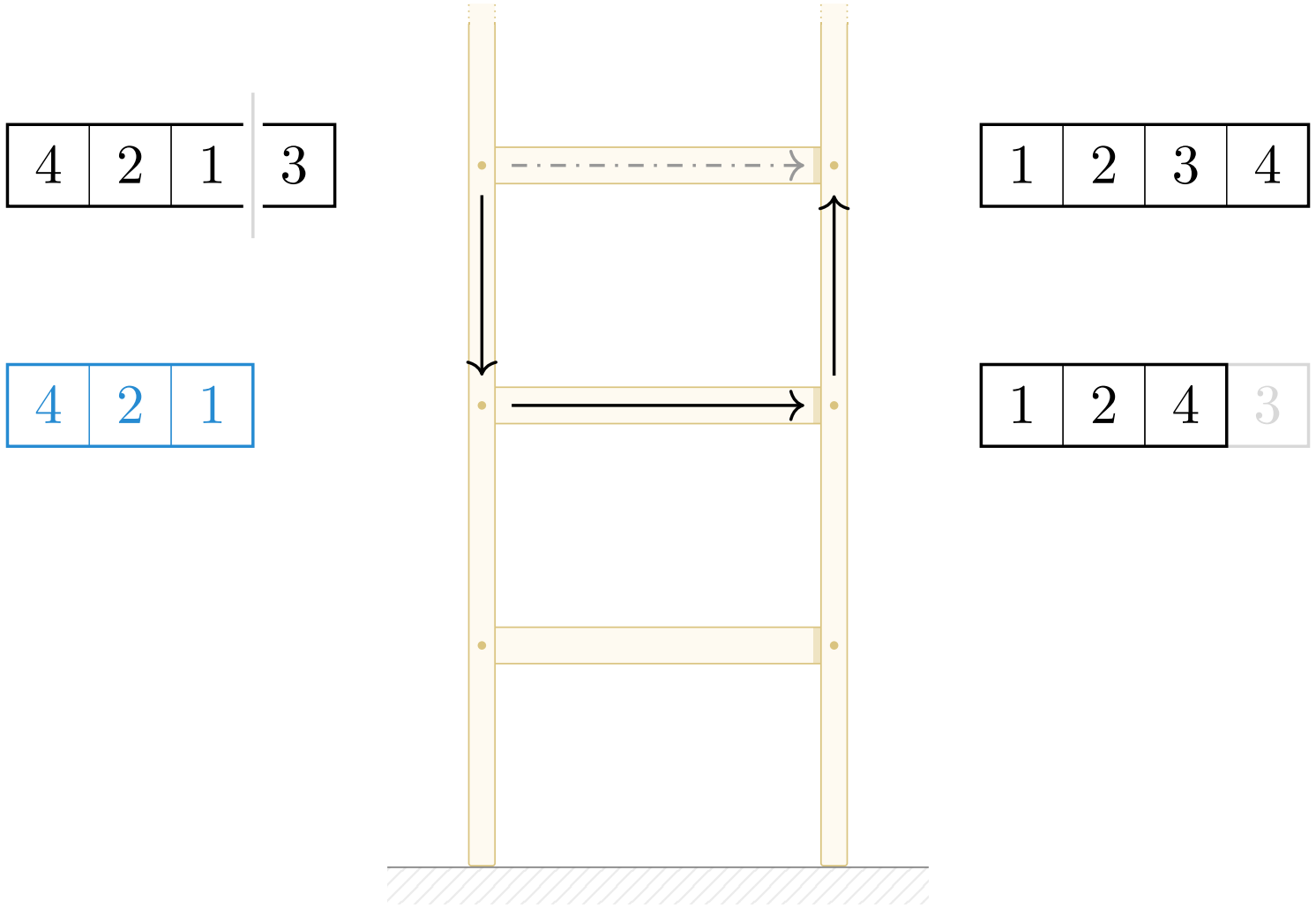
Vi har nå en delløsning, men ikke en ferdig løsning



Bygg løsning fra delløsning: Sett inn siste element

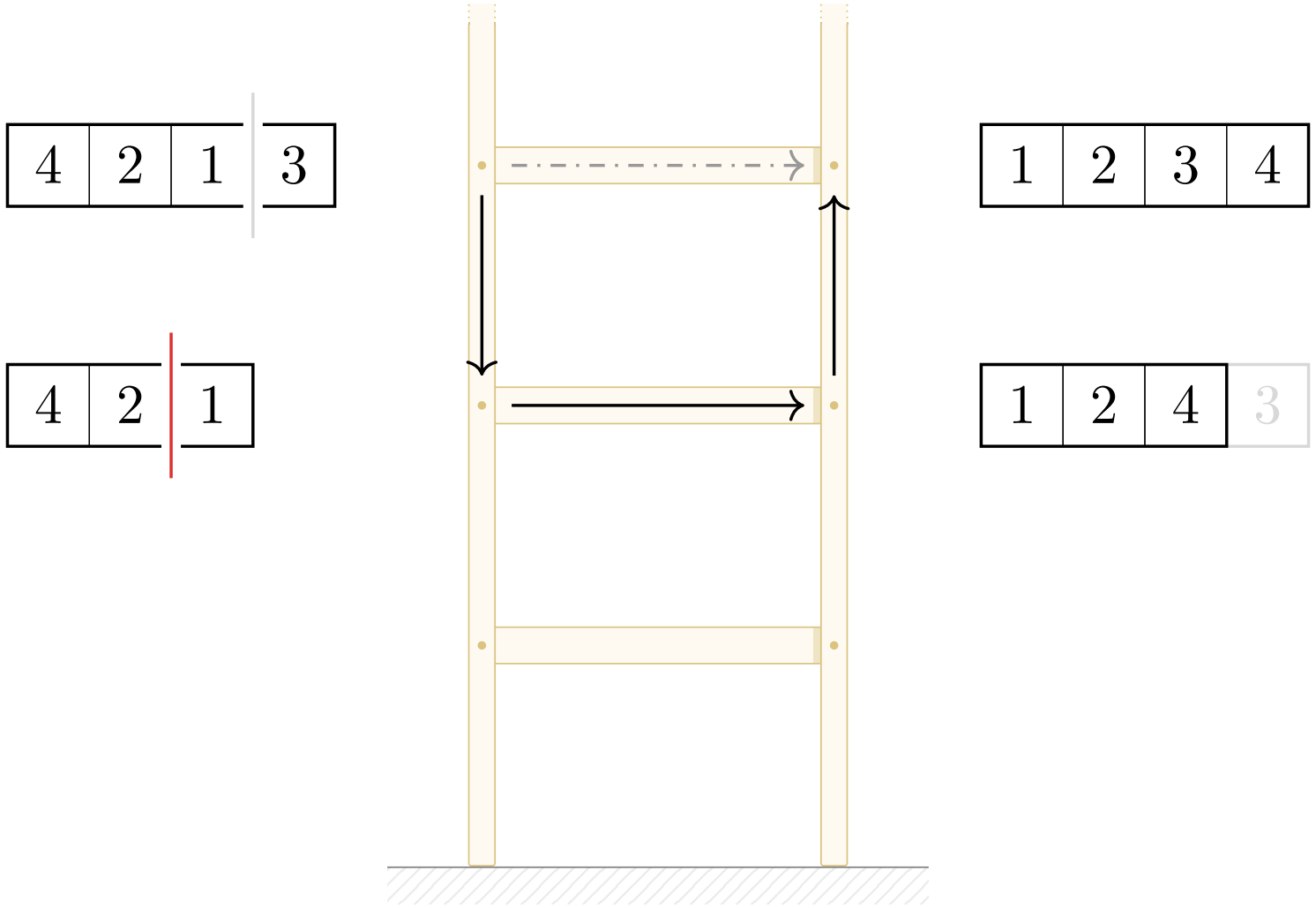


Bygg løsning fra delløsning: Sett inn siste element

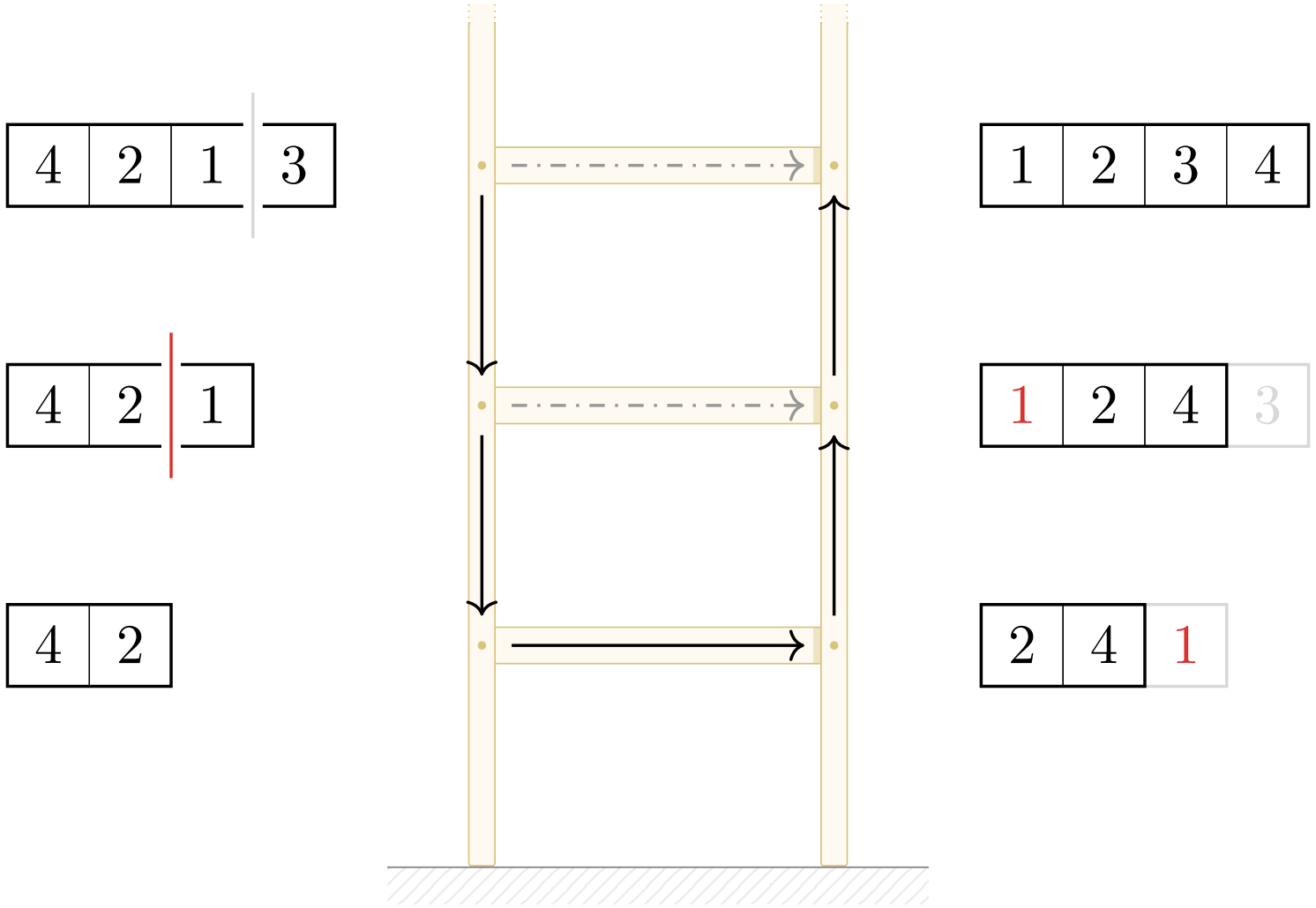


Delinstansen 4 2 1 løser vi på samme måte (rekursivt, induktivt)

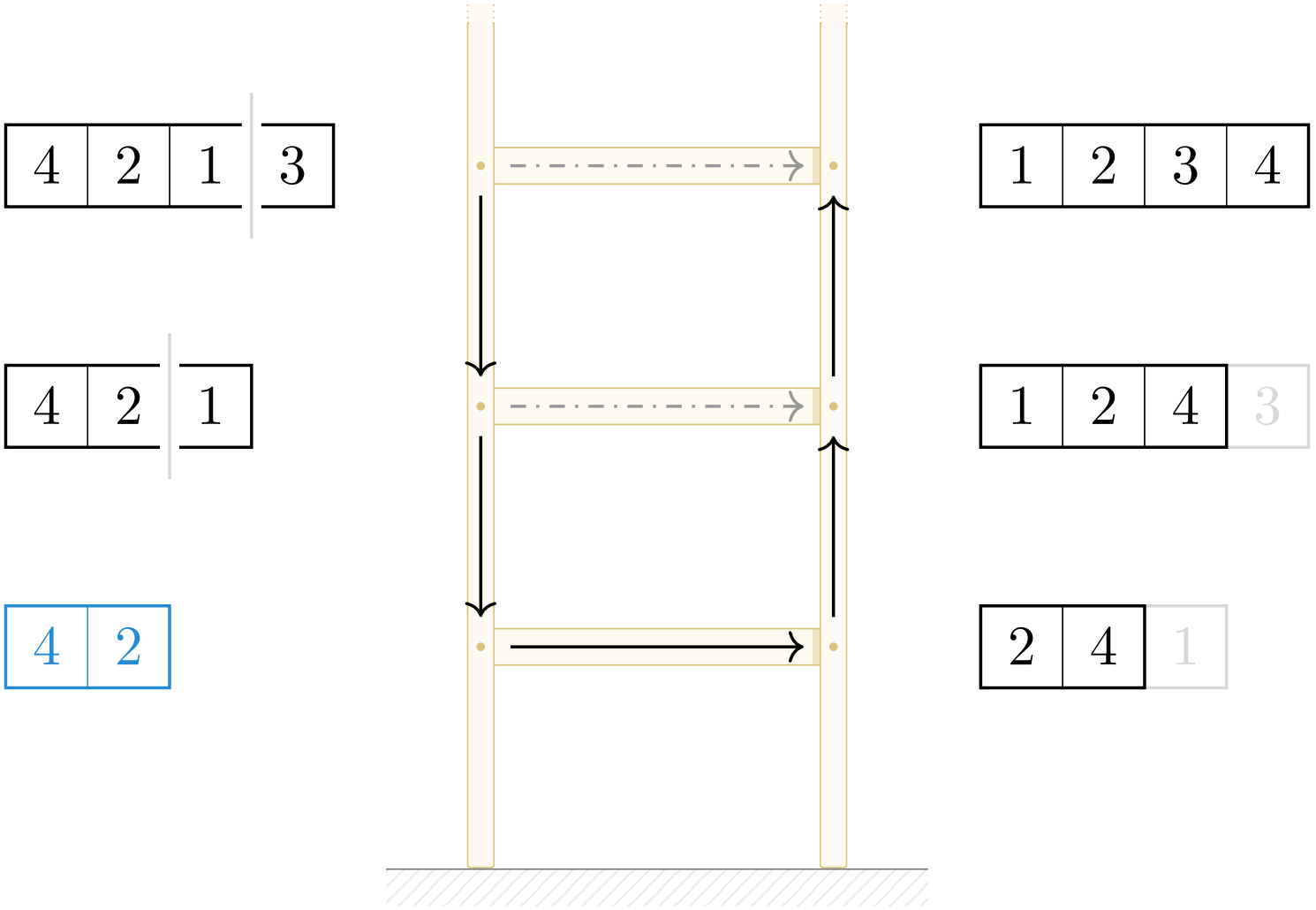
mlh@ntnu.no



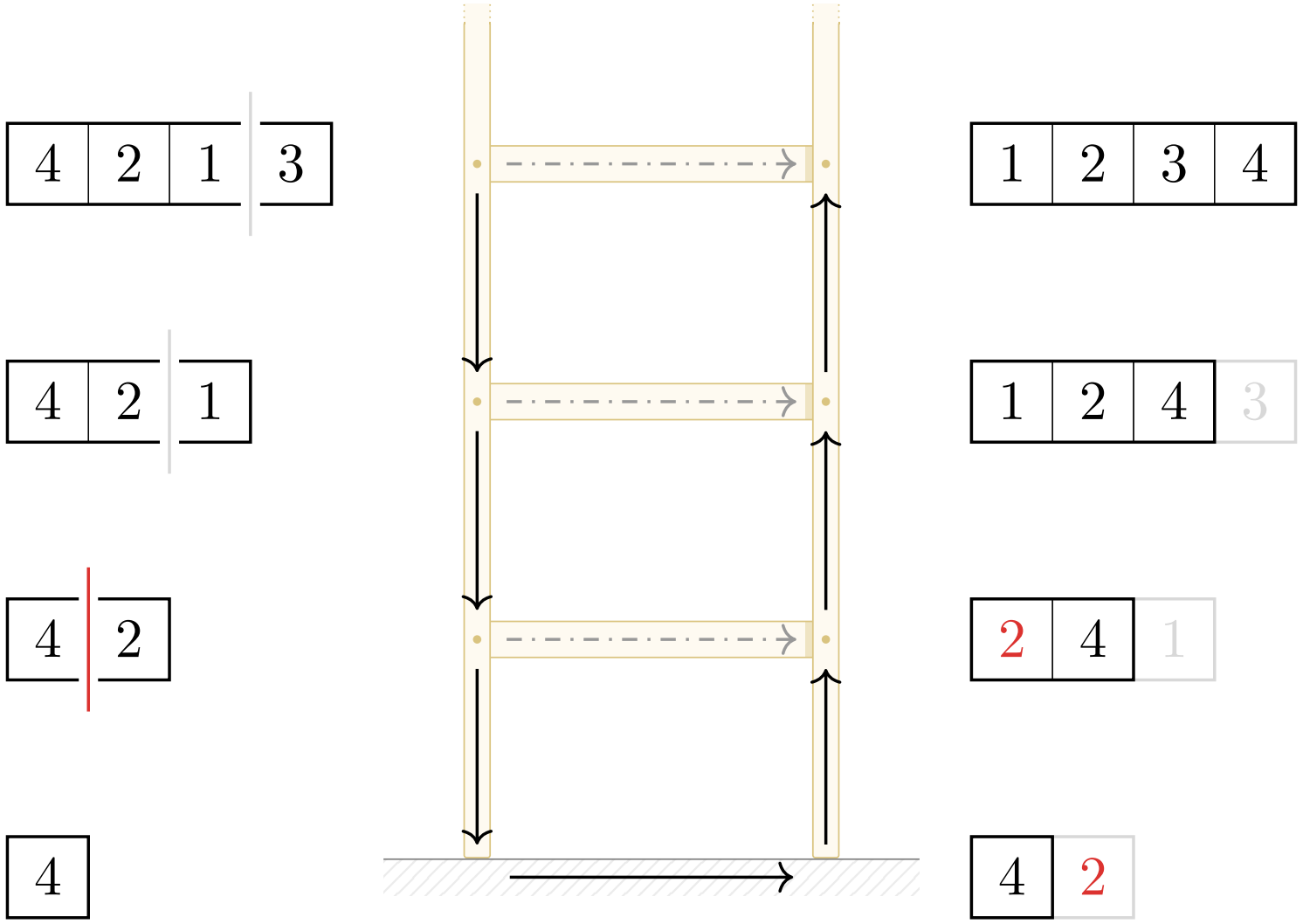
Delinstansen | 4 | 2 | 1 | løser vi på samme måte (rekursivt, induktivt)



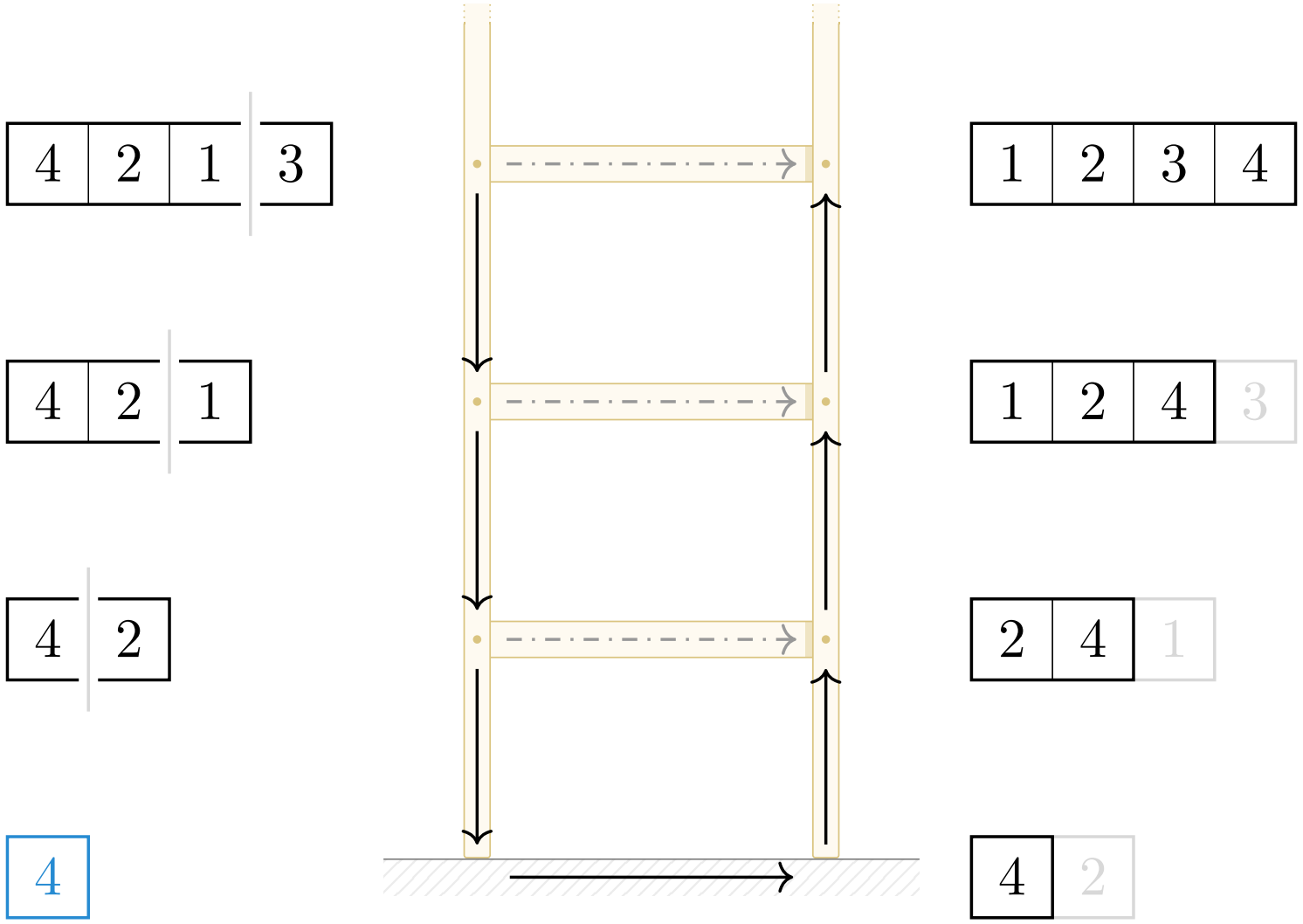
Delinstansen $\boxed{4}\boxed{2}\boxed{1}$ løser vi på samme måte (rekursivt, induktivt)



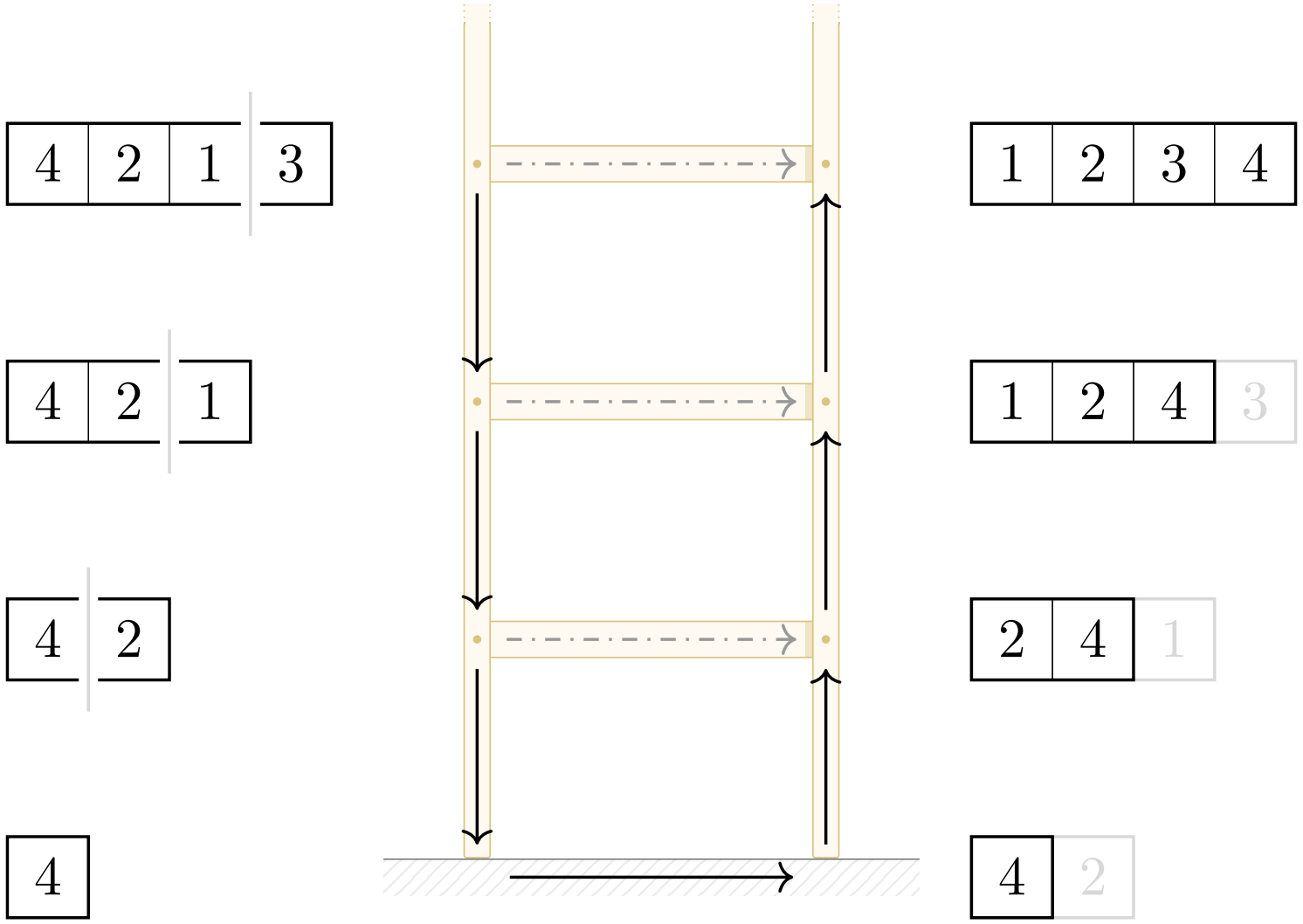
Og hva med 4 2 ? Løses på samme måte

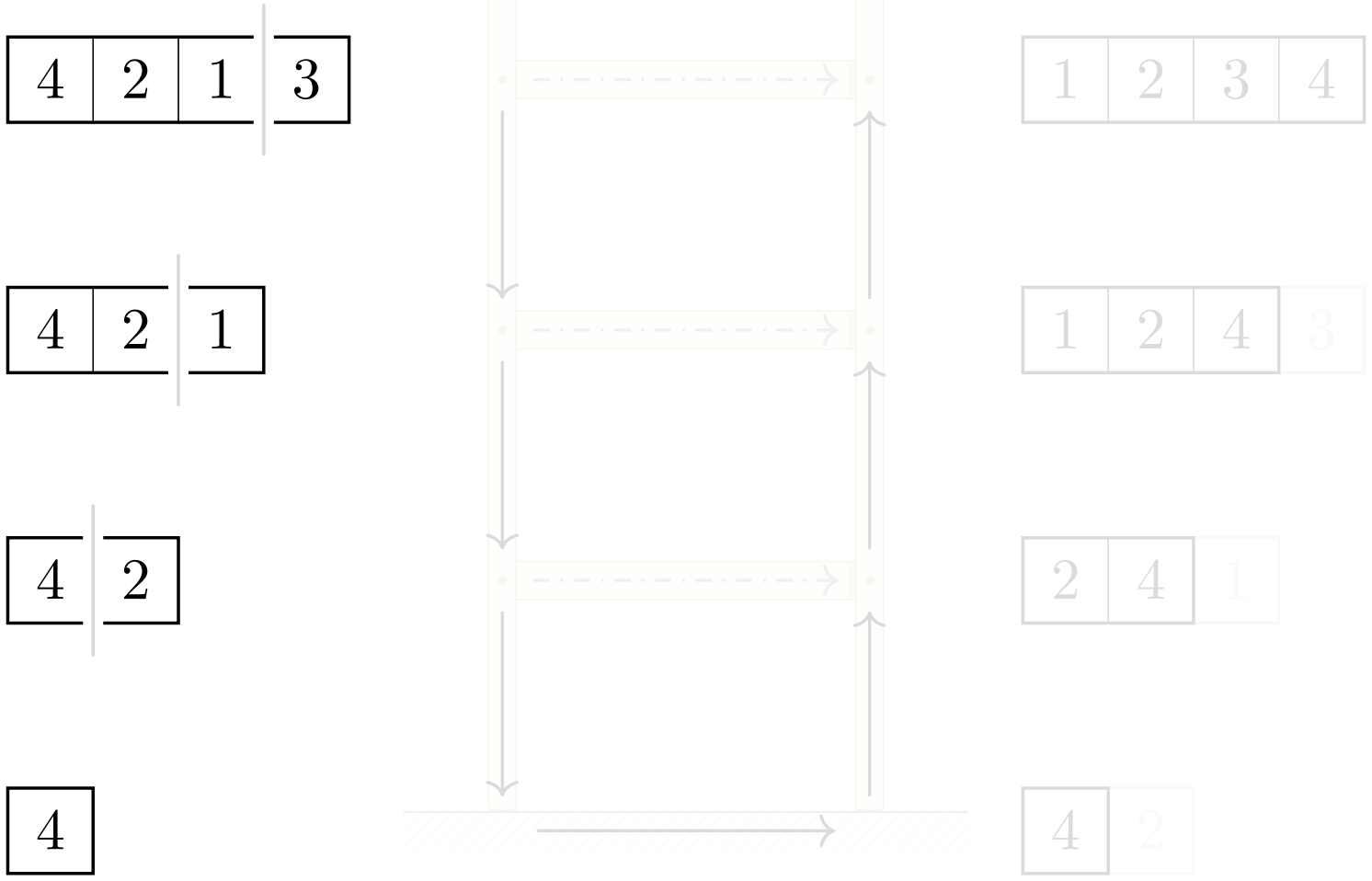


Og hva med $\boxed{4\,|\,2}$? Løses på samme måte



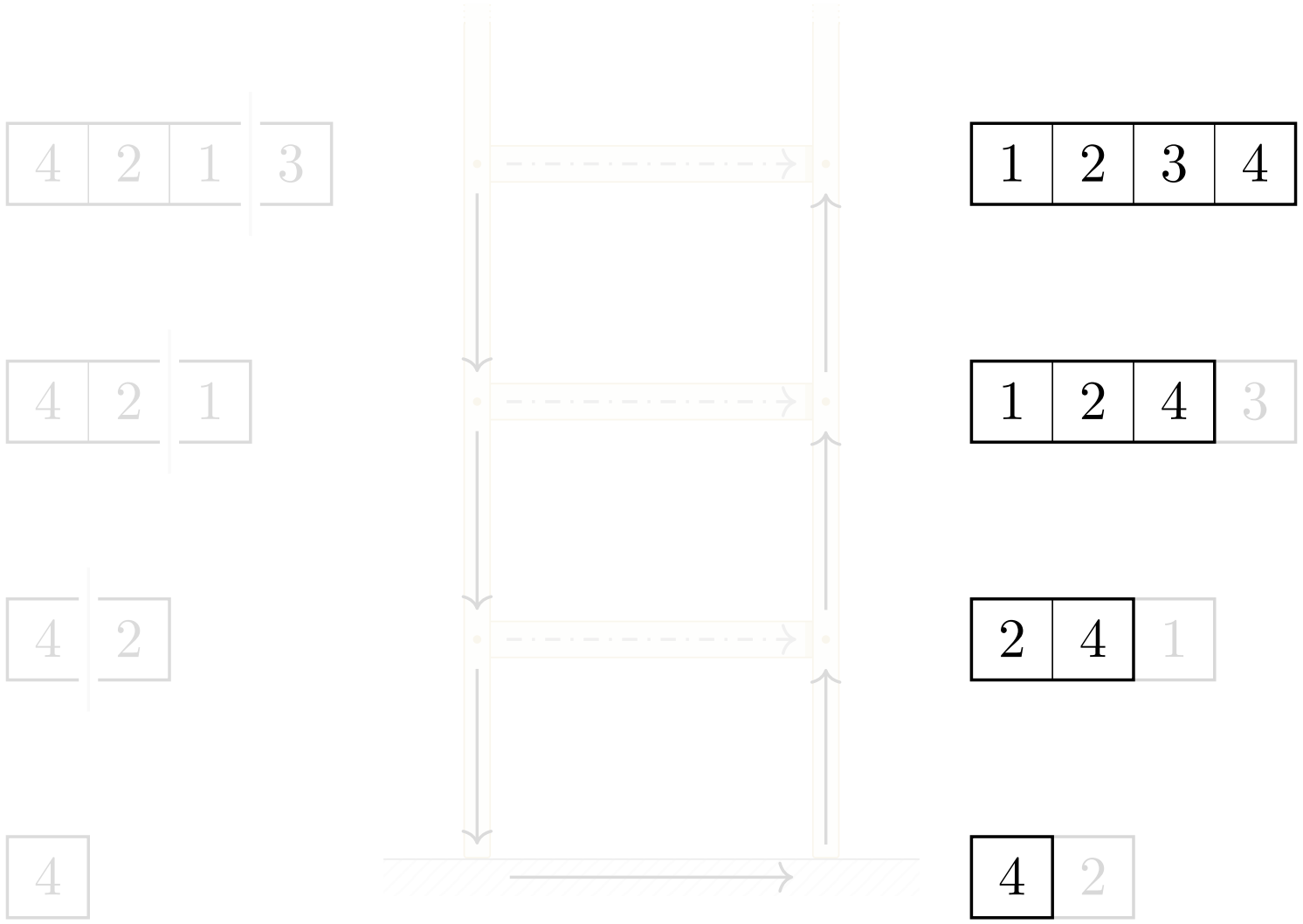
4 er et grunntilfelle, og løses lett



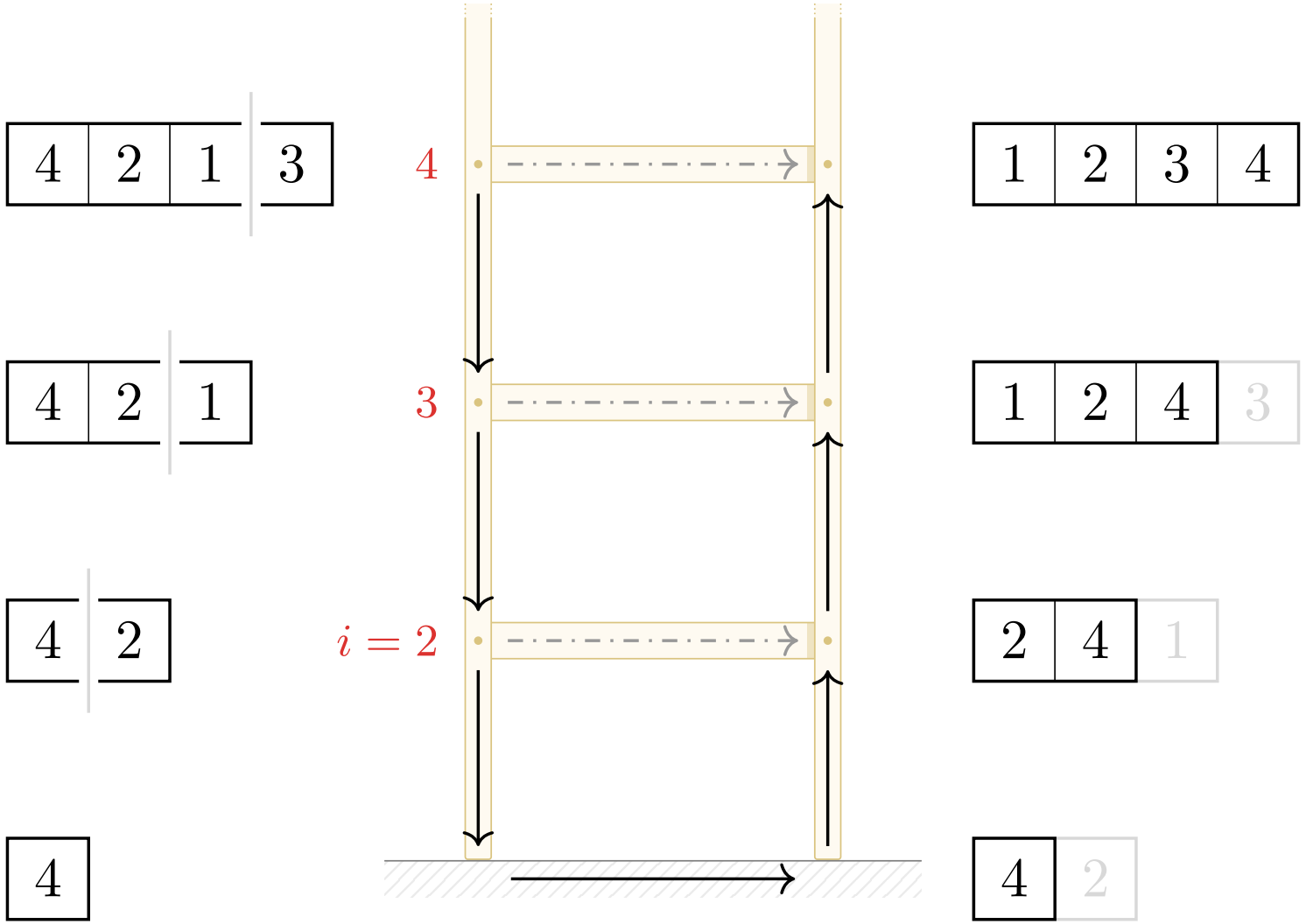


Dekomponeringen her er veldig enkel . . .

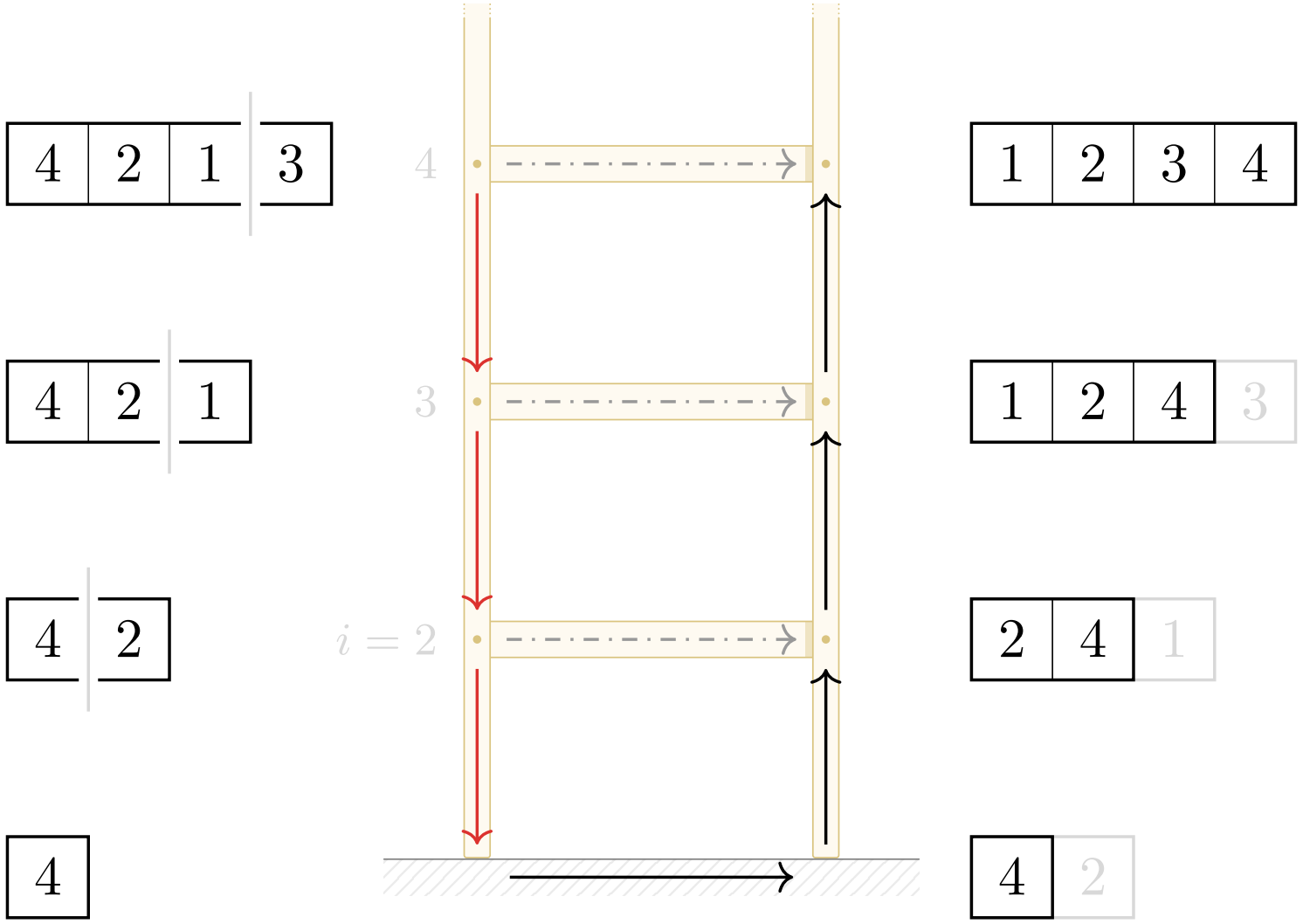
mlh@ntnu.no



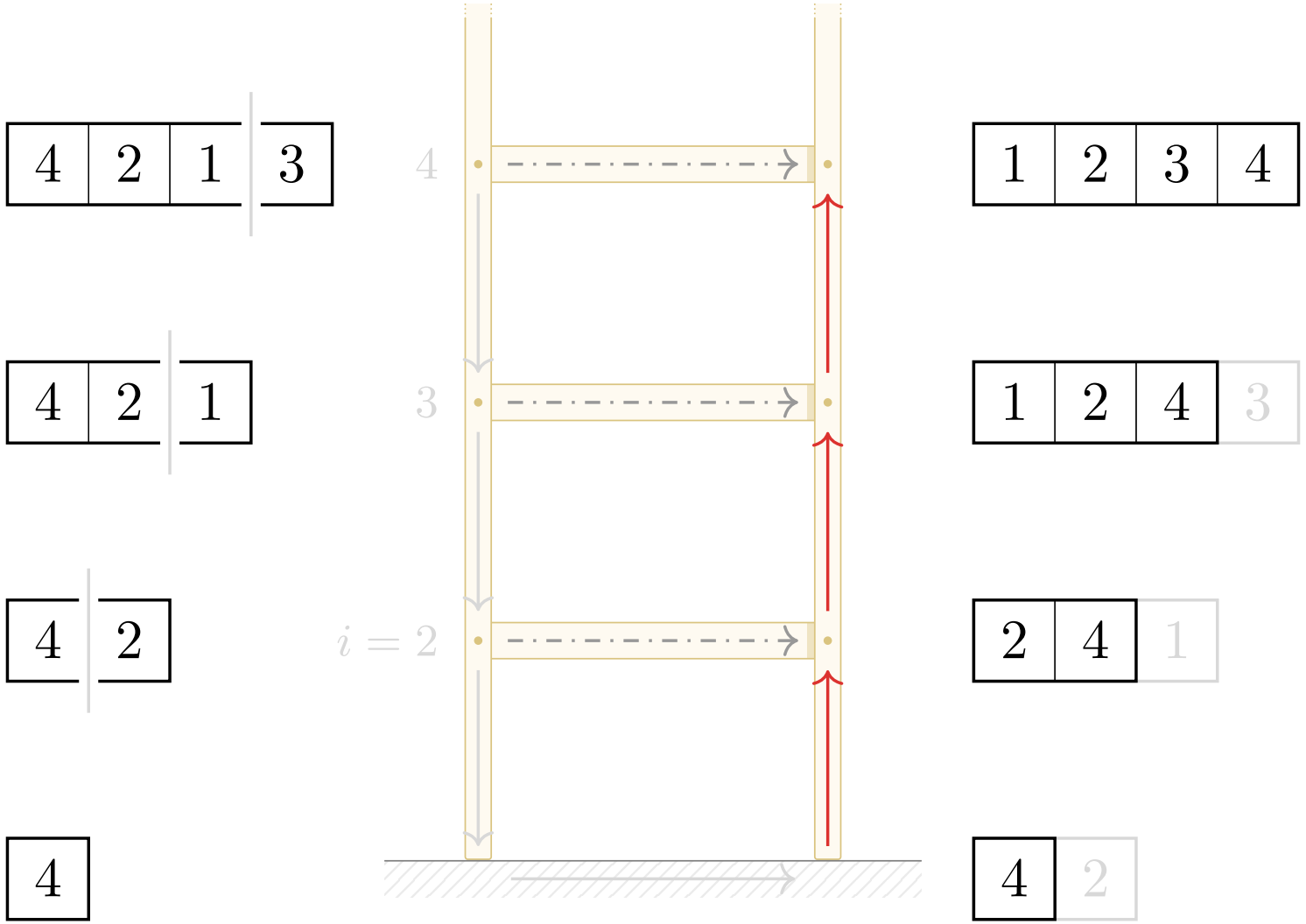
. . . så alt arbeidet ligger i «løsningsbyggingen»



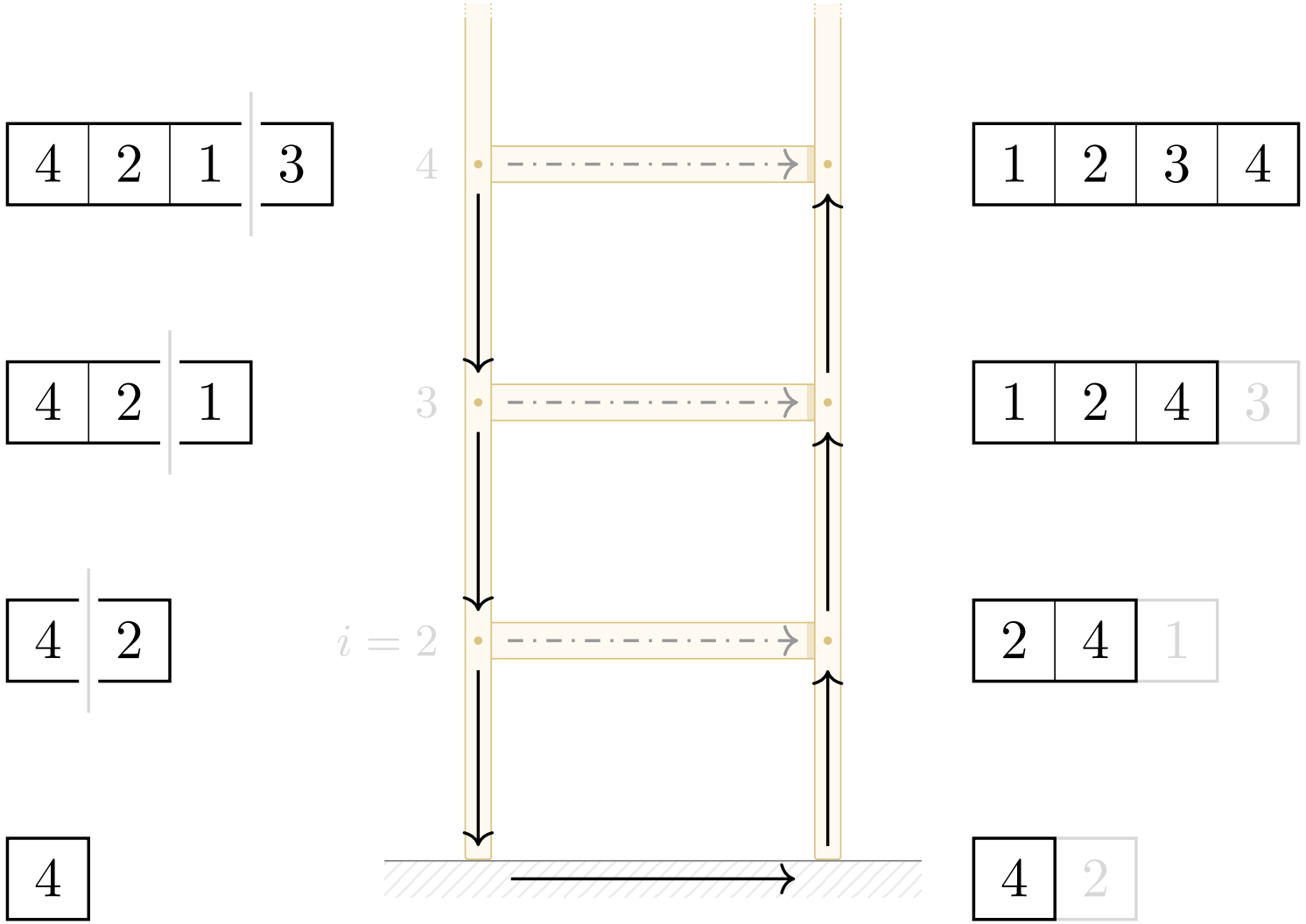
For $i = 2 \ldots n$



For $i = 2 \ldots n$, anta at $\mathrm{A}[1 : i - 1]$ er sortert



For $i = 2 \ldots n$, anta at $\mathrm{A}[1 : i-1]$ er sortert og sett inn $\mathrm{A}[i]$



For $i = 2 \ldots n$, anta at $\mathrm{A}[1 : i - 1]$ er sortert og sett inn $\mathrm{A}[i]$

Om vi heller gjør hovedarbeidet i dekomponeringen, og f.eks. henter ut største element: «Sortering ved utvelgelse», kjent som selection-sort

Vi kan da se på det å fjerne største element som en enkel reduksjon