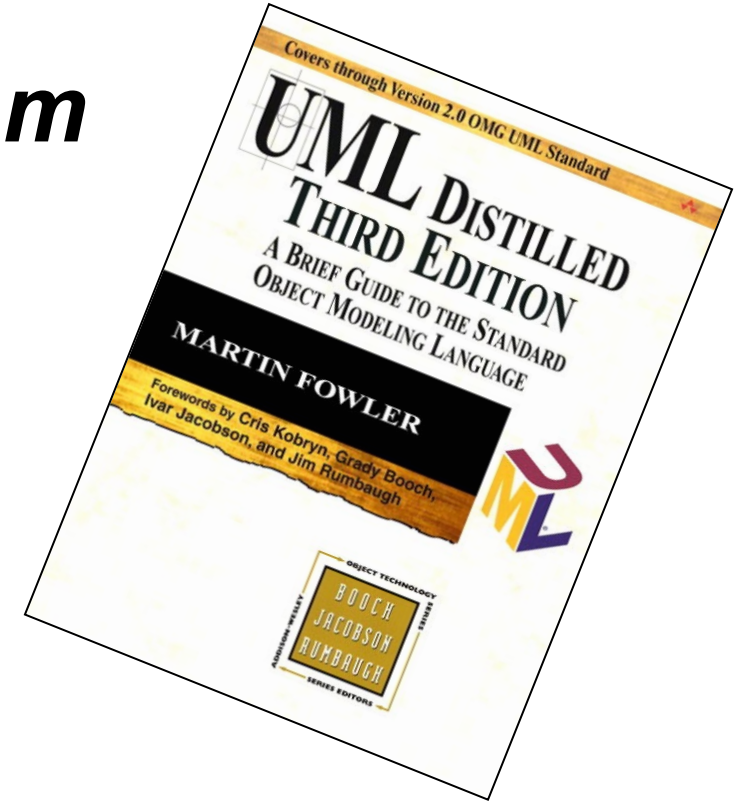
UML

Tilstandsmaskindiagram, aktivitetsdiagram og sekvensdiagram



TTK4235

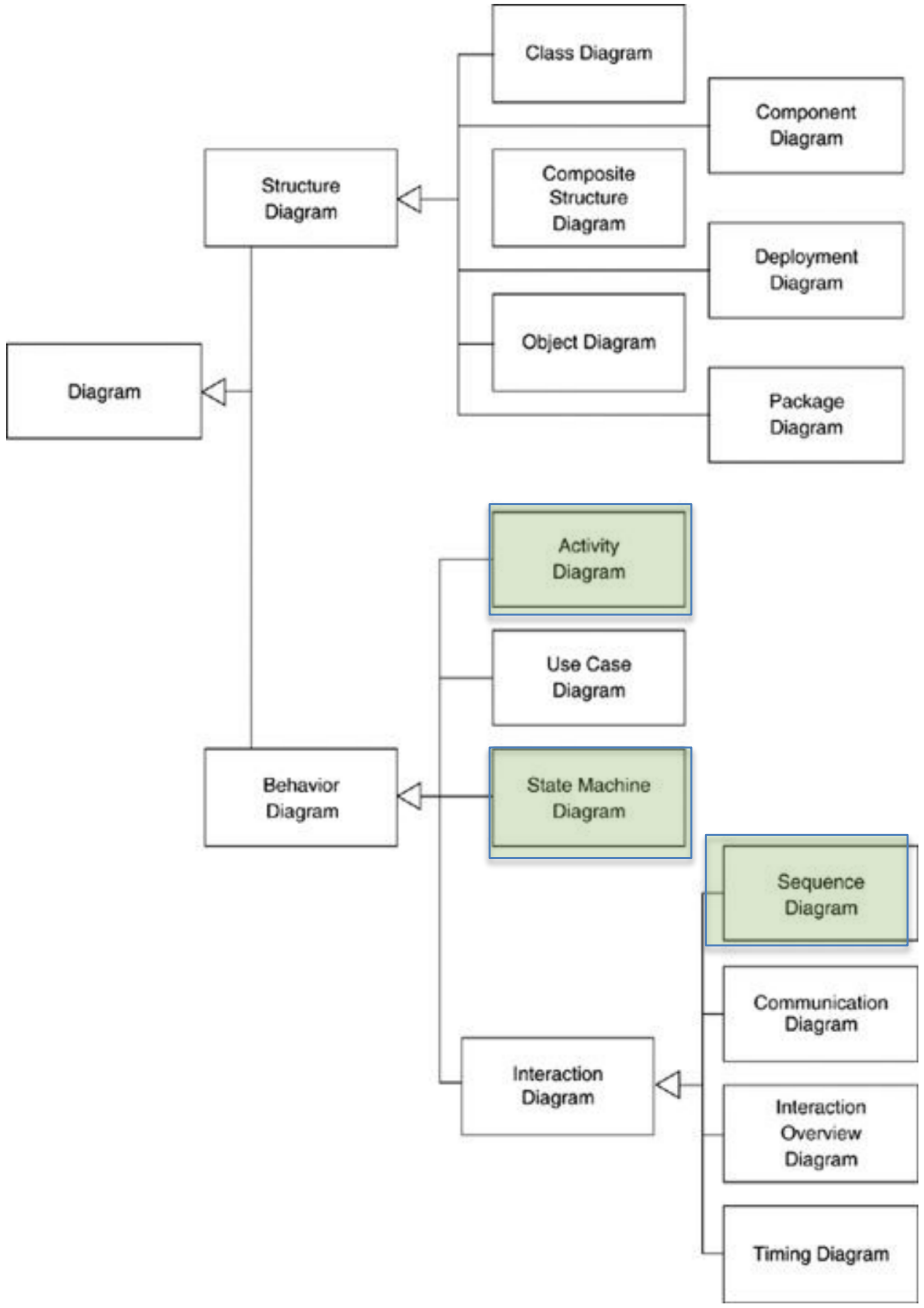
Kilder:

UML Distilled, 3rd ed., Martin Fowler

Dagens tema

Vi ser på tre av diagrammene som beskriver atferd:

- Tilstandsmaskindiagram
- Aktivitetsdiagram
- Sekvensdiagram



Tilstandsmaskindiagram

Tilbakeblikk fra logikkstyring

Tilstandsmaskinformalismen og eksempelet om gardintrekkmekanismen:

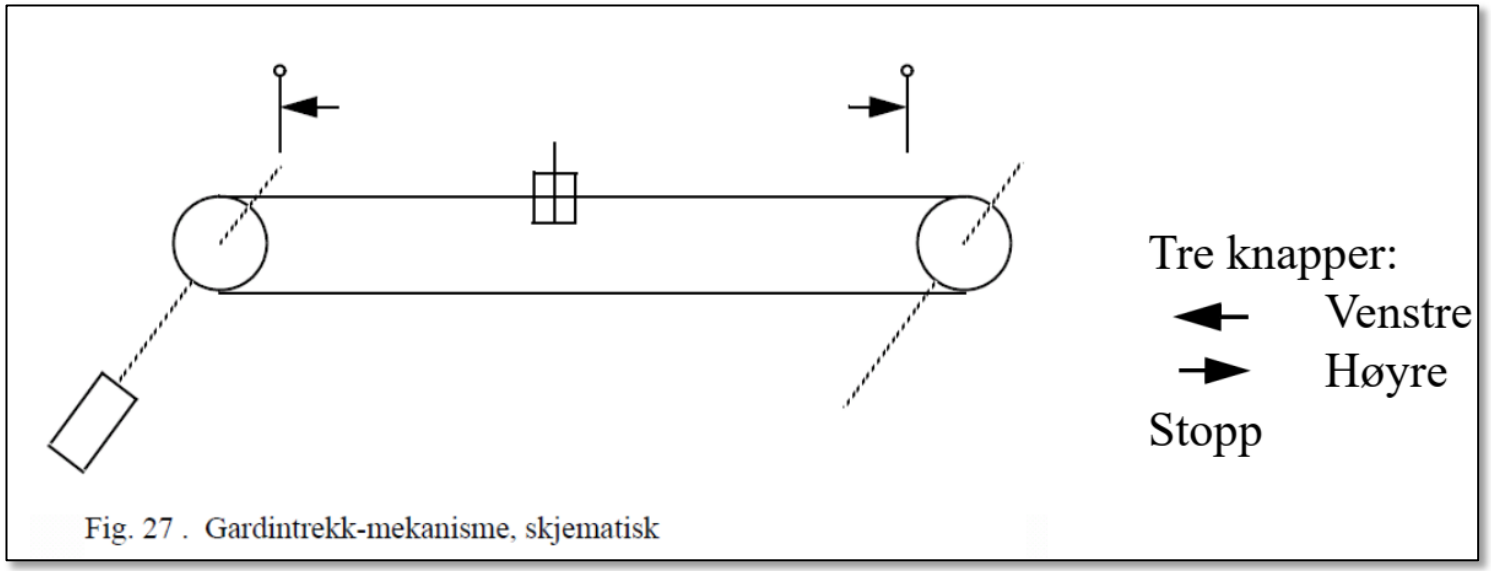
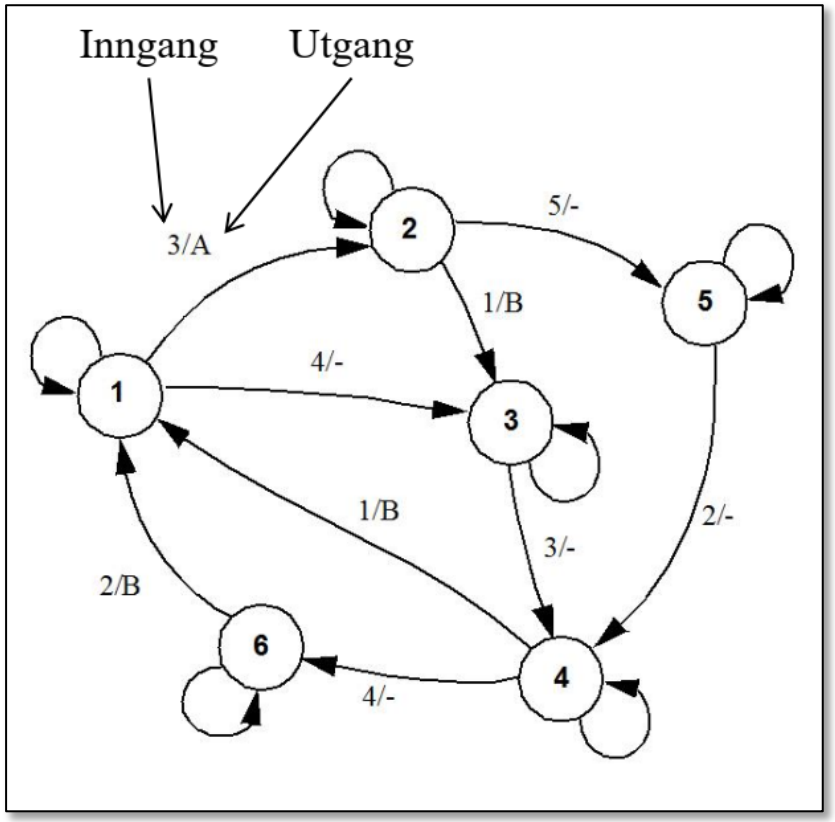


Fig. 27 . Gardintrekk-mekanisme, skjematisk

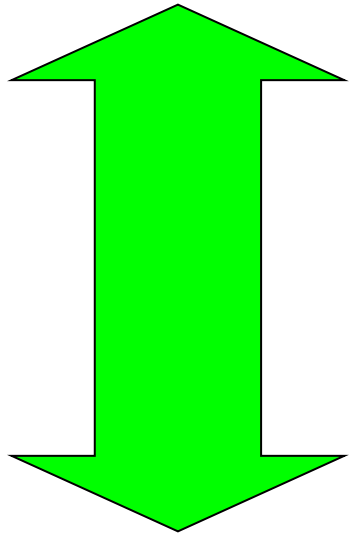


Tilstandsdiagram kan vise oppførsel på forskjellige nivå

- Hele systemet
- Et delsystem
- Instanser av en gitt klasse

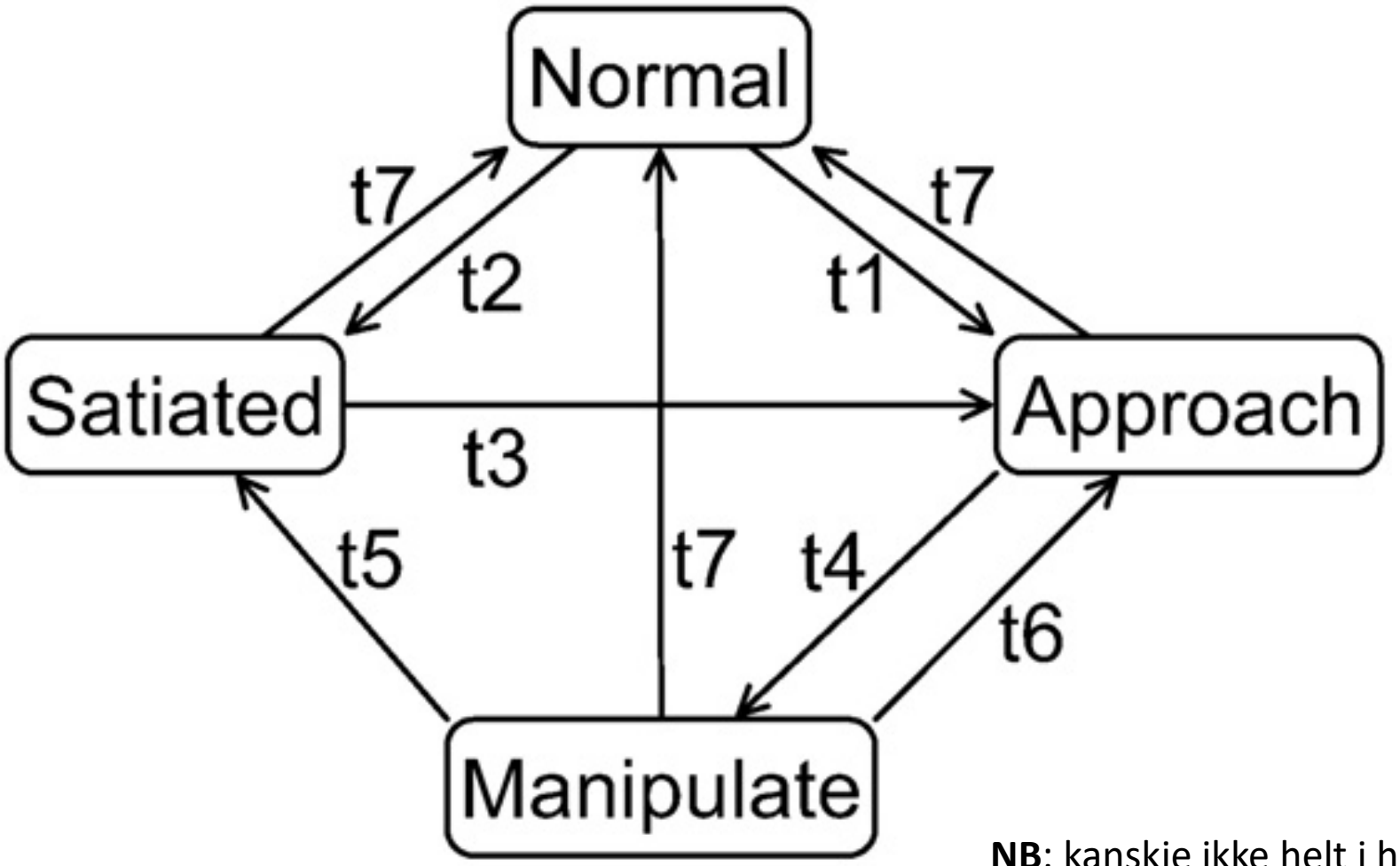
Forskjellige abstraksjons- og detaljnivå

Spesifikasjon/oppførsels-modell, fjernt fra koden



Kode-nært

Eksempel (?): fiskemodell



NB: kanskje ikke helt i henhold til standarden

Eksempel: styring av kran

Styr kran

Precondition: Kranfører i setet, kranen er i ro, tilkoplet strøm og klar til å bli betjent.

Trigger: Dødmannsknapp aktivert og joystick i nøytralposisjon

Suksess-scenario:

1. Kranfører beveger joystick
2. Kranens motorer kjøres med hastighet gitt av joystickens vinkelutslag
3. Dødmannsknapp slippes
4. Kranen stopper.

Utvidelser:

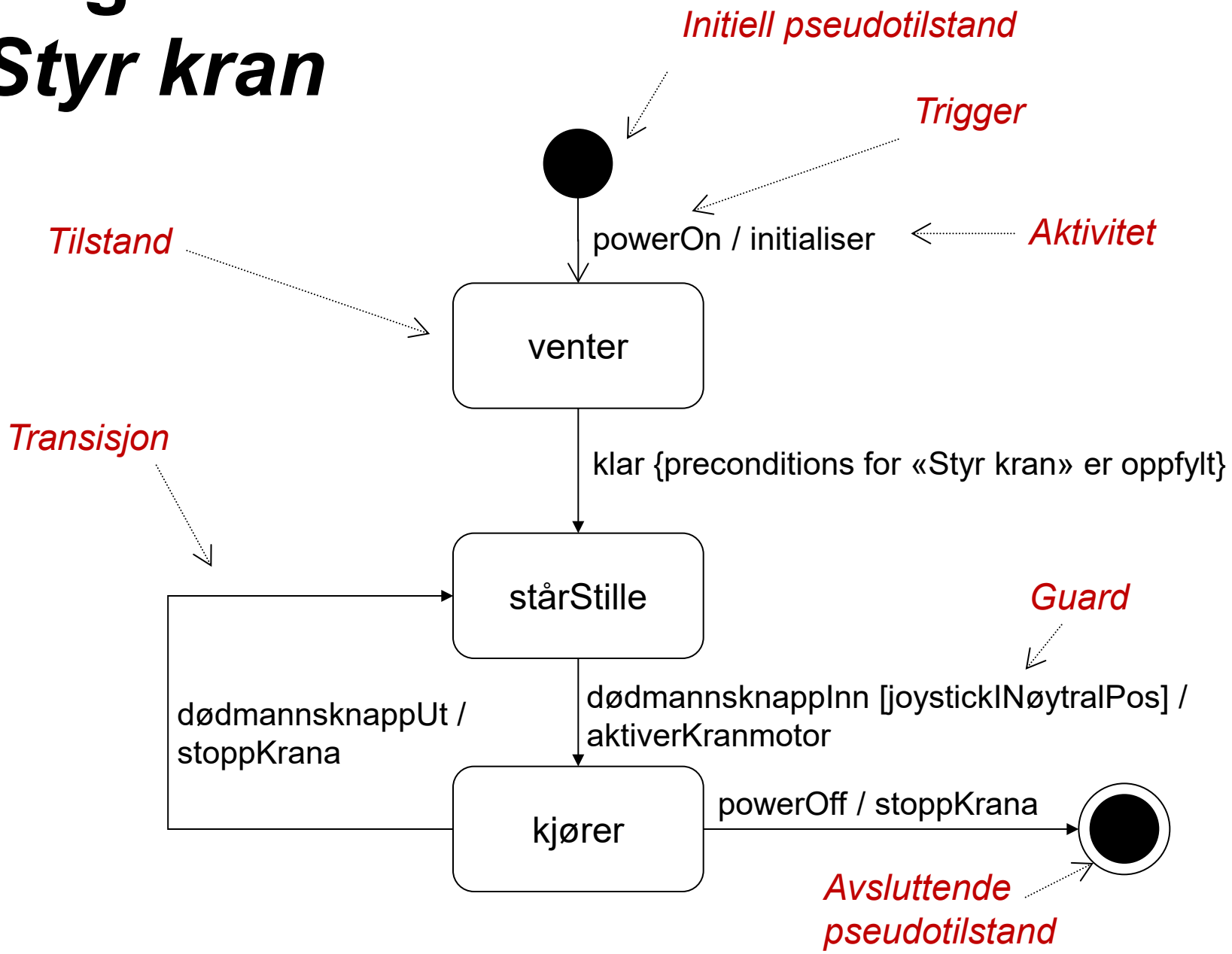
1a: Dødmannsknapp slippes
1a.1 Hopp til punkt 4

Suksessgaranti:
Samme som Precondition

Minimal garanti:
Kran i ro

- Use case-beskrivelsen omfatter ett eller flere scenarier
- Et komplett tilstandsdiagram må muliggjøre *alle* scenariene!

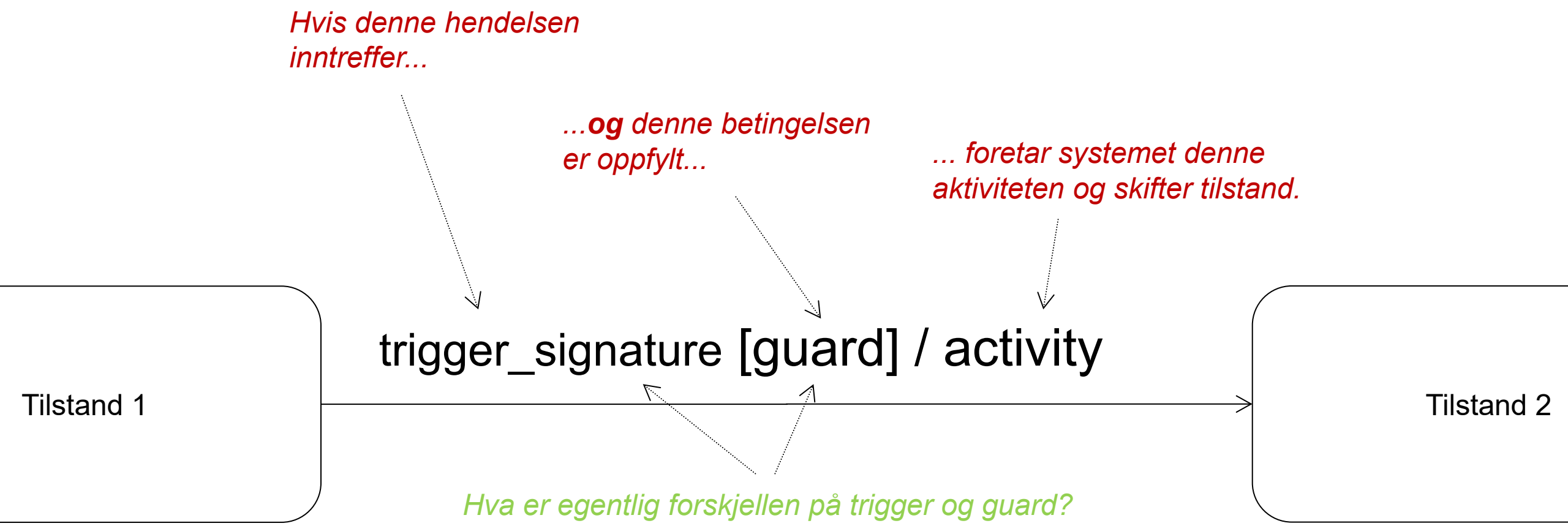
Tilstandsdiagram for use case *Styr kran*



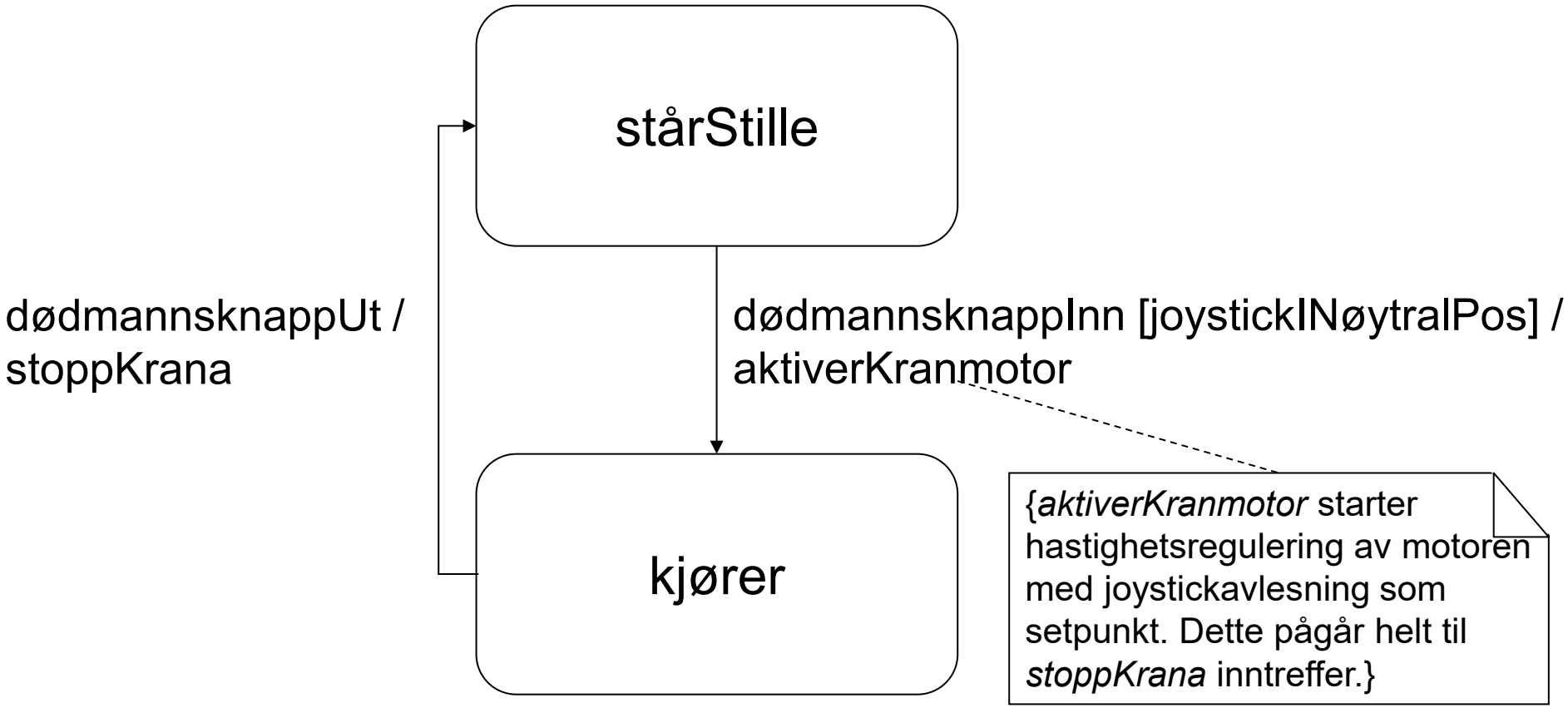
Transisjoner - syntaks

Transisjoner fra en tilstand skal være gjensidig utelukkende – dvs. bare en kombinasjon av **trigger** og **guard** kan oppfylles samtidig.

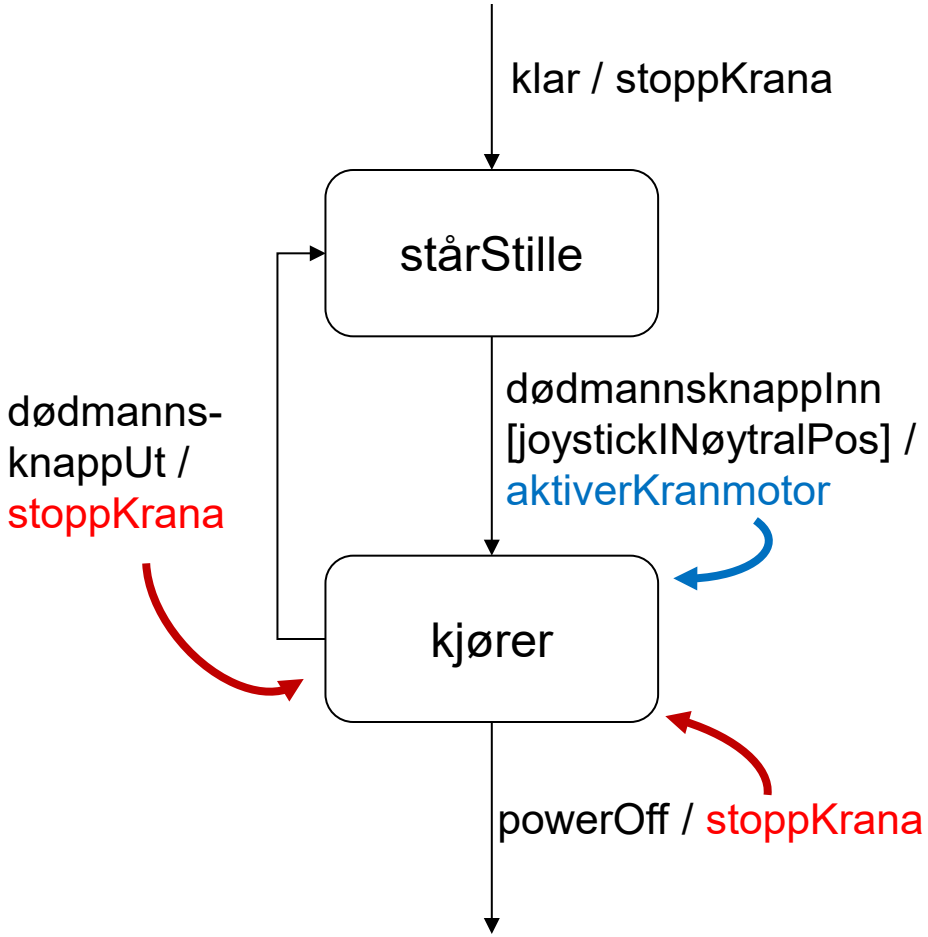
Transisjoner - syntaks



Transisjoner - eksempel



Interne aktiviteter



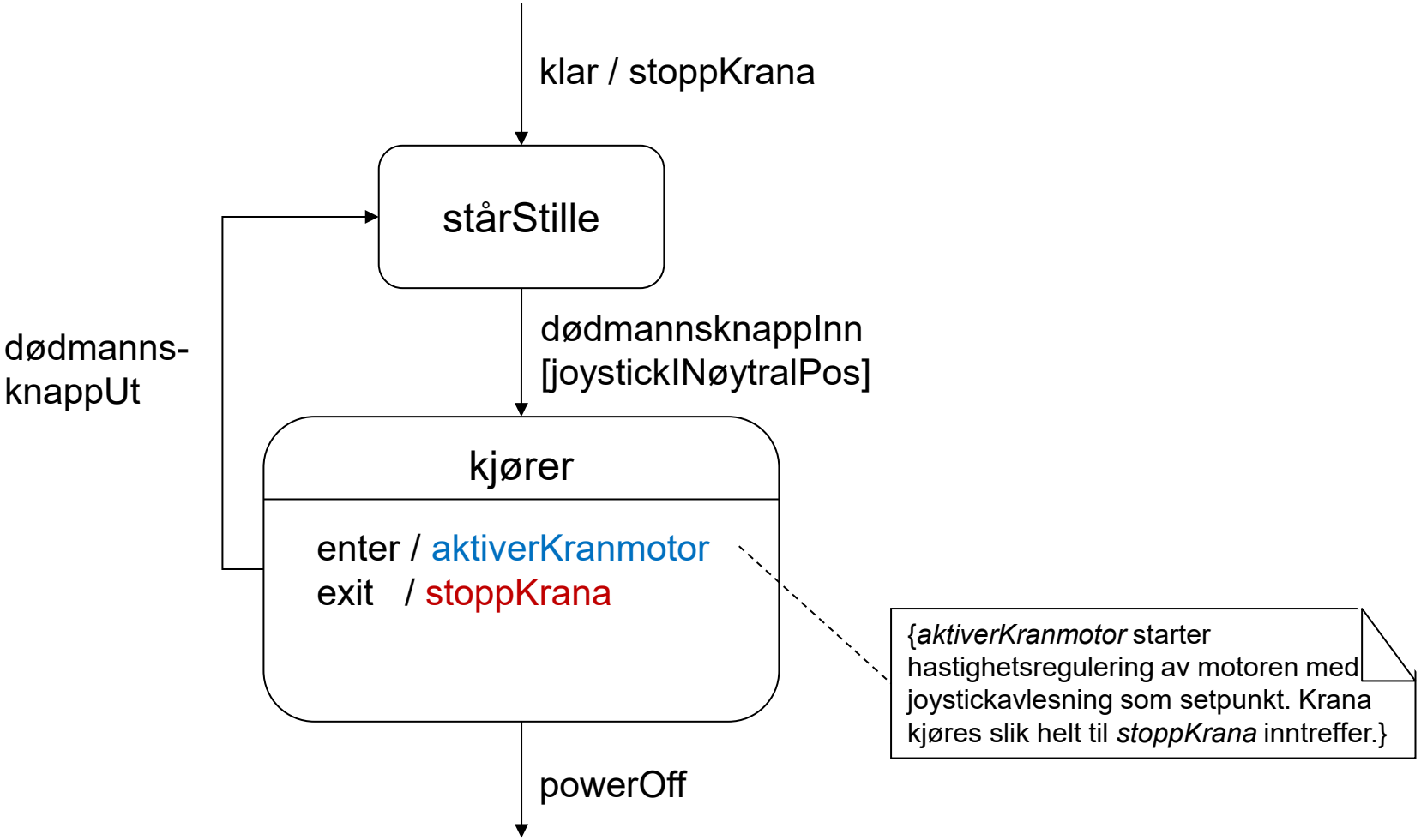
Alle transisjonene ut fra tilstanden *kjører* medfører aktiviteten *stoppKrana*.

Alle transisjonene inn til tilstanden *kjører* medfører aktiviteten *aktiverKranmotor*.

Da kan vi forenkle!

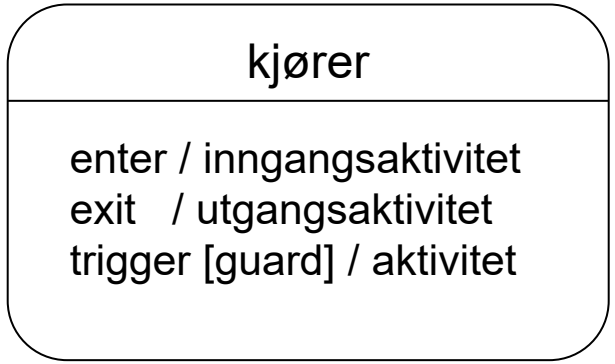
Interne aktiviteter

UML lar oss uttrykk dette mer kompakt som *interne aktiviteter*:



Interne aktiviteter

Interne aktiviteter defineres i et eget felt under navnet på tilstanden:

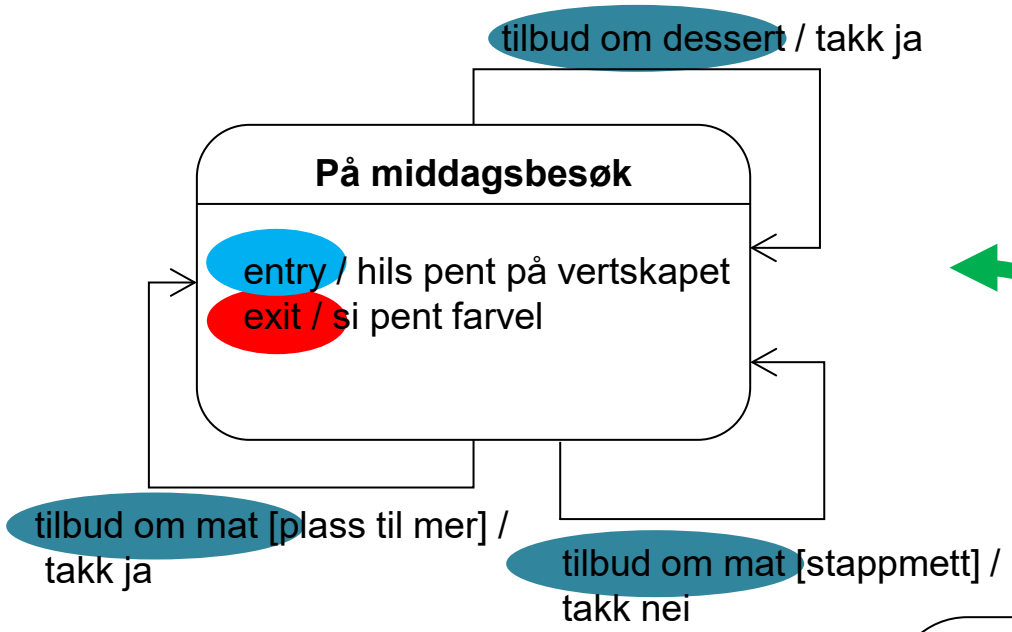


Hver av disse kan ha trigger, guard og navn på aktivitet.

Spesielle triggere er:

- **enter**: transisjon inn til tilstanden
- **exit** : transisjon ut fra tilstanden
- **do**: transisjon inn til tilstanden (og utløser ny transisjon når den blir fullført)

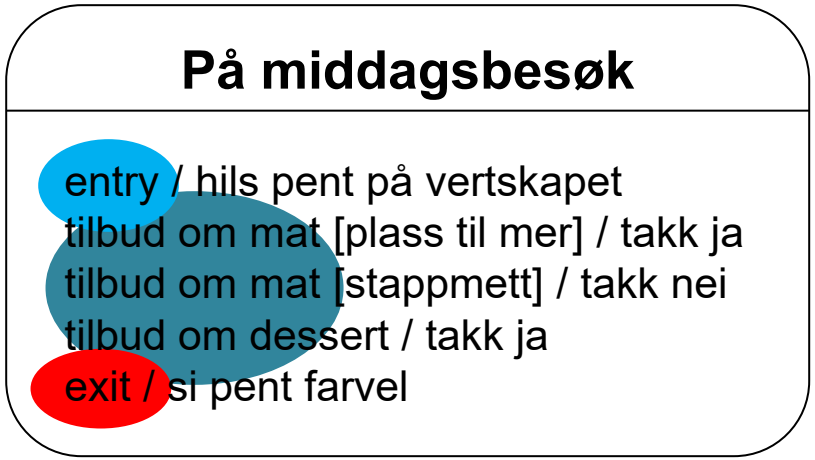
Interne aktiviteter vs. selvtransisjoner



Virker disse modellene på samme måte?

(Hvorfor/hvorfor ikke?)

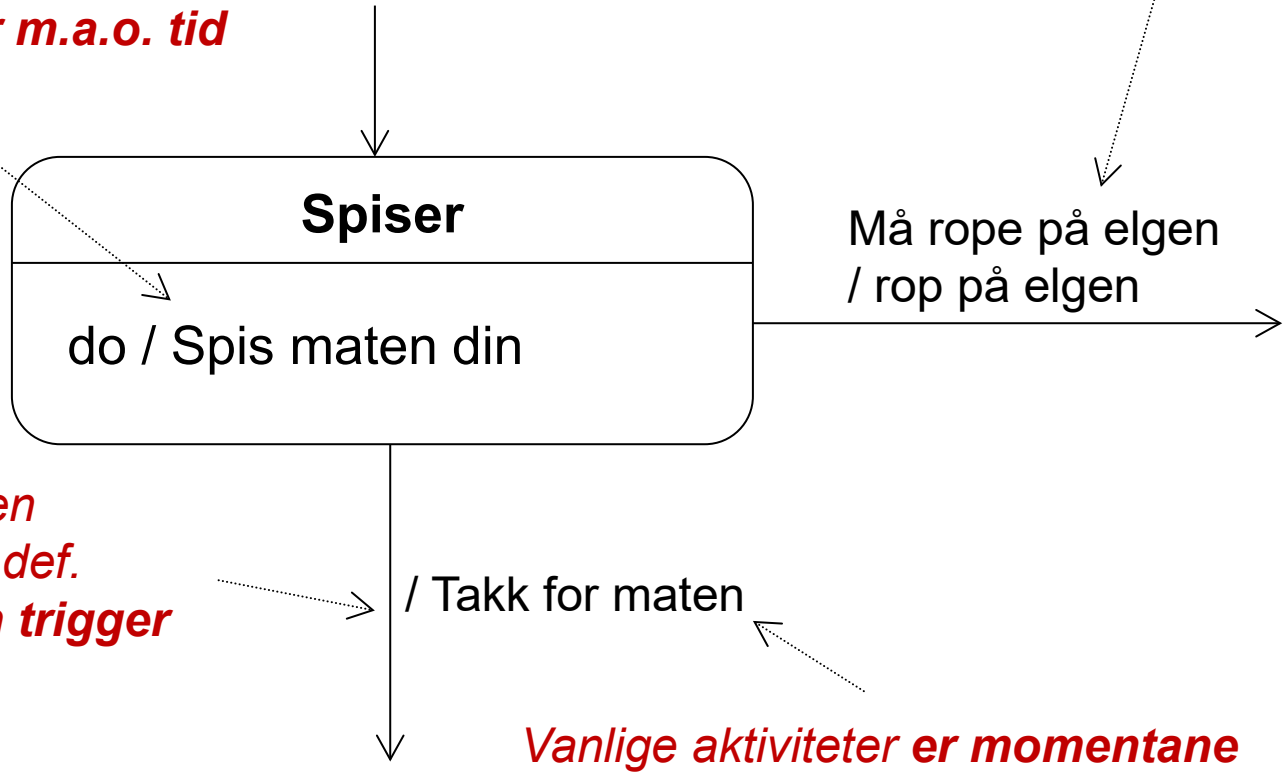
Nei: interne aktiviteter trigger ikke entry- og exit-aksjoner!



Aktivitetstilstander og *do-activities*

*En **Do-aktivitet** er en aktivitet der systemet er i **kontinuerlig aktivitet.** Do-aktiviteter **tar m.a.o. tid***

*Do-aktiviteter er pr.def. **preemptive,** dvs. de kan **avbrytes** av en annen transisjon*

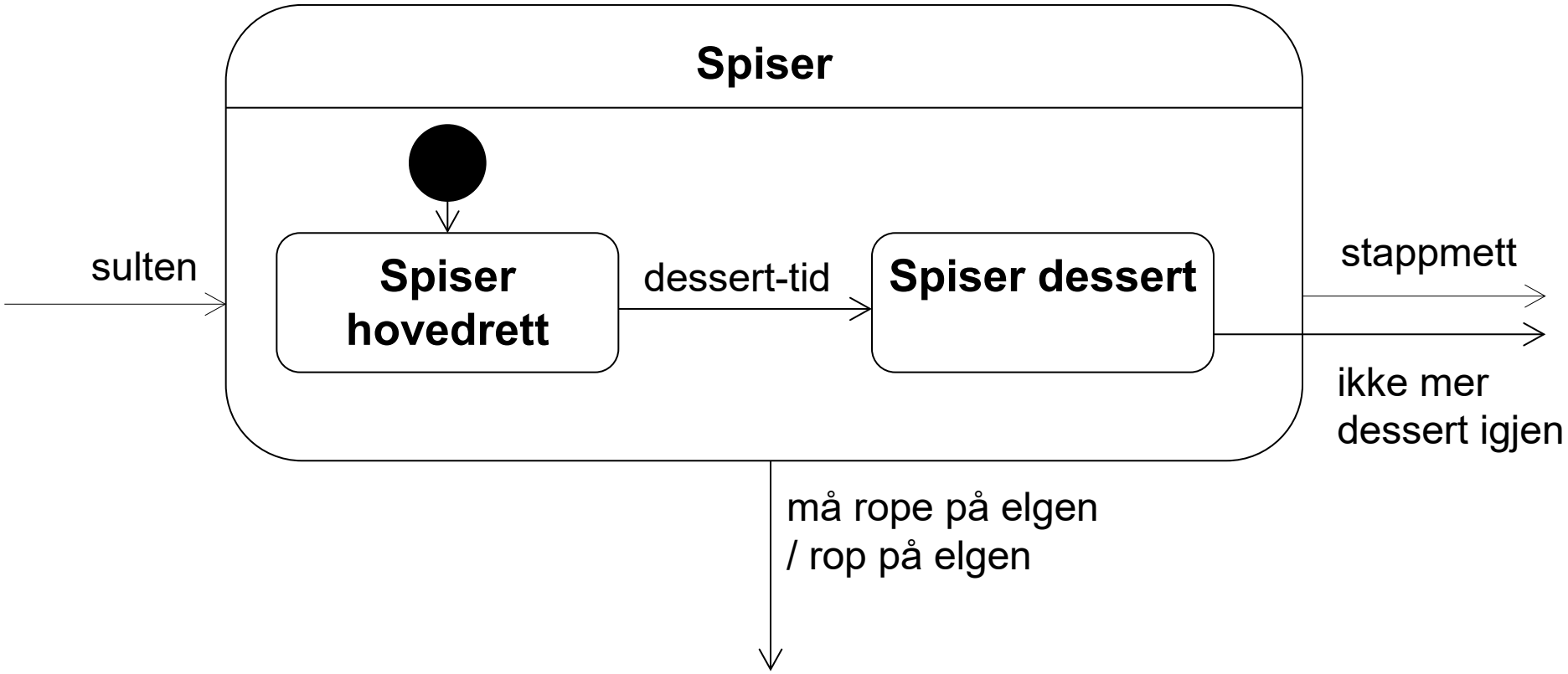


*Når/hvis aktiviteten er ferdig, skjer pr.def. transisjonen **uten trigger***

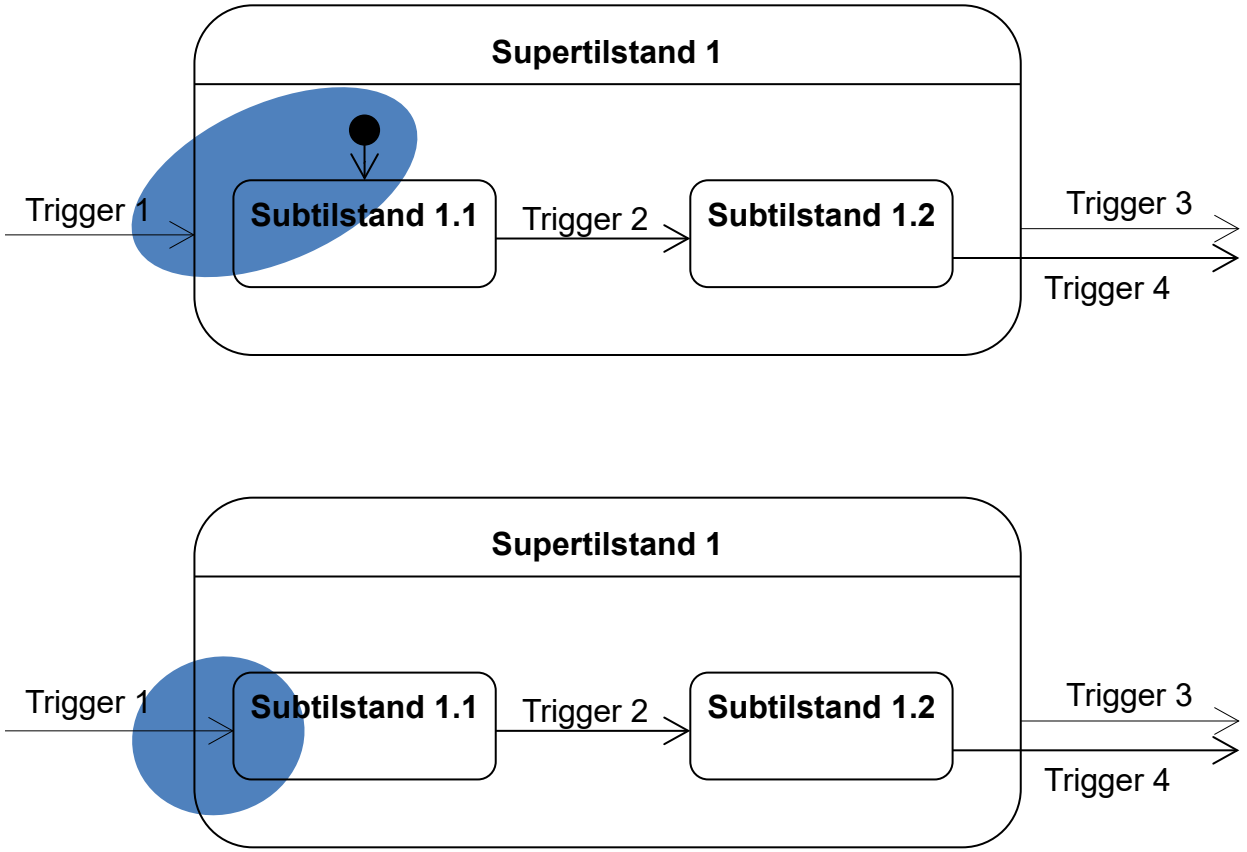
*Vanlige aktiviteter **er momentane** (derfor ikke preemptive).*

Nøstede tilstander

Hvis flere tilstander har samme transisjoner og interne aktiviteter, kan de grupperes i en *supertilstand* (her: *Spiser*)



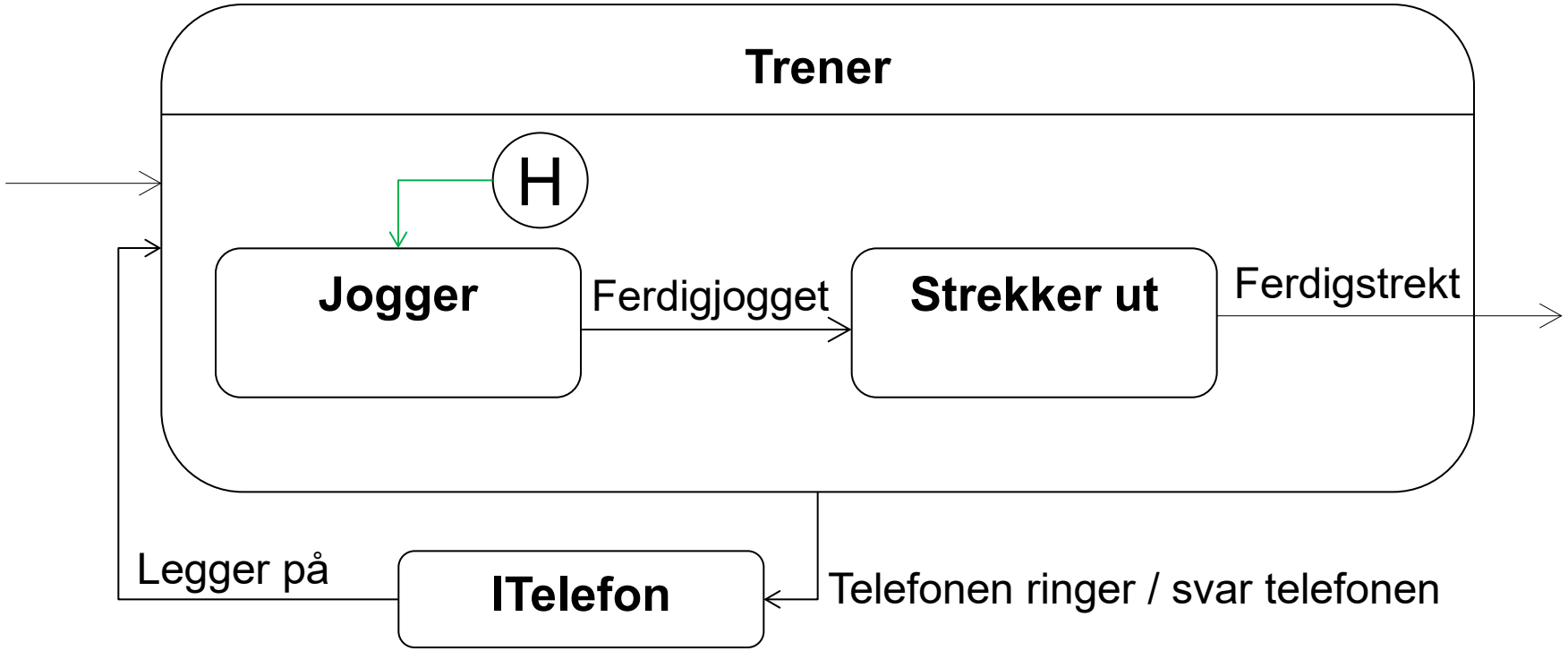
Disse er ekvivalente:



History – pseudotilstand: *husker undertilstanden*

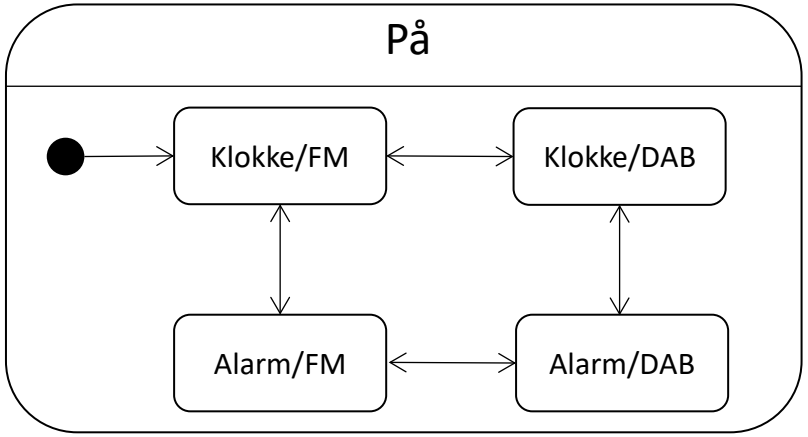
Systemet starter i den siste undertilstanden den var i

Pilen følges hvis historie ikke finnes (dvs. første gang supertilstanden entres)



Ortogonale tilstander

- Systemet vi modellerer kan ha *ortogonale* tilstander – dvs. separate valg som ikke påvirker hverandre
- F.eks.:
  - Klokkeradio som enten viser klokkeslett eller alarmtid, og (samtidig) enten spiller FM- eller DAB-radio
  - Tastatur som kan være i normal eller caps-lock-modus, og med det numeriske tastaturet i numerisk eller piltast-modus
- Dette kan gi uforholdsmessig mange tilstander i diagrammet:

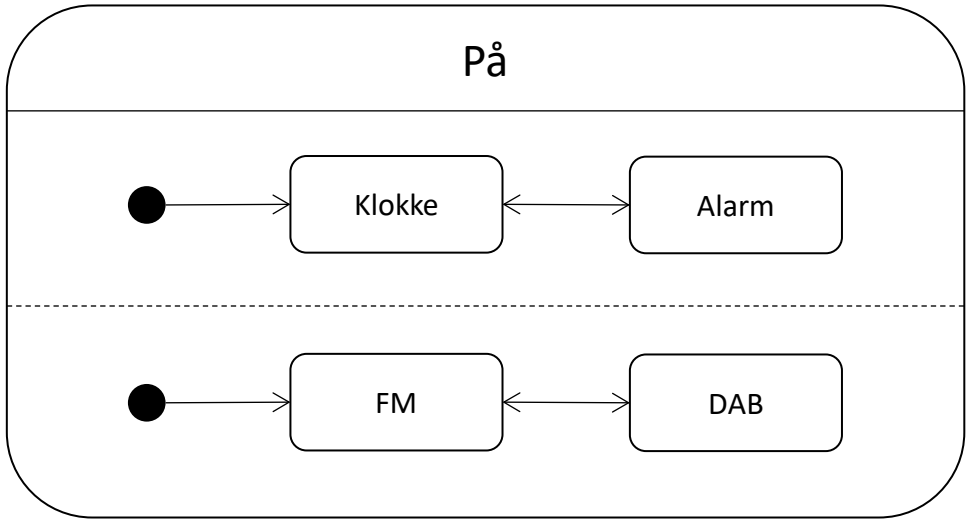


Hva skjer om vi legger til AM som radioalternativ?

Eller med et nytt valg: skjermbelysning av/på?

Ortogonale tilstander

UML har derfor egen notasjon for ortogonale parallelle tilstander:



Dersom det innføres flere ortogonale valg trenger vi bare flere seksjoner

Merk: ortogonale tilstander impliserer ikke nødvendigvis parallelle tråder i systemet

Gardintrekkmekanisme

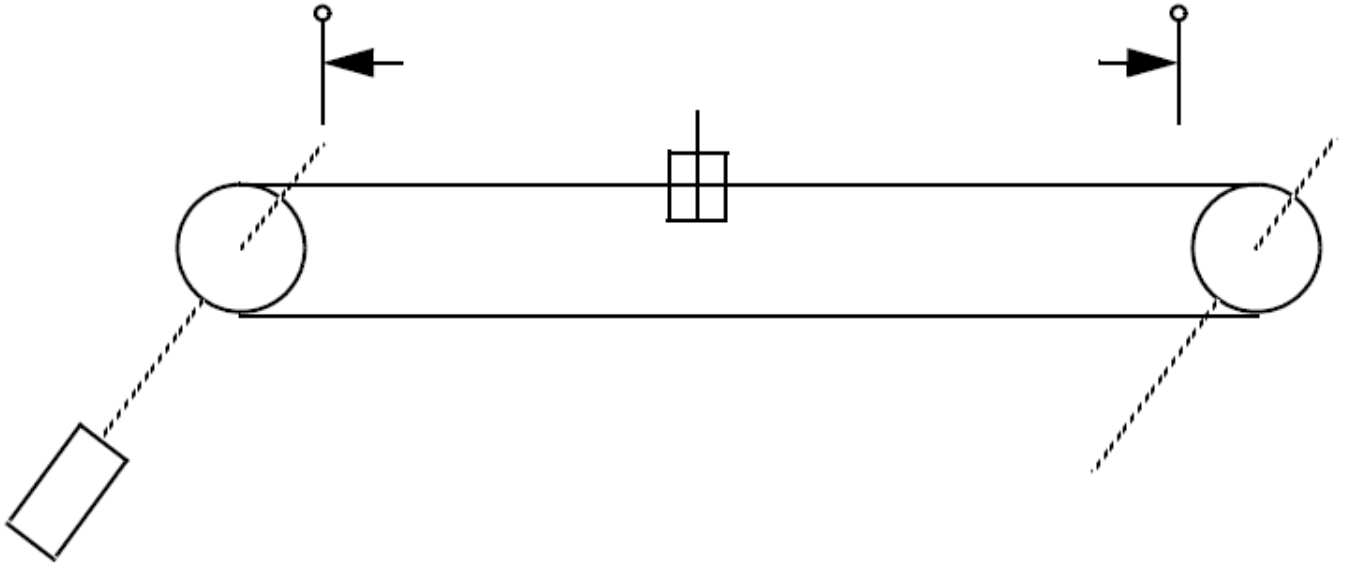


Fig. 27 . Gardintrekk-mekanisme, skjematisk

Tre knapper:

◄ Venstre

► Høyre

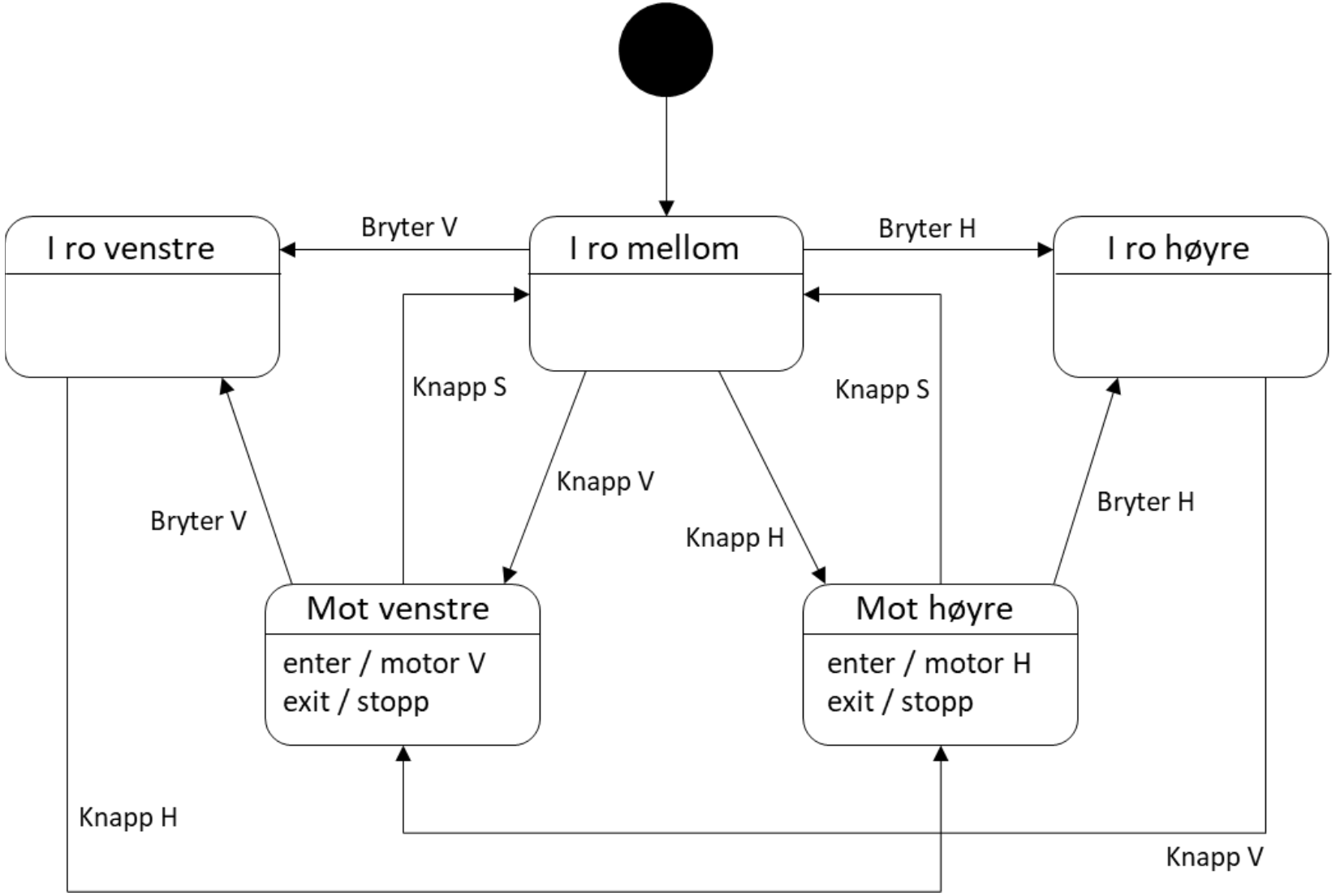
Stopp

Gardintrekkmekanisme

Tilstander **q**	Innganger				
	Trykknapper			**Endebrytere**	
	←	→	**STOPP**	**venstre**	**høyre**
	1	2	3	4	5
1. I ro venstre	1/-	5/H	1/-	1/-	(1/-)
2. I ro høyre	4/V	2/-	2/-	(2/-)	2/-
3.I ro mellom	4/V	5/H	3/-	1/-	2/-
4. Mot venstre	4/-	4/-	3/S	1/S	(3/S)
5. Mot høyre	5/-	5/-	3/S	(3/S)	2/S

Fig. 28 Huffman-tabell for gardintrekk-mekanismen

Tilstandsmaskindiagram for gardintrekkmekanismen

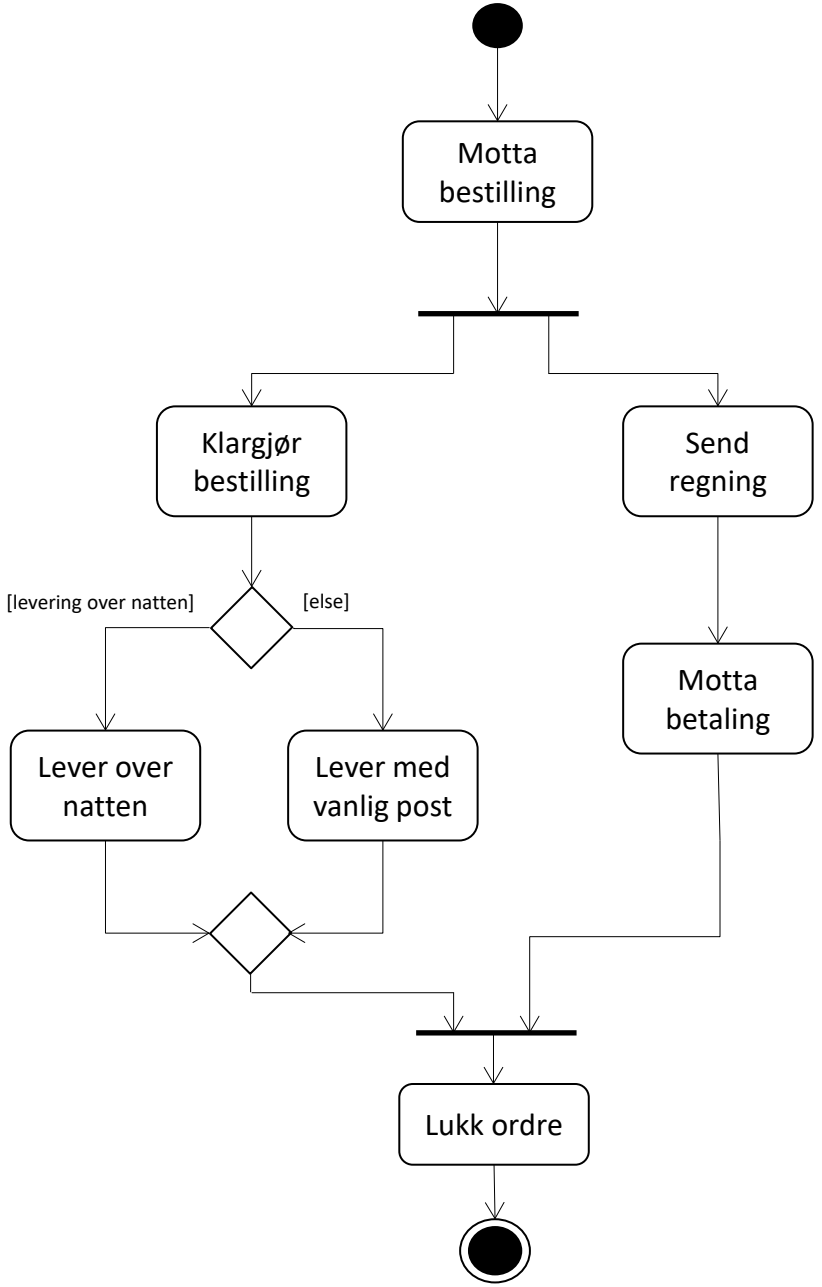


Aktivitetsdiagram

Aktivitetsdiagram

Aktivitetsdiagram beskriver en aktivitet sammensatt av sekvensielle og parallelle aksjoner og beslutninger

Tilsvarer «flytskjema» som finnes i varianter utenfor UML-standarden



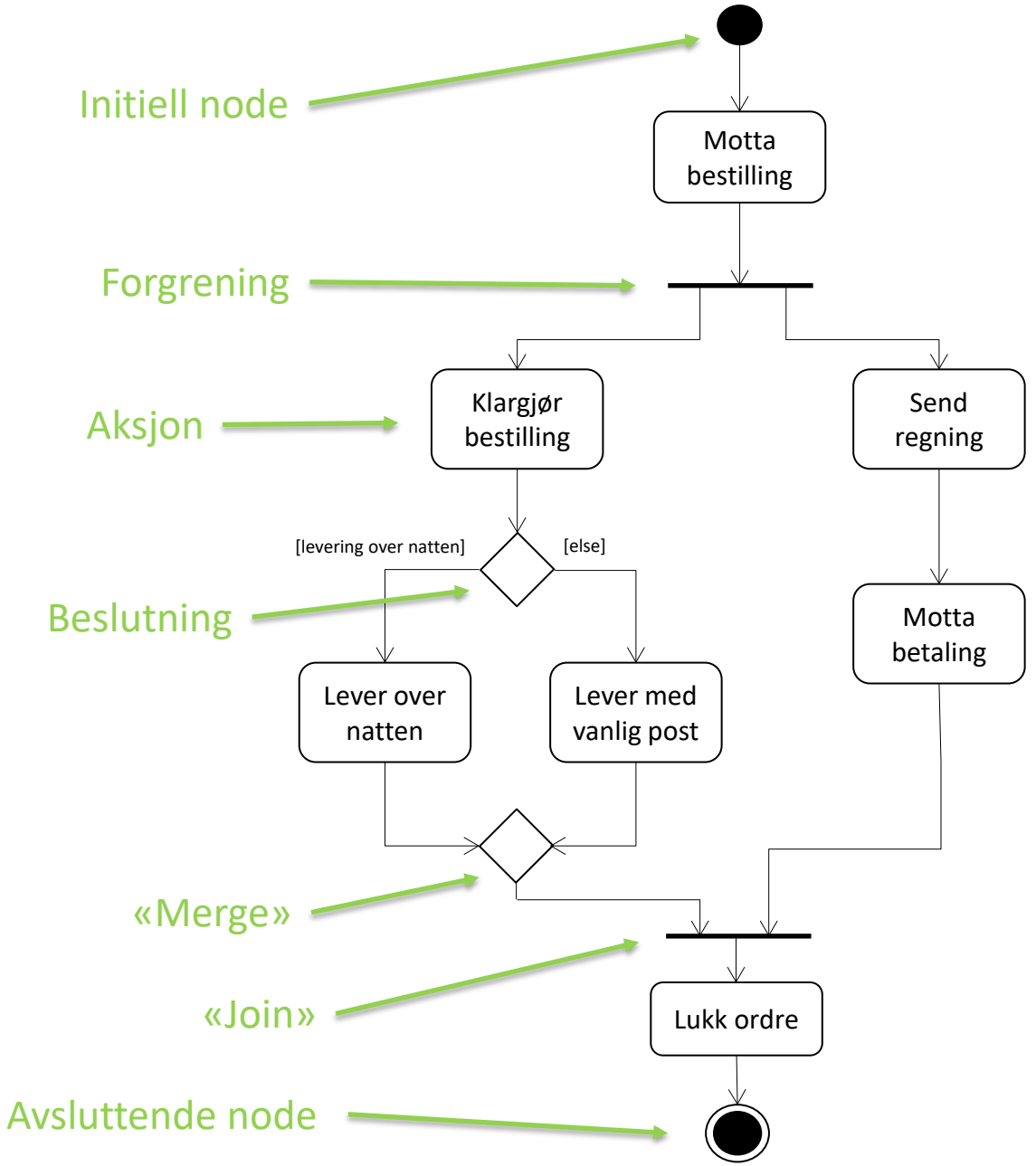
Notasjon

Forgrening: alle grenene utføres i parallell

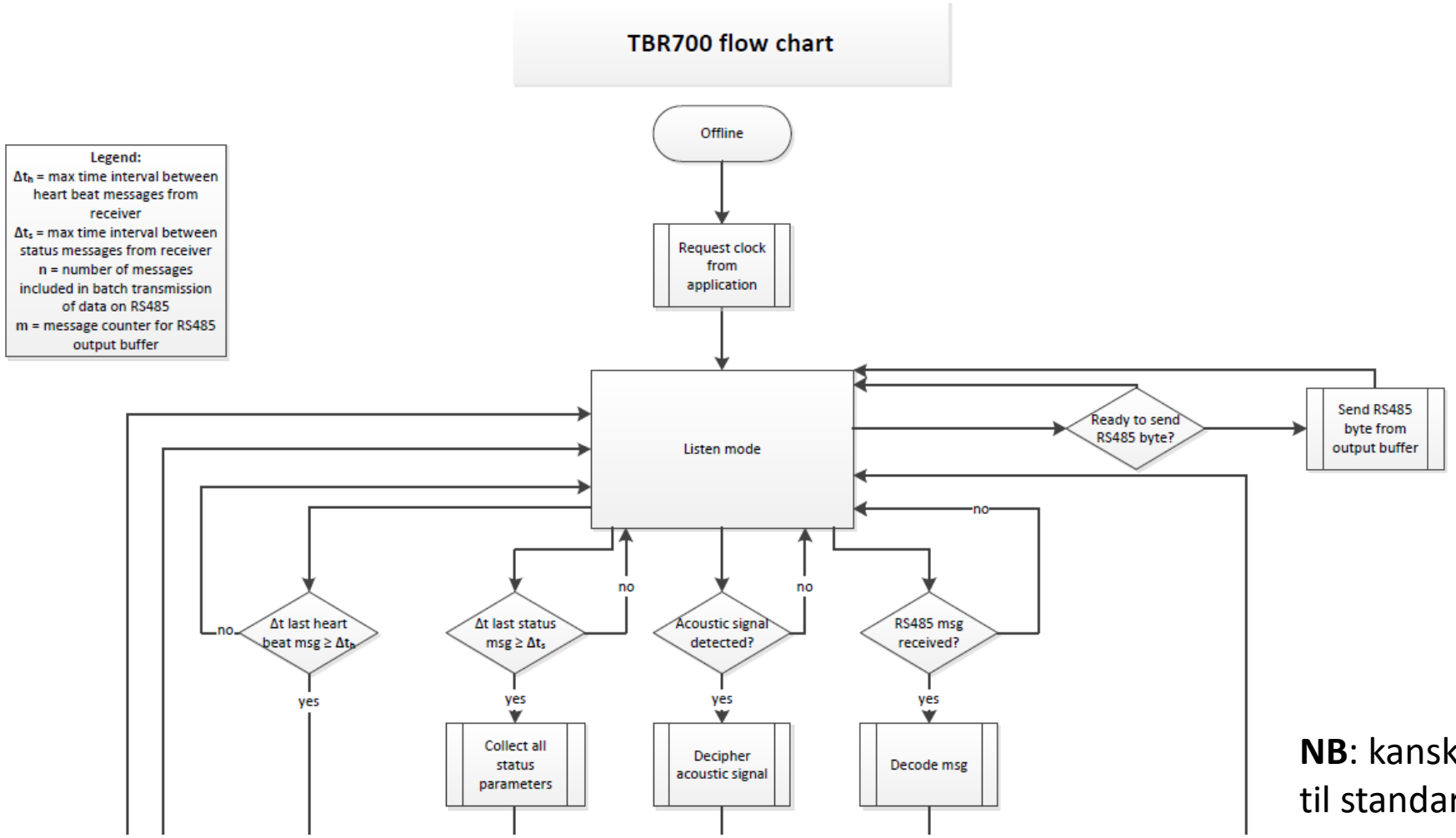
Beslutning: bare den valgte grenen utføres

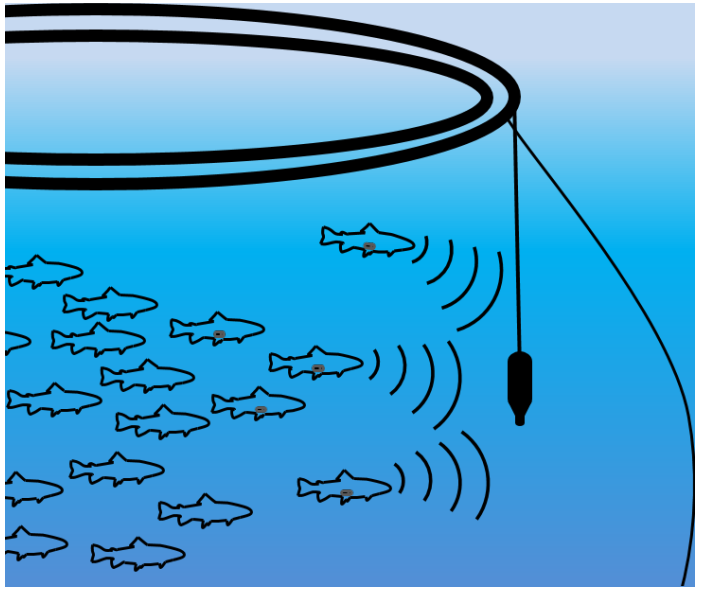
Merge: Flyten går videre når den valgte grenen er ferdig.

Join: Flyten går videre når alle grenene er ferdige.



Eksempel (?): mottaker for telemetridata





NB: kanskje ikke helt i henhold til standarden

Eksempel: babycall

Baby-enheten skal bl.a.

- Måle lydnivå og temperatur
- Sende temperaturmåling via radio til foreldreenheten
- Hvis lydnivået overstiger et gitt nivå, send lydstrøm via radio

Foreldre-enheten skal bl.a.

- Spille av evt. lydstrøm
- Motta målt temperatur T og vise på displayet
- Gi alarm hvis T er utenfor gitte grenser

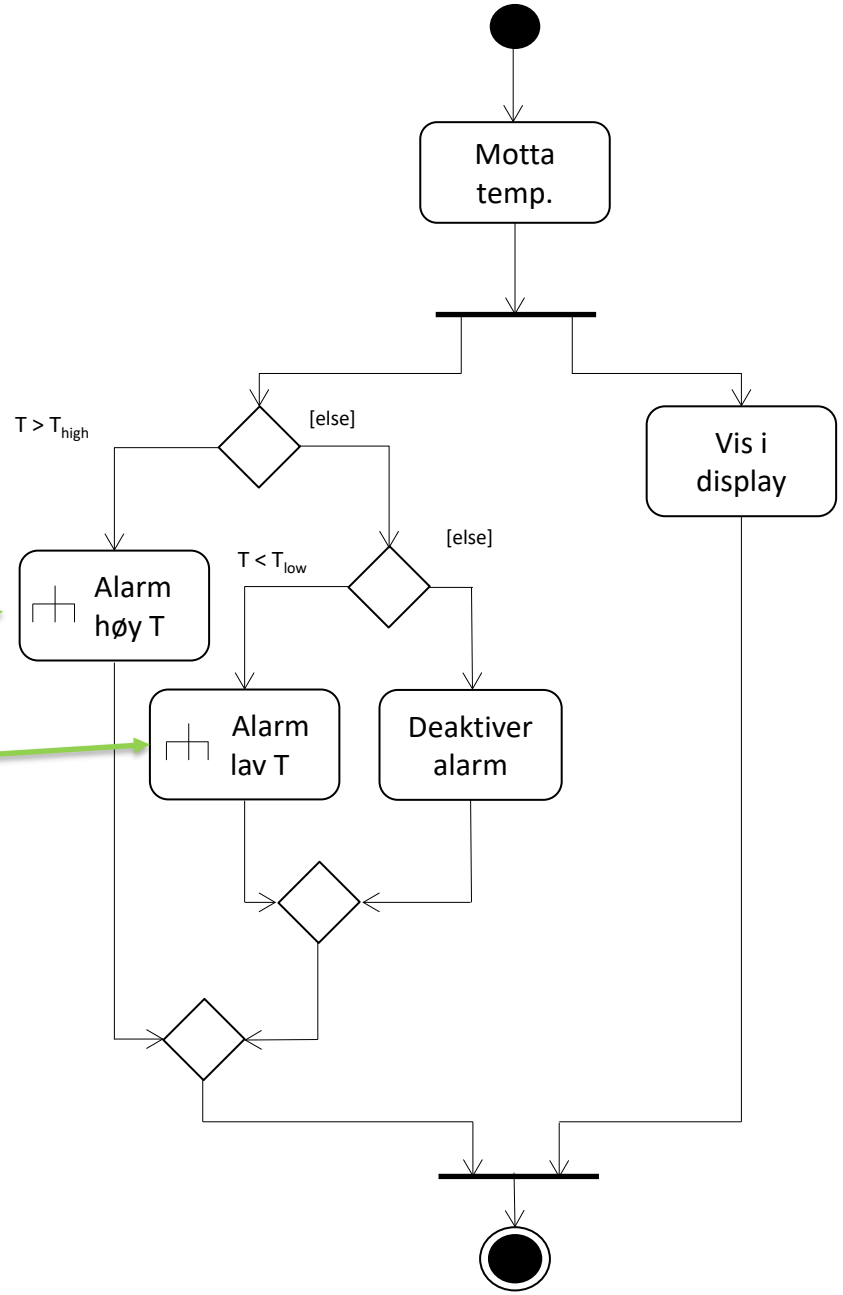
Delaktiviteter

Alarm består i:

1. Vise tekst i display
2. Lage lydsignal
3. Blinke med displaylyset

Aksjonene «Alarm høy T» og «alarm lav T» er derfor sammensatte. For å begrense kompleksiteten i diagrammet kan vi definere disse i egne diagram.

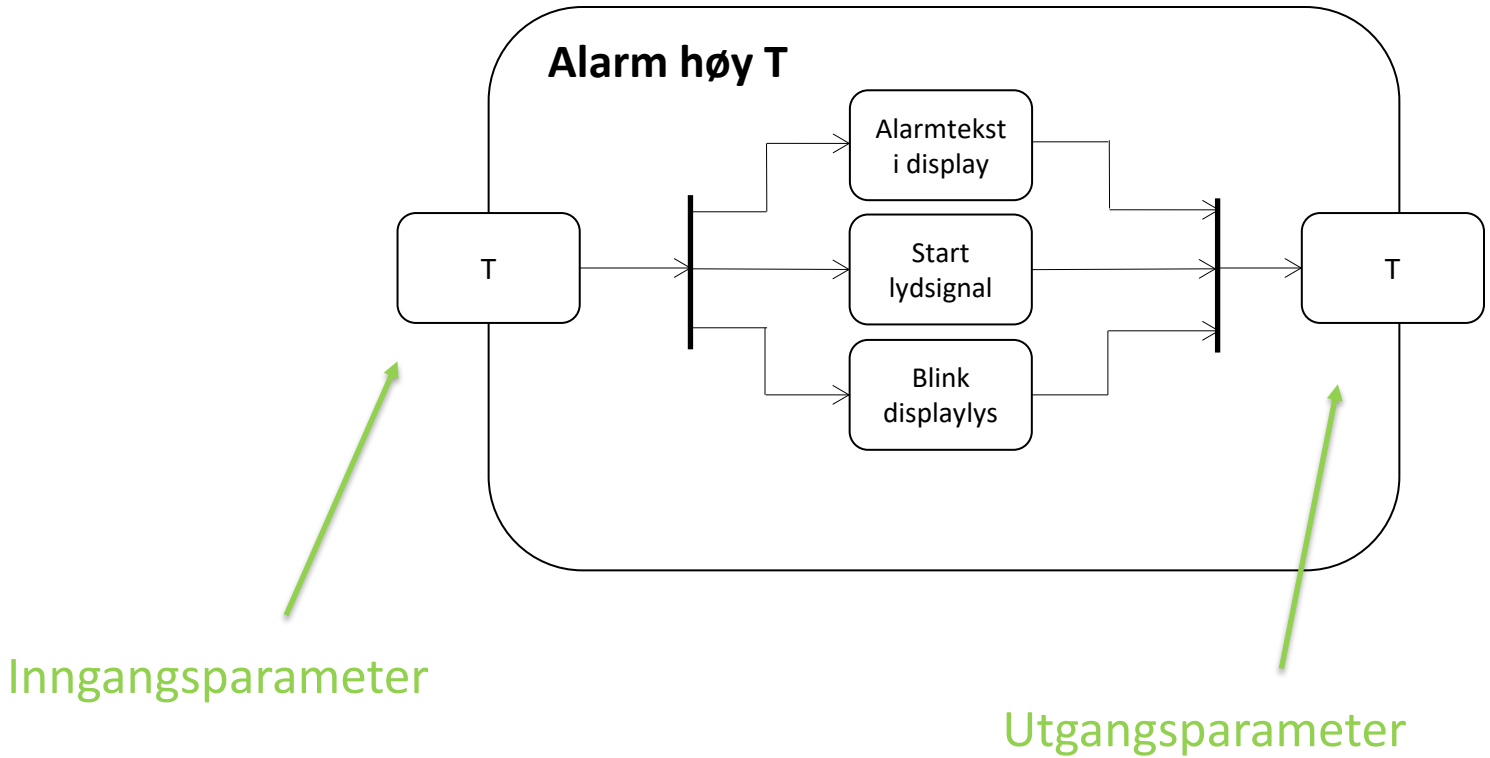
Sammensatte aksjoner markeres med



Delaktiviteter

Diagram for «Alarm høy T»

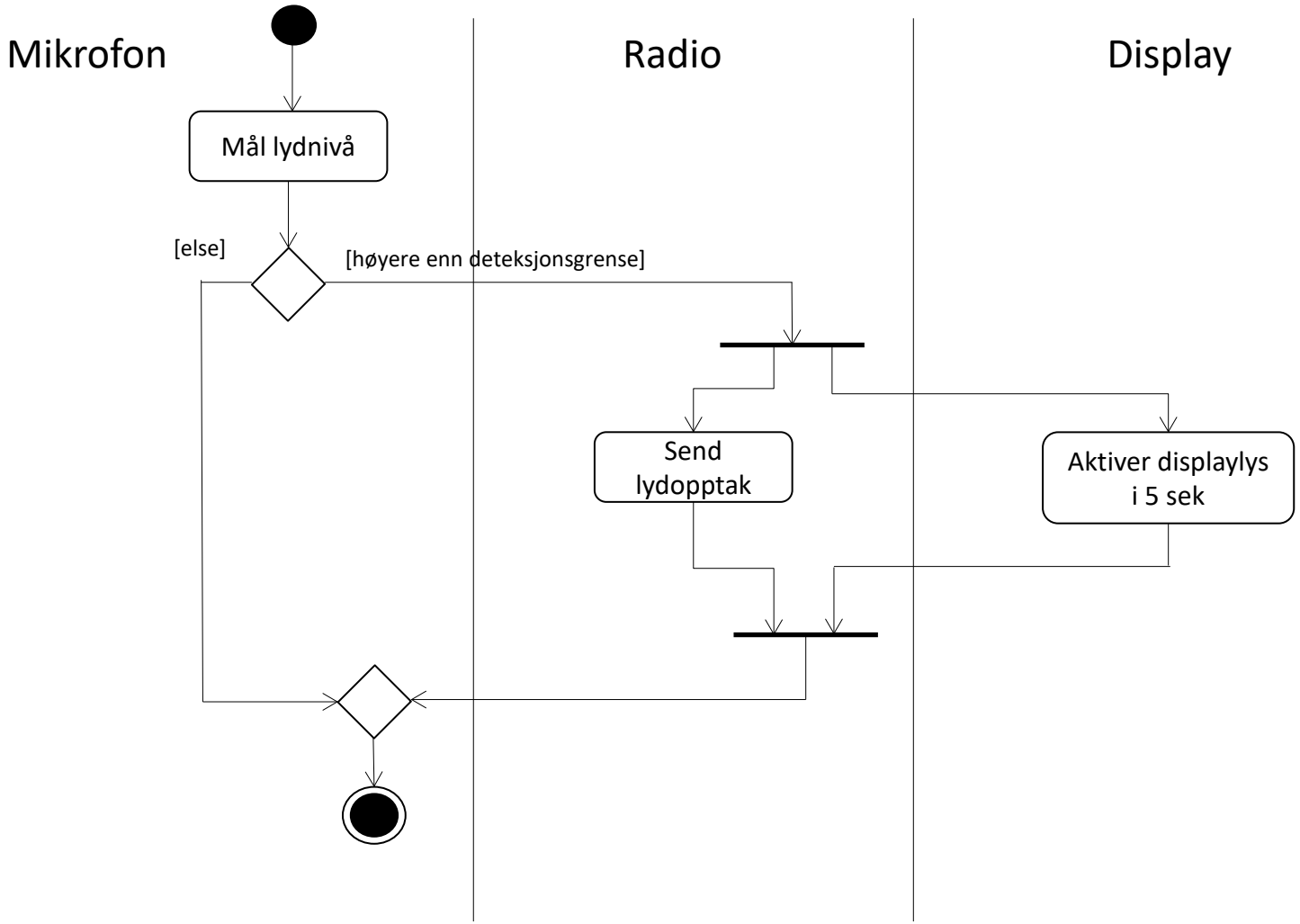
De tre aksjonene som inngår kan utføres parallelt eller i vilkårlig sekvens



Partisjoner

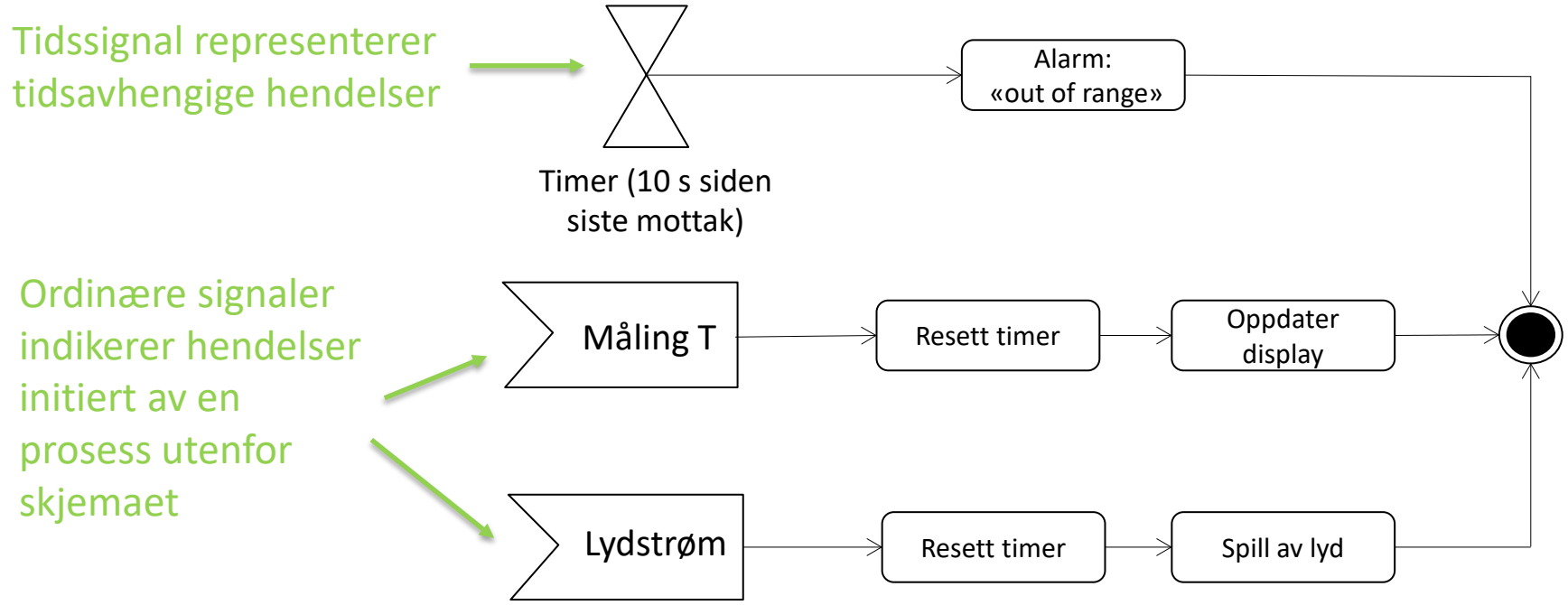
Partisjoner («svømmebaner») kan brukes for å indikere hvilke klasser som utfører de forskjellige aktivitetene

Eksempel: babycall (babyenhet)



Signaler

Mottak av signaler (foreldreenhet):



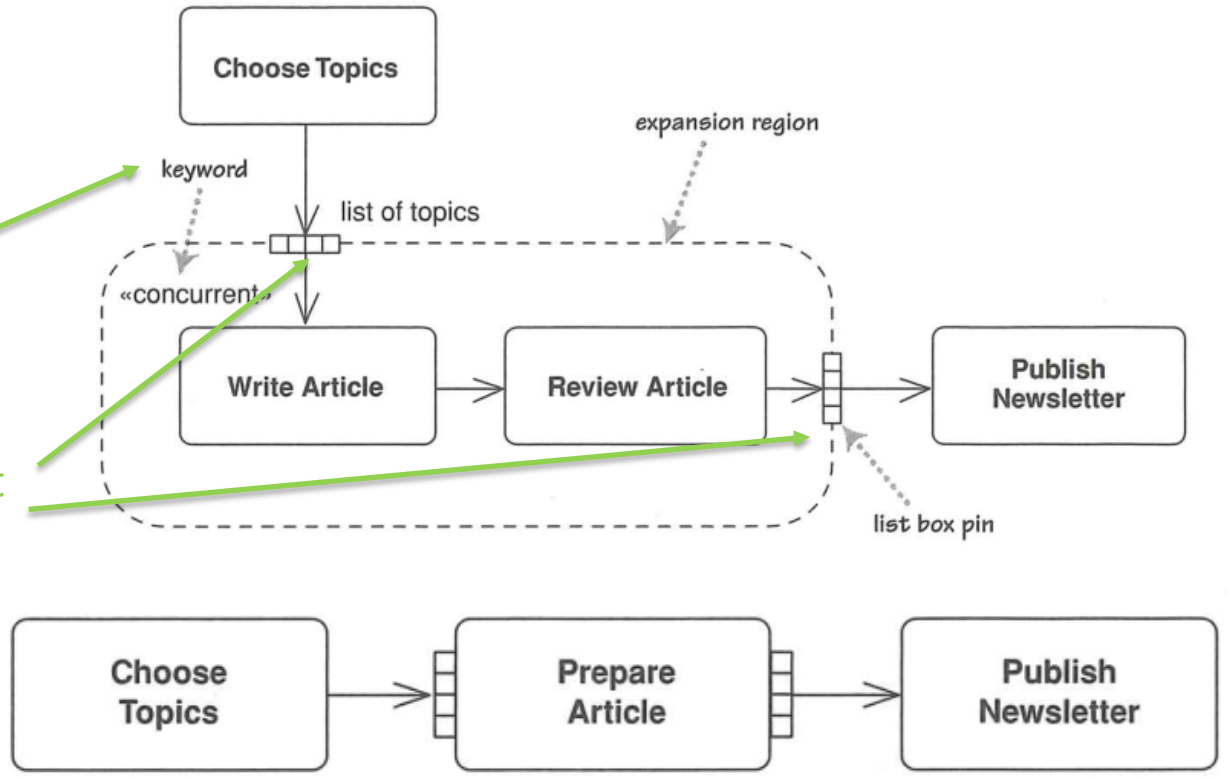
Expansion region

Hvis en aksjon trigger flere parallelle eller sekvensielle (iterative) aksjoner kan man bruke denne notasjonen:

Nøkkelordet angir parallelitet (parallelt eller iterativt)

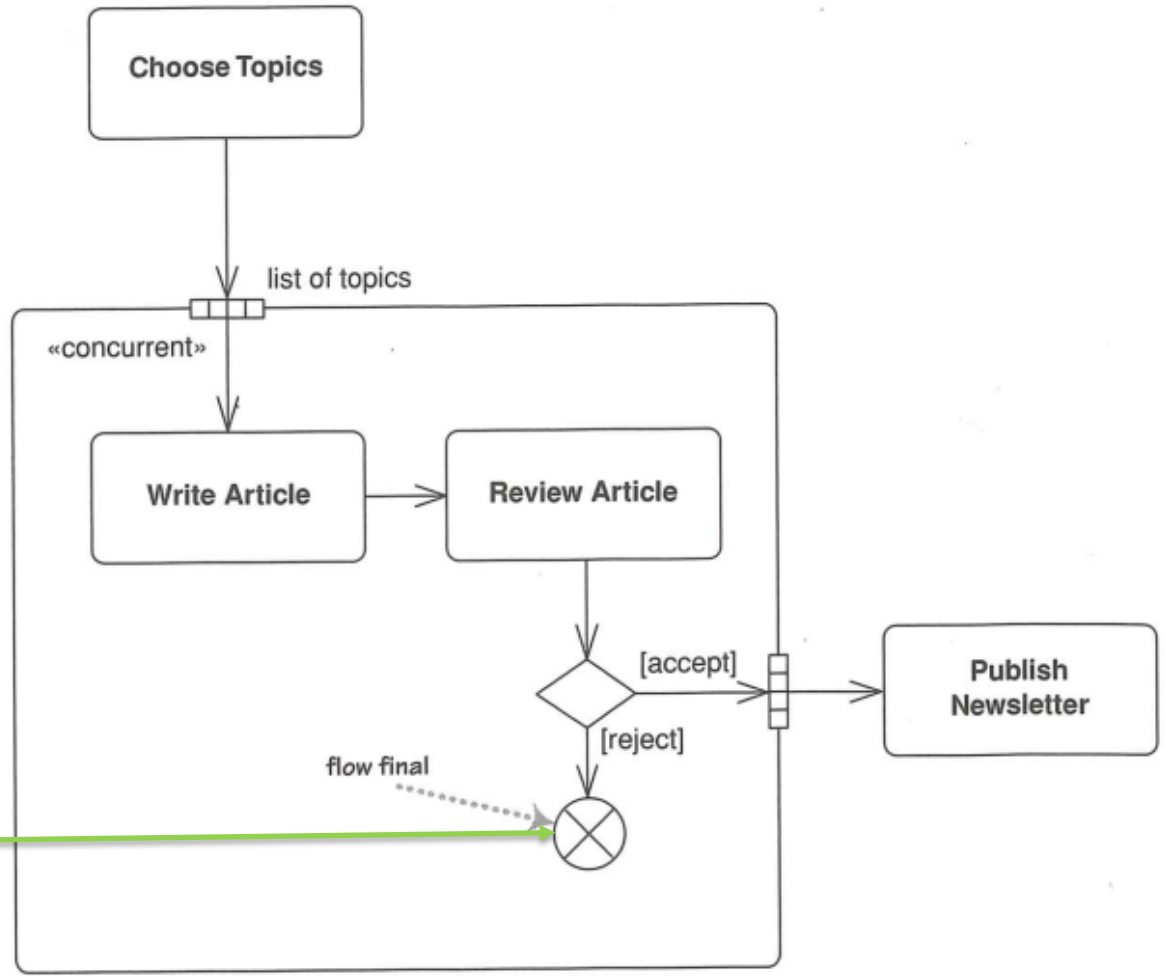
Antall elementer ut tilsvarer antall inn

Kompakt notasjon



Flow final

Hvis flyten avsluttes i enkelte iterasjoner av en ekspansjonsregion kan dette angis slik:



Flyten avsluttes her for noen av artiklene

Bruksområder for aktivitetsdiagram

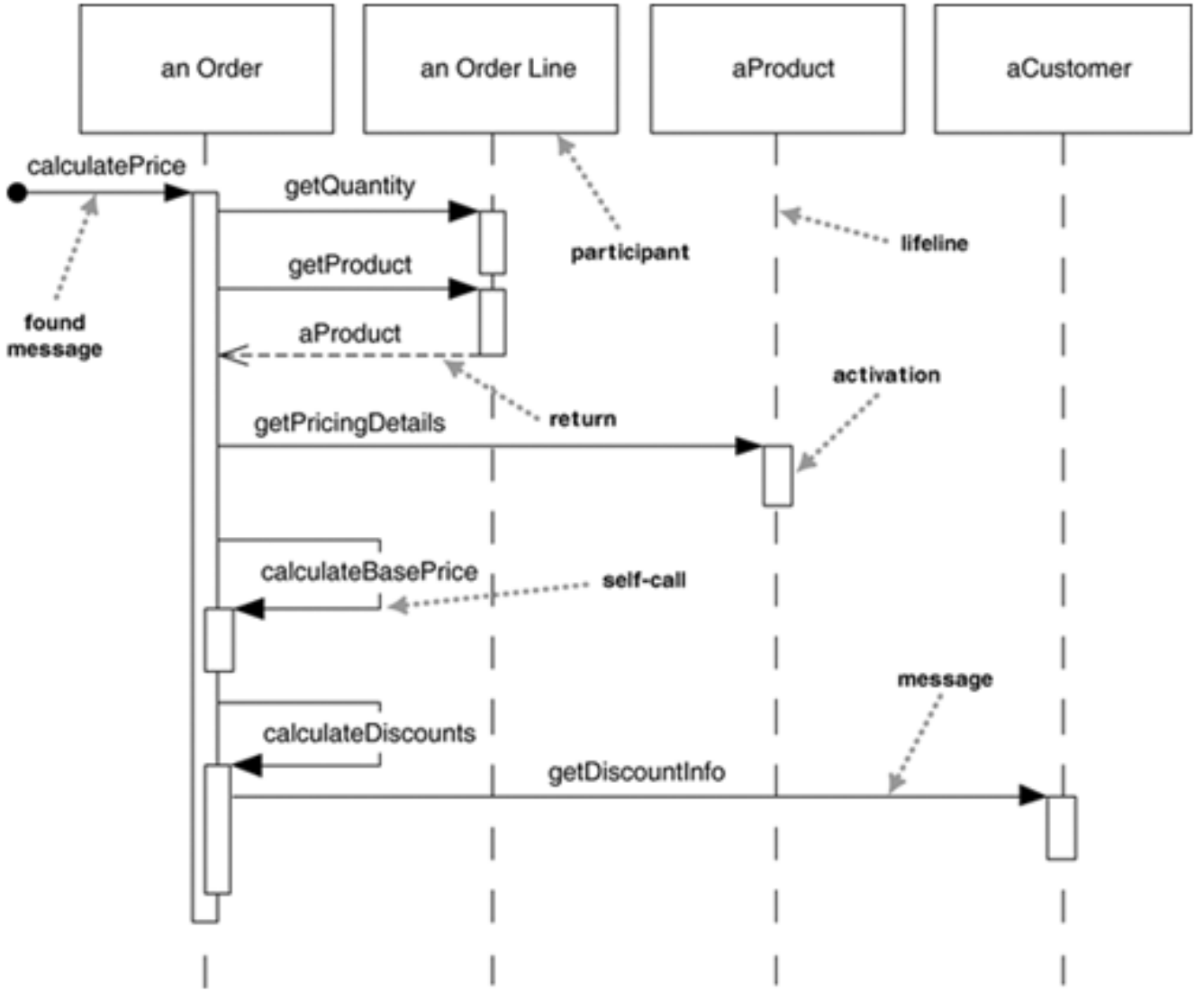
- Formalisert notasjon for flytskjema
- Kan brukes i use case-beskrivelser
- Spesielt egnet for å beskrive arbeidsflyt med parallelle aktiviteter
- … men ikke veldig egnet for å håndtere de spesielle utfordringene man har ved parallell programmering

Sekvensdiagram

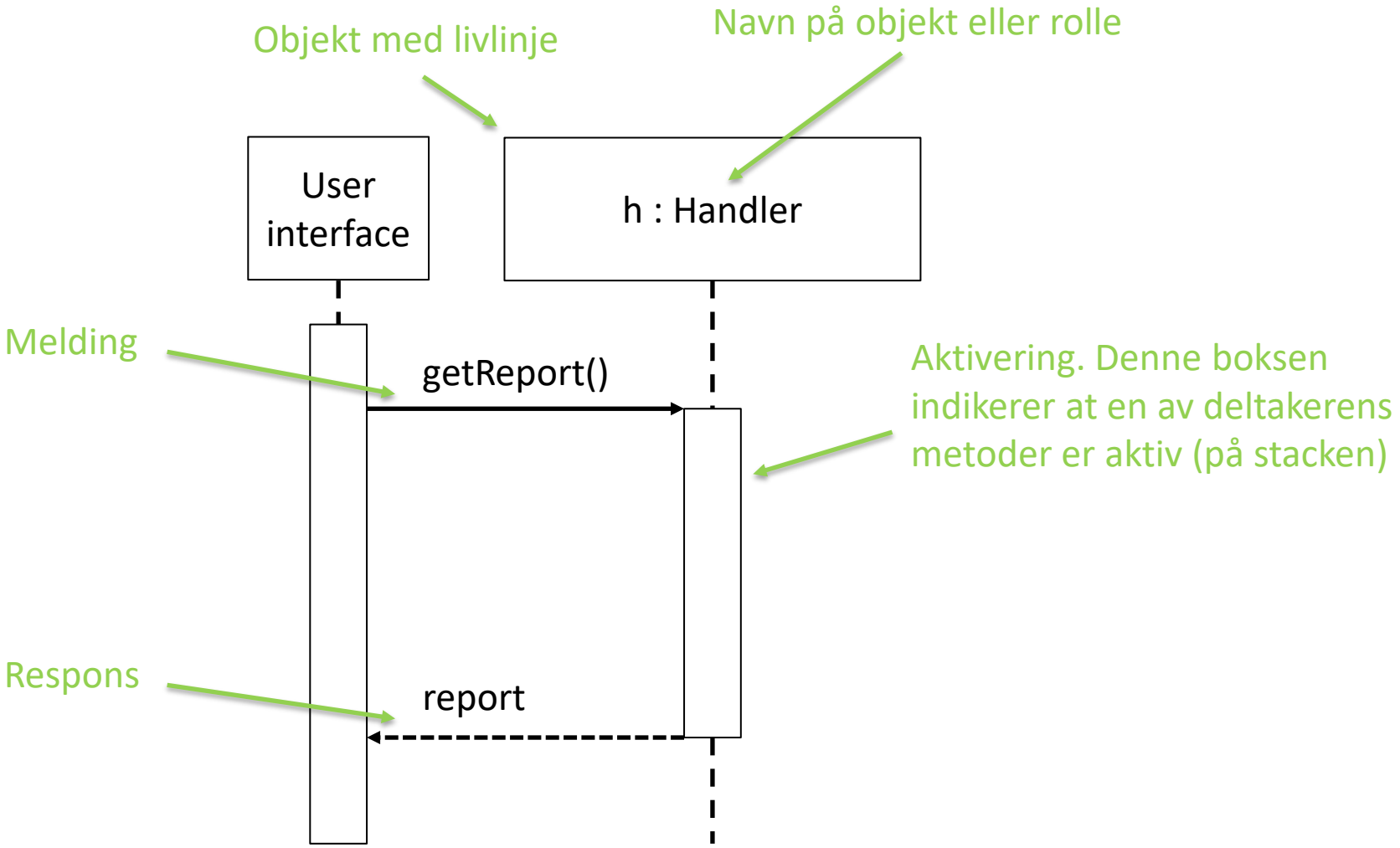
Sekvensdiagram

Sekvensdiagrammet viser:

- Hvordan en gruppe objekter **samarbeider** for å realisere en oppførsel
- Deltakerne i et **scenario** (sub-sett av et use case)
- Sekvensen og naturen i **meldingsutvekslingen** mellom deltakerne
- Deltakernes **livsløp**

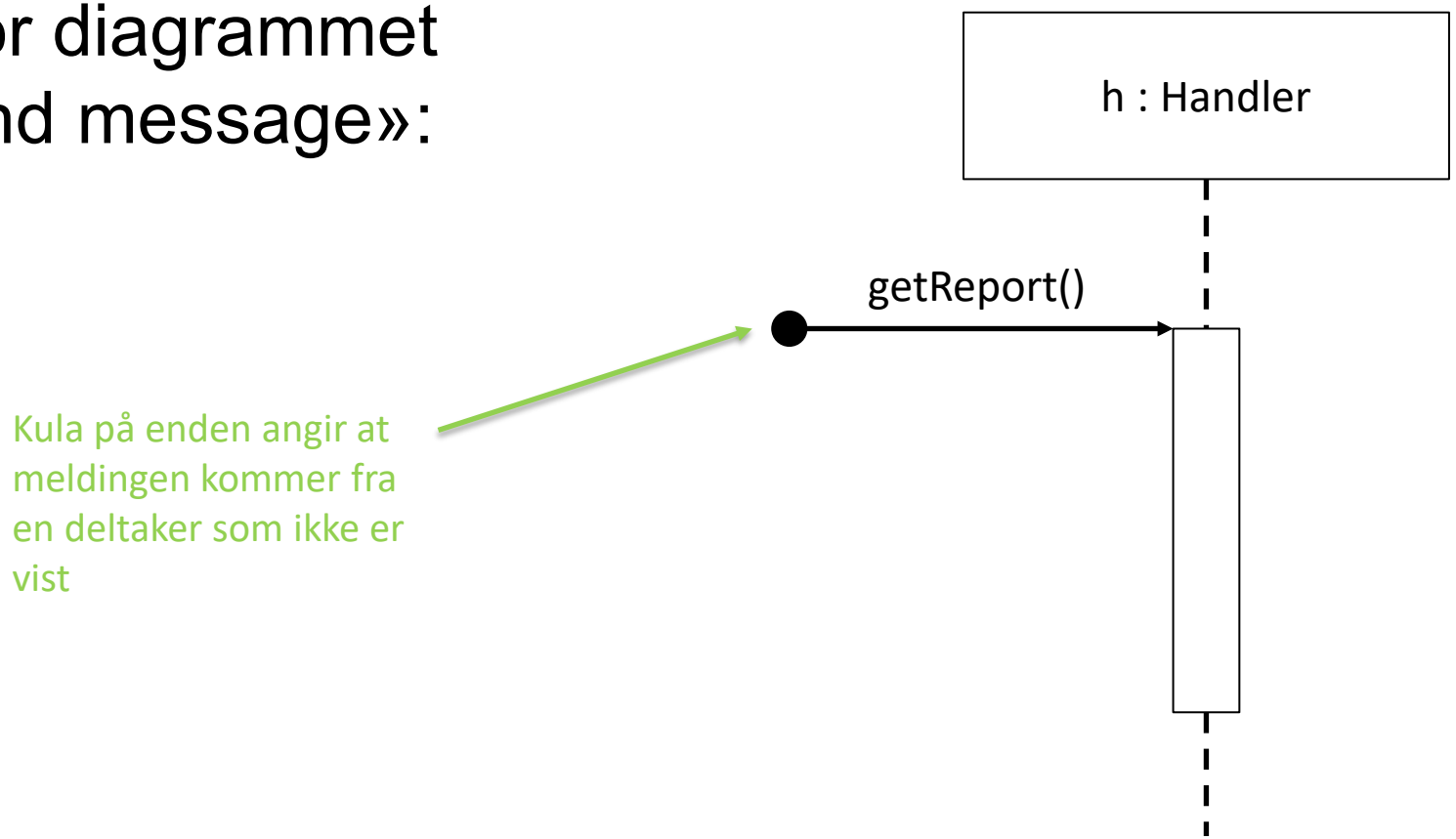


Notasjon


Objekt med livlinje
Navn på objekt eller rolle
User interface
h : Handler
Melding
getReport()
Aktivering. Denne boksen indikerer at en av deltakerens metoder er aktiv (på stacken)
Respons
report

Notasjon

Melding som kommer fra en deltaker utenfor diagrammet kalles en «found message»:



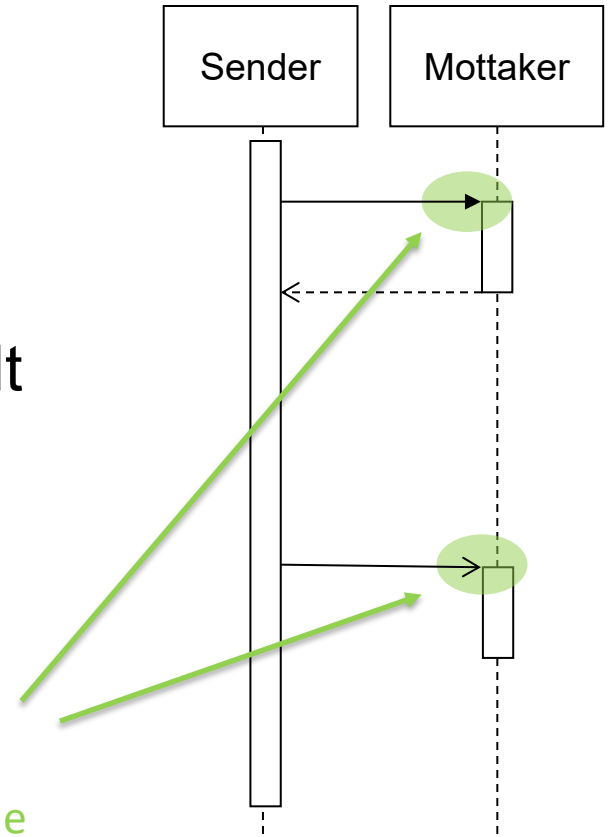
Synkrone vs. asynkrone meldinger

Synkron:

Senderen må vente til meldingen er ”utført”
(typisk: metodekall innenfor samme tråd)

Asynkron:

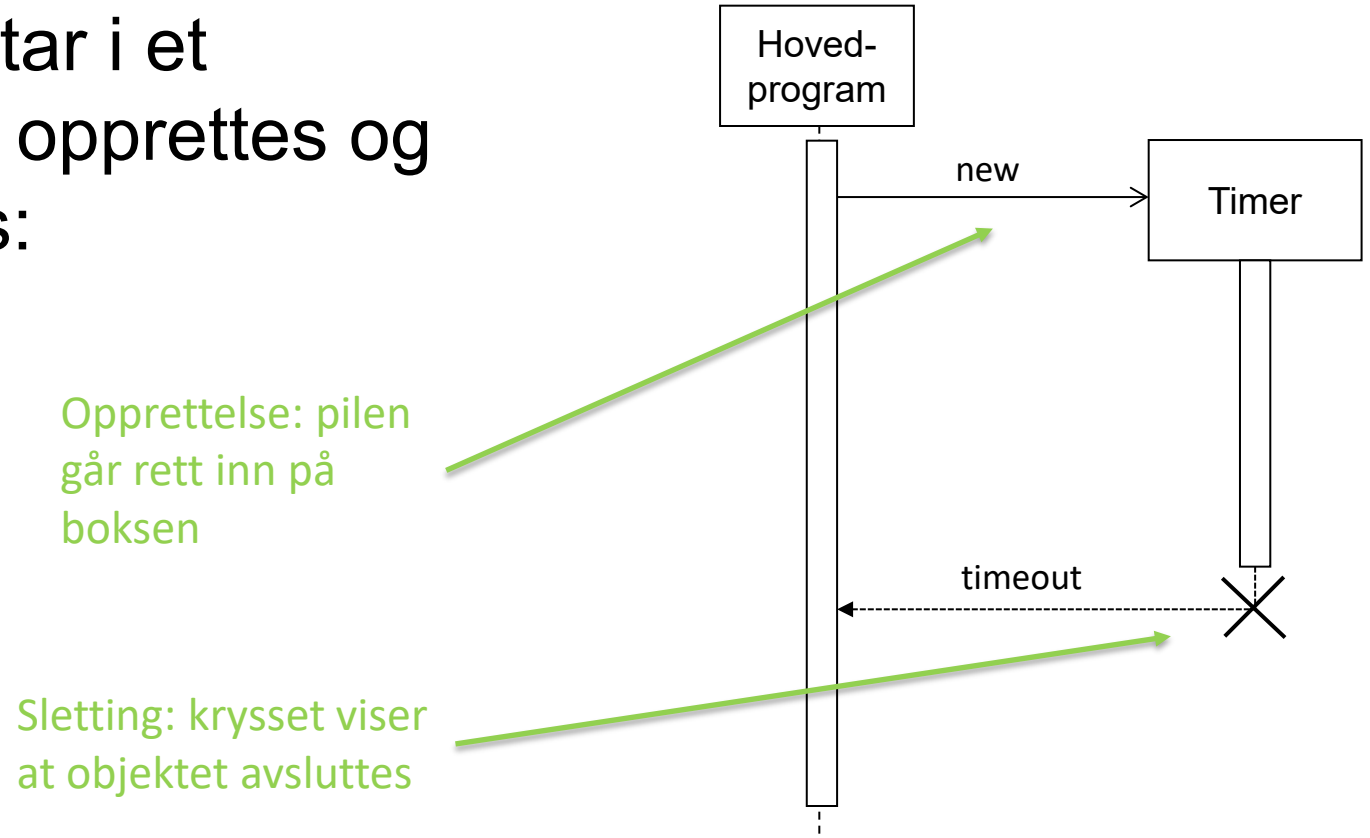
Senderen fortsetter så snart meldingen er sendt
(typisk: meldingskø)



Forskjellen indikeres med
forskjellige hoder på pilene

Opprette og slette deltakere

En kan indikere det dersom objekter som deltar i et sekvensdiagram opprettes og slettes underveis:

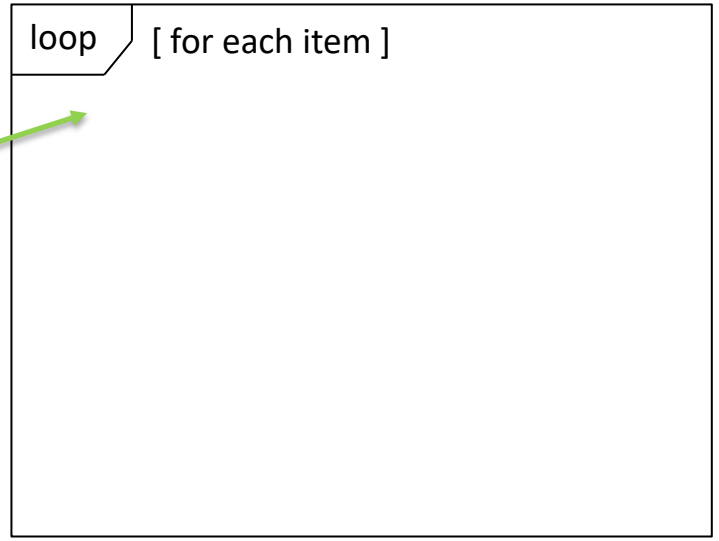


Kontrollflyt i sekvensdiagram

Interaction frames brukes til å indikere kontrollflyt, som:

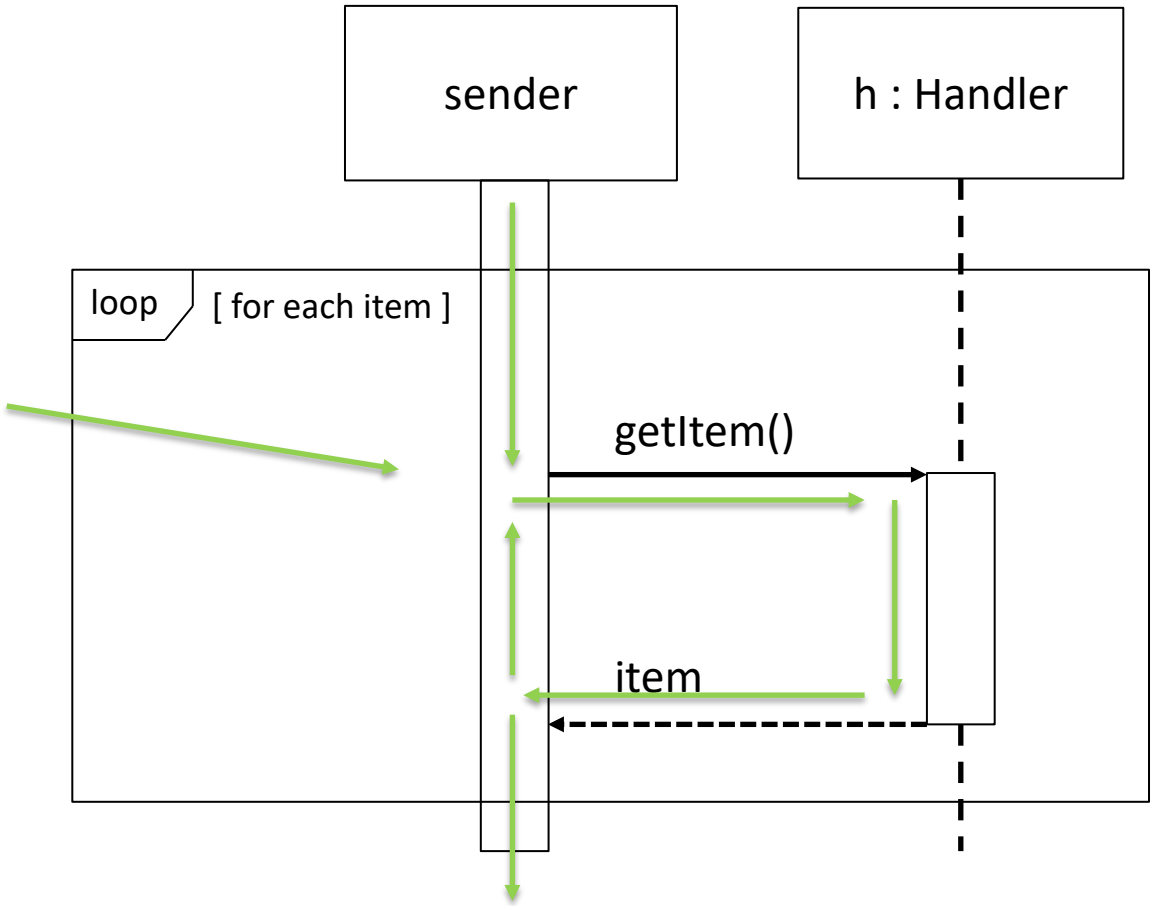
- Løkker (*for, while* ...)
- Forgreninger (*if, switch*, …)
- Parallelle sekvenser

Type struktur og kriterie
skrives oppe i venstre hjørne



Eksempel: løkke

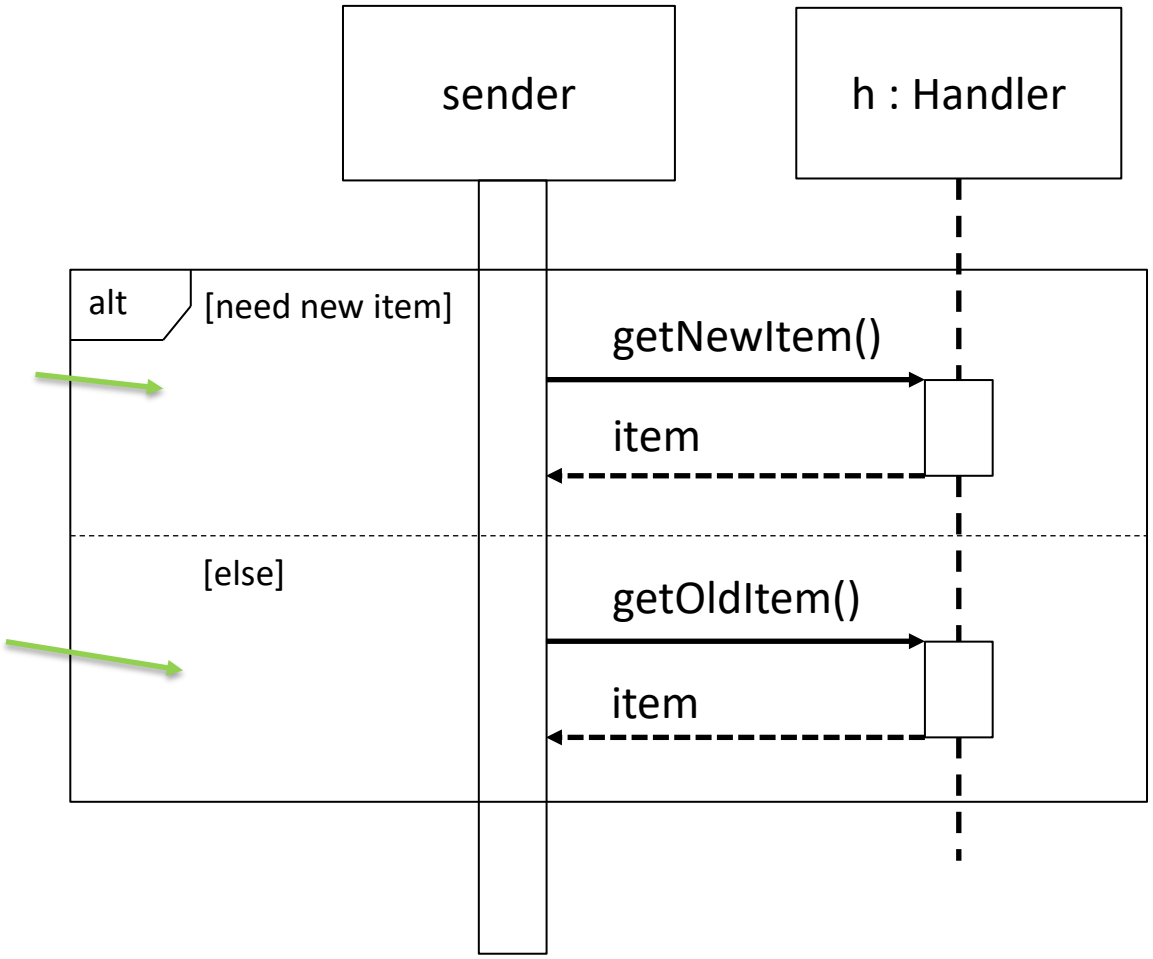
Sekvensen innenfor rammen gjentas som spesifisert



Eksempel: forgreining

Øverste del utføres hvis betingelsen er sann

…. ellers utføres nederste del



Operatorer for *interaction frames*

Noen av de vanligste operatorene som brukes:

Operator	Betydning
alt	Alternative sekvenser – bare den ene vil utføres avhengig av kriterium
opt	Utføres bare dersom kriteriet er sant
par	Parallelle tråder – segmentene utføres parallelt
loop	Løkke – utføres gjentatte ganger som spesifisert av kriteriet
region	Kritisk region som bare kan utføres av en tråd av gangen

Eksempel: Sekvensdiagram for suksess-scenariet under use case *Styr kran*

Styr kran

Precondition:

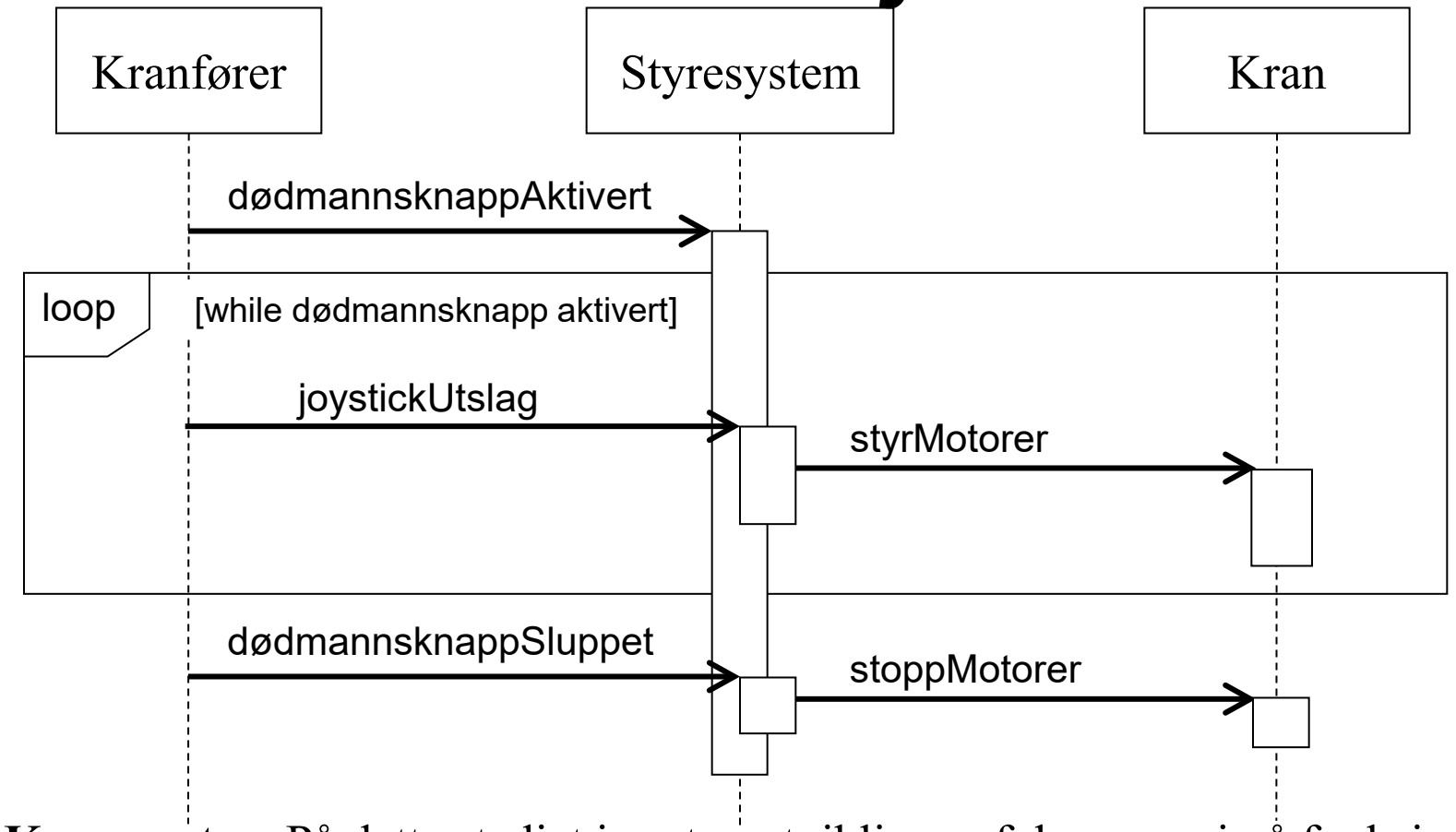
Kranfører i setet,
kranen tikoplet strøm
kranen i ro

Trigger: dødmannsknapp aktivert

Suksess-scenario:

1. Kranfører beveger joystick
2. Kranens motorer kjøres med hastighet gitt av joystick-utslag
3. Dødmannsknapp slippes
4. Kranen stopper.

Eksempel: Sekvensdiagram for suksess-scenariet under use case *Styr kran*



Kommentar: På dette stadiet i systemutviklingen fokuserer vi på funksjon, ikke implementasjon. Vi har derfor *ikke* modellert inn hastighetstilbakekoplingen fra Kran til Styresystem.

Hvordan (ikke) bruke sekvensdiagrammet

Illustrere scenarier

- Kartlegge hvilke metoder/funksjoner som trengs i utførelsen av scenariene
- **Ikke** komplette/komplekse algoritmer
- **Ikke** egnet for presis oppførselsemodellering
- **Ikke** egnet for å vise kompleks flytkontroll

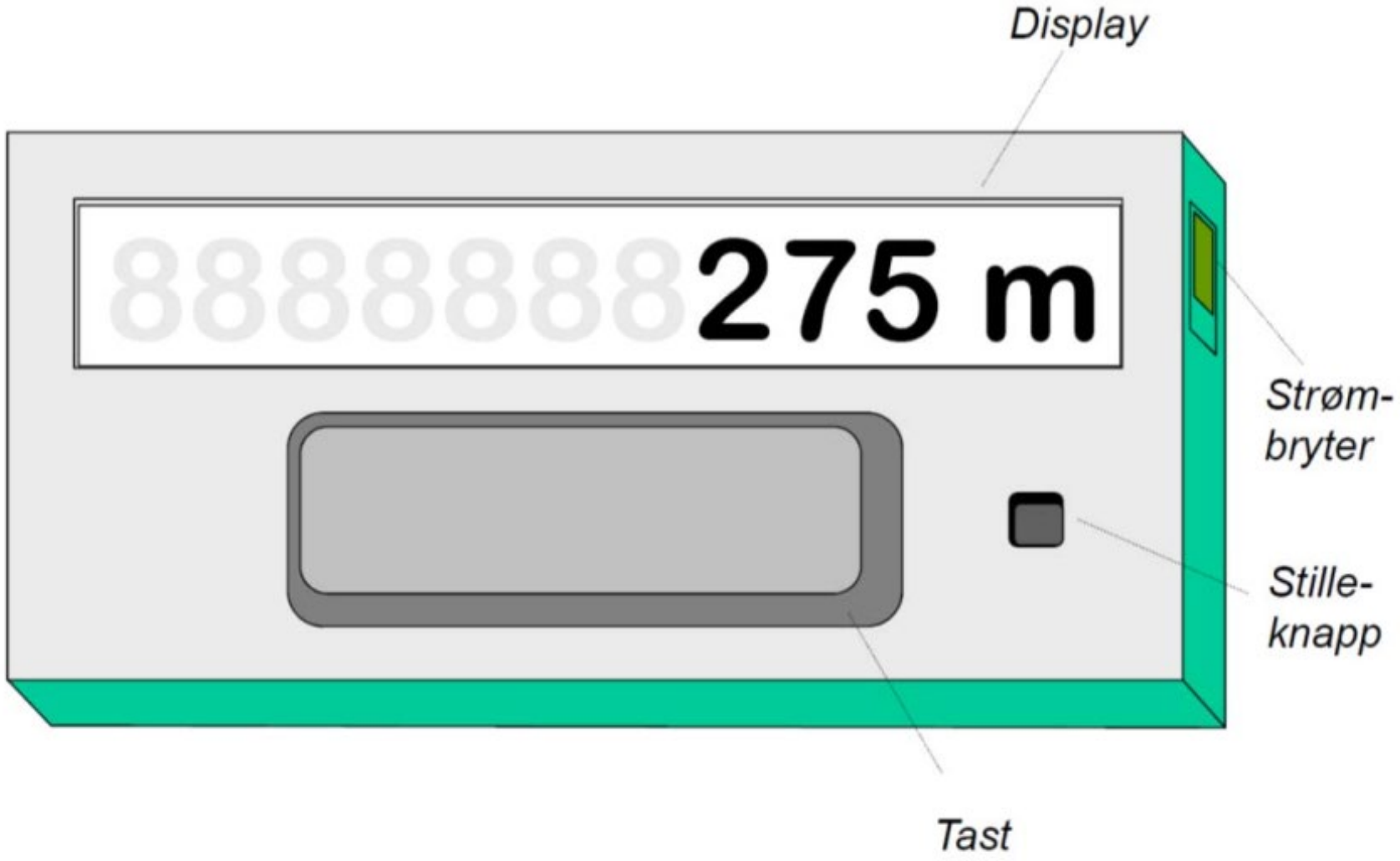
Rundetelleren

Eksempel på systemmodellering med UML:

- Use case-analyse
- Klassediagram
  (modulenes relasjoner og grensesnitt)

- Tilstandsmaskin (modulen(e)s oppførsel)
  - Kokebok for konstruksjon av tilstandsmaskin
- Supplerende diagrammer som sekvensdiagram

Rundetellereksempelet



Funksjonsspesifikasjon (prosaversjon)

Triptelleren slås på med strømbryteren og festes med sugekopper på bassengveggen der treningsøkten starter.

Telleverket nullstilles ved at den store knappen trykkes inn i 3 s.

Ved normal bruk vises svømt distanse kontinuerlig på displayet. Hver gang svømmeren vender ved den bassengenden der telleren er festet, trykkes det kort (< 3 s) på den store knappen. Når dette skjer økes svømt distanse med to ganger bassenglenden (dvs. lengden av én runde).

Dersom den lille knappen trykkes inn med en spiss gjenstand, kan bassenglenden stilles inn ved å trykke på den store knappen.

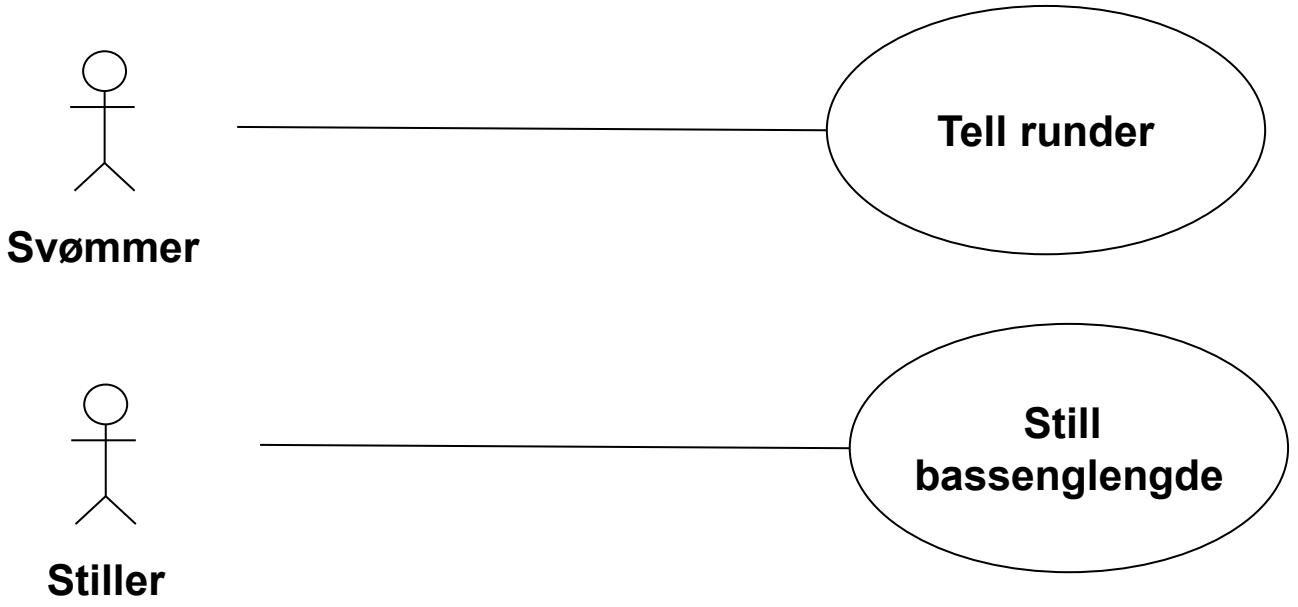
Displayet veksler da mellom tallene {12.5, 25, 50, 12.5, ...}, nytt tall for hvert trykk. Når det er gått 3 s siden siste tastetrykk, lagres det sist viste tallet som ny bassenglengde og systemet er klart til normal bruk igjen.

Når treningsøkten er over, slås telleren av med strømbryteren.

For enkelthets skyld antar vi at alle lagrede data (bassenglengde og svømt distanse) automatisk huskes selv om triptelleren slås av og på igjen.

Use case-diagram

Funksjonsspesifikasjonen antyder to use cases og to aktører:



Use case-beskrivelse:

Tell runder («normal bruk»)

Precondition: Riktig bassenglende er stilt inn,
systemet er slått på.

Trigger: Den store knappen trykkes inn

Hovedscenario:

1. Den store knappen slippes igjen før det har gått 3 s
2. (2 x bassenglengden) legges til svømt distanse
3. Svømt distanse vises på displayet

Utvidelse:

1a: Den store knappen holdes inne i 3 s

.1: Svømt distanse nullstilles

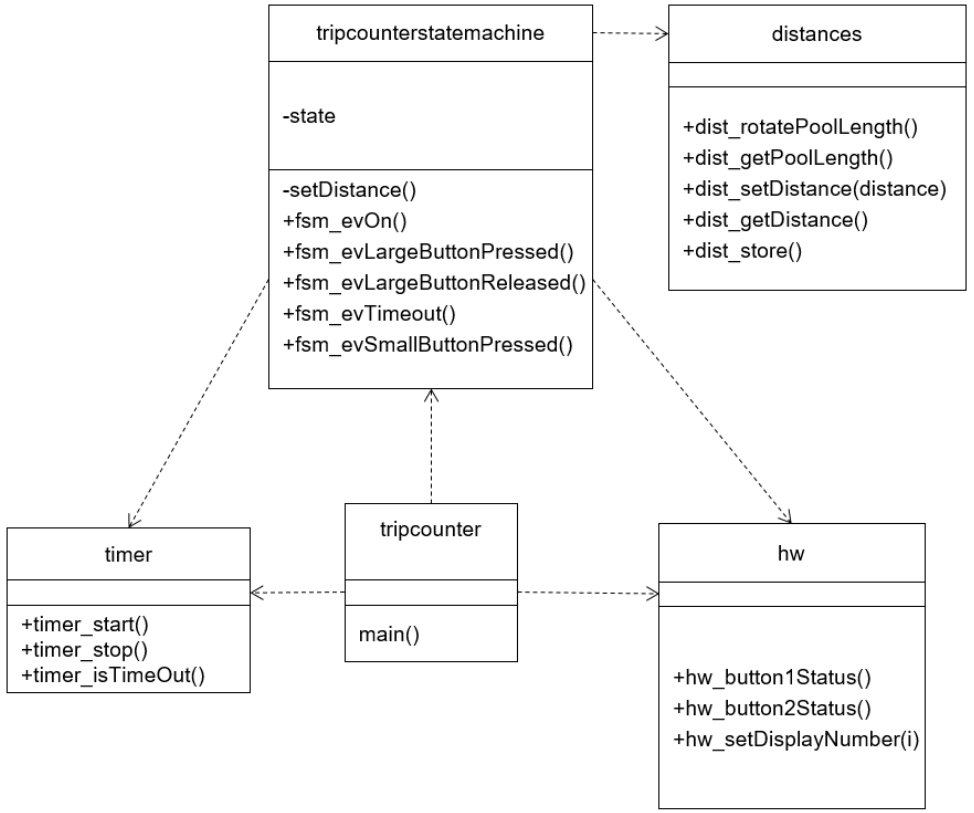
.2: Returnerer til hovedscenariets trinn 3.

Garanti:

Svømt distanse vises på displayet.
(kanskje overflødig)

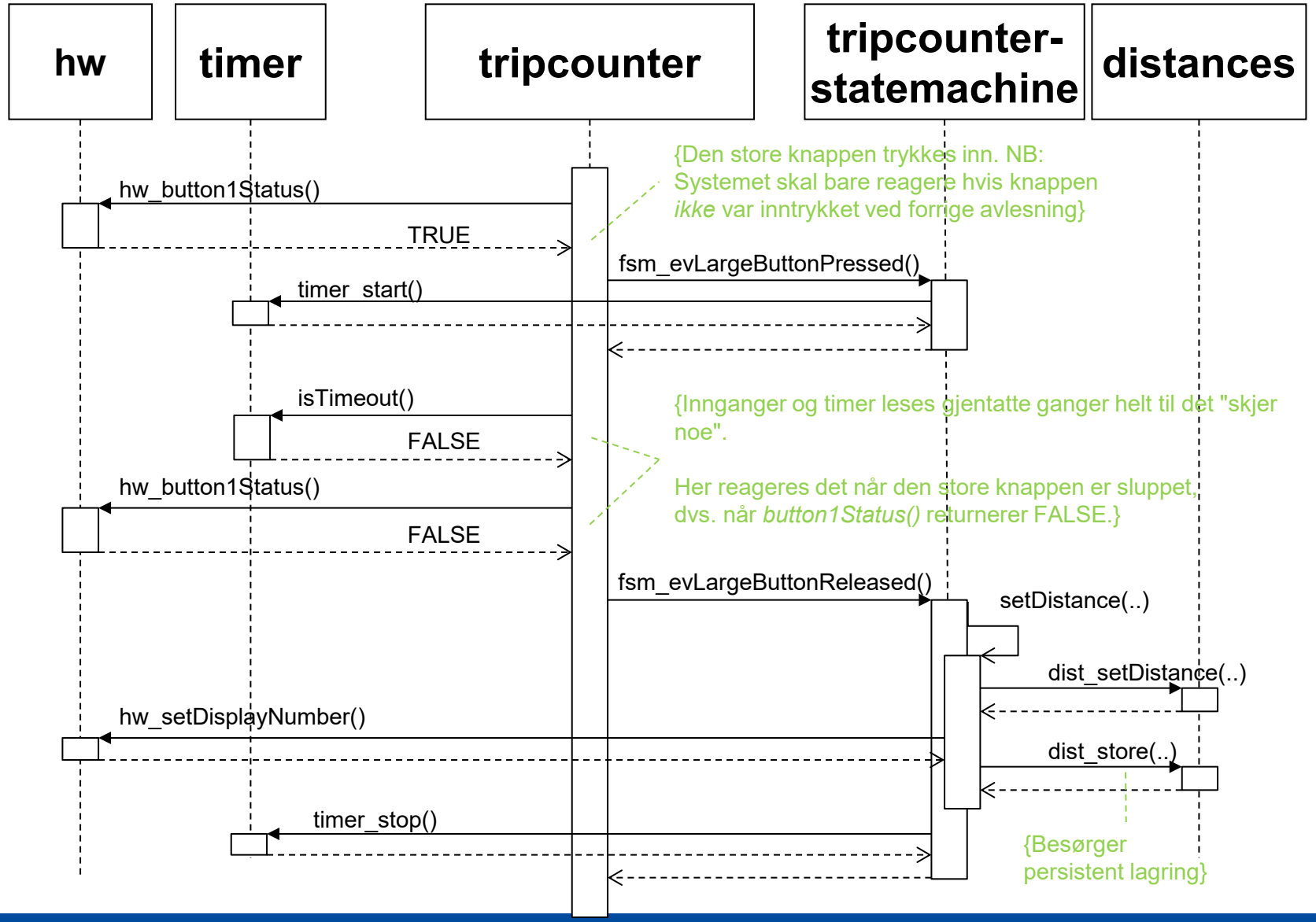
Sekvenser under normal bruk

- Aktørene i sekvensene er de samme som i klassediagrammet:
  - **hw**: hardware; rapporterer status på knapper og viser tallverdier
  - **timer**: en enhet som måler tid fra et starttidspunkt
  - **tripcounter**: hovedprogrammet
  - **tripcounter-statemachine**: tilstandsmaskin som definerer den logiske funksjonen til rundetelleren
  - **distances**: persistent lagring av data



Klassediagram for rundetelleren

Scenario: tell runder, hovedscenario



Konstruksjon av tilstandsmaskin

Med utgangspunkt i en beskrivelse (f.eks. en use case-beskrivelse eller et diagram som detaljerer ett av scenariene), kan en grei fremgangsmåte være som følger:

1. List opp hendelsene i hovedscenariet i naturlig rekkefølge
   - dette er triggerne
2. Plasser en tilstand mellom/før hver av dem
3. Bind tilstandene sammen med transisjoner i henhold til den beskrevne oppførselen
4. Suppler med aksjoner knyttet til transisjoner og/eller tilstander
5. Gjør dette for ett scenario om gangen inntil hele funksjonsspesifikasjonen er dekket

Tell runder, hovedscenario

1. List opp hendelsene (=triggerne) i scenariet

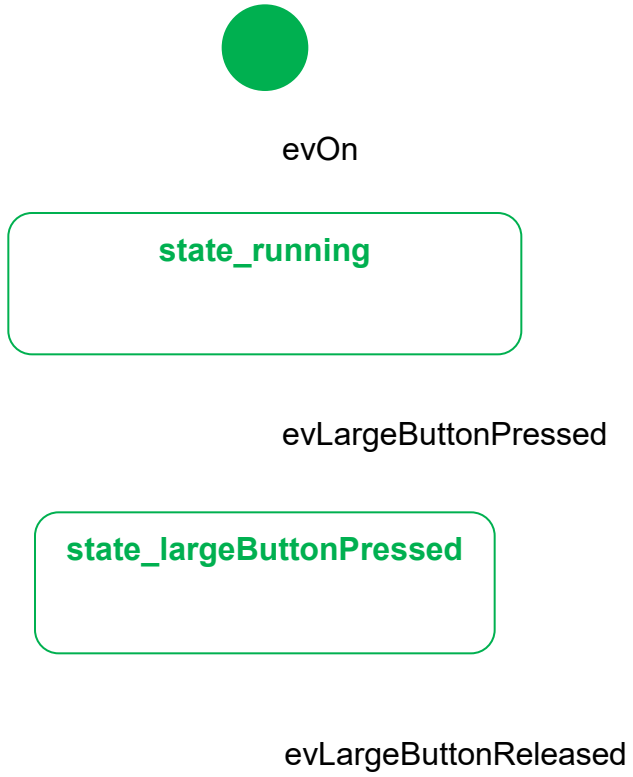
evOn

evLargeButtonPressed

evLargeButtonReleased

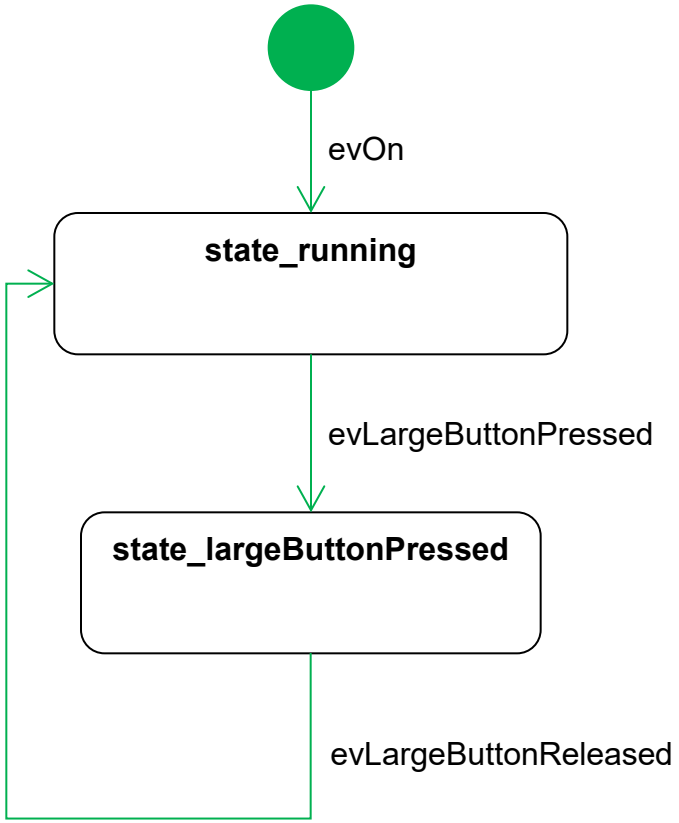
Tell runder, hovedscenario

2. Plasser en tilstand mellom/før hver trigger

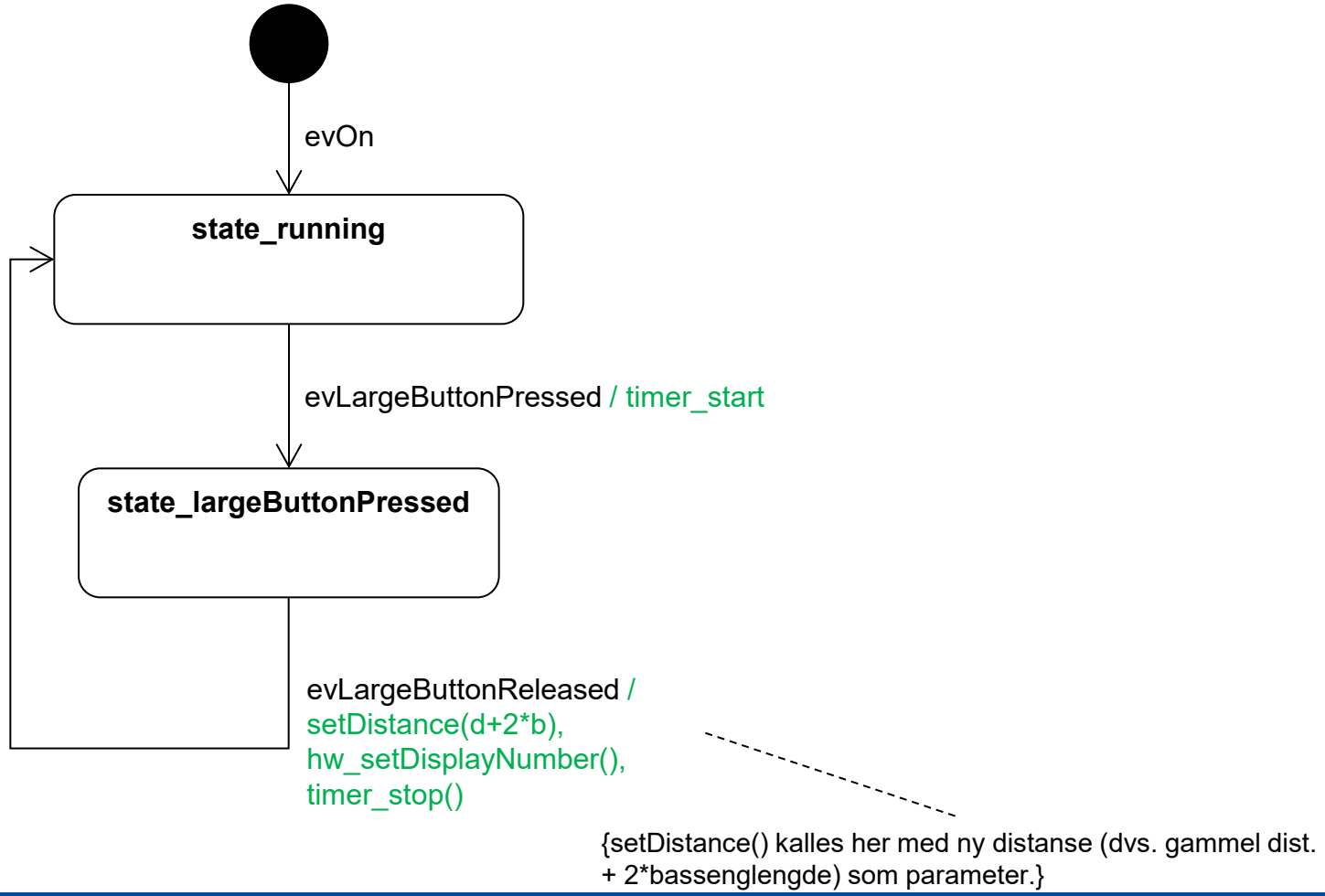


Tell runder, hovedscenario

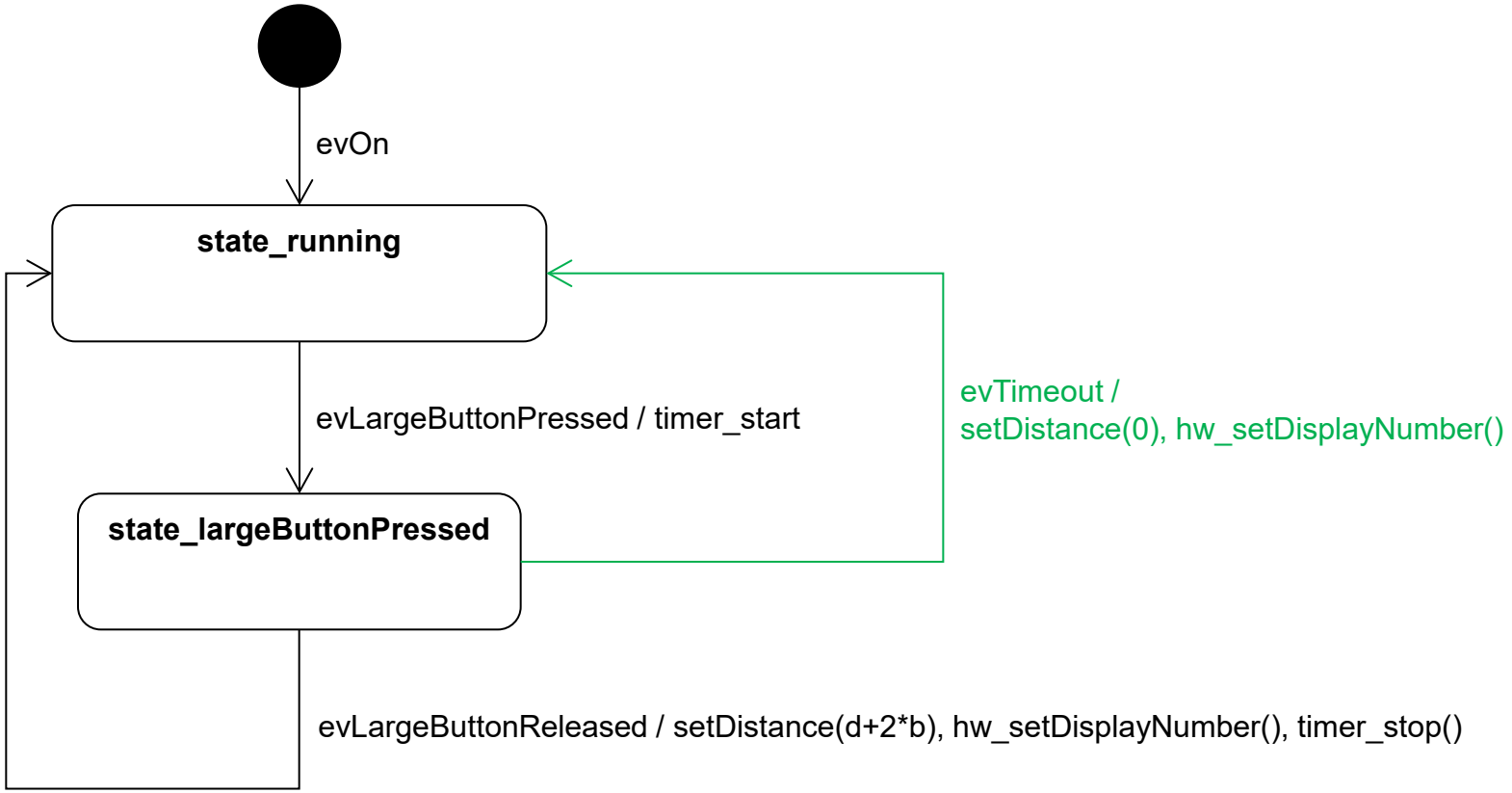
3. Bind tilstandene sammen med transisjoner



Tell runder, hovedscenario
4. Suppler med aksjoner



Tell runder, utvidelse 1a



Scenario:
Still bassenglengde

Precondition:

Systemet er slått på.

Trigger:

Den lille knappen trykkes inn.

Hovedscenario:

1. Displayet viser neste tall i rekka av tillatte bassenglengder.
2. Det går 3 s uten at den store knappen trykkes
3. Bassenglengden lagres

Utvidelse:

2a: Den store knappen trykkes innen 3 s.

.1: Gå til hovedscenariets trinn 1.

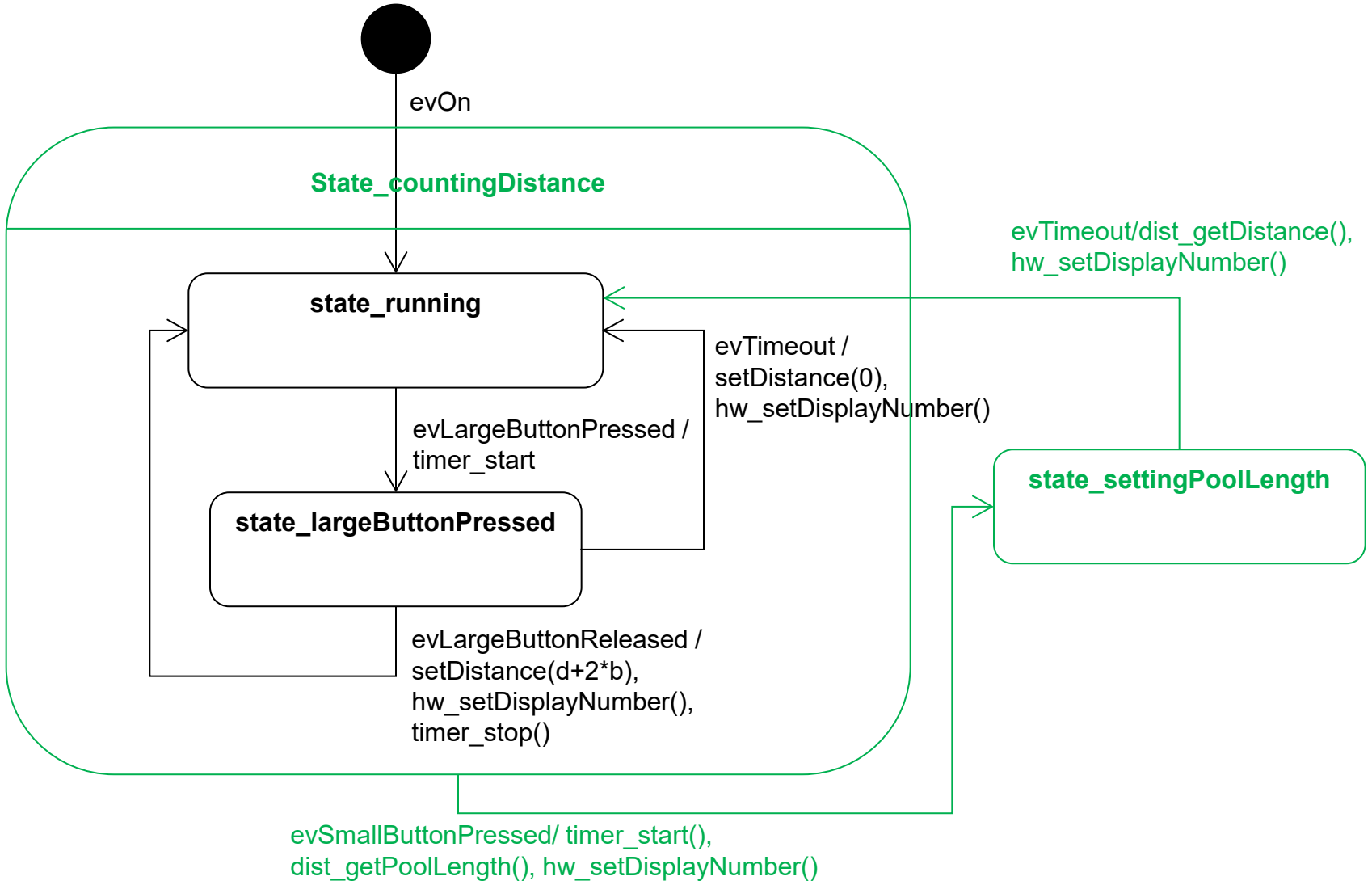
Garanti:

Preconditions for use case *Tell runder er* oppfylt.

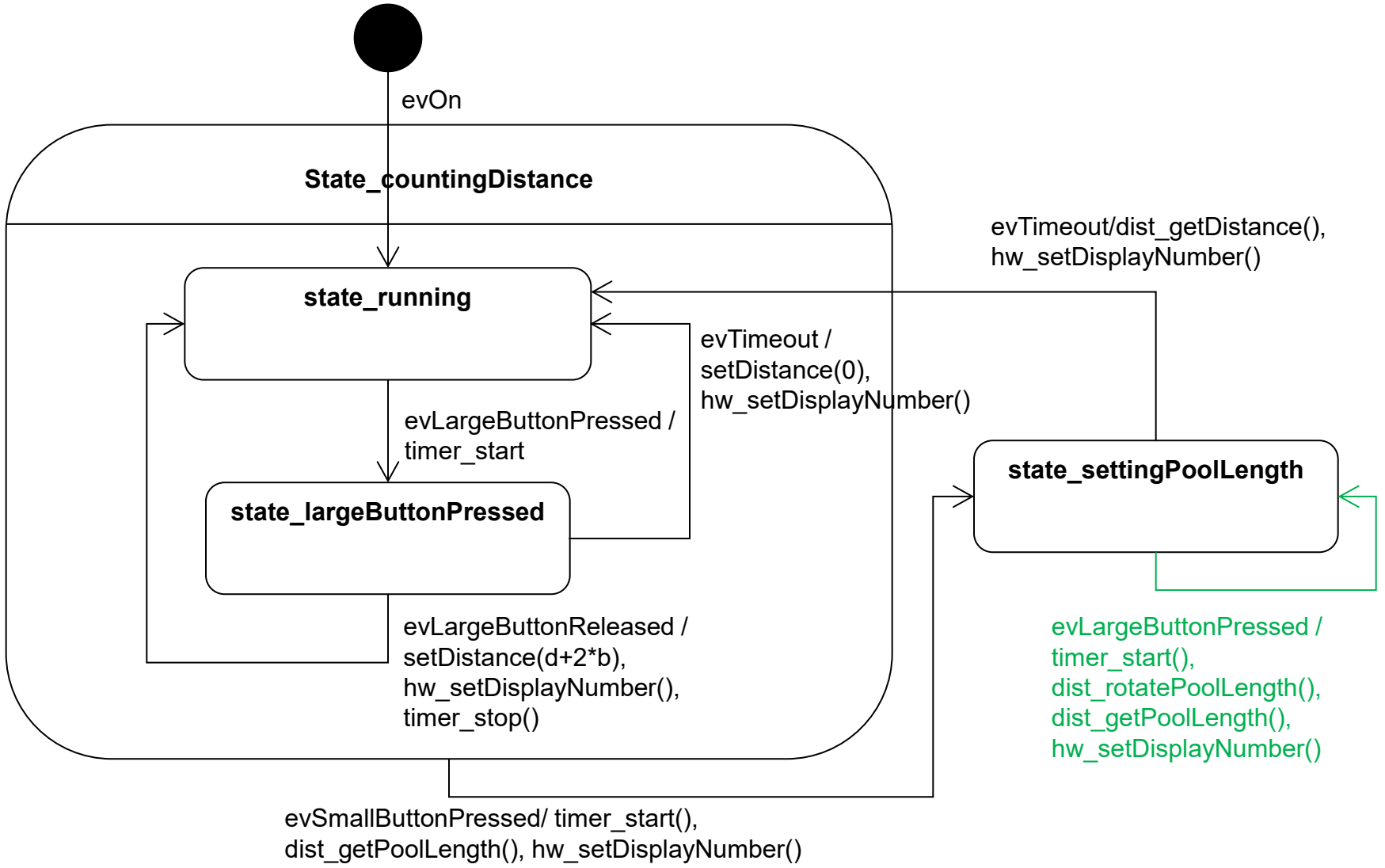
Her går vi fram på samme måte for å utvide tilstandsmaskina.

Use caset som dette scenariet tilhører, har en trigger ("*den lille knappen trykkes inn*") som i prinsippet kan inntreffe når som helst.
Vi velger derfor å lage en supertilstand, og knytte denne triggeren til en transisjon ut fra denne.

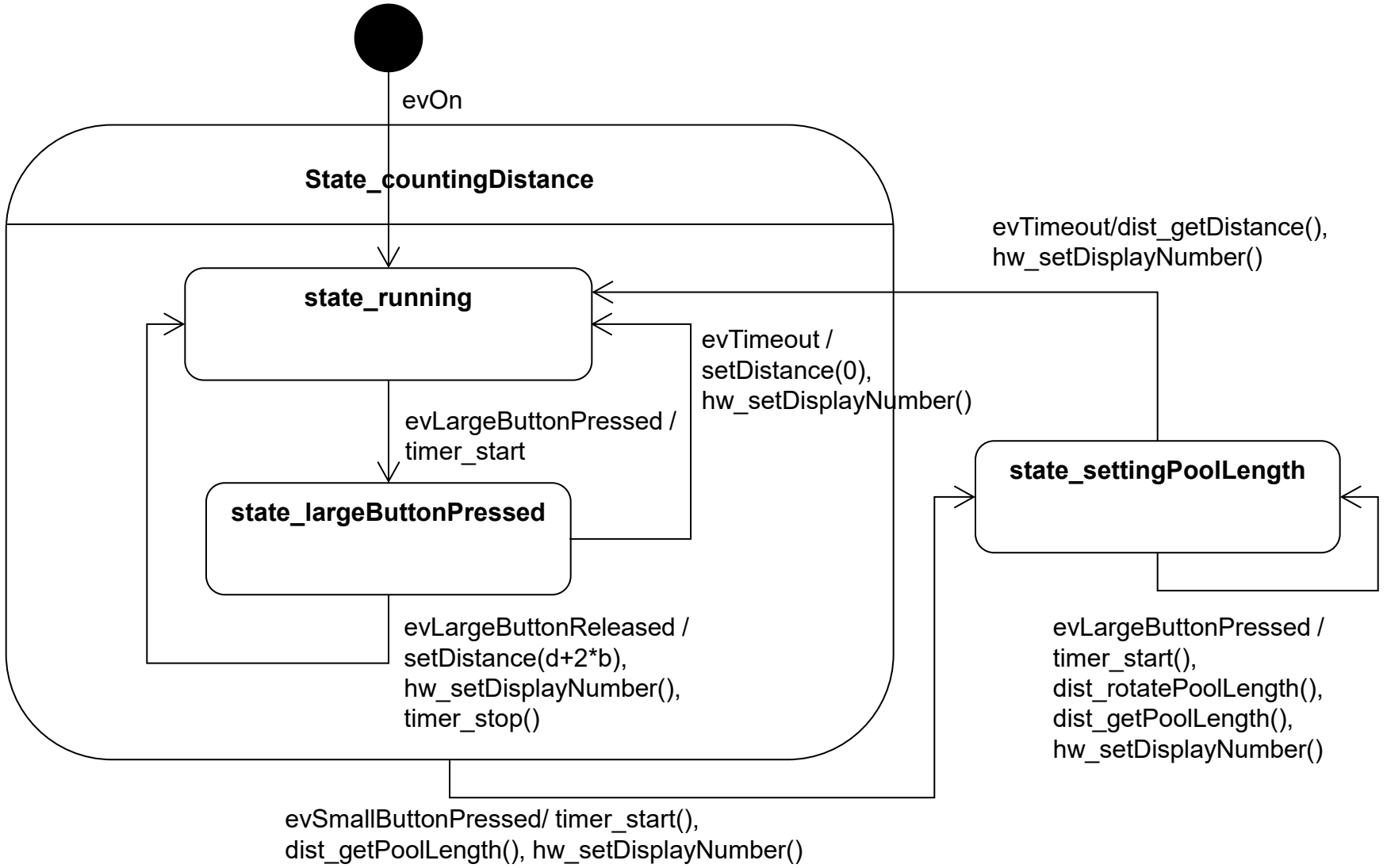
Still bassenglengde, hovedscenario



Still bassenglengde, utvidelse 2a



Still bassenglengde, utvidelse 2a



Oppsummering UML