Replikering

Olav Lysne

Hvorfor replikere I?

- Forbedret ytelse
  - Flere servere tilbyr samme tjeneste - parallellitet
  - Distribuerte kopier av data fører til mindre nettverksforsinkelse
  - Caching av data gir mindre nettverksbelastning

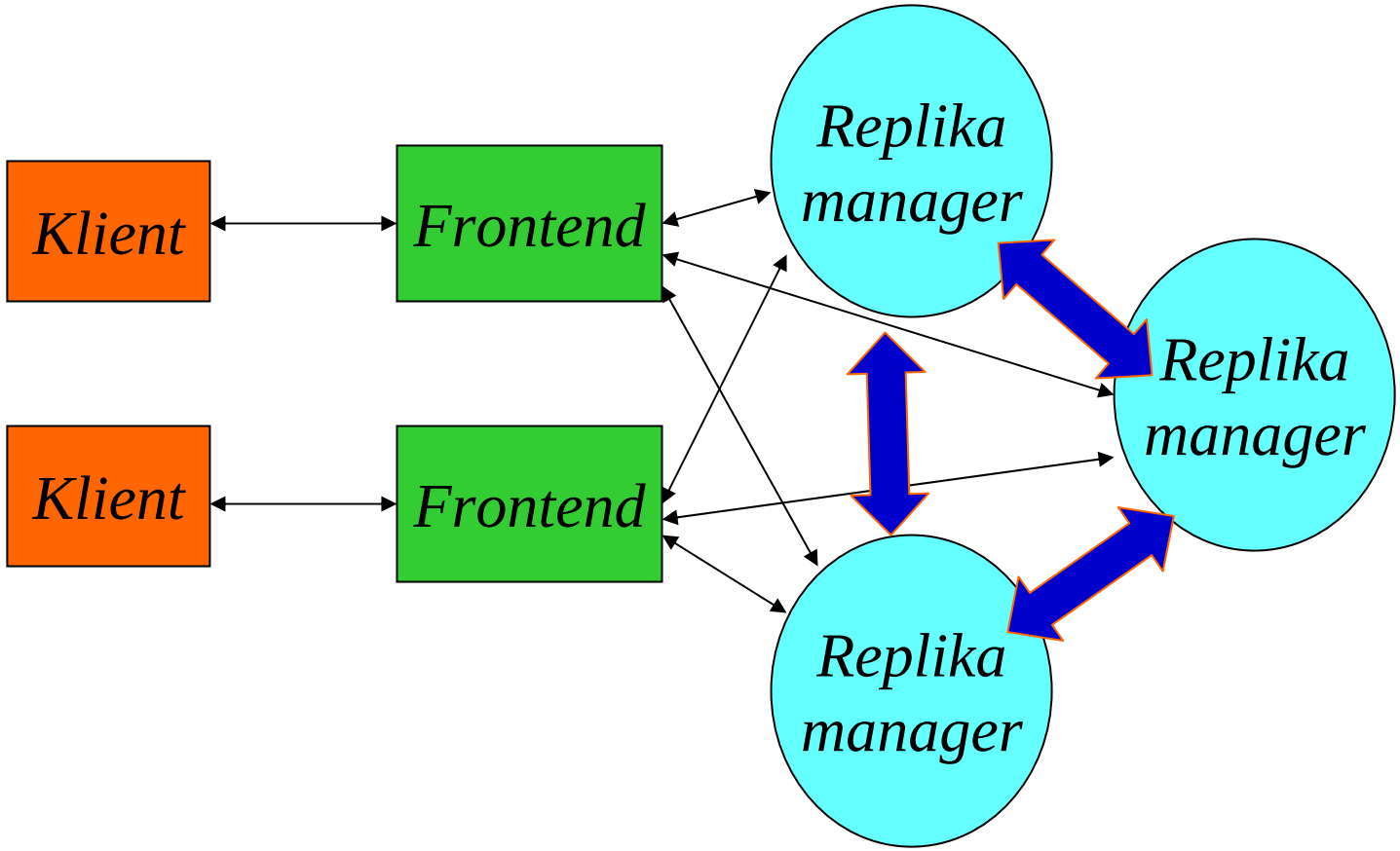
Hvorfor Replikere II?

- Bedret tilgjengelighet
  - For enkelte tjenester (slik som navnetjenester) er det viktig at tilgjengeligheten med akseptabel responstid er tilnærmet 100%, til tross for at ...
  - tjenere kan feile - gå ned
  - Deler av nettverket kan feile.
- Eks: 5% sjanse for serverfeil i en gitt periode - to uavhengige servere gir 99.75% tilgjengelighet.

Hvorfor replikere III?

- Feiltoleranse
  - Krav om korrekt operasjon ved feil.
  - Krever ofte tett samarbeid mellom replika for å vedlikeholde bildet av feiltransparens.
  - Dette samarbeidet kommer ofte i direkte konflikt med kravet om tilgjengelighet.
  - Ved bysantinske feil trengs 2n+1 replika for å kunne tolerere n feilende servere...i prinsippet
  - Ved crash-feil trengs kun n+1 replika for å kunne tåle n feil...i prinsippet

Replikeringsarkitektur



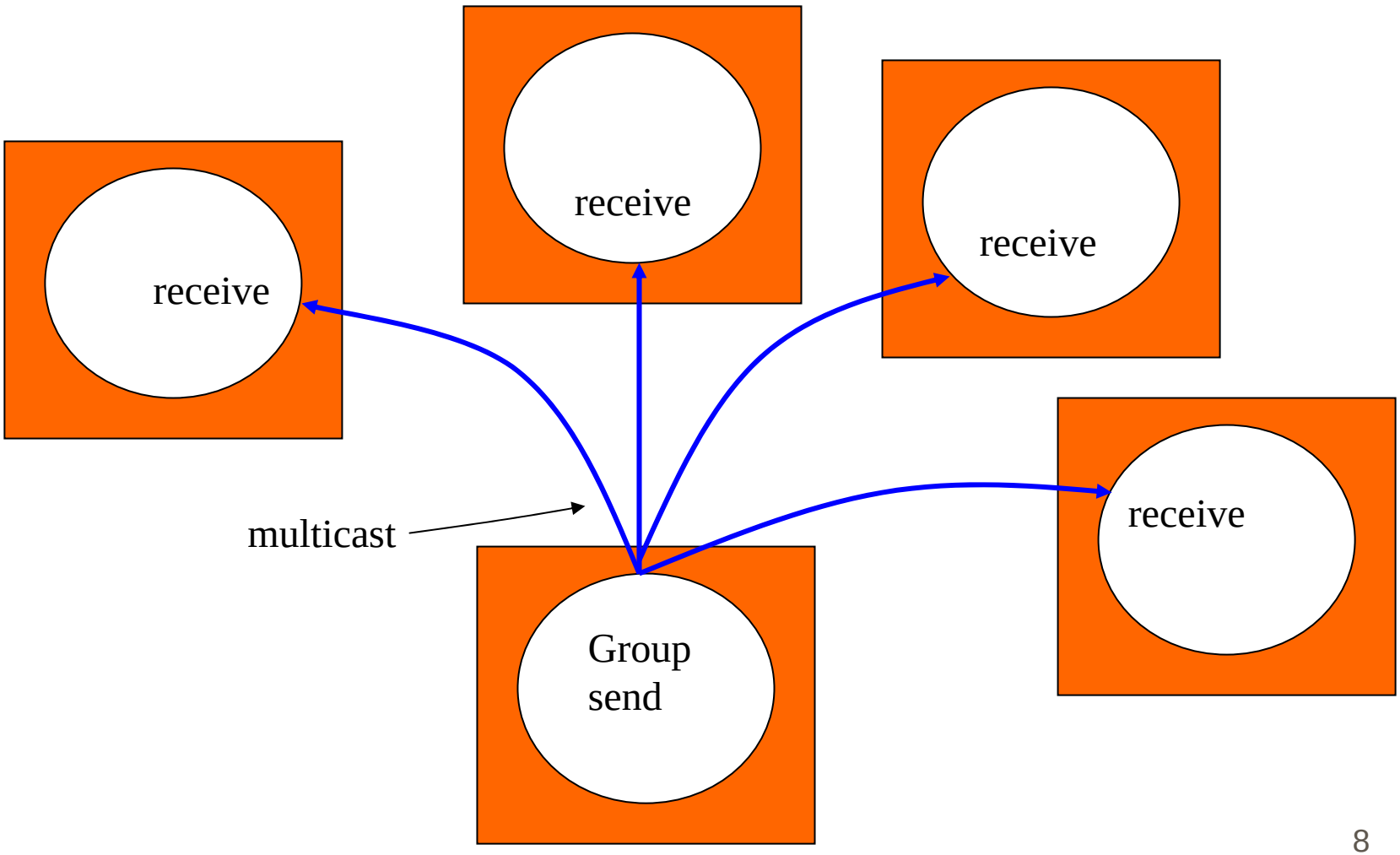
Replikeringsarkitektur II

- Denne arkitekturen tillater
  - At replikeringen er transparent
  - Dynamiske replika - nye replika kan komme, andre kan forsvinne uten at klienten behøver å bli informert.
  - Feiltransparens.
- Det meste av det som foreleses idag er knyttet til kommunikasjon mellom frontend og replikamanagere.

Forespørsel til replikert tjeneste - fem faser

- Forespørsel fra frontend til en eller flere replikamanagere
- Koordinering mellom replikamanagere
  - Fifo -ordning
  - kausal ordning (hendte før relasjonen)
  - Total ordning (alle replika er enige om rekkefølgen)
- Eksekvering - forespørselen utføres
- Enighet om resultatet mellom replikamanagere
- Respons til frontend

Gruppekommunikasjon

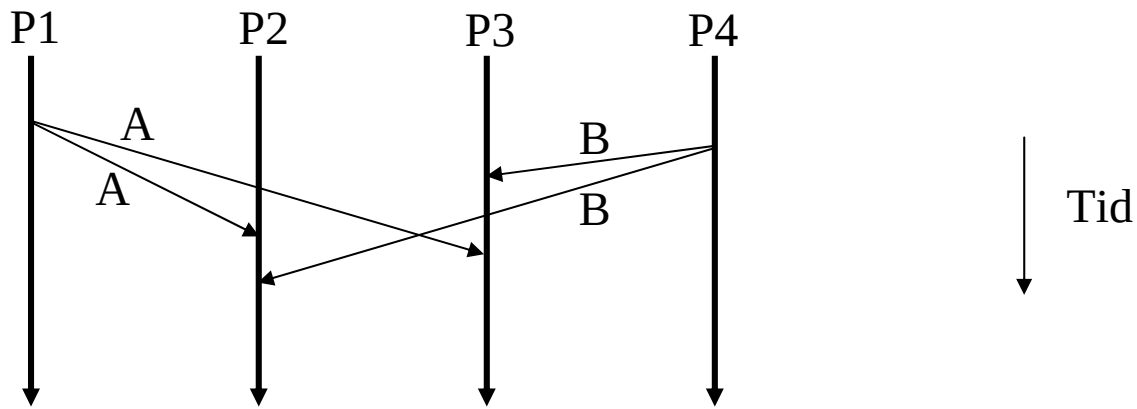


Gruppekommunikasjon II

- Atomisitet:
- ***Atomisk multicast***: En melding som overføres ved atomisk multicast mottas enten av alle prosesser som er medlem i den mottakende gruppe, eller meldingen blir ikke mottat av noen av dem.
- ***Pålitelig multicast***: En melding som overføres ved pålitelig multicast vil leveres til alle prosesser som er medlem i den mottakende gruppe, etter beste evne ("best effort"), men det gis ingen garantier.
- ***Upålitelig multicast***: En melding som overføres ved upålitelig multicast sendes kun en gang.

Gruppekommunikasjon III

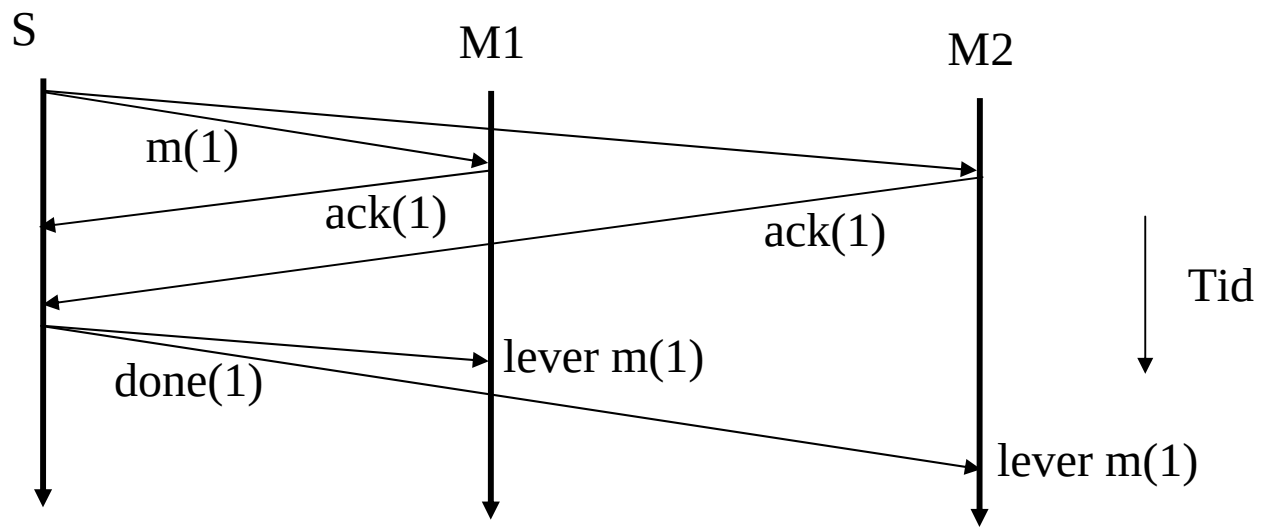
- Ordning:



- *Totalt ordnet multicast*: når flere meldinger overføres til en gruppe prosesser ved totalt ordnet multicast, vil alle meldingene mottas i samme rekkefølge av alle prosessene i gruppen.
  - Feiltoleranse basert på replikerte tjenester, kan realiseres ved totalt ordnet multicast

Implementasjon Gruppekommunikasjon

- Utnytte broadcast/multicast muligheter på nettlaget
- Pålitelighet: krever monitorering (av den som venter på svar)
- Pålitelig meldingsutveksling:



- Problem: Dårlig ytelse (mange meldinger)
- Effektiviseringer: Negative kvitteringer + meldingshistorier

Implementasjon Gruppekommunikasjon II

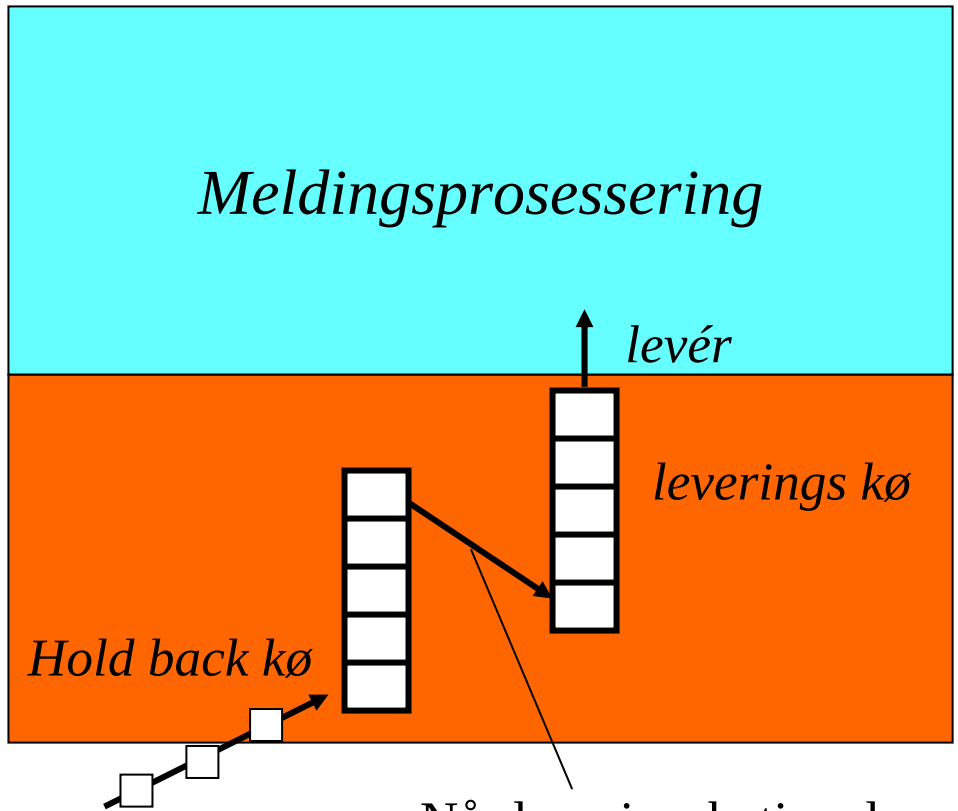
- Totalt ordnet atomisk multicast
  - Kan implementeres under antagelsen:
    Alle meldinger kan tilordnes en identifikator med en global felles ordning
- Mulige tilnærminger:
  - timestamps fra logiske eller fysiske klokker
  - sequencer prosess som alle meldinger sendes via
  - egen protokoll mellom medlemmene i en gruppe for å generere identifikatorer

Gruppemedlemskaps-tjeneste

- Grensesnitt for oppretting av gruppe, avmelding og påmelding
- Feildeteksjon
- Distribuere informasjon om endringer i gruppesammensetningen
- Adresseekspansjon - en adresse for multicast til hele gruppen.

Inne i replikeringsmanageren

Replikamanageren leverer ikke meldinger til prosessering før den vet at evt. konsistensbetingelser er oppfylt.
F. eks ordnet multicast, atomisk multicast...



View - bilde av gruppens tilstand

- For enkelte anvendelser er det nødvendig at hvert eneste replika har oversikt over hvilke replika som var medlem av gruppen idet en forespørsel kom inn.
- For å oppnå dette benyttes et begrep som kalles view-synkronitet.

View

- Et “view” er et bilde av hvilke prosesser som er med i en gruppe på et git tidspunkt.
- Når en prosess mottar besjed om at gruppen har endret seg, kan den selv styre når den vil levere (dvs. Prosessere) beskjeden.
- Dette skal imidlertid skje i henhold til visse konsistenskrav.

Konsistenskrav

- Global orden
  - alle prosesser er enige om sekvensen av views
- Integritet
  - en prosess leverer bare views den selv er med i
- Ikke-trivialitet
  - Dersom to prosesser har meldt seg på og forblir i samme gruppe, skal de etterhvert være med i hverandres view av den gruppen.

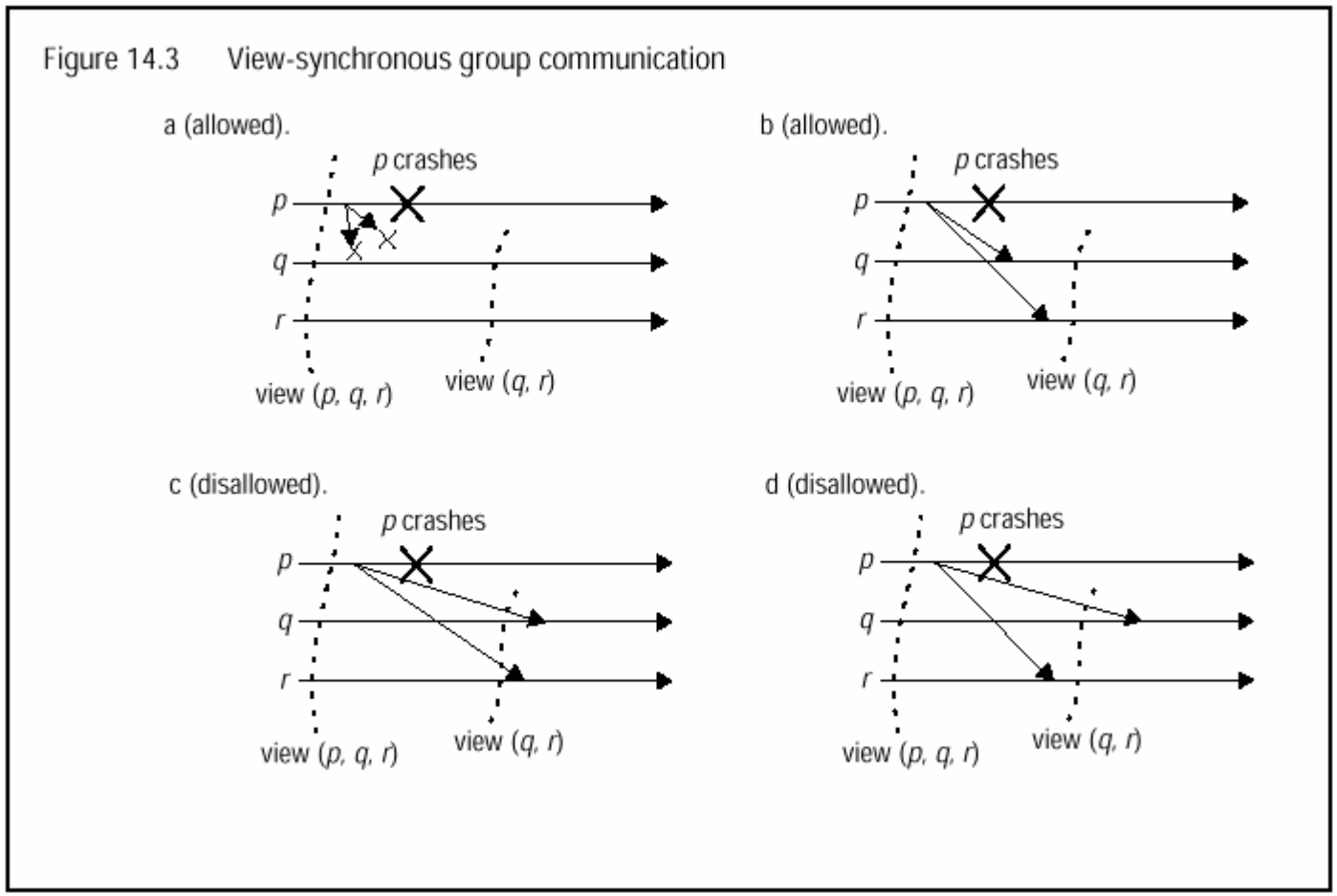
View-synkron kommunikasjon

- Intuisjon: kommunikasjonen i en gruppe er view synkron dersom alle medlemmene i gruppen leverer de samme meldingene i samme view.
- Kompliserende faktorer:
  - det behøver ikke være noen prosesser som er med i gruppen hele tiden.
  - Prosesser kan feile etter at man har blitt enige om å levere en melding, men før den har rukket å gjøre det

View-synkronitet II

- Dersom prosessen *p* leverer *m* i *v(g)* og deretter leverer *v(g'),* vil alle prosesser *q* som er med i både *v(g)* og *v(g')* levere *m* i *v(g).*
- Dersom *p* leverer *m*
  - blir *m* levert bare én gang,
  - er *p* med i multicastgruppen til *m*,
  - og ble *m* ble sendt til denne gruppen.
- Dersom *p* leverer *m* i *v(g)*, og en prosess *q* ikke leverer *m* i *v(g)*, vil det neste viewet *p* leverer ikke inneholde *q*.

Illustrasjon (fra boken)



Feiltoleranse

- En tjeneste er feiltolerant dersom den framstår som feilfri selv om enkelte komponenter feiler.
- Eksempel:
  - en bank har to replika av kontodatabasen.
  - Alle klienter kan gjøre oppdateringer på hvilket replika de vil - replikaene oppdaterer hverandre.
  - Klient 1 setter først inn 100 kroner på konto **x** vha replika A, og deretter 100 kroner på konto **y** vha replika B.
  - Ved korrekt oppførsel skal ingen noensinne observere at det har kommet inn 100 kroner på **y** og ikke på **x**

Men...

Klient 1

settinn$_B$(x, 100)

B feiler

settinn$_A$(y, 100)

Klient 2

saldo$_A$(y) → 100

saldo$_A$(x) → 0

Hvordan oppnå feiltoleranse i dette tilfellet?

Sekvensialiserbarhet

- Intuisjon: For enhver eksekvering av systemet skal det eksistere en sekvensiell eksekvering som:
  - gjør det samme som den opprinnelige eksekveringen
  - oppfyller spesifikasjonen av én enkelt kopi av objektene
  - sekvensen av operasjonene er i “hendte før” orden.

Eksempel I

Klient 1	***Klient 2***
$settinn_B(x, 100)$	
	$saldo_A(y) \rightarrow 0$
	$saldo_A(x) \rightarrow 0$
$settinn_A(y, 100)$	
	$saldo_A(y) \rightarrow 0$
	$saldo_A(x) \rightarrow 0$
$settinn_B(x, 100)$	
$settinn_A(y, 100)$	

Eksempel II

Klient 1

settinn$_B$(x, 100)

settinn$_A$(y, 100)

Klient 2

saldo$_A$(y) ⟶ 100

saldo$_A$(x) ⟶ 0

Dette er IKKE serialiserbart, fordi det ikke finnes en sekvens som tilfredsstiller bildet av et enkelt, ikke-replikert system

Passiv replikering I

- Passiv replikering
  - En (eller noen få) er *primær* replika manager, og alle frontender kommuniserer dit
  - Den primære manageren sender oppdateringer til de sekundære tjenere.
  - Når den primære manageren dør, overtar en av de sekundære som primærmnager.

Passiv replikering - forespørselfaser

- Forespørsel
  - Frontend sender en forespørsel med unik ID
- Koordinering
  - Primær manager håndterer forespørslene i den rekefølge han får dem.
- Eksekvering
  - Primær m. behandler forespørselen, og lagrer resultatet
- Enighet
  - Primær m. sender resultatet av oppdateringen til de andre replika
- Respons
  - resultatet sendes tilbake til frontend.

Passiv replikering - egenskaper

- Sekvensialiserbarhet er opplagt tilfredsstilt
- Feiltoleranse
  - At en sekundær replika m. feiler er uproblematisk
  - Dersom den primære feiler -
    - En unik sekundær må ta over
    - Transparens oppnås ved at den registrerer seg hos en navnetjener
    - Den må/bør ta over i samme tilstand som den gamle primære var i idet den døde.
    - Alle sekundære må bli enige om hvilke operasjoner som hadde blitt utført idet den døde (krever view-synkronitet)

Aktiv replikering for feiltoleranse

- Alle replika har samme rolle.
- Frontender multicaster forespørsler til alle replika
- Alle replika svarer, og derfor er det i prinsippet tilstrekkelig med et replika som ikke feiler.

Aktiv replikering - forespørselfase

- Forespørsel
  - Frontend *multicaster* en forespørsel med unik ID
- Koordinering
  - Multicastsystemet gir totalt ordnet multicast
- Eksekvering
  - Alle replika eksekverer forespørselen - dette skjer automatisk i samme rekkefølge hos alle replika
- Enighet
  - Unødig - pga. multicast semantikken.
- Respons
  - Alle replika sender resultatet tilbake til frontend - som da kan velge å svare når første svar kommer, eller f.eks når det har kommet tilstrekkelig med svar til å håndtere evt. Byzantinske feil

Aktiv replikering - Egenskaper

- Sekvensialiserbarhet er oppfylt ved totalt ordnet multicast.
- Feiltoleranse for forskjellige feilmodeller kan ordnes av frontend.
- Antar totalt ordnet og pålitelig multicast
  - Dette er ingen banal antagelse, da denne multicastfunksjonaliteten er svært problematisk. Les seksjonene 11.4 og 11.5 i Coulouris.

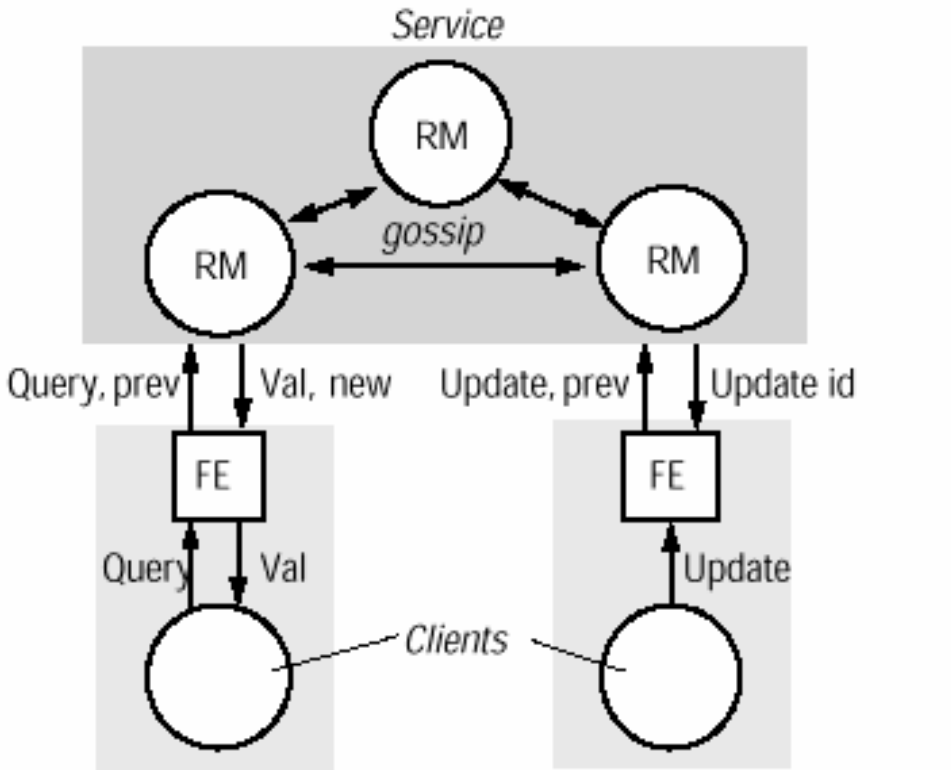
Tilgjengelighet

- Det vil vanligvis være en tradeoff mellom sekvensialiserbarhet og tilgjengelighet
- For enkelte applikasjoner er man villig til å gi avkall på feiltoleranse (sekvensialiserbarhet) for å bedre tilgjengeligheten
  - Distribuerte agendaer, distribuerte oppslagstavler...
- Dette begrepet er spesielt sentralt i mobile omgivelser, hvor brukeren ofte vil veksle mellom å være tilkoblet og frakoblet.

Gossip architecture

- En arkitektur for å implementere tjenester med høy tilgjengelighet, ved å replikere data i nærheten av der brukerene er.
- Garantier:
  - Hver klient observerer en konsistent tjeneste over tid. Selv om han veksler mellom flere tjenere vil han aldri få svar som indikerer at “tiden går bakover”.
  - Fleksibel konsistenskontroll
    - causal
    - forced (total og causal)
    - immediate (konsistent i forhold til *alle* oppdateringer).

Gossip arkitekturen II



Fault tolerant CORBA

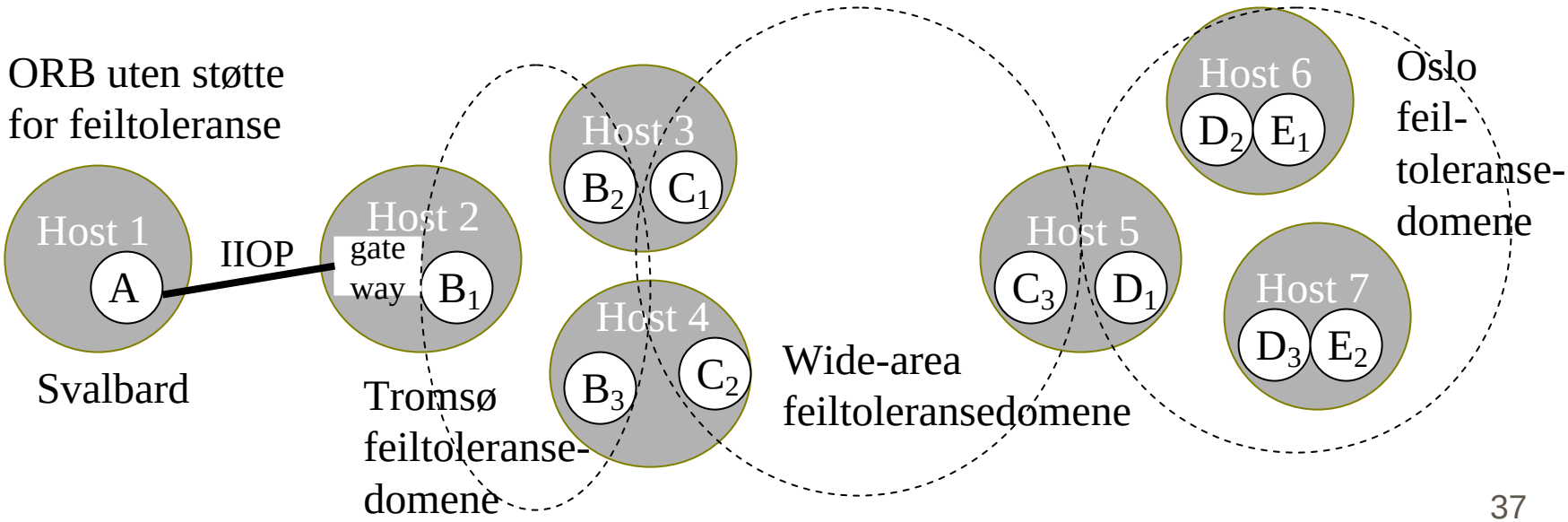
- Spesifikasjon: OMG TC Document orbos/99-12-19
- Basert på:
  - entitetsredundans (replikering (flere kopier) av objekter)
  - feildeteksjon (mulig å oppdage at server feiler)
  - recovery (f.eks. logg-basert)
- Understøtter ulike strategier for feiltoleranse
  - request retry (gjentar anrop ved uteblivelse av svar)
  - bruk av alternativ server (ved uteblivelse av svar)
  - passiv replikering (løst synkronsiert gruppe)
  - aktiv replikering (tett synkronisert gruppe)

Replikering og objektgrupper

- replika av et objekt opprettes og forvaltes som en objektgruppe
- hvert individuelle objekt i en gruppe har sin egen objektreferanse
- gruppen som helhet refereres ved en *interoperable object group reference* (IOGR)
- klienter benytter IOGR ved metodeanrop =>
  - replikeringstransparens (de individuelle objekt i gruppen skjules)
  - feiltransparens (feil i replika og recovery etter feil skjules)

Feiltoleranse-domener

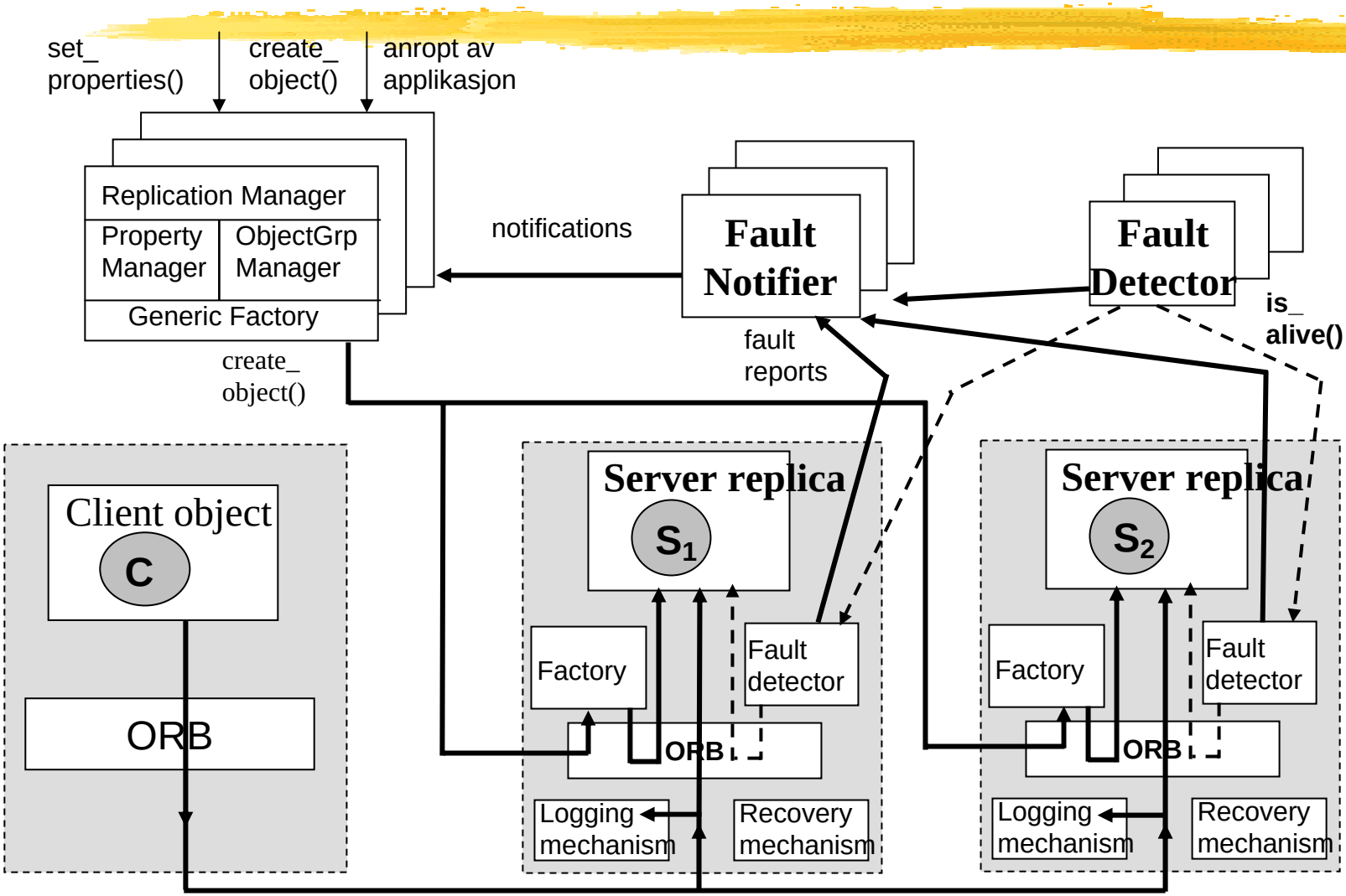
- Introduseres for å handtere skalerbarhet og kompleksitet
- Alle objektgrupper innen et feiltoleranse-domene
  - opprettes og forvaltes av en eneste *ReplicationManager*
  - kan anrope objekter i andre feiltoleranse-domener
  - kan anropes av objekter i andre feiltoleranse-domener



Feiltoleranse-egenskaper

- Hver objektgruppe har en tilhørende mengde feiltoleranse-egenskaper
  - *ReplicationStyle*
    - COLD_PASSIVE: løst synkronisert *uten* checkpointing
    - WARM_PASSIVE: løst synkronsiert *med* checkpointing
    - ACTIVE: tett synkronsiert gruppe
  - *IntialNumberOfReplicas*
  - *MinimumNumberOfReplicas*
  - *m.m.*
- Alternativt kan et feiltoleranse-domene tilordnes en mengde feiltoleranse-egenskaper
  - => alle objekt-grupper som opprettes i domenet får de samme feiltoleranse-egenskaper

Arkitektur FT CORBA



Oppsummering

- Grunner til å replikere er
  - Bedret ytelse
  - Tilgjengelighet
  - feiltoleranse
- Basale begreper i oppbygging av en replikert tjeneste er
  - Ordnet multicast
  - View-synkronitet
  - sekvensialiserbarhet
- Fault tolerant CORBA
  - tilbyr både aktiv og passiv replikering, m.m.