UNIVERSITETET
I OSLO

Systemfeil og logging

Basert på lysark laget av Hector Garcia-Molina

Oversatt og bearbeidet av Ragnar Normann

Integritet eller korrekthet av data

- Vi ønsker at data alltid skal være riktige og nøyaktige

Ansatt

Navn	Alder
Hansen	56
Lie	2489
Olsen	1

Integritets- eller konsistensregler

- Predikater som data må tilfredsstille
- Eksempler
  - X er nøkkel i relasjonen R
  - X → Y holder i R
  - Domain(farge) = {R(ød),G(rønn),B(lå)}
  - Ingen ansatt får tjene mer enn det dobbelte av gjennomsnittslønnen

Definisjoner:

- Konsistent tilstand:
  Tilfredsstiller alle integritetsregler
- Konsistent database:
  Databasen er i en konsistent tilstand

Integritetsreglene (som vi virkelig håndhever) trenger ikke å uttrykke "den fulle sannhet"

Eksempel 1 Transaksjonsregler

- Hver gang lønnen oppdateres, gjelder at

  ny lønn > gammel lønn

- Hver gang en konto-forekomst fjernes, gjelder at

  kontoens saldo = 0

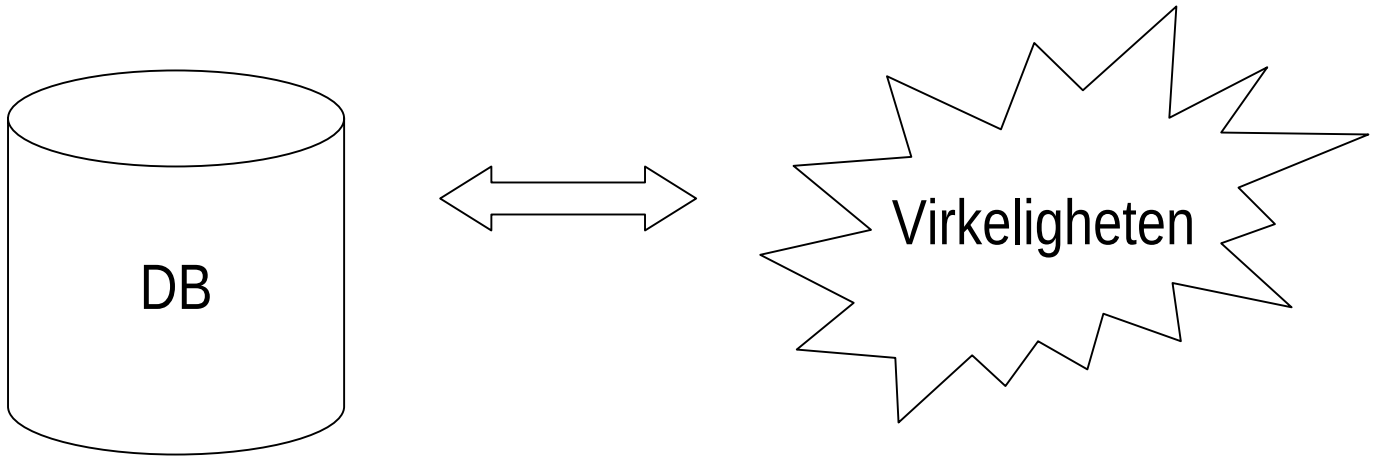
Merk:

Det siste eksemplet kan vi “fikse” med enkle midler som å legge til et boolsk attributt:

Konto	Kto#	…	saldo	fjernet?

Integritetsreglene (som vi virkelig håndhever) trenger ikke å uttrykke “den fulle sannhet”

Eksempel 2 — Databasen bør gjenspeile den virkelige verden



uansett, fortsett å bruke integritetsregler …

Observasjon:
Databaser kan ikke alltid være konsistente

Eksempel: $a_1 + a_2 + \ldots + a_n = \text{total}$ (*integritetsregel*)

Sett inn 1000 kroner på konto a_2:

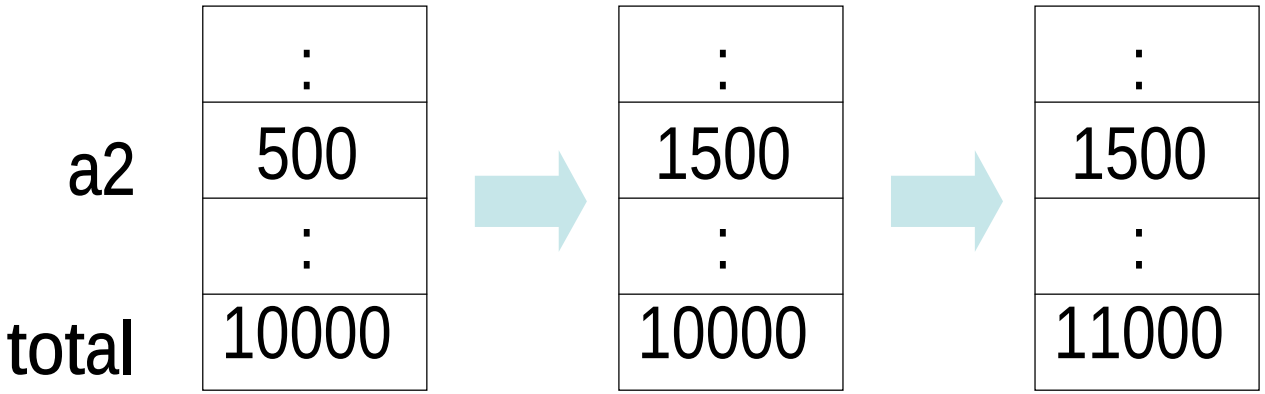
$a_2 \leftarrow a_2 + 1000$
$\text{total} \leftarrow \text{total} + 1000$

Integritetsregelen er midlertidig brutt

Eksempel: $a_1 + a_2 + \ldots + a_n = \text{total}$ (*integritetsregel*)

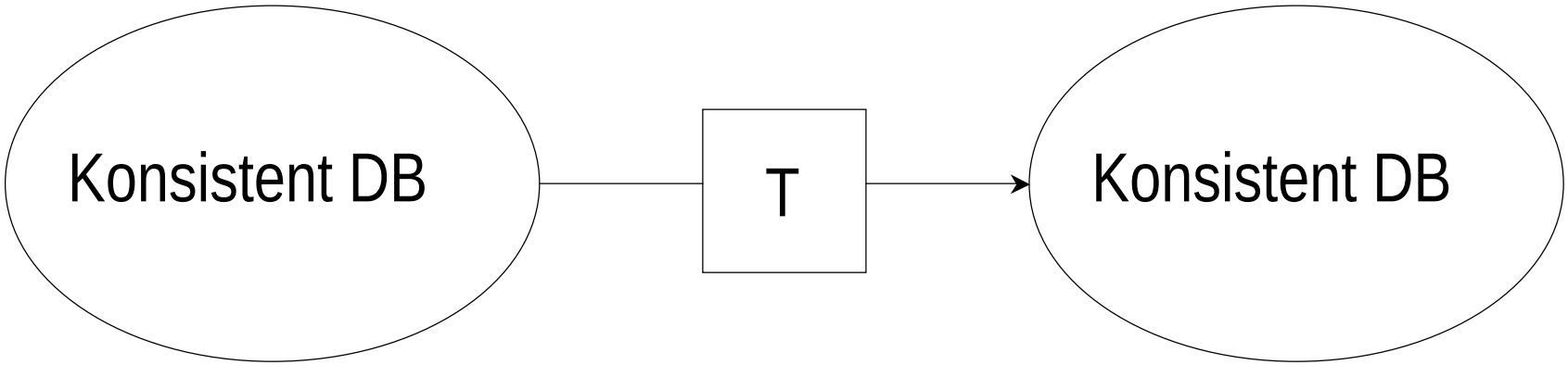
Sett inn 1000 kroner på a_2: $a_2 \leftarrow a_2 + 1000$

$\text{total} \leftarrow \text{total} + 1000$



Transaksjoner

En ***transaksjon*** er en samling aksjoner som bevarer konsistens



- **Viktig antagelse**

  Hvis T starter i en konsistent tilstand,
  og T eksekverer alene,
  så vil T avslutte med å etterlate
  databasen i en konsistent tilstand

- Korrekthet (uformelt):
  - Hvis vi stopper å utføre transaksjoner, etterlater vi DB i en konsistent tilstand
  - Alle transaksjoner opplever en konsistent DB

Årsaker til inkonsistens og brudd på integritetsreglene

- Feil i en transaksjon
- Feil i DBMS
- Hardware-feil
  - F.eks.: Et diskkræsj endrer saldo på en konto
- Deling av data
  - F.eks.: T1: Øk lønnen til alle med tittel "programmerer" med 4,5%
    T2: Forandre Tove Hansens tittel fra "programmerer" til "systemerer"

Viktige ting INF3100 ikke omfatter

- Hvordan å skrive/programmere feilfrie transaksjoner
- Hvordan å skrive/programmere et feilfritt DBMS
- Hvordan å kontrollere integritetsreglene i databaseskjemaet og rette opp brudd på dem

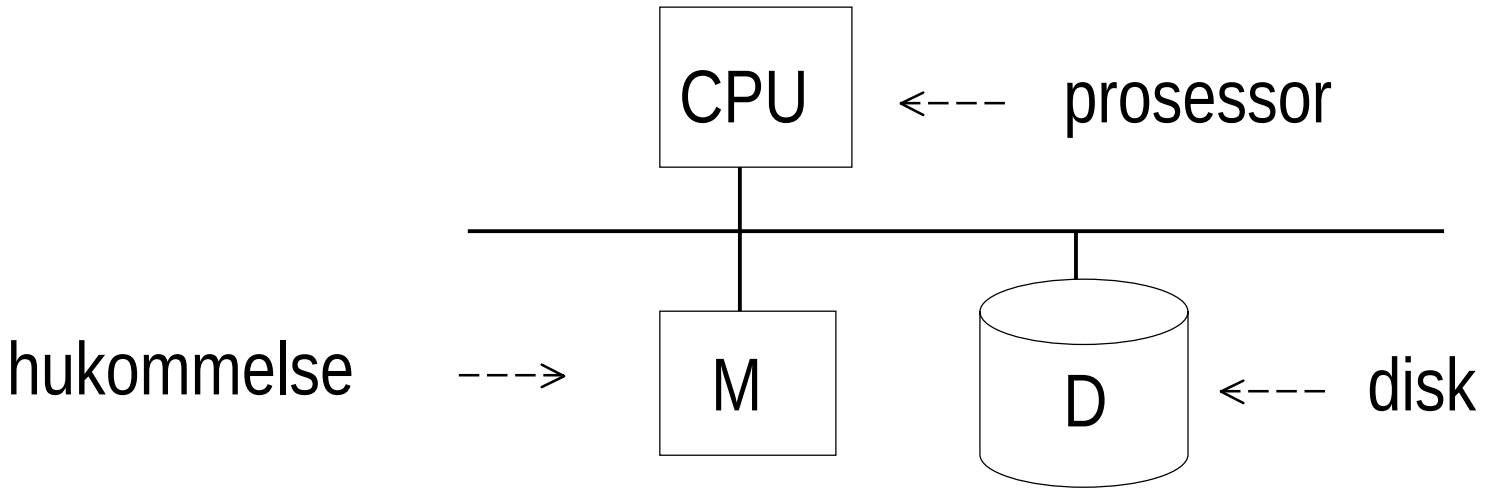
  (Det betyr at de løsningene vi gir på håndtering av feilsituasjoner, ikke trenger å ta hensyn til hvilke integritetsregler som er definert i databaseskjemaet)

Gjenoppretting (Recovery)

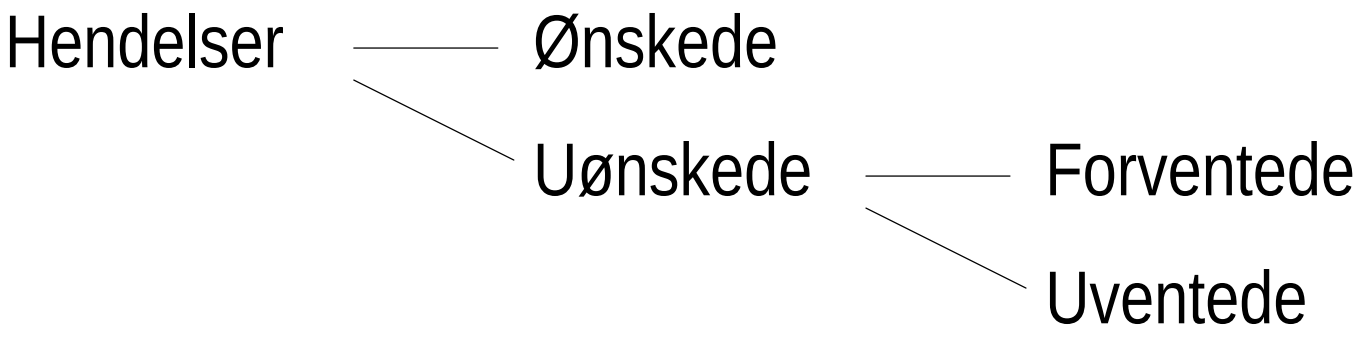
Første punkt på agendaen:

En modell for beskrivelse av feil

Del 1: Konfigurasjonsmodellen:



Klassifikasjon av hendelser



- Ønskede hendelser: Se produkthåndbøkene …
- Forventede uønskede hendelser:
  - Systemkræsj:
    - tap av hukommelsen
    - CPUen stopper eller må resettes

det er det hele!

- Uventede uønskede hendelser: Alt annet!

Uventede uønskede hendelser: Alt annet!

Eksempler:

- Diskdata går tapt
- Hukommelsen går tapt uten at CPUen stopper
- CPUen tar kontroll over alle datamaskiner hos Viken Energinett, setter opp spenningen og brenner av alle strømledninger i Oslo og Akershus
- Kapittel 1 i “The Hitch Hiker’s Guide to the Galaxy” viser seg å ikke være Science Fiction likevel ….

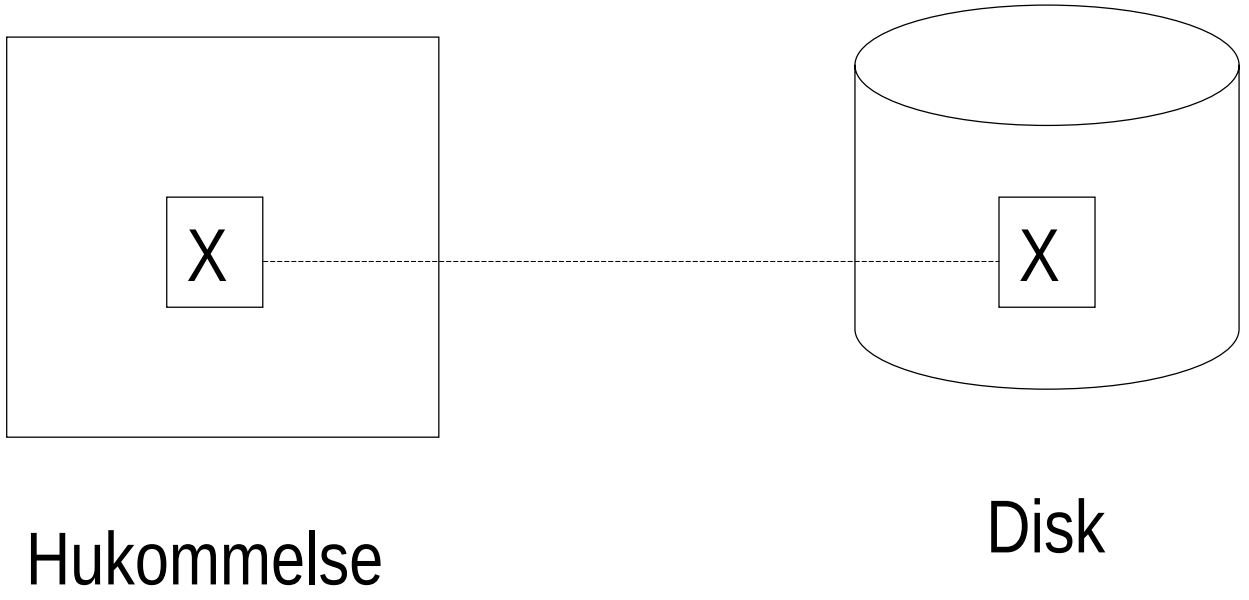
Et forslag til løsning

Strategi: Adder lavnivå kontroller og redundans for å øke sannsynligheten for at modellen holder

Eksempler: Replikert disk (stable store)
Paritetssjekk av hukommelsen
CPU-sjekker

Andre punkt på agendaen

- ***Lagringshierarkiet:***



Operasjoner

- Input(x): diskblokk med x → hukommelsen
- Read(x,v): utfør, om nødvendig, input(x)
  v ← verdien av x i blokken
- Write(x,v): utfør, om nødvendig, input(x)
  verdien av x i blokken ← v
- Output(x): hukommelsesblokk med x → disk

Ufullførte transaksjoner

- Dette er hovedproblemet i transaksjonshåndtering

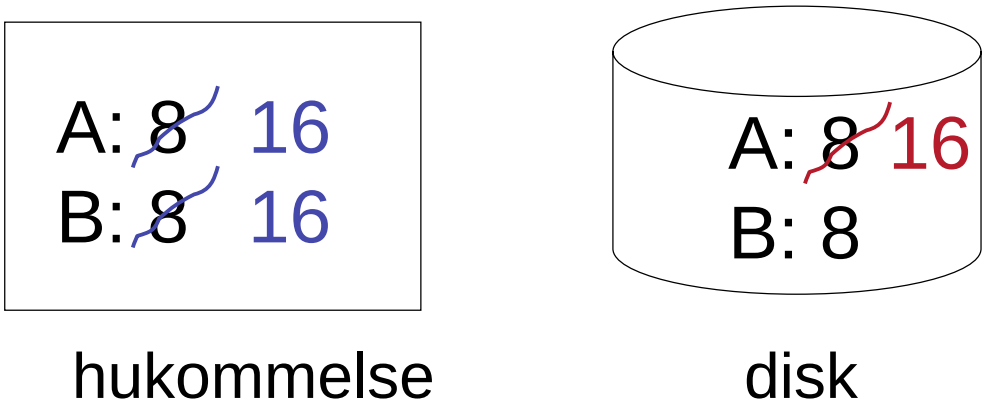
- Eksempel: Integritetsregel: $A = B$

T1: $A \leftarrow A \times 2$
$B \leftarrow B \times 2$

Ufullførte transaksjoner (forts.)

T1: Read(A,t); t ← t × 2;
Write(A,t);
Read(B,t); t ← t × 2;
Write(B,t);
Output(A);
Output(B);

feil!



Atomisitet

- Transaksjoner må være ***atomære***
- Dette gir oss bare to muligheter:

1) Å utføre alle operasjonene i transaksjonen

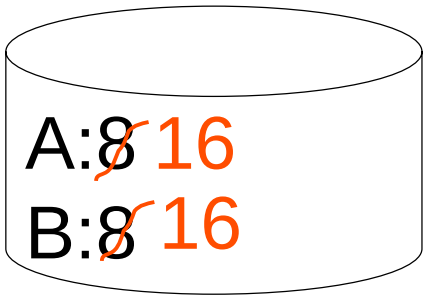
   eller

2) Å ikke utføre noen av operasjonene i transaksjonen

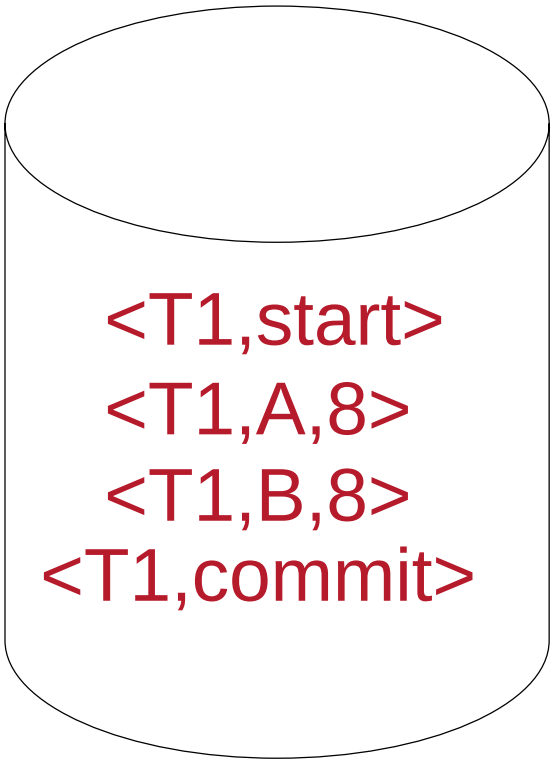
Undo-logging

T1: Read(A,t); $t \leftarrow t \times 2$; Invariant: A = B
Write(A,t);
Read(B,t); $t \leftarrow t \times 2$;
Write(B,t);
Output(A);
Output(B);

hukommelse



disk

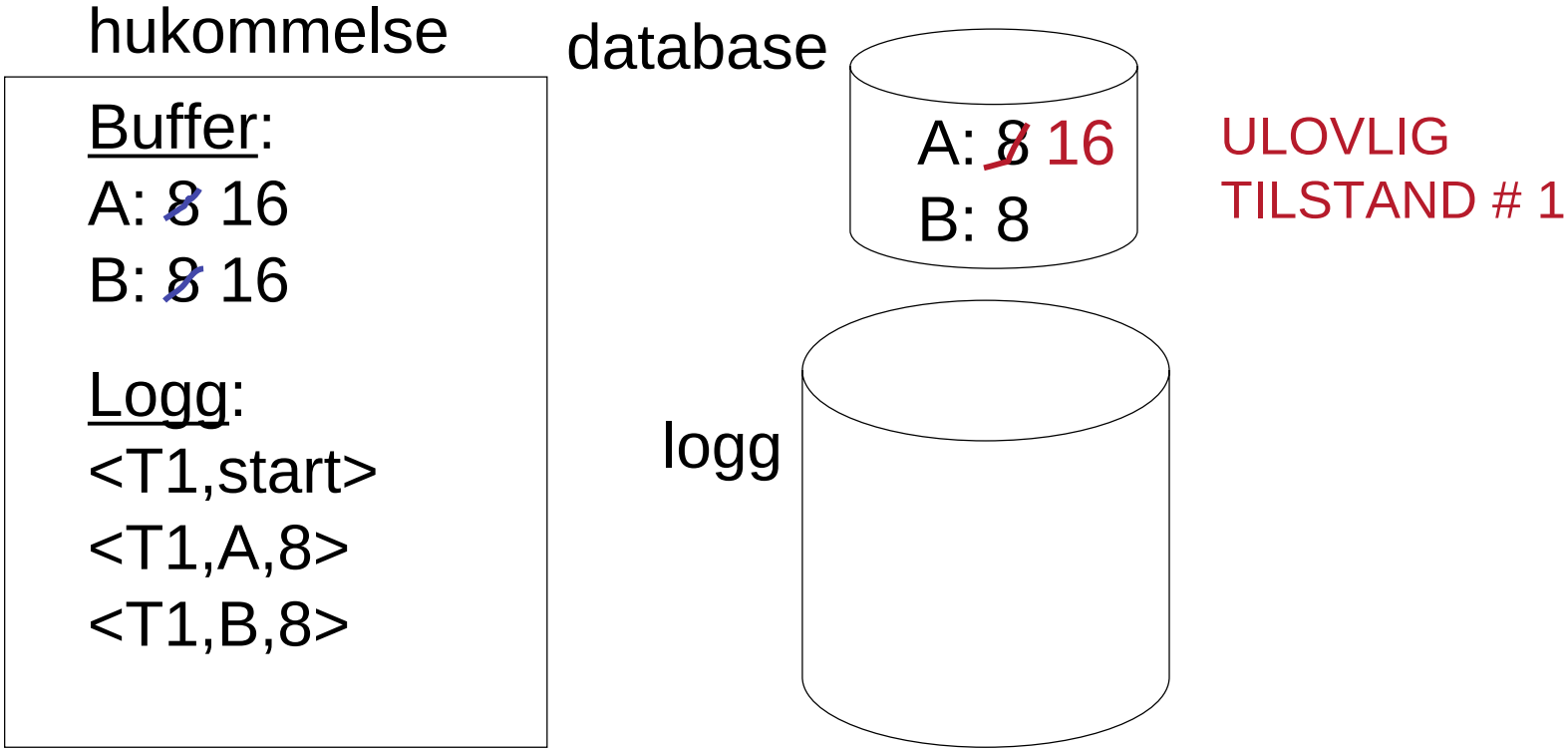


logg

Undo-logging (forts.)

En kompliserende faktor:

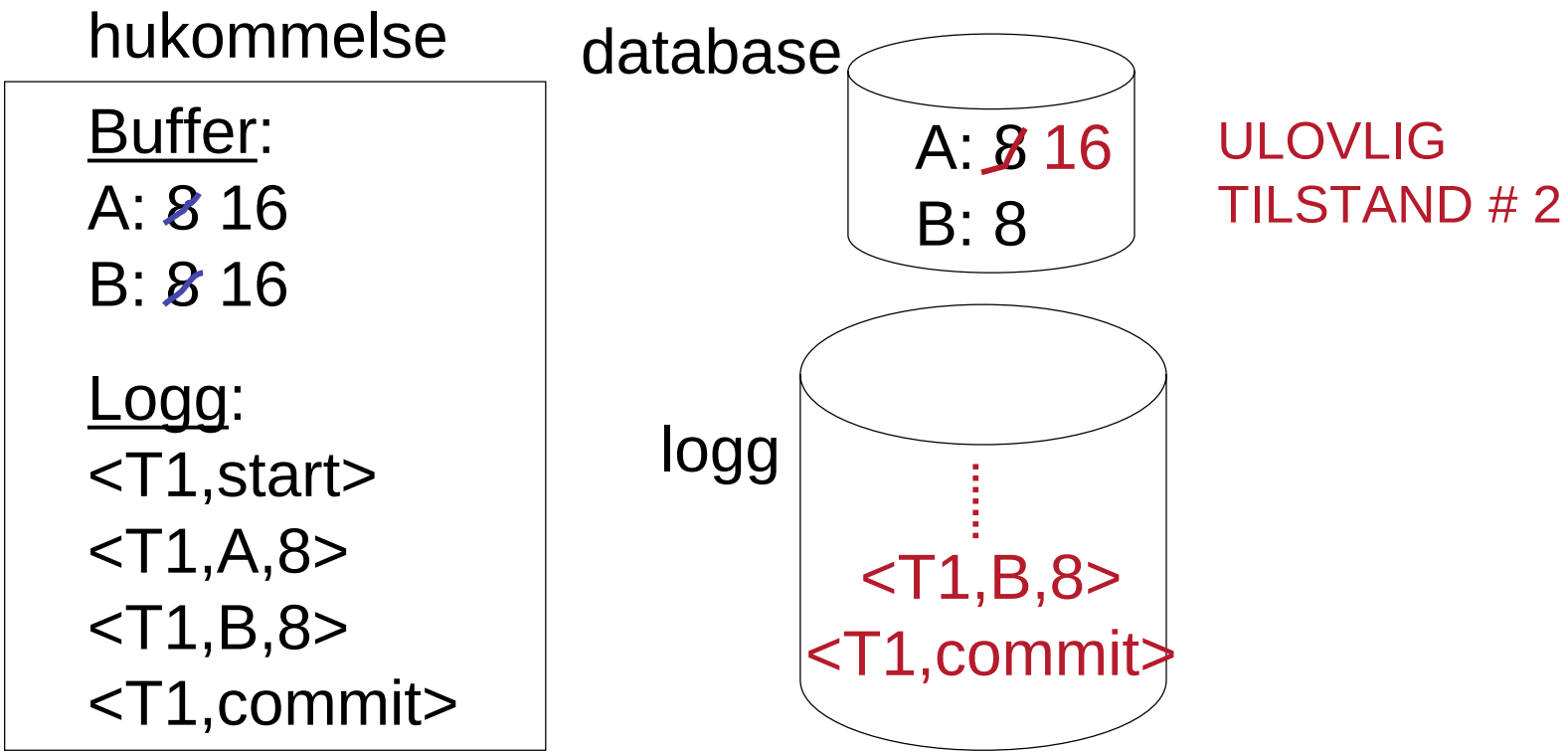
- Loggen skrives i hukommelsen ***før*** den skrives til disk
- Loggen skrives ikke til disk for hver enkeltoperasjon



Undo-logging (forts.)

En kompliserende faktor:

- Loggen skrives i hukommelsen ***før*** den skrives til disk
- Loggen skrives ikke til disk for hver enkeltoperasjon



Undo-logging (forts.)

- Regler for undo-logging:
  1) For hver skriveoperasjon write(X,v):
     Skriv en linje i loggen som inneholder
     den gamle verdien av X
  2) Før X endres på disken, må alle logglinjer
     som gjelder X, være skrevet til disk
     (Strategi: Skriv først til logg, så til disk)
  3) Før commit kan skrives i loggen, må alle
     skriveoperasjonene i transaksjonen være
     overført til disken
- Strategi: Alt må oppdateres før commit

Undo-logging (forts.)

- Algoritme for gjenoppretting :
  1) Sett S = mengden av transaksjoner T_i hvor $<T_i$, start>, men hverken $<T_i$, commit> eller $<T_i$, abort> finnes i loggen
  2) Les loggen i bakvendt orden (fra sist til først). For hver $<T_i$, X, v> i loggen:
     Hvis $T_i \in S$ så
     write(X,v)
     output(X)
  3) For hver $T_i \in S$:
     Skriv $<T_i$, abort> i loggen

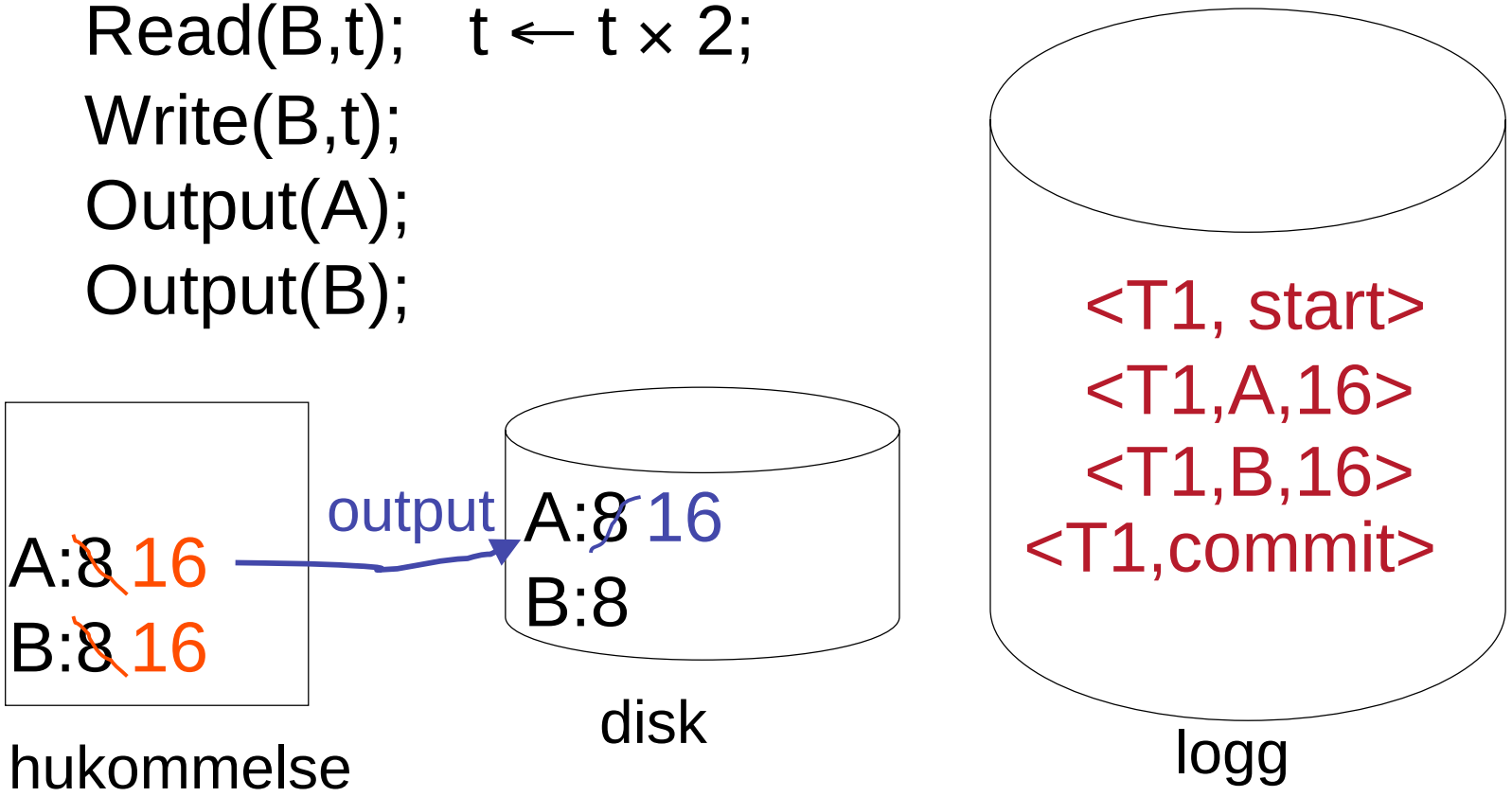
Undo-logging (forts.)

- Hva hvis det skjer en feil under gjenoppretting?

- Ikke noe problem!
  Undo er idempotent
  (Det betyr at å gjøre undo mange ganger gir samme resultat som å gjøre undo én gang)

Redo-logging

T1: Read(A,t); $t \leftarrow t \times 2$; Invariant: A = B
Write(A,t);
Read(B,t); $t \leftarrow t \times 2$;
Write(B,t);
Output(A);
Output(B);



Redo-logging (forts.)

- Regler for redo-logging:

  1) For hver skriveoperasjon write(X,v):
     Skriv en linje i loggen som inneholder
     den nye verdien av X
  2) Flush loggen (dvs. skriv loggen til disk) ved commit
  3) Før X endres på disken, må alle logglinjer som
     gjelder X (inklusiv commit), være skrevet til disk

- Strategi:
  Skriv ikke til disk før loggen er ferdigskrevet

Redo-logging (forts.)

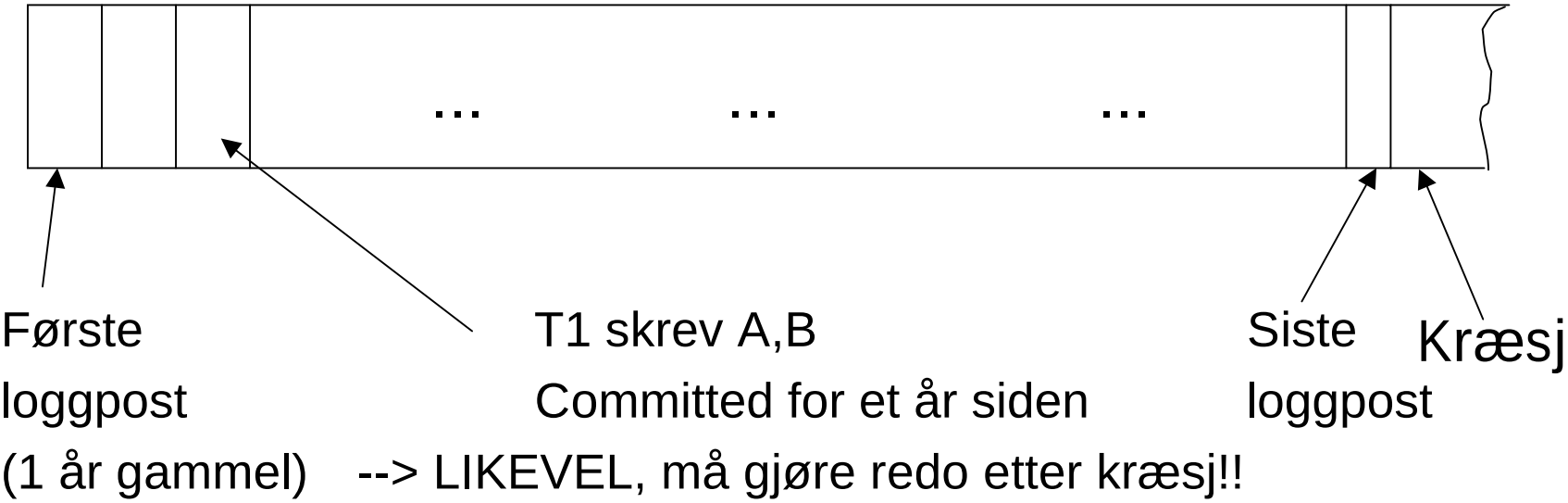
- Algoritme for gjenoppretting :
  1) Sett S = mengden av transaksjoner T_i hvor $<T_i$, commit> finnes i loggen
  2) Les loggen forlengs (fra først til sist).
     For hver $<T_i$, X, v> i loggen:
     Hvis $T_i \in S$ så
     write(X,v)
     output(X)

ikke nødvendig

Redo-logging (forts.)

- Med tiden blir gjenoppretting en **TREG** prosess

Redo-logg:



Redo-logging (forts.)

Løsning: Sjekkpunkt (enkel versjon)

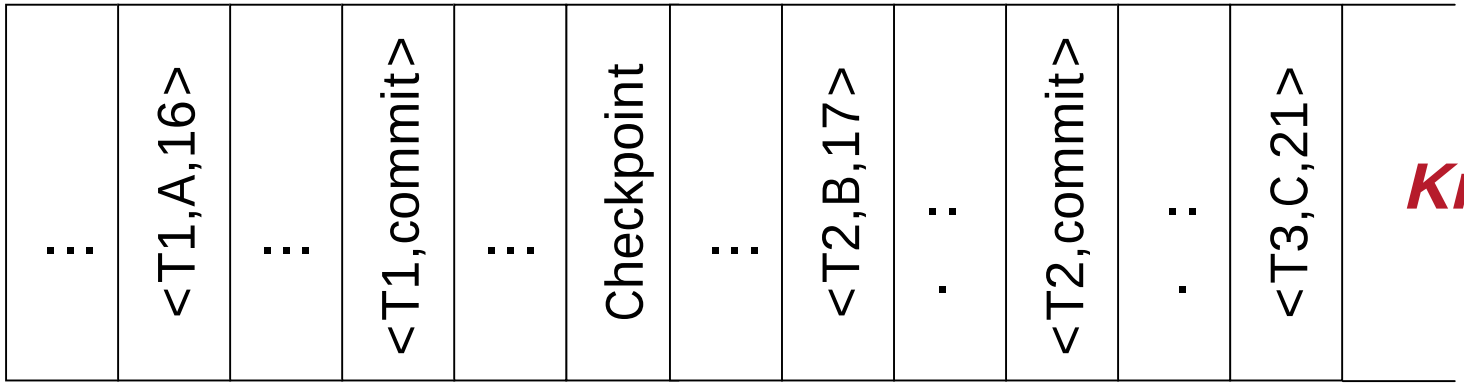
Utfør regelmessig:

1) Stans start av nye transaksjoner
2) Vent til alle transaksjoner har avsluttet
3) Skriv alle loggposter til disk (flush loggen)
4) Skriv alle buffere til disk (DB)
   (behold dem i hukommelsen)
5) Skriv en sjekkpunkt-post til disk (i loggen)
6) Gjenoppta transaksjonsbehandlingen

Redo-logging (forts.)

Eksempel: Hva må gjøres ved gjenoppretting?

Redo-logg (disk):



Undo- vs redo-logging

- Hovedproblemene med de to strategiene er:
- *Undo-logging:* Vi kan ikke holde en backup-DB oppdatert til enhver tid
- *Redo-logging:* Vi må holde alle oppdaterte blokker i hukommelsen helt til commit (og litt til)

Undo- vs redo-logging (forts.)

- Løsningen er: *Undo/redo-logging*

- For alle skriveoperasjoner skriver vi i loggen:
  <T_i, X, ny X-verdi, gammel X-verdi>
- I en undo/redo-logg er X alltid en hel blokk (page)

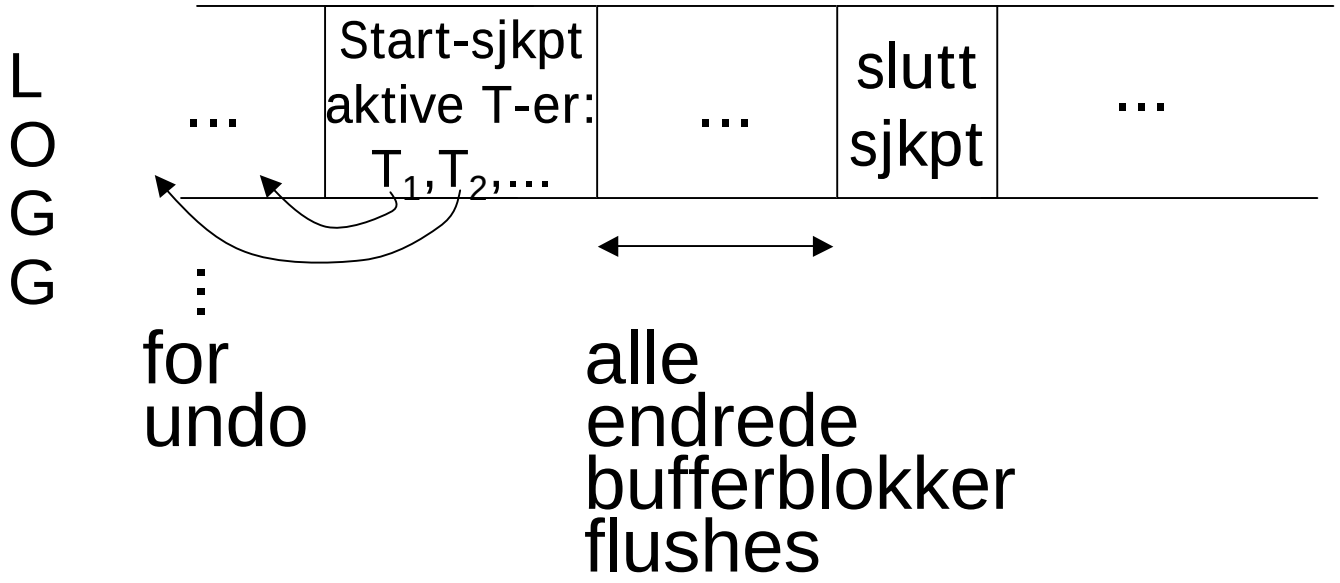
  (Det er for å unngå problemer som at A og B ligger i samme blokk og blir oppdatert av hver sin transaksjon, og så comitter den ene av transaksjonene.
  Da både skal og skal ikke blokken skrives til disk.)

Undo/redo-logging

- Regler for undo/redo-logging:
  1) Ingen skriveoperasjon write(X,v) får utføres før transaksjonen vet at den ikke må abortere
  2) For hver write(X,v) (der X er en blokk):
     Skriv en linje (post) i loggen som inneholder både den gamle og den nye verdien av X
  3) Skriv loggposten til disk før X oppdateres i DB
  4) Flush loggen ved commit
- Merk: Blokken X kan skrives til disk både før og etter at transaksjonen committer.
  De eneste kravene som må oppfylles før X skrives, er punkt 1 og 3 ovenfor

Ikke-passiviserende sjekkpunkt

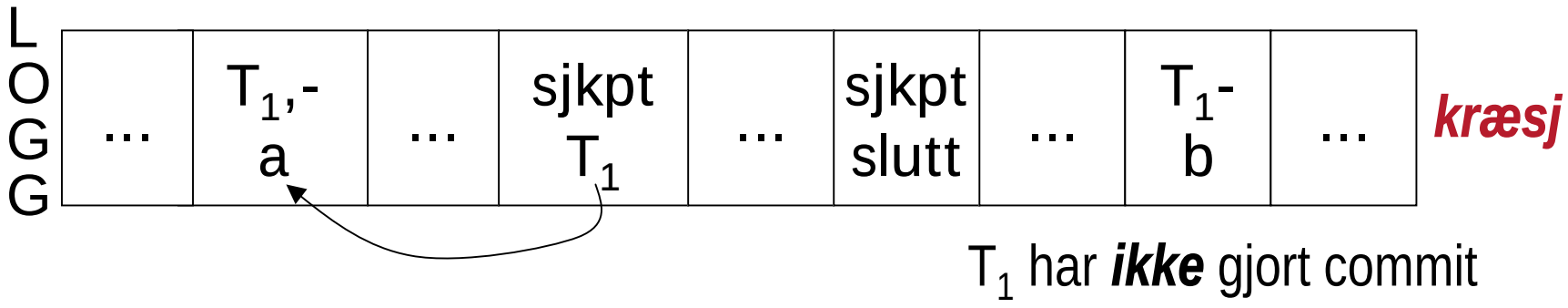
Sjekkpunkt som ikke hindrer start av nye transaksjoner:



Sjekkpunktet er slutt når alle endrede bufferblokker er skrevet til disk. Loggen flushes både ved start og stopp

Ikke-passiviserende sjekkpunkt (forts.)

Eksempel: Hva gjøres ved gjenoppretting?

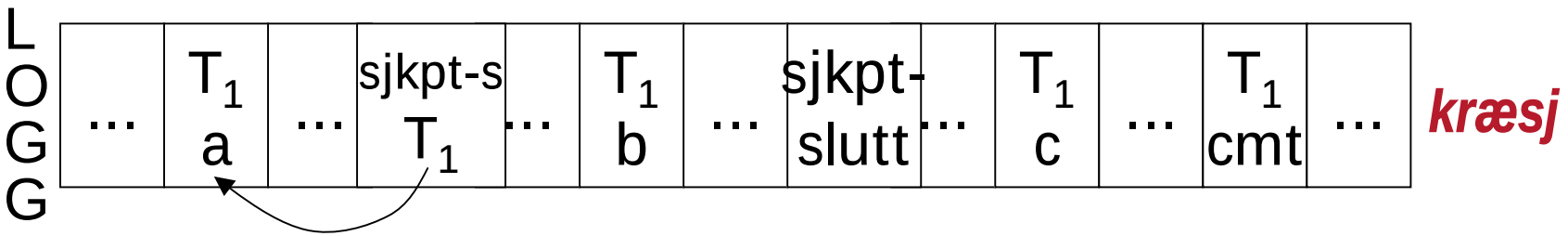


⊗ Undo T_1 (undo a,b)

Husk: Undo tas bakfra frem til starten av eldste ikke-comittede transaksjon

Ikke-passiviserende sjekkpunkt (forts.)

Eksempel: Hva gjøres ved gjenoppretting? (forts.)

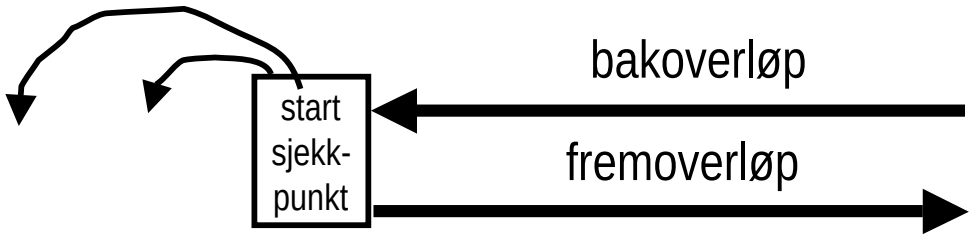


⊗ Redo T_1 (redo b,c)

Husk: Redo tas forfra fra siste sjekkpunkt-start

Undo/redo-gjenopprettingsprosessen

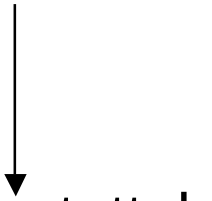
- Bakoverløp (fra loggslutt til siste start sjekkpunkt):
  - bygg opp en mengde S av committede transaksjoner
  - gjør Undo på alle operasjoner gjort av transaksjoner som ikke er i S
- Oppryddingsfase (foran siste start sjekkpunkt):
  - gjør Undo på resten av operasjonene gjort av de transaksjonene i sjekkpunktets aktiv-liste som ikke er i S
- Fremoverløp (fra siste start sjekkpunkt til loggslutt):
  - gjør Redo på operasjonene gjort av transaksjonene i S



Fysiske hendelser

Eksempel: Uttak av penger i en bankautomat

$T_i = a_1\ a_2\ \ldots\ a_k\ \ldots\ a_n$



kontantuttak

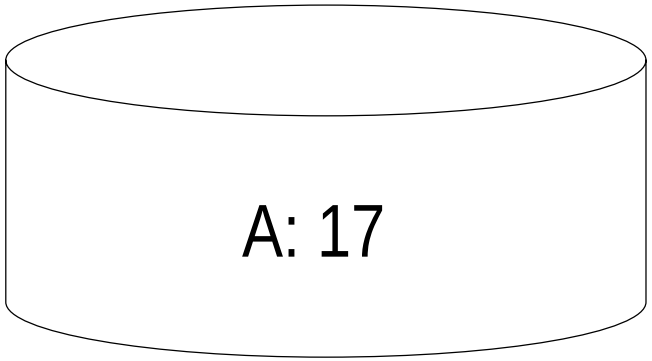
Løsning:

1) utfør fysiske operasjoner etter commit

2) prøv å gjøre fysiske operasjoner idempotente

Håndtering av fysiske feil

Eksempel: Diskkræsj eller lignende ødeleggelse av et permanent datalagringsmedium

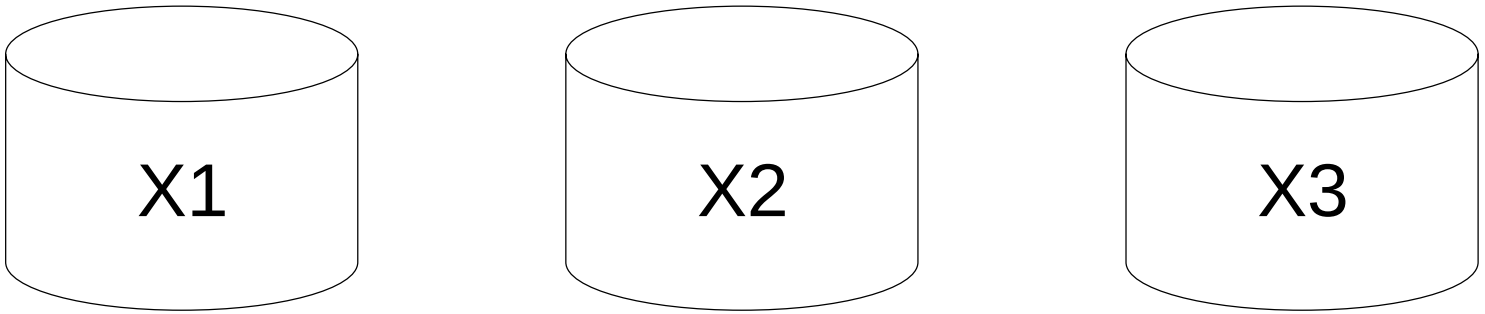


Løsning: Lag kopier av data!

Håndtering av fysiske feil (forts.)

Forslag 1: Fysisk trippel-lagring av alle data

- Ha tre kopier på hver sin disk
- Output(X) ⇒ tre fysiske output(X)
- Input(X) ⇒ tre fysiske input(X) **+** votering



Håndtering av fysiske feil (forts.)

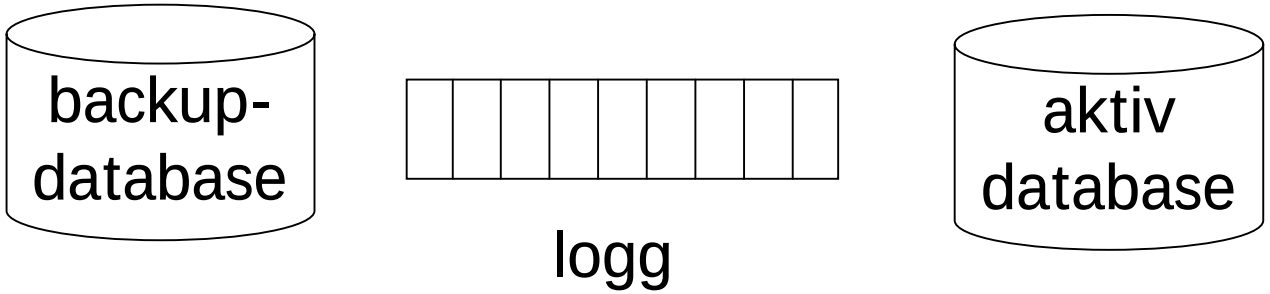
Forslag 2: Redundant skriving, enkel lesning

- Ha N kopier på hver sin disk
- Output(X) ⇒ N fysiske output(X)
- Input(X) ⇒ en fysisk input(X)
  - hvis OK, ferdig
  - ellers, prøv en annen

Strategien forutsetter at korrupte data kan oppdages

Håndtering av fysiske feil (forts.)

Forslag 3: Dump av DB **+** logg



- Hvis den aktive databasen går tapt, så
  - kopier backupdatabasen til en ny aktiv database
  - bruk redo-dataene i loggen til å gjøre databasen up-to-date

Sletting av loggfilen