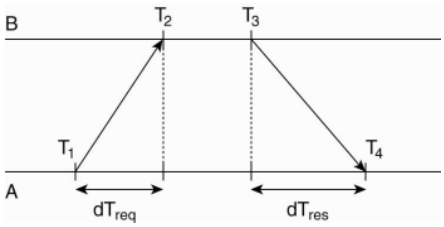
5. Time

Tid er ikke ens for maskiner i et DS – men tid bruges til at synkronisere, derfor skal en fælles tid findes eller estimeres.

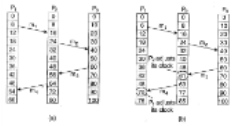


Fysisk tid

- Cristian's algorithm (kun en-vejs synkronisering)
  - Offset: $\theta = \left(T_3 + \frac{(T_2-T_1)+(T_4-T_3)}{2}\right) - T_4 = \frac{(T_2-T_1)+(T_3-T_4)}{2}$
- Stilles afhængigt af θ. Hvis uret skal "stilles" tilbage, skal det gøres gradvist, ved at "sløve" det.
- NTP (to-vejs synkronisering)
  - Delay: $\delta = \frac{(T_2-T_1)+(T_4-T_3)}{2}$
  - 8 (offset, delay)-par udregnes. Det par med mindst delay væges.
  - Nogle maskiner bedre end andre til at bestemme tid. Servere opdeles i grupper efter deres Stratum-level. En server med et reference ur (WWV-modtager) er en stratum-1 server og har stratum 0. Når 2 servere synkroniserer, bruges den med det laveste stratum k som reference for den anden. Den vil derefter have stratum k+1.

Logisk tid

- Ikke interesseret i specifik tid, men bare rækkefølgen af hændelser.
- Happens-before-relation: a → b (transistiv relation)
  - Gælder inden for samme proces
  - Hvis events har noget med hinanden at gøre inden for flere processer
  - Men ikke hvis to events ikke har noget med hinanden at gøre → concurrent
- Tid herfra: a→b => C(a)<C(b).
- Lamport's algoritme justerer tiden (frem) udfra tiderne fra processerne så ovenstående holder.
  1. Tid for afsendende proces P_i sættes til C_i=C_i+1
  2. Besked m sendes med ts(m)=C_i
  3. Den modtagende proces P_j sætter sin tid til C_j=max{C_j,ts(m)} **[TRANSPARENT]**



- Fanger ikke om events afhænger af hinanden (kausalitet).
- **Totally ordered multicast:** Alle processer synkroniserer på beskeder, så til sidst vil de være enige om en tid – samt være enige om, hvilke opgaver der skal udføres først. Opgaver holdet i en kø, sorteret efter timestamp. Da de multicaster bekræftelse, ved alle, hvornår de kan udføre en besked.

Vektor ure

- Kan bruges til at fange kausalitet imellem hændelser (som mangler i Lamports logiske ure)
- Hvis VC(a)<VC(b), så kommer a kausalt før b.
- Hver proces P_i opretholder et VC_i:
  1. Før hver hændelse i Pi: VCi[i] = VCi[i] + 1
  2. P_i: m -> P_j - ts(m) = VC_i efter (1) er forekommet (P_i's viden om alle andre processer)
  3. P_j modtager m – for alle k VC_j[k] = max(VC_j[k] , ts(m)[k]), derefter (1) og aflever besked til applikation
     - Bemærk ts(a) – 1 betegner alle hændelser der kausalt kommer før m **[TRANSPARENT]**
- **Causally ordered multicast:** Besked modtagelse vil blive forsinket indtil:
  1. Vi ved at vi kun mangler den ene besked der kommer nu.
  2. Vi har set samme antal eller flere beskeder fra alle andre processer end den afsendende.