Chapter 6.4 Synchonization in UDBS

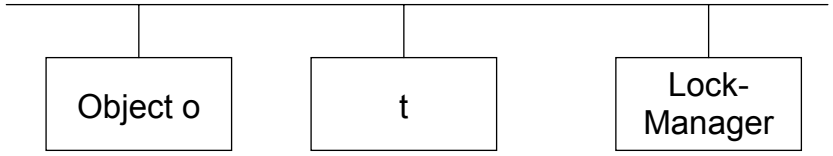
Chapter 6.4.1 Distributed Graph:

- Every data object has home node:
  H(y) for y
  H(y) responsible for locks on y,
  knows G(y): granted group
  Q(y) or W(y)
- alternatively: Trans. t has H(t) and H(t) knows
  $t_2 \ldots \rightarrow t_1 \rightarrow t$

Problem: Communication for locks: how to collect G_w for cycle search?
i. e. distributed cycle search via several computer nodes

Chapter 6.4.2 Central Graph and Lock Manager

Communication for lock request, lock and object are on different nodes.



(t, o, μ)

Chapter 6.4.3 Timeout

if lock request (t, o, μ) ist not granted after s seconds
then abort t
Used commercially!
Safe solution: all t in cycle wait, oldest is aborted.

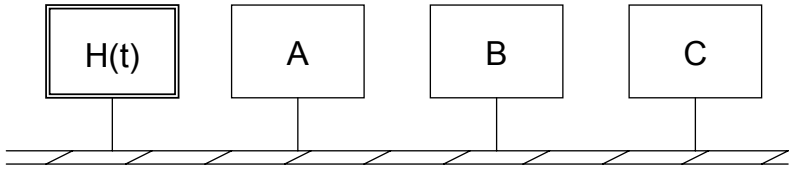
Chapter 6.4.4 Time Stamps and Time Intervals

- t gets time stamp T(t), unique system wide.
- e. g. of BOT
- e. g. of 1. conflict of locks, dynamic assignment:
  let $t_2 \rightarrow t_1$,
  try: $T(t_2) > T(t_1)$
- deadlock:
  $t_2 \rightarrow t_1 \wedge T(t_2) < T(t_1)$
  check is simple, t carries T(t) with itself
- dynamic time intervals

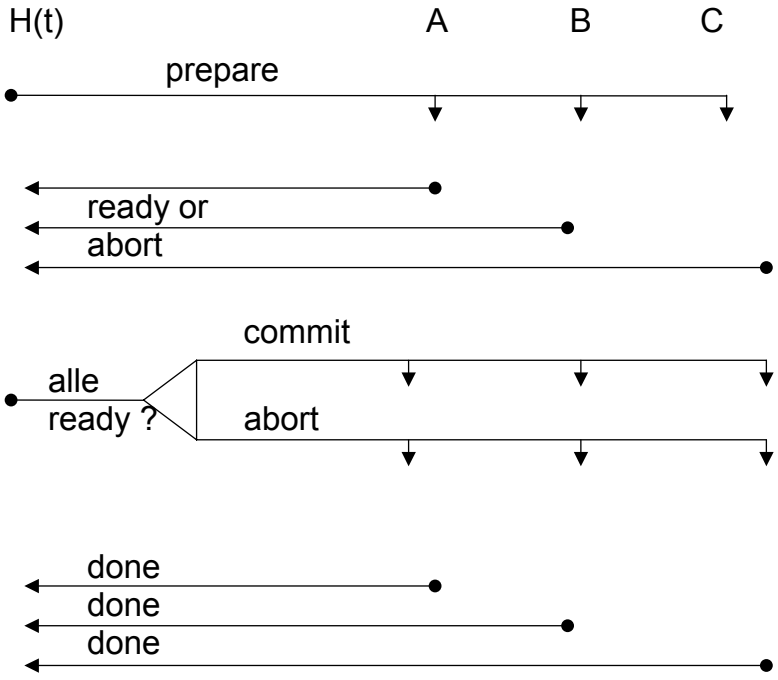
Communication for lock request, no G_w, no cycle search.

H(t) issues and knows T(t) resp. time interval I(t).
Optimization of comminication?

Chapter 6.4.5 2-Phase Commit-Protocol



H(t) is commit-coordinator, determined somehow



delete t-context on H(t)

Exception Handling

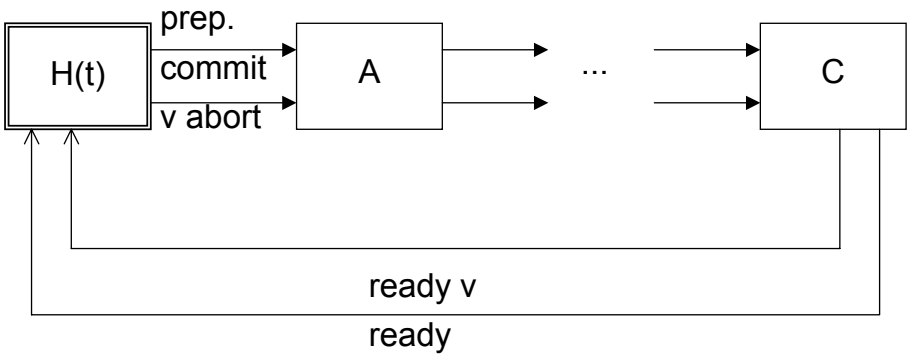
- timeout for "ready/abort" messages:
  ⇒ abort all
- if "done" message is not received:
  ⇒ commit(t) resp. abort(t) are idempotent, i.e. repeat until acknowledgement is received.

Comm.-Cost:

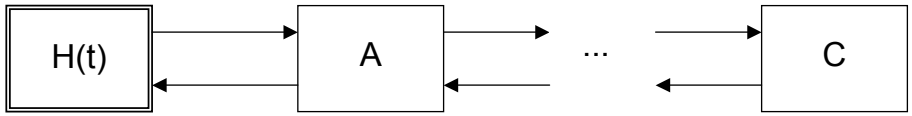
4 • (n-1) messages for n participating nodes for simple 2-phase commit

⇒ cheaper?

Linear 2-phase Commit



⇨ H(t) makes **commit** or **abort** decision, cost: 2*n.
H(t) stores general management info.



⇨ last node C makes decision, cost: 2 * (n-1)

H(t) stores management info