The Dictionary Problem by Wuu & Bernstein

· Replicated Log to achieve mutual consistency

· The log contains: record of updates on the objects.

<u>Model</u>

- same as before - N nodes.
- $(E, \rightarrow)$

  C1. events at the same node are totally ordered

  C2. send(e_1) and receive(e_2): $e_1 \rightarrow e_2$.

The Log Problem.

```
type Event =
        record
            op : OpnType
            time: TimeType
            node: NodeID.
```

event record of e eR.

eR.op op(e) — operation

eR.time time(e) —

eR.node node(e)

$f \rightarrow e$ iff $fR \in L(e)$

Dictionary

- Same as before.

P2. $x \in V(e)$ iff $e_x \to e$ $\not\exists$ x-delete event e' s.t. $e' \to e$.

$$V(e) = \{ x \mid e_x R \in L(e) \land \not\exists\, gR \in L \text{ s.t. } gR.op = delete(x) \}$$

O1. The entire log is sent in each message.

O2. A new view is repeated computed

O3. The entire log is maintained

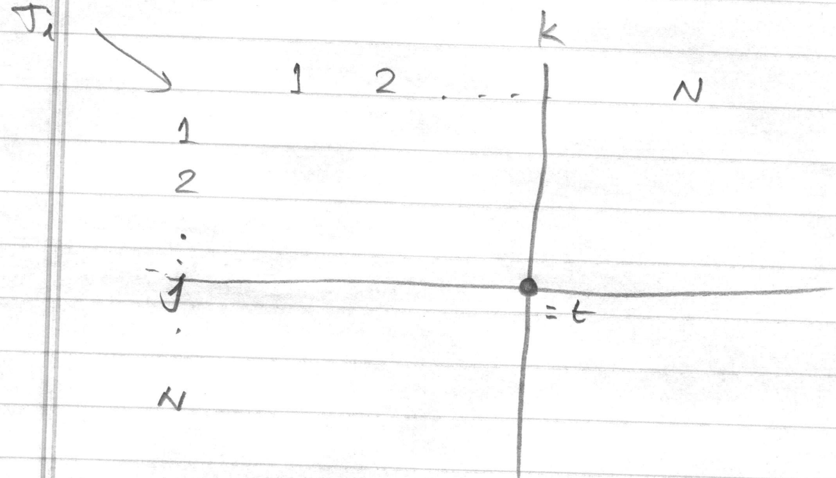
Each node N_i maintains:

1. $Clock_i$

2. A two dimensional time-table T_i

$T_i[1..N][1..N]$.

$T_i[j,k]$



$T_i[j,k] = t$ – N_i knows that N_j has learned of all events ~~up to~~ at N_k up to time $\underline{t @ N_k}$

$T_i[i,i] =$ N_i's $clock_i$.

$T_i[i,k]$ – N_i's knowledge of events at N_k.

3. A copy of the Log.

$$hasrecvd(T_i, eR, k) \equiv T_i[k, eR.node] \geq eR.time$$

$\Rightarrow$ N_i knows that N_k has learned of e.

Init

$V_i = \emptyset$

$PL_i = \emptyset$

$T_i[*,*] = \emptyset$

insert(x):

$T_i[i,i] = clock_i{+}{+}$

$PL_i = PL_i \cup \{< ins(x), T_i[i,i], \ i >\}$

$V_i = V_i \cup \{x\}$

delete(x):

$T_i[i,i] = clock_i{+}{+};$

$PL_i = PL_i \cup \{< del(x), T_i[i,i], \ i >\}$

$V_i = V_i \setminus \{x\}.$

send(m) to N_K.

$NP = \{eR \mid eR \in PL \wedge \neg hasrecvd(T_i, eR, k)\}$

send $\langle NP, T_i \rangle$ to N_K.

receive $\langle NP_k, T_k \rangle$ from N_k.

$$NE = \{ fR \mid fR \in NP_k \wedge \neg hasrecord(T_i, fR, i) \}$$

$$V_i = \{ x \mid x \in V_i \text{ or } e_x R \in NE \}$$
$$\wedge$$
$$\{ \nexists\, dR \in NE \text{ s.t. } dR.op = del(x) \}.$$

$$\forall I \text{ do } T_i[i, I] = \max \{ T_i[i, I], T_k[k, I] \}.$$

$$\forall I, J \quad T_i[I, J] = \max \{ T_i[I, J], T_k[I, J] \}$$

$$PL_i = (eR \mid eR \in PL_i \cup NE) \wedge$$
$$(\exists j \text{ s.t. } \neg hasRecord(T_i, eR, j))$$

Global Snapshots in Distributed Systems.

Source: Chandy & Lamport
ACM TOCS, 3(1): 63-75, 1985.

Motivation:
- global state of a distributed computation.

- why: i) deadlock in the system?
ii) computation has terminated?
iii) checkpointing the system state (why?)

Why is it difficult?
- process can record its own state + the messages it sends + receives.
- it can record nothing else.
- processes must cooperate with others
- all processes cannot record their local states at the same time.
- no common clock / no shared memory.

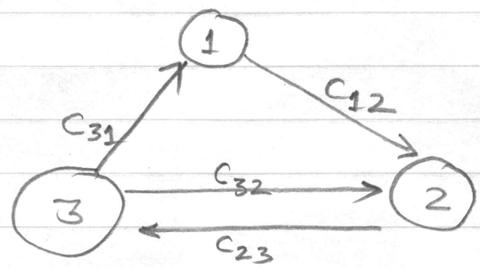
- the problem is compute global snapshot: local states + comm. channel states.

Finally, should not impede/interfere with the underlying computation.

System Model.

A distributed system:

- a finite set of processes, and
- a finite set of channels



Channels:

- infinite capacity
- error-free
- deliver messages in the order sent

The <u>state</u> of a channel:

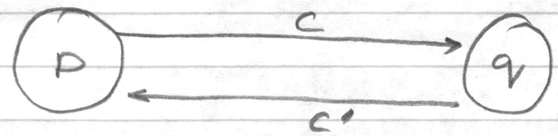
sequence of messages sent excluding the messages received

$$\underbrace{m_1 m_2 m_3}_{\text{sent}} \longrightarrow \underbrace{m_1}_{\text{received}}$$

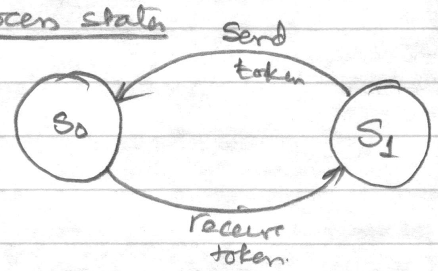
State: $\underline{m_2 \cdot m_3}$

Example 2.1

System.

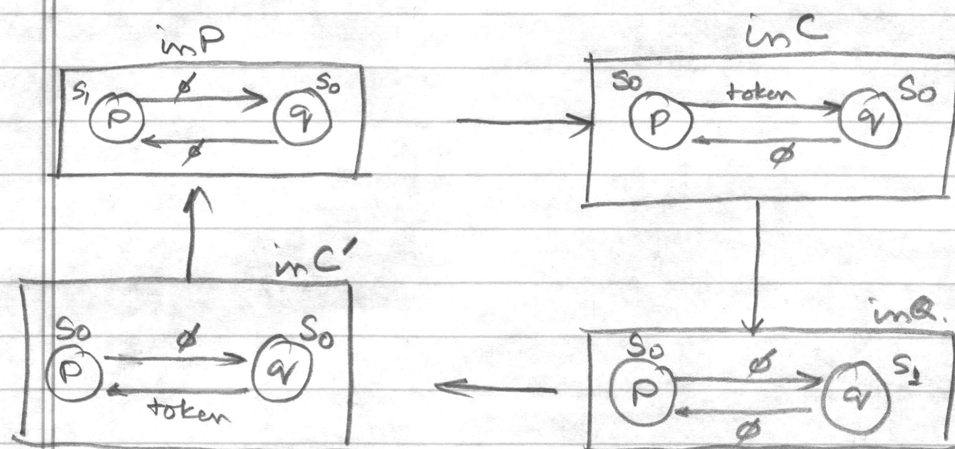


process states



single-token conservation

p in S1; q in S0.



The Algorithm.

- use an example to motivate the steps of the algorithm.
- assume that can record the state of the channel instantaneously.

Ex.

$\boxed{\text{record state of } p}$ in in-P.

↓

in-C

↓

record: state of q

state of c

state of c'

⟹ 2 tokens in-P + in-C.

Inconsistency: state of p **before** p sent a message along c

+

State of c **after** p sent the message.

Let n be the # of messages sent along c before p's state rec

Let n' be the # " " " c before c's state

Global state inconsistent if

$n < n'$.

Alternate scenario:

State(c) recorded in-P,

State(P), State(q), State(c') recorded in-C.

$\Rightarrow$ No token.

Inconsistent if c recorded before P sends a message along c and state of P after P send that if

$$n > n'$$

$\Rightarrow$ Consistent global state

$$n = n'.$$

Similarly if m = # of messages recvd along c before q
m' = # of messages recvd along c ~~after~~ before c' state is recorded.

$m = m'$ for consistency.

$$\underbrace{n'}_{\text{sent}} \geq \underbrace{m'}_{\text{recvd}}$$

$$n \geq m$$

Alg.

Marker-Sending Rule for a Process p.

For each output channel c of p:

p sends a "marker" along c after recording its state and before sending any further messages.

Marker-receiving Rule for a Process q:

On receiving a marker along a channel c:

if q has not recorded its state then:
    q records its state
    q records the state c as the empty seq.
    q sends marker on all outgoing c'
else
    q records the state of c as the seq. of messages recvd. along c after q's state was recorded and before q recvd. the marker along c.