Wait-freedom vs. t-resiliency and the robustness of wait-free hierarchies

T. Chandra, V. Hadzilacos, P. Jayanti, S. Toueg

PODC 1994

Shared Objects

- Shared objects
  - Defined by input/output interface, and sequential specification
  - Accessed by *n* asynch, fault-prone processes
  - Output is linearizable
  - Is wait-free if object returns output when *n*-1 processes fail
  - Ex. register, test&set, queue
  - Ex. n-process consensus, cons(p_1,…,p_n)
    - p_1,…,p_n each *propose* 0 or 1, and gets response *u*
    - If first op applied is *propose u*, every op gets response *u*

A Hierarchy of Shared Objects

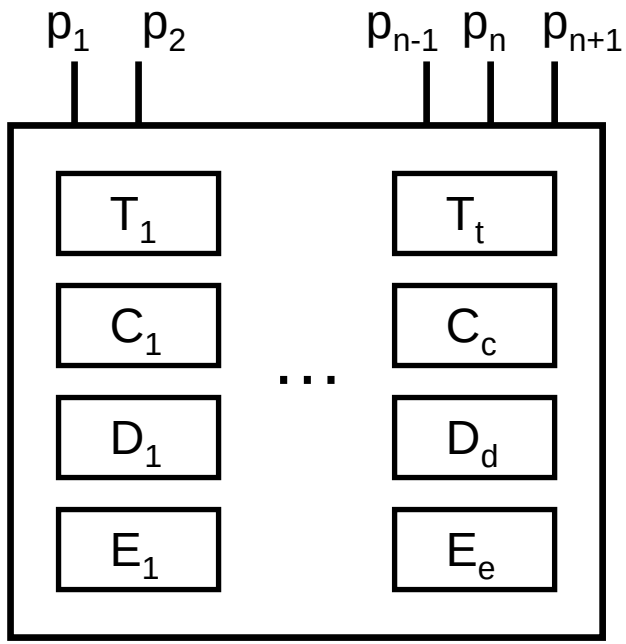
- Shared objects can be combined to implement other shared objects
  - Which SO's can/cannot implement which other SO's?
- Map a SO to a reduction-based hierarchy
  - If *T* has level *n*, then any *n*-process SO can be implemented by *T*'s + registers (wait-freedom)
  - SO at low levels cannot implement SO at higher levels (robustness)
- Herlihy's [Her91] hierarchy $\mathbf{h_m}$ maps SO *T* to level *n* if *T*'s + registers can implement cons(p_1,…,p_n)
  - By universality, *T*'s + registers can implement any n-process SO, so $\mathbf{h_m}$ is wait-free
  - Is $\mathbf{h_m}$ robust?

Robustness of $\mathbf{h_m}$

- $\mathbf{h_m}$ is robust iff $\mathbf{h_m}(\{T,T'\}) = \max(\mathbf{h_m}(T),\mathbf{h_m}(T'))$
- Question is open for general T, T' (ca. 1994)
- **Theorem 1** $\mathbf{h_m}(\{T,cons_n\}) = \max(\mathbf{h_m}(T), n)$
  - If T can "boost" n-consensus to $(n+1)$-consensus, then T can solve $(n+1)$-consensus
- **Lemma** If $\{T,$ reg, $cons_n\}$ can implement $cons_{n+1}$, then $\{T,$ reg, $cons_{n-1}\}$ can implement $cons_n$
  - So $\{T,$ reg, $cons_{n-2}\}$ can implement $cons_{n-1}$, etc.
  - So $\{T,$ reg, $cons_{n-1}\}$ can implement $cons_{n+1}$, etc.
- So if $\{T,$ reg, $cons_n\}$ can implement $cons_{n+1}$, then $\{T,$ reg$\}$ can implement $cons_{n+1}$, so Thm. 1 follows

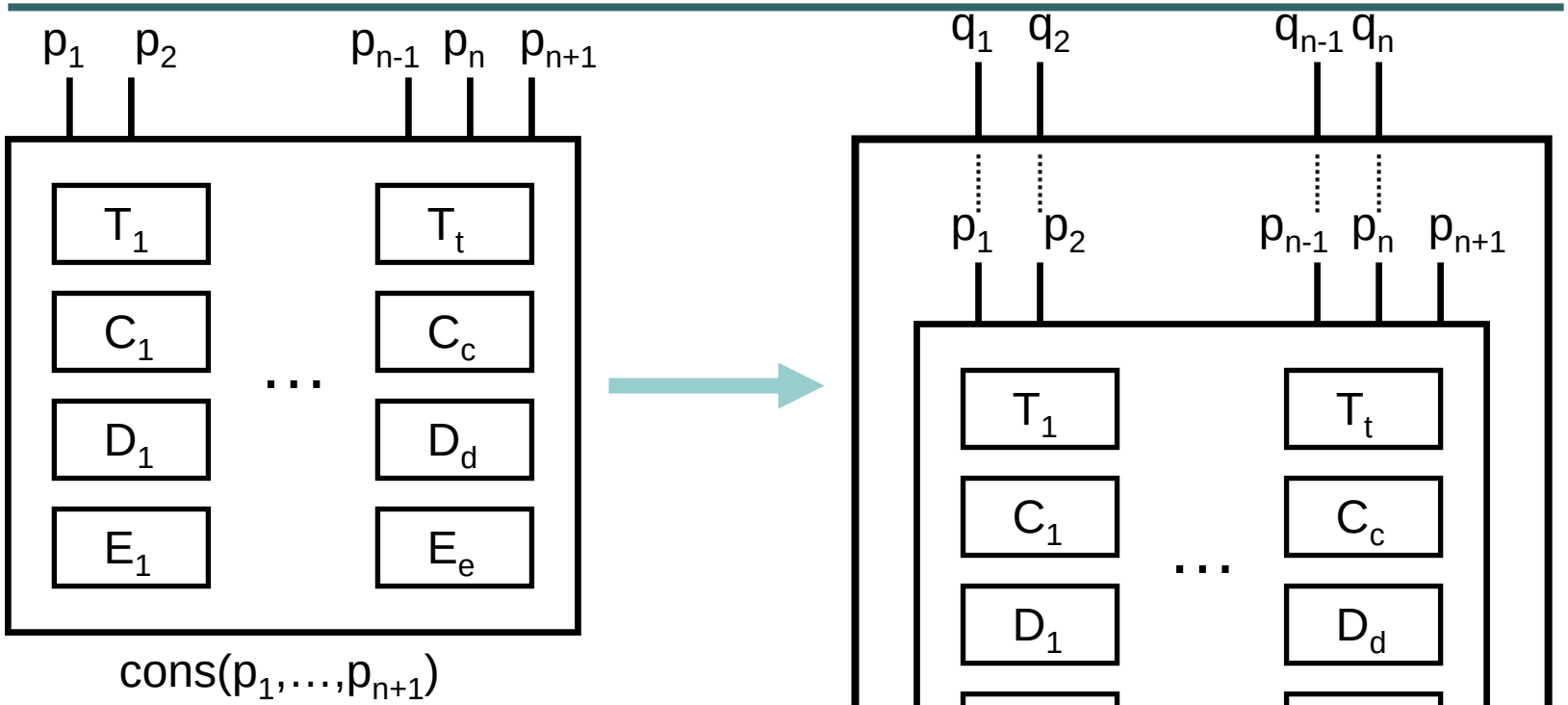
Proof of Lemma (1)

- T_i of type T
- C_i, D_i, E_i are *n*-consensus objects
  - D_i accessed by $p_1,\ldots,p_n$
  - E_i accessed by $p_1,\ldots,p_{n-1},p_{n+1}$
  - C_i accessed by the rest, i.e. n-2 processes among $p_1,\ldots,p_{n-1}$, plus p_n, p_{n+1}
  - Ex. n=4
    - Consensus for p_1, p_2, p_4, p_5 uses C_i
    - Consensus for p_1, p_2, p_3, p_4 uses D_i
    - Consensus for p_1, p_2, p_3, p_5 uses E_i



Cons($p_1,\ldots,p_{n+1}$)

Proof of Lemma (2)



- But cons(q_1,…,q_n) only allowed (*n*-1)-consensus
- Replace C_i,D_i,E_i by (*n*-1)-cons objects

Proof of Lemma (3)

- n+1 process consensus object GroupSolo(p_1,…,p_n,p)
- O_1, O_2 = cons(p_1,…,p_n)

```
1 GP ← propose(O1, vi)
2 if SP = ⊥ then
3   votei ← GP
4 else
5    votei ← SP
6 DEC ← propose(O2,votei)
7 return DEC
```

code for p_i

```
1' SP ← v
2' if GP = ⊥ then
3'   DEC ← u
4' else
5'    loop until DEC ≠ ⊥
6' return DEC
```

code for p

Proof of lemma (4)

- **Claim** GroupSolo($p_1,\ldots,p_n,p$) solves (n+1)-consensus. $p_1,\ldots,p_n$ never block, but p may block
- If 1' before 1, then
  - $vote_i$ = v, by line 5
  - DEC = v, by lines 6, 3'
- If 1' before 1, then
  - $p_1,\ldots,p_n$ agree, by line 6
  - p agrees with $p_1,\ldots,p_n$, by line 6'
- If GP $\neq \perp$, DEC = $\perp$, and $p_1,\ldots,p_n$ fail, then p blocks

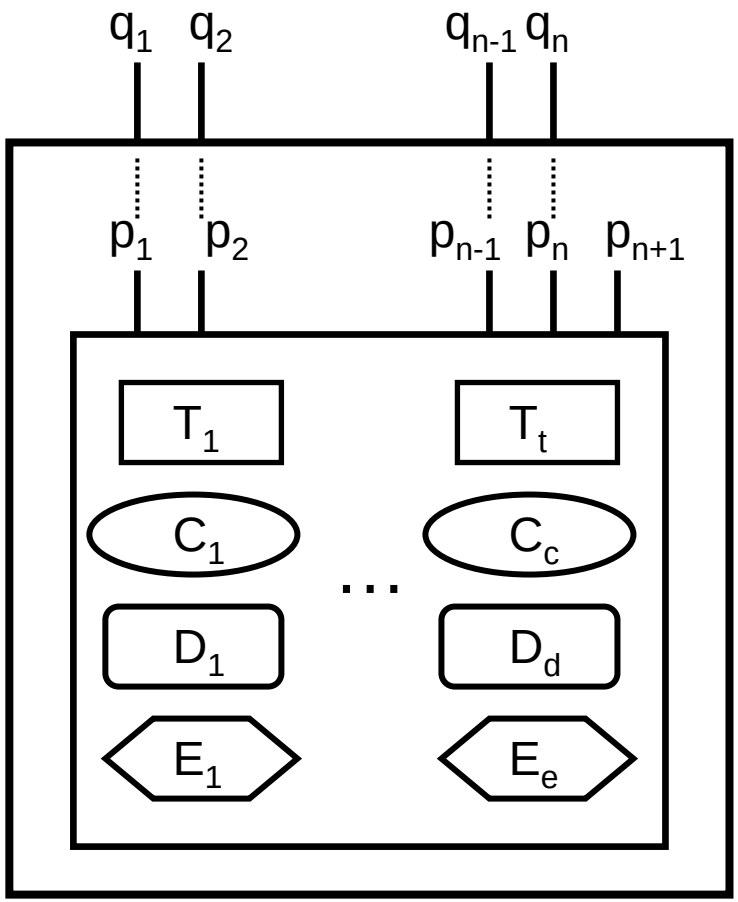
Proof of lemma (5)

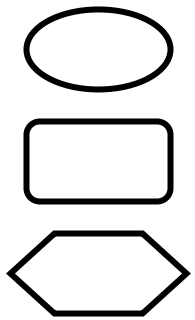
- *n*+1 process consensus object NonConcurrent(p_1,…,p_{n-1},p,p’)
- O = cons(p_1,…,p_{n-1},p), O’ = cons(p_1,…,p_{n-1},p’)

1 $u_i \leftarrow$ propose(O, v_i)	1’ if DEC = $\perp$ then	1’’ if DEC = $\perp$ then
2 DEC $\leftarrow$ propose(O’,u_i)	2’ DEC $\leftarrow$ propose(O,v)	2’’ DEC $\leftarrow$ propose(O’,v’)
3 return DEC	3’ return DEC	3’’ return DEC
code for p_i	*code for p*	*code for p’*

- **Claim** NonConcurrent(p_1,…,p_{n-1},p,p’) works correctly if p, p’ are not concurrent

Proof of lemma (6)





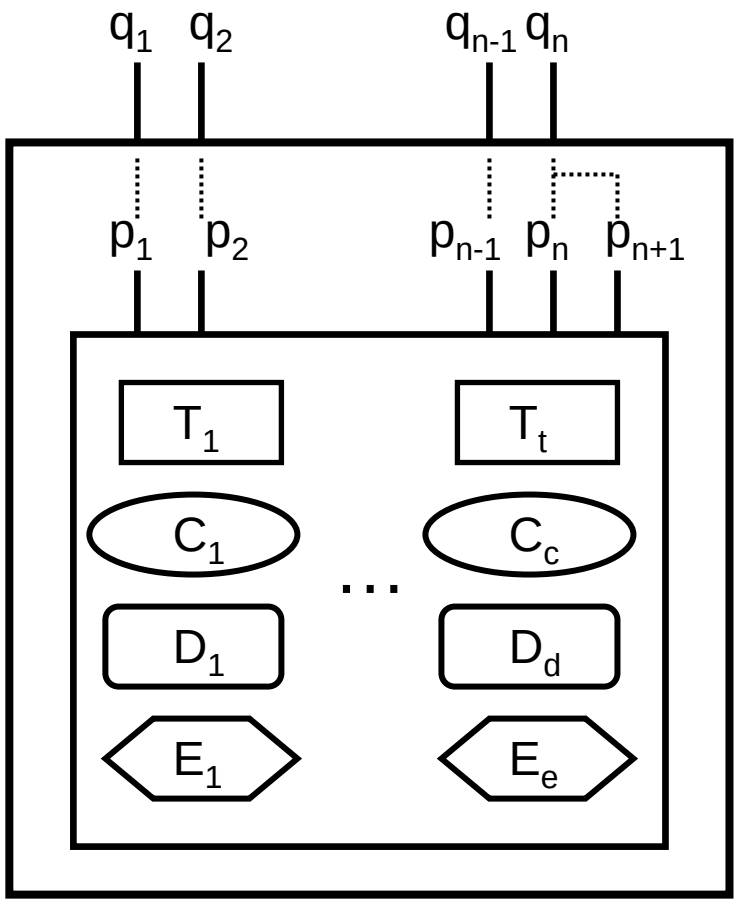
NonConcurrent($p_{i1},\ldots p_{in-2},p_n,p_{n+1}$)

GroupSolo($p_1,\ldots,p_{n-1},p_n$)

GroupSolo($p_1,\ldots,p_{n-1},p_{n+1}$)

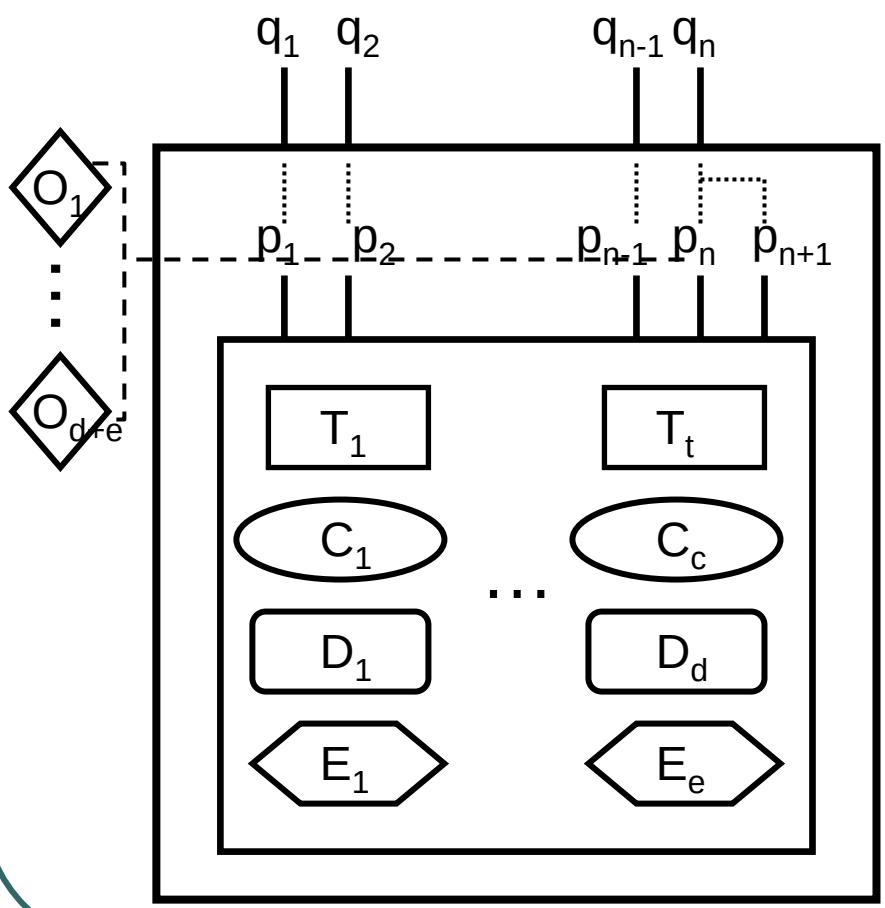
- This only uses (*n*-1)-consensus objects
- $q_1,\ldots,q_{n-1}$ never block
- q_n simulates p_n, can block if GroupSolo($p_1,\ldots,p_{n-1},p_n$) blocks

Proof of Lemma (7)



- If p_n blocks, then q_n simulates p_{n+1}
- If p_{n+1} blocks, then q_n simulates p_n
- But p_n,p_{n+1} can both block
- p_n only blocks when $p_1,\ldots,p_{n-1}$ set GP$\neq\perp$, DEC=$\perp$, for some D_i or E_i, then all fail; similarly for p_{n+1}
- If we force $p_1,\ldots,p_{n-1}$ to access only one D_i or E_i at a time, then they can only block p_n or p_{n+1}, but not both
- Create consensus objects $O_1,\ldots,O_{d+e}$, accessible only by $p_1,\ldots,p_n$

Proof of Lemma (8)



- To propose v object F (F = D_i or E_i), p_i proposes F to O_1
  - p_i asks O_1 permission to use F
- If response is F
  - p_i is the first process to access F
  - p_i proposes v to F
- If response is G≠ F
  - Another process p_j accessed F first
  - p_i still proposes v to F
    - If p_j set GP≠⊥, DEC=⊥ at F then failed, p_i sets DEC≠⊥ at F
    - p_i "helps" p_j
  - Then p_i proposes F to O_2,O_3, etc., until O_j returns F
    - So only one object with GP≠⊥, DEC=⊥
- Note that each time p_1,…,p_{n-1} uses D_i or E_i, we need a new O_i'

Proof of Lemma (9)

- We're done!
  - Final construction uses only (*n*-1)-consensus objects
  - $p_1,\ldots,p_{n-1}$ never block, by property of NonConcurrent and Group Solo
    - So $q_1,\ldots,q_{n-1}$ never block
  - Since only one object with GP$\neq\perp$, DEC=$\perp$, only one of p_n,p_{n+1} can block
    - So q_n never blocks
  - Agreement follows by correctness of (n+1)-consensus among the p_i's

Wait-freedom vs. *t*-resiliency

- Is *t*-resilient consensus easier for n+1 processes, than for n processes?
  - For n+1 processes, the fraction of failed processes is lower than for n processes
- **Theorem 2** For any $t > 1$, type T, $cons(p_1,\ldots,p_n)$ has a *t*-resilient implementation iff $cons(q_1,\ldots,q_{t+1})$ has a *t*-resilient (wait-free) implementation
  - Only the number of failed processes matters, not the fraction

Proof of Theorem 2 (1)

- cons(q_1,…,q_{t+1}) has a *t*-resilient impl $\Rightarrow$ cons(p_1,…,p_n) has a *t*-resilient impl
- p_i simulates q_i, for $1 \leq i \leq t+1$
- If at most *t* p_i's fail, then at most *t* simulated q_i's
  - cons(q_1,…,q_{t+1}) terminates correctly
  - So does cons(p_1,…,p_n)
  - p_i, $i \geq t+1$, just copies p_1's decision value

Proof of Theorem 2 (2)

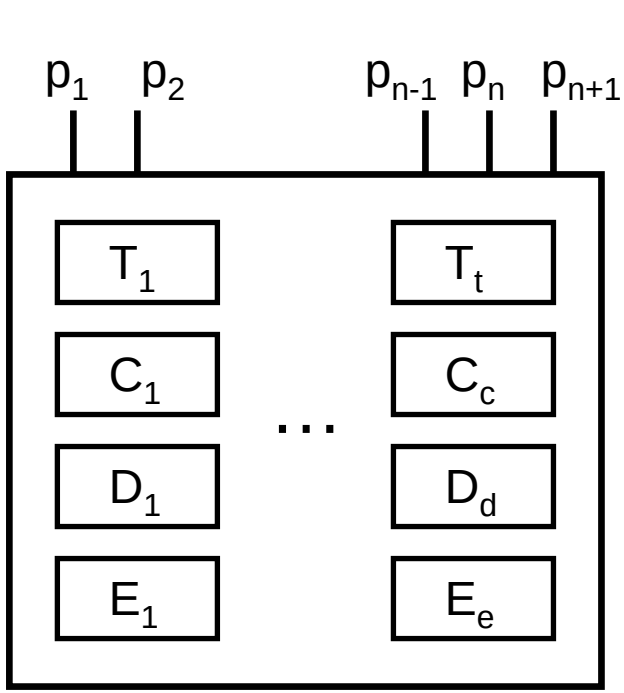
- cons(p_1,…,p_n) has a *t*-resilient impl $\Rightarrow$ cons(q_1,…,q_{t+1}) has a *t*-resilient impl
- q_i's simulate the p_j's, using T + test&set
  - test&set(q_1,…,q_{t+1}) can be implemented from {cons(q_1,q_2), reg} [AGMT92]
  - If cons(q_1,…,q_{t+1}) has a wait-free impl using {T, cons(q_1,q_2), reg}, then by Thm. 1, cons(q_1,…,q_{t+1}) has a wait-free impl using {T, reg}
- Want that if at most *t* q_i's fail, at most *t* simulated p_i's fail
  - Then cons(p_1,…,p_n) terminates correctly, and so does cons(q_1,…,q_{t+1})

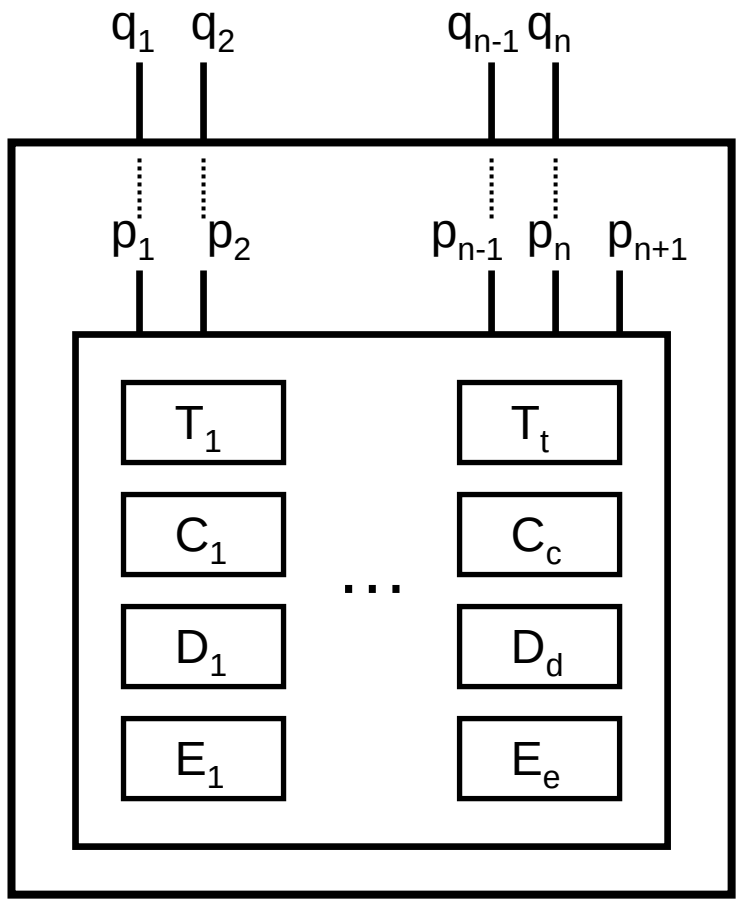
Proof of Theorem 2 (3)

- Each q_i tries to simulate the p_j's in round-robin order
- To simulate a step of p_j, q_i must obtain S_j, the j'th test&set
  - If q_i succeeds
    - q_i observes p_j's state
    - executes instruction of p_j's program counter
    - updates p_j's state and program counter
    - resets S_j
  - If q_j fails, it tries to simulate q_{j+1}

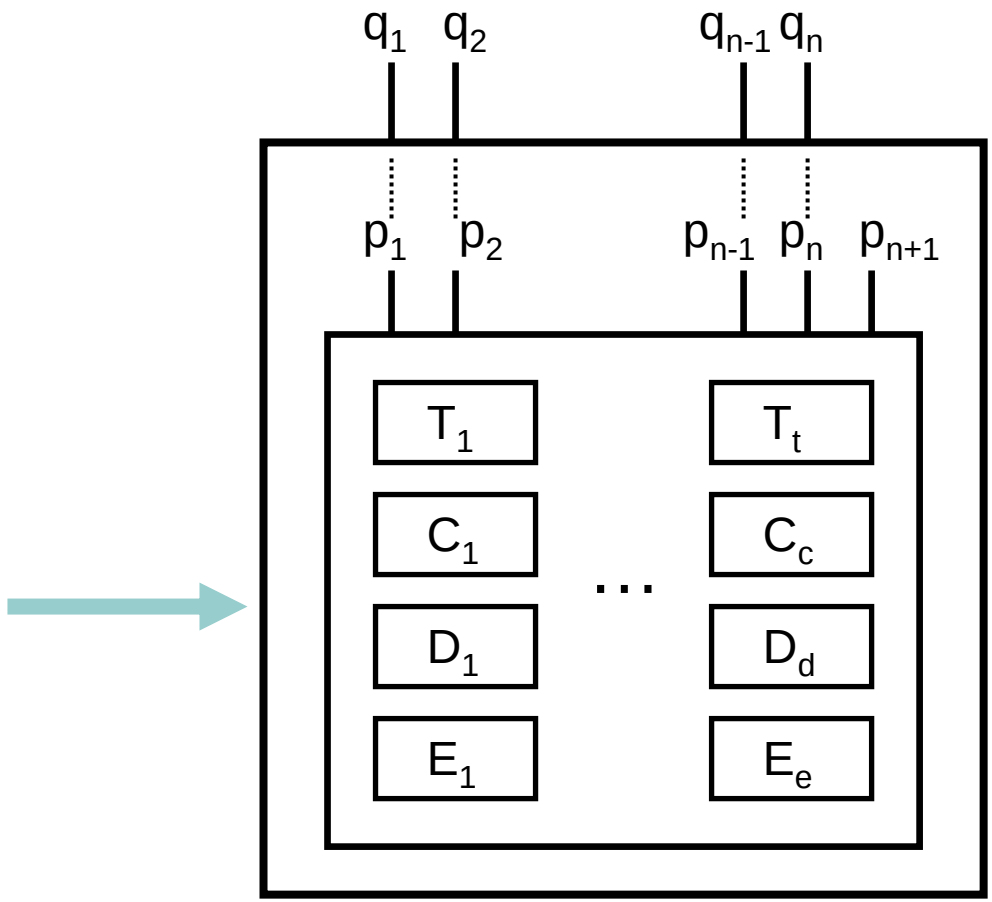
Proof of Theorem 2 (4)

- If q_i fails while simulating p_j, then no other $q_{i'}$ can simulate p_j
  - But any other $p_{j'}$ not affected
  - One q failure causes at most one simulated p failure
  - At most *t* q failures causes at most *t* simulated p failures
  - The simulation is correct


p1
p2
pn-1
pn
pn+1
T1
Tt
C1
Cc
D1
Dd
E1
Ee


q1
q2
qn-1
qn
p1
p2
pn-1
pn
pn+1
T1
Tt
C1
Cc
D1
Dd
E1
Ee

- asdf



- asdf

```
1 GP ← propose(O1, vi)
2 if SP = ⊥ then
3   votei ← GP
4 else
5    votei ← SP
6 dec ← propose(O2,votei)
7 return dec
```

```
1 SP ← v
2 if GP = ⊥ then
3   dec ← u
4 else
5    loop until dec ≠ ⊥
6 return dec
```