TDC1:
Universal construction

So far…

- 2-process consensus cannot be solved using registers
- N-process consensus can be solved using registers and Ω
- 2-process consensus can be solved using registers and T&S or queues
  - ✓but not 3-process consensus!

Why consensus is interesting?

Because it is *universal* !

- If we can solve consensus among N processes, then we can *implement* every object shared by N processes

- A key to implement a generic fault-tolerant service (replicated state machine)

Today's lecture

- Herlihy's universal construction
- Consensus numbers
- From shared memory to message passing and back
  - ✓ In the presence of malicious adversary

What is an *object* ?

Object O is defined by the tuple (Q,O,R,σ):

- Set of states Q
- Set of operations O
- Set of outputs R
- Sequential specification σ, a subset of OxQxRxQ:
  - ✓(o,q,r,q') is in σ ⇔ if operation o is applied to an object in state q, then the object *can* return r and change its state to q'

Deterministic objects

- An operation applied to a *deterministic* object results in exactly one (output,state) in RxQ, i.e., σ can be seen a function OxQ -> RxQ

- E.g., queues, counters, T&S are deterministic
- Unordered set (put/get) – non-deterministic

Example: queue

Let V be the set of possible elements of the queue

$Q=V^*$

$O=\{enq(v)_{v \text{ in } V}, deq\}$

$R=V \cup \{empty\} \cup \{ok\}$

$\sigma(enq(v),q)=(ok,q.v)$

$\sigma(deq(),q.v)=(v,q)$

$\sigma(deq(),empty)=(empty,empty)$

Implementation: definition

A distributed algorithm A that, for each operation o in S and for every pi, describes the corresponding sequence of steps on the base objects

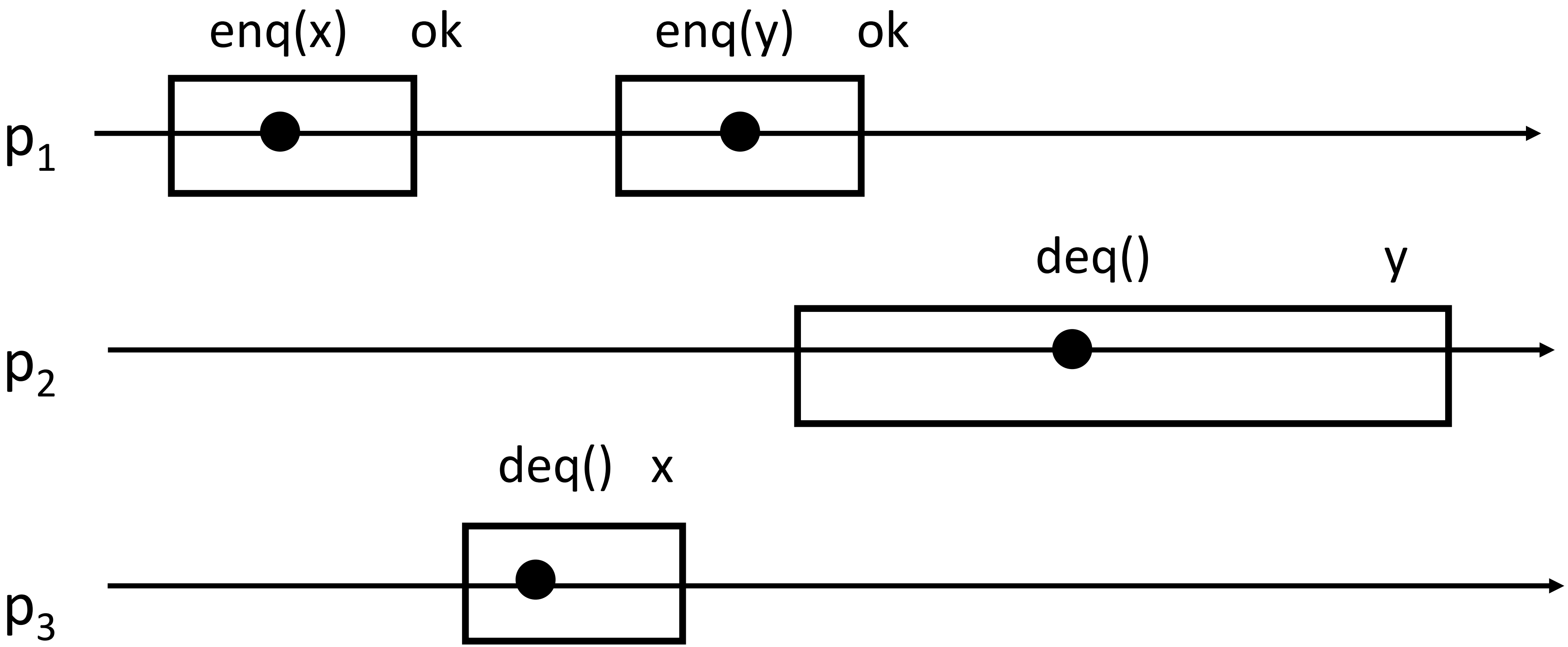
A run of A is *well-formed* if no process invokes a new operation on the implemented object before returning from the old one

Implementation: correctness

A (wait-free) implementation A is correct if in every well-formed run of A

- Wait-freedom: every operation run by p_i returns in a finite number of steps of p_i
- Linearizability ≈ operations "appear" instantaneous (the corresponding *history* is *linearizable*)

Linearization



p_1-enq(x); p_1-ok; p_3-deq(); p_3-x;
p_1-enq(y); p_1 –ok; p_2-dequeue(); p_2-y

Universal construction

Theorem 1 [Herlihy, 1991] If N processes can solve consensus, then N processes can (wait-free) implement every object $O=(Q,O,R,\sigma)$

A moment of meditation

Suppose you are given an unbounded number of consensus objects and atomic read-write registers

You want to implement an object $O=(Q,O,R,\sigma)$

How would you do it?

Universal construction: idea

Every process that has a pending operation does the following:

- Publish the corresponding *request*
- Collect published requests and use consensus instances to serialize them: the processes agree on the order in which the requests are executed

Universal construction: variables

Shared abstractions:

N atomic registers R[0,…,N-1], initially Ø

N-process consensus instances C[1], C[2], …

Local variables for each process p_i:

integer *seq*, initially 0

// the number of executed requests of p_i

integer *k*, initially 0

// the number of batches of

// executed requests

sequence *linearized*, initially empty

//the sequence of executed requests

Universal construction: algorithm

Code for each process p_i:

Implementation of operation op

```
seq++
R[i] := (op,i,seq)                        // publish the request
repeat
        V := read R[0,…,N-1]              // collect all requests
        requests := V-{linearized}        //choose not yet linearized requests
        if requests≠Ø then
            k++
            decided:=C[k].propose(req)
            linearized := linearized.decided
            //append decided request in some deterministic order
until (op,i,seq) is in linearized
return the result of (op,i,seq) in linearized
            // using the sequential specification σ
```

Universal construction: correctness

- Linearization of a given run: the order in which operations are put in the *linearized list*
  - ✓ well-defined: all *linearized* lists are related by containment
  - ✓ Can it violate the temporal order?
  - ✓ In every finite run, the longest *linearized* list consists of all complete operations and a subset of incomplete ones

Universal construction: correctness

- Wait-freedom:
  - ✓Termination and validity of consensus: there exists k such that the request of p_i gets into *req* list of every processes that runs C[k].*propose(req)*

- Linearizability: if op1 precedes op2, then op2 cannot be linearized before op1
  - ✓Validity of consensus: a value cannot be decided unless it was previously proposed

Another universal abstraction: CAS

Compare&Swap (CAS) stores a *value* and exports operation CAS(u,v) such that:

- If the current value is u, CAS(u,v) replaces it with v and returns u
- Otherwise, CAS(u,v) returns the current value

A variation: CAS returns a boolean (whether the replacement took place) and an additional operation read() returns the value

N-process consensus with CAS

Shared objects:

CAS CS initialized Ø

// Ø cannot be an input value

Code for each process p_i ($i=0,\ldots,N-1$):

v_i := input value of p_i

v :=CS.CAS(Ø,v_i)

if v = Ø

return v_i

else

return v

M-consensus object

M-consensus stores a value in {Ø} U V and exports operation propose(v), v in V:

For 1st to Mth propose() operations:

- If the value is Ø, then propose(v) sets the value to v and returns v
- Otherwise, returns the value

All other operations do not change the value and return Ø

M-process consensus with M-consensus

Immediate: every process pi simply invokes C.propose(input of pi) and returns the result of it

(M+1)-consensus using M-consensus?

Impossible: M+1-th process is ignorant

Consensus number

An object O has consensus number k (we write cons(O)=k) if

- k processes can solve consensus using registers and any number of copies of O
- but k+1 processes cannot

If no such number k exists for O, then cons(O)=∞

(k=cons(O) is the maximal number of processes that can be perfectly synchronized using copies of O and registers)

Consensus numbers

- cons(register)=1
- cons(T&S)=cons(queue)=2
- …
- cons(N-consensus)=N
  - ✓N-consensus is N-universal!
- …
- cons(CAS)=∞

Open questions

- Robustness

  Suppose we have two objects A and B, cons(A)=cons(B)=k

  Can we solve (k+1)-consensus using registers and copies of A and B?

- Can we implement an object of consensus power k shared by N processes (N>k) using k-consensus objects?

- What about message passing?
- What about malicious (Byzantine) processes?

Message-passing

- Which results for shared memory can be translated into message-passing models?

- Consider a network where every two processes are connected via a reliable channel
  - ✓ no losses, no creation, no duplication

Implementing message-passing

Theorem 1 A reliable message-passing channel between two processes can be implemented using two 1W1R registers

Corollary 1 Consensus is impossible to solve in an asynchronous message-passing system if at least one process may crash

Implementing shared memory

Theorem 2 A one-writer N-reader regular register can be implemented in a (reliable) message-passing model where a majority of processes are correct

Corollary 2 N-process consensus can be solved in a message-passing where a majority of processes are correct using Ω

Implementing a 1W1R register

```
Upon write(v)
   t++
   send [v,t] to all
   wait until received [ack,t] from a majority
   return ok

Upon read()
   r++
   send [?,r] to all
   wait until received {(t',v',r)} from a majority
   return v' with the highest t'
```

Implementing a 1W1R register, contd.

Upon receive [v,t]

if $t>t_i$ then

$v_i := v$

$t_i := t$

send [ack,t] to the writer

Upon receive [?,r]

send [v_i,t_i,r] to the reader

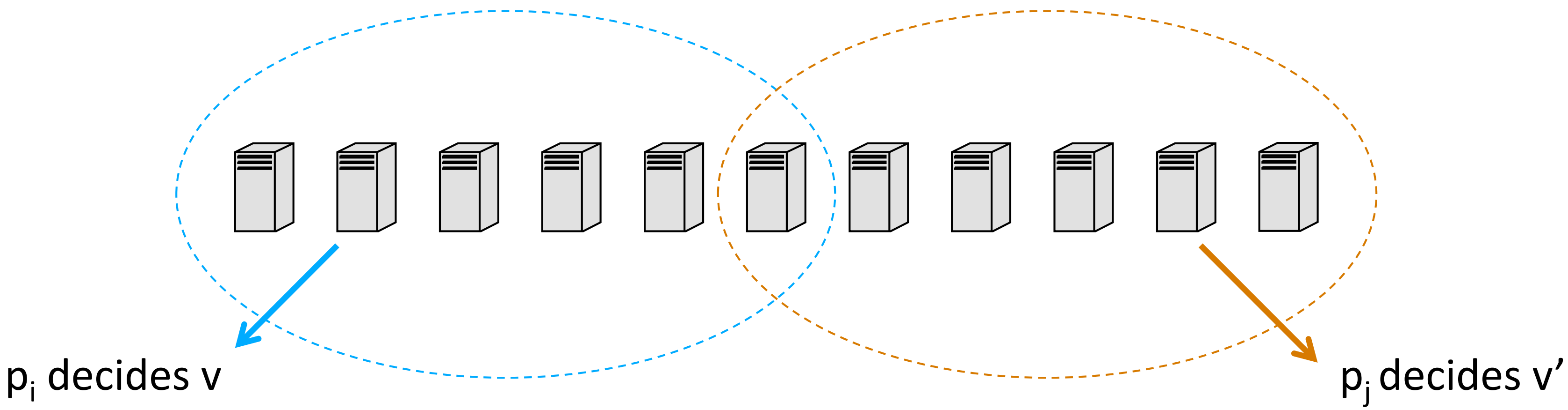
Is the majority assumption crucial?

- The reader can miss the value if a preceding write
- Suppose there is a message-passing N-process consensus algorithm using Ω that tolerates f≤N/2 failures
  - ✓Different values can be decided

A majority must be correct

Any two decisions must involve at least one process in common (*decision quorums* must intersect)

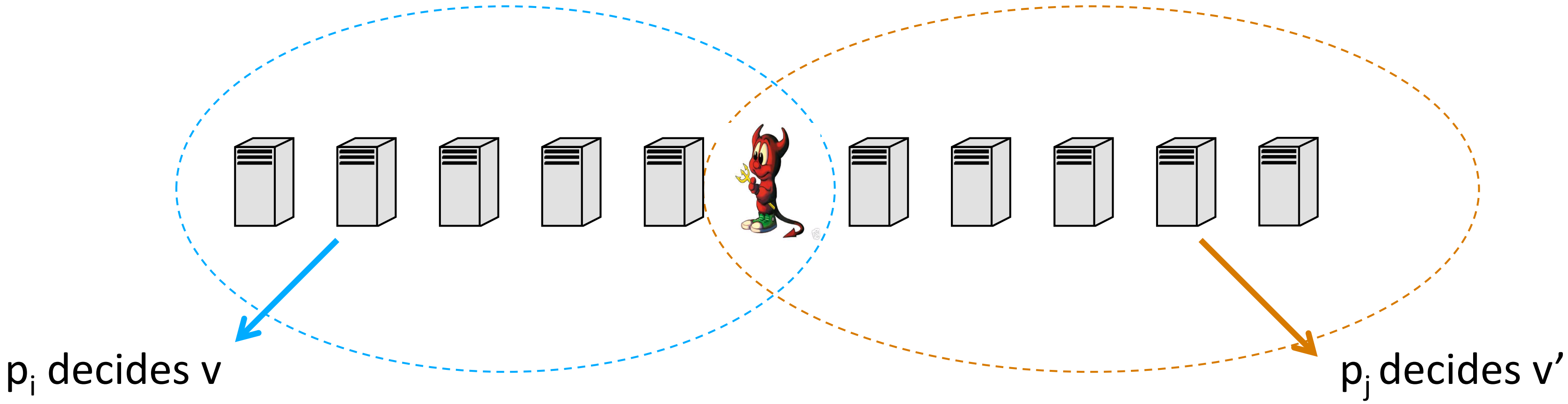
If f failure are tolerated, then the (worst-case) decision quorums of size N-f intersect in at least N-2f processes => f<N/2



What if processes are malicious?

Any two decisions must involve at least one correct process in common

Otherwise quorums may miss each other (a Byzantine process plays oblivious)



More than two third must be correct!

If f failure are tolerated, then the decision quorums of size N-f intersect in at least N-2f processes

$$\Rightarrow N-2f>f+1$$

$$\Rightarrow f< N/3$$

There is more to this

- Renaming and adaptive algorithms
- Sub-consensus problems
- Non-uniform computing models
- Transactional memory
- Failure detection

Check http://www.net.t-labs.tu-berlin.de/~petr/ for more information