Shared Objects

Shared Objects

- Invoked operations have a non-zero duration
  - Invocations can overlap
- Useful for:
  - modeling distributed shared memory
  - Objects can be combined together to implement higher level objects
- The problem: Specify object behavior under concurrent access

Sequential vs Concurrent

- Type: same as before
  - Function f is called *sequential specification*
- Sequential:
  - Object has meaningful state between invocations
- Concurrent
  - Because invocations overlap, object might **never** be between invocations

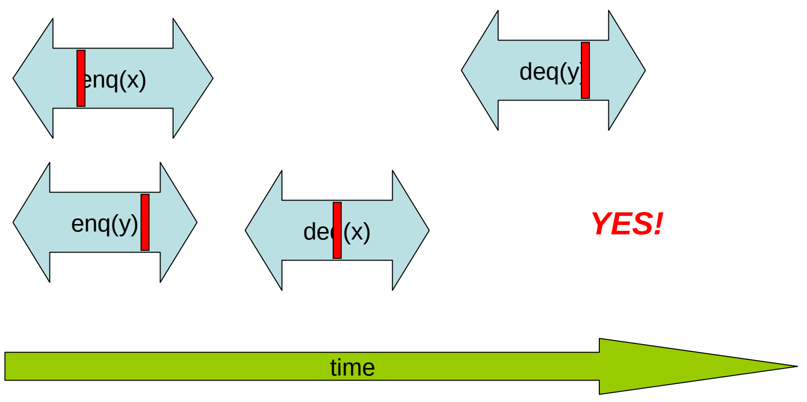
Exercise: Define a shared Consensus object
Exercise: Define a shared Non-Blocking Atomic Commit (NBAC) object or argue that no such meaningful object can be defined

Atomicity

- A safety property of concurrent objects
- A sequence α of invocations and responses of an object O satisfies atomicity if
  - Every complete (and some incomplete) operation in α appears to take effect in a point between the invocation and the response

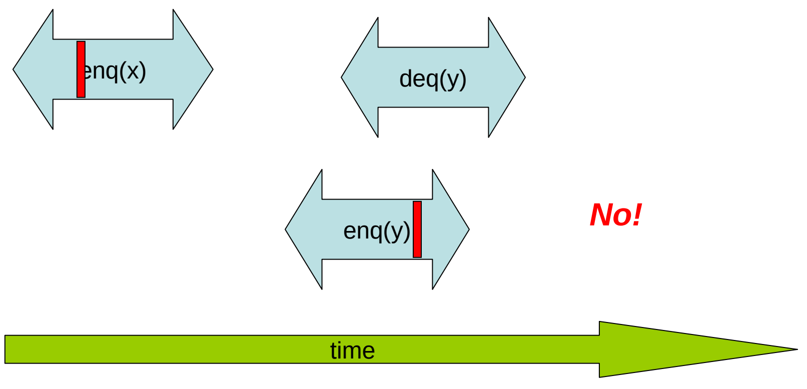
Shared FIFO Queue

Atomic or not?



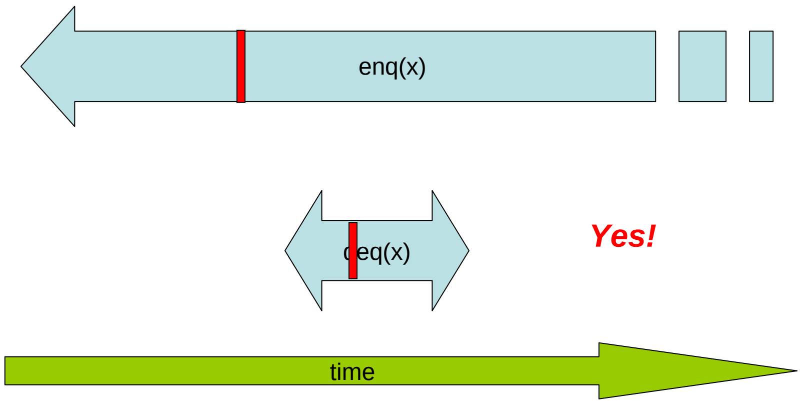
Shared FIFO Queue

Atomic or not?



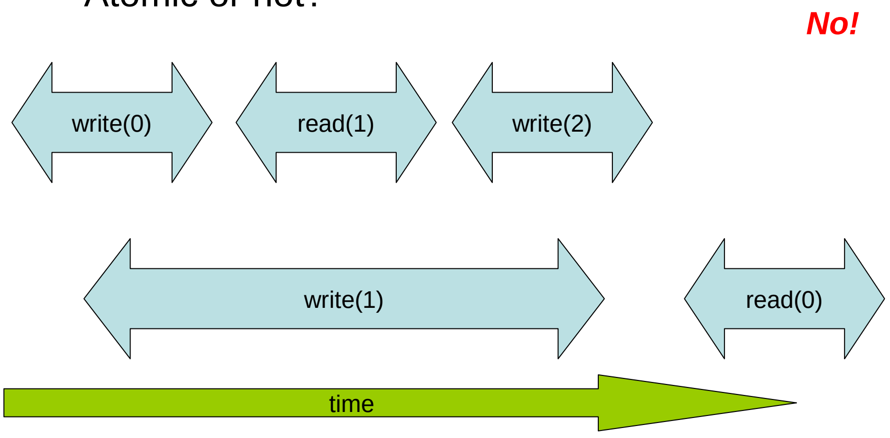
Shared FIFO Queue

Atomic or not?



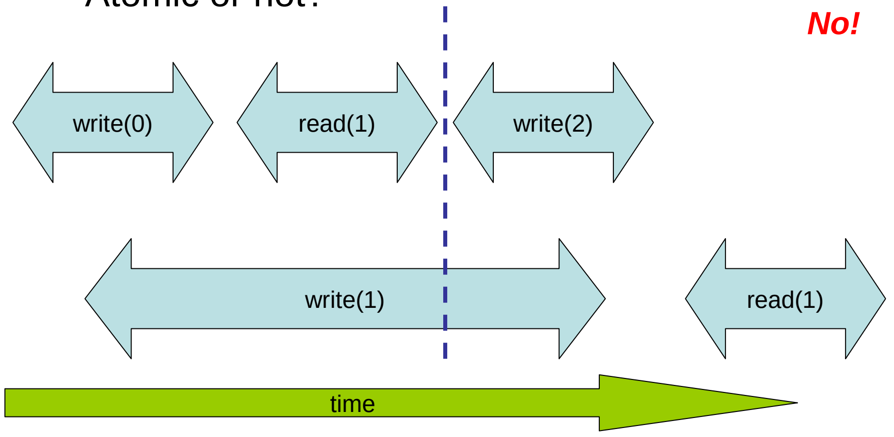
Shared R/W register

Atomic or not?



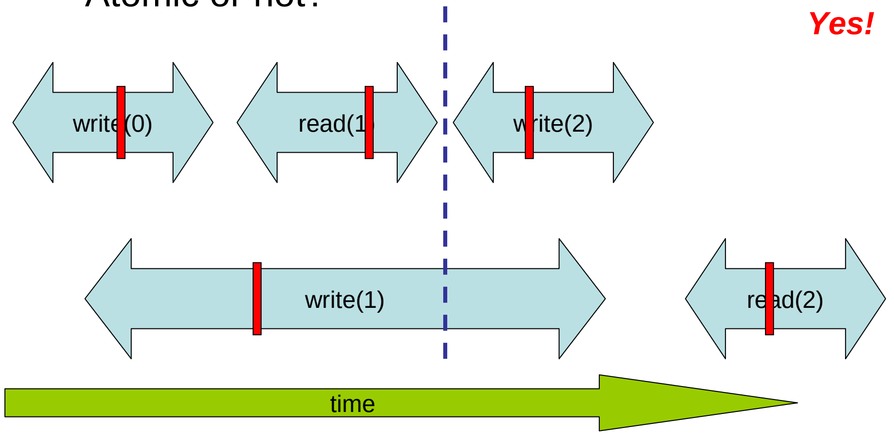
Shared R/W register

Atomic or not?



Shared R/W register

Atomic or not?



Power of Atomic Objects

- Consensus # of atomic registers is 1
  - Weaker than atomic variables
- Consensus # of FIFO Queue is 2
  - Stack, Set, priority queue, double-ended queue

Implementability

- Is FIFO queue implementable from R/W registers?
  - No. Otherwise, we can solve wait-free Consensus for 2 processes using registers
- Are CAS, TS, FAA, etc., are implementable using R/W registers?
  - No. Otherwise can solve wait-free Consensus for $n \geq 2$ processes using registers

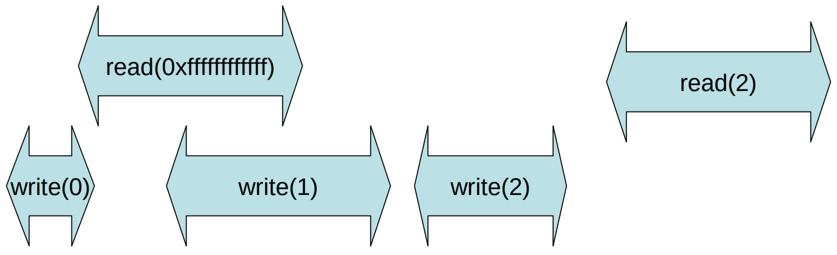
Universality of Consensus

- For any object O with a deterministic sequential specification, there exists a wait-free implementation of O from a collection of atomic Consensus objects and R/W registers

Comment: What about a simple algorithm for that?

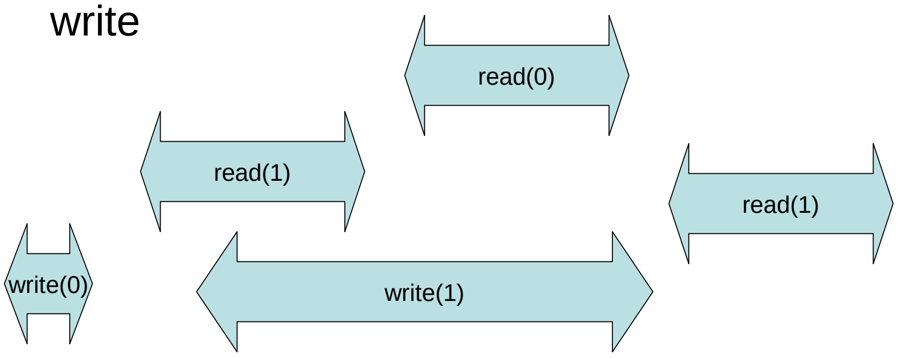
More Safety Properties for R/W registers

- Single-Writer/Multi-Reader registers
- Safe register:
  - Each read that does not overlap a write returns the value of the latest non-overlapping write
  - The result is undefined for a read overlapping a write (may be any value in the register domain)



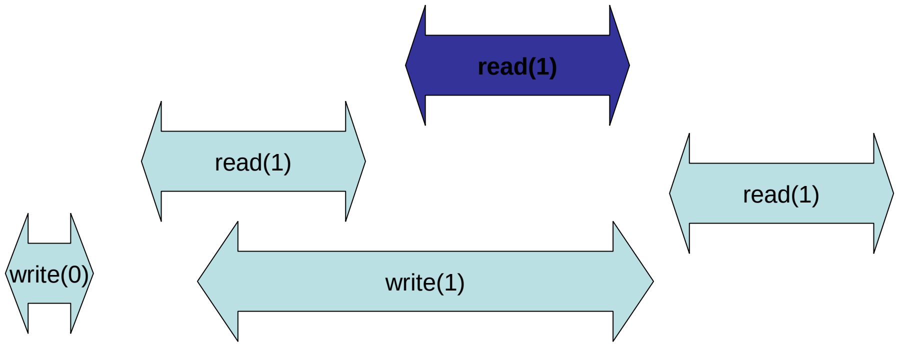
Regular Register

- Each read that does not overlap a write returns the value that is not older than the value written by the latest non-overlapping write

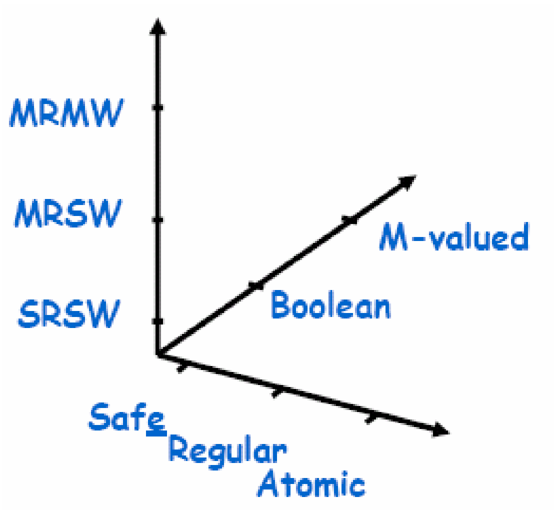


Comment: It might be interesting to talk about multi-writer regular registers

Atomic Register



Register Space



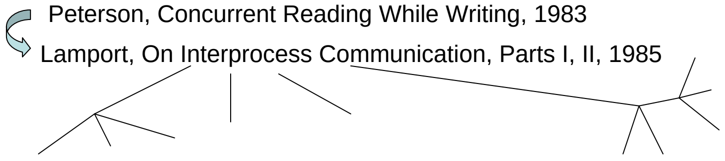
Picture © Maurice Herlihy and Nir Shavit

Computability

- The weakest possible register is SRSW Boolean safe register
- Wait-free MRMW atomic registers are implementable from SRSW Boolean safe registers!
  - Involves a long, hard to follow reduction chain spanning about 30 papers

Peterson, Concurrent Reading While Writing, 1983

Lamport, On Interprocess Communication, Parts I, II, 1985



Some constructions

- From SRSW safe bit to SRSW regular bit:

```
write(v):
    if (old != v) then
        write(val, v);
        old := v;

read():
    return val;
```

Some Constructions

- SRSW m-valued regular from regular bits
- An array x[m] of regular bits

```
write(v):
   write(x[v], 1);
   for (i=v-1; i >=1; i--)
       write(x[i],0);
read()
   for (i=1; i<= m; i++)
       if (read(x[i]) != 0)
           return i;
```

Constructions Summary

- It is possible to construct SRSW m-valued regular register from SRSW Boolean safe registers
  - etc…
- The second construction requires Θ(m) space
- There are many Θ(log(m)) constructions

Termination Conditions

- Wait Freedom: Each operation by a correct process is complete
- Lock Freedom: Each operation by a correct process is complete unless infinitely many operations are complete
- Obstruction Freedom: Each operation by a correct process is complete unless it is concurrent to an operation by a correct process

Comment: resilience and different termination conditions are actually orthogonal

Termination Conditions

- Wait-free and lock-free Consensus are both impossible with registers
- Obstruction-free Consensus is possible with registers!
  - Paxos
- Objects can be implemented simpler and more efficient with obstruction freedom
  - Use a separate “contention manager” to resolve contention

Exercise: Prove that lock-free Consensus is impossible with registers
Exercise: Give an obstruction-free atomic snapshot algorithm