The Limitations of Registers

R. Guerraoui
Distributed Programming Laboratory
Registers

• **Question 1:** what objects can we implement with registers? **Counters** and **snapshots** (previous lecture)

• **Question 2:** what objects we cannot implement? (this lecture)
Shared memory model
Shared memory model

- Registers
- Snapshots
- Counters

P1, P2, P3
Shared memory model

Counters  Registers  Fetch&Inc?

Snapshots  Queue?

P1  P2  P3
Fetch&Inc

- A counter that contains an integer

- Operation `fetch&inc()` increments the counter and returns the new value
The consensus object

- One operation \textit{propose()} which returns a value. When a propose operation returns, we say that the process decides

- No two processes decide differently

- Every decided value is a proposed value
The consensus object

- **Proposition:**
  - **Consensus** can be implemented among two processes with *Fetch&Inc* and *registers*

- Proof (algorithm): consider two processes $p_0$ and $p_1$ and two *registers* $R_0$ and $R_1$ and a *Fetch&Inc* $C$. 
2-Consensus with Fetch&Inc

- Uses two registers R0 and R1, and a Fetch&Inc object C (with one fetch&inc() operation that returns its value)
- (NB. The value in C is initialized to 0)

- Process pi:
  - propose(vl)
  - R1.write(vl)
  - val := C.fetch&inc()
  - if(val = 1) then
    - return(vl)
  - else return(R{1-I}.read())
Impossibility [FLP85,LA87]

- **Proposition:** there is no asynchronous deterministic algorithm that implements consensus among two processes using only registers

- **Corollary:** there is no algorithm that implements *Fetch&Inc* among two processes using only registers
Queue

- The queue is an object container with two operations: \texttt{enq()} and \texttt{deq()}

- Can we implement a (atomic wait-free) \textit{queue}?
2-Consensus with queues

Uses two registers R0 and R1, and a queue Q
Q is initialized to \{winner, loser\}

Process pI:

```plaintext
propose(vI)
  R{1-I}.write(vI)
  item := Q.dequeue()
  if item = winner return(vI)
  return(R{1-I}.read())
```
\[
\begin{align*}
&P0 \quad W(0) \quad \text{Deq()} \rightarrow \text{winner} \quad \text{Return}(0) \\
&\quad \quad \quad \quad \quad R0 \quad Q \\
&P1 \quad W(1) \quad \text{Deq()} \rightarrow \text{loser} \quad \text{Return}(0) \\
&\quad \quad \quad \quad \quad R1 \quad Q
\end{align*}
\]
Correctness

Proof (algorithm):

- (wait-freedom) by the assumption of a wait-free register and a wait-free queue plus the fact that the algorithm does not contain any wait statement
- (validity) If \( p_I \) dequeues winner, it decides on its own proposed value. If \( p_I \) dequeues loser, then the other process \( p_J \) dequeued winner before. By the algorithm, \( p_J \) has previously written its input value in \( RJ \). Thus, \( p_I \) decides on \( p_J \)’s proposed value;
- (agreement) if the two processes decide, they decide on the value written in the same register.
More consensus implementations

- A **Test&Set** object maintains binary values x, init to 0, and y; it provides one operation: `test&set()`
  - Sequential spec:
  - `test&set() {y := x; x := 1; return(y);}`

- A **Compare&Swap** object maintains a value x, init to ⊥, and provides one operation: `compare&swap(v,w);`
  - Sequential spec:
    - `c&s(old,new) {if x = old then x := new; return(x);}`
2-Consensus with Test&Set

- Uses two registers R0 and R1, and a Test&Set object T.

- Process pi:

  - propose(vl)
    - Ri.write(vl)
    - val := T.test&set()
    - if(val = 0) then
      - return(vl)
    - else return(R{1-i}.read())
N-Consensus with C&S

- Uses a C&S object C

- Process pi:

  - propose(vl)
  - val := C.c&s(⊥, vl)
  - if(val = ⊥) then
    - return(vl)
    - else return(val)
Impossibility [FLP85,LA87]

- **Proposition:** there is no *asynchronous deterministic* algorithm that implements *consensus* among two processes using only *registers*

- **Corollary:** there is no algorithm that implements a *queue* (*Fetch&Inc,*...) among two processes using only *registers*
Registers

• **Question 1:** what objects can we implement with registers? *Counters* and *snapshots* (previous lecture)

• **Question 2:** what objects we cannot implement? All objects that (together with *registers*) can implement *consensus* (this lecture)
Impossibility (Proof)

- **Proposition:** there is no algorithm that implements *consensus* among two processes using only *registers*

- Proof (by contradiction): consider two processes $p_0$ and $p_1$ and any number of *registers*, $R_1..R_k$.
  
  Assume that a consensus algorithm $A$ for $p_0$ and $p_1$ exists.
Elements of the model

- A *configuration* is a global state of the distributed system

- A new configuration is obtained by executing a *step* on a previous configuration: the step is the unit of execution
Elements of the model

- The adversary decides which process executes the next step and the algorithm deterministically decides the next configuration based on the current one
What is distributed computing?
A game
A game between an adversary and a set of processes
The adversary decides which process goes next.

The processes take steps.
Elements of the model

- The adversary decides which process executes the next step and the algorithm deterministically decides the next configuration based on the current one.
Elements of the model

- **Schedule**: a sequence of steps represented by process ids
- The schedule is chosen by the system
- An asynchronous system is one with no constraint on the schedules: any sequence of process ids is a schedule
Consensus

- The algorithm must ensure that *agreement* and *validity* are satisfied in every schedule.

- Every process that executes an infinite number of steps eventually decides.
Impossibility (elements)

- (1) a (initial) **configuration** $C$ is a set of (initial) values of $p_0$ and $p_1$ together with the values of the registers: $R_1..R_k$;

- (2) a **step** is an elementary action executed by some process $p_I$: it consists in reading or writing a value in a register and changing $p_I$’s state according to the algorithm $A$;

- (3) a **schedule** $S$ is a sequence of steps; $S(C)$ denotes the configuration that results from applying $S$ to $C$. 
Impossibility (elements)

- Consider $u$ to be 0 or 1; a configuration $C$ is $u$-valent if, starting from $C$, no matter how the processes behave, no decision other than $u$ is possible.

- We say that the configuration is univalent. Otherwise, the configuration is called bivalent.
P0(0) \[\begin{align*} W(X) & \quad R() \rightarrow Y \\ RI & \quad RJ \end{align*}\] Return(0)

P1(0) \[\begin{align*} W(Z) & \quad W(V) \\ RK & \quad RL \end{align*}\] Return(0)
\[ \text{Return}(1) \]

\[ P_0(1) \quad \overline{W(X)} \quad \overline{R()} \rightarrow Y \quad \overline{\text{Return}(1)} \]

\[ \begin{array}{c}
P_1(1) \\
\overline{W(Z)}
\end{array} \quad \overline{W(V)} \quad \overline{\text{Return}(1)} \]

\[ \begin{array}{c}
\text{RI} \\
\text{RJ}
\end{array} \quad \begin{array}{c}
\text{RK} \\
\text{RL}
\end{array} \]
\[ P_0(1) \quad W(X) \quad R() \rightarrow Y \quad \text{Return}(1/0) \]

\[ P_1(0) \quad W(Z) \quad W(V) \quad \text{Return}(1/0) \]
Impossibility (structure)

- **Lemma 1:** there is at least one initial *bivalent* configuration

- **Lemma 2:** given any bivalent configuration $C$, there is an *arbitrarily long schedule* $S(C)$ that leads to another bivalent configuration
The conclusion

- Lemmas 1 and 2 imply that there is a configuration $C$ and an *infinite* schedule $S$ such that, for any prefix $S'$ of $S$, $S'(C)$ is bivalent.

- In infinite schedule $S$, at least one process executes an infinite number of steps and does not decide

- A contradiction with the assumption that $A$ implements consensus.
Lemma 1

The initial configuration $C(0,1)$ is bivalent

Proof: consider $C(0,0)$ and $p_1$ not taking any step: $p_0$ decides 0; $p_0$ cannot distinguish $C(0,0)$ from $C(0,1)$ and can hence decides 0 starting from $C(0,1)$; similarly, if we consider $C(1,1)$ and $p_0$ not taking any step, $p_1$ eventually decides 1; $p_1$ cannot distinguish $C(1,1)$ from $C(0,1)$ and can hence decides 1 starting from $C(0,1)$. Hence the bivalency.
Lemma 2

Given any bivalent configuration C, there is an arbitrarily long schedule S such that S(C) is bivalent.

Proof (by contradiction): let S be the schedule with the maximal length such as D = S(C) is bivalent; p0(D) and p1(D) are both univalent: one of them is 0-valent (say p0(D)) and the other is 1-valent (say p1(D))
Lemma 2

- Proof (cont’d): To go from $D$ to $p_0(D)$ (vs $p_1(D)$) $p_0$ (vs $p_1$) accesses a register; the register must be the same in both cases; otherwise $p_1(p_0(D))$ is the same as $p_0(p_1(D))$: in contradiction with the very fact that $p_0(D)$ is 0-valent whereas $p_1(D)$ is 1-valent
Lemma 2

Proof (cont’d): To go from D to p0(D), p0 cannot read R; otherwise R has the same state in D and in p0(D); in this case, the registers and p1 have the same state in p1(p0(D)) and p1(D); if p1 is the only one executing steps, then p1 eventually decides 1 in both cases: a contradiction with the fact that p0(D) is 0-valent; the same argument applies to show that p1 cannot read R to go from D to p1(D)

Thus both p0 and p1 write in R to go from D to p0(D) (resp., p1(D)). But then p0(p1(D))= p0(D) (resp. p1(p0(D))= p1(D)) --- a contradiction.
The conclusion (bis)

 Lemmas 1 and 2 imply that there is a configuration C and an infinite schedule S such that, for any prefix S’ of S, S’(C) is bivalent.

 In infinite schedule S, at least one process executes an infinite number of steps and does not decide

 A contradiction with the assumption that A implements consensus.