# Shared Memory and Impossibility of Wait-Free Consensus

## Outline

- Asynchronous shared memory model
- Wait-free Consensus in shared memory with R/W variables
- Power of shared variables
- Wait-free Consensus hierarchy

## Shared Memory Models

- Asynchronous
  - No rounds: computation step is a *single* state transition of *some* process
- Two types of IPC:
  - Strongly coupled: through shared *variables*
    - Mutual exclusion, lower bounds
  - Loosely coupled: through shared *objects*
    - More general and flexible

## Shared Variables

- S. v. semantics are defined by its *type*
- Shared variable type:
  - A set V of values
  - An initial value $v_0 \in V$
  - A set of invocations
  - A set of responses
  - A function f: invocations $\times$ V $\rightarrow$ responses $\times$ V

## Read/Write Variable Type (register)

- Arbitrary set V of values
- An arbitrary initial value $v_0 \in V$
- Invocations: read, write(v), $v \in V$
- Responses: $v \in V$, ack
- f:

  f(read,v)=(v,v);

  f(write(w),v)=(ack,w)

## Computation

- State is a vector of the local process states and the shared variable values
- A computation step:
  - Process i chooses a shared variable x to access based on the current state
  - Invoke an operation on x and get a response
  - Perform a state transition based on the response and the current state
- An execution is an alternating sequence of states and steps, starting from an initial state

## Consensus in an Asynchronous Shared Memory with Registers

- n processes
- n 1-writer/multi-reader shared registers
- Initial value of i is in a local variable $x_i$
- Decision is written to a local variable $y_i$
- Any number of $t<n$ of processes can fail by stopping

## Consensus Requirements

- Agreement and Validity as before
- Termination: In a fair execution, each correct process eventually decides
  - Fairness: In an infinite execution, all correct processes must take infinitely many steps
  - This termination requirement is called *wait freedom*

# Lemma 1

- If
  - $C_1$ and $C_2$ are univalent states
  - $C_1$ and $C_2$ are indistinguishable to process i
- Then,
  - $C_1$ is v-valent iff $C_2$ is v-valent for $v \in \{0,1\}$

# Proof

- Run i alone from $C_1$. Eventually i must decide v. Why?
  - Otherwise, i does not decide in a fair execution where all processes but i have failed → not wait-free!
- Run the same steps from $C_2$. Process i must decide v
- Since both $C_1$ and $C_2$ are univalent → they are both v-valent. QED

## Lemma 2

- There exists a bivalent initial state

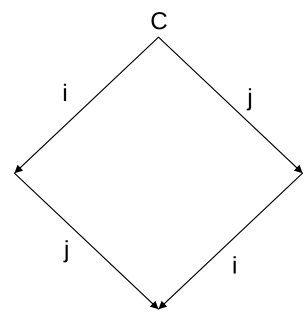
- 0…0 → 0-valent, 1…1→1-valent
- 01…1 looks like 0…0 to process 1
  - Lemma 1→01…1 is 0-valent
- 01…1 looks like 1…1 to process 2
  - Lemma 1→01…1 is 1-valent
- 01…1 is bivalent

## Critical Processes

- Every state has at most n successors
  - One for each process that can take a step
- Let C be a bivalent state
- A process i is *critical* in C if C·i is univalent

- **Lemma 3 (Diamond Lemma)**: If C is bivalent, then not all processes are critical in C

# Proof of Lemma 3




# Are we done?

- Yes! Start from an initial bivalent state
- For every subsequent state, use a non-critical process (Lemma 3) to make a step
- The resulting state is bivalent
- We constructed an execution where all states are bivalent
  - Nobody decides. Why?

## Read-Modify-Write variables

- RMW variable x has RMW operations
- Process i calling RMW operation performs all of the following steps *atomically*:
  - Read x
  - Perform some computation on x changing both internal state of i and x
  - Write the new value to x
- All synchronization primitives you've seen in the OS course are RMW variables

## Examples of RMW variables

- Test-And-Set (TS):

```
TS():
   tmp := x;
   x := 1;
   return tmp;
```

- f(TS,v) = (v, 1)

## Consensus using TS

- n=2
- Shared:
  - R[0], R[1]: R/W variables, initially ⊥
  - x: is a TS variable, V={0,1}, initially 0
- Process i:

```
write(R[i],init_i);
win := TS();
if (win = 0) decide R[i];
else decide R[1-i];
```

## Consensus using TS

- Upper bound: Wait-free Consensus can be solved using TS and registers for $n \geq 2$
- Lower bound: Wait-free Consensus cannot be solved using TS and registers for $n>2$
  - Proof: Re-prove the “diamond” lemma with $n>2$ and TS

## Compare-And-Swap

- CAS(old,new):

```
tmp := x;
if (x=old) then
        x := new;
return tmp;
```

- f: f(CAS(u,v),w) = (w,v), if w=u
  f(CAS(u,v),w) = (w,w), otherwise

## Consensus using CAS

Shared x: CAS variable, initially $\perp$

Process i:

prev := CAS($\perp$,$init_i$);
if (prev = $\perp$)
        decide $init_i$;
else
        decide prev;

## Consensus and CAS

- Wait-free Consensus can be solved using CAS for any n>0

## Consensus Numbers

- A variable type T has a Consensus number k if
  - Wait-free Consensus can be solved using shared variables of type T and registers for n=k
  - Wait-free Consensus cannot be solved using shared variables of type T and registers for n=k+1

# Wait-Free Consensus Hierarchy

| |
|---|
| CAS,…: Consensus number ∞ |
| ⋮ |
| |
| Test-and-Set, Fetch-and-Add, Swap,…: Consensus number 2 |
| R/W variables (registers): Consensus number 1 |

# Some Generalizations

- If T has non-trivial RMW operations, then Consensus # of T is $\geq 2$
- If T has only operations a, b that
  - commute: $f(a,f(b,v))=f(b,f(a,v))$, $\forall v \in V$, or
  - overwrite: $f(a,f(b,v))=f(a,v)$, $\forall v \in V$
    $f(b,f(a,v))=f(b,v)$, $\forall v \in V$

  Then, $C\#(T)<3$
- TS, FAA, Swap