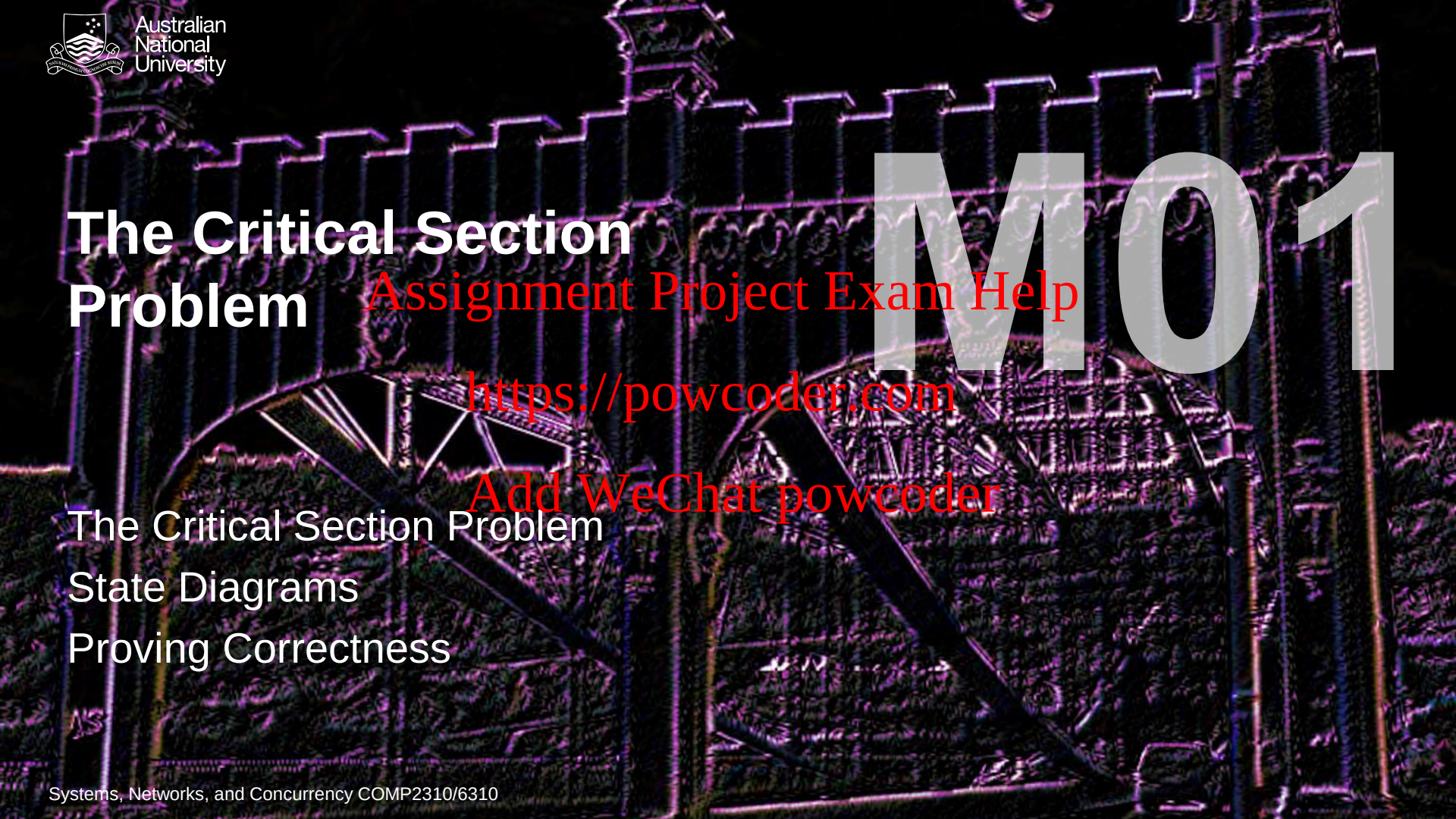


The Critical Section Problem

M01

The Critical Section Problem

State Diagrams

Proving Correctness

The Critical Section Problem

- N processes execute (infinite) instruction sequences concurrently
- Each process is divided into two sub-sequences: *critical* section and *non-critical* section
- Correctness properties:
  - **Mutual exclusion**: Instructions from critical sections of two or more processes must never be interleaved
  - **No deadlock**: If *some* processes are trying to enter their critical sections, then *one* of them must eventually succeed
  - **No starvation**: If *any* process tries to enter its critical section, then it must eventually succeed

M. Ben-Ari *Principles of Concurrent and Distributed Programming*, Addison-Wesley, second edition, 2006

The Critical Section Problem

- Further assumptions:
  - Synchronization mechanism consisting of pre-protocol and post-protocol before and after each critical section
  - Protocols use variables not accessed by critical or non-critical section
  - Processes may delay infinitely in non-critical sections
  - Processes do not delay infinitely in critical sections
- Further requirement: efficiency
  - Pre- and post-protocols require as little time and memory as possible, particularly in the case of no contention

Atomic Load and Store

- Assumption 1: every individual base memory cell (word) load and store access is *atomic*
- Assumption 2: there is *no* atomic combined load-store access


```
G : Natural := 0; -- assumed to be mapped on a 1-word cell in memory
```

```
task body P1 is
begin
G := 1;
G := G + G;
end P1;
```

```
task body P2 is
begin
G := 2;
G := G + G;
end P2;
```

```
task body P3 is
begin
G := 3;
G := G + G;
end P3;
```

What will be the value of `G` after it is initialized, and at program exit?

First Attempt

```
   type Task_Token is range 1 .. 2;
   Turn: Task_Token := 1;

   task body P is
   begin
     loop
p1     -- non-critical section P
p2     loop exit when Turn = 1; end loop;
p3     -- critical section P
p4     Turn := 2;
     end loop;
   end P;
```

```
   task body Q is
   begin
     loop
q1     -- non-critical section Q
q2     loop exit when Turn = 2; end loop;
q3     -- critical section Q
q4     Turn := 1;
     end loop;
   end Q;
```


- Mutual exclusion? Deadlock? Starvation?

Proving Correctness: State Diagrams

- Markov Property: no history

```
p5   a := a + 1;
p6   ...
```


Value of `a` at p6 only depends on value at p5

- Program state is tuple of program counters and variable values e.g. (p2, q3, 1)
  - process p is at statement p2
  - process q is at statement q3
  - Turn = 1

- State diagram: states (tuples), transitions, starting state

State Diagrams

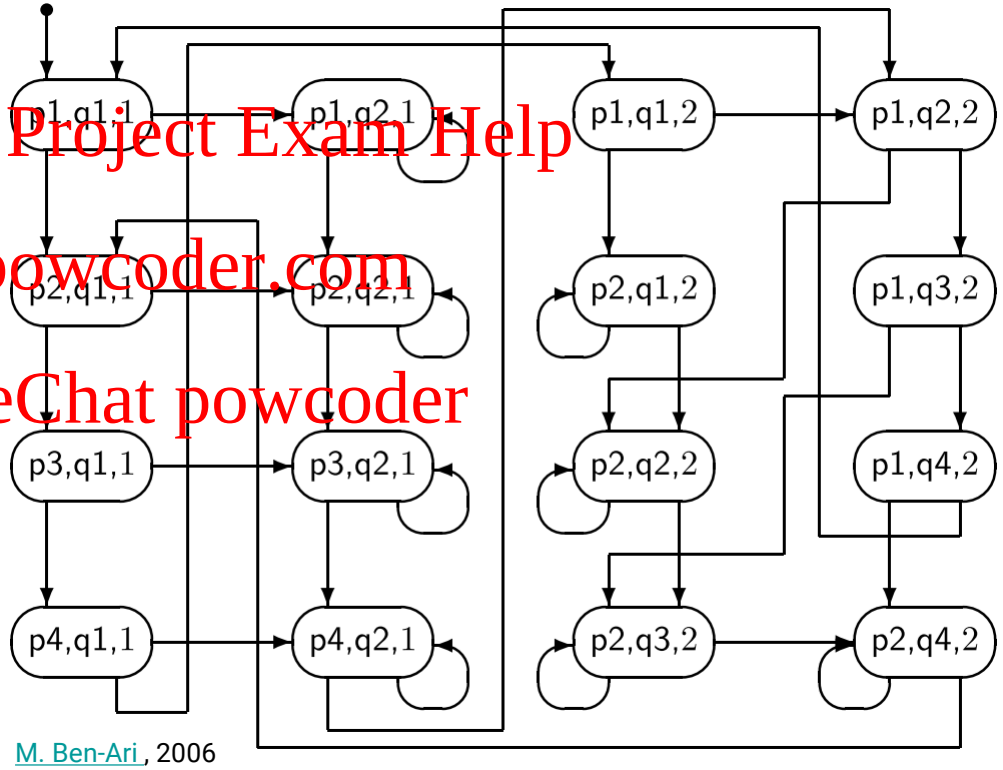
```
type Task_Token is range 1 .. 2;
Turn: Task_Token := 1;
```

```
task body P is
begin
  loop
p1   -- non-critical section P
p2   loop exit when Turn = 1; end loop;
p3   -- critical section P
p4   Turn := 2;
  end loop;
end P;
```

```
task body Q is
begin
  loop
q1   -- non-critical section Q
q2   loop exit when Turn = 2; end loop;
q3   -- critical section Q
q4   Turn := 1;
  end loop;
end Q;
```

Proof: Mutual Exclusion

- Too many states!
  4 states *
  4 states *
  2 values of turn
  = 32
  (only 16 reachable)
- Mutual exclusion:
  $\Box\neg(p3 \wedge q3)$



M. Ben-Ari , 2006

Collapsing States

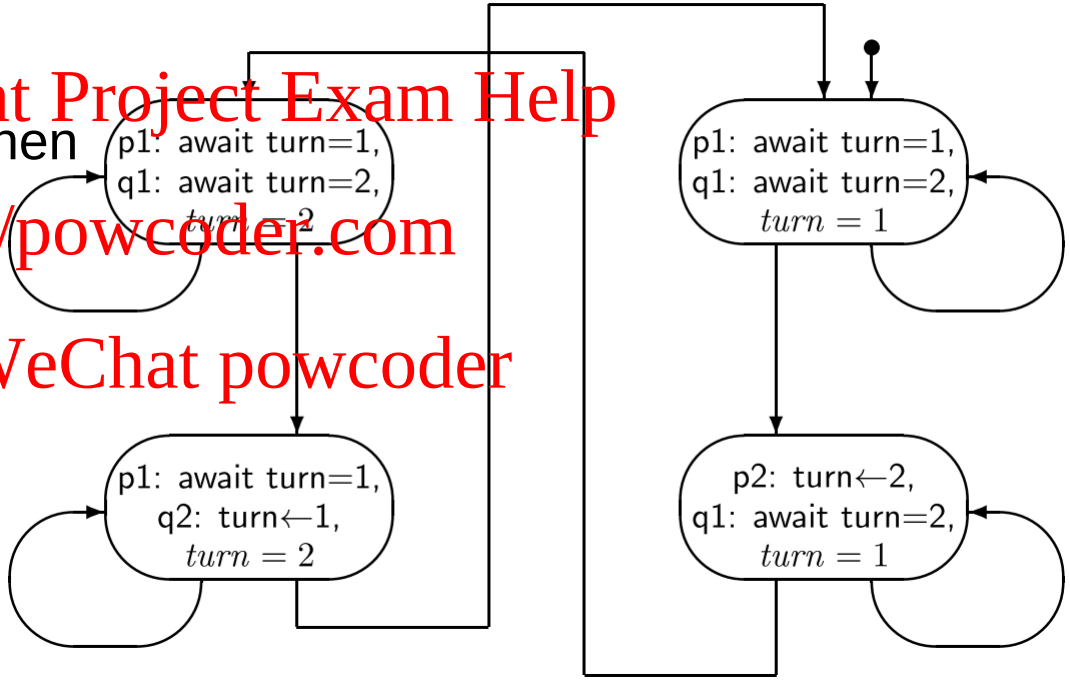
```
type Task_Token is range 1 .. 2;
Turn: Task_Token := 1;

task body P is
begin
  loop
p1   -- await Turn = 1;
p2   Turn := 2; -- critical
  end loop;
end P;
```

```
task body Q is
begin
  loop
q1   -- await Turn = 2;
q2   Turn := 1; -- critical
  end loop;
end Q;
```

Proof: No Deadlock

If *some* processes are trying to enter their critical section then *one* must eventually succeed

$(p1 \wedge q1) \rightarrow \Diamond(p2 \vee q2)$



M. Ben-Ari , 2006

(Non-Abbreviated) First Attempt

```
type Task_Token is range 1 .. 2;
Turn: Task_Token := 1;
```

```
task body P is
begin
  loop
p1   -- non-critical section P
p2   loop exit when Turn = 1; end loop;
p3   -- critical section P
p4   Turn := 2;
  end loop;
end P;
```

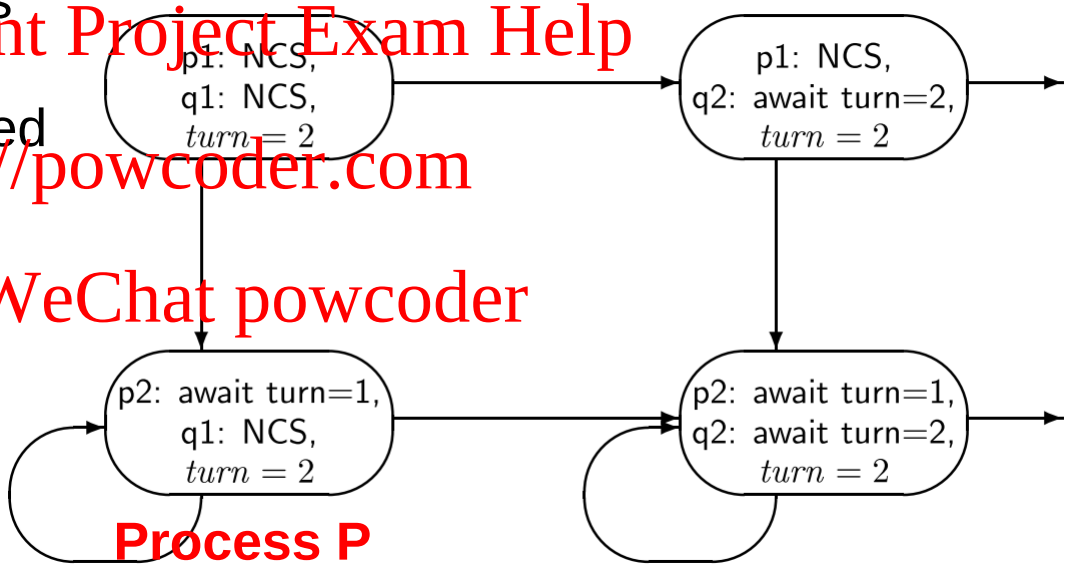
```
task body Q is
begin
  loop
q1   -- non-critical section Q
q2   loop exit when Turn = 2; end loop;
q3   -- critical section Q
q4   Turn := 1;
  end loop;
end Q;
```


- Mutual exclusion? Deadlock? Starvation?

Proof: No Starvation

If *any* process tries to enter its critical section, then it must eventually succeed

p2 → ◇p3
q2 → ◇q3



M. Ben-Ari , 2006