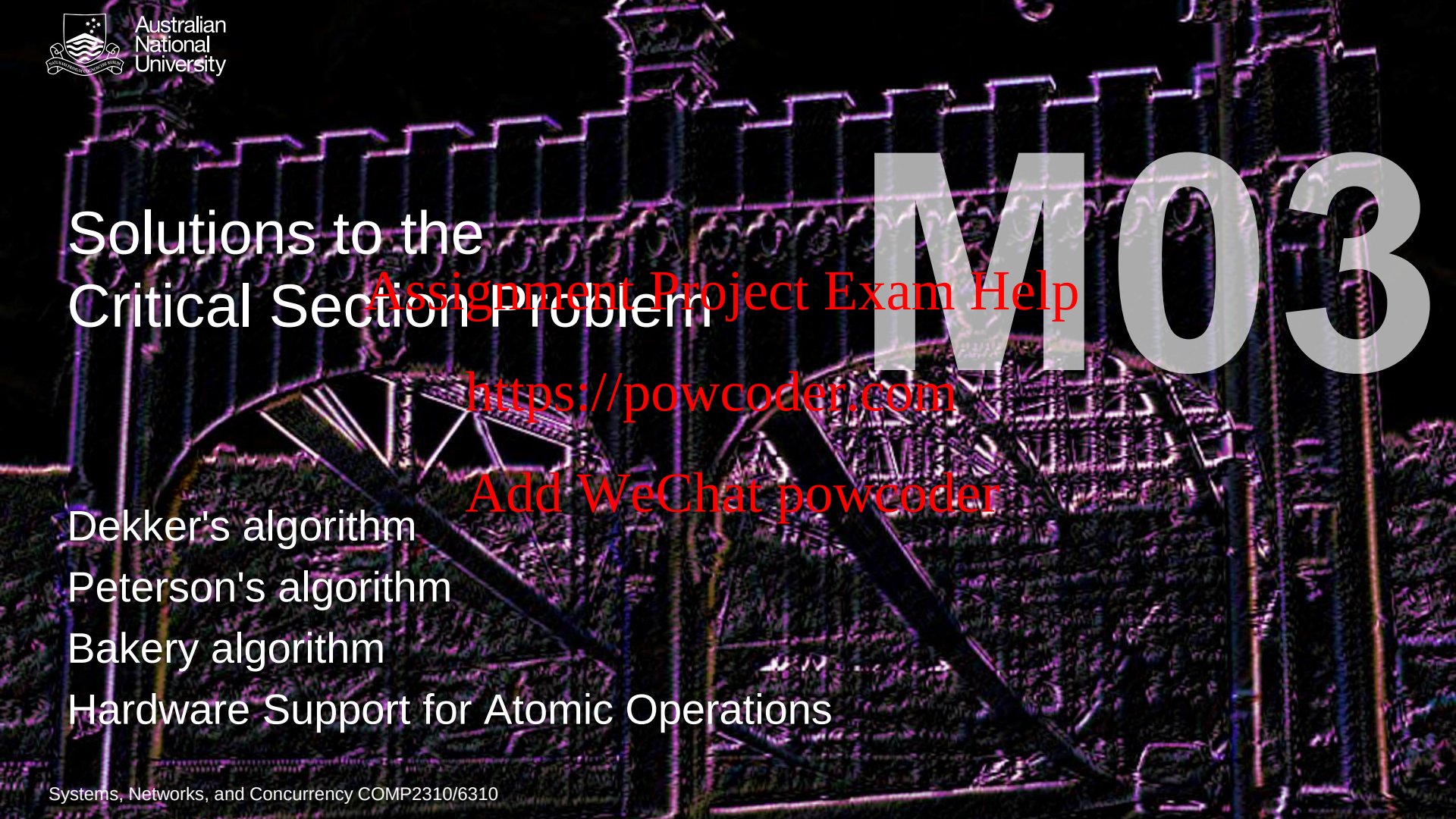

Australian National University

M03

Solutions to the Critical Section Problem

- Dekker's algorithm
- Peterson's algorithm
- Bakery algorithm
- Hardware Support for Atomic Operations

Dekker's Algorithm

```
Want_P, Want_Q : Boolean := False;
type Task_Token is range 1 .. 2;
Turn: Task_Token := 1;
```

`Turn` = right to insist on entering

Satisfies

- mutual exclusion
- no deadlock
- no starvation
- (for two tasks only)

```
task body P is
begin
   loop
p1    -- non-critical section P
p2    Want_P := True;
p3    loop exit when Want_Q = False;
p4       if Turn = 2 then
p5          Want_P := False;
p6          await Turn = 1;
p7          Want_P := True;
          end loop;
p8    -- critical section P
p9    Turn := 2;
p10   Want_P := False;
   end loop;
end P;
```

Peterson's Algorithm

```
Want_P, Want_Q : Boolean := False;
type Task_Number is range 1 .. 2;
Last: Task_Number := 1;

task body P is
begin
  loop
p1  -- non-critical section P
p2  Want_P := True;
p3  Last := 1;
p4  loop exit when Want_Q = False;
    end loop;
p5  -- critical section P
p6  Want_P := False;
  end loop;
end P;
```

```
task body Q is
begin
  loop
q1  -- non-critical section Q
q2  Want_Q := True;
q3  Last := 2;
q4  loop exit when Want_P = False;
    end loop;
q5  -- critical section Q
q6  Want_Q := False;
  end loop;
end Q;
```

Bakery Algorithm

- Processes $P_1 \ldots P_N$ competing for access to own critical regions. Each process P_i supplies a globally readable number t_i (‘ticket’) (initialized to 0).
- Before a process P_i enters a critical section:
  - P_i draws a new number $t_i > t_j$; $\forall j \neq i$
  - P_i is allowed to enter the critical section iff: $\forall j \neq i : t_i < t_j \vee t_j = 0$
- After a process P_i leaves a critical section, reset its ticket $t_i \leftarrow 0$
- Can we ensure that processes won’t read each other's ticket numbers while still calculating?
- Can we ensure that no two processes draw the same number?

Bakery Algorithm

```
 No_Of_Tasks : constant Positive := …;
 type Task_Range is mod No_Of_Tasks;
 Choosing : array (Task_Range) of Boolean := (others => False);
 Ticket : array (Task_Range) of Natural := (others => 0);

task type P (this_id: Task_Range);
task body P is begin
  loop
    -- non-critical section;
    …
    -- critical section;
    …
  end loop;
end P;
```

Bakery Algorithm

```
task body P is begin loop
  -- non-critical section;
  Choosing (this_id) := True;
  Ticket (this_id) := Max (Ticket) + 1;
  Choosing (this_id) := False;
  for id in Task_Range loop
    if id /= this_id then
      loop exit when not Choosing (id); end loop;
      loop exit when Ticket (id) = 0 or else Ticket (this_id) < Ticket (id)
        or else (Ticket (this_id) = Ticket (id) and then this_id < id);
      end loop;
    end if;
  end loop;
  -- critical section;
  Ticket (this_id) := 0;
end loop; end P;
```

Realistic Hardware Support

Atomic test-and-set operations:

- `[L := C; C := 1]`

Atomic exchange operations:

- `[Temp := L; L := C; C := Temp]`

Memory cell reservations:

- `L :=`^{R} `C;` read by using special instruction that puts ‘reservation’ on `C`
- … calculate a <new value> for `C` …
- `C :=`^{T} `<new value>;` – succeeds iff `C` was not manipulated by other processors or devices since the reservation

Mutual exclusion: Atomic Test-and-Set

```
type Flag is Natural range 0..1; C : Flag := 0;

task body P is
  L : Flag;
begin
  loop
    loop
      [L := C; C := 1];
      exit when L = 0;
    end loop;
    ------ critical section;
    C := 0;
  end loop;
end P;
```

Mutual exclusion: Atomic Test-and-Set

```
type Flag is Natural range 0..1; C : Flag := 0;

task body P is
  L : Flag;
begin
  loop
    loop
      [L := C; C := 1];
      exit when L = 0;
    end loop;
    ------ critical section;
    C := 0;
  end loop;
end P;
```


Works for any number of processes

Mutual exclusion

No deadlock

No global livelock

~~No starvation~~ (busy waiting loops)

Mutual exclusion: Atomic Exchange

```
type Flag is Natural range 0..1; C : Flag := 0;

task body P is
  L : Flag := 1;
begin
  loop
    loop
      [Temp := L; L := C; C := Temp];
      exit when L = 0;
    end loop;
    ------ critical section;
    L := 1; C := 0;
  end loop;
end P;
```

Mutual exclusion: Atomic Exchange

```
type Flag is Natural range 0..1; C : Flag := 0;

task body P is
  L : Flag := 1;
begin
  loop
    loop
      [Temp := L; L := C; C := Temp];
      exit when L = 0;
    end loop;
    ------ critical section;
    L := 1; C := 0;
  end loop;
end P;
```


Works for any number of processes

Mutual exclusion

No deadlock

No global livelock

~~No starvation~~ (busy waiting loops)

Mutual exclusion: Memory Reservation

```
type Flag is Natural range 0..1; C : Flag := 0;

task body P is
  L : Flag;
begin
  loop
    loop
      L :=R C; C :=T 1;
      exit when Untouched and L = 0;
    end loop;
    ------ critical section;
    C := 0;
  end loop;
end P;
```

Mutual exclusion: Memory Reservation

```
type Flag is Natural range 0..1; C : Flag := 0;

task body P is
  L : Flag;
begin
  loop
    loop
      L :=R C; C :=T 1;
      exit when Untouched and L = 0;
    end loop;
    ------ critical section;
    C := 0;
  end loop;
end P;
```


Works for any number of processes

Mutual exclusion

No deadlock

No global livelock

~~No starvation~~ (busy waiting loops)