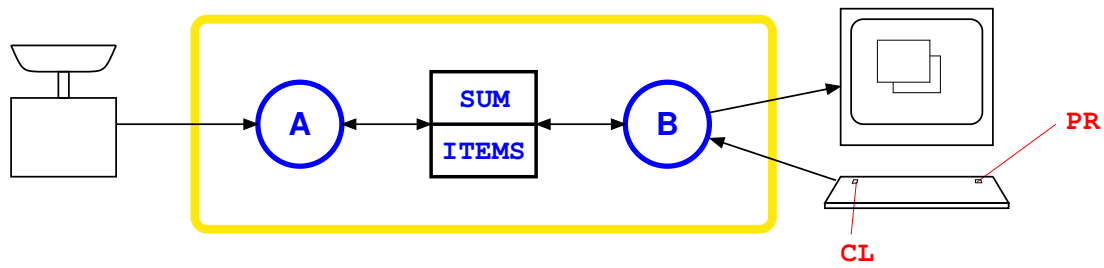
Course 02158

Critical Regions

Hans Henrik Løvengreen

DTU Compute

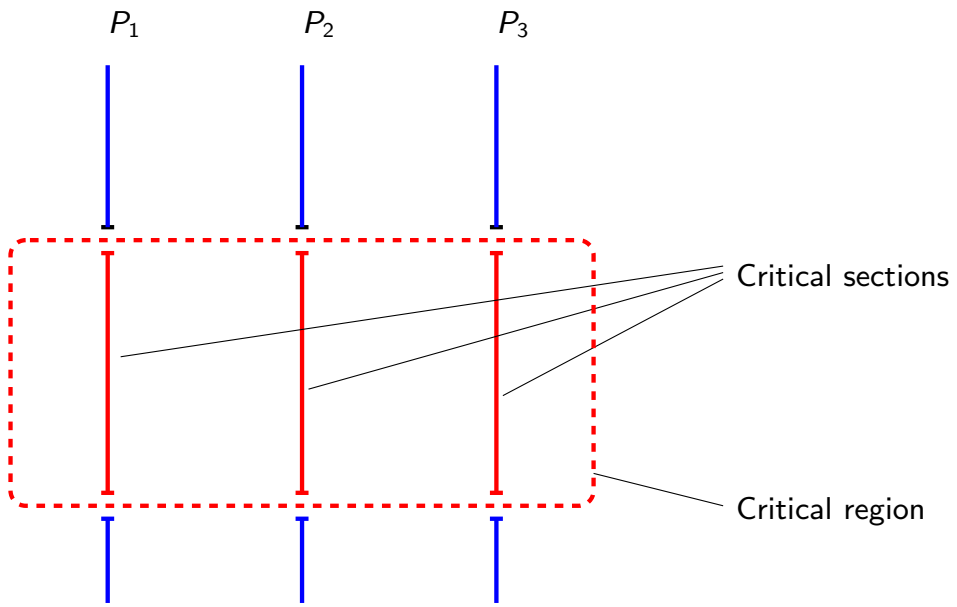
The Sharing Problem




```
process A =
  loop
    read w
    SUM := SUM + w
    ITEMS := ITEMS + 1
  end loop
```

```
process B =
  loop
    read key
    case key
      PR : if ITEMS ≠ 0 then
             print SUM/ITEMS
      CL : SUM := 0
           ITEMS := 0
  end loop
```

Critical Regions and Critical Sections



Critical Region Properties

Assumptions

- Critical sections are established with *entry* and *exit protocols*
- A process *eventually leaves* the critical section

Safety property

- **Mutual Exclusion** At most one process in a region at a time

Liveness properties

- **Obligingness** A lonely process will succeed
  [Absence of unnecessary delay]
- **Resolution** One out of several processes will succeed
  [Absence of deadlock and livelock]
- **Fairness** Any process will succeed
  [Eventual entry]

Critical Region with Shared Variables

Coarse-grained solution

- **var** $in1, in2 : bool := false$;

process P_1
loop
$\langle \neg in2 \rightarrow in1 := true \rangle$;
critical section$_1$;
$in1 := false$;
noncritical section$_1$
end loop

process P_2
loop
$\langle \neg in1 \rightarrow in2 := true \rangle$;
critical section$_2$;
$in2 := false$;
noncritical section$_2$
end loop

Critical Region with Shared Variables

Fine-grained attempt I

- **var** $in1, in2 : bool := false$;

process P_1
loop
while $in2$ **do skip**;
$in1 := true$;
critical section$_1$;
$in1 := false$;
noncritical section$_1$
end loop

process P_2
loop
while $in1$ **do skip**;
$in2 := true$;
critical section$_2$;
$in2 := false$;
noncritical section$_2$
end loop

Critical Region with Shared Variables

Fine-grained attempt II

- **var** $in1, in2 : bool := false$;

process P_1
loop
$in1 := true$;
while $in2$ **do skip**;
critical section$_1$;
$in1 := false$;
noncritical section$_1$
end loop

process P_2
loop
$in2 := true$;
while $in1$ **do skip**;
critical section$_2$;
$in2 := false$;
noncritical section$_2$
end loop

Critical Region with Shared Variables

Fine-grained attempt III (Back-off)

- **var** $in1, in2 : bool := false$;

process P_1
loop
$in1 := true$;
while $in2$ **do** { $in1 := false$;
delay;
$in1 := true$}
critical section$_1$;
$in1 := false$;
noncritical section$_1$
end loop

process P_2
loop
$in2 := true$;
while $in1$ **do** { $in2 := false$;
delay;
$in2 := true$}
critical section$_2$;
$in2 := false$;
noncritical section$_2$
end loop

Critical Region with Shared Variables

Dekker's Algorithm

- **var** $in1, in2 : bool := false;$
  $turn : int := 1;$

process P_1
loop
$in1 := true;$
while $in2$ **do**
if $turn = 2$ **then**
$\{ in1 := false;$
while $turn = 2$ **do skip**
$in1 := true\};$
critical section$_1$;
$in1 := false;$
$turn := 2;$
noncritical section$_1$
end loop

process P_2
loop
$in2 := true;$
while $in1$ **do**
if $turn = 1$ **then**
$\{ in2 := false;$
while $turn = 1$ **do skip**
$in2 := true\};$
critical section$_2$;
$in2 := false;$
$turn := 1;$
noncritical section$_2$
end loop

Critical Region with Shared Variables

Peterson's Algorithm

- **var** $in1, in2 : bool := false;$
  $turn : int := 1;$

process P_1
loop
$in1 := true;$
$turn := 2;$
while $in2 \wedge turn = 2$ **do skip**;
critical section$_1$;
$in1 := false;$
noncritical section$_1$
end loop

process P_2
loop
$in2 := true;$
$turn := 1;$
while $in1 \wedge turn = 1$ **do skip**;
critical section$_2$;
$in2 := false;$
noncritical section$_2$
end loop

Concurrency in real life: The Bascule Bridge of CP Frederik

- Since 2018 remotely controlled from Copenhagen
- Serious safety incidient in Aug 2022
  +

Bascule Bridge Fault

Remote control

- **var** *leaf_up*, *boom_up* : *bool*;
- **process** *RaiseLeaf*
  if ¬*boom_up* **then**
  start bridge alarm;
  sleep(10 *s*);
  leaf_up := *true*;
  start raising bridge leaf;
  ⋮

process *RaiseBoom*
start barrier alarm;
sleep(2 *s*);
if ¬*leaf_up* **then**
boom_up := *true*;
start raising boom;
⋮

- The leaves were down (¬*leaf_up*)
- The boom barrier had been lowered (¬*boom_up*)
- Both raise processes were (by manual mistake?) started concurrently

Critical Region with Shared Variables

Test-and-Set Solution

- Let $TS(\textbf{var } l) = \langle \textbf{var } r : bool;\ (r, l) := (l, true);\ \textbf{return } r \rangle$
- **var** $lock : bool := false$;

process P_1
  loop
    while $TS(lock)$ **do skip**;
    critical section$_1$;
    $lock := false$;
    noncritical section$_1$
  end loop

process P_2
  loop
    while $TS(lock)$ **do skip**;
    critical section$_2$;
    $lock := false$;
    noncritical section$_2$
  end loop

Critical Region with Shared Variables

Test-and-Set Solution

- Let $TS(\textbf{var } l) = \langle \textbf{var } r : bool;\ (r, l) := (l, true);\ \textbf{return } r \rangle$
- **var** $lock : bool := false$;

process $P[i : 1..n]$
  loop
    while $TS(lock)$ **do skip**;
    critical section$_i$;
    $lock := false$;
    noncritical section$_i$;
  end loop

Critical Region with Shared Variables

Test-and-Test-and-Set Solution

- Let $TS(\textbf{var } l) = \langle \textbf{var } r : bool;\ (r, l) := (l, true);\ \textbf{return } r \rangle$
- **var** $lock : bool := false$;
  process $P[i : 1..n]$
  loop
  while $lock$ **do skip**;
  while $TS(lock)$ **do**
  while $lock$ **do skip**;
  critical section$_i$;
  $lock := false$;
  noncritical section$_i$
  end loop

Critical Region with Shared Variables

Ticket Algorithm

- **var** $number, next : integer := 1$;
  $turn[i : 1..n] : integer := 0$;
  process $P[i : 1..n]$
  loop
  $\langle\, turn_i := number;\ \ number := number + 1 \,\rangle$;
  $\langle\, \textbf{await } turn_i = next \,\rangle$;
  critical section$_i$;
  $\langle\, next := next + 1 \,\rangle$;
  noncritical section$_i$;
  end loop

Invariant

- $next > 0 \wedge$
  $\forall i:\ turn_i < number \wedge$
  $(in\ crit_i \Rightarrow turn_i = next) \wedge$
  $(turn_i > 0 \Rightarrow \forall j \neq i : turn_i \neq turn_j)$

Critical Region with Shared Variables

Bakery Algorithm (coarse-grained)

- **var** $turn[i : 1..n] : integer := 0;$
  process $P[i : 1..n]$
  loop
  $\langle\, turn_i := \max(turn[i : 1..n]) + 1 \,\rangle;$
  for $[j : 1..n, j \neq i]$
  $\langle$ **await** $turn_j = 0 \vee turn_i < turn_j \,\rangle;$
  critical section$_i$;
  $turn_i := 0;$
  noncritical section$_i$;
  end loop

Invariants

- $\forall i : (turn_i > 0 \Rightarrow \forall j \neq i : turn_i \neq turn_j)$
- $\forall i : in\ crit_i \Rightarrow \left(\begin{array}{l} turn_i > 0 \wedge \\ (\forall j \neq i : turn_j = 0 \vee turn_j > turn_i) \end{array} \right)$