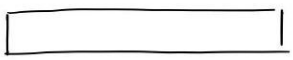
Lec 25 Concurrent Data Structure & Work Stealing Scheduling

Key ideas
- Lock-free data structure
- Linearisation
- Compare and swap (CAS)
- Concurrent deque
- Randomised stealing

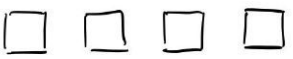
Recall greedy scheduling $T = \frac{W}{P} + S$
But in real world we need to find work to schedule

Working with async. parallel processors

Model



memory



cores — can delay
can get unscheduled
can have different clock rate
⋮

Assume arbitrary interleaving

P1	P2
$r_1 \leftarrow mem[a]$	$r_2 \leftarrow mem[a]$
$r_1 = r_1 + 1$	$r_2 = r_2 + 1$
$mem[a] \leftarrow r_1$	$mem[a] \leftarrow r_2$

race condition possible
mem[a] can end up
+1 or +2

Lock-free data structure

Def Lock free data structure
- Supports certain operations
- Shared across processes
- At least one process making process
  (puts "lock-free lock" i.e. some lauda around critical code)

Linearisability

Operations : load, increment push, pop

→ They can appear interleaved but correctness captured sequentially

Compare and swap

On x86: CMPXCHG

Analogous to:

```
CAS : α ref → (α × α) → bool
CAS r (old, new) =
  let
    a = !r
  in
    if a = old then (r := new ; true)
    else false
  end
```

Note this code is not safe. Processor implements this on hardware level as instruction

Linearisable increment ← no lock involved

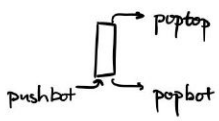
```
Inc (r : int ref) =
  let
    a = !r
  in
    if CAS r (a, a+1) then ()
    else Inc r
  end
```

Work stealing scheduler (randomised)

How to do f || g ?

→ Forking puts job into shared data structure
Idle threads find the job and do it

Each processor keeps a deque DQ
lock-free, linearisable



- when encountering f || d on processor p

```
DQp . pushbot(g)
run f
wait for result of g
```

- If processor p done or while waiting

case DQ_p.popbot() of
  Some f ⇒ run f
  None ⇒ repeat (randomly steal from top of another processor's DQ)

Analysis — why this works well

1. Most of the time stealing over work
   since we prioritise push and pop from bottom ⇒ good locality

2. Minimise sequentialisation of deque operations, low contention
   since most of the time DQs are long so 1 happens and we don't sequentialise
   also randomness helps spread out contentions

Thm number of steals to attempt is in $O(PS)$
time is in $O(\frac{W}{P} + S)$

P = num processors
S = span