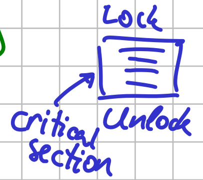
Parallel machine – PC, threads with shared memory (model: parallel RAM)

↓

D.S.?

↓

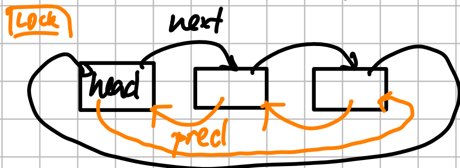
inconsistency → solve by locking  e.g. Mutex (mutual exclusion)
↳ ops: Lock, Unlock




↓

granularity
- global
- per instance
- per item
- per bucket (problem: re-hash?)

problem: Doubly linked lists with per list locking




Goal: Move item from list A to list B atomically.

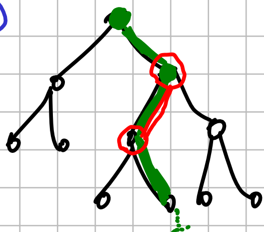
↳ Lock(A)
Lock(B)
Remove item from A
Insert item to B
Unlock(B)
Unlock(A)

Lock(B)
Lock(A)
↯ deadlock

What if another thread tries to move from B to A at the same time?

Solution:
Establish linear order of all locks (e.g. by address in memory) & always lock in this order

## Binary Search Trees

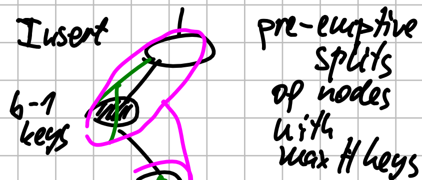
① 

per-node locks
locking a path
problem: root lock serializes everything

② lock the "active" part of the path (e.g., the current node & its parent)

works for Find
Insert without balancing
~ Delete

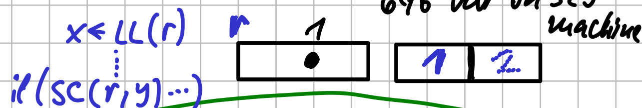
(a,b)-trees with top-down operations with $b \geq 2a$




lock current node & its parent

Problems of locking:
- deadlocks
- fairness
- priority inversion
- performance
- fault tolerance

→ design lock-free data structures



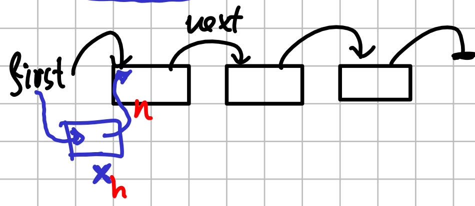

Atomic operations:
- always [ atomic register (read/write)
- some of [ exchange, test & set bit, fetch & add
- one of [ LL/SC (load locked / store conditional), compare & exchange (swap) (CAS): register, expected value, new value

HW supports

```
atomic {
  old = reg
  if old == expected:
    reg = new
} return old
```

Df: Linearizability

# Lock-free stack




stack: atomic ptr first

item: atomic ptr next
data...
atomic int ref

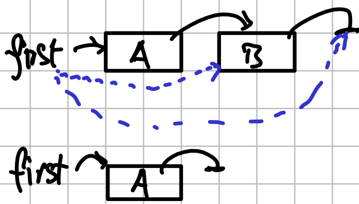
Push(x):
loop:
h ← first
x.next ← h
~~first ← x~~
if CAS(first, h, x) = h:
return
retry

Pop:
loop:
h ← first
h.ref++
if first ≠ h:
h.ref--
retry
n ← h.next
if CAS(first, h, n) = h:
h.ref--
return h
h.ref--
retry

① Is this finite? ... in a sense yes ...

② Is this correct? ... NO

"ABA-problem"




Thread 1:
Pop → A
Pop → B
Push(A)

Thread 2 inside Pop
h ← first A
n ← h.next B

CAS(first, A, B)   first → B

Solutions: a) replace CAS by LL/SC

b) double CAS (CAS2)   CAS2(<first, h.next>, <h, n>, <n, n>)
implemented almost nowhere

c) wide CAS (WCAS/DCAS) → versioning of pointers
DCAS(<first, version>, <h, ver>, <n, ver+1>)
commonly implemented

d) avoid recycling memory

Memory allocation:
a) keep a free list & empty it later
b) reference counting
c) hazard pointers : for each thread, keep pointers to accessed items

for each thread: a block of k pointers

Parallel DS:
- blocking
- obstruction-free (if other threads stay stopped, we will succeed in finite time)
- lock-free (at least 1 thread succeeds in finite time)   Our stacks
- wait-free (every thread —— " —— )
- bounded wait-free + upper bound on time to success