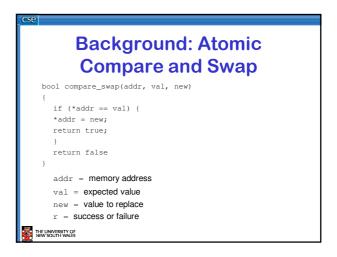




Can we avoid locking? • Yes - in some cases • Lock-free data structures - Need hardware help • compare-and-swap() • exchange() • test_and_set()

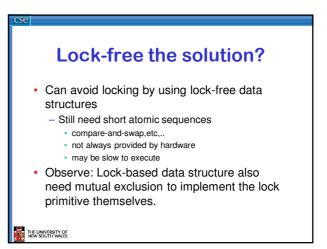


```
CAS Example

• Lock-free atomic increment

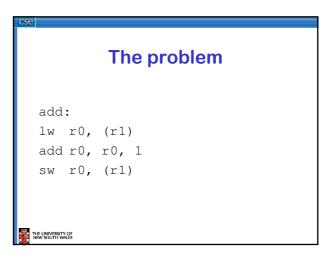
atomic_inc(int *addr)
{
    do {
        old = *addr;
        new = old + 1;
    } while (!compare_and_swap(addr, old, new));
}

• Lock-free does not preclude starvation
• Tricky to implement more complex structures
```



How do we provide efficient atomic sequences? Interrupt disabling? Syscalls? Processor Instructions?

THE UNIVERSITY OF NEW SOUTH WALES



Optimistic Approach

Assume the critical code runs atomically

- Atomic Sequence

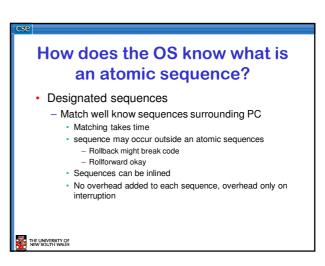
If an interrupt occurs, OS recovers such that atomicity is preserved

Two basic mechanisms

- Rollback

Only single memory location update
Guarantee progress???

- Rollforward



• Static Registration

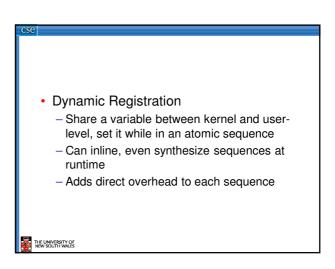
- All sequences are registered at program startup

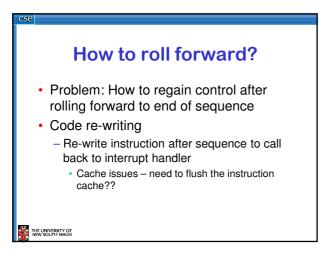
• No direct overhead to sequences themselves

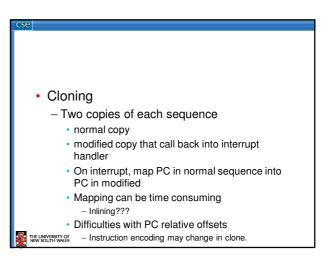
• Limited number of sequences

- Reasonable to identify on interrupt

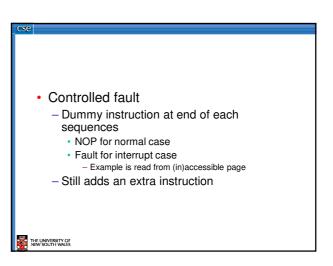
- No inlining







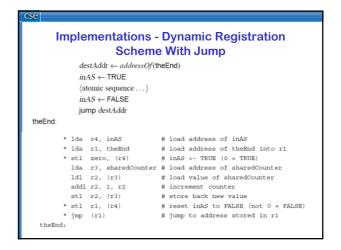
Computed Jump
 Every sequence uses a computed jump at the end
 Normal sequence simply jmp to next instruction
 Interrupted sequence jumps to interrupt handler
 Adds a jump to every sequence



Limiting Duration of Roll forward

• Watchdog

• Restriction on code so termination can be inspected for



```
Implementations - Dynamic Registration
Scheme With Fault

destAddr ← addressOf(theEnd)
inAS ← TRUE
⟨atomic sequence ...⟩
theEnd:
inAS ← *falseOrFault
```

