Shareable Data Structures

MALCOLM CROWE

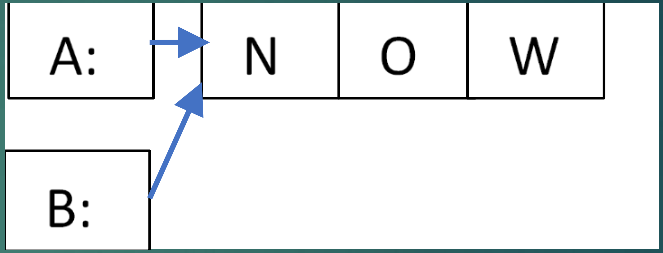
OCTOBER 2018

Shareable data structures

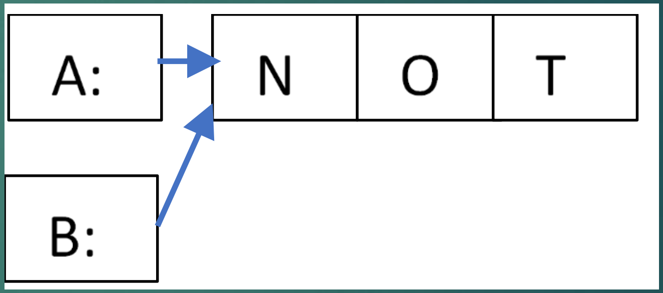
- Data Structures are in all Computing courses
  - Revisited when student has reached Threading
- Threading examples show need for locking
  - Students learn this is why strings are immutable
    - At least in C# and Java – “value semantics”
- But why do we use unsafe data structures?
  - In this course we focus on SAFE data structures
  - For sharing and copying between threads
- The reduces the need for complex locking

What is unsafe?

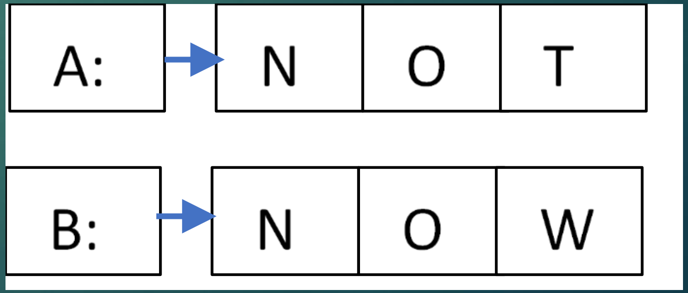
- Example: Arrays A and B – in Java (say)
- After B=A we have



- Then A[2]=‘T’ gives
- (correct, maybe?)



- A safe array would give
- A=A.Set(2,’T’)
- Change is just to A



We learn about cloning?

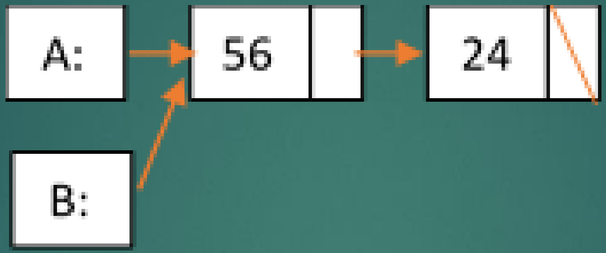
- A concept often not grasped by students
- When a list is passed “by value” to a proc
- There is nothing to stop the proc changing it
- With value semantics this shouldn’t occur
- So maybe we need to stop using lists!
- Immutable strings are still useful, so
  - Our data structures have immutable contents
- We will still need locking for mutable things
  - We keep it to a minimum to simplify our design

In database technology

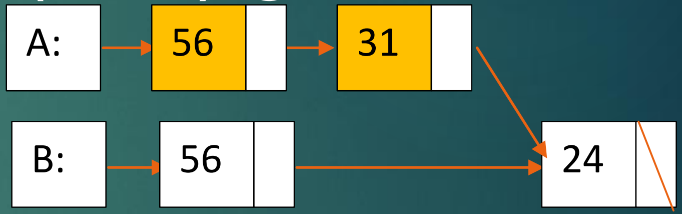
- Once we have enough structures
  - We show how a full DBMS can be built
- Taking a snapshot is as easy as B=A above
  - People with copies can consider changes
  - On ROLLBACK they can simply be forgotten
  - On COMMIT we need to check for conflicts
    - And the DBMS can accept the changes in master copy
- The list of master copies of databases in use
  - Will be the DBMS' only unsafe data structure!

Example: a safe linked list

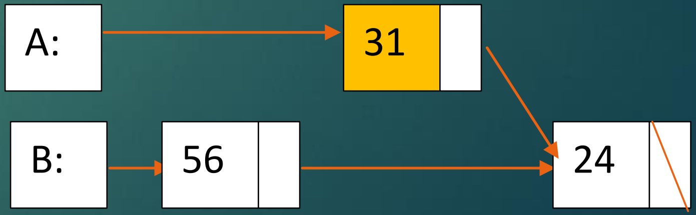
- After B=A suppose we have linked list (56,24)



- For a safe list, A=A.InsertAt(1,31) gives:
- Coloured nodes are new



- Then A=A.RemoveAt(0):
- Note B still has the old list



Implementation in Java

- Shareable data structures have all fields `public final`
- So a safe linked list of integers might be:

```
public class SListOfInt {
  public final int element;
  public final SListOfInt next;
//.. And we need at least one constructor
//   and the methods InsertAt, RemoveAt
}
```