Some Non-Blocking Data Structures
Some Non-Blocking Data Structures
Handout: Pg 126 of MLS
Sequential semantics of a Queue

- **Enqueue**
  - Prev tail node $T \rightarrow \textit{next}$ points to new node $N$
  - Tail pointer points to node $N$

- **Dequeue (if queue is empty)**
  - Returns empty

- **Dequeue (if queue is not empty)**
  - Returns node $H$ pointed to by head
  - New head points to node in $H \rightarrow \textit{next}$
CAS names

In enqueue:

- CAS1: if CAS(&t.p->next) ...
- CAS2: (void) CAS(&tail, t, <n.p, t.c+1>)
- CAS3: (void) CAS(&tail, t, <w, t.c+1>)

In dequeue

- CAS4: (void) CAS(&tail, t, <n.p, t.c+1>)
- CAS5: (void) if CAS(&head, h, <n.p, h.c+1>)

Which of these CASes enforce the semantics?
Which are the linearization points?
Memory Management in M&S Queue

- new in enqueue
  - Why the fence(W||W)?
- free_for_reuse in dequeue
  - What is this?
  - Could we not simply:

```c
rtn := n.p->val.load()
if CAS(&head, h, <n.p, h.c+1>)
    fence(W||W)
    free(h.p)
    break
```
How do we prevent premature reclamation of allocated data?

- while somebody is holding a reference to it?
Performance

- No way to theoretically model the performance of these concurrent data structures yet.
  - in advance
- Run microbenchmarks on concurrent data structures
  - measure throughput