

9 Wait-free Synchronization

Design data structures in a way that allows safe concurrent access.

- No mutual exclusion necessary.
- No possibility of deadlock.

Approach: use a single atomic operation to

- commit all changes
- move the shared data structure from one consistent state to another

Wait-free bounded buffer: (use busy-waiting for condition synchronization)

```
char buffer[BUF-SIZE];
int head = 0;
int tail = 0;

void Produce(char item) {
    while((tail + 1) % BUF-SIZE == head);
    buffer[tail] = item;
    tail = (tail + 1) % BUF-SIZE;
}

char Consume() {
    char item;
    while(tail == head);
    item = buffer[head];
    head = (head + 1) % BUF-SIZE;
    return item;
}
```

Simple example: singly-linked queue insertion

```
QElem *queue;
```

```
void Insert(item) {  
    QElem *new = malloc(sizeof(QElem));  
    new->item = item;  
    do {  
        new->next = ldl(&queue);  
    } while(!stc(&queue, new));  
}
```

This only works for simple data structures where changes can be committed with one store instruction. More general approach:

Maintain a pointer to the “master copy” of the data structure. To modify:

1. remember current value of the master pointer
2. copy shared data structure to a scratch location
3. modify scratch copy
4. *atomically*:
 - verify that master pointer has not changed
 - write pointer to refer to new master
5. if verification fails (another process interfered), start over at step 1.

Works for arbitrarily complex data structure, but: copying cost may be a concern.