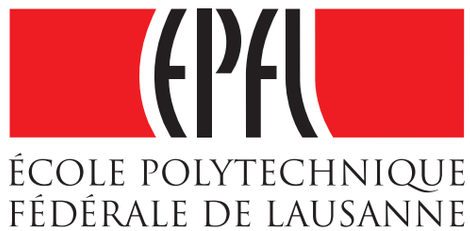
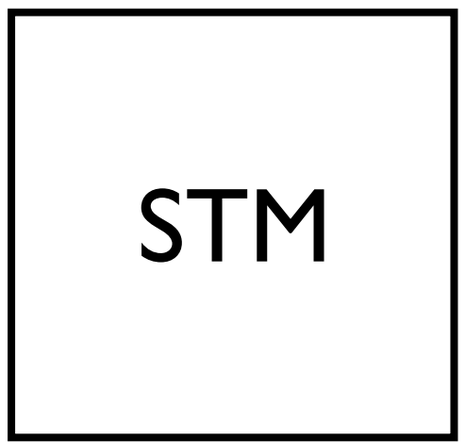
Solution for Exercise 10

Concurrent Algorithms 2010



Problem 1

Problem 1




```
tx_start
tx_commit
tx_read
tx_write
```

Problem 1

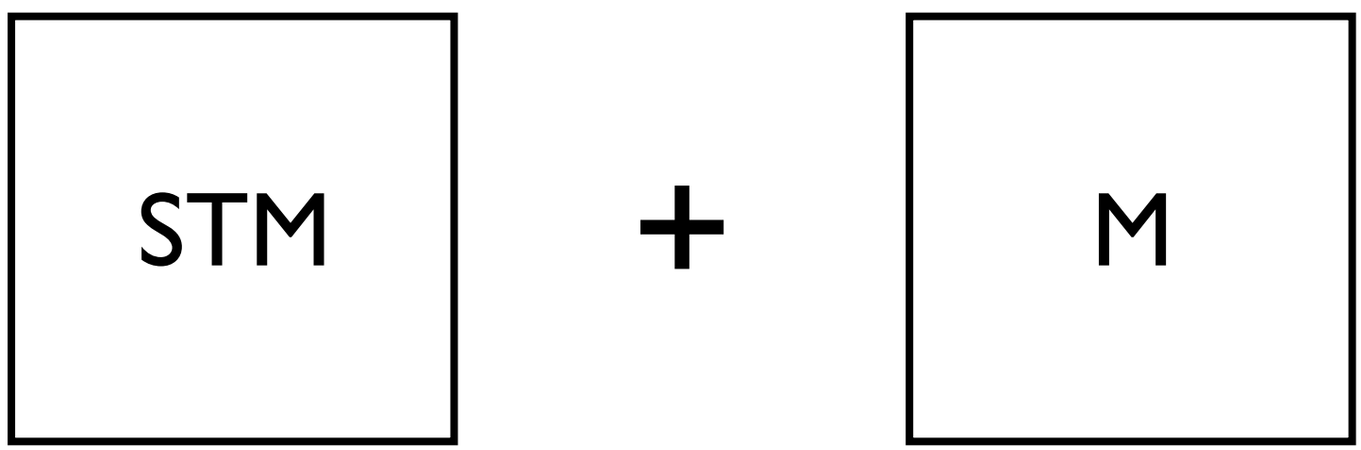
STM

```
tx_start
tx_commit
tx_read
tx_write
```

M

```
alloc(size)
free(ptr)
```

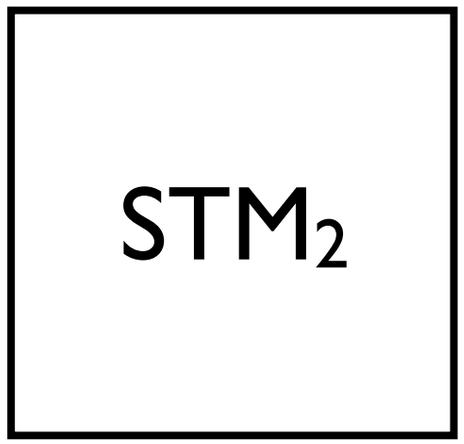
Problem 1




```
tx_start
tx_commit
tx_read
tx_write
```

```
alloc(size)
free(ptr)
```

Problem 1




```
tx_start
tx_commit
tx_read
tx_write
tx_alloc
tx_free
```

Solution

- **Rollback** `alloc` **and** `free` **on abort**
- `free` **rolls back** `alloc`
  - maintain a list of allocated memory
  - on rollback free everything
- Cannot rollback `free`
  - maintain a list of deallocated memory
  - on commit free everything

Algorithm

Algorithm

```
1: tx_start()
2:    tx_start_old()
3:    _alloc_log = {}
4:    _dealloc_log = {}
```

Algorithm

```
1: tx_alloc(size)
2:    ptr = alloc(size)
3:    add_alloc_log(ptr)
4:    return ptr
```

Algorithm

```
1: tx_free(ptr)
2:    add_dealloc_log(ptr)
```

Algorithm

```
1: tx_commit()
2:    tx_commit_old()
3:    for ptr in _dealloc_log
4:       free(ptr)
```

Algorithm

```
1: rollback()
2:    rollback_old_no_jmp()
3:    for ptr in _alloc_log
4:       free(ptr)
5:    rollback_jmp()
```

Assumptions

- STM implementation cannot access deallocated data
  - true for visible reads
  - not true for invisible reads (SwissTM)

SwissTM read

```
...
4:    version := read(r_lock)
5:    while true
6:        if version = 0x1
7:            version := read(r_lock)
8:            continue
9:        value := read(addr)
10:       version2 := read(r_lock)
11:       if version = version2 break
12:       version := version2

                ...
```

Solution

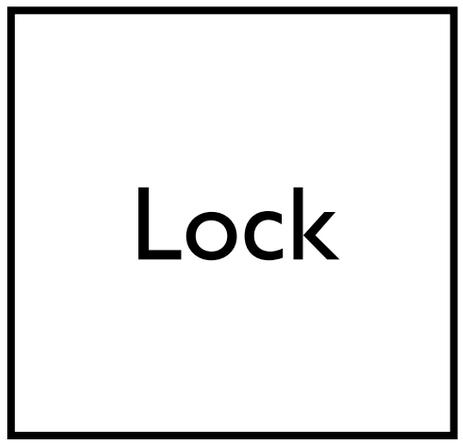
- Postpone deallocation further
  - until all live transactions have observed changes by deallocating transaction
- SwissTM
  - use commit counter

Algorithm

```
1:  tx_commit()
2:     ts = tx_commit_old()
3:     for ptr in _dealloc_log
4:        add dealloc old log(ptr,ts)
5:     for entry in _dealloc_old_log
6:        for tx in concurrent_tx
7:           if tx._valid_ts < entry.ts
8:              continue
9:        free(entry.ptr)
10:       remove_dealloc_old_log(entry)
```

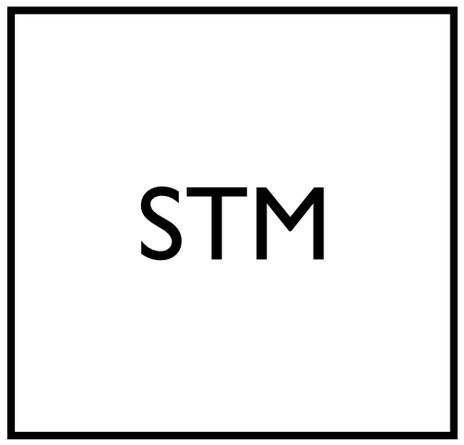
Problem II

Problem II



acquire
release

Problem II




```
tx_start
tx_commit
tx_read
tx_write
```

Solution

- Acquire lock on start
- Release lock on commit

Algorithm

Algorithm

```
1: tx_start()
2:    Lock.acquire()
```

Algorithm

```
1: tx_commit()
2:    Lock.release()
```

Algorithm

```
1: tx_read(addr)
2:     return read(addr)
```

Algorithm

```
1: tx_write(addr,val)
2:    write(addr,val)
```

Think about this

How to efficiently implement `tx_alloc` and `tx_free` in this implementation?

Questions?