# 236755
# Distributed Algorithms

Winter 2019-20

Hagit Attiya

---

# Distributed Systems

- Are everywhere
  - share resources
  - communicate
  - increase performance (replication, speed, fault tolerance)
- Are characterized by
  - independent activities (concurrency)
  - loosely coupled parallelism (heterogeneity)
  - inherent uncertainty
  - need for synchronization

# Example I: Coordinated Clubbing

Coordinate meeting in a club by texting

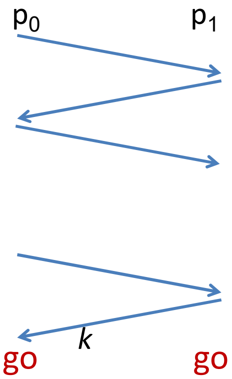
- Only one club & one time to go

It is absolutely bad if only one party shows up

**Theorem: If message delivery is not guaranteed, then coordinated clubbing cannot be achieved**

---

# Example I: Coordinated Clubbing



Ping-pong execution w/o message loss
k smallest number of messages s.t. some participant, e.g., $p_0$, decides go
Agreement ⇨ $p_1$ also decides go

Remove last message, from $p_1$ to $p_0$
$p_1$ still decides go

☞Execution with k-1 messages!

**Theorem: If message delivery is not guaranteed, then coordinated clubbing cannot be achieved**

# Uncertainty in Distributed Systems

- differing process speeds
- varying communication delays
- (partial) failures

☞To ensure that a system is still correct
- identify fundamental problems and state them precisely
- design algorithms to solve these problems and prove the correctness of these algorithms
- analyze their complexity (e.g., time, space, messages)
- prove impossibility results and lower bounds

# Applications of Distributed Computing

Classic problems come from:

multi-threaded operating systems

communication networks

multicore processors

replicated servers

(distributed) database systems

software fault-tolerance

# Example II: Online Accounts

A (single) account on a multi-processor

Two operations:

**Deposit** (one $), **Withdraw** (one $)

Simple implementation by reads and writes does not work

```
lval = read(balance)
lval++
write(balance,lval)
```

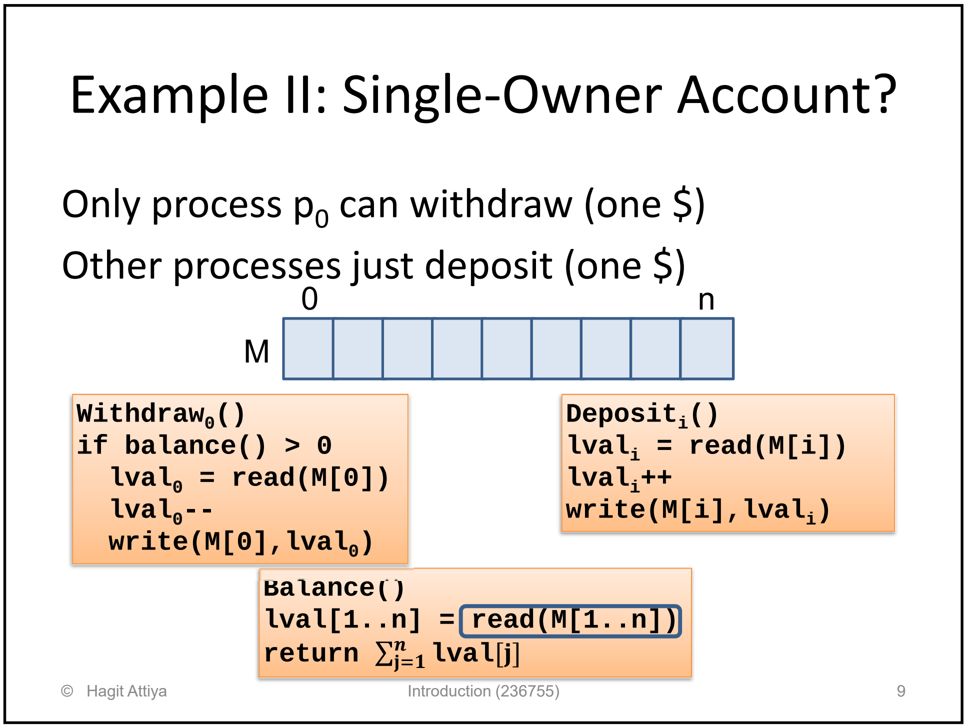
# Example II (Cont.)

```
process A                 process B
lval = read(balance)
lval++
                           lval = read(balance)
                           lval++
write(balance,lval)
                           write(balance,lval)
```

One $ added despite two deposits

Need stronger primitives

```
lval = read(balance)
lval++
write(balance,lval)
```



# Example II: Single-Owner Account?

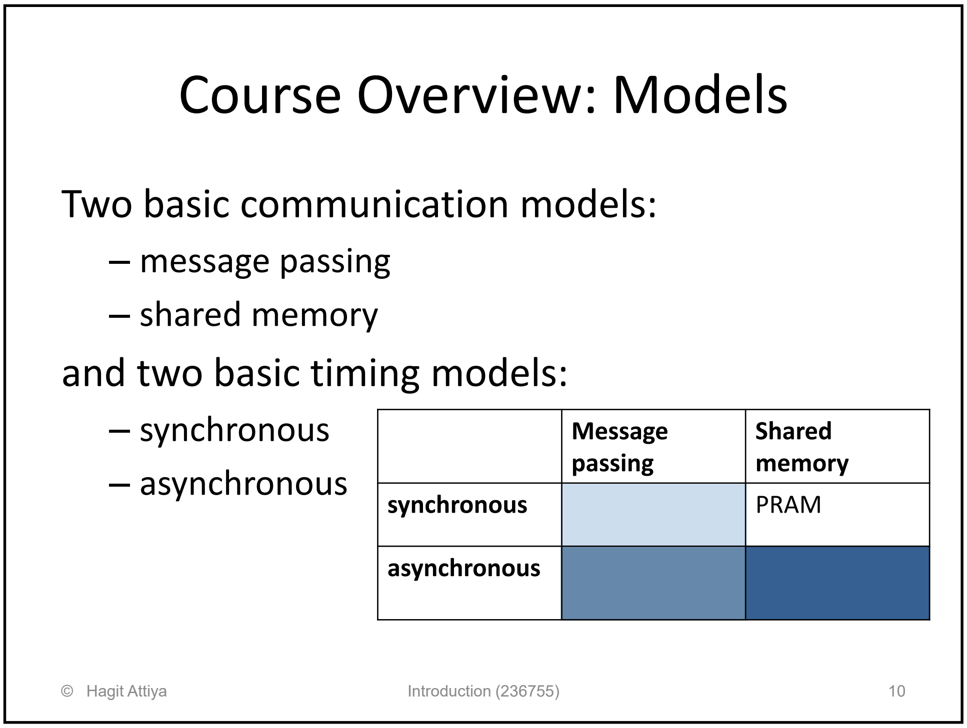
Only process $p_0$ can withdraw (one $)

Other processes just deposit (one $)

```
Withdraw_0()
if balance() > 0
  lval_0 = read(M[0])
  lval_0--
  write(M[0],lval_0)
```

```
Deposit_i()
lval_i = read(M[i])
lval_i++
write(M[i],lval_i)
```

```
Balance()
lval[1..n] = read(M[1..n])
return $\sum_{j=1}^{n}$ lval[j]
```



# Course Overview: Models

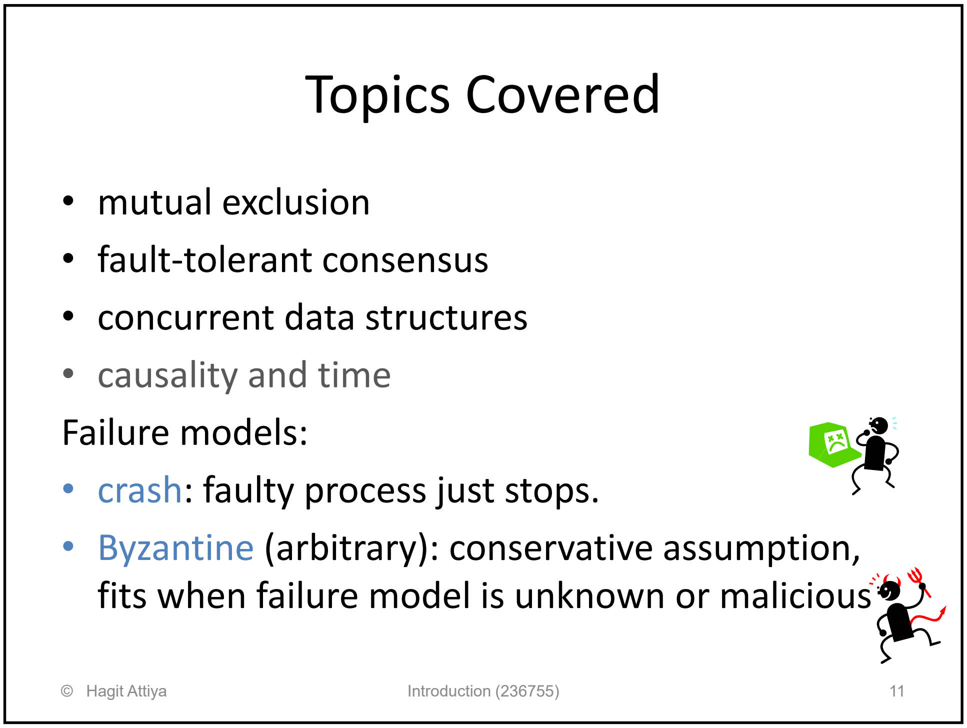
Two basic communication models:

- message passing
- shared memory

and two basic timing models:

- synchronous
- asynchronous

| | Message passing | Shared memory |
|---|---|---|
| **synchronous** | 🟦 | PRAM |
| **asynchronous** | 🟦 | 🟦 |



# Topics Covered

- mutual exclusion
- fault-tolerant consensus
- concurrent data structures
- causality and time

Failure models:

- crash: faulty process just stops.
- Byzantine (arbitrary): conservative assumption, fits when failure model is unknown or malicious