CS 334
lecture 22
Locking Problem.

0) Deadlock - No Thread can make progress.

1) Missing lock operations. 
   => race conditions

2) Atomicity Violation.
Atomic Transactions

- runtime system enforces atomicity.

- Implementation

  1. Pessimistic (One Big Lock)
     - works
     - easy to implement.
     - pay on every atomic call.
     - loose concurrency.
     - hard to divide “Big lock” into smaller locks
Optimistic

- log all changes to memory.
- commit at end of block

\[\text{run atomically}\]
- verify no conflicting read or write.
- perform updates.

- Retry on Failure

STM - Software Transactional Memory

HTM - Hardware Transactional Memory
+ more concurrency.
+ pay most at conflicts.
  (infrequently.)

+ small transactions
- large transactions,
- irrevocable ops.

    atomic void f() {
        launchMissle();
        X++;
    }
Go Routines & Channels
Acasters & message passing

1. Send & receiver op.

2. Synchronous vs asynchronous.

3. Guarantees
   - delivery.
   - latency.
   - ordering.
Actors

Object w/

1) local state.
2) send/recv ops
3) create new Actors.

(Erlang)
0. loop
1. react
2. case...
3.
4. p ! message.
5. sender ! message.
6. p !? message
send & wait for response
Reactive Model

\[
\text{Dispetchers} \quad \text{Cores}
\]

\[
\text{# Dispatchers} = \text{# Cores}
\]
A
T
Rec've
model bigger protocols,
- each Actor is a Thread.

Summary
- react: lightweight,
  “single-message” protocols.
- receive: heavier threads for
  richer protocols.