# Lecture 2

- Administration
  - Piazza
  - Assignment Update

- Definition of a Distributed System

# Distributed System Issues

- Concurrency
- Communication
- Synchronization
- Availability
- Performance
- Scalability
- Security
- Mobility

# Concurrency

In computer science, **concurrency** is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other. The computations may be executing on multiple cores in the same chip, preemptively time-shared threads on the same processor, or executed on physically separated processors. A number of mathematical models have been developed for general concurrent computation including Petri nets, process calculi, the Parallel Random Access Machine model, the Actor model and the Reo Coordination Language.


**Concurrent computing** is a form of computing in which programs are designed as collections of interacting computational processes that *may* be executed in parallel.[1] Concurrent programs (processes or threads) can be executed on a single processor by interleaving the execution steps of each in a time-slicing way, or can be executed in parallel by assigning each computational process to one of a set of processors that may be close or distributed across a network. The main challenges in designing concurrent programs are ensuring the correct sequencing of the interactions or communications between different computational executions, and coordinating access to resources that are shared among executions.[1]

# New problems

# deadlock

A **deadlock** is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.

# Livelock

A **livelock** is similar to a deadlock, except that the states of the processes involved in the livelock constantly change with regard to one another, none progressing.

Livelock is a special case of resource starvation; the general definition only states that a specific process is not progressing.[12]

A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time.

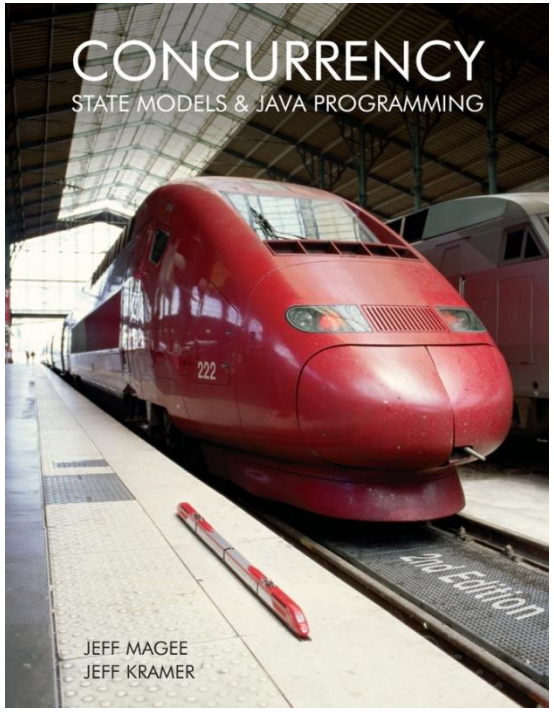
# Race Conditions

A **race condition** or **race** hazard is the behavior of an electronic or software system where the output is dependent on the sequence or timing of other uncontrollable events. It becomes a bug when events do not happen in the order the programmer intended. The term originates with the idea of two signals racing each other to influence the output first.

In computer science, a **nondeterministic algorithm** is an algorithm that can exhibit different behaviors on different runs, as opposed to a deterministic algorithm. There are several ways an algorithm may behave differently from run to run. A concurrent algorithm can perform differently on different runs due to a race condition.

# Fairness

# Concurrency – formal introduction



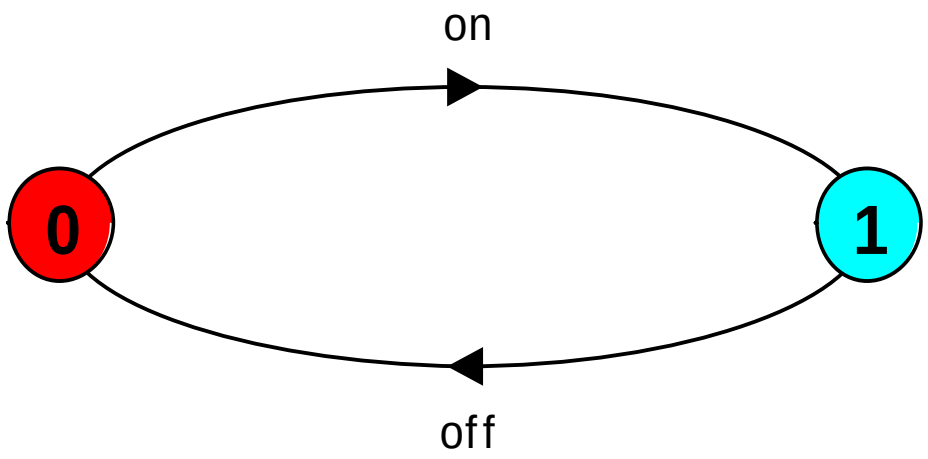

Concurrency: State Models and Java Programs
2nd Edition
By Jeff Kramer and Jeff Magee

http://www.wileyeurope.com/college/magee

Slides from Kramer/Magee

# modeling processes

A process is the execution of a sequential program. It is modeled as a finite state machine which transits from state to state by executing a sequence of atomic actions.



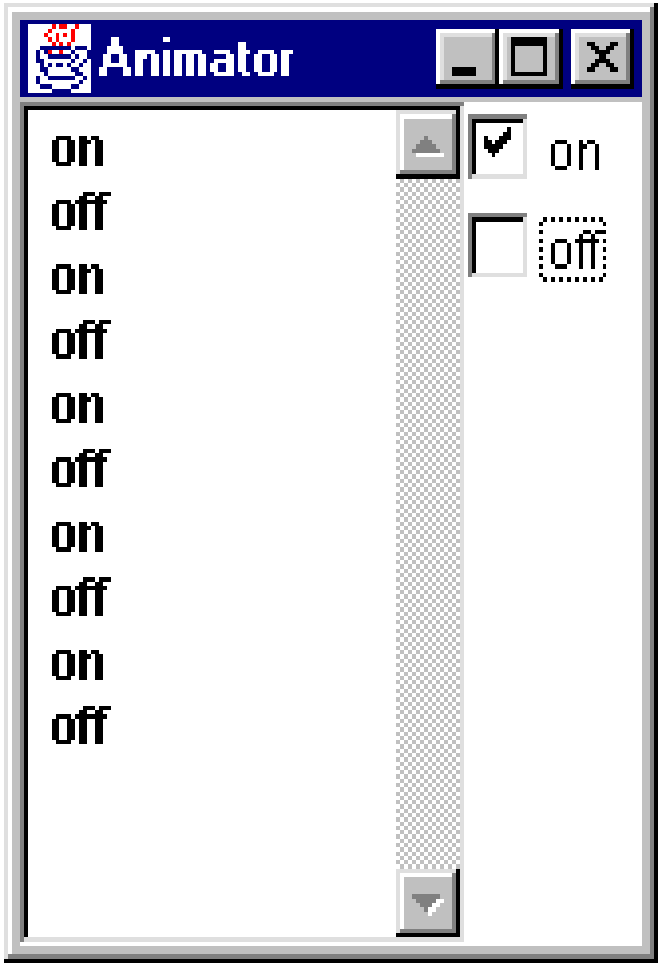

a light switch **LTS**

**on→off→on→off→on→off→ .....**

a sequence of actions
or *trace*

*Can finite state models produce infinite traces?*

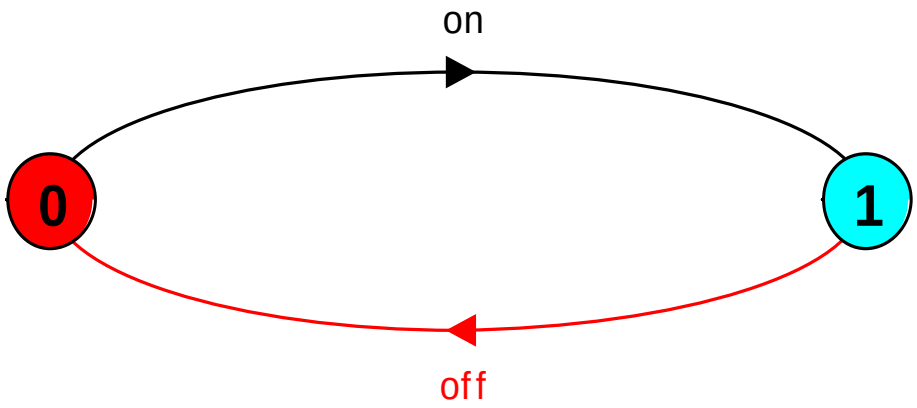
# animation using LTSA




The *LTSA* animator can be used to produce a trace.

Ticked actions are eligible for selection.

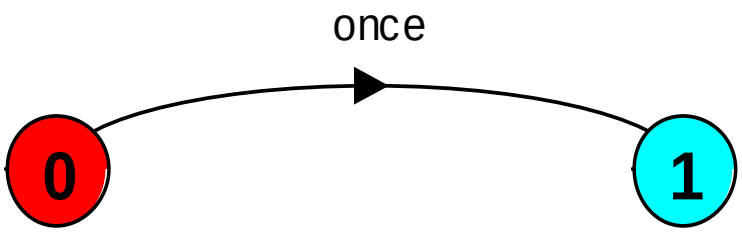
In the LTS, the last action is highlighted in red.



# FSP - action prefix

If **x** is an action and **P** a process then **(x-> P)** describes a process that initially engages in the action **x** and then behaves exactly as described by **P**.

```
ONESHOT = (once -> STOP).
```




ONESHOT state machine

(terminating process)

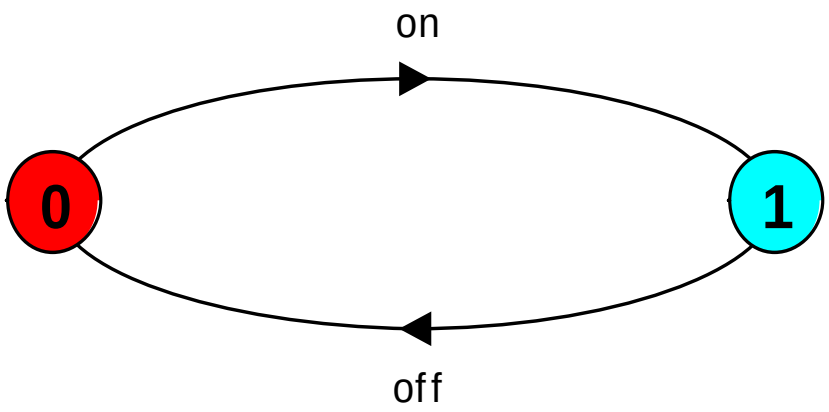
Convention: actions begin with lowercase letters

PROCESSES begin with uppercase letters

# FSP - action prefix & recursion

Repetitive behaviour uses recursion:

```
SWITCH = OFF,
OFF    = (on -> ON),
ON     = (off-> OFF).
```




Substituting to get a more succinct definition:

```
SWITCH = OFF,
OFF    = (on ->(off->OFF)).
```

And again:

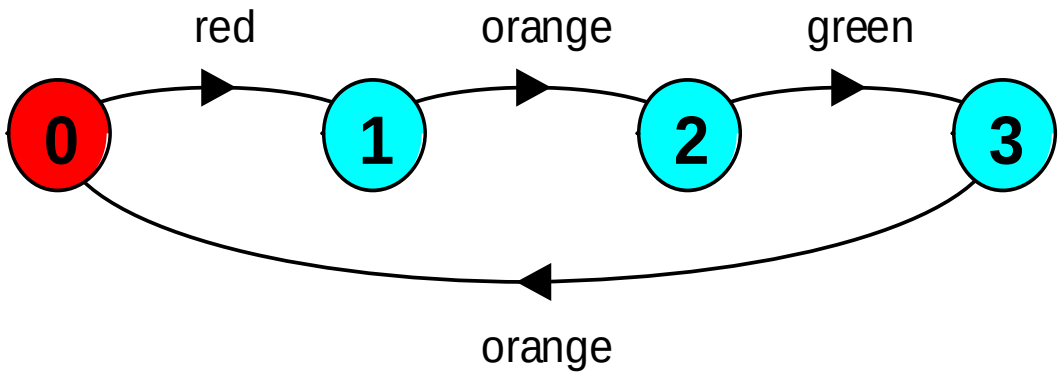
```
SWITCH = (on->off->SWITCH).
```

# FSP - action prefix

FSP model of a traffic light :

```
TRAFFICLIGHT = (red->orange->green->orange
                -> TRAFFICLIGHT).
```

LTS generated using *LTSA*:




Trace:

**red→orange→green→orange→red→orange→green** …

# FSP - choice

If **x** and **y** are actions then **(x-> P | y-> Q)** describes a process which initially engages in either of the actions **x** or **y**. After the first action has occurred, the subsequent behavior is described by **P** if the first action was **x** and **Q** if the first action was **y**.

*Who or what makes the choice?*

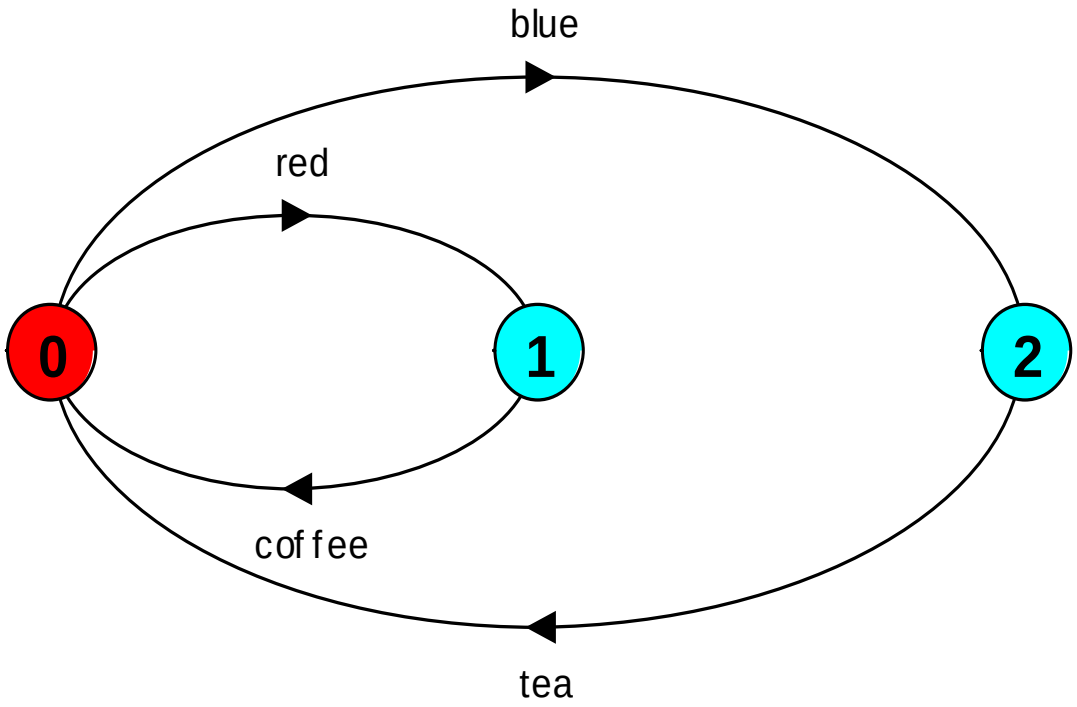
*Is there a difference between input and output actions?*

# FSP - choice

FSP model of a drinks machine :

```
DRINKS = (red->coffee->DRINKS
         |blue->tea->DRINKS
         ).
```

LTS generated using *LTSA*:



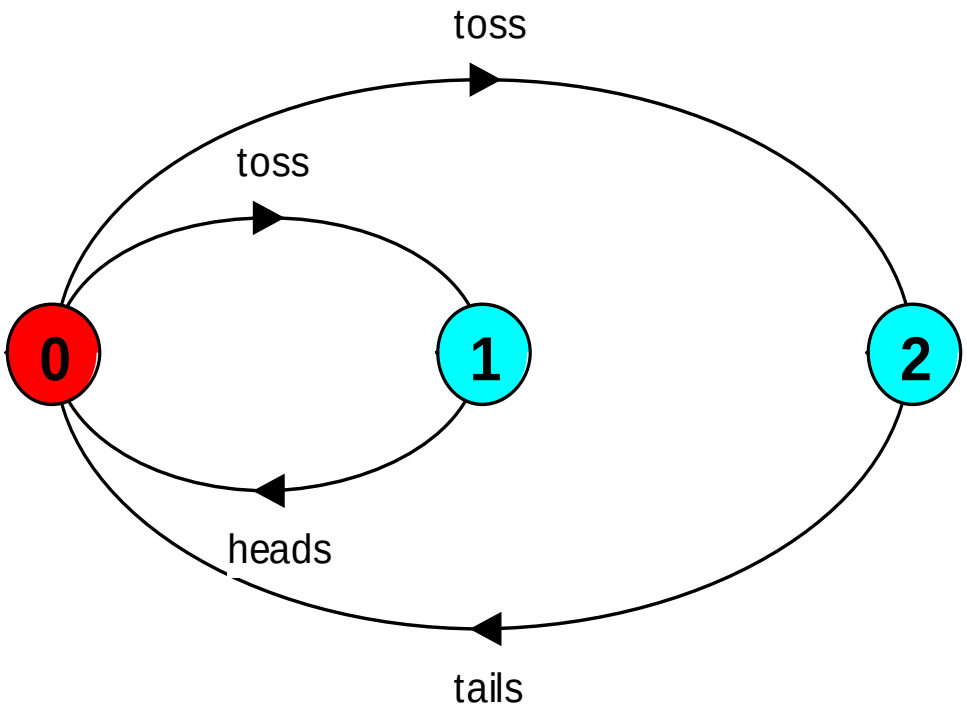

Possible traces?

# Non-deterministic choice

Process (x-> P | x -> Q) describes a process which engages in **x** and then behaves as either P or Q.

```
COIN = (toss->HEADS|toss->TAILS),
HEADS= (heads->COIN),
TAILS= (tails->COIN).
```

**Tossing a coin.**




Possible traces?