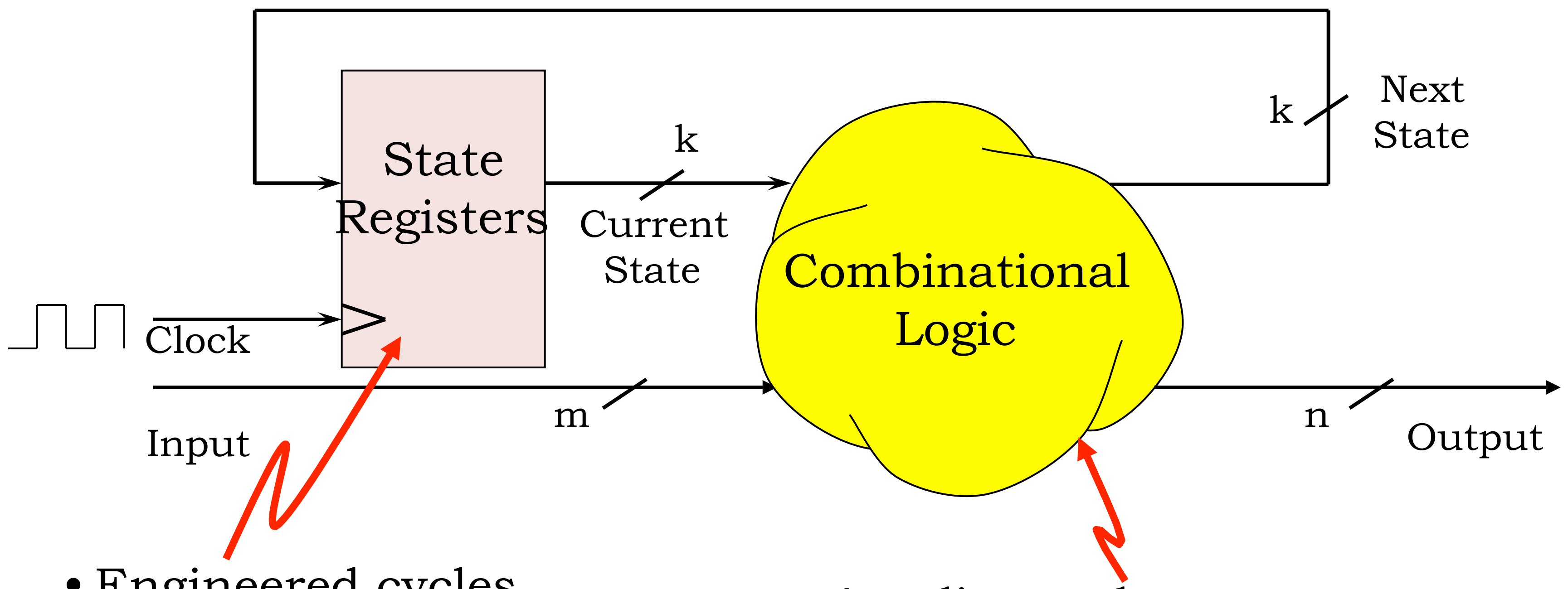
6. Finite State Machines

6.004x Computation Structures
Part 1 – Digital Circuits

Our New Machine

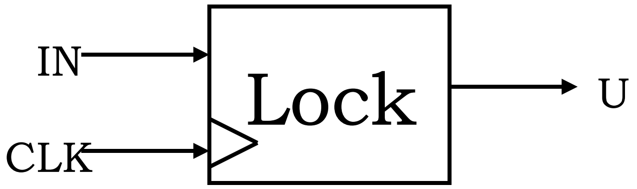


- Engineered cycles
- Works only if dynamic discipline obeyed
- Remembers k bits for a total of 2^k unique combinations

- Acyclic graph
- Obeys static discipline
- Can be exhaustively enumerated by a truth table of 2^{k+m} rows and k+n output columns

A Simple Sequential Circuit…

Lets make a digital binary *Combination Lock:*

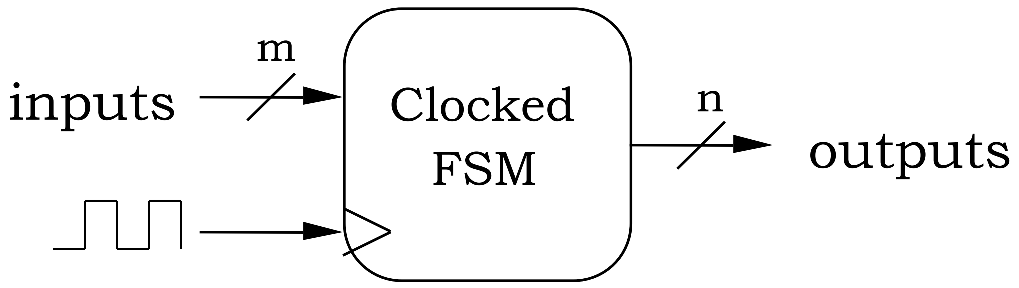


Specification:

- One input ("0" or "1")
- One output ("Unlock" signal)
- UNLOCK is 1 if and only if:

  Last 4 inputs were the "combination": 0110

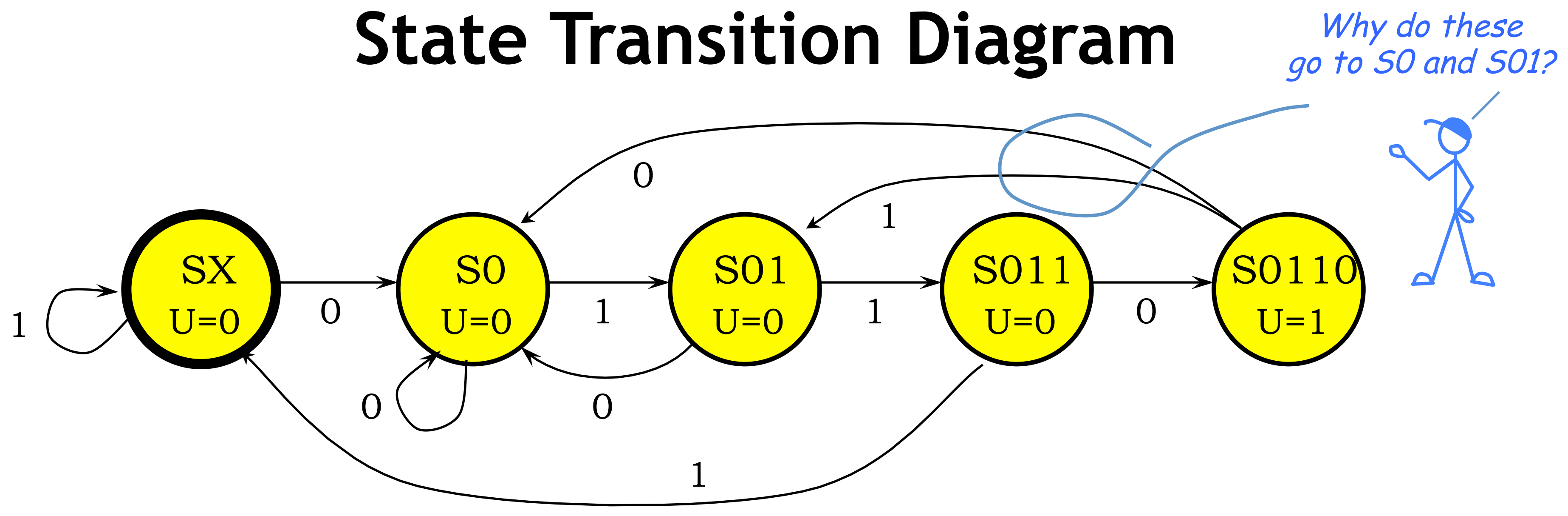
Abstraction *du jour*: Finite State Machines



A FINITE STATE MACHINE has

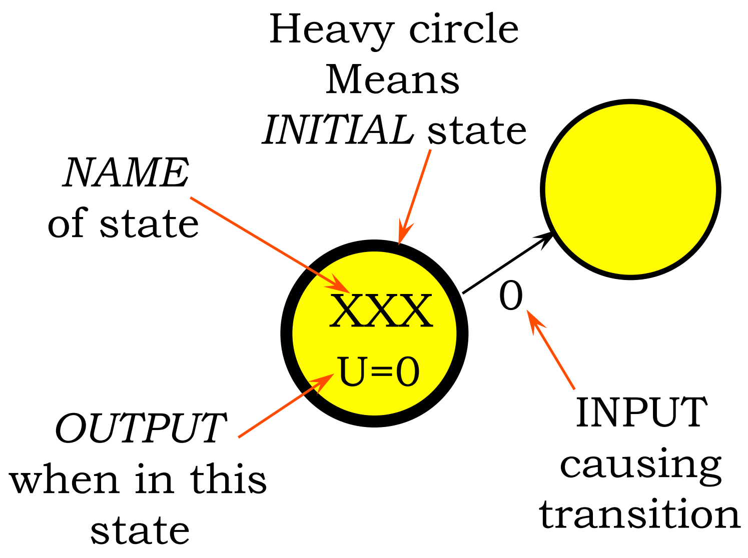
- k STATES: S_1 … S_k (one is "initial" state)
- m INPUTS: I_1 … I_m
- n OUTPUTS: O_1 … O_n
- Transition Rules: s'(s, I) for each state s and input I
- Output Rules: Out(s) or Out(s, I) for each state s and input I

State Transition Diagram

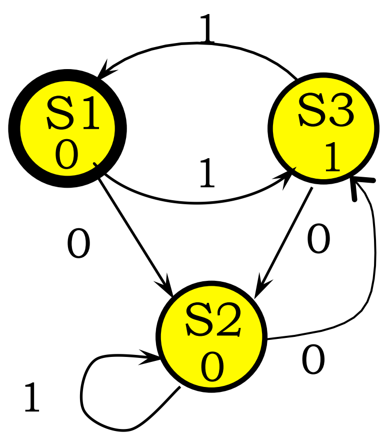


Designing our lock …

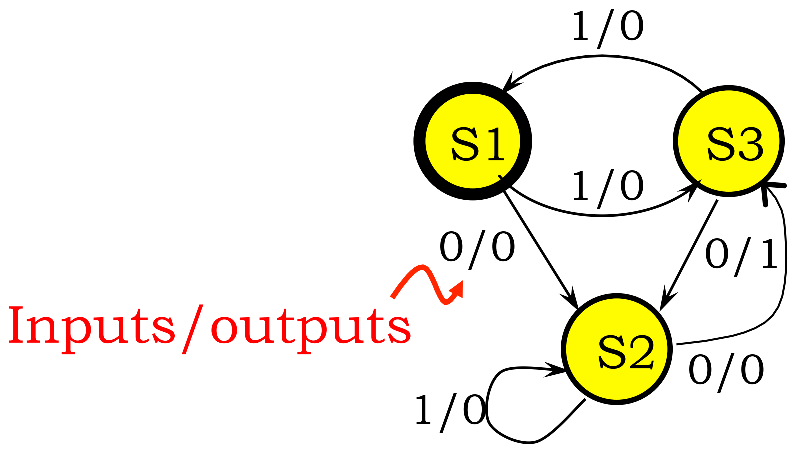
- Need an initial state; call it SX.
- Must have a separate state for each step of the proper entry sequence
- Must handle other (erroneous) entries


Heavy circle Means *INITIAL* state
NAME of state
XXX
U=0
0
OUTPUT when in this state
INPUT causing transition

Valid State Diagrams



MOORE Machine:
Outputs on States



MEALY Machine:
Outputs on Transitions

Arcs leaving a state must be:

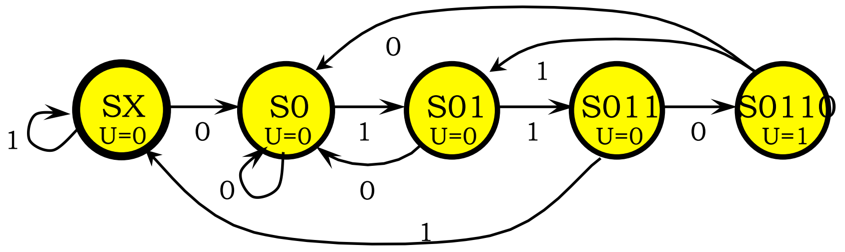
(1) mutually exclusive

– *can't have two choices for a given input value*

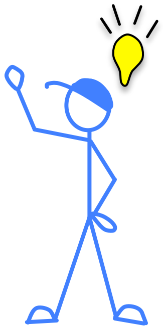
(2) collectively exhaustive

– *every state must specify what happens for each possible input combination. "Nothing happens" means arc back to itself.*

State Transition Diagram as a Truth Table



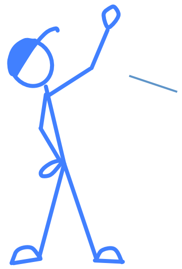
IN		Current State	Next State		Unlock
0	000	SX	S0	001	0
1	000	SX	SX	000	0
0	001	S0	S0	001	0
1	001	S0	S01	011	0
0	011	S01	S0	001	0
1	011	S01	S011	010	0
0	010	S011	S0110	100	0
1	010	S011	SX	000	0
0	100	S0110	S0	001	1
1	100	S0110	S01	011	1



All state transition diagrams can be described by truth tables…

Binary encodings are assigned to each state (a bit of an art)

The truth table can then be simplified using the reduction techniques we learned for combinational logic

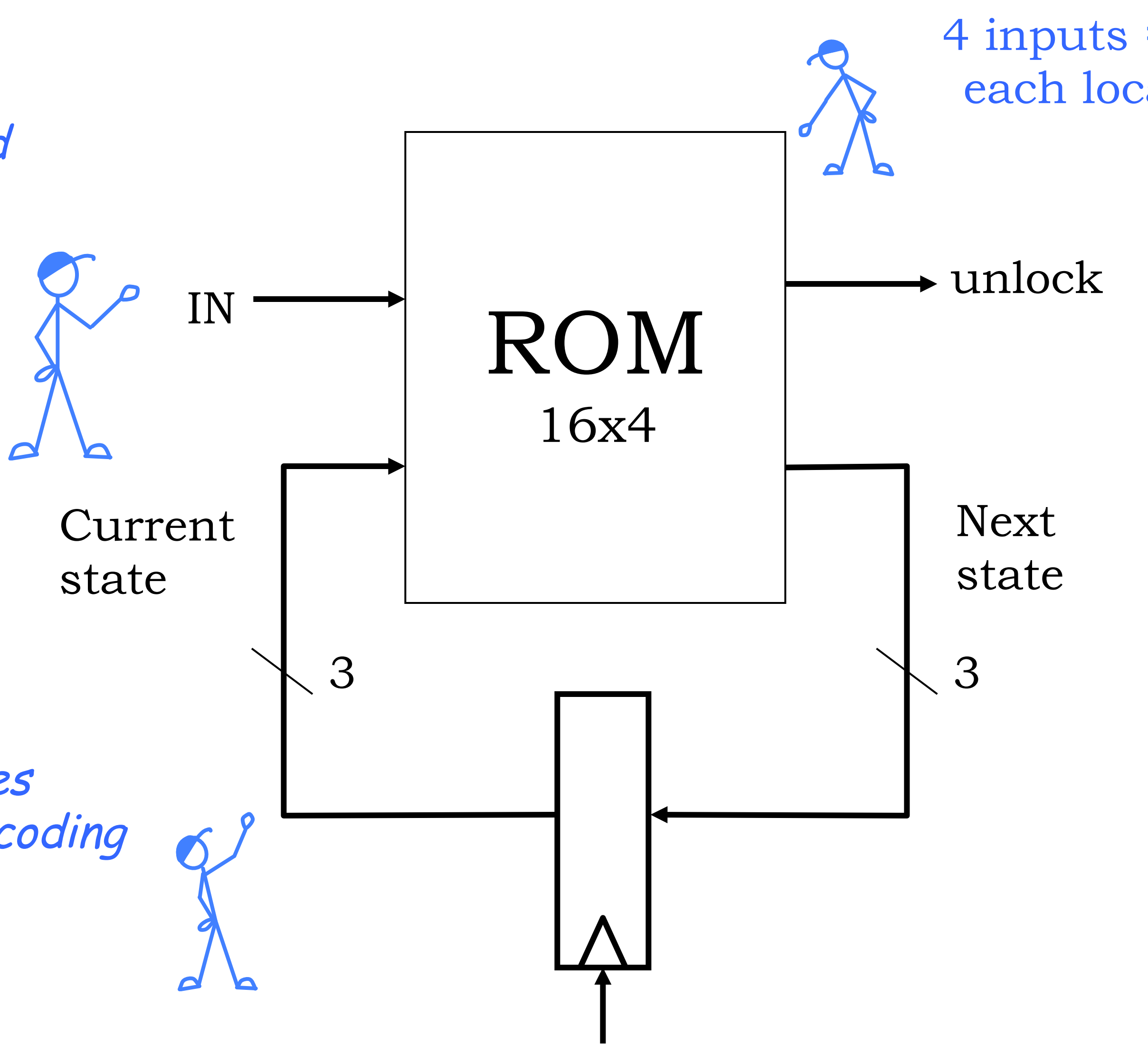


The assignment of codes to states can be arbitrary, however, if you choose them carefully you can greatly reduce your logic requirements.

Now Put It In Hardware!

We assume inputs are synchronized with clock...

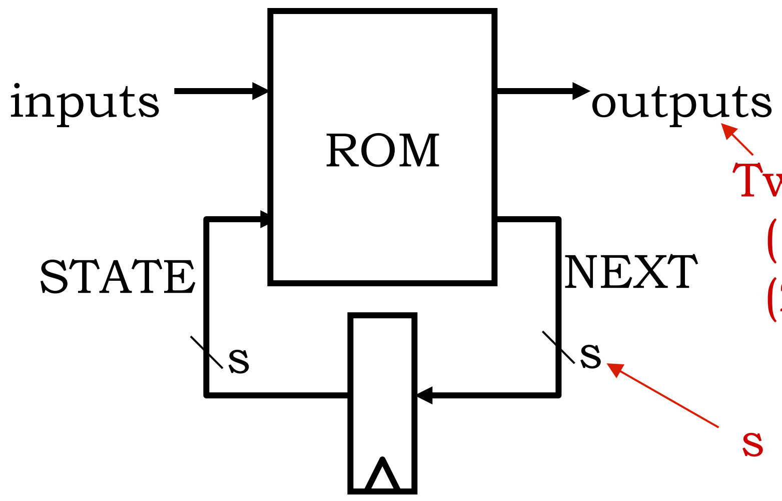
4 inputs ⇒ 2^4 locations, each location supplies 4 bits



5 states ⇒ 3-bit encoding

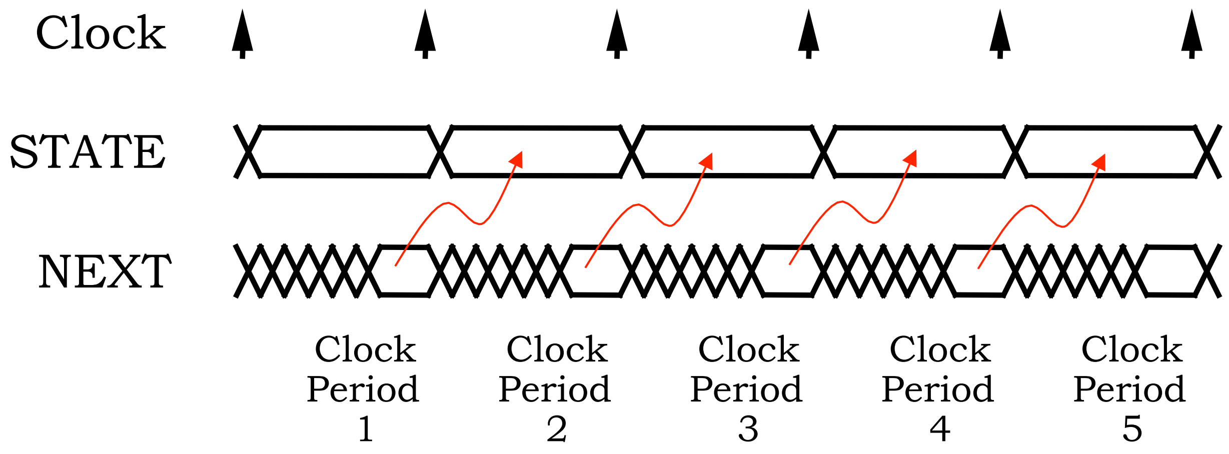
Trigger update periodically ("clock")

Discrete State, Discrete Time

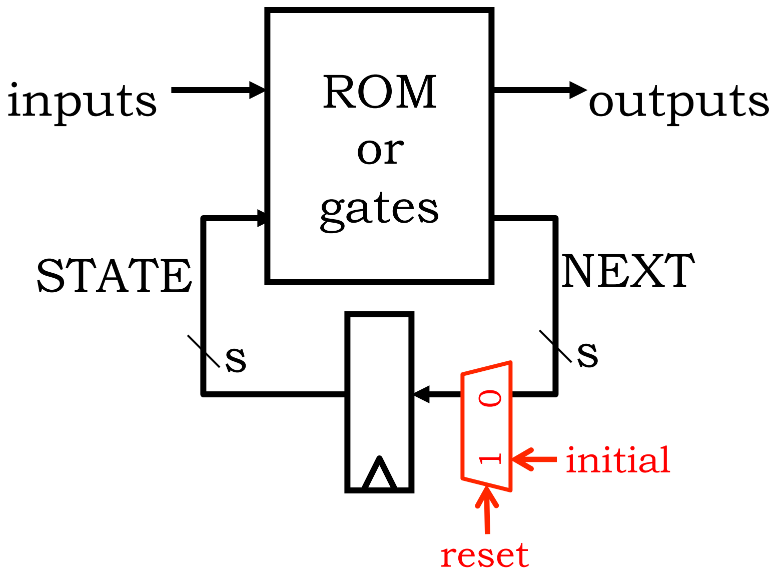


Two design choices:
(1) outputs only depend on state (Moore)
(2) outputs depend on inputs + state (Mealy)

s state bits ⇒ 2^s possible states



Housekeeping Issues…

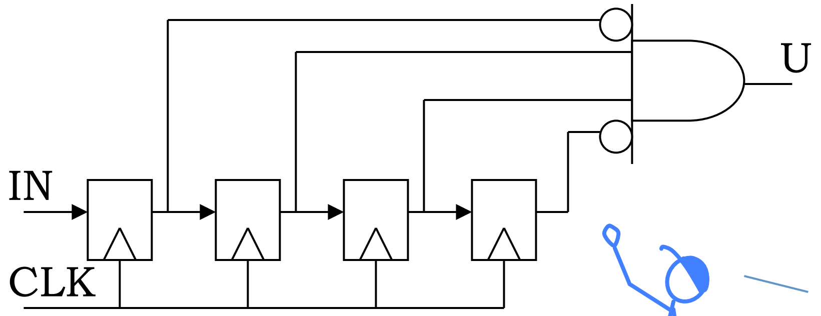


1. Initialization? Clear the memory?

2. Unused state encodings?
   - waste ROM (use gates)
   - what does it mean?
   - can the FSM recover?

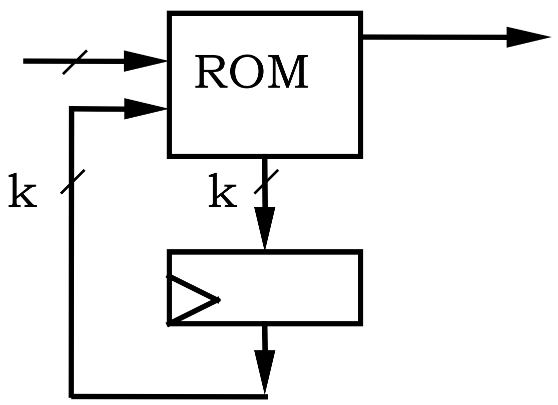
3. Choosing encoding for state?

4. Synchronizing input changes with state update?

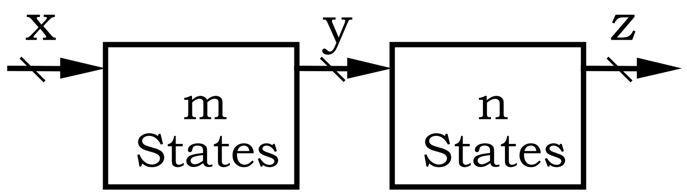


FSM States

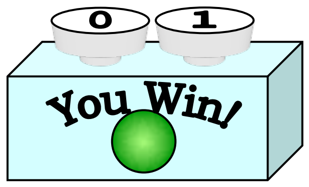
1. What can you say about the number of states?



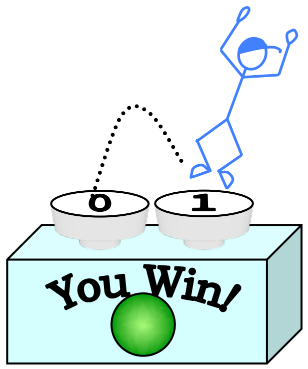
2. Same question:



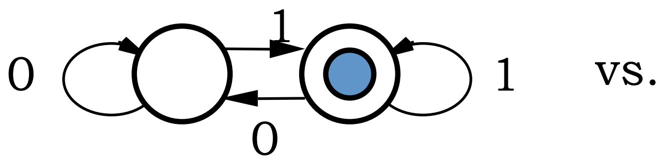
3. Here's an FSM. Can you discover its rules?



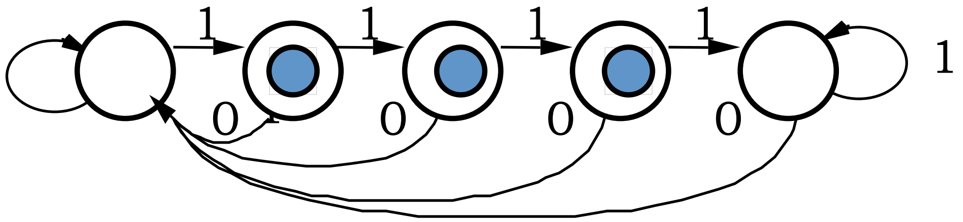
What's My Transition Diagram?



0=OFF,
1=ON?



"1111" = 0
Surprise!

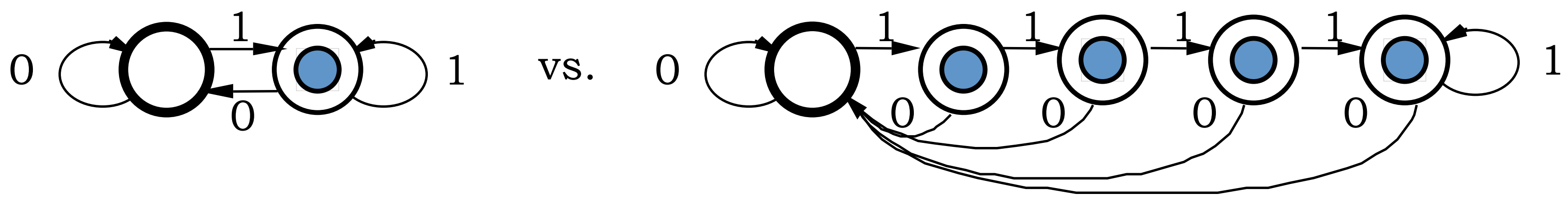


- If you know NOTHING about the FSM, you're never sure!
- If you have a BOUND on the number of states, you can discover its behavior:

  K-state FSM: Every (reachable) state can be reached in < k steps.

BUT ... FSMs may be equivalent!

FSM Equivalence



ARE THEY DIFFERENT?

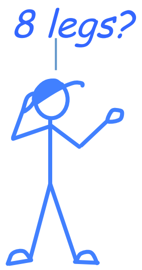
NOT in any practical sense! They are EXTERNALLY INDISTINGUISHABLE, hence interchangeable.

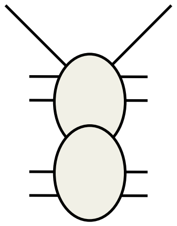
FSMs are *EQUIVALENT* iff every input sequence yields identical output sequences.

ENGINEERING GOAL:

- HAVE an FSM which *works...*
- WANT <u>simplest</u> (ergo cheapest) equivalent FSM.

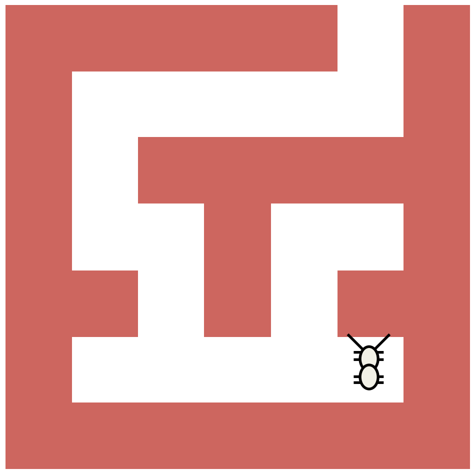
Let’s Build a RoboAnt





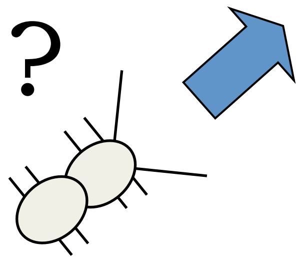
- SENSORS: antennae L and R, each 1 if in contact with something.
- ACTUATORS: Forward Step F, ten-degree turns TL and TR (left, right).

GOAL: Make our ant smart enough to get out of a maze like:

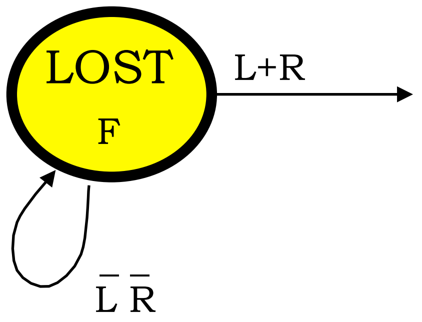


STRATEGY: "Right antenna to the wall"

Lost In Space

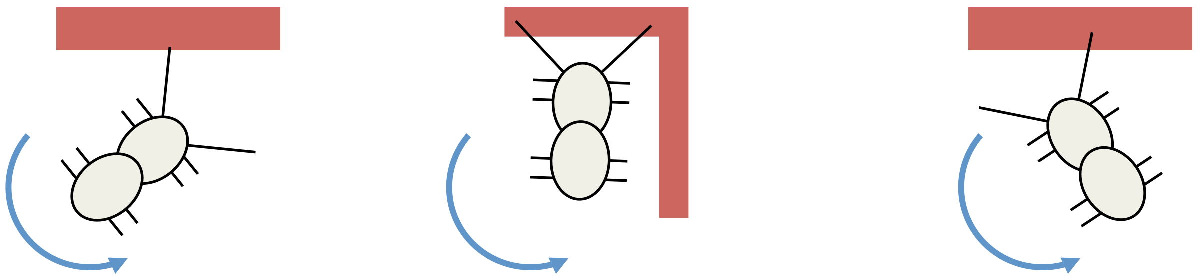


Action: Go forward until we hit something.

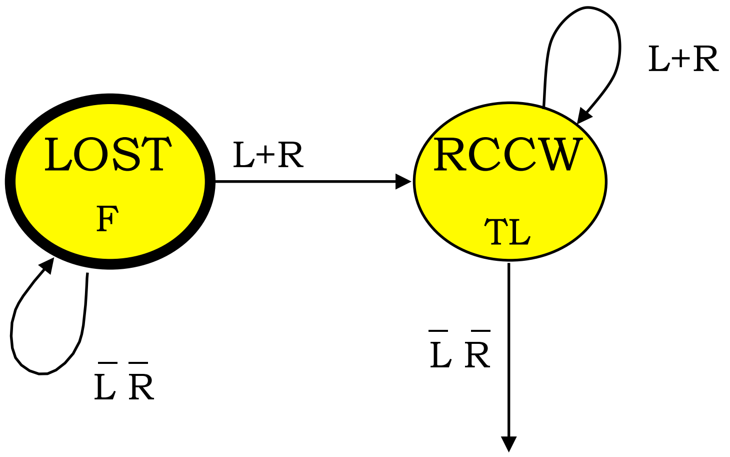


“lost” is the initial state

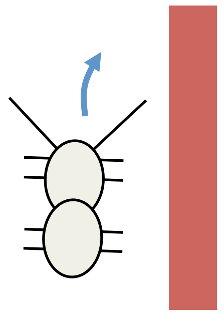
Bonk!



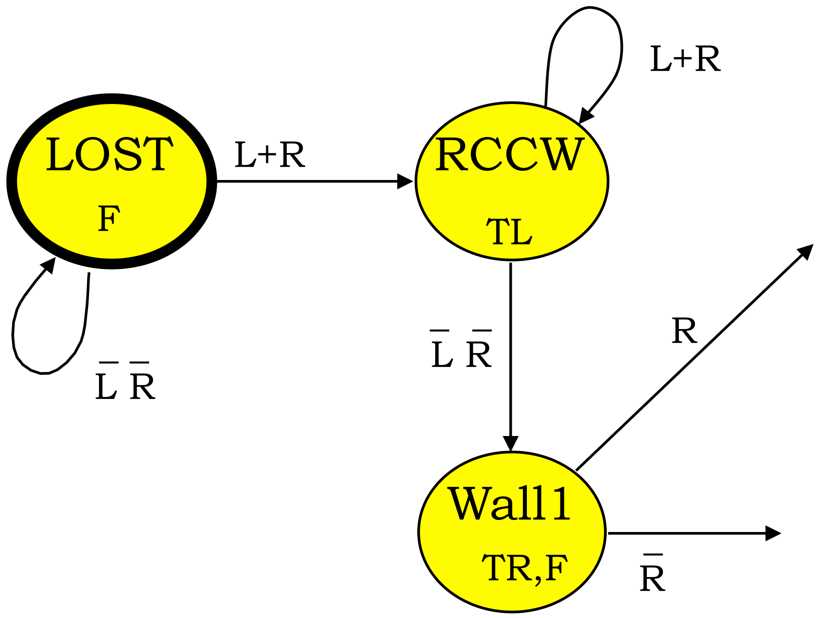
Action: Turn left (CCW) until we don't touch anymore



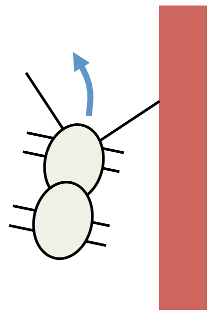
A Little to the Right…



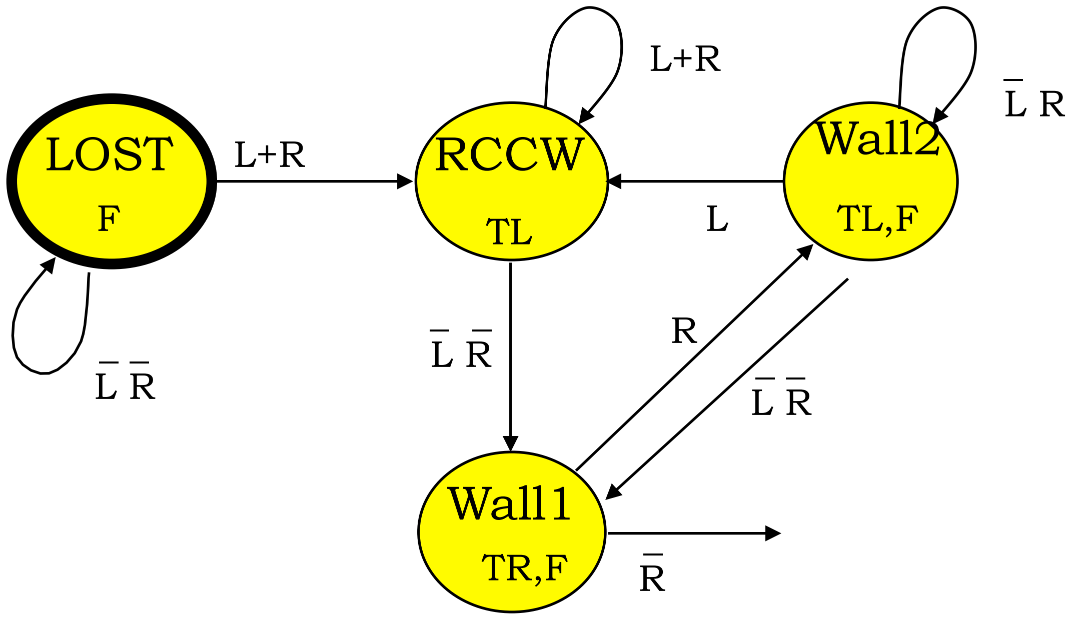
Action: Step and turn right a little, look for wall



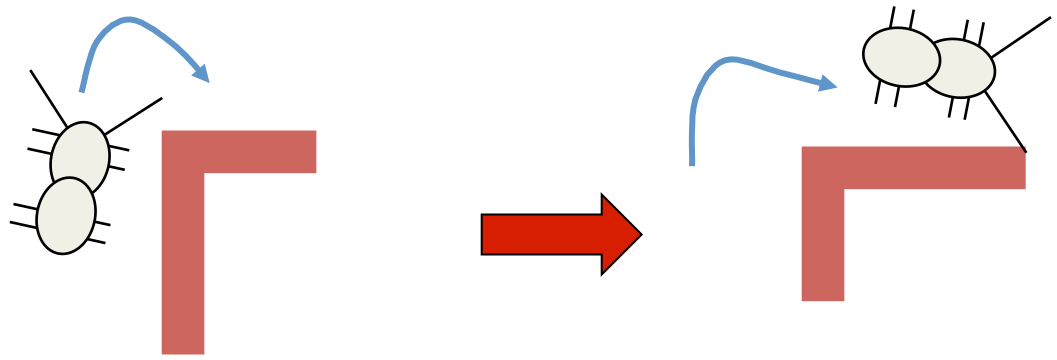
Then a Little to the Left



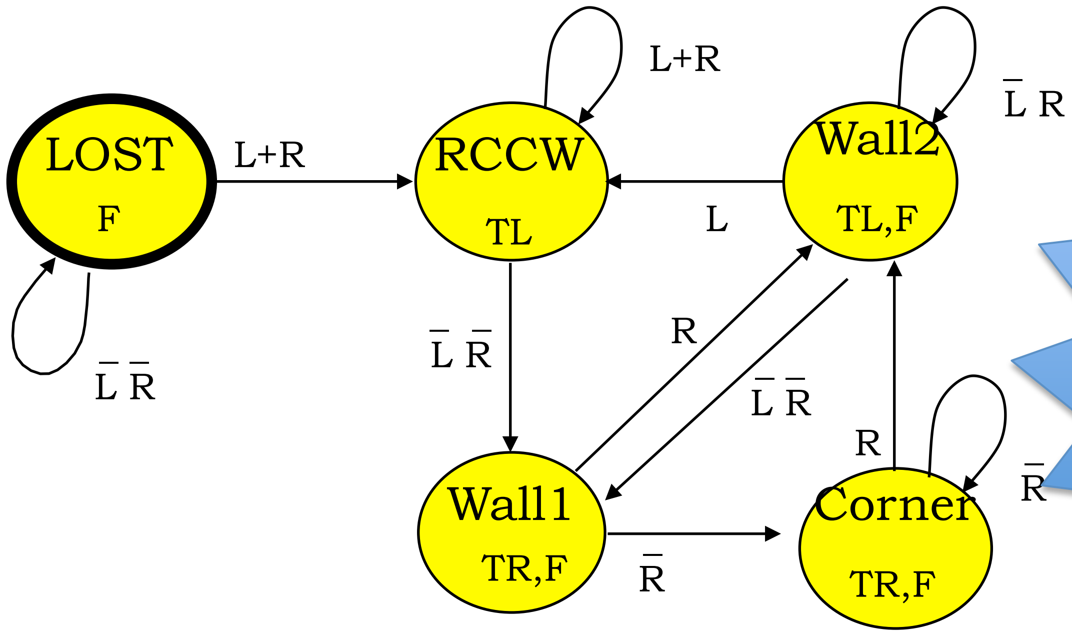
Action: Step and turn left a little, till not touching (again)



Dealing With Outside Corners



Action: Step and turn right until we hit perpendicular wall



Hey, this might even work!

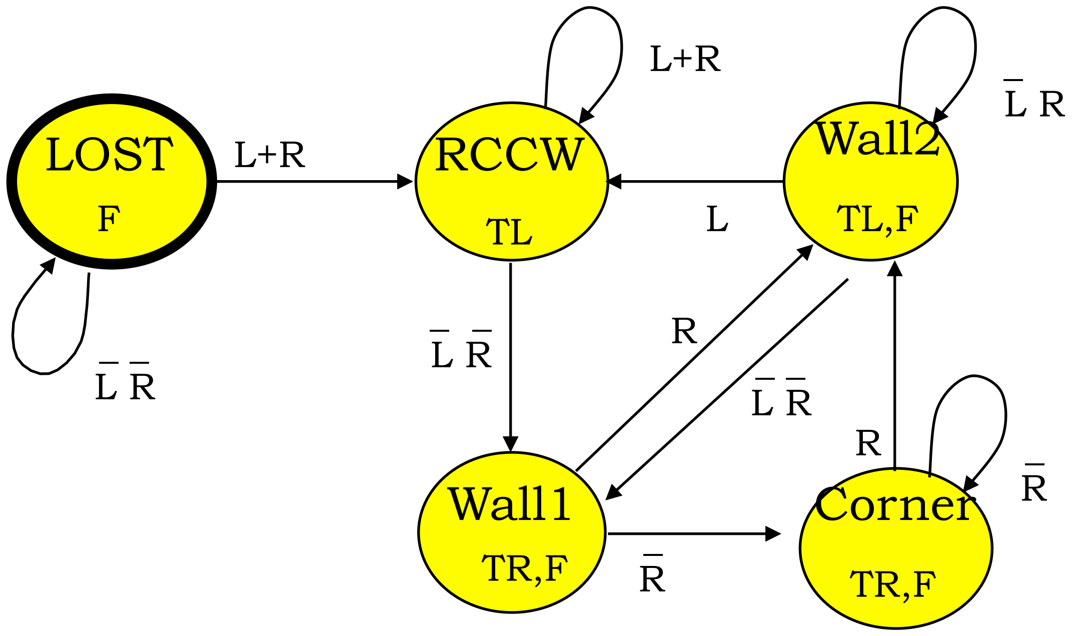
Equivalent State Reduction

Observation: two states are equivalent if

1. Both states have identical outputs; <u>AND</u>
2. Every input ⇒ equivalent states.

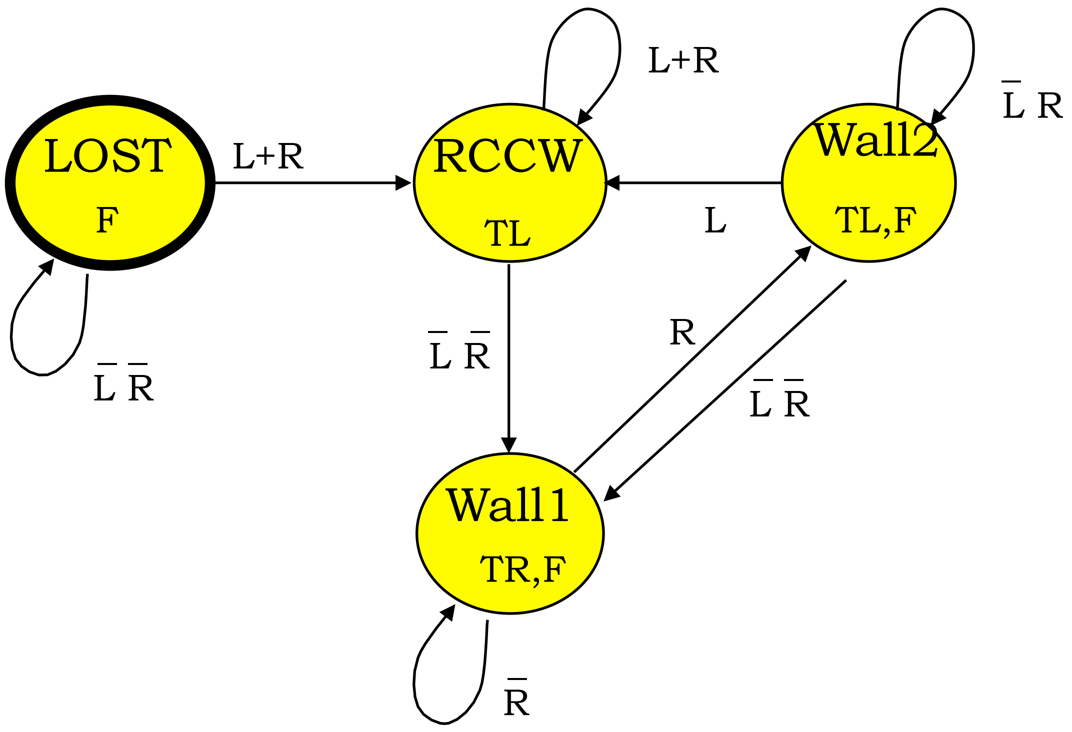
Reduction Strategy:

Find pairs of equivalent states, MERGE them.



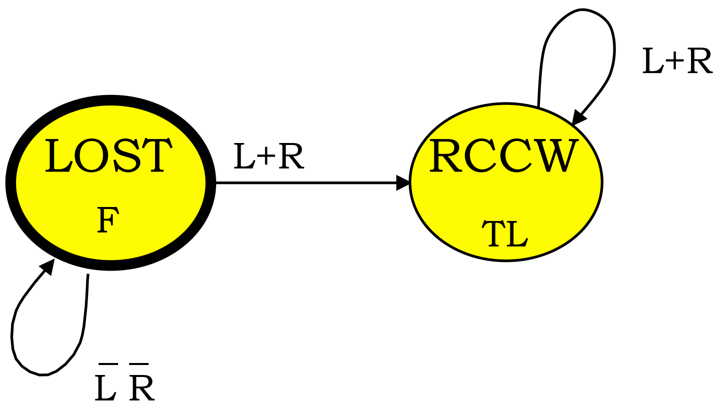
An Evolutionary Step

Merge equivalent states Wall1 and Corner into a single new, combined state.



Behaves exactly as previous (5-state) FSM, but requires <u>half</u> the ROM in its implementation!

Building The Transition Table

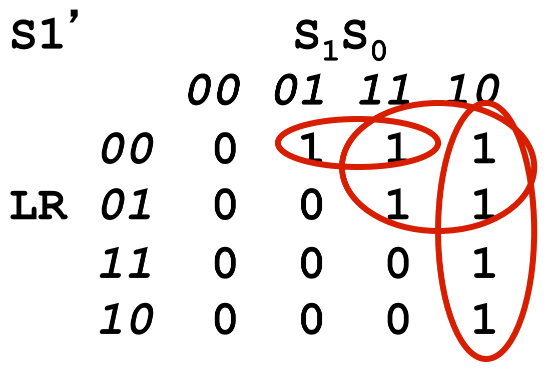


S	L	R	S'	TR	TL	F
00	0	0	00	0	0	1
00	0	1	01	0	0	1
00	1	0	01	0	0	1
00	1	1	01	0	0	1
01	0	1	01	0	1	0
01	1	0	01	0	1	0
01	1	1	01	0	1	0

Implementation Details

	S	L	R	S'	TR	TL	F
	00	0	0	00	0	0	1
LOST	00	1	-	01	0	0	1
	00	-	1	01	0	0	1
	01	1	-	01	0	1	0
RCCW	01	-	1	01	0	1	0
	01	0	0	10	0	1	0
WALL1	10	-	0	10	1	0	1
	10	-	1	11	1	0	1
	11	1	-	01	0	1	1
WALL2	11	0	0	10	0	1	1
	11	0	1	11	0	1	1

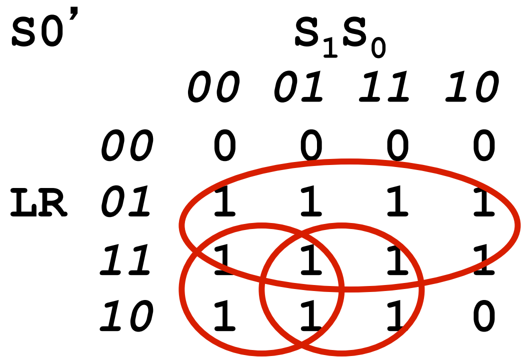
Complete Transition table



S1'

LR \ S_1S_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	1
10	0	0	0	1

$$S_1' = S_1\overline{S_0} + \overline{L}S_1 + \overline{L}\,\overline{R}S_0$$

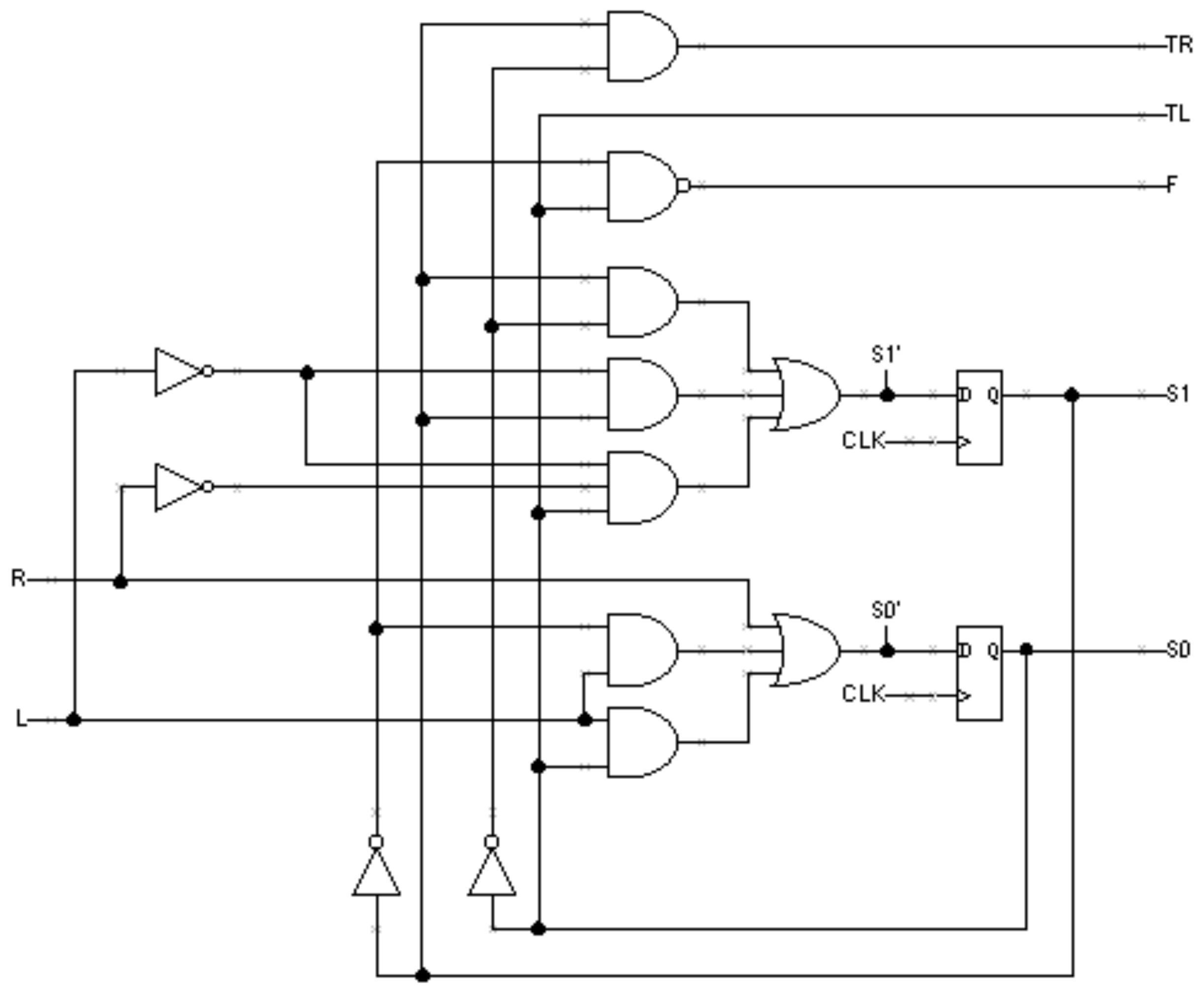


S0'

LR \ S_1S_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	0

$$S_0' = R + L\overline{S_1} + LS_0$$

Ant Schematic



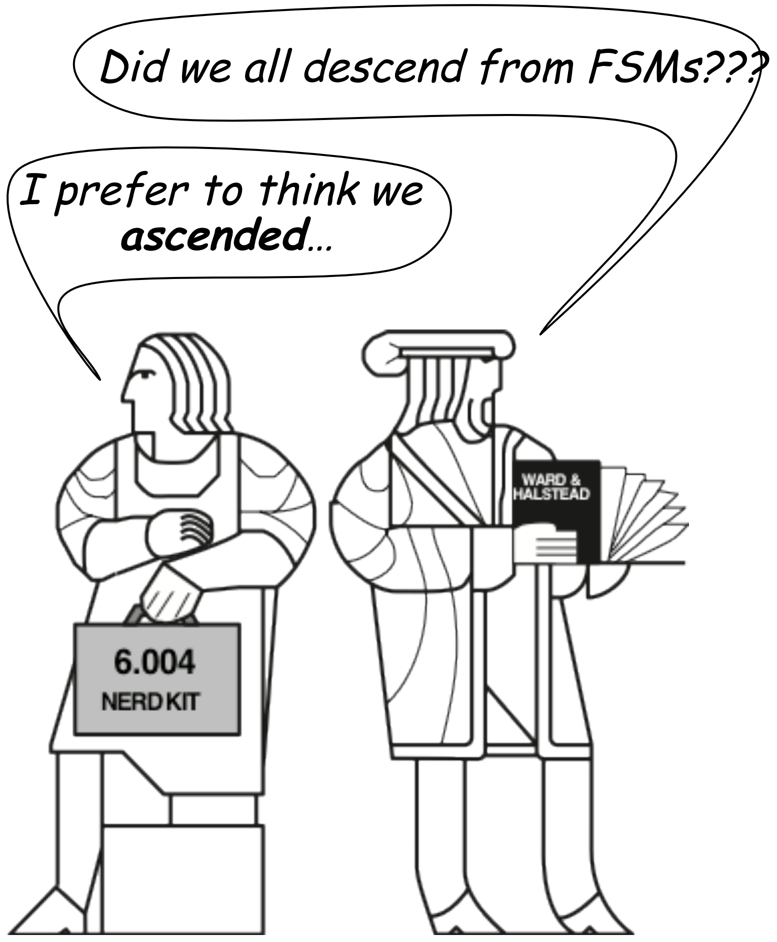
FSMs All the Way Down?

- More than ants:
  Swarming, flocking, and schooling can result from collections of very simple FSMs

- Perhaps most physics:
  Cellular automata, arrays of simple FSMs, can more accurately model fluids than numerical solutions to PDEs

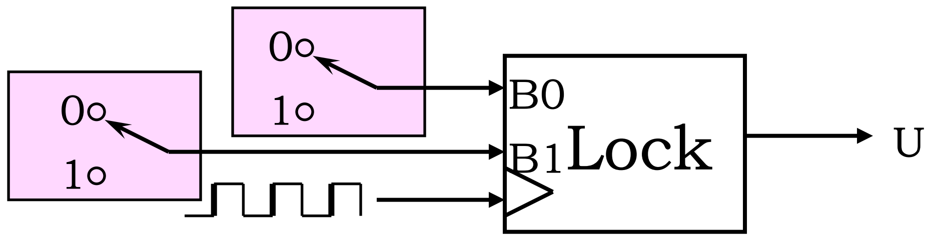
- What if:
  We replaced the ROM with a RAM and have outputs that modify the RAM?

... You'll see FSMs for the rest of your life!


Did we all descend from FSMs???
I prefer to think we **ascended**...
6.004 NERD KIT
WARD & HALSTEAD

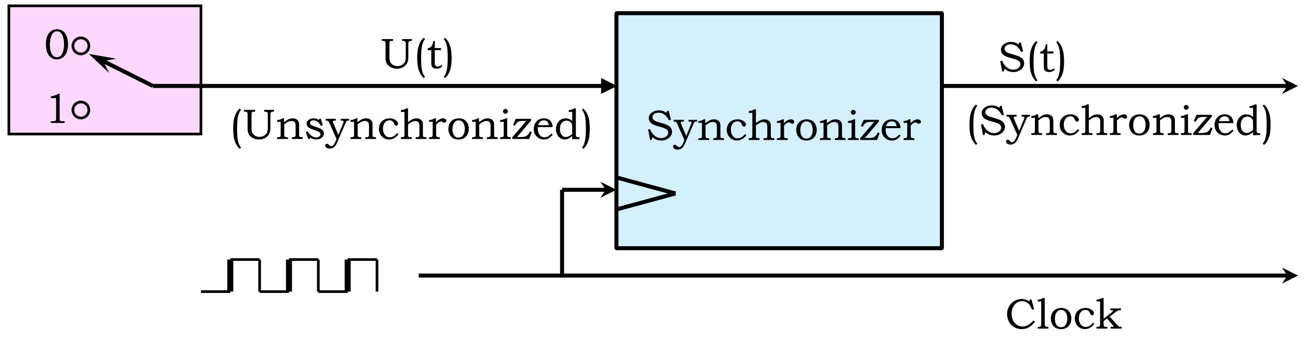
The World Doesn't Run on Our Clock!

What if each button input is an asynchronous 0/1 level?



To build a system with asynchronous inputs, we have to break the rules: *we cannot guarantee that setup and hold time requirements are met at the inputs!*

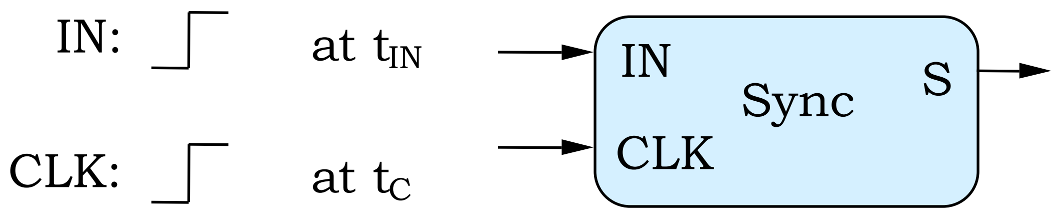
So, we need a "synchronizer" at each input:



Valid except for brief periods following active clock edges

The Bounded-time Synchronizer

A classic $\wedge$ problem
UNSOLVABLE

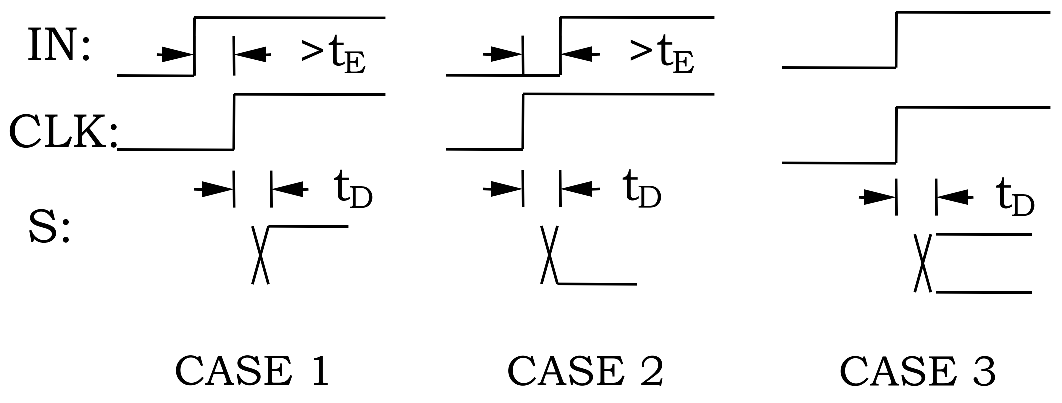


For NO finite values of t_E and t_D is this spec realizable, even with reliable components!

Synchronizer specifications:

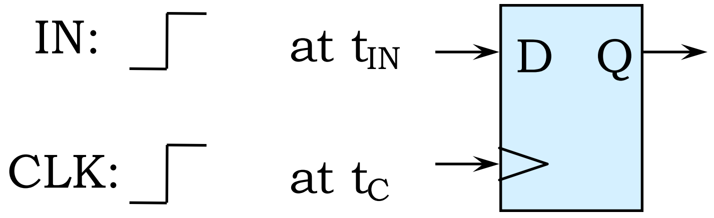
- finite t_D (decision time)
- finite t_E (allowable error)
- value of S at time t_C+t_D:

1	if $t_{IN} < t_C - t_E$	
0	if $t_{IN} > t_C + t_E$	
0, 1		otherwise



Unsolvable? That can't be true…

Let's just use a D Register:



We're lured by the digital abstraction into assuming that Q must be either 1 or 0. But let's look at the input latch in the flip flop when IN and CLK change at about the same time…

DECISION TIME is T_{PD} of register.

ALLOWABLE ERROR is $\max(t_{SETUP}, t_{HOLD})$

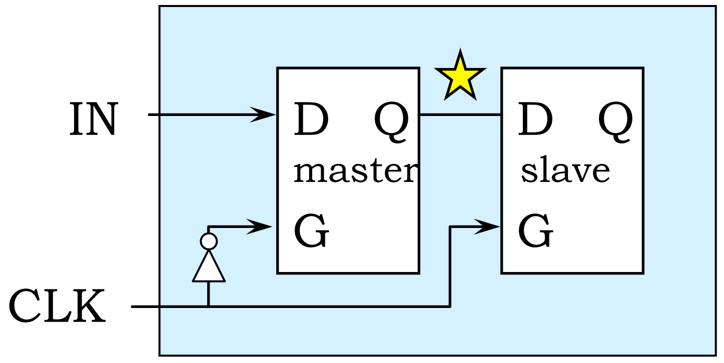
Our logic:

T_{PD} after T_C, we'll have

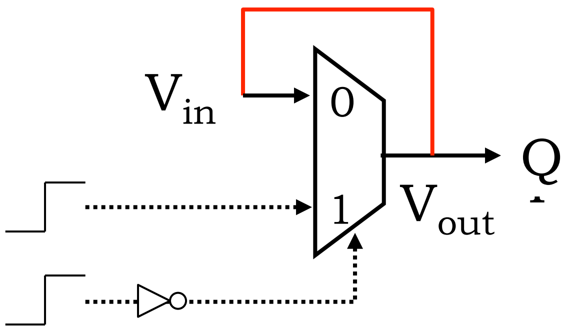
$Q=1$ iff $t_{IN} + t_{SETUP} < t_C$

$Q=0$ iff $t_C + t_{HOLD} < t_{IN}$

$Q=0$ or 1 otherwise.

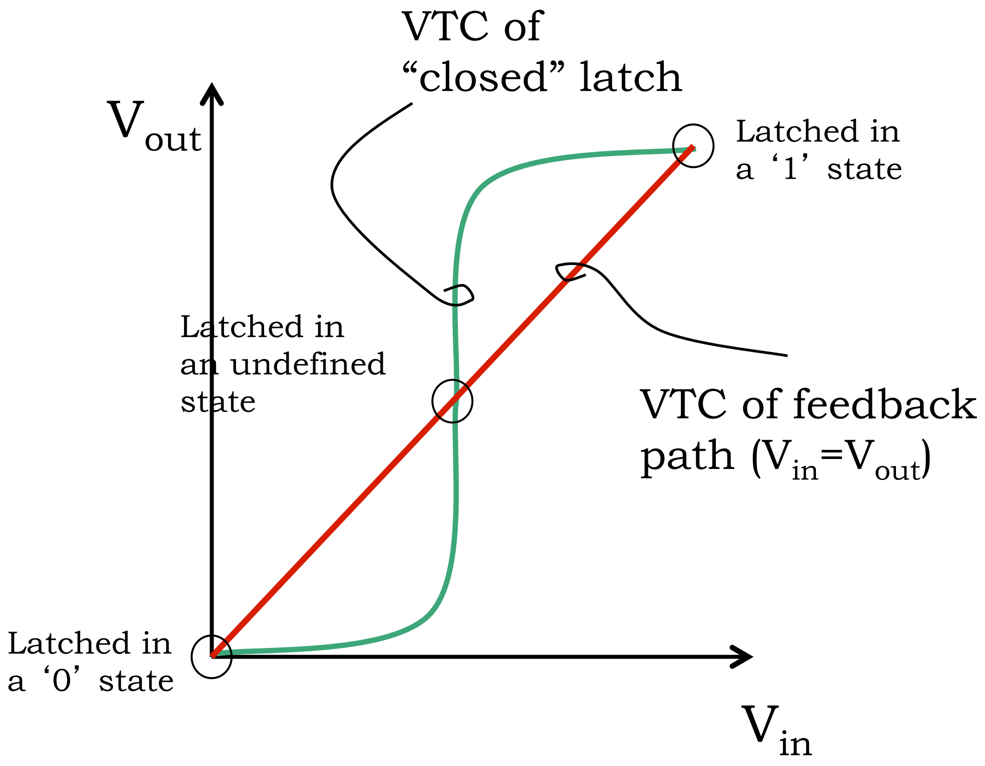


The Mysterious Metastable State



Recall that the latch output is the solution to two simultaneous constraints:

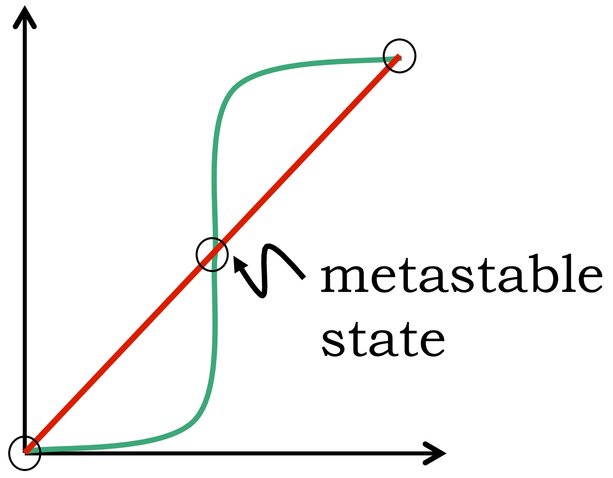
1. The VTC of path thru MUX; and
2. $V_{in} = V_{out}$



In addition to our expected stable solutions, we find an unstable equilibrium in the forbidden zone called the "Metastable State"

Metastable State: Properties

1. It corresponds to an *invalid* logic level.
2. It's an *unstable* equilibrium; a small perturbation will cause it to move toward a stable 0 or 1.
3. It will settle to a valid 0 or 1… eventually.
4. BUT – depending on how close it is to the $V_{in}=V_{out}$ "fixed point" of the device – it may take arbitrarily long to settle out.
5. EVERY bistable system exhibits at least one metastable state!



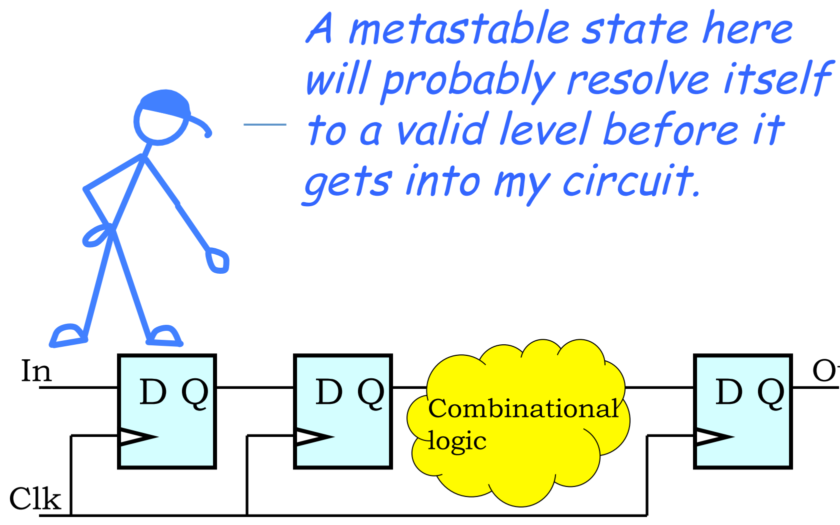
If metastable at t_0:

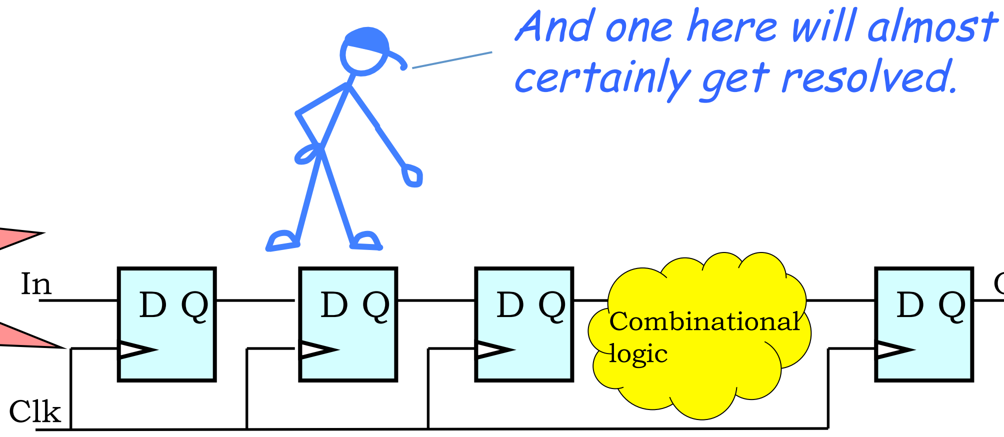
- p(metastable at t_0+T) > 0 for finite T
- p(metastable at t_0+T) decreases exponentially with increasing T

Solution: Delay Increases Reliability

Extra registers between the asynchronous input and your logic are the best insurance against metastable states.

For higher clock rates, consider adding additional registers.


A metastable state here will probably resolve itself to a valid level before it gets into my circuit.
In
D Q
D Q
Combinational logic
D Q
Out
Clk


And one here will almost certainly get resolved.
In
D Q
D Q
D Q
Combinational logic
D Q
Out
Clk