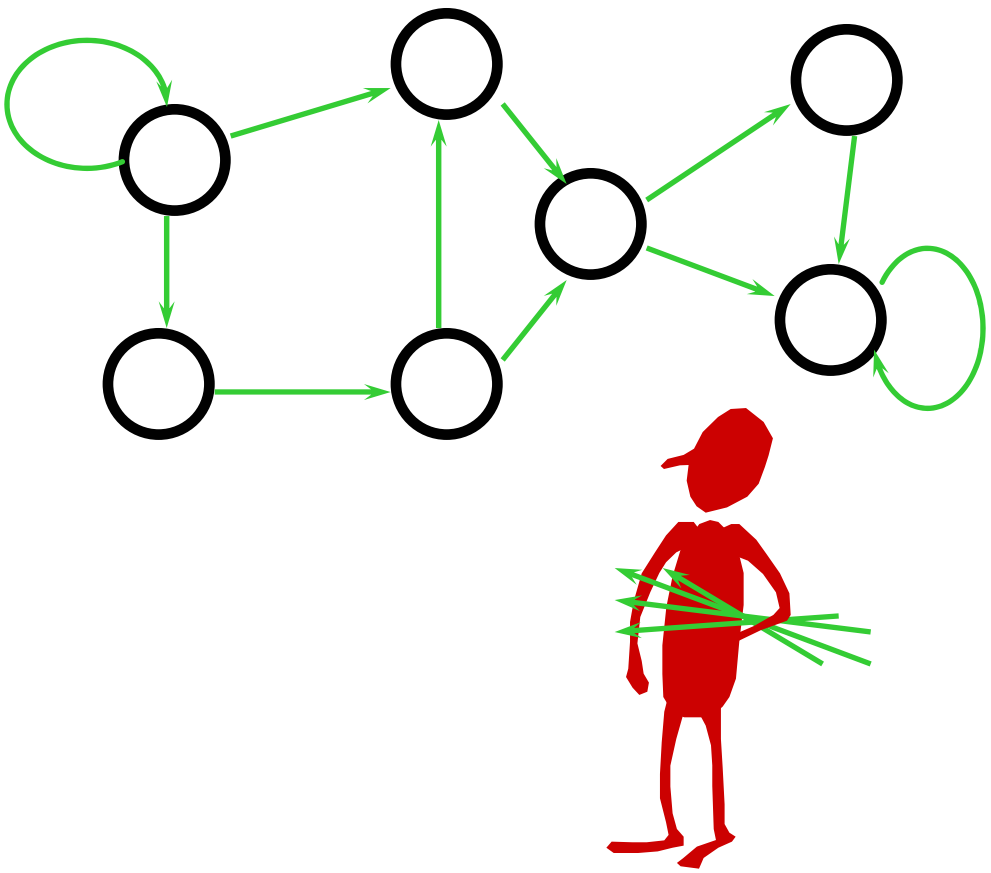
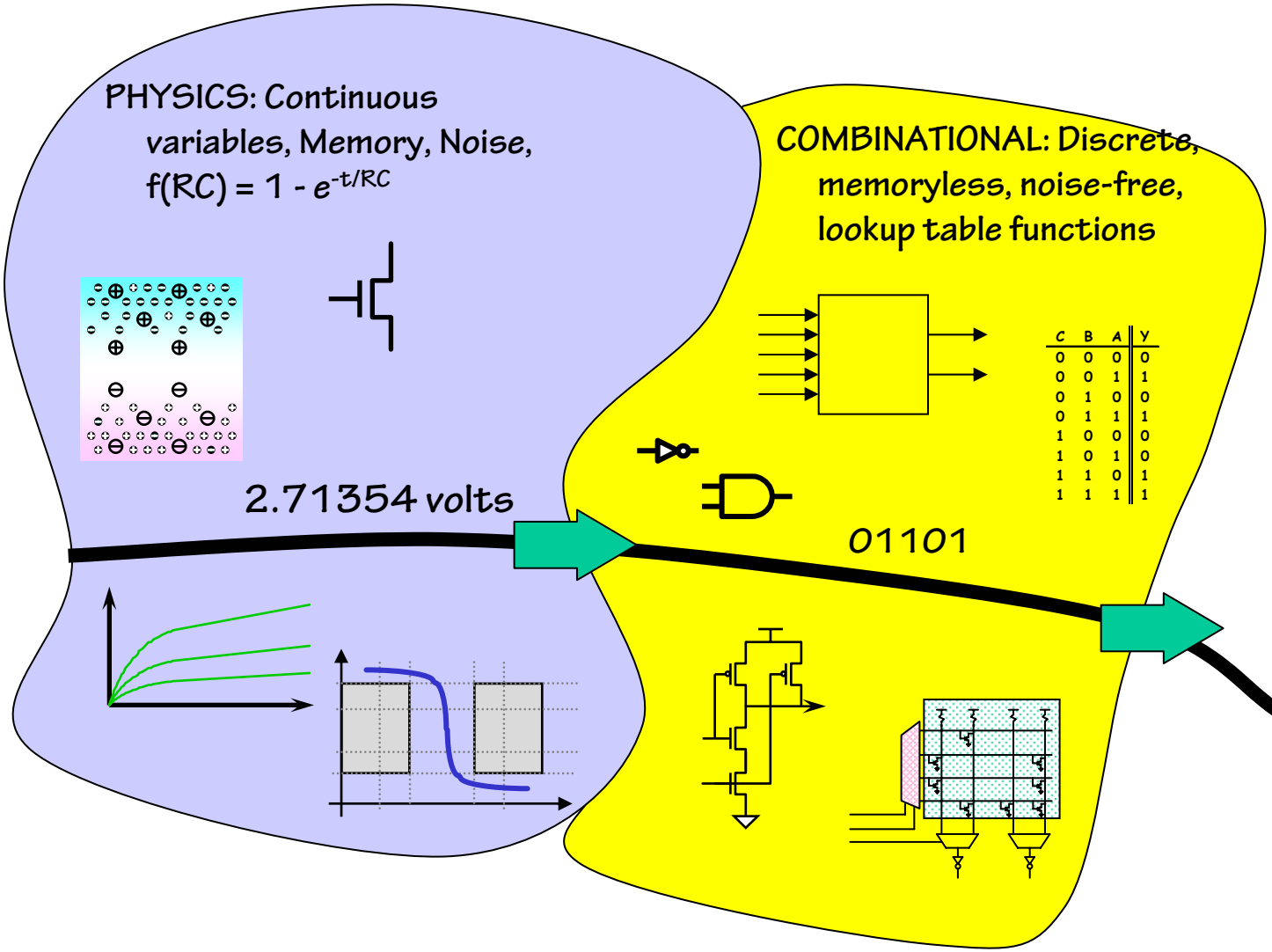
Sequential Logic



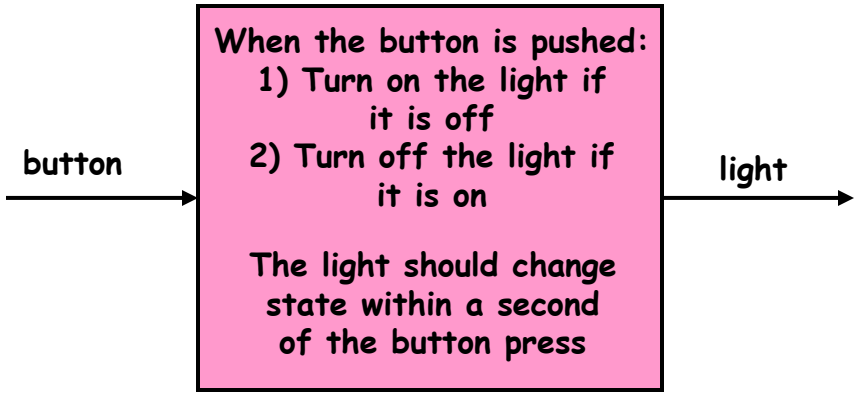
Handouts: Lecture Slides

6.004: Progress so far…



Something We Can't Build (Yet)

What if you were given the following design specification:

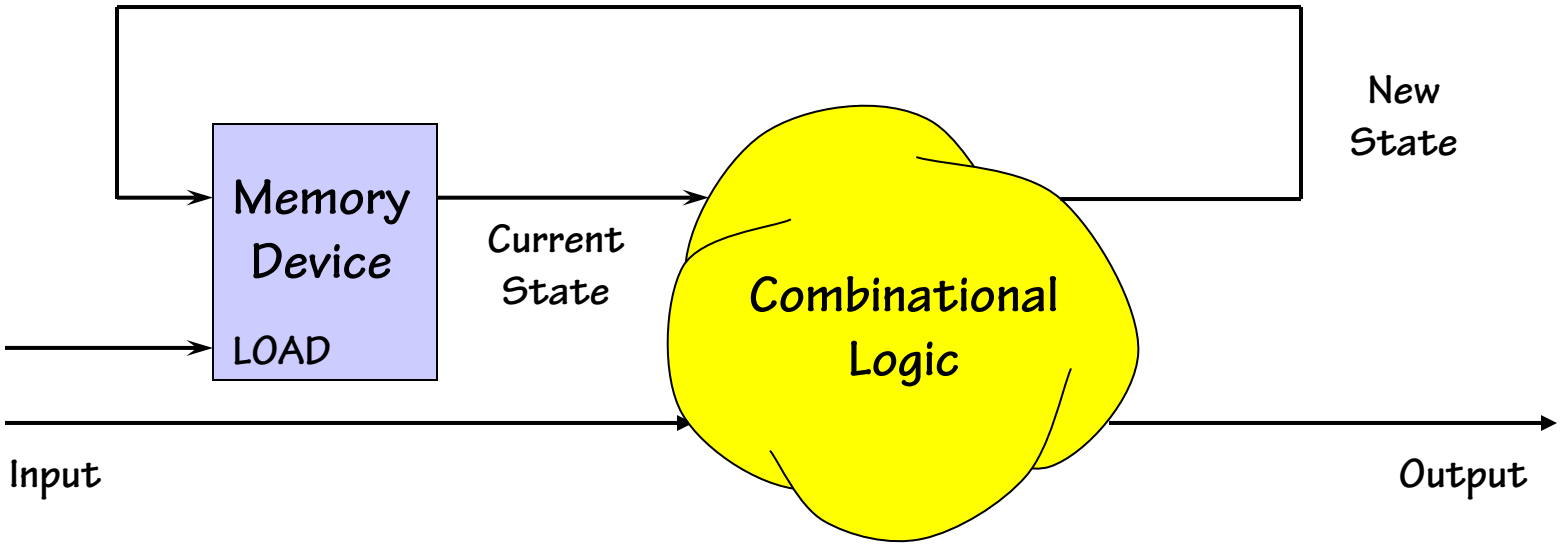


What makes this circuit so different from those we've discussed before?

1. "State" – i.e. the circuit has memory
2. The output was changed by a input "event" (pushing a button) rather than an input "value"

Digital State

One model of what we'd like to build



Plan: Build a Sequential Circuit with stored digital STATE –

- Memory stores CURRENT state, produced at output
- Combinational Logic computes
  - NEXT state (from input, current state)
  - OUTPUT bit (from input, current state)
- State changes on LOAD control input

Needed: *Storage*

Combinational logic is *stateless*:

valid outputs always reflect current inputs.

To build devices with state, we need components which *store* information (*e.g.*, state) for subsequent access.

ROMs (and other combinational logic) store information "wired in" to their truth table

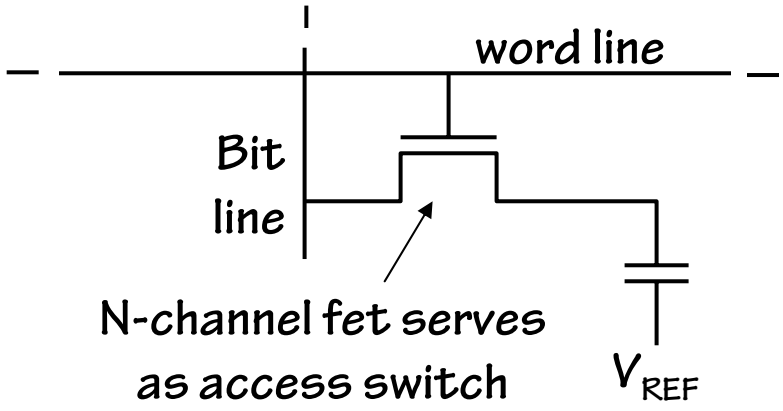
Read/Write memory elements are required to build devices capable of changing their contents.

How can we store – and subsequently access -- a bit?

- Mechanics: holes in cards/tapes
- Optics: Film, CDs, DVDs, …
- Magnetic materials
- Delay lines; moonbounce
- Stored charge

Storage: Using Capacitors

We've chosen to encode information using voltages and we know from 6.002 that we can "store" a voltage as charge on a capacitor:



To write:
Drive bit line, turn on access fet, force storage cap to new voltage

To read:
precharge bit line, turn on access fet, detect (small) change in bit line voltage

Pros:
- compact – low cost/bit (on BIG memories)

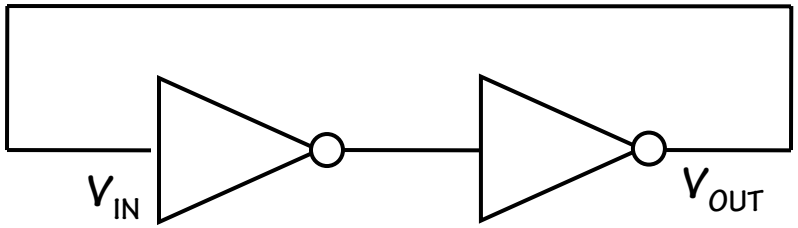
Cons:
- complex interface
- stable? (noise, …)
- it leaks! ⇒ refresh

Suppose we refresh CONTINUOUSLY?

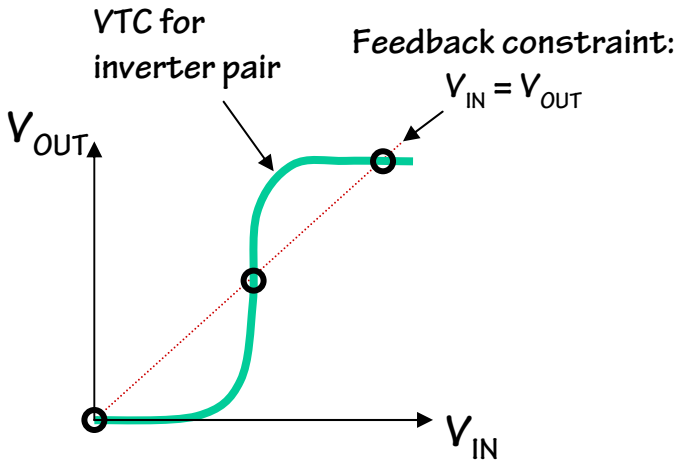
Storage: Using Feedback

IDEA: use positive feedback to maintain storage indefinitely.

Our logic gates are built to restore marginal signal levels, so noise shouldn't be a problem!



Result: a bistable storage element



Not affected by noise

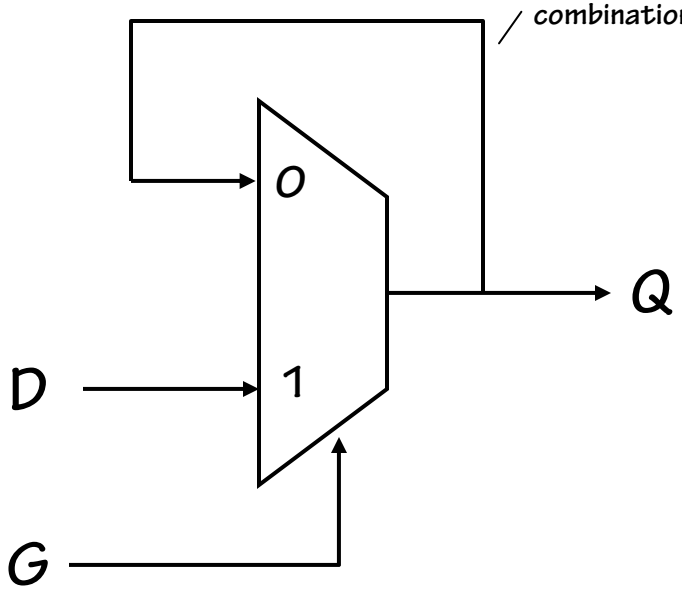
Three solutions:

- two end-points are stable
- middle point is unstable

We'll get back to this!

Settable Storage Element

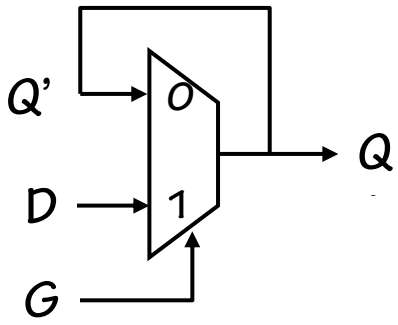
It's easy to build a settable storage element (called a latch) using a *lenient* MUX:

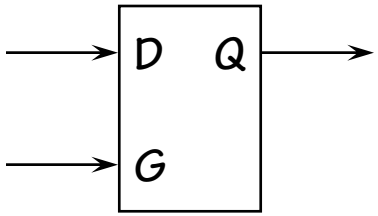


"state" signal appears as both input and output

G	D	Q_{IN}	Q_{OUT}	
0	--	0	0	Q stable
0	--	1	1	
1	0	--	0	Q follows D
1	1	--	1	

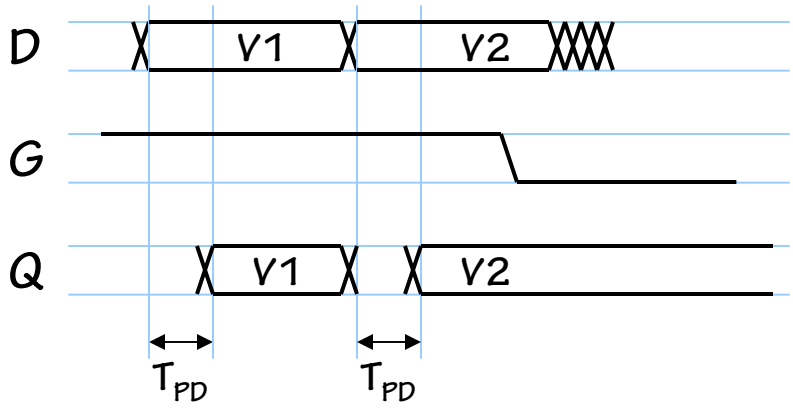
New Device: D Latch





G=1: Q follows D

G=0: Q holds

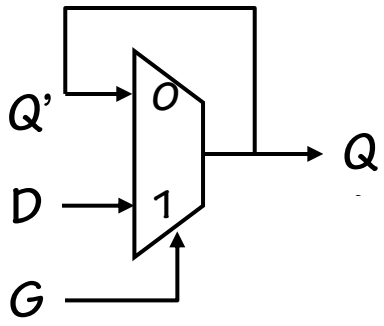


BUT… A change in D or G contaminates Q, hence Q' … how can this possibly work?

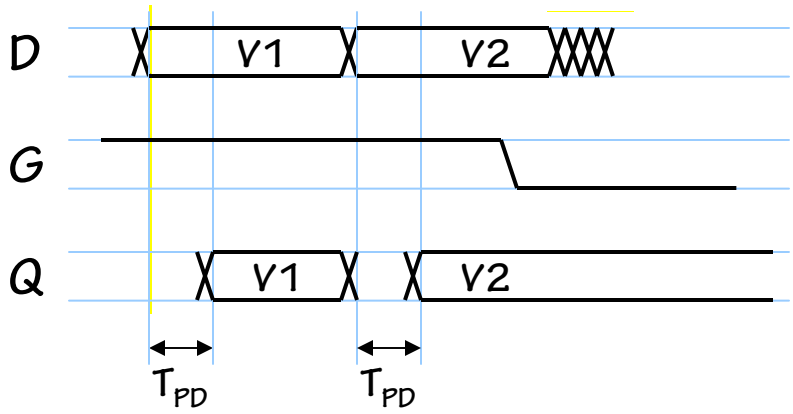
G=1: Q Follows D, *independently of Q'*

G=0: Q Holds stable Q', *independently of D*

A Plea for Lenience…



G	D	Q'	Q
1	0	X	0
1	1	X	1
X	0	0	0
X	1	1	1
0	X	0	0
0	X	1	1



Assume LENIENT Mux, propagation delay of T_{PD}

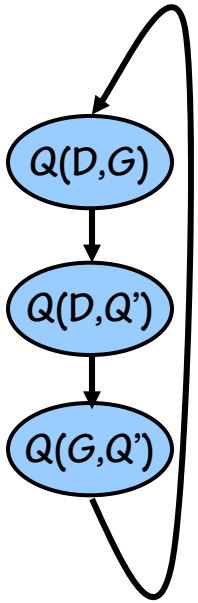
Then output valid when

- G=1, D stable for T_{PD}, *independently of Q';*

  or

- Q'=D stable for T_{PD}, *independently of G; or*

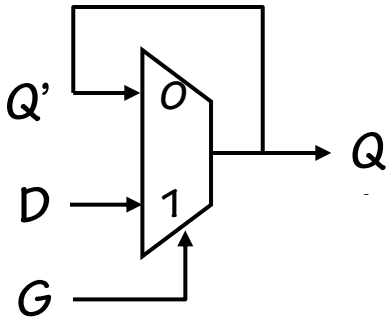
- G=0, Q' stable for T_{PD}, *independently of D*

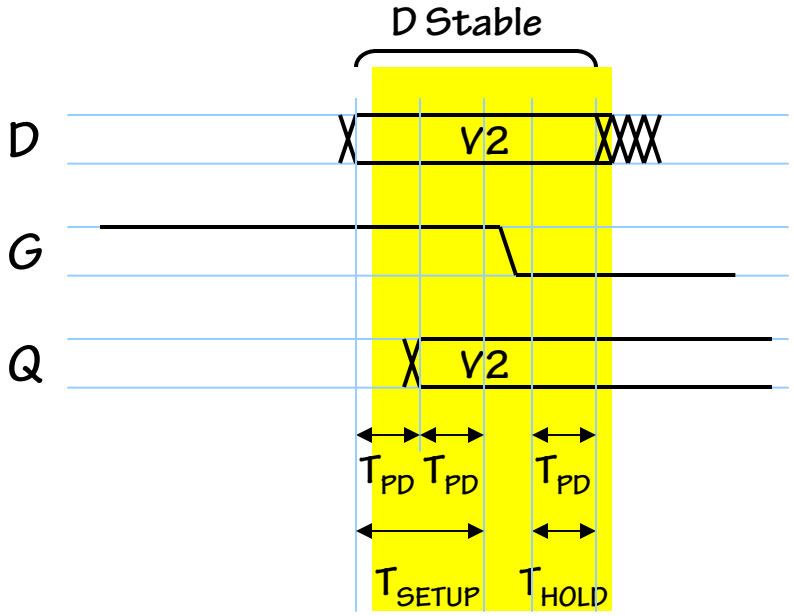


Does lenience *guarantee* a working latch?

What if D and G change at about the same time…

… with a little discipline





To reliably latch V2:

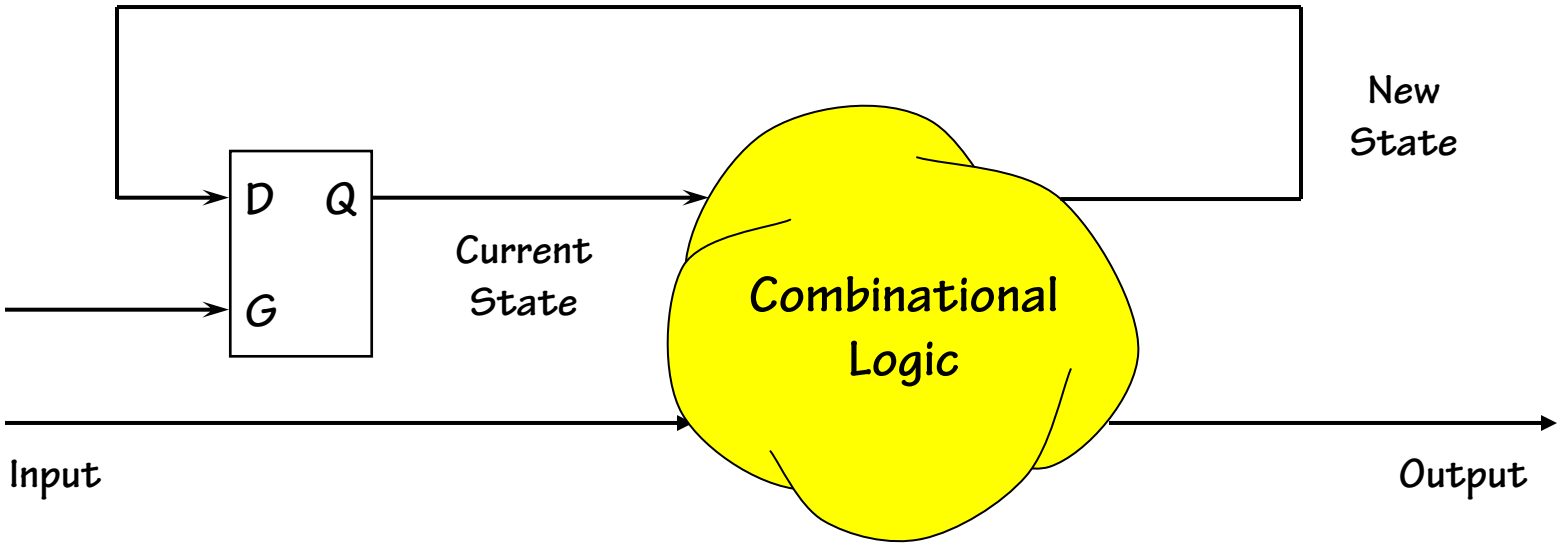
- Apply V2 to D, holding G=1
- After T_{PD}, V2 appears at Q=Q'
- After another T_{PD}, Q' & D both valid for T_{PD}; *will hold Q=V2 independently of G*
- Set G=0, while Q' & D hold Q=D
- After another T_{PD}, G=0 and Q' are sufficient to hold Q=V2 *independently of D*

Dynamic Discipline for our latch:

$T_{SETUP} = 2T_{PD}$: interval *prior to* G transition for which D must be stable & valid

$T_{HOLD} = T_{PD}$: interval *following* G transition for which D must be stable & valid

Lets try it out!

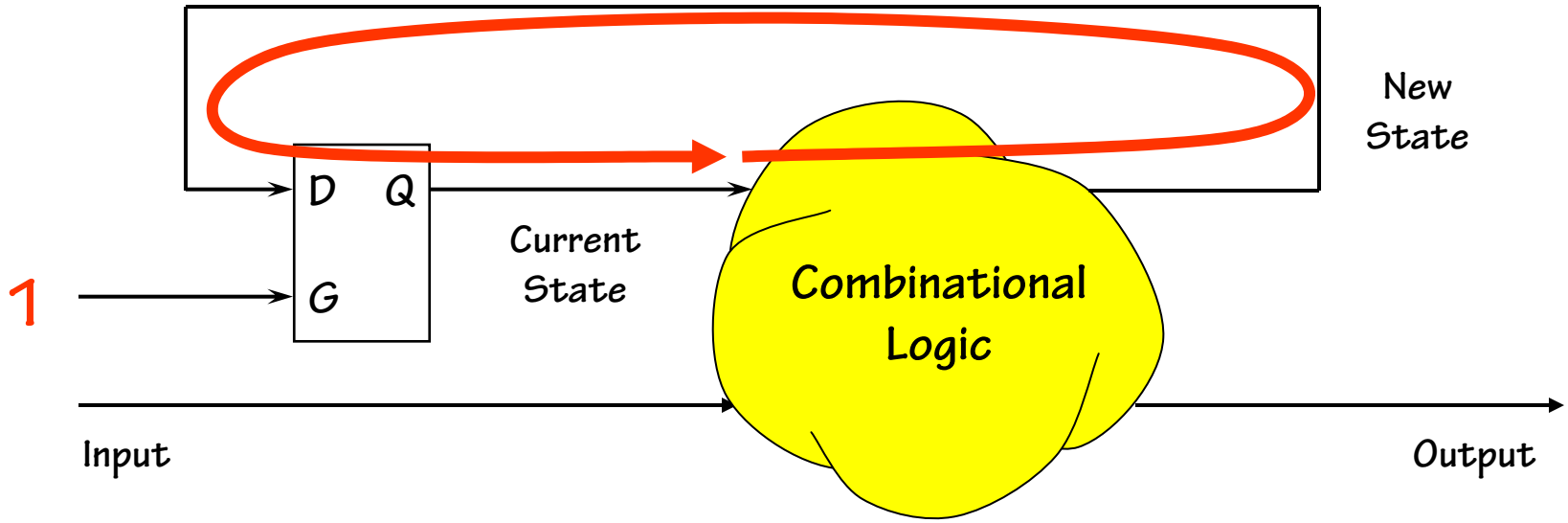


Plan: Build a Sequential Circuit with one bit of STATE –

- Single latch holds CURRENT state
- Combinational Logic computes
  - NEXT state (from input, current state)
  - OUTPUT bit (from input, current state)
- State changes when G = 1 (briefly!)

What happens when G=1?

Combinational Cycles



When G=1, latch is *Transparent…*

… provides a combinational path from D to Q.

Can't work without tricky timing constrants on G=1 pulse:

- Must fit within contamination delay of logic
- Must accommodate latch setup, hold times

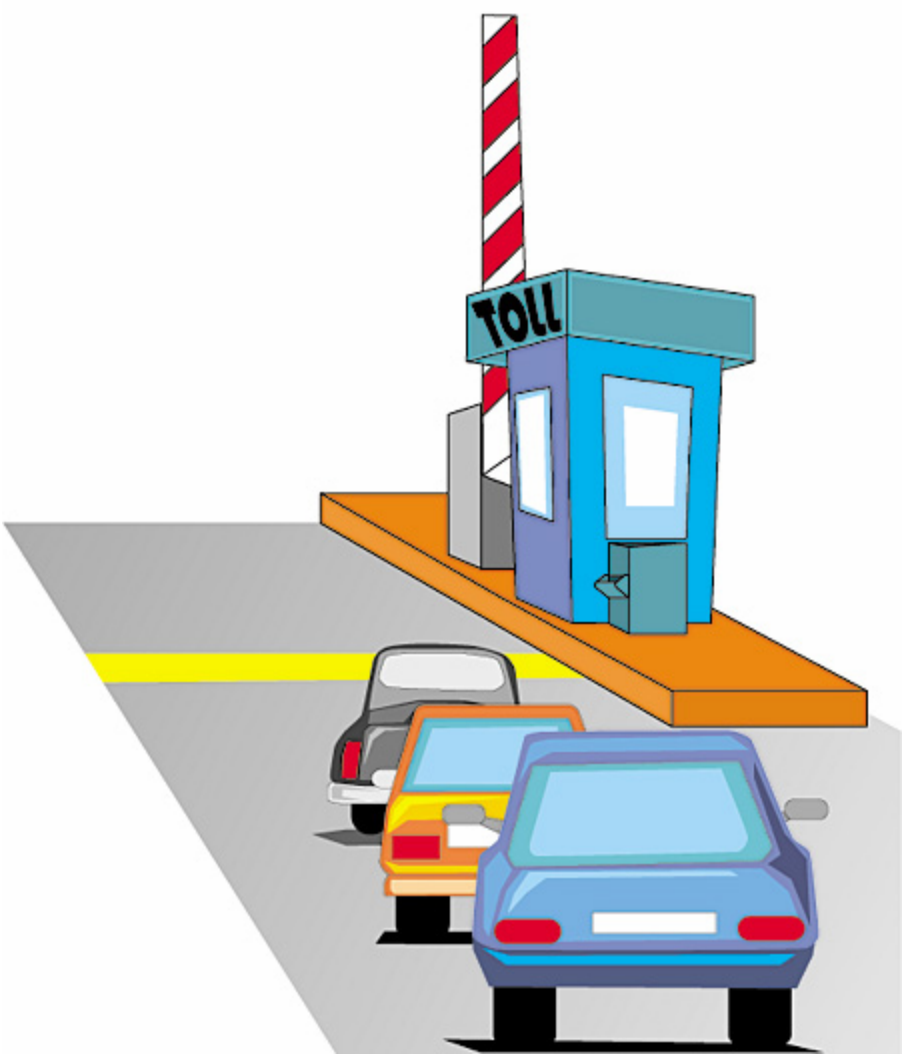
Want to signal an INSTANT, not an INTERVAL…

Flakey Control Systems

Here's a strategy for saving 2 bucks on the Sumner Tunnel!

Flakey Control Systems

Here's a strategy for saving 2 bucks on the Sumner Tunnel



Flakey Control Systems

Here's a strategy for saving 2 bucks on the Sumner Tunnel

Escapement Strategy

The Solution:
Add two gates and only open one at a time.

Escapement Strategy

The Solution:
Add two gates and only open one at a time.

Escapement Strategy

The Solution:
Add two gates and only open one at a time.

Escapement Strategy

The Solution:
Add two gates and only open one at a time. (Psst… Don't tell Massport)

Escapement Strategy

The Solution:
Add two gates and only open one at a time. (Psst… Don't tell Massport)

Escapement Strategy

The Solution:
Add two gates and only open one at a time. (Psst… Don't tell Massport)

Escapement Strategy

The Solution:
Add two gates and only open one at a time. (Psst… Don't tell Massport)

Escapement Strategy

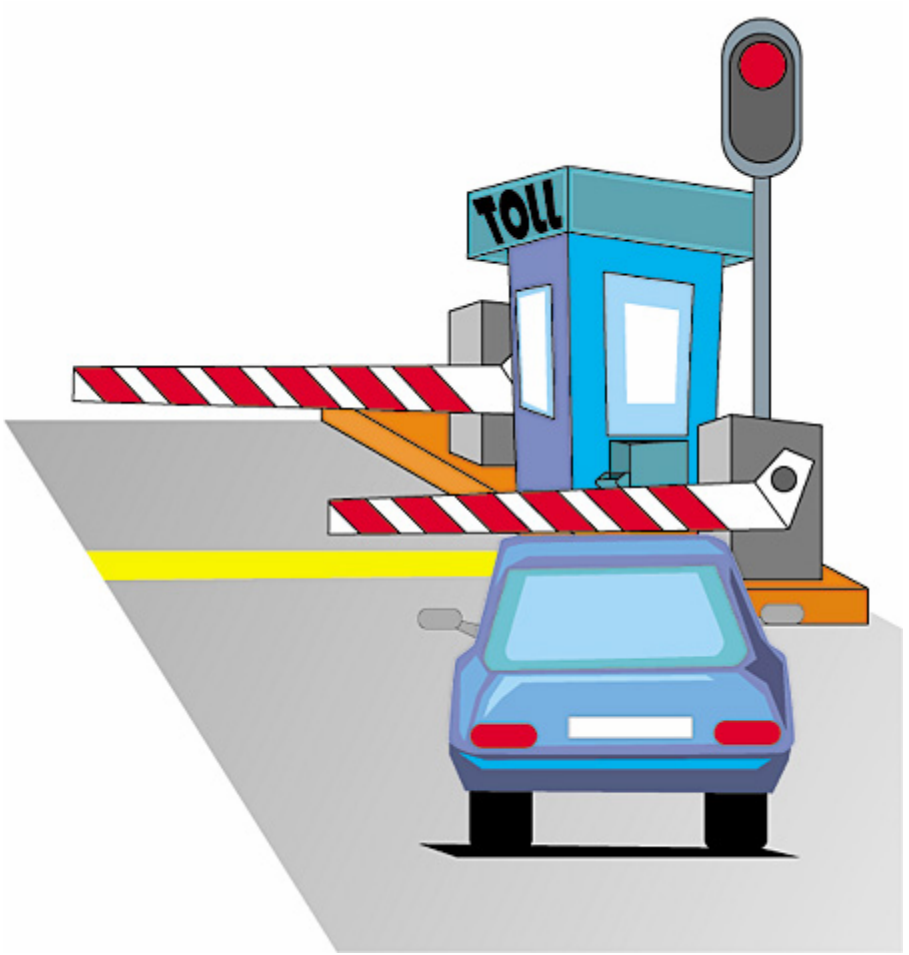
The Solution:
Add two gates and only open one at a time.

Escapement Strategy

The Solution:
Add two gates and only open one at a time.

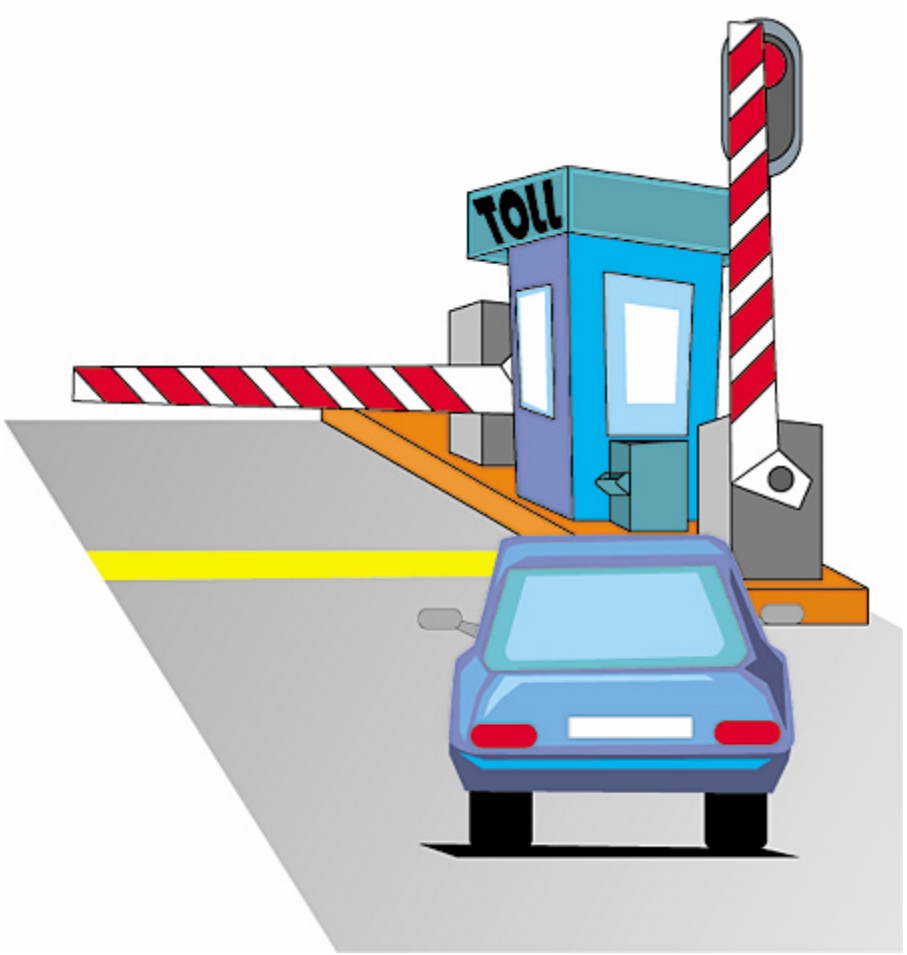
Escapement Strategy

The Solution:
Add two gates and only open one at a time. (Psst… Don't tell Massport)



Escapement Strategy

The Solution:
Add two gates and only open one at a time. (Psst… Don't tell Massport)

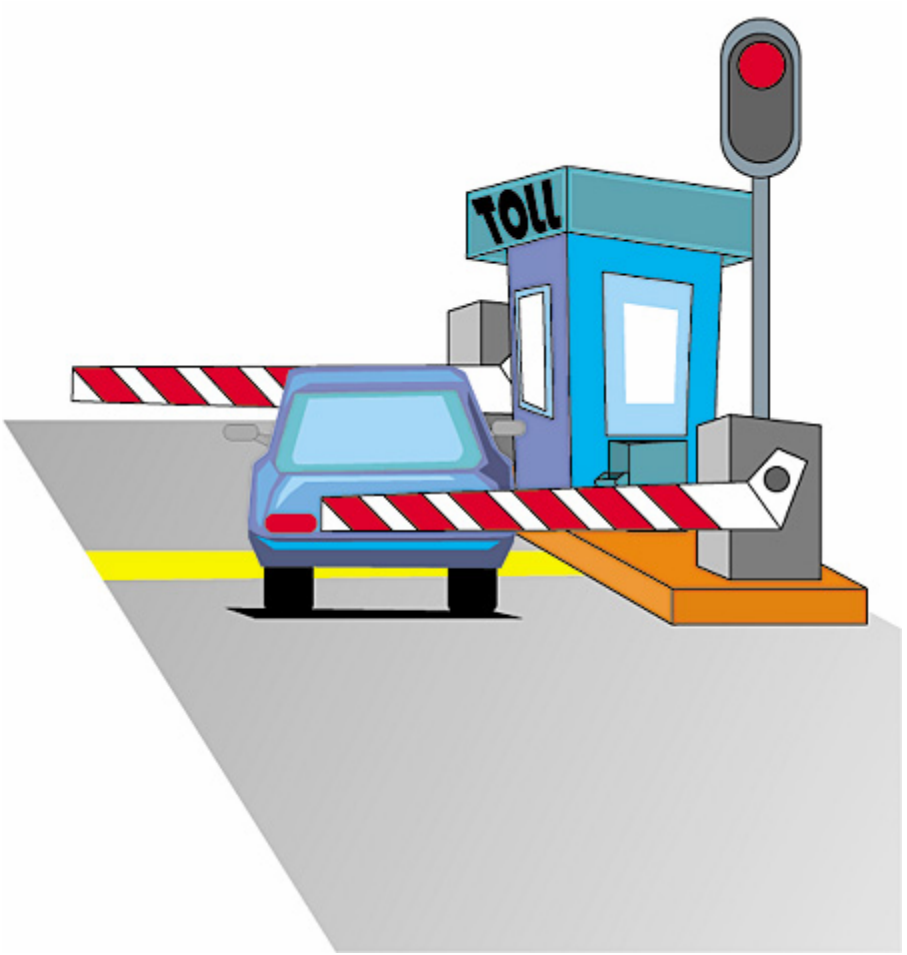


Escapement Strategy

The Solution:
Add two gates and only open one at a time. (Psst… Don't tell Massport)

Escapement Strategy

The Solution:
Add two gates and only open one at a time. (Psst… Don't tell Massport)



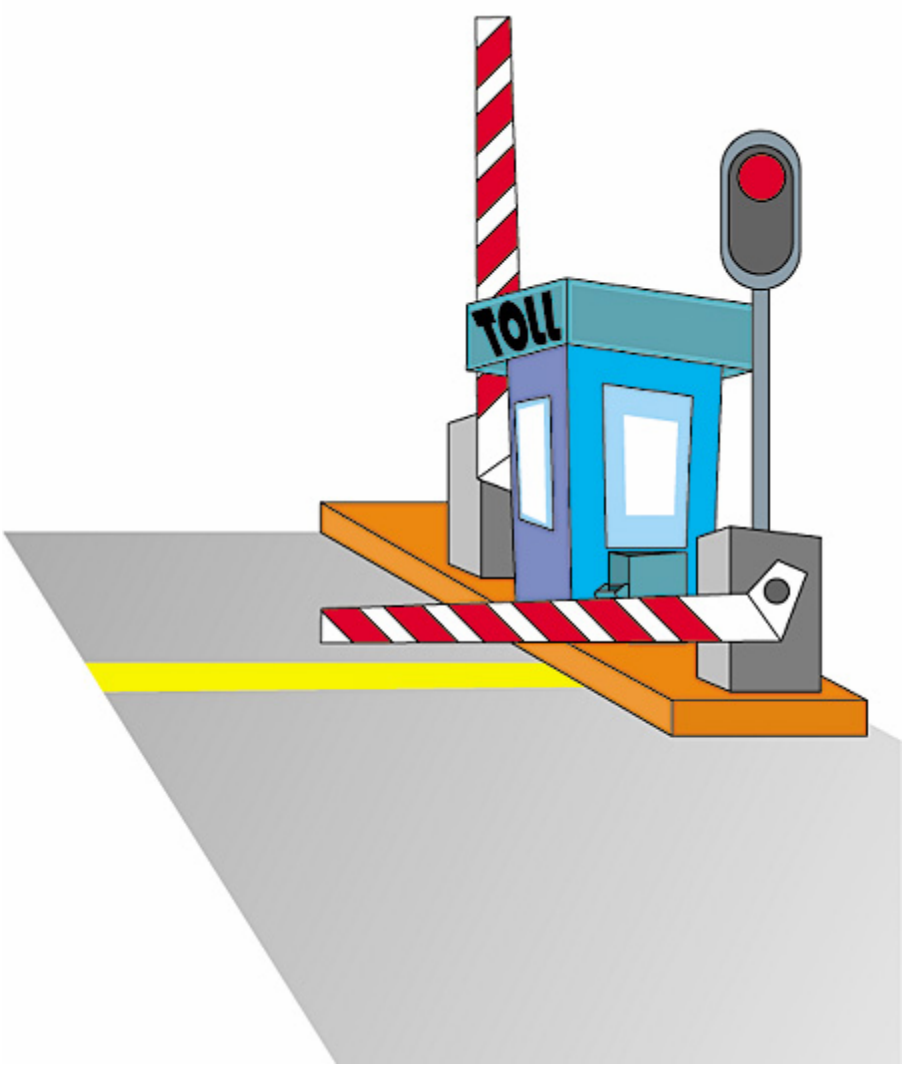
Escapement Strategy

The Solution:

Add two gates and only open one at a time.

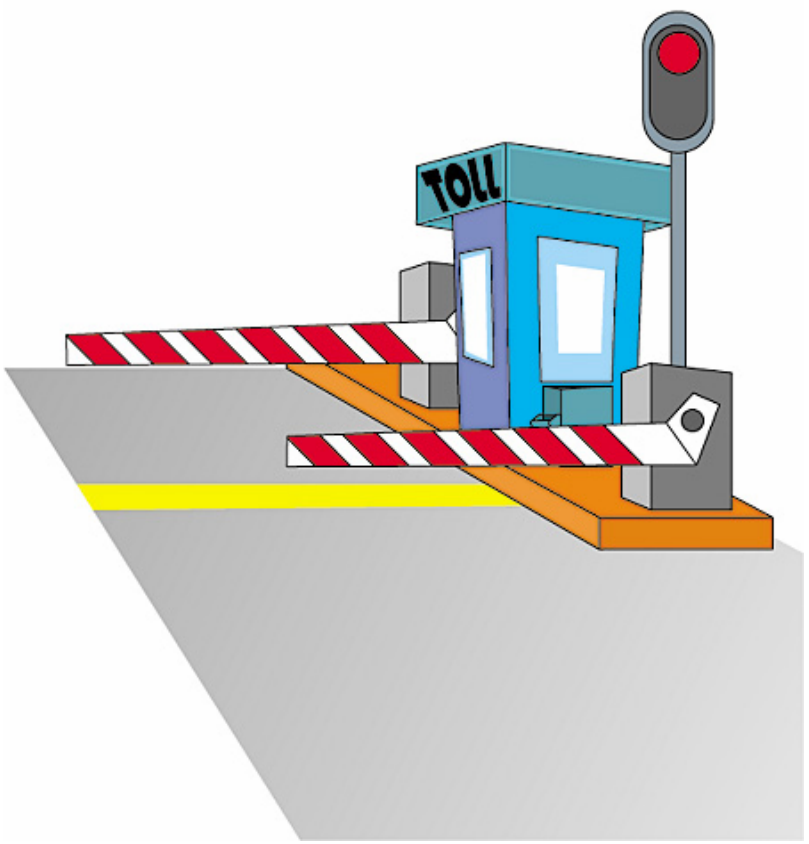
Escapement Strategy

The Solution:
Add two gates and only open one at a time.



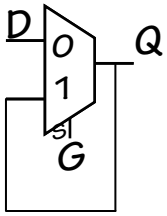
Escapement Strategy

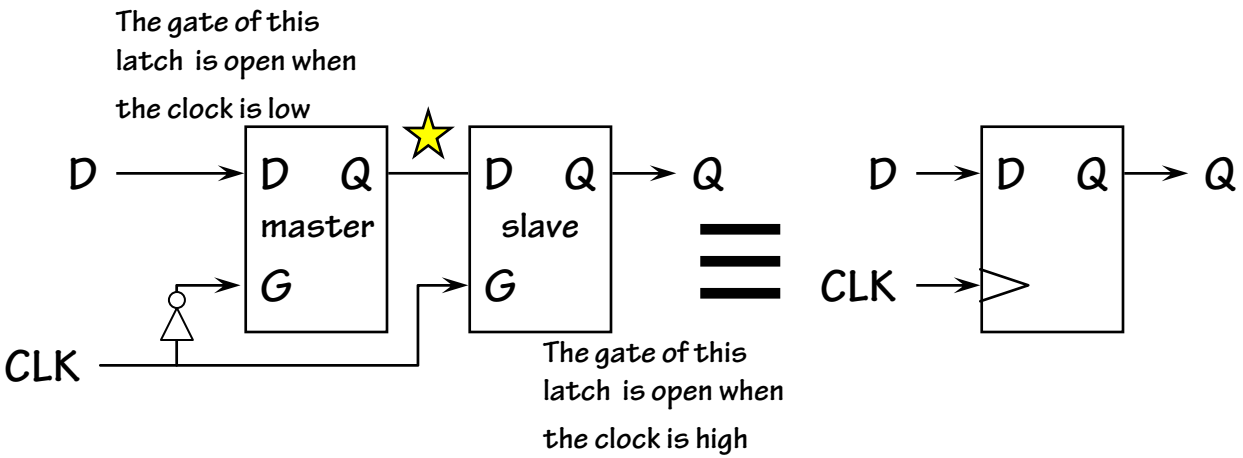
The Solution:
Add two gates and only open one at a time. (Psst… Don't tell Massport)



KEY: At no time is there an open path through both gates…

Edge-triggered Flip Flop





Observations:

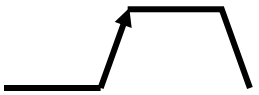
- only one latch "transparent" at any time:
  - master closed when slave is open
  - slave closed when master is open

→ no combinational path through flip flop

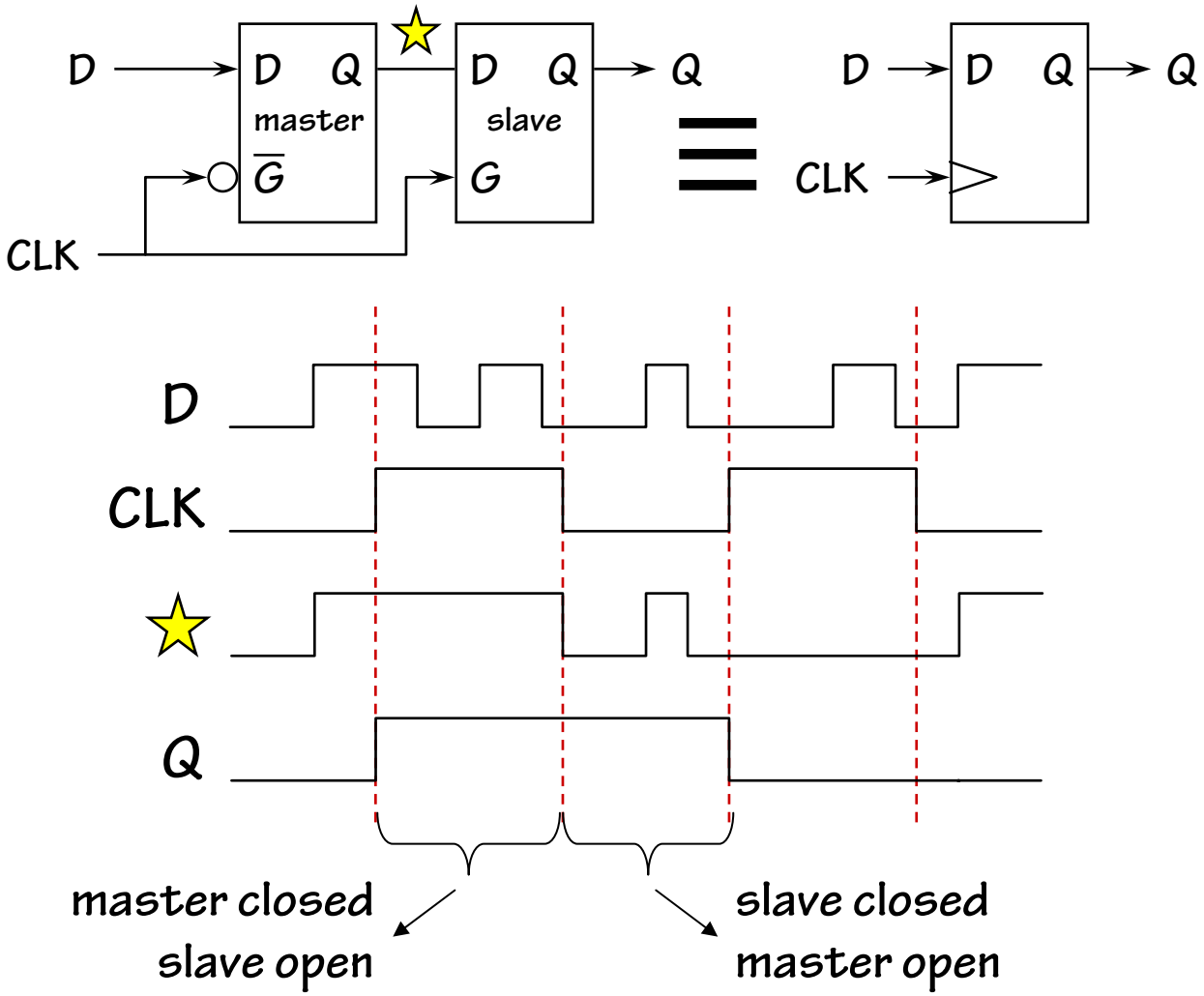
(the feedback path in one of the master or slave latches is always active)

Transitions mark *instants*, not intervals

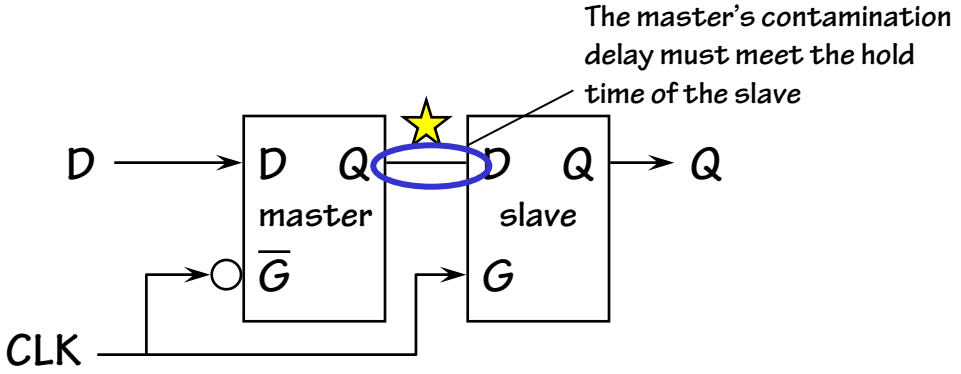
- Q only changes shortly after 0 →1 transition of CLK, so flip flop appears to be "triggered" by rising edge of CLK



Flip Flop Waveforms



Um, about that hold time…

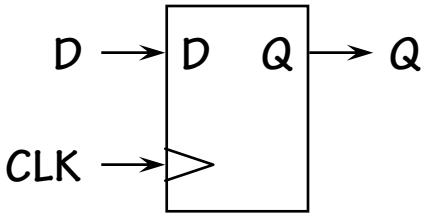


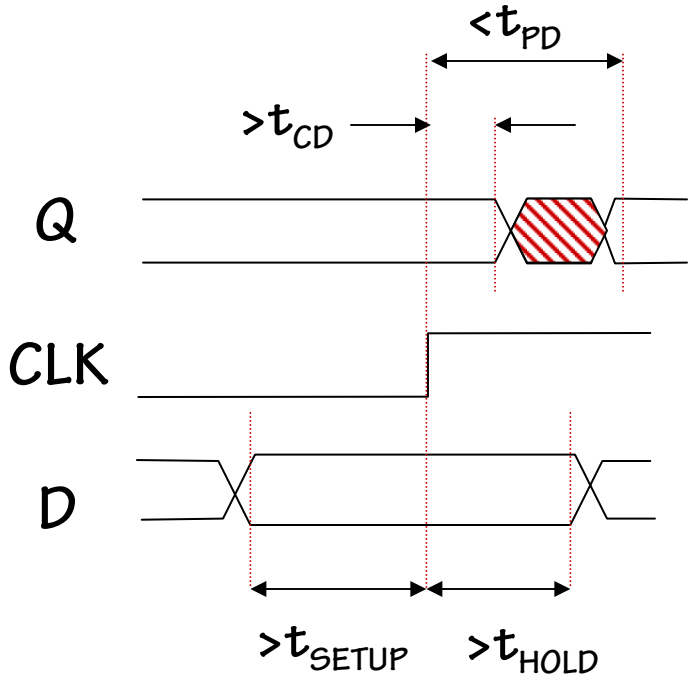
Consider HOLD TIME requirement for slave:

- Negative (1 →0) clock transition → slave freezes data:
  - SHOULD be no output glitch, since master held constant data; BUT
  - master output contaminated by change in G input!
- HOLD TIME of slave not met, UNLESS we assume sufficient contamination delay in the path to its D input!

Accumulated t_{CD} thru inverter, G → Q path of master must cover slave t_{HOLD} for this design to work!

Flip Flop Timing - I





t_{PD}: maximum propagation delay, CLK →Q

t_{CD}: minimum contamination delay, CLK →Q

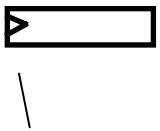
t_{SETUP}: setup time

guarantee that D has propagated through feedback path before master closes

t_{HOLD}: hold time

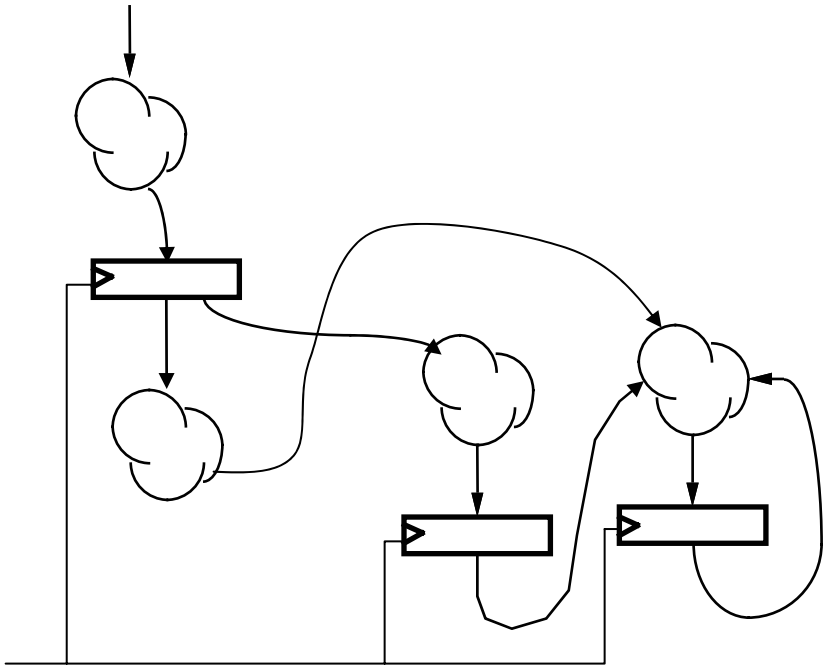
guarantee master is closed and data is stable before allowing D to change

Single-clock Synchronous Circuits



Does that symbol register?

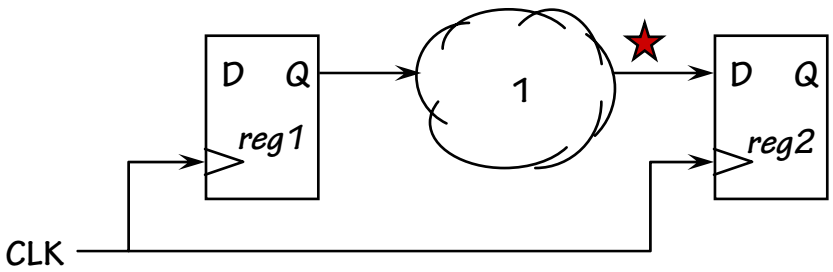
We'll use Flip Flops and *Registers* – groups of FFs sharing a clock input – in a highly constrained way to build digitial systems:

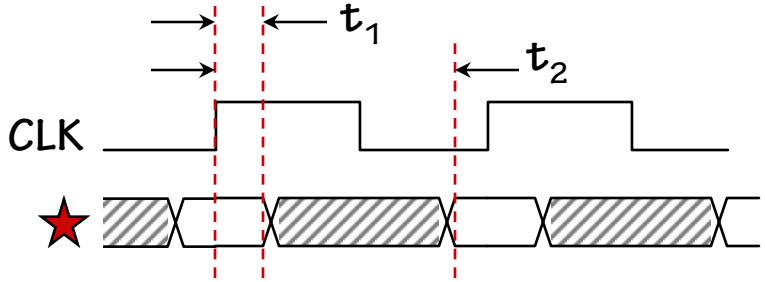


Single-clock Synchronous Discipline

- No combinational cycles
- Single clock signal shared among all clocked devices
- Only care about value of combinational circuits just before rising edge of clock
- Period greater than every combinational delay
- Change saved state after noise-inducing logic transitions have stopped!

Flip Flop Timing - II





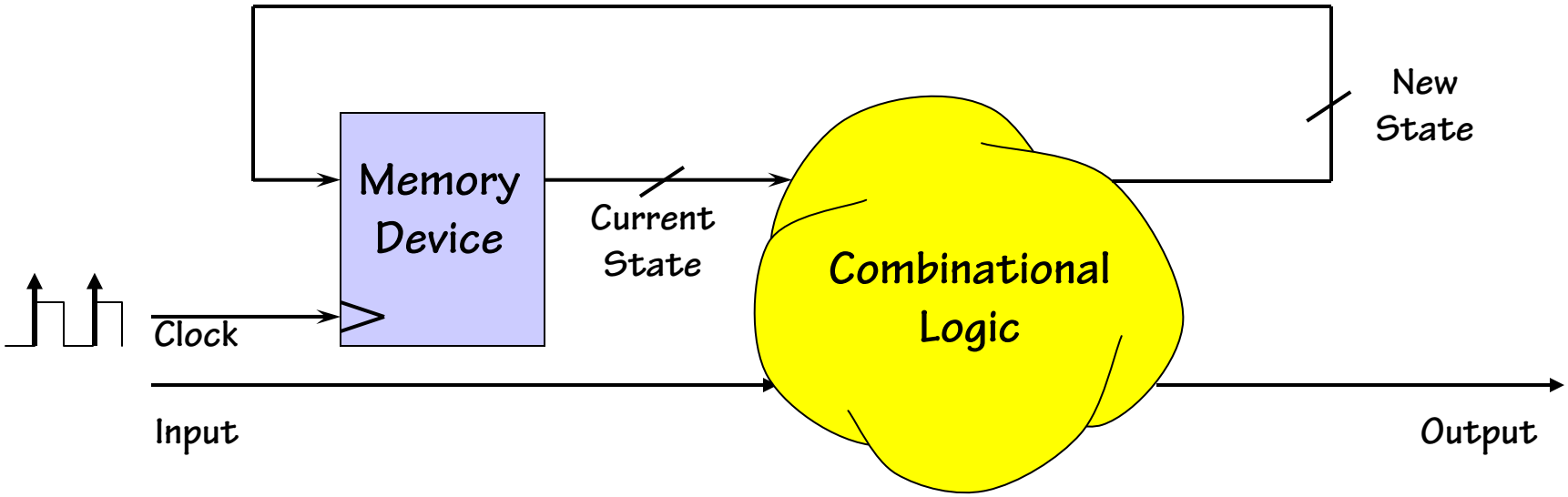
$t_1 = t_{CD,reg1} + t_{CD,1} > t_{HOLD,reg2}$

$t_2 = t_{PD,reg1} + t_{PD,1} < t_{CLK} - t_{SETUP,reg2}$

Questions for register-based designs:

- how much time for useful work (i.e. for combinational logic delay)?
- does it help to guarantee a minimum t_{CD}? How 'bout designing registers so that $t_{CD,reg} > t_{HOLD,reg}$?
- what happens if CLK signal doesn't arrive at the two registers at exactly the same time (a phenomenon known as "clock skew")?

Model: Discrete Time

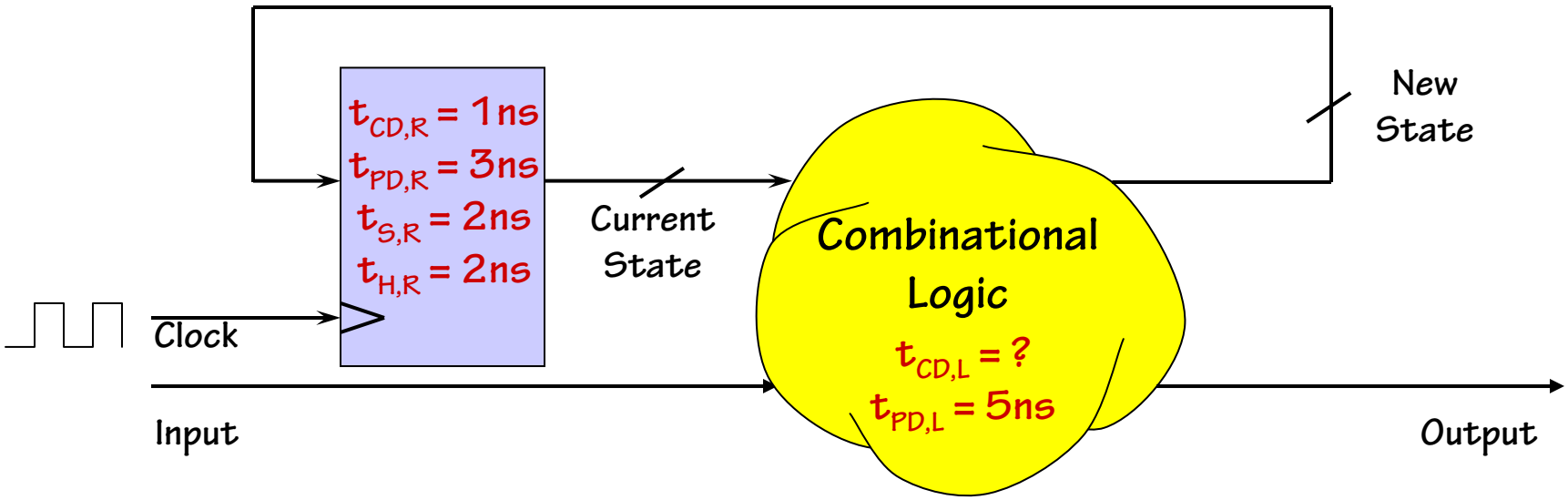


Active Clock Edges punctuate time ---

- Discrete Clock periods
- Discrete State Variables
- Discrete specifications (simple rules – *eg* tables – relating outputs to inputs, state variables)

- ABSTRACTION: Finite State Machines (next lecture!)

Sequential Circuit Timing



Questions:

- Constraints on T_{CD} for the logic? > 1 ns
- Minimum clock period? > 10 ns ($T_{PD,R}+T_{PD,L}+T_{S,R}$)
- Setup, Hold times for Inputs? $T_S = T_{PD,L} + T_{S,R}$, $T_H = T_{H,R} - T_{CD,L}$

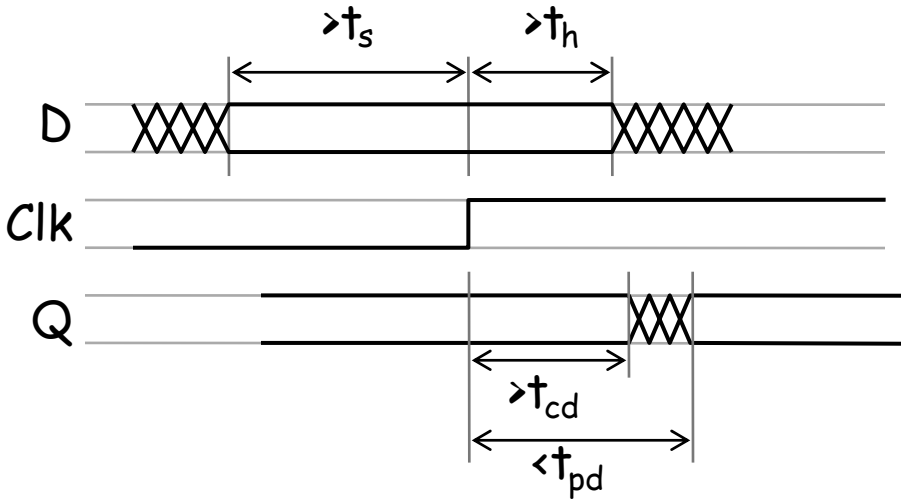
This is a simple *Finite State Machine* . . . more on Thursday!

Summary

"Sequential" Circuits (with memory):

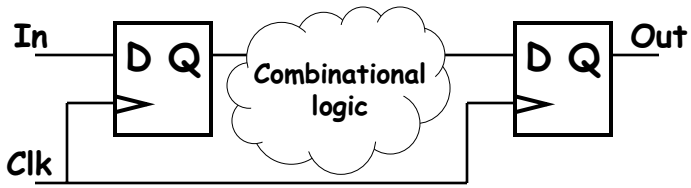
Basic memory elements:

- Feedback, detailed analysis => basic level-sensitive devices (eg, latch)
- 2 Latches => Flop
- Dynamic Discipline: constraints on input timing



Synchronous 1-clock logic:

- Simple rules for sequential circuits
- Yields clocked circuit with T_S, T_H constraints on input timing



Finite State Machines

Thursday's Topic!