

Sequential Logic

Jinyang Li

What we've learnt so far

- Combinatorial logic
 - E.g. Multiplexors (Mux), Decoders
 - Two ways of building a CL
 - Truth table \rightarrow sum of products circuits
 - ROM
- ALU
 - Compute all operations (+, OR, AND, NOR), multiplexer picks the result
 - Building a 64-bit adder
 - Ripple carry chains together 1-bit adders
 - Carry lookahead

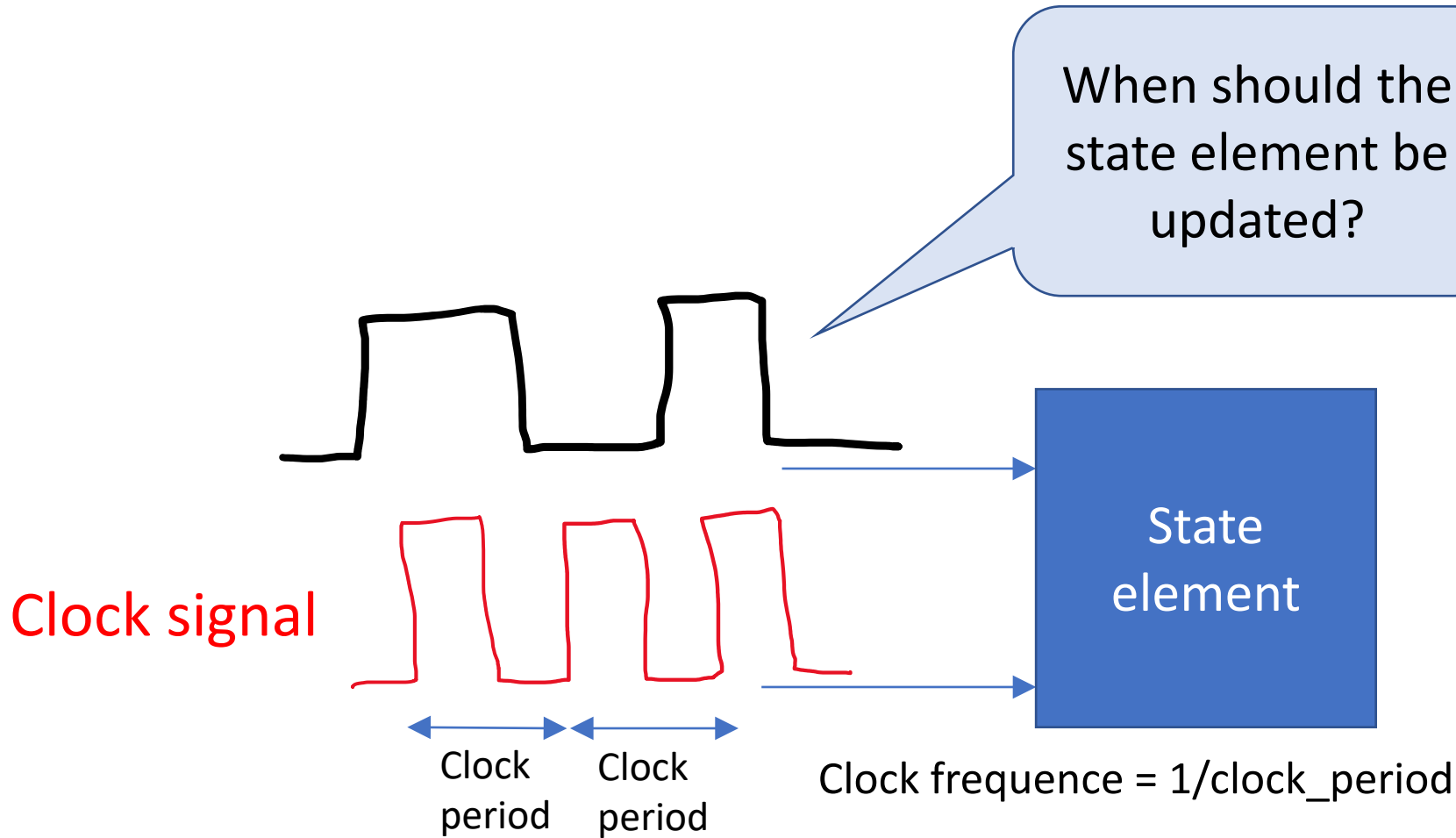
Two types of logic circuits

- Combinatorial circuit
 - Truth table \rightarrow sum of products
 - ROM
- Sequential circuit
 - output is dependent on both input and state (memory elements)

Today's lesson

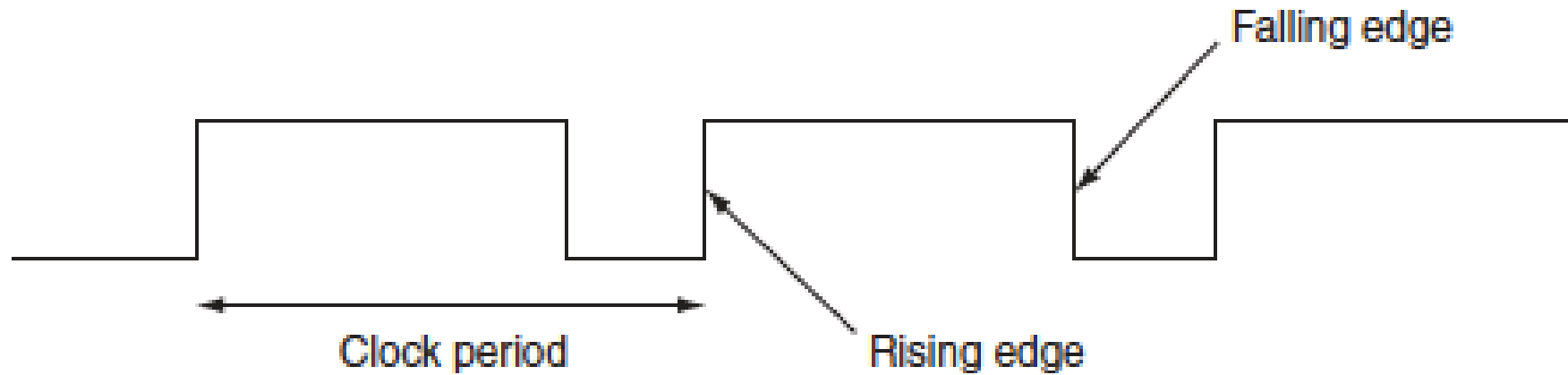


Sequential logic requires clock

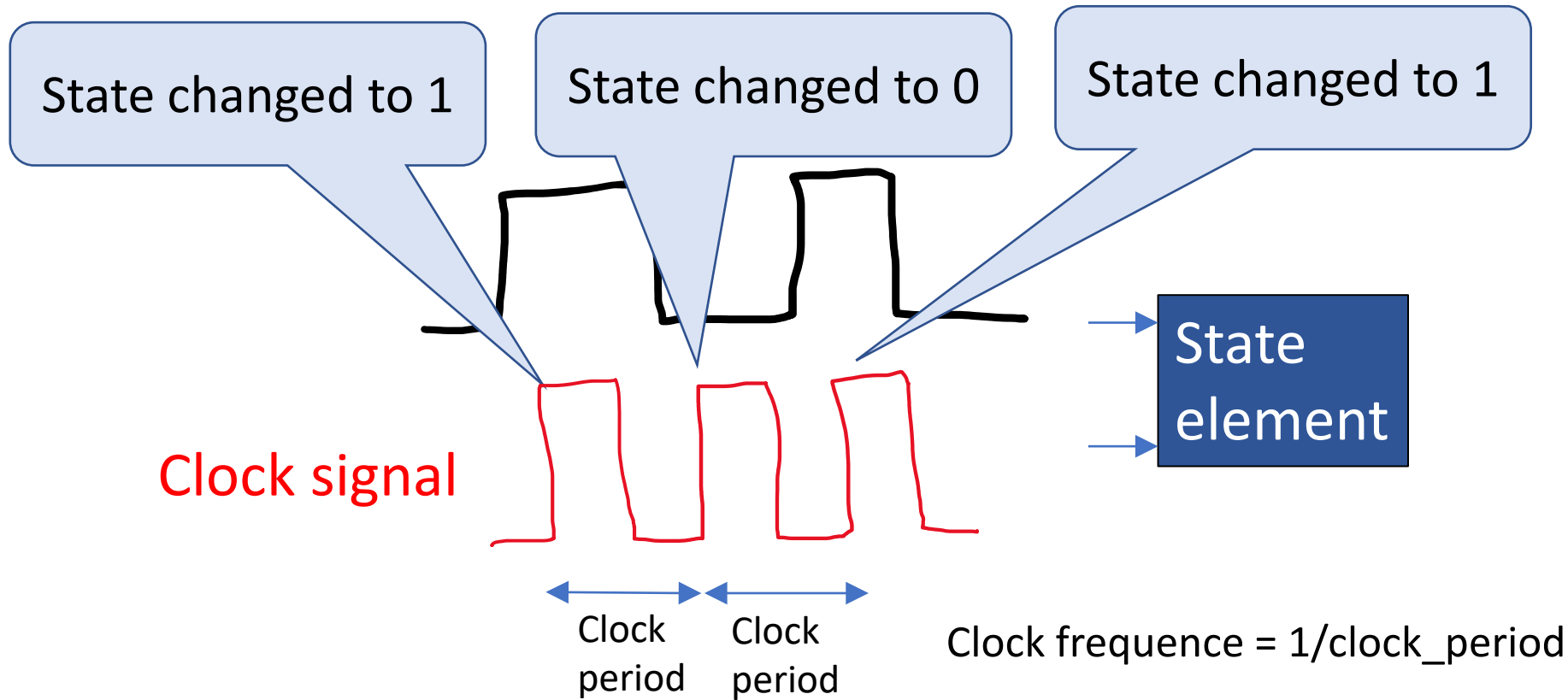


Clocks

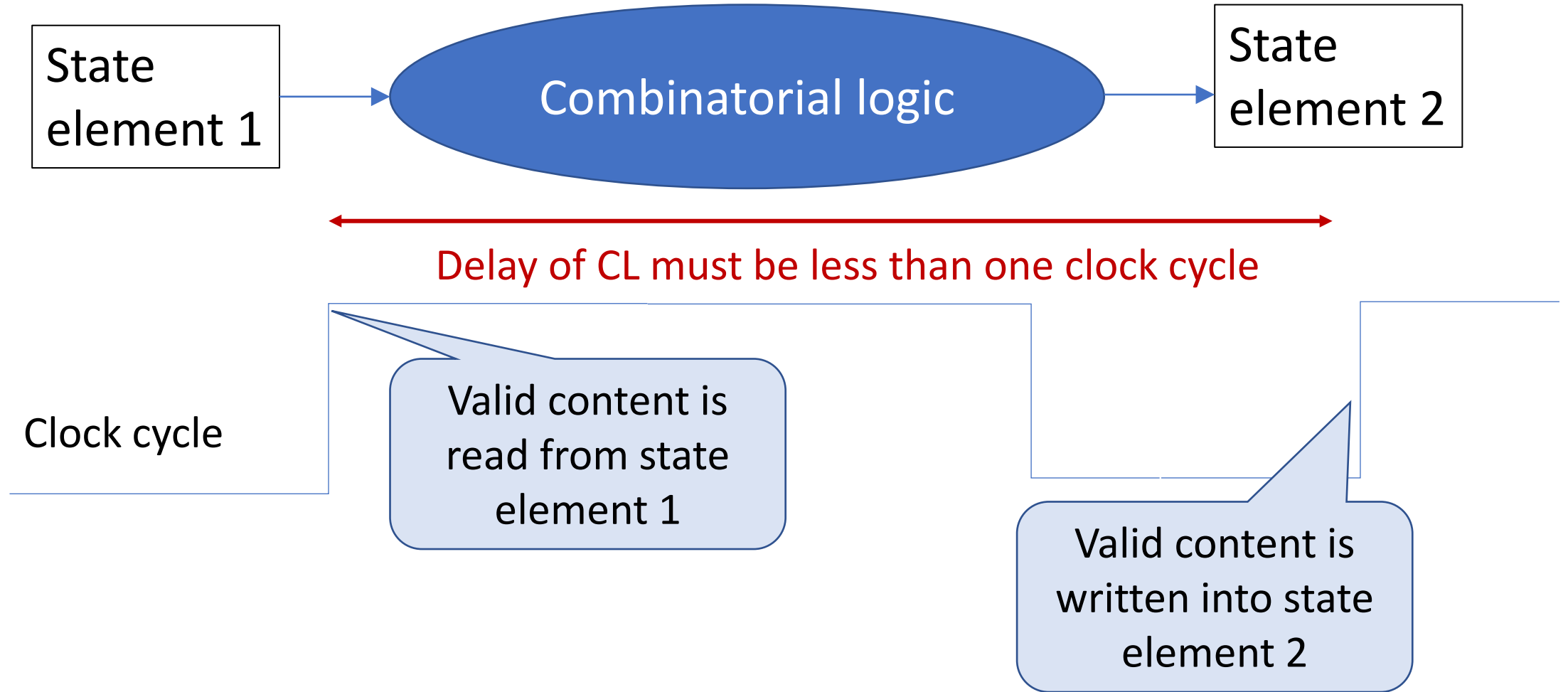
- Edge-triggered clocking: state content only changes on active clock edge



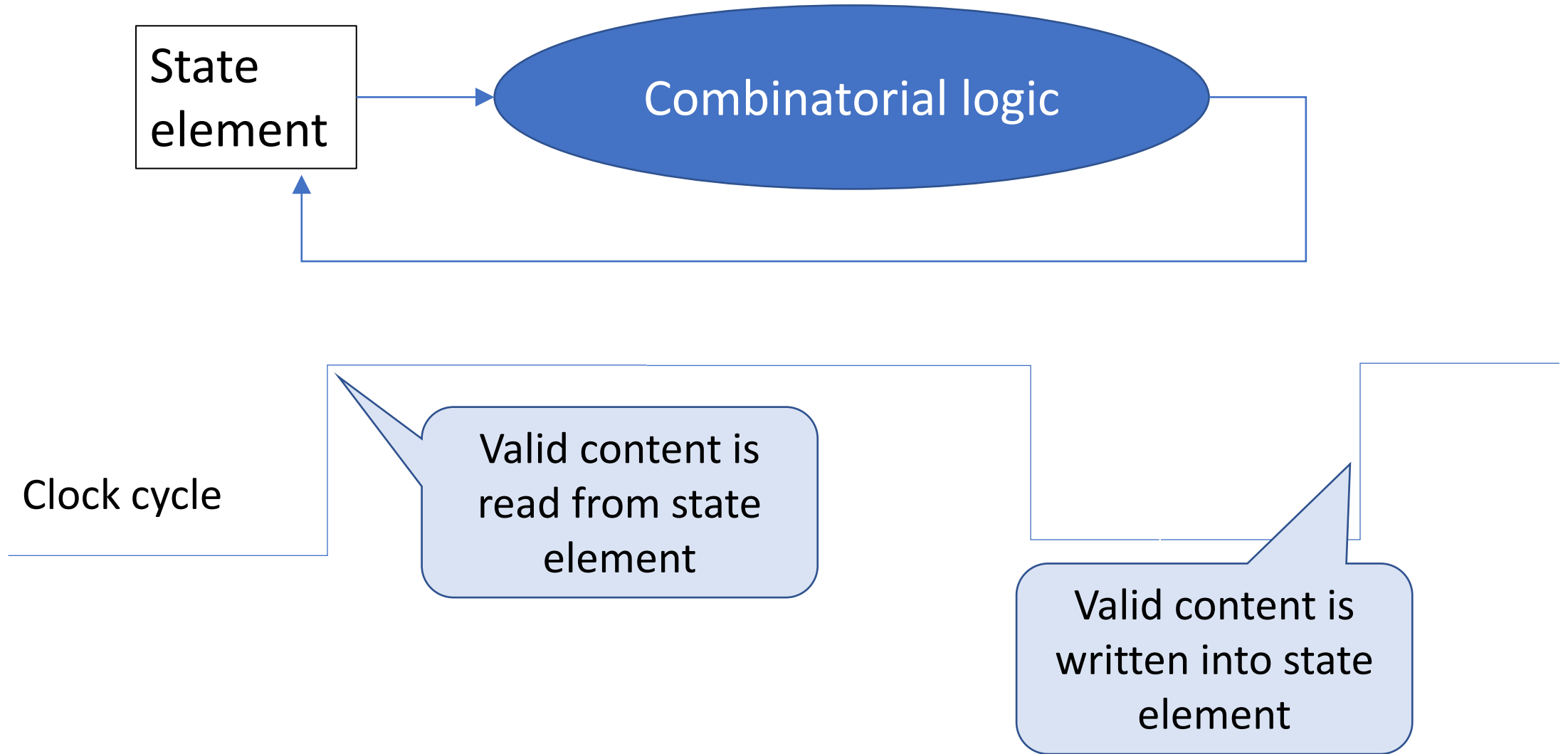
Sequential logic requires clock



Sequential logic requires clock

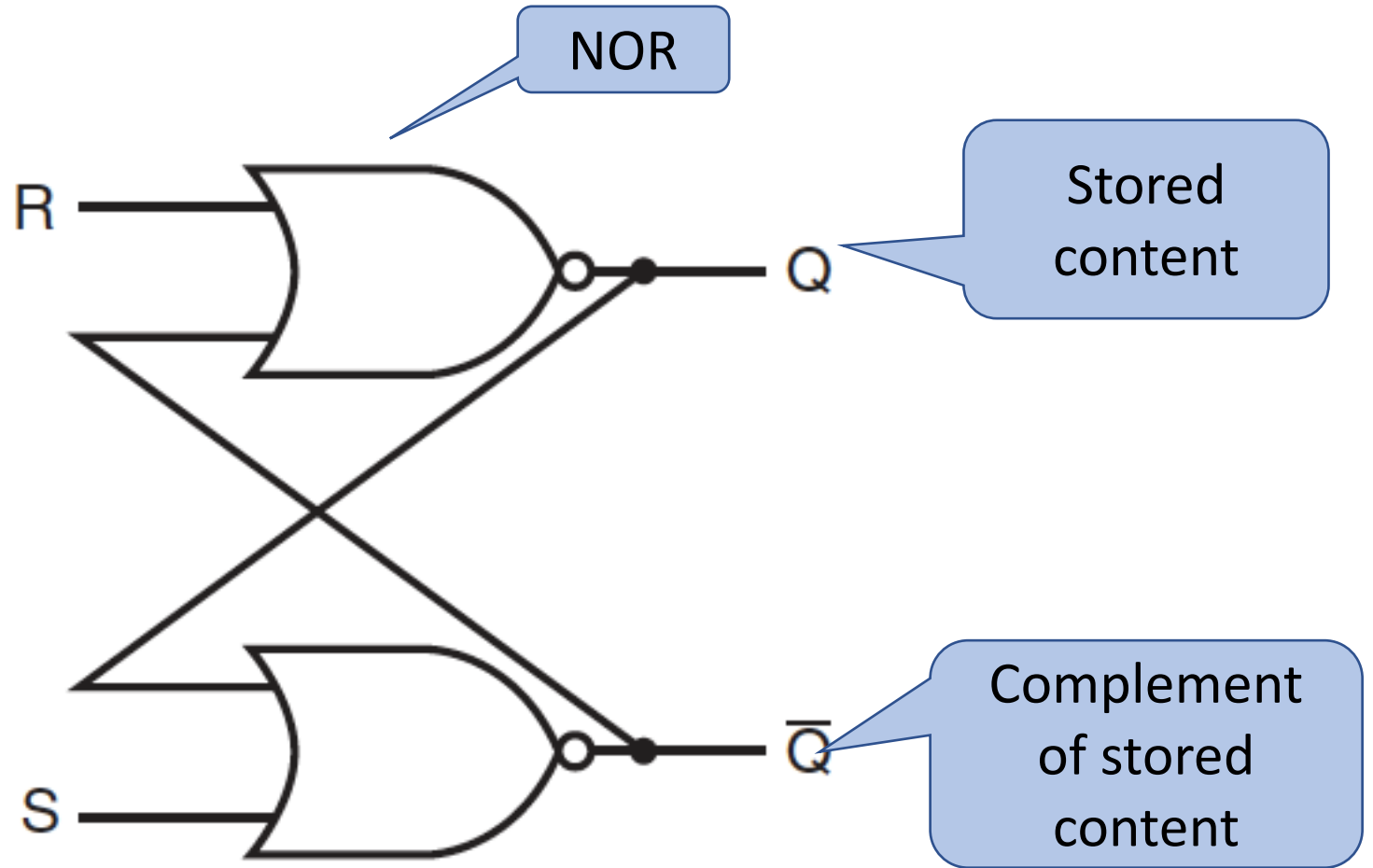


Sequential logic requires clock



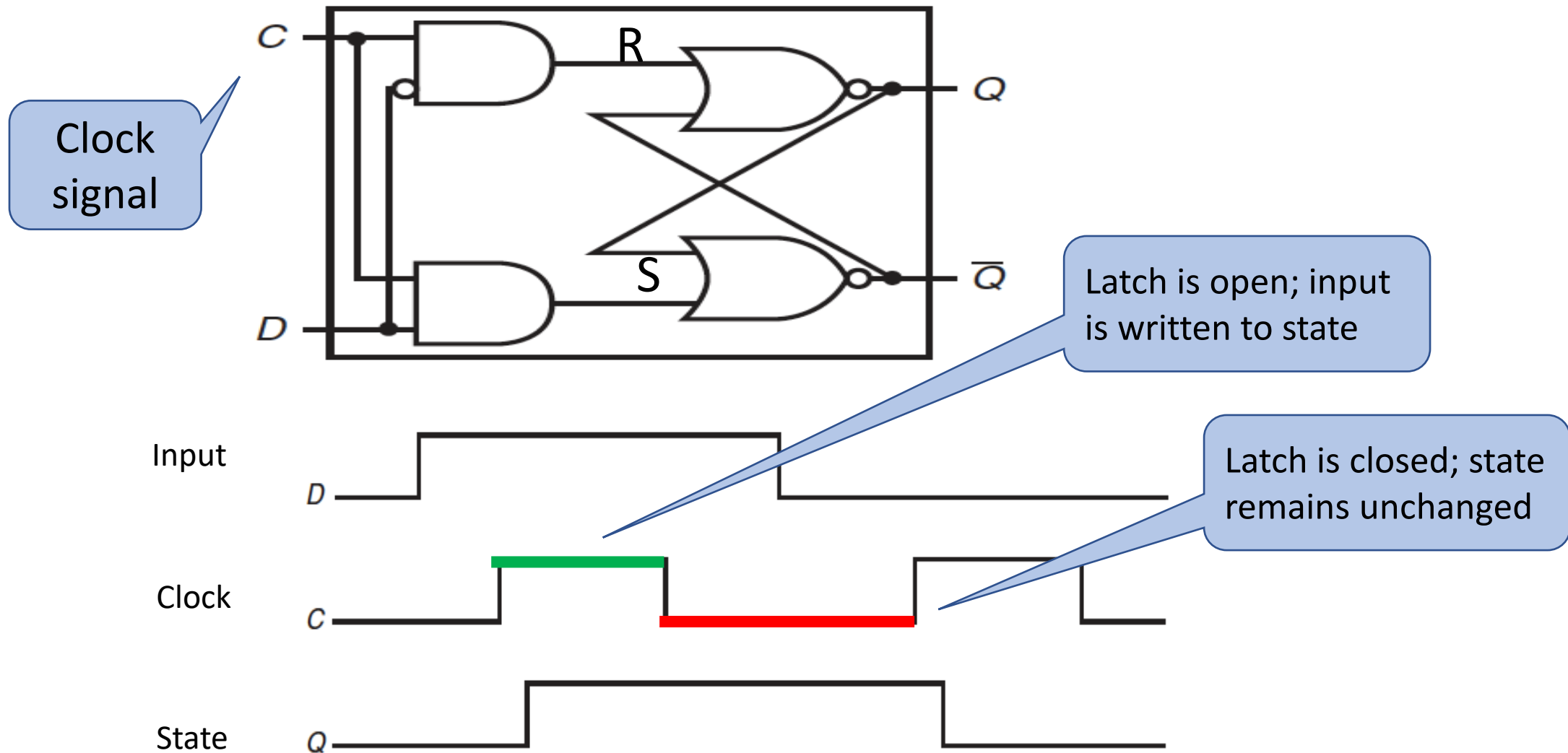
Memory (state) elements: unlocked S-R Latch

S (set)	R (reset)	
0	0	
1	0	
0	1	
1	1	



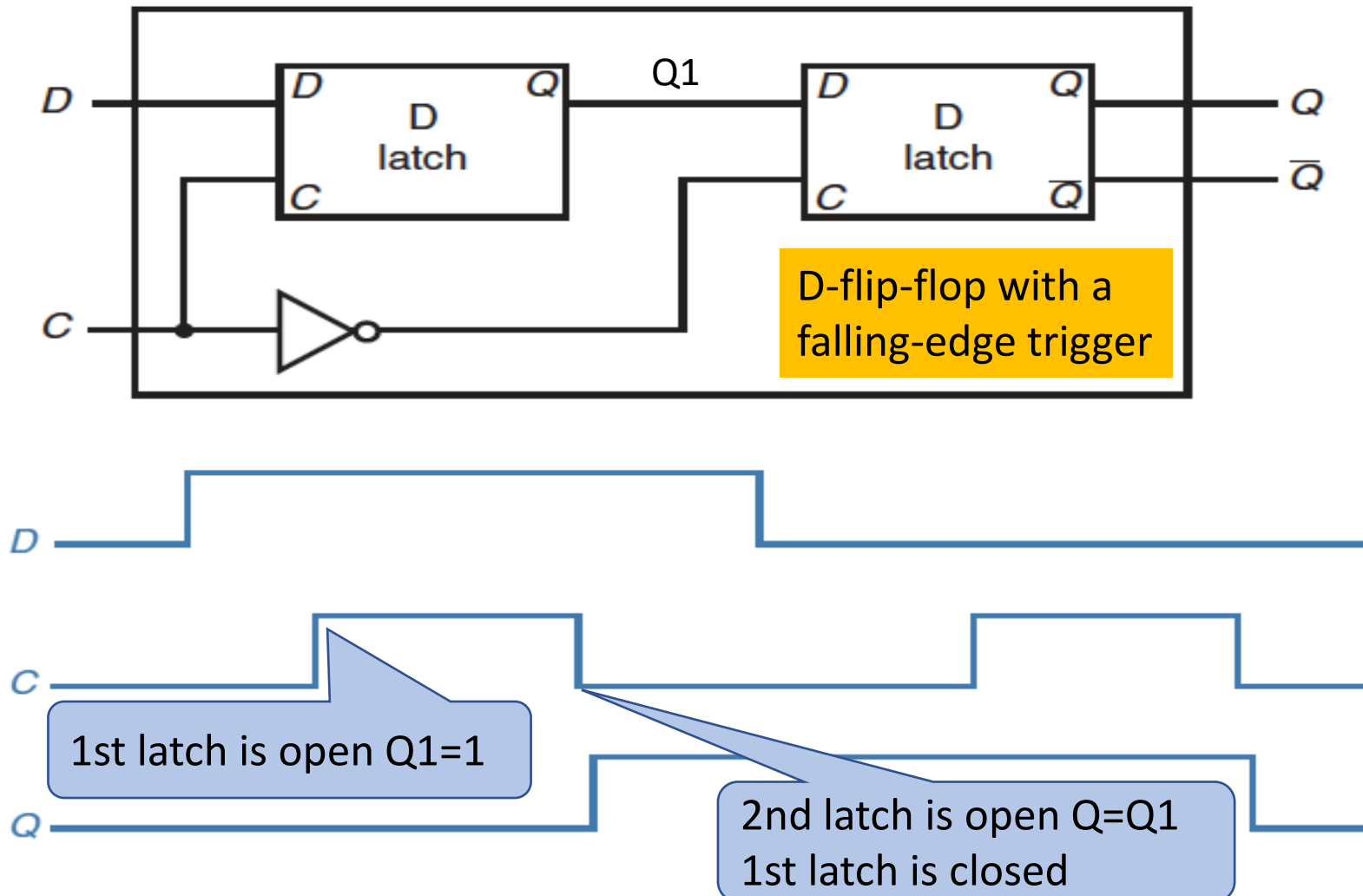
Memory element: clocked D latch

- D latch: state is changed as long as clock is asserted



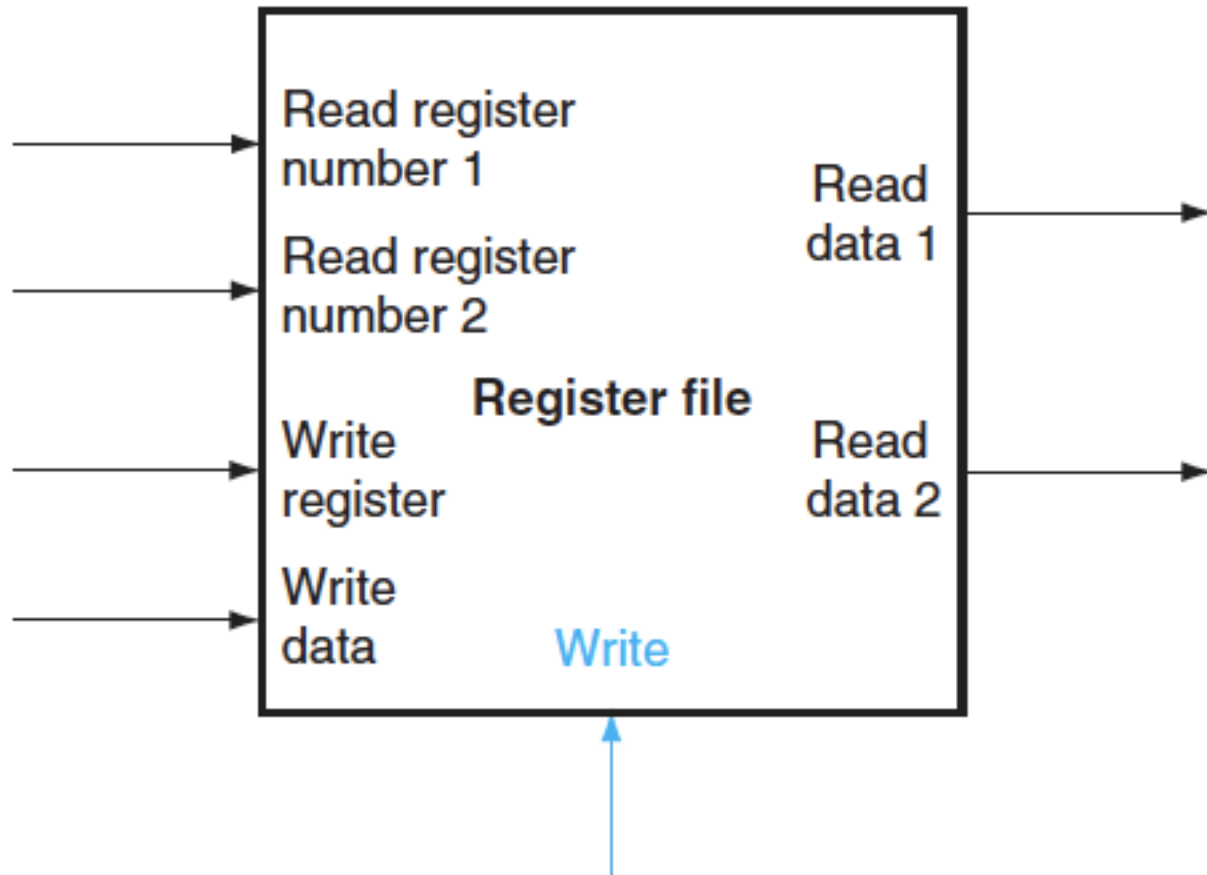
Memory element: Flip-flop

- Flip-flop: state is changed only on (rising or falling) clock edge

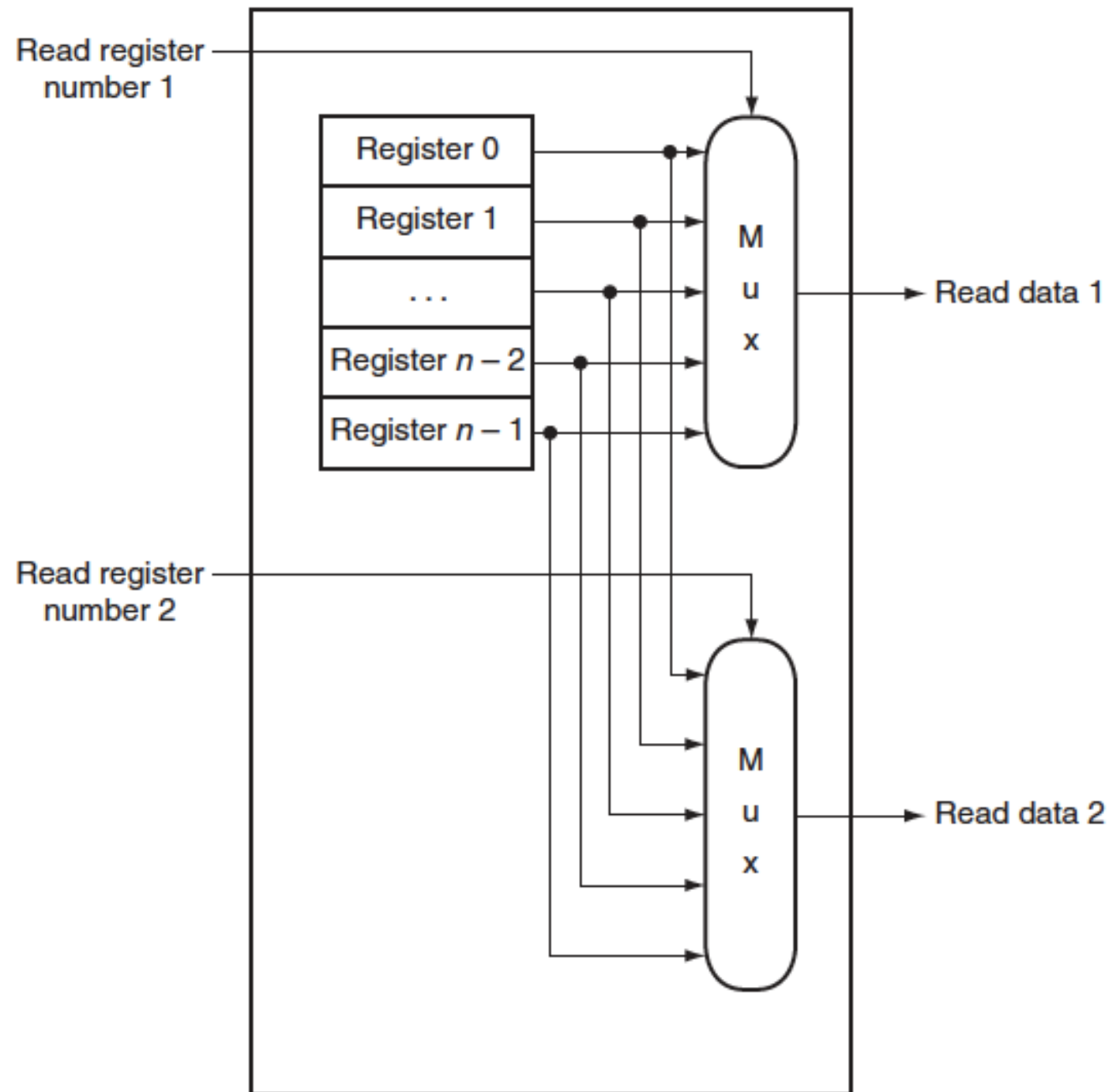


Memory element: Register file

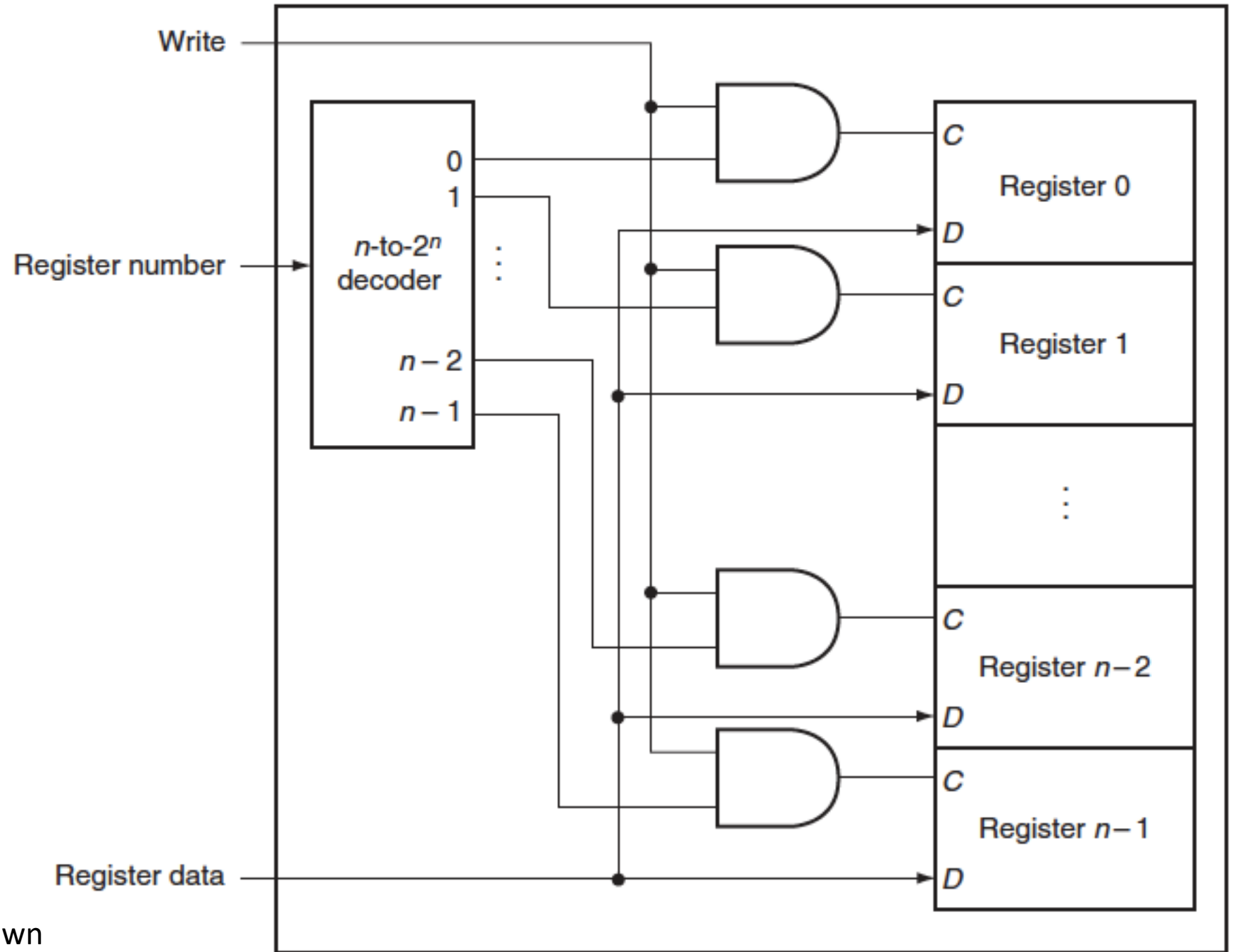
- Register file: a set of registers that can be read and written



☹️ Why reading two registers at a time?



Clock signal is assumed and not drawn



Clock signal is assumed and not drawn

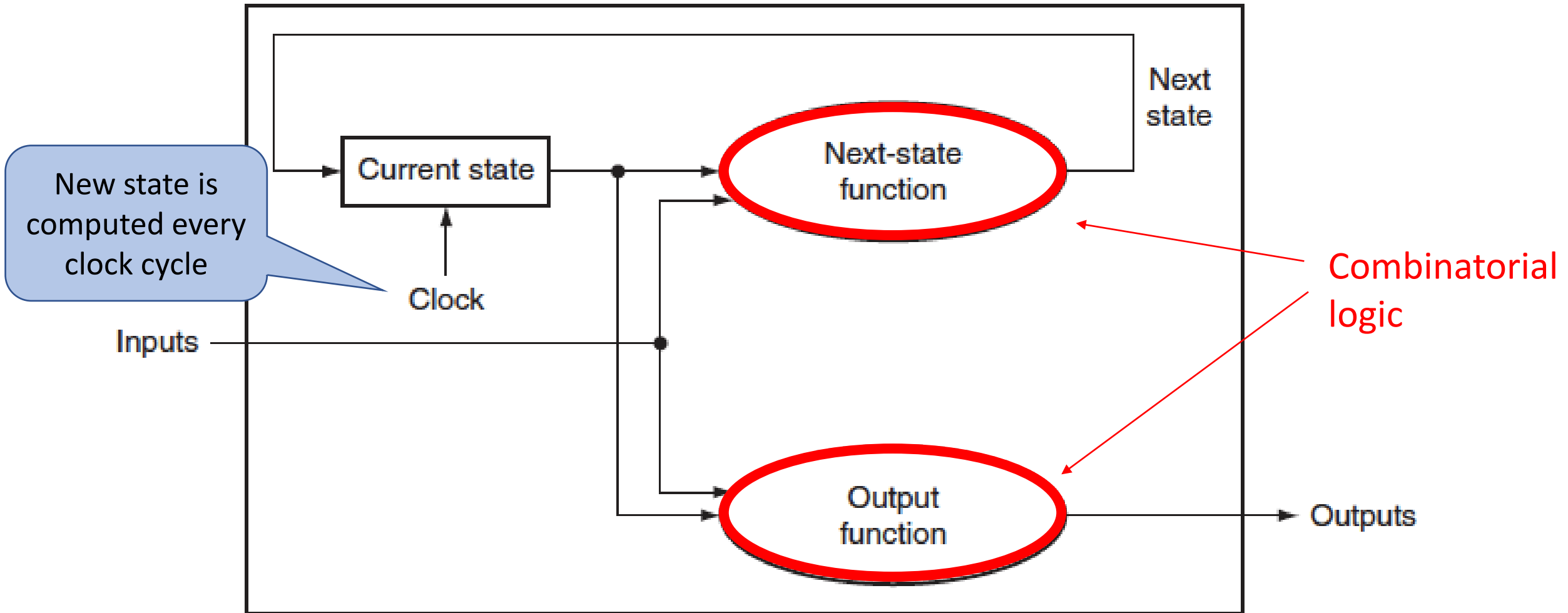
Register file

- What if the same register is read and written in the same clock cycle?
 - Return register value written in an earlier cycle
 - Write of new value occurs on the clock edge (at the end of the current cycle)
- Some register file can read value currently being written
 - Requires additional logic in the register file

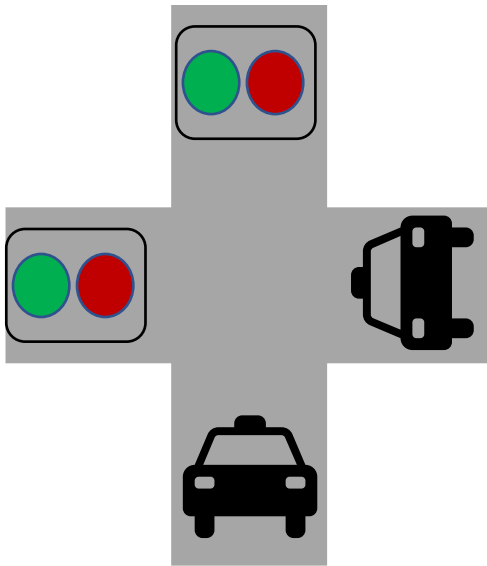
Finite State Machine

- Combinatorial logic \rightarrow truth table
- Sequential logic \rightarrow F(inite) S(tate) M(achine)
 - Input and current state determine next state and outputs

Finite State Machine



FSM example: traffic light control



State:

NSgreen: traffic light is green in N-S (red in E-W)

EWgreen: traffic light is green in E-W (red in N-S)

Inputs:

NScar: car detected in N-S

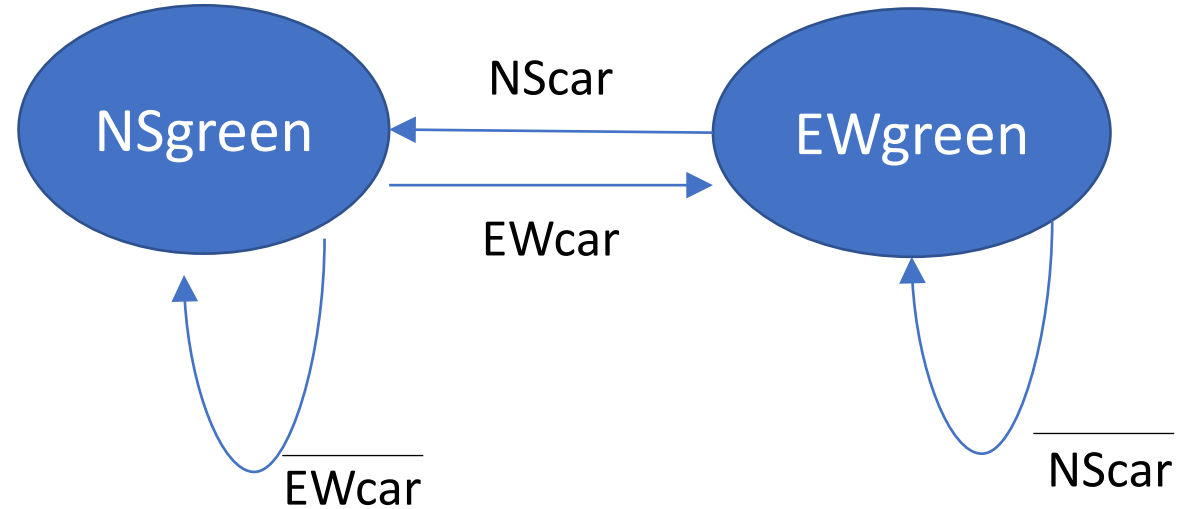
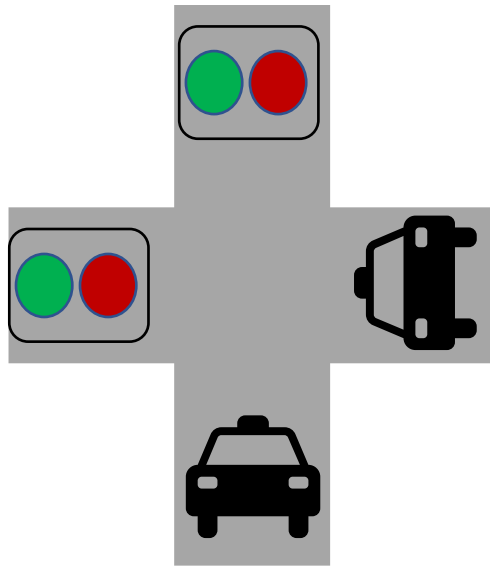
EWcar: car detected in E-W

Outputs:

NSlite: 1 if state=NSgreen

EWlite: 1 if state=EWgreen

FSM example: traffic light control



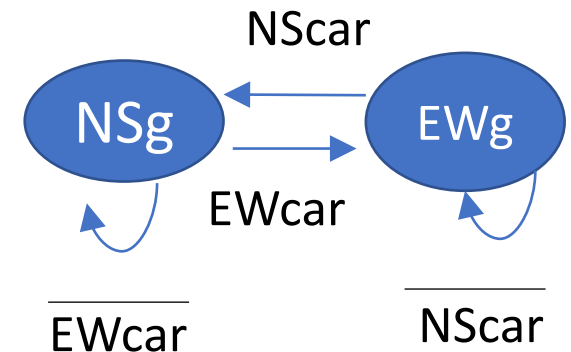
Clock cycles once every 30 seconds

FSM example: traffic light

- FSM is determined by NextState function and Output function

How many bits
needed to
represent state?

	Inputs		
		EWcar	Next state
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen



FSM example: traffic light

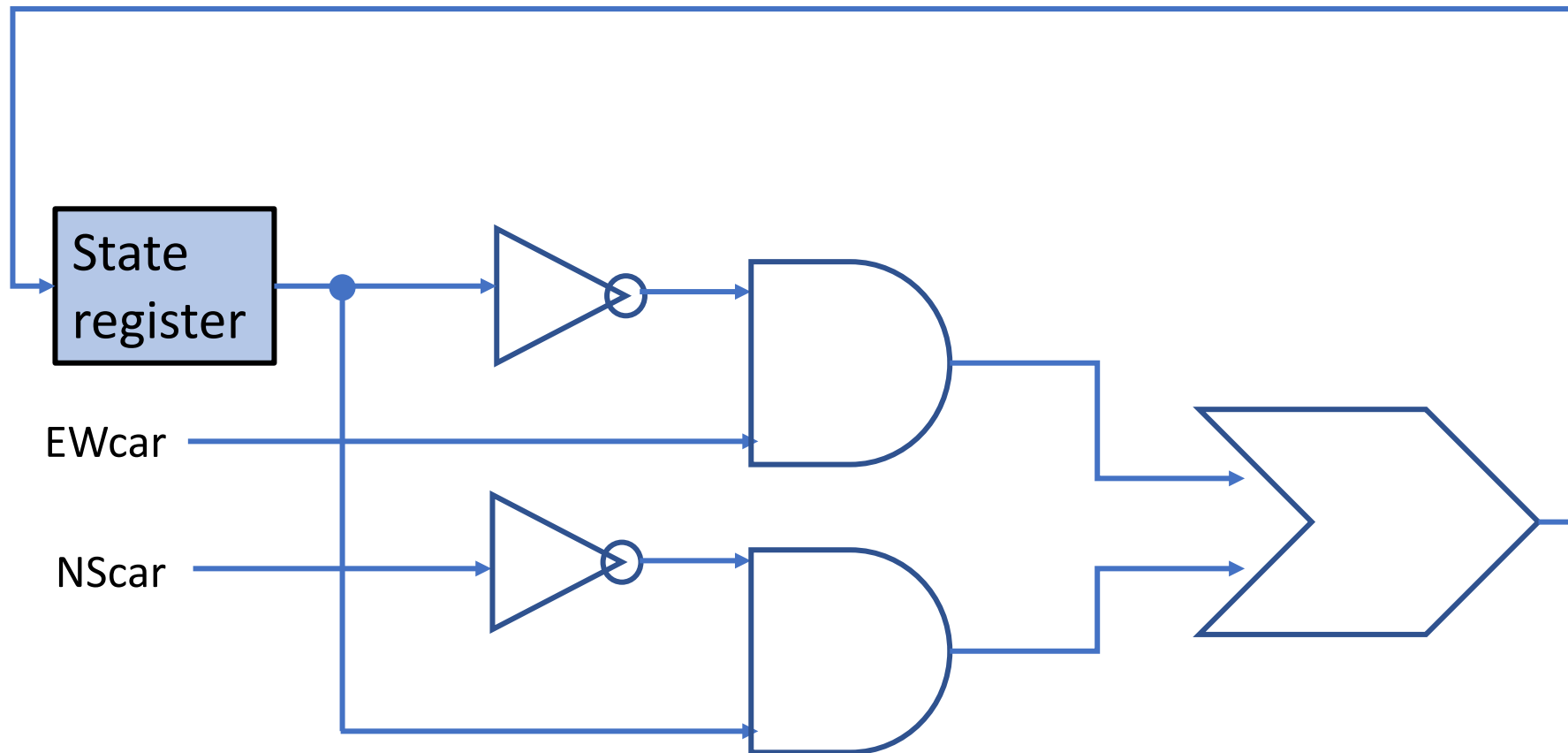
- FSM is determined by NextState function and Output function

Current State	Inputs		
	NScar	EWcar	Next state
0 (Nsgreen)	0	0	0 (Nsgreen)
0 (Nsgreen)	0	1	1 (Ewgreen)
0 (Nsgreen)	1	0	0 (Nsgreen)
0 (Nsgreen)	1	1	1 (Ewgreen)
1 (Ewgreen)	0	0	1 (Ewgreen)
1 (Ewgreen)	0	1	1 (Ewgreen)
1 (Ewgreen)	1	0	0 (Nsgreen)
1 (Ewgreen)	1	1	0 (Nsgreen)

$$\begin{aligned}
 \text{Next} = & \overline{\text{Curr}} \cdot \overline{\text{NScar}} \cdot \text{EWcar} \\
 & + \overline{\text{Curr}} \cdot \text{NScar} \cdot \text{EWcar} \\
 & + \text{Curr} \cdot \overline{\text{NScar}} \cdot \overline{\text{EWcar}} \\
 & + \text{Curr} \cdot \overline{\text{NScar}} \cdot \text{EWCar}
 \end{aligned}$$

FSM traffic light: next state function

$$Next = \overline{Curr} \cdot EWcar + Curr \cdot \overline{NScar}$$



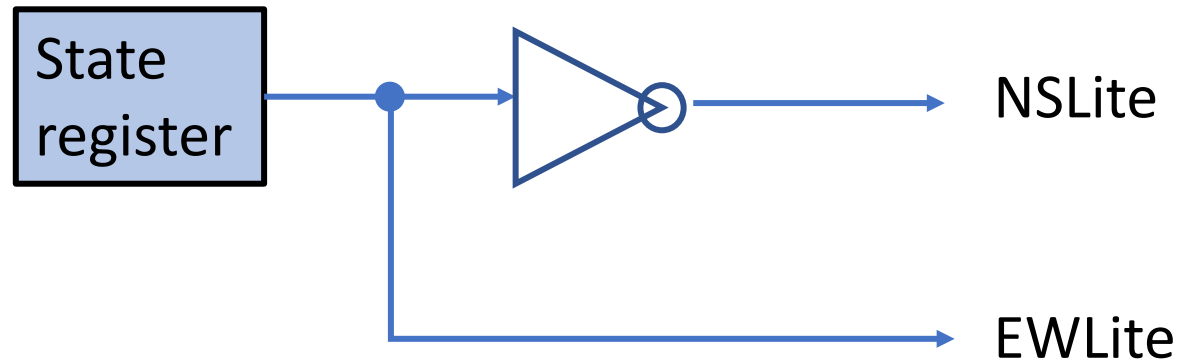
FSM traffic light: output function

	Outputs	
	NSlite	EWlite
0 NSgreen	1	0
1 EWgreen	0	1

$$\text{NSLite} = \overline{\text{Curr}}$$

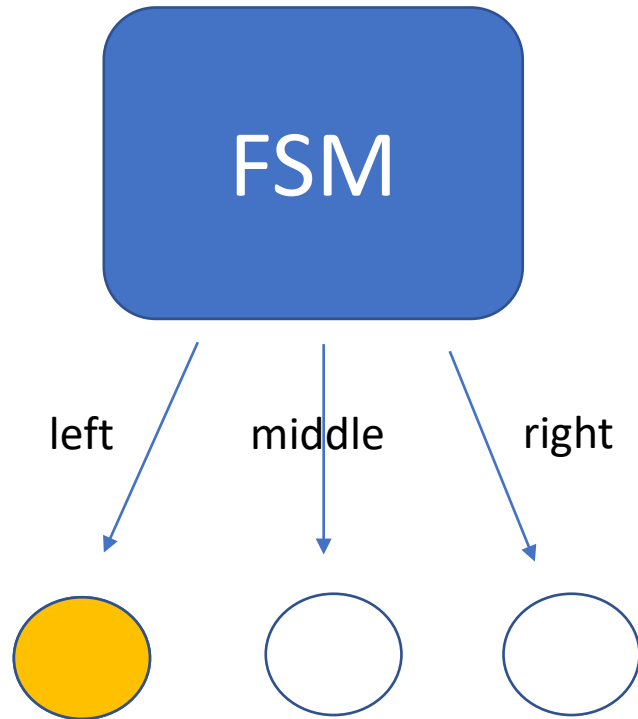
$$\text{EWLite} = \text{Curr}$$

FSM traffic light: output function



Another FSM example: Electronic eye

State transition diagram?

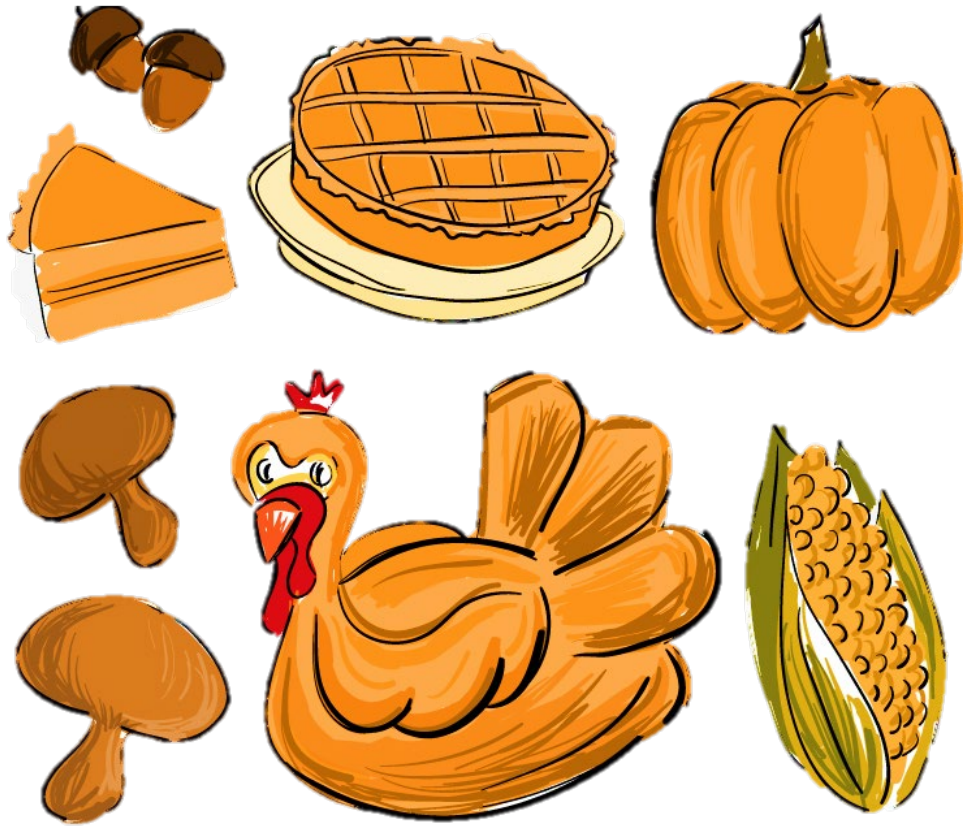


Lights are lit from left to right, then right to left and so on

Summary

- Memory (state) elements
 - Requires a clock signal to know when to update state value
 - Unclocked S-R latch → Clocked D latch → Flip-flop
- Sequential logic
 - Finite state machine
 - Decompose into two CL functions
 - Next state function: compute next state value based on current state value and inputs
 - Output function: compute output based on current state value and inputs

Happy Thanksgiving!



Stay safe!