

# Chapter 3

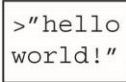


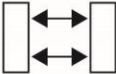
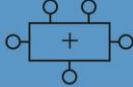

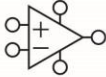

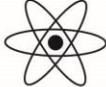
## ***Digital Design and Computer Architecture, 2<sup>nd</sup> Edition***

---

David Money Harris and Sarah L. Harris

# Chapter 3 :: Topics

- Introduction
- Latches and Flip-Flops
- Synchronous Logic Design
- Finite State Machines
- Timing of Sequential Logic
- Parallelism

Application Software	
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Analog Circuits	
Devices	
Physics	

# Introduction

- Outputs of sequential logic depend on current *and* prior input values – it has ***memory***.
- Some definitions:
  - **State:** all the information about a circuit necessary to explain its future behavior
  - **Latches and flip-flops:** state elements that store one bit of state
  - **Synchronous sequential circuits:** combinational logic followed by a bank of flip-flops

# Sequential Circuits

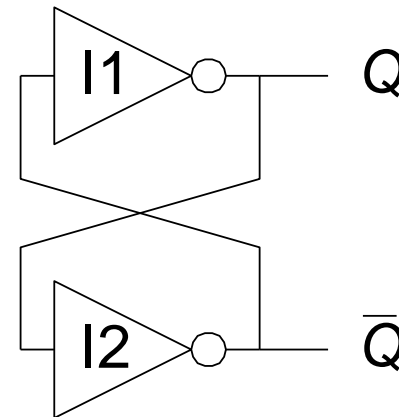
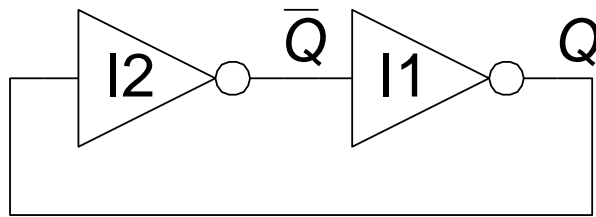
- Give sequence to events
- Have memory (short-term)
- Use feedback from output to input to store information

# State Elements

- The state of a circuit influences its future behavior
- State elements store state
  - Bistable circuit
  - SR Latch
  - D Latch
  - D Flip-flop

# Bistable Circuit

- Fundamental building block of other state elements
- Two outputs:  $Q, \bar{Q}$
- No inputs

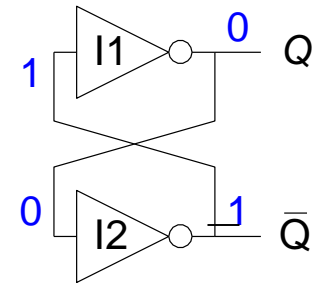


# Bistable Circuit Analysis

- Consider the two possible cases:

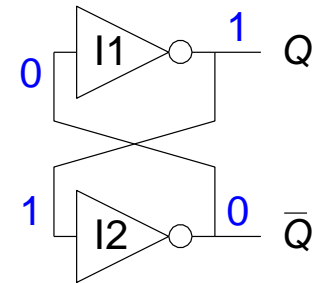
–  $Q = 0$ :

then  $\bar{Q} = 1$ ,  $Q = 0$  (consistent)



–  $Q = 1$ :

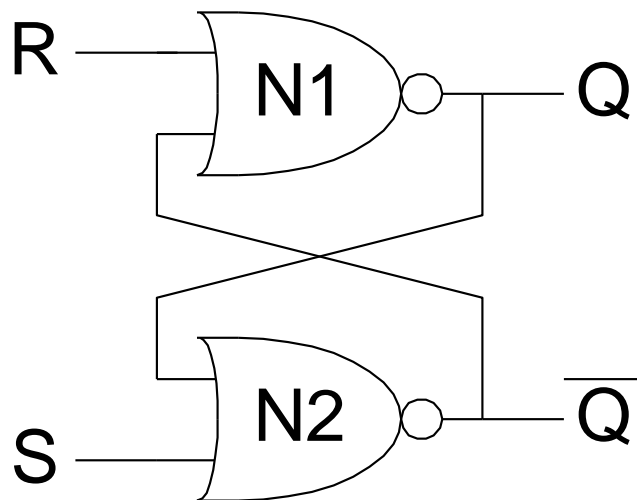
then  $\bar{Q} = 0$ ,  $Q = 1$  (consistent)



- Stores 1 bit of state in the state variable,  $Q$  (or  $\bar{Q}$ )
- But there are **no inputs to control the state**

# SR (Set/Reset) Latch

- SR Latch



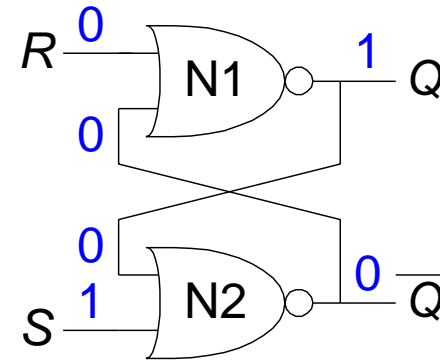
- Consider the four possible cases:
  - $S = 1, R = 0$
  - $S = 0, R = 1$
  - $S = 0, R = 0$
  - $S = 1, R = 1$



# SR Latch Analysis

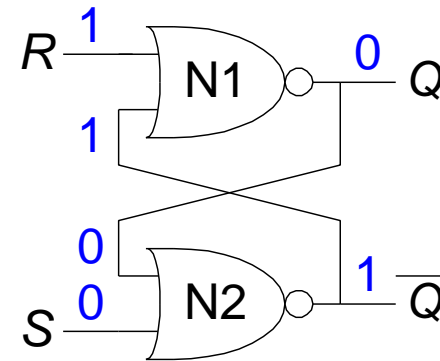
–  $S = 1, R = 0$ :

then  $Q = 1$  and  $\bar{Q} = 0$



–  $S = 0, R = 1$ :

then  $Q = 0$  and  $\bar{Q} = 1$

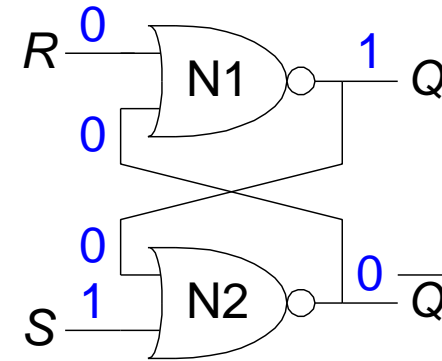


# SR Latch Analysis

–  $S = 1, R = 0$ :

then  $Q = 1$  and  $\bar{Q} = 0$

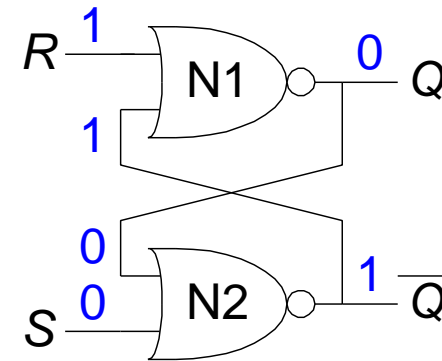
**Set the output**



–  $S = 0, R = 1$ :

then  $Q = 0$  and  $\bar{Q} = 1$

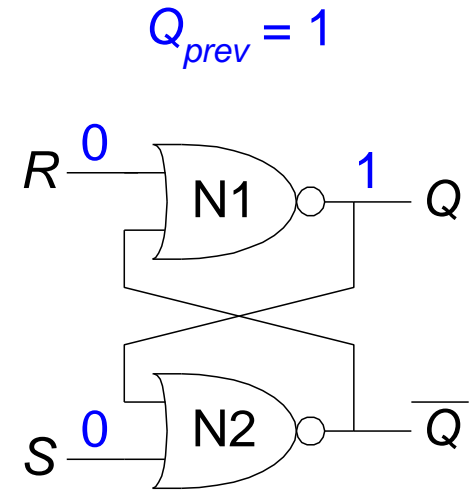
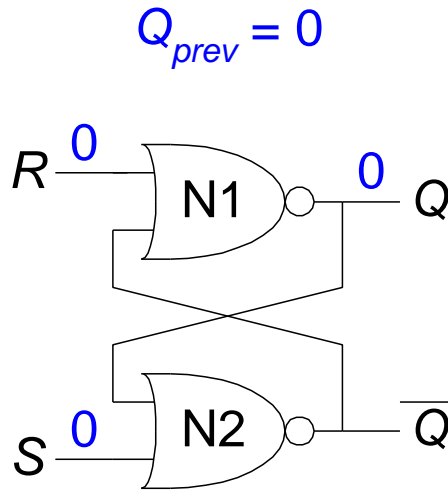
**Reset the output**



# SR Latch Analysis

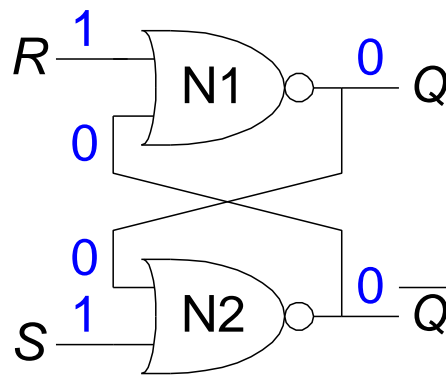
–  $S = 0, R = 0$ :

then  $Q = Q_{prev}$



–  $S = 1, R = 1$ :

then  $Q = 0, \bar{Q} = 0$

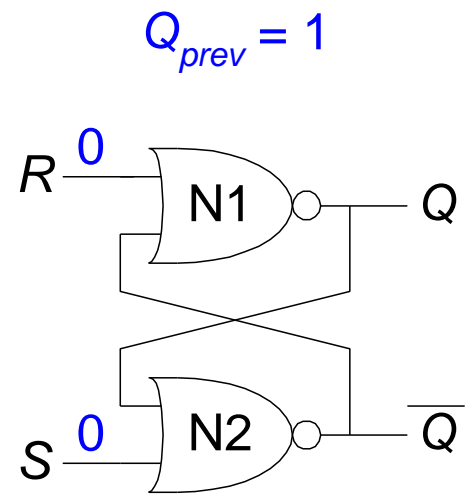
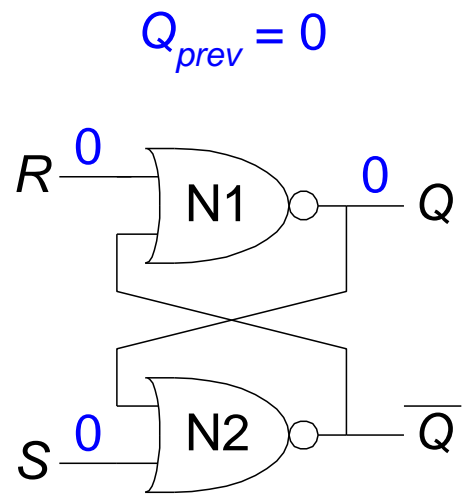


# SR Latch Analysis

–  $S = 0, R = 0$ :

then  $Q = Q_{prev}$

**Memory!**

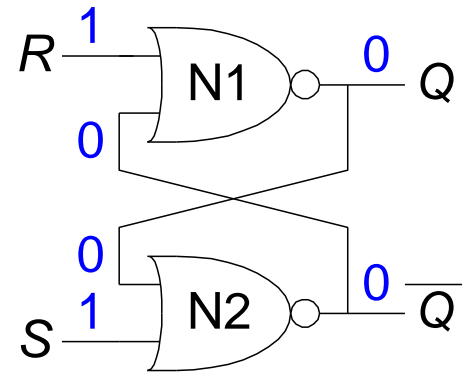


–  $S = 1, R = 1$ :

then  $Q = 0, \bar{Q} = 0$

**Invalid State**

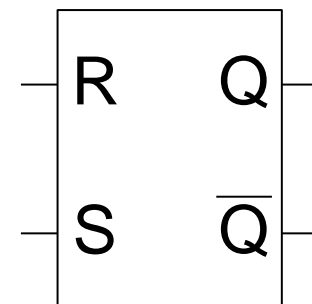
$\bar{Q} \neq \text{NOT } Q$



# SR Latch Symbol

- SR stands for Set/Reset Latch
  - Stores one bit of state ( $Q$ )
- Control what value is being stored with  $S$ ,  $R$  inputs
  - **Set:** Make the output 1  
( $S = 1, R = 0, Q = 1$ )
  - **Reset:** Make the output 0  
( $S = 0, R = 1, Q = 0$ )

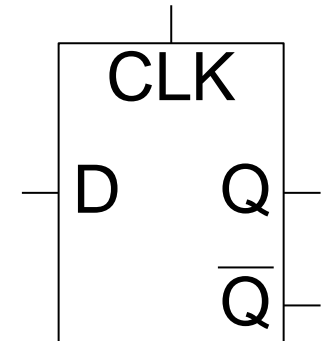
SR Latch Symbol



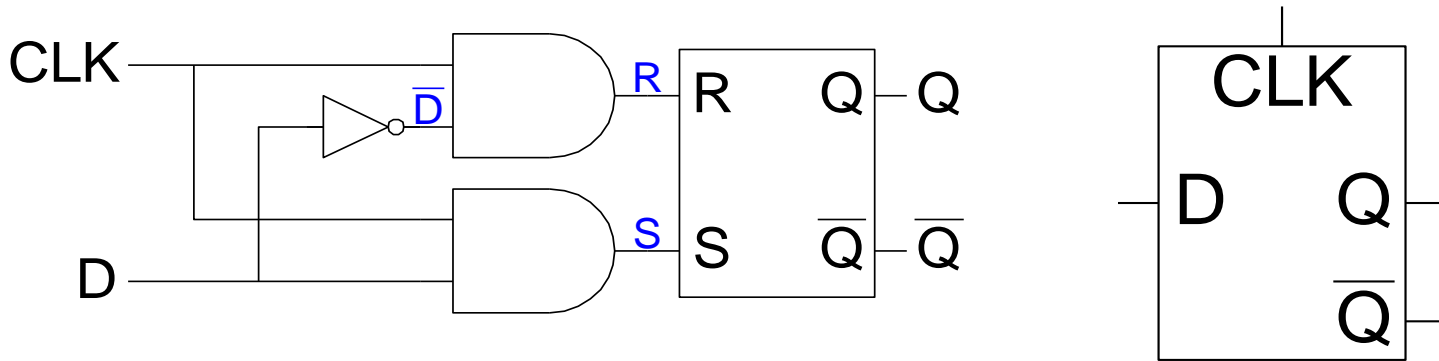
# D Latch

- Two inputs:  $CLK$ ,  $D$ 
  - $CLK$ : controls *when* the output changes
  - $D$  (the data input): controls *what* the output changes to
- Function
  - When  $CLK = 1$ ,  
 $D$  passes through to  $Q$  (*transparent*)
  - When  $CLK = 0$ ,  
 $Q$  holds its previous value (*opaque*)
- Avoids invalid case when  
 $Q \neq \text{NOT } \bar{Q}$

D Latch  
Symbol

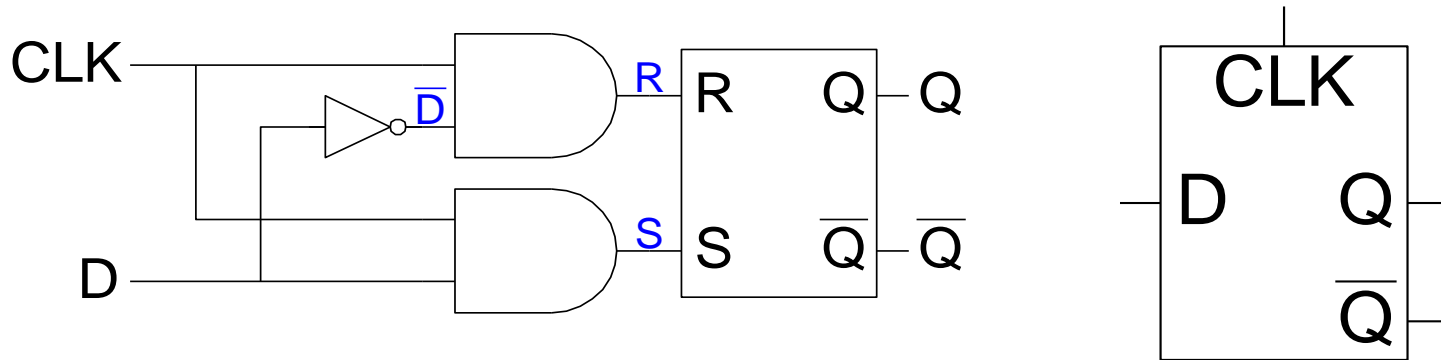


# D Latch Internal Circuit



$CLK$	$D$	$\overline{D}$	$S$	$R$	$Q$	$\overline{Q}$
0	X					
1	0					
1	1					

# D Latch Internal Circuit



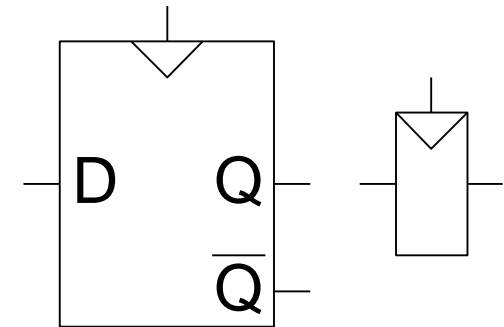
$CLK$	$D$	$\overline{D}$	$S$	$R$	$Q$	$\overline{Q}$
0	X	X	0	0	$Q_{prev}$	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0



# D Flip-Flop

- **Inputs:**  $CLK$ ,  $D$
- **Function**
  - Samples  $D$  on rising edge of  $CLK$ 
    - When  $CLK$  rises from 0 to 1,  $D$  passes through to  $Q$
    - Otherwise,  $Q$  holds its previous value
  - $Q$  changes only on rising edge of  $CLK$
- Called *edge-triggered*
- Activated on the clock edge

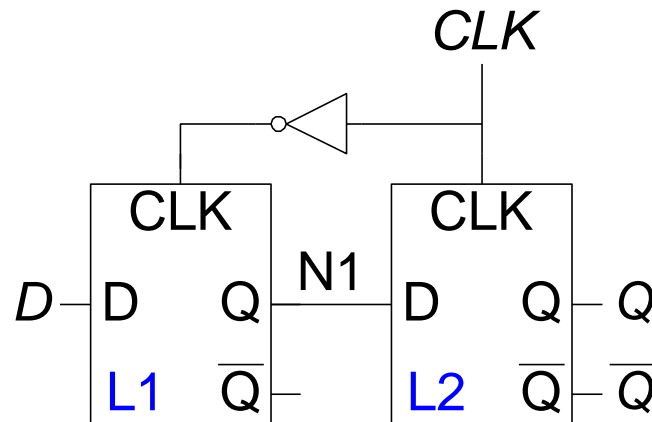
D Flip-Flop Symbols



# D Flip-Flop Internal Circuit

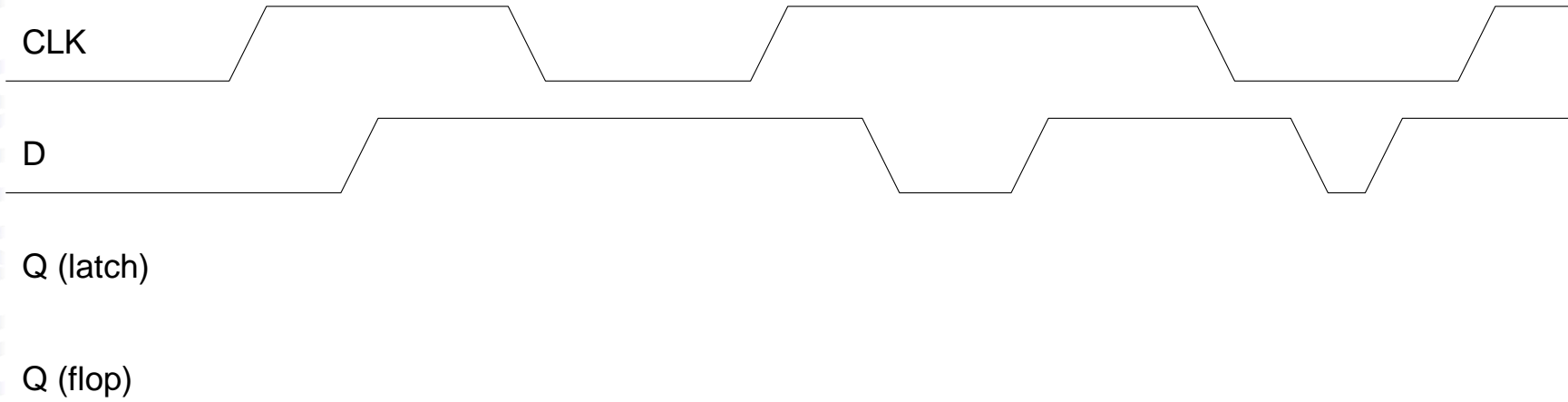
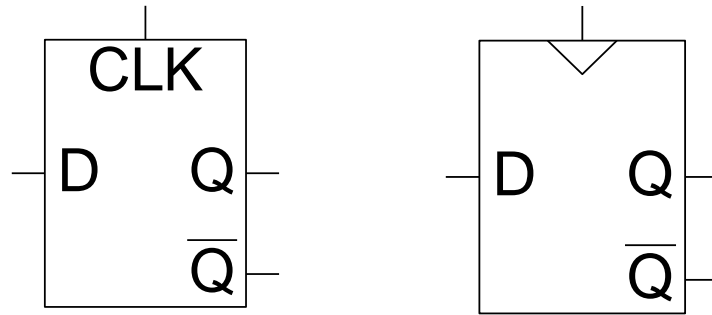
- Two back-to-back latches (L1 and L2) controlled by complementary clocks

- When  $CLK = 0$ 
  - L1 is transparent
  - L2 is opaque
  - $D$  passes through to N1

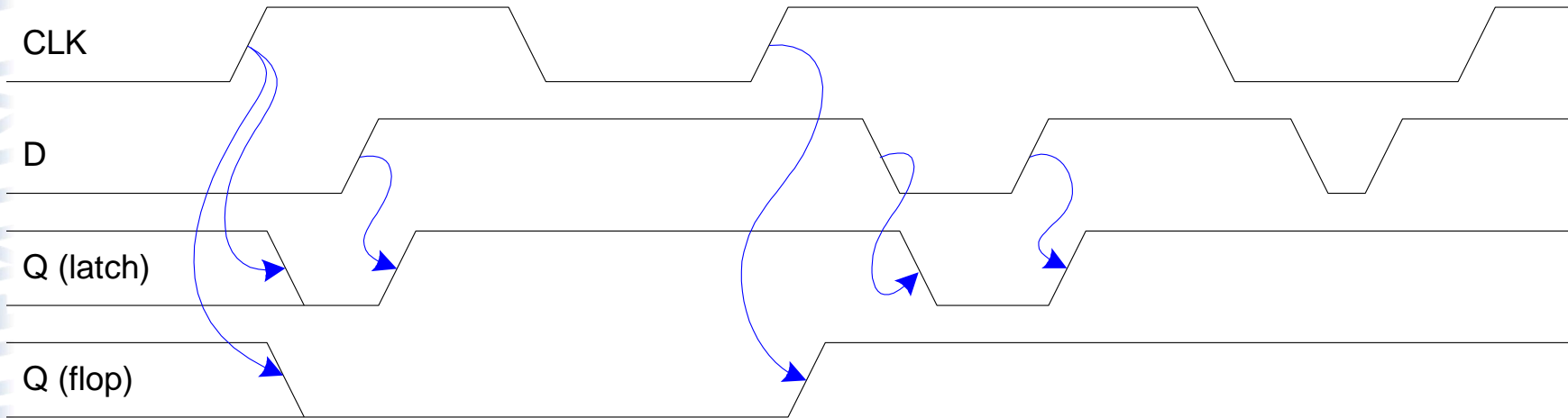
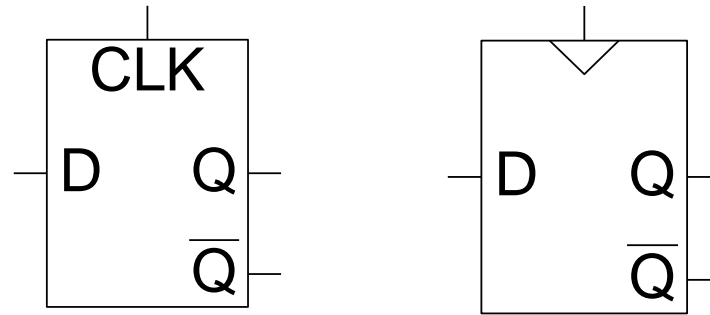


- When  $CLK = 1$ 
  - L2 is transparent
  - L1 is opaque
  - N1 passes through to  $Q$
- Thus, on the edge of the clock (when  $CLK$  rises from  $0 \rightarrow 1$ )
  - $D$  passes through to  $Q$

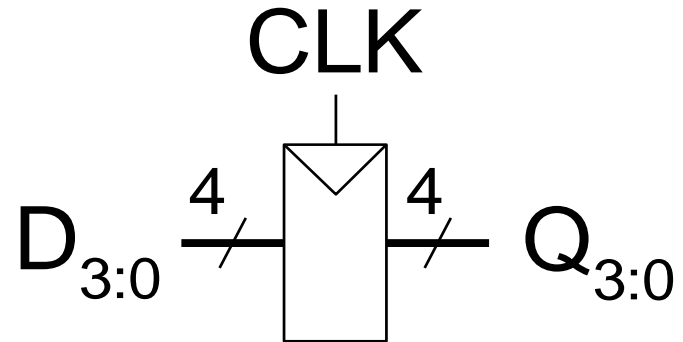
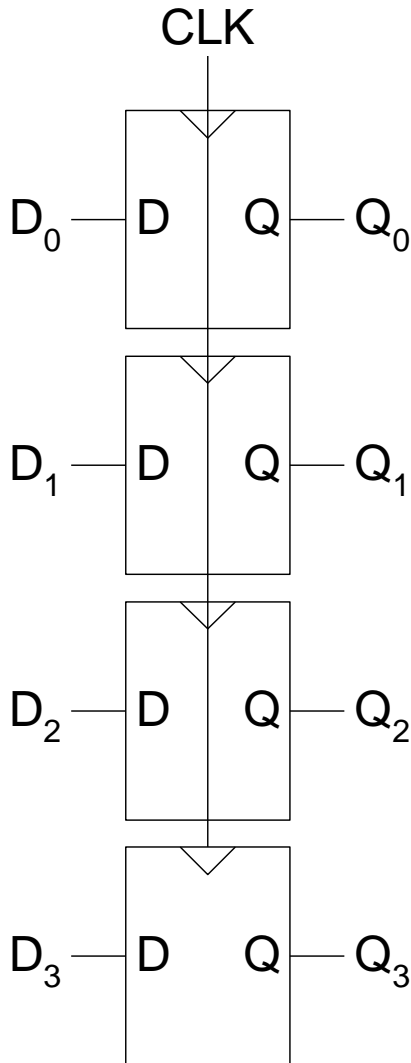
# D Latch vs. D Flip-Flop



# D Latch vs. D Flip-Flop



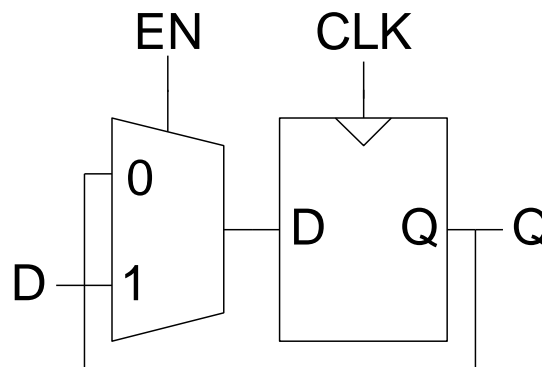
# Registers



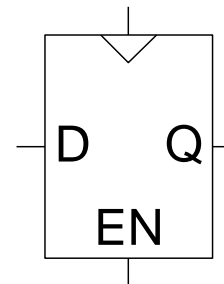
# Enabled Flip-Flops

- **Inputs:**  $CLK$ ,  $D$ ,  $EN$ 
  - The enable input ( $EN$ ) controls when new data ( $D$ ) is stored
- **Function**
  - $EN = 1$ :  $D$  passes through to  $Q$  on the clock edge
  - $EN = 0$ : the flip-flop retains its previous state

Internal Circuit



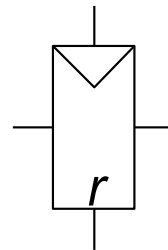
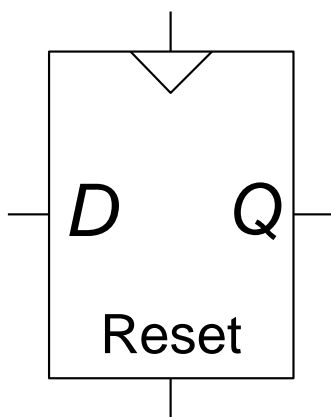
Symbol



# Resettable Flip-Flops

- **Inputs:** *CLK, D, Reset*
- **Function:**
  - *Reset = 1*: *Q* is forced to 0
  - *Reset = 0*: flip-flop behaves as ordinary D flip-flop

## Symbols



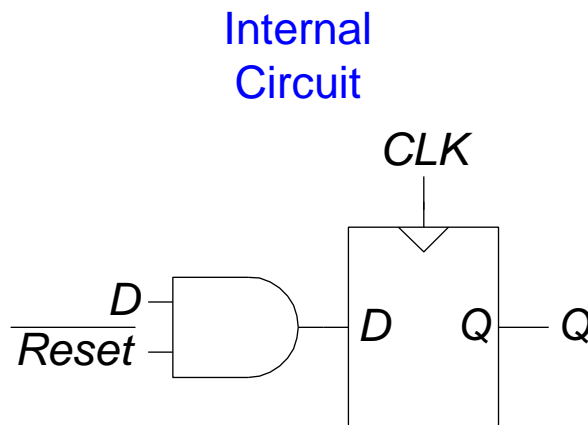
# Resettable Flip-Flops

- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when  $Reset = 1$
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop?



# Resettable Flip-Flops

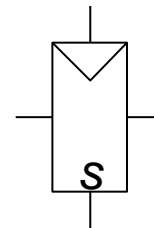
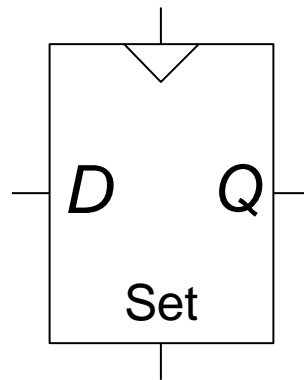
- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when  $Reset = 1$
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop?



# Settable Flip-Flops

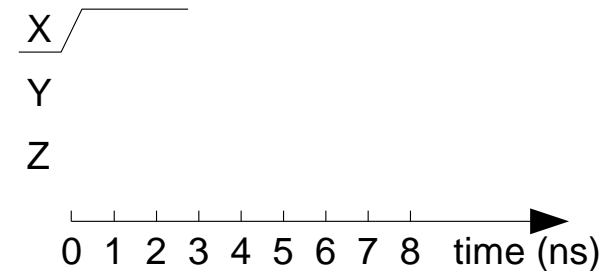
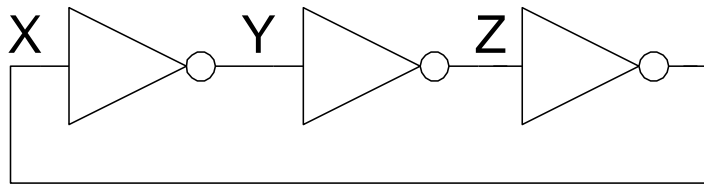
- **Inputs:**  $CLK$ ,  $D$ ,  $Set$
- **Function:**
  - $Set = 1$ :  $Q$  is set to 1
  - $Set = 0$ : the flip-flop behaves as ordinary D flip-flop

## Symbols



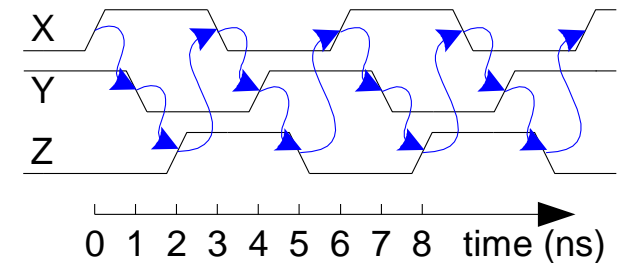
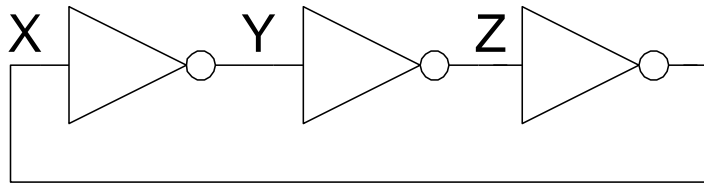
# Sequential Logic

- Sequential circuits: all circuits that aren't combinational
- A problematic circuit:



# Sequential Logic

- Sequential circuits: all circuits that aren't combinational
- A problematic circuit:



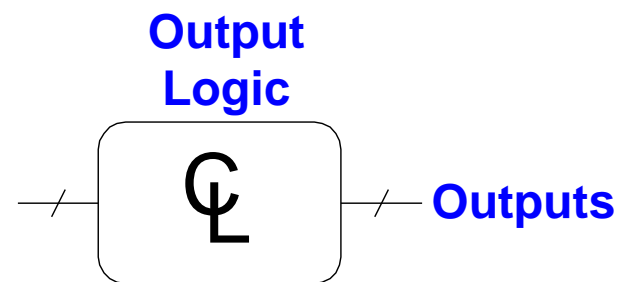
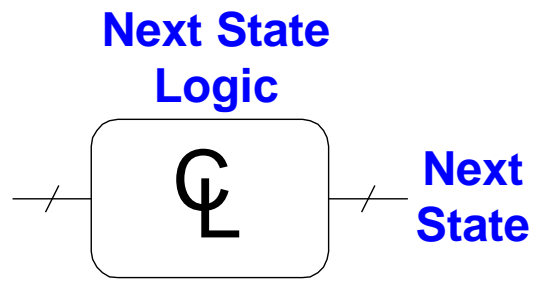
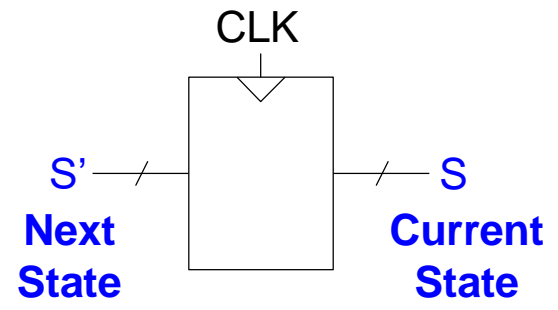
- No inputs and 1-3 outputs
- Astable circuit, oscillates
- Period depends on inverter delay
- It has a *cyclic path*: output fed back to input

# Synchronous Sequential Logic Design

- Breaks cyclic paths by **inserting registers**
- Registers contain **state** of the system
- State changes at clock edge: system **synchronized** to the clock
- **Rules** of synchronous sequential circuit composition:
  - Every circuit element is either a register or a combinational circuit
  - At least one circuit element is a register
  - All registers receive the same clock signal
  - Every cyclic path contains at least one register
- Two common synchronous sequential circuits
  - Finite State Machines (FSMs)
  - Pipelines

# Finite State Machine (FSM)

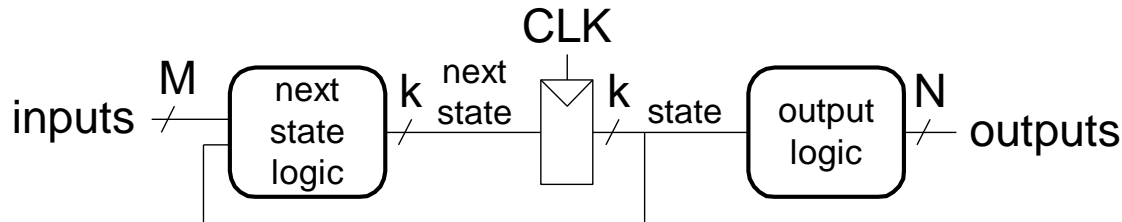
- Consists of:
  - **State register**
    - Stores current state
    - Loads next state at clock edge
  - **Combinational logic**
    - Computes the next state
    - Computes the outputs



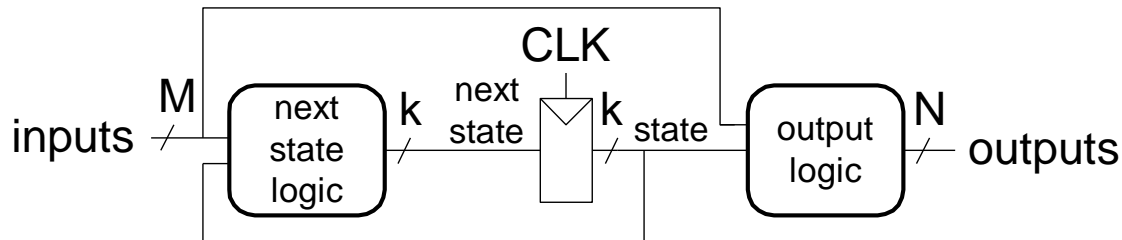
# Finite State Machines (FSMs)

- Next state determined by current state and inputs
- Two types of finite state machines differ in output logic:
  - **Moore FSM:** outputs depend only on current state
  - **Mealy FSM:** outputs depend on current state *and* inputs

Moore FSM

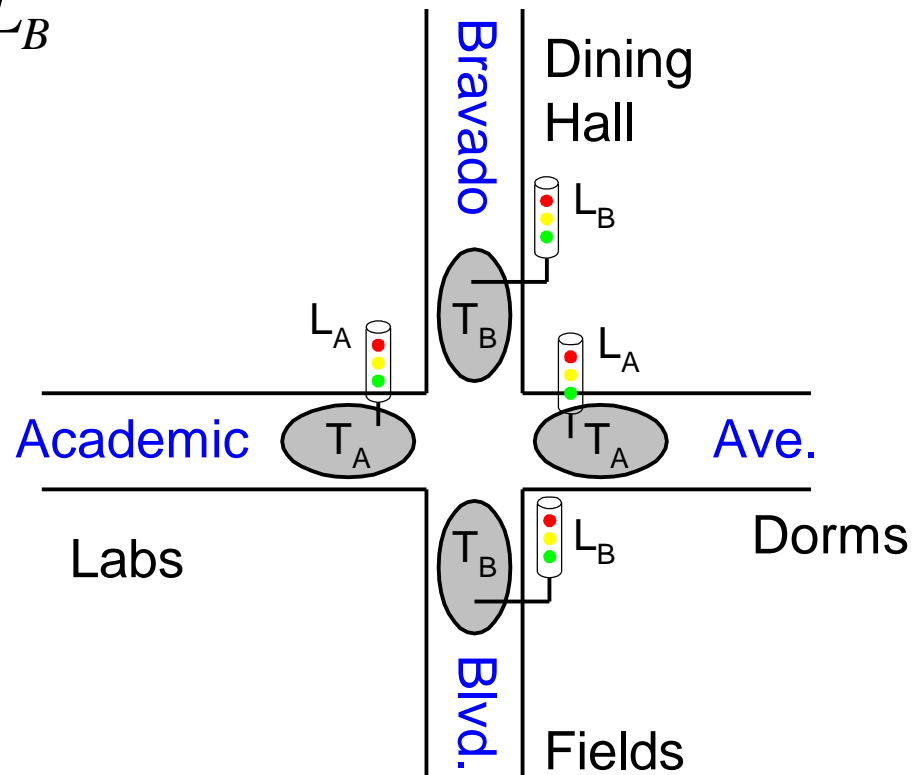


Mealy FSM



# FSM Example

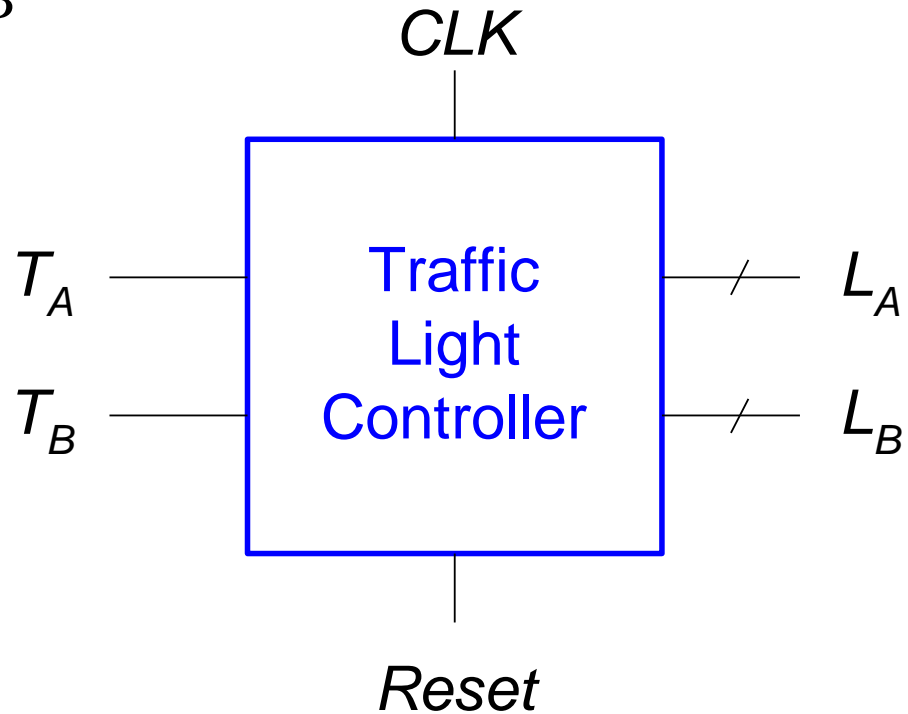
- Traffic light controller
  - Traffic sensors:  $T_A$ ,  $T_B$  (TRUE when there's traffic)
  - Lights:  $L_A$ ,  $L_B$





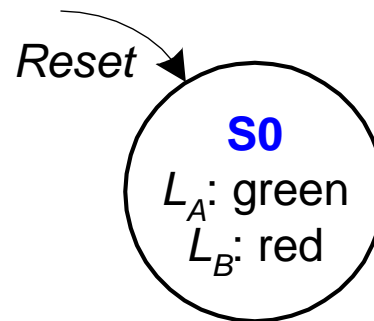
# FSM Black Box

- Inputs:  $CLK$ ,  $Reset$ ,  $T_A$ ,  $T_B$
- Outputs:  $L_A$ ,  $L_B$



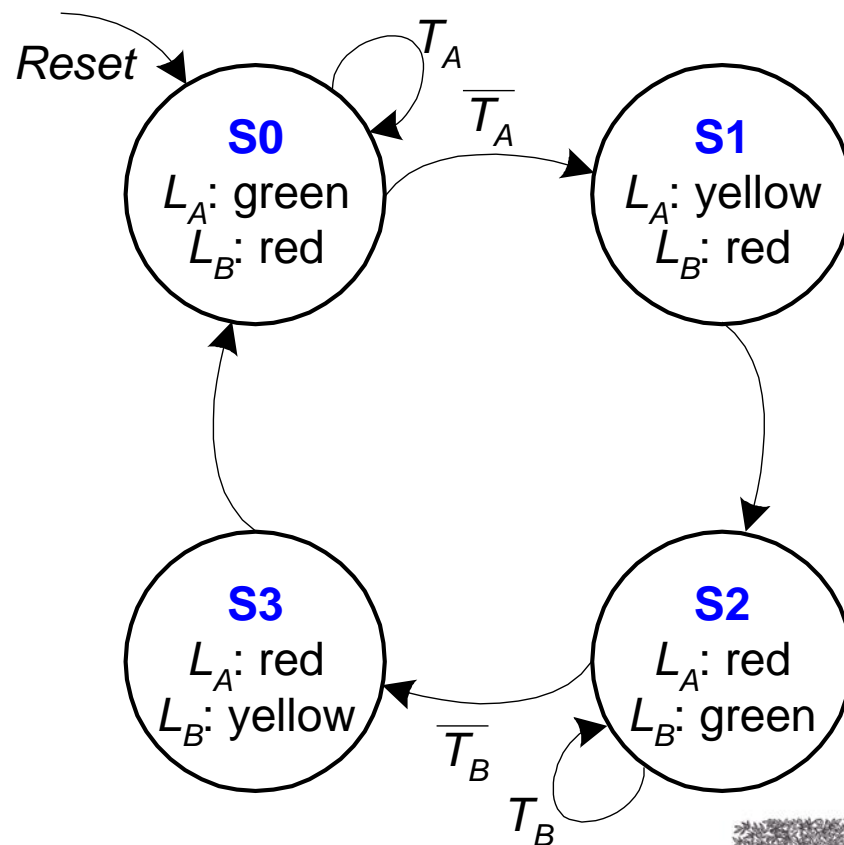
# FSM State Transition Diagram

- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs



# FSM State Transition Diagram

- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs



# FSM State Transition Table

Current State	Inputs		Next State
	$T_A$	$T_B$	
$S$	$T_A$	$T_B$	$S'$
S0	0	X	
S0	1	X	
S1	X	X	
S2	X	0	
S2	X	1	
S3	X	X	



# FSM State Transition Table

Current State	Inputs		Next State
	$T_A$	$T_B$	
$S$	$T_A$	$T_B$	$S'$
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0



# FSM Encoded State Transition Table

Current State		Inputs		Next State	
$S_1$	$S_0$	$T_A$	$T_B$	$S'_1$	$S'_0$
0	0	0	X		
0	0	1	X		
0	1	X	X		
1	0	X	0		
1	0	X	1		
1	1	X	X		

State	Encoding
S0	00
S1	01
S2	10
S3	11



# FSM Encoded State Transition Table

Current State		Inputs		Next State	
$S_1$	$S_0$	$T_A$	$T_B$	$S'_1$	$S'_0$
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1} \overline{S_0} T_A + S_1 \overline{S_0} \overline{T_B}$$

# FSM Output Table

Current State		Outputs			
$S_1$	$S_0$	$L_{A1}$	$L_{A0}$	$L_{B1}$	$L_{B0}$
0	0				
0	1				
1	0				
1	1				

Output	Encoding
green	00
yellow	01
red	10



# FSM Output Table

Current State		Outputs			
$S_1$	$S_0$	$L_{A1}$	$L_{A0}$	$L_{B1}$	$L_{B0}$
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Output	Encoding
green	00
yellow	01
red	10

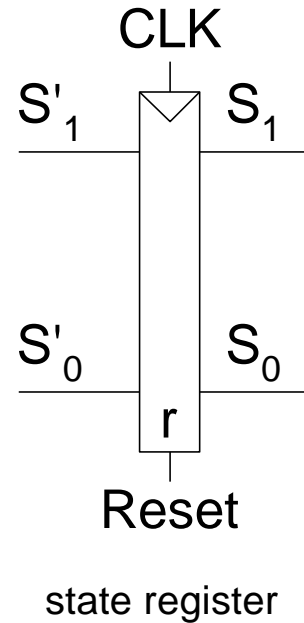
$$L_{A1} = S_1$$

$$L_{A0} = \overline{S_1}S_0$$

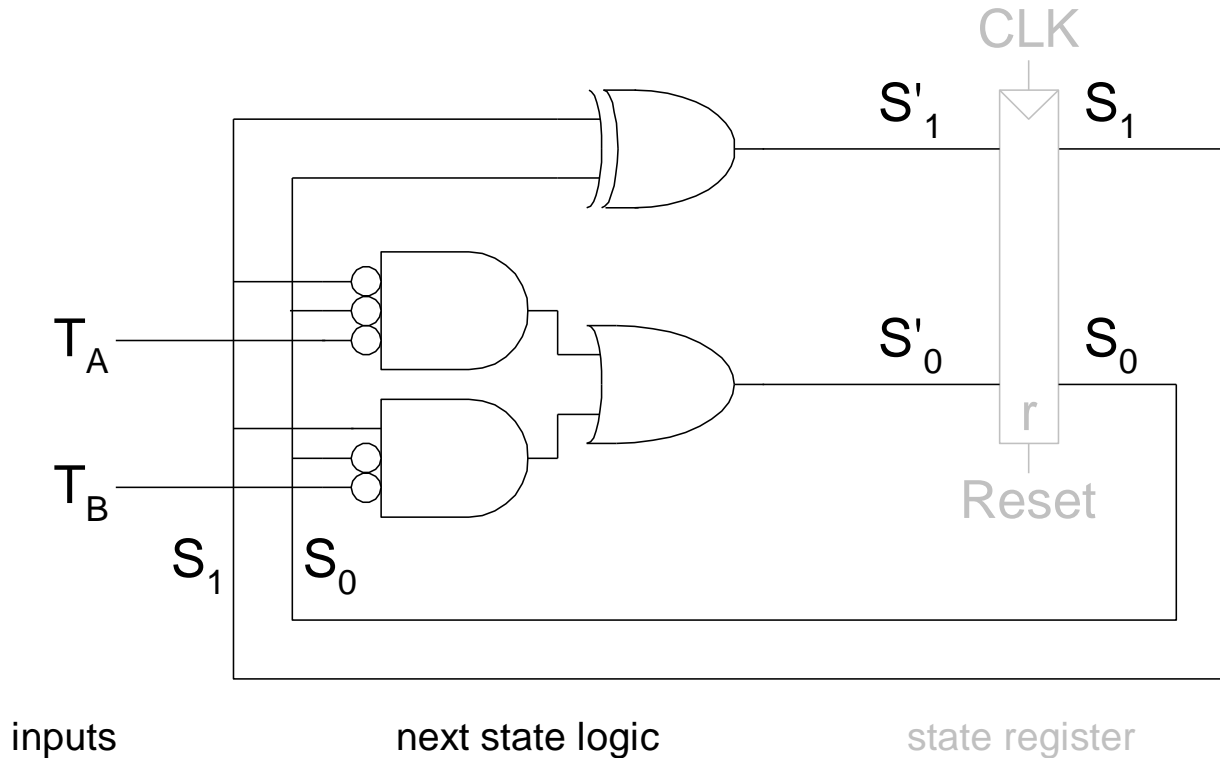
$$L_{B1} = S_1$$

$$L_{B0} = S_1S_0$$

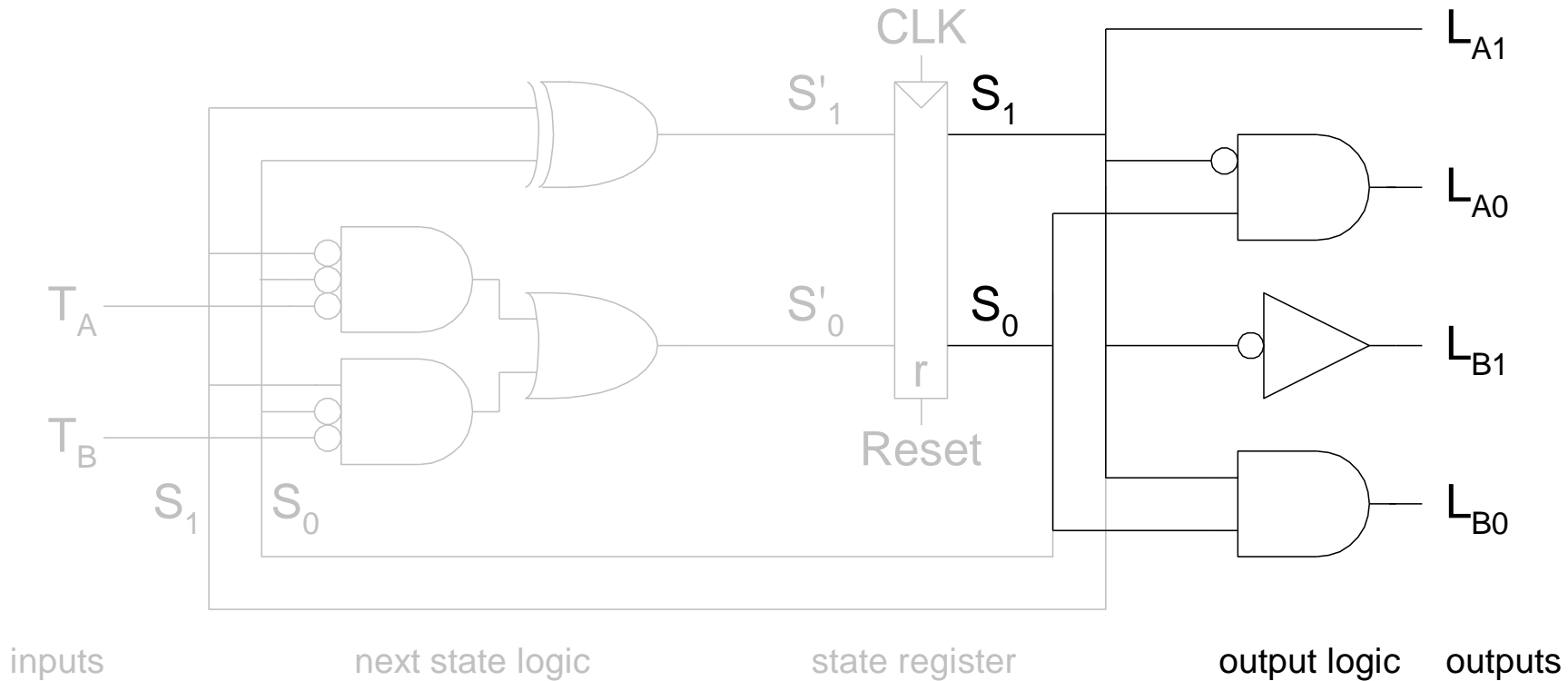
# FSM Schematic: State Register



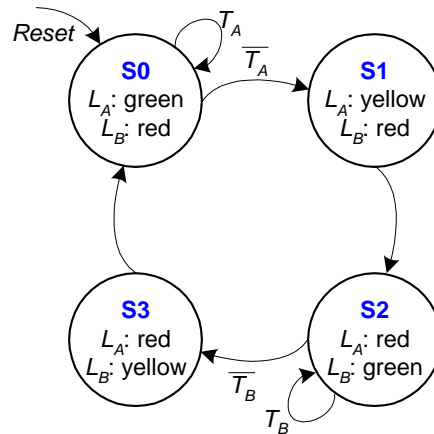
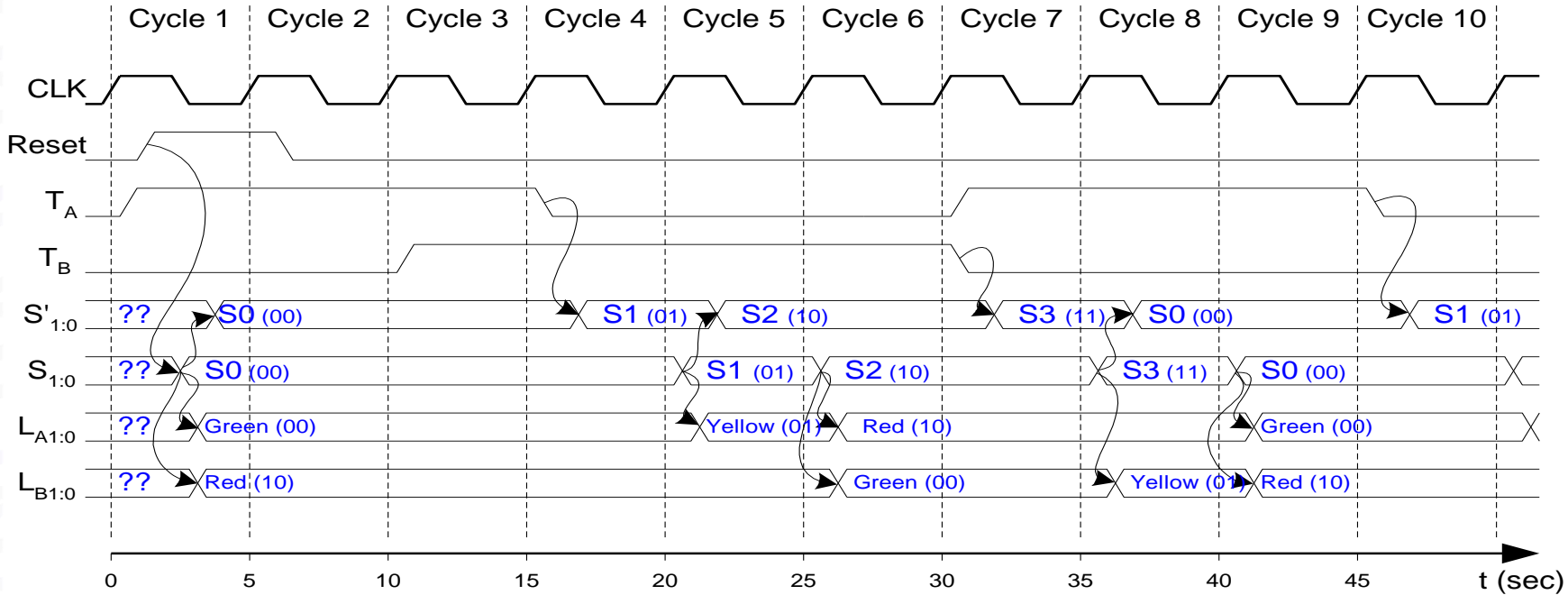
# FSM Schematic: Next State Logic



# FSM Schematic: Output Logic



# FSM Timing Diagram



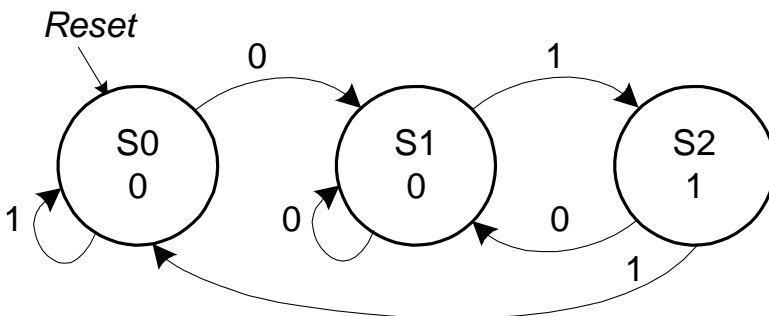
# FSM State Encoding

- **Binary** encoding:
  - i.e., for four states, 00, 01, 10, 11
- **One-hot** encoding
  - One state bit per state
  - Only one state bit HIGH at once
  - i.e., for 4 states, 0001, 0010, 0100, 1000
  - Requires more flip-flops
  - Often next state and output logic is simpler

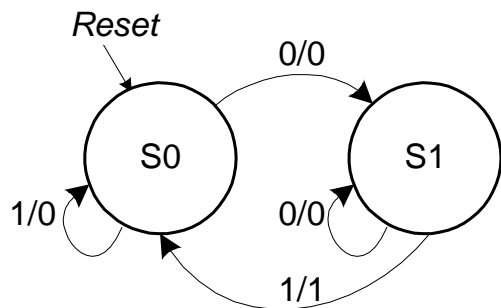


# State Transition Diagrams

## Moore FSM



## Mealy FSM



Mealy FSM: arcs indicate input/output



# Moore FSM State Transition Table

Current State		Inputs	Next State	
$S_1$	$S_0$		$S'_1$	$S'_0$
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		

State	Encoding
S0	00
S1	01
S2	10



# Moore FSM State Transition Table

Current State		Inputs	Next State	
$S_1$	$S_0$		$S'_1$	$S'_0$
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

State	Encoding
S0	00
S1	01
S2	10

$$S'_1 = S_0 A$$

$$S'_0 = \overline{A}$$

# Moore FSM Output Table

Current State		Output
$S_1$	$S_0$	$Y$
0	0	
0	1	
1	0	



# Moore FSM Output Table

Current State		Output
$S_1$	$S_0$	$Y$
0	0	0
0	1	0
1	0	1

$$Y = S_1$$



# Mealy FSM State Transition & Output Table

Current State	Input	Next State	Output
$S_0$	$A$	$S'_0$	$Y$
0	0		
0	1		
1	0		
1	1		

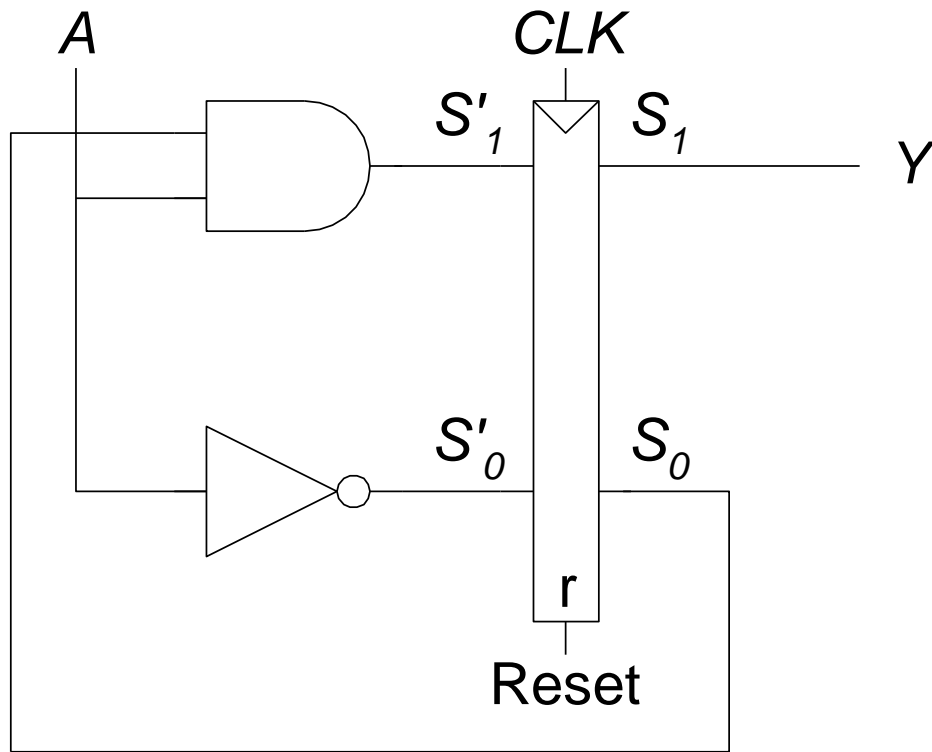
State	Encoding
S0	00
S1	01

# Mealy FSM State Transition & Output Table

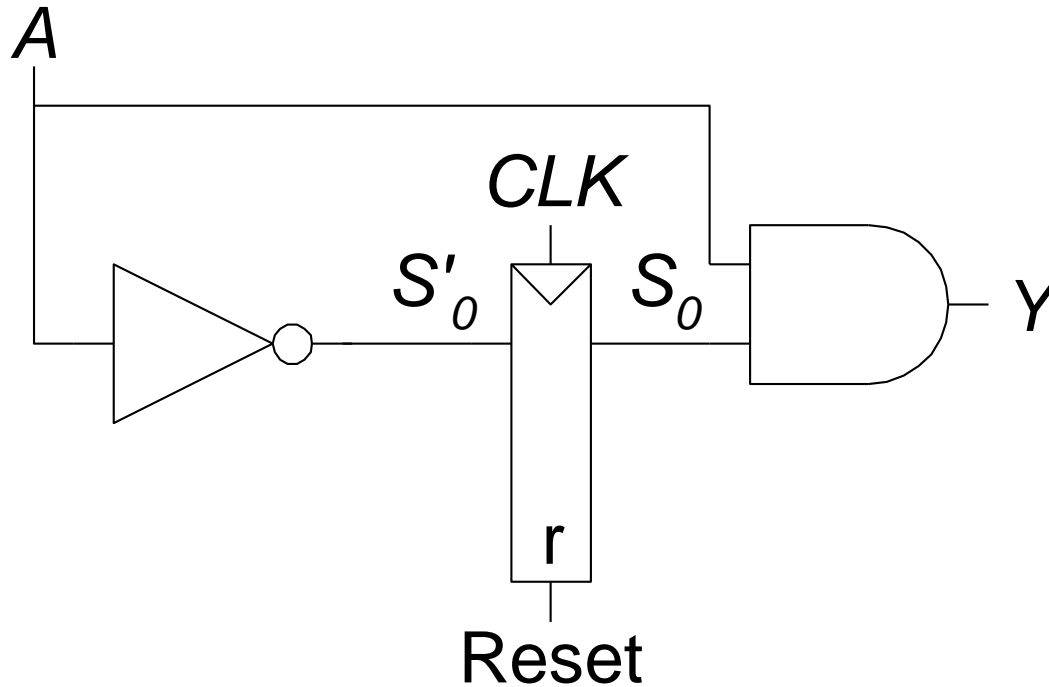
Current State	Input	Next State	Output
$S_0$	$A$	$S'_0$	$Y$
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

State	Encoding
S0	00
S1	01

# Moore FSM Schematic

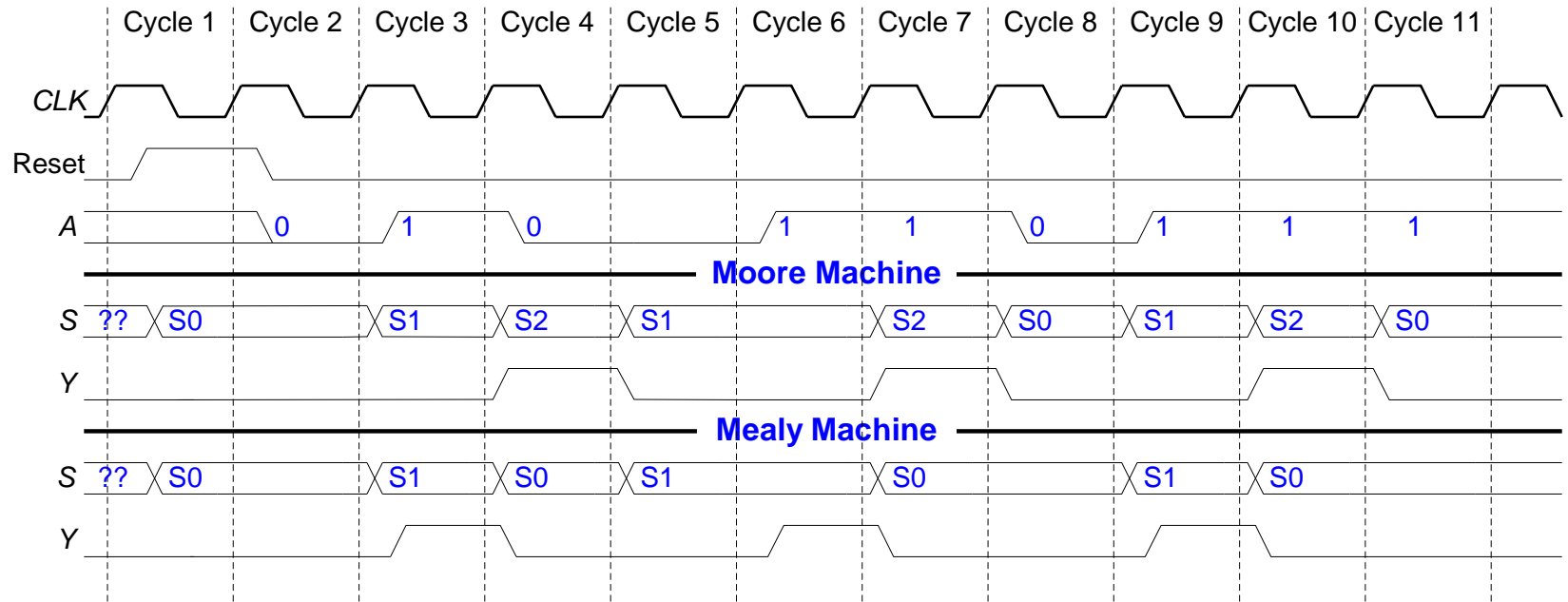


# Mealy FSM Schematic





# Moore & Mealy Timing Diagram



# Factoring State Machines

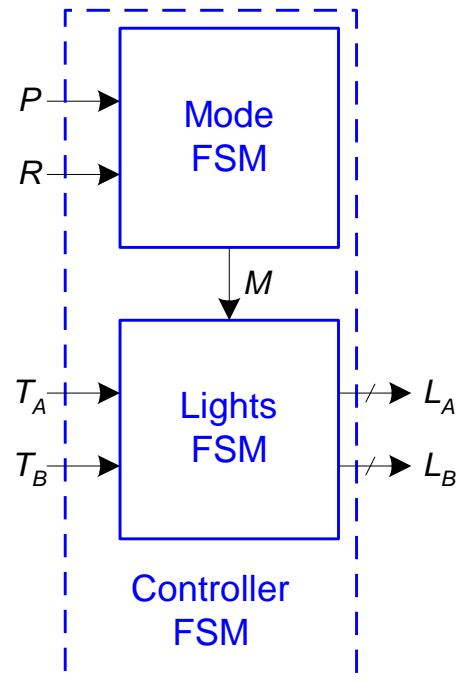
- Break complex FSMs into smaller interacting FSMs
- Example: Modify traffic light controller to have Parade Mode.
  - Two more inputs:  $P$ ,  $R$
  - When  $P = 1$ , enter Parade Mode & Bravado Blvd light stays green
  - When  $R = 1$ , leave Parade Mode

# Parade FSM

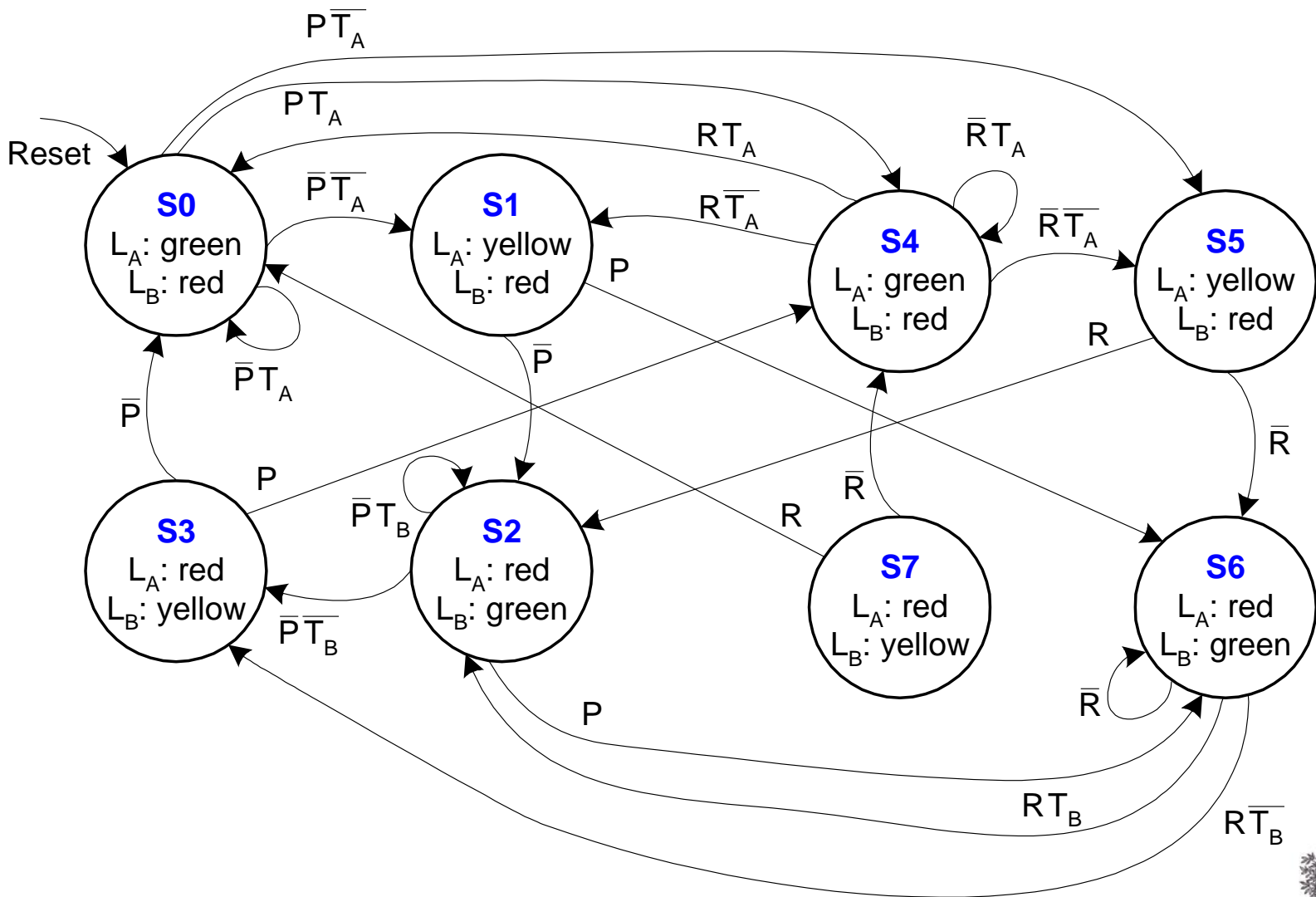
## Unfactored FSM



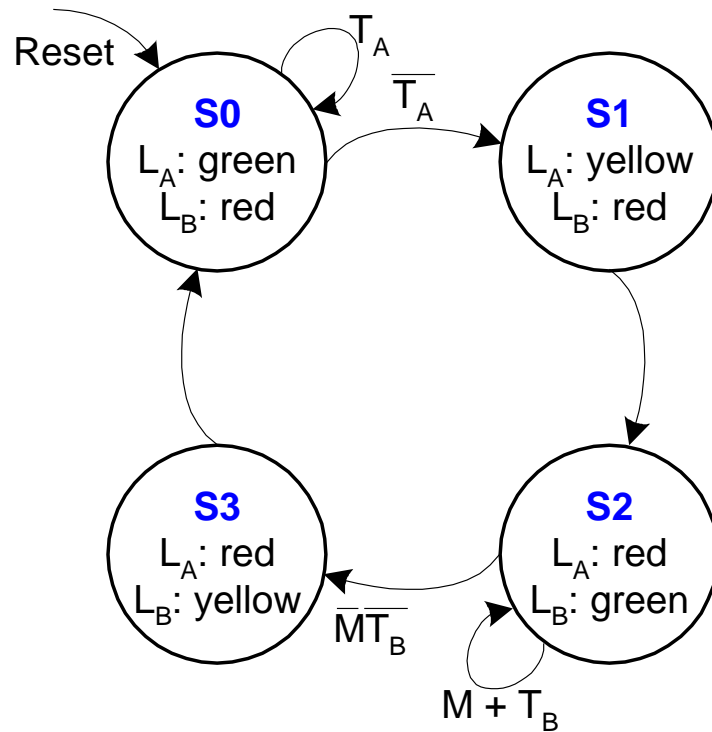
## Factored FSM



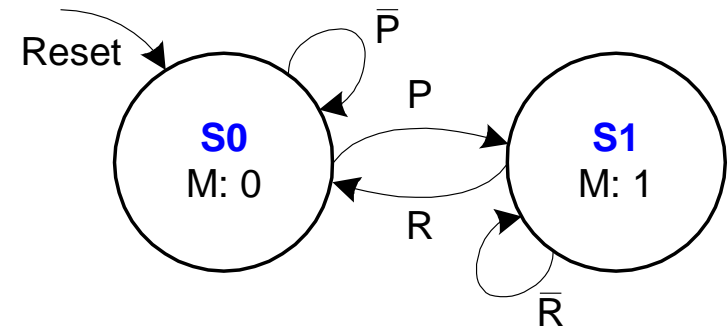
# Unfactored FSM



# Factored FSM



Lights FSM



Mode FSM

# FSM Design Procedure

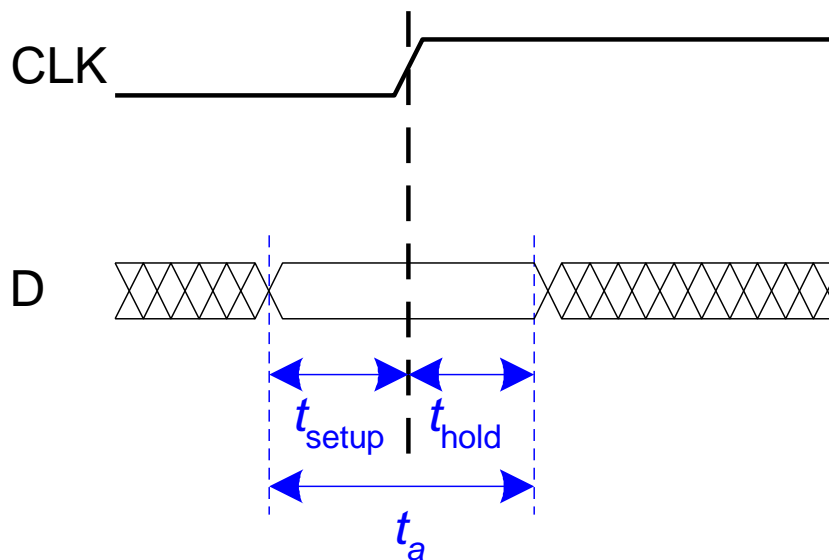
1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. For Moore machine:
  1. Rewrite state transition table with state encodings
  2. Write output table
6. For a Mealy machine:
  1. Rewrite combined state transition and output table with state encodings
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic

# Timing

- Flip-flop samples  $D$  at clock edge
- $D$  must be stable when sampled
- Similar to a photograph,  $D$  must be stable around clock edge
- If not, metastability can occur

# Input Timing Constraints

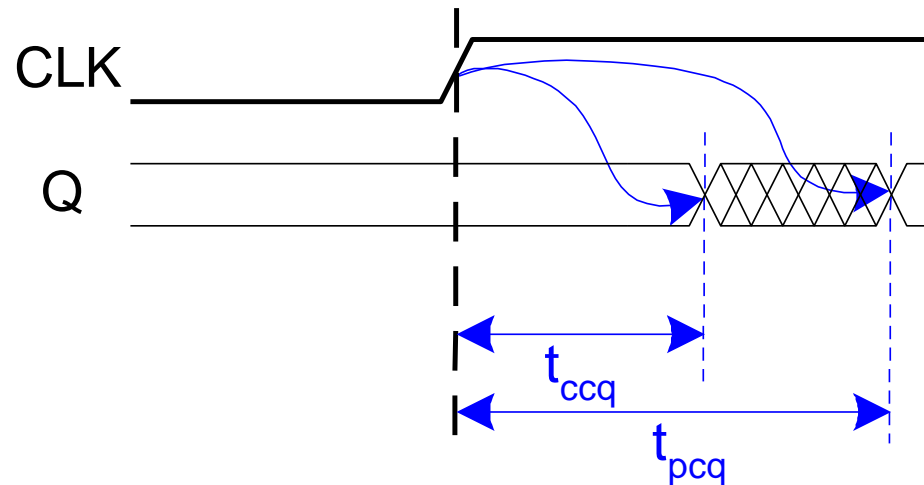
- **Setup time:**  $t_{\text{setup}}$  = time *before* clock edge data must be stable (i.e. not changing)
- **Hold time:**  $t_{\text{hold}}$  = time *after* clock edge data must be stable
- **Aperture time:**  $t_a$  = time *around* clock edge data must be stable ( $t_a = t_{\text{setup}} + t_{\text{hold}}$ )





# Output Timing Constraints

- **Propagation delay:**  $t_{pcq}$  = time after clock edge that the output  $Q$  is guaranteed to be stable (i.e., to stop changing)
- **Contamination delay:**  $t_{ccq}$  = time after clock edge that  $Q$  might be unstable (i.e., start changing)

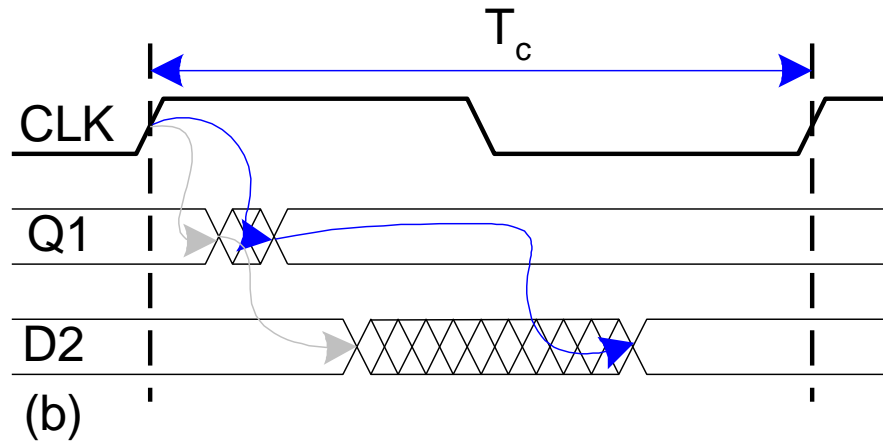
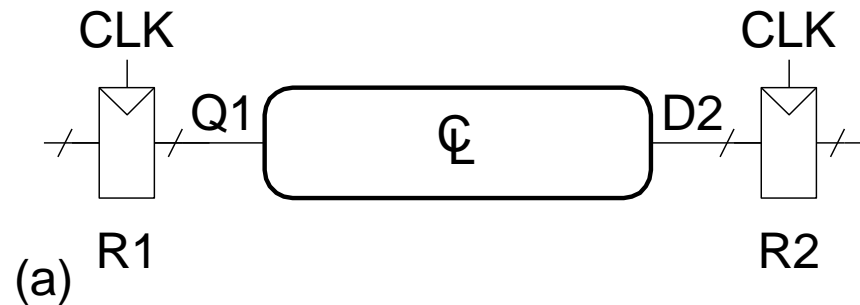


# Dynamic Discipline

- Synchronous sequential circuit inputs must be stable during aperture (setup and hold) time around clock edge
- Specifically, inputs must be stable
  - at least  $t_{\text{setup}}$  before the clock edge
  - at least until  $t_{\text{hold}}$  after the clock edge

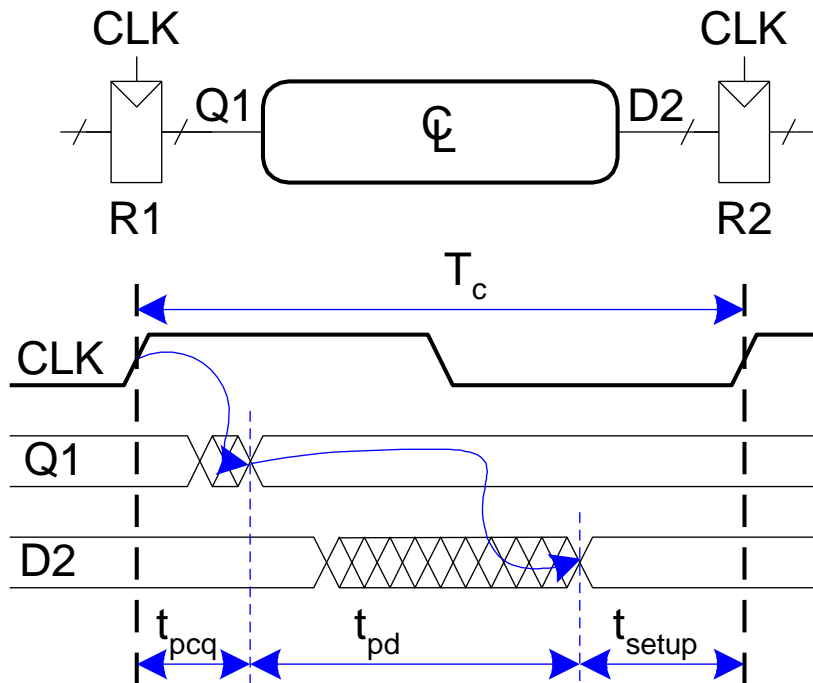
# Dynamic Discipline

- The delay between registers has a **minimum** and **maximum** delay, dependent on the delays of the circuit elements



# Setup Time Constraint

- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least  $t_{\text{setup}}$  before clock edge

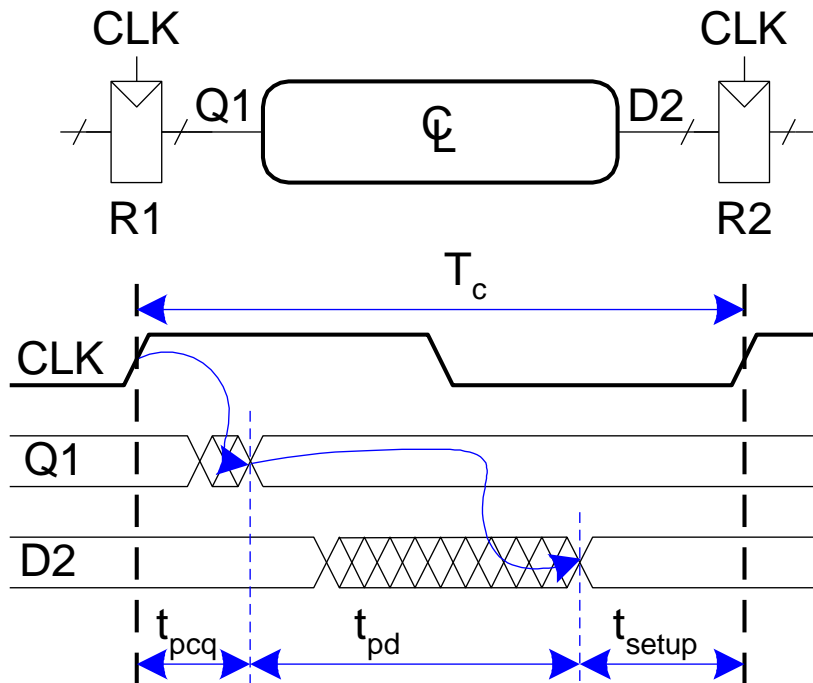


$$T_c \geq t_{pcq} + t_{pd} + t_{\text{setup}}$$



# Setup Time Constraint

- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least  $t_{\text{setup}}$  before clock edge



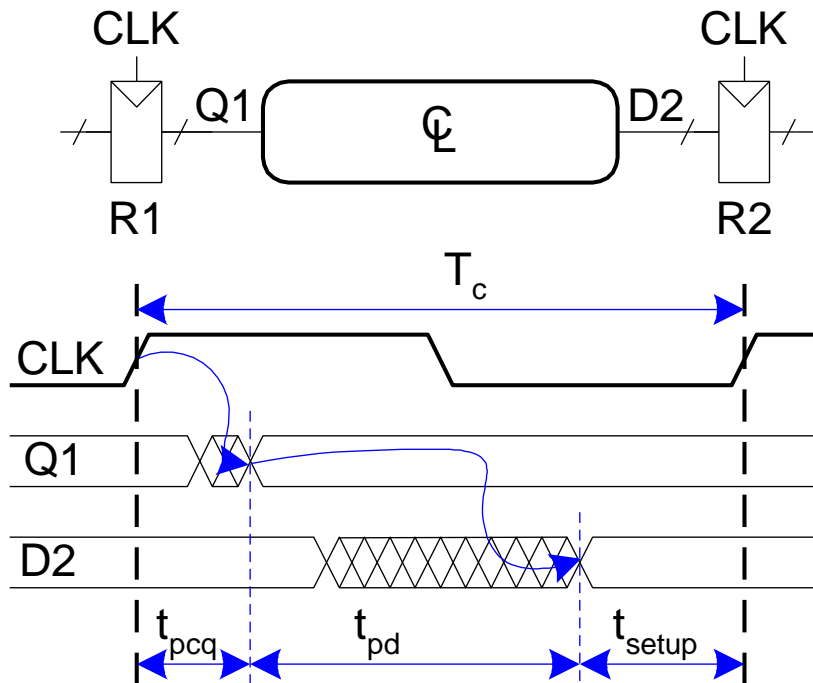
$$T_c \geq t_{pcq} + t_{pd} + t_{\text{setup}}$$

$$t_{pd} \leq$$



# Setup Time Constraint

- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least  $t_{\text{setup}}$  before clock edge



$$T_c \geq t_{pcq} + t_{pd} + t_{\text{setup}}$$

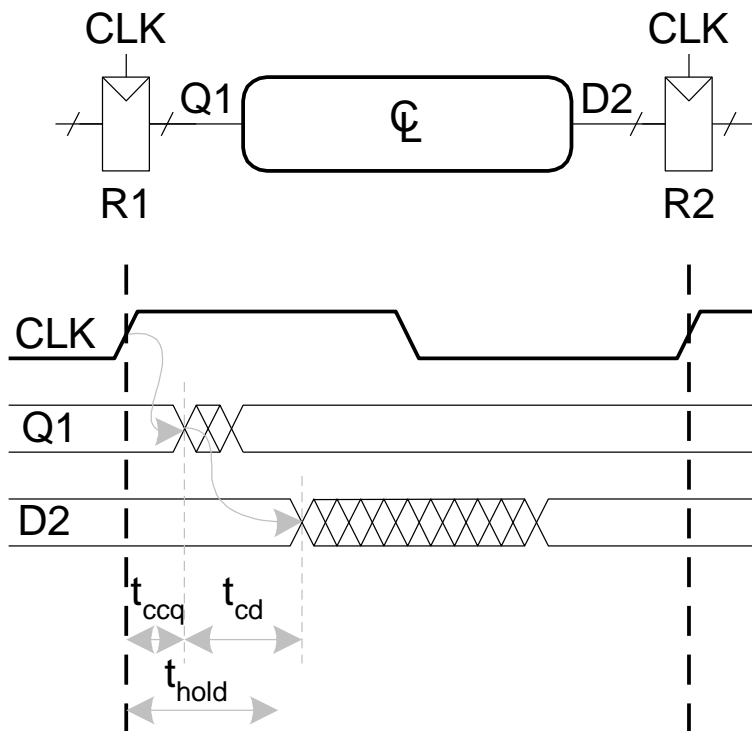
$$t_{pd} \leq T_c - (t_{pcq} + t_{\text{setup}})$$

$(t_{pcq} + t_{\text{setup}})$ : sequencing overhead



# Hold Time Constraint

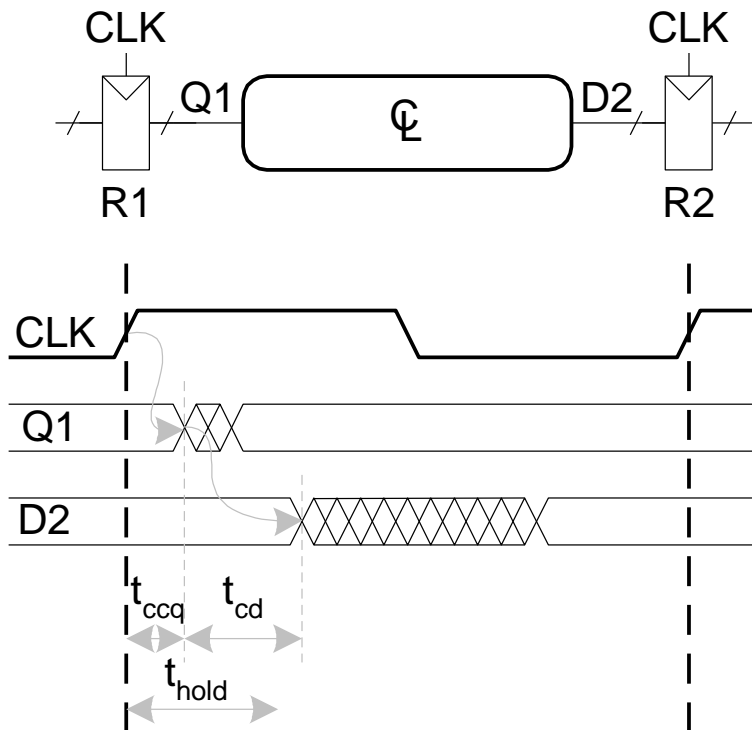
- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least  $t_{hold}$  after the clock edge



$$t_{hold} <$$

# Hold Time Constraint

- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least  $t_{hold}$  after the clock edge



$$t_{hold} < t_{ccq} + t_{cd}$$

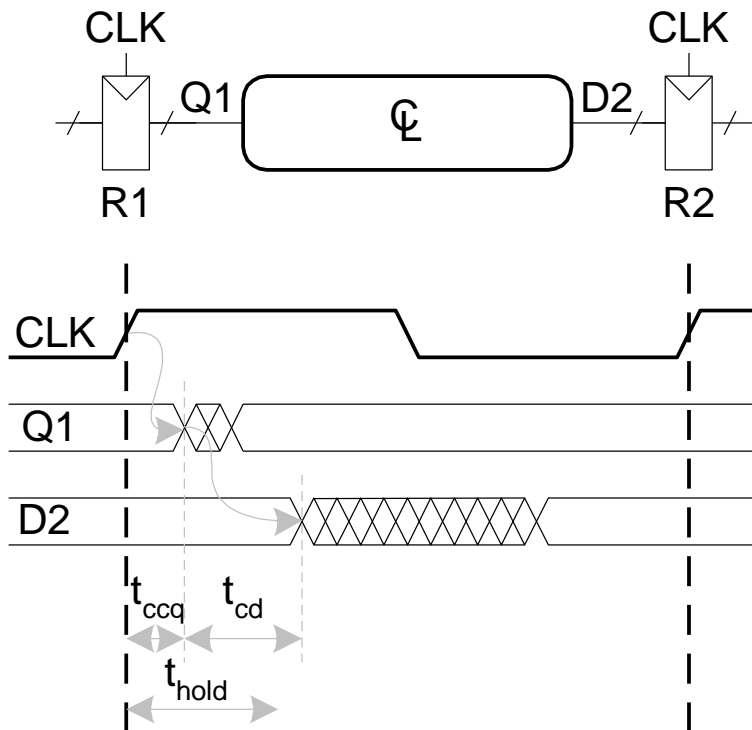
$$t_{cd} >$$





# Hold Time Constraint

- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least  $t_{hold}$  after the clock edge

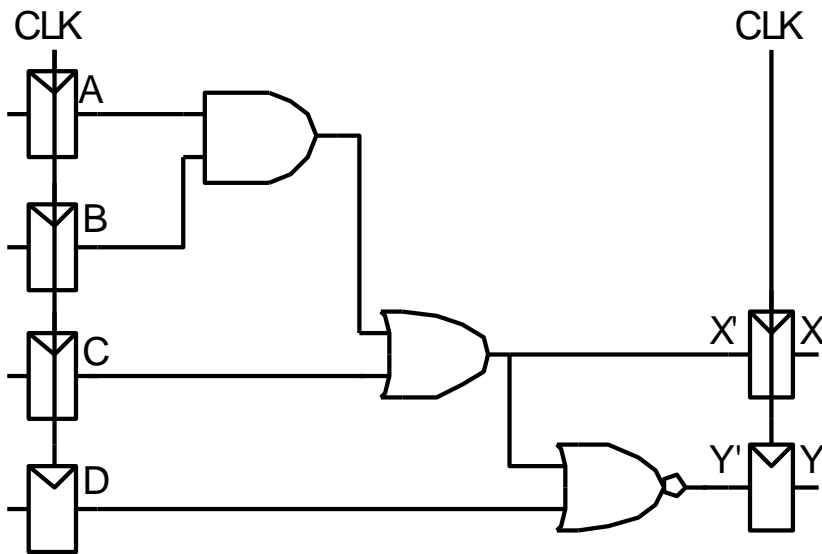


$$t_{hold} < t_{ccq} + t_{cd}$$

$$t_{cd} > t_{hold} - t_{ccq}$$



# Timing Analysis



## Timing Characteristics

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$t_{\text{hold}} = 70 \text{ ps}$$

per gate

$$\left[ \begin{array}{l} t_{pd} = 35 \text{ ps} \\ t_{cd} = 25 \text{ ps} \end{array} \right.$$

$$t_{pd} =$$

$$t_{cd} =$$

Setup time constraint:

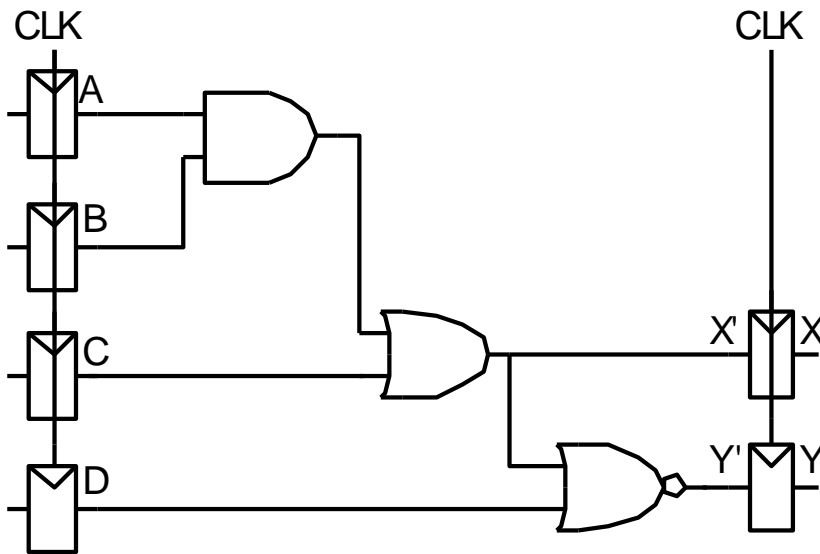
$$T_c \geq$$

$$f_c =$$

Hold time constraint:

$$t_{ccq} + t_{cd} > t_{\text{hold}} ?$$

# Timing Analysis



## Timing Characteristics

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$t_{\text{hold}} = 70 \text{ ps}$$

$$\text{per gate} \left[ \begin{array}{l} t_{pd} = 35 \text{ ps} \\ t_{cd} = 25 \text{ ps} \end{array} \right.$$

$$t_{pd} = 3 \times 35 \text{ ps} = 105 \text{ ps}$$

$$t_{cd} = 25 \text{ ps}$$

Setup time constraint:

$$T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps}$$

$$f_c = 1/T_c = 4.65 \text{ GHz}$$

Hold time constraint:

$$t_{ccq} + t_{cd} > t_{\text{hold}} ?$$

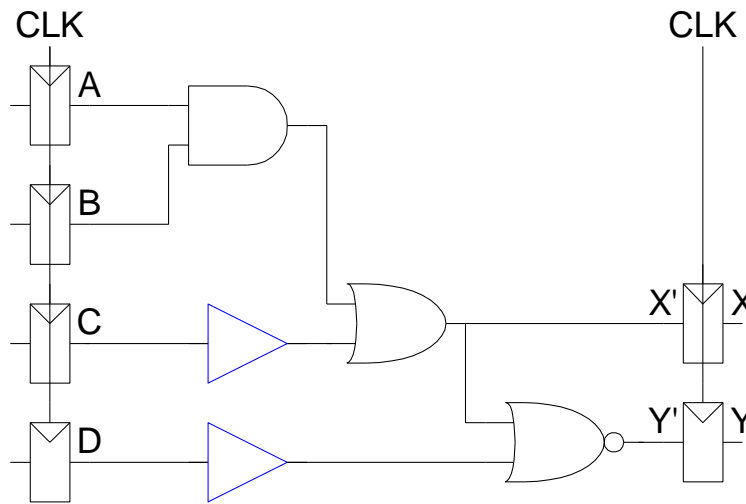
$$(30 + 25) \text{ ps} > 70 \text{ ps} ? \text{ No!}$$





# Timing Analysis

Add buffers to the short paths:



$$t_{pd} = 3 \times 35 \text{ ps} = 105 \text{ ps}$$

$$t_{cd} = 2 \times 25 \text{ ps} = 50 \text{ ps}$$

Setup time constraint:

$$T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps}$$

$$f_c = 1/T_c = 4.65 \text{ GHz}$$

## Timing Characteristics

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$t_{\text{hold}} = 70 \text{ ps}$$

$$\text{per gate} \begin{cases} t_{pd} = 35 \text{ ps} \\ t_{cd} = 25 \text{ ps} \end{cases}$$

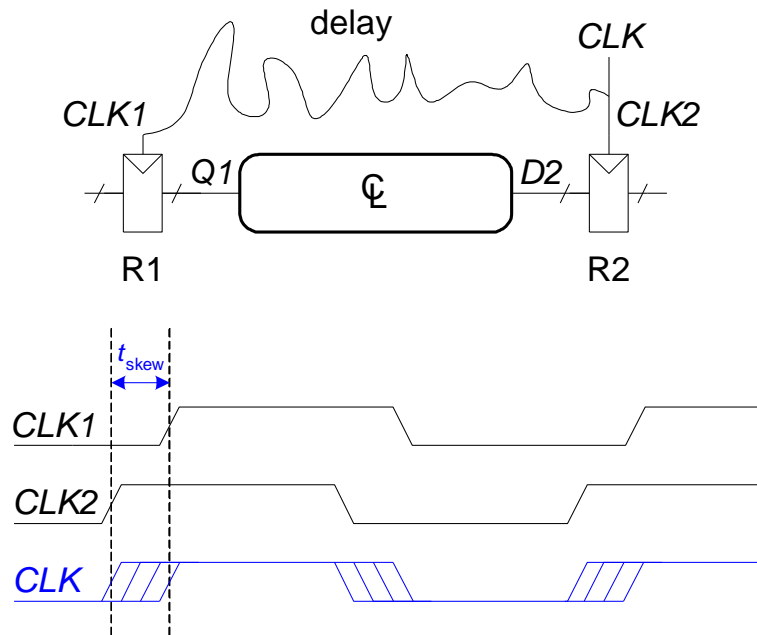
Hold time constraint:

$$t_{ccq} + t_{cd} > t_{\text{hold}} ?$$

$$(30 + 50) \text{ ps} > 70 \text{ ps} ? \text{ Yes!}$$

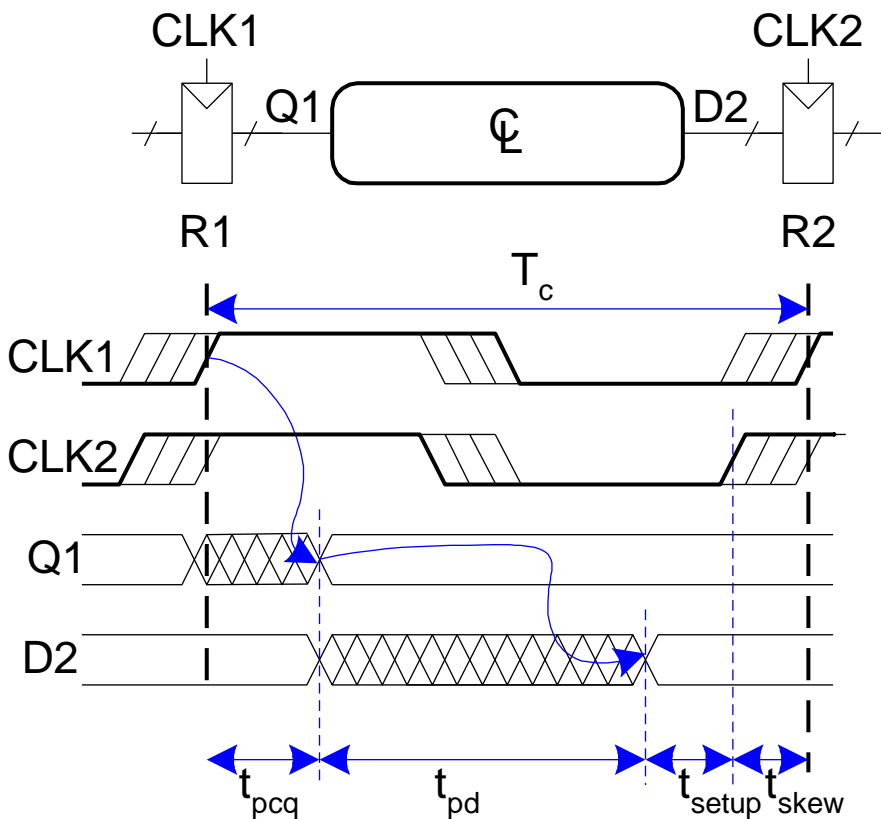
# Clock Skew

- The clock doesn't arrive at all registers at same time
- **Skew:** difference between two clock edges
- Perform **worst case analysis** to guarantee dynamic discipline is not violated for any register – many registers in a system!



# Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1

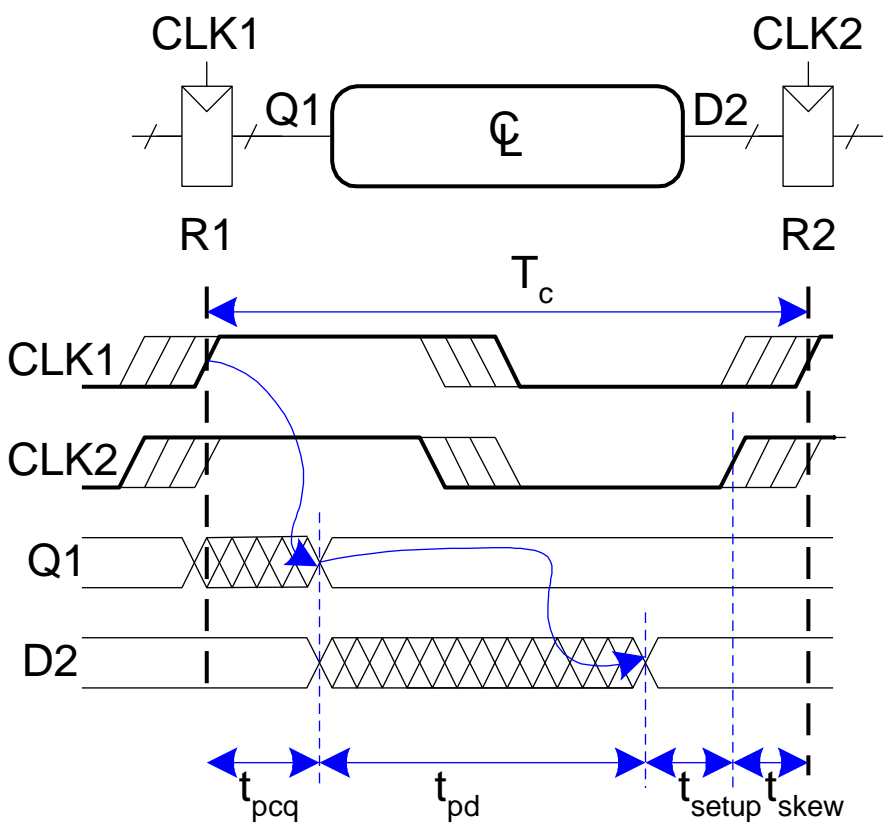


$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$



# Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1



$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

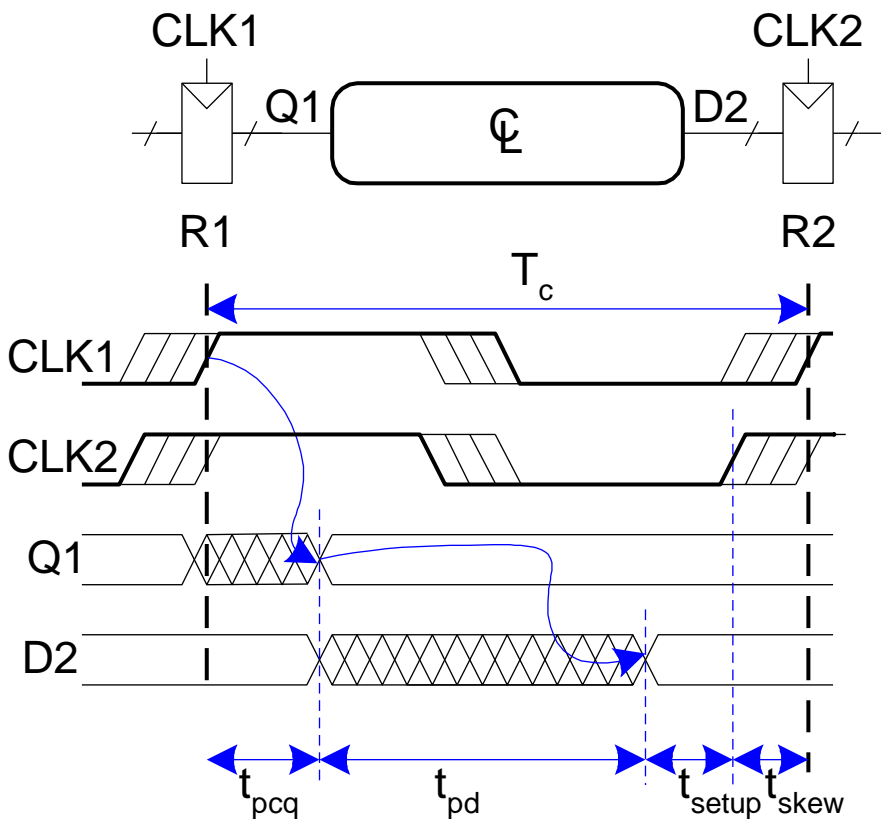
$$t_{pd} \leq$$





# Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1



$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

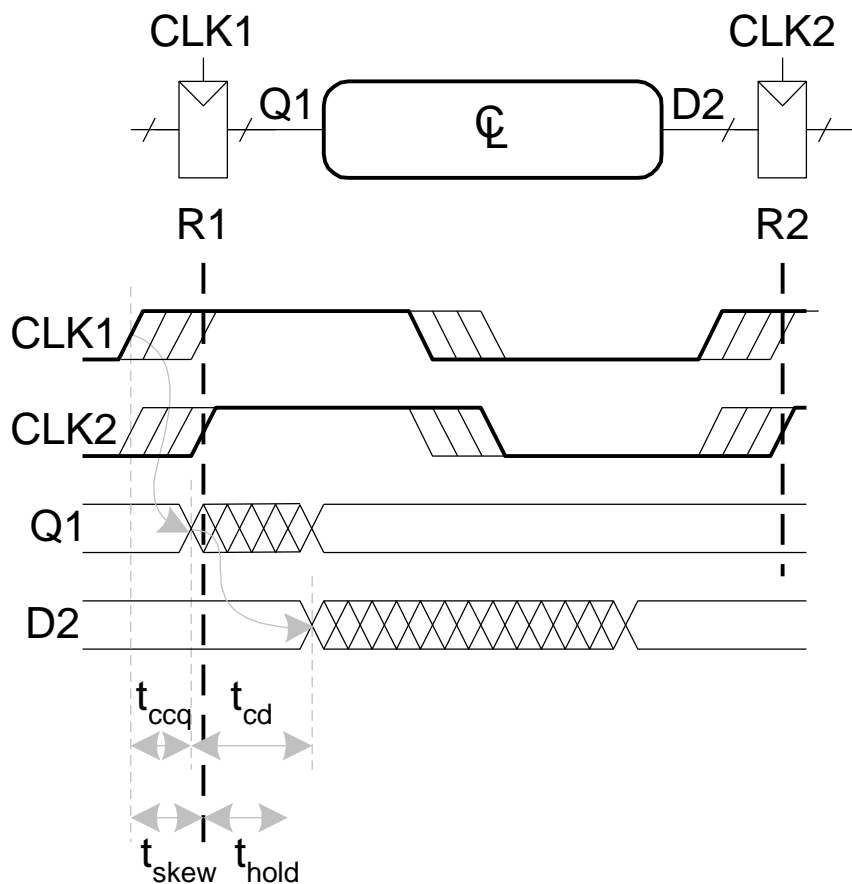
$$t_{pd} \leq T_c - (t_{pcq} + t_{setup} + t_{skew})$$





# Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1

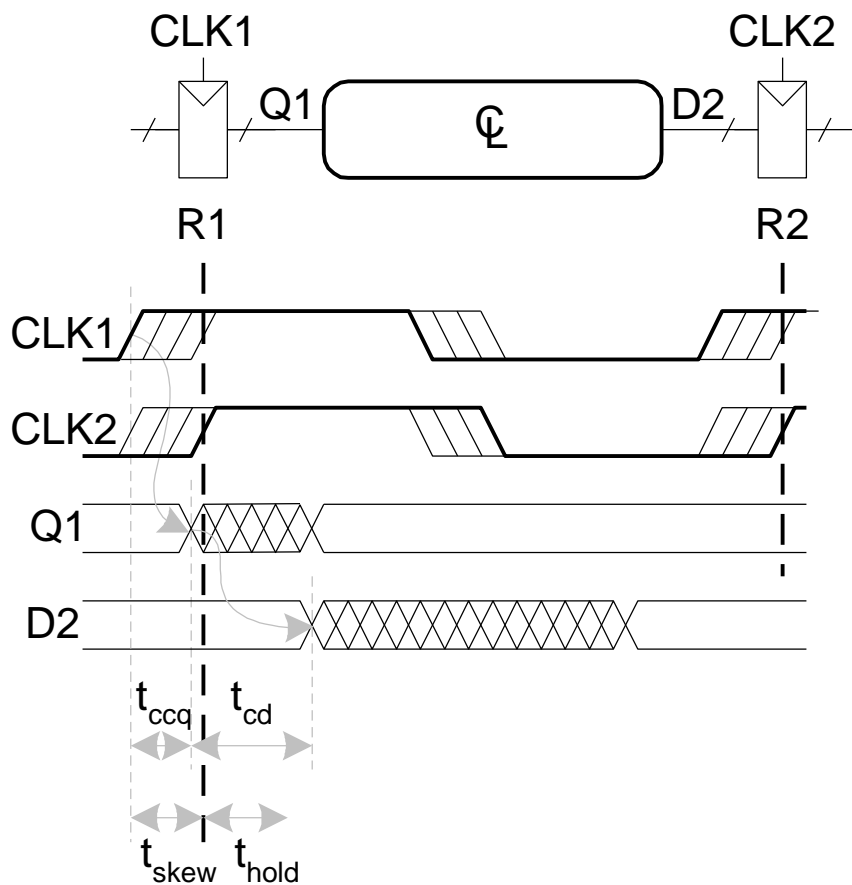


$$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$$

$$t_{cd} >$$

# Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1

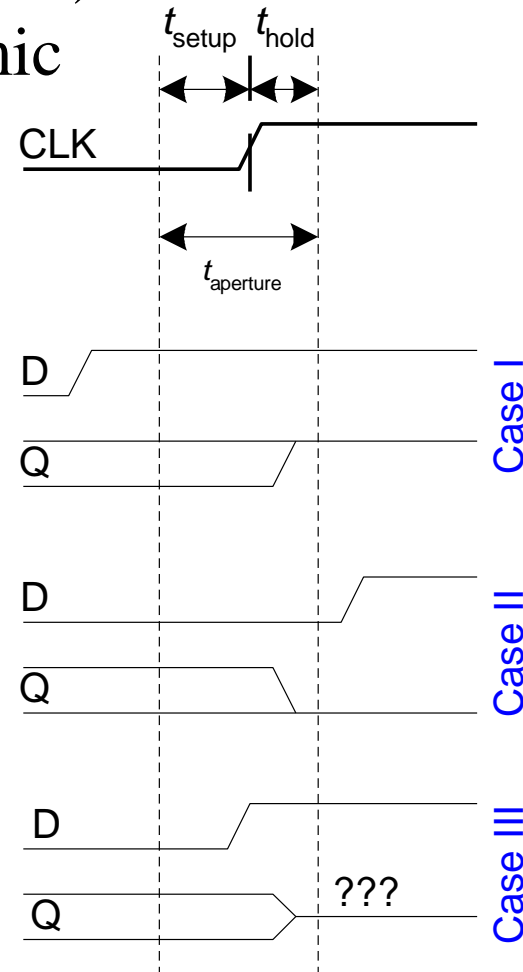
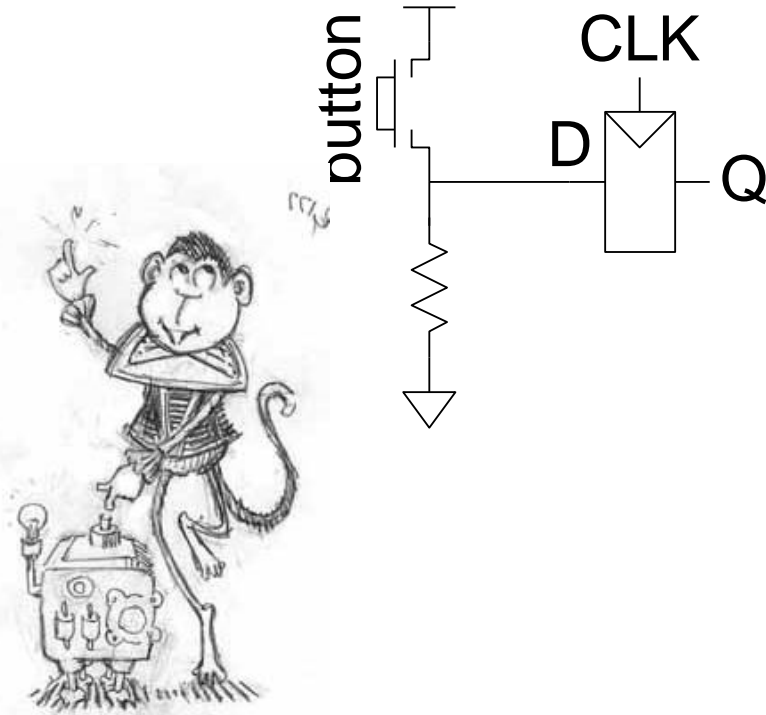


$$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$$

$$t_{cd} > t_{hold} + t_{skew} - t_{ccq}$$

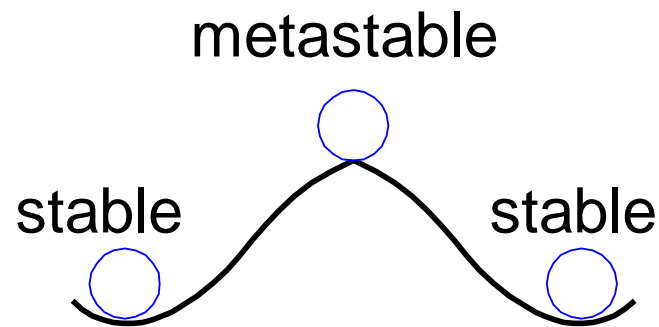
# Violating the Dynamic Discipline

- Asynchronous (for example, user) inputs might violate the dynamic discipline



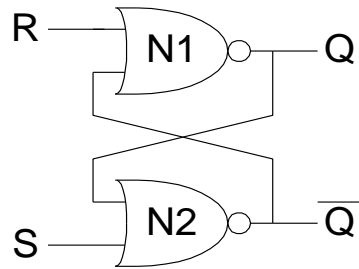
# Metastability

- **Bistable devices:** two stable states, and a metastable state between them
- **Flip-flop:** two stable states (1 and 0) and one metastable state
- If flip-flop lands in metastable state, could stay there for an undetermined amount of time



# Flip-Flop Internals

- Flip-flop has **feedback**: if  $Q$  is somewhere between 1 and 0, cross-coupled gates drive output to either rail (1 or 0)



- Metastable signal**: if it hasn't resolved to 1 or 0
- If flip-flop input changes at random time, **probability** that output  $Q$  is metastable after waiting some time,  $t$ :

$$P(t_{\text{res}} > t) = (T_0/T_c) e^{-t/\tau}$$

$t_{\text{res}}$  : time to resolve to 1 or 0

$T_0, \tau$  : properties of the circuit

# Metastability

- **Intuitively:**

$T_0/T_c$ : probability input changes at a bad time (during aperture)

$$P(t_{\text{res}} > t) = (T_0/T_c) e^{-t/\tau}$$

$\tau$ : time constant for how fast flip-flop moves away from metastability

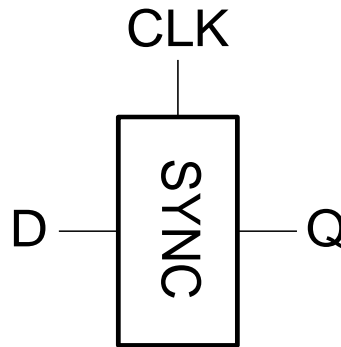
$$P(t_{\text{res}} > t) = (T_0/T_c) e^{-t/\tau}$$

- In short, if flip-flop samples metastable input, if you wait long enough ( $t$ ), the output will have resolved to 1 or 0 with high probability.



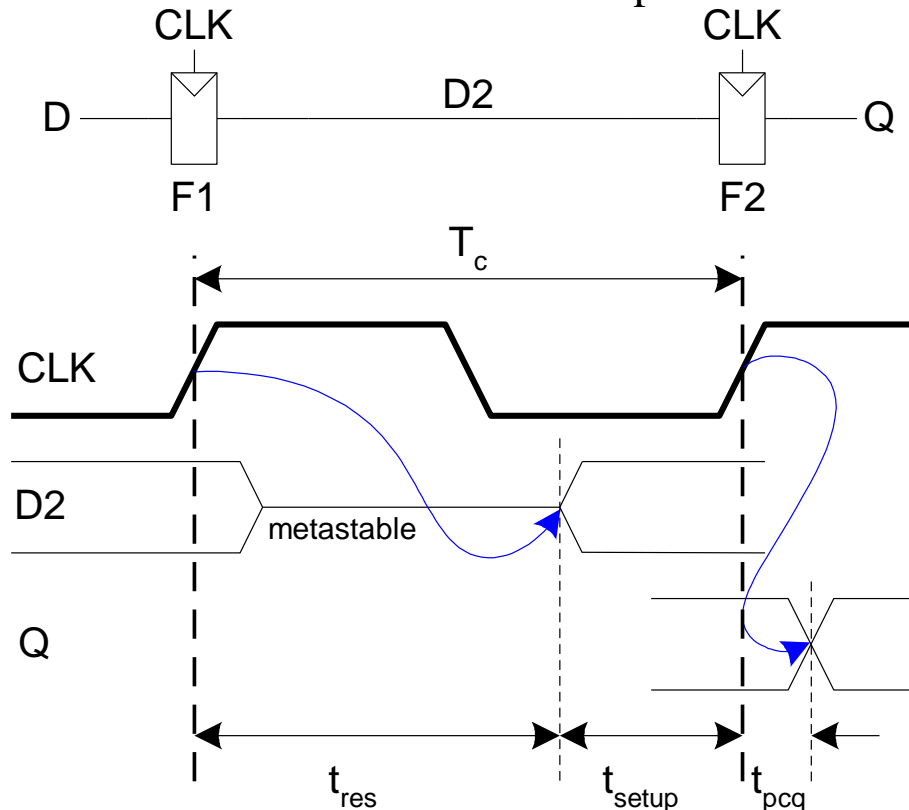
# Synchronizers

- **Asynchronous inputs are inevitable** (user interfaces, systems with different clocks interacting, etc.)
- **Synchronizer goal:** make the probability of failure (the output  $Q$  still being metastable) low
- Synchronizer cannot make the probability of failure 0



# Synchronizer Internals

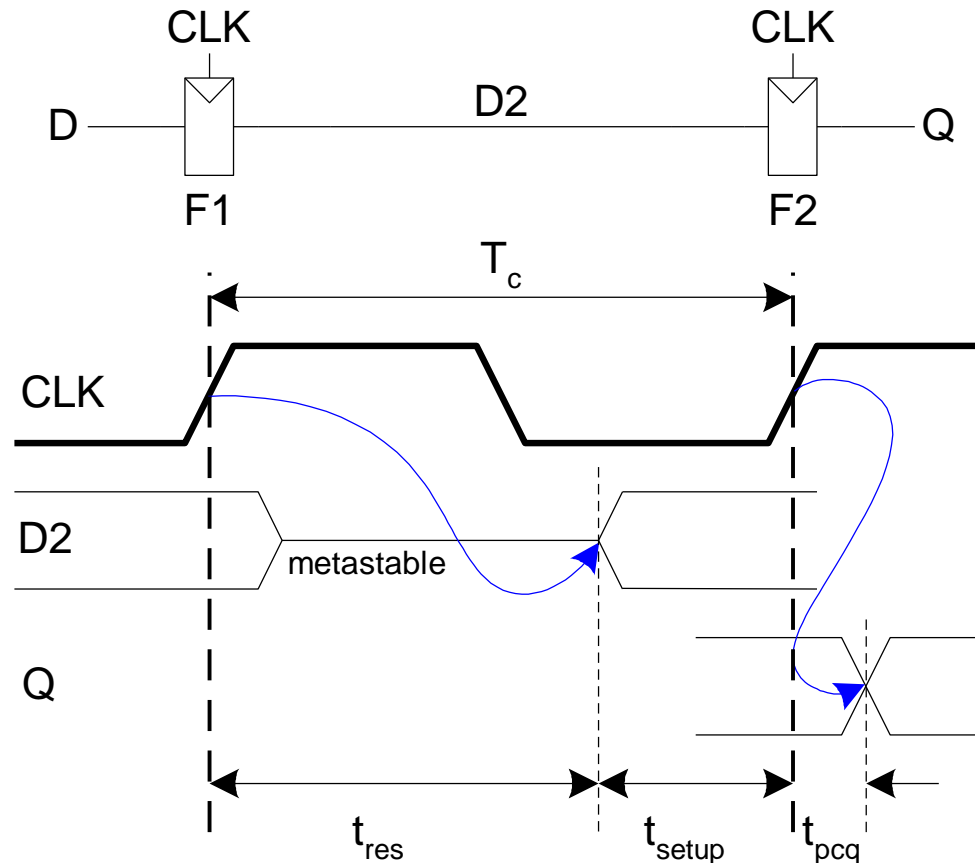
- Synchronizer: built with two back-to-back flip-flops
- Suppose D is transitioning when sampled by F1
- Internal signal D2 has  $(T_c - t_{setup})$  time to resolve to 1 or 0



# Synchronizer Probability of Failure

For each sample, probability of failure is:

$$P(\text{failure}) = (T_0/T_c) e^{-(T_c - t_{setup})/\tau}$$



# Synchronizer Mean Time Between Failures

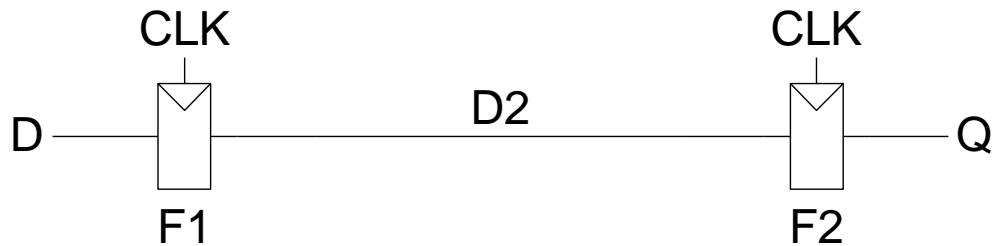
- If asynchronous input changes once per second, probability of failure per second is  $P(\text{failure})$ .
- If input changes  $N$  times per second, probability of failure per second is:

$$P(\text{failure})/\text{second} = (NT_0/T_c) e^{-(T_c - t_{\text{setup}})/\tau}$$

- Synchronizer fails, on average,  $1/[P(\text{failure})/\text{second}]$
- Called *mean time between failures*, MTBF:

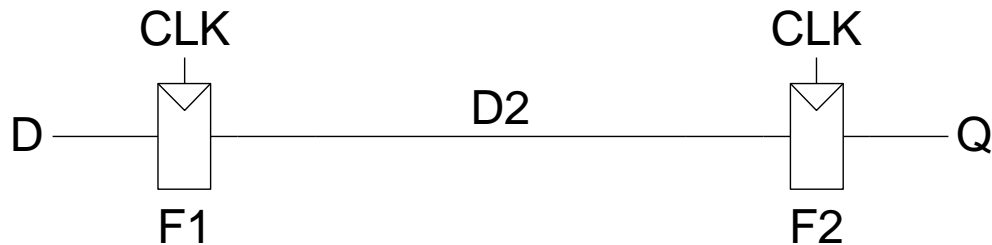
$$\text{MTBF} = 1/[P(\text{failure})/\text{second}] = (T_c/NT_0) e^{(T_c - t_{\text{setup}})/\tau}$$

# Example Synchronizer



- Suppose:  $T_c = 1/500 \text{ MHz} = 2 \text{ ns}$     $\tau = 200 \text{ ps}$   
 $T_0 = 150 \text{ ps}$     $t_{\text{setup}} = 100 \text{ ps}$   
 $N = 10 \text{ events per second}$
- What is the probability of failure? MTBF?

# Example Synchronizer



- Suppose:  $T_c = 1/500 \text{ MHz} = 2 \text{ ns}$      $\tau = 200 \text{ ps}$   
 $T_0 = 150 \text{ ps}$                        $t_{\text{setup}} = 100 \text{ ps}$   
 $N = 10 \text{ events per second}$

- What is the probability of failure? MTBF?

$$P(\text{failure}) = (150 \text{ ps}/2 \text{ ns}) e^{-(1.9 \text{ ns})/200 \text{ ps}}$$
$$= 5.6 \times 10^{-6}$$

$$P(\text{failure})/\text{second} = 10 \times (5.6 \times 10^{-6})$$
$$= 5.6 \times 10^{-5} / \text{second}$$

$$\text{MTBF} = 1/[P(\text{failure})/\text{second}] \approx 5 \text{ hours}$$

# Parallelism

- **Two types of parallelism:**
  - **Spatial parallelism**
    - duplicate hardware performs multiple tasks at once
  - **Temporal parallelism**
    - task is broken into multiple stages
    - also called pipelining
    - for example, an assembly line

# Parallelism Definitions

- **Token:** Group of inputs processed to produce group of outputs
- **Latency:** Time for one token to pass from start to end
- **Throughput:** Number of tokens produced per unit time

**Parallelism increases throughput**



# Parallelism Example

- Ben Bitdiddle bakes cookies to celebrate traffic light controller installation
- 5 minutes to roll cookies
- 15 minutes to bake
- What is the latency and throughput without parallelism?

# Parallelism Example

- Ben Bitdiddle bakes cookies to celebrate traffic light controller installation
- 5 minutes to roll cookies
- 15 minutes to bake
- What is the latency and throughput without parallelism?

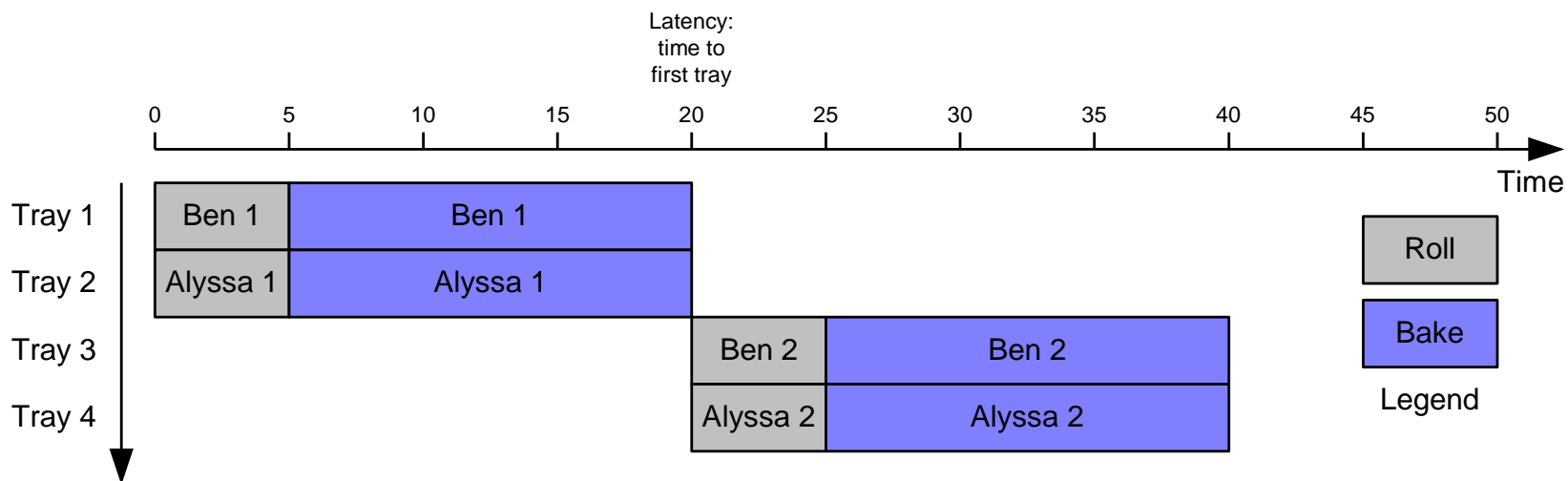
**Latency** = 5 + 15 = 20 minutes = **1/3 hour**

**Throughput** = 1 tray/ 1/3 hour = **3 trays/hour**

# Parallelism Example

- What is the latency and throughput if Ben uses parallelism?
  - **Spatial parallelism:** Ben asks Allysa P. Hacker to help, using her own oven
  - **Temporal parallelism:**
    - two stages: rolling and baking
    - He uses two trays
    - While first batch is baking, he rolls the second batch, etc.

# Spatial Parallelism

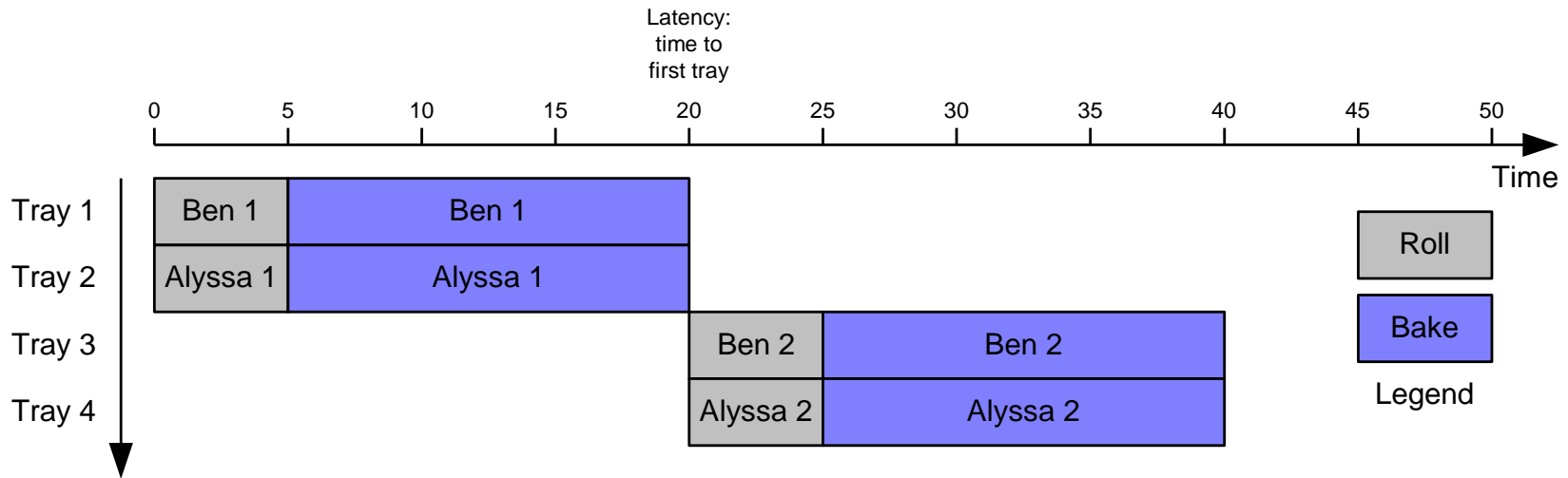


**Latency = ?**

**Throughput = ?**



# Spatial Parallelism

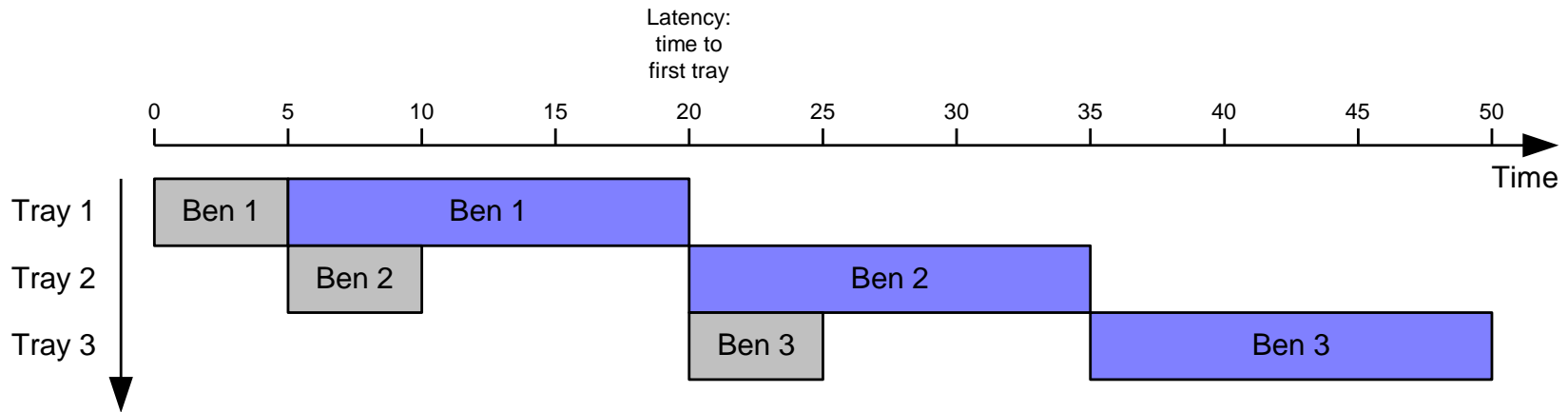


**Latency** = 5 + 15 = 20 minutes = **1/3 hour**

**Throughput** = 2 trays/ 1/3 hour = **6 trays/hour**



# Temporal Parallelism

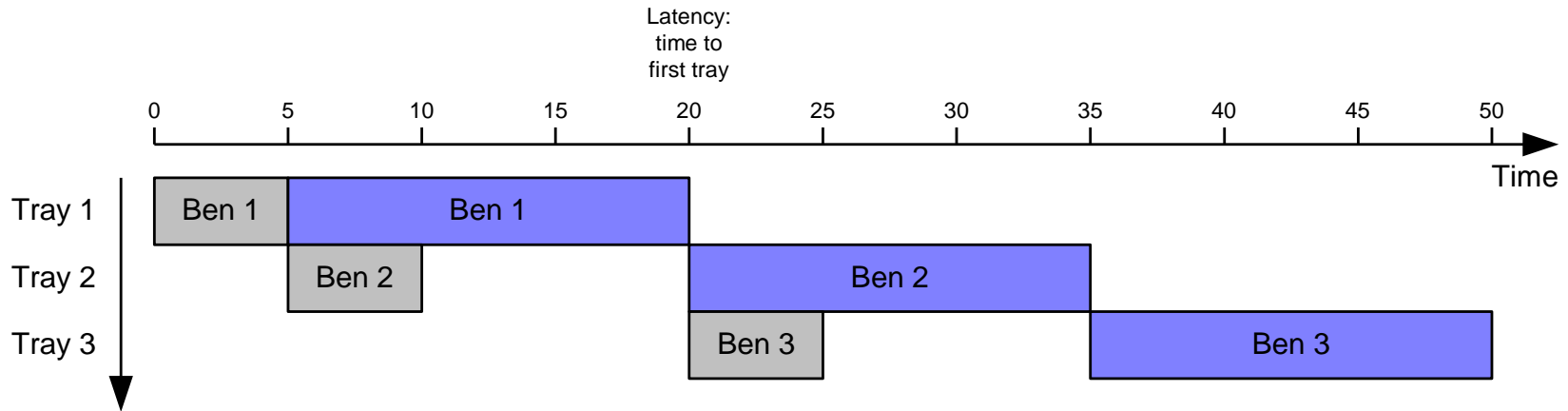


**Latency = ?**

**Throughput = ?**



# Temporal Parallelism



**Latency** =  $5 + 15 = 20$  minutes =  **$1/3$  hour**

**Throughput** = 1 trays/  $1/4$  hour = **4 trays/hour**

Using both techniques, the throughput would be **8 trays/hour**

