Digital Integrated Circuits

(83-313)

Lecture 7:

Designing Sequential Logic Circuits

Prof. Adam Teman

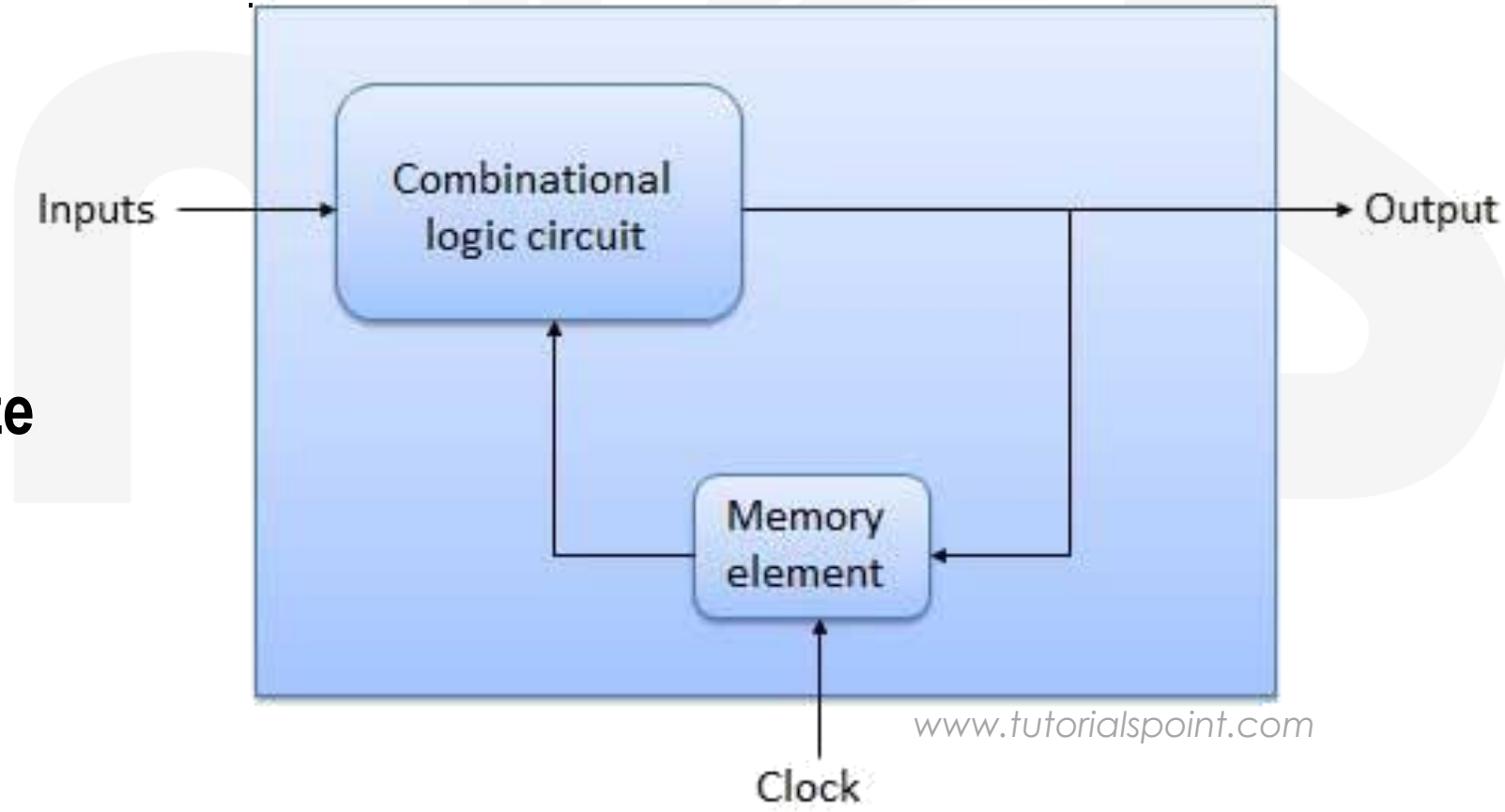
27 April 2021

Emerging Nanoscaled Integrated Circuits and Systems Labs

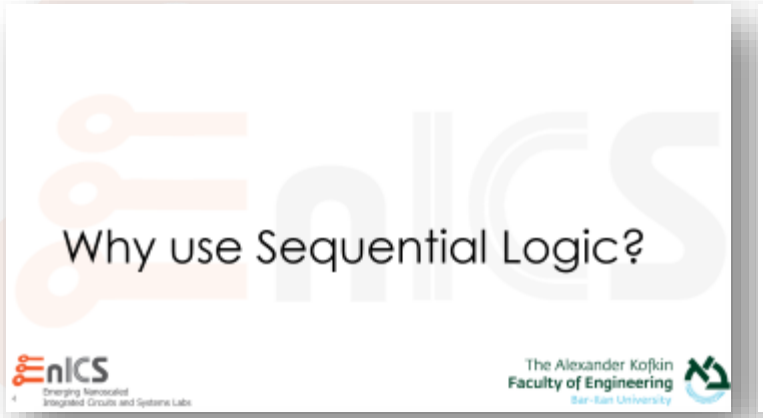
The Alexander Kofkin Faculty of Engineering
Bar-Ilan University

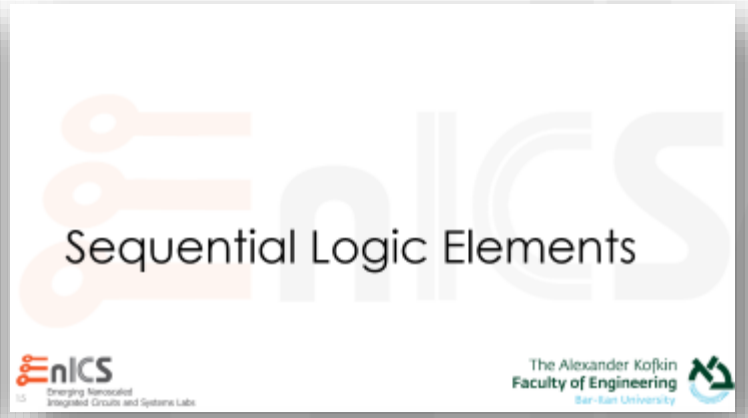
Sequential Logic

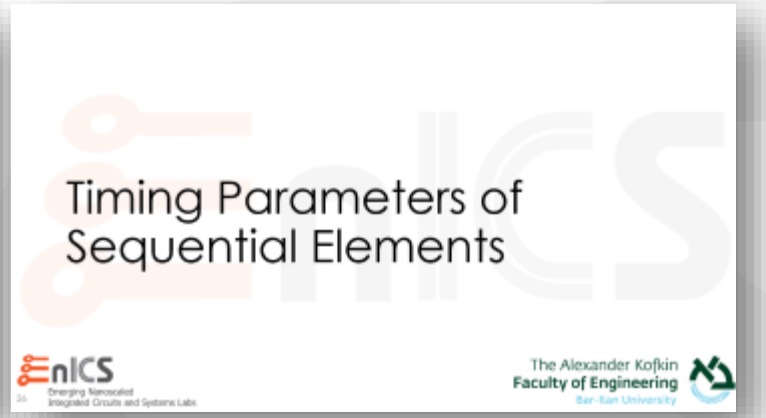
- Sequential circuits are a function of both the *current* state and the *previous* state.
- In other words, they have *memory*.
- The majority of sequential circuits are *Synchronous*, using a clock to synchronize the logic paths.

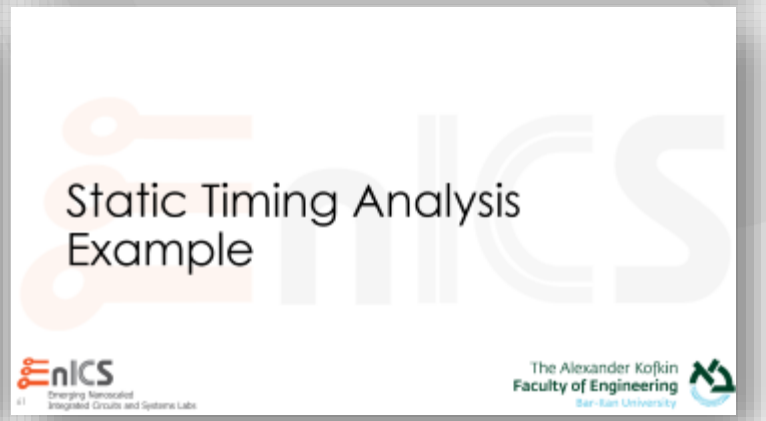


Lecture Content









Why use Sequential Logic?

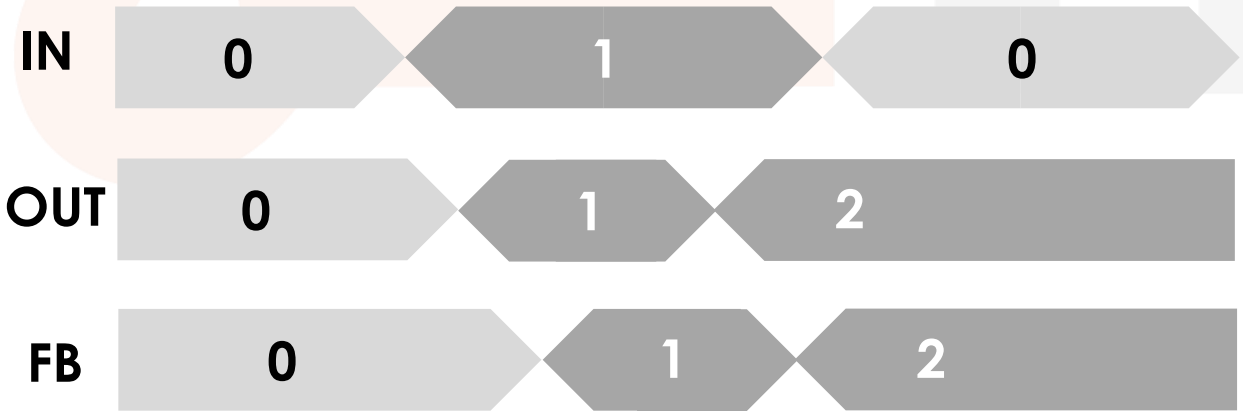
Explanation through example

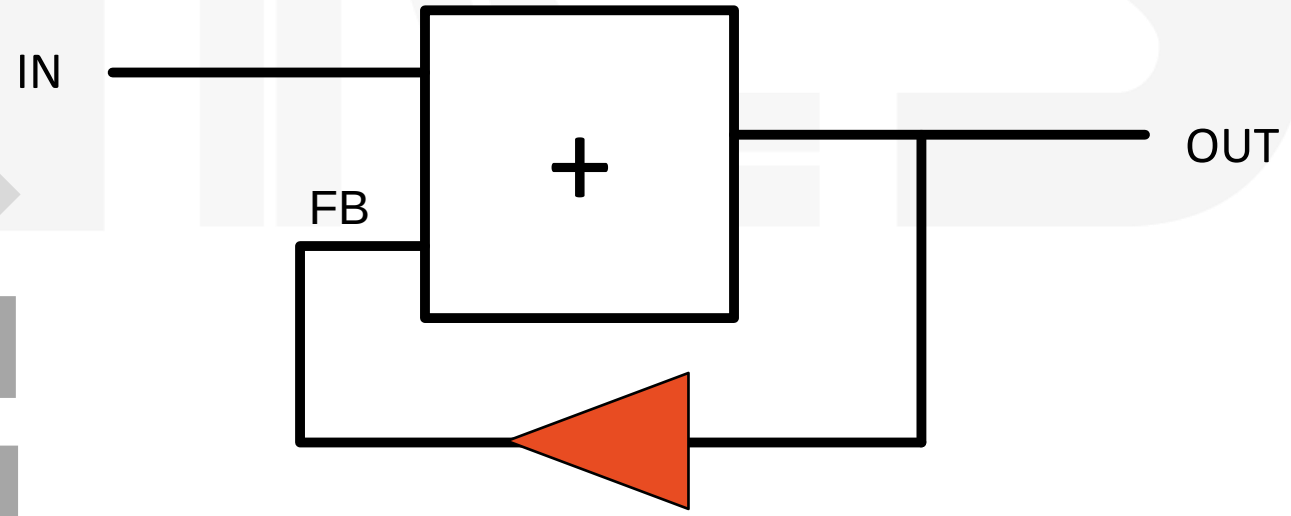
- **We will look at two examples:**
  - An accumulator circuit, where sequential methods are essential to eliminate races.
  - A pipelined system, where sequential methods improve throughput.

What would happen if there were no traffic lights?

Accumulator Example

- An accumulator is a register that sums a list of numbers. Therefore, it feeds back the output back to the input.
- Without a register, there would be the possibility that races would occur, causing erroneous outputs.
- We need to delay the output until the original calculation is finished.



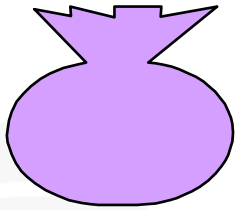


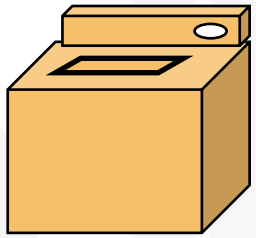
Accumulator Example

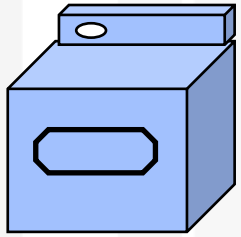
- It is essential to use sequential logic when paths have different delays but need to converge together.
- We always have to *slow* our fast paths down so they arrive along with our slowest path.
- If we could make all paths have equal delays, we wouldn't need sequential logic, but this is really hard (almost impossible) to do.

Laundry Example

- **Small laundry has one washer, one dryer and one operator, it takes 90 minutes to finish one load:**
  - Washer takes 30 minutes
  - Dryer takes 40 minutes
  - “Operator folding” takes 20 minutes

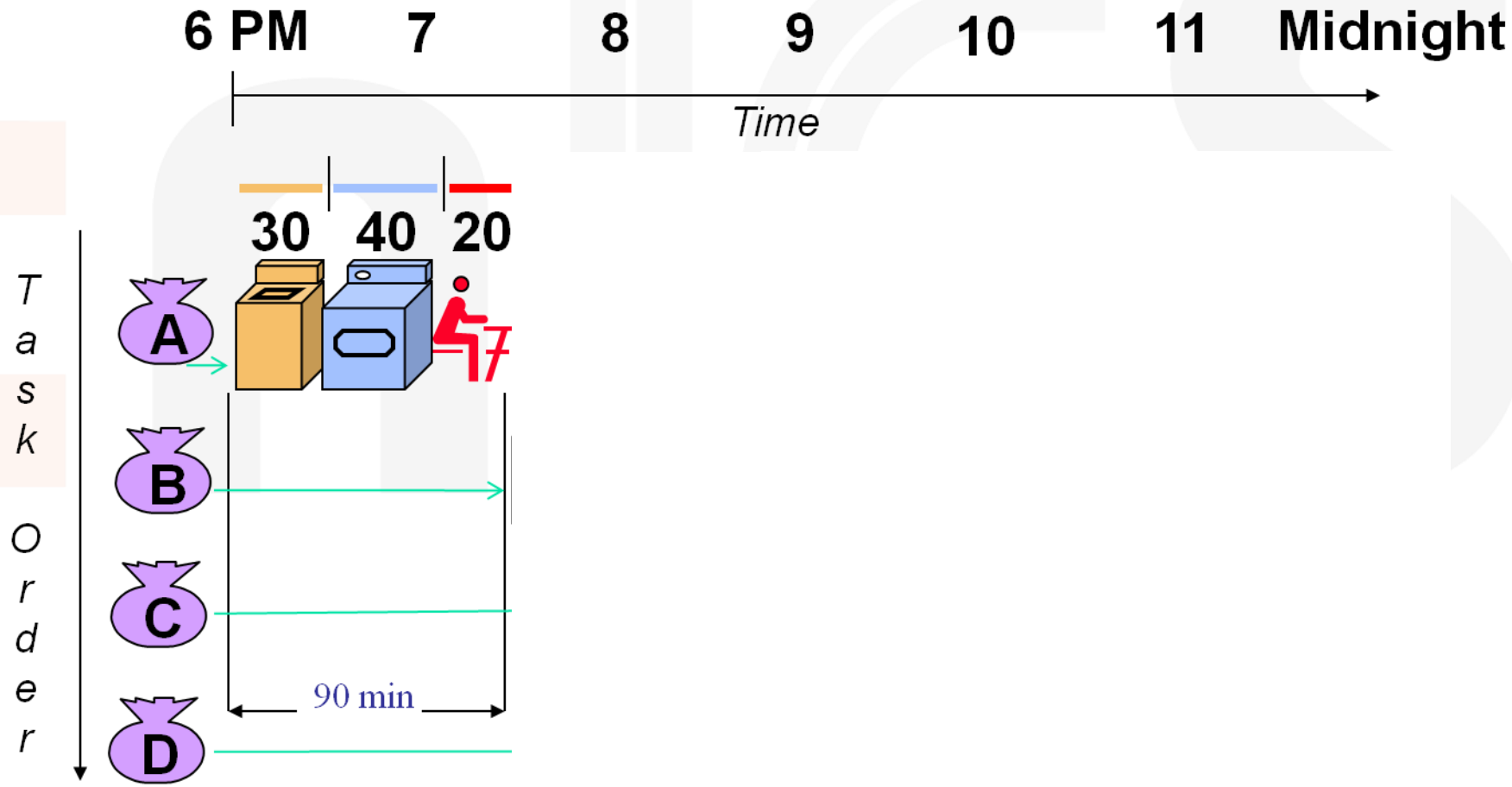






Sequential Laundry

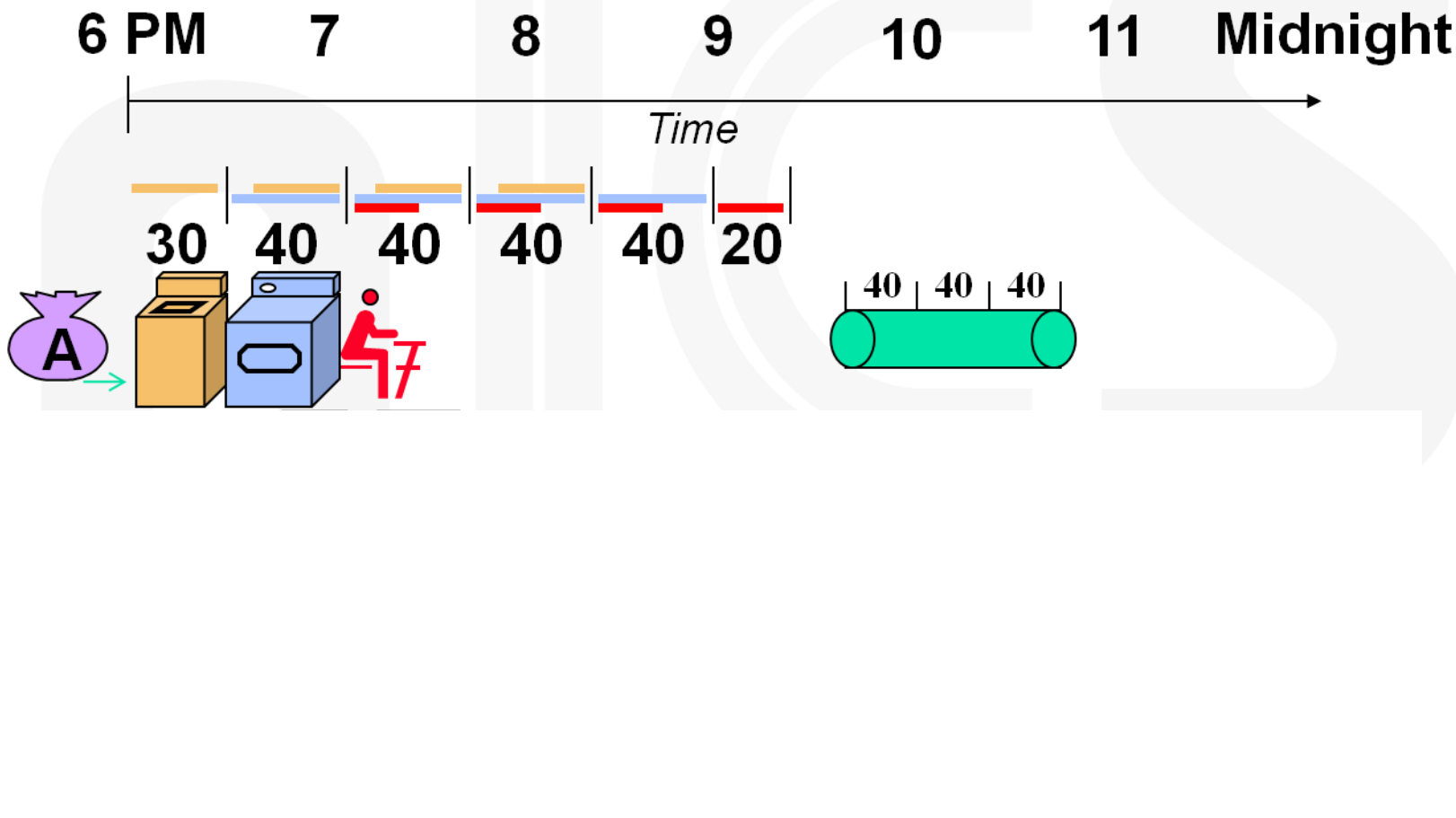
- It takes 90 minutes to finish one load.
- The process is sequential.



- Sequential laundry takes 6 hours for 4 loads.

Pipelined Laundry

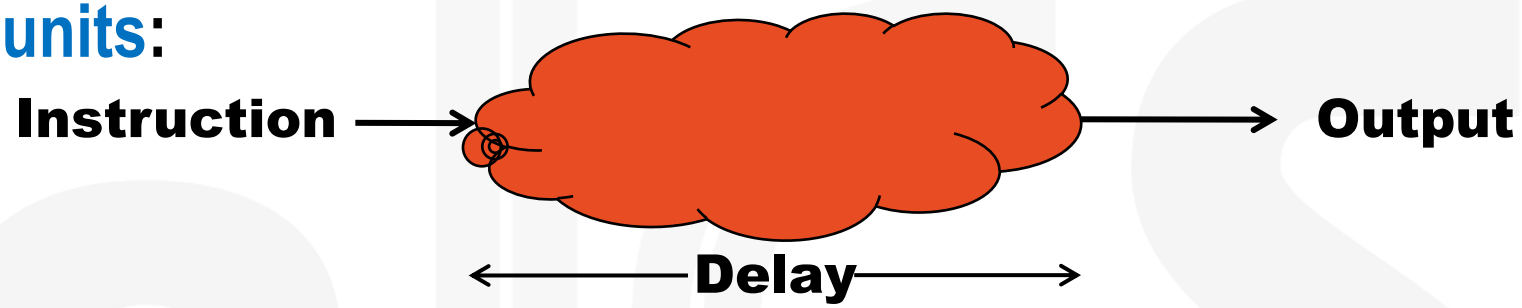
- Every 40 minutes a new load starts and a new load ends.



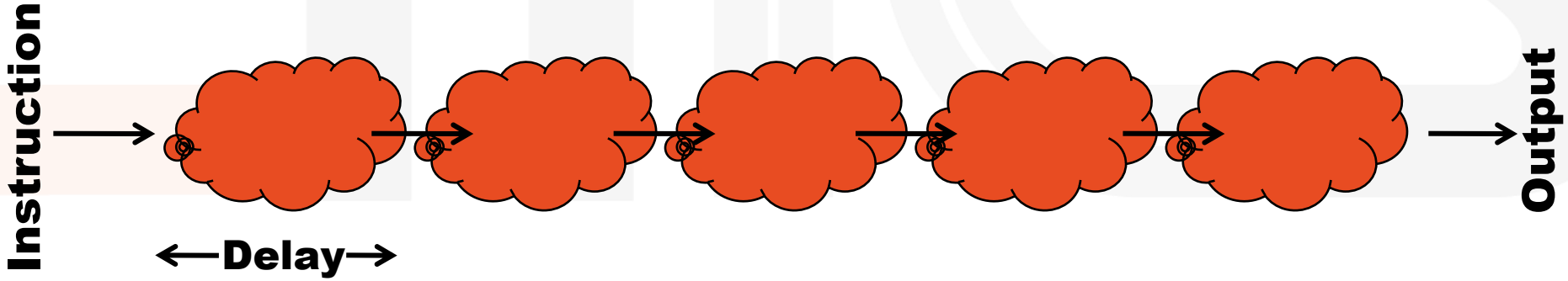
- Pipelined laundry takes 3.5 hours for 4 loads

Pipelining Data

- If it takes 10 time units to process an instruction, we could perform one instruction every 10 time units:



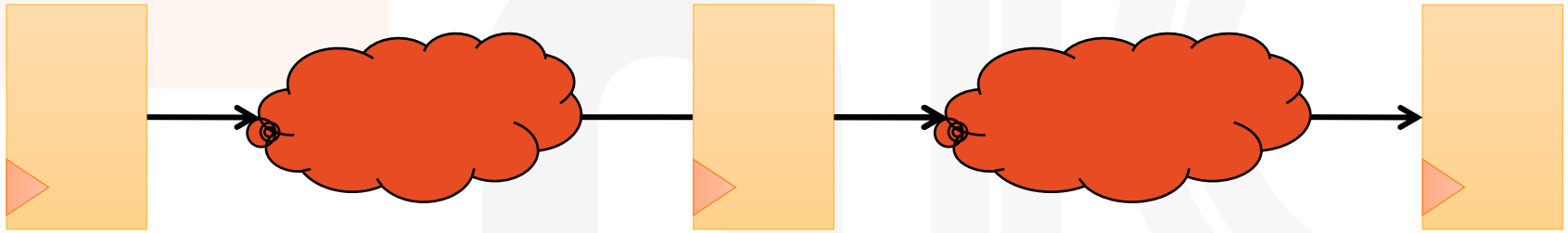
- But if we divide the process into 5 tasks that take 2 time units each:



- We can start a new instruction every 2 time units.
- And after filling the pipe, we finish an instruction every 2 units.

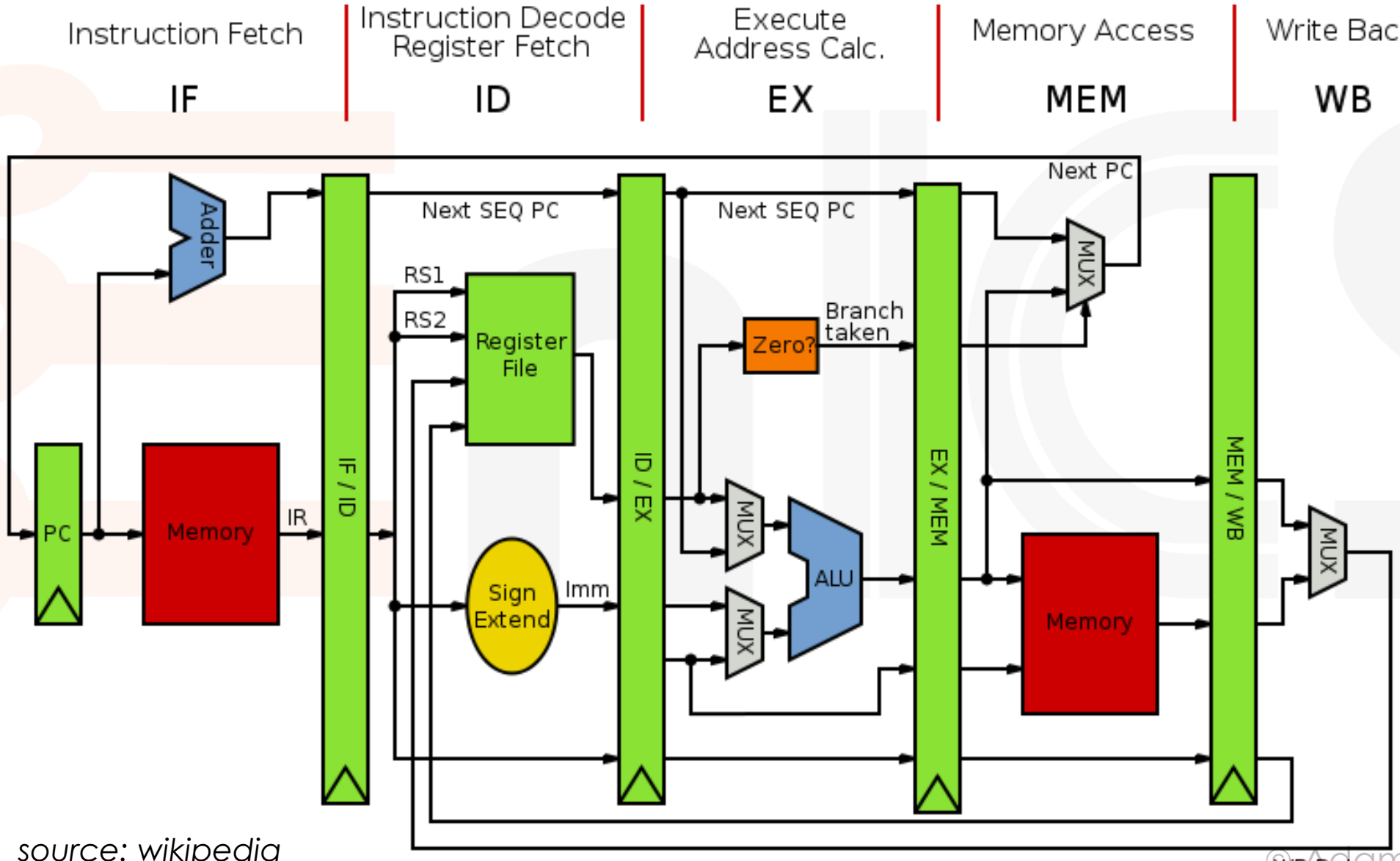
Pipelining Data

- But some stages may be faster than others, so we need to hold the input to each stage constant until the previous stage is done.
- We achieve this by adding a *register* in between the stages.



- So by using a pipeline, we can make our slowest path shorter and therefore reduce the delay between actions.
- All data paths are built using a pipeline of some sort, either to *eliminate races* or to *increase throughput*.

Classic 5-Stage RISC Pipeline

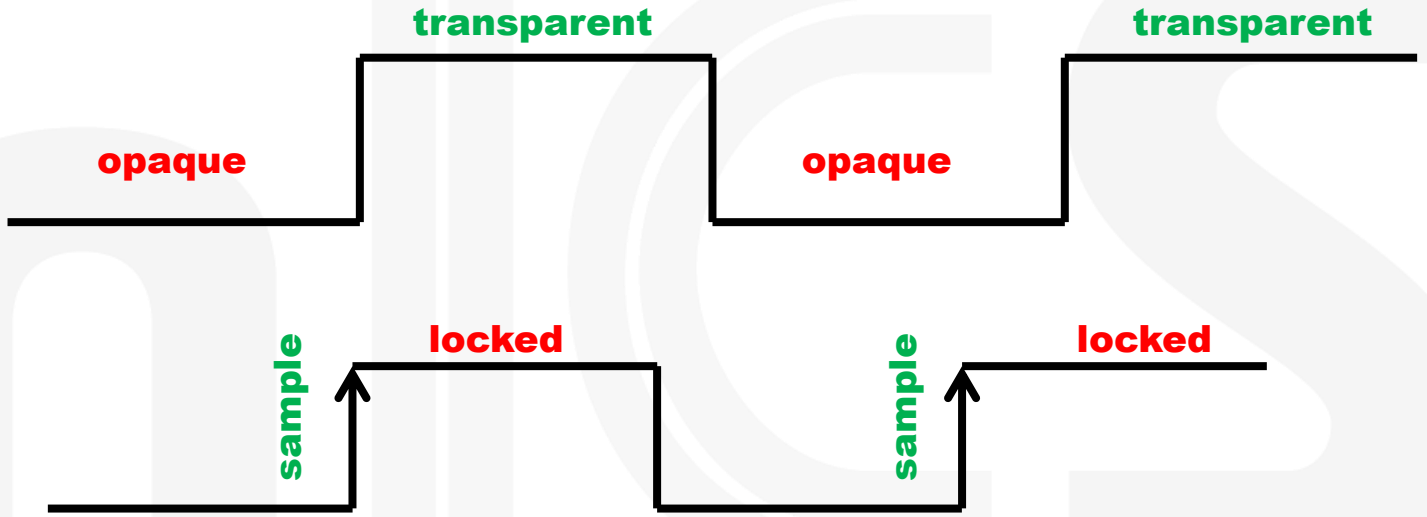


source: wikipedia

Sequential Logic Elements

Naming Conventions

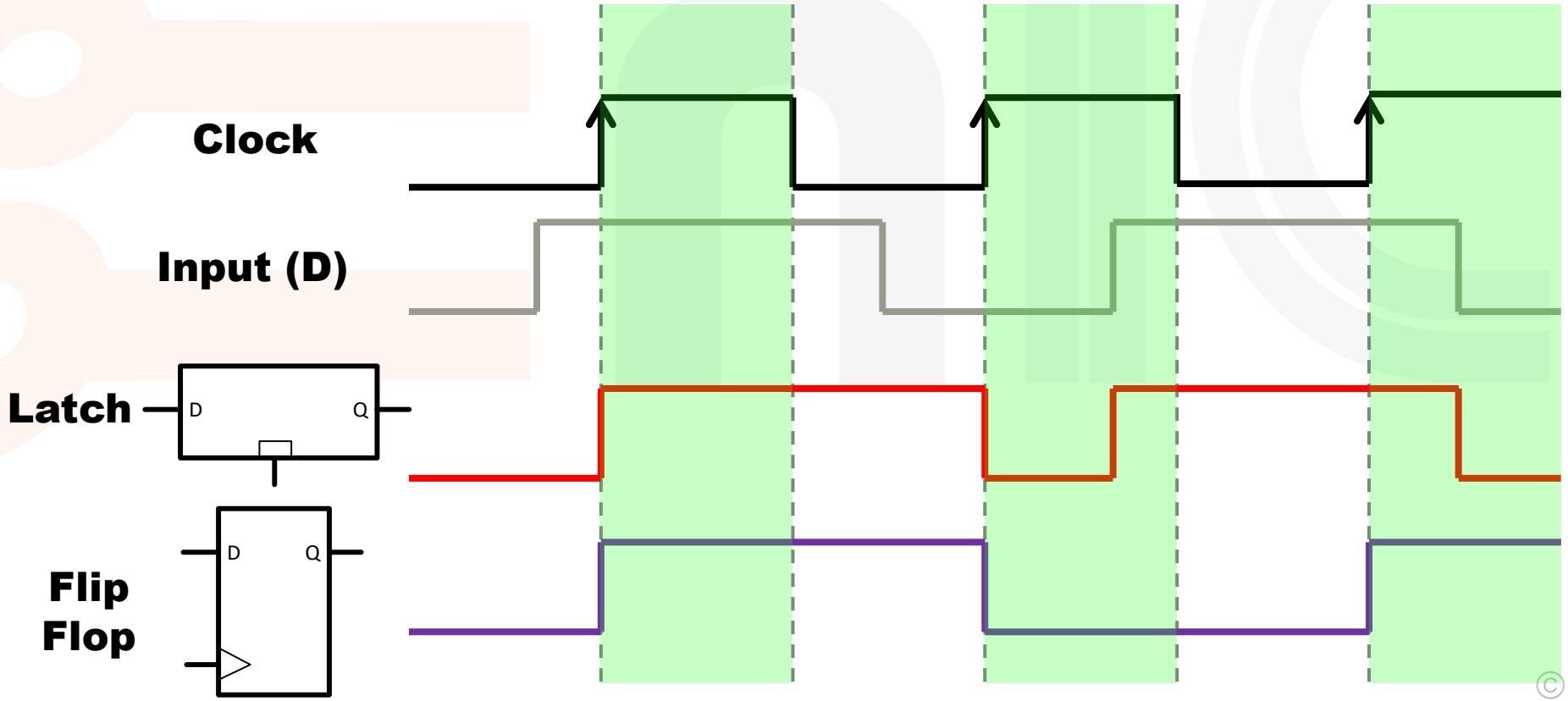
- **In our course we relate to *registers* as follows:**
  - a *latch* is level sensitive
  - a *flip-flop* is edge-triggered



- **There are many different naming conventions**
  - For instance, many books call any bi-stable element a flip-flop (such as an SR Latch)
  - However, this leads to confusion, so we will use the convention above (as used in industry).

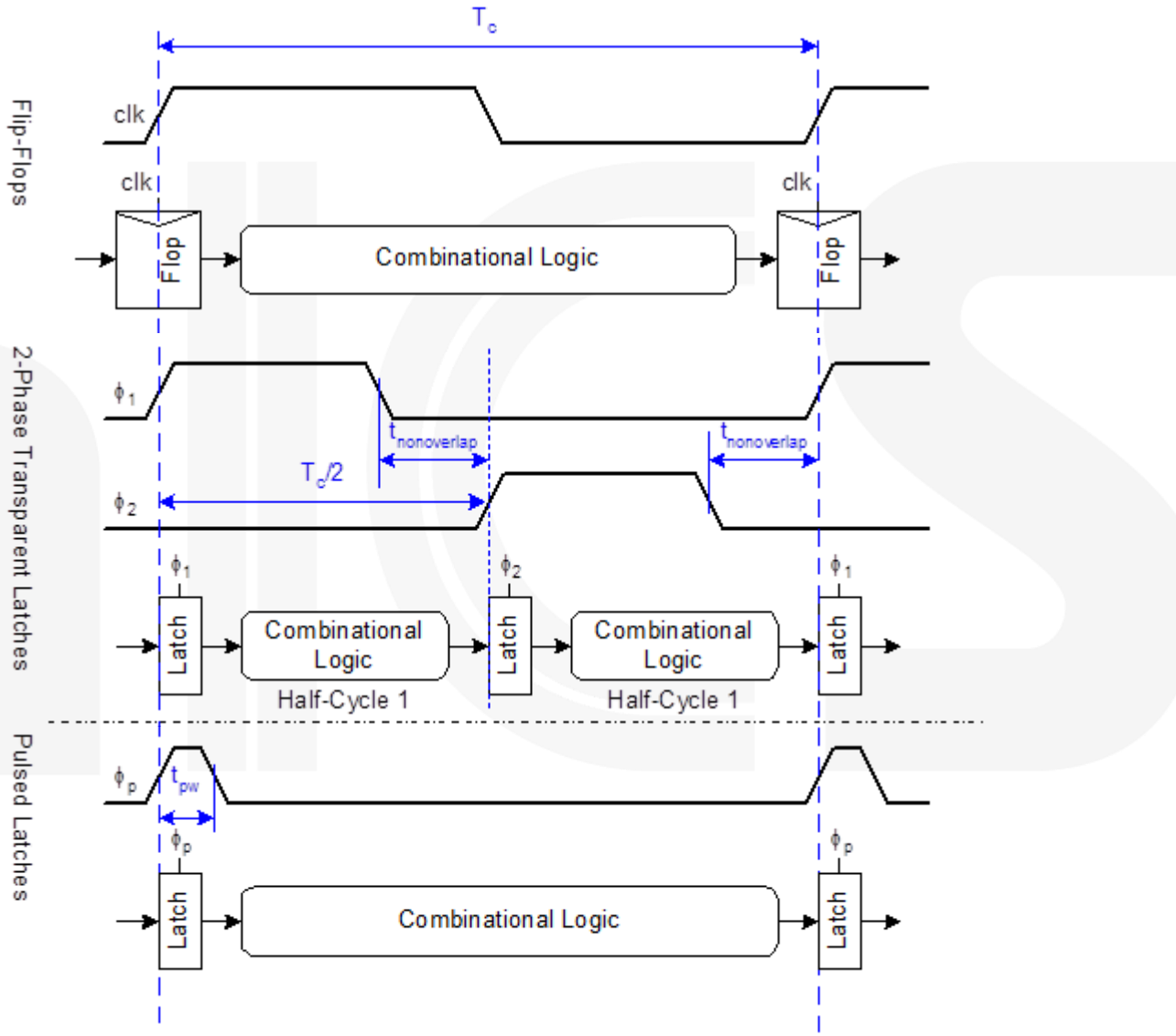
Latch Vs. Register

- During high clock phases, a latch is transparent, *latching* the input on the falling edge.
- However, a Flip Flop only *samples* the input on the rising edge.



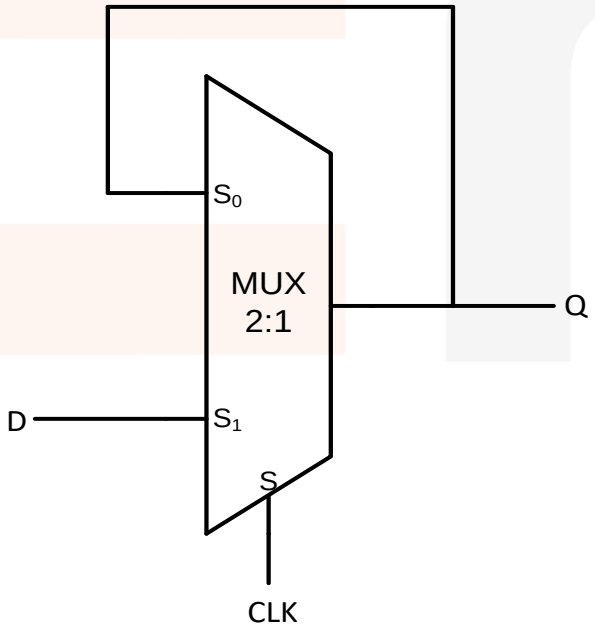
Latch Vs. Register

- **3 main options for sequential timing:**
  - Using Flip-Flops
  - Using Transparent Latches
  - Using Pulsed Latches

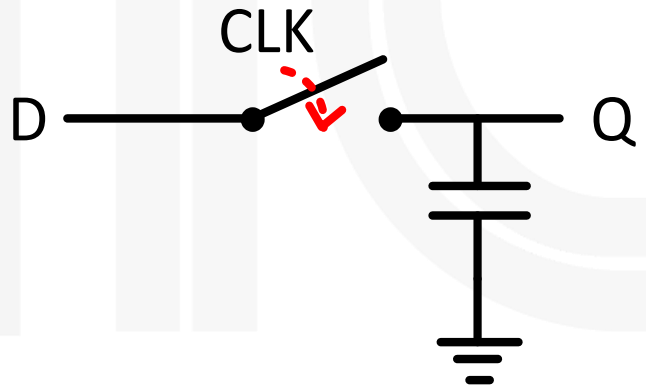


Static Vs. Dynamic Latch

- A *static latch* stores its output in a static state
- A *dynamic latch* uses temporary capacitance to store its state.
- As with logic, this provides a trade-off between area, speed and reliability.



Static Latch

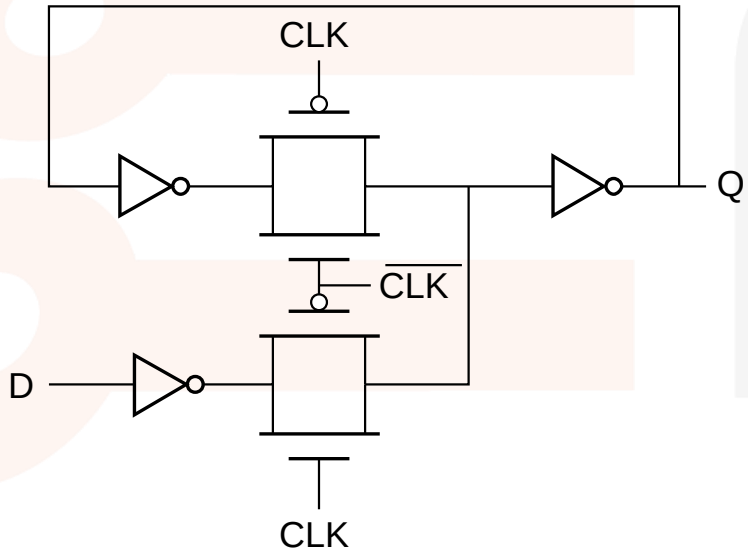


Dynamic Latch

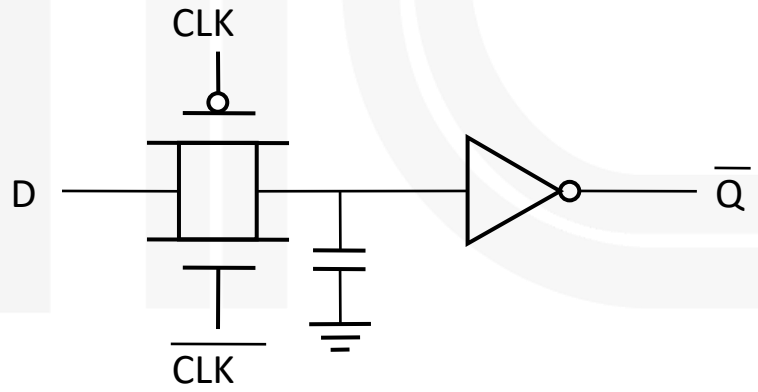
Static Vs. Dynamic Latch

- **Some basic implementations of static and dynamic latches.**

Static

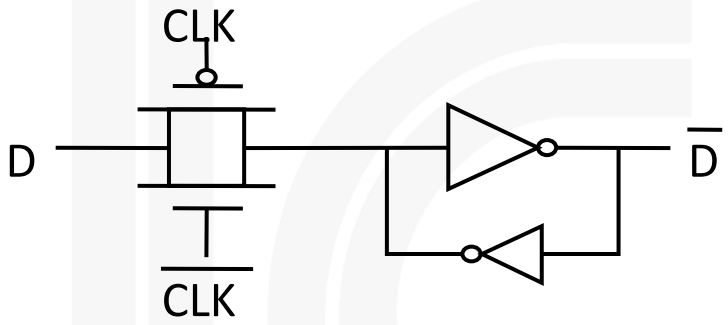


Dynamic

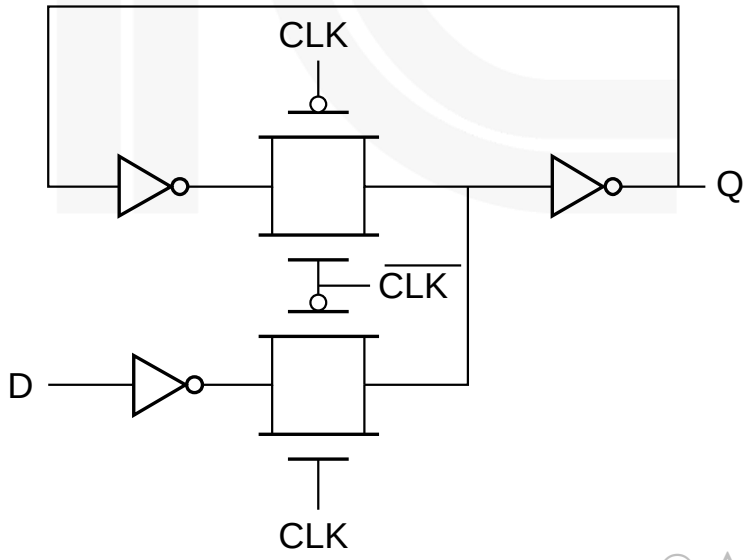


Ratioed vs. Non-Ratioed Latch

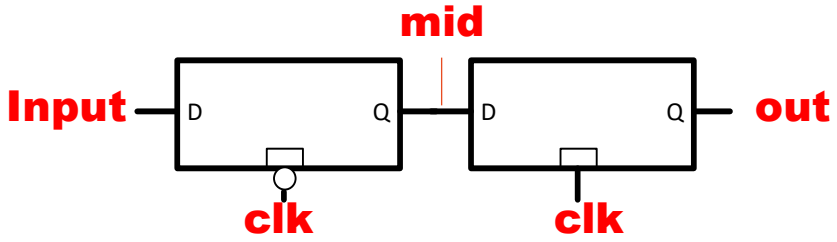
- **A static latch can be made by using a feedback inverter.**
  - The TG (with the driver before it) must overcome the feedback inverter to write into the latch.



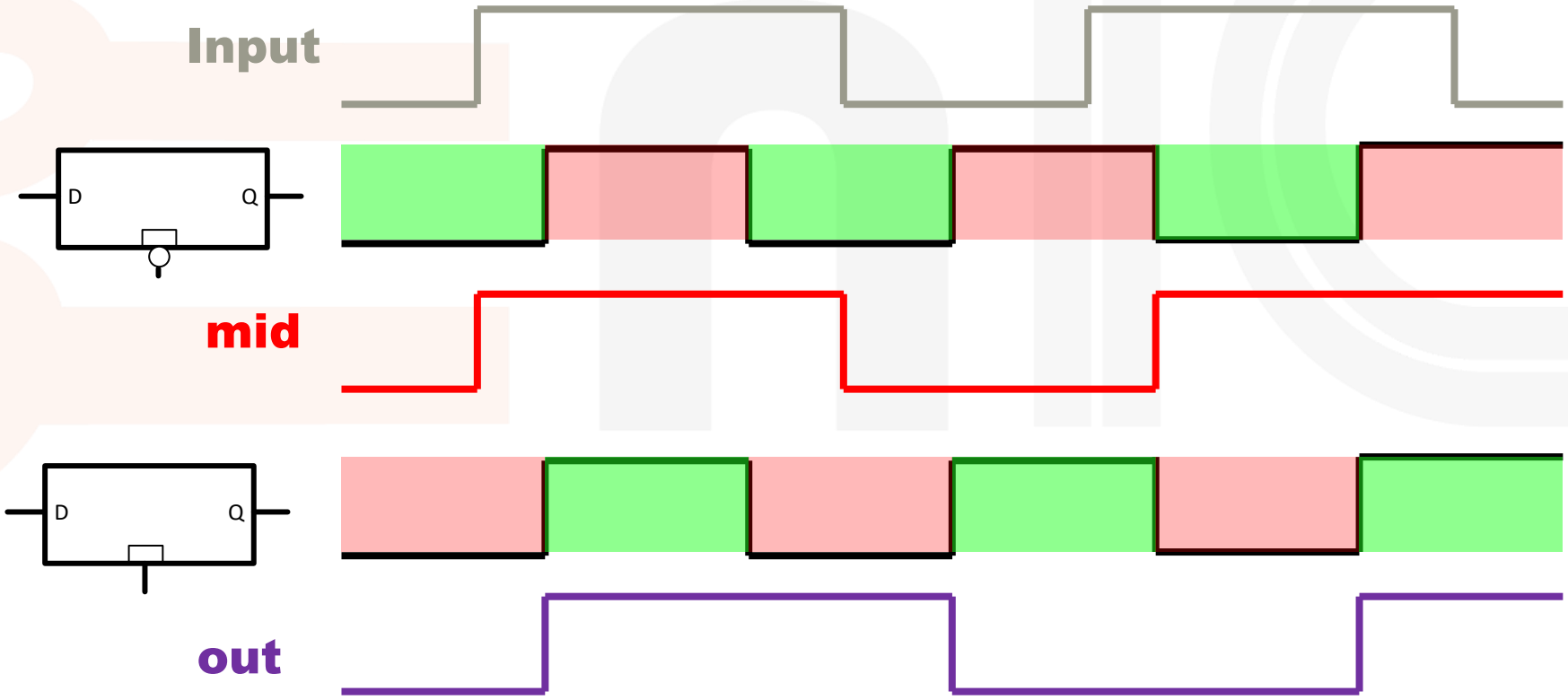
- **But it is usually more robust to create a mux-based non-ratioed latch.**
  - At the expense of size.



Making a Flip Flop

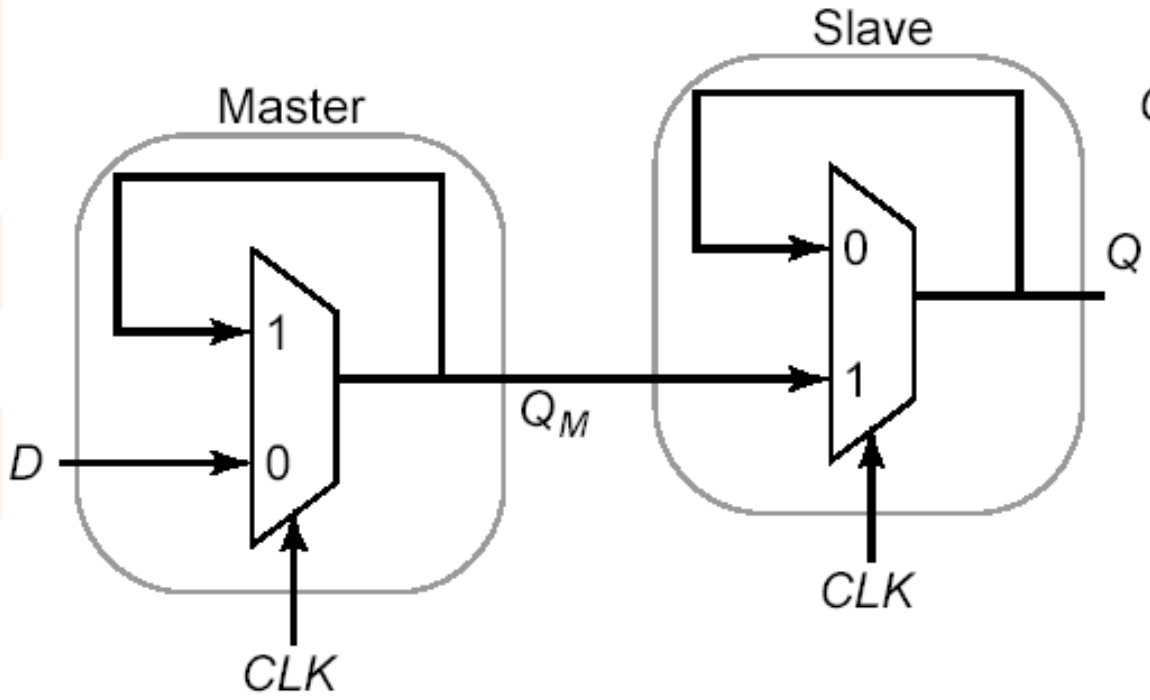


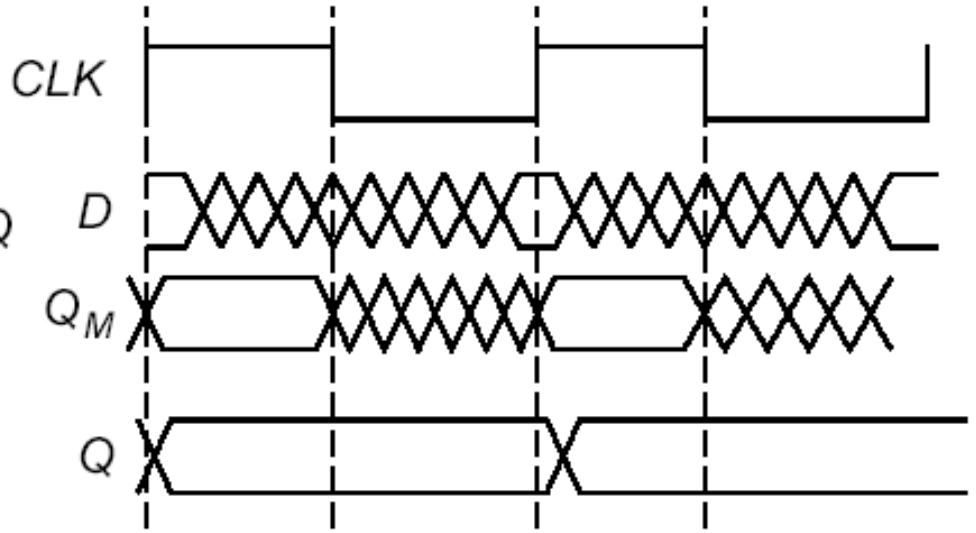
- Conceptually, we can create an *edge triggered flip-flop* by combining two *opposite polarity latches*:



Master-Slave (Edge-Triggered) Register

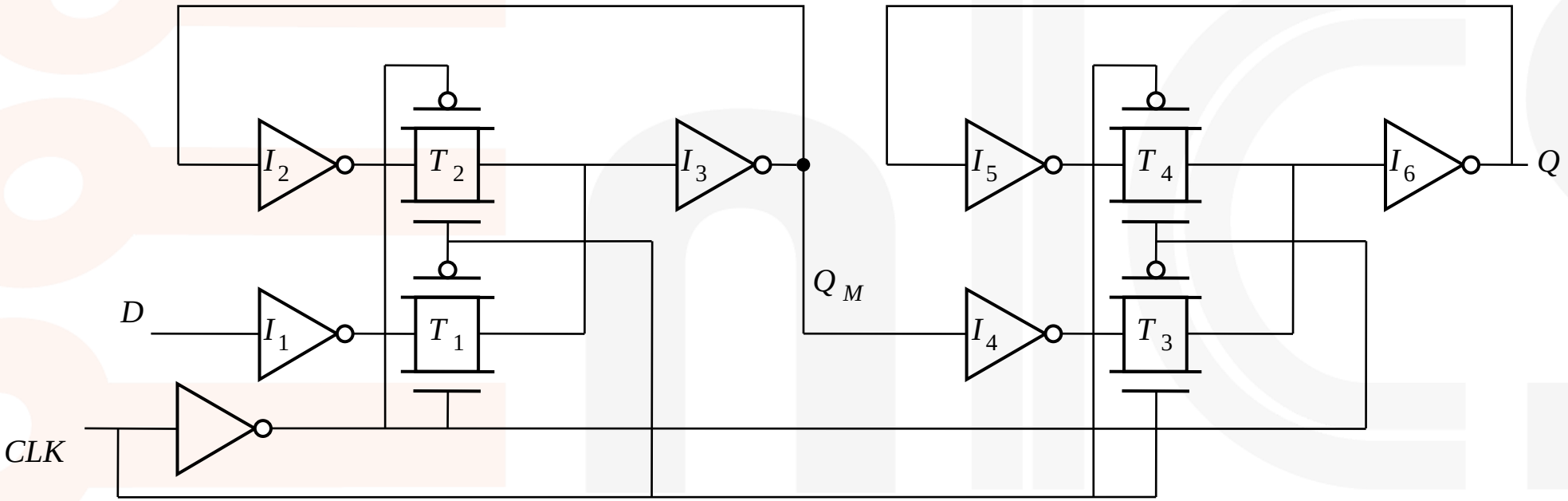
- Two opposite latches trigger on edge
- Also called master-slave latch pair





Master-Slave Register

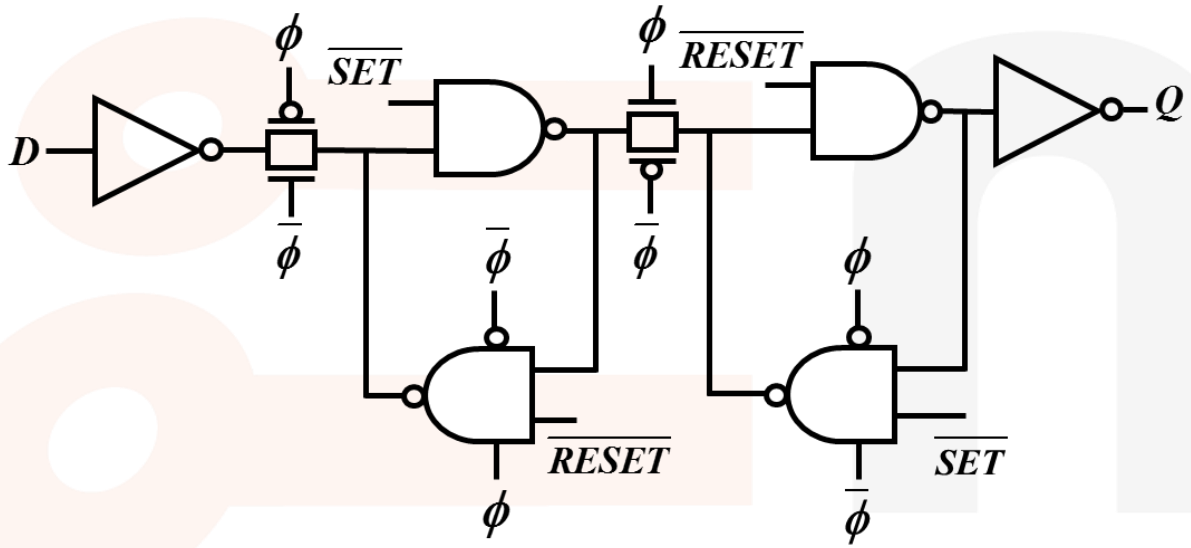
- **Multiplexer-based latch pair**



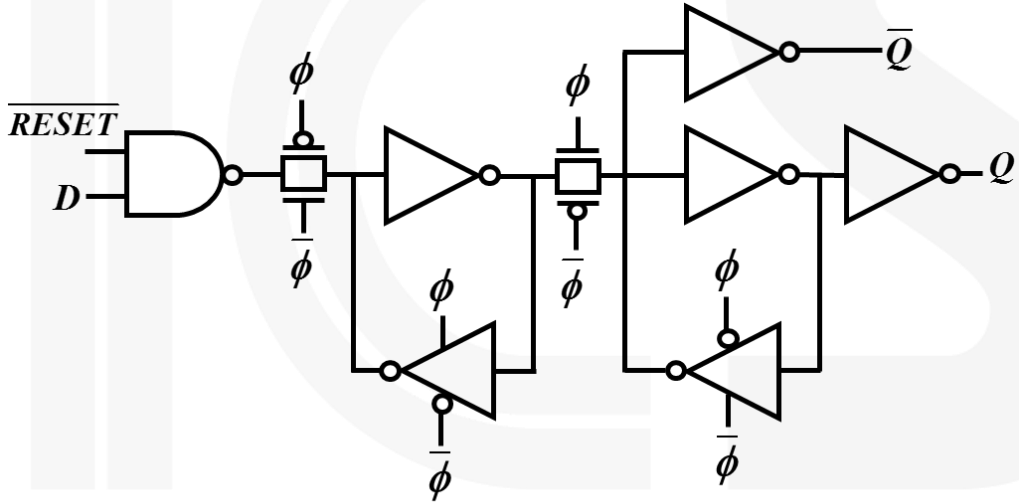
- **How many *transistors* make up this flip-flop?**
- **What is its *clock load*?**

Resettable Flip Flops

Asynchronous Set/Reset Flip-Flop

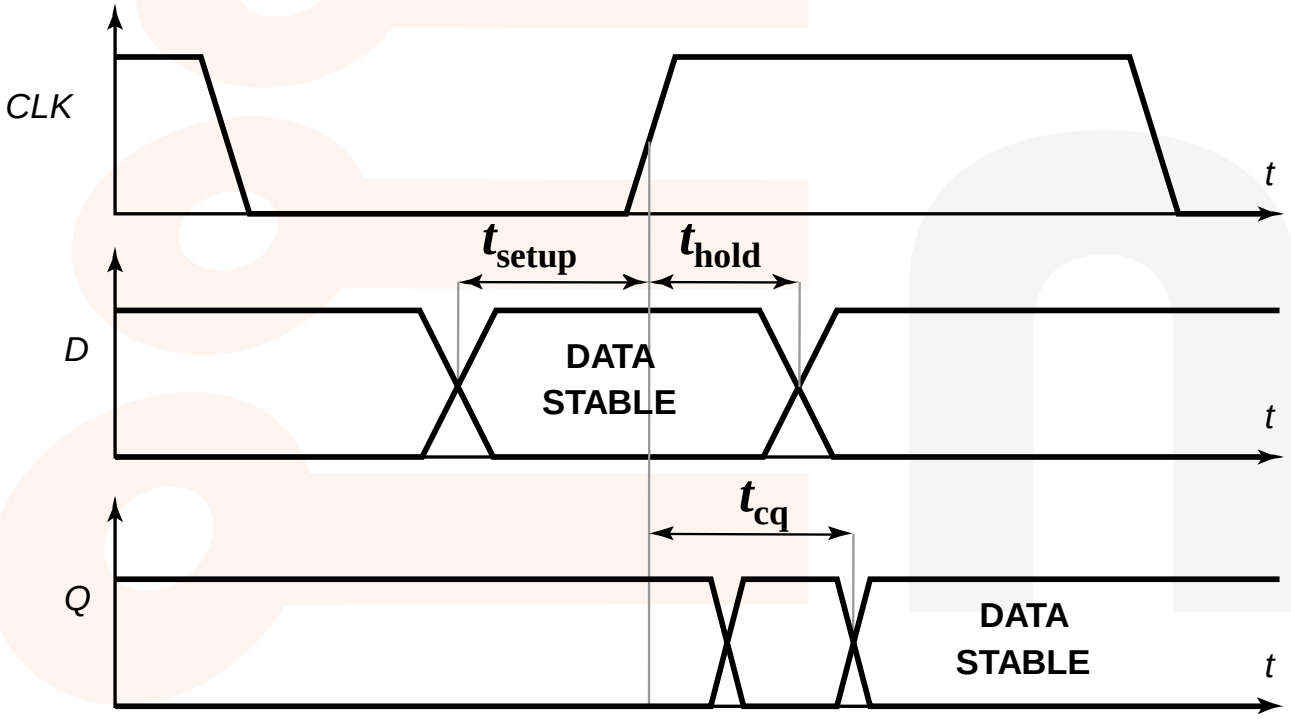


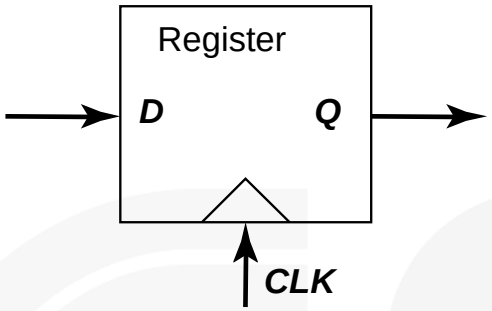
Synchronous Reset Flip-Flop



Timing Parameters of Sequential Elements

Timing Definitions





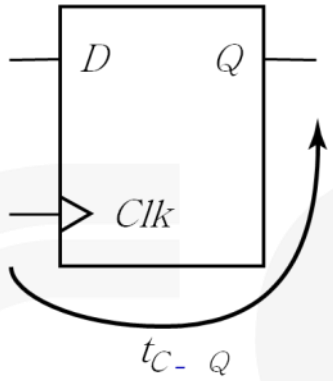
t_{cq} – propagation delay

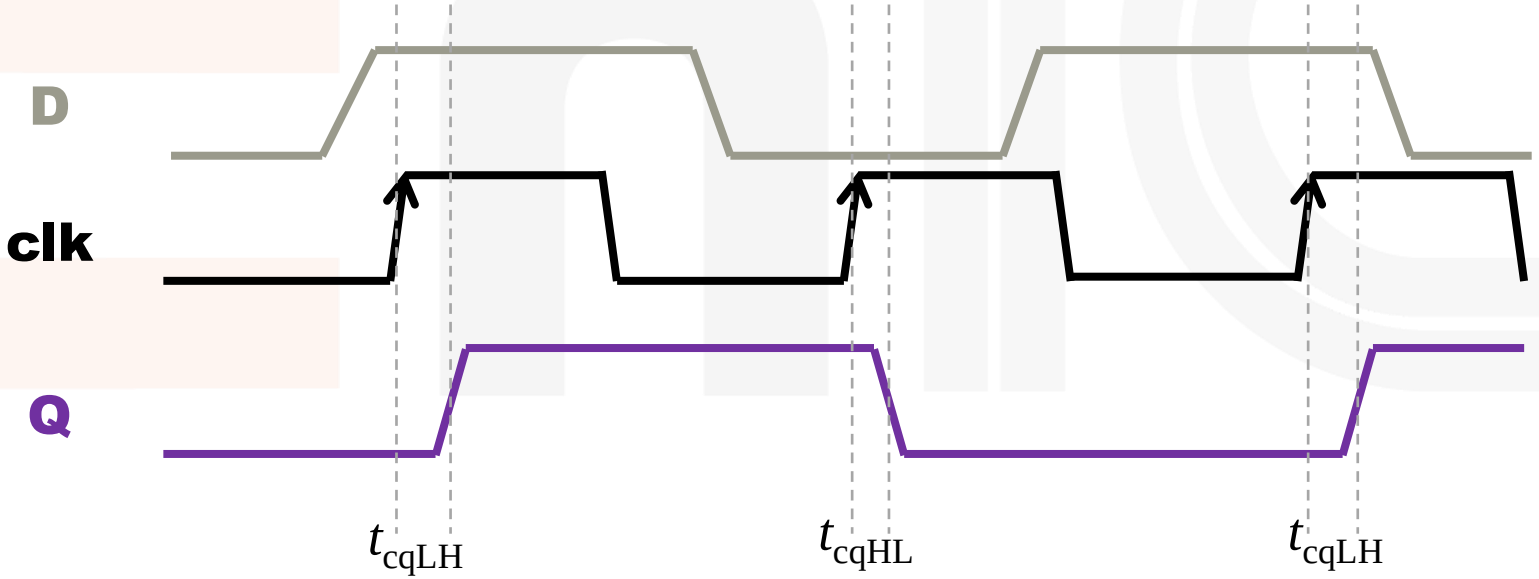
t_{setup} – setup time

t_{hold} – hold time

Clk-Q Delay - t_{cq}

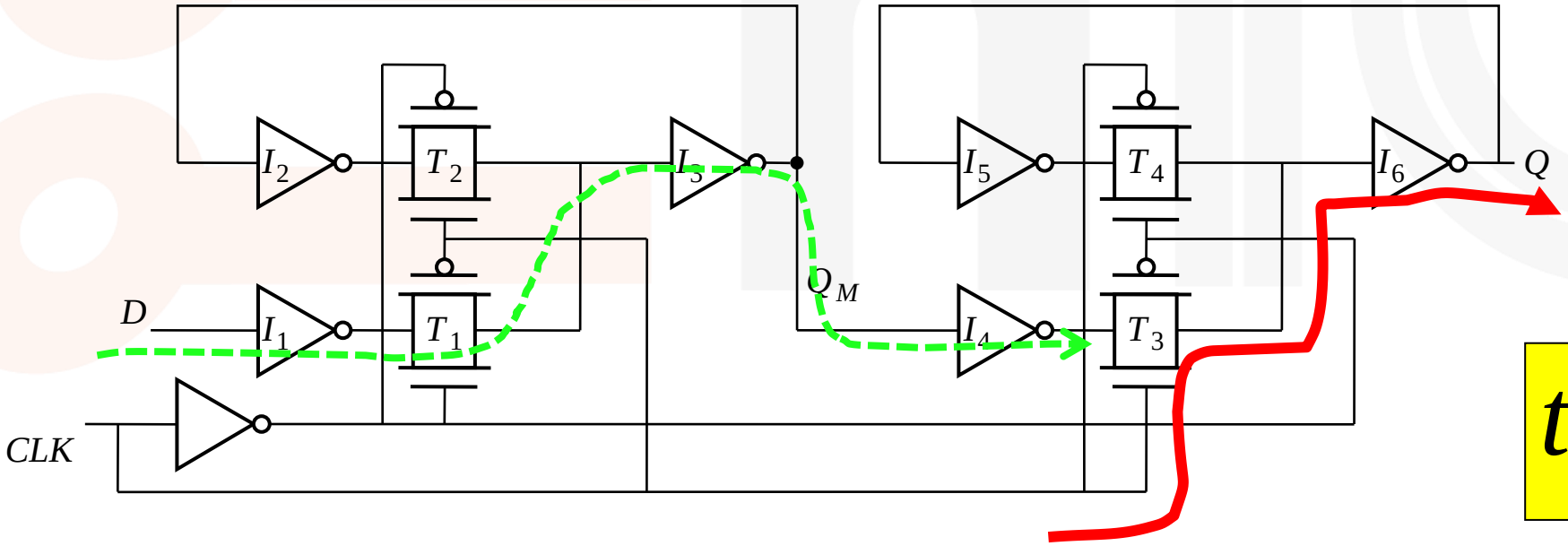
- t_{cq} is the time from the clock edge until the data appears at the output.
- The t_{cq} for rising and falling outputs is different.





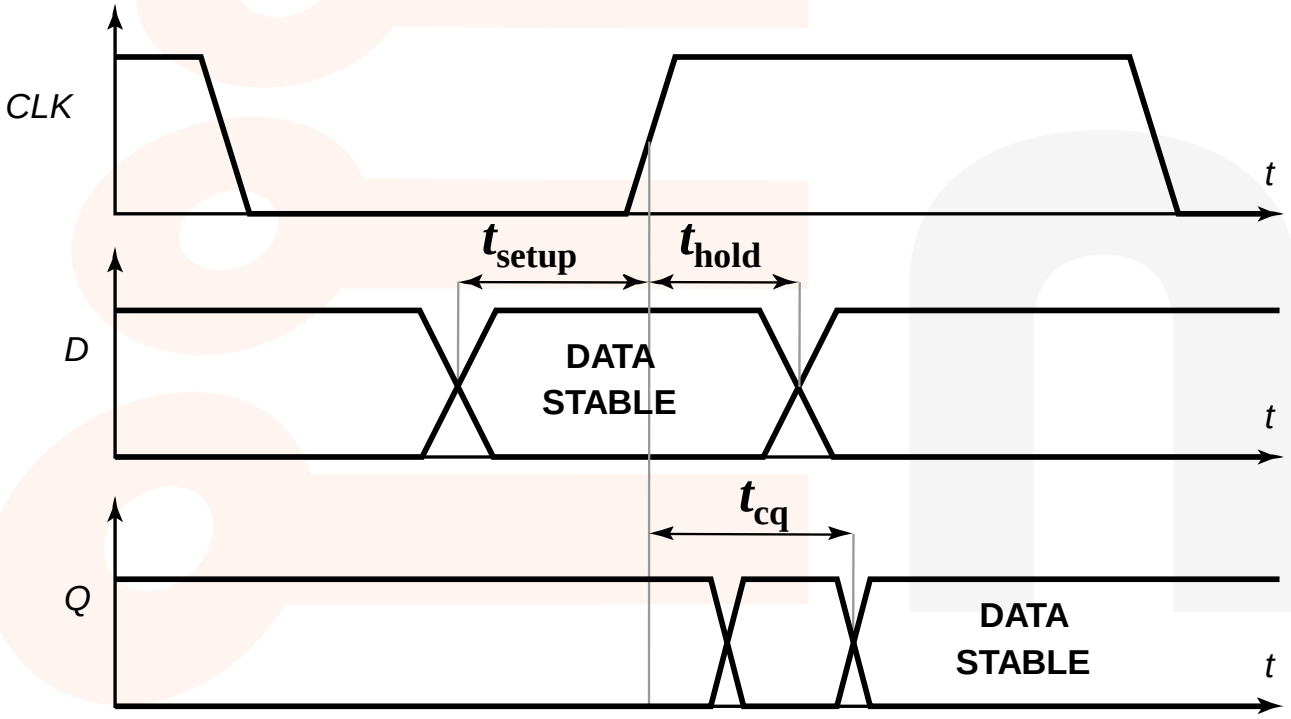
Mux based FF – t_{cq} Calculation

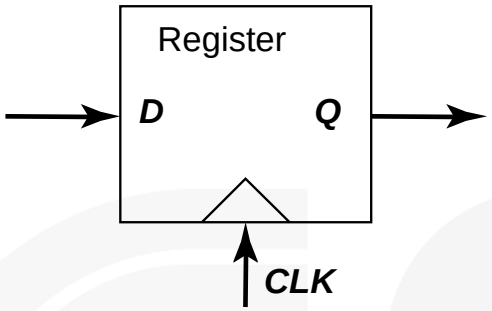
- During low clock edge, data traverses slave and "waits" for the clock at pass gate input.
- When clock rises, data has to go through pass gate and inverter.



$$t_{cq} = T_3 + I_6$$

Timing Definitions





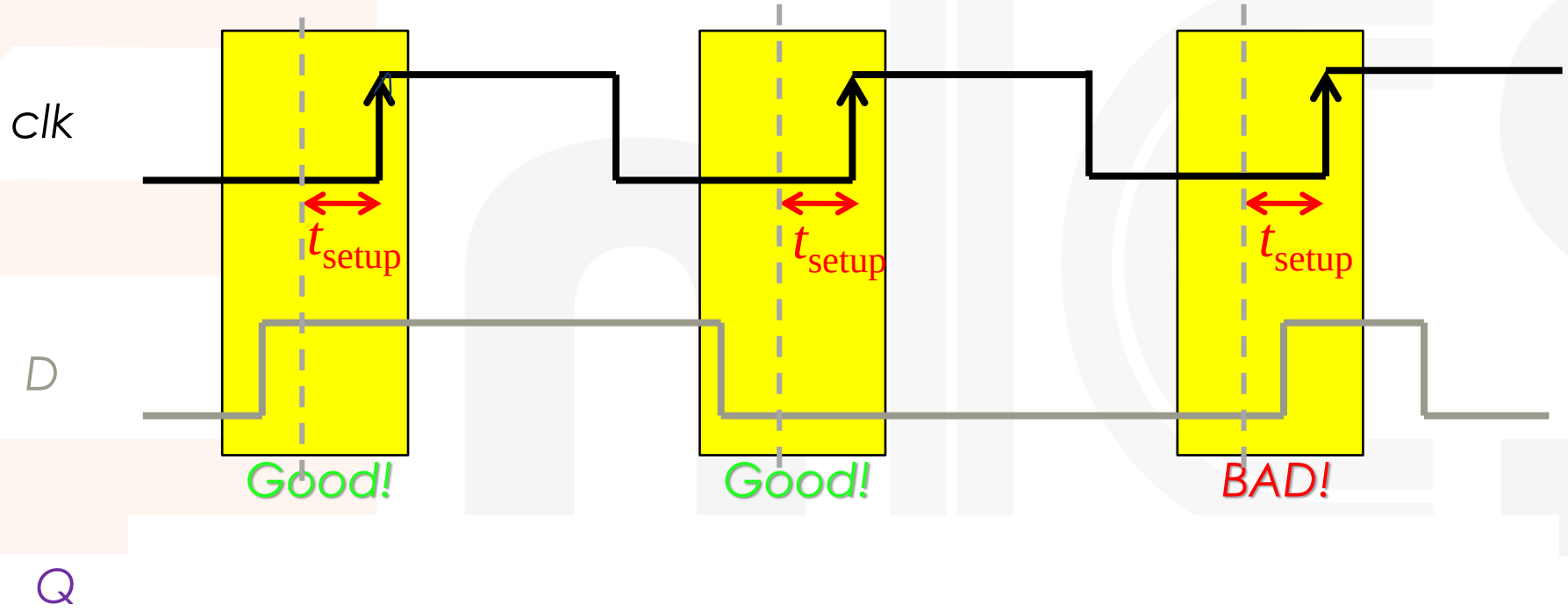
t_{cq} – propagation delay

t_{setup} – setup time

t_{hold} – hold time

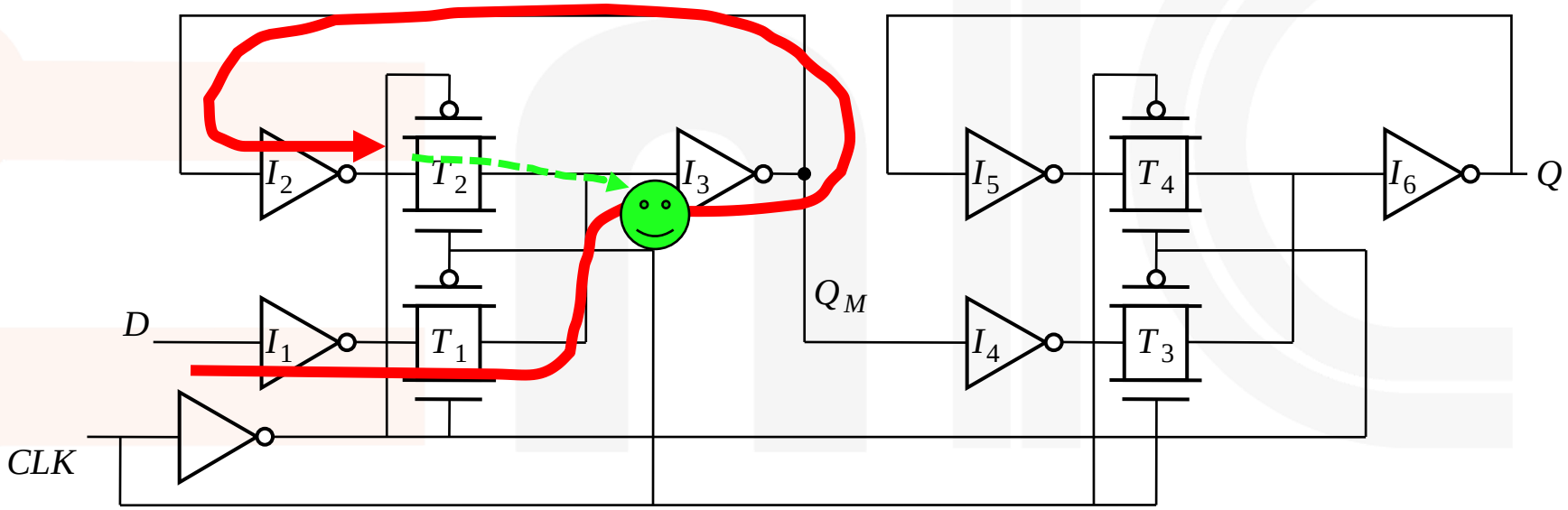
Setup Time - t_{setup}

- **Setup time** is the time the data has to arrive ***before the clock*** to ensure correct sampling.



Mux based FF – t_{setup} Calculation

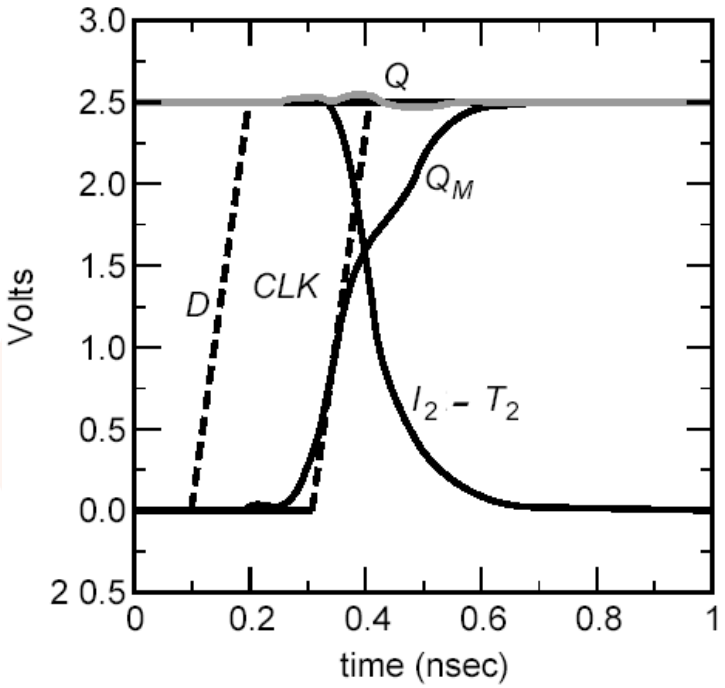
- **Before clock edge, data should have propagated to the latching pass gate, or else data will be restored to the previous state.**



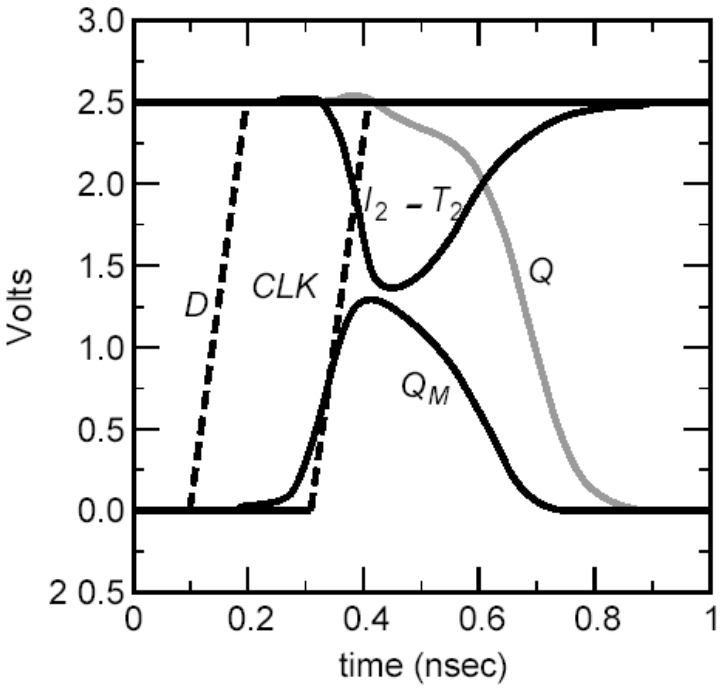
$$t_{setup} = I_1 + T_1 + I_3 + I_2 = T + 3I$$

Timing Analysis - Setup Time

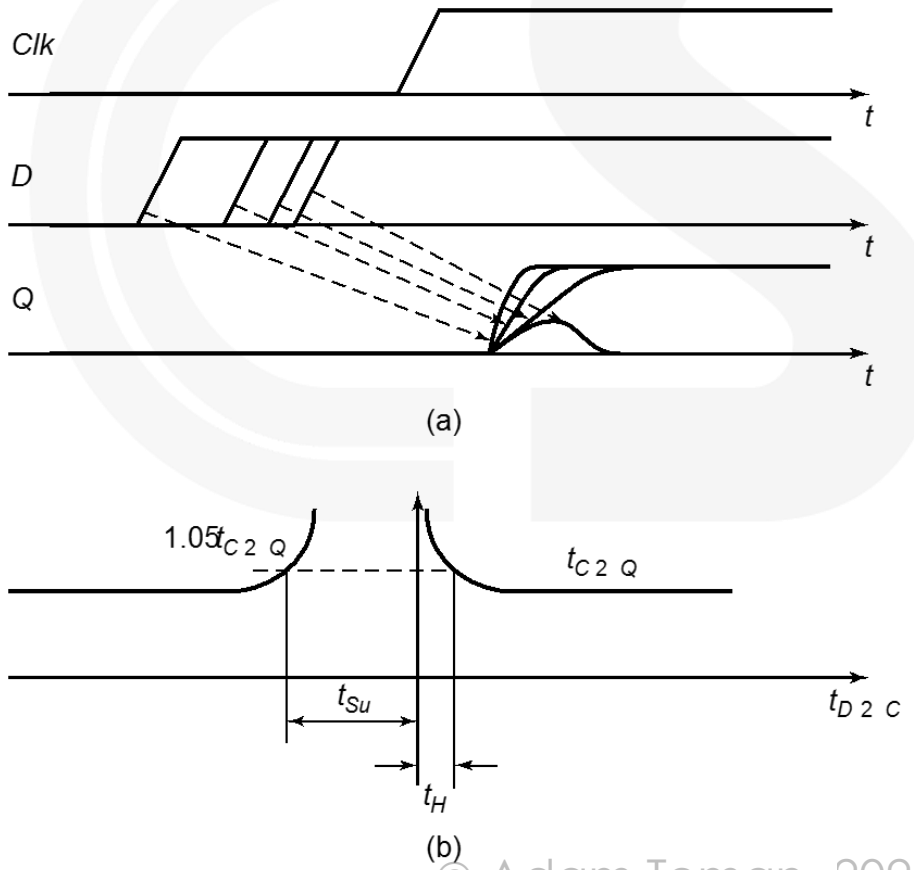
- To obtain the setup time of the register while using SPICE, we progressively skew the input with respect to the clock edge until the circuit fails.



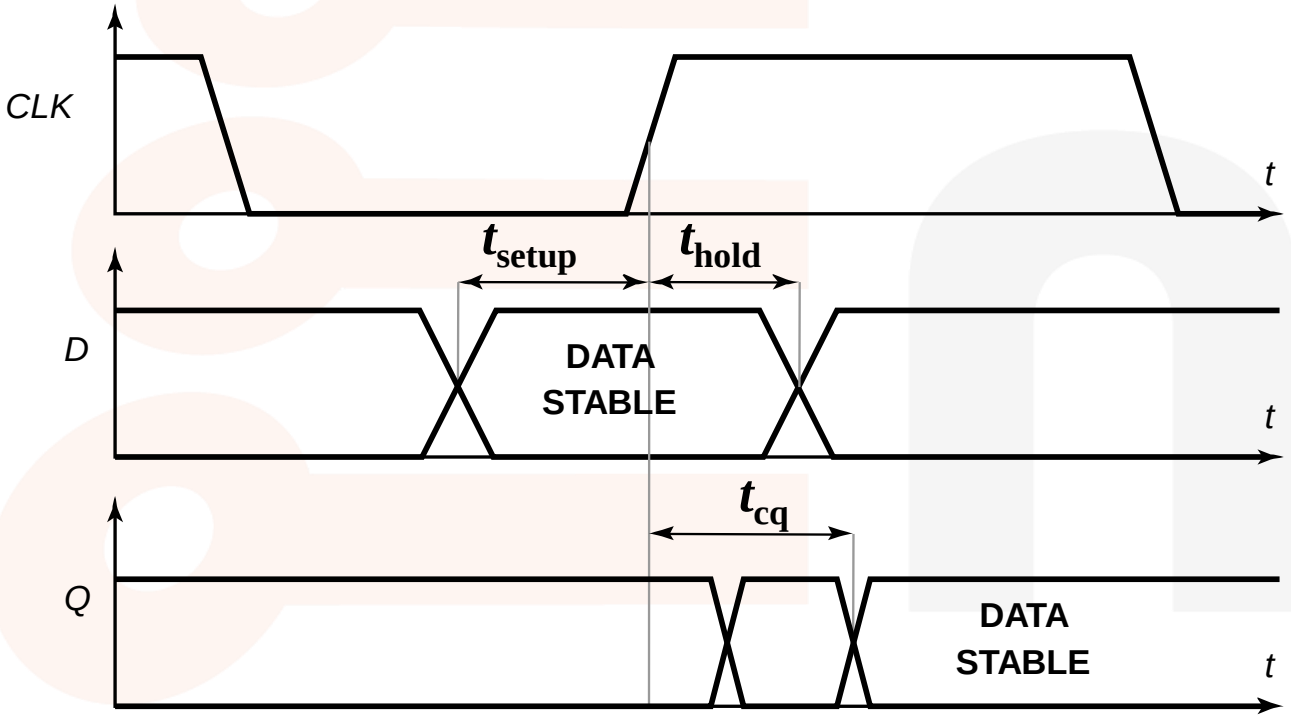
(a) T_{setup}= 0.21 nsec

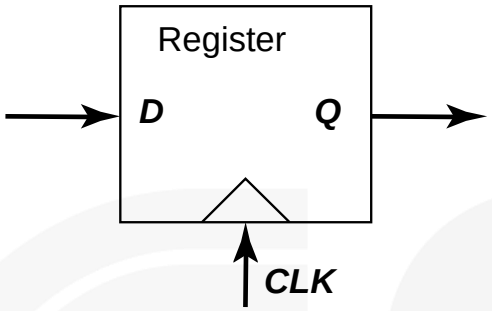


(b) T_{setup} = 0.20 nsec



Timing Definitions





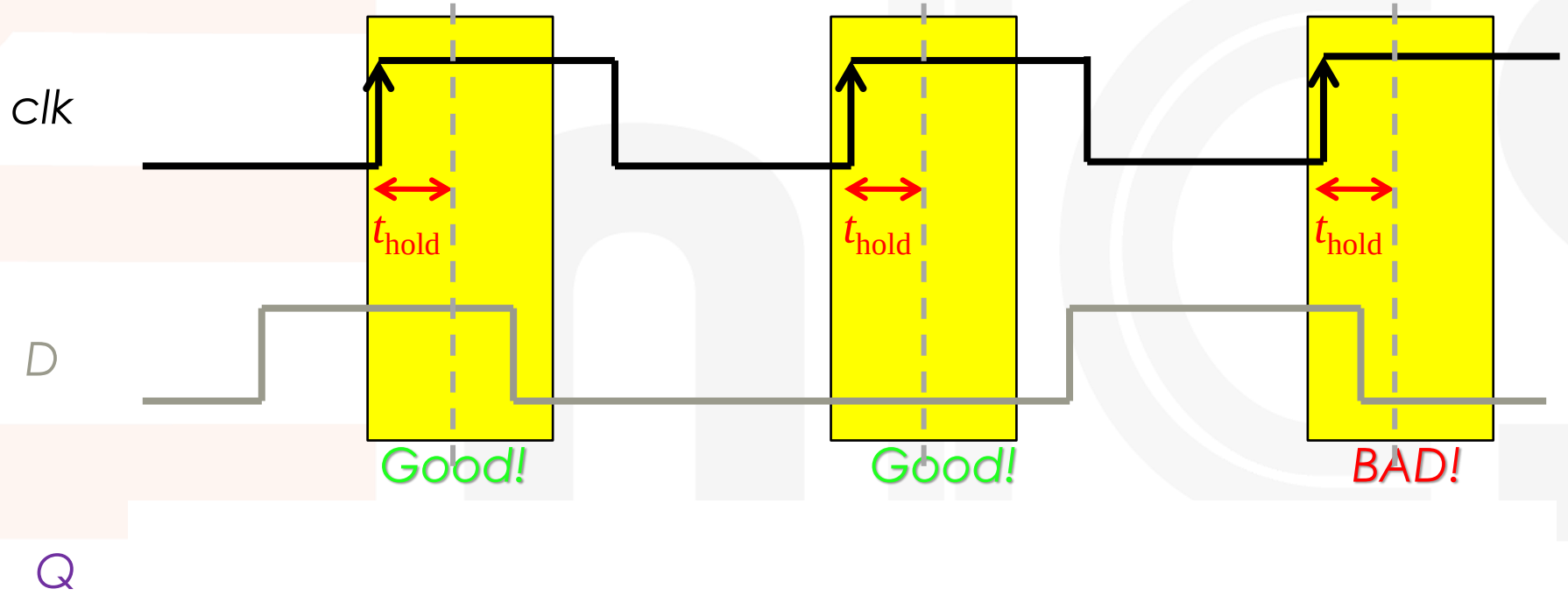
t_{cq} – propagation delay

t_{setup} – setup time

t_{hold} – hold time

Hold Time - t_{hold}

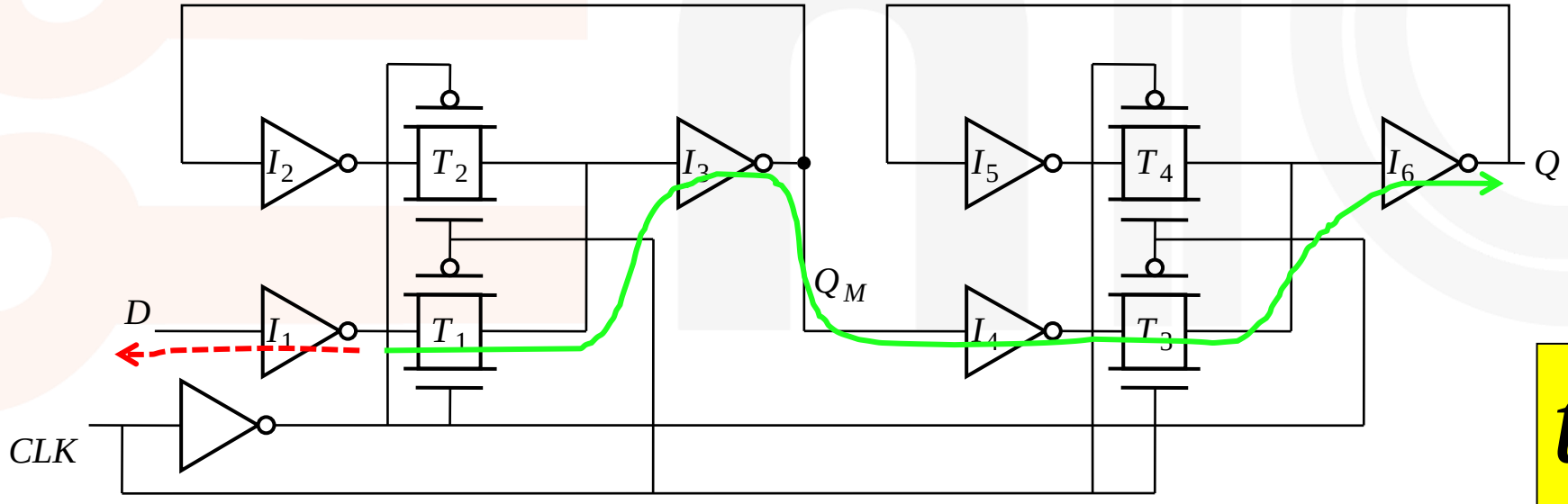
- **Hold time is the time the data has to be stable *after the clock* to ensure correct sampling.**



- **Often (optimally), Hold Time is *negative*!**

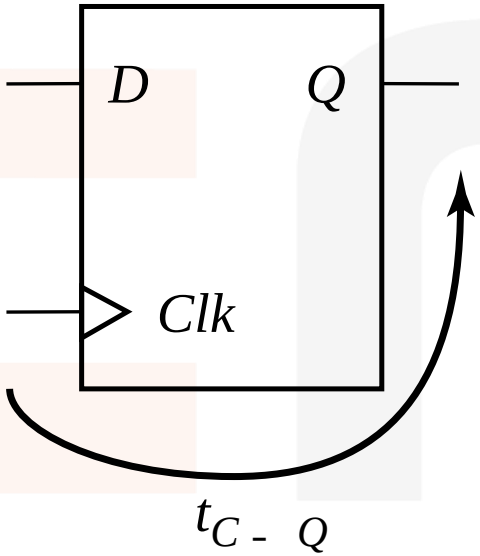
Mux based FF – t_{hold} Calculation

- When the clock rises, T_1 closes, latching the data at the output of I_1.
- Therefore, any changes made $t_{pd}(I_1)$ before the clock will not traverse.
- The hold time is $-t_{pd}(I_1)$

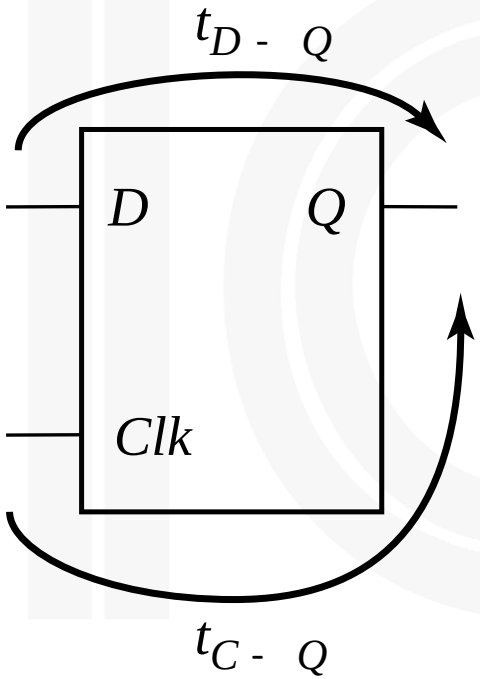


$$t_{hold} = -I_1$$

Characterizing Timing



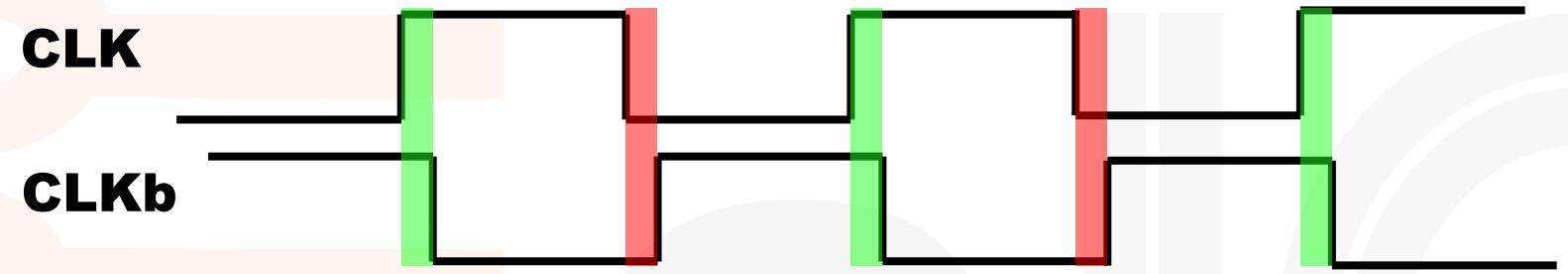
Register

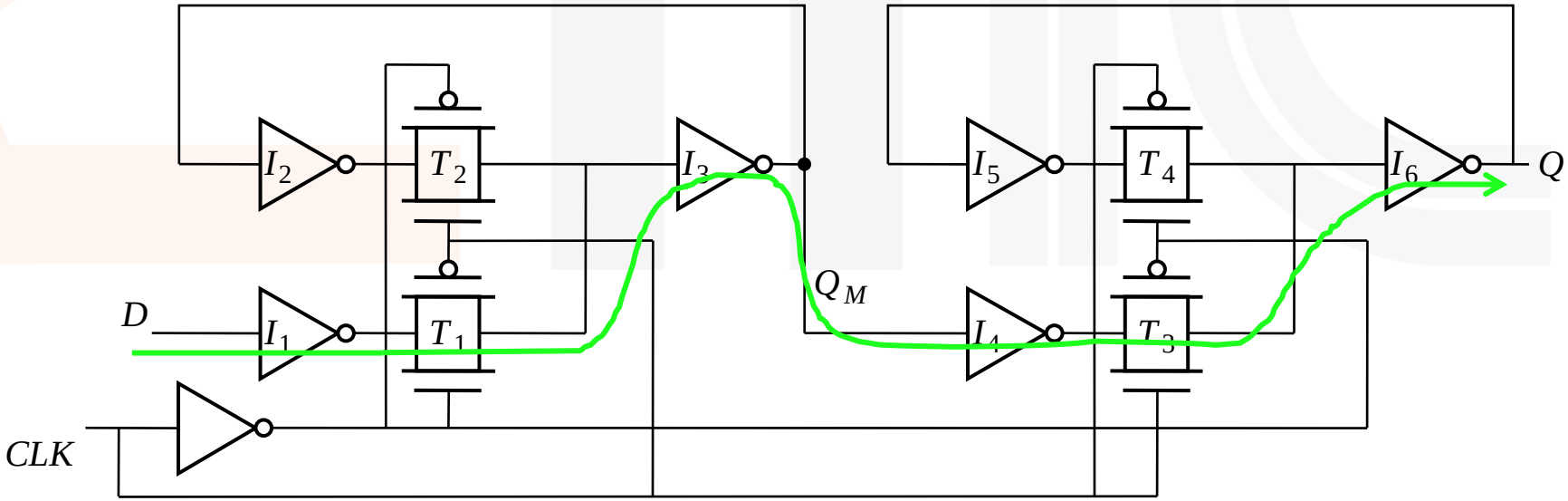


Latch

Other Flip Flop Implementations

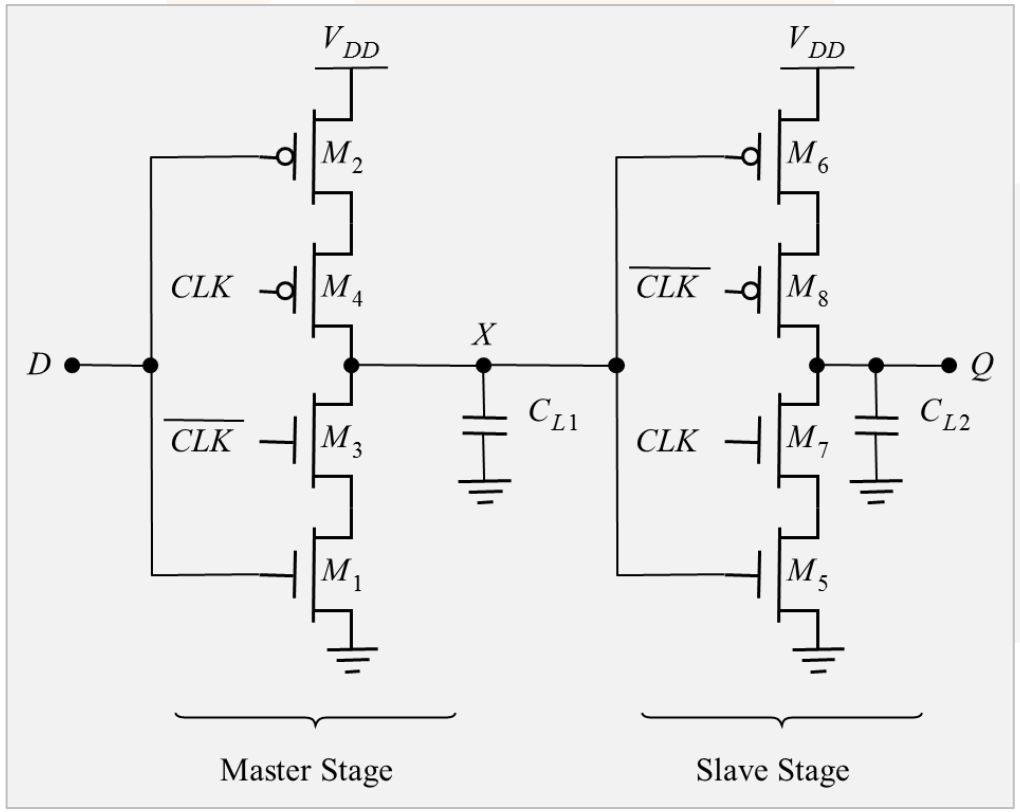
Problem – Clock Overlap



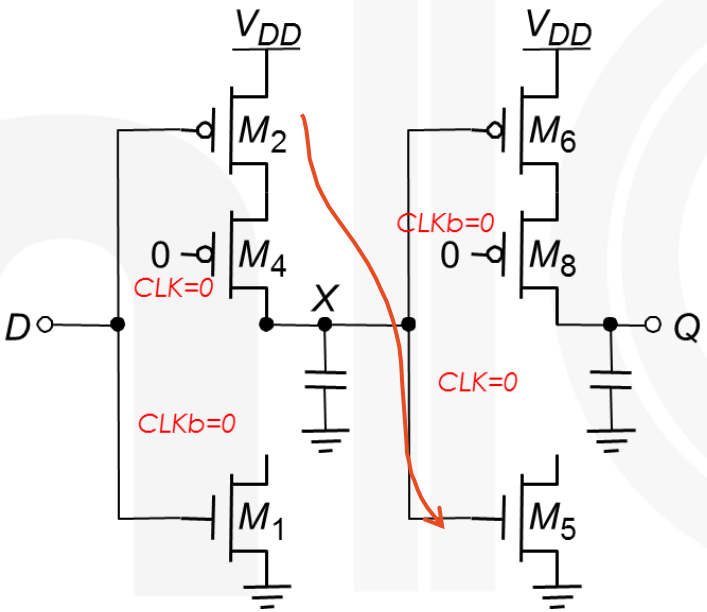


C²MOS – clocked CMOS

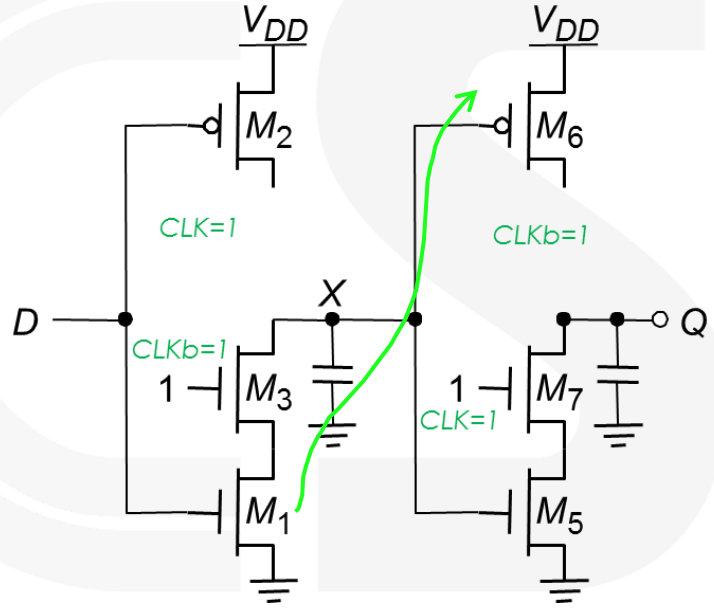
- **Insensitive to clock overlap.**

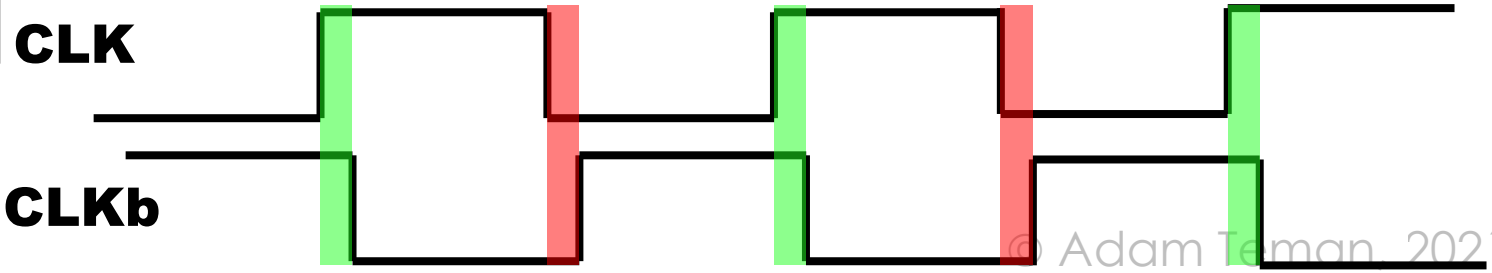


Low Phase Overlap

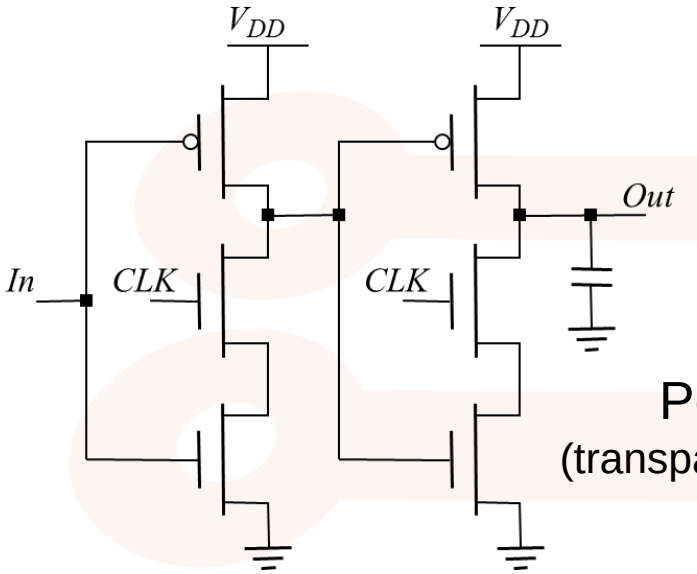


High Phase Overlap

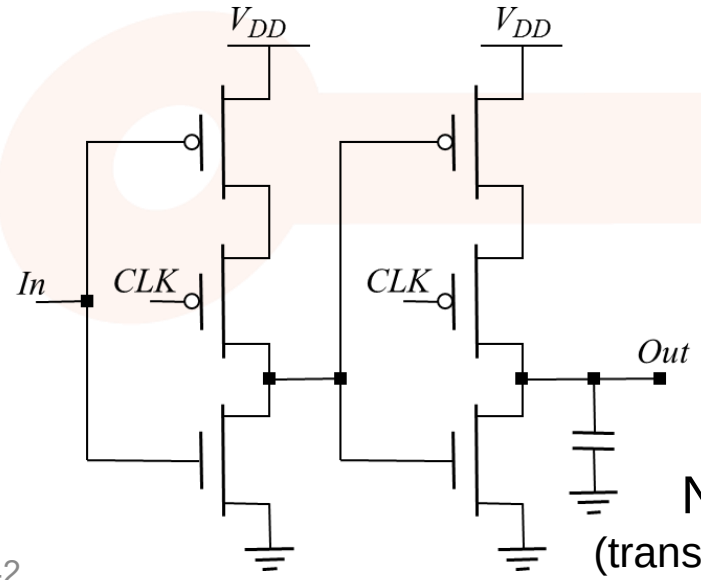




TSPC – True Single-Phase Clocked Register

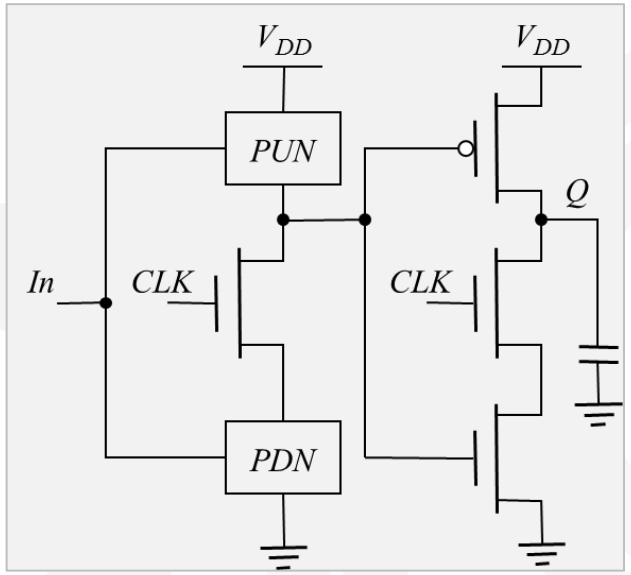


Positive latch
(transparent when CLK= 1)

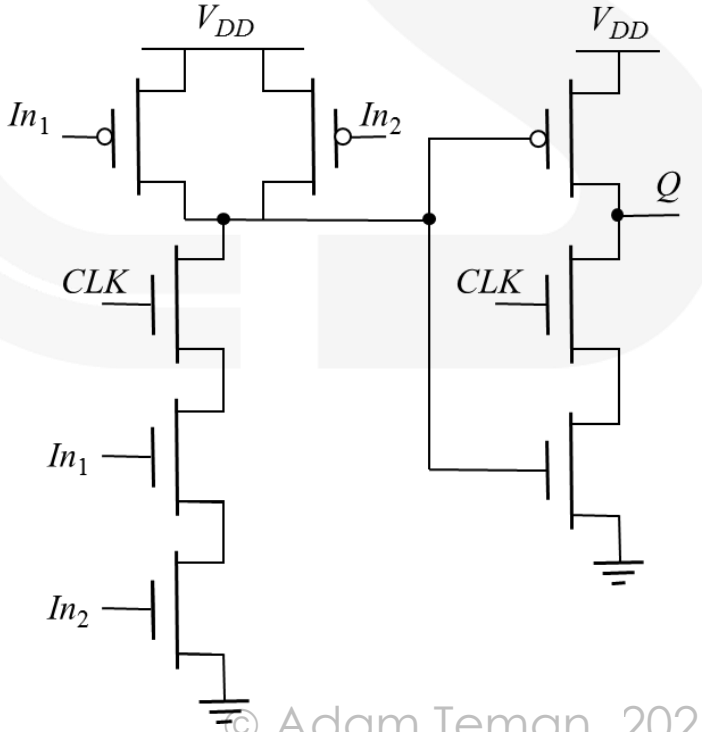


Negative latch
(transparent when CLK= 0)

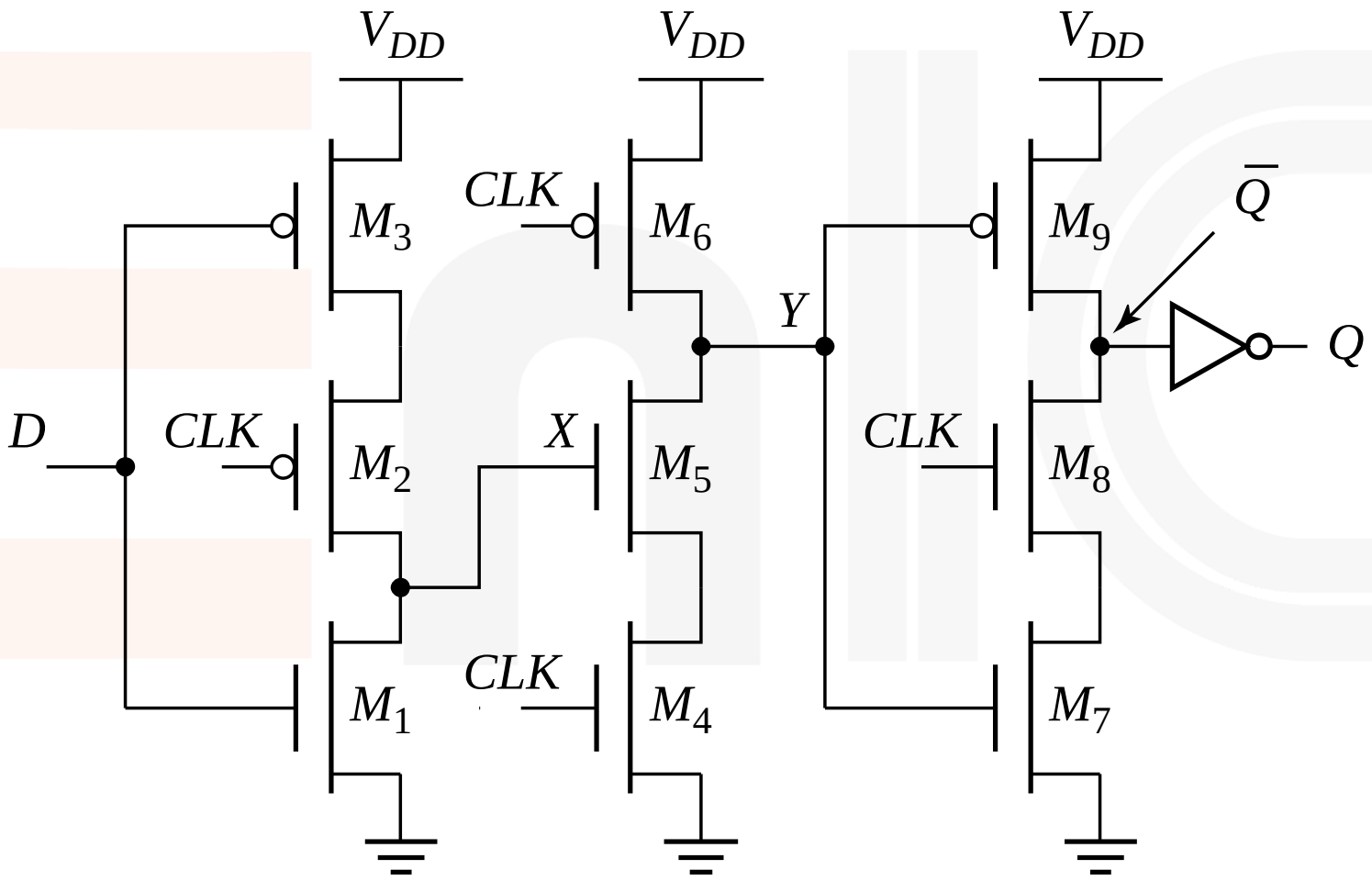
TSPC enables including
logic inside the latch!



Example:
AND latch



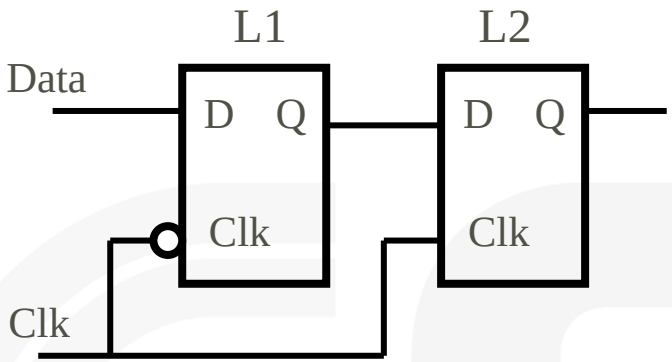
TSPC Flip Flop



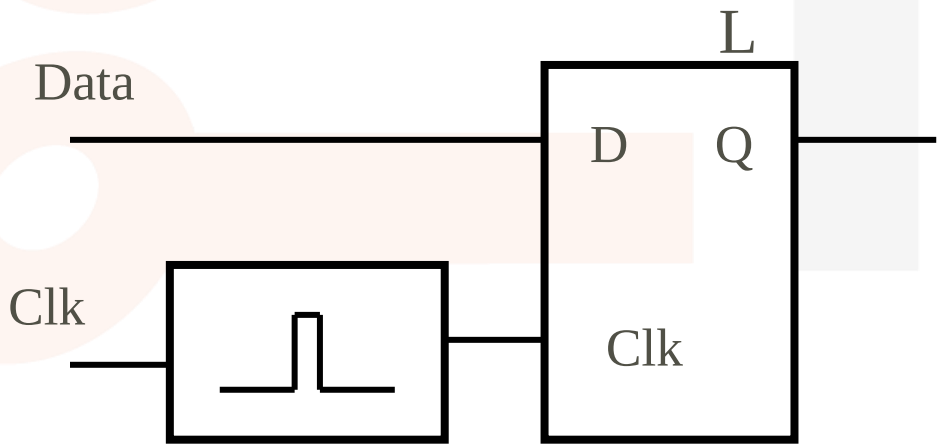
Pulse-Triggered Latches

- **Instead of a full set of master-slave latches**
- **We can *emulate* an edge with a short clock pulse:**

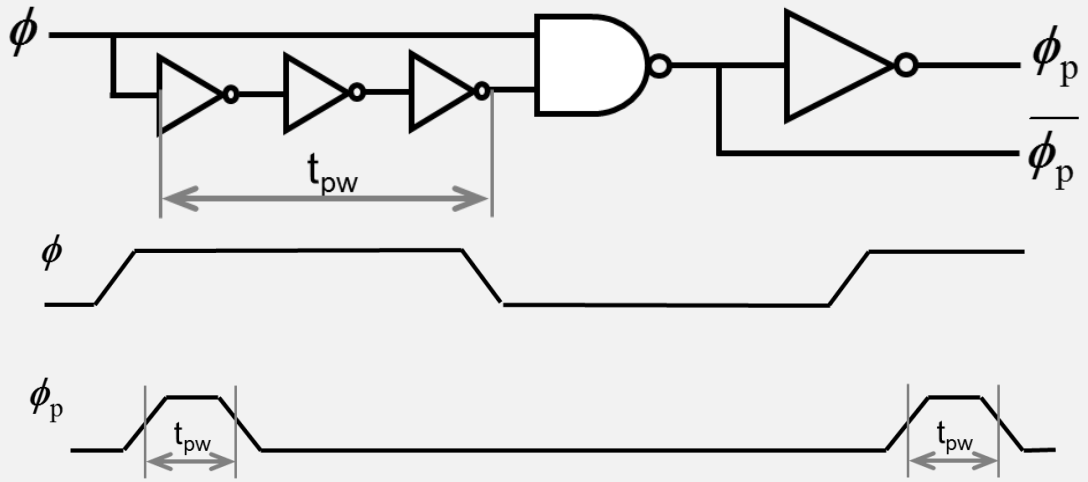
Master-Slave Latches



Pulse-Triggered Latch

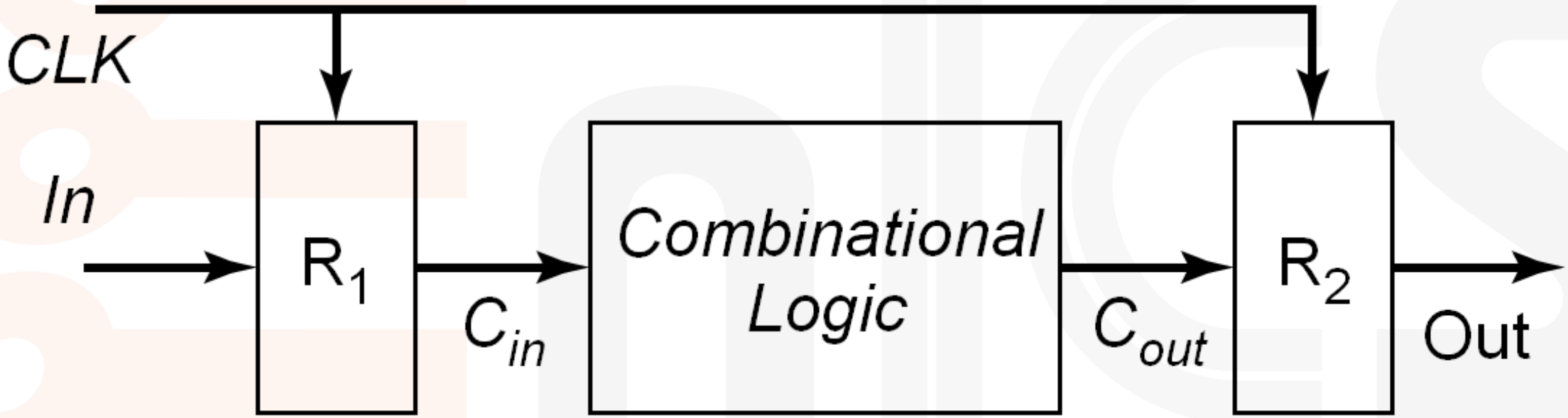


Design a clock pulse with a “clock chopper”



Basic Timing Constraints

Synchronous Timing

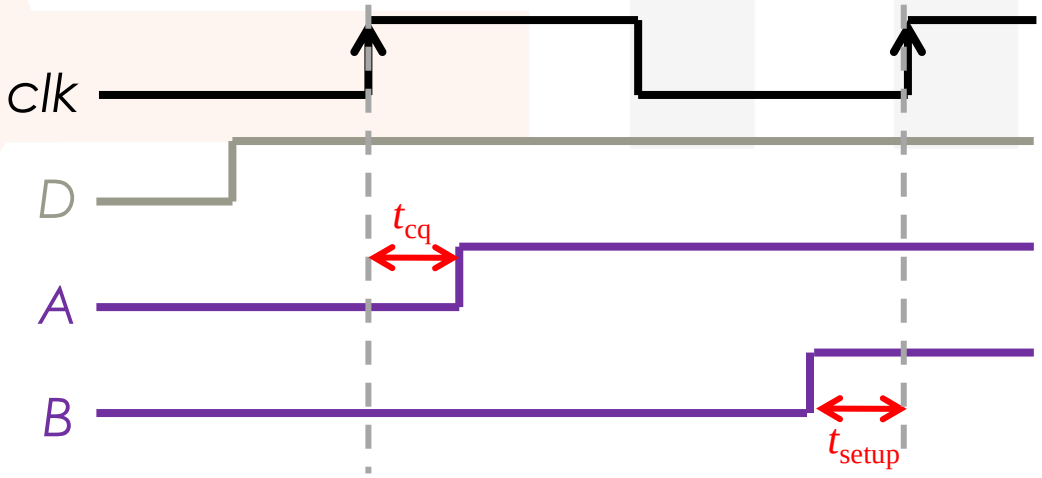


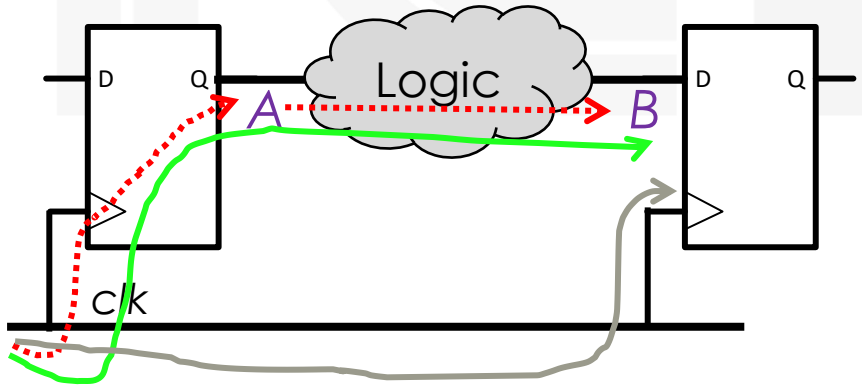
Timing Constraints

- **There are two main problems that can arise in synchronous logic:**
  - Max Delay: The data doesn't have enough time to pass from one register to the next before the next clock edge.
  - Min Delay: The data path is so short that it passes through several registers during the same clock cycle.
- **Max delay violations are a result of a slow data path, including the registers' t_{setup}, therefore it is often called the "Setup" path.**
- **Min delay violations are a result of a short data path, causing the data to change before the t_{hold} has passed, therefore it is often called the "Hold" path.**

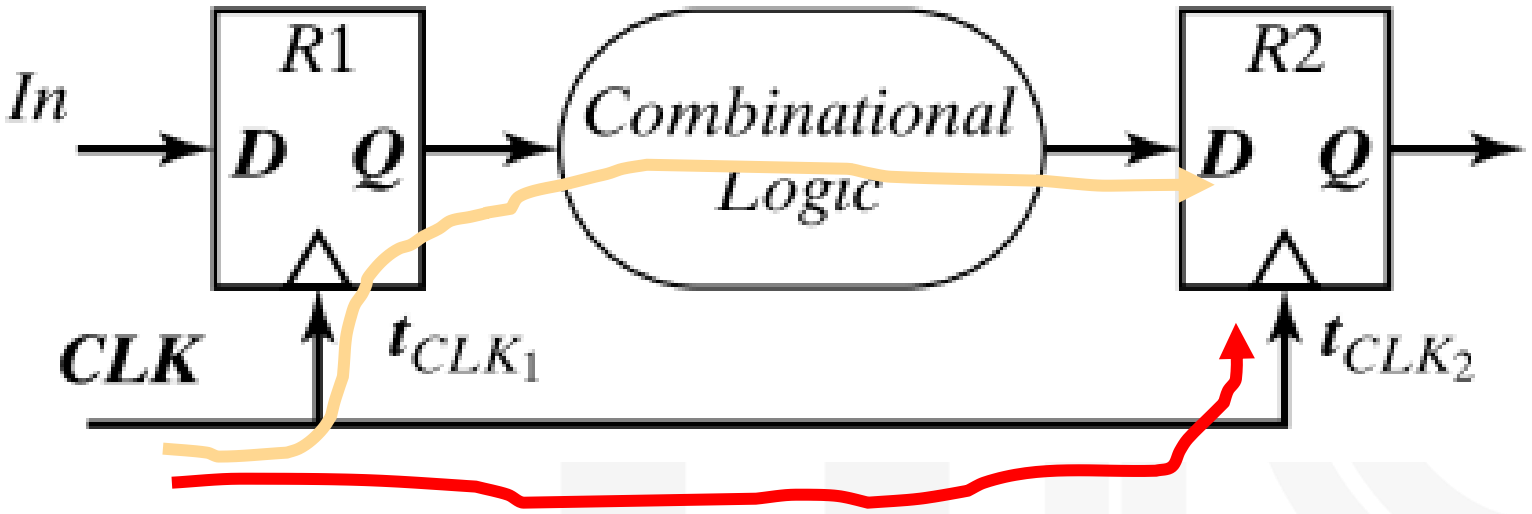
Setup (Max) Constraint

- **Let's see what makes up our clock cycle:**
  - After the clock rises, it takes t_{cq} for the data to propagate to point *A*.
  - Then the data goes through the delay of the logic to get to point *B*.
  - The data has to arrive at point *B*, t_{setup} before the next clock.
- **In general, our timing path is a race:**
  - Between the Data Arrival, starting with the *launching* clock edge.
  - And the Data Capture, one clock period later.





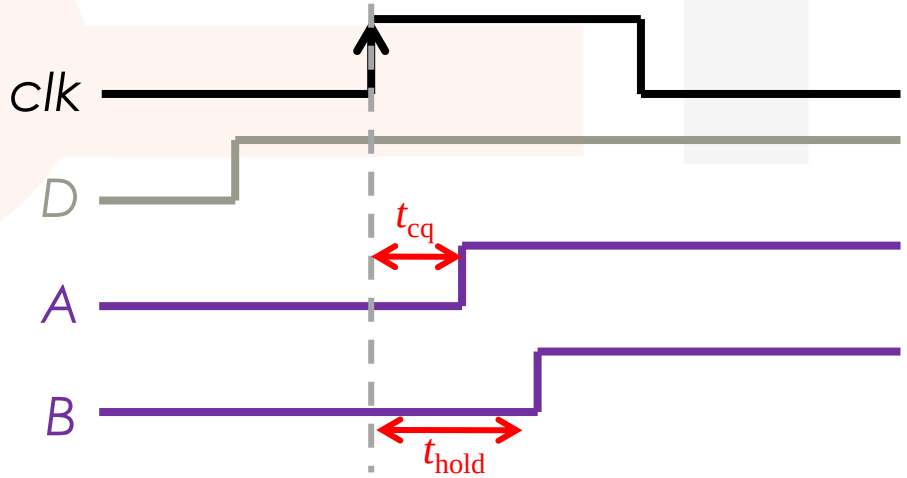
Setup (Max) Constraint

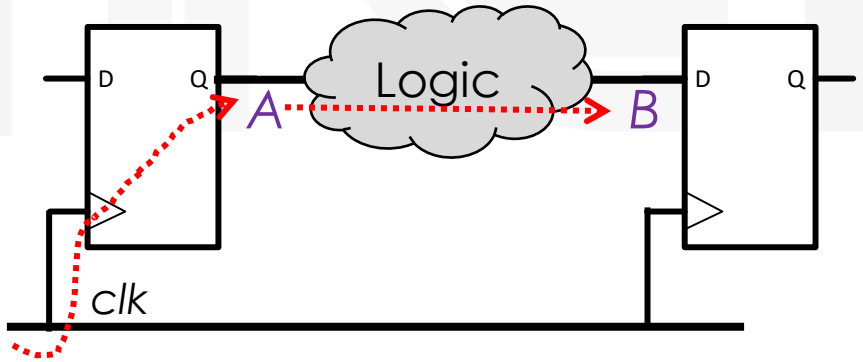


$$T > t_{cq} + t_{logic} + t_{setup}$$

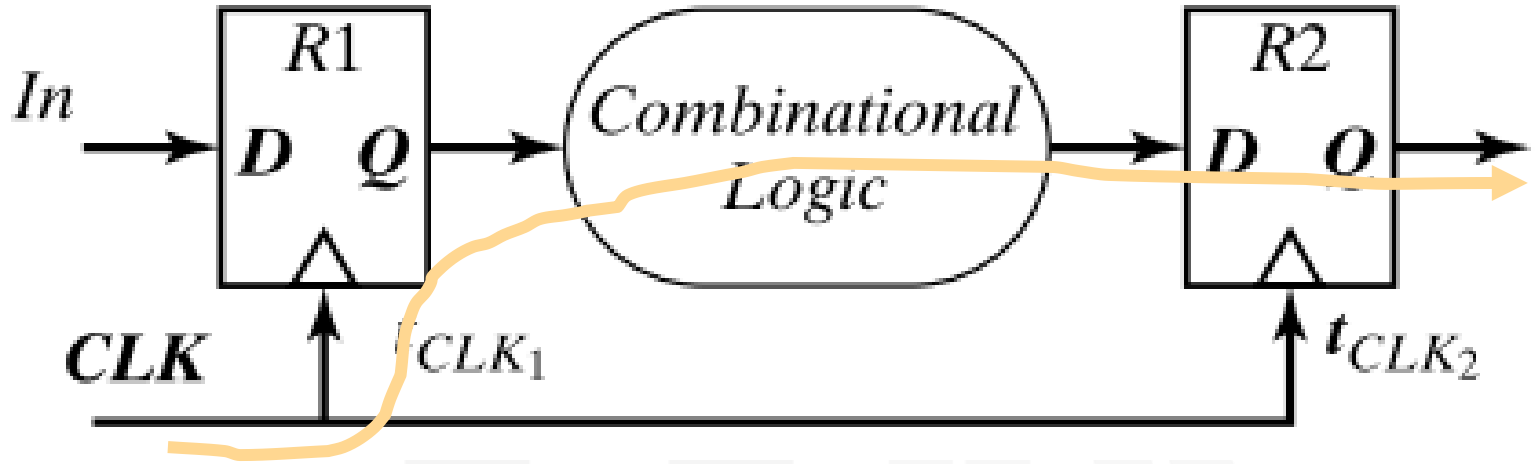
Hold (Min) Constraint

- **Hold problems occur due to the logic changing before t_{hold} has passed.**
- **This is not a function of cycle time – it is relative to a single clock edge!**
- **Let's see how this can happen:**
  - The clock rises and the data at **A** changes after t_{cq}.
  - The data at **B** changes t_{pd}(logic) later.
  - Since the data at **B** had to stay stable for t_{hold} after the clock (for the second register), the change at **B** has to be at least t_{hold} **after** the clock edge.





Hold (Min) Constraint



$$t_{cq} + t_{logic} > t_{hold}$$

Summary

- **For *Setup* constraints, the clock period has to be longer than the data path delay:**
  - This sets our maximum frequency.
  - If we have setup failures, we can always just **slow down the clock**.

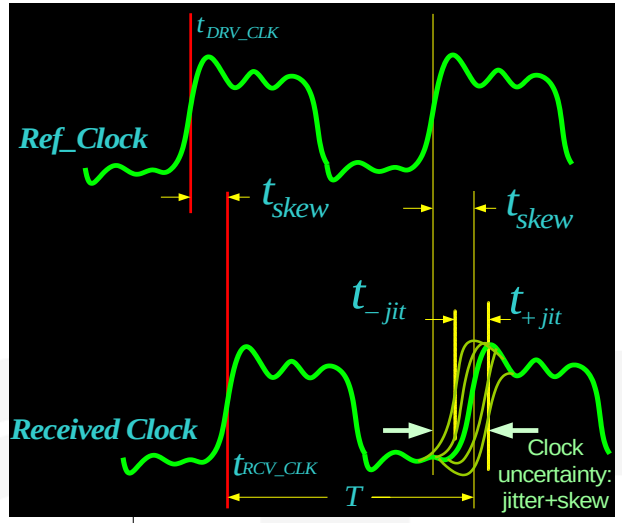
$$T > t_{cq} + t_{logic} + t_{setup}$$

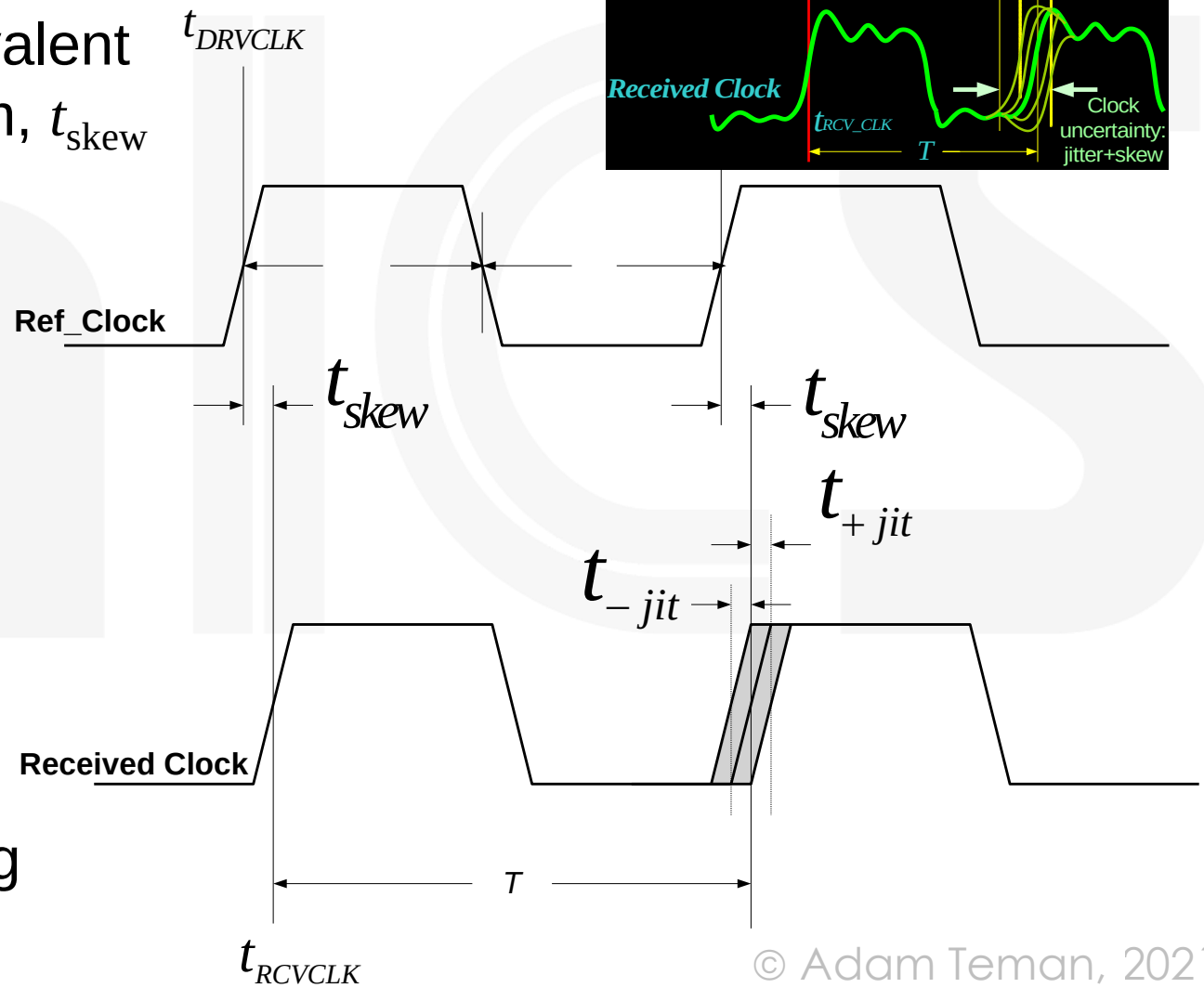
- **For *Hold* constrains, the data path delay has to be longer than the hold time:**
  - This is **independent of clock period**.
  - If there is a hold failure, you can throw your chip away!

$$t_{cq} + t_{logic} > t_{hold}$$

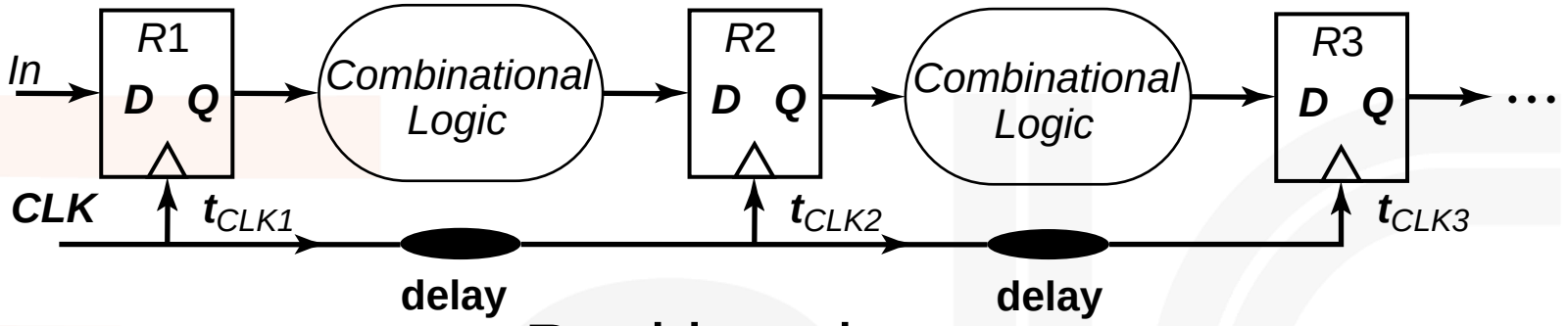
Clock Nonidealities

- **Clock skew**
  - Spatial variation in temporally equivalent clock edges; deterministic + random, t_{skew}
- **Clock jitter**
  - Temporal variations in consecutive edges of the clock signal; modulation + random noise
  - Cycle-to-cycle (short-term) $t_{\text{Jit},S}$
  - Long term $t_{\text{Jit},L}$
- **Variation of the pulse width**
  - Important for level sensitive clocking

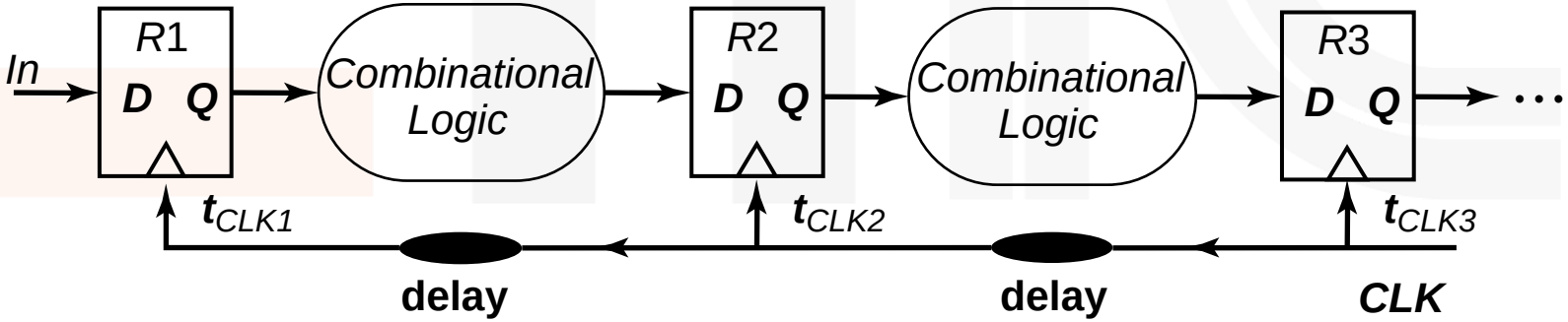




Positive and Negative Skew



Positive skew



Negative skew

Setup (Max) Constraint

- **The Launch path (still) consists of:**
  - $t_{cq} + t_{logic} + t_{setup}$
  - But if jitter makes the launch clock later, we need to add it to the data path delay.
- **The Capture path consists of:**
  - The clock period (T)
  - Positive skew means the capture clock path is longer.
  - If jitter makes the capture clock earlier, we need to subtract it.
- **Our max constraint is:**
- **So we get:**

$$t_{launch} = t_{cq} + t_{logic} + t_{setup} + t_{jitter}$$

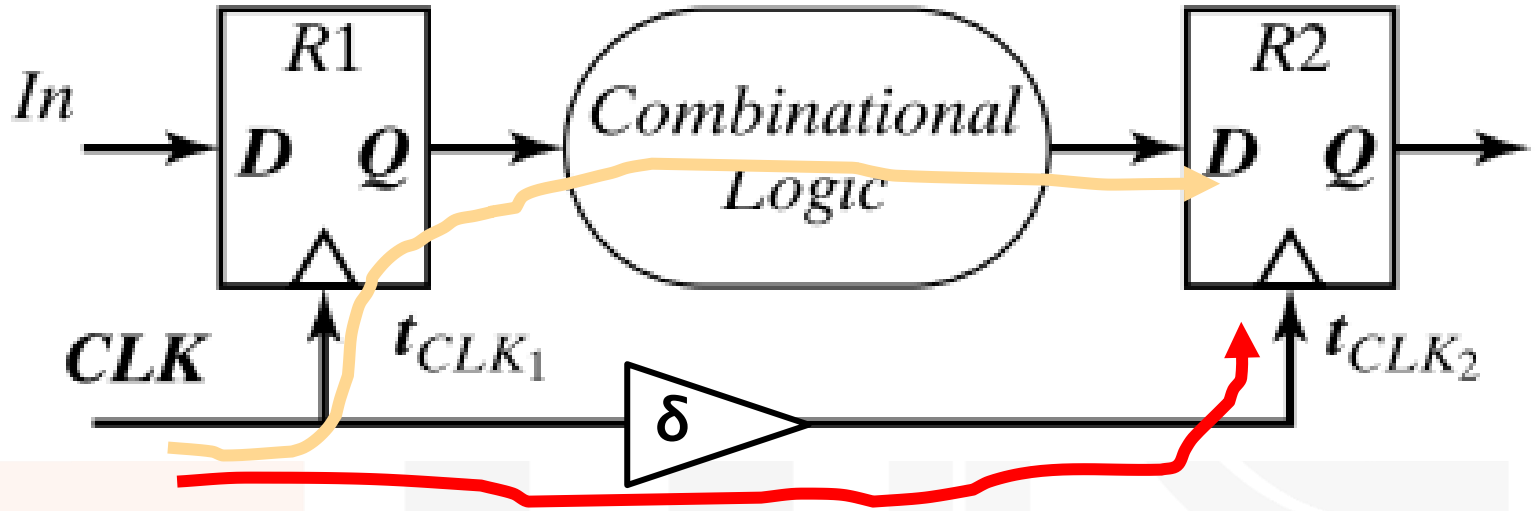
$$t_{capture} = T + \delta_{skew} - t_{jitter}$$

$$t_{capture} \geq t_{launch}$$

$$T \geq t_{cq} + t_{logic} + t_{setup} + 2t_{jitter} - \delta_{skew}$$

Setup (Max) Constraint

- **Data has to arrive before *next* clock edge.**



$$T + \delta_{\text{skew}} > t_{\text{cq}} + t_{\text{logic}} + t_{\text{setup}} + 2t_{\text{jitter}}$$

Hold (Min) Constraint

- **The Launch path (still) consists of:**
  - $t_{cq} + t_{logic}$
  - But if jitter makes the launch clock *later*, we need to subtract it from the data path delay.

$$t_{launch} = t_{cq} + t_{logic} - t_{jitter}$$

- **The Capture path consists of:**
  - Skew that makes the clock edge arrive at the capture register later than at the launch register.
  - Actually, since it is a single clock edge, jitter should effect the capture clock the same as the launch clock.
  - But as a worst case, we will add it as spatial jitter.

$$t_{capture} = \delta_{skew} + t_{jitter} + t_{hold}$$

- **Our min constraint is:**

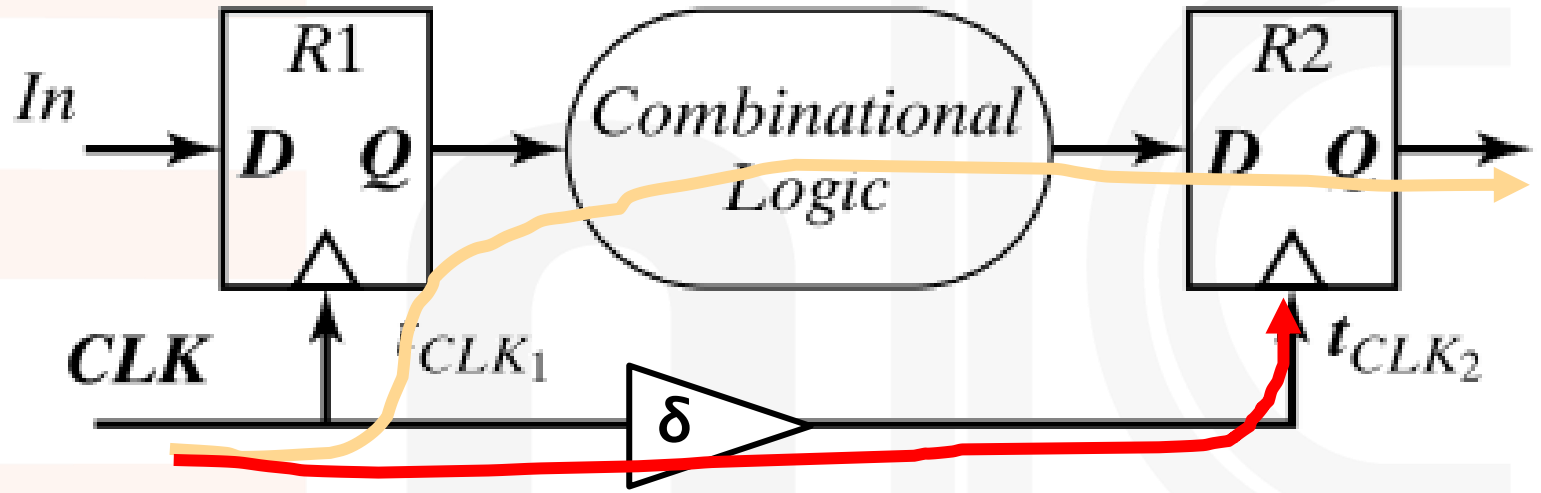
$$t_{launch} \geq t_{capture}$$

- **So we get:**

$$t_{cq} + t_{logic} \geq \delta_{skew} + t_{hold} + 2t_{jitter}$$

Hold (Min) Constraint

- **Data has to arrive *after* the same clock edge has arrived at capture reg.**



$$t_{cq} + t_{logic} > t_{hold} + \delta_{skew} + 2t_{jitter}$$

Adding in Variation

- **As previously discussed, variations in both fabrication and operating conditions occur and are taken into account through "corner" simulation.**
- **For global variation we have defined three primary simulation corners:**
  - Typical Corner: our gates operate under ***nominal*** conditions and variation.
  - Slow Corner: our gates ***slower*** (i.e., high VT, high temperature, low voltage).
  - Fast Corner: our gates ***faster*** (i.e., low VT, low temperature, high voltage).
- **To assume worst-case conditions:**
  - Calculate ***max-delay*** with the ***slowest*** possible transitions →Slow Corner.
  - Calculate ***min -delay*** with the ***fastest*** possible transitions →Fast Corner.

The Computer Hall of Fame

- **The machine that made IBM dominate the computer industry for 20 years**

IBM System/360

Source: techradar.com

- Announced April 7th, 1964, the first “upgradable” and fully “compatible” computer.
- Thomas Watson “bet the business” on this machine with a $5B investment that was to cannibalize all of IBM’s existing computers.
- Ranked as one of the all-time top 3 business accomplishments alongside Ford’s Model T and the Boeing 707.
- The machine that pioneered the 8-bit byte and the peripheral components made by third parties.

Source:computerhistory.org

Static Timing Analysis Example

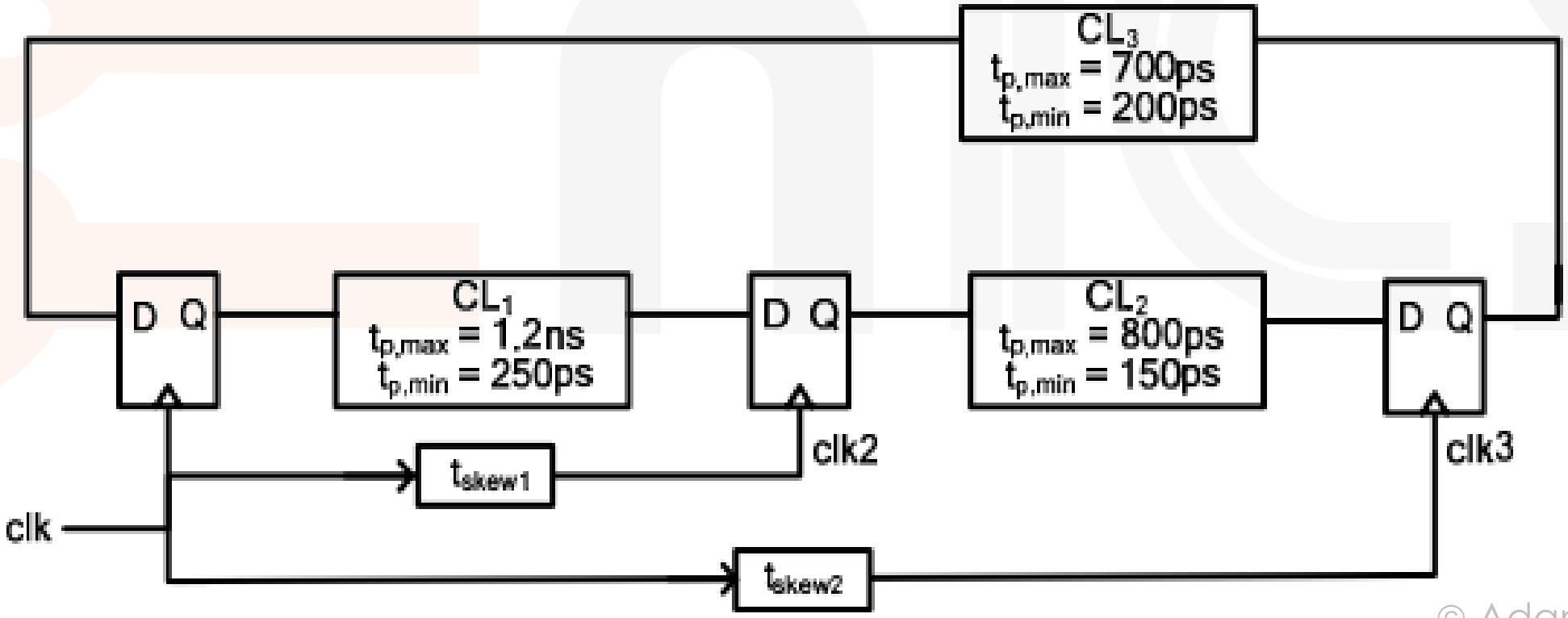
STA Example

- **We are given a synchronous network with:**

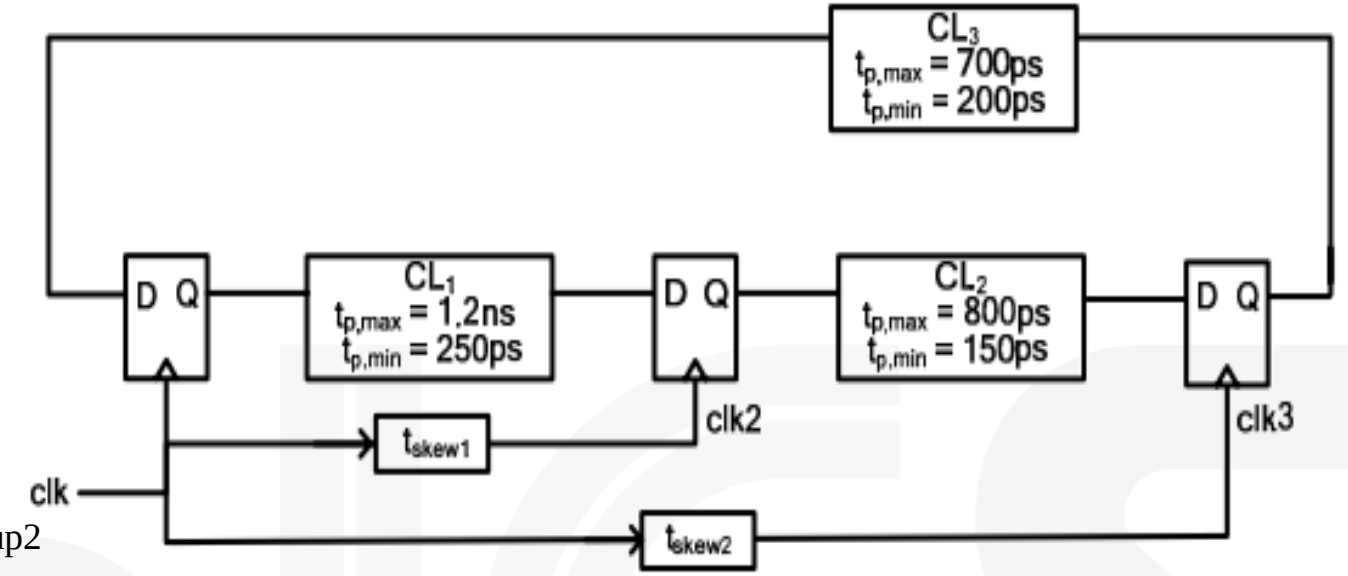
$$t_{CQ} = 150\text{ps}, t_{setup} = 50\text{ps}, \; t_{hold} = 100\text{ps}, \; t_{jitter} = 0$$

- **In addition:**

$$t_{skew1} = -100\text{ps}, \quad t_{skew2} = 50\text{ps}$$



STA Example



- **We'll find the setup constraints for each path:**

Path 1: $T_1 + t_{skew1} > t_{cq1} + t_{p,max}(CL_1) + t_{setup2}$

$$T_1 > 150p + 1.2n + 50p + 100p = 1500p = 1.5ns \rightarrow 666MHz$$

Path 2: $T_2 + (t_{skew2} - t_{skew1}) > t_{cq2} + t_{p,max}(CL_2) + t_{setup3}$

$$T_2 > 150p + 800p + 50p - 150p = 850p \rightarrow 1.17GHz$$

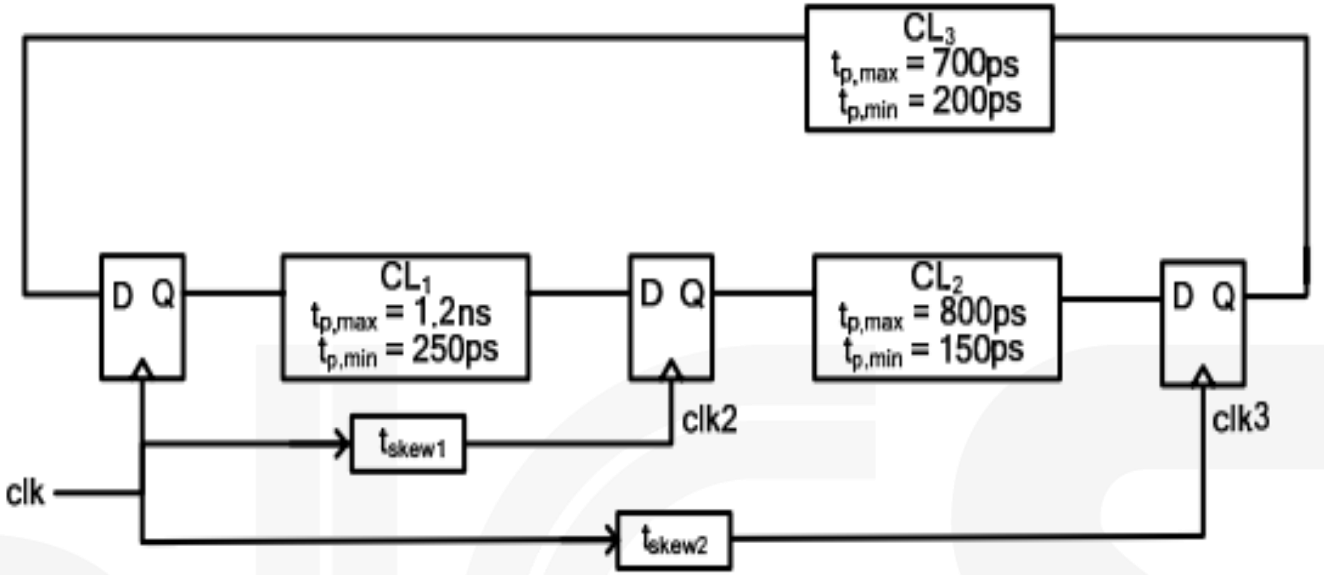
Path 3: $T_3 + (0 - t_{skew2}) > t_{cq3} + t_{p,max}(CL_3) + t_{setup1}$

$$T_3 > 150p + 700p + 50p + 50p = 950p \rightarrow 1.05GHz$$

- ***So the critical path is Path 1 and the maximum frequency is 666MHz.***

STA Example

- **Now, we'll find the hold constraints for t_{skew1} and t_{skew2}:**



Path 1: $t_{skew1} + t_{hold2} < t_{cq1} + t_{p,min}(CL_1)$

$$t_{skew1} < 150p + 250p - 100p = 300p$$

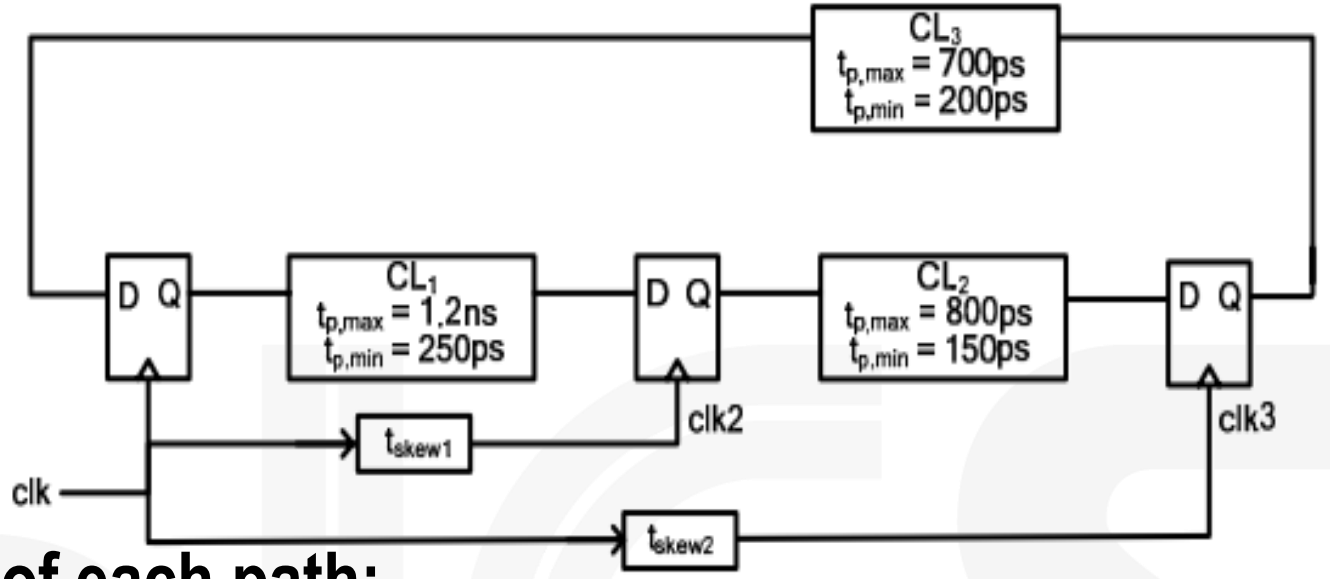
Path 2: $(t_{skew2} - t_{skew1}) + t_{hold3} < t_{cq2} + t_{p,min}(CL_2)$

$$(t_{skew2} - t_{skew1}) < 150p + 150p - 100p = 200p$$

Path 3: $(0 - t_{skew2}) + t_{hold1} < t_{cq3} + t_{p,min}(CL_3)$

$$-t_{skew2} < 150p + 200p - 100p = 250p \rightarrow t_{skew2} > -250p$$

STA Example



- If we could set t_{skew1} and t_{skew2}, could we use them to maximize our frequency?
- If we could equally divide the delay of each path:

$$t_{total} = 3 \cdot t_{cq} + t_{p,max}(CL_1 + CL_2 + CL_3) + 3 \cdot t_{setup} =$$

$$= 450p + 2.7n + 150p = 3.3ns$$

- So to get the max frequency, set all delays to 1.1ns:

$$1.1ns + t_{skew1} = 150p + 1.2n + 50p \rightarrow t_{skew1} = 300ps$$

$$1.1ns + (t_{skew2} - t_{skew1}) = 150p + 800p + 50p \rightarrow t_{skew2} = 200ps$$

$$1.1ns + (0 - t_{skew2}) = 150p + 700p + 50p \rightarrow t_{skew2} = 200ps$$

Further Reading

- J. Rabaey, "*Digital Integrated Circuits*" 2003, Chapter 7
- Weste, Harris "CMOS VLSI Design" Chapter 7
- E. Alon, Berkeley *EE-141*, Lectures 23,24 (Fall 2010) http://bwrc.eecs.berkeley.edu/classes/icdesign/ee141_f10/
- Berkeley *CS-150*, Lecture 4 http://inst.eecs.berkeley.edu/~cs150/
- Oklobdzija, Stojanovic, Markovic, Nedovic, "*Digital System Clocking*"