Digital VLSI Design

Lecture 3: Timing Analysis

Semester A, 2016-17

Lecturer: Dr. Adam Teman

20 November 2016

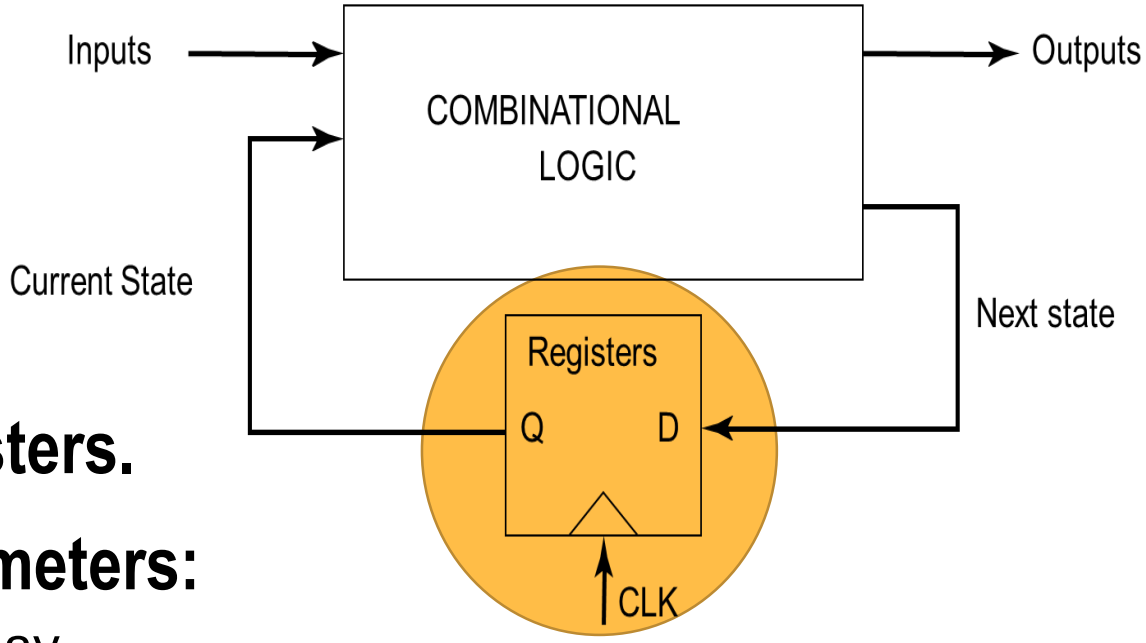
Emerging Nanoscaled Integrated Circuits and Systems Labs

Bar-Ilan University
אוניברסיטת בר-אילן

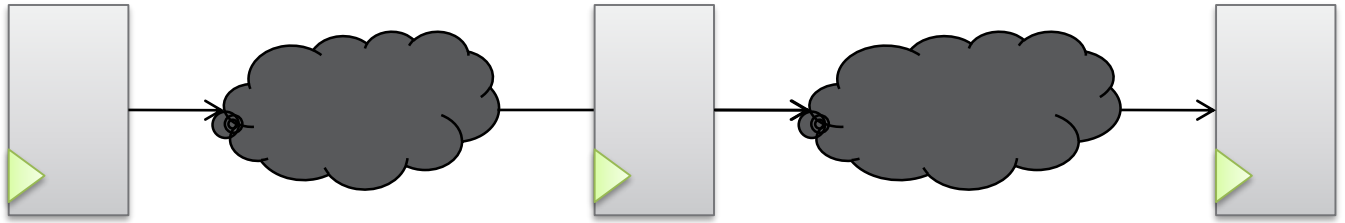
Sequential Clocking

Synchronous Design - Reminder

- **The majority of digital designs are *Synchronous* and constructed with *Sequential Elements*.**
  - Synchronous design eliminates races (like a traffic light).
  - Pipelining increases throughput.

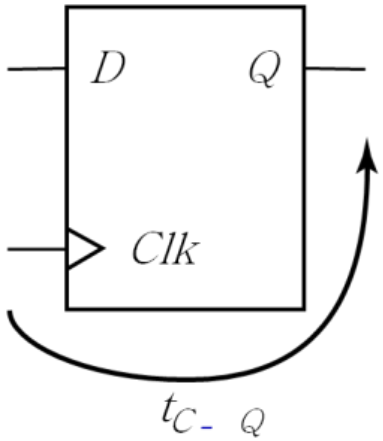


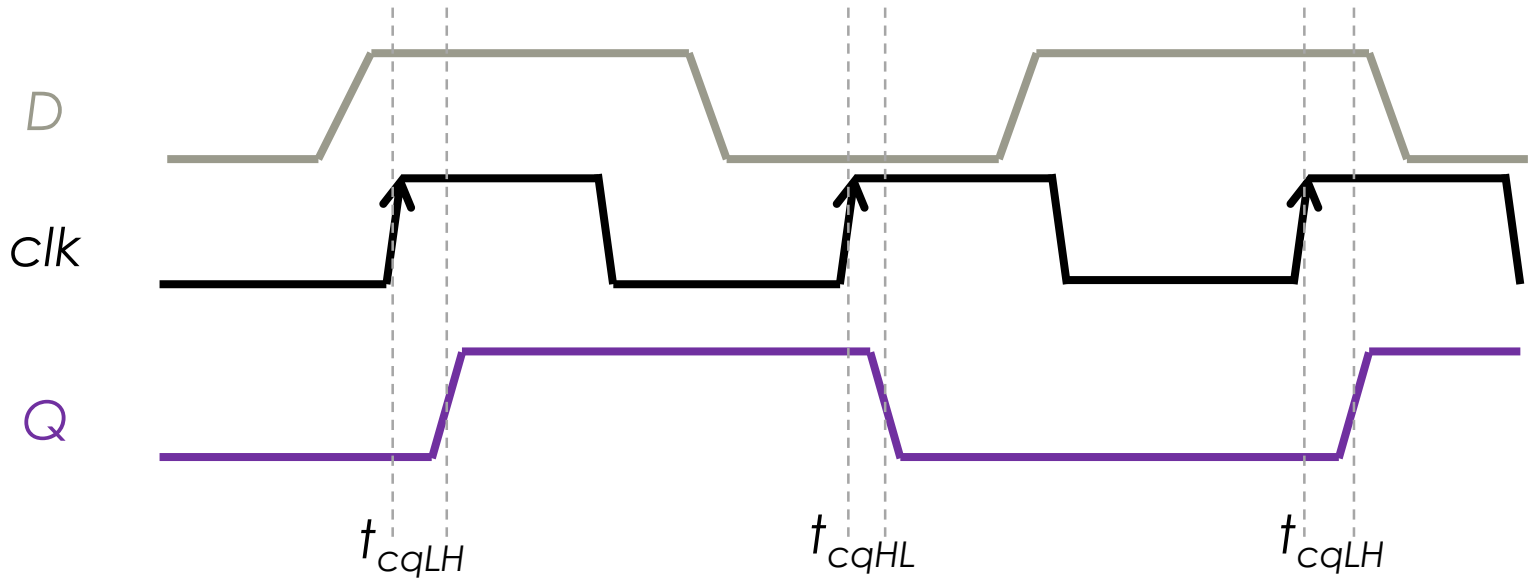
- **We will assume that all *sequentials* are *Edge-Triggered*, using D-Flip Flops as registers.**
- **D-Flip Flops have three critical timing parameters:**
  - t_{cq} – clock to output: essentially a propagation delay
  - t_{setup} – setup time: the time the data needs to arrive before the clock
  - t_{hold} – hold time: the time the data has to be stable after the clock



Timing Parameters - t_{cq}

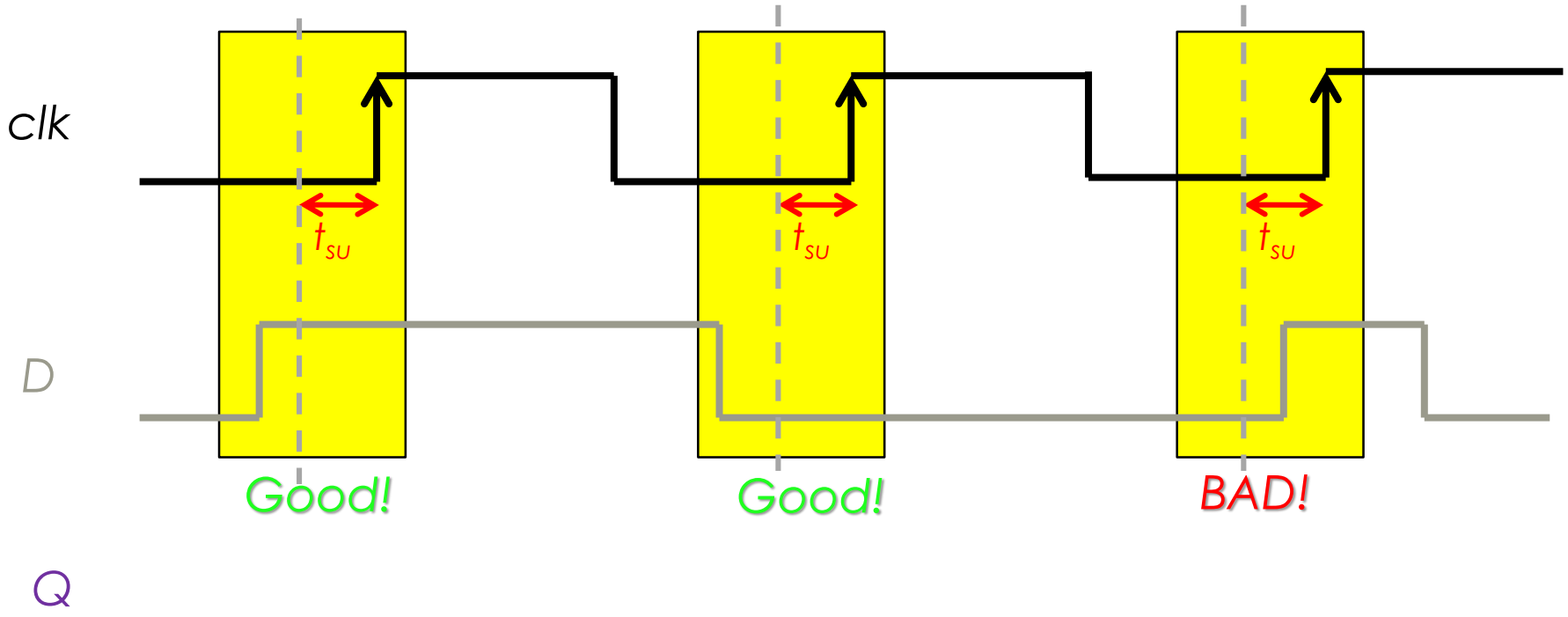
- t_{cq} is the time from the clock edge until the data appears at the output.
- The t_{cq} for rising and falling outputs is different.





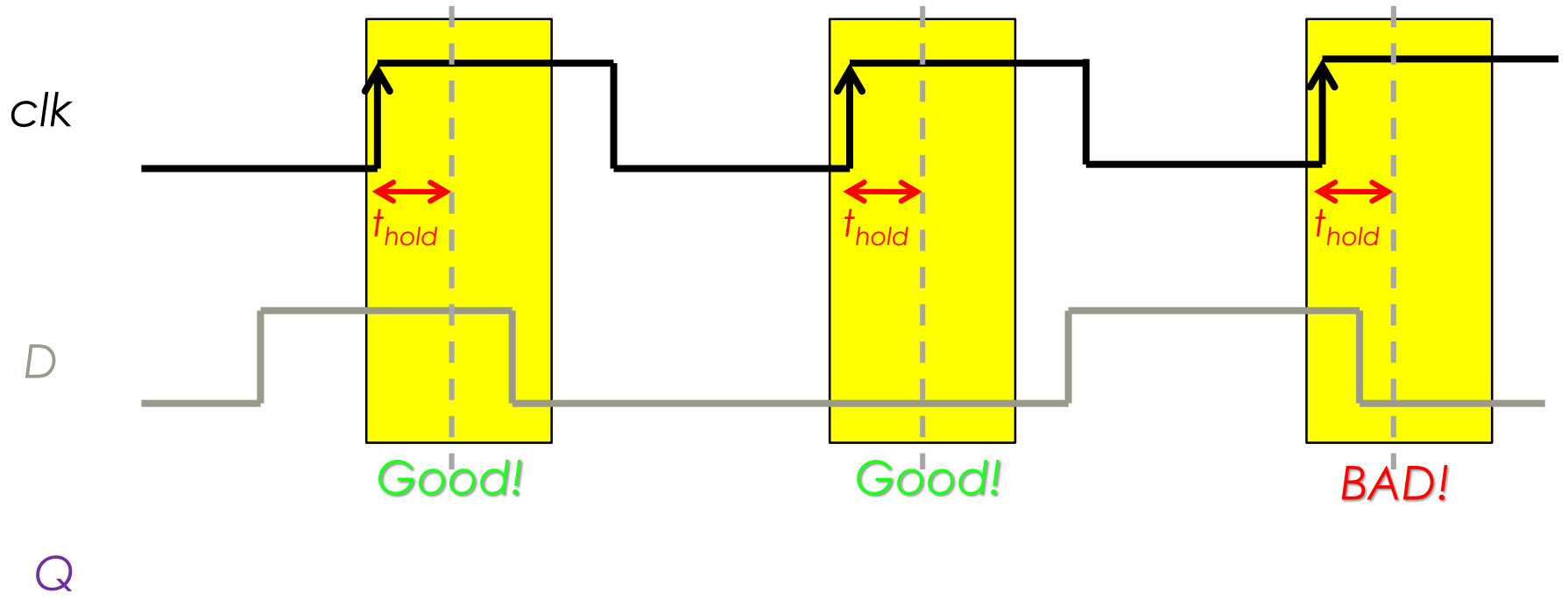
Timing Parameters - t_{setup}

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



Timing Parameters - t_{hold}

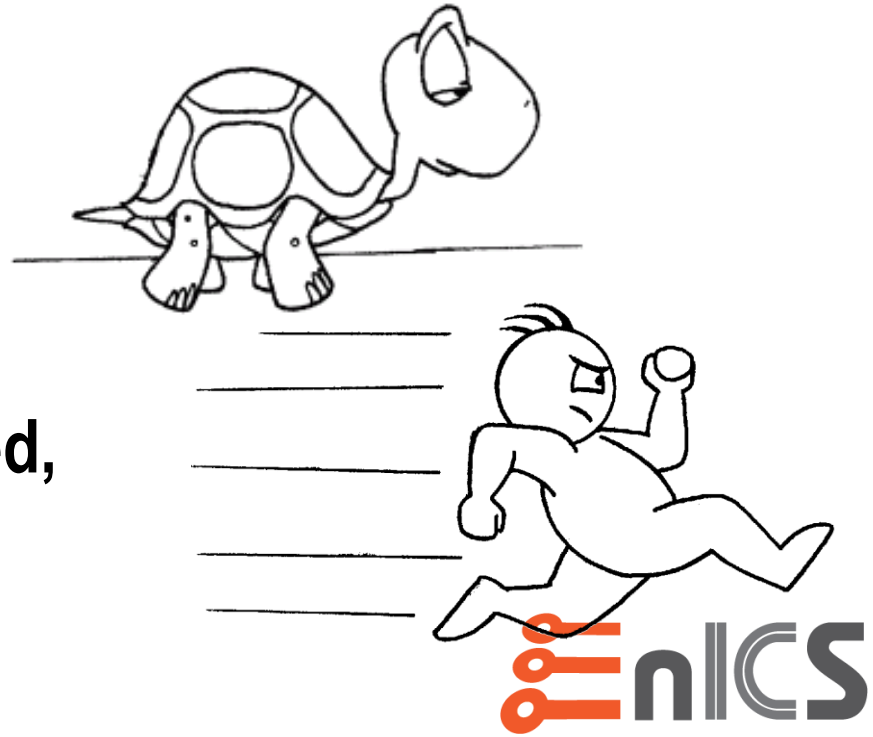
- t_{hold} - Hold time is the time the data has to be stable *after the clock* to ensure correct sampling.



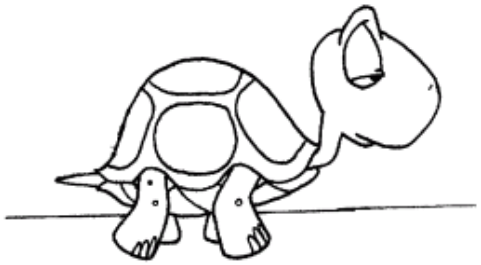
Timing Constraints

- **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***.

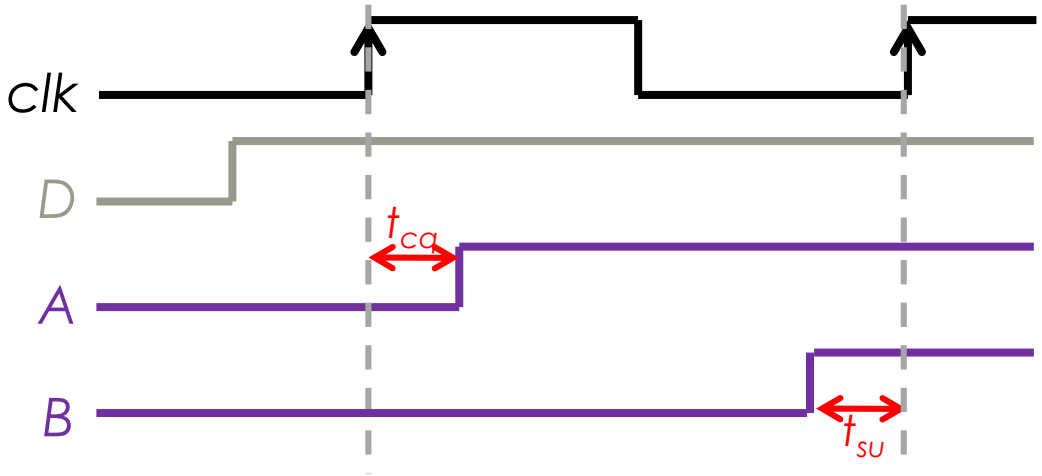
- **Max delay violations are a result of a slow data path, including the registers' t_{su}, therefore it is often called the "Setup" path.**
- **Min delay violations are a result of a short data path, causing the data to change before the t_{hold} has passed, therefore it is often called the "Hold" path.**

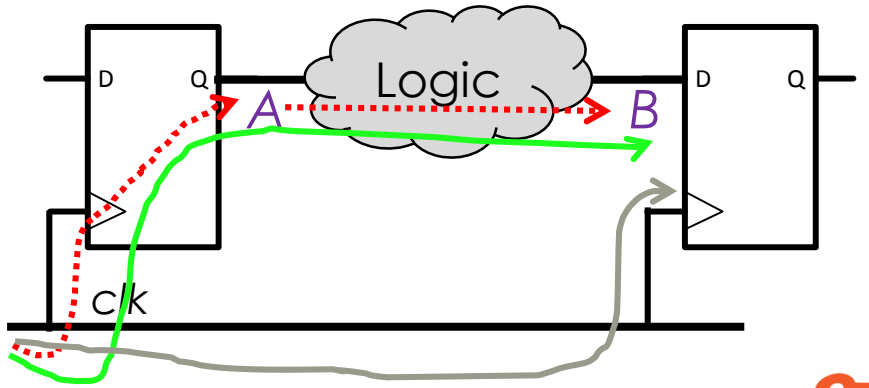


Setup (Max) Constraint

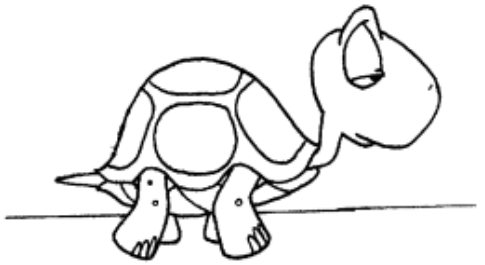


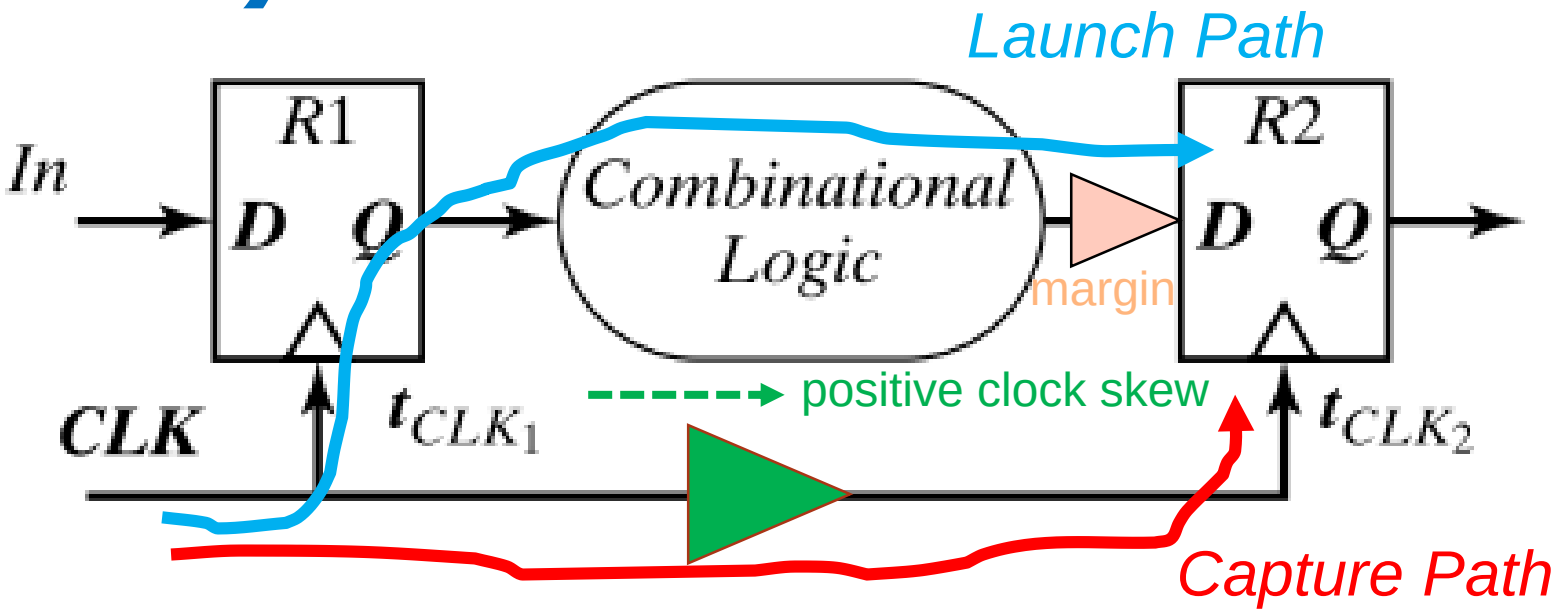
- **Let's see what makes up our clock cycle:**
  - After the clock rises, it takes t_{cq} for the data to propagate to point *A*.
  - Then the data goes through the delay of the logic to get to point *B*.
  - The data has to arrive at point *B*, t_{su} before the next clock.
- **In general, our timing 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



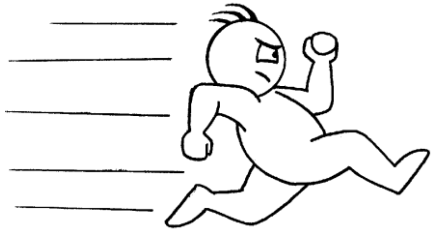


$$T > t_{CQ} + t_{\text{logic}} + t_{SU}$$

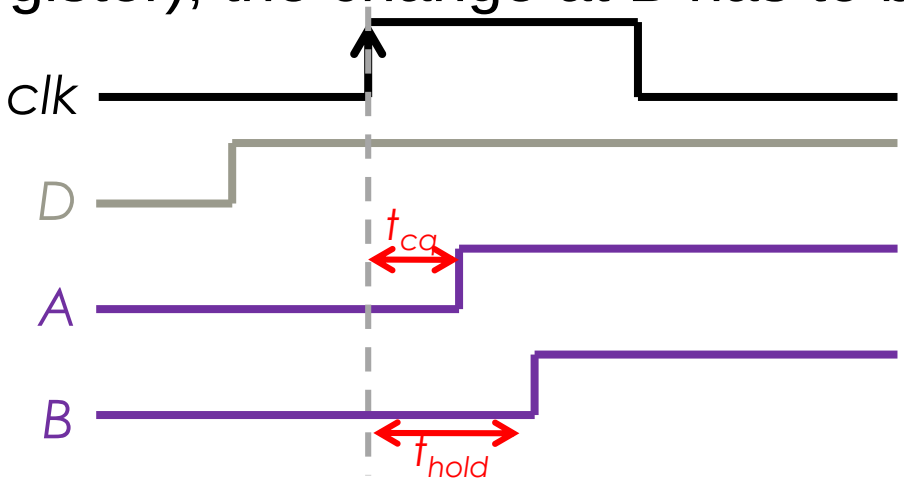
Adding in clock skew and other guardbands:

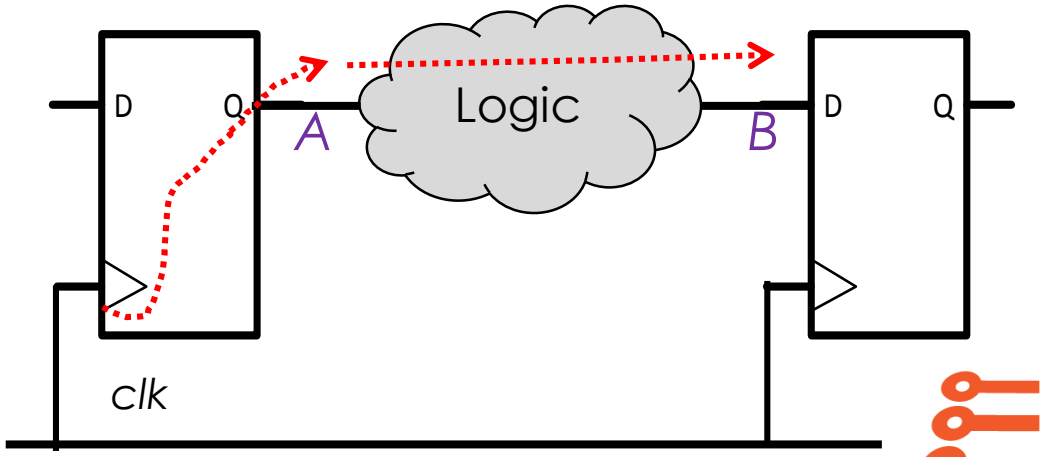
$$T + \delta_{\text{skew}} > t_{CQ} + t_{\text{logic}} + t_{SU} + \delta_{\text{margin}}$$

Hold (Min) Constraint

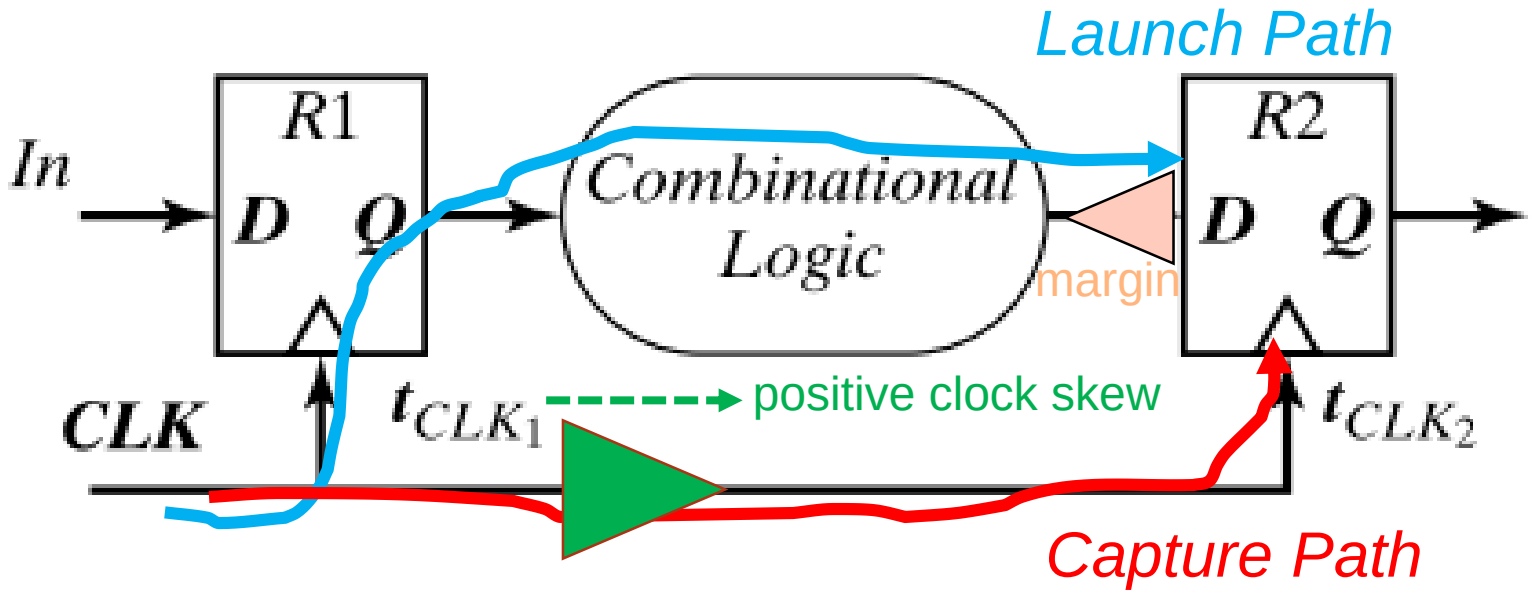


- **Hold problems occur due to the logic changing before t_{hold} has passed.**
- **This is not a function of cycle time – it is relative to a single clock edge!**
- **Let's see how this can happen:**
  - The clock rises and the data at A changes after t_{cq}.
  - The data at B changes $t_{pd}(logic)$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

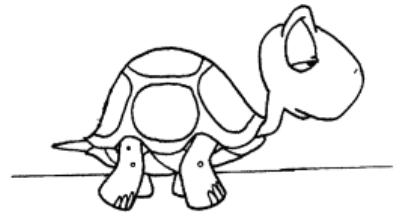


$$t_{CQ} + t_{\text{logic}} > t_{hold}$$

Adding in clock skew and other guardbands:

$$t_{CQ} + t_{\text{logic}} - \delta_{\text{margin}} > t_{hold} + \delta_{\text{skew}}$$

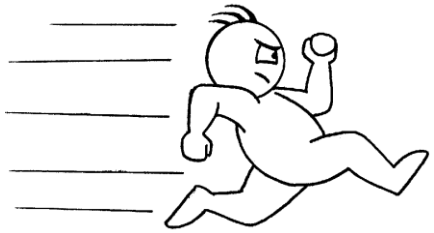
Summary



- **For *Setup* constraints, the data has to propagate fast enough to be captured by the next clock edge:**
  - This sets our maximum frequency.
  - If we have setup failures, we can always just **slow down the clock**.

$$t_{\text{launch}} < T + t_{\text{capture}}$$

$$T + \delta_{\text{skew}} > t_{CQ} + t_{\text{logic}} + t_{SU} + \delta_{\text{margin}}$$



- **For *Hold* constrains, the data path delay has to be long enough so it isn't accidentally captured by the same clock edge:**
  - This is **independent of clock period**.
  - If there is a hold failure, you can throw your chip away!

$$t_{\text{launch}} > t_{\text{capture}}$$

$$t_{CQ} + t_{\text{logic}} - \delta_{\text{margin}} > t_{hold} + \delta_{\text{skew}}$$

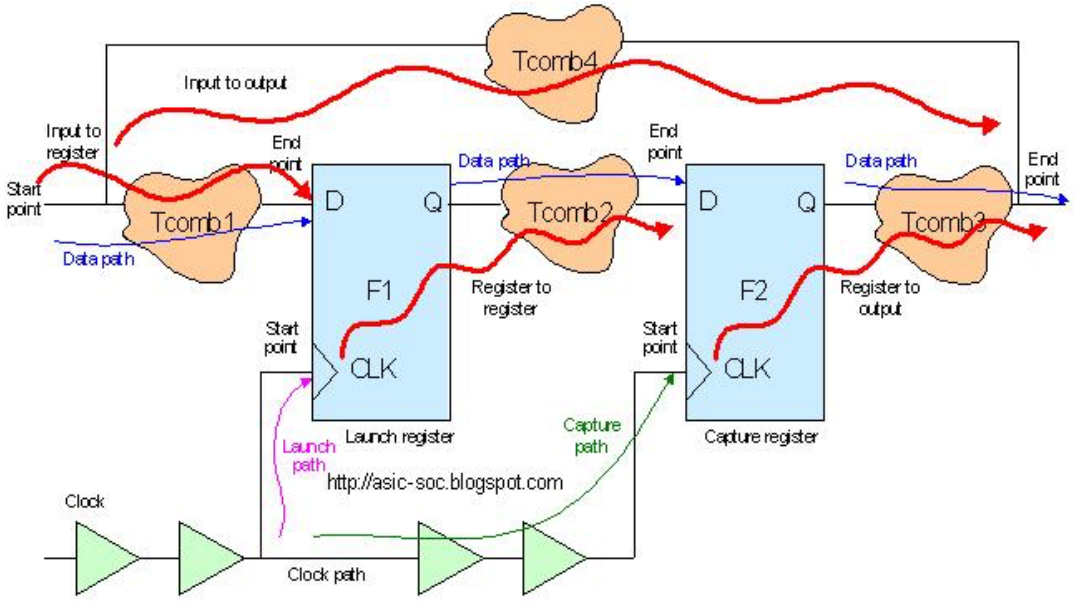
Static Timing Analysis

Or why and how to calculate slack.

This section is heavily based on Rob Rutenbar's "From Logic to Layout", Lecture 12 from 2013. For a better ☺ and more detailed explanation, do yourself a favor and go see the original!

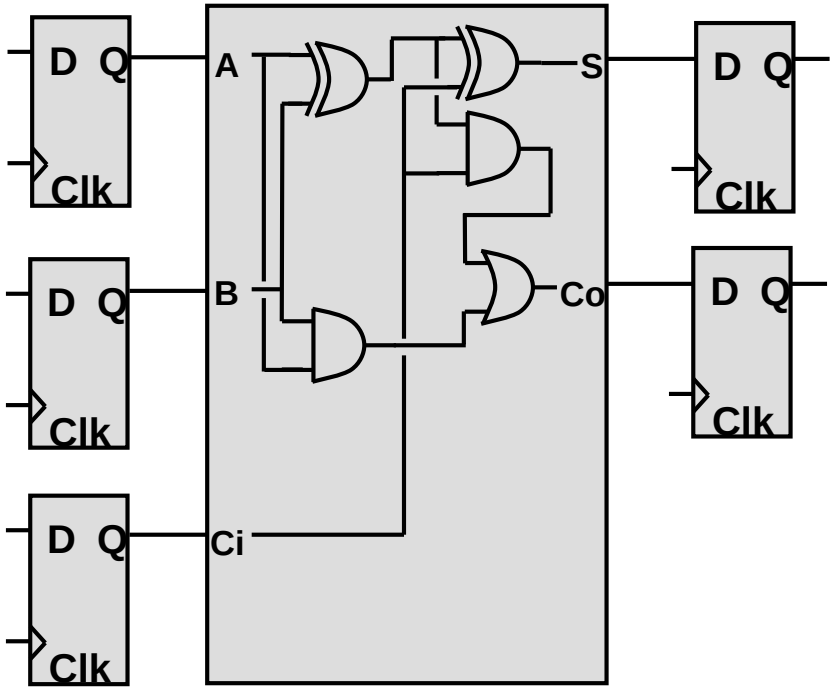
Static Timing Analysis (STA)

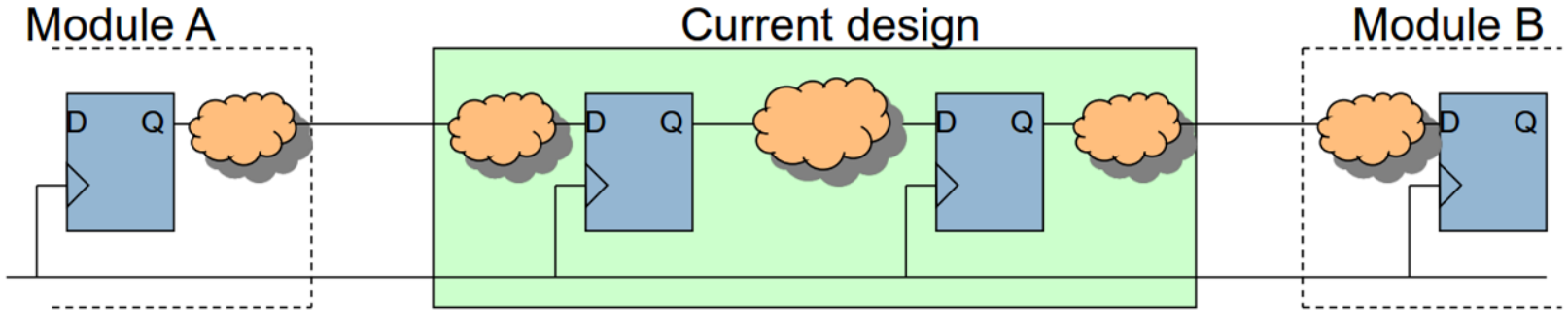
- **STA checks the worst case propagation of *all* possible vectors for min/max delays.**
- **Advantages:**
  - Much faster than timing-driven, gate-level simulation
  - Exhaustive, i.e., **every (constrained) timing path is checked**.
  - Vector generation NOT required
- **Disadvantages:**
  - Proper circuit functionality is NOT checked
  - Must define timing requirements/exceptions (garbage in → garbage out!)
- **Limitations:**
  - Only useful for *synchronous* design
  - Cannot analyze combinatorial feedback loops
    - e.g., a flip-flop created out of basic logic gates
  - Cannot analyze asynchronous timing issues
    - such as clock domain crossing
  - Will not check for glitching effects on asynchronous pins
    - Combinatorial logic driving asynch (set/reset) pins of sequential elements will not be checked for glitching



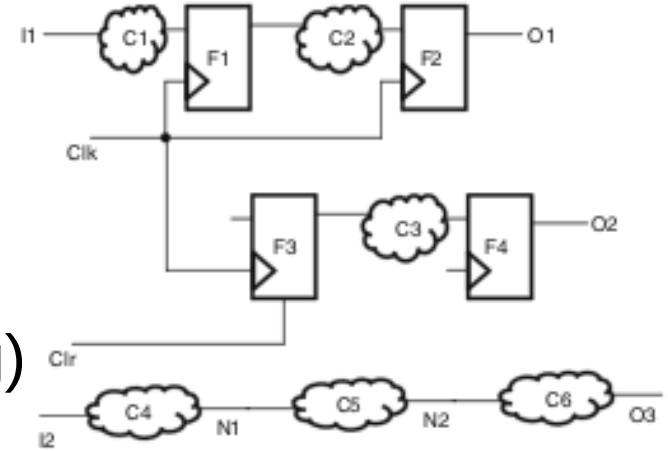
Timing Paths

- **A path is a route from a Startpoint to an Endpoint.**
- **Startpoint**, a.k.a Primary Inputs (PI)
  - Clock pins of the flip flops
  - Input ports
- **Endpoints**, a.k.a Primary Outputs (PO)
  - Input pins of the flip flops (except the clock pins)
  - Output ports
  - Memories / Hard macros
- **There can be:**
  - Many paths going to any one endpoint
  - Many paths for each start-point and end-point combination

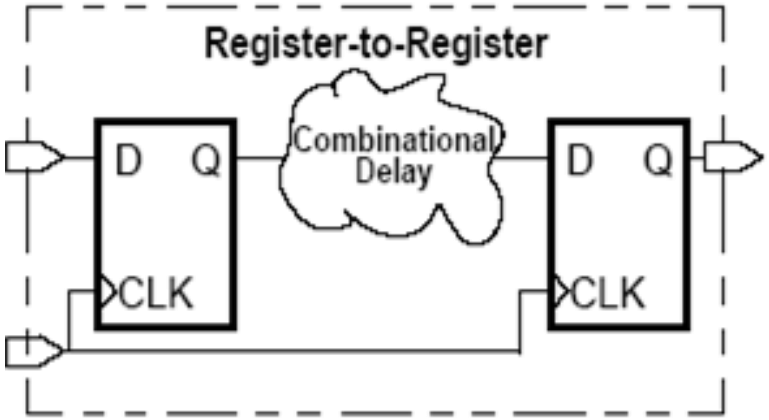


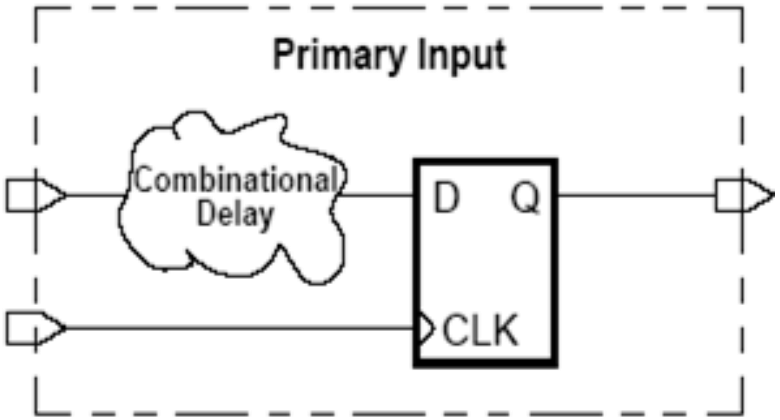


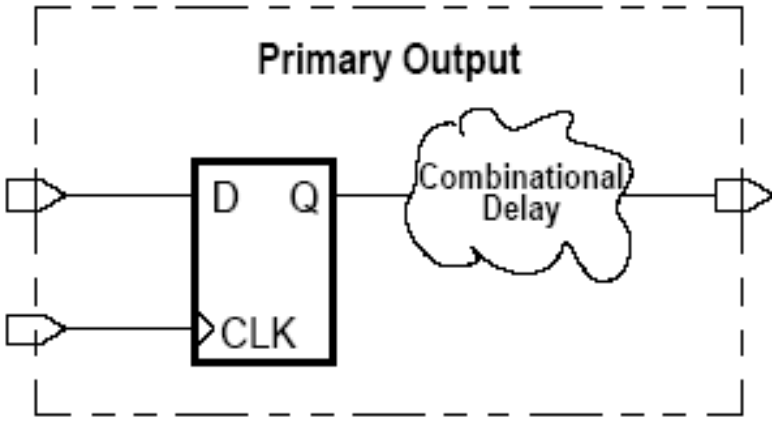
Static Timing Analysis

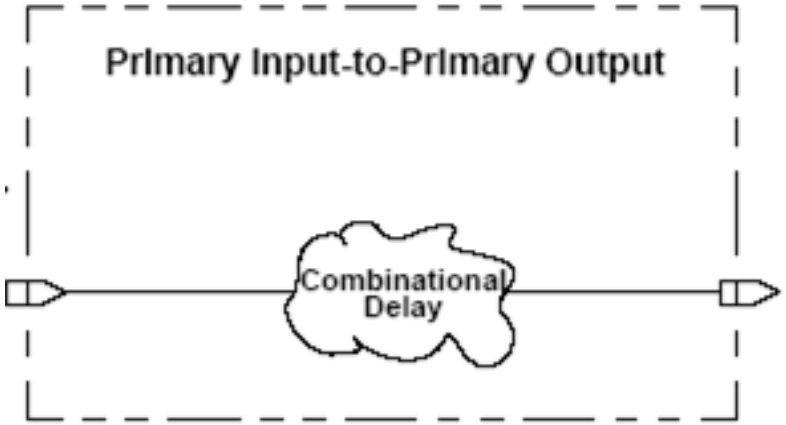


- **Four categories of timing paths**
  - Register to Register (reg2reg)
  - Register to Output (reg2out)
  - Input to Register (in2reg)
  - Input to Output (in2out)









Goals of Static Timing Analysis

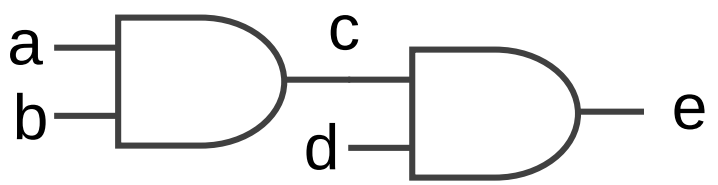
- **Verify max delay and min delay constraints are met for all paths in a design.**
  - Start with a Gate-Level Netlist.
  - Timing Models are provided for every gate in the library.
  - Static Timing Analysis needs to report if any path violates the max/min delay constraints.
- **But is this enough?**
  - No!
  - We want to know *all* the paths that violate the timing constraints.
  - In fact, we want to know the timing of *all* paths reported *in order* of length.
  - And we want to know *where* the problems are so we can go about fixing them.
- **Let's see the basic idea of how this can be done.**

Some basic assumptions

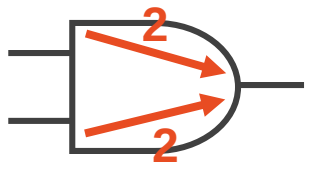
- **Our design is *synchronous***
  - In addition, we will only be showing how to deal with combinational elements and max delay constraints.
- **We will assume a *pin-to-pin* delay model**
  - In other words, each gate has a single, constant delay from input to output.
  - In the real world, gate delay is affected by many factors, such as gate type, loading, waveform shape, transition direction, particular pin, and random variation.
  - We will see how a real design gets all this data in the next lecture.
- **We will take a topological approach**
  - In other words, we disregard the logical functionality of the gates and therefore, consider all paths, though some of them cannot logically happen.
  - More on this later…

Simple path representation

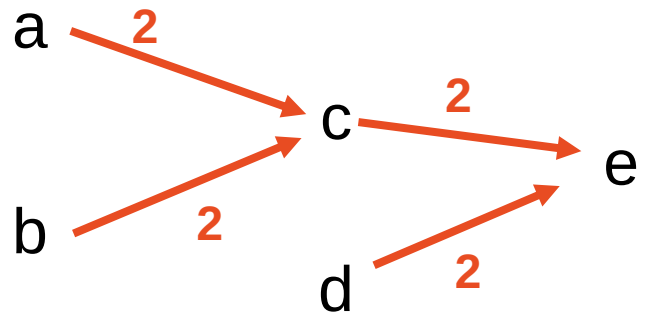
- **Let's say we have the following circuit:**



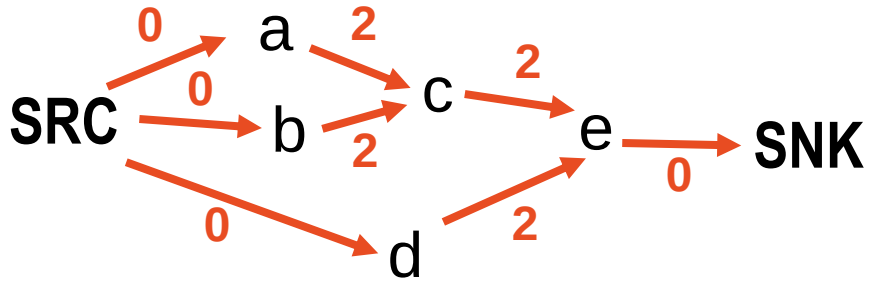
  - And the timing model of our AND gate is:



- **We will build a graph:**
  - Vertices: Wires, 1 per gate output and 1 for each PI and PO.
  - Edges: Gates, input pin to output pin, 1 edge per input with a delay for each edge.



- **Finally, add Source/Sink Nodes:**
  - 0-weight edge to each PI and from each PO.
  - That way all paths start and end at a single node.

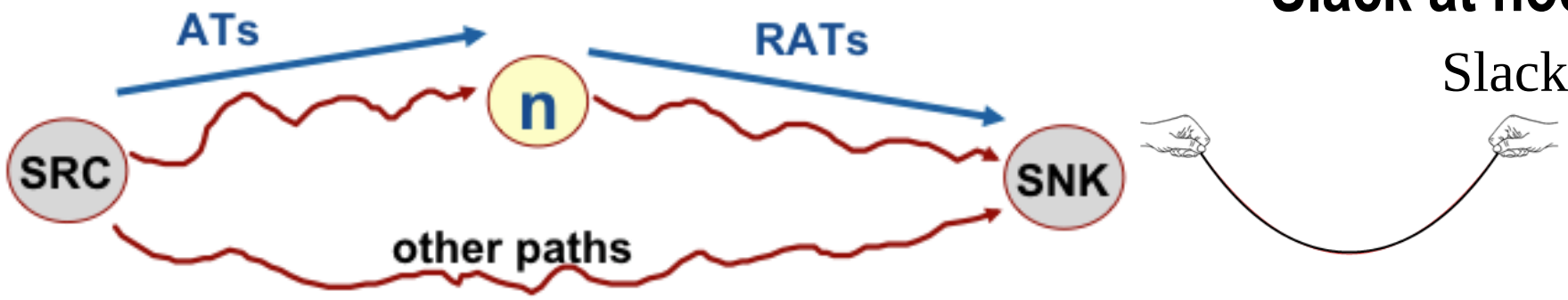


Node oriented timing analysis

- **If we would *enumerate* every path, we would quickly get *exponential explosion* in the number of paths.**
- **Instead, we will use *node-oriented* timing analysis**
  - For *each node*, find the *worst delay* to the node along *any path*.
- **For this, we need to define two important values:**
  - **Arrival Time at a node (AT)**: the longest path from the source to the node.
  - **Required Arrival Time at node (RAT)**: the latest time the signal *is allowed* to leave the node to make it to the sink in time.

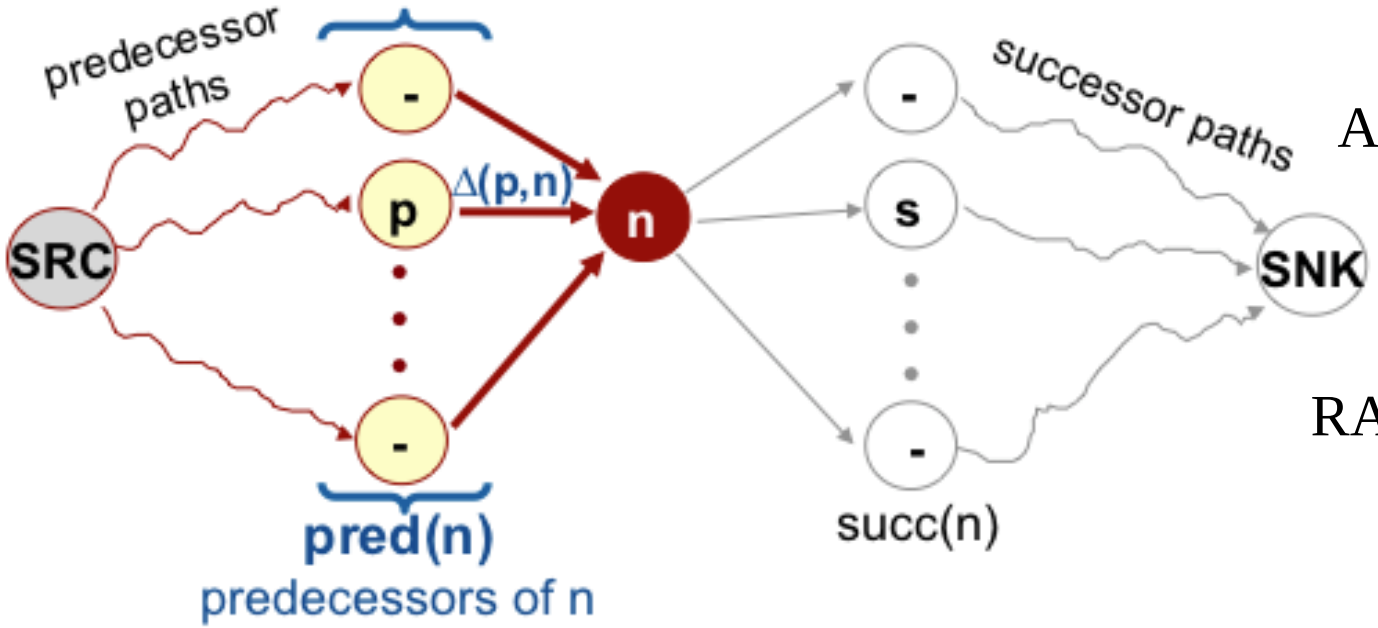
Slack at node n is defined as:

$$\text{Slack}(n) = \text{RAT}(n) - \text{AT}(n)$$



How do we compute ATs and RATs?

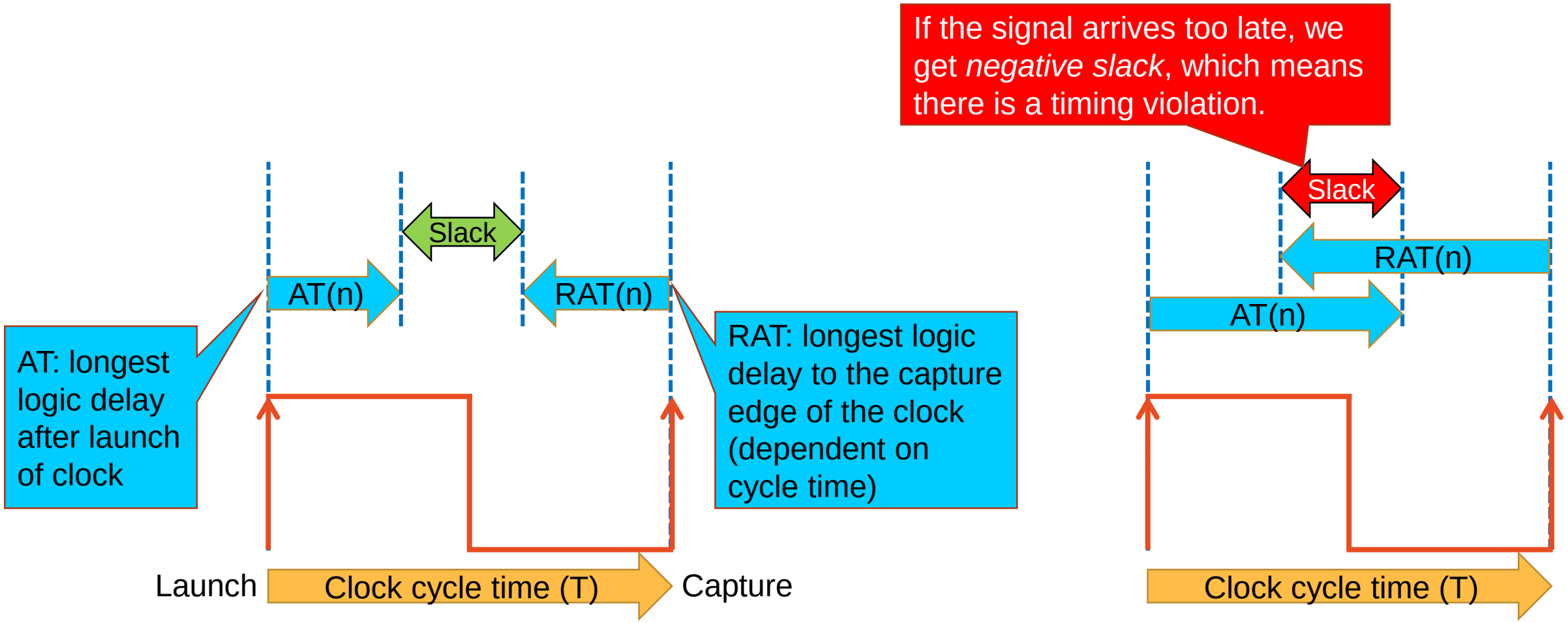
- **Recursively!**
  - The **Arrival Time** at a node is just the maximum of the *ATs* at the *predecessor* nodes plus the *delay from that node*.
  - The **Required Arrival Time** to a node is just the minimum of the *RATs* at the *successor* nodes minus the *delay to that node*.



$$\mathrm{AT}(n) = \begin{cases} 0 & n = \mathrm{SRC} \\ \max\limits_{p \in \mathrm{pred}(n)} \left[\mathrm{AT}(p) + \Delta(p,n) \right] & n \neq \mathrm{SRC} \end{cases}$$

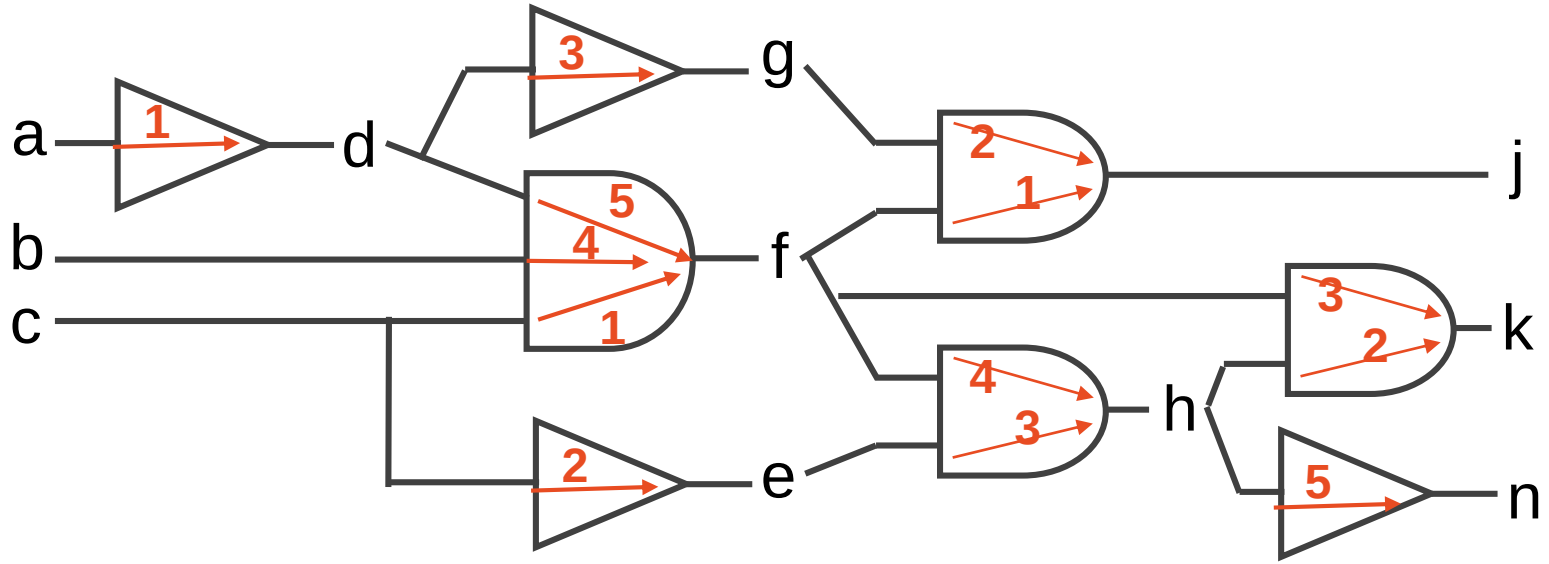
$$\mathrm{RAT}(n) = \begin{cases} T & n = \mathrm{SNK} \\ \max\limits_{s \in \mathrm{succ}(n)} \left[\mathrm{RAT}(s) + \Delta(n,s) \right] & n \neq \mathrm{SNK} \end{cases}$$

So let's try to understand AT, RAT, and Slack



Now let's see an example

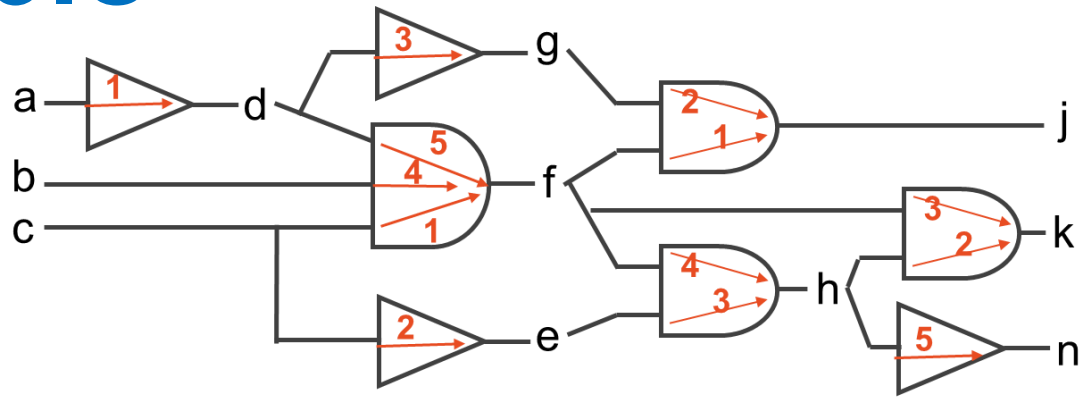
- **Just look at this path and try to find the worst path.**
  - Does it meet a cycle time of T=12 ?

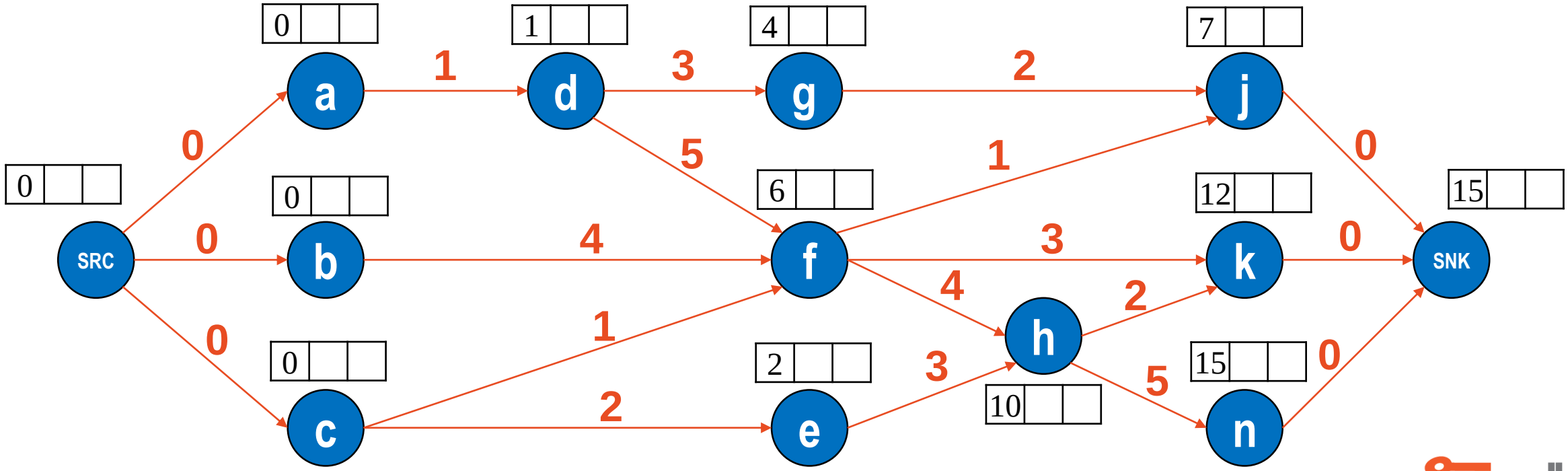


- **Now let's fill in the RAT, AT, and SLACK of each node and:**
  - Quickly find out if we meet timing
  - Figure out what the worst path is

Now let's see an example

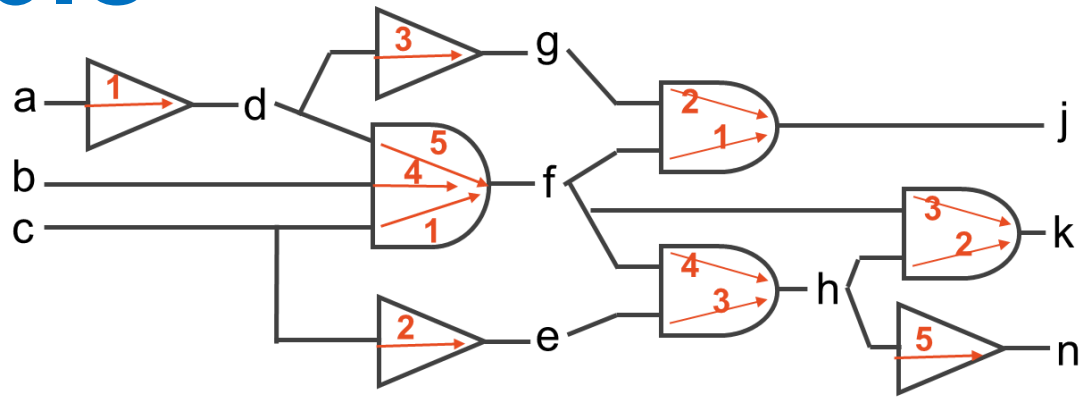
- We'll start by representing it as a directed acyclic graph (DAG)
- Next, we'll compute ATs from SRC to SNK

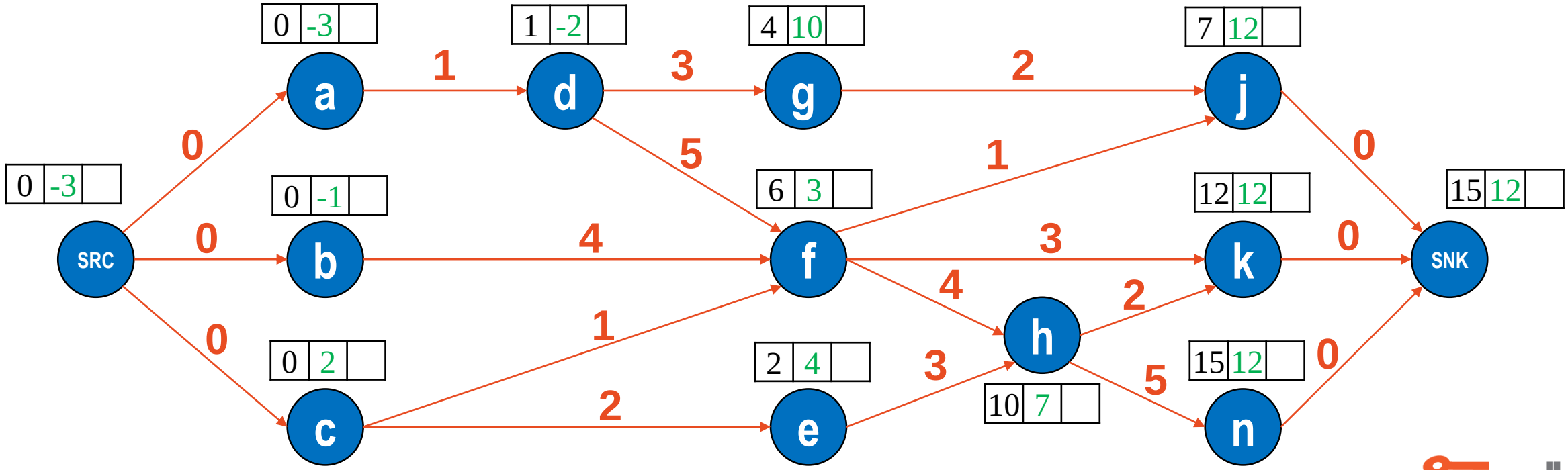




Now let's see an example

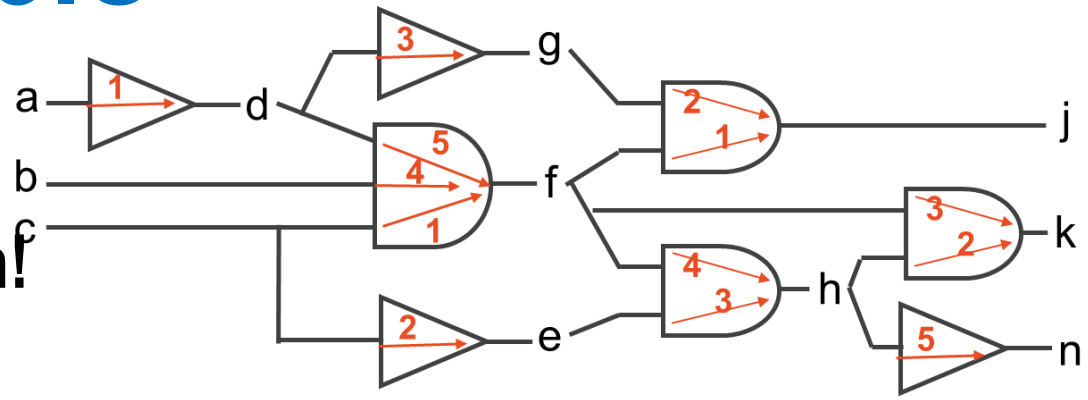
- **And now RAT from SNK to SRC**

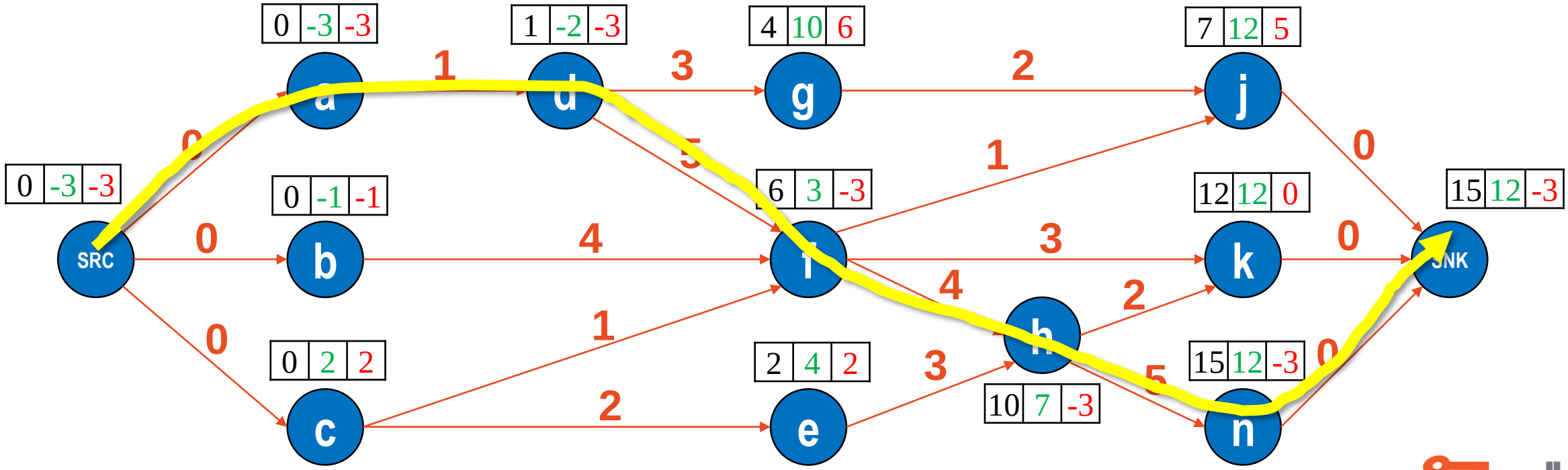




Now let's see an example

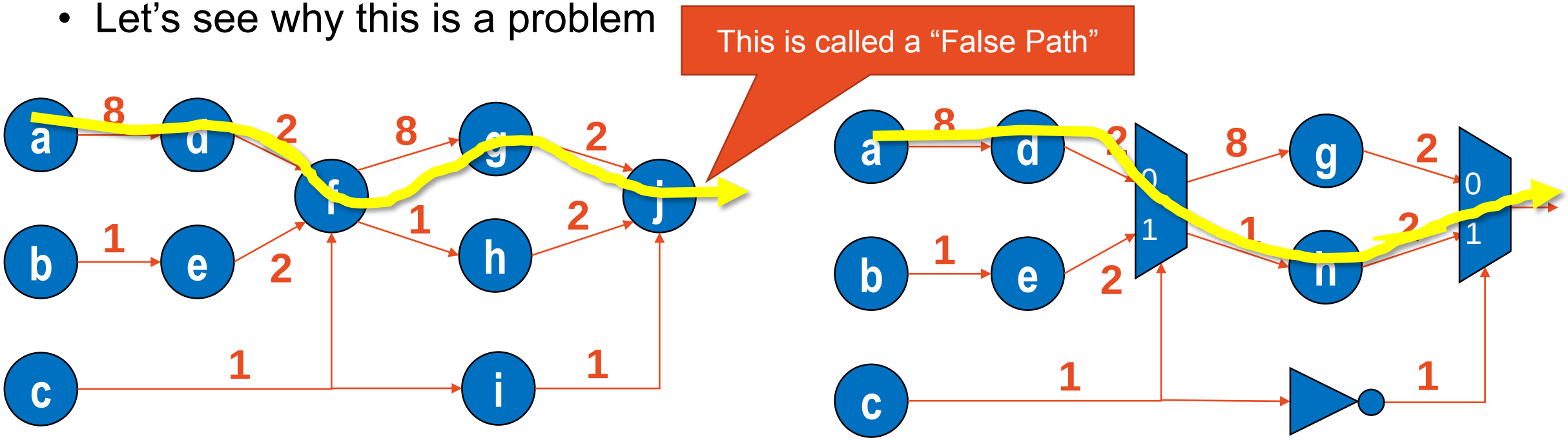
- And finally, we can calculate the slack.
- And guess what – we found the critical path!

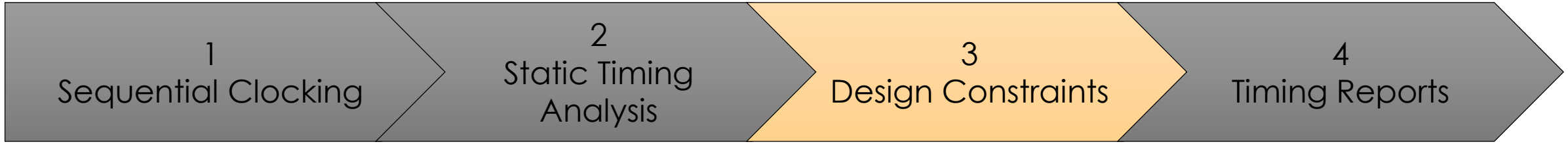




False Paths

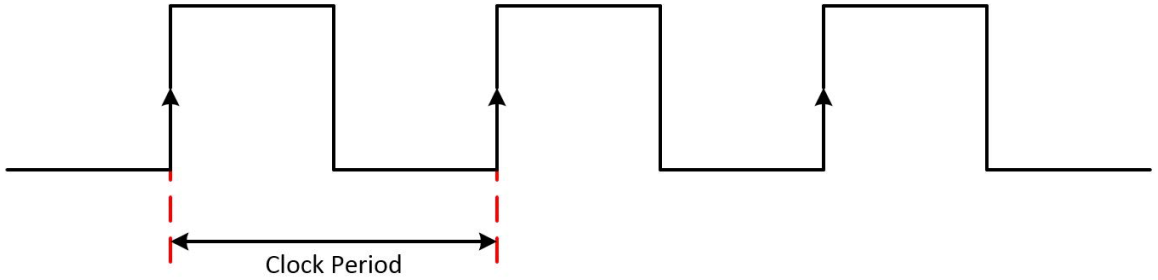
- **We saw how to find the RAT, AT and Slack at every node.**
  - All of this can be done very efficiently and be adapted for min timing, sequential elements, latch-based timing, etc.
  - Even better, we can quickly report the order of the critical paths.
- **However, this was all done topologically (i.e., without looking at logic).**
  - Let's see why this is a problem





Design Constraints

Timing Constraints



- **"Stupid Question":**
  - How does the STA tool know what the required clock period is?
- **Obvious Answer…**
  - We have to tell it!
  - We have to define *constraints* for the design.
  - This is usually done using the Synopsys Design Constraints (SDC) syntax, which is a superset of TCL.
- **Three main categories of timing constraints:**
  - Clock definitions
  - Modeling the world external to the chip
  - Timing exceptions

Collections

- **So you think you know `TCL`, right?**
  - Well EDA tools sometimes use a different data structure called a "`collection`"
- **A collection is similar to a `TCL` list, but:**
  - The value of a collection is not a string, but rather a pointer, and we need to use special functions to access its values.
  - For example, if you were to run `foreach` on a collection, it would just have one element (the pointer to the collection). Instead, use `foreach_in_collection`.
  - I won't go into the specifics here (see SynopsysCommandsReference), but these are some of the collection accessing functions:

```
foreach_in_collection
index_collection
sizeof_collection
sort_collection
```

```
filter_collection
add_to_collection
compare_collections
```

```
copy_collection
get_object_name
remove_from_collection
```

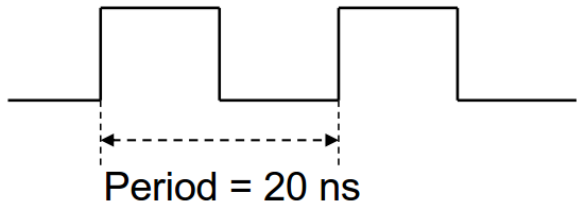
SDC helper functions

- **Before starting with constraints, let's look at some very useful built in commands:**
  - Note that all of these return `collections` and not `TCL lists`!
  - These will only work after design elaboration!
- **"`get`" commands:**
  - `[get_ports string]` – returns all ports that match string.
  - `[get_pins string]` – returns all cell/macro pins that match string.
  - `[get_nets string]` – returns all nets that match string.
    - Note that adding the `-hier` option will search hierarchically through the design.
- **"`all`" commands:**
  - `[all_inputs]` – returns all the primary inputs (ports) of the block.
  - `[all_outputs]` – returns all the primary outputs (ports) of the block.
  - `[all_registers]` – returns all the registers in the block.

Clock Definitions

- **To start, we must define a clock:**
  - Where does the clock come from? (i.e., input port, output of PLL, etc.)
  - What is the clock period? (=operating frequency)
  - What is the duty-cycle of the clock?

```
create_clock –period 20 –name my_clock [get_ports clk]
```



- **Can there be more than one clock in a design?**
  - Yes, but be careful about clock domain crossings! (…more later)
  - If a clock is produced by a clock divider, define a “generated clock”:

```
create_generated_clock –name gen_clock \
       -source [get_ports clk] –divide_by 2 [get_pins FF1/Q]
```

Clock Definitions (2)

- But during synthesis, we assume the clock is ideal, so:

```
set_ideal_network [get_ports clk]
```

- However, for realistic timing, it should have some transition:

```
set_clock_transition 0.2 [get_clocks my_clock]
```

- And we may want to add some jitter, so:

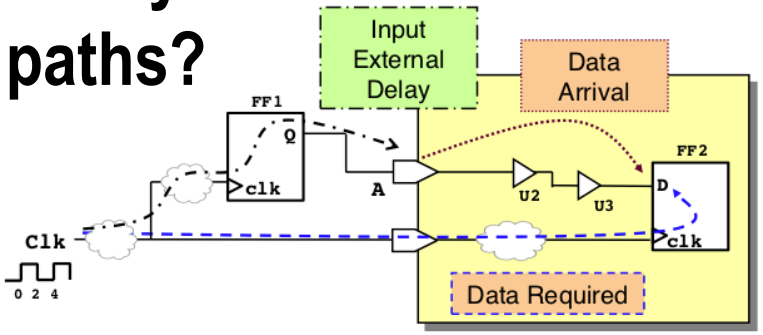
```
set_clock_uncertainty 0.2 [get_clocks my_clock]
```

- Finally, after building a clock tree, we do not want the clock to be ideal anymore, so:

```
set_propagated_clock [get_clocks my_clock]
```

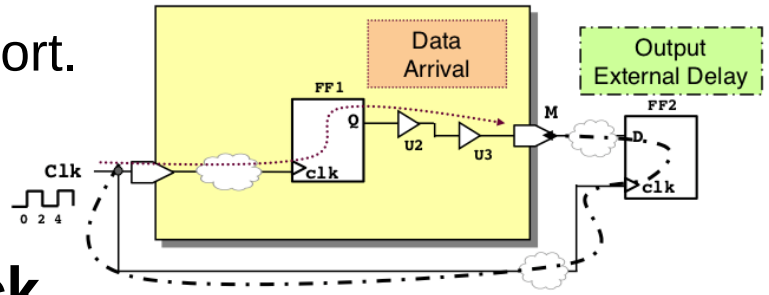
I/O Constraints

- **Now that the clock is defined, `reg2reg` paths are sufficiently constrained. However, what about `in2reg`, `reg2out`, and `in2out` paths?**
  - First, what clock toggles an I/O port?
  - And what about the time needed outside the chip?



- **A virtual clock is good practice for constraining I/O:**
  - Define a clock with the main clock period, but without a source port. This is a “virtual clock”.

```
create_clock -period 10 -name off_chip_clk
```



- **Now define I/O constraints according to the virtual clock.**
  - Input and output delays model the length of the path outside the block:

```
set_input_delay 0.8 -clock off_chip_clock \
       [remove_from_collection [all_inputs] [get_ports clk]]
set_output_delay 2.5 -clock off_chip_clk [all_outputs]
```

I/O Constraint (2)

- **An alternative approach is to define max delays to/from I/Os:**

```
set_max_delay 5 \
       -from [remove_from_collection [all_inputs] [get_ports clk]]
set_max_delay 5 -to [all_outputs]
```

- **Additionally, we must model the transitions on the inputs:**

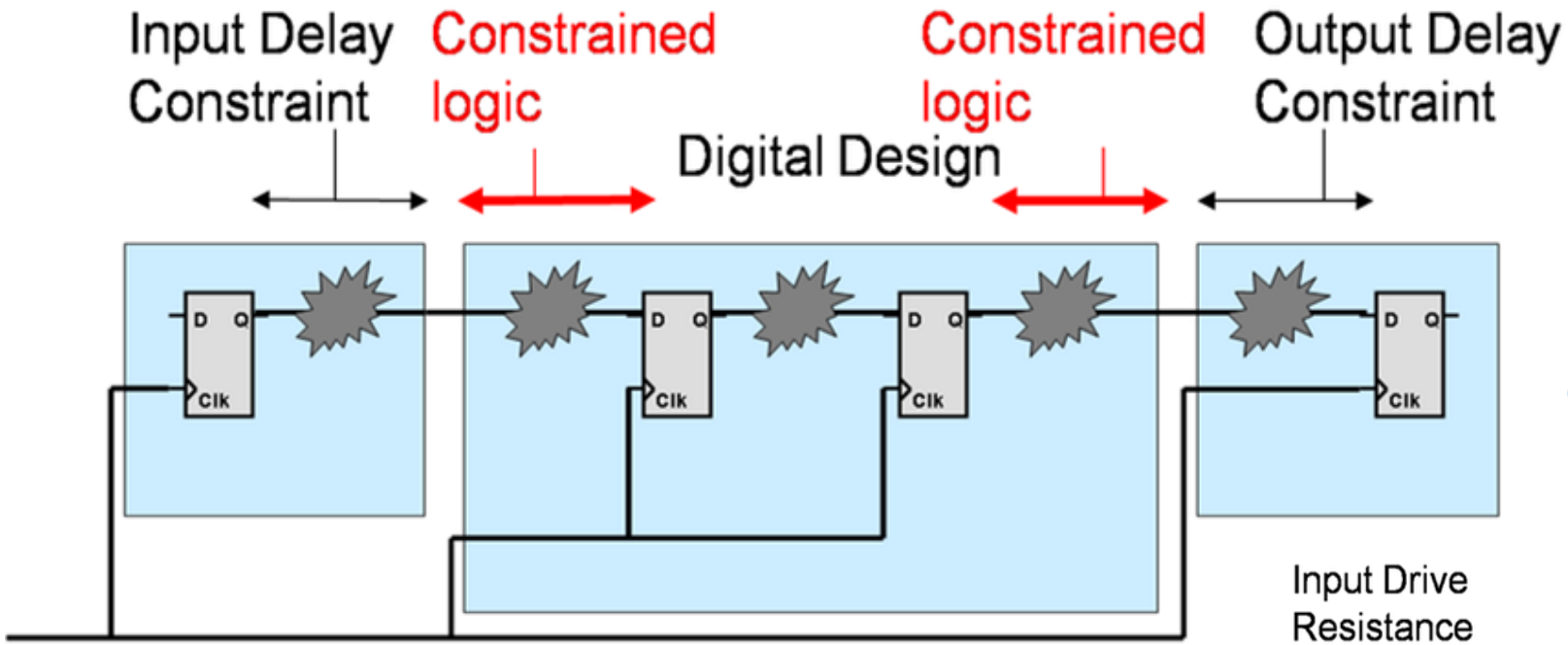
```
set_driving_cell -cell [get_lib_cells INV4] -pin Z \
       [remove_from_collection [all_inputs] [get_ports clk]]
```

- **And capacitance of the outputs:**

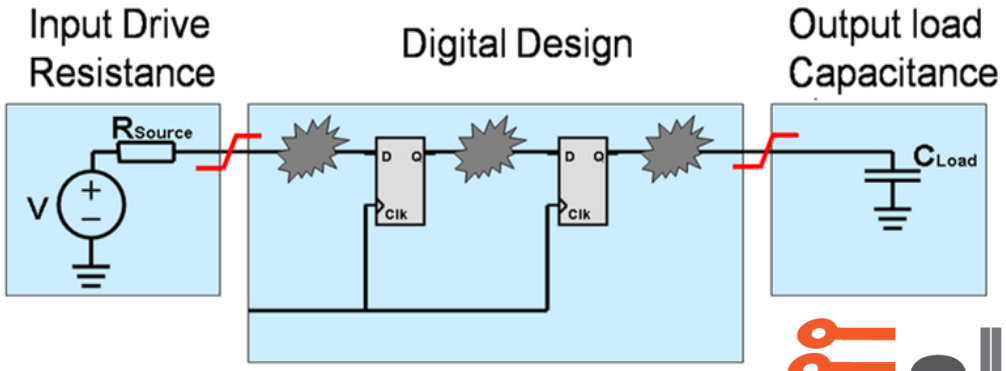
```
set_load 1 [all_outputs]
```

I/O Constraint (3)

- **Graphically, we can summarize the I/O constraints, as follows:**



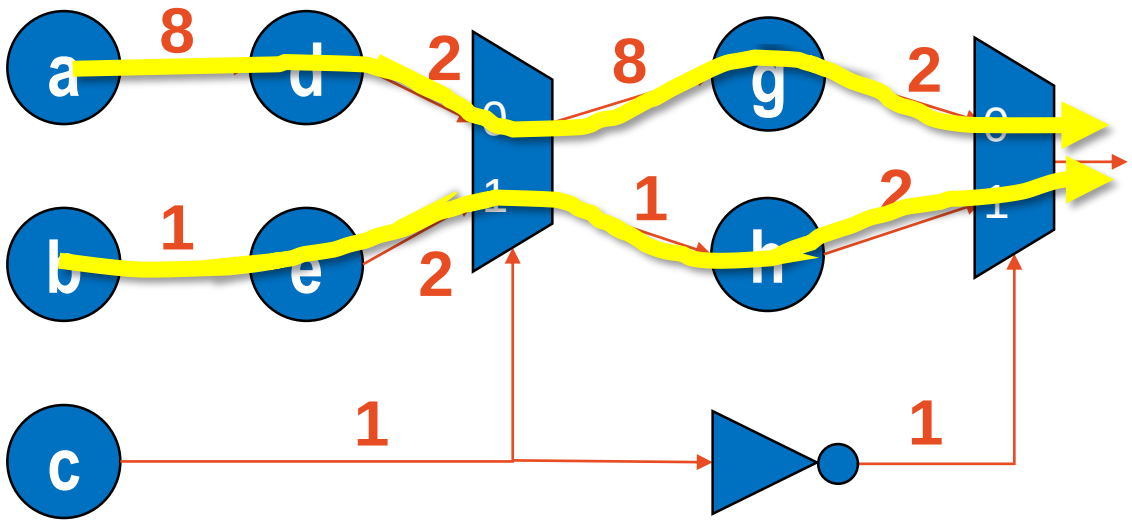
Input and Output Delays



Input drive and output cap modeling

Timing Exceptions

- There are several cases when we need to define exceptions that should be treated differently by STA.
- For example, looking into the topology of the network we saw earlier:



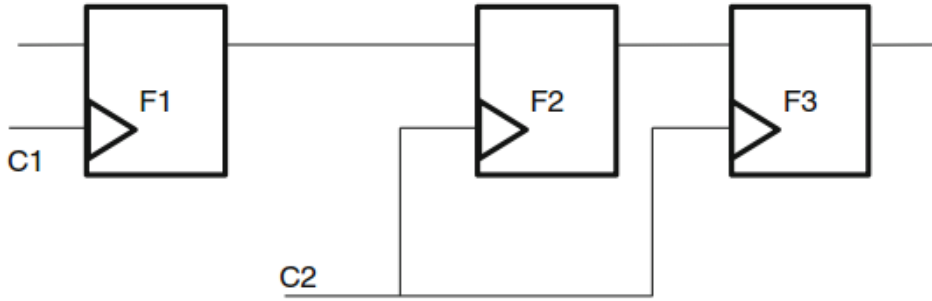
- In this case, we would define a *false path*:

```
set_false_path -through [get_pins mux1/I0] -through [get_pins mux2/I0]
set_false_path -through [get_pins mux1/I1] -through [get_pins mux2/I1]
```

Timing Exceptions (2)

- Another common case of a false path is a clock domain crossing through a synchronizer:

```
set_false_path –from F1/CP –to F2/D
```

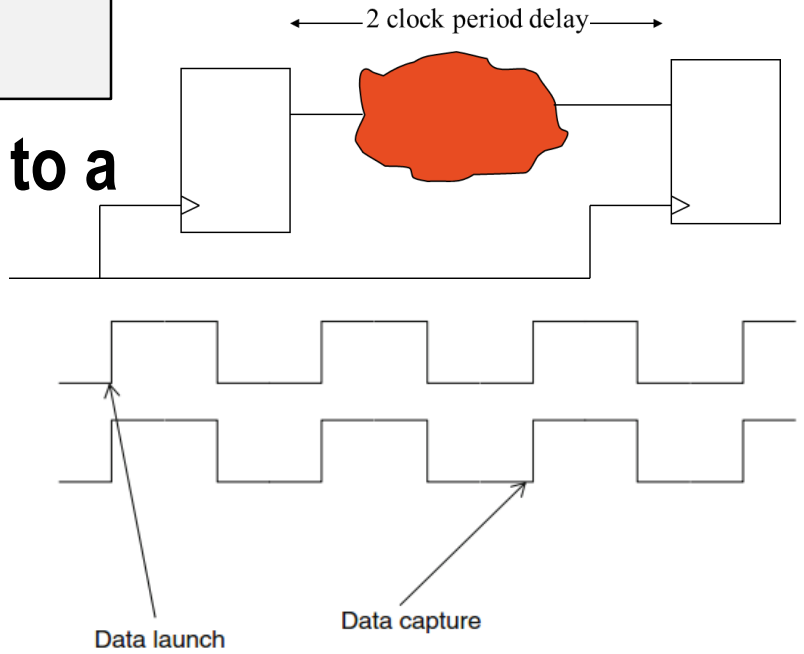


- Alternatively, this can be defined with:

```
set_clock_groups –logically_exclusive \
       –group [get_clocks C1] –group [get_clocks C2]
```

- If an equal-phase (divided) slow clock is sending data to a faster clock, a multi-cycle path may be appropriate:

```
set_multicycle_path –setup –from F1/CP –to F1/D   2
set_multicycle_path –hold  –from F1/CP –to F1/D   1
```



Case Analysis

- **A common case for designs is that some value should be assumed constant**
  - For example, setting a register for a certain operating mode.
- **In such cases, many timing paths are false**
  - For example, if the constant sets a multiplexer selector.
  - Or a ‘0’ is driven to one of the inputs of an AND gate.
- **To propagate these constants through the design and disable irrelevant timing arcs, a `set_case_analysis` constraint is used:**

```
set_case_analysis 0 [get_ports TEST_MODE]
```

Design Rule Violations (DRV)

- **You can set specific design rules that should be met, for example:**
  - Maximum transition through a net.

```
set_max_transition 0.1
```

  - Maximum Capacitive load of a net.

```
set_max_capacitance 0.1
```

  - Maximum fanout of a gate.

```
set_max_fanout 20
```

Yield-driven and Advanced STA

- **There are many more concepts, approaches, and terminologies used in timing analysis for high-yield signoff:**
  - On-chip Variation (OCV)
  - Advanced On-Chip Variation (AOCV)
  - Signal Integrity (SI)
  - and more and more…*
- **We will end with the basics now and get back to this towards the end of the course.**

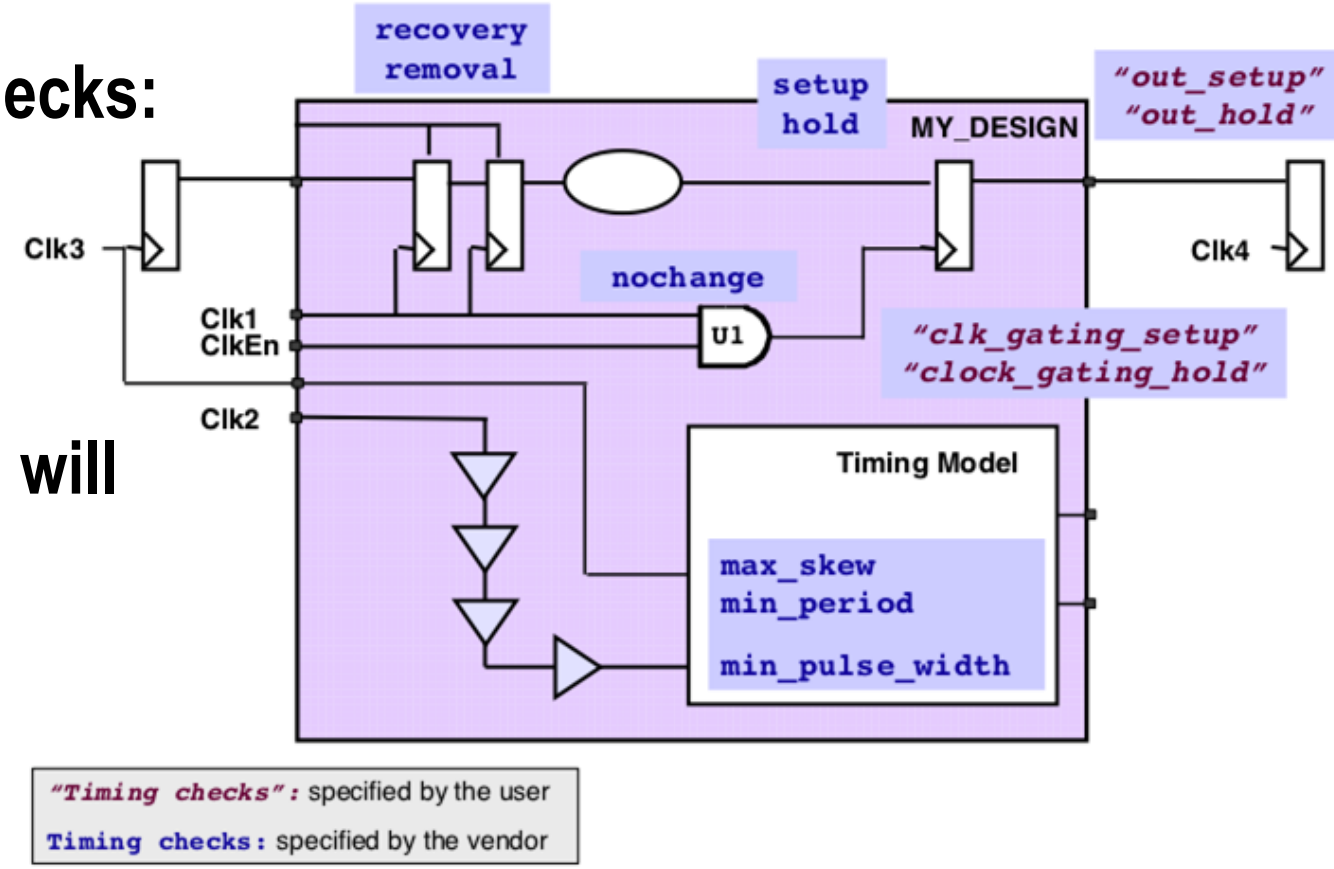
** Between the time I wrote this slide and presented it to you, each EDA vendor has presented another method for timing closure that you just must know about and have to use ☺.*

Constraint Checking and Timing Reports

Emerging Nanoscaled
Integrated Circuits and Systems Labs
Bar-Ilan University
אוניברסיטת בר-אילן

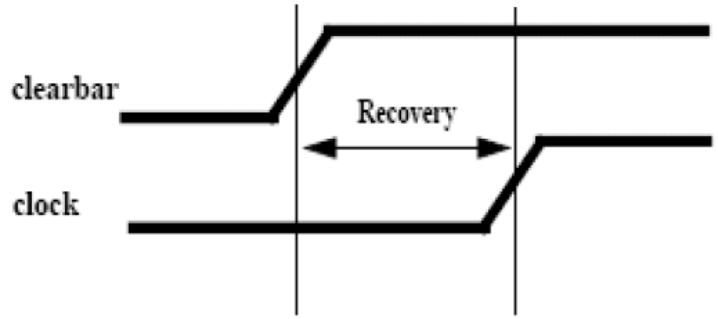
Check Types

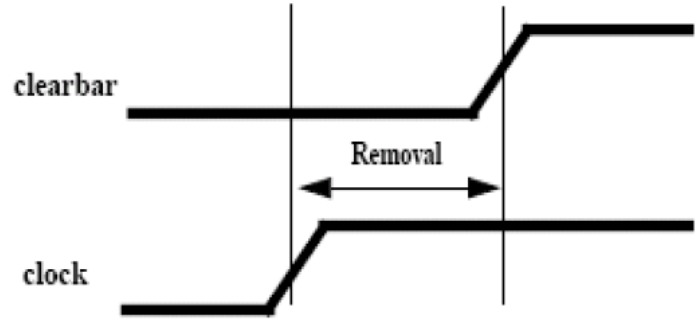
- **Throughout this lecture, we have discussed the two primary timing checks:**
  - Setup (max) Delay
  - Hold (min) Delay
- **However, in practice, there are other categories of timing checks that you will encounter:**
  - Recovery
  - Removal
  - Clock Gating
  - Min Pulse Width
  - Data-to-Data

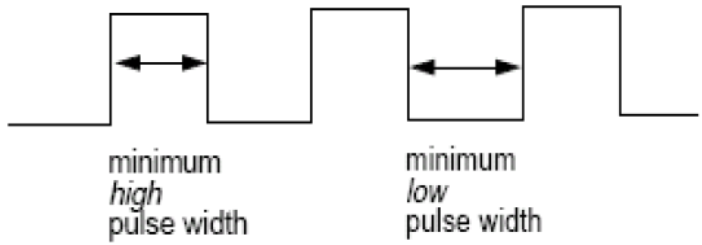


Recovery, Removal and MPW

- **Recovery Check**
  - The minimum time that an asynchronous control input pin must be stable *after* being *deasserted* and before the *next* clock transition (active-edge)
- **Removal Check**
  - The minimum time that an asynchronous control input pin must be stable *before* being *deasserted* and *after* the previous clock transition (active edge)
- **Minimum Clock Pulse Width (MPW)**
  - The amount of time *after* the rising/falling edge of a clock that the clock signal must remain stable.

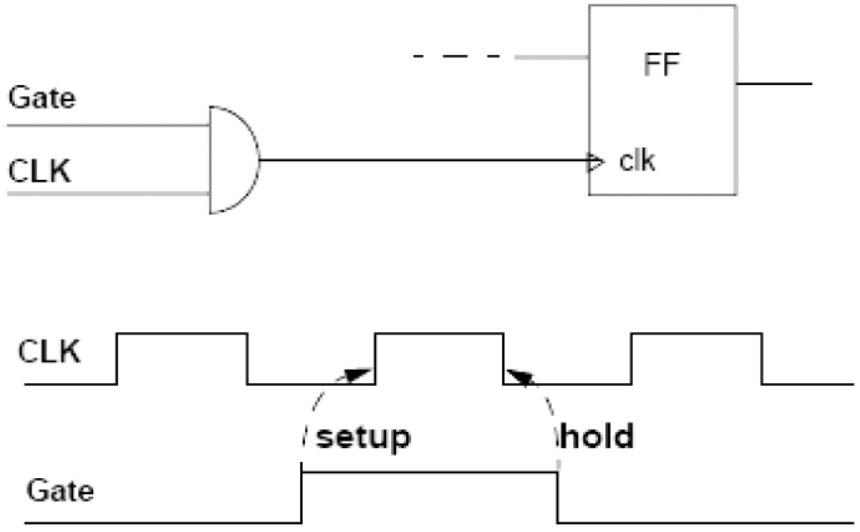




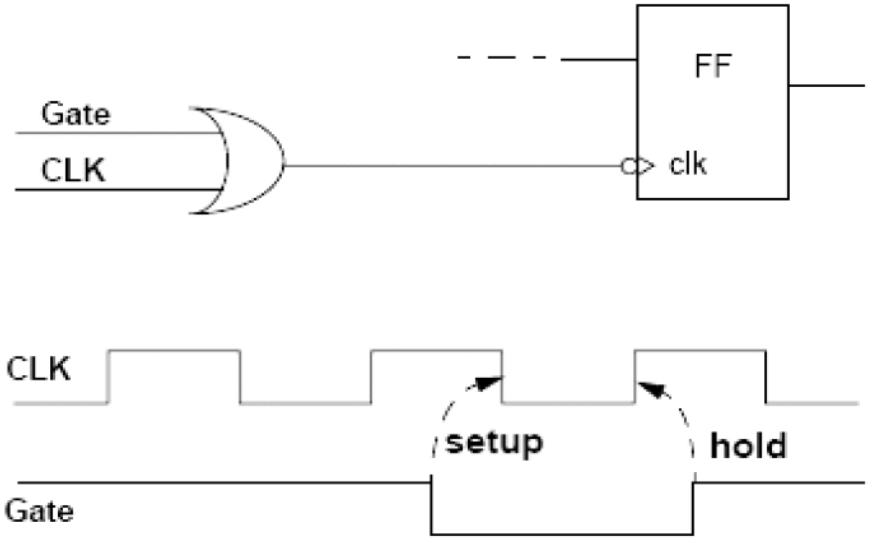


Clock Gating Check

- **Clock gating occurrences are any signals on the clock path that block (gate) the clock from propagating.**
- **The enable path of the clock gate must arrive enough time before the clock itself to ensure glitch-free functionality (and similarly hold after the edge).**



Ex. 1: Gating signal should only change when the clock is in the low state



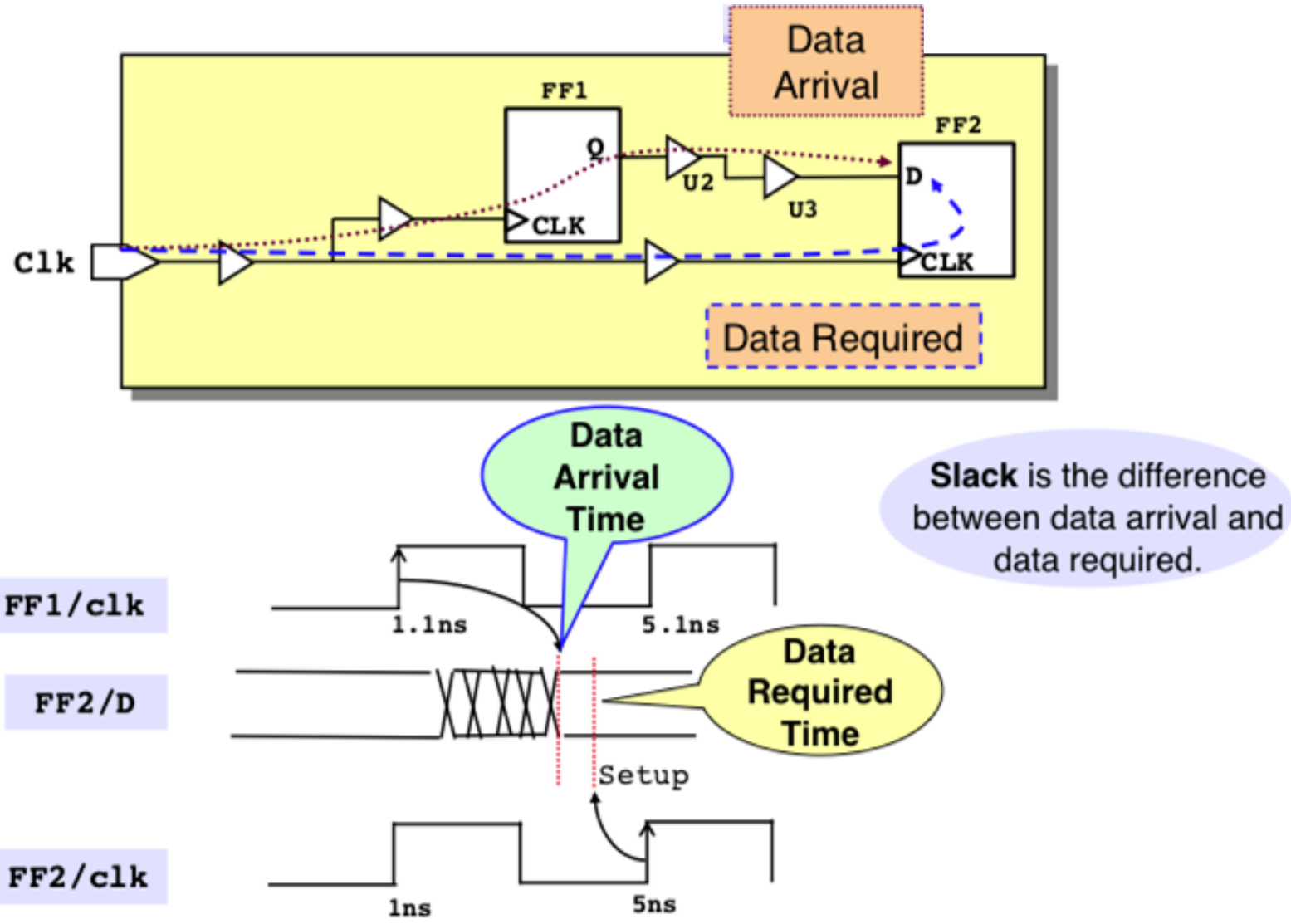
Ex. 2: Gating signal should only change when the clock is in high low state

Analysis Coverage

- **Use `report_analysis_coverage` and `check_timing` to ensure that you have fully constrained your design.**

Type of Check	Total	Met	Violated	Untested
setup	6724	5366 (80%)	0 (0%)	1358 (20%)
hold	6732	5366 (80%)	0 (0%)	1366 (20%)
recovery	362	302 (83%)	0 (0%)	60 (17%)
removal	354	302 (85%)	0 (0%)	52 (15%)
min_pulse_width	4672	4310 (92%)	0 (0%)	362 (8%)
clock_gating_setup	65	65 (100%)	0 (0%)	0 (0%)
clock_gating_hold	65	65 (100%)	0 (0%)	0 (0%)
out_setup	138	138 (100%)	0 (0%)	0 (0%)
out_hold	138	74 (54%)	**64 (46%)**	0 (0%)
All Checks	19250	15988 (84%)	64 (0%)	3198 (16%)

Report Timing - Terminology



Report Timing - Structure

`report_timing`

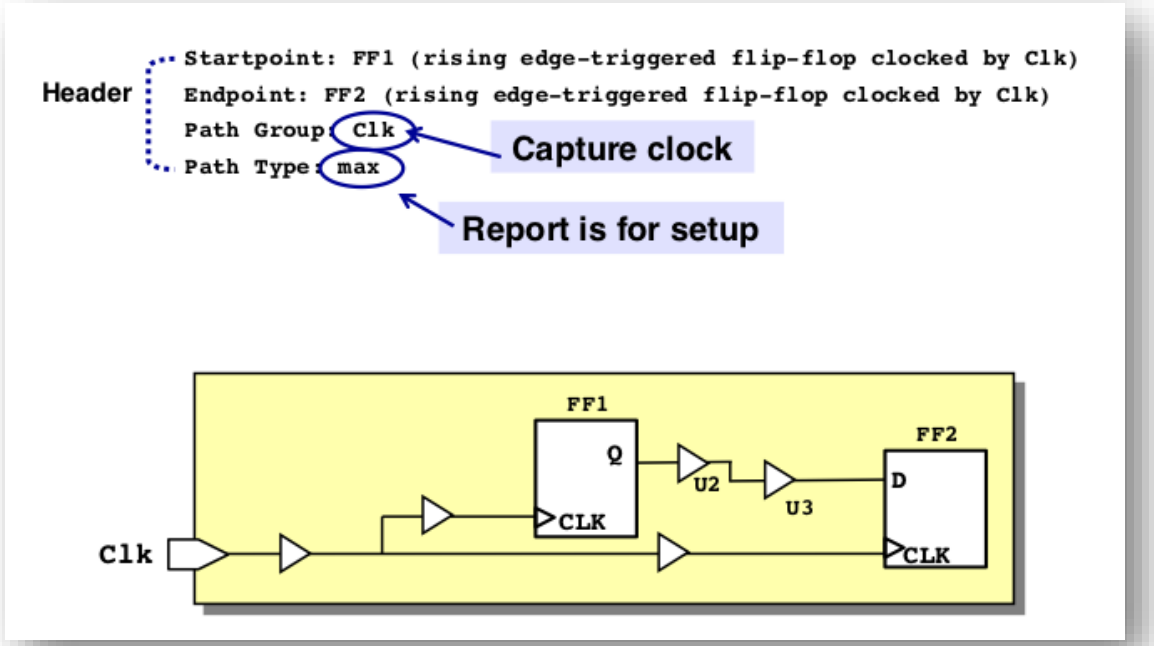
```
Startpoint: FF1 (rising edge-triggered flip-flop clocked by Clk)
Endpoint: FF2 (rising edge-triggered flip-flop clocked by Clk)
Path Group: Clk
Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock Clk (rise edge)                    0.00       0.00
  clock network delay (propagated)         1.10 *     1.10
  FF1/CLK (fdef1a15)                       0.00       1.10 r
  FF1/Q (fdef1a15)                         0.50 *     1.60 r
  U2/Y  (buf1a27)                          0.11 *     1.71 r
  U3/Y  (buf1a27)                          0.11 *     1.82 r
  FF2/D (fdef1a15)                         0.05 *     1.87 r
  data arrival time                                   1.87

  clock Clk (rise edge)                    4.00       4.00
  clock network delay (propagated)         1.00 *     5.00
  FF2/CLK (fdef1a15)                                  5.00 r
  library setup time                      -0.21 *     4.79
  data required time                                  4.79
  -----------------------------------------------------------
  data required time                                  4.79
  data arrival time                                  -1.87
  -----------------------------------------------------------
  slack (MET)                                         2.92
```

Header

Data arrival

Data required

Slack



Report Timing - Structure

Data Arrival Section

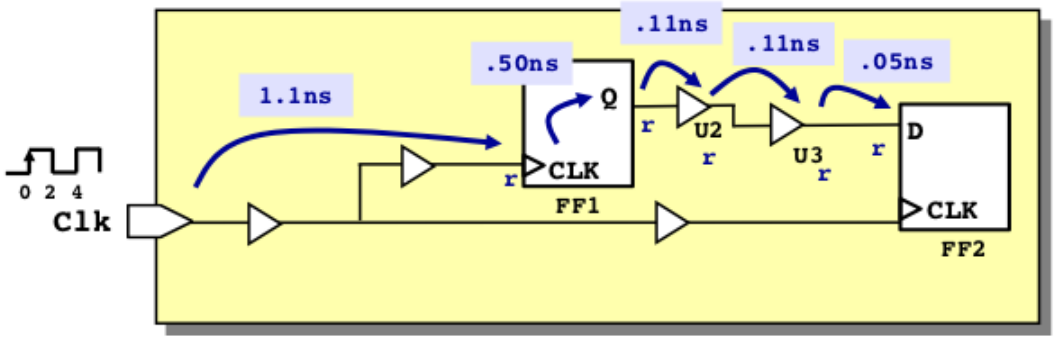
Calculated latency

SDF

Point	Incr		Path	
clock Clk (rise edge)	0.00		0.00	
clock network delay (propagated)	1.10	*	1.10	
FF1/CLK (fdef1a15)	0.00		1.10	r
FF1/Q (fdef1a15)	0.50	*	1.60	r
U2/Y (buf1a27)	0.11	*	1.71	r
U3/Y (buf1a27)	0.11	*	1.82	r
FF2/D (fdef1a15)	0.05	*	1.87	r
data arrival time			1.87	

Data arrival

Library reference names

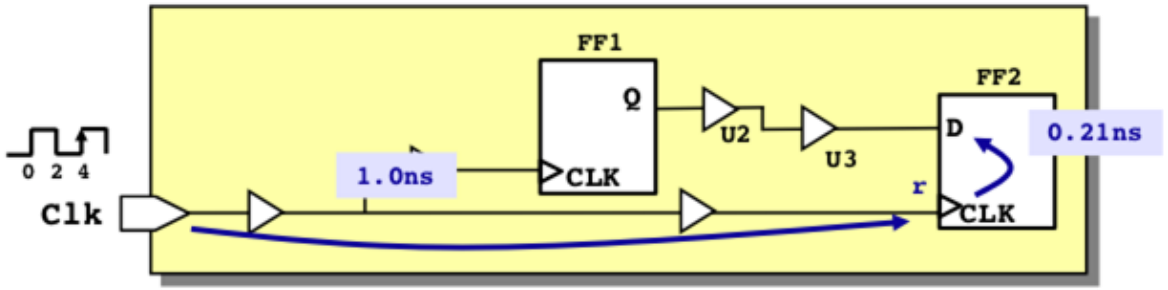


Data Required Section

Point	Incr		Path	
clock Clk (rise edge)	0.00		0.00	
clock network delay (propagated)	1.10	*	1.10	
FF1/CLK (fdef1a15)	0.00		1.10	r
FF1/Q (fdef1a15)	0.50	*	1.60	r
U2/Y (buf1a27)	0.11	*	1.71	r
U3/Y (buf1a27)	0.11	*	1.82	r
FF2/D (fdef1a15)	0.05	*	1.87	r
data arrival time			1.87	
clock Clk (rise edge)	4.00		4.00	
clock network delay (propagated)	1.00	*	5.00	
FF2/CLK (fdef1a15)			5.00	r
library setup time	-0.21	*	4.79	
data required time			4.79	

Data required

SDF



Slack

data required time	4.79
data arrival time	-1.87
slack (MET)	2.92

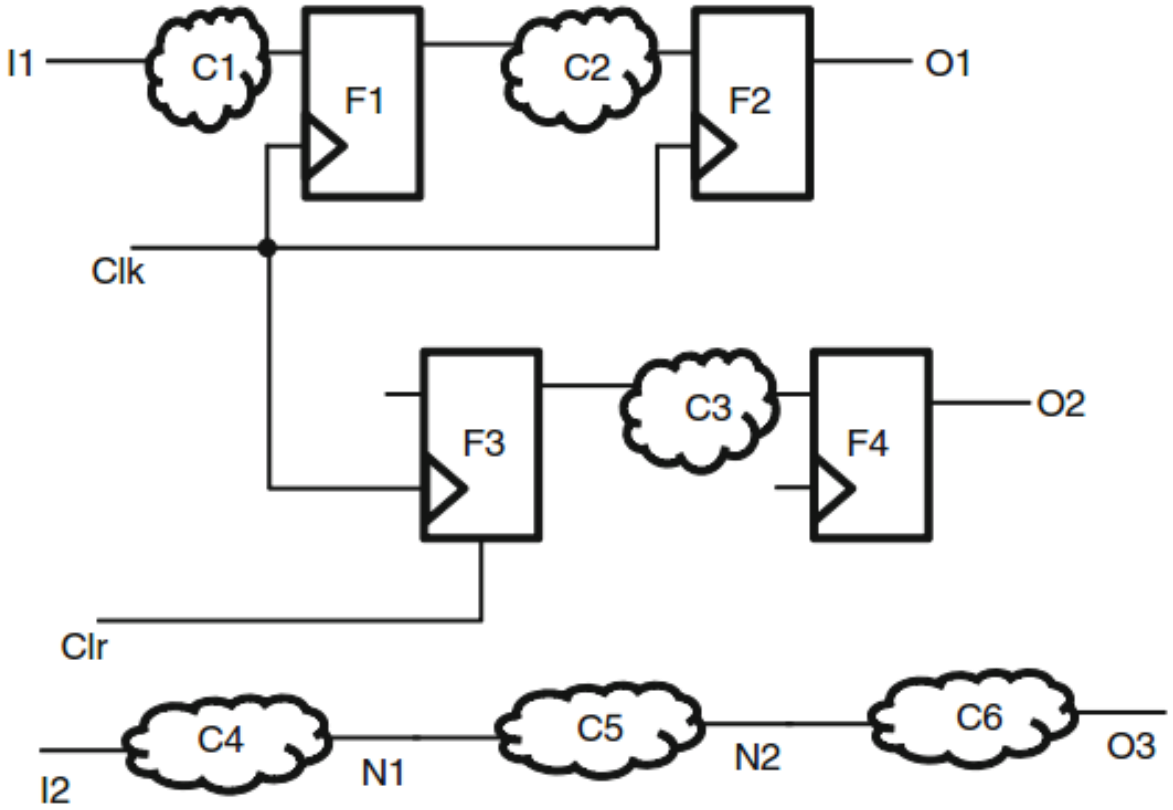
Slack

Example Hold Timing Report

```
Startpoint: FF1 (rising edge-triggered flip-flop clocked by Clk)
Endpoint: FF2 (rising edge-triggered flip-flop clocked by Clk)
Path Group: Clk
Path Type: min

Point                                          Incr       Path
-----------------------------------------------------------------
clock Clk (rise edge)                          0.00       0.00
clock network delay (propagated)               1.10 *     1.10
FF1/CLK (fdef1a15)                             0.00       1.10 r
FF1/Q (fdef1a15)                               0.40 *     1.50 f
U2/Y  (buf1a27)                                0.05 *     1.55 f
U3/Y  (buf1a27)                                0.05 *     1.60 f
FF2/D (fdef1a15)                               0.01 *     1.61 f
data arrival time                                         1.61

clock Clk (rise edge)                          0.00       0.00
clock network delay (propagated)               1.00 *     1.00
FF2/CLK (fdef1a15)                                        1.00 r
library hold time                              0.10 *     1.10
data required time                                        1.10
-----------------------------------------------------------------
data required time                                        1.10
data arrival time                                        -1.61
-----------------------------------------------------------------
slack (MET)                                               0.51
```

Path Groups

- **By default, all timing paths will be separated into standard path groups:**
  - `Reg2Reg`
  - `In2Reg`
  - `Reg2Out`
  - `In2Out`
  - `Clock Gating`



- **You can also define your own path groups to easily report them separately and/or to optimize them independently.**

Path Groups – Interface Timing Report

- For example, let's look at a `Reg2Out` path timing report:

```
Point                                          Incr       Path
-----------------------------------------------------------------
clock Clk (rise edge)                          0.00       0.00
clock network delay (propagated)               1.10 *     1.10
FF1/CLK (fdef1a15)                             0.00       1.10 r
FF1/Q (fdef1a15)                               0.50 *     1.60 r
U2/Y  (buf1a27)                                0.11 *     1.71 r
U3/Y  (buf1a27)                                0.11 *     1.82 r
M     (out)                                    0.05 *     1.87 r
data arrival time                                         1.87

clock Clk (rise edge)                          4.00       4.00
clock network delay (propagated)               0.00 *     4.00
output external delay                         -0.21 *     3.79
data required time                                        3.79
-----------------------------------------------------------------
data required time                                        3.79
data arrival time                                        -1.87
-----------------------------------------------------------------
slack (MET)                                               1.92
```

Report Timing Syntax

- **The syntax for the Innovus (Encounter) `report_timing` command is partially:**

```
report_timing  [-clock_from edge_from] [-clock_to clk_signame_list]
       [-early | -late] [-net]
       [-check_type {setup | hold | clock_gating_setup  | recovery | removal} ]
       [-max_paths integer] | [-nworst integer ]
       [{-from | -from_rise | -from_fall} pin_list ]
       [{-through | -through_rise | -through_fall} pin_list ]
       [{-not_through | -not_rise_through | -not_fall_through} object_list ]
       [{-to | -to_rise | -to_fall} pin_list ]
       [-point_to_point]
       [-path_group groupname_list ]
       [-path_type {end | summary | full | full_clock}]
       [-max_slack float ] [-min_slack float ]
       -unconstrained [-view { viewName }] [-format column_list ] [-collection]
       [-machine_readable | -tcl_list]
```

Report Timing Syntax

- **The Innovus (Encounter) `report_timing` format options are:**
  - `adjustment, annotation, arc, arrival, cell, delay, direction, edge, fanin, fanout, incr_delay, instance, instance_location, load, aocv_derate, net, phase, pin, hpin, pin_location, pin_load, wire_load, required, retime_delay, retime_incr_delay, retime_slew, slew, stolen, stage_count, timing_point, power_domain, user_derate, total_derate, and aocv_weight`

  - For example:

```
report_timing -check_type setup \
       -path_group Clock -path_type full_clock -max_paths 50 -net \
       -format {hpin cell delay required arrival required edge} \
       > timing_report.rpt
```

References

- Gil Rahav, BGU
- Gangadharan, Churiwala “Constraining Designs for Synthesis and Timing Analysis: A Practical Guide to Synopsys Design Constraints (SDC)”, Springer, 2013
- Synopsys SourceLink (+Synthesis Quick Reference)
- Cadence Support (+Genus and Innovus Text Command References)