Static Timing Analysis

Dr Noor Mahammad Sk

Introduction

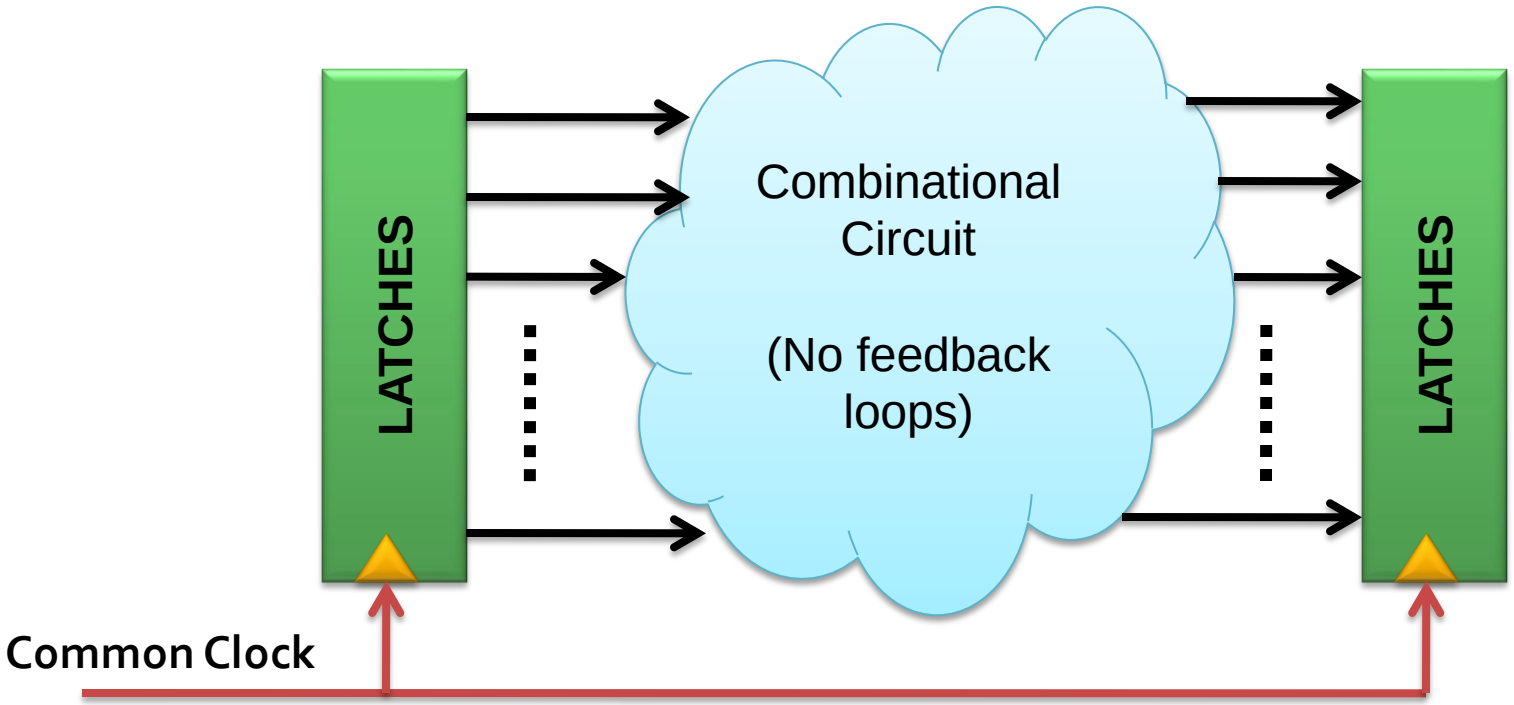
- Static timing analysis will be done at gate level netlist
- STA is computed at each and every stage of the design hierarchy

Analyzing Design Performance

- Basic Question
  - Does the design meet a given timing requirement, or
  - How fast can I run the design?
  - Assume we know the delays of blocks in the network
- Why not just use ordinary gate-level delay simulations
  - Requires too many parameters
    - Exponential in the number of design inputs
    - Even worse if we consider sequences needed to initialize latches
- So what do we do instead?
  - Separate function from time
  - Determine when transitions occur without worrying about how

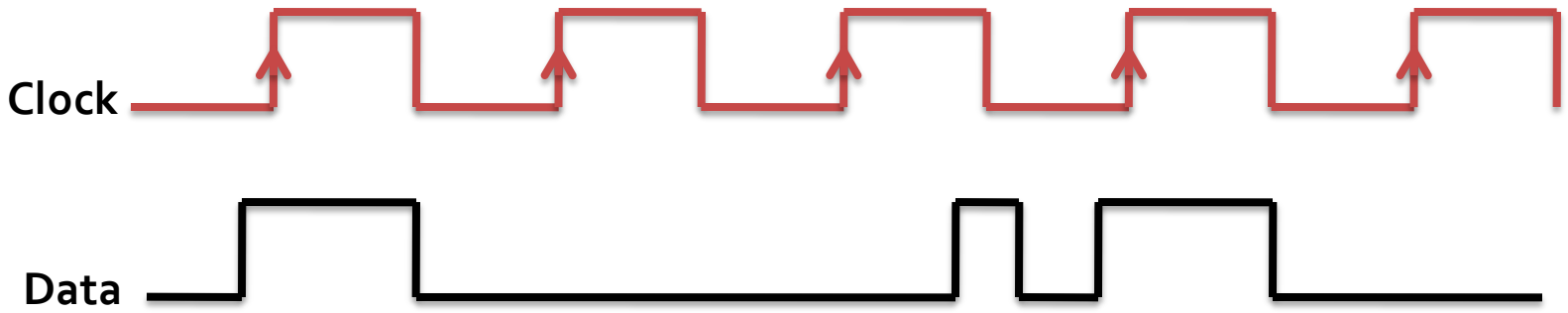
Analyzing Design Performance

- Assume design is synchronous
  - All storage is in explicit latch or flip-flop elements
  - All cycles cut by clocked storage elements



Analyzing Design Performance

- Consider an arbitrary signal in a clocked design
  - Takes on a value every cycle, sometimes one, sometimes zero
  - Change occur at different times in each cycle
    - Specific time of change depends on pattern causing it
    - May not change at all in some cycles
    - May make multiple changes before setting to final value

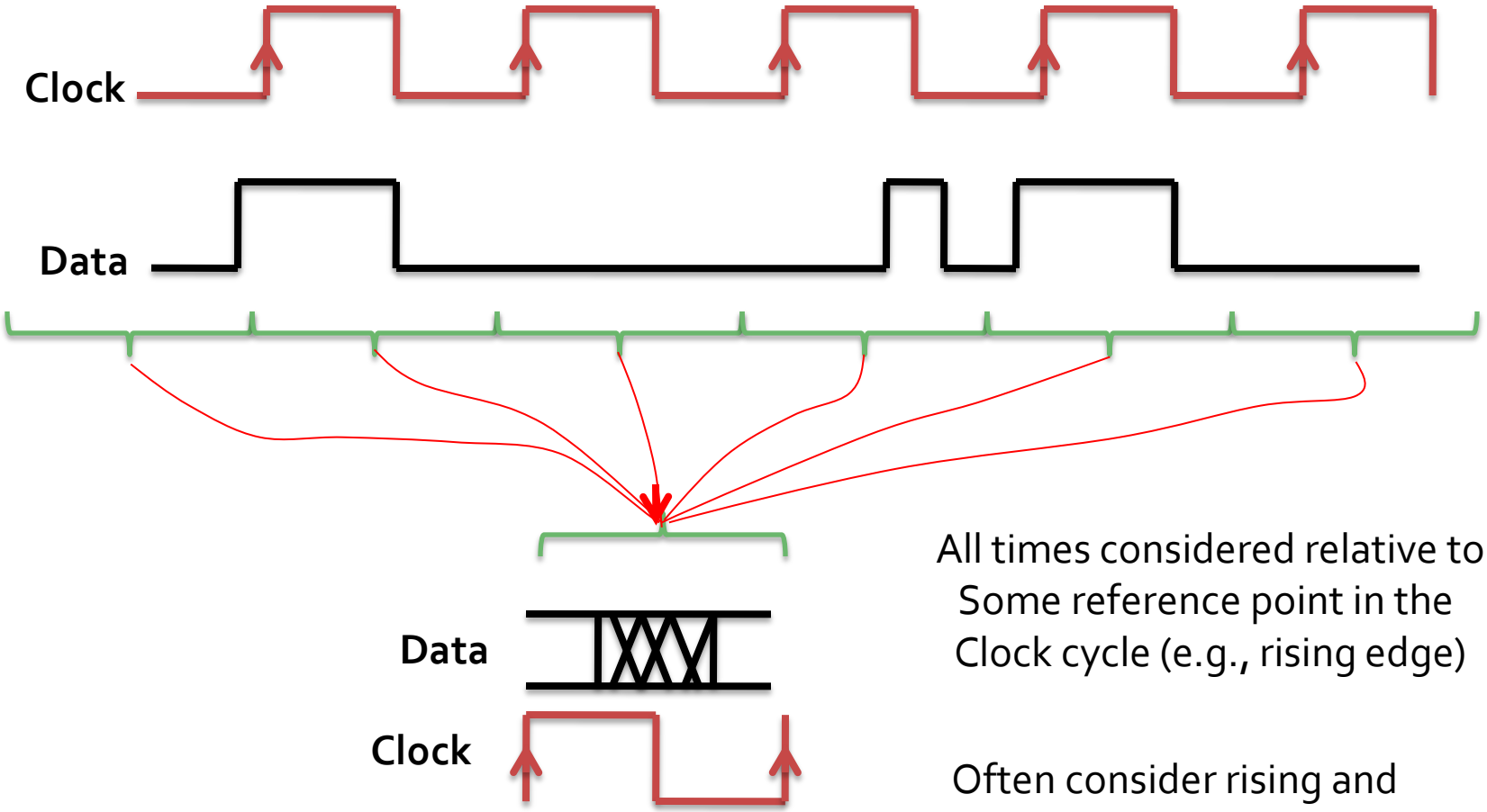


Static Timing Analysis

- Basic idea of static timing analysis
  - Instead of considering an infinitely long simulation sequence
  - Fold all possible transitions back into a single clock cycle
  - Assume that signal becomes stable at **latest** possible time
  - Assume signal becomes unstable at the **earliest** possible time
  - If the design works at these extremes, we can guarantee it always will
  - “Static” part just mean we aren’t doing simulation (dynamic)

Static Timing Analysis

- Look at our data signal again



All times considered relative to Some reference point in the Clock cycle (e.g., rising edge)

Often consider rising and falling times separately

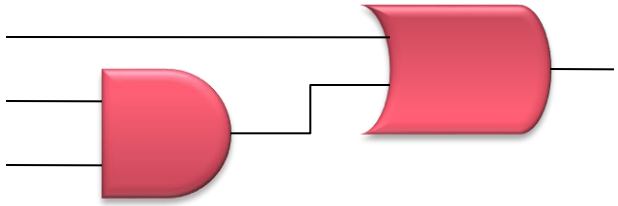
Timing Analysis: Basic Model

- So the basic questions are:
  - Does data always **reach** a stable value at all latch inputs in time for the clock to capture it?
    - Determine this by looking at late node timing or longest path
  - Does data always stay stable at all latch inputs long enough after the clock to get stored?
    - Determine this by looking at early mode timing, or shortest path

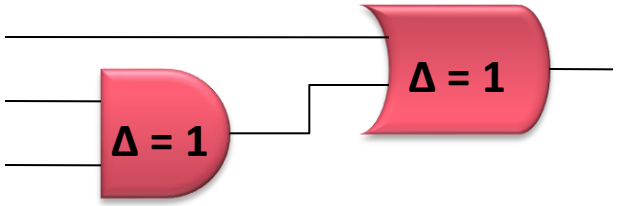
- What do we need to answer this?
  - First thing we need are "delay models" of the logic network
  - Surprising variety of options here
  - Depends on accuracy you need vs. computation you can afford

Delay Models

- Example gate network
  - 3 primary inputs (PIs) and 1 primary output (PO)



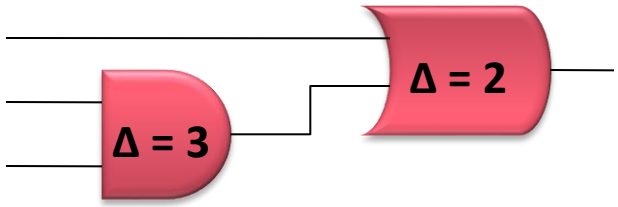
- Simplest model: unit delay
  - The delay through a gate – ANY gate – is equal to 1 time unit. period



Longest path = 2

Delay Models

- Better Model: Arbitrary but fixed delay per gate
  - Each gate is allowed to have its own fixed delay
  - This delay is constant – doesn't depend on circuit netlist

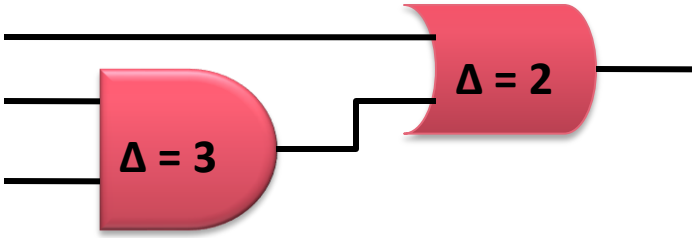


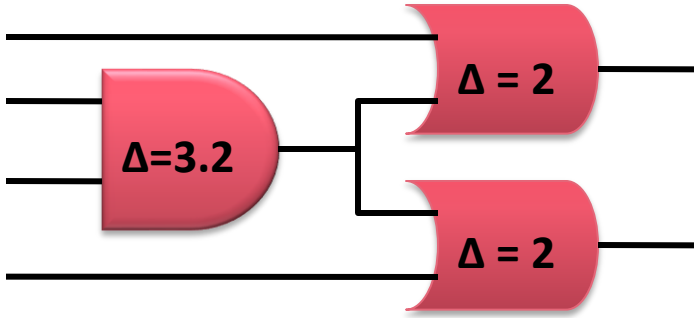
Longest path = 5

- Why isn't this enough?
  - Unfortunately, real circuits are made from gates made out of transistors, and a lot of other circuit effects are present

Delay Models

- The gate "loading" matters for delay
  - Gates with more fanout are slower than gates with less fanout
  - Look at the AND gate on left and right



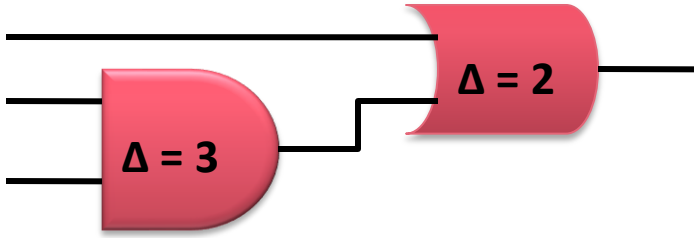


Gate output has to electrically drive all the fanout gates. More fanout means more load ➔ slower

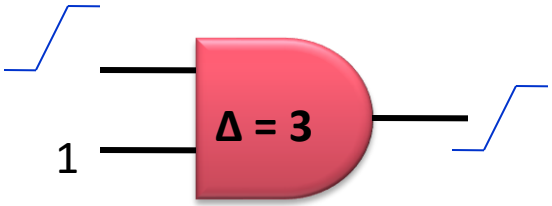
  - In real circuit, the loading presented by the connecting wires is actually the dominant contribution to the delay
  - Gate's delay model will usually depend on load of driven wires & gates
  - Delay through wires can be longer than delays through gates

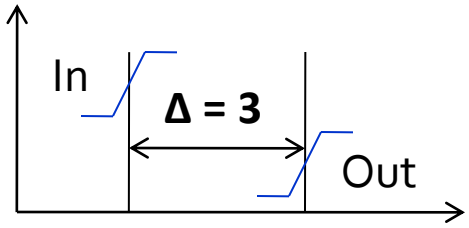
Delay Models

- The waveforms of the signals actually matter for delay
  - Rising signal versus falling signal matters. Delays may be asymmetric
  - Slope of the waveform seriously affects delay (RC circuit stuff)

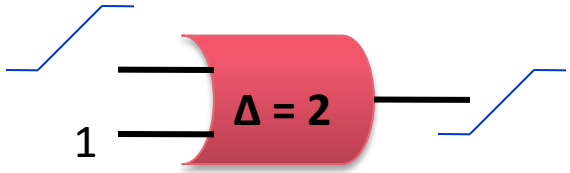


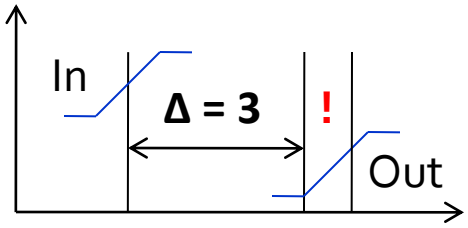
Sharp slope, fast rise





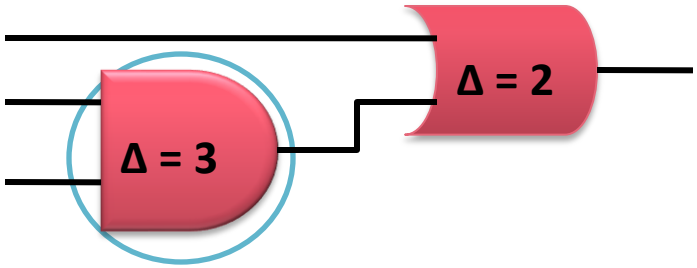
Poor slope, slow rise

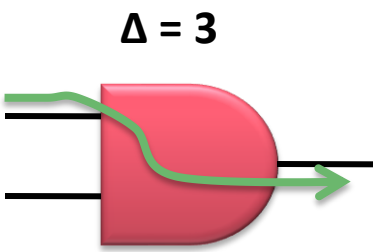


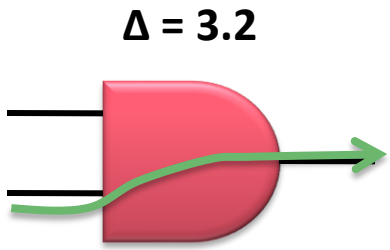


Delay Models

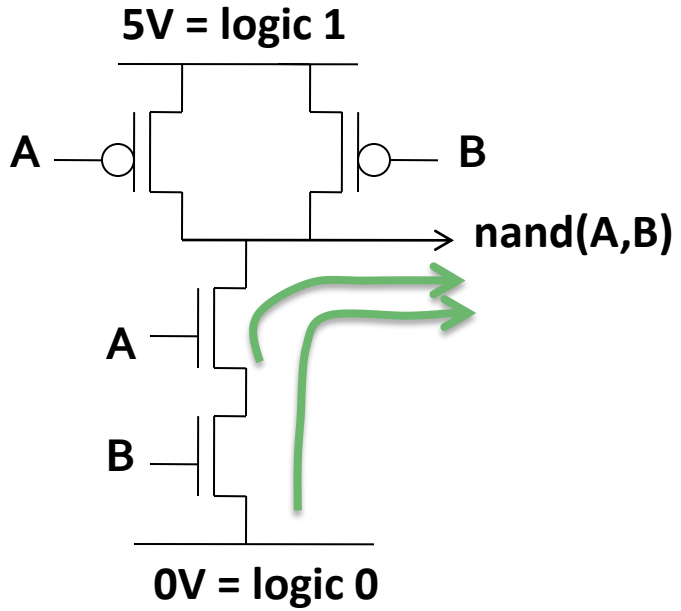
- Not all points are created equal
  - Delay is not really “through” a gate
  - Delay is from each individual pin to gate output(s); all can be different





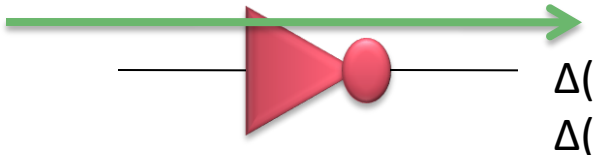


Why? Different transistor level circuit paths input to output



Delay Models

- Not all transistors are created equal
  - Separate transistors are used to drive a gate output to high/low values
  - Transistors may be different sizes, P & N devices have different mobilities, and topology of pull-up and pull-down paths differ
  - So delay can be different



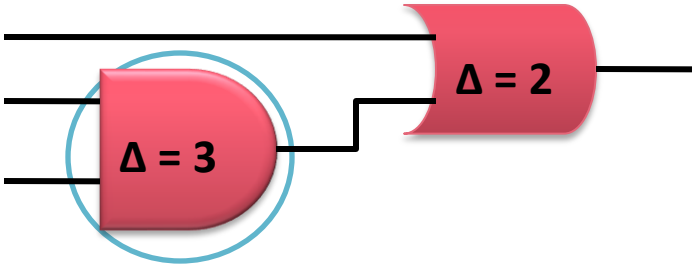
Δ(output falling) = 3.1
Δ(output raising) = 3.5

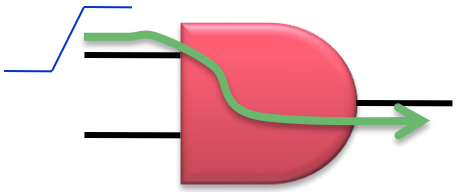
  - More complicated for non-monotonic functions

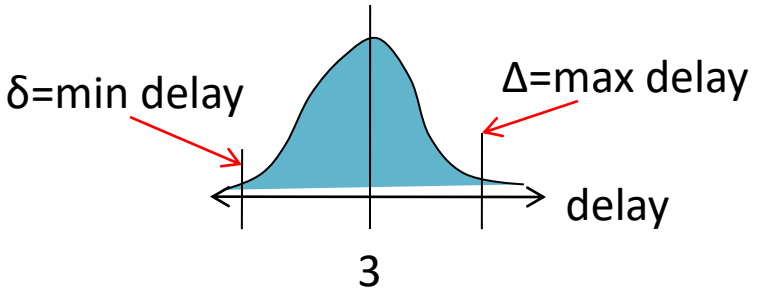
Δ(input falling, output falling) = 3.1
Δ(input falling, output raising) = 3.5
Δ(input raising, output falling) = 3.6
Δ(input raising, output raising) = 3.8

Delay Models

- Delays may not even be scalars; may be a distribution
  - Simplest is [min, max] which tries to quantify reasonable extremes on the manufacturing process
  - In most elaborate case, it's a real probability distribution that gives you a real probability of the signal arriving with a given delay
  - And this distribution can still be a function of ALL these factors: waveform slope, output loading, different delay per pin, etc.,

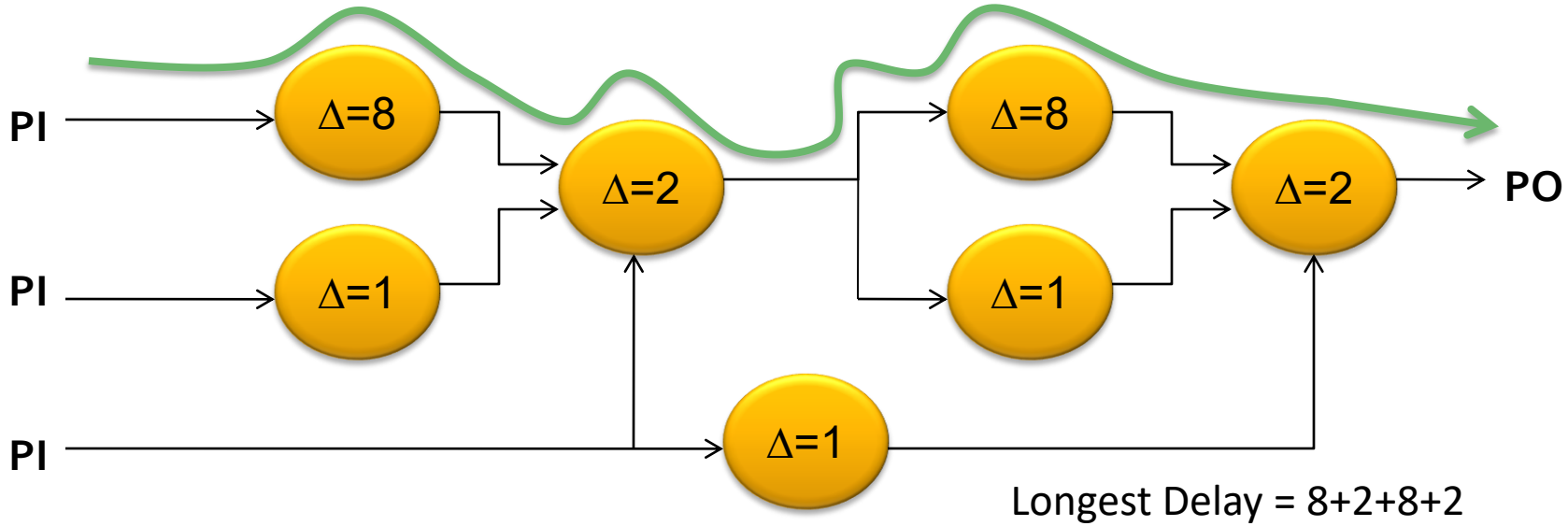






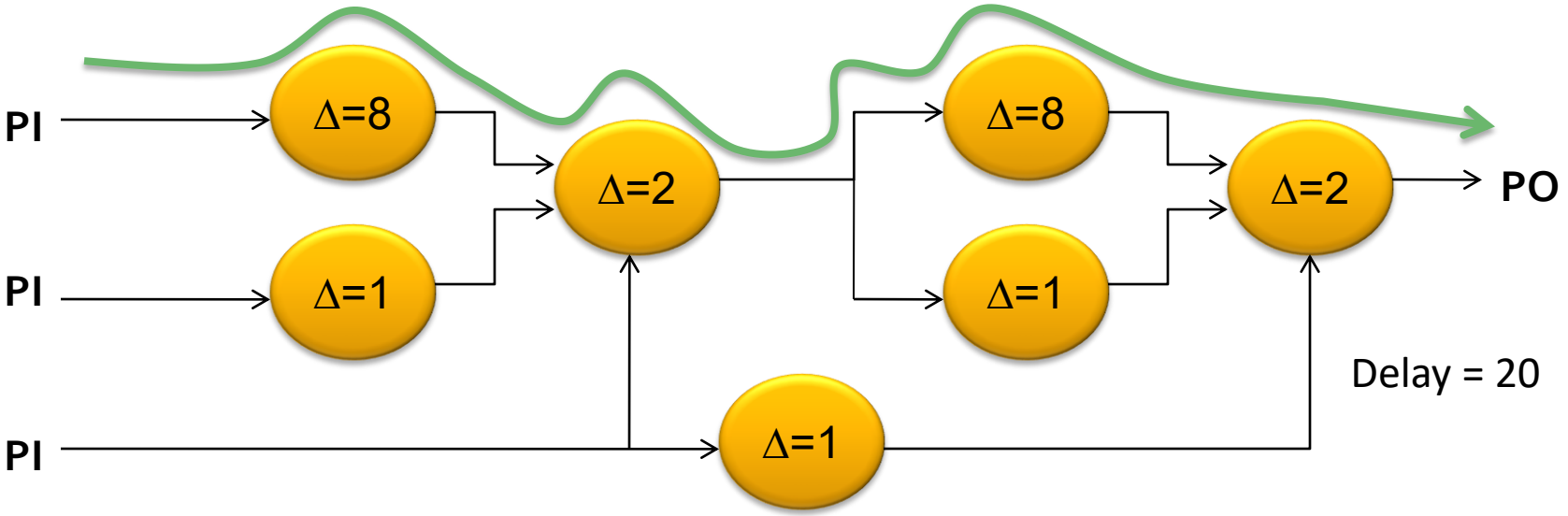
Timing Analysis: Topological vs. Logical

- Another problem: do we worry about the "function"?
  - Logical timing analysis: YES, we care that the gates actually do
  - Topological timing analysis: NO, we don't care what gates do
- What's the difference? Try an example..
  - Topological analysis means we only worry about the delay through the paths through the graph shown below, not the logical function of the modules (which we hide here!)



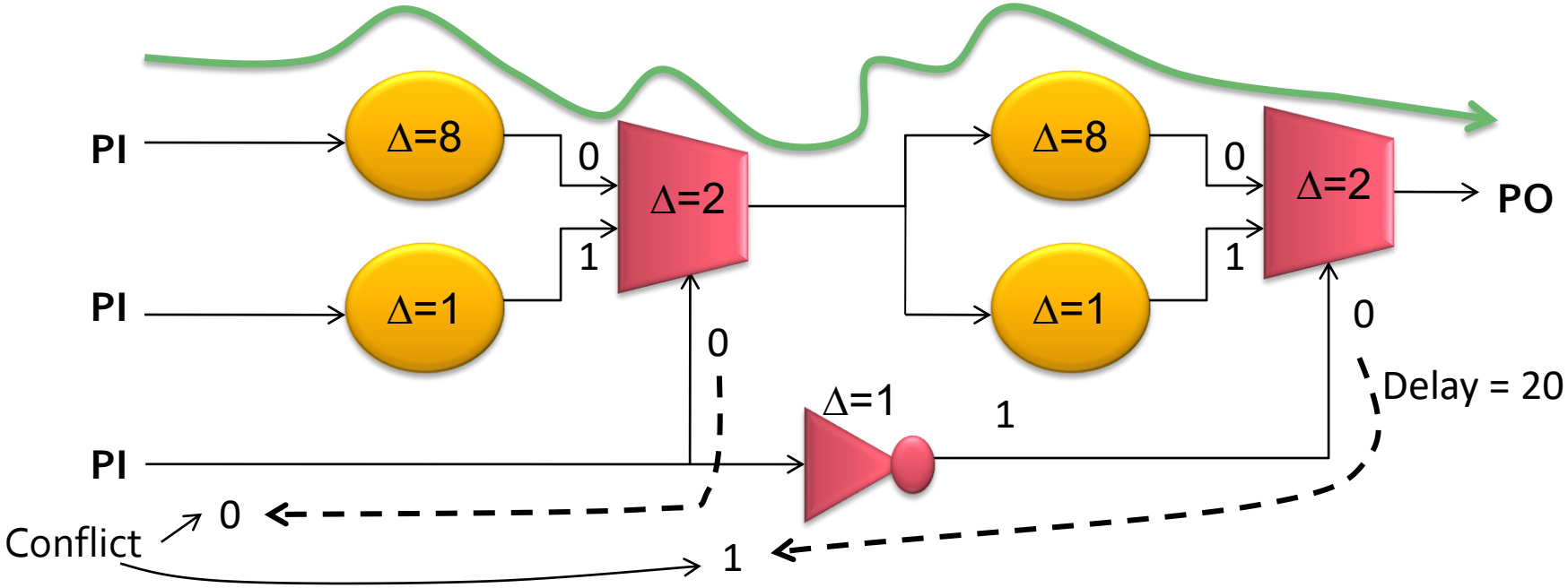
Topological Vs. Logical Timing Analysis

- Topological



Topological Vs. Logical Timing Analysis

- Logical – we tell what gates are

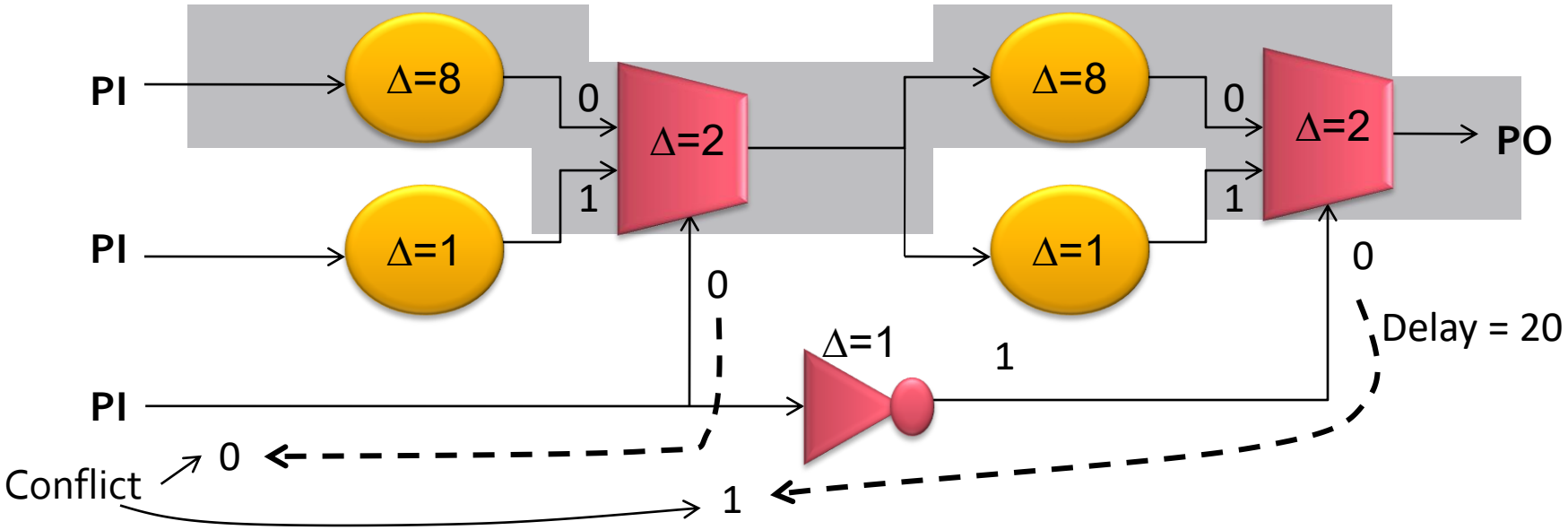


False Paths and Path Sensitization

- We got a false path
  - It is not possible to apply a set of inputs that will cause a logic signal to propagate down this supposed “longest” path from PI to PO
  - This path we found by topological analysis is called a FALSE PATH
  - We got this because we didn’t care what the gates did
- Sensitization
  - A path is said to be sensitized when it allows a logic signal to propagate along it.

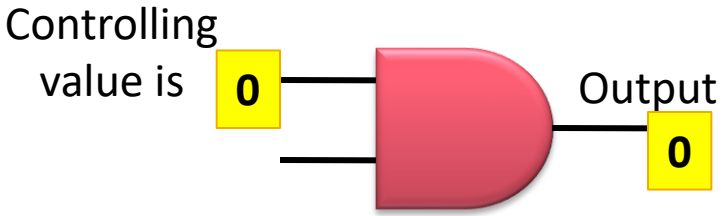
False Paths and Path Sensitization

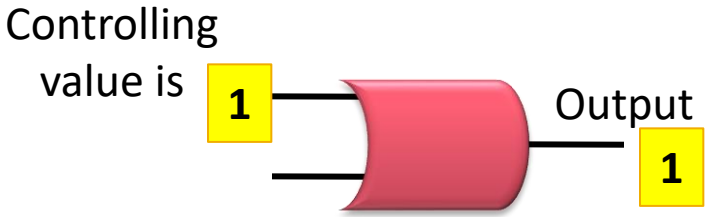
- In this example, there is no way to sensitize this path



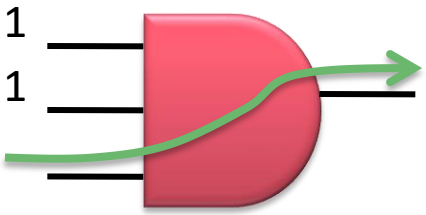
Sensitization

- Definitions
  - Controlling value for a gate is a single input value to a gate that uniquely forces the output to a known constant, independent of the other inputs to the gate.



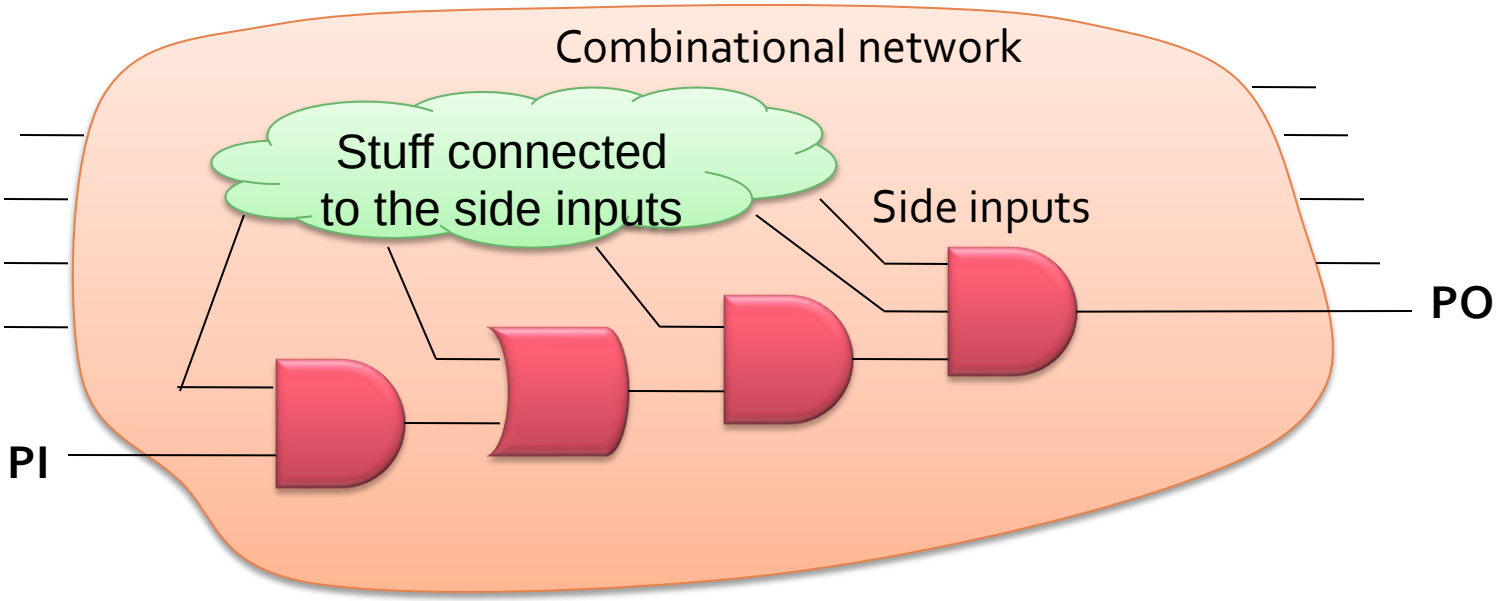


  - A gate is sensitized so a logic signal can propagate through it from one particular input to the output if the other inputs have stable noncontrolling values



Sensitization

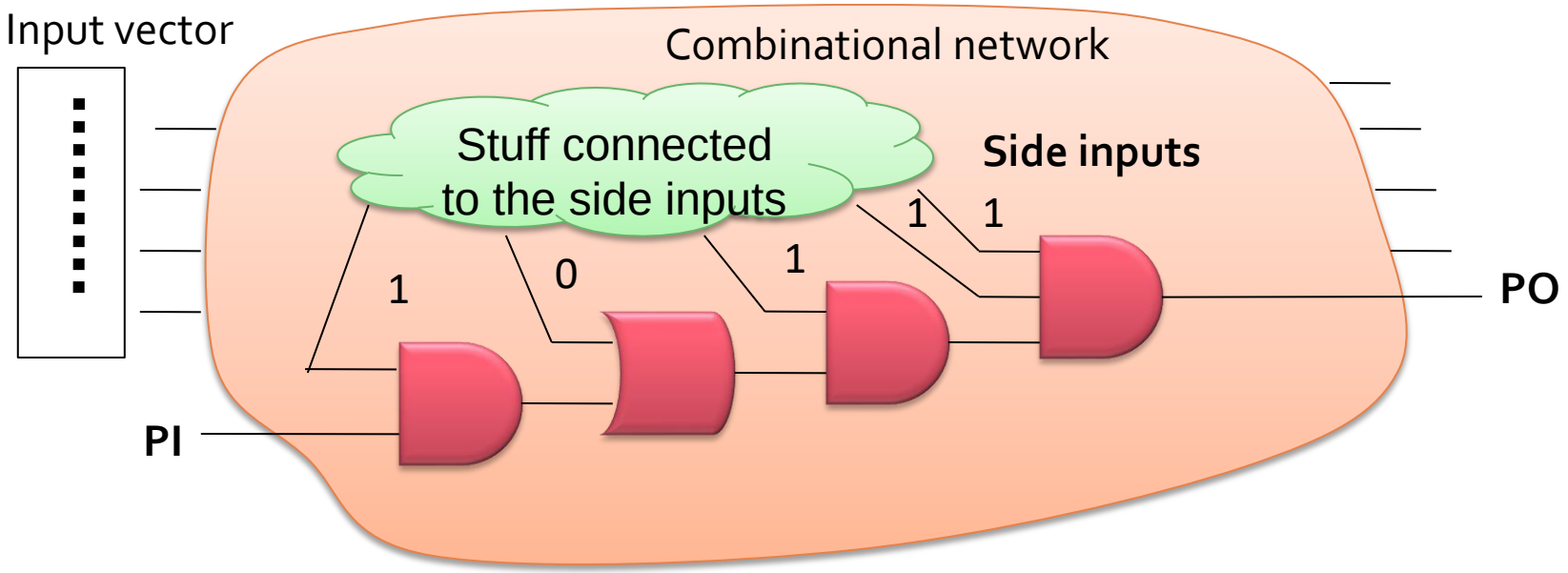
- Definitions
  - A path is a set of connected gates and wires that starts with some PI and ends with some PO. Path is defined by 1 input and 1 output per gate
  - Side inputs on a path are the “other” inputs to these gates on the path



Static Sensitization

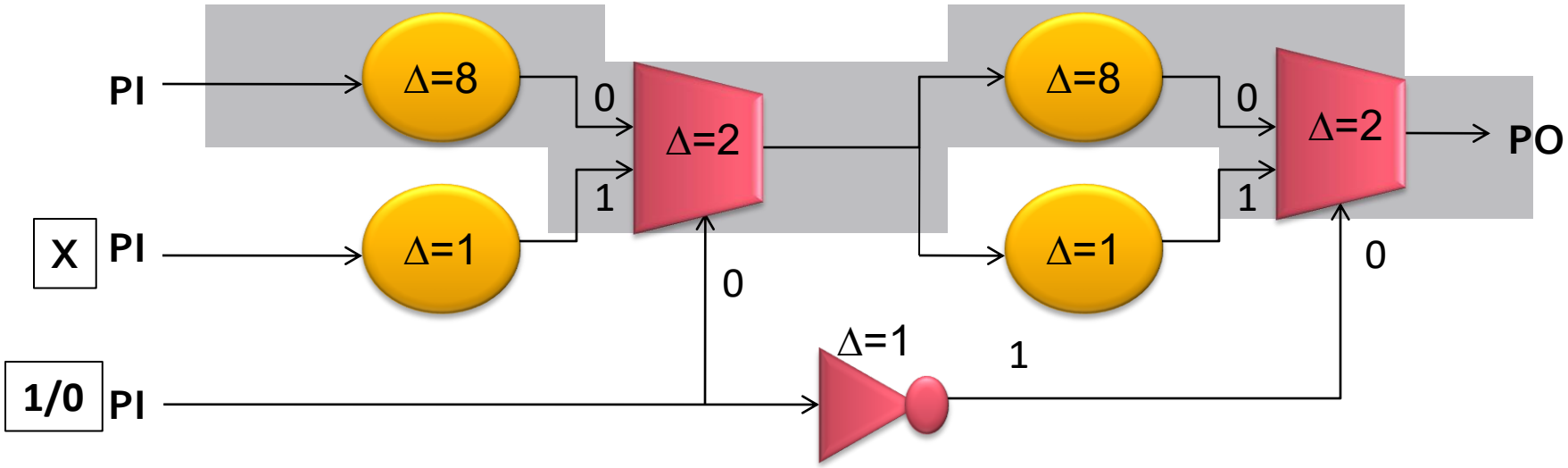
- A path is statically sensitizable when ..

There is an input vector which generates stable noncontrolling values to all side inputs on the path



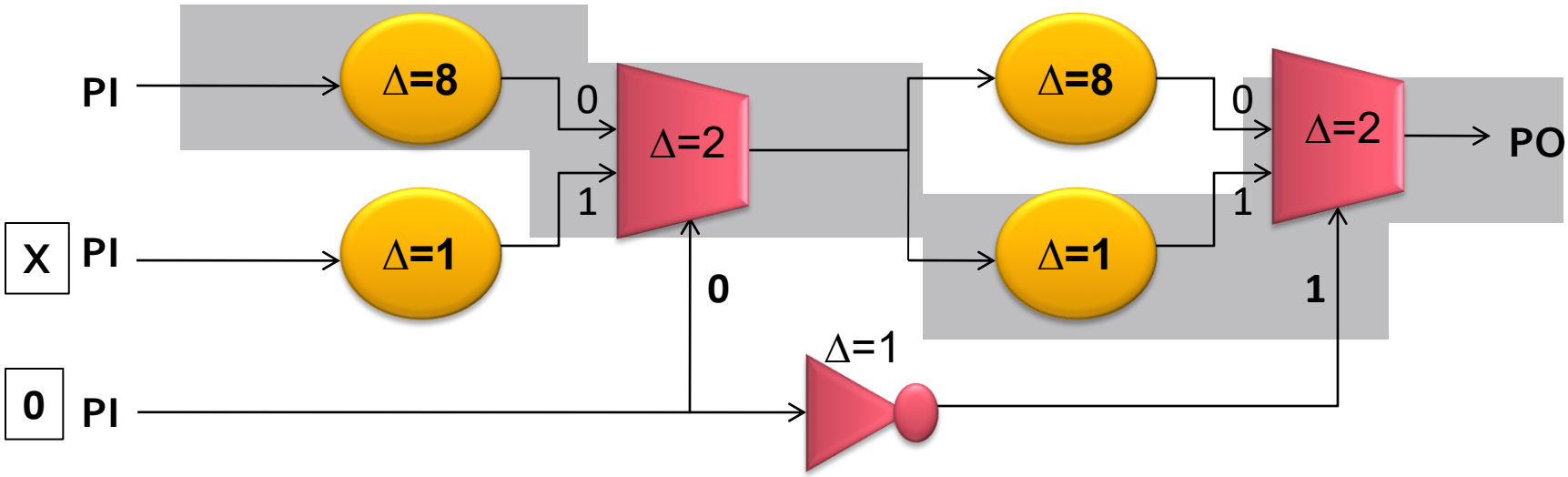
Static Sensitization

- NOT statically sensitizable



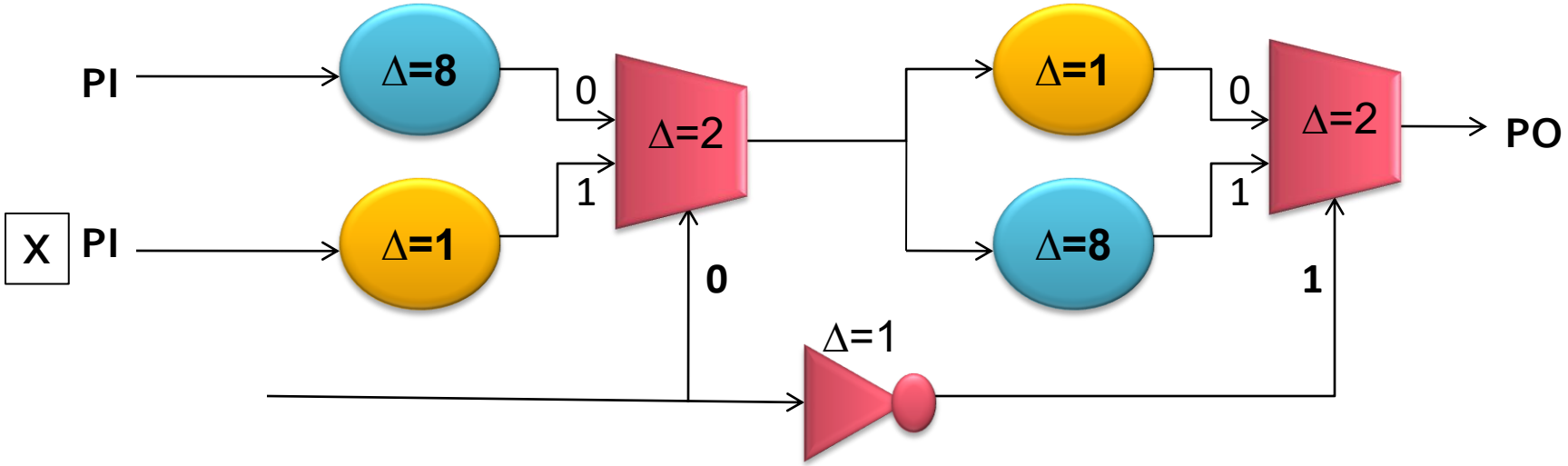
Static Sensitization

- Statically sensitizable



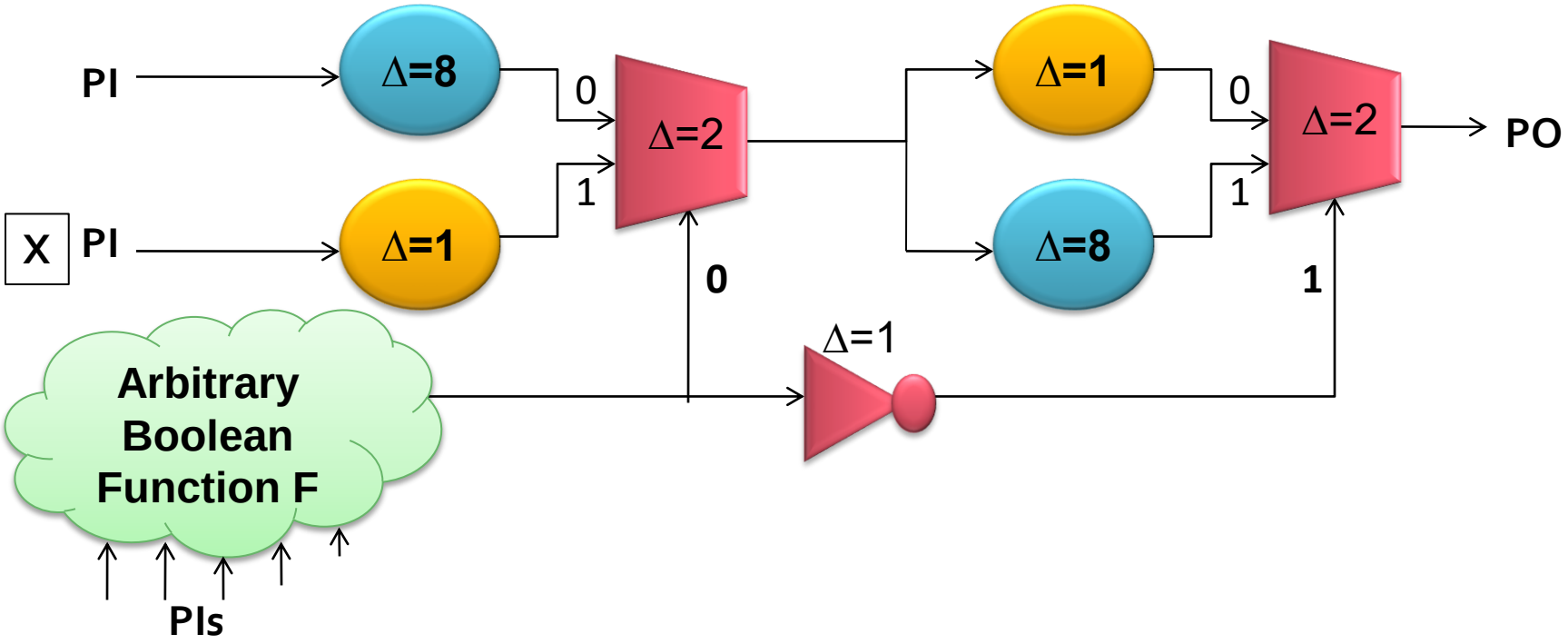
Sensitization

- How hard is it really to do this?
  - In general, very hard, through there are many good heuristics
  - As hard as Boolean satisfiability (find a pattern of inputs to make an arbitrary Boolean function == 1), which is NP hard
  - New example below:



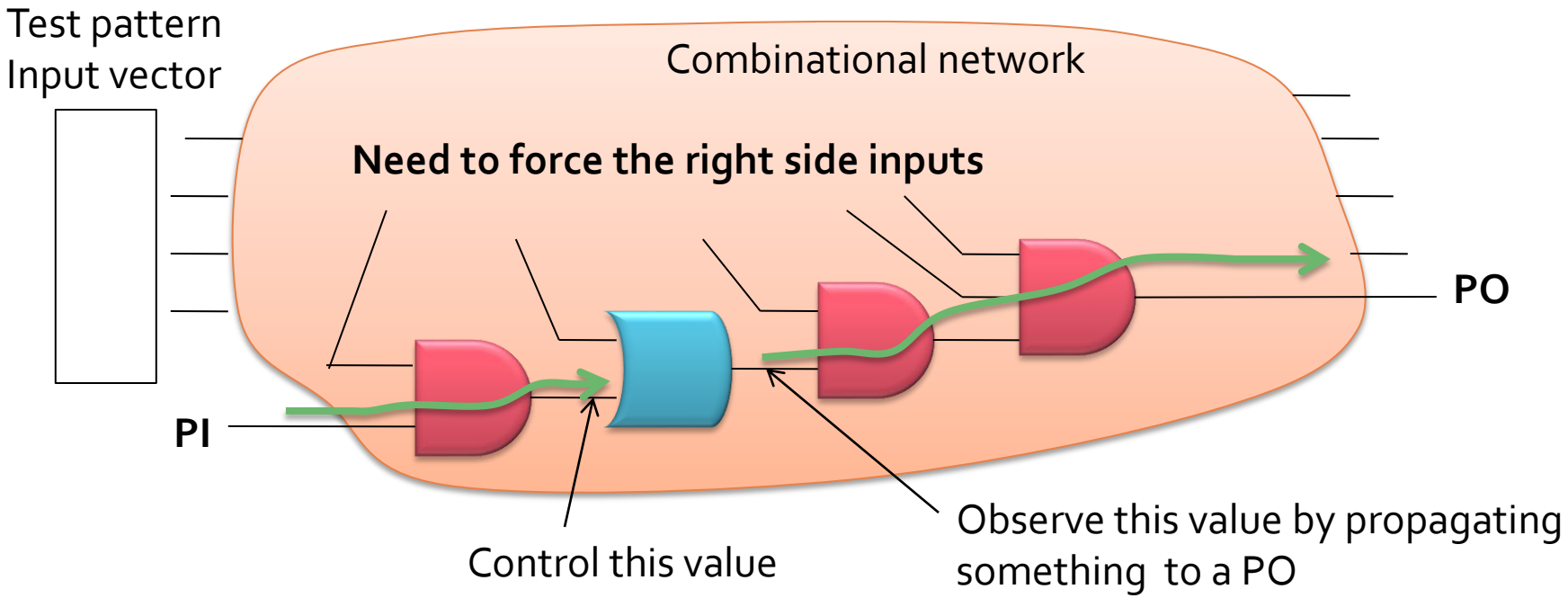
Sensitization

- How hard is it really to do this?
  - In general, very hard, through there are many good heuristics
  - As hard as Boolean satisfiability (find a pattern of inputs to make an arbitrary Boolean function == 1), which is NP hard
  - New example below:



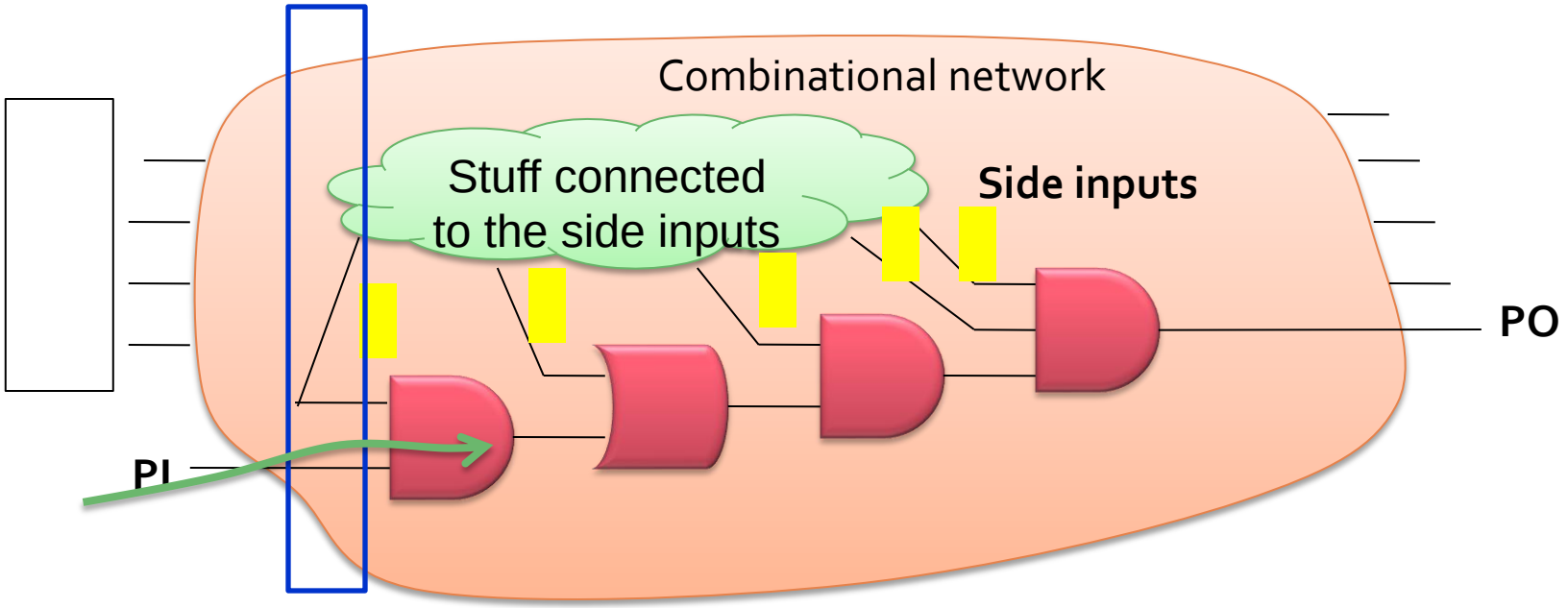
Aside: Related to Testing for Gate Level Circuits

- What's testing about
  - Find inputs to a gate network that force a particular value on a particular input of a particular gate
  - And that also allow the output of the gate to propagate to some output



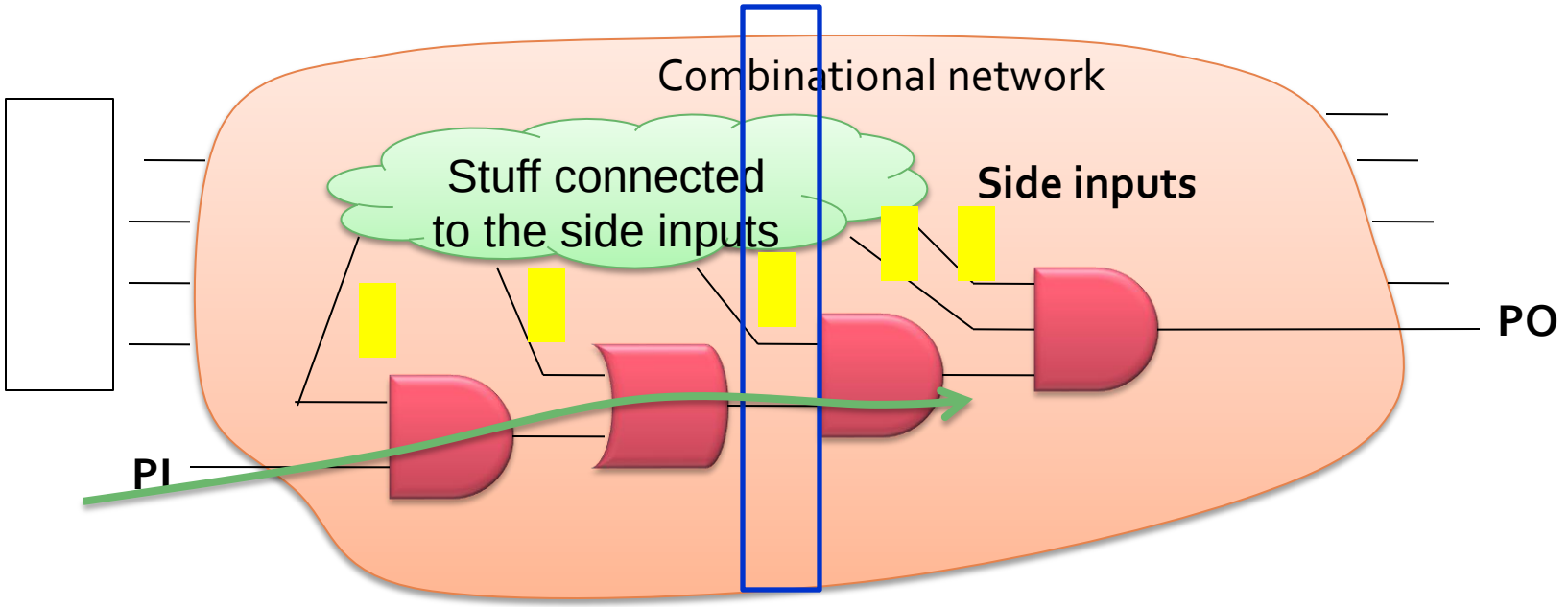
Beyond Static Sensitization..?

- Dynamic sensitization
  - Try to find vectors to apply at different times so that the right noncontrolling value appears at each side input when the propagating signal gets to that particular gate
  - hard to do



At time t0 need a 1 on this AND

Dynamic sensitization



At time t2 need a 1 on this AND

So what are we doing Here?

- Simple fixed delay gate model
  - No slopes, etc. any loading effects are “bundled” back into the gate delay number itself

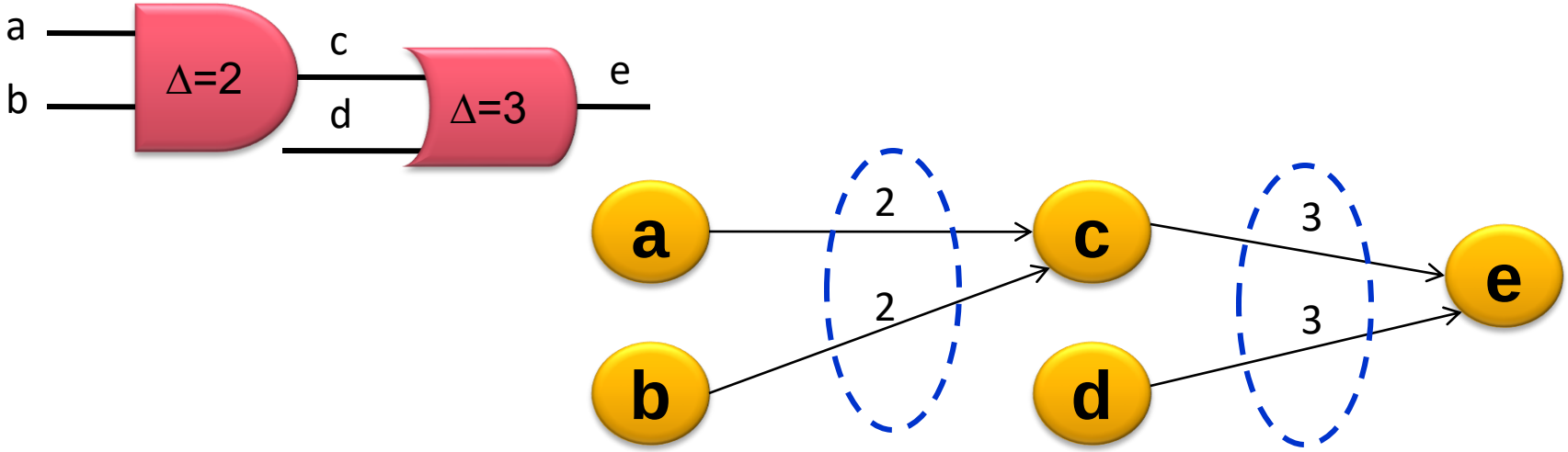
- Topological path analysis
  - We don’t worry about what the gates do
  - We only look at paths through the connected gates
  - Aside: means we assume all paths statically sensitizable
  - We know we will get false paths – too bad
  - This is usually a pessimistic timing model – delay numbers too big since we find false paths first that are usually overly long

Topological Path Analysis

- Generally what people mean by static timing analysis
- PRO
  - Fast (pattern independent)
  - Bounds true worst path delay
- CON
  - Can be pessimistic (includes false paths)

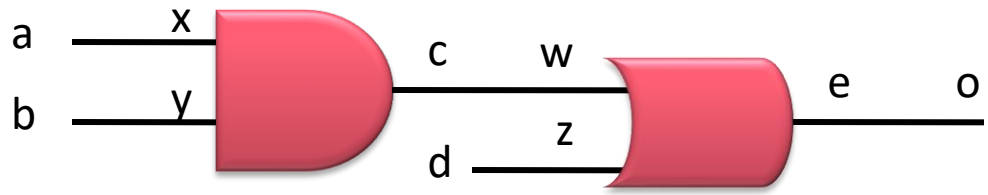
Representation of Delay Graph

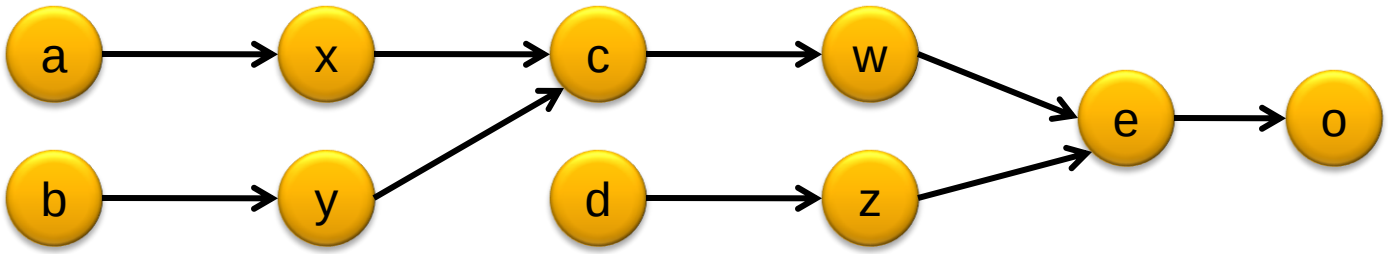
- How do we model gate network? Delay graph
  - Gates = edges, 1 edge per input pin
  - Numbers on edges = delay through gates
  - Wires (signals) = vertices. 1 per gate output
    - Also 1 for each PI, PO
    - Leave latches out for now
  - Note: this ends up as a directed, acyclic graph, a DAG



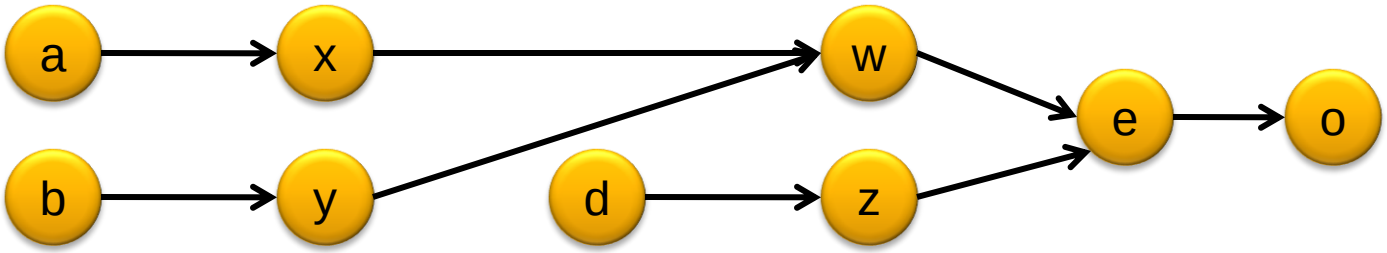
Representation: Delay Graph

- What about inter connect delay?
  - Can use delay graph with node for each pin instead of each net





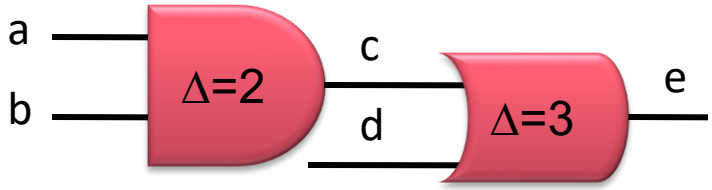
  - Gate and net delays interact – can have delay edge from input to input

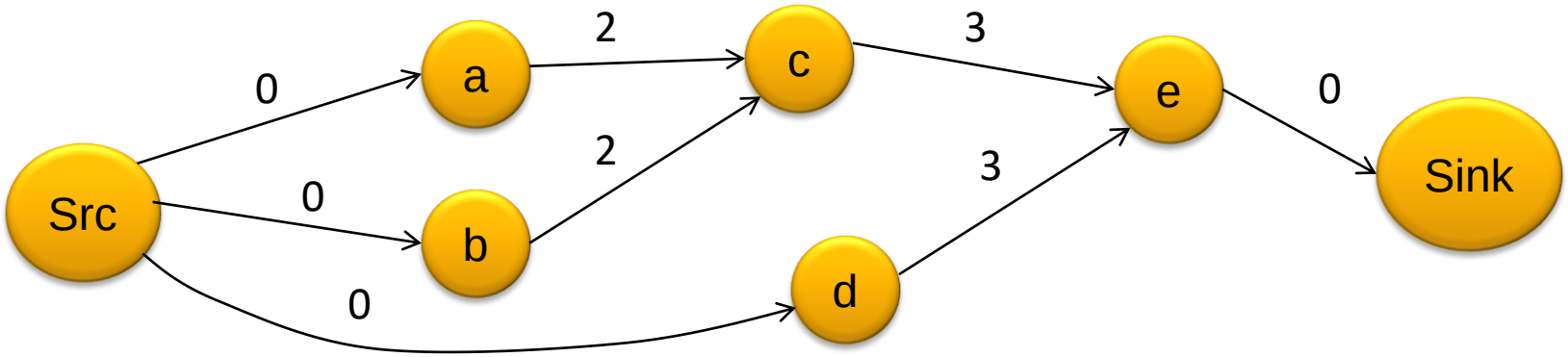


  - we will stick with one node per net for simplicity

Delay Graph

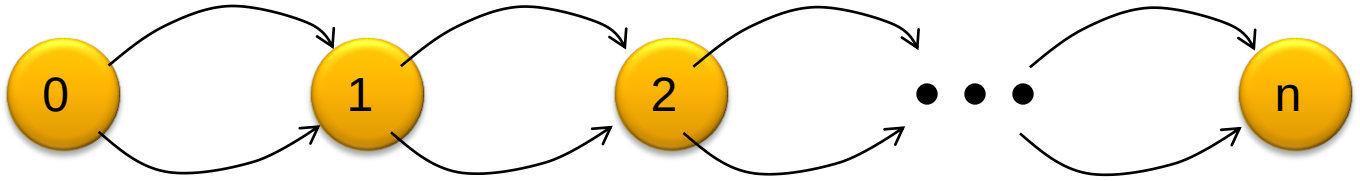
- Source/Sink nodes (pure combinational logic)
  - Often add 1 “source” node that has a 0 – weight edge to each PI
  - And 1 “sink” node with 0-weight edge from each PO
  - Now network has 1 clear “entry” node and 1 clear “exit” node
  - Even timers that don’t explicitly add these nodes do something similar
    - Loop through all PIs (POs) ←→ loop through fanout (fanin) of source (sink) node





Path Enumeration

- Problem is number of paths
  - Can be exponential in length of paths
  - Our “search” algorithm doesn’t visit paths in any useful order



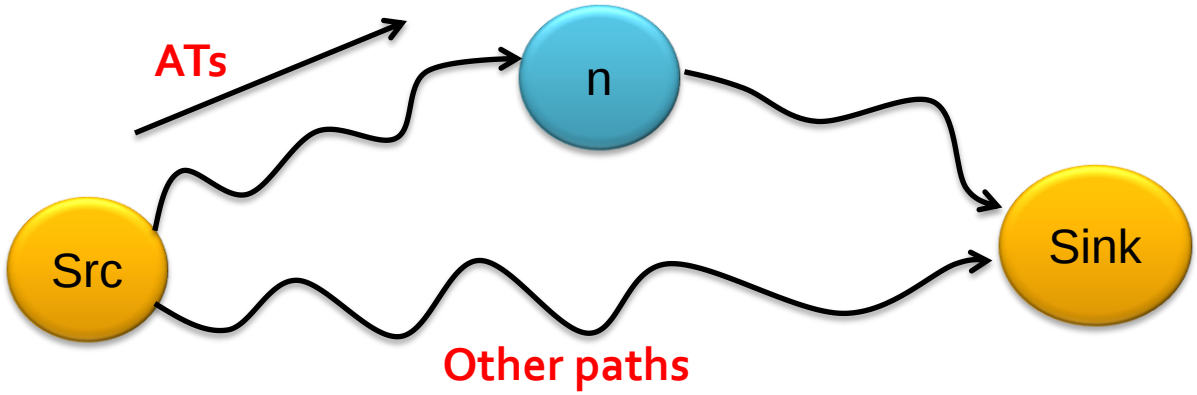
  - How many paths from node 0 to node n in here? 2^n
  - Some timing analyzers do this away
    - May use pruning methods to control exponential behavior

Operations on Delay Graph

- Instead we'll use what's been called block-oriented analysis
  - Don't look for paths to the sink (primary outputs)
  - Instead find for each node the worst delay to the node along any path

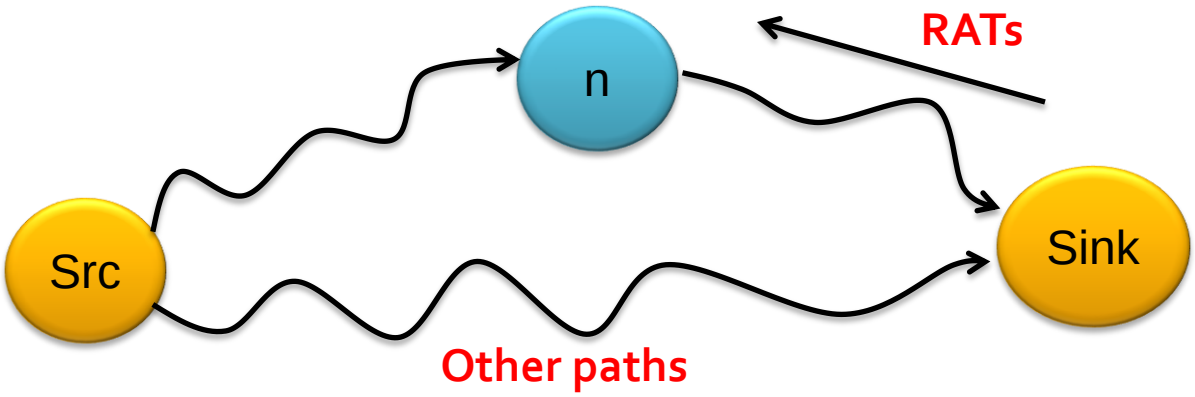
Values on Nodes in Delay Graph

- Arrival Times at a node (ATs)
  - $AT_E(n)$ = Earliest signal can become unstable at node n
    - Determined by shortest path from source
  - $AT_L(n)$ = Latest time signal can become stable at node n
    - Determined by longest path from source
  - Sometimes called “delay to node”



Values on Nodes in Delay Graph

- Required Arrival Times at a node (RATs)
  - $RAT_E(n)$ = Earliest that signal is allowed to become unstable at node n
    - Determined by shortest path to sink
  - $RAT_L(n)$ = Latest time signal is allowed to become stable at node n
    - Determined by longest path to sink
  - Related to what is sometimes called “delay from node”



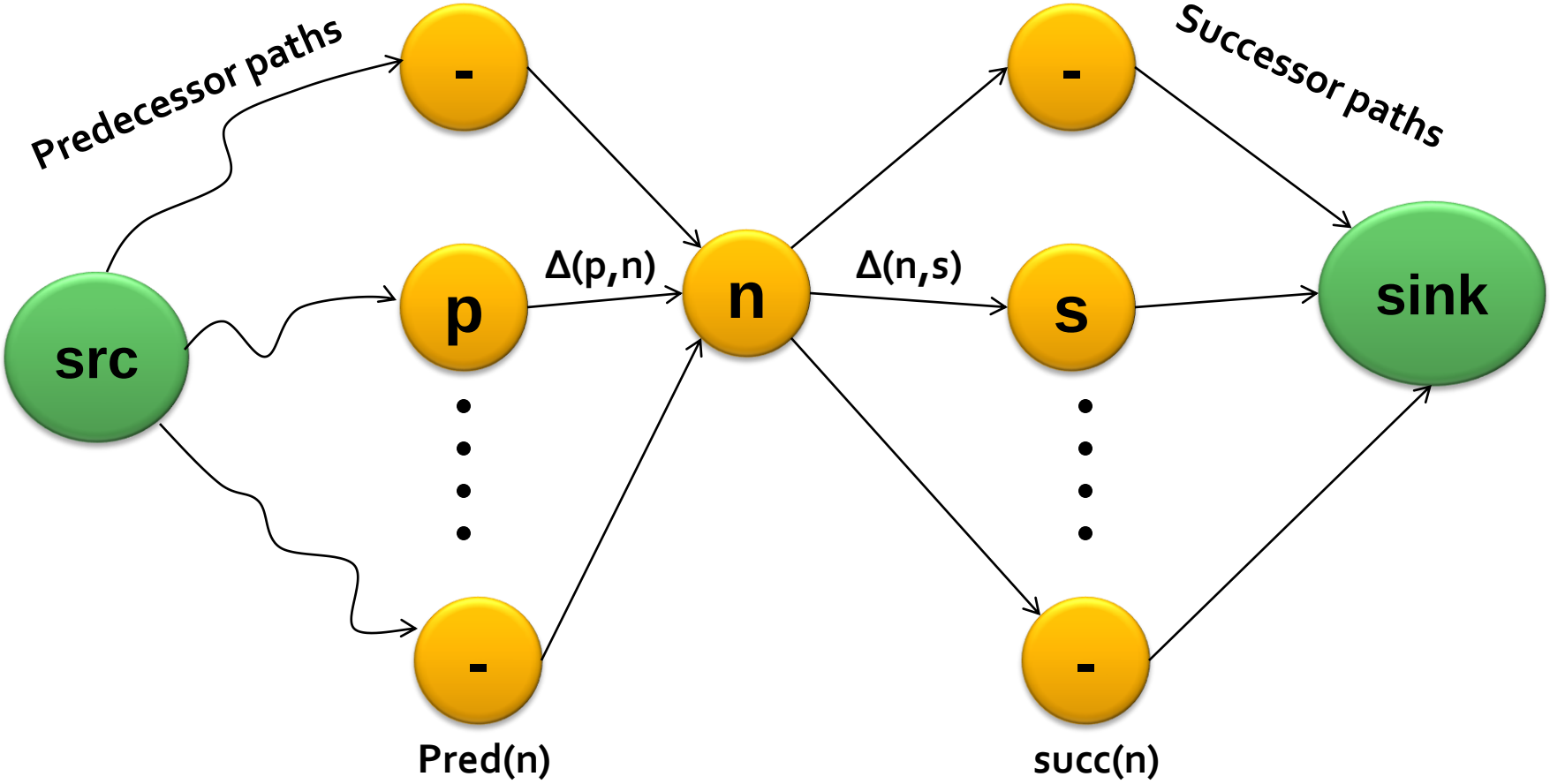
Values on Nodes in Delay Graph

- Slacks at a node
  - $Slack_E(n) = AT_E(n) - RAT_E(n)$
    - Amount of margin in time signal goes unstable
    - Determined by shortest path through node
    - Amount by which a signal can be sped up at a node and not decrease the length of the shortest path through the network

Values on Nodes in Delay Graph

- $Slack_L(n) = RAT_L(n) - AT_L(n)$
  - Amount of margin in time signal becomes stable
  - Determined by longest path through node
  - Amount by longest path through node
  - Amount by which a signal can be delayed at a node and not increase the length of the longest path through the network
  - Can increase delay at a node (to minimize power, circuit area) with positive late mode slack and not degrade overall performance
- Defined so negative slack always indicates a timing problem
- Measure sensitivity of network to this node's delay

How To Compute?

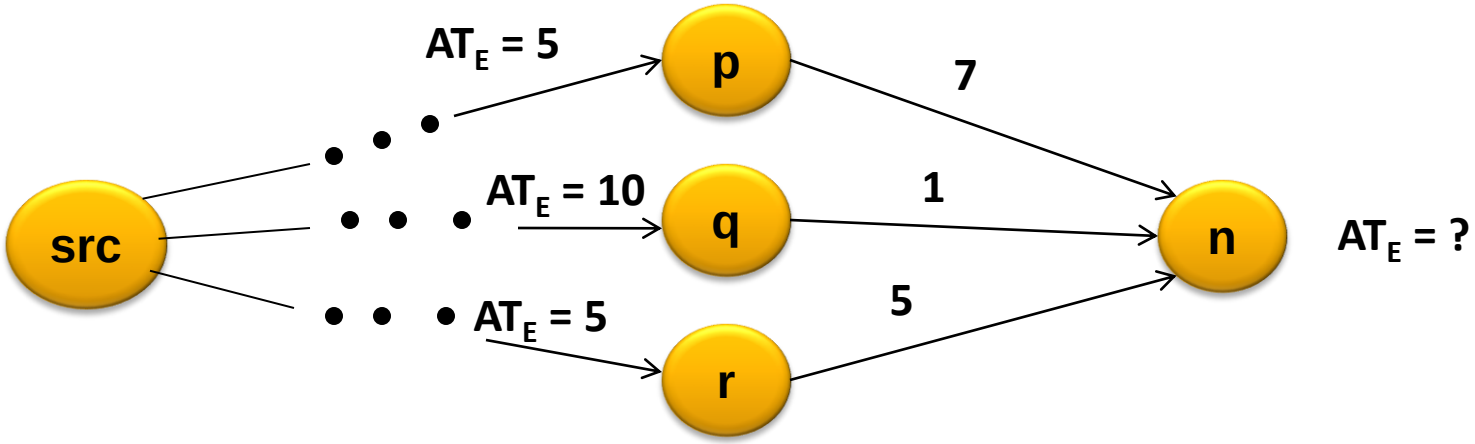


Arrival Times for a Node n

$$AT_E(n) = \min\ delay\ \ to\ n = \begin{cases} 0\ if\ n == src \\ \underset{p=pred(n)}{Min}\{AT_E\ (p)\ + \delta(p,n)\ \} \end{cases}$$

$$AT_L(n) = \min\ delay\ \ to\ n = \begin{cases} 0\ if\ n == src \\ \underset{p=pred(n)}{Min}\{AT_L\ (p)\ + \Delta(p,n)\ \} \end{cases}$$

Example

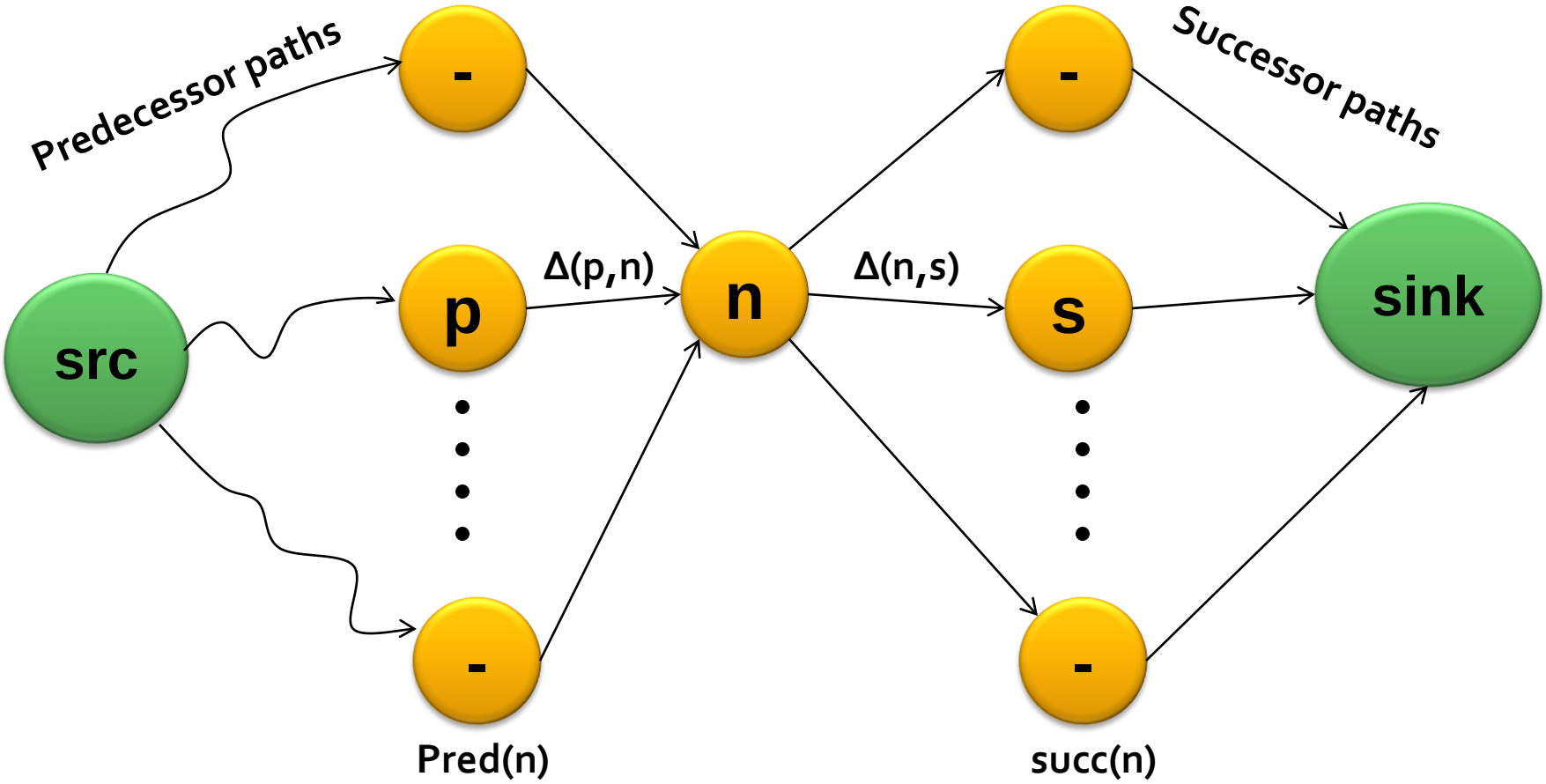


$$AT_E(n) = \min\ delay\ \ to\ n = \begin{cases} 0\ if\ n == src \\ \underset{p=pred(n)}{Min}\{AT_E\ (p)\ + \delta(p,n)\ \} \end{cases}$$

= Min (5+7, 10+1, 5+5)

= 10

Required Arrival Times for a None n

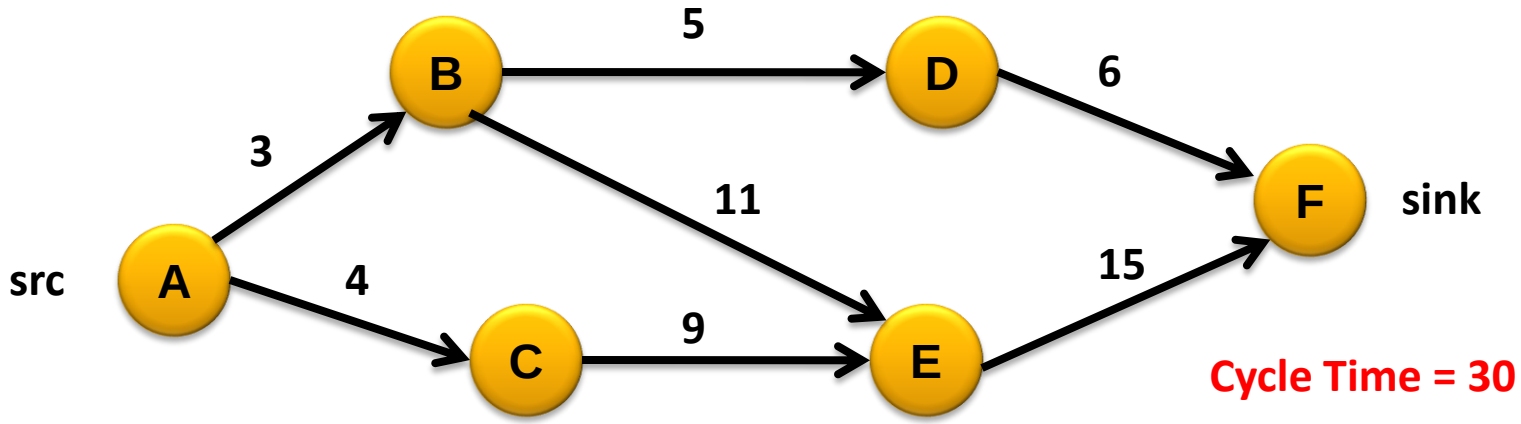


Required Arrival Times for a None n

$$RAT_E(n) = \begin{cases} 0 \; if \; n == sink \\ \underset{s=succ(n)}{Max}\{RAT_E(s) - \delta(n,s)\} \end{cases}$$

$$RAT_L(n) = \begin{cases} Cycle\ time \; if \; n == sink \\ \underset{s=succ(n)}{Min}\{RAT_L(s) - \Delta(n,s)\} \end{cases}$$

Example



- $AT_E(E) = 4+9 = 13$
- $AT_L(E) = 3+11 = 14$
- $RAT_E(B) = 0 – 5 – 6 = -11$
- $RAT_L(B) = 30 – 11 – 15 = 4$
- $Slack_E(B) = 3-(-11) = 14$
- $Slack_L(B) = 4 – 3 = 1$

Slack

- Interesting slack property
  - All nodes on a critical (longest) path have same slack
  - Consider a late mode analysis