ECE6703J

Computer-Aided Design of Integrated Circuits

Timing Analysis

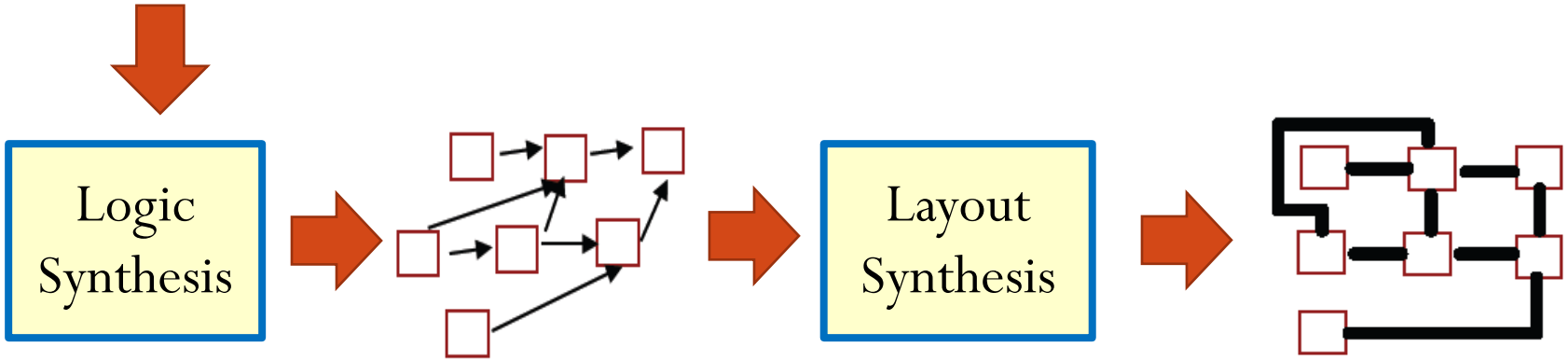
Outline

- Timing Analysis: Basics, Assumptions, and Models
- Static Timing Analysis: Basic Idea

ASIC Timing: Role of CAD Tools

- ASIC timing has **deep interactions** with logic and layout synthesis.

High-level description
+ **Timing Specifications**

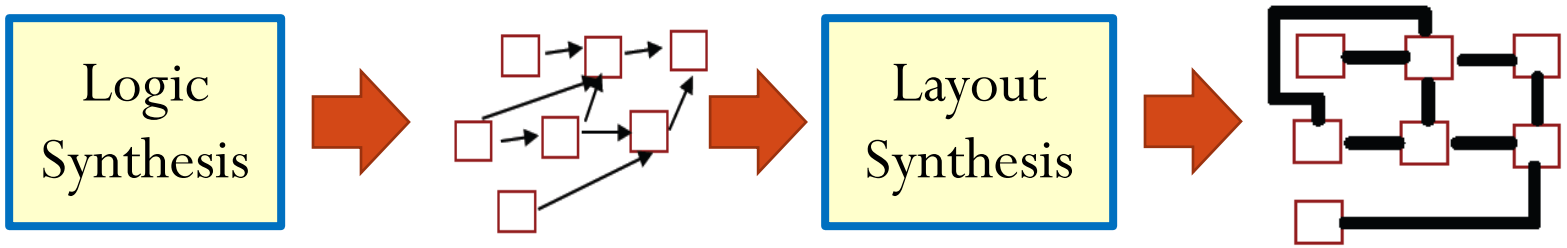


Connected cells with delay constraints on signal paths

Placed cells with real locations, real connecting wires

ASIC Timing: Role of CAD Tools

- Requirement on timing analysis
  - Logic-side tools must **estimate delays** through unplaced/unrouted logic.
  - Layout-side tools must **estimate delays** through placed/routed logic.



Our Topics for ASIC Timing

- Logic-side: **Static Timing Analysis**
  - How do we estimate the worst-case timing through a logic network?
  - Turns out to be **longest paths** through a graph that properly models the gates and wires.

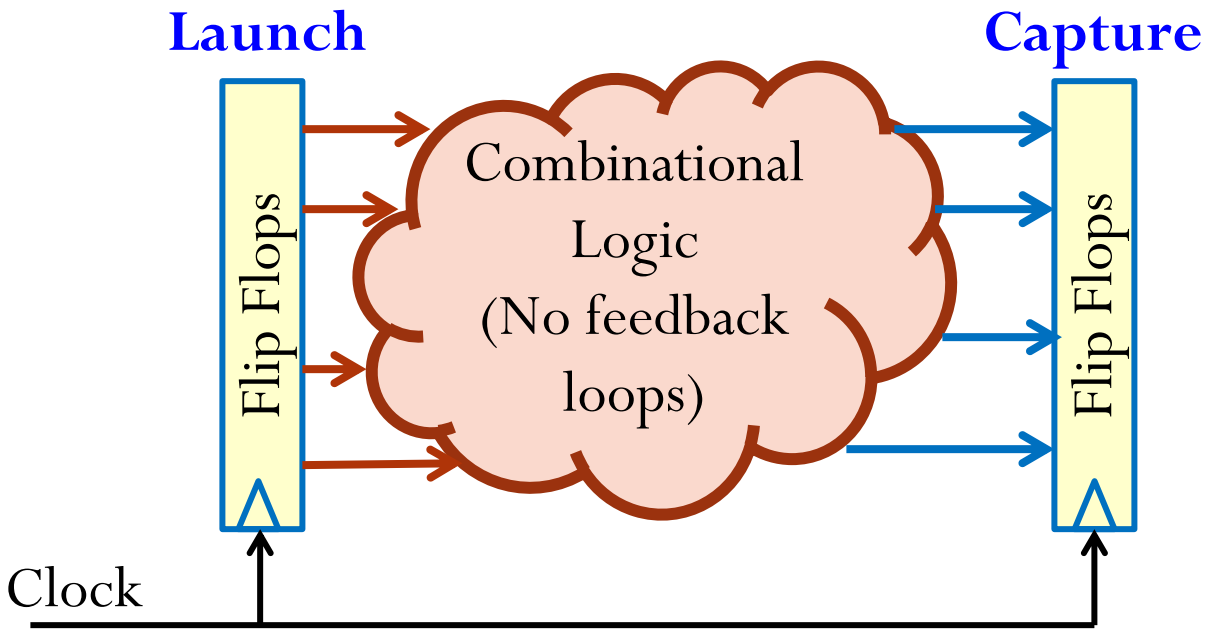
- Layout-side: **Interconnect Delay Analysis**
  - We place the gates, route the wires. Then, how do we estimate **wire delays**?
  - The problem is built up on electrical circuit model. We will show key methods.

Timing Analysis at the Logic Level

- Goal: Verify timing behavior of our logic design
- Input:
  - A gate-level netlist.
  - **Timing models** of the gates and/or wires.
- Output:
  - **Signal arrival time** at various points in the network.
  - **Longest delays** through gate network.
  - Does the netlist **satisfy** the timing requirement? If not, where are the key problems?
- This is surprisingly complicated in the real world…

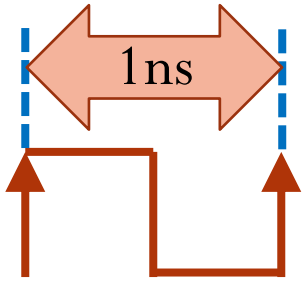
Assumption: Design Is Synchronous

- All storage is in explicit sequential elements, e.g., flip-flop elements.
- Consequence: we can just focus on delays through combinational gates.

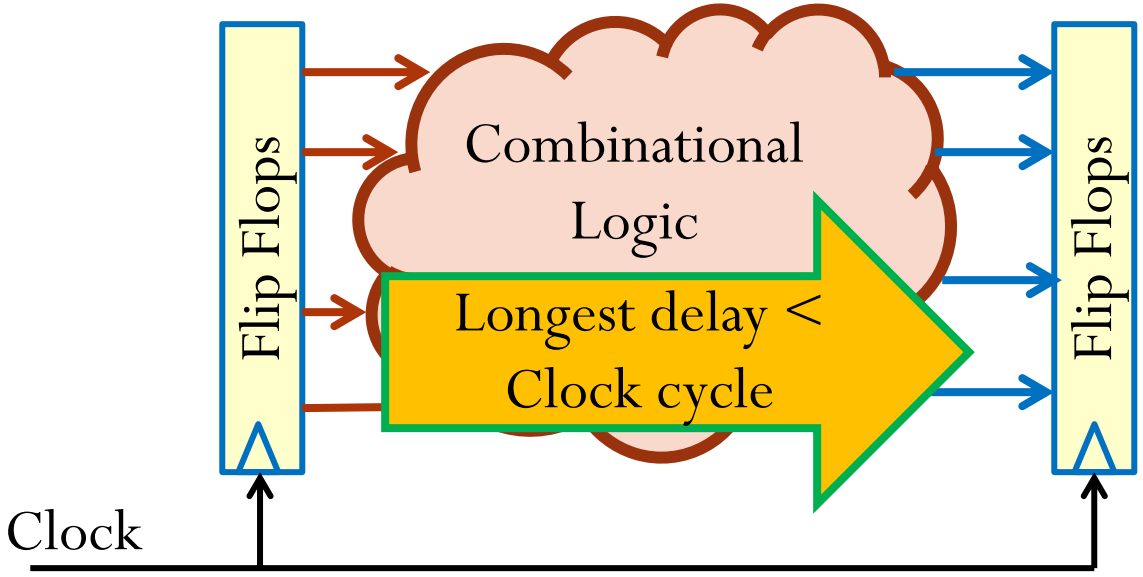


Timing Analysis: Basic Model

- Assume we know clock cycle
  - E.g., 1GHz clock, cycle = 1ns.



- For logic to work **correctly**, **longest delay** through network must **be shorter than the clock cycle**.

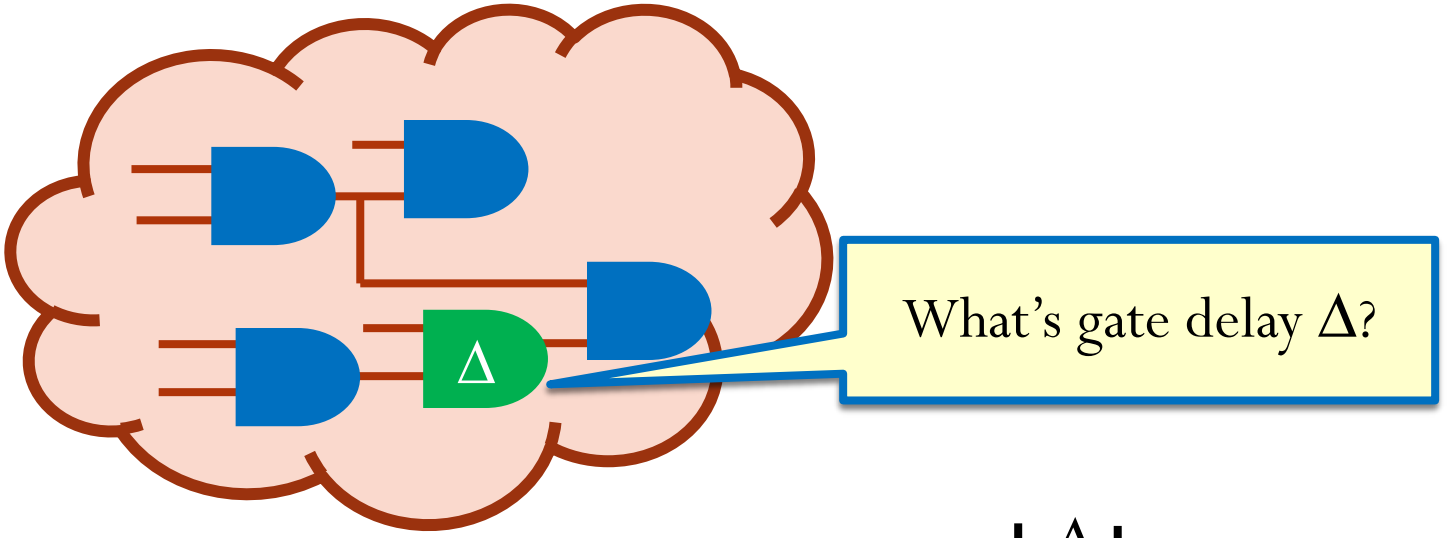


Question: Can't We Just Simulate Logic?

- What logic simulation does?
  - Determines how a system will behave by simulating the netlist.
  - Gives the most **accurate** answer with good simulation models.
  - … but it is (practically) impossible to give a complete answer – especially timing.
- Requires examination of an **exponential** number of cases.
  - All possible input vectors …
  - With all possible relative timings …
- We need a **different**, **faster** solution…

Timing Analysis: Gate Delay Models

- First: we need a model of delay through each logic gate.

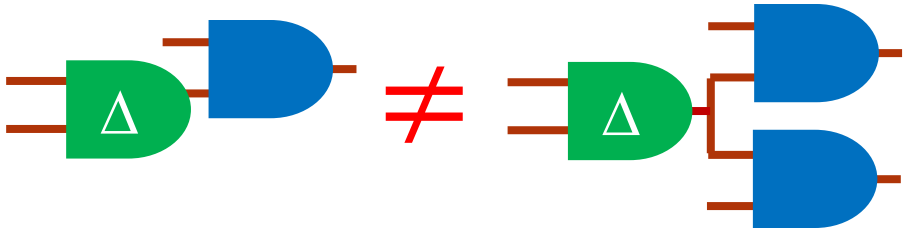


- Delay of a single gate:

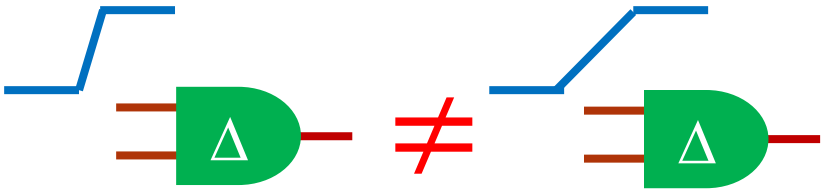
In Reality: Gate Delay is Very Complex

- Gate **type** affects delay

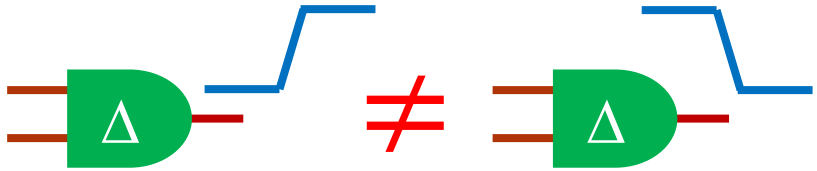
- Gate **loading** affects delay



- **Waveform shape** affects delay

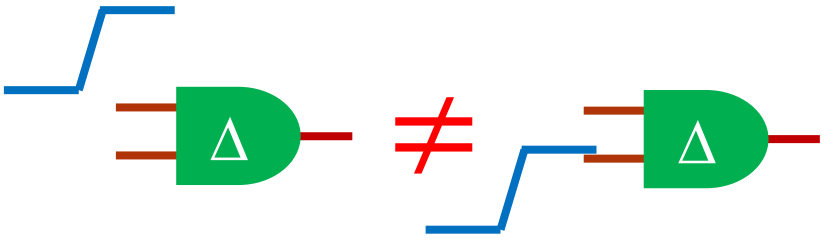


- **Transition direction** affects delay

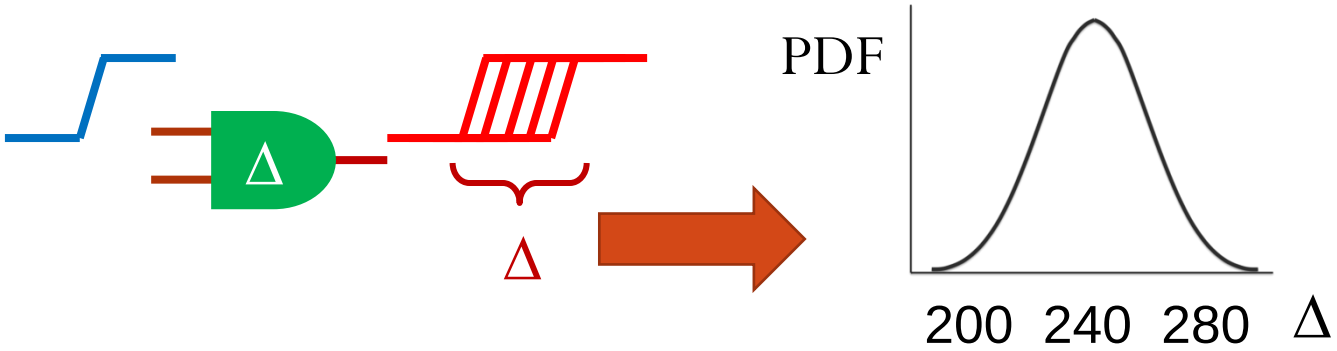


In Reality: Gate Delay is Very Complex

- Gate **input pin** affects delay
  - Why? At transistor level, inputs are not symmetric.

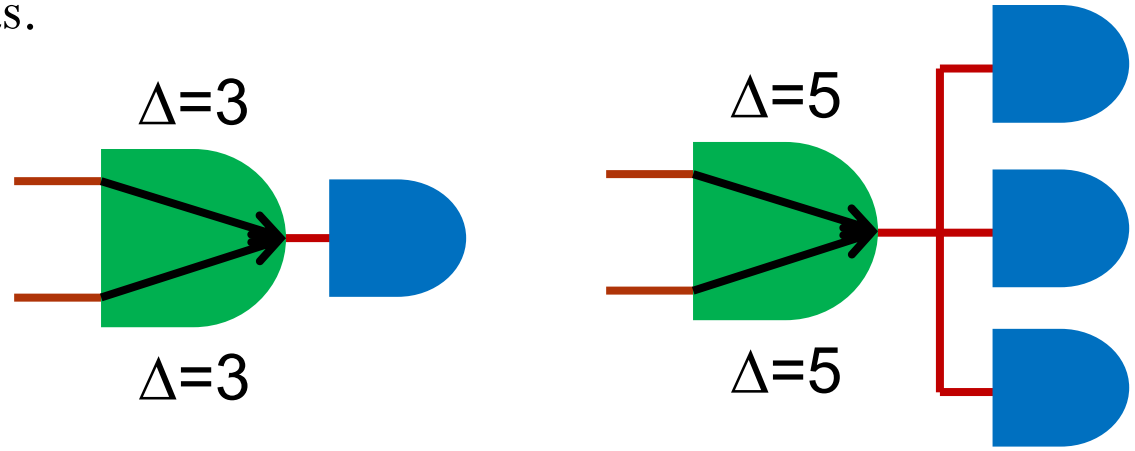


- At nanoscale, delays are even **statistical**
  - Why? Depends on process, voltage, and thermal (PVT) variations.



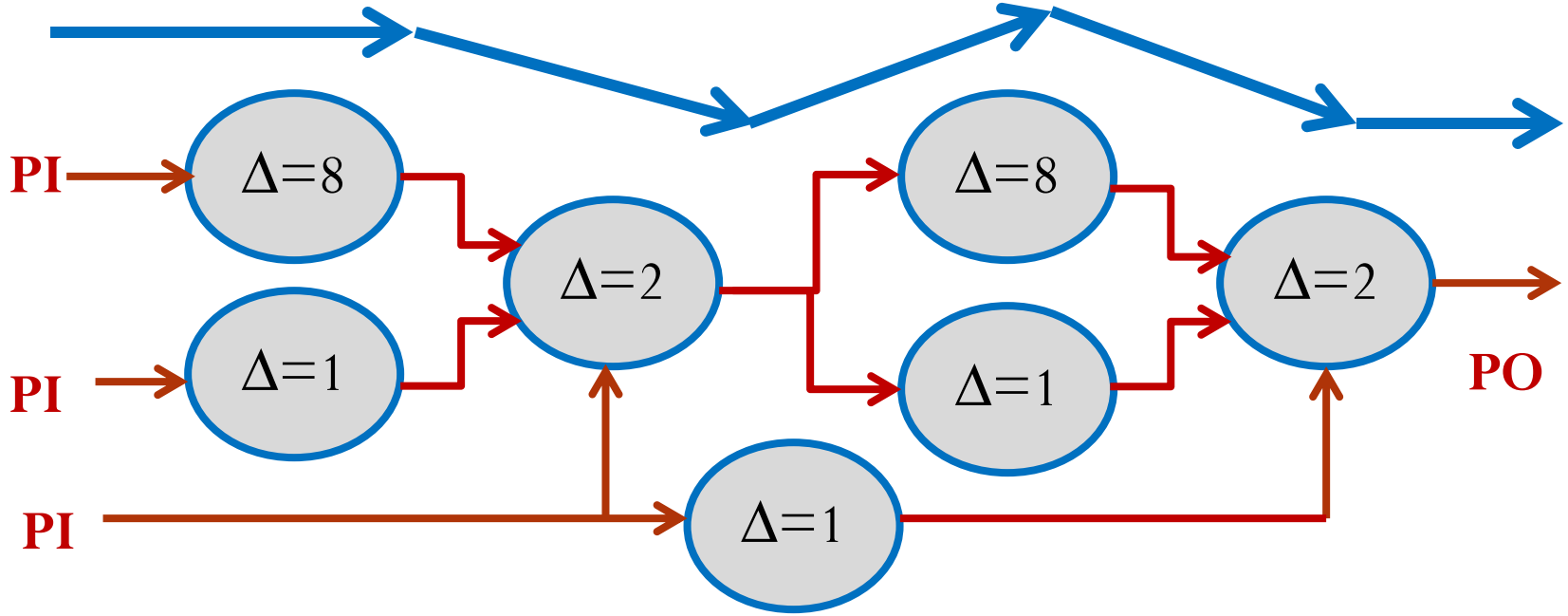
Our Model: Pin-to-Pin Delay

- In our lecture, we keep it simple: **Fixed, pin-to-pin delay model**
  - No waveform, transition direction, distributions. Loading effects "pushed" into gate delay itself.
  - Per-pin delays are essential, but we will use **just 1 value per gate**, for simplicity.
  - Turns out this is enough to see all the interesting algorithm ideas.



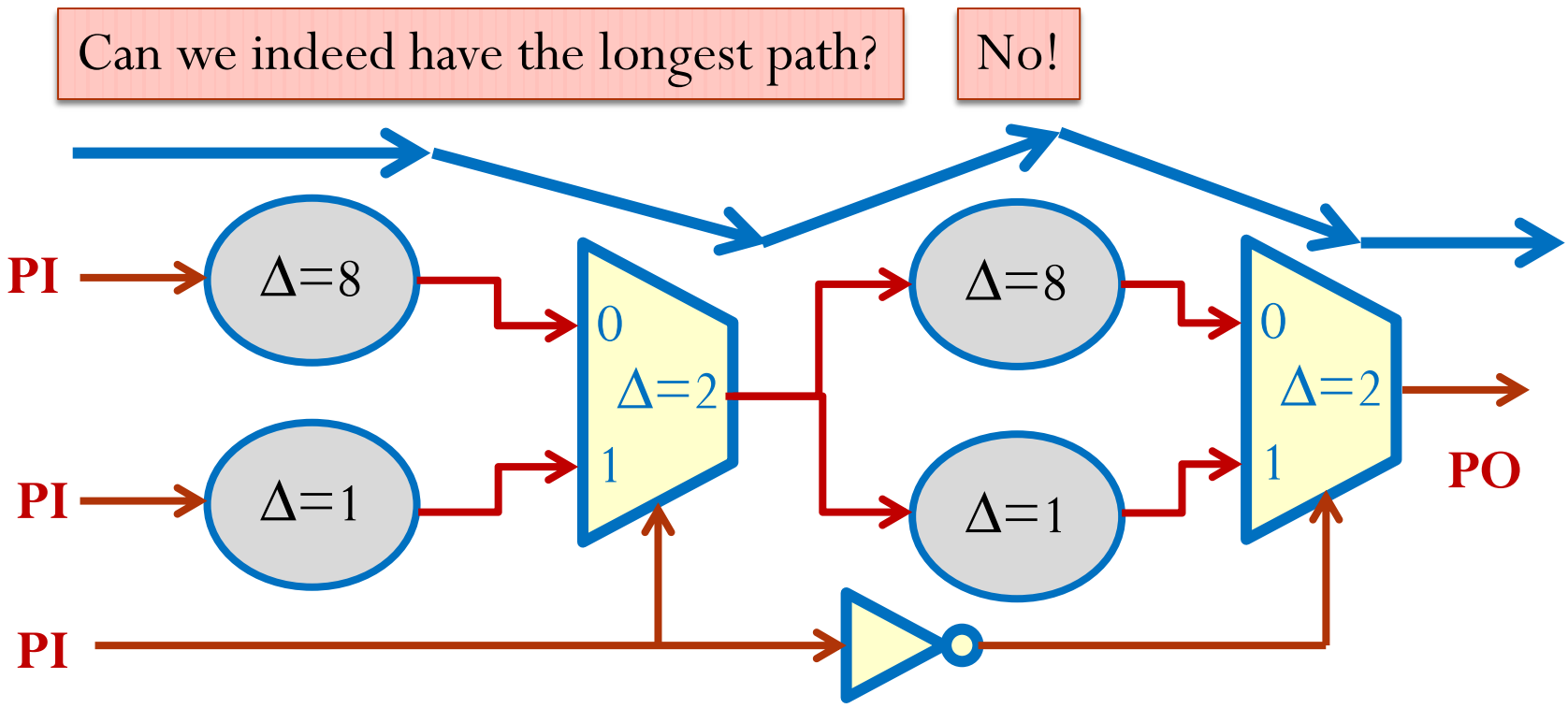
Do We Consider Logical Function?

- Does logic function matter?
- Try an example, where we "erase" gates.
  - In this example: **PI** = Primary Input, **PO** = Primary Output



What is the longest delay? 20

Now, Suppose We Know Logic Gates



- We cannot **sensitize** this path: cannot make a logic change at this input propagate down this path to change this output.

Topological vs Logical Timing Analysis

- When we ignore logic, this is called **Topological Analysis**.
  - We only work with graph and delays, **don't** consider logic.
  - We can get wrong answers: what we found was called a **False Path**.
  - This gives an upper bound
- Going forward: we **ignore logic** (Too tough to deal with)
  - Assume that all paths are **sensitizable**.

- This timing analysis has a name: **Static Timing Analysis (STA)**.

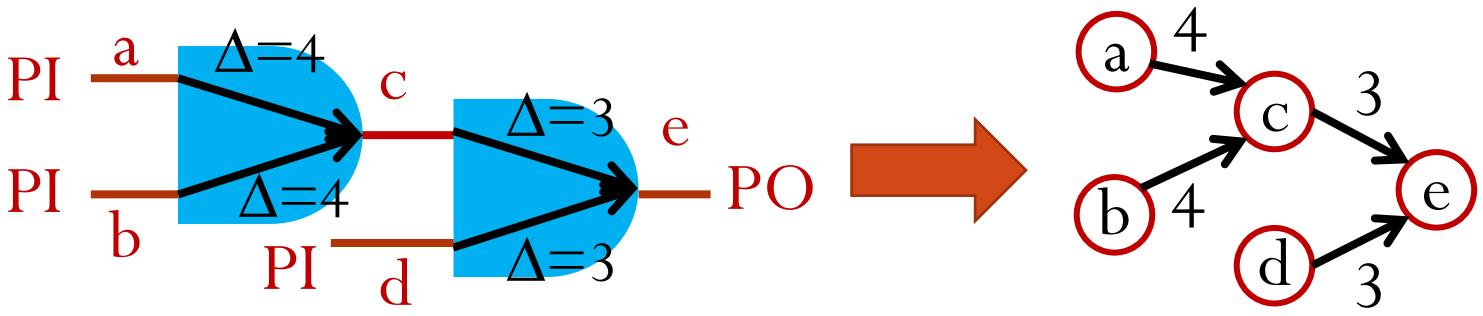
Outline

- Timing Analysis: Basics, Assumptions, and Models

- Static Timing Analysis: Basic Idea

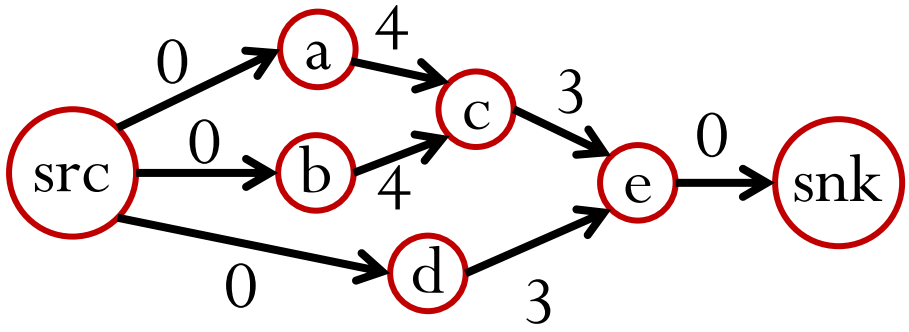
STA Representation: Delay Graph

- From gate-level network, we build a **delay graph**.
  - **Vertices**: **Wires** in gate network, one per gate output, also one for each PI.
  - **Edges**: Input pin to output pin of **gate** in network (one edge per input pin). Put gate delays on edges.



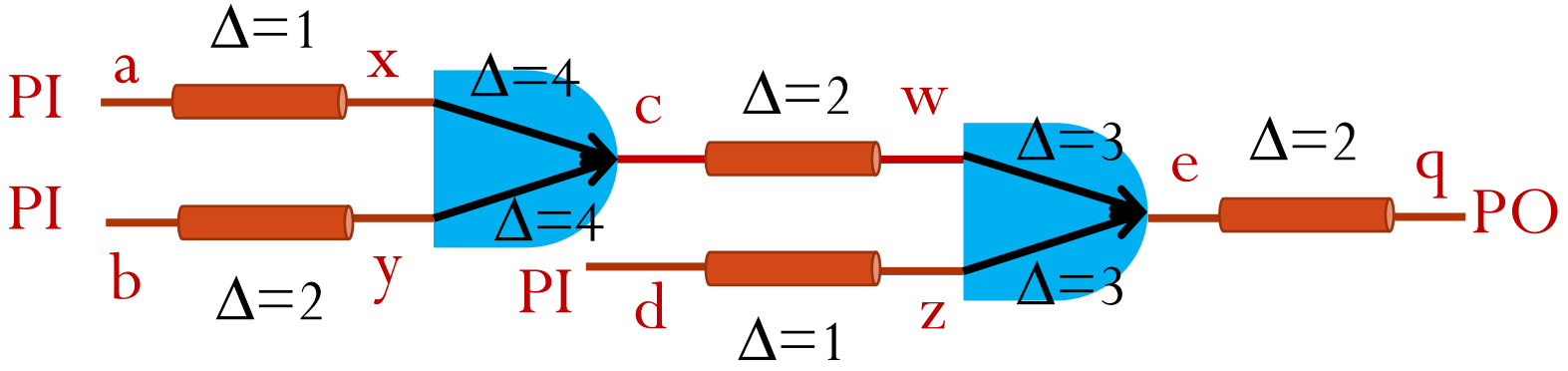
Delay Graph

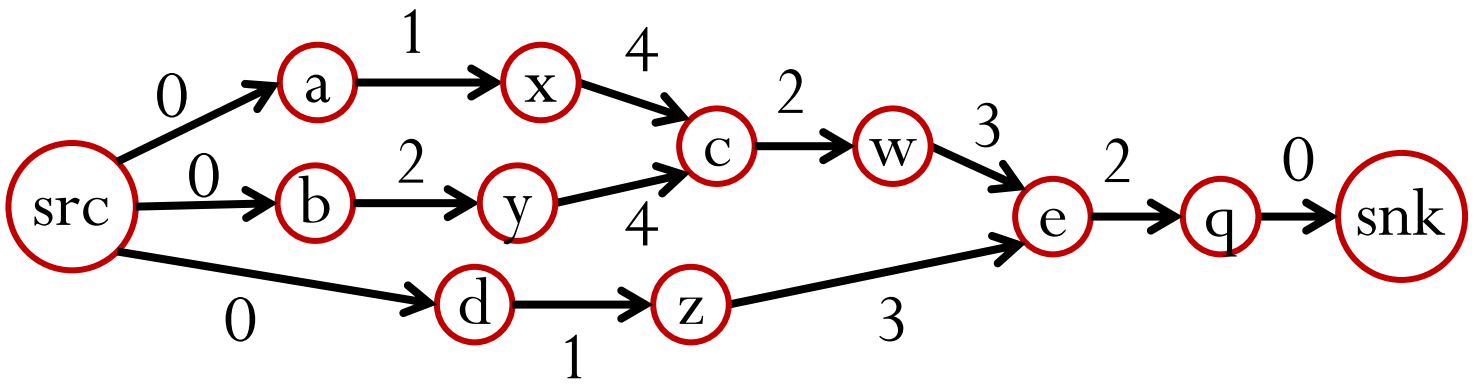
- Common convention: Add **Source/Sink** nodes
  - Add one "**source**" (**src**) node that has a **0-weight edge to each PI**.
  - Add one "**sink**" (**snk**) node that has a **0-weight edge from each PO**.
- Why do this?
  - Now, the network has exactly **1** "entry" node, and **1** "exit" node.
  - All the longest (or shortest) path questions have same start/end nodes.



Delay Graph

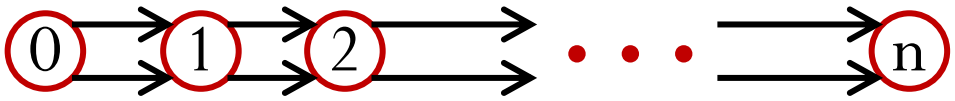
- What about **interconnect** delay?
  - Can still use delay graph: model each wire as a "special" gate that just has a **delay**.





Operations on Delay Graph

- So how do we use delay graph to do **timing analysis**?
  - What we **don't** do: Try to enumerate all the source-to-sink paths.
  - Why not? **Exponential** explosion in number of paths, even for small graph.



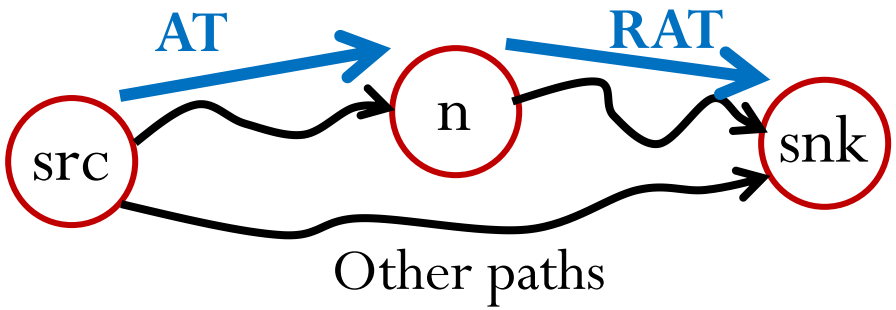
How many paths from node 0 to n?

2^n

- There's a smarter answer: **Node-oriented** timing analysis
  - Find, for **each node** in delay graph, **worst** delay to the node **along any path**.

Define Values on Nodes in Delay Graph

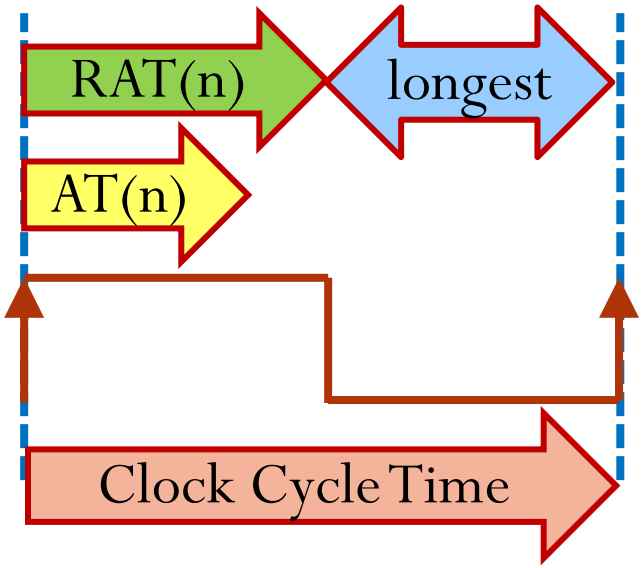
- **Arrival Time** at a node (AT)
  - AT(n) = **Latest** time the signal **can become stable** at node n.
  - Think: **Longest path from source**
- **Required Arrival Time** at node (RAT)
  - RAT(n) = **Latest time** in cycle where n could change and signal would still propagate to sink before end of cycle.
  - Think: **Longest path to sink**



Difference between ATs and RATs

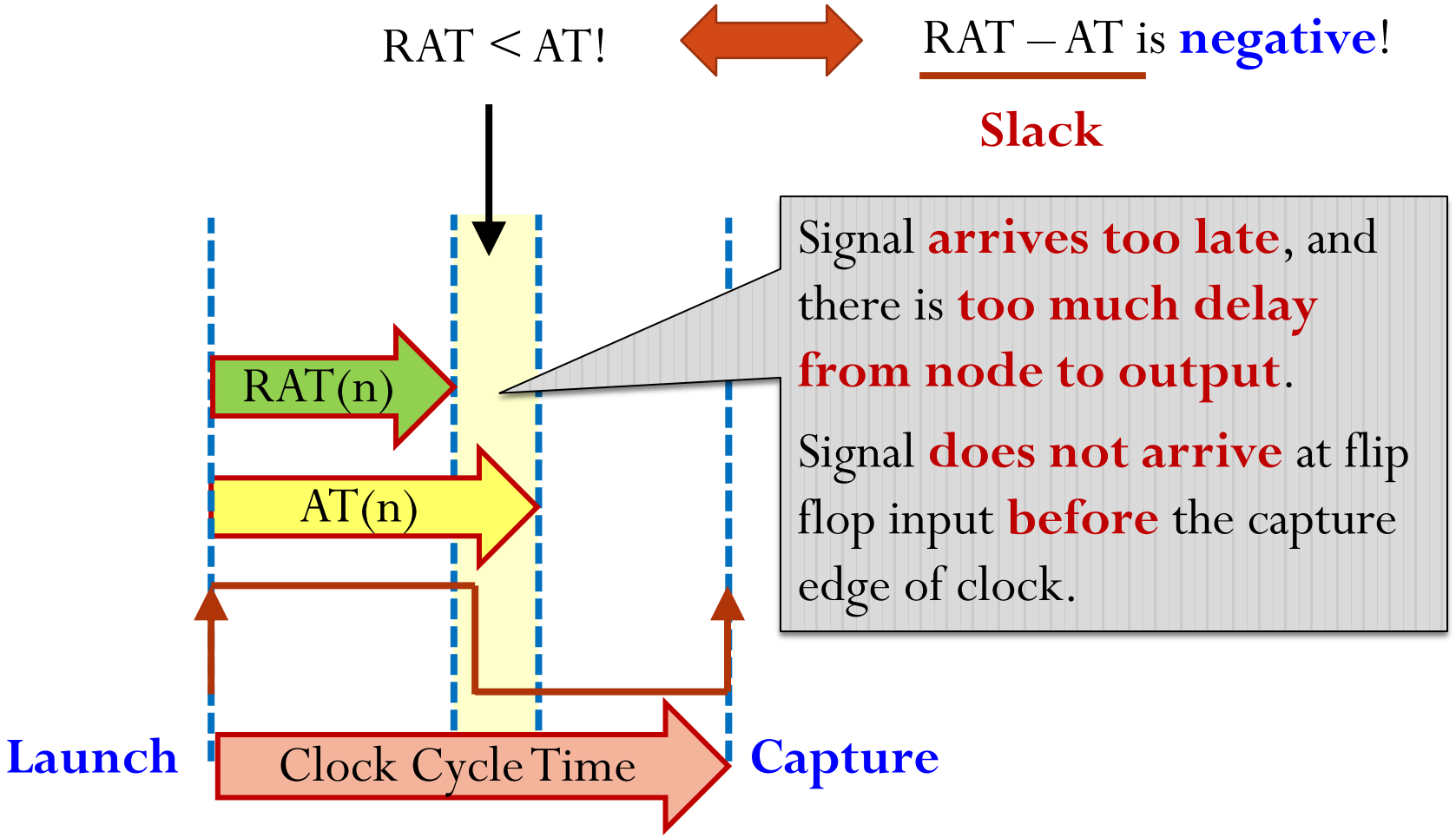
- Look at clock cycle

AT: **longest** logic delay after **launch** edge of clock.



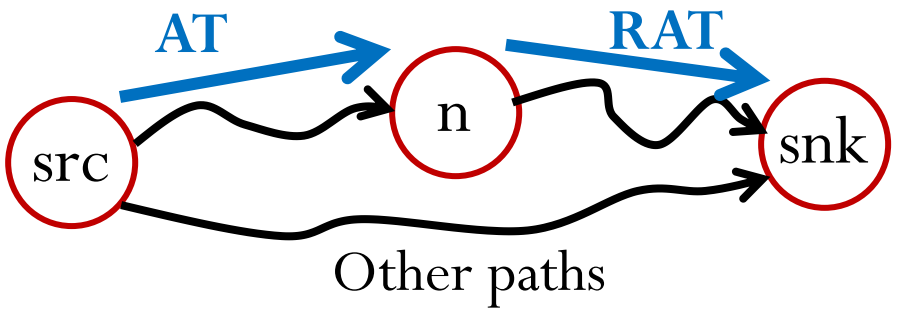
RAT: (essentially) **longest** logic delay to the **capture** edge of clock

RAT < AT is BAD!


RAT < AT!
RAT – AT is negative!
Slack
Signal arrives too late, and there is too much delay from node to output.
Signal does not arrive at flip flop input before the capture edge of clock.
RAT(n)
AT(n)
Launch
Clock Cycle Time
Capture

Slack

- **Slack** at node n: Slack(n) = RAT(n) – AT(n)
  - Amount of timing "margin" for the signal: positive is good, negative is bad.
  - Determined by **longest** path **through** node.
  - Measures "**sensitivity**" of network to this node's delay.

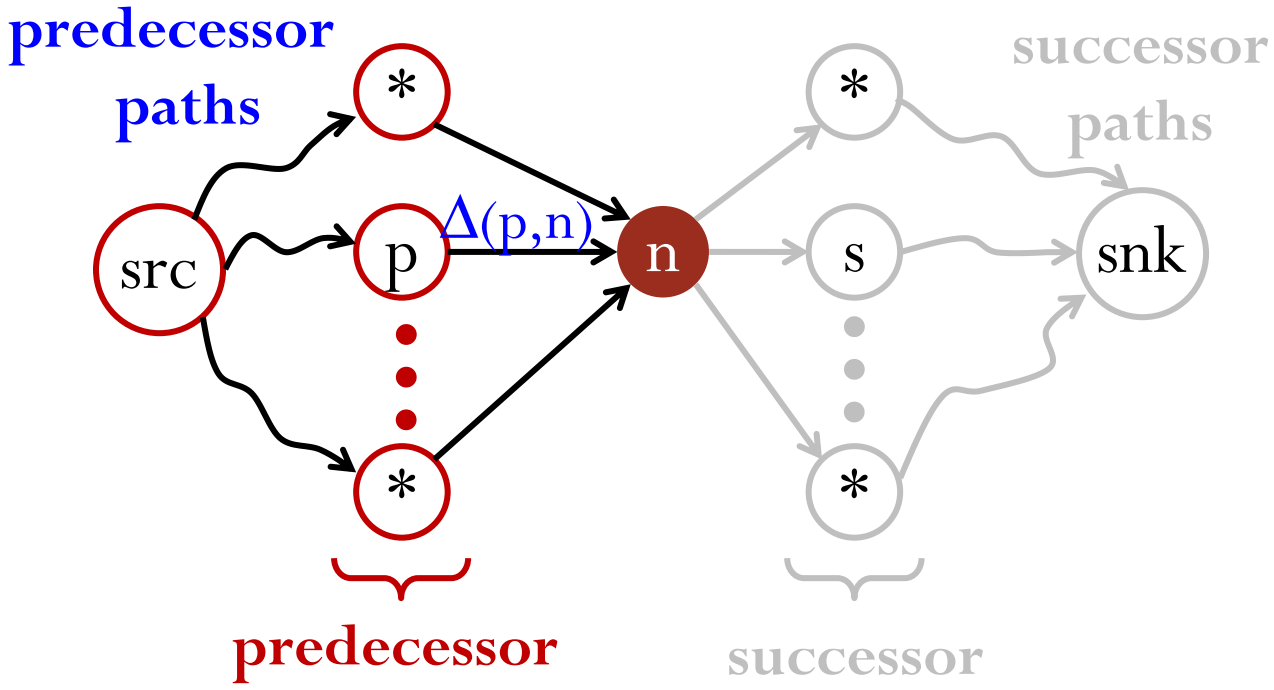


Slack(n) = RAT(n) – AT(n)

Slack is Hugely Important in Timing Analysis

- Positive slack
  - **Good**: Amount by which a signal can be **delayed** at node and **not increase** the **longest path** through the network.
  - Can change something at this node, and not hurt network's overall timing.
  - Example: make this node slower, maybe save some power, not hurt timing.
- Negative slack
  - **Bad**: have problem at this node; more negative the slack, bigger the problem.
  - Looking for a node to "fix" to help timing? These nodes are where to look first. These affect the critical paths the most.

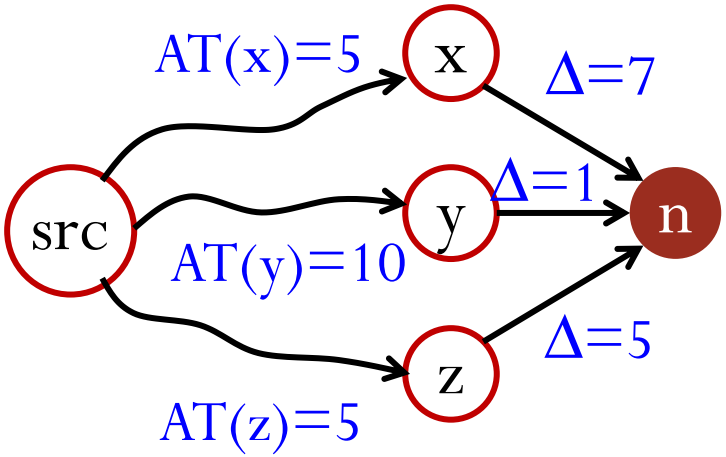
How To Compute ATs? Recursively



$$AT(n) = \text{maximum delay to } n = \begin{cases} 0, & \text{if } n \text{ is source} \\ \max_{p \in prec(n)} \{AT(p)+\Delta(p,n)\}, & \text{else} \end{cases}$$

How To Compute ATs?

- Big idea
  - If we know the longest path to each predecessor of n, it's a simple "**Maximum**" operation to compute the longest path to n itself.

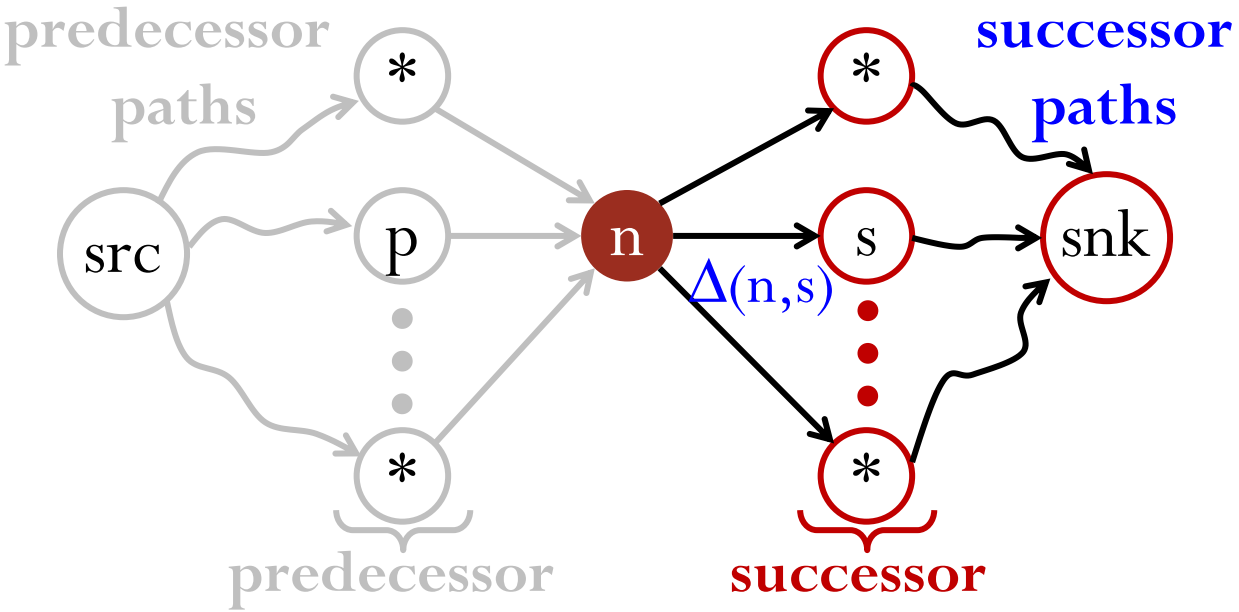


$$AT(n) = \max_{p \in \{x,y,z\}} \{AT(p)+\Delta(p,n)\}$$

$$= \max \{5+7, 10+1, 5+5\}$$

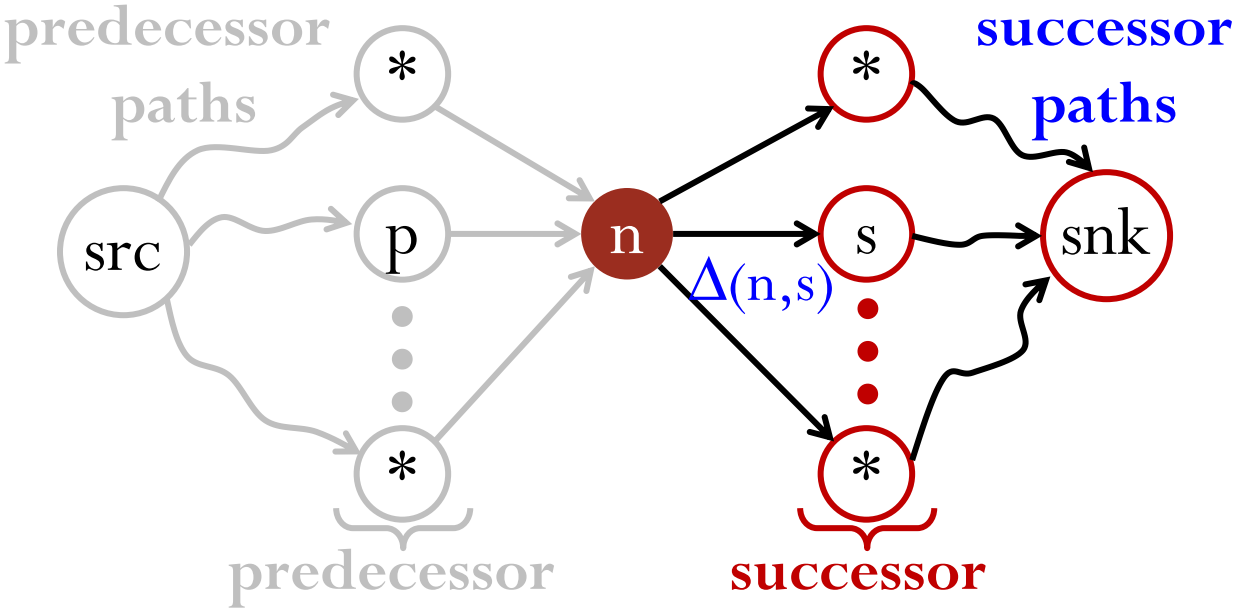
$$=12$$

How To Compute RATs?



- RAT(n): **Latest time** in cycle where n could change and signal would still propagate to sink before end of cycle.
- First, what is RAT(snk)? $RAT(snk) = Cycle\,Time$
- How about internal node n? $RAT(n) = \min_{s \in succ(n)} \{RAT(s) - \Delta(n,s)\}$

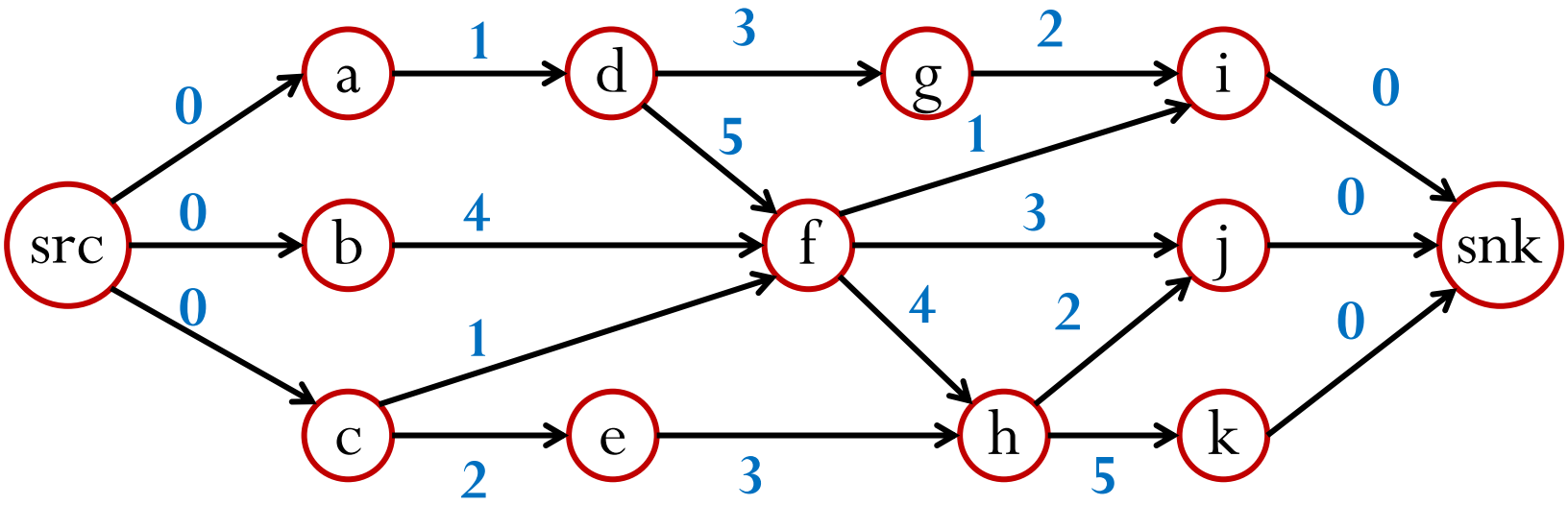
How To Compute RATs? Recursively



$$RAT(n) = \begin{cases} \text{Cycle Time}, & \text{if } n \text{ is sink} \\ \min_{s \in succ(n)} \{RAT(s) - \Delta(n,s)\}, & \text{else} \end{cases}$$

Example

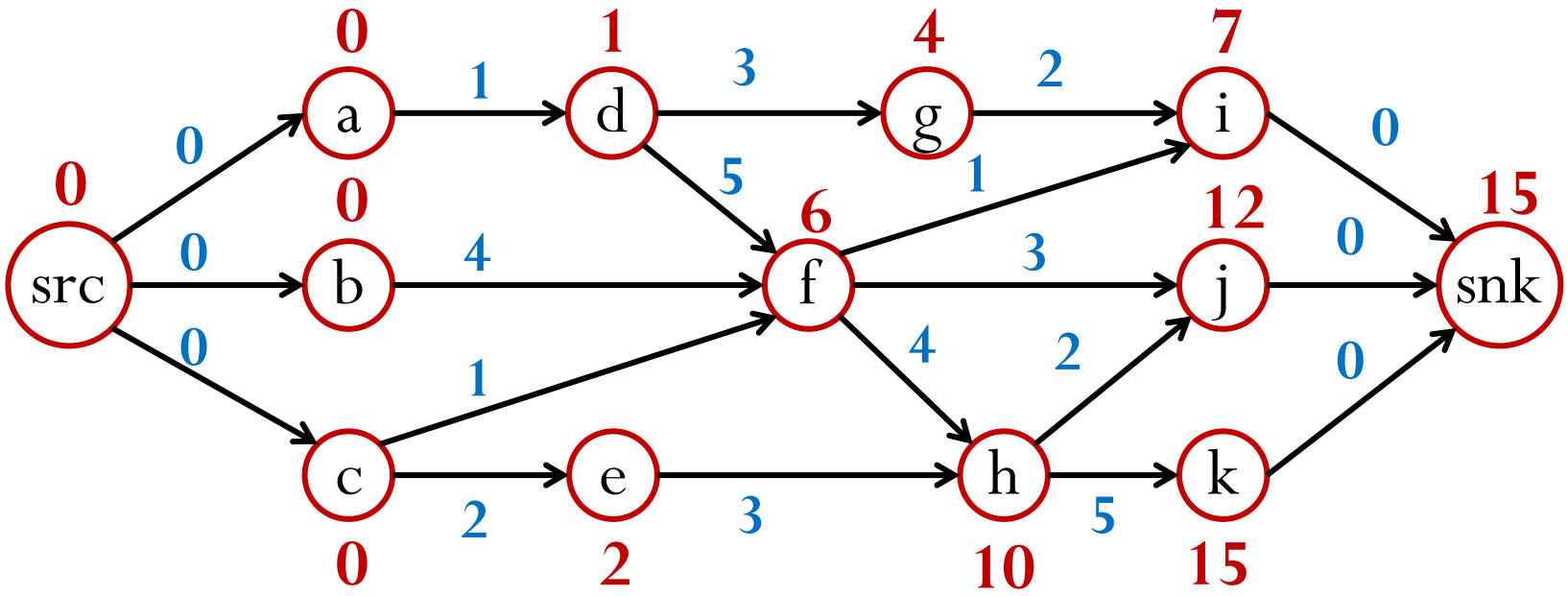
- Suppose clock cycle is 12.
- **AT**=longest path from source **TO** node.
- **RAT**=(cycle time 12) – (longest path **FROM** node to sink).
- **Slack** = **RAT** – **AT**



Compute ATs

$$AT(n) = \begin{cases} 0, & \text{if } n \text{ is source} \\ \max_{p \in prec(n)} \{AT(p)+\Delta(p,n)\}, & \text{else} \end{cases}$$

Compute ATs from src to snk

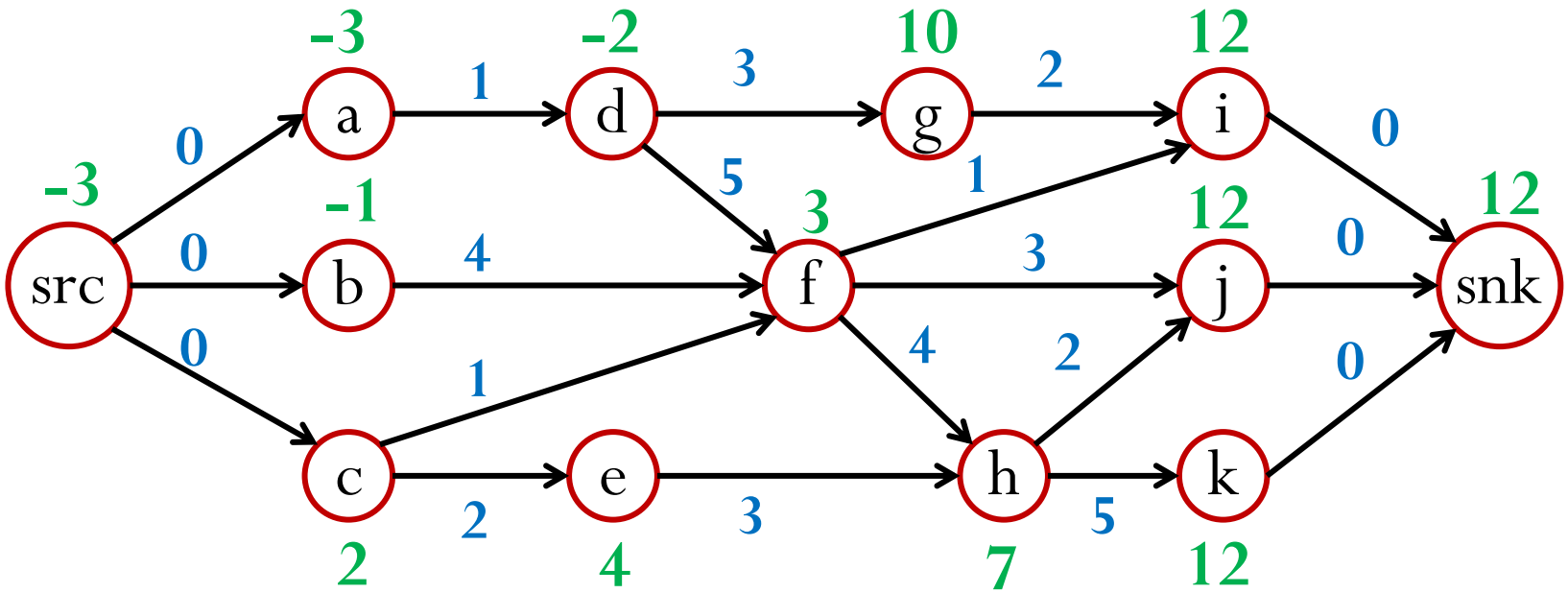


Compute RATs

- Clock cycle is 12.

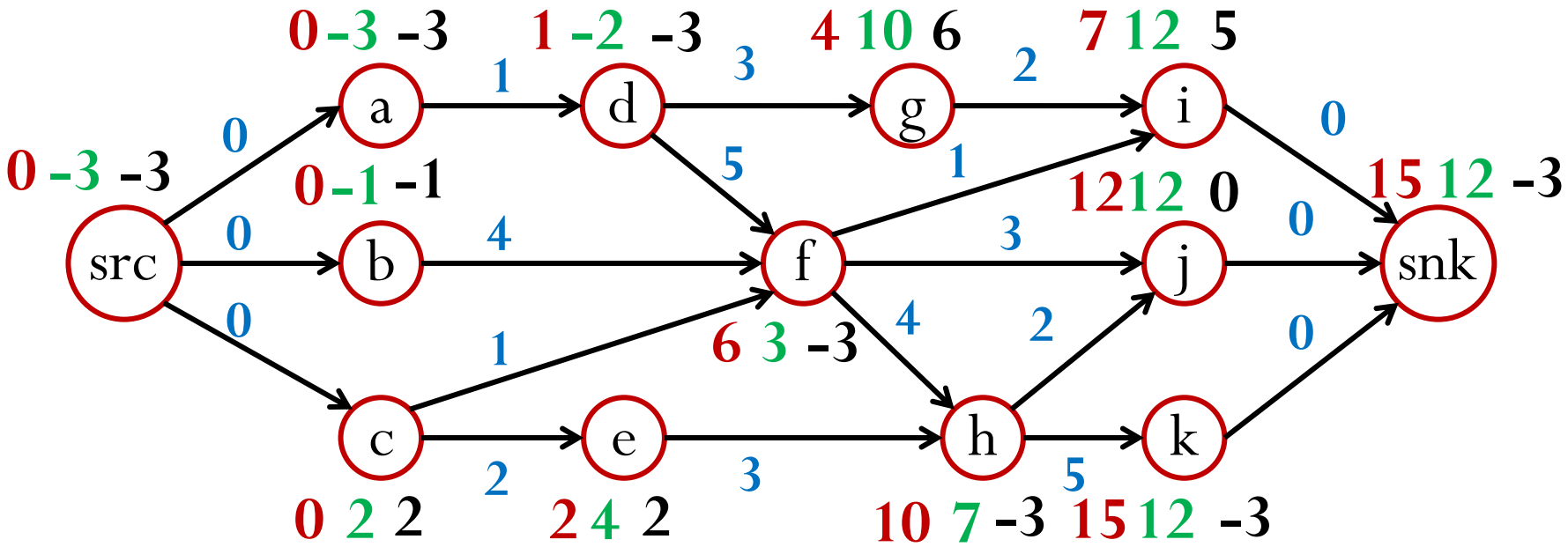
$$RAT(n) = \begin{cases} \text{Cycle Time}, & \text{if } n \text{ is sink} \\ \min_{s \in succ(n)} \{RAT(s) - \Delta(n,s)\}, & \text{else} \end{cases}$$

Compute RATs from snk to src



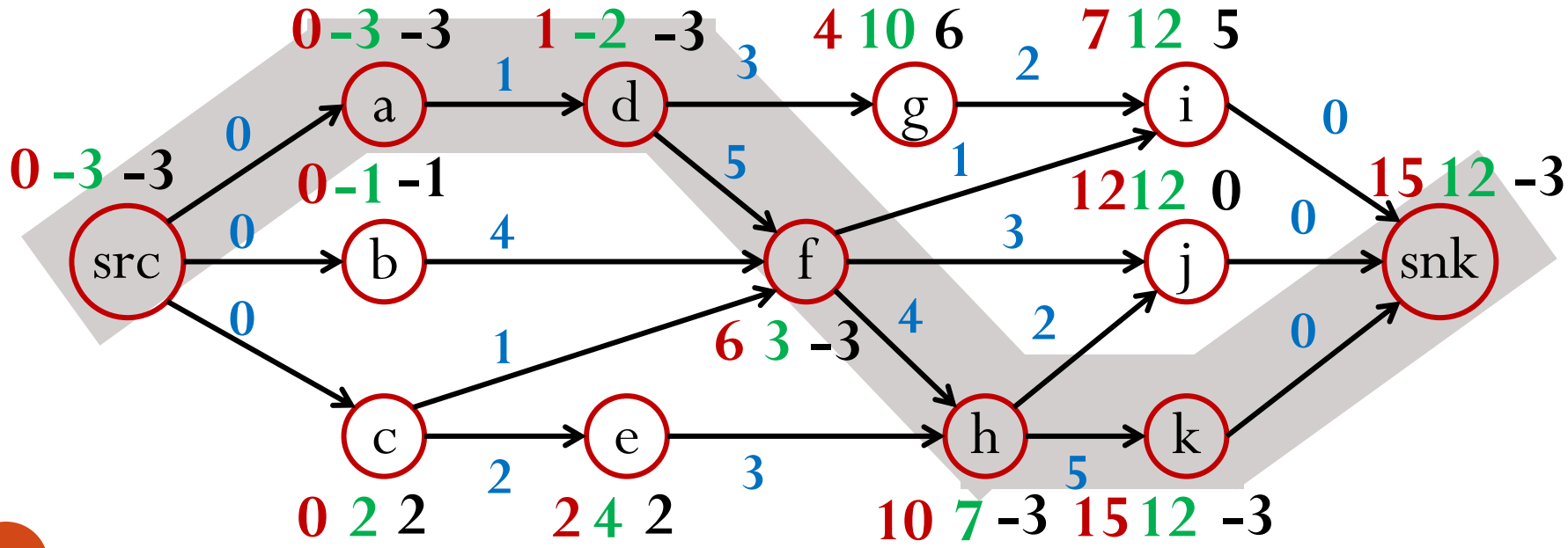
Compute Slack

- **Slack = RAT - AT**



Analyzing the Example

- Worst (most negative) slack is -3.
- Big results:
  - Your timing violation at sink = the worst slack value.
  - The worst slack appears along the **entire worst path**.



Analyzing the Example

- Look at those slacks
  - A negative slack at an output (PO) means a **failed** timing requirement.
  - A negative slack on internal node n means there is a path which has problem from some PI to some PO going through n.

- So, slacks are **hugely useful**!
  - Beyond just knowing what is the worst path, slacks tell us **problematic gates** on this path.