

# Lecture 19: Timing Analysis – I

---

Tsung-Wei (TW) Huang

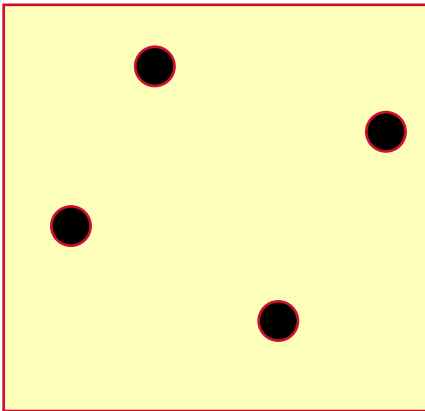
Department of Electrical and Computer Engineering

University of Utah, Salt Lake City, UT

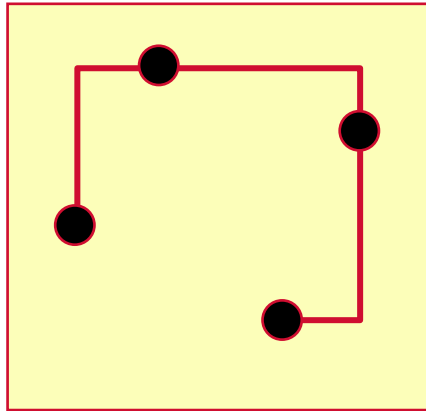


# Programming Assignment #3: Routing

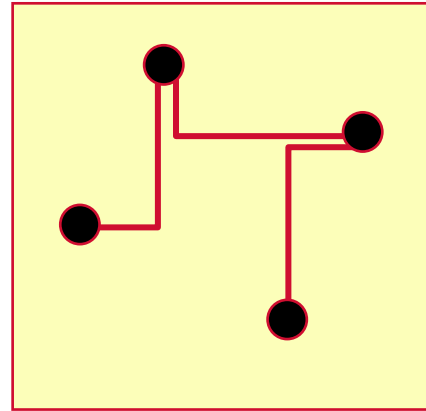
- Goal: Implement a Steiner Tree Construction Algorithm



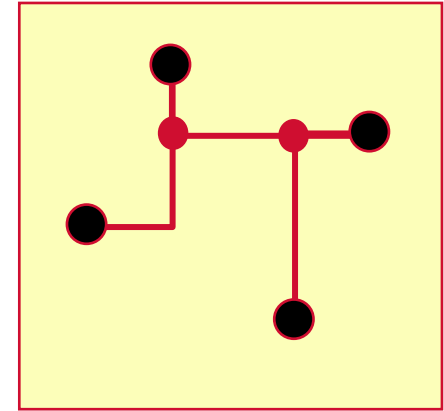
Pins to connect



Route it so we guarantee each 2-point path is shortest;



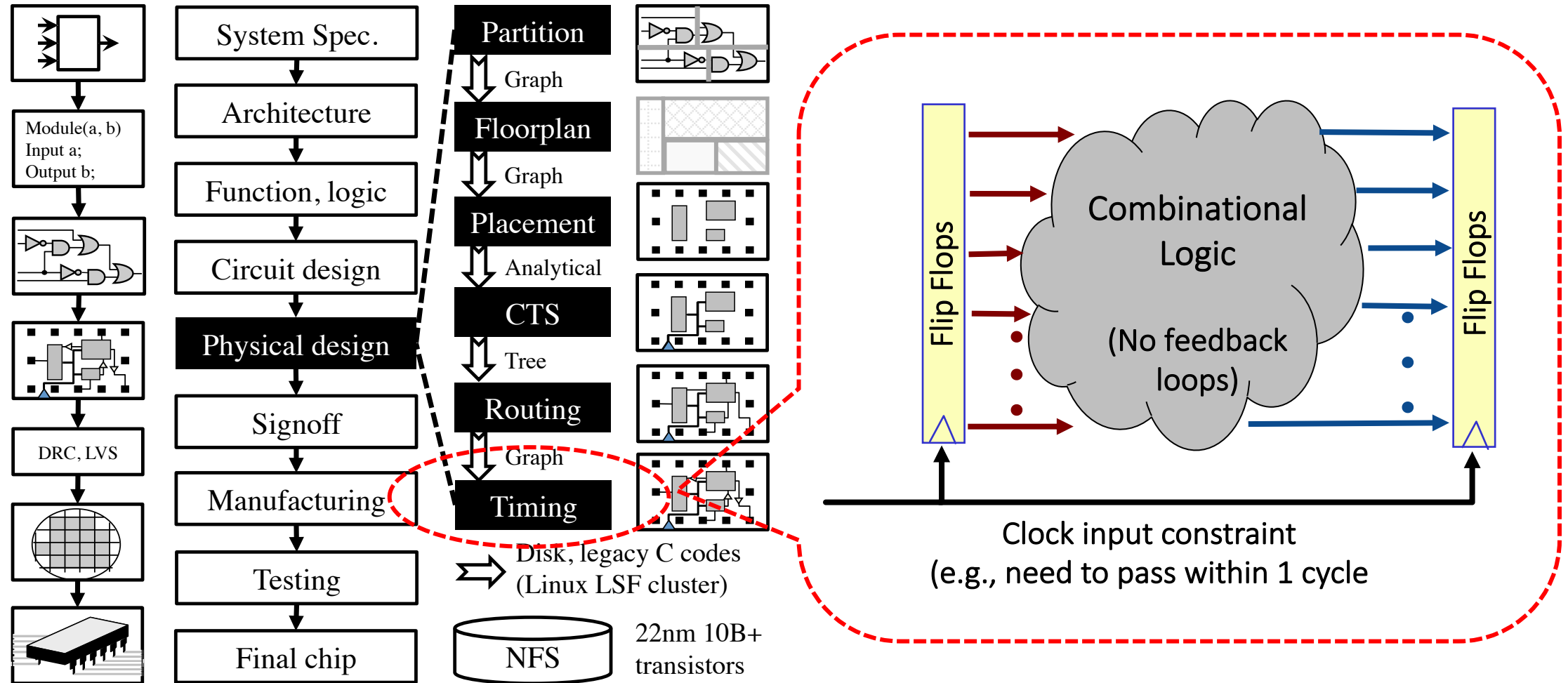
Redraw it--different orientations of 2-point paths



Now we can see the better (shorter) Steiner tree

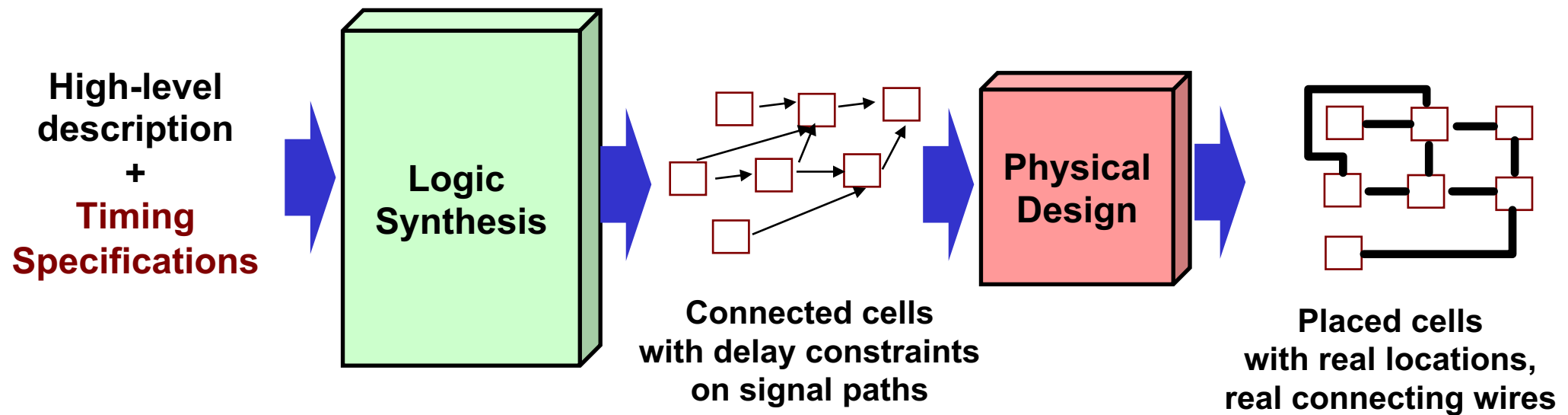
**Due 12/16:** <https://github.com/tsung-wei-huang/ece5960-physical-design/tree/main/PA3>

# Physical Design Flow



# Timing Analysis in Design Automation

- Deep interactions between logic synthesis and layout



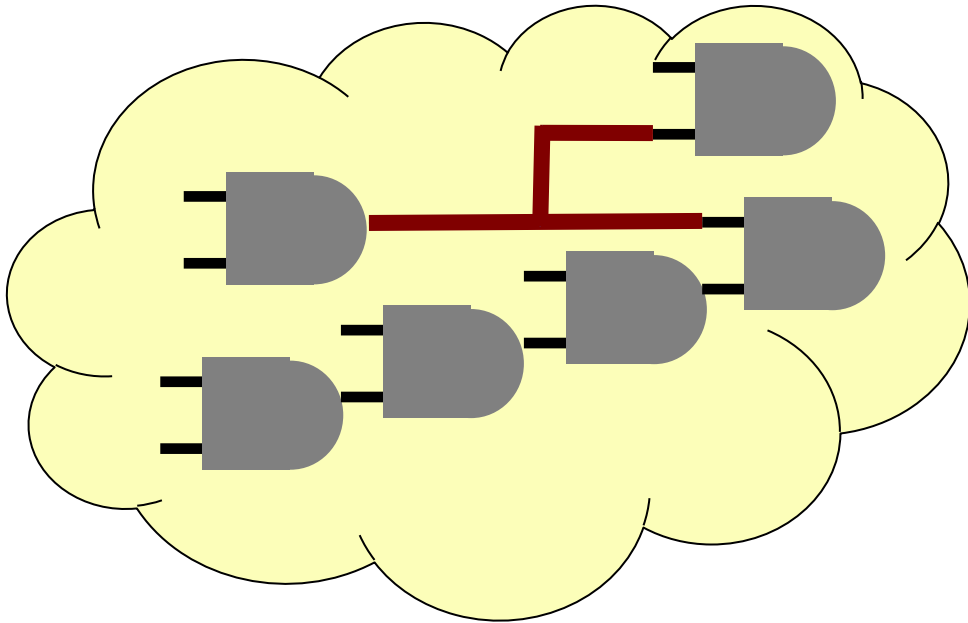
- **Important facts**

- Logic-side tools **estimate delays** through unplaced/unrouted logic
- Layout tools **estimate delays** through placed/routed logic

# Logic-Side Timing Analysis

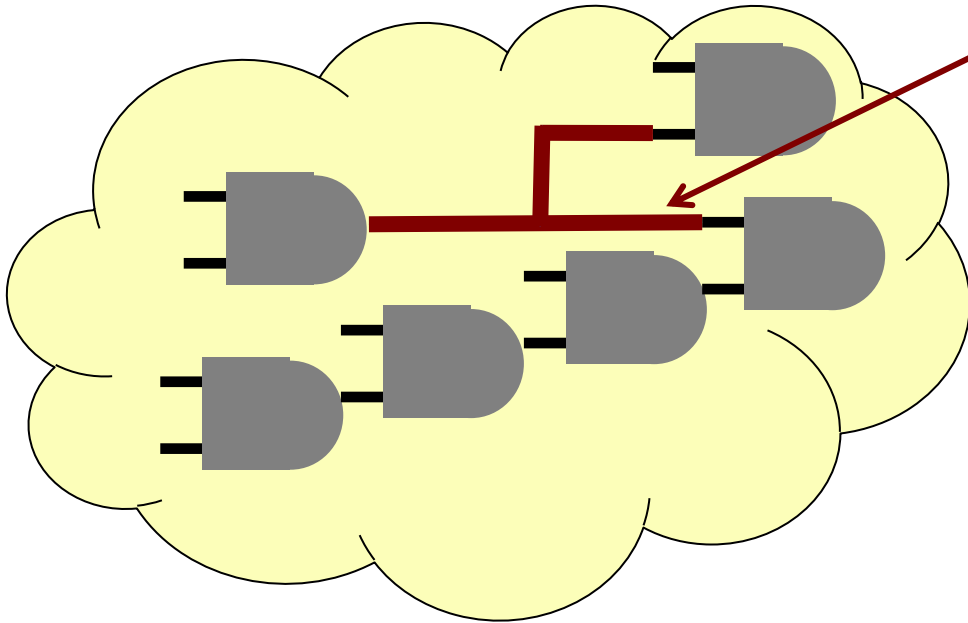
---

**Logic-side:** How do we estimate the worst-case timing through a logic network?

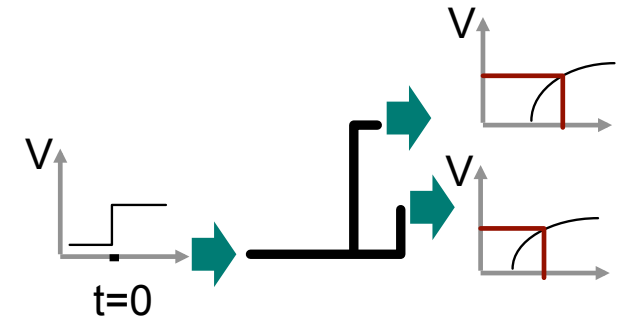
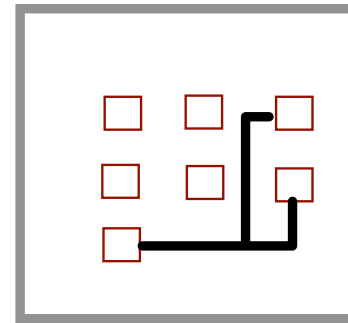


# Layout-Side Timing Analysis

**Logic-side:** How do we estimate the worst-case timing through a logic network?



**Layout-side:** We place the gates, route the wires: how do we estimate wire delays?



# Big Picture

---

- **On the logic side:**

- All problems look like **longest** (or shortest) **paths through a graph** that properly models the gates, and (maybe) the wires

- **On the layout side:**

- The problem starts as an electrical circuit model (this is unavoidable)
- However, we skip circuit details, and just show key results

- **Surprisingly, both problems can be easily and efficiently solved using shortest path algorithms!**

# Timing Analysis at Logic Level

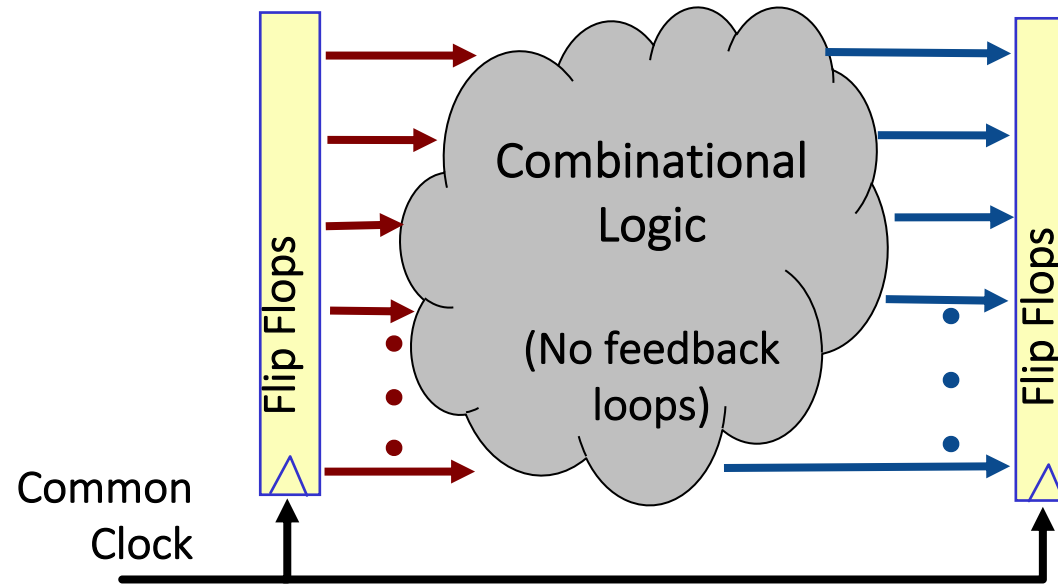
---

- **Goal: Verify timing behavior of our logic design**
  - I give you a gate-level netlist
  - I give you some **timing models** of the gates and (after place/route) the wires too
  - You tell me:
    - When signals **arrive** at various points in the network
    - **Longest** delays through gate network
    - Does the netlist **satisfy** the timing requirement? If not – **where** are key problems?
- **Challenge: How do you estimate the timing correctly?**
  - We can't! But we know the worst and best-case timing



# Analyze Design Performance

- Practical designs are **synchronous**
  - All storage is in explicit sequential elements, e.g., flip-flop elements
  - We can just focus on delays through combinations gates



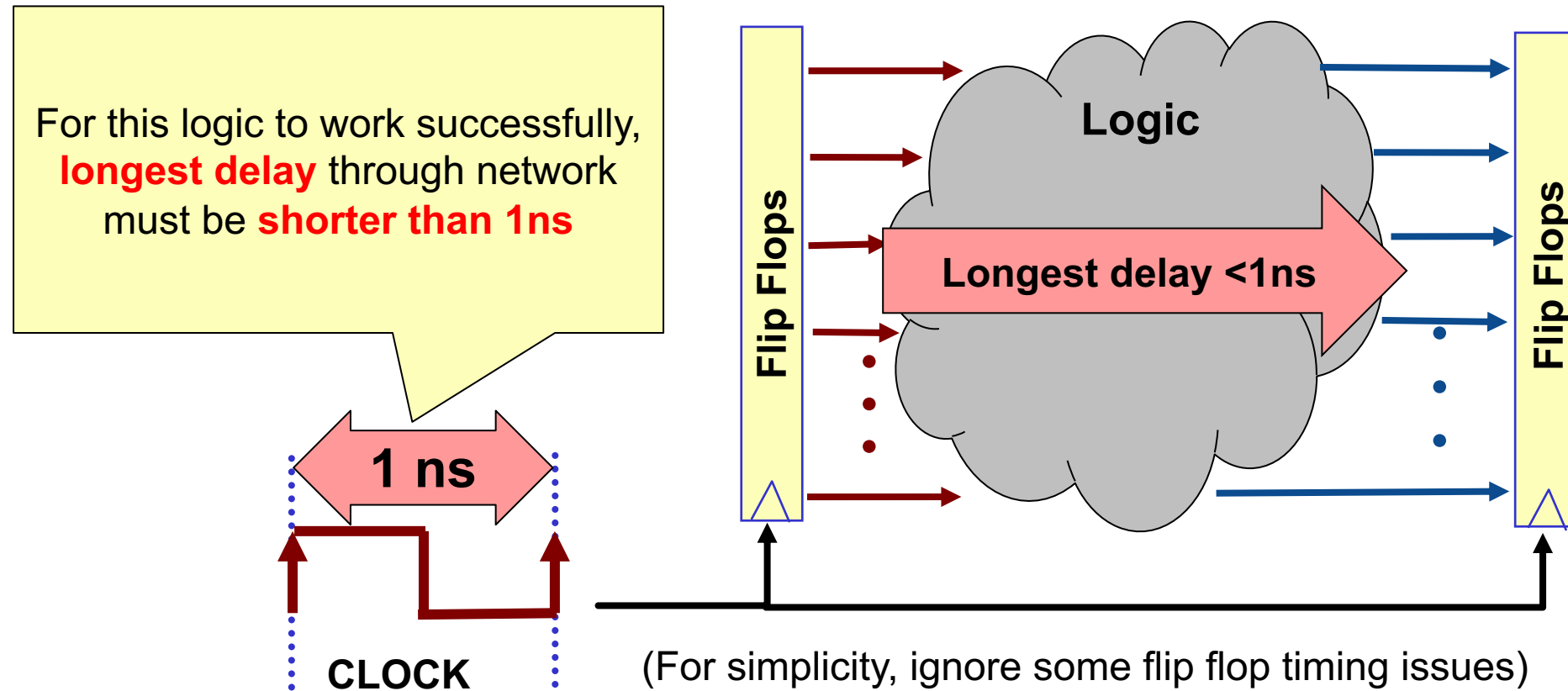
# Can't We Just Simulate Logic?

---

- **What logic simulation does**
  - Determine how a system will behave, simulates the logical function
  - 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 ...
  - Under all possible manufacturing variations ...
- **We need a **different, faster** solution**

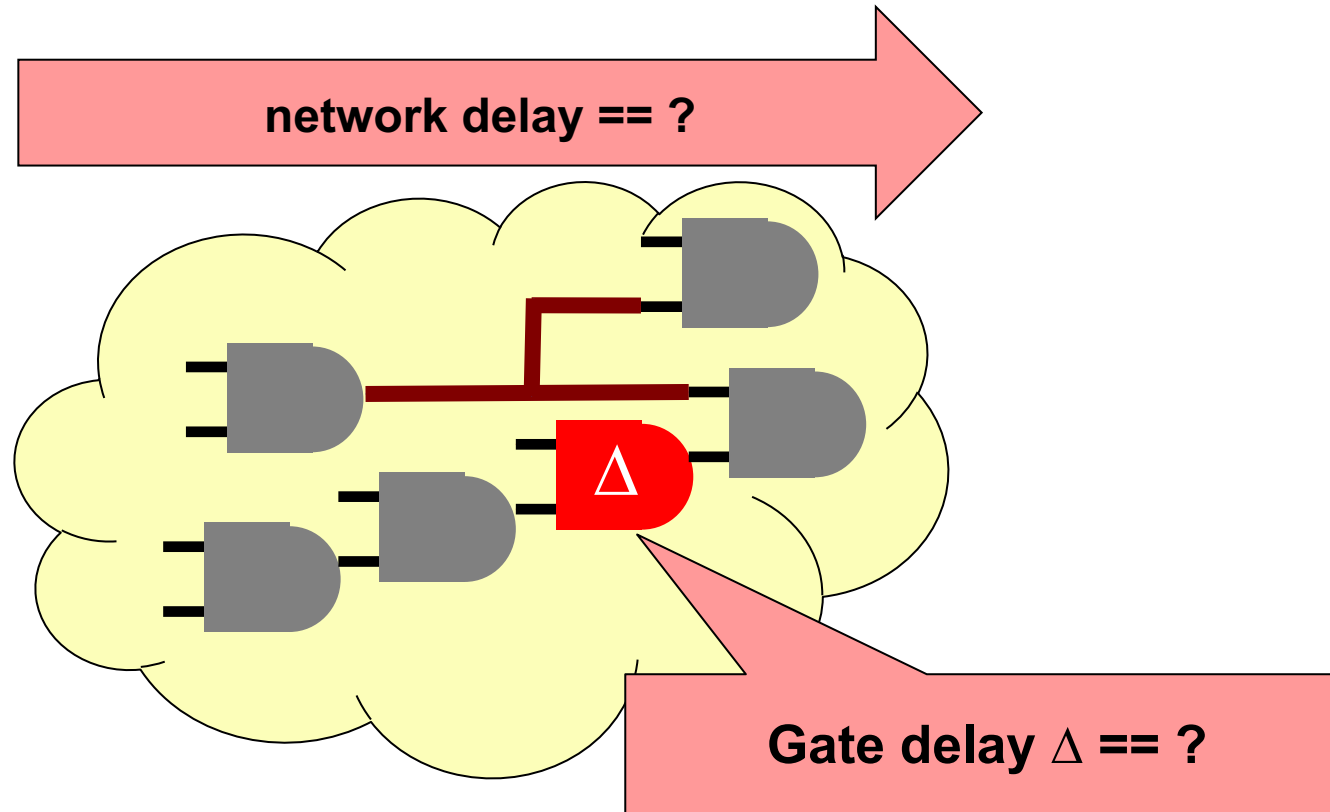
# Timing Analysis: Basic Model

- Assume we know **clock cycle**: e.g., 1GHz clock, cycle = 1ns



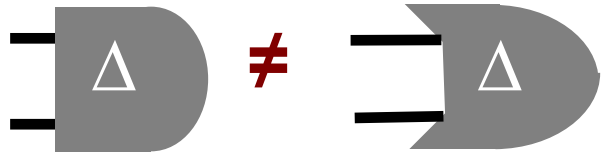
# Timing Analysis: Basic Model (cont'd)

- We need a model of **delay** through each logic gate

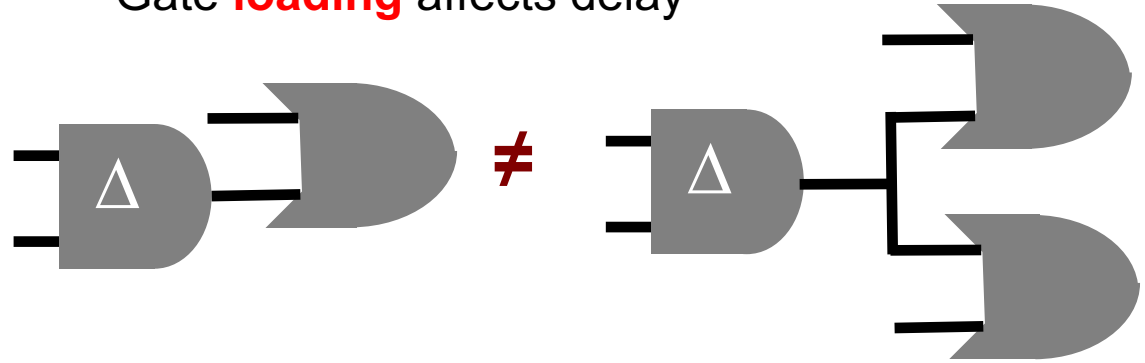


# In the Real World ...

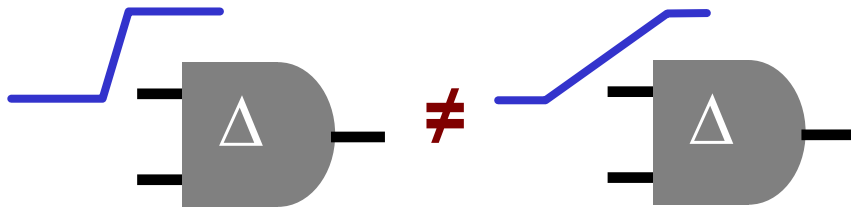
Gate **type** affects delay



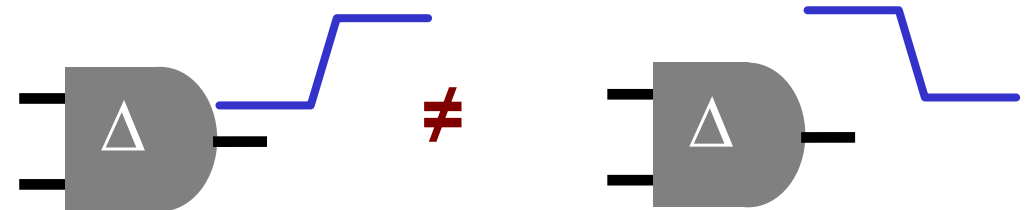
Gate **loading** affects delay



**Waveform shape** affects delay

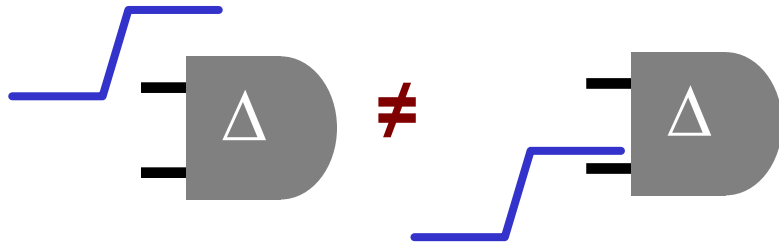


**Transition direction** affects delay

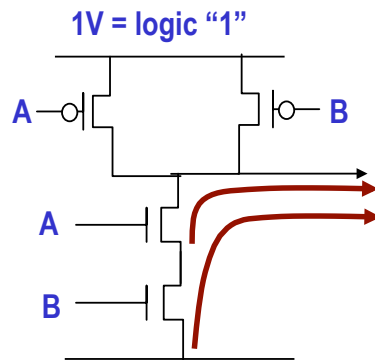


# In the Real World ... (cont'd)

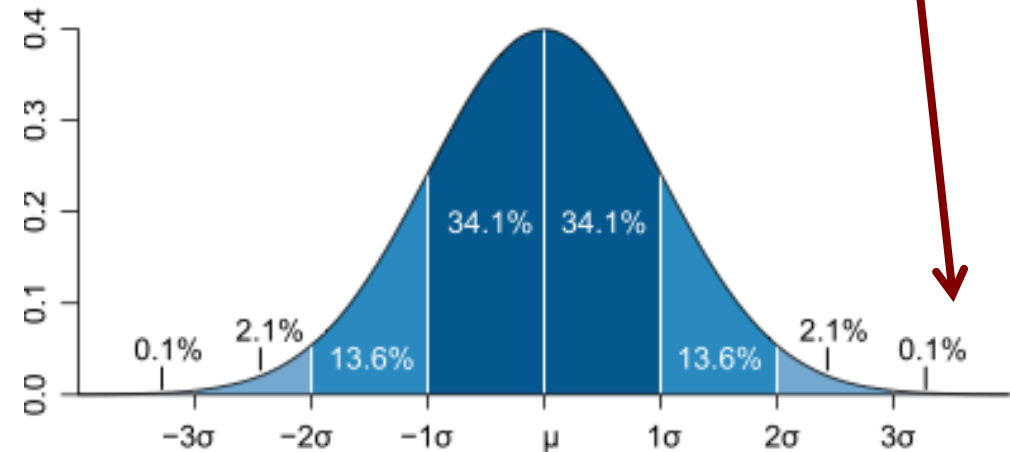
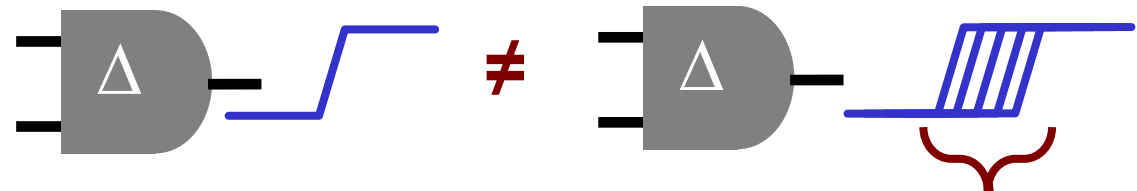
Gate **input pin** affects delay



Why? Different transistor-level circuit paths input to output  
Simple ex: NAND



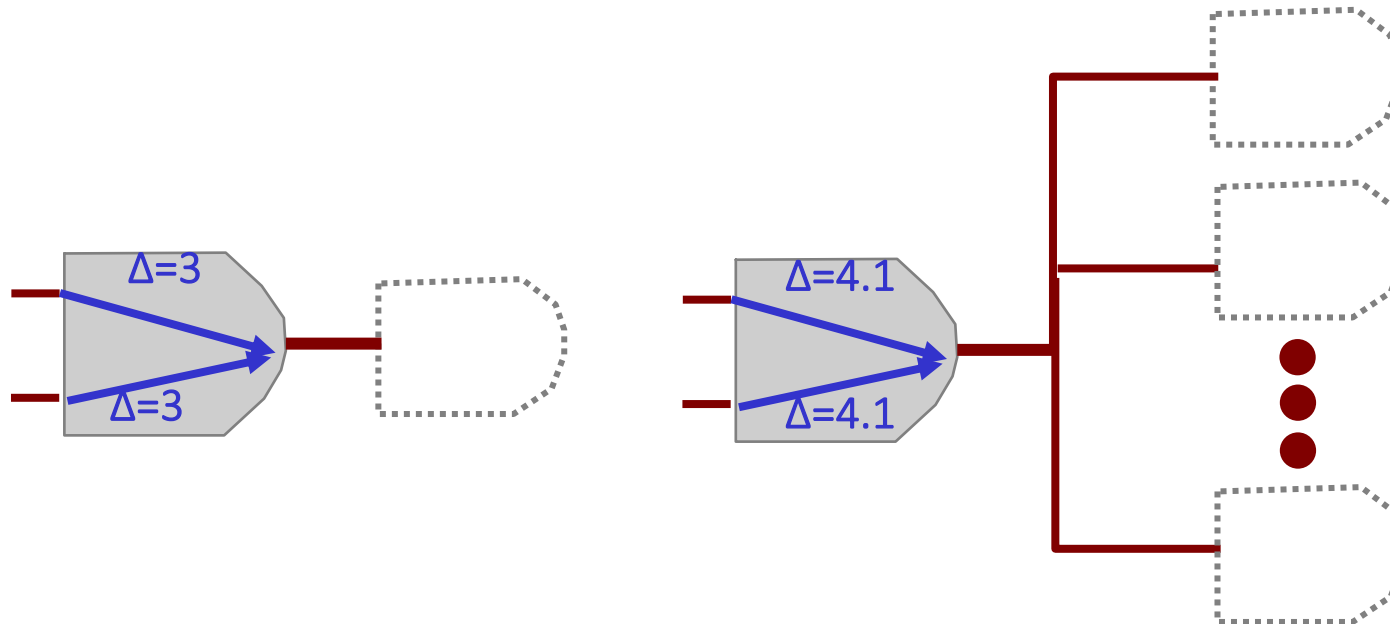
At nanoscale, delays are really **statistical**



[http://upload.wikimedia.org/wikipedia/commons/8/8c/Standard\\_deviation\\_diagram.svg](http://upload.wikimedia.org/wikipedia/commons/8/8c/Standard_deviation_diagram.svg)

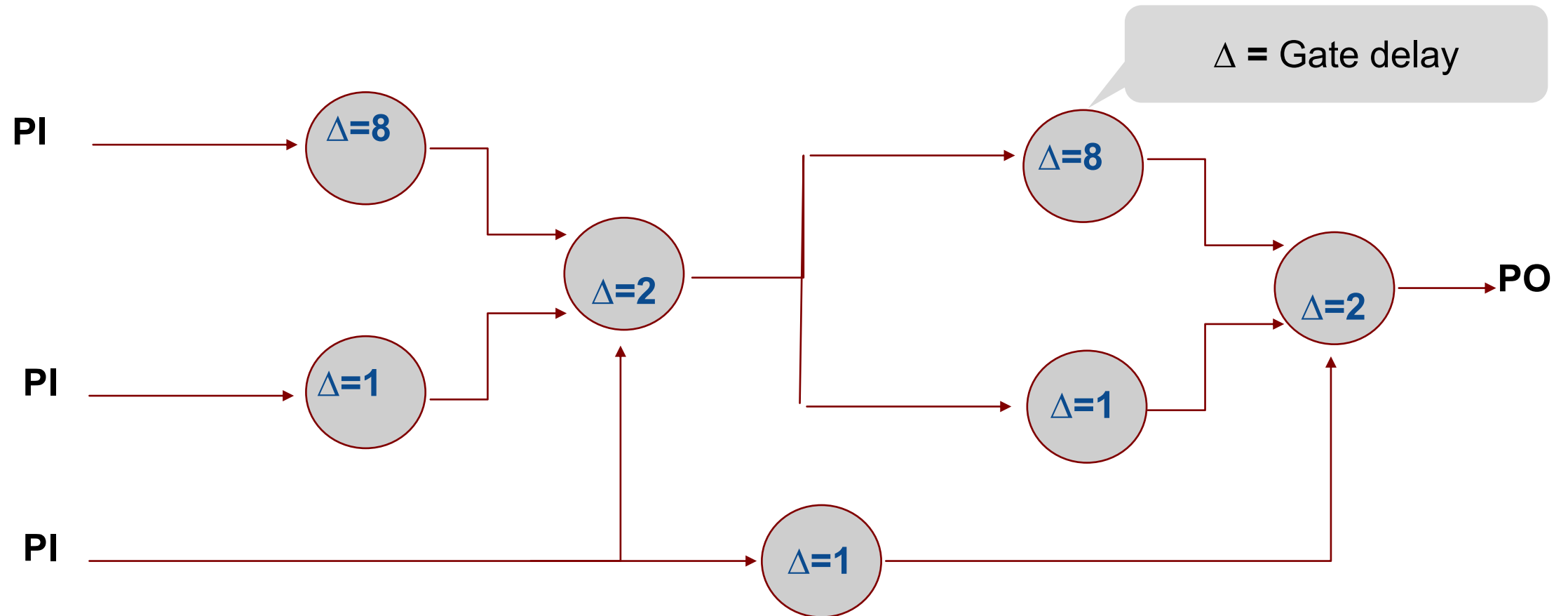
# Our Model: Pin-to-Pin Delay

- We will keep it simple: **Fixed, pin-to-pin delay model**
  - No slopes, electricity, distributions, etc. – Just gate delay itself!
  - Per-pin delays are essential, but we'll use *just 1 value per gate*
  - Turns out this is enough to see all the interesting algorithm ideas



# Example

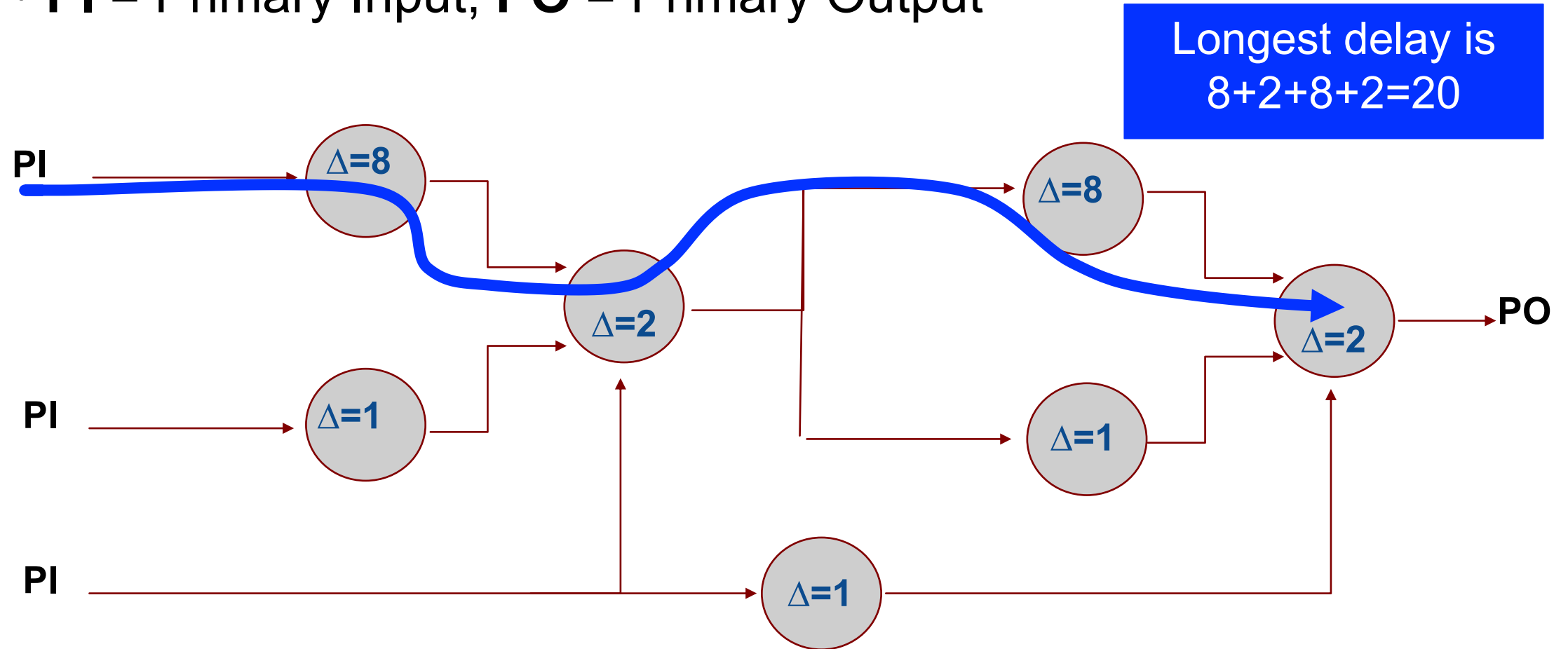
- **PI** = Primary Input, **PO** = Primary Output





# Example (cont'd)

- **PI** = Primary Input, **PO** = Primary Output



# Static Timing Analysis (STA)

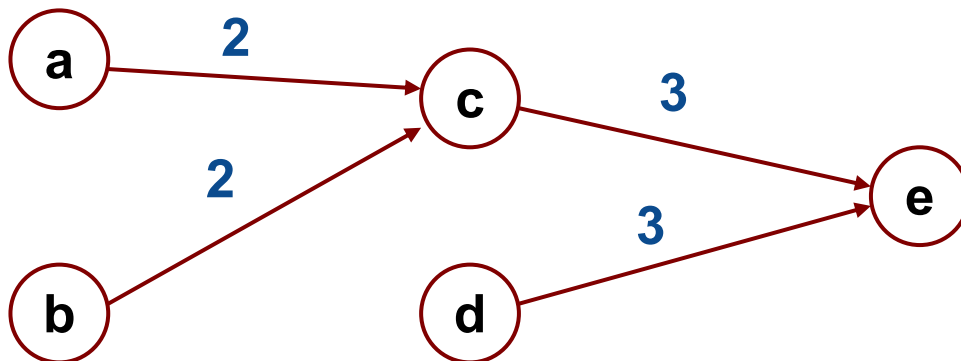
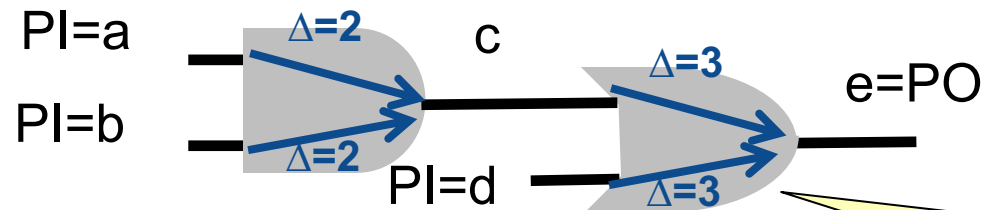
---

- **When we ignore logic, this is called Topological Analysis**
  - We only work with the graph and the delays – **don't** consider the logic
  - We can get wrong answers: what we found was called a **False Path**
- **Going forward: we ignore the logic**
  - Assume that all paths are **statically sensitizable**
    - **Means:** Can find a constant pattern of inputs to *other* PIs that makes some output sensitive to some input
- **This timing analysis is called Static Timing Analysis (STA)**
  - Consider only the best- and worst-case timing results
  - Consider no logic (otherwise called dynamic timing analysis)

# STA Representation: Delay Graph

- **From gate-level network, we build a delay graph**

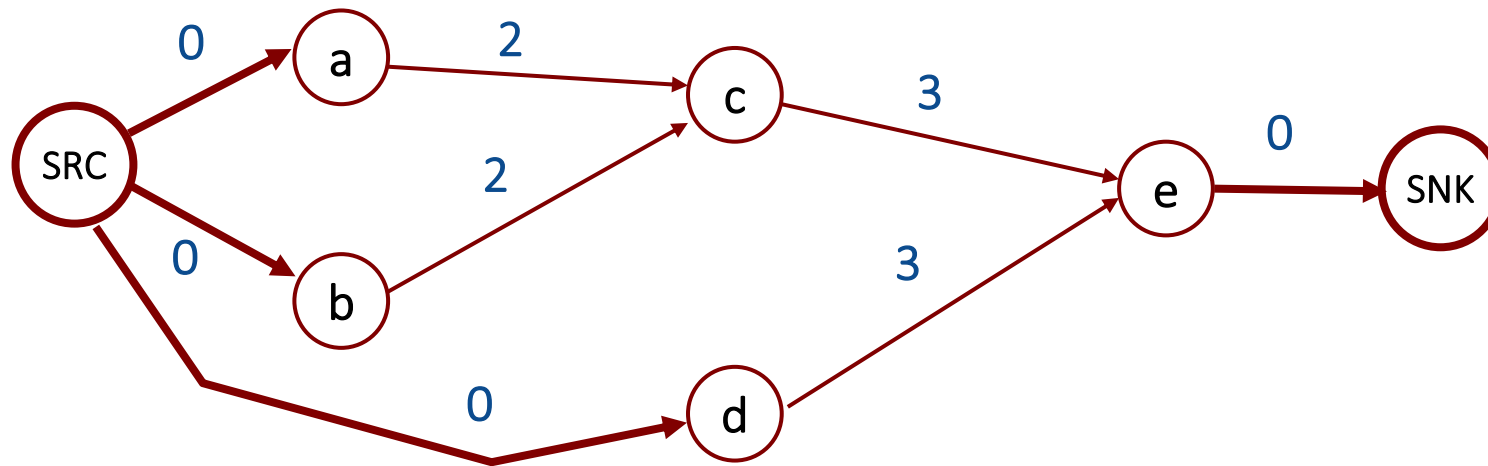
- **Vertices: Wires** in gate network, 1 per gate output, 1 for each PI and PO
- **Edges: Gates**, input pin to output pin (1 edge per input). Put gate delays on edges



Called **Cell arcs**. Because they are *edges*, that explain *timing*, for each *cell* in tech library.

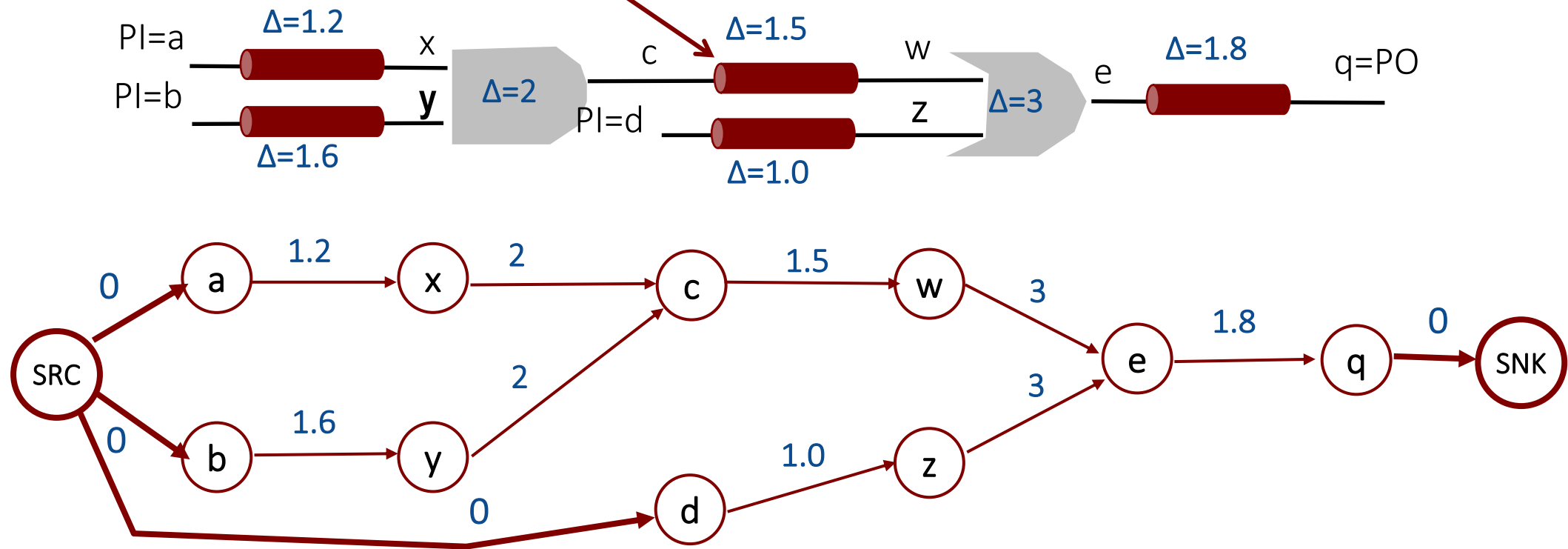
# Source and Sink in Delay Graph

- **Common convention: Add Source / Sink nodes**
  - Add 1 “source” (SRC) node that has a 0-weight edge to each PI
  - Add 1 “sink” (SNK) node with 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 question have same start / end nodes



# What about Interconnect among Gates?

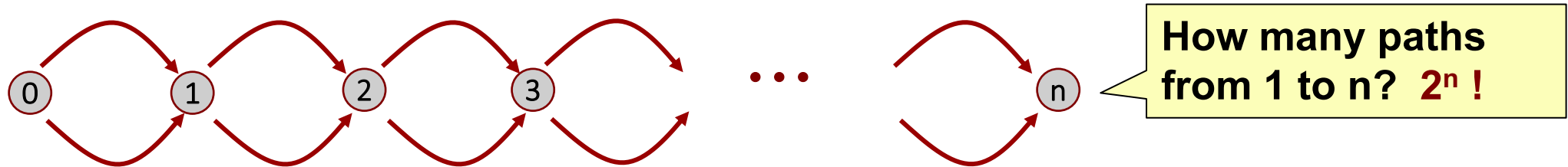
- Can still use delay graph: model each wire as a “special” gate that just has a **delay**



# Operations on Delay Graph

- **So how do we use this graph to do timing analysis?**

- What we do **not** do: Try to *enumerate* all the source-to-sink paths
- Why not? Exponential explosion in number of paths, even for small graph

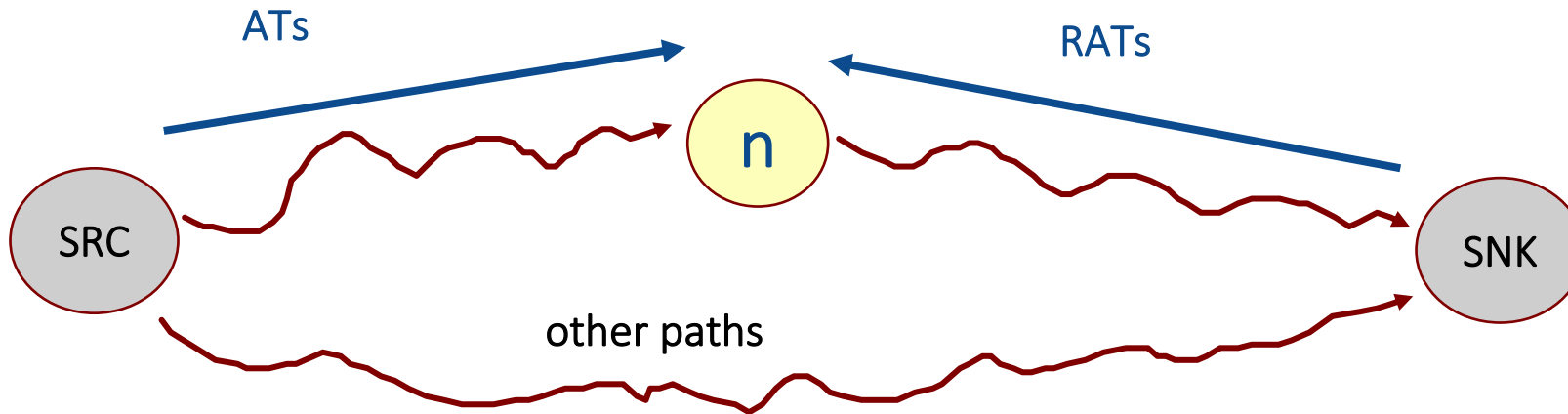


- **There's a smarter answer: Node-oriented timing analysis**

- Find, for **each node** in delay graph, **worst** delay to the node **along any path**

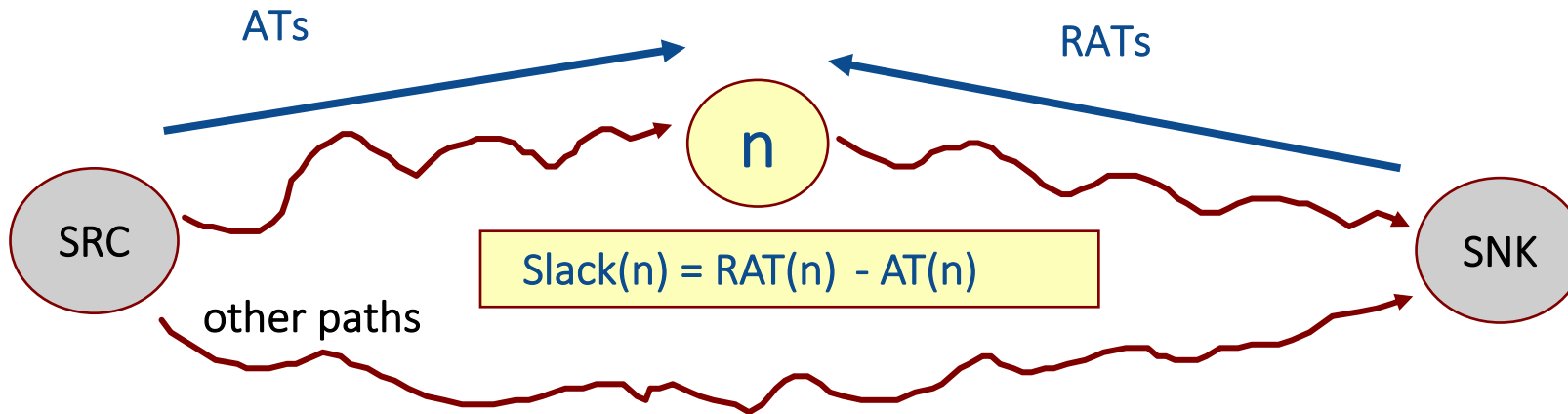
# Define Values on Nodes in Delay Graph

- Arrival Time at a node (AT)
  - **AT(n) = Latest** time the signal **can** become stable node **n**
  - Think: **Longest path from source**
  - Called: **Delays TO node**
- Required Arrival Time at node (RAT)
  - **RAT(n)=Latest** time the signal is **allowed** to become stable at node **n**
  - Think: **Longest path to sink (sort of...)**
  - Called: **Delays FROM node**



# Measure Timing Margin at a Node

- **Slack at node n:  $\text{Slack}(n) = \text{RAT}(n) - \text{AT}(n)$** 
  - Amount of timing “margin” for the signal: positive is **good**, negative is **bad**
  - Determined by **longest path through node**
  - Amount by which a signal can be **delayed** at node and **not increase the longest path** through the network
  - Can **increase delay** at node (to minimize power, circuit area) with **positive slack** and **not** degrade overall performance





# Slack is Important in Timing Analysis

---

- **About slacks**

- Defined so **negative slack *always* bad** --, it indicates a timing problem
- Measures “**sensitivity**” of network to this node’s delay

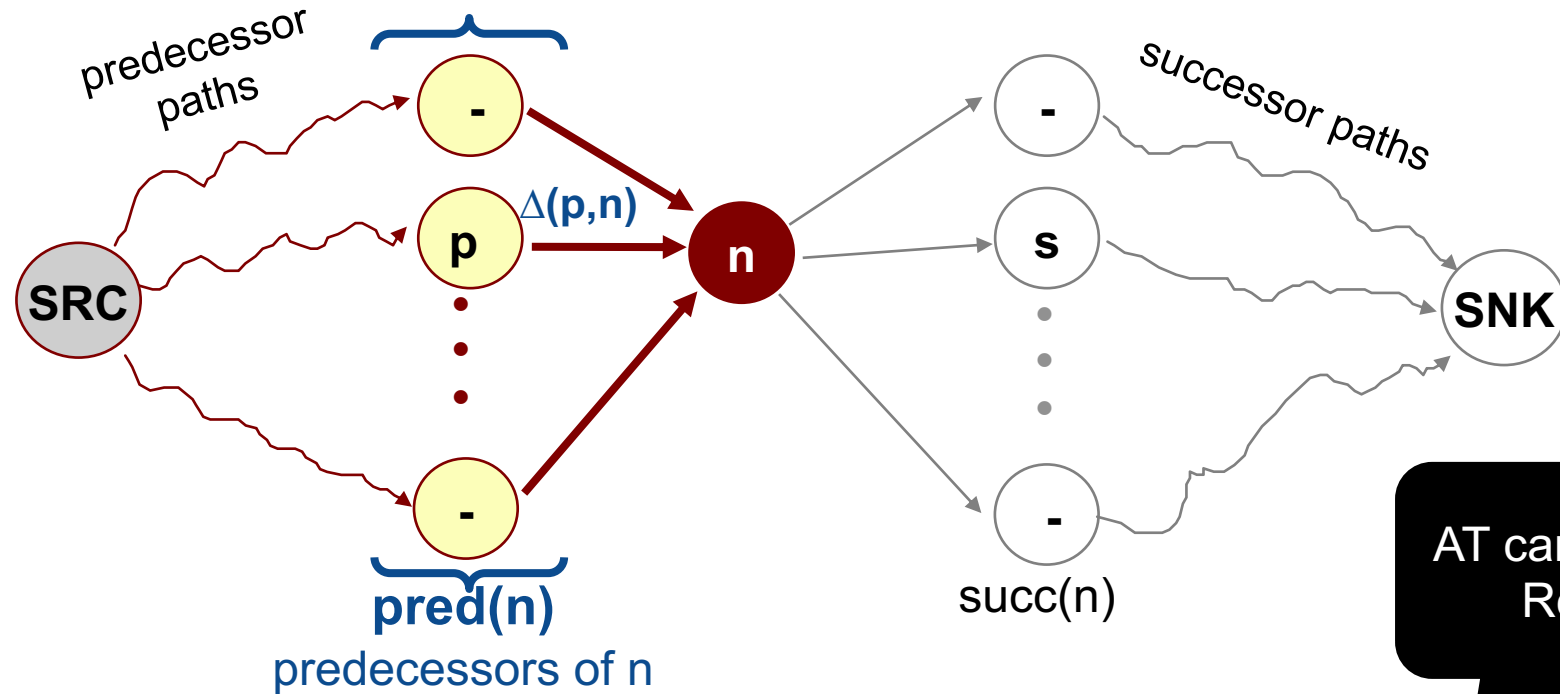
- **Positive slack**

- **Good:** I can change something at this node, and not hurt network’s overall timing
- Example: I can make this node slower, maybe save some power, not hurt timing

- **Negative slack**

- **Bad:** I 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 my critical paths the most

# How to Compute Arrival Time (AT)?



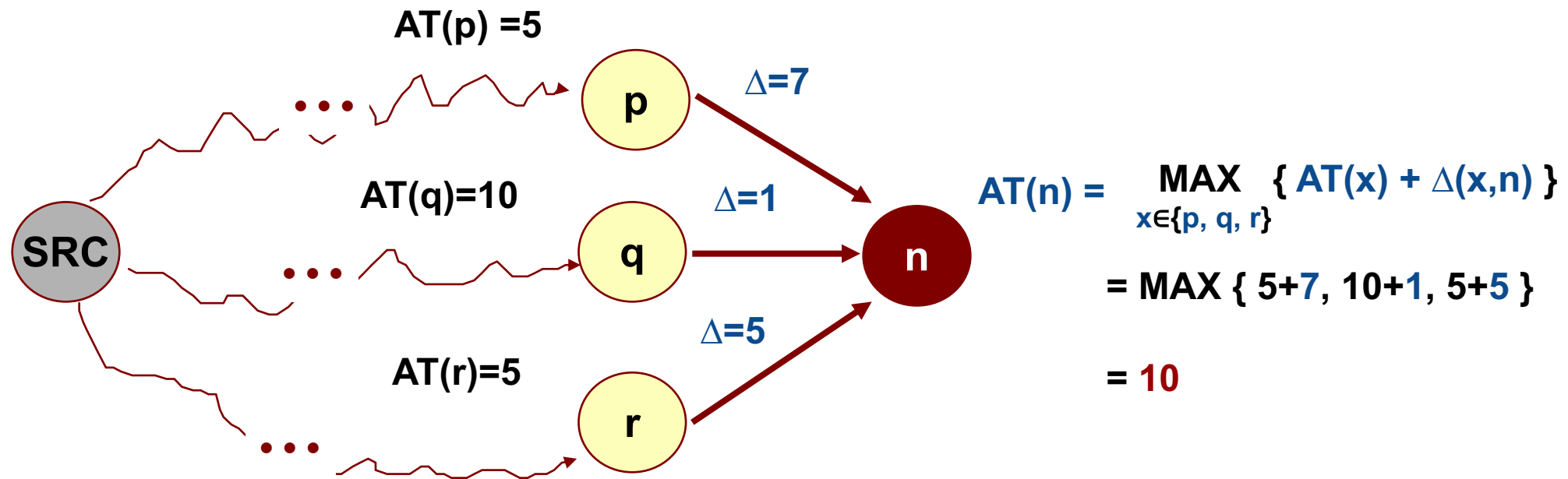
AT can be computed Recursively!

$AT(n)$  = maximum delay to  $n$  =

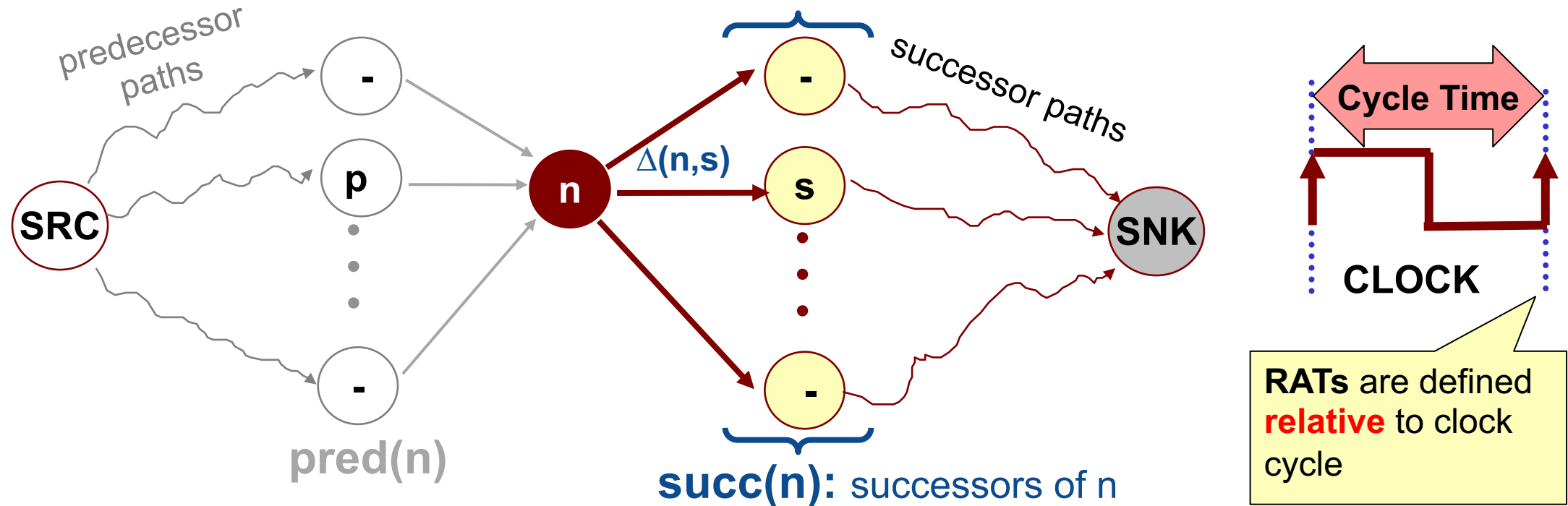
$$\begin{cases} 0 & \text{if } n == SRC \\ \text{MAX}_{p \in pred(n)} \left[ AT(p) + \Delta(p,n) \right] & \text{else} \end{cases}$$

# Example: Compute AT

- 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—Yes, it is shortest-path algorithm again!



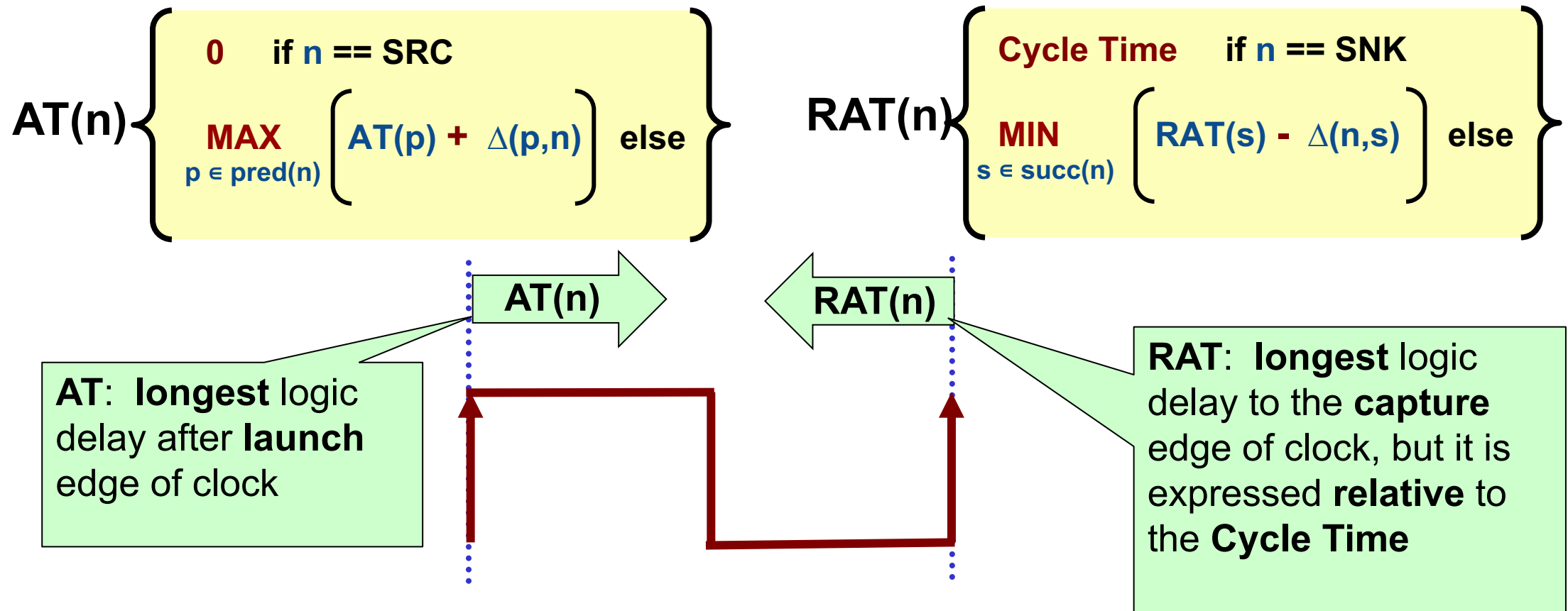
# How to Compute Required Arrival Time (RAT)?



$$\text{RAT}(n) = \begin{cases} \text{Latest time in cycle where } n \text{ could change and signal would still propagate to sink before end of cycle} & \text{if } n == \text{SNK} \\ \text{MIN}_{s \in \text{succ}(n)} \left[ \text{RAT}(s) - \Delta(n,s) \right] & \text{else} \end{cases}$$

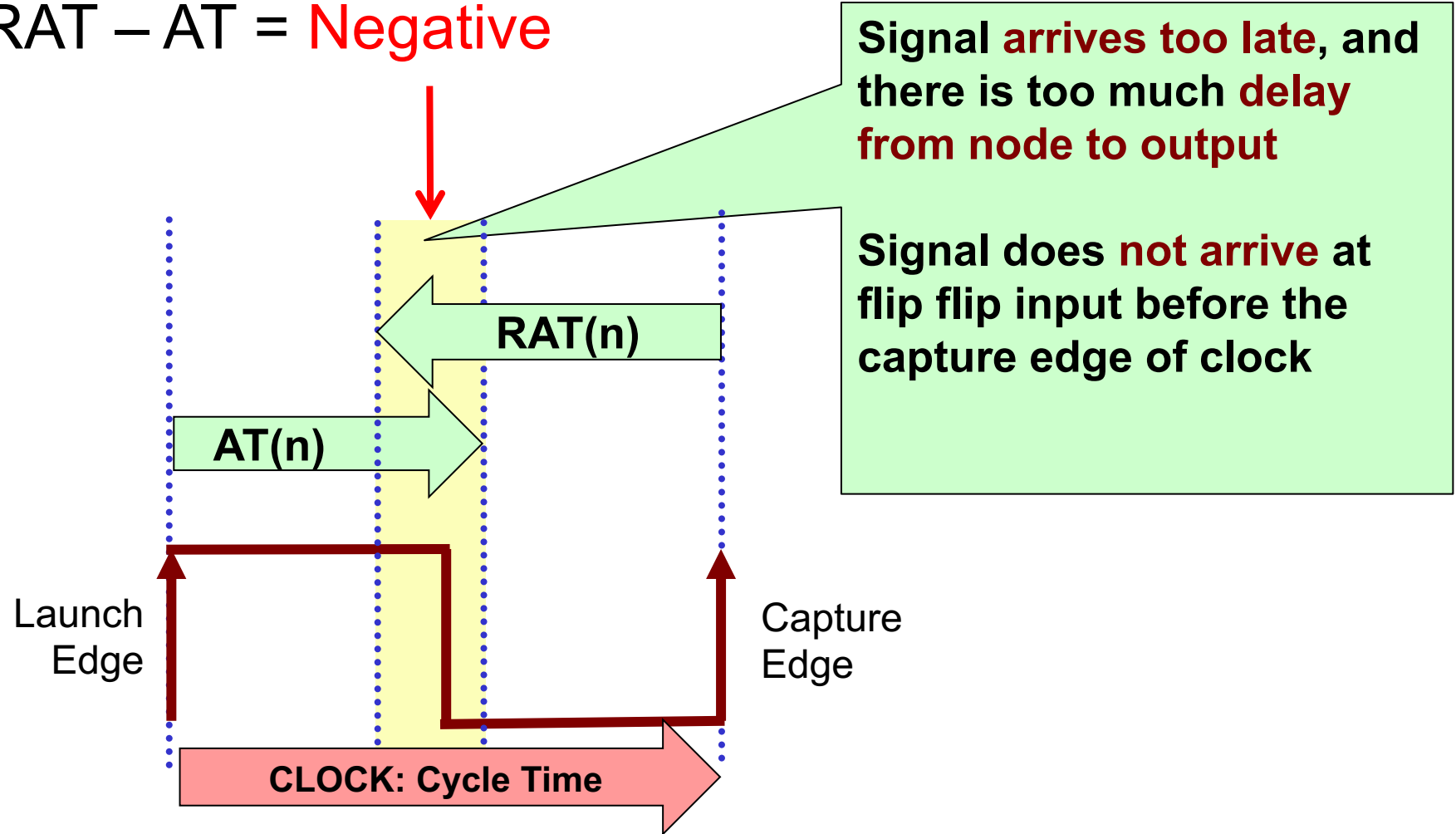
# ATs vs RATs: Look at the Clock Cycle

- Why the difference between ATs and RATs?



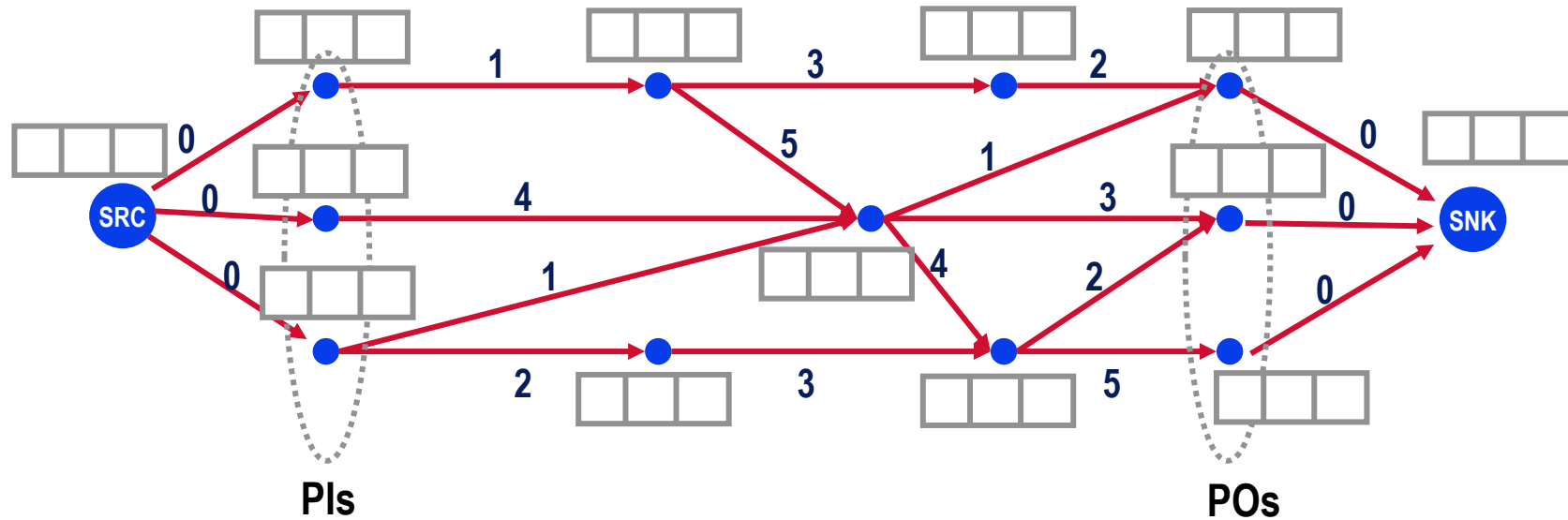
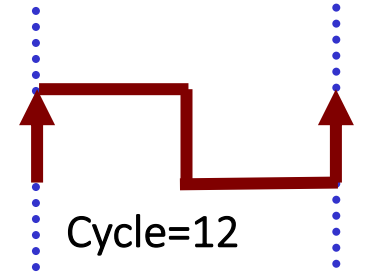
# Bad Things Happen When We See This

- $SLACK = RAT - AT = \text{Negative}$

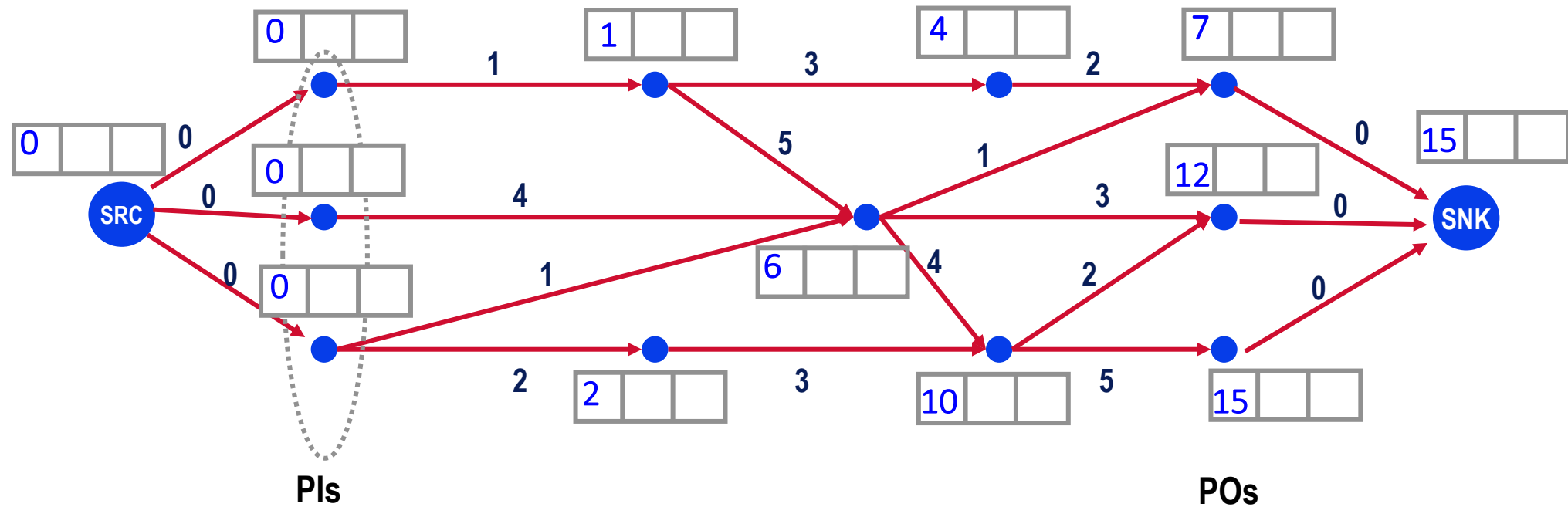
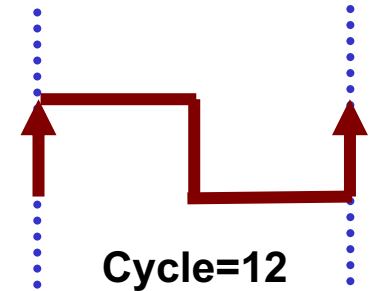


# Let's Do a Bigger Example

- **Delays are on edges; let clock cycle be 12**
  - Compute the min/max delays “by eye” for now
  - **AT**=longest path from SRC **TO** node;
  - **RAT**=(cycle time 12) – (longest path **FROM** node to SNK)
  - **Slack** = RAT - AT

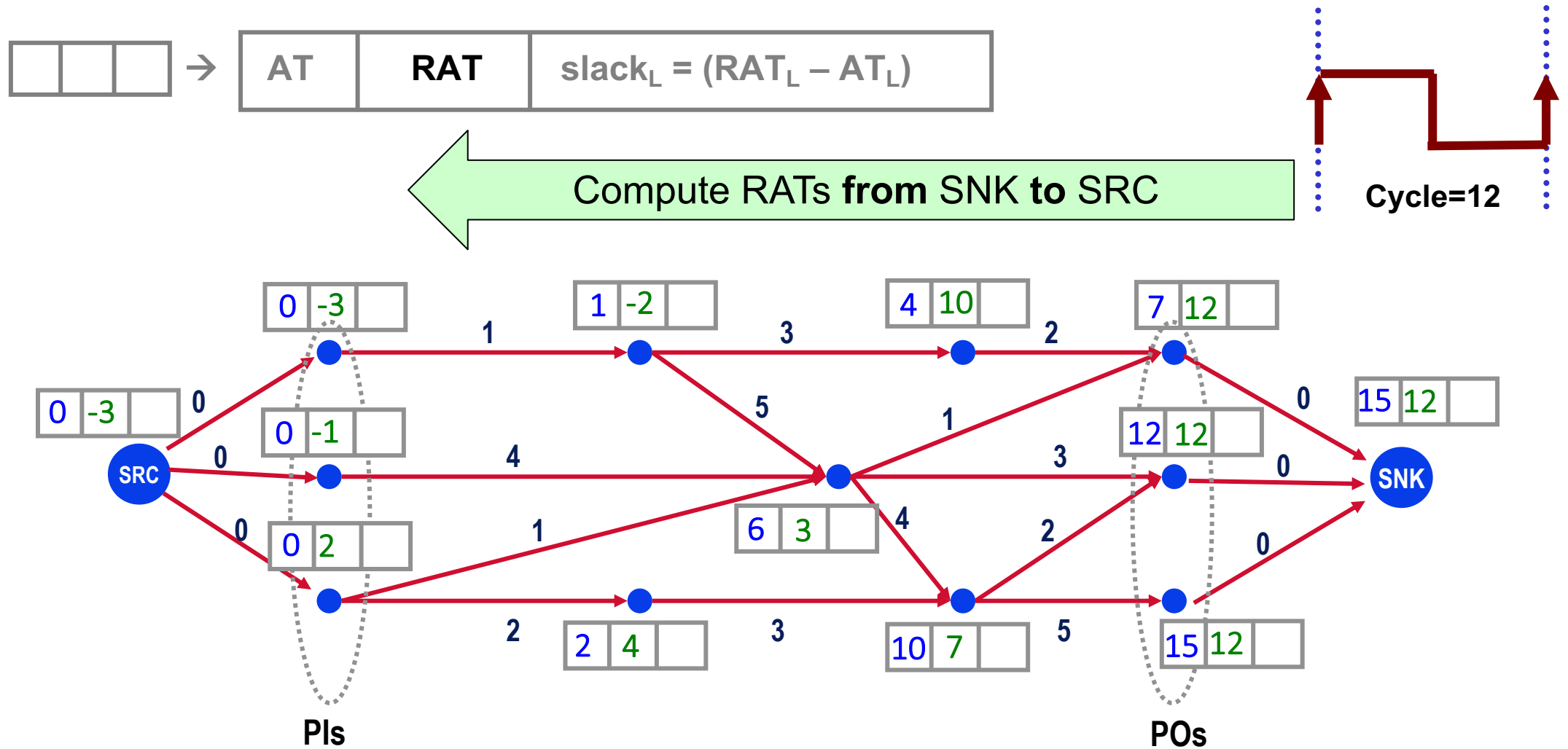


# Compute ATs ...





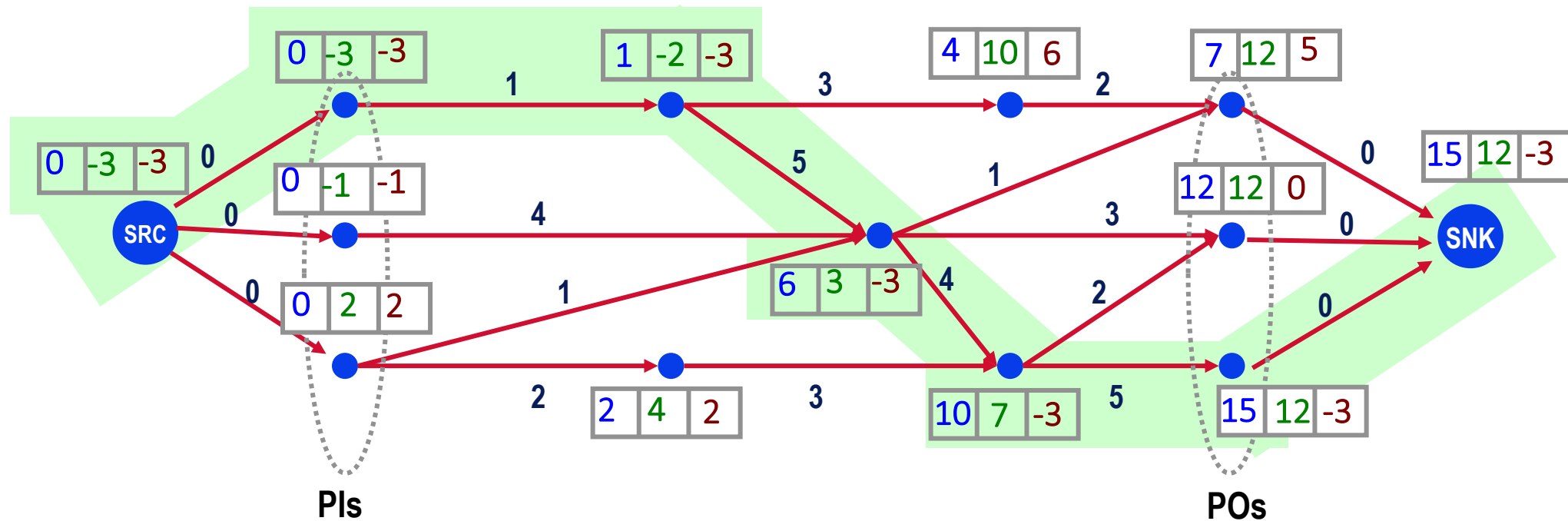
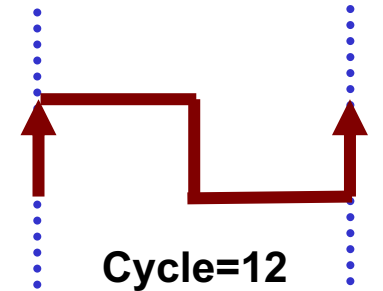
# Compute RATs ...



# Compute Slacks ...



Worst (most negative slack) is **-3**. Trace **worst path**, SRC  $\rightarrow$  SNK



# Debrief

---

- **Look at those slacks**

- A negative slack at an output (PO) means a missed requirement
- A negative slack on internal node n means it feeds a problem PO
  - So, there is a path from n to some problem PO

- **Key: negative slack appears along this entire worst path**

- Your worst timing violation at an output (PO) = the most negative slack value
- You can always trace a path with this slack value back to a PI

- **So, slacks are hugely useful**

- Beyond just knowing what is the worst path; slacks tell us problem gates on this path

# Summary – Happy Thanksgiving!

---

- **We have discussed the timing analysis problem**
- **We have discussed the static timing analysis (STA) model**
- **We have discussed computational models to STA**
- **We have discussed measurement of STA results**

