VE527

Computer-Aided Design of Integrated Circuits

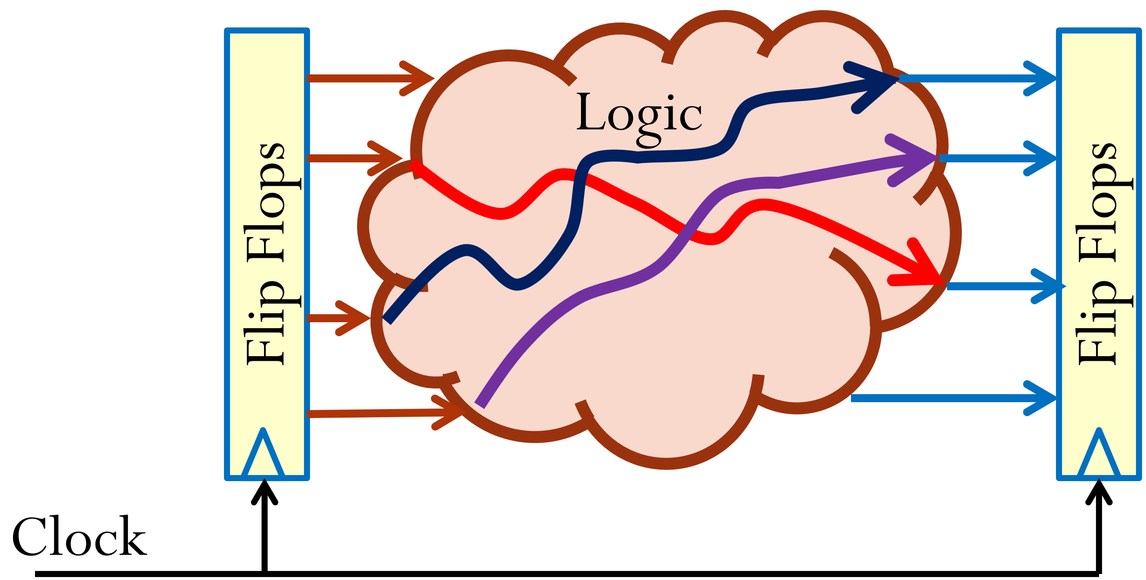
Static Timing Analysis; Interconnect Modeling

Outline

- Static Timing Analysis: Algorithm
- Interconnect Timing
  - Electrical Models of Wire Delay
  - Elmore Delay Model

The Most Typical STA Problem

- Answer this problem: What are all the too-slow paths that violate timing?
- Most useful report:
  - Report paths **in order**, from slowest to fastest.
  - In other words: **Enumerate** these paths, in delay order.



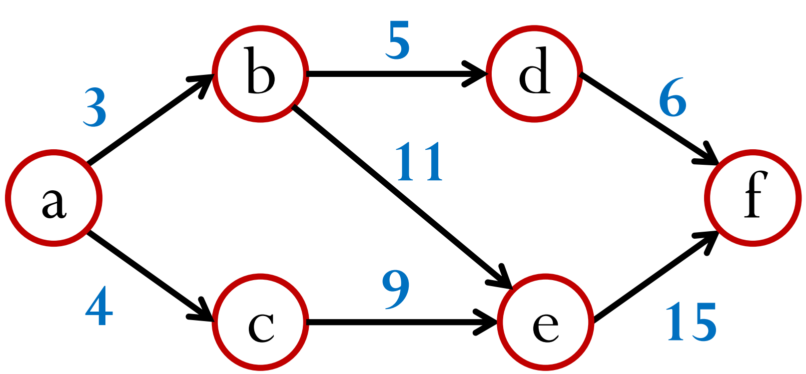
What Do We Need?

- Calculate all the **ATs**.
- Calculate all the **RATs**.
- Calculate all the **Slacks**.
- … do all of these very **efficiently**: Delay graphs are huge!

- **Enumerate** the violating paths, in worst delay order.

Computational Strategy

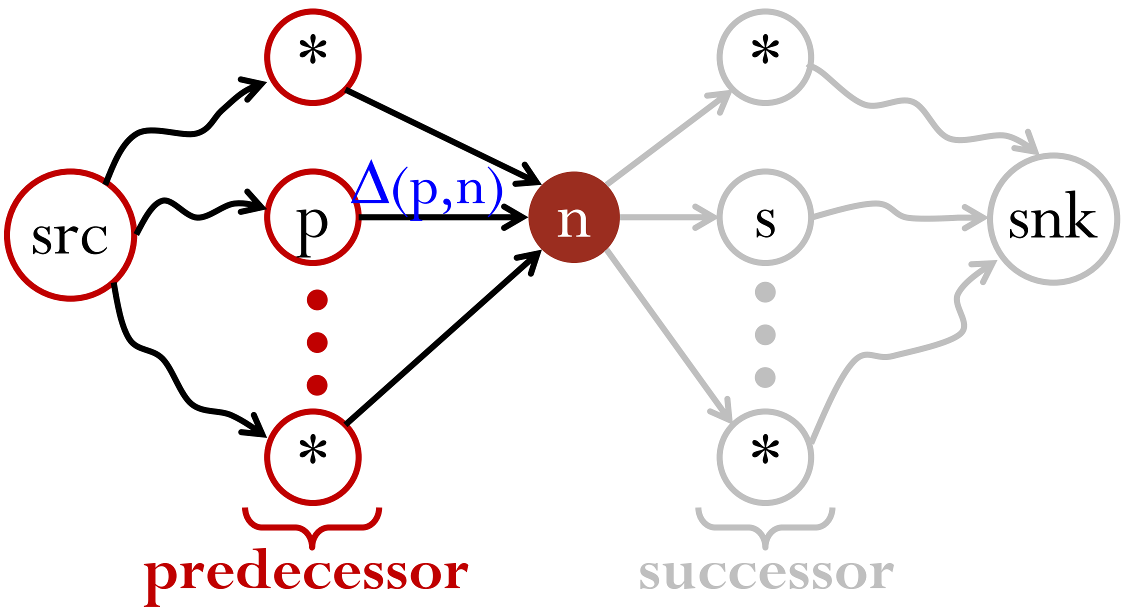
- **Topological sorting** (“**Topsorting**”) the delay graph.
  - Sort the vertices in the delay graph into one single ordered list.
  - Essential property: if there is an edge from p to s, then p appears before s in sorted order.
- Compute ATs by going **forward** through the sorted list.
- Compute RATs by going **backward** through the sorted list.



Legal Topsorting Order
a, b, c, d, e, f
a, b, d, c, e, f

Compute ATs

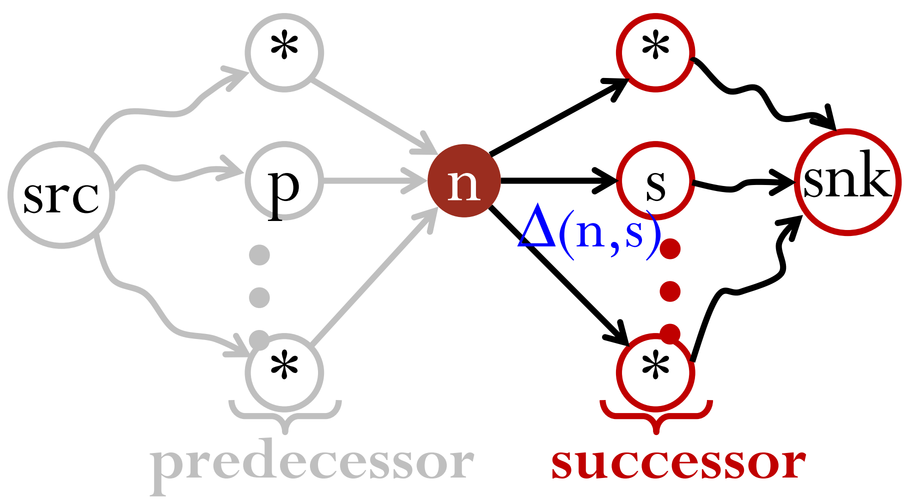
```
computeATs() {
   AT(SRC) = 0;
   foreach ( n in topsort order ) {
      AT(n) = -∞;
      foreach ( node p in pred(n) )
         AT(n) = max( AT(n), AT(p) + Δ(p,n) );
   }
}
```



Compute RATs

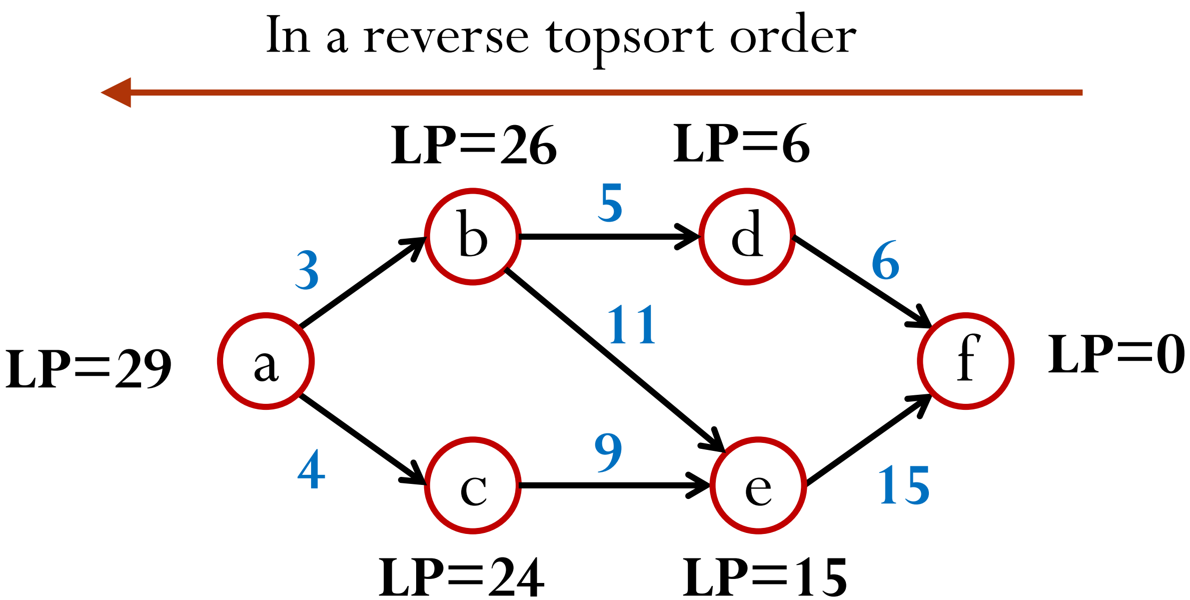
- Pretend all edges are **reversed**, they point from SNK to SRC, and walk graph **backwards**.

computeRATs() {
  RAT(sink) = CycleTime;
  foreach (n in **reverse** topsort order) {
    RAT(n) = ∞;
    foreach (successor **s** in **succ(n)**)
      RAT(n) = **min**(**RAT(n)**, **RAT(s) - Δ(n,s)**);
  }
}



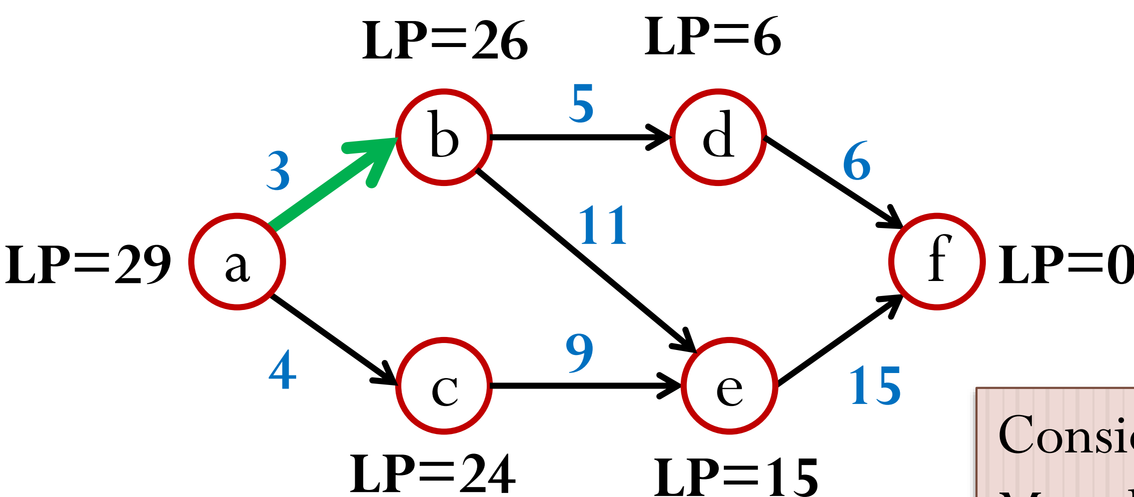
Path Reporting

- If only the longest path, can use slack property: all nodes on longest path have **same worst slack value**.
- How can we find **N worst (longest) paths**?
  - We will use **longest path (LP) to sink** for each node (related to **RAT**)



N-Worst Path Reporting

- We evolve **partial paths**; each partial path stores 3 things: (Path itself, Delay of this path, Max delay **through** this partial path to sink)
  - Max delay through this partial path to sink = Delay of this path + longest path (LP) from last node on the path to sink

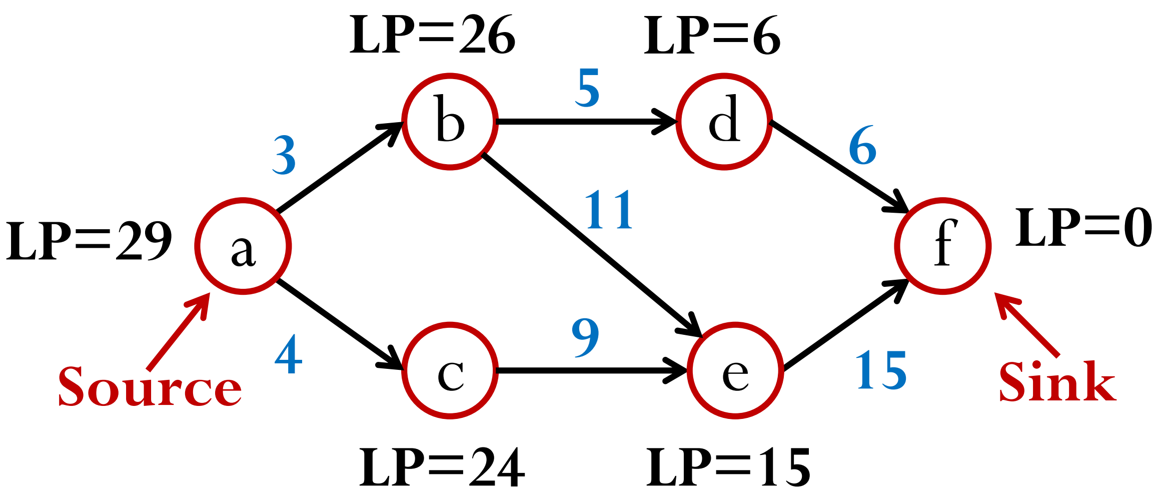


Consider partial path (a,b).
Max delay through it=3+26=29

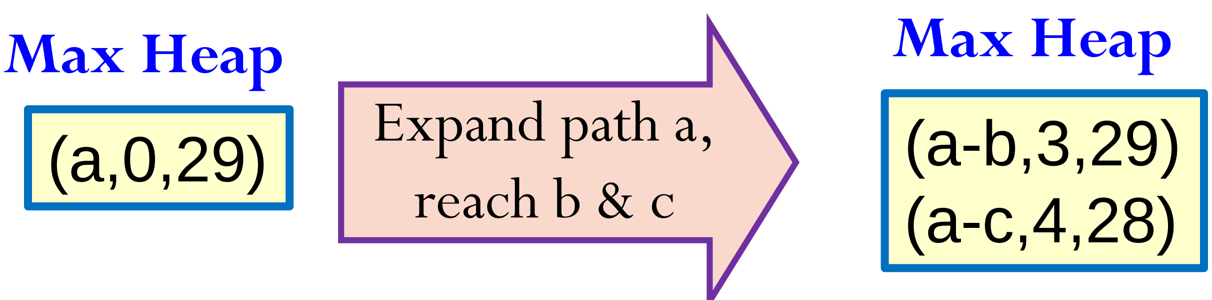
N-Worst Path Reporting

- We store the partial paths in a **max heap**, which is indexed on the **last entry**, max delay through the path (i.e., delay+LP).
  - Initially this heap contains only the **source node**.
- Algorithm is quite simple (and just like maze routing!).
  - **Expand**: Pop the partial path has the largest last entry off the heap
    - Reach target? If its end node is the sink, print out the path
  - **Reach**: Else add each **successor** node to make **new** partial paths, push them back onto the heap, each with (Path, Delay, Delay+LP) labeled.
  - Go pop next partial path until N paths are reported.

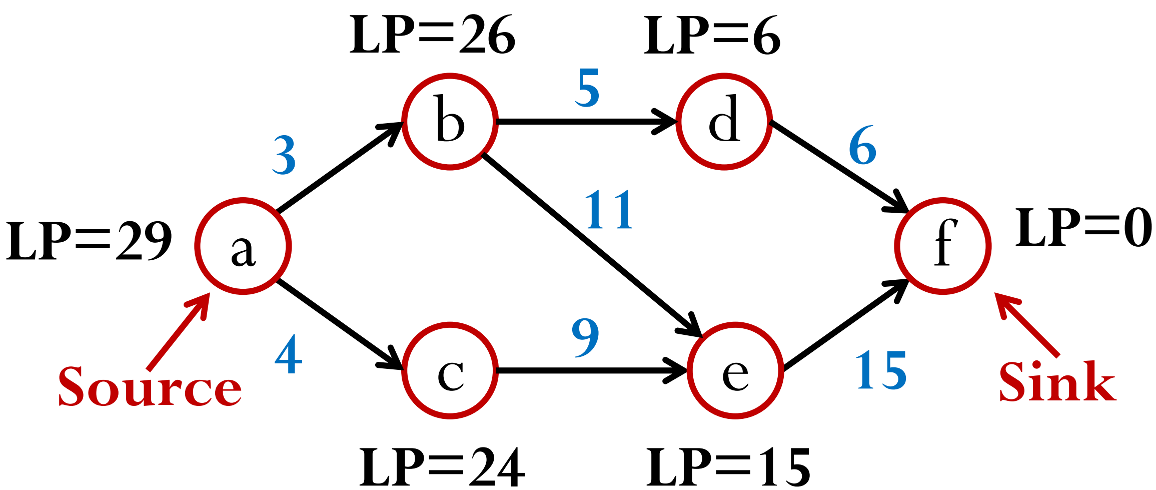
Worst Case Path Reporting: Example

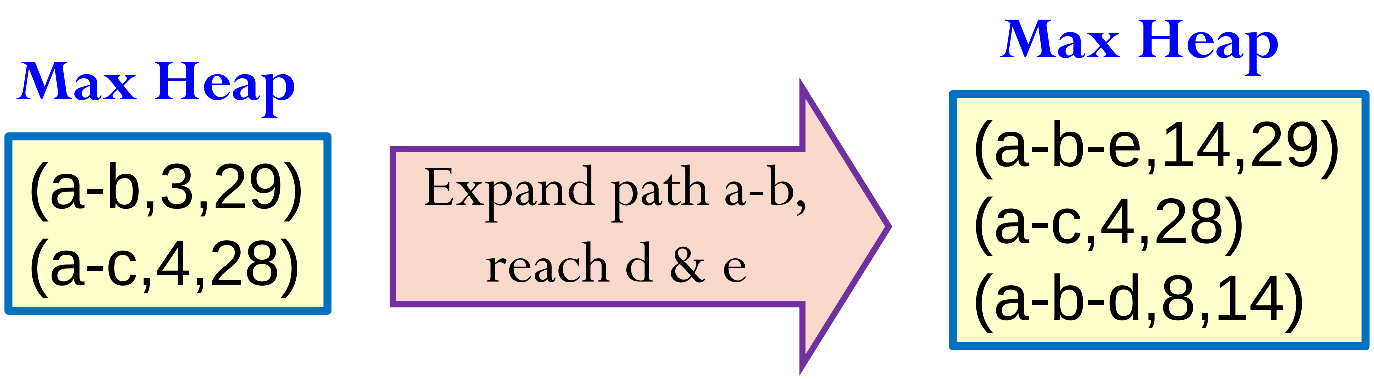


- Max heap entry of the form (Path, Delay, Delay+LP)
  - Initially, heap contains only the **source node**.

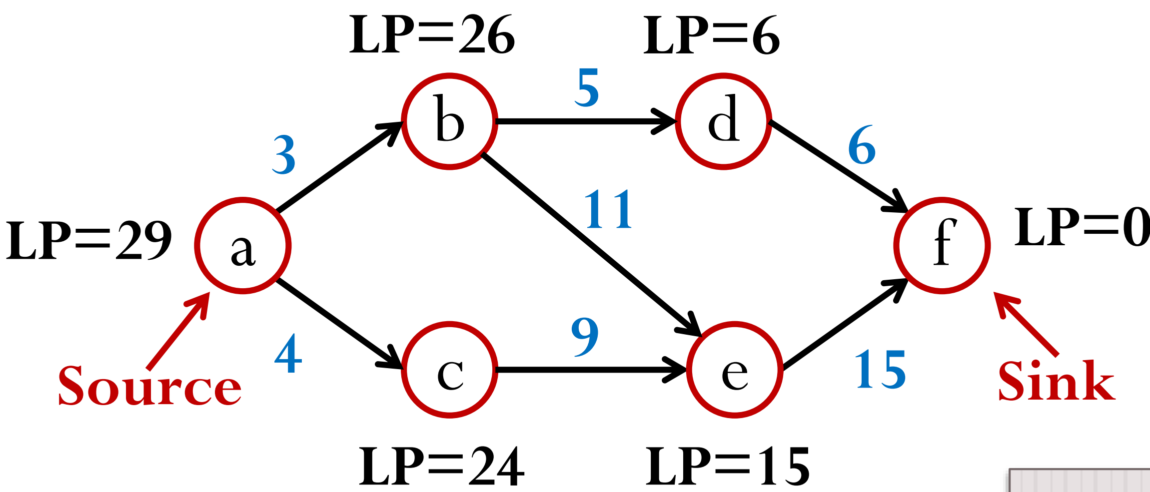


Worst Case Path Reporting: Example





Worst Case Path Reporting: Example



f is **sink**! Report 1st worst path a-b-e-f, with delay=29

Max Heap

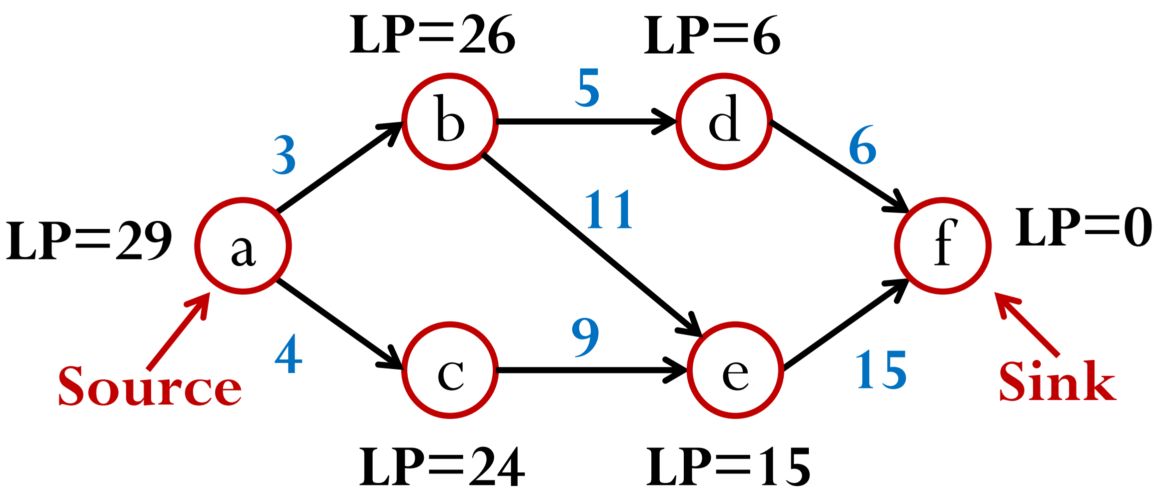
(a-b-e,14,29)
(a-c,4,28)
(a-b-d,8,14)

Expand path a-b-e, reach f

Max Heap

(a-c,4,28)
(a-b-d,8,14)

Worst Case Path Reporting: Example



Max Heap

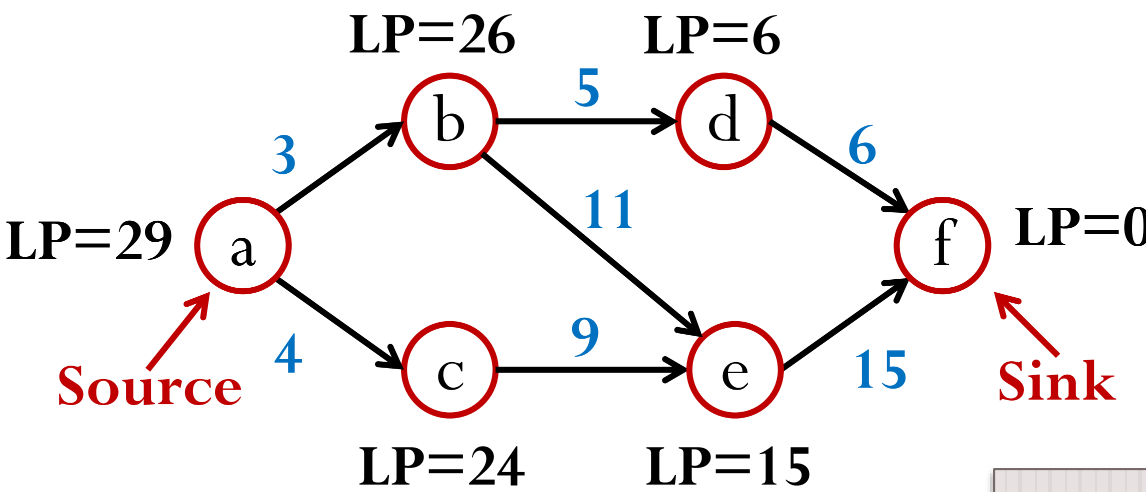
(a-c,4,28)
(a-b-d,8,14)

Expand path a-c, reach e

Max Heap

(a-c-e,13,28)
(a-b-d,8,14)

Worst Case Path Reporting: Example



f is **sink**! Report 2nd worst path a-c-e-f, with delay=28

Max Heap

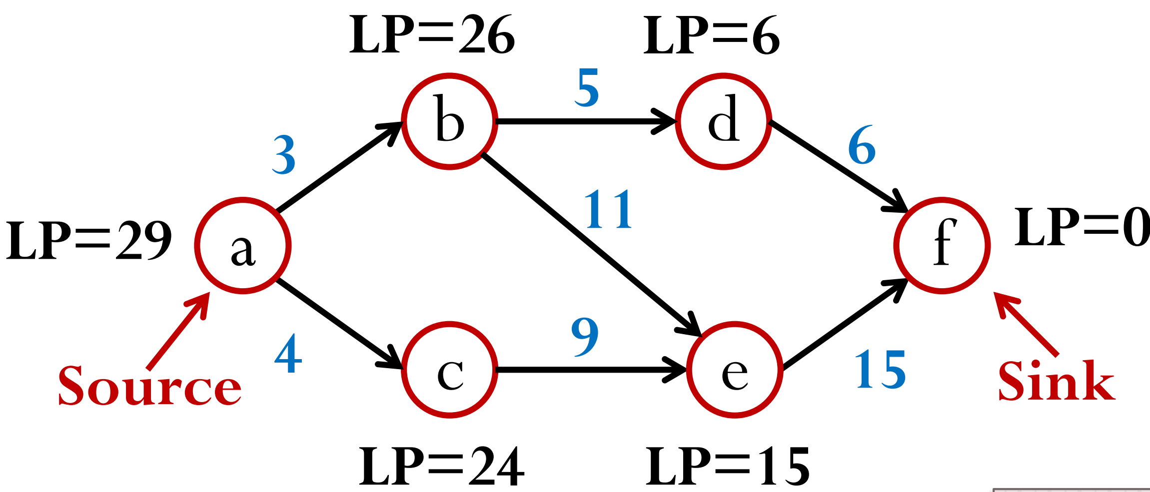
(a-c-e,13,28)
(a-b-d,8,14)

Expand path a-c-e, reach f

Max Heap

(a-b-d,8,14)

Worst Case Path Reporting: Example



Max Heap

(a-b-d,8,14)

Expand path a-b-d, reach f

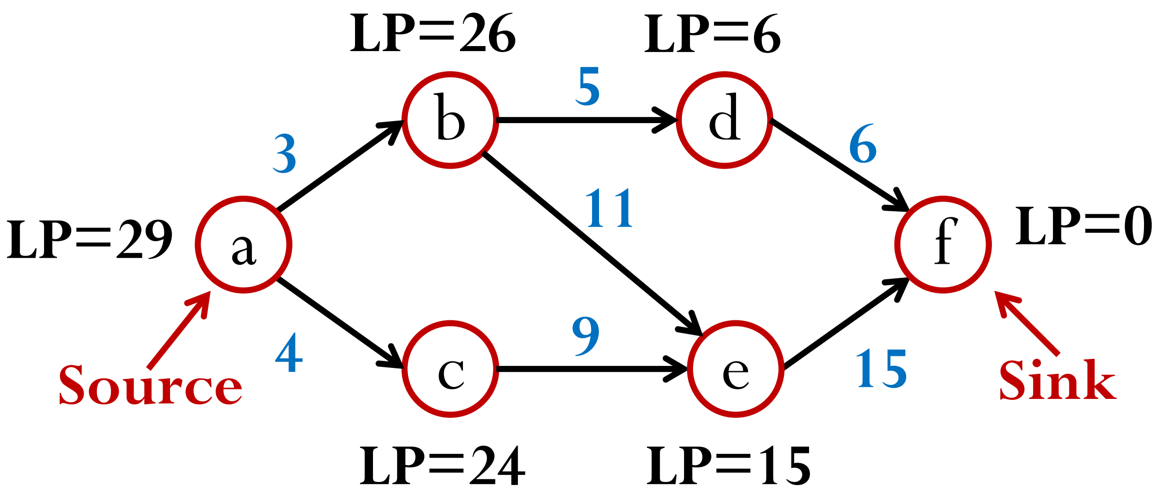
f is **sink**! Report 3rd worst path a-b-d-f, with delay=14

Max Heap

(EMPTY)

Done!

Worst Case Path Reporting: Example



- We find three paths:
  - a-b-e-f, delay = 29
  - a-c-e-f, delay = 28
  - a-b-d-f, delay = 14.

<u>Note</u>: only 3 possible paths from source to sink in graph, so we found them correctly in **delay order**!

Static Timing Analysis: Summary

- STA is a **very important** step in design of complex ASICs.
  - It's a critical "**sign off**" step, which means: you don't get to fabricate unless you pass.
- Several big ideas
  - **Gate level delay** models matter, and can be pretty complex in real world.
  - Logical ≠ Topological path analysis (i.e., STA).
  - Build **delay graph**, calculate **ATs**, **RATs**, **slacks** recursively.
  - Concept of **slack** is **<u>important</u>**: lets us locate worst paths, and problem gates on path.
  - A similar idea to maze routing lets us find **worst paths in delay order**.

Static Timing Analysis: Aside

- STA is a **huge** topic – several things we did not cover.
- **STA for sequential elements**
  - How do we model flip flops and latches, so we can verify, e.g., that setup and hold times are met? More tricks with delay graph.
- **Early mode versus late mode timing**
  - Our development was only so-called late mode timing, where we care about longest path. Early mode focuses on shortest paths, and is critical for more advanced timing.
- **Incremental STA**
  - In practice, you change 10,000 gates out of 1,000,000 gates, you don't want to **redo** the whole STA analysis. Advanced methods can update **incrementally**.

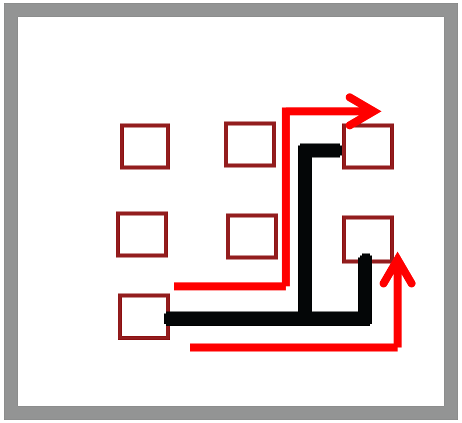
Outline

- Static Timing Analysis: Algorithm

- Interconnect Timing
  - Electrical Models of Wire Delay
  - Elmore Delay Model

Interconnect (Wire) Delay Modeling

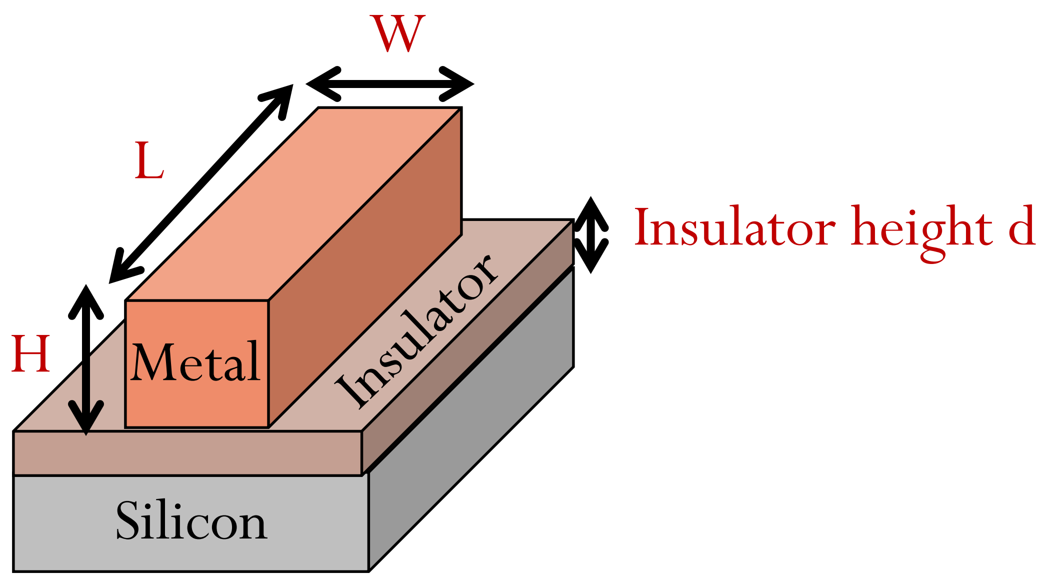
- You place the logic. Pins are put at a certain distance apart.
- You route the wires. Each wire has an input-to-output **delay**.
- Questions:
  - Where does the delay come from?
  - How **accurately** can we **predict** this delay?
  - How **efficiently** can we **model** this delay for use in layout synthesis or STA?



Delay!

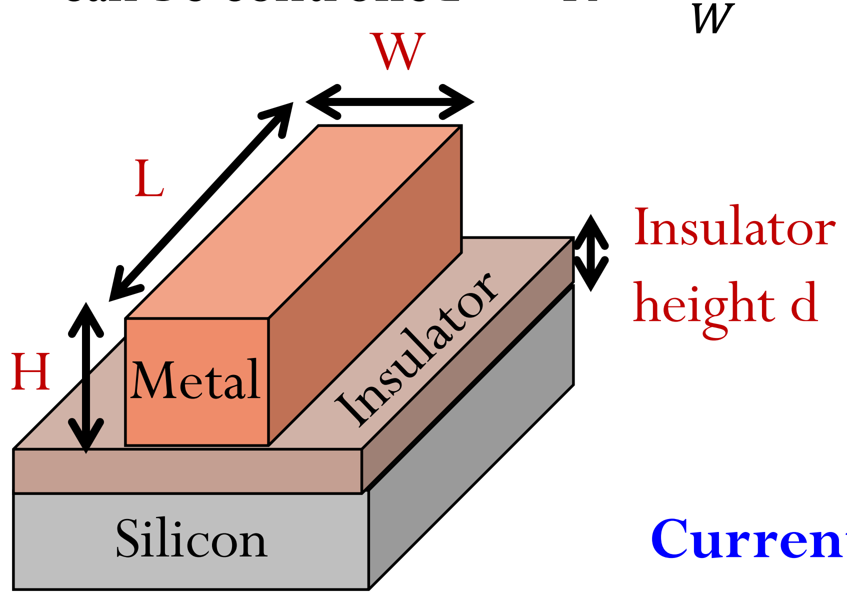
Interconnect Model

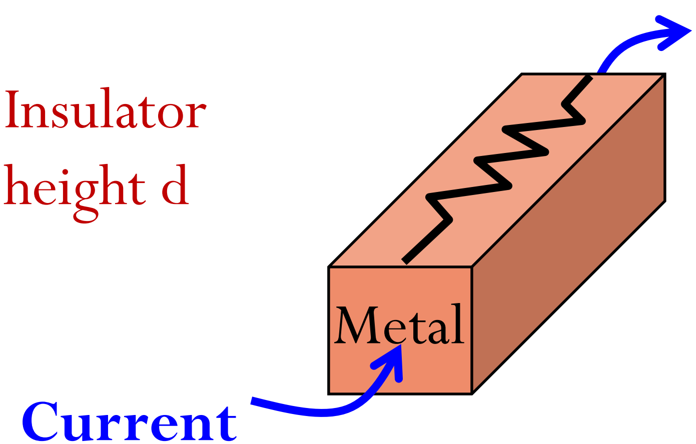
- We model interconnect as an **electrical circuit** and analyze it as a circuit.
- The model parameters depend critically on **exact geometry** of the wired net.
- Most popular interconnect model used in layout applications: **RC trees**.



Interconnect Model: Resistance

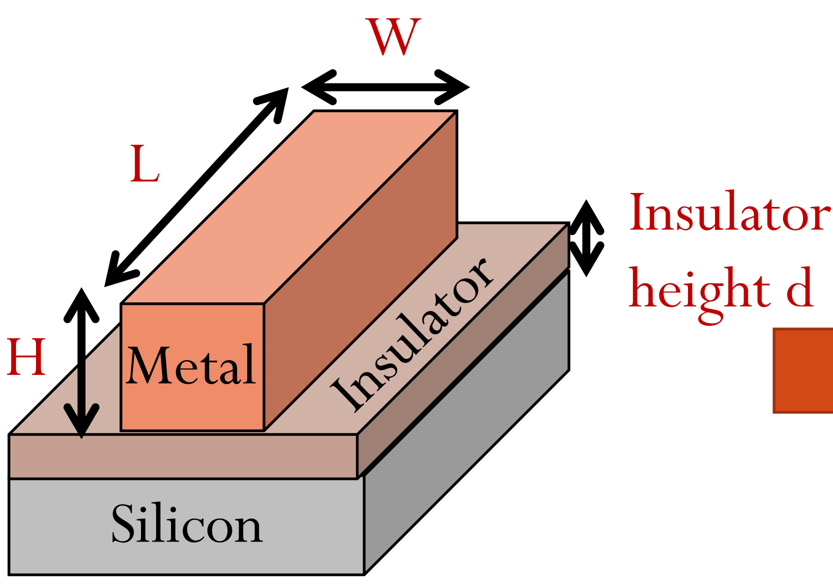
- Metal wire has **resistance R** to current flowing down its length.
  - **In physics**: $R = \frac{\rho L}{S} = \frac{\rho L}{WH}$
  - **In ASIC design**: H is a manufacturing factor. Only W and L can be controlled $\Rightarrow R = \frac{rL}{W}$

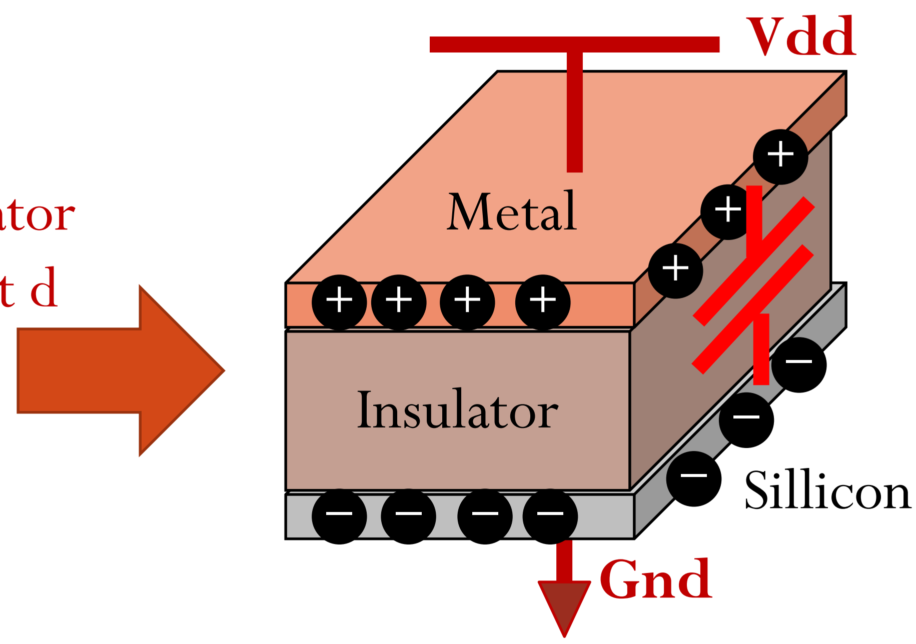




Interconnect Model: Capacitance

- Metal wire has **capacitance C** to silicon substrate, with insulator between.
  - **In physics**: $C = \frac{\varepsilon W L}{d}$
  - **In ASIC design**: d is a manufacturing factor. Only W and L can be controlled $\Rightarrow$ $C = cWL$

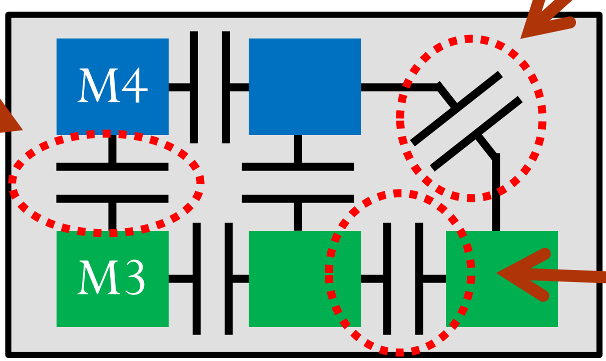




Aside: About Real Capacitance (Cap)

- Note: this model is very **simplistic**.
- You really get capacitance between **any pair of conducting surfaces**.
  - So, in a multi-layer metal process you get caps **between all the layers**.

Overlap cap between 2 adjacent wires on different layers

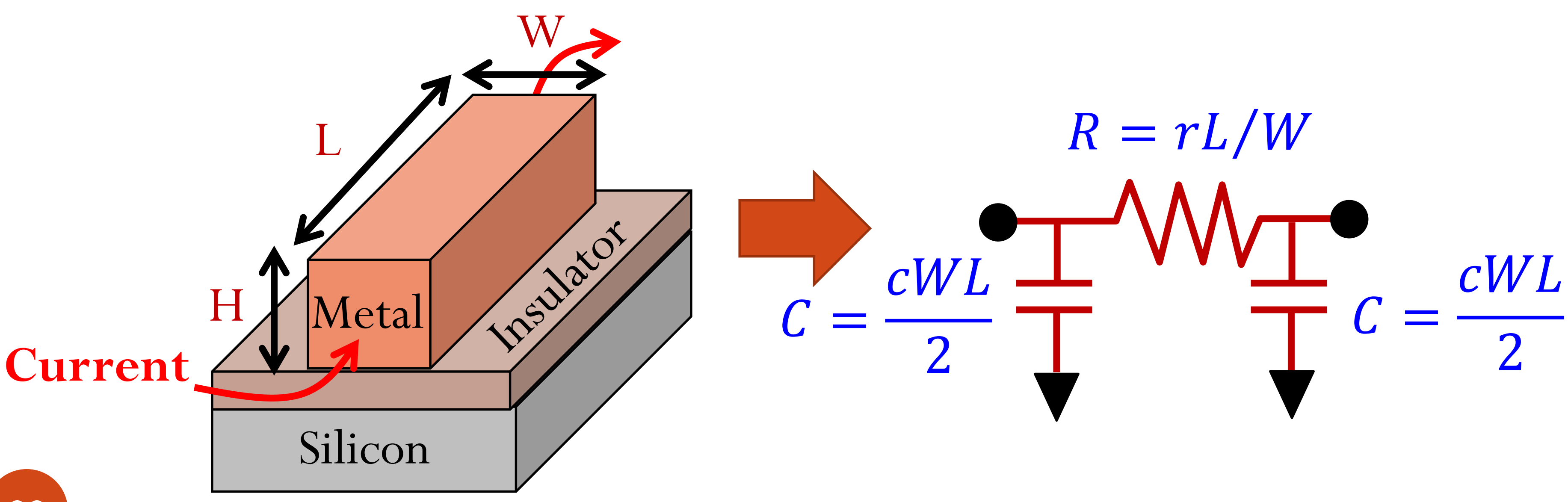


Sidewall fringe cap from side of one layer to the conductors below it

Fringe cap between 2 adjacent wires on the same layers.

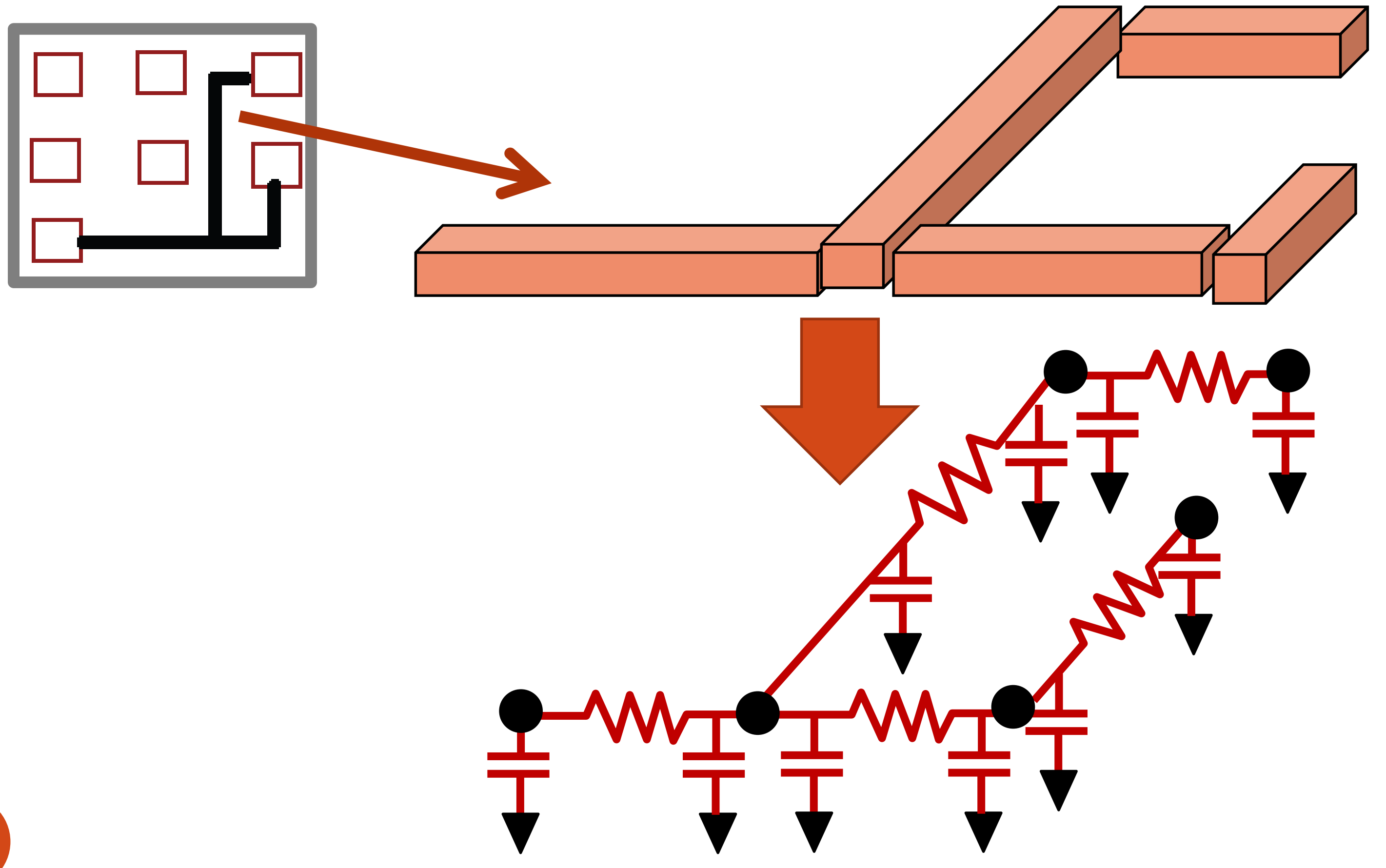
Typical Interconnect Model: π Model

- Accounts for the resistance R and the capacitance C of wire segment.
- It is a **small** model: only need 2 numbers.
- It is a **symmetric** model: **split** capacitance into two halves.



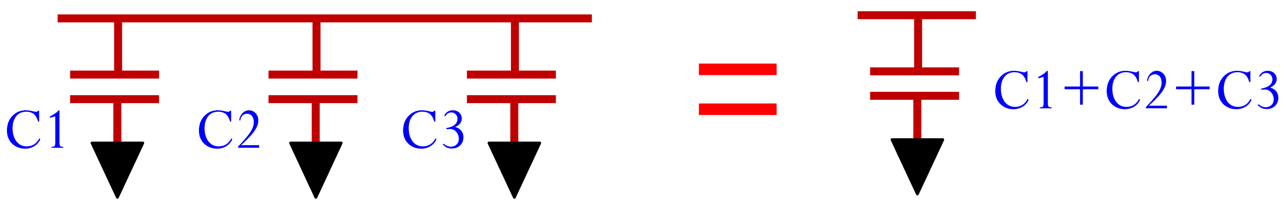
From Wire Segments to RC Tree

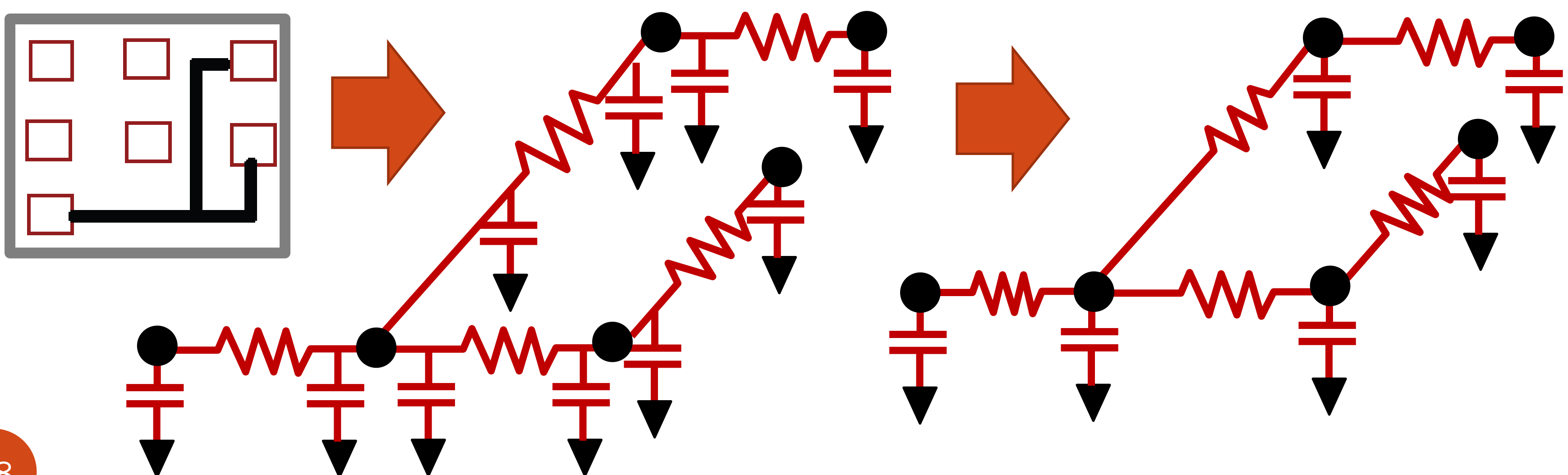
- Big idea: Replace **every** straight wire segment with **π model**.



From Wire Segments to Final RC Tree

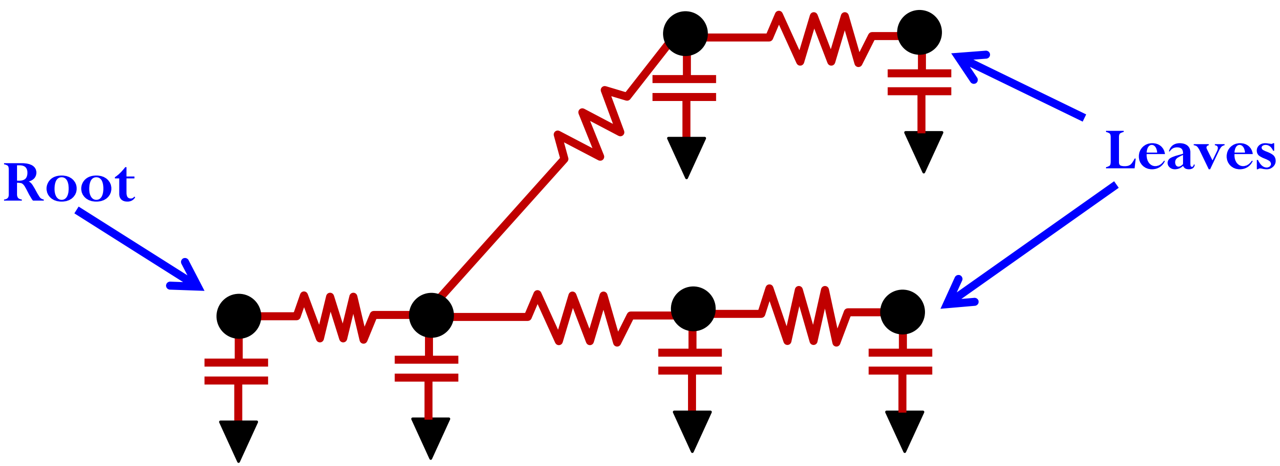
- Simplification: Recall a rule from basic circuits:
  - **Parallel** capacitors can be replaced by 1 capacitor with ΣC_i





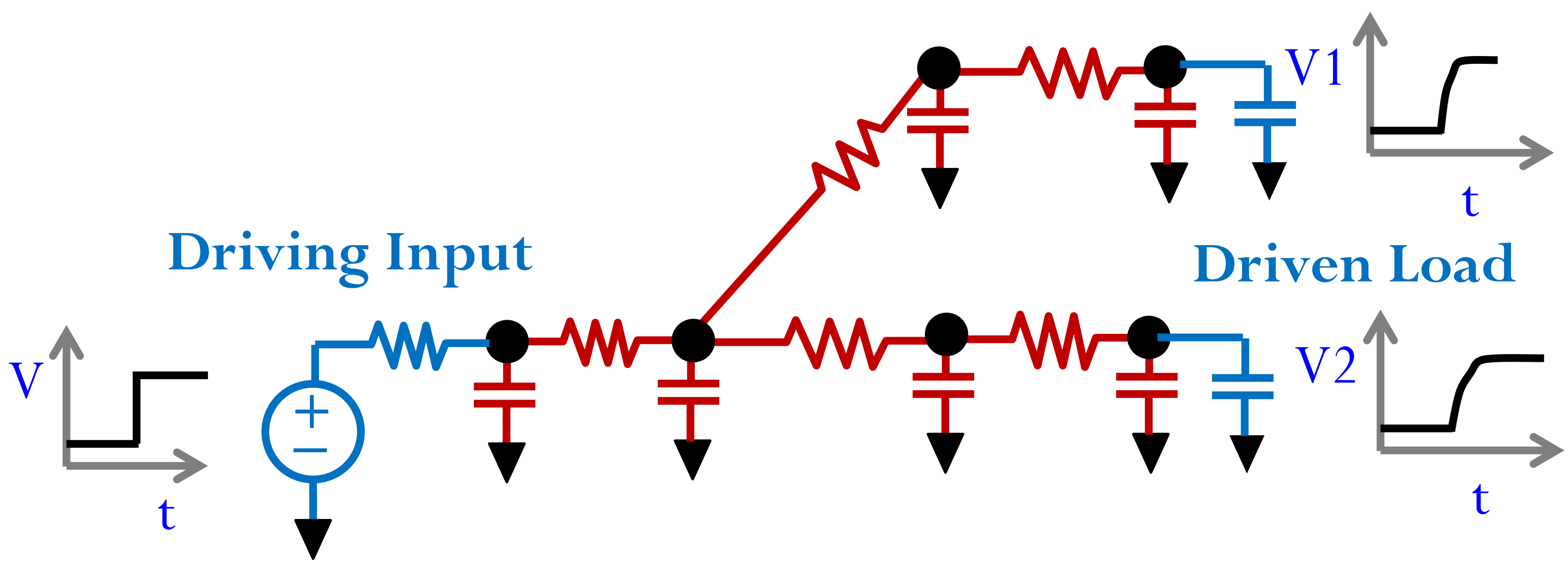
General Form of RC Tree

- Capacitors "**hanging off**" all tree nodes.
- Tree edges are **resistors**.
- **Root** of tree is where signal is input.
- **Leaves** of tree are the driven outputs.

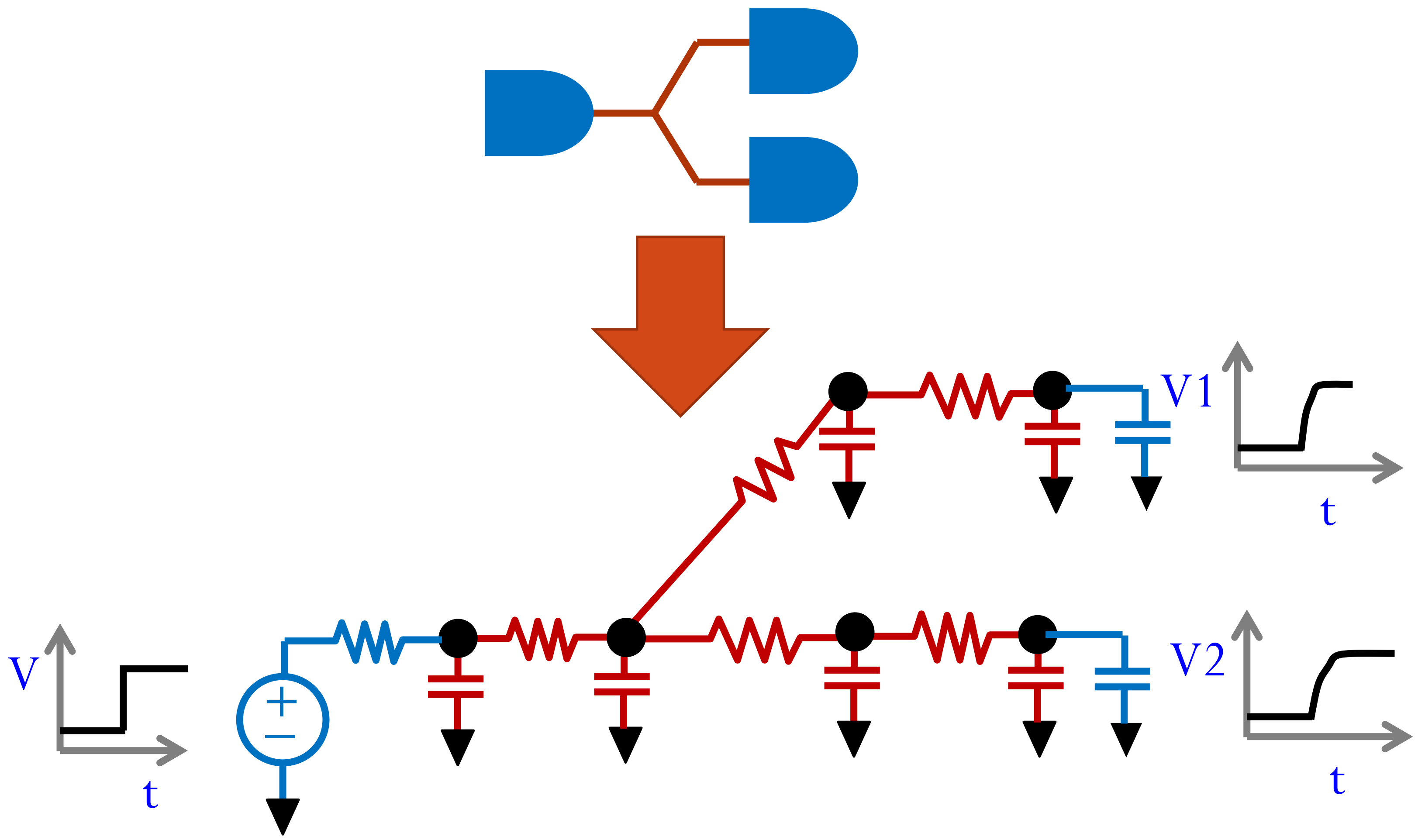


Model the Driving and Driven Gates

- We also need to model **driving gate** and **driven gates**.
  - **Voltage source** + **resistor** as input at root. This models **driving** gate.
  - **Capacitor** as load at each leaf. Each models a **driven** gate.



Summary: RC Tree Is Built from Gates + Wires



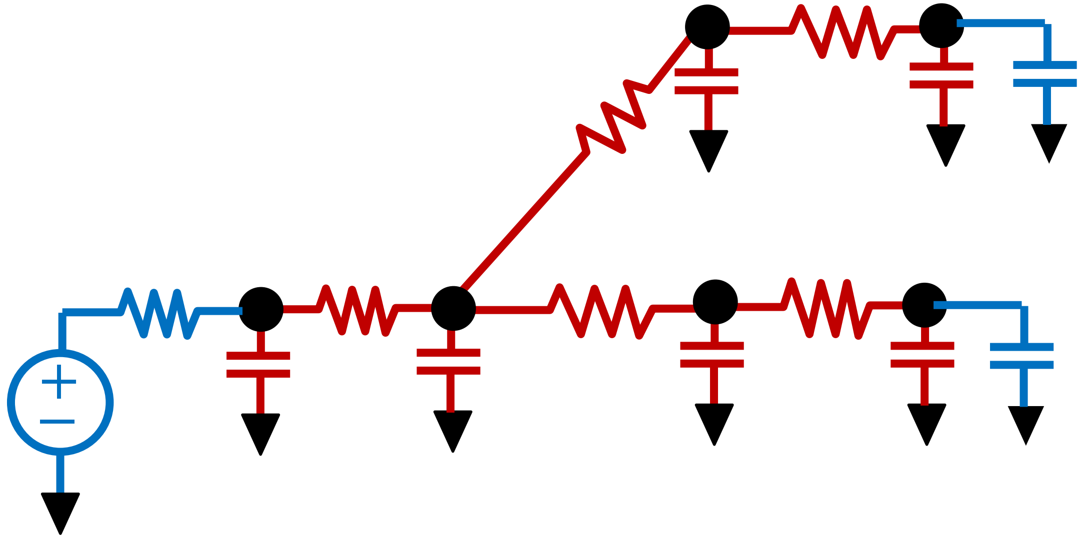
Outline

- Static Timing Analysis: Algorithm

- Interconnect Timing
  - Electrical Models of Wire Delay
  - Elmore Delay Model

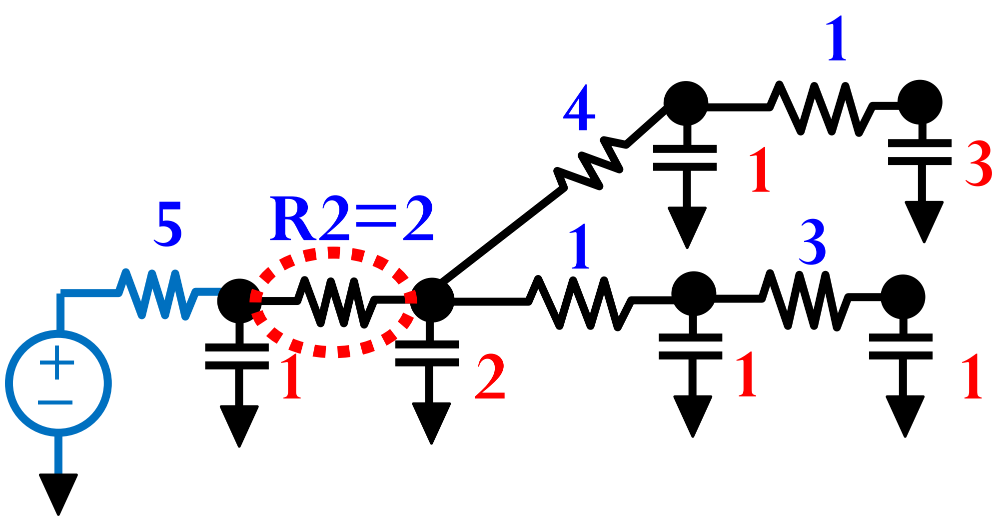
Calculating Delay

- How to calculate a **delay** number for **each output** of tree?
- Famous formula: **"Elmore" delay**
  - Derived in 40s for circuits applications.
  - Resurrected in 80s by Penfield et al. for RC trees.
  - A very **simple**, but very **useful** computational recipe.



Elmore Delay τ: Tree Walk Computing Recipe

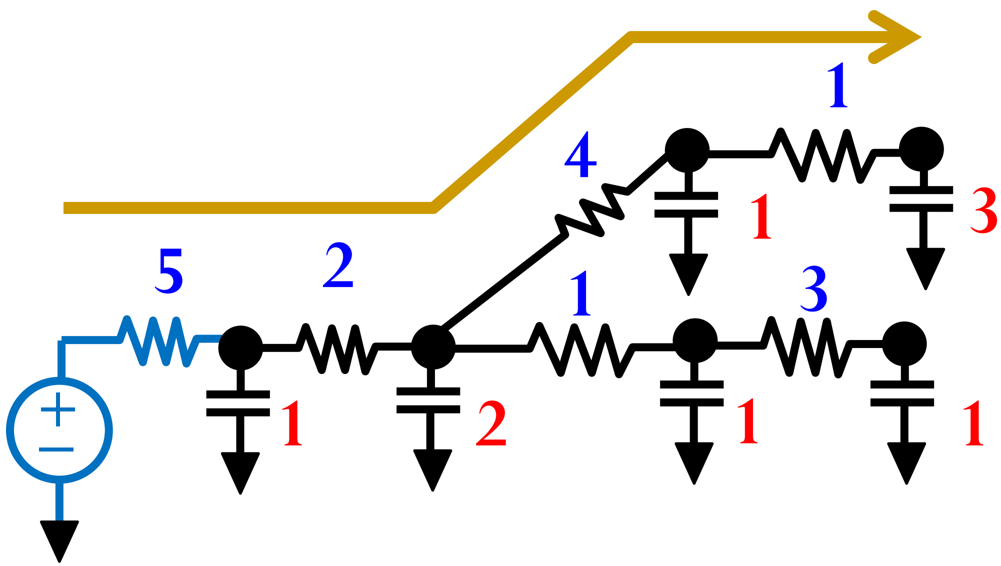
- Set $\tau = 0$. Walk down path of resistors from **Root** to **Leaf** where you want to calculate delay.
- At each resistor, let
  $\tau = \tau + R \cdot \Sigma$ (all capacitors **downstream**)
  - **Downstream capacitor** = any C that is reachable in tree below this resistor.



Example: at R2 resistor in RC tree, the downstream capacitors are, 2, 1, 1, 3, 1

Calculating Elmore Delay: Example

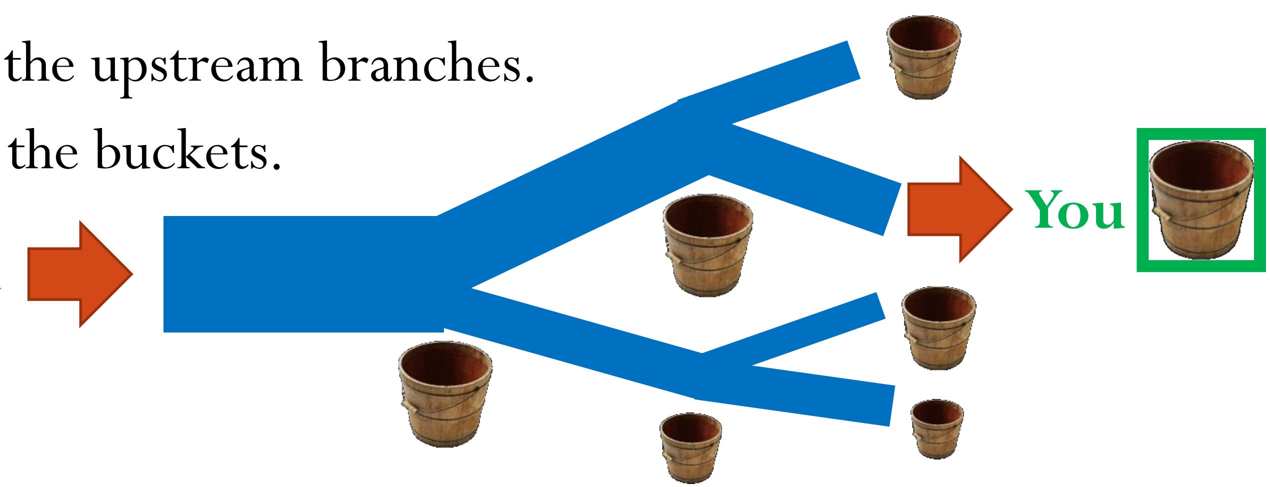
- Set $\tau = 0$. Walk down path of resistors from **Root** to **Leaf** where you want to calculate delay.
- At each resistor, let $\tau = \tau + R \cdot \Sigma$ (all capacitors downstream)



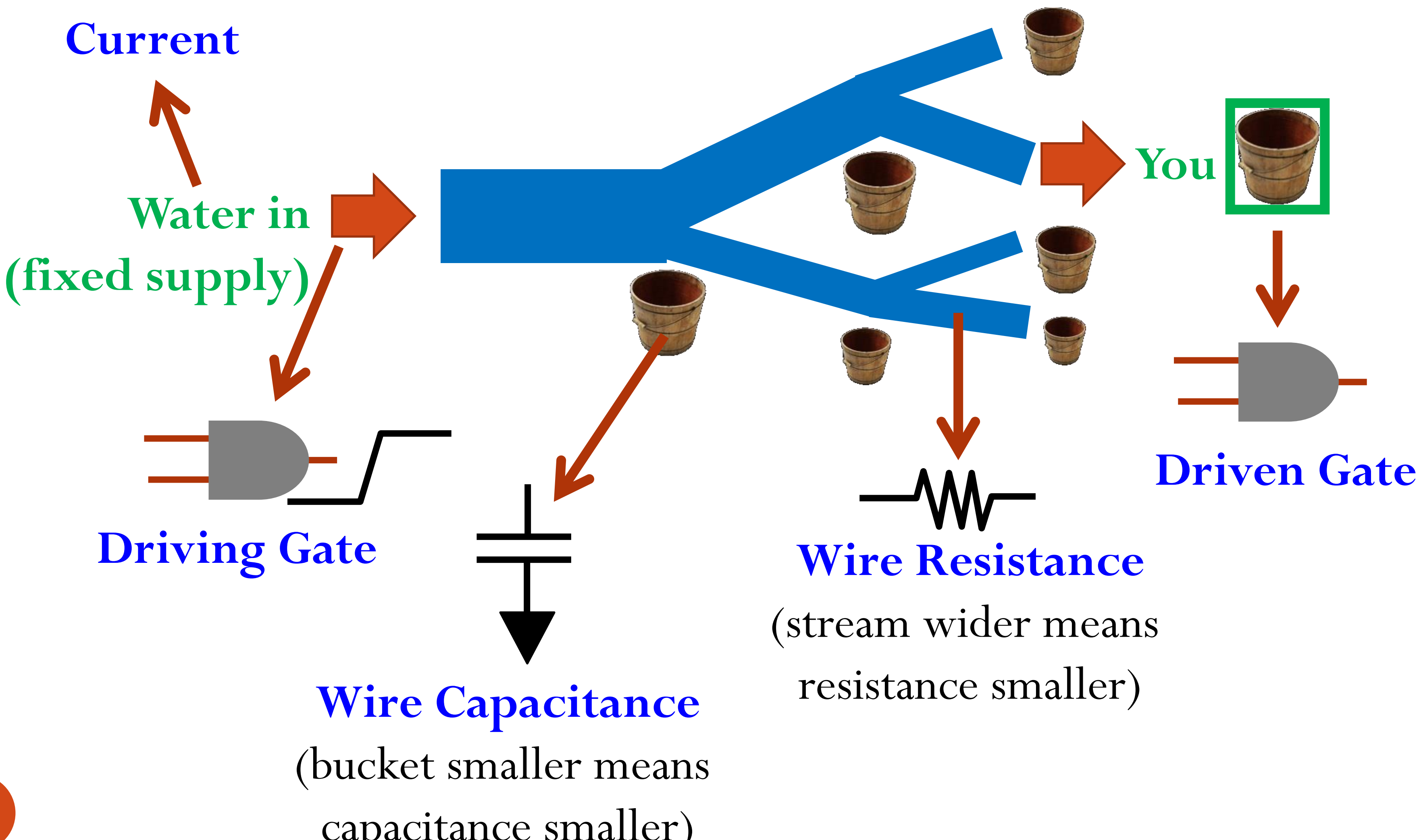
$$
\begin{aligned}
\tau &= 0 \\
&+ 5\times(1+2+1+3+1+1) \\
&+ 2\times(2+1+3+1+1) \\
&+ 4\times(1+3) \\
&+ 1\times3 \\
&= 45+16+16+3 \\
&= 80
\end{aligned}
$$

Insight: Stream Analogy

- Think of RC tree like branching stream, current like water.
  - Goal: You are downstream, trying to fill your bucket. How **fast** can you fill it?
  - However, at **every** branch point, somebody else has a **bucket**.
  - The **farther** you are downstream, the **less** water you get from upstream.
- What matters here?
  - **Width** of the upstream branches.
  - **Size** of all the buckets.

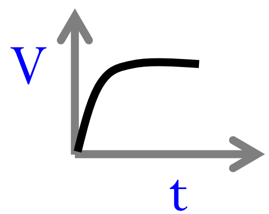


From Stream to Circuit



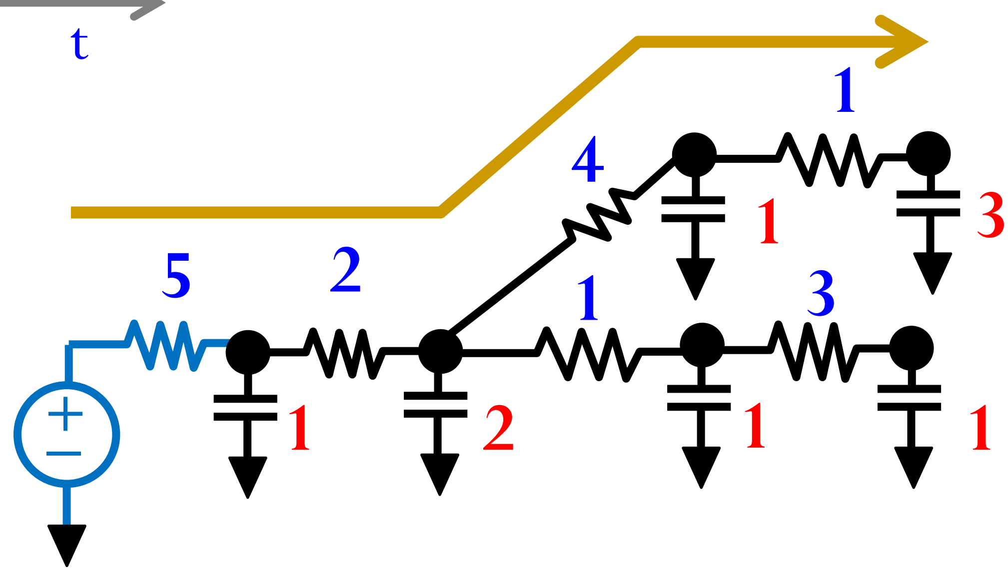
Circuit Aside: What Is Elmore Delay?

- If you want to model the path from input to an output as a simplified circuit with **exactly one** R and **one** C, the **best approximation** you have will have its R•C = Elmore delay.
  - Note: for a circuit with one R and one C, R•C is the **time constant** τ.



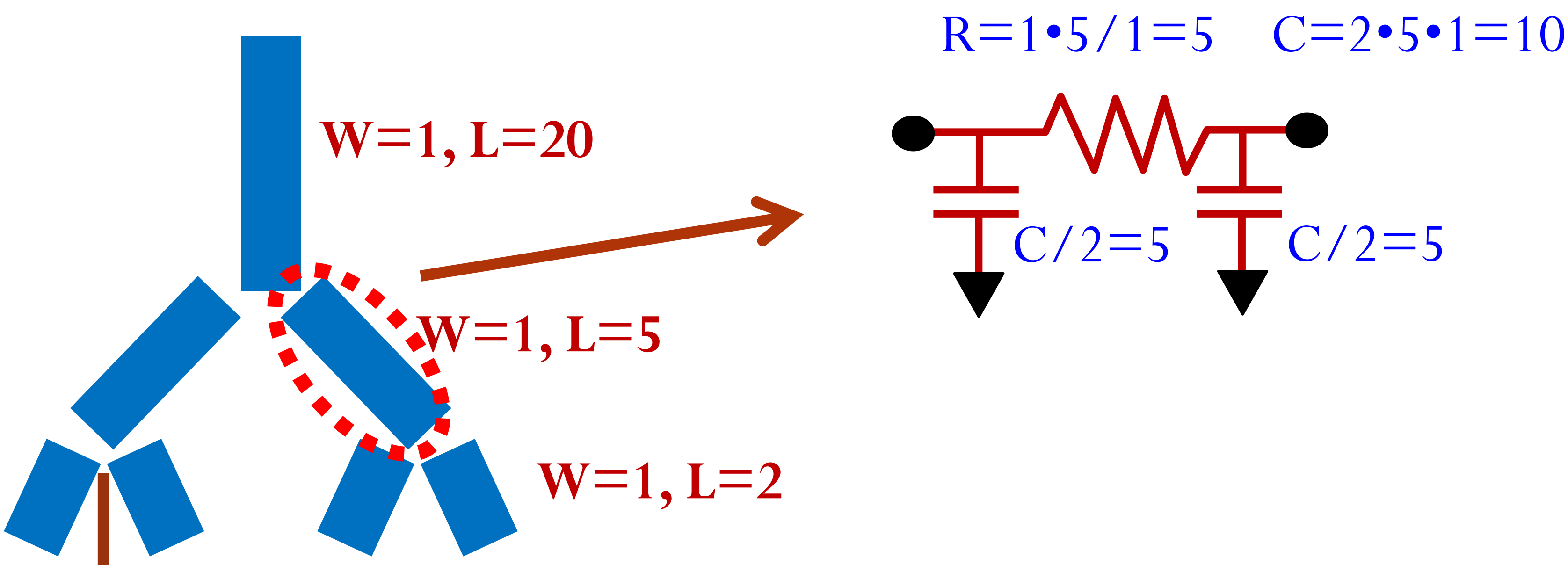
$$V = V_{in}(1 - e^{-t/\tau})$$

Elmore delay is the time constant of the best approximating circuit with one R and one C



Elmore Delay Example

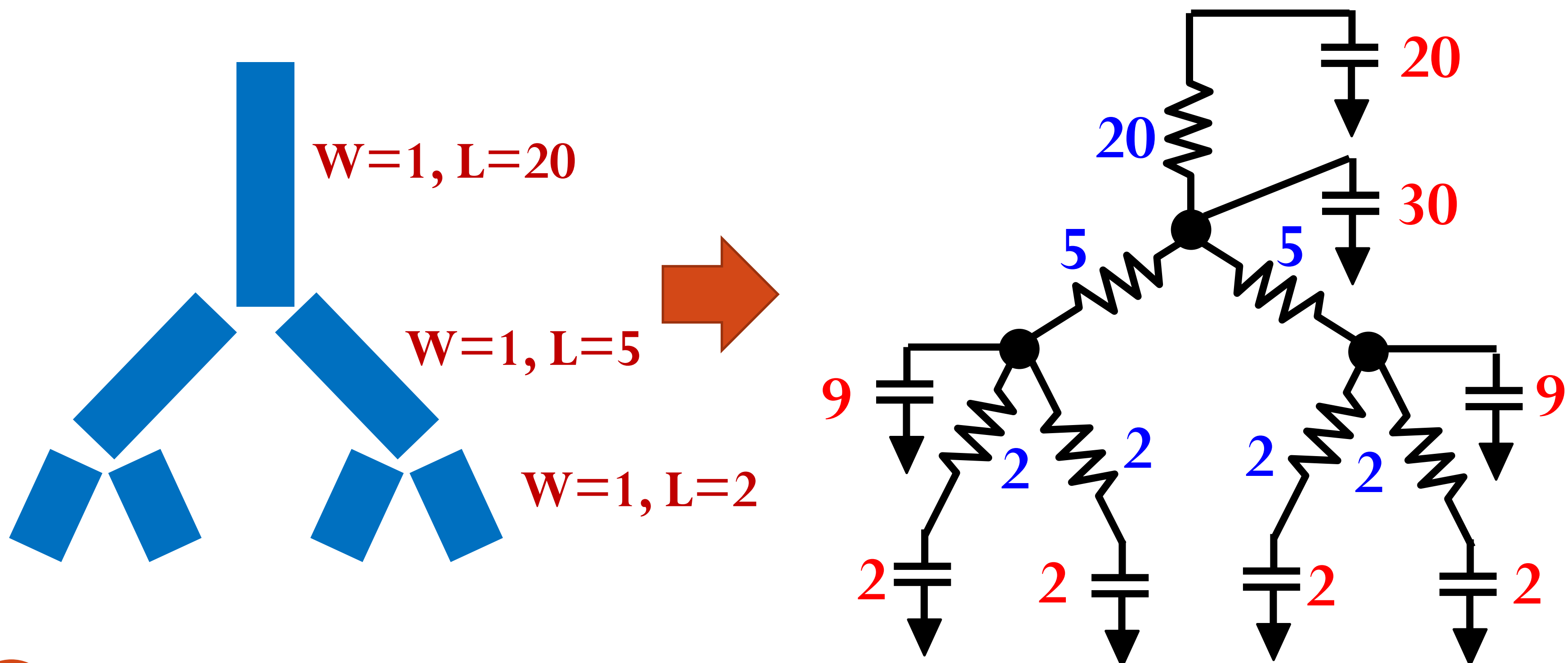
- Simple tree with 4 leaf nodes
  - For each segment, total R=rL/W, C=cWL.
  - Electrical parameters: r=1 , c=2.



Remember: **Add** C's at each node in the tree! **3** C's to add at this node!

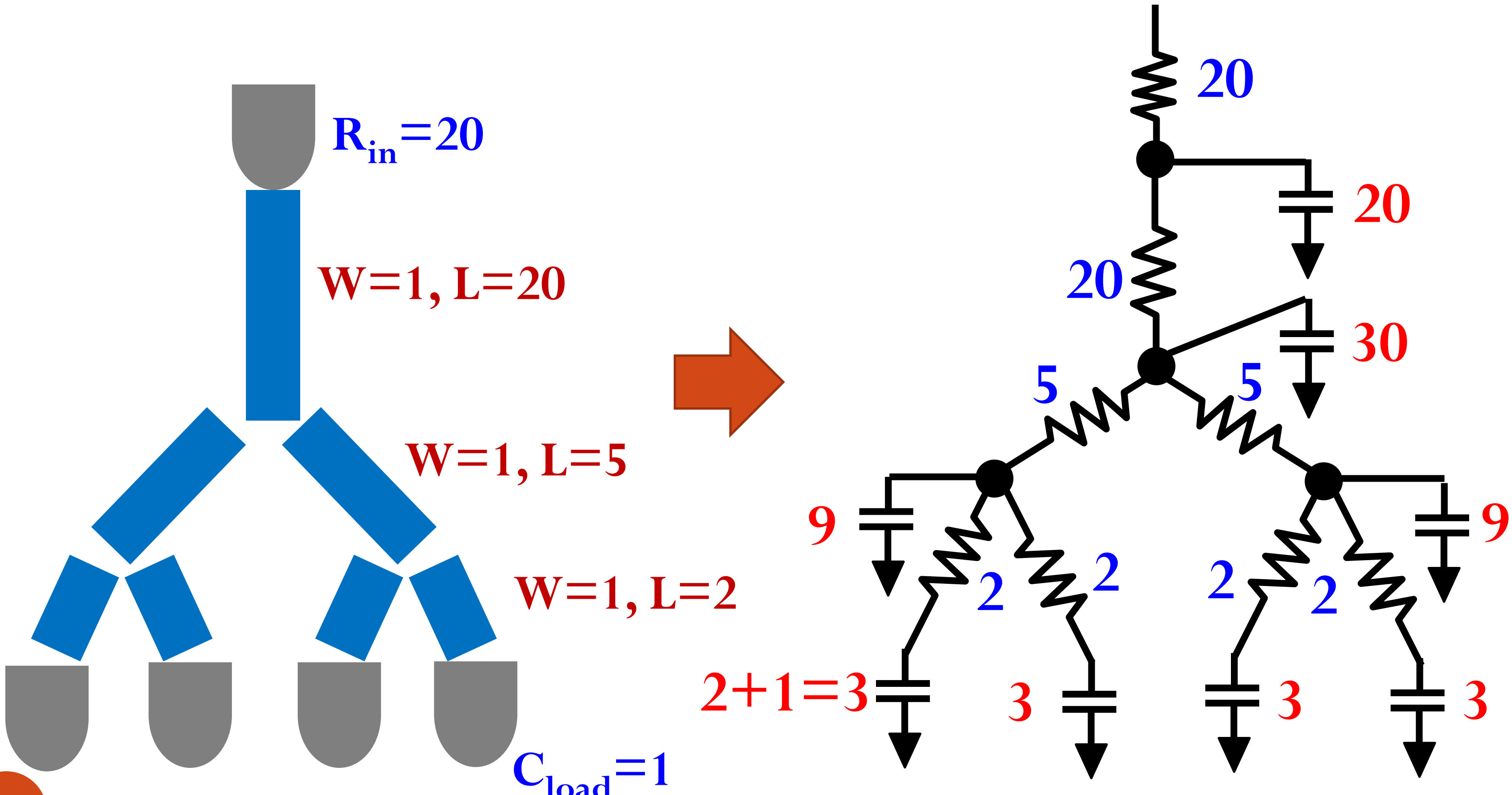
Elmore Delay Example

- RC Tree for the interconnect alone
  - Remember to add up C's hanging off each internal node of tree



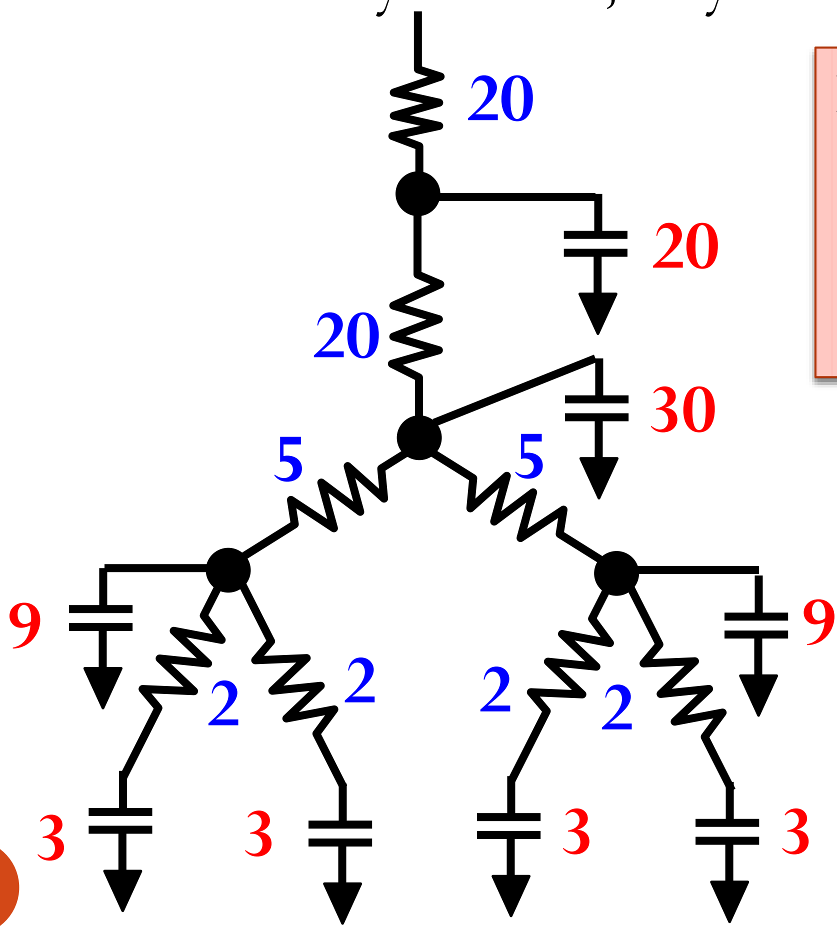
Elmore Delay Example

- Add driver and driven gates



Elmore Delay Example

- Now we can compute delay to each leaf.
  - Since symmetric, only need to compute 1 path.

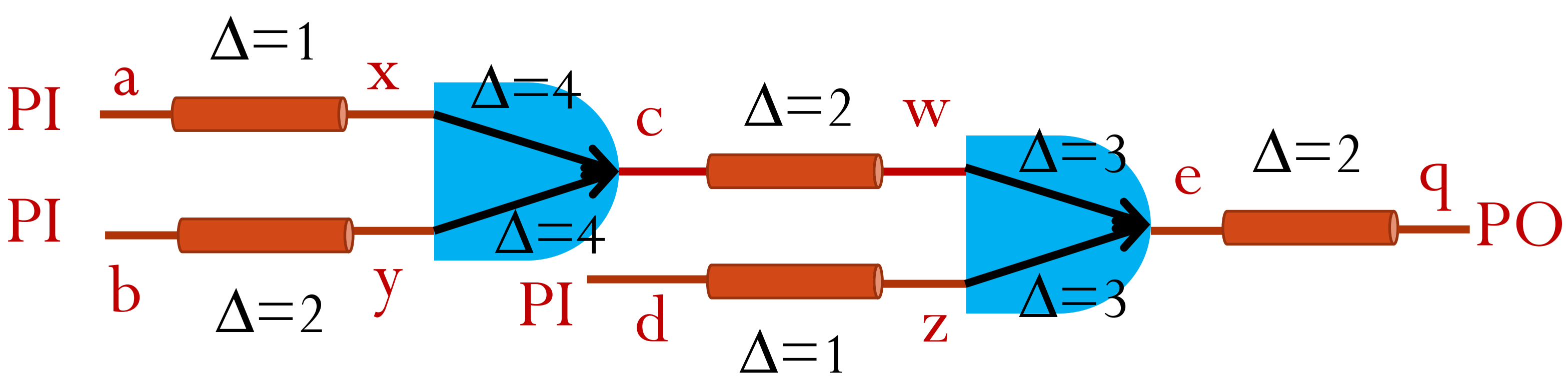


Remember the recipe:
1. Set $\tau = 0$, walk from root to leaf
2. At each R,
   τ += R • Σ (all C's downstream)

$$\tau = 0 \\ +20\times(20+30+2\times9+4\times3) \\ +20\times(30+2\times9+4\times3) \\ +5\times(9+2\times3) \\ +2\times3 \\ = 2881$$

Elmore Delay Applications

- Timing verification
  - Can use this to give realistic post-layout **wire delays** for final STA.



Elmore Delay Applications

- During placement
  - Estimate **wire shape** (e.g., a simple Steiner tree), then you can get **very quick** delay estimate.
  - Analytical placers use to adjust **weights** on wires, force critical wires to be **short**.

Summary

- Interconnect has a **huge** impact on chip speed.
  - Cannot ignore delays caused by the electrical properties of real wires
- Individual wires are today modeled as complex circuits.
  - RC tree is the most useful model; Elmore delay is easiest to compute
  - Can use for both verification, and for layout optimizations
  - Accurate enough that they beat simple length-based schemes.
  - Unfortunately, not so accurate that you can avoid later verification with "**higher order**" models that incorporate **more than one** time constant.
  - There are sophisticated estimators beyond Elmore, but they takes more time.