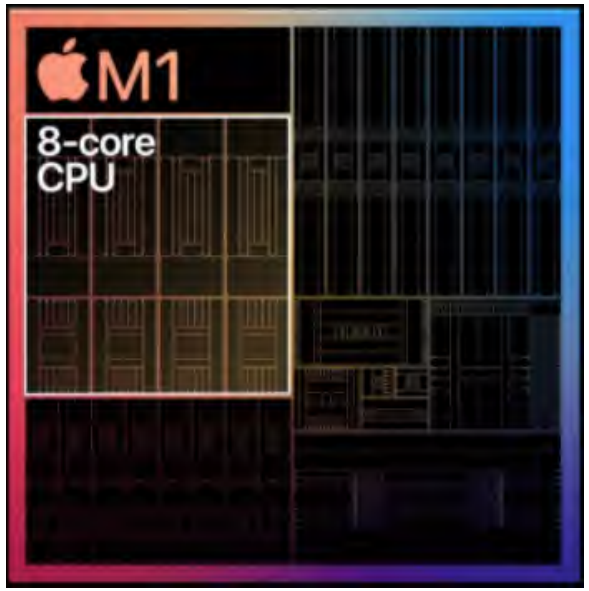
《芯片设计自动化与智能优化》
Static Timing Analysis

The slides are partly based on the lecture notes of Prof. David Z. Pan at UT Austin and Prof. Tsung-Wei Huang at UW Madison.

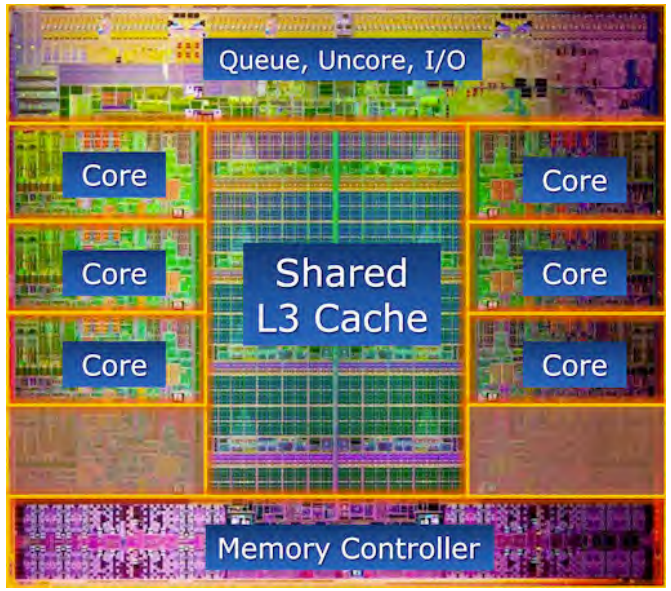
Yibo Lin

Peking University

How Fast A Chip Can Run?



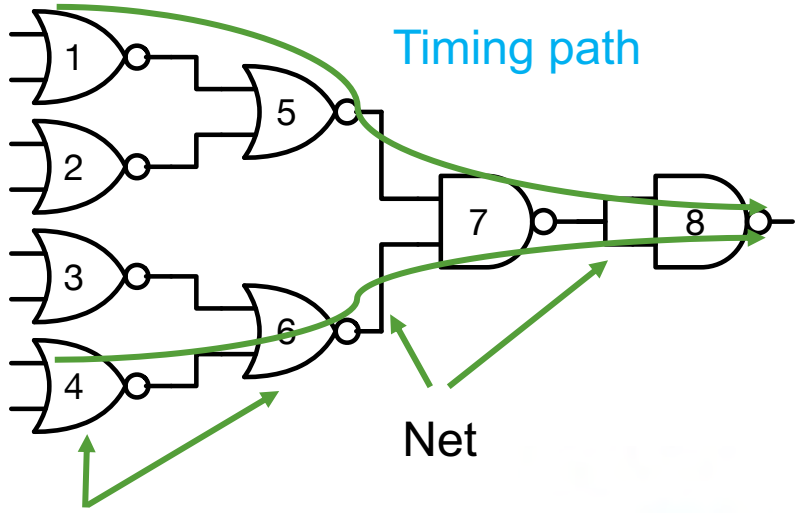
Technology: 5nm
Max. CPU clock: 3.2GHz
L2 cache: 12MB
Transistors: 16 billion



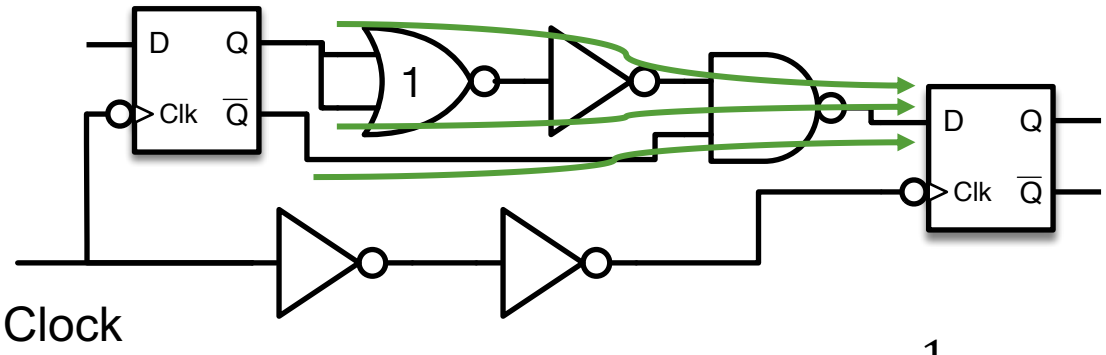
Technology: 32nm
Max. CPU clock: 3.3GHz (3.9GHz w. Turbo)
L2 cache: 15MB

Maximum Clock Frequency

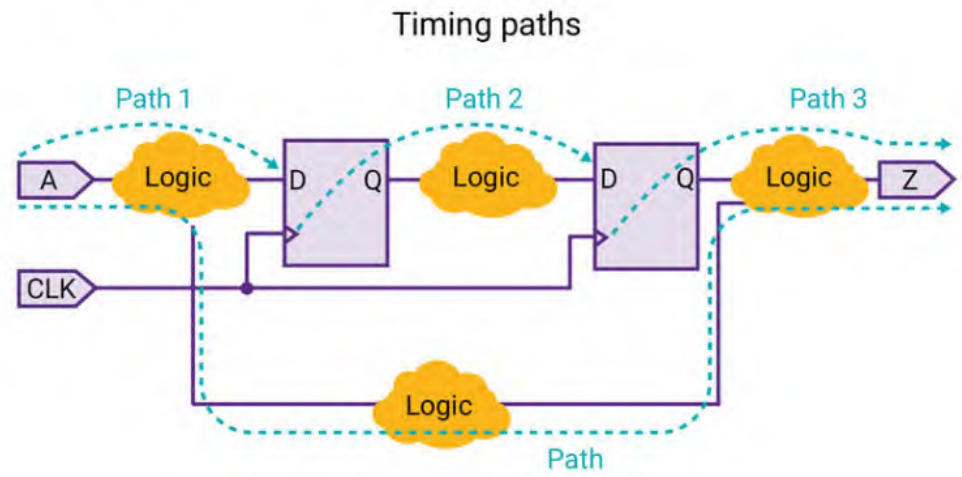
Combinational logic



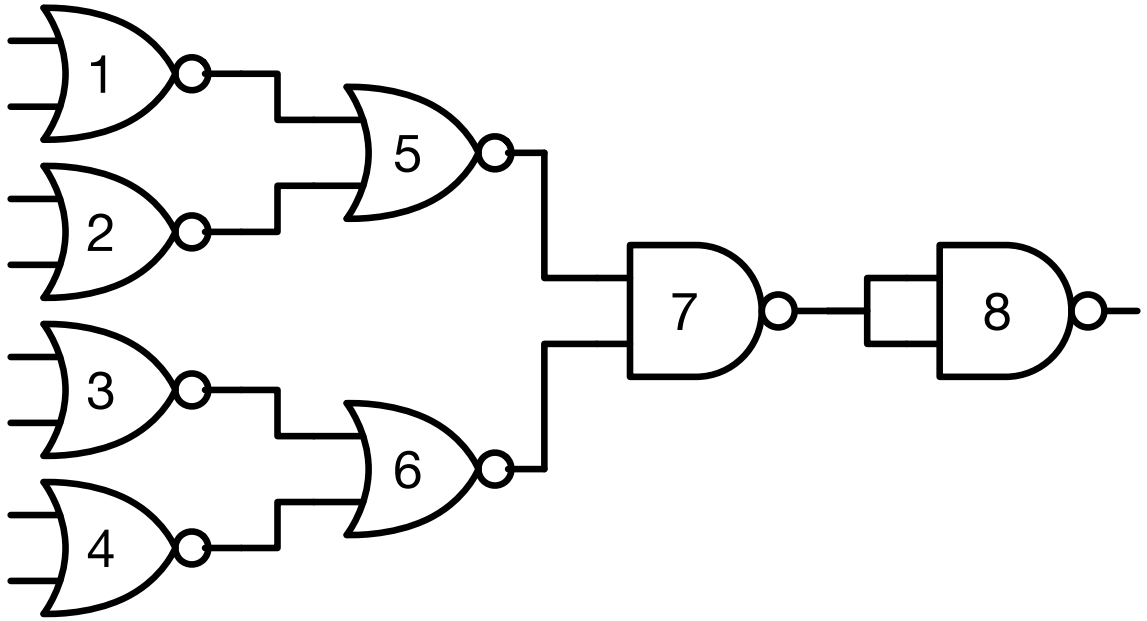
Sequential logic



$$f = \frac{1}{d}$$

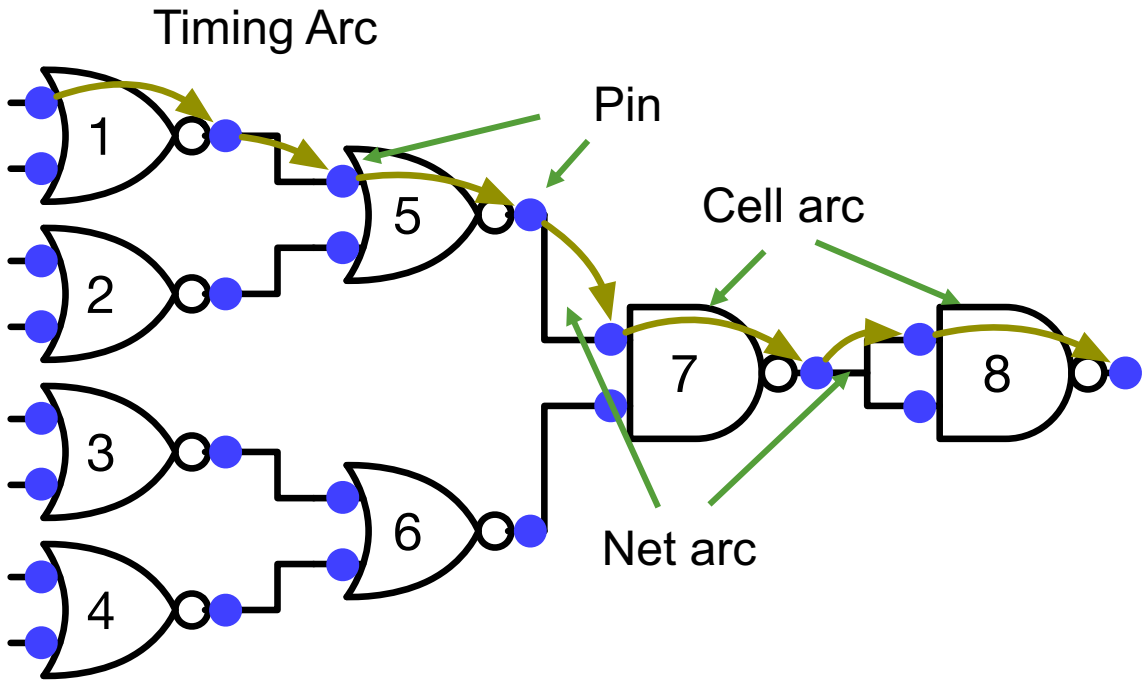


Timing Arc

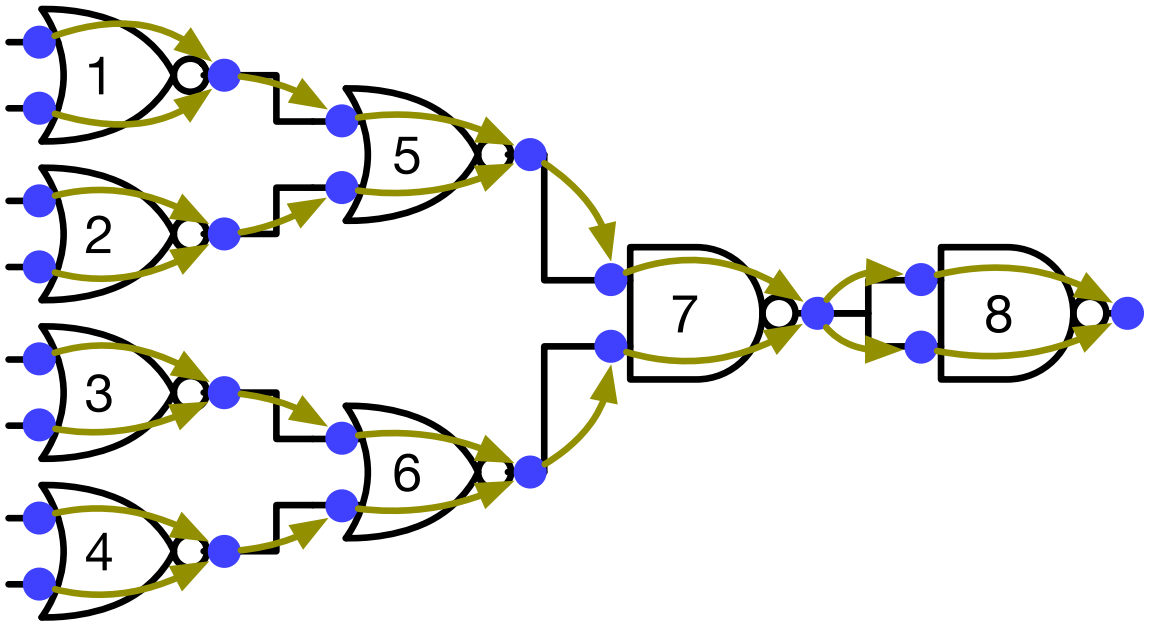


Timing Arc and Timing Graph

- Timing arc can be categorized to cell arc or net arc

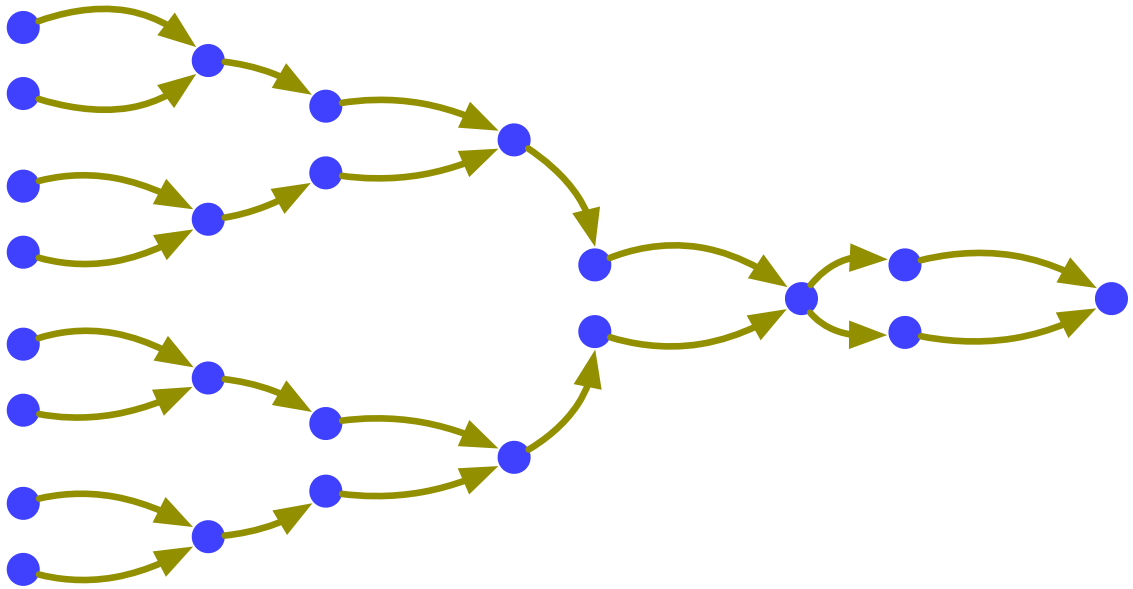


Timing Arc and Timing Graph



Timing Graph

- Timing graph consists of pins and timing arcs
- Given the delay of each timing arc, we can derive the path delays

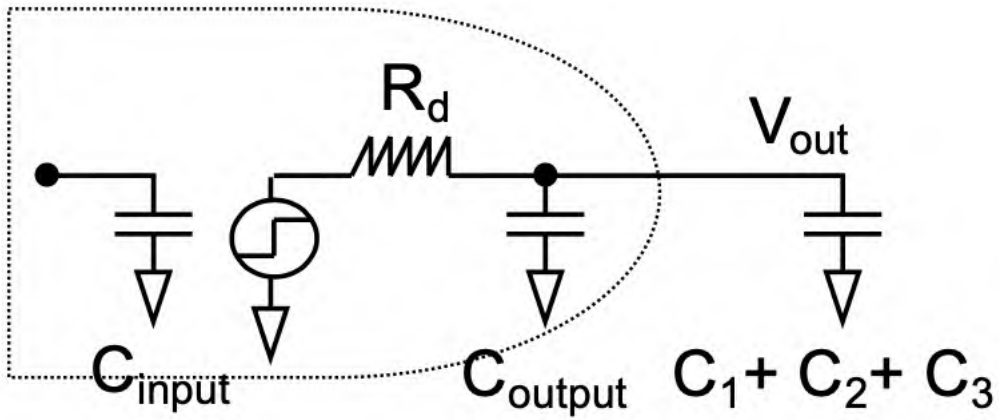


Net Delay

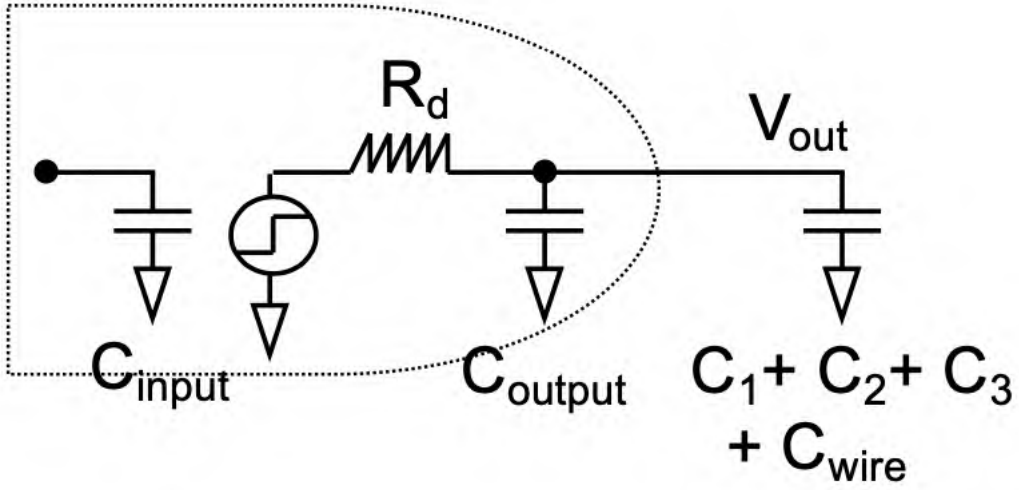
- Ignoring interconnects
- Lumped capacitance model
- RC tree model
- RLC tree model
- Transmission line models (RC, LC, RLC)
- RC/RLC network

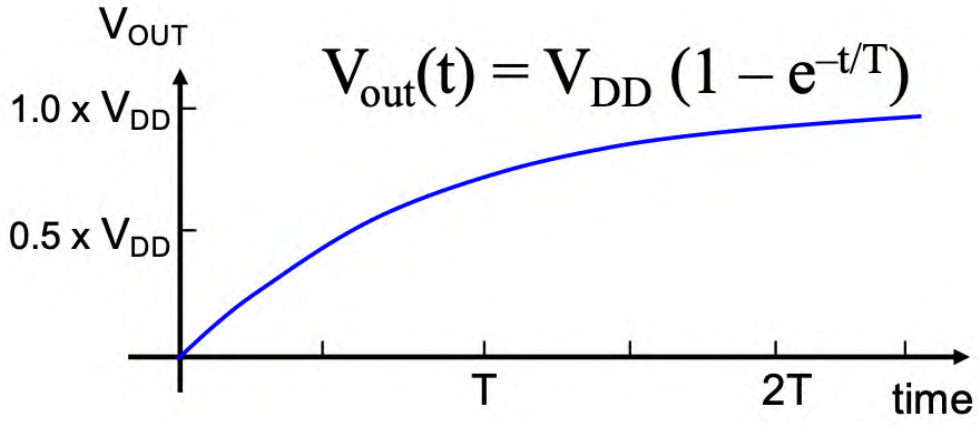
Interconnect Models as a Capacitor

- Ignoring interconnects



- Lumped capacitance model





$$V_{out}(t) = V_{DD}\,(1 - e^{-t/T})$$

$$T = Rd\,(C_{output} + C_1 + C_2 + C_3)$$

or

$$T = Rd\,(C_{output} + C_1 + C_2 + C_3 + C_{wire})$$

$$V_{out}^{-1}(0.5V_{DD}) = (\ln 2)\,T = 0.693\,T$$

$$V_{out}(T) = 0.632\,V_{DD}$$

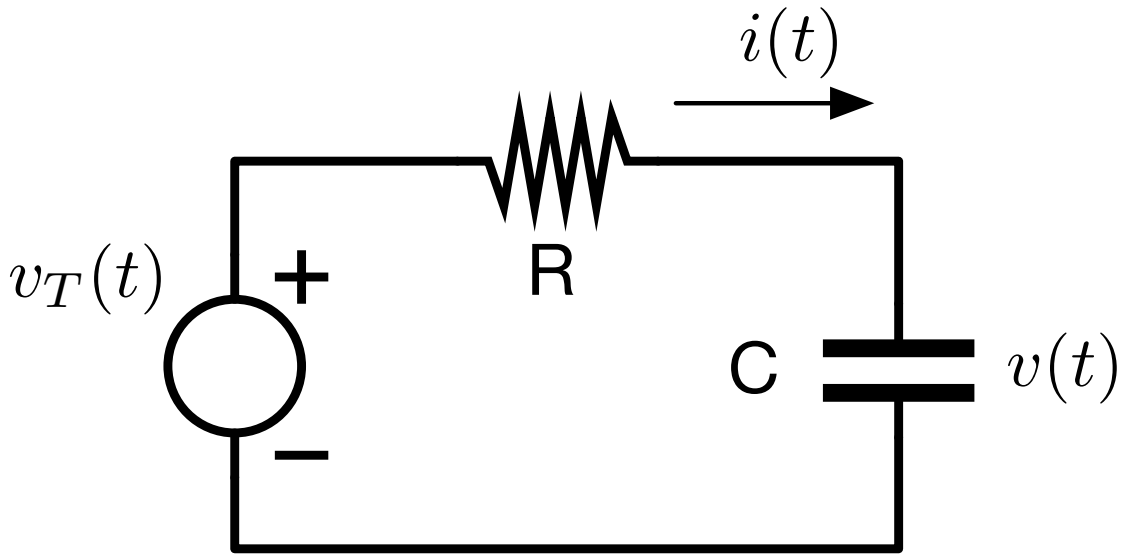
Analysis of Simple RC Circuit

- $R \cdot i(t) + v(t) = v_T(t)$
- $i(t) = \frac{d(Cv(t))}{dt} = C\frac{dv(t)}{dt}$
- $RC\frac{dv(t)}{dt} + v(t) = v_T(t)$

State variable

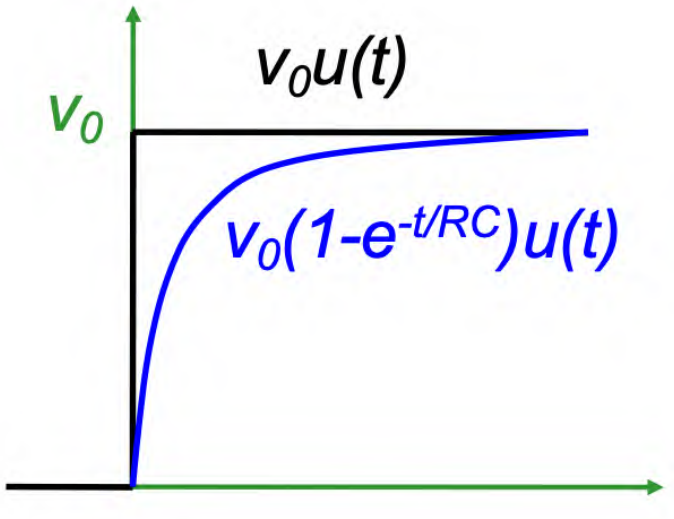
Input waveform

First-order linear differential equation
with constant coefficients



Analysis of Simple RC Circuit

- Zero-input response (natural response)
  - $RC\frac{dv(t)}{dt} + v(t) = 0 \rightarrow v(t) = Ke^{-\frac{t}{RC}}$
- Step-input response
  - $RC\frac{dv(t)}{dt} + v(t) = v_0u(t) \rightarrow v(t) = Ke^{-\frac{t}{RC}} + v_0u(t)$
  - Match initial state
  - $v(0) = 0 \rightarrow K + v_0u(t) = 0$
  - Output response for step-input
  - $v(t) = v_0\left(1 - e^{-\frac{t}{RC}}\right)u(t)$
- You can get same result by Laplace Transform
  - Try google it (“RC circuit Laplace Transform”)

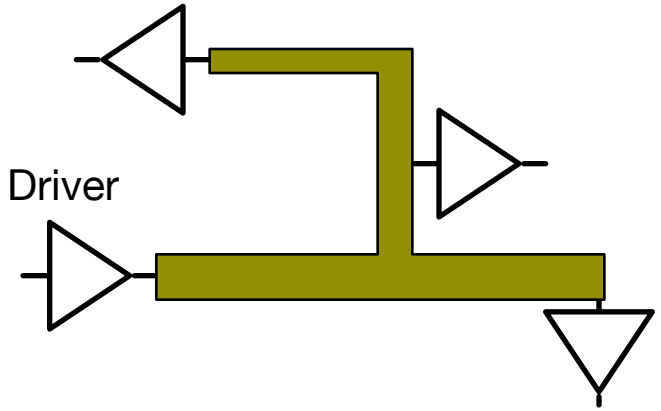


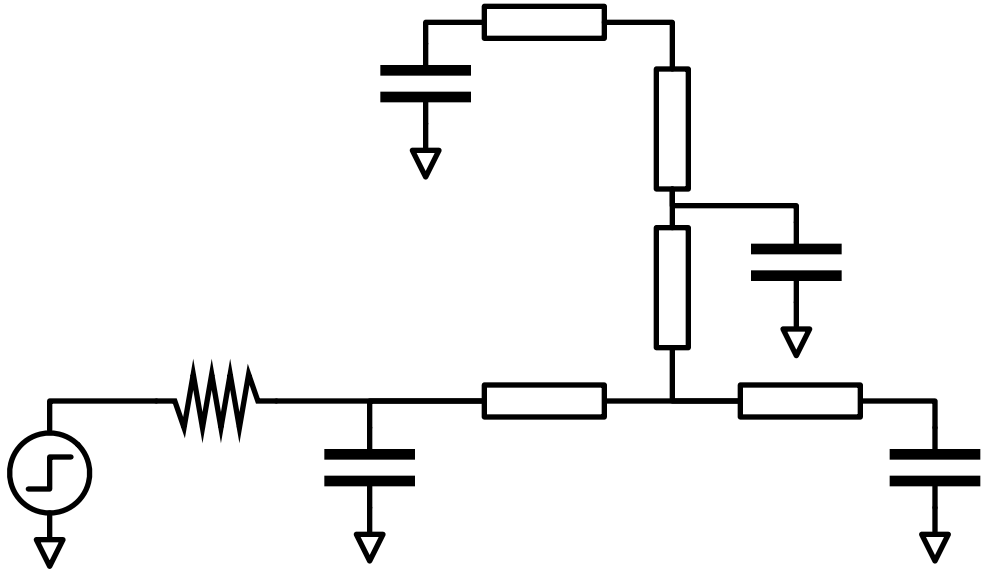
Delays of Simple RC Circuit

- $v(t) = v_0(1 - e^{-t/RC})$
  - Waveform under step input $v_0 u(t)$
- $v(t) = 0.5v_0$ ➔ $t = 0.7RC$
  - i.e., $delay = 0.7RC$ (50% delay)
- $v(t) = 0.1v_0$ ➔ $t = 0.1RC$
- $v(t) = 0.9v_0$ ➔ $t = 2.3RC$
  - i.e., $rise\ time = 2.2RC$ (if defined as time from 10% to 90% of Vdd)
- Commonly used metric
  - $T_D = RC$ (Elmore delay)

Interconnect Models as a Tree

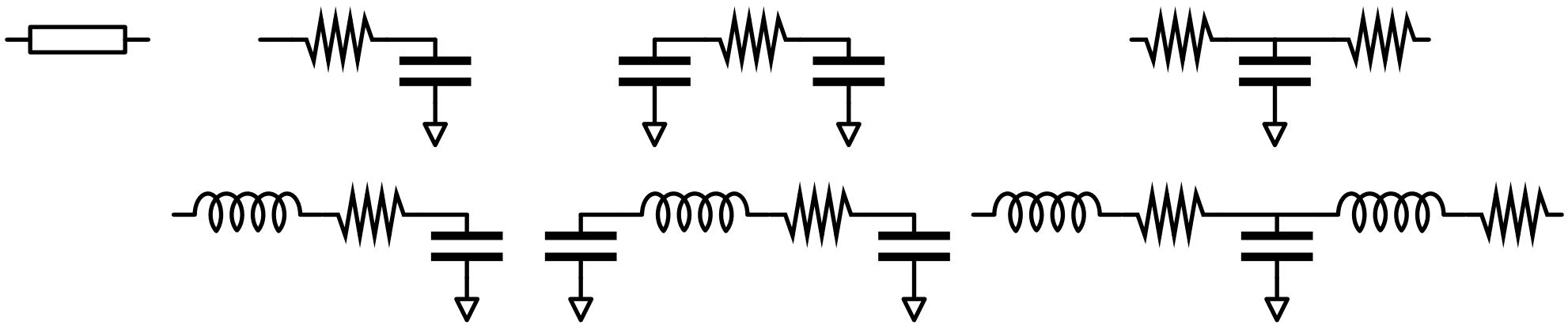
- RC tree
- RLC tree
- Transmission line models (RC, LC, RLC)





Interconnect Models as a Tree

- RC tree
- RLC tree
- Transmission line models (RC, LC, RLC)



Determining Which Model to Use

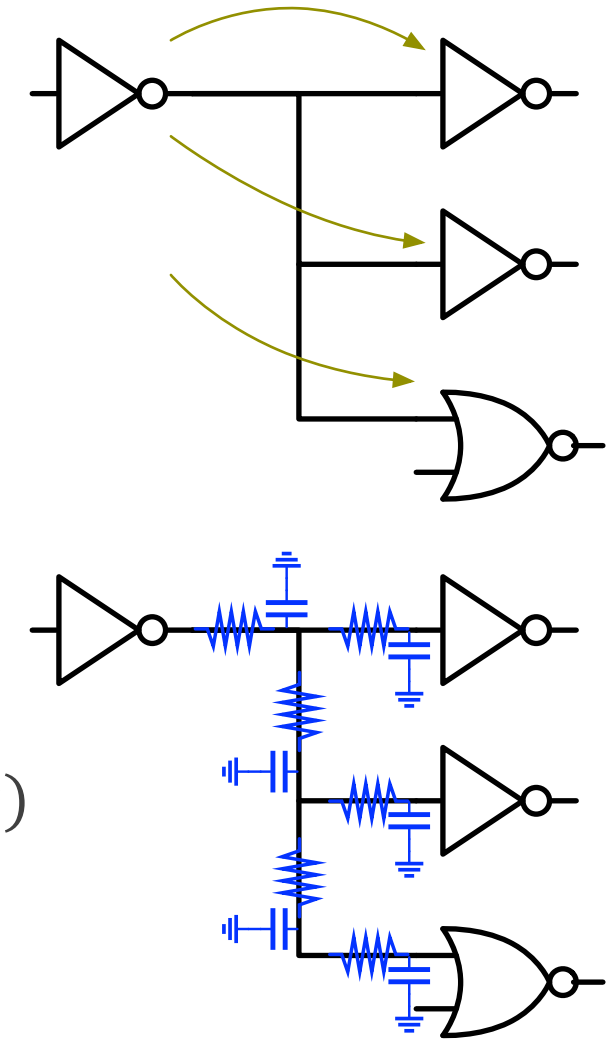
- Some rule-of-thumbs
- Need to consider C
  - if interconnect C is comparable to C of gates driven
- Need to consider R
  - if interconnect R is comparable to R of driver
- Need to consider L
  - if ωL is comparable to R of interconnect

More Interconnects as RC Trees

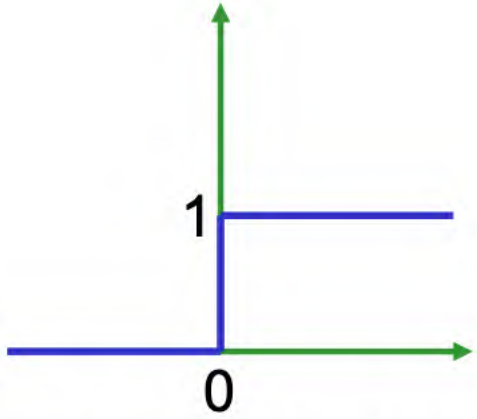
- Each wire maybe segmented into several edges
- Each edge E modeled as a π-type or L-type circuit
- $r_E = unit\ res. \times length(E)$
- $c_E = unit\ cap. \times length(E)$

Elmore Delay for RC Tree

- Definition
  - $H(t) =$ step response
  - $h(t) =$ impulse response = rate of change of H(t)
  - $T_{50\%} =$ median of $h(t)$
- Elmore delay [Elmore, 1948]
  - $T_D = mean\ of\ h(t) = \int_0^\infty h(t) \cdot t \cdot dt$
  - Approximates the median of $h(t)$ by mean
- Computation: $T_D(s_o, s_i) = \sum_{\substack{node\ k\ in \\ src-to-sink\ path}} R_k \cdot Cap(k)$
  - [Rubinstein et al, TCAD 1983]

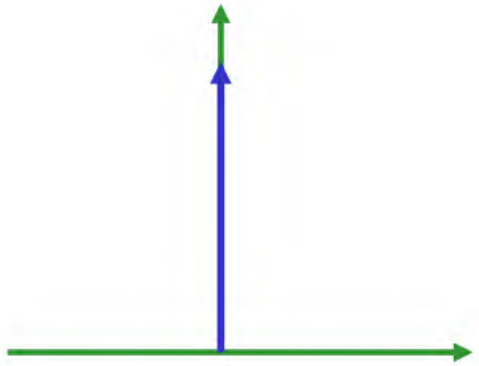


Basic Input Waveform



unit step function

$$u(t)=\begin{cases}0 & t<0\\ 1 & t\geq 0\end{cases}$$



unit impulse function

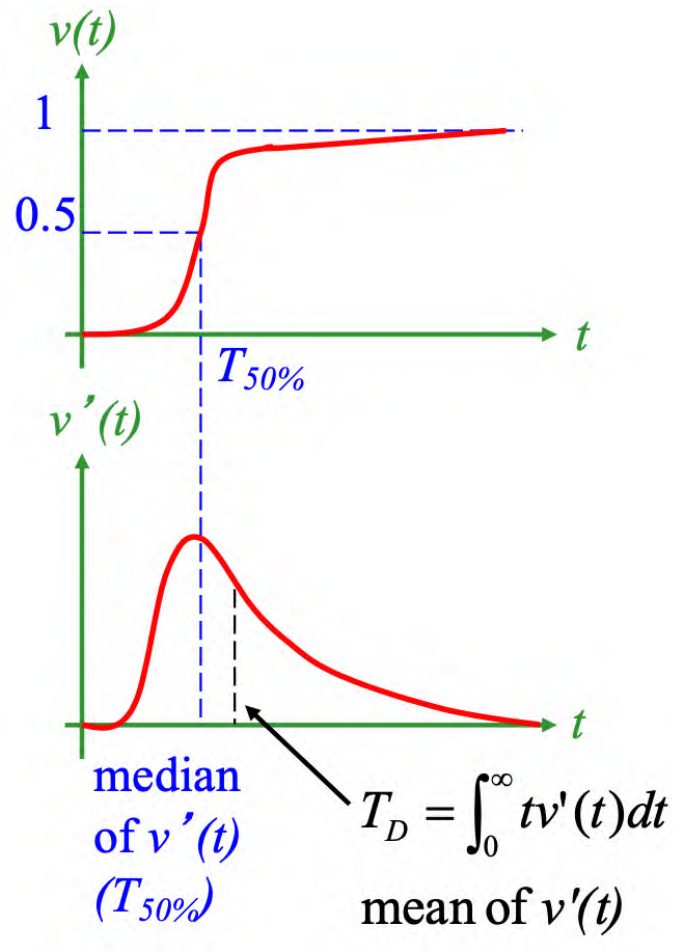
$$\delta(t): P_T(t) \quad \text{when } T\to 0$$

$$\delta(t)=\begin{cases}0 & \text{for } t\neq 0\\ \text{singular} & \text{for } t=0\end{cases}$$

$$\text{s.t. } \forall \zeta>0,\ \int_{-\zeta}^{\zeta}\delta(t)dt=1$$

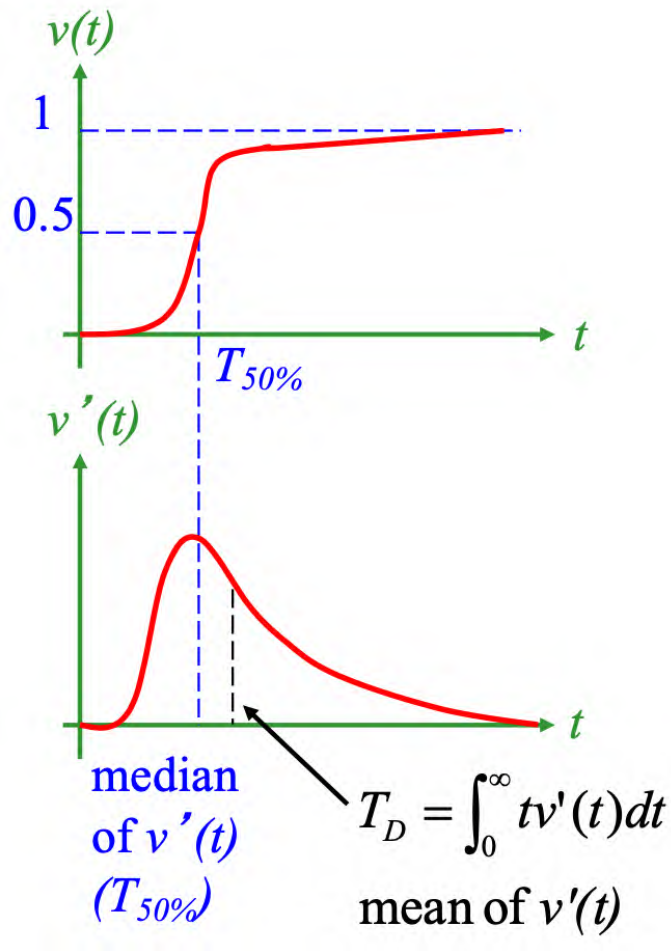
Elmore Delay for Monotonic Responses

- [Elmore, J. App. Phy. 1948]
- Assumptions
  - Unit step input
  - Monotone output response
  - $v(t)$: output response (monotone)
  - $v'(t)$: rate of change of $v(t)$
- Basic idea
  - Use mean of $v'(t)$ to approximate median of $v'(t)$



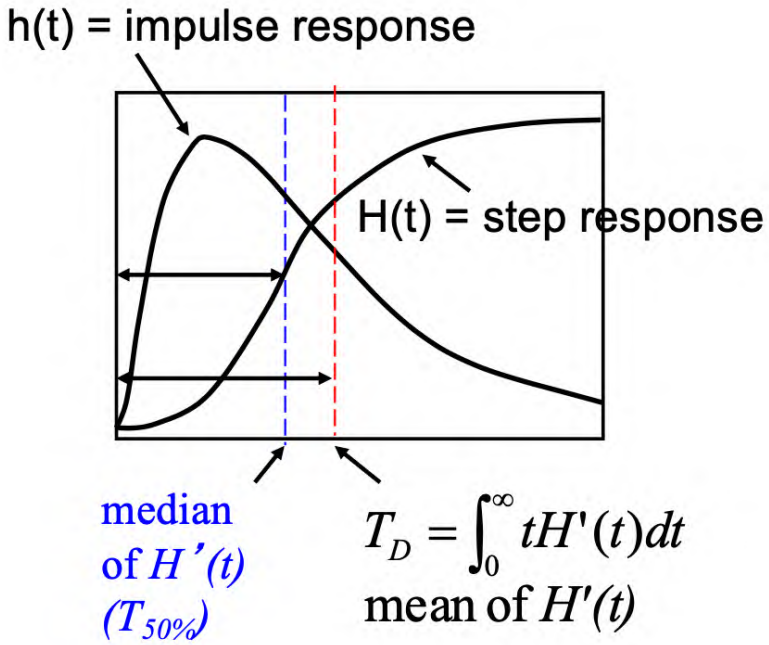
Elmore Delay for Monotonic Responses

- $T_{50\%}$: median of $v'(t)$, since
  - $\int_0^{T_{50\%}} v'(t)dt = \int_{T_{50\%}}^{+\infty} v'(t)dt$
  - Half of final value of $v(t)$
- Elmore delay T_D = mean of $v'(t)$
  - $T_D = \int_0^{\infty} tv'(t)dt$



Elmore Delay for Monotonic Responses

- Definition
  - $h(t) =$ impulse response
  - $T_D =$ mean of $h(t) = \int_0^\infty h(t) \cdot t \cdot dt$
- Interpretation
  - $H(t) =$ output of response (step response)
  - $h(t) =$ rate of change of $H(t)$
  - $T_{50\%} =$ median of $h(t)$
  - Elmore delay approximates the median of $h(t)$ by the mean of $h(t)$



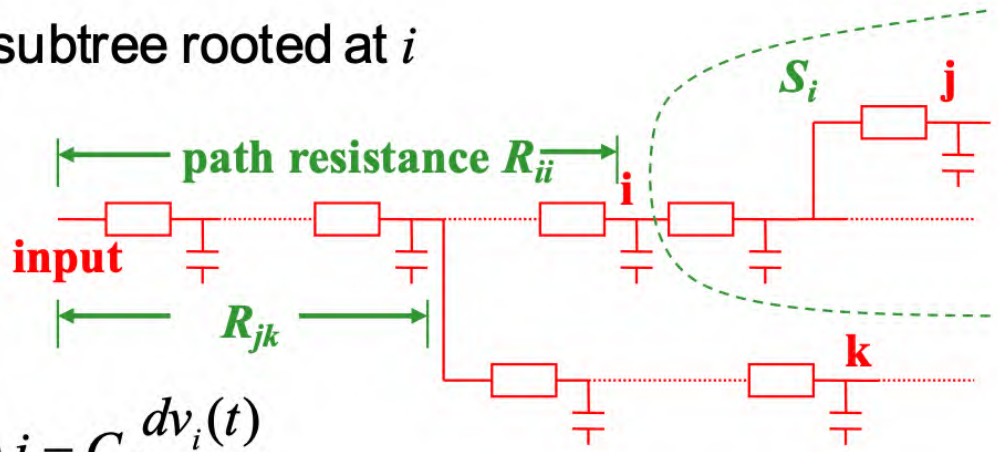
Elmore Delay of a RC Tree [Rubinstein et al, TCAD'83]

P_i : path from input to node i ; S_i : subtree rooted at i

R_{jk} : resistance of common path $P_j \cap P_k$ from input to j & k

Theorem : Elmore delay to node i

$$T_{D_i} = \sum_k R_{ki} C_k$$



Proof : The current to cap. of node $i = C_i \dfrac{dv_i(t)}{dt}$

$1 - v_i(t)$ = The voltage drop on $P_i = \sum_{k \in P_i} R_k \cdot$ (current to all cap's in S_k)

$$= \sum_k (\text{current to cap } k) \cdot (\text{common path res. between } P_i \text{ and } P_k)$$

$$= \sum_k C_k \frac{dv_k(t)}{dt} \cdot R_{ki} = \sum_k R_{ki} C_k \frac{dv_k(t)}{dt}$$

$$T_{D_i} = \int_0^\infty v'_i(t) t \cdot dt = v_i(t) \cdot t \,|_0^\infty - \int_0^\infty v_i(t) dt$$

$$= \lim_{T \to \infty} [v_i(T) \cdot T - \int_0^T v_i(t) dt] = \lim_{T \to \infty} (v_i(T) - 1) \cdot T + \int_0^\infty (1 - v_i(t)) dt$$

Elmore Delay of a RC Tree [Rubinstein et al, TCAD'83]

We can show that $\lim_{T\to\infty}(1-v_i(T))\cdot T = 0$

i.e. 1-v_i(T) goes to 0 at a much faster rate than 1/T when T$\to\infty$

Let $f_i(t) = \int_0^t [1-v_i(x)]dx$

$$f_i(t) = \int_0^t \sum_k R_{ki}C_k \frac{dv_k(x)}{dx}dx$$

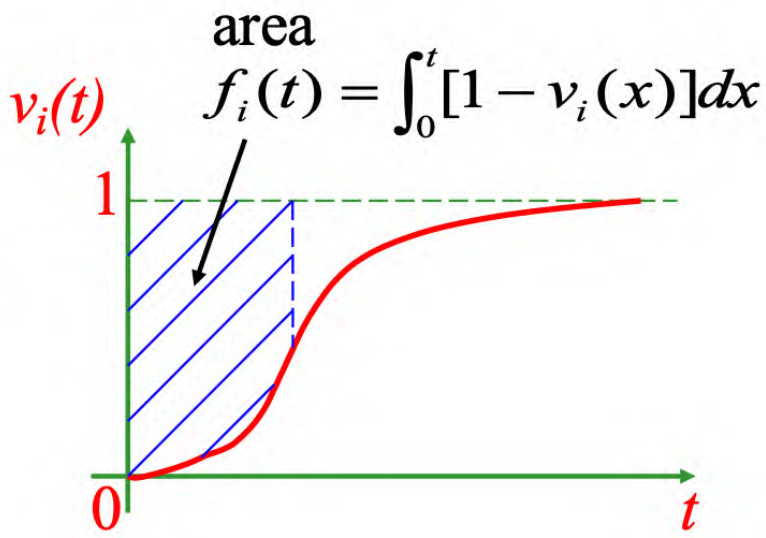
$$= \sum_k R_{ki}C_k v_k(t)$$

$$= \sum_k R_{ki}C_k - \sum_k R_{ki}C_k[1-v_k(t)] \quad (1)$$

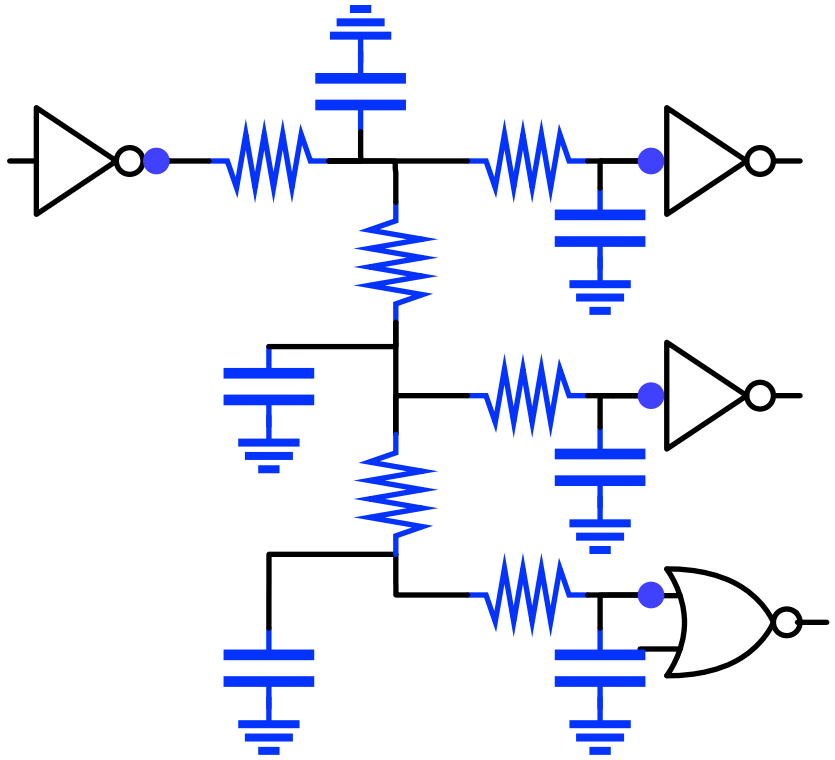
$$f_i(\infty) = \sum_k R_{ki}C_k$$

$$\therefore T_{D_i} = \lim_{T\to\infty}(1-v_i(T))T + \int_0^\infty [1-v_i(t)]dt$$

$$= f_i(\infty) = \sum_k R_{ki}C_k$$

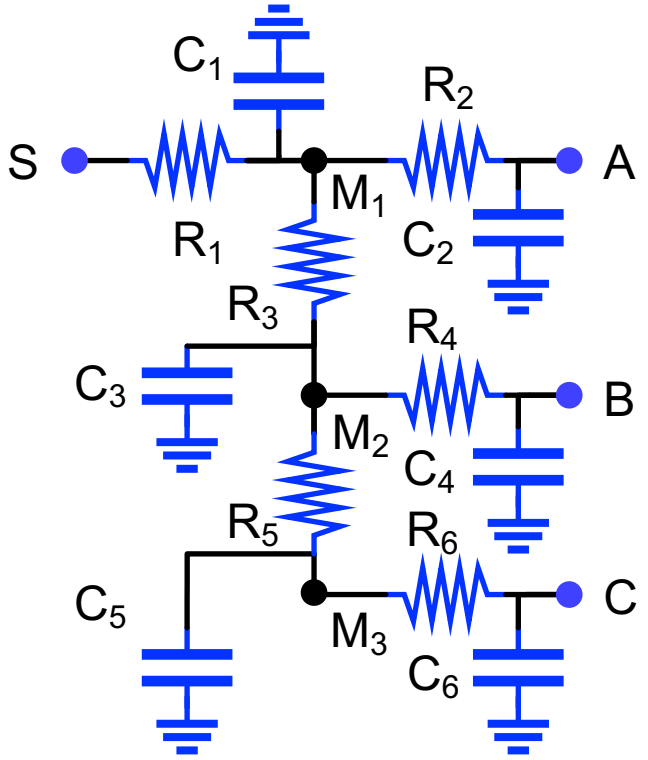


Elmore Delay: Example



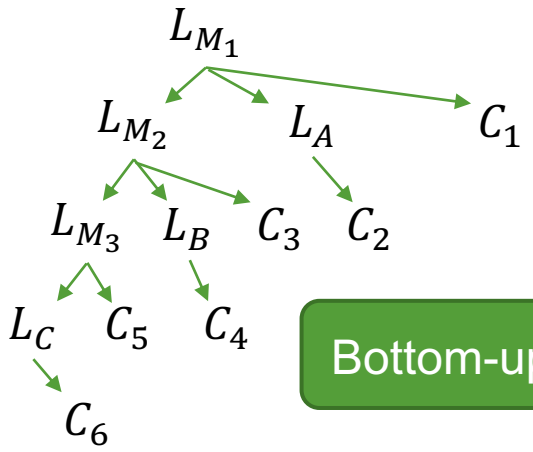
Elmore Delay: Example

- Elmore delay model
  - $R_{S\to A} = R_1(C_1 + C_2 + \cdots + C_6) + R_2C_2$
  - $R_{S\to B} = R_1(C_1 + C_2 + \cdots + C_6) + R_3(C_3 + C_4 + C_5 + C_6) + R_4C_4$
  - $R_{S\to C} = R_1(C_1 + C_2 + \cdots + C_6) + R_3(C_3 + C_4 + C_5 + C_6) + R_5(C_5+C_6) + R_6C_6$
  - Essentially dynamic programming!



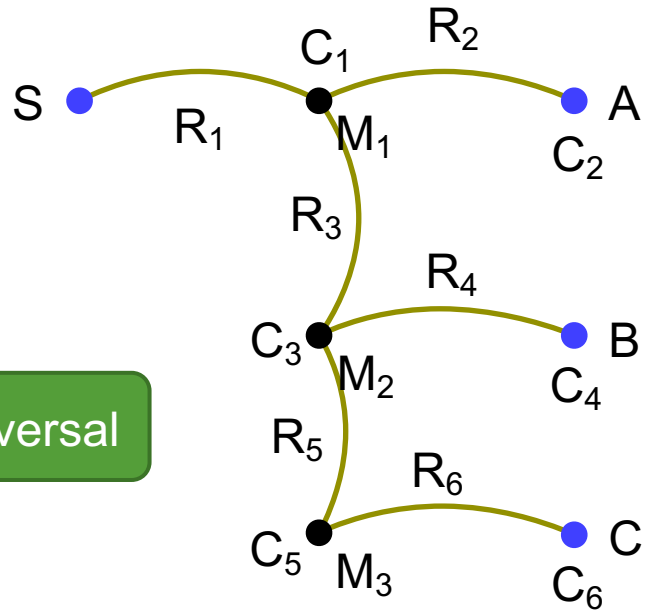
Elmore Delay: Example

- Elmore delay model
  - Compute load for each node
    - $L_A = C_2, L_B = C_4, L_C = C_6$
    - $L_{M_3} = C_5 + L_C$
    - $L_{M_2} = C_3 + L_{M_3} + L_B$
    - $L_{M_1} = C_1 + L_{M_2} + L_A$
  - Compute delay
    - $D_{S\to M_1} = R_1 L_{M_1}$
    - $D_{S\to A} = D_{S\to M_1} + D_{M_1\to A}$
    - $D_{S\to B} = D_{S\to M_2} + D_{M_2\to B} = D_{S\to M_1} + D_{M_1\to M_2} + D_{M_2\to B} = D_{S\to M_1} + R_3 L_{M_2} + D_{M_2\to B}$
    - $D_{S\to C} = D_{S\to M_3} + D_{M_3\to C} = D_{S\to M_2} + D_{M_2\to M_3} + D_{M_3\to C} = D_{S\to M_2} + R_5 L_{M_3} + D_{M_3\to C}$



Bottom-up traversal

Tree Representation



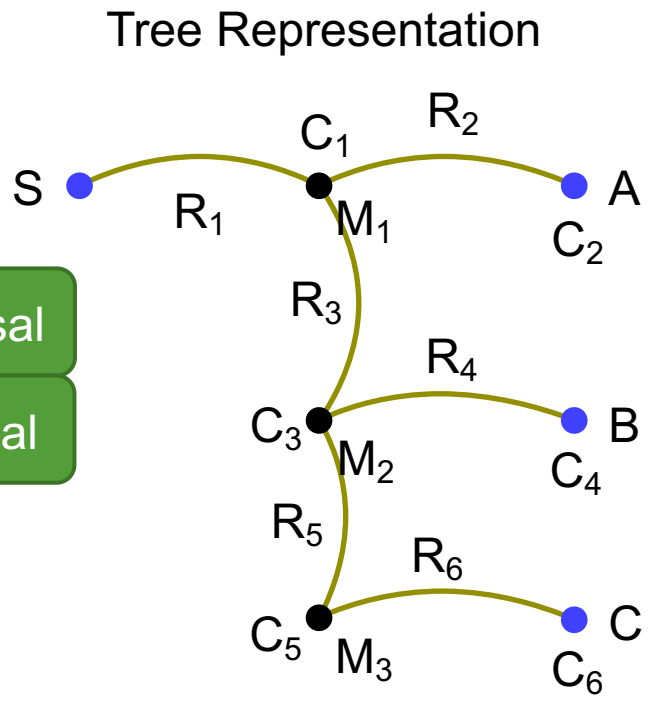
Top-down traversal

Elmore Delay: Example

- Elmore delay model
  - Compute β for each node
    - $\beta_{M_1} = R_1(C_1d_1 + C_2d_2 + C_3d_3 + C_4d_4 + C_5d_5 + C_6d_6)$
    - $\beta_A = \beta_{M_1} + R_2C_2d_2$
    - $\beta_{M_2} = \beta_{M_1} + R_3(C_3d_3 + C_4d_4 + C_5d_5 + C_6d_6)$
    - $\beta_B = \beta_{M_2} + R_4C_4d_4$
  - Compute output slew
    - $s_A \approx \sqrt{s_S^2 + \hat{s}_A^2} = \sqrt{s_S^2 + 2\beta_A^2 + d_A^2}$
    - $(\hat{s}_A \approx \sqrt{2\beta_A - d_A^2})$

Bottom-up traversal

Top-down traversal

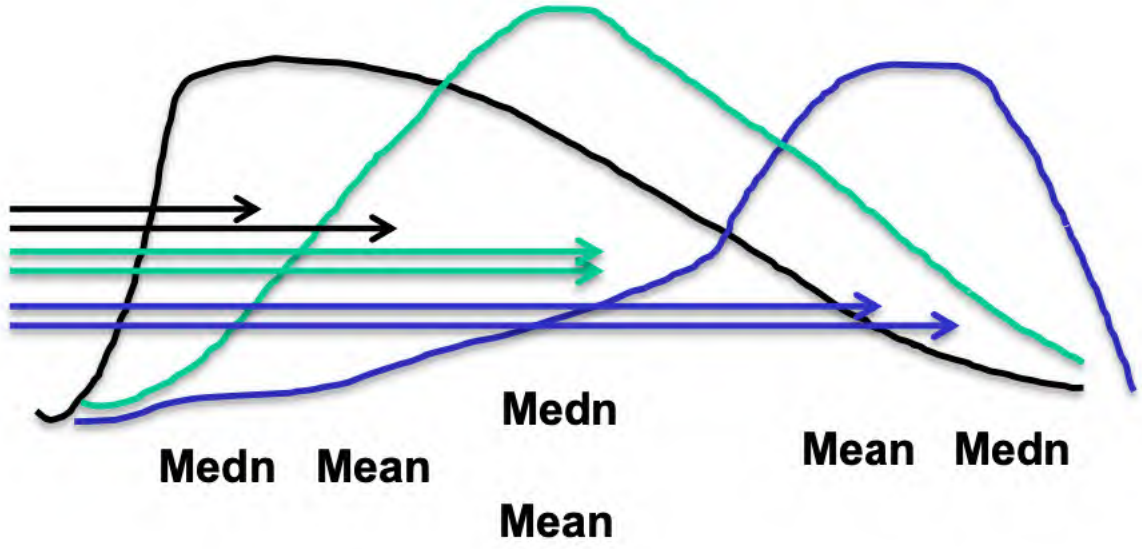


Properties of Elmore Delay

- Advantages
  - Simple closed-form expression
    - Useful for interconnect optimization
  - Upper bound of 50% delay [Gupta et al, 97]
    - Actual delay asymptotically approaches Elmore delay as input signal rise time increases
  - High fidelity [Boese et al., 93],[Cong-He,96]
    - Good solutions under Elmore delay are good solutions under actual (SPICE) delay
- Disadvantages
  - Low accuracy, no waveform information
  - Inherently cannot handle inductance effect

Insights on Elmore Delay

- Why Elmore delay is an upper bound of 50% signal delay for RC tree?
  - What is an upper bound
  - Analyze one R and C network
- When is the mean equal to the median



More Accurate Delay Models with Higher Order Moments

- Applicable to general RLC networks
- Used for waveform approximation (e.g. Asymptotic Waveform Evaluation (AWE))
- More accurate than Elmore delay
- Much more computational intensive than Elmore delay model too!

Transfer Function of Interconnect Network

- Each RC/RLC network transforms the input signal into an output signal
- This transformation is characterized by a “transfer function”
- Transfer function of RLC network can be written as a proper
- rational function (i.e., m<n) in frequency domain (recall Laplace?):

$$H(s) = \frac{B_m s^m + B_{m-1} s^{m-1} + \ldots + B_1 s + B_0}{A_n s^n + A_{n-1} s^{n-1} + \ldots + A_1 s + A_0}$$

- Benefit?
  - Once such H(S) is computed, getting output is very easy – find INP(s),
  - find H(s) * INP(s), take inverse Laplace transform = output waveform
- Thus, serves like “signature” of an interconnect network

Moments of Impulse Response $h(t)$

- Definition of moments

$$h(t) \xrightarrow{\text{L}} H(s)$$

$$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$$

$$H(s) = \int_0^\infty h(t)e^{-st}dt = \int_0^\infty h(t)\left(\sum_{i=0}^{\infty}\frac{1}{i!}(-st)^i\right)dt$$

$$= \sum_{i=0}^{\infty} s^i \left(\frac{(-1)^i}{i!}\int_0^\infty h(t)\cdot t^i dt\right) = \sum_{i=0}^{\infty} s^i m_i$$

i-th moment: $m_i = \frac{(-1)^i}{i!}\int_0^\infty h(t)\cdot t^i dt$

- Note that $-m_1$ gives Elmore delay when $h(t)$ is positive voltage response of impulse input

From previous slides:

Elmore Delay [Elmore, 1948]

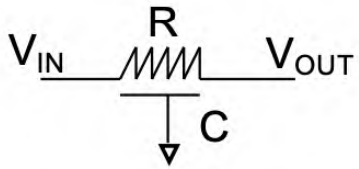
- T_D = mean of h(t) = $\int_0^\infty h(t)\cdot t\,dt$

Significance of Moments?

- Elmore delay is the first “moment” of transfer function.
- k-th moment is the expression which gets multiplied by s^k (where s is frequency, in Laplace).
- Each moment has information about transfer function.
- For current technology nodes (< 32nm) important to consider m1, m2, m3, and even more!
- Some microprocessor companies use as many as 20 moments.
  - Using infinite moments would give exact waveform.
  - But would spend too much CPU time

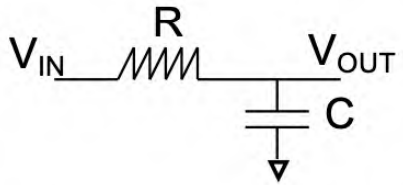
Distributed vs Lumped RC Lines: Analysis

Assume Vin is a rising step signal

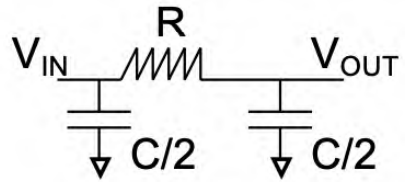


$V_{out}(t) \xrightarrow[\text{Transform}]{\text{Laplace}} V_{out}(s)$

$$V_{out}(s) = \frac{1}{s\cosh\sqrt{sRC}}$$



$$V_{out}(t) = 1 - e^{-t/RC}$$



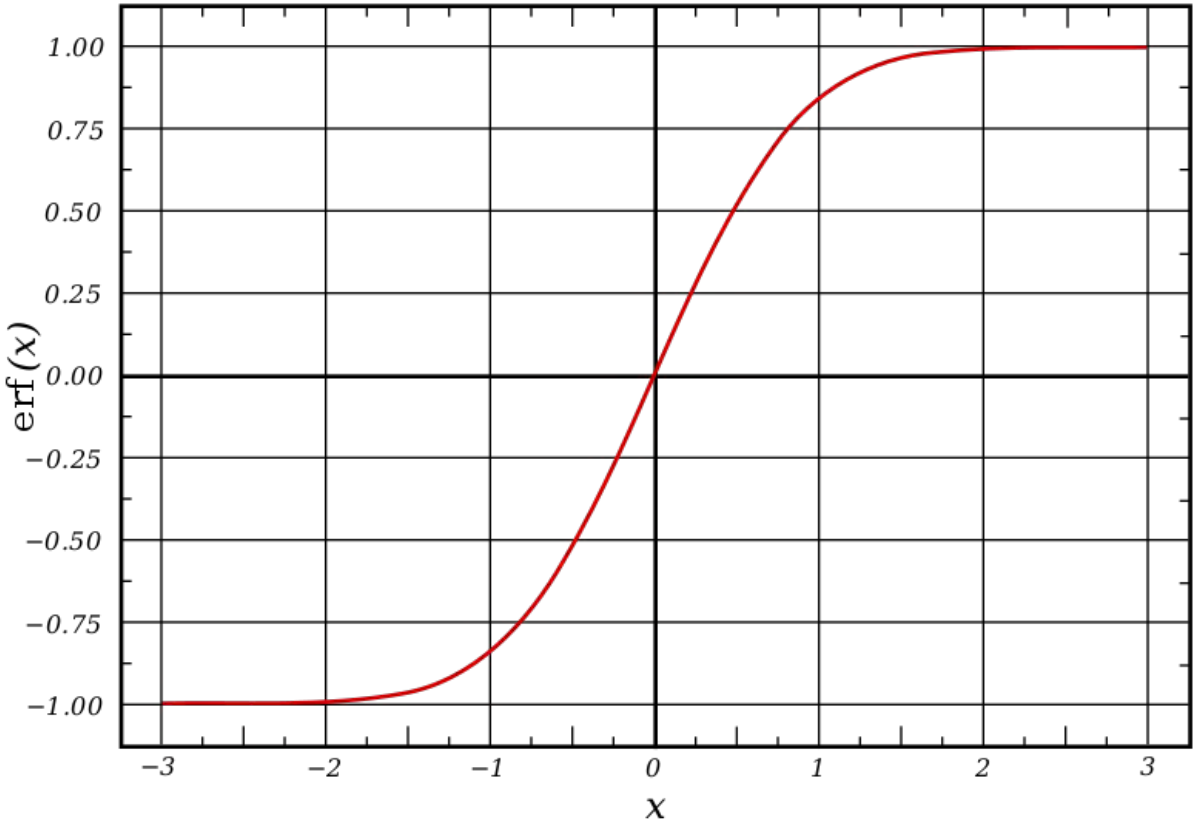
$$V_{out}(t) = 1 - e^{-t/0.5RC}$$

where $\cosh(x) = \frac{e^x + e^{-x}}{2} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots\ldots \approx \begin{cases} e^x/2 & x >> 1 \\ 1 + \frac{x^2}{2!} + \frac{x^4}{4!} & x << 1 \end{cases}$

$$V_{out}(t) = \begin{cases} erfc\sqrt{\frac{RC}{4t}} & t << RC \\ 1 - 1.366e^{-2.5359t/RC} + 0.366e^{-9.4641t/RC} & t >> RC \end{cases}$$

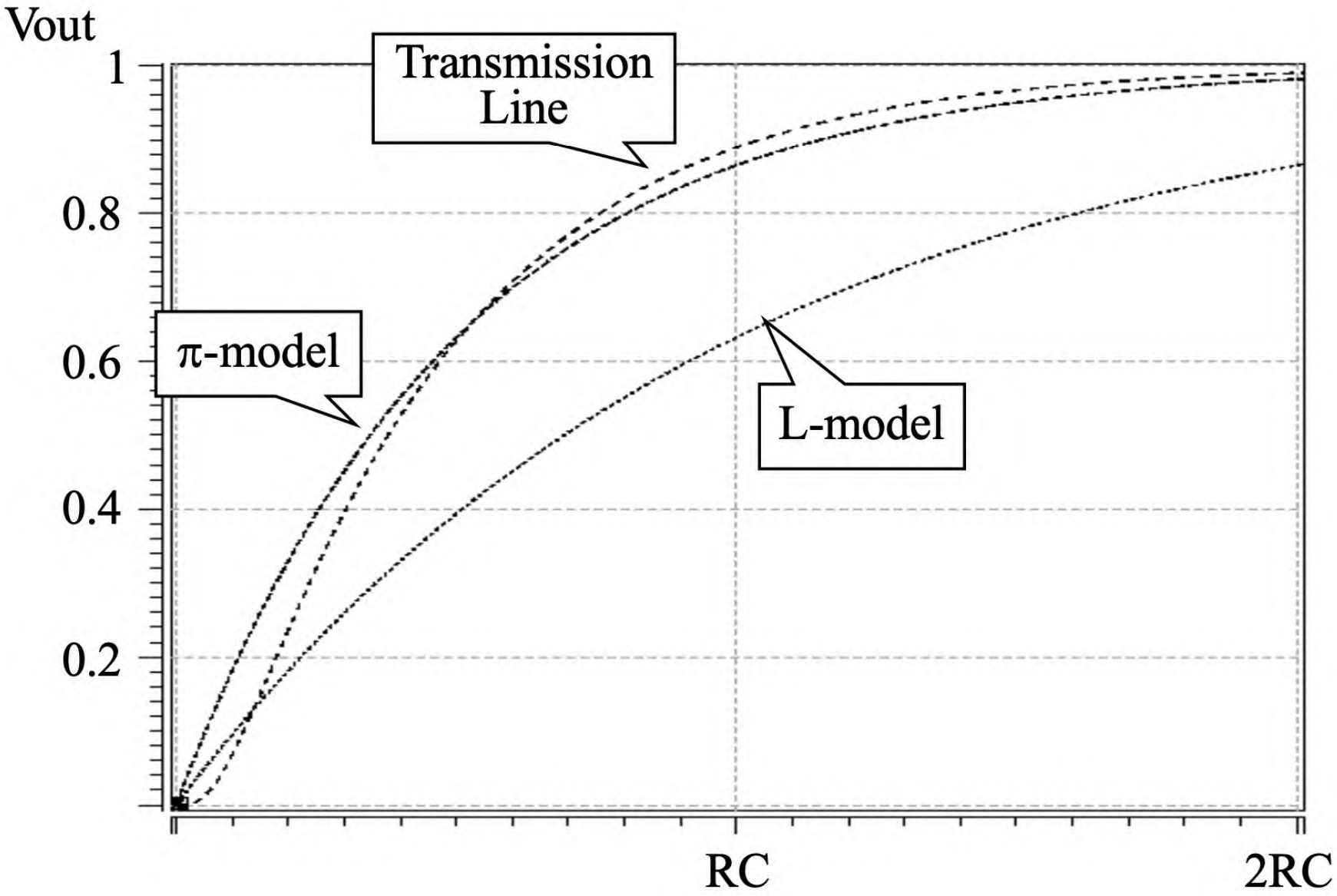
Gauss Error Function

$$\operatorname{erf} z = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\, dt.$$



https://en.wikipedia.org/wiki/Error_function

Distributed vs Lumped RC Lines: Waveform

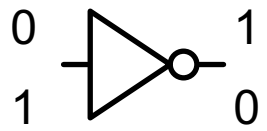


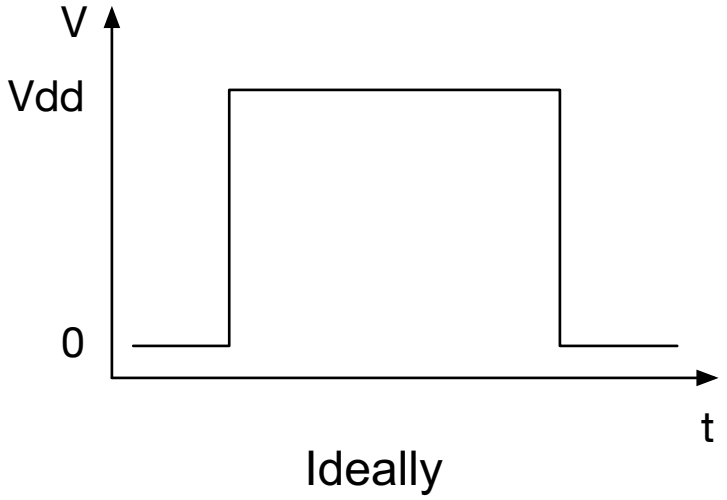
Distributed vs Lumped RC Lines: Delay

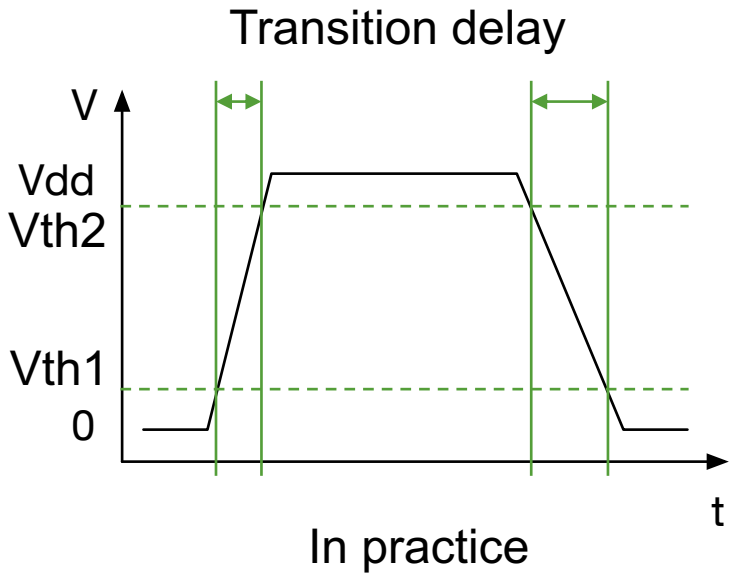
Output potential range	Time elapsed (RC tranmission Line Model)	Time elapsed (L-type RC Network)	Time elapsed (π-type RC Network)
0 to 90%	1.0 *RC*	2.3 *RC*	1.15 *RC*
10% to 90% (rise time)	0.9 *RC*	2.2 *RC*	1.1 *RC*
0 to 63%	0.5 *RC*	1.0 *RC*	0.5 *RC*
0 to 50% (delay)	0.4 *RC*	0.693 *RC*	0.347 *RC*
0 to 10%	0.1 *RC*	0.105 *RC*	0.105 *RC*

Cell Delay

- Transition delay
  - Slew fall/rise
  - Typical Vth1/Vth2: 10%/90%

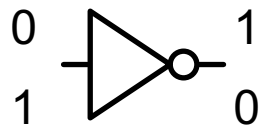


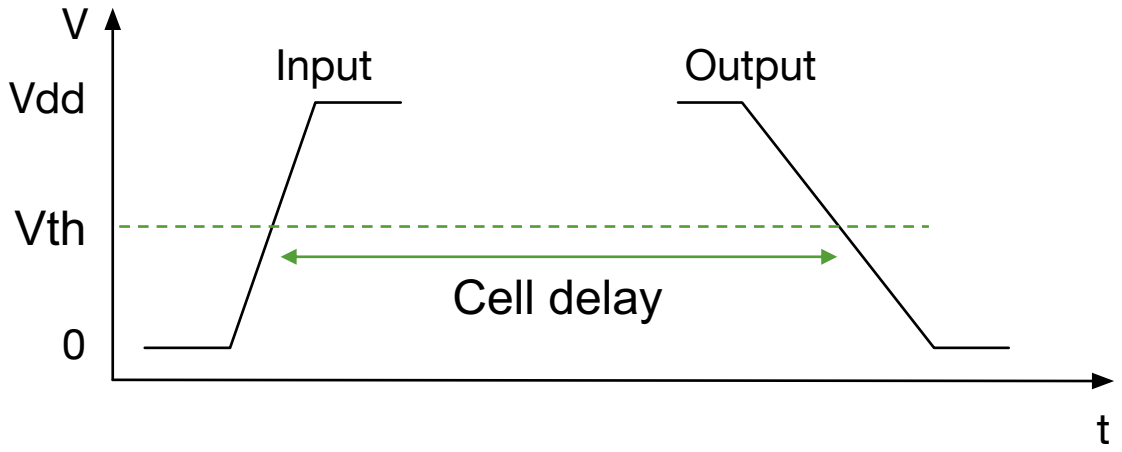




Cell Delay

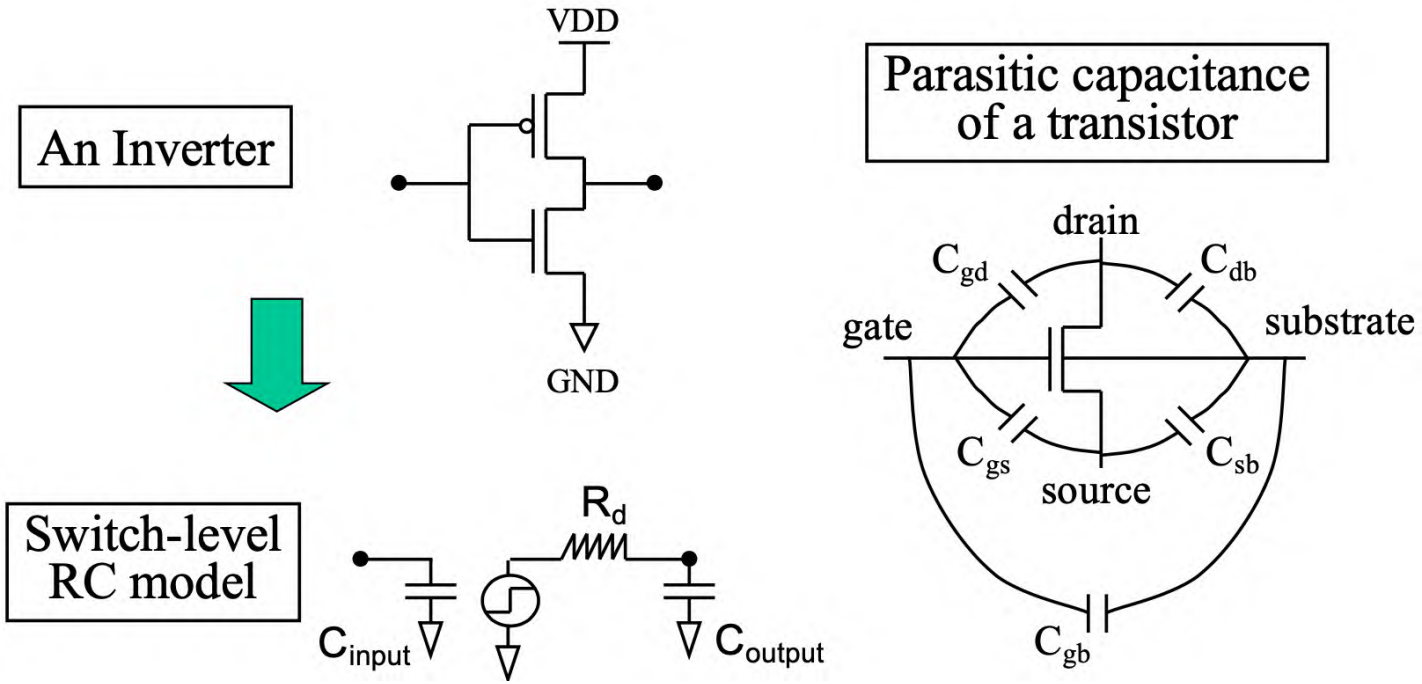
- Cell/gate delay
  - Delay fall/rise
  - Typical Vth: 50%





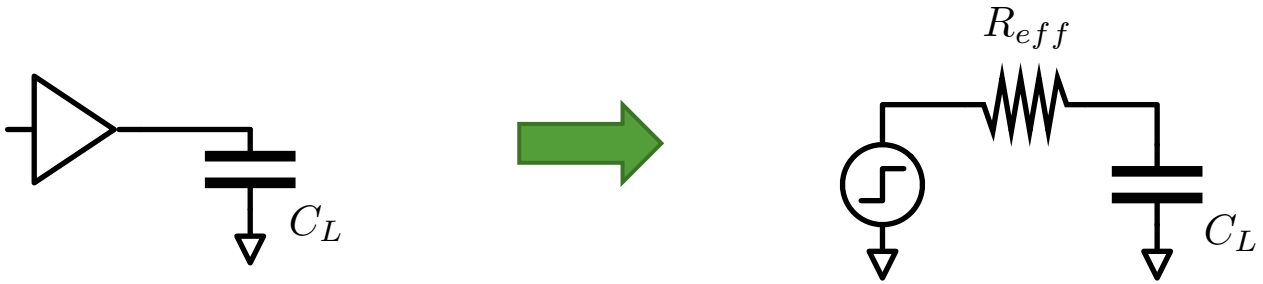
Simple Cell Model

- Want to model a gate by a simple RC circuit.
- Together with RC (or similar) model of interconnects, signal analysis can be done easily.



Effective Resistance

- Model the driver by an effective linear resistance
  - Select an appropriate load C_L
  - Simulate and get 50% delay
  - Match the delay by that of an equivalent RC circuit ($R_{eff}C_L$)



- Effective resistance may depend on input slope

Driver Characterization

- Table lookup model
  - Pre-characterize driver delay and transition by
    - input transition time t_t
    - total load capacitance C_L
  - typical entry: { t_t, C_L, (t_{delay}, $t_{rise/fall}$)}
- Table lookup and interpolation to obtain delay and transition time
- Can be very accurate
- Costly to create and store a multi-dimensional table

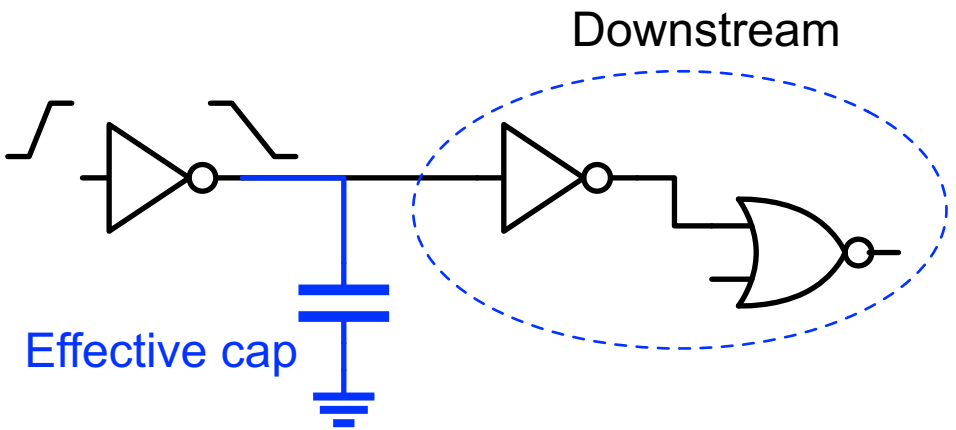
Driver Characterization

- k-factor equations for delay and output transition [Ousterhout, 1984]
  - $t_{delay} = (k_1 + k_2 \cdot C_L) \cdot t_t + k_3 \cdot C_L^3 + k_4 \cdot C_L + k_5$
  - $t_{fall} = (k_1' + k_2' \cdot C_L) \cdot t_t + k_3' \cdot C_L^3 + k_4' \cdot C_L + k_5'$

- Determine coefficients $k_i \& k_i'$
  - SPICE simulations for different combinations of input transition and load
  - Curve fitting by linear regression of least square fits

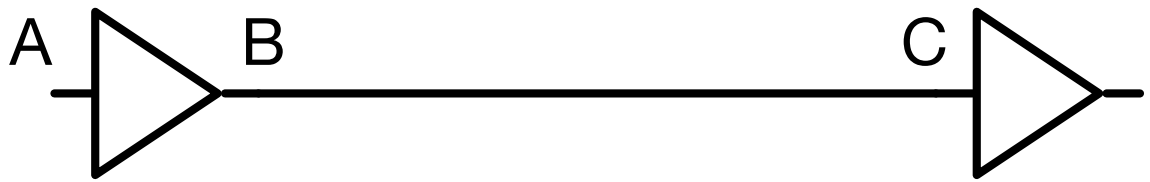
Cell Delay

- Typical delay model: nonlinear delay model (NLDM)
  - Delay = **f**(input slew, output load)
  - Slew = **g**(input slew, output load)
  - **f** and **g** are nonlinear functions, usually a look-up table



Put Cells and Nets Together

- Stage delay = delay from driver input A to loading gate C
- Two cases
  - **When interconnect resistance = 0**
  - **When interconnect = distributed RC**

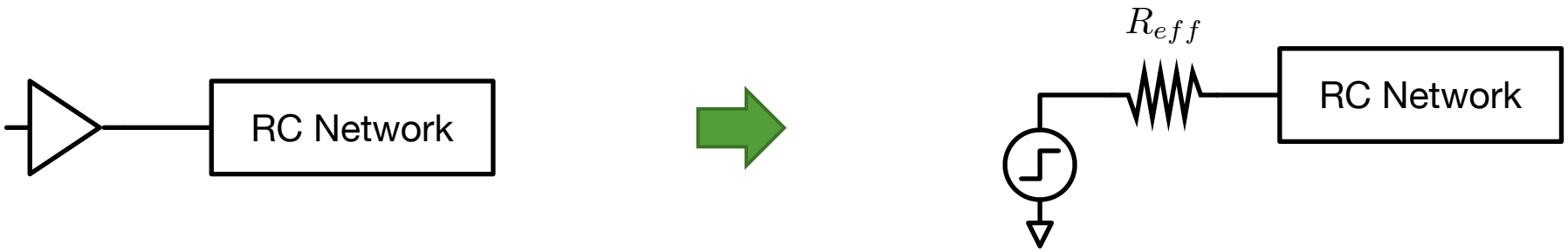


When Interconnect Resistance = 0

- C_L = total interconnect capacitance + sink capacitances
- Effective resistance model
  - Stage delay = $R_{eff}C_L$
- Pre-characterized driver model
  - Compute stage delay from table lookup or k-factor equations

When Interconnect = Distributed RC

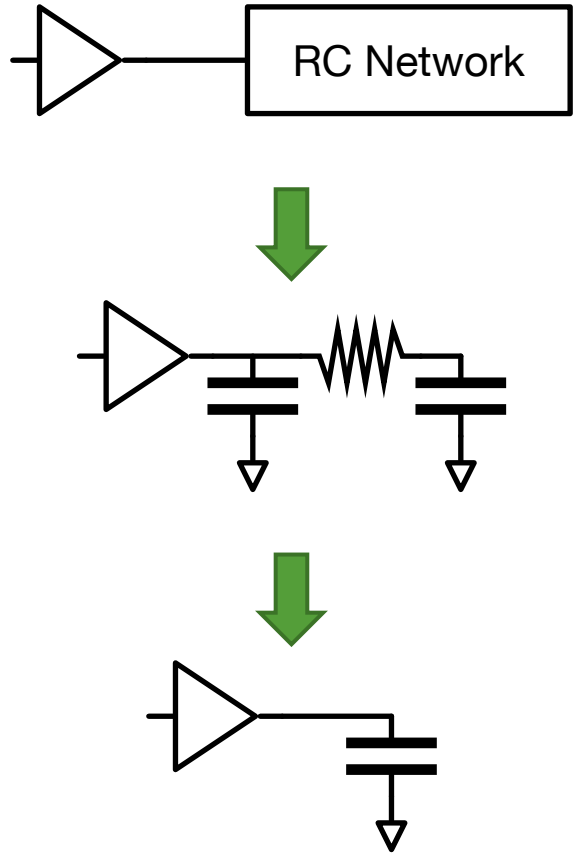
- Effective resistance model
  - **Augment the RC network with the effective driver resistance**



- Compute stage delay using RC delay models

Distributed RC with Pre-characterized Driver Model

- Compute driver delay and interconnect delay separately
- **Compute the effective capacitance seen by driver**
  - **Difficulty: must capture shielding effect of interconnect resistance**
- **Solution:**
  - Transform interconnect circuit into a π-model
  - Compute effective capacitance from π-model
- **TIP: Important! Not many people know how this is done. Must know this at least conceptually!**

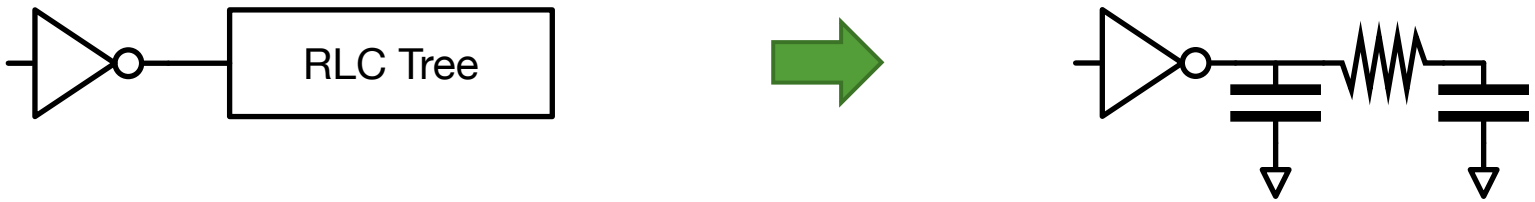


Construction of π-Model

- [O'Brian-Savarino, 89]
- Bottom-up computation of first three moments of driving point admittance y_1, y_2, y_3
  - i.e., moments of response current under voltage impulse at input
  - $C_1 = \frac{y_2^2}{y_3}, C_2 = y_1 - C_1, R = -\frac{y_3^2}{y_2^3}$

Construction of π-Model

- [O'Brian-Savarino, 89]
- For an unbranched uniform distributed RC segment
  - $C_1 = \frac{5C_w}{6}$
  - $C_2 = \frac{C_w}{6}$
  - $R = \frac{12R_w}{25}$



Effective Capacitance Computation

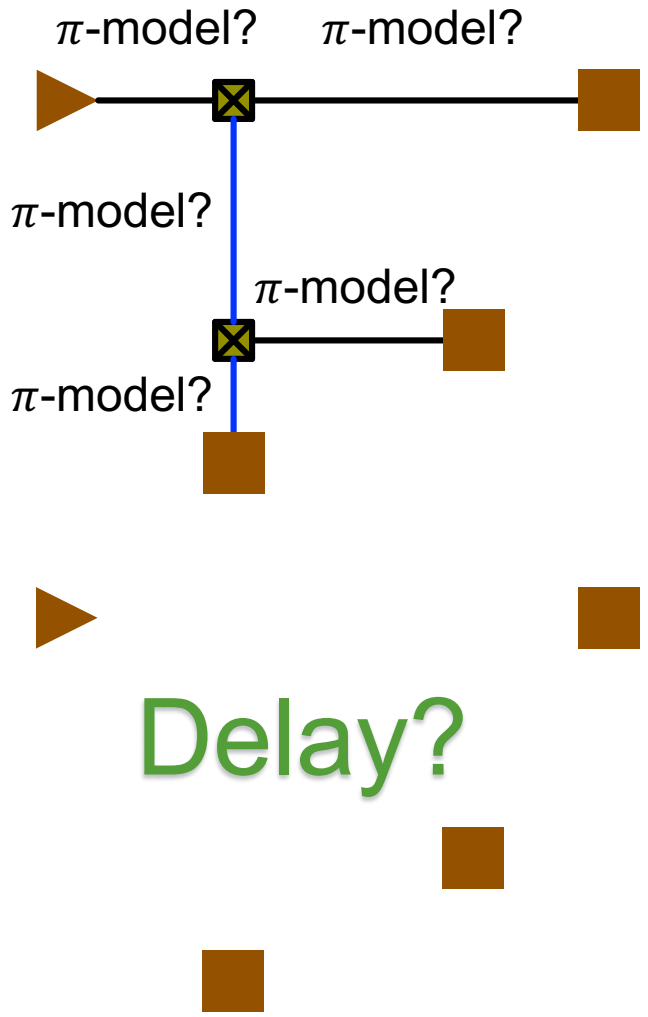
- [Qian-Pullela-Pileggi, 94]
- Transform π-model to effective capacitance model
- **Iterative computation of effective cap., t_{delay} & $t_{rise/fall}$**
  - **Start with C_{eff} = total interconnect + sink capacitances**
  - **Get** tdelay & trise/fall **from pre-characterized driver model**
  - $C_{eff} = f(R, C_1, C_2, t_{delay}, t_{rise/fall})$

Open Problems for Delay Modeling

- How to model cell/net delay accurately and efficiently

- If we are given routing of nets
  - Require **RC extraction** to get accurate RC networks
  - Require **SPICE simulation** to get accurate delay
  - Timing-consuming, e.g., current source model

- What if we are NOT given the routing
  - Pin locations only
  - E.g., in placement



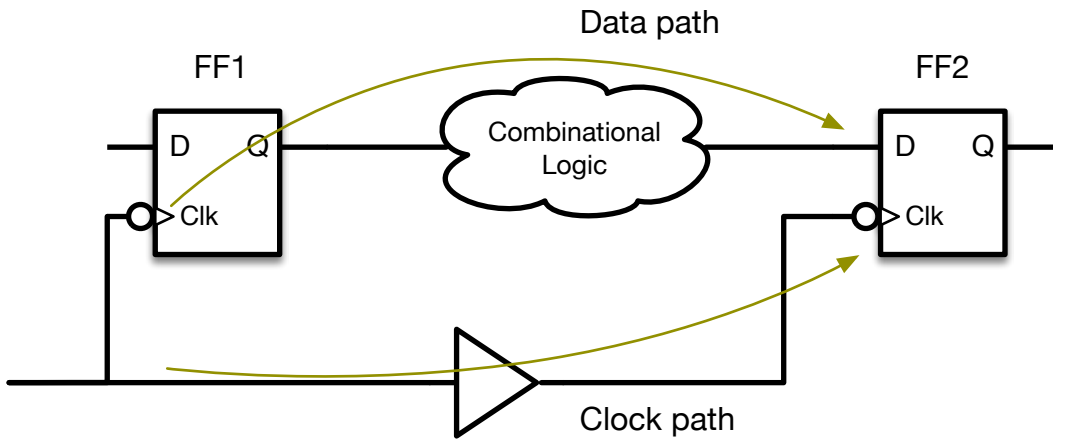
Now, we have gate and wire delay

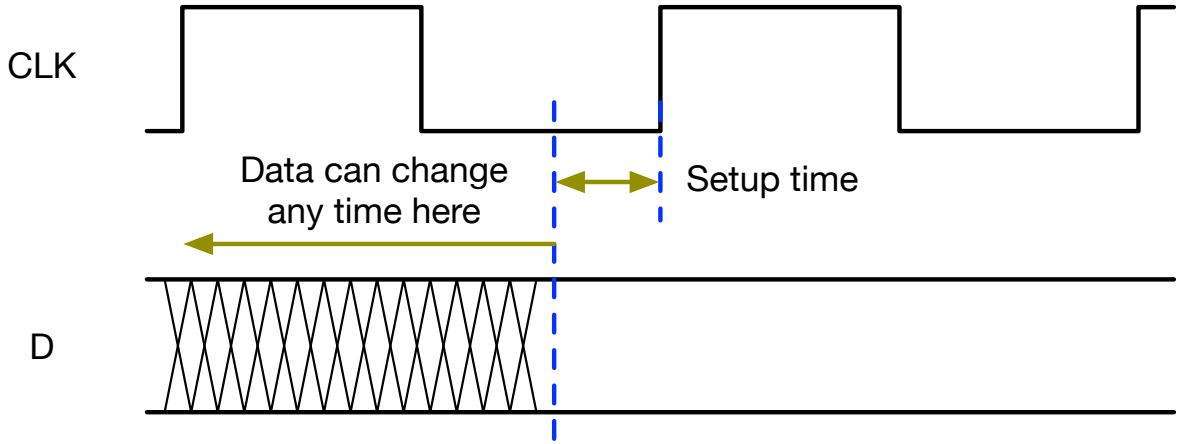
- Then what?
  – Need to be tied together to “time” the entire circuit design
- Timing:
  – **Static timing analysis (no input vector involved)**
  – Dynamic timing – very time consuming as you need to worry about input vectors/transitions, false paths, ...

Setup Time

- A setup constraint specifies how much time is necessary for data to be available at the input of a sequential device **before** the clock edge that captures the data in the device
- This constraint enforces a **maximum** delay on the data path relative to the clock path

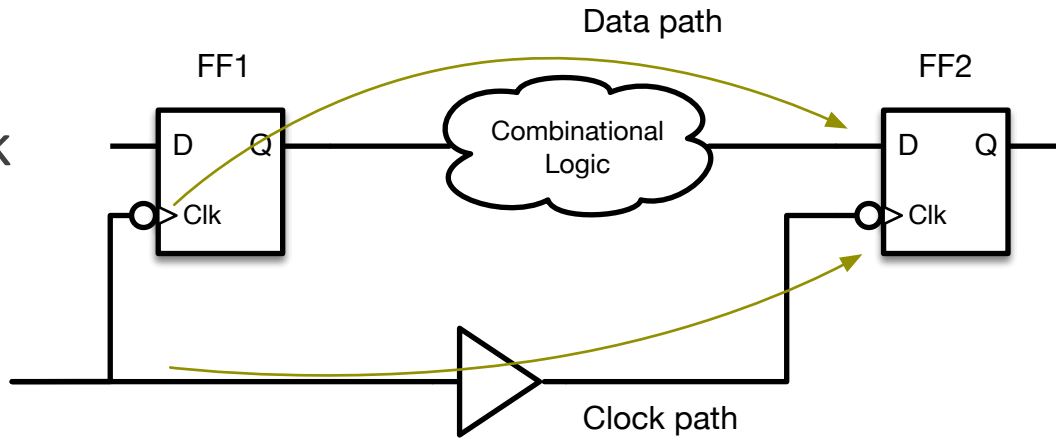
How to fix a setup time violation?



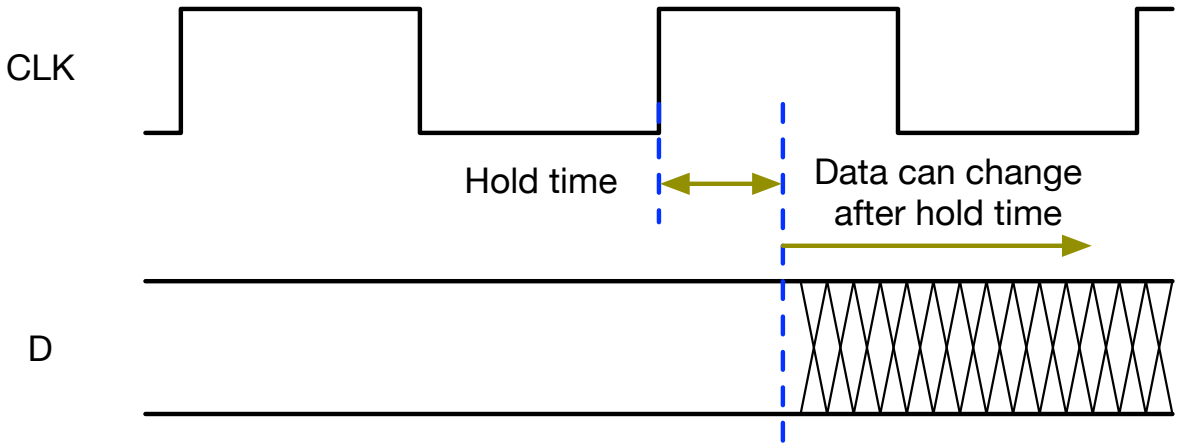


Hold Time

- A hold constraint specifies how much time is necessary for data to be stable at the input of a sequential device **after** the clock edge that captures the data in the device
- This constraint enforces a **minimum** delay on the data path relative to the clock path



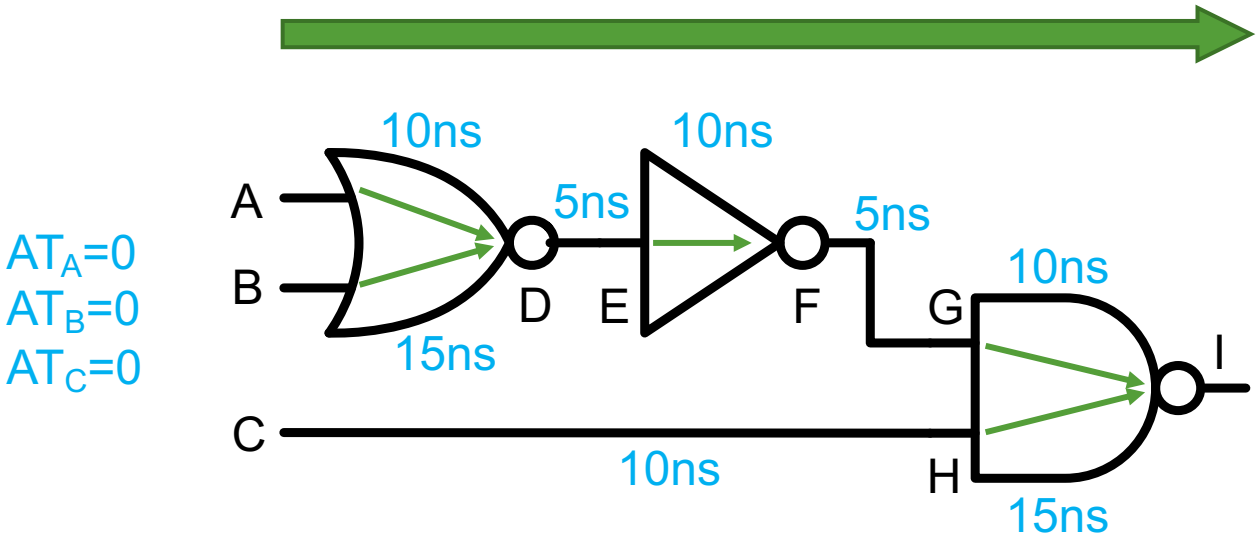
How to fix a hold time violation?



Arrival Time, Required Arrival Time & Slack

- Arrival time (AT)
  - Each node is updated with its parents

Traversal with topological order

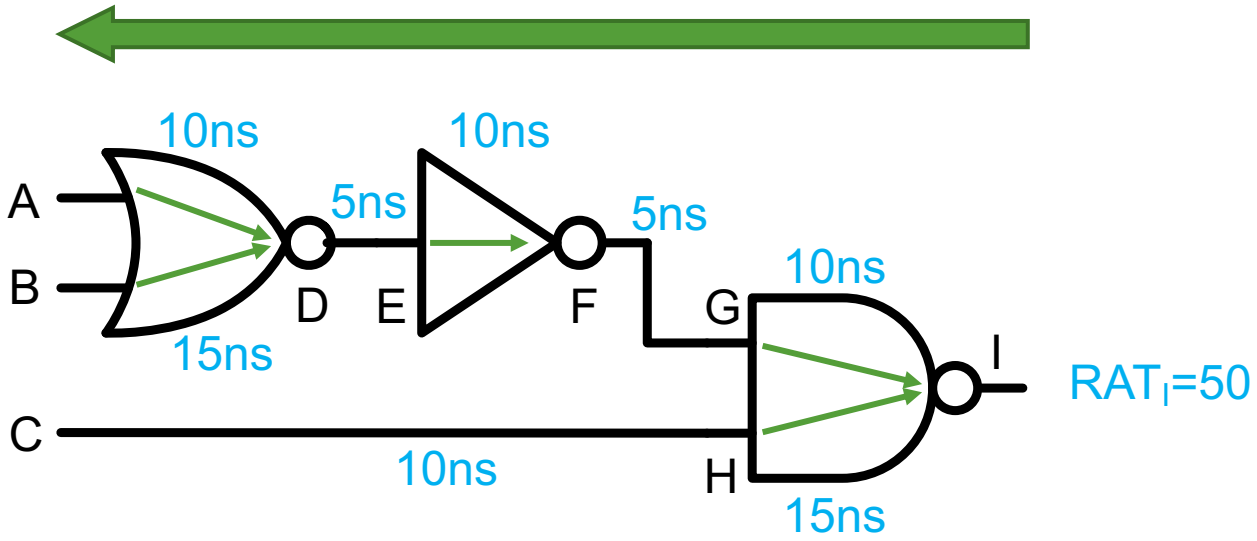


$AT_D = \max(AT_A+10, AT_B+15) = 15$
$AT_E = AT_D+5 = 20$
$AT_F = AT_E+10 = 30$
$AT_G = AT_F+5 = 35$
$AT_H = AT_C+10 = 10$
$AT_I = \max(AT_G+10, AT_H+15) = 45$

Arrival Time, Required Arrival Time & Slack

- Required arrival time (RAT)
  - Each node is updated by its children

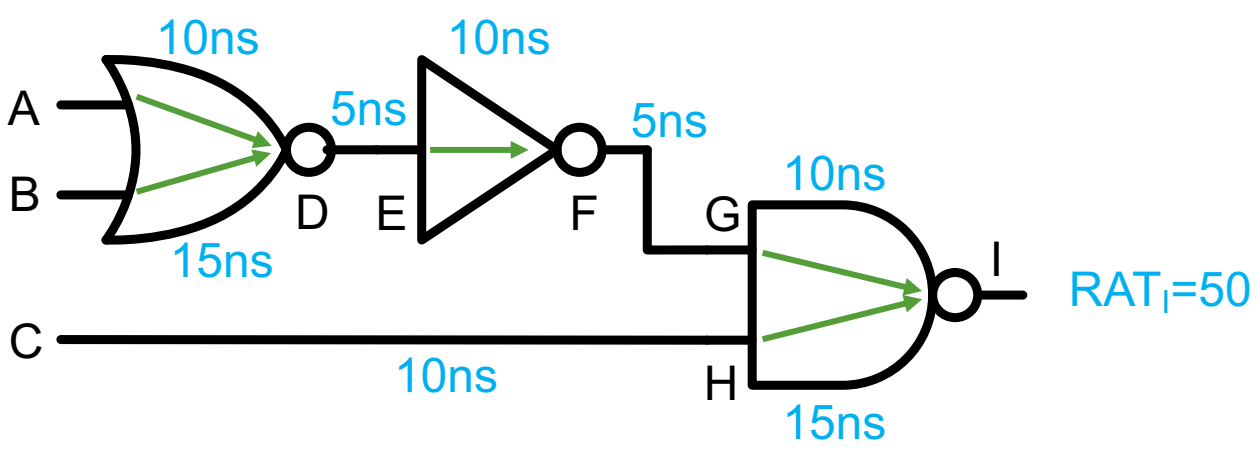
Traversal with topological order



$RAT_G = RAT_I - 10 = 40$
$RAT_H = RAT_I - 15 = 35$
$RAT_F = RAT_G - 5 = 35$
$RAT_E = RAT_F - 10 = 25$
$RAT_D = RAT_E - 5 = 20$
$RAT_A = RAT_D - 10 = 10$
$RAT_B = RAT_D - 15 = 5$
$RAT_C = RAT_H - 10 = 25$

Arrival Time, Required Arrival Time & Slack

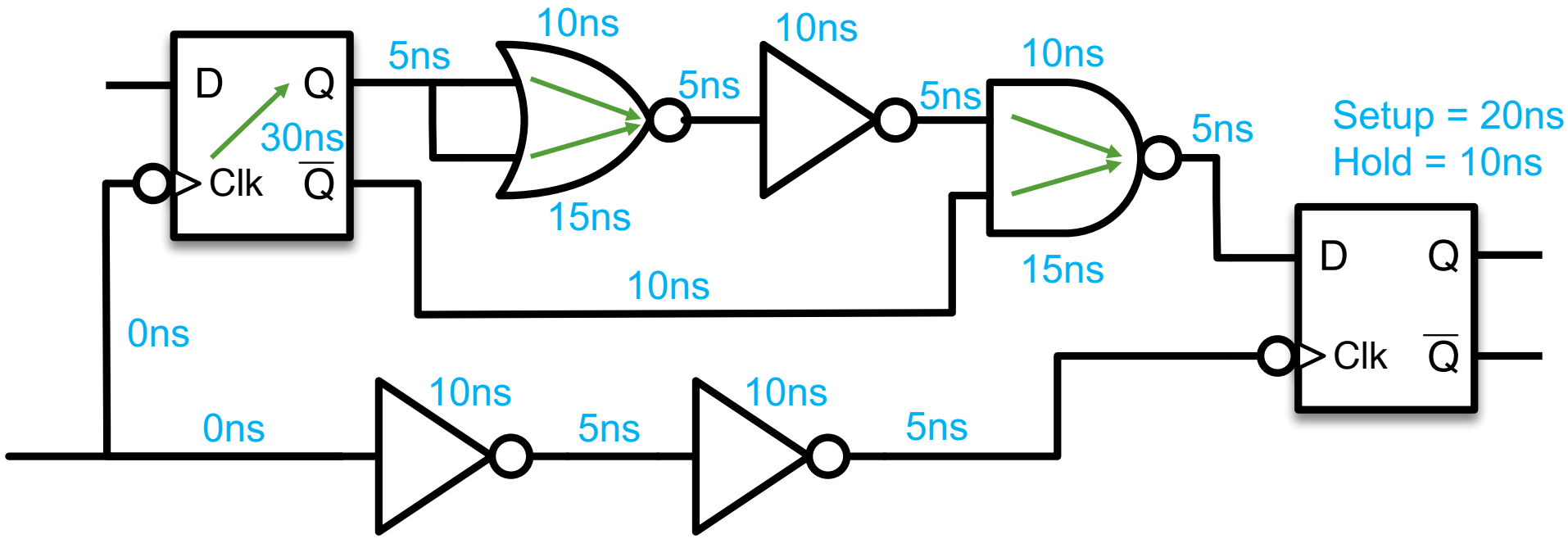
- Slack
  - $Slack_i = RAT_i - AT_i$
  - Critical path: B → D → E → F → G → I
  - **Negative** slack means timing violation!



Slack$_A$ = 10 – 0 = 10
Slack$_B$ = 5 – 0 = 5
Slack$_C$ = 25 – 0 = 25
Slack$_D$ = 20 – 15 = 5
Slack$_E$ = 25 – 20 = 5
Slack$_F$ = 35 – 30 = 5
Slack$_G$ = 40 – 35 = 5
Slack$_H$ = 35 – 10 = 25
Slack$_I$ = 50 – 45 = 5

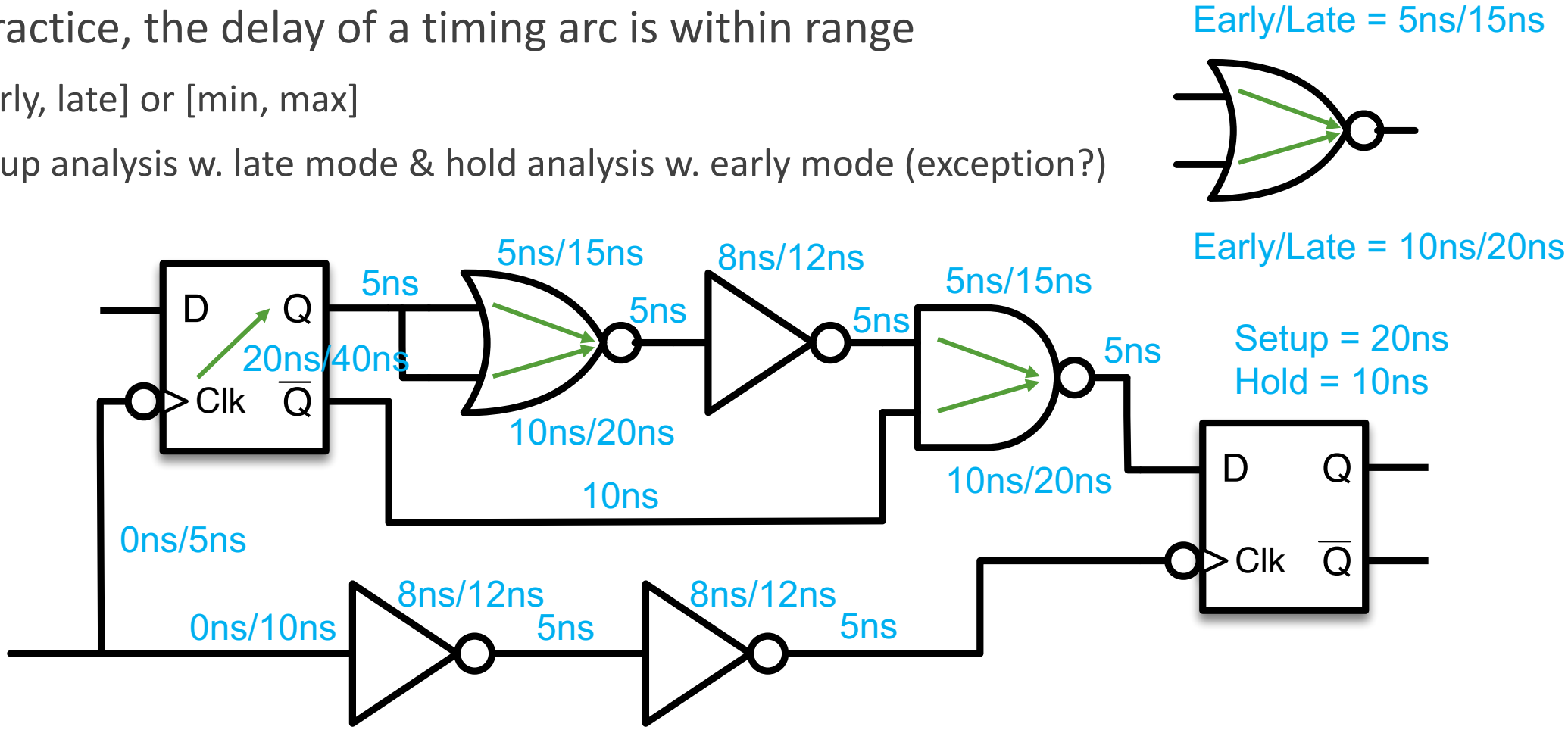
How Does STA Determine the Frequency

- Given delay of timing arcs and setup/hold time of FFs
- Can you compute the maximum clock frequency?



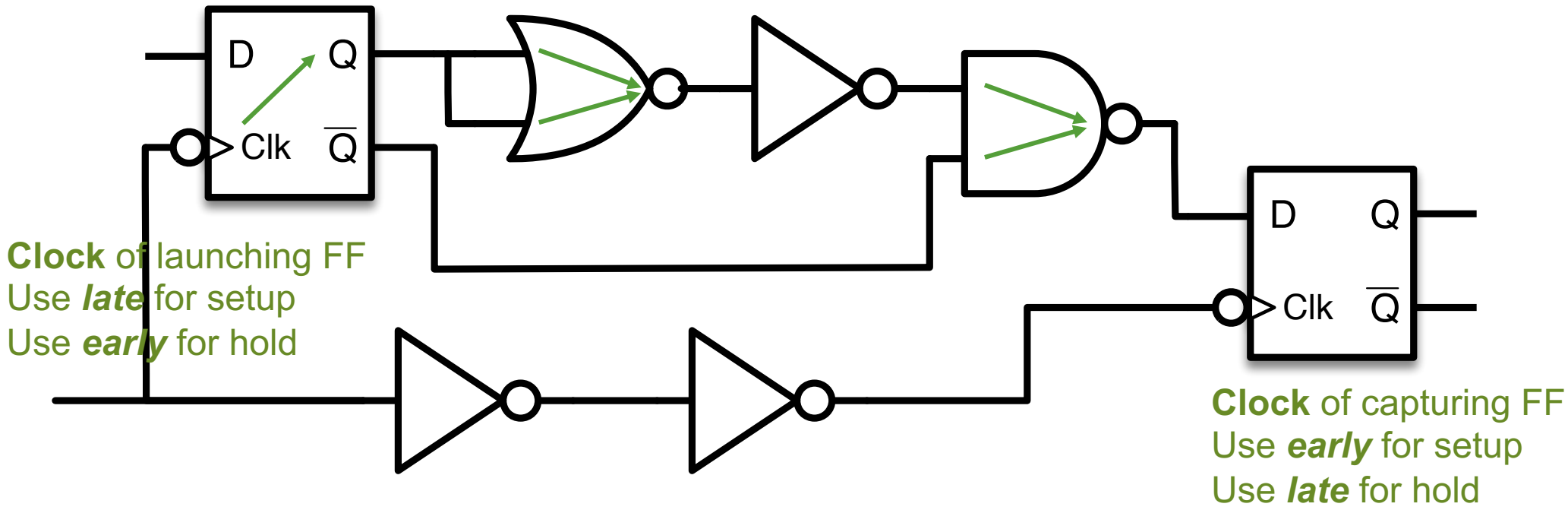
Early/Late Modes

- In practice, the delay of a timing arc is within range
  - [early, late] or [min, max]
  - Setup analysis w. late mode & hold analysis w. early mode (exception?)



Early/Late Modes

- In practice, the delay of a timing arc is within range
  - [early, late] or [min, max]
  - Setup analysis w. late mode & hold analysis w. early mode (exception?)



Early/Late Modes

- In practice, the delay of a timing arc is within range

Setup tests

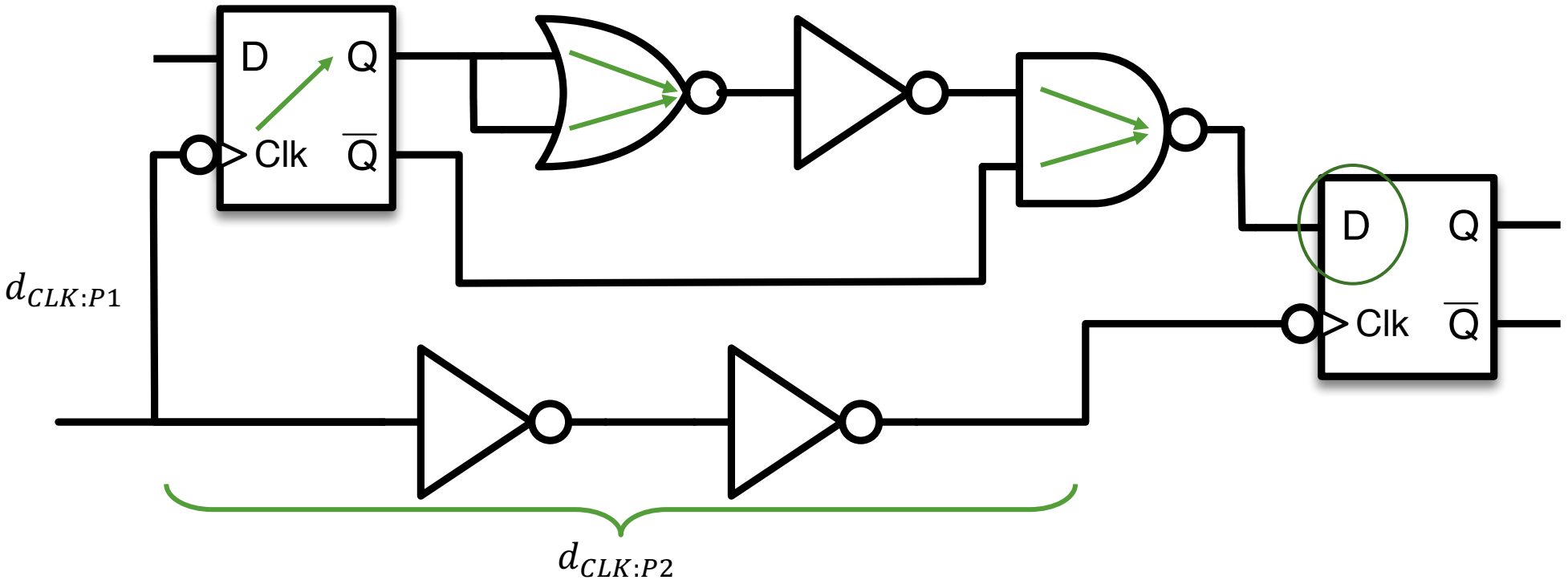
$$AT_D^{late} = d_{CLK:P1}^{late} + d_{Clk \to Q} + d_{comb}^{late}$$

$$RAT_{setup} = RAT_D^{late} = T + d_{CLK:P2}^{early} - t_{setup}$$

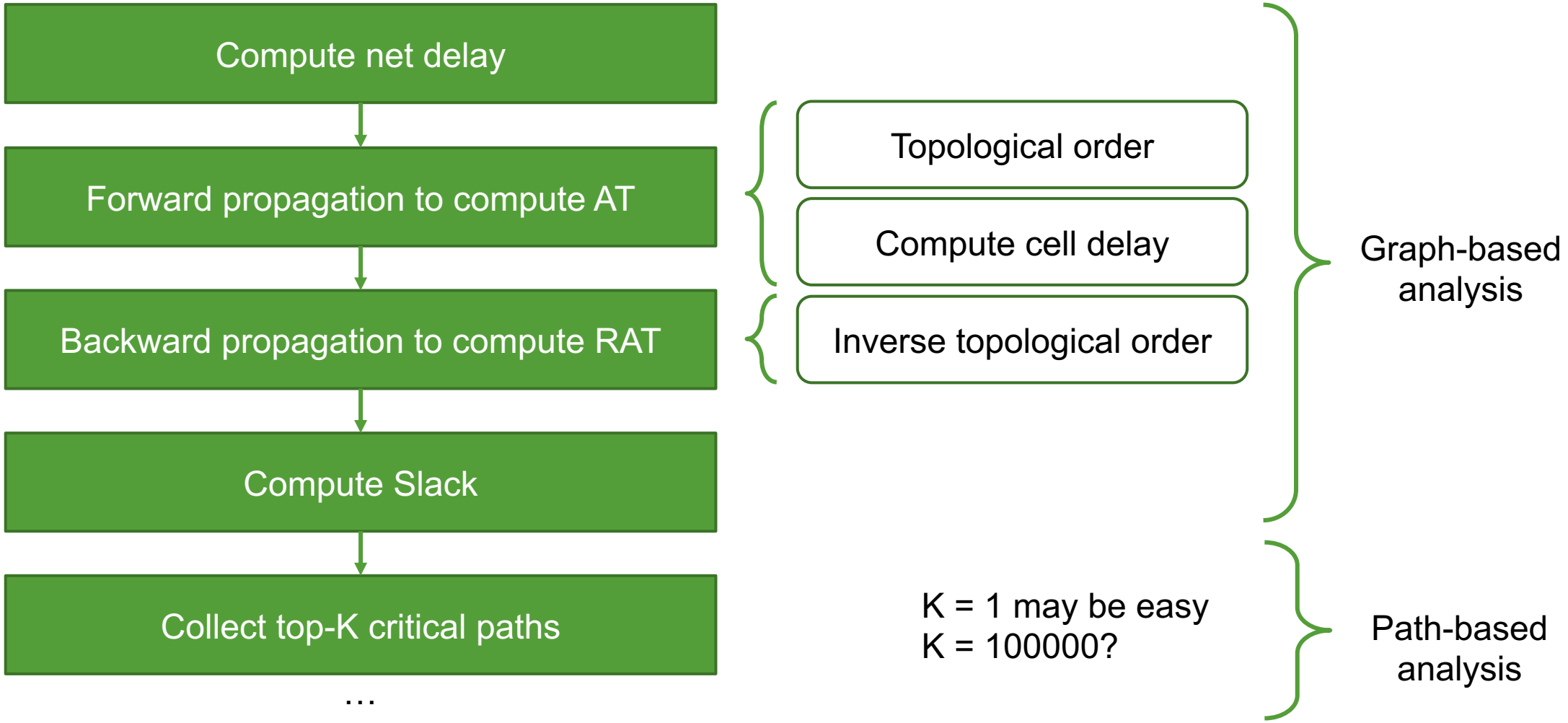
Hold tests

$$AT_D^{early} = d_{CLK:P1}^{early} + d_{Clk \to Q} + d_{comb}^{early}$$

$$RAT_{hold} = RAT_D^{early} = d_{CLK:P2}^{late} + t_{hold}$$



Graph-based Analysis

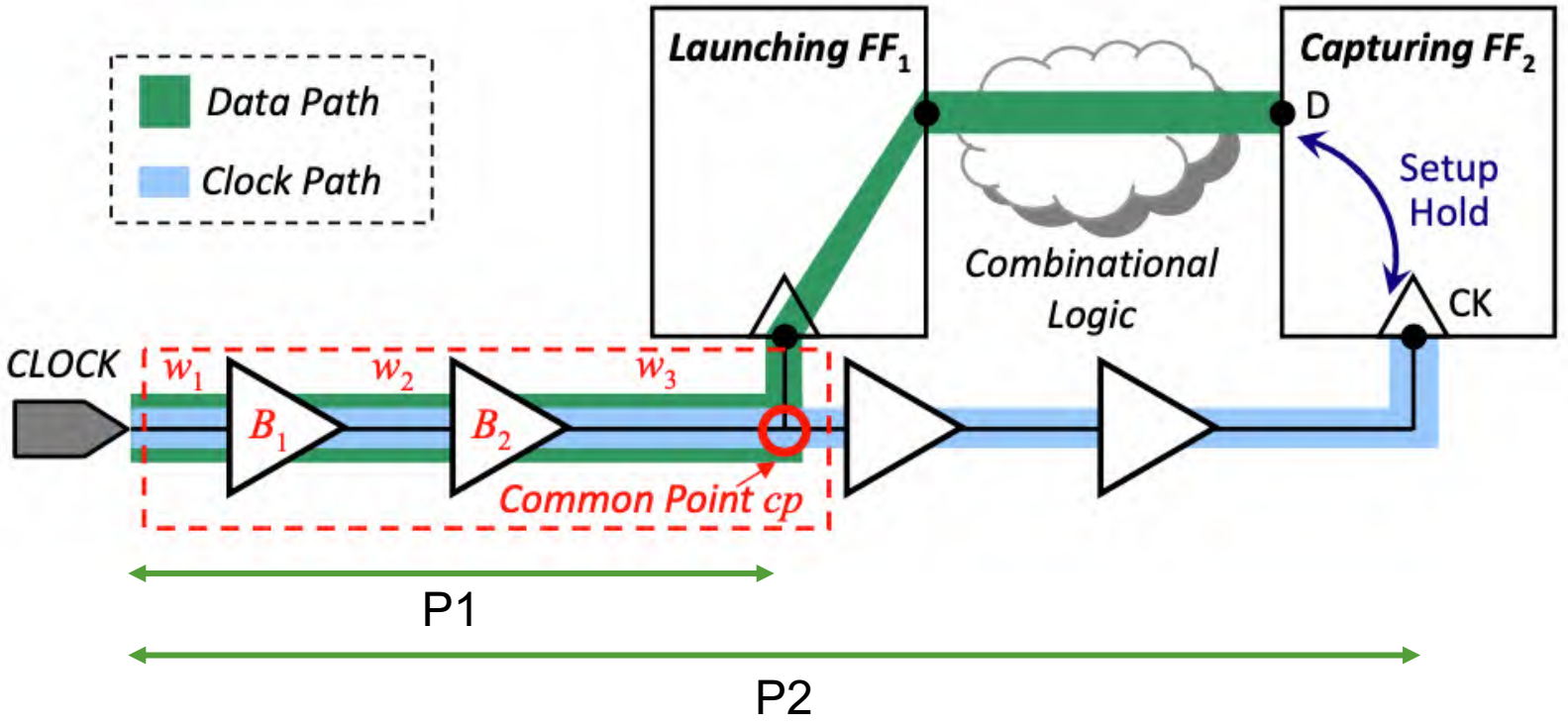


Path-based Analysis

- Given the results of graph-based analysis
- Extract top-K critical paths
- Considering common path pessimism removal (CPPR), noise, aging, …
- Timing-consuming when K is large
  - 1~100000

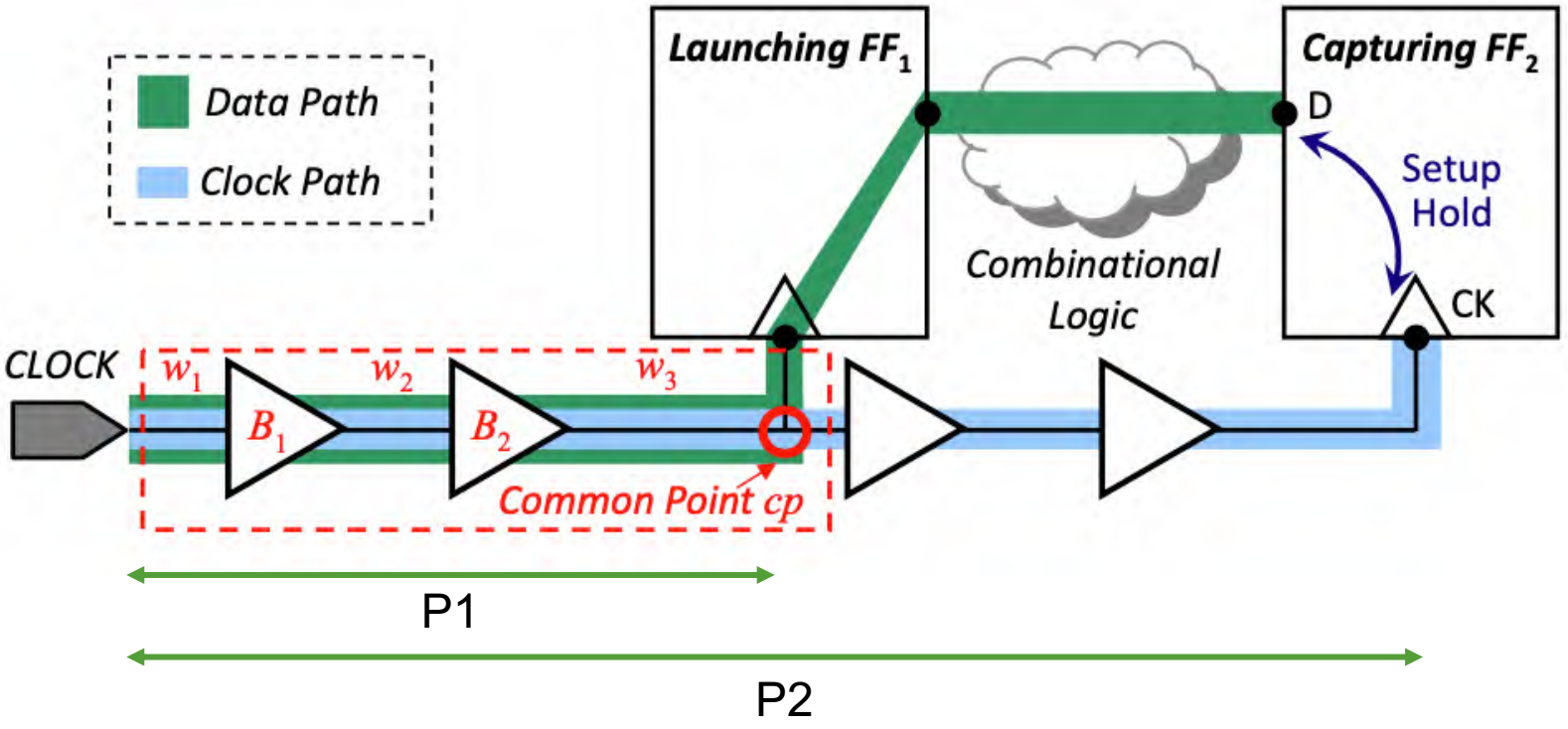
Common Path Pessimism Removal (CPPR)

- Graph-based analysis is too pessimistic
  - In hold analysis, use P1 as early and P2 as late
  - P1 CANNOT be early and late at the same time



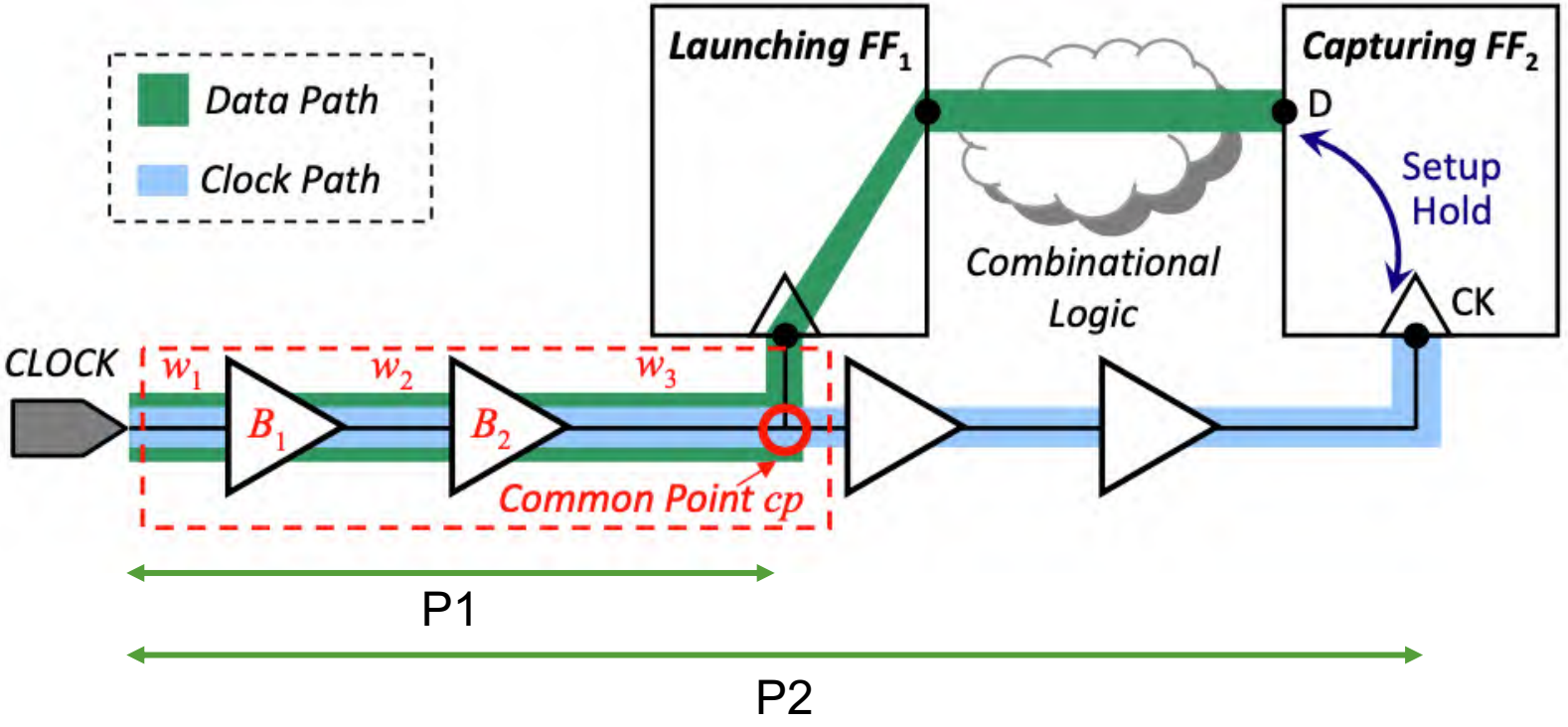
Common Path Pessimism Removal (CPPR)

- Remove the pessimism on P1
  - Hold tests: $credit_{test}^{hold} = slack_{post-CPPR}^{hold} - slack_{pre-CPPR}^{hold}$
  - Setup tests: $credit_{test}^{setup} = slack_{post-CPPR}^{setup} - slack_{pre-CPPR}^{setup}$



Common Path Pessimism Removal (CPPR)

- Remove the pessimism on P1
  - Hold tests: $credit^{hold} = AT_{cp}^{late} - AT_{cp}^{early}$
  - Setup tests: $credit^{setup} = \sum_{p \in CP}(d_p^{late} - d_p^{early}), CP = \{w_1, B_1, w_2, B_2, w_3\}$



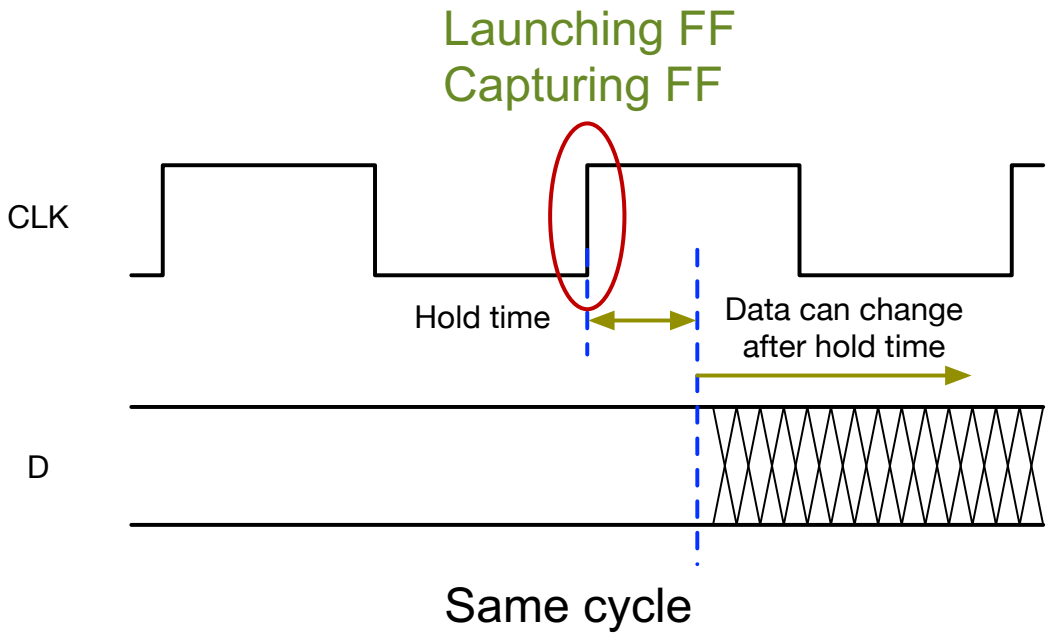
Common Path Pessimism Removal (CPPR)

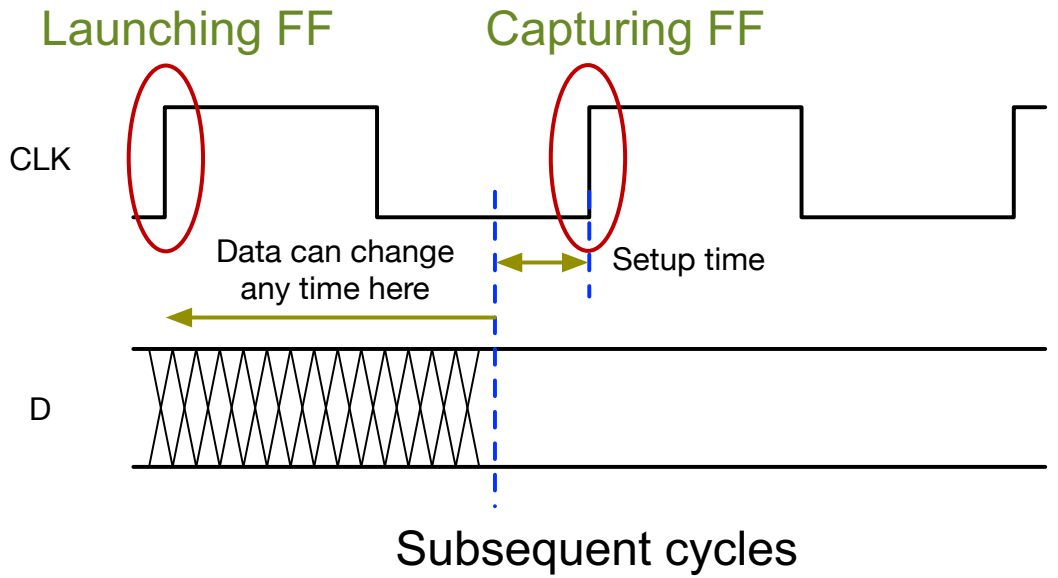
- Remove the pessimism on P1
  - Hold tests: $credit^{hold} = AT_{cp}^{late} - AT_{cp}^{early}$
  - Setup tests: $credit^{setup} = \sum_{p \in CP}(d_p^{late} - d_p^{early}), CP = \{w_1, B_1, w_2, B_2, w_3\}$

w. credit for clock root

w.o. credit for clock root

What's the difference?
Why different?





Early/late delay of clock root **CANCELLED OUT**

Exercise

Pre-CPPR Slack

$slack^{late}_{FF3:D}|_{DP1}$

$slack^{late}_{FF3:D}|_{DP2}$

Post-CPPR Slack

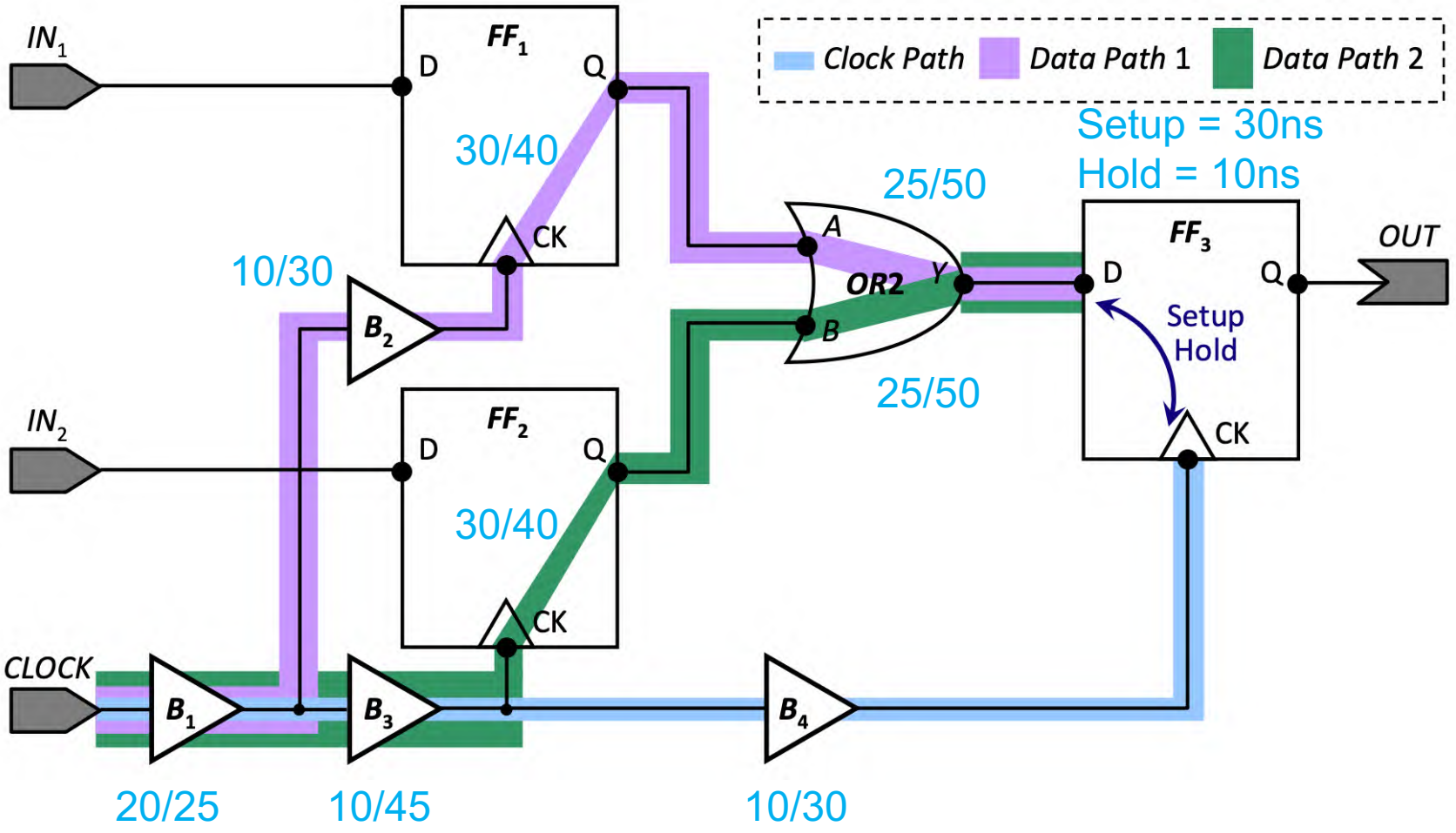
$credit_{setup|DP1}$

$credit_{setup|DP2}$

$slack^{late}_{FF3:D}|^{Post-CPPR}_{DP1}$

$slack^{late}_{FF3:D}|^{Post-CPPR}_{DP2}$

$T_{CLK} = 120$



Exercise

$$slack^{late}_{FF3:D} = RAT^{late}_{FF3:D} - AT^{late}_{FF3:D} = T_{CLK} + AT^{early}_{FF3:CK} - t^{setup}_{FF3} - AT^{late}_{FF3:D}$$

$$slack^{late}_{FF3:D}|_{DP1} = T_{CLK} + \left(d^{early}_{B1} + d^{early}_{B3} + d^{early}_{B4}\right) - t_{setup} - \left(d^{late}_{OR2:A\rightarrow OR2:Y} + d^{late}_{FF1:CK\rightarrow FF1:Q} + d^{late}_{B1} + d^{late}_{B2}\right)$$
$$= 120 + (20 + 10 + 10) - 30 - (50 + 40 + 30 + 25) = -15$$

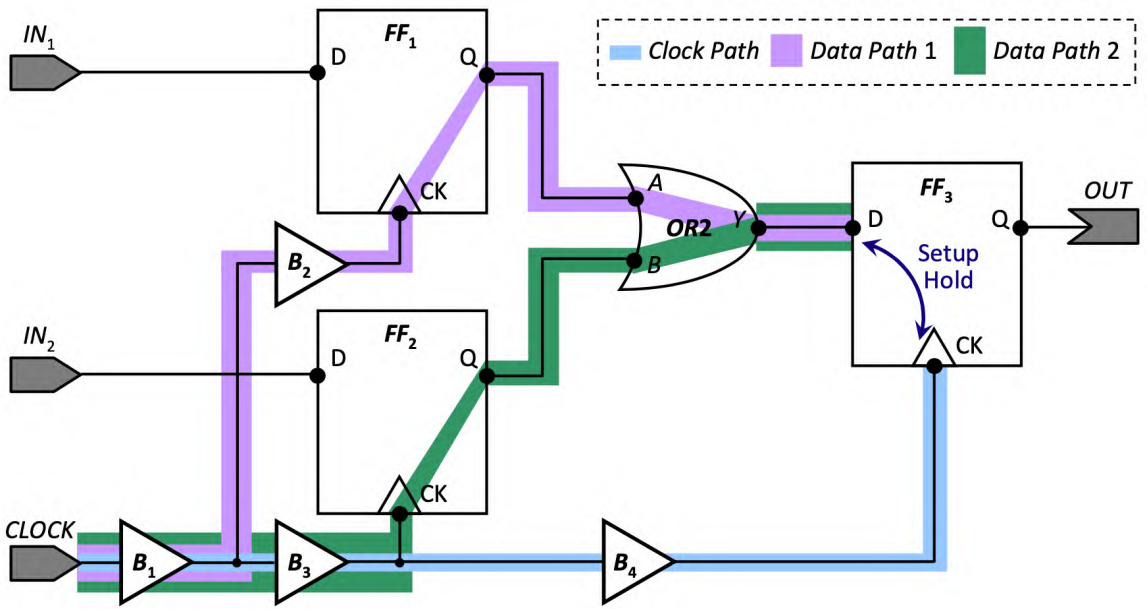
$$slack^{late}_{FF3:D}|_{DP2} = T_{CLK} + \left(d^{early}_{B1} + d^{early}_{B3} + d^{early}_{B4}\right) - t_{setup} - \left(d^{late}_{OR2:B\rightarrow OR2:Y} + d^{late}_{FF2:CK\rightarrow FF2:Q} + d^{late}_{B1} + d^{late}_{B3}\right)$$
$$= 120 + (20 + 10 + 10) - 30 - (50 + 40 + 45 + 25) = -30$$

$$credit_{setup}|_{DP1} = \left(d^{late}_{B1} - d^{early}_{B1}\right) = (25 - 20) = 5$$

$$credit_{setup}|_{DP2} = \left(d^{late}_{B1} - d^{early}_{B1}\right) + \left(d^{late}_{B3} - d^{early}_{B3}\right)$$
$$= (25 - 20) + (45 - 10) = 40$$

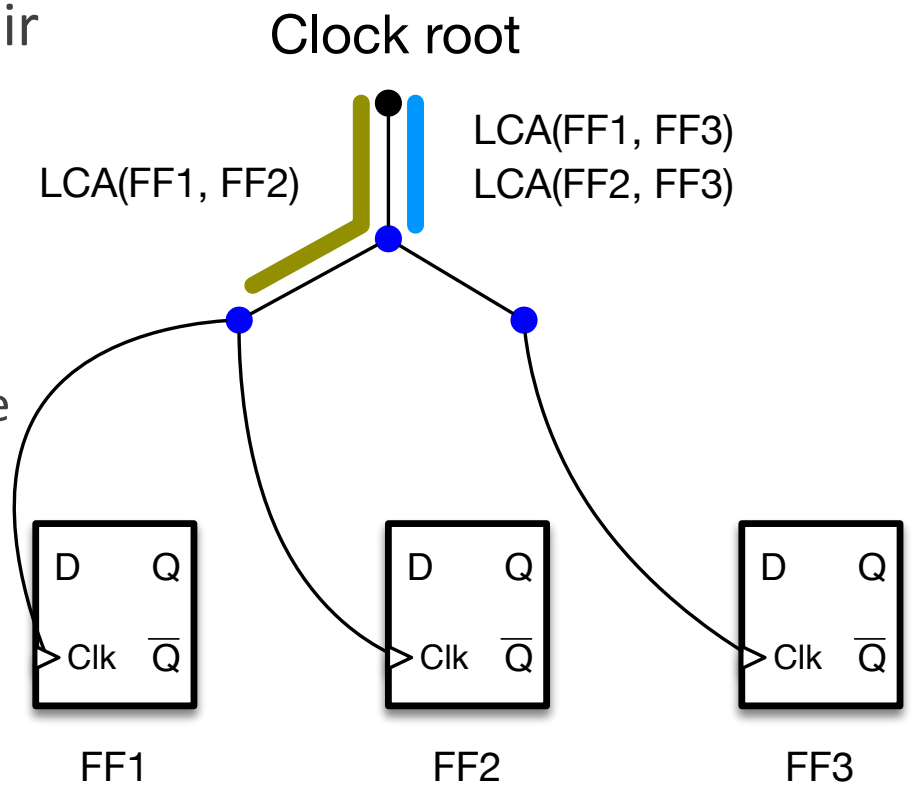
$$slack^{late}_{FF3:D}|^{Post-CPPR}_{DP1} = slack^{late}_{FF3:D}|_{DP1} + credit_{setup}|_{DP1}$$
$$= (-15 + 5) = -10$$

$$slack^{late}_{FF3:D}|^{Post-CPPR}_{DP2} = slack^{late}_{FF3:D}|_{DP2} + credit_{setup}|_{DP2}$$
$$= (-30 + 40) = 10$$



Basic Idea in CPPR Algorithms

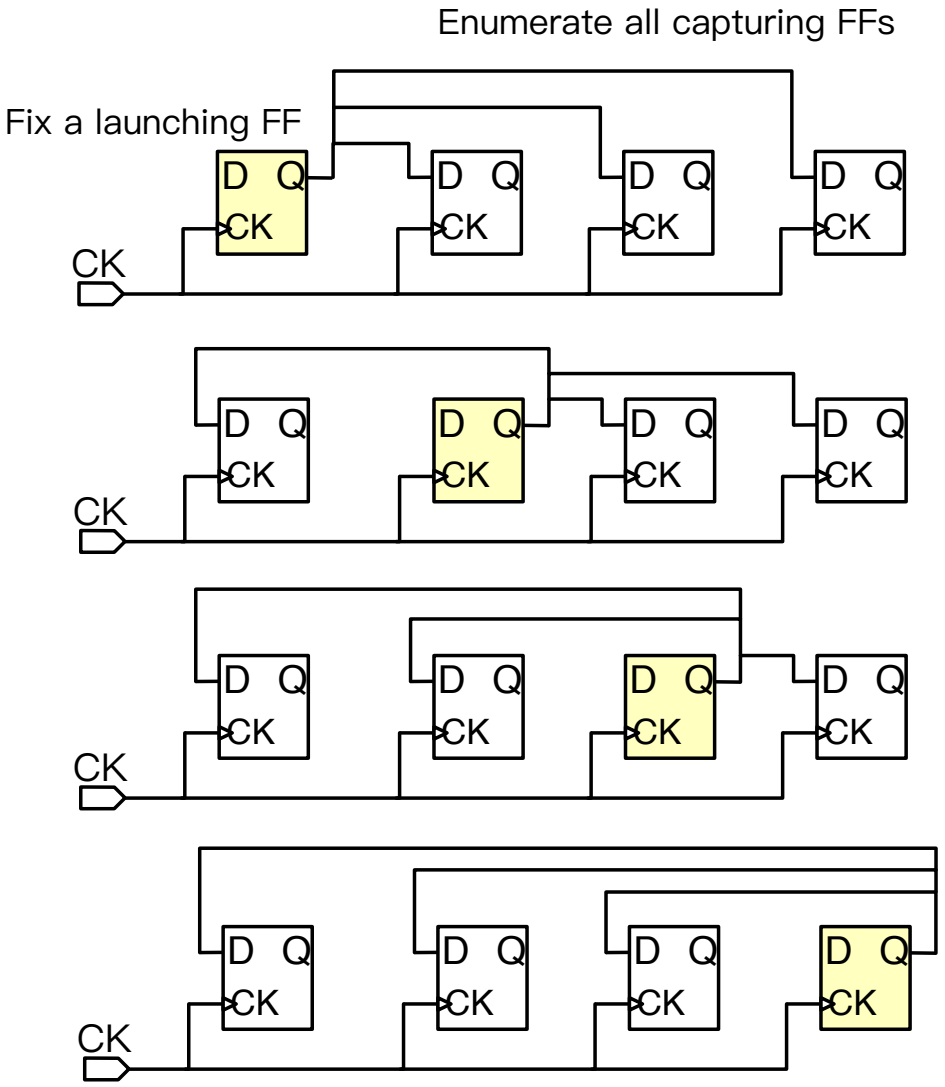
- Enumerate paths
  - Pairs of launching and capturing FFs
- Compute lowest common ancestor (LCA) for each pair
- Naïve method
  - Find a path from root to FF1:Clk and store it in a vector
  - Find a path from root to FF2:Clk and store it in a vector
  - Traverse both paths till the values in the vectors are the same
  - Time complexity $O(n)$, where n is #nodes in the clock tree
  - Better algorithms exist
- Generate top-k paths



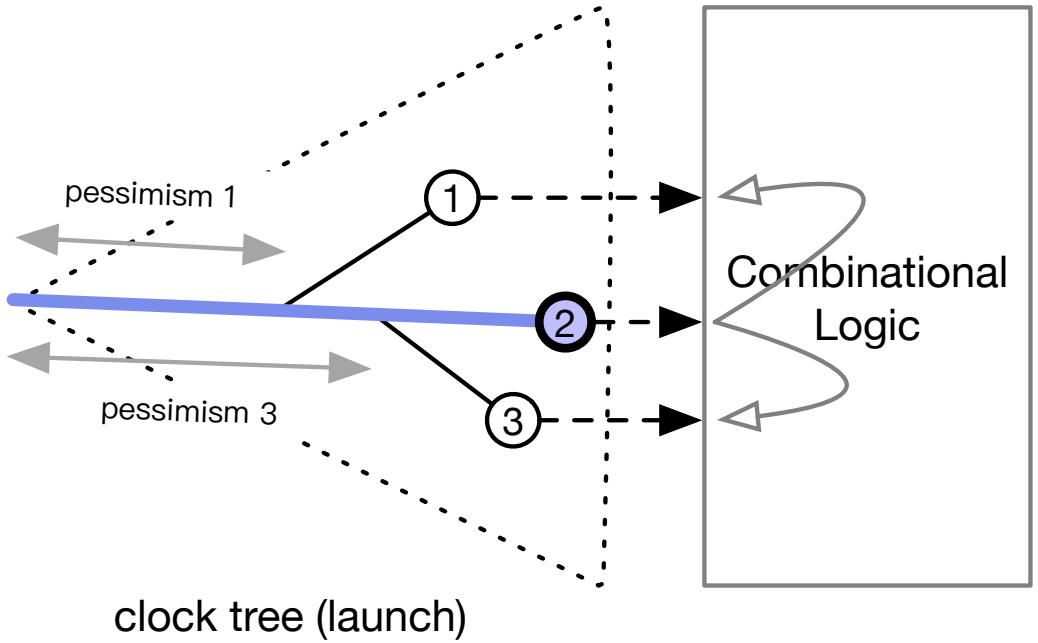
Improved CPPR Algorithms

- Tsung-Wei Huang, P.-C. Wu, and Martin Wong, "UI-Timer: An ultra-fast clock network pessimism removal algorithm", in Proc. ICCAD, pp. 596-599, 2014.
- P.-Y. Lee, I. H.-R. Jiang, C.-R. Li, W.-L. Chiu, and Y.-M. Yang, "iTimerC 2.0: Fast incremental timing and cppr analysis," in Proc. ICCAD. IEEE, 2015, pp. 890–894.
- B. Jin, G. Luo, and W. Zhang, "A fast and accurate approach for common path pessimism removal in static timing analysis," in Proc. ISCAS. IEEE, 2016, pp. 2623–2626.
- C. Peddawad, A. Goel, B. Dheeraj, and N. Chandrachoodan, "iitrace: A memory efficient engine for fast incremental timing analysis and clock pessimism removal," in Proc. ICCAD, 2015, pp. 903–909.
- T. Chung-Hao and M. Wai-Kei, "A fast parallel approach for common path pessimism removal," in Proc. ASPDAC, 2015, pp. 372–377.
- Z. Guo, M. Yang, T. Huang, and Y. Lin, "A Provably Good and Practically Efficient Algorithm for Common Path Pessimism Removal in Large Designs", in IEEE TCAD, 2021

CPPR with FF enumeration [Huang, ICCAD'14]

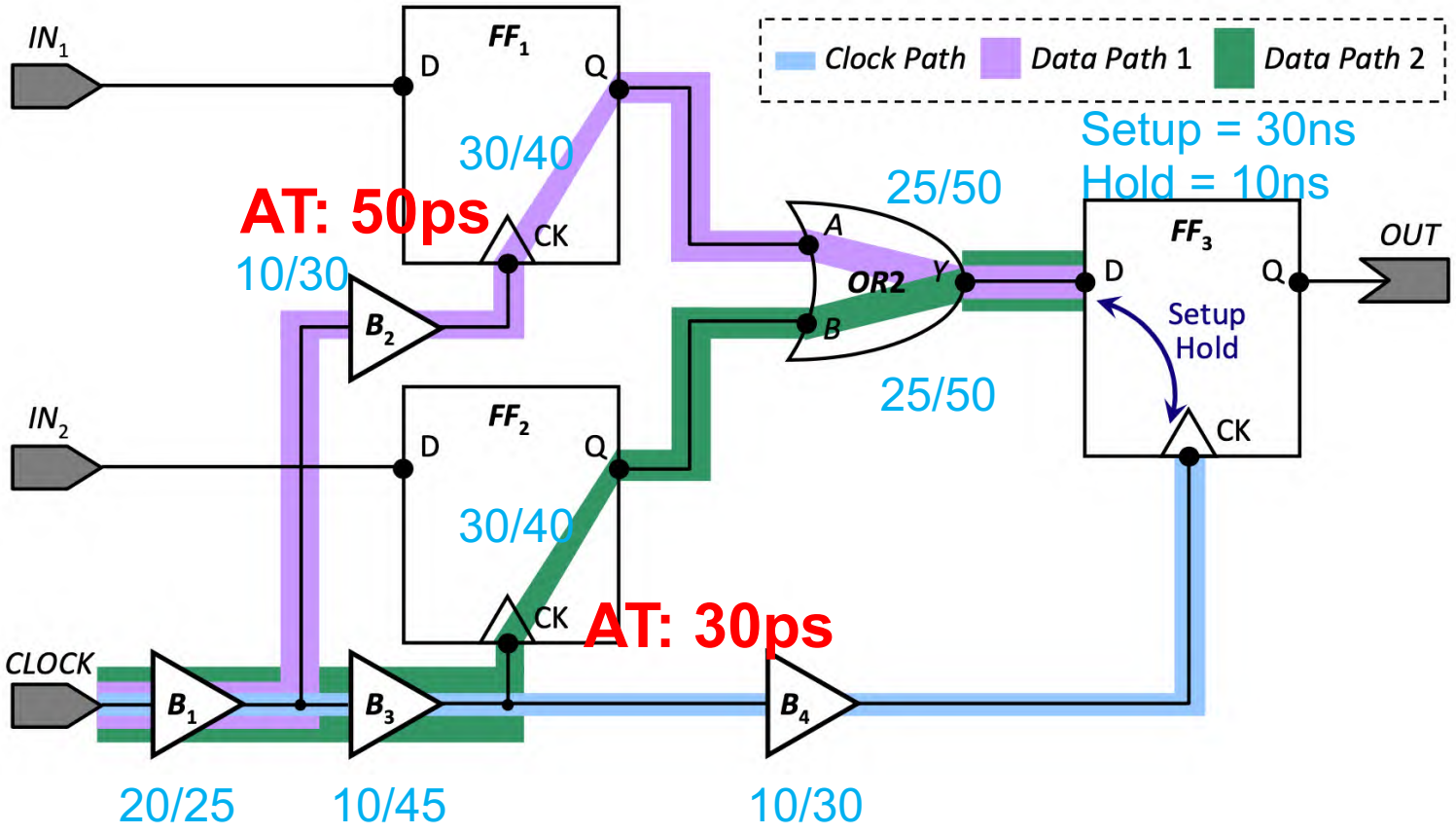


- Fix a launching FF and enumerate all capturing FFs (or, fix a capturing FF and enum. all launching FFs)
- Construct a Pessimism-Free Graph
  - Offset the arrival time by the CPPR credits in advance



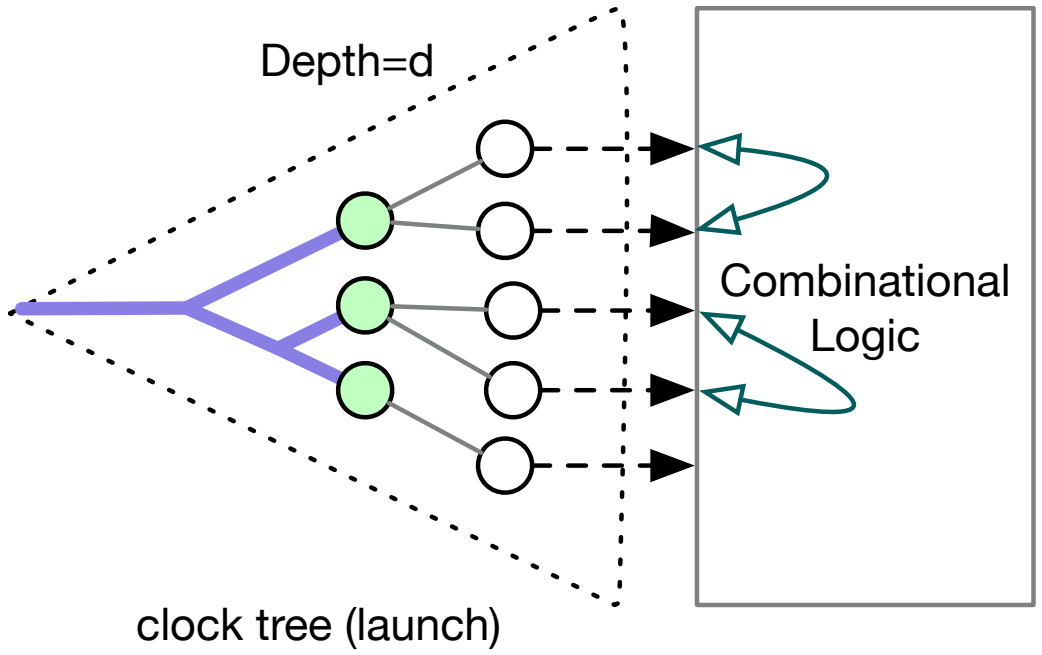
CPPR with FF enumeration [Huang, ICCAD'14]

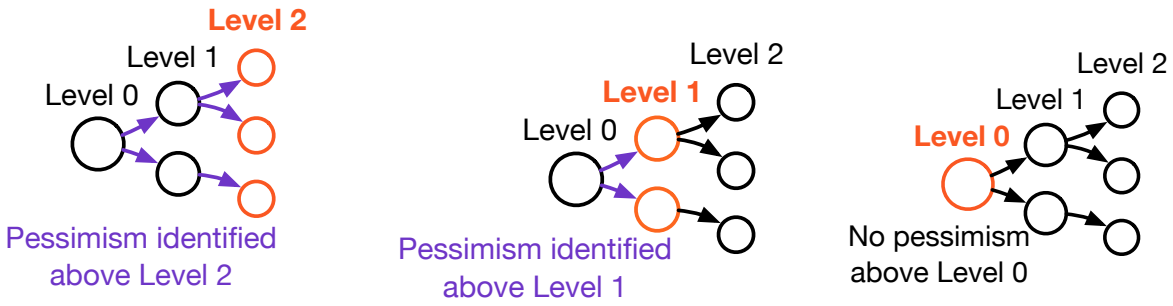
- Example
- Fix FF3 as capturing FF
- Compute credits:
  - FF1, FF3: 5ps
  - FF2, FF3: 40ps
- Add credits to arrival time
  - FF1.CK: 25 + 30 – 5 = 50ps
  - FF2.CK: 25 + 45 – 40 = 30ps
- Propagate the arrival time
  => Pessimism-Free Graph
- Compute all paths that arrive at FF3, ignoring other destinations; Repeat the process for other capturing FFs.



CPPR with Depth Enumeration [Guo, DAC'21]

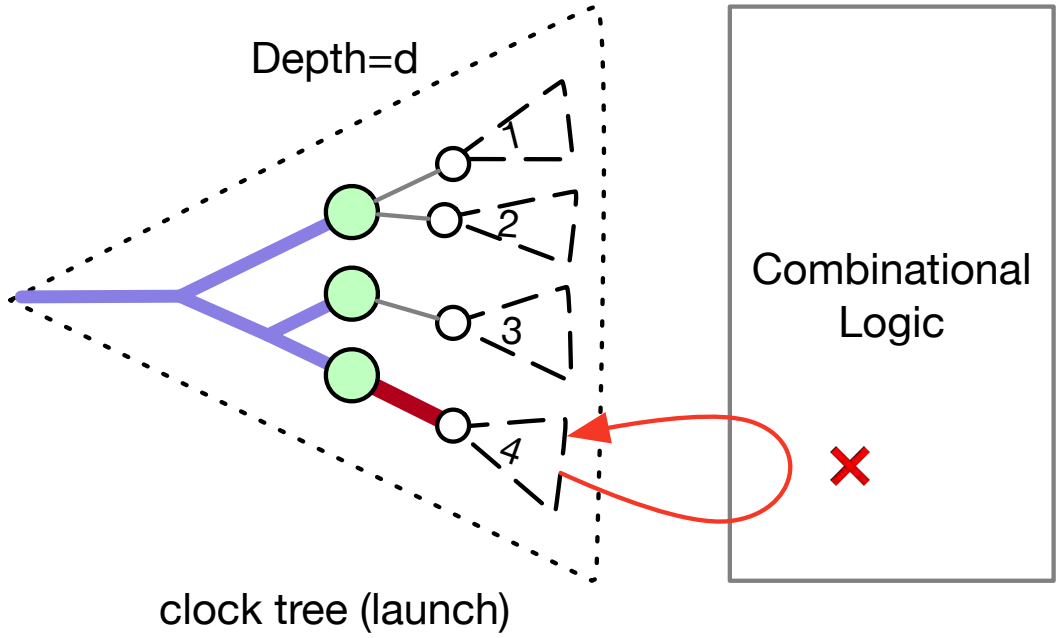
- #FFs > 10,000. FF enumeration is very slow, but clock tree depth *D* < 100.
- Solution: identify FF pairs with **lowest common ancestors (LCA) of depth *d=0,1,…,D-1***, and process these FF pairs in a single propagation.
- The clock edges above the fixed depth *d* are recognized as common paths.
- Computation repeated for different *d*, instead of different launching FFs/capturing FFs. => 1000x fewer iters!





CPPR with Depth Enumeration [Guo, DAC'21]

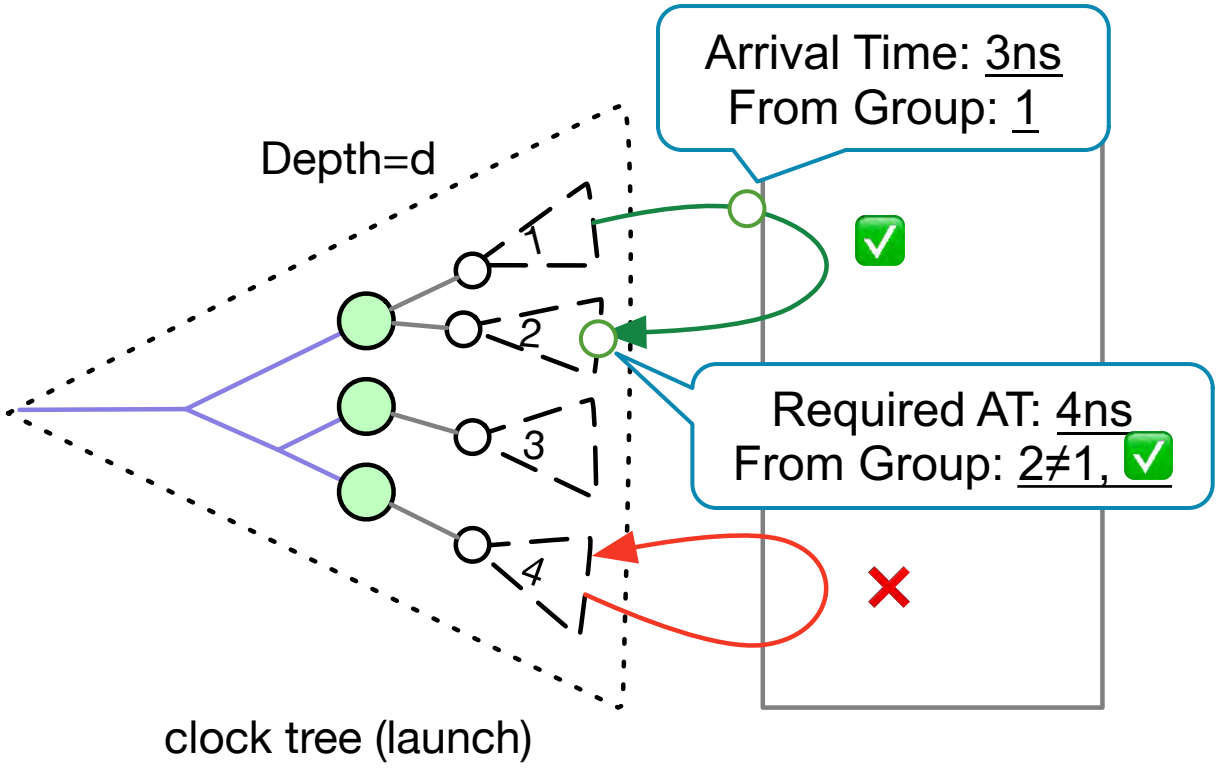
- A depth *d* fixed. How to find all FF pairs with such LCA depth?
- Solution: Group the FFs by their depth-d+1 ancestors.
- If launching FF and capturing FF come from the same group, they have unhandled pessimism (red tree edge below)



Group Constraint:
Launching FF and capturing FF must come from different groups.

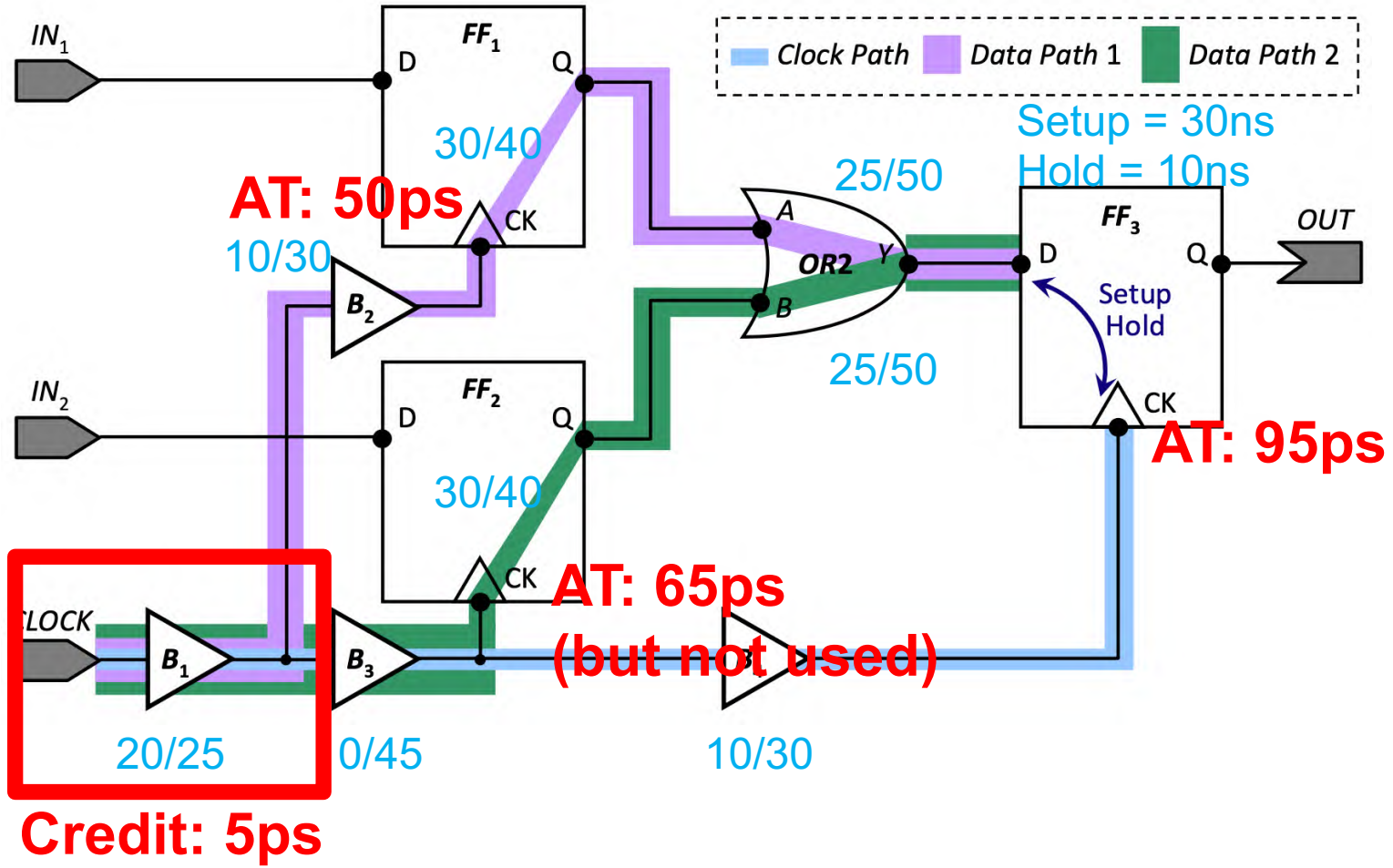
CPPR with Depth Enumeration [Guo, DAC'21]

- How to enforce the group constraint when we propagate the arrival times?
- Solution: propagate the arrival time with a tag on its launching FF's group index.



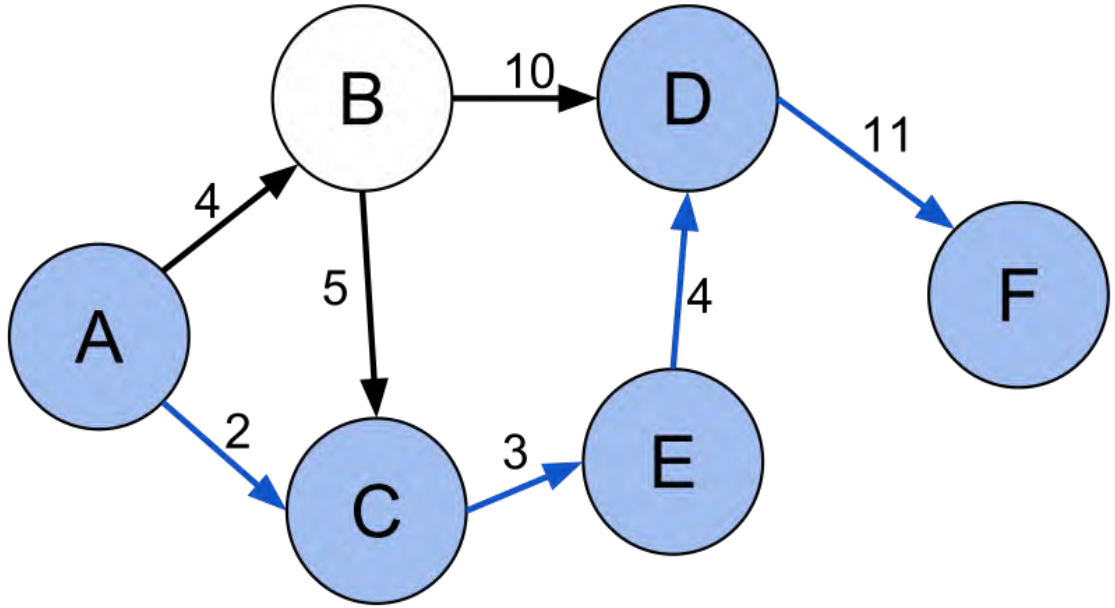
CPPR with Depth Enumeration [Guo, DAC'21]

- Example
- Fix depth *d=1*
  - Only B1 on common path.
  - Arrival Times:
    FF1.CK: 50ps, FF2.CK: 65ps,
    FF3.CK:
- Groups
  - Group 1: {FF1}
  - Group 2: {FF2, FF3}
- Data path 1 valid,
  path 2 invalid (same group)



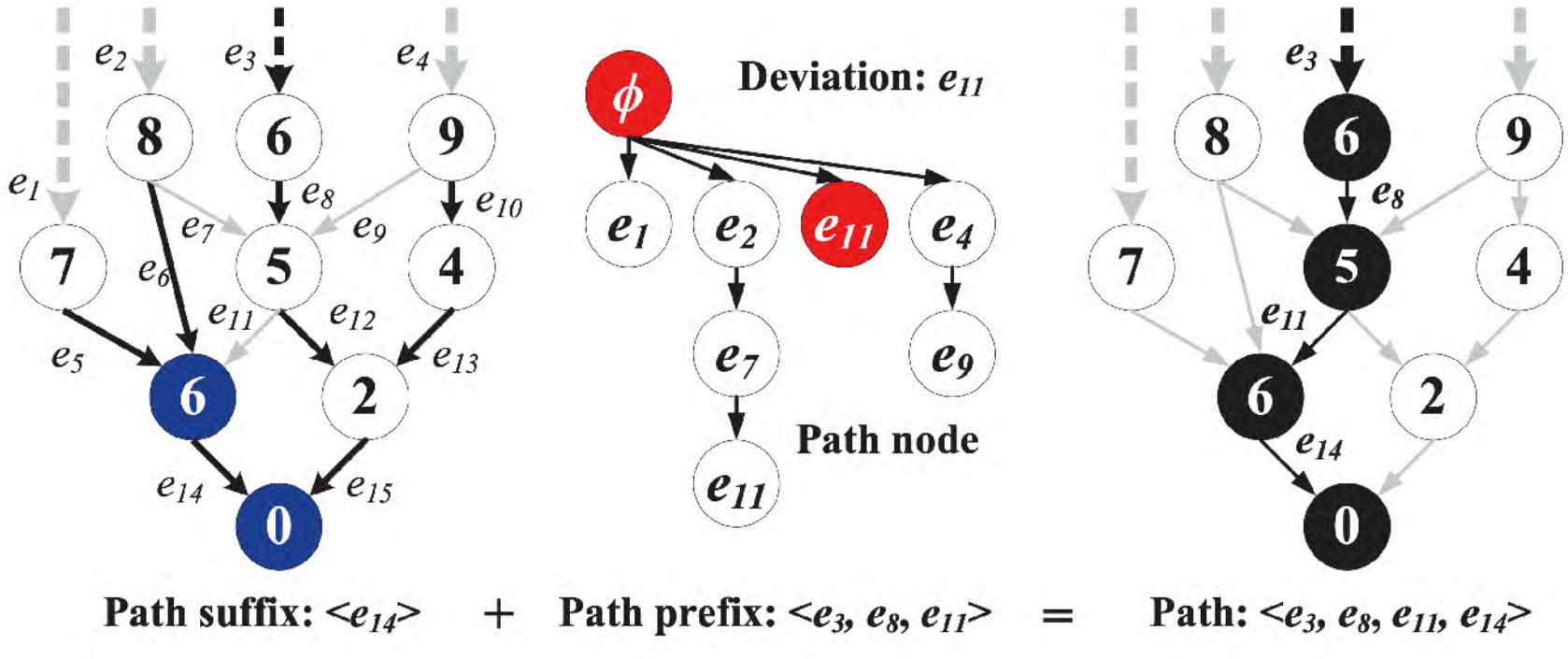
Generate Top-1 Critical Path

- Working on DAG
  - Setup check: longest path
  - Hold check: shortest path



Generate Top-K Critical Paths

- OpenTimer adopts implicit path representation
  - Each path is represented using $O(1)$ space and time
  - Each path is ranked through a *prefix* tree & a *suffix* tree



Huang, Tsung-Wei, et al. "OpenTimer v2: A new parallel incremental timing analysis engine." *IEEE TCAD* 2020.

Summary

- Net delay
  - Elmore delay
- Cell delay
  - Lookup table
- Graph-based analysis
  - Setup time, hold time
  - Arrival time, required arrival time, slack
- Path-based analysis
  - CPPR

Other Topics

- Incremental timing analysis (TAU 2015 contest)
- Timing macros (TAU 2016-2017 contest)
- Timing reports (TAU 2018 contest)
- Timing-driven design optimization (TAU 2019 contest)
- Current source delay models (TAU 2020 contest)