IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Lecture 38: TIMING-DRIVEN ROUTING

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Timing-Driven Routing

- In modern VLSI chips, interconnects contribute significantly to total signal delay.
  - Interconnect delay is a major concern during routing.
- Timing-driven routing seeks to minimize one or both of:
  - ***Maximum sink delay***, which is the maximum interconnect delay from the source node to any sink of a given net.
  - ***Total wirelength***, which affects the load-dependent delay of the net's driving gate.

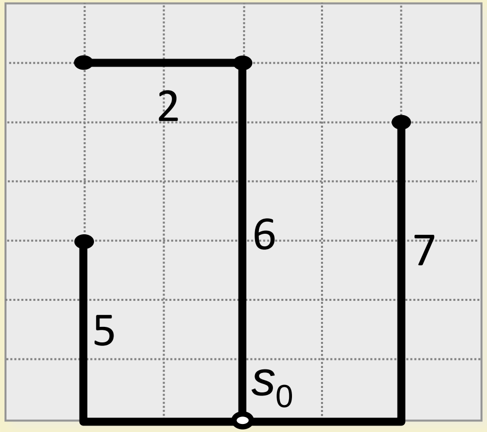
Some Notations

- For a given signal net *net*, let
  - s_0 be the source node
  - $sinks = \{s_1, \ldots, s_n\}$ be the sinks
  - $G = (V,E)$ be a weighted graph where $V = \{v_0, v_1, \ldots, v_n\}$ represents the source and sink nodes of *net*, and the weight of an edge $e(v_i,v_j) \in E$ represents the routing cost between v_i and v_j .
  - For any spanning tree T over G, let *radius(T)* be the length of the longest source-sink path in T, and let *cost(T)* be the total edge weight of T.

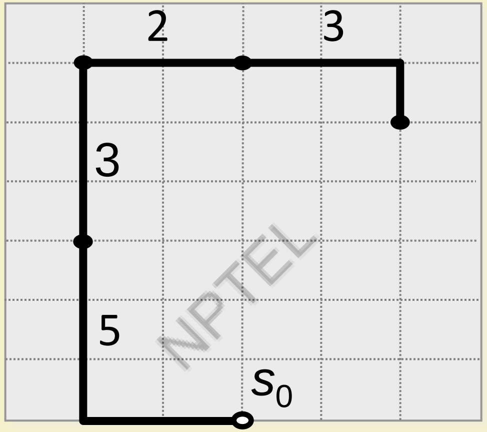
An Observation

- Because source-sink wirelength reflects source-sink signal delay, a routing tree ideally minimizes both radius and cost.
- However, for most signal nets, radius and cost cannot be minimized at the same time.
- The next slide illustrates this radius ("*shallow*") versus cost ("*light*") tradeoff, where the labels represent edge costs.
  - The first tree has *minimum radius* :: a shortest-path tree that can be constructed using the *Dijkstra's algorithm*.
  - The second tree has *minimum cost* :: a minimum spanning tree that can be constructed using *Prim's algorithm*.

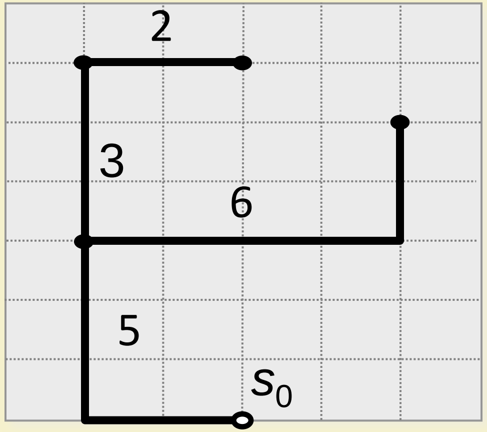
- Due to their large cost and large radius, neither of these trees may be desirable in practice.
- The third tree as shown is a *compromise* that has both shallow and light properties.



$radius(T)$ = 8
$cost(T)$ = 20
"Shallow"



$radius(T)$ = 13
$cost(T)$ = 13
"Light"



$radius(T)$ = 11
$cost(T)$ = 16
Tradeoff between shallow and light

The Bounded-Radius Bounded-Cost Algorithm

- The BRBC algorithm finds a shallow-light spanning tree with provable bounds on both radius and cost.
  - Each of these parameters is within a constant factor of its optimum value.
- From the graph *$G=(V,E)$*, the algorithm first constructs a subgraph *G'* that contains all $v \in V$, and has small cost and small radius.
  - The shortest-path tree T_{BRBC} in G' will also have small radius and cost because it is a subgraph of G'.
  - T_{BRBC} is determined by a parameter $\varepsilon \geq 0$, which trades off between radius and cost.
  - When $\varepsilon = 0$, T_{BRBC} has *minimum radius*; when $\varepsilon = \infty$, T_{BRBC} has *minimum cost*.

- T_{BRBC} satisfies both the following inequalities:

$$radius\ (T_{BRBC}) \leq (1+\varepsilon) \cdot radius\ (T_S)$$

$$cost\ (T_{BRBC}) \leq \left(1+\frac{2}{\varepsilon}\right) \cdot cost\ (T_M)$$

where T_S is a shortest-path tree of G, and T_M is a minimum spanning tree of G.

Prim-Dijkstra Tradeoff

- This method trades off radius and cost by using an explicit, quantitative metric.
- The minimum cost and minimum radius objectives are typically optimized using Prim's algorithm and Dijkstra's algorithm.
  - Though they target different objectives, these algorithms construct spanning trees over the set of terminals in very similar ways.
  - From the set of sinks S, each algorithm begins with tree T consisting only of s_0, and iteratively adds a sink s_j and the edge connecting a sink s_i in T to s_j.
  - The algorithms differ only in the cost function by which the next sink and edge are chosen.

- In Prim's algorithm, sink s_j and edge $e(s_i,s_j)$ are selected to minimize the edge cost between sinks s_i and s_j, where $s_i \in T$ and $s_j \in S - T$.

$$cost(s_i, s_j)$$

- In Dijkstra's algorithm, sink s_j and edge $e(s_i,s_j)$ are selected to minimize the path cost between source s_0 and sink s_j, where $s_i \in T$ and $s_j \in S - T$.

$$cost(s_0, s_i) + cost(s_i, s_j)$$

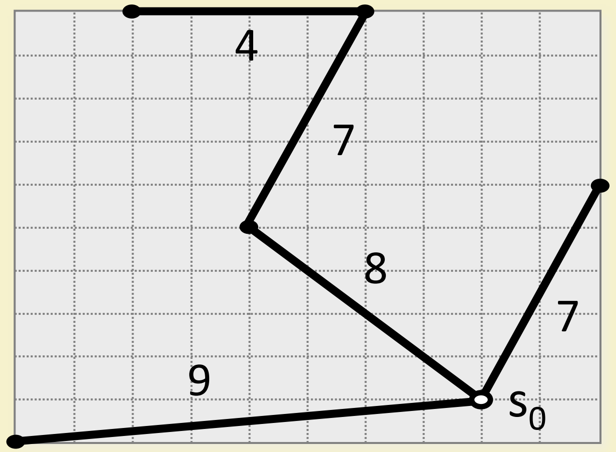
- To combine the two objectives, ***PD Tradeoff*** iteratively adds the sink s_j and the edge $e(s_i,s_j)$ to T such that

$$\gamma \cdot cost(s_0, s_i) + cost(s_i, s_j)$$

is minimum over all $s_i \in T$ and $s_j \in S - T$ for a prescribed constant $0 \le \gamma \le 1$.

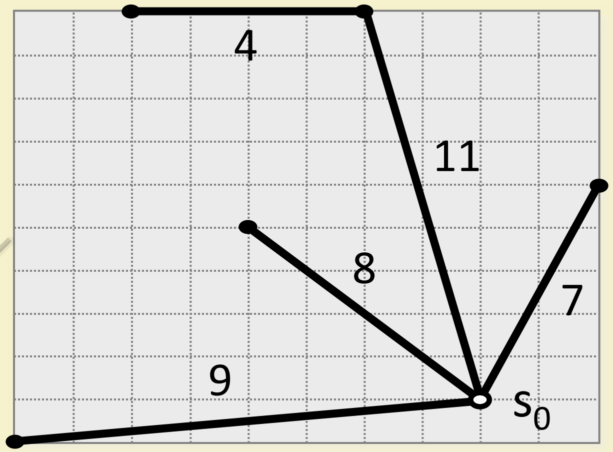
The Tradeoff

- When $\gamma = 0$, the PD tradeoff is identical to Prim's algorithm, and T is a minimum spanning tree T_M.
- As γ *increases*, the PD tradeoff constructs spanning trees with progressively higher cost but lower radius.
- When $\gamma = 1$, the PD tradeoff is identical to Dijkstra's algorithm, and T is the shortest-paths tree T_S.



radius(*T*) = 19
cost(*T*) = 35

γ = 0.25



radius(*T*) = 15
cost(*T*) = 39

γ = 0.75

Minimization of Source to Sink Delay

- For a multi-pin net, the sink with the least timing slack is called the ***critical sink*** of the net.
  - A routing tree construction that does not consider critical-sink information may create avoidable negative slack, and degrade the overall timing performance of the design.
  - Several routing tree construction approaches exist that address the critical-sink routing problem.
- We iteratively form a tree by adding sinks, and optimizes the critical sinks.

Critical-sink Routing Tree (CSRT) Problem

- Given a signal net with source s_0, sinks $S = \{s_1, s_2, \ldots, s_n\}$, and sink criticalities $\alpha(i) \geq 0$, for each $s_i \in S$ construct a routing tree T such that

$$\sum_{i=1}^{n} \alpha(i) \cdot t(s_0, s_i)$$

is minimized, where $t(s_0, s_i)$ is the signal delay from source s_0 to sink s_i, and the sink criticality $\alpha(i)$ reflects the timing criticality of the sink s_i.

- If a sink is on the critical path, then its timing criticality will be greater than that of other sinks.

Critical-sink Steiner Tree Heuristic

- This heuristic first constructs a heuristic minimum-cost Steiner tree T_0 over all terminals of S except the critical sink s_c.
- Then, to reduce $t(s_0, s_c)$, the heuristic adds s_c into T_0 by heuristic variants:
  - **H_0**: introduce a single wire from s_c to s_0.
  - **H_1**: introduce the shortest possible wire that can join s_c to T_0, so long as the path from s_0 to s_c is monotone (i.e. shortest possible total length).
  - **H_{best}**: try all shortest connections from s_c to edges in T_0 as well as from s_c to s_0. Perform timing analysis on each of these trees and return the one with the lowest delay at s_c.

RAT Tree Problem

- For a signal net with source s_0 and sink set S, find a minimum-cost routing tree T such that

  $min\ (RAT(s) - t(s_0,s)) \geq 0$, for all $s \in S$

  where $RAT(s)$ is the required arrival time for sink s, and $t(s_0,s)$ is the signal delay in T from source s_0 to sink s.

END OF LECTURE 38

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Lecture 39: PHYSICAL SYNTHESIS (PART 1)

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Physical Synthesis

- Physical synthesis is a collection of timing optimizations to fix negative slack.
  - Consists of creating timing budgets and performing timing corrections.
- Timing budgets include:
  - Allocating target delays along paths or nets.
  - Often done during placement and routing stages.
  - Can also be done during timing correction operations.

- Timing corrections can be done using various techniques:
    a) Gate sizing
    b) Buffering
    c) Netlist restructuring

(a) Gate Sizing

- Modern-day VLSI chip design is predominantly based on the standard-cell based design style.
  - Each logic gate (e.g. NAND or NOR) is typically available in multiple sizes that correspond to different drive strengths.
  - The drive strength is the amount of current that the gate can provide during switching.
- When a gate drives other gates, the gate output node faces some load capacitance depending on the fanout and sizes of the driven gates.
  - Illustrative example with CMOS gates.

- Suppose that a gate *v* has three versions *A*, *B* and *C* of differing sizes:

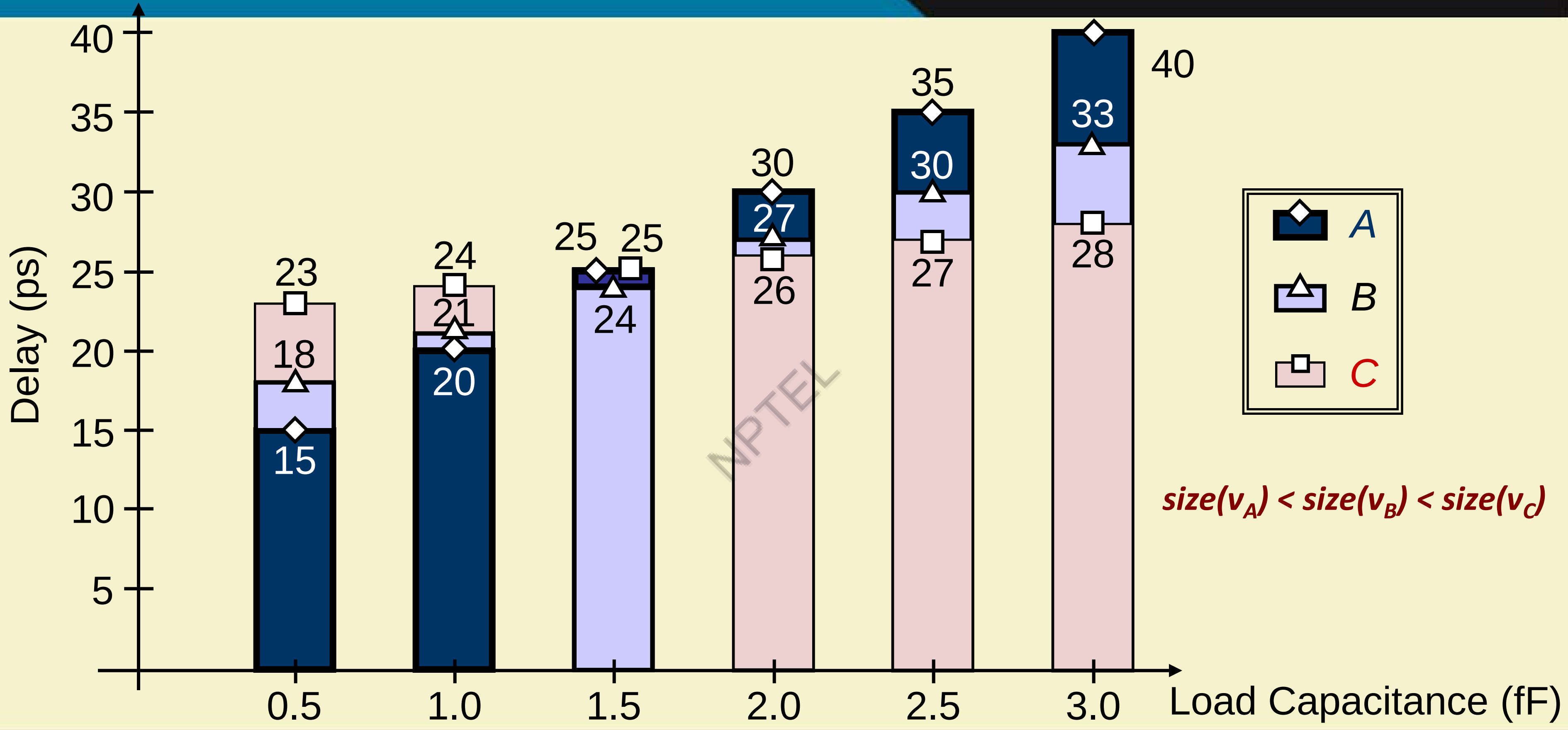
  $size(v_A) < size(v_B) < size(v_C)$
- A gate with larger size will have lower output resistance and can drive a larger load capacitance with smaller load-dependent delay.
- However, a larger gate also has larger intrinsic delay due to the parasitic output capacitance of the gate itself.
  - When the load capacitance is large, the load-dependent delay will dominate:

    $t(v_C) < t(v_B) < t(v_A)$

– When the load capacitance is small, the intrinsic delay will dominate:

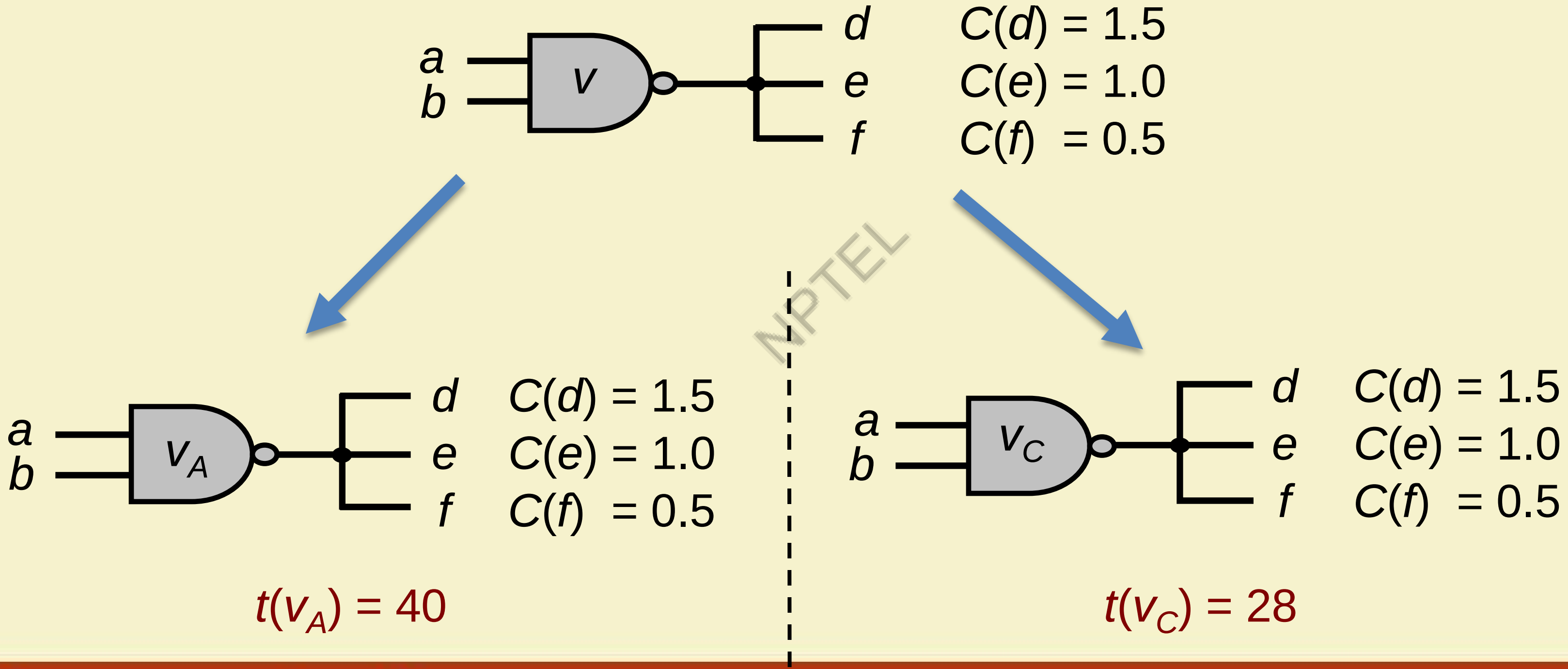
  $t(v_A) < t(v_B) < t(v_C)$

– Increasing *size(v)* also increases the gate capacitance of *v*, which in turn increases the load capacitance seen by the fanin drivers.



size(v_A) < size(v_B) < size(v_C)

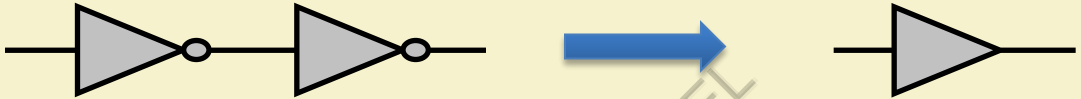
- Resizing transformations adjust the size of a gate v to achieve a lower delay.



- The total load capacitance driven by gate v is $C(d)+C(e)+C(f) = 3\ fF$.
- Gate delays are calculated based on the load-delay relations shown in the graph.
- For a load capacitance of 3 fF, gate delay reduces from 40 ps to 28 ps if v_C is used instead of v_A.

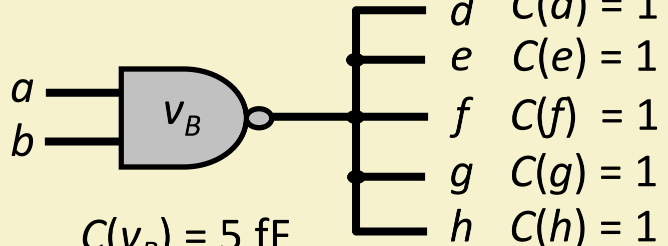
(b) Buffering

- A buffer is a gate, typically two serially-connected inverters, that regenerates a signal without changing the functionality.



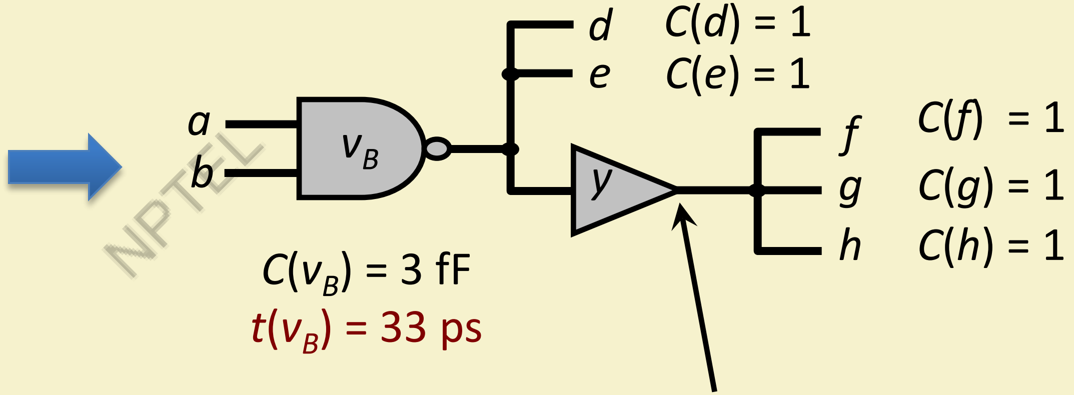
- Buffers can improve delays by:
  - Speeding up the circuit or by serving as delay elements.
  - Modify transition times to improve signal integrity and coupling-induced delay variation.
  - Shielding capacitive load on the output side.

- Drawbacks of using buffers:
  - They consume more area.
  - They increase power consumption.
- In spite of judicious use of buffering by modern EDA tools, the number of buffers has been steadily increasing in large designs due to technology scaling trends.
  - Interconnect delays dominate as compared to gate delays.
- In modern high-performance designs, buffers can comprise 10-20% of all standard cell instances, and more than 40% in some designs.



$C(v_B) = 5$ fF

$t(v_B) = 45$ ps



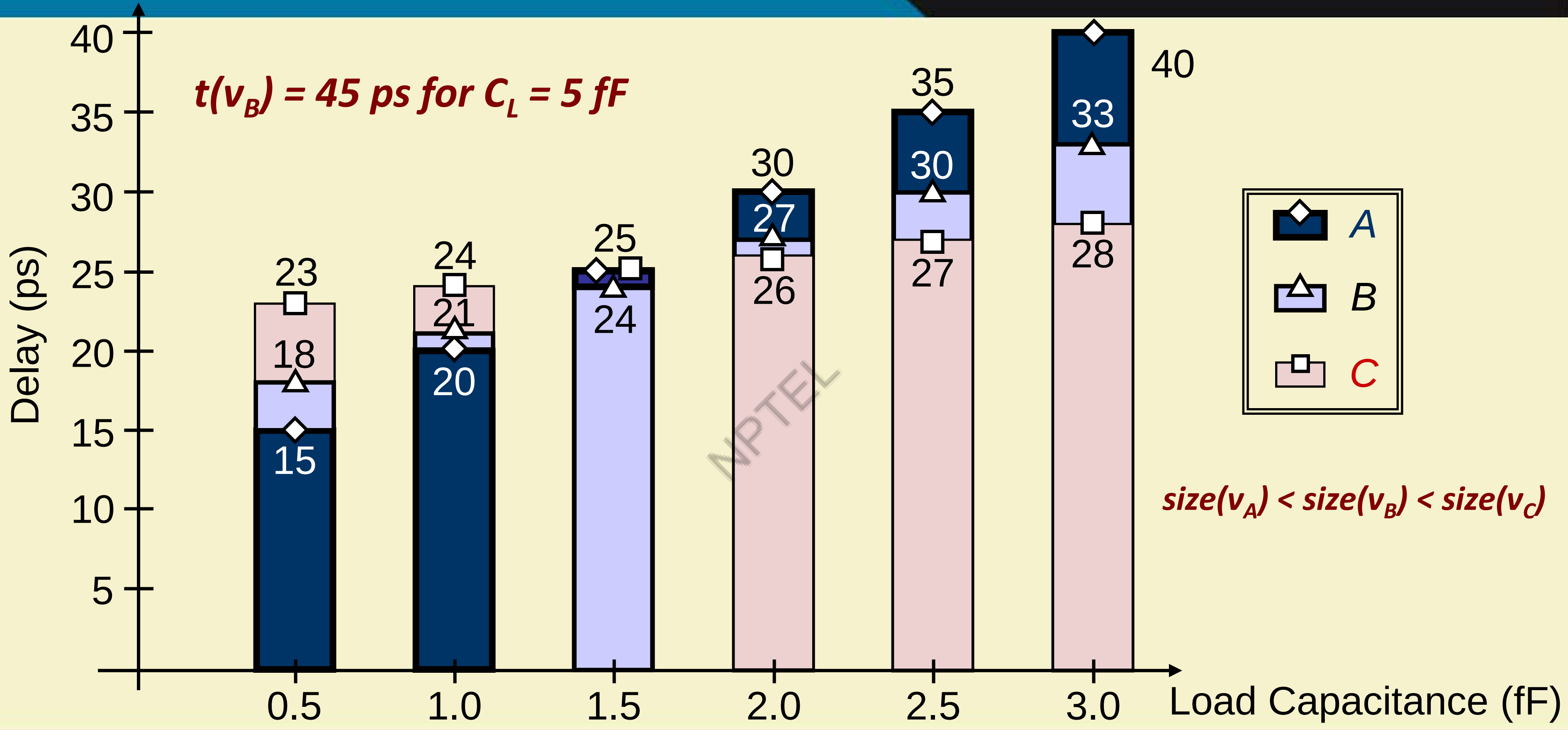
$C(v_B) = 3$ fF

$t(v_B) = 33$ ps

$C(y) = 3$ fF

$t_{tot} = t(v_B) + t(y) = 66$ ps

$t(v_B) = 45$ ps for $C_L = 5$ fF



$size(v_A) < size(v_B) < size(v_C)$

- In the example, the (actual) arrival time at fanout pins d-h for gate v_B is $t(v_B) = 45\ ps$.
  - Let pins d and e be on the critical path with RAT's below $35\ ps$.
  - Let the input pin capacitance of buffer y be $1\ fF$.
- Adding y reduces the load capacitance of v_B from 5 to 3, and reduces the arrival times at d and e to $t(v_B) = 33\ ps$.
- After y is inserted, the arrival times at pins f, g and h becomes

  $t(v_B) + t(y) = 33 + 33 = 66\ ps$.

END OF LECTURE 39

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Lecture 40: PHYSICAL SYNTHESIS (PART 2)

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

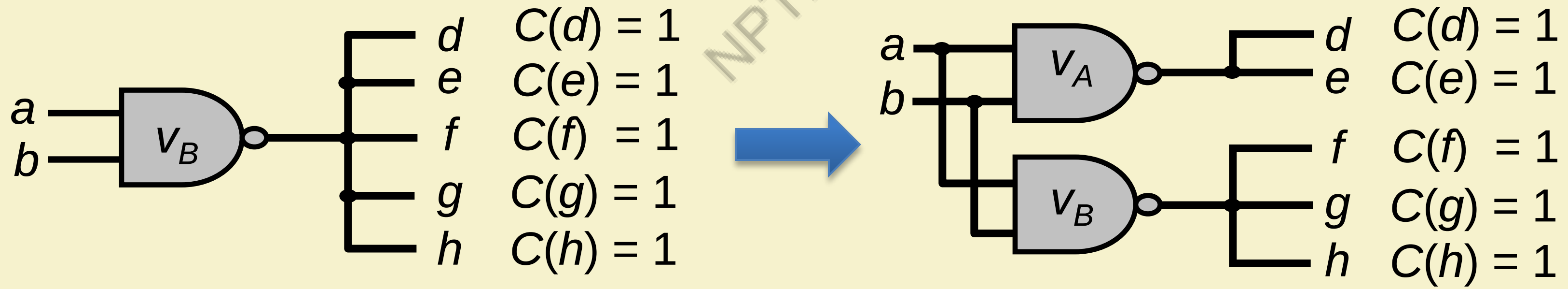
(c) Netlist Restructuring

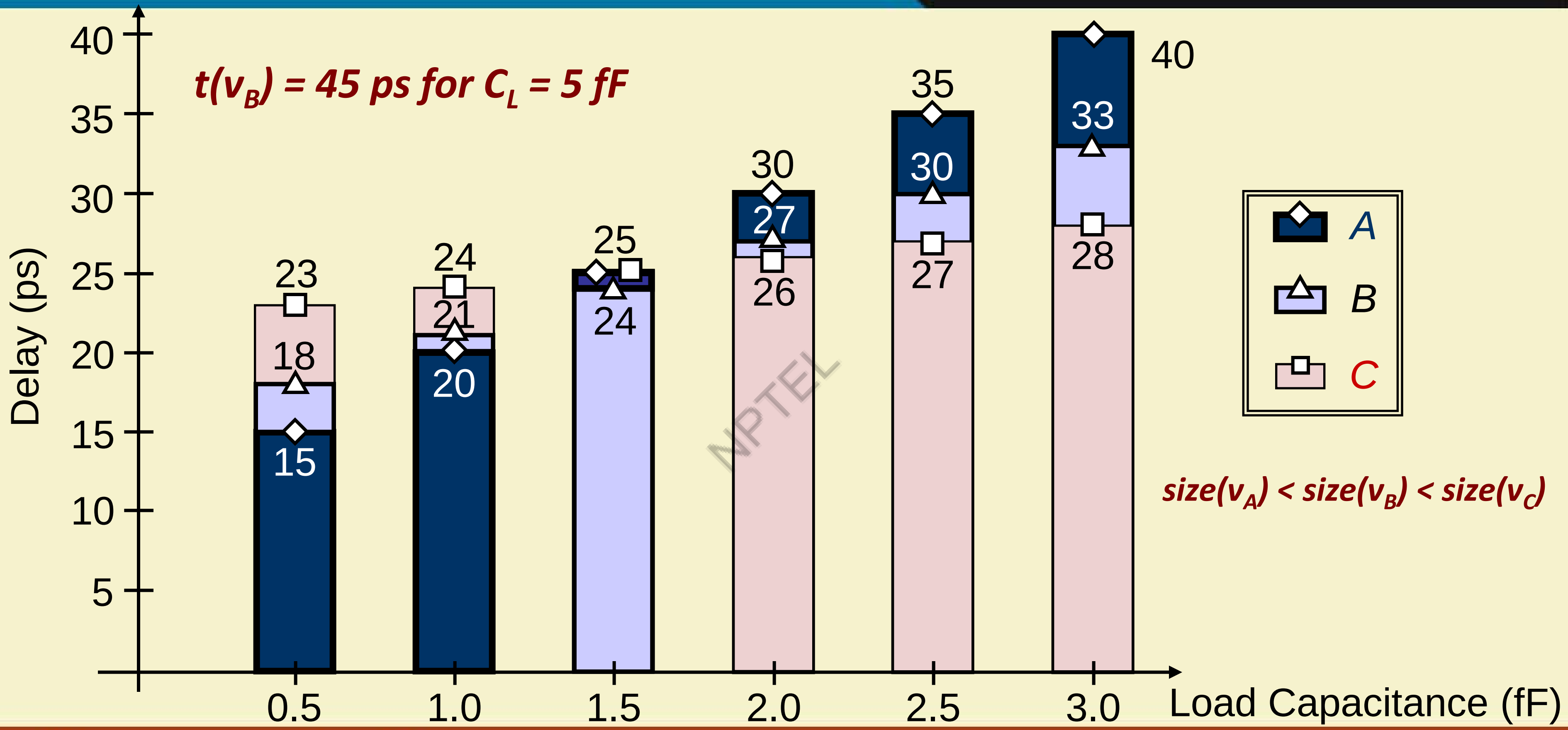
- Here we modify the netlist itself to improve timing.
  - Such changes must not alter the circuit functionality, but can use additional gates or modify (rewire) the connections between existing gates to improve driving strength and signal integrity.
- Some common netlist modification techniques include:
  - *Cloning*: duplicate gates
  - *Redesign fanin or fanout tree*: changing the topology of gates
  - *Swapping communicative pins*: changing the connections
  - *Gate decomposition*: e.g. changing AND-OR to NAND-NAND
  - *Boolean restructuring*: applying Boolean algebra rules to change circuit gates

Cloning

- Gate duplication can reduce delay in two situations:
  - When a gate with significant fanout may be slow due to its fanout capacitance.
  - When a gate's output fans out in two different directions, making it impossible to find a good placement for this gate.
- The effect of cloning is to split the driven capacitance between two equivalent gates, at the cost of increasing the fanout of upstream gates.
- The second application of cloning allows the designers to replicate gates and place each clone closer to its downstream logic.

- Using the same load-delay relation as shown earlier, the initial gate delay *$t(v_B) = 45$ ps*.
- After cloning, *$t(v_A) = 30$ ps* and *$t(v_B) = 33$ ps*.

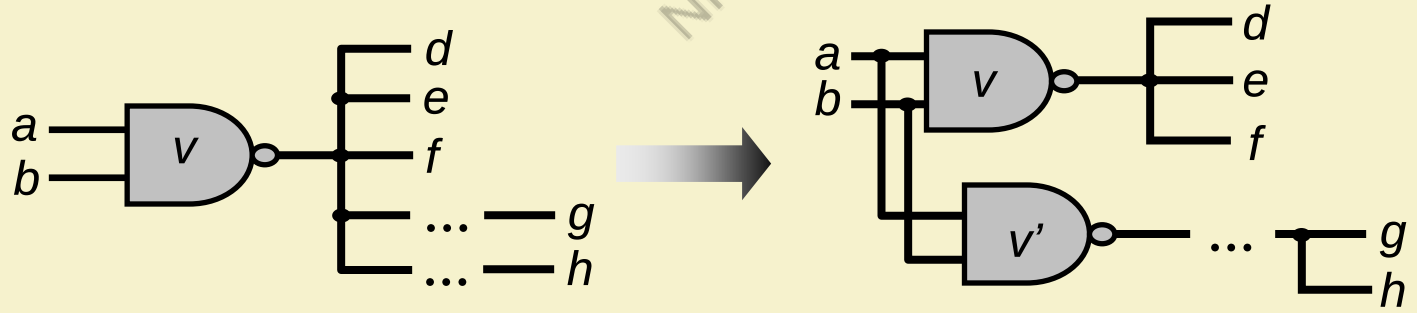




$t(v_B)$ = 45 ps for C_L = 5 fF
Delay (ps)
40
35
30
25
20
15
10
5
0.5
1.0
1.5
2.0
2.5
3.0
Load Capacitance (fF)
23
18
15
24
21
20
25
24
30
27
26
35
30
27
40
33
28
A
B
C
NPTEL
$size(v_A) < size(v_B) < size(v_C)$

- When the downstream capacitance is large, buffering may be a better alternative than cloning because buffers do not increase the fanout capacitance of upstream gates.
- However, buffering cannot replace placement-driven cloning.

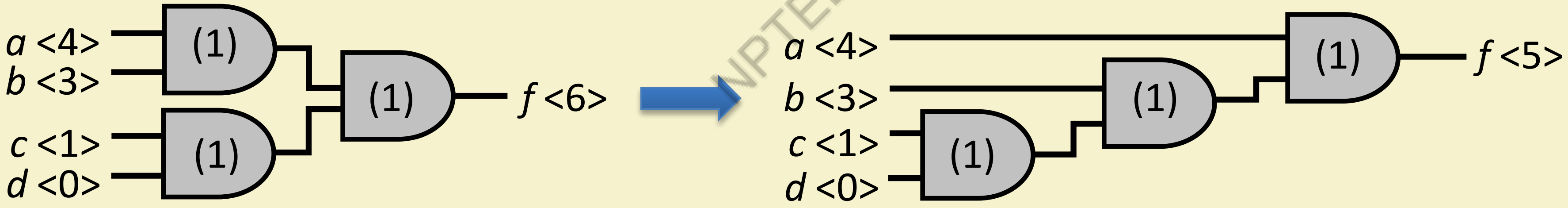
- The gate *v* drives five signals *d-h*, where signals *d-f* are close, and *g* & *h* are located much farther away.
- To mitigate the large interconnect delay, the gate is cloned and a new copy *v'* is placed closer to *g* & *h*.



Redesign of Fanin Tree

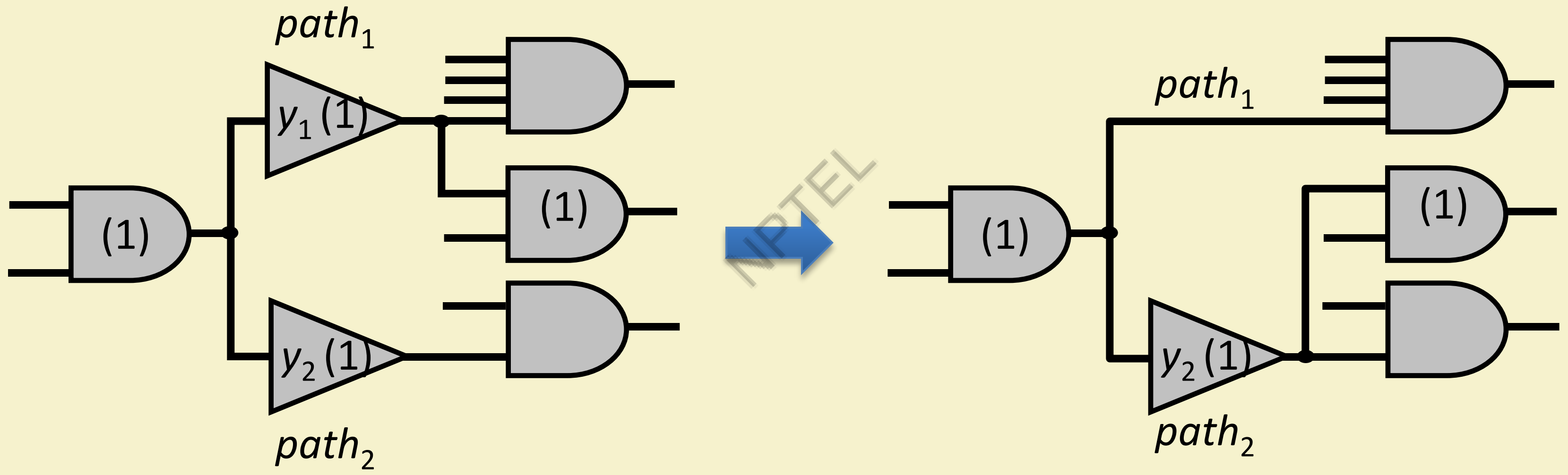
- Logic design often provides a netlist with the minimum number of logic levels.
- Minimizing the maximum number of gates on a path between sequential elements tends to produce a balanced circuit with similar path delays from inputs to outputs.
  - However, input signals may arrive at varied times, so the level minimized circuit may not be timing-optimal.

- The arrival time *AAT(f) = 6* in the original circuit.
- The modified unbalanced network has a shorter input-output path for the latest arriving signal.



Redesign of Fanout Tree

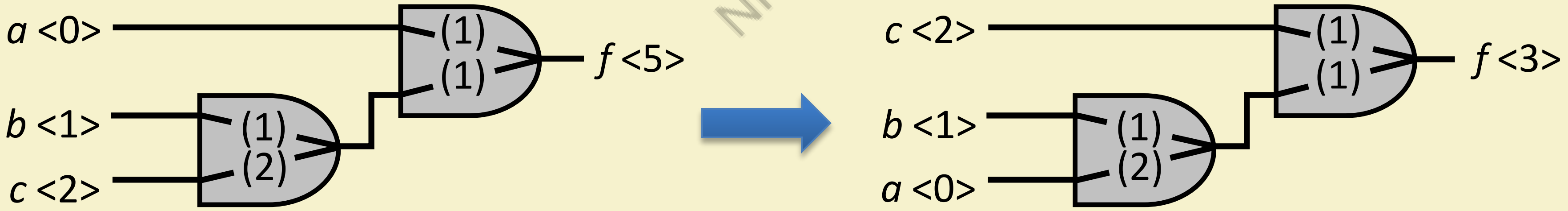
- It is possible to improve timing by rebalancing the output load capacitance in a fanout tree so as to reduce the delay of the longest path.
- In the example shown on the next slide, the buffer *y_1* is needed because the load capacitance of critical path *$path_1$* is large.
- By redesigning the fanout tree to reduce the load capacitance of *$path_1$*, use of buffer *y_1* can be avoided.
  - Increased delay on *$path_2$* may be acceptable if that path is not critical even after the load capacitance of buffer *y_2* is increased.


$path_1$
y_1 (1)
(1)
(1)
y_2 (1)
$path_2$
$path_1$
(1)
(1)
y_2 (1)
$path_2$

Swapping Commutative Pins

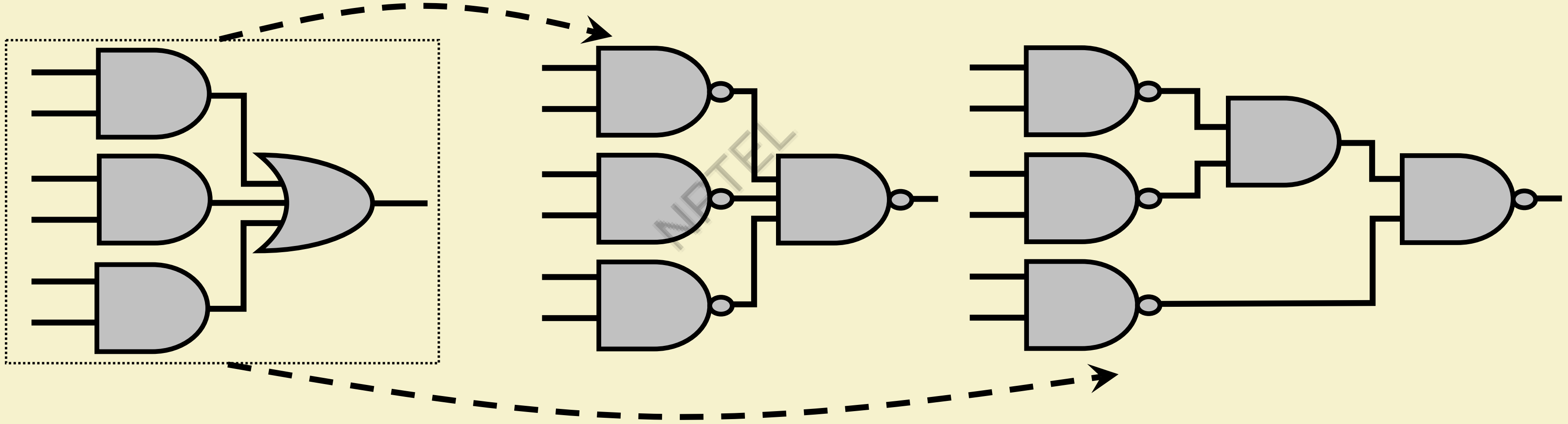
- Although the input pins of some gates (e.g. a NAND gate) are logically equivalent, in the actual transistor-level netlist they can have different delays to the output pin.
  - Path delays can change when the input pin assignment is changed.
- Rule-of-thumb:
  - Assign a later (sooner) arriving signal to an equivalent input pin with shorter (longer) input-output delay.

- The internal timing arcs are labeled with corresponding delays in parentheses, and the pins are labeled with corresponding arrival times in angular brackets.
  - The arrival time at *f* can be improved from *5* to *3* by swapping pins *a* and *c*.



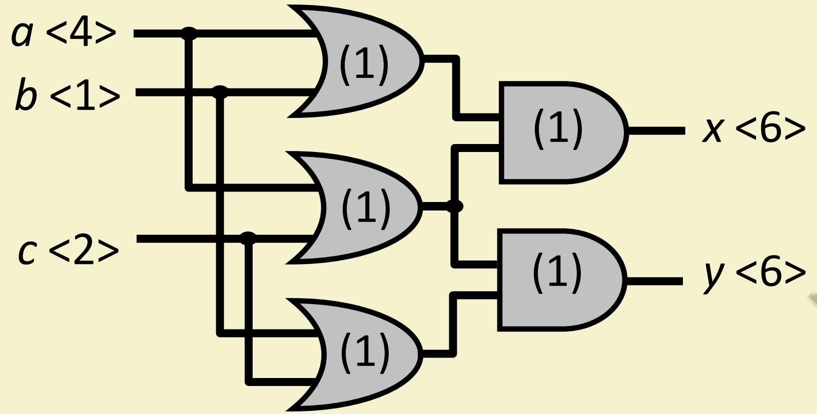
Gate Decomposition

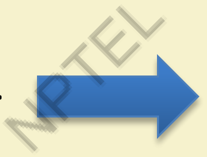
- In CMOS, a gate with multiple inputs usually has larger size and capacitance, as well as a more complex transistor-level network.
- Decomposition of multiple-input gates into smaller, more efficient gates can decrease delay and capacitance.
- An example is shown on the next slide, where two equivalent NAND-level networks are shown.

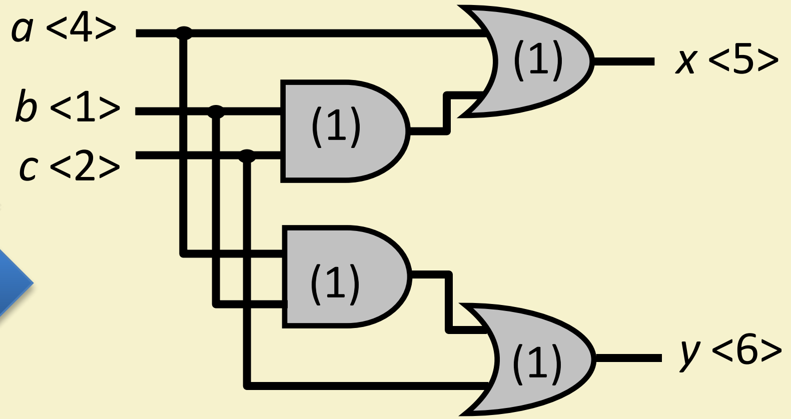


Boolean Restructuring

- Rules of Boolean algebra can be used to implement a function in various ways.
- An example is shown on the next slide.
  - The distributive law $f(a,b,c) = (a + b)(a + c) = a + bc$ is exploited to improve timing.
  - The two functions $x = a + bc$ and $y = ab + c$ are realized with signal arrival times $AAT(a) = 4$, $AAT(b) = 1$, and $AAT(c) = 2$.
  - When implemented with a common node $a + c$, we get $AAT(x) = AAT(y) = 6$.
  - Implementing x and y separately achieves $AAT(x) = 5$ and $AAT(y) = 6$.







$x(a,b,c) = (a + b)(a + c)$
$y(a,b,c) = (a + c)(b + c)$

$x(a,b,c) = a + bc$
$y(a,b,c) = ab + c$

END OF LECTURE 40

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Lecture 41: PERFORMANCE-DRIVEN DESIGN FLOW

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

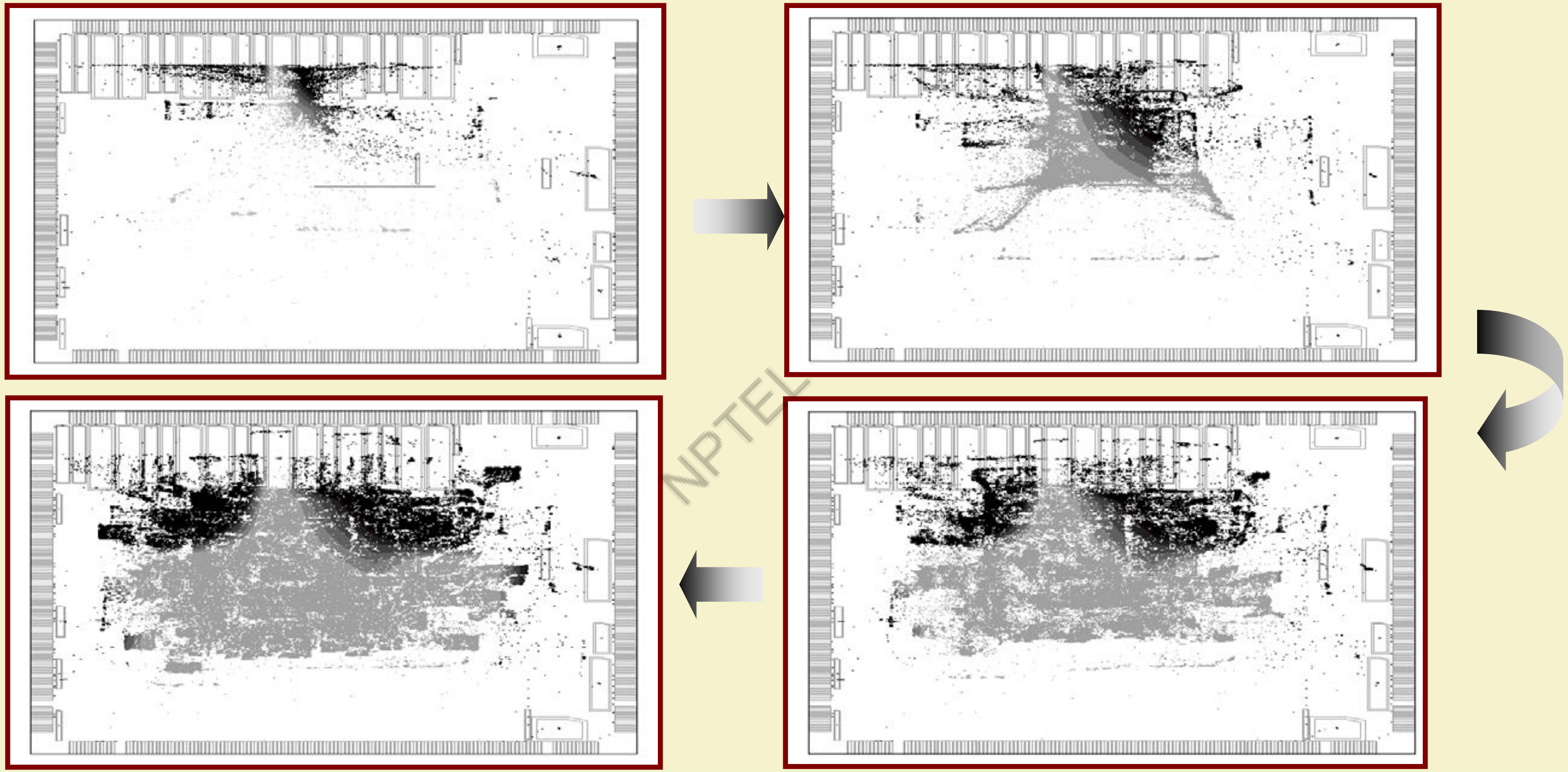
Introduction

- We have discussed several techniques to improve timing of digital circuits.
- All these optimization techniques can be combined in a consistent ***performance-driven physical design flow***.
- Due to the nature of performance optimizations, their ordering is important.
  - Evaluation steps (like STA) must be invoked several times.
  - Some optimizations like buffering must be redone multiple times to facilitate a more accurate evaluation.

Typical Physical Design Flow

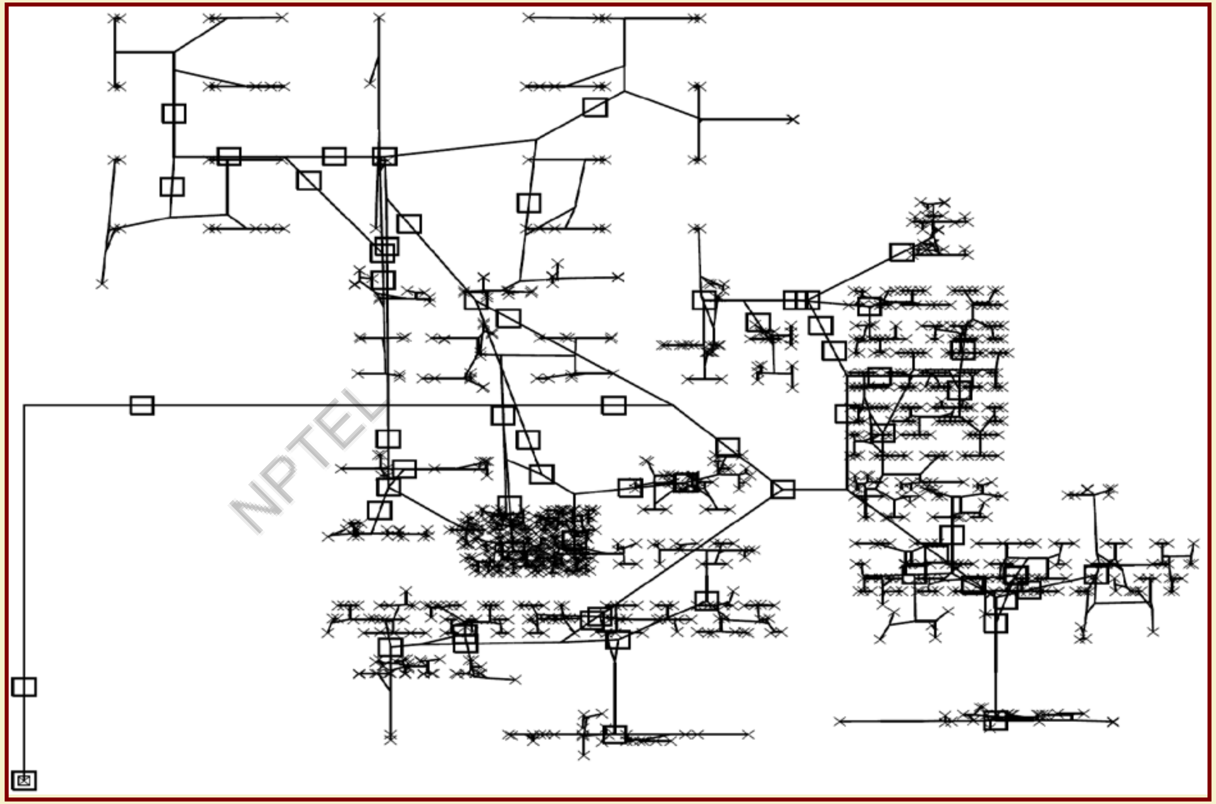
- A typical design flow starts with *chip planning*.
  - Includes *I/O placement*, *floorplanning*, and *power planning*.
- *Trial synthesis* provides the floorplanner with an estimate of the total area needed by modules.
  - Besides logic area, additional space must be provided to account for buffers, routing, and gate sizing.
- Then *logic synthesis* and *technology mapping* produce a gate-level or cell-level netlist from a high-level specification, tailored to a specific technology library.

- Next, *global placement* assigns locations to each movable object.
  - Most of the cells are clustered in highly concentrated regions (colored black).
  - As the iteration proceeds, the cells are gradually spread across the chip, such that they no longer overlap (colored light gray).



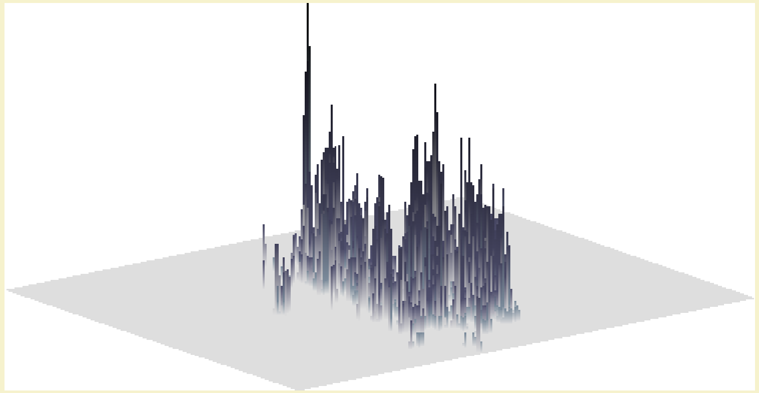
- After global placement, the sequential elements are finalized.
  - Then a *clock network* is generated, either in the form of a clock tree, or in the form of a network consisting of trees and meshes.

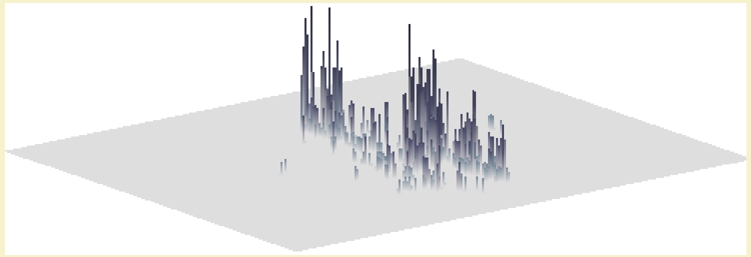
Buffered Clock Tree in a Small CPU Design

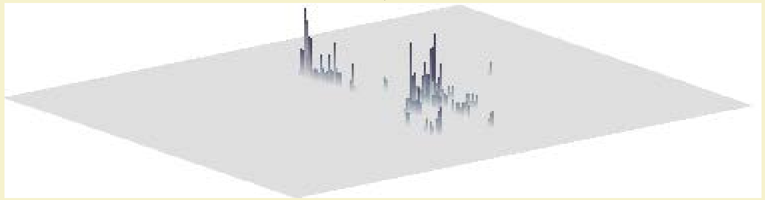


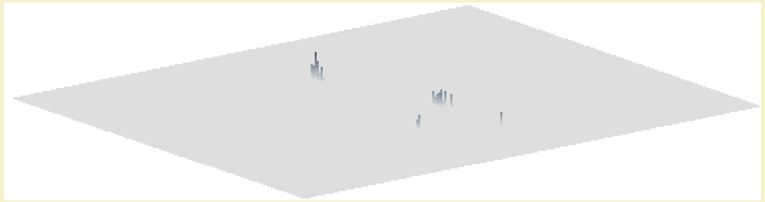
- The locations of globally placed cells are first temporarily rounded to a uniform grid, and then these rounded locations are connected during *global routing* and *layer assignment* (where each route is assigned to a specific metal layer).
- The routes indicate areas of wiring congestion.
  - This information is used to guide *congestion-driven detailed placement*.

Progression of Congestion Maps Through Iterations of Global Routing

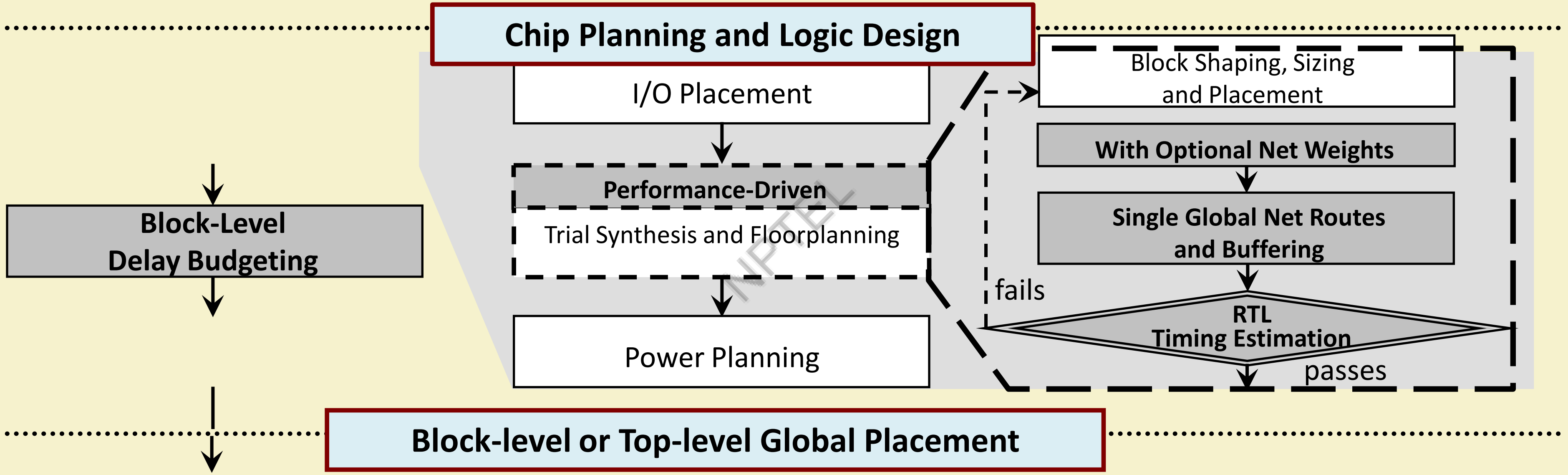


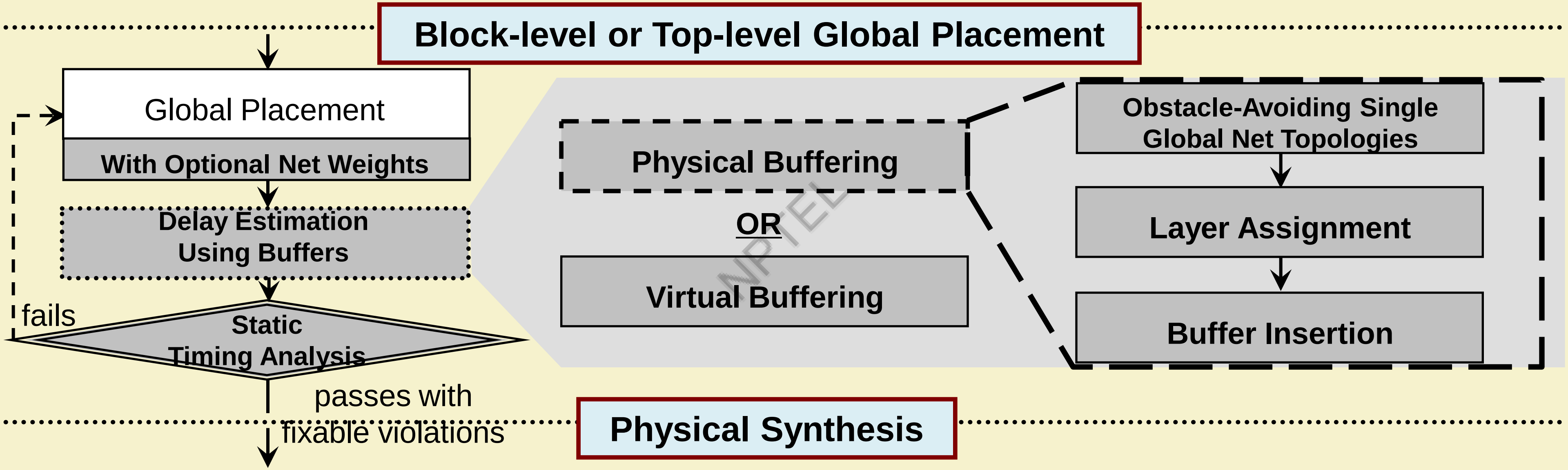


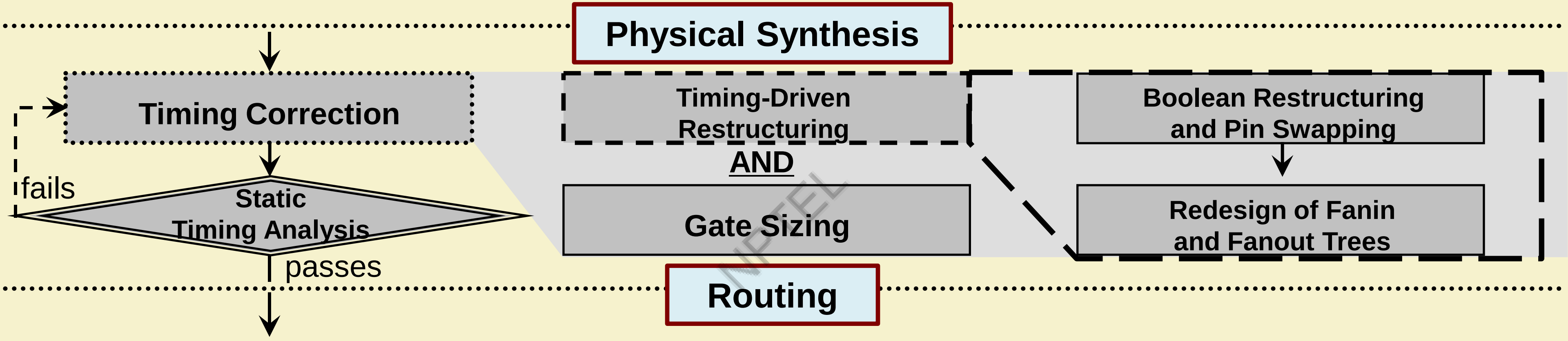


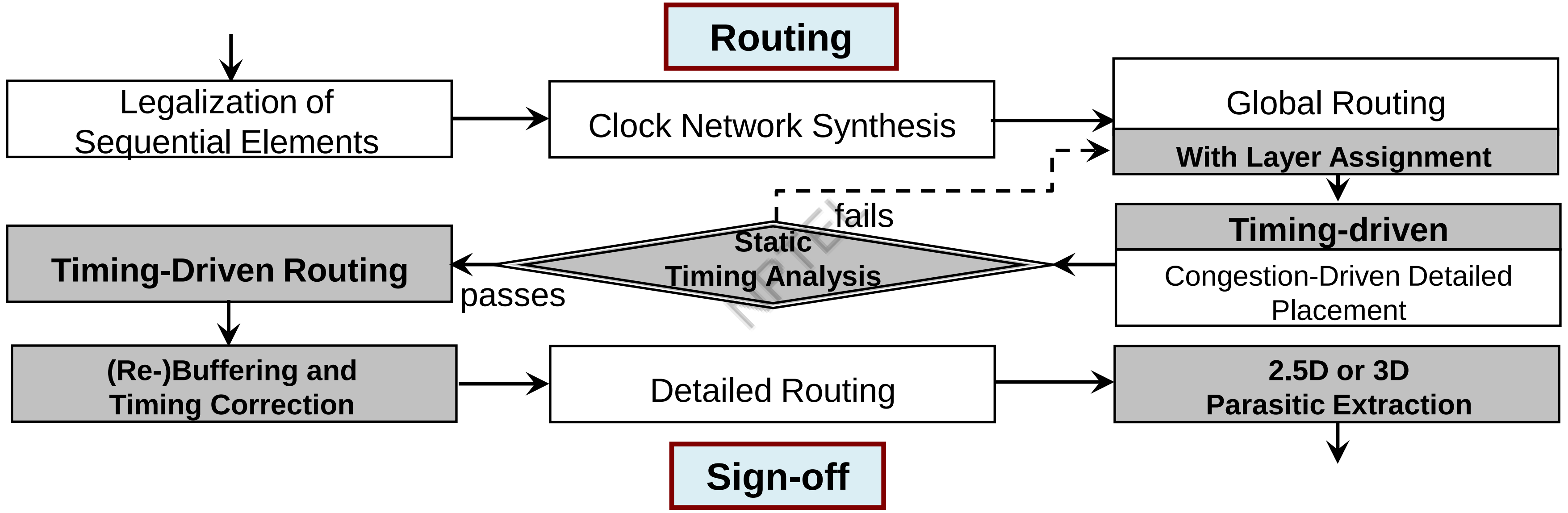


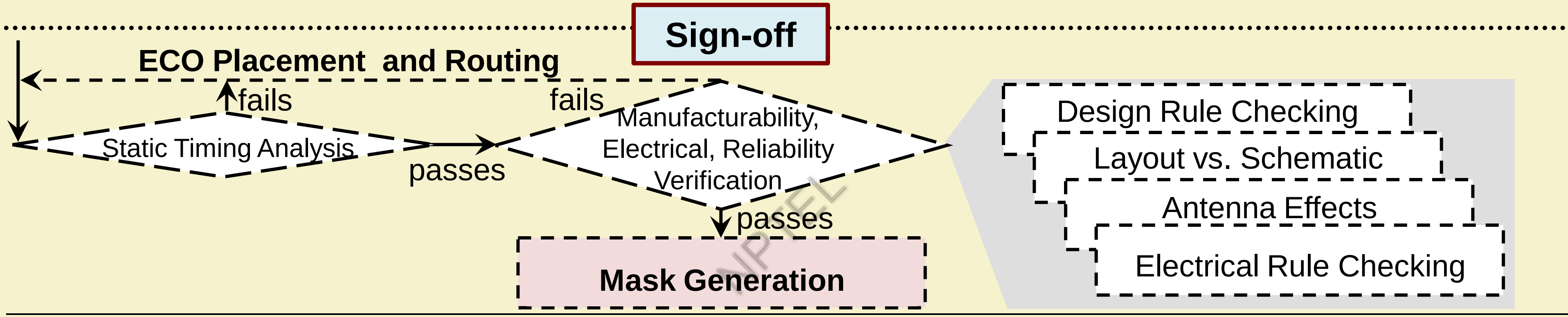
- The global routes of signal nets are then assigned to physical routing tracks during *detailed routing*.
- The layout generated during the place-and-route stage is subjected to *reliability*, *manufacturability*, and *electrical verification*.
- During *mask generation*, each standard cell and each route are represented by collections of rectangles in a format suitable for generating optical lithography masks for chip fabrication.


Chip Planning and Logic Design
I/O Placement
Performance-Driven
Trial Synthesis and Floorplanning
Power Planning
Block Shaping, Sizing and Placement
With Optional Net Weights
Single Global Net Routes and Buffering
fails
RTL Timing Estimation
passes
Block-Level Delay Budgeting
Block-level or Top-level Global Placement


Block-level or Top-level Global Placement
Global Placement
With Optional Net Weights
Delay Estimation Using Buffers
Static Timing Analysis
fails
passes with fixable violations
Physical Buffering
OR
Virtual Buffering
Obstacle-Avoiding Single Global Net Topologies
Layer Assignment
Buffer Insertion
Physical Synthesis


Physical Synthesis
Timing Correction
Static Timing Analysis
fails
passes
Timing-Driven Restructuring
AND
Gate Sizing
Boolean Restructuring and Pin Swapping
Redesign of Fanin and Fanout Trees
Routing


Routing
Legalization of Sequential Elements
Clock Network Synthesis
Global Routing
With Layer Assignment
Timing-driven
Congestion-Driven Detailed Placement
Static Timing Analysis
fails
passes
Timing-Driven Routing
(Re-)Buffering and Timing Correction
Detailed Routing
2.5D or 3D Parasitic Extraction
Sign-off



Engineering Change Order (ECO):: how last minute changes are incorporated in a design.

END OF LECTURE 41

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Lecture 42: MISCELLANEOUS APPROACHES TO TIMING OPTIMIZATION

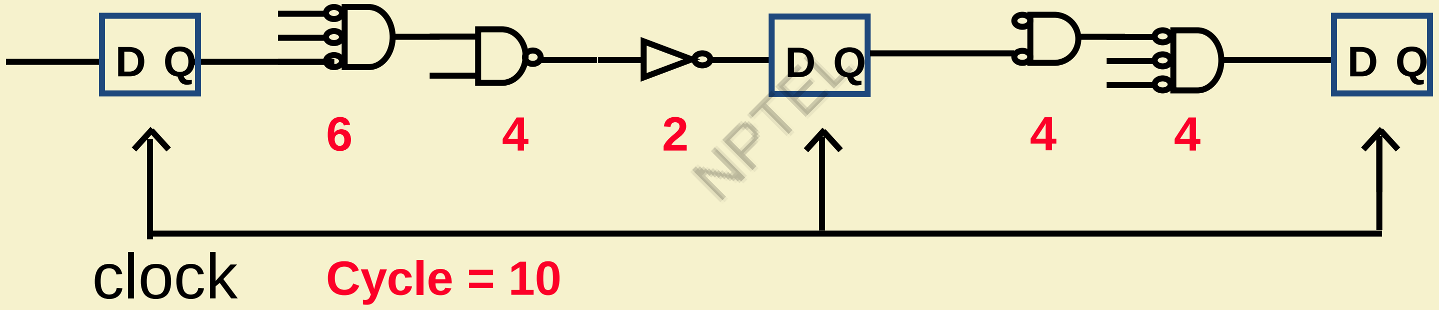
PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Post-Route Optimization Approaches

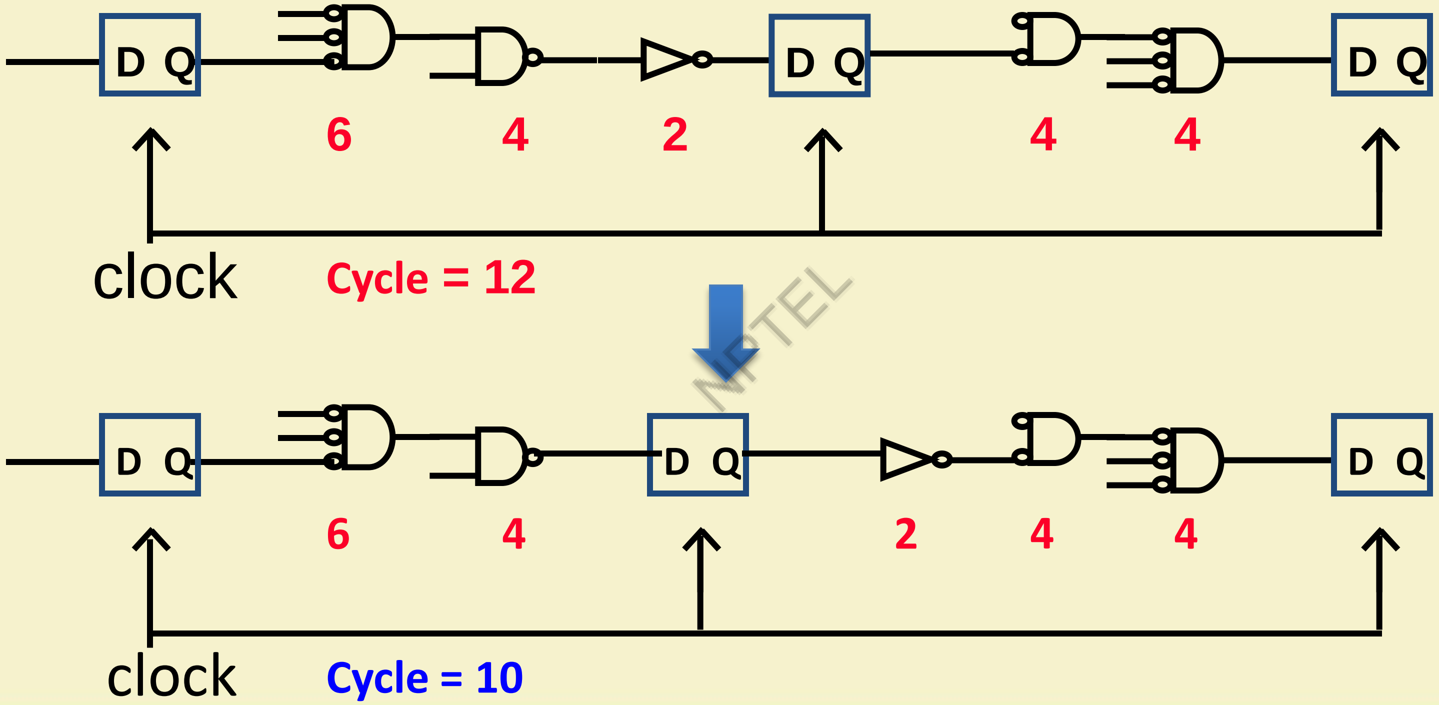
- *Retiming* and *Useful Clock Skew* based approaches.
  - Make modifications to the circuit netlist so as to meet timing requirements.
  - To be explained with the help of illustrative examples.

Re-Timing

- How to meet the 10 ns clock cycle time requirement?

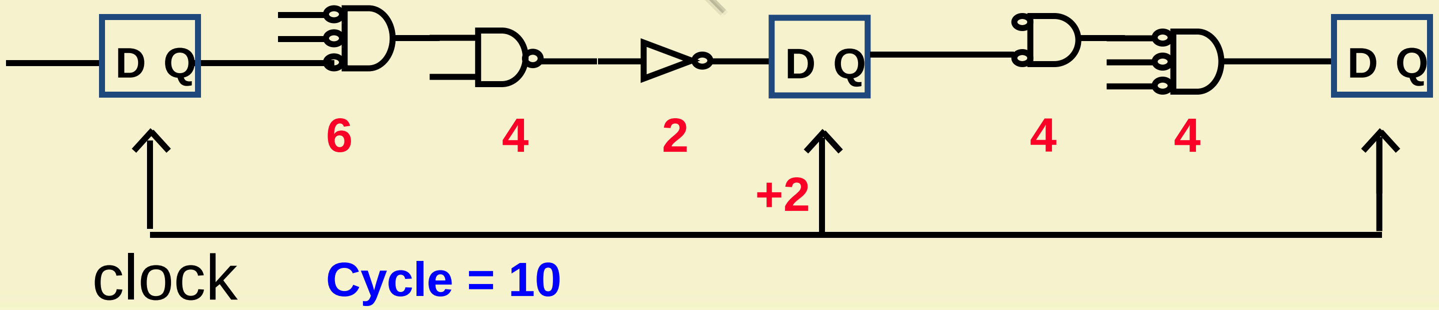


- Re-order sequential elements and combinational logic.

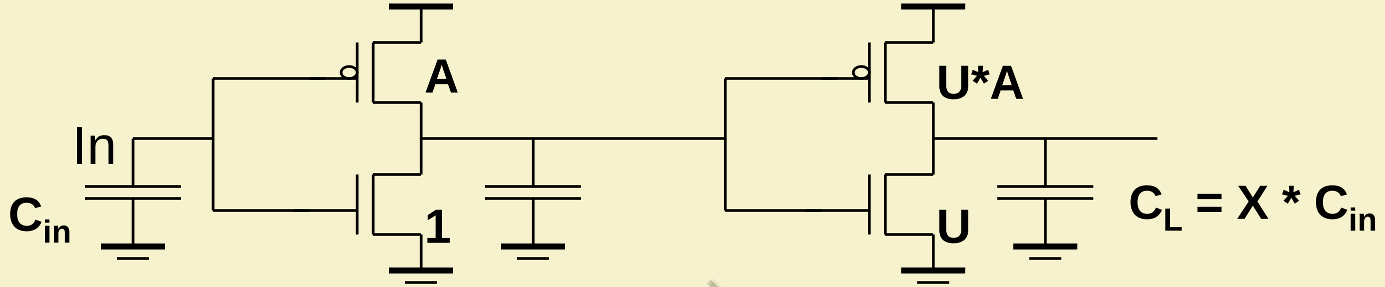


Useful Clock Skew

- Adding skew to a clock signal to meet clock period requirement.
- Several ways:
  - Insert delay cells
  - Add dummy capacitive load
  - Snaking of interconnects



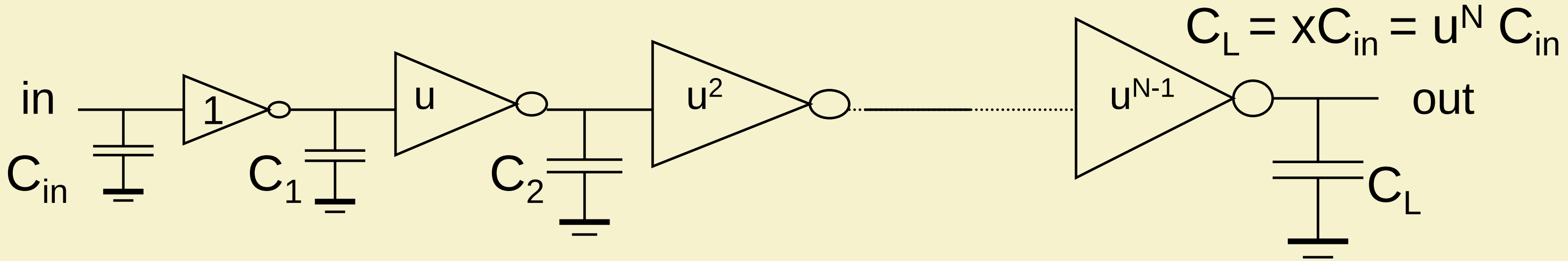
Driving Large Capacitances: Inverter as Buffer



- Total propagation delay = t_p(inv) + t_p(buffer)
- t_{p0} = delay of min-size inverter with single min-size inverter as fanout load
- Minimize $t_p = U * t_{p0} + X/U * t_{p0}$

  $U_{opt} = \sqrt{X}$; $t_{p,opt} = 2\ t_{p0} * \sqrt{X}$
- Use only if combined delay is less than that for unbuffered case.

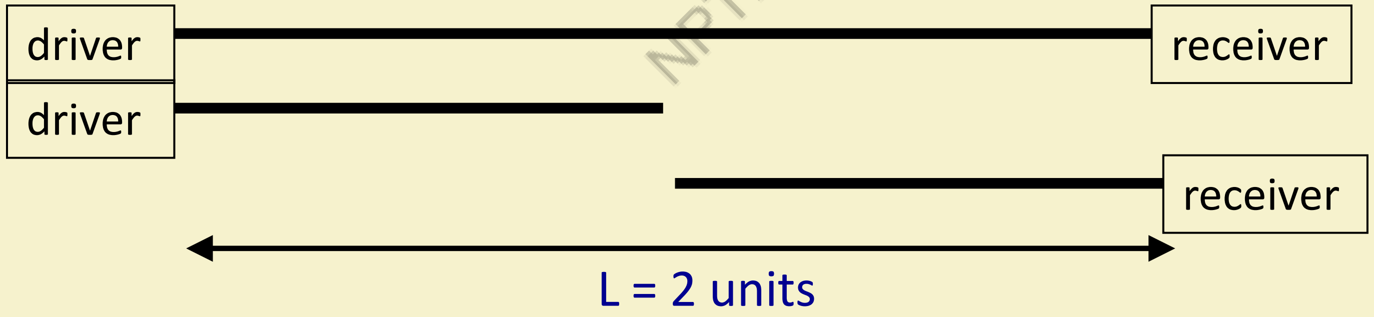
Delay Reduction with Cascaded Buffer



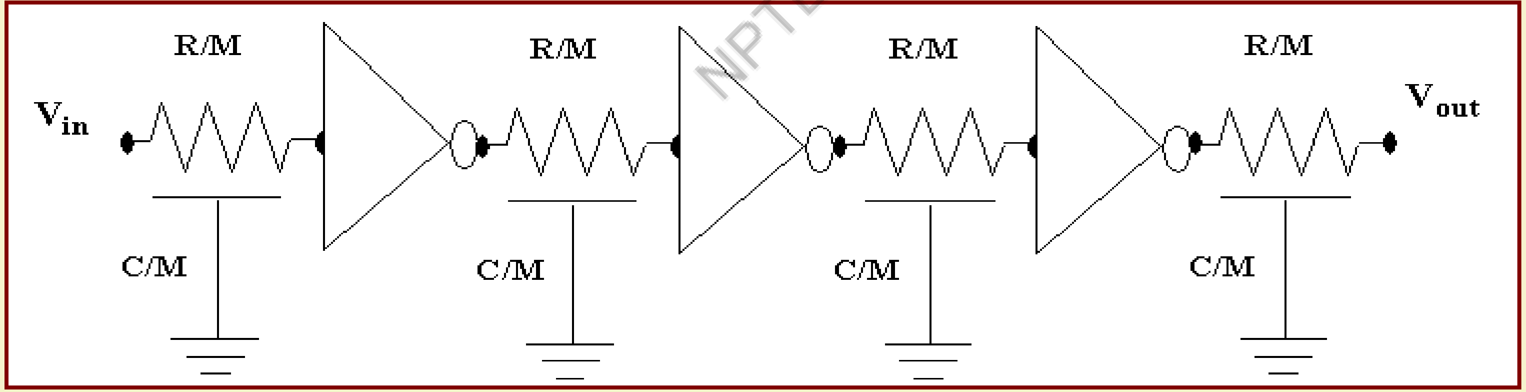
- Cascaded buffers with increasing sizes can reduce delay (*u: scaling factor*).
- If load is driven by a large transistor (which is driven by a smaller transistor), then its turn-on time dominates overall delay.
- Each buffer charges the input capacitance of the next buffer in the chain and speeds up charging, reducing total delay.
- Cascaded buffers are useful when $R_{int} < R_{tr}$.

Reducing RC Delay with Repeaters

- RC delay is quadratic in length → must reduce length of interconnects.
- Observation: $2^2 = 4$ and $1 + 1 = 2$, but $1^2 + 1^2 = 2$.



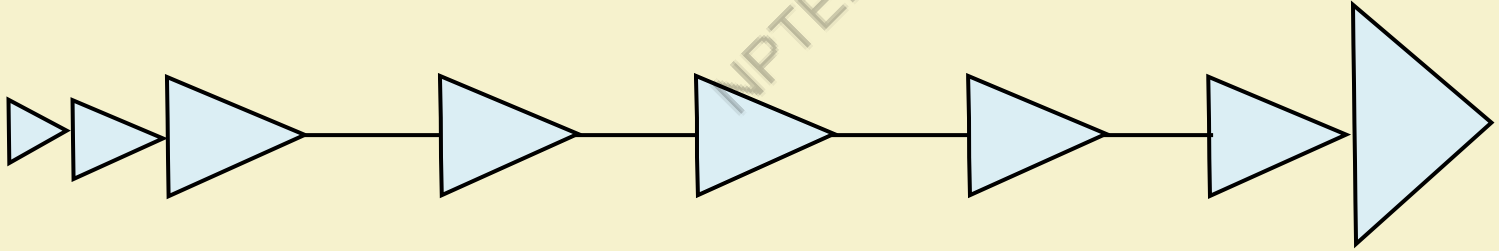
- *Repeater* :: strong driver (usually inverter of pair of inverters) that is placed along a long RC line to *break-up* the line and reduce delay.



Repeaters versus Cascaded Buffers

- Repeaters are used to drive long RC lines.
  - Breaking up the quadratic dependence of delay on line length is the goal.
  - Typically sized identically.
- Cascaded buffers are used to drive large capacitive loads, where there is no parasitic resistance.
  - We put all buffers at the beginning of the load.
  - Useless for a long RC wire since the wire RC delay would be unaffected and would dominate the total delay.

- Optimum buffering for a uniform long interconnect.
  - Cascaded buffers at source and sink.
  - Identical sized and spaced repeaters in between.



END OF LECTURE 42