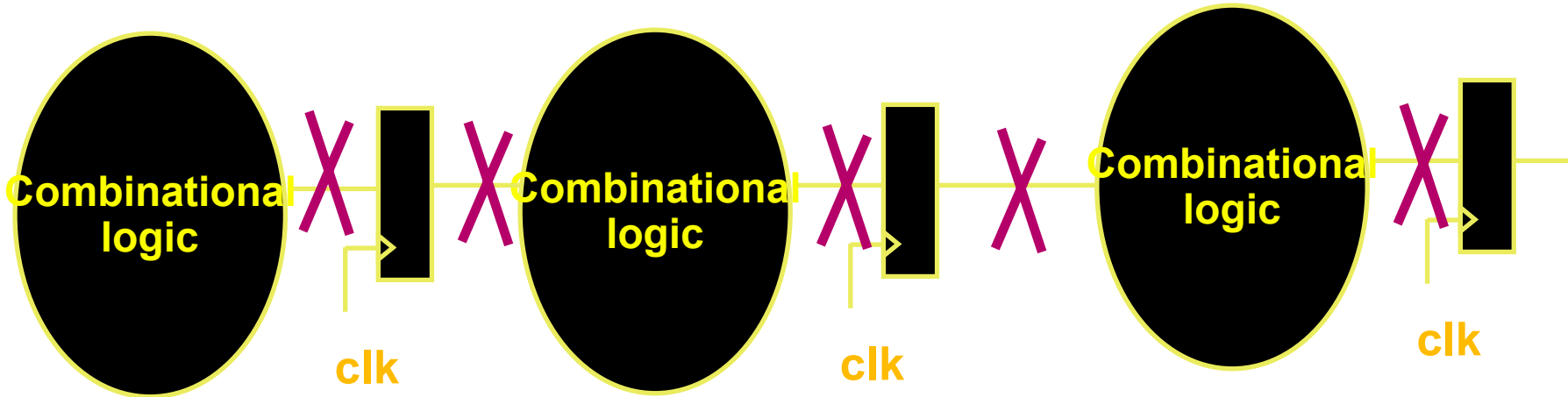
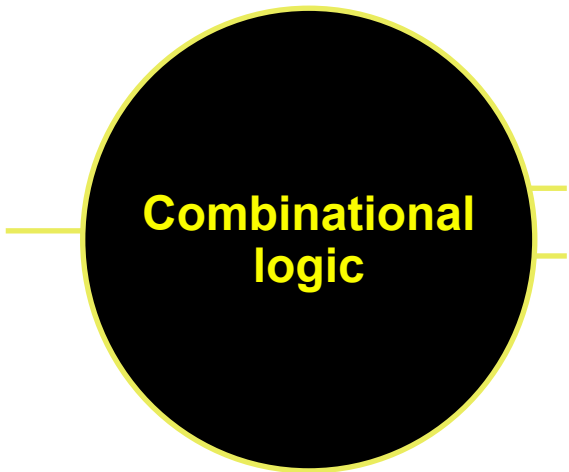
Static Timing Analysis

Approach – Reduce to Combinational



original circuit

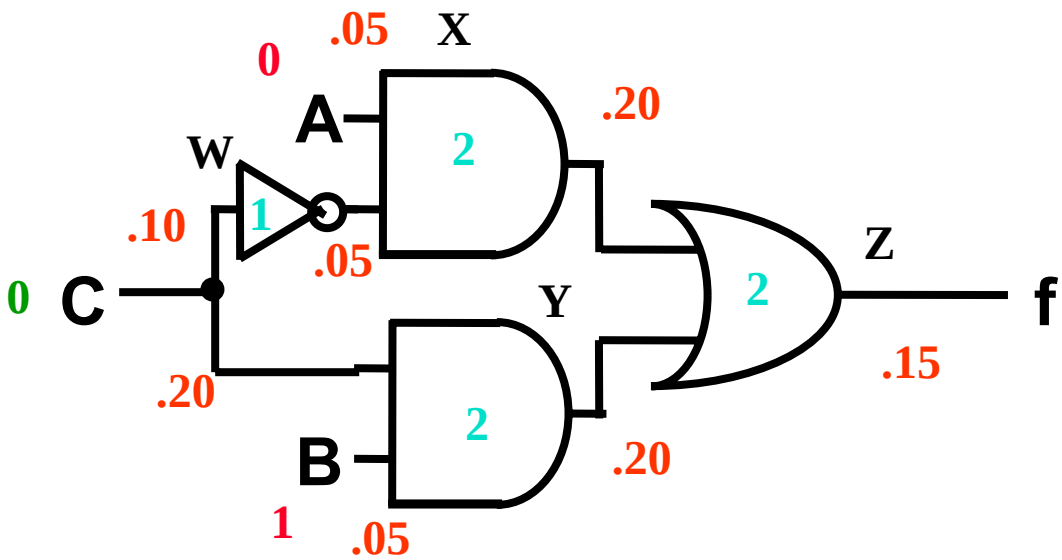


extracted block

Courtesy K. Keutzer et al. UCB

Combinational Block

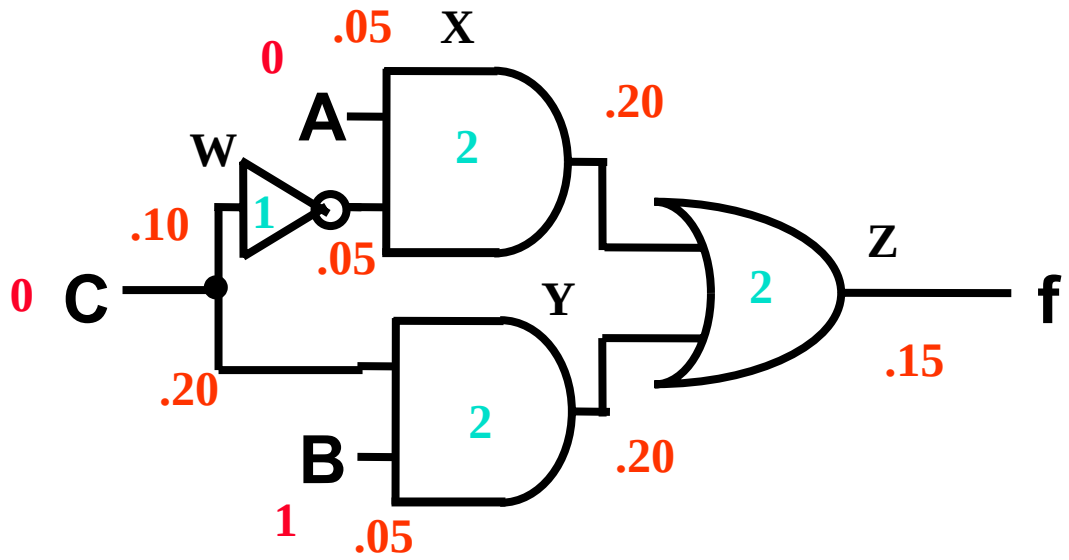
- Arrival time in green
- Interconnect delay in red
- Gate delay in blue

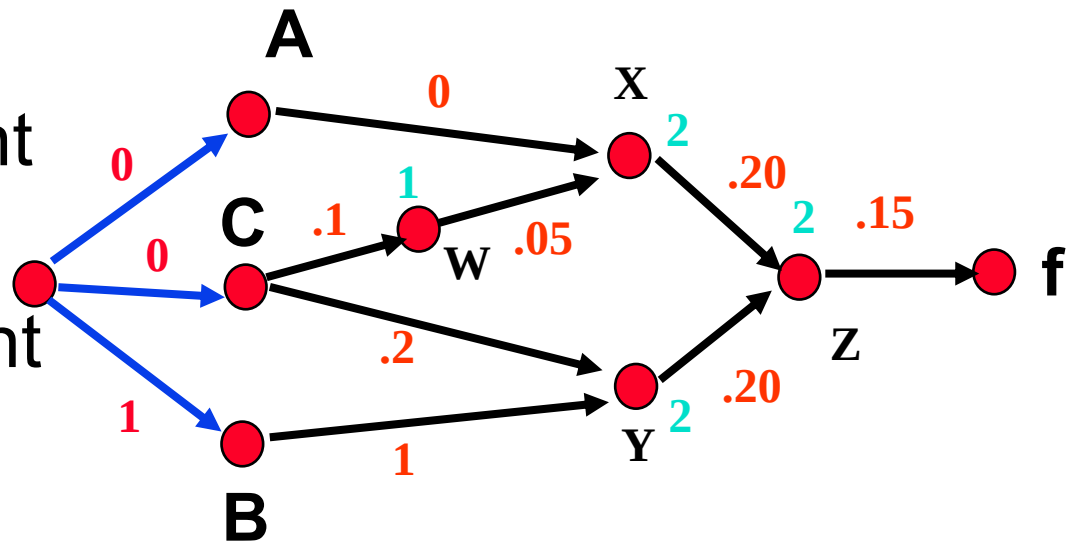


- What's the right mathematical object to use to represent this physical object?

Problem Formulation - 1

- Use a labeled *directed* graph
- $G = \langle V, E \rangle$
- *Vertices* represent gates, primary inputs and primary outputs
- *Edges* represent wires
- *Labels* represent delays

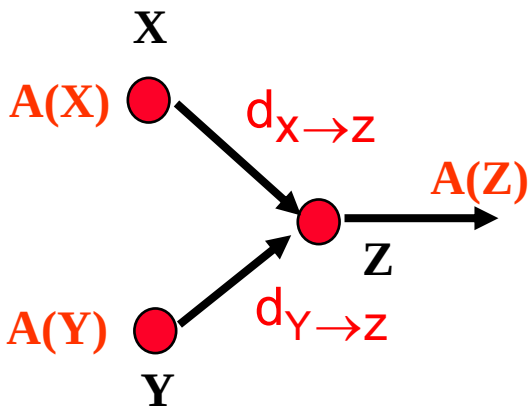




Courtesy K. Keutzer et al. UCB

Problem Formulation – Actual Arrival Time

- Actual arrival time A(v) for a node v is latest time that signal can arrive at node v

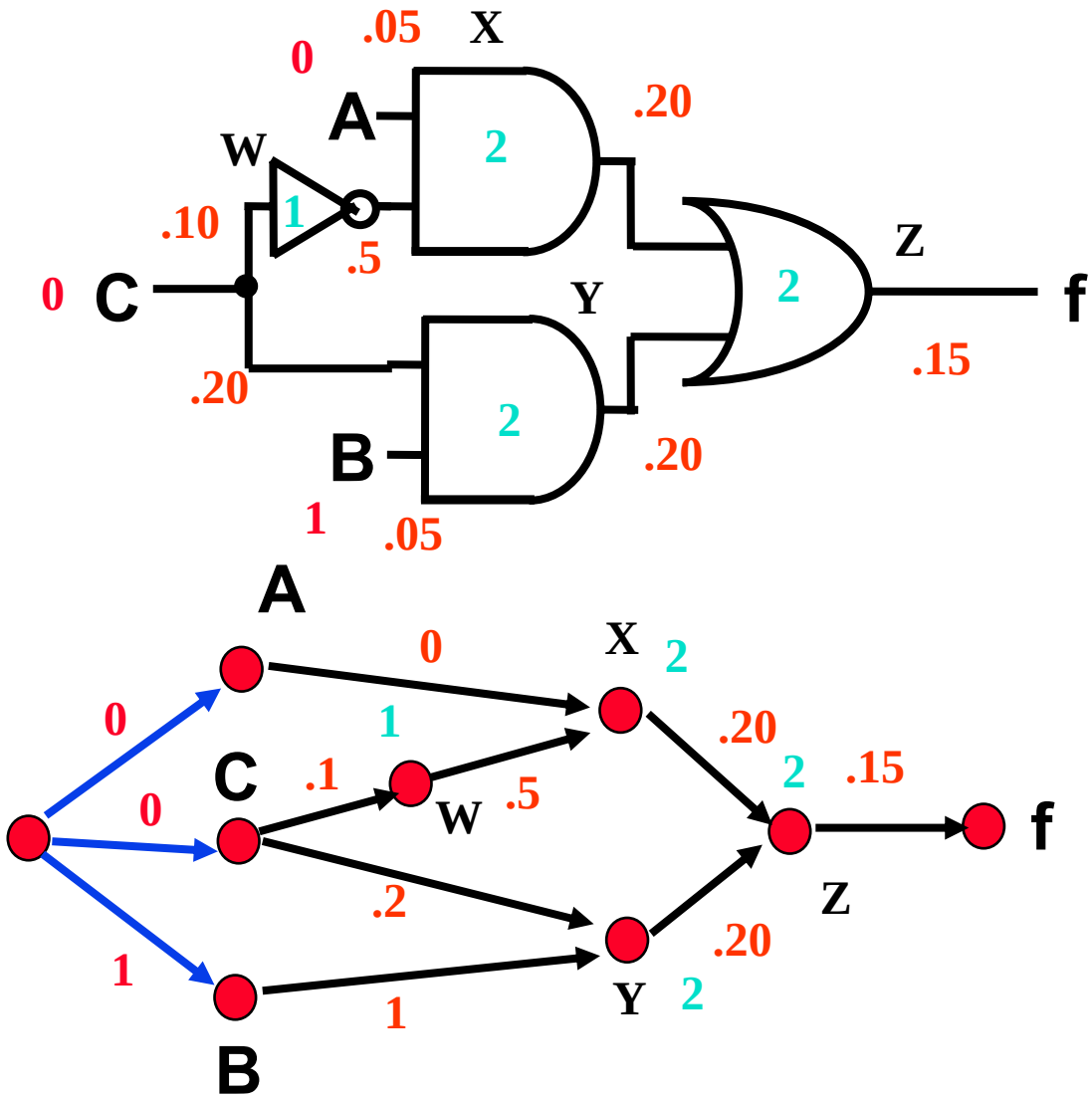


$$A(\upsilon) = \max_{u \in FI(\upsilon)} (A(u) + d_{u \to \upsilon})$$

where $d_{\upsilon \to u}$ is delay from υ to u, $FI(u)$ = {X, Y}, and υ = {Z}.

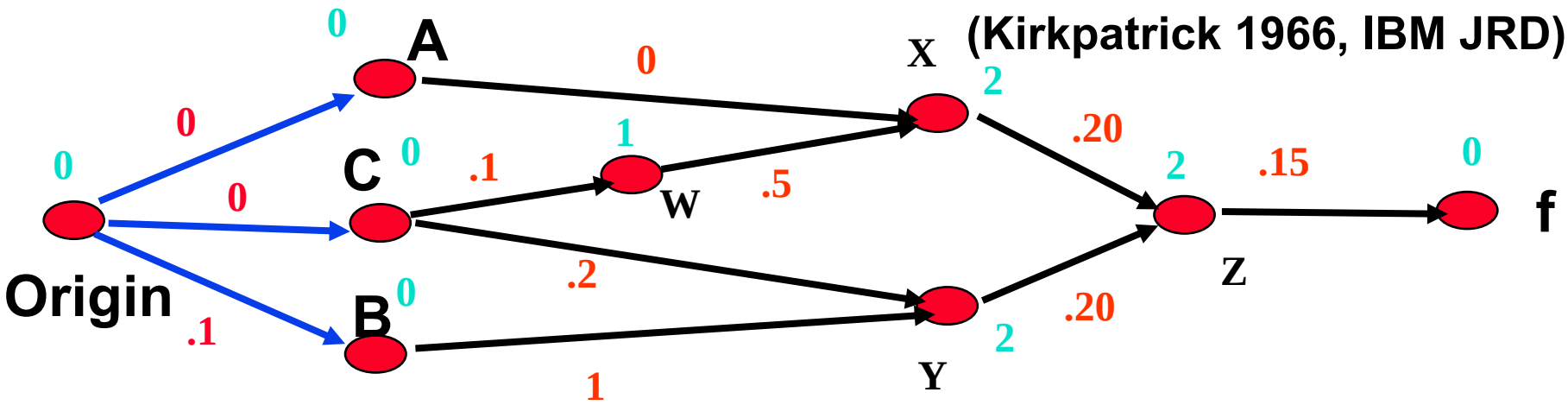
Problem Formulation - 2

- Use a labeled *directed* graph
- *G = <V,E>*
- *Enumerate all paths - choose the longest?*



Courtesy K. Keutzer et al. UCB

Problem Formulation - 3



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

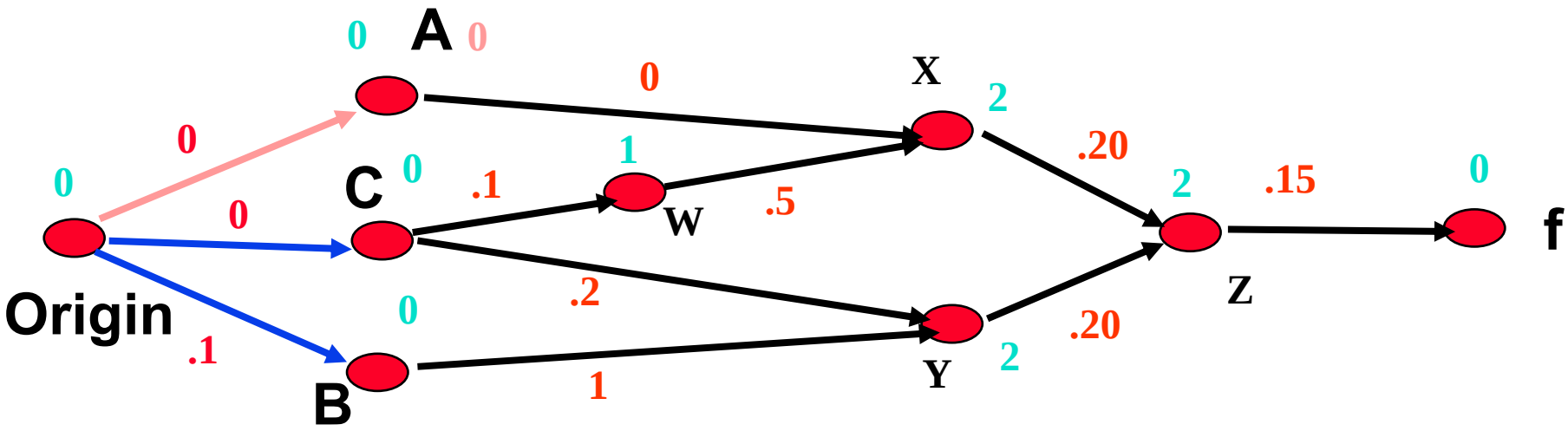
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

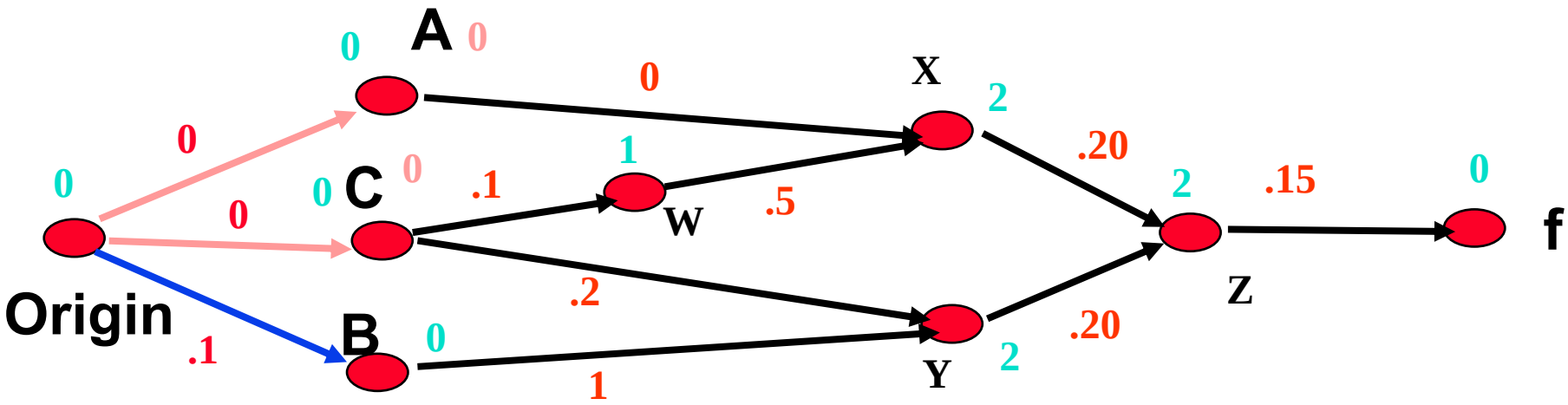
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

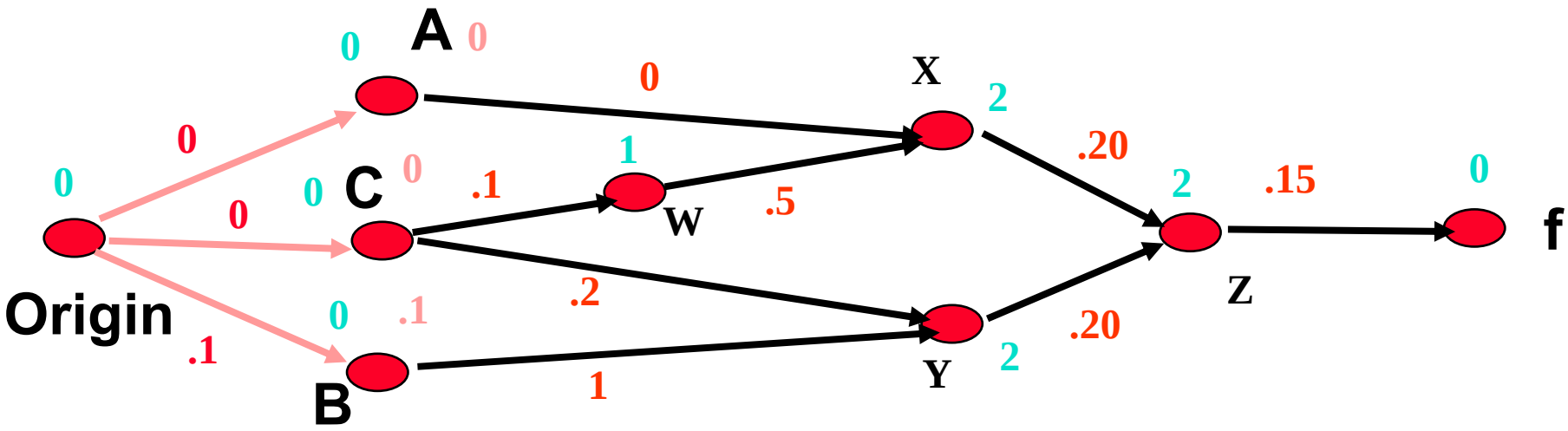
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

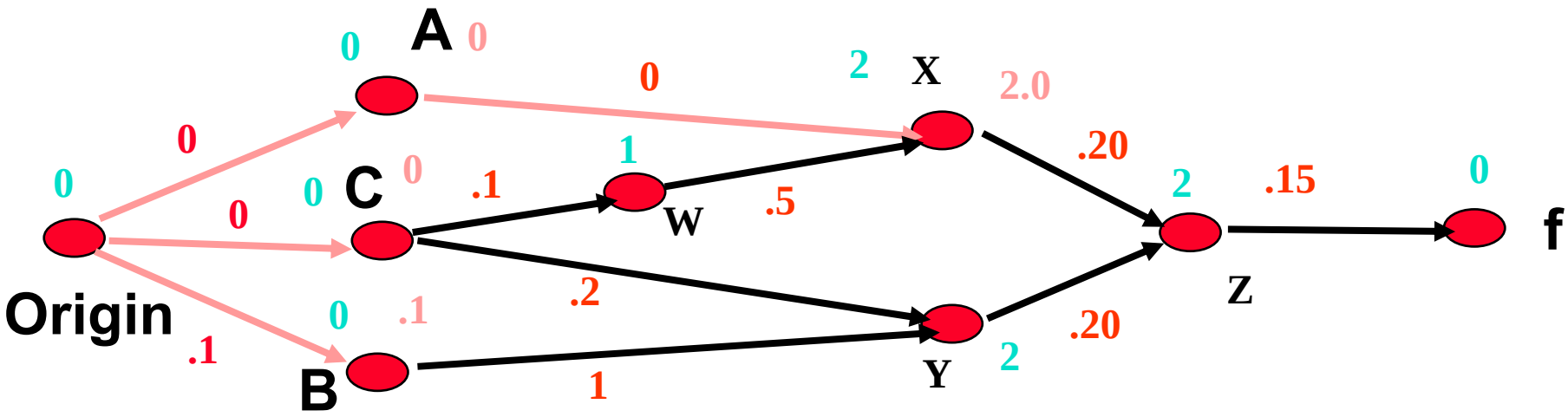
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

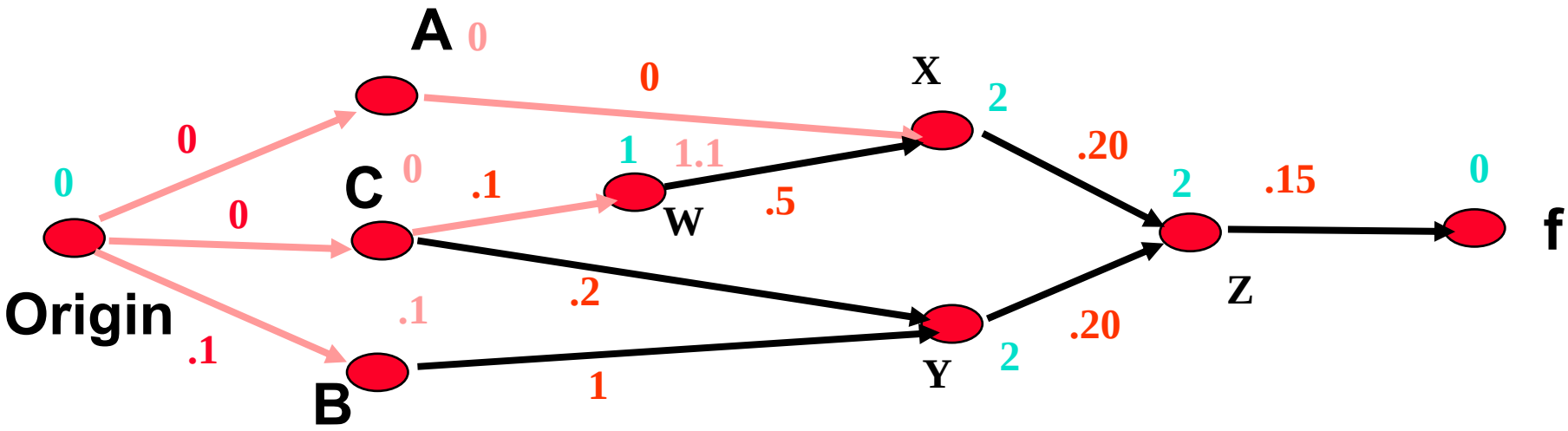
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

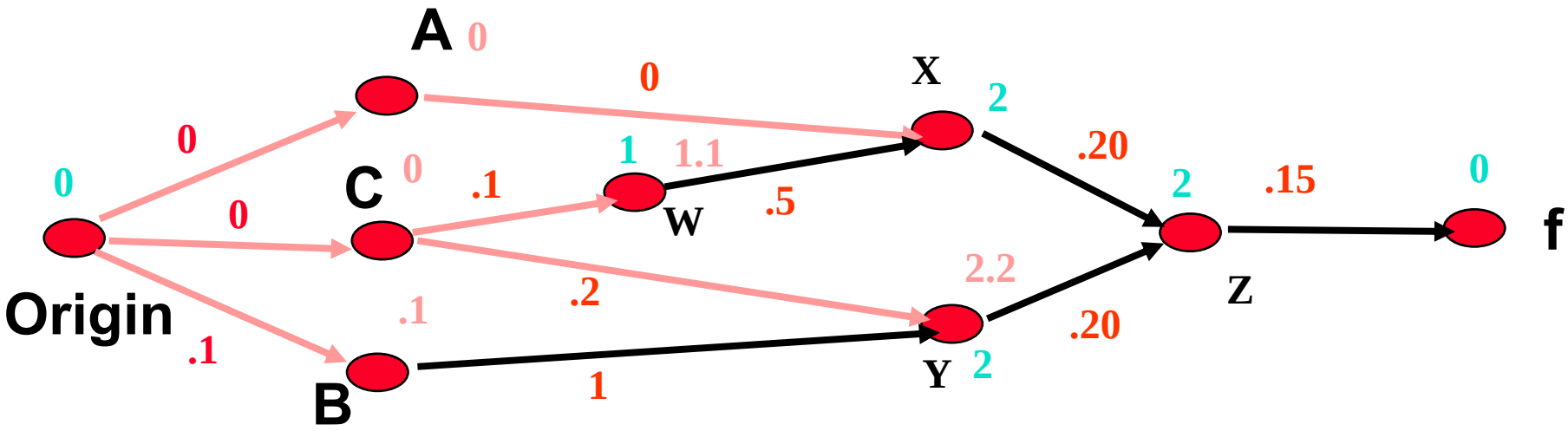
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

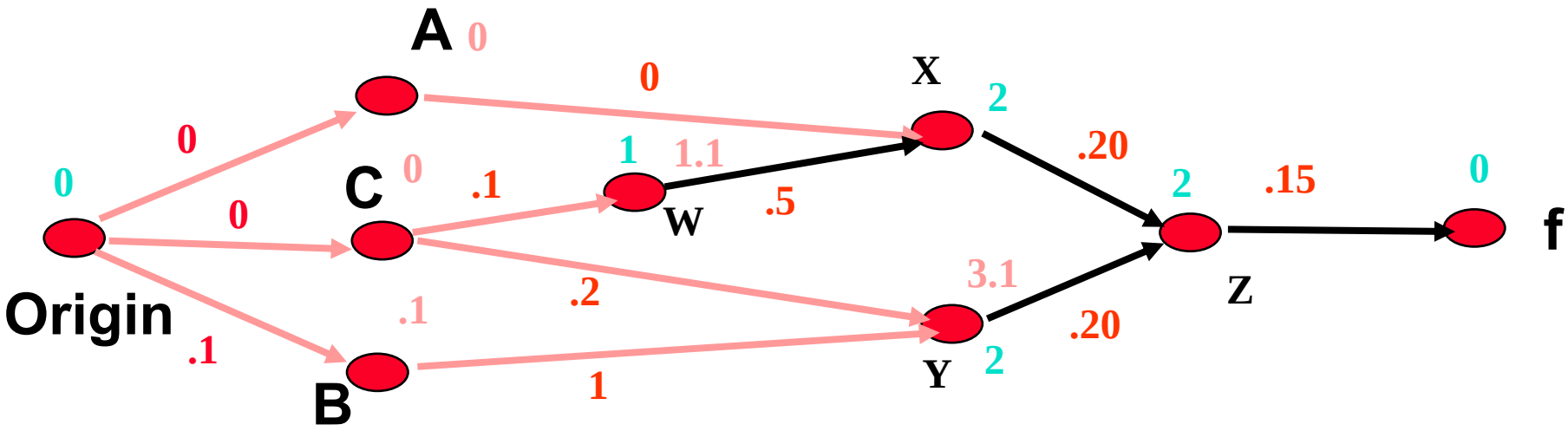
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

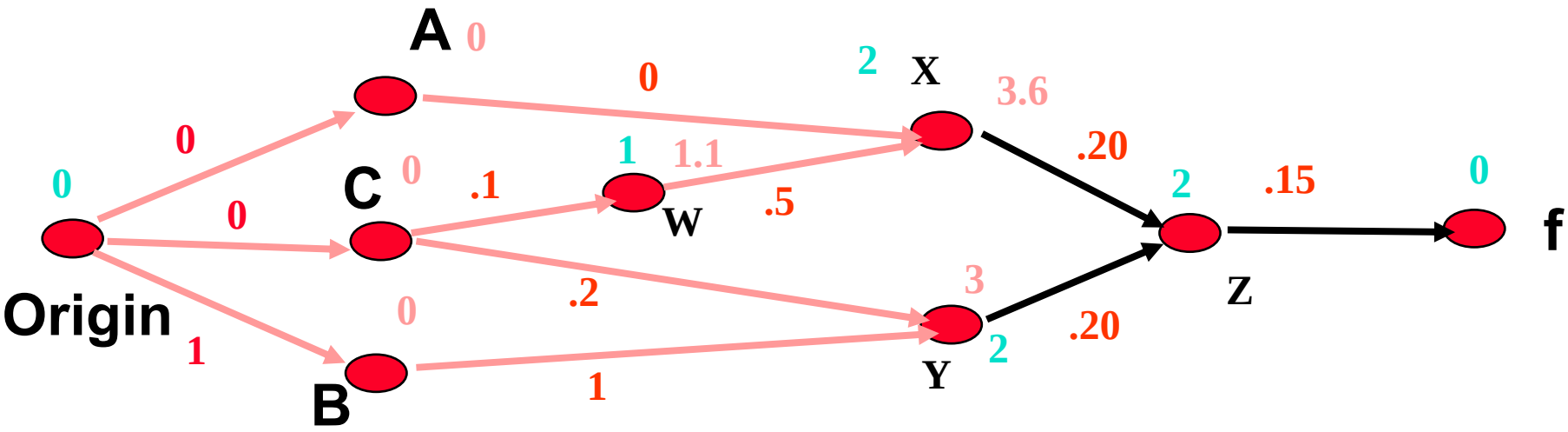
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

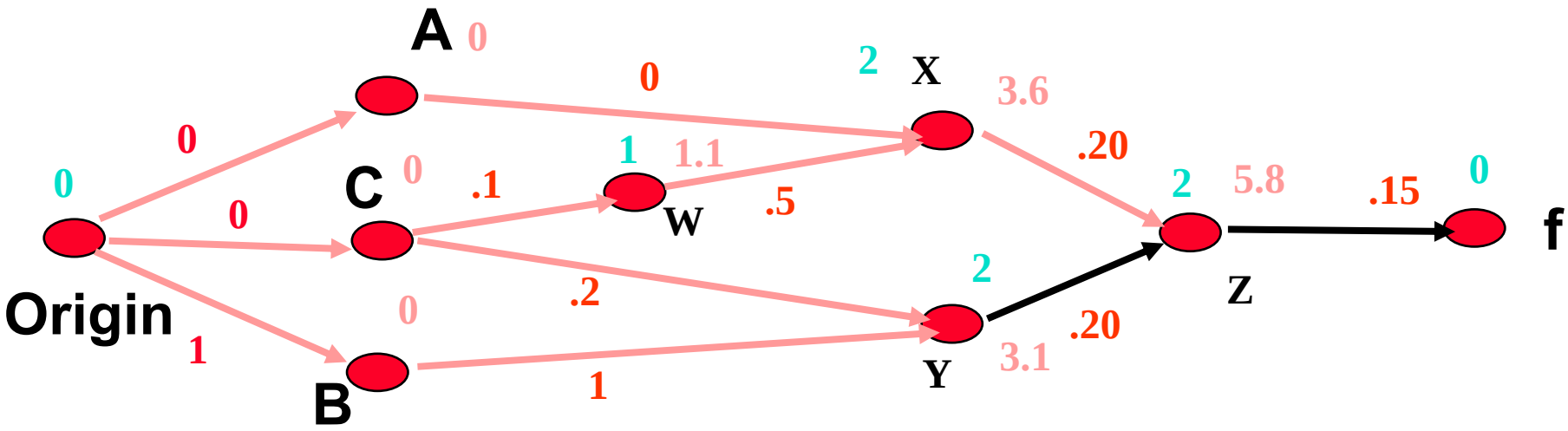
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

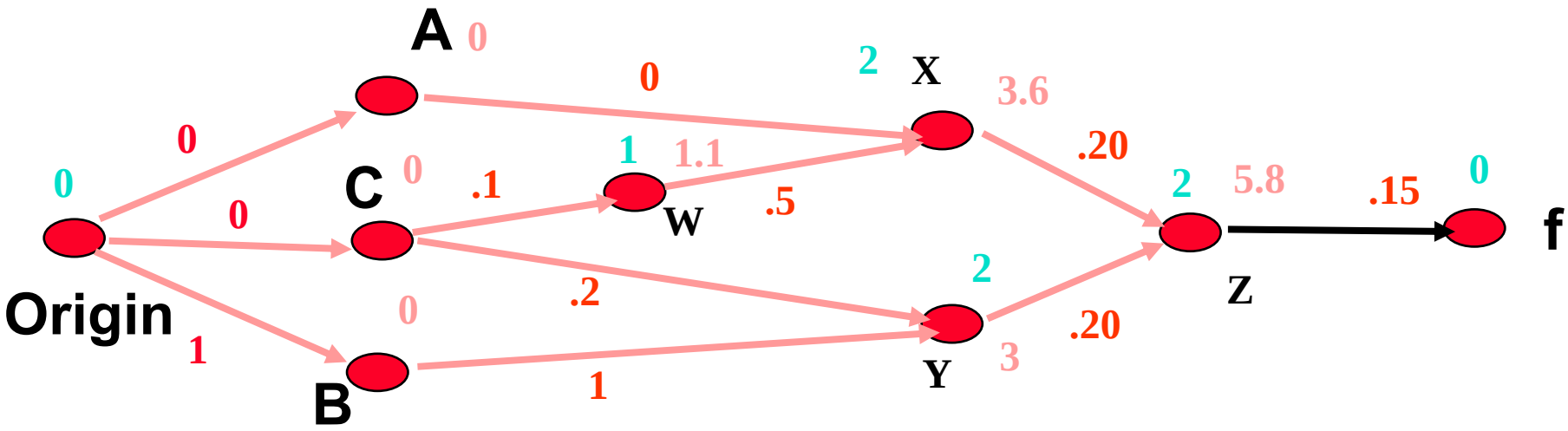
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

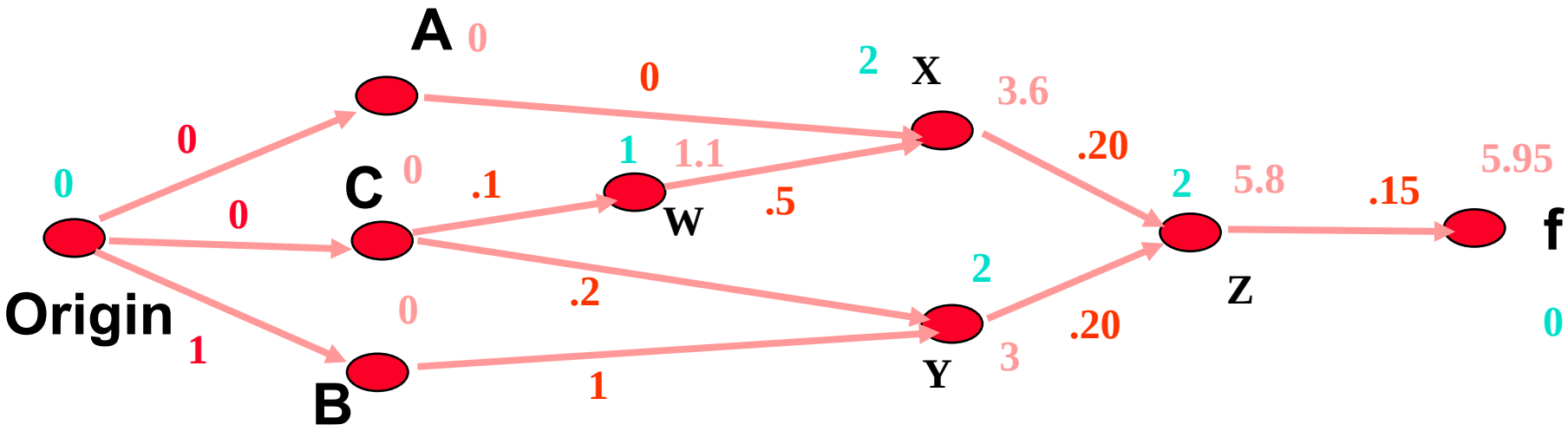
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Algorithm Execution



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

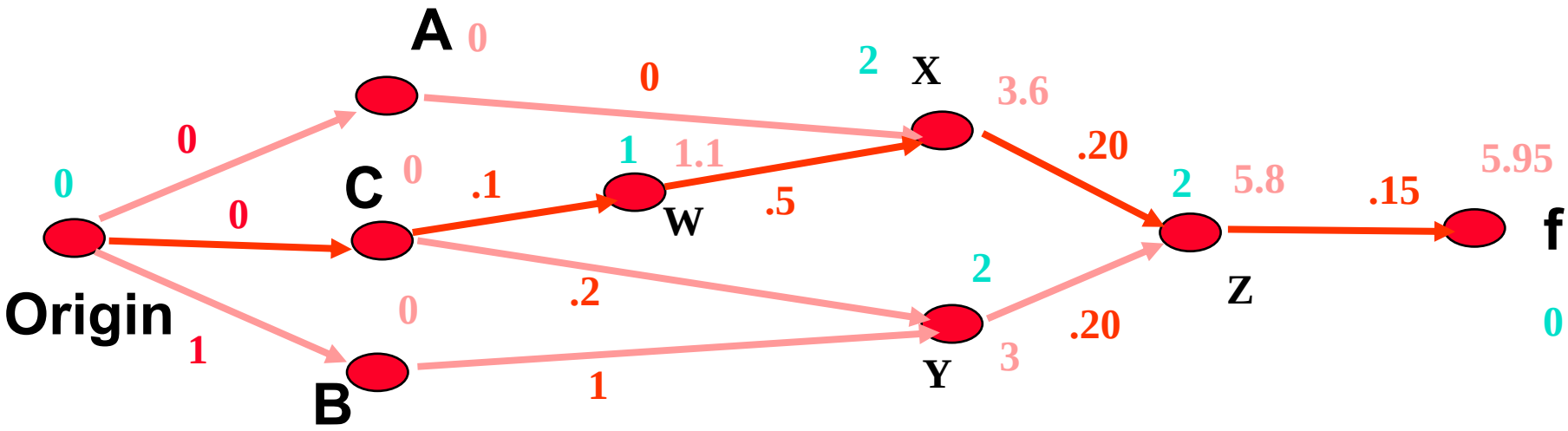
if all incoming edges of *w* have been traversed, add *w* to *W*

}

Longest path(G){

Forward_prop(Origin) }

Critical Path (sub-graph)



Compute longest path in a graph *G = <V,E,delay,Origin>*

// delay is set of labels, Origin is the super-source of the DAG

Forward-prop(*W*){

for each vertex *v* in *W*

for each edge *<v,w>* from *v*

Final-delay(w) = max(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))

if all incoming edges of *w* have been traversed, add *w* to *W*

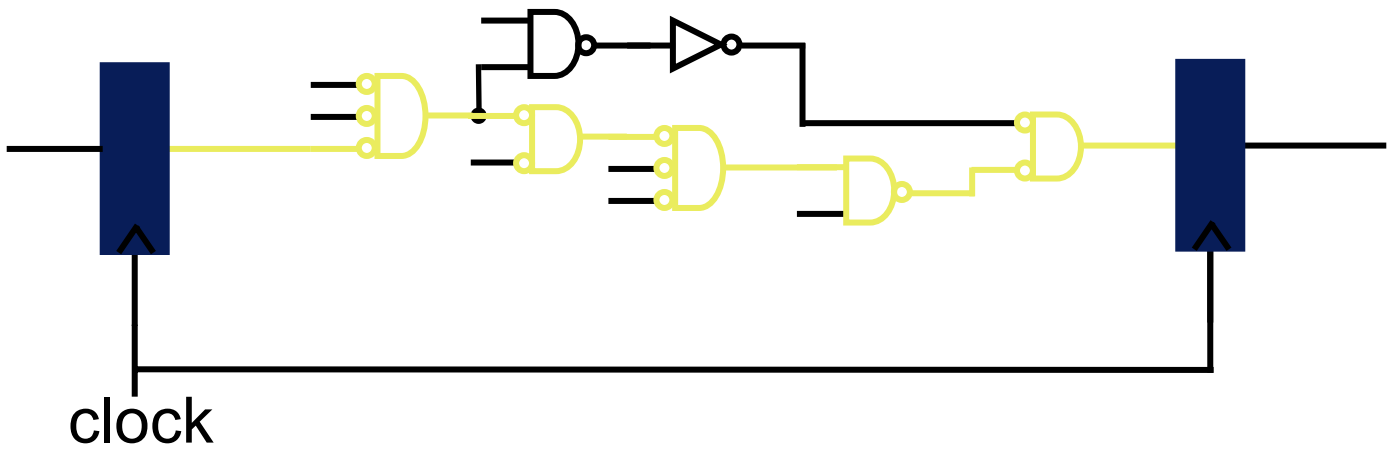
}

Longest path(G){

Forward_prop(Origin) }

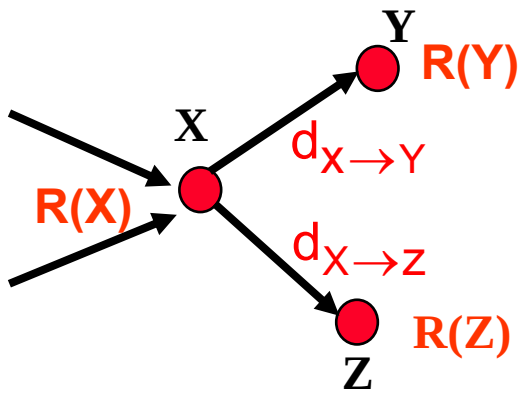
Timing for Optimization: Extra Requirements

- Longest-path algorithm computes arrival times at each node
- If we have constraints, need to propagate slack to each *node*
  - A measure of how much timing margin exists at each node
  - Can optimize a particular branch
- Can trade slack for power, area, robustness



Required Arrival Time

- Required arrival time R(v) is the time before which a signal must arrive to avoid a timing violation



Required time is user defined at output:

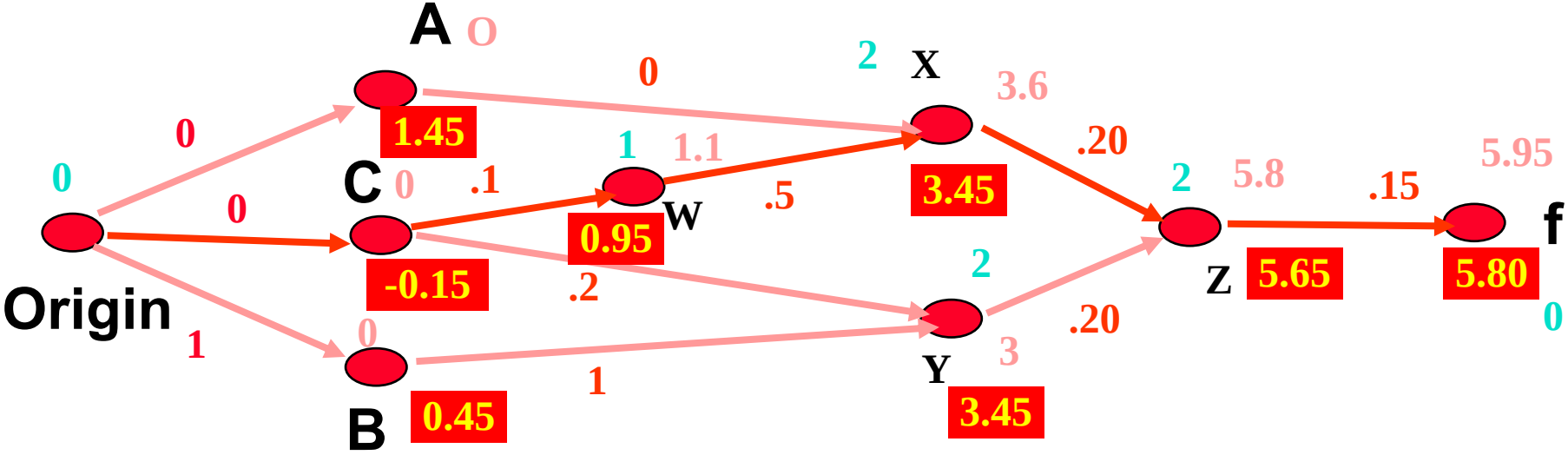
$$R(v) = T - T_{setup}$$

$$R(\upsilon) = \min_{u \in FO(\upsilon)} (R(u) - d_{\upsilon \to u})$$

- Then recursively

where $FO(\upsilon) = \{Y, Z\}$ and $\upsilon = \{X\}$

Required Time Propagation: Example



- Assume required time at output R(f) = 5.80
- Propagate required times backwards

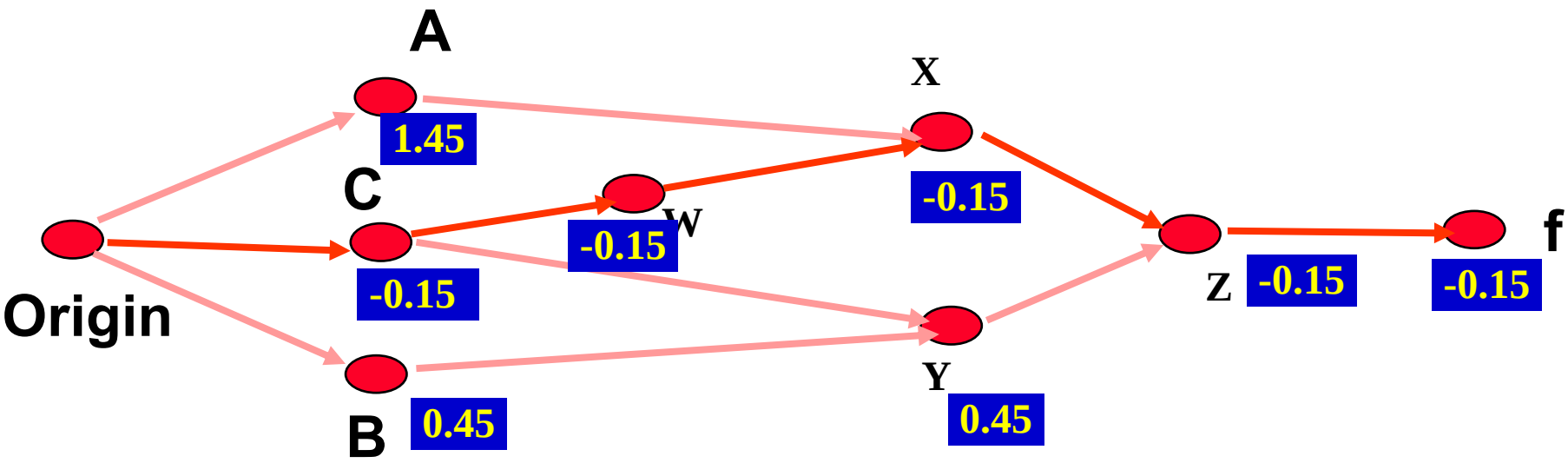
Timing Slack

- From arrival and required time can compute slack. For each node v:

$$S(\upsilon) = R(\upsilon) - A(\upsilon)$$

- Slack reflects criticality of a node
- Positive slack
  - Node is not on critical path. Timing constraints met.
- Zero slack
  - Node is on critical path. Timing constraints are barely met.
- Negative slack
  - There is a timing violation
- Slack distribution is key for timing optimization!

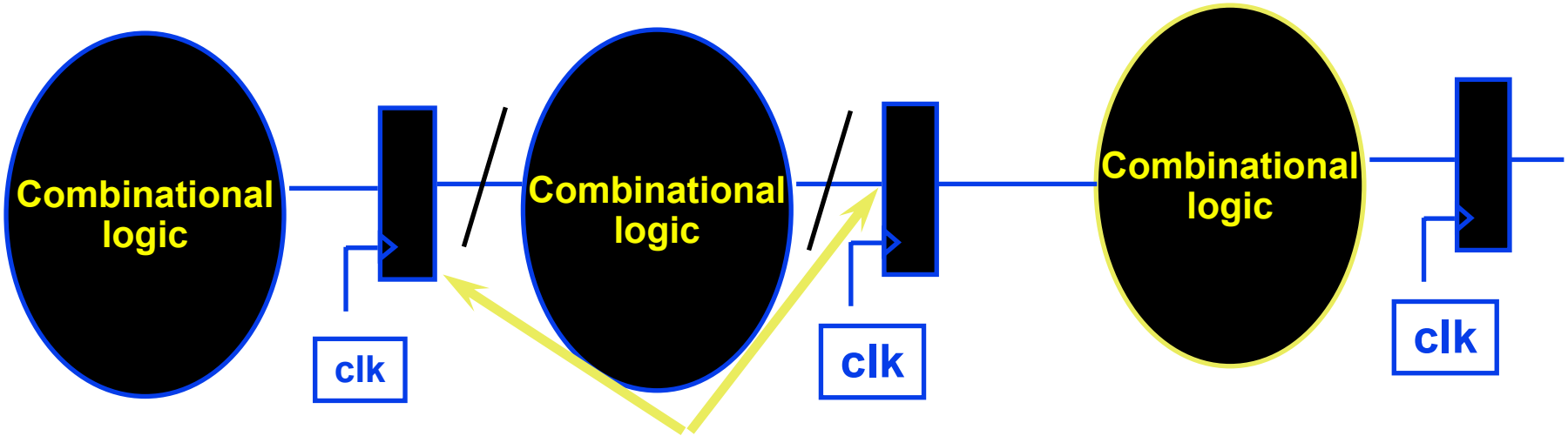
Timing Slack Computation: Example



- Compute slack at each node

$$S(\upsilon) = R(\upsilon) - A(\upsilon)$$

Approach of Static Timing Verification



- Computing longest path delay
  - Full path enumeration - potentially exponential
  - Longest path algorithm on DAG (Kirkpatrick 1966, IBM JRD) (*O(v+e)* or *O(g + p)*)
- Currently applied to even the largest (>10M gate) circuits
- Two challenges
  - Asynchronous subcircuits
  - False paths

Enhancements of STA

- Incremental timing analysis
- Nanometer-scale process effects – variation (→ probabilistic timing analysis)
- Interference – crosstalk
- Multiple inputs switching
- Conservatism of delay propagation

Timing Correction

- Driven by STA
  - “Incremental performance analysis backplane”
- Fix electrical violations
  - Resize cells
  - Buffer nets
  - Copy (clone) cells
- Fix timing problems
  - Local transforms (bag of tricks)
  - Path-based transforms

Local Synthesis Transforms

- Resize cells
- Buffer or clone to reduce load on critical nets
- Decompose large cells
- Swap connections on commutative pins or among equivalent nets
- Move critical signals forward
- Pad early paths
- Area recovery

Transform Example

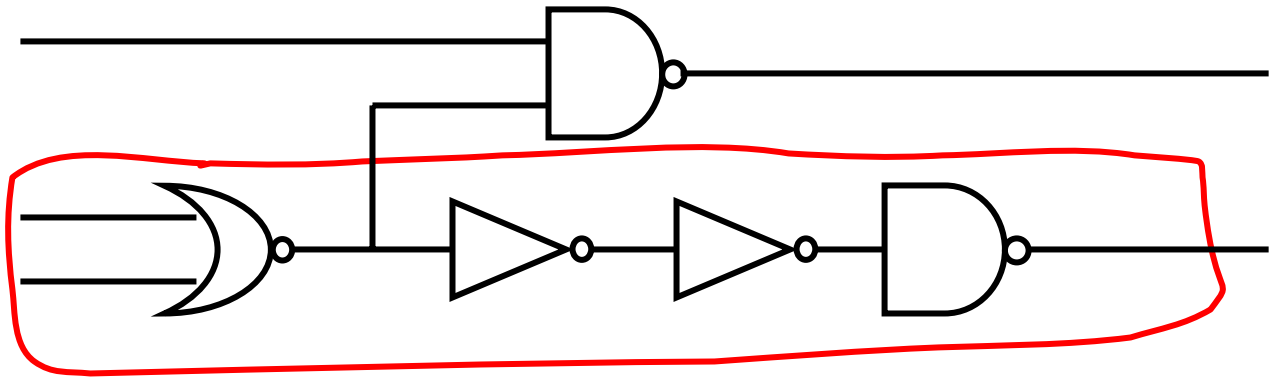
…..

Double Inverter

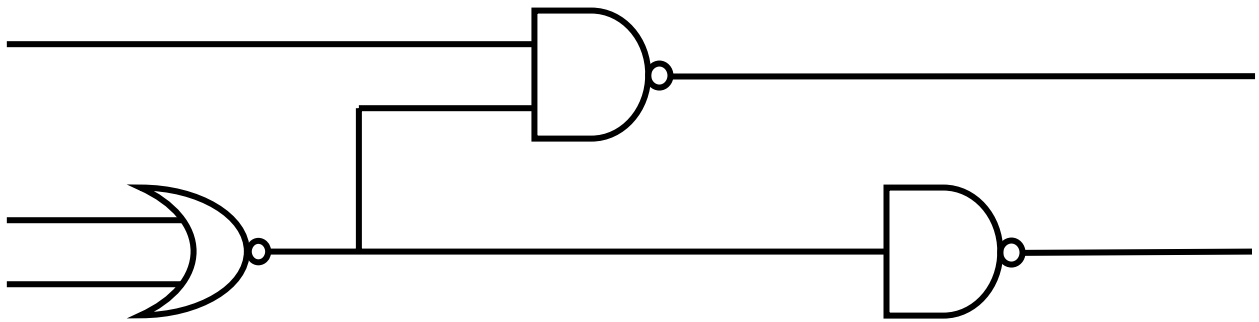
Removal

…..

…..

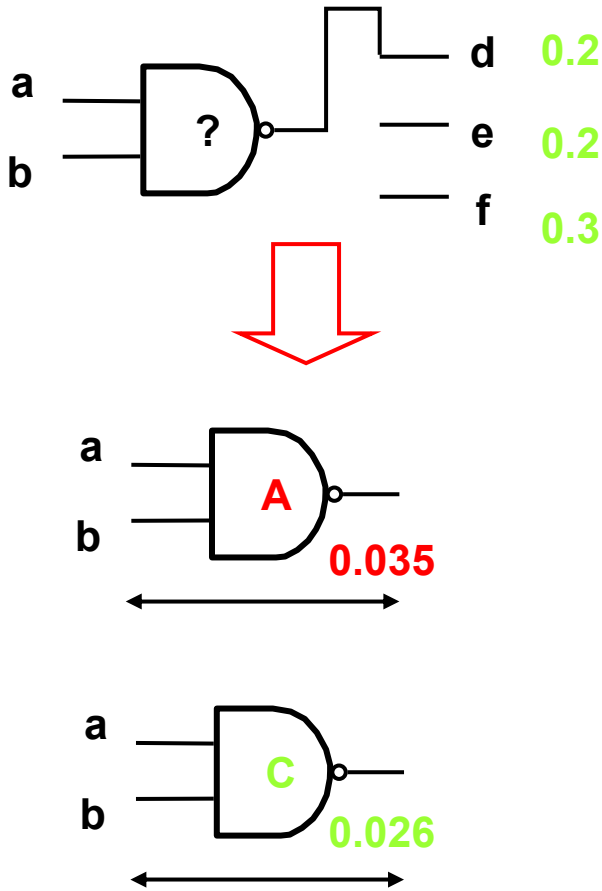


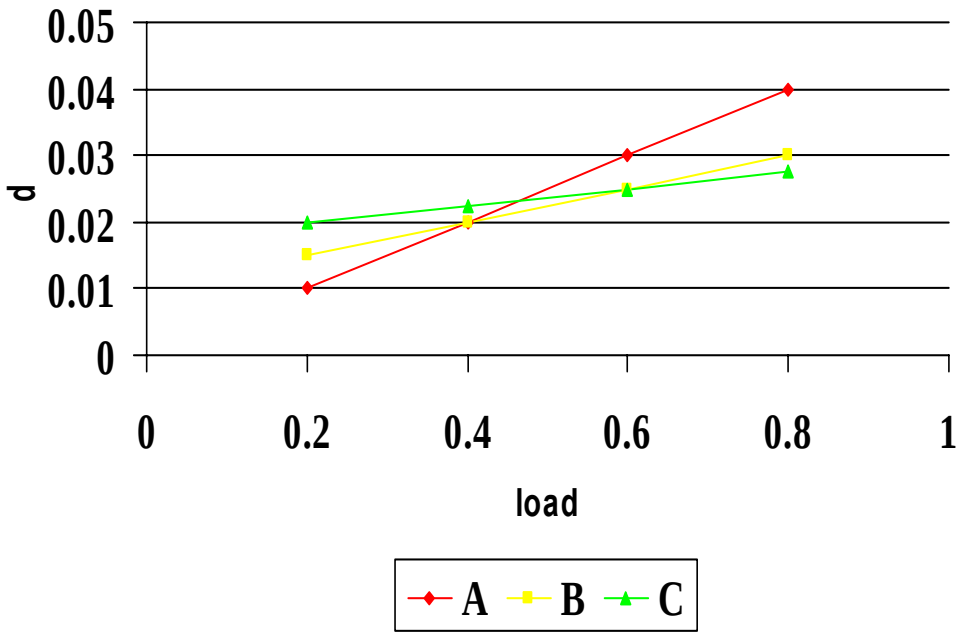
Delay = 4



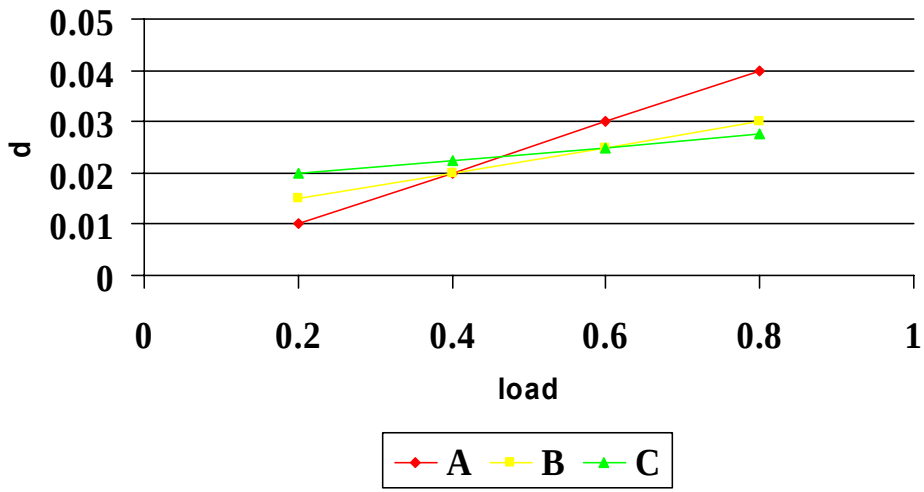
Delay = 2

Resizing


a
b
?
d
e
f
0.2
0.2
0.3
a
b
A
0.035
a
b
C
0.026

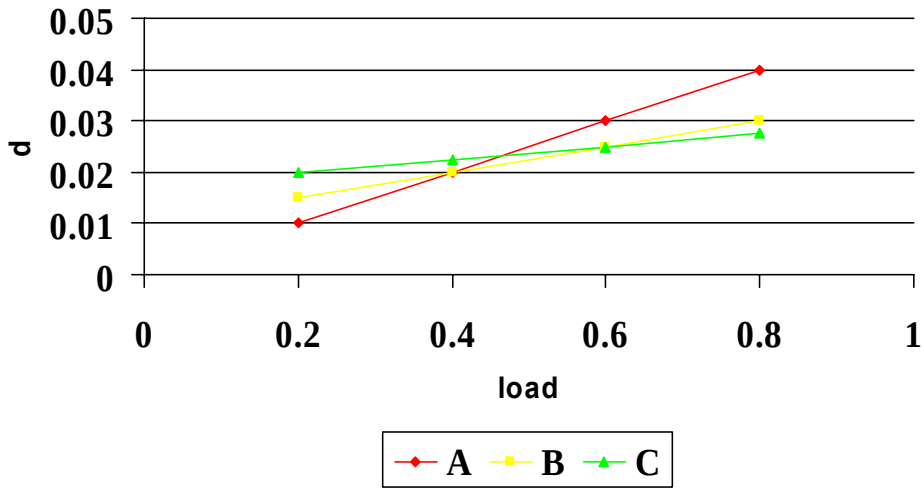

0.05
0.04
0.03
0.02
0.01
0
d
0
0.2
0.4
0.6
0.8
1
load
A
B
C

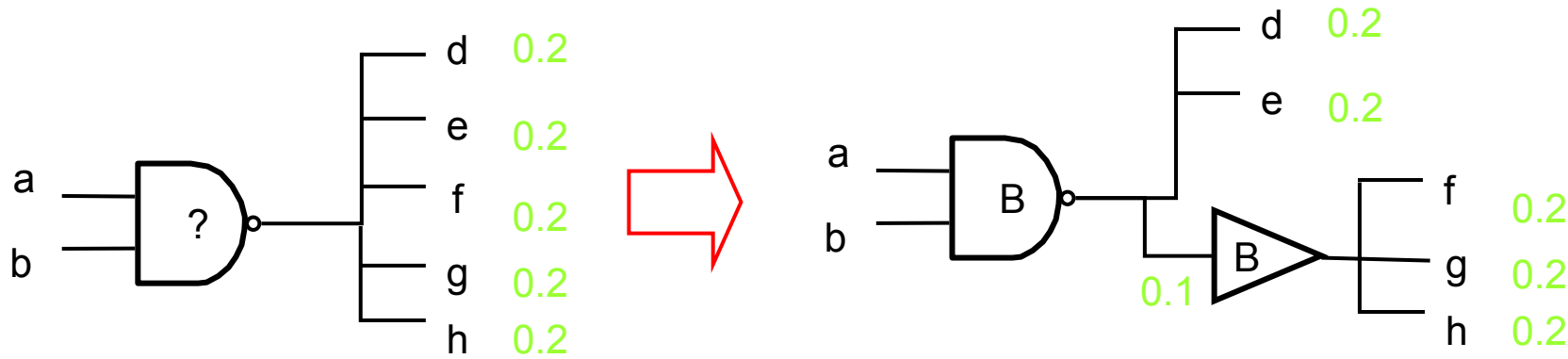
Cloning


0.05
0.04
0.03
0.02
0.01
0
d
0
0.2
0.4
0.6
0.8
1
load
A
B
C

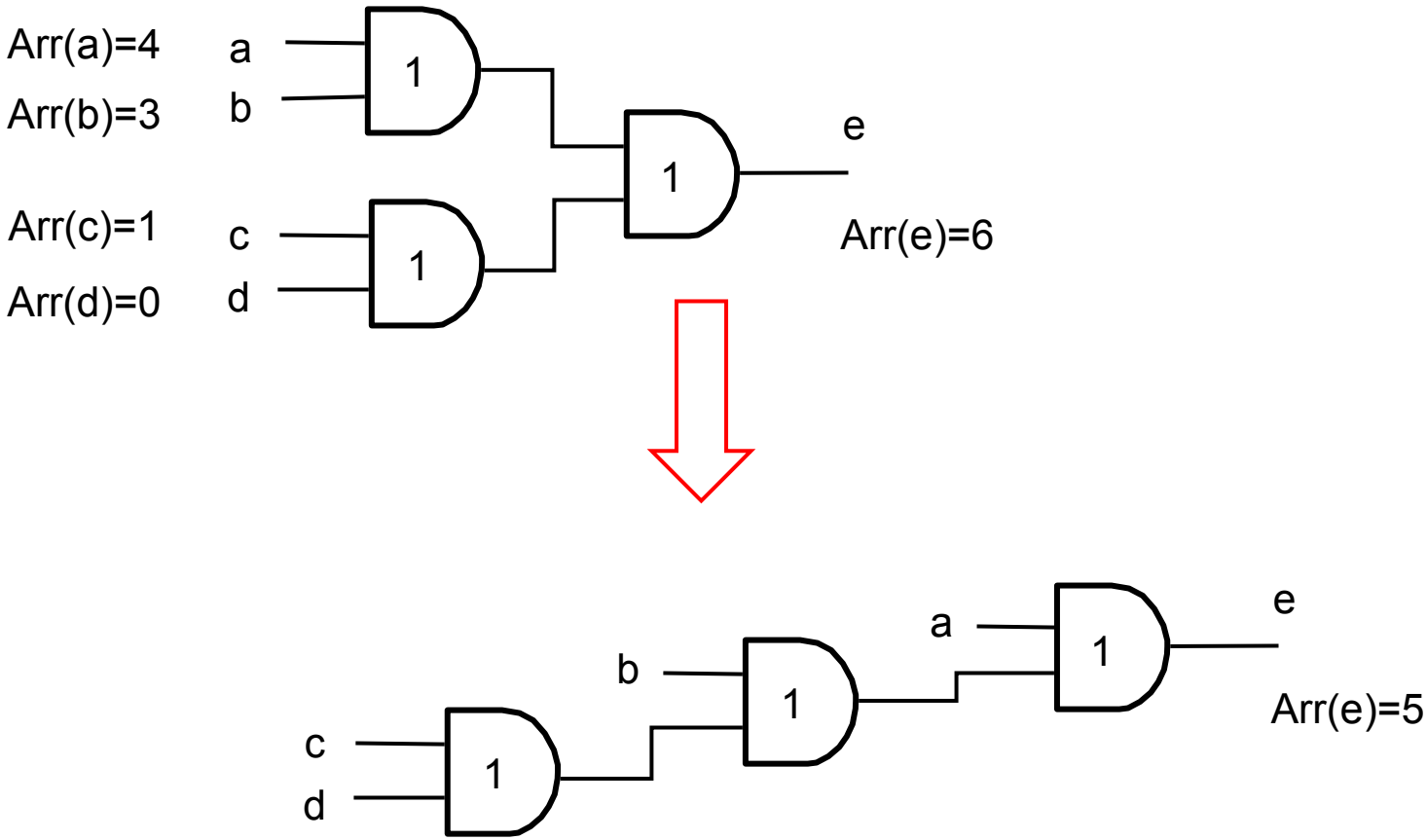

a
b
?
d
e
f
g
h
0.2
0.2
0.2
0.2
0.2
A
B
a
b
d
e
f
g
h

Buffering

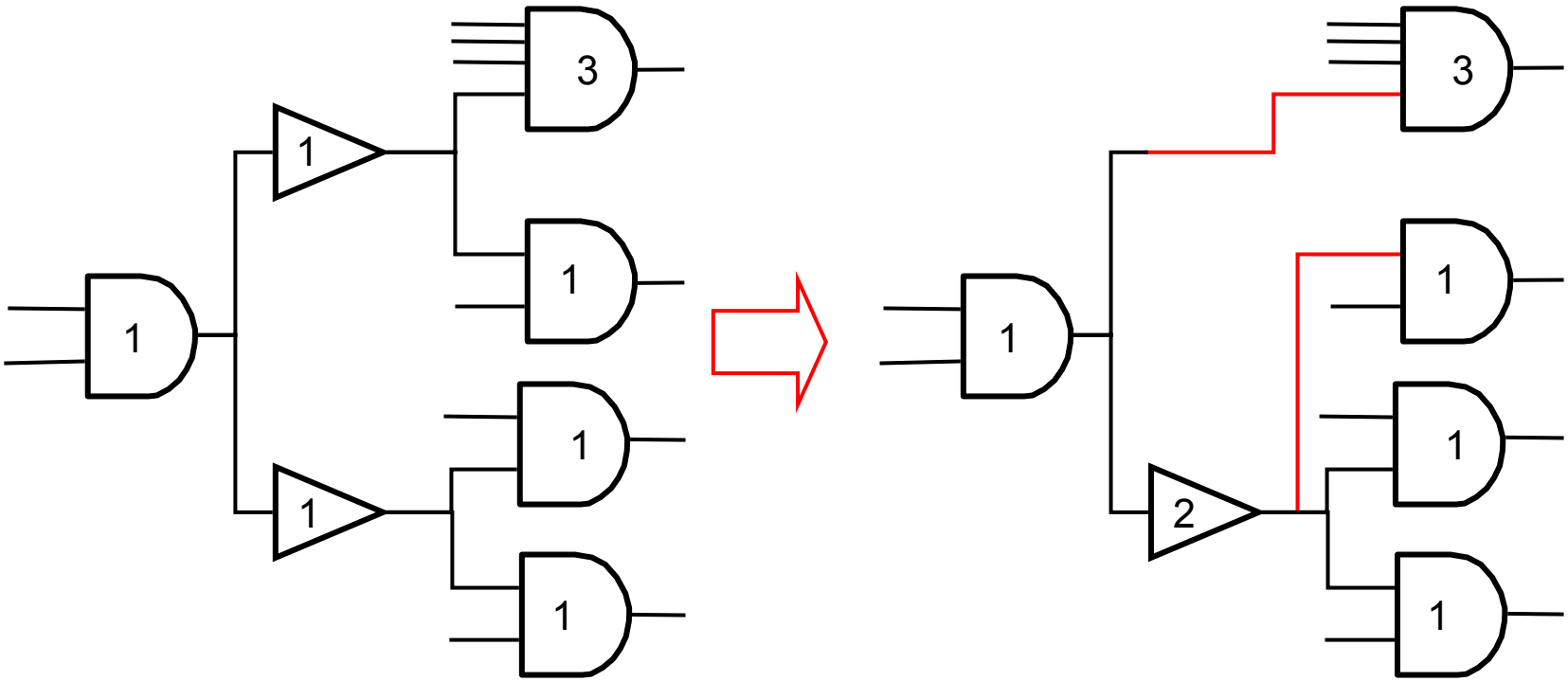

0.05
0.04
0.03
0.02
0.01
0
d
0
0.2
0.4
0.6
0.8
1
load
A
B
C


a
b
?
d
e
f
g
h
0.2
0.2
0.2
0.2
0.2
a
b
B
B
0.1
d
e
f
g
h
0.2
0.2
0.2
0.2
0.2

Redesign Fan-in Tree



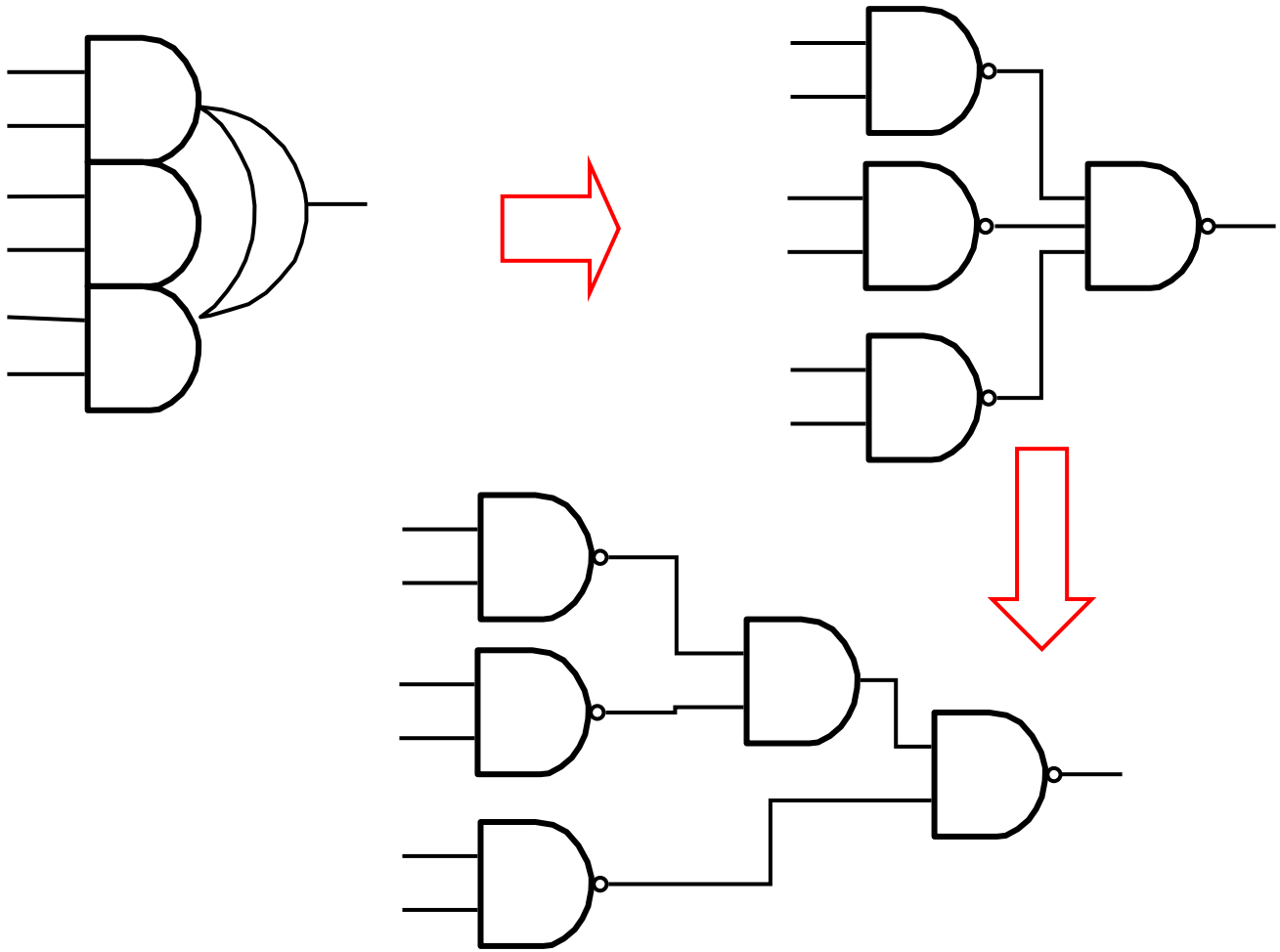
Redesign Fan-out Tree



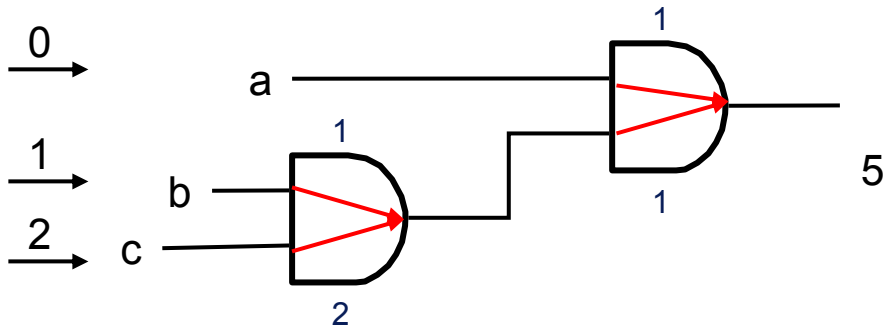
Longest Path = 5

Longest Path = 4
Slowdown of buffer due to load

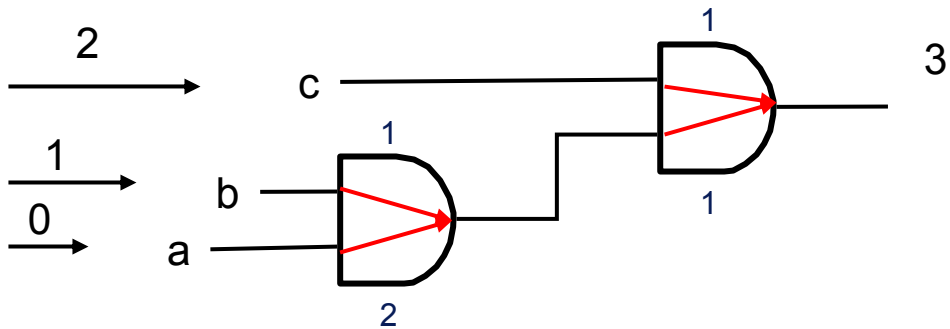
Decomposition



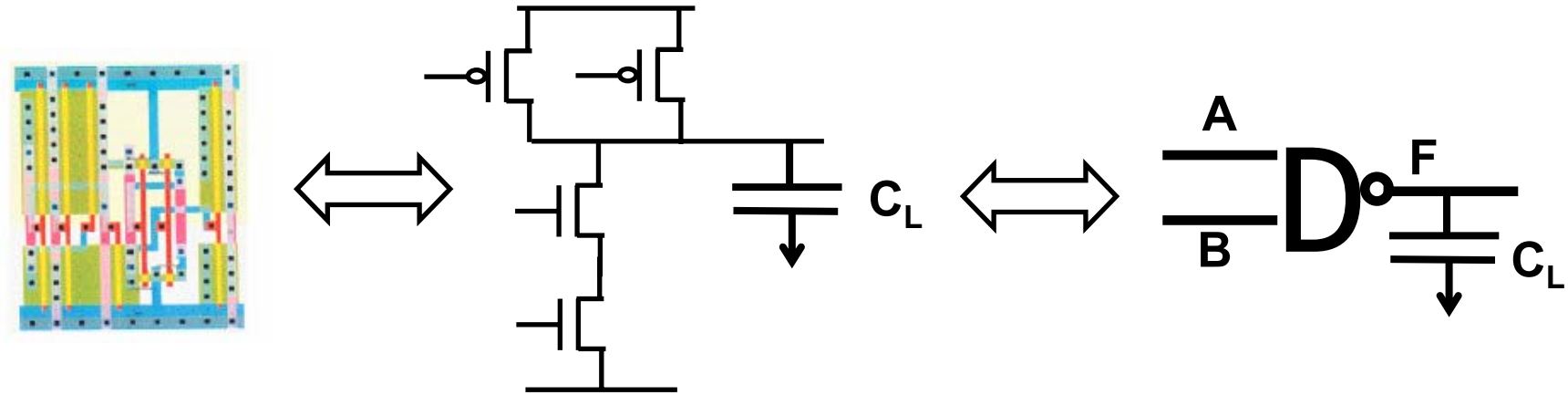
Swap Commutative Pins



Simple sorting on arrival times and delay works



Gate Timing Characterization



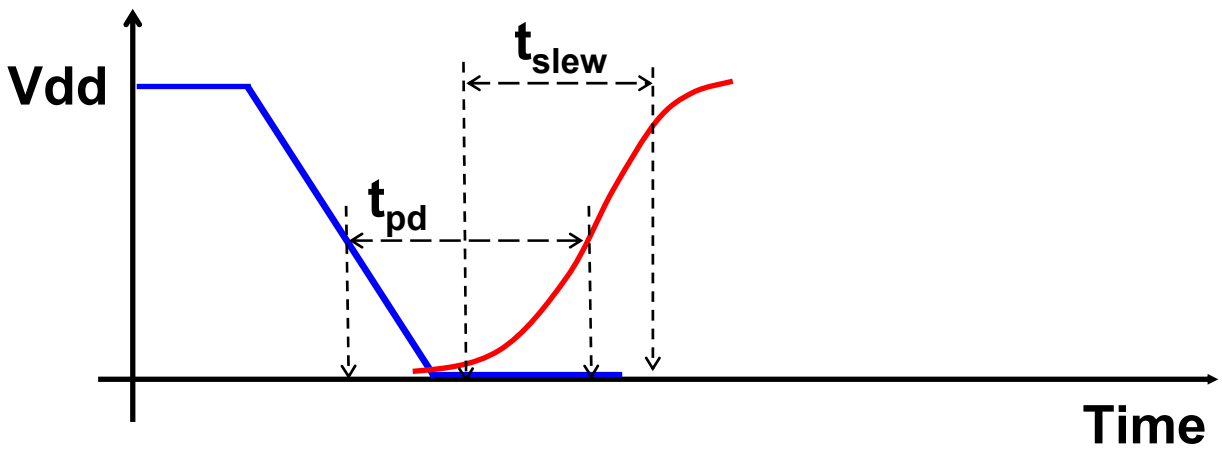
- "Extract" exact transistor characteristics from layout
  - Transistor width, length, junction area and perimeter
  - Local wire length and inter-wire distance

- Compute all transistor and wire capacitances

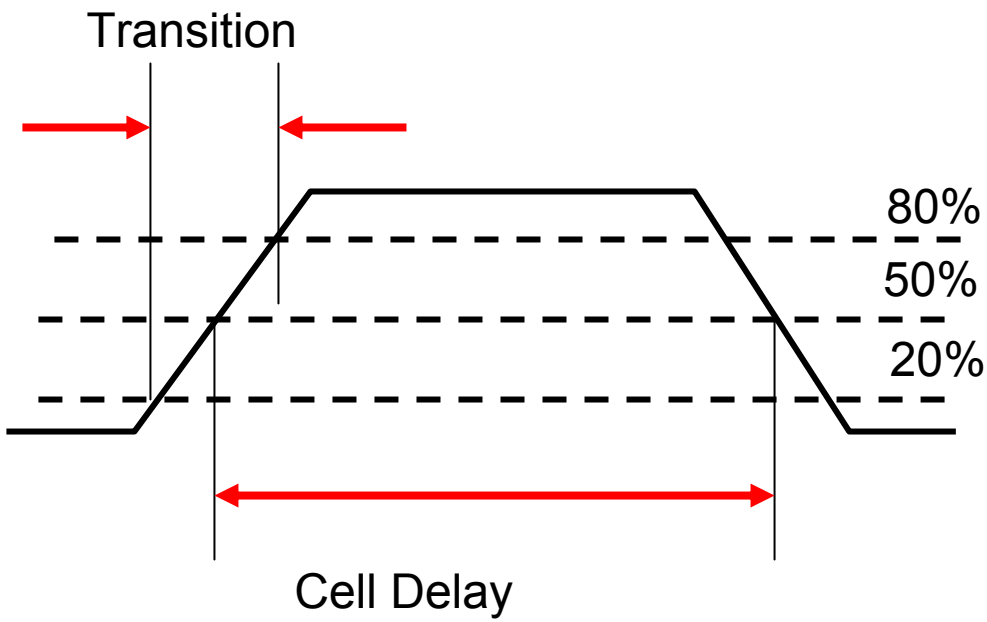
Cell Timing Characterization

- Delay tables generated using a detailed transistor-level circuit simulator SPICE (differential-equations solver)

- For a number of different input slews and load capacitances simulate the circuit of the cell
  - Propagation time (50% Vdd at input to 50% at output)
  - Output slew (10% Vdd at output to 90% Vdd at output)

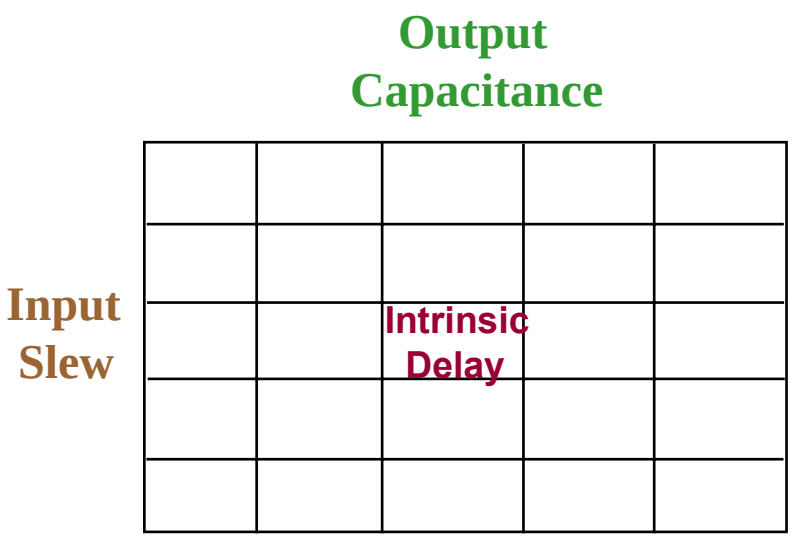


Delay and Transition Measurement

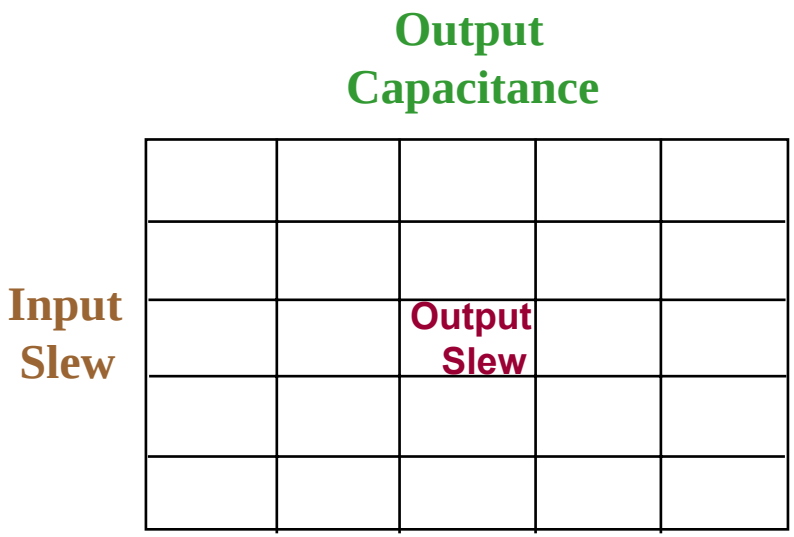


Non-linear effects reflected in tables

- $D_G = f(C_L, S_{in})$ and $S_{out} = f(C_L, S_{in})$
  - Non-linear
- Interpolate between table entries
- Interpolation error is usually below 10% of SPICE



Delay at the gate



Resulting waveform

Timing Library Example (.lib)

```
library(my_lib) {
  delay_model : table_lookup;
  library_features (report_delay_calculation);
  time_unit : "1ns";
  voltage_unit : "1V";
  current_unit : "1mA";
  leakage_power_unit : 1uW;
  capacitive_load_unit(1,pf);
  pulling_resistance_unit : "1kohm";
  default_fanout_load : 1.0;
  default_inout_pin_cap : 1.0;
  default_input_pin_cap : 1.0;
  default_output_pin_cap : 0.0;
  default_cell_leakage_power : 0.0;

  nom_voltage : 1.08;
  nom_temperature : 125.0;
  nom_process : 1.0;
  slew_derate_from_library : 0.500000;

 operating_conditions("slow_125_1.08") {
  process     : 1.0 ;
  temperature : 125 ;
  voltage     : 1.08 ;
  tree_type   : "worst_case_tree" ;
}
default_operating_conditions : slow_125_1.08 ;

lu_table_template("load") {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  index_1( "1, 2, 3, 4" );
  index_2( "1, 2, 3, 4" );
}
```

```
cell("INV") {
pin(A) {
    max_transition : 1.500000;
    direction : input;
    rise_capacitance : 0.0739000;
    fall_capacitance : 0.0703340;
    capacitance : 0.07278646;
  }
  pin(Z) {
    direction : output;
    function : "!A";
    max_transition : 1.500000;
    max_capacitance : 5.1139;
    timing() {
      related_pin : "A";
      cell_rise(load) {
        index_1( "0.0375, 0.2329, 0.6904, 1.5008" );
        index_2( "0.0010, 0.9788, 2.2820, 5.1139" );
        values ( \
          "0.013211, 0.071051, 0.297500, 0.642340", \
          "0.028657, 0.110849, 0.362620, 0.707070", \
          "0.053289, 0.165930, 0.496550, 0.860400", \
          "0.091041, 0.234440, 0.661840, 1.091700" );
      }
      cell_fall(load) {
        index_1( "0.0326, 0.1614, 0.5432, 1.5017" );
        index_2( "0.0010, 0.4249, 3.6538, 8.1881" );
        values ( \
          "0.009472, 0.072284, 0.317370, 0.688390", \
          "0.009992, 0.095862, 0.360530,  0.731610", \
          "0.009994, 0.126620, 0.477260,  0.867670", \
          "0.009996, 0.144150, 0.644140,  1.127700" );
      }
```

```
fall_transition(load) {
        index_1( "0.0326, 0.1614, 0.4192, 1.5017" );
        index_2( "0.0010, 0.4249, 2.1491, 8.1881" );
        values ( \
          "0.011974, 0.071668, 0.317800, 1.189560", \
          "0.033212, 0.101182, 0.328540,  1.189562", \
          "0.059282, 0.155052, 0.389900, 1.202360", \
          "0.162830, 0.317380, 0.628160, 1.441260" );
      }
      rise_transition(load) {
        index_1( "0.0375, 0.1650, 0.5455, 1.5078" );
        index_2( "0.0010, 0.4449, 1.7753, 5.1139" );
        values ( \
          "0.016690, 0.115702, 0.418200, 1.189060", \
          "0.038256, 0.139336, 0.422960,  1.189081", \
          "0.076248, 0.213280, 0.491820, 1.203700", \
          "0.170992, 0.353120, 0.694740, 1.384760" );
      }
    }
```

Delay Calculation

Cell Fall

Cap\Tr	0.05	0.2	0.5
0.01	0.02	0.16	0.30
0.5	0.04	0.32	0.60
2.0	0.08	0.64	1.20

0.178

Cell Rise

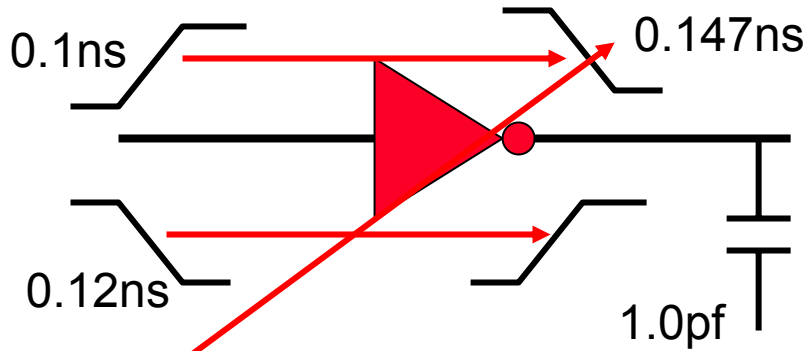
Cap\Tr	0.05	0.2	0.5
0.01	0.03	0.18	0.33
0.5	0.06	0.36	0.66
2.0	0.09	0.72	1.32

0.261

Fall Transition

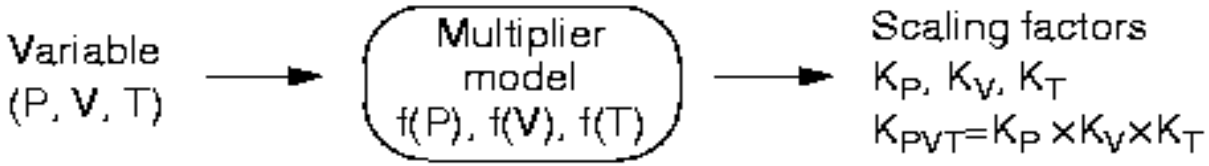
Cap\Tr	0.05	0.2	0.5
0.01	0.01	0.09	0.15
0.5	0.03	0.27	0.45
2.0	0.06	0.54	0.90

0.147



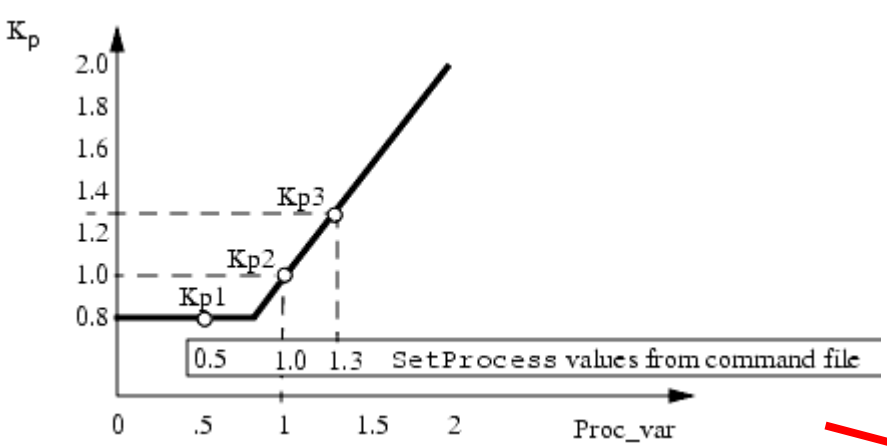
Fall delay = 0.178ns
Rise delay = 0.261ns
Fall transition = 0.147ns
Rise transition = …

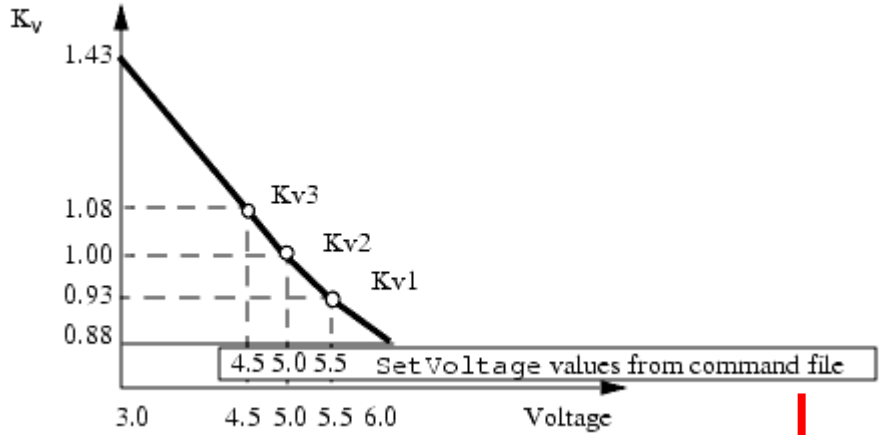
PVT (Process, Voltage, Temperature) Derating

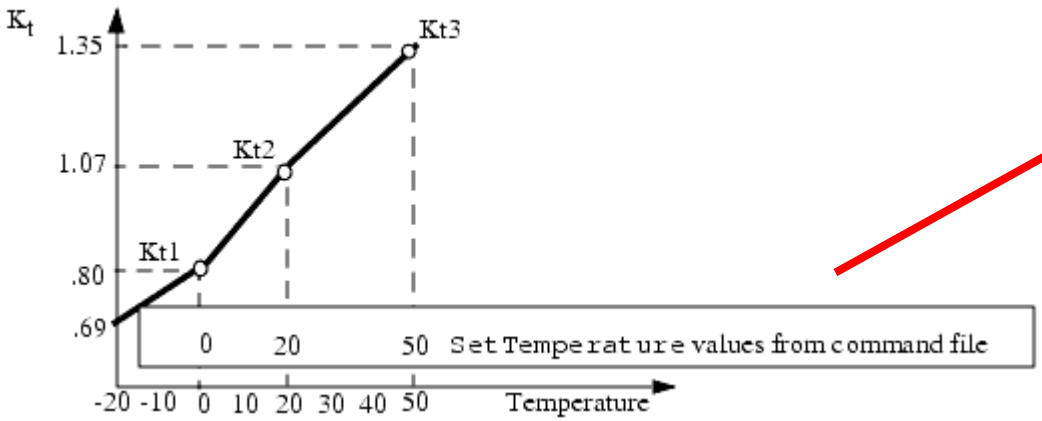


Actual cell delay = Original delay x K_{PVT}

PVT Derating: Example + Min/Typ/Max Triples







Proc_var (0.5:1.0:1.3)
Voltage (5.5:5.0:4.5)
Temperature (0:20:50)

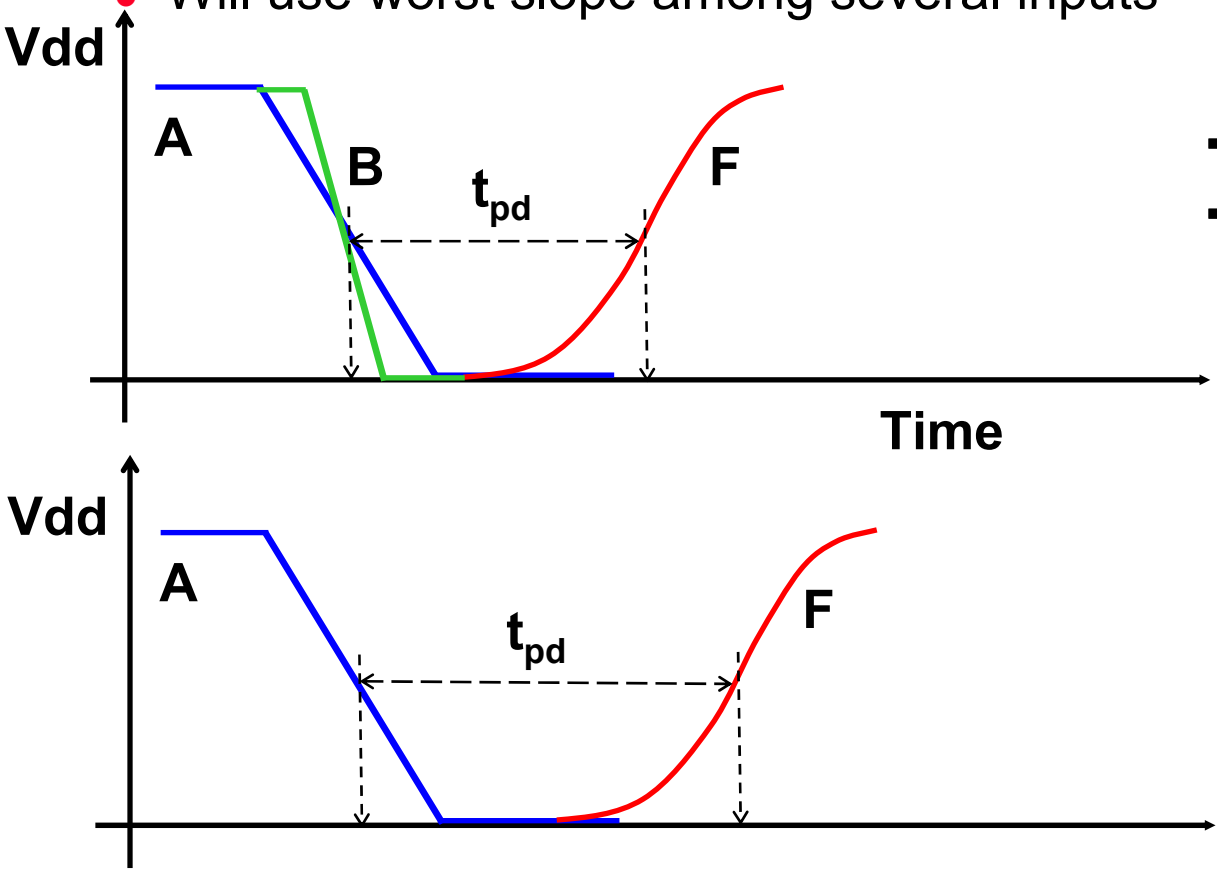
K_P = 0.80 : 1.00 : 1.30
K_V = 0.93 : 1.00 : 1.08
K_T = 0.80 : 1.07 : 1.35

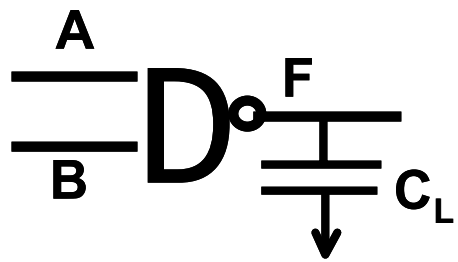
K_{PVT} = 0.60 : 1.07 : 1.90

Cell delay = 0.261ns
Derated delay = 0.157 : 0.279 : 0.496 {min : typical : max}

Conservatism of Gate Delay Modeling

- True gate delay depends on input arrival time patterns
  - STA will assume that only 1 input is switching
  - Will use worst slope among several inputs





Formats: Synopsys .LIB

- Synopsys supports many Modeling Options
  - CMOS Generic Delay Model
  - CMOS Nonlinear Delay Model
  - CMOS Piecewise Linear Delay Model
  - CMOS2 Delay Model
  - DCM Delay Mode (IEEE Std.)
- Interconnect Models in .LIB
  - Best Case (Near End)
  - Worst Case (Far End)
  - Balanced Case (Distributed)

Formats: Synopsys .LIB (2)

- Synopsys General Delay Equation
  - $D_{total} = D_I + D_S + D_C + D_T$
  - D_I → Intrinsic Gate Delay
  - D_S → Slope Delay due to ramp time of input signal
  - D_C → Connection Delay due to media (wire delay)
  - D_T → Transition delay sue to loading of output
- All Synopsys Models (besides DCM) are based upon this general equation
  - Generic Model
  - NLDM (Non-Linear)

Formats: Synopsys .LIB (3)

- CMOS Generic Delay Model
  - $D_S = S_S \times D_{T(prevstage)}$
    - Slope sensitivity (S_S) → `slope_rise, slope_fall`
  - $D_T = R_{driver}(C_{wire}+C_{pins})$
    - R_{driver} → `rise_resistance, fall_resistance` parameters
    - C_{wire} → Function of wire model used
    - C_{pins} → `capacitance` parameter of fanout gates
  - $D_C = R_{wire}(C_{wire}+C_{pins})$
    - 0 for BC, above for WC, and 1/N for Balanced Case
  - D_I
    - Delay of unloaded gate with step input, due to internal capacitances and parasitics

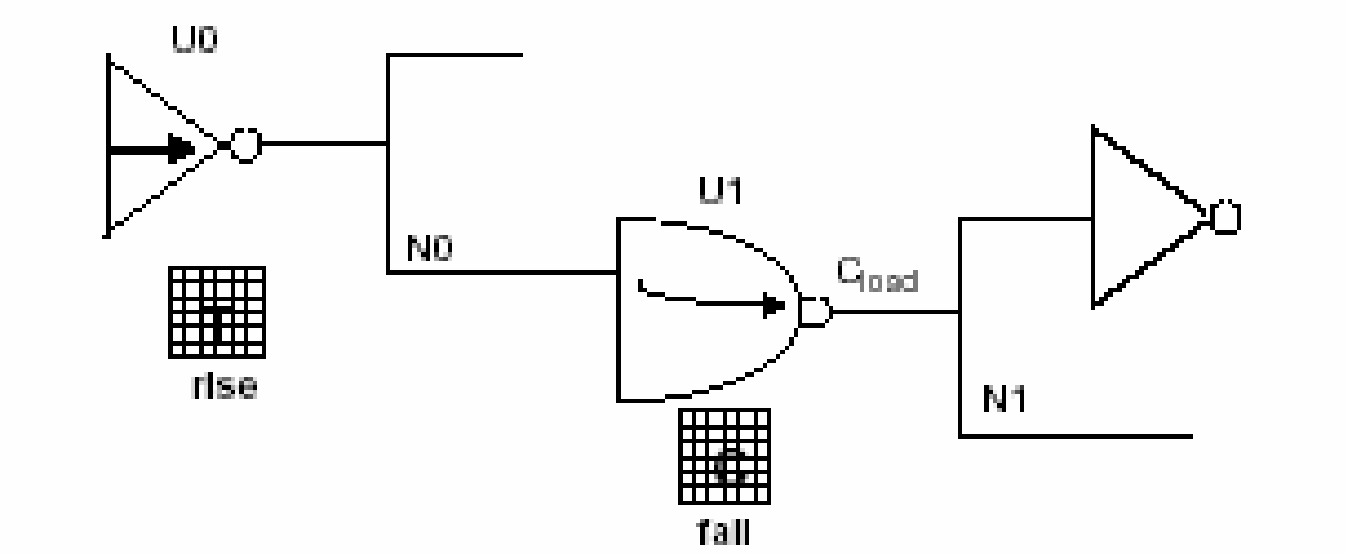
Formats: Synopsys .LIB (4)

- CMOS Generic Delay Model

```
cell (inv1) {
   area : 1 ;
   pin (a) {
      direction : input ;
      capacitance : 1 ;
   }
   pin (b) {
      direction : output ;
      capacitance : 0 ;
      timing () {
        related_pin : ”a” ;
        rise_resistance : 1.2 ;
        fall_resistance : 0.8 ;
        intrinsic_rise : 1.5 ;
        intrinsic_fall : 1.2 ;
      }
   }
}
```

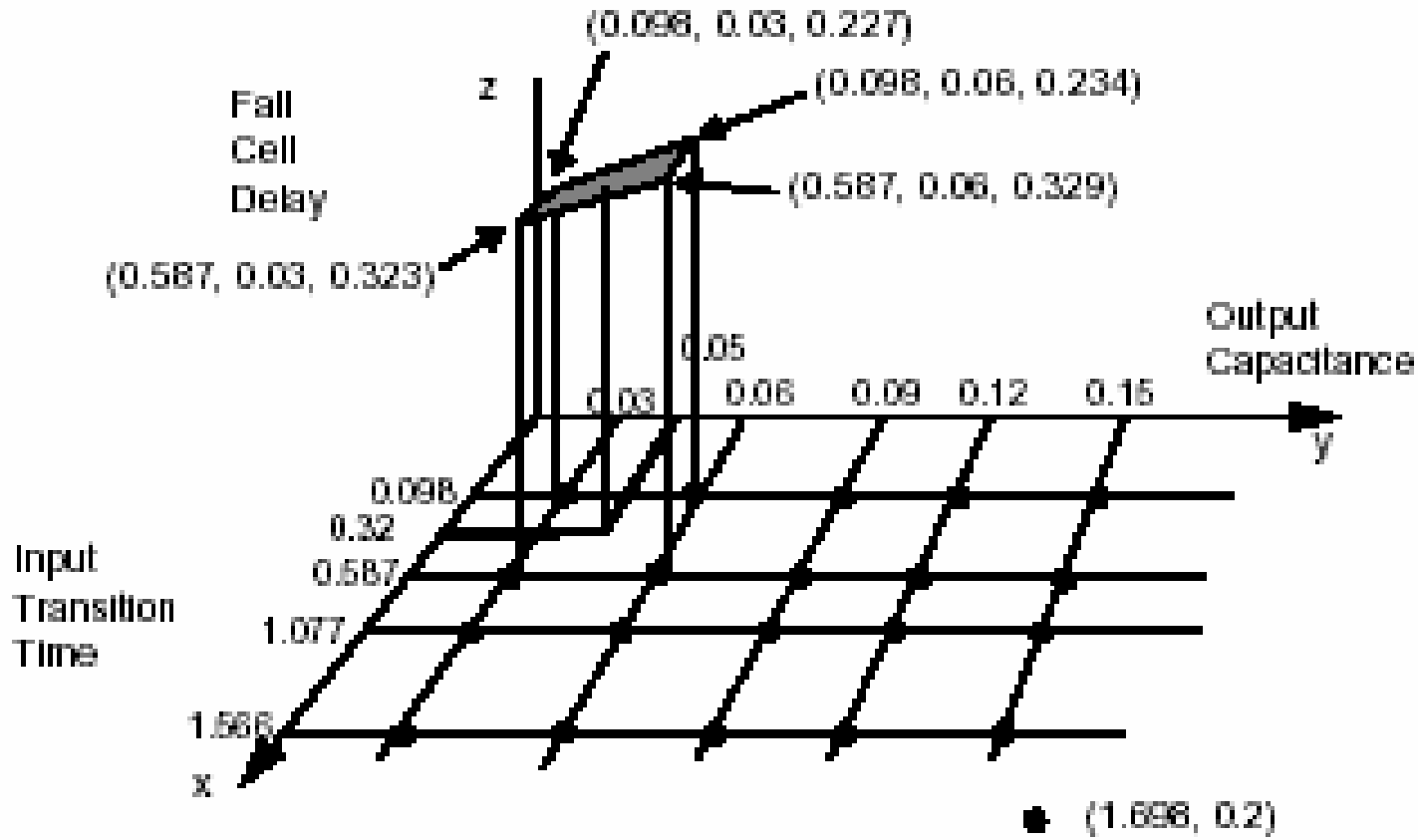
Formats: Synopsys .LIB (5)

- CMOS Non-Linear Delay Model
  - Default Equation → $D_{total} = D_{cell} + D_C$
    - D_{cell} accounts for input slew, output load, and intrinsic delay
  - D_{cell} is typically implemented as a 2-D lookup table



Formats: Synopsys .LIB (6)

- CMOS Non-Linear Delay Model (interpolation)



Formats: Synopsys .LIB (7)

- CMOS Non-Linear Delay Model

```
pin(I) {
  direction : input;
  capacitance : 0.04204 ;
  fanout_load : 0.04204 ;
   }
 pin(OEN) {
  direction : input;
  capacitance : 0.04206 ;
  fanout_load : 0.04206 ;
   }
 pin(PAD) {
  direction : inout;
  is_pad : true;
  output_voltage : ttl;
  input_voltage : ttl;
  drive_current : 24;
  function : "I";
  three_state : "OEN";
 timing() {
   timing_sense : "positive_unate"
   related_pin : "I"
   cell_rise( tpz013g2_IO5x6d3 ) {
    values("1.8940, 2.2490, 2.5920, 2.9240, 3.4110, 4.2800", \
           "1.9050, 2.2610, 2.6040, 2.9370, 3.4230, 4.2920", \
           "1.9190, 2.2760, 2.6200, 2.9530, 3.4400, 4.3090", \
           "1.9250, 2.2830, 2.6270, 2.9600, 3.4470, 4.3170", \
           "1.9360, 2.2950, 2.6400, 2.9740, 3.4620, 4.3330");
   }
   rise_transition( tpz013g2_IO5x6d3 ) {
    values("1.7552, 2.3620, 2.9740, 3.5900, 4.5190, 6.2390", \
           "1.7540, 2.3620, 2.9740, 3.5900, 4.5190, 6.2380", \
           "1.7510, 2.3600, 2.9730, 3.5890, 4.5190, 6.2390", \
           "1.7500, 2.3590, 2.9720, 3.5880, 4.5190, 6.2390", \
           "1.7440, 2.3560, 2.9700, 3.5870, 4.5180, 6.2400");
   }
```

```
cell_fall( tpz013g2_IO5x6d3 ) {
    values("1.5620, 1.9160, 2.2470, 2.5640, 3.0230, 3.8420", \
           "1.5840, 1.9370, 2.2680, 2.5850, 3.0440, 3.8630", \
           "1.6220, 1.9760, 2.3070, 2.6240, 3.0840, 3.9020", \
           "1.6410, 1.9940, 2.3250, 2.6420, 3.1020, 3.9200", \
           "1.7250, 2.0790, 2.4100, 2.7270, 3.1860, 4.0050");
   }
   fall_transition( tpz013g2_IO5x6d3 ) {
    values("1.9212, 2.4387, 2.9812, 3.5435, 4.4080, 6.0250", \
           "1.9213, 2.4390, 2.9815, 3.5438, 4.4090, 6.0250", \
           "1.9229, 2.4392, 2.9817, 3.5440, 4.4090, 6.0250", \
           "1.9222, 2.4388, 2.9812, 3.5450, 4.4080, 6.0250", \
           "1.9180, 2.4354, 2.9780, 3.5410, 4.4060, 6.0220");
   }
    }
 timing() {
   timing_type : three_state_enable ;
   timing_sense : "negative_unate"
   related_pin : "OEN"
   cell_rise( tpz013g2_IO5x6d3 ) {
    values("1.7830, 2.1910, 2.5600, 2.9090, 3.4100, 4.2950", \
           "1.8060, 2.2130, 2.5820, 2.9310, 3.4320, 4.3170", \
           "1.8280, 2.2360, 2.6050, 2.9540, 3.4550, 4.3400", \
           "1.8440, 2.2510, 2.6200, 2.9690, 3.4710, 4.3550", \
           "1.8950, 2.3020, 2.6720, 3.0200, 3.5220, 4.4060");
   }
   rise_transition( tpz013g2_IO5x6d3 ) {
    values("1.6971, 2.3480, 2.9790, 3.6040, 4.5380, 6.2540", \
           "1.6970, 2.3480, 2.9790, 3.6040, 4.5380, 6.2550", \
           "1.6960, 2.3480, 2.9790, 3.6040, 4.5380, 6.2550", \
           "1.6970, 2.3470, 2.9790, 3.6040, 4.5390, 6.2550", \
           "1.6970, 2.3470, 2.9790, 3.6040, 4.5380, 6.2550");
   }
```

Formats: SDF Files

- Typically used to back-annotate functional simulations to include post-PD delay calculation
  - IEEE Standard Format
- General Format
- Many parameters (~100 page document)
  - Delay of Cells
  - Delay of Interconnect

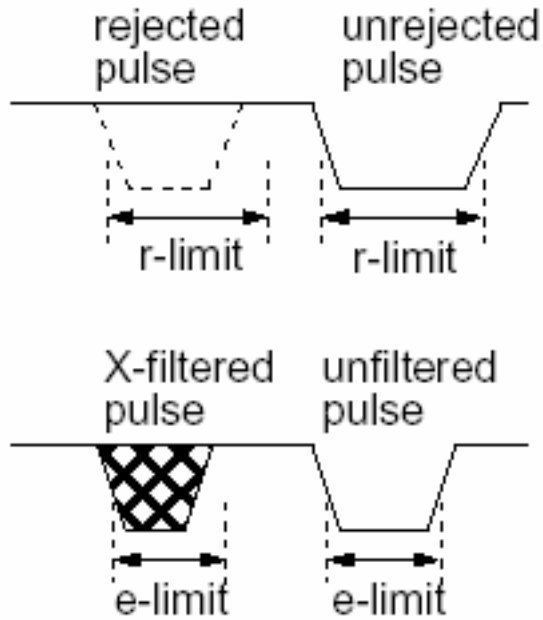
Formats: SDF Files (2)

```
(DELAYFILE
(SDFVERSION "OVI 1.0")
(DESIGN "EQ_ENTRY")
(DATE "Tue Apr 23 19:55:18 2002")
(VENDOR "typical")
(PROGRAM "Synopsys Design Compiler cmos")
(VERSION "2000.11-SP1")
(DIVIDER /)
(VOLTAGE 1.80:1.80:1.80)
(PROCESS)
(TEMPERATURE 25.00:25.00:25.00)
(TIMESCALE 1ns)
(CELL
  (CELLTYPE "EQ_ENTRY")
  (INSTANCE)
  (DELAY
    (ABSOLUTE
    (INTERCONNECT EQ_Aldst_offset_in[15] U1433/A0 (0.000:0.000:0.000))
    (INTERCONNECT U882/Y U1433/A1 (0.000:0.000:0.000))
    (INTERCONNECT EQ_srcB_reg\[0\]/Q U1433/B0 (0.000:0.000:0.000))
    (INTERCONNECT control_in[3] U1433/B1 (0.000:0.000:0.000))
```

Formats: SDF Files (3)

```
(INTERCONNECT EQ_Aldst_offset_in[15] U1433/A0 (0.000:0.000:0.000))
```

- INTERCONNECT
  - Specifies point to point interconnect delays
  - INTERCONNECT Src Dest (delval tokens)
- Delay Value Tokens
  - (MIN:TYP:MAX)
  - Can be expressed with r-limits and e-limits for more accuracy



Formats: SDF Files (4)

```
(CELL
  (CELLTYPE "OAI211X4")
  (INSTANCE U1354)
  (DELAY
    (ABSOLUTE
    (IOPATH A0 Y (0.262:0.262:0.262) (0.174:0.174:0.174))
    (IOPATH A1 Y (0.370:0.370:0.370) (0.170:0.170:0.170))
    (IOPATH B0 Y (0.168:0.250:0.250) (0.172:0.185:0.185))
    (IOPATH C0 Y (0.187:0.289:0.289) (0.193:0.202:0.202))
    )
  )
)
(CELL
  (CELLTYPE "OAI211X4")
  (INSTANCE U1353)
  (DELAY
    (ABSOLUTE
    (IOPATH A0 Y (0.262:0.262:0.262) (0.174:0.174:0.174))
    (IOPATH A1 Y (0.370:0.370:0.370) (0.170:0.170:0.170))
    (IOPATH B0 Y (0.168:0.250:0.250) (0.172:0.185:0.185))
    (IOPATH C0 Y (0.187:0.289:0.289) (0.193:0.202:0.202))
    )
  )
)
```

Formats: SDF Files (5)

```
(IOPATH A0 Y (0.262:0.262:0.262) (0.174:0.174:0.174))
```

- IOPATH
  - Specifies delay from input pin A0 to output Y
  - Contains 2,3,6 or 12 delay token values
  - (MIN:TYP:MAX)
  - Can also use r-limits and e-limits

Table 1—Deriving 12 delay values

	Number of *delval* tokens in *delval_list*		
Transition	**2**	**3**	**6**
0→1	0→1	0→1	0→1
1→0	1→0	1→0	1→0
0→Z	0→1	?→Z	0→Z
Z→1	0→1	0→1	Z→1
1→Z	1→0	?→Z	1→Z
Z→0	1→0	1→0	Z→0
0→X	0→1	min(0→1, ?→Z)	min(0→1, 0→Z)
X→1	0→1	0→1	max(0→1, Z→1)
1→X	1→0	min(1→0, ?→Z)	min(1→0, 1→Z)
X→0	1→0	1→0	max(1→0, Z→0)
X→Z	max(0→1, 1→0)	?→Z	max(0→Z, 1→Z)
Z→X	min(0→1, 1→0)	min(0→1, 1→0)	min(Z→0, Z→1)