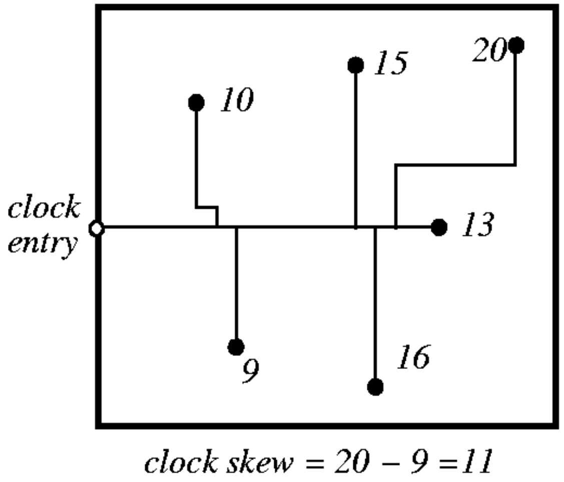
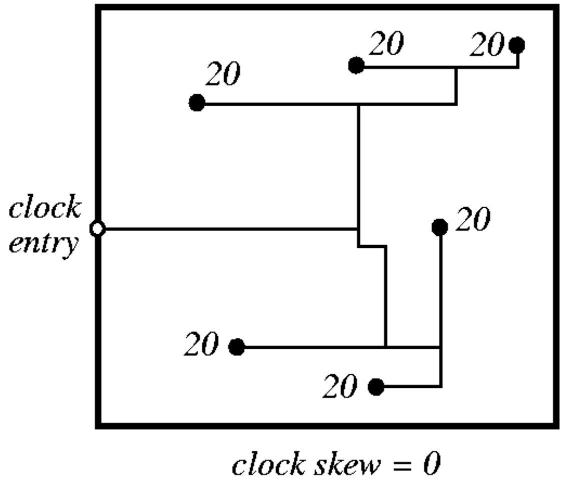
Clock Routing

- **Clock skew** is defined as the difference in the minimum and the maximum arrival time of the clock.





- Route clock nets such that
  - Clock signals arrive simultaneously
  - Clock delay is minimized

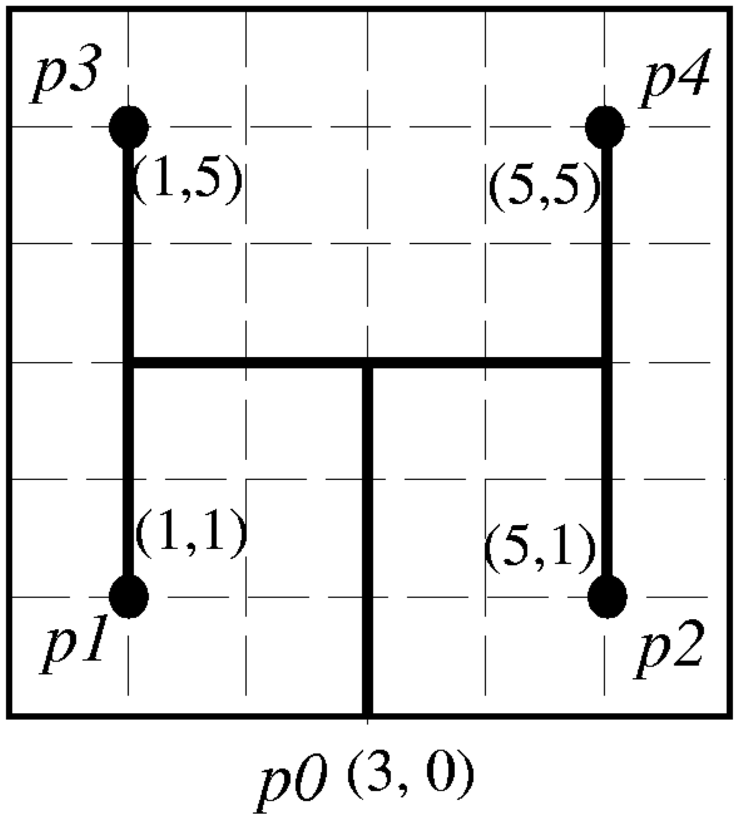
    * Other issues: total wirelength, power consumption.

Clock Routing Problem

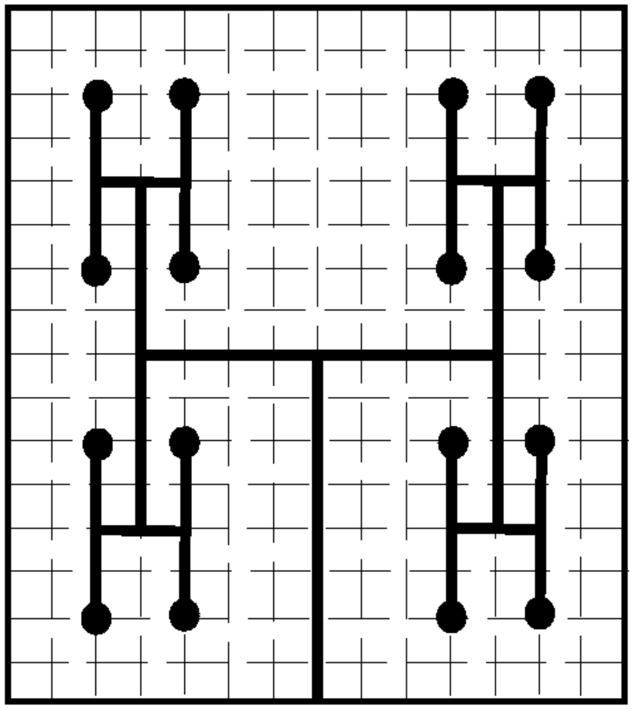
- Given the routing plane and a set of points $P=\{p_1,p_2,\ldots,p_n\}$ within the plane and clock entry point p_0 on the boundary of the plane, the **Clock Routing Problem (CRP)** is to interconnect each $p_i \in P$ such that $\max_{i,j\in P}|t(0,i)-t(0,j)|$ (skew) and $\max_{i\in P} t(0,i)$ (delay) are both minimized.
- Pathlength-based approaches
  - H-tree: Dhar, Franklin & Wang, *ICCD,* 1984.
  - Methods of means & medians (MMM): Jackson, Srinivasan & Kuh, *DAC*, 1990.
  - Geometric matching: Kahng, Cong & Robins, *IEEE TCAD*, 1993.
- RC-delay based approaches:
  - Exact zero skew: Tasy, *ICCAD*, 1991.
  - Deferred merge embedding (DME) algorithm: Boese & Kahng, ASICON-92; Chao, Hsu & Ho, DAC-92; Edahiro, NEC R&D, 1991.

H-Tree Based Algorithm

- H-tree: Dhar, Franklin, Wang, "Reduction of clock delays in VLSI structure," *ICCD*, 1984.



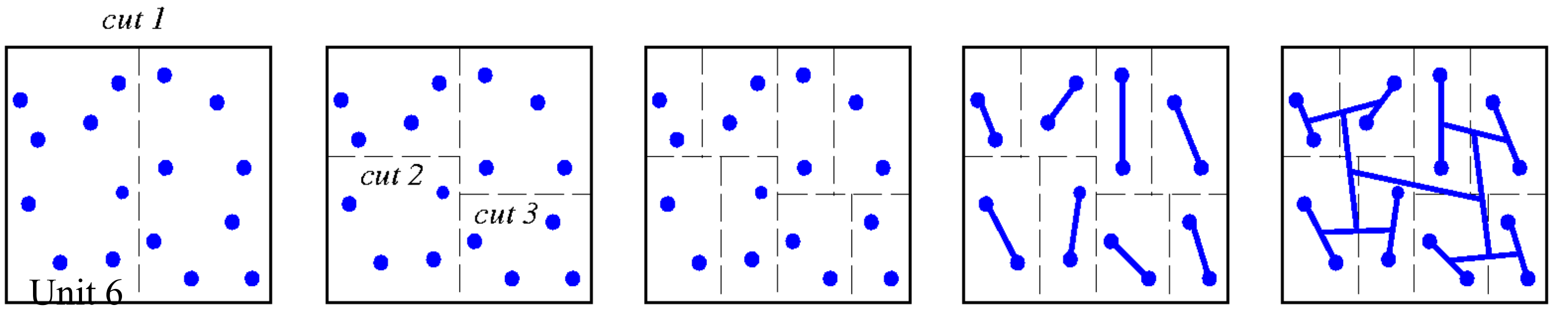
H−tree over 4 points



H−tree over 16 points

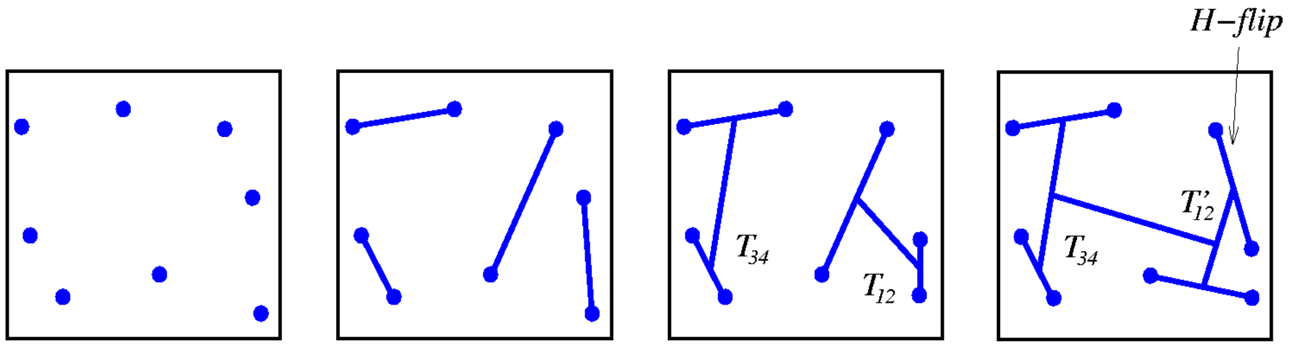
The MMM Algorithm

- Jackson, Srinivasan, Kuh, "Clock routing for high-performance ICs," *DAC*, 1990.
- Each clock pin is represented as a point in the region, S.
- The region is partitioned into two subregions, S_L and S_R.
- The center of mass is computed for each subregion.
- The center of mass of the region S is connected to each of the centers of mass of subregion S_L and S_R.
- The subregions S_L and S_R are then recursively split in Y-direction.
- The above steps are repeated with alternate splitting in X- and Y-direction.
- Time complexity: $O(n \log n)$.



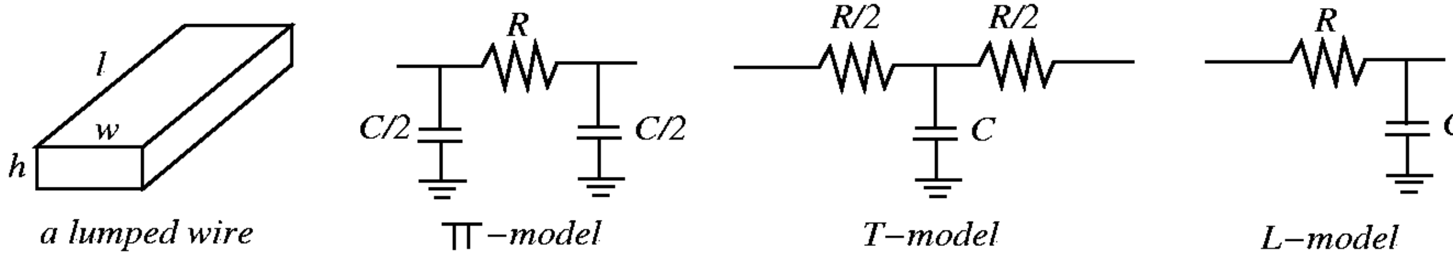
The Geometric Matching Algorithm

- Kahng, Cong, Robins, “Matching based models for high-performance clock routing,” *IEEE TCAD*, 1993.
- Clock pins are represented as n nodes in the clock tree (n=2^k).
- Each node is a tree itself with clock entry point being node itself.
- The minimum cost matching on n points yields n/2 segments.
- The clock entry point in each sebtree of two nodes is the point on the segment such that length of both sides is the same.
- The above steps are repeated for each segment.
- Apply H-flipping to further reduce clock skew (and to handle edges intersection).

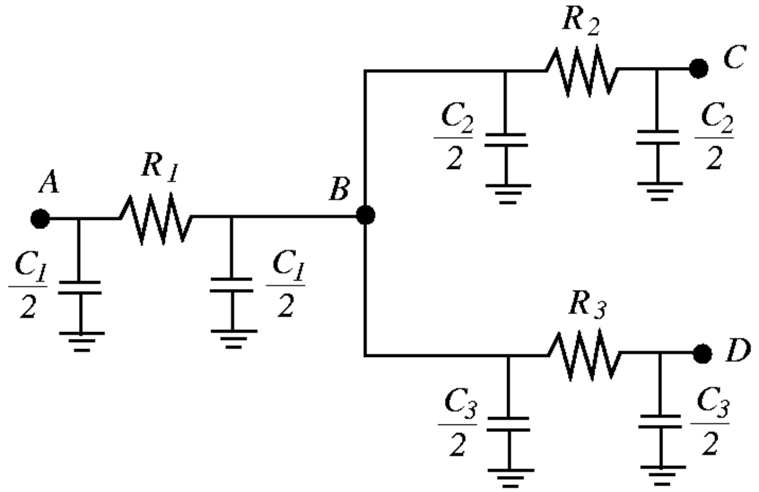


Delay Calculation

- Need to consider a more accurate delay model for general circuits.
- **RC Delay:** The delay caused by wires is due to their capacitance and resistance. $(R \propto \frac{l}{wh};\ C \propto wl)$
- Lumped circuit approximations for distributed RC lines: π-model, *T*-model, *L*-model.

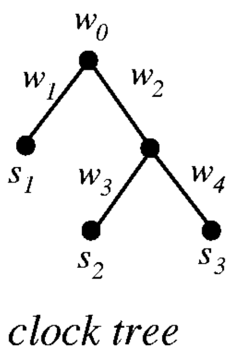


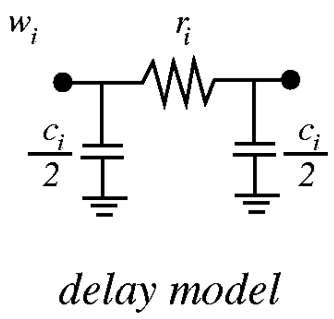
- RC delay: $D_{AB} = R_1(\frac{C_1}{2} + C_2 + C_3);\ B\ to\ C:\ D_{BC} = R_2(\frac{C_2}{2});\ B\ to\ D:\ D_{BD} = R_3(\frac{C_3}{2})$

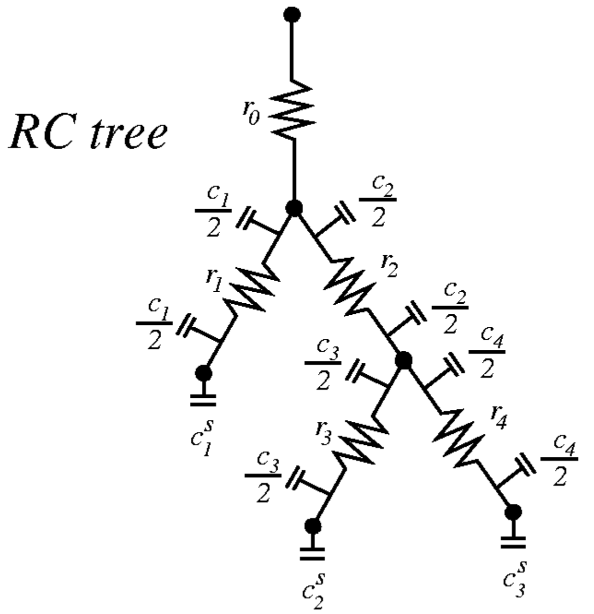


Delay Calculation for a Clock Tree

$$t_{03} = r_0(c_1 + c_2 + c_3 + c_4 + c_1^s + c_2^s + c_3^s) + r_2(\frac{c_2}{2} + c_3 + c_4 + c_2^s + c_3^s) + r_4(\frac{c_4}{2} + c_3^s).$$







Exact Zero Skew Algorithm

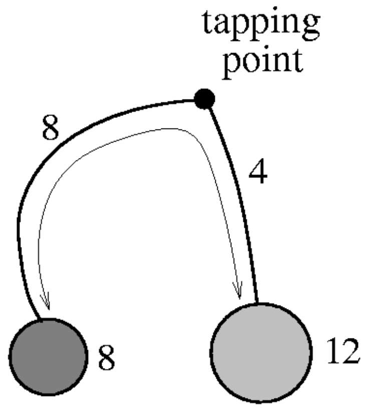
- Tsay, "Exact zero skew algorithm," *ICCAD*, 1991.
- To ensure the delay from the **tapping point** to leaf nodes of subtrees T_1 and T_2 being equal, it requires that

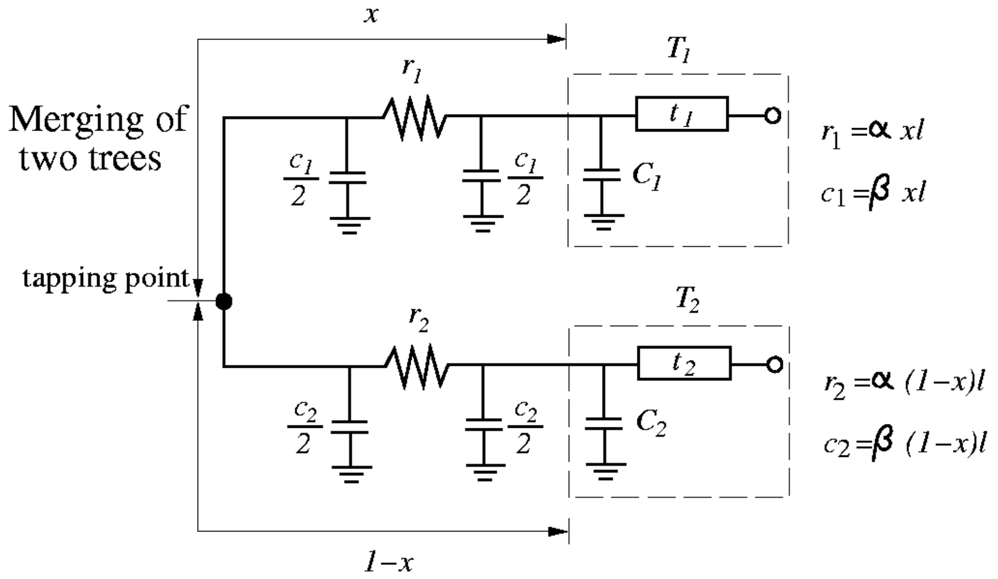
$$r_1(\frac{c_1}{2}+C_1)+t_1 = r_2(\frac{c_2}{2}+C_2)+t_2$$

- Solving the above equation, we have

$$x = \frac{(t_2-t_1)+\alpha l(C_2+\frac{\beta l}{2})}{\alpha l(\beta l + C_1 + C_2)}$$

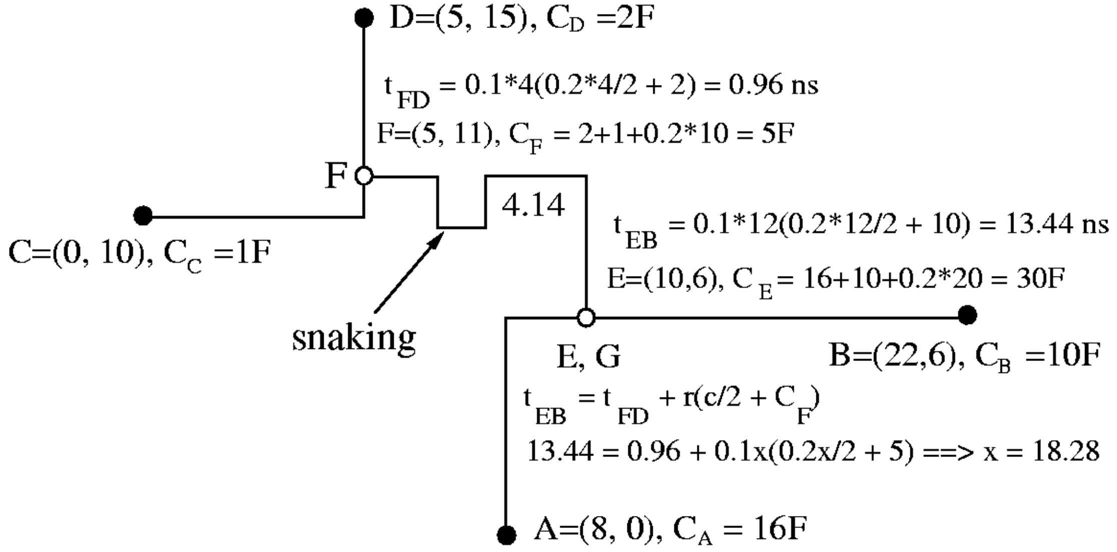
- α and β are the per unit values of resistance and capacitance, *l* the length of the interconnecting wire, $r_1=\alpha x l$, $c_1=\beta x l$, $r_2=\alpha(1-x)l$, $c_2=\beta(1-x)l$.





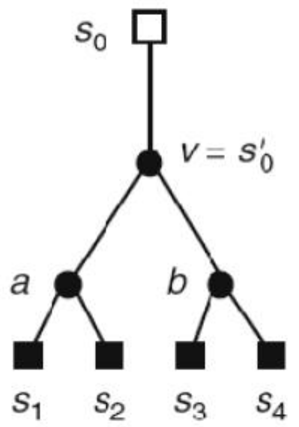
Zero-Skew Computation

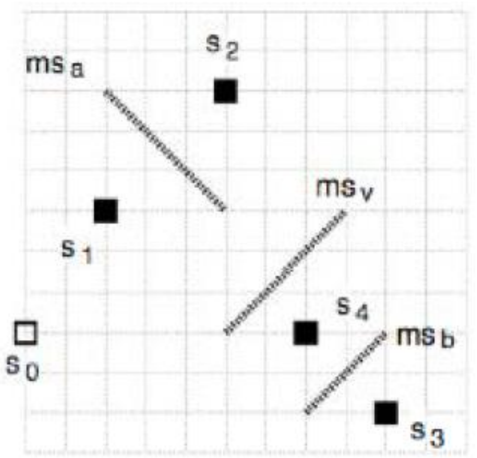
- **Balance delays:** $r_1(\frac{c_1}{2}+C_1)+t_1 = r_2(\frac{c_2}{2}+C_2)+t_2.$
- **Compute tapping points:** $x = \frac{(t_2-t_1)+\alpha l(C_2+\frac{\beta l}{2})}{\alpha l(\beta l + C_1 + C_2)}$, α(β): per unit values of resistance (capacitance); *l*: length of the wire; r_1=α*xl*, c_1=β*xl*; r_2 = α(1-*x*)*l*, c_2=β(1-*x*)*l*.
- If $x \notin [0,1]$, we need **snaking** to find the tapping point.
- Exp: α=0.1Ω/*unit*, β=0.2*F*/*unit*. (Find tapping points *E* for *A* and *B*, *F* for *C* and *D*, and *G* for *E* and *F*.)

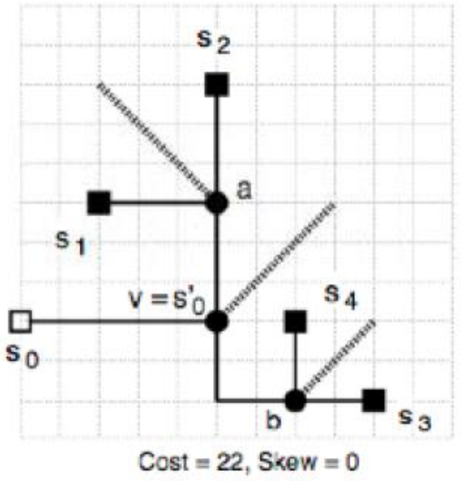

D=(5, 15), C_D =2F
t_FD = 0.1*4(0.2*4/2 + 2) = 0.96 ns
F=(5, 11), C_F = 2+1+0.2*10 = 5F
F
4.14
C=(0, 10), C_C =1F
snaking
t_EB = 0.1*12(0.2*12/2 + 10) = 13.44 ns
E=(10,6), C_E = 16+10+0.2*20 = 30F
E, G
B=(22,6), C_B =10F
t_EB = t_FD + r(c/2 + C_F)
13.44 = 0.96 + 0.1x(0.2x/2 + 5) ==> x = 18.28
A=(8, 0), C_A = 16F

Deferred Merge Embedding (DME)

- Boese & Kahng, ASICON-92; Chao, Hsu, & Ho, DAC-92; Edahiro, NEC R&D, 1991
- Two stages
  - Bottom up: build a **segment** for potential tapping points
  - Top down: determine exact tapping points

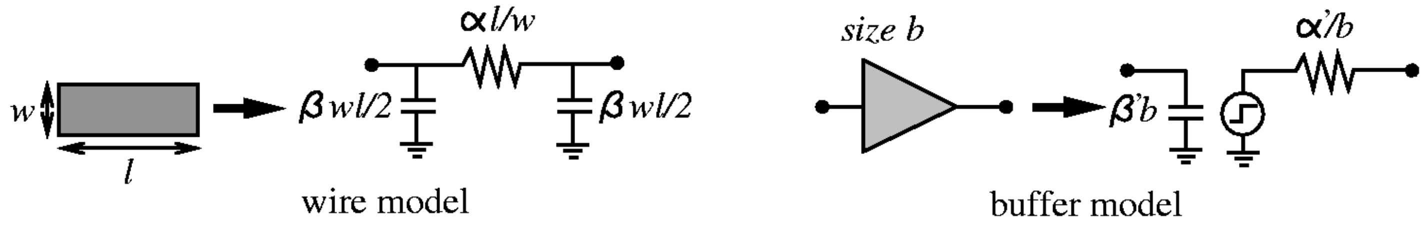


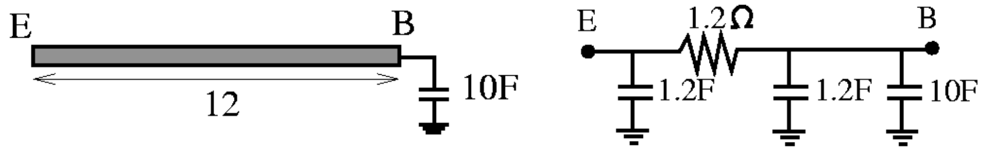




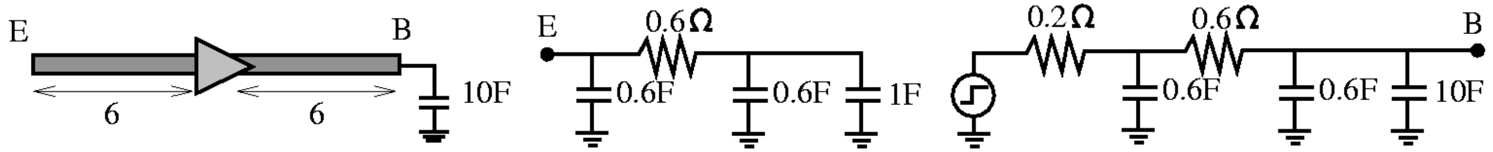
Delay Computation for Buffered Wires

- Wire: α=0.1Ω /*unit* size, β=0.2*F*/*unit* size; buffer: α'=0.2Ω/*unit* size, β'=1*F*/*unit* size; unit-sized wire and buffer.

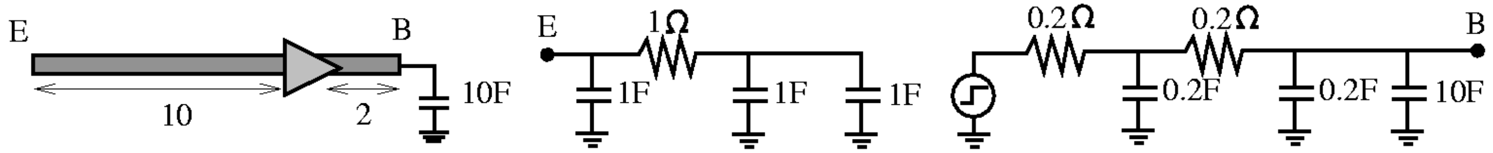




t_{EB} = 1.2 (1.2+10) = 13.44



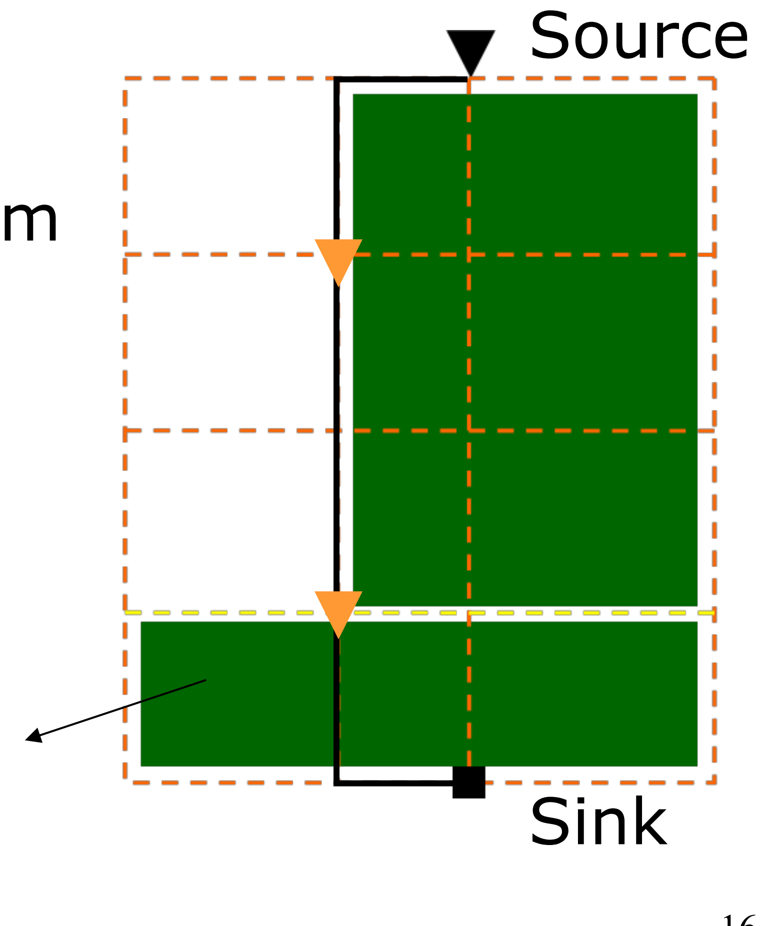
t_{EB} = 0.6(0.6+1) + 0.2(0.6+0.6+10) + 0.6(0.6+10) = 9.56



t_{EB} = 1(1+1) + 0.2(0.2+0.2+10) + 0.2(0.2+10) = 6.12

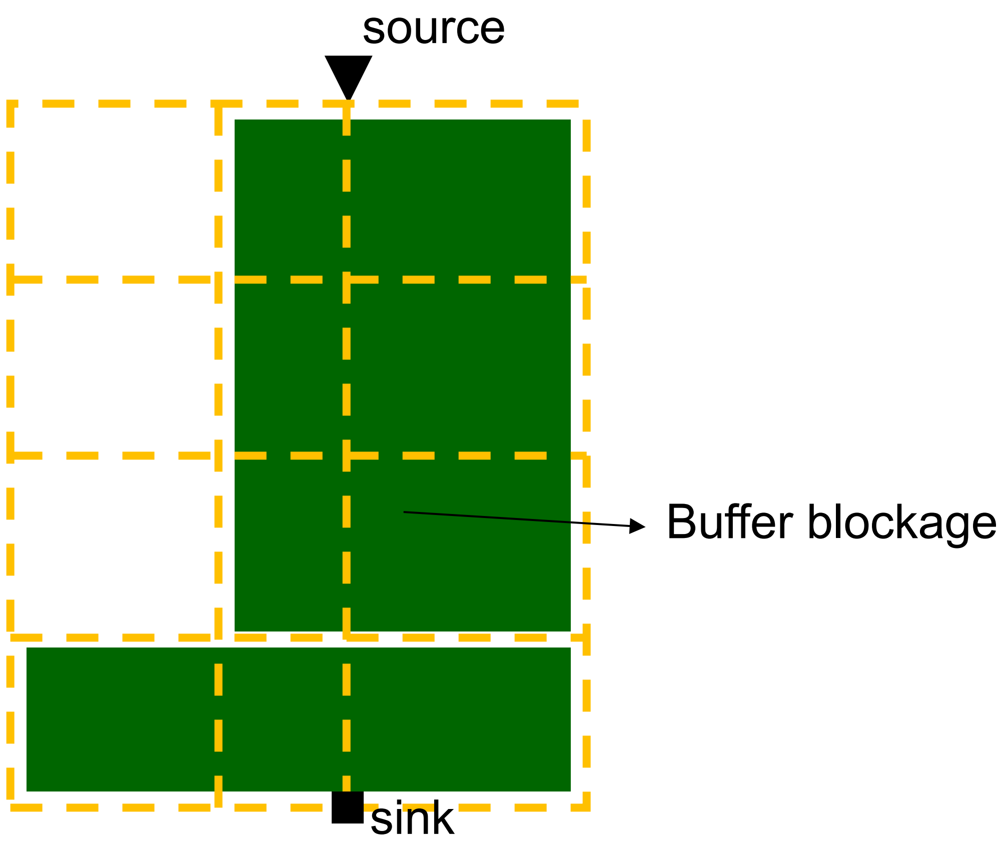
Buffered Path Construction

Goal: Find a minimum-delay buffered path from source to sink with blockage avoidance

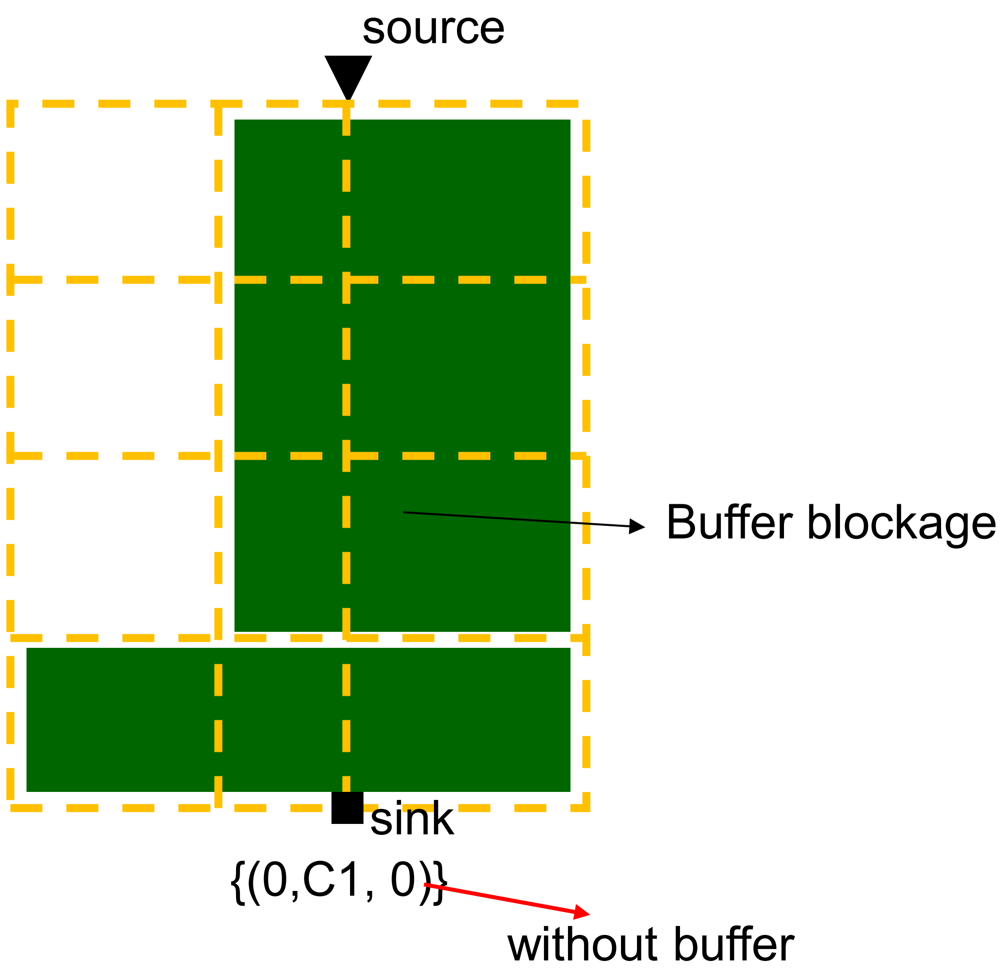


Fast Path Algorithm

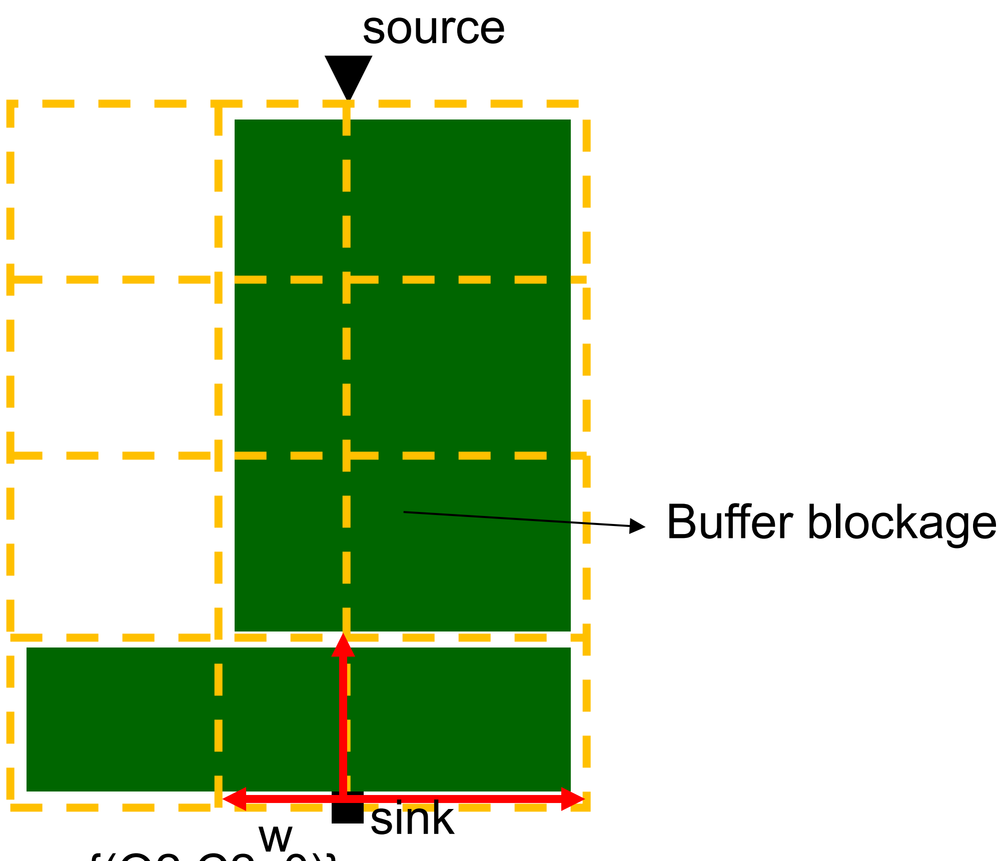
Proposed by Zhou *et al.*, DAC 1999



Fast Path Algorithm



Fast Path Algorithm



Q2= Q1+R (C/2 + C1)

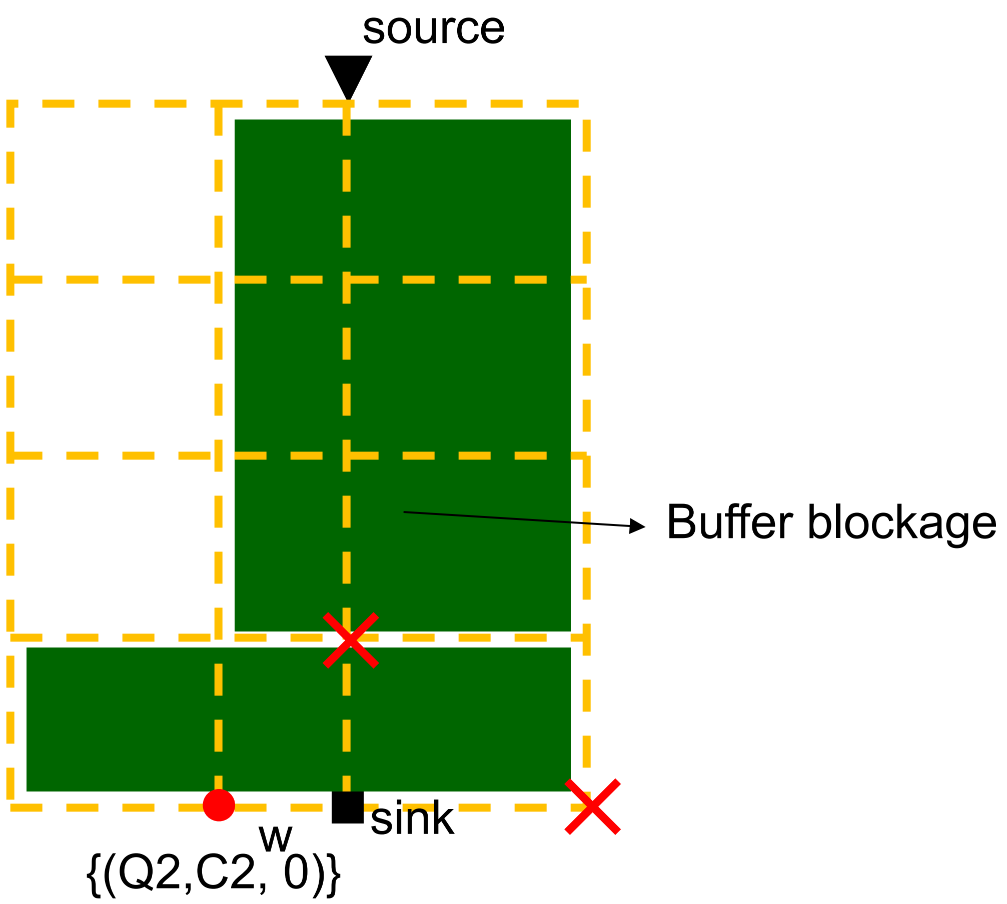
C2= C+C1

C is capacitance of wire w,

R is resistance of w

{(Q2,C2, 0)}

Fast Path Algorithm



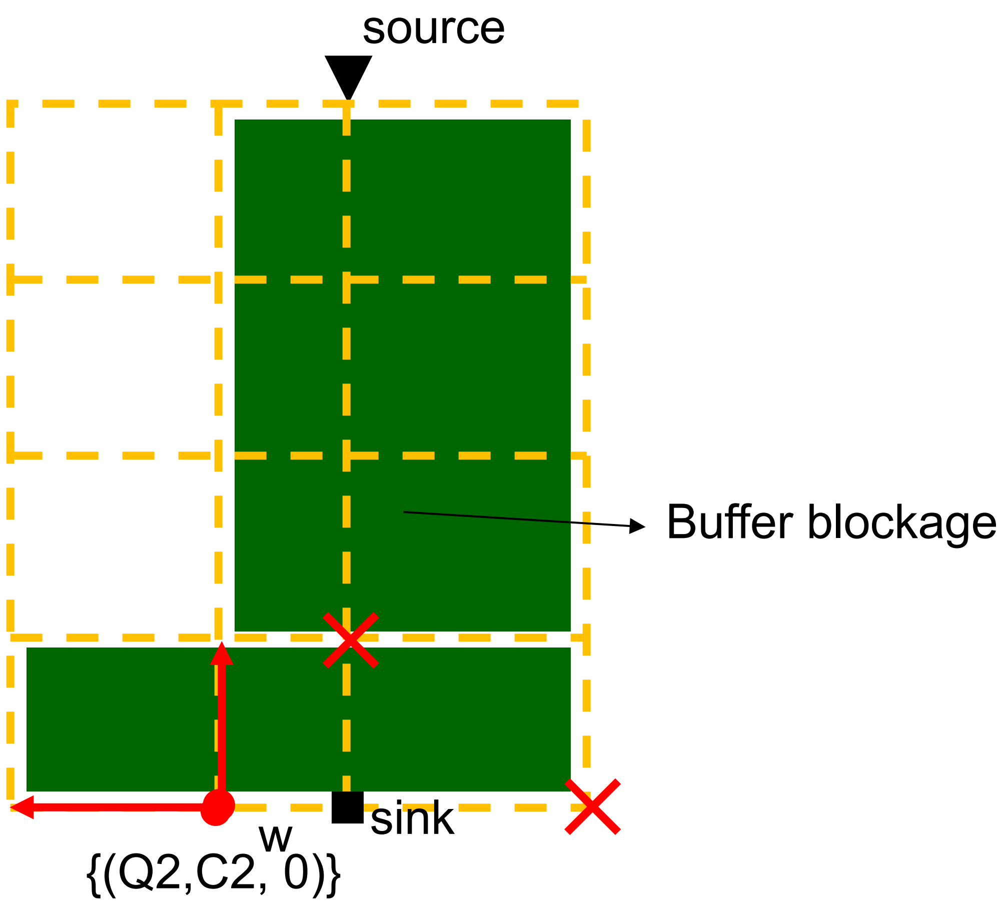
Q2= Q1+R (C/2 + C1)

C2= C+C1

C is capacitance of wire w,

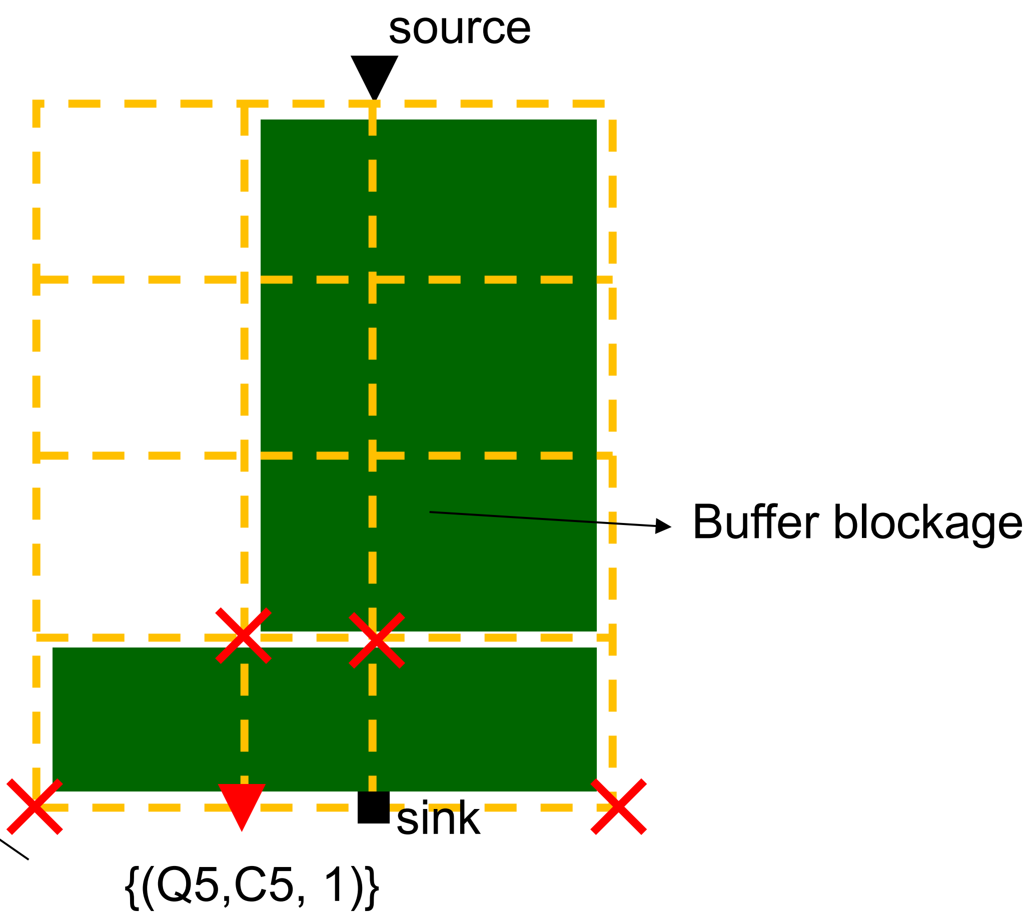
R is resistance of w

Fast Path Algorithm



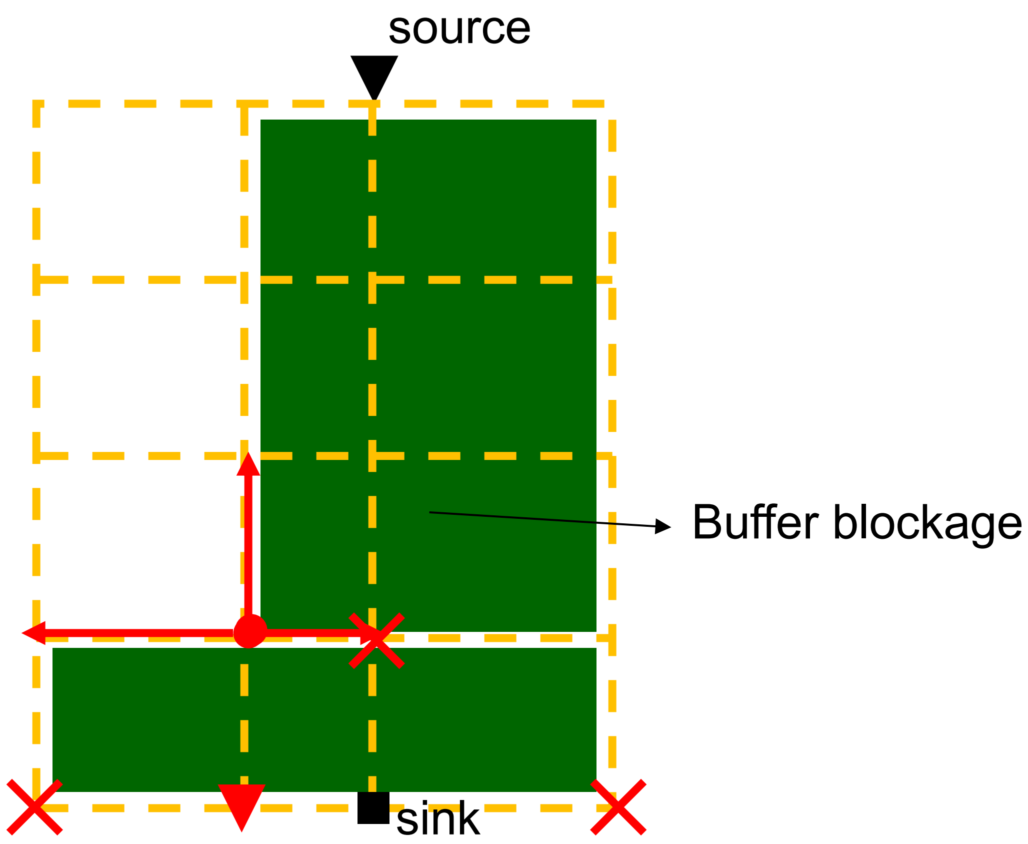
Q2= Q1+R (C/2 + C1)
C2= C+C1
C is capacitance of wire w,
R is resistance of w

Fast Path Algorithm

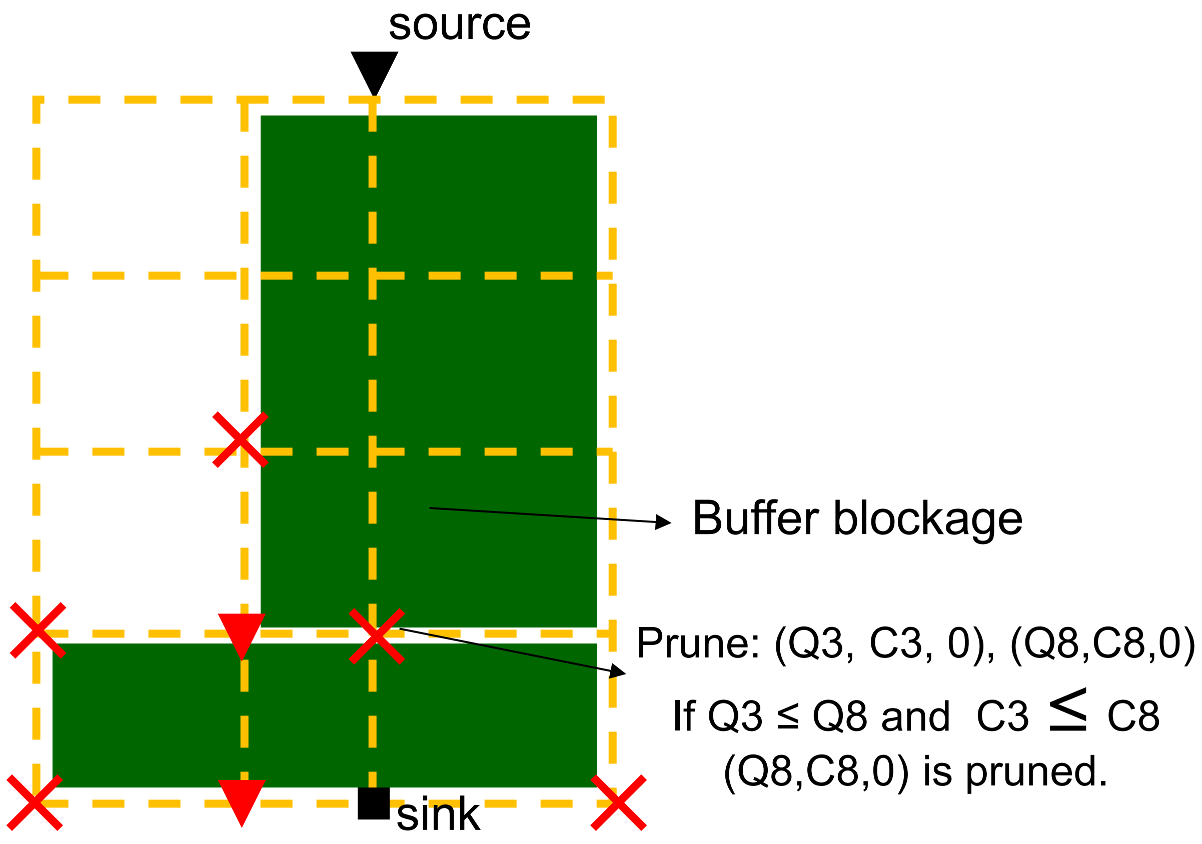


Q5=Q2 + Db + Rb*C2,
C5= Cb ,where
Db is intrinsic delay of buffer
Cb is input capacitance of buffer
Rb is output resistance of buffer

Fast Path Algorithm



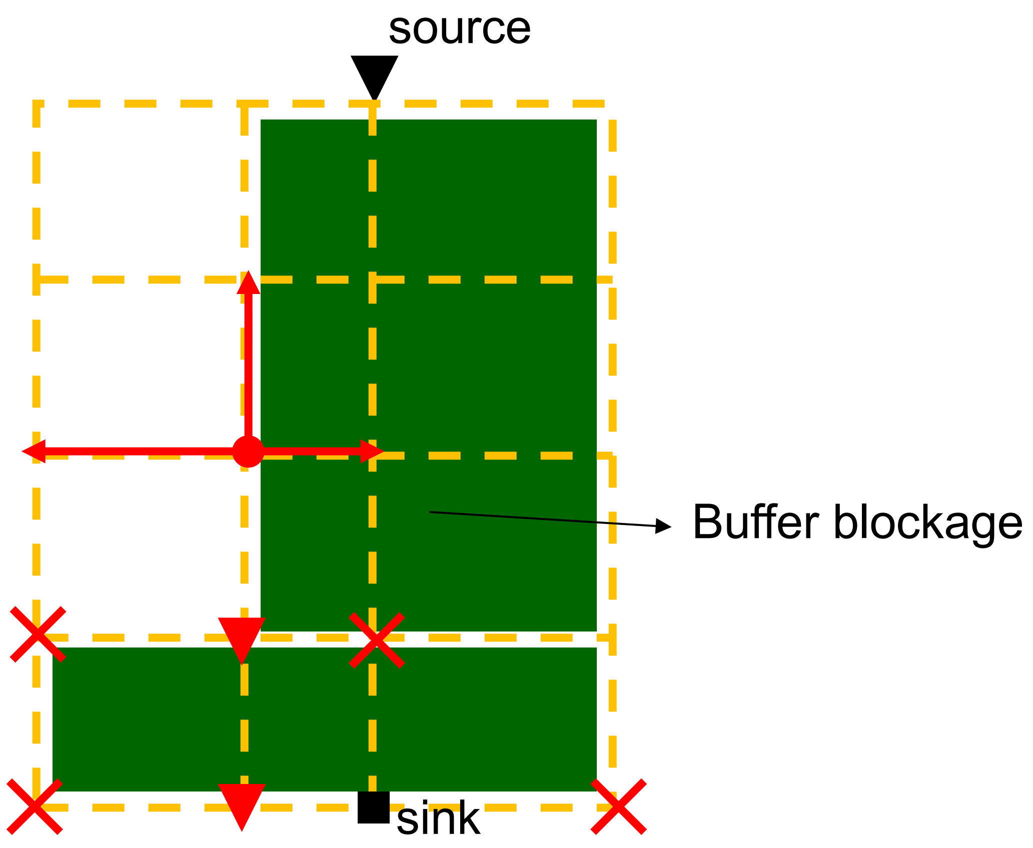
Fast Path Algorithm



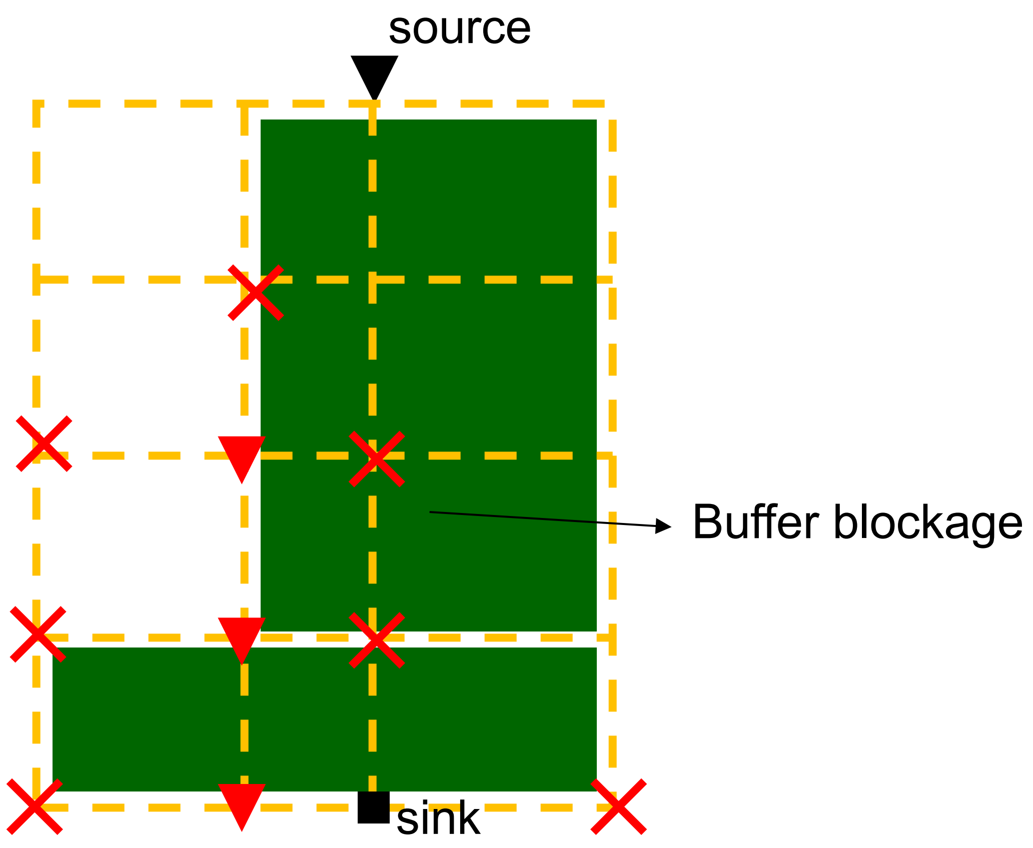
Prune: (Q3, C3, 0), (Q8,C8,0)

If Q3 ≤ Q8 and C3 ≤ C8
(Q8,C8,0) is pruned.

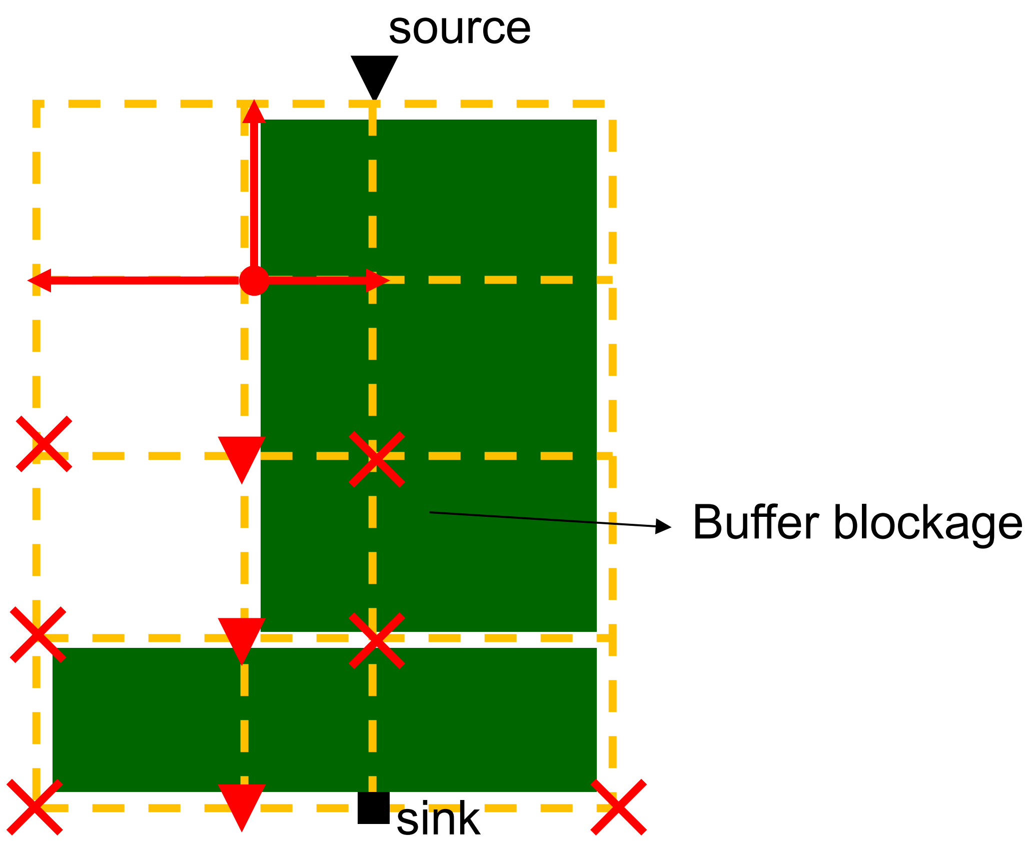
Fast Path Algorithm



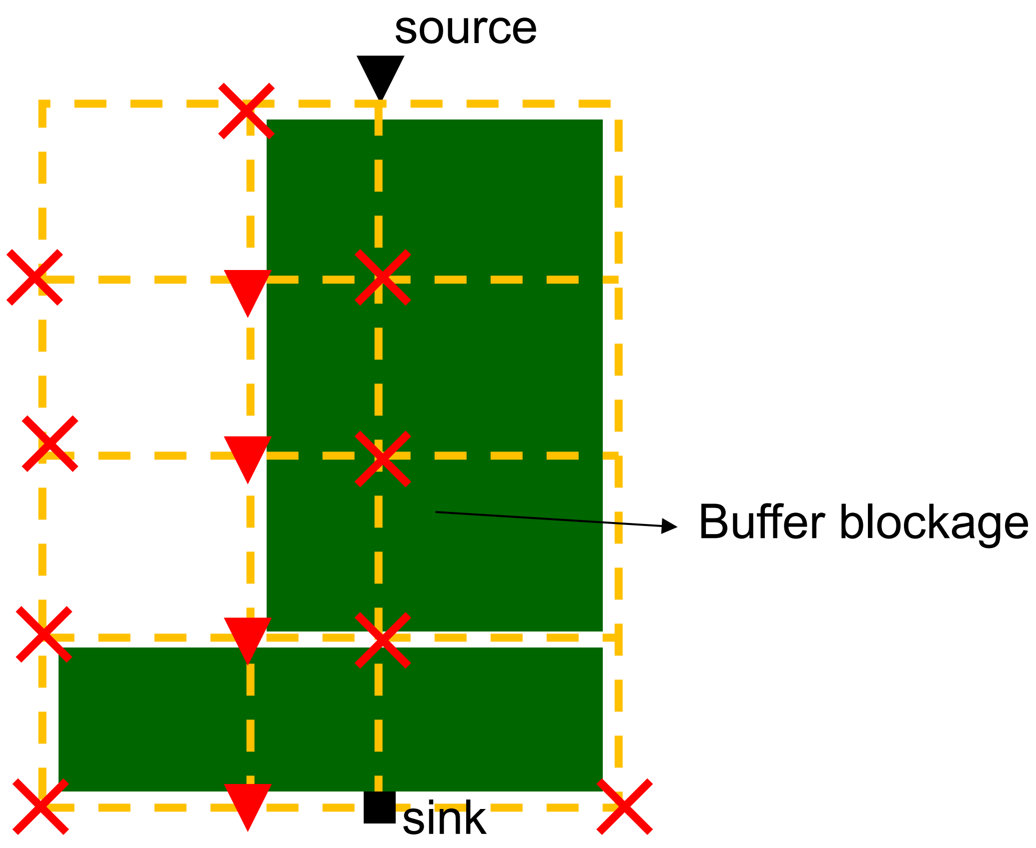
Fast Path Algorithm



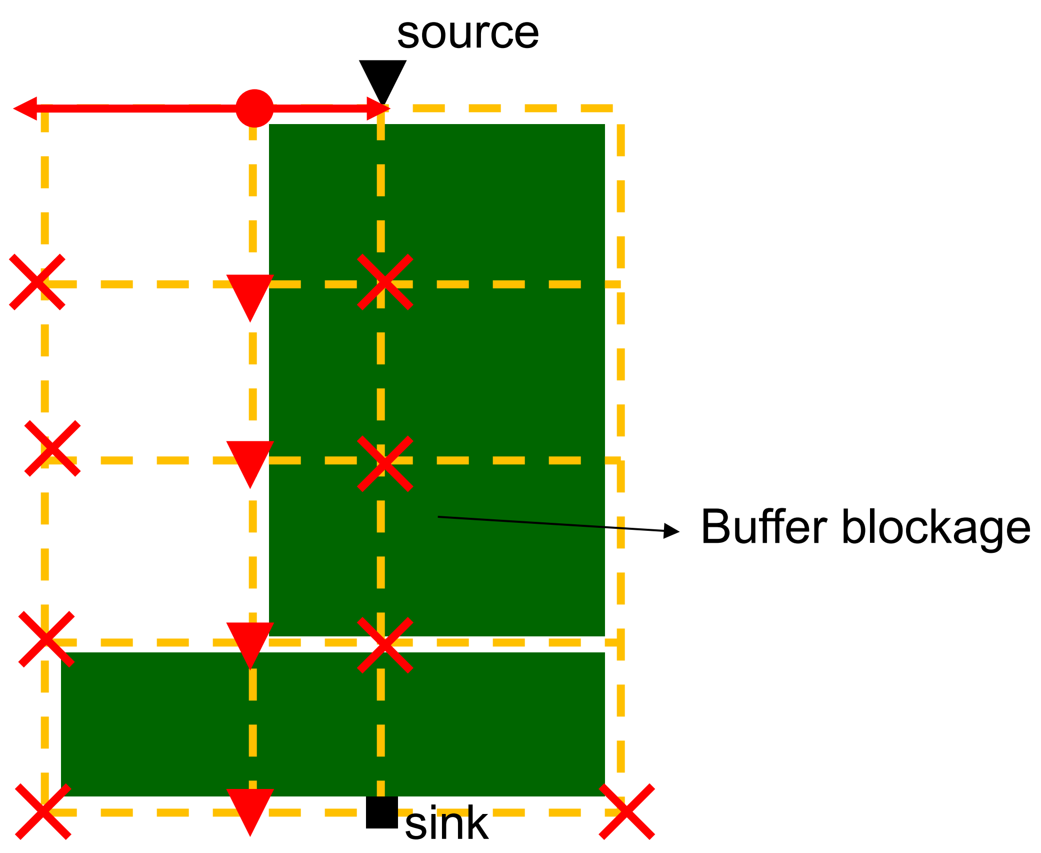
Fast Path Algorithm



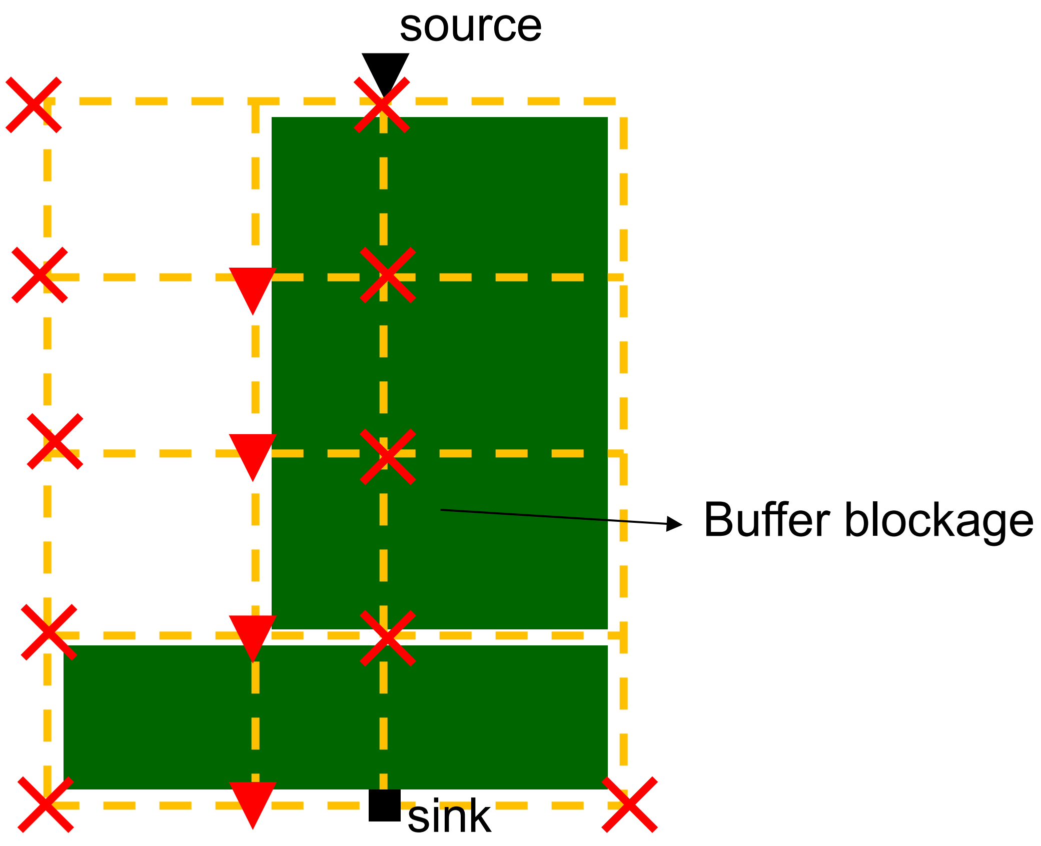
Fast Path Algorithm



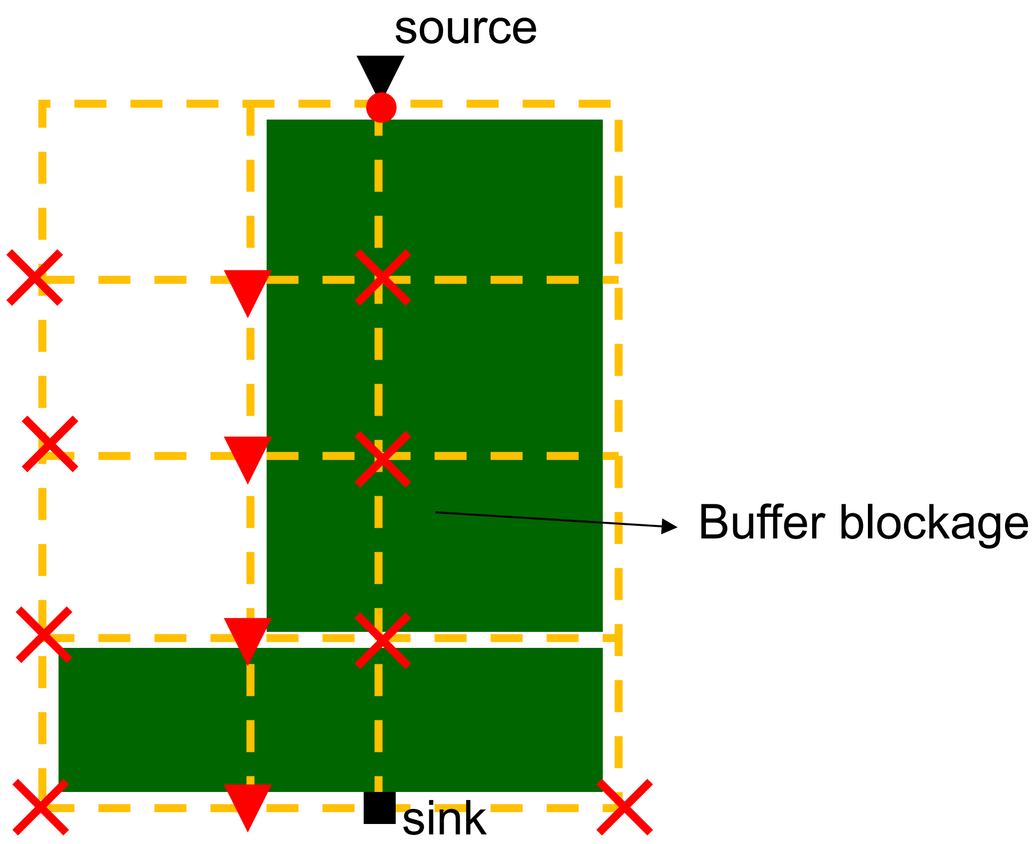
Fast Path Algorithm



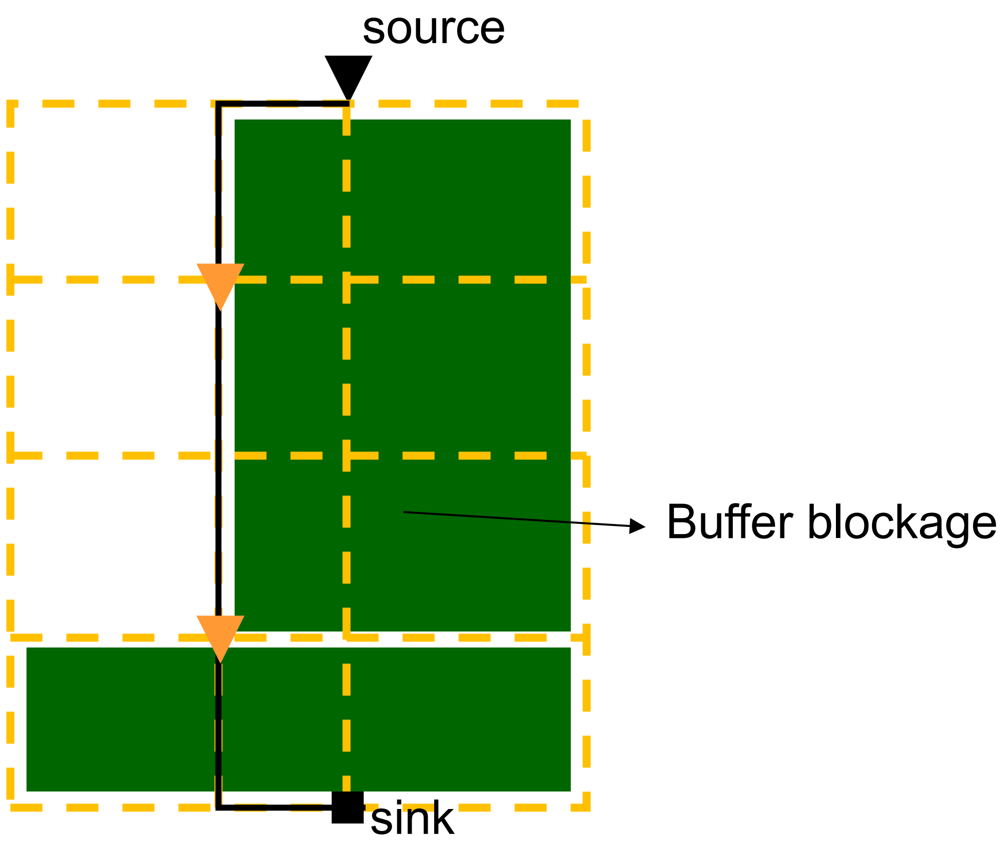
Fast Path Algorithm



Fast Path Algorithm



Fast Path Algorithm


source
Buffer blockage
sink