Digital VLSI Design

Lecture 7: Placement

Semester A, 2018-19

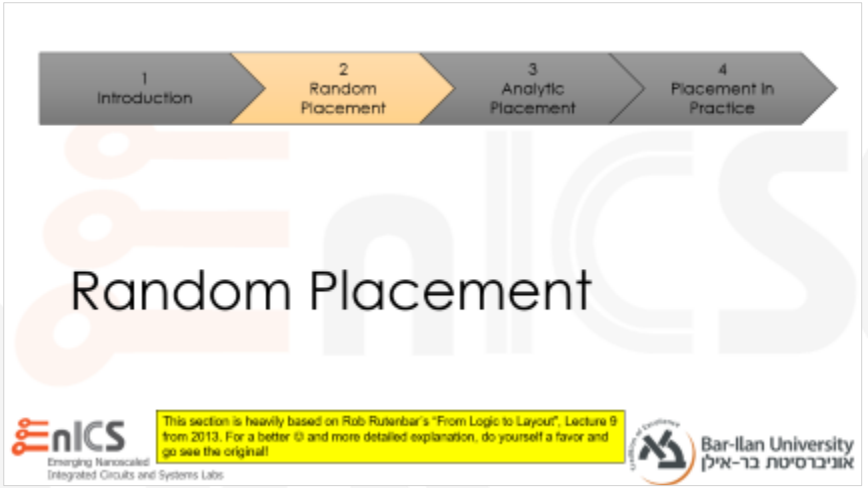
Lecturer: Dr. Adam Teman

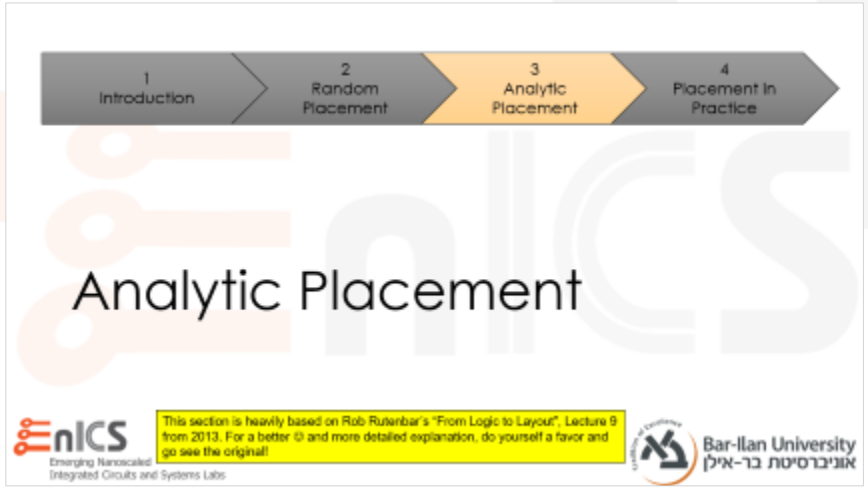
28 December 2018

Emerging Nanoscaled
Integrated Circuits and Systems Labs

Tradition of Excellence
Bar-Ilan University
אוניברסיטת בר-אילן

Lecture Outline

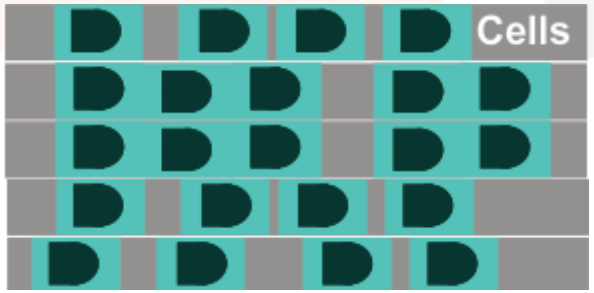


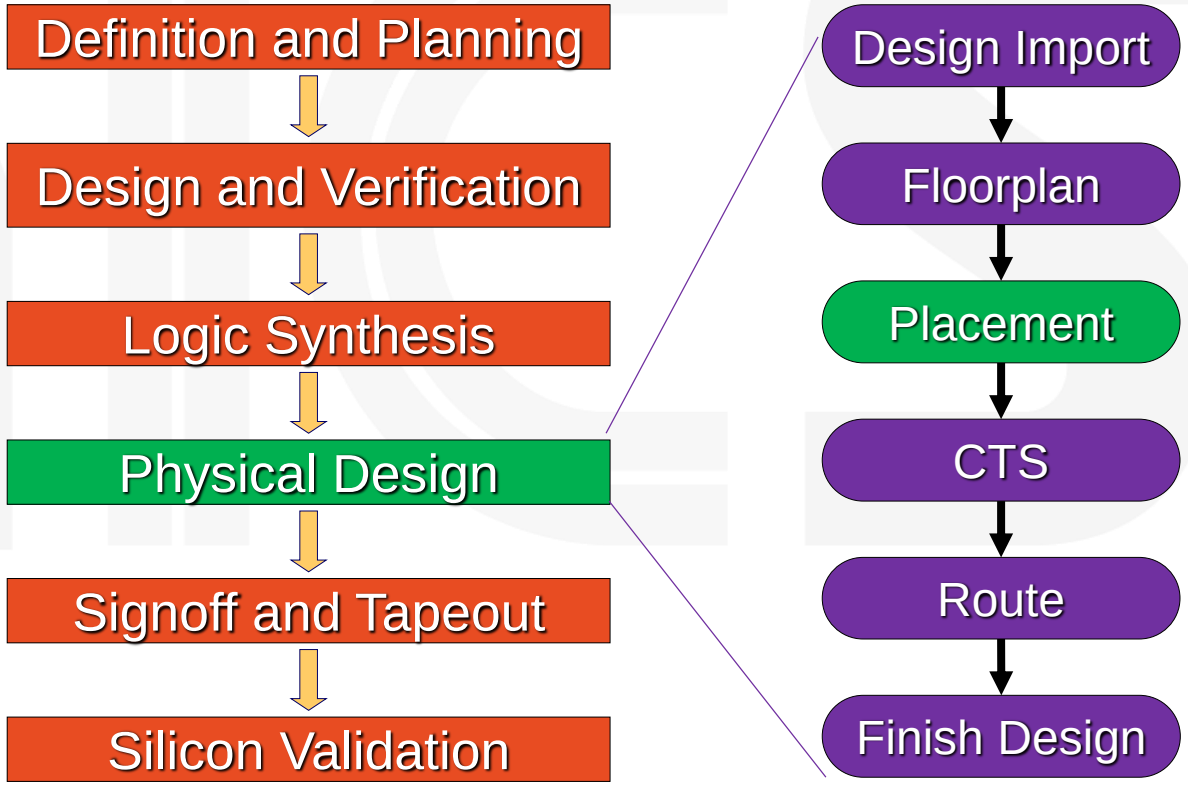


Introduction

Where are we in the design flow?

- We have successfully synthesized our design into a technology mapped gatelevel netlist.
- We have designed a floorplan with pre-placed blocks.
- Now we will move into detailed placement of the standard cells.

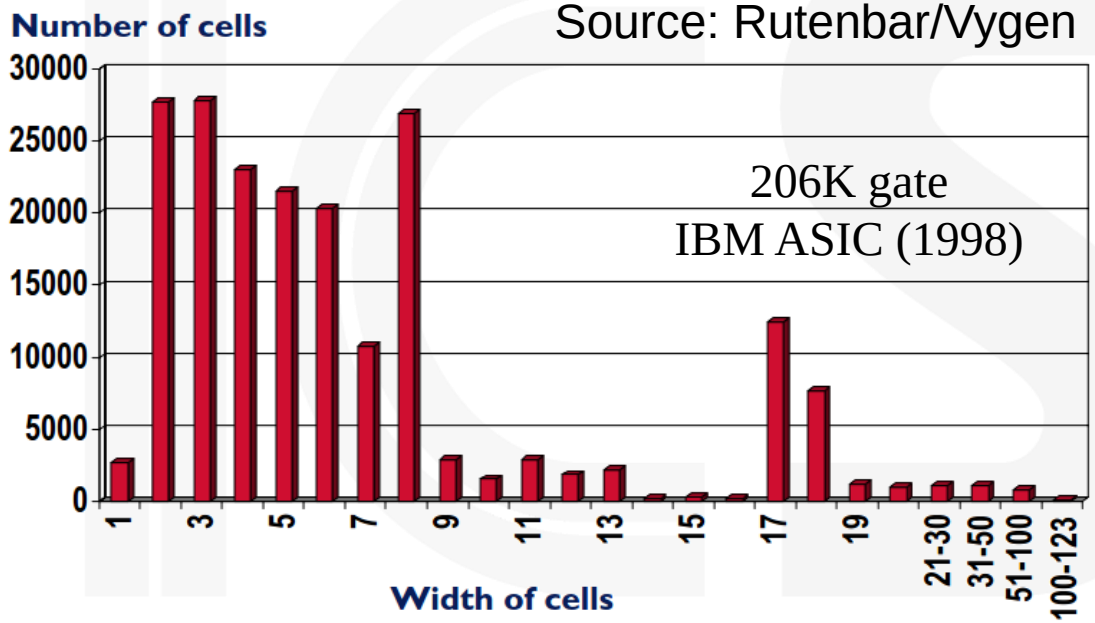


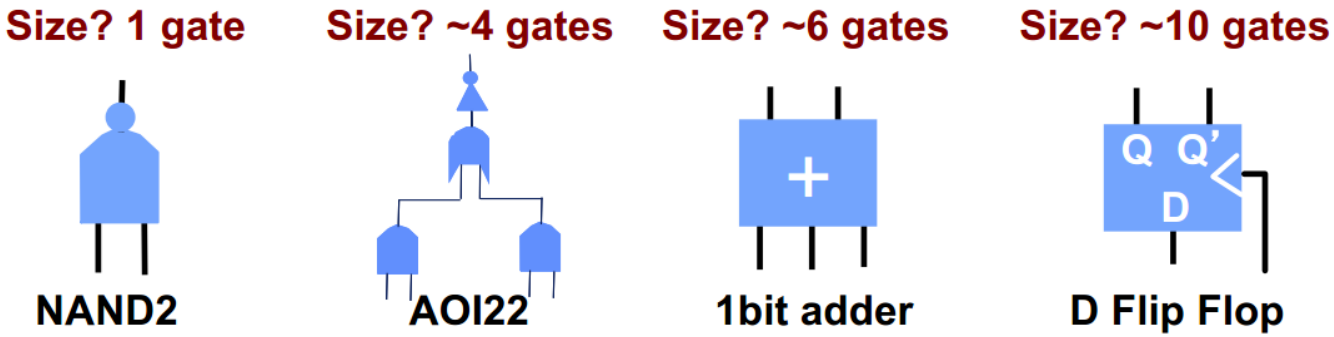


Chip Size…?

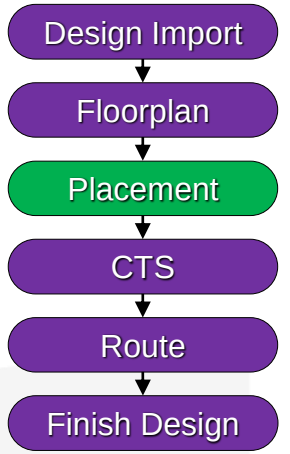
- **Before we start discussing how to place the standard cells, we would like to make sure we understand how big a problem it is.**
  - How big is a “100 Million gate ASIC”?
- **In real designs, small cells dominate.**
  - Therefore, numbers are usually given as “*equivalent small gates*”
  - May also consider a hard macro to be many, many small gates.
- **Conclusion:**
  - *Gates==Equivalent* NAND2 *gates*
  - Instances: # of things placed!
  - Rule of thumb: *Instances=Gates/4..5*

Source: Rutenbar/Vygen

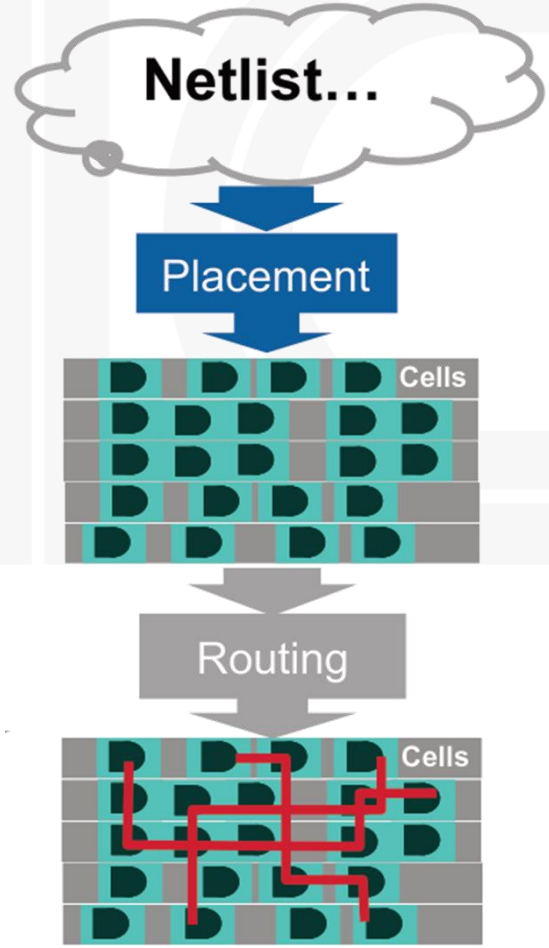




Placement

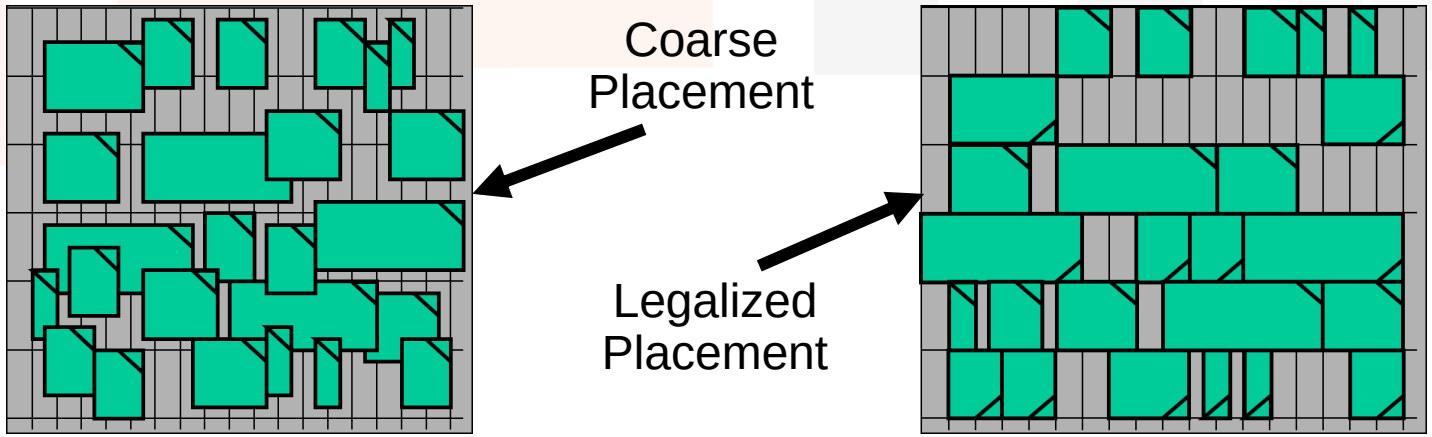


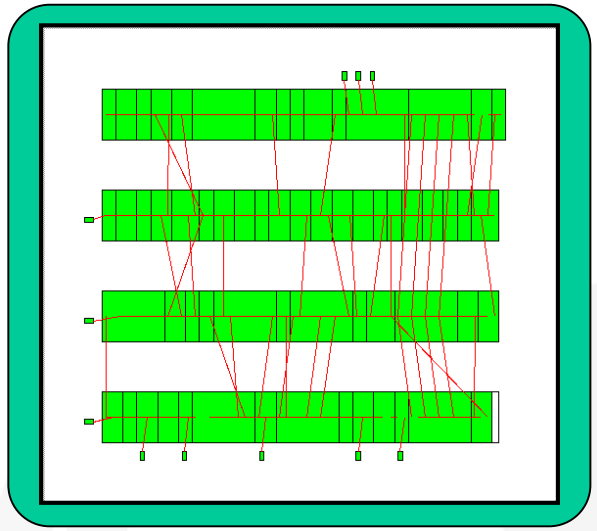
- **Placement is the stage of the design flow, during which each instance (standard cell) is given an exact location.**
- **Inputs:**
  - Netlist of gates and wires.
  - Floorplan and Technology constraints
- **Output:**
  - All cells located in the floorplan.
- **Goal**
  - Provide *legal* location of entire netlist
  - Enable detailed route of all nets
  - Meet timing, area, and power targets



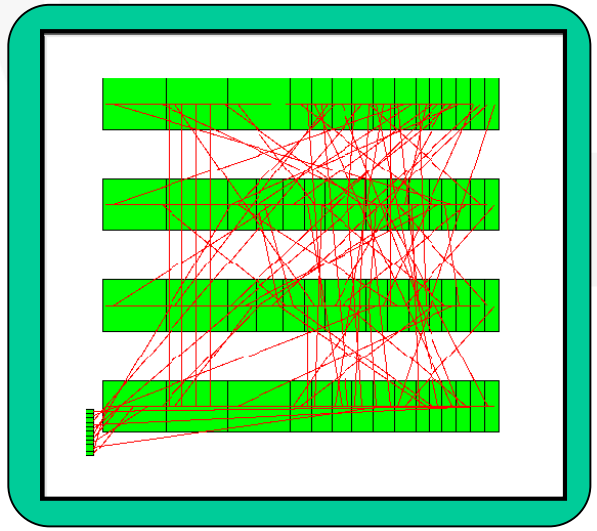
Placement Flow

- **In general, most tools partition the placement task into two stages:**
  - Global placement:
    - Quickly divide each cell into “bins” to try and minimize the number of connections between groups.
  - Detailed placement:
    - Provide a legal placement for each instance
    - Try and minimize wirelength (or other cost metrics)
    - Try to finish with uncongested design.

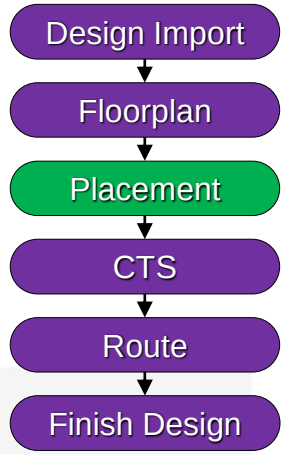




Good Placement



Bad Placement



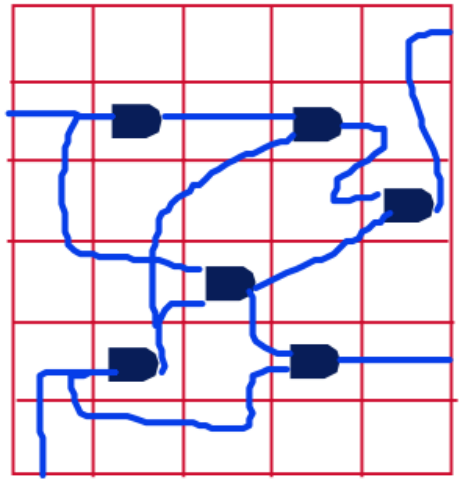
Random Placement

This section is heavily based on Rob Rutenbar's "From Logic to Layout", Lecture 9 from 2013. For a better ☺ and more detailed explanation, do yourself a favor and go see the original!

Problem Formulation

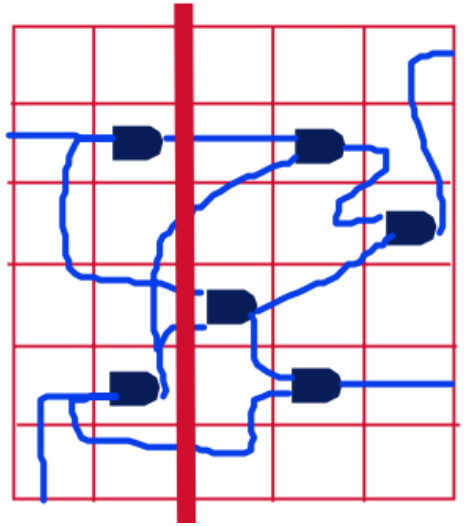
- **Given a netlist, and fixed-shape cells (small, standard cell), find the exact location of the cells to minimize area and wire-length**
  - Consistent with the standard-cell design methodology
    - Row-based, no hard-macros
  - Modules
    - Usually fixed, equal height (exception: double height cells)
    - Some fixed (I/O pads)
    - Connected by edges or hyperedges
- **Objectives**
  - Cost components: area, wire length
  - Additional cost components: timing, congestion

Wirelength Minimization



Add up the estimated length of all nets and try to minimize

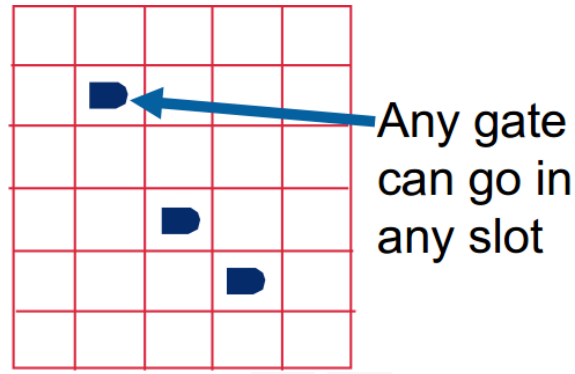
Congestion Minimization



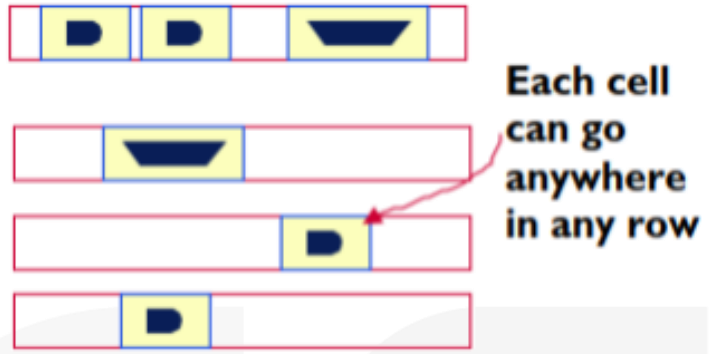
Take any cut through the placement and try to minimize the number of nets that cross it.

Simple Placer

- **Assume a very simple chip model:**
  - Simple grid – cells go in squares
  - Pins fixed at edges
- **Assume simple gate model:**
  - All gates same size
  - Each grid slot can hold one gate.
- **Simple bounding box wirelength estimator:**
  - "Half-Perimeter Wirelength" (HPWL)
  - Put the smallest box around all gates on net.
  - Measure Width (ΔX) and Height (ΔY) of box.
  - **HPWL**= $\Delta X + \Delta Y$

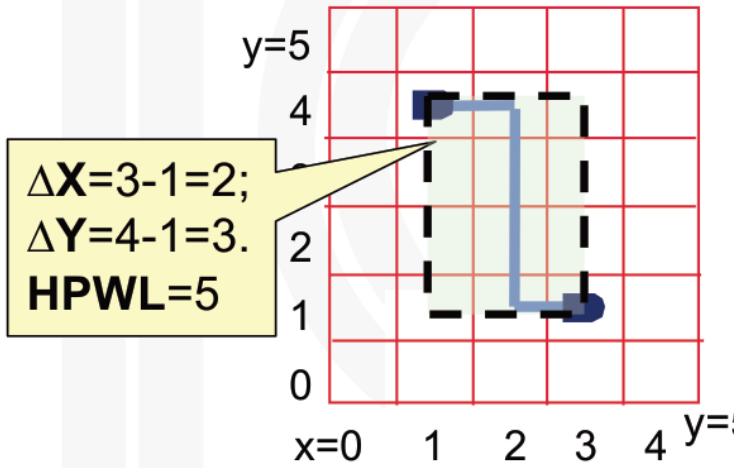


Simplistic Model

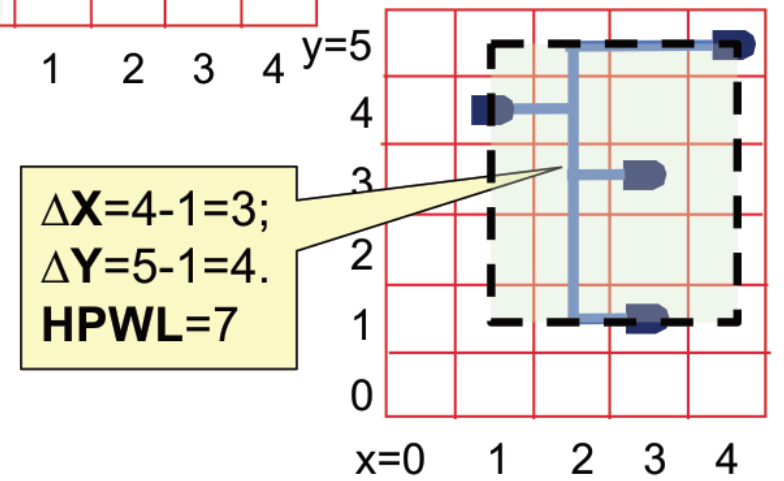


More realistic Model

A "2-point net"



A "4-point net"



Simple Placer

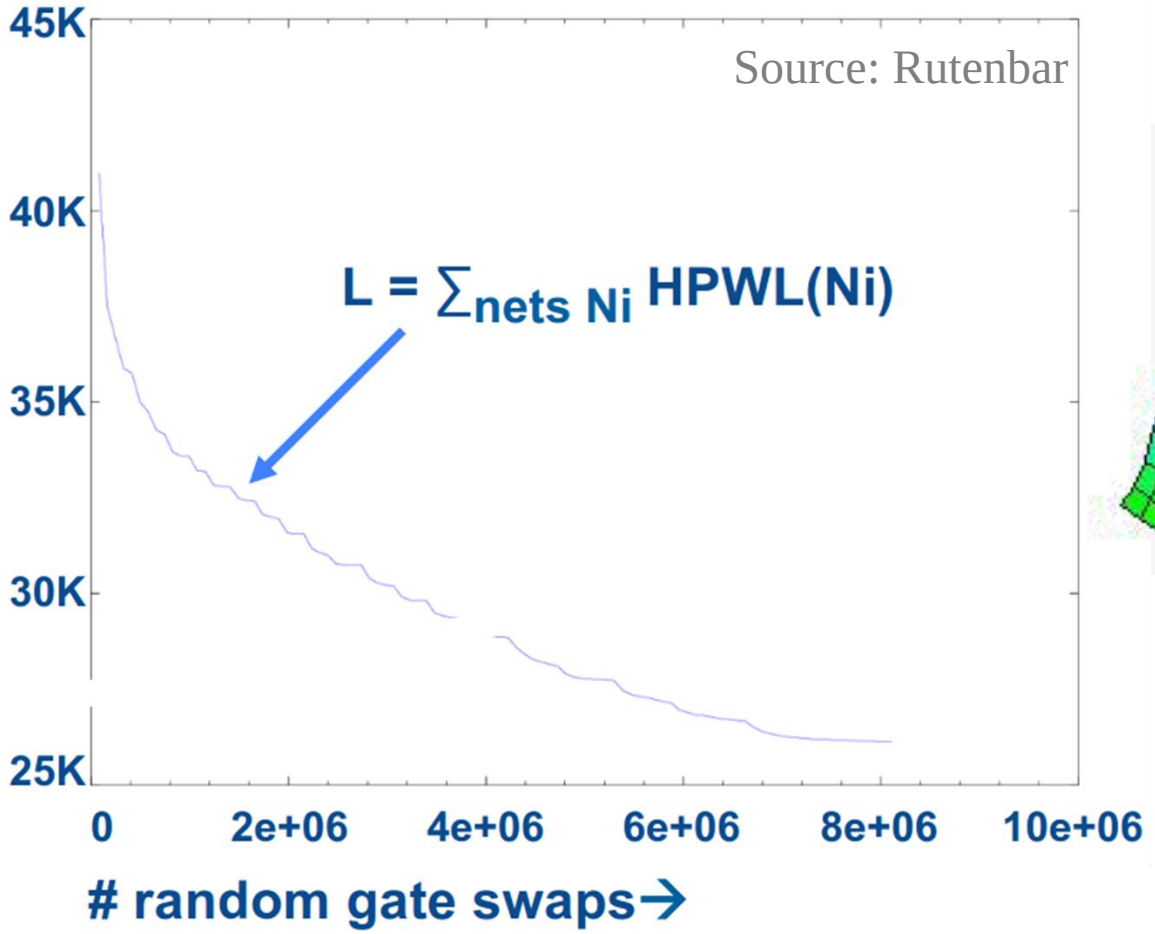
Let's define a simple algorithm:

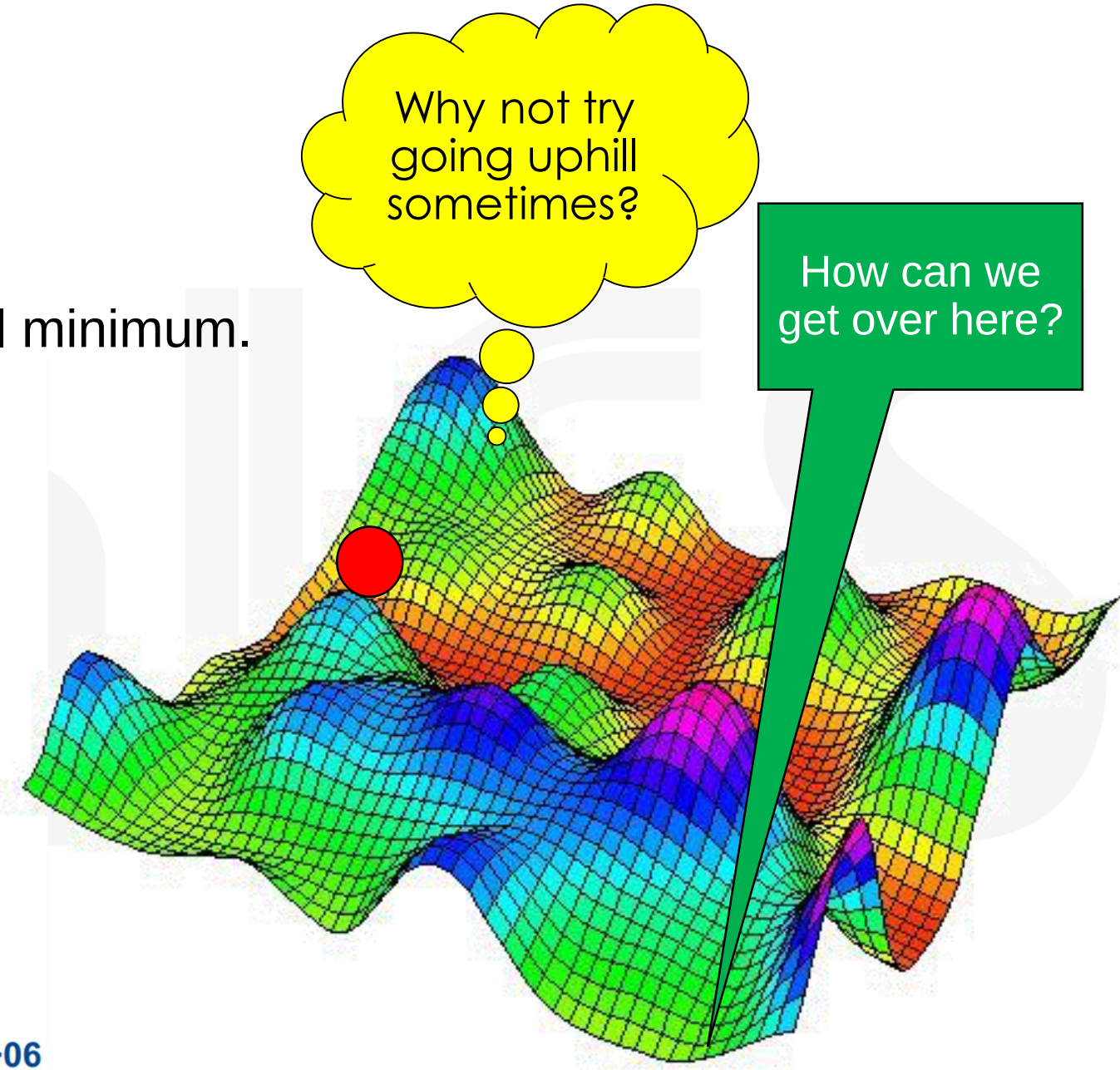
- **Start with a random placement:**
  - Randomly assign each gate to a grid location.
- **Apply random iterative improvement:**
  - Pick a random pair of gates.
  - Swap their locations and evaluate the change in total wirelength.
  - If wirelength got smaller – accept the swap.
    If wirelength got bigger – undo the swap.
  - Repeat until wirelength stops improving.

```
// Random initial placement
foreach (gate Gi in netlist)
  place Gi in random (x,y) not occupied.

// calculate initial HPWL
L=0
foreach (net Ni in netlist)
  L = L + HPWL(Ni)

// random iterative improvement
while (overall HPWL is improving)
  pick two random gates Gi, Gj
  swap Gi and Gj
  evaluate ΔL = new HPWL - old HPWL
  if (ΔL < 0) then keep the swap
  if (ΔL > 0) undo the swap
```

Simple Placer

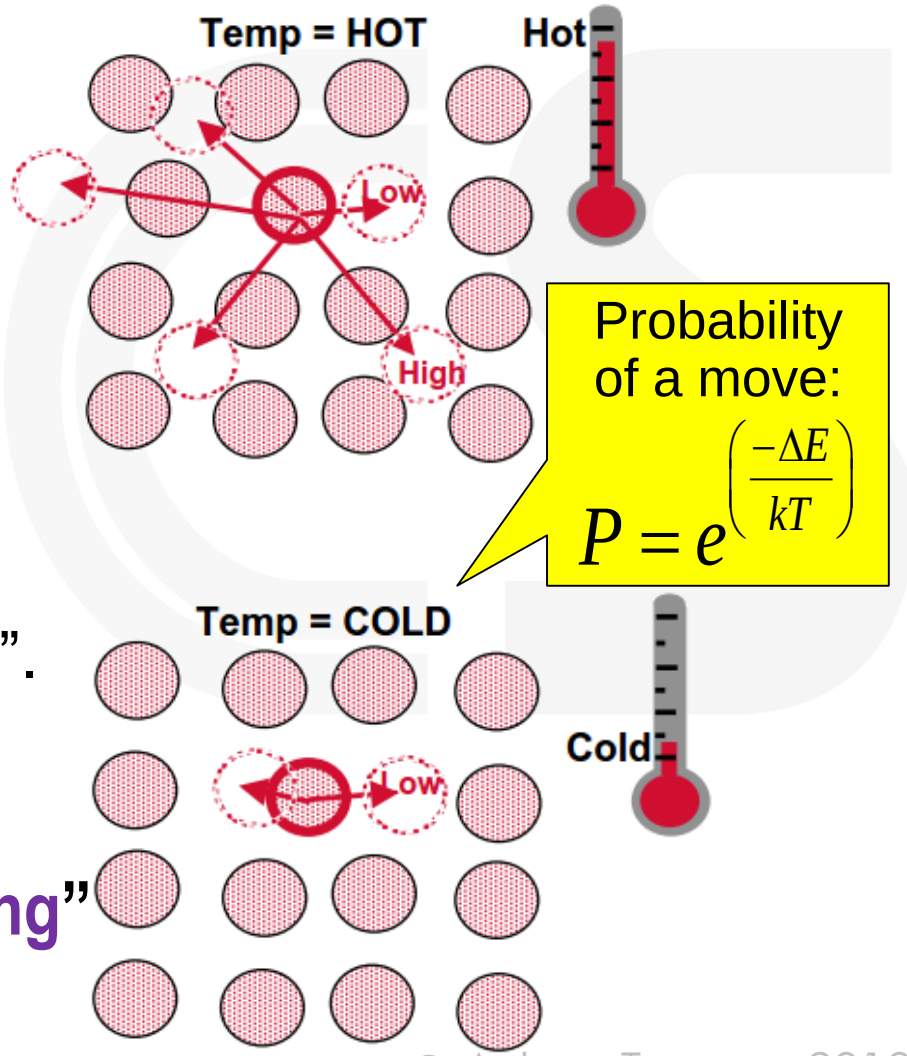
- **Was this any good?**
  - Well, we quickly get stuck in a local minimum.





Simulated Annealing

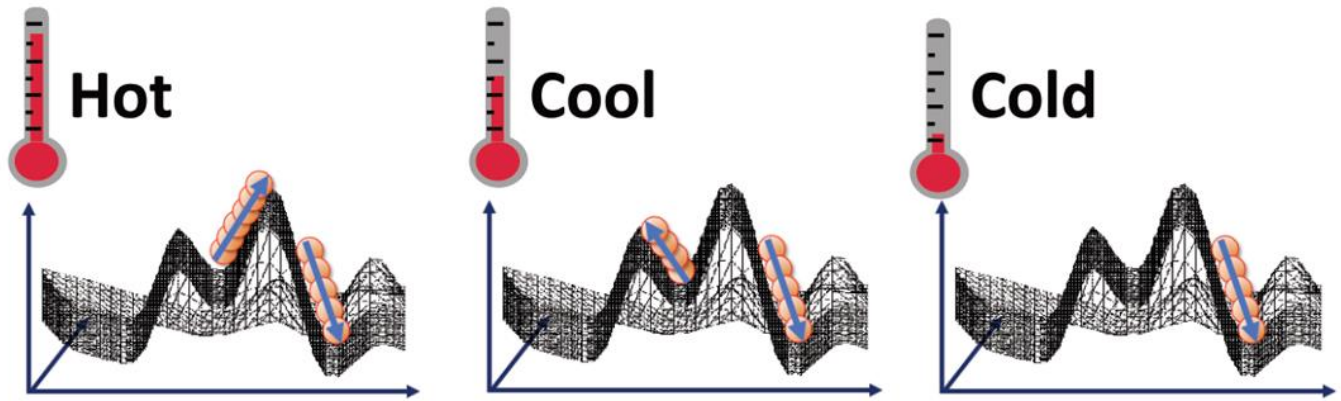
- **When we learned about semiconductors, we discussed "annealing":**
  - The lowest energy state of a crystal lattice is when all atoms are lined up.
  - So if we have a messy crystal:
    - Heat it up – give atoms energy to move around.
    - Cool it slowly –
      - At first, atoms will move a lot.
      - But as the crystal cools, the movement will be restricted.
- **What if we apply this idea to random hill climbing?**
  - At first ("hot temperature") we'll "climb a lot of hills".
  - As we progress ("cool down") we will make fewer big jumps.
- **This very famous idea is called "Simulated Annealing"**



Simulated Annealing Algorithm

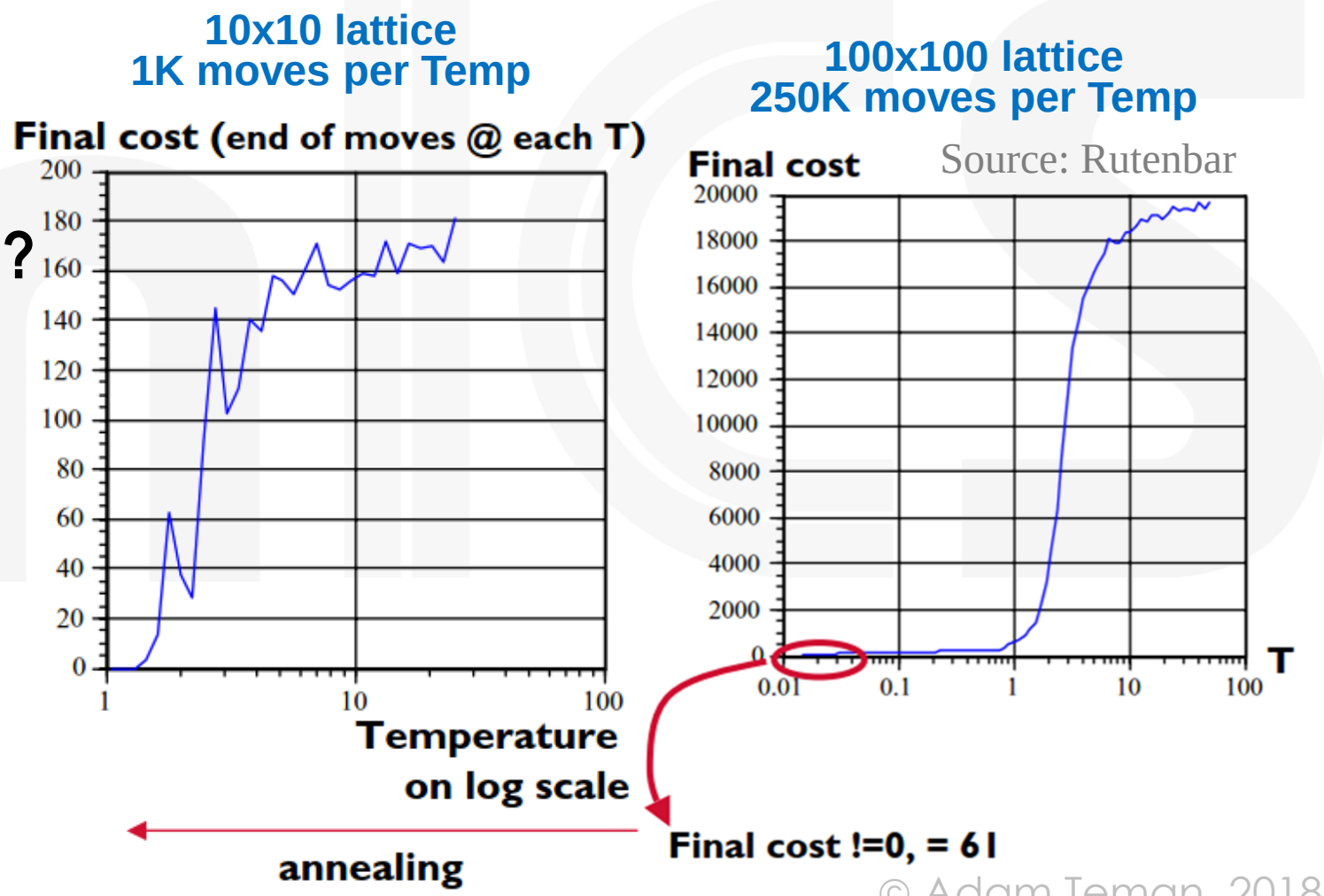
- **Start with the same basic algorithm:**
  - Random initial placement
  - Swap two Random gates
  - Evaluate change in HPWL (ΔL)
  - If wirelength improves, accept the change.
- **But what if wirelength *increases* ($\Delta L>0$)?**
  - Evaluate annealing probability function $P = \exp\left(\frac{-\Delta L}{T}\right)$
  - Choose a uniform random number (R) between 0 and 1
  - If $R<P$ then keep the swap.
- **What is T?**
  - T is the simulated temperature.
  - Start hot. Cool down.

```
T=HOT; frozen = false
while (!frozen)
  swap two random gates Gi, Gj
  evaluate ΔL
  if (ΔL < 0) then
    keep the swap
  else
    if (random() < exp(-ΔL/T))
      accept swap
    else
      undo swap
  if (HPWL still decreasing)
    T = 0.9*T
  else
    frozen=true
```



Simulated Annealing

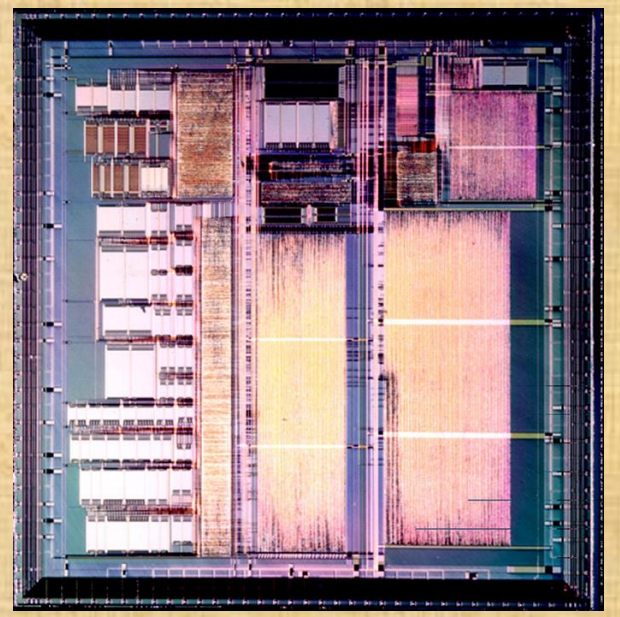
- **How well does this work?**
  - Really well!
  - Many EDA algorithms use Simulated Annealing.
- **Does it find an optimal solution?**
  - No. But it's good at avoiding local minima.
- **What happens if I run it again?**
  - I will get a different answer each time!
- **NOT how placers work today!**



The Chip Hall of Fame

- **As AI is becoming a larger part of our lives, let's remember the granddaddy of them all:**

IBM DeepBlue



- **The first computer to beat a chess champion (Gary Kasparov)**
  - First tournament win: May 1997
  - Transistor Count: 1.5M 11.38 GFLOPS
  - Could make 200M Chess Moves per second
  - Kasparov said the moves were "uncomputerlike" a that they "exerted great psychological pressures"

2017 Inductee to the IEEE Chip Hall of Fame

Analytic Placement

This section is heavily based on Rob Rutenbar's "From Logic to Layout", Lecture 9 from 2013. For a better ☺ and more detailed explanation, do yourself a favor and go see the original!

Analytical Placement Approach

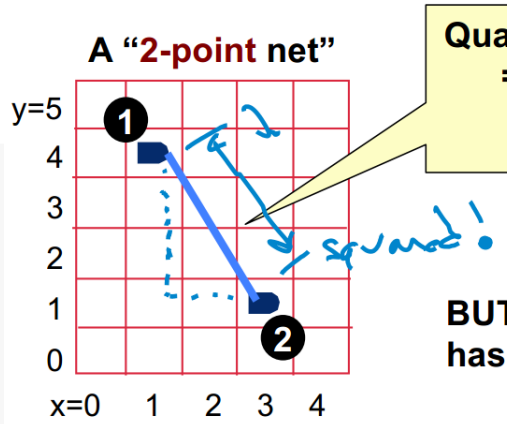
- **Question:**
  - Can we write an equation whose minimum is the placement?
  - If we have a cost function (such as wirelength) that is a function of the gate coordinates (x_i, y_i), i.e.: $L_{\text{wire}} = f\left(x_1, x_2, ..., x_N, y_1, y_2, ..., y_N\right)$
  - Then we could find the minimum of f and this would be our optimal placement!

- **Sounds crazy, but the answer is YES!**
  - All modern placers are based on analytical placement.
  - We need to write the cost function in a mathematically friendly way.
  - Then, we can just differentiate and equate to 0!

$$\frac{\partial f}{\partial x} = 0, \frac{\partial f}{\partial y} = 0$$

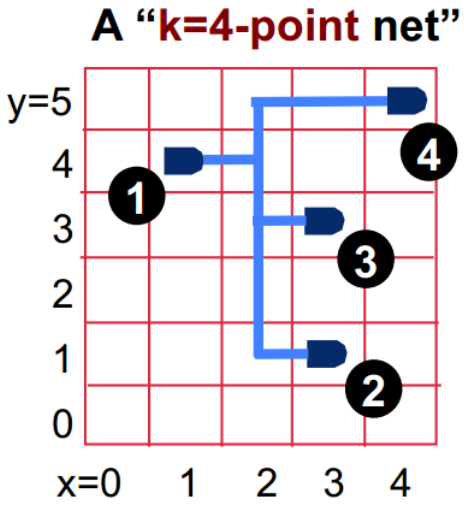
Analytical Placement Cost Function

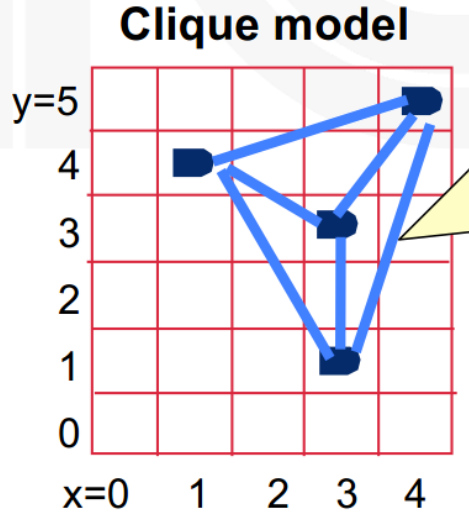
- **Instead of HPWL, let's define a new wirelength model**
  - Quadratic wirelength: $L = (x_1 - x_2)^2 + (y_1 - y_2)^2$
- **What about a *k*-point net (*k*>2)?**
  - Instead of one "real" net, replace with a *fully connected clique model*.
  - So each gate on the net has a one-to-one connection with the other.
  - Altogether $k(k\text{-}1)/2$ nets
  - Compensate by weighting each new net by $1/(k\text{-}1)$
- **One last point:**
  - Assume that gates are *dimensionless* points.



Quadratic wirelength
$=(3\text{-}1)^2 + (4\text{-}1)^2$
$=13$

BUT… what happens if your net has *more* than 2 points in it?

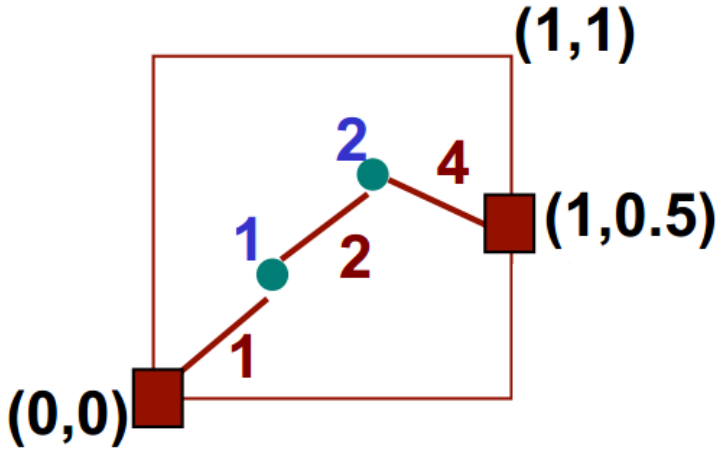




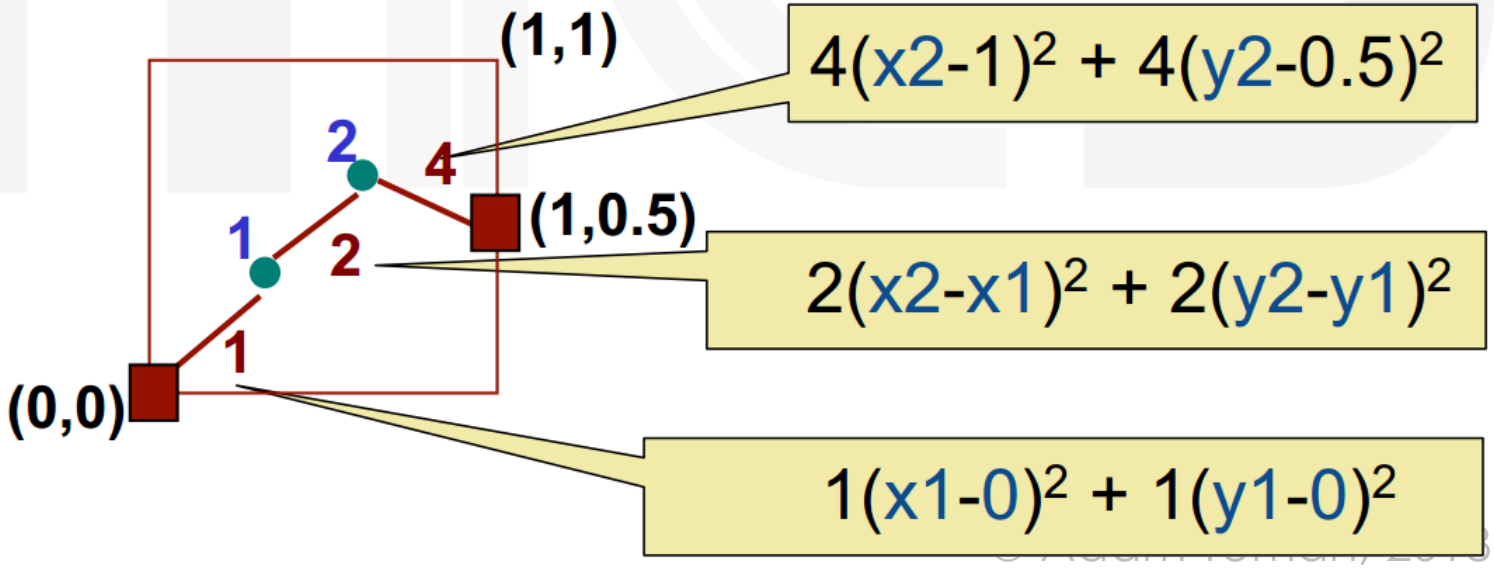
Quadratic estimate:
$(1/3)[(4\text{-}1)^2 + (5\text{-}4)^2]$
$+(1/3)[(4\text{-}3)^2 + (5\text{-}1)^2]$
$+(1/3)[(3\text{-}1)^2 + (4\text{-}1)^2]$
$+(1/3)[(3\text{-}1)^2 + (4\text{-}3)^2]$
$+(1/3)[(4\text{-}3)^2 + (5\text{-}3)^2]$
$+(1/3)[(3\text{-}3)^2 + (3\text{-}1)^2]$
=sum of 6 weighted 2-point lengths

Analytical Placement Calculation

- **Example of quadratic wirelength calculation:**
  - Each point has an unknown (x_i,y_i) coordinate
  - Each net has a pre-assigned weight.
  - Pads are fixed pins on the edge.

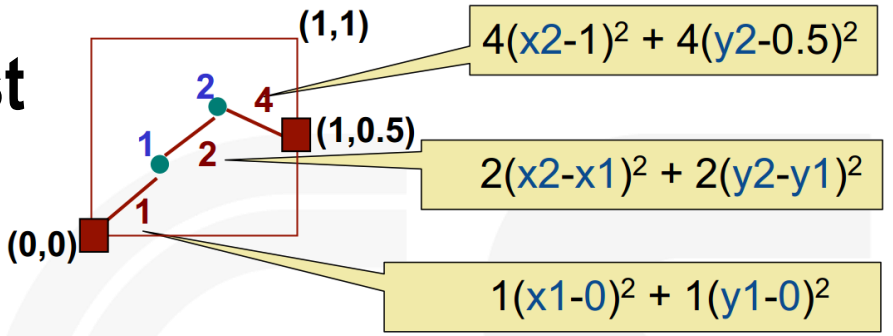


- **Important:**
  - There are no terms with $x_i * y_i$
  - Therefore, we can separate x and y terms in the sum!



Analytical Placement Calculation

- **Now that we have an analytic expression for the cost function, we can use basic calculus to minimize it!**



$$Q(x) = 4(x_2 - 1)^2 + 2(x_2 - x_1)^2 + 1(x_1 - 0)^2$$

$$\frac{\partial Q(x)}{\partial x_1} = 0 + 4(x_2 - x_1)(-1) + 2(x_1) = 6x_1 - 4x_2 = 0$$

$$\frac{\partial Q(x)}{\partial x_2} = 8(x_2 - 1) + 4(x_2 - x_1) + 0 = -4x_1 + 12x_2 - 8 = 0$$

$$\begin{pmatrix} 6 & -4 \\ -4 & 12 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 8 \end{pmatrix}$$

$$x_1 = 0.571$$
$$x_2 = 0.857$$

$$Q(y) = 4(y_2 - 0.5)^2 + 2(y_2 - y_1)^2 + 1(y_1 - 0)^2$$

$$\frac{\partial Q(y)}{\partial y_1} = 0 + 4(y_2 - y_1)(-1) + 2(y_1) = 4y_1 - 4y_2 = 0$$

$$\frac{\partial Q(y)}{\partial y_2} = 8(y_2 - 0.5) + 4(y_2 - y_1) + 0 = -4y_1 + 12y_2 - 4 = 0$$

$$\begin{pmatrix} 6 & -4 \\ -4 & 12 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$$

$$y_1 = 0.286$$
$$y_2 = 0.429$$

Analytical Placement Example Result

$$\begin{pmatrix} 6 & -4 \\ -4 & 12 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 8 \end{pmatrix}$$

$G_1 : (0.571, 0.286)$
$G_2 : (0.857, 0.429)$

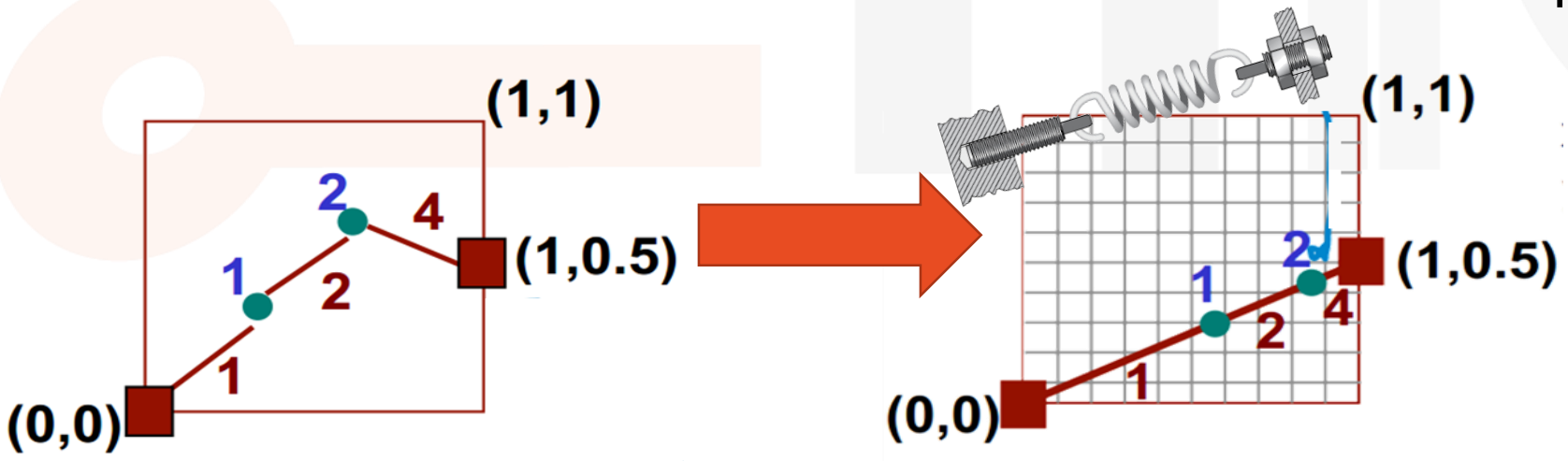
$$\begin{pmatrix} 6 & -4 \\ -4 & 12 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$$

Observations:

- **Placement makes visual sense:**
  - All points are on a straight line.
  - The placement *minimizes spring weights*.
  - Bigger weight → Shorter wire
- **Algebraically:**
  - For N gates, we get two equivalent NxN matrices ($\mathbf{A}$) for two linear systems:

$$\mathrm{A}x = b_x, \mathrm{A}y = b_y$$

  - b vectors represent Pad coordinates!



$$\mathrm{A} = \begin{pmatrix} 6 & -4 \\ -4 & 12 \end{pmatrix}$$

$$b_x = \begin{pmatrix} 0 \\ 8 \end{pmatrix}, b_y = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$$

Building Analytical Network

Recipe for success

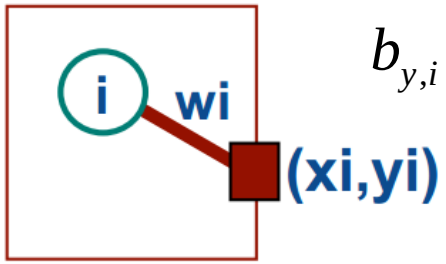
- **Build *connectivity matrix* (C) as follows.**
  - $\mathbf{C}(i,j)=\mathbf{C}(j,i)=w$ for a net with weight w connected between gates i and j.
  - If no net connects gates i and j, $\mathbf{C}(i,j)=0$.
- **Build A matrix:**
  - Off diagonal, $\mathbf{A}(i,j)= -\mathbf{C}(i,j)$
  - $\mathbf{A}(i,i)$ is sum of row i + weight of net from i to pad.
- **Build b vectors:**
  - If gate i connects to a pad at (x_i,y_i) with weight w_i then $b_x(i)=w_i*x_i$, $b_y(i)=w_i*y_i$

$$\mathrm{C}_{i,j}=\mathrm{C}_{j,i}=\begin{cases}0 & i=j\\ 0 & \text{no net}_{i,j}\\ w_{i,j} & \text{net}_{i,j}\end{cases}$$

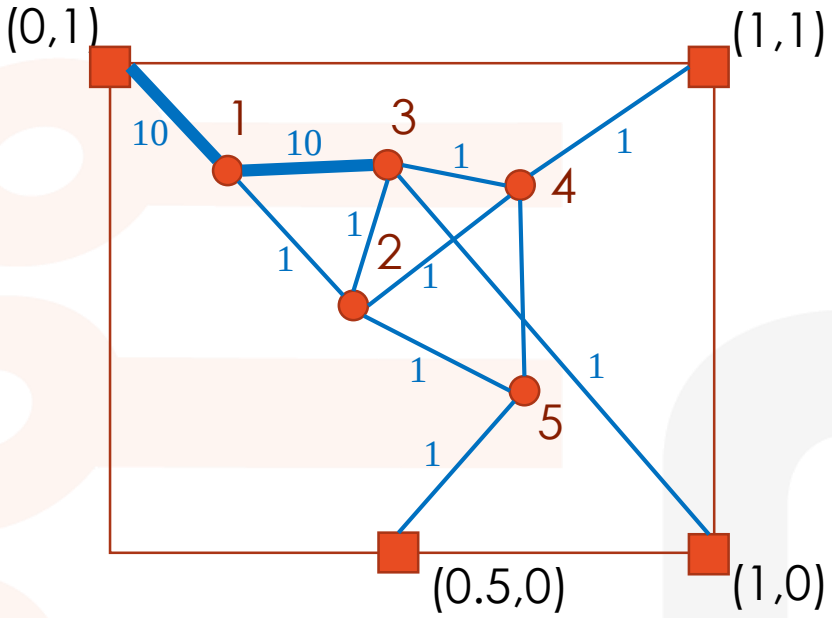
$$\mathrm{A}_{i,j}=\begin{cases}-\mathrm{C}_{i,j} & i\neq j\\ \sum_{j=1}^{N}\mathrm{C}_{i,j}+w_{j,\text{pads}} & i=j\end{cases}$$

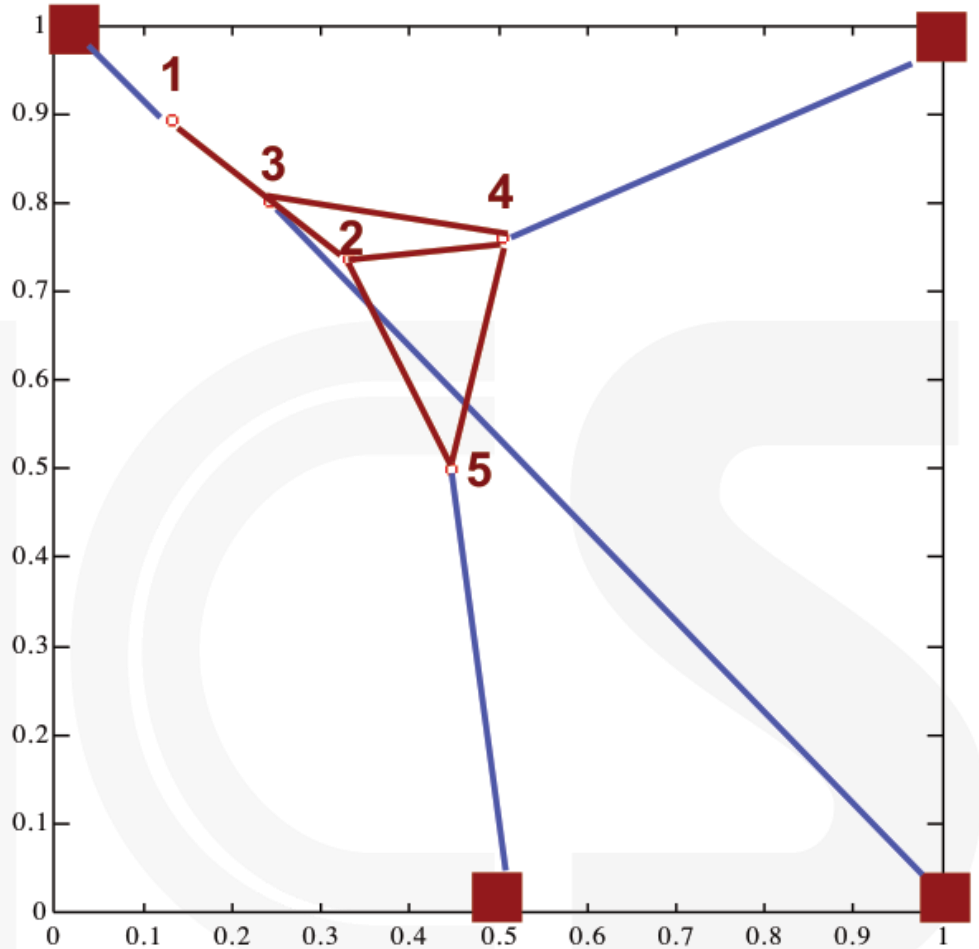
$$b_{x,i}=\begin{cases}0 & \text{no pad}\\ w_i\cdot x_i & \text{pad } (x_i,y_i,w_i)\end{cases}$$

$$b_{y,i}=\begin{cases}0 & \text{no pad}\\ w_i\cdot y_i & \text{pad } (x_i,y_i,w_i)\end{cases}$$



Five Gate Example

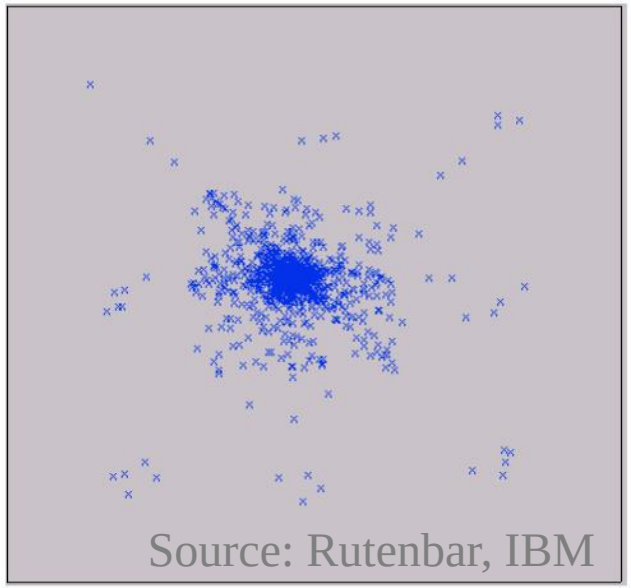


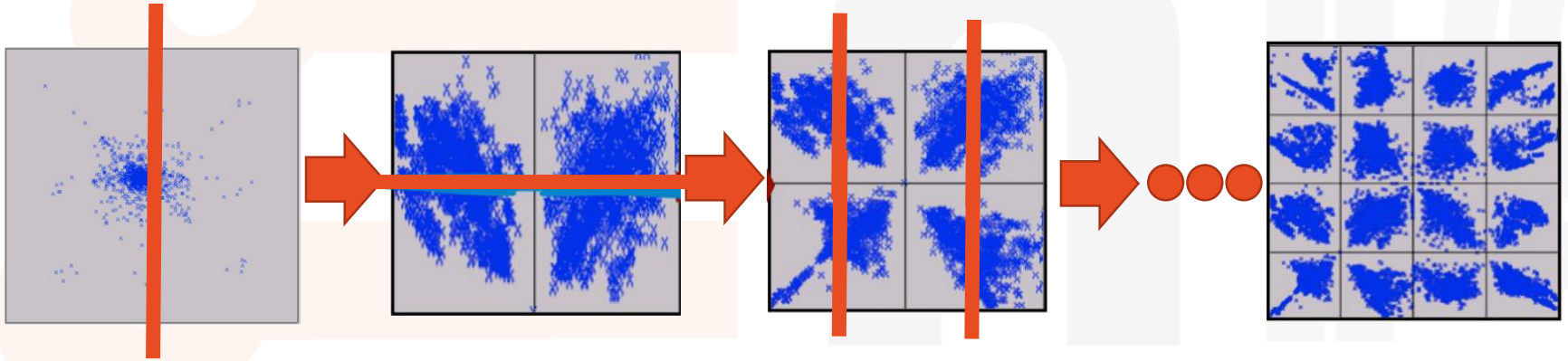


$$C = \begin{pmatrix} 0 & 1 & 10 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 10 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad A = \begin{pmatrix} 21 & -1 & -10 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ -10 & -1 & 13 & -1 & 0 \\ 0 & -1 & -1 & 4 & -1 \\ 0 & -1 & 0 & -1 & 3 \end{pmatrix} \quad b_x = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0.5 \end{pmatrix} \quad b_y = \begin{pmatrix} 10 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Problem – Gate Clustering

- **What does a real quadratic placement look like?**
  - All the gates want to be in the same place!
- **How can we solve this?**
  - Recursive Partitioning!



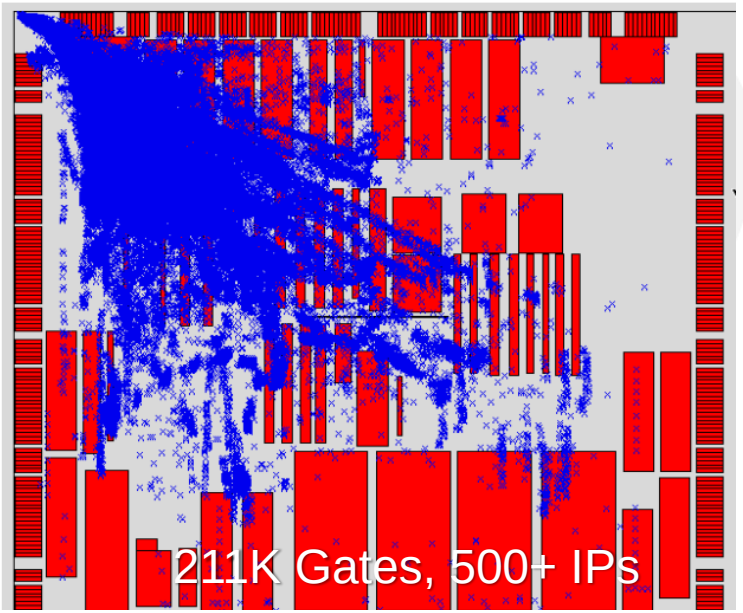


1st Quadratic Place (QP) Solve

Partition chip. Select which gates go in each partition. Solve 2 new smaller QP tasks

Repeat partition on other axis. Solve 4 new smaller QP tasks.

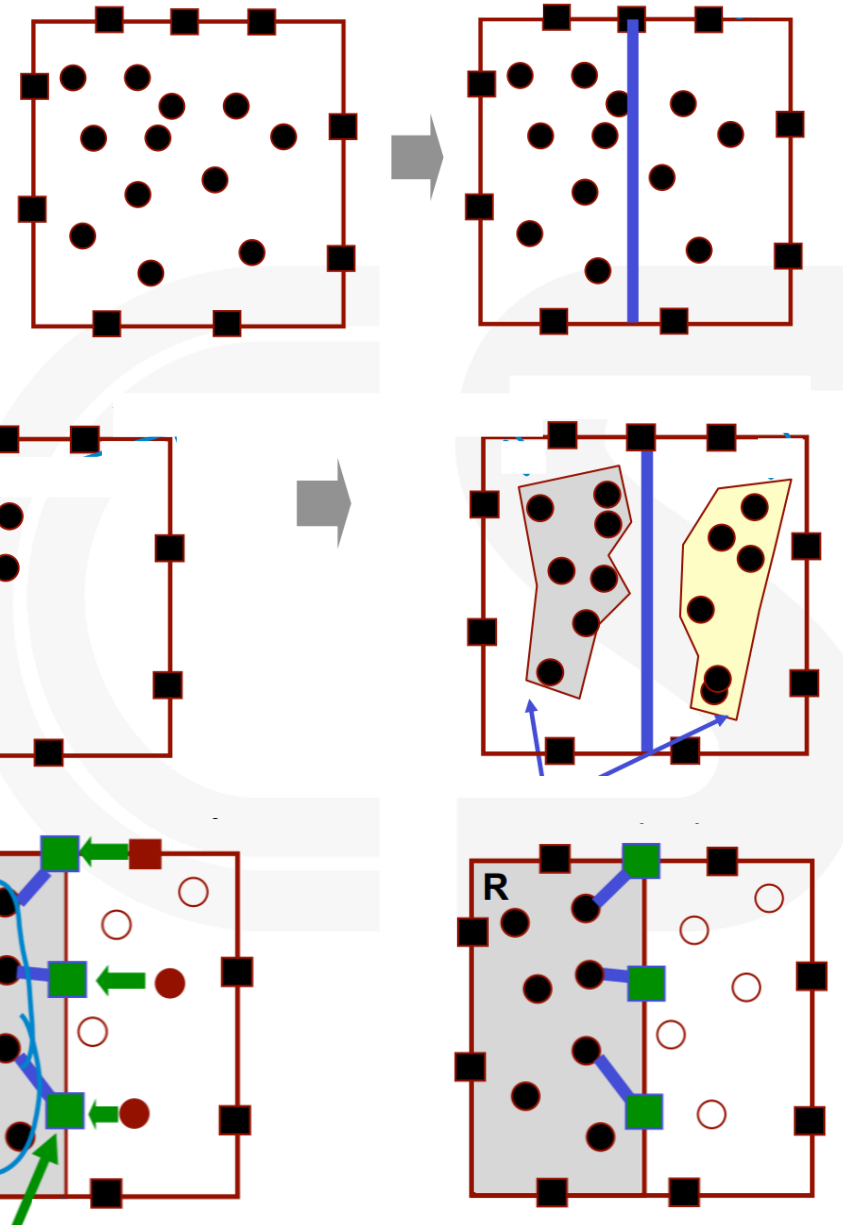
Keep going until the partitions are small enough.



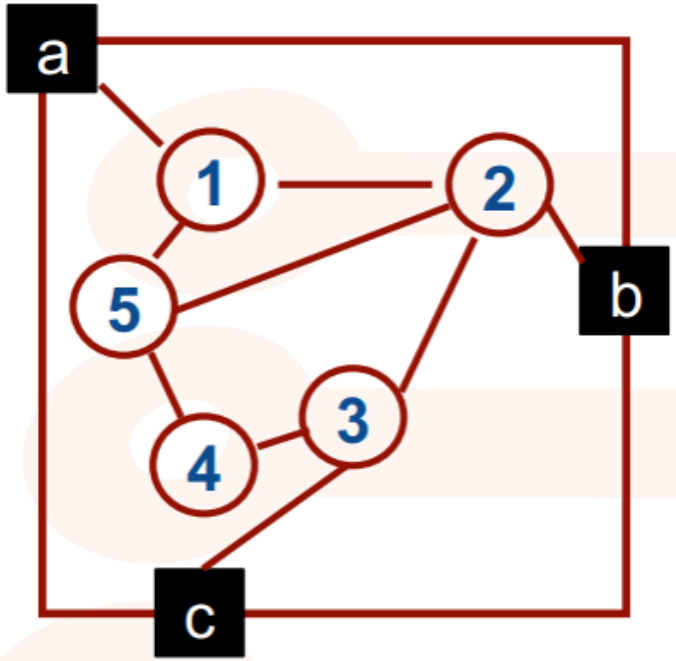
Source: Rutenbar, IBM

Recursive Partitioning

- **Partition**
  - Divide the chip into new, smaller placement tasks.
  - Divide it in half!
- **Assignment**
  - Assign gates into new, smaller region.
  - Sort the gates and distribute to each half.
- **Containment**
  - Formulate new QP matrix that keeps gates in new regions.
  - Create "pseudo pads" –
    - Every gate and pad NOT inside the partition R is modeled as a pad on the boundary of R.
    - Propagate the pseudo pads to the their nearest point on R.



Recursive Partitioning Example

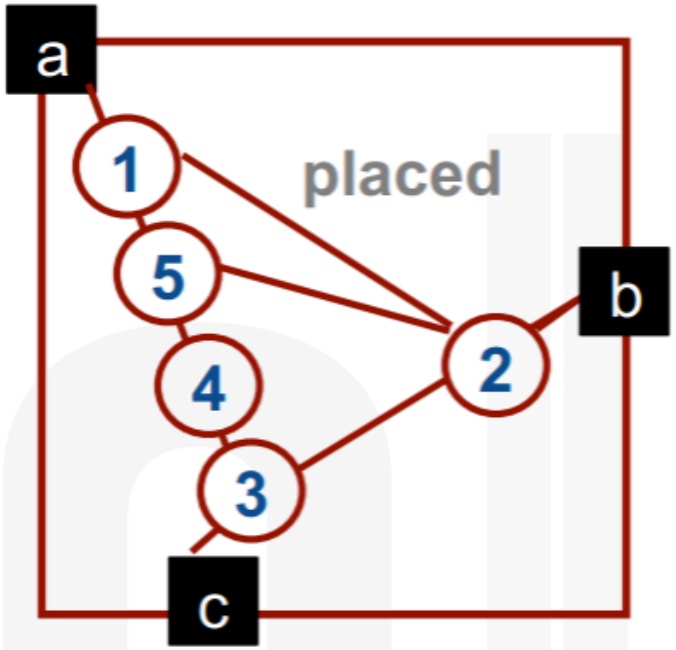


1. Initial netlist

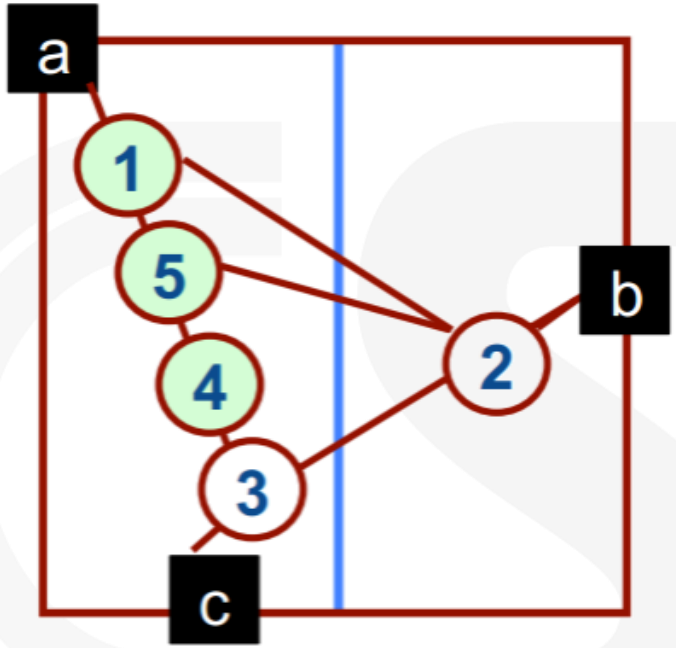
5 gates (1,2,3,4,5)

9 wires

3 pads (a,b,c)



2. Initial QP



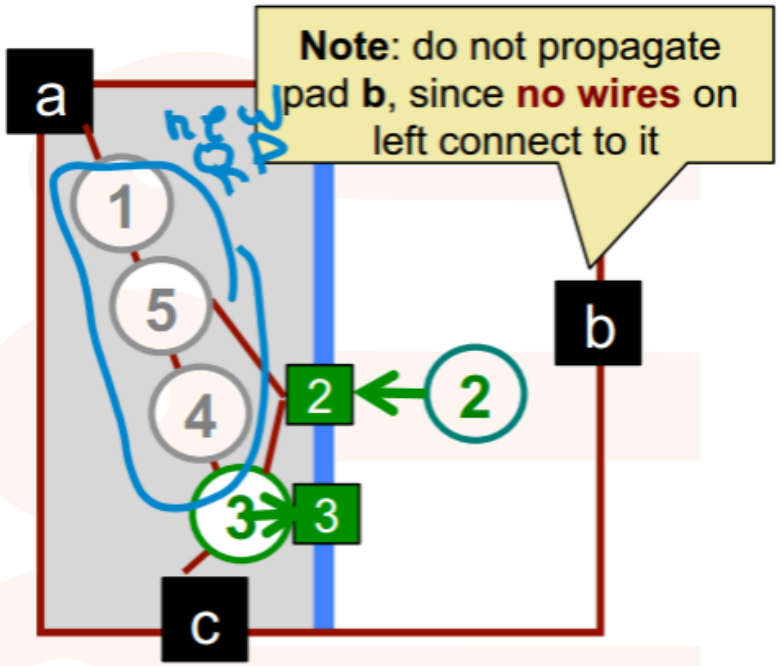
3. First partition

Sort on **X**:

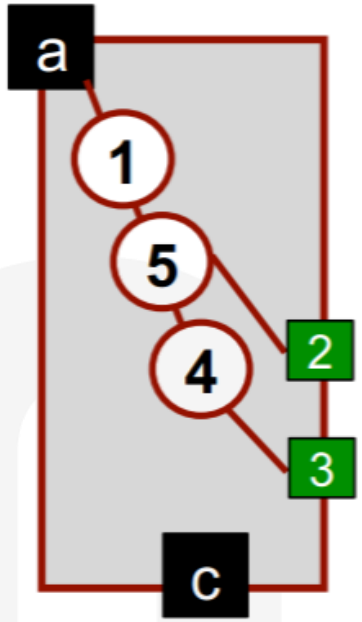
Gate order 1 5 4 3 2

Pick: **1 5 4** on left

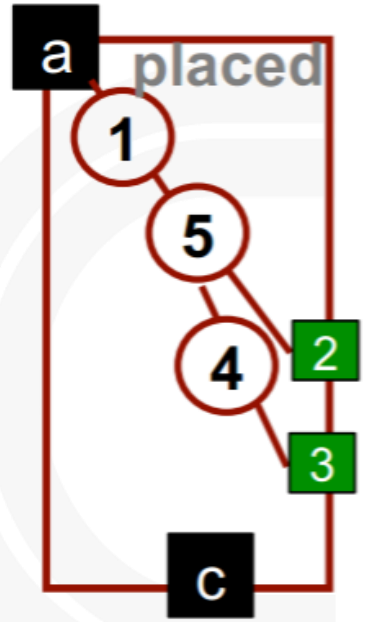
Recursive Partitioning Example



4. Propagate gates/pads
Right-side gates: 2,3
Right-side pads: b
Push to cut, using
y coordinates

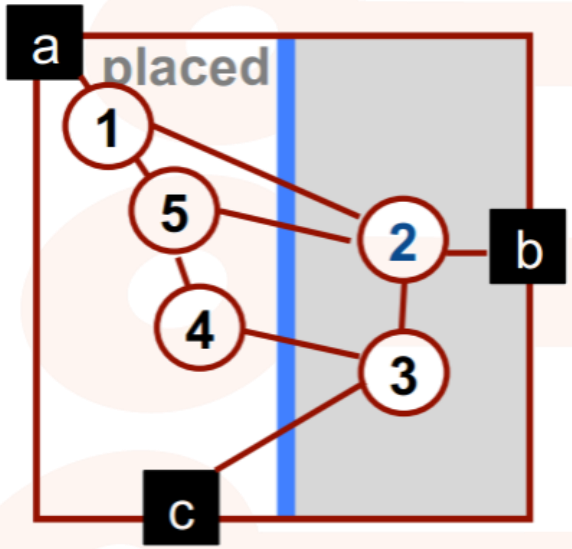


5. 2nd QP input
This is set up for this new smaller placement

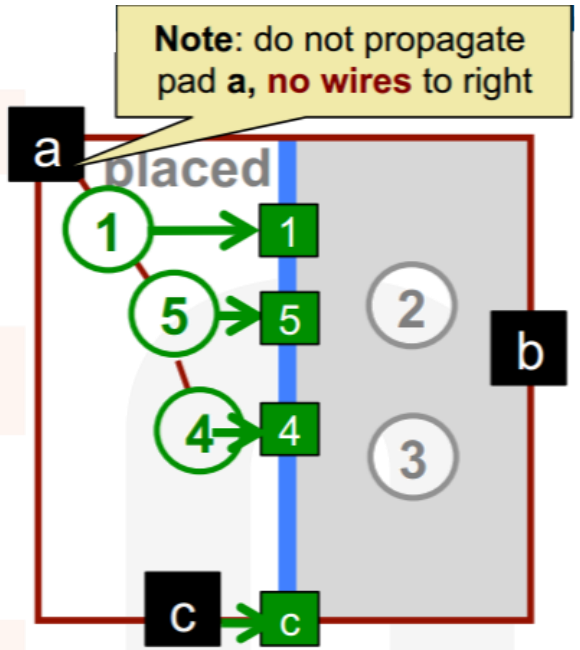


6. 2nd QP solved
New placement

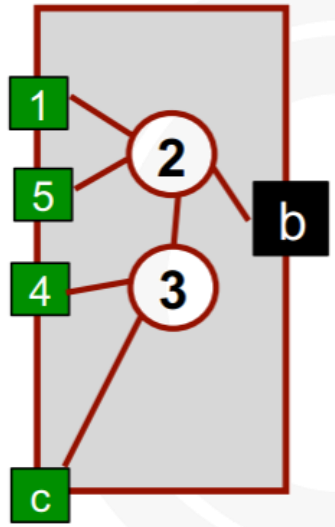
Recursive Partitioning Example



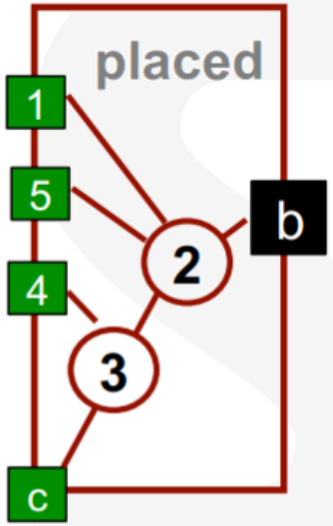
7. Left side placed.
Now, re-place right-side gates.



8. Propagate gates/pads
This is set up for next, new smaller placement



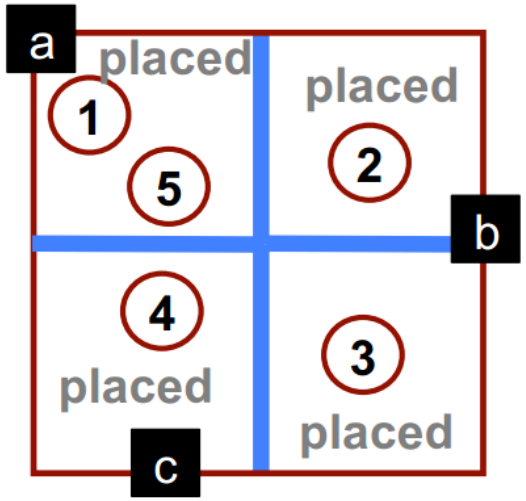
9. 3nd QP input
This is set up for this new smaller placement

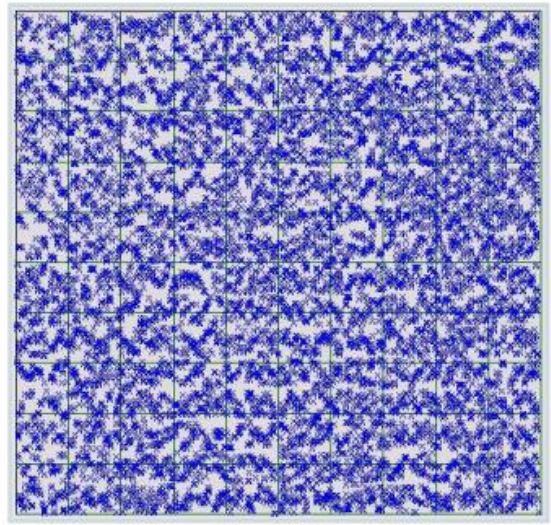


10. 3nd QP solve

Placement Legalization

- **Keep recursively partitioning…**
  - Usually, continue until you have a "small" number of gates in each region (i.e., 10-100 gates)
  - In these regions we will still have overlaps
- **Still need to force gates in precise rows for final result**
  - This is known as "**legalization**"
  - One easy way to do this is simulated annealing!
  - Just use a low temperature to start with, so you don't make drastic moves.



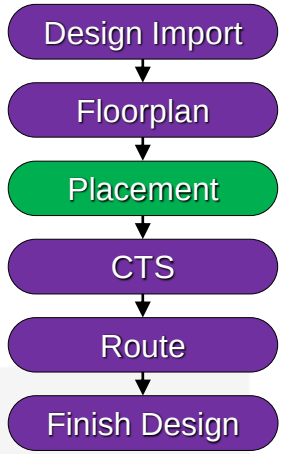


Placement in Practice

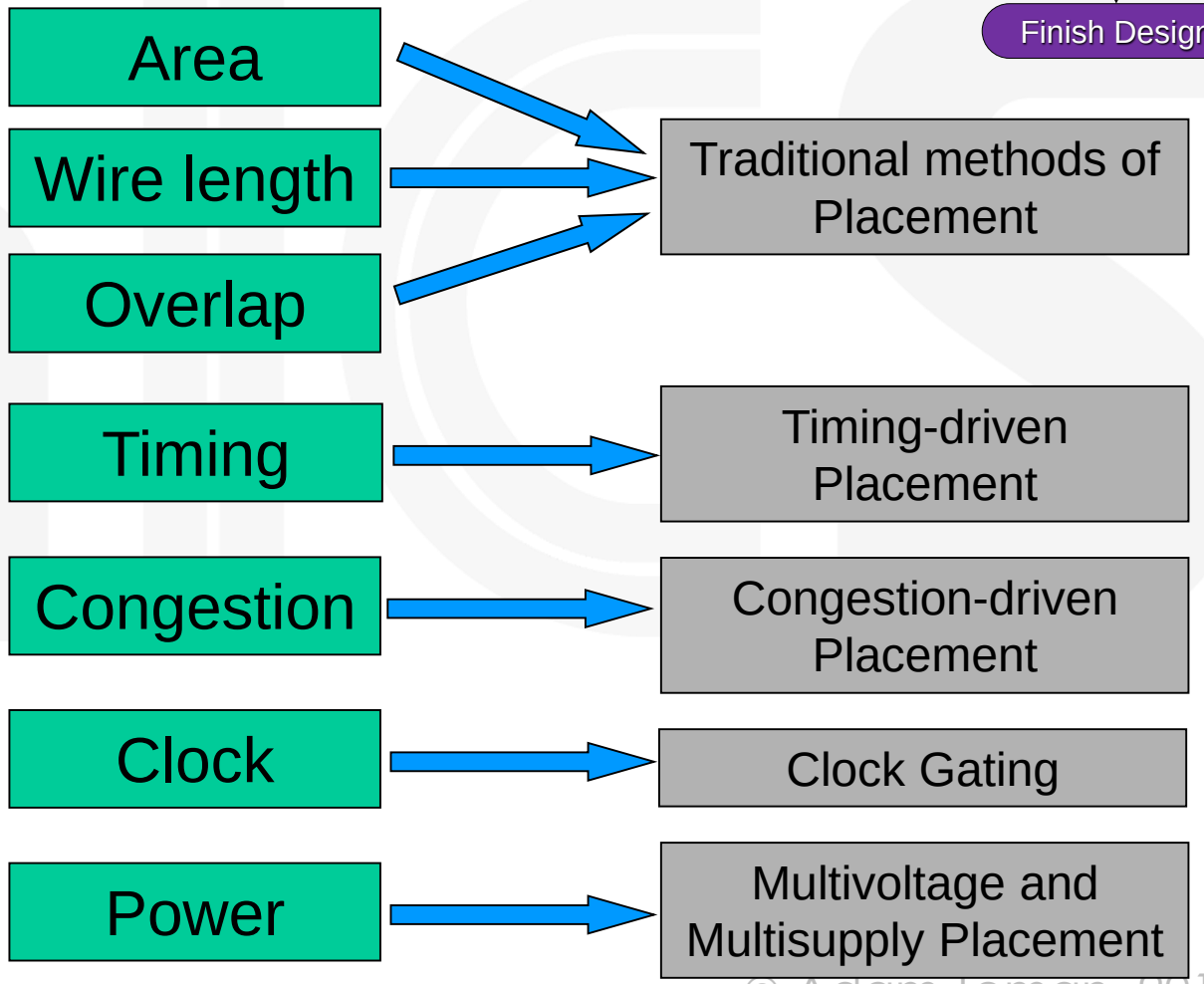
Emerging Nanoscaled Integrated Circuits and Systems Labs

Bar-Ilan University
אוניברסיטת בר-אילן

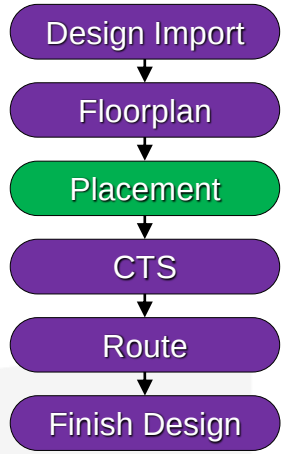
Placement Targets



- **In addition to wire-length minimization, placement can be driven by two additional primary targets:**
  - Timing Optimization (Timing-driven placement)
  - Congestion Minimization (Congestion-driven placement)
- **Further targets include clock tree and power optimizations**

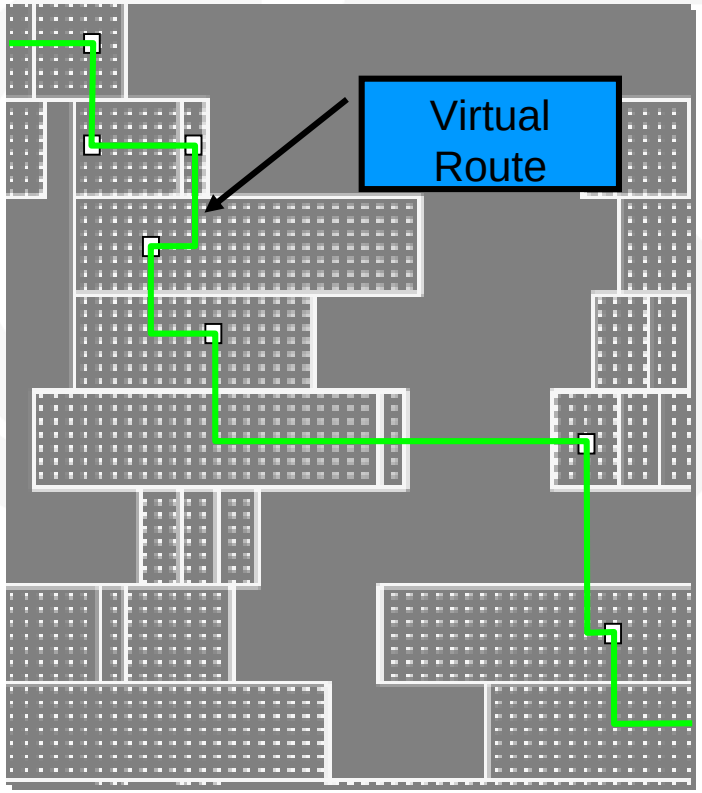


Timing-Driven Placement

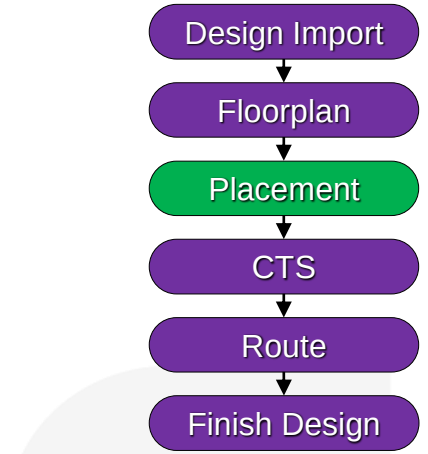


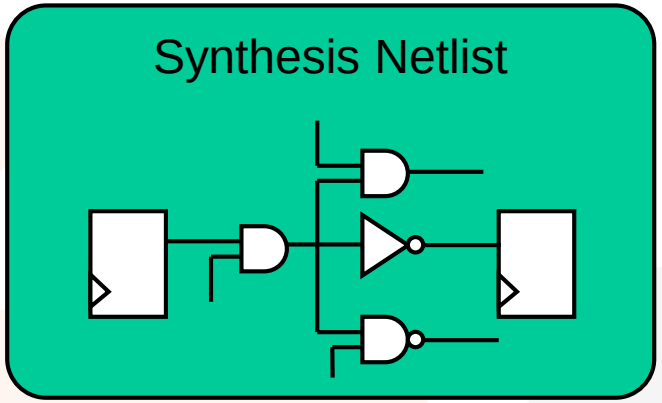
- **Timing-driven placement tries to place critical path cells close together to reduce net RCs and to meet setup timing**

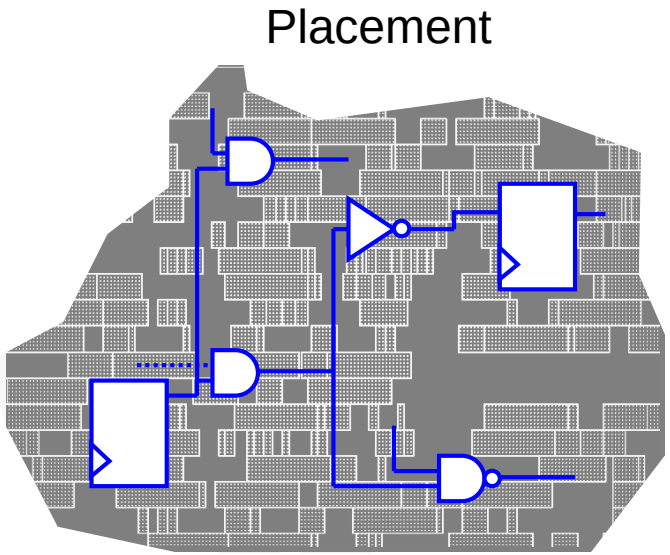
- **RCs are based on Virtual Route (VR)**
  - Layers are not taken into consideration
- **Timing-driven placement based on Virtual Route**
  - Tries to place cells along timing-critical paths close together to reduce net RCs and meet setup timing
  - Net RCs are based on Virtual Routing estimates



Timing-Driven Placement





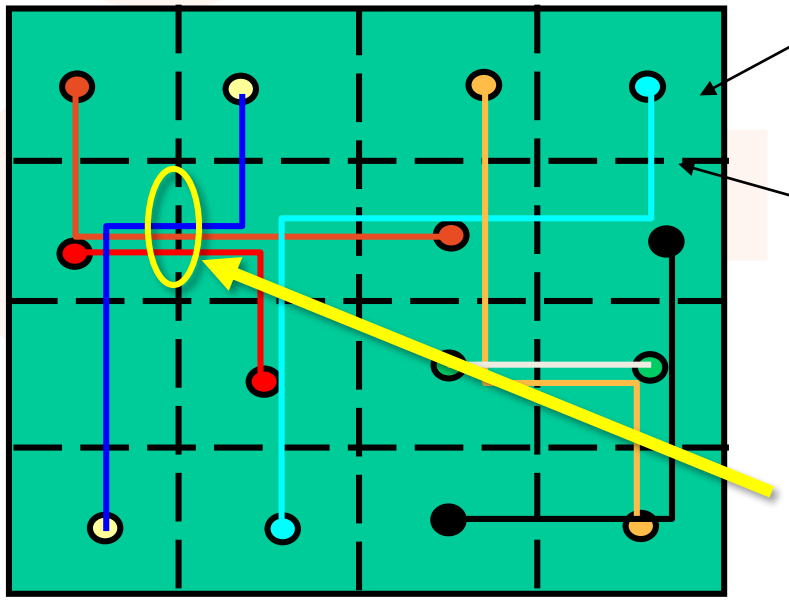


Statistically Based
Wire Load Model (WLM)

Net Fanout	Resistance KΩ	Capacitance pF
1	0.0498	0.045
2	0.1295	0.0812
3	0.2092	0.1312
4	0.2888	0.1811

Congestion

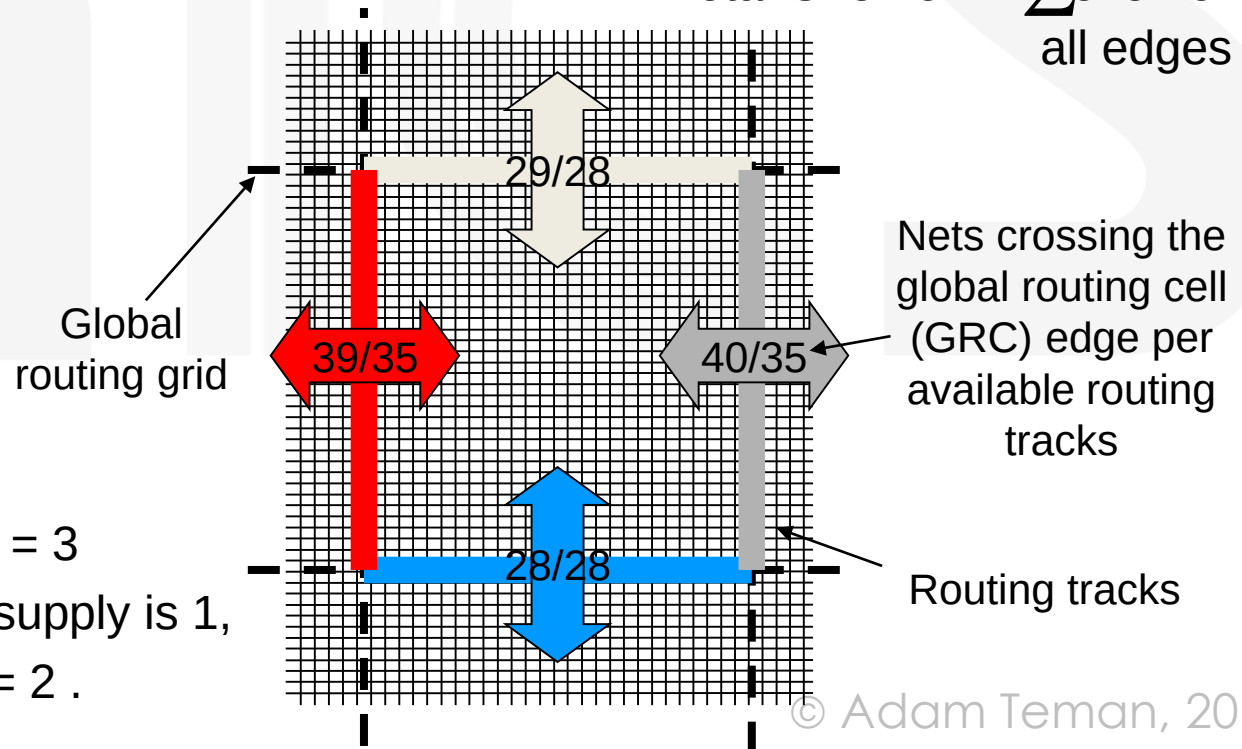
- **Congestion occurs when the number of required routing tracks exceeds the number of available tracks.**
  - Congestion can be estimated from the results of a quick global route.
  - Global bins with routing overflow can be identified.



Overflow on each edge =

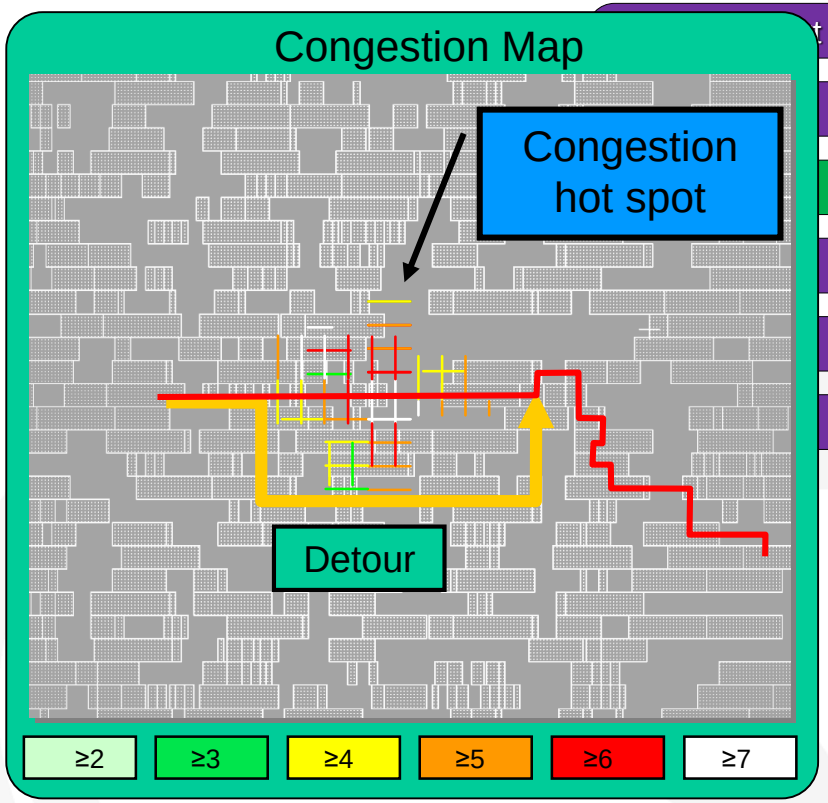
$$\begin{cases} \text{Routing Demand - Routing Supply} \\ 0 \quad \text{(otherwise)} \end{cases}$$

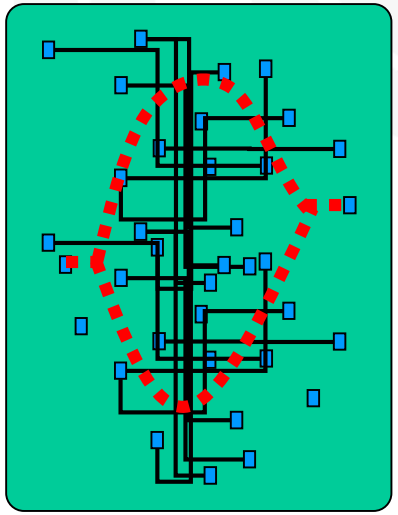
$$\text{Total Overflow} = \sum_{\text{all edges}} \text{overflow}$$



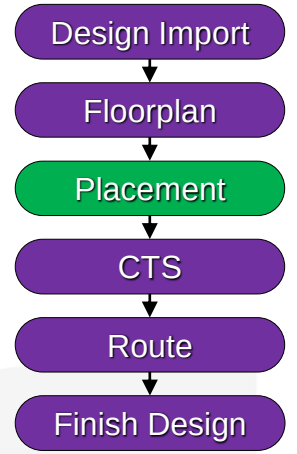
Congestion

- **Issues with Congestion**
  - If congestion is not too severe, the actual route can be detoured around the congested area
  - The detoured nets will have worse RC delay compared to the VR estimates
  - In highly congested areas, delay estimates during placement will be optimistic.
- **Not routable or severely congested design**
  - It is important to minimize or eliminate congestion before continuing
  - Severe congestion can cause a design to be un-routable

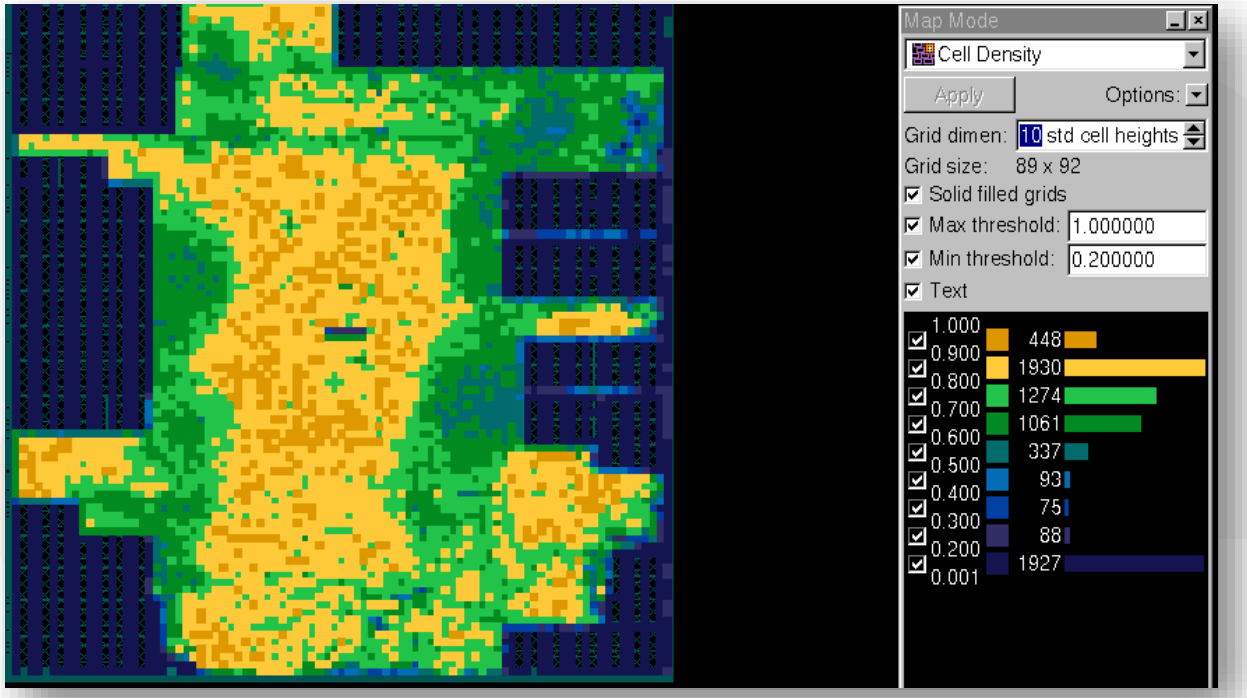


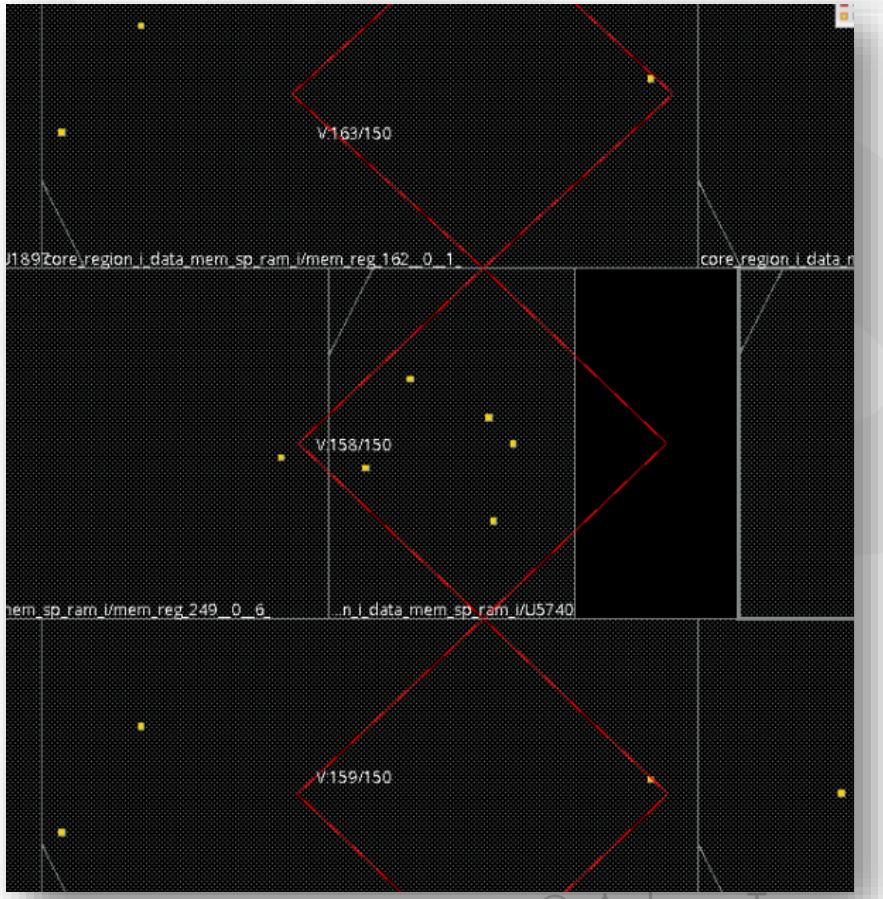


Congestion Maps



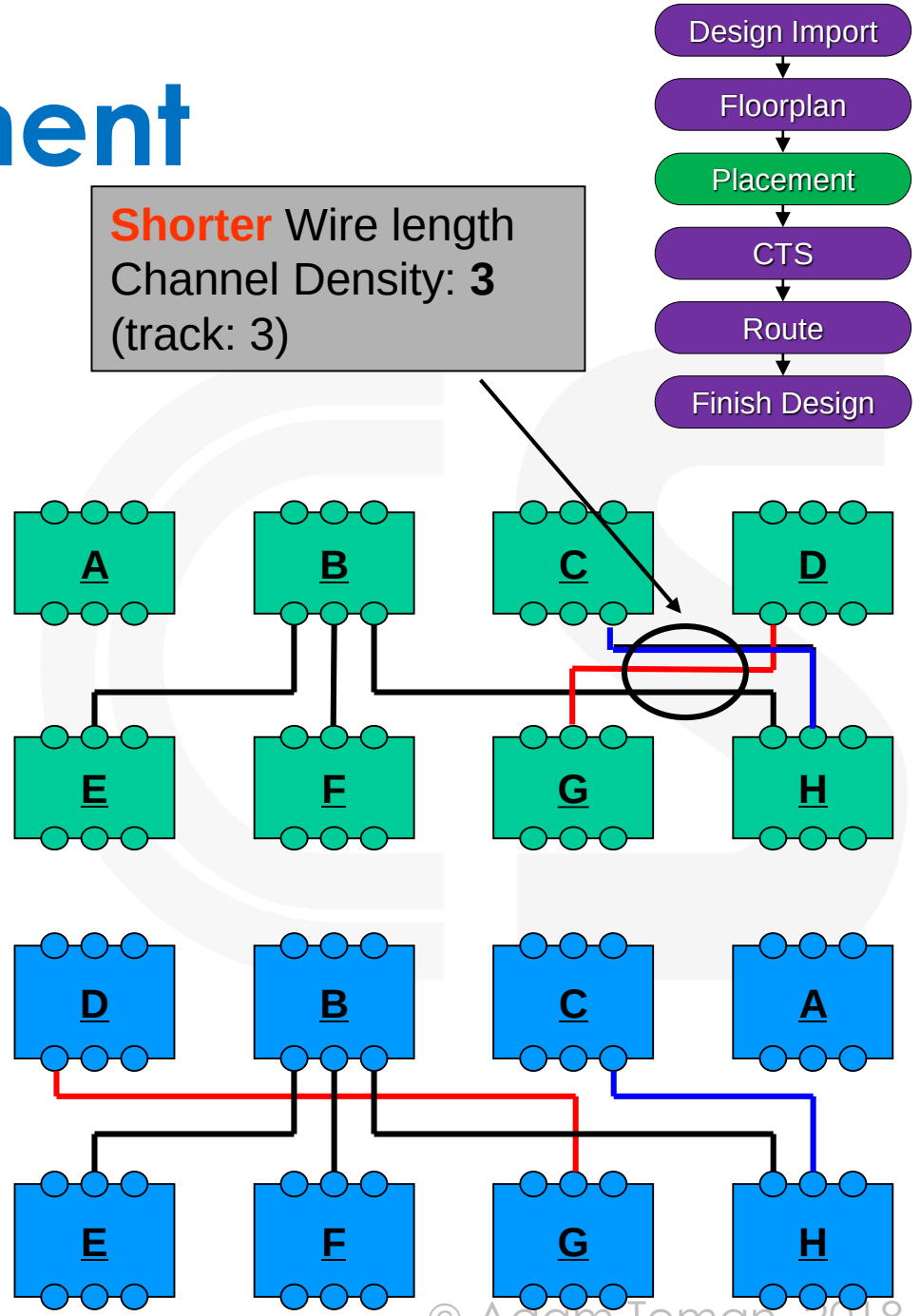
- **Congestion maps are displayed by the backend tool to help us evaluate the total congestion, identify and fix congestion hot spots.**



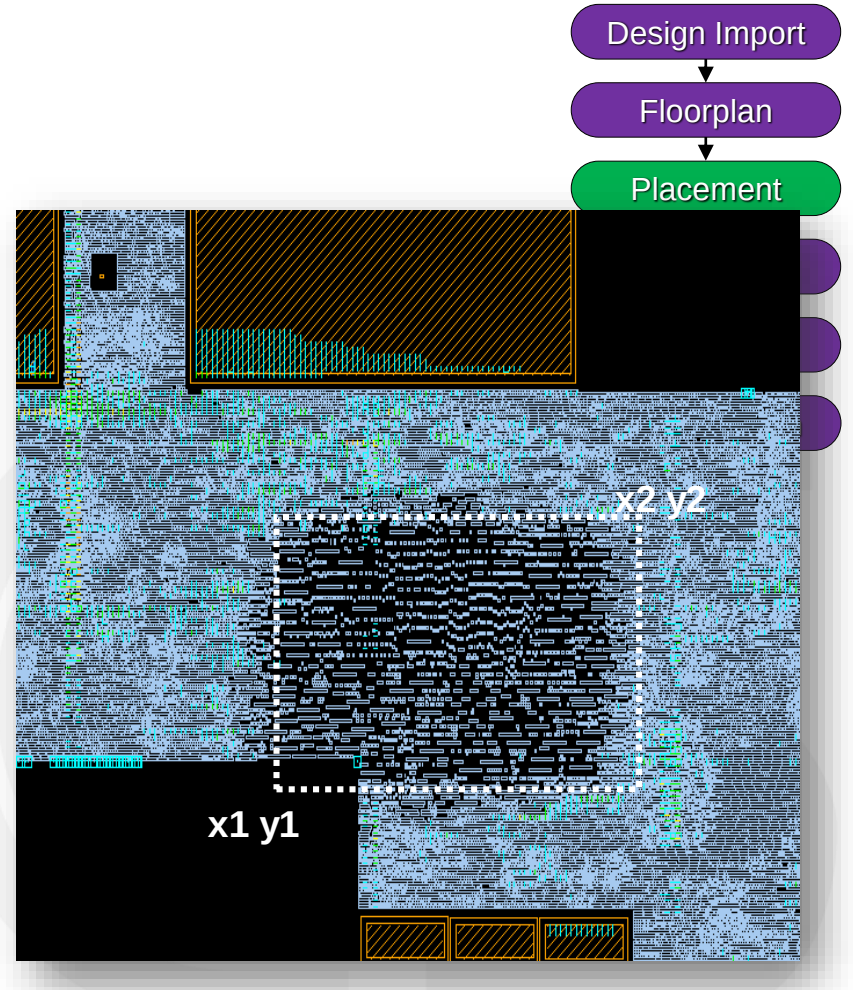


Congestion-driven Placement

- **Congestion Reduction**
  - The tool tries to evaluate congestion hotspots and spread the cells (lower utilization) in the area to reduce congestion.
  - The tool can also choose cell location based on congestion, rather than wire-length.



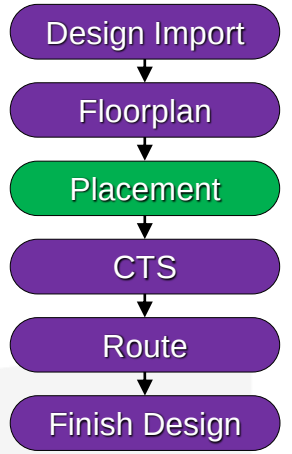
Strategies to Fix Congestion



Modify the floorplan:

- **Mark areas for low utilization.**
- **Top-level ports**
  - Changing to a different metal layer
  - Spreading them out, re-ordering or moving to other sides
- **Macro location or orientation**
  - Alignment of bus signal pins
  - Increase of spacing between macros
  - Add blockages and halos
- **Core aspect ratio and size**
  - Making block taller to add more horizontal routing resources
  - Increase of the block size to reduce overall congestion
- **Power grid**
  - Fixing any routed or non-preferred layers

Placement in Innovus

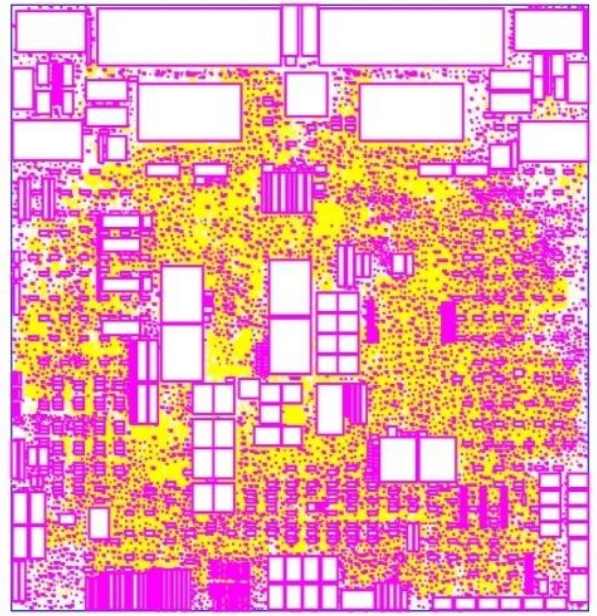


- **In the traditional flow, placement was achieved in two steps:**
  - Placement:

```
set_db place_global_cong_effort high
place_design
```

  - Post Placement Optimization

```
set_db opt_effort high
opt_design -pre_cts
```



- **In Innovus, this has been replaced by gigaPlace, which runs timing driven concurrent placement and optimization**

```
set_db place_global_cong_effort high
place_opt_design
```

- **If you manually move something and need to legalize placement:**

```
place_detail
```

Main References

- Ron Rutenbar “From Logic to Layout”
- Synopsys University Courseware
- IDESA
- Cadence Documentation