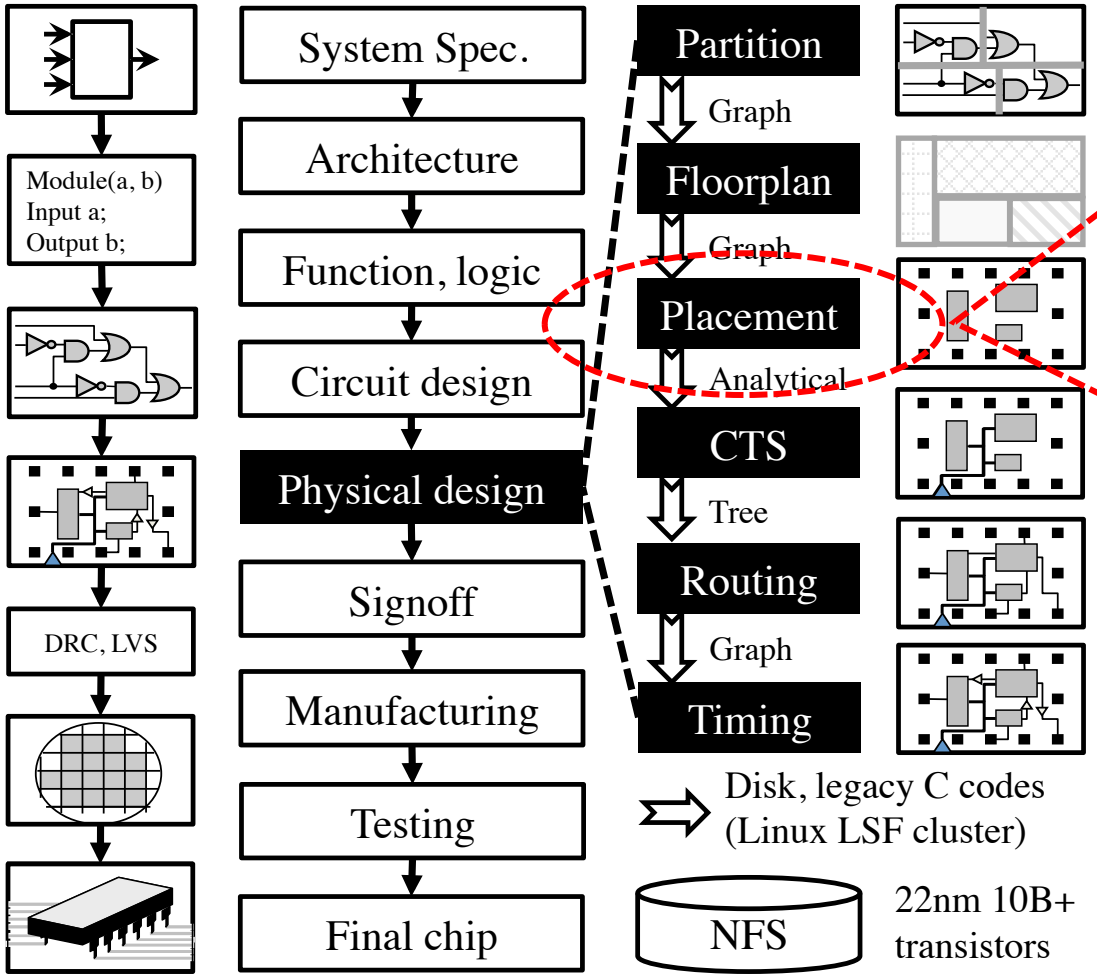
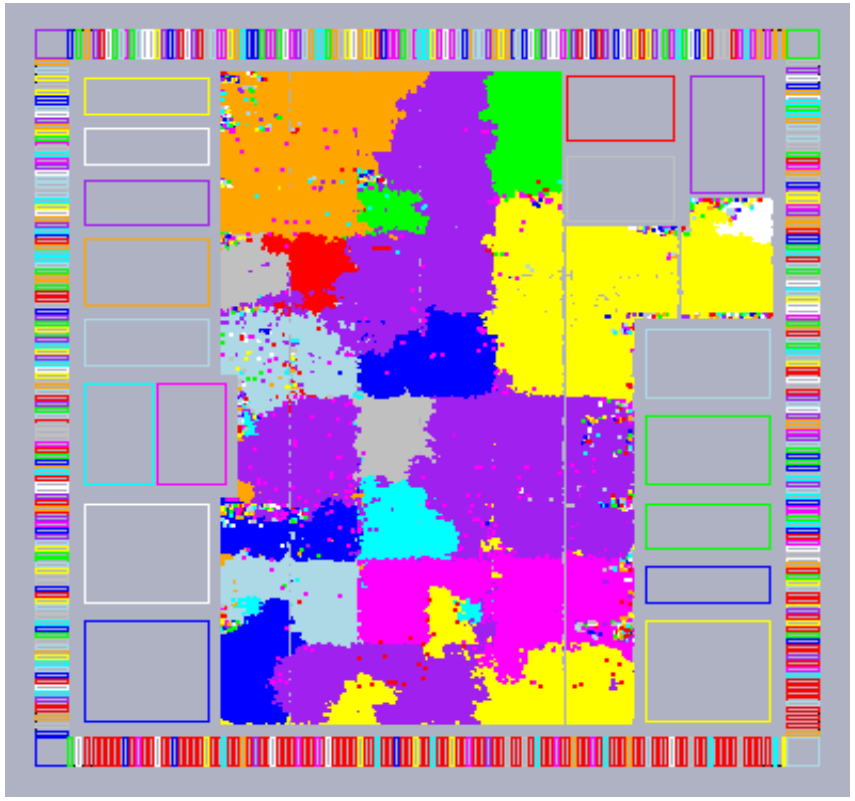
# Lecture 11: Placement – I

Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

# Please Try to Attend Friday's Seminar!

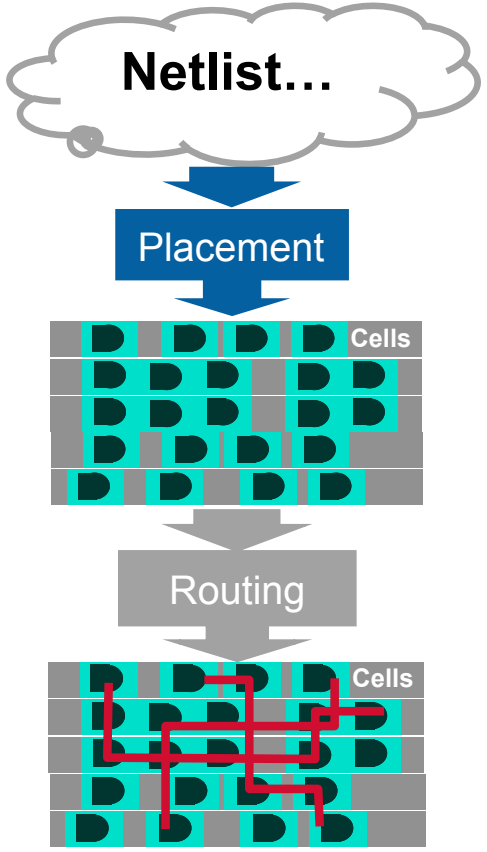
# Physical Design Flow

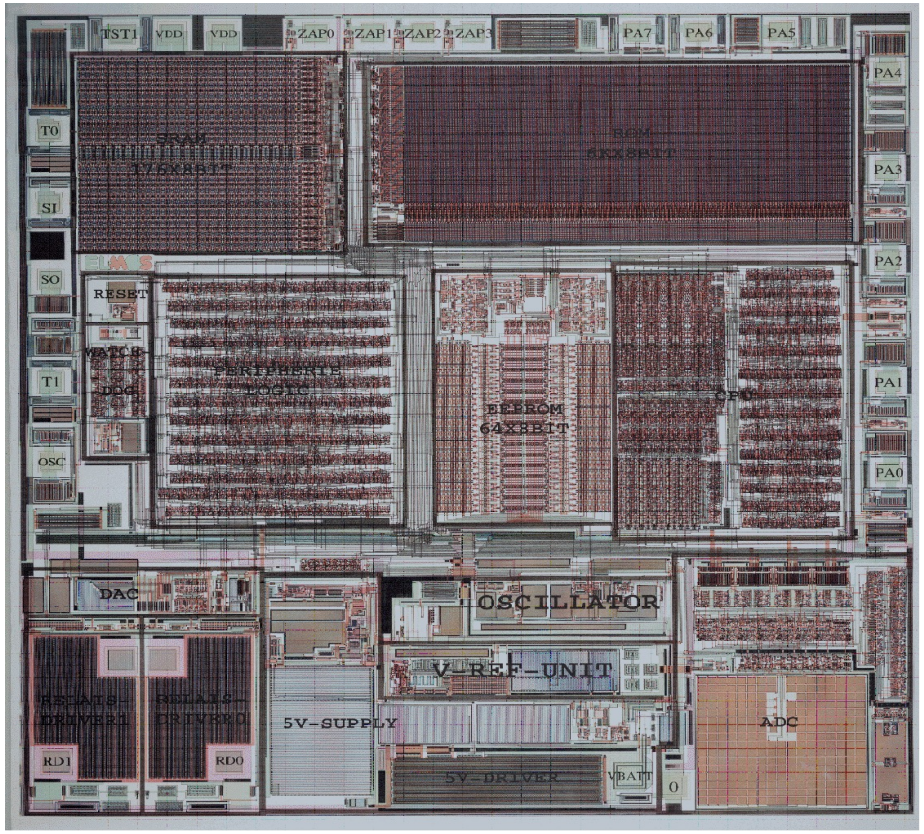


# Placement

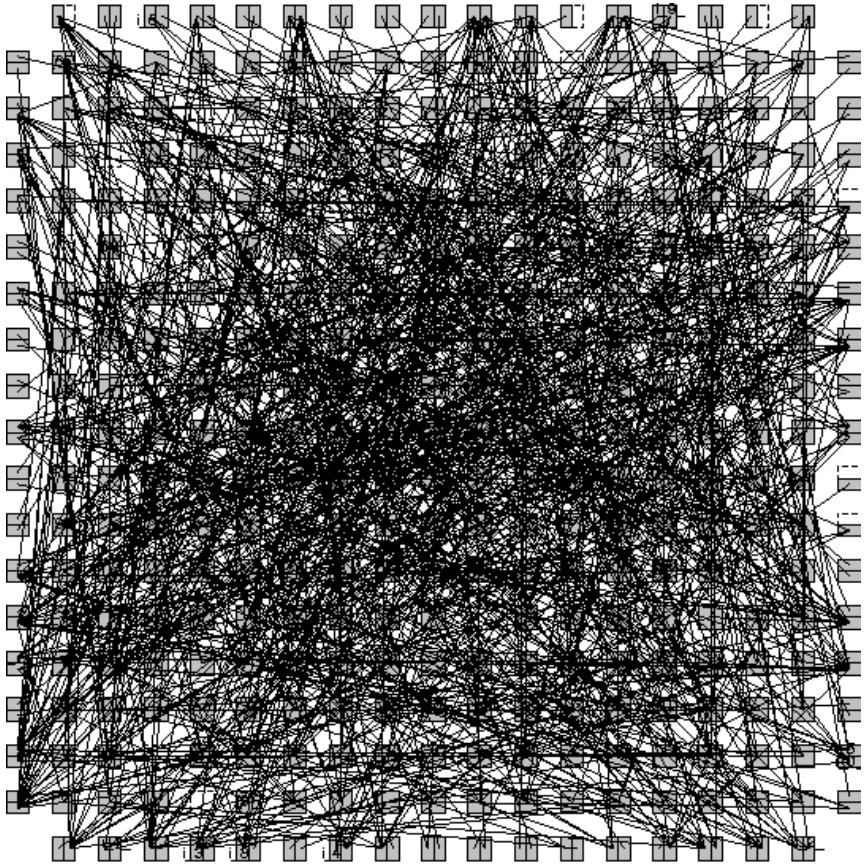
- **Places a netlist of gates and wires in a chip**
  - Min wirelength
  - Min area
  - Min power
  - Min delay
  - …



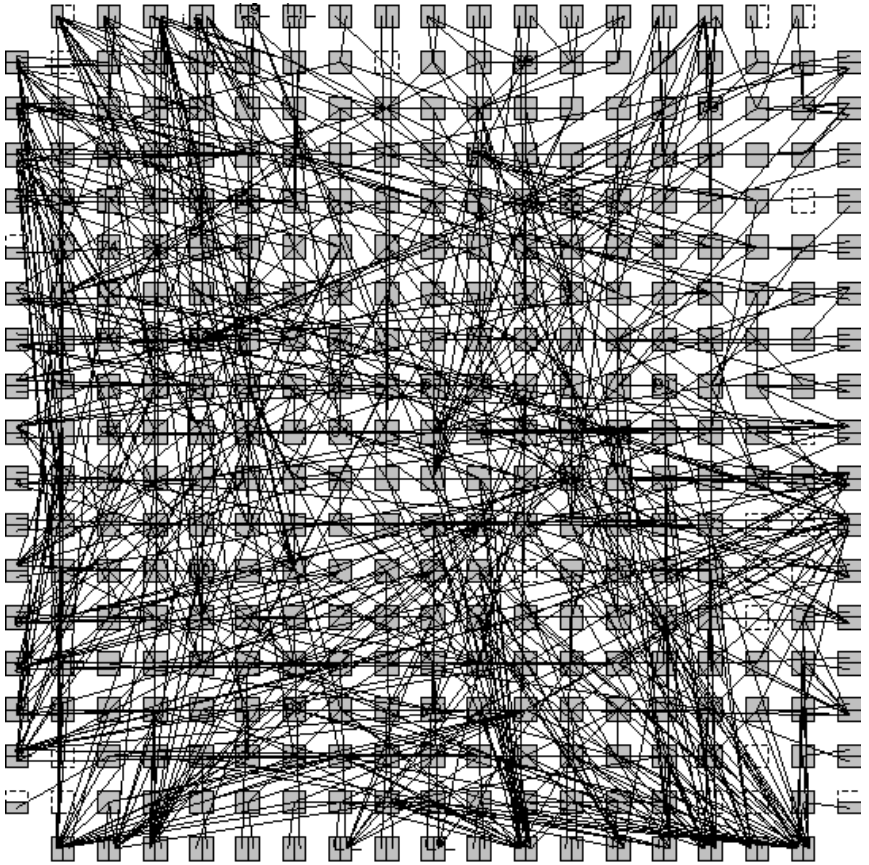



# Example

- Which placement result is better?



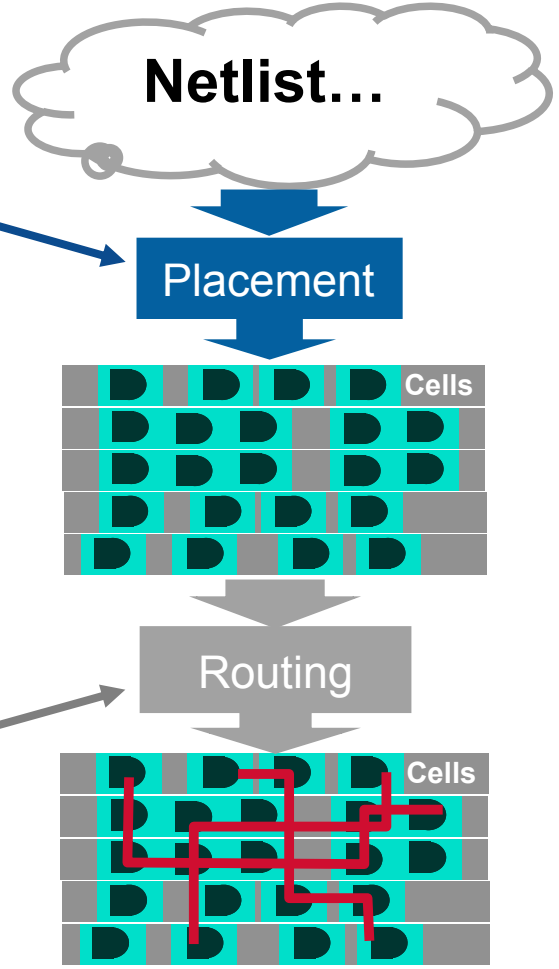
Initial Placement. Cost: 74.5562. Channel Factor: 100



Final Placement. Cost: 28.5384. Channel Factor: 100

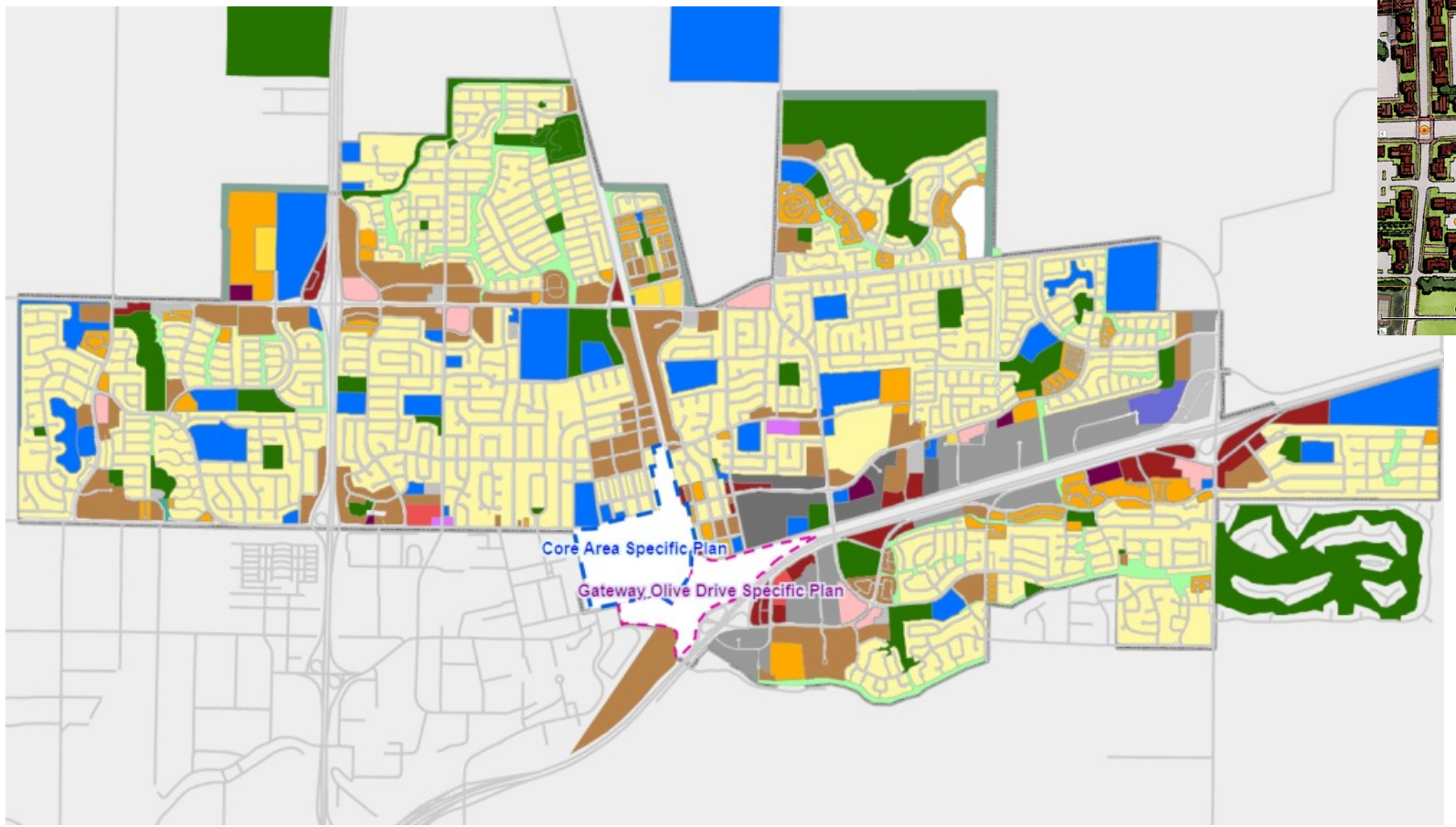
# Placement Problem

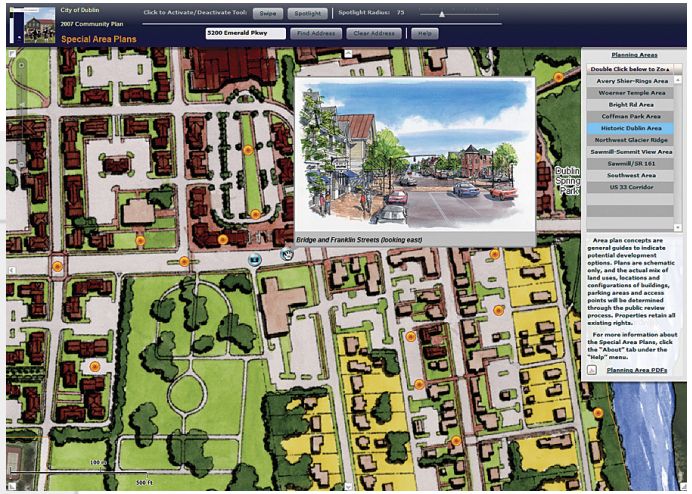
- **What does a placer do?**
  - **Input:** Netlist of gates & nets
  - **Output:** Exact location of each gate
  - **Goal:** Able to **route** (connect) all wires
- **Placement is VERY HARD!**
  - Large designs have 10-500M cells to place!
  - Bad placement → Much more wire
  - More wire: bigger, slower chip
  - If placement is *very* bad, next tool in the flow—the **router**—unable to connect all wires, or meet timing



# Analogy to City Planning

- How to plan a city for 5-10M+ people?



# Requirement for Placement

- **Must handle 10-100M cells, 1000s macros**
  - *64 bits + near-linear asymptotic complexity*
- **Accept fixed ports/pads/pins + fixed cells**
- **Place macros, esp. with var. aspect ratios**
  - Non-trivial heights and widths (ex: height=2rows)
- **Honor targets and limits for net length**
- **Respect floorplan constraints**
- **Handle a wide range of placement densities**
  - From <25% to 100% occupied

# Requirement for Placement (cont'd)

- **Add / delete filler cells and Nwell contacts**
- **Ignore clock connections**
- **ECO (Engineering Change Order) placement**
  - Fix overlaps after logic restructuring
  - Place a small number of unplaced blocks
- **Datapath planning services**
  - Ex: many-core processor units
- **Provide dialog services to enable cooperation across tools**
  - Ex: between placement and synthesis

# Importance of Placement

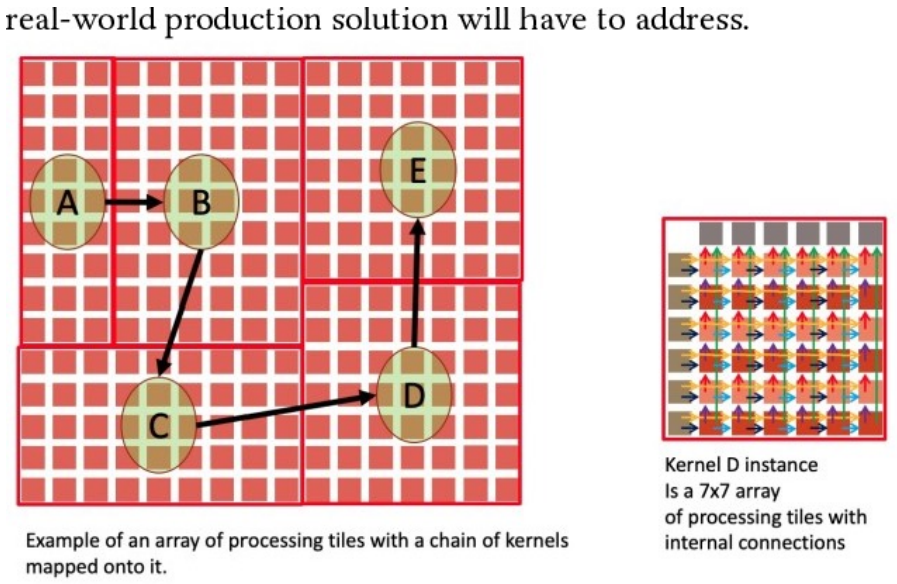
- **Placement is a fundamental problem for physical design**
- **Becomes very active again in recent years:**
  - Many new academic placers since 2000
  - Many other publications to handle timing, routability, etc.
- **Reasons:**
  - Serious interconnect issues (delay, routability, etc.) in advanced nodes
    - Placement determines interconnect to the first order
    - Need placement information even in early design stages (ex: logic synthesis)
  - Placement problem becomes significantly larger
  - Cong et al. [ASPDAC-03, ISPD-03, ICCAD-03] point out that existing placers are far from optimal, not scalable, and not stable

# Industry-Academia Collaboration



# Placement for Different Design Styles

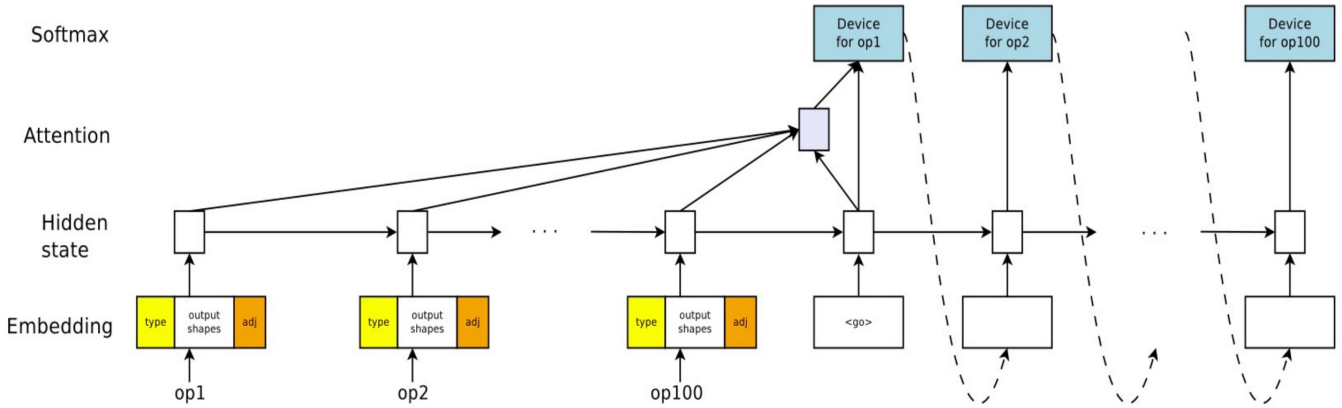
- **Standard cell placement (main focus of this course!)**
- **Gate array / FPGA placement**
- **Macro block placement**
  - Typically called floorplanning
- **Mixed-size placement**
  - A few large macro blocks
  - A large number of small cells
- **Wafer-scale placement**
  - Thousands of processor units



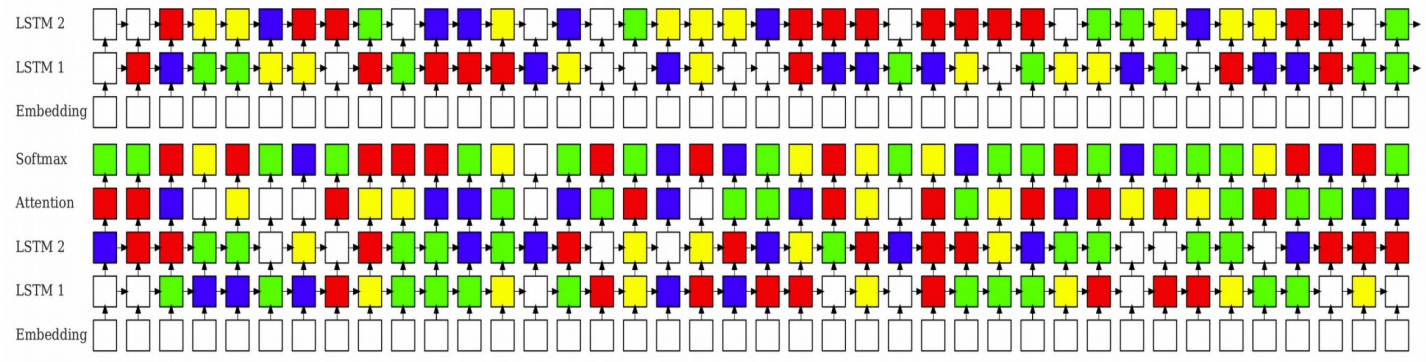

Wafer-scale placement:
https://github.com/Cerebras/ispd2021

# Placement in Machine Learning

- GPU device placement optimization for neural networks



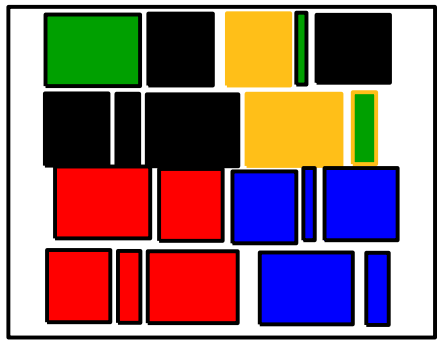

Place each tensor operator onto a GPU for improved performance



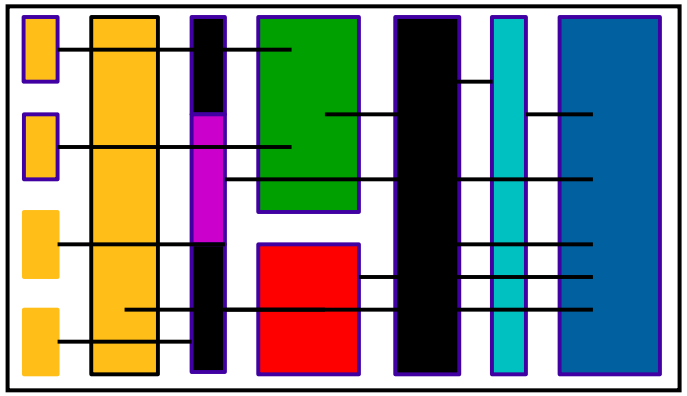

https://arxiv.org/abs/1706.04972

# Placement Footprints

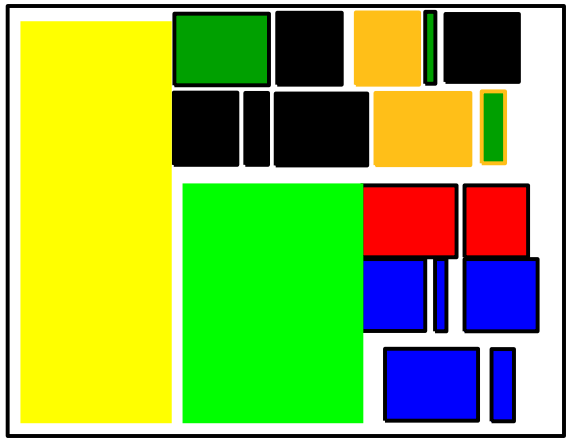
Standard Cell



Data Path

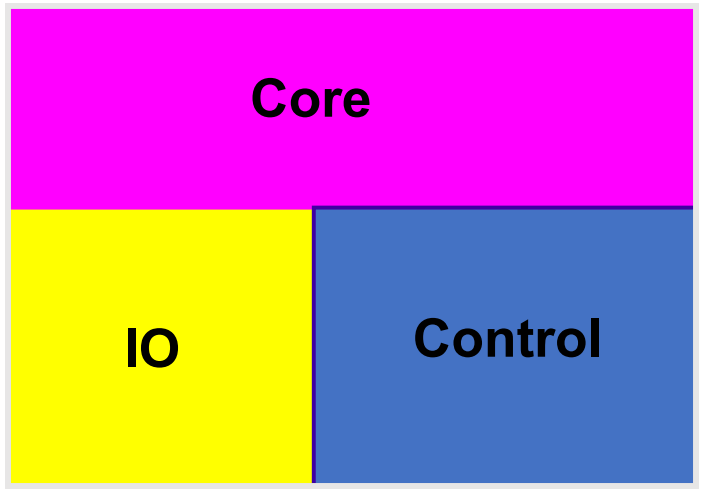


IP - Floorplanning

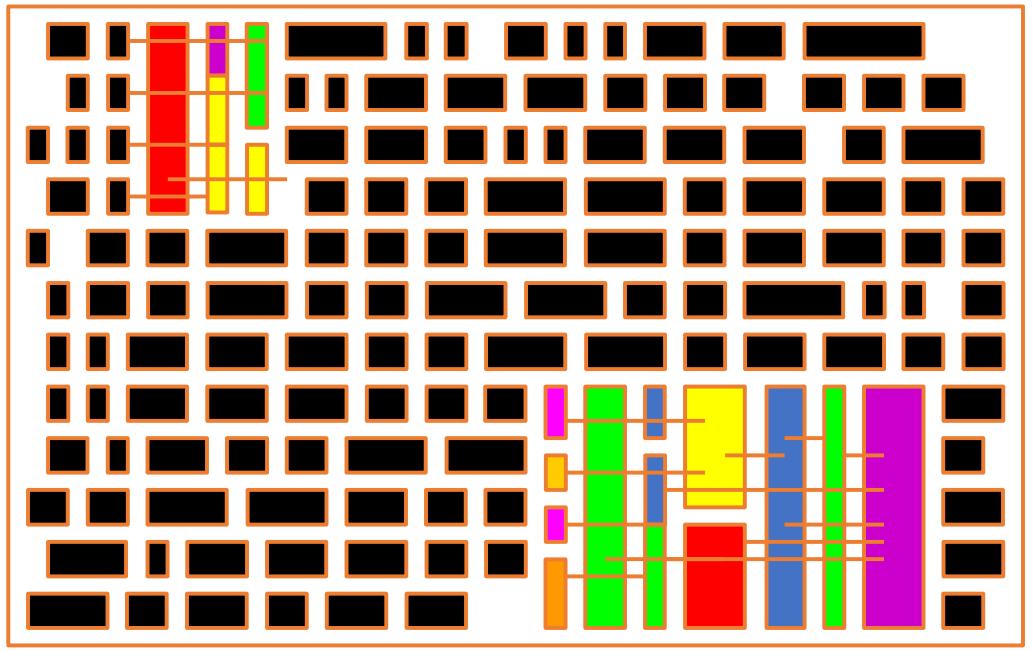


# Placement Footprints (cont'd)

Reserved areas

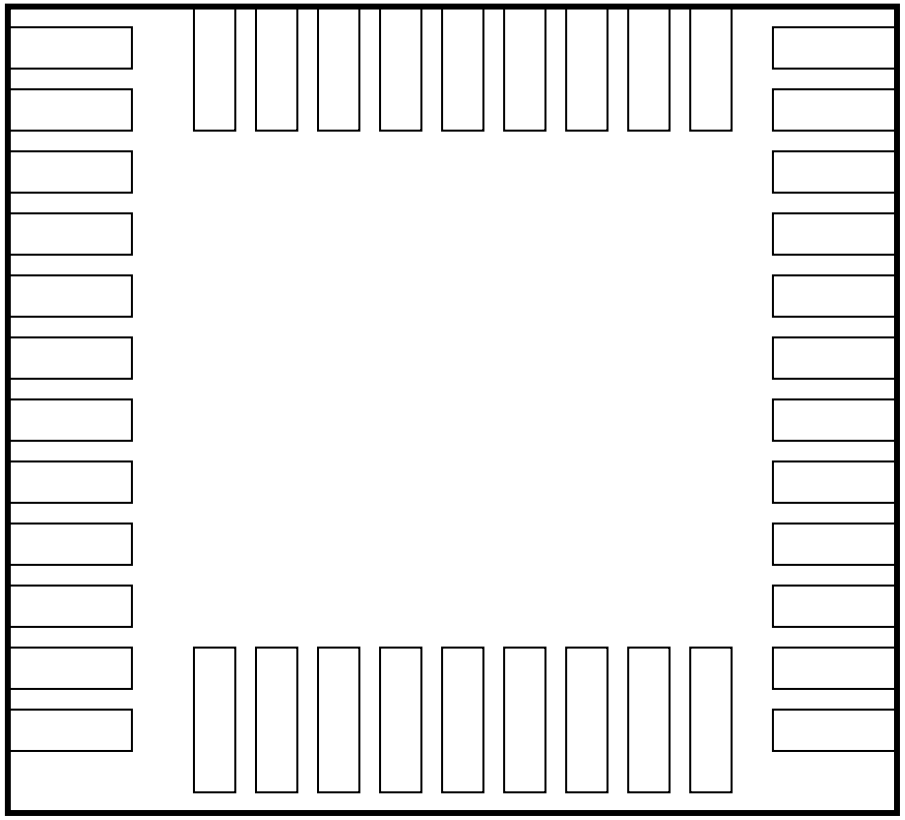



Mixed data path & sea of gates

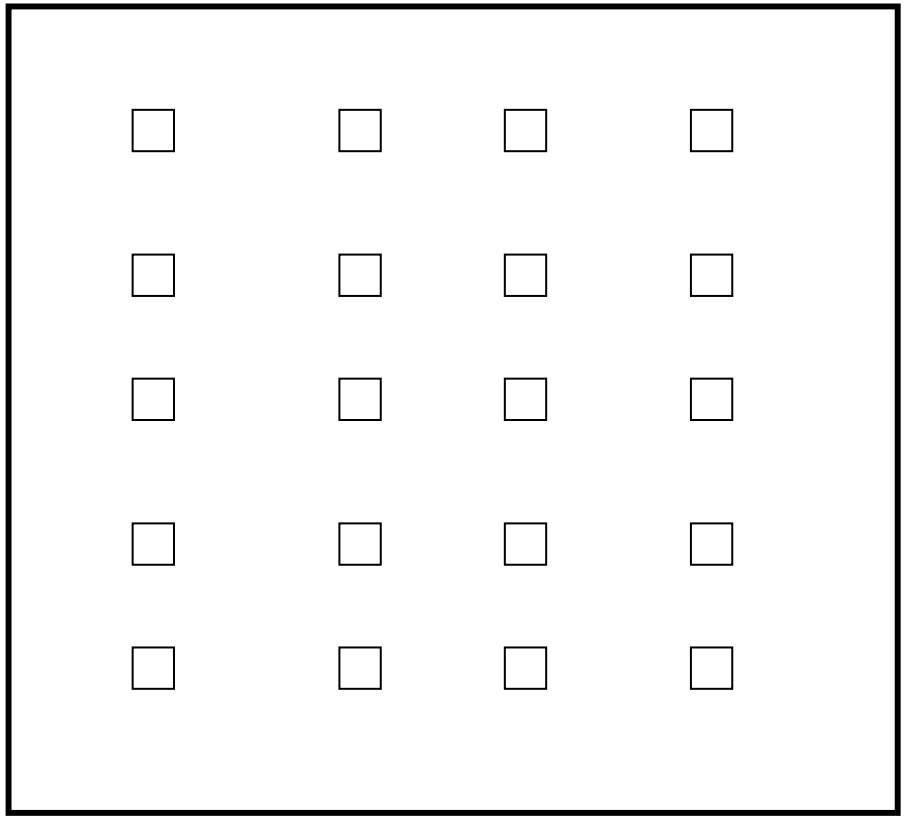


# Placement Footprints

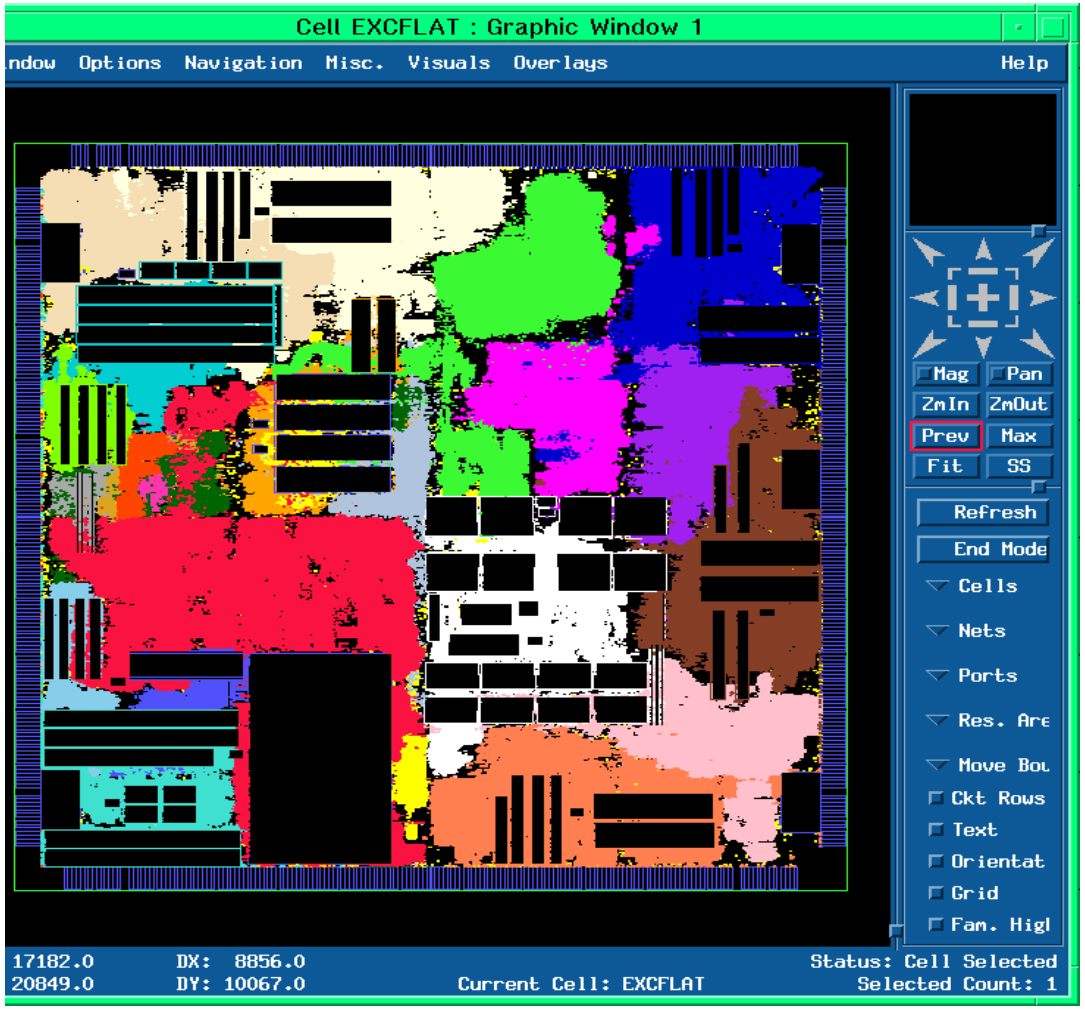
Perimeter IO

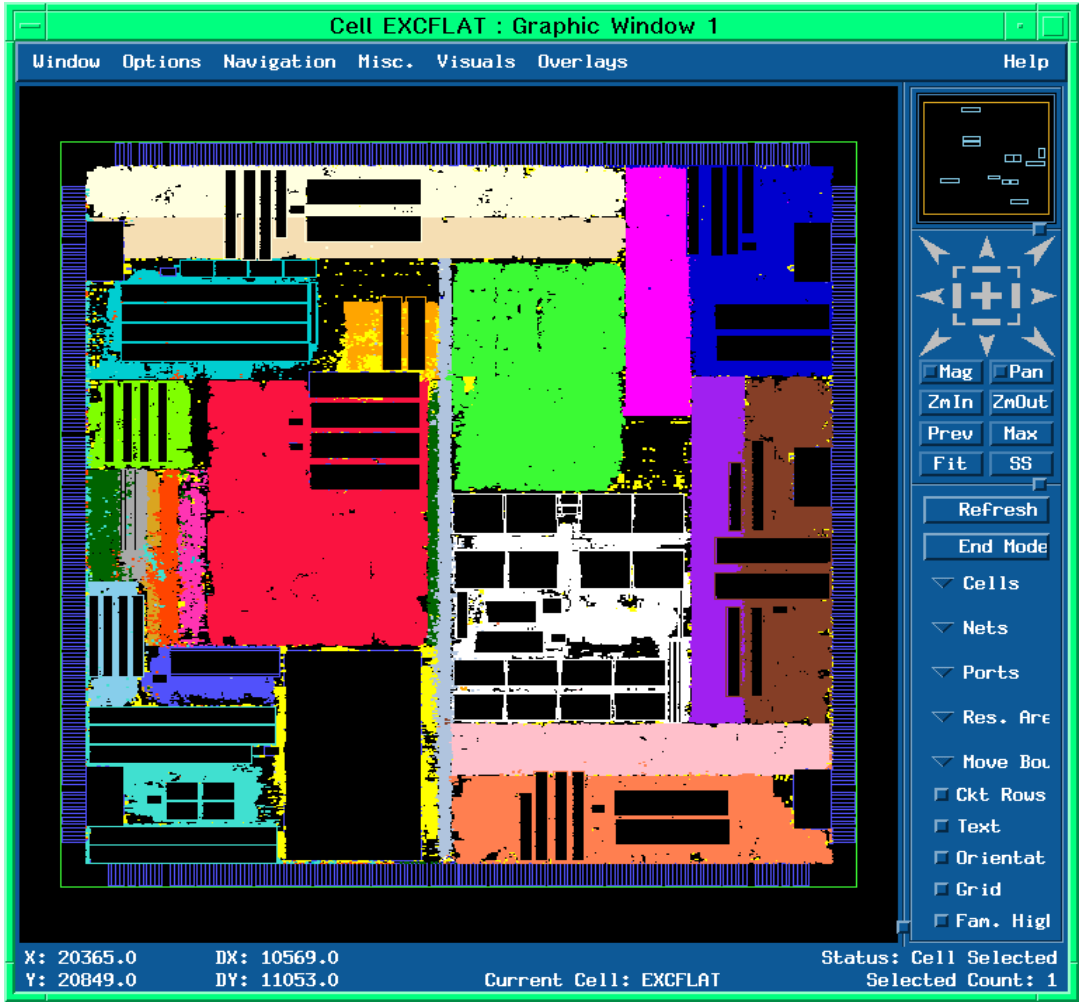


Area IO



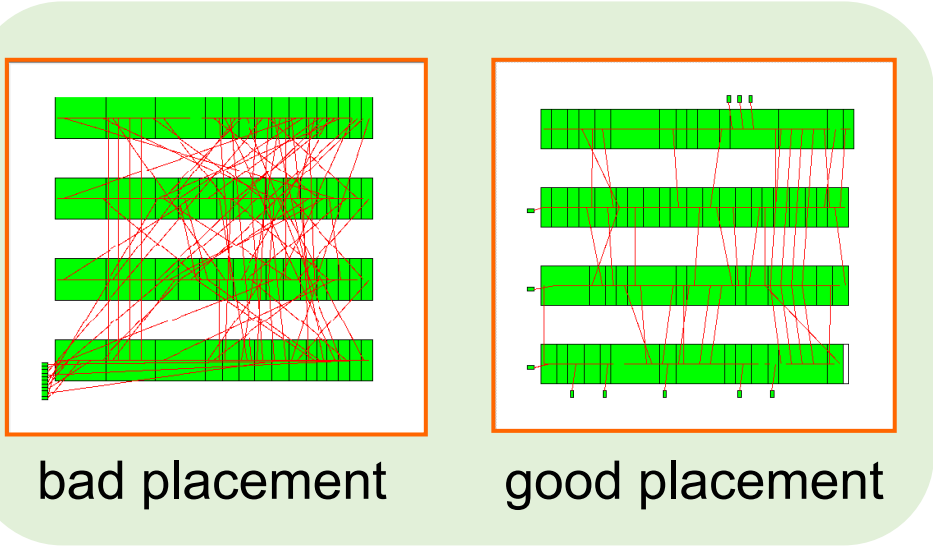
# Commercial Tools



# Placement is VERY HARD

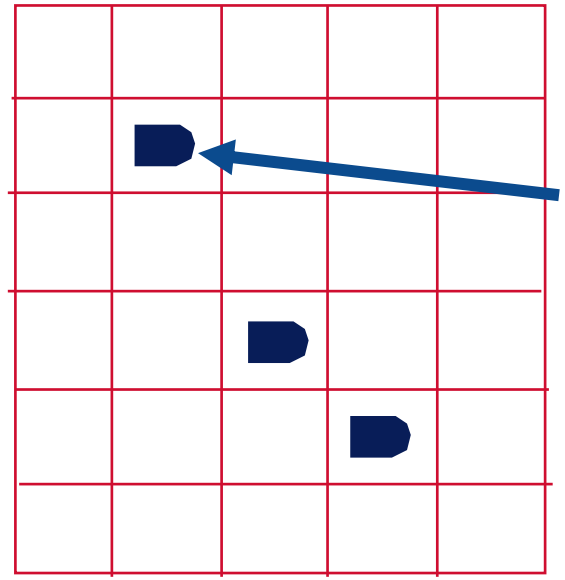
- **Even this simplified placement problem is NP-complete**
  - 1D placement
  - 2-pin nets only
  - All modules are of the same size
  - Wirelength as the only cost function
- **Realistic placement problems are**
  - 2D placement
  - Multi-pin nets
  - Modules of different sizes
  - Various cost function in addition to total wirelength
  - Various constraints for different target applications



bad placement good placement

# Let's Start with a Simple Placer

- **Very simple model of the chip**
  - A simple grid – like a chess board
  - Cells (gates) go in grid slots
  - Pins (connect off-chip, fixed at edges)




- **Very simple model of gates**
  - All gates are exactly the same size.
    - Unrealistic, I know, but simplifies things …
  - Each grid slot can hold 1 gate
    - Unrealistic, again, but simplifies things …

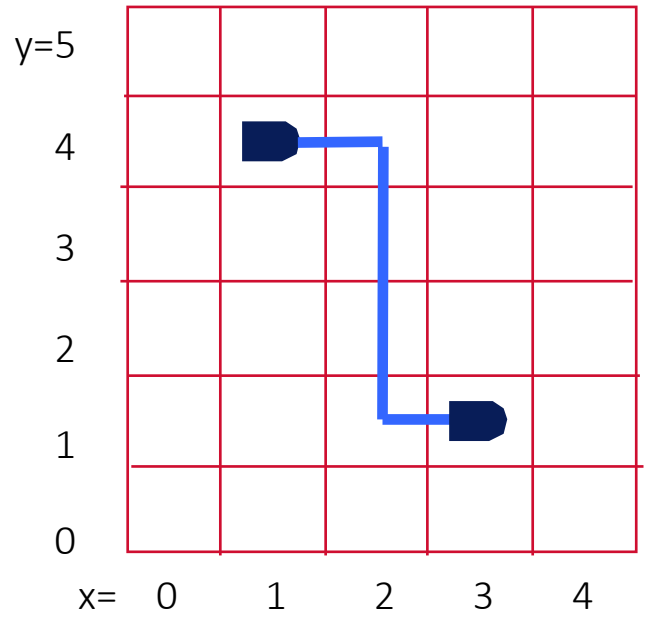
# Objective of a Placer

- **Placer optimizes the ability of router to connect all the nets**
  - But routers are computationally expensive tools. We can't run one "inside" placer
  - We need a simplified **approximation** **at this stage**
- **Every real placer minimizes expected wirelength**
  - For each wire in the design, **estimate the expected length** of the routed wire
  - Minimize this objective: $\sum_{\text{wires net}i}$ **EstimatedLength(net$_i$)**
- **Placer finds gate locations to minimize this objective**

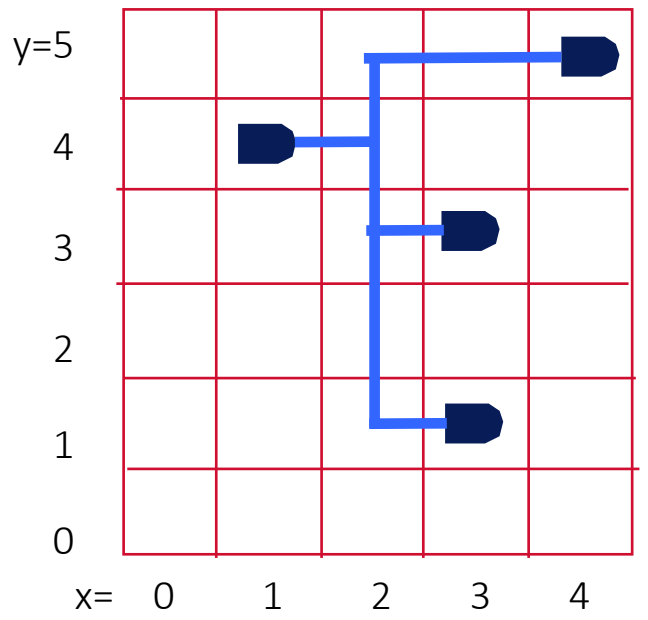
# Placement Terminology

- **Common term for a wire in a layout is called Net**
  - The whole set of gates and wires is called netlist
  - Net categorized by how many things it connects.

A "2-point net"

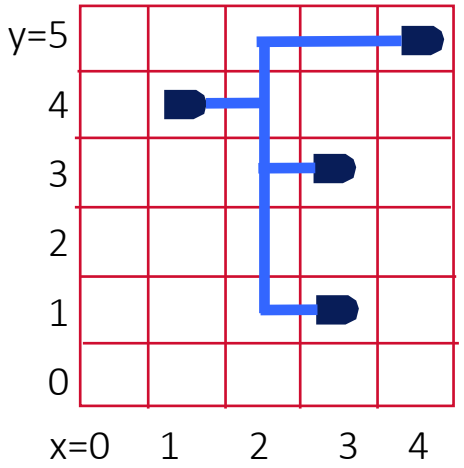


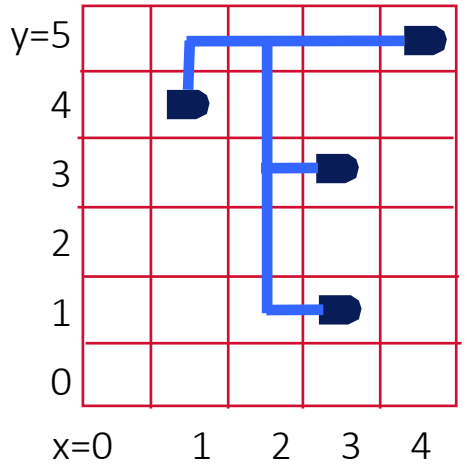

A "4-point net"

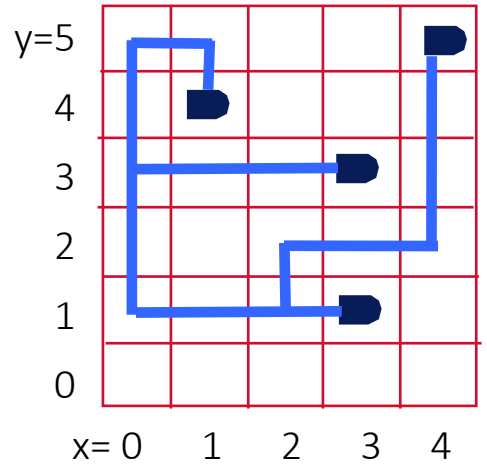


# Wirelength Estimation

- **Many different estimators for different placer methods**
  - Multi-point nets can be routed in many different paths
  - In a dense layout, nets do not all get routed in a “shortest path”

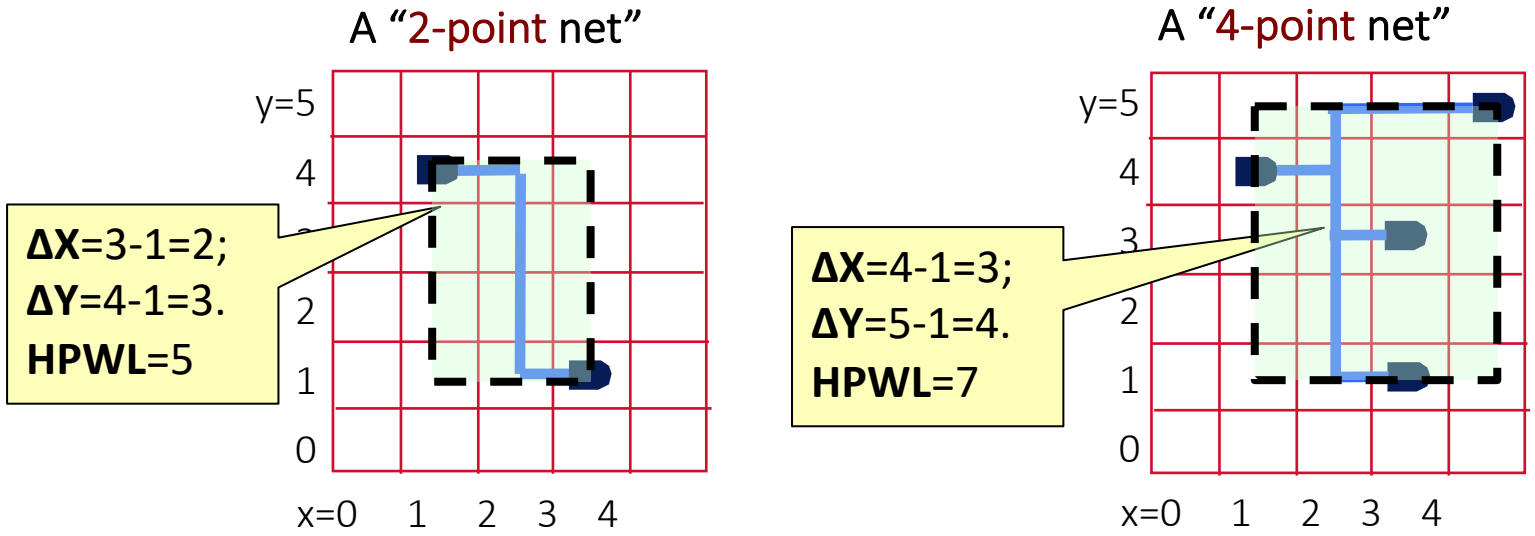


# Half-perimeter Wirelength (HPWL)

- **Most placers adopt Half-Perimeter Wirelength (HPWL)**
  - Also know as Bounding Box (BBOX) wirelength
  - Put smallest "bounding" box around all gates
  - Assume gate lives in "center" of the grid slot
  - Add width (ΔX) and height (ΔY) of the BBOX for the wirelength estimate



# About HPWL

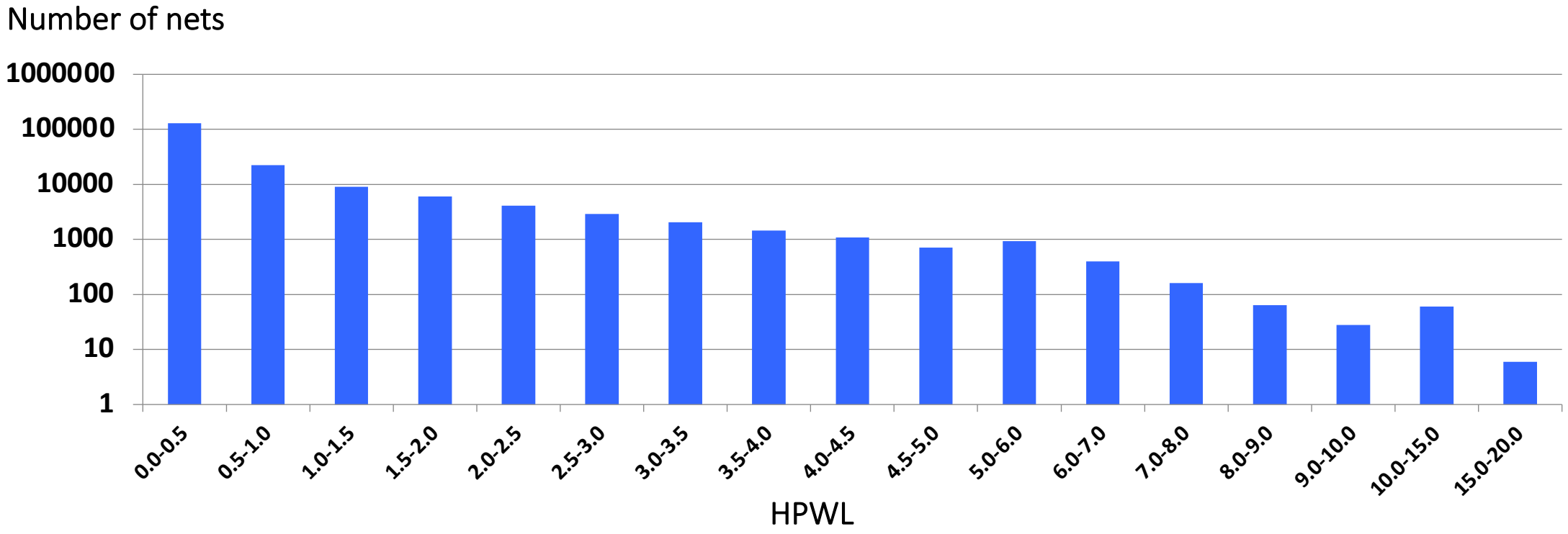
- **Easy to calculate, even for a multi-point net:**

[max{X coordinates of all gates) – min{X coordinates of all gates}]
+ [max{Y coordinates of all gates) – min{Y coordinates of all gates}]

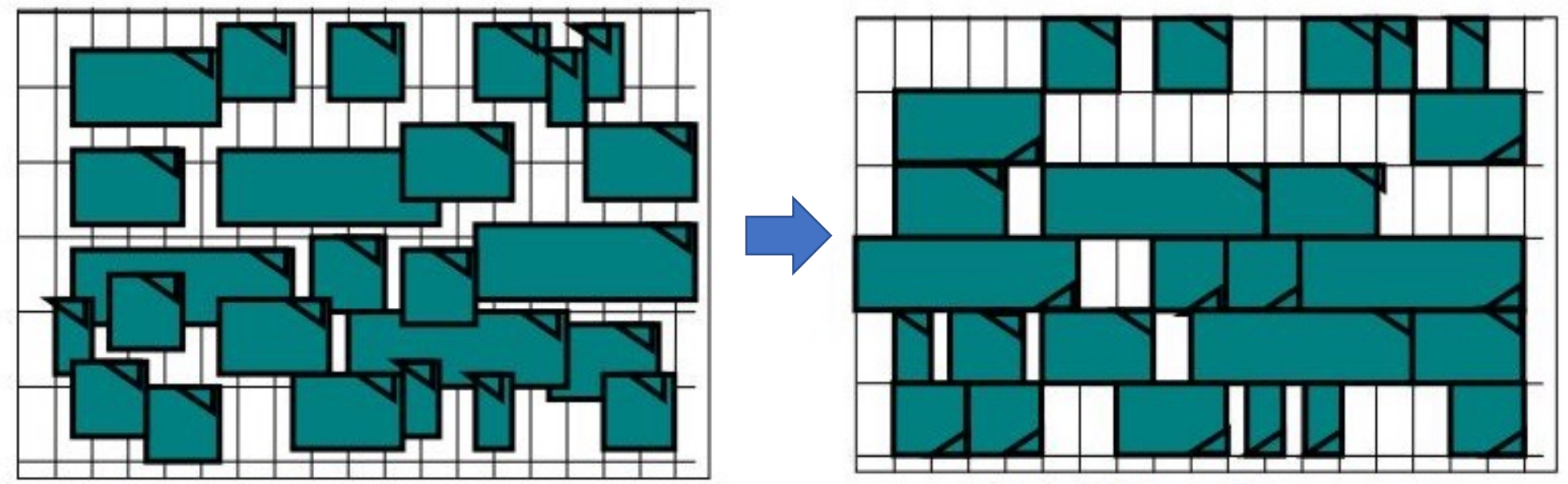
- **Always a lower bound on the real wire length**
  - No matter how complex the final routed wire path is…
  - …you need at least this much wire to connect everything
  - Aside: all wiring on big chips (and most boards) is strictly horizontal & vertical – no “arbitrary angles” for manufacturing reasons which is another reason HPWL is good

# About HPWL (cont'd)

- **Real distribution of HPWL for 200K gate IBM ASIC [DAC97]**
  - **181K nets**. Note LOG scale. **Most nets are short**. But there is a long tail to distribution.



# How would You Solve Placement?



# A Simple Iterative Placer

- **Start with a random placement**
  - Just randomly assign each gate to a grid location (1 gate per entry)
  - Yes – this is a terrible placement. Really big **L** = $\sum_{\text{nets Ni}}$ **HPWL(Ni)**
- **Random iterative improvement**
  - Pick **a random pair** of gates in the grid and swap their locations
  - Evaluate the change in
    - **L** = $\sum_{\text{nets Ni}}$ **HPWL(Ni);**
    - **ΔL= [new HPWL – old HPWL]**
  - If **ΔL<0,** new **L** got *smaller* → Improvement! ***Keep* this swap**
  - If **ΔL>0,** new **L** got *bigger* → Worse! ***Undo* this swap**
  - Keep doing this for many, many random swaps, until **L** stops improving

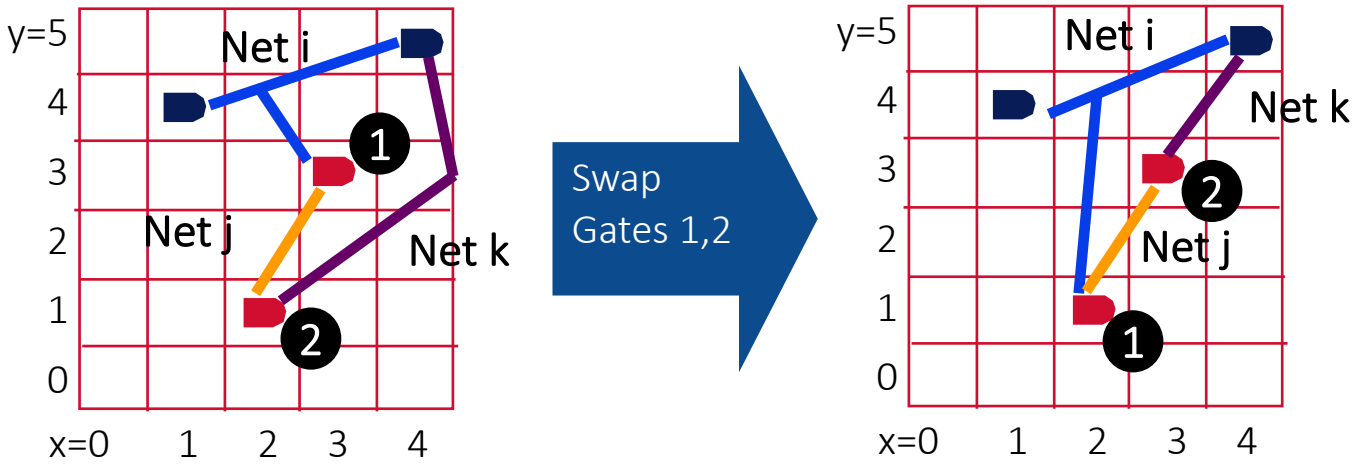
# A Simple Iterative Placer: Pseudocode

```
foreach( gate Gi in netlist )                          //random initial placement
   place Gi in random location (x,y) in grid not already occupied

L=0                                                    //calculate initial HPWL wirelength for whole netlist

foreach( net Ni in the netlist )
  L = L + HPWL(Ni);

while ( overall HPWL wirelength L is improving ) {     //main improvement loop

   pick random gate Gi;  pick random gate Gj;
   swap gates Gi and Gj
   evaluate ΔL = new HPWL  - old HPWL;
   if ( ΔL < 0 )  {                                    // improved placement! Update HPWL
      L = L + ΔL
   else                                                //  ΔL > 0, this is a worse placement
       undo swap of Gi and Gj
}
```

# Computing ΔL Efficiently (Incrementally)

- **Incremental wirelength update calculation**
  - Cannot afford to re-compute length of each net in entire placement after each swap! – most nets did not change
  - Compute ΔL incrementally – just look at nets that could change

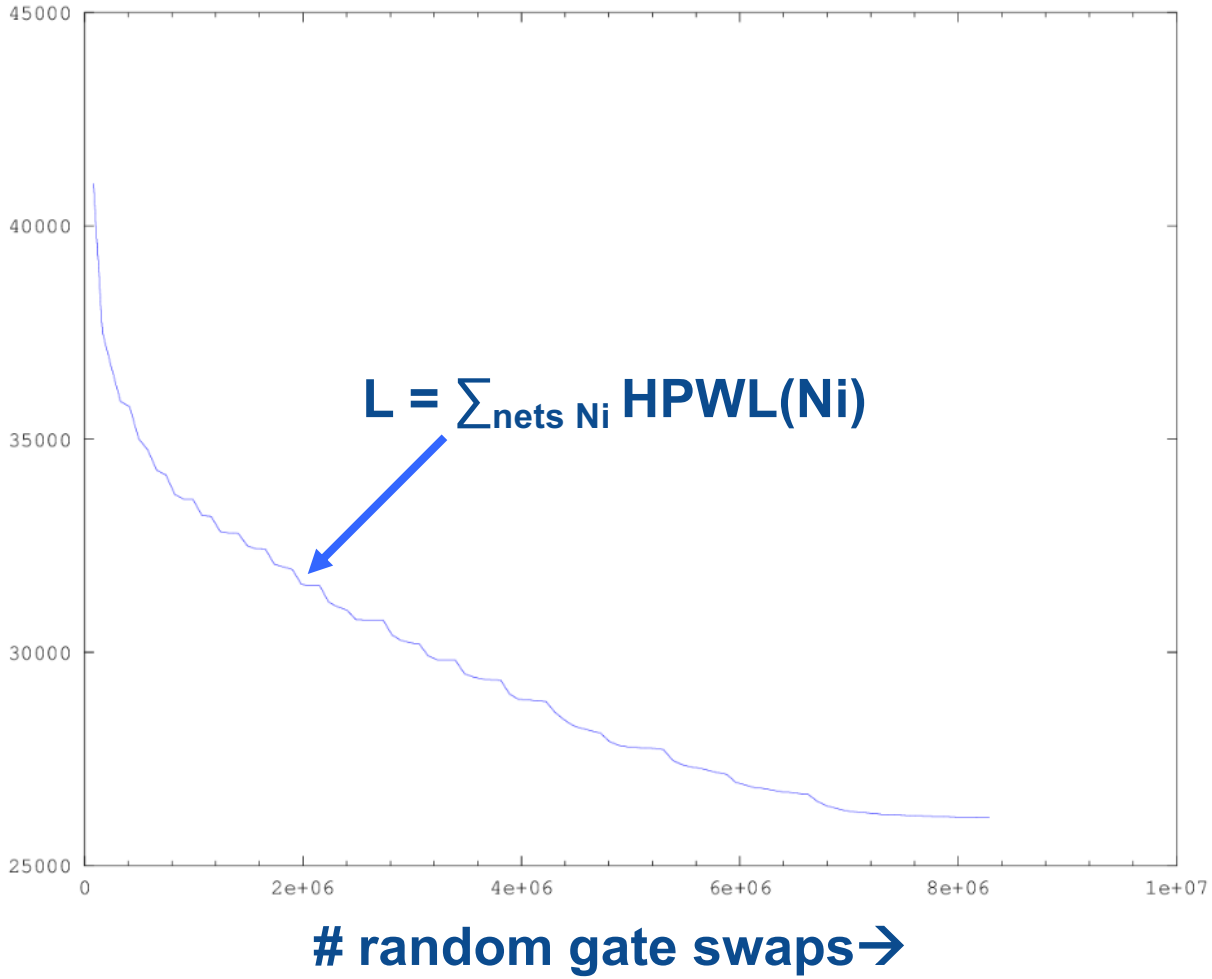
ΔL = [ HPWL(i) + HPWL(j) + HPWL(k) after swap] - [ HPWL(i) + HPWM(j) + HPWL(k) before swap]



# How does this Simple Placer Work?

- **It works ok …** 😂
  - Small benchmark
  - ~2500 gates + ~500 pins
  - Placed in a 50x50 grid

- **Graph shows progress**
  - Y axis: **L = $\sum_{\text{nets Ni}}$ HPWL(Ni)**
  - X axis: # swaps
  - Yes, this ran for **8M swaps** until progress on **L** stopped

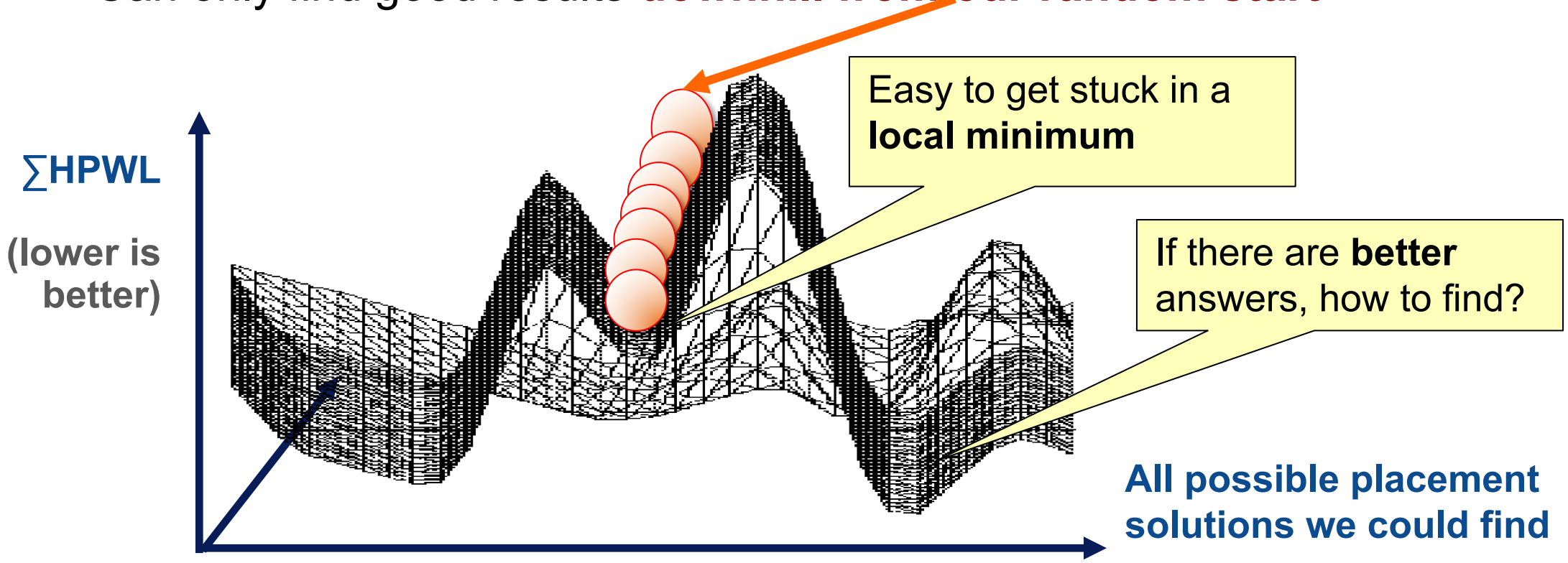


# Takeaways for the Simple Placer

- **Easy to understand and to implement**
  - Start with a random placement and randomly improve the wirelength until it stops getting better
  - Can optimize what we care about: estimated wirelength $\sum_{\textbf{nets Ni}}$ **HPWL(Ni)**
  - It will improve at each iteration always
- **However, the final result is still not very good**
  - We are being too greedy to go out of the local optimal
- **We can do better for sure**

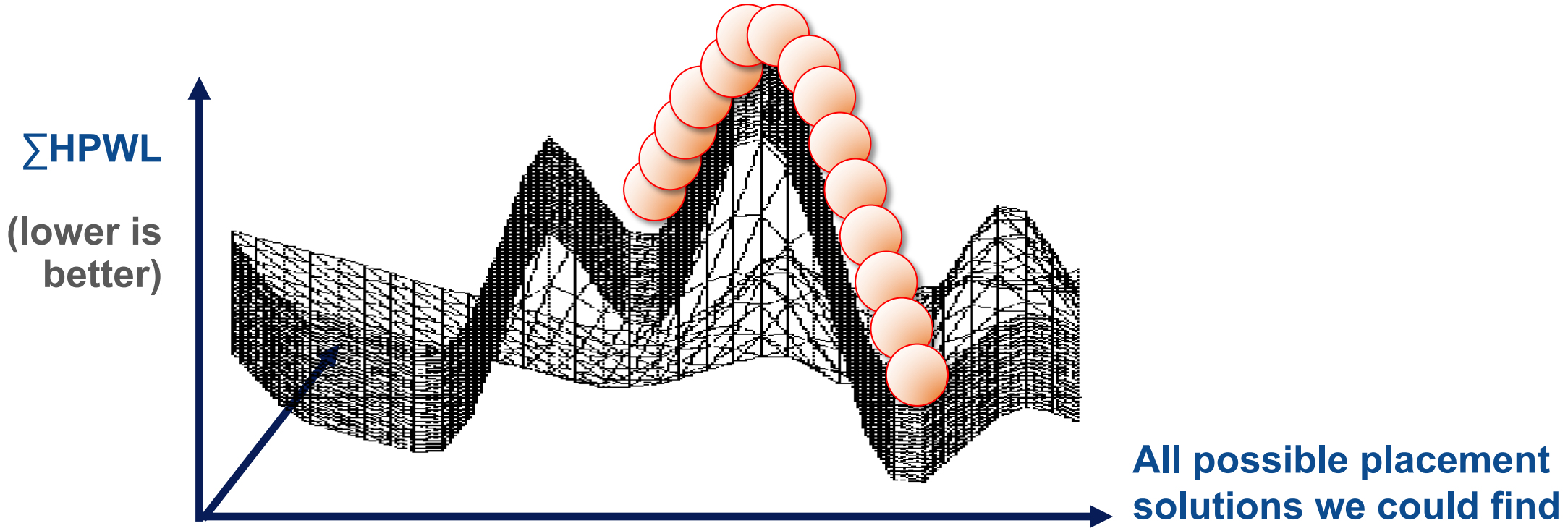
# Problem of the Simple Placer?

- **Imagine you could plot ∑HPWL for all possible placements**
  - Our random method only takes swaps that **improve wirelength**
  - Can only find good results **downhill from our random start**



# Downhill and Uphill Movement

- **If we could go uphill, we may get better placements**
  - How to control this?



# Simulated Annealing (SA)-based Placer

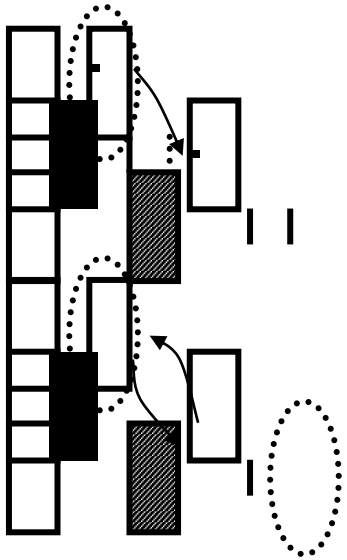
```
Start with a random initial placement with wirelength = ∑HPWL(net)
T = temperature = hot;    frozen = false;
while ( ! frozen ) {
      for (s=1; s< M* #gates in layout;  s++) { // M is how many swaps-per-gate
            Swap 2 random gates Gi and Gj
            Compute ΔL = [∑HPWL(net) before swap] - [∑HPWL(net) after swap]
            if (ΔL < 0 )
            then   accept this swap  //this is a new, better placement
            else   {
                  if( uniform_random() < exp( -ΔL/T ) )
                  then   accept this 'uphill' swap //this is a new, worse placement
                  else   undo this 'uphill' swap
      }
      if (∑HPWL(net) still decreasing over the last few temperatures)
      then T = 0.9 * T   // cool the temperature; do more gate swaps
      else  frozen = true
}
return (final placement as best solution)
```

Annealing temperature cooling "outer loop"

New:  Probabilistic swap accept

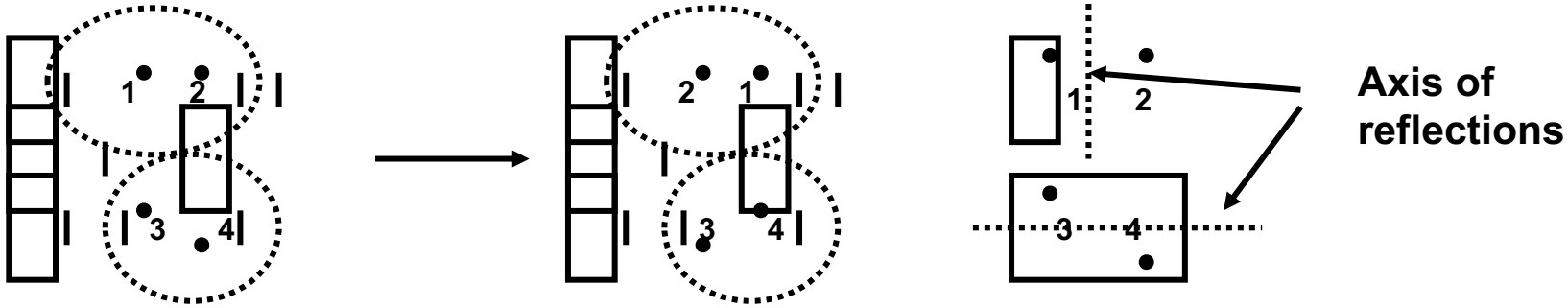
# Neighboring Solutions

**M1: Displace a module to a new location**



**M2: Swap two gates**

**M3: Change the orientation of a macro, if any**



# Move Selection Strategy

- **Select a move between M1 and M2**
  - Prob(M1)=4/5
  - Prob(M2)=1/5
- **If a move of type M1 is chosen and is rejected**
  - Take a move of type M3 (for the same gate) with probability 1/10
- **Restriction on**
  - How far a module can be displaced
  - What pairs of modules can be interchanged

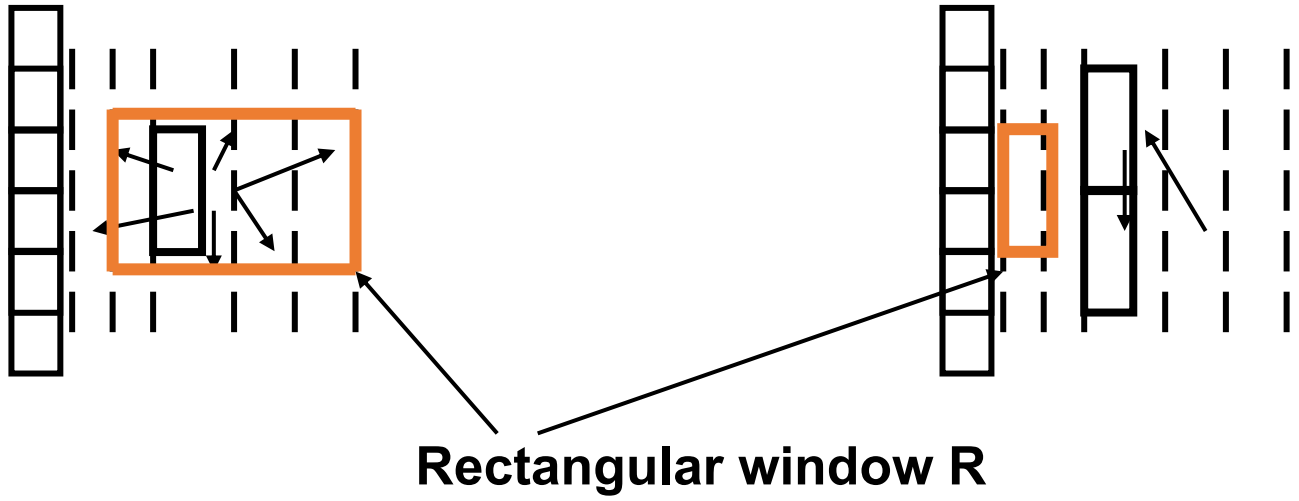
**M1: Displacement**

**M2: Interchange**

**M3: Reflection**

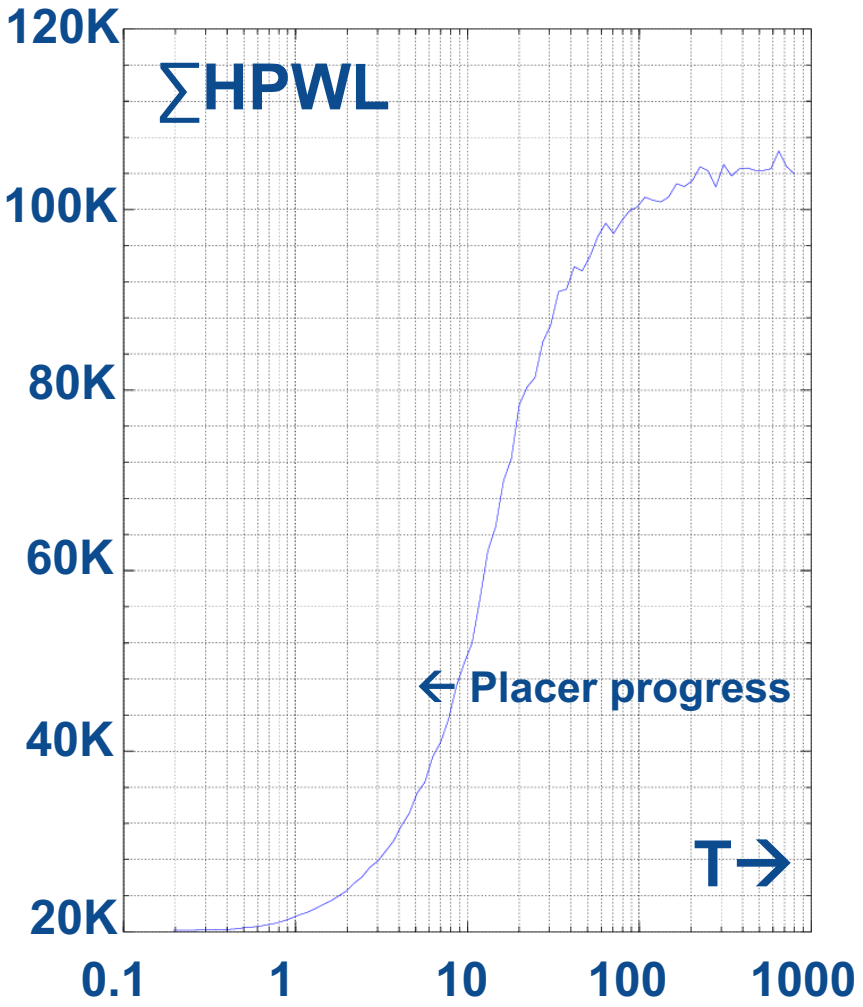
# Move Restriction

- **Range Limiter**
  - At the beginning, R is very large, big enough to contain the whole chip
  - Window size shrinks slowly as the temperature decreases. In fact, height and width of R $\propto$ log(T)
  - Switching move begins when window size are so small that no inter-row modules interchange is permitted



# How does an SA-based Placer Work?

- **It works better**
  - Same 2500 gate benchmark
  - **HOT**=800, **M**=100 moves/gate
  - About 30% less HPWL wirelength compared to the simple placer
- **Typical annealing "cost" curve**
  - **X** axis is **T**=temperature, LOG scale
    - Do 100x2500 swaps/temperature
    - Read right-to-left
  - **Y** axis is ∑**HPWL**, but LOG scale
  - Cool fact:  curves **always** look like this!



# Problem of SA-based Placer

- **Simulated annealing is a very successful method**
  - Much better than dumb random improvement
  - Dominant method for placers in 1980s, 1990s
  - And still today – used in lots of other VLSI CAD tasks (e.g., partitioning, floorplanning, etc.)
- **However, this is not not how we really do placers today**
  - Annealing works well for up to 100K – 500K gates
  - Annealing too inefficient for things with 1M+ gates
    - Today's designs can easily have 500M+ gates
- **We need another new set of ideas**

# A Famous SA-based Placer: TimberWolf



## The TimberWolf Placement and Routing Package

CARL SECHEN AND ALBERTO SANGIOVANNI-VINCENTELLI, FELLOW, IEEE

*Abstract*—**TimberWolf is an integrated set of placement and routing optimization programs. The general combinatorial optimization technique known as simulated annealing is used by each program. Programs for standard cell, macro/custom cell, and gate-array placement, as well as standard cell global routing have been developed. Experimental results on industrial circuits show that area savings over existing layout programs ranging from 15 to 62 percent are possible.**

### I. Introduction

TIMBERWOLF is an integrated set of placement and routing optimization programs. Extensions and modifications of the general combinatorial optimization technique known as simulated annealing [1] are used by each program. Four basic optimization programs of the TimberWolf package have been developed.

*1) A Standard-Cell Placement Program:* This program places standard cells into rows and/or columns in addition to allowing user-specified macro blocks and pads. The program was interfaced to the CIPAR standard cell placement package developed by American Microsystems, Inc. For the largest circuits tested (800 to 2700 cells), TimberWolf reduced total estimated wire lengths by 45 to 66 percent in comparison with CIPAR alone. Furthermore, final chip areas were reduced by 30 to 57 percent as a result of the improved placement. For a circuit of 1000 cells, TimberWolf reduced the final chip area by 31 percent in comparison to CIPAR and by 21 percent over another commercially available standard cell placement program in a benchmark performed at AMI.

*2) A Standard Cell Global Router Program:* The global router reduced by 10 to 15 percent the number of wiring tracks used by the CIPAR router. This translated to an overall area savings for 6 to 8 percent. Vecchi and Kirkpatrick [2] recently described the use of simulated annealing for global routing.

*3) A Macro/Custom Cell Placement Program:* This program places cells of any rectilinear shape. Furthermore, the cells may have fixed geometry including pin locations (macro cells) or they may have fixed area with a given aspect ratio range and with pins that need to be placed (custom cells). All rotations and reflections of each cell are considered. TimberWolf also has the ability to place cells among user-defined subregions of the chip. TimberWolf allows multiple chips to be placed simultaneously. This package can also be used to place circuits on one or more printed circuit boards.

The macro/custom cell placement program is currently under test on industrial circuits. However, the program has been tested on a Honeywell Information Systems Italy printed circuit board. The processor board required the placement of 613 variable-sized circuits. TimberWolf reduced the total wire length by 10 percent over the manually placed board.

*4) A Generalized Gate-Array Placement Program:* This program allows user-specified macros and primary terminals. This program found placement with a 6- to 27-percent reduction in total estimated wire length for several benchmark problems in comparison to the best published results. This program optionally includes in the cost calculation a measure of the local routing congestion.

This paper presents the algorithms used by each of the programs comprising the TimberWolf package and also presents the results that have been obtained. In particular, Section II describes the basic algorithm. In Section III, the standard cell placement optimization algorithm and program are described. Section IV presents details on the standard cell global router. In Section V, the macro/custom placement optimization algorithm and program are described and in Section VI the gate-array placement algorithm and implementation are presented. Finally, Section VII is devoted to concluding remarks and future research.

### II. The Basic Algorithm

Simulated annealing has been proposed by Kirkpatrick *et al.* [1] as an effective method for the determination of global minima of combinatorial optimization problems involving many degrees of freedom. Its basic feature is the possibility of exploring the configuration space of the optimization problem allowing *hill climbing* moves, i.e., the acceptance of new configurations of the problem which increase the cost. These moves are controlled by a parameter, in analogy with temperature in the annealing process, and are less and less likely towards the end of the process.

Manuscript received August 31, 1984; revised December 18, 1984. The algorithmic part of this research has been supported by DARPA under Grant N00039-83-C-0107. The TimberWolf placement and routing package has been supported by a Grant from the MICRO of the State of California.

The authors are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

0018-9200/85/0400-0510$01.00 ©1985 IEEE


http://www.twolf.com/

*"The Timberwolf Placement and Routing Package", Sechen, Sangiovanni; IEEE Journal of Solid-State Circuits, vol SC-20, No. 2(1985) 510-522*

# Summary

- **We have discussed basic ideas of placement**
- **We have discussed importance and difficulty of placement**
- **We have discussed the problem formulation of placement**
  - Terminology
  - Input, output, objective
  - Wirelength estimation
- **We have discussed two simple placement solutions**
  - Random iterative improvement-based placement algorithm
  - Simulated annealing-based placement