VLSI CAD: Logic to Layout

Lecture 9.1

ASIC Placement: Basics

Rob A. Rutenbar
University of Illinois

From Logic → Layout!

- **New topics!**

coursera Courses Universities About Rob A. Rutenbar

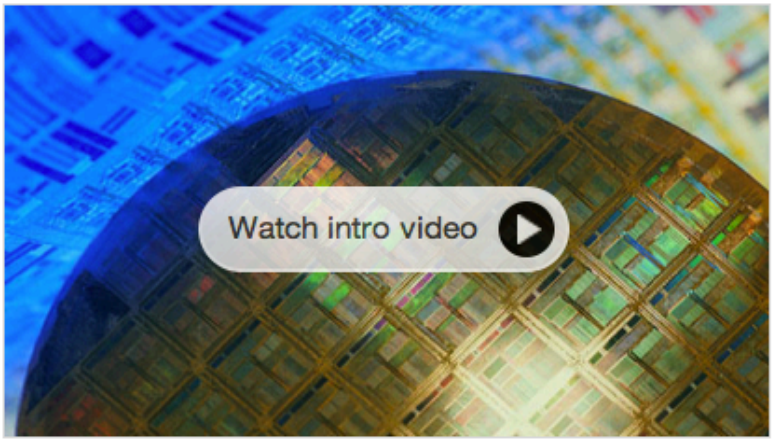
Edit Course Description Edit Session Descriptions Edit Session Materials

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

VLSI CAD: Logic to Layout

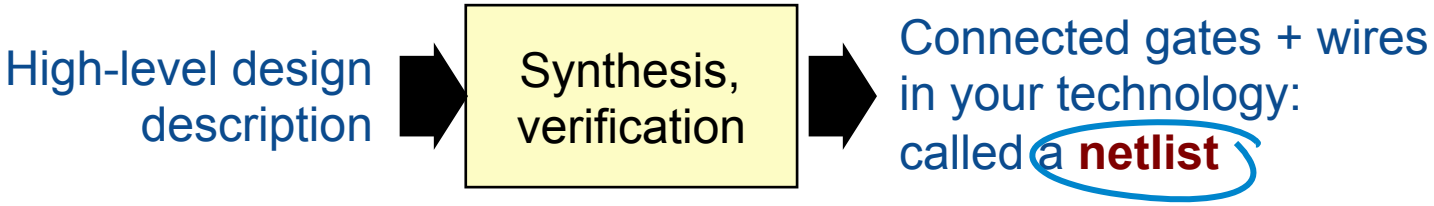
Rob A. Rutenbar

A modern VLSI chip has a zillion parts -- logic, control, memory, interconnect, etc. How do we design these complex chips? Answer: CAD software tools. Learn how to build these tools in this class.



About Layout…

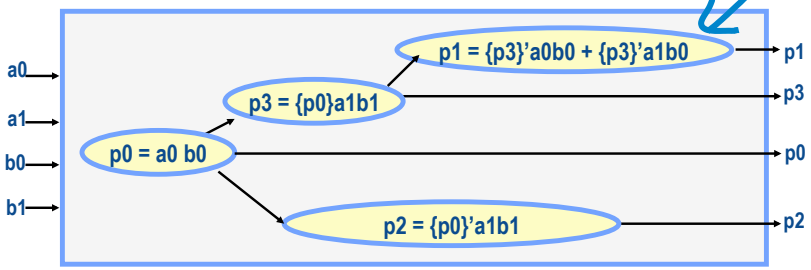
- **What you know...**
  - Computational Boolean algebra, representation, some verification, some synthesis
  - This is what happens in the **"front end"** of the ASIC design process



- **What you don't know...**
  - The **"back end"** problem -- turning these into IC masks to build real chips
  - Called "**layout**" or "**physical design**"
  - Where we start: the **placement problem**

Aside: Missing One Key Step

- **Fact: what comes out of multi-level synthesis is NOT gates**
  - It's a Boolean network model: a graph of nodes, each a 2-level SOP form
  - It is **NOT** the logic gates we use in real chip layout



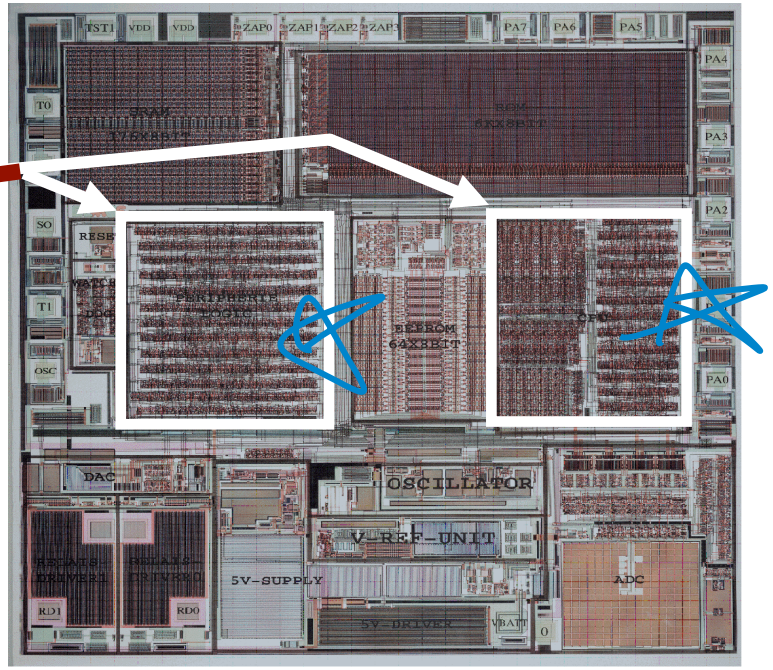
- **Missing a key step, called Technology Mapping**
  - This transforms the abstract form of the above network into real logic gates
  - Cover this next week, since we need placer info for next Programming Assignment

Placement for ASICs

- **Focusing on the most common tasks in layout**
  - **Row-based standard cell layouts** for ASICs and SOCs

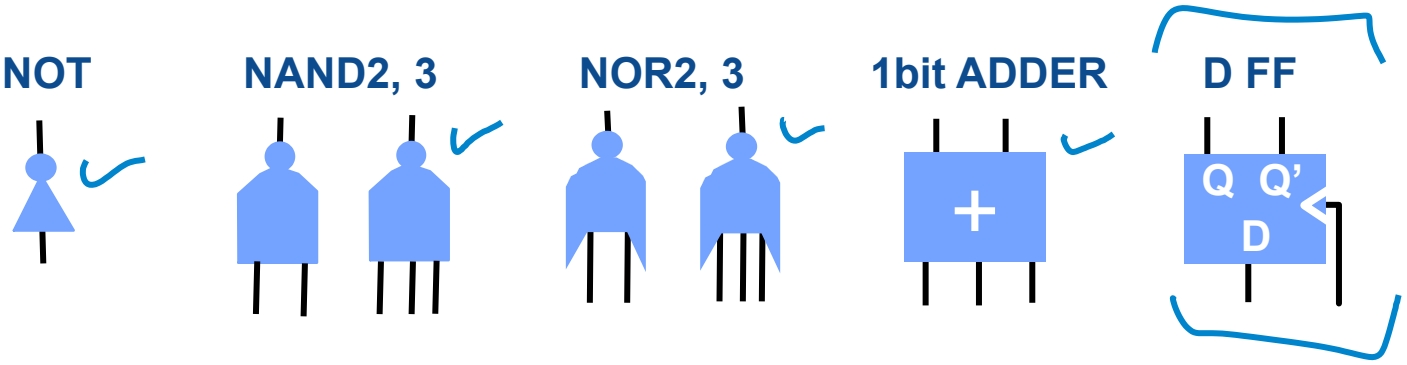
Synthesis & mapping give a **netlist** of gates + wires

Our job is to **place** them optimally in rows in regions of the chip, and to **route the wires** to connect everything



Start With: A Standard Cell Library

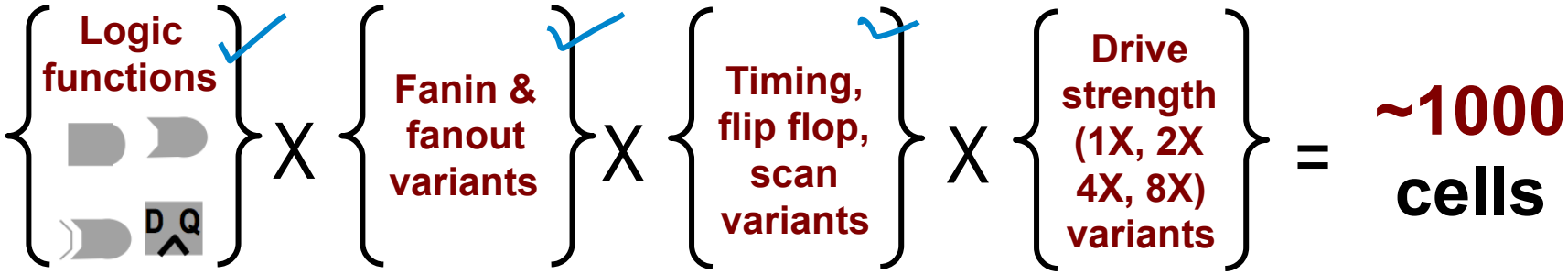
- **Confusing name: this is a library of things called "Standard cells"**
  - A **standard cell** is a basic logic gate or small logic element, e.g., a flip flop (FF)
  - This is the set of **allowed logic elements** to build your chip. Tiny example:



- **Why?**
  - Lots of complicated **electrical** stuff going on at the transistor level
  - Each cell **hides** these electrical details, presents a simple, geometric abstraction

How Big is a Library--How Many Cells?

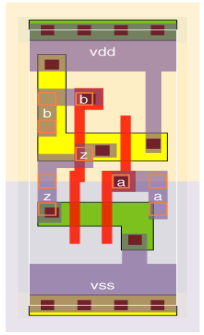
- **Often, very big**
  - For all logic functions, input/output variants, timing variants, electrical drive strengths…



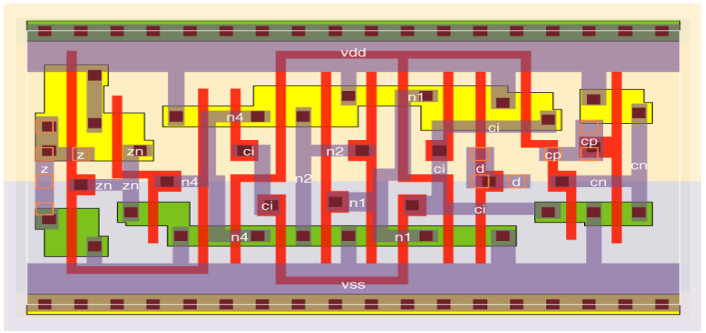
- **How to *think* about a standard cell**
  - Simple abstraction of a geometric "container" for the circuits you need to make logic.
  - *Inside* the cell: Complex device & mask & electrical issues
  - *Outside* the cell: **A box with pins**

Some Real Standard Cells

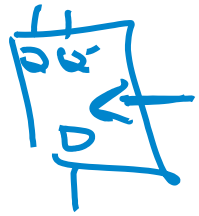
- **In older technology (130nm CMOS)**
  - Geometric fact #1: All have **same height** (so we can arrange them in rows)
  - Geometric fact #2: Cells can have **different widths**, depending on circuit complexity



NAND2



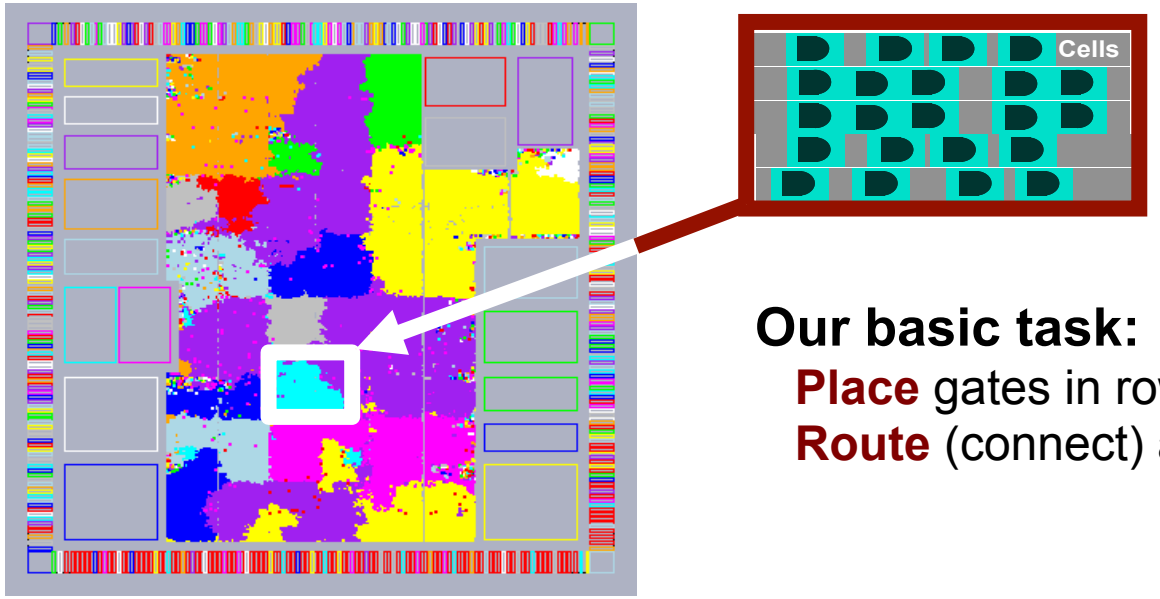
Edge-triggered D Flip Flop



From: **Graham Petley's** www.vlsitechnology.org site, which has open source cells
http://www.vlsitechnology.org/html/cells/wsclib013/lib_gif_index.html

Realistic Context: Place One Block on SOC

- **Big SOCs often designed *hierarchically*, composed from blocks which represent parts of overall system**



Courtesy Larry Pileggi, CMU

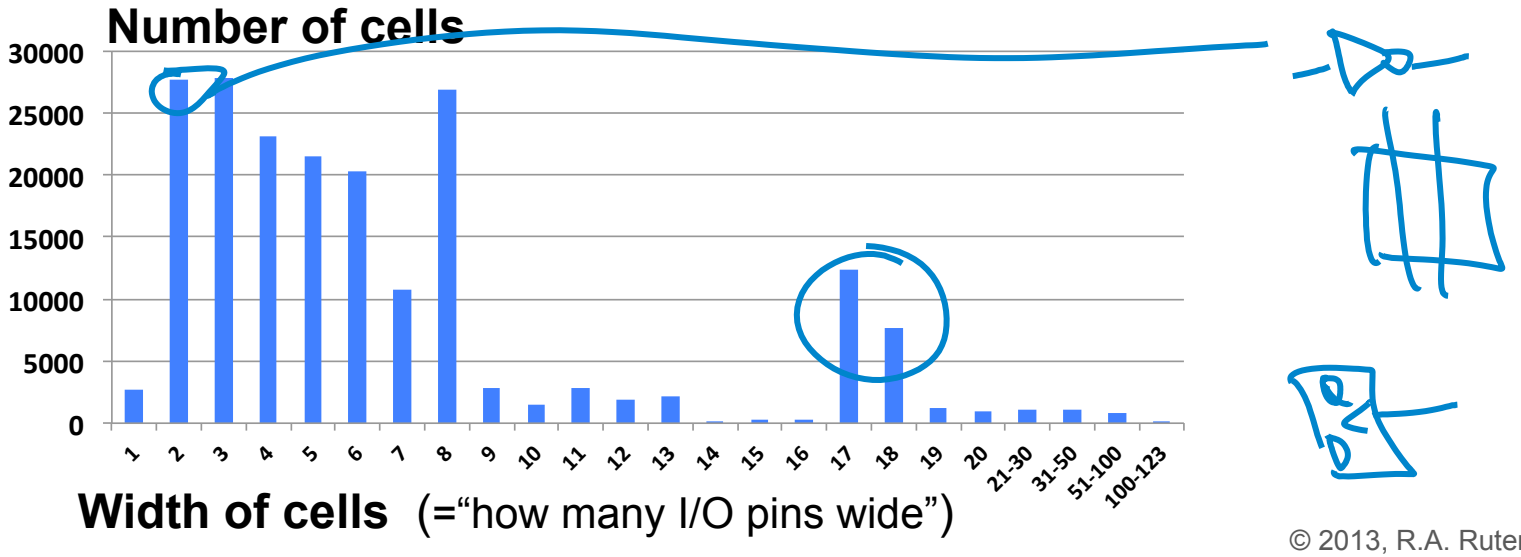
Our basic task:

Place gates in rows

Route (connect) all gates with wires

About Standard Cells: Size Distribution

- **In real designs, small cells dominate, but lots of wide cells too**
  - Old example: **206K gate** IBM ASIC
    - [J. Vygen, "Algorithms for Detailed Placement of Standard Cells", Proc. IEEE Design Automation & Test Europe Conference (DATE), 1998].
  - This would be one small block (among many) on a big SOC today (eg, 20-100M gates)



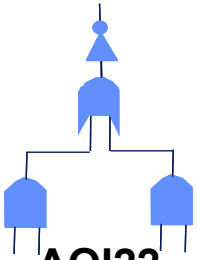
One More Aside: About Chip "Size"

- **Strange question: How big is a "100 million gate ASIC"**
  - Surprisingly, it is almost certainly **NOT** 100,000,000 logic gates
  - Numbers are usually "equivalent small gates" – transform all logic into **2-input NANDs**

Size? 1 gate

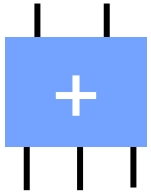
NAND2

Size? ~4 gates



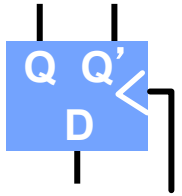
AOI22

Size? ~6 gates



1bit adder

Size? ~10 gates



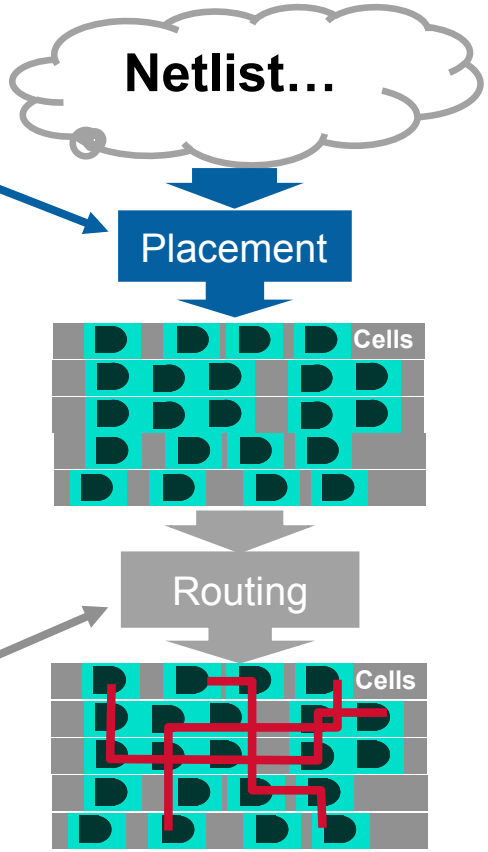
D Flip Flop

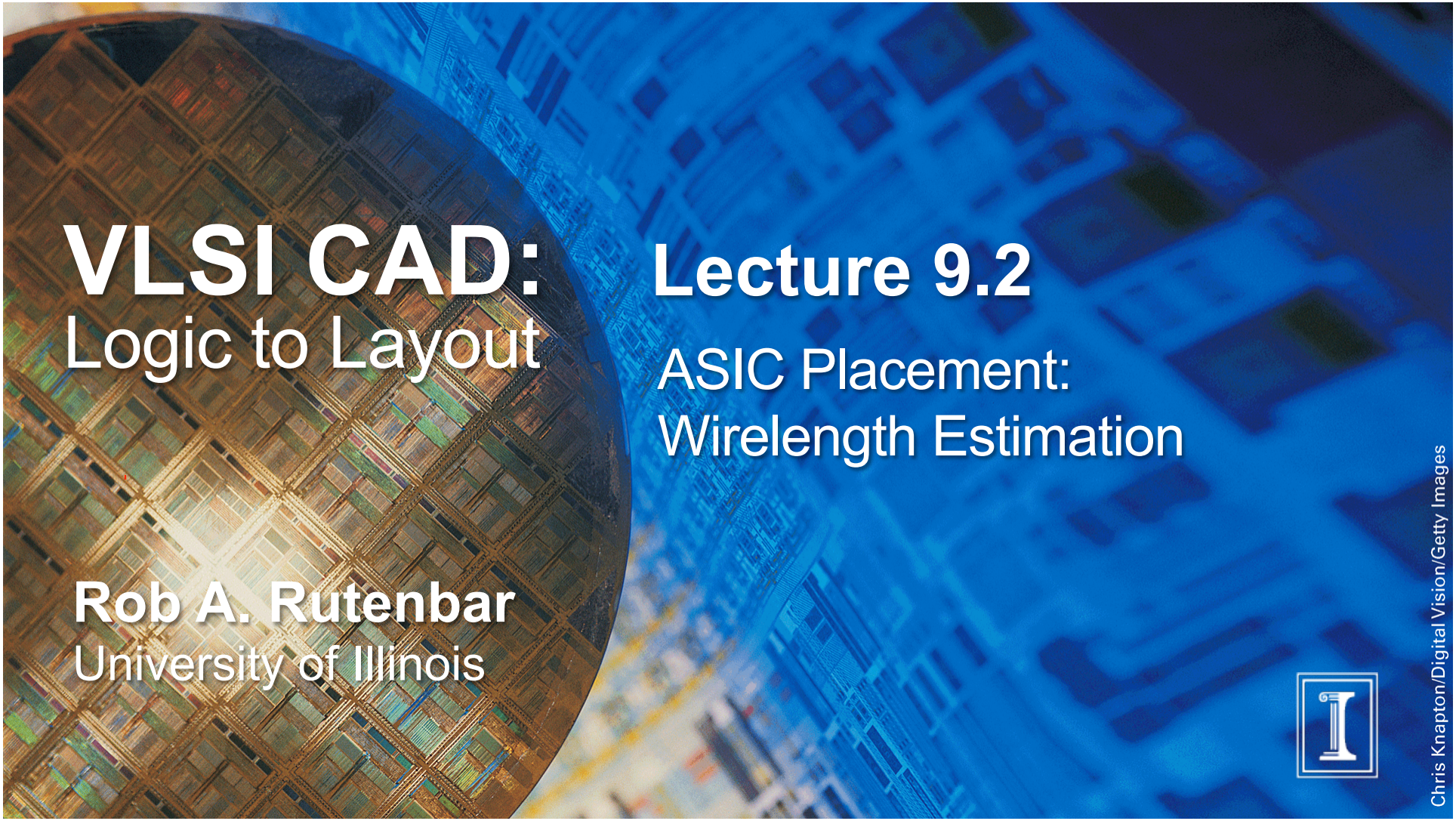
- **Consequence: 2 measures people use for "size" here**
  - **Gates**: This is "equivalent little NAND gates". Usually a big number
  - **Instances**: # things *really* placed. **~Rule: Instances = Gates ÷ 4 or 5**

First Problem: ASIC Placement

- **What does a placer do?**
  - **Input:** Netlist of gates & wires
  - **Output:** Exact location of each gate
  - **Goal:** Able to **route** (connect) all wires

- **Is this hard?** **Yes!**
  - 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





VLSI CAD: Logic to Layout

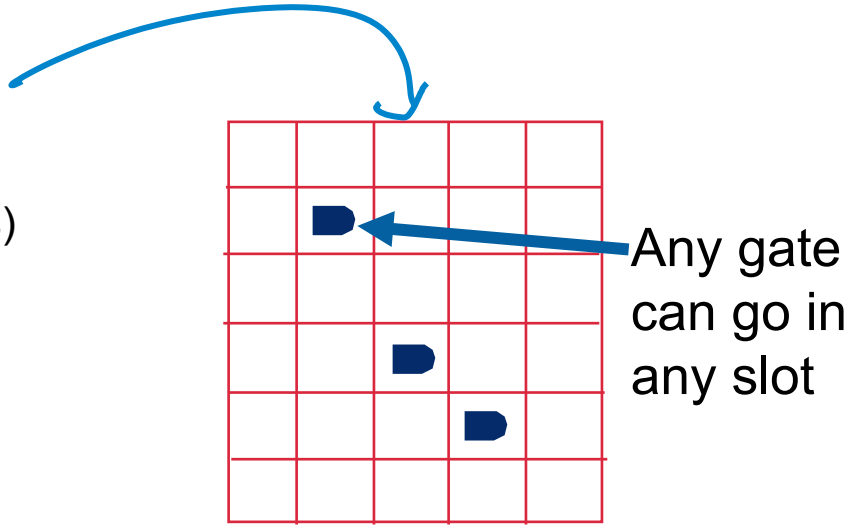
Lecture 9.2

ASIC Placement: Wirelength Estimation

Rob A. Rutenbar
University of Illinois

Let's Build a Very *Simple* Placer, To Start

- **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, but simplifies things)
  - Each grid slot can hold 1 gate

What Does a Placer Do…?

- **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 simplifying **approximation**

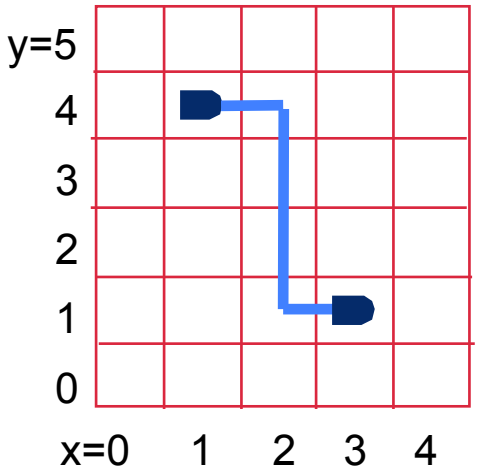
- **What every real placer does: Minimizes expected wirelength**
  - For each wire in the design, **estimate the expected length** of the routed wire
  - Minimize this objective: $\sum_{\text{wires } W_i}$ **EstimatedLength(W_i)**

- **Placer "solves" for gate locations to minimize this objective**

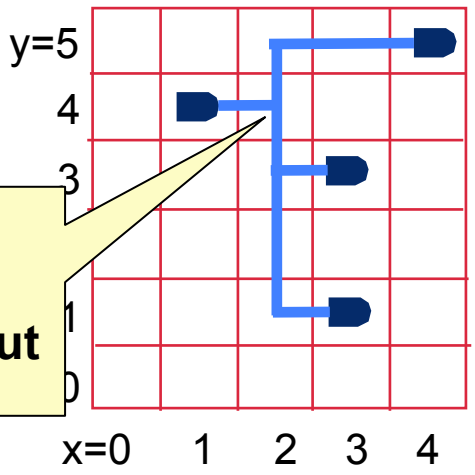
First, Some Terminology: Nets

- **Common term for a wire in a layout is: Net**
  - So, we call them "**nets**." And the whole set of gates+wires is "**the netlist**"
  - Net categorized by how many things it connects. Call these "**points**" for short

A "2-point net"



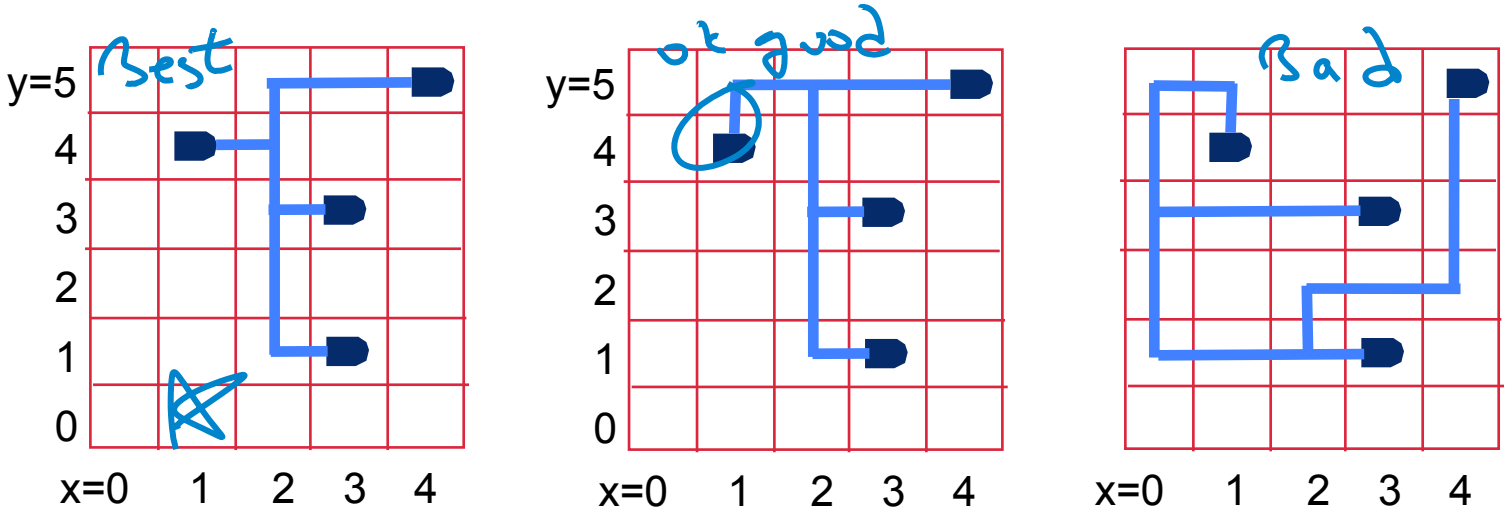
A "4-point net"



Why >2 points? Logic **fanout**

About Wirelength Estimation

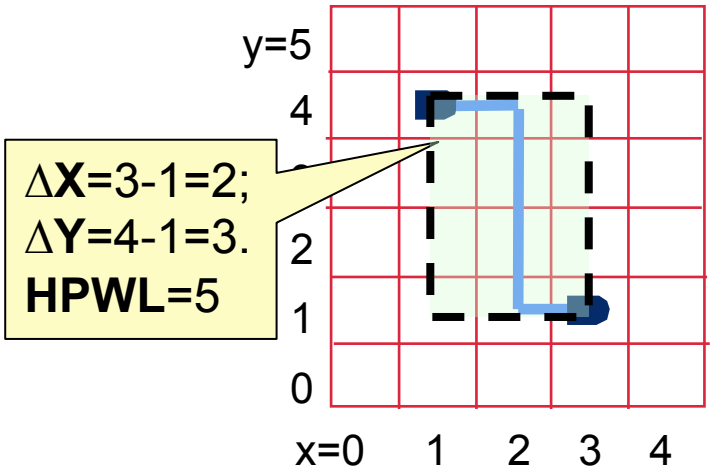
- **Many different estimators, adapted to different placer methods**
- **Why is this hard?**
  - Multi-point nets can be routed in many **different paths**.
  - In a dense layout, nets do **not** all get routed in a “shortest path.” Hard to predict.



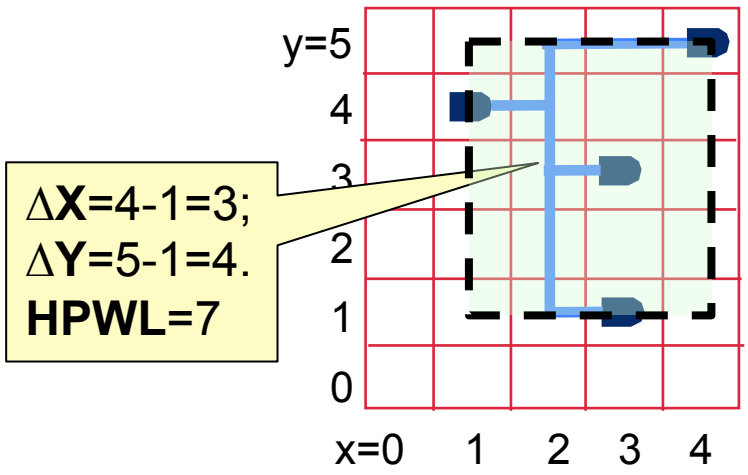
Most Famous Estimator: Half-Perimeter

- **Called Half-Perimeter Wirelength (HPWL), also Bounding Box (BBOX)**
  - 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. That's the wirelength estimate.

A "2-point net"



A "4-point net"

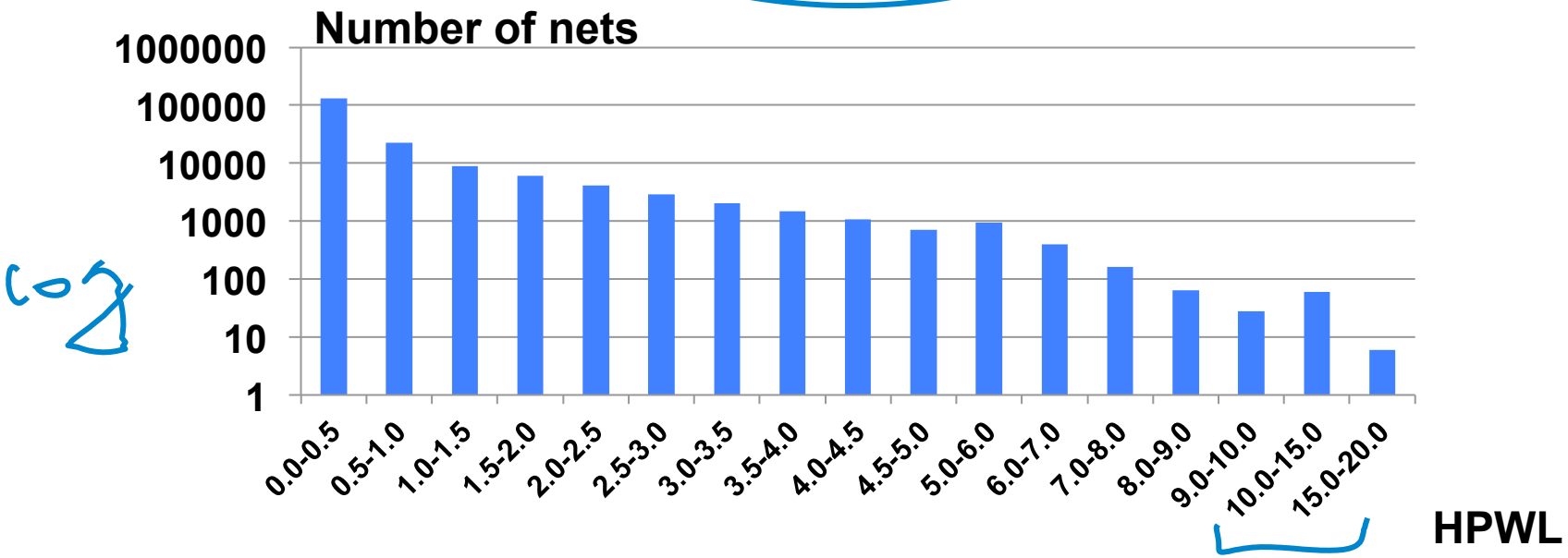


About HPWL Estimator

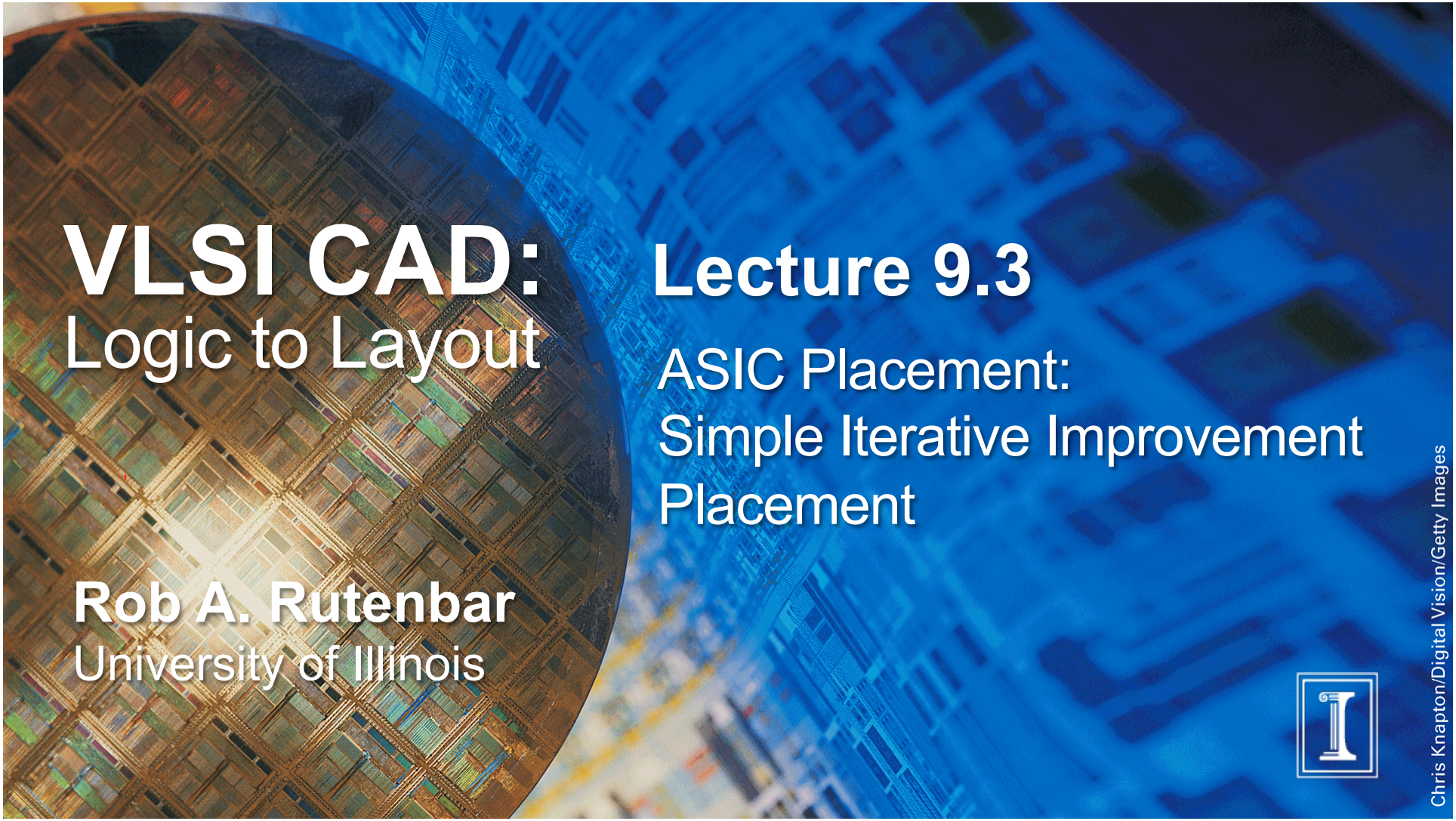
- **Easy to calculate, even for a multi-point net**
  - General formula:
  - **HPWL = [max{X coordinates of all gates) – min{X coordinates of all gates}] + [max{Y coordinates of all gates) – min{Y coordinates of all gates}]**
- **Always a lower bound on the real wire length**
  - No matter how complex the final routed wire path is…
  - …you need **at least this much wire** to connect everything.
  - Aside: *all* wiring on big chips (and *most* wire on boards) is **strictly horizontal & vertical** – no "arbitrary angles" for manufacturing reasons.
    - Another reason HPWL is a good estimator.

HPWL Wirelength Estimation

- **Real distribution of HPWL for 165K gate IBM ASIC**
  - **181K nets**. Note LOG scale. **Most nets are short**. But there is a long tail to distribution.



[J. Vygen, "Algorithms for Large-Scale Flat Placement," Proc. ACM/IEEE Design Automation Conference, 1997].



VLSI CAD: Logic to Layout

Lecture 9.3

ASIC Placement: Simple Iterative Improvement Placement

Rob A. Rutenbar
University of Illinois

A Very Simple Placer

- **2 big ideas**
- **Start with a random placement**
  - Just **randomly assign** each gate to a grid location (only 1 gate per grid, please!)
  - 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. **Exchange** their locations (called a "**swap**")
  - Evaluate the **change** in **L = $\sum_{\text{nets Ni}}$ HPWL(Ni);** compute **ΔL= [new HPWL – old HPWL]**
  - If **ΔL<0,** new **L** got *smaller*: Good! Improvement! ***Keep* this swap**
  - If **ΔL>0,** new **L** got *bigger*: Bad! Worse! ***Undo* this swap**
  - Keep doing this for many, many random swaps, until **L** stops improving

Random Iterative Improvement Place

- **Pseudo-code**

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

//calculate initial HPWL wirelength for whole netlist
L=0
foreach( net Ni in the netlist )
  L = L + HPWL(Ni);

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

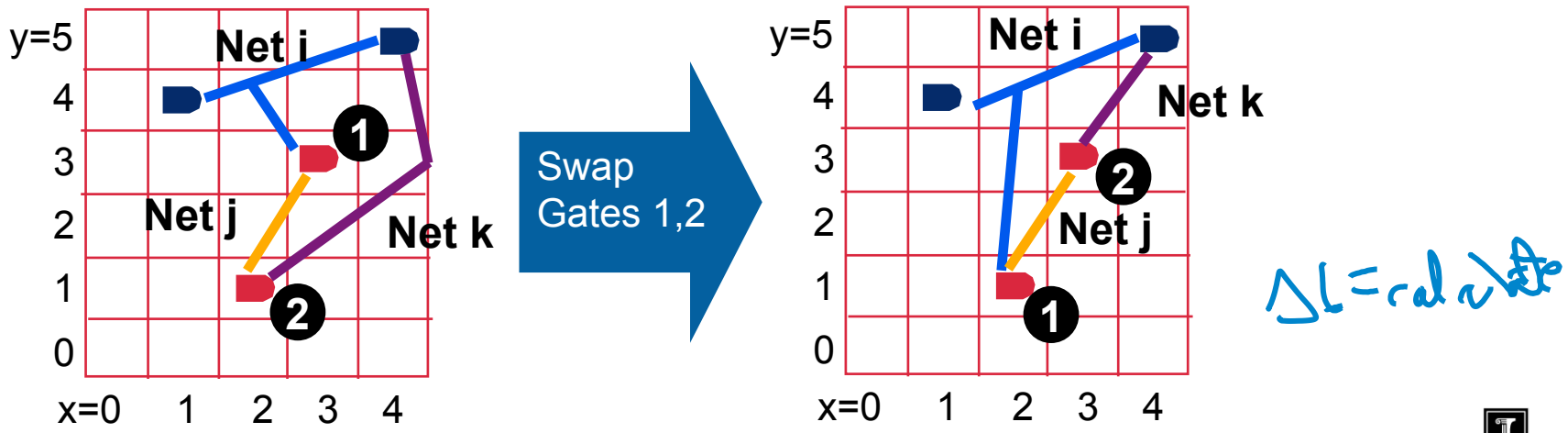
keep!

undo swap!

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! Do this **incrementally**--just look at nets that **could** change

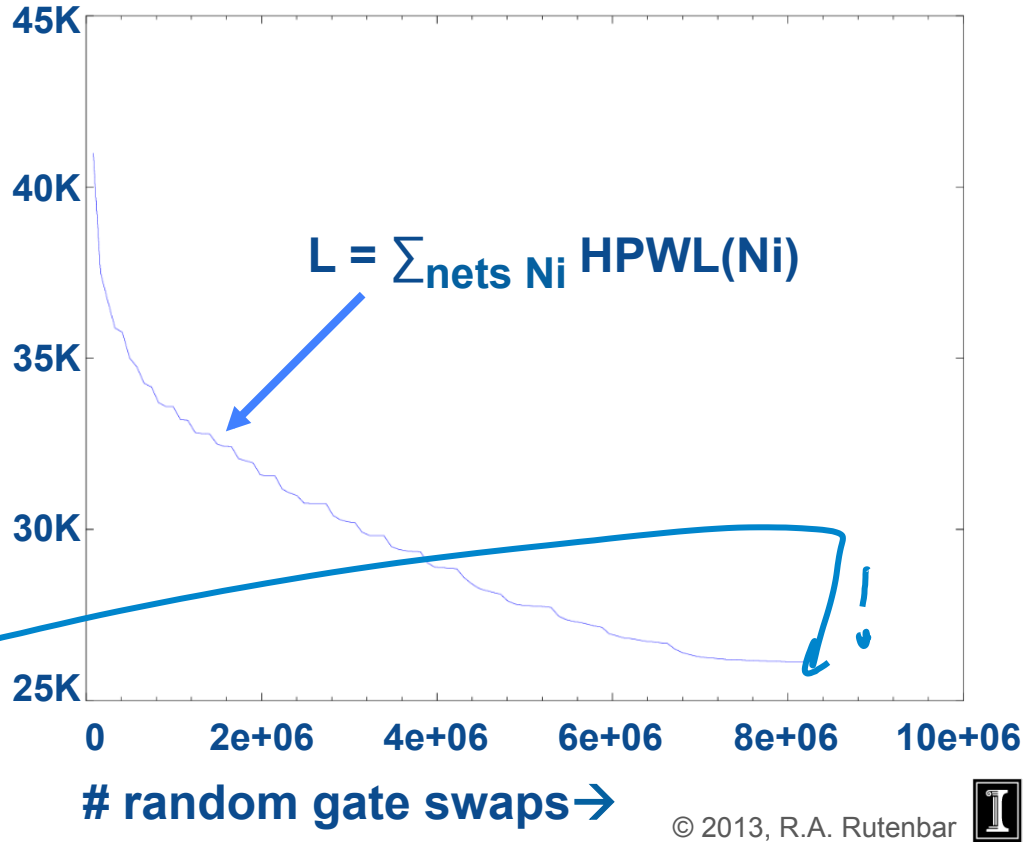
ΔL = [HPWL(i) + HPWL(j) + HPWL(k) after swap] - [HPWL(i) + HPWM(j) + HPWL(k) before swap]



How Does This 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



Random Iterative Improvement

- **Easy to understand and to implement**
  - Start with a random placement. Randomly improve until it stops getting better
  - Can optimize what we care about: estimated wirelength $\sum_{\text{nets Ni}}$ **HPWL(Ni)**
  - It will improve. Sometimes, a lot.

- **But… final result is still not very good**

- **We can do better. A lot (really, a *lot*) better…**

VLSI CAD: Logic to Layout

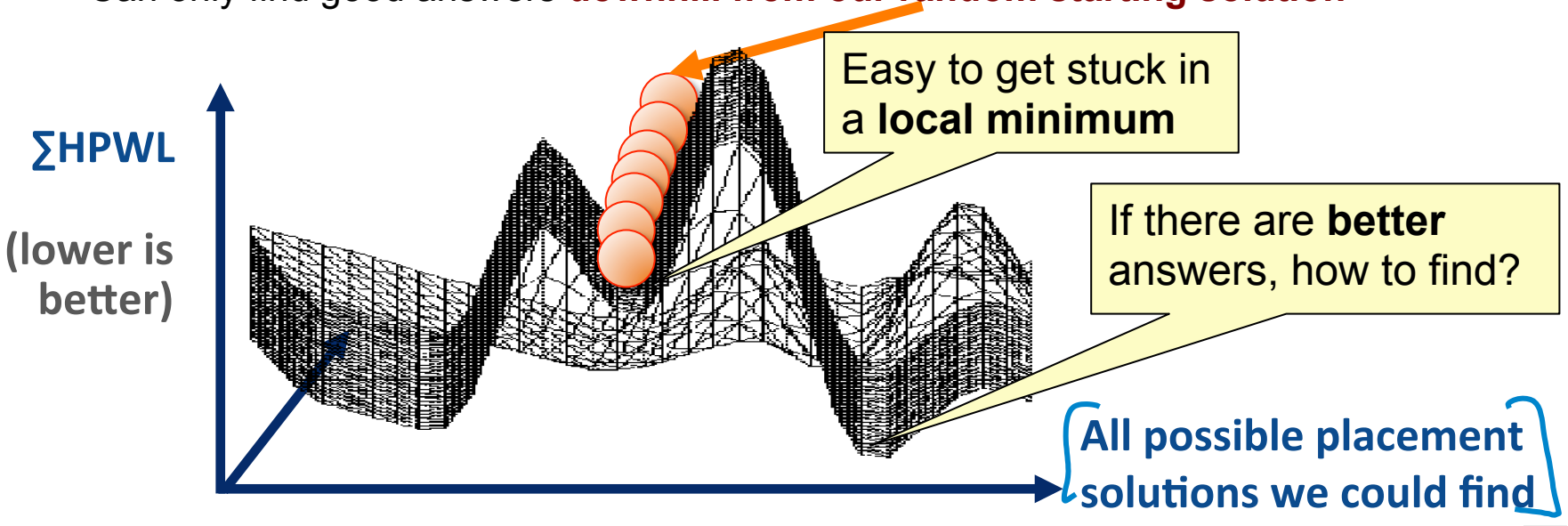
Lecture 9.4

ASIC Placement: Iterative Improvement With Hill-Climbing

Rob A. Rutenbar
University of Illinois

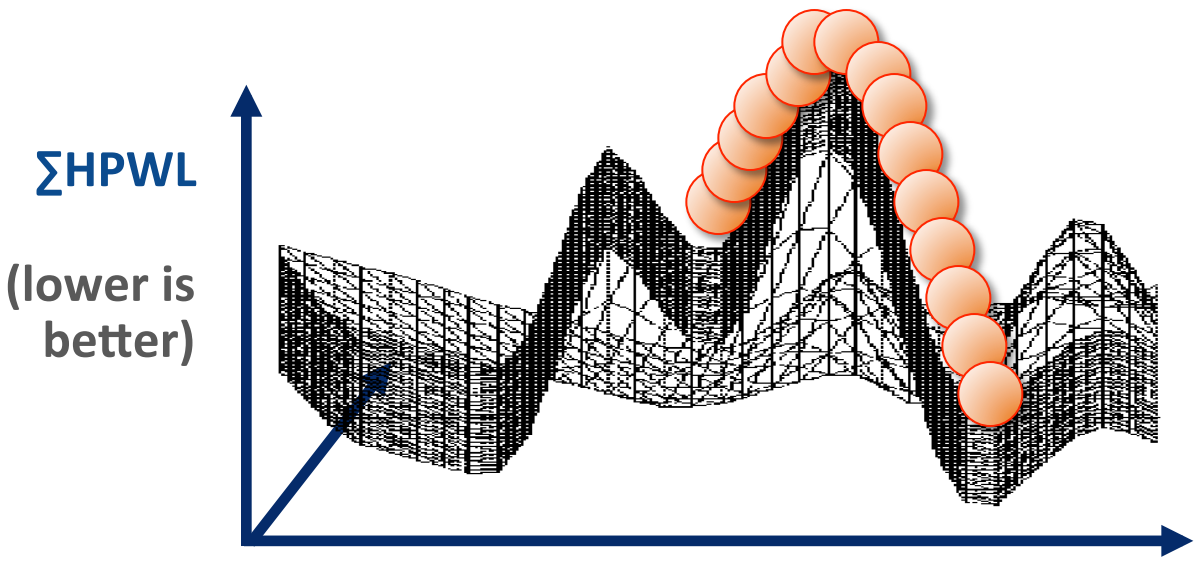
Iterative Improvement Placer: Problem?

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

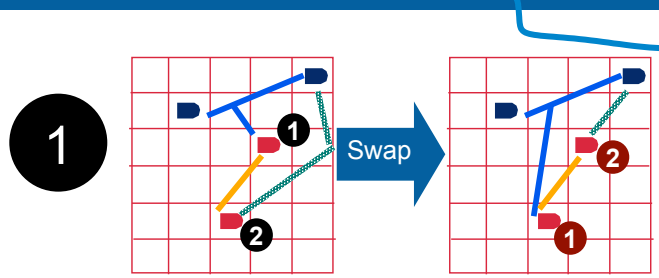


Getting Stuck: Downhill vs Uphill Methods

- **If we could go uphill, we could maybe get better placements**
  - **New problem:** how to **control** this? Such changes make placement **worse**!



You Know: *Greedy* Random Improvement



2 Evaluate **ΔL = Change in ∑HPWL**

3 If (**ΔL** = Length got smaller?)

YES

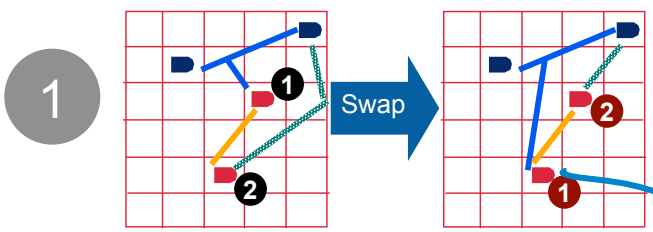
NO

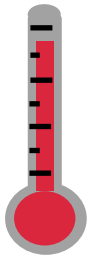
4 **Keep swap**

5 **UNDO swap**

Need another BIG idea here

New Idea: Improvement with *Hill-Climbing*





New:
Hill-climbing control parameter,
T=temperature

2 Evaluate **ΔL = Change in ∑HPWL**

3 If (**ΔL** = Length got smaller?)

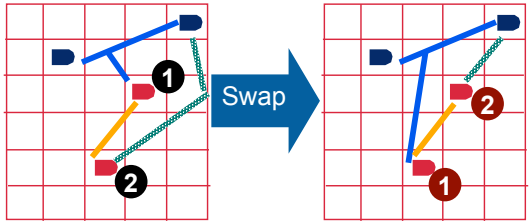
YES

NO

4 **Keep swap**

5 Evaluate function **P(ΔL, T)**
P = probability we accept swap
Randomly keep "**with probability P**"

Idea: *Probabilistic* Swap Accept Criterion



ΔL = Change in length
= **Longer (positive)**

T

$$e^{\frac{-\Delta L}{T}}$$

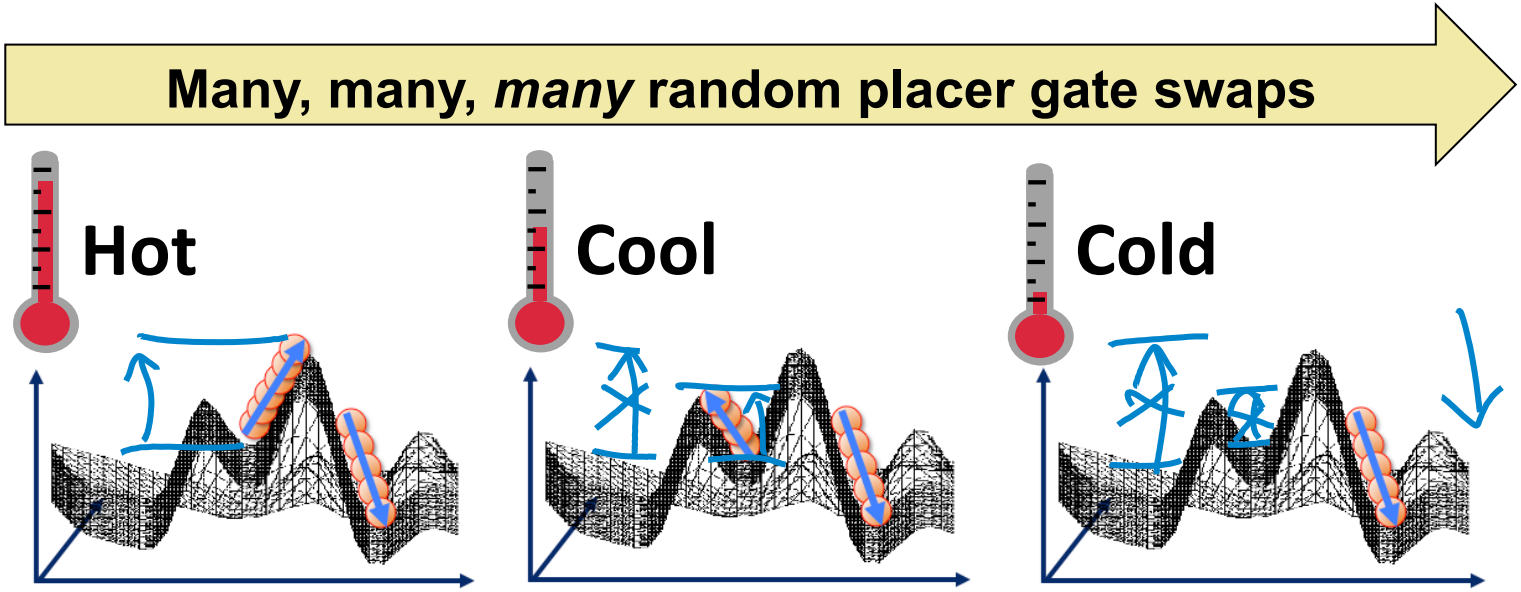
…a **probability P**, like 0.72

R = uniform random number between 0 and 1

if (**R < P**)
then **Keep** the swap
else **Undo** the swap

New Random Improvement Algorithm

- **Almost same as old, greedy one, with 2 big changes**
  - Hill-climbing **T**, start **T=Hot=BIG**, slowly reduce **T** over many swaps
  - If a swap makes wirelength worse, **randomly accept it**, with probability **P(ΔL, T)**



VLSI CAD: Logic to Layout

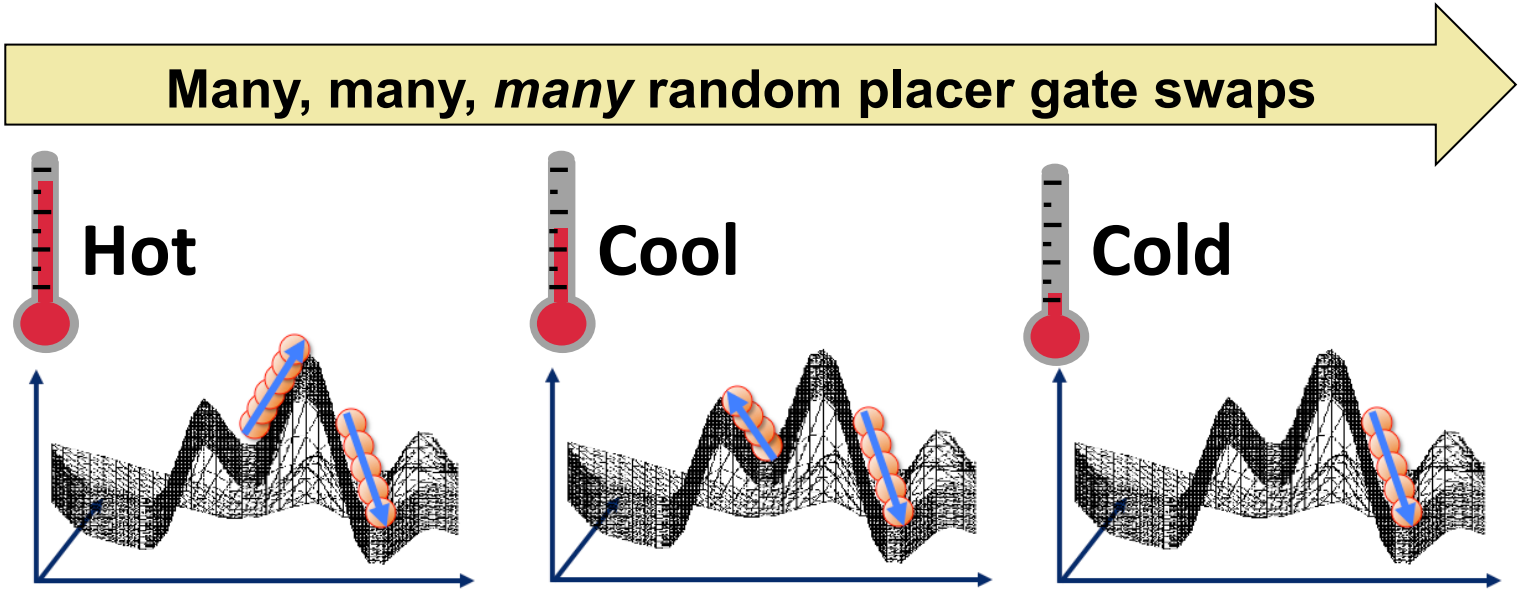
Lecture 9.5

ASIC Placement: Simulated Annealing Placement

Rob A. Rutenbar
University of Illinois

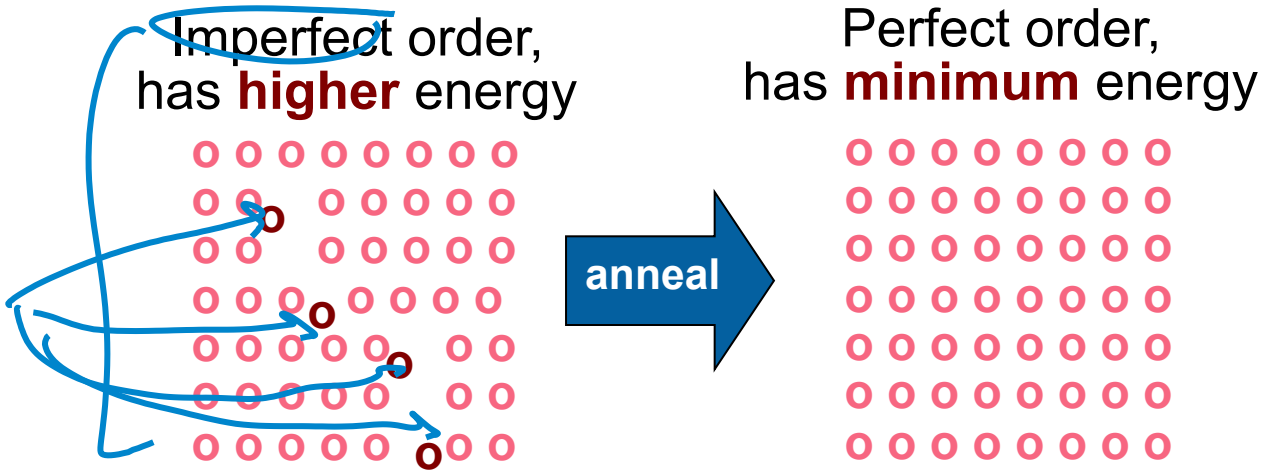
New Famous Algorithm: Simulated Annealing

- **Almost same as old, greedy one, with 2 big changes**
  - Hill-climbing **T**, start **T=Hot=BIG**, slowly reduce **T** over many swaps
  - If a swap makes wirelength worse, **randomly accept it**, with probability **P(ΔL, T)**



Physical Analogy Here: *Annealing*

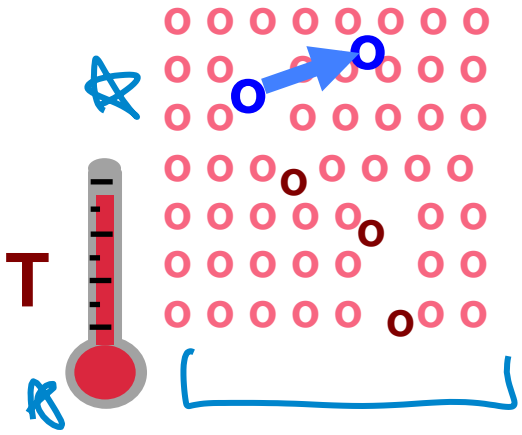
- **Suppose you want to make a *perfect crystal***
  - **Perfect** = all atoms lined up on crystal lattice sites; lowest energy "state" of the atoms
  - **How you do it:** Get the material really **hot** (so atoms have energy to move around, even to bad places). **Cool** very, very slowly (so atoms settle in "good" spots)
  - This process has a name: **Annealing**



Physical Analogy Here: *Annealing*

- **Suppose you want to simulate this crystal with a computer program**

Move 1 atom
Evaluate **ΔE** =
change in **Energy**



To model a real physical crystal, what is **probability** of this atomic move, if **ΔE<0**?

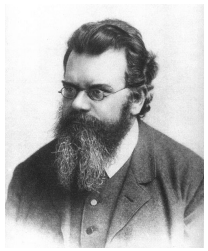
Answer: 1

To model a real physical crystal, what is **probability** of this atomic move, if **ΔE>0**?

Answer:

$$e^{\frac{-\Delta E}{KT}}$$

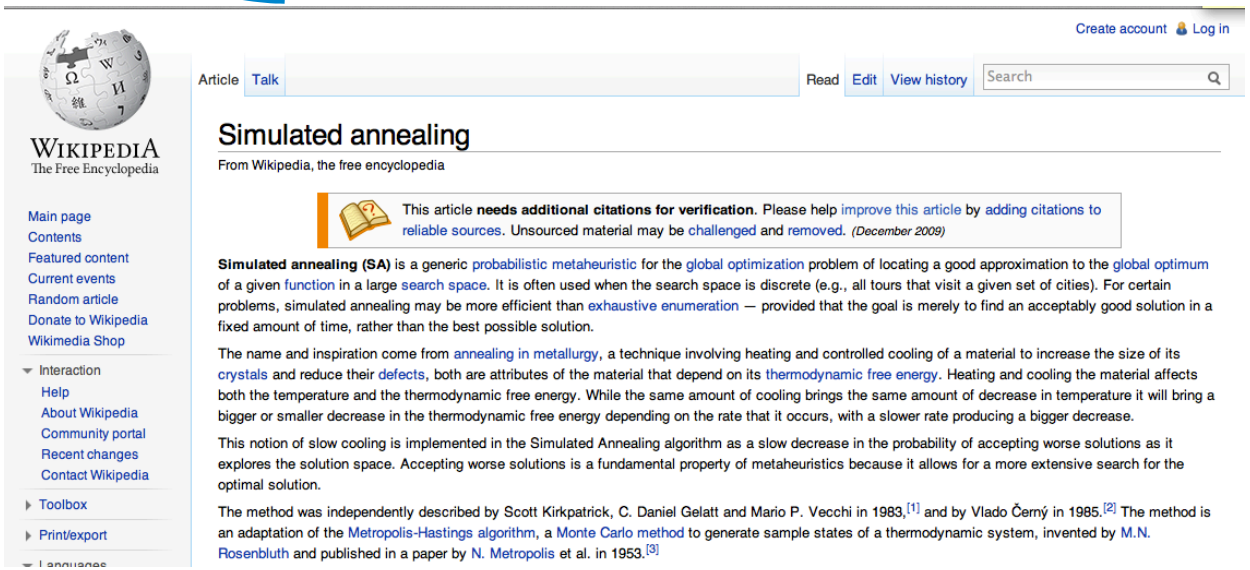
K=Boltzmann constant. In *real* physics, **K** converts units of Temperature to units of Energy



http://en.wikipedia.org/wiki/Ludwig_Boltzmann

This New Method: *Simulated* Annealing

- **General optimization method, used widely in VLSI CAD**
  - Invented at IBM in early 1980s.
  - Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. (1983). "Optimization by Simulated Annealing". Science 220 (4598): 671–680.



Create account Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export
Languages

Simulated annealing

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(December 2009)*

Simulated annealing (SA) is a generic probabilistic metaheuristic for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems, simulated annealing may be more efficient than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects, both are attributes of the material that depend on its thermodynamic free energy. Heating and cooling the material affects both the temperature and the thermodynamic free energy. While the same amount of cooling brings the same amount of decrease in temperature it will bring a bigger or smaller decrease in the thermodynamic free energy depending on the rate that it occurs, with a slower rate producing a bigger decrease.

This notion of slow cooling is implemented in the Simulated Annealing algorithm as a slow decrease in the probability of accepting worse solutions as it explores the solution space. Accepting worse solutions is a fundamental property of metaheuristics because it allows for a more extensive search for the optimal solution.

The method was independently described by Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983,[1] and by Vlado Černý in 1985.[2] The method is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system, invented by M.N. Rosenbluth and published in a paper by N. Metropolis et al. in 1953.[3]

Annealing Placer: Pseudo-code

```
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) after swap] - [∑HPWL(net) before 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)
```

New: Annealing Temperature cooling "outer loop"

New: Probabilistic swap accept

Probabilistic Acceptance of Swaps

if(**uniform_random**() < **exp**(**-ΔL/T**))
then accept this 'uphill' swap *//this is a new, **worse** placement*
else undo this 'uphill' swap

- **This little piece of code implements a famous idea**
  - **Idea**: that "randomly accepting" a perturbation of system, with specific probability, will correctly simulate the real physics of that real system
  - **Name**: **Metropolis Criterion**
  - Metropolis, along with Von Neuman and Ulam, invented the first **Monte Carlo** methods for simulation

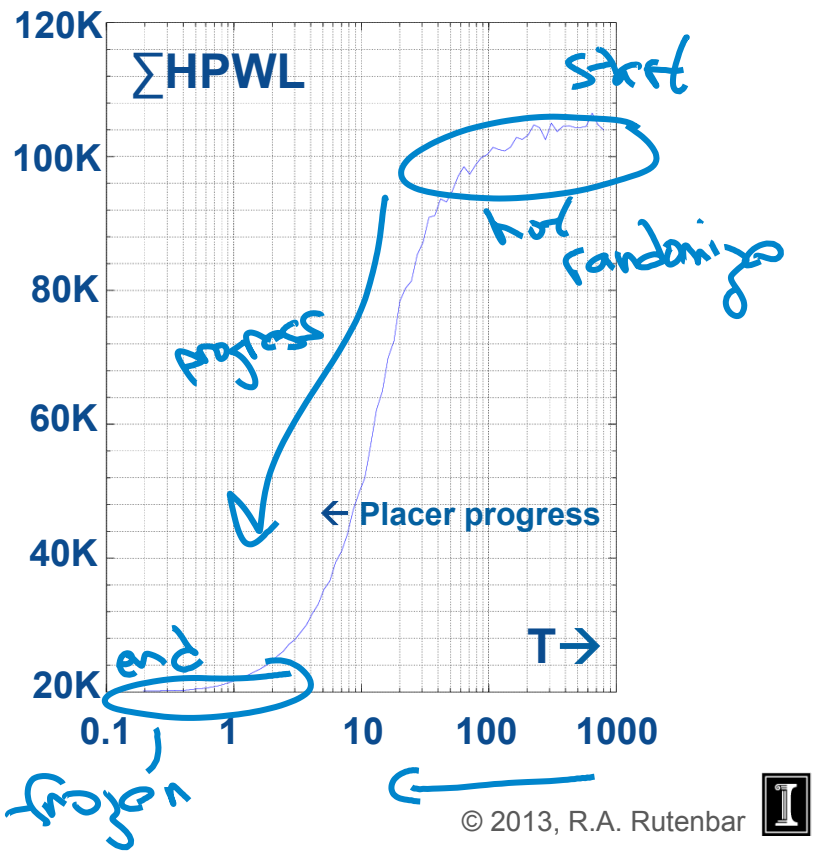
Metropolis Criterion: Behavior

if(**uniform_random**() < **exp**(**-ΔL/T**)
then accept this 'uphill' swap *//this is a new, **worse** placement*
else undo this 'uphill' swap

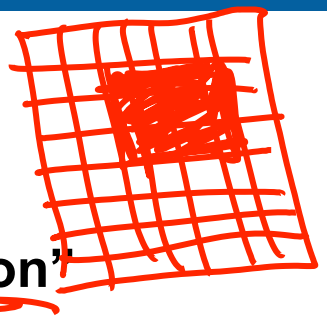
- **How does this really work?**
  - It really is **random**.
  - You might accept swap, you might not. Depends on **random number** you generate.
  - Suppose **T=10** and **ΔL=+20**. **exp(-ΔL/T) = 0.14**. So, **14%** probability to accept swap.
  - Suppose that at this temperature **T**, you try **5000 different** swaps that happen to each evaluate to **ΔL=+20**. Then, roughly **0.14×5000 = 700** of them will be accepted.
  - If you **change** the random number seed and run code again, get a **different** result…
  - …but, if you try **5000** moves with same **ΔL=+20,** still expect **~700** (different) accepts

How Well Does Annealing Work?

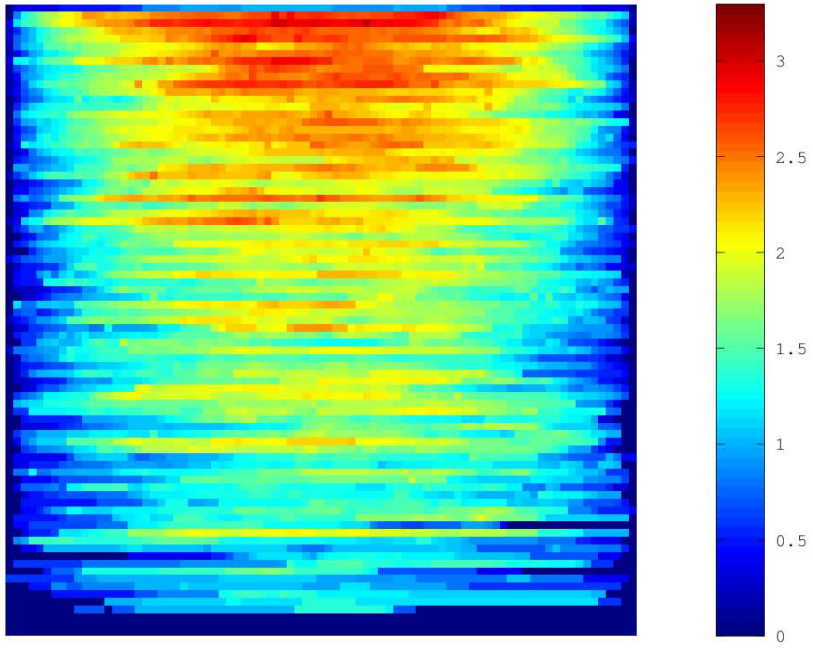
- **It works great**
  - Same 2500 gate benchmark
  - **HOT**=800, **M**=100 moves/gate
  - **About 30% less HPWL wirelength!**
- **Typical annealing "cost" curve**
  - **X** axis is **T**=temperature, LOG scale
    - Do 100x2500 swaps/temperature
    - Read right-to-left
  - **Y** axis is **∑HPWL**,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



Another View of Annealing Dynamics



- **Annealing placer**
  - ~6500 gates
- **Map is "net congestion"**
  - Picture at end of each temperature
  - Compute the BBOX of each net
  - We estimate a "routing congestion" number for each of these BBOX's
  - We add to each gate slot in grid, the "congestion" number for each net BBOX
    - Simple model of wire "complexity"
    - Red = BAD, Blue = GOOD



About Simulated Annealing, In General

- **Frequently Asked Questions (FAQs)**
  - Does annealing always find the perfect, **best global** optimum?
    - **NO**. It is just good at avoiding a lot of local minimums
  - Does annealing work on **every type** of optimization problem?
    - **NO**. But it does work on many. But it is not always the most efficient option
  - Is annealing always **slow**– doing all those many swaps over many temperatures?
    - **NO.** Lots of engineering tricks to speed it up
  - Do I have to set all the parameters – HOT, M swaps/gate – by **trial and error**?
    - **NO**. There are fancy adaptive techniques to determine these automatically
  - What happens if I run my annealer several **different times** (different random seeds)?
    - You get a **different** (random!) answer each time. Yes, this feels strange…

Annealing for Placement: Summary

- **Simulated annealing is a very famous, 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. **Important to understand!**

- **But… not how we really do placers today**
  - Annealing works well for up to 100K – 500K gates
  - Annealing too inefficient for things with 1M+ gates
  - Need yet **another** new set of ideas…

VLSI CAD: Logic to Layout

Lecture 9.6

ASIC Placement:
Analytical Placement:
Quadratic Wirelength Model

Rob A. Rutenbar
University of Illinois

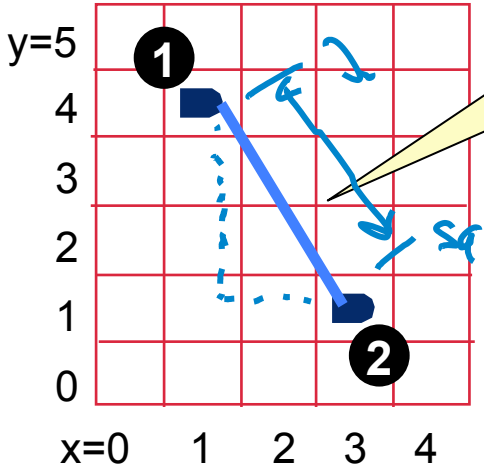
Analytical Placers: The Problem

- **Goal: Write an *equation* whose *minimum* is the placement(!)**
  - If you have a million gates, need a million **(xi,yi)** values as result
  - Formulate an appropriate **cost function** for all the gate-level **(xi, yi)**:
    $F(x_1, x_2, \ldots x_{1M}, y_1, y_2, \ldots y_{1M})$
  - …then solve **analytically** for **$X^*=(x_1, x_2, \ldots x_{1M})$**, **$Y^*=(y_1, y_2, \ldots y_{1M})$** to **minimize F( )**
  - The resulting set of values of **X*, Y*** give you the placement of all 1M gates

- **This sounds sort of crazy…but it works great**
  - All modern placers for big ASICs and SOCs are "analytical"
  - Big trick is write the **wirelength** in mathematically "friendly" form we can optimize

Idea: Optimize *Quadratic* Wirelenth Model

- **For 2-point net: squared length of "distance" line between points**
  - Quadratic length = $(x1-x2)^2 + (y1-y2)^2$



Quadratic wirelength
$=(3-1)^2 + (4-1)^2$
$=13$

squared!

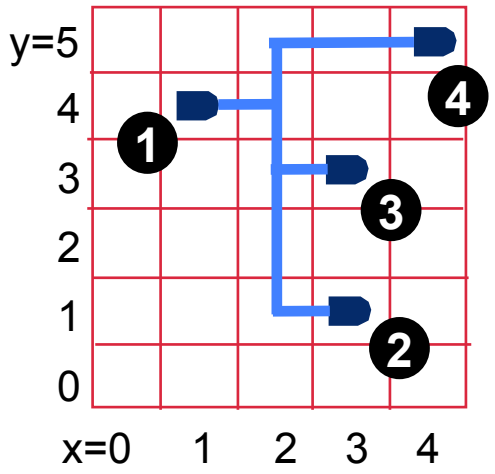
BUT… what happens if your net has *more* than 2 points in it?

What About k-point Net, k>2?

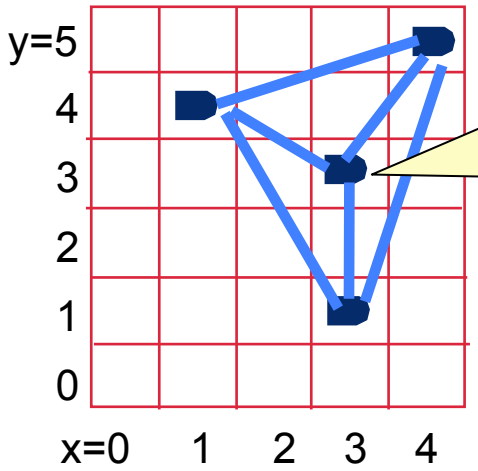
- **For k-point net:**
  - **Replace** one "real" net with **k(k-1)/2** 2-point nets
  - Add a new net between every **pair of points**. Called a **fully-connected clique model**
  - (We use this model for all our subsequent examples)

clique

A "k=4-point net"



Clique model



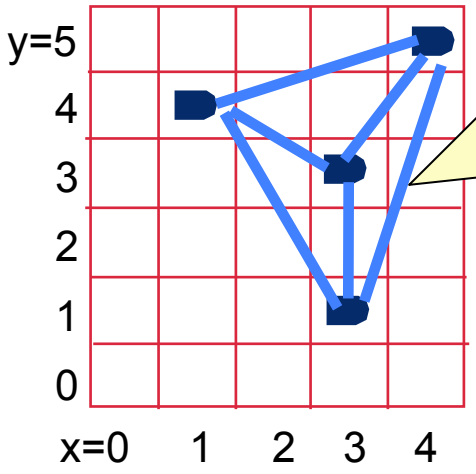
k(k-1)/2
=4(4-1)/2
=6 2-point nets

What About k-point Net, k>2?

Note also: when **k=2**, this weight is just **1/(2-1)=1**, so no special treatment for common 2-point nets

- **k-point net, continued**
  - And, each new 2-point net is **weighted (multiplied)** by term: **1/(k-1)**
  - Why? **1** net became **k(k-1)/2** nets. Need to **compensate** so we don't "overestimate"

Clique model



Quadratic estimate:

$$(1/3)[(4-1)^2 + (5-4)^2]$$
$$+(1/3)[(4-3)^2 + (5-1)^2]$$
$$+(1/3)[(3-1)^2 + (4-1)^2]$$
$$+(1/3)[(3-1)^2 + (4-3)^2]$$
$$+(1/3)[(4-3)^2 + (5-3)^2]$$
$$+(1/3)[(3-3)^2 + (3-1)^2]$$

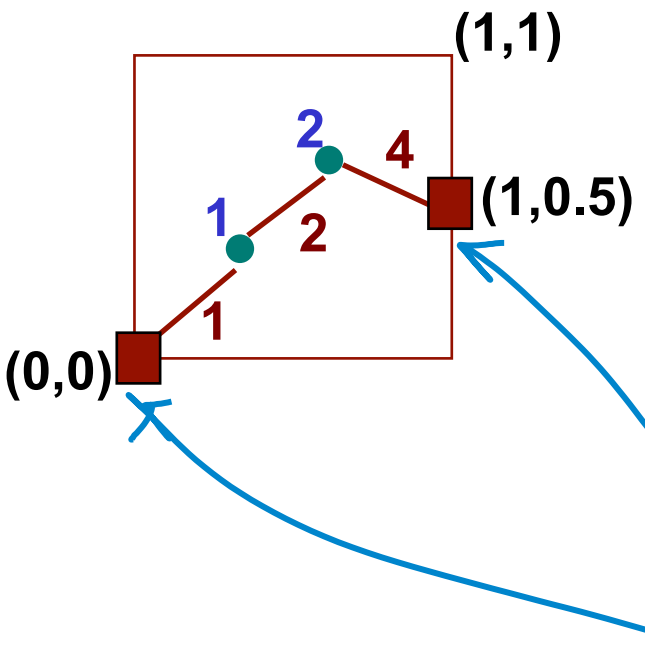
=sum of 6 weighted 2-point lengths

One More Big Idea: Gates as "Points"

- **To make the math work out easily, one more simplification:**
  - Ignore the physical size of all the gate – pretend **gates are dimensionless points**
  - And, we will **ignore** (for now…) constraint that gates can't go on top of each other

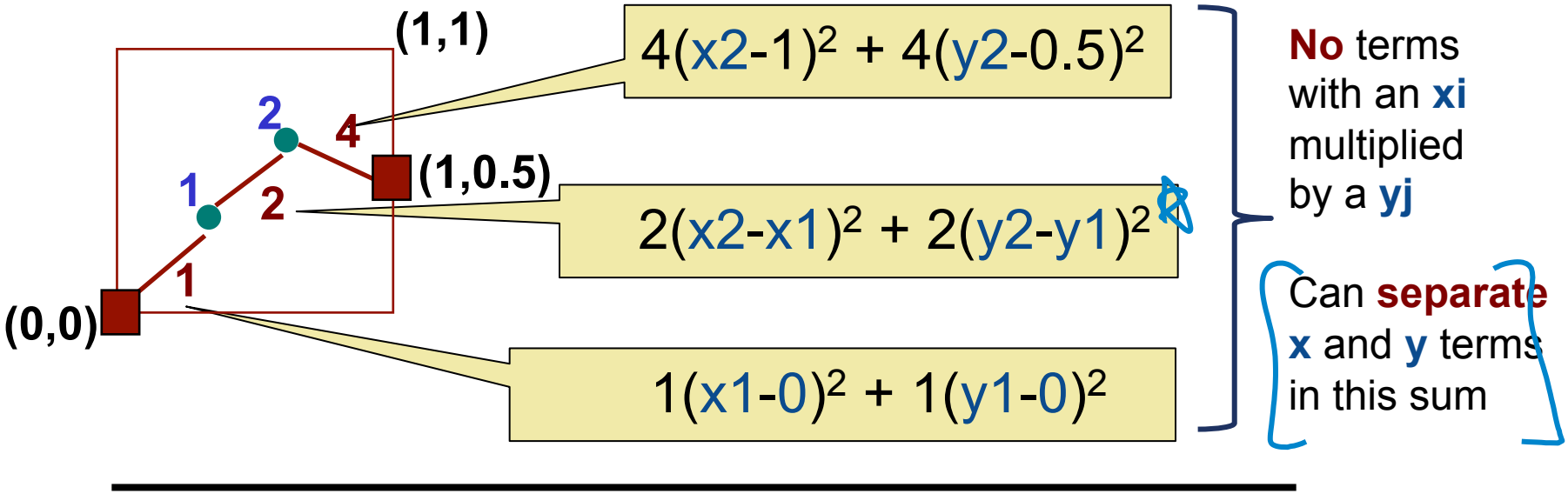
- **Sounds strange. Why…?**
  - Allows us to write a very simple, very elegant "equation" for the placement
  - And, we can solve it, quickly and effectively
  - And, we can then use another set of methods – later in this lecture – to repair this

Easiest to See With Small Example



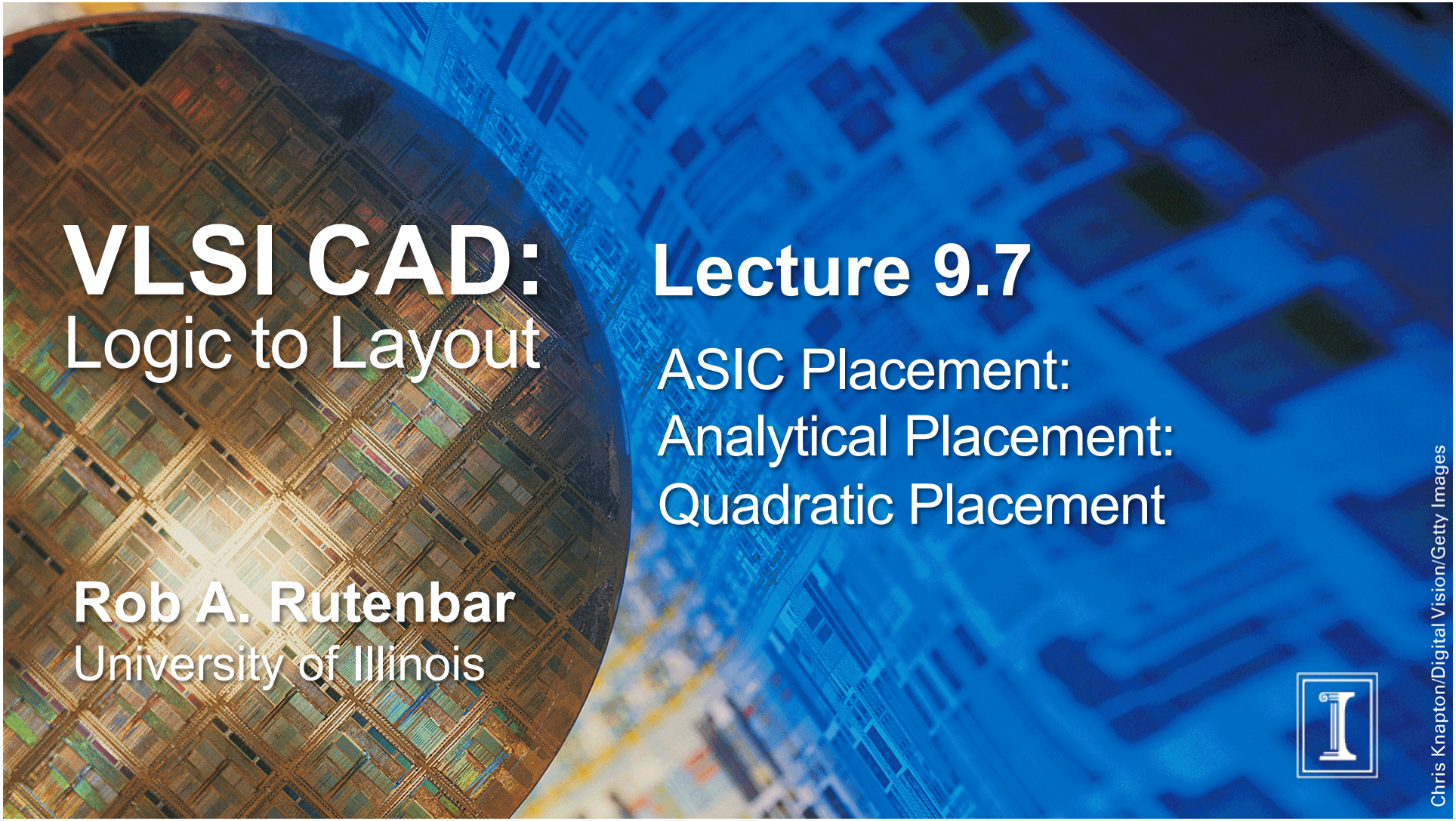
- **Chip surface is a rectangle**
  - **X** from 0 to 1; **Y** from 0 to 1
  - This it totally arbitrary, btw
- **2 gate "points", index 1 and 2**
- **3 nets, each with a weight**
  - Each net is 2 points to keep manual example small and easy
  - Weights are **1, 2, 4** in diagram
- **2 pads**
  - **Pad** = **fixed** pin (red square) on the edge of the chip. These do not move.

Easy to Write the Quadratic Wirelength



Add these together: this is the **Quadratic Wirelength**

So… how do we optimize this…?



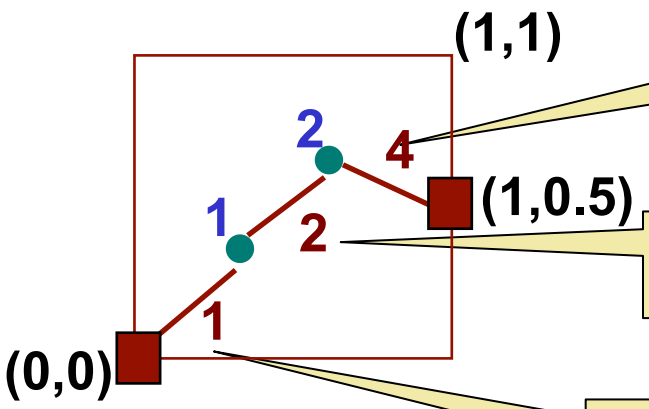
VLSI CAD: Logic to Layout

Lecture 9.7

ASIC Placement:
Analytical Placement:
Quadratic Placement

Rob A. Rutenbar
University of Illinois

Example of Quadratic Wirelength Model



$4(x2-1)^2 + 4(y2-0.5)^2$

$2(x2-x1)^2 + 2(y2-y1)^2$

$1(x1-0)^2 + 1(y1-0)^2$

No terms with an **xi** multiplied by a **yj**

Can **separate** **x** and **y** terms in this sum

Add these together: this is the **Quadratic Wirelength**

Minimize it!

How Do We *Minimize* This?

- **Basic calculus!** **Differentiate, set derivative to 0, then solve!**
  - But this is multiple variables? So, we do **partial derivatives**, set each to 0, solve

Q(X): $4(\mathbf{x2}-1)^2 + 2(\mathbf{x2}-\mathbf{x1})^2 + 1(\mathbf{x1}-0)^2$

$\partial Q/\partial \mathbf{x1} = 0 + 4(\mathbf{x2}-\mathbf{x1})(-1) + 2(\mathbf{x1})$
$= 6\,\mathbf{x1} - 4\,\mathbf{x2} = 0$

$\partial Q/\partial \mathbf{x2} = 8(\mathbf{x2}-1) + 4(\mathbf{x2}-\mathbf{x1}) + 0$
$= -4\,\mathbf{x1} + 12\,\mathbf{x2} - 8 = 0$

Q(Y): $4(\mathbf{y2}-0.5)^2 + 2(\mathbf{y2}-\mathbf{y1})^2 + 1(\mathbf{y1}-0)^2$

$\partial Q/\partial \mathbf{y1} = 0 + 4(\mathbf{y2}-\mathbf{y1})(-1) + 2(\mathbf{y1})$
$= 6\,\mathbf{y1} - 4\,\mathbf{y2} = 0$

$\partial Q/\partial \mathbf{y2} = 8(\mathbf{y2}-0.5) + 4(\mathbf{y2}-\mathbf{y1}) + 0$
$= -4\,\mathbf{y1} + 12\,\mathbf{y2} - 4 = 0$

How Do We Minimize This?

- **Hey – these are linear equations! We know how to solve these!**

Q(X): $4(x2-1)^2 + 2(x2-x1)^2 + 1(x1-0)^2$ **Minimize**

$$\begin{pmatrix} 6 & -4 \\ -4 & 12 \end{pmatrix} \begin{pmatrix} x1 \\ x2 \end{pmatrix} = \begin{pmatrix} 0 \\ 8 \end{pmatrix}$$

x1 = 0.571 **x2** =0.857

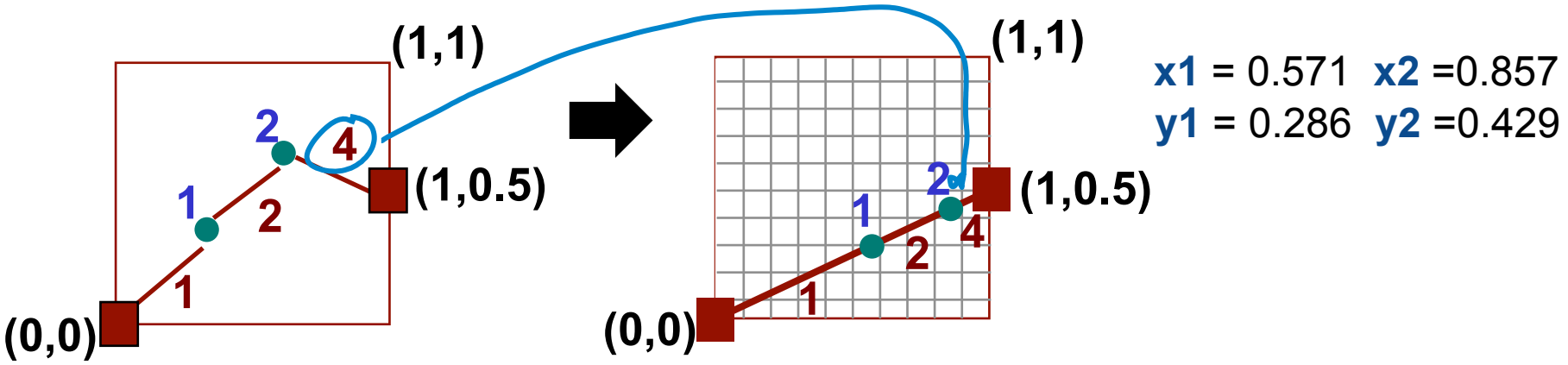
Q(Y): $4(y2-0.5)^2 + 2(y2-y1)^2 + 1(y1-0)^2$ **Minimize**

$$\begin{pmatrix} 6 & -4 \\ -4 & 12 \end{pmatrix} \begin{pmatrix} y1 \\ y2 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$$

y1 = 0.286 **y2** =0.429

- Two **matrix** equations: $Ax=b_x$ and $Ay=b_y$. If you have **N** gates, matrix is **NxN**
- **Same** matrix for **X,Y, different b** vectors. **X, Y, b** vectors also have **N** elements

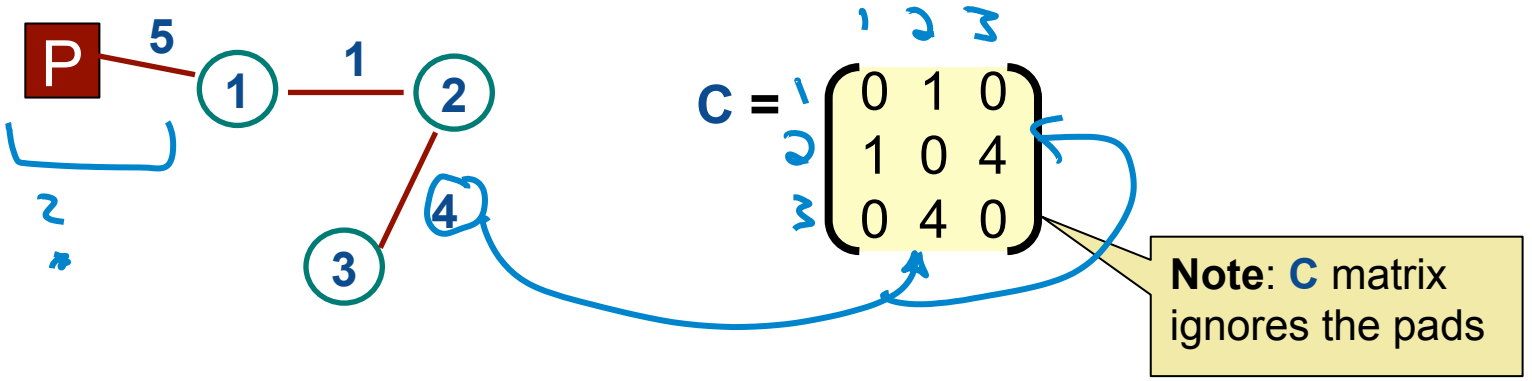
Placement Result



- **Observations**
  - Placement makes visual sense. All points on a **straight line** between the pads
    - Analog: each 2-point wire is like a **spring**. Placement minimizes all spring **lengths**
  - **Bigger** weight on the wire → **shorter** wire. Gives us lots of control over placement
  - **Same** matrix, **different** right-hand-side **b** vectors. Why? Different **x, y** pad coordinates

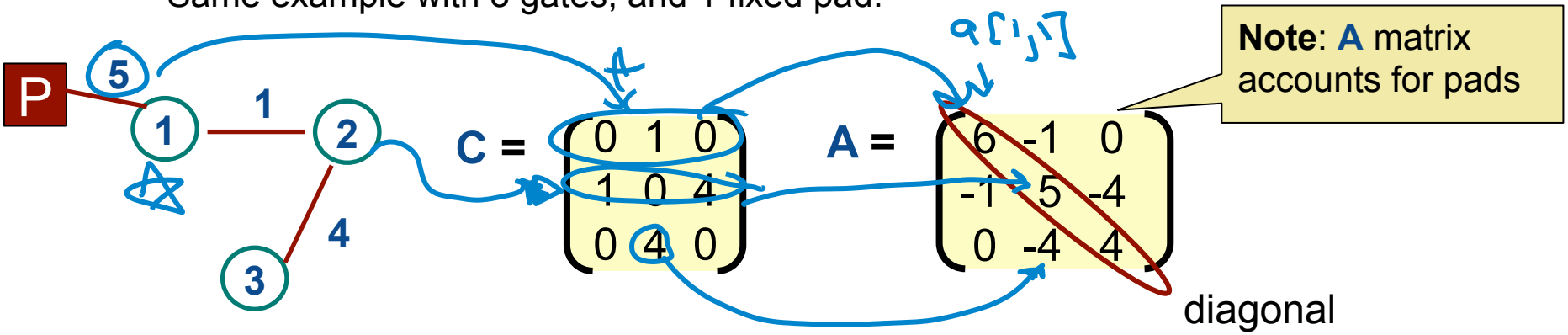
Quadratic Placement: What is Matrix **A**?

- **Surprisingly simple recipe to build the required A matrix**
  - First, build the **NxN** connectivity matrix, called **C**
  - If gate **i** has a 2-point wire to gate **j** with weight **w**, the **c[i,j] = c[j,i] = w**, else = **0**
  - New (bigger) example, with 3 gates, 3 wires (with weights) and 1 pad (**P**)

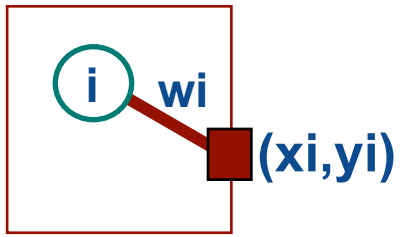


Quadratic Placement: What is Matrix **A**?

- **Use the connectivity C matrix to build A matrix**
  - Elements **a[i,j] not** on the matrix diagonal are just **a[i,j] = -c[i,j]**
  - Elements **on the diagonal** are **a[i,j]= $\sum_{j=1,n}$ c[i,j] + (weight of any pad wire)**
    - …ie, add up the **ith** row of **C** **and then** add in weight on a (possible) wire to pad
  - Same example with 3 gates, and 1 fixed pad.



How to Build **b** Vectors?



- **For Ax = b_x vector…**
  - If gate **i** connects to a pad at **(xi, yi)** with a wire with weight **wi**
  - Then set **b_x[i] = wi • xi**

$$\begin{bmatrix} A \end{bmatrix}\begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} b_x \end{bmatrix}$$

i^{th} element of b_x vector

- **For Ay = b_y vector…**
  - If gate **i** connects to a pad at **(xi, yi)** with a wire with weight **wi**
  - Then set **b_y[i] = wi • yi**

$$\begin{bmatrix} A \end{bmatrix}\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} b_y \end{bmatrix}$$

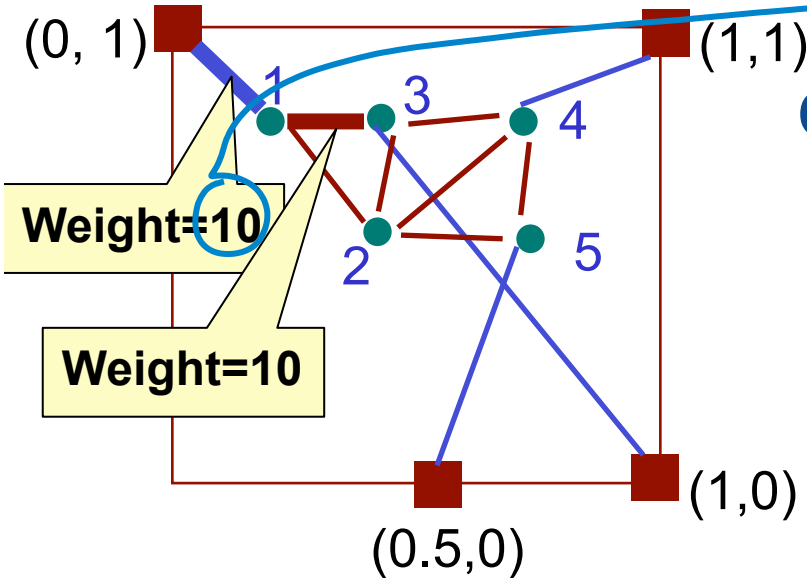
i^{th} element of b_y vector

About the **Ax=b** Matrix Solves

- **Are these difficult to do, in practice?**
  - If we have 1M gates, this is a 1M x 1M **A** matrix, with 1M element **x** and **b** vectors!

- **No – these are very easy to solve, even when very large**
  - The **A** matrix has a special form. It is sparse, symmetric, diagonally dominant
  - Mathematically: **A** is **positive semi-definite**. Very simple to solve!
  - We use **iterative**, approximate solvers, in practice (ie,not Gaussian elimination)
    - This means the solver converges gradually to the right answer
    - But, also means that the answers can be a little bit "off", not quite perfect

Example: 4 pads + Bigger 5-gate Netlist

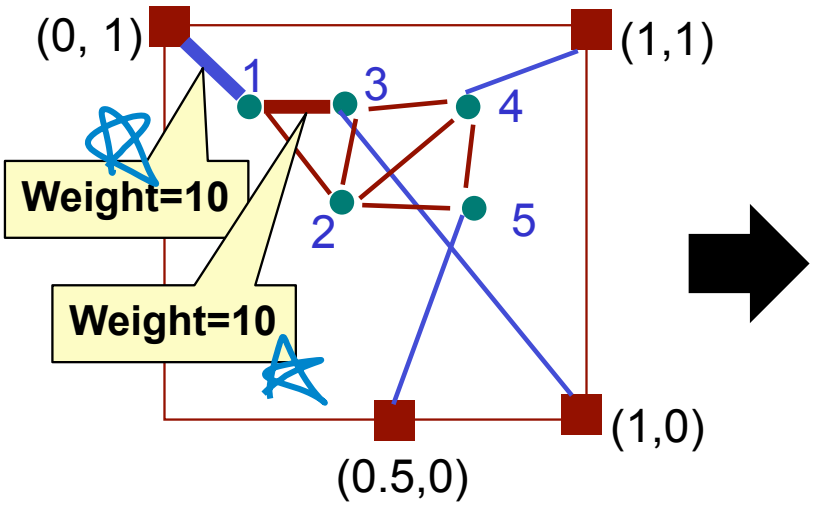


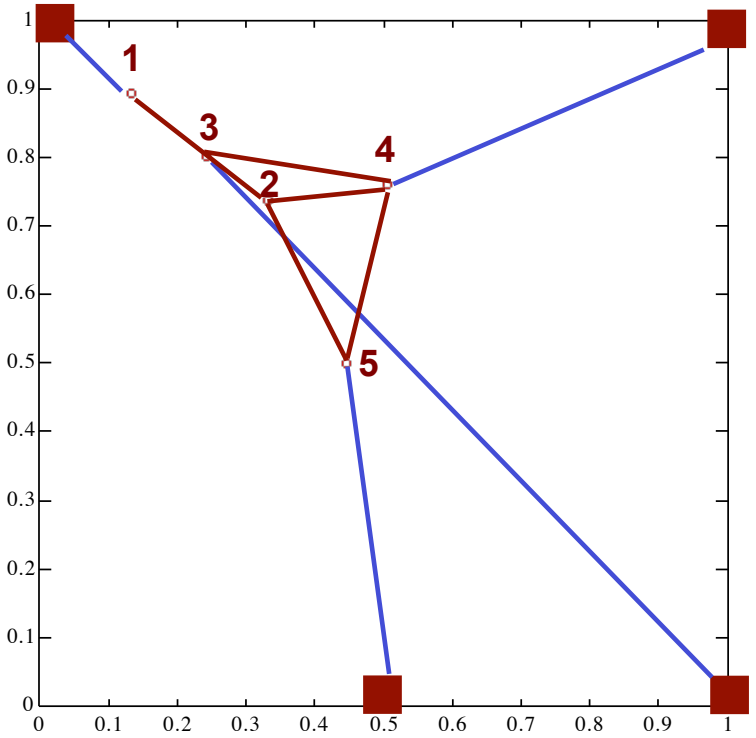
$$C = \begin{pmatrix} 0 & 1 & 10 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 10 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad A = \begin{pmatrix} 21 & -1 & -10 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ -10 & -1 & 13 & -1 & 0 \\ 0 & -1 & -1 & 4 & -1 \\ 0 & -1 & 0 & -1 & 3 \end{pmatrix}$$

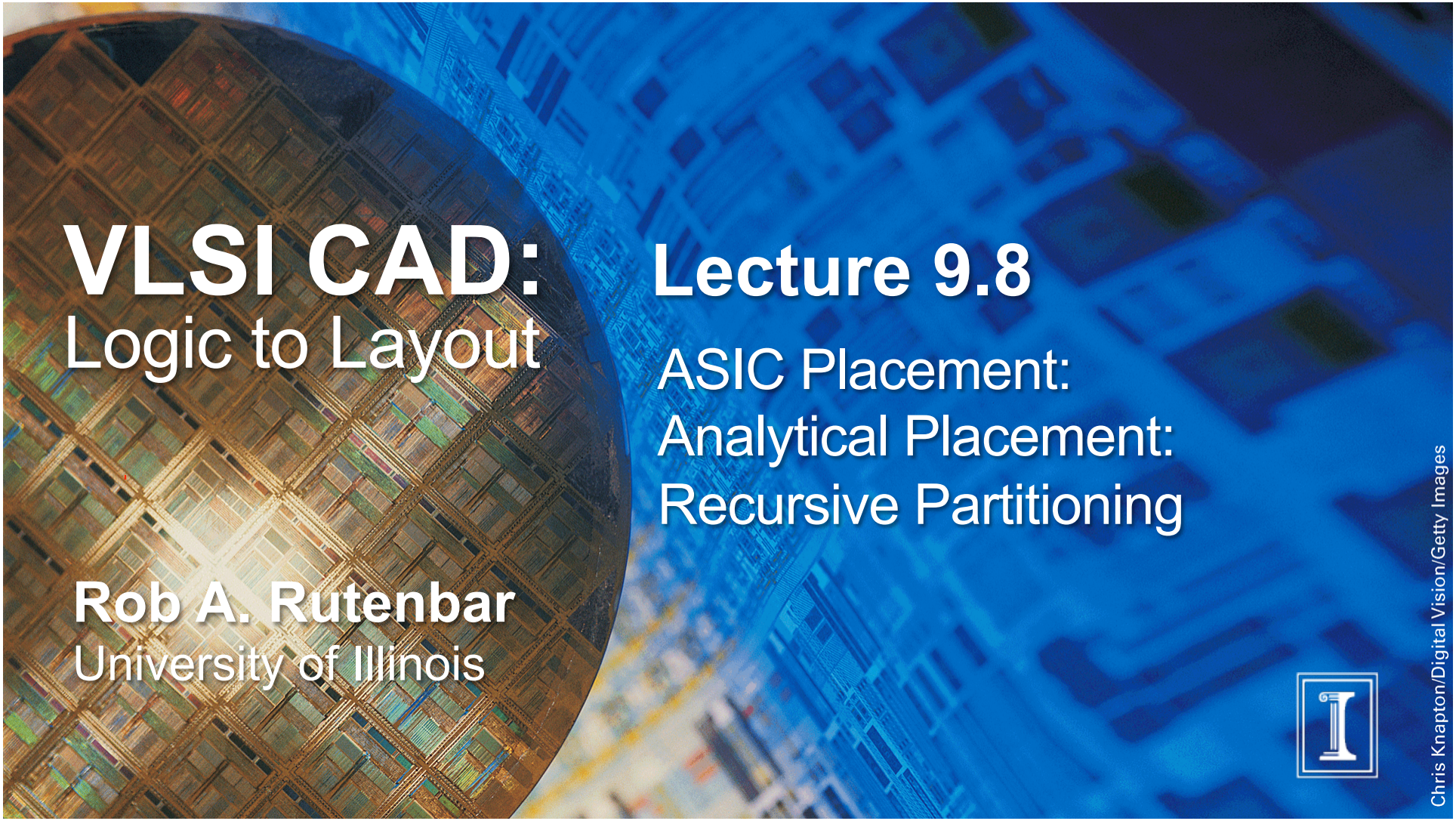
$$b_x = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0.5 \end{pmatrix} \quad b_y = \begin{pmatrix} 10 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

All wire weights = 1
except two highlighted:
gate1 to **pad**,
gate1 to **gate2**

Quadratic Placement Result







VLSI CAD: Logic to Layout

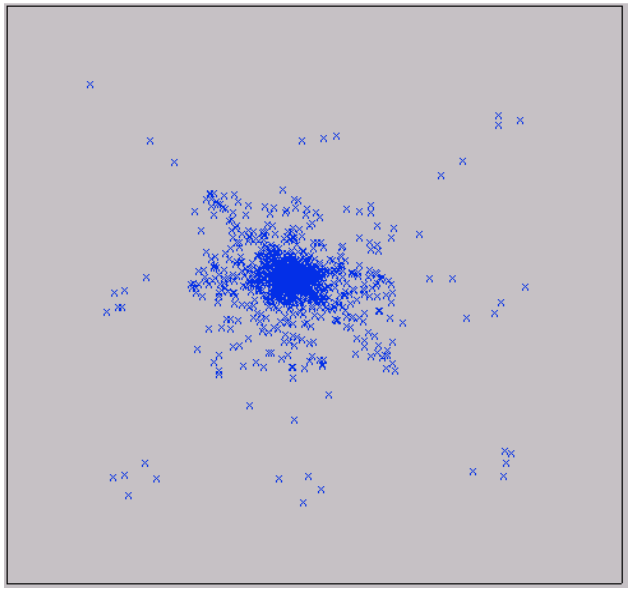
Lecture 9.8

ASIC Placement:
Analytical Placement:
Recursive Partitioning

Rob A. Rutenbar
University of Illinois

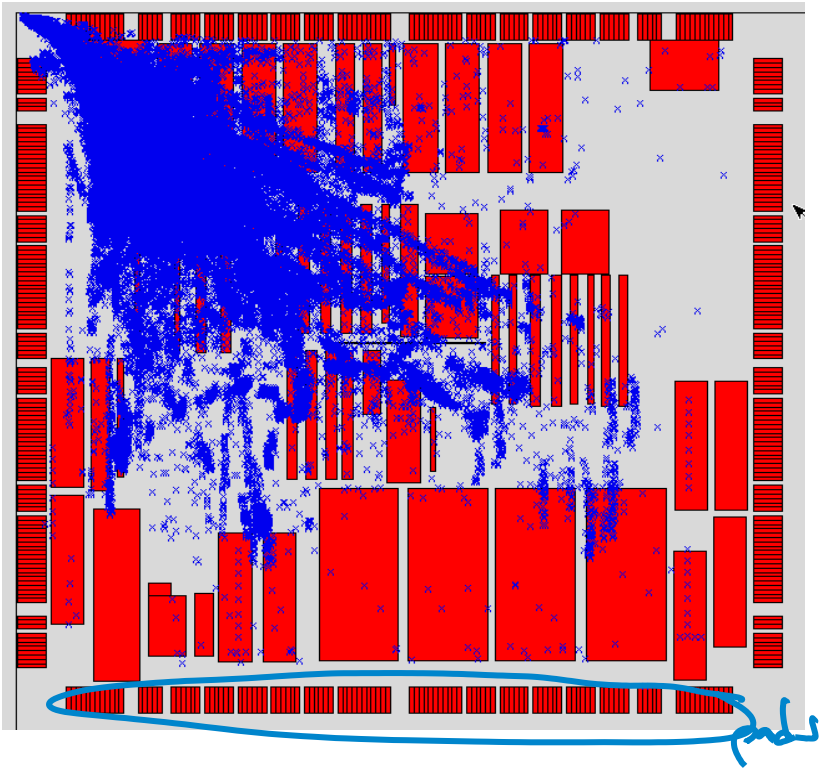
What Does A Real Quadratic Placement Look Like?

- **Like this:**
  - Small IBM ASIC, few thousand gates
- **New problem:**
  - Quadratic model minimizes wirelength for **big** netlists, in a numerical way
  - But ignores that gates have **physical size**, cannot be on top of each other
  - Now, we have to fix this…
  - Our solution: **recursive partitioning**

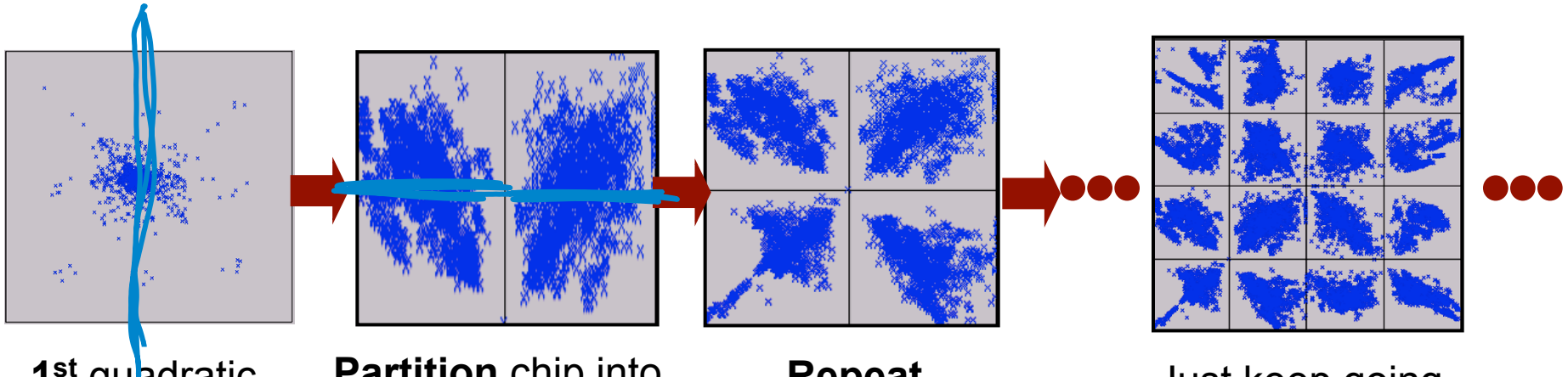


A Bigger Industrial Example

- **This benchmark is from IBM**
  - 210,904 gates (blue)
  - 543 fixed blocks (red) –like SRAMS
  - Image shows where gates "want" to go if we model blocks like "big pads"
  - This is a quadratic placement of gates
  - (Additional problems: gates need to go between these blocks!)
- **Why we are showing this**
  - Example of how **badly imbalanced** the quadratic placement can be



Big Idea: Recursive Partitioning



1st quadratic place (**QP**) solve

Partition chip into left / right. **Select** which gates on each side. **Solve** 2 new, smaller **QP** tasks

Repeat. **Partition** each side top/bottom. **Solve** 4 new smaller **QP** solves

Just keep going

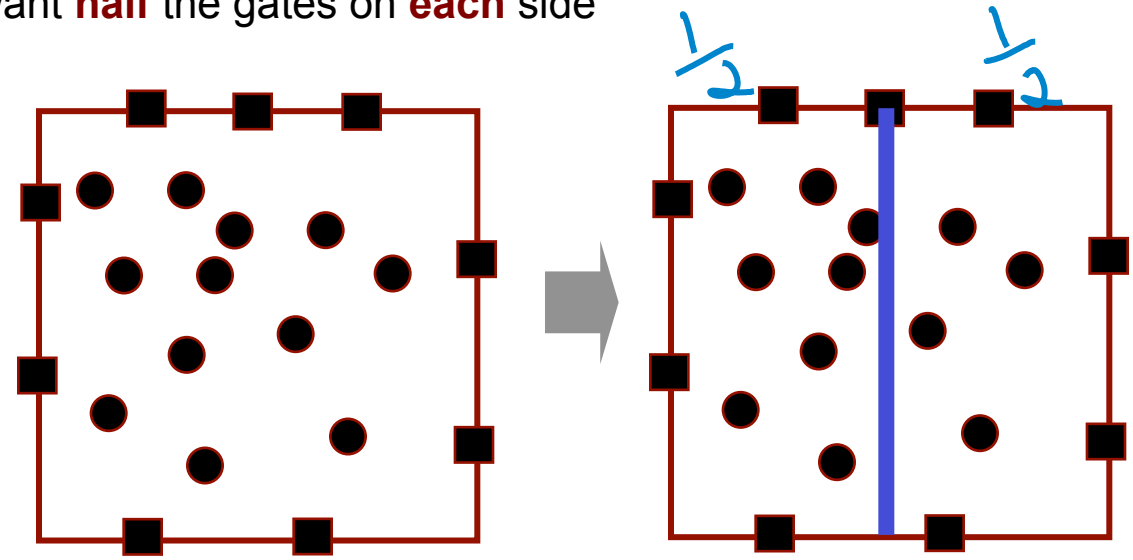
Recursive Partitioning: Basic Steps

- **Partition**
  - How do we divide the chip into new, smaller placement tasks?
- **Assignment**
  - Which gates should go into each new, smaller region?
- **Containment**
  - Formulate new **QP** matrix solves so gates stay in new regions, with short wirelength

- **Discuss one early strategy from a classical paper**
  - Ren Song Tsay, Ernest Kuh, Chi Ping Hsu, "PROUD: A Sea-Of-gates Placement Algorithm," *IEEE Design & Test of Computers*, Dec 1988.

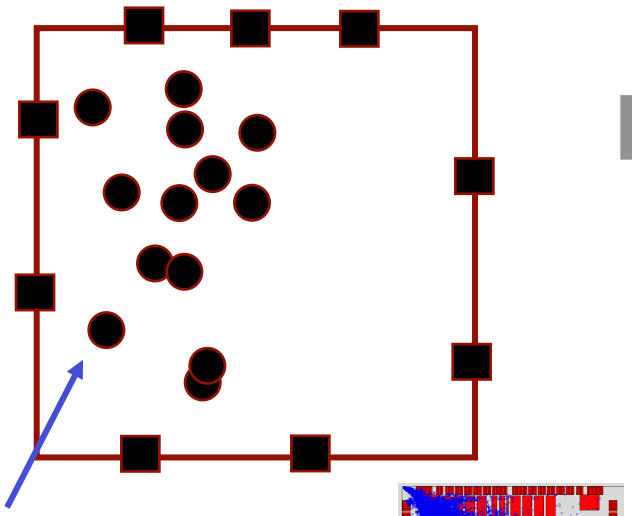
Recursive Partitioning: How to Partition

- **Solution**
  - After 1st quadratic placement (**QP**), divide chip area **exactly in half**, vertically
    - Note: this is arbitrary. Horizontal is OK too.
  - We want **half** the gates on **each** side

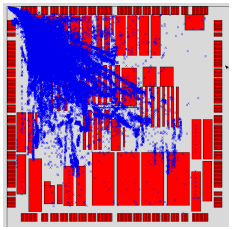


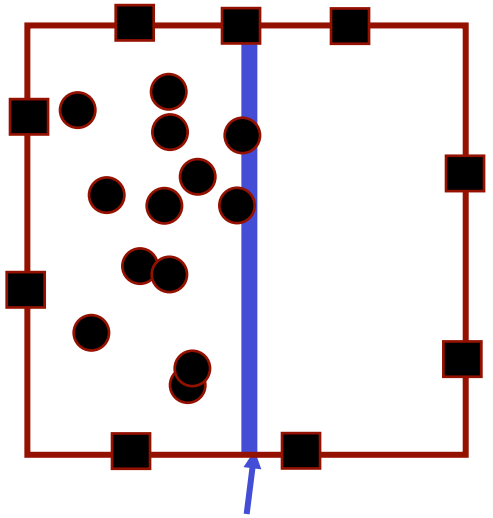
Recursive Partitioning: How to Assign

- **Problem:** What if **QP** does not spread gates **evenly** between halves?



How do we know which gates to put **left vs right** if this is initial **QP**?

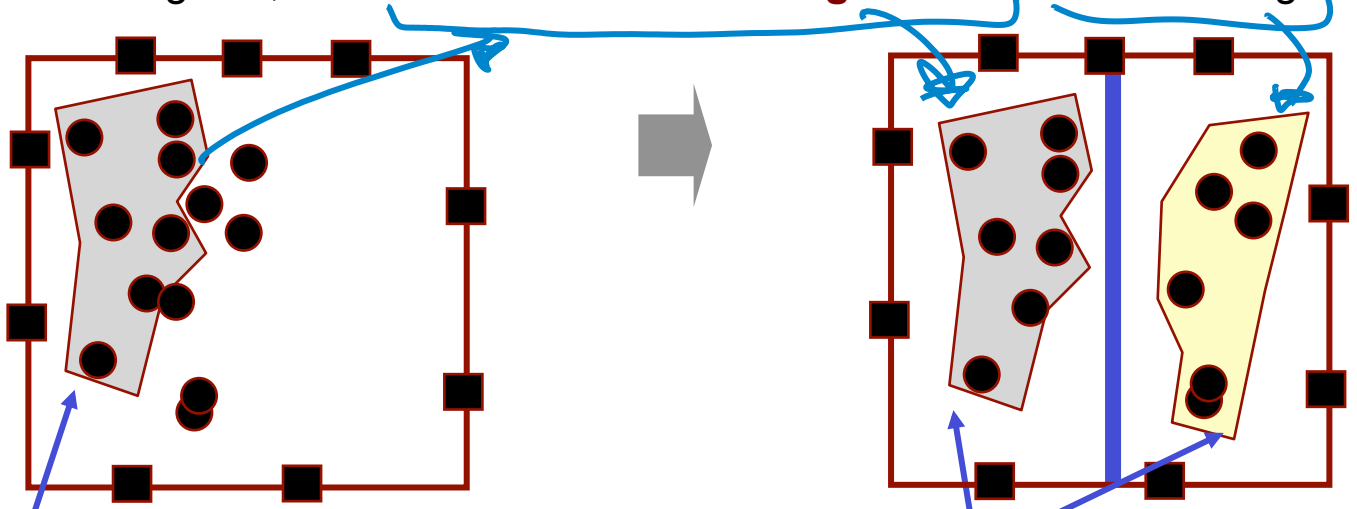




We partition region at **exact center**. How do we reformulate a new **QP** to **re-do** the gates on each side?

Recursive Partitioning: How to Assign

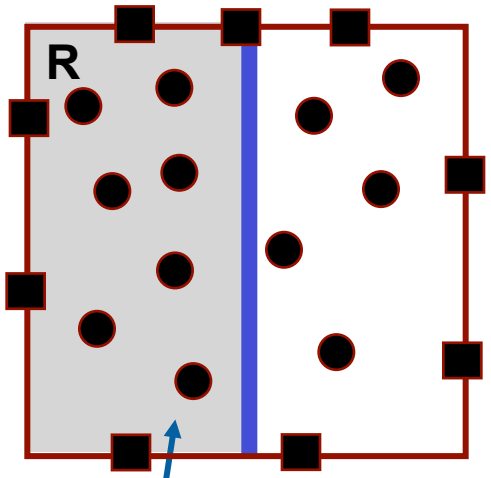
- **Solution: Sort the gates**
  - Sort placed gates on **X** coordinate, then **Y** (For horizontal cut – sort on **Y** first, then **X**)
  - If **N** total gates, then **first N/2 in sorted list go on left**. Others on right



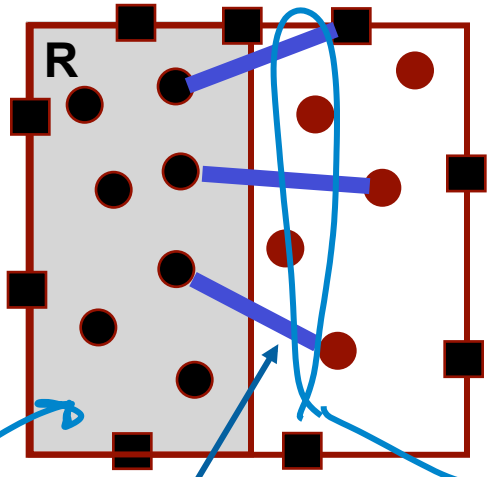
Sorted on **X** coordinate, these are left-most **N/2** of **N** gates

This **sorting** assigns which gates go left, and go right

Recursive Partitioning: How to Contain



Focus on the gates inside this shaded region **R** on the **left** side of the cut.



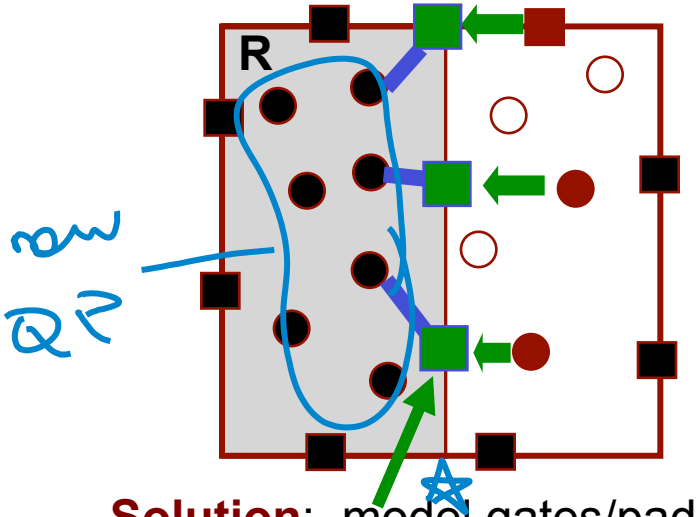
Big questions:

(1) How do we **keep** the gates assigned to left side actually **in** the left side?

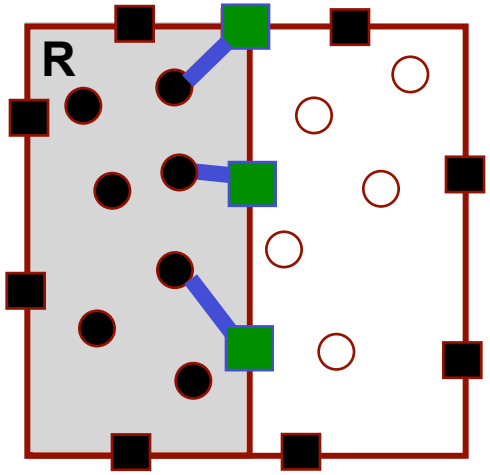
(2) How do we model wires that **connect** to gates/pads on **right**? Can't ignore these!

Recursive Partitioning: How to Contain

- **Idea: Pseudo-pads**
  - Every **gate & pad not** inside region **R** is modeled as a **pad on boundary** of **R**
  - **Propagate** these outside gates using their current **(x,y)** location to **nearest point** on **R**
  - For this simple first cut, we just take the **y coordinate**, and put pad on center cut line **x**



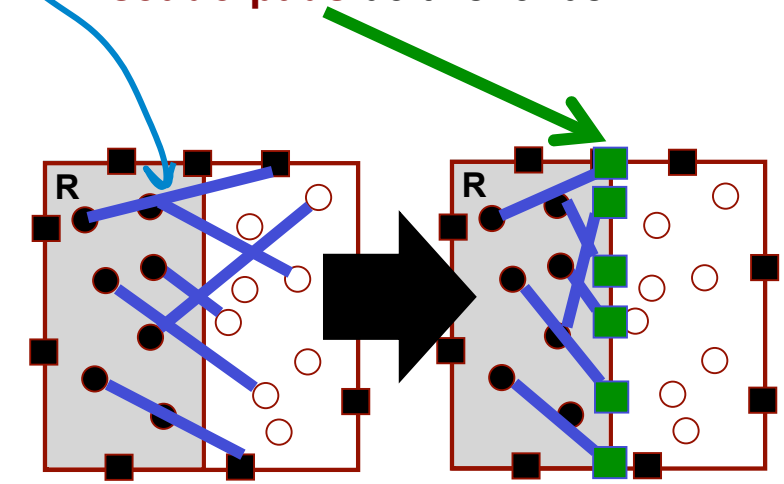
Solution: model gates/pads on right as "fake" pads on left.



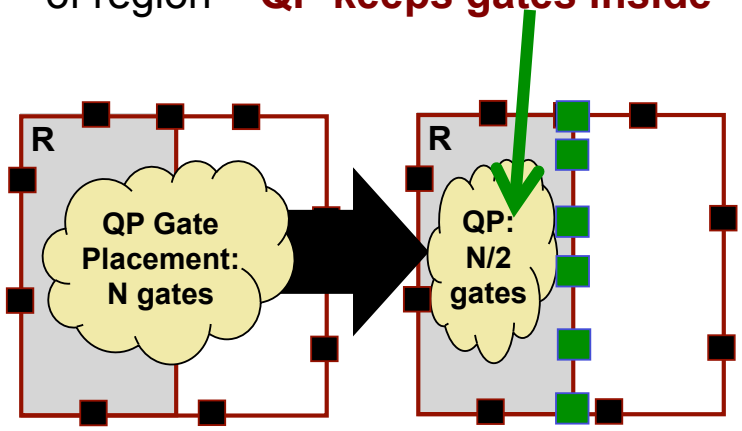
Resulting **new QP** problem for gates in left region

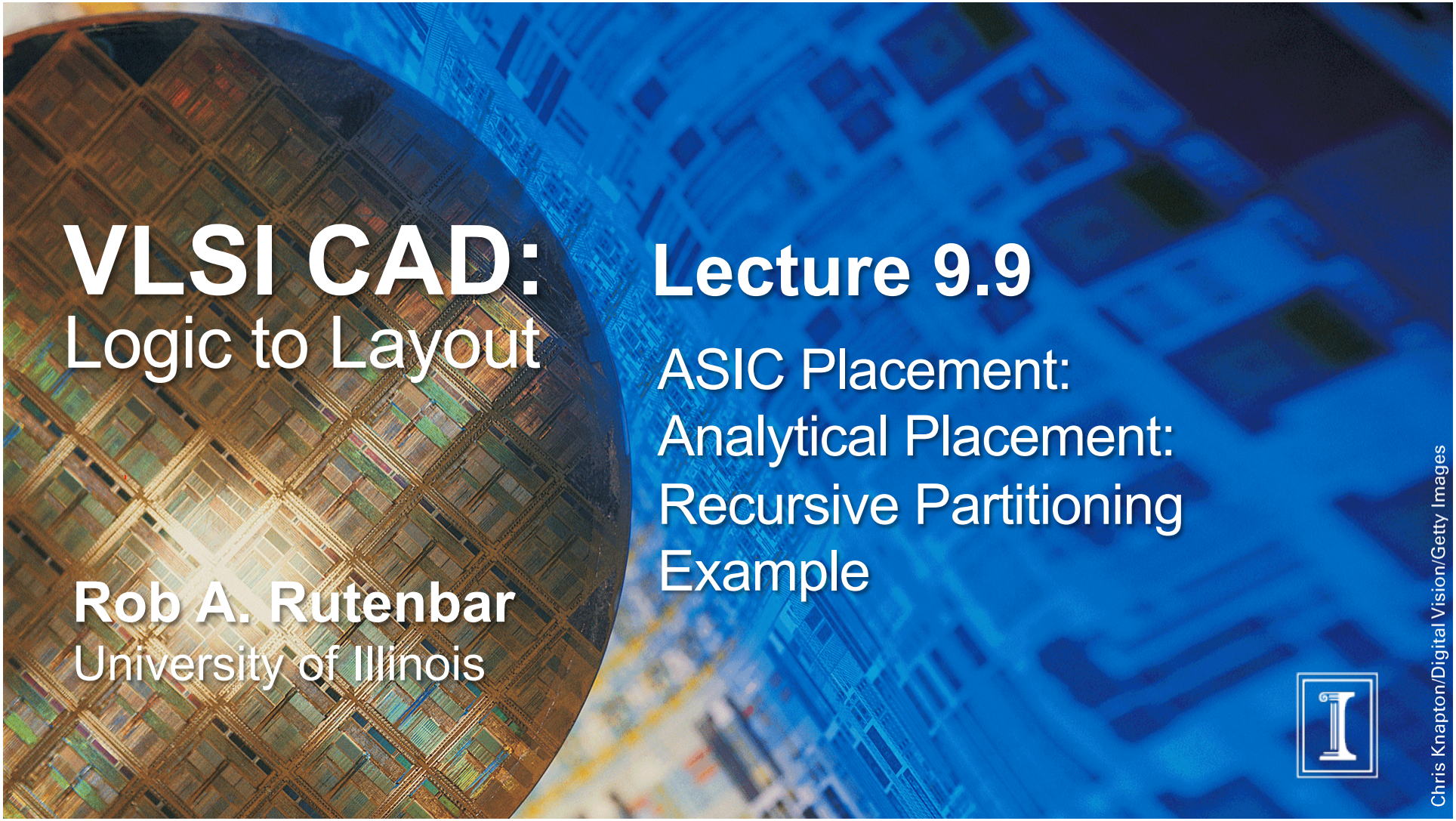
Why Containment+Propagation Is *Critical*

- **Cannot ignore gates outside region we are re-placing**
  - Want gates inside to "**feel pull**" from wires to **gates outside** region
  - **Pseudo-pads** do this for us



- **Pseudo-pads guarantee all gates re-locate inside region**
  - Think of wires as 'springs' that each pull gates **toward** other gates or pads
  - If pads (real & pseudo) are on edges of region – **QP keeps gates inside**





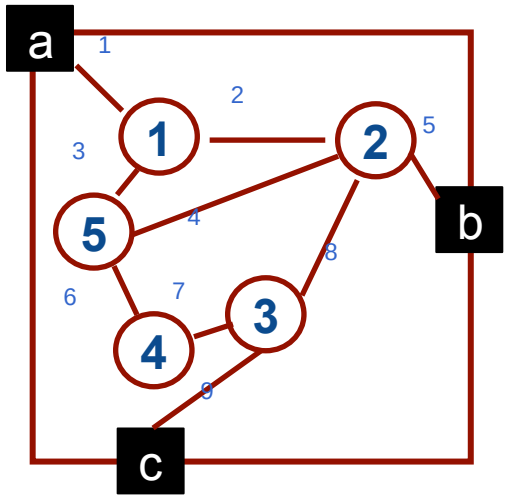
VLSI CAD: Logic to Layout

Lecture 9.9

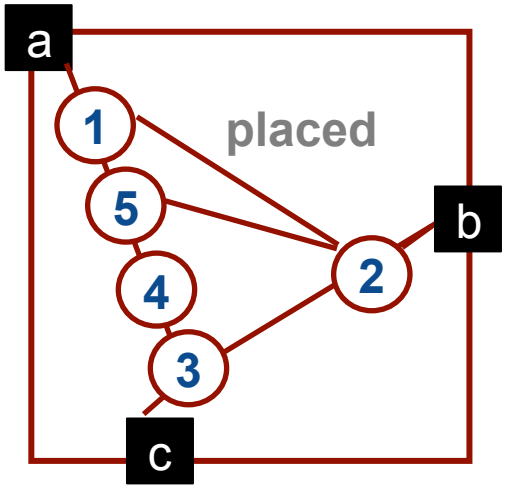
ASIC Placement:
Analytical Placement:
Recursive Partitioning
Example

Rob A. Rutenbar
University of Illinois

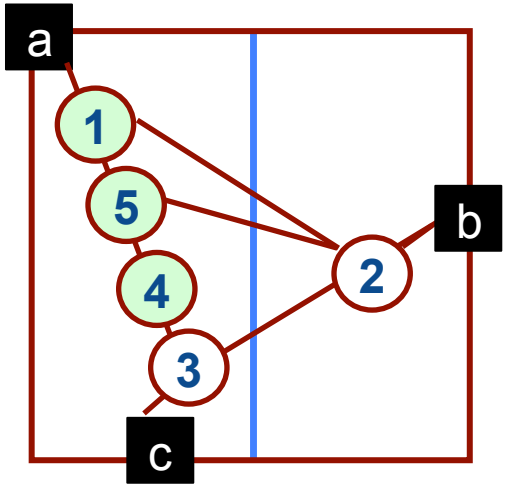
Small Partitioning Example



1. Initial netlist
5 gates (1,2,3,4,5)
9 wires
3 pads (a,b,c)

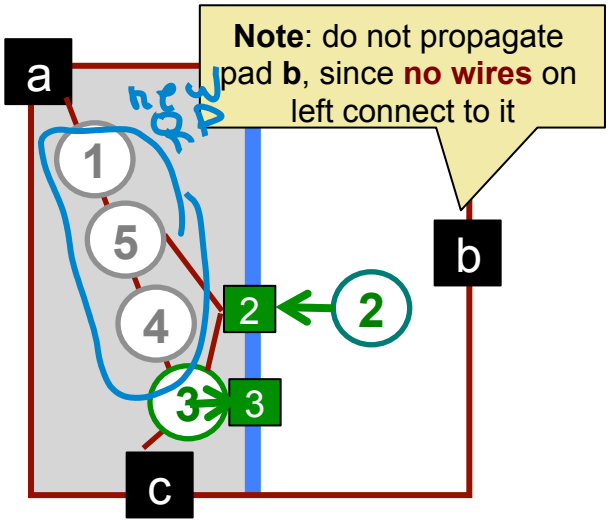


2. Initial QP

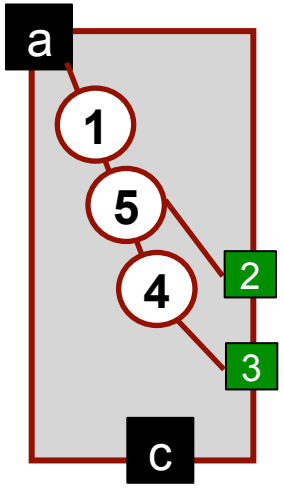


3. First partition
Sort on **X**:
Gate order 1 5 4 3 2
Pick: **1 5 4** on left

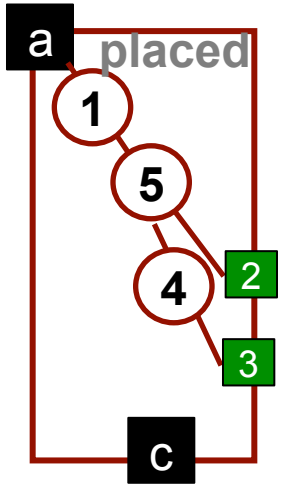
Small Partitioning Example



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

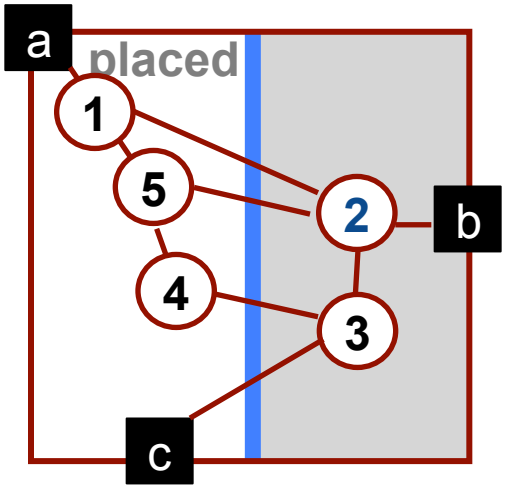


5. 2nd QP input
This is set up for this new smaller placement

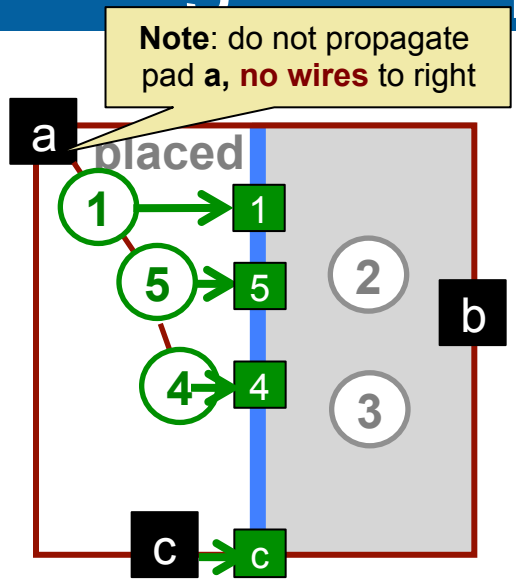


6. 2nd QP solved
New placement

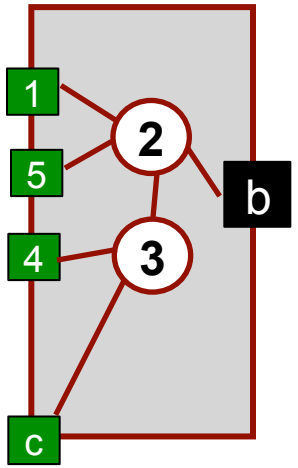
Small Partitioning Example



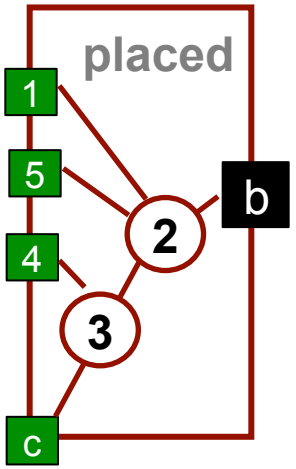
7. Left side placed.
Now, re-place right-side gates.



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

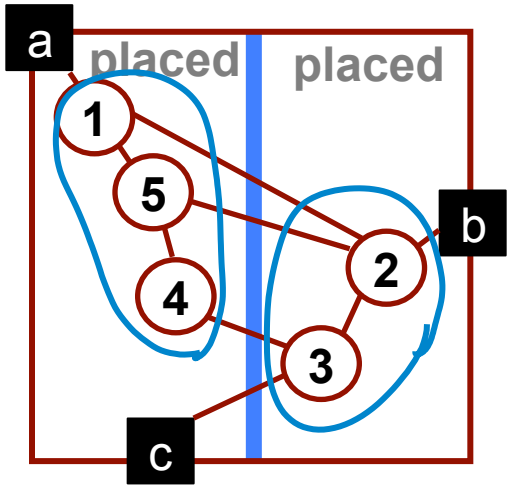


9. 3nd QP input
This is set up for this new smaller placement

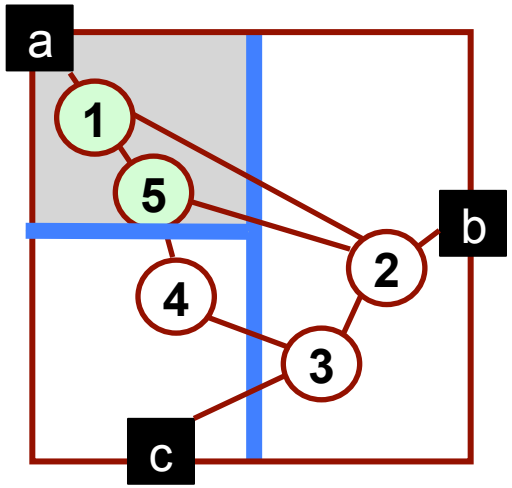


10. 3nd QP solve

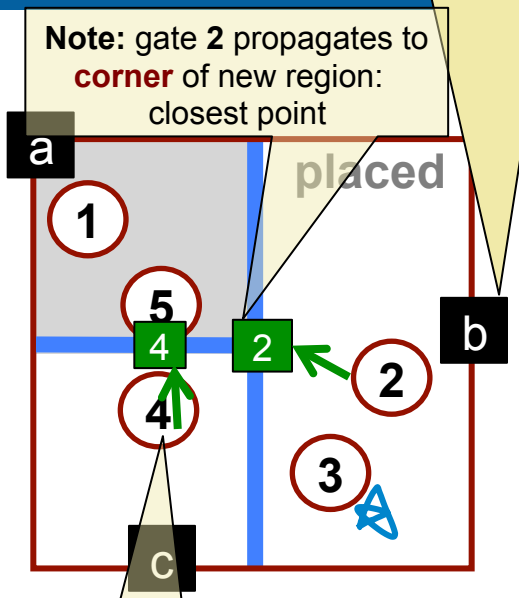
Small Partitioning Example



Repeat: Horizontal partition on left



Focus on top.
Sort gates on **Y**
Assign gates 1,5 to region.



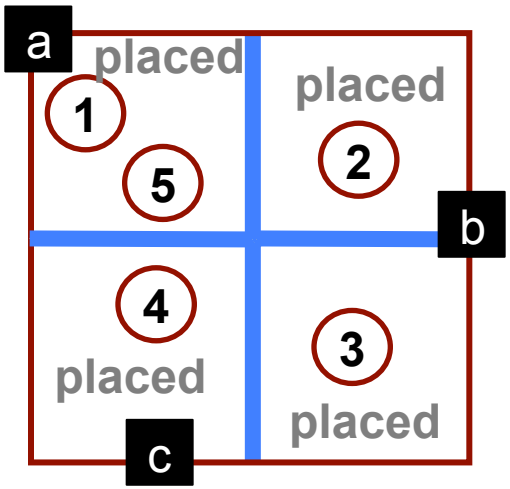
Propagate gates & pads

Note: do not propagate pad **b** or gate **3, no wires** to **1,5**

Note: gate **2** propagates to **corner** of new region: closest point

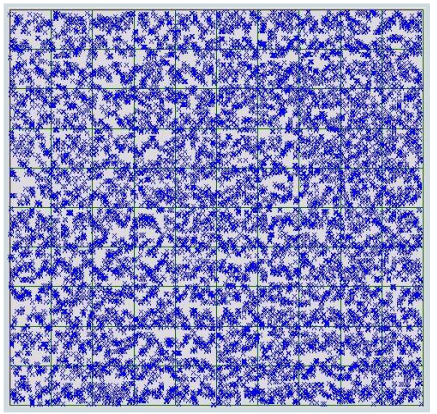
Note: gate **4** propagates up to **bottom** of new region

Keep Repeating this Recursion



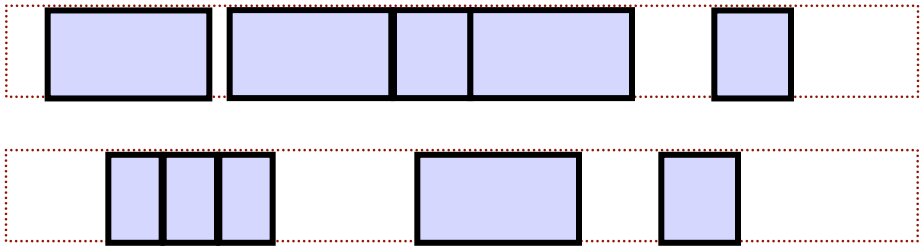
Continue…

- **Keep recursively partitioning…**
  - Usually, you continue until you have a "small" number of gates in each region
  - Small >> 1 typically. **10-100** for example
  - Get a good, "global" placement, but not a "final" placement



Final Placement Step: Legalization

- **Still need to force gates in precise rows for final result**
  - QP methods **cannot** force individual gates into standard cell rows, without overlaps



- **Solution step is called: Legalization**
  - Many different algorithms. One easy way to do this is by **annealing**!
  - Do short-distance, local improvement based on **swaps of nearby gates**
  - To anneal, set **T=HOT** to be very small (cold), so don't disrupt QP result

Placement Summary

- **Early placers based on iterative improvement**
  - **Simulated annealing** is a very good, famous example
  - Annealing stuff used widely in VLSI CAD – but not for placers. Too inefficient
- **Modern placers are all analytical**
  - Many different mathematical formulations, but all similar
  - Numerically optimize a mathematically friendly model of wirelength
    - Many modern placers can do some “legalization” at same time as wirelength!
  - **Quadratic placement** is a famous, important, practical example