ECE6703J

Computer-Aided Design of Integrated Circuits

Placement Basics

Outline

- ASIC Placement: Basics
- Wirelength Estimation
- Simple Iterative Improvement Placement

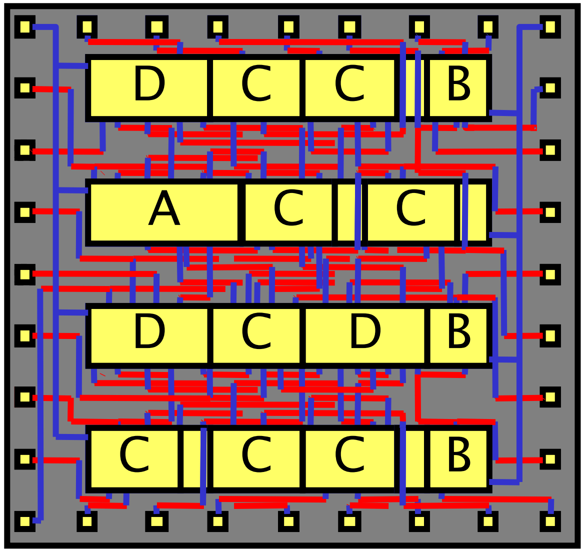
About Layout…

High-level design description → **Logic Synthesis** → Connected gates + wires in your technology. Called "netlist"

- Now we will look at how to turn the netlist into IC masks to build real chips.
  - This is called "**layout**" or "**physical design**".
  - The starting topic: the **placement** problem.

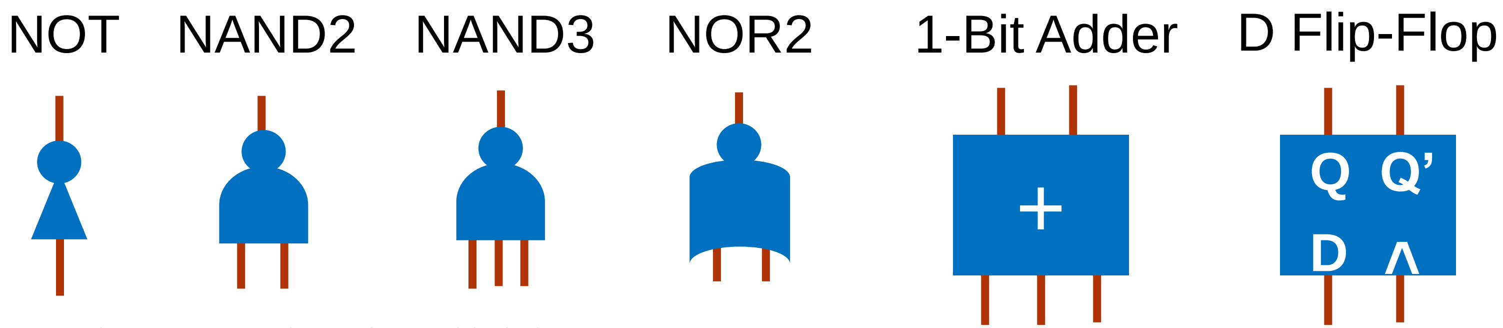
Placement for ASICs

- Focusing on the most common tasks in layout.
  - **Row-based standard cell layouts** for ASICs and SOCs.
- Logic synthesis gives a **netlist** of gates + wires.
- Our job is to **place** gates optimally in regions of the chip, and to **route the wires** to connect everything.



Standard Cell Library: Revisit

- This is a library of things called "**standard cells**".
  - A **standard cell** is a basic logic gate or small logic element, e.g., a NAND2 gate, a D flip-flop (DFF).
  - This is the set of **allowed logic elements** to build your chip.



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

How to think about a standard cell?

- Simple **abstraction** of a **geometric** "container" for the circuit to make the logic gate.
- Inside the cell: Complex device & mask & electrical issues.
- Outside the cell: **A box with pins**.

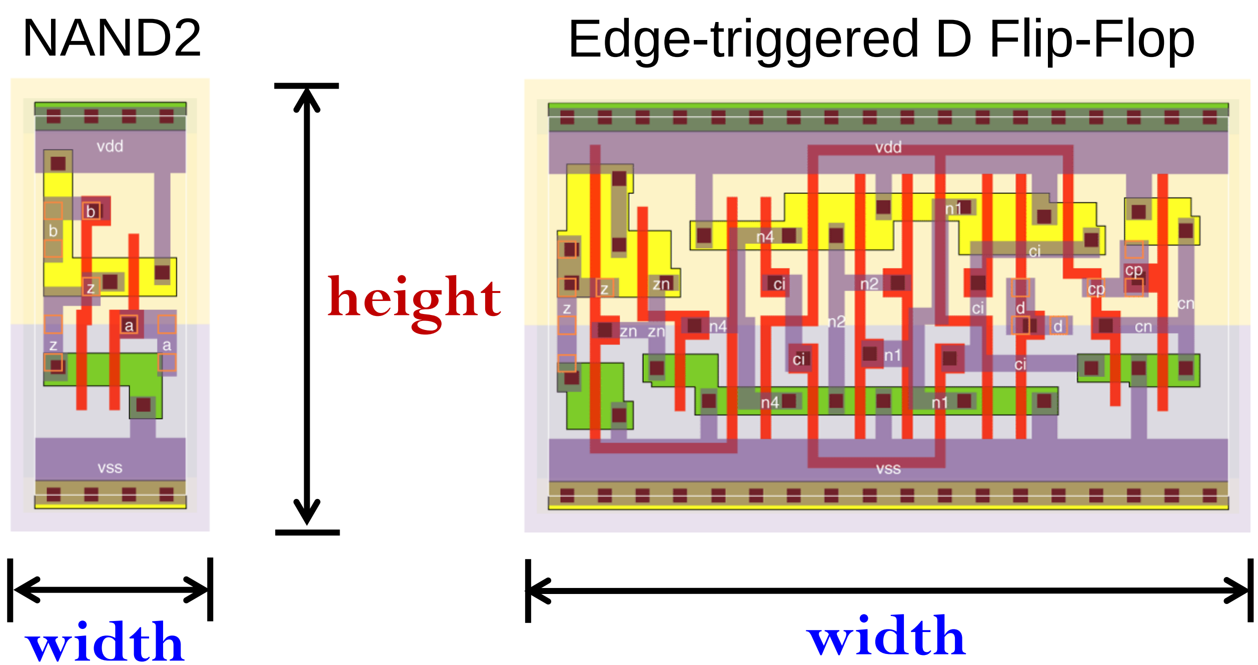
How Big is a Library--How Many Cells?

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

$$\left\{\begin{array}{l}\text{Logic}\\ \text{functions}\\ \text{(AND,}\\ \text{OR …)}\end{array}\right\} \times \left\{\begin{array}{l}\text{Input/}\\ \text{output}\\ \text{variants}\end{array}\right\} \times \left\{\begin{array}{l}\text{Timing}\\ \text{variants}\\ \text{(fast,}\\ \text{slow)}\end{array}\right\} \times \left\{\begin{array}{l}\text{Drive}\\ \text{strength}\\ \text{variants}\\ \text{(1X, 2X}\\ \text{…)}\end{array}\right\} \approx 1000 \text{ Cells}$$

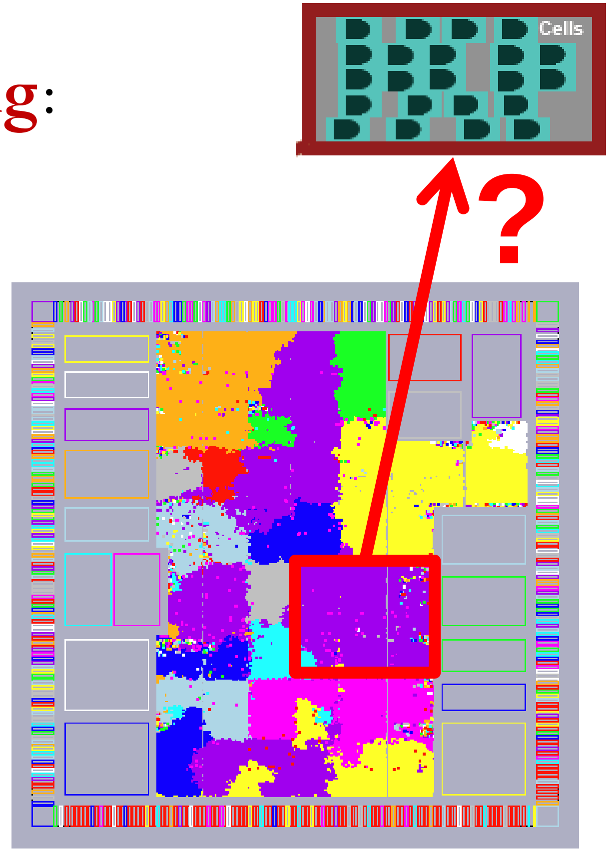
Some Real Standard Cells

- Example 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



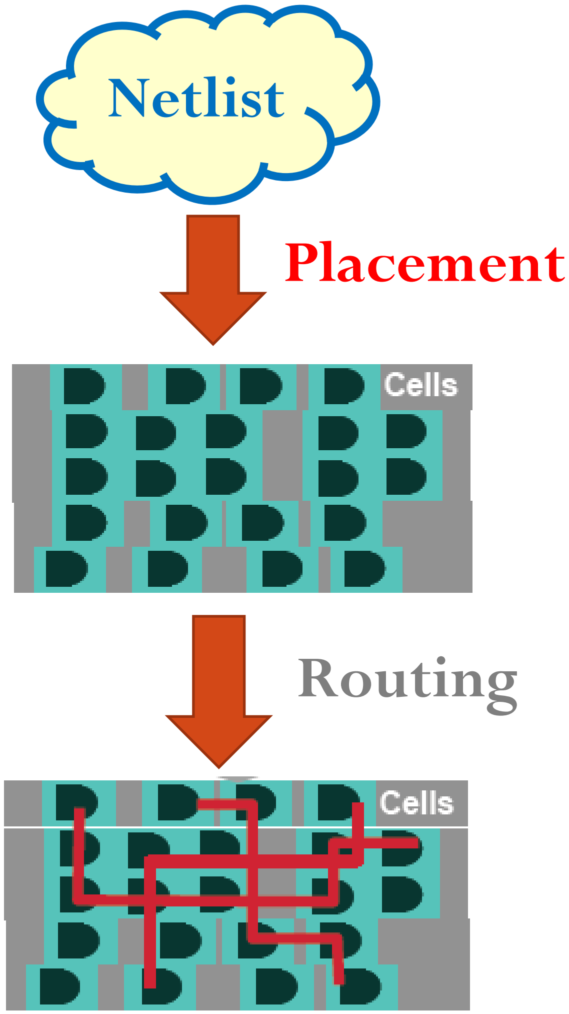
Realistic Context: Place One Block on SOC

- Big SOCs often designed **hierarchically**, composed from blocks which represent parts of the overall system.
- The first step is to do **floorplanning**:
  - How to place these "blocks"?
- Once floorplanning is done, we do **placement**:
  - Within each blocks, how to place gates that compose the block?



ASIC Placement Problem

- What does a **placer** do?
  - **Input**: Netlist of gates & wires
  - **Output**: **Exact location** of each gate
  - **Goal**:
  1. Able to **route** all wires.
  2. **Meet/optimize** timing requirements.
- Is this important? **Yes!**
  - Bad placement → Much more wire → bigger, slower chip
  - If placement is very bad, next tool in the flow — the **router** — is even unable to connect all wires, or meet timing.

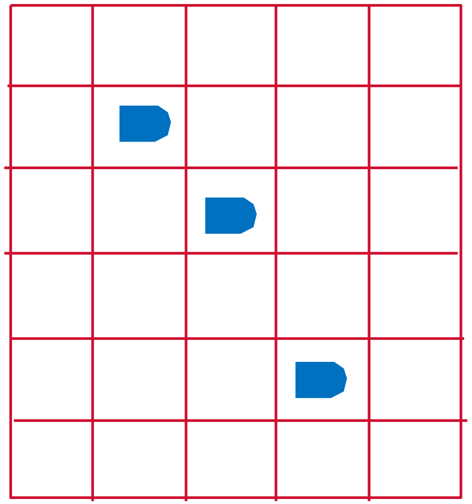


Outline

- ASIC Placement: Basics
- Wirelength Estimation
- Simple Iterative Improvement Placement

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**.

Target of Placer

- Placer optimizes the ability of router to connect all the nets.
  - However, routers are computationally expensive tools. We can't run a router "inside" placer.
  - How can we proceed? We need a simple **approximation**.

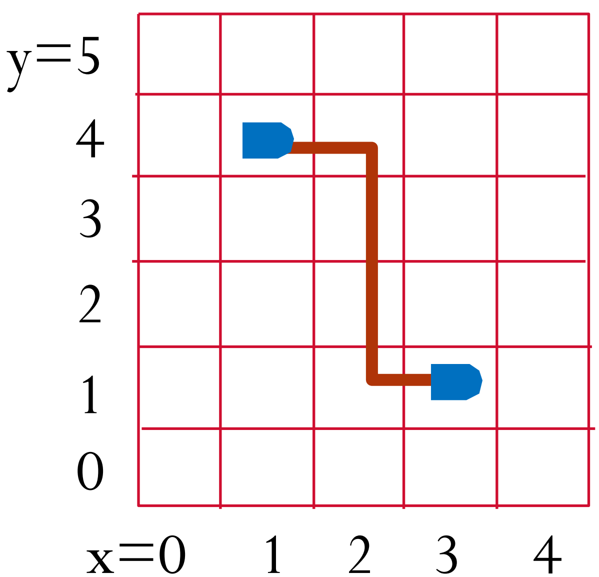
- What does a placer do?
  - Minimizes the **total expected wirelength**.
    - For each wire in the design, **estimate** the **expected length** of the routed wire.
    - Then, **minimize** the objective: $\Sigma_{\text{wire } W_i}$ EstimatedLength(Wi)

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

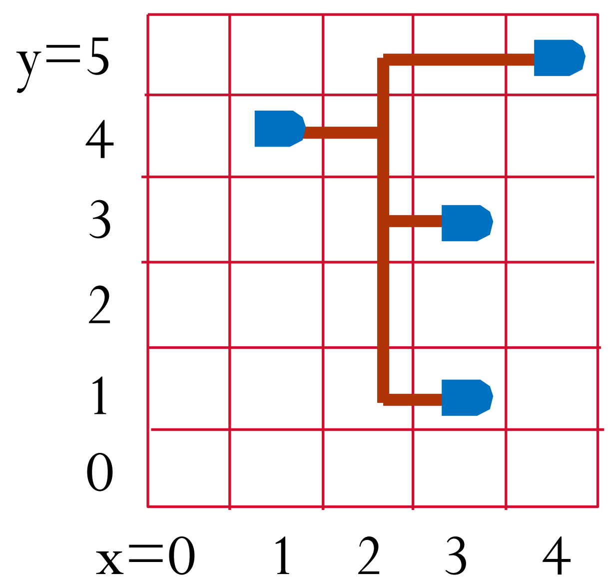
Terminology

- A wire in a layout is "a **net**".
- The whole set of gates+wires is "the **netlist**".
- A "***k*-point net**" is a net that connects k objects.

A "**2-point** net"



A "**4-point** net"

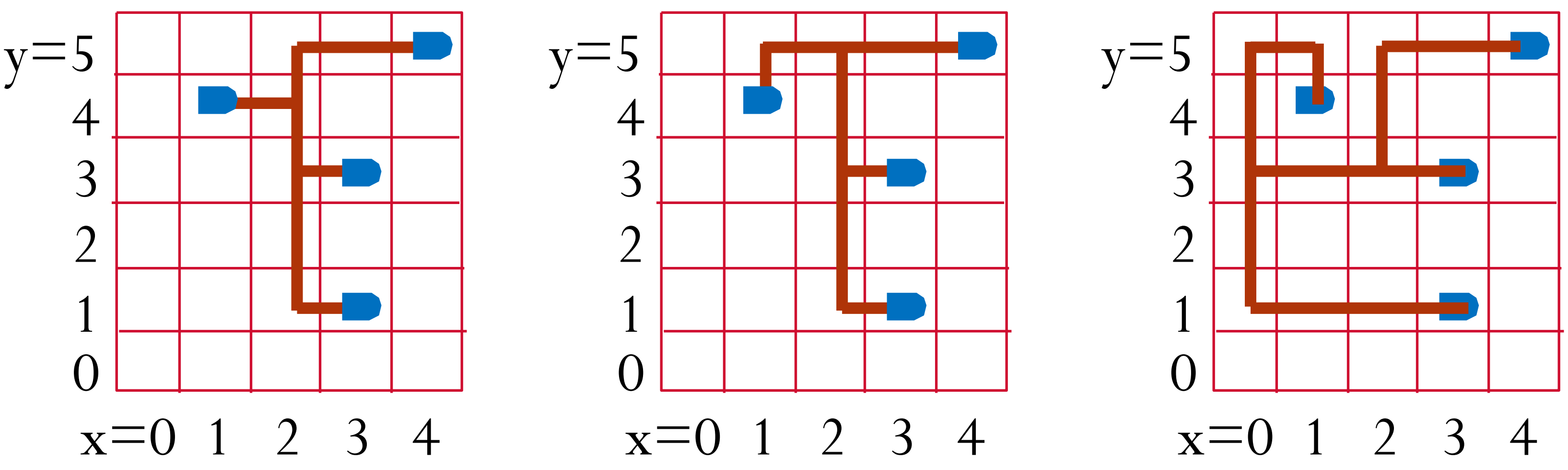


Why >2 points?

Logic fanout

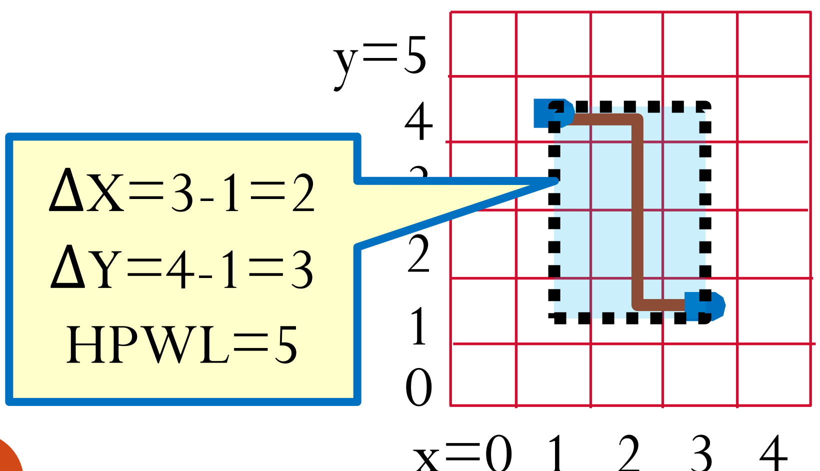
Wirelength Estimation

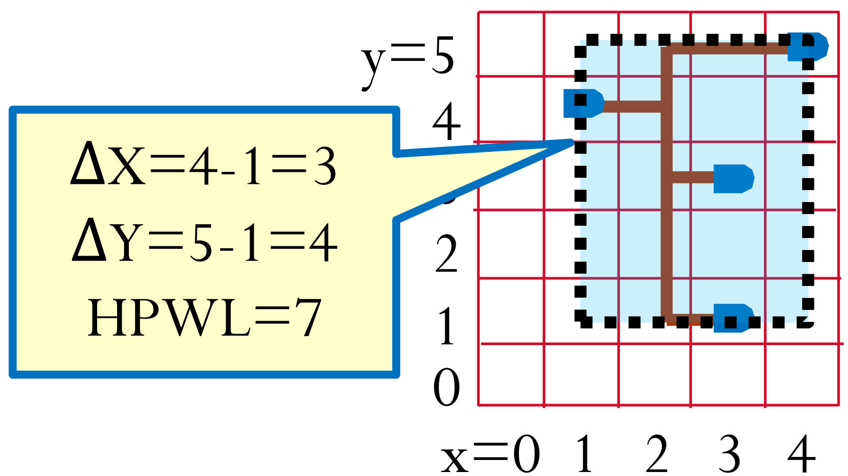
- Many different estimators, adapted to different placer methods.
- Wirelength estimation is hard. Why?
  - Multi-point nets can be routed in many **different paths**.
  - In a dense layout, nets do **not** all get routed in a "shortest path." **Hard to predict**!



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.





About HPWL Estimator

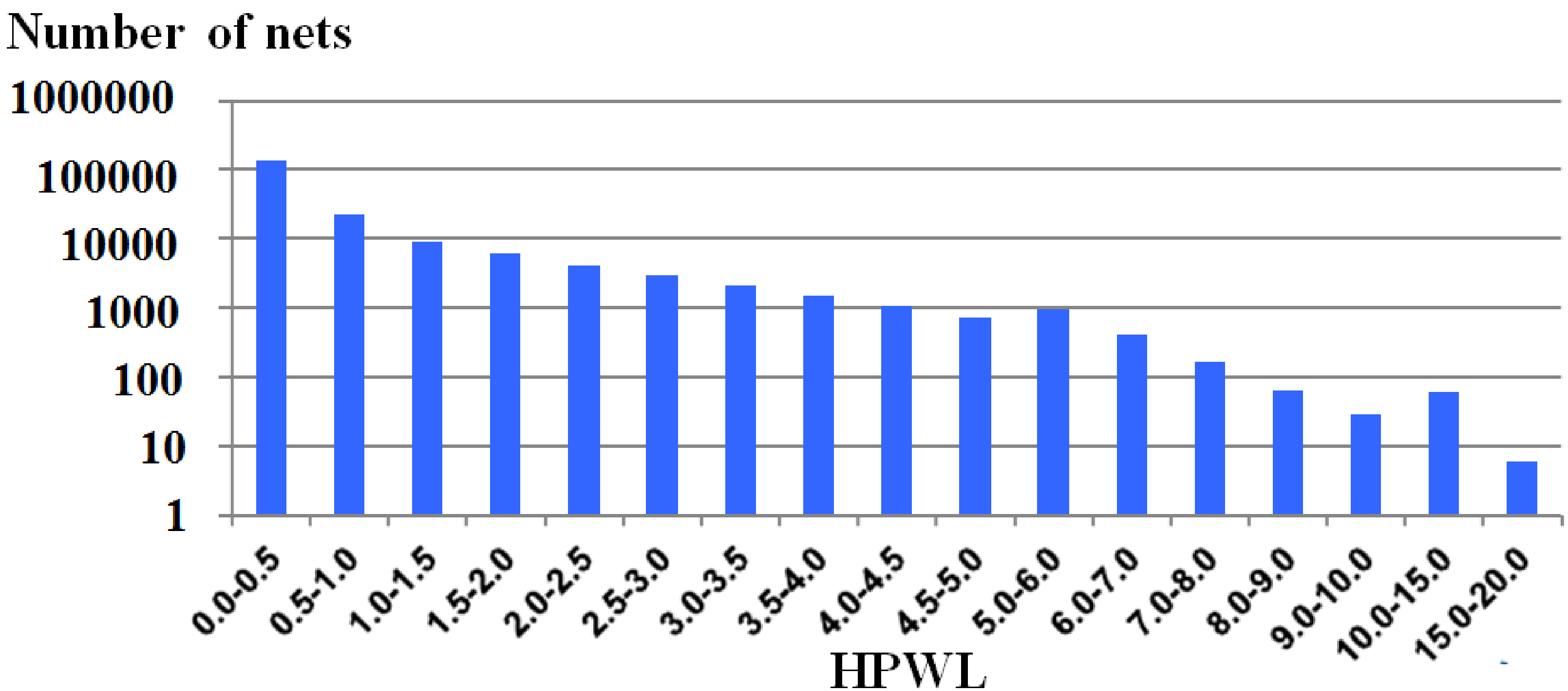
- **Easy** to calculate, even for a multi-point net
- General formula:

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
  - **Fact**: all wiring on big chips is **strictly horizontal & vertical** – no "arbitrary angles" for manufacturing reasons.
  - **Result**: No matter how complex the final routed wire path is, you need **at least** this much wire to connect everything.

HPWL Wirelength Estimation

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



Outline

- ASIC Placement: Basics
- Wirelength Estimation
- Simple Iterative Improvement Placement

A Very Simple Placer

- Two big ideas

1. Start with a **random** placement
   - Just **randomly assign** each gate to a grid location (only 1 gate per grid!)
   - Result: a **terrible** placement.
     Really big $L = \sum_{\text{nets } Ni} \text{HPWL}(Ni)$.

2. **Random iterative improvement**

Random Iterative Improvement

- **Random iterative improvement**
  - Pick a **random pair** of gates in the grid. **Exchange** their locations (called a "**swap**").
  - Evaluate the **change** in $L = \sum_{\text{nets } Ni} \text{HPWL}(Ni)$, i.e., compute $\Delta L = [new\_HPWL - old\_HPWL]$.
  - If $\Delta L < 0$, new L got smaller: Good! **Keep** this swap.
  - If $\Delta L > 0$, new L got bigger: Bad! **Undo** this swap.
  - Keep doing this for many, many random swaps, until L stops improving.

Algorithm

```
//random initial placement
foreach( gate Gi in netlist )
   place Gi in random location (x,y) in grid not already occupied;
//calculate initial HPWL for whole netlist
L=0;
foreach( net Ni in the netlist ) L = L + HPWL(Ni);
//main improvement loop
while ( total HPWL L is improving ) {
   pick random gate Gi;  pick random gate Gk; swap gates Gi and Gk;
   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 Gk;
}
```

Computing ΔL Efficiently: Incrementally

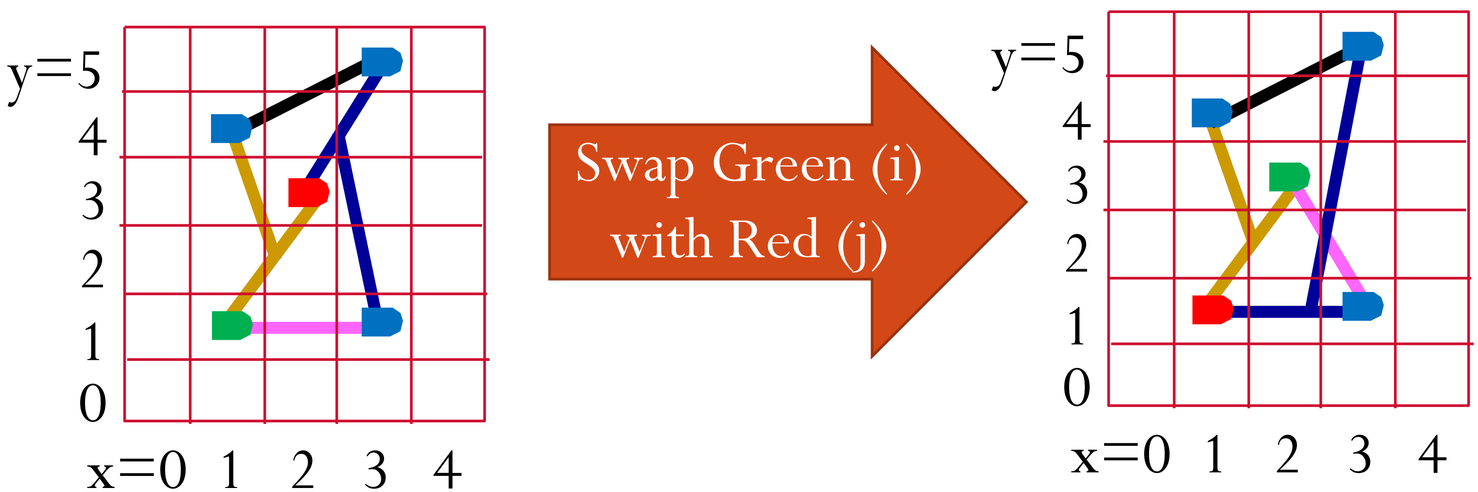
- **Incremental** wirelength update calculation.
  - Cannot afford to re-compute length of each net in entire placement after each swap!
  - Most nets did **not** change! Do this incrementally--just look at nets that **could** change.

Computing ΔL Efficiently: Incrementally

- Suppose we swap gate i with j. For each net N, it falls in the following 4 cases:

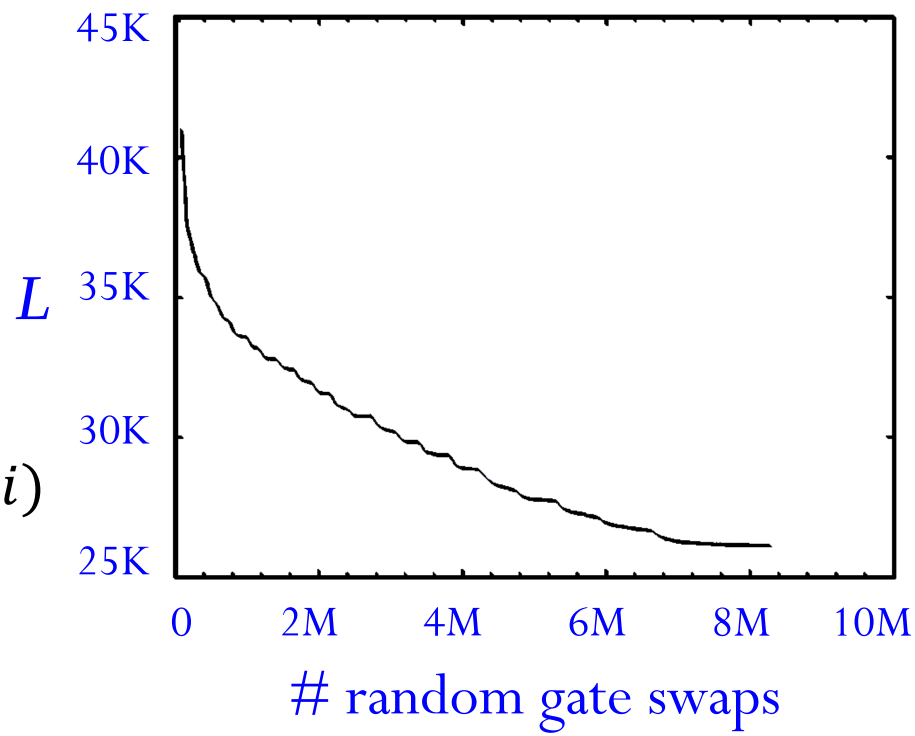
Could HPWL(N) change?

1. $i \notin N$ and $j \notin N$. **No!**
2. $i \in N$ and $j \notin N$. **Yes!**
3. $i \notin N$ and $j \in N$. **Yes!**
4. $i \in N$ and $j \in N$. **No!**



How Does This Work...?

- It works ok….
  - Small benchmark
  - ~2500 gates + ~500 pins
  - Placed in a 50x50 grid
- Graph shows "progress"
  - Y axis: $L = \sum_{\text{nets } Ni} \text{HPWL}(Ni)$
  - X axis: # swaps
  - 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 **<u>stops</u>** getting better.
  - Can optimize what we care about: estimated total wirelength $L = \sum_{\text{nets } Ni} \text{HPWL}(Ni)$.
  - It will improve. Sometimes, a lot.

- But… final result is still not very good.
  - We can do better. A lot better…