

Paper Review due Feb 3:

“SPRoute: A Scalable Parallel Negotiation-based Global Router”

Jiayuan He - Univ. of Texas at Austin

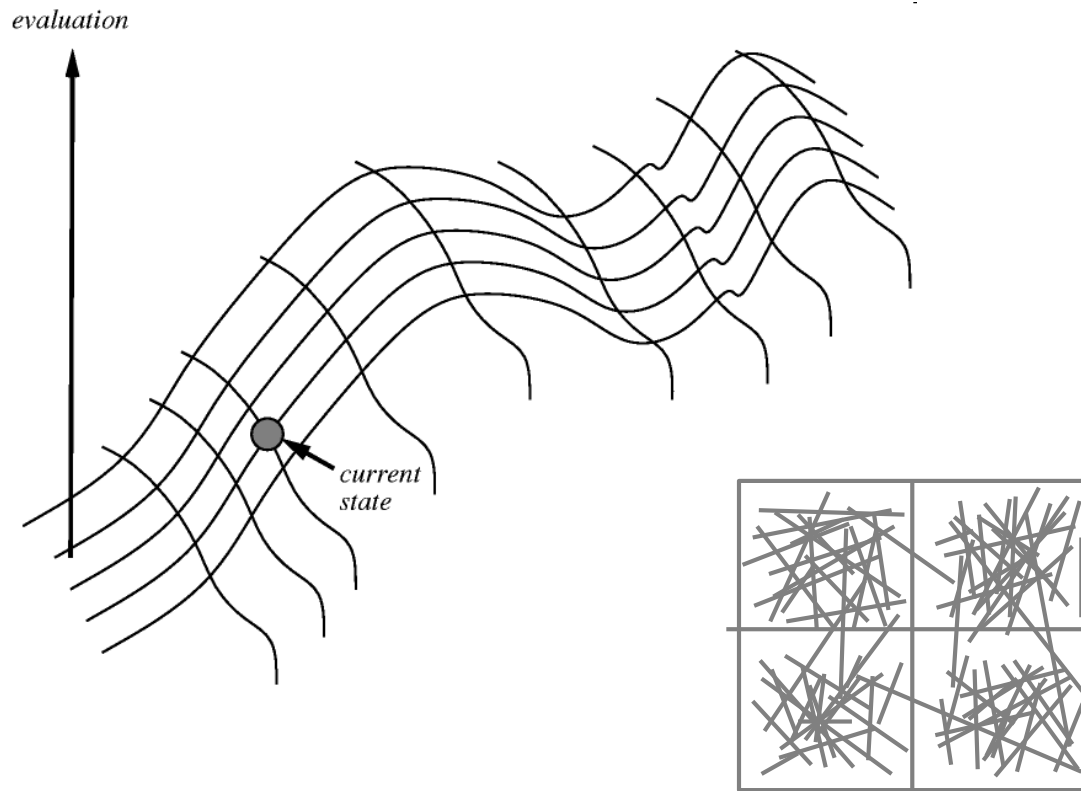
Martin Burtscher - Texas A&M Univ.

Rajit Manohar - Yale Univ.

Keshav Pingali - Univ. of Texas at Austin

2019 International Conf. on Computer Aided Design, Nov. 2019

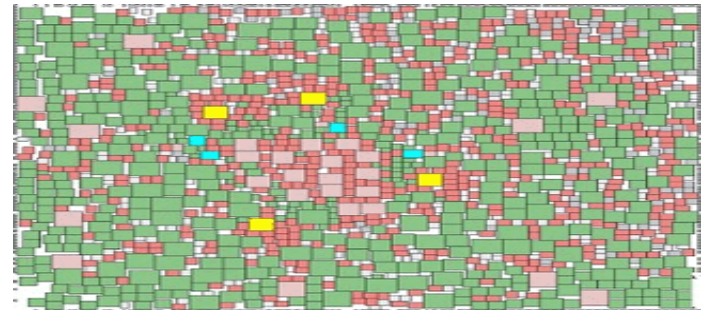
<https://userweb.cs.txstate.edu/~mb92/papers/iccad19.pdf>



Placement

CPEN 513

2019/2020 Term 2



Some slides based on those from Lei He (UCLA) and D. Pan (UTexas)

Easy to read paper: Kirkpatrick

More comprehensive paper: Cong05

- on the course web site
- no paper review required!

Simulated Annealing Based Methods

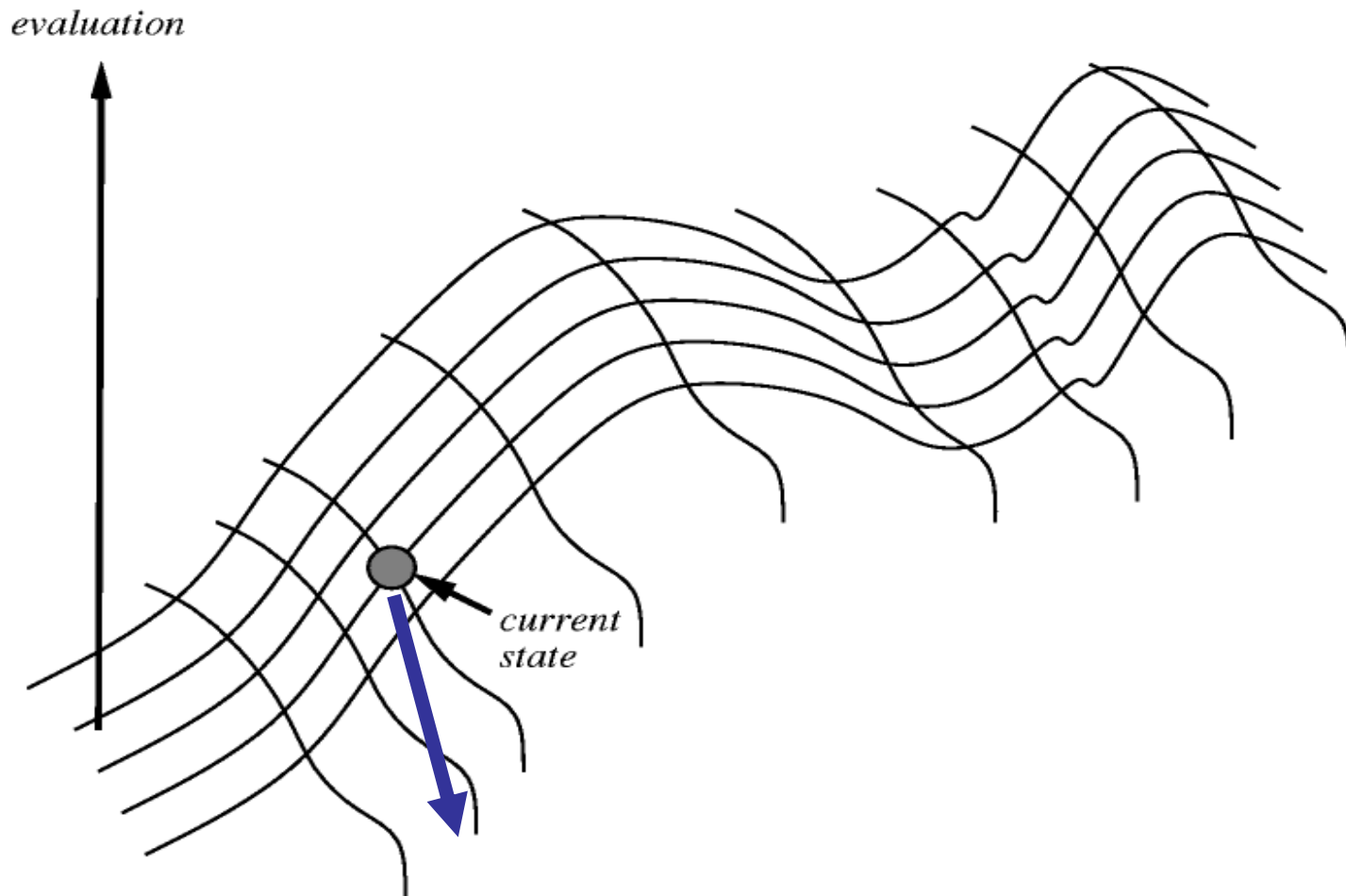
We talked about this earlier

Details in:

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi,
“Optimization by simulated annealing,” *Science*, vol. 220,
no. 4598, pp. 671–680, 1983.

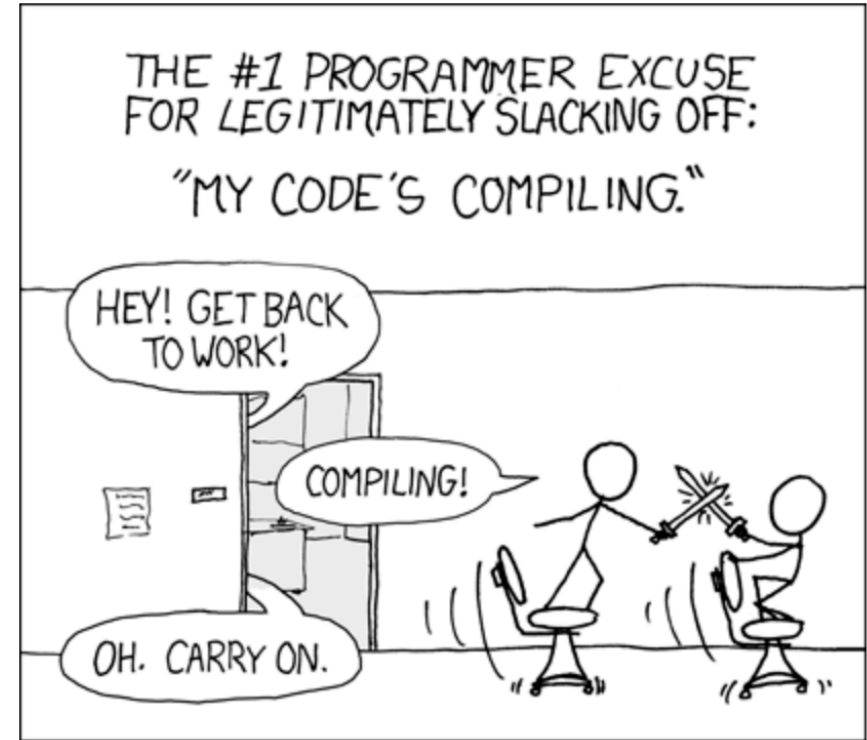
(no paper review required)

Aside: Why not gradient descent?



Compile times:

Hardware Compilation:
A "Go Home" Event



Software
Compilation: Time
to Goof Off

Fast and/or Scalable Placement:

Any suggestions?

J. Coole, G. Stitt, “BPR: Fast FPGA Placement and Routing using Macroblocks”, CODES+ISSS’12.

Y. Sankar, J. Rose, “Trading Quality for Compile Time: Ultra-Fast Placement for FPGAs”

J. Goeders, G. Lemieux, S. Wilton, “Deterministic Timing-Driven Parallel Placement by Simulated Annealing using Half-Box Window Decomposition”, Reconfig 2011.

M. An, G. Steffen and V. Betz, "Speeding Up FPGA Placement: Parallel Algorithms and Methods," FCCM, 2014.

Previous
EECE 583
Project!

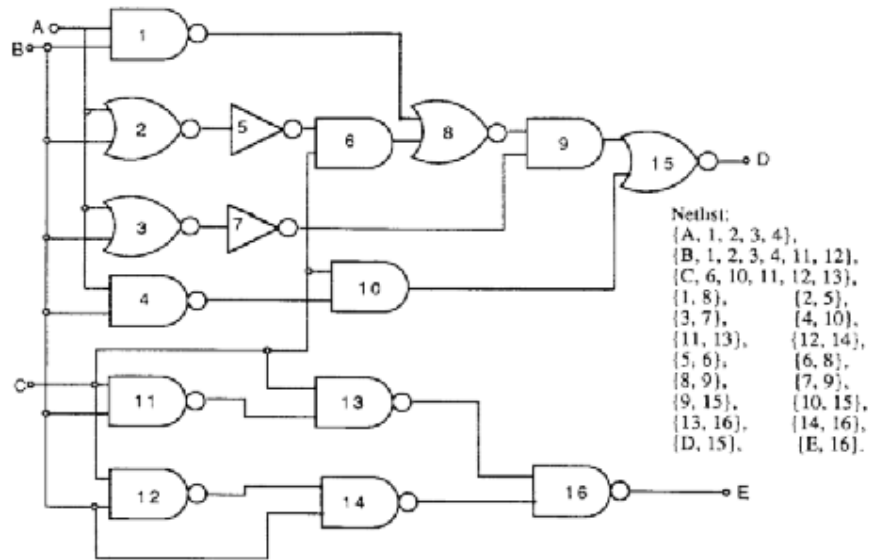
Partitioning-based Approach

Try to group closely connected modules together.

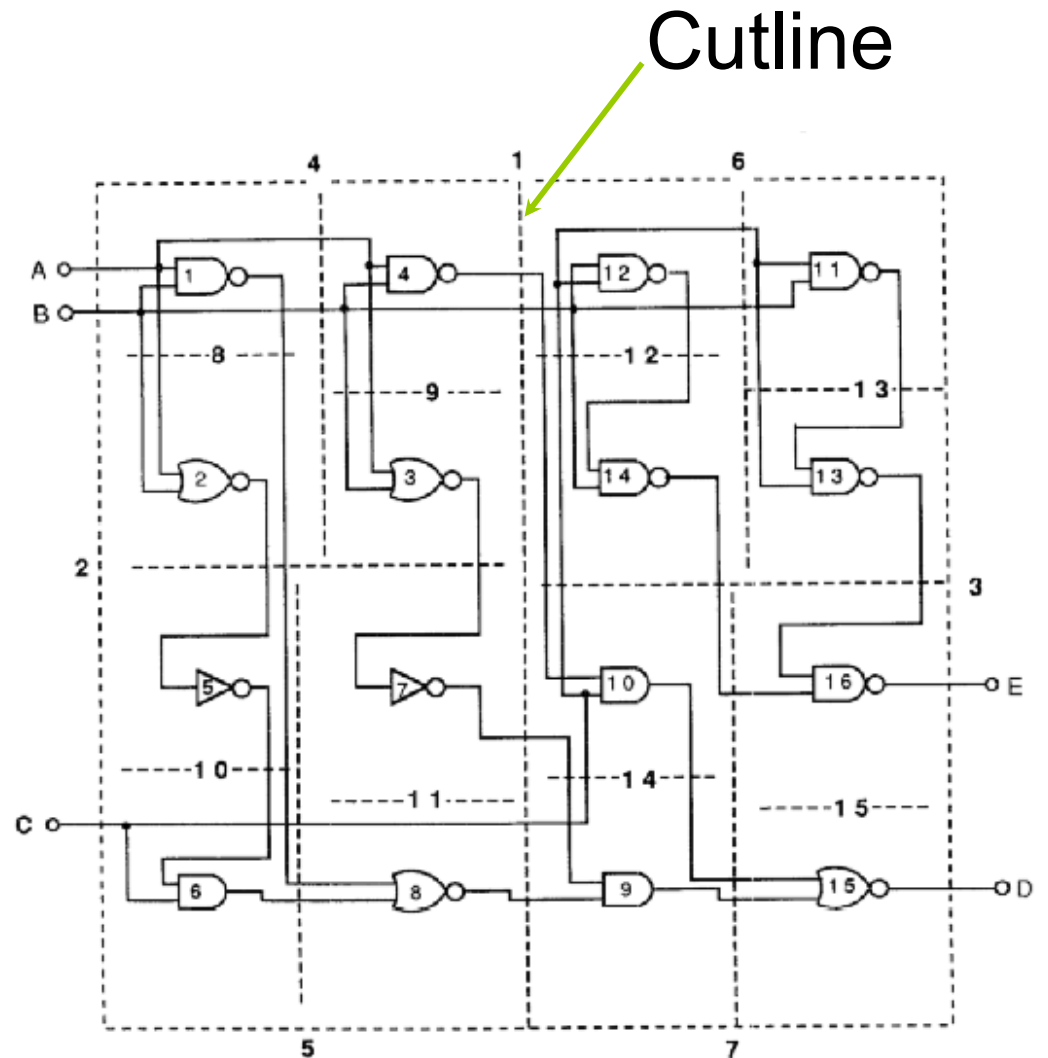
- Repeatedly divide a circuit into subcircuits such that the cut value is minimized.
- Also, the placement region is partitioned (by cutlines) accordingly.
- Each subcircuit is assigned to one partition of the placement region.

Note: Also called min-cut placement approach.

An Example



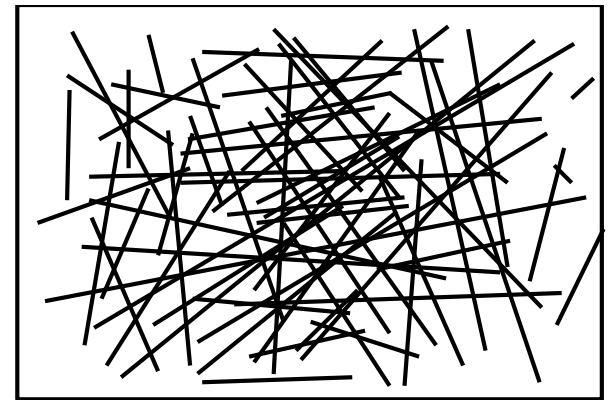
Circuit



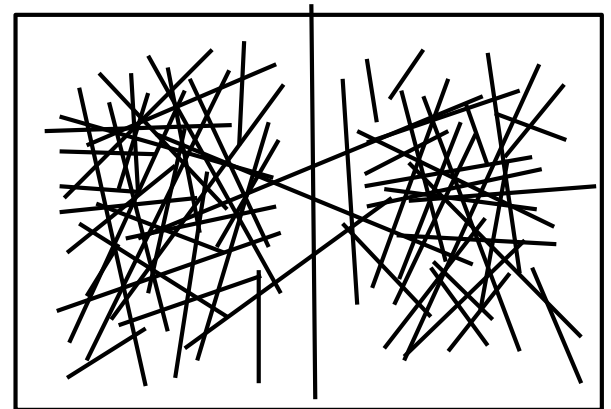
Placement

FM Partitioning:

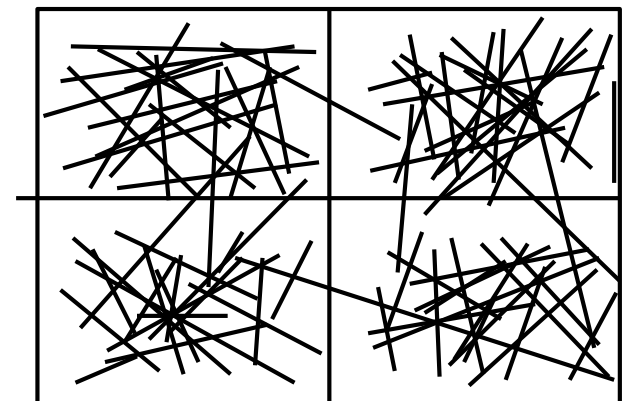
```
list_of_sets = entire_chip;  
while(any_set_has_2_or_more_objects(list_of_sets))  
{  
    for_each_set_in(list_of_sets)  
    {  
        partition_it();  
    }  
    /* each time through this loop the number of */  
    /* sets in the list doubles. */  
}
```



Initial Random Placement



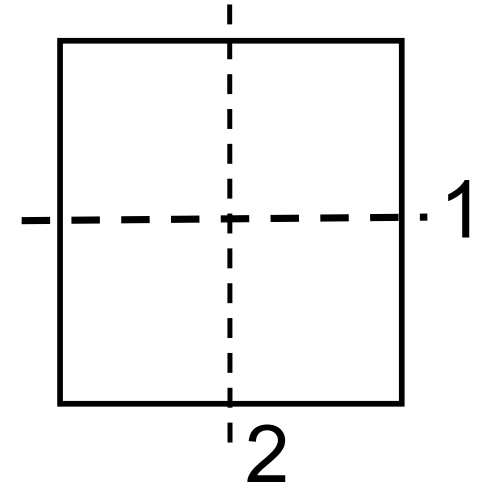
After Cut 1



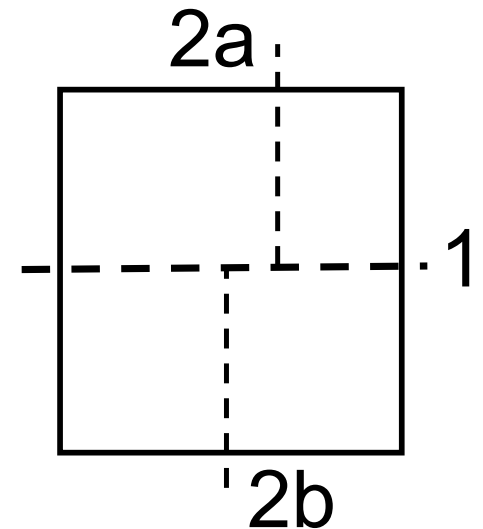
After Cut 2

Two Cutline Styles

Cut Oriented Min-Cut Placement :
Use the same cutlines for all sub-regions.

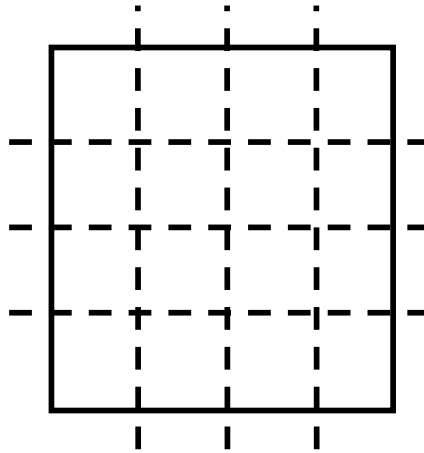


Block Oriented Min-Cut Placement:
Different sub-regions have separate cutlines.
More flexible.



3 Cutline Selection Schemes

Suppose we want to partition as follows:

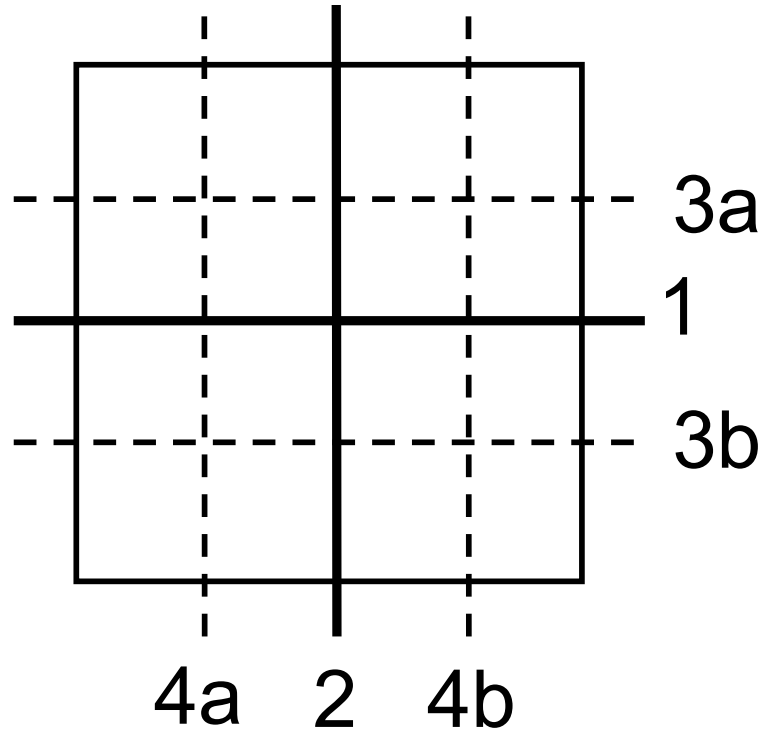


Question: Which cutline to use first?

3 Cutline Selection Schemes:

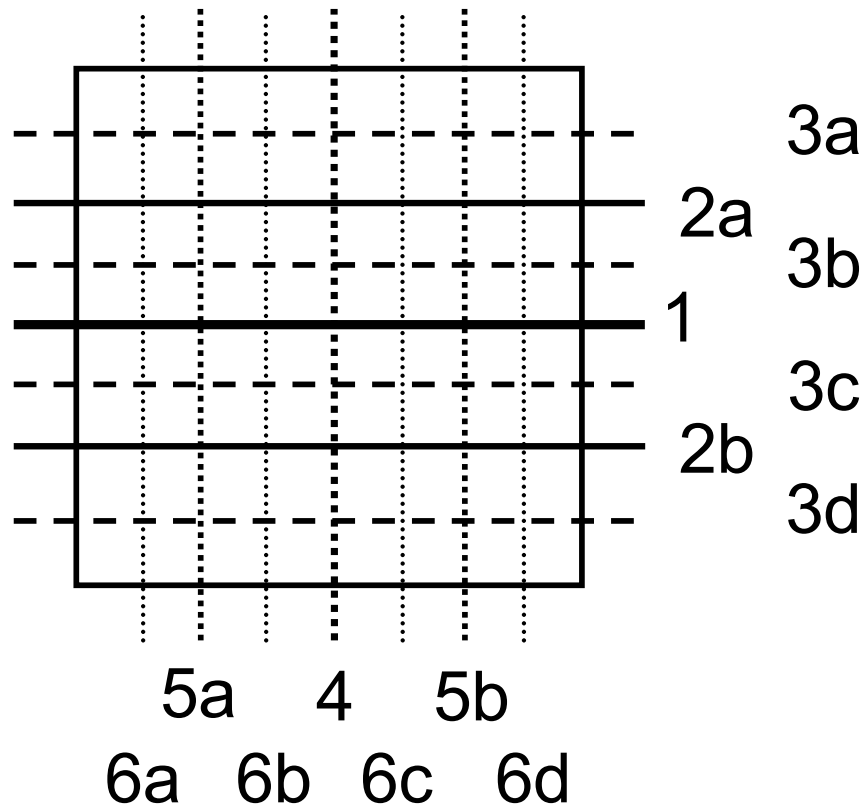
- Quadrature Placement Procedure.
- Bisection Placement Procedure.
- Slice/Bisection.

Quadrature Placement Procedure



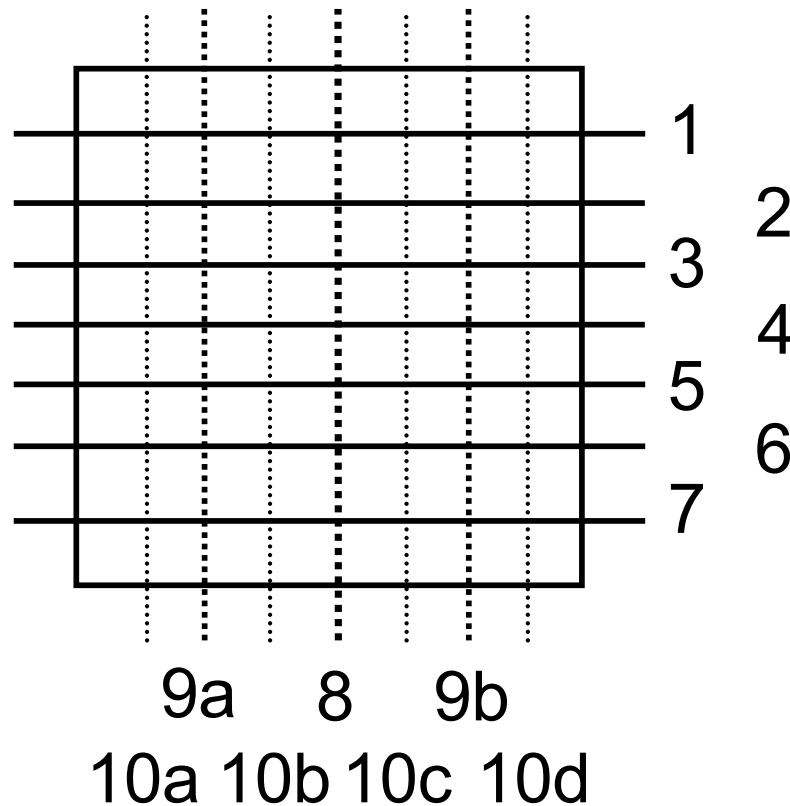
Very suitable for circuits with high routing density in the centre.

Bisection Placement Procedure



Good for standard-cell placement.

Slice/Bisection Procedure



Most suitable when there is a high interconnect density at the periphery.

Variations

There are many variations in the partitioning-based approach. They are different in:

- The objective function used.
- The partitioning algorithm used.
- The selection of cutlines.

Problems with these methods: irreversible decisions are made early in the placement process, when little information about the final placement is available.

Analytical Placement

Write down the placement problem as an analytical mathematical problem

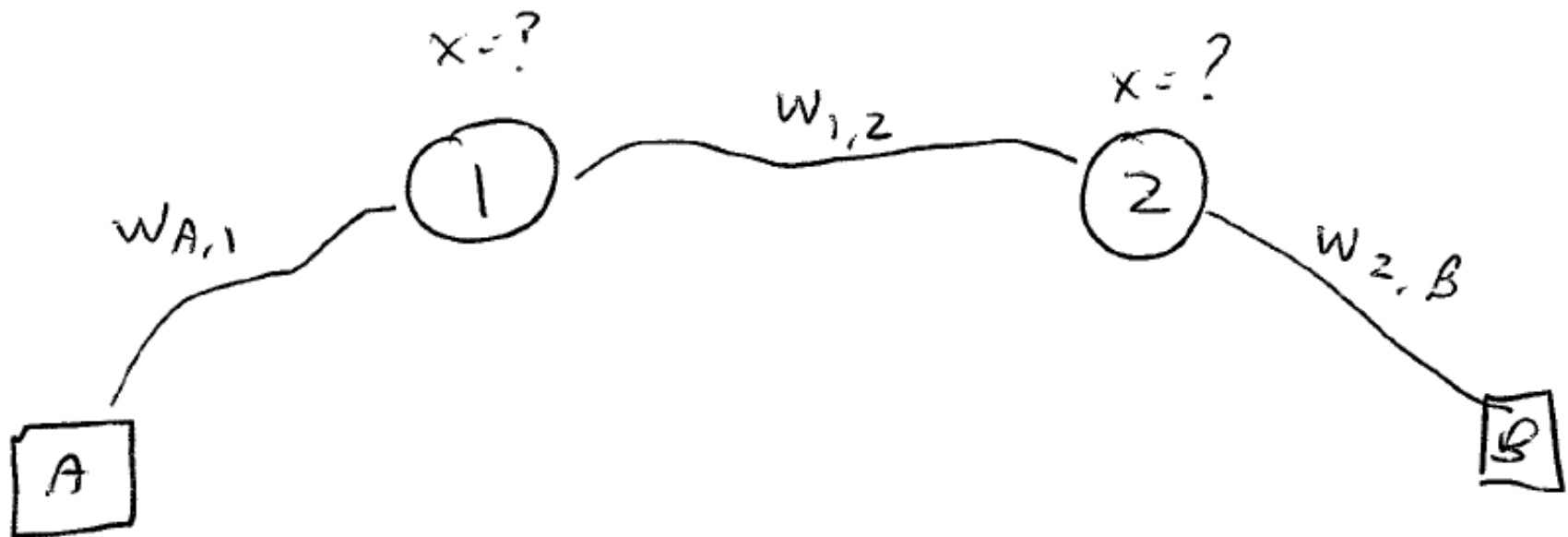
Solve the placement problem **directly**

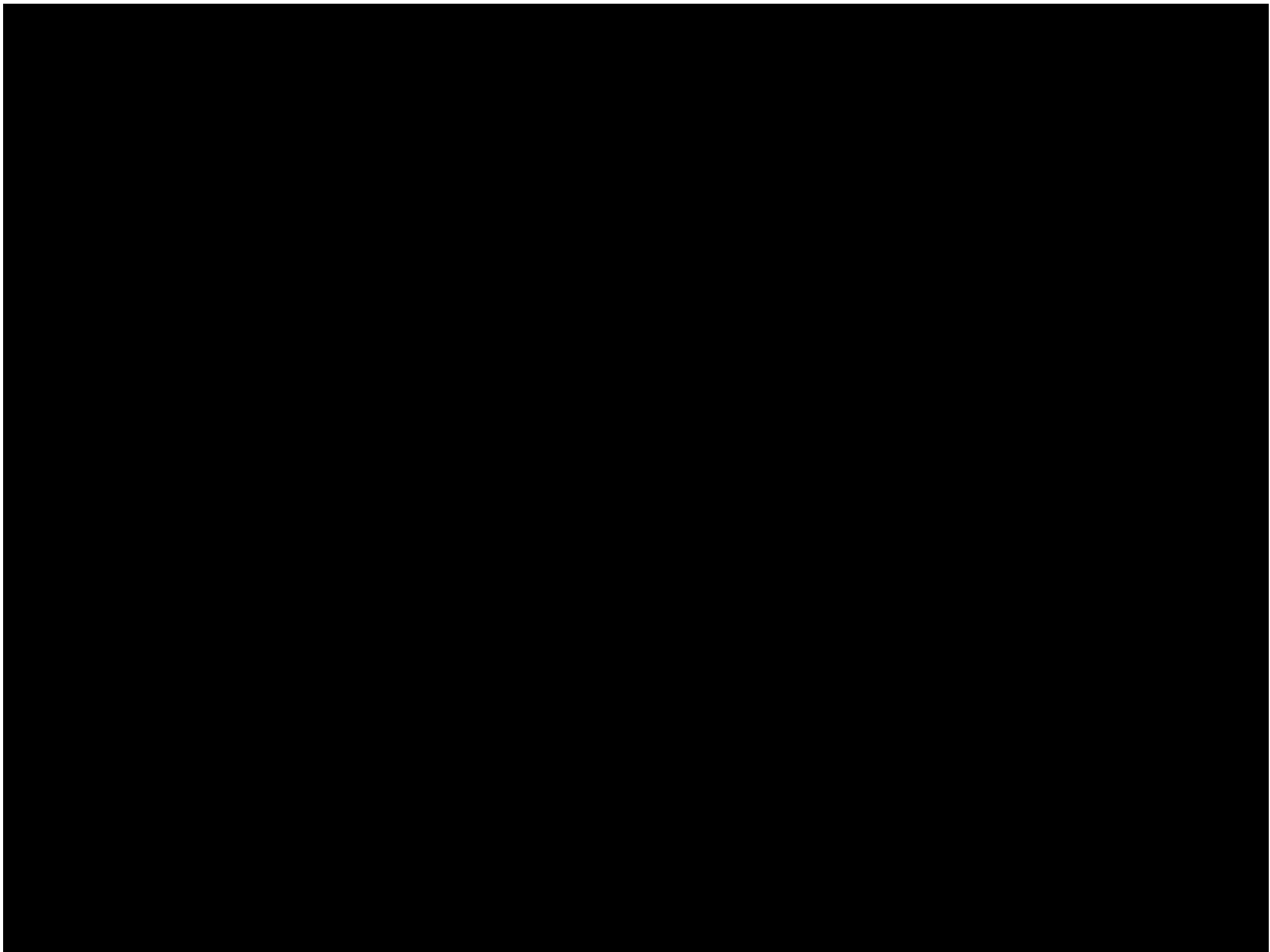
Example:

- Sum of **squared wire length** is quadratic in the cell coordinates.
- So the wirelength minimization problem can be formulated as a quadratic program.
- It can be proved that the **quadratic program** is convex, hence polynomial time solvable

Problem: This gives overlapping cells: Key is to remove overlaps.

Toy Example: ... on the board





The Gordian Knot (from Wikipedia)

The ancient capital of Phrygia found themselves w/o a king

- The next man to enter the city with an ox-cart will be king
- Gordias, a poor peasant came into town, and so became king

Gordias was so happy, he dedicated his ox-cart to the God Sabazios. He tied the ox-cart to a shaft with an intricate knot. Whoever could untie the knot would be the king of Asia.

In 333, Alexander the Great tried to untie the knot

- Couldn't find an end to the rope

So he sliced it in half with a stroke of his sword



Jean-Simon Berthélemy (1743 - 1811): Alexander durchschlägt den gordischen Knoten

Gordian:

Global Optimization and Rectangle Dissection

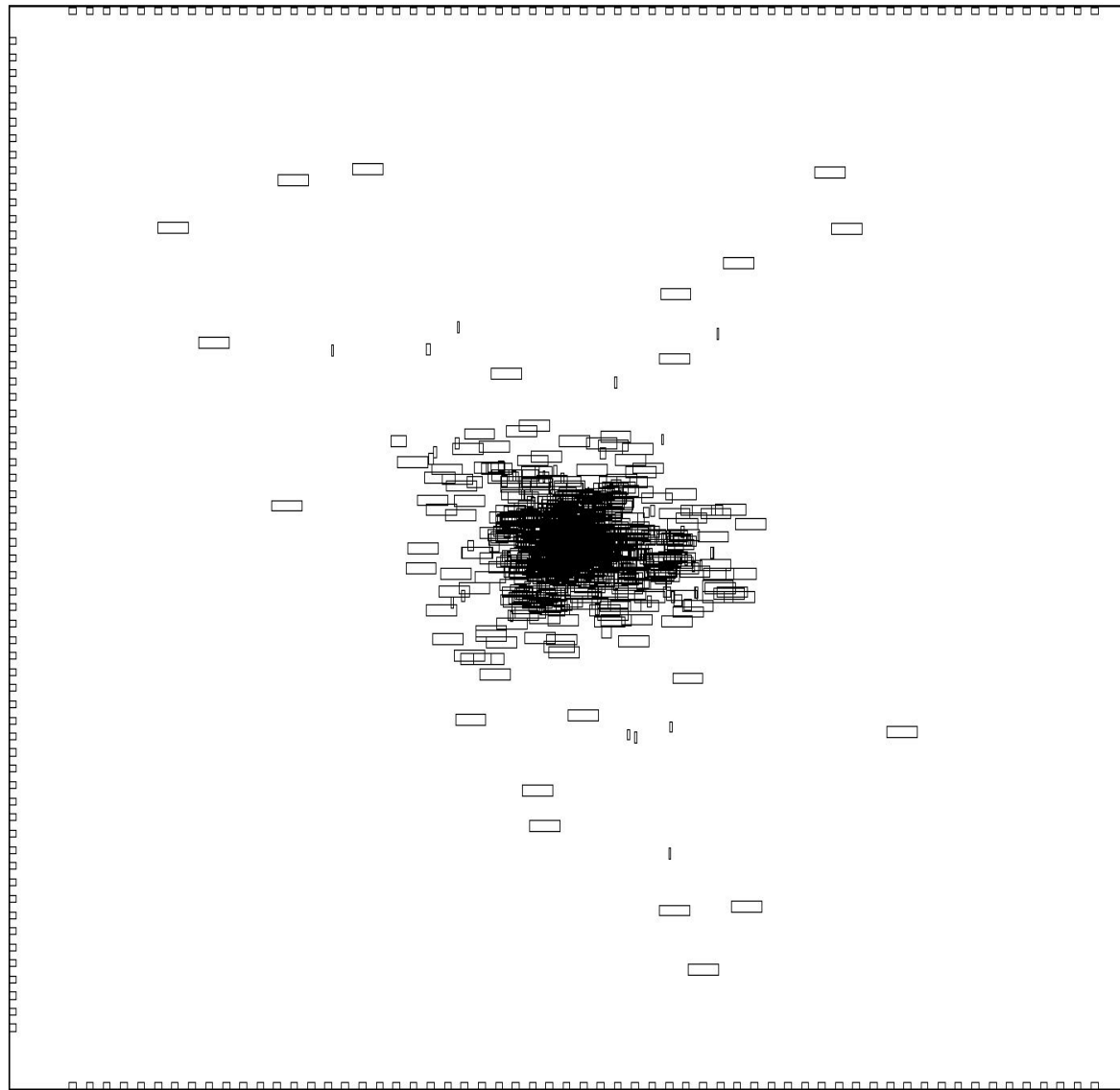
Use Global Optimization at all steps

Start with placing all modules inside the chip area

Alternate between placement and partitioning

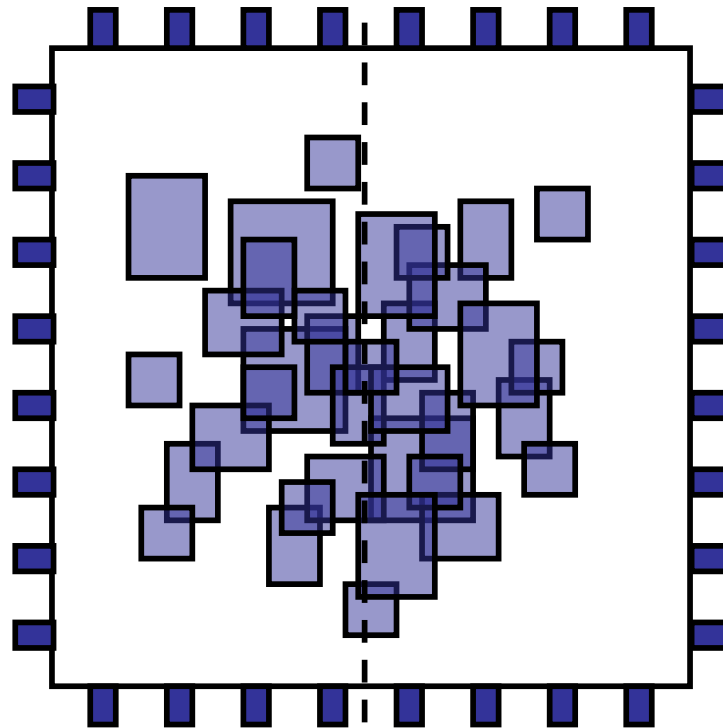
When groups have been broken up into groups of less than k modules, place modules within their sliced area

Solution of the Original QP



Partitioning

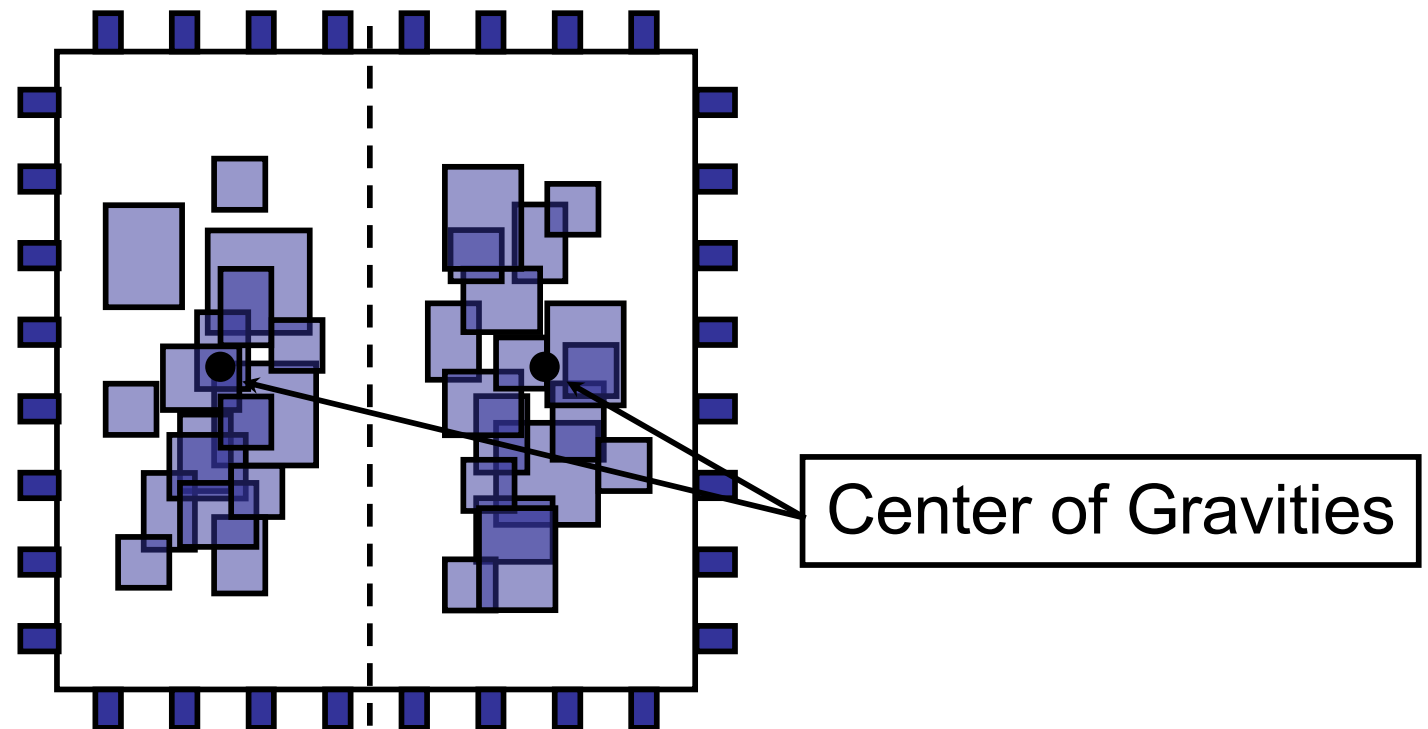
Find a good cut direction and position.



Improve the cut value using partitioning (to be discussed next week)

Applying the Idea Recursively

Before every level of partitioning, do the Global Optimization again with additional constraints that the center of gravities should be in the center of regions.

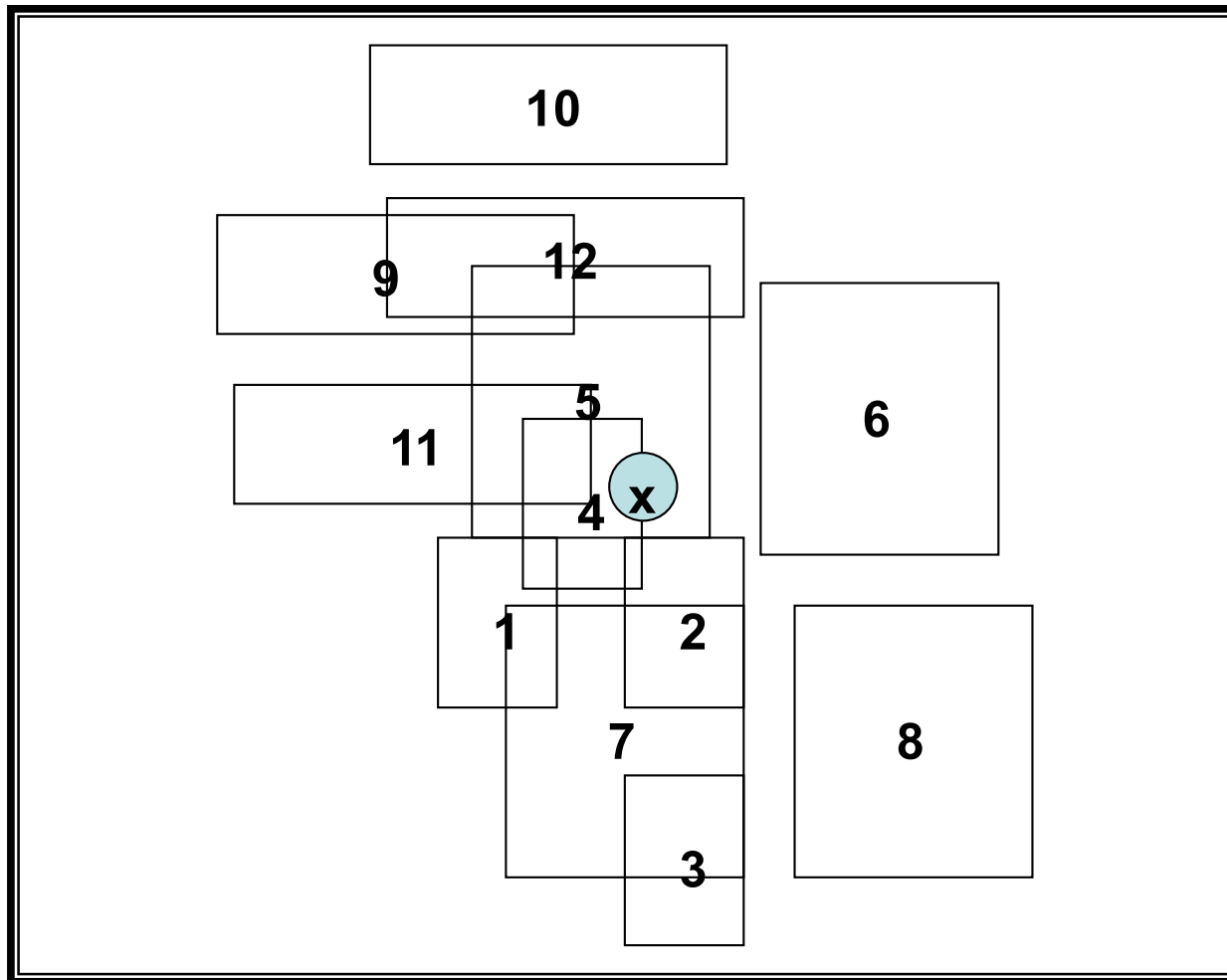
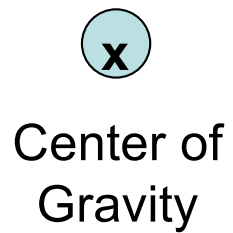


Always solve a single QP (i.e., global).

Example Placement

K = 3


Chip Area

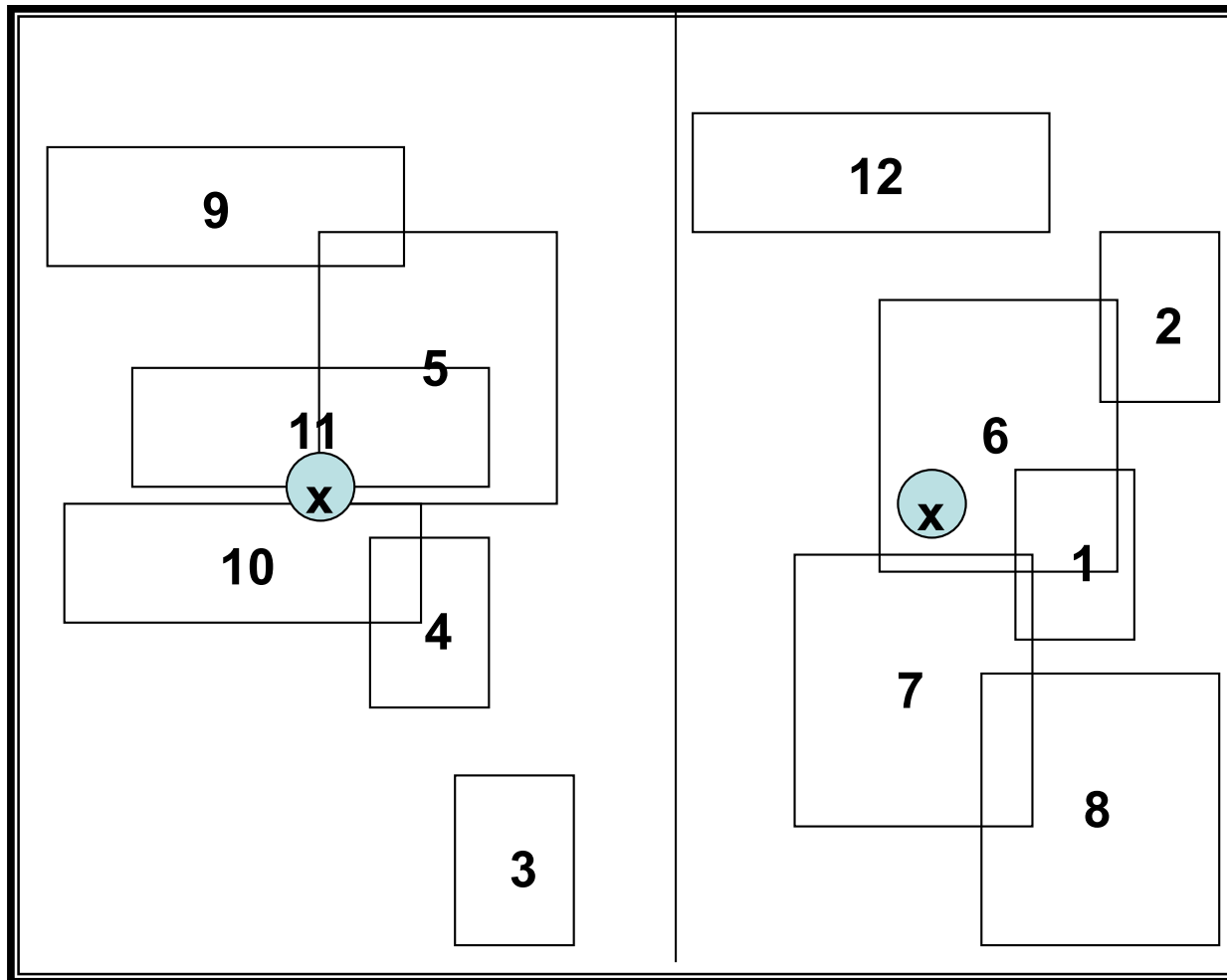


Example Placement

$K = 3$

Chip Area



Center of
Gravity

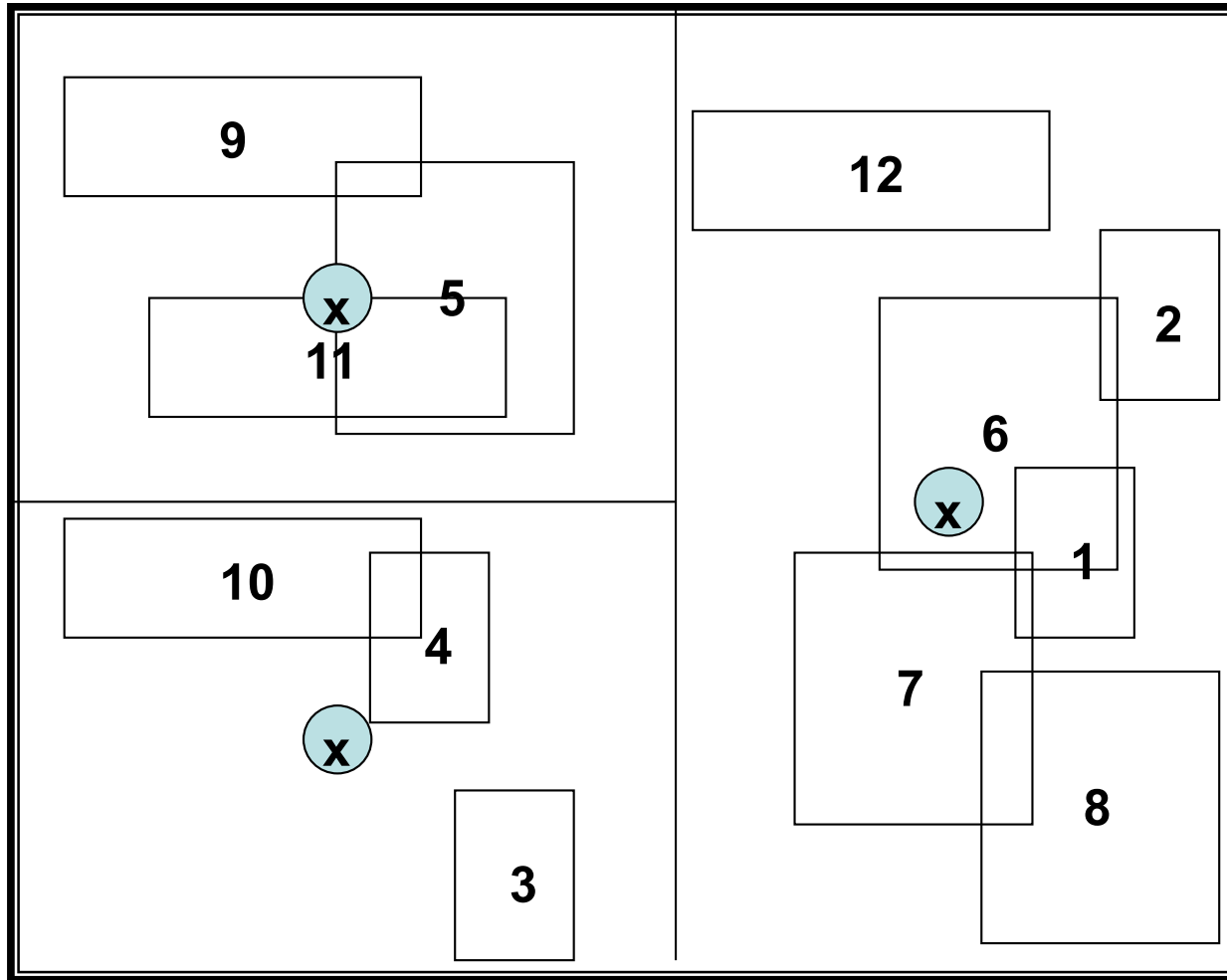


Example Placement

K = 3

Chip Area



Center of
Gravity

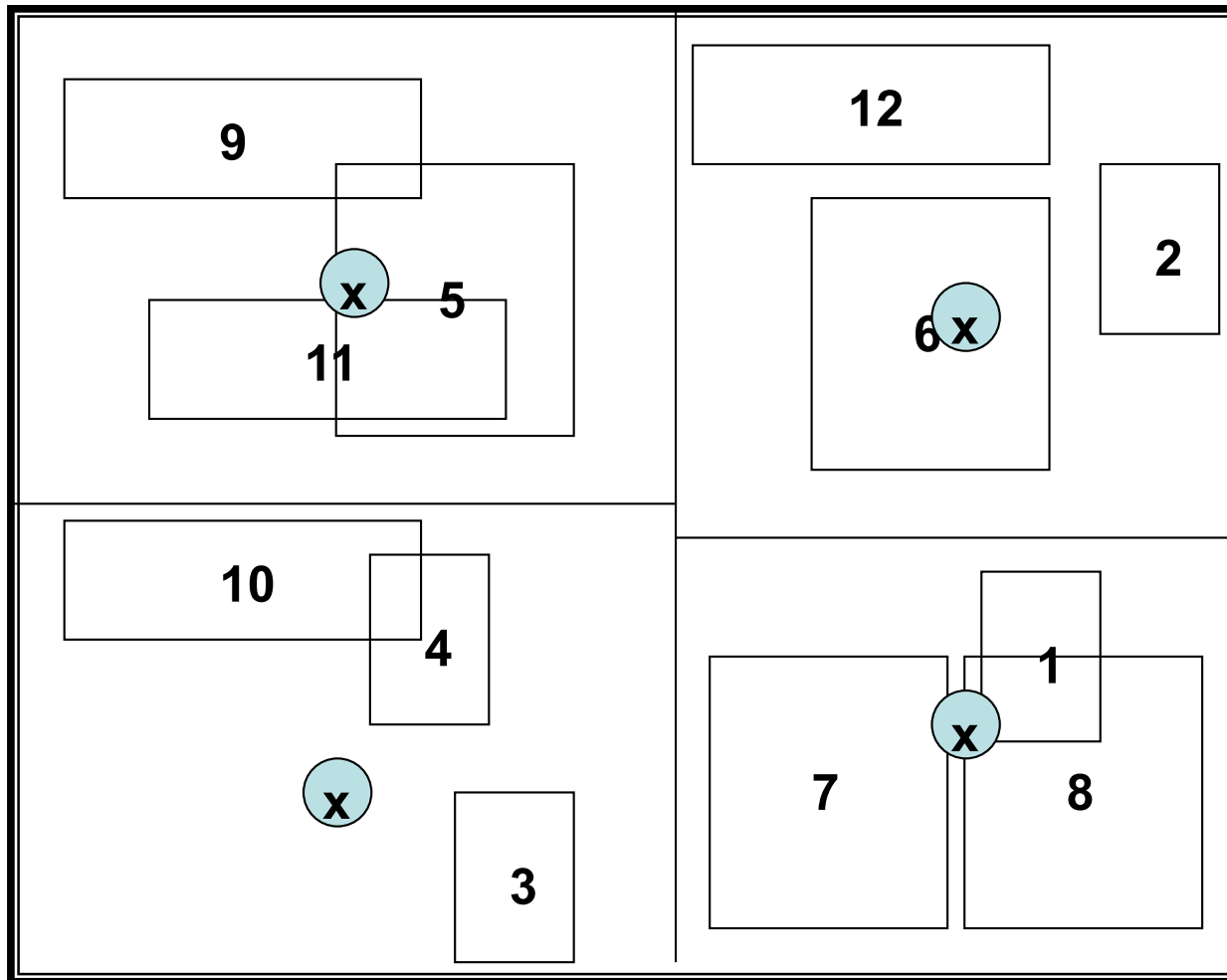


Example Placement

K = 3

Chip Area



Center of
Gravity

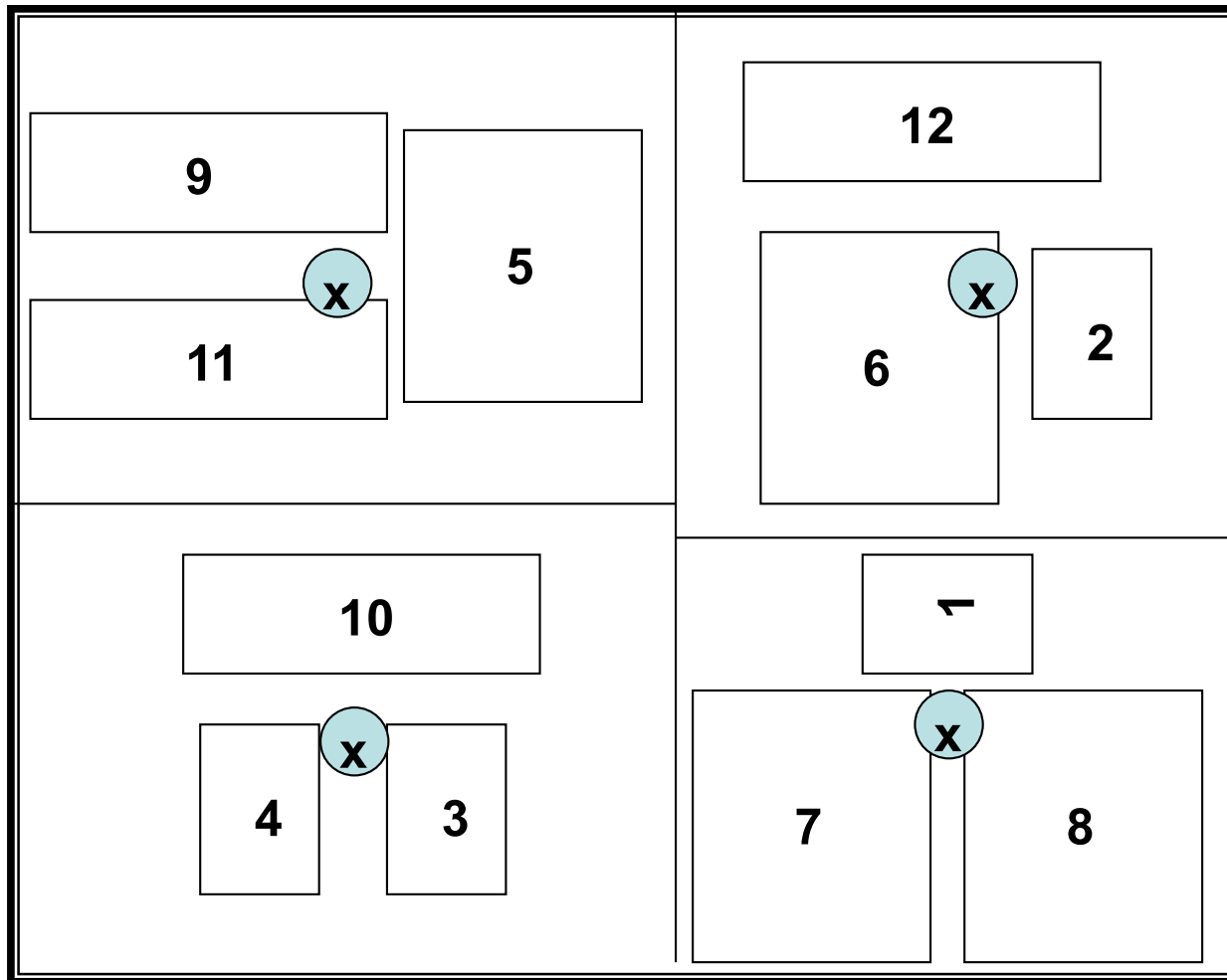


Example Placement

$K = 3$

Chip Area


Center of
Gravity



Improving Partitioning

Variation of cut direction and position

- Going through a sorted list of module coordinates, you can calculate C_p for every value of α by drawing the partition line after each module in sequence

Module Interchange

- Take a small set of modules in the partition and apply a min-cut approach

Repartitioning

- In the beginning steps of global optimization, modules are usually clustered around the centers of their regions
- If regions are cut near the center, placing a module on either side of the region could be fairly arbitrary
- Apply a heuristic, if two modules overlap near a cut then they are merged into one of the regions

See Gordian paper on web site for more details
(no paper review required!)

Legalization:

With any sort of analytical technique, we need to legalize the solution to remove overlaps.

Can you suggest an algorithm to do this?

Legalization: Straight-forward approach

sort cells

for each cell i {

 get x_i, y_i from Analytical solution

 find slot closest to (x_i, y_i)

 if this slot is empty, put cell there

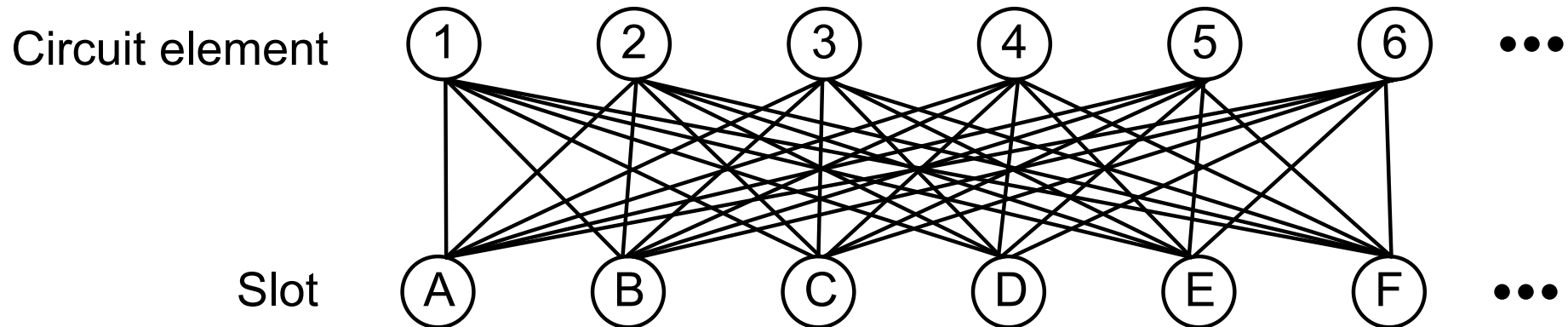
 else

 find “closest empty slot” to (x_i, y_i) and put cell there

}

Can you see some problems with this approach?

Legalization: Domino



Label each edge with the absolute difference between the location determined by AP and the actual location of slot

Find a minimum-weight matching.

This is an easy problem, and exact techniques are available, but they are $O(V^2E)$

Force Directed Spreading:

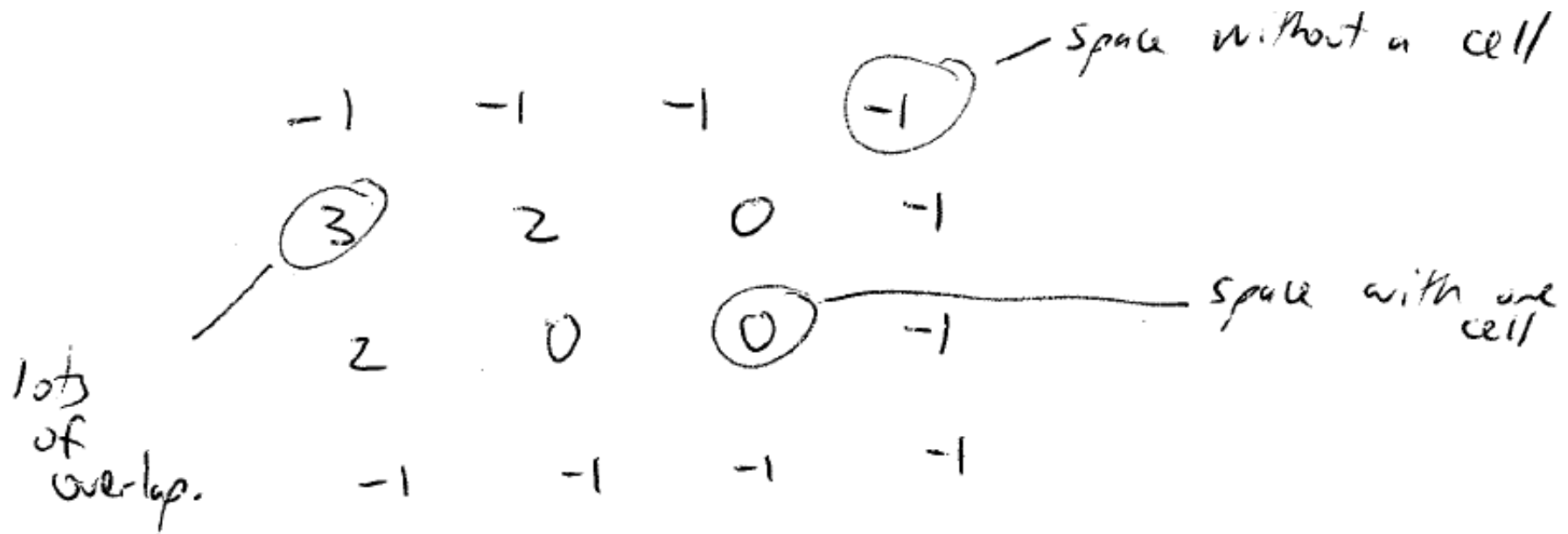
Change the formulation during analytical placement:

$$Q \vec{x} = \vec{c} + \vec{f}_x$$

Forces that pull cells
away from congested areas
and towards empty spaces

How do we calculate f ?

Start by defining a Matrix D:



Say a cell is at (x,y) . This will be pulled or repulsed by every other site (x',y') , depending on how full or empty that site is.

- We will calculate this force, and sum for all (x',y')
- And then repeat this for each (x,y)

Define $r = (x, y)$

$$r' = (x', y')$$

Then, $r - r'$ is the vector from r' to r

$|r - r'|$ is the distance from r' to r

So, the force on each cell can be written:

$$f(x, y) = k \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} D(x', y') \frac{\vec{r} - \vec{r}'}{|\vec{r} - \vec{r}'|^2} dx' dy'$$

After computing the forces on each object, get a new system of equations

- Solve and repeat

Overlaps are reduced each iteration

- Stop when there are “few” overlaps

For more details, see Eisenmann and Vorwerk papers on the web site (no paper review required!)

Timing-Driven Placement

A. Marquardt, B. Betz, J. Rose, “Timing Driven Placement for FPGAs”. On Connect site (no paper review req'd)

Break the cost function into two parts:

$$\text{wiring cost} = \sum_{\text{All nets } i} q(i) * (\text{bounding box for net } i)$$

$q(i)$ depends on number of terminals on i
- Scale up cost for large nets

$$\text{timing cost} = \sum_{\text{All nets } i} (\text{estimated delay of net } i) \text{Crit}^{\text{exp}}$$

2 or 3 works well

as defined last week

$$\Delta C = \lambda \frac{\Delta \text{timing cost}}{\text{previous timing cost}} + (1 - \lambda) \frac{\Delta \text{wiring cost}}{\text{previous wiring cost}}$$

lets you trade off timing for routability

Incremental Placement

Most FPGA CAD tools offer incremental placement (and routing):

- If part of the design hasn't changed, you don't need to motivate it

In current tools:

- If a Verilog module is touched, it is recompiled
- Does not try to look inside a module to find parts that haven't changed

Even the very simplest changes will require a recompile of a module

Some academic work has been presented that looks at a gate-level circuit and find gates that can be re-used. Is this too low level?

Towards “Agile Methods” for Hardware:

Software designers use “Agile Methods”:

```
implement the simplest “shippable” demo version
while there is money left for the project {
    add a small number of capabilities, creating a new “shippable”
}
```

Characterized by tight (short) design loops, often organized in “sprints”

FPGA incremental design flows are not set up this way

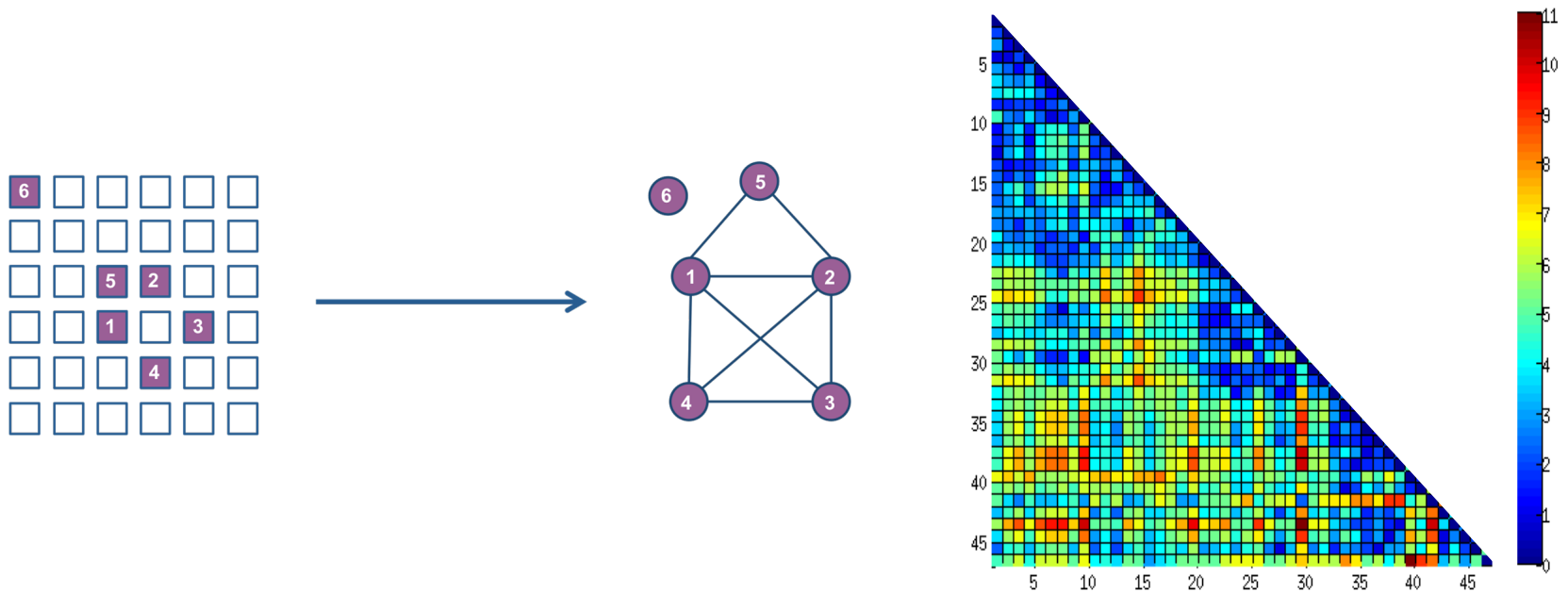
- They operate on the level of modules
- When we add functionality, we change more than one module!

Need to re-think incremental design techniques...

Iterative placement:

Iterative placements: how does placement i relate to placement $i-1$?

Lots of work in this area, but lots left to do:



Courtesy of Farnaz Gharibian, Simon Fraser University

Machine Learning – Google Brain

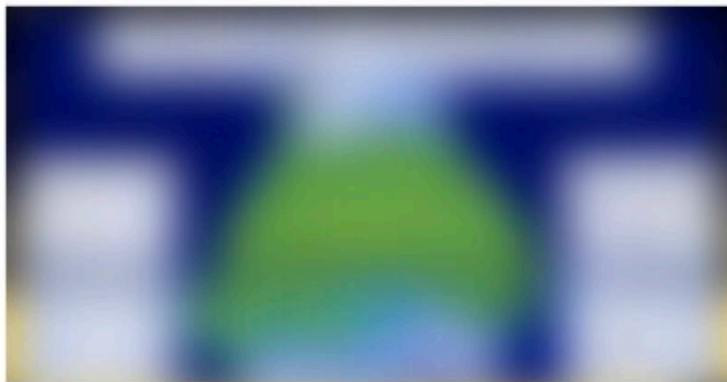
From NeurIPS 2019 (Dec 2019):

Google Research

Results on a TPU Design Block

White blurred area are macros (memory) and green blurred area are standard cell clusters (logic)
ML placer finds smoother, rounder macro placements to reduce the wirelength

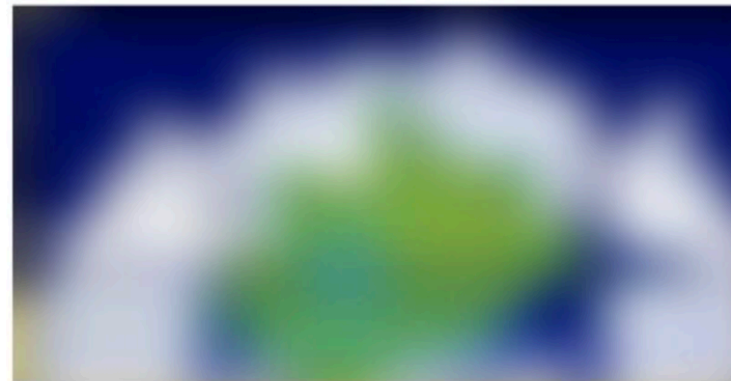
Human Expert



Time taken: **~6-8 person weeks**
Total wirelength: 57.07m
Route DRC* violations: 1766

DRC: Design Rule Checking

ML Placer



Time taken: **24 hours**
Total wirelength: 55.42m (-2.9% shorter)
Route DRC violations: 1789 (+23 - negligible difference)

Summary

1. Simulated Annealing Based Placement
 - Slow, but very general
 - Adapts to any constraint that is easily computable
2. Partitioning-Based Methods: not used so much now
3. Analytical Methods (Gordian)
 - State of the art in ASIC placers
 - Becoming more common in FPGA placers
4. Spreading / Legalization
5. Timing-Driven Placement