The Physical Placement Problem in Integrated Circuits

Prof. Kurt Keutzer

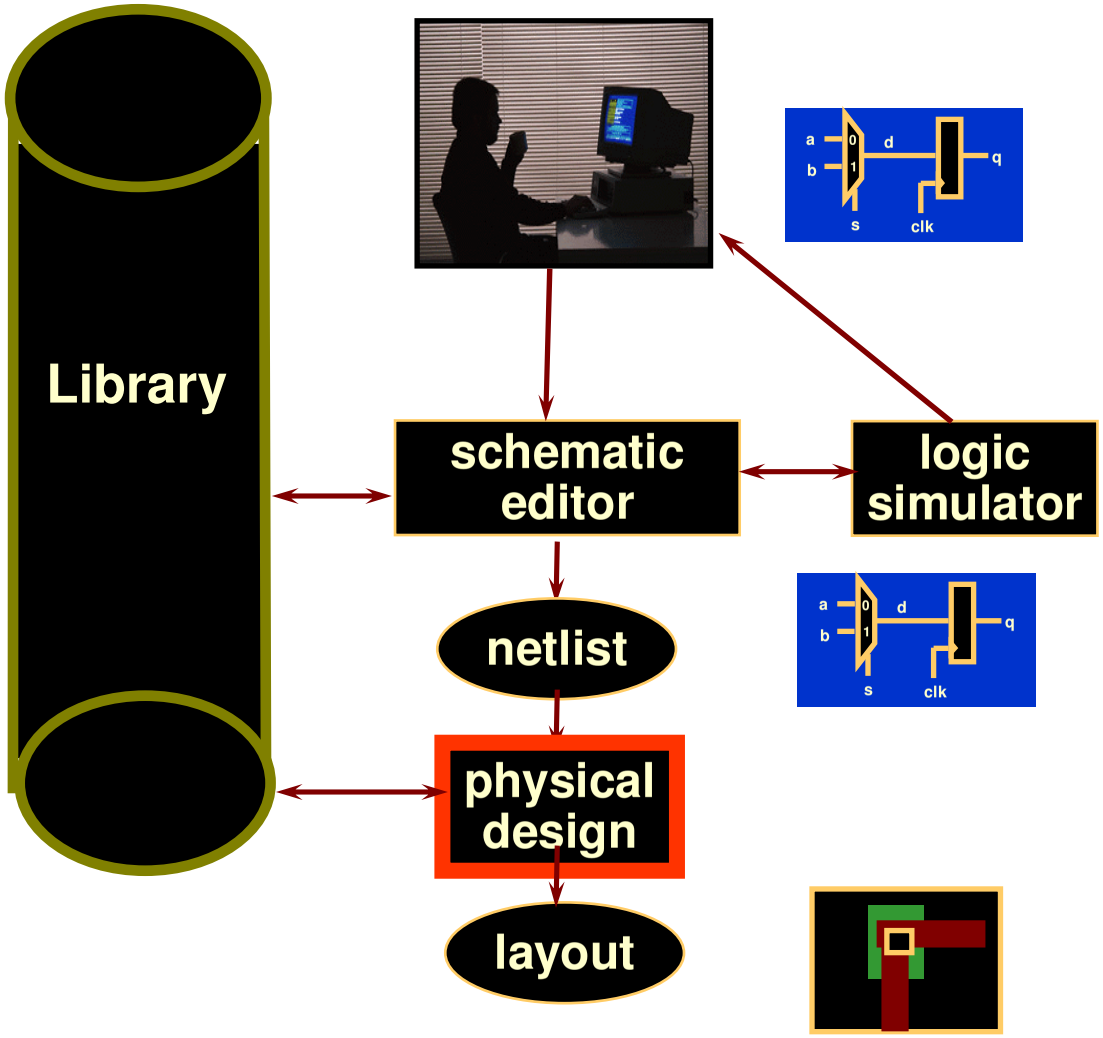
EECS

University of California

Berkeley, CA

Thanks to Prof. A. Kahng

Schematic Entry Design Flow



Designer designs the circuit on napkins and blackboard

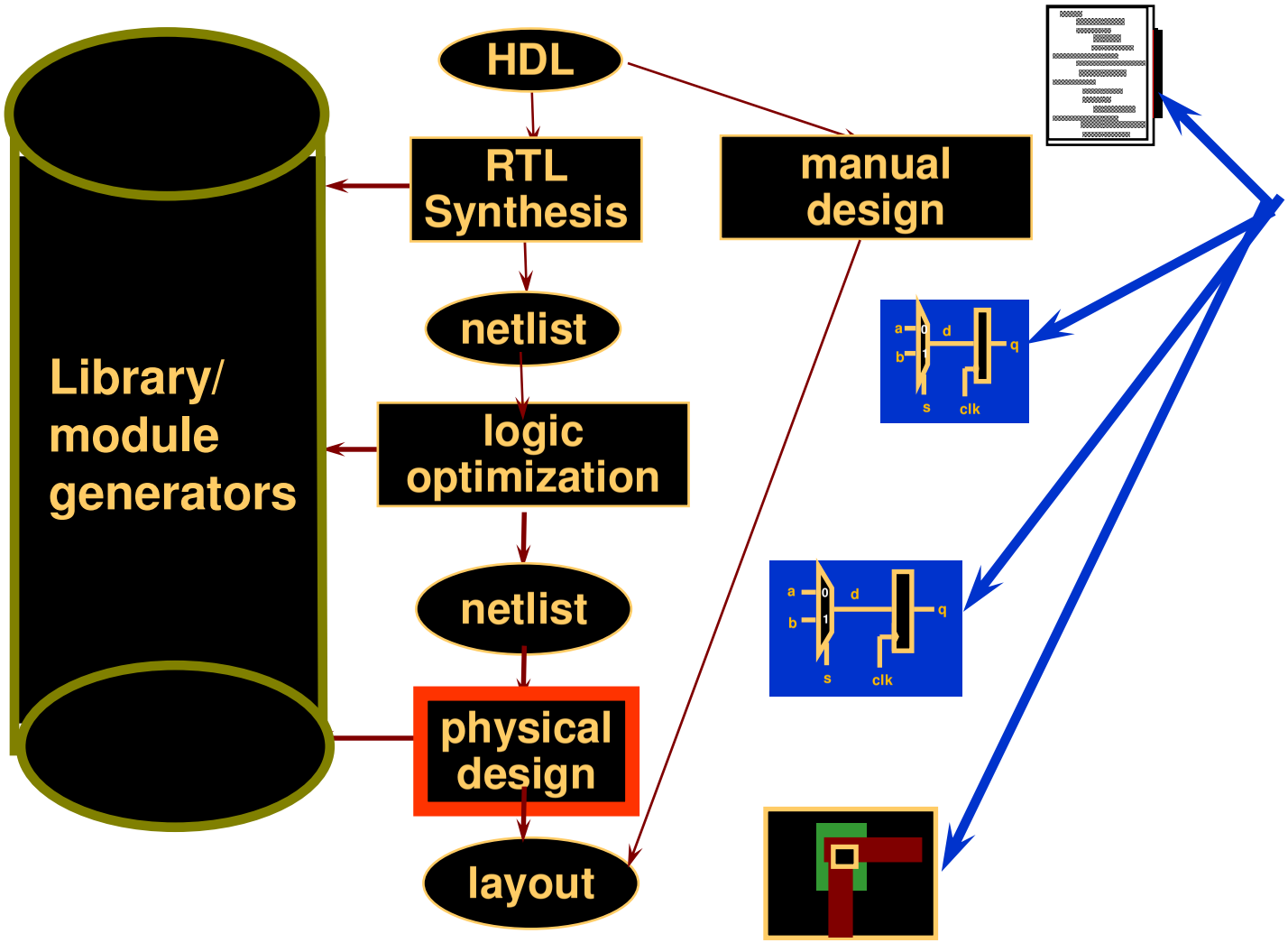
Gate-level details of the circuit are entered in a schematic entry tool

Vectors are generated to verify the circuit

When logic is correct the netlist is passed off to another group to lay out

Automated place and route tools create layout

RTL Design Flow

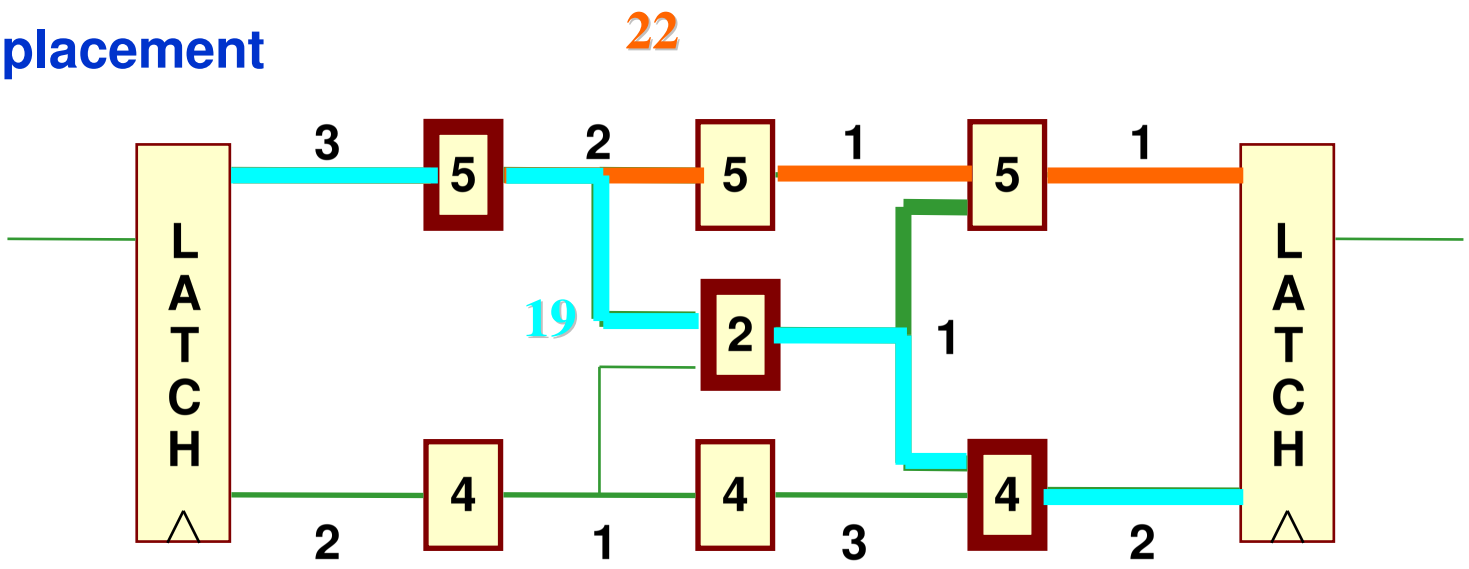


The Netlist

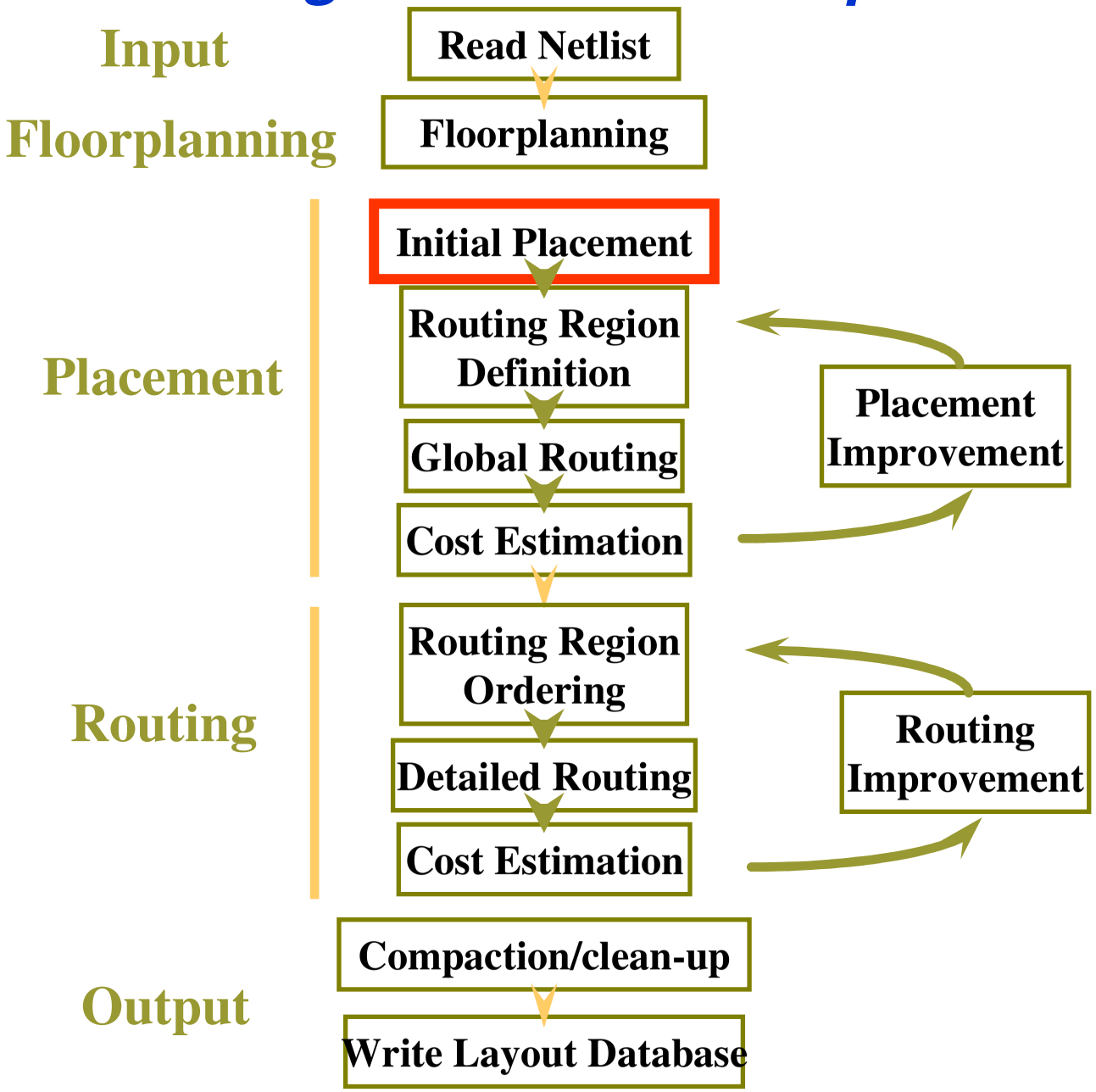
When we finished synthesis or schematic entry we produced a netlist:

- Functionally correct
- Correct timing relative to the models we used

Now we wish to turn that into a correctly functioning placement



Physical Design: Overall Conceptual Flow



Formulation of the Placement Problem

Given:

- A netlist of cells from a pre-defined semiconductor library
- A mathematical expression of that netlist as a vertex-, edge-weighted graph
- Constraints on pin-locations expressed as constraints on vertex locations / aspect ratio that the placement needs to fit into
- One or more of the following: chip-level timing constraints, a list of critical nets, chip-level power constraints

Find:

- Cell/vertex locations to minimize placement objective subject to constraints

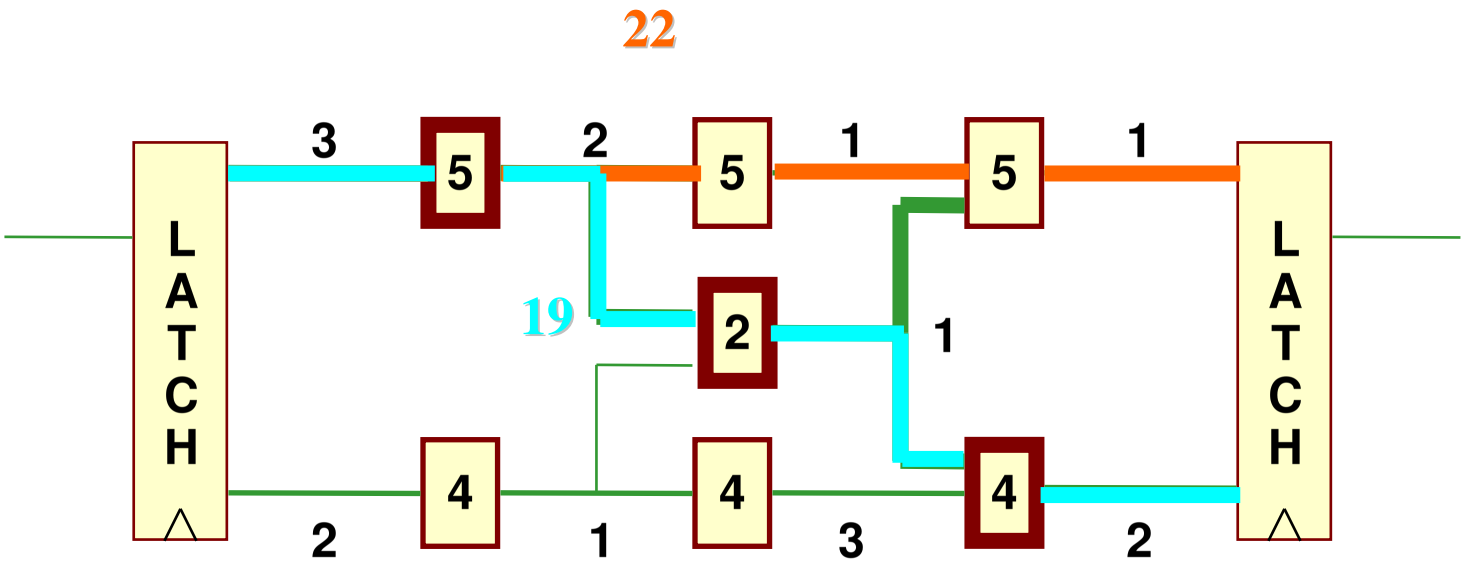
Objectives:

- minimal delay (fastest cycle time)
- minimal area (least die area/cost
- other niceties: e.g. power

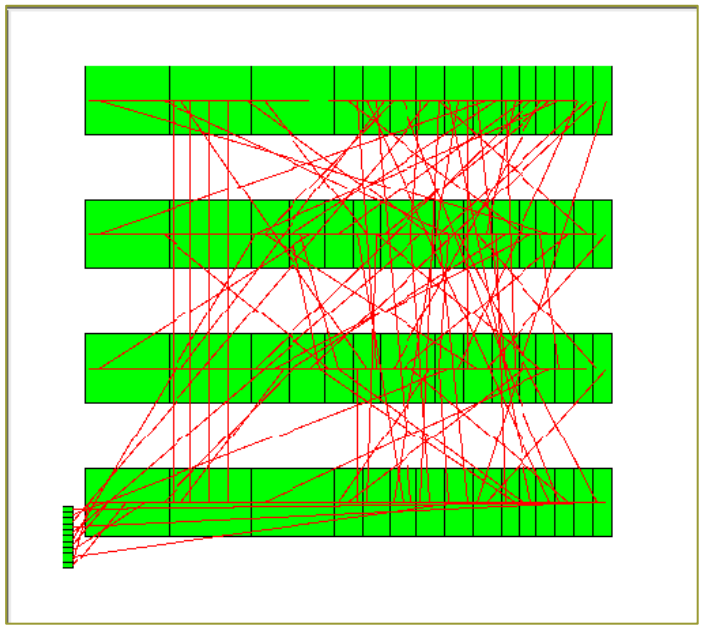
Constraint-driven design

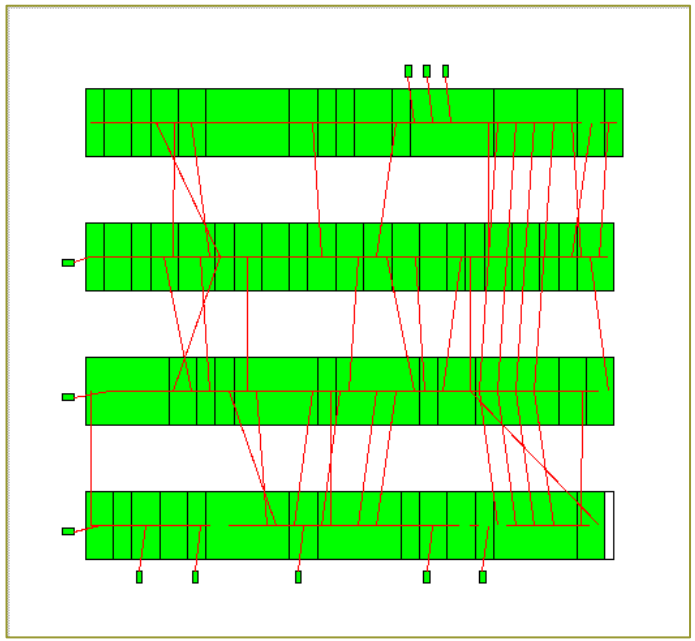
When we finished synthesis or schematic entry we believed that our circuit met its timing constraints based on expected values of interconnect delays

We want downstream physical design tools to honor those constraints!!!



Results of Placement





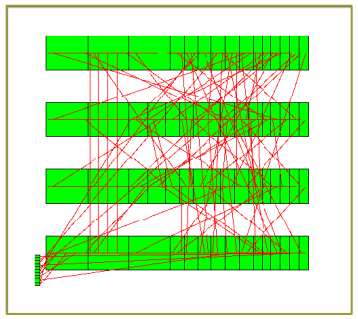
A bad placement

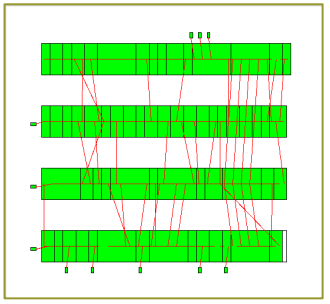
A good placement

What's good about a good placement?
What's bad about a bad placement?

A. Kahng

Results of Placement





Bad placement causes routing congestion resulting in:

- **Increases in circuit area (cost) and wiring**
- **Longer wires → more capacitance**
  - **Longer delay**
  - **Higher dynamic power dissipation**

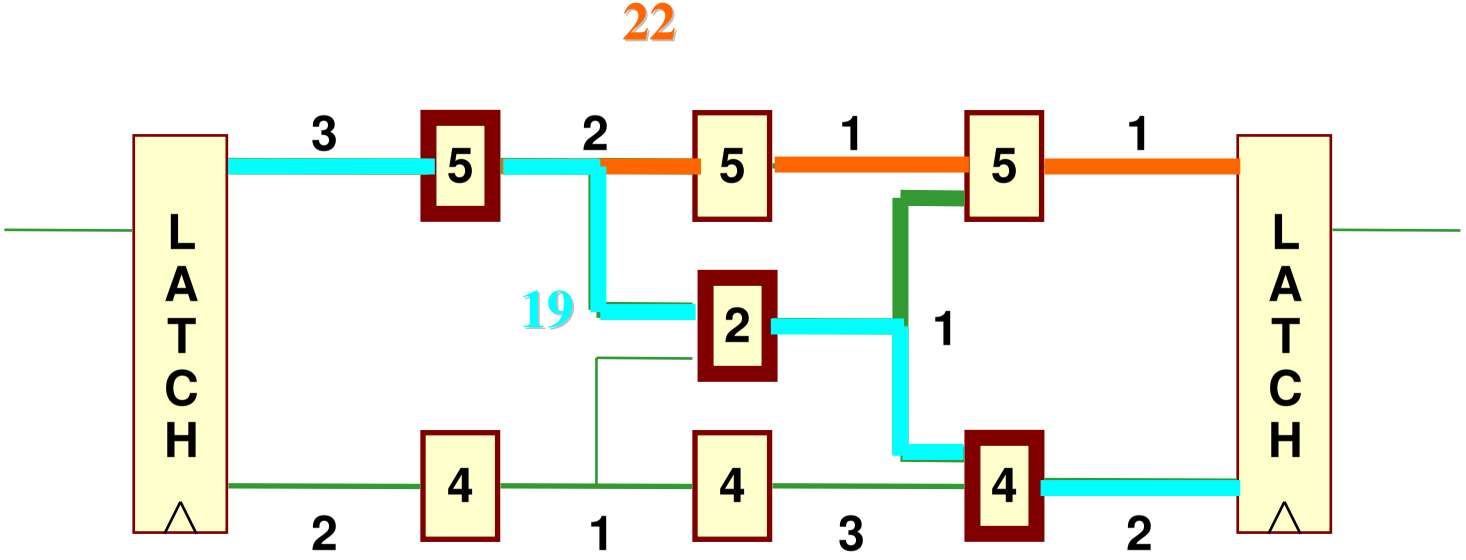
Good placment

- **Circuit area (cost) and wiring decreases**
- **Shorter wires → less capacitance**
  - **Shorter delay**
  - **Less dynamic power dissipation**

Why can't PD obey constraints?

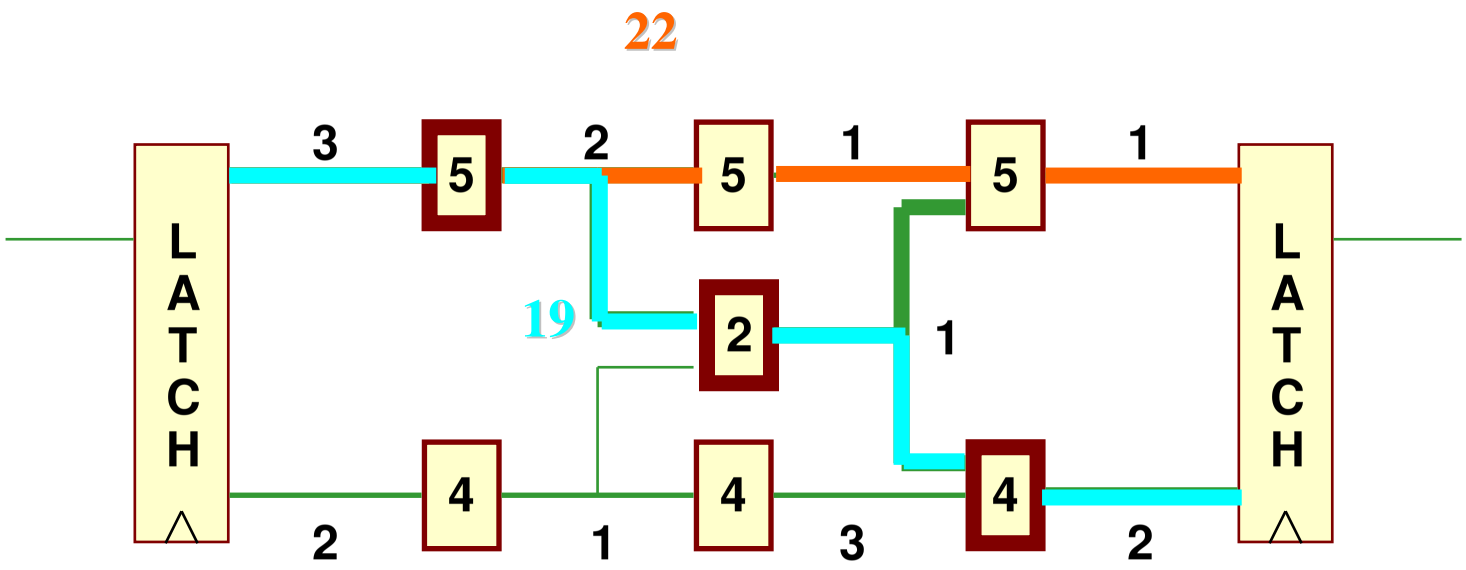
High level (synthesis, schematic entry) estimates of wire delay may be very naïve

As interconnect delay increases relative to gate delay and becomes less predictable it may be impossible to place and route a circuit in a way that honors initial constralnts



Timing and PD

Stan Chow Ammocore
Andrew B. Kahng UCSD
Majid Sarrafzadeh UCLA



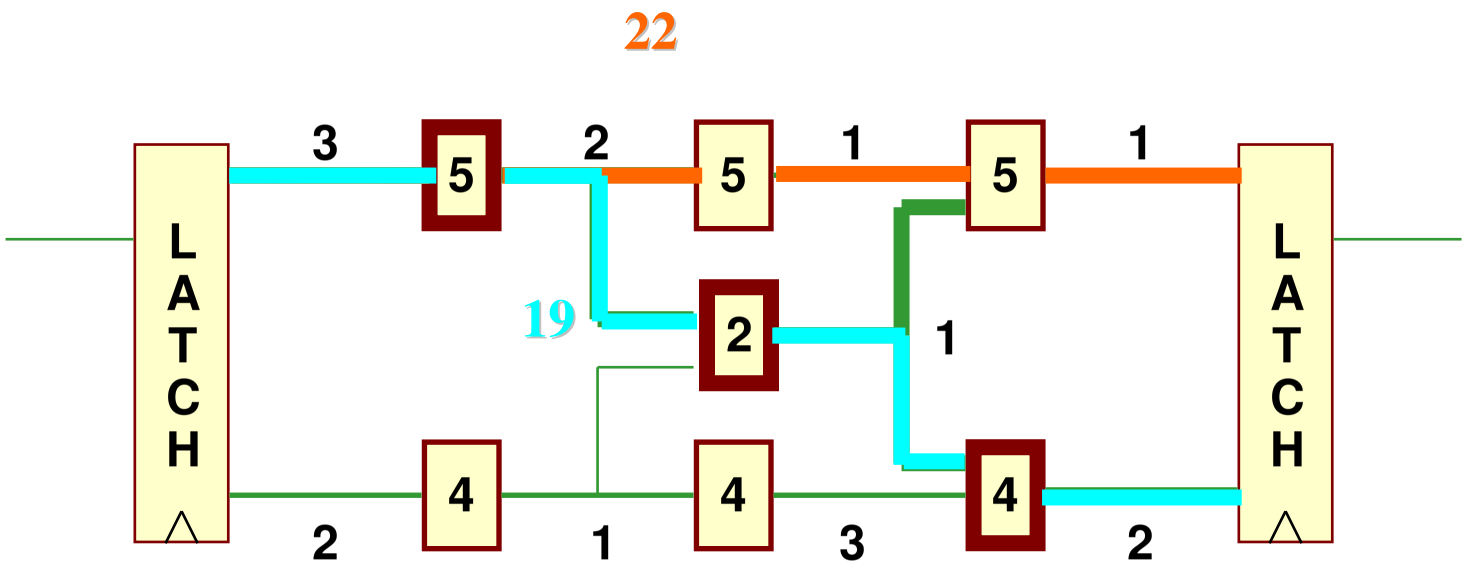
How do we get [reflect] the delay numbers on the gate/interconnect?

KK: We have a few choices about what to do with the netlist during placement:

- **Leave it exactly the same**
- **Resize gates and add buffers as we go**
- **Re-factor/duplicate logic**

Timing and PD - 2

Stan Chow Ammocore
Andrew B. Kahng UCSD
Majid Sarrafzadeh UCLA

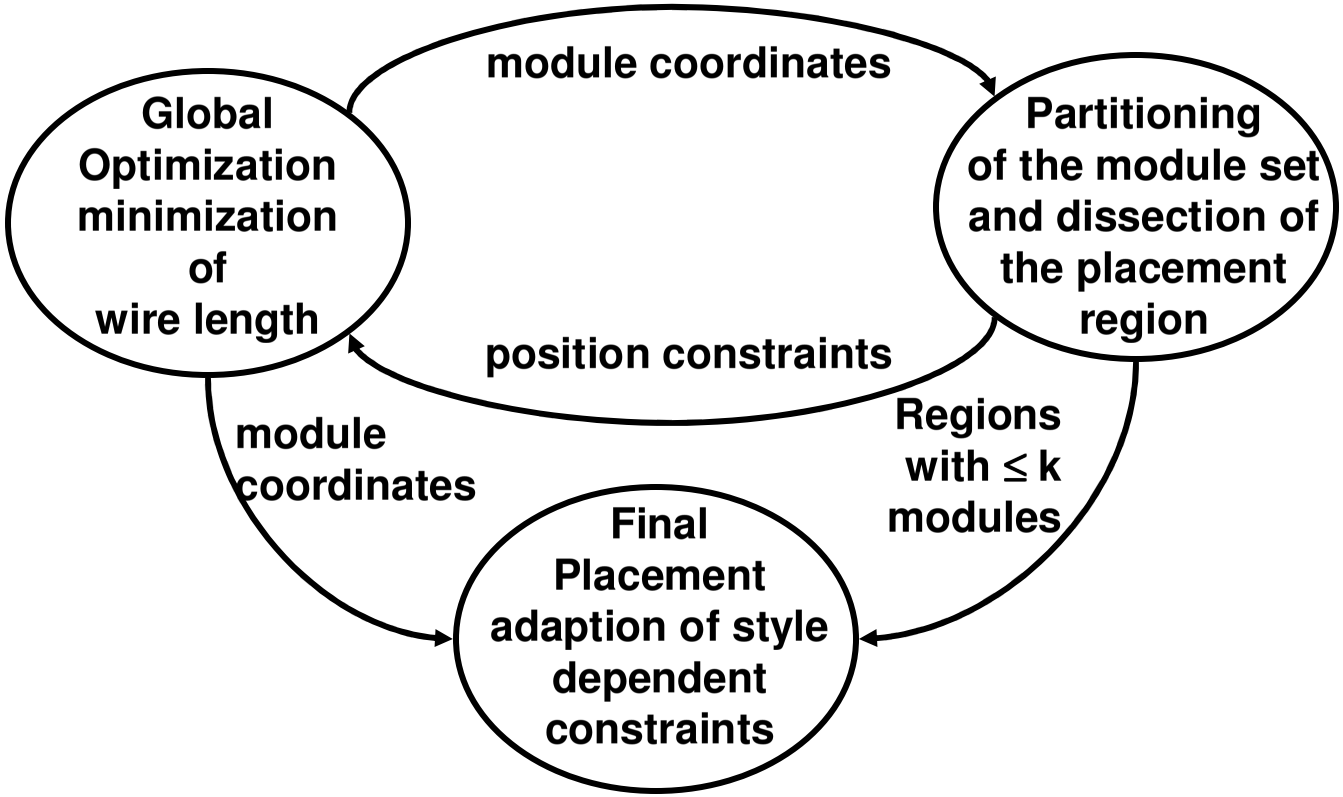


How do we get [reflect] the delay numbers on the gate/interconnect?

KK: We have a few choices about what to do with the netlist during placement:

- **Leave it exactly the same – simplifying assumption**
- **Resize gates and add buffers as we go**
- **Re-factor/duplicate logic**

Gordian Placement Flow



Data flow in the placement procedure GORDIAN

Complexity

space: O(m) time: $Q(m^{1.5} \log^2 m)$

Final placement

- standard cell
- macro-cell &SOG

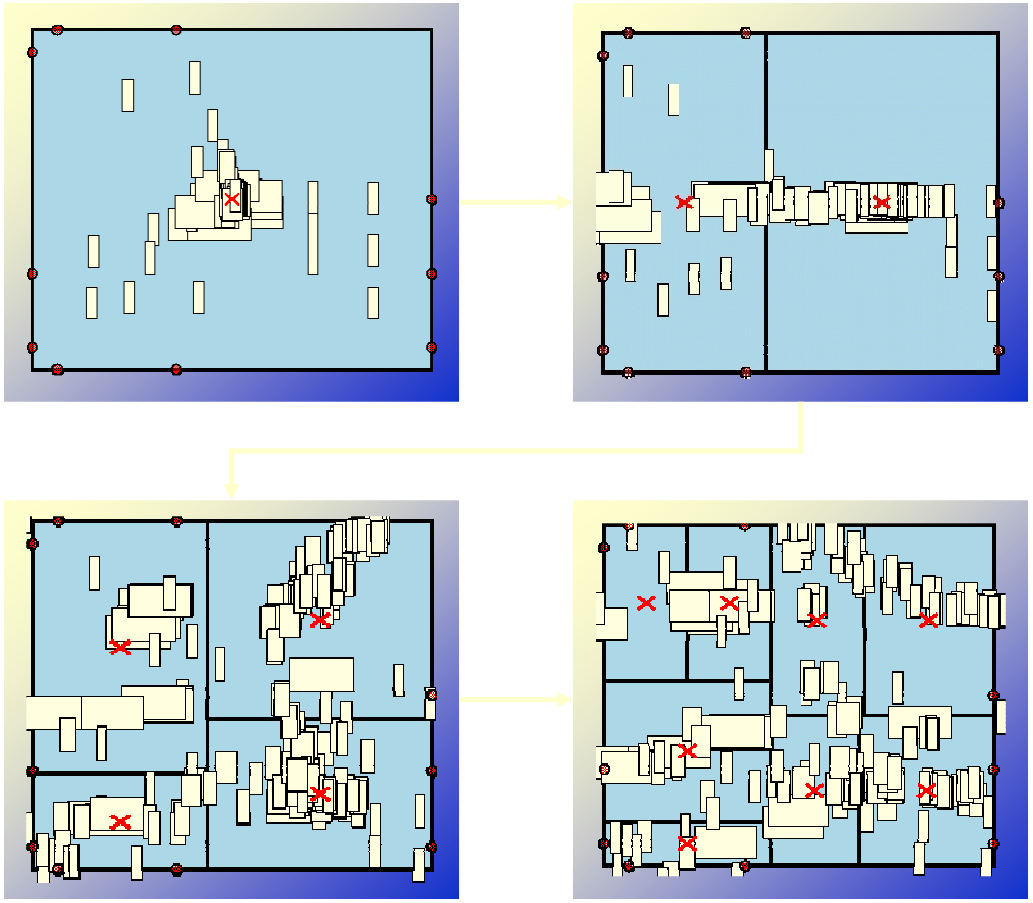
Gordian: A Quadratic Placement Approach

- **Global optimization:**
  solves a sequence of quadratic programming problems
- **Partitioning:**
  enforces the non-overlap constraints

GORDIAN (quadratic + partitioning)

Initial
Placement

Partition
and Replace



A. Kahng

Gordian Placement Flow

J. Kleinhaus, G. Sigl, F. Johannes, K. Antreich,
GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization,
IEEE Trans. on CAD, March, 1991, pp. 356 - 365

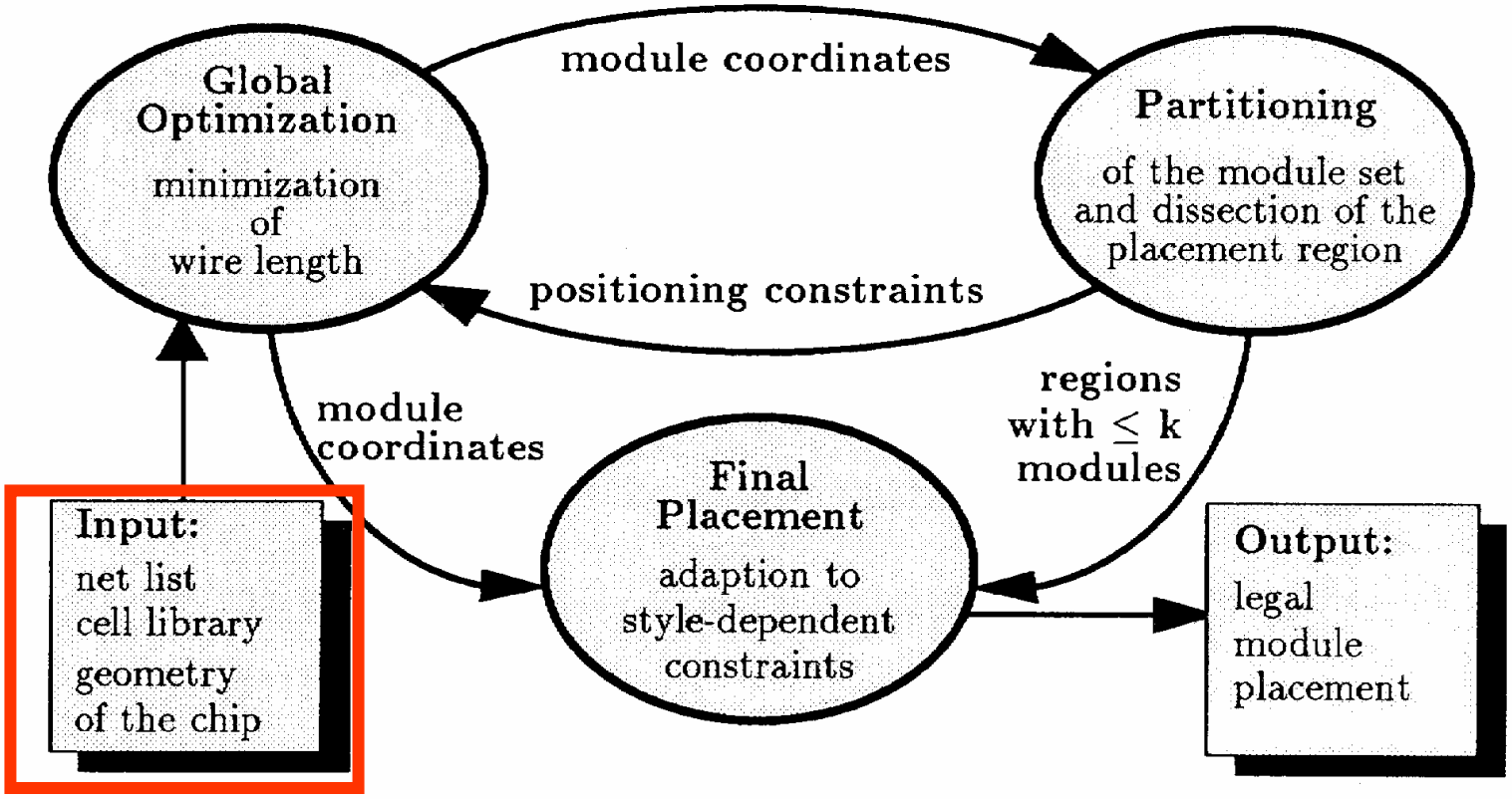


Fig. 1. Data flow in the placement procedure GORDIAN.

Library

Contains for each cell:

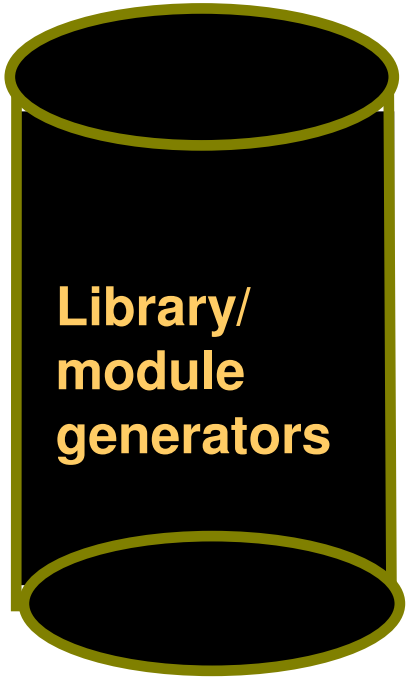
- Functional information: cell = a *b * c
- Timing information: function of
  - input slew
  - intrinsic delay
  - output capacitance

  non-linear models used in tabular approach
- Physical footprint (area)
- Power characteristics

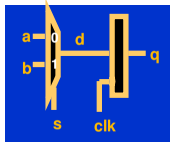
Wire-load models - function of

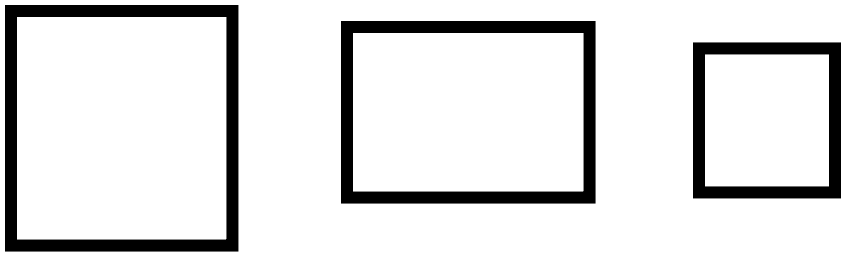
- Block size
- Wiring

Library, Netlist, and Aspect Ratio



Netlist - >100K -> 10M cells from library





Size and aspect ratio of core die

Setting up Global Optimization

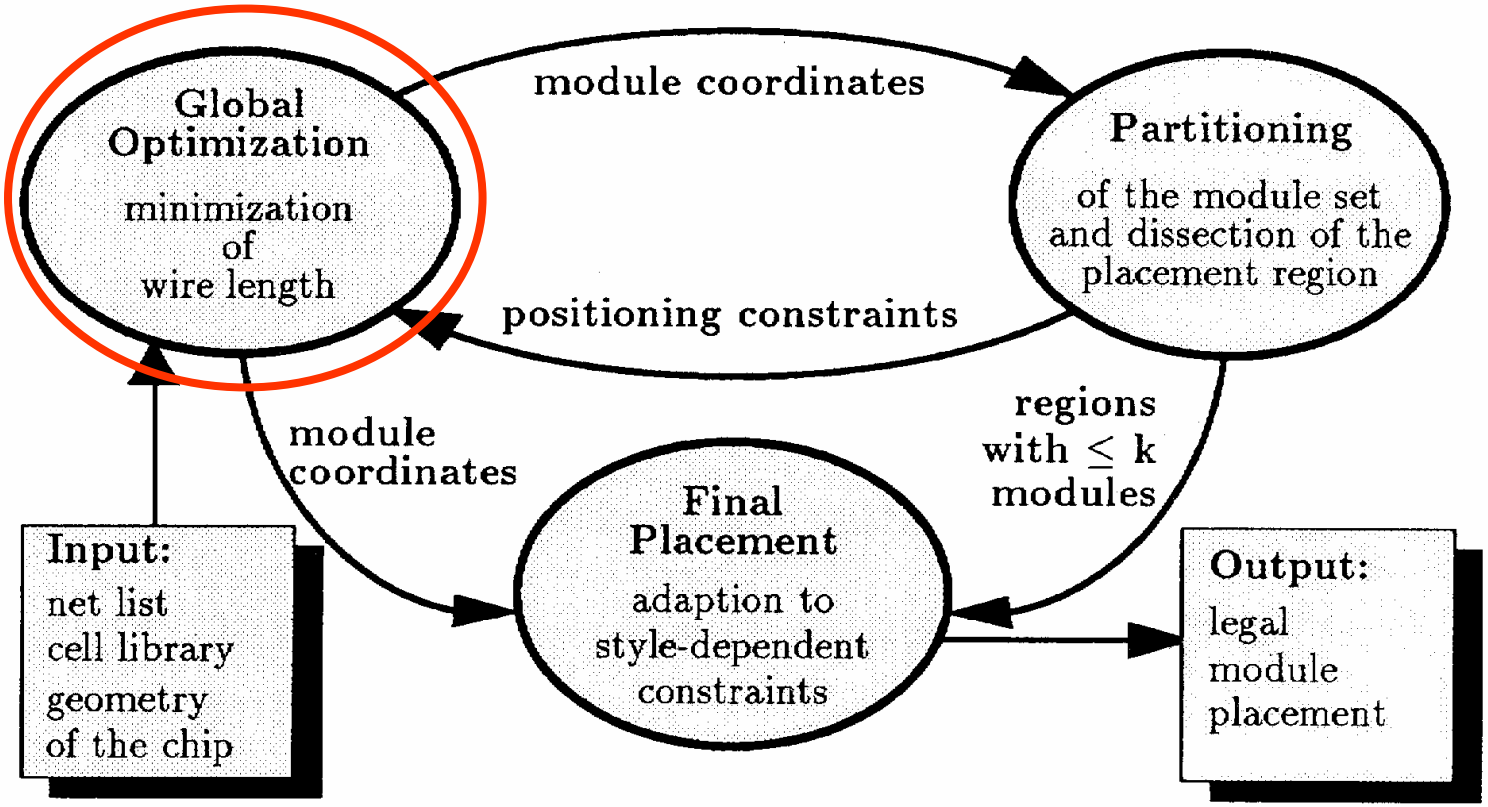


Fig. 1. Data flow in the placement procedure GORDIAN.

GORDIAN: Global Placement

We want to optimize the delay of critical paths

Instead we:

- optimize the sum of squares of net-lengths times a static weight

Global placement by quadratic wire-length optimization

- Problem is computationally tractable and well behaved
- Global connectivity is considered at all stages
  - An increasing number of constraints is imposed
  - Global placement of modules is obtained simultaneously for all sub-problems
  - No dependence on processing sequence

Quadratic placement clumps cells in center

Partitioning spreads cells and imposes new constraints on further optimization

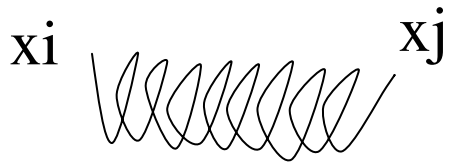
Intuitive formulation

Given a series of points x1, x2, x3, … xn

and a connectivity matrix *C* describing the connections between them

(If cij = 1 there is a connection between xi and xj)

Find a location for each xj that minimizes the total sum of all spring tensions between each pair <xi, xj>



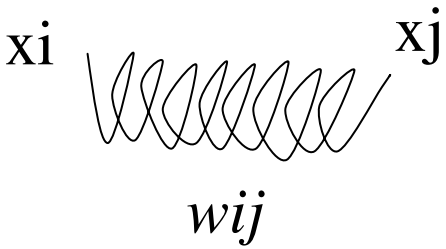
Problem has an obvious (trivial) solution – what is it?

Improving the intuitive formulation

To avoid the trivial solution add constraints: *Hx=b*

- **These may be very natural - e.g. endpoints (pads)**

x1 **xn**



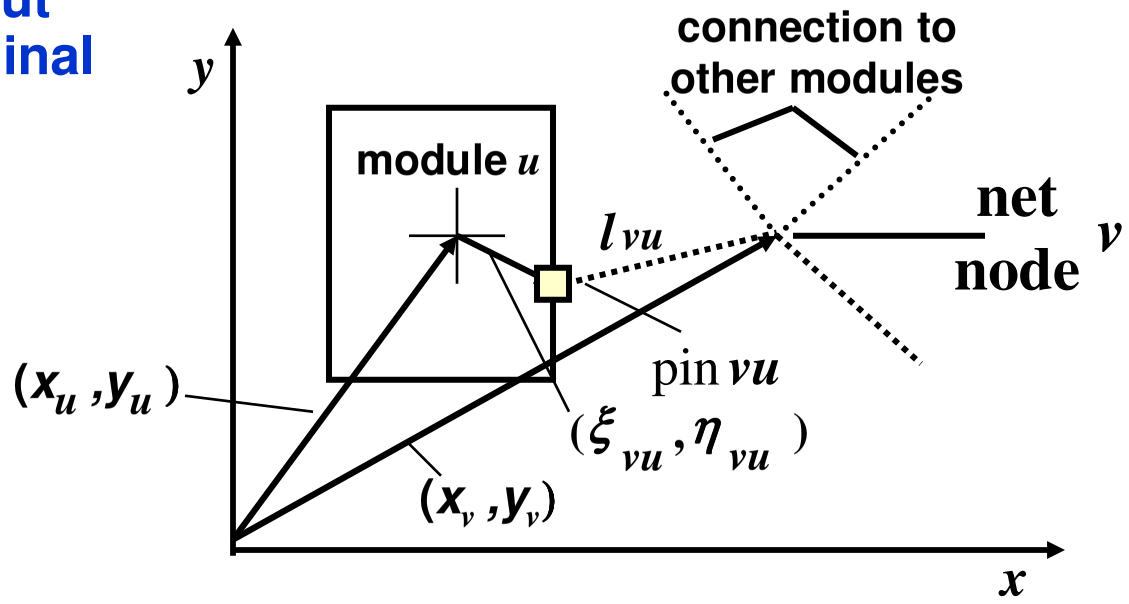
To integrate the notion of ``critical nets''

- **Add *weights wij* to nets**

***wij* - some springs have more tension should pull associated vertices closer**

Modeling the Net's Wire Length

What about multiterminal nets?

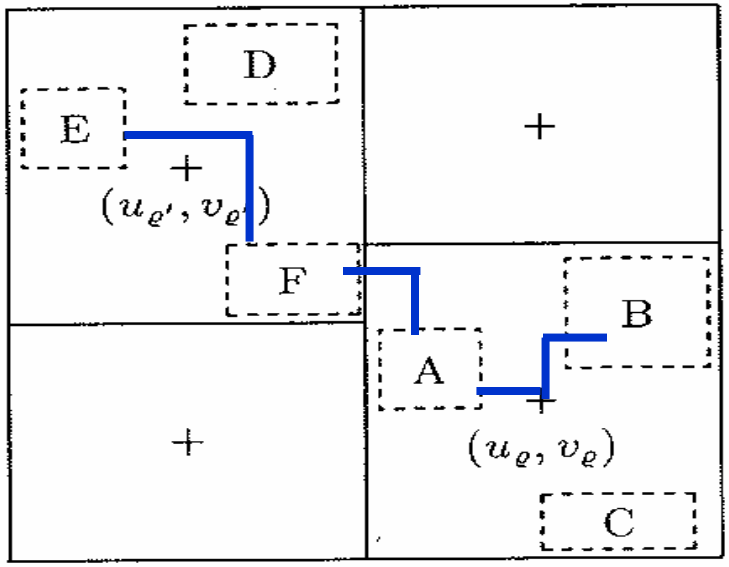


The length Lv of a net v is measured by the squared distances from its points to the net's center

$$L_v = \sum_{u \leftarrow M_v} [(x_{uv} - x_v)^2 + (y_{uv} - y_v)^2]$$

$$(\; x_{uv} = x_u + \xi_{uv} \quad ; \quad y_{uv} = y_u + y_{vu} \;)$$

What do we really want to optimize?



For high-performance circuits, we want to minimize longest path through network

- **In addition to total wire length**

Approximate delay minimization by minimization of squared wire length

- **Penalizes long wires**

Quadratic Objective Function

Objective function: weighted sum of the squared net lengths

$$\Phi = \sum_{v \in N} L_v w_v$$

Where N is nets set, w_v is net weight, and L_v is the measure of net length

Can re-formulate it in terms of block coordinates (in one 1-D)

$$\Phi(x) = \sum w_{ij}(x_i - x_j)^2 = 0.5 x^T C x$$

- ***xi*** **- locations of vertices (modules)**
- ***wij*** **- edge weights (net weights)**
- ***C*** **- the system matrix**

Generalize to 2-D

$$\Phi(x, y) = \sum w_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2] = 0.5 x^T C x + 0.5 y^T C y$$

Formulating Optimization Problem

For simplicity, we'll consider a 1-D formulation

Need to add a term to account for fixed modules

$$\Phi(x) = 0.5x^T Cx + d^T x$$

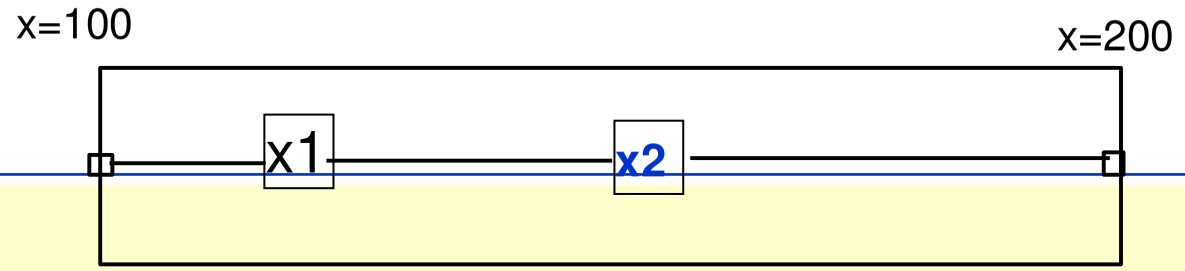
The vector d is constructed based coordinates of the fixed modules and pin coordinates of all modules

Also need to constraint placement s.t. center-of-gravity (the area weighted mean of all coordinates) corresponds to the center of placement region:

$$Ax = u$$

The vector *u* contains geometric centers of placement regions, the matrix *A* reflects the assignment of modules to placement regions

Toy Example:



$$Cost = (x_1 - 100)^2 + (x_1 - x_2)^2 + (x_2 - 200)^2$$

$$\frac{\partial}{\partial x_1} Cost = 2(x_1 - 100) + 2(x_1 - x_2)$$

$$\frac{\partial}{\partial x_2} Cost = -2(x_1 - x_2) + 2(x_2 - 200)$$

setting the partial derivatives = 0 we solve for the minimum Cost:

Ax + B = 0

$$\begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -200 \\ -400 \end{bmatrix} = 0$$

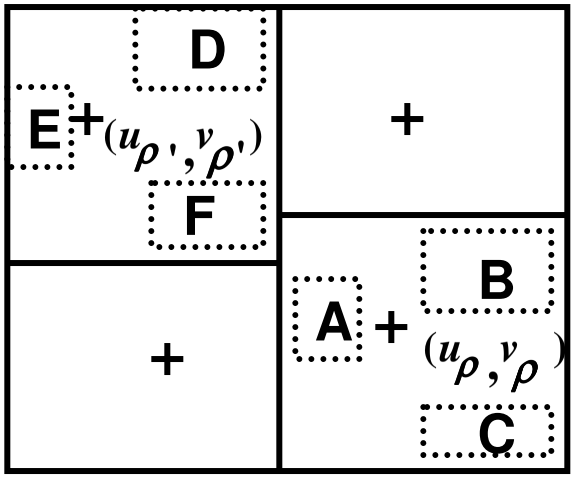
$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -100 \\ -200 \end{bmatrix} = 0$$

x1=400/3 x2=500/3

Kurt Keutzer

D. Pan

Quadratic Optimization Problem



$$A^{(l)} = \begin{array}{c} \\ \vdots \\ \rho \\ \rho' \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & * & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & * & * & * & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \end{array}$$

- **Linearly constrained quadratic programming problem**

$$\min_{x \in R^m} \{ \Phi(x) = x^T C x + d^T x \}$$

$$\text{s.t. } A^{(l)} x = u^{(l)}$$

Accounts for fixed modules

Wire-length for movable modules

Center-of-gravity constraints

Problem is computationally tractable, and well behaved

Commercial solvers available: mostek

What do I need to know about QP?

- Get an intuitive sense of what quadratic programming is trying to solve
  - Create a good placement of the netlist by placing cells so as to minimize the squares of the wirelengths in the netlist
- Understand strengths and weaknesses of the formulation
  - Strength- optimal solution!
  - Weakness –
    - quadratic vs. linear wirelengths
    - Never knows (or optimizes) the length of a path- only the individual 2-pin nets
- Don't need to understand the underlying mathematics

Global Optimization Using Quadratic Placement

Quadratic placement clumps cells in center

Partitioning divides cells into two regions

- Placement region is also divided into two regions

New center-of-gravity constraints are added to the constraint matrix to be used on the next level of global optimization

- Global connectivity is still conserved

Setting up Global Optimization

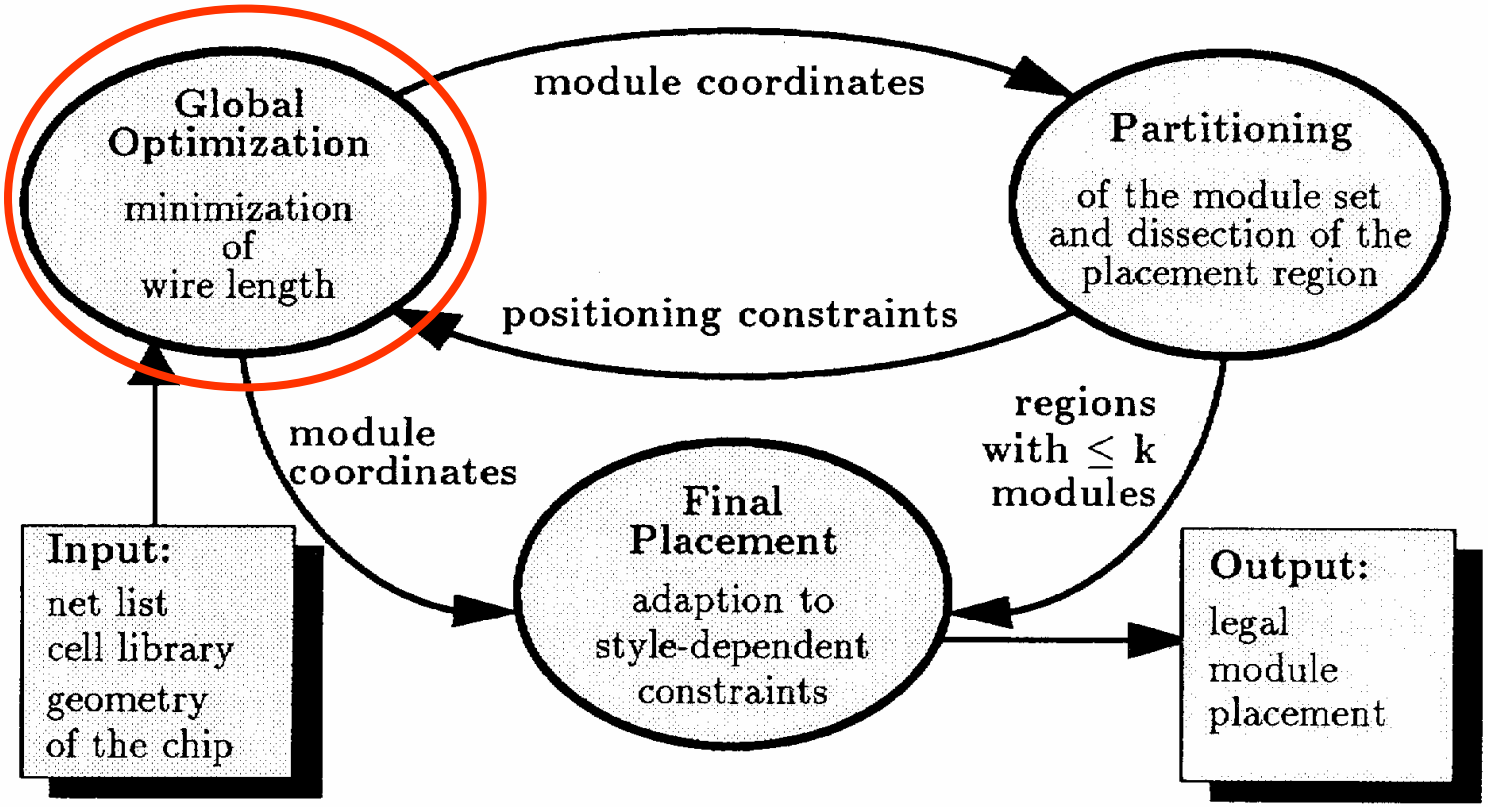
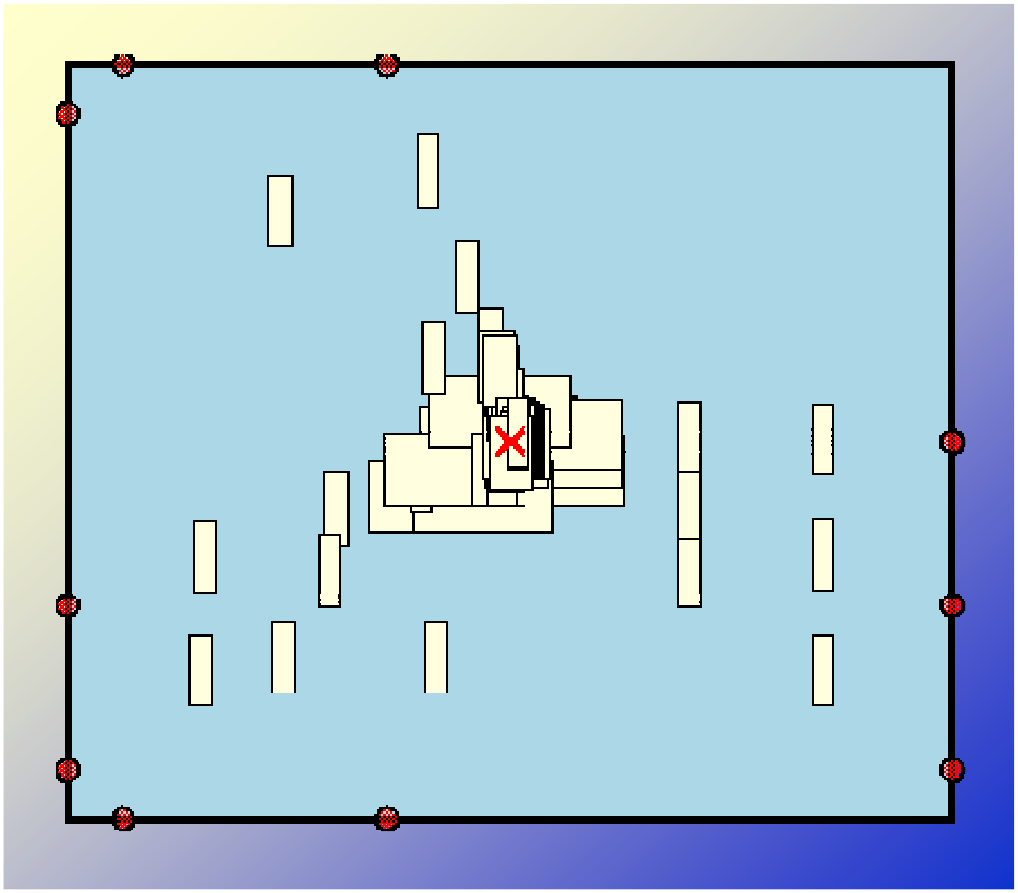


Fig. 1. Data flow in the placement procedure GORDIAN.

Layout After Global Optimization



A. Kahng

Partitioning

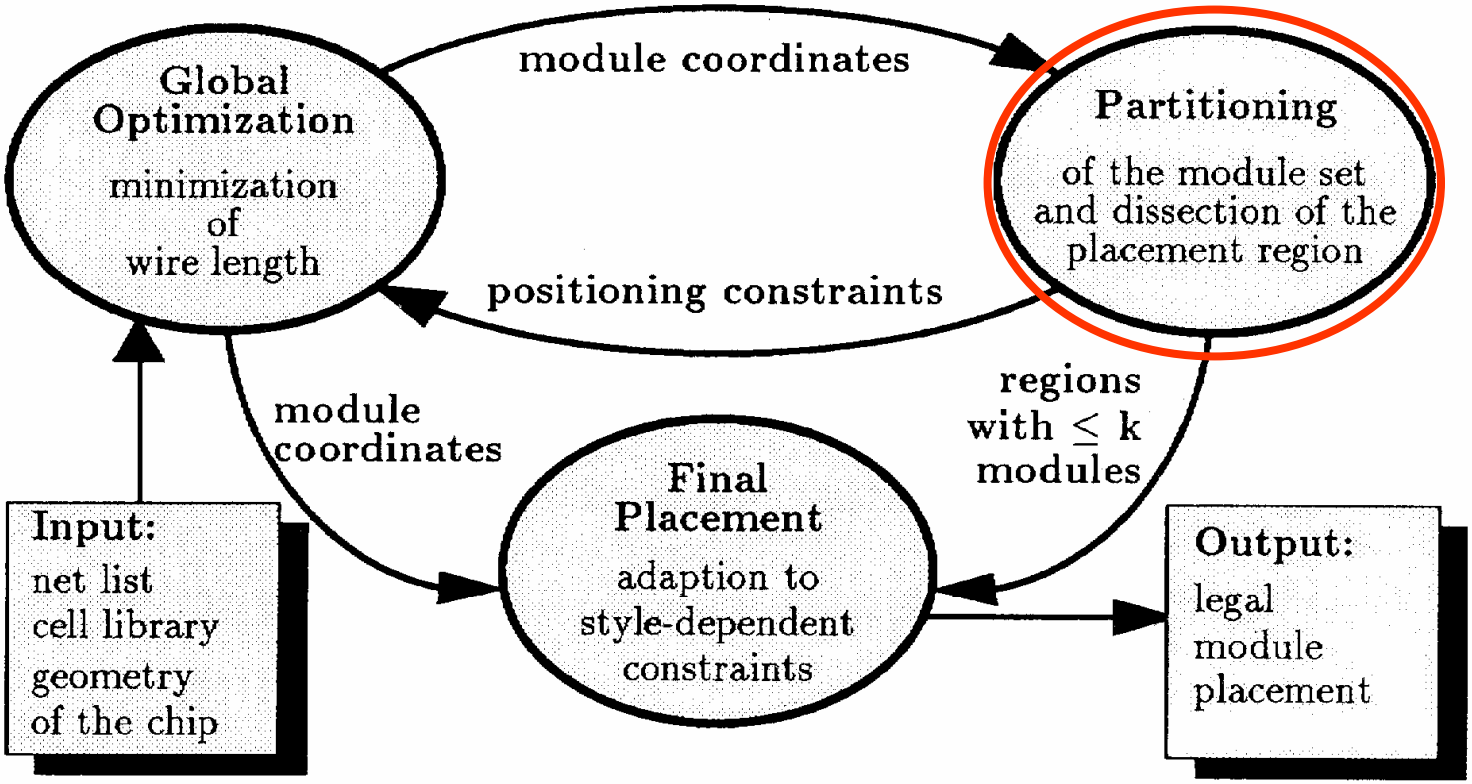


Fig. 1. Data flow in the placement procedure GORDIAN.

Partitioning

In GORDIAN, partitioning is used to constraint the movement of modules rather than reduce problem size

By performing partitioning, we can iteratively impose a new set of constraints on the global optimization problem

- Assign modules to a particular block

Partitioning is determined by

- Results of global placement – initial starting point
  - Spatial (x,y) distribution of modules
- Partitioning cost
  - Want a min-cut partition

Partitioning due to Global Optimization

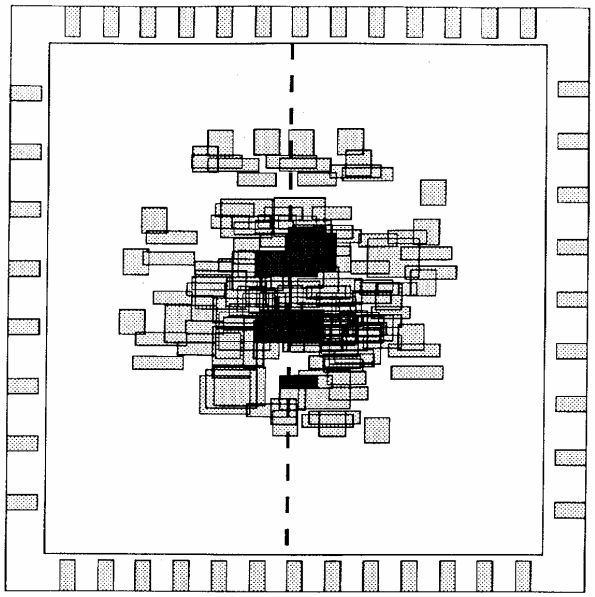
Sort the modules by their x coordinate (for a vertical cut)

Choose a cut line such that area is approximately equal between two sides of cut

$$M_p \rightarrow M_{p'}, M_{p''}$$

$$x_{u'} \leq x_{u''} \quad u' \in M_{p'}, u'' \in M_{p''}$$

$$\alpha = \sum_{u' \in M_{p'}} F_u \Big/ \sum_{u \in M_p} F_u \approx 0.5$$



Partitioning Improvement - I

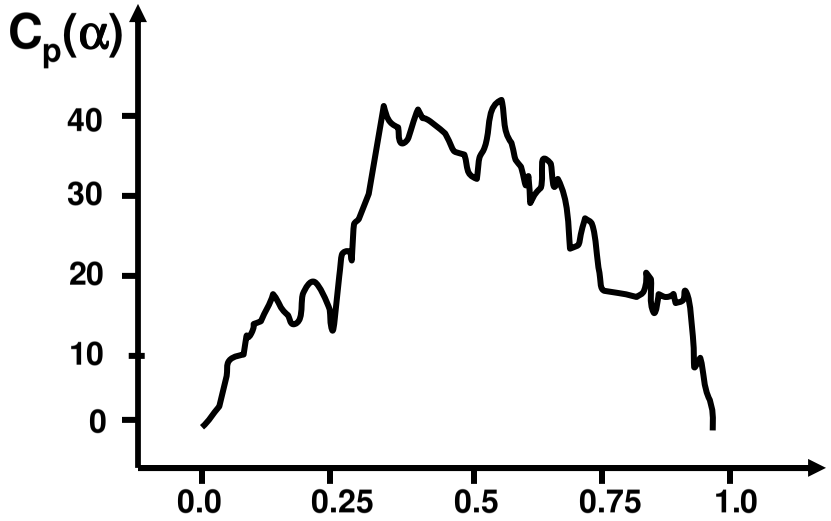
- The cost of initial partition may be too high
- Can change position of the cut to reduce the cost
- Plot the cost function, choose “best” position

$$M_p \rightarrow M_{p'}, M_{p''}$$

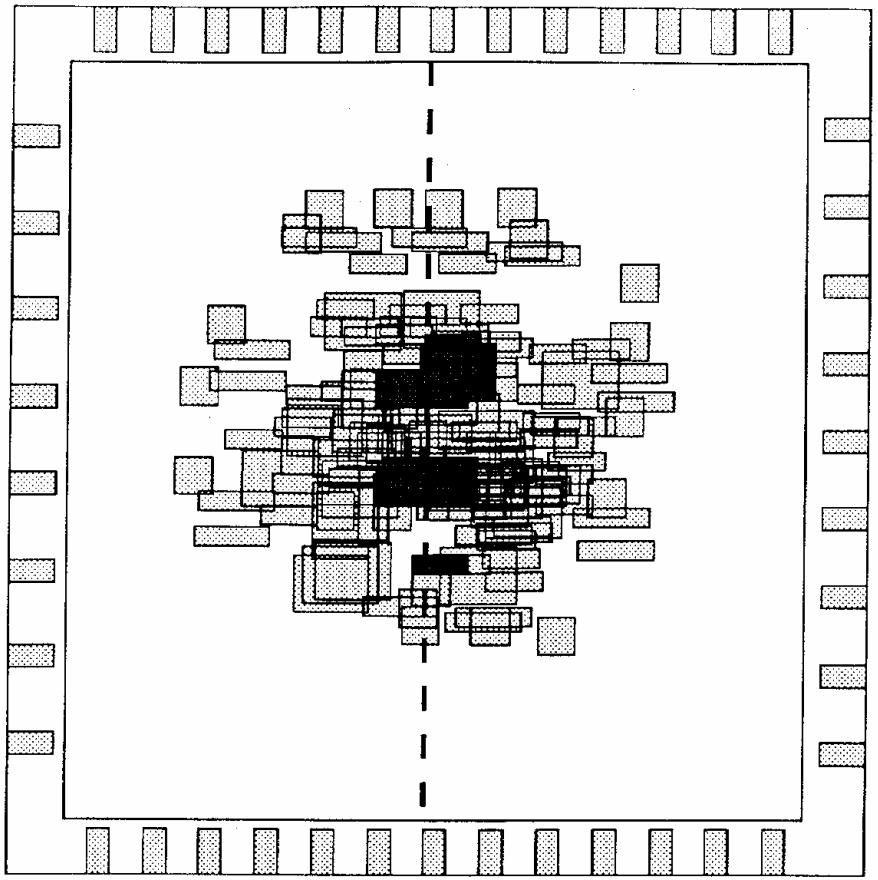
$$x_{u'} \leq x_{u''} \quad u' \in M_{p'}, u'' \in M_{p''}$$

$$\alpha = \sum_{u' \in M_{p'}} F_u \Big/ \sum_{u \in M_p} F_u \approx 0.5$$

$$\text{cut value: } C_p(\alpha) = \sum_{v \in Nc} w_v$$



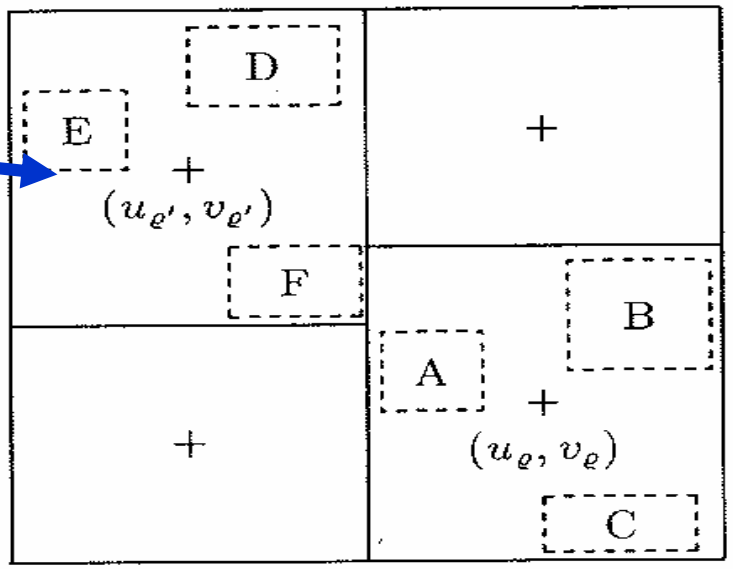
Layout after Min-cut



Now global placement problem will be solved again with two additional center_of_gravity constraints

Adding Positioning Constraints

- Partitioning gives us two new "center of gravity" constraints
- Simply update constraint matrix
- Still a single global optimization problem
- Partitioning is not "absolute"
  - modules can migrate back during optimization
  - may need to re-partition



$$\mathbf{A}^{(l)} = \begin{array}{c} \\ \vdots \\ \varrho \\ \varrho' \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} & \mathrm{E} & \mathrm{F} & \cdots \end{array} \\ \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & * & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & * & * & * & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \end{array}$$

Fig. 4. The constraints for global placement.

Continue to Iterate

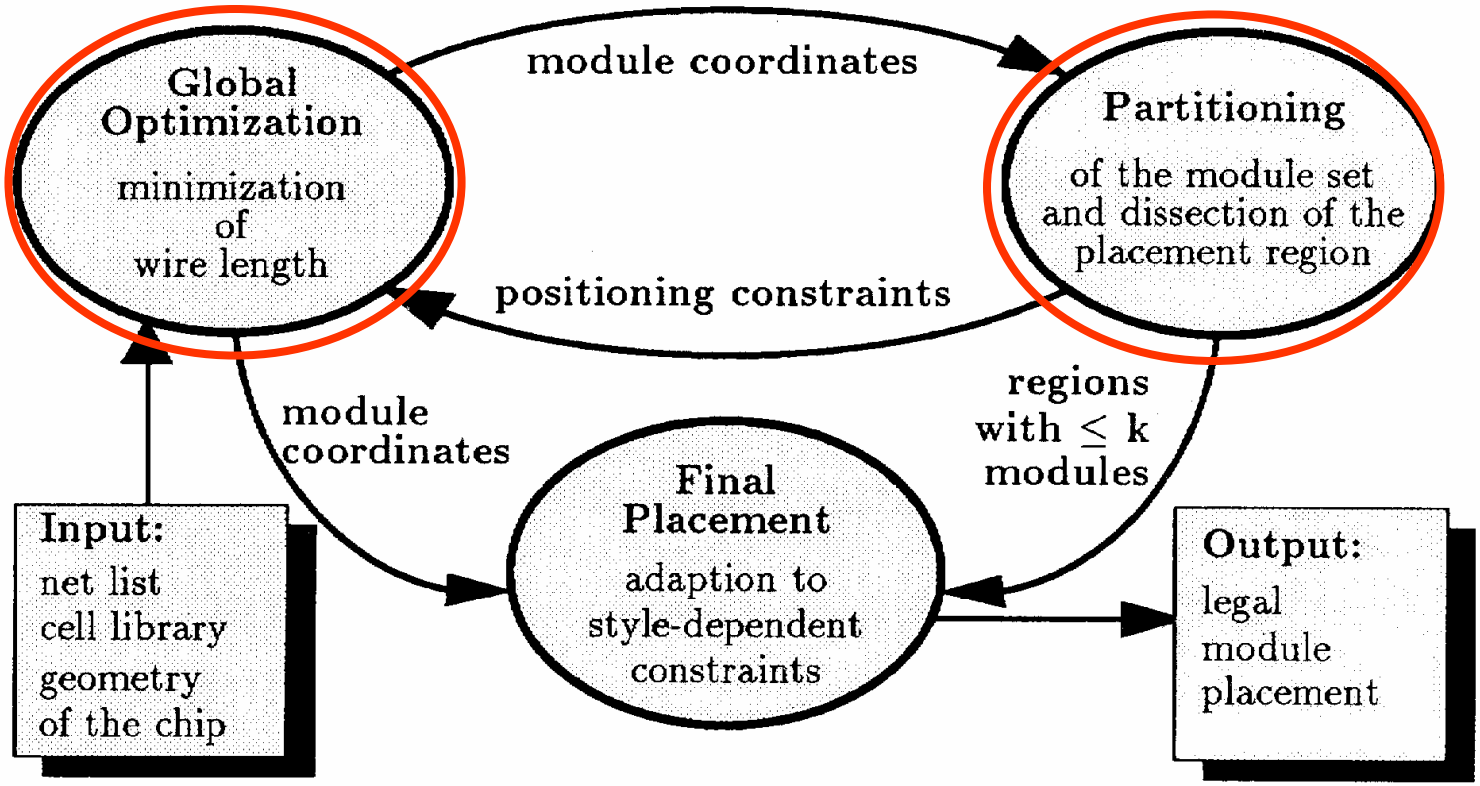
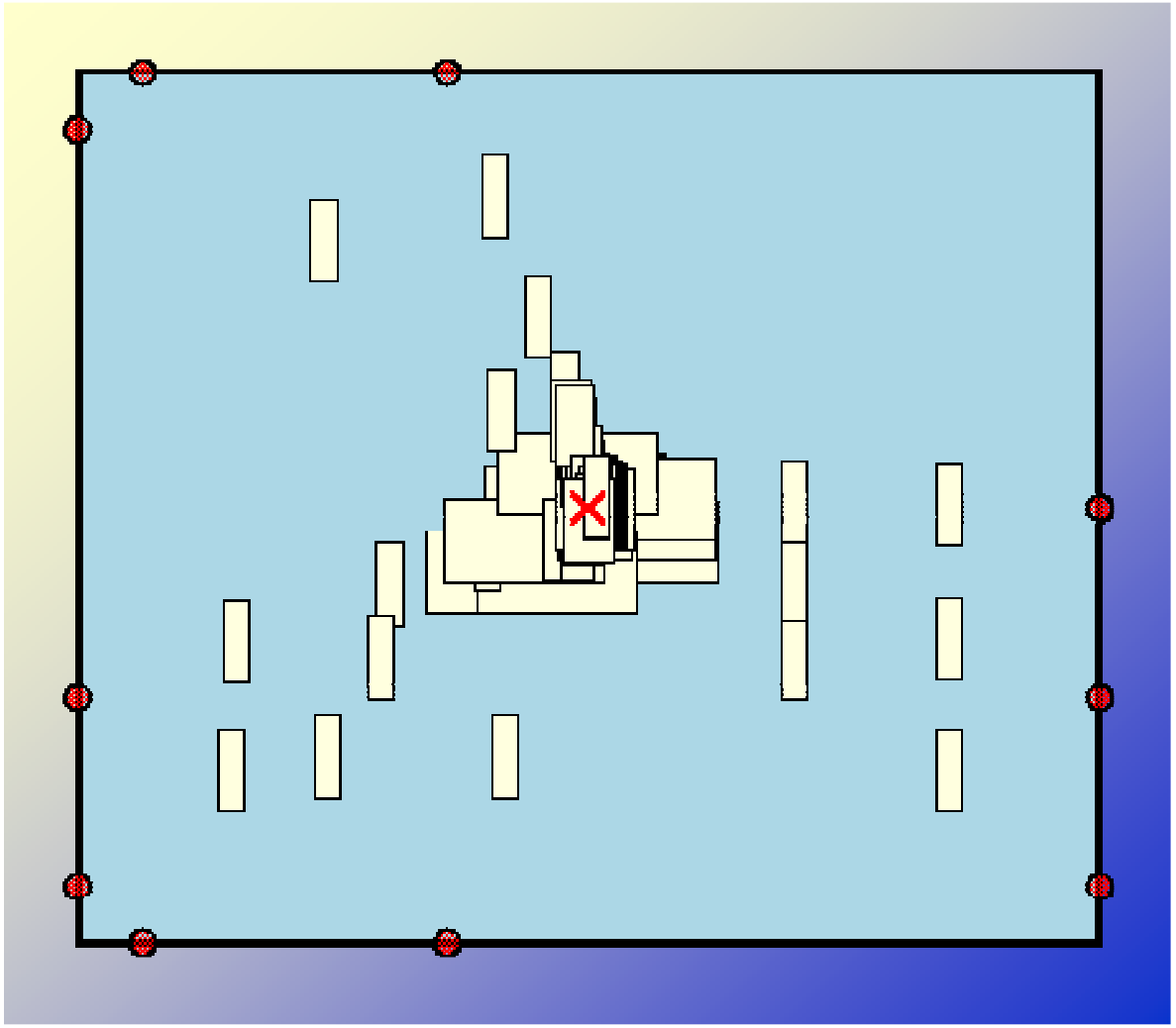


Fig. 1. Data flow in the placement procedure GORDIAN.

GORDIAN

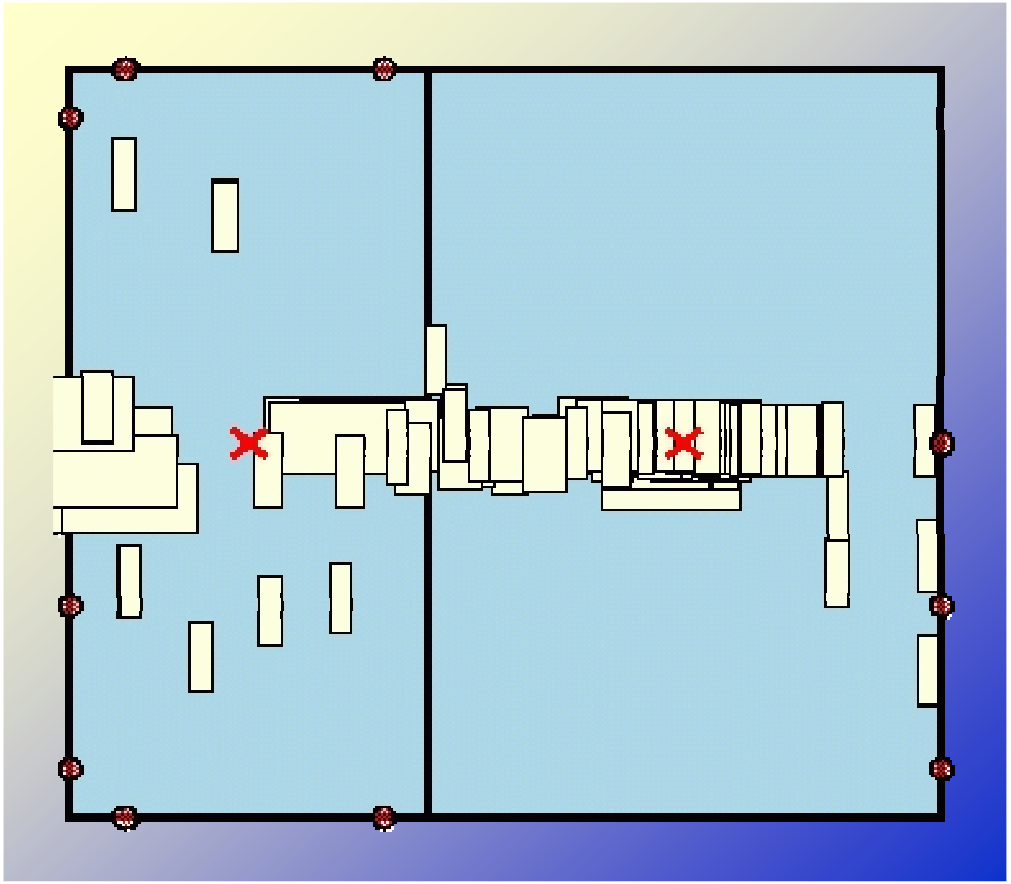
```
Procedure Gordian
  l:=1;
  global-optimize(l);
  while(∃|M_l|>k)
          for each ρ∈R(l)
            partition(ρ,ρ',ρ");
          endfor
          l:=l+1;
          setup-constraints(l);
          global-optimize(l);
          /* extras
          repartition(l); */
  endwhile
  final-placement(l);
endprocedure
```

First Iteration



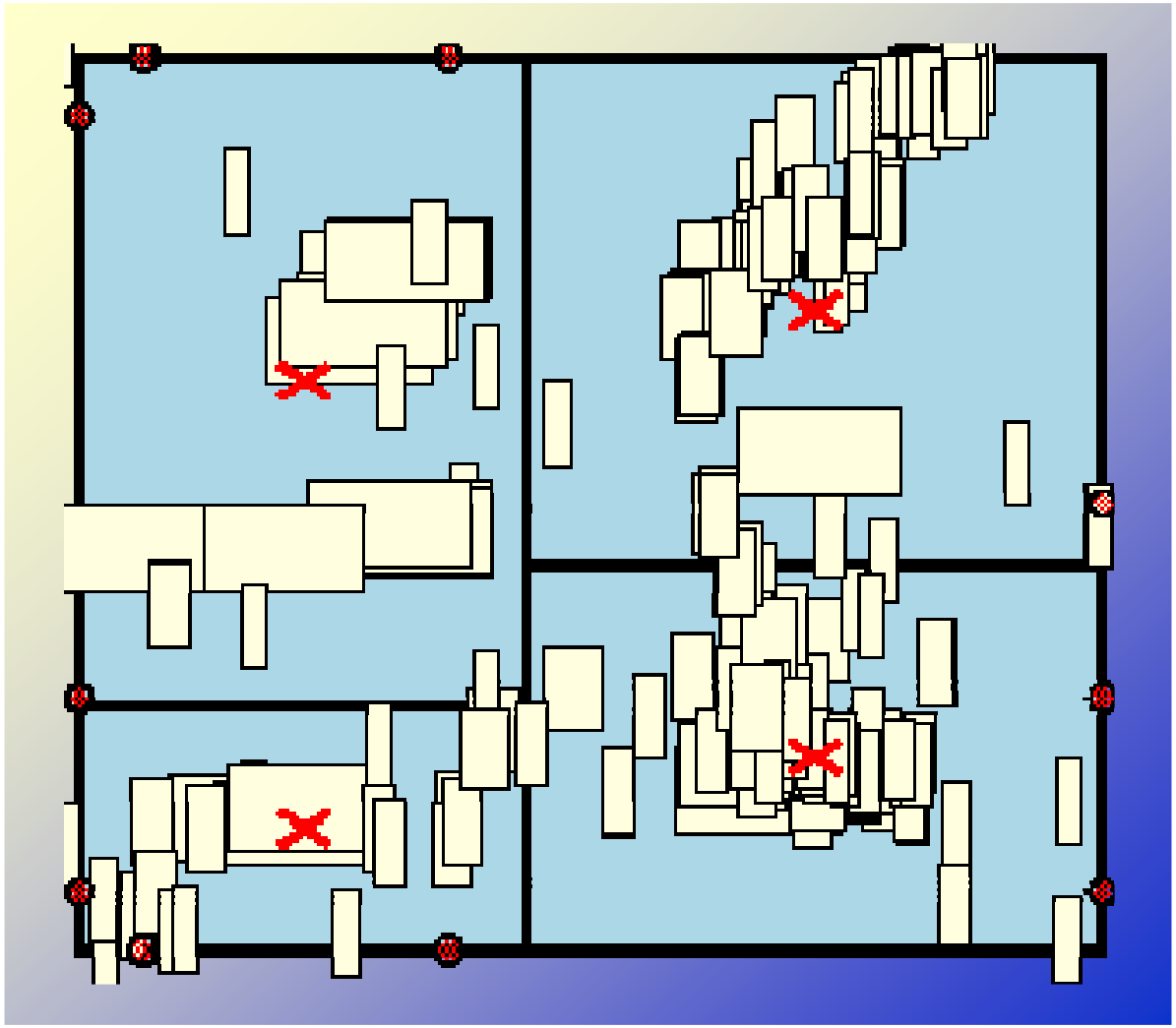
A. Kahng

Second Iteration



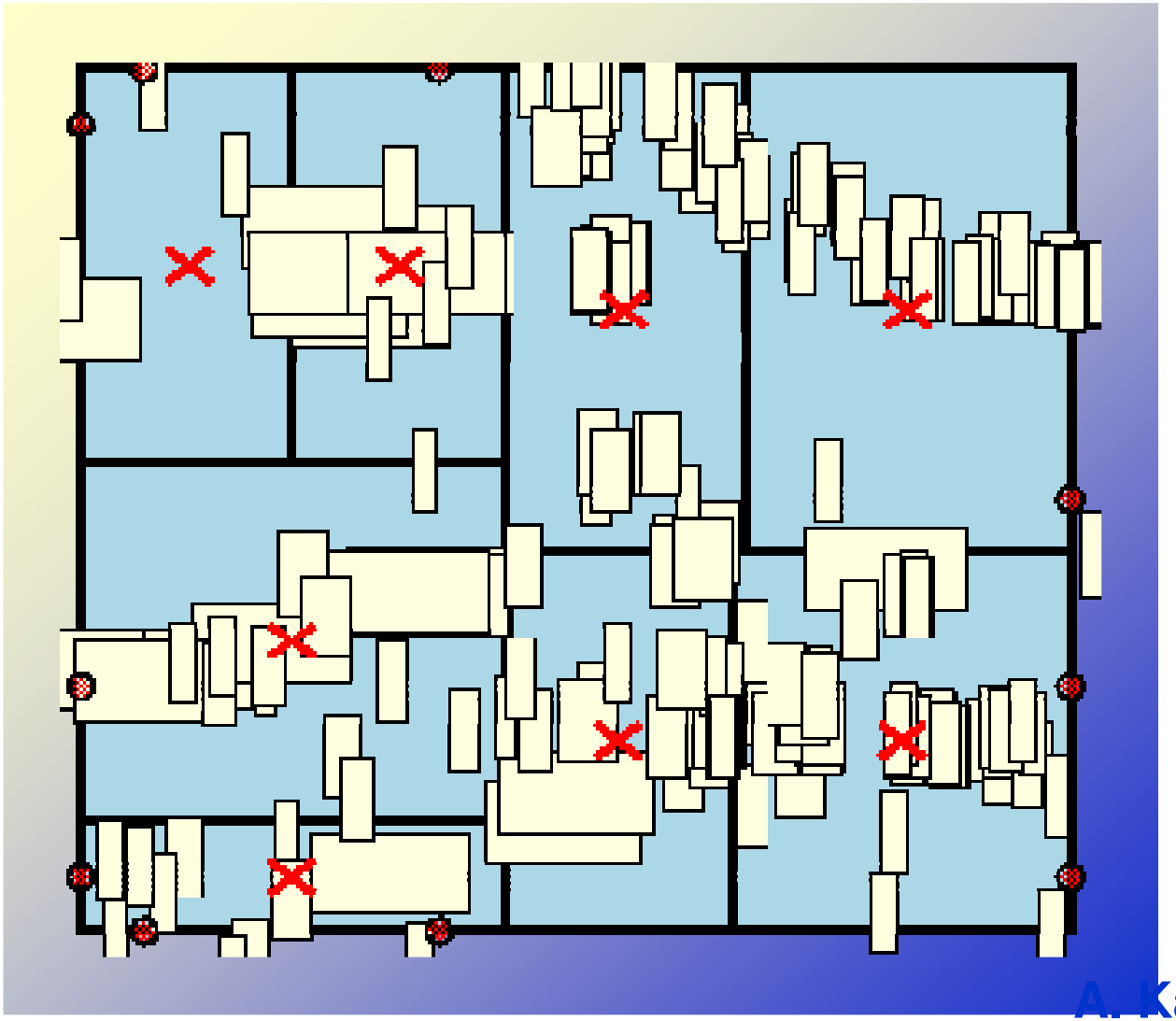
A. Kahng

Third Iteration



A. Kahng

Fourth Iteration

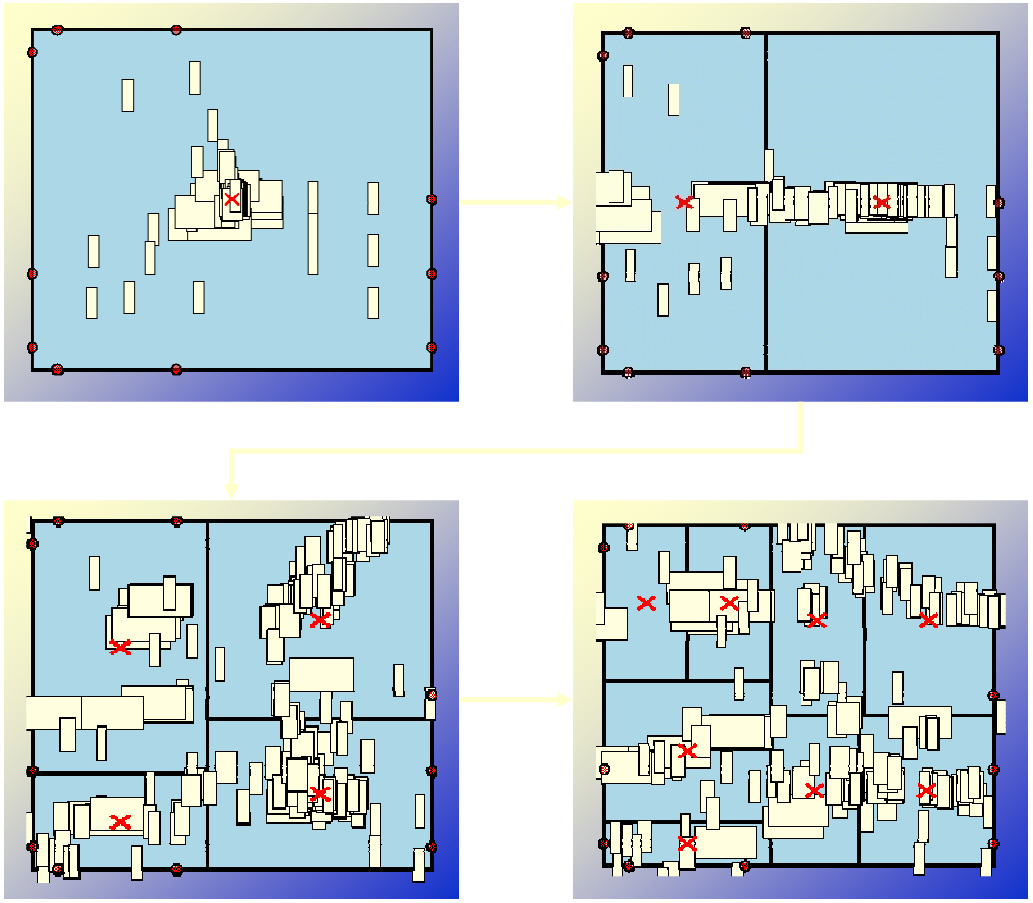


A. Kahng

GORDIAN (quadratic + partitioning)

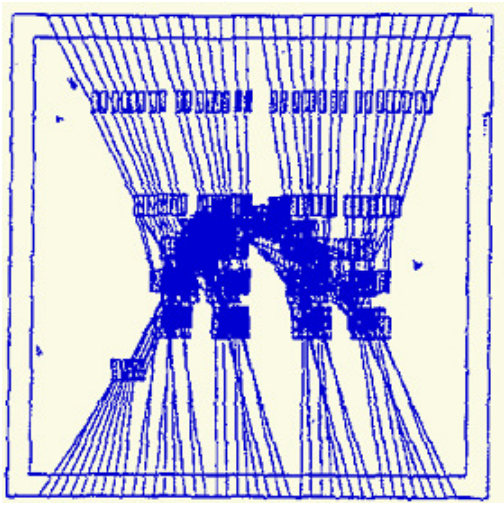
Initial
Placement

Partition
and Replace

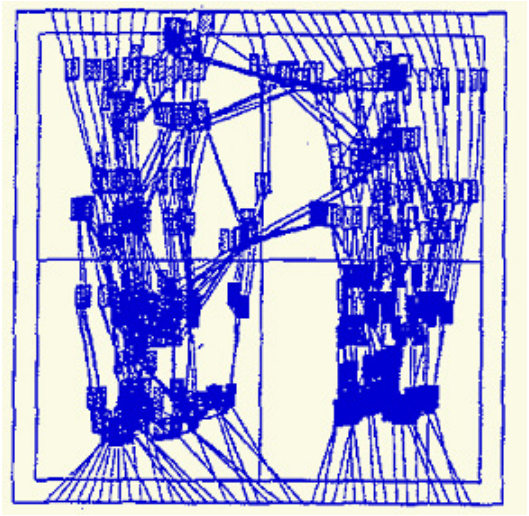


A. Kahng

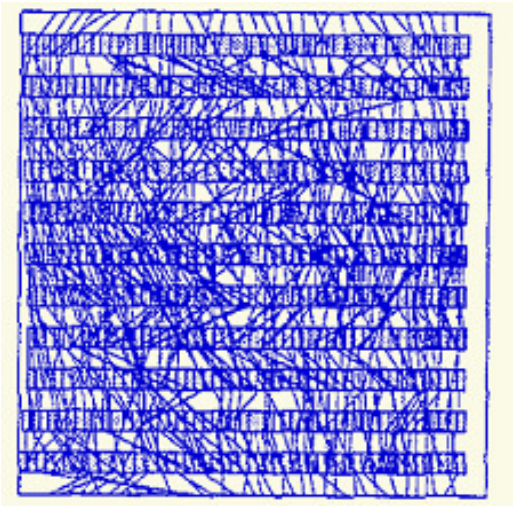
Another Series of Gordian



(a) Global placement with 1 region



(b) Global placement with 4 region



(c) Final placements

D. Pan – U of Texas

Final Placement

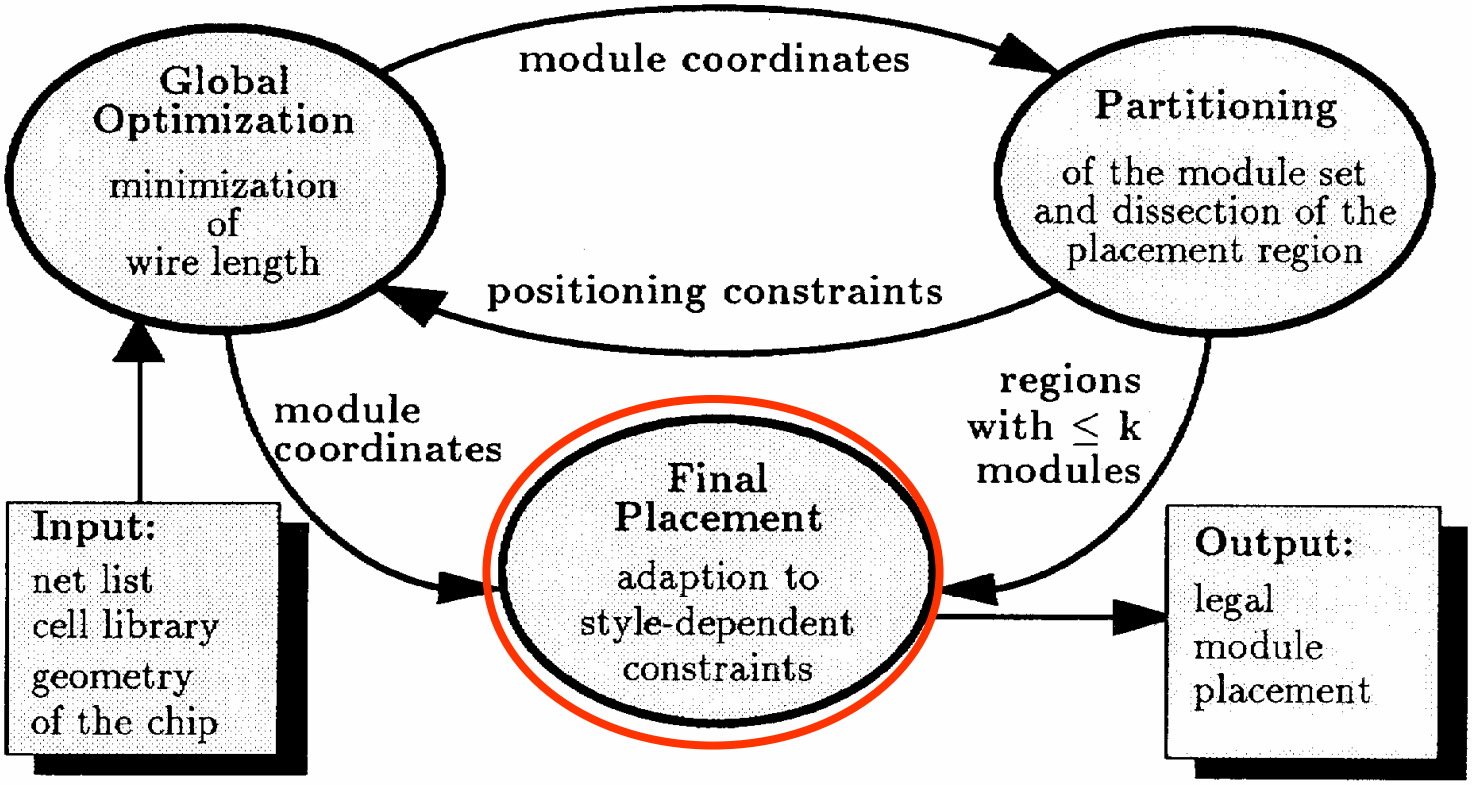


Fig. 1. Data flow in the placement procedure GORDIAN.

Final Placement - 1

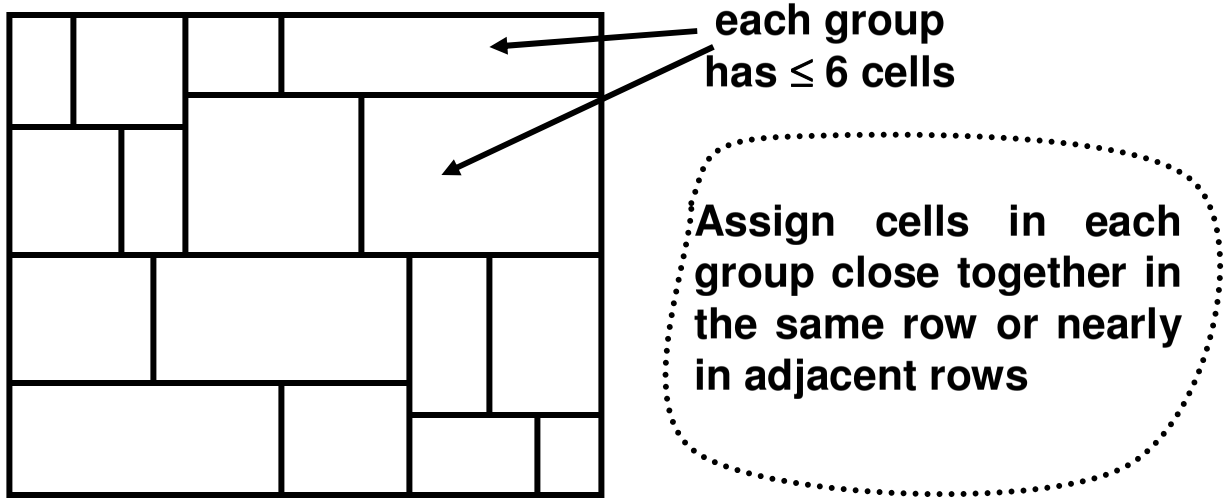
Earlier steps have broken down the problem into a manageable number of objects

Two approaches:

- Final placement for standard cells/gate array – row assignment
- Final placement for large, irregularly sized macro-blocks – slicing – look over in "Extra" slides at the end

Final Placement – Standard Cell Designs

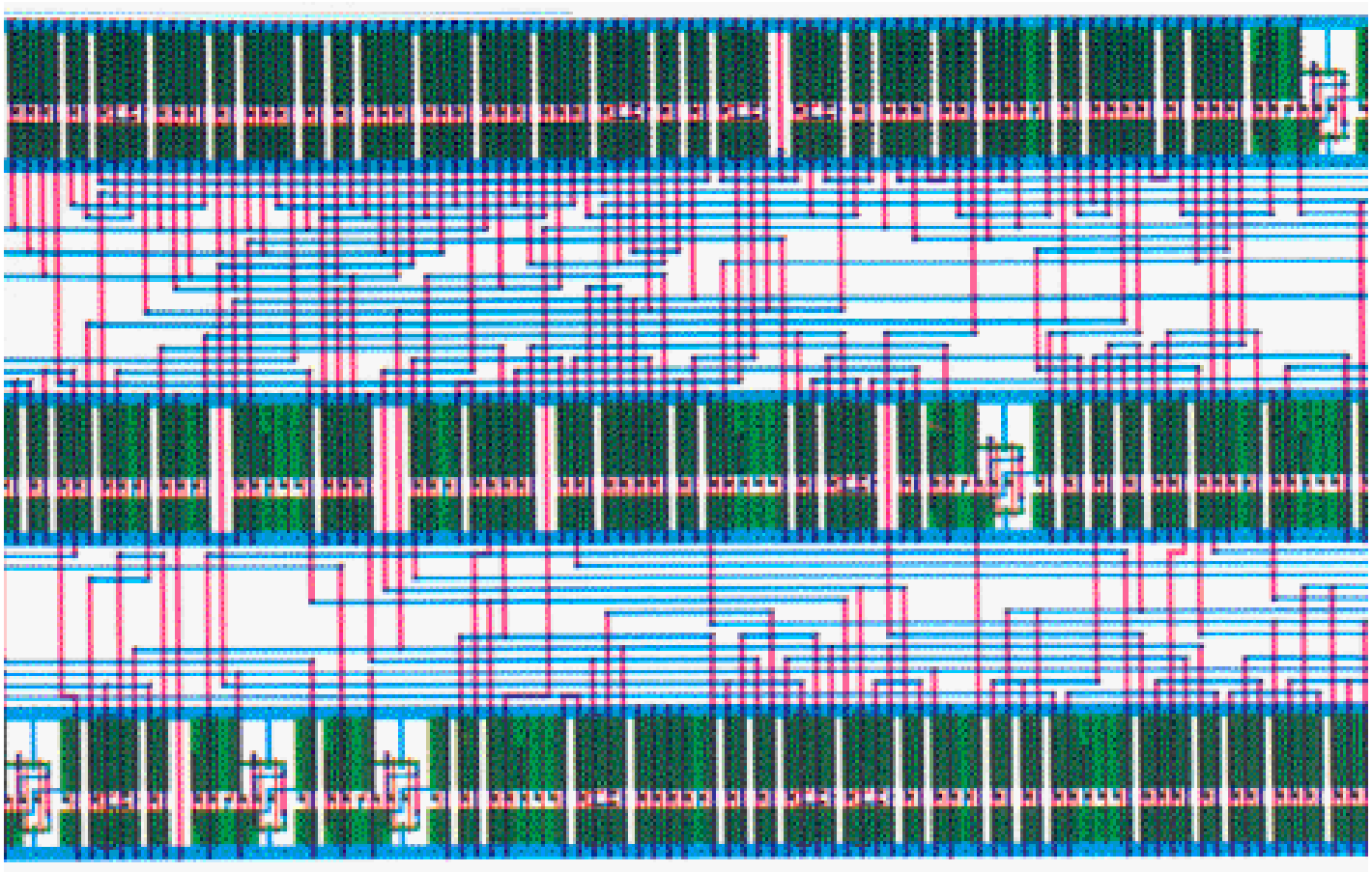
This process continues until there are only a few cells in each group(≈ 6)



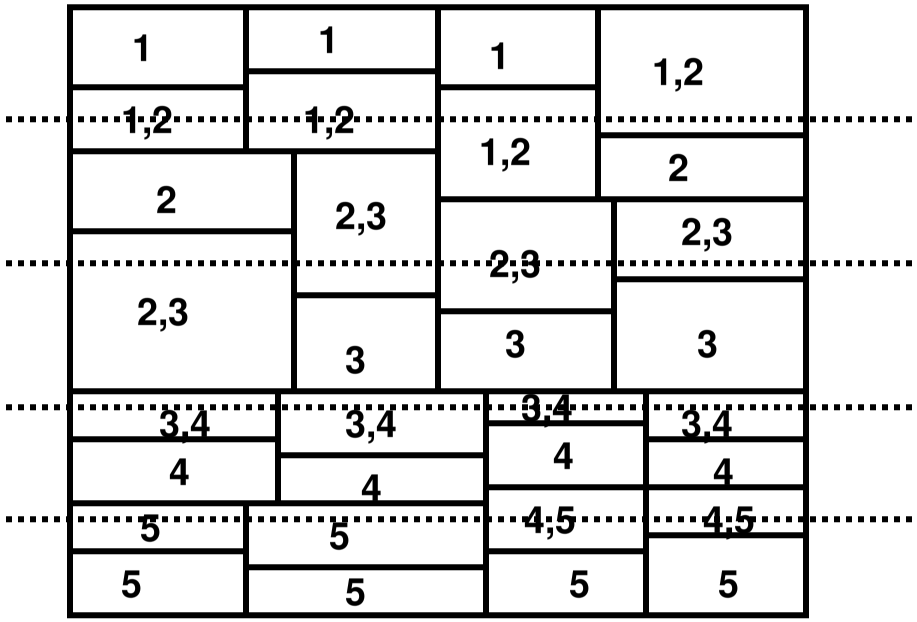
group: smallest partition

A. E. Dunlop, B. W. Kernighan,
A procedure for placement of standard-cell VLSI circuits, IEEE Trans. on CAD, Vol. CAD-4, Jan , 1985, pp. 92- 98

Standard Cell Layout



Final Placement – Creating Rows



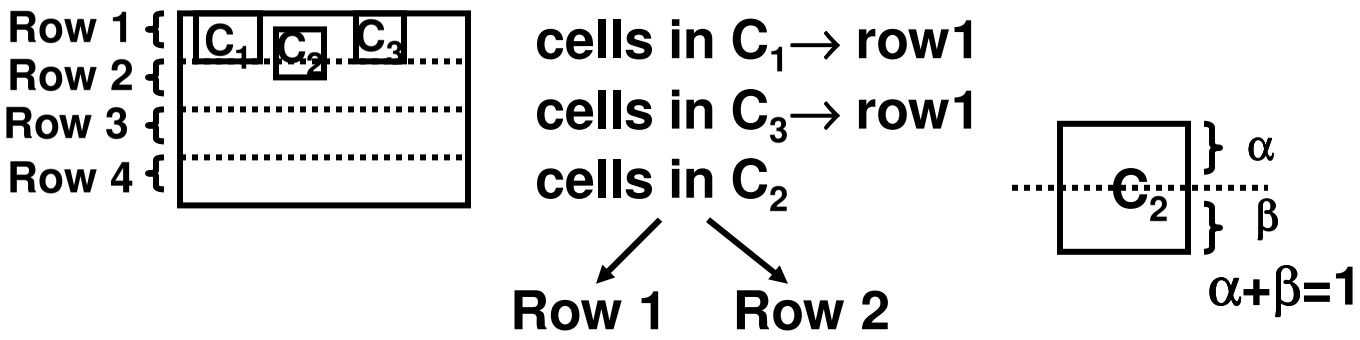
a four-row standard cell design

Partitioning of circuit into 32 groups. Each group is either assigned to a single row or divided into 2 rows

Final Placement – Creating Rows

- **Partitioning must be done breadth-first not depth first**

Creating Rows



Choose α and β preferably to balance row to balance row length (During re-arrangement)

Final Placement

Earlier steps have broken down the problem into a manageable number of objects

Two approaches:

- Final placement for standard cells – row assignment
- Final placement for large, irregularly sized macro-blocks – slicing – see Extras

Generating Final Placement

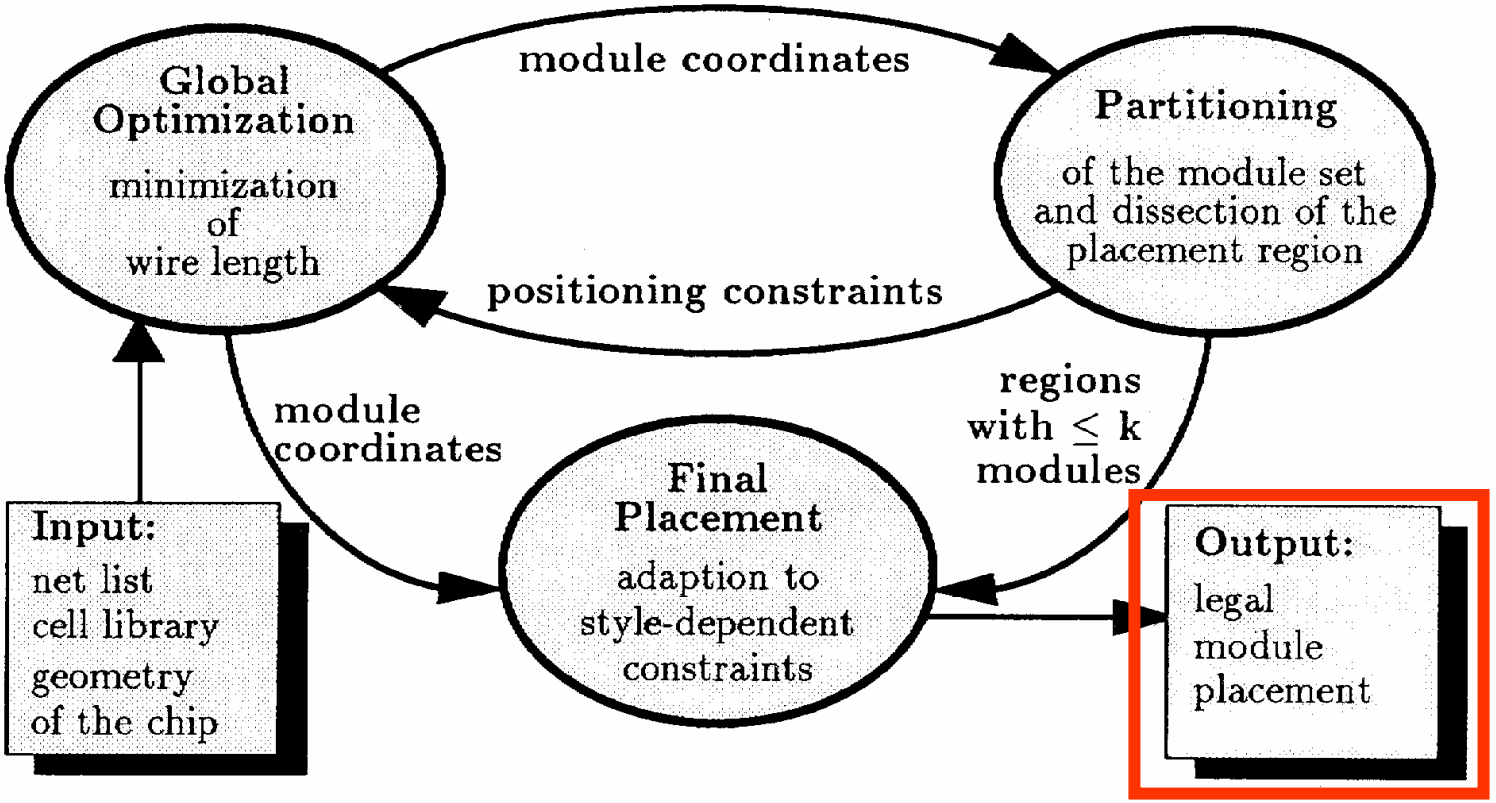


Fig. 1. Data flow in the placement procedure GORDIAN.

Example Final Placement

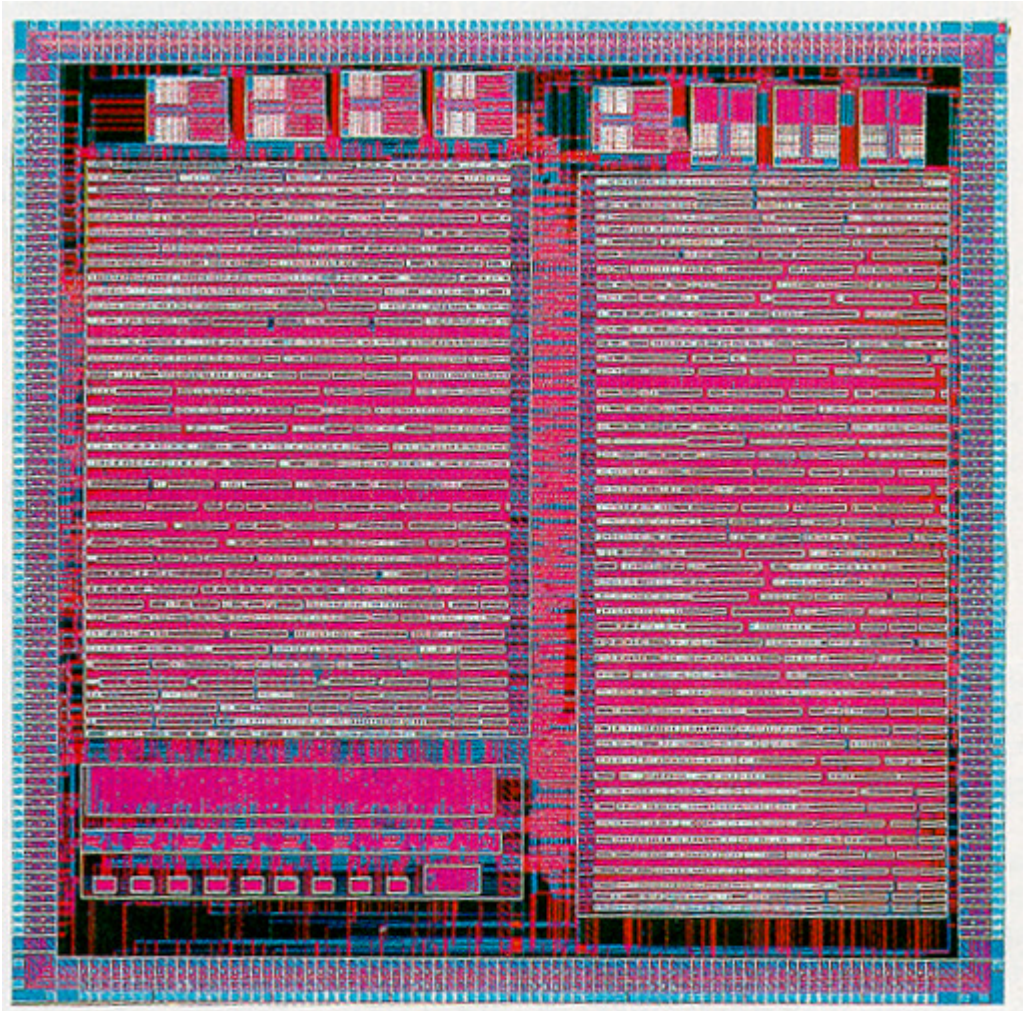


Fig. 10. Macrocell design with standard cell blocks *scb8* and *scb9*.

Summary of status on placement

Most place and route tools use a series of ``half-truths'' regarding timing modeling and delay constraints

Not uncharacteristically, the Gordian approach uses three different approaches to attack the problem - each approach makes a number of different simplifications

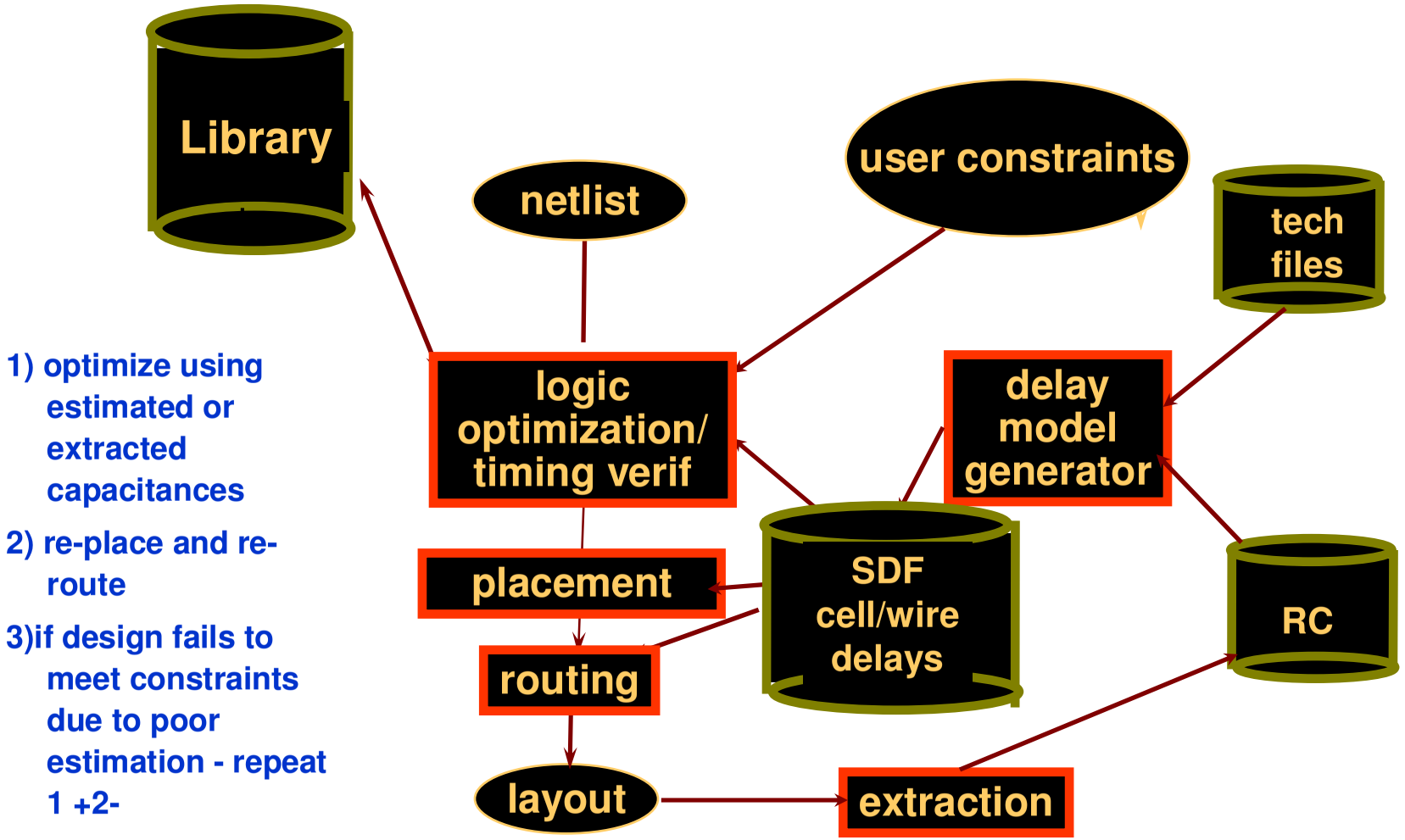
Current gate-level placement tools (e.g. Apollo) are able to place and route hundreds of thousands → millions of cells flat (without hierarchy)

Despite the lack of direct correlation between the internal timing model and the actual integrated circuit timing - timing closure between synthesis and physical design is improving but still a problem

Principally due to:

- **Better delay *estimation* in synthesis**
- **Better delay *calculation* in physical design**
- **Automated buffer insertion in routing**

Today's high-perf logical/physical flow

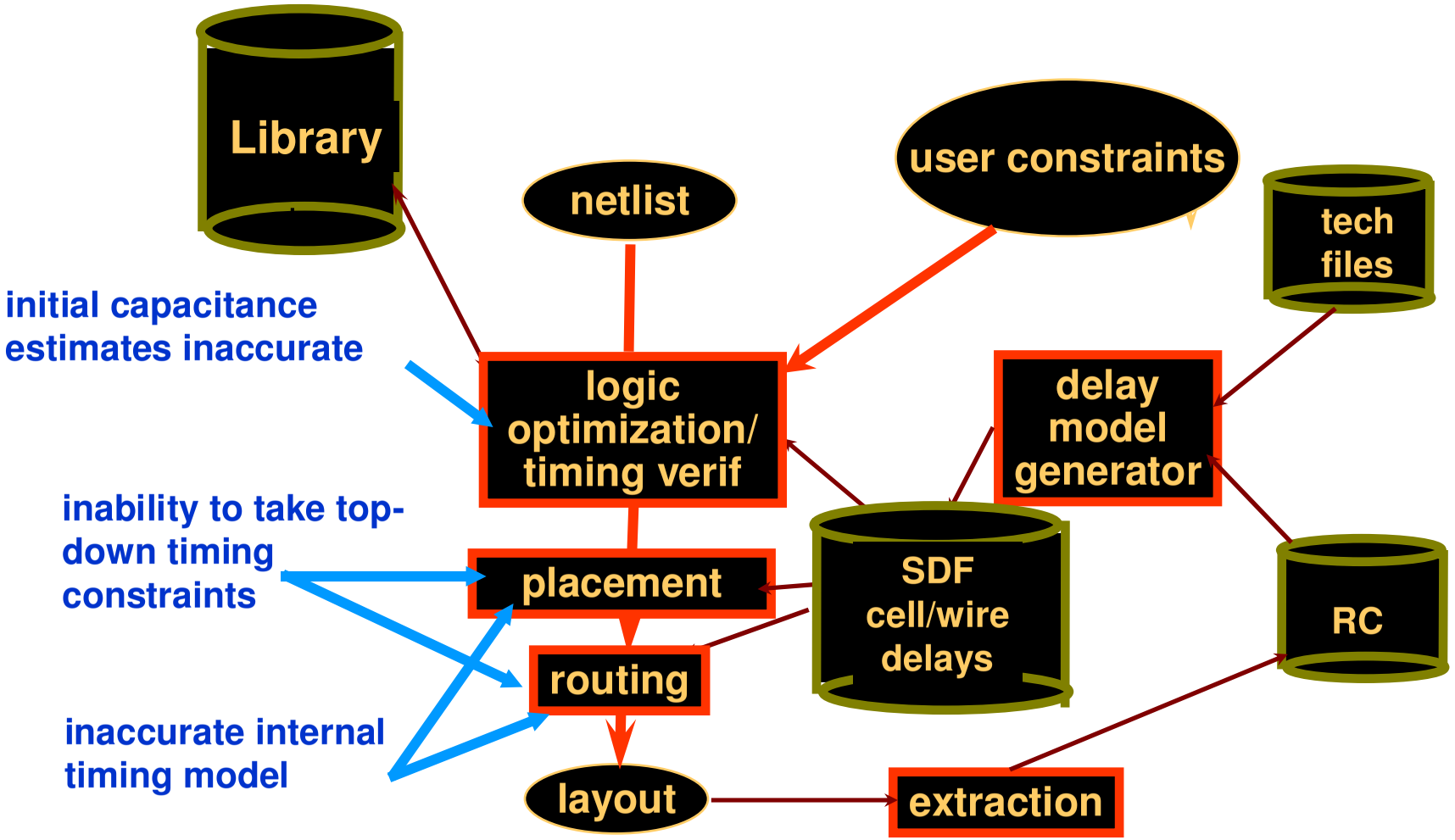


1) optimize using estimated or extracted capacitances

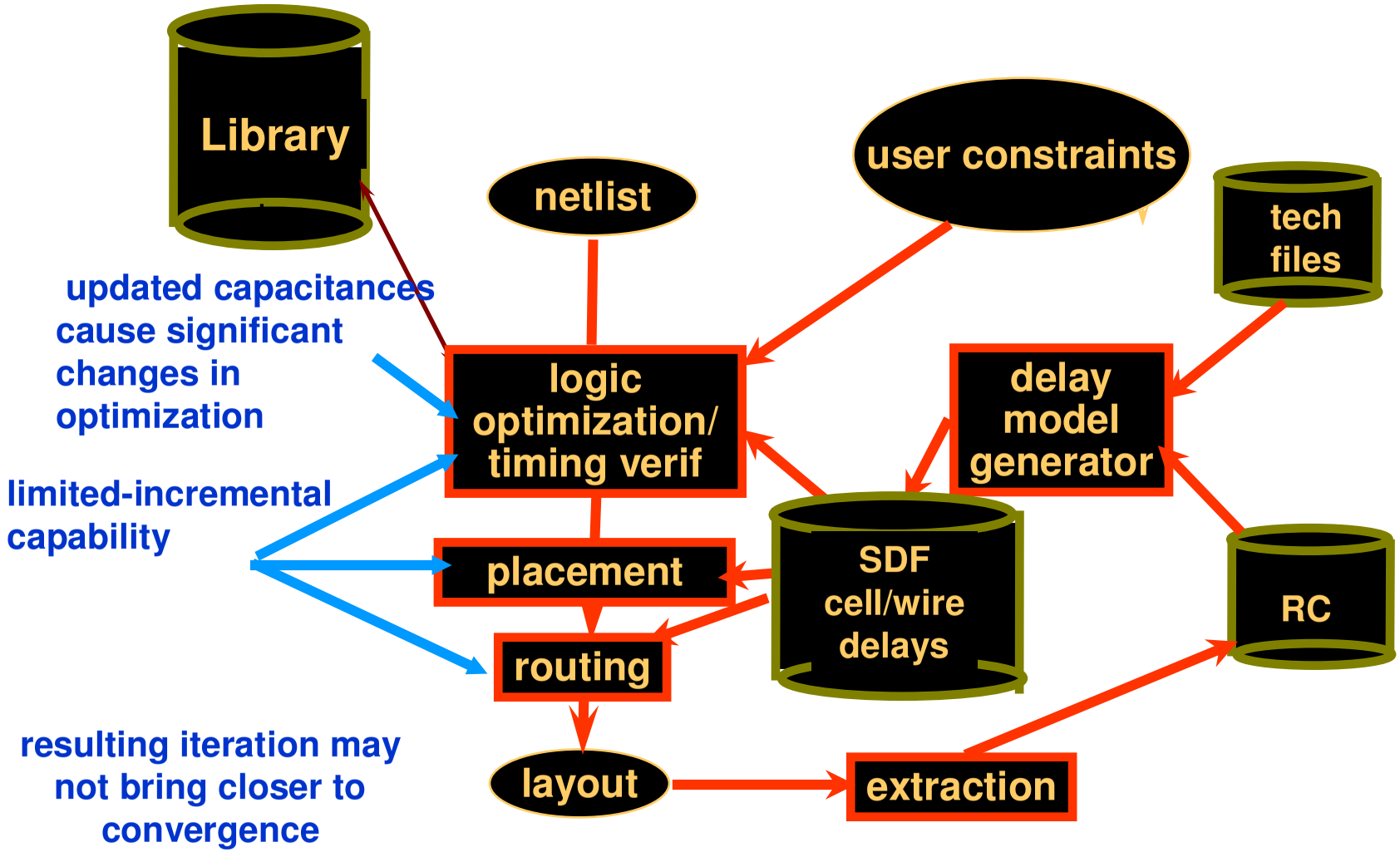
2) re-place and re-route

3)if design fails to meet constraints due to poor estimation - repeat 1 +2-

Top-down problems in the flow



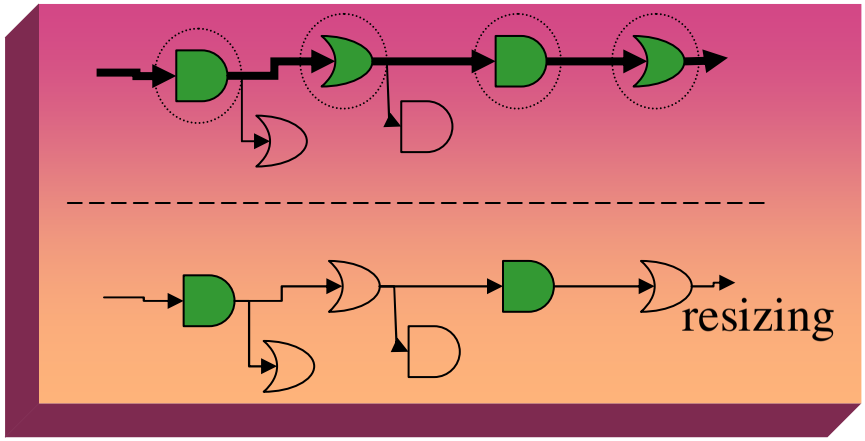
Iteration problems in the flow

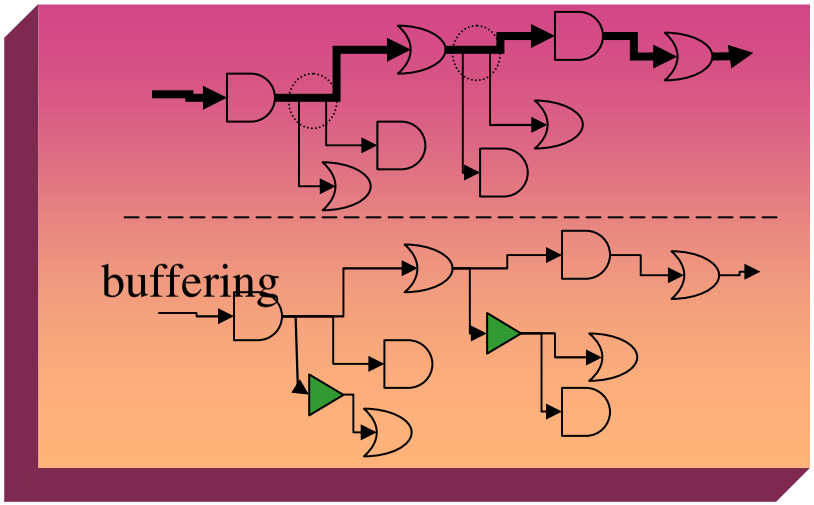


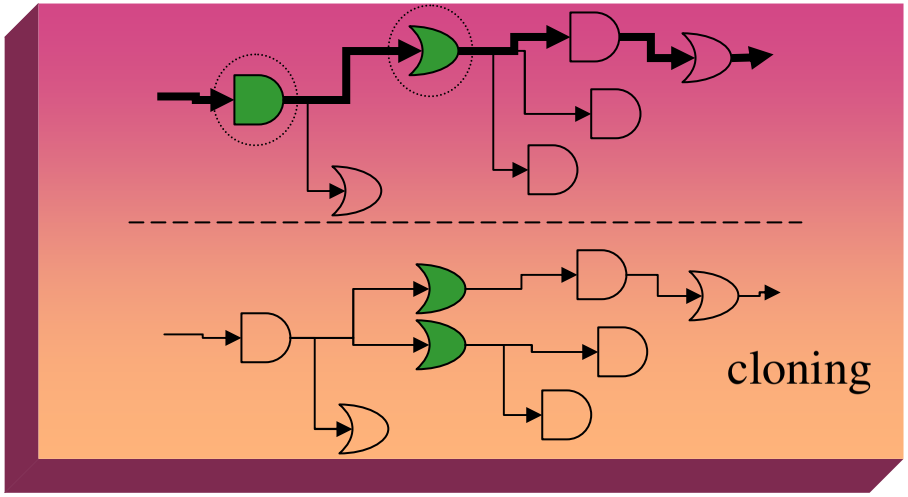
Research opportunities in placement

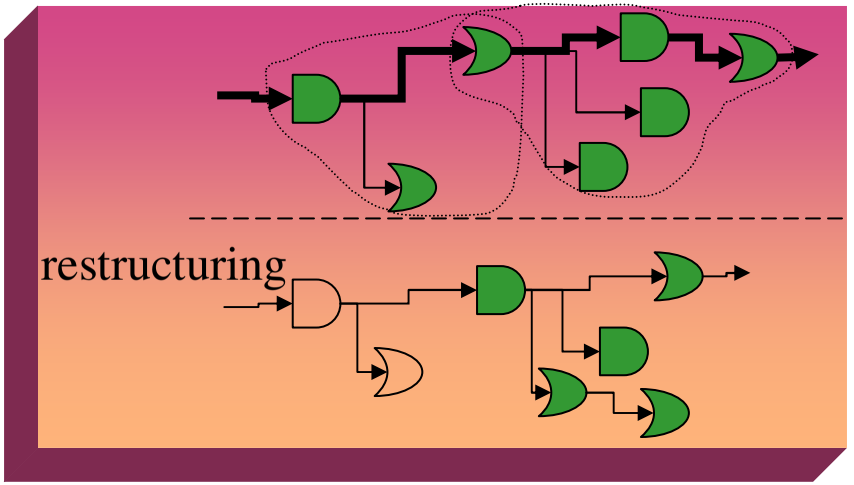
- **Better delay *estimation and modeling* in synthesis**
- **Better delay *calculation* in physical design**
- **Automated resynthesis during placement**
- **Automated sizing and buffer insertion in routing**

Integrating Synthesis and Placement



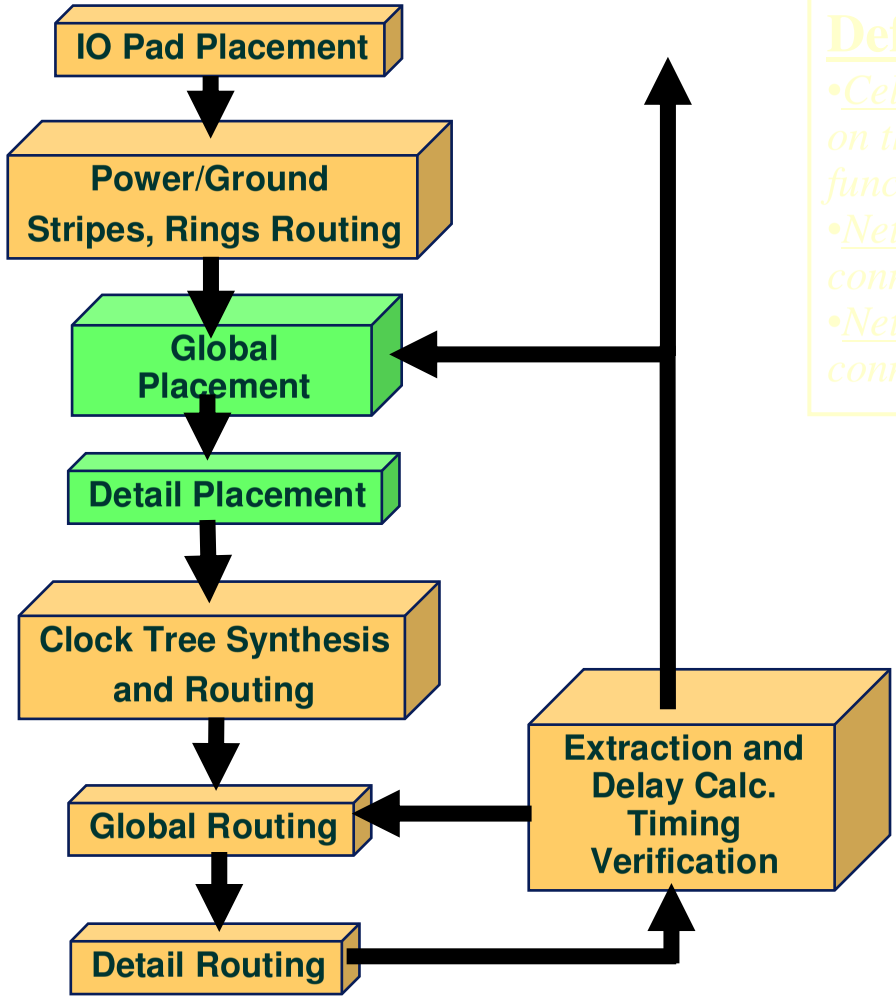






Stan Chow Ammocore
Andrew B. Kahng UCSD
Majid Sarrafzadeh UCLA

Other Elements of Industrial Flow



Definitions:

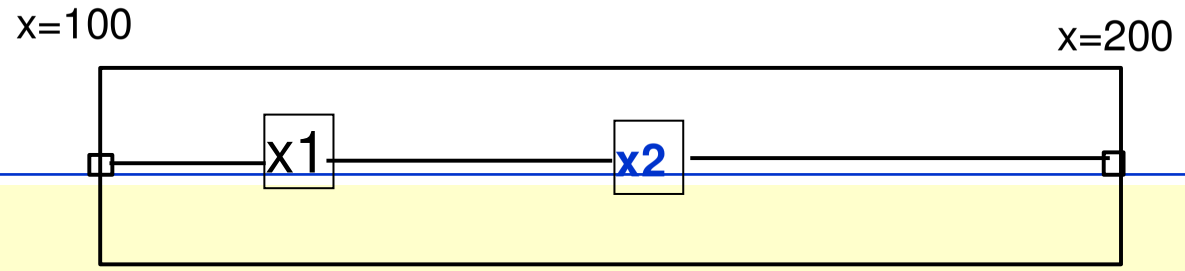
- *Cell: a circuit component to be placed on the chip area. In placement, the functionality of the component is ignored.*
- *Net: specifying a subset of terminals, to connect several cells.*
- *Netlist: a set of nets which contains the connectivity information of the circuit.*

A. Kahng

Extras

- Worked out example for quadratic programming – D. Pan, U of Texas
- Brief survey of approaches
- More on quadratic placement
- Using slicing to handle macro-blocks

Toy Example:



$$Cost = (x_1 - 100)^2 + (x_1 - x_2)^2 + (x_2 - 200)^2$$

$$\frac{\partial}{\partial x_1} Cost = 2(x_1 - 100) + 2(x_1 - x_2)$$

$$\frac{\partial}{\partial x_2} Cost = -2(x_1 - x_2) + 2(x_2 - 200)$$

setting the partial derivatives = 0 we solve for the minimum Cost:

Ax + B = 0

$$\begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -200 \\ -400 \end{bmatrix} = 0$$

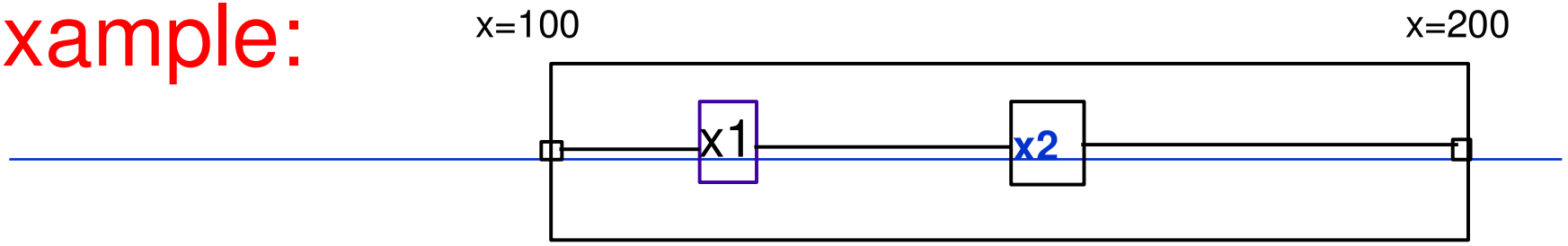
$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -100 \\ -200 \end{bmatrix} = 0$$

x1=400/3 x2=500/3

Kurt Keutzer

D. Pan

Example:



setting the partial derivatives = 0 we solve for the minimum Cost:

$$Ax + B = 0$$

$$\begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -200 \\ -400 \end{bmatrix} = 0$$

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -100 \\ -200 \end{bmatrix} = 0$$

x1=400/3 x2=500/3

Interpretation of matrices A and B:

The diagonal values A[i,i] correspond to the number of connections to xi
The off diagonal values A[i,j] are -1 if object i is connected to object j, 0 otherwise
The values B[i] correspond to the sum of the locations of fixed objects connected to object i

Traditional Approaches

- **Quadratic Placement**
- **Simulated Annealing**
- **Bi-Partitioning**
- **Quadrisection**
- **Force Directed Placement**
- **Hybrid**

Overview of Gordian Package

GORDIAN with repartitioning procedure

```
Procedure Gordian
  l:=1;
  global-optimize(l);
  while(∃|M_l|>k)
        for each ρ∈R(l)
          partition(ρ,ρ',ρ");
        endfor
        l:=l+1;
        setup-constraints(l);
        global-optimize(l);
        repartition(l);
  endwhile
  final-placement(l);
endprocedure
```

Cost Function

Overall objective

$$\phi = \frac{1}{2} \sum_{v \in N} L_v \, w_v$$

$$\phi\ (x, y) = \mathrm{X}^T C X + d_x^T X + Y^T C Y + d_y^T Y$$

$$\phi\ (x) = \mathrm{X}^T C X + d^T X \qquad \xi_{uv}$$

Global Placement and Constraints

- **The center of gravity constraints**

  At level l, chip is divided into q($\leq 2^l$) regions

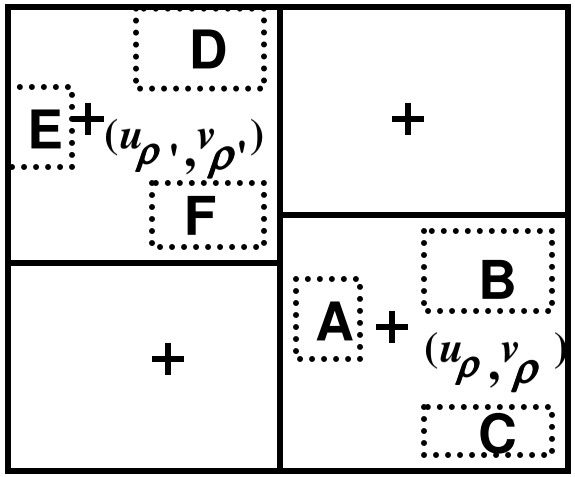
  For region p. the center coordinates: (u_p, v_p)

$$\textbf{constraints}: \quad \sum_{u \in M_p} F_u x_u = u_p \sum_{u \in M_p} F_u$$

(Mp: set of modules in region p)
Matrix form for all regions

$$A^{(l)} X = u^{(l)}, \qquad a_{pu} = \begin{cases} F_i \Big/ \sum_{i \in M_p} F_i & \text{If } i \in M_p \\ 0 & \text{otherwise} \end{cases}$$

Problem Formulation



$$A^{(l)} = \begin{array}{c} \\ \vdots \\ \rho \\ \rho' \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & * & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & * & * & * & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \end{array}$$

- **Linearly constrained quadratic programming problem**

$$\text{LQP}: \min_{x \in R^m} \{ \Phi(x) = X^T C X + d^T X \quad \text{s.t.} A^{(l)} X = u^{(l)} \}$$

Solution Method

$$A_{q\times m}=[D_{q\times q}B_{q\times(m-q)}]$$

$$[D,B]\begin{bmatrix}X_d\\X_i\end{bmatrix}=u$$

X_d ***dependent variables***

X_i ***independent variables***

$$X_d=-D^{-1}BX_i+D^{-1}u$$

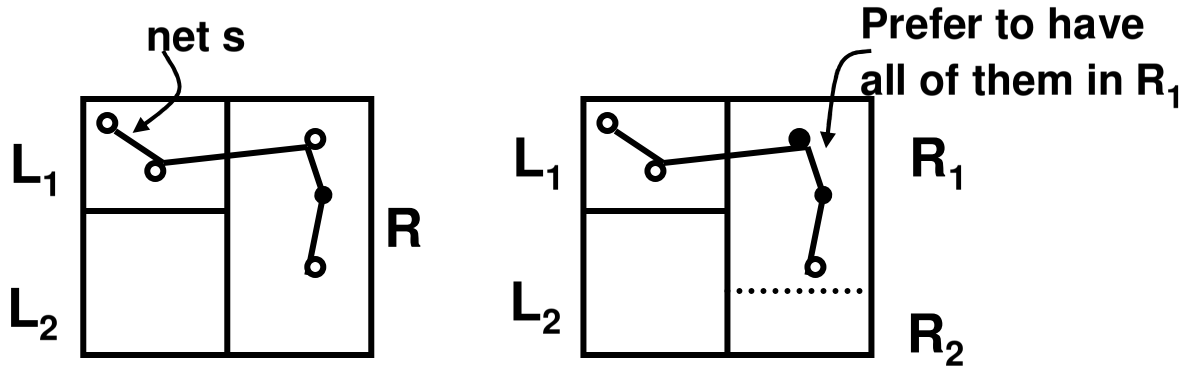
$$X=\begin{bmatrix}-D^{-1}B\\I\end{bmatrix}X_i+\begin{bmatrix}D^{-1}u\\0\end{bmatrix}=ZX+x_0$$

unconstrained quadratic programming problem

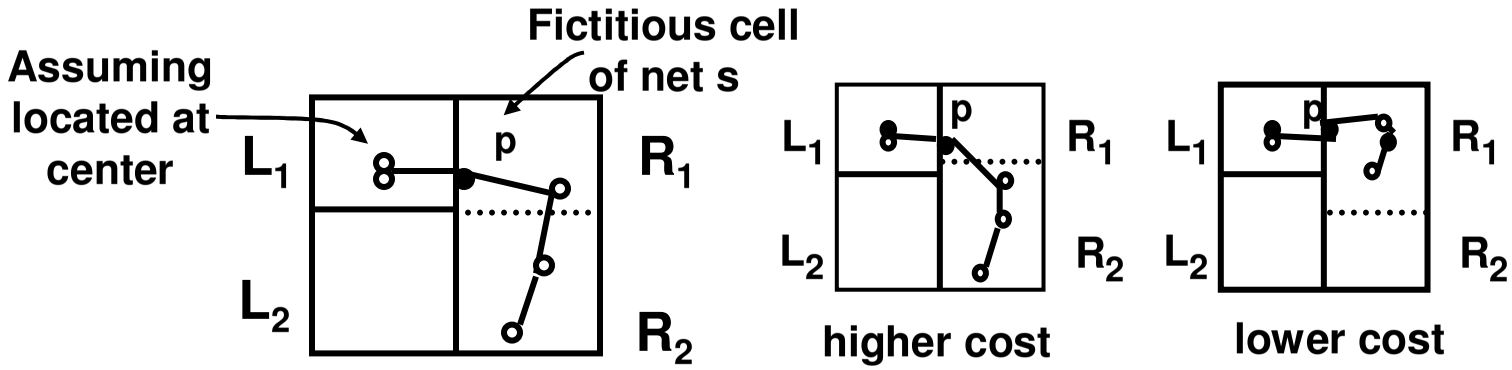
$$\mathbf{UQP:}\ \min_{x_i\in R^{m-q}}\{\psi(x_i)=X_i^{\mathrm{T}}Z^{\mathrm{T}}CZX+C^{\mathrm{T}}X_i\}\ (\mathrm{C}^{\mathrm{T}}=CX_0+d)$$

Solved by conjugate --gradient method

Terminal Propagation



- **We should use the fact that s is in L_1!**



p will stay in R_1 for the rest of partitioning

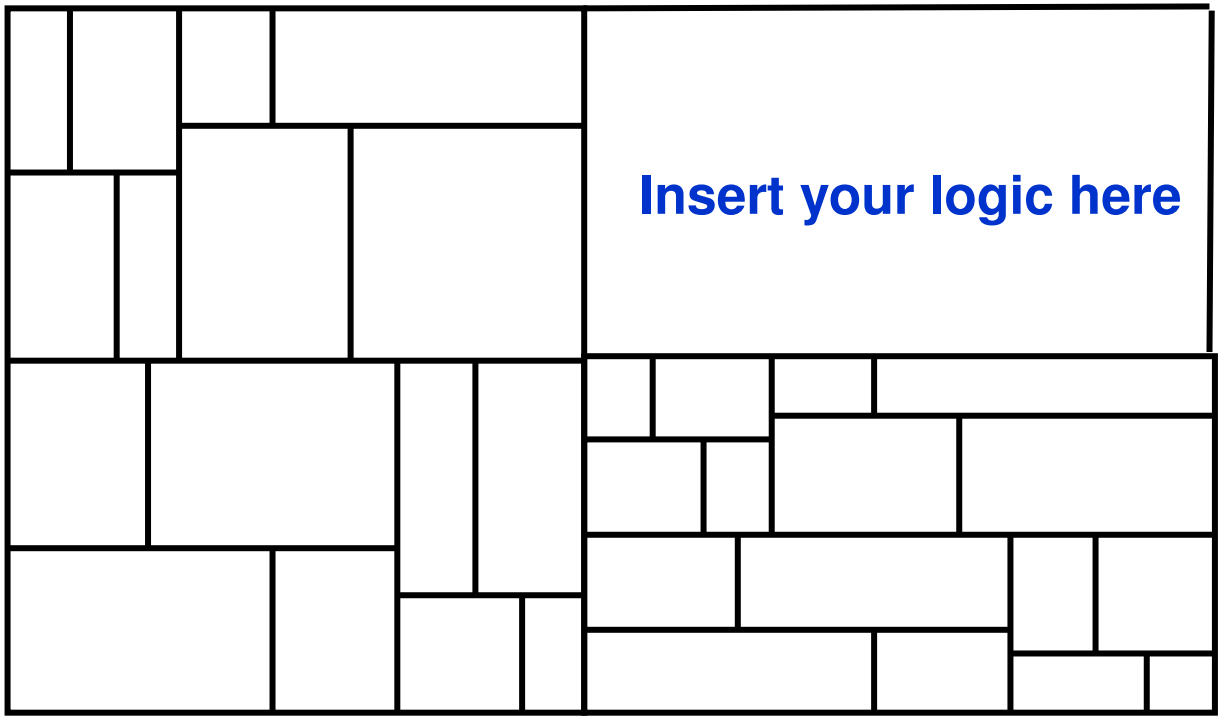
Macro Placement by Slicing

Constraining Placement of Macros

Global optimization and partitioning assigns <=k cells to each physical region

There are different ways of placing cells in each region

Want to choose placements that minimize total chip area



Macro-blocks: Exhaustive Slicing Optimization

L. van Ginnekin, R. H. Otten, *Optimal Slicing of Point Placements,* EDAC, 1990, pp. 322-326

procedure ESO

for all regions ϱ with $|\mathcal{M}_\varrho| \leq k$ modules

Determine the shape function of region ϱ by
enumeration of all slicing structures
that can be derived from the global
placement coordinates of the modules;

endfor

Recursively compute the shape function of the root region
bottom up from the shape functions of all ESO regions;

Select one shape for the root region;

Traverse the slicing tree top down
to determine the module coordinates and shapes;

endprocedure

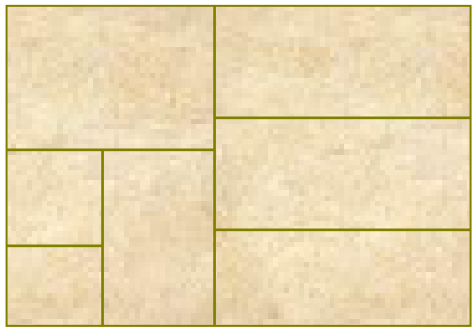
Fig. 8. Exhaustive slicing optimization procedure.

Slicing vs. Non-Slicing Floorplans

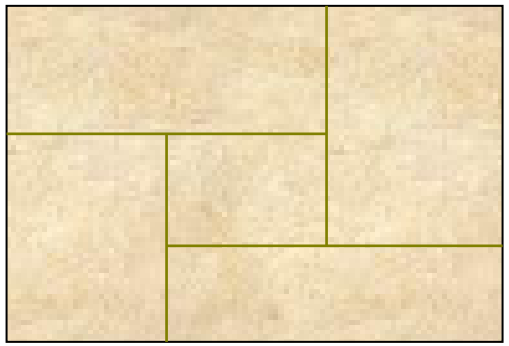
Slicing Floorplan **(slicing structure): A rectangle dissection which can be obtained by recursively dissecting the the base rectangle into smaller rectangles by vertical and horizontal slicing lines.**

Slicing structures are good for routing

Slicing Floorplan

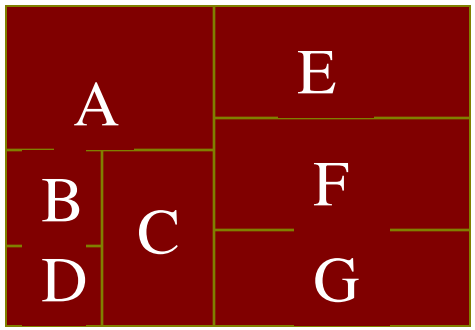


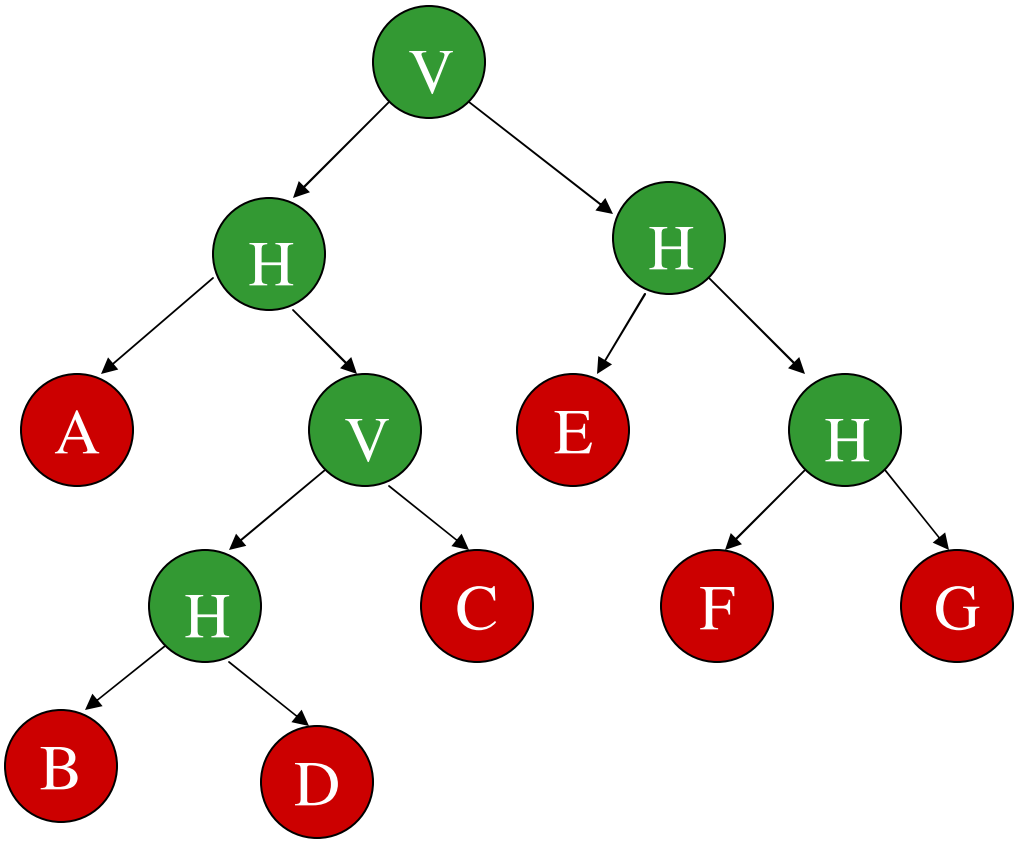
Non-Slicing Floorplan*(wheel)*



Slicing Tree

Slicing structure is described by a slicing tree





The Shape Algorithm

Input: Slicing Tree

Shape Constraints for modules

Cost Function (non-decreasing in w, h)

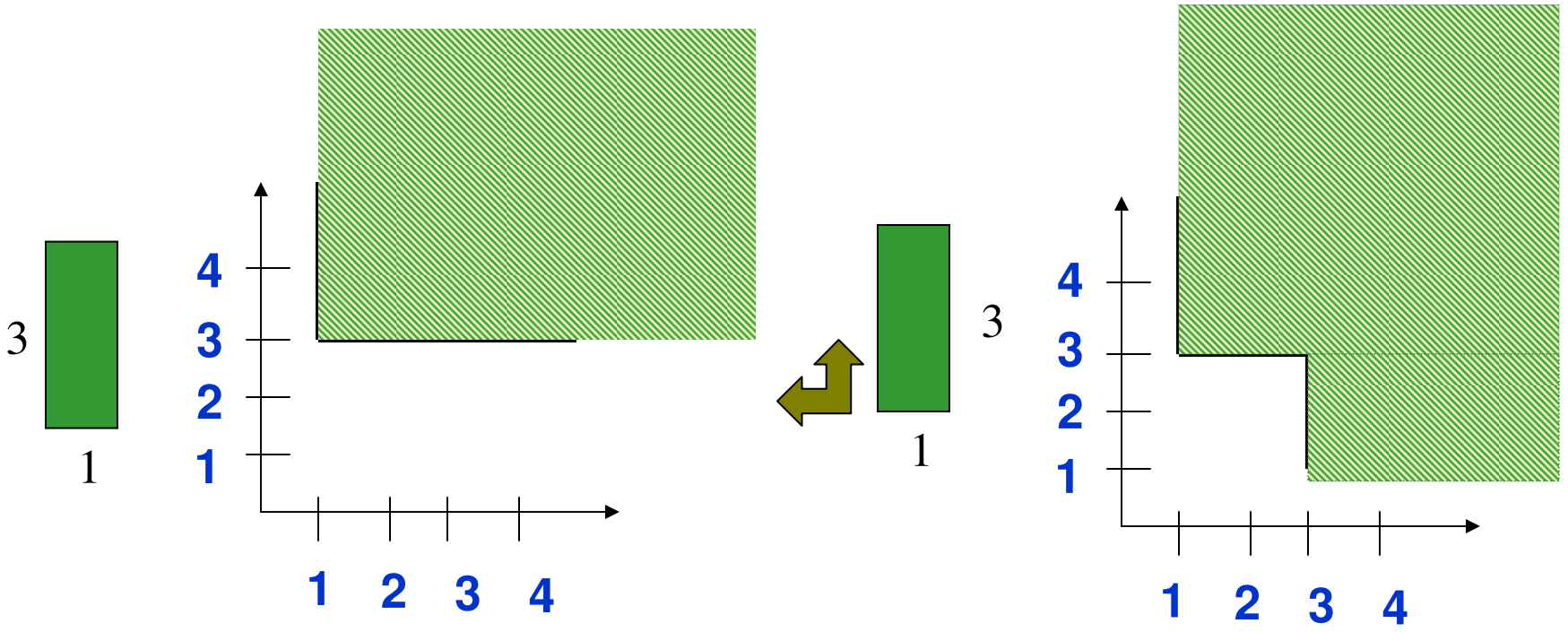
Output: Shapes/Implementation for each module

Algorithm

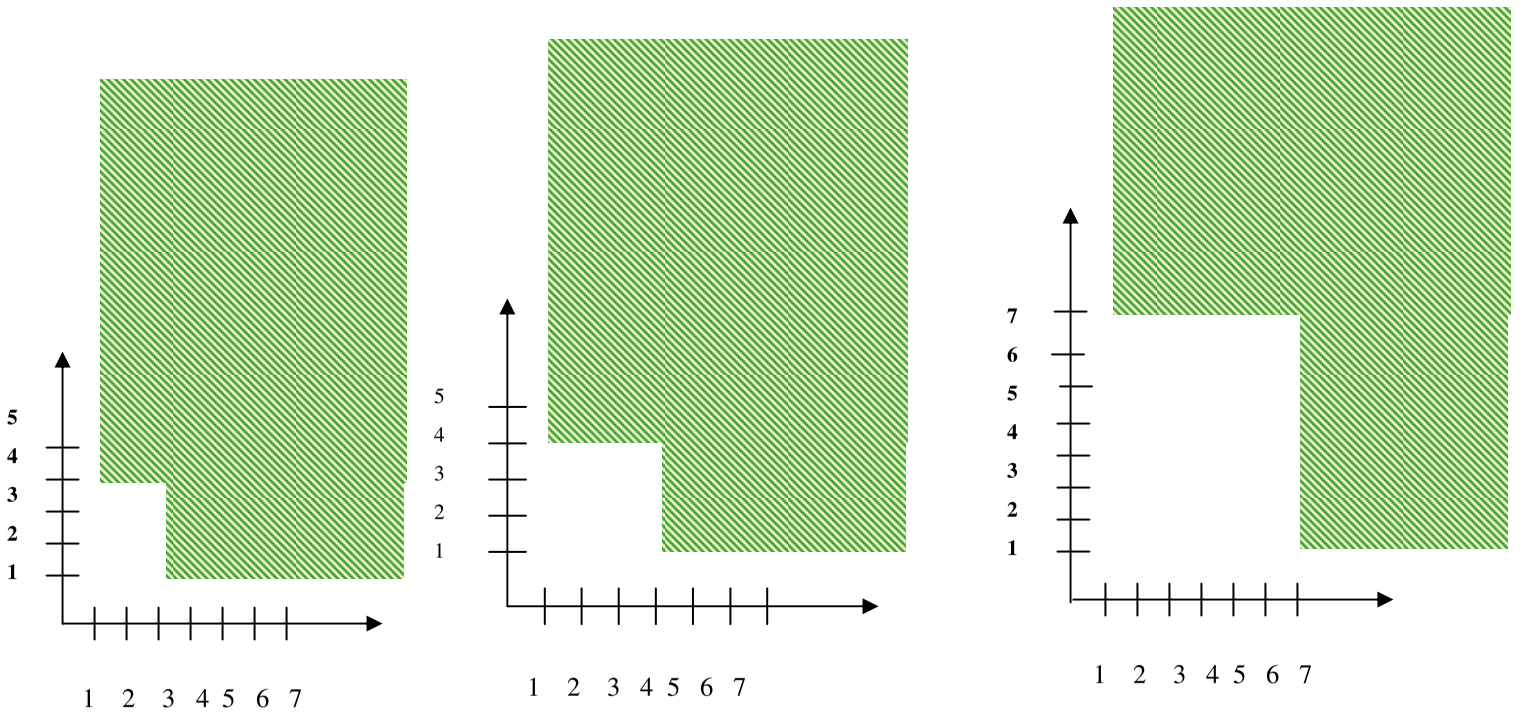
- Compose shape constraints bottom-up in the slicing tree
- Apply cost function to compute boundary point on shape constraint of base block (root)
- Propagate boundary point top-down in slicing tree to obtain implementation for each module

Shape Constraints

Given a rectangular module, the shape constraint relation R is the set of y-x pairs so that a rectangle with width equal to x and height equal to y contains at least a shape/orientation realization of the module.



Composing Shape Constraints



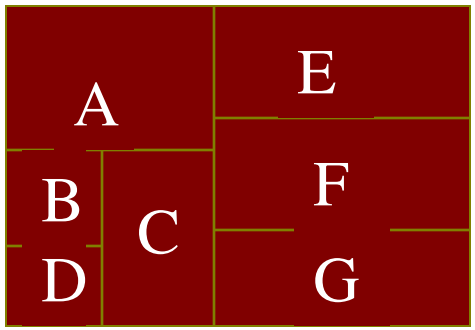
3 by 1

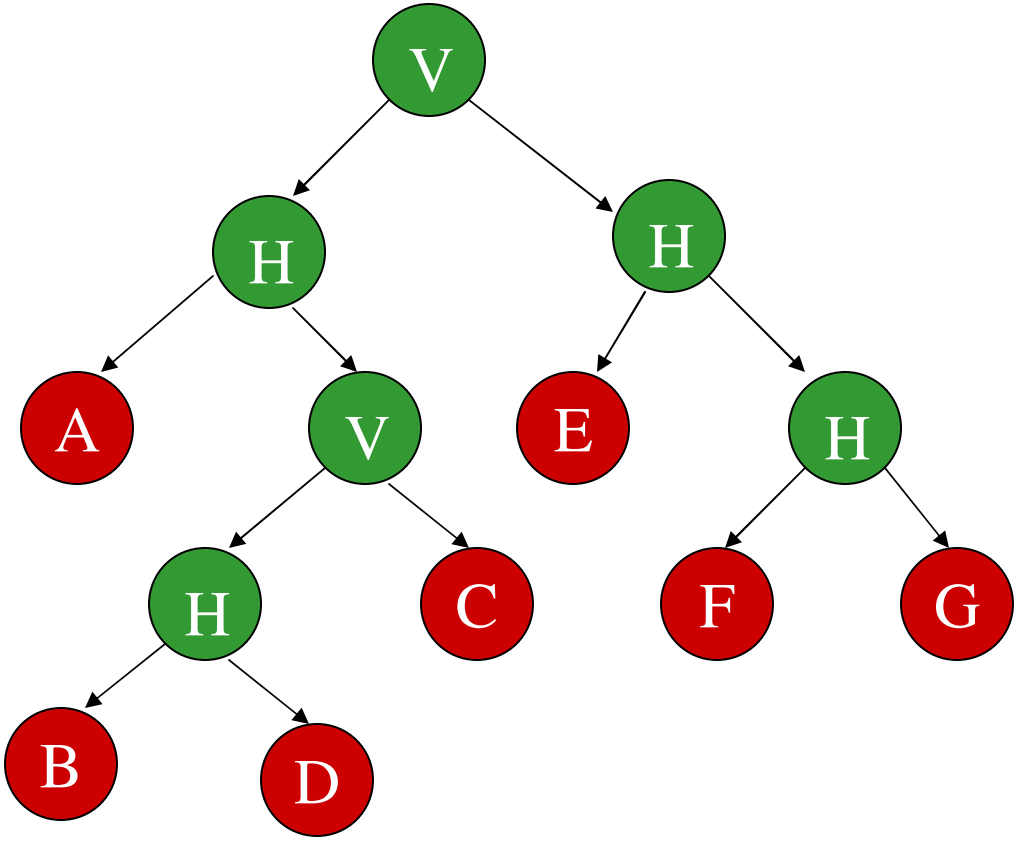
4 by 1

7 by 1

Slicing Tree

Traverse tree bottom-up deriving composed shape functions





The Shape Algorithm

Input: Slicing Tree

Shape Constraints for modules

Cost Function (non-decreasing in w, h)

Output: Shapes/Implementation for each module

Algorithm

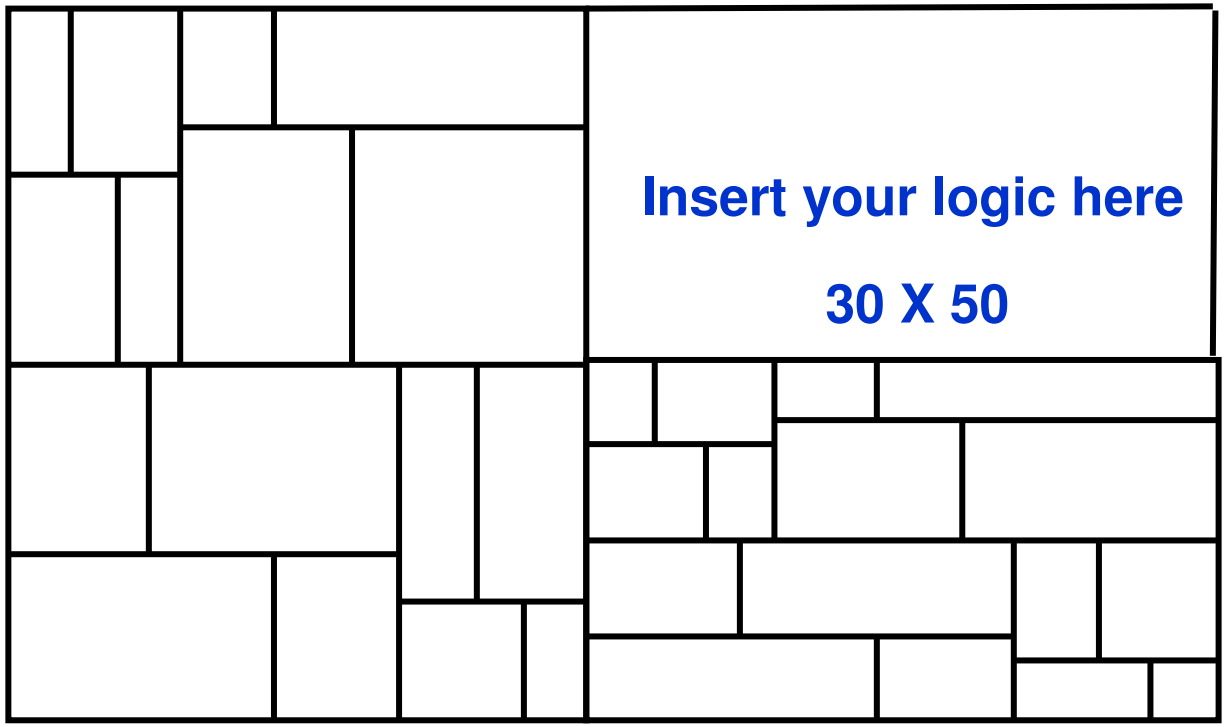
- Compose shape constraints bottom-up in the slicing tree
- Apply cost function to compute boundary point on shape constraint of base block (root)
- Propagate boundary point top-down in slicing tree to obtain implementation for each module

Constraining Placement of Macros

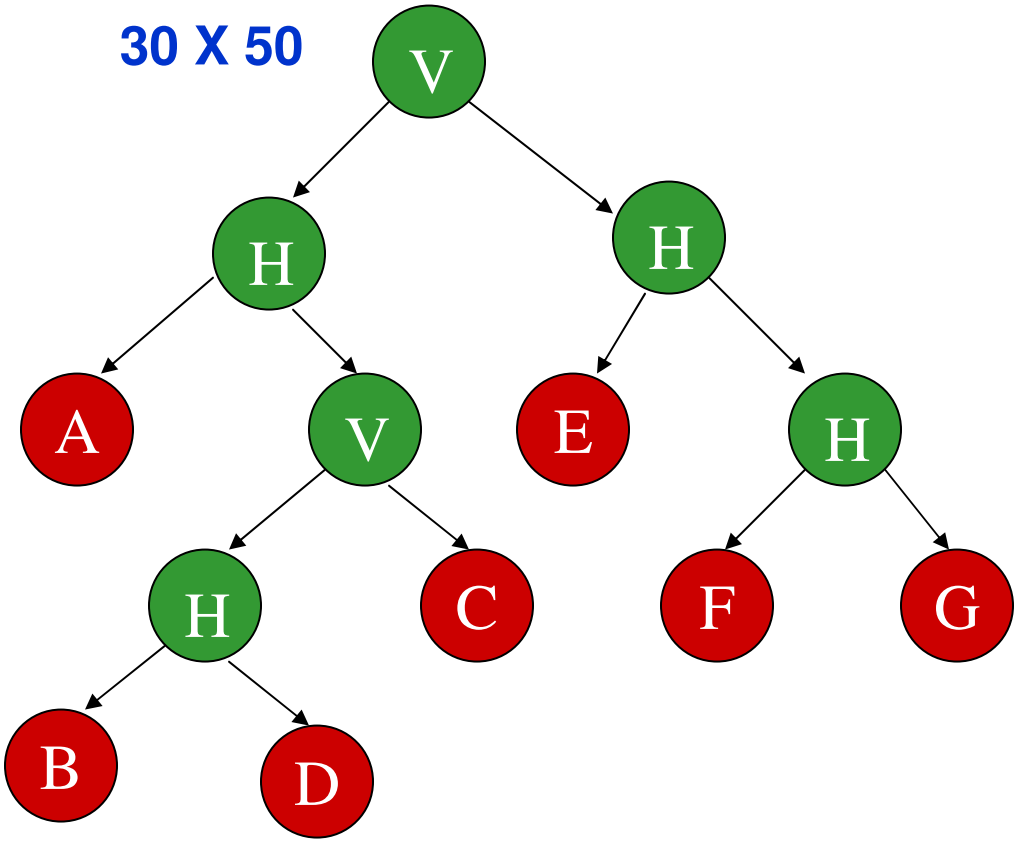
At top level can have complex shape functions

Apply a cost function and select lowest score point, e.g.

- **cost(w,h) = wh**
- **Cost(w,h) = 2(w+h)**



Slicing Tree



The Shape Algorithm

Input: Slicing Tree

Shape Constraints for modules

Cost Function (non-decreasing in w, h)

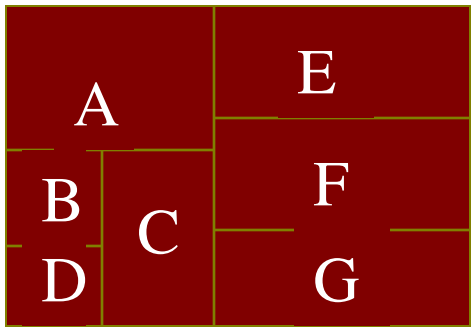
Output: Shapes/Implementation for each module

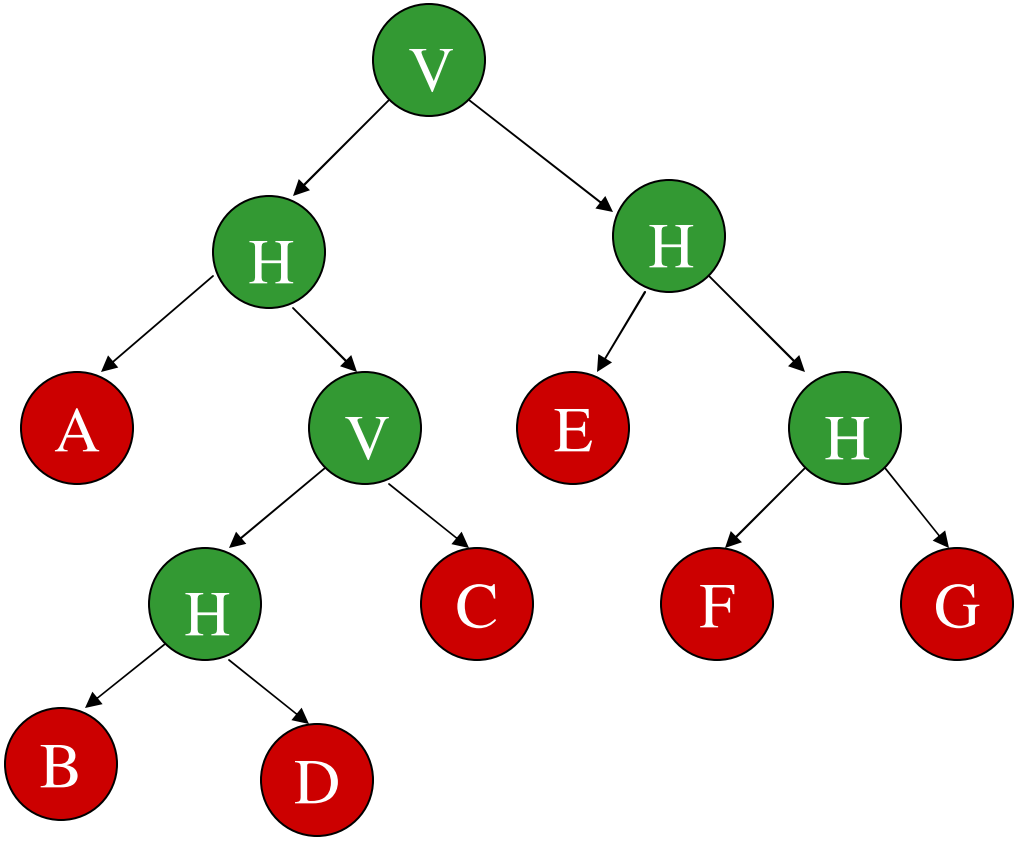
Algorithm

- Compose shape constraints bottom-up in the slicing tree
- Apply cost function to compute boundary point on shape constraint of base block (root)
- Propagate boundary point top-down in slicing tree to obtain implementation for each module

Slicing Tree

Propagate optimal choices top-down to all the leaves





Min-Cut Based Placement (Cont'd)