Solutions:
Custom Layout Design

June 25, 1999

Custom Layout Design

- No predefined building blocks
- No predefined structure
- Building blocks and structure tailored to problem at hand
- Usually transistor level

Why Resort to Custom Layout

- Exploit flexibility of transistor level layout.
- Analog circuits (tune for noise, gain etc..)
- High Performance design (highest speed, lowest power)
- Low cost mass produced (smallest die)
- IP differentiation

Design Objectives

- Speed
- Power
- Area
- Reliability

Characteristics of Custom Layouts

- Block size 100-10K transistors
  - Gate delays important
- Block put together to yield larger blocks
  - Wire delay is important

Session Overview

- Compaction Algorithms
  - Minimize area, leverage existing layouts
- Optimization
  - Maximize performance, minimize power …
- Layout Synthesis
  - Automated methods for Xtr level cells/blocks
- Automated Custom Tools and Methodologies
  - Interactive

What is Layout Compaction

- Move/stretch/shrink objects while preserving ***topology***
- Move/stretch/shrink to:
  - Fix DRC violations (migrate, fix bad layout)
  - Optimize objective function
    - Area
    - Yield
    - Delay/Cross-talk etc..

Why Study Layout Compaction

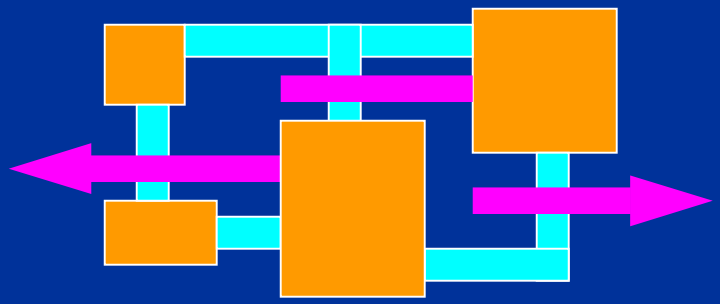
- Needed for IP migration in Section 4.
- Excellent example of formally mapping a layout problem into a mathematical problem
- Also good example of where formal modeling break down
  - Numerous special case handling situations

Types of Compaction

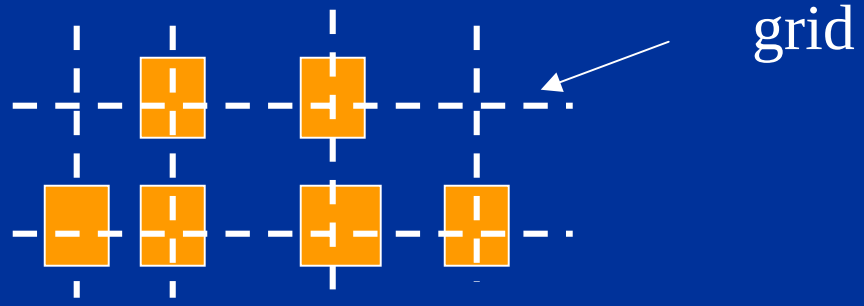
- 2 Dimensional (X and Y simultaneously)
  - Real: Branch and Bound /simulated annealing
  - Pseudo: 1D with perturbation in other direction
- 1 Dimensional (first X then Y..). In detail
  - Flat
  - Hierarchical

Types of 1D Compaction

- Shear Line

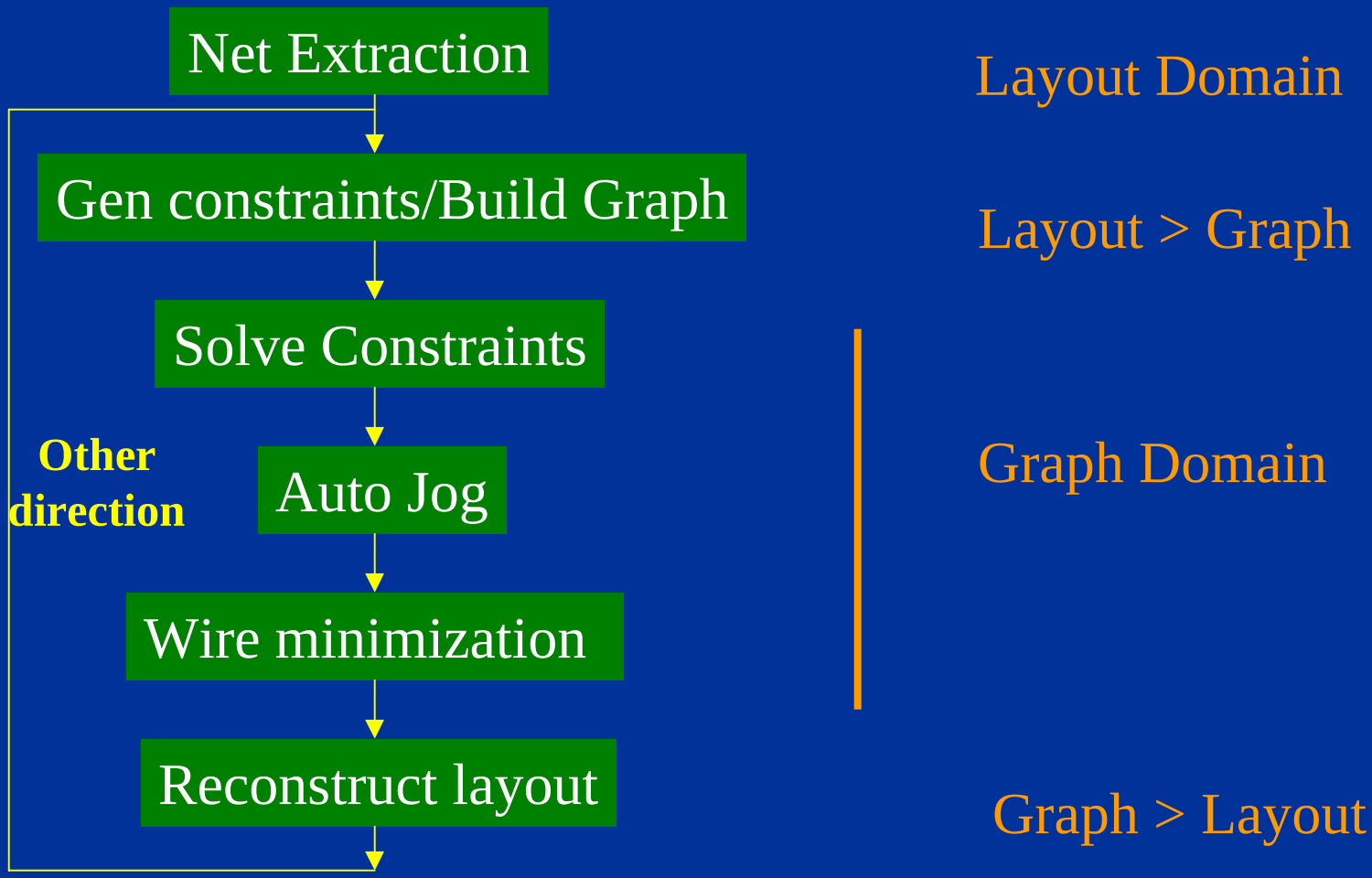


- Virtual Grid



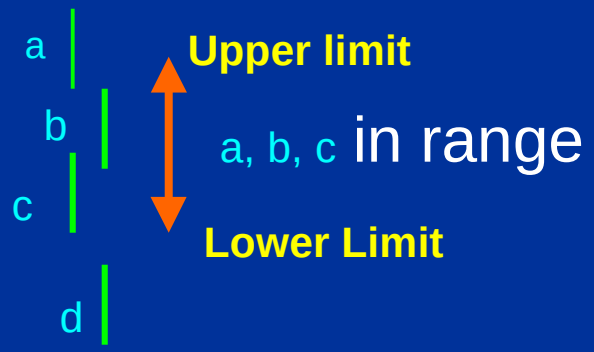
- Graph based (in detail)

Compaction Building Blocks



Scan-line

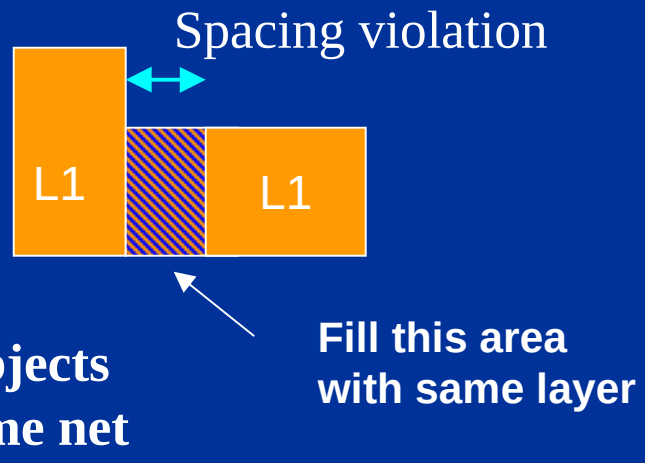
- Used in: Net extraction and Constraint generation
- Implements a collection of line segments
  - Efficient operations for: Add, Delete, enumerate segments in a range (overlapping)
- Possible implementations
  - Link list (slow), Segment tree (efficient)



Net Extraction

Why

- Two polygons on the **same layer and electrically connected** need/should not have a spacing design rule between them
  - Need to know if polygons have same net
- Can add material on same layer to fill gaps and fix spacing violations

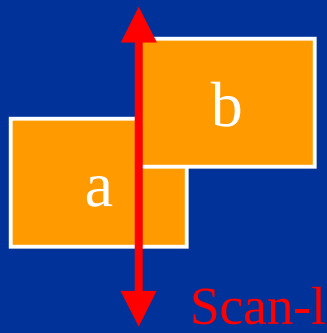


Net Extraction

How

- Give each rectangle a different net
- Report set of overlapping rectangles
  - Use scan-line (contains rectangles intersecting scan-line)

Sweep scan-line left to right



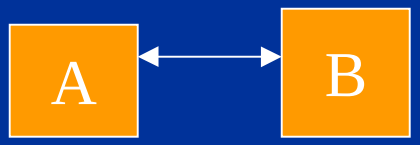
Find rectangles in scan-line intersecting this range

- Merge nets of each overlapping pair
  - Use union-find (merge classes of nets a and b)

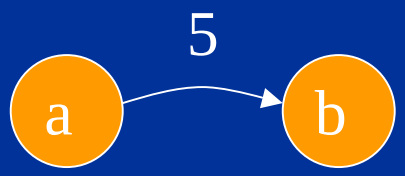
Design Rule Constraints

- Each layout object given a vertex (X dir)
  - Stretchable objects have > 1 vertex
- Create design rule constraints <=> add graph edges between vertices

A and B separated by 5u



Layout

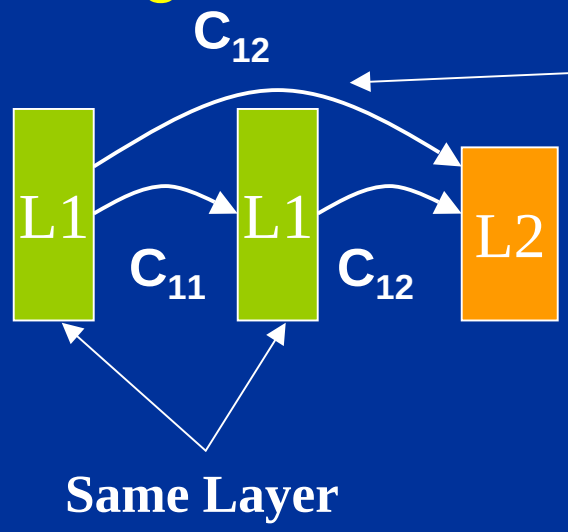


Graph

Constraint Generation

Brute force

- Add a constraint between all objects that overlap in Y ($X_{c_{12}}$ compaction)
  - Huge # of constraints. Most are redundant

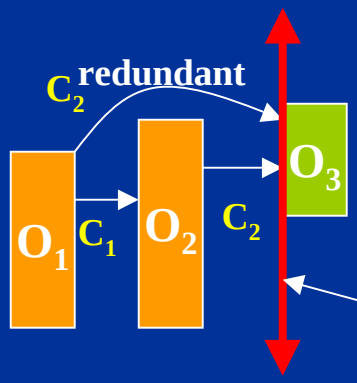


This constraint is always redundant because:

$C_{12} < C_{11} + C_{12}$

Constraint Generation
Shadowing Method

- Use scan line to prune useless constraints



If O_1 and O_2 have the same layer they both have the same spacing value C_2 to O_3. Since $C_1 > 0$ we have $C_1 + C_2 > C_1$. Hence the constraint from O_1 to O_3 is always redundant

Scan line does not contain O_1 when O_3 is processed

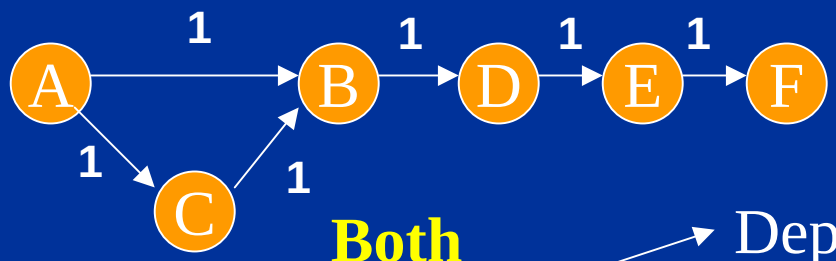
  - Delete all objects on same layer in O_2 range
  - Insert O_2
  - Enumerate all objects in range of O_3
- Prunes vast majority of useless constraints

Graph Solution

- Find a legal layout solution <=>
  Find positions for vertices that satisfy all edge constraints
  - 1) No solution. Detect and report to the user
  - 2) Find a solution that minimizes objective function
    - Usually objective function is area =>
      Longest path in graph

Graph Solution
Methods

- Depth/breath first and variants
  - *Push* all fanouts just enough to the right.
  - Trick is to:
    - Choose most efficient vertex to process first
    - Detect over-constraints (no solution case)



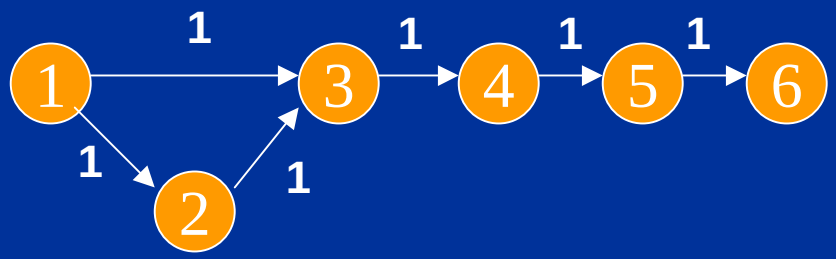
Both inefficient

Depth First Search: **ABDEFCBDEF**

Breath First Search: **ABCDBEDFEF**

Graph Solution
Methods

- Sort vertices topologically (levelization)
  - Vertex successor have higher sort number
- Process vertices, lowest sort numbers first



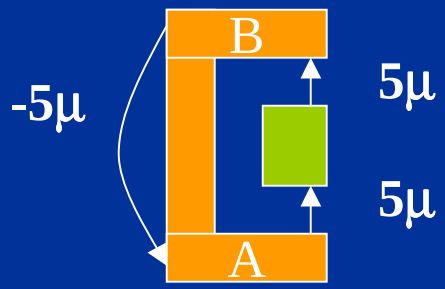
Levelized Search Order: **123456**

Over-constraints

- Manifest as positive cycles in the graph

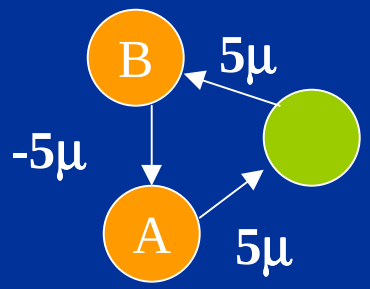
Over-constraint

A and B should not be too far apart



(*y compaction)

Positive Cycle



5μ+5μ -5μ > 0

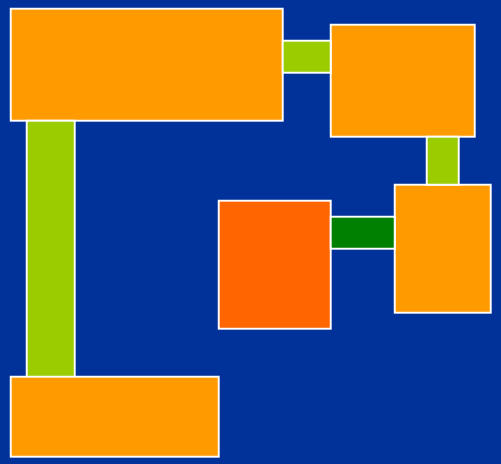
- Classify edges into forward and backward edges
  - Graph of forward edges acyclic (use depth first search)
  - Check for positive cycles when examining backward edges

Secondary Optimization Objectives

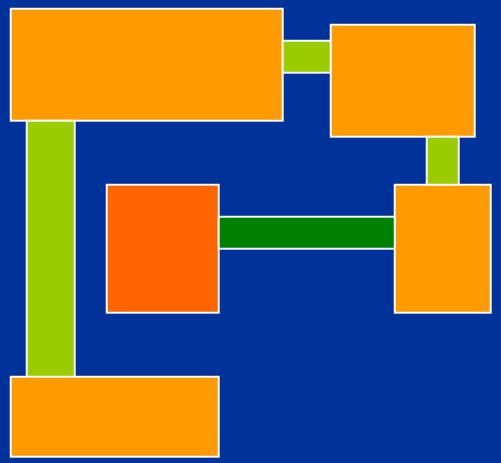
- **There are usually a large number of solutions all of whom have min area (internal slack redistribution).**
- **Select the one(s) with some additional desirable properties. Keep area at a min.**
- **Smallest wire length, best yield, best Xtalk**

Wire length minimization

- Without WLM wires can get too long



With WLM

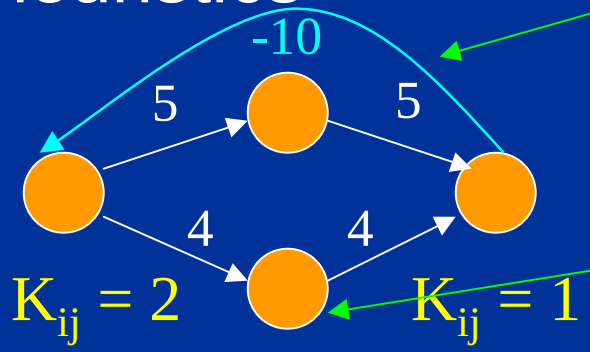


Without WLM

Reduces total wire capacitance (usually also resistance)

Wire Length Minimization Problem Formulation

- Minimize (keeping minimum size) $\Sigma_{ij}(K_{ij*}\text{Length}(\text{Wire}_{ij}))=\Sigma_{ij}(K_{ij}(V_i-V_j))$
- This is a network flow problem.
- Generally solved/approximated using efficient heuristics



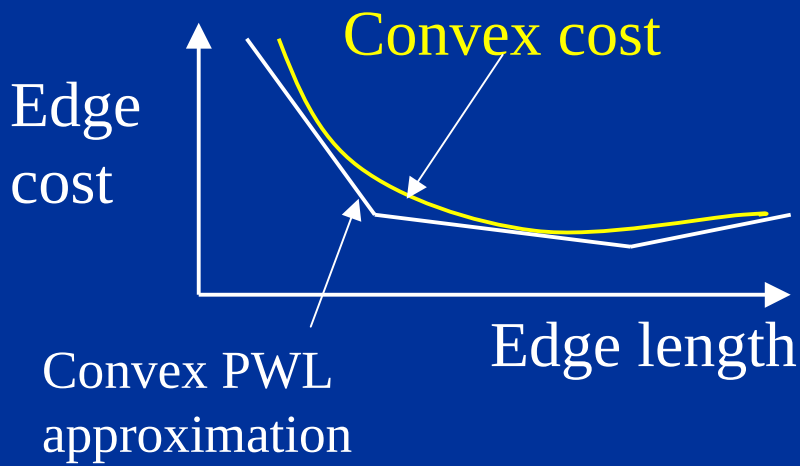
Edge added to maintain minimum size.

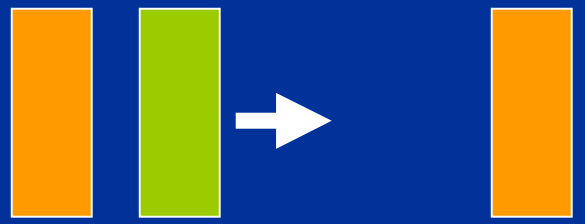
This Vertex can assume any position in [4,6]
Choose loc = 4 for min cost

Optimization of other Objectives

- Extend WLM method to Yield, Xtalk etc..
  - Cost (E_{ij}) = f_{ij}(length(E_{ij}))
  - Minimize Σ (Cost(E_{ij}))
- Can be approximated by network flow pb for many commonly occurring costs (convex cost)
- Convex function => PWL convex approximation
  => Network Flow Problem
  => Use WLM heuristics to solve

Convex Cost Objectives

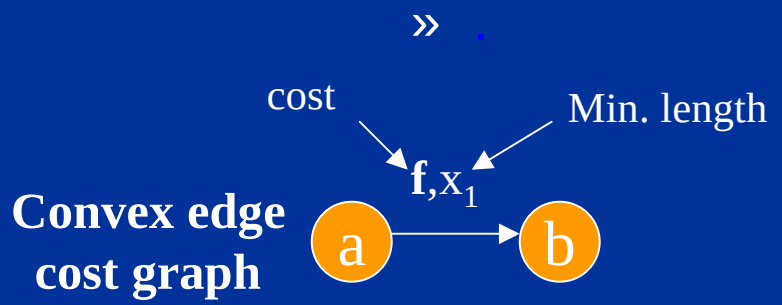


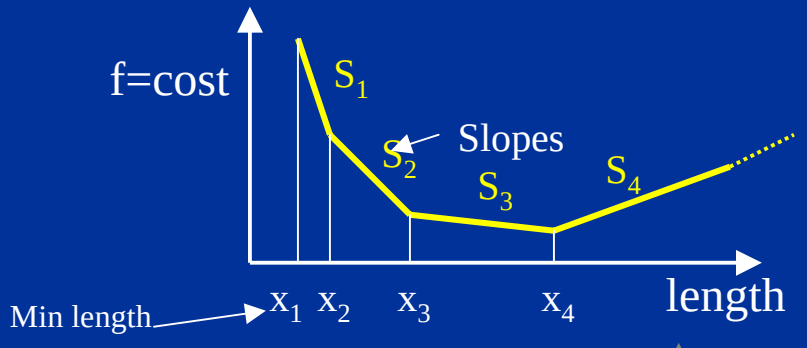


Convex => Move objects to fill empty space

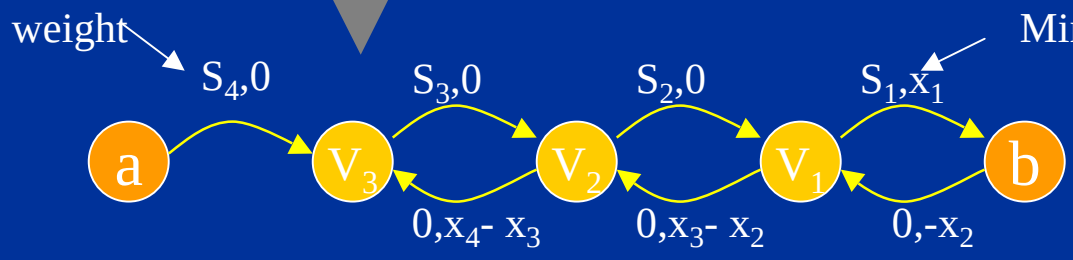
- Yield, Cross Cap,Wire length (linear) are convex
  - Sum of convex functions is convex
- Optimize weighted sum of convex cost functions. (Yield and WLM minimally mutually interfere)

Enhanced Network Flow



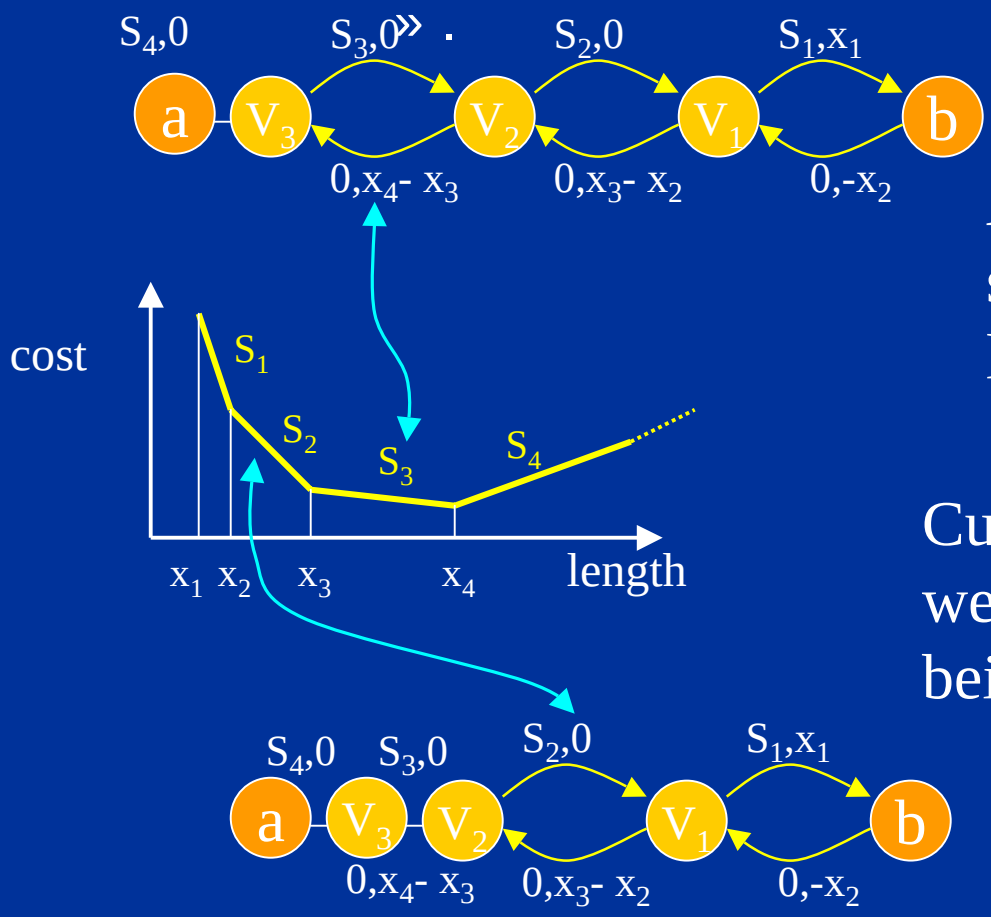


Minimum cost of **this** graph (as a function of length) tracks **this** curve



Linear edge cost graph

ENFA: Graph cost tracks function



As length is shortened segments get compressed left to right (highest cost first)

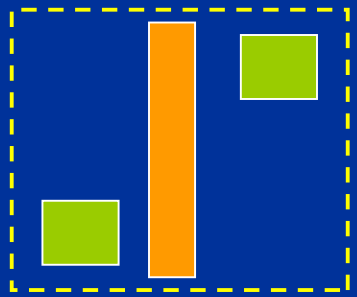
Current slope corresponds to weight of segment currently being compressed (blue arrow)

ENFA: How to solve

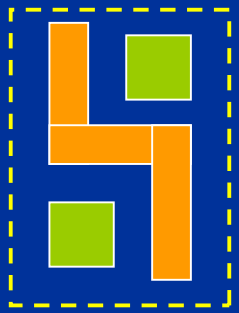
- Optimization of PWL convex cost objectives transformed into network flow problem
  - Replace each convex cost edge in graph by sub-graph previously shown
  - New graph edges costs are linear with length => Network flow problem
  - Solve new graph using WLM heuristics

Automatic Jogging

- Break wires into jogged sections to reduce area (or other objective)



Without Jogging

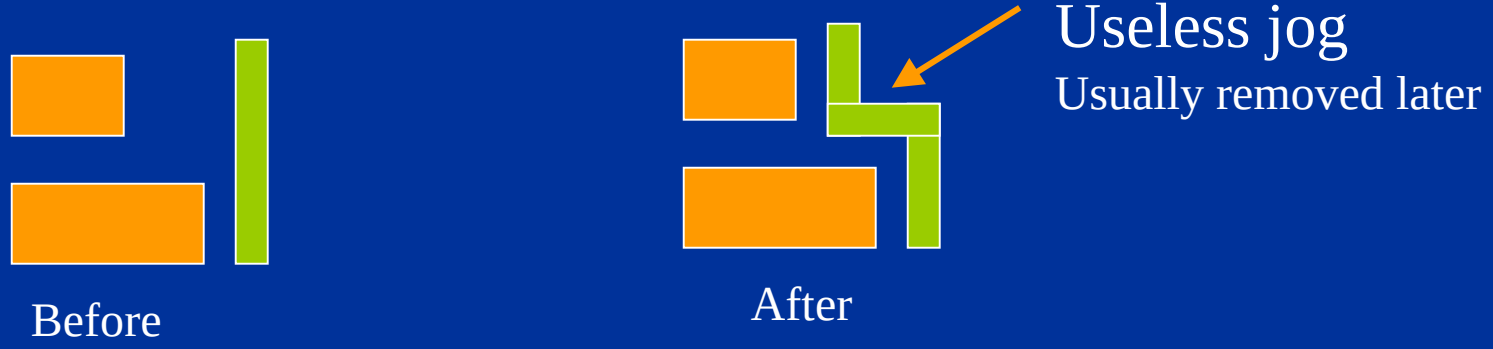


With Jogging

- Two basic methods
  - Exhaustive
  - Incremental

Exhaustive Jogging

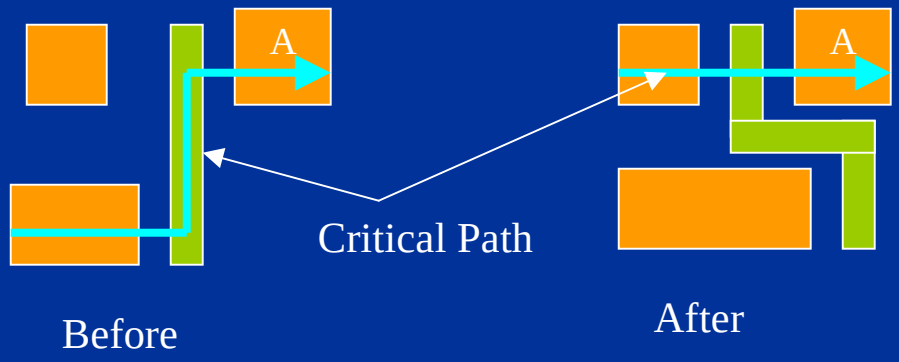
- Jog wires at all locations which may result in an area decrease. No critical path info



  - Advantages: Simple, single pass,breaks up wires for secondary objectives (e.g. yield)
  - Disadvantages: Can result in more memory and runtime

Incremental Jogging

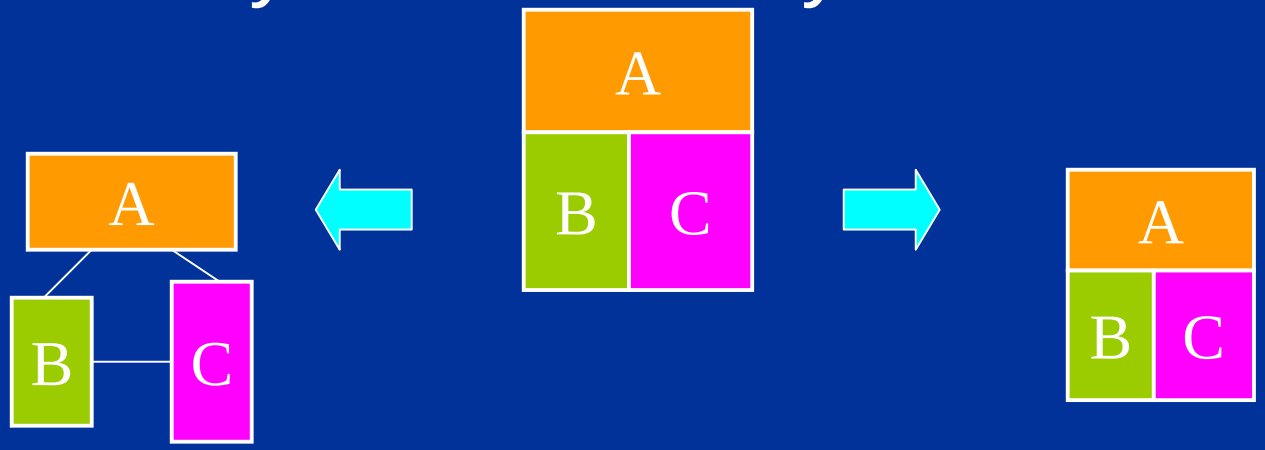
- Jog wire if improves critical path
  - Wire jogged only if object “A” is present



- Advantages: Less mem. & CPU (large layouts)
- Disadvantage: Complexity, iterative solution

Hierarchical Compaction
Different Kinds

- Hierarchy in = Hierarchy out



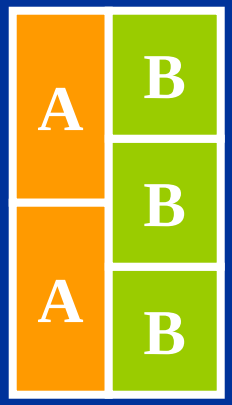
Blocks A,B,C compacted **separately**. Routed together after compaction.

Routing Compaction

Blocks A,B,C compacted **together**. **True hierarchical compaction.**

Pitchmatching

Hierarchical Compaction Is an LP Problem



$2*A = 3*B$

Cannot be expressed in terms of graph constraints

- Added complexity in solver and representation
- Must reduce complexity
  - Approximate solutions
  - -Reduced Representation

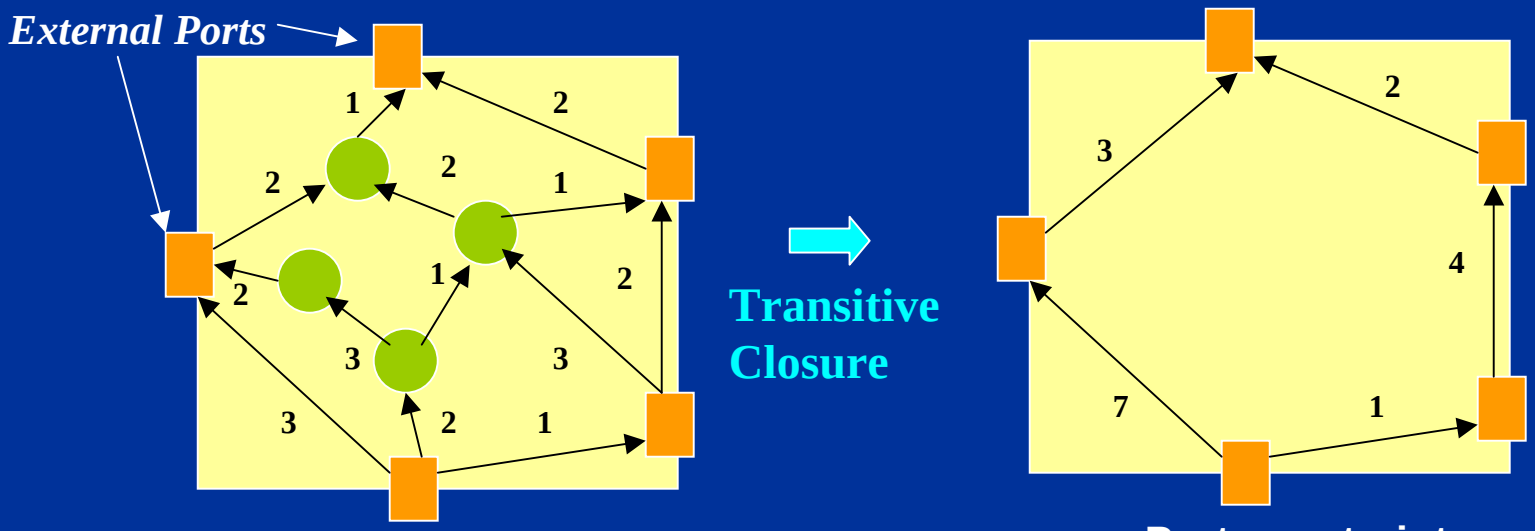
Compaction Problem Size

	Variables.	Constraints
Flat 100K Xtrs	7M	35M
Hier Mem 500K	100K	300K
Random Block 20K	200K	600K
Datapath 20K	200K	1M

- Problem too big for normal LP/ILP

Hierarchical Compaction
LP Complexity Reduction

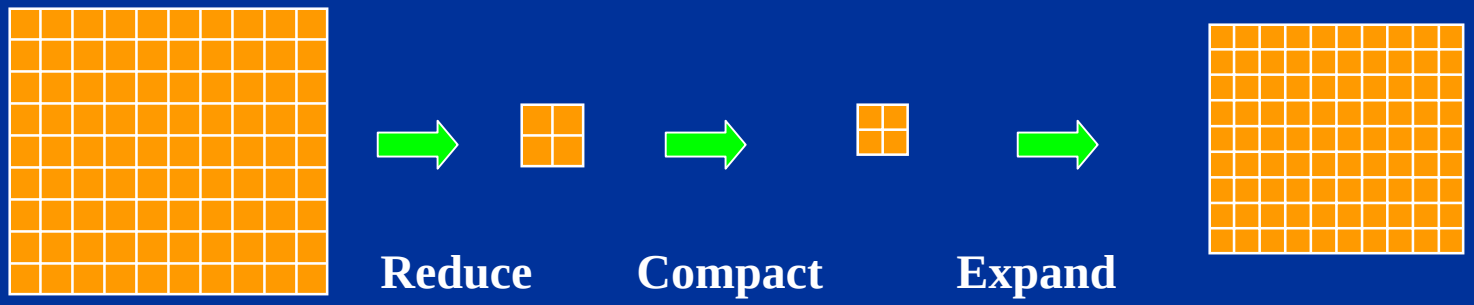
- Solve as much as possible in graph domain
  - Simplify each cell by removing internal objects



Internal objects can always be put back if port constraints are satisfied

Hierarchical Compaction Exploiting Regularity

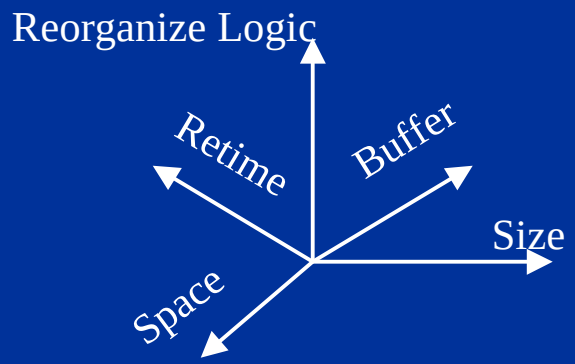
- Exploit Repetition (compact repeating configurations once)
  - 100X100array => compact 2X2 array
  - More complex forms of regularity can be exploited



Optimization
Design Tradeoffs

- Speed / Power / Area
- Must compromise and choose between often competing criteria
- For a given criteria (constraints) on some variables make best choice for free variables (min cost) => Need to be on boundary of feasible region

Optimization Methods

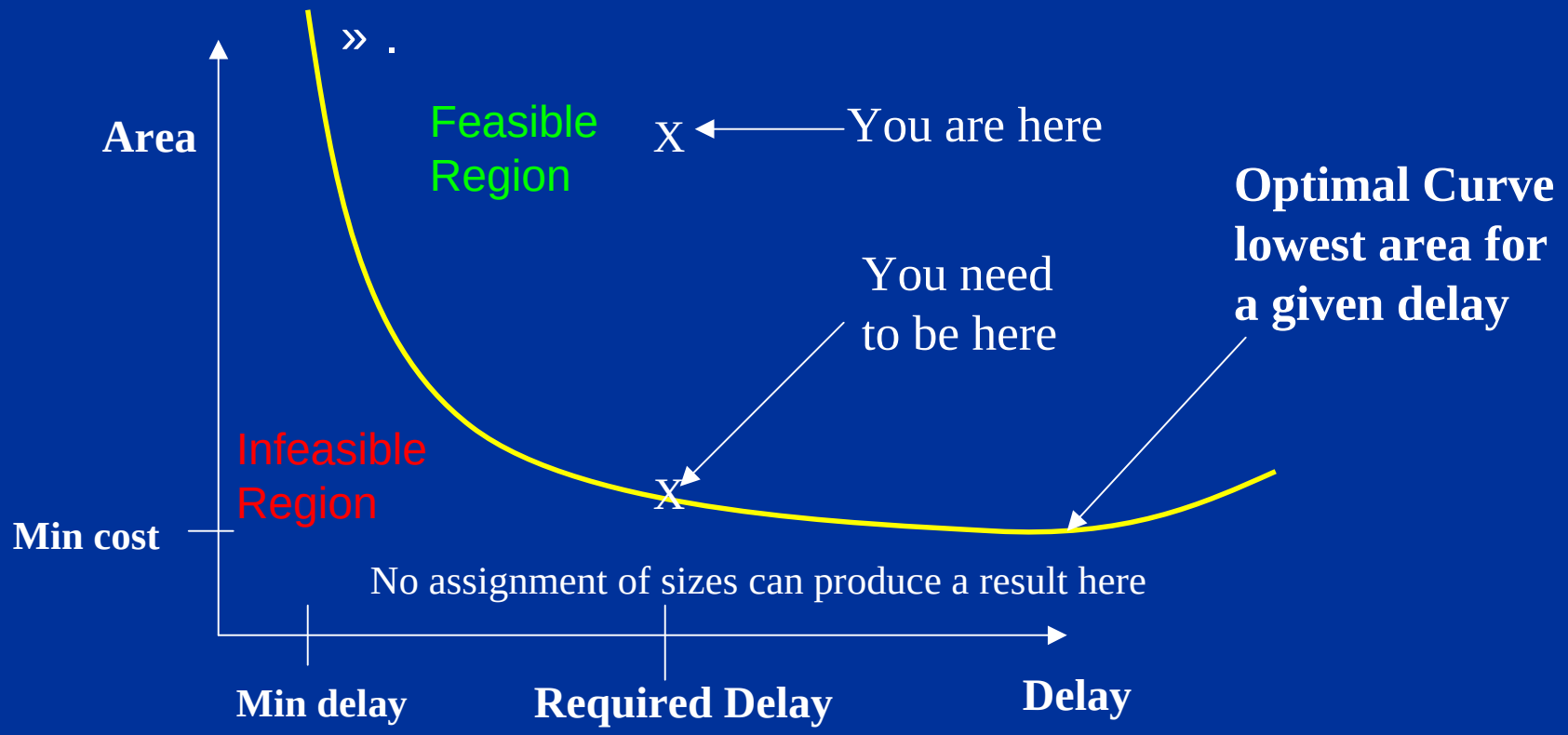


- Many different kinds of delay/area optimization are possible
- Many optimizations are somewhat independent
  - Use several different optimizations. Apply whichever ones are applicable

Optimization at Layout Level

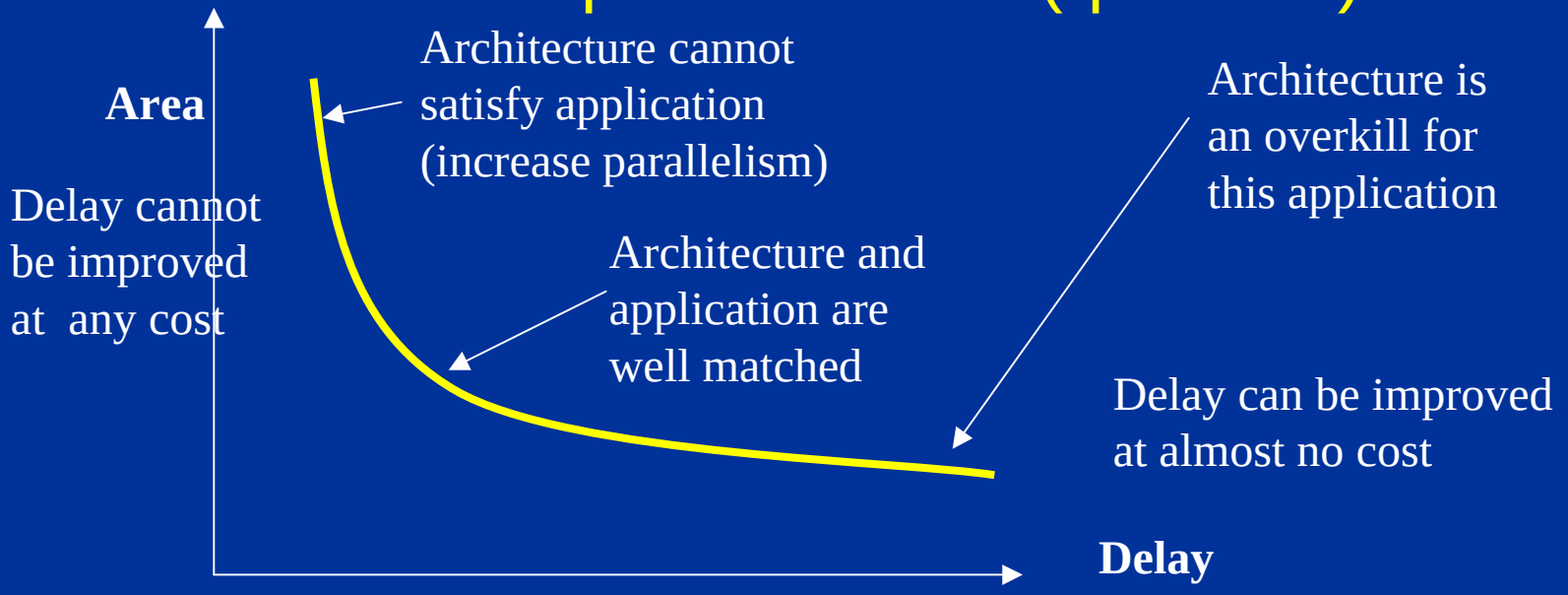
- Size Transistors
- Space/size wires
- Add/delete buffers
- Modify circuit locally

Transistor Sizing
Area Delay Curve



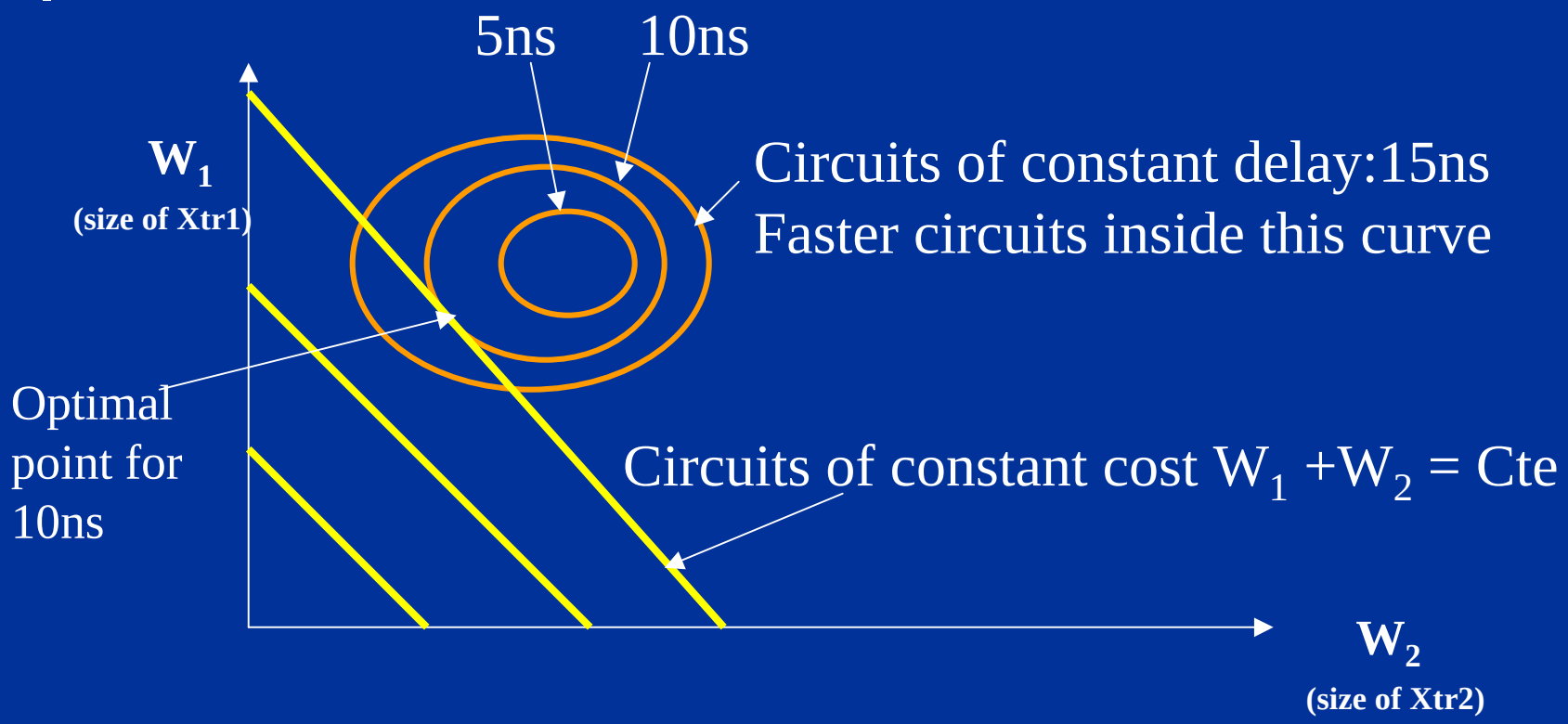
Transistor sizing
What will it buy me?

- Scenario: Lots of capacitance in wires
  - Will it buy me speed: Yes
  - Will is save me power: "Yes" (qualified)



Transistor Sizing
Convexity + Dual Goals

- .



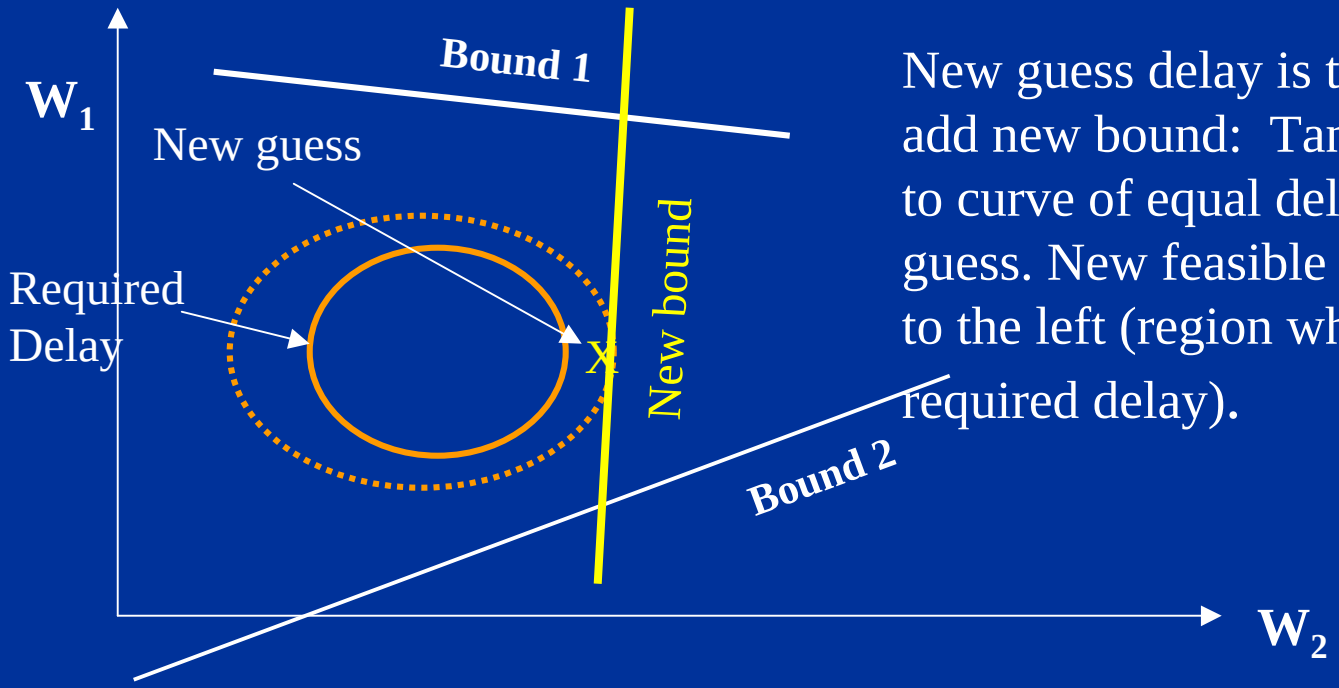
Note: Actually circuit delay is Posynomial ~ Convex

Transistor Sizing Methods

- Exact Solutions
  - Gradient Search
  - Convex Programming
- Approximate methods (very good solutions)
  - Iterative improvement on critical path (e.g. TILOS)

Convex Programming Outside Delay Case

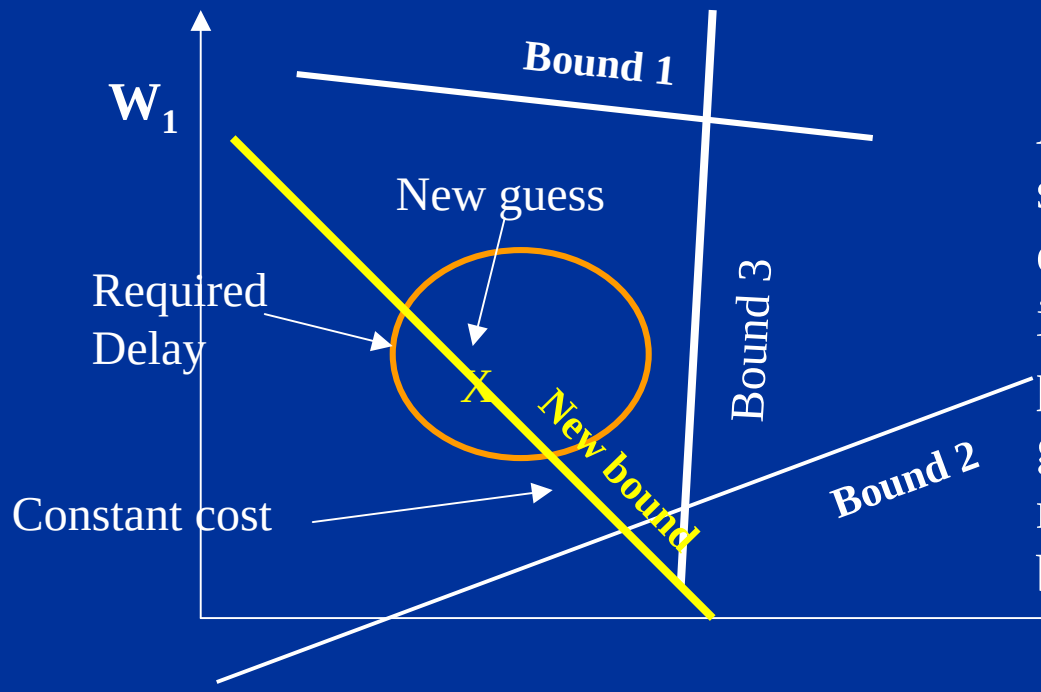
- Add more and more bounds
  - Guess new solution (deep) inside bounds



New guess delay is too slow so add new bound: Tangent to curve of equal delay at new guess. New feasible region is to the left (region which contains required delay).

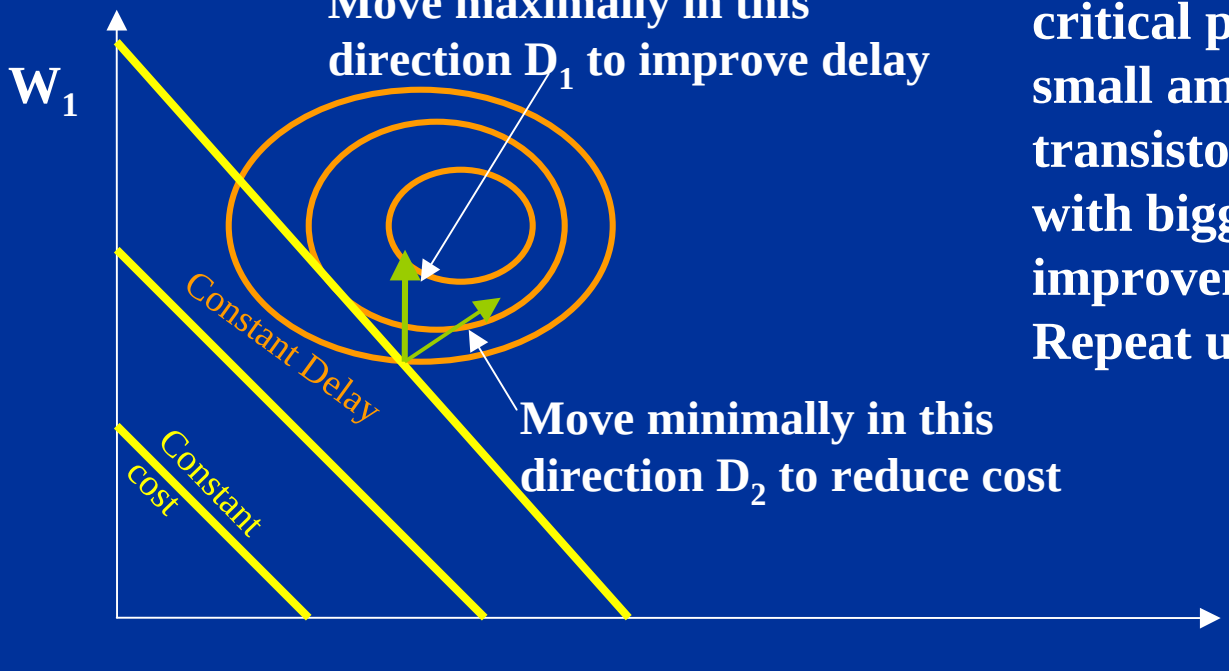
Convex Programming Inside Delay Case

- New guess delay is adequate but try and improve cost



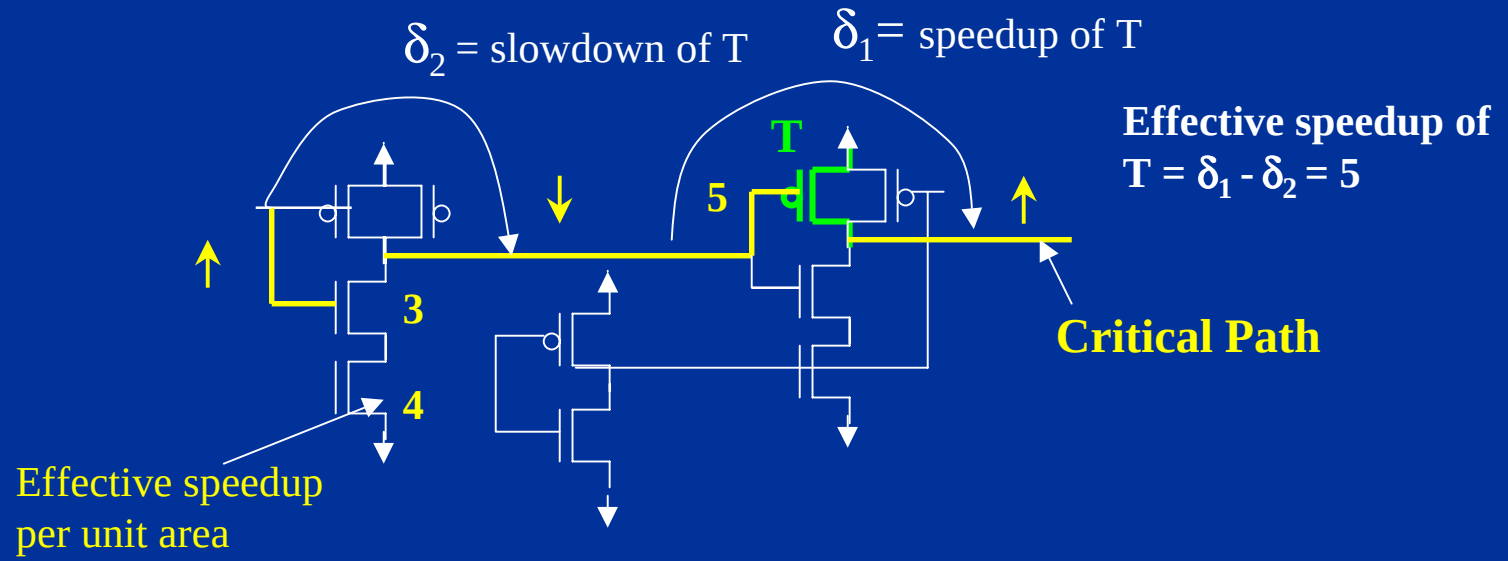
Add a bound to force search into region of lower cost. New bound is constant cost curve passing through new guess. New feasible region is below new bound.

Transistor Sizing Approximate Solutions



Circuit delay affected only by delay of critical path. Upsize by small amount transistors on crit path with biggest D1/D2 = improvement/cost. Repeat until timing met

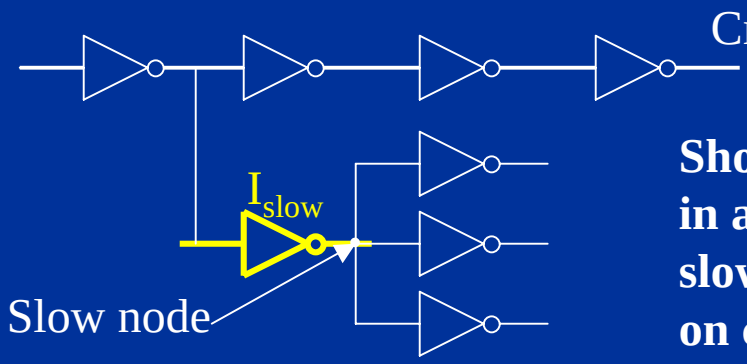
Transistor Sizing
TILOS method



- Increase Xtr on critical path with largest per unit effective speedup: T

Short Circuit Power Optimization

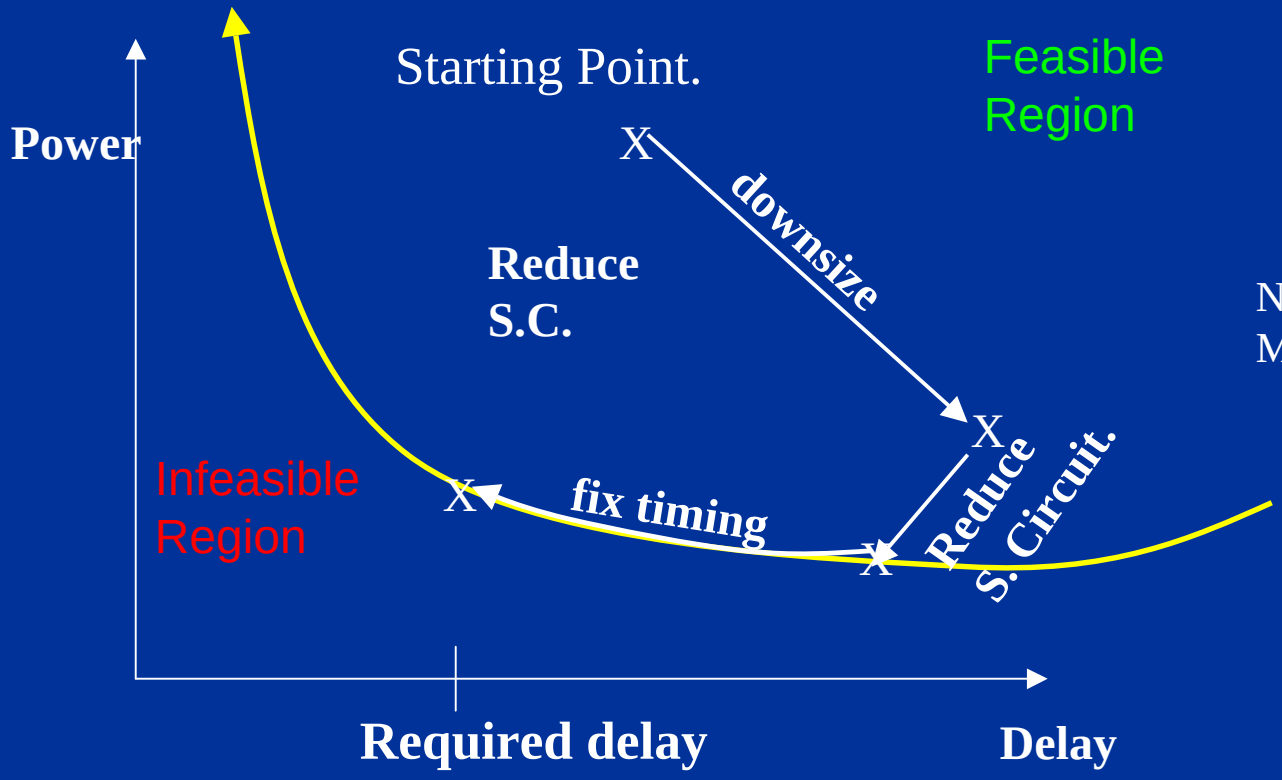
- Critical path methods miss short circuit power



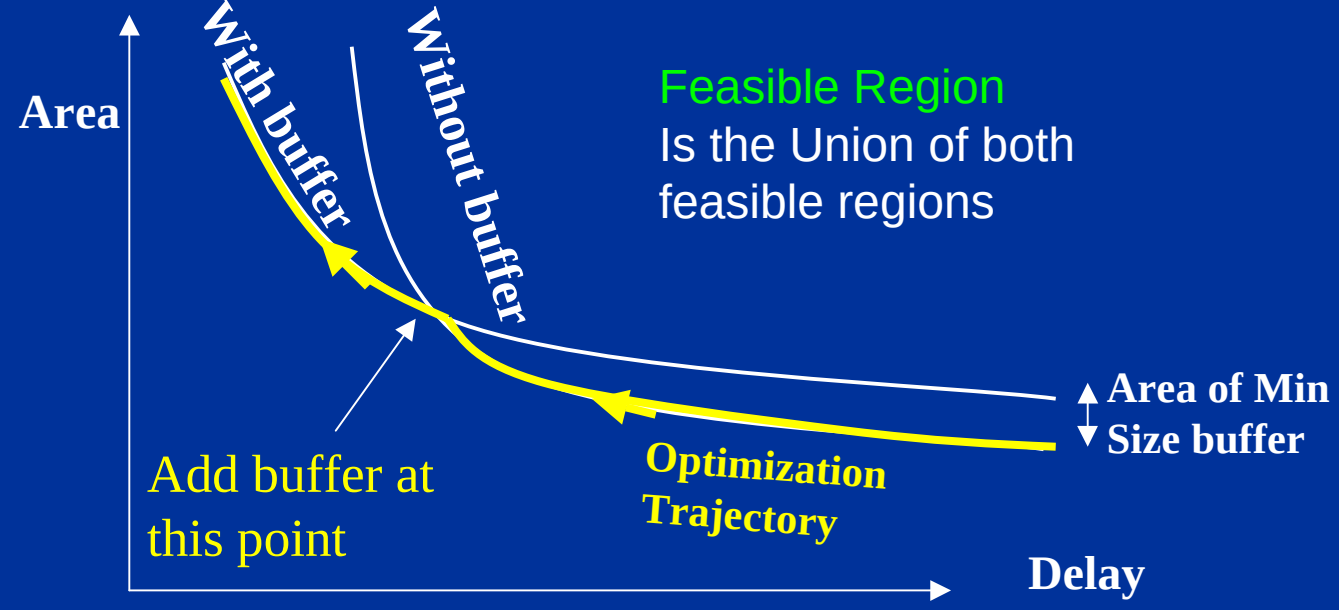
Short circuit power burned in all of these gates due to slow input rise time. Gates not on critical path

- Increase I_{slow} until capacitive power increase for driving I_{slow} is more than decrease in S.C. power
  - Sweep circuit from outputs to inputs

TILOS Optimization Trajectory



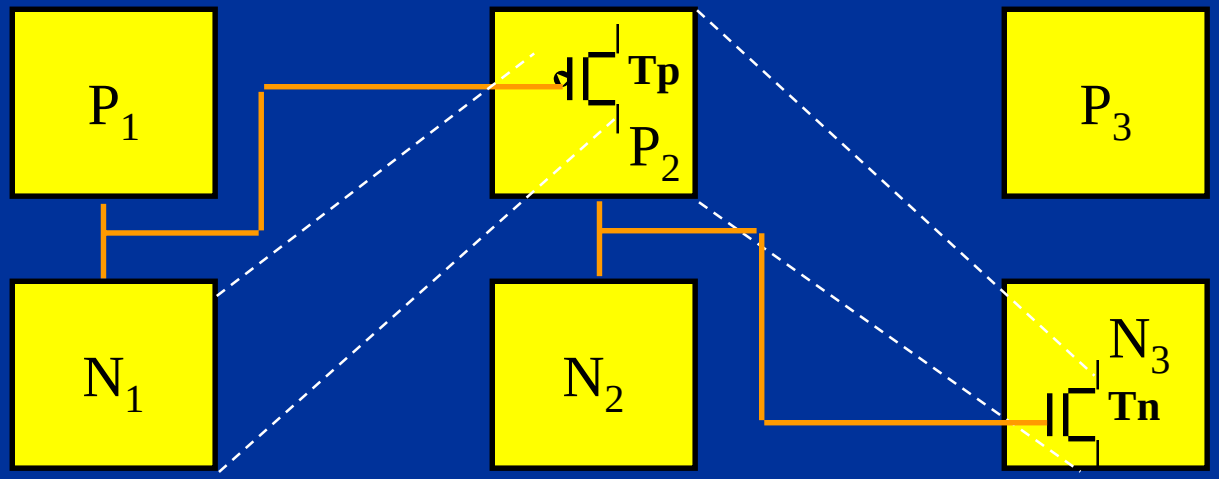
Buffer Insertion
Area delay tradeoffs



- Optimal curve is envelope of curves
  - Jump to buffered curve during timing optimization

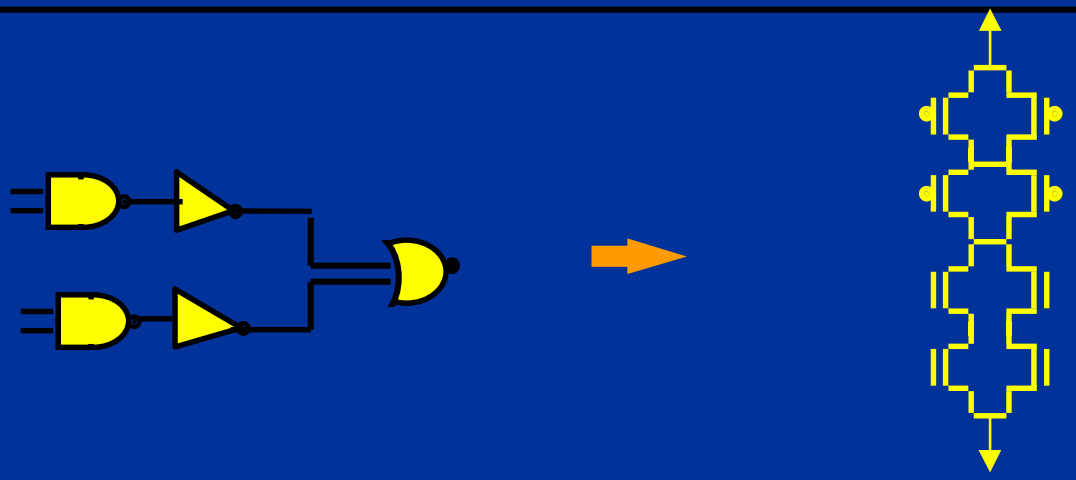
Local Re-synthesis

- Pass Xtr re-synthesis, logic reorganization
- Gate collapsing



- T_P conducts <=> N_1 conducts. Replace T_P with N_1
  - Repeat for P_2 and T_n for correct NMOS/PMOS

Gate Collapsing Example



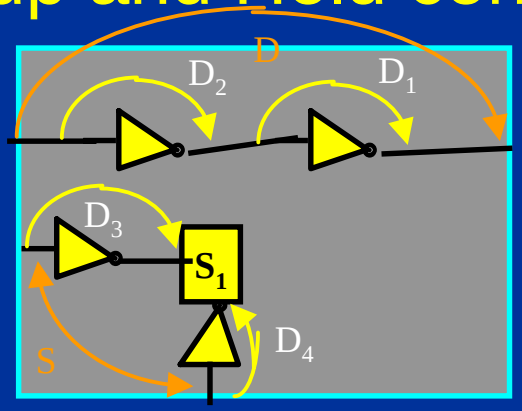
- Trade off drive-capability/logic-levels
  - Intrinsic Delay ↘ RC Delay ↗
  - Reduce number of transistors (area ↘)

Timing Abstraction
Why

- Timing Model depends on layout parasitics and optimization
- Needs to be computed from layout (bottom up) unlike functional information
- Model needed for assembly/verification at next level of hierarchy

Timing Abstraction
Inter-block (Black Box)

- External interface characteristics of block
  - Pin to Pin delays
  - Setup and Hold constraints

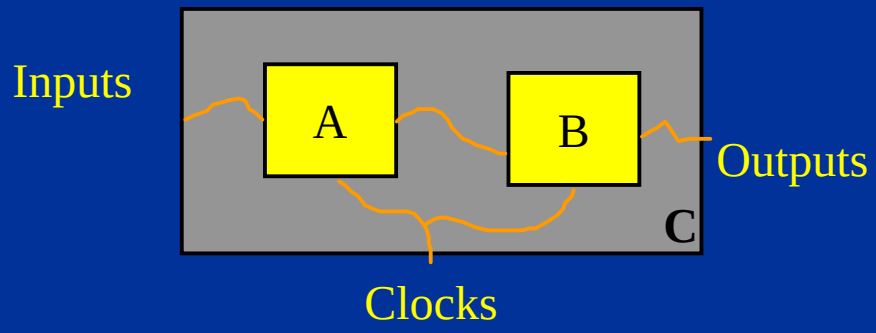


$D = D_1 + D_2$ **(delay)**
$S = S_1 + D_3 - D_4$ **(setup)**

- D, S depend on input slews & output loads => Use a table model (1 entry/slew-load pair)

Timing Abstraction Intra-block

- Tells you if *insides* of block are working
  - Models internal setup and hold
  - Usually constraints on the clock



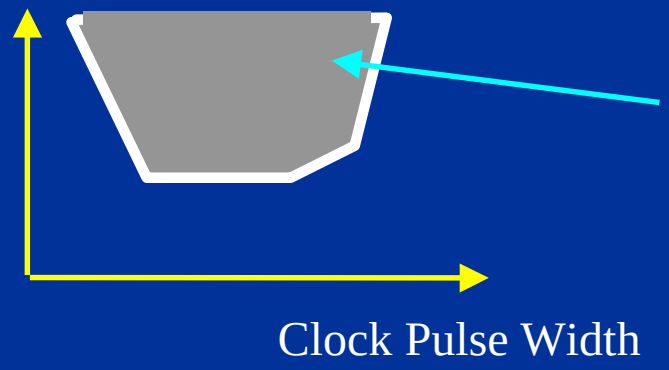
Are A and B working properly?

- For block C to work properly
  - Interblock constraints must be satisfied
  - Each subblock must itself work

Timing Abstraction
Intra-block (Grey Box, Clock Model)

- Grey Box: Reduced representation of circuit
  - OK/Not OK answer (Boolean answer)
- Clock Model: Analytical expression of feasible region of clocking (linear inequalities)

Example for Regular Clocking



Cell Layout Synthesis

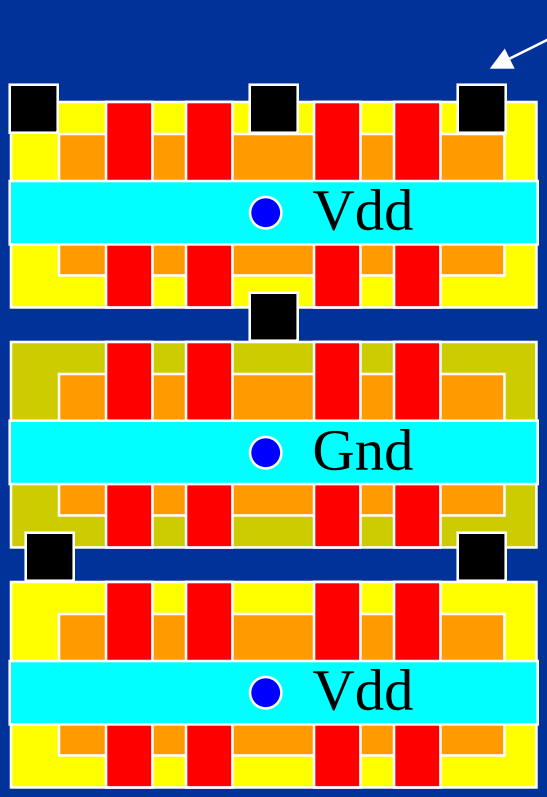
- *Closed Library* approach
  - Most appropriate for ASIC
- *Open Library* approach
  - Custom Design
  - Driven by Power Delay Optimization
  - Synthesis cells on demand.
- Technology Independent
  - Essential for rapidly advancing process

Cell Image

- Control geometric properties of cell so that they can be efficiently assembled (intercell considerations)
- Random Logic
  - Height fixed. As small as possible pitch
- Custom
  - Tailored to structure of design (e.g. datapath)
    - Direction of M1 and M2 in cell flexible

Cell Image Layout

- Power, ground, ports, layers, PN regions



P transistor region

N transistor region
(multiple rows optional)

P transistor region
(for double height cells)

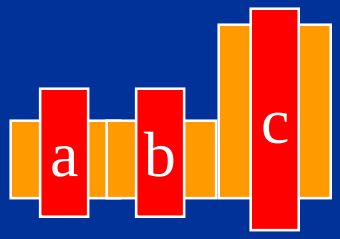
Cell height

Automatic Cell Generation Flow

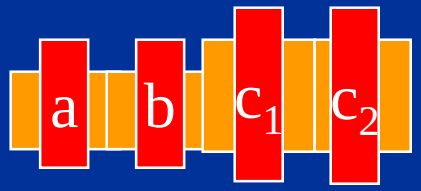
- Input: cell image, netlist, layer preferences
- Flow
  - Placement
    - Euler path (diffusion sharing), Min cut order, Annealing, DFS, Branch-Bound
  - Intracell routing (maze)
  - Compaction
    - Reduce area, insert jogs, WLM, fit image

Transistor Size Consideration

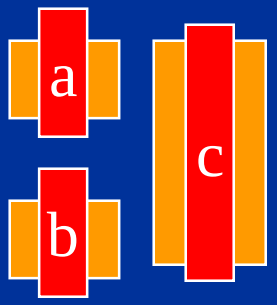
- Mixture of small and large transistors
  - Occurs for large drive cells (size optimization)
  - Must reduce wasted area



Unoptimized Placement



Transistor Folding



Transistor Stacking

Block Layout Synthesis
Why

- Automate Block Layout without overhead and restriction of fixed cell library
  - New processes for which cells do not yet exist
- Handle custom netlist
  - Allow wide range of sizes and gate types
  - Facilitate use of transistor sizing

Block Layout Synthesis How

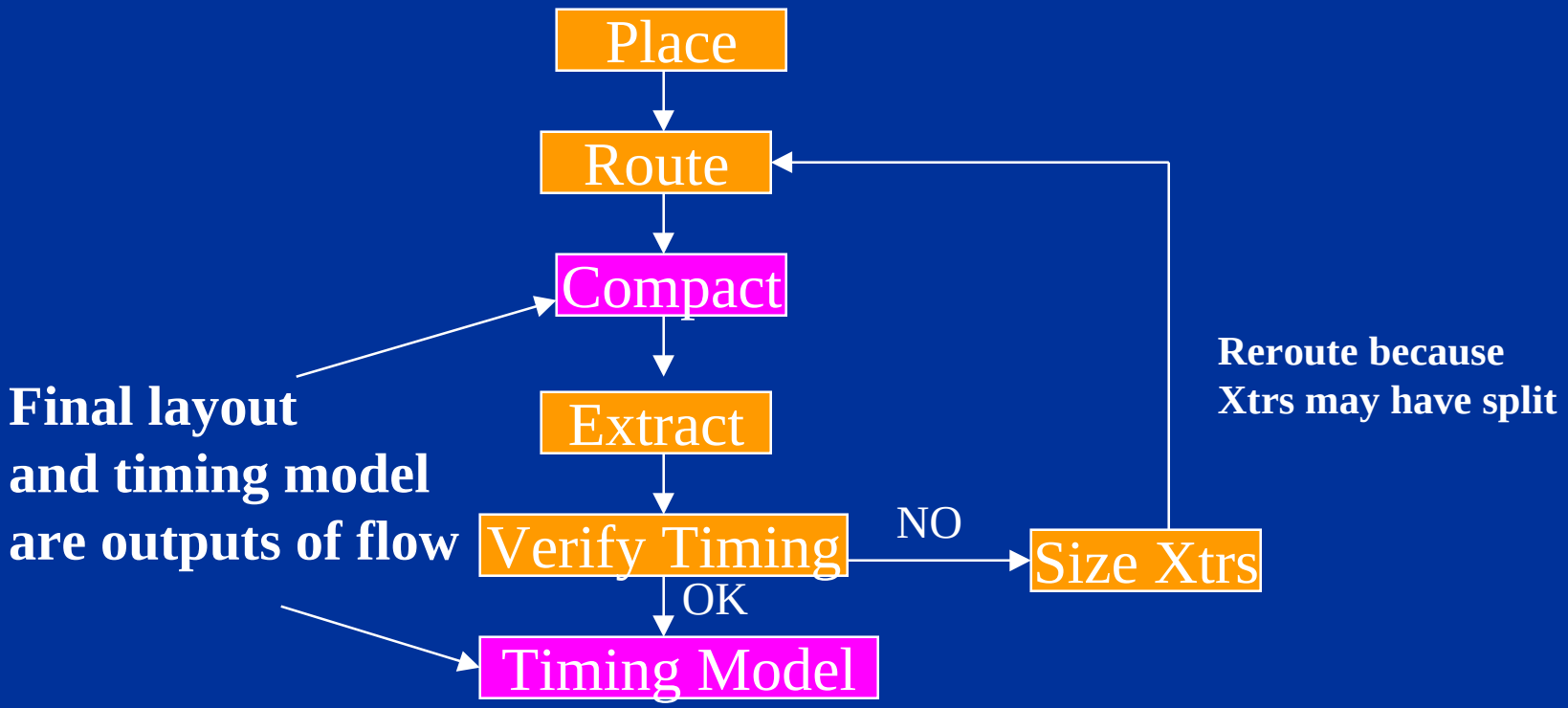
- Build on top of cell layout synthesis
  - Pair and chain transistors to maximize diffusion sharing and minimize cross-overs
- Place transistor chains like cells
  - Quadratic, anneal, min cut etc..
- Use area or symbolic routing to route cells
  - Compact if placement and routing is symbolic

Block Layout Synthesis
Physical vs Symbolic

- Physical
  - Placement, routing based on actual locations
  - Better density, performance predictability
- Symbolic
  - Placement on movable grid. Routing uses lines so always completes. Compaction step required
  - Greater capacity. (Fix topology then compact)

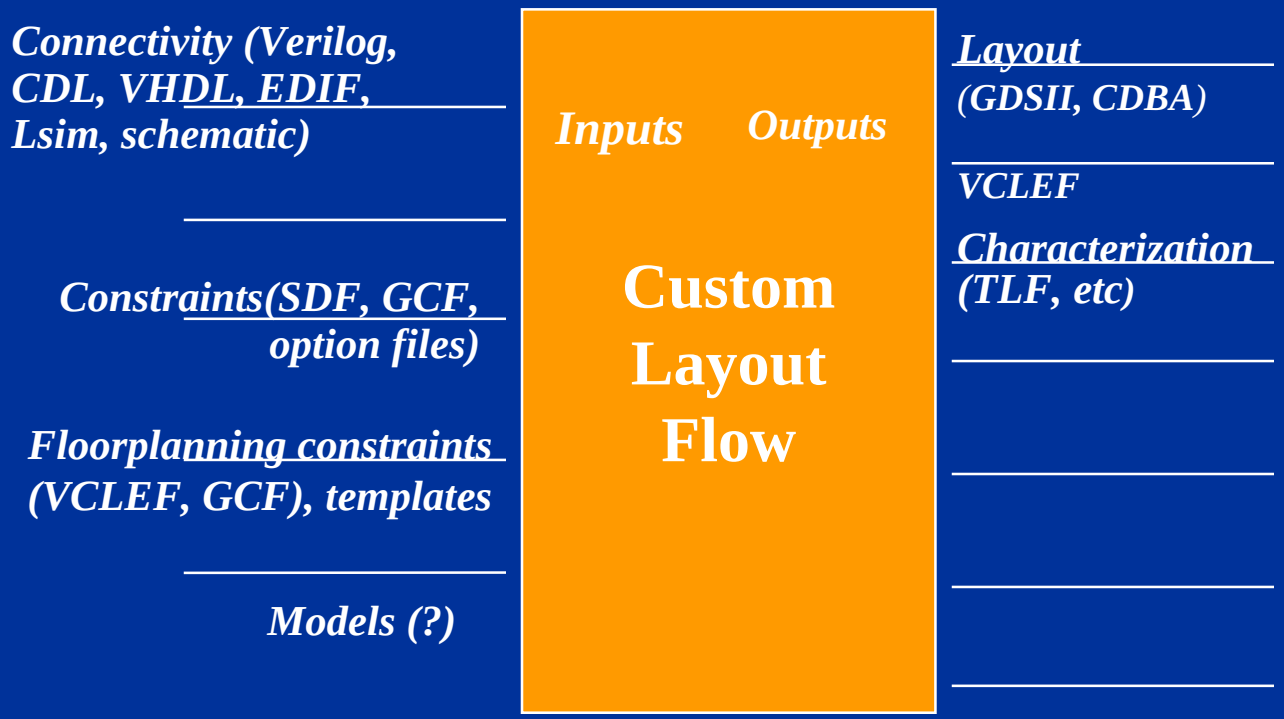
Block Layout Synthesis Optimization Flow (symbolic)

- Symbolic flow is well suited for Optimization



Custom Layout Flow

- Block Inputs and Outputs

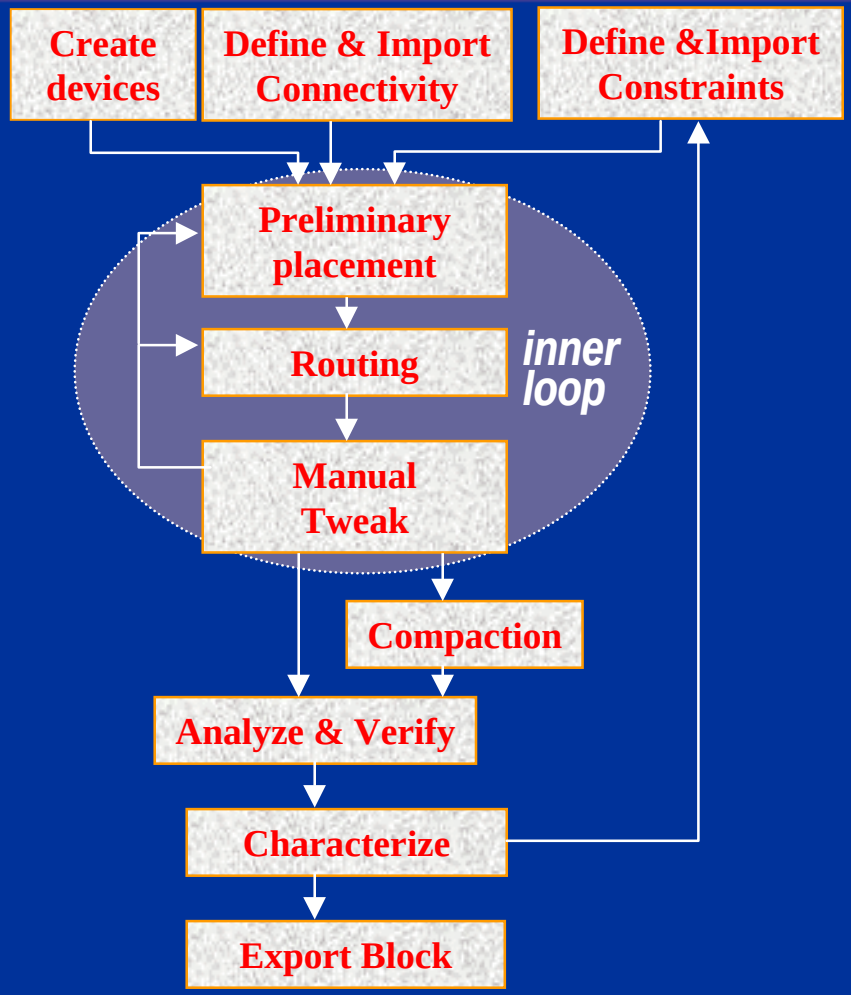


Connectivity Driven Constraint Assisted Layout

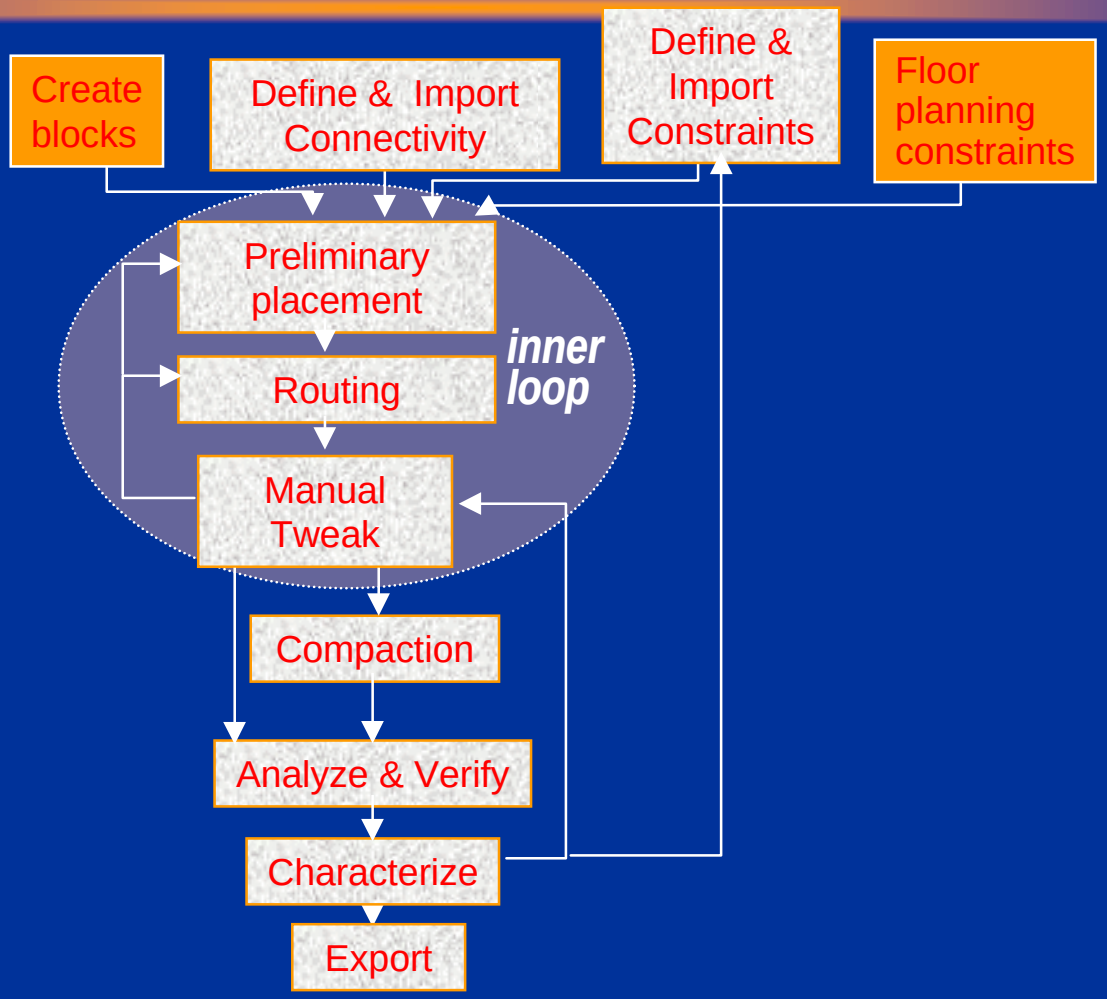
- Maintain Existing Flow Handoffs
- Flow decomposition based on existing tools
  - What layout designers are used to
- Help guide users/tools
  - Use connectivity
  - Assist enforcement of constraints
- Find/Fix violations ASAP
  - Do not wait till verification phase to catch errors

Custom Layout Flow Intra-block

Tasks

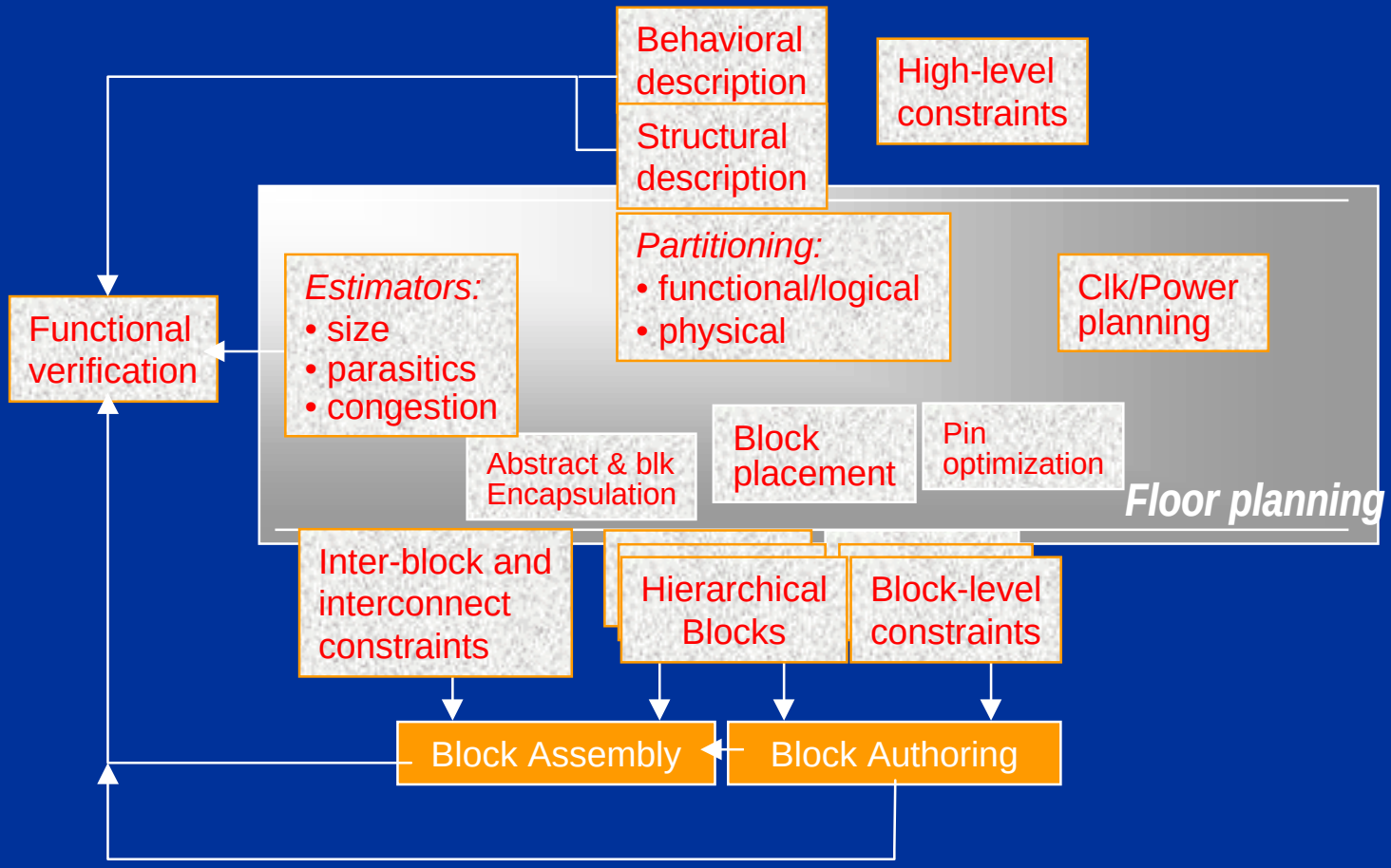


Custom Layout Flow Inter-block

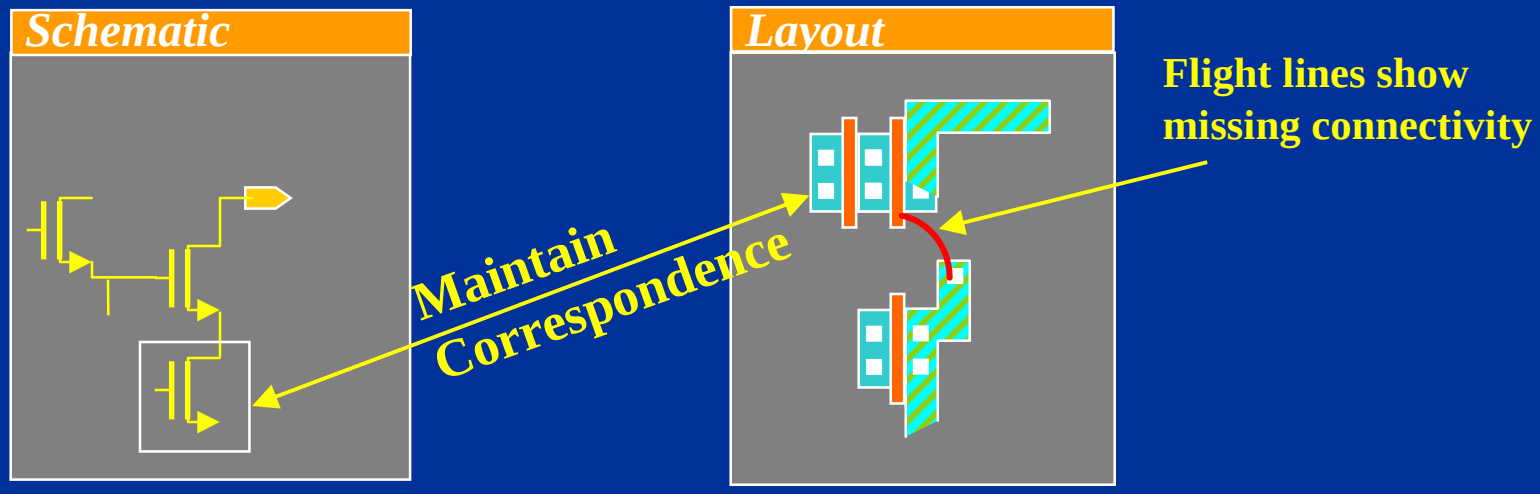


Tasks

Top Down Custom Flow

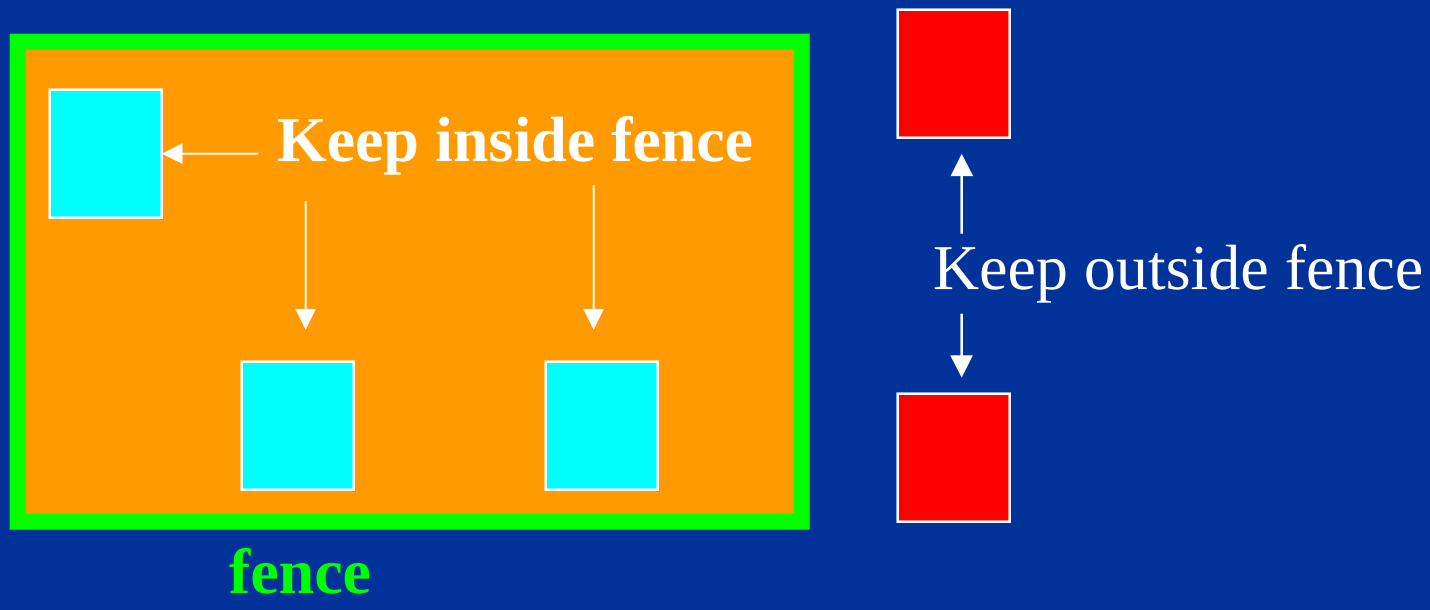


Connectivity Driven Layout Example



- On line connectivity checking
- Correct by construction (size, props., etc)

Interactive Constraint Enforcement Example



Connectivity Driven Constraint Assisted Layout: Summary

- Constraints and Connectivity
  - Drive Back End to reduce errors
  - Reduce overall time by making smaller iteration loops
  - Early detection of errors
  - Easy checking and updates
- Large improvement in productivity