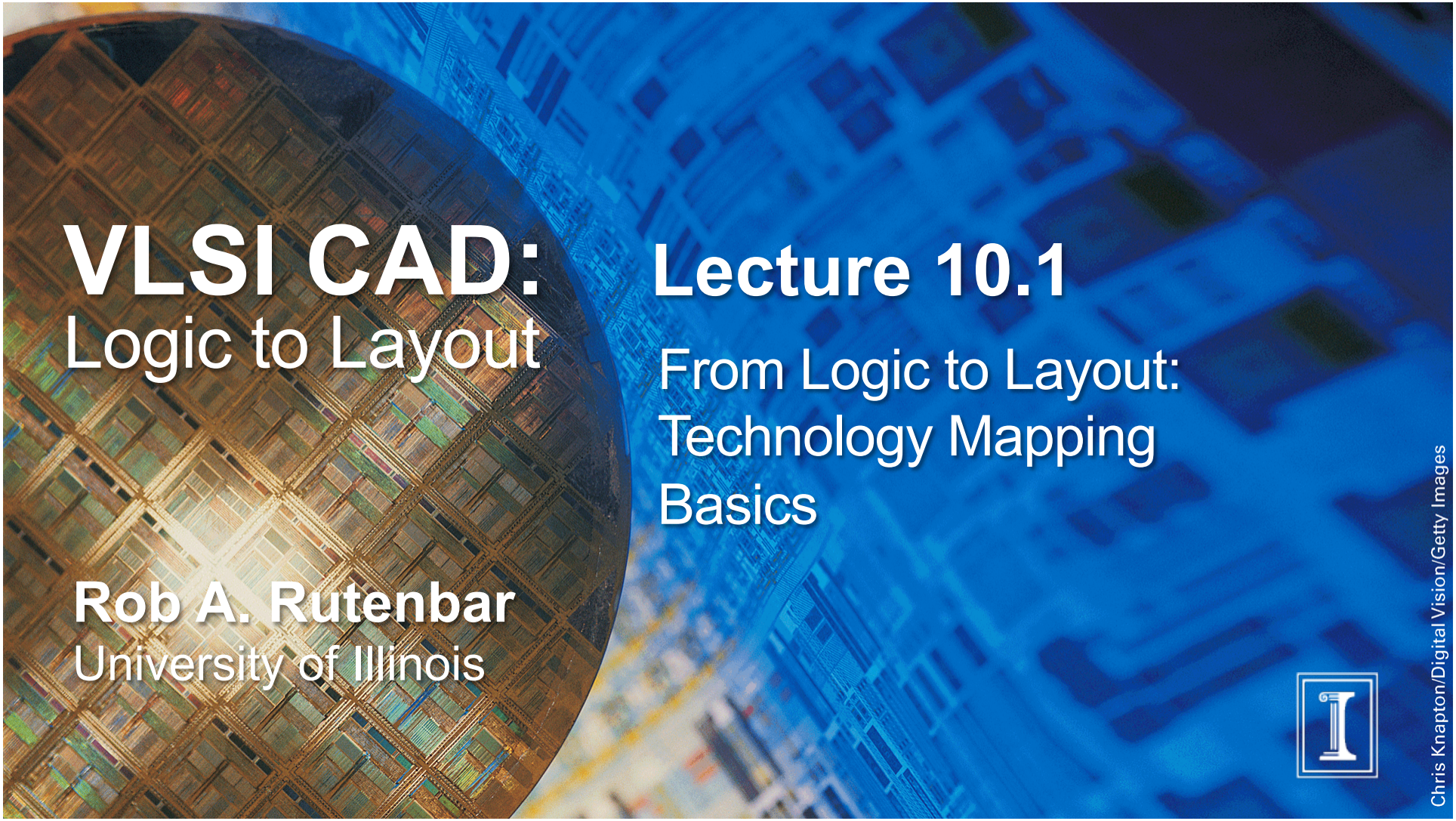


VLSI CAD: Logic to Layout

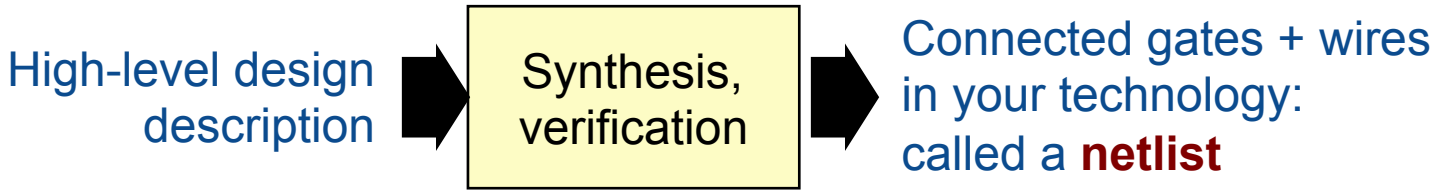
Lecture 10.1

From Logic to Layout: Technology Mapping Basics

Rob A. Rutenbar
University of Illinois

From Logic… To Layout…

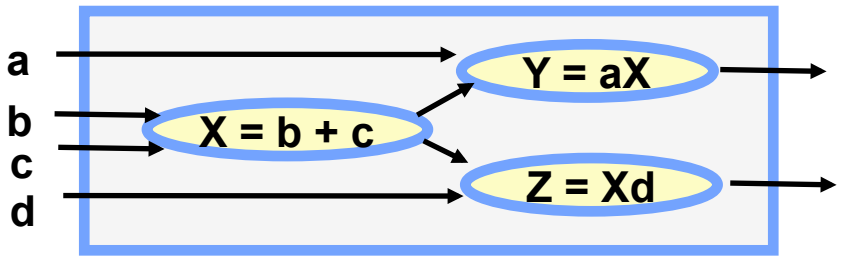
- **What you know...**
  - Computational Boolean algebra, representation, some verification, some synthesis
  - This is what happens in the **"front end"** of the ASIC design process



- **One key missing step:**
  - How to **transform** result of multi-level synthesis into real gates for layout task
  - Called: **Technology Mapping**. Or **Tech Mapping**. Or **Mapping, Fitting, Binding**
    - Reference: Giovanni DeMicheli, *Synthesis and Optimization of Digital Circuits*, McGraw Hill, 1994, Chapter 10

Tech Mapping: The Problem

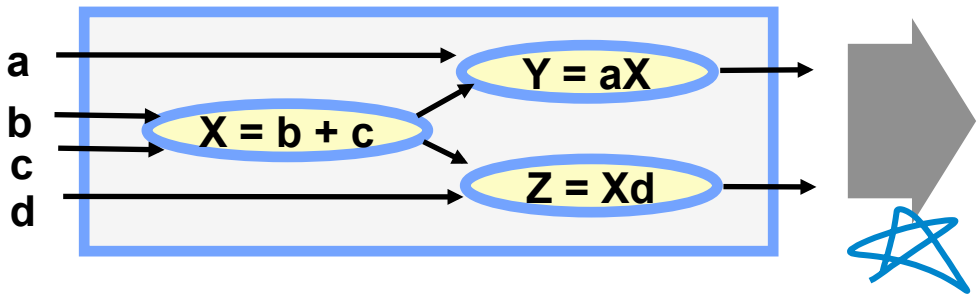
- **Multi-level model is still a little abstract**
  - Structure of the Boolean logic network is fixed
  - ESPRESSO-style 2-level simplification done on each node of network...
  - But that still does **not** give us the actual gate-level netlist
- **Trivial example**



Tech Mapping: Problem

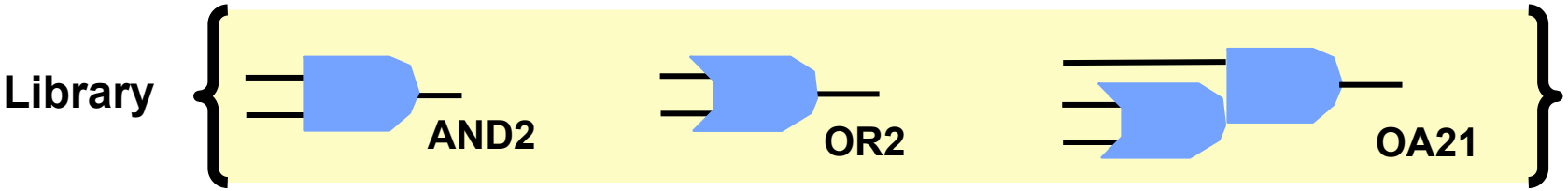
- **Suppose we have these gates in our "library"**
  - This is called "**the technology**" we are allowed to use to build this optimized netlist

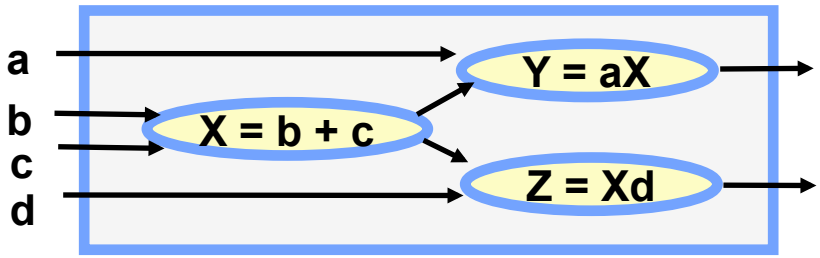
  - **OA21** is an OR-AND, a so-called **complex gate** in our library

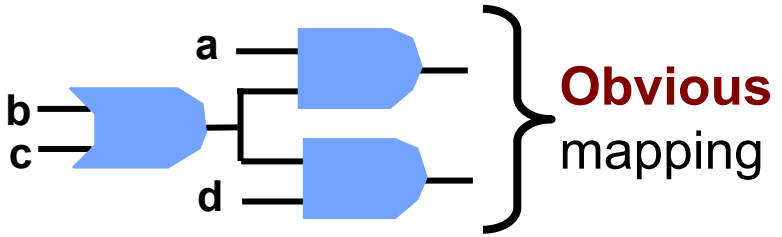


How do we build the 2 functions specified in this Boolean Logic Network using **only** these gates from **our library**?

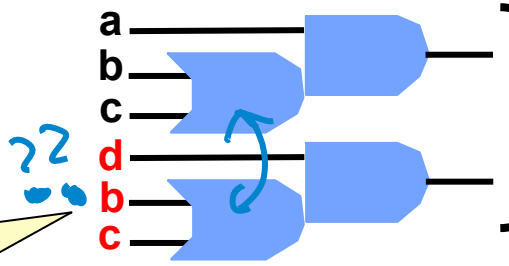
Tech Mapping: Simple Example







Obvious mapping

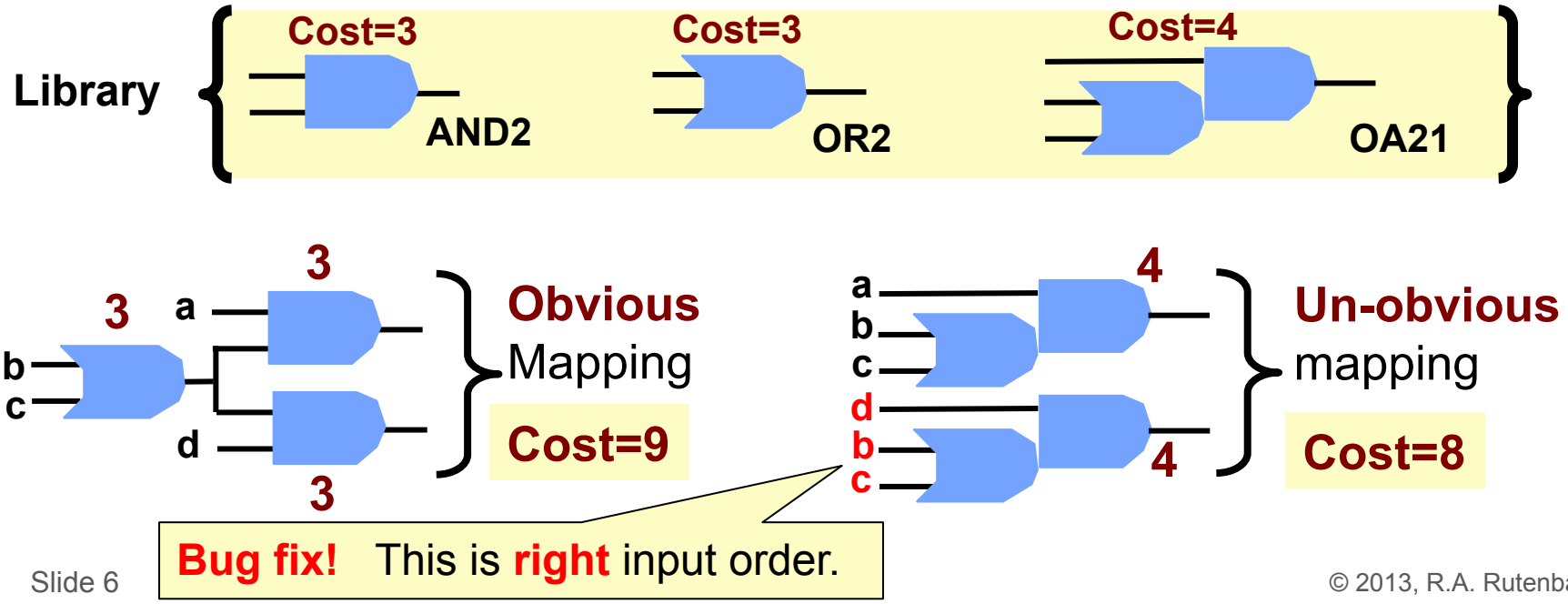


Un-obvious mapping

Bug fix! This is the **right** input order. Original slide had **b** as input to **AND**, which is wrong

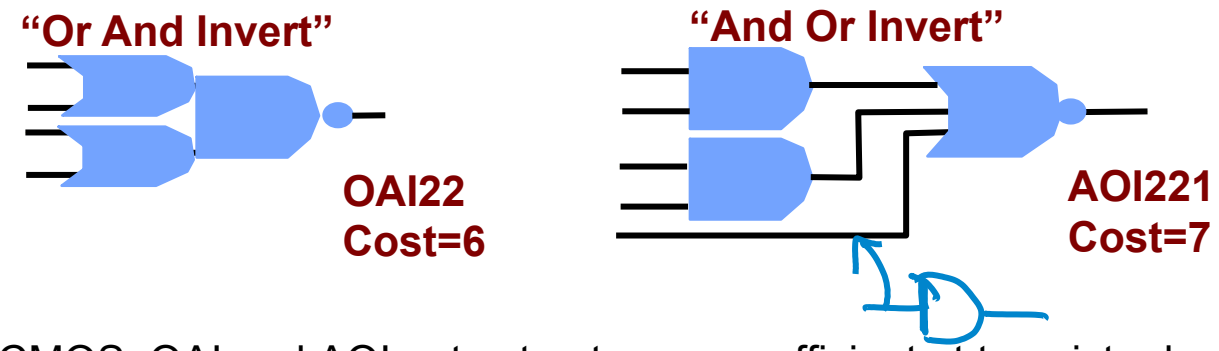
Tech Mapping: Simple Example

- **Why choose a non-obvious mapping?**
  - **Answer 1:** **Cost**. Suppose each gate in library has a cost associated with it.
  - Think of this as, eg, the **silicon area** of the standard cell gate



Tech Mapping

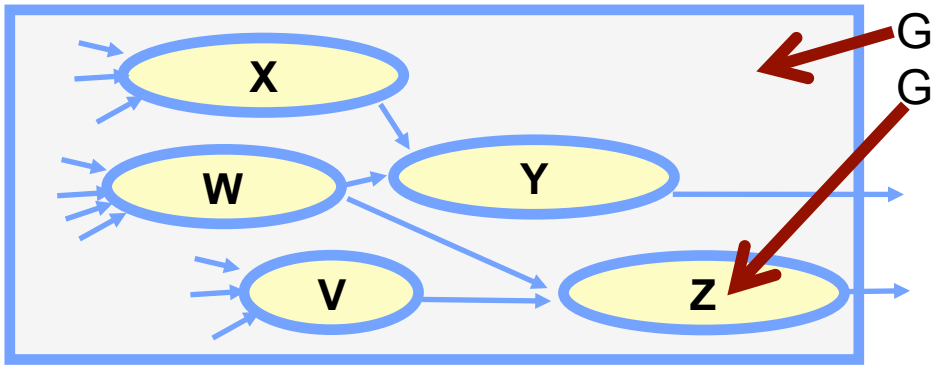
- **Why choose a non-obvious mapping?**
  - **Answer 2:** **Complexity** (which really is just cost, in a *different* way)
  - Your library might have complex gates that are hard to map "by eye"



  - In CMOS, OAI and AOI gate structures are efficient at transistor level

What "Multilevel Synthesis" Does…

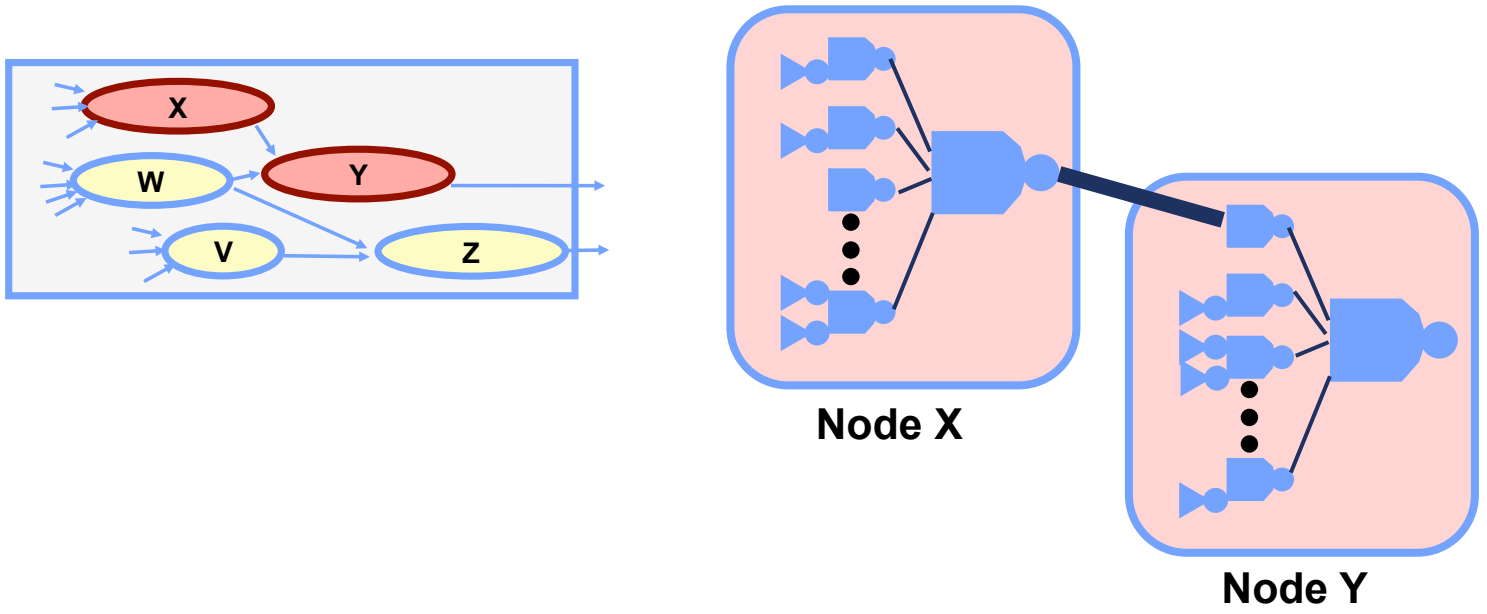
- **Helpful "mental" model to use: Multi-level synthesis does this...**
  - **Structures** the multiple-vertex Boolean logic network **"well"**
  - **Minimizes SOP** contents of each vertex in the network **"well"**, ie, # of literals
  - But it does **not create real logic gates**
  - This result is called: **"uncommitted"** logic, or **"technology independent"** logic



Good **global** structure (nodes)
Good **local** structure (SOP in nodes)

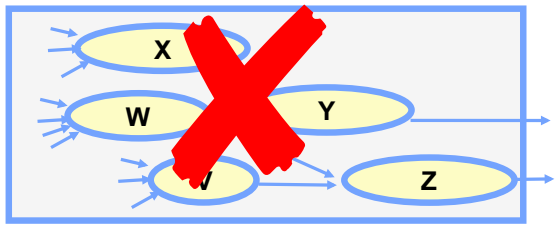
What "Technology Mapping" Does

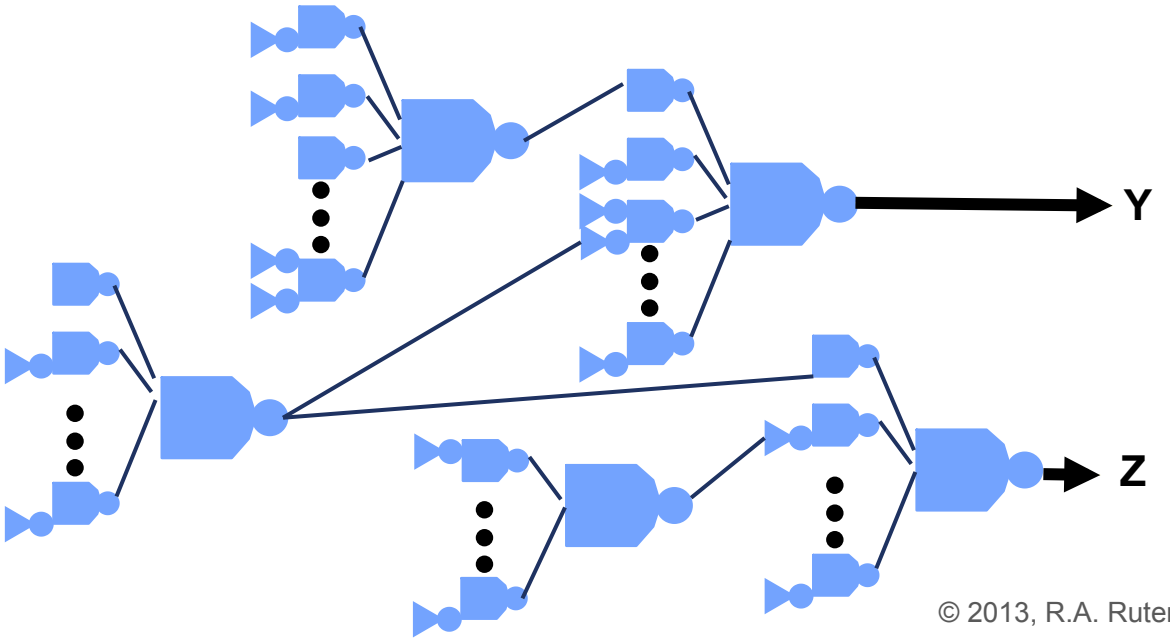
- **Model**
  - First, we transform uncommitted logic into simple, **real** gates
  - We transform **every** SOP form in **each** node into **NAND** & **NOT** gates. Nothing else!



What "Technology Mapping" Does

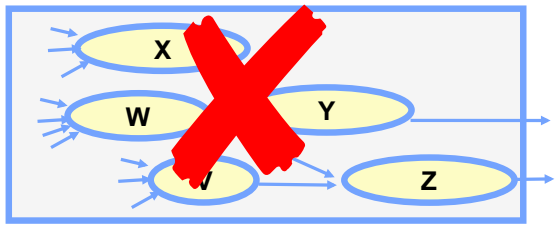
- **Model**
  - **Final result:** Boolean logic network **disappears**. **X, Y, W, V, Z** boundaries go away
  - We have one **BIG** "flat" network of **NANDs** and **NOTs**. This is what we map.

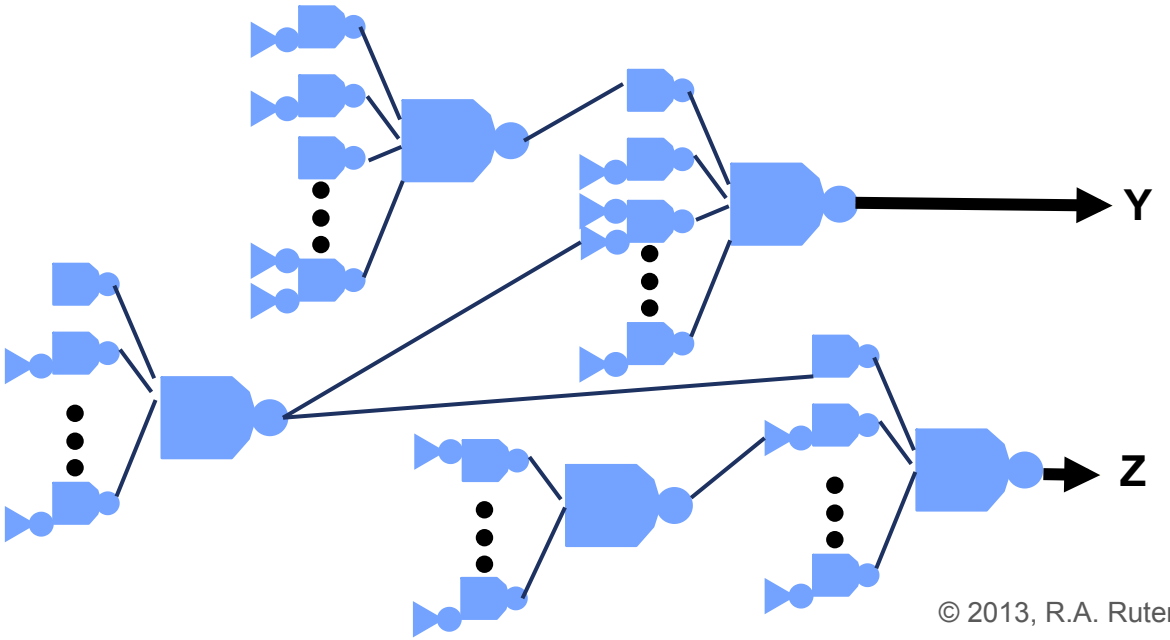


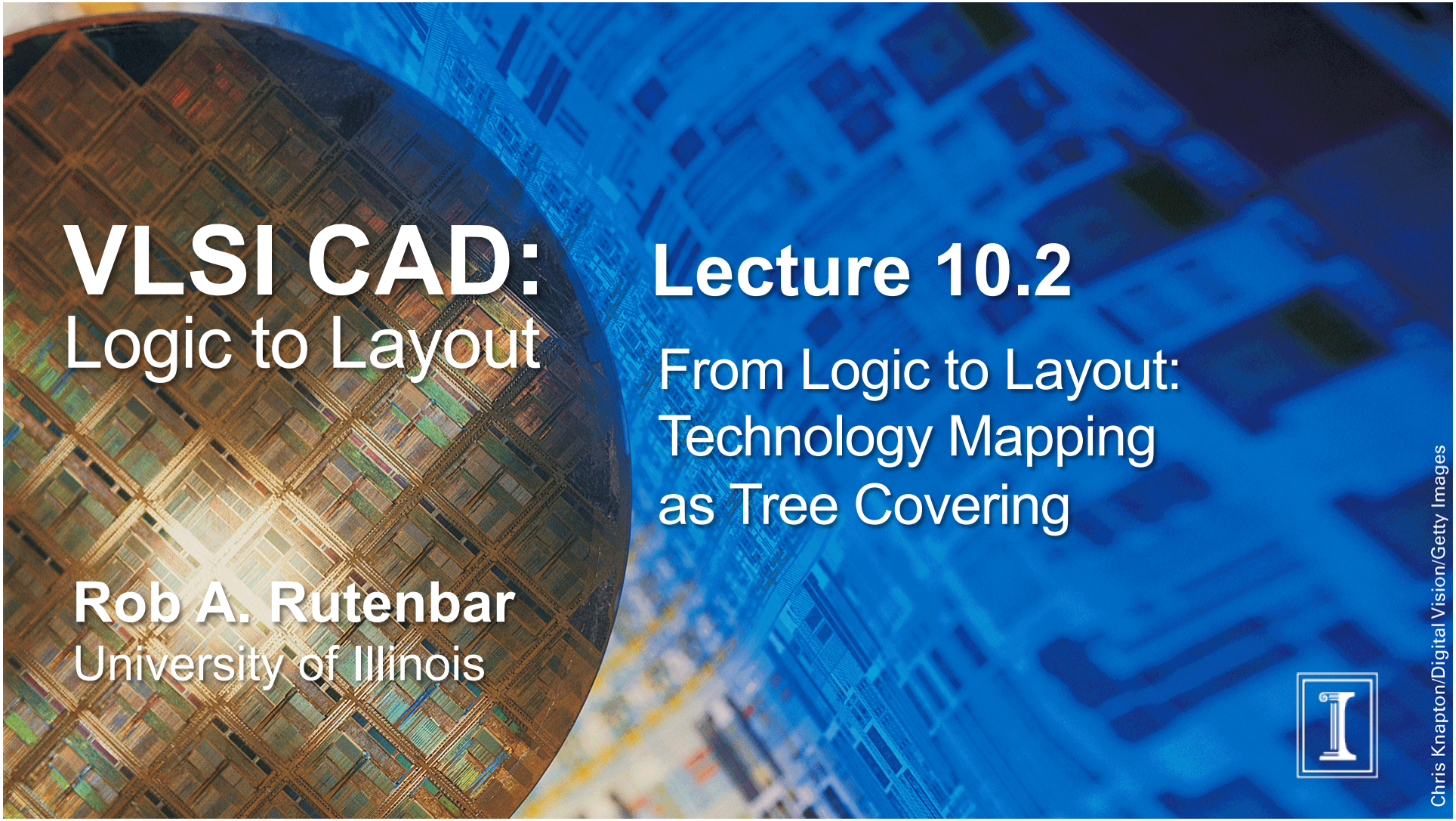


This is Our Starting Point…

- **Multi-level synthesis produces this big network of simple gates**
- **How to transform – map – this onto standard cells in our library?**







VLSI CAD: Logic to Layout

Lecture 10.2

From Logic to Layout:
Technology Mapping
as Tree Covering

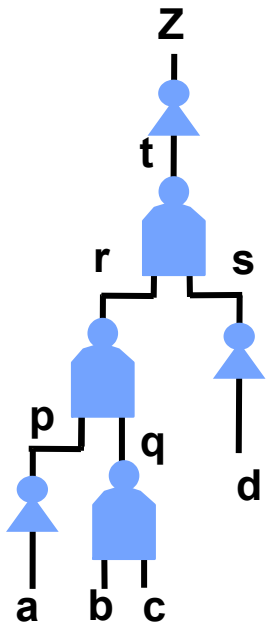
Rob A. Rutenbar
University of Illinois

Technology Mapping as *Tree Covering*

- **One famous, simple model of problem**
  - Your logic network to be mapped is a **tree of simple gates**
    - Easiest to assume absolutely minimal gate types
    - For us: uncommitted form is 2 input **NAND** (“**NAND2**”) and **NOT** gates, only
  - Your library of available “real” gate types is also **“represented”** in this form
    - Each gate is represented as a **tree** of **NAND2** and **NOT** gates, with associated **cost**
- **Method is surprisingly simple and optimal**
  - Original reference: Kurt Keutzer, “DAGON: Technology Binding and Local Optimization by DAG Matching,” Proc. ACM/IEEE Design Automation Conference (DAC), 1987

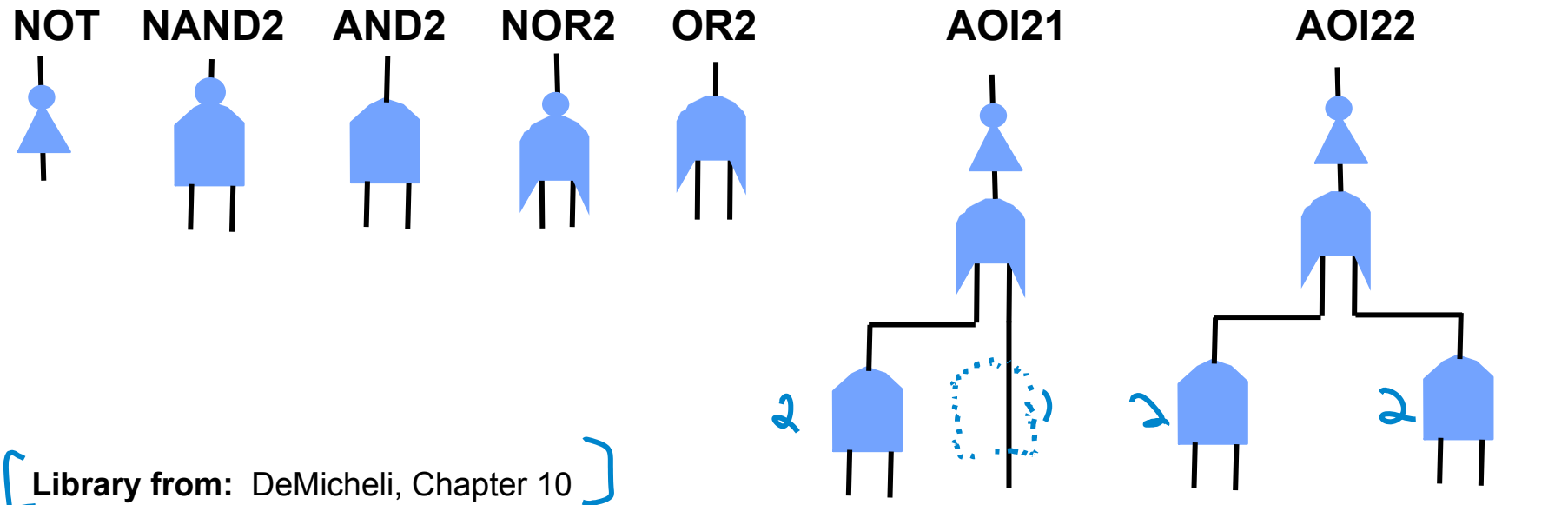
Tree Covering Example: Start with…

- **Here is your uncommitted logic to be tech mapped**
  - This is what results from our multilevel synthesis optimization…
  - … after replacing all SOP forms in the network nodes with **NAND/NOT**
  - Called the **subject tree.**
  - (Restrict to NAND2 to keep this simple, to map this example in detail. Also label not only inputs but all internal wires too.)



Tree Covering: Your Technology Library

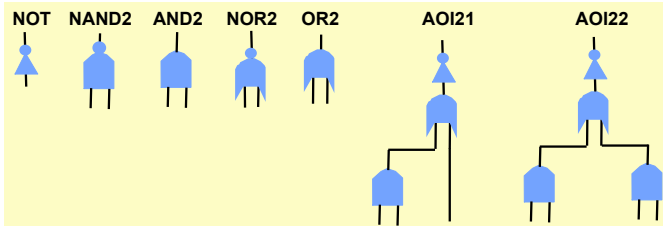
- **And, here is a very simple technology library**
  - **First problem:** this is **not** in the required **NAND2/NOT-only** form. Must **transform**

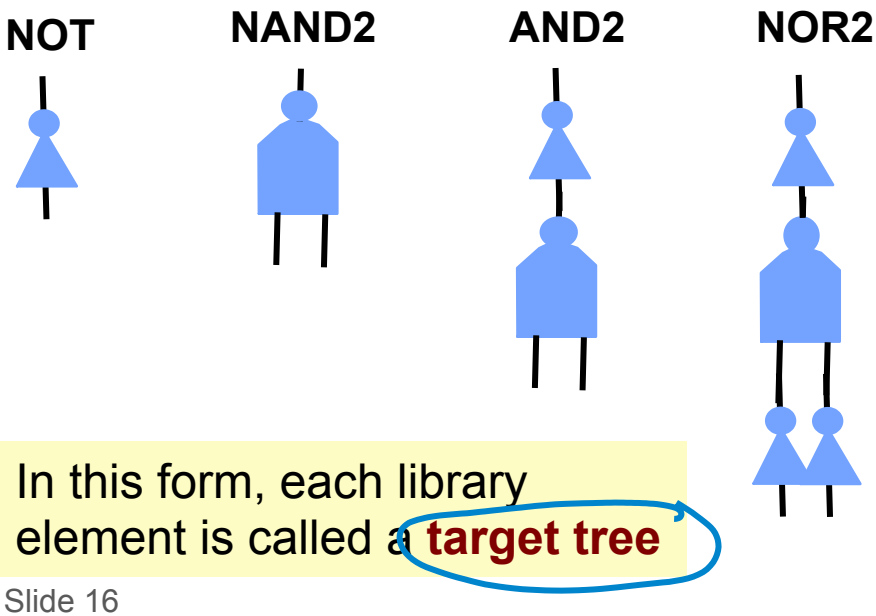


Library from: DeMicheli, Chapter 10

Tree Covering: Representing Library

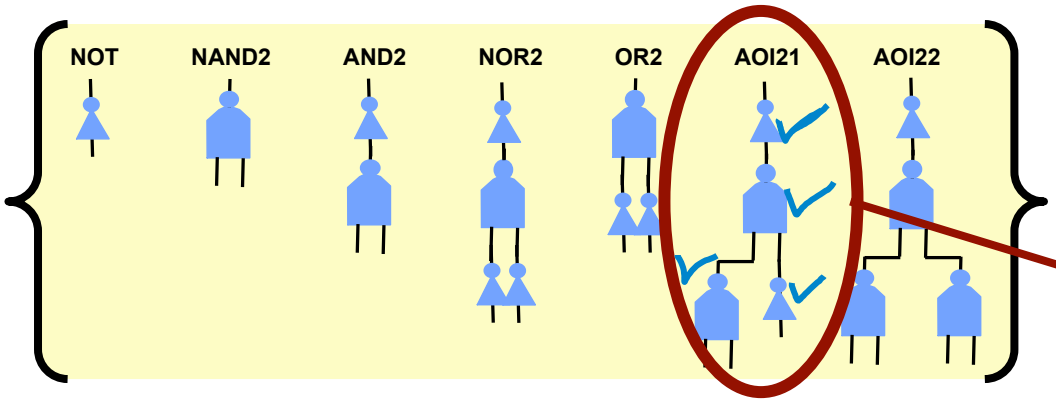
- **Transforming to NAND2/NOT form is easy**
  - Just some DeMorgan law Boolean algebra.

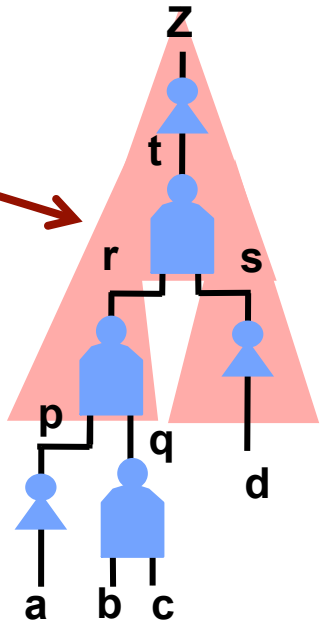




In this form, each library element is called a **target tree**

Essential Idea in "Tree Covering"



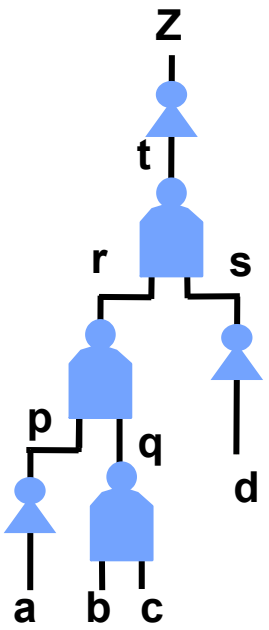


- **No Boolean algebra here!**
- **This is called: structural mapping**
  - Find where, in subject graph, the library pattern "matches" everywhere
  - NAND matches NAND, NOT matches NOT, etc

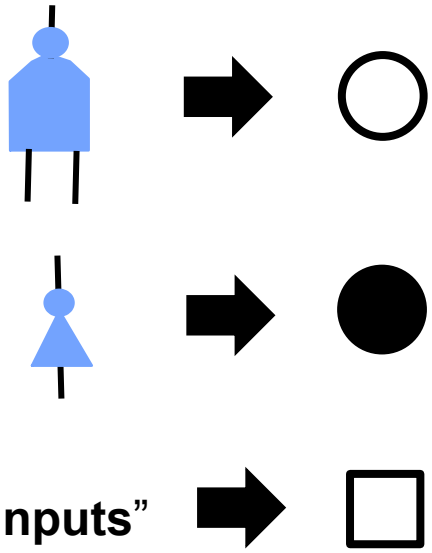
The General Tree Cover Representation

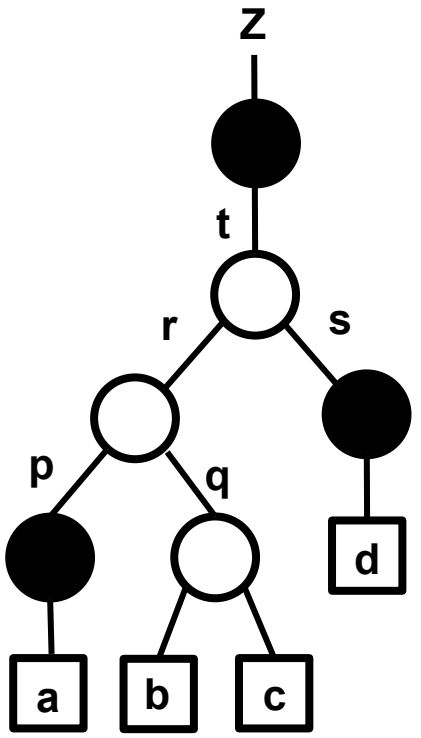
- **Again: this is called a structural tech mapper**
- **Why?**
  - Because there is no Boolean algebra anywhere in here!
  - We just match the gates, wires in a simple **pattern-match** way
- **Result**
  - Surprisingly simple covering algorithm for cost-optimal cover
  - But first, lets simplify the way we draw these, to emphasize “structural” match

Representing Trees for Covering

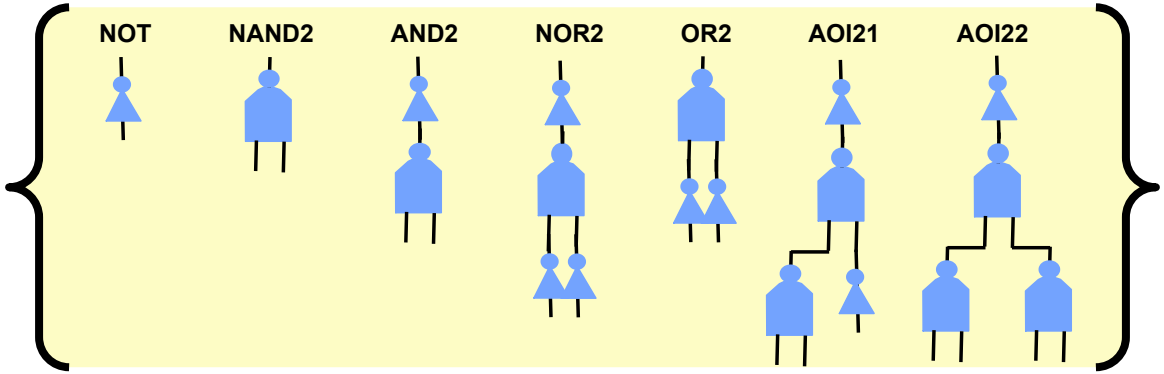


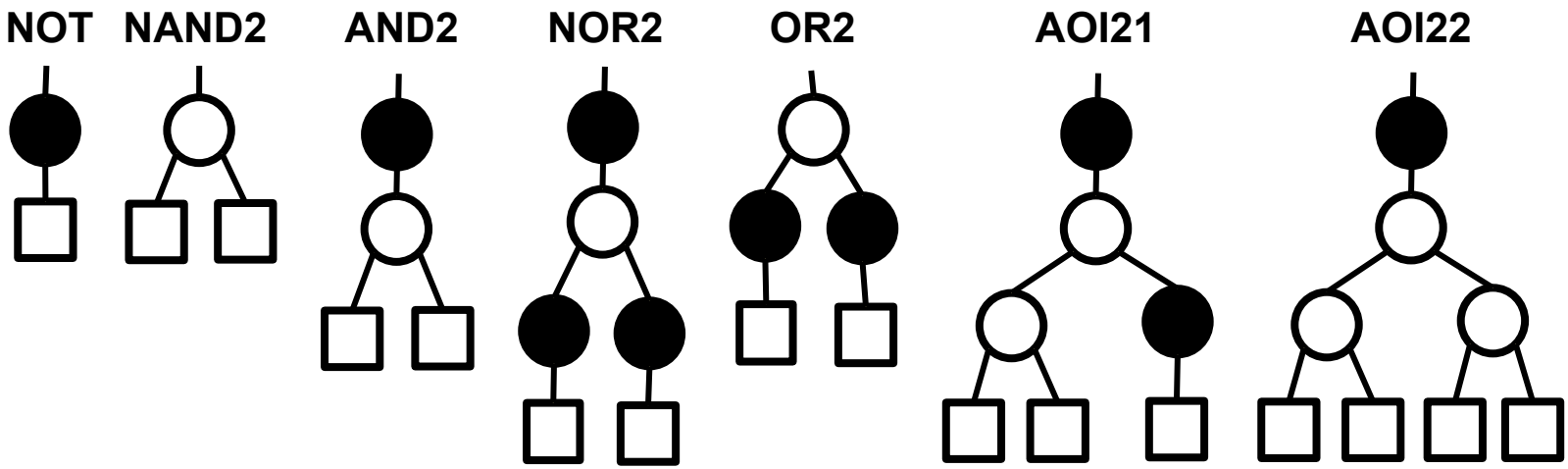
Only **three** kinds of structures
that we need **match** in any tree



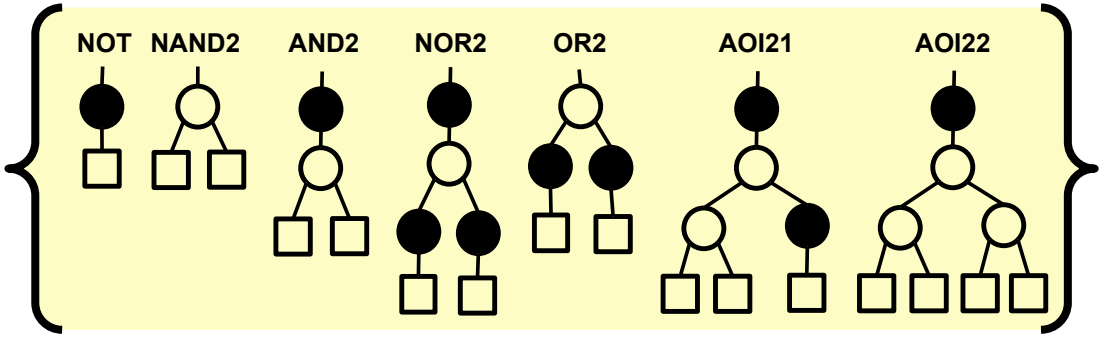


Represent Library in this *Same* Style

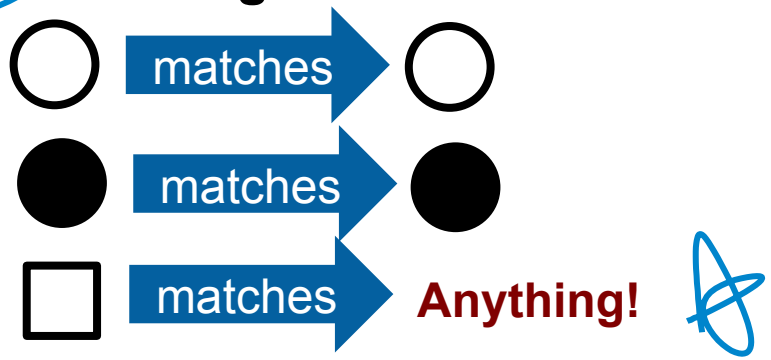


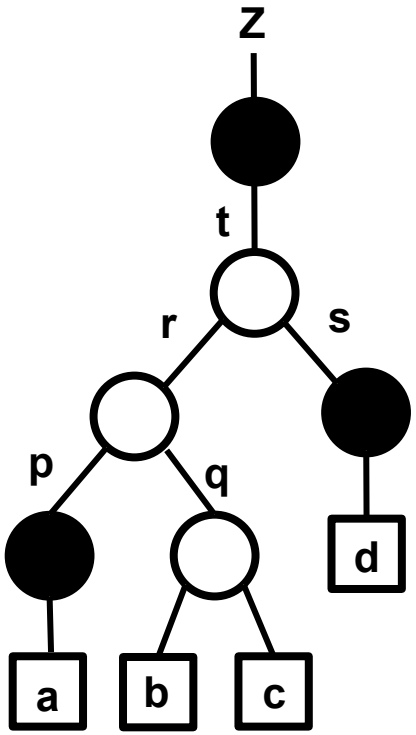


Structural Mapping Problem: Revisited

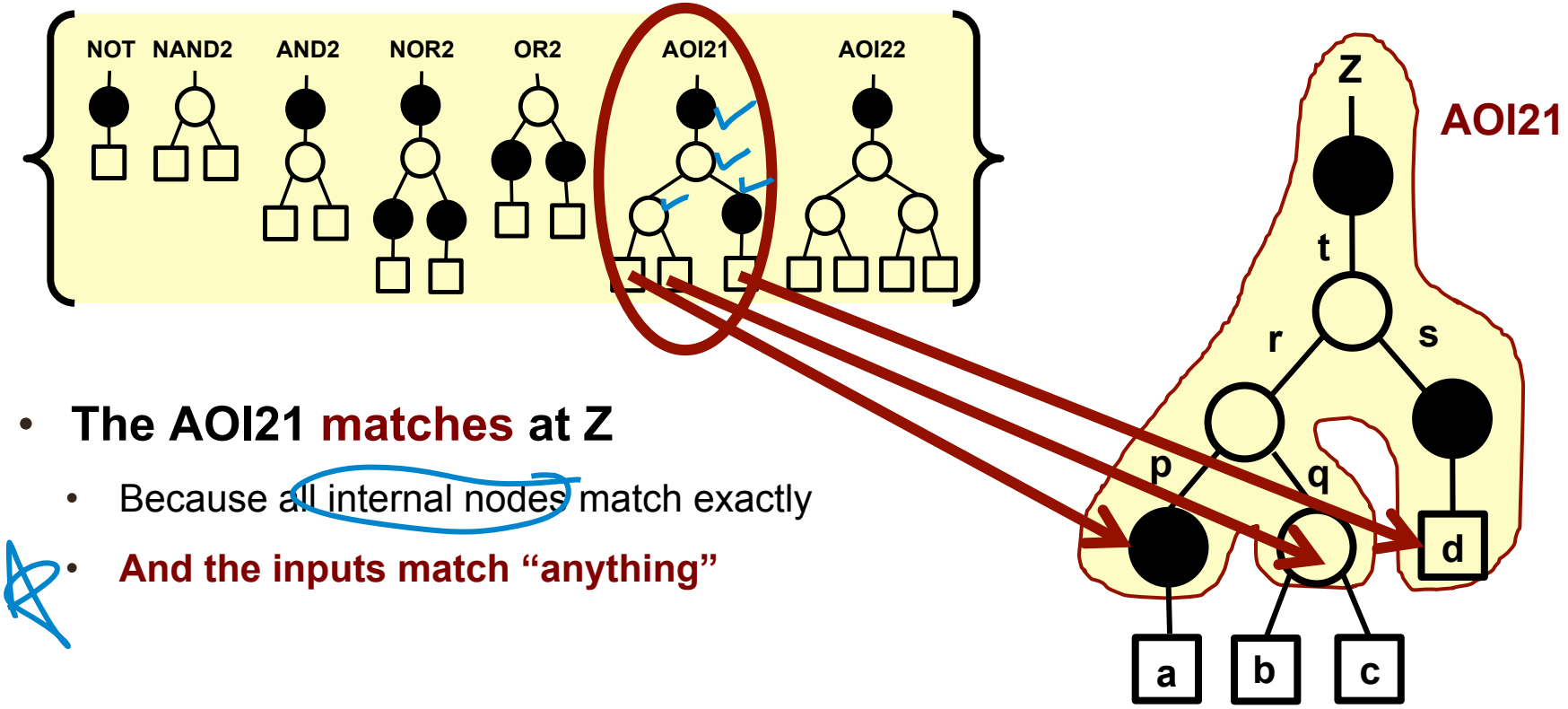


- **No Boolean algebra here at all!**
- **Structural matching rules:**



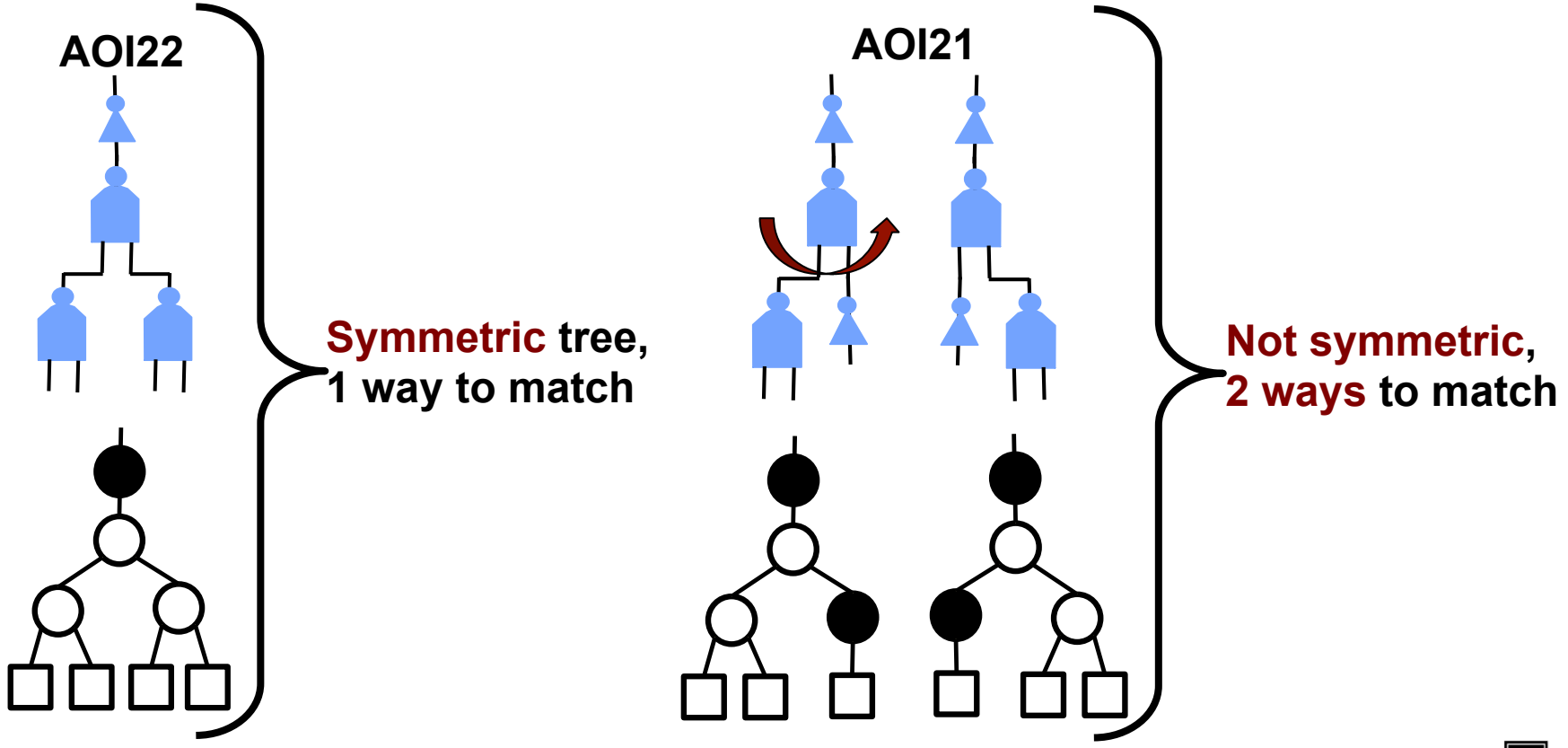


How a "Target Gate" Matches Subject Tree



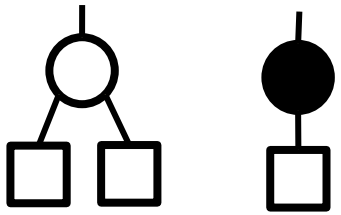
- **The AOI21 matches at Z**
  - Because all internal nodes match exactly
  - **And the inputs match "anything"**

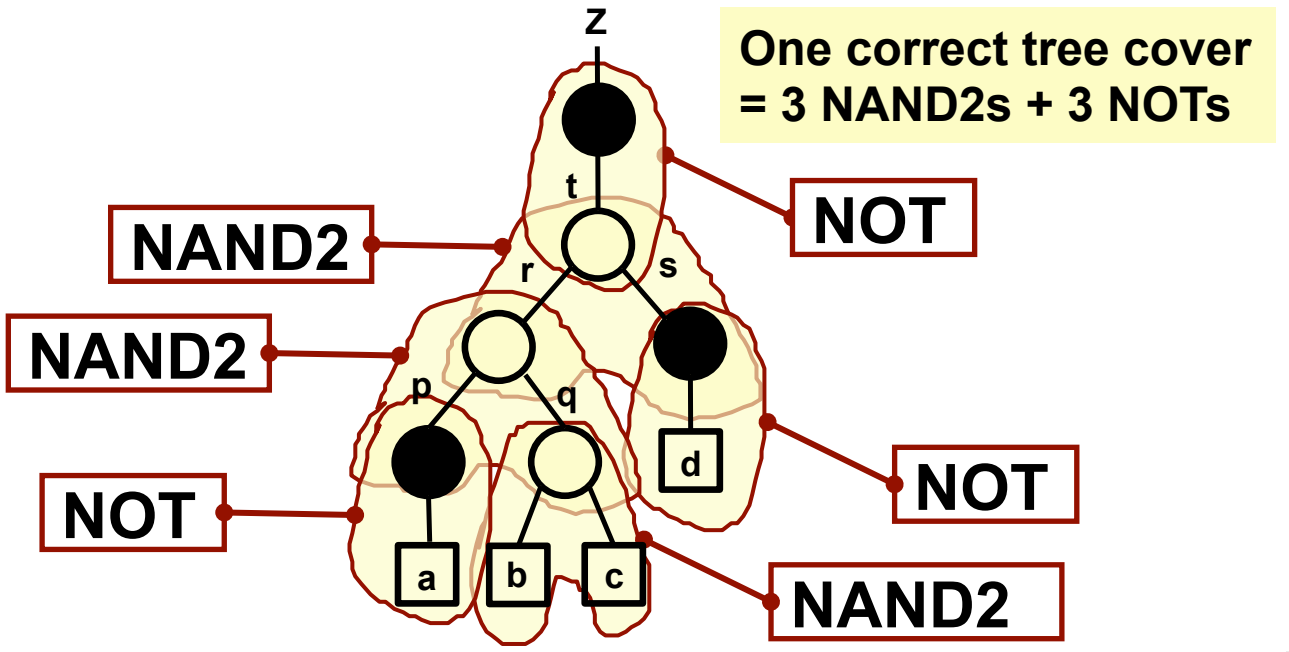
Be Careful: Symmetries Matter!



Rules for a *Complete* Tree Cover

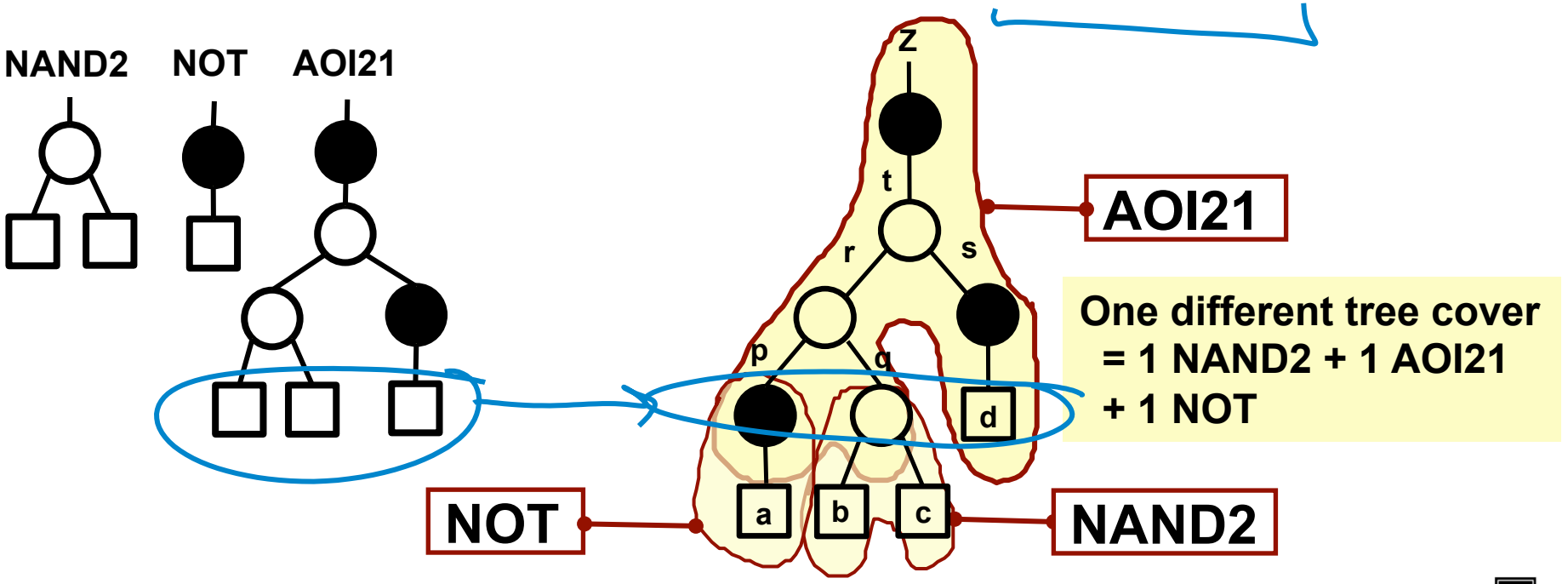
- **Every node in subject tree is covered some library tree**
- **Output of every library gate overlaps input of next library pattern**

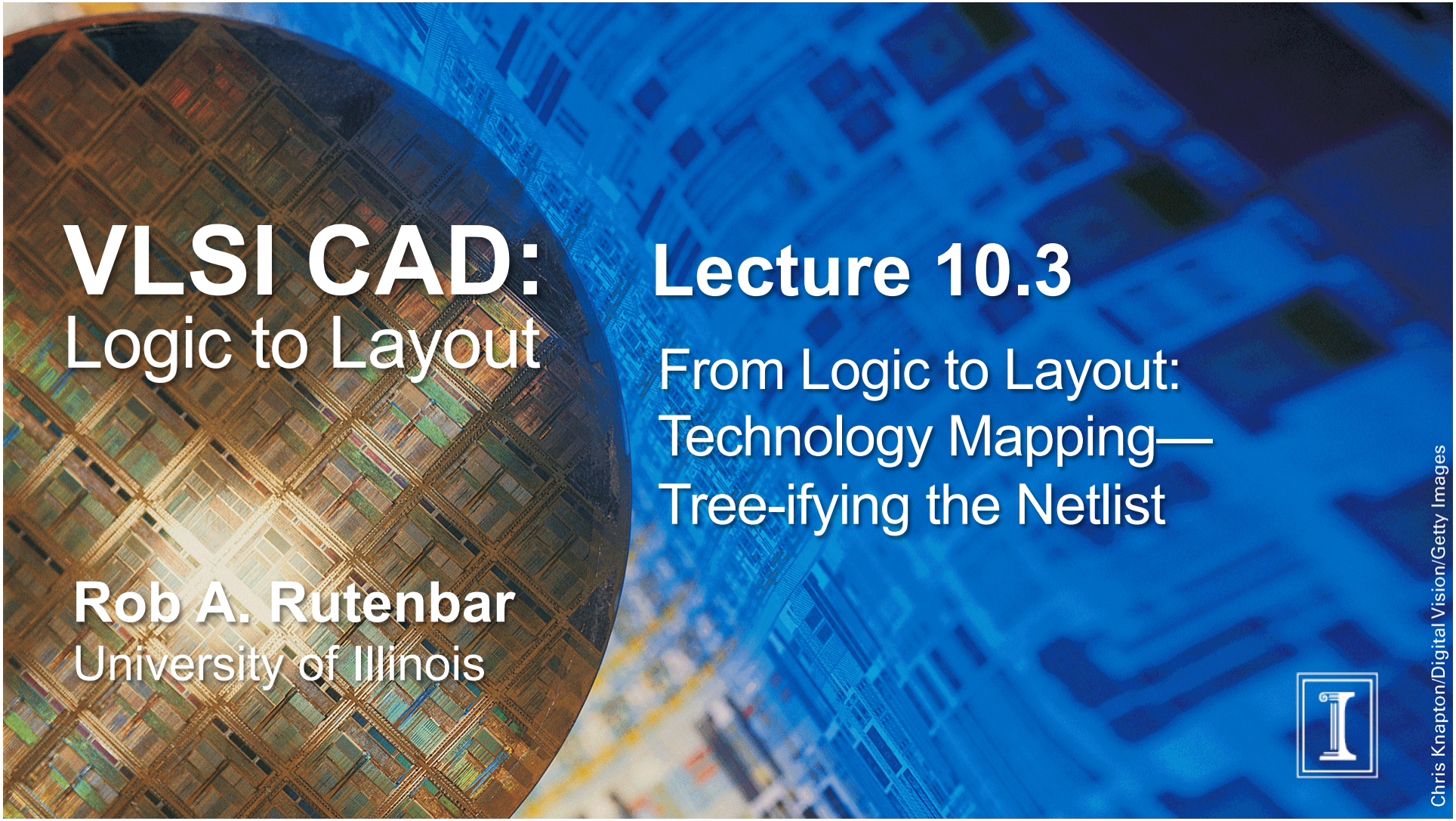




Rules for a *Complete* Tree Cover

- **Note: usually there are many different legal covers**
- **Which one do we choose? The one with minimum cost**





VLSI CAD: Logic to Layout

Lecture 10.3

From Logic to Layout:
Technology Mapping—
Tree-ifying the Netlist

Rob A. Rutenbar
University of Illinois

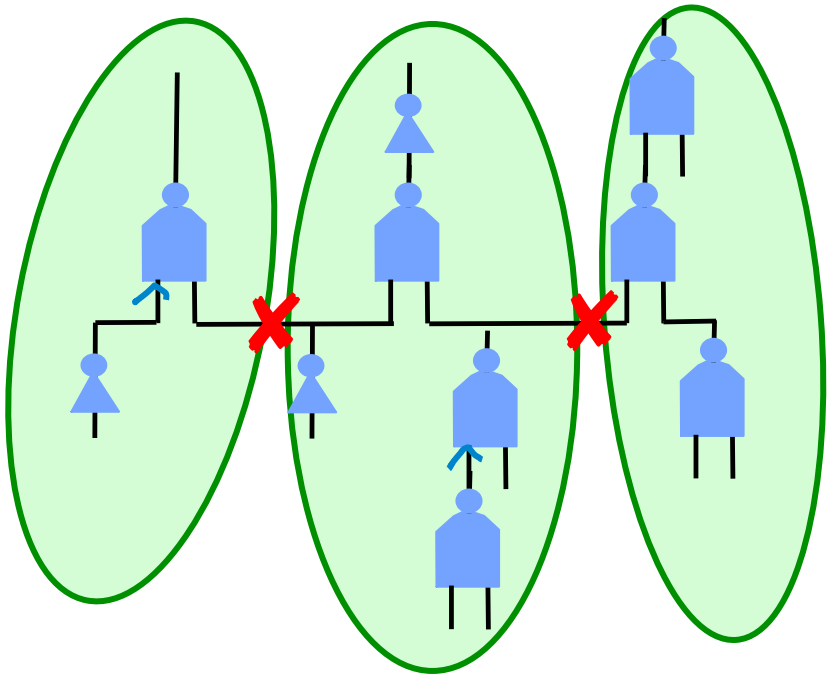
Tech Mapping via Tree Covering

- **What do we need for a complete algorithm?**

- **Tree-ifying the input netlist**
  - One key assumption needs discussion
- **Tree matching**
  - For each node in your subject tree, find the target pattern trees in library match here
- **Minimum-cost covering**
  - Assume you know what can match at each node of subject tree
  - ..so, which ones do you pick for a minimum cost cover?

Tree-ifying the Netlist

- **These algorithms only work on trees, not more general graphs**
  - Gate netlists are **Directed Acyclic Graphs** (DAGs): directed edges, no loops DAGs
  - Every place you see a gate with fanout (> 1 other gate), you **need to split**

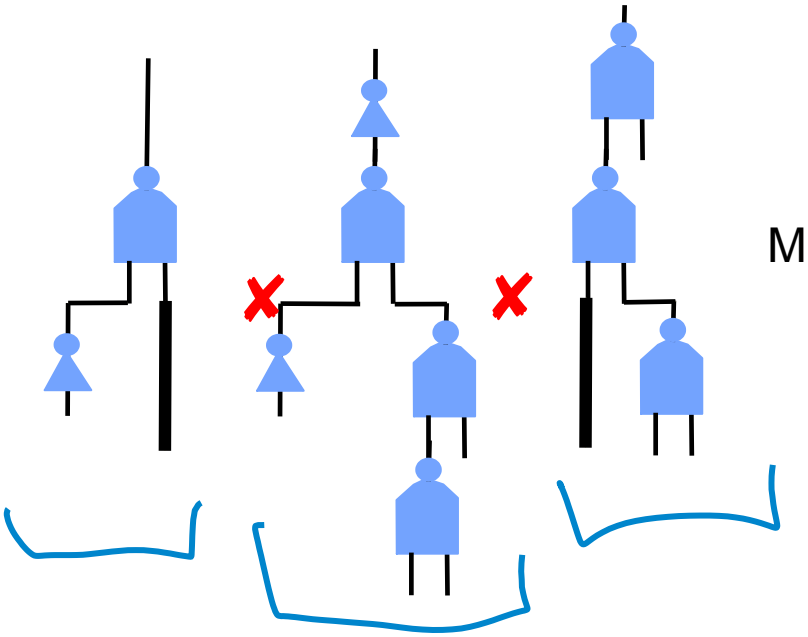


Must **split** this DAG with fanouts into **3 separate trees**, map each separately

This entails some clear loss of optimality, since cannot map across trees

Tree-ifying Netlist: Result

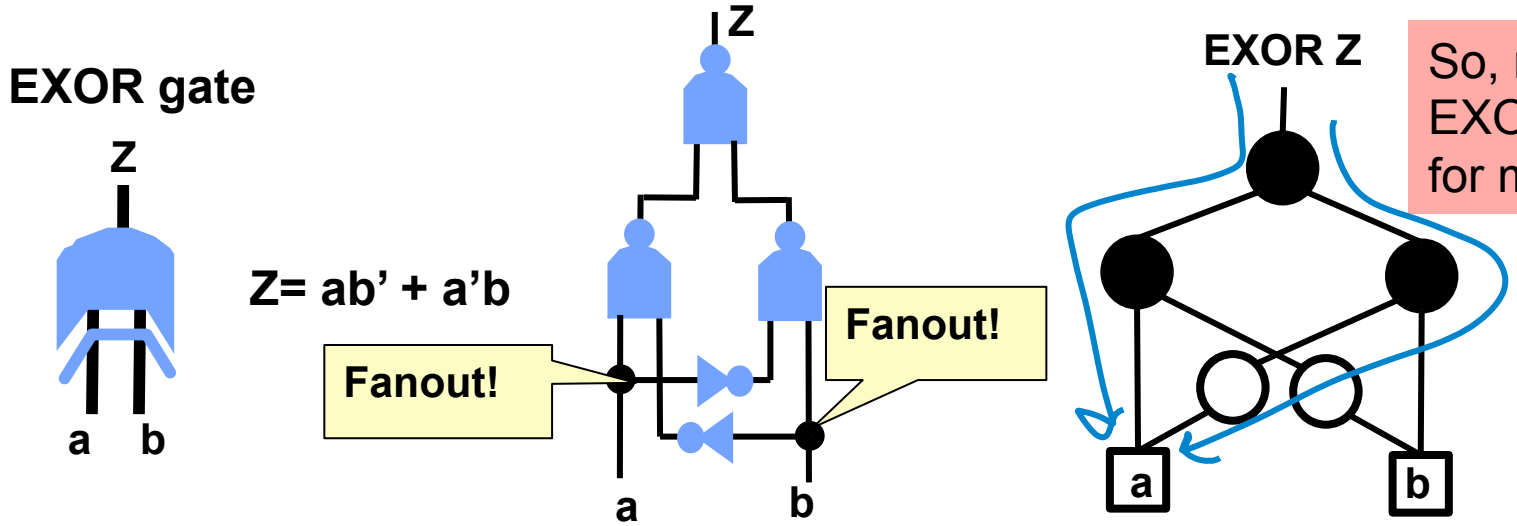
- **Our algorithms mandate trees, not DAGs**
  - Every place there is fanout from gate output > 1, you have to cut
  - Loses some optimality. There are ways around these, but we won't discuss these

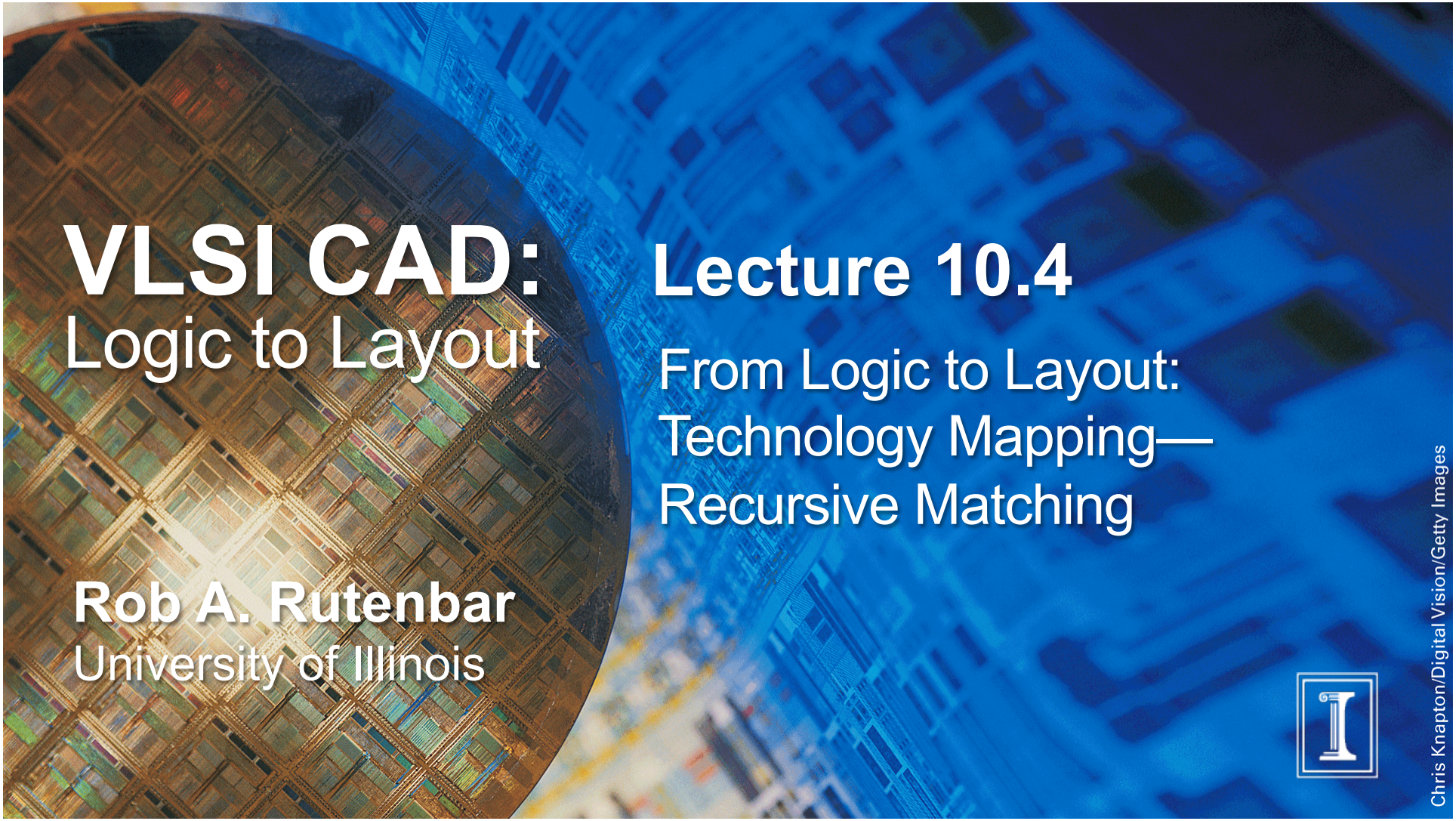


Map these **3 trees** separately

Aside: How Restrictive is "Tree" Assumption?

- **Subject graph and each pattern graph must be trees**
  - Subject tree must be tree-ify-ed. **What about pattern trees?**
  - Are there ordinary, common – useful-- 'gates' that cannot be trees?
  - Yes! There are tricks to deal with this – but for us, these are forbidden!





VLSI CAD: Logic to Layout

Lecture 10.4

From Logic to Layout: Technology Mapping—Recursive Matching

Rob A. Rutenbar
University of Illinois

Tech Mapping via Tree Covering

- **What do we need for a complete algorithm?**

- **Tree-ifying the input netlist**
  - One key assumption needs discussion
- **Tree matching**
  - For each node in your subject tree, find the target pattern trees in library match here
- **Minimum-cost covering**
  - Assume you know what can match at each node of subject tree
  - ..so, which ones do you pick for a minimum cost cover?

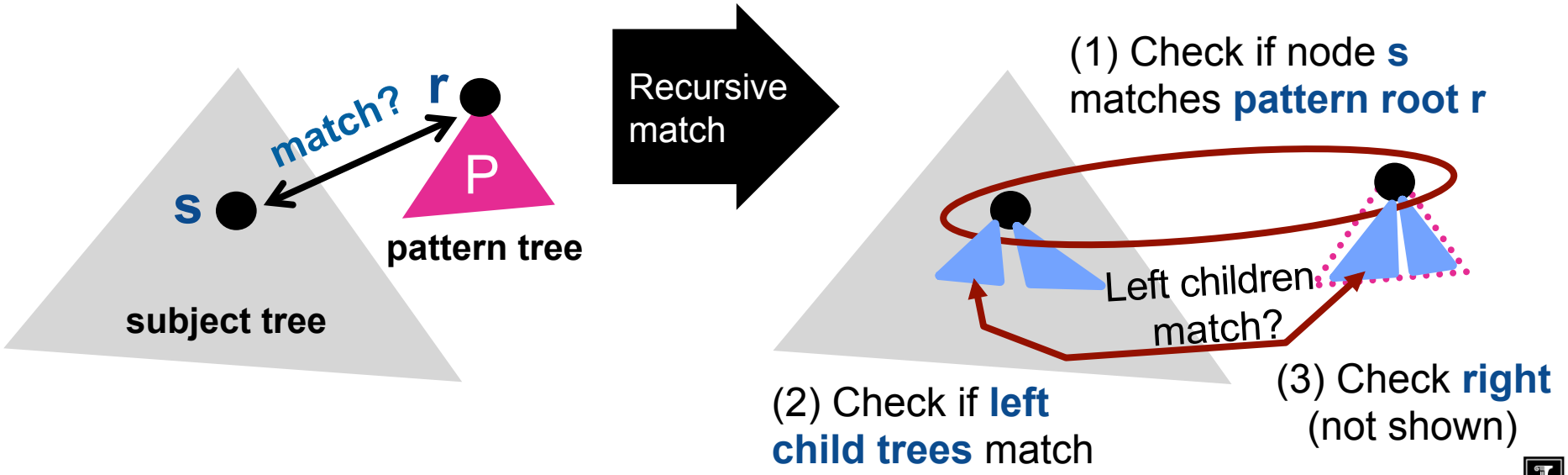
Tree Matching

- **Goal:**
  - Determine, for every node in subject tree, what library gate can **match** (structurally)

- **Straightforward approach: Recursive matching**
  - Simple idea is to just try **every library gate** at **every node of subject tree**
  - Library gates are small patterns – this is not too much work
  - **Recursive means:** match **root** of subject with **root** of pattern, and then recursively match **children** of subject to children of pattern

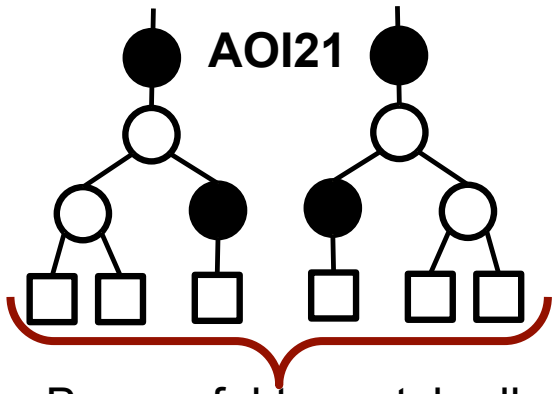
Recursive Tree Matching

- **Does library pattern tree P match node s in our subject tree?**
  - First, check if node **s** matches **r=root(P)** node of pattern **P**
  - If so, **recursively** match **left child tree** of **s** and **r**, and then **right child trees**

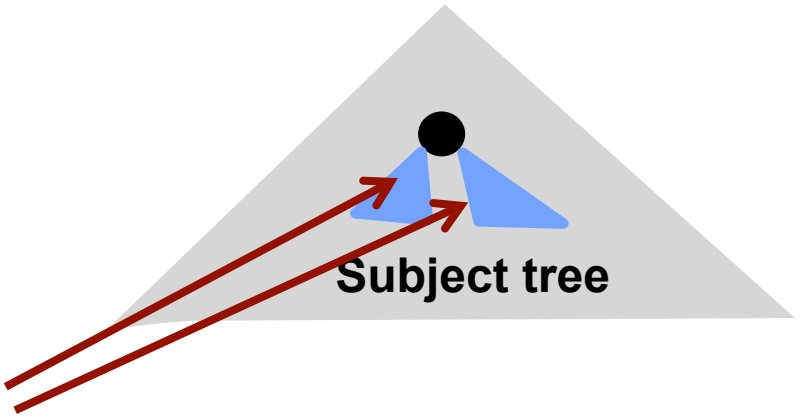


Tree Matching: One Subtlety

- **Be careful matching asymmetric library patterns**
  - Our example was AOI21. Need to remember to check all possible match possibilities by "**rotating**" the pattern tree

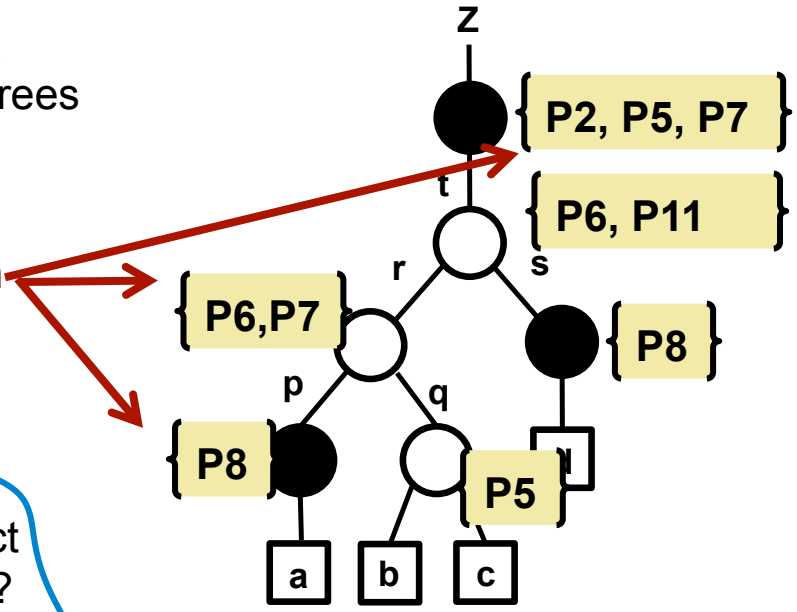


Be careful to match all **possible orientations** of these asymmetric patterns



Result After Matching

- **Where are we?**
  - For each “gate type” node of subject tree, we know which library pattern trees **match** that node.
  - We **annotate** each such node in the tree with this **matching information**



- **Next problem:**
  - What is the ***best cover*** of the subject tree with patterns from target library?

VLSI CAD: Logic to Layout

Lecture 10.5

From Logic to Layout: Technology Mapping—Minimum Cost Covering

Rob A. Rutenbar
University of Illinois

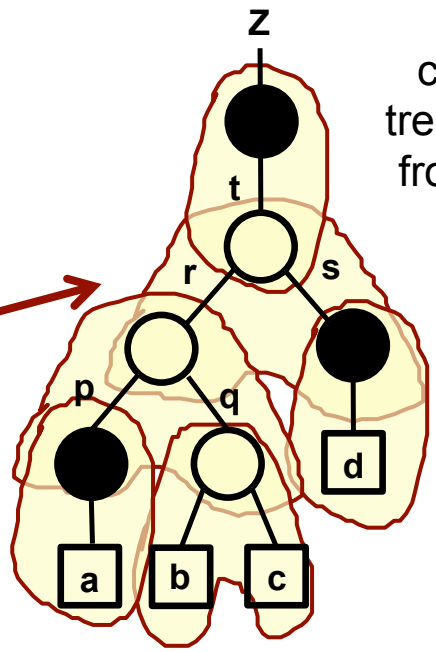
Tech Mapping via Tree Covering

- **What do we need for a complete algorithm?**

- **Tree-ifying the input netlist**
  - One key assumption needs discussion
- **Tree matching**
  - For each node in your subject tree, find the target pattern trees in library match here
- **Minimum-cost covering**
  - Assume you know what can match at each node of subject tree
  - ..so, which ones do you pick for a minimum cost cover?

Minimum Cost Tree Covering

- **Where are we?**
  - We **tree-ified** subject tree
  - For each node of subject tree, we know which library pattern trees **match** it
- **Next problem:**
  - What is the ***best cover*** of the subject tree with patterns from target library?



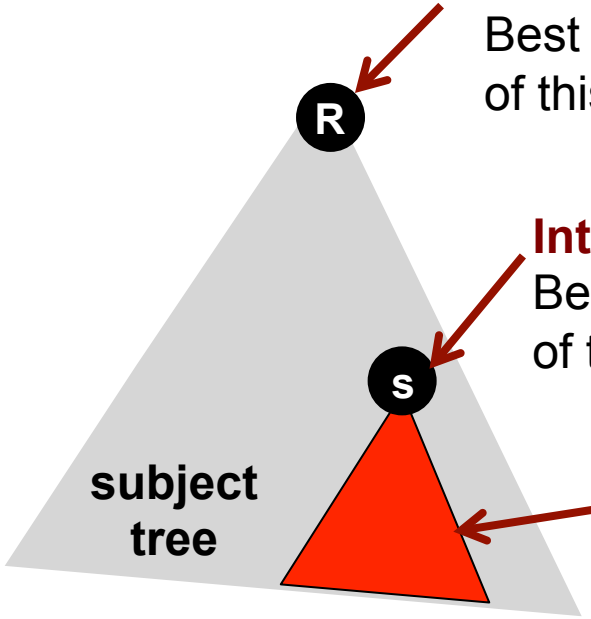
One possible cover of subject tree with 6 patterns from target library

Minimum Cost Covering of Subject Tree

- **What cover do we choose?**
  - We assign a **cost** to each library pattern
  - We choose the (or, a) **minimum cost** ("**mincost**") cover of the subject tree

- **One big idea makes this easy (!) to do:**
  - If pattern **P** is a mincost match at some node **s** of subject tree, then...
  - … each leaf of pattern tree must also be the root of some mincost matching pattern
  - Leads to a nice recursive algorithm for **mincost(node in subject tree)**
    - (This is actually a ***dynamic programming*** algorithm, if you know that term...)

Some Terminology First

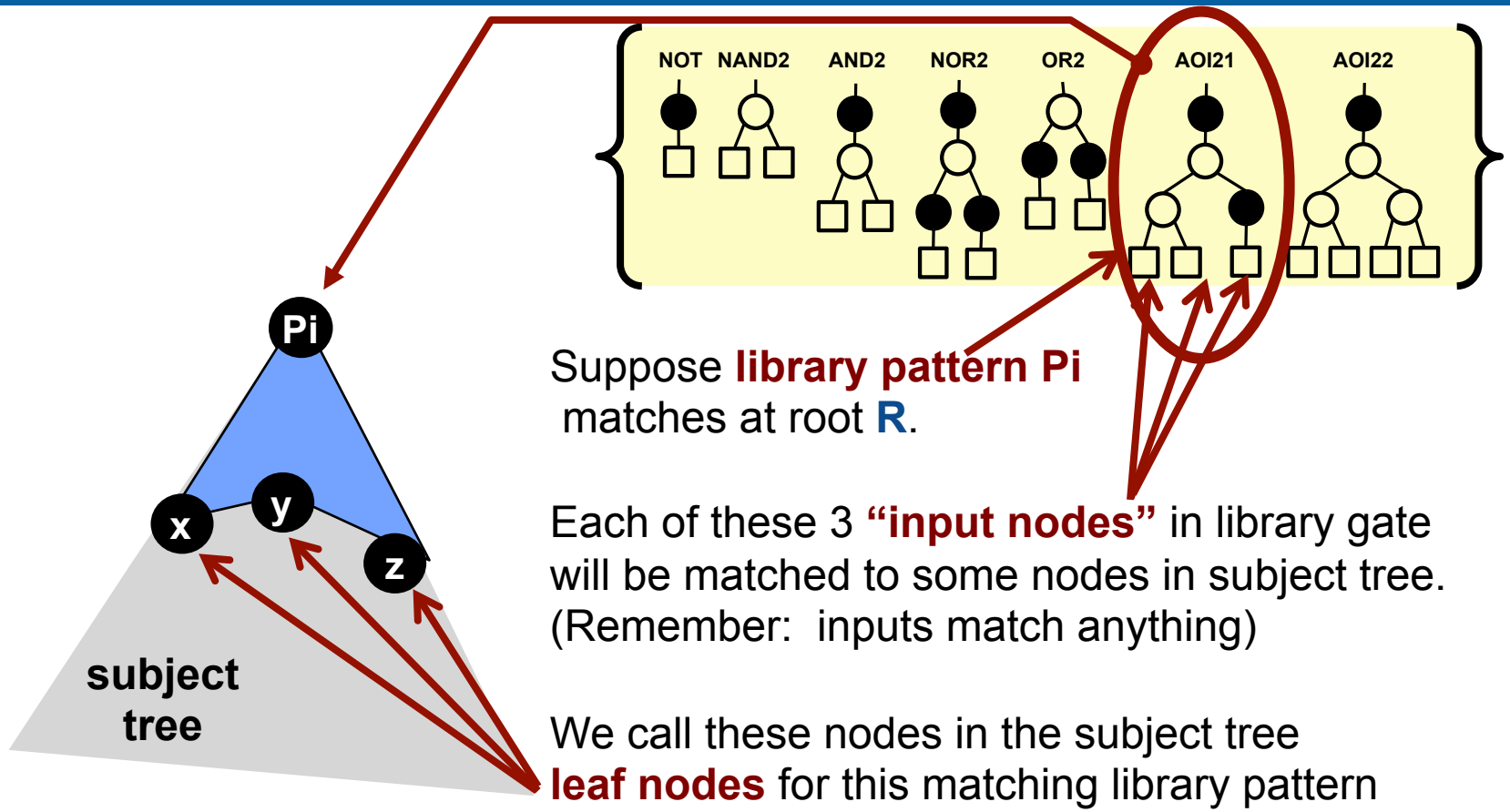


Root R of subject tree.
Best cover – mapping –
of this tree has cost=**mincost(R)**

Internal node s in subject tree.
Best cover – mapping –
of this **sub**-tree has cost=**mincost(s)**

Remember: we **tree-ified** the netlist.
This is a **tree**. Indeed, every internal node is the **root** of another, **smaller tree.**
This is crucial for our mapping algorithm.

More Terminology …

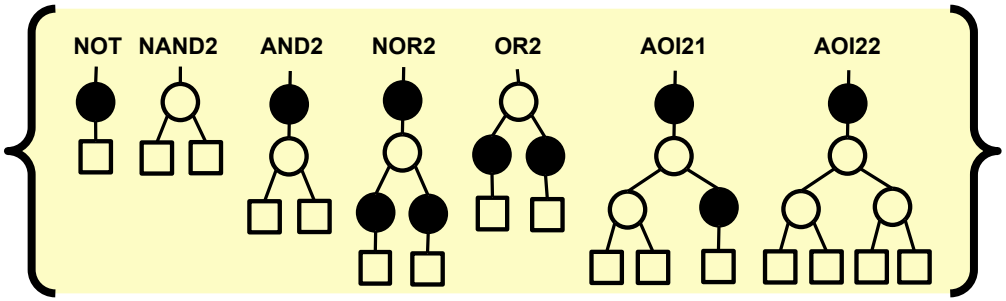


Suppose **library pattern Pi** matches at root **R**.

Each of these 3 **"input nodes"** in library gate will be matched to some nodes in subject tree. (Remember: inputs match anything)

We call these nodes in the subject tree **leaf nodes** for this matching library pattern

Final Terminology …

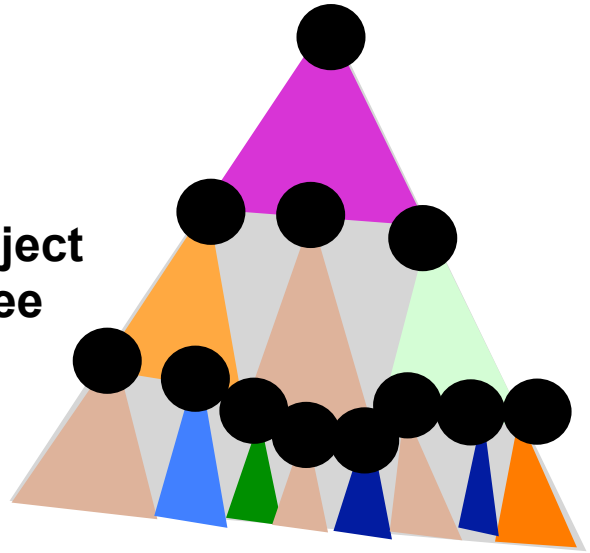


Every gate pattern in target library has a **cost**

Gate pattern **Pi** has **cost(Pi)**

We use library costs to calculate cost to map our complete subject netlist

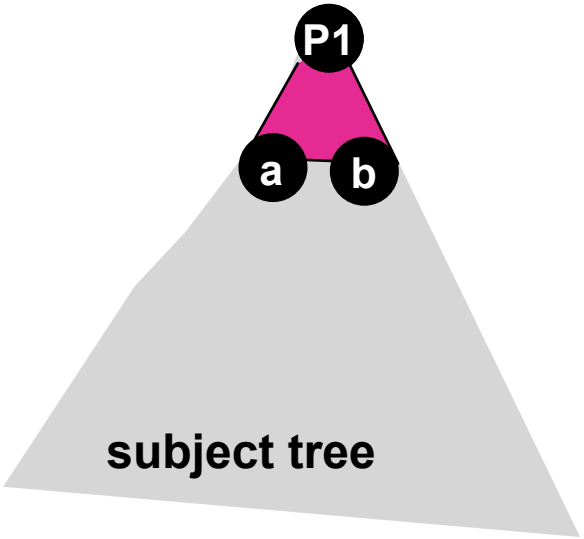
We add up **cost(node)** for each node where a pattern matches, for all patterns covering subject

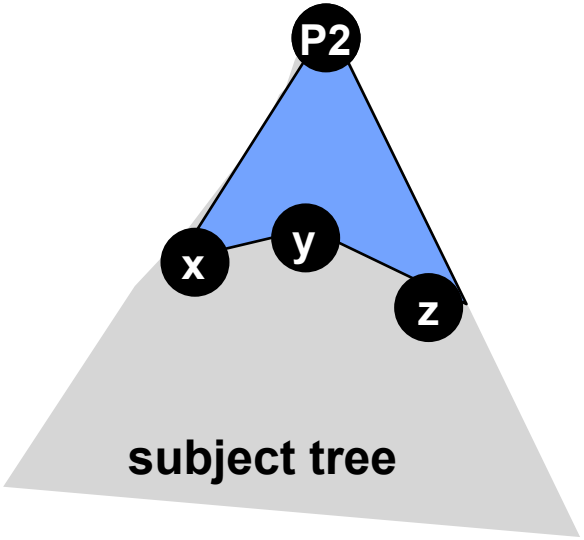


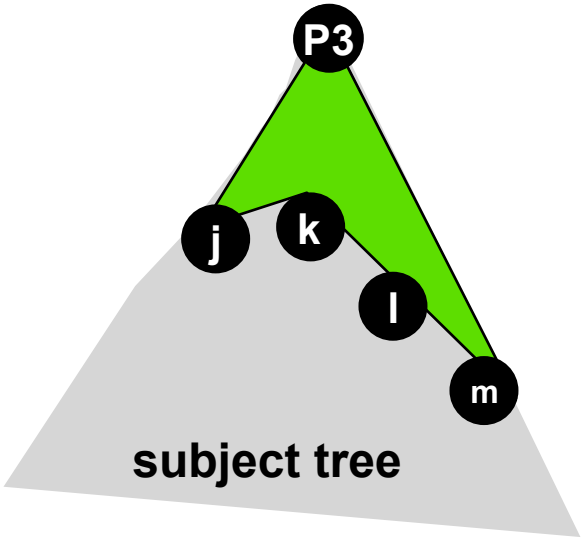
Mincost Tree Covering: The Idea

- **Assume 3 different library patterns match at root R of subject tree**
  - Pattern **P1** has **2** leaf nodes: **a b**
  - Pattern **P2** has **3** leaf nodes: **x y z**
  - Pattern **P3** has **4** leaf nodes: **j k l m**

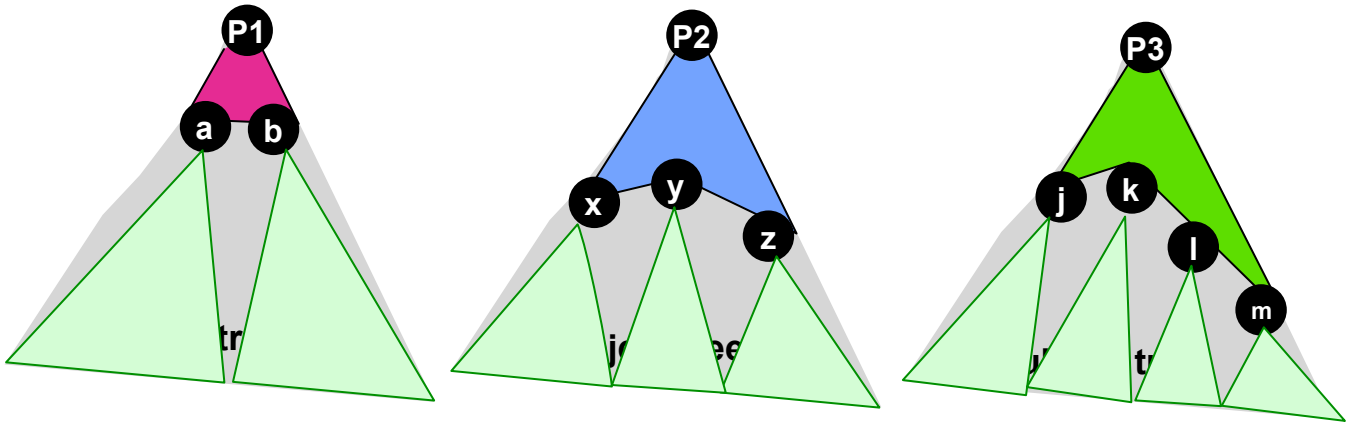
Which of these gates produces the smallest value of **mincost(R)**?







Mincost Tree Cover: Recursive Formula



- **Cheapest cover of root of subject has mincost(root) =**

Minimum over patterns that match at root R

{
cost(P1) + mincost(a) + mincost(b) ,

cost(P2) + mincost(x) + mincost(y) + mincost(z) ,

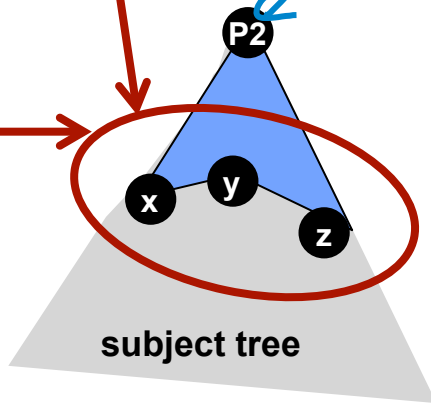
cost(P3) + mincost(j) + mincost(k) + mincost(l) + mincost(m)
}

Mincost Cover: Basic Recursive Algorithm

```
mincost( treenode) {
   cost = ∞
   foreach( pattern P matching at subject treenode) {

      let L = {nodes in subject tree corresponding to leaf nodes in P
                when P is placed with its root at treenode }

      newcost = cost(P)
      foreach( node n in L } {
         newcost = newcost +  mincost( n );
      }
      if ( newcost < cost )
      then  {
         cost = newcost;
         treenode.BestLibPattern = P;
      }
   }
```



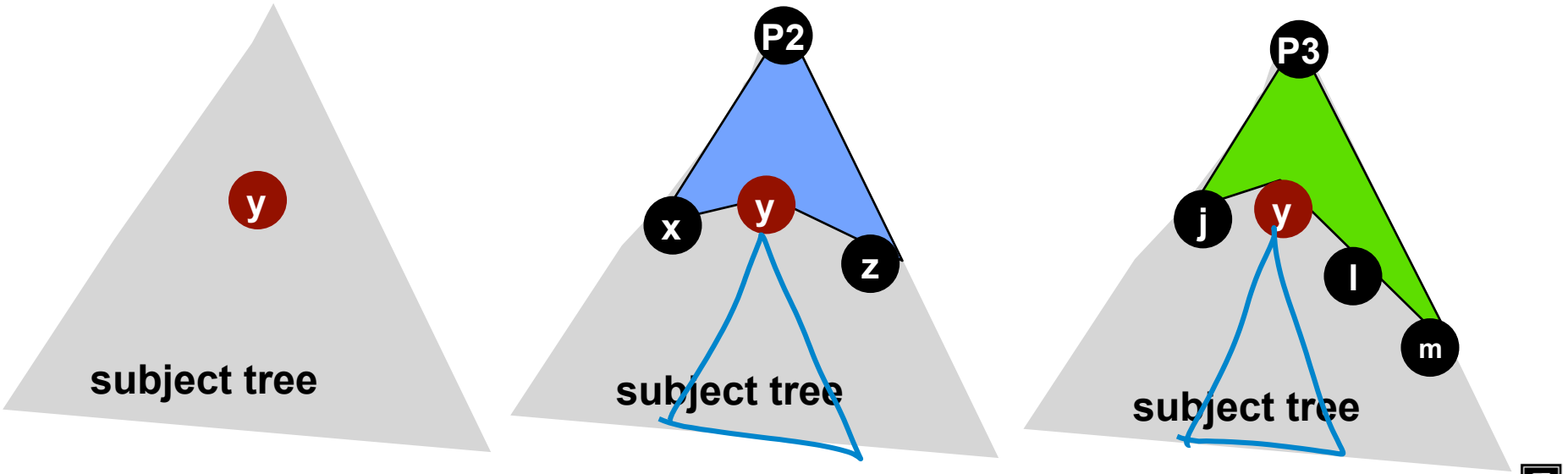
subject tree nodes "under" **x, y, z**

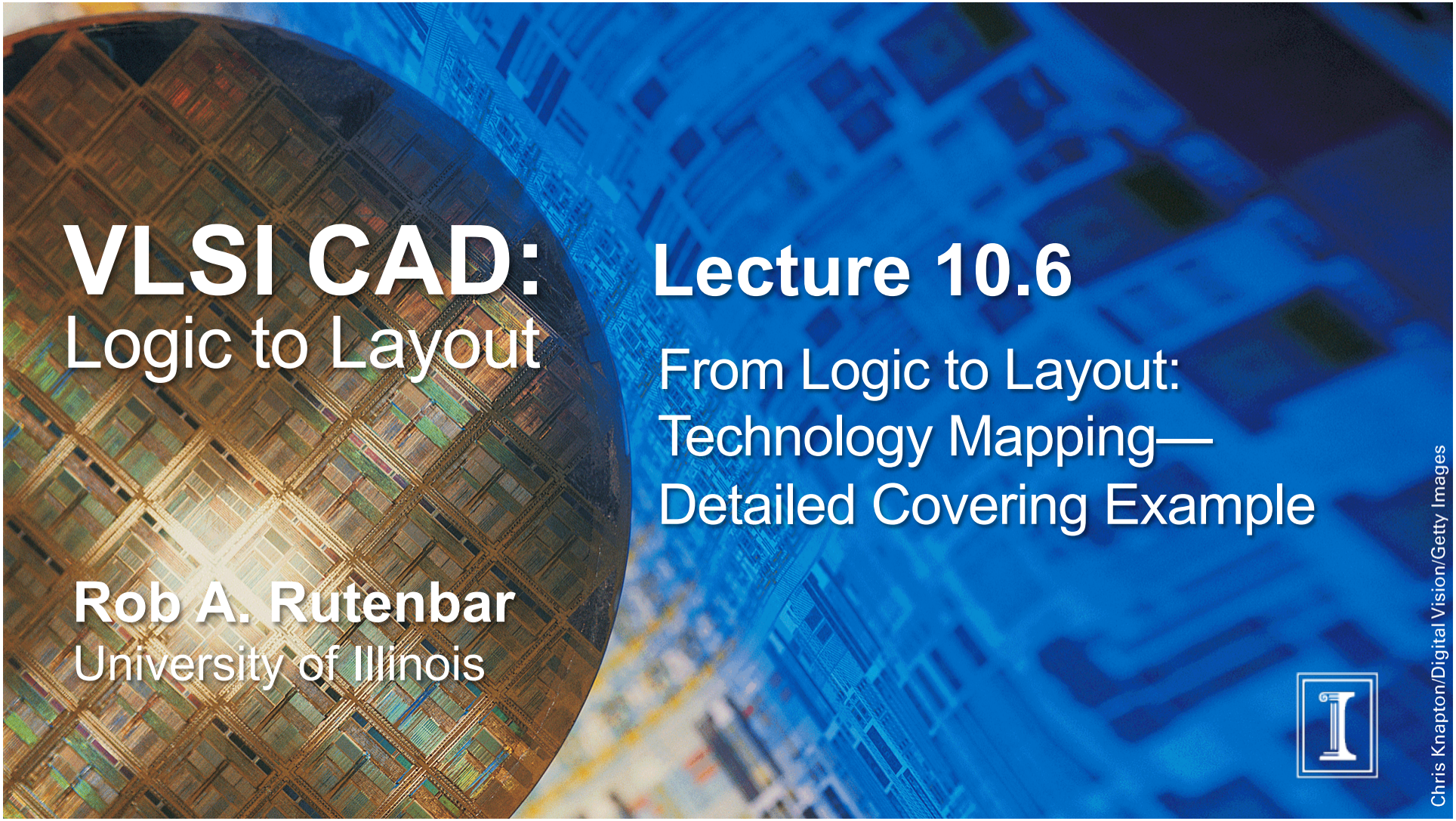
Min Cost Tree Cover

- **One useful optimization we must note:**
  - This algorithm will **revisit** same treenode many times during recursions...
  - ...and it will **recompute** the mincost cover for that node each time.
- **Can we do better...?**
  - Yes, just keep a table with min cost value for each node
  - Starts with value ∞ and then when node's cost get computed, this value gets updated
  - Check first to see if node has been visited before computing it--saves computing again

Illustration

- **Node "y" in this subject tree**
  - Will get its **mincost(y)** cover computed when we put **P2** at root of subject tree...
  - ...and again when we put **P3** at the root
  - **Better solution:** just compute it **once**, first time, **save** it, and **look it up** later!





VLSI CAD: Logic to Layout

Lecture 10.6

From Logic to Layout: Technology Mapping—Detailed Covering Example

Rob A. Rutenbar
University of Illinois

Tech Mapping via Tree Covering

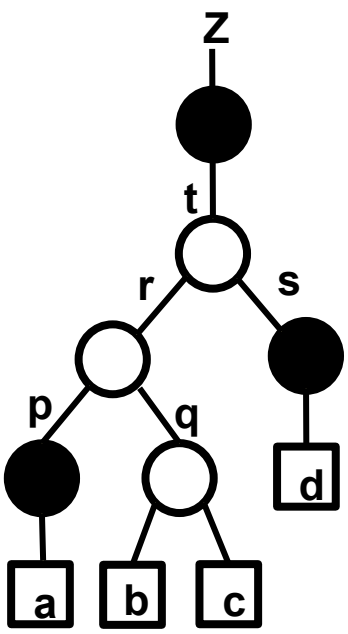
- **What do we need for a complete algorithm?**

- **Tree-ifying the input netlist**
  - One key assumption needs discussion
- **Tree matching**
  - For each node in your subject tree, find the target pattern trees in library match here
- **Minimum-cost covering**
  - Assume you know what can match at each node of subject tree
  - ..so, which ones do you pick for a minimum cost cover?

EXAMPLE

Min Cost Tree Cover Example

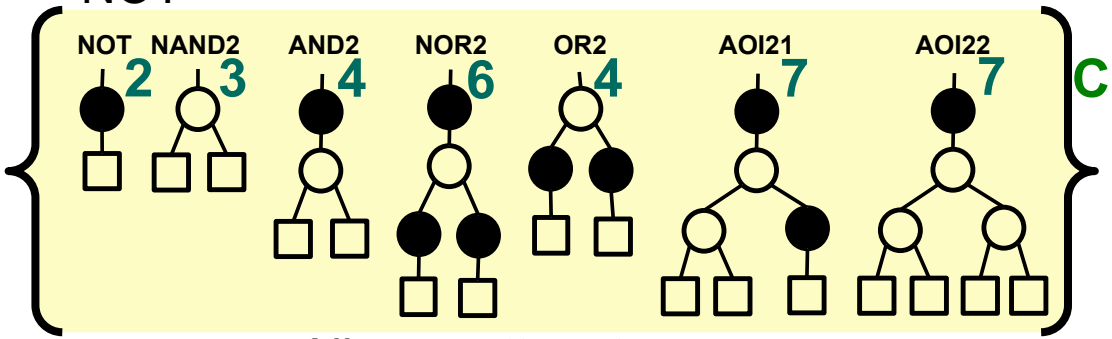
Subject tree



At node	Can Match	With min cost
Z	NOT	**2** + mincost(**t**)
	AND2	**4** + mincost(**r**) + mincost(**s**)
	AOI21	**7** + mincost(**p**) + mincost(**q**)
t	NAND2	**3** + mincost(**r**) + mincost(**s**)
r	NAND2	**3** + mincost(**p**) + mincost(**q**)
p	NOT	**2**
q	NAND2	**3**
s	NOT	**2**

Minimum

Bug Fix: AND2 matches at Z

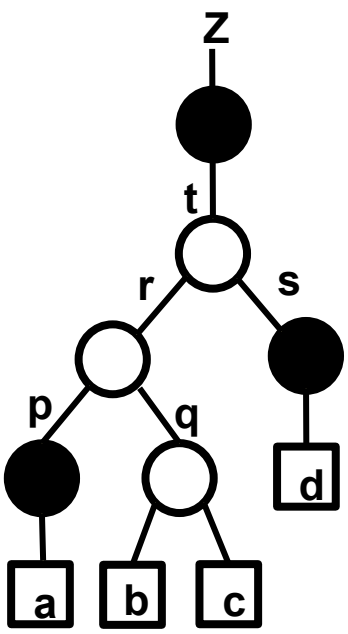


COSTS

All our pattern trees

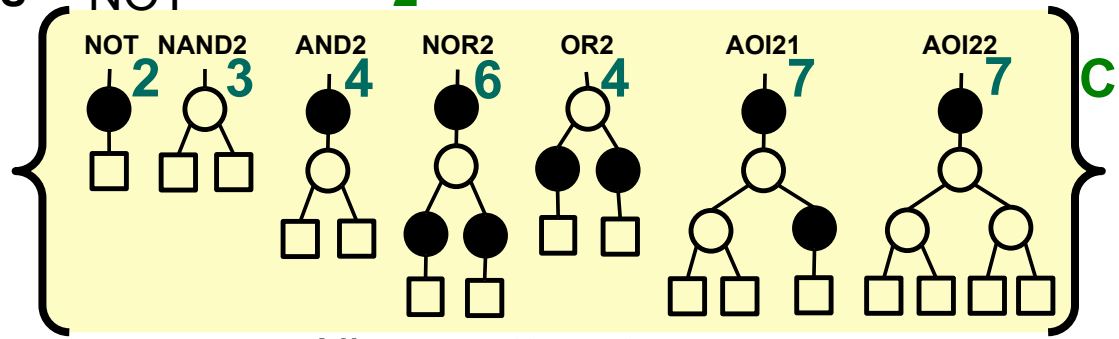
Min Cost Tree Cover Example

Subject tree



At node	Can Match	With min cost
z	NOT	2 + mincost(**t**)
	AND2	4 + mincost(**r**) + mincost(**s**)
	AOI21	7 + mincost(**p**) + mincost(**q**)
t	NAND2	3 + mincost(**r**) + mincost(**s**)
r	NAND2	3 + mincost(**p**) + mincost(**q**)
p	NOT	2
q	NAND2	3
s	NOT	2

Minimum

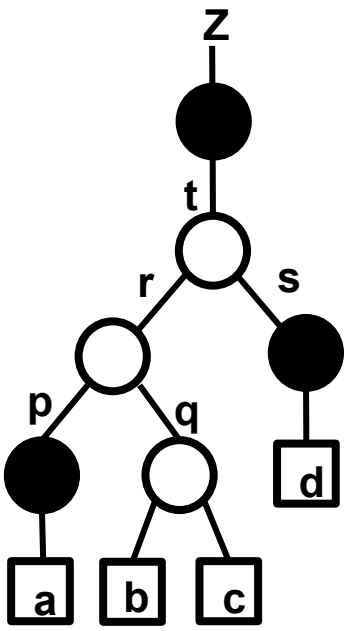


COSTS

All our pattern trees

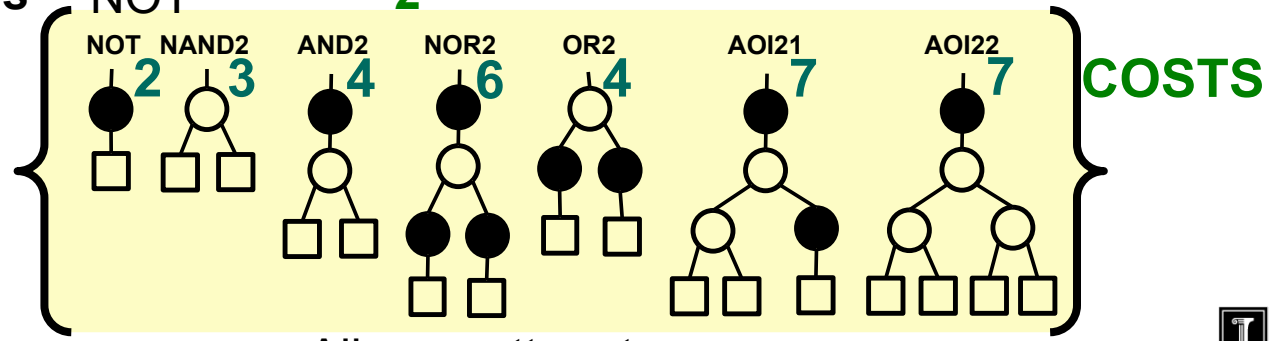
Min Cost Tree Cover Example

Subject tree



At node	Can Match	With min cost
z	NOT	2 + mincost(**t**)
	AND2	4 + 2 + mincost(r) = 8
	AOI21	7 + 2 + 3 = **12**
t	NAND2	3 + mincost(**r**) + 2 = 8
r	NAND2	3 + 2 + 3 = **8**
p	NOT	2
q	NAND2	3
s	NOT	2

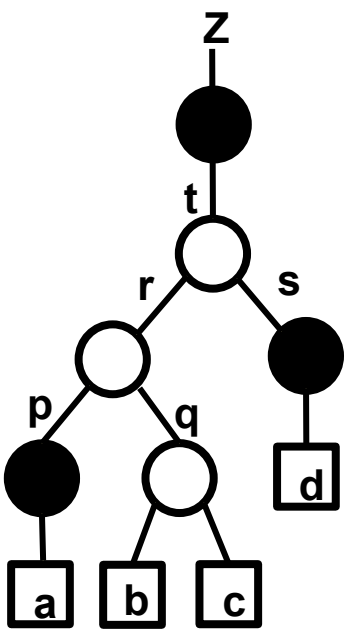
Minimum



All our pattern trees

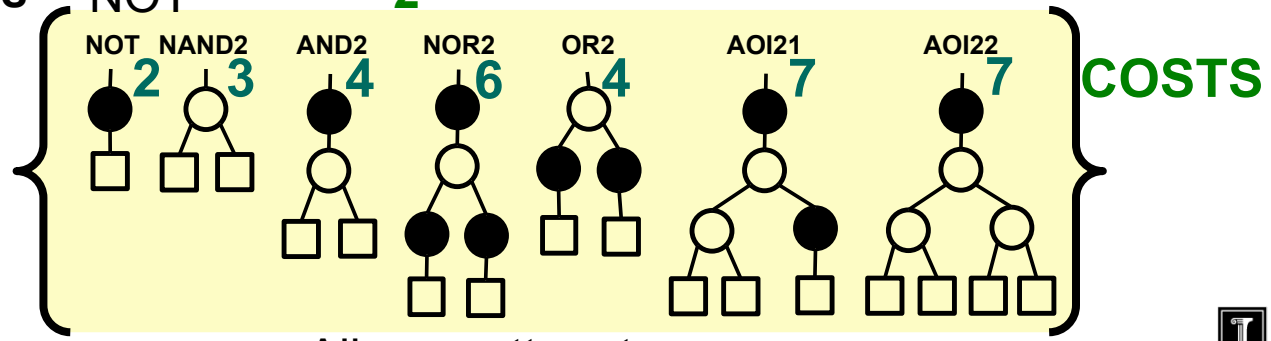
Min Cost Tree Cover Example

Subject tree



At node	Can Match	With min cost
Z	NOT	2 + mincost(t) 13
	AND2	4 + 2 + 8 = 14
	AOI21	7 + 2 + 3 = 12
t	NAND2	3 + 8 + 2 = 13
r	NAND2	3 + 2 + 3 = 8
p	NOT	2
q	NAND2	3
s	NOT	2

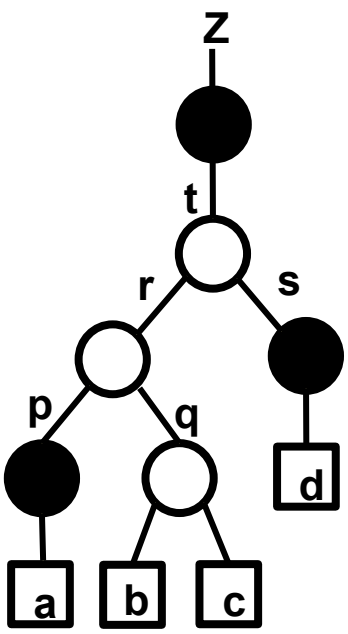
Minimum



All our pattern trees

Min Cost Tree Cover Example

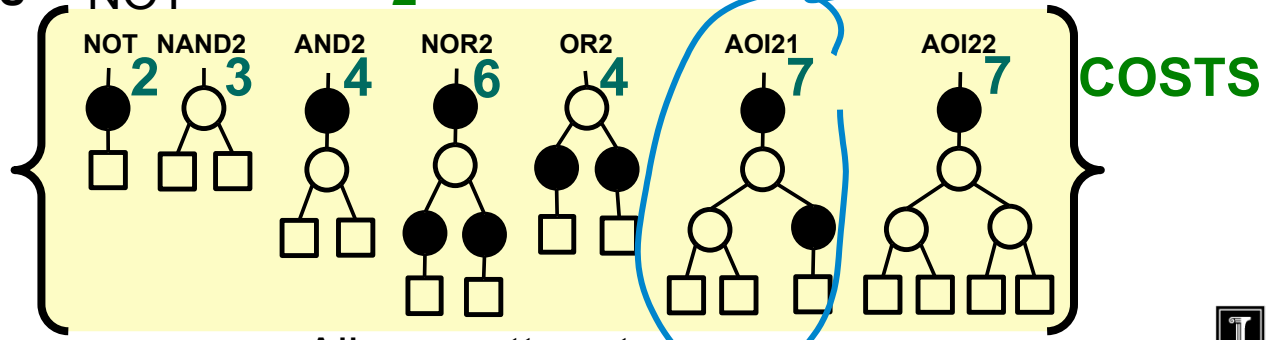
Subject tree



At node	Can Match	With min cost
Z	NOT	2 + 13 = 15
	AND2	4 + 2 + 8 = 14
	AOI21	7 + 2 + 3 = 12
t	NAND2	3 + 8 + 2 = 13
r	NAND2	3 + 2 + 3 = 8
p	NOT	2
q	NAND2	3
s	NOT	2

Minimum

Best! AOI21 @ Z



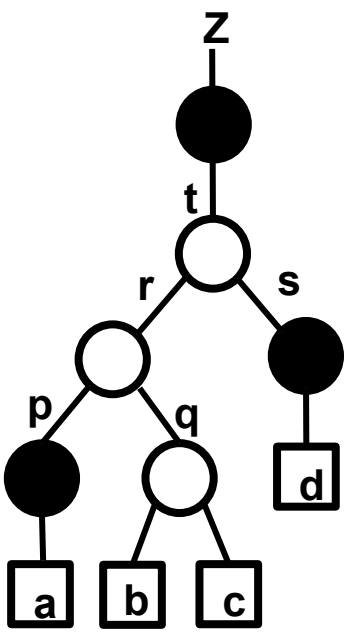
All our pattern trees

Min Cost Cover: How To Get Final Cover?

- **Look at best cost at subject root. Find pattern P with that cost.**
  - In our example: this is **cost=12** associated with AOI21 at node **Z**
- **Find the leaf nodes in the subject tree with P at the root**
  - In our example: this is nodes **p** and **q**
  - Look at the **best cost** at each of these leaf nodes
  - Find the pattern **Pi** that is associated with each of these best costs at leaf nodes
  - Choose best cost pattern **Pi** for each leaf node, that look again at **leaf nodes** inside each of these patterns **Pi**, placed inside the subject tree
  - **Repeat**…

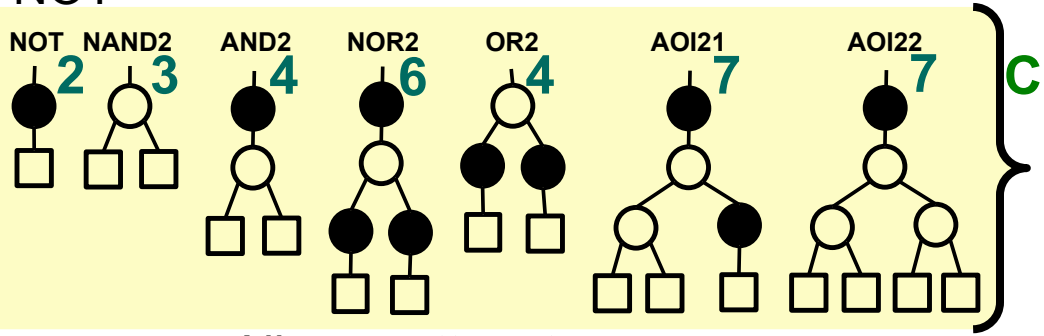
Min Cost Tree Cover Example

Subject tree



	At node	Can Match	With min cost
	Z	NOT	**15** = 2 + mincost(**t**)
		AND2	**14** = 4 + mincost(**r**) + mincost(**s**)
Best!		AOI21	**12** = 7 + mincost(**p**) + mincost(**q**)
	t	NAND2	**13** = 3 + mincost(**r**) + mincost(**s**)
	r	NAND2	**8** = 3 + mincost(**p**) + mincost(**q**)
Choose	**p**	NOT	2
Choose	**q**	NAND2	3
	s	NOT	2

Min



COSTS

All our pattern trees

Min Cost Tree Cover

- **Example**
  - If costs for NOT, NAND2, AND2, AOI21 are as shown...
  - Best cover is to use one AOI21 (@ Z), one NAND2 (@ q), and one NOT (@ p)

- **Turns out to be several nice extensions possible**
  - Can modify algorithm a little to minimize **delay** instead of cost
  - Need to deal with some technical issues associated with capacitive loads for driven gates, but some simple discrete loads can be modeled and handled
  - Many interesting and useful variations, starting from this algorithm skeleton

Summary

- **Technology mapping ...**
  - Synthesis gives you “uncommitted” or “technology independent” design, eg, NAND2 and NOT
  - **Mapping turns this into real gates in your own library**
  - Can determine difference between good implementation and a bad one

- **Tree covering**
  - One nice, simple, elegant approach to the problem
  - 3 parts: tree-ify input, match all lib patterns, find min cost cover
  - Yes, there are other ways to do this. Some work with real Boolean algebra in mapping. Other applications, like for Lookup-Table (LUT) FPGA, are yet different