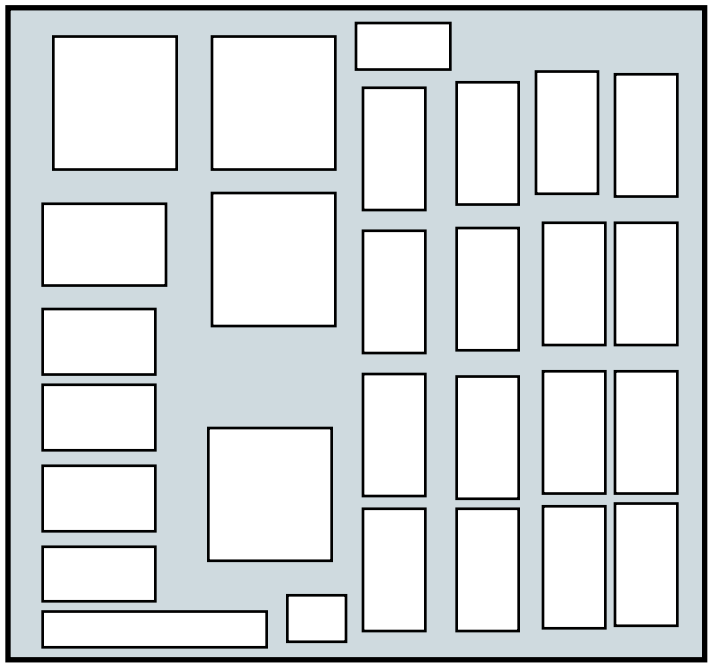
VLSI CAD: Logic to Layout

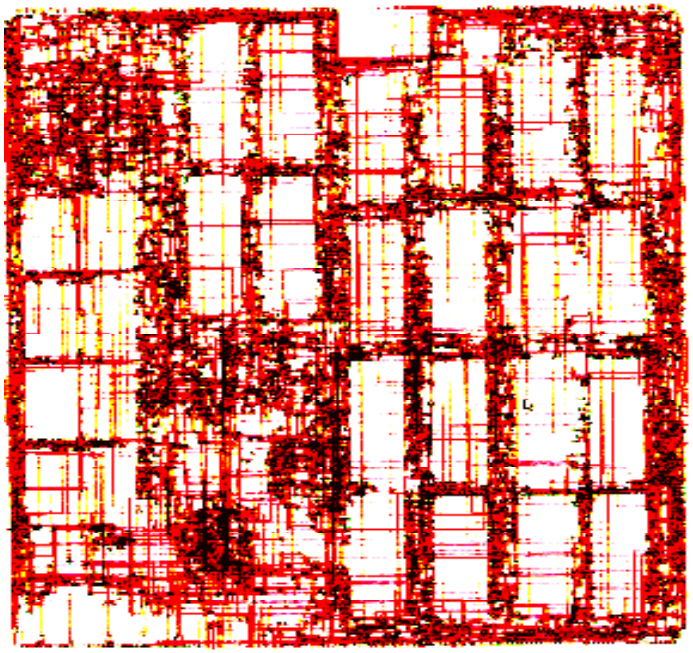
Lecture 11.1

ASIC Layout: Routing Basics

Rob A. Rutenbar
University of Illinois

Routing: The Problem



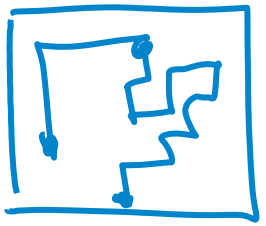


Thousands of macro blocks
Millions of gates
Millions of wires

Kilometers of wire.

Basic Routing Problems

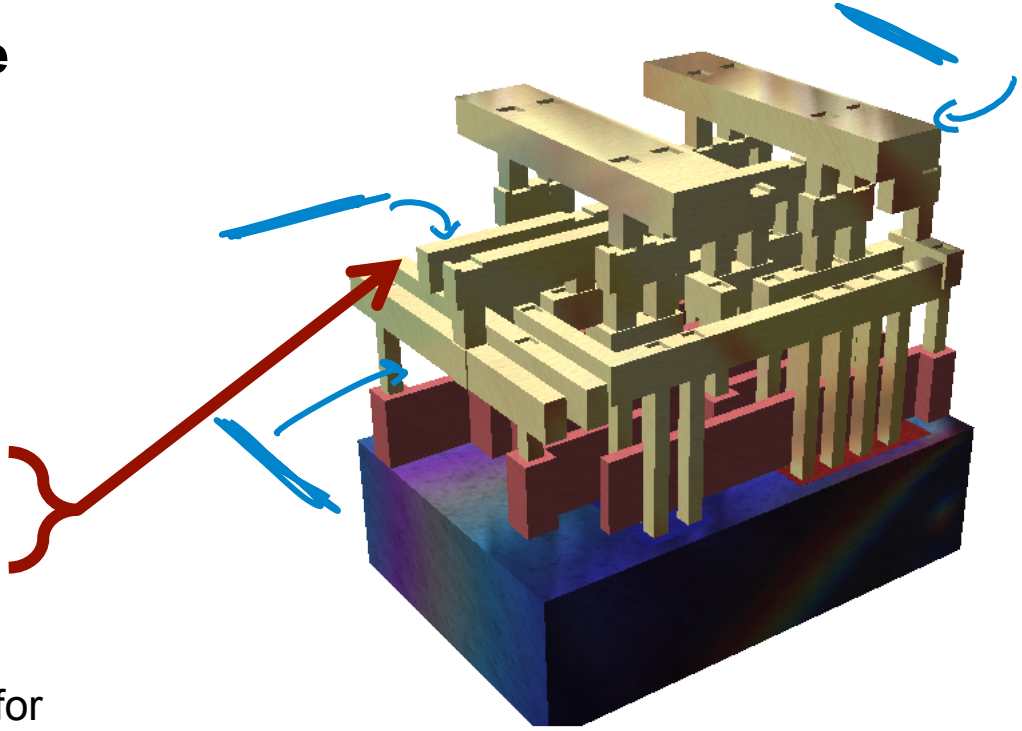
- **Scale**
  - Big chips have an enormous number (**millions**) of wires
  - Not every wire gets to take an "easy" path to connect its pins
  - **Must** connect them all--can't afford to embed many wires manually



- **Geometric complexity**
  - At nanoscale, **geometry rules are complex** – makes routing hard
  - Nevertheless, we use a simple **grid representation** of layout – right place to start
- **Electrical complexity**
  - It's not enough to make sure you connect all the wires
  - You also must ensure that the **delays** thru the wires are not too big
  - And that wire-to-wire **interactions** (crosstalk) don't mess up behavior

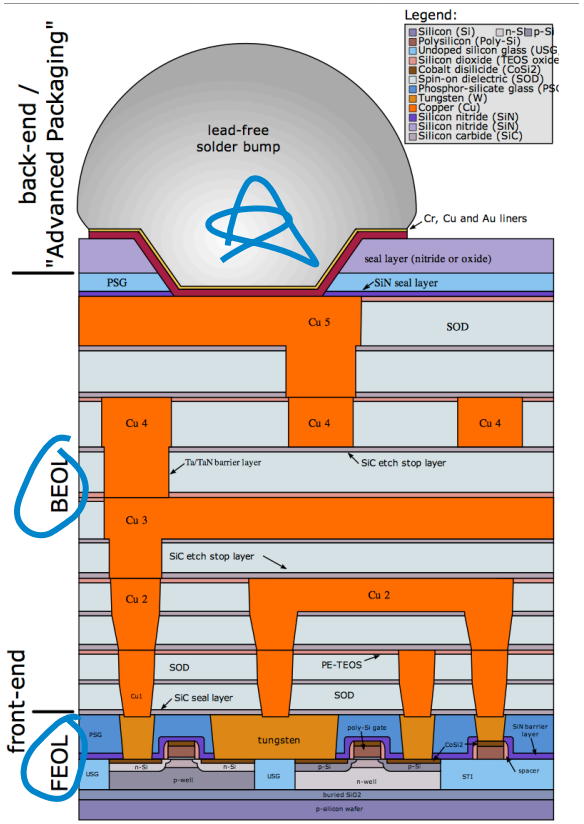
Physical Assumptions

- **Many layers of wiring are available for routing**
  - Made of metal (today, copper)
  - We can connect wires in different layers with **vias**
- **A simplified view**
  - Standard cells are using metal on layers 1,2
  - Routing wires on layers 3-8
  - Upper layers (9, 10) reserved for power and clock distribution



Wikipedia: search "standard cell"
http://en.wikipedia.org/wiki/File:Silicon_chip_3d.png

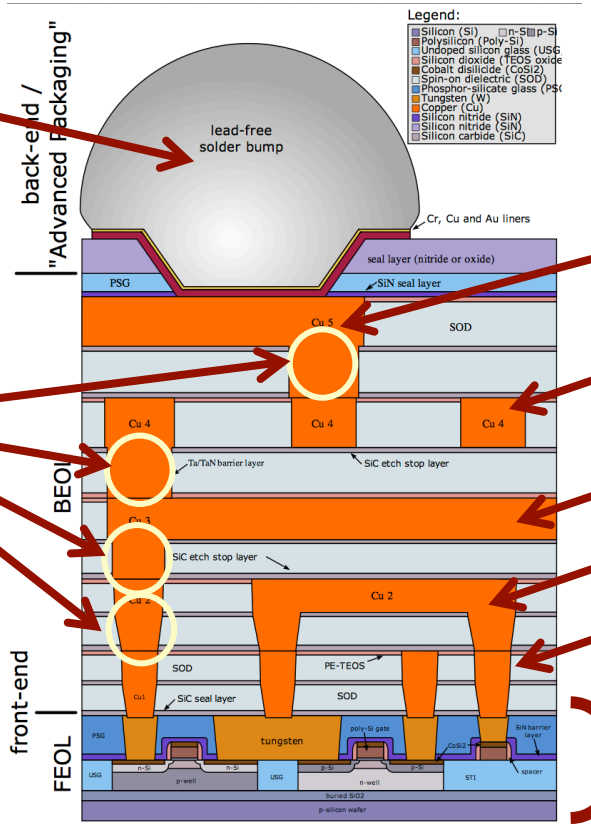
Typical Example (5 Layers, Circa 2000)



- **Some terminology**
  - **FEOL**: Front end of line. Chip fabrication steps that make transistors on Si wafer
  - **BEOL**: Back end of line. Fabrication steps that make layers of wires on top of transistors
- **This picture**
  - Cross sectional view of FEOL and BEOL structures in 5-layers of wiring…
  - … along with packaging connection (“bump”) which is how chip connects to pins on package

http://upload.wikimedia.org/wikipedia/commons/e/ee/Cmos-chip_structure_in_2000s_%28en%29.svg

Typical Example (5 Layers, Circa 2000)



Package interconnect, (like a via), but from chip to board

Vias, connect Metal layers k and k+1

Layer 5 metal

Layer 4 metal

Thicker layers (less resistance) for clock, power

Layer 3 metal

Layer 2 metal

Layer 1 metal

Wiring for logic

Transistors to make gate-level logic

http://upload.wikimedia.org/wikipedia/commons/e/ee/Cmos-chip_structure_in_2000s_%28en%29.svg

Placement vs Routing

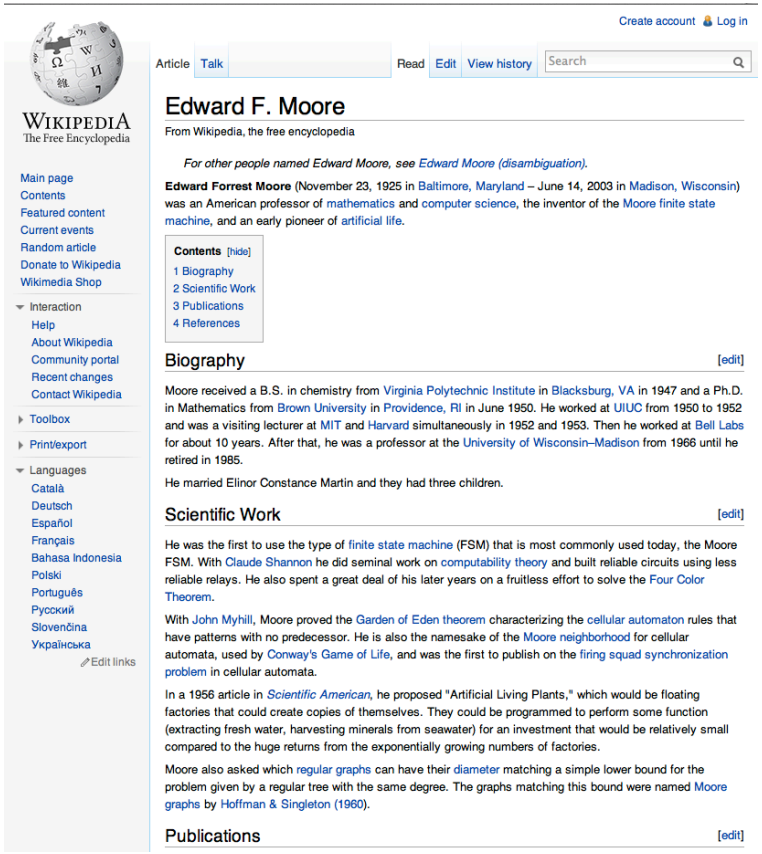
- **There are lots of different placement algorithms**
  - Iterative methods. Used mainly for high-level floorplanning, not gate layout
  - Many analytical methods based on solving/optimizing large systems of equations

- **There are not quite so many routing algorithms**
  - There are lots of routing data structures – to represent the geometry efficiently
  - But there is **one very, very big idea** at core of most real routers…

Big Idea Called "Maze Routing"

- **From one famous early paper:**
  - E.F. Moore. "**The shortest path through a maze**"
  - International Symposium on the Theory of Switching Proceedings, pp 285--292, Cambridge, MA, Apr. **1959**. Harvard University Press

- **Yes – it's that Moore of "Moore state machines" fame**
  - Given a maze (or a graph), find a shortest path from entrance to exit



Create account Log in

Article Talk | Read Edit View history | Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Languages
Català
Deutsch
Español
Français
Bahasa Indonesia
Polski
Português
Русский
Slovenčina
Українська
Edit links

Edward F. Moore

From Wikipedia, the free encyclopedia

For other people named Edward Moore, see Edward Moore (disambiguation).

Edward Forrest Moore (November 23, 1925 in Baltimore, Maryland – June 14, 2003 in Madison, Wisconsin) was an American professor of mathematics and computer science, the inventor of the Moore finite state machine, and an early pioneer of artificial life.

Contents [hide]
1 Biography
2 Scientific Work
3 Publications
4 References

Biography [edit]

Moore received a B.S. in chemistry from Virginia Polytechnic Institute in Blacksburg, VA in 1947 and a Ph.D. in Mathematics from Brown University in Providence, RI in June 1950. He worked at UIUC from 1950 to 1952 and was a visiting lecturer at MIT and Harvard simultaneously in 1952 and 1953. Then he worked at Bell Labs for about 10 years. After that, he was a professor at the University of Wisconsin–Madison from 1966 until he retired in 1985.

He married Elinor Constance Martin and they had three children.

Scientific Work [edit]

He was the first to use the type of finite state machine (FSM) that is most commonly used today, the Moore FSM. With Claude Shannon he did seminal work on computability theory and built reliable circuits using less reliable relays. He also spent a great deal of his later years on a fruitless effort to solve the Four Color Theorem.

With John Myhill, Moore proved the Garden of Eden theorem characterizing the cellular automaton rules that have patterns with no predecessor. He is also the namesake of the Moore neighborhood for cellular automata, used by Conway's Game of Life, and was the first to publish on the firing squad synchronization problem in cellular automata.

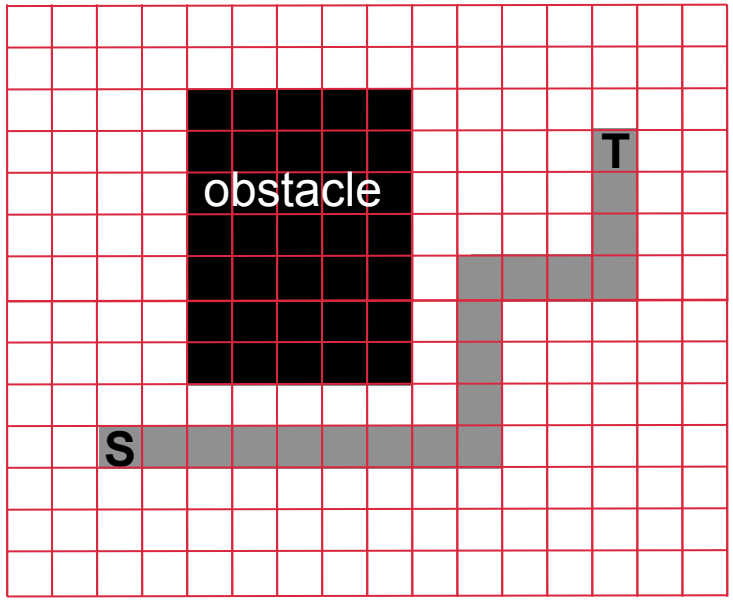
In a 1956 article in *Scientific American*, he proposed "Artificial Living Plants," which would be floating factories that could create copies of themselves. They could be programmed to perform some function (extracting fresh water, harvesting minerals from seawater) for an investment that would be relatively small compared to the huge returns from the exponentially growing numbers of factories.

Moore also asked which regular graphs can have their diameter matching a simple lower bound for the problem given by a regular tree with the same degree. The graphs matching this bound were named Moore graphs by Hoffman & Singleton (1960).

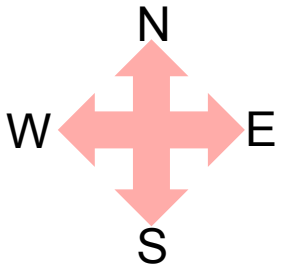
Publications [edit]

How Do We Get from "Mazes" to "Wires"?

- **Make a big geometric assumption: Gridded routing**
  - The layout surface is a **grid of regular squares**
  - A legal wire path = a **set of connected grid cells**, through unobstructed cells in grid
  - We can mark **obstacles** (also called **blockages**) where we are not allowed to route



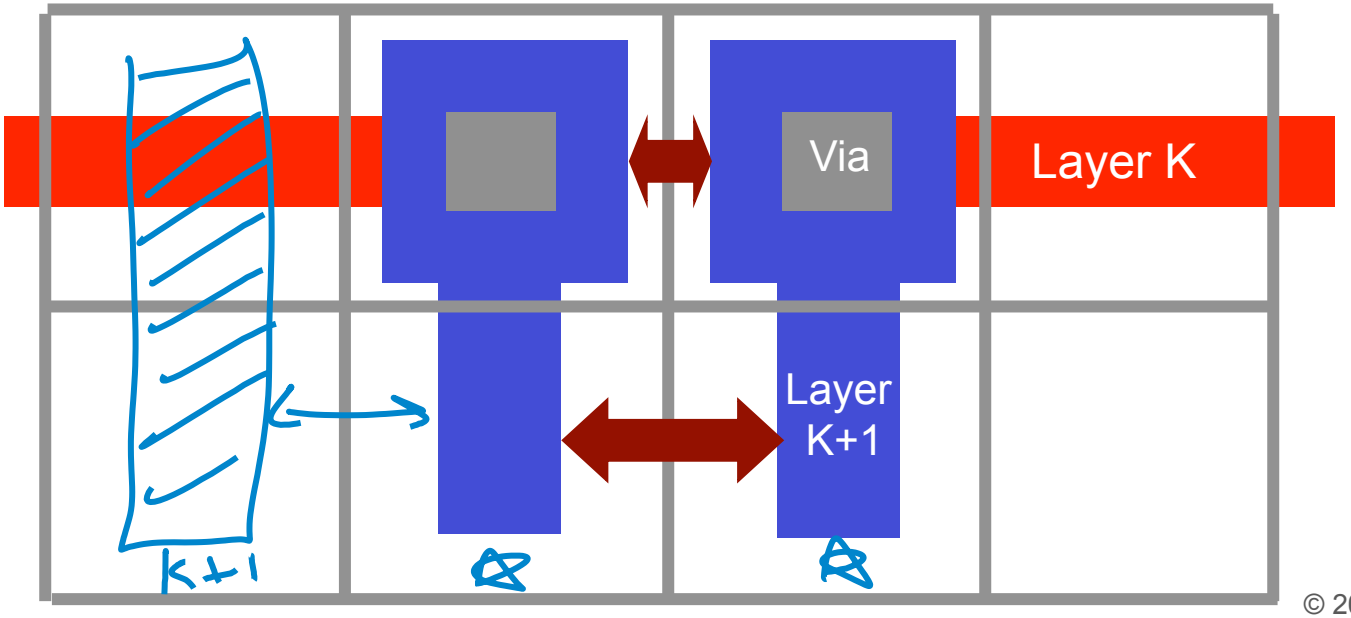
For us: wires are also **strictly horizontal and vertical.** No diagonal (eg, 45°) angles. A path goes east/west, or north/south, in this grid.



Compass

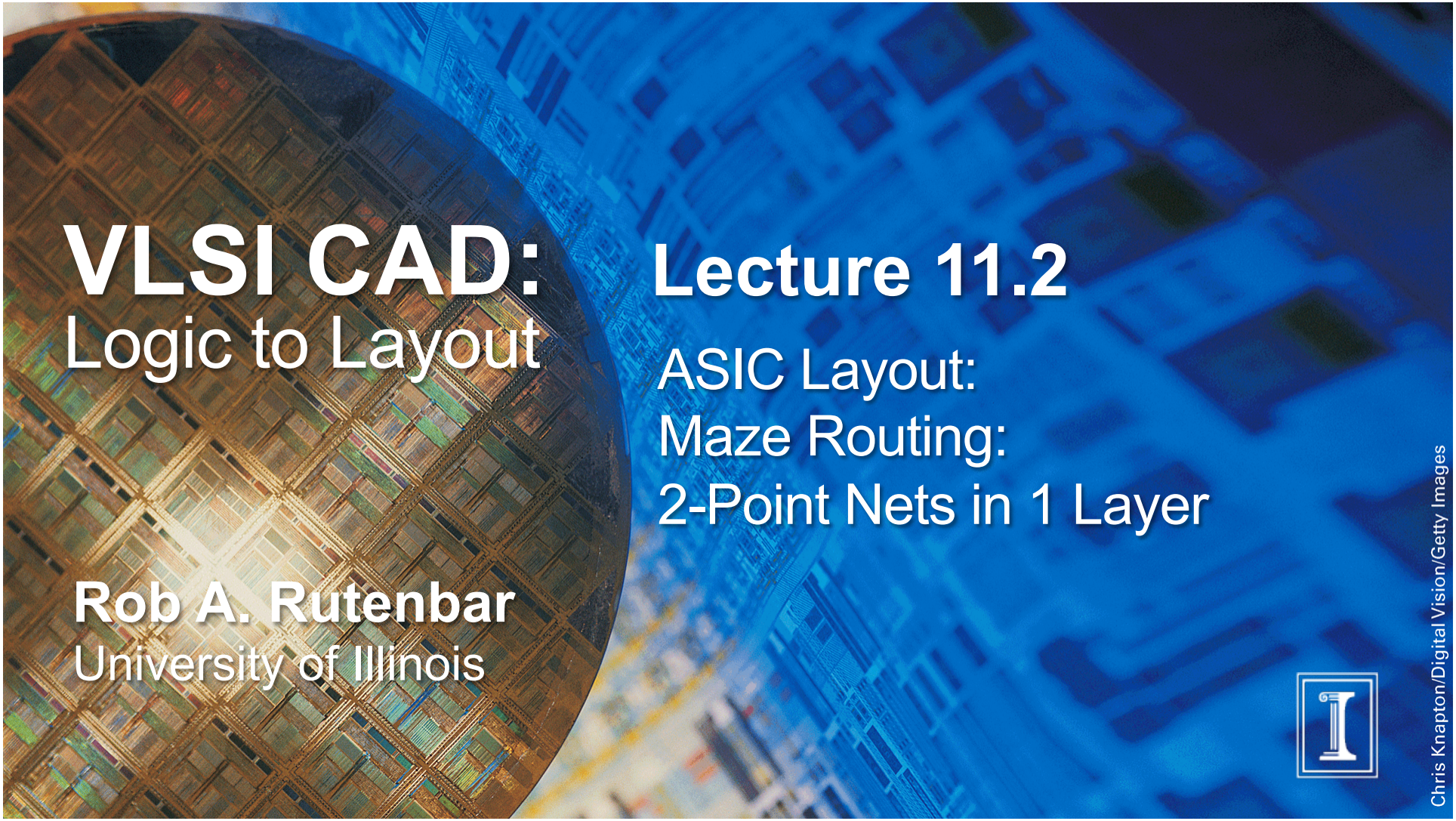
Grid Assumption

- **This is a critical assumption – implies many constraints on wires**
  - All wires are roughly the same size (width)
  - Wires and their vias fit in the grid, without any geometry rule (eg, spacing) violations
  - All pins we want to connect are also "on grid" ie, center of grid cell



Routing: Maze Routers

- **Our Topics:**
  - **Router functionality**
    - Two-point nets in one layer - unit cost
    - Multi-point nets
    - Multiple layers
    - Non-uniform grid costs
  - **Implementation mechanics**
    - Expansion
    - Data structures & constraints
    - Depth-first search
  - **Divide & Conquer: Global routing**



VLSI CAD: Logic to Layout

Lecture 11.2

ASIC Layout:
Maze Routing:
2-Point Nets in 1 Layer

Rob A. Rutenbar
University of Illinois

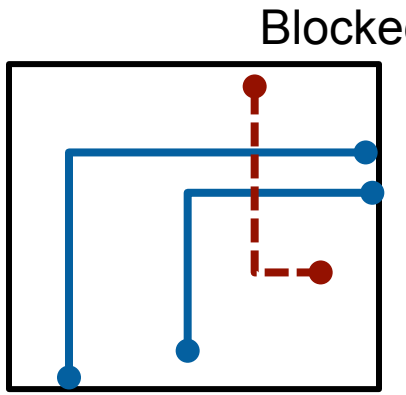
Maze Routing: History After Moore's Mazes

- **1959 – Basic Idea**
  - E.F. Moore. The shortest path through a maze. In International Symposium on the Theory of Switching Proceedings, Apr. 1959. Harvard University Press
- **1961 – Applied to electronics (board wiring)**
  - Lee, C. Y., "An algorithm for path connections and its applications", IRE Trans. on Electronic Computers, pp. 346-365, Sept. 1961
  - Chester Lee of Bell Labs invents the algorithm; gets famous for "Lee routers"
- **1974**
  - Rubin, F., "The Lee path connection algorithm", IEEE Trans. on Computers, vol. c-23, no. 9, pp. 907-914, Sept. 1974.
  - Frank Rubin applies some ideas from recent AI results to make it go much faster
- **1983**
  - Hightower, D., "The Lee router revisited", ICCAD, pp. 136-139, 1983.
  - Dave Hightower uses fact that "modern" computers have big virtual memory, builds great router

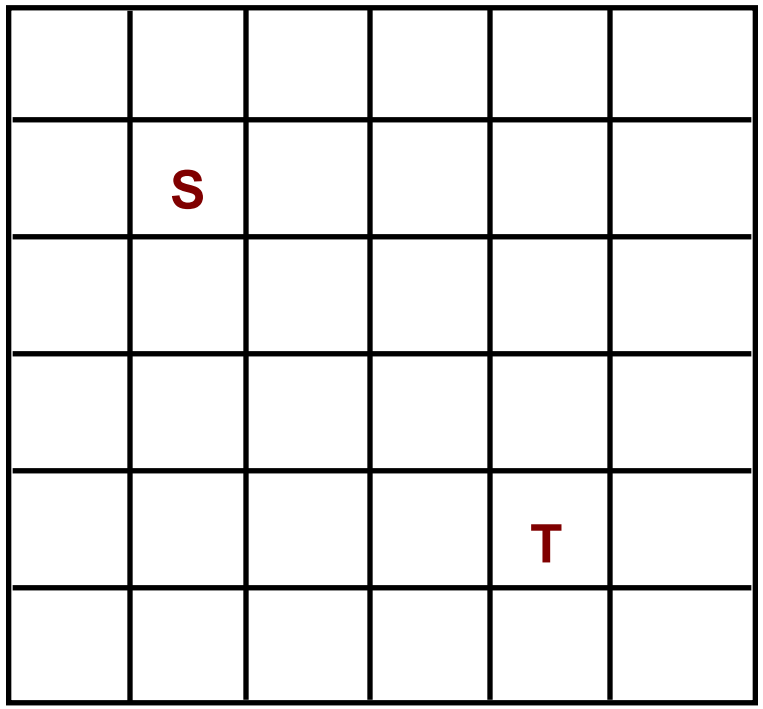
Maze Routing: Strategy

- **Strategy**
  - **One net at a time**: completely wire **one net**, then move onto next net
  - **Optimize net path:** find the **best** wiring path

- **Problems**
  - Early nets wired may **block** path of later nets
  - Optimal choice for one net may block later nets
  - We are just going to ignore this one for the moment…

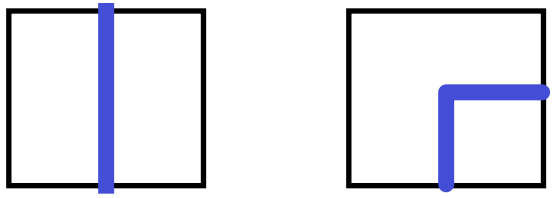


Maze Router: Basic Idea for 2-Pin Nets



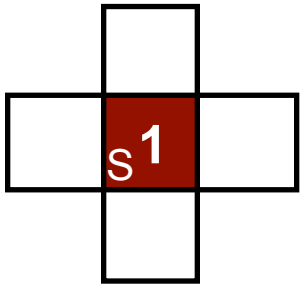
- **Given:**
  - **Grid** - each square cell represents where **one** wire can cross
  - A ***source*** and ***target***
- **Problem:**
  - Find shortest path connecting **source** cell (**S**) and **target** cell (**T**)
  - When using cells, a wire can:

cross **or** **bend**

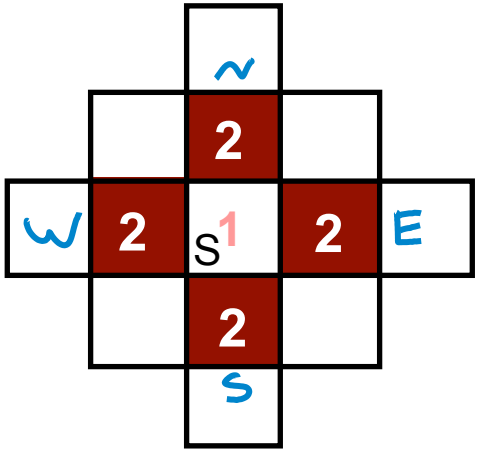


Maze Routing: *Expansion*

Start at the **source**



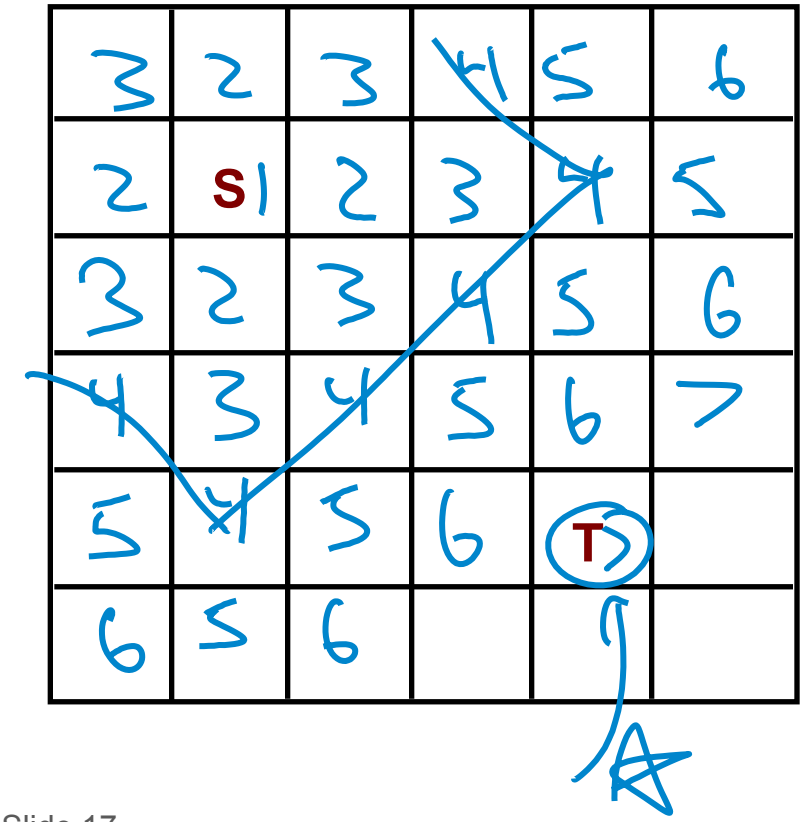
Find all new cells that are reachable at **pathlength 1**, ie, all paths that are just 1 unit in total length (just 1 cell) - mark all with this the pathlength



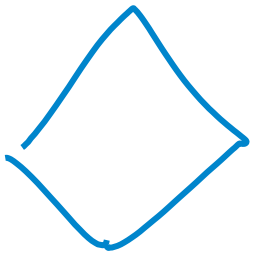
Using the **pathlength 1** cells, find all new cells which are reachable at **pathlength 2**

Repeat until the target is reached.

Maze Router Step 1: Expand



- **Strategy**
  - Expand **one cell at a time** until all the shortest paths from **S** to **T** are found.
  - Expansion creates a **wavefront** of paths that search broadly out from source cell until target is reached
  - "Reached" means specifically we **label** it with a pathlength number

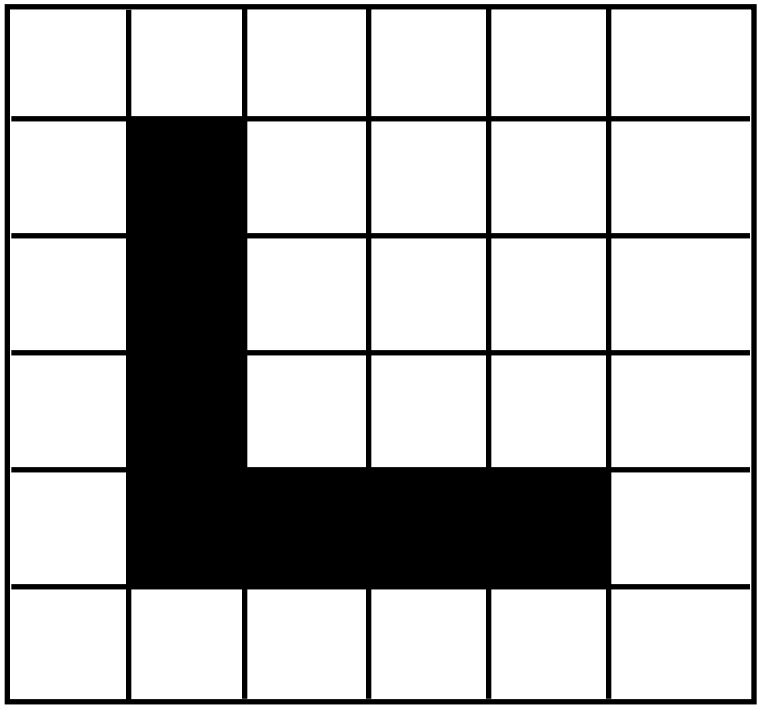


Maze Router Step 2: Backtrace

3	2	3	4	5	6
2	S 1	2	3	4	5
3	2	3	4	5	6
4	3	4	5	6	7
5	4	5	6	T 7	
6	5	6	7		

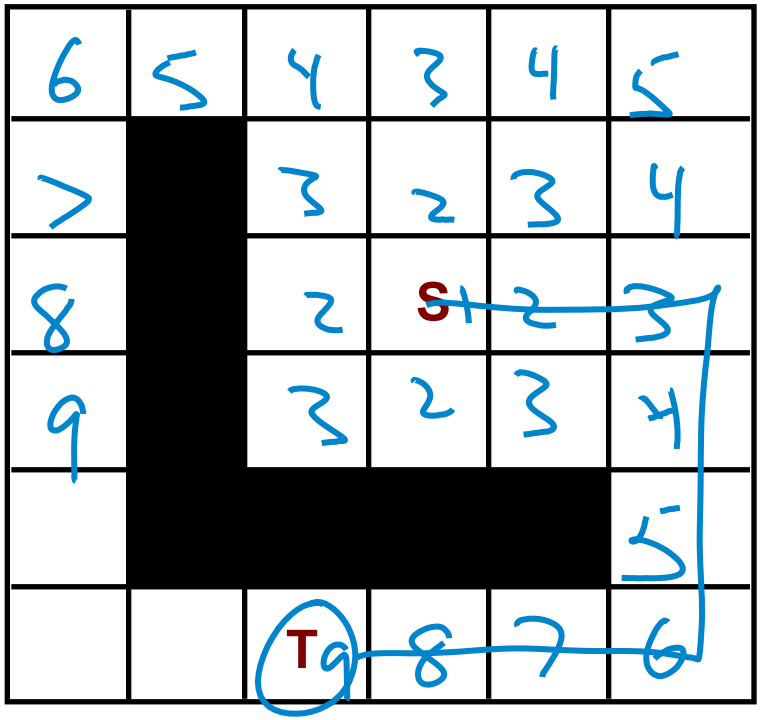
- **Now what? Backtrace**
  - Select a shortest-path (**any** shortest-path) from target back to source
  - Mark its cells so they cannot be used again – mark them as **obstacles** for later wires we want to route
  - Since there are many paths back, optimization information can be used to select the best one
  - Here, just follow the pathlengths in the cells **in descending order**

Maze Router Step 3: Clean-Up



- **Now what? Clean-up**
  - Clean up the grid for the next net, leaving the new **S** to **T** path as an **obstacle**
  - Now, ready to route the **next** net with the obstacles from the previously routed net in place in the grid

Maze Router: Obstacles



- **Also called "Blockages"**
  - Any cell you cannot use for a wire is a an **obstacle** or a **blockage**
  - There may be parts of the routing surface you just cannot use
  - But most importantly, you **label each newly routed net as a blockage**
  - Thus, all future nets must **route around** this blockage

Classical Maze Router

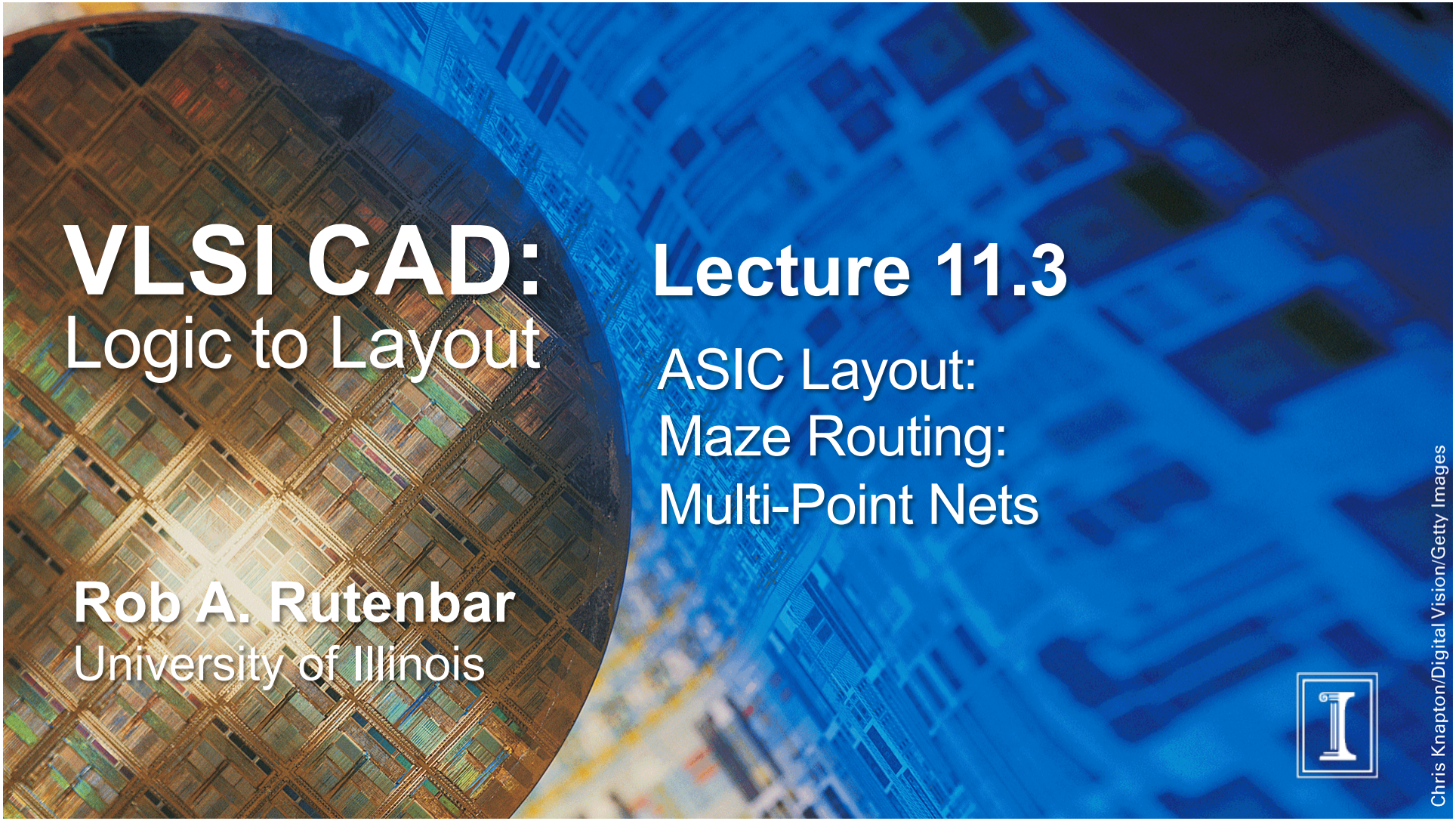
- **Summarizing three main steps:**

- **Expand**
  - Breadth-first-search to find all paths from source to target, in pathlength order
- **Backtrace**
  - Walk shortest path back to the source and mark path cells as used (obstacles)
- **Clean-Up**
  - Erase all distance marks from other grid cells before next net is routed

Maze Router: Issues

- **Applications & features**
  - How to route more than 2 point nets? How to handle more than 1 wiring layer?
  - How do we deal with vias (connecting different routing layers?)
- **Implementation**
  - Do we need a really big grid to represent a big routing problem?
  - What information is required in each cell of this grid?
  - Do we really have to search the whole grid each time we add a wire?

- **We will look at both kinds of concerns in remainder of lecture**



VLSI CAD: Logic to Layout

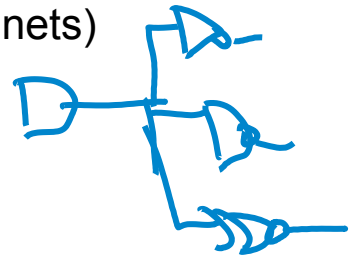
Lecture 11.3

ASIC Layout:
Maze Routing:
Multi-Point Nets

Rob A. Rutenbar
University of Illinois

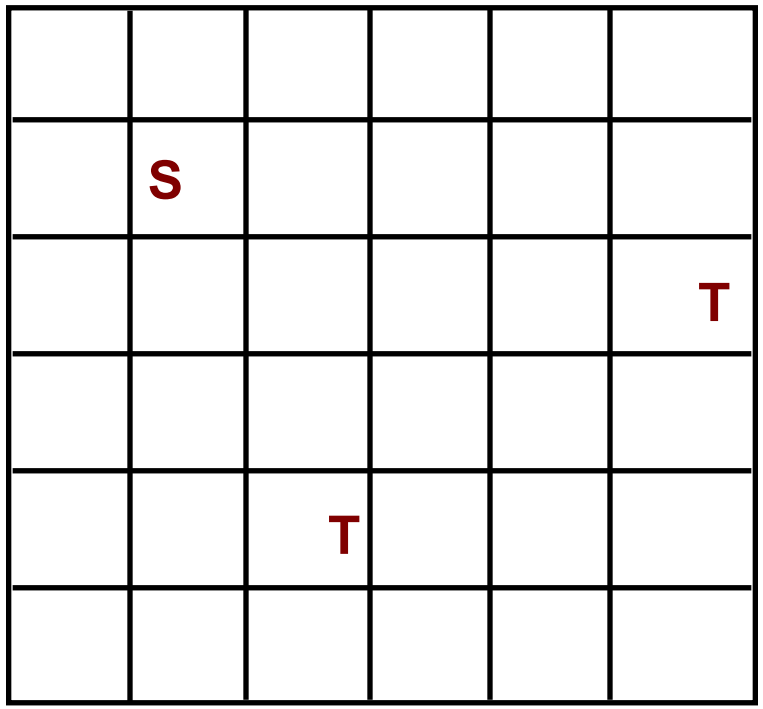
Applications: Multi-point Nets

- **Multi-point Nets**
  - **One** source → **Many** targets
  - You get this with any net that represents **fanout** (ie, almost all real nets)



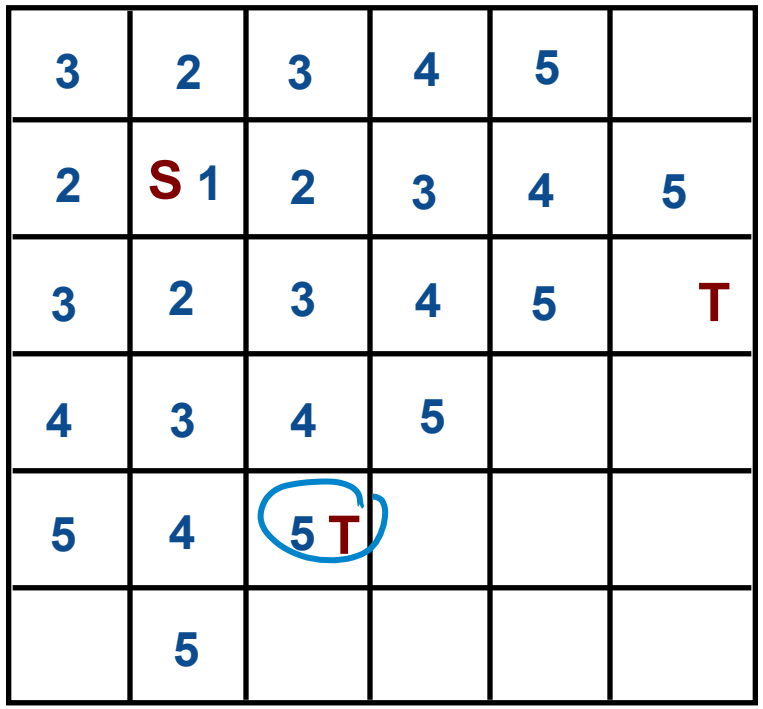
- **Simple strategy**
  - **Start**: Pick **one** point as source, label **all the others** as targets
  - **First**: Use maze route algorithm to find path from source to **nearest** target
    - **Note**: You don't know which one this "nearest" is just yet, routing will find it
  - **Next**: Re-label all cells on found path as sources, then rerun maze router using all sources simultaneously
  - **Repeat**: For each remaining unconnected target point

Multi-point Nets



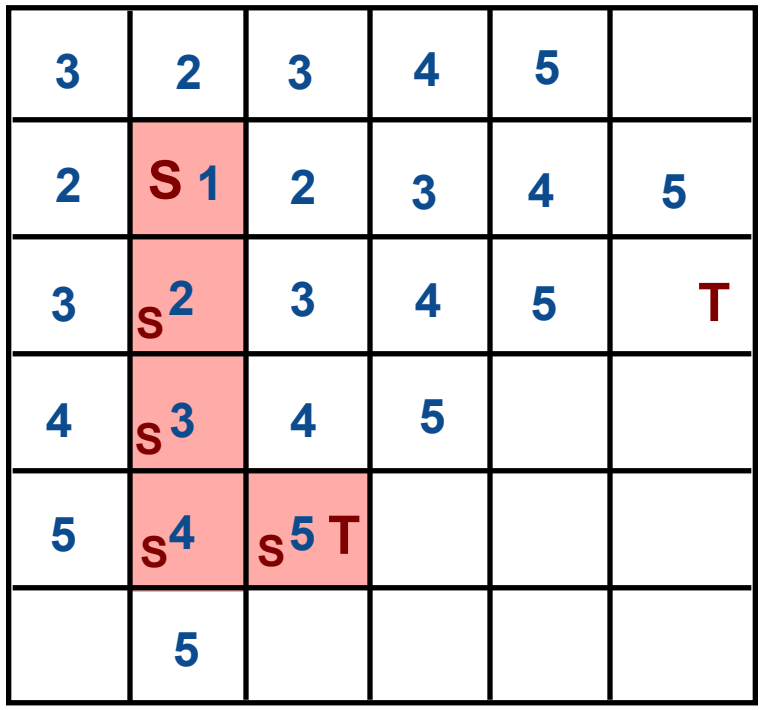
- **Given:**
  - A source and **many targets**
- **Problem:**
  - Find a **short** path connecting source and targets

Multi-point Nets



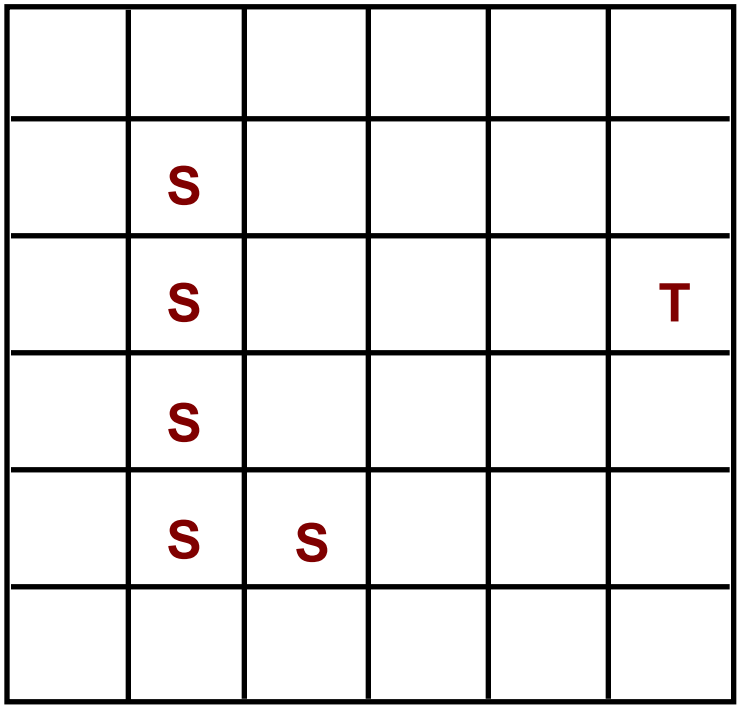
- **First segment of path...**
  - Run maze route to find the closest target
  - Start at source, go till we find **any target**

Multi-point Nets



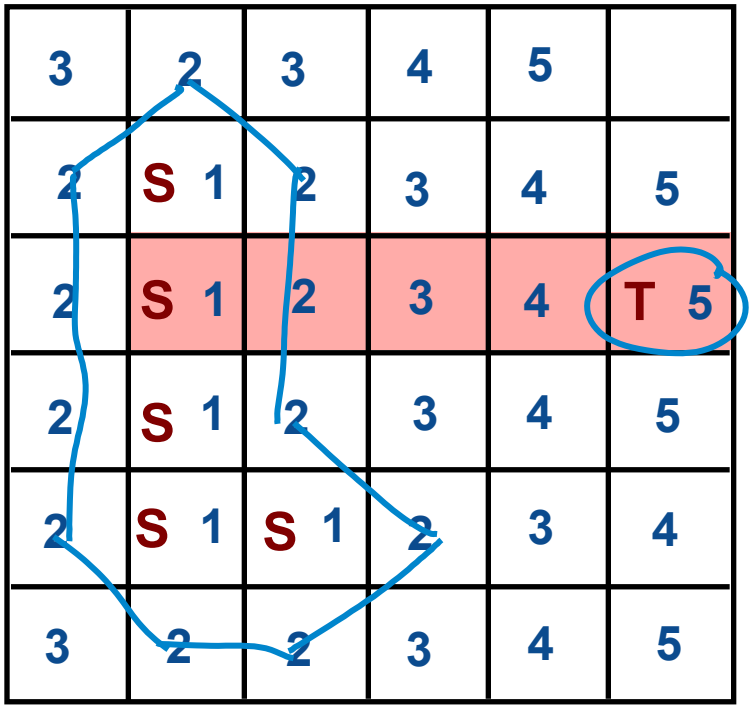
- **Second...**
  - **Backtrace and re-label the whole route as Sources for the next pass**
- **Note – this is different**
  - We don't relabel the path as blockage (yet), as we did before
  - We label it as source, so we can find paths from any point on this segment, to the rest of the targets

Multi-point Nets



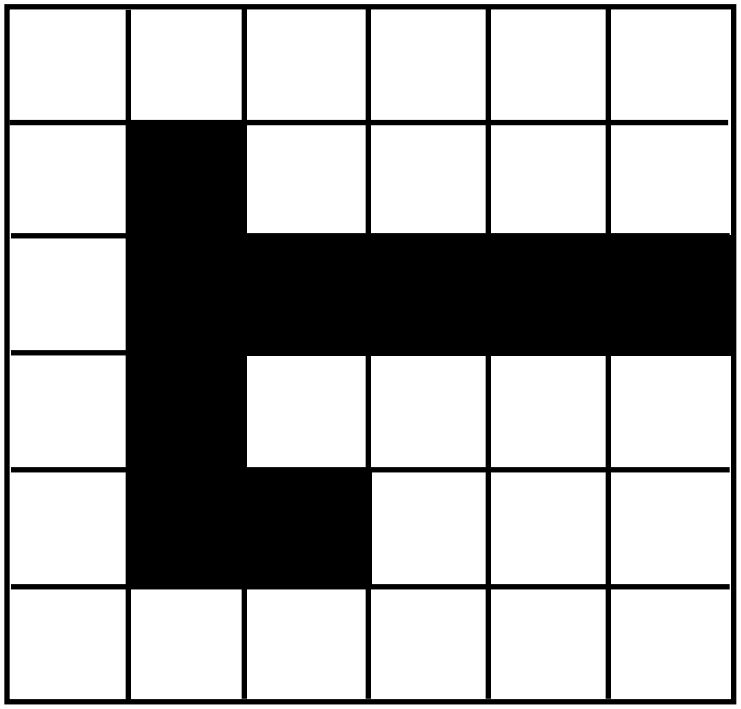
- **Second...**
  - We will **expand this entire set** of source cells to find next segment of the net
  - Idea is we will look for paths of length 1 away from this whole set of sources, then length 2, 3, etc.
  - Go till we reach another target

Multi-point Nets



- **Trick**
  - Expand from **all these sources** to find the shortest path from the existing route to the next target
- **Next: Backtrace as before**
  - Follow pathlengths in decreasing order from target, to **some** source cell

Multi-point Nets



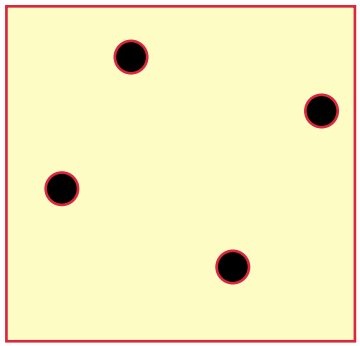
- **Finally**
  - Do usual cleanup
  - Mark all of the segment cells as used and clean-up the grid
  - Now, have embedded a **multi-point net**, and rendered it an obstacle for future nets

Aside: Is This Strategy "Optimal"?

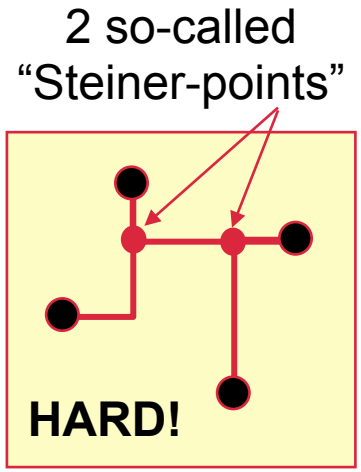
- **Does this method give us guaranteed shortest multi-point net?**
- **No!**
  - Maybe surprising, but this is just a good heuristic
  - The optimal path has a name: called a **Steiner Tree**

- **How hard is to get the optimal Steiner Tree?**
  - NP-hard! (ie, exponentially hard)
  - Yet another example of why CAD is full of tough, important problems to solve

Aside: About Steiner Tree Constructions

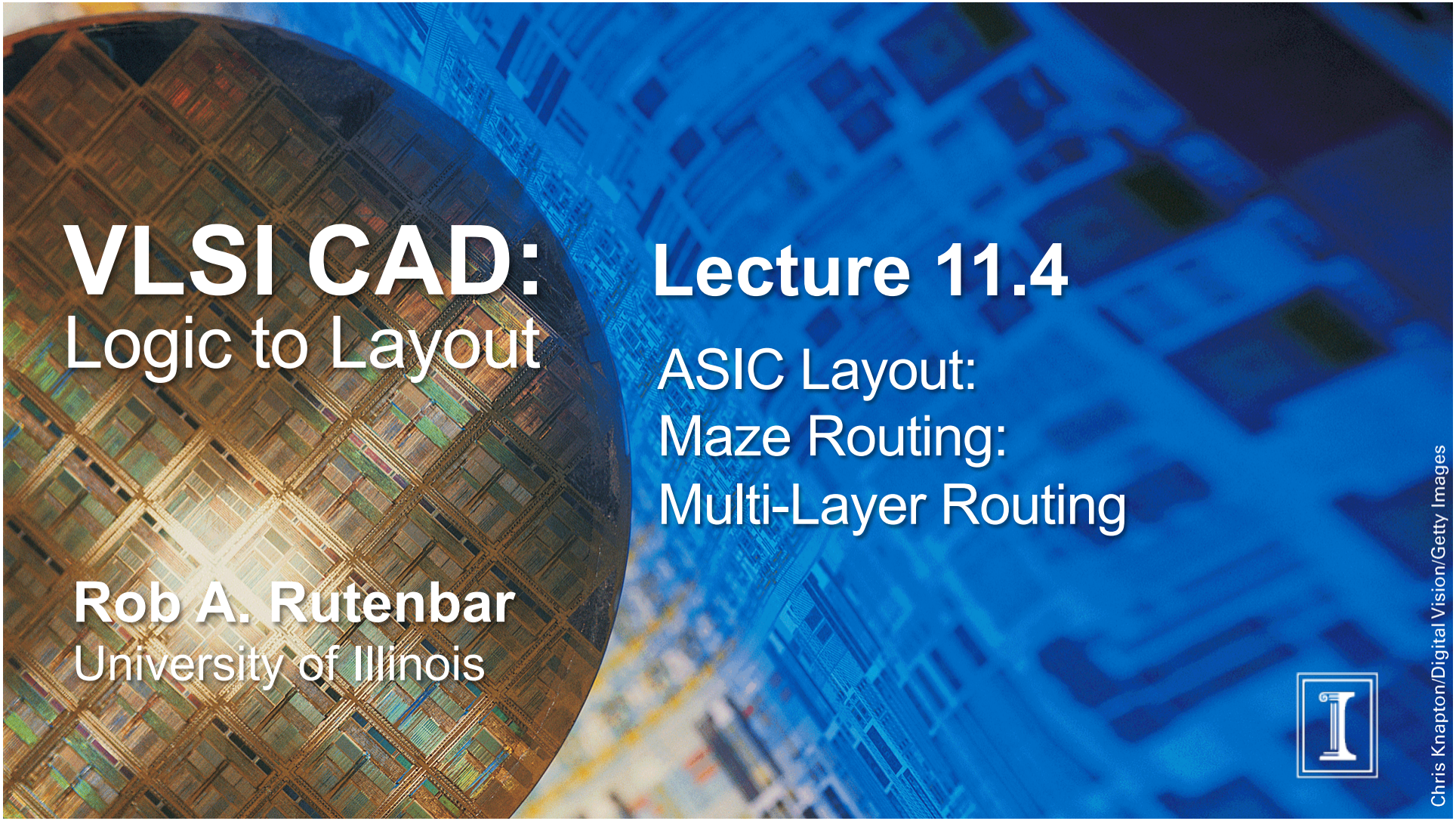


Pins to connect



Best possible route:
Minimum Steiner Tree

- **Why this is hard, in general**
  - You don’t know at start where the right “Steiner points” are, to make shortest wire
  - Simple heuristics work well – but no guarantee to get the very best Steiner tree



VLSI CAD: Logic to Layout

Lecture 11.4

ASIC Layout:
Maze Routing:
Multi-Layer Routing

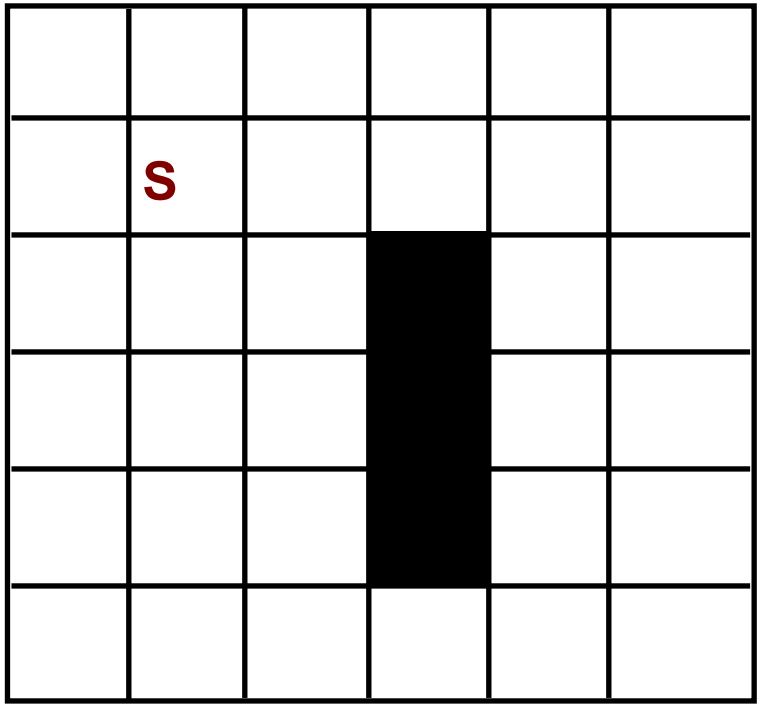
Rob A. Rutenbar
University of Illinois

Application: Multi-Layer Routing

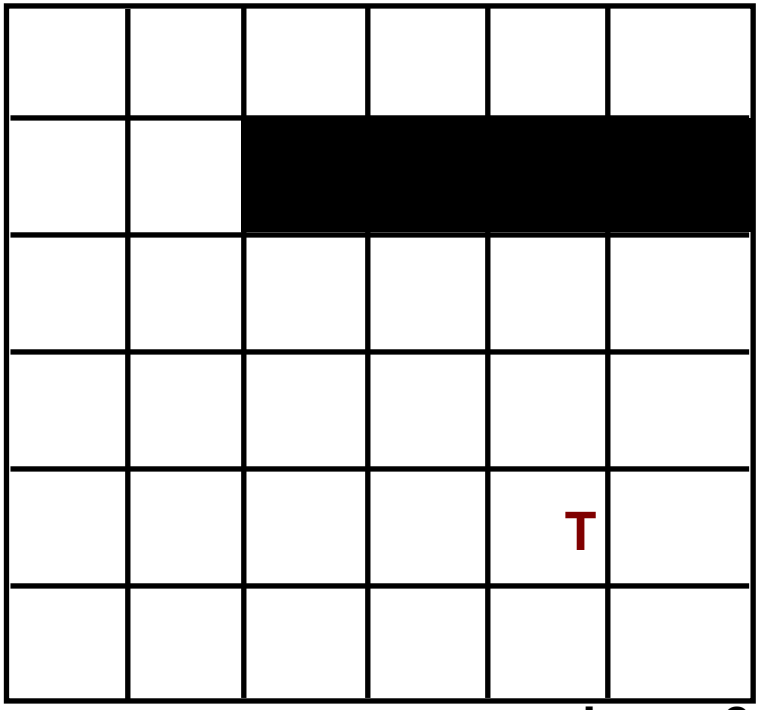
- **All chips (and all boards) have many parallel wiring layers**
  - **Chips**: 10+ layers of metal in modern technologies
  - **Boards**: 20+ layers in the most advanced boards
- **How – mechanically – do we handle multiple wiring layers?**
  - **Idea: Parallel grids, vertically stacked, one for each layer**
  - Use **vias** to access other layers.
- **New expansion process**
  - Out on each layer – to north/south/east/west neighbors
  - To adjacent layers -- **up/down** (between layers) using a via

2 Layer Example: Parallel Grids, Vertically Stacked

- **Expansion can now go UP (Layer 1→2) and DOWN (Layer 2→1)**



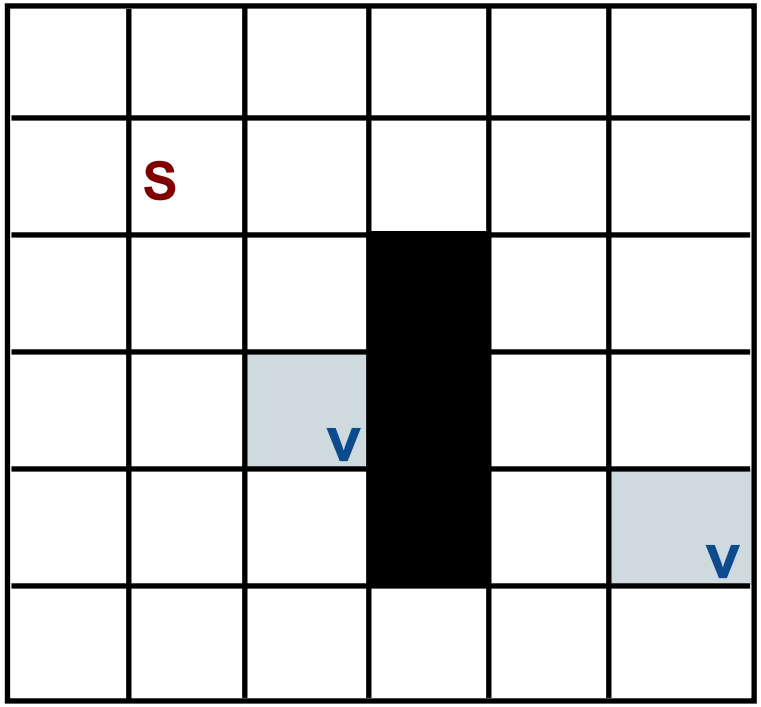
Layer 1



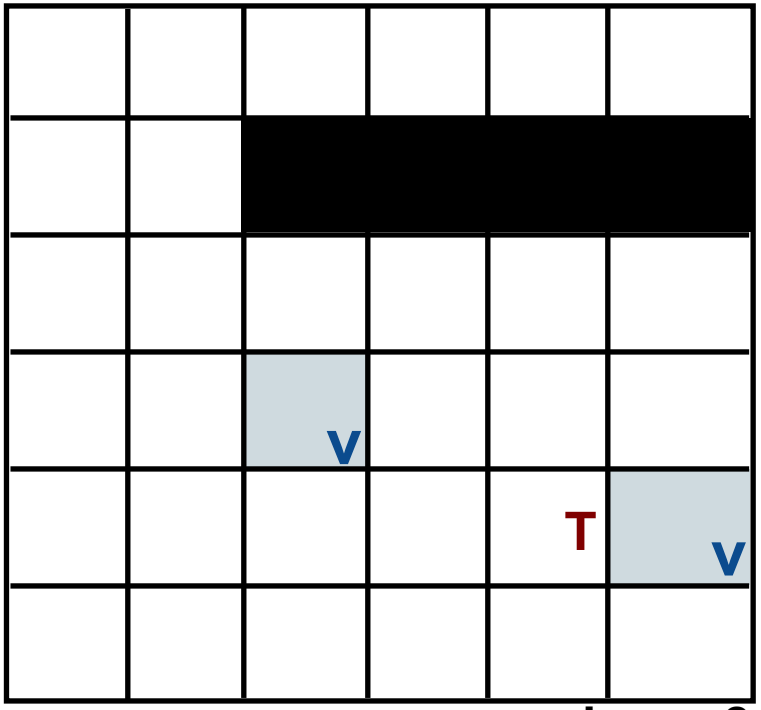
Layer 2

2 Layer Example: Expand Step

- **Artificial constraint: vias only allowed where "v" in grid, for simplicity**



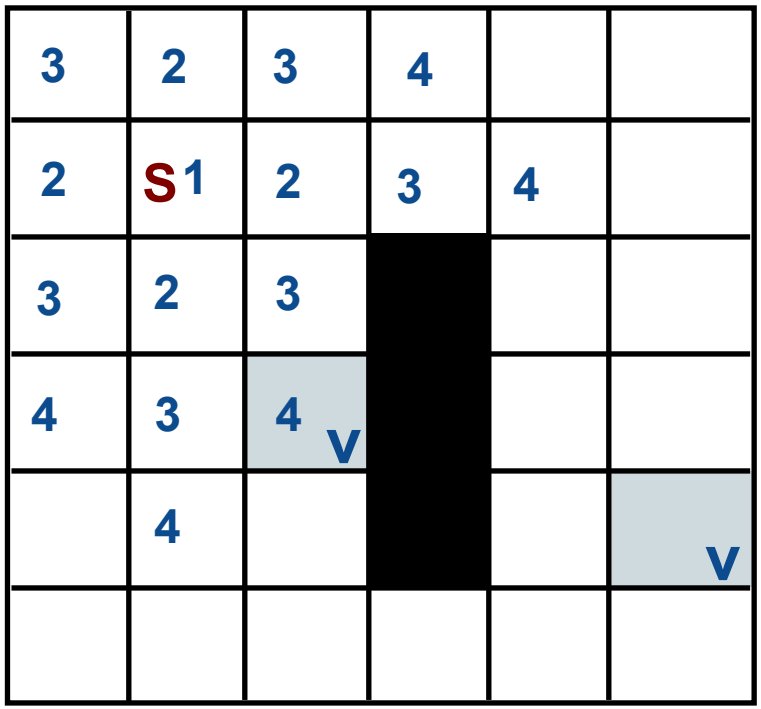
Layer 1



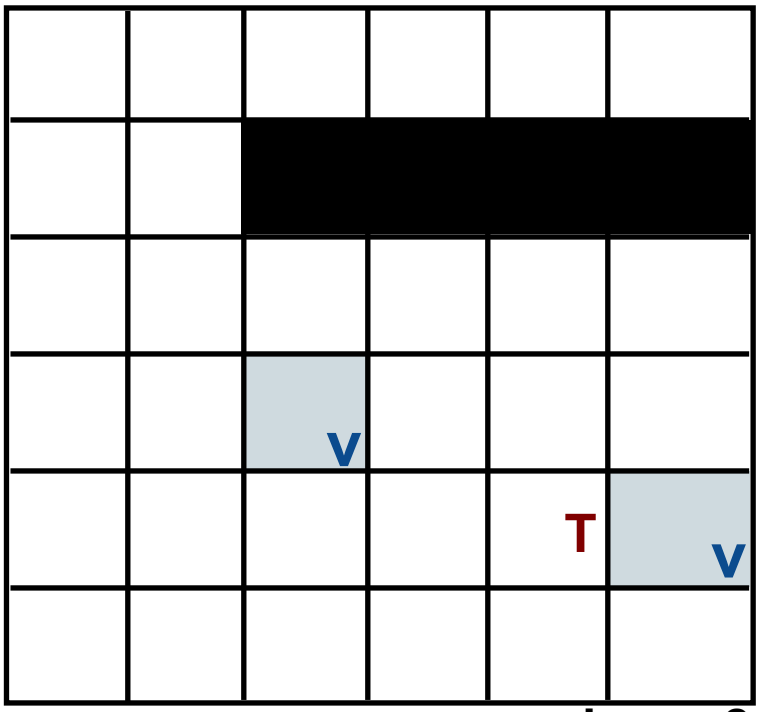
Layer 2

2 Layer Example: Expand Step

- **New: What happens– when we reach a place we can put a via?**



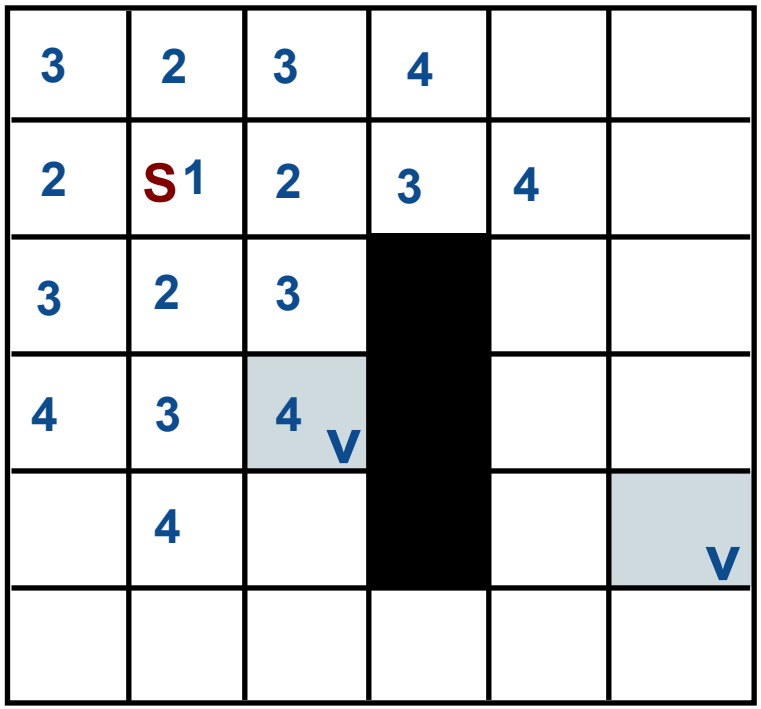
Layer 1



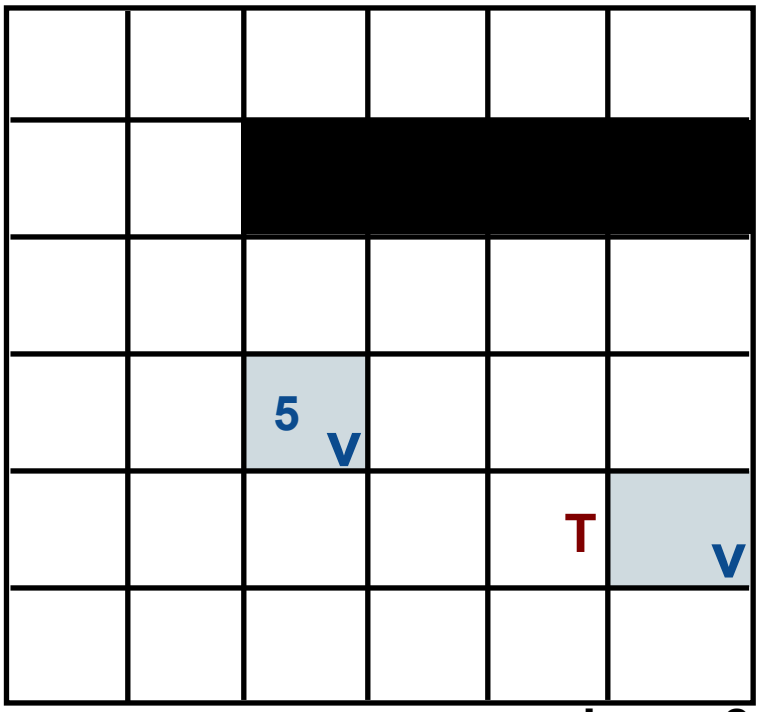
Layer 2

2 Layer Example: Expand Step

- **Expand on the other layer, keep adding unit cell costs…**



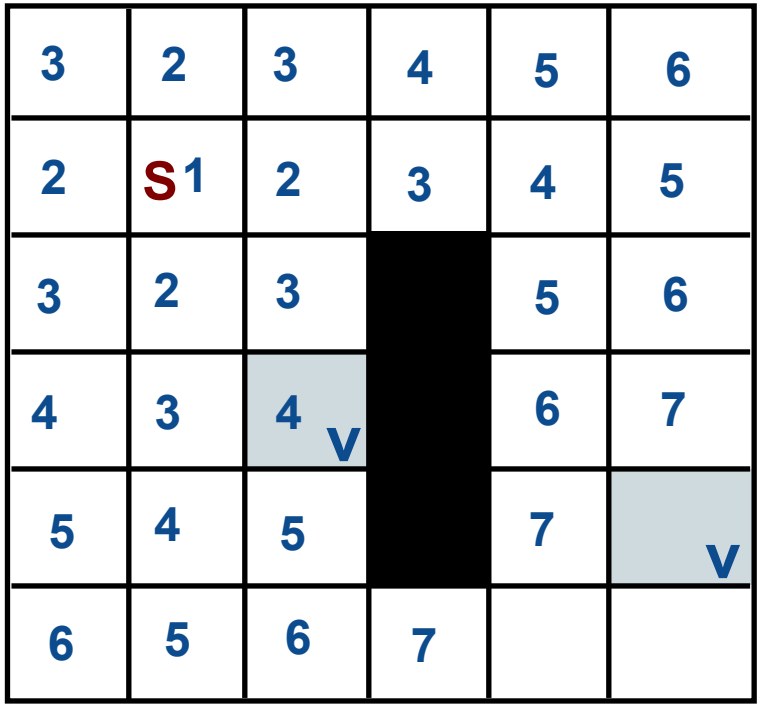
Layer 1

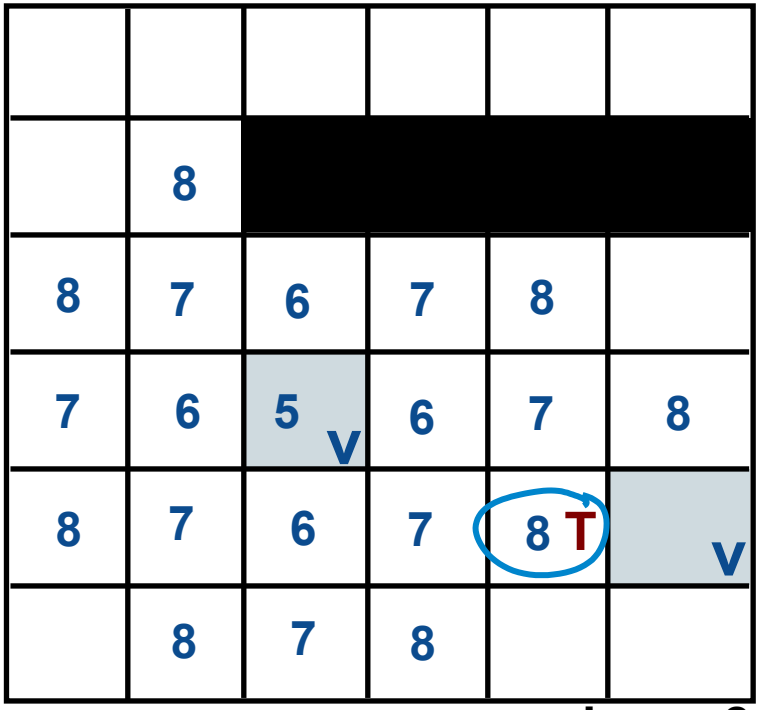


Layer 2

2 Layer Example: Expand Step

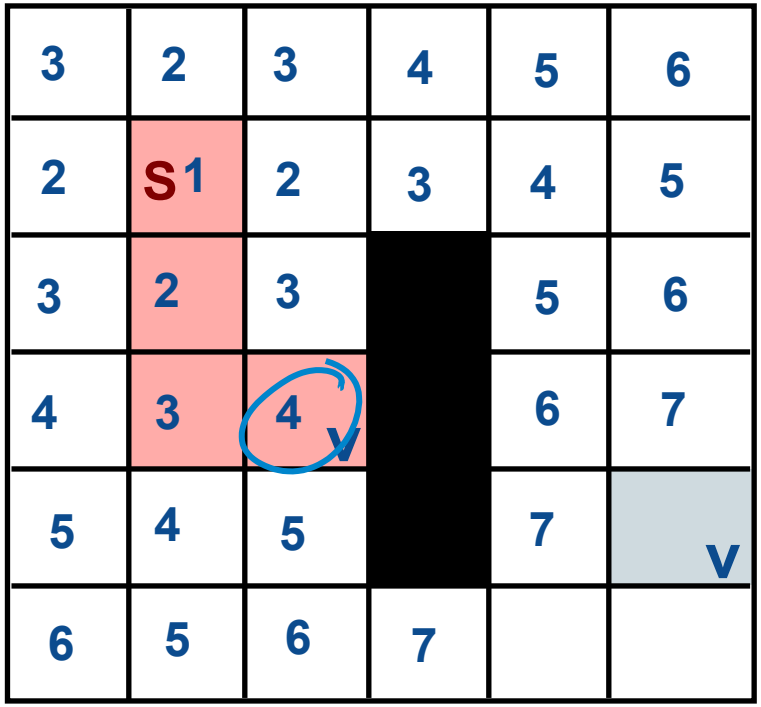
- **Now, expanding on both layers**



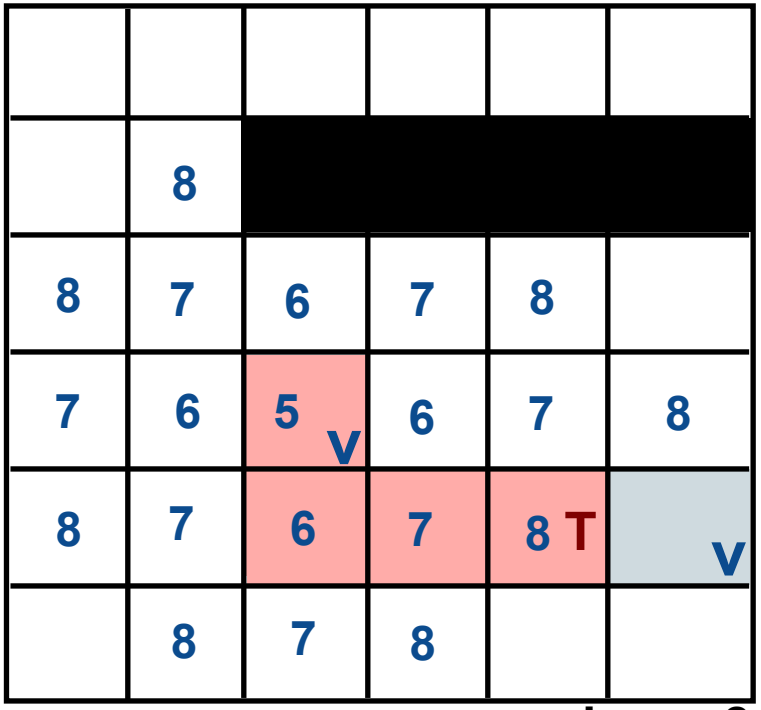


2 Layer Example: Expand Step

- **Next, backtrace, through cells and also through vias**



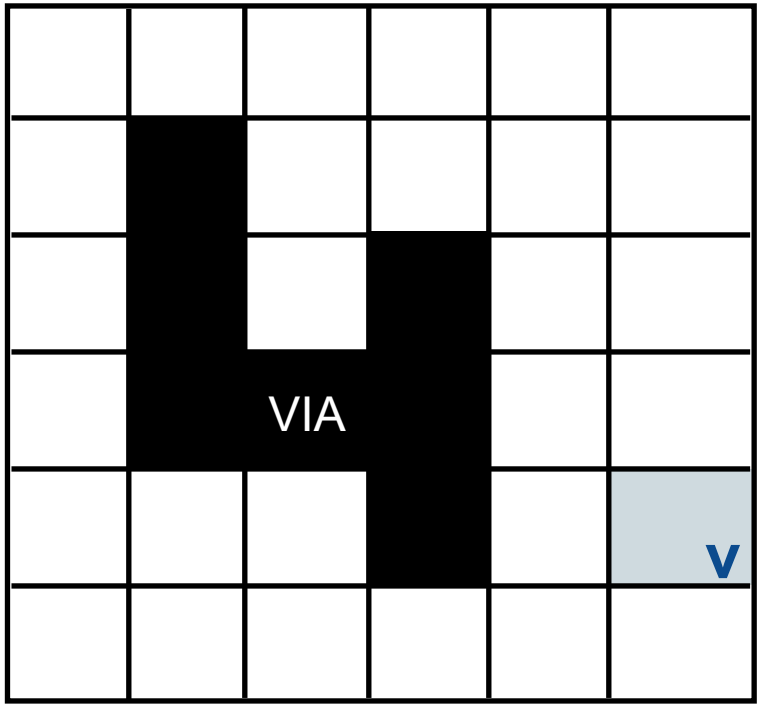
Layer 1



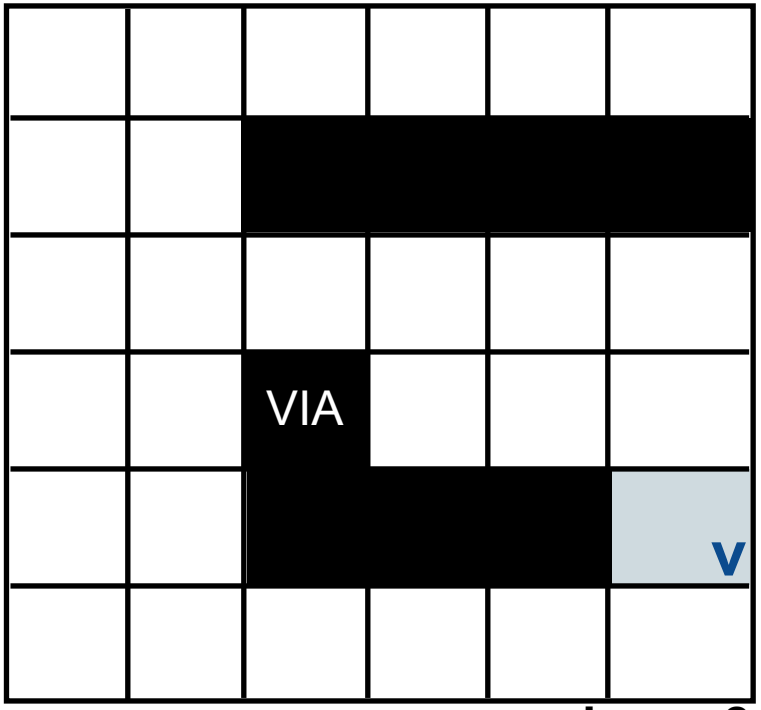
Layer 2

2 Layer Example: Expand Step

- **Next, cleanup as usual. Now, new paths (obstacles) on both layers**



Layer 1



Layer 2

Implementation Preview….

- **Real routers deal with many layers like this**
  - Technology will tell you exactly where vias can go, and to what other layers

- **But this raises an immediate question**
  - Are vias expensive? (**Yes!**) Should they just cost "1" like all other cells? (**No!**)
  - In real routers, you make vias expensive so you only use them where essential
  - How do we "make vias expensive"?
  - We need a more general notion of "**cost**" in the routing grid … do this next

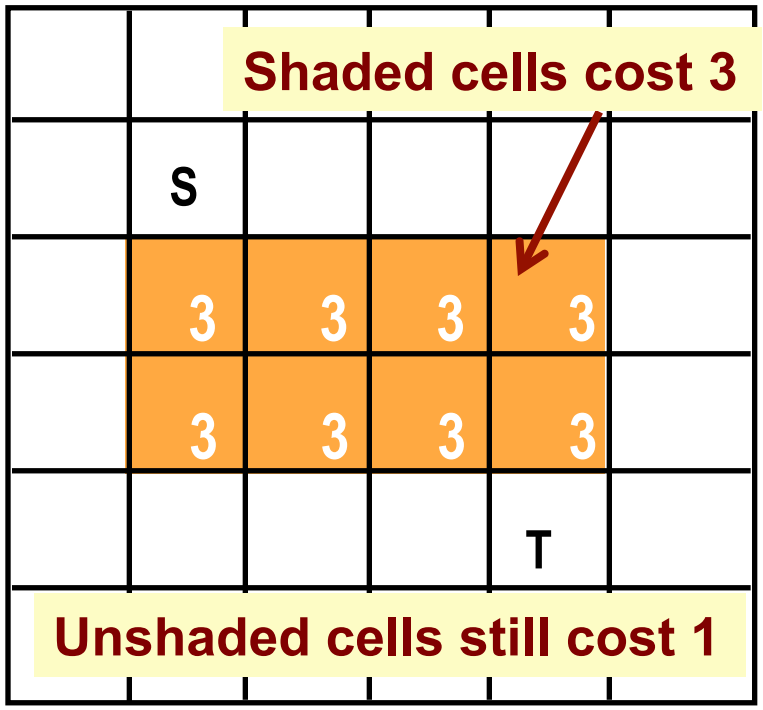
VLSI CAD: Logic to Layout

Lecture 11.5

ASIC Layout:
Maze Routing:
Non-Uniform Grid Costs

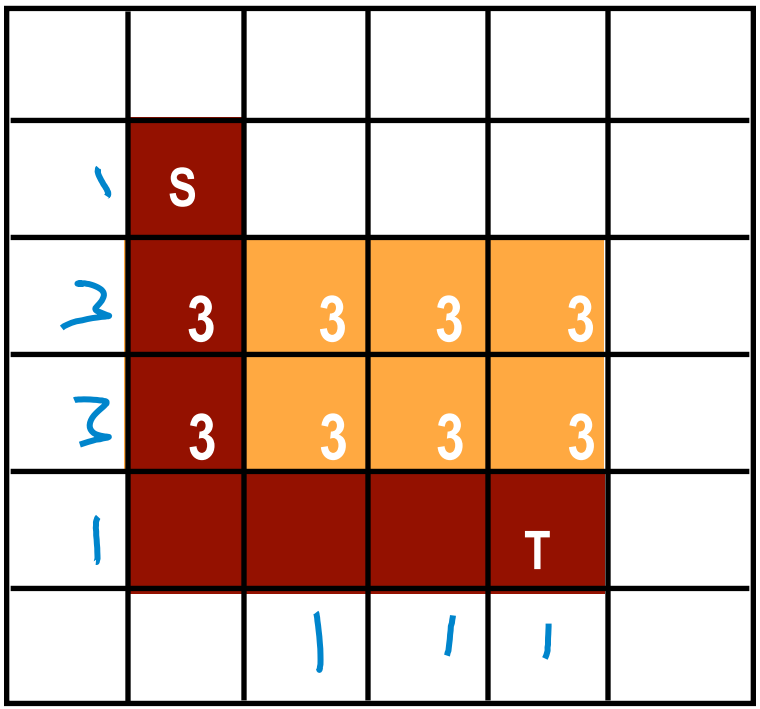
Rob A. Rutenbar
University of Illinois

New Feature: Non-Uniform Grid Costs



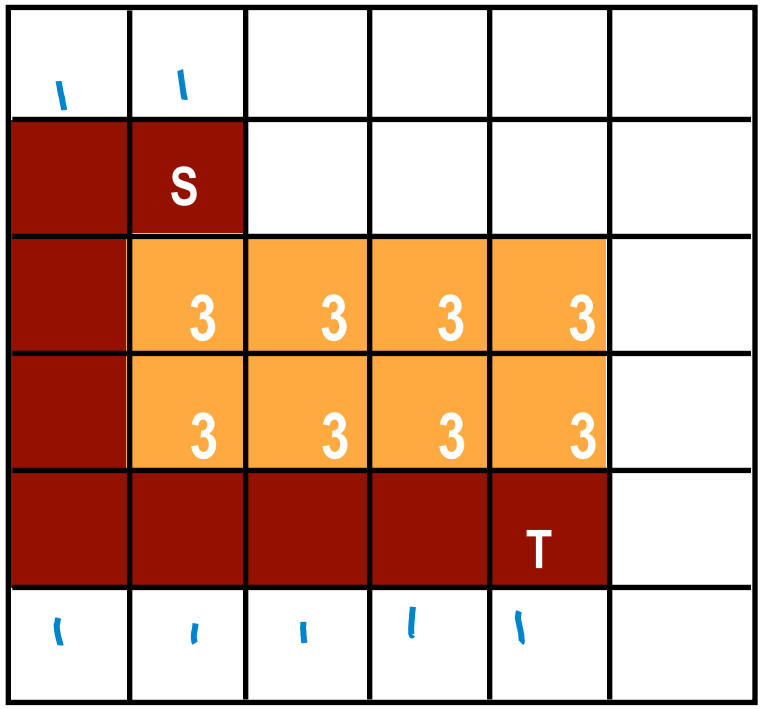
- **Old problem**
  - Each cell in grid costs the **same** to cross it with a wire
  - **Cost==1, ie, unit-cost, uniform all over**
  - Is this necessary? No!
- **Now**
  - Given grid, Source and target
  - **Different costs for each cell**
- **New problem:**
  - Find **minimum cost** path connecting source and target

Define Path Cost == $\sum_{path}$ (cell costs)



$\sum_{path}$ (cell costs) = 11

Short. Expensive!



$\sum_{path}$ (cell costs) = 9

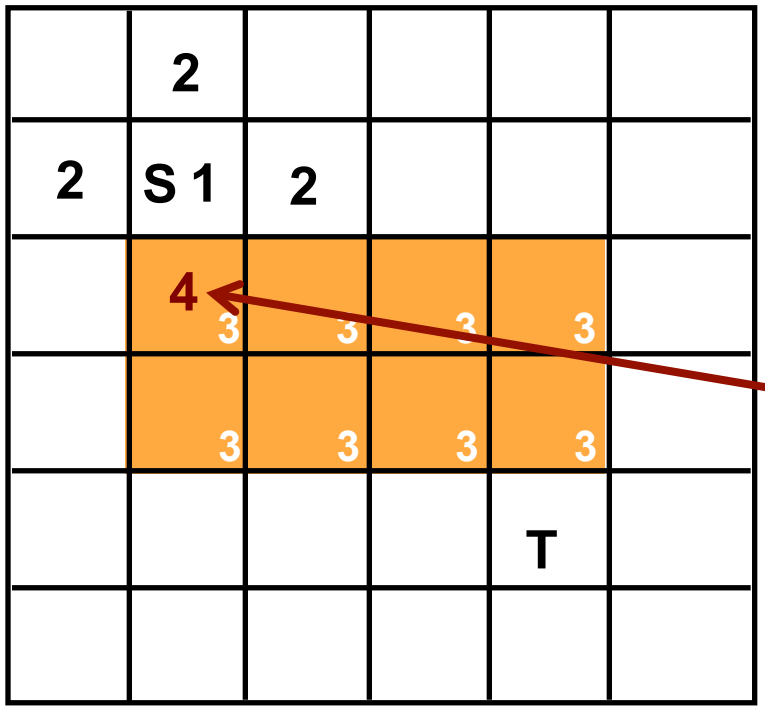
Long. Cheap!

Feature: Non-Uniform Grid Costs

- **Vital feature of all real routers**
  - **Use costs to encourage wires to take shapes and paths we prefer**
  - Can make the router **avoid congested areas** (too many wires want to be here)
  - Can make **different layers** have different expense to use
  - Can make **different vias** have different expense to use
  - Can make **different directions of expansion** have different expense
    - Example: you want metal 2 mostly vertical, so left-right expansions cost more...

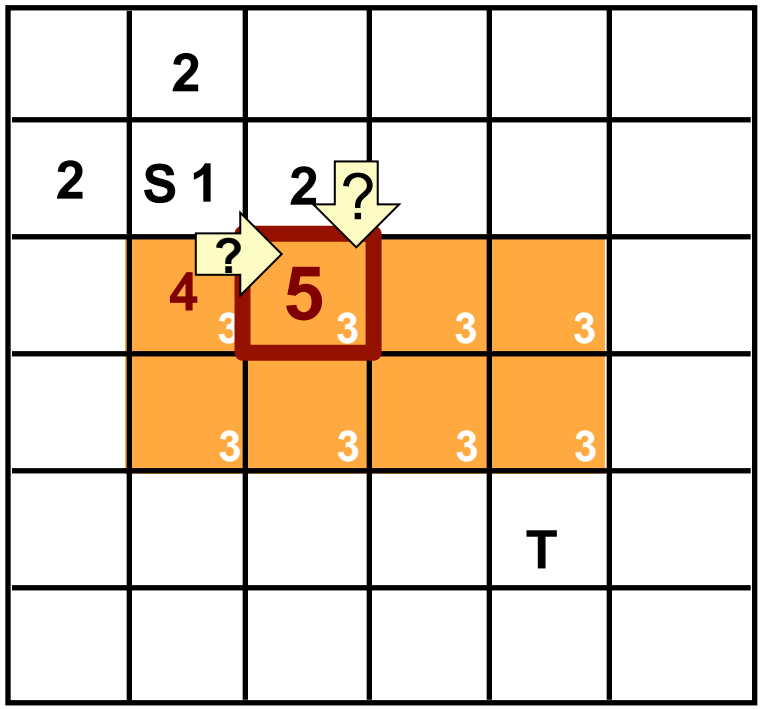
- **Search process of expanding out from source uses costs….**
  - What does it **cost to reach** one of the non-unit cost cells? To **put a via** here?

Subtle Search Issues with Non-Unit Costs



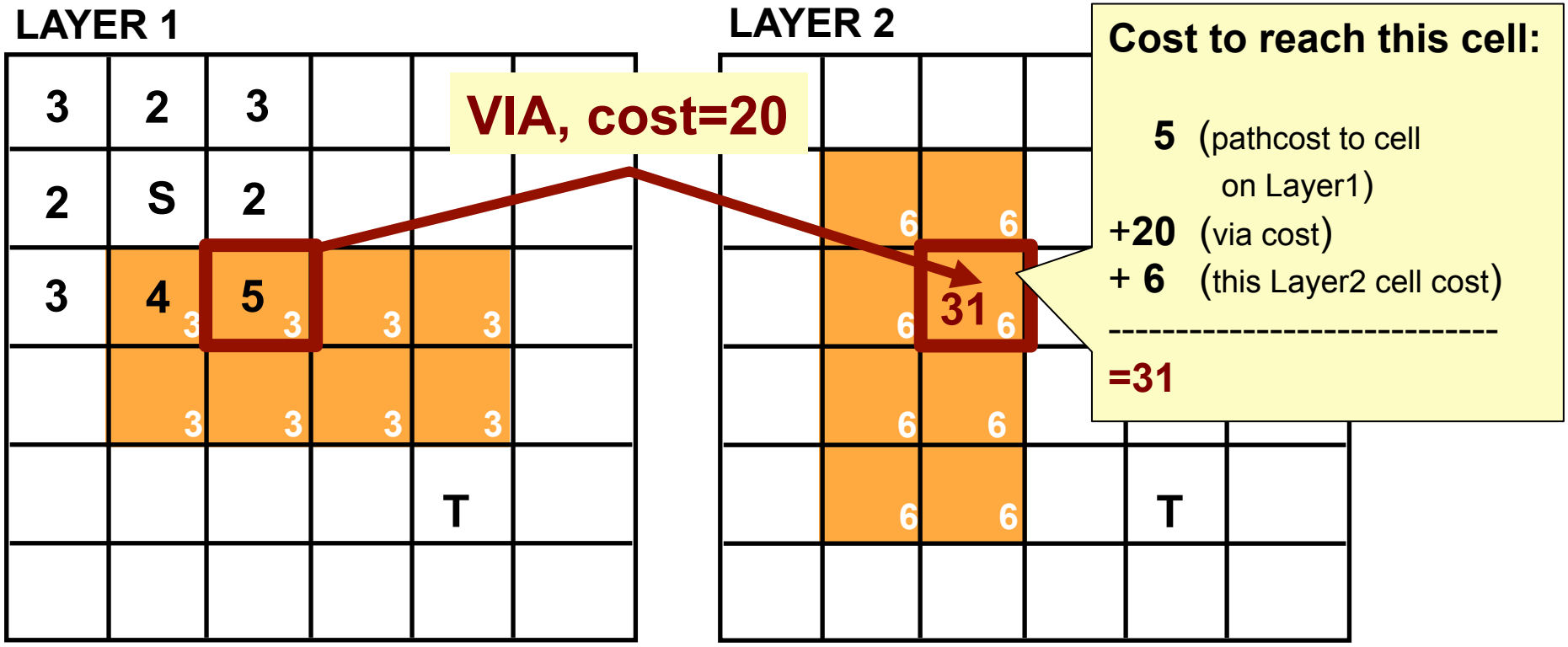
- **Search all paths that are 1 cell distant from source S**
  - When we reach a new cell, we label it with a **pathcost**
  - **pathcost** = pathcost of neighbor used to "reach" this cell, plus cell's own cost
  - So, this cell is **1+3=4**
- **So far, so good…**

Subtle Search Issues with Non-Unit Costs



- **But, what is this cell's pathcost to be "reached"?**
  - Is it **2+3=5**, reached "from" cell with cost **2** that is to the North?
  - Is it **4+3=7**, reached "from" cell with cost **4** that is to the West?
- **Answer: 5**
  - Always want to label cells with the **minimum pathcost to reach that cell**
  - Again: this is sum of costs of all cells from source to this cell, on this path

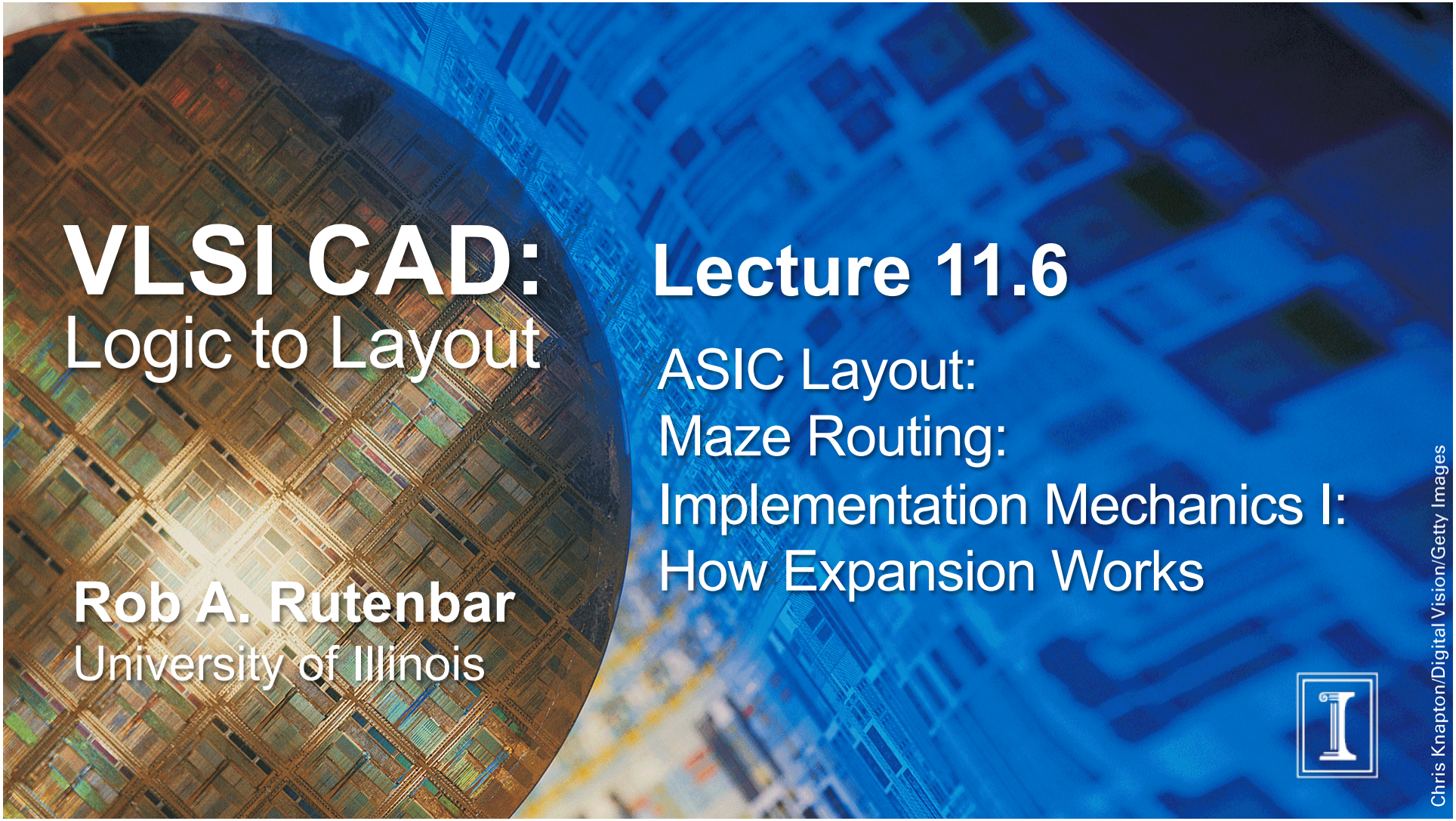
Subtle Search Issues: Via Costs



Suppose we want to "expand" from this "5" on Layer1, **through a via**, to Layer2

Maze Routing: Mid-Point Summary

- **What do we know?**
  - Grid-based expansion, one net at a time
  - Can use costs in grid to get different effects
  - Can deal with multiple wiring layers, multi-point nets
- **What don't you know?**
  - Real implementation strategies, real data structures
  - How cells gets "touched" during search: Expanding vs Reaching a cell
  - Subtle interactions between cost strategy and search strategy
- **Next topics: Real implementation mechanics**



VLSI CAD: Logic to Layout

Lecture 11.6

ASIC Layout:
Maze Routing:
Implementation Mechanics I:
How Expansion Works

Rob A. Rutenbar
University of Illinois

Implementation Concerns

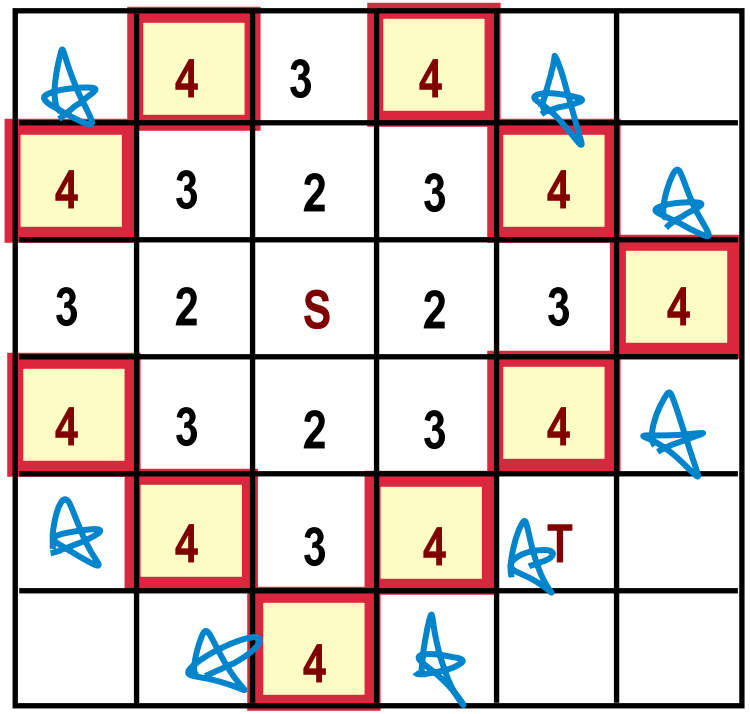
- **Representation**
  - How do we store the routing grid?
  - What do we need in each cell?
  - How do we represent the state of the advancing path search process?

- **Algorithms**
  - **Question**: if we can only process one cell at a time...
  - ...then, which cell is next to “label” in the search process
  - Does the order matter? How can we do this as fast as possible?

Big Idea: Search Wavefront

- **One big goal**
  - Efficient storage: Big layout needs a big grid; put as little in each cell as possible
- **Big Idea: Do not store path costs in grid cells**
  - Big costs → many bits per cell
  - Only cells **most recently labeled** during search are used to **expand** the search
  - Important terminology: these cells comprise the **search wavefront**
- **It is the wavefront that is really most important….**

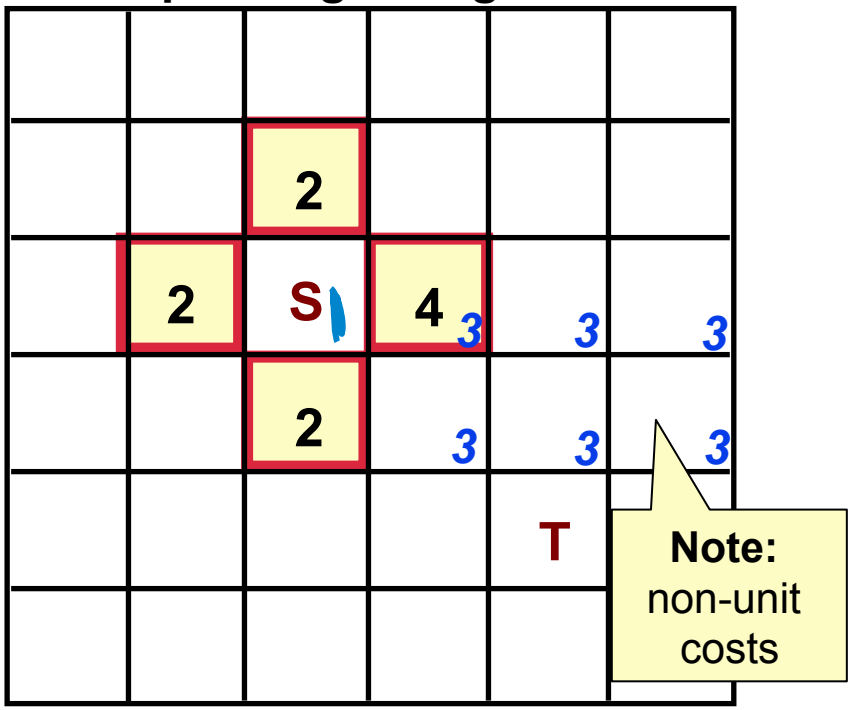
Example Wavefront for Simple Search



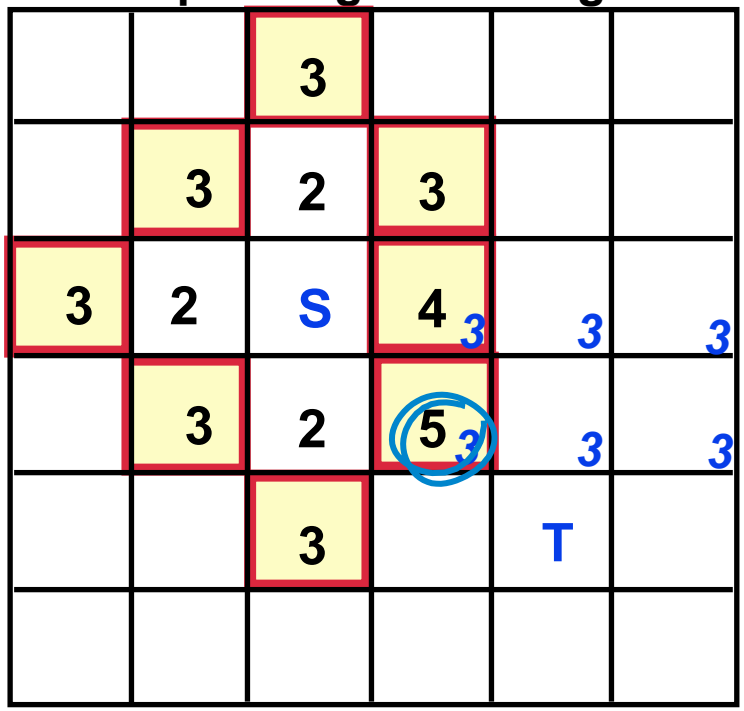
- **Wavefront is...**
  - The **frontier** of the active search for new paths
  - The **neighbors** of the new cells worth looking at to try to extend the evolving path search
  - The **only cells** we need to look at to decide how to continue the search process
- **Implication**
  - **Don't** store the pathcost numbers **in the grid**
  - Just store the wavefront cells themselves in a **special data structure**

More Complex Wavefront

After expanding 4 neighbors of S

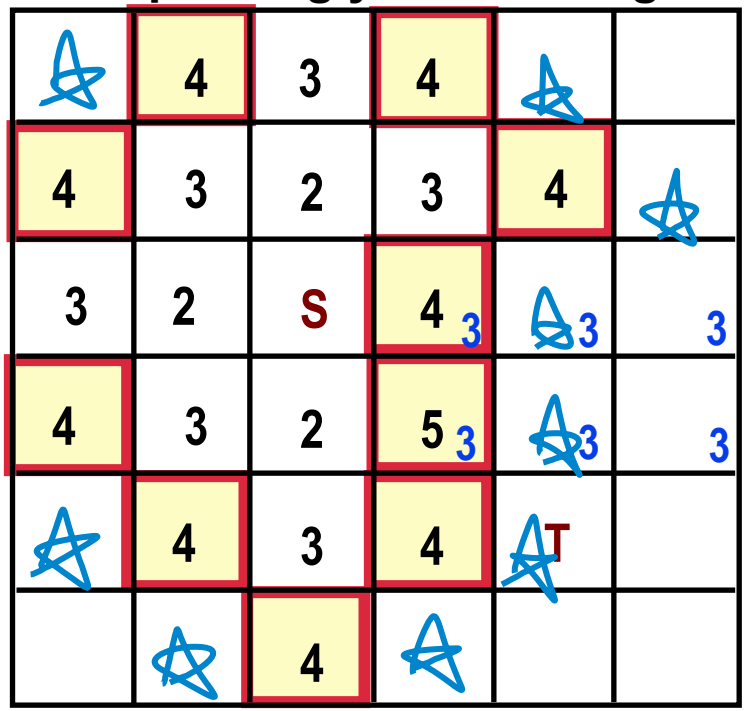


After expanding more neighbors



More Complex Wavefront

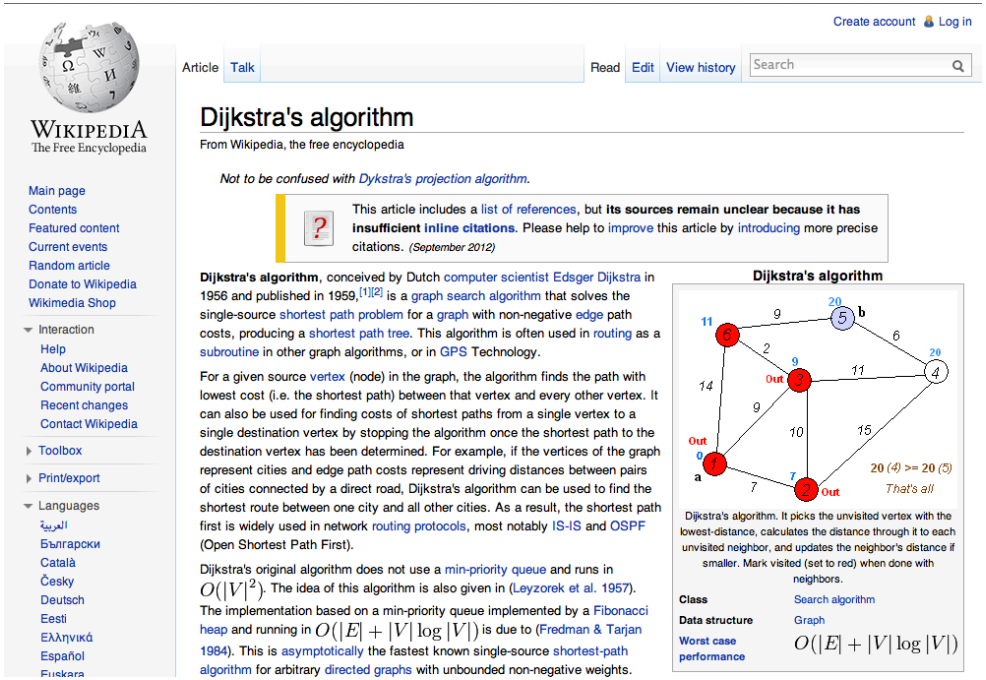
After expanding yet more neighbors



- **What wavefront is...**
  - Set of cells already **reached** in the expansion process...
  - ...that have **neighbors that** we have **not** yet reached
  - Indexed by **pathcost** of cells reached (== cost of path that starts at source **S** and ends at this cell)
  - **Expanded** in pathcost order, cheapest cells before more expensive cells

Outline of Expansion Algorithm

- **Cheapest-cell-first search**
  - Variant of **Dijkstra's algorithm**
  - Originally invented in 1950s for shortest path search through a graph with weighted edges
  - Works for gridded maze routing
  - Assume wavefront is a cost-indexed list of cells already visited during search, and "labeled" with **pathcost**
    - **Pathcost**: cost of partial path from source cell to this cell
    - **Sum** of all cell costs on this path



Create account Log in

Article Talk | Read Edit View history | Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export
Languages
العربية
Български
Català
Česky
Deutsch
Eesti
Ελληνικά
Español
Euskara

Dijkstra's algorithm

From Wikipedia, the free encyclopedia

Not to be confused with Dykstra's projection algorithm.

This article includes a list of references, but **its sources remain unclear because it has insufficient inline citations.** Please help to improve this article by introducing more precise citations. *(September 2012)*

Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959,[1][2] is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is often used in routing as a subroutine in other graph algorithms, or in GPS Technology.

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First).

Dijkstra's original algorithm does not use a min-priority queue and runs in $O(|V|^2)$. The idea of this algorithm is also given in (Leyzorek et al. 1957). The implementation based on a min-priority queue implemented by a Fibonacci heap and running in $O(|E| + |V| \log |V|)$ is due to (Fredman & Tarjan 1984). This is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights.

Dijkstra's algorithm

Dijkstra's algorithm. It picks the unvisited vertex with the lowest-distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors.

Class	Search algorithm
Data structure	Graph
Worst case performance	$O(\|E\| + \|V\| \log \|V\|)$

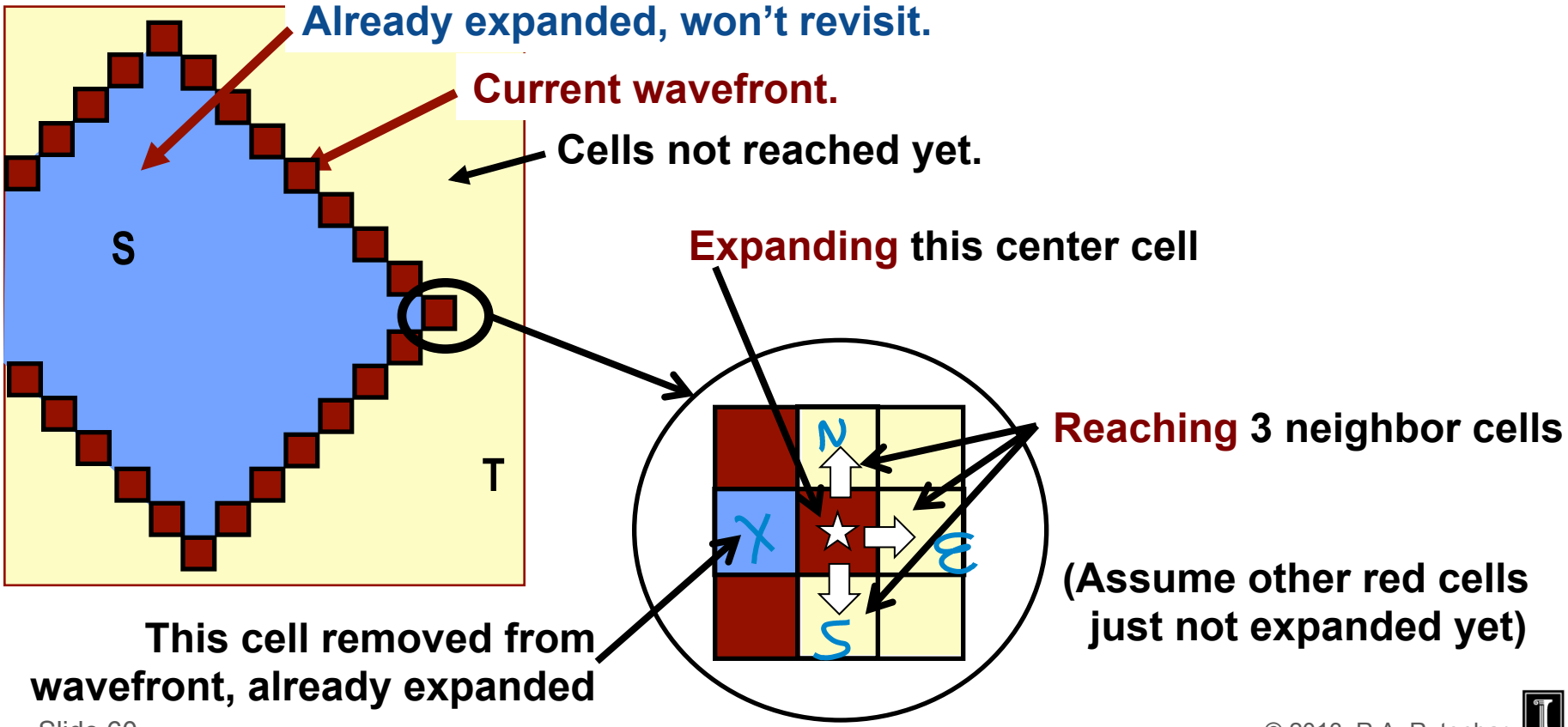
Outline of Expansion Algorithm

- **How does the wavefront grow?**
  - **Find** a cheapest cell **C** from the wavefront
  - **Find** the neighbors **{N1, N2, ... Nk}** of cell **C** that you have ***not reached*** yet
  - **Compute** the cost of expanding this path to reach these new cells in **{N1, N2, ... Nk}**
  - **Add** these new cells **{N1, N2, ... Nk}** to the wavefront data structure, indexed by their newly computed pathcosts, **{pathcost(N1), pathcost(N2), …, pathcost(Nk)**}
    - **Also, add** to each of these new cells a pointer to cell **C** as the ***predecessor***
    - **Mark** the routing grid to remember that we have ***reached*** these cells **{N1, N2, ... Nk}**
  - **Remove** cell **C** from the wavefront
  - **Repeat** with the next cheapest cell on wavefront...

We Need Some Precise Terminology

- **Pathcost**
  - **Sum** of cell costs on a partial path from Source to this cell; used to index wavefront
- **Wavefront**
  - **Frontier** of cells for which we have computed the minimum pathcost
- **Reached**
  - A cell is **reached** when we compute a minimum pathcost for it, and add it to wavefront
- **Expanded**
  - A cell is **expanded** when we **remove** it from the wavefront, and use it to reach its unreached neighbors
- **Dijkstra's Approach**
  - Expand cells in their minimum pathcost order

Illustrating the Terminology



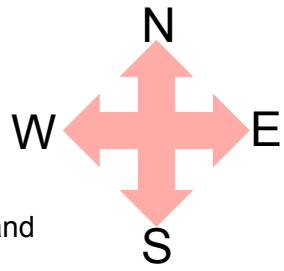
More Terminology: Cell Predecessor

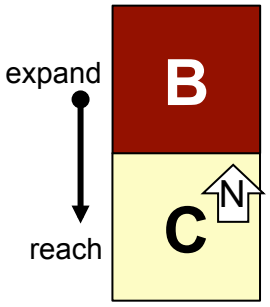
- **Predecessor of cell C on the wavefront: pred(C)**
  - **pred(C)** is a "tag" that tells **direction** from which cell **C** was reached; it's how we found **C**
  - We **mark the cell C** data structure when we **add** it to the wavefront
  - We **mark the grid** itself, at location **C**, when the cell is reached

- **Why do we have to remember this?**
  - Because we **do not mark** pathcosts in the grid any longer
  - To backtrace the path, we cannot simply follow the numbers in decreasing order
  - We need a real "trail" that points, cell to cell, from target back to source

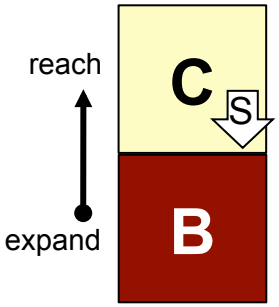
Marking the Predecessor

- **Assume 2 routing layers. 6 pred(C) tags: N, S, E, W, Up, Down**
  - **Expand** (red) cell **B**, **reach** (yellow) cell **C**

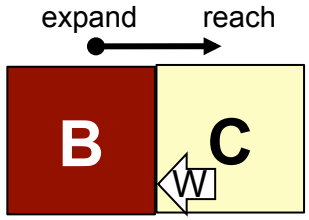




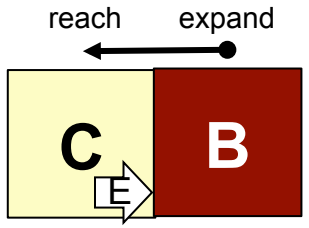
pred(C)=North



pred(C)=South



pred(C)=West

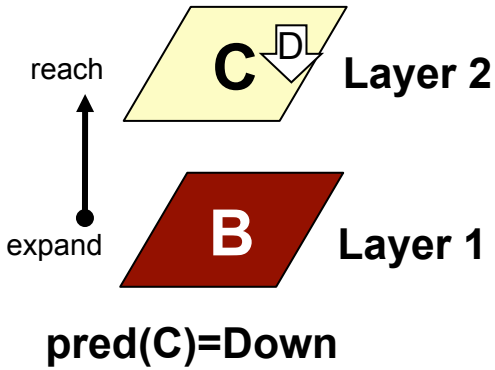


pred(C)=East

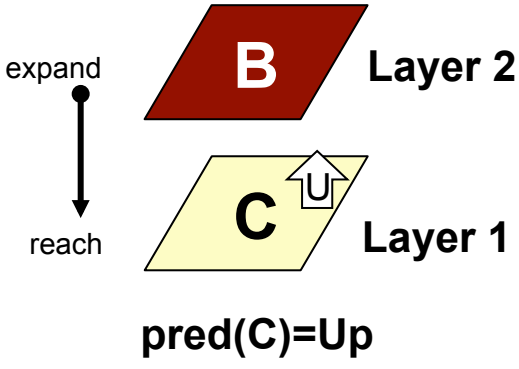
Predecessor **pred(C)** is the direction from which cell **C** was reached

Marking the Predecessor, Continued…

- **Assume 2 routing layers. 6 pred(C) tags: N, S, E, W, Up, Down**
  - **Expand** (red) cell **B**, **reach** (yellow) cell **C**



pred(C)=Down

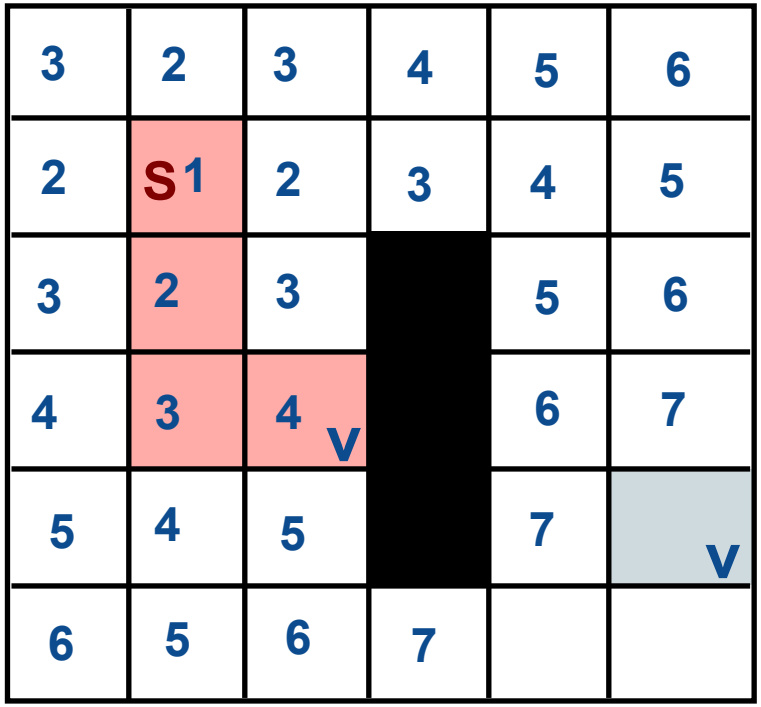


pred(C)=Up

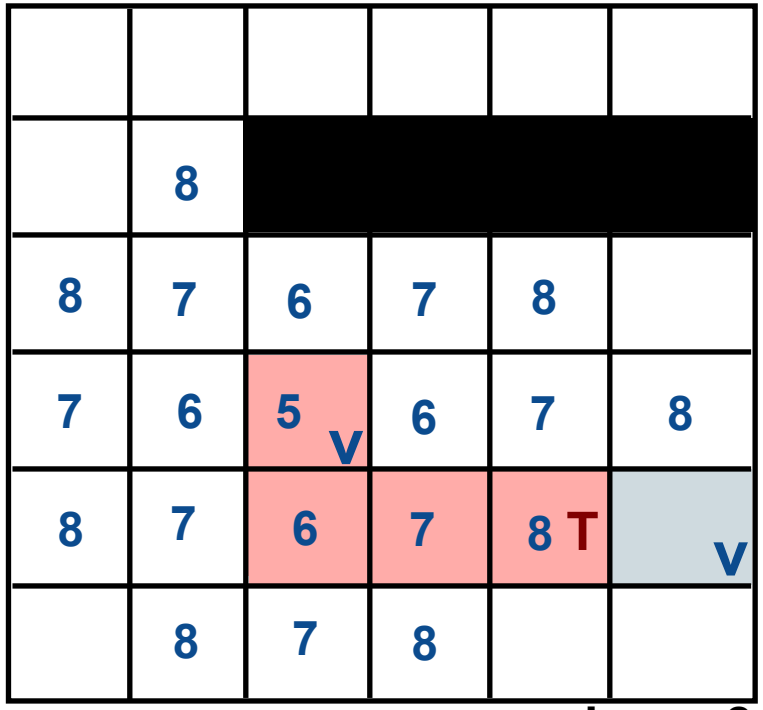
Predecessor **pred(C)** is **still** the direction from which cell **C** was reached

Why Are We Marking Predecessor?

- **Because we cannot do this anymore. No pathcosts in grid.**



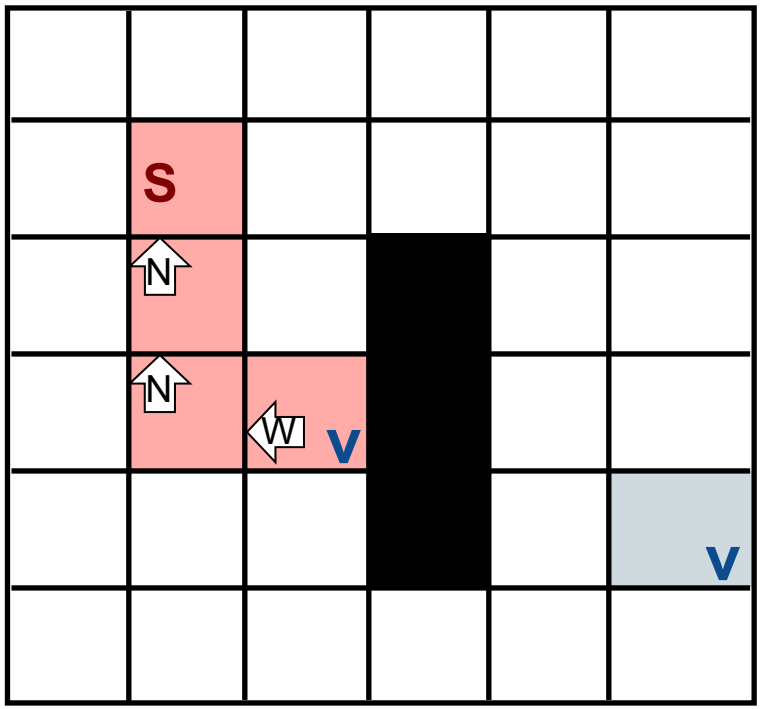
Layer 1



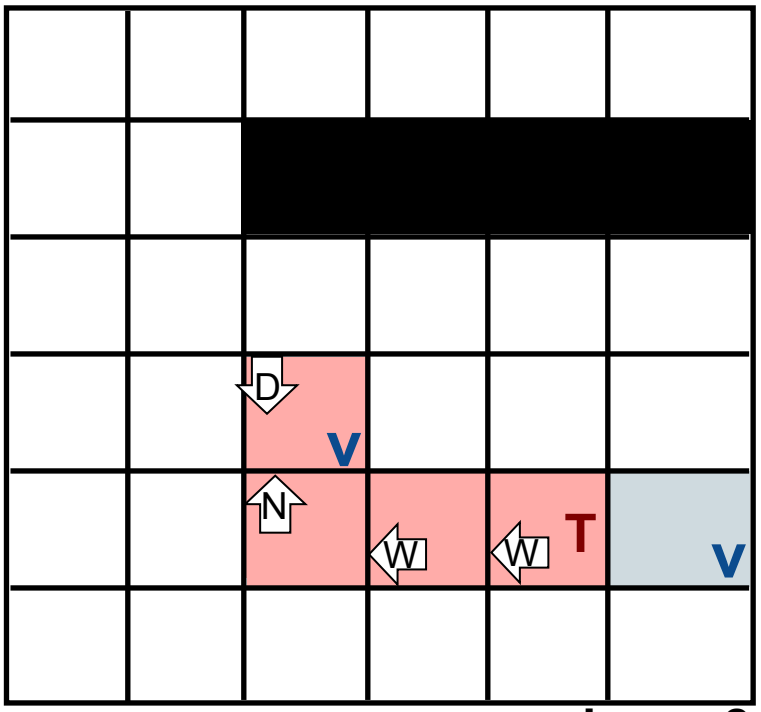
Layer 2

When Cell C Expanded, Mark Pred(C) in Grid

- **Result: Backtrace is easy! Follow pred(C) marks in grid!**

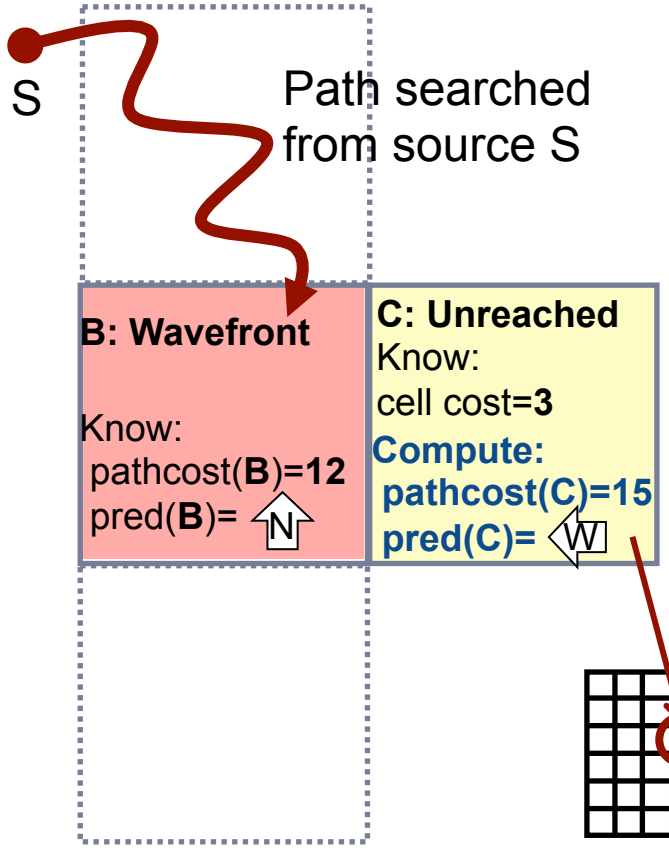


Layer 1



Layer 2

Mechanics: Expanding & Reaching



- **Expand cell B, reach cell C**
  - Grab cell **B, pathcost=12**, from wavefront
  - See unreached neighbor **C**
  - Compute cost to reach **C: 12+3=15**
  - Add this new "cell object" to the wavefront:

<layer=L, cell location=C; pathcost=15; pred=WEST>

  - Mark grid cell **C** as **Reached(C)=true**; now, we won't try to put it on the wavefront again, ie, we won't try to reach it again

Basic Maze Routing Algorithm

```
wavefront= {  source cell }
while (we have not reached target cell) {
   if ( wavefront == empty )
      quit -- no path to be found

   C = get lowest cost cell on wavefront structure
   if ( C == target )  }
      do backtrace path in grid // follow pred( ) pointers to source
      do cleanup
      return -- we found a path
   }
   foreach ( unreached neighbor N of cell C ) {
      mark N cell in grid as reached
      //compute cost to reach
      pathcost(N) = pathcost(C) + cellcost(N)
      mark N cell in grid with pred(N) direction back to cell C from N
      add this cell N to wavefront, indexed by pathcost(N)
   }
   delete cell C from wavefront
}
```

fail

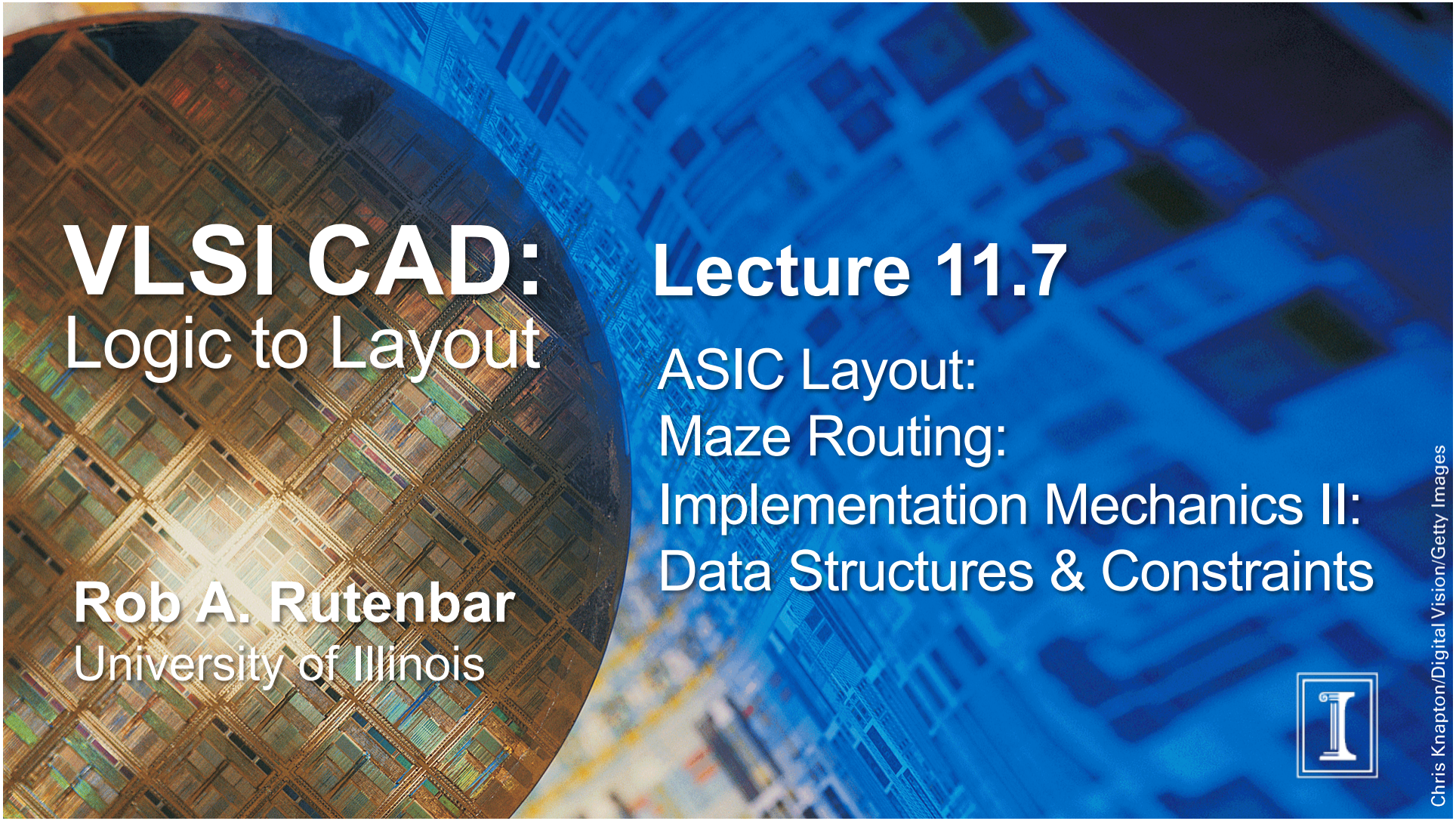
success

expand

Observations

- **Core algorithm is fairly simple**
- **Extends naturally to other features**
  - **Multi-point nets:** just mark all the cells of intermediate paths as Source, put them all in the wavefront to route to net point
  - **2 routing layers:** just use parallel grids. When you expand to reach neighbors, check UP, DOWN to other layers. Use pred(Cell) = U, D to do backtrace
  - **Non uniform costs:** just mark the cost in the grid. And for vias, whenever you reach a neighbor on a different layer, remember to add in the Via Cost.

- **What is still missing?** **Real data structure(s).** **Do this next…**



VLSI CAD: Logic to Layout

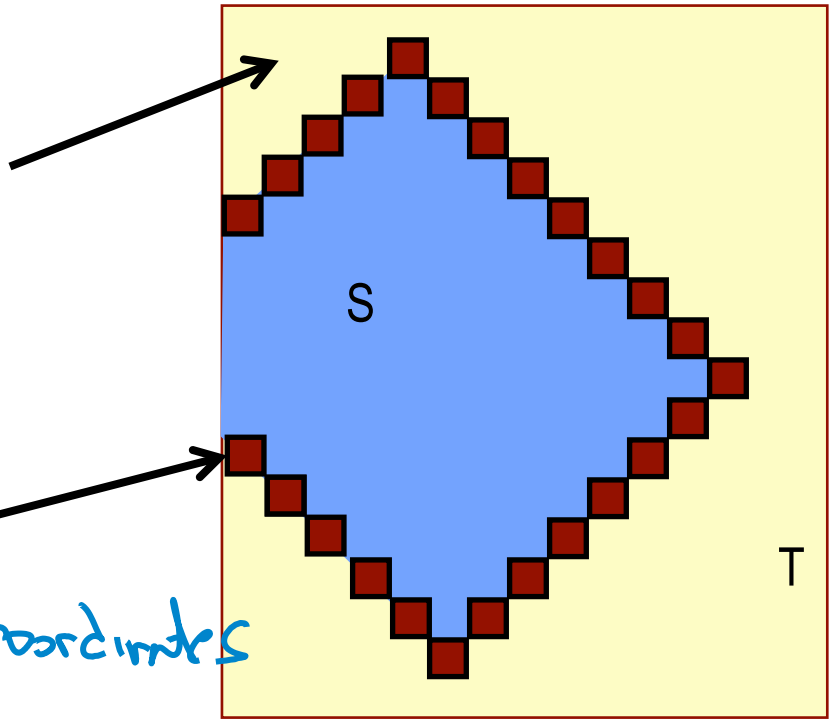
Lecture 11.7

ASIC Layout:
Maze Routing:
Implementation Mechanics II:
Data Structures & Constraints

Rob A. Rutenbar
University of Illinois

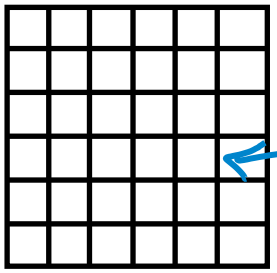
Two Key Data Structures

- **Routing grid**
  - Routing surface, holds costs of each cell, blockages
  - Mark these cells to know what cells you have already reached
  - Mark predecessor in here too
- **Wavefront**
  - Holds active cells to expand
  - Cells store pathcost, predecessor coordinates
  - Indexed on pathcost; always expand cheapest cell (pathcost) next



Data Structures

- **Routing grid**
  - **2-dimensional array per layer** is good
  - Index it by **(x,y)**



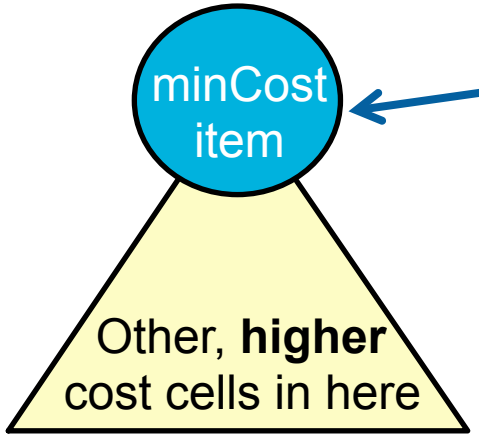
- **Each grid cell C stores:**
  - **Cost(C)** (a *small* number)
  - **Pred(C) tag** = N,S,E,W,U,D
  - **Reached(C)**, 1 bit Boolean true/false

- **Wavefront**
  - We need something clever here
  - Need fast insert/delete and sorted, cost-based indexing

- **Each cell C in wavefront stores:**
  - **Coordinates** in grid **(x, y)** of cell **C**
  - **Layer** of grid cell, ie, *which* grid is it in?
  - **Pathcost(C)** = sum of all costs up to **C**
  - **Pred(C) tag** = N,S,E,W,U,D

Wavefront Structure: *Heap*

- **Store cells of wavefront in a HEAP**
  - (Also called a ***priority queue*** -- consult your favorite data structures book)
  - Classical data structure designed for fast insertion &retrieval of lowest cost item
  - All operations (add, delete, etc.) have **O(logN)** time complexity for **N** objects.



Minimum cost item is always at the **top**.

Insert operation "bubbles" the items in heap around to ensure the cheapest item always on top.

Ditto for **delete**.

Plain Maze Routing Revisited

- **Key assumptions**
  - Always expand cheapest cell next
  - Reach each cell just once; expand each cell just once
  - Guaranteed to find the min cost path
- **New question**
  - What **constraints** must be met for this simple search strategy to get the **best path**?
  - **Said differently:** is there anything we could do **wrong** that would **break** any of the nice assumptions in the above list…?

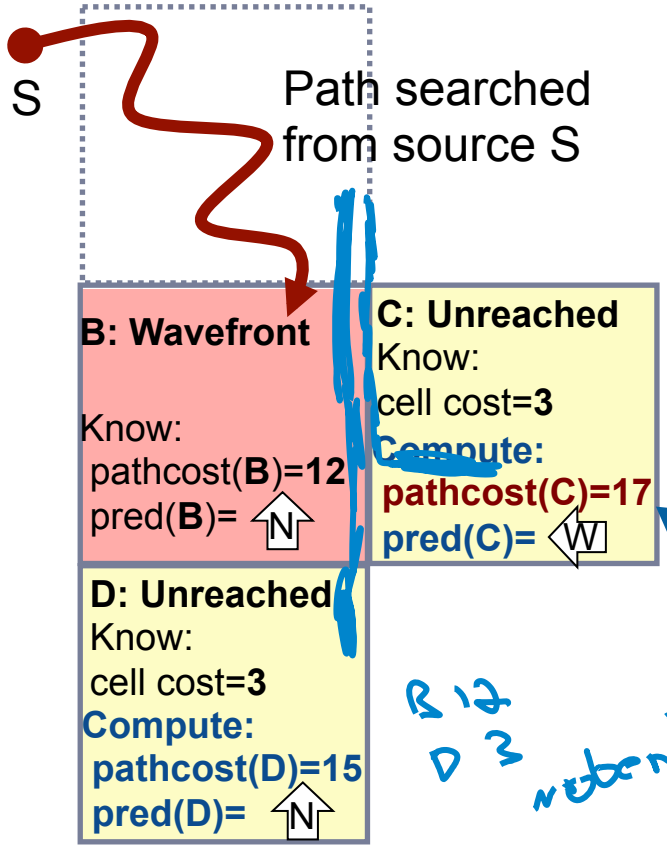
YES!

Pathcost Constraints

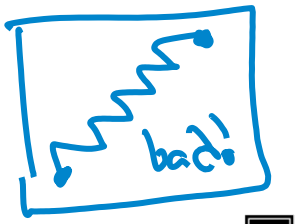
- **Basic constraint: Consistency**
  - Cost of adding a cell to a path (reaching it) must be **independent** of the path itself
  - It does ***NOT*** matter how you reached this new cell, it still adds same cost to the path
  - Guarantees we reach it once (from a cheapest path) and thus expand it just once

- **Surprise! Easy to create a cost scheme that is inconsistent**
  - …which violates all these nice properties
  - ..and which makes good, physical, geometric sense in a router!

Inconsistent Cost Function

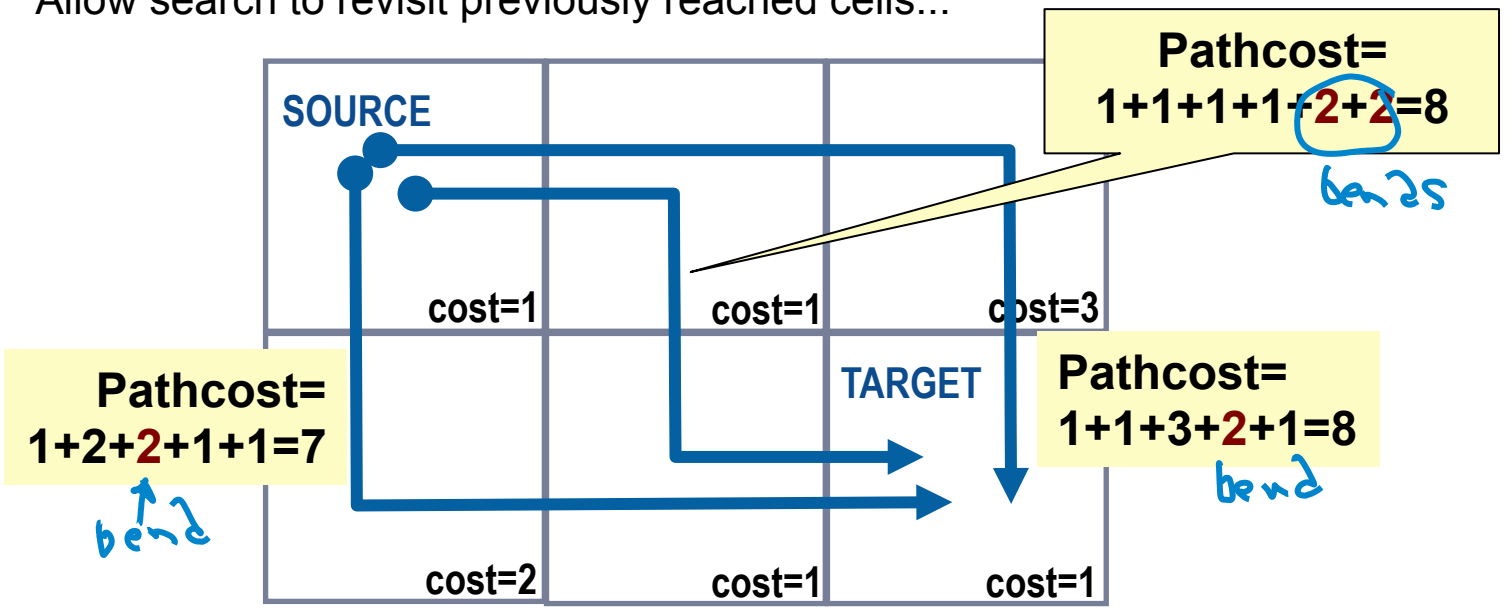


- **Expand B, reach C and D**
- **New: Penalize paths with *bends***
  - Now add **another** cost when you reach a cell that requires a turn (a bend) from the direction that reached the expanding cell
  - **Bend:** **pred(reached) ≠ pred(expanded)**
    - **pred(C) ≠ pred(B)** → Add bend penalty
    - **pred(D) = pred(B)** → No bend penalty
- Suppose **Bend penalty = 2**

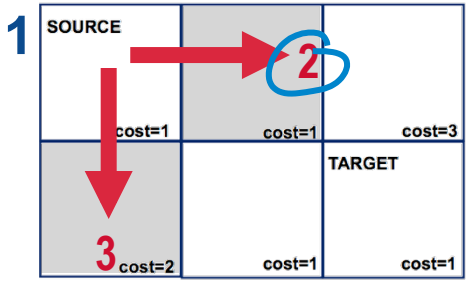


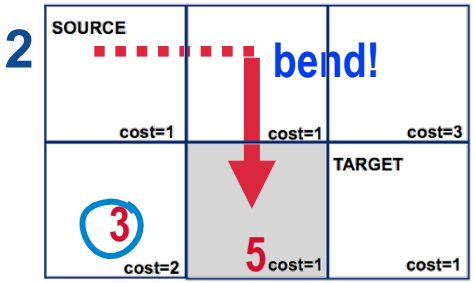
Inconsistent Cost Function

- **Try this example with bend penalty = 2**
  - Don't mark the "reached" bit in each grid cell when you first touch (reach) the cell
  - Allow search to revisit previously reached cells...



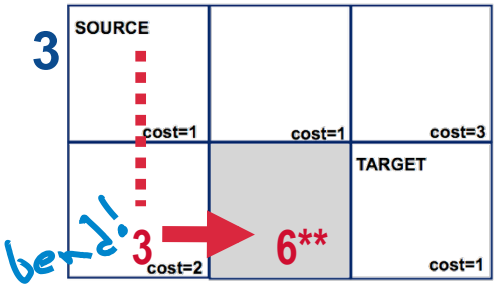
Inconsistent Cost Example

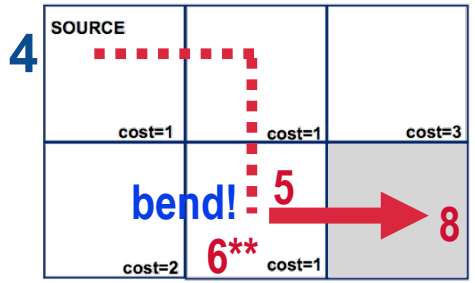




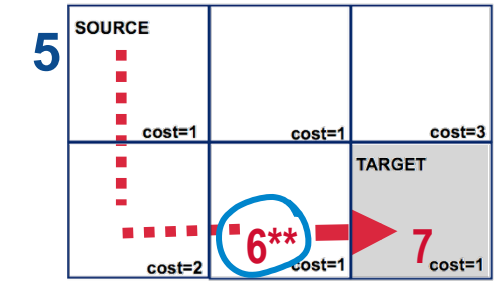
2+4=2 bend

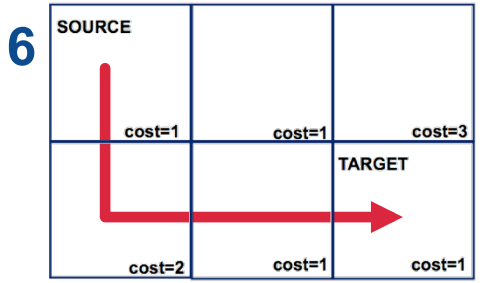
Revisit this cell, reach it again at more cost





Revisit target, but it's cheaper now



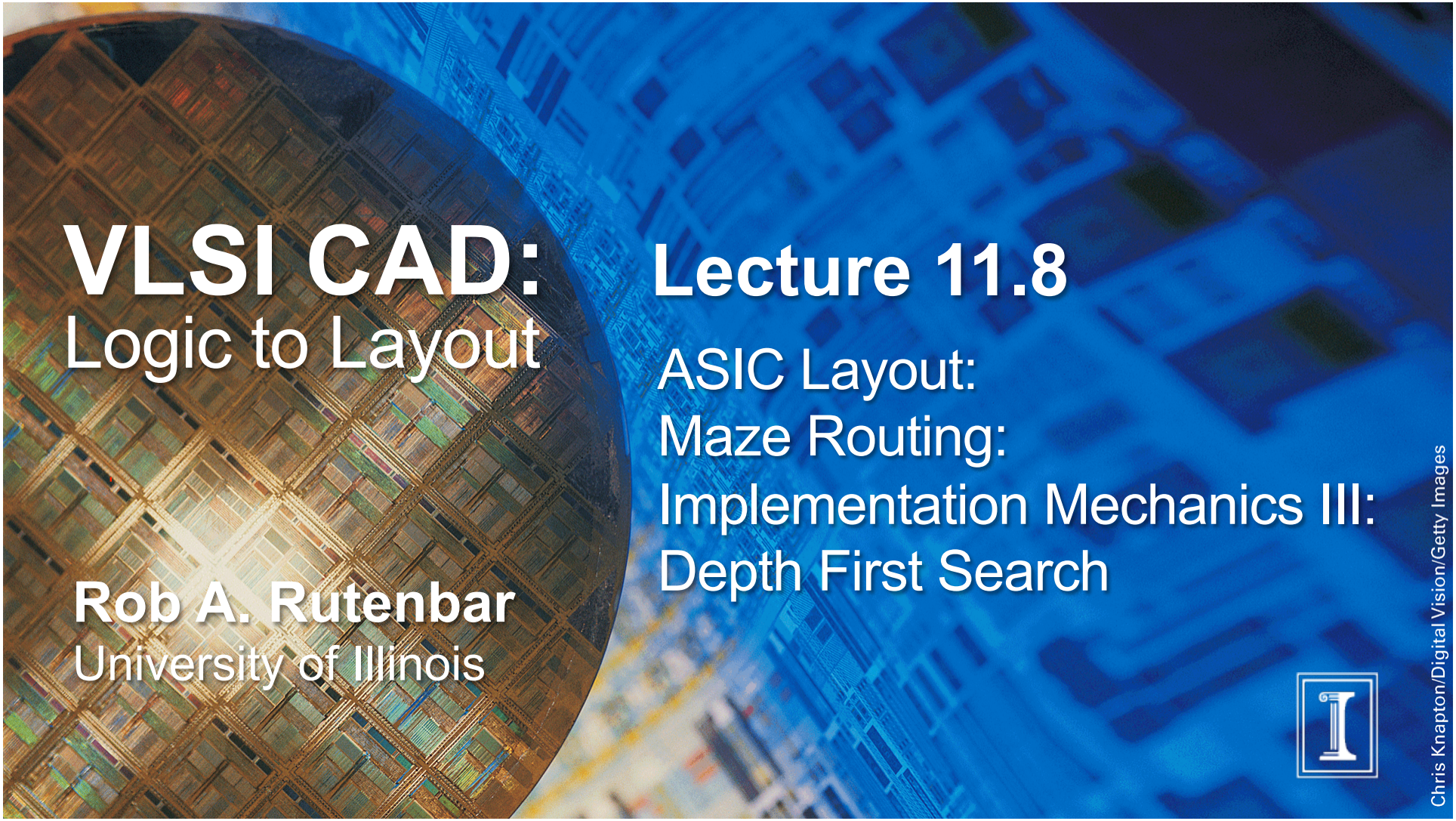


Inconsistent Cost Function: Implications

- **Notice what happened**
  - Reached same cell, **later**, at a **higher** cost, but it was the **cheaper** source-to-target path
- **Implications**
  - You will reach cells **multiple times** at **different costs**
  - You will have **same cell** in wavefront **multiple times** at **different costs**
  - Can still expand cheapest first, but **cannot quit when you reach target!**
- **Termination of search?**
  - Cannot quit until every cell in wavefront has a cost so big that it is **impossible** to reach target any cheaper than current cheapest path
  - May reach, expand **lot more cells** with an inconsistent cost function...
  - ..but you can do a lot of cool things with such functions

Inconsistent Cost Function: Practicalities

- **Do people really do this? In real routers?**
  - Oh, yes. Every real industrial router does something like this!
- **How do we know when to quit search?**
  - Usually, some heuristic
  - If you expanded **M** cells to hit the target, pick a number, like **M' = (0.1) * M**, and commit to do not more than **M'** additional expands. Take best path you find
  - It does add a lot of complexity to the core implementation, but it helps quality
- **For us – good enough just to know this can happen**
  - We **don't** expect you to implement **any** of this in your own Program Assignment(s)



VLSI CAD: Logic to Layout

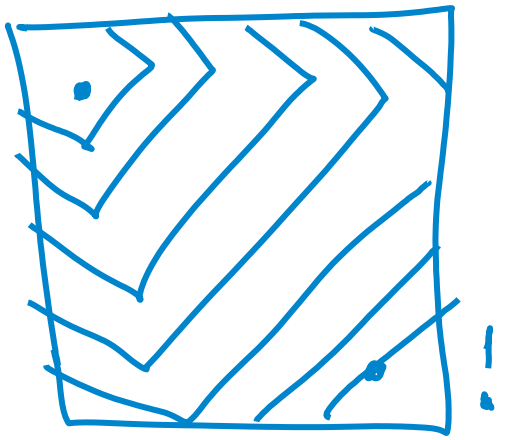
Lecture 11.8

ASIC Layout:
Maze Routing:
Implementation Mechanics III:
Depth First Search

Rob A. Rutenbar
University of Illinois

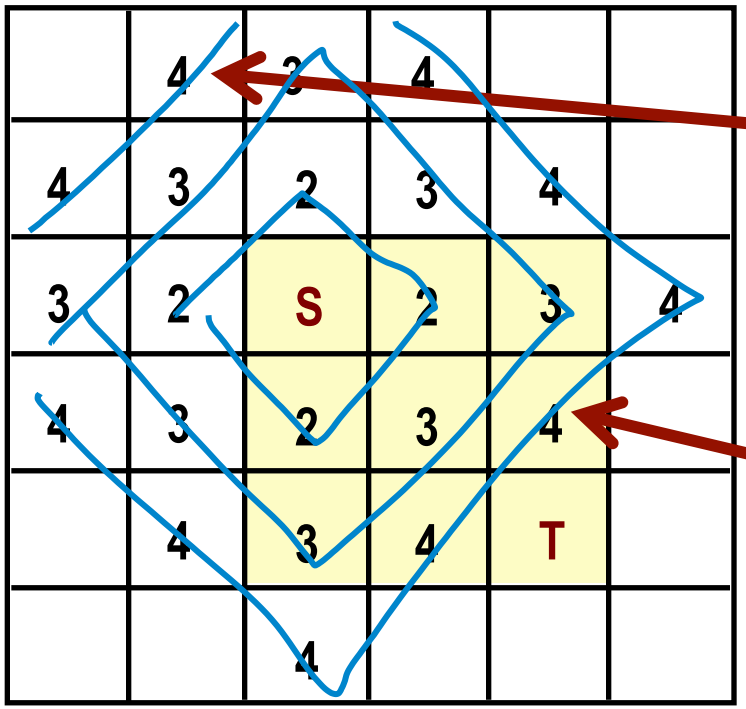
Expansion Process, Revisited

- **Problem:**
  - Expand **lots** of cells to find one path to the target
  - CPU time is proportional to number of cells you expand
  - **No attempt to search in direction of target first**



- **Questions:**
  - Can we actually bias expansion so we search **toward** the target?
  - Can we do this and still keep **guarantees** of reaching target with minimum cost path?

Motivation for Smart Search



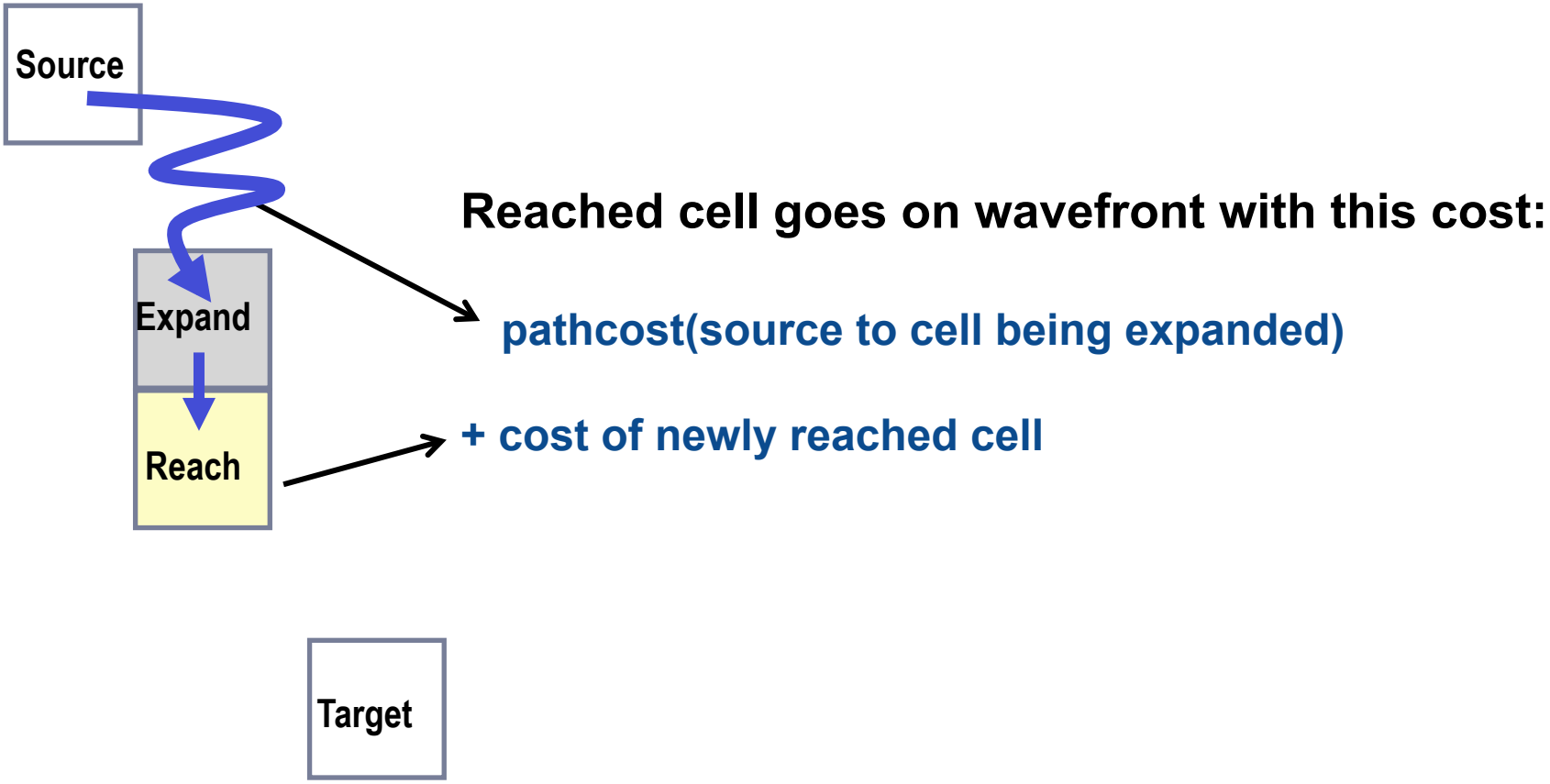
Expanding **away** from the target seems to be a waste of time

Searching **toward** the target in the shaded region, sometimes called the "source-target box" seems a good idea to try first

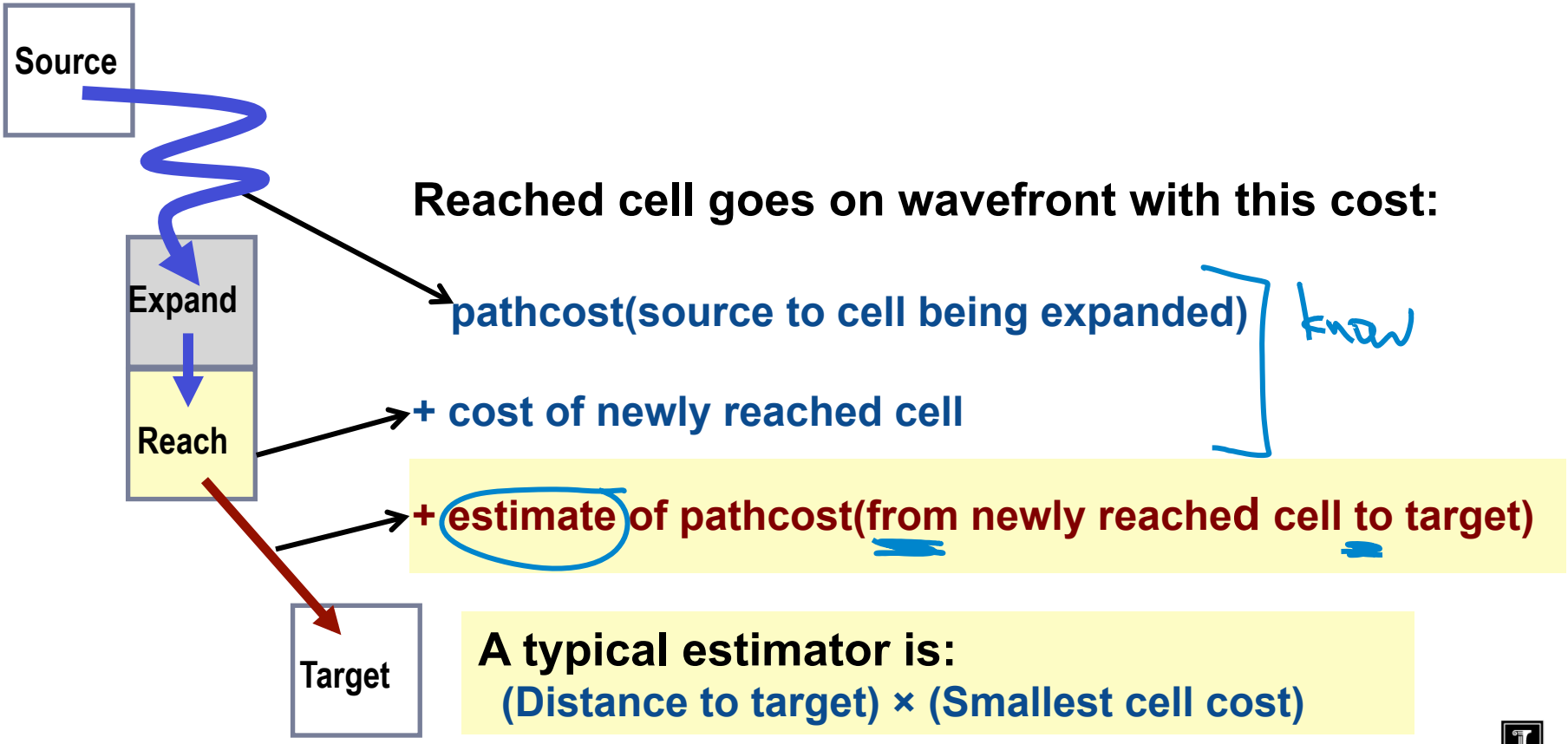
Smarter Search: Maze Search With Predictor

- **Two parts:**
  - Add **predictor function** to the cost, to direct the search toward the target
- **Plain maze router**
  - Add a cell **C** to wavefront with cost that measures **cost of partial path**, **source-to-cell(C)**
- **Smarter maze router**
  - Add cell **C** to wavefront with cost that **estimates *entire* source-to-target cost of path**
  - **Trick:** estimate this as **pathcost(source to cell C)** + **predictor(cell C to target)**
  - Today, we recognize this as difference between smart depth first search (**DFS**) with a predictor, versus classical breadth first search (BFS) as per Dyjkstra
  - This is famous application of a classical idea: **A* search**

Plain Maze Routing



Add A Depth-First Predictor



A typical estimator is:
(Distance to target) × (Smallest cell cost)

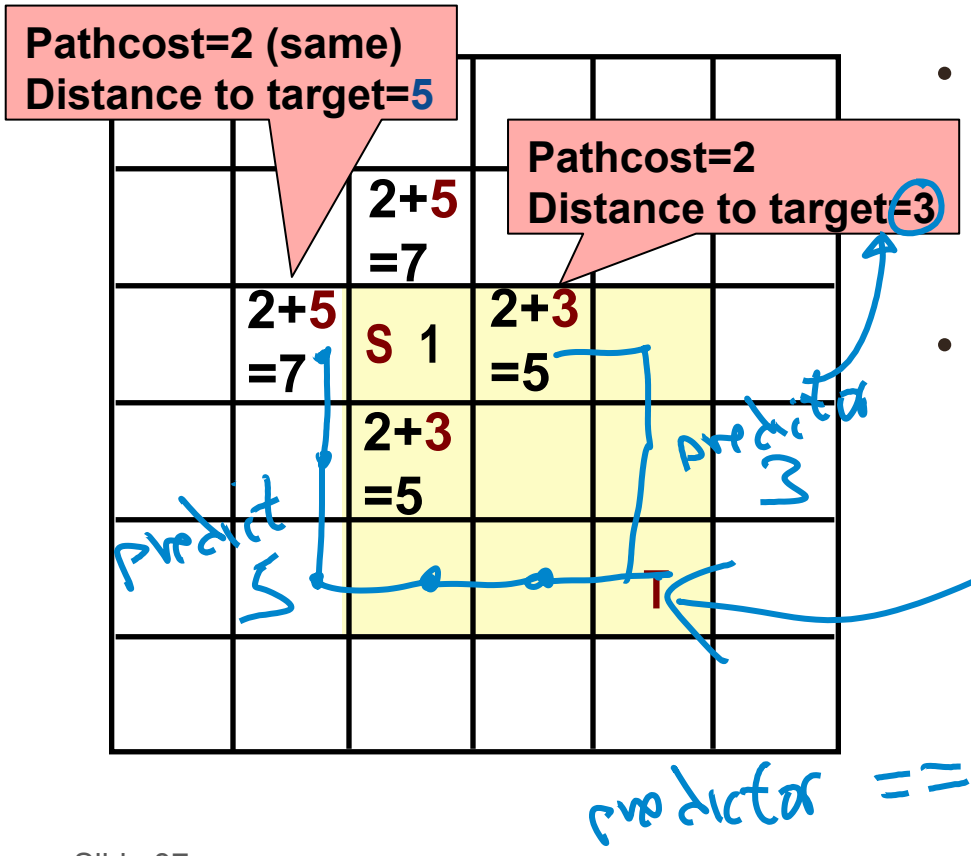
Technical Results

- **Depth first predictor**
  - If predictor is a **lower bound on** (ie, less than) extra pathcost you add to reach target...
  - ...you will still get the minimum cost path, **guaranteed**

- **What does it do?**
  - It **alters the order** in which we expand cells
  - It prefers to expand cells that are **closer to the target first**

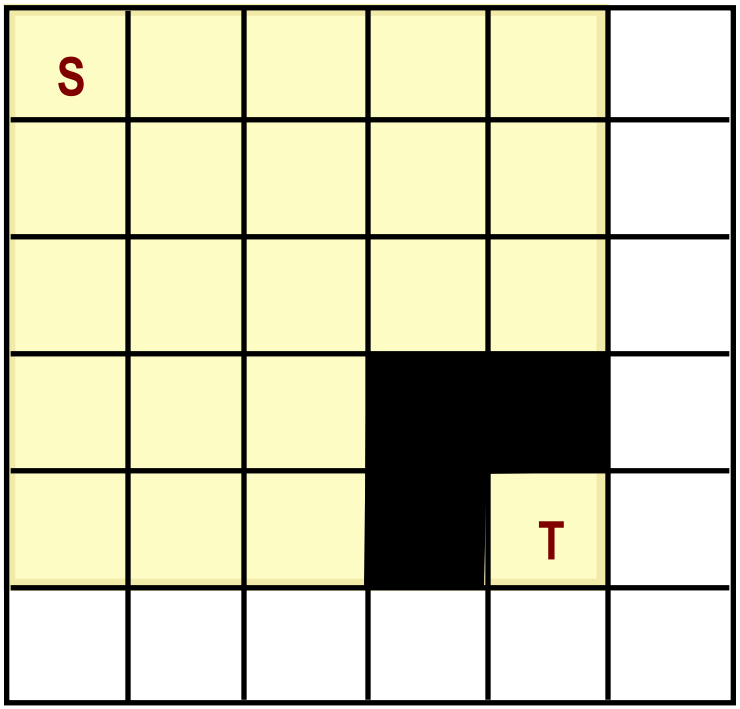
- **Look at an example...**

Depth-First Expansion Example



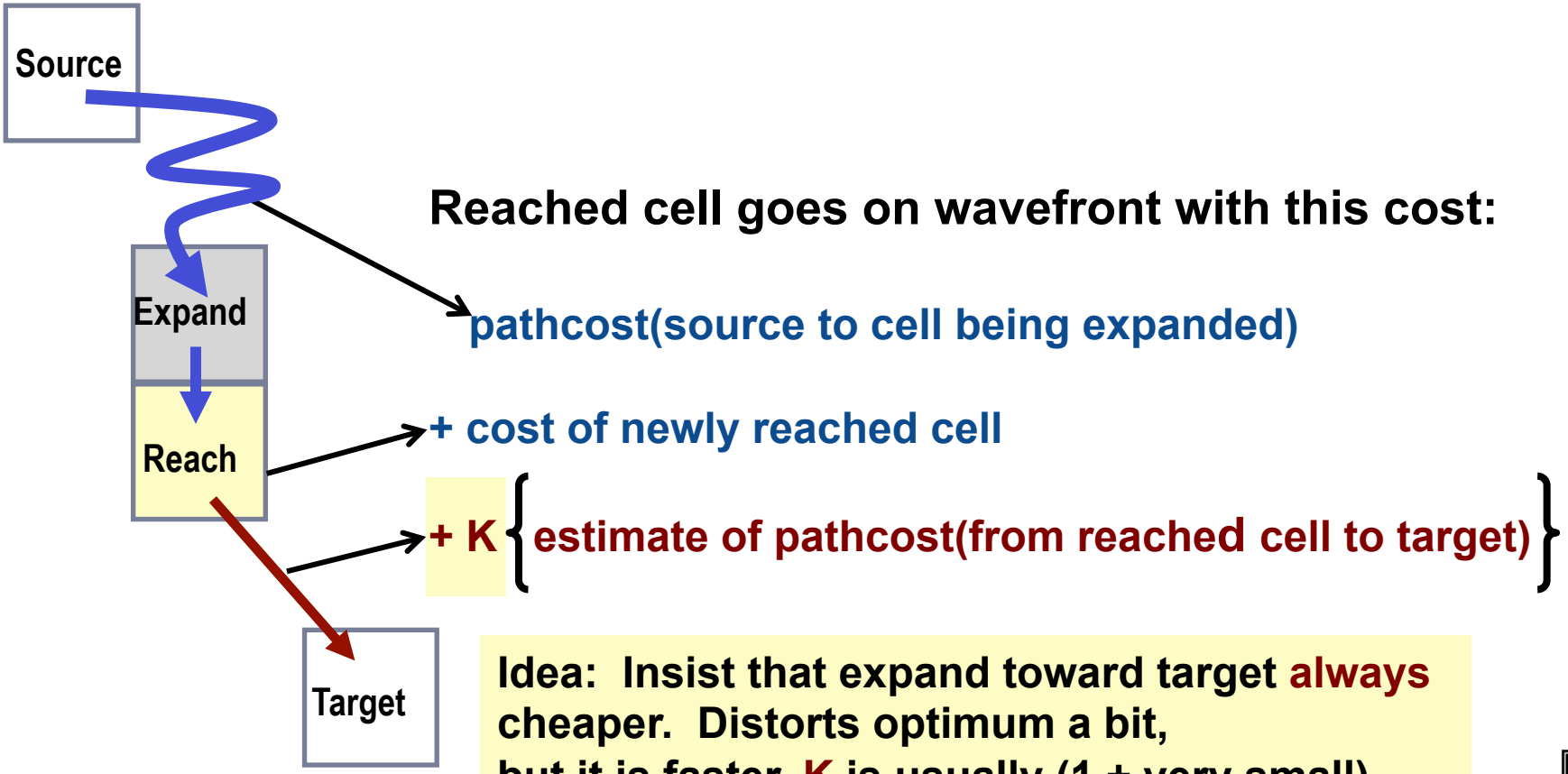
- **Observe**
  - Search prefers to stay inside the bounding box of the source-target rectangle before it expands other cells.
- **Why?**
  - Turns out, all the cells in this box now cost the same!
  - **pathcost(source→cell) + pathcost(cell→target)** = constant in box
  - So – it expands **toward** the target!

DFS Expansion: Unintended Consequences



- **This is "the" example of where "stay in the box" is inefficient**
  - The target is blocked inside the source to target rectangle.
  - Problem: DFS explores the *whole* rectangle before it tries anyplace else.
  - Maybe faster to search outside if this rectangle is *very* big...?
  - Maybe it's OK we don't get the lowest-cost path, just a *decent* path, quick?

Another Heuristic…



Idea: Insist that expand toward target always cheaper. Distorts optimum a bit, but it is faster. K is usually (1 + very small)

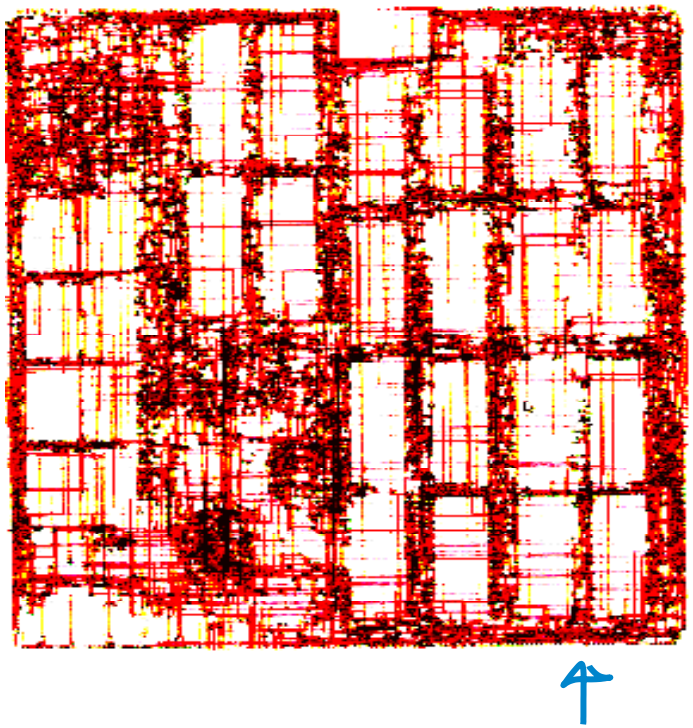
VLSI CAD: Logic to Layout

Lecture 11.9

ASIC Layout:
Maze Routing:
From Detailed Routing
to Global Routing

Rob A. Rutenbar
University of Illinois

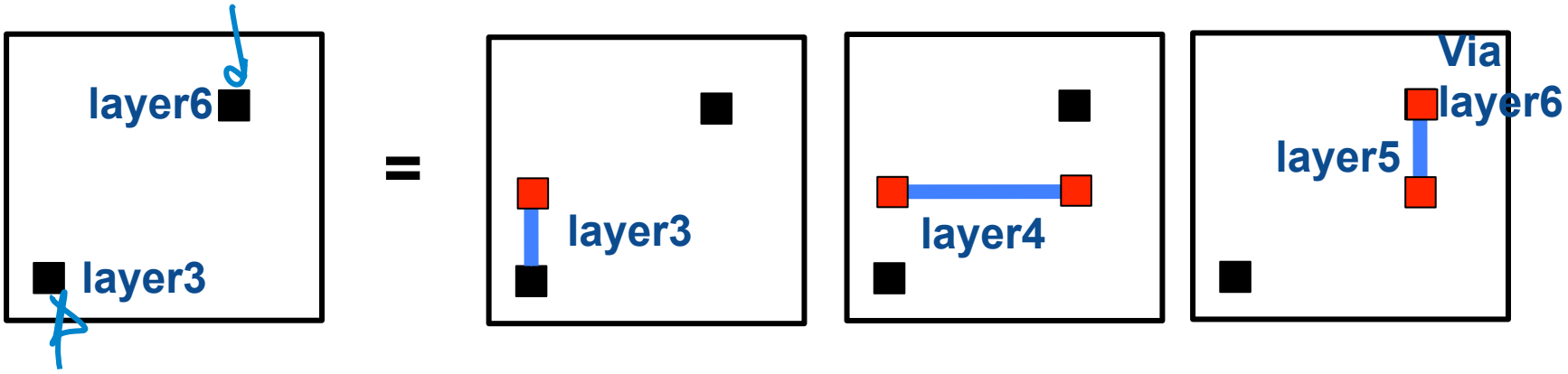
Reality Check: ASIC Scale & Complexity



- **Big chip, 1cm x 1cm**
- **20-50 million nets**
- **Modern IC technology, ~100nm pitch for wires (grid size)**
- **So, 100K x 100K routing grid!**
- **x10 routing layers?**
- **100 billion grid cells!**
- **Do we really do it like this?**

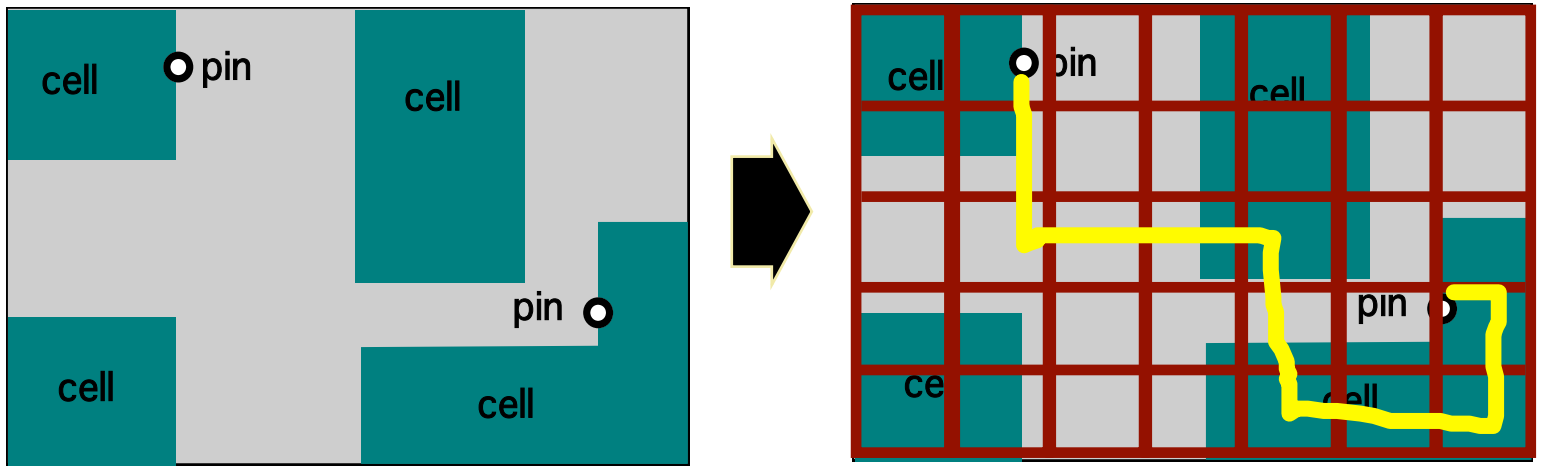
First Fact: Preferred Routing Directions

- **Every other layer of metal wiring has a preferred direction**
  - Metal 3 5 7 9 are **vertical**. Metal 4 6 8 10 are **horizontal**. Mostly, preferred.
  - **Idea**: all the wires on these layers look like (mostly) straight lines
  - If you need to bend: use a **via** (or vias). Makes it easier to embed lots of wires.

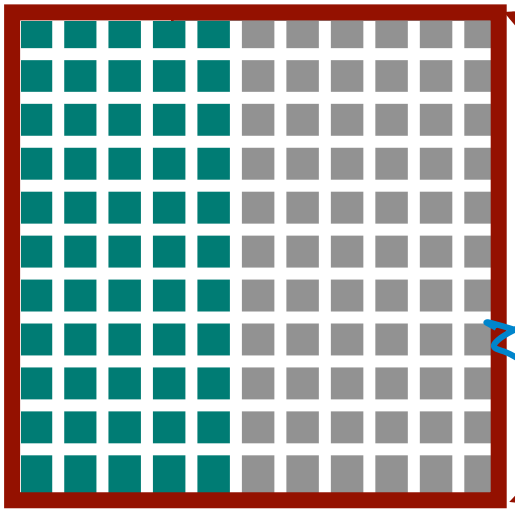


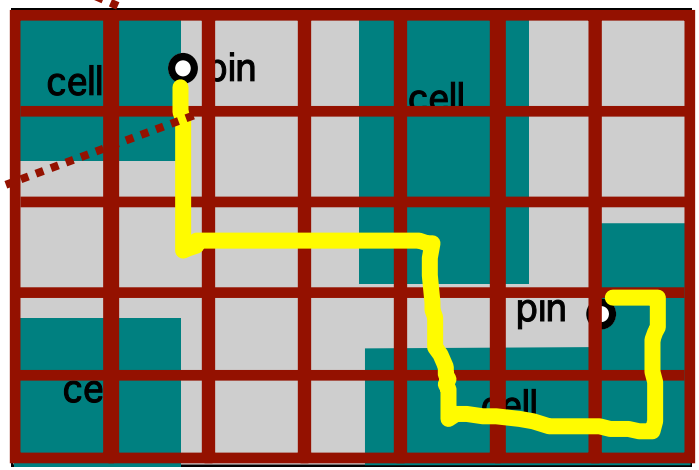
Divide & Conquer: Global Routing

- **How to deal with huge scale of chips? Global Routing**
  - Global routing again imposes a grid on surface of the chip
  - But now, this grid is much coarser, eg, each cell ~ 200 wires X 200 wires in size
- **Big Idea: Maze route thru these big, coarse regions**



Global vs Detailed Routing: Geometry

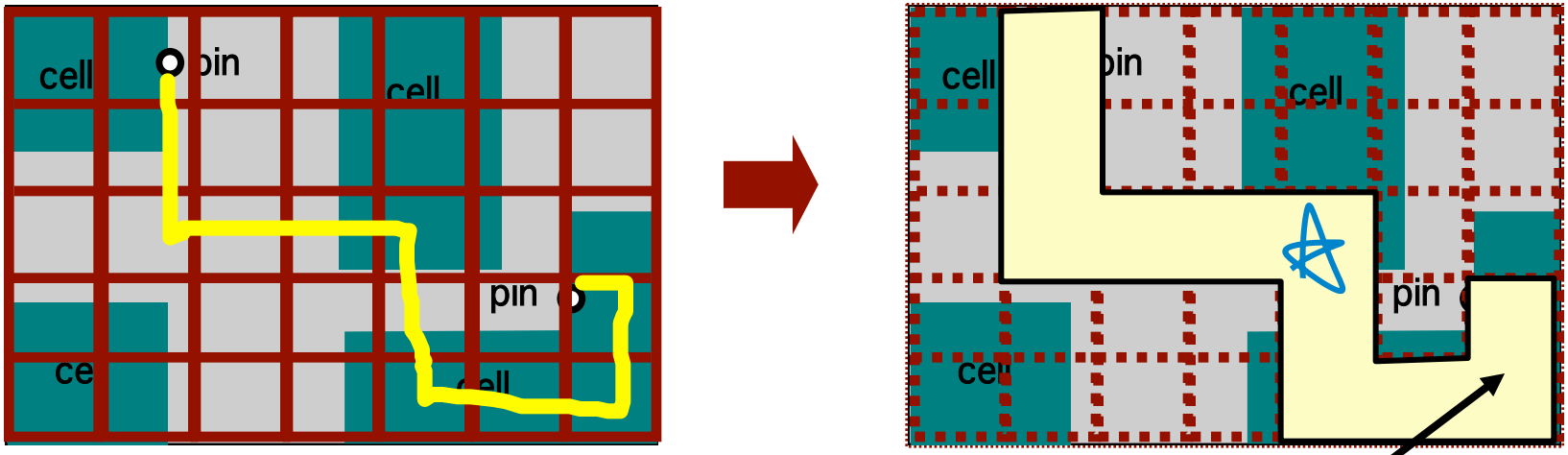




All the routing we have done so far is called **Detailed Routing**.
Means we plan exact, final path of wire.
Each cell in **Global Routing** grid is a box (called a **GBOX**) with size typically 100-200 wire grids on a side.
(This example just has ~10 for clarity)

What Global Routing Does

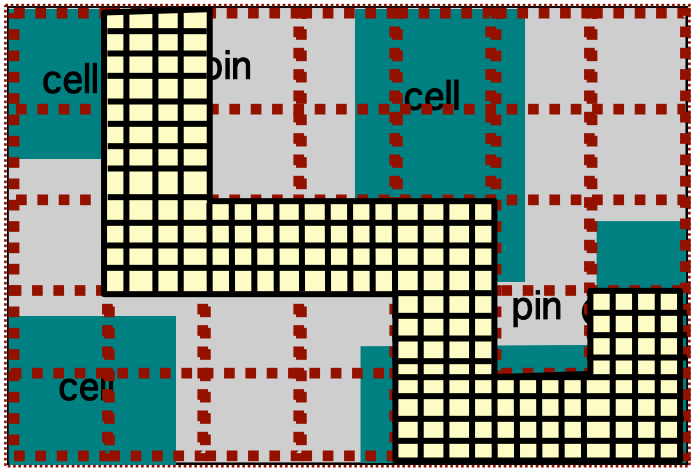
- **Makes sure overall wiring congestion is reasonable**
  - Balance **supply** (how much available space) vs **demand** (how many paths *want* to go here)
  - Global routing generates **regions of confinement** (ie, coarse path) for a wire



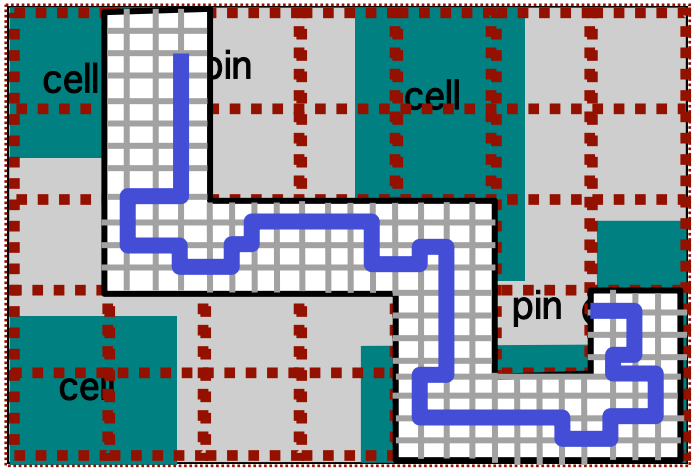
Global routing tells us we want this net to use this rough path; but not exact path inside this region

What Global Routing Does

- **Detailed routing embeds *exact* paths in these regions**
  - Simplest model is **grid**: require wires, pins to use tracks **on** this grid



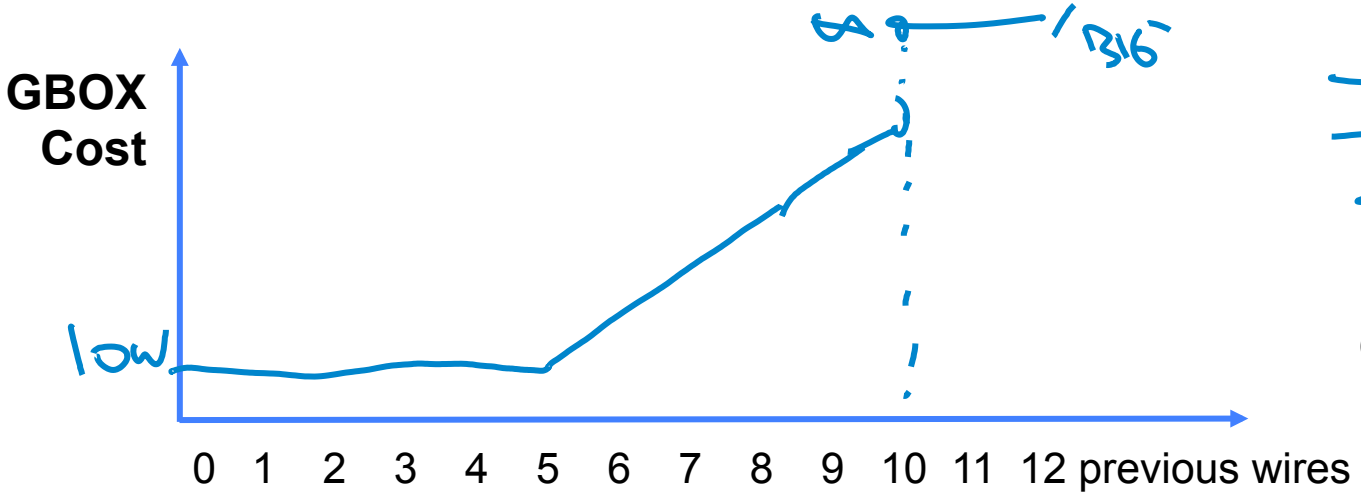
Global router tells us to search for detailed paths only here

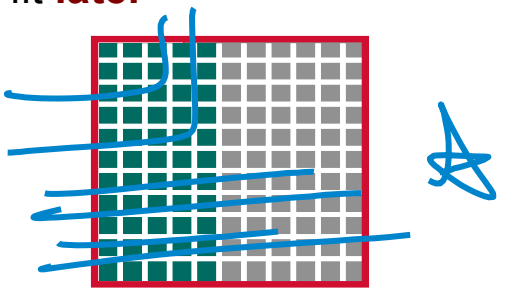


Detailed router tells us exact final path in this region

Global Routing as Maze Routing

- **What's different for a global router?** **Cell Cost Function**
  - **Detailed router:** grid cells are blocked (cost=infinity) or they have a cost proportional to how much we want to avoid them
  - **Global router:** grid cells have ***dynamic*** cost, proportional to how many ***previously*** routed wires used this cell, how many more can fit **later**





Suppose global GBOX is ~ 10 tracks for wires on each side

Maze Routing: Summary

- **Been around a *long* time**
  - Very flexible cost-based search; can be recast to attack many problems
  - **Dominates** most modern routers…
    - Some of the big chip-level routers use fancy dynamic grids
    - Some use rather sophisticated ”grid-less” representation of “space”
- **Lots of of ancillary problems (and solutions)**
  - Still routes one net at a time: early nets block later nets
  - Lots of iterative improvement strategies here (I didn’t talk about)
  - Lots of hierarchy/abstraction (I did talk about Global Routing a bit)