VE527

Computer-Aided Design of Integrated Circuits

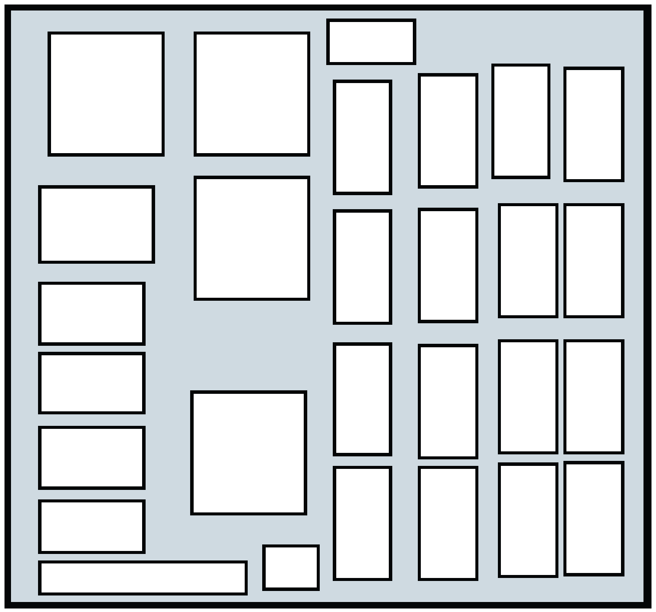
Routing Basics and Maze Routing

Outline

- Routing Basics

- Maze Routing
  - 2-Point Nets in 1 Layer
  - Multi-Point Nets
  - Multi-Layer Routing
  - Non-Uniform Grid Costs

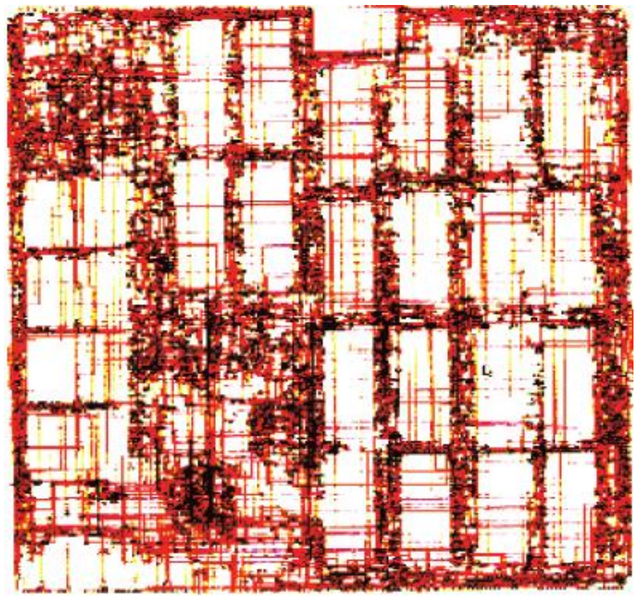
Routing: The Problem



Thousands of macro blocks

Millions of gates

Millions of wires



Kilometers of wire

Basic Routing Challenges

- **Scale**: Big chips have an enormous number (**millions**) of wires.
  - So many. Turns out **NOT** every wire gets to take an "**easy**" path to connect its pins.
  - Must connect them all. However, can't afford to route many wires manually.

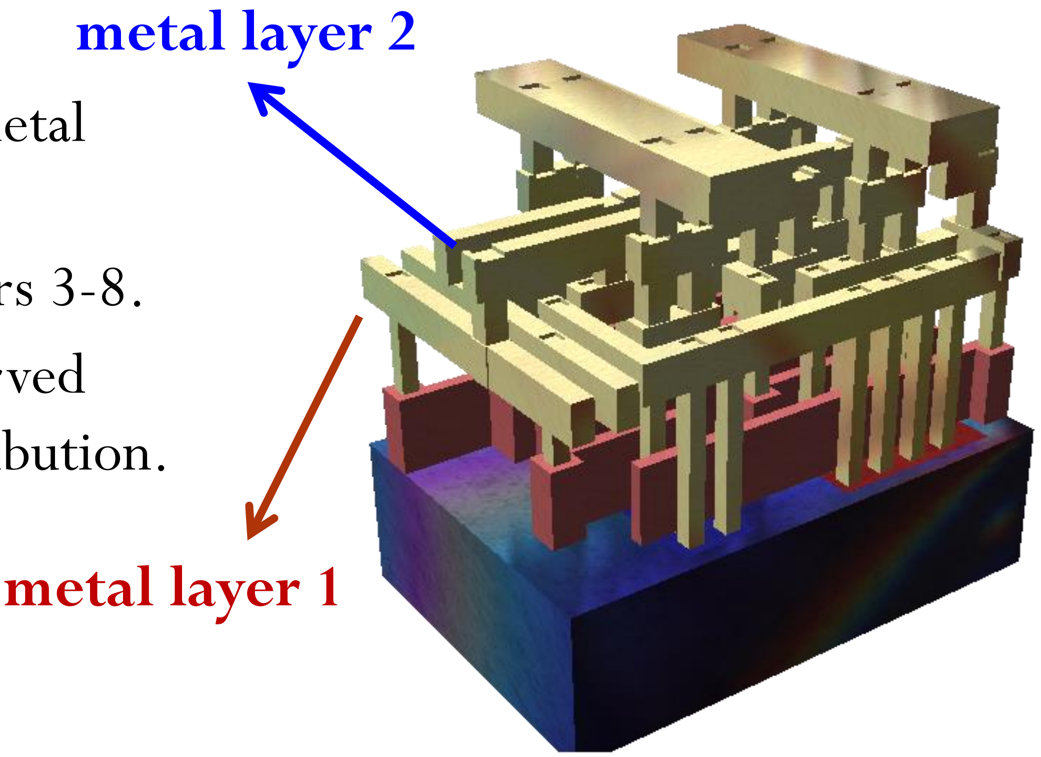
- **Geometric complexity**: At nanoscale, **geometry rules** (e.g., the distance between two adjacent wires) are complex – makes routing hard.

Basic Routing Challenges (cont.)

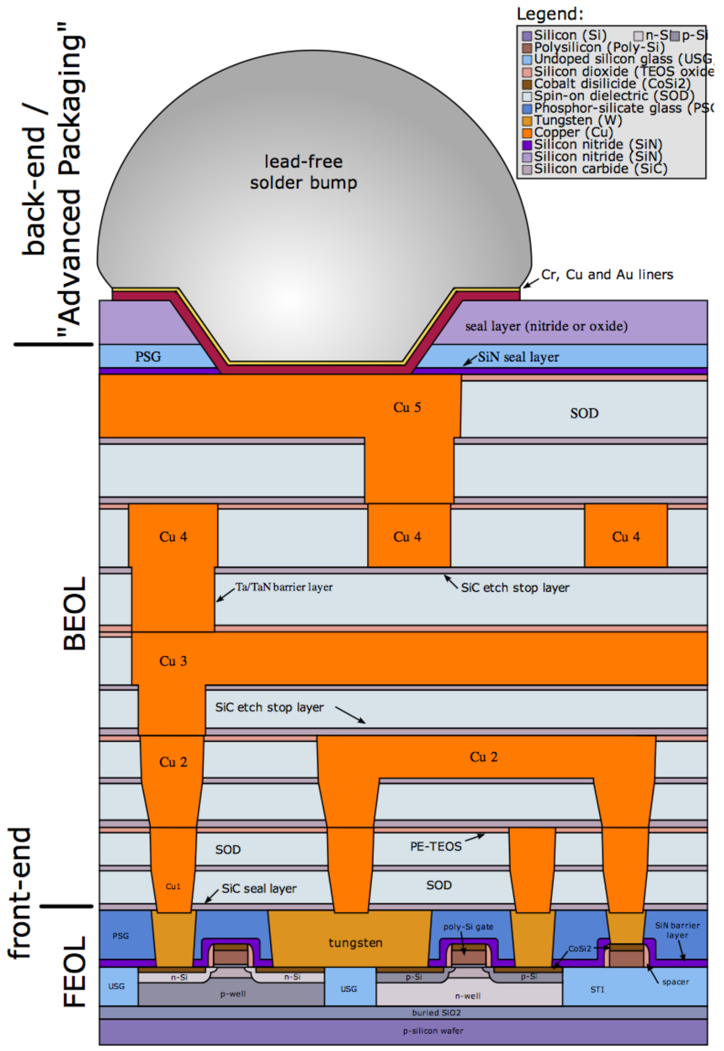
- **Electrical complexity**
  - It's **not enough** to make sure you connect all the wires.
  - You also must ensure that the **delays** through the wires are not too big.
  - And that wire-to-wire **interactions** (crosstalk) don't mess up behavior.

Physical Assumptions

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

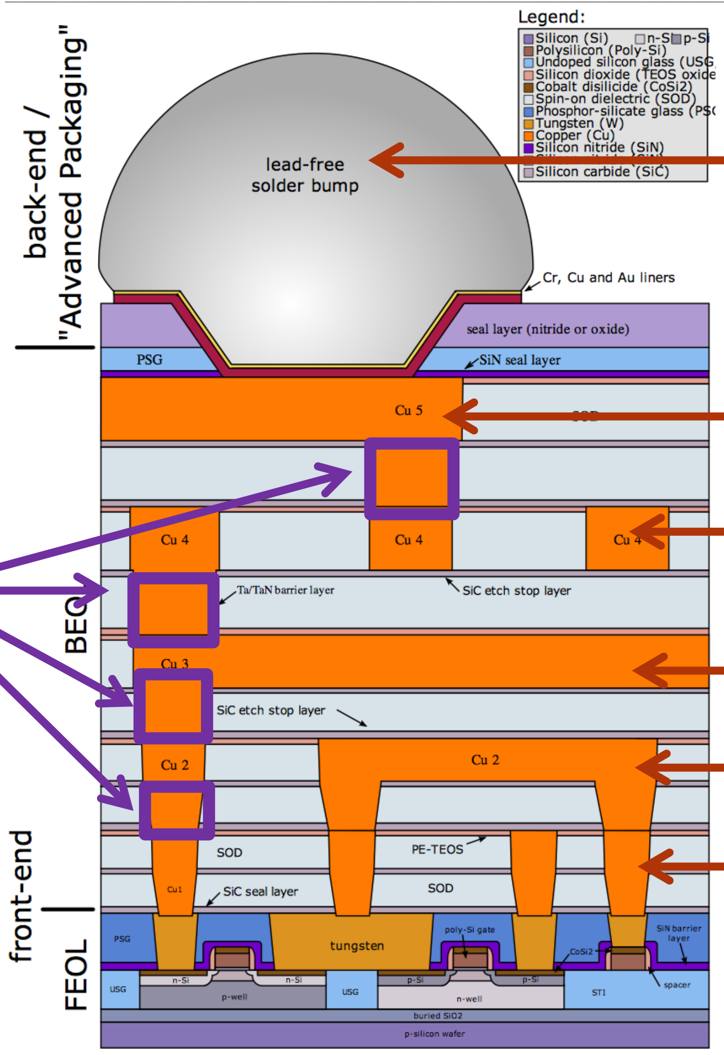


Typical Example (5 Layers)



- Cross sectional view of a chip with 5 layers of wiring, along with packaging connection ("bump").

Typical Example (5 Layers)



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

metal layer 5 Thicker layer (less resistance) for clock, power

metal layer 4

metal layer 3 } For connecting gates

metal layer 2

metal layer 1 } Wiring for cells

} Transistors to make gate-level logic

Vias, connect Metal layers k and k+1

Algorithm Comparison: Placement vs Routing

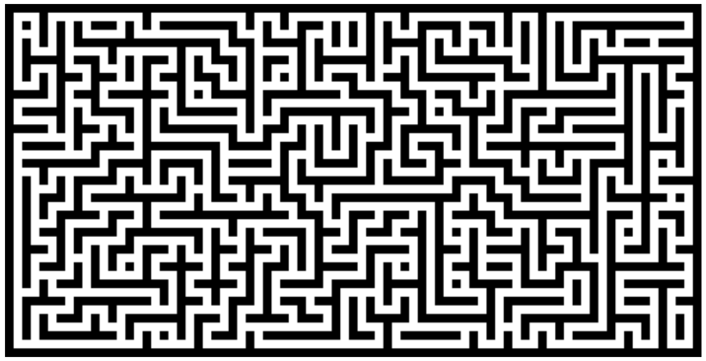
- There are lots of different placement algorithms.
  - Iterative methods, such as simulated annealing-based method.
  - Analytical methods based on solving/optimizing large systems of equations.

- There are **not quite so many** routing algorithms.
  - There are several 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”

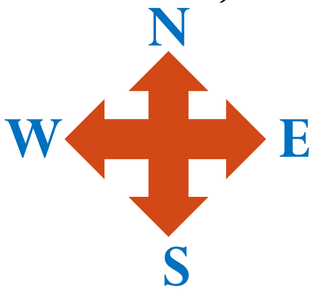
- Given a maze, find a shortest path from entrance to exit.

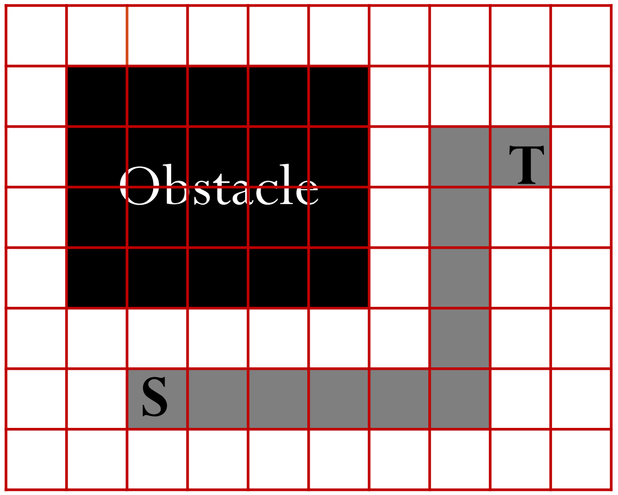
- From one famous early paper:
  - E.F. Moore. “The shortest path through a maze”, in Proceedings of the International Symposium on the Theory of Switching, pp 285--292, Cambridge, MA, Apr. 1959.



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

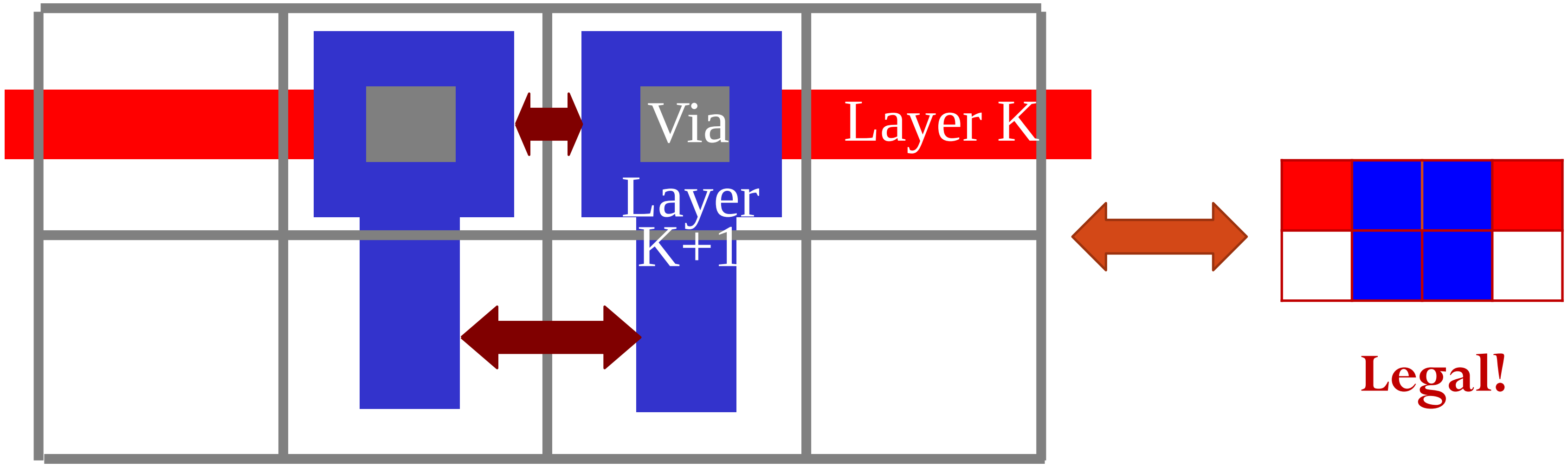
- Make a big geometric assumption: **Gridded routing**.
  - The layout surface is a **grid of regular squares**.
  - We can mark **obstacles** (also called **blockages**) where we are not allowed to route.
  - A legal wire path = **a set of connected grid cells**, through **unblocked** cells in grid.
- Wires are also strictly **horizontal and vertical**.
  - No diagonal (e.g., 45°) angles. A path goes east/west, or north/south, in this grid.





Grid Assumption

- Wires and their vias fit in the grid.
- All pins we want to connect are at the **center** of grid cell.
- Grid assumption is a critical assumption – implies many real geometrical constraints on wires (designers use these constraints!).
  - All wires are roughly the **same** size (width).
  - If we choose the grid size proper, then there are no geometry rule (e.g., spacing) violations.



Maze Routers: Topics

- Router functionality
  - Two-point nets in one layer with unit cost
  - Multi-point nets
  - Multiple layers
  - Non-uniform grid costs
- Implementation mechanics
  - Expansion (a key step in routing)
  - Data structures
  - Depth-first search
- Divide & Conquer: Global routing

Outline

- Routing Basics

- Maze Routing
    - 2-Point Nets in 1 Layer
    - Multi-Point Nets
    - Multi-Layer Routing
    - Non-Uniform Grid Costs

Maze Routing: History

- 1959 – Basic Idea
  - E.F. Moore. "The shortest path through a maze," In Proceedings of the International Symposium on the Theory of Switching, Apr. 1959, Harvard University Press.

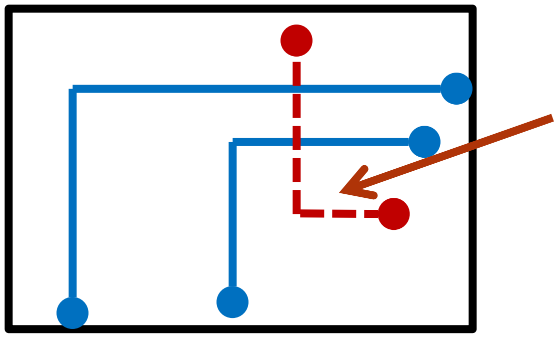
- 1961 – Applied to electronics (board wiring)
  - C.Y. Lee, "An algorithm for path connections and its applications", IRE Trans. on Electronic Computers, pp. 346-365, Sept. 1961.
  - Lee of Bell Labs invents the algorithm; gets famous for "Lee routers".

Maze Routing: History

- 1974 – Apply ideas from AI to improve the routing speed
  - F. Rubin, "The Lee path connection algorithm", IEEE Trans. on Computers, vol. c-23, no. 9, pp. 907-914, Sept. 1974.

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.
  - We are just going to ignore this one for the moment…

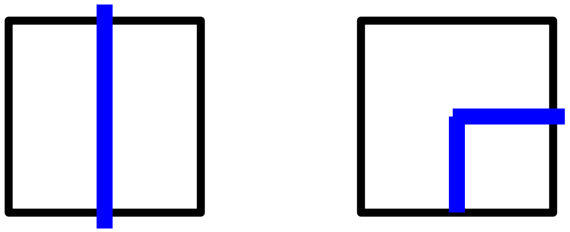


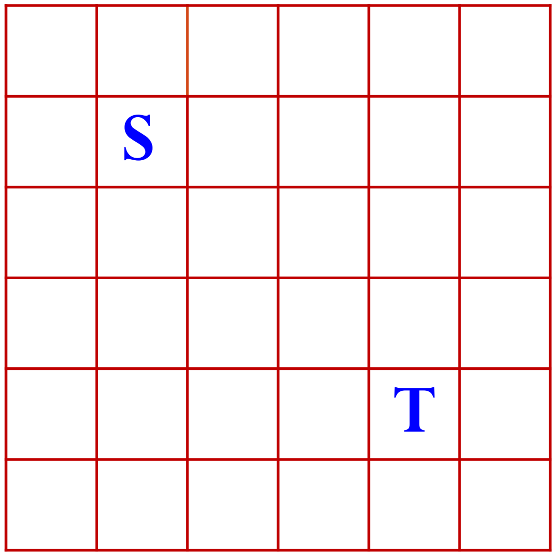
Blocked!
Impossible to route

Maze Router: Basic Idea for 2-Pin Nets

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

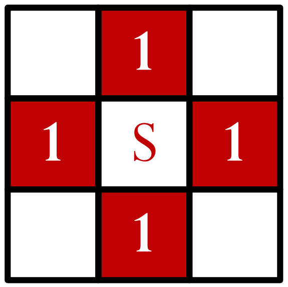
cross or **bend**



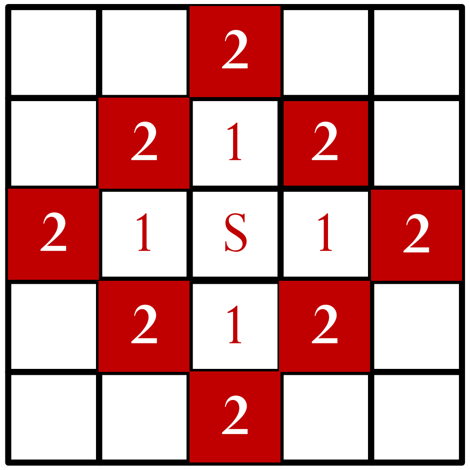


Maze Routing Key Step: Expansion

Start at the **source**



Find all new cells that are reachable at **pathlength 1**, i.e., 1 unit from S in total path length.

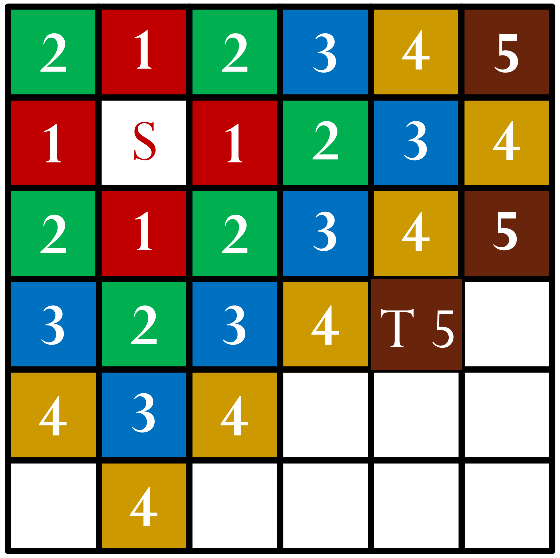


Using the **pathlength 1** cells, find all new cells which are reachable at **pathlength 2**.

Repeat until the target is reached …

Maze Router Step 1: Expand

- Strategy
  - Expand **one cell at a time** until a shortest path 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.



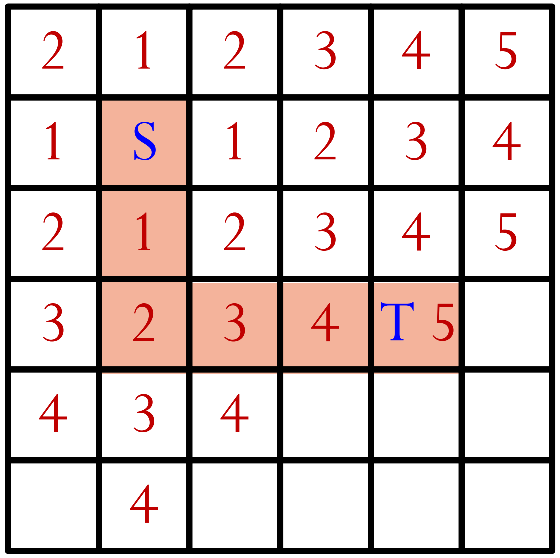
How to Implement Expand?

- A **queue** of reachable squares from the source is used.
- The cell of the source is set with a distance value of 0.
- It is enqueued into an initial empty queue.
- **While** the queue is not empty.
  - A cell is **dequeued** from the queue and made the **examine cell**.
  - For each **unreached unblocked** square adjacent to the **examine cell**
    - Mark its distance as "1 + the distance value of the **examine cell**"
    - Is this cell the target? If yes, path found and return.
    - Otherwise, **enqueued** the cell into the queue.
- When queue becomes empty but has not reached target yet, means no path found.

Breadth-first search!

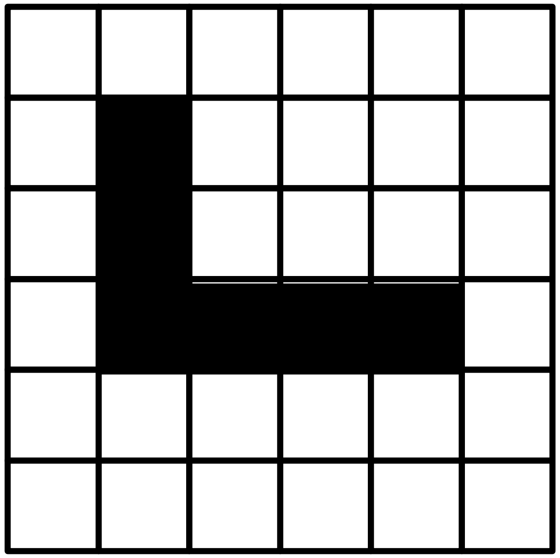
Maze Router Step 2: Backtrace

- Now what? **Backtrace**
  - Select a shortest path (**any** shortest path) from target back to source.
  - Here, just follow the pathlengths in the cells in **descending order**.
  - Mark these cells as **obstacles**, so they cannot be used again for later wires we want to route.



Maze Router Step 3: Clean-Up

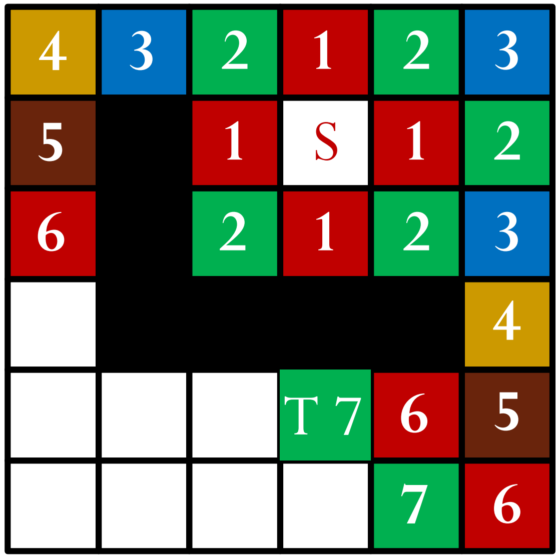
- Now what? **Clean-up**
  - Clean up the grid for the next net, i.e., removing path lengths, etc.
  - Leave the new S to T path as an **obstacle**.
  - Now, ready to route the **next** net!



Maze Router with Obstacles

- 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.

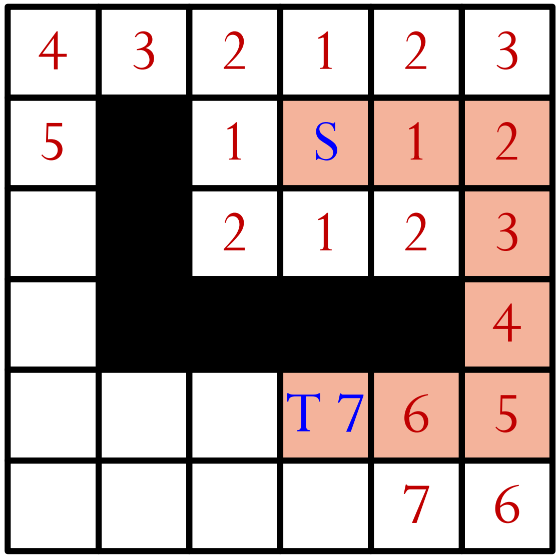
Expand



Maze Router with Obstacles

- 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.

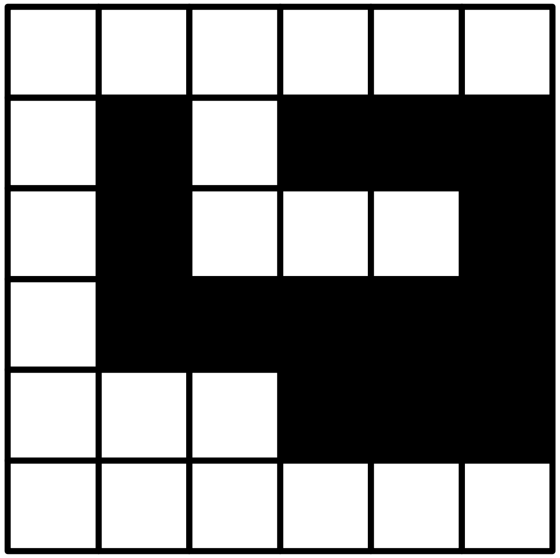
Backtrace



Maze Router with Obstacles

- 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.

Clean-up



Classical Maze Router: Summary

- Expand
  - **Breadth-first-search** to find a path from source to target.
- Backtrace
  - Walk shortest path back to the source and mark the path.
- Clean-Up
  - Erase all distance marks from other grid cells before next net is routed. The cells on the new path are set as used (obstacles).

Outline

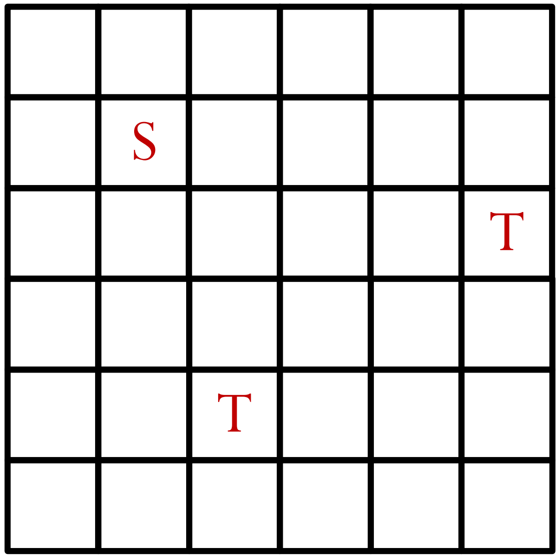
- Routing Basics
- Maze Routing
  - 2-Point Nets in 1 Layer
  - Multi-Point Nets
  - Multi-Layer Routing
  - Non-Uniform Grid Costs

Applications: Multi-point Nets

- Multi-point Nets
  - **One** source → **Many** targets
  - You get this with any net that has >1 **fanout** (i.e., 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 target is "nearest" 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.

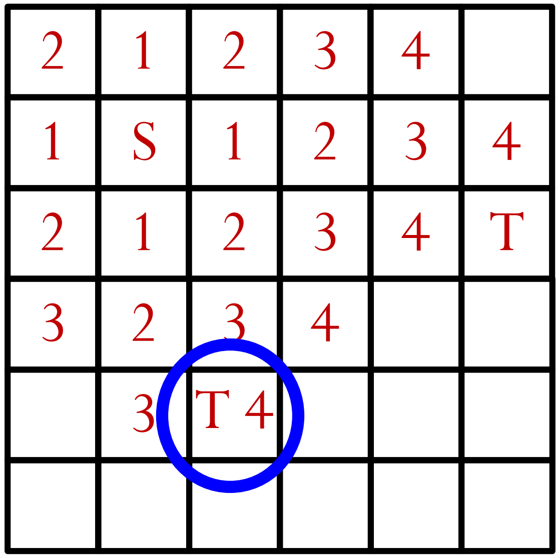
Multi-point Nets

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



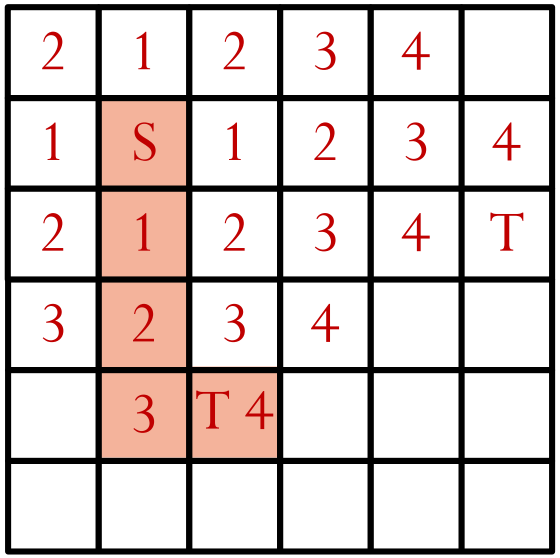
Multi-point Nets

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



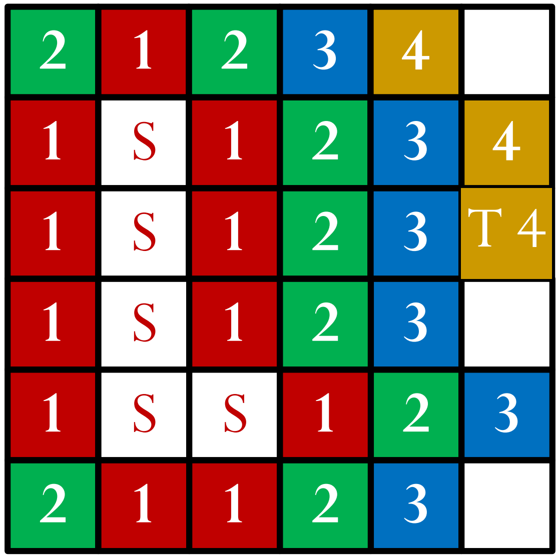
Multi-point Nets

- Next …
  - 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**, 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.



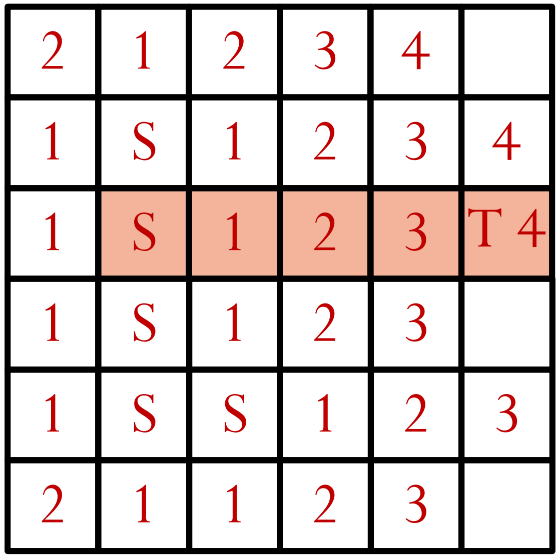
Multi-point Nets

- Next…
  - 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, …
  - … until we reach another target.



Multi-point Nets

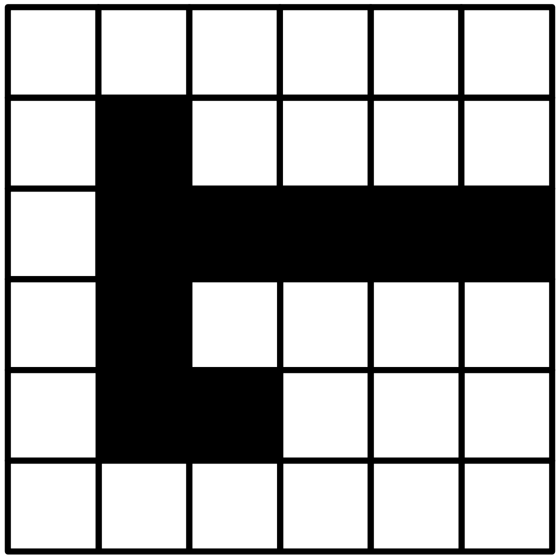
- Next: **Backtrace** as before.
  - Follow pathlengths in decreasing order from target, to **<u>some</u>** source cell.



2	1	2	3	4	
1	S	1	2	3	4
1	S	1	2	3	T 4
1	S	1	2	3	
1	S	S	1	2	3
2	1	1	2	3	

Multi-point Nets

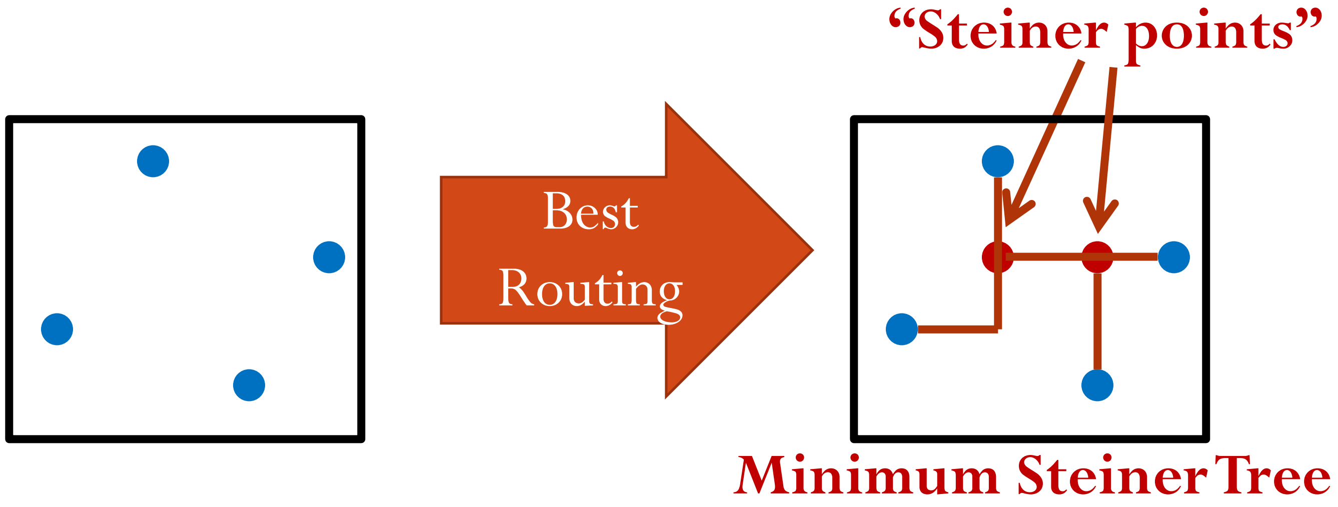
- Finally…
  - Do usual cleanup.
  - Mark all of the segment cells as used and clean-up the grid.
  - Now, have embedded a **multi-point net**, and rendered it as an **obstacle** for future nets.



Aside: Is This Strategy “Optimal”?

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

Aside: About Steiner Tree Constructions



- How hard is to get the optimal Steiner Tree?
  - NP-hard!
- Why this is hard, in general?
  - You don't know at start where the right "**Steiner points**" are, to make shortest wire.
  - There are simple heuristics work well – but no guarantee to get the **best** Steiner tree.

Outline

- Routing Basics

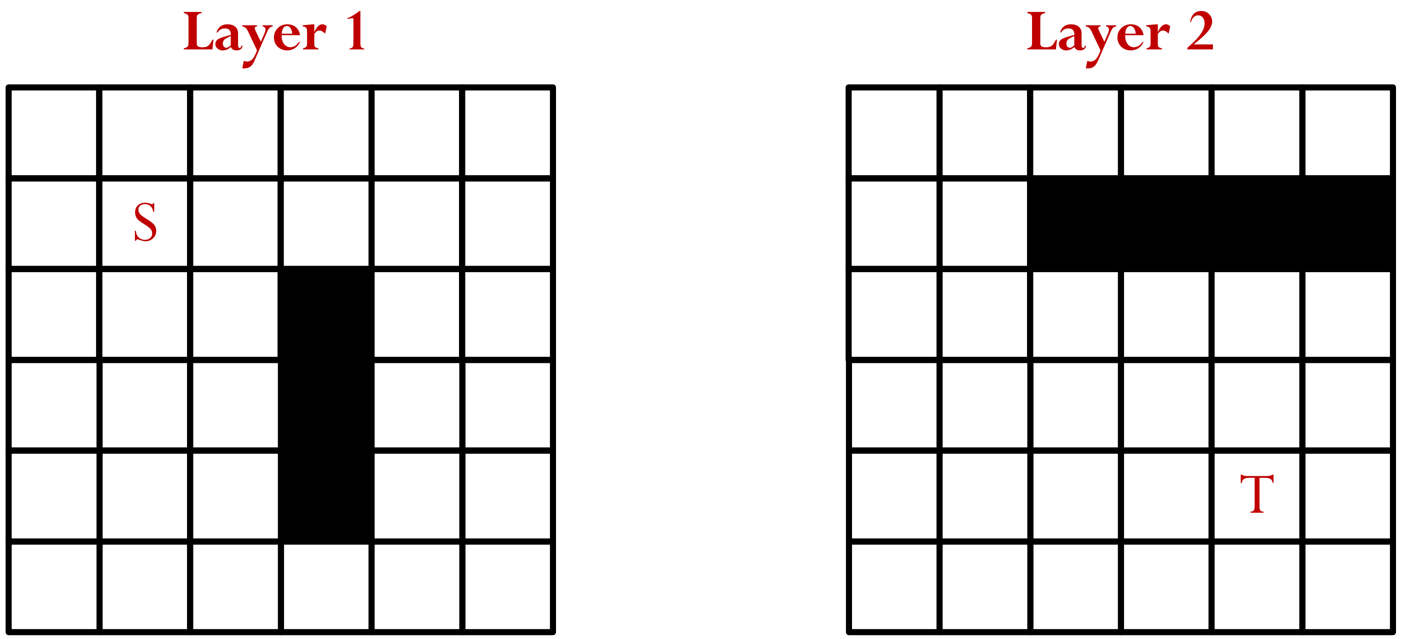
- Maze Routing
  - 2-Point Nets in 1 Layer
  - Multi-Point Nets
  - Multi-Layer Routing
  - Non-Uniform Grid Costs

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
  - On each individual layer: to north/south/east/west neighbors.
  - To **adjacent** layers: **up/down** (between layers) using a via.

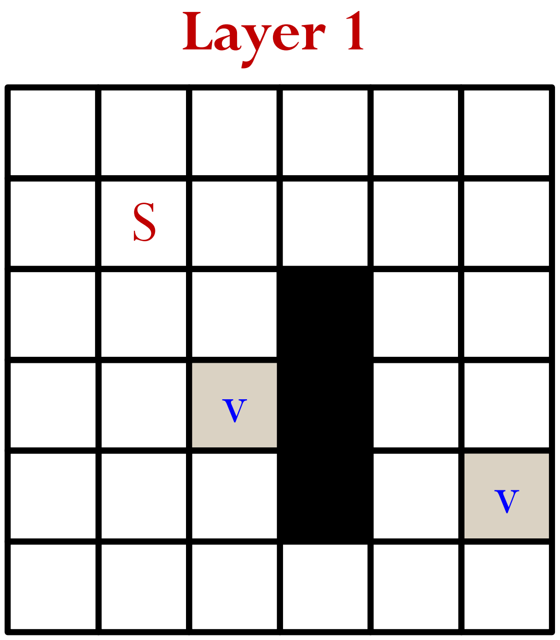
2-Layer Example: Parallel Grids, Vertically Stacked

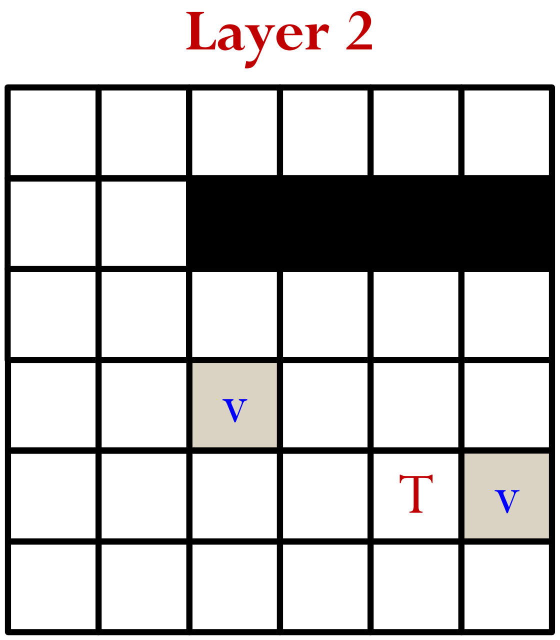
- Expansion can now go **UP** (Layer 1→2) and **DOWN** (Layer 2→1).



Vias

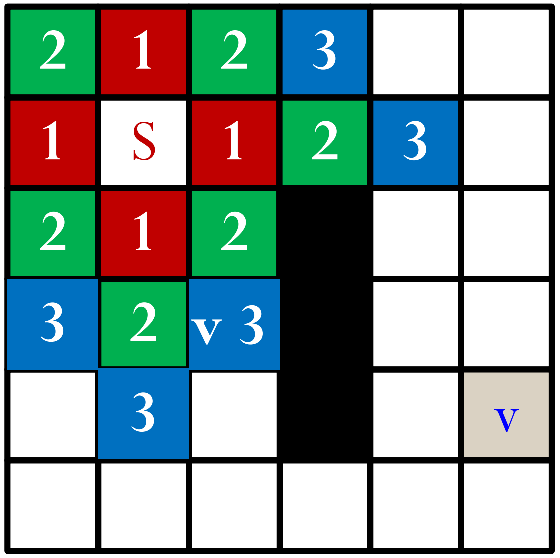
- **Constraint**: vias **cannot** be put everywhere – they are **only** allowed where "v" in grid.



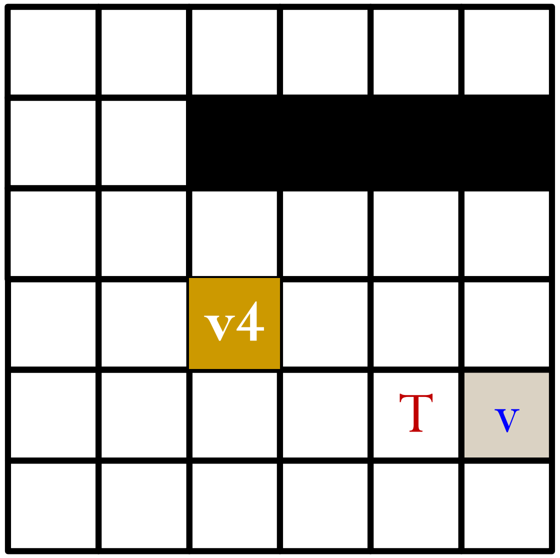


2-Layer Example: Expand Step

Layer 1



Layer 2

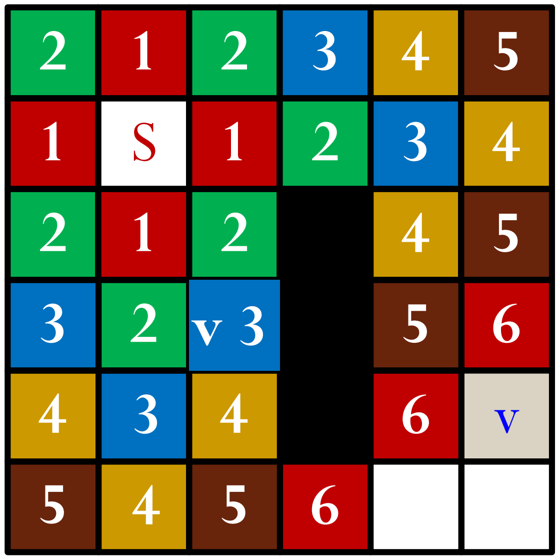


- New: What happens when we reach a place we can put a via?
  - Answer: we can expand on **the other** layer, keep adding unit cell costs (For now, assume that the cost of going through a via is just 1)…

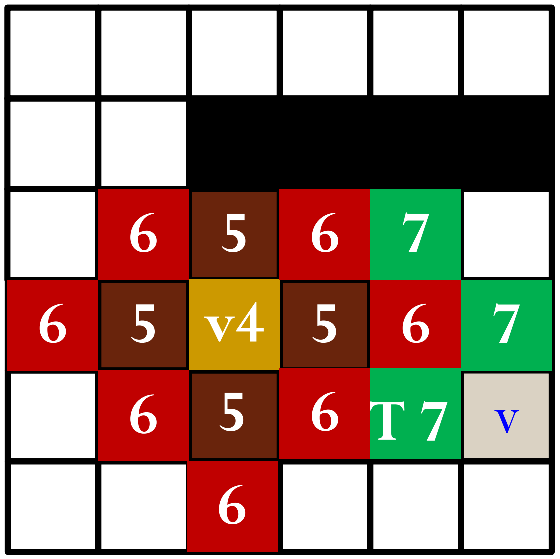
2-Layer Example: Expand Step

- Now, expanding on both layers

Layer 1



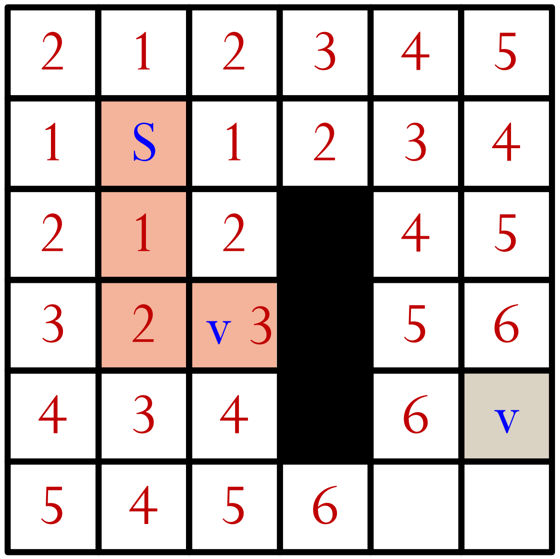
Layer 2



2-Layer Example: Backtrace Step

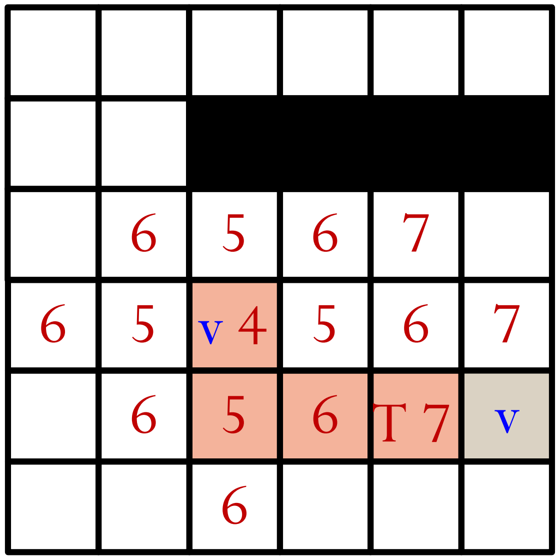
- Backtrace: go through cells and also through vias.

Layer 1



2	1	2	3	4	5
1	S	1	2	3	4
2	1	2	■	4	5
3	2	v 3	■	5	6
4	3	4	■	6	v
5	4	5	6		

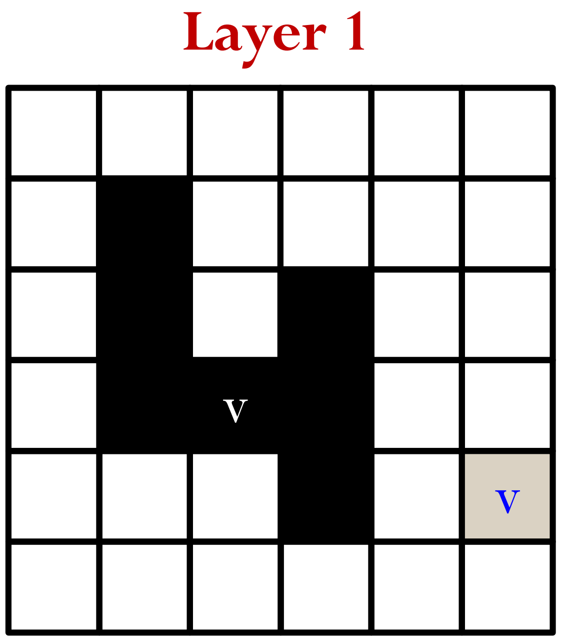
Layer 2

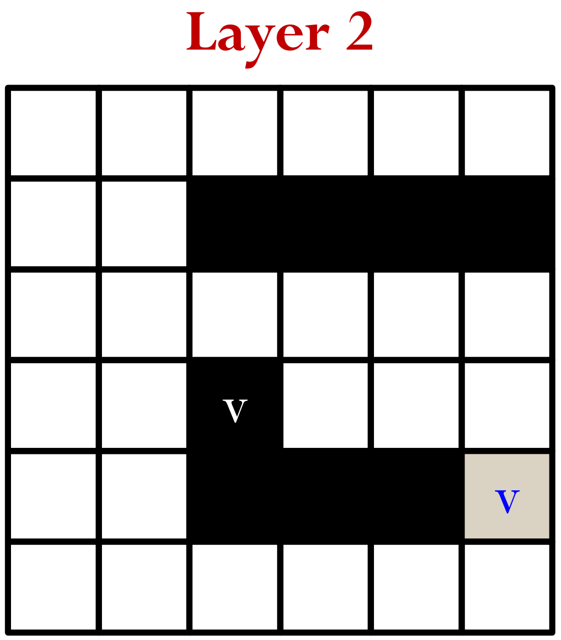


		■	■	■	■
	6	5	6	7	
6	5	v 4	5	6	7
	6	5	6	T 7	v
		6			

2-Layer Example: Clean-up Step

- Cleanup as usual. Now, new obstacles on **both** layers.





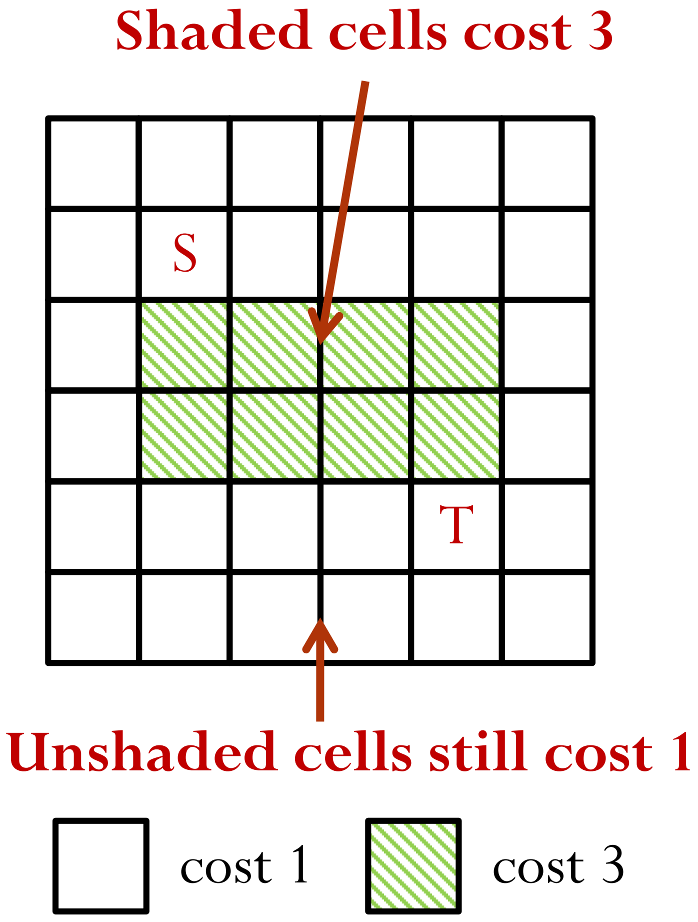
Outline

- Routing Basics

- Maze Routing
    - 2-Point Nets in 1 Layer
    - Multi-Point Nets
    - Multi-Layer Routing
    - Non-Uniform Grid Costs

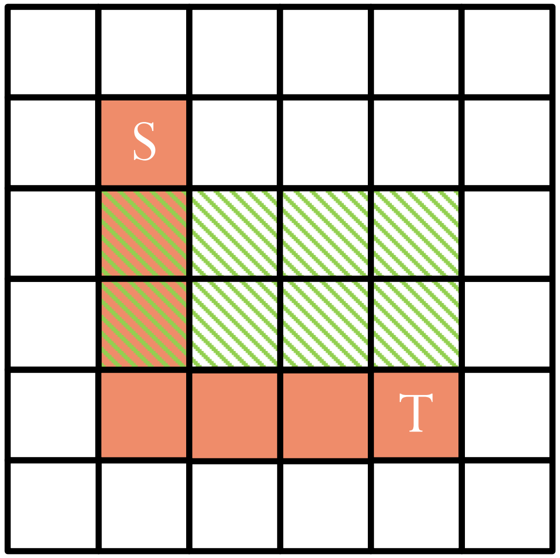
New Feature: Non-Uniform Grid Costs

- Old problem
  - Each cell in grid costs the **same** to cross it with a wire.
  - Cost=1 over all the cells.
  - Is this necessary? **No!**
- Now
  - Given grid, source and target, we have **different** costs for each cell.
- The objective:
  - Find **minimum cost** path connecting source and target.

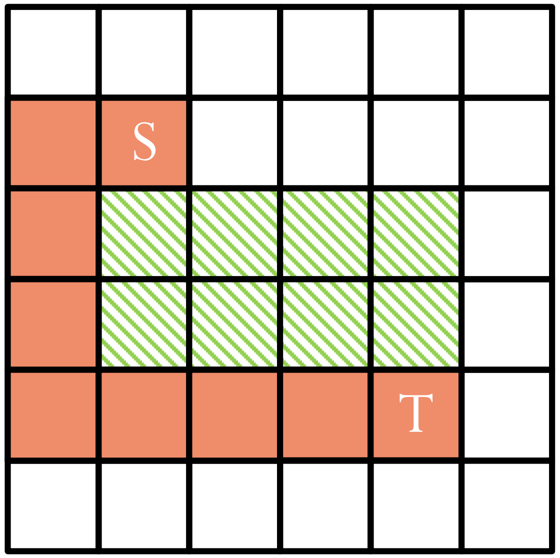


Path Cost

- Define path cost = Σpath (cell costs)



Path Cost = 10



Path Cost = 8

□ cost 1 ▨ cost 3

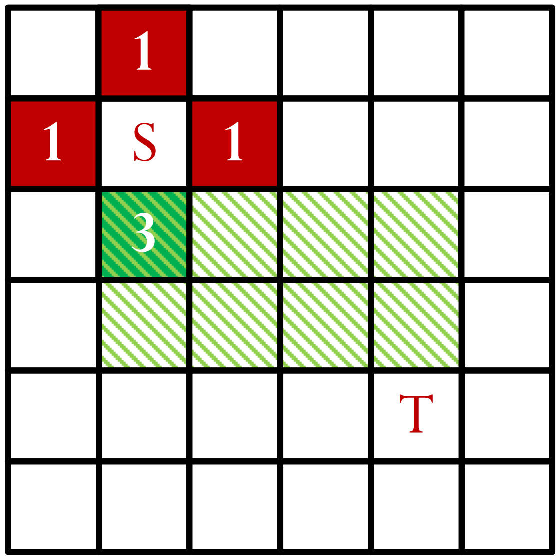
Feature: Non-Uniform Grid Costs

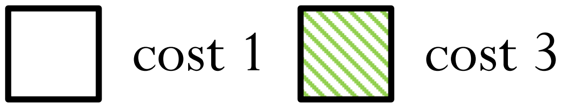
- Vital feature of **all** real routers
  - Use costs to **encourage** wires to take shapes and paths we prefer.
  - Can make the router **avoid congested areas** (too many wires want to be here).
  - Can make **different layers** have different expense to use.
  - Can make **different vias** have different 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…
- Expansion process **uses costs**.

Subtle Search Issues with Non-Unit Costs

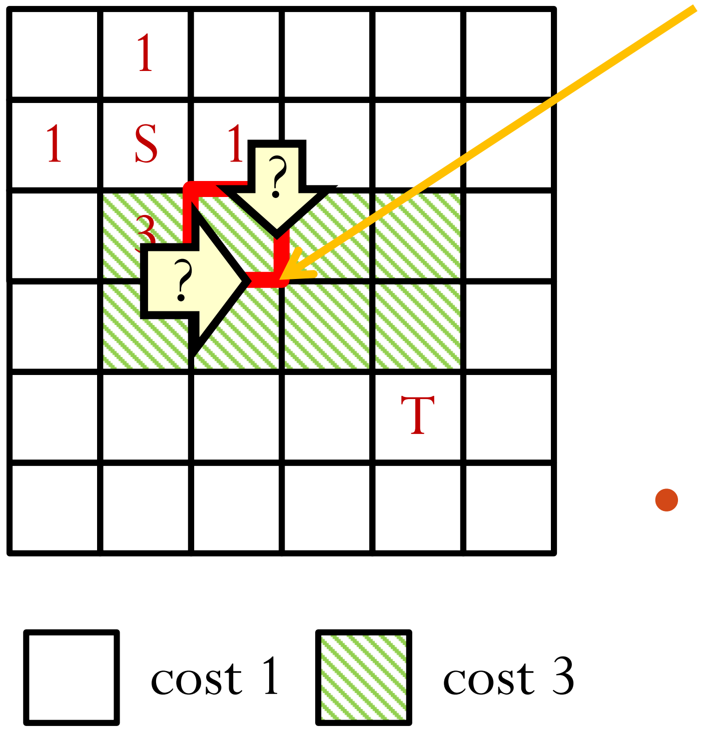
- Search all cells that are **adjacent** to source S.
  - When we reach a new cell, we label it with a **pathcost**.
  - **pathcost** = **pathcost of neighbor** plus cell's own cost.

- So far, so good…





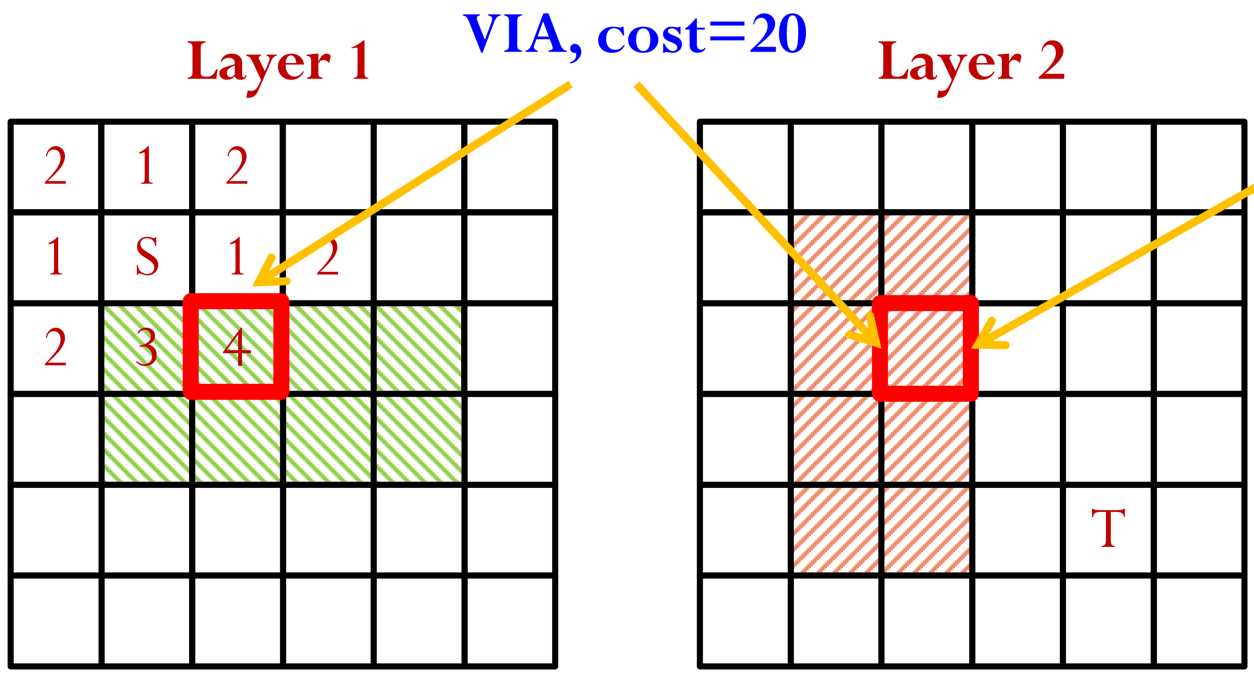
Subtle Search Issues with Non-Unit Costs



- **What is this cell's pathcost?**
  - Is it 1+3=4, reached "from" cell with cost 1 that is to the North?
  - Is it 3+3=6, reached "from" cell with cost 3 that is to the West?
- Answer: 4
  - Always want to label cells with the **minimum pathcost** to reach that cell.
- From this example: the order of expansion is important.

Subtle Issues: Via Costs

- Suppose we want to "expand" from "4" on Layer 1, through a via, to Layer 2.



Cost to reach this cell is 4+20+6=30:

- 4: **pathcost** to cell on Layer 1
- 20: **via cost**
- 6: this Layer 2 **cell cost**

☐ cost 1 ▨ cost 3 ▨ cost 6