

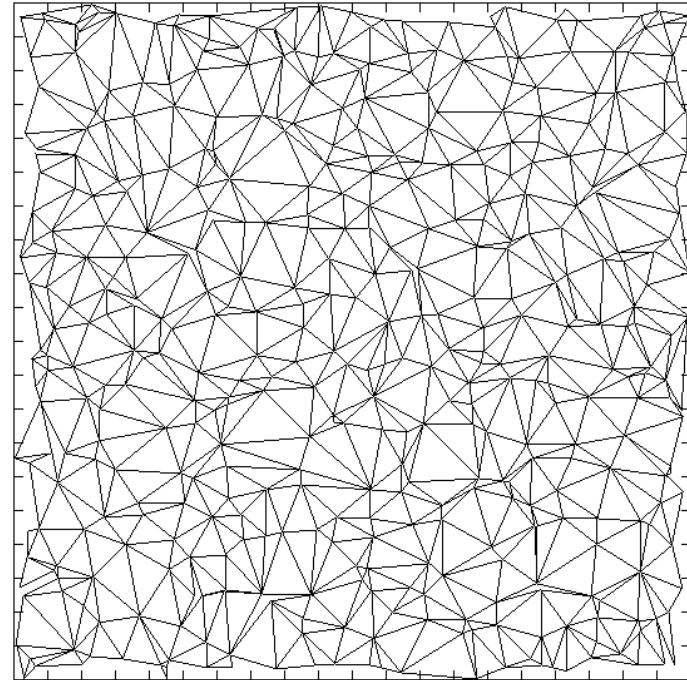
- Previous Lecture:
 - Comparing different sorting algorithms
 - Recursion

- Today's Lecture:
 - Recursion review
 - Efficiency: dealing with “expensive” function evaluation
 - A model to quantify importance: Google “Page Rank”

- Announcement:
 - Discussion this week in the computer lab (UP B7)
 - P6 due Thursday at 11pm

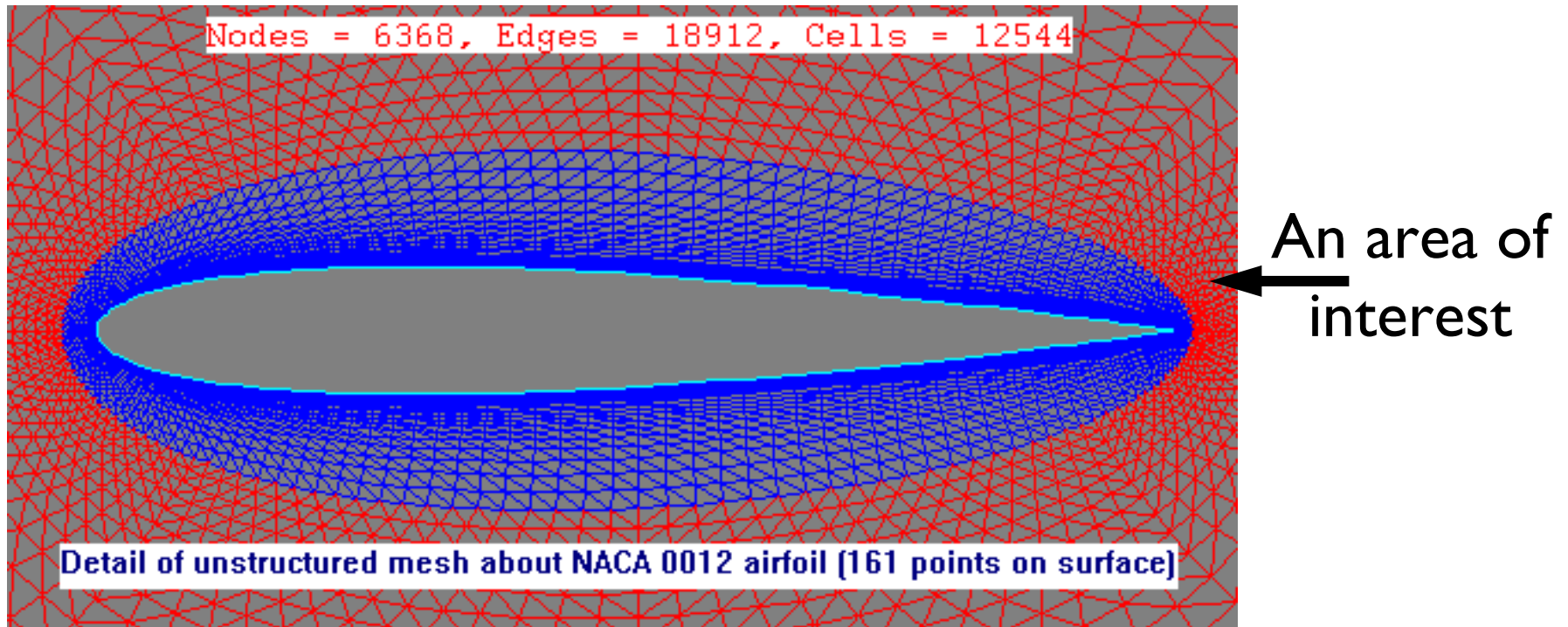
Divide-and-conquer methods also show up in geometric situations

Chop a region up into triangles with smaller triangles in “areas of interest”



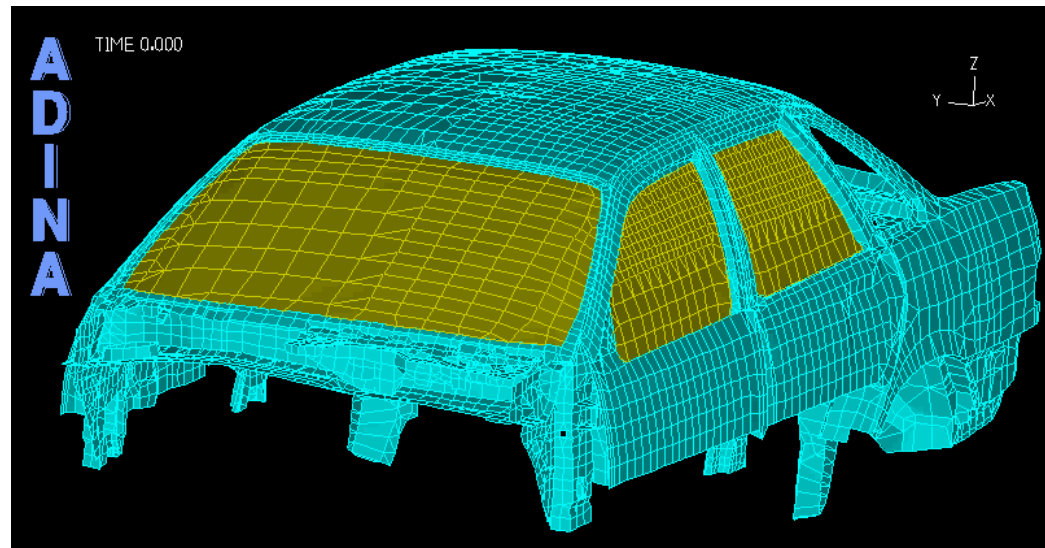
Recursive mesh generation

Mesh Generation



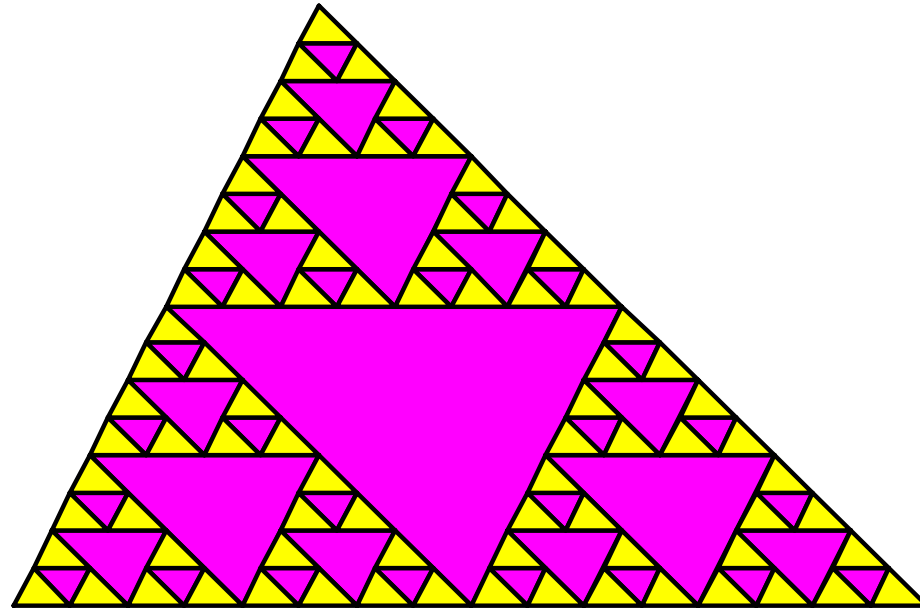
Step one in simulating flow around an airfoil is to generate a mesh and (say) estimate velocity at each mesh point.

Mesh Generation in 3D

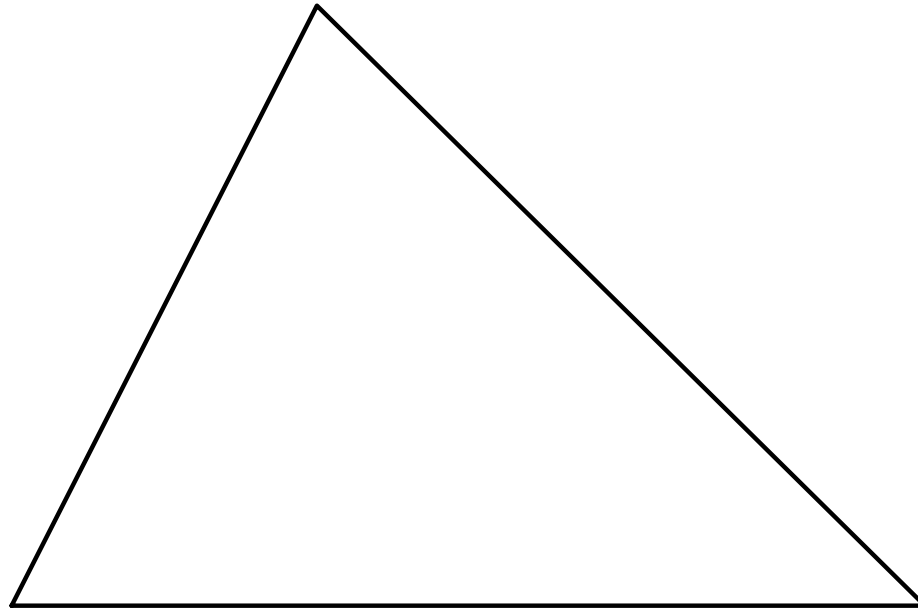


Why is mesh generation a divide-&-conquer process?

Let's draw this graphic

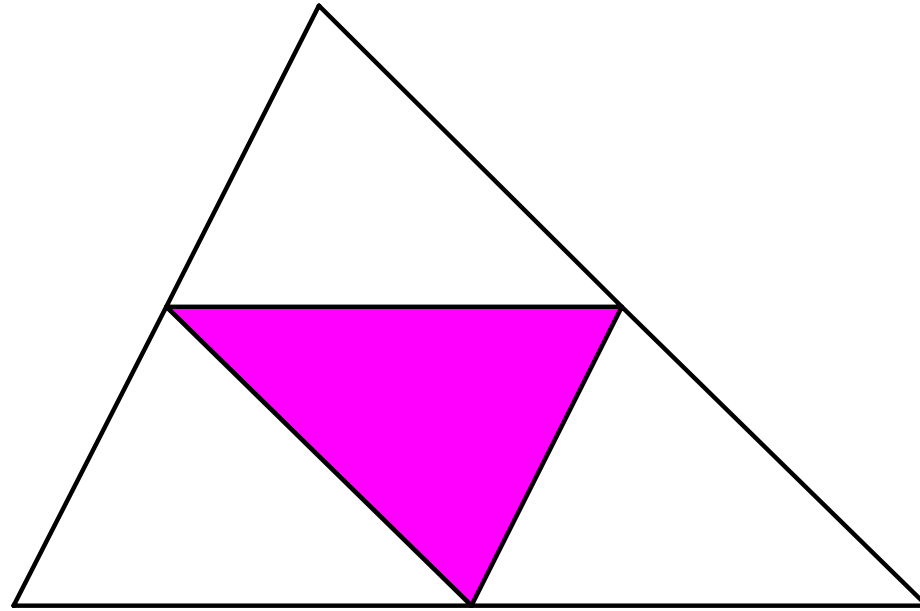


Start with a triangle



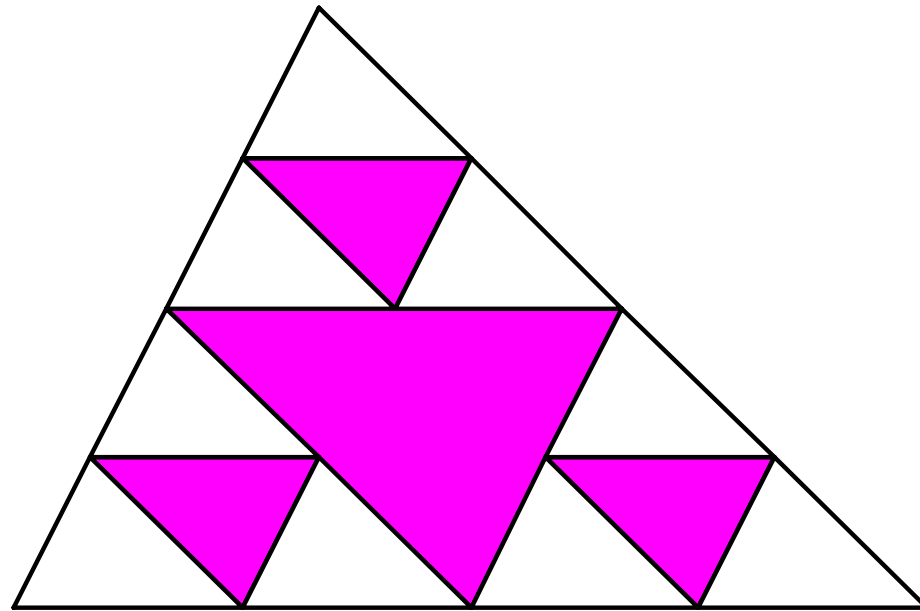
A “level-1” partition of the triangle

(obtained by connecting the midpoints of the sides of the original triangle)

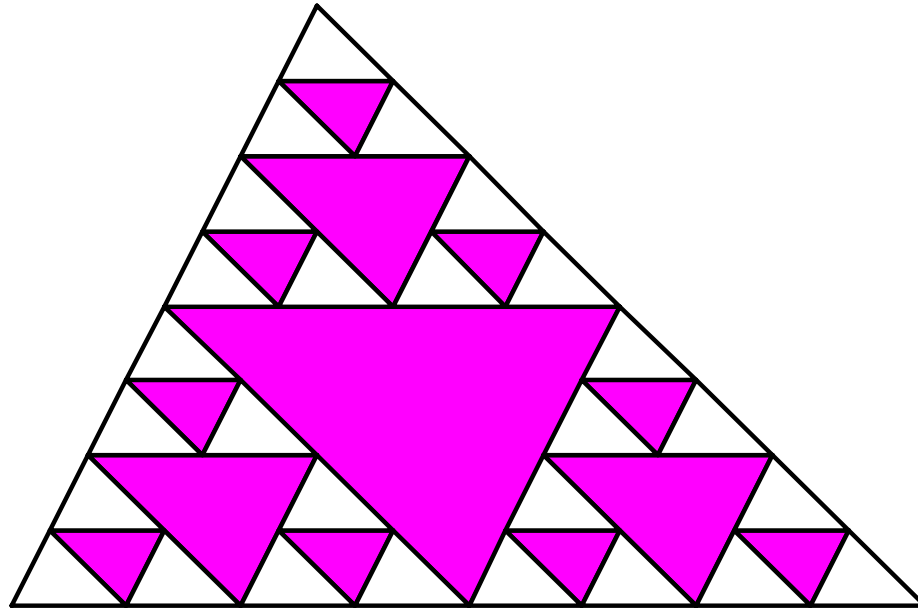


Now do the same partitioning (connecting midpts) on each corner (white) triangle to obtain the “level-2” partitioning

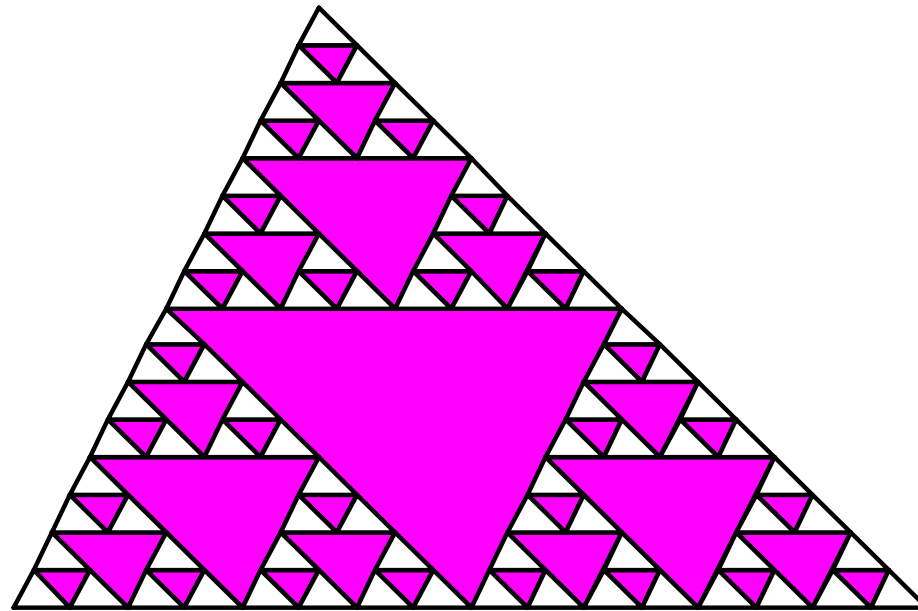
The “level-2” partition of the triangle



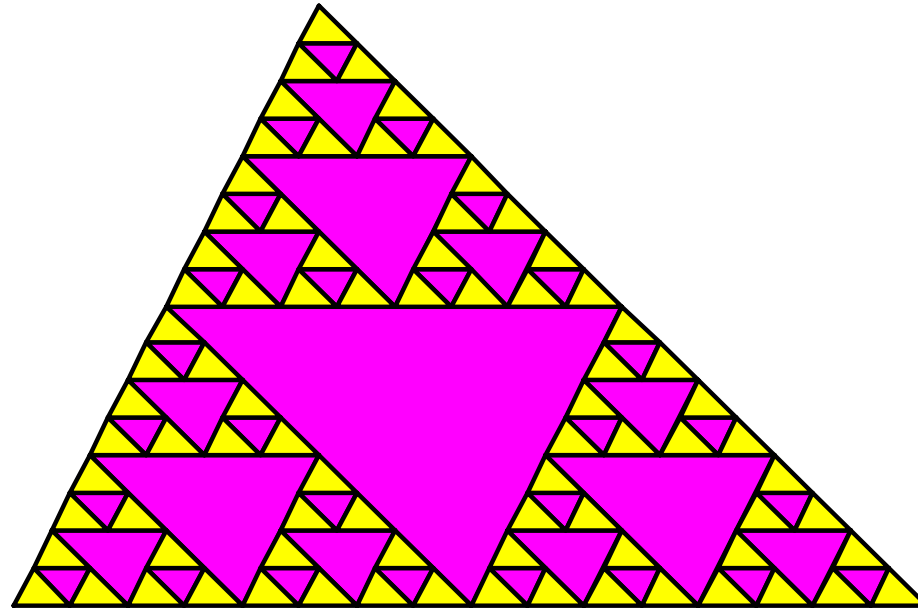
The “level-3” partition of the triangle



The “level-4” partition of the triangle



The “level-4” partition of the triangle



The basic operation at each level

if *the triangle is small*

Don't subdivide and just color it yellow.

else

Subdivide:

Connect the side midpoints;

color the interior triangle magenta;

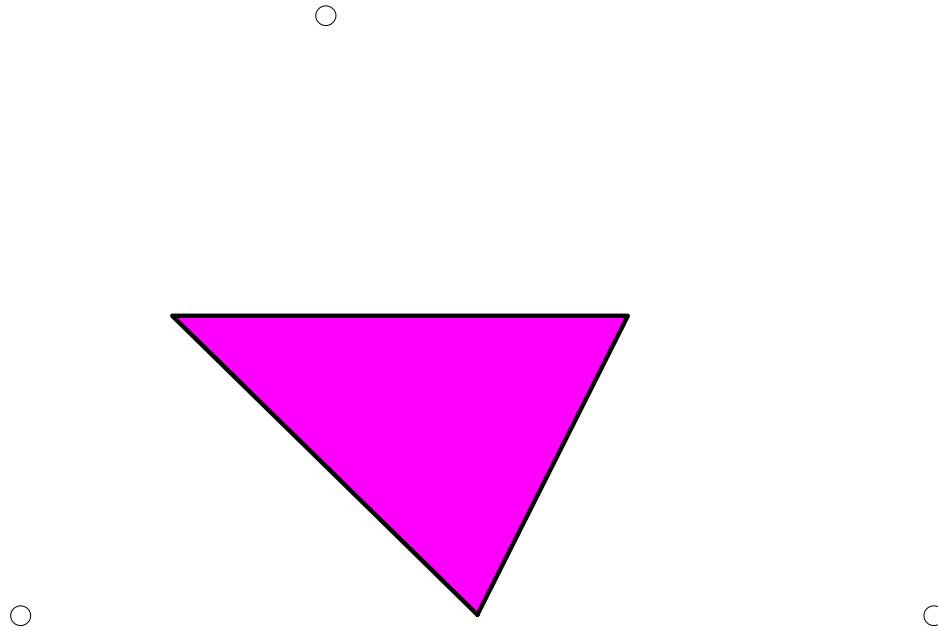
apply same process to each outer triangle.

end

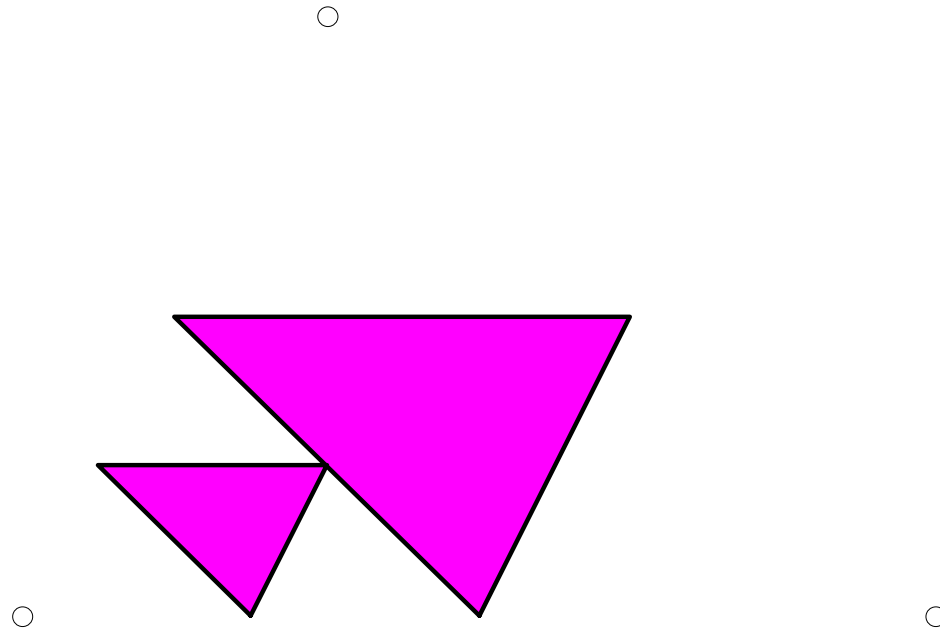
Draw a level-4 partition of the triangle with these vertices



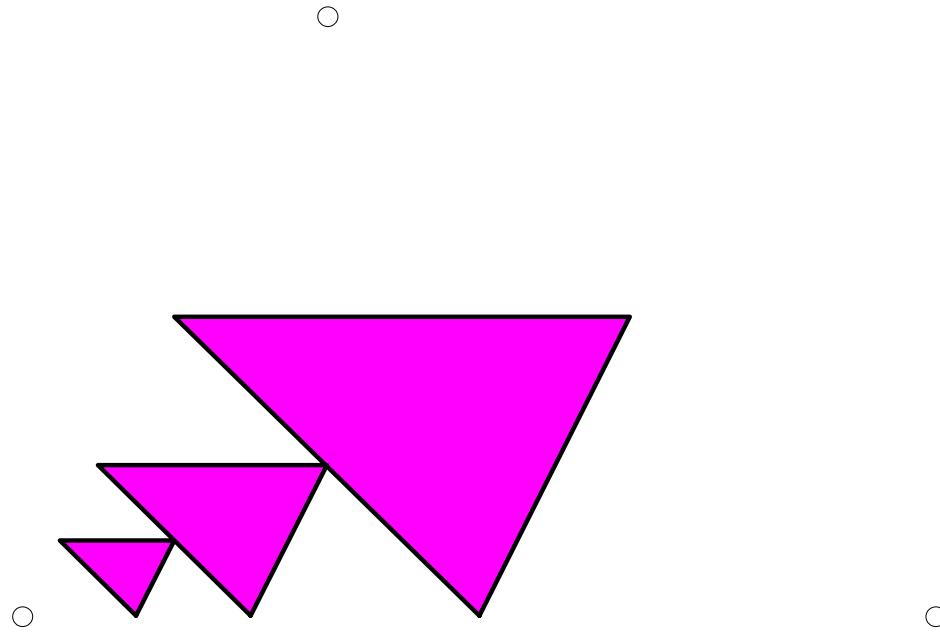
At the start...



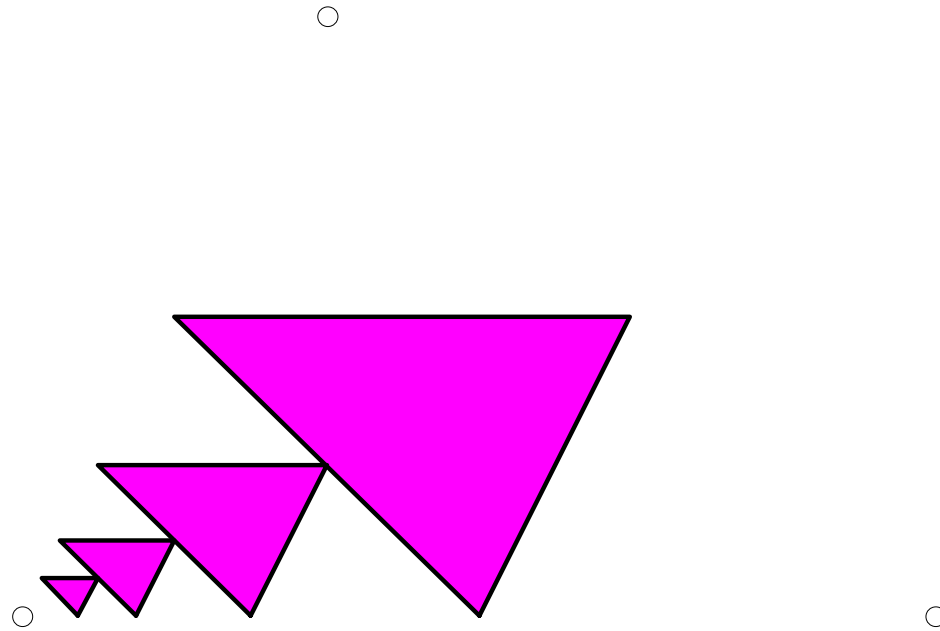
Recur: apply the same process on the lower left triangle



Recur again

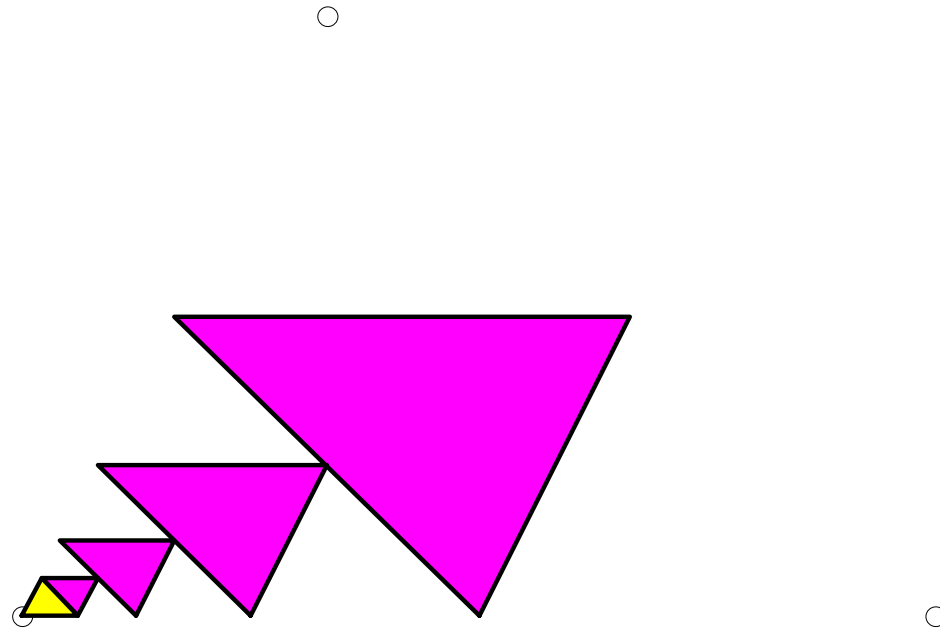


... and again

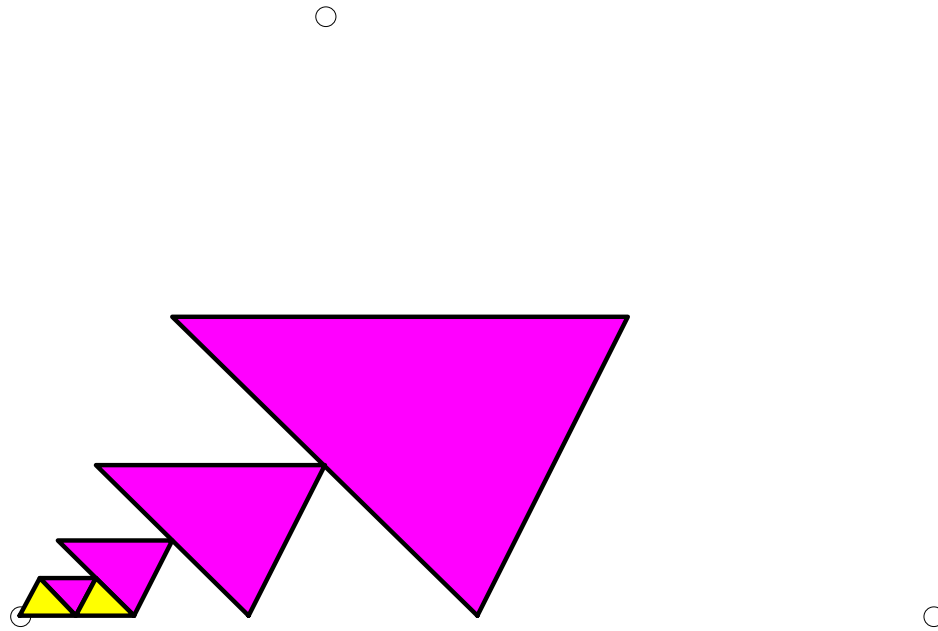


The next lower left corner triangle (white) is small—no more subdivision and just color it yellow.

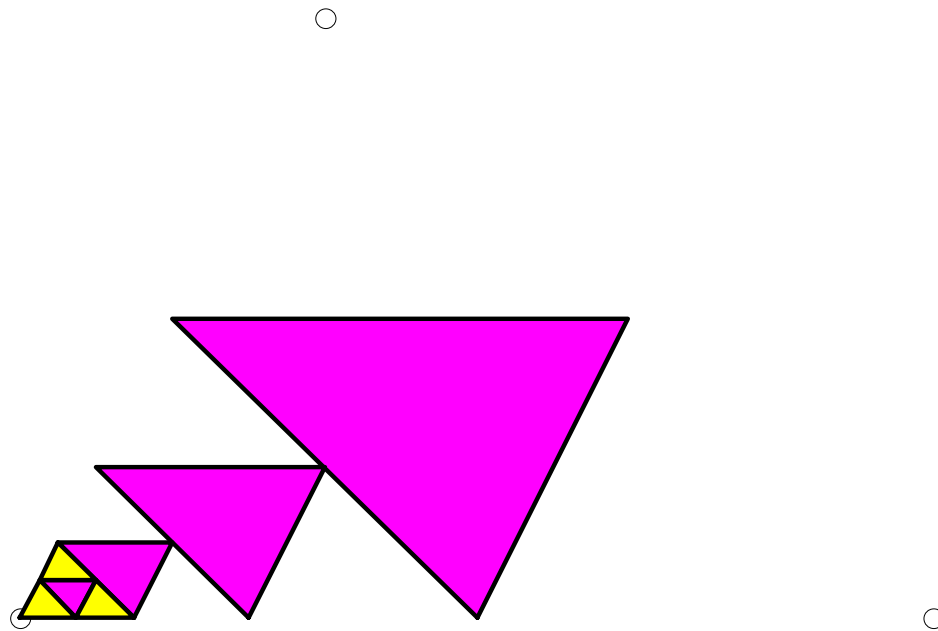
Now lower left corner triangle of the “level-4” partition is done. Continue with another corner triangle

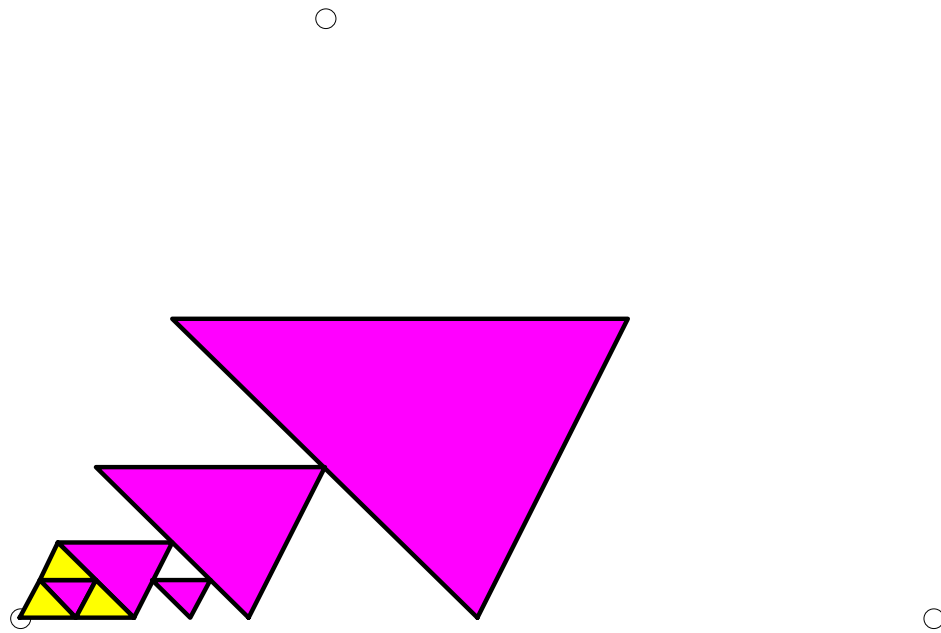


... and continue

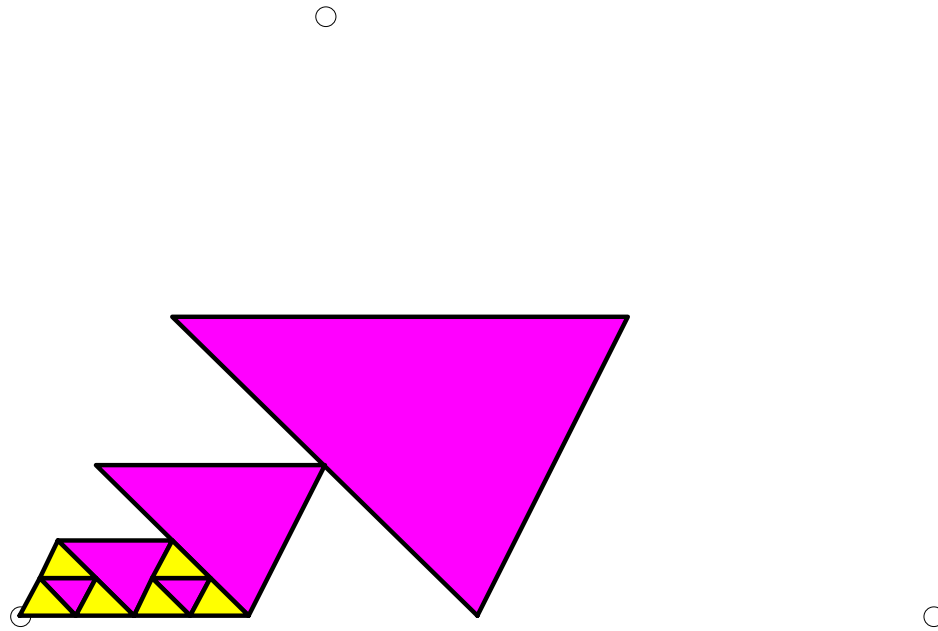


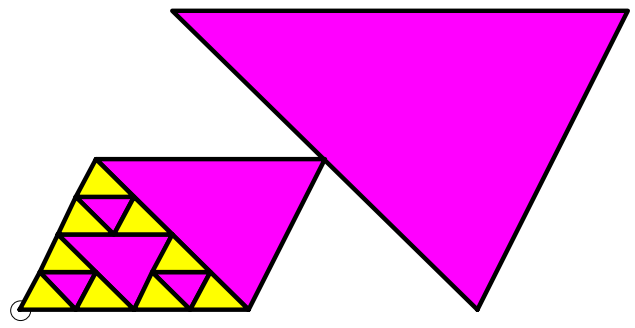
Now the lower left corner triangle of the “level-3” partition is done. Continue with another corner triangle...

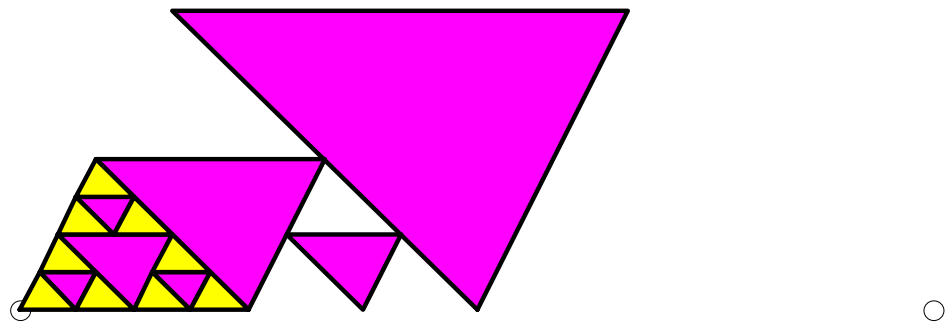


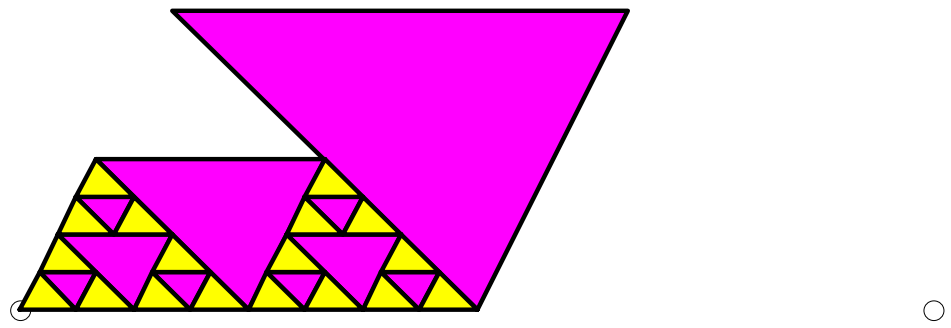


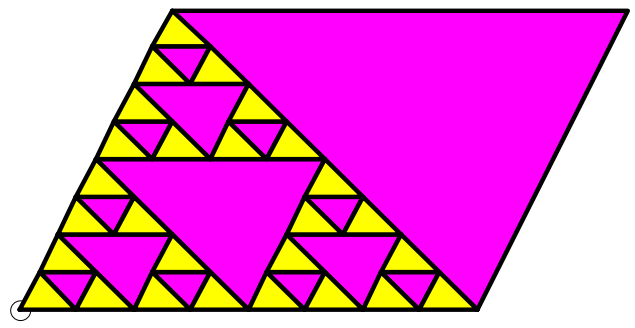
We're "climbing our way out" of the deepest level of partitioning

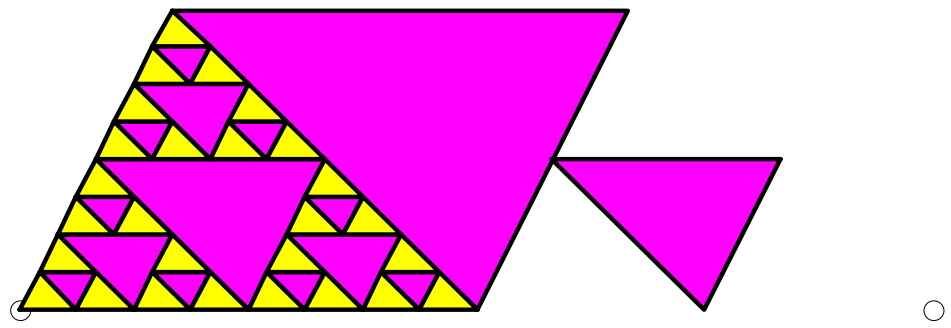


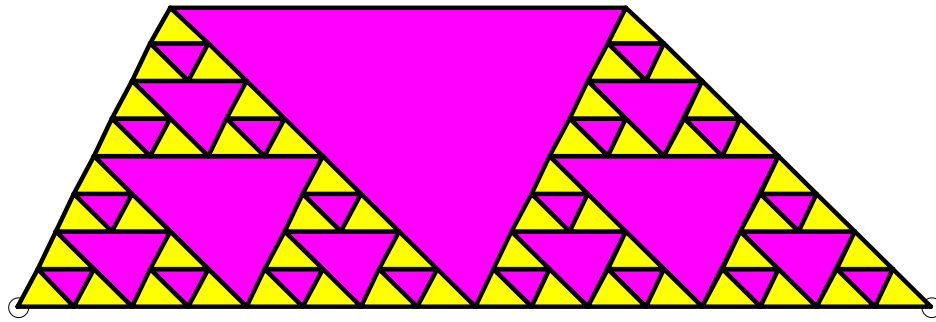


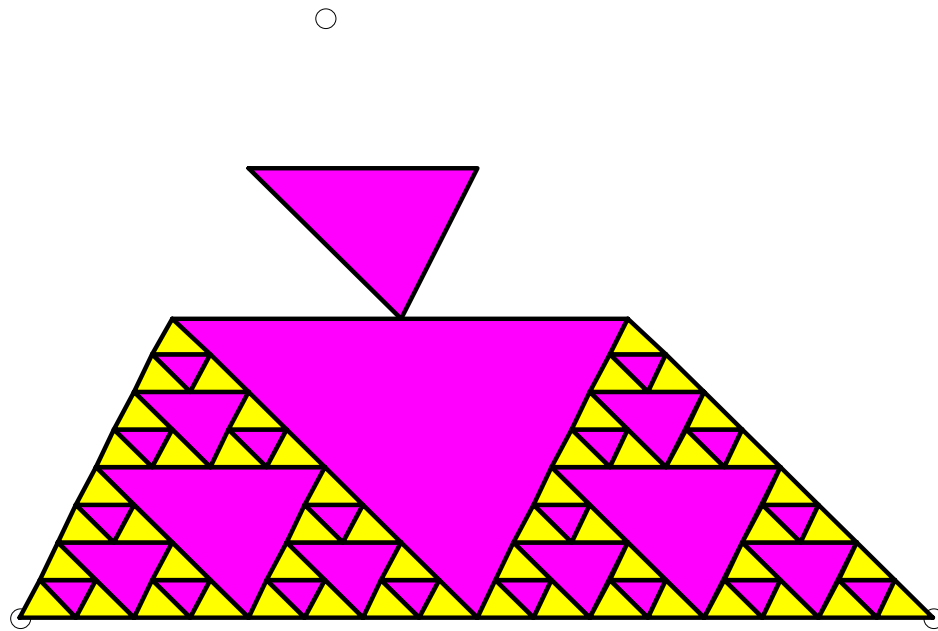




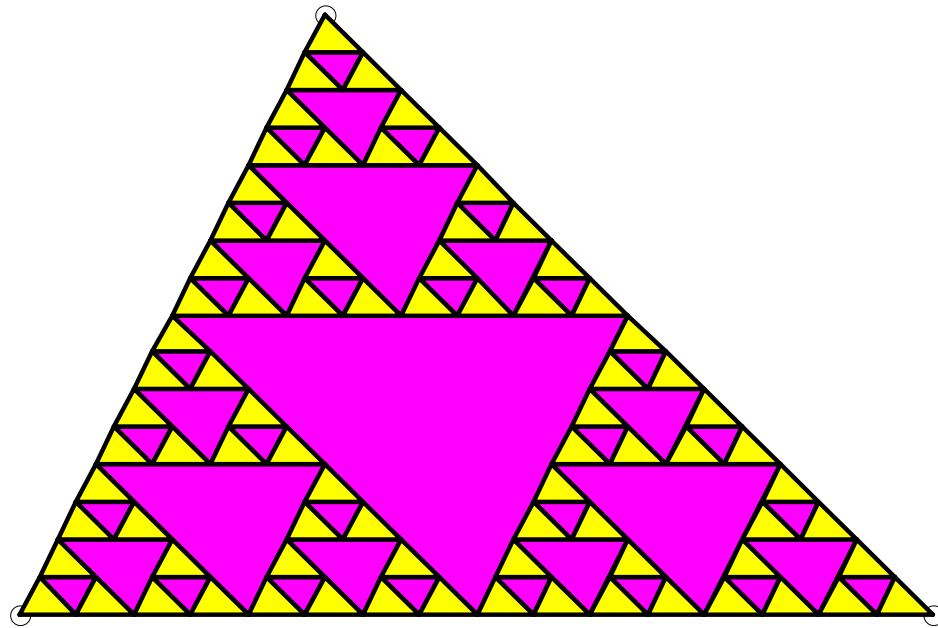








Eventually climb all the way out to get the final result



The basic operation at each level

if *the triangle is small*

Don't subdivide and just color it yellow.

else

Subdivide:

Connect the side midpoints;

color the interior triangle magenta;

Apply same process to each outer triangle.

end

```

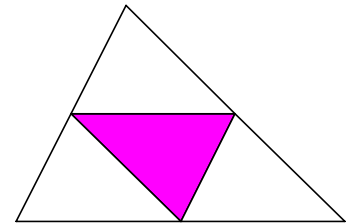
function MeshTriangle(x,y,L)
% x,y are 3-vectors that define the vertices of a triangle.
% Draw level-L partitioning.  Assume hold is on.

if L==0
    % Recursion limit reached; no more subdivision required.
    fill(x,y,'y') % Color this triangle yellow
else
    % Need to subdivide:  determine the side midpoints; connect
    % midpts to get "interior triangle"; color it magenta.

    % Apply the process to the three "corner" triangles...

end

```



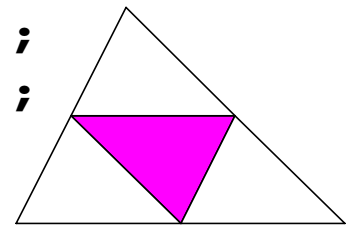

```

function MeshTriangle(x,y,L)
% x,y are 3-vectors that define the vertices of a triangle.
% Draw level-L partitioning.  Assume hold is on.

if L==0
    % Recursion limit reached; no more subdivision required.
    fill(x,y,'y') % Color this triangle yellow
else
    % Need to subdivide:  determine the side midpoints; connect
    % midpts to get "interior triangle"; color it magenta.
    a = [(x(1)+x(2))/2 (x(2)+x(3))/2 (x(3)+x(1))/2];
    b = [(y(1)+y(2))/2 (y(2)+y(3))/2 (y(3)+y(1))/2];
    fill(a,b,'m')
    % Apply the process to the three "corner" triangles...

end

```



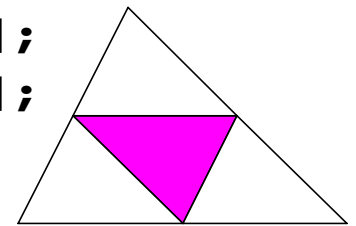
```

function MeshTriangle(x,y,L)
% x,y are 3-vectors that define the vertices of a triangle.
% Draw level-L partitioning. Assume hold is on.

if L==0
    % Recursion limit reached; no more subdivision required.
    fill(x,y,'y') % Color this triangle yellow
else
    % Need to subdivide: determine the side midpoints; connect
    % midpts to get "interior triangle"; color it magenta.
    a = [(x(1)+x(2))/2 (x(2)+x(3))/2 (x(3)+x(1))/2];
    b = [(y(1)+y(2))/2 (y(2)+y(3))/2 (y(3)+y(1))/2];
    fill(a,b,'m')

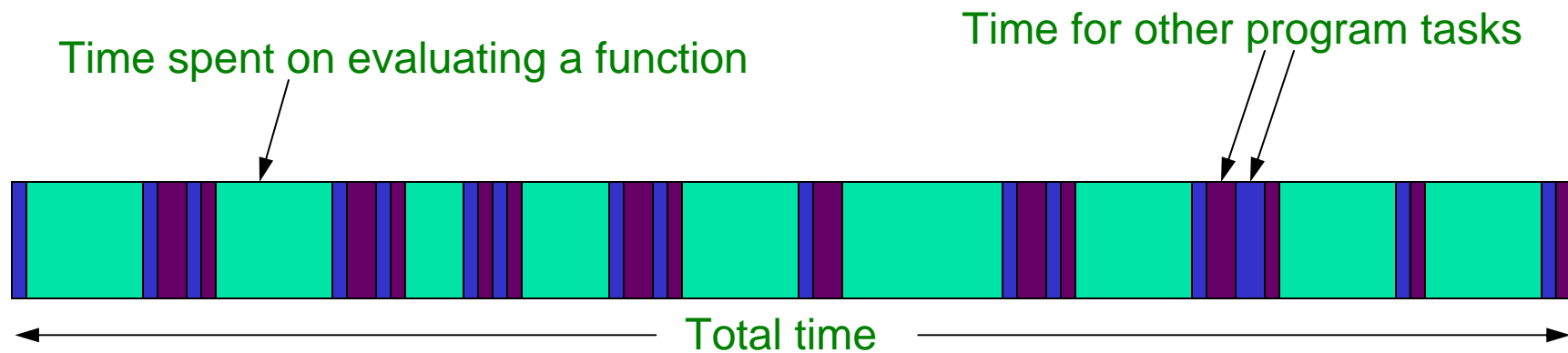
    % Apply the process to the three "corner" triangles...
    MeshTriangle([x(1) a(1) a(3)], [y(1) b(1) b(3)], L-1)
    MeshTriangle([x(2) a(2) a(1)], [y(2) b(2) b(1)], L-1)
    MeshTriangle([x(3) a(3) a(2)], [y(3) b(3) b(2)], L-1)
end

```



Expensive function evaluations

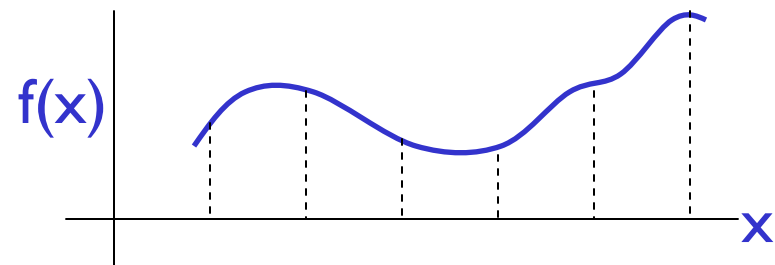
- Consider the execution of a program that is dominated by multiple calls to an expensive-to-evaluate function (e.g., climate simulation models)



- Can try to improve efficiency by dealing with the expensive function evaluations

Dealing with expensive function evaluations

- Can the function code be improved?
- Can we do fewer function evaluations?
- Can we **pre-compute and store** specific function values so that during the main program execution the program can just **look up** the values?
 - Consider function $f(x)$. If there are many function calls and few distinct values of x , can get substantial speedup
 - Only speeds up main program execution—it still takes time to do the pre-computation



What are some issues and potential problems with the “table look-up” strategy?

\mathbf{x}	$\mathbf{f}(\mathbf{x})$
1	1.01
2	2.67
3	5.71
4	9.12
5	7.98
:	:

Pre-calculate and store these values (e.g., in a vector \mathbf{H})

- Accuracy—need a “dense grid” to get high accuracy
→ significant memory usage
- If an exact x -value is not found, need some kind of approximation
- Incur searching cost if the x -values are not simple indices
- Feasible in high dimensions (multiple dependent variables)?

To be continued in this week’s lab.

Quantifying Importance

How do you rank web pages for importance given that you know the link structure of the Web, i.e., the in-links and out-links for each web page?

A related question:

How does a deleted or added link on a webpage affect its “rank”?

Background

Index all the pages on the Web from 1 to n . (n is around ten billion.)

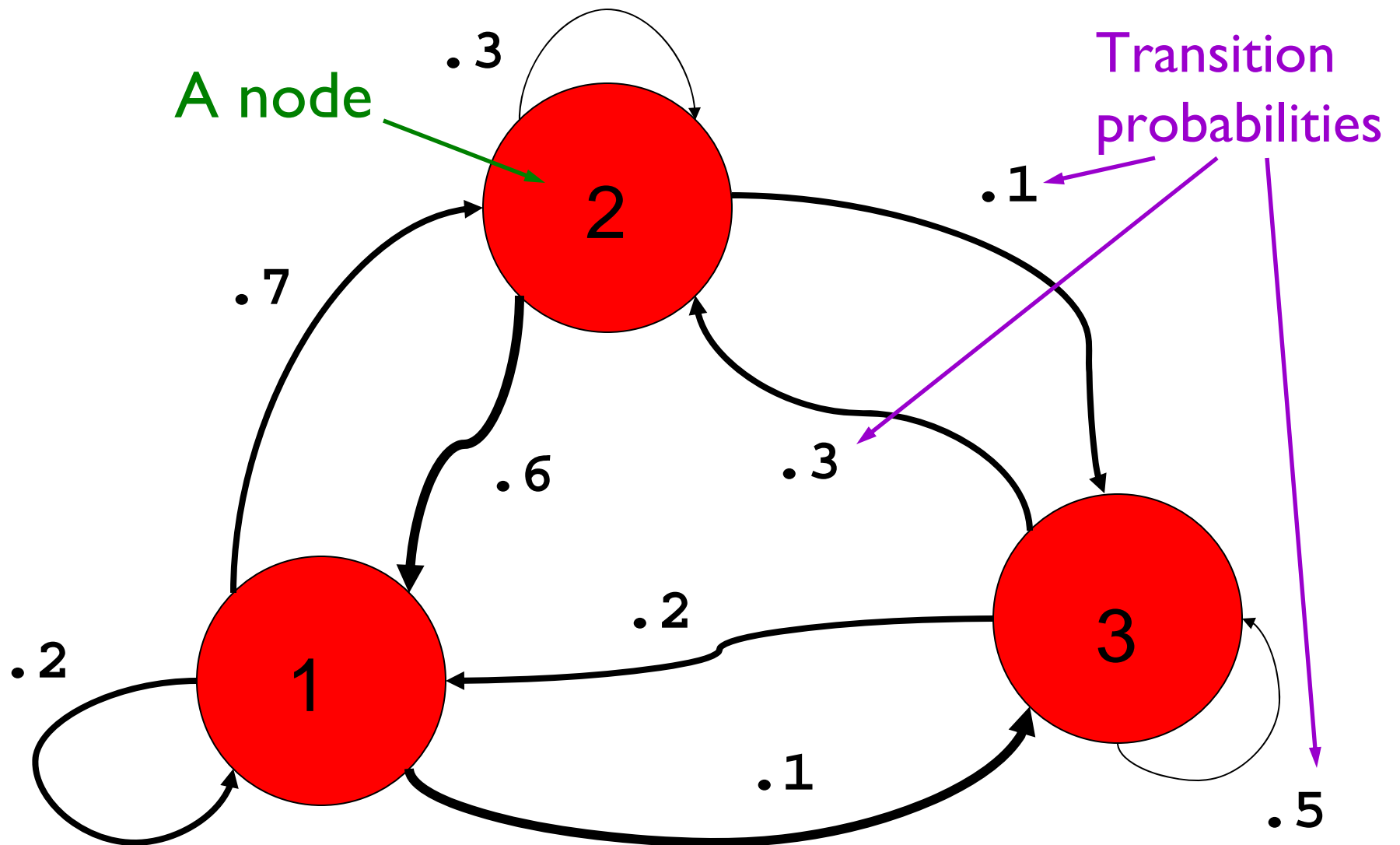
The **PageRank** algorithm orders these pages from “most important” to “least important.”

It does this by **analyzing links, not content.**

Key ideas

- There is a random web surfer—a special **random walk**
- The surfer has some random “surfing” behavior—a **transition probability matrix**
- The transition probability matrix comes from the link structure of the web—a **connectivity matrix**
- Applying the transition probability matrix → **Page Rank**

A 3-node network with specified transition probabilities



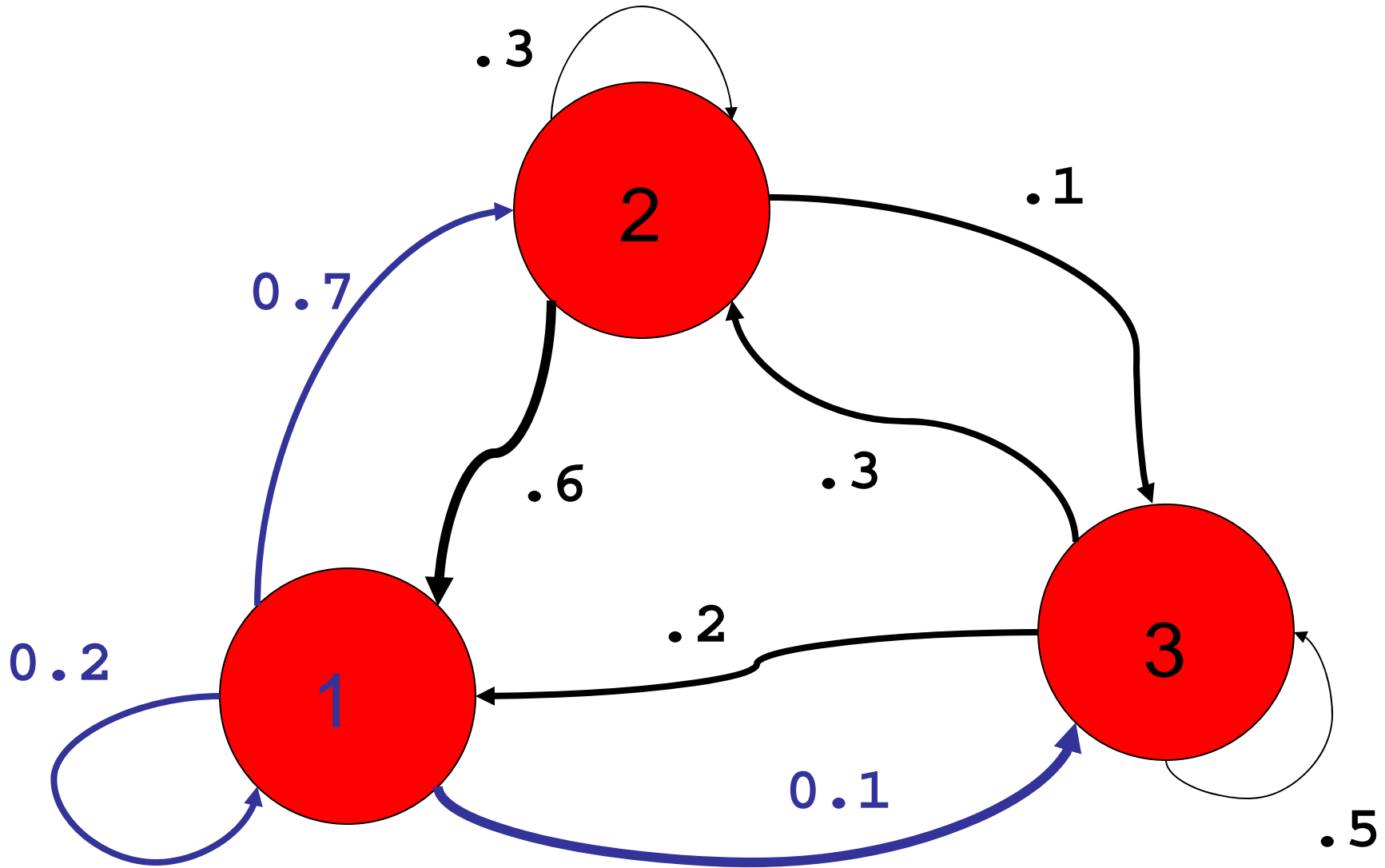
A special random walk

Suppose there are a 1000 people on each node.

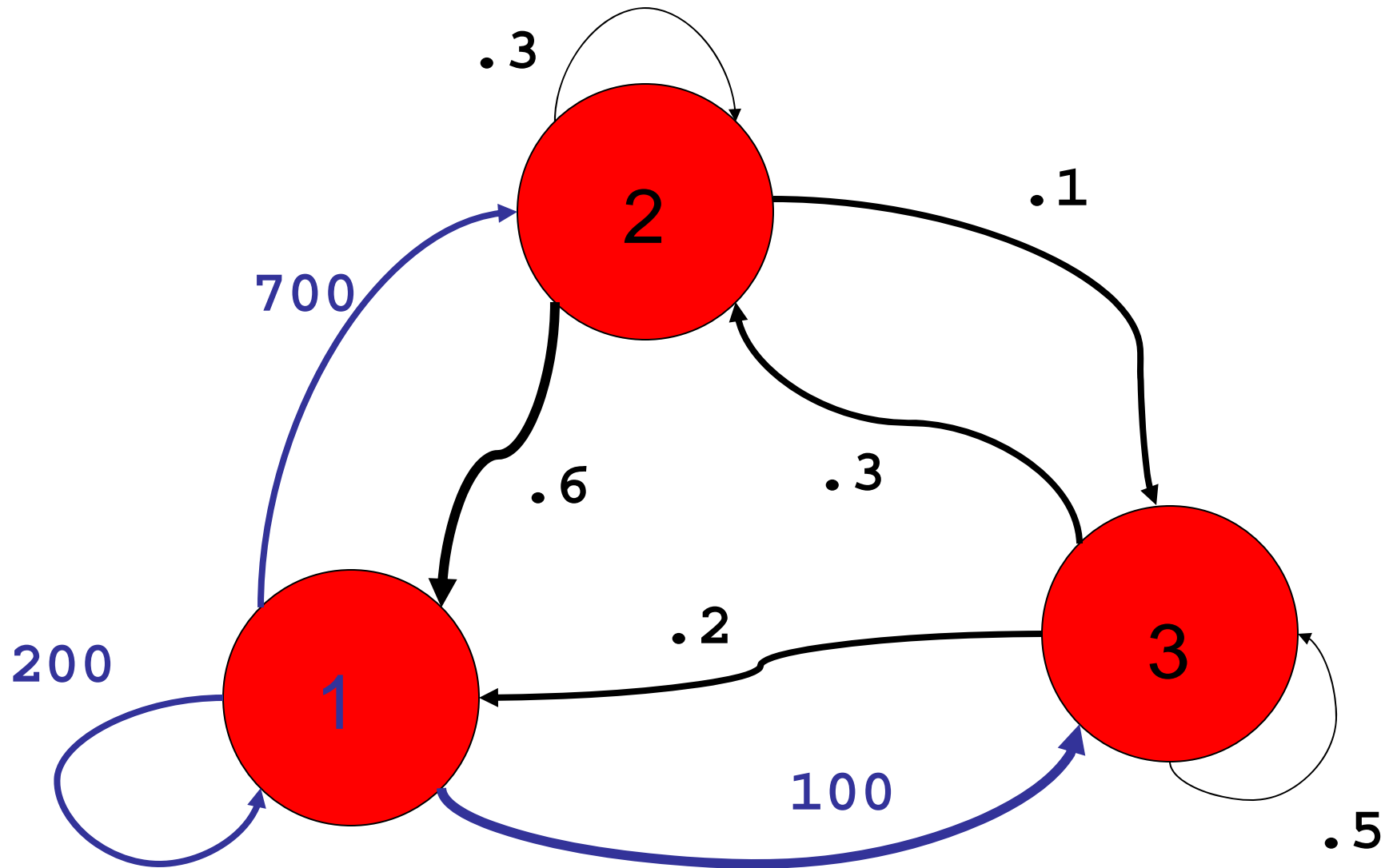
At the sound of a whistle they hop to another node in accordance with the “outbound” probabilities.

For now we assume we know these probabilities. Later we will see how to get them.

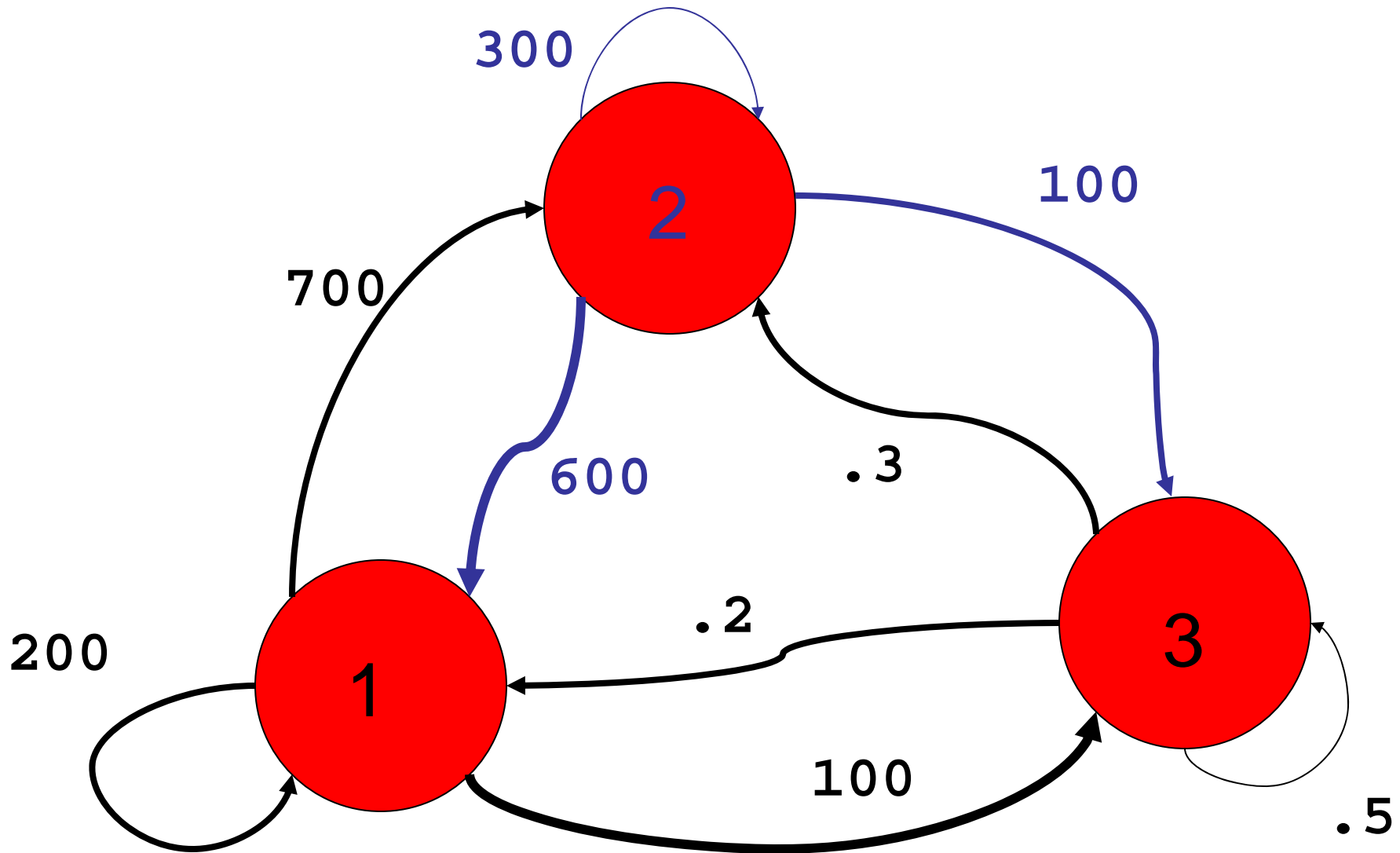
At Node 1



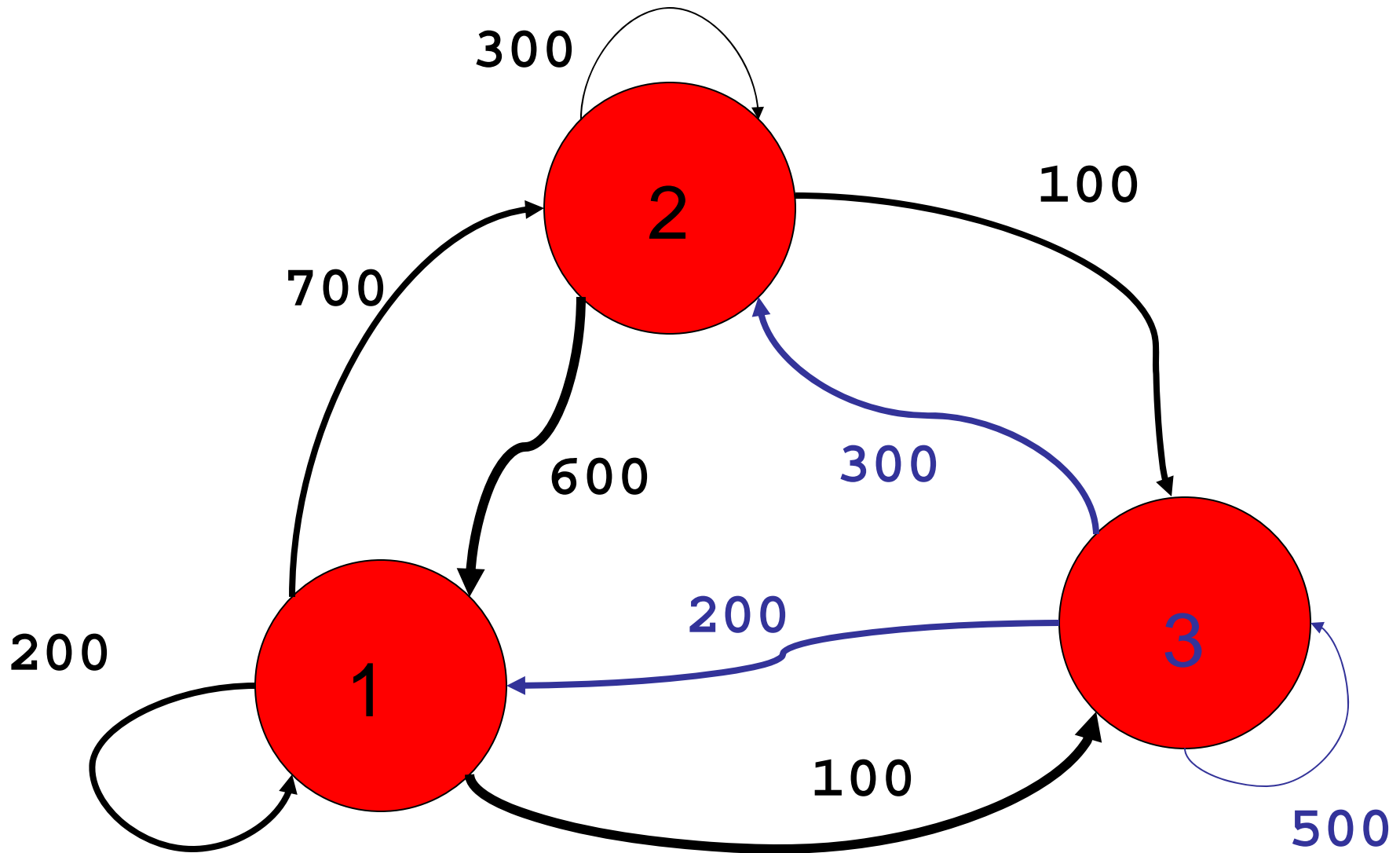
At Node 1



At Node 2



At Node 3



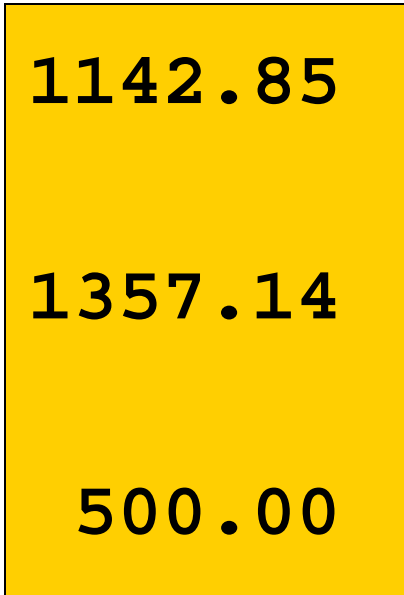
State Vector:

describes the state at each node at a specific time

$$\begin{array}{ccc} T=0 & T=1 & T=2 \\ \left(\begin{array}{c} 1000 \\ 1000 \\ 1000 \end{array} \right) & \left(\begin{array}{c} 1000 \\ 1300 \\ 700 \end{array} \right) & \left(\begin{array}{c} 1120 \\ 1300 \\ 580 \end{array} \right) \end{array}$$

After 100 iterations

	T=99	T=100
Node 1	1142.85	1142.85
Node 2	1357.14	1357.14
Node 3	500.00	500.00



Appears to reach a **steady state**

Call this the **stationary vector**