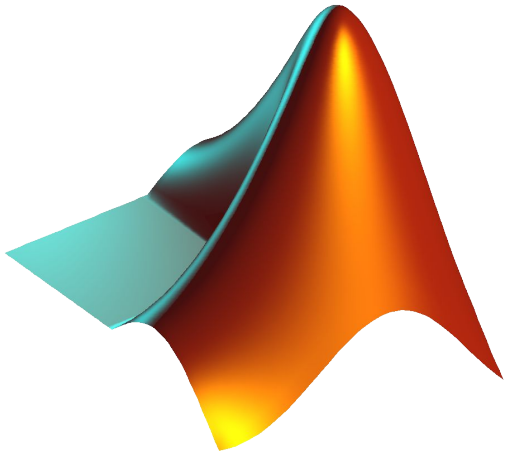
CS 1112 Introduction to Computing Using MATLAB



Instructor: Dominic Diaz

Website: https://www.cs.cornell.edu/courses/cs1112/2022fa/

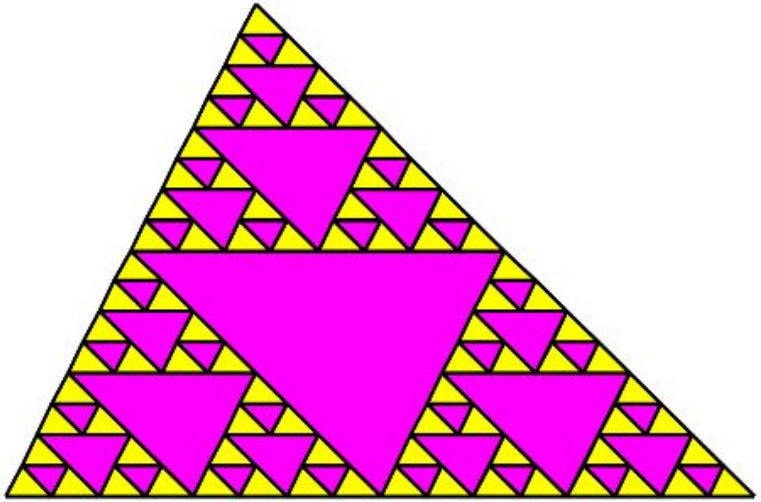
Today: Recursion

Agenda and announcements

- Last time
  - Object-oriented programming
    - Inheritance
    - Public, private, and protected access
- Today
  - Recursion
- Announcements
  - Project 6 released (due Dec 5th)
    - If you need a partner, fill out partner service by Tuesday 11/22
  - Code ‘til you drop session on Dec 14th
  - No exercise or discussion this week. Enjoy break!
  - Final page is on the website
  - All consulting hours on today 11/22 will be online!

Recursion

A method of problem solving by breaking a problem into smaller and smaller instances of the same problem until an instance is so small that it's trivial to solve



Recursion

- The Fibonacci sequence is defined recursively
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, …
  - Starting with F(1) = 1 and F(2) = 1, then F(k) = F(k-2) + F(k-1)
  - It is recursive because it is defined in terms of itself.

- Algorithms and function can be recursive as well. I.e., a function can call itself.

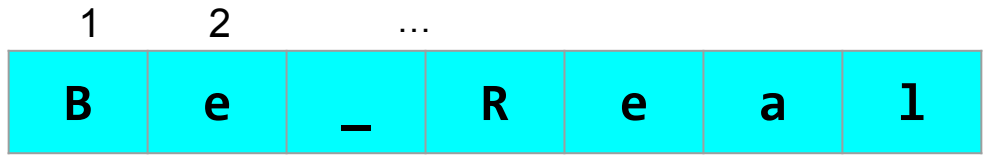
- Example: remove all occurrences of a character from a char array

  `'gc aatc gga c '` → `'gcaatcggac'`

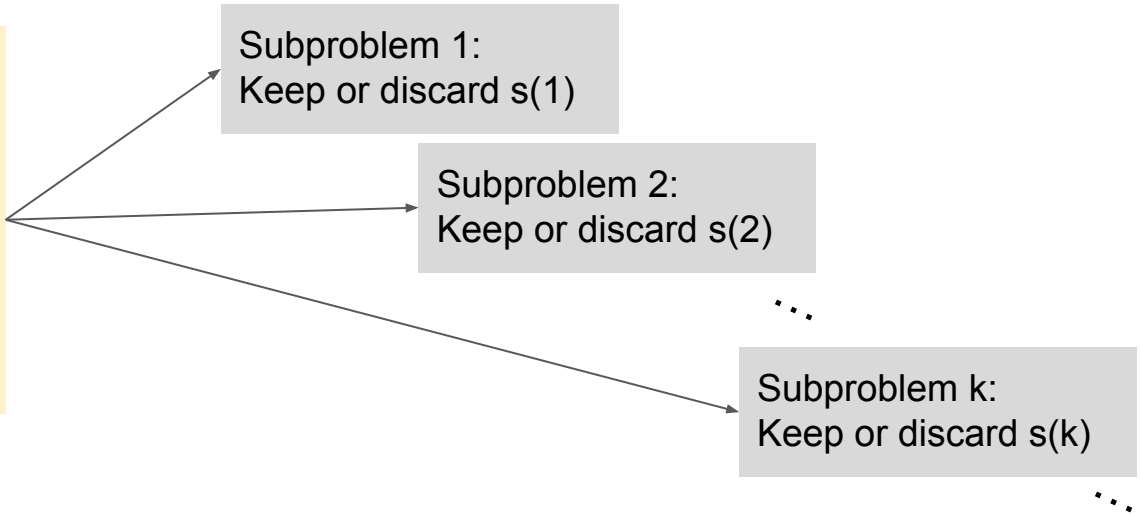
Example: removing all occurrences of a character

- Can solve this problem using iteration—check one character at a time

Character to remove: '_'

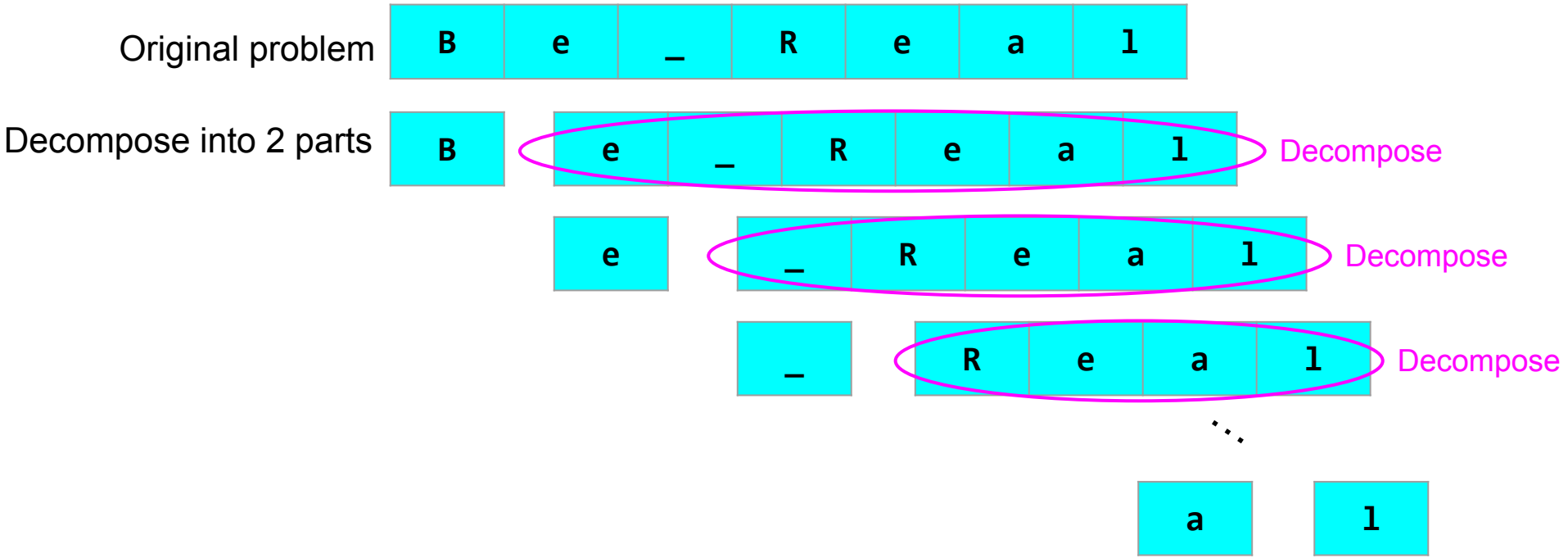


Iteration:
Divide the problem into a sequence of equal-sized, identical subproblems



Example: removing all occurrences of a character

- Can solve this problem using recursion!
  - Original problem: remove all `'_'` in char array `s`
  - Decompose problem into two parts
    - Remove the `'_'` in `s(1)` if there is one
    - Remove the `'_'`s in `s(2:end)` if there are any



```
function s = removeChar(c, s)
% Outputs char array s with all instances of character c removed

if length(s) == 0  % Base case: do nothing
    s = s;
else



end
```

removeChar('_', '')

```
function s = removeChar(c, s)
% Outputs char array s with all instances of character c removed

if length(s) == 0  % Base case: do nothing
    s = s;
else
    if s(1) ~= c

    else


    end
end
```

removeChar('_', 'd_o_g')

```
function s = removeChar(c, s)
% Outputs char array s with all instances of character c removed

if length(s) == 0  % Base case: do nothing
    s = s;
else
    if s(1) ~= c
        % output is s(1) and s(2:end) with char c removed

    else



    end
end
```

removeChar('_', 'd_o_g')

```
function s = removeChar(c, s)
% Outputs char array s with all instances of character c removed

if length(s) == 0  % Base case: do nothing
    s = s;
else
    if s(1) ~= c
        % output is s(1) and s(2:end) with char c removed

    else
        % output is s(2:end) with char c removed

    end
end
```

```
removeChar('_', '_o_g')
```

```
function s = removeChar(c, s)
% Outputs char array s with all instances of character c removed

if length(s) == 0  % Base case: do nothing
    s = s;
else
    if s(1) ~= c
        % output is s(1) and s(2:end) with char c removed
        s = [s(1) ];
    else
        % output is s(2:end) with char c removed
        
    end
end
```

```
function s = removeChar(c, s)
% Outputs char array s with all instances of character c removed

if length(s) == 0  % Base case: do nothing
    s = s;
else
    if s(1) ~= c
        % output is s(1) and s(2:end) with char c removed
        s = [s(1) removeChar(c, s(2:length(s)))];
    else
        % output is s(2:end) with char c removed
        s = removeChar(c, s(2:length(s)));
    end
end
```

removeChar('_', '')

removeChar('_', 'd_o_g')

removeChar('_', '_o_g')

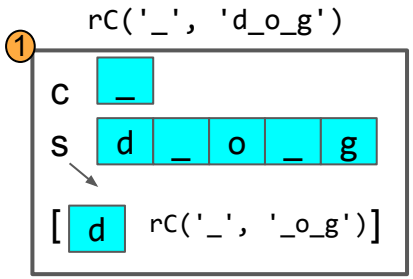
```
function s = rC(c, s)
if length(s) == 0
     s = s;
else
     if s(1) ~= c
          s = [s(1) rC(c, s(2:length(s)))];
     else
          s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

`rC('_', 'd_o_g')`

Output should be: `'dog'`

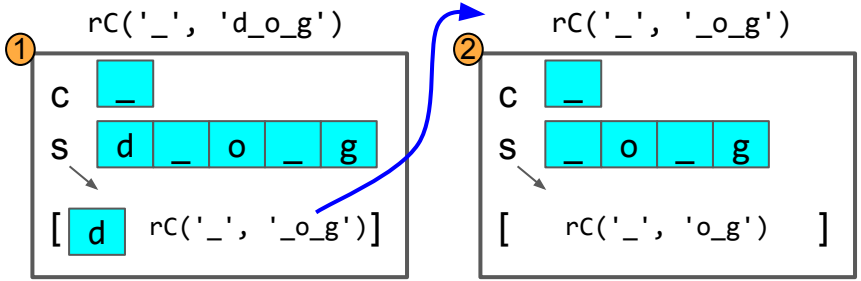
```
function s = rC(c, s)
if length(s) == 0
     s = s;
else
     if s(1) ~= c
        1 s = [s(1) rC(c, s(2:length(s)))];
     else
          s = rC(c, s(2:length(s)));
     end
end
```



Let's look at the example call:

```
rC('_', 'd_o_g')
```

```
function s = rC(c, s)
if length(s) == 0
     s = s;
else
     if s(1) ~= c
       ① s = [s(1) rC(c, s(2:length(s)))];
     else
       ② s = rC(c, s(2:length(s)));
     end
end
```



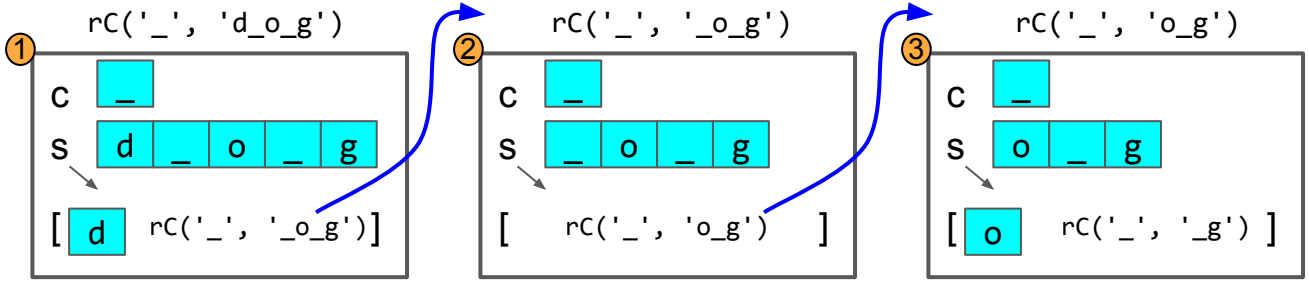
Let's look at the example call:

```
rC('_', 'd_o_g')
```

```
function s = rC(c, s)
if length(s) == 0
     s = s;
else
     if s(1) ~= c
      ③① s = [s(1) rC(c, s(2:length(s)))];
     else
       ② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

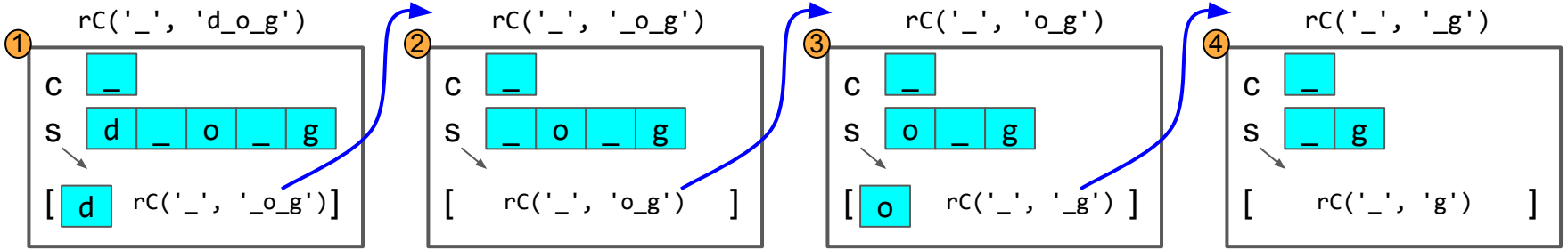
```
rC('_', 'd_o_g')
```



```
function s = rC(c, s)
if length(s) == 0
     s = s;
else
     if s(1) ~= c
      ③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

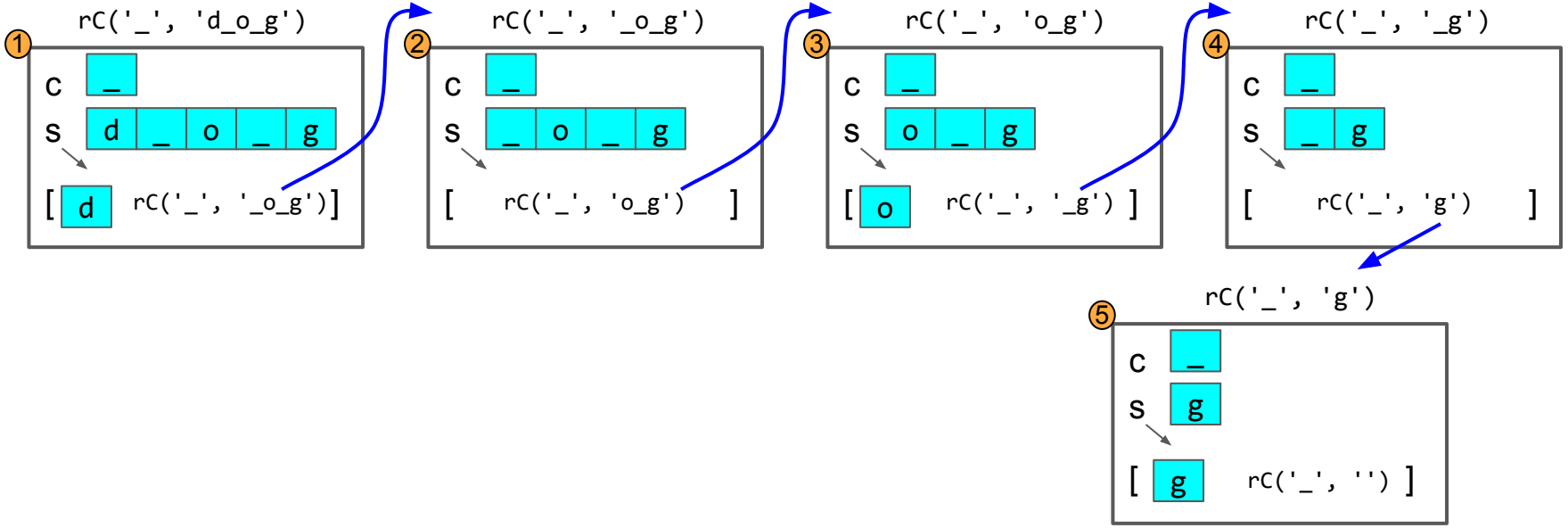
```
rC('_', 'd_o_g')
```



```
function s = rC(c, s)
if length(s) == 0
     s = s;
else
     if s(1) ~= c
    ⑤③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

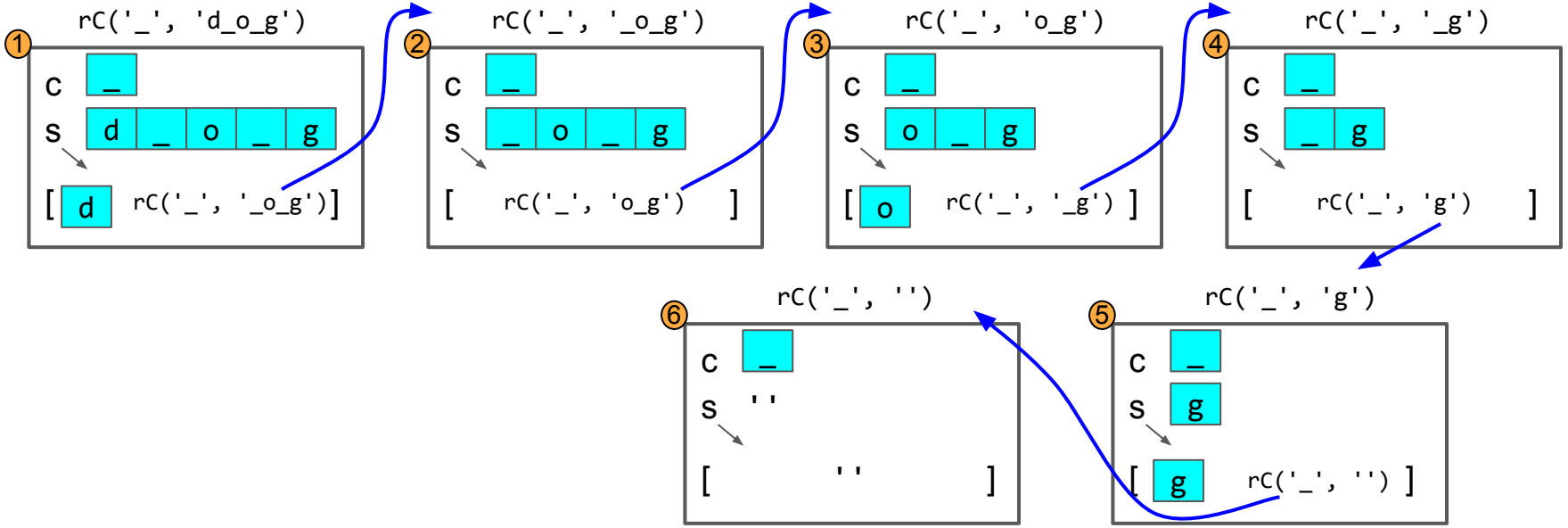
```
rC('_', 'd_o_g')
```



```
function s = rC(c, s)
if length(s) == 0
  ⑥ s = s;
else
     if s(1) ~= c
    ⑤③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

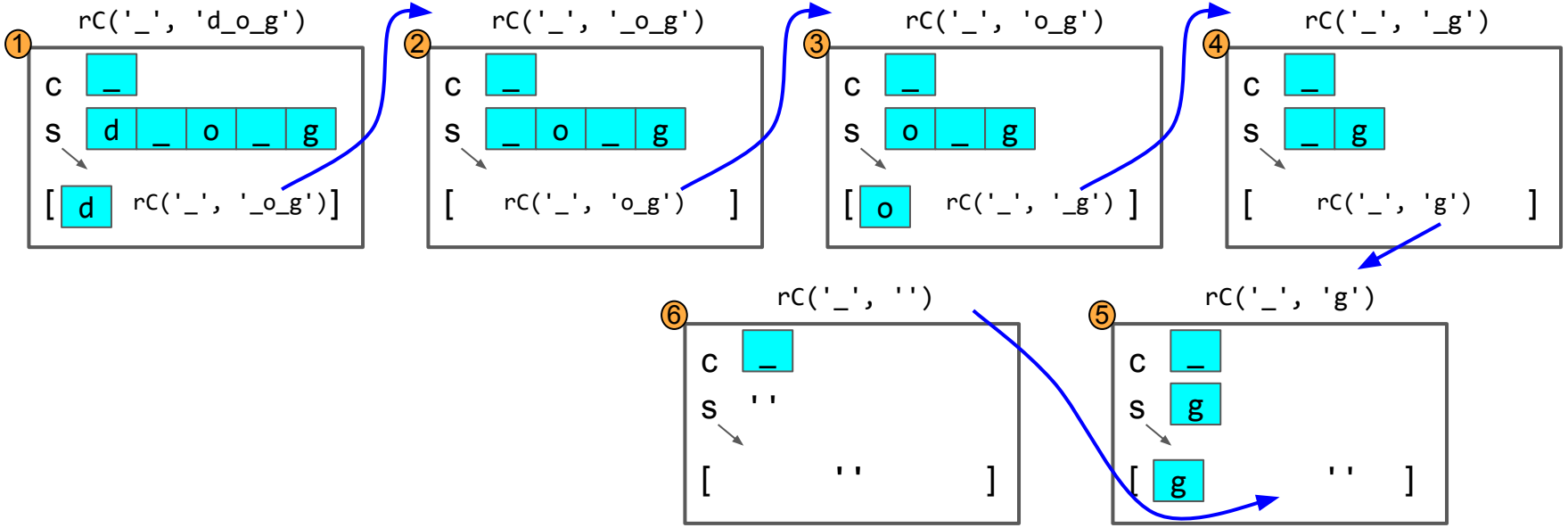
```
rC('_', 'd_o_g')
```



```
function s = rC(c, s)
if length(s) == 0
  ⑥ s = s;
else
     if s(1) ~= c
    ⑤③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

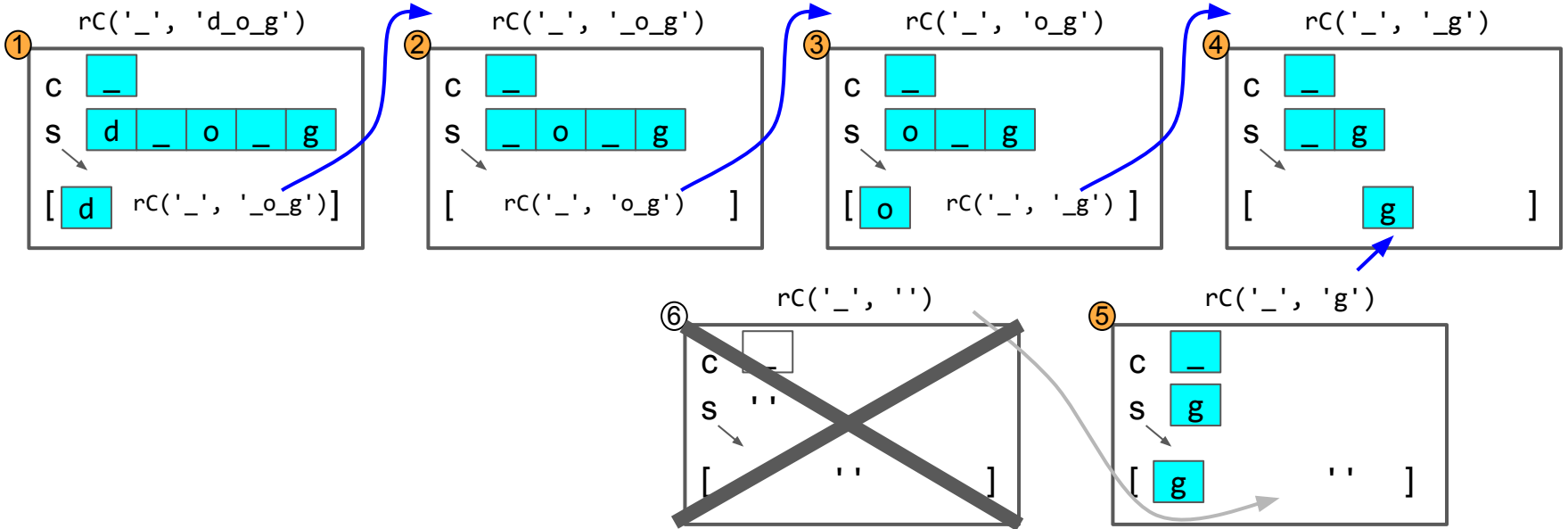
```
rC('_', 'd_o_g')
```



```
function s = rC(c, s)
if length(s) == 0
  ⑥ s = s;
else
     if s(1) ~= c
    ⑤③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

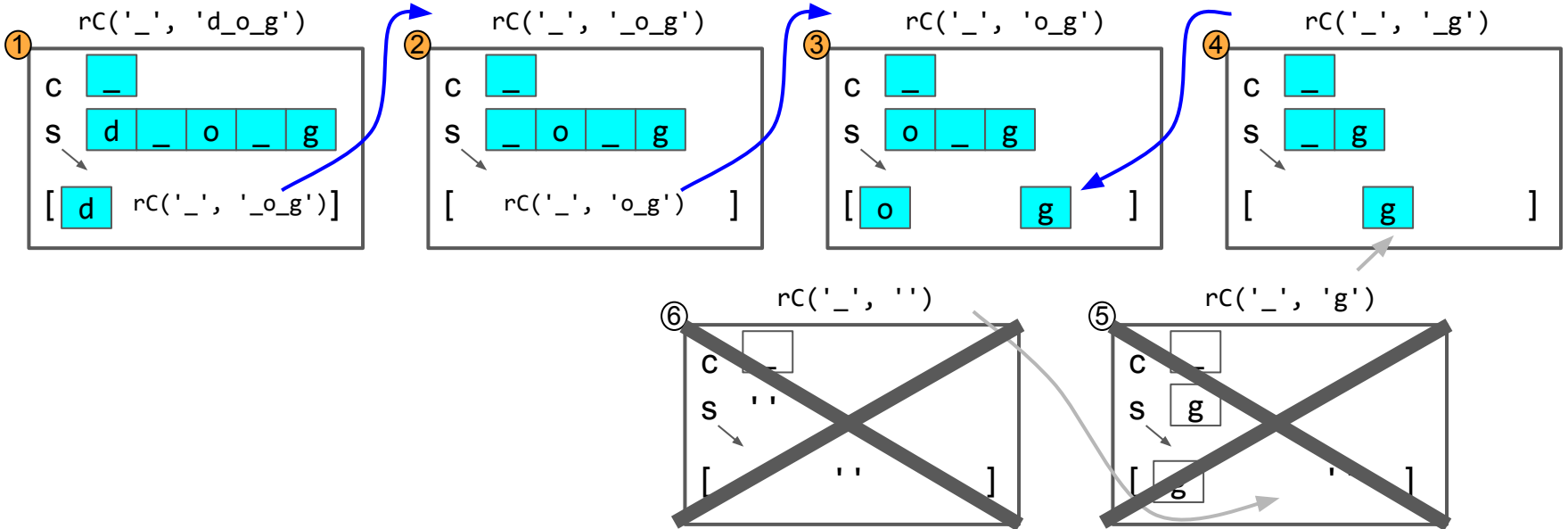
```
rC('_', 'd_o_g')
```



```
function s = rC(c, s)
if length(s) == 0
  ⑥ s = s;
else
     if s(1) ~= c
    ⑤③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

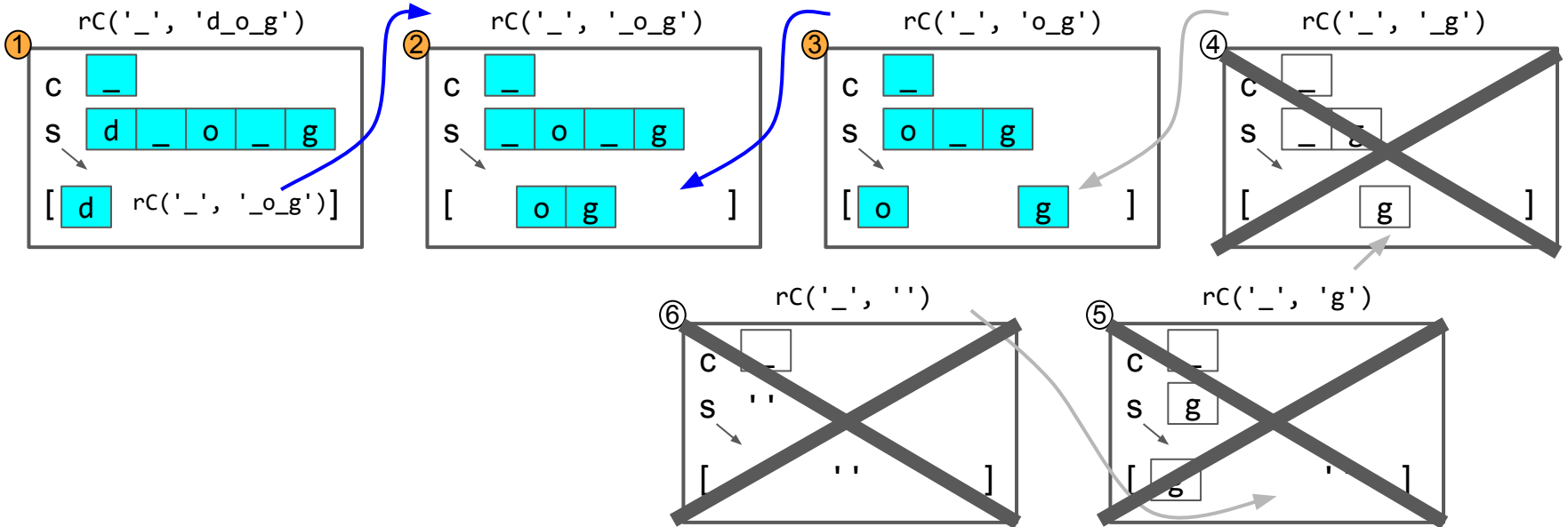
```
rC('_', 'd_o_g')
```



```
function s = rC(c, s)
if length(s) == 0
  ⑥ s = s;
else
     if s(1) ~= c
    ⑤③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

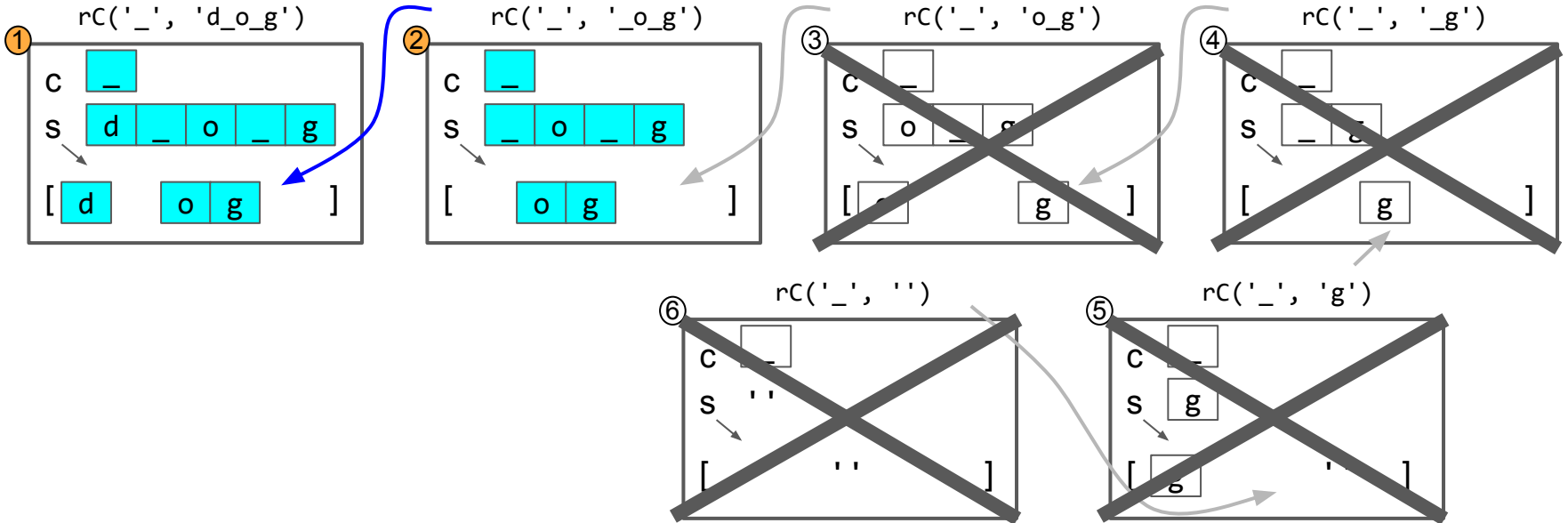
```
rC('_', 'd_o_g')
```



```
function s = rC(c, s)
if length(s) == 0
  ⑥ s = s;
else
     if s(1) ~= c
    ⑤③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

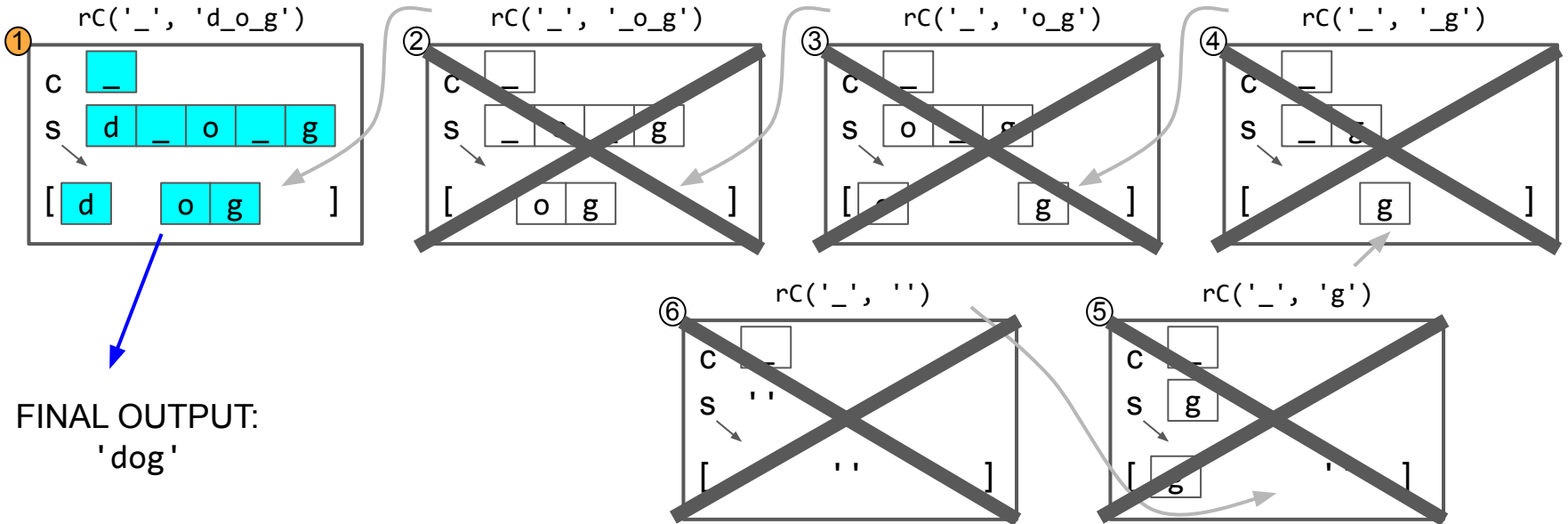
```
rC('_', 'd_o_g')
```



```
function s = rC(c, s)
if length(s) == 0
  ⑥ s = s;
else
     if s(1) ~= c
    ⑤③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

```
rC('_', 'd_o_g')
```

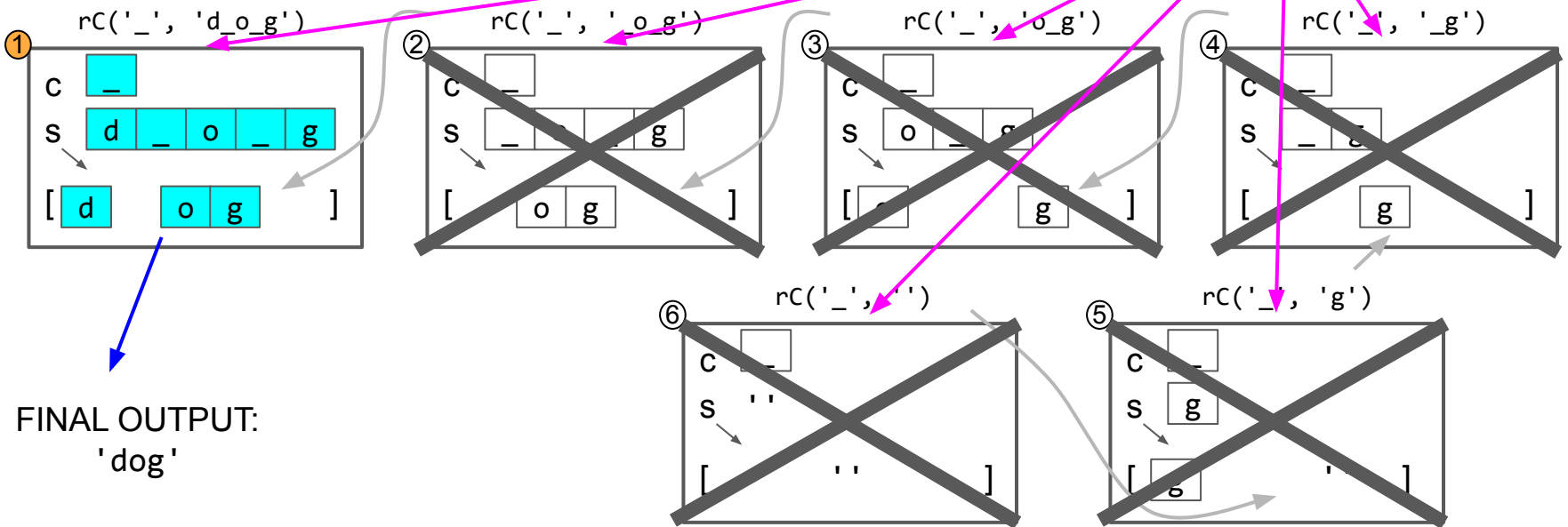


```
function s = rC(c, s)
if length(s) == 0
  ⑥ s = s;
else
     if s(1) ~= c
    ⑤③① s = [s(1) rC(c, s(2:length(s)))];
     else
      ④② s = rC(c, s(2:length(s)));
     end
end
```

Let's look at the example call:

`rC('_', 'd_o_g')`

Note: the maximum number of call frames we had open at any point was 6.



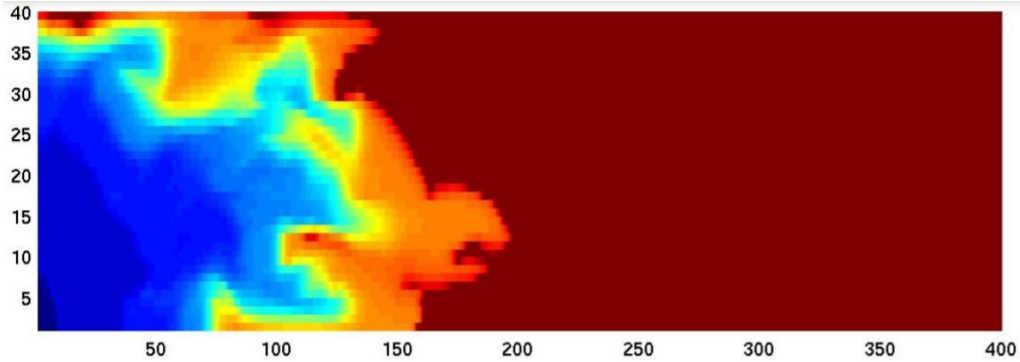
- A recursive function calls itself on progressively smaller inputs until it reaches a base case:
  - `rC('_', 'd_o_g')` called `rC('_', '_o_g')`
  - `rC('_', '_o_g')` called `rC('_', 'o_g')`
  - `rC('_', 'o_g')` called `rC('_', '_g')`
  - `rC('_', '_g')` called `rC('_', 'g')`
  - `rC('_', 'g')` called `rC('_', '')`
  - `rC('_', '')` is the base case

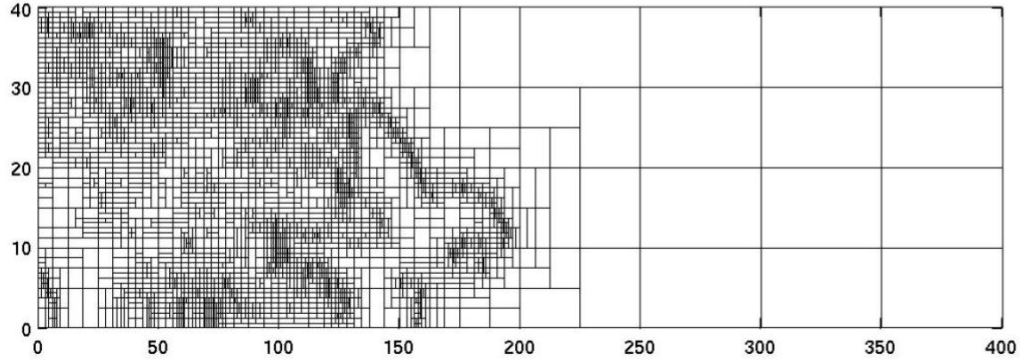
Keys to recursion

Must identify (at least one) base case, the trivially simple case
- No recursion is done in this case

The recursive case(s) must reflect progress towards the base case!

Application of recursion: mesh refinement





Nilsson, Gerritsen, Younis 2004

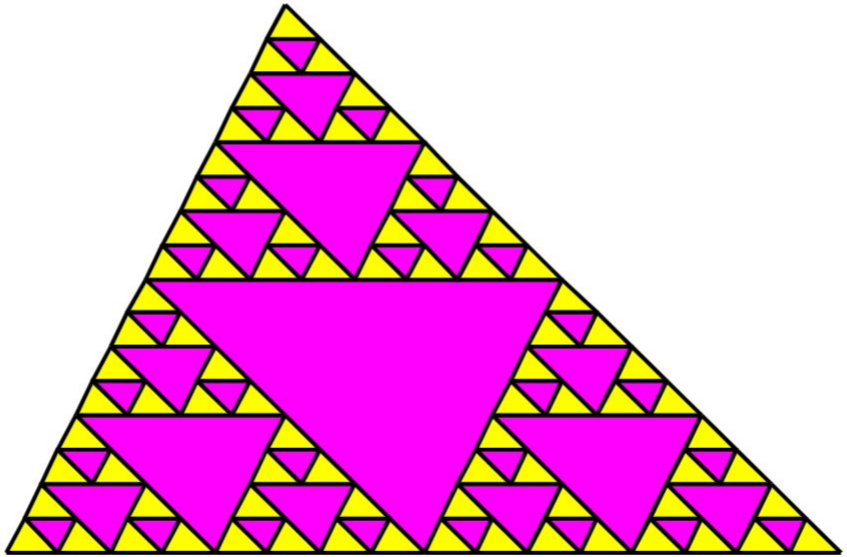
When physics is too complicated for one big region, divide it into two smaller regions.

- Subproblem: solve physics inside one region
- Division: split region in half
- Base case: solution looks smooth in entire region

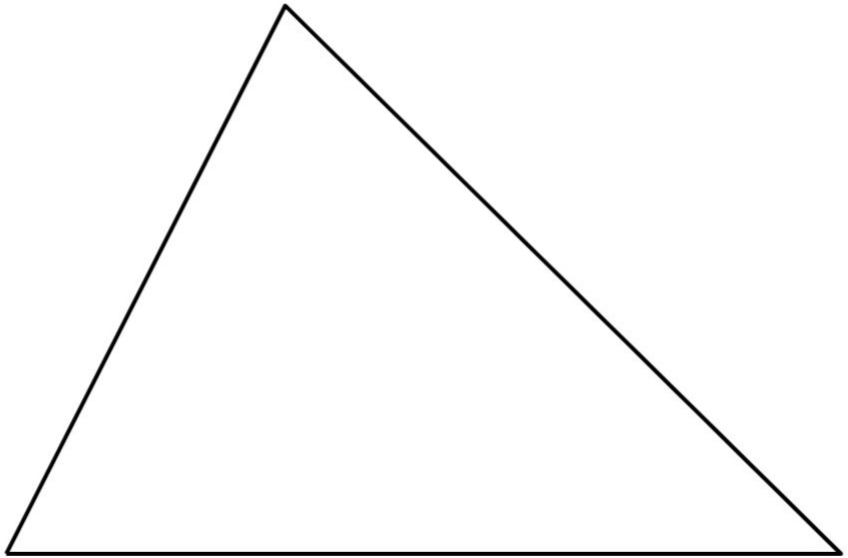
We would call this divide-and-conquer

Why is mess generation a divide-and-conquer process?

Let’s draw this graphic

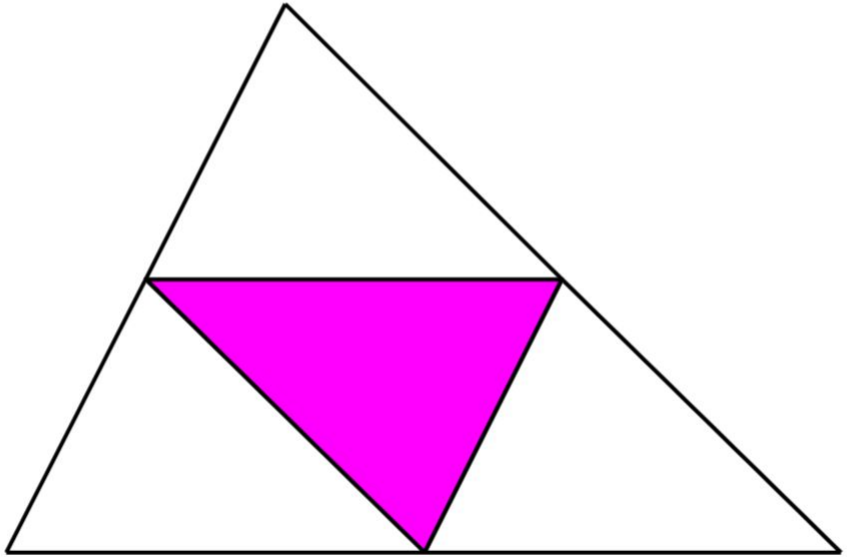


Start with a triangle



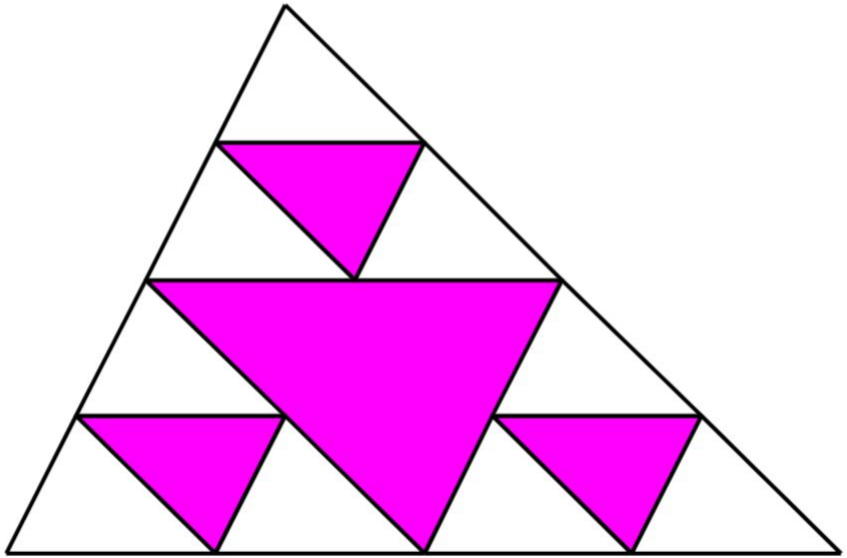
A “level 1” partition of the triangle

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

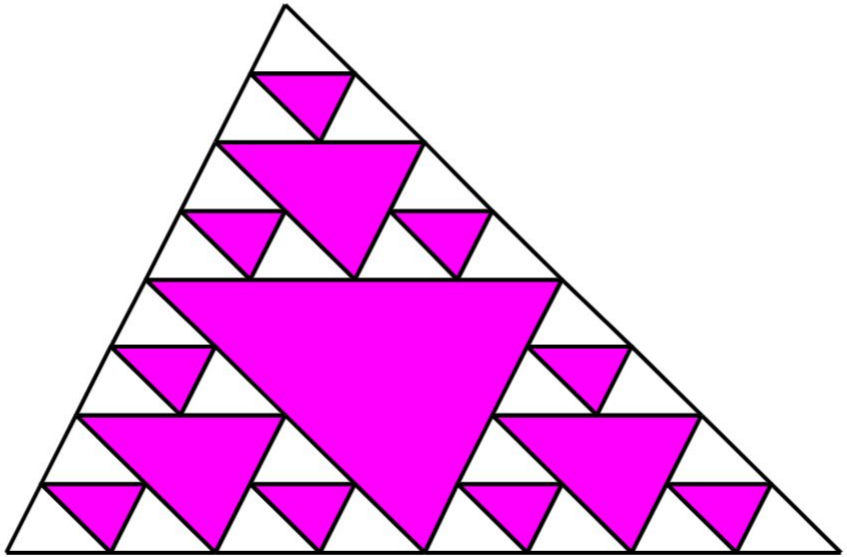


Now do the same partitioning (connecting midpoints) to each corner (white) triangle to obtain the “level 2” partitioning.

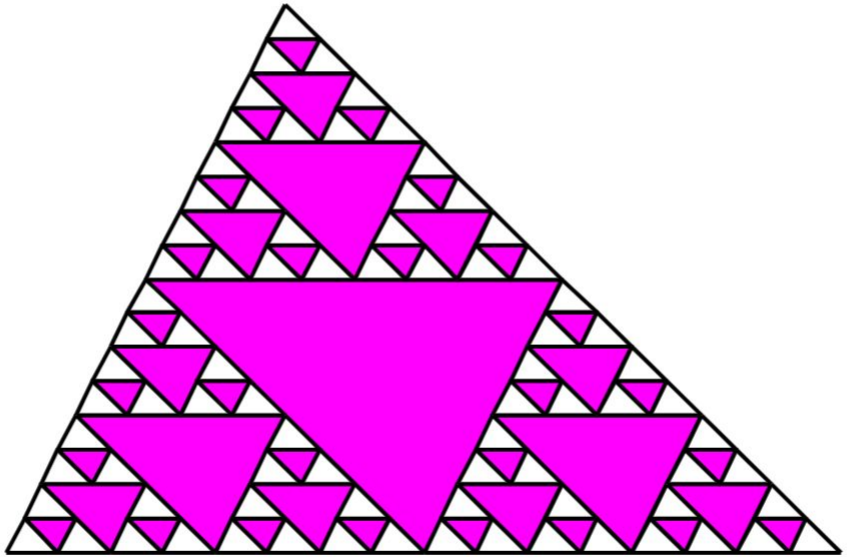
The “level 2 partition of the triangle



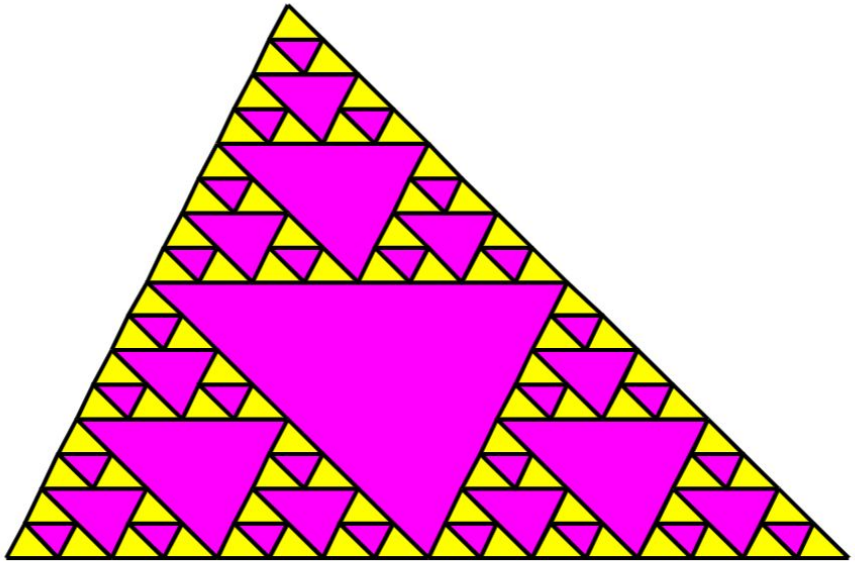
The level 3 partition of the triangle



The “level 4” partition of the triangle



If we are drawing a maximum of 4 levels, stop after the 4th level and color in the remaining triangles yellow



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.
    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
```