

(+ generación de kits)

Search more testing.

# Algoritmos Genéticos

Ingeniería del Software 2

Juan P. Galeotti



DEPARTAMENTO  
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA



Instituto de Ciencias  
de la Computación

# Repaso: Random Testing



- Es una búsqueda (casi) completamente no guiada
- Si la técnica sistemática no es mejor que random testing, entonces no es valiosa
- Barata & fácil de implementar
- Funciona bastante bien en muchos casos

# Repasso: Concolic Testing

- Ejecuta concretamente el test pero guarda la path condition
- Utiliza un constraint solver para crear nuevos inputs



# Limitaciones - Ejemplo

```
def testMe(x, y):  
    if x == 2 * (y + 1):  
        return True  
    else:  
        return False
```

probabilidad

- **Random Testing:** poca posibilidad de alcanzar la rama “True”
- **Concolic Testing:** la rama “True” no es alcanzable si el constraint solver no soporta aritmética no-lineal

dependencia del constraint solver  
(y en dominio)

# Limitaciones

- Random Testing:



- Dificultad en generar inputs que alcancen código poco probable (distribución uniforme)

- Concolic Testing:

- Tamaño de la path condition
- Capacidad del Constraint Solver



# SB Testing E Search-Based Software Engineering

- Transformar los problemas de la Ingeniería de Software en problemas de optimización
- Los espacios de búsqueda en Ingeniería del Software son **GRANDES**
- Aplicar algoritmos de búsqueda meta-heurísticos para resolver estos problemas

Optimización Colaborativa

# Heurísticas

- Pueden no siempre encontrar la mejor solución
  - Pero encuentran una buena solución en una cantidad razonable de tiempo
  - Sacrifican **completitud**, pero ganan **eficiencia**  
*(mejor solución posible)*
  - Útiles en resolver problemas difíciles:
    - Que no pueden ser resueltos por ningún otro
    - Que tomarían mucho tiempo en ser computados

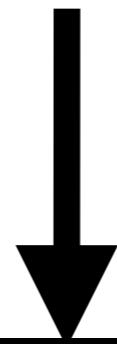
# Random Testing

---

Input  
Interesante

que nos interesan

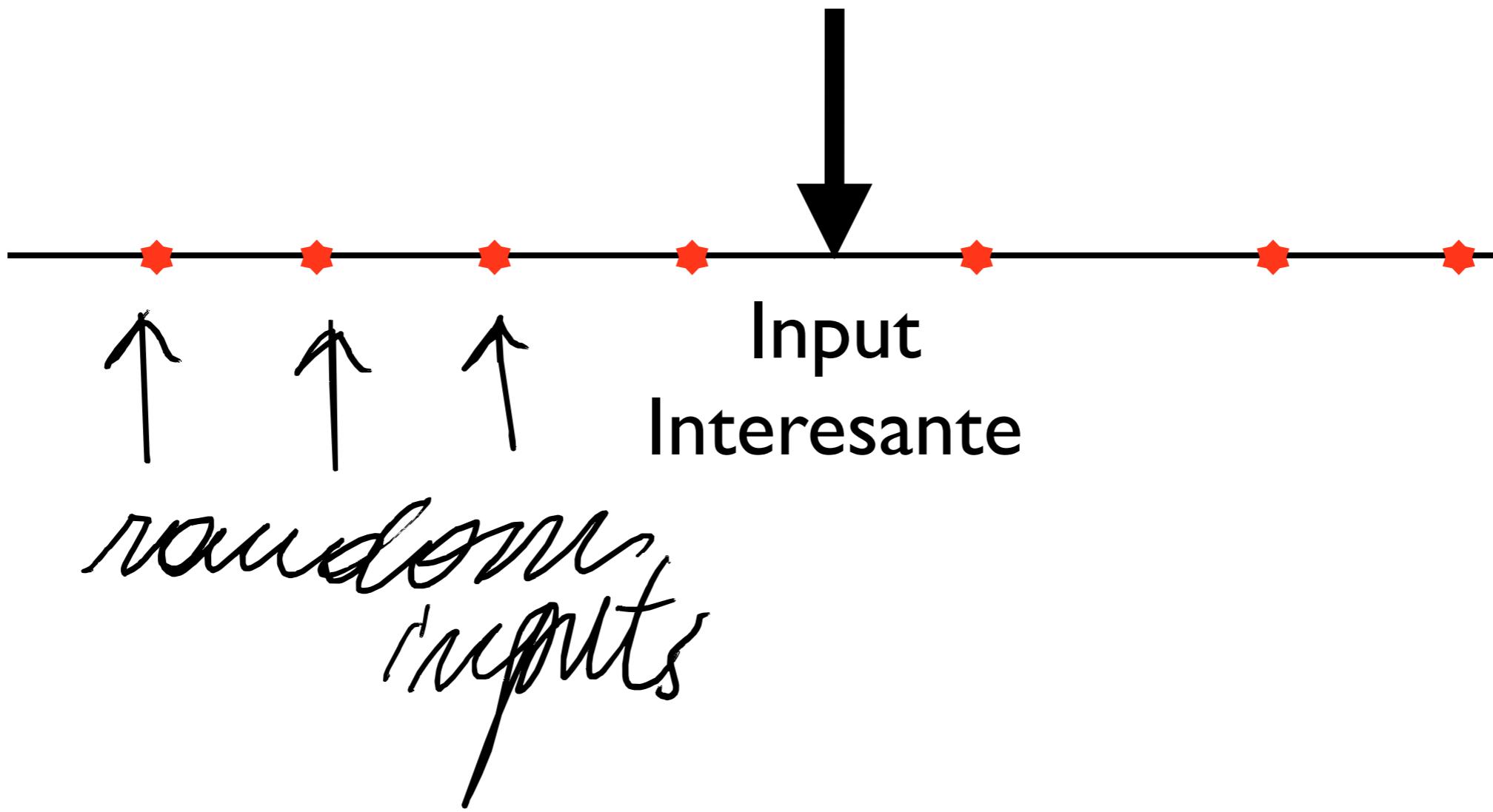
# Random Testing



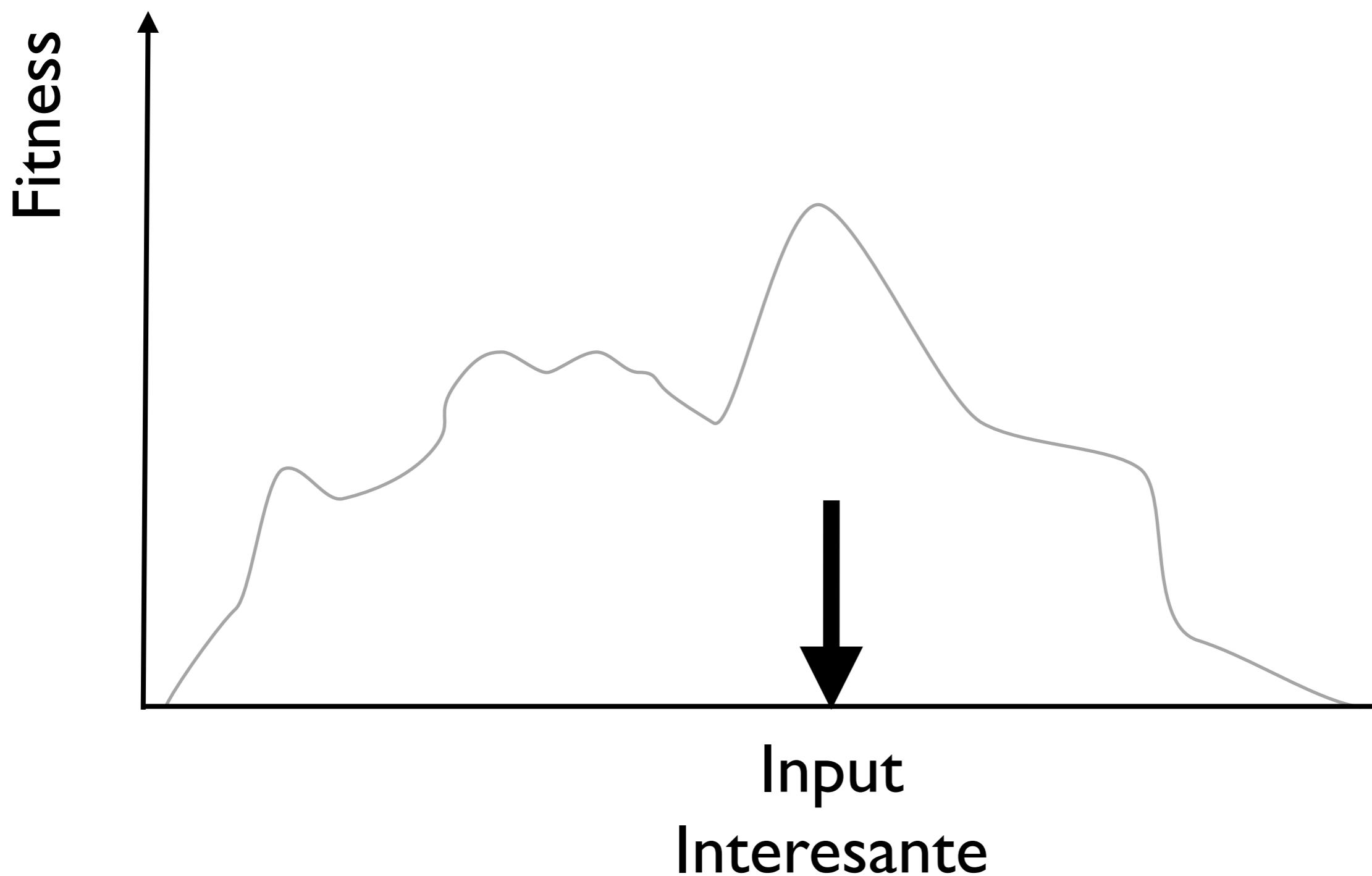
---

Input  
Interesante

# Random Testing

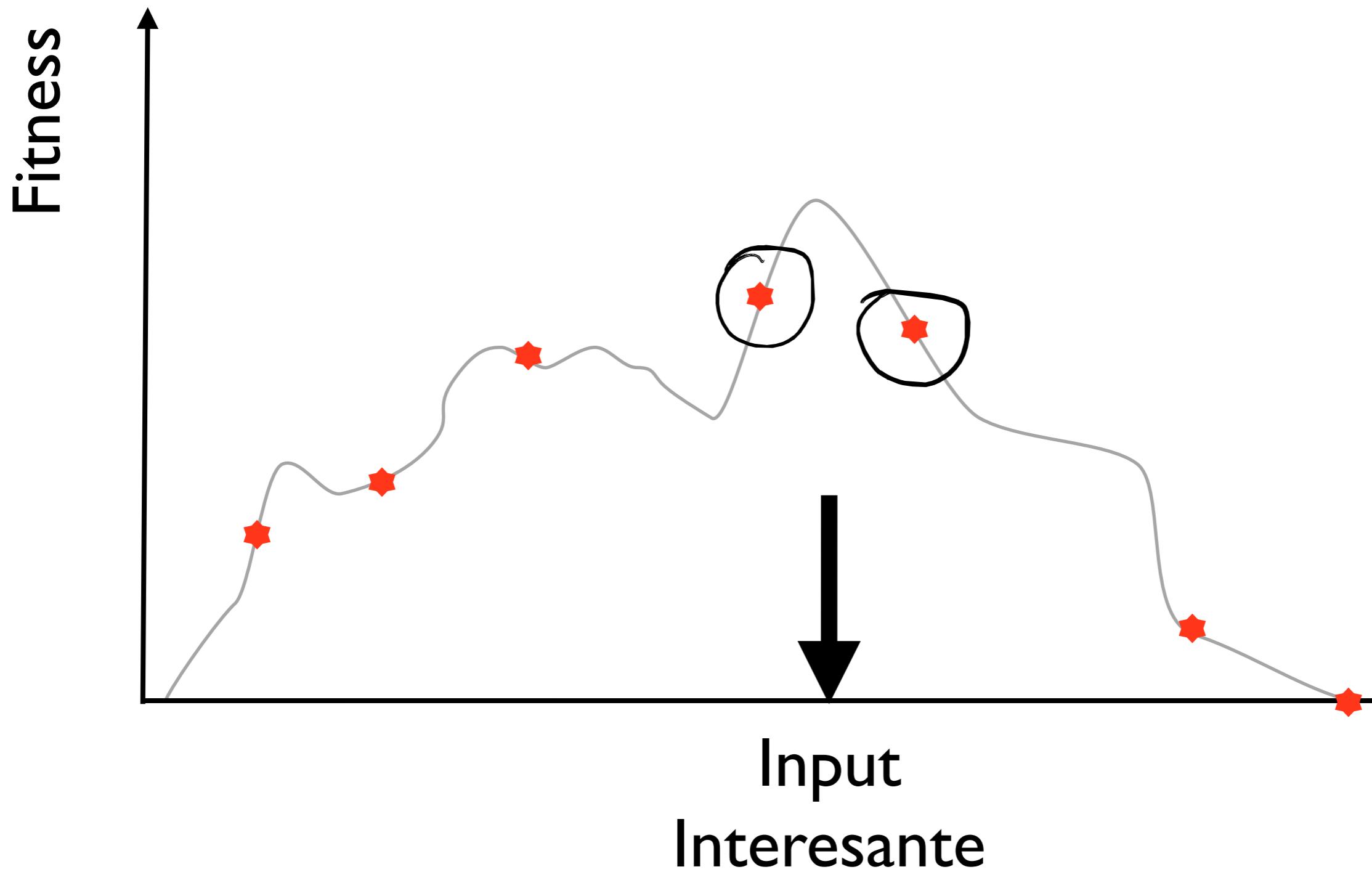


# Search-Based Testing

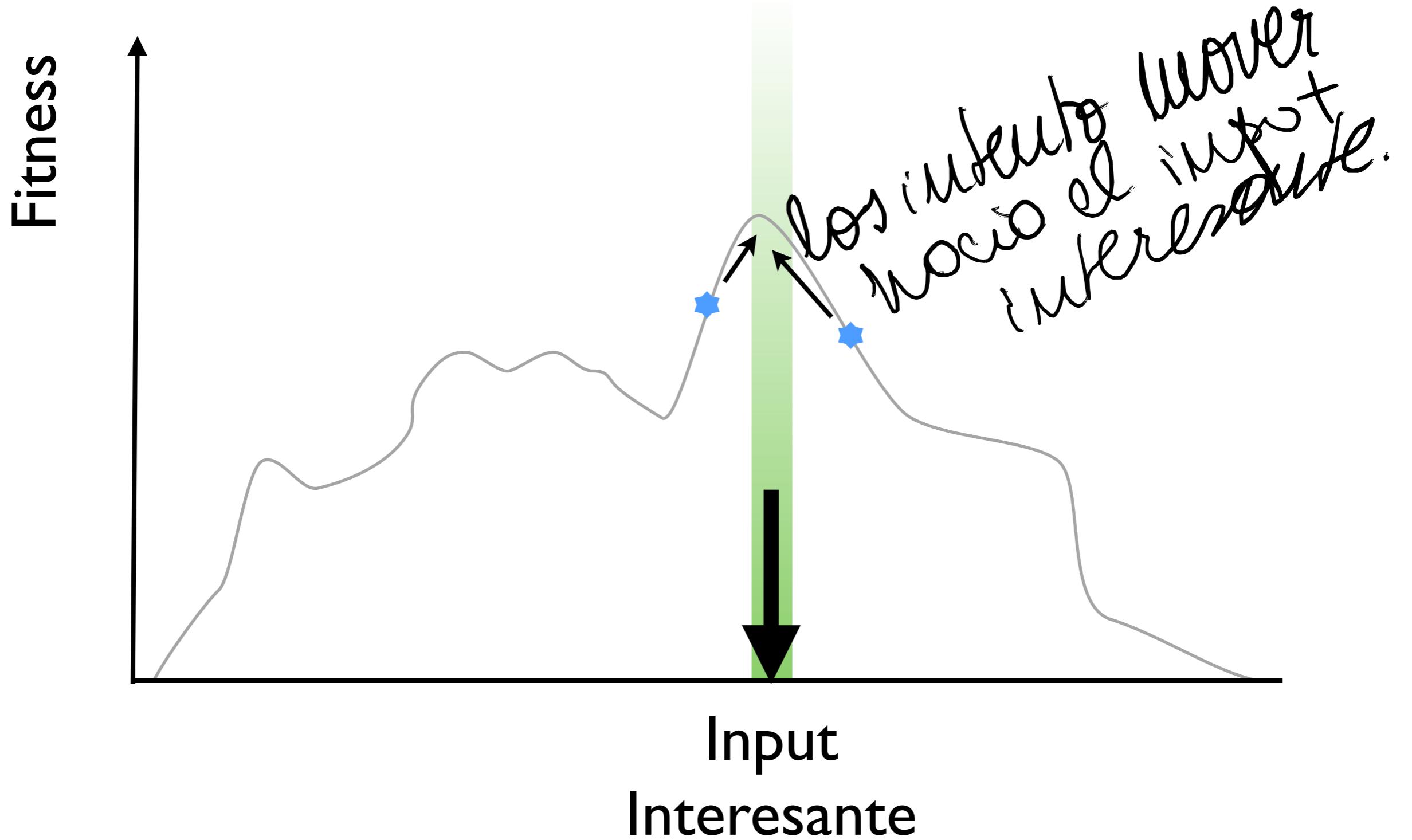


que han llevado a mi test  
el selector de inputs interesantes.

# Search-Based Testing

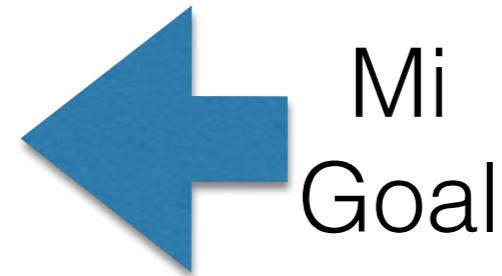


# Search-Based Testing



# Random Testing

```
def testMe(x, y):  
    if x == y:  
        return True  
    else:  
        return False
```

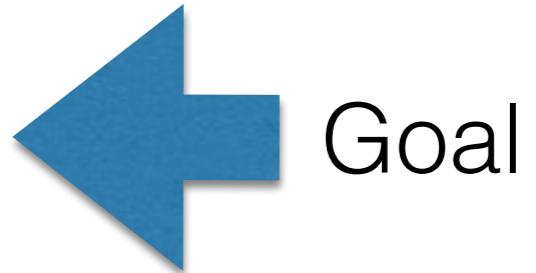


- ¿Qué podemos hacer para hacer random testing sobre este valor hasta saber si llegamos al goal?

# Random Testing

SUT

```
def testMe(x, y):  
    if x == y:  
        return True  
    else:  
        return False
```



Goal

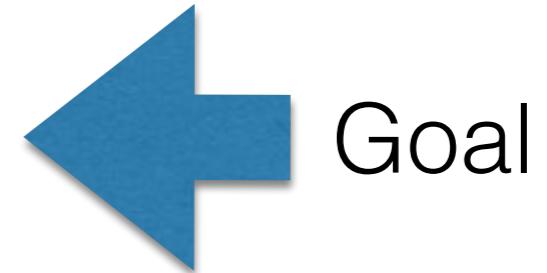
Test Driver

```
MAX = sys.maxsize  
MIN = -sys.maxsize -1  
  
ret_val = False  
while ret_val!=True:  
    x = random.randint(MIN, MAX)  
    y = random.randint(MIN, MAX)  
    ret_val = testMe(x, y)
```

# Random Testing

SUT

```
def testMe(x, y):  
    if x == y:  
        return True  
    else:  
        return False
```



Goal

```
MAX = sys.maxsize  
MIN = -sys.maxsize -1
```

Test Driver

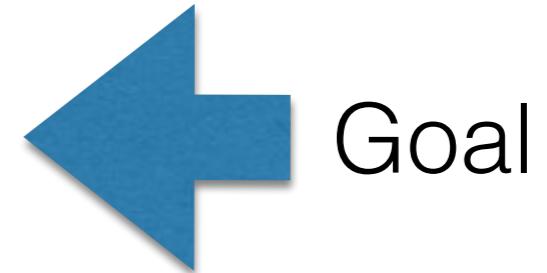
```
ret_val = False  
while ret_val!=True:  
    x = random.randint(MIN,MAX)  
    y = random.randint(MIN,MAX)  
    ret_val = testMe(x,y)
```

- ¿Cuál es la probabilidad que encontramos un valor que retorne **True**?

# Random Testing

SUT

```
def testMe(x, y):  
    if x == y:  
        return True  
    else:  
        return False
```



Goal

```
MAX = sys.maxsize  
MIN = -sys.maxsize -1
```

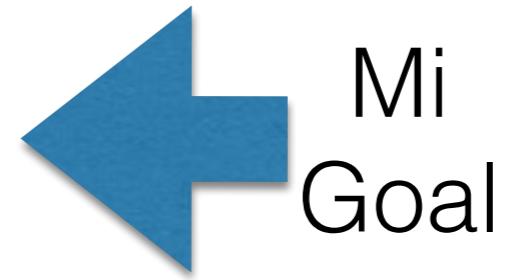
Test Driver

```
ret_val = False  
while ret_val!=True:  
    x = random.randint(MIN,MAX)  
    y = random.randint(MIN,MAX)  
    ret_val = testMe(x,y)
```

- ¿Cuál es la probabilidad que encontramos un valor que retorne **True**?

$2^{-32}$

```
def testMe(x, y):  
    if x == y:  
        return True  
else:  
    return False
```



- Si random no funciona ¿Cómo podemos usar una táctica “search-based” para cubrir el goal?



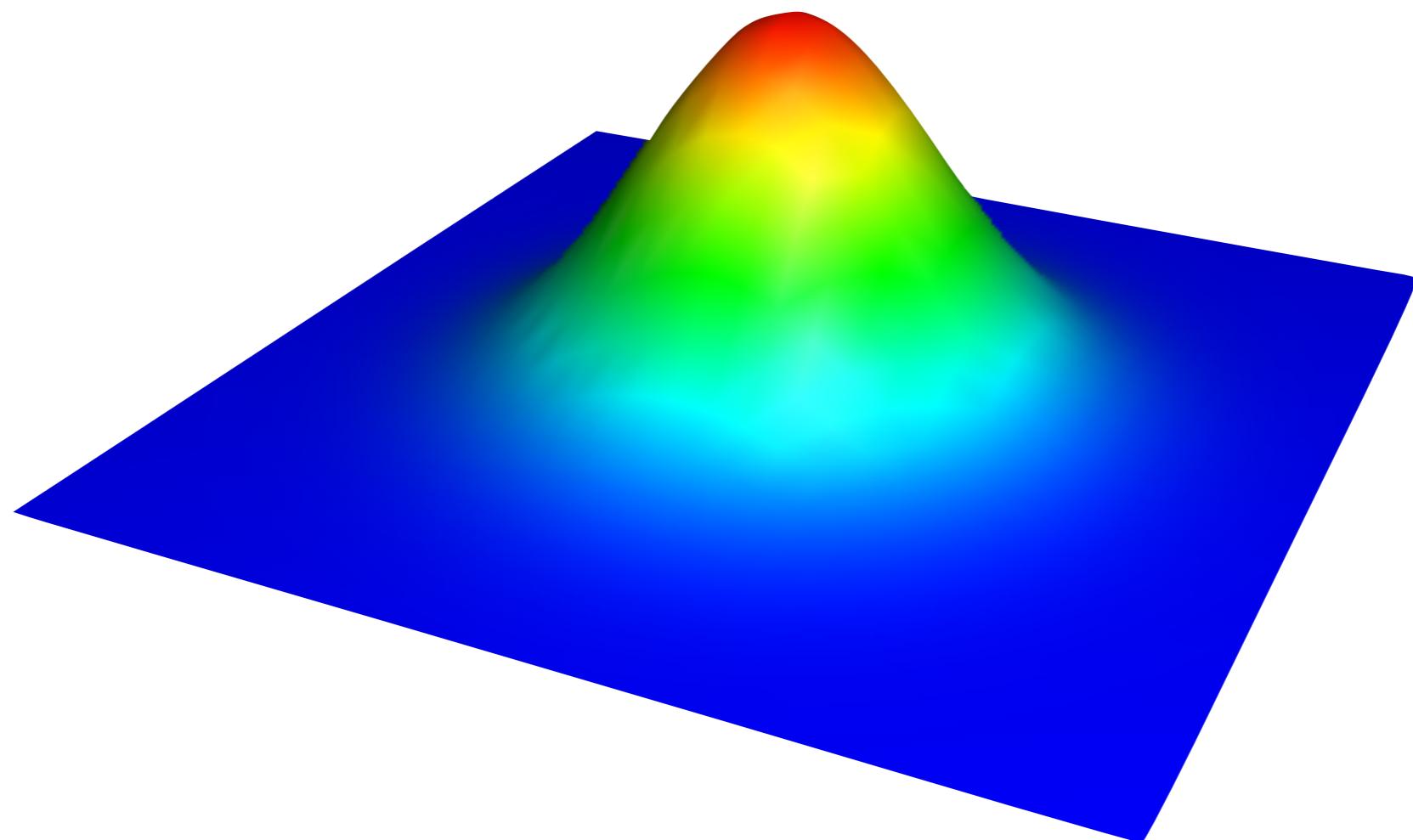
Gradient Descent / Hill Climbing

## Idea

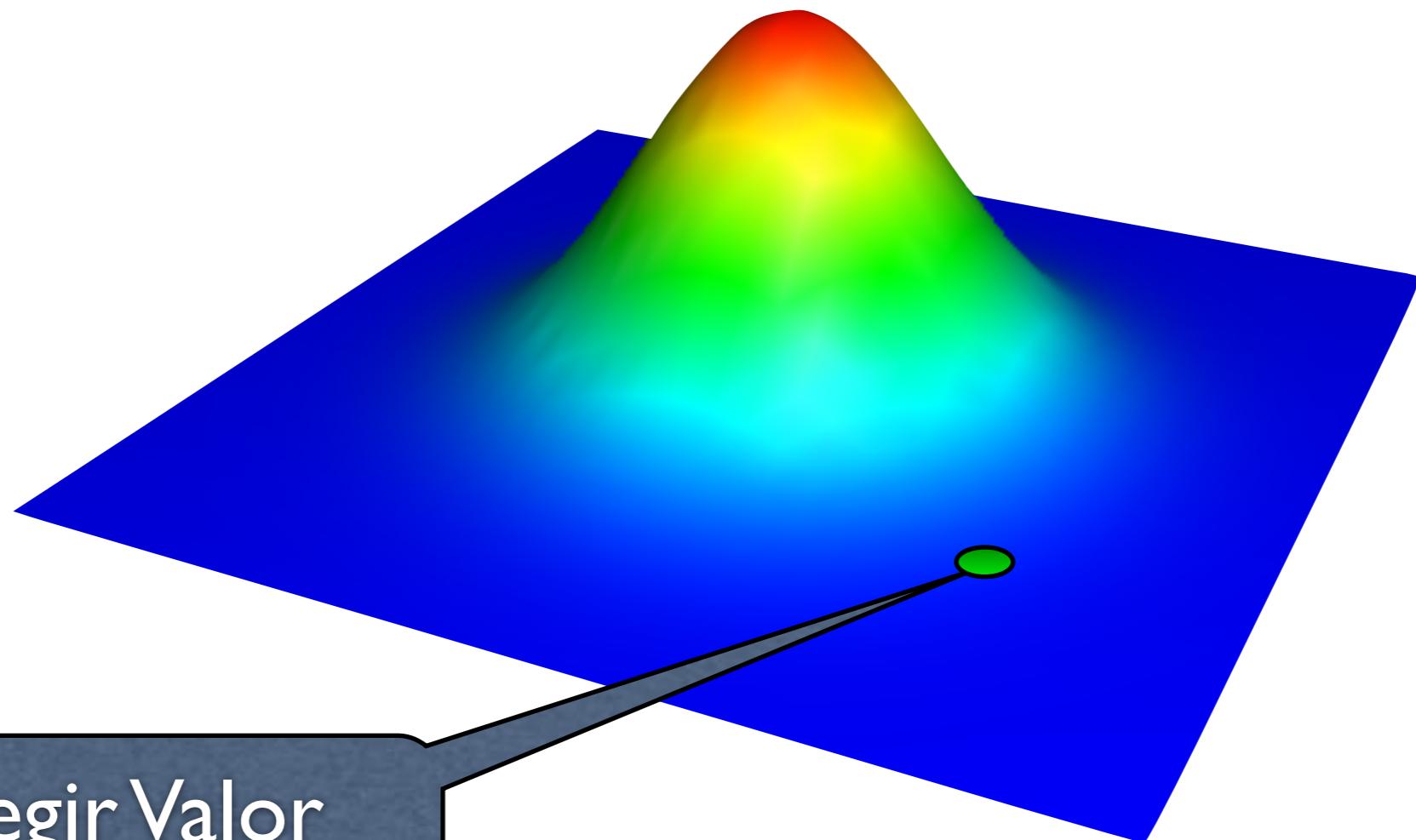
1. Definir una noción de “**cercanía**” del input con respecto al **goal** que queremos alcanzar
2. Por cada nueva solución, explorar el “vecindario” de todas las soluciones
3. Elegir la solución del vecindario que sea mejor que la actual (*Precedy Algoritmu*)
4. Repetir hasta no poder encontrar mejoras

Criterio de parada = PRESUPUESTO

# Hill Climbing

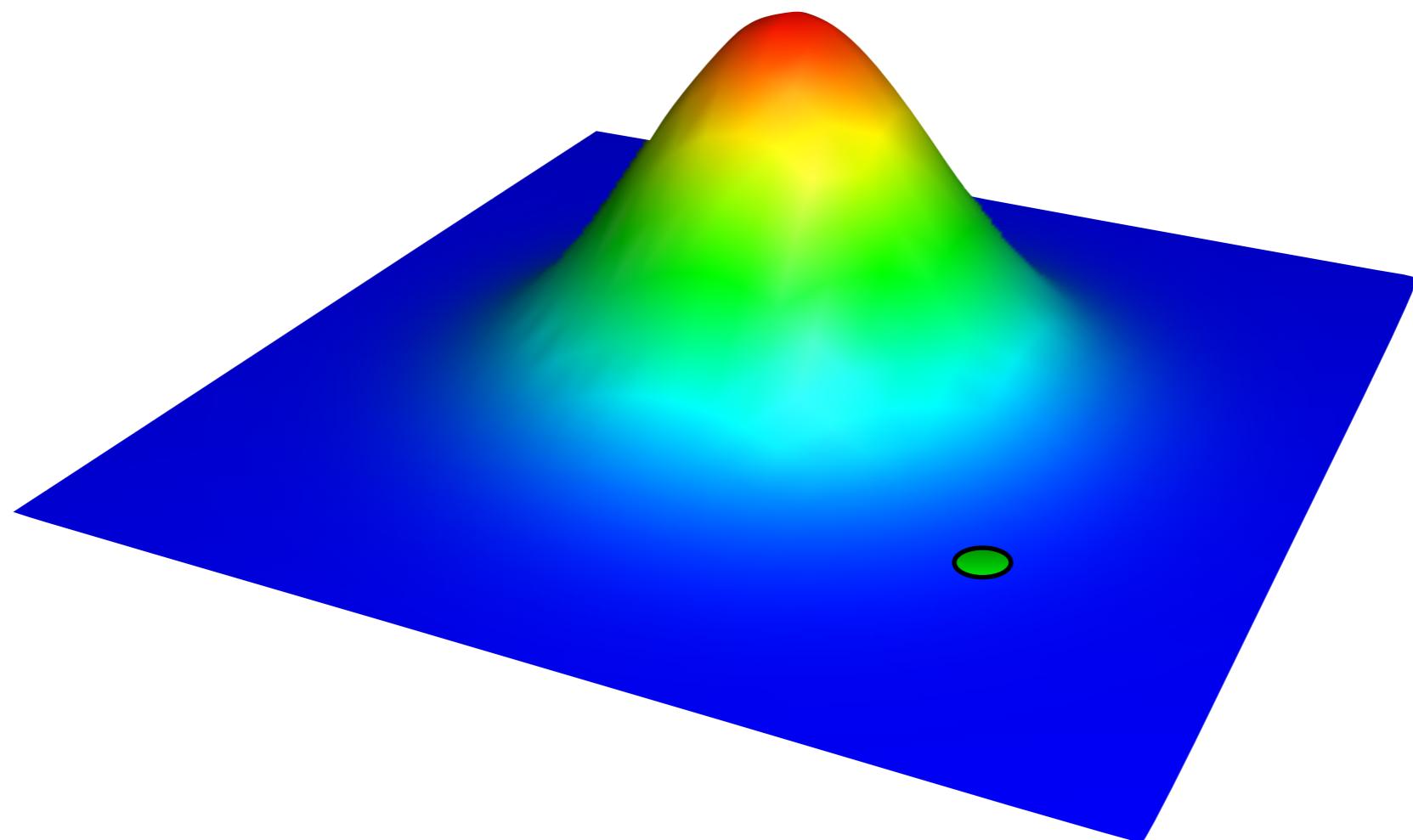


# Hill Climbing

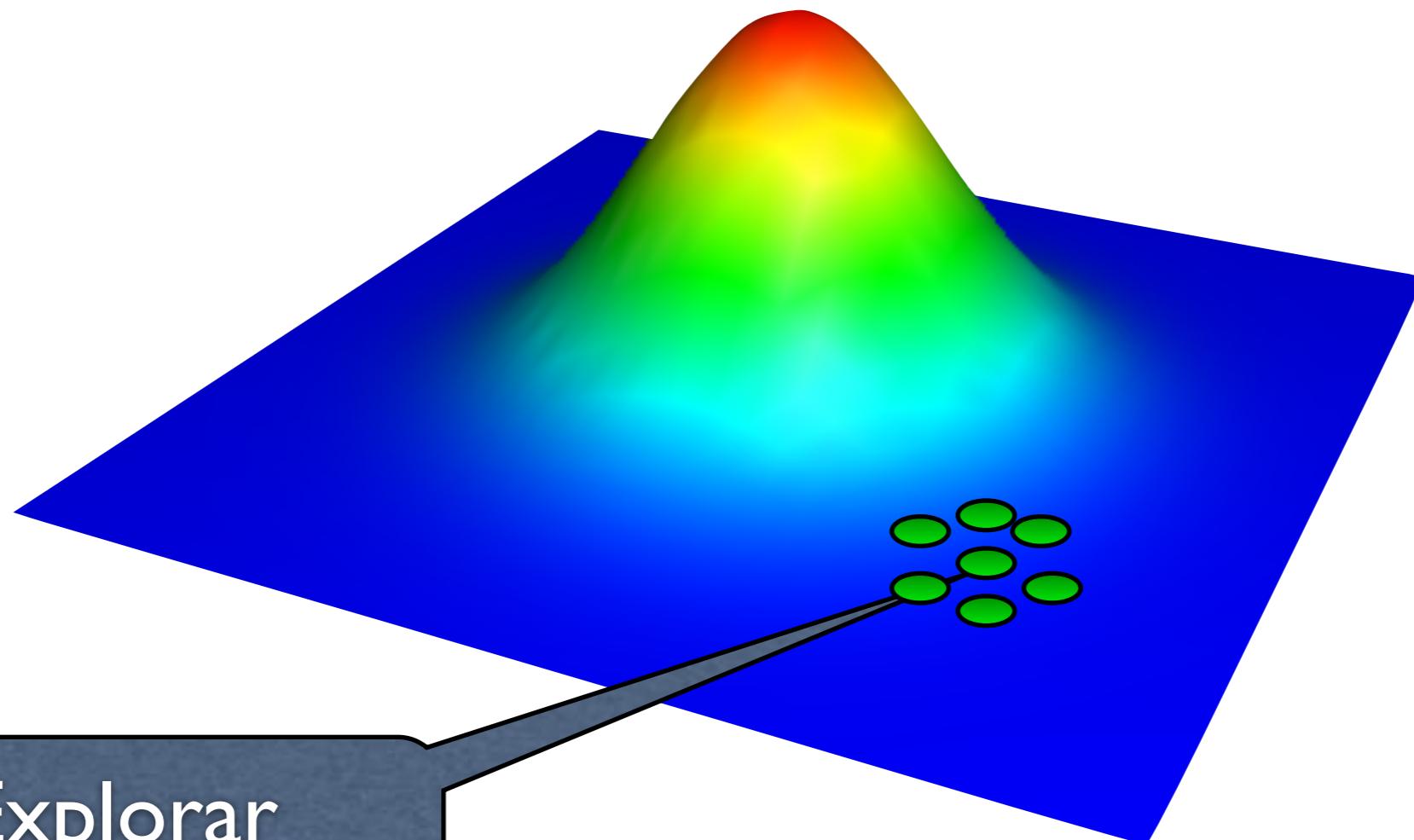


I. Elegir Valor  
Aleatorio

# Hill Climbing

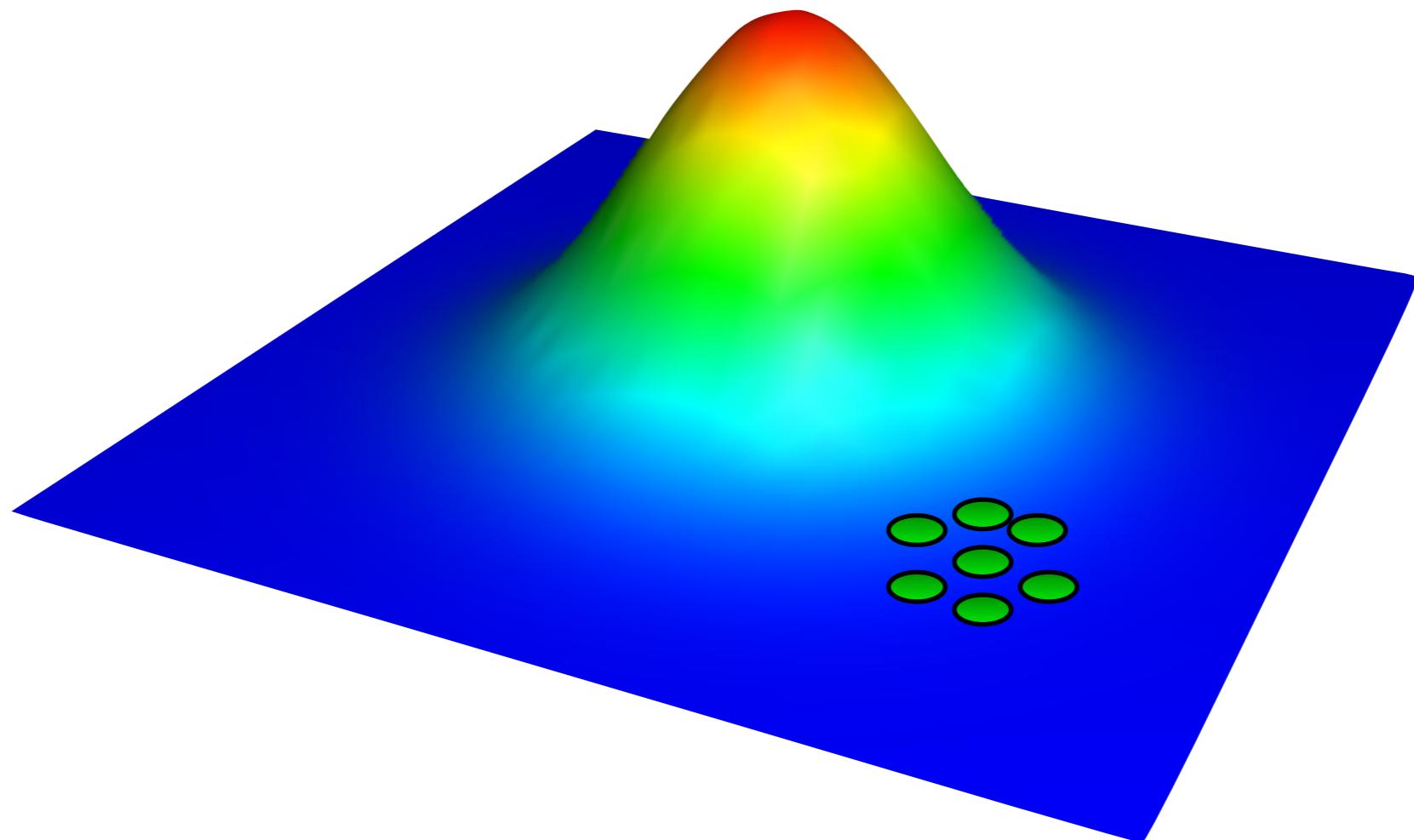


# Hill Climbing

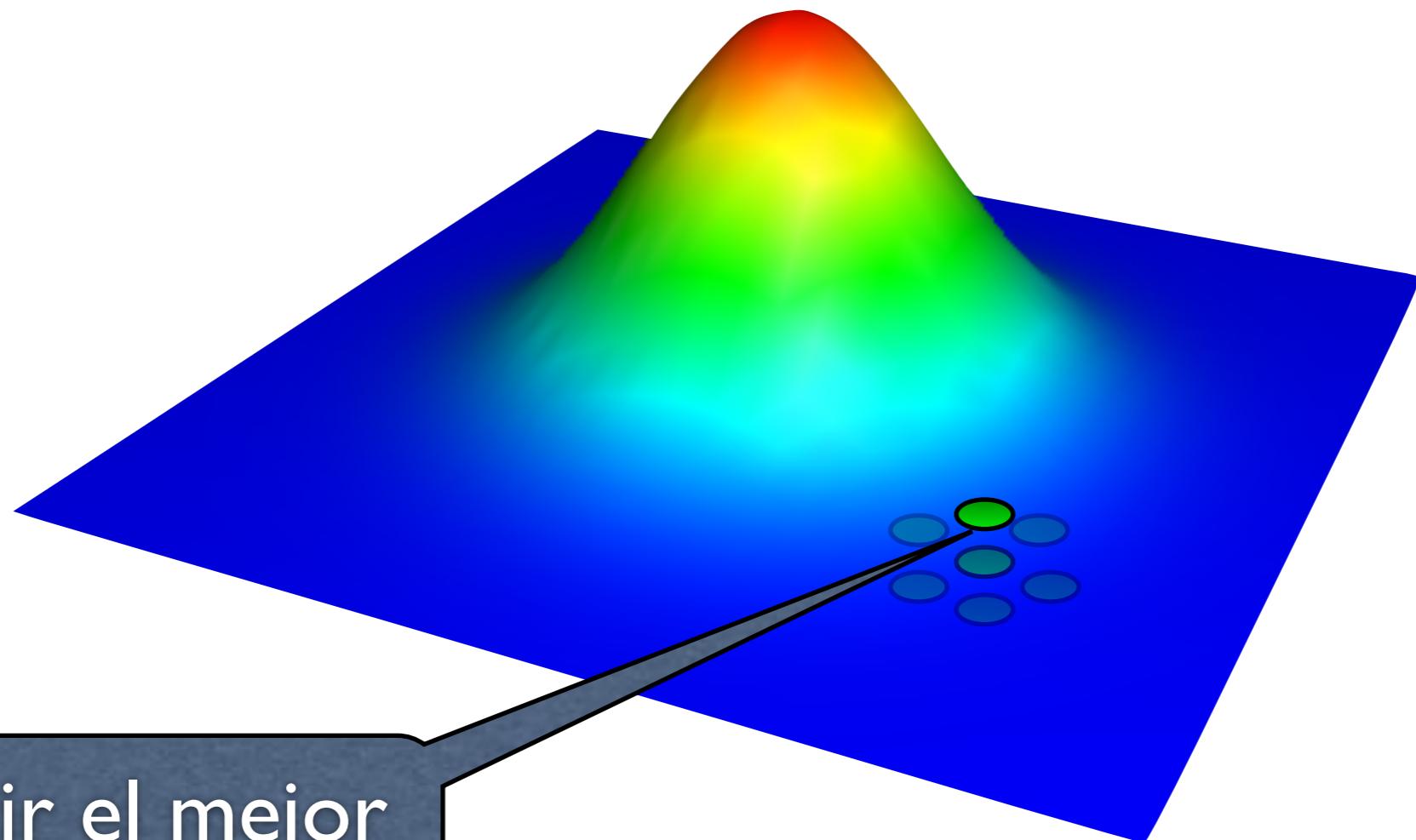


2. Explorar  
Vecindario

# Hill Climbing

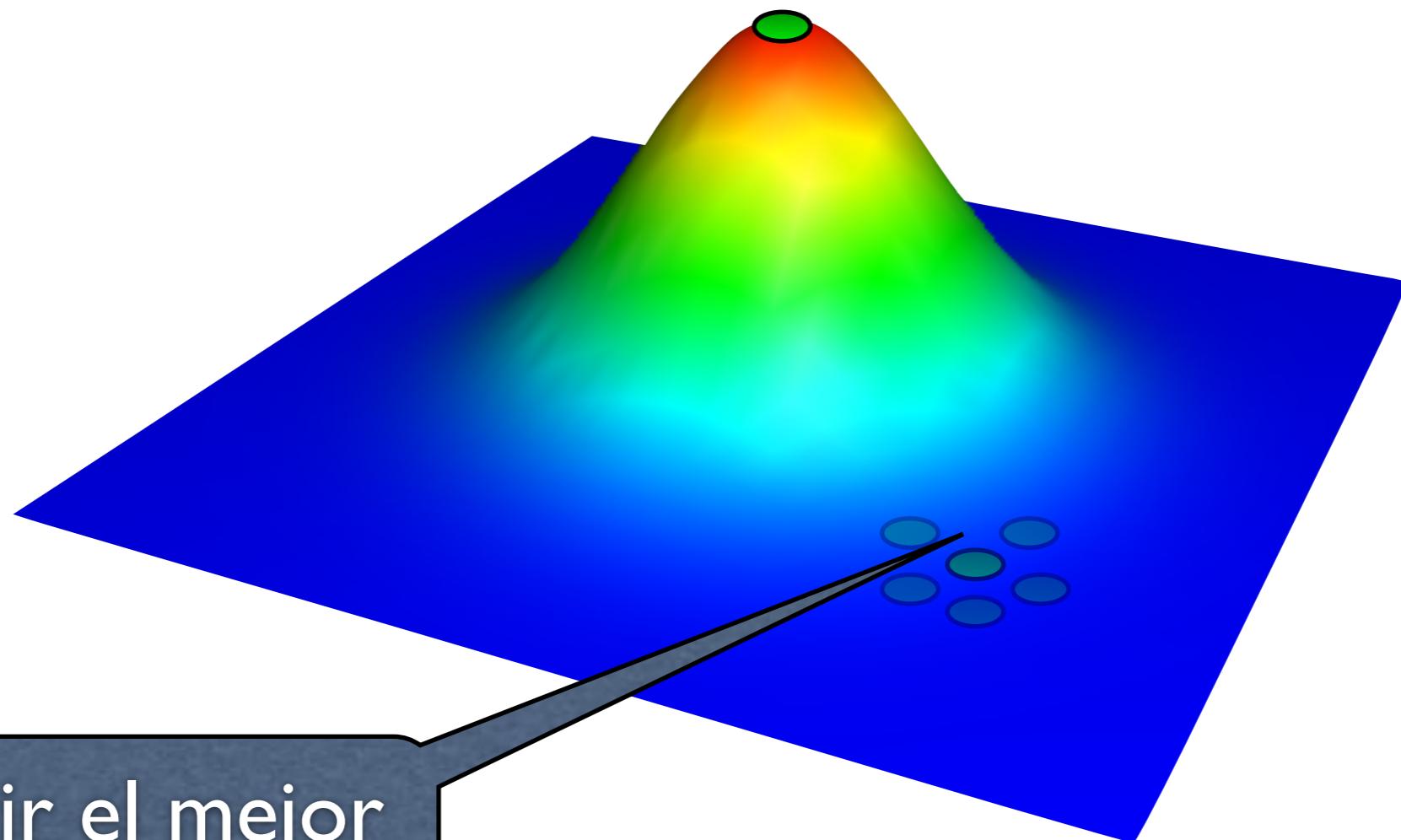


# Hill Climbing



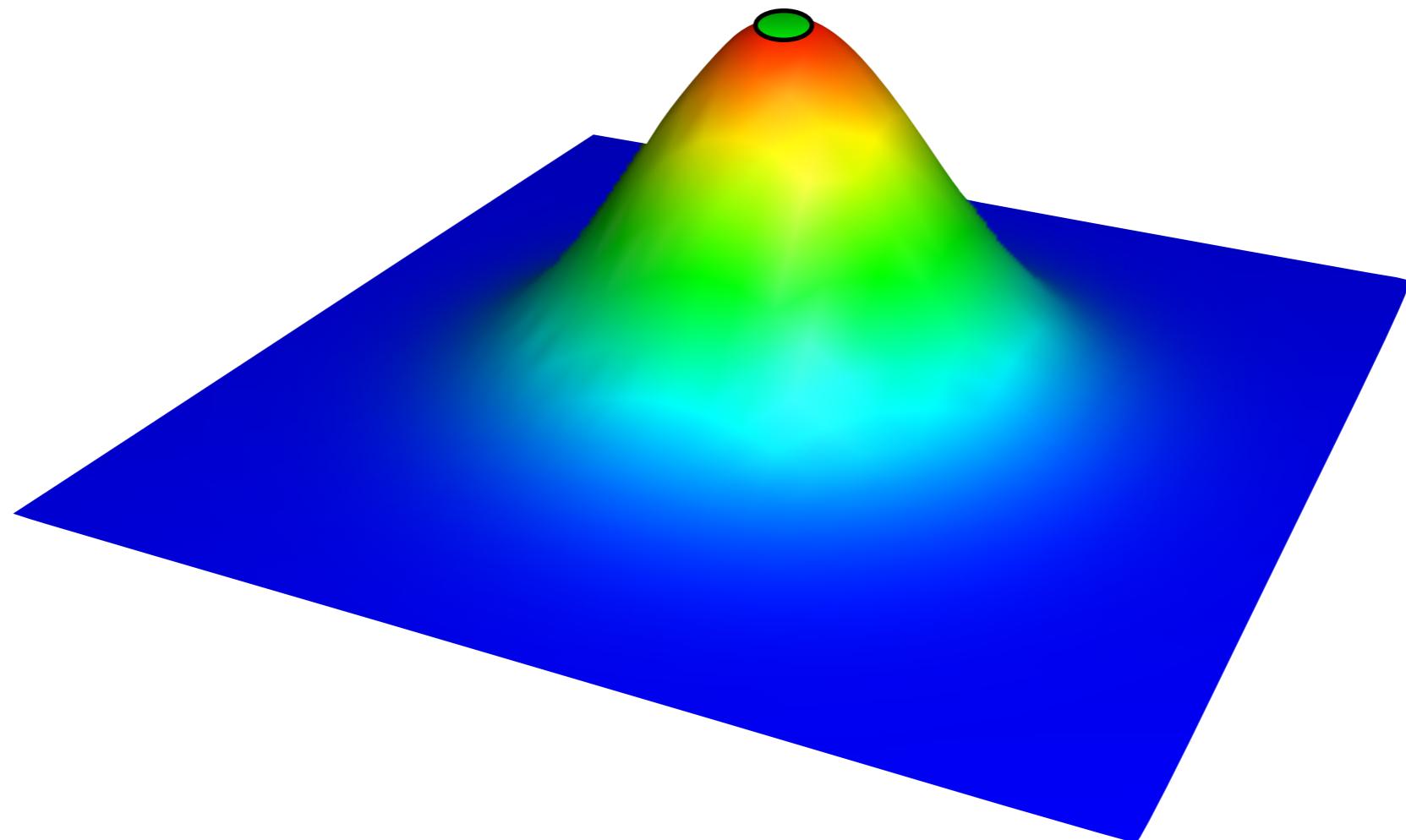
3. Elegir el mejor vecino

# Hill Climbing

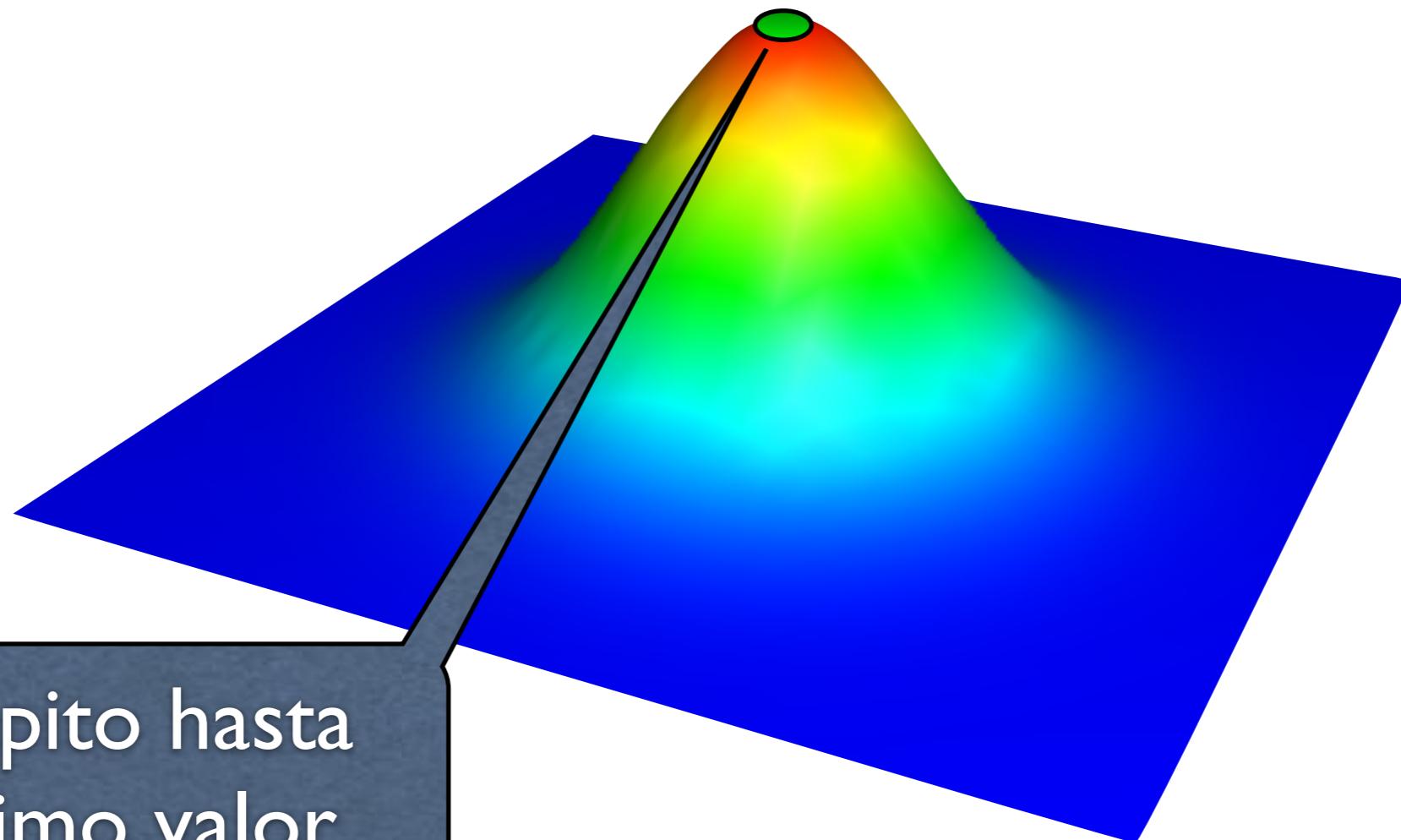


3. Elegir el mejor vecino

# Hill Climbing



# Hill Climbing



4. Repito hasta  
máximo valor

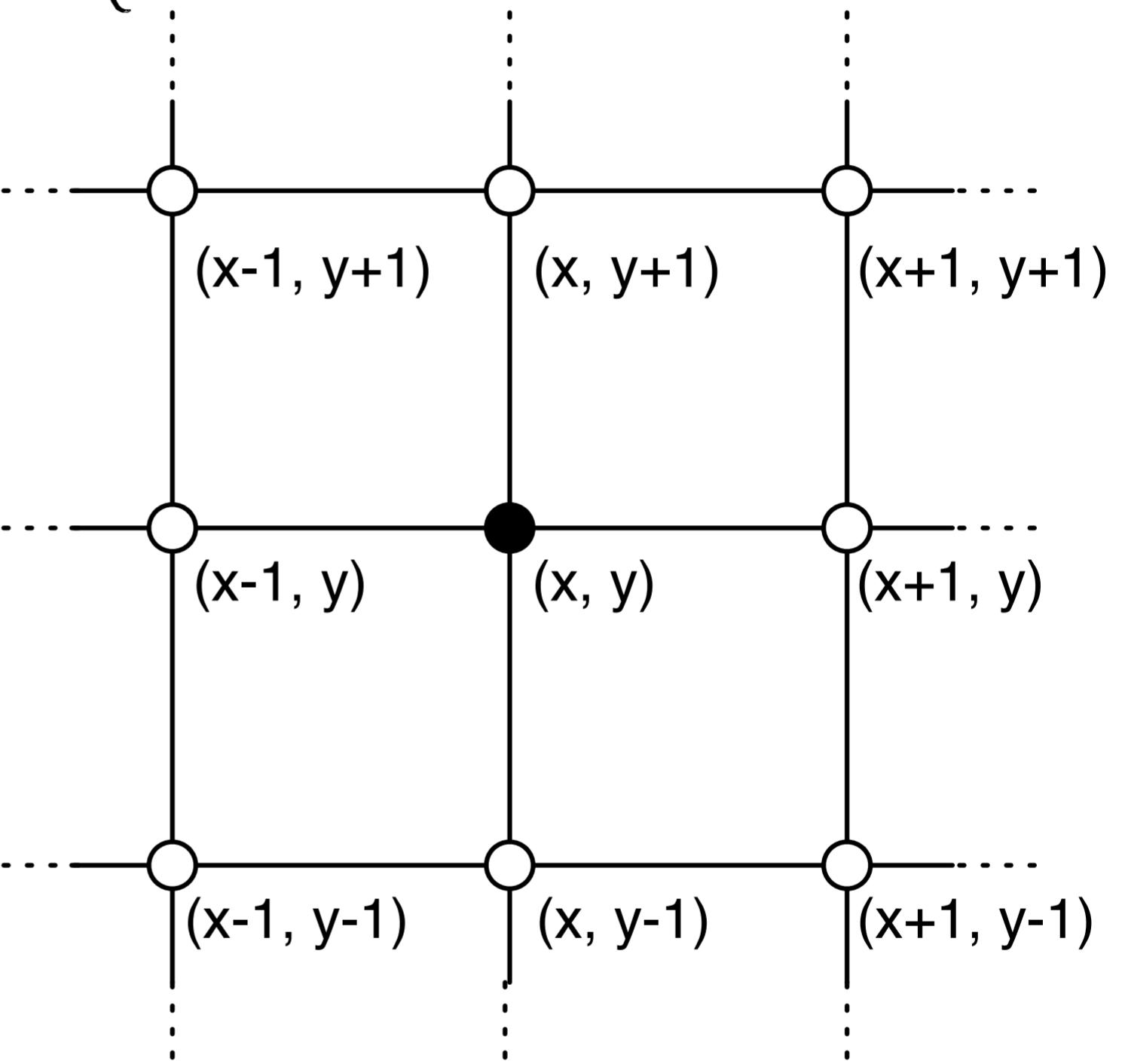
# Hill Climbing

Pseudocódigo iterativo

```
# Elijo al azar una primera solución  
# Mientras no haya alcanzado el goal  
    # Busco entre los vecinos  
        # SI Existe un vecino que es mejor!  
            # Actualizo la nueva posición  
    # SINO termino la búsqueda
```

# Neighbourhood

(Vecinos)



```
def testMe(x, y):  
    if x == y:  
        return True  
    else:  
        return False
```

Should be small enough  
to be able to evaluate & find  
best neighbor

# Hill Climbing

```
def testMe(x, y):  
    if x == y:  
        return True, abs(x-y)  
    else:  
        return False, abs(x-y)
```

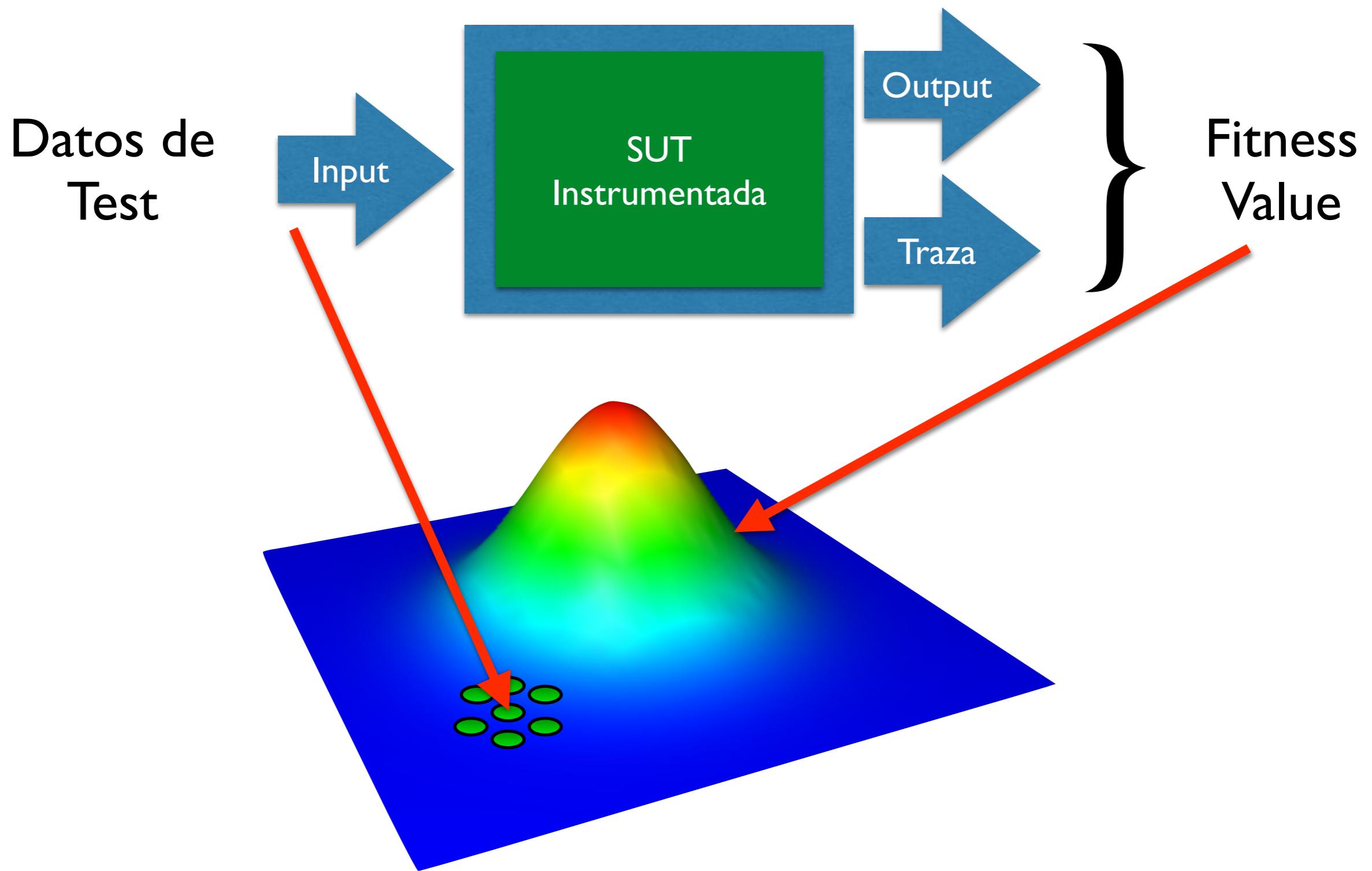


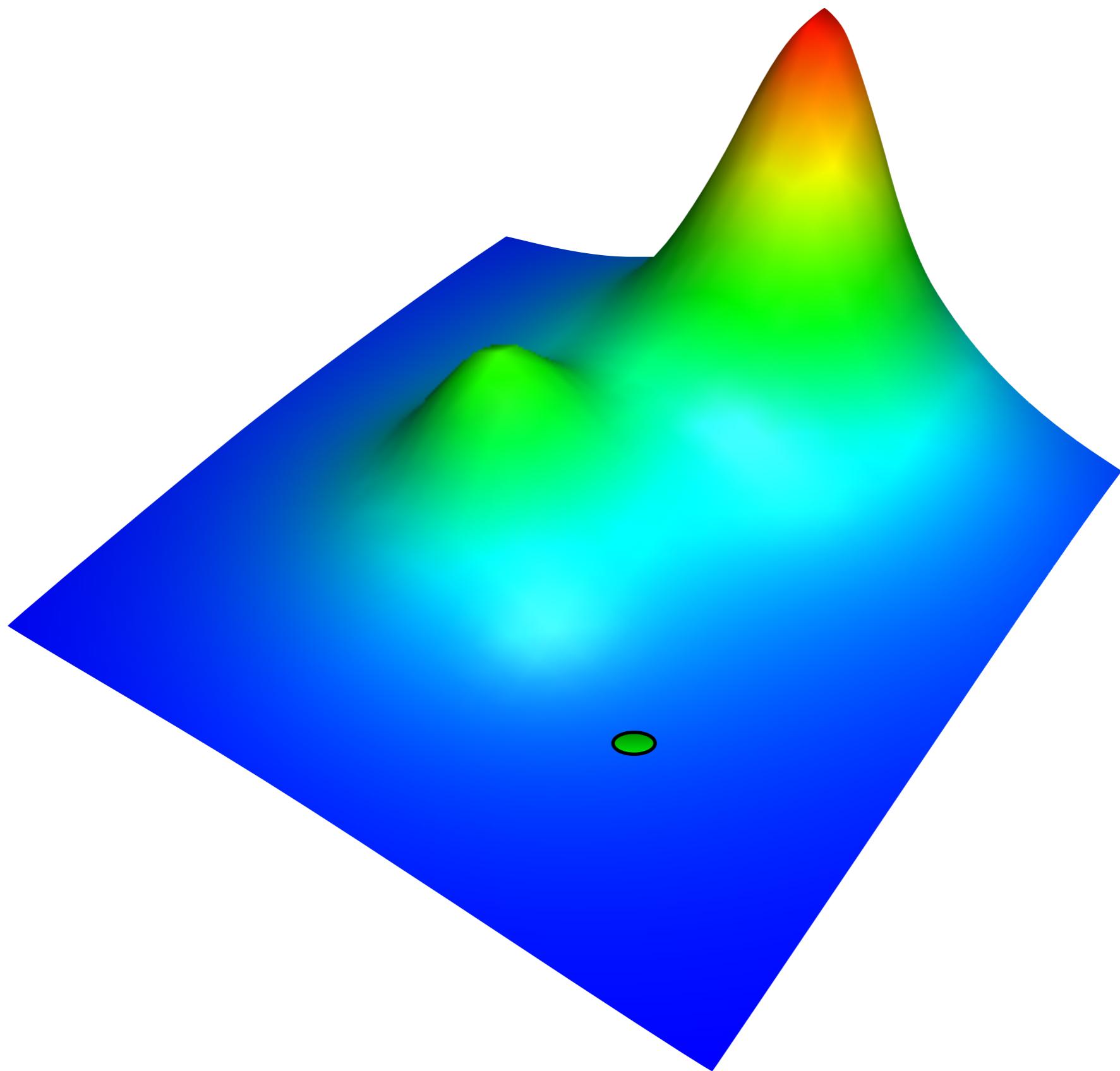
```
MAX = sys.maxsize
MIN = -sys.maxsize -1
x = random.randint(MIN,MAX)
y = random.randint(MIN,MAX)
ret_val,fitness = testMe(x,y)
while ret_val!=True:
    for [x',y'] in neighbours(x,y):
        ret_val,fitness' = testMe(x'y')
        if ret_val==True or fitness'<fitness:
            x=x', y=y'
            break
    if fitness==fitness':
        break
```

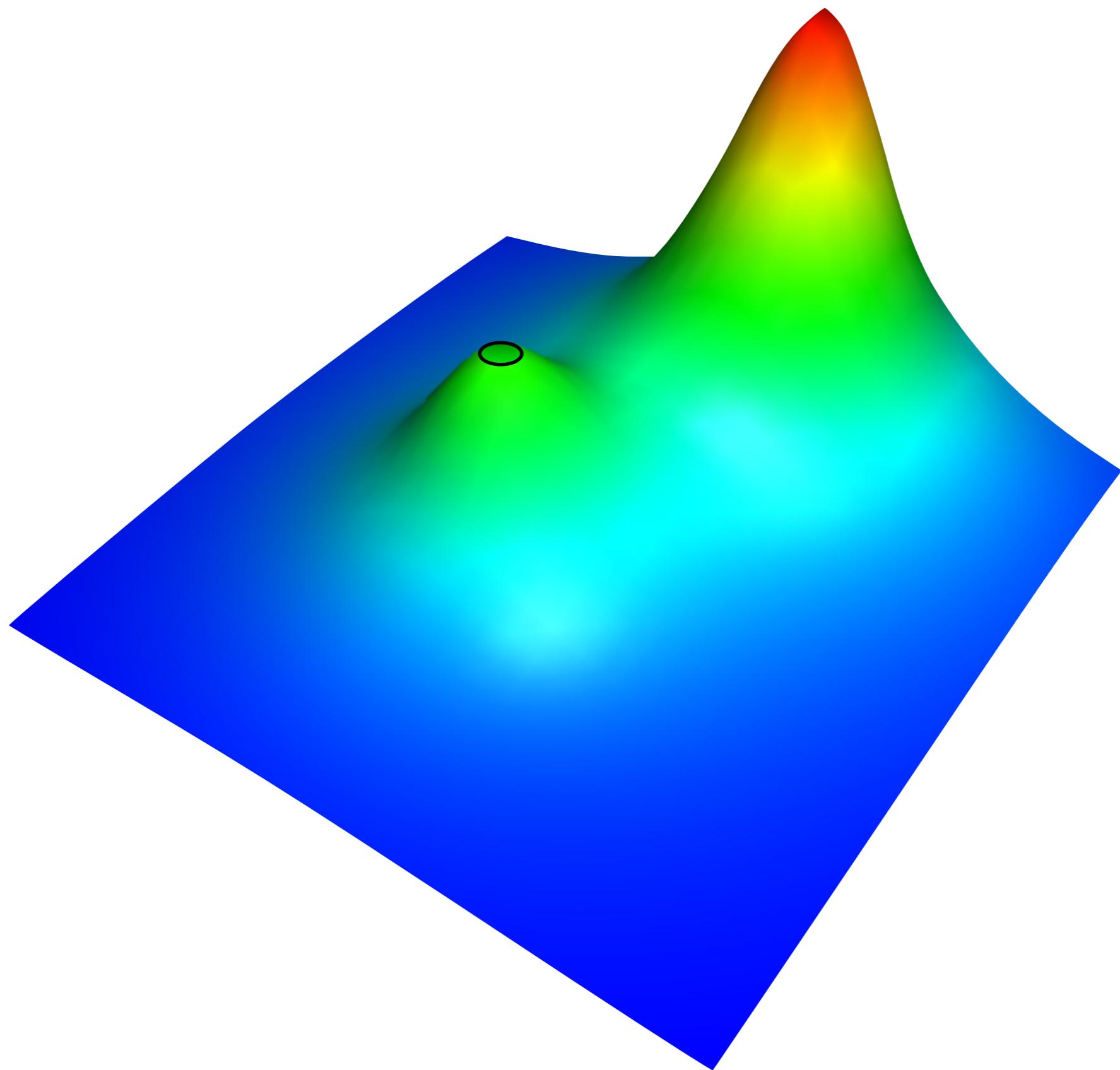
# SUT (instrumentada)

# Test Driver

# Possible implementation







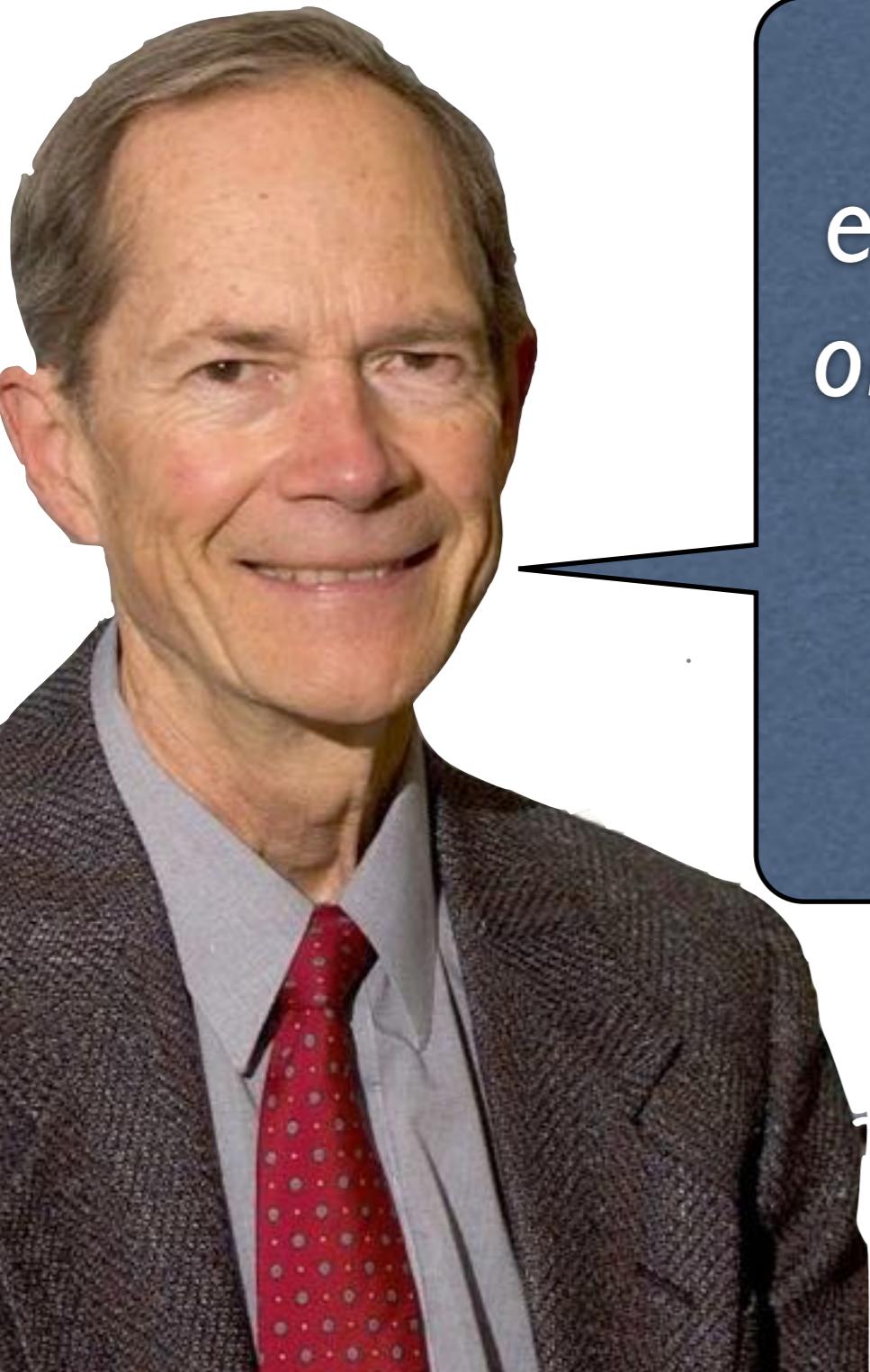
You can't  
improve too  
quickly

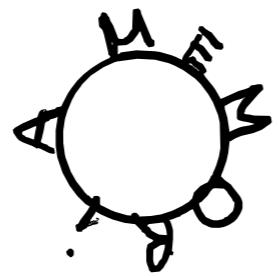


# Tabu Search

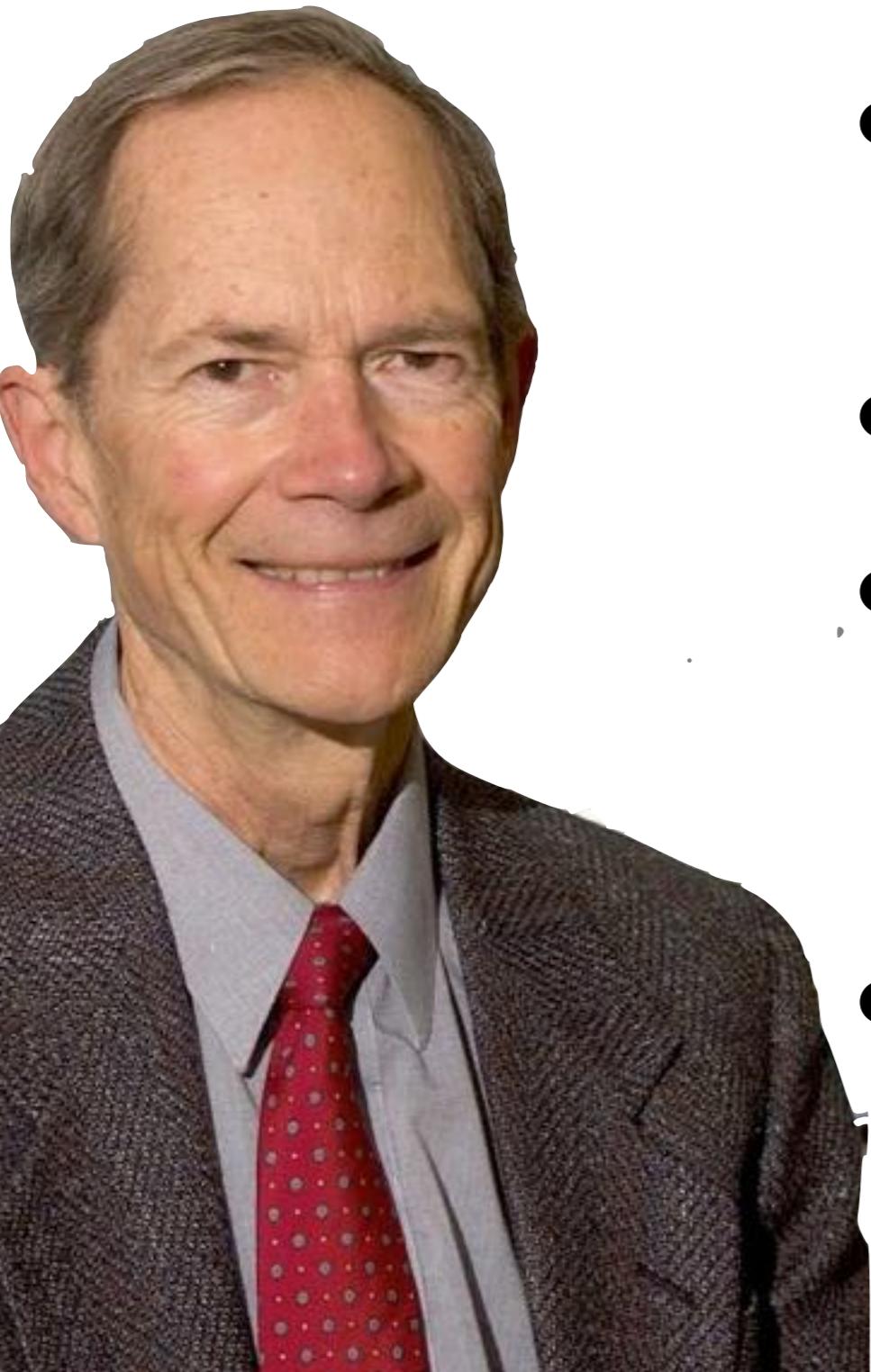
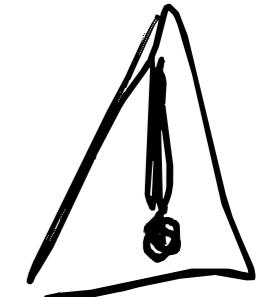
*The overall approach is to avoid entrainment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited.*

(Fred Glover 1987)

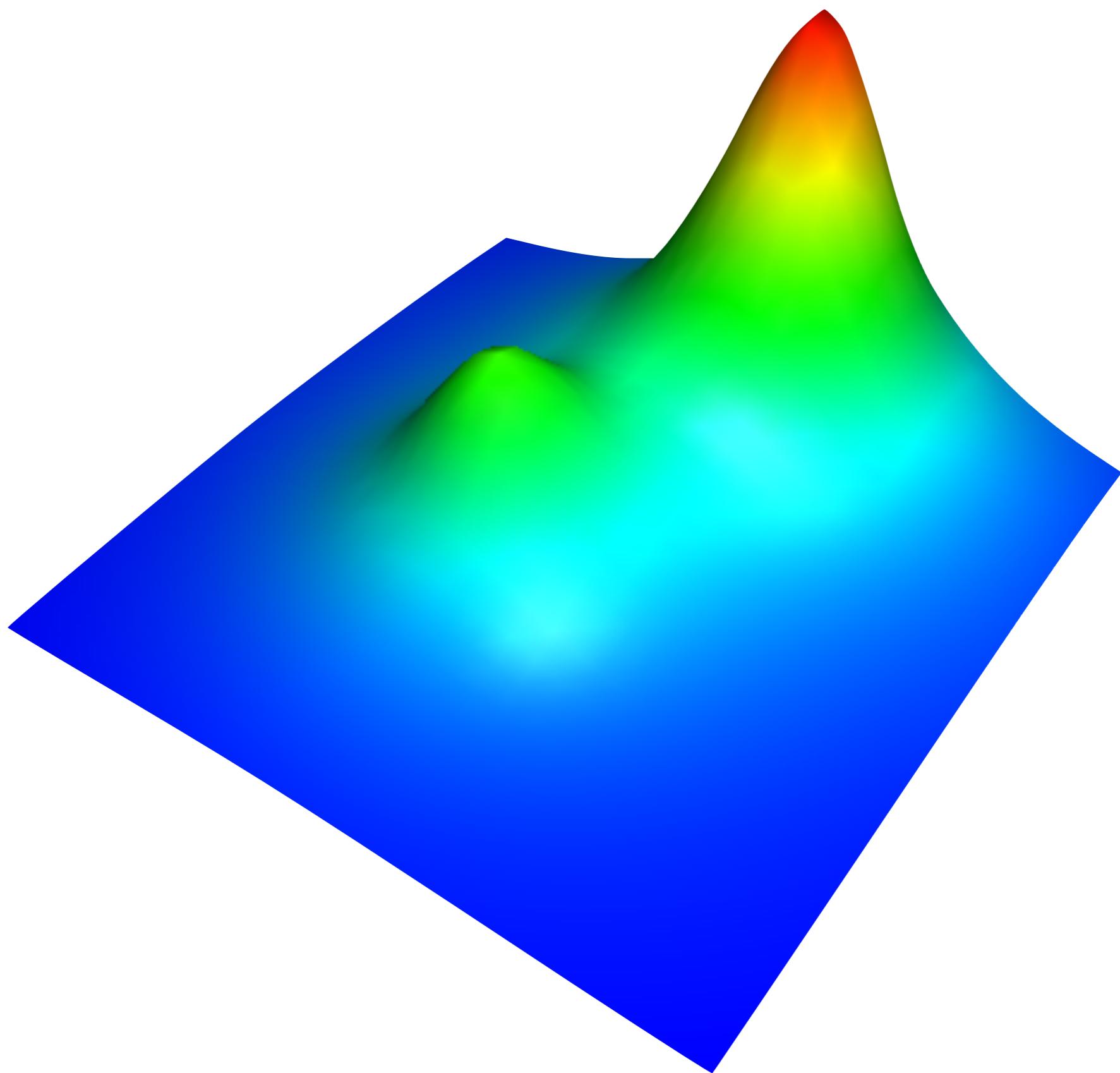


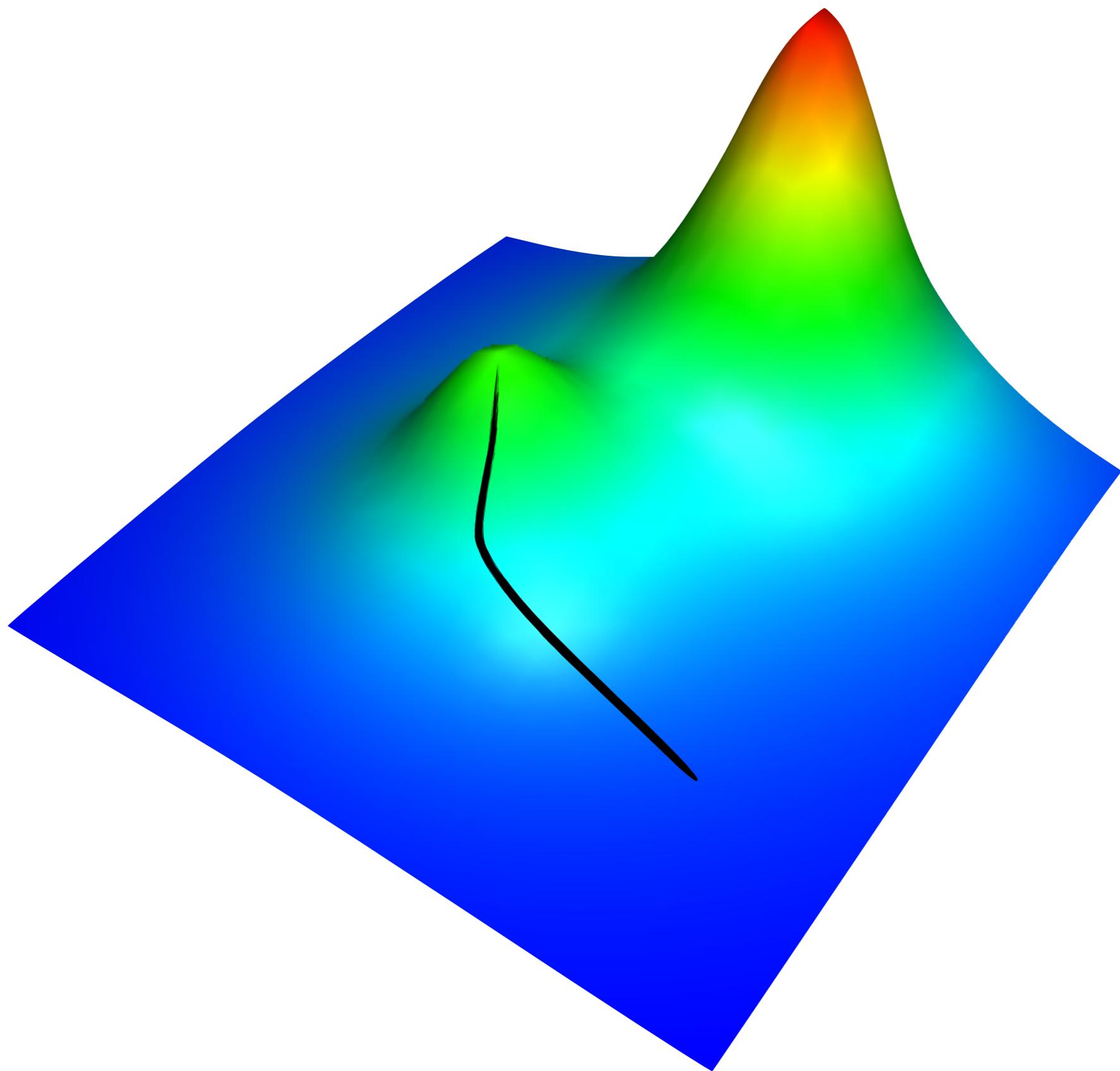


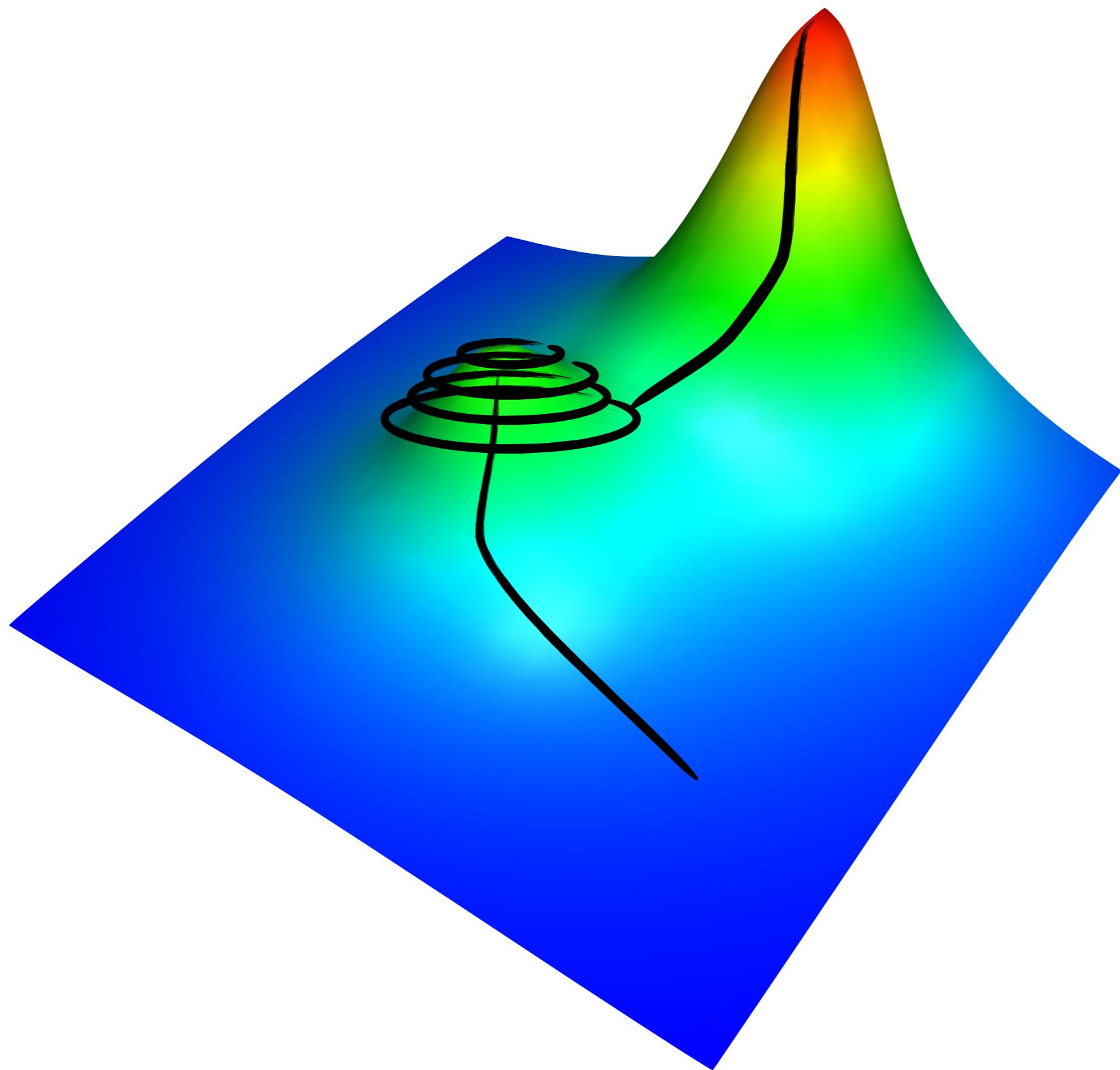
# Tabu (taboo) Search



- Almacena los últimos k-movimientos
- “Tabu list”
- Cuando elijo un nuevo input, no puedo tomar ninguna solución que esté en la “Tabu list”
- Evita ciclos de longitud k







# Hill Climbing

```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True  
    else:  
        return False
```

- ¿Qué pasa con este ejemplo?

# Hill Climbing

```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True  
    else:  
        return False
```

→ no lento  
→ 2 big  
neighbourhood

- ¿Qué pasa con este ejemplo?
  - Cuando el neighbourhood es demasiado grande, Hill-Climbing se torna demasiado lento

# Hill Climbing

```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True  
    else:  
        return False
```

- ¿Qué pasa con este ejemplo?
  - Cuando el neighbourhood es demasiado grande, Hill-Climbing se torna demasiado lento
  - Necesitamos mecanismos más sofisticados de búsqueda

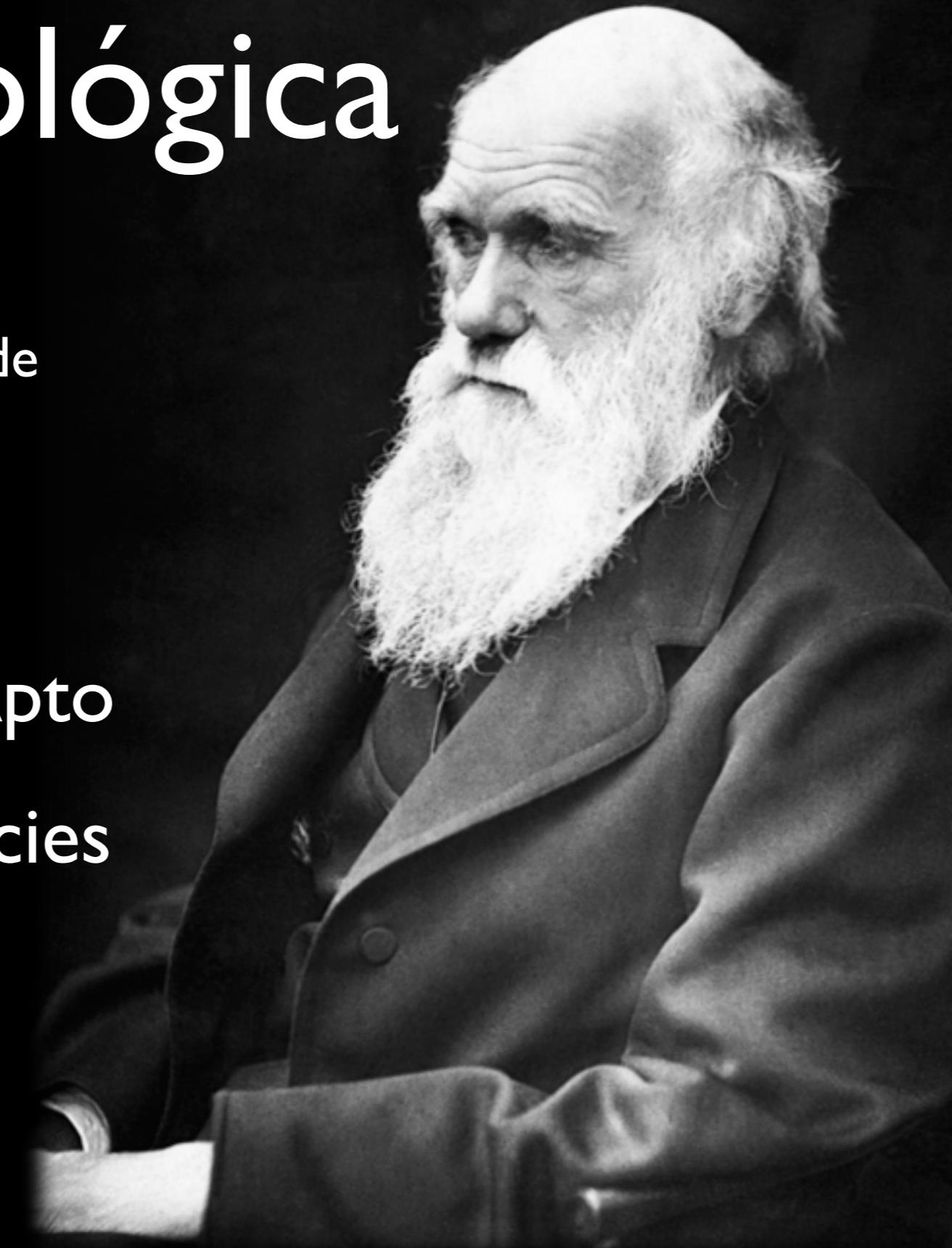
Alg. genéticos  
e

# Algoritmos Evolutivos



# Evolución Biológica

- Gen  
Unidad de información • pasado de una generación a la otra
- Selección Natural
- Supervivencia del Más Apto
- Origen de Nuevas Especies

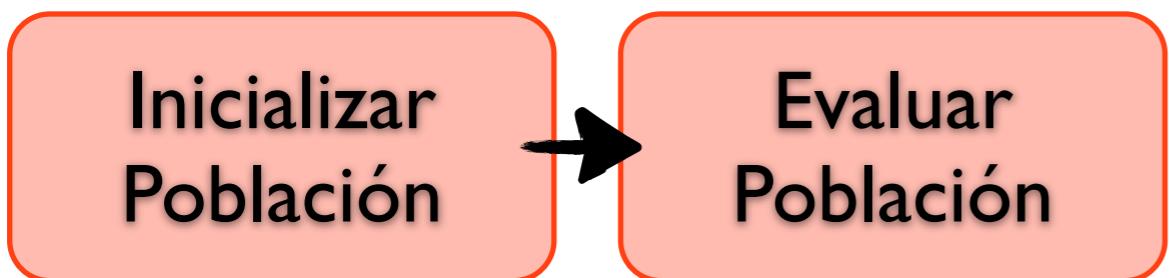


# Algoritmos Genéticos

# Algoritmos Genéticos

Inicializar  
Población

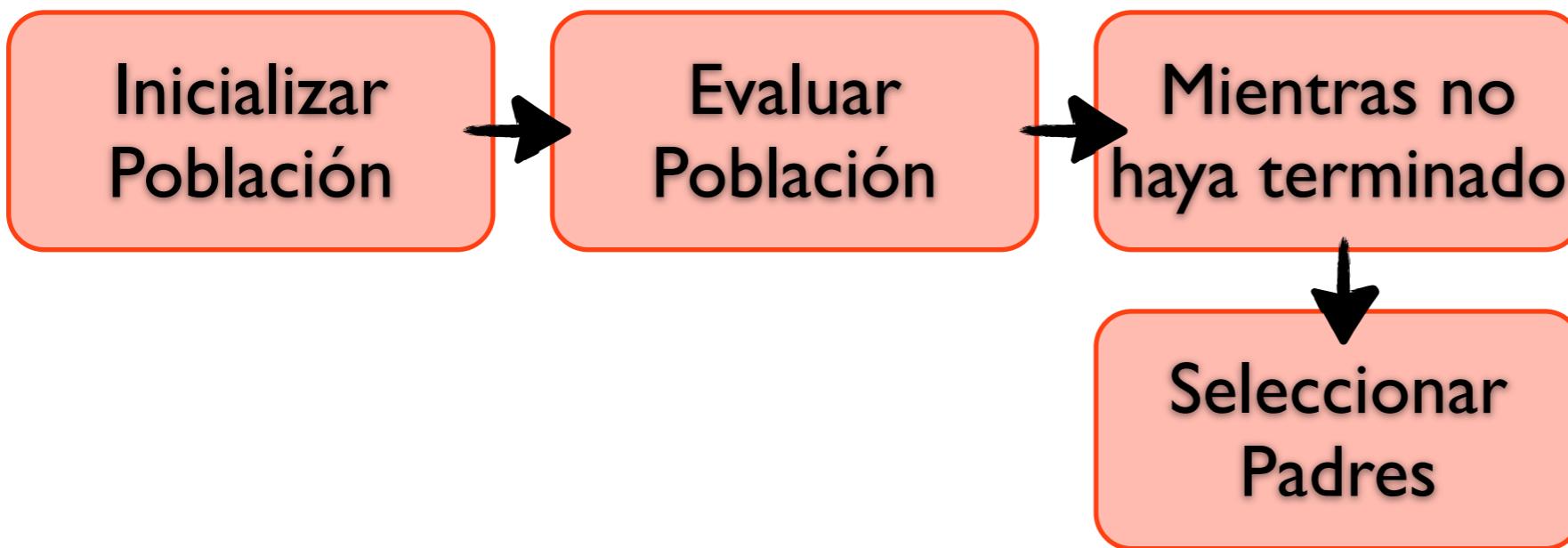
# Algoritmos Genéticos



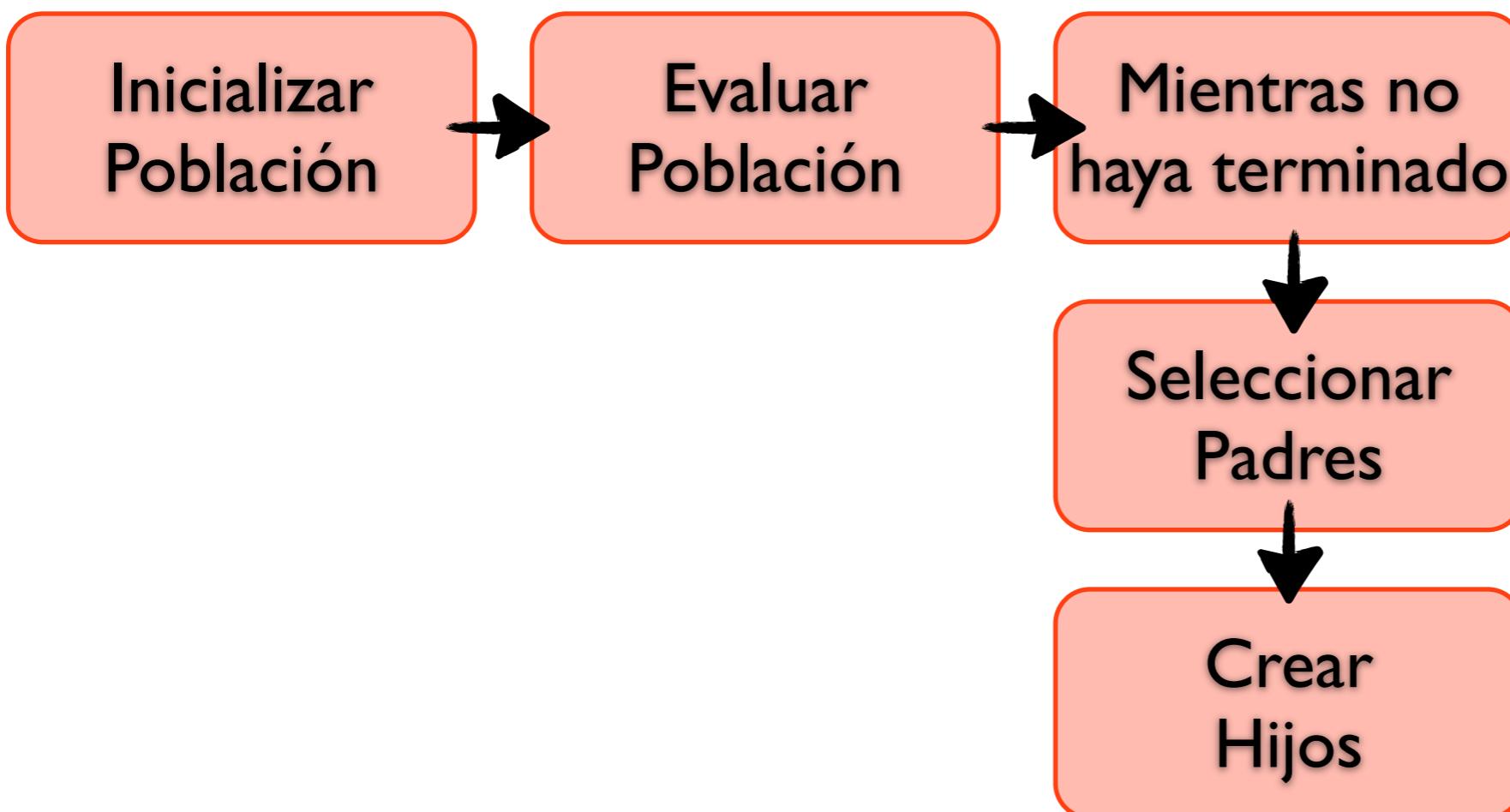
# Algoritmos Genéticos



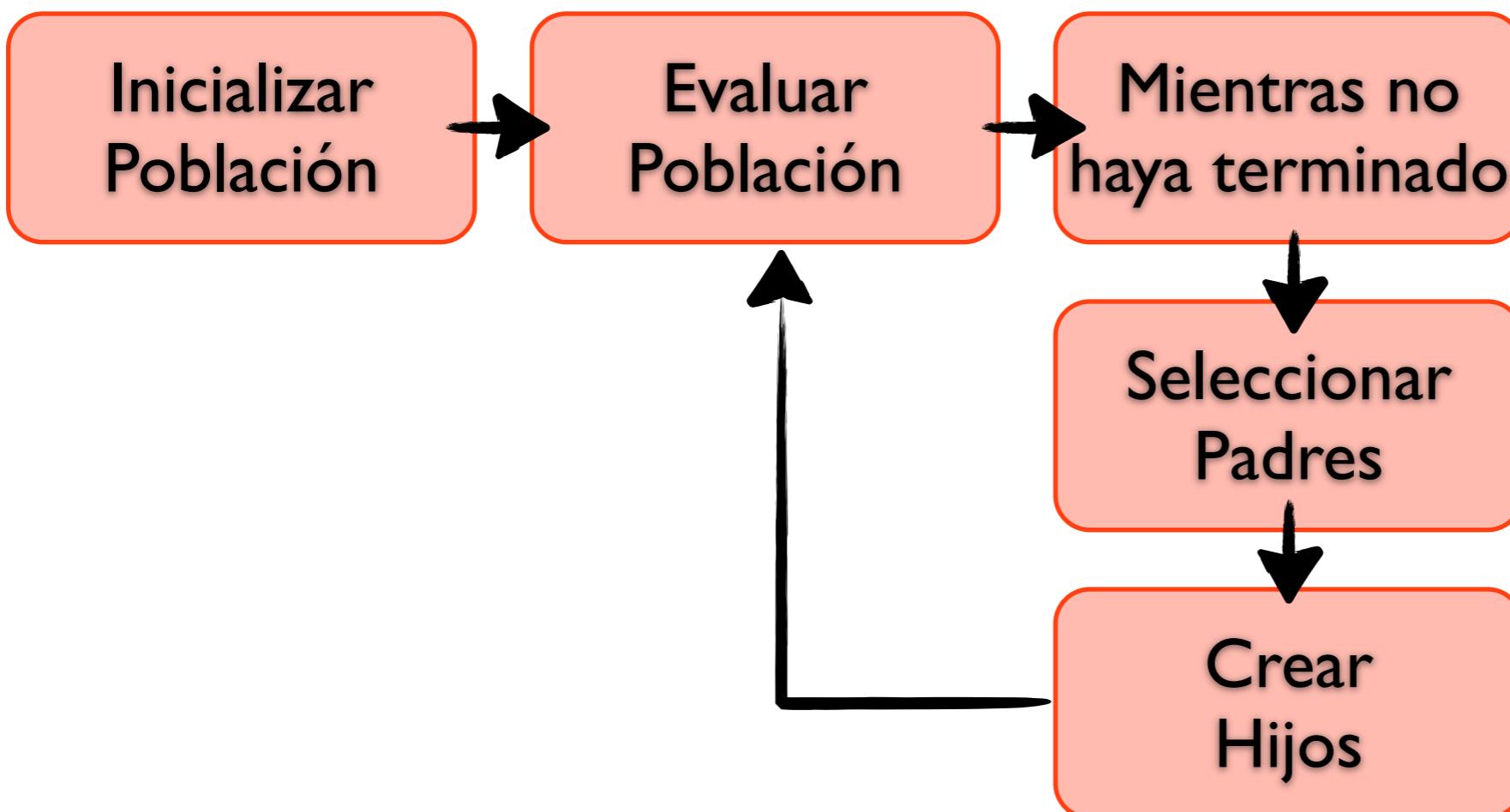
# Algoritmos Genéticos



# Algoritmos Genéticos



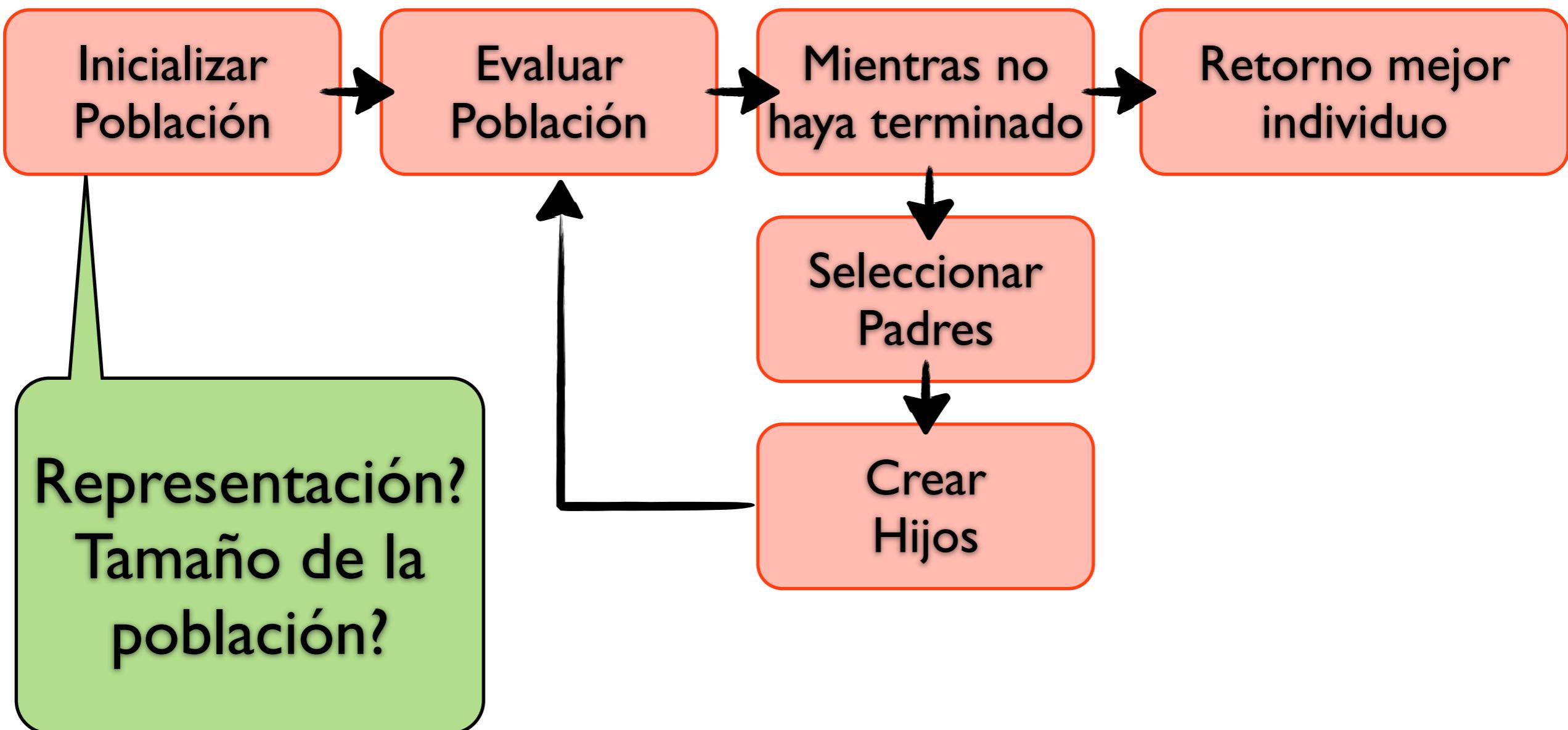
# Algoritmos Genéticos



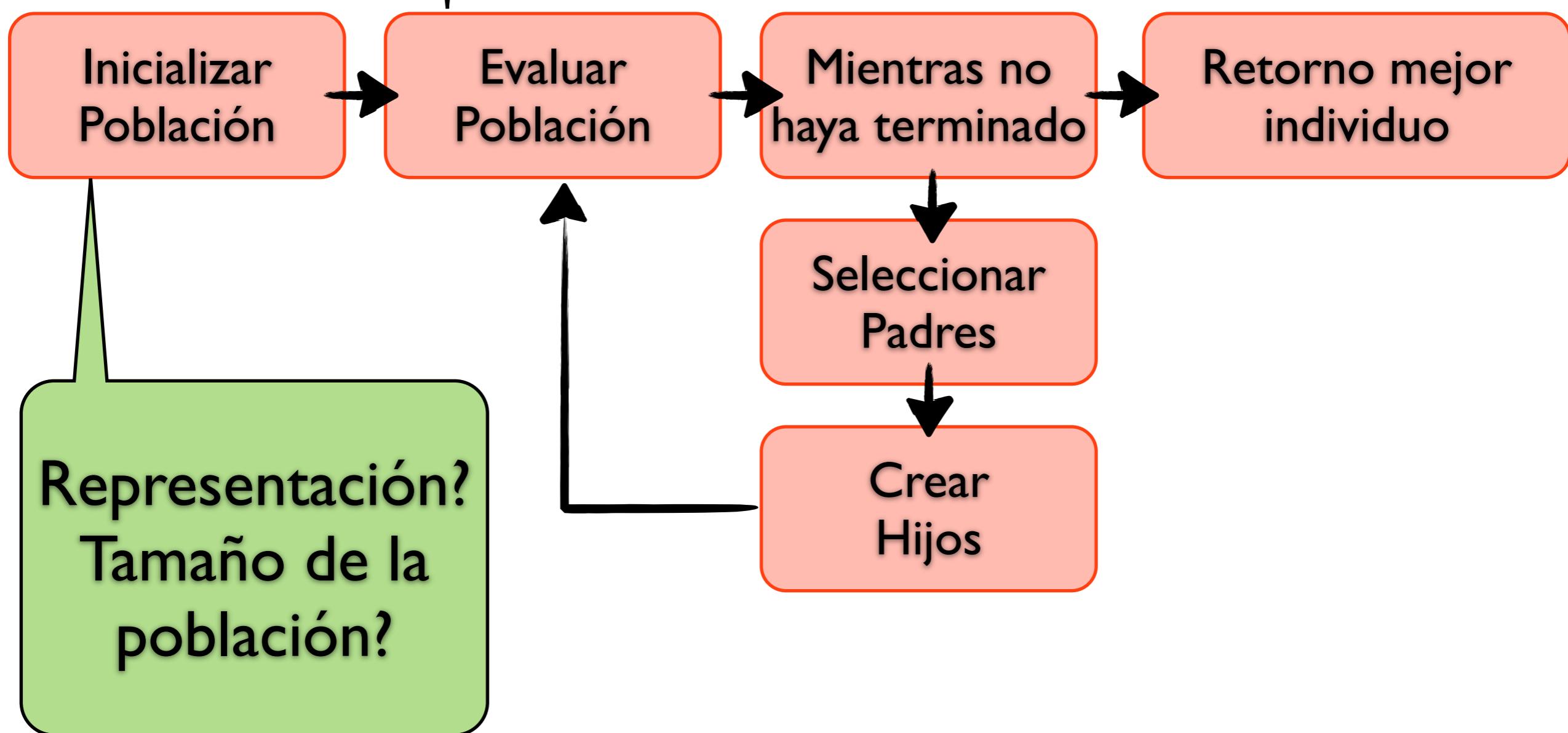
# Algoritmos Genéticos

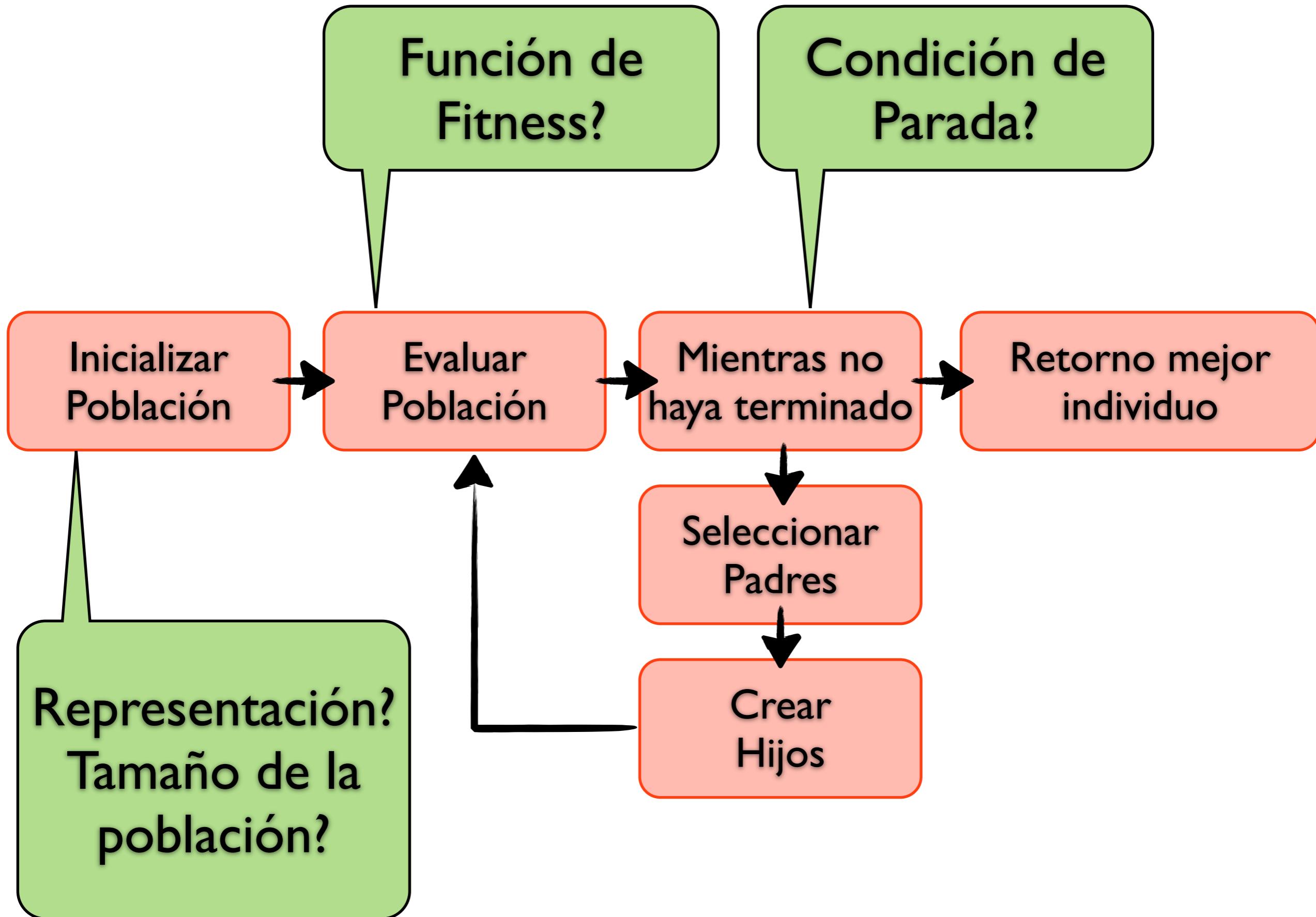


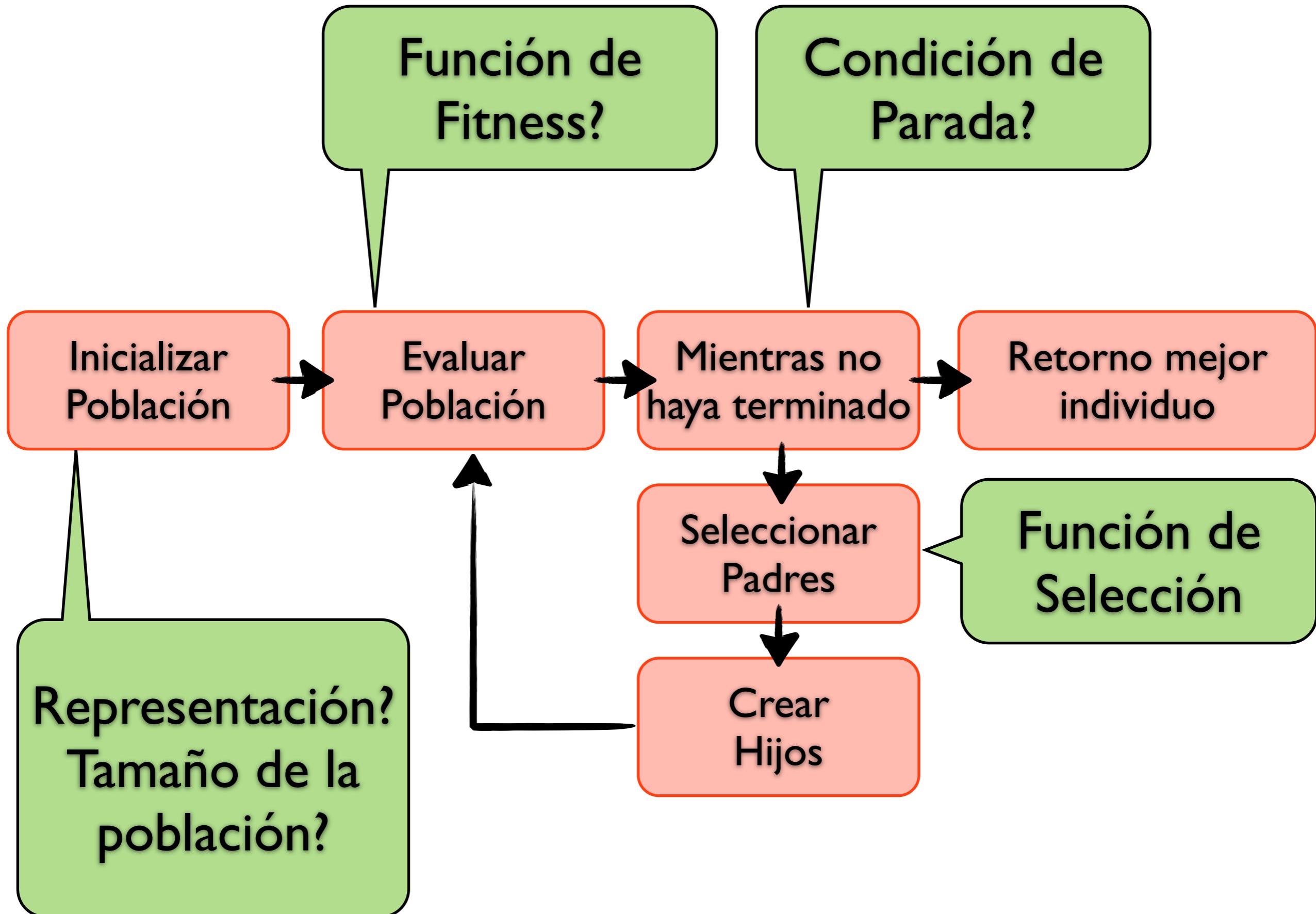




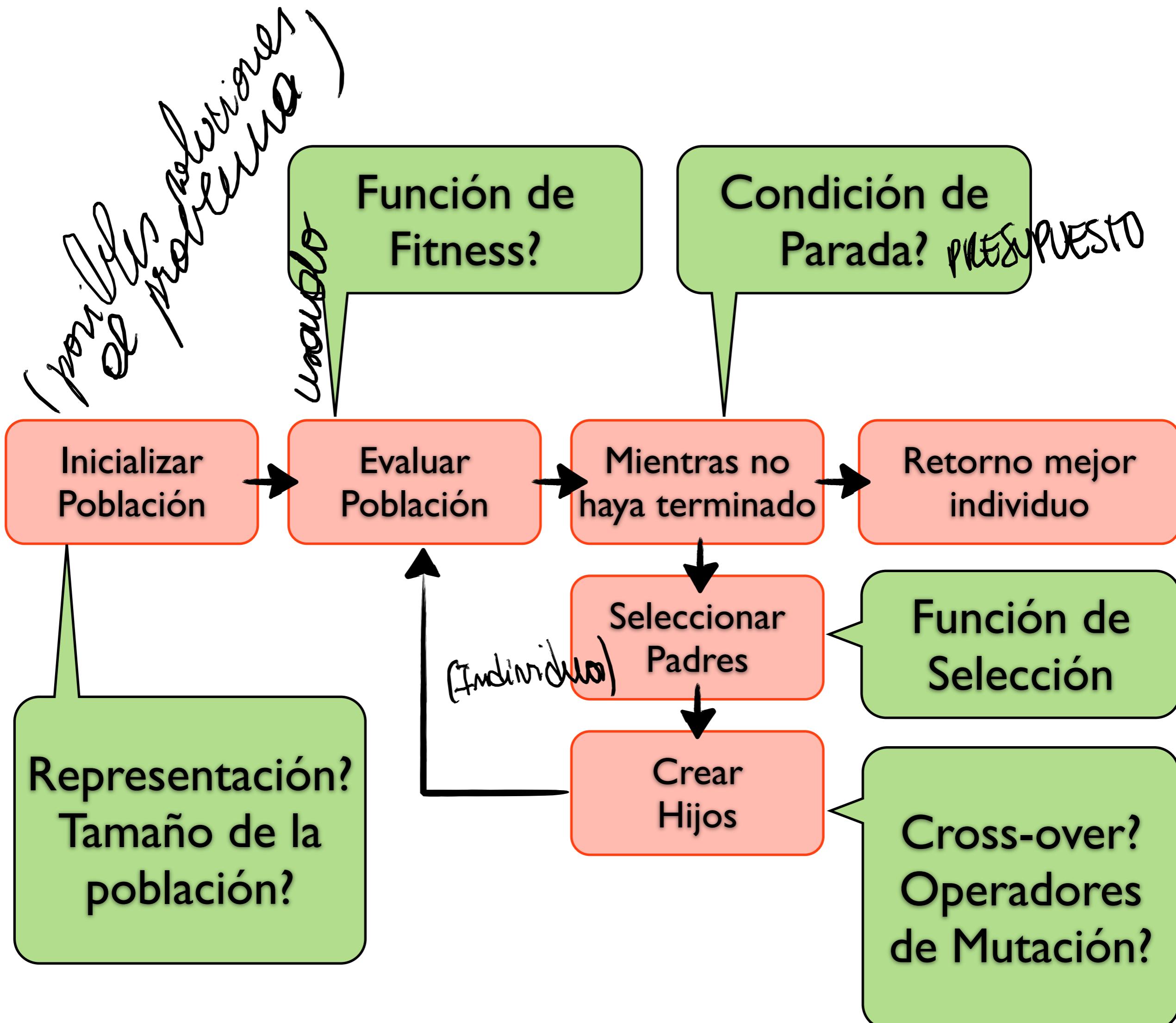
Función de Fitness?

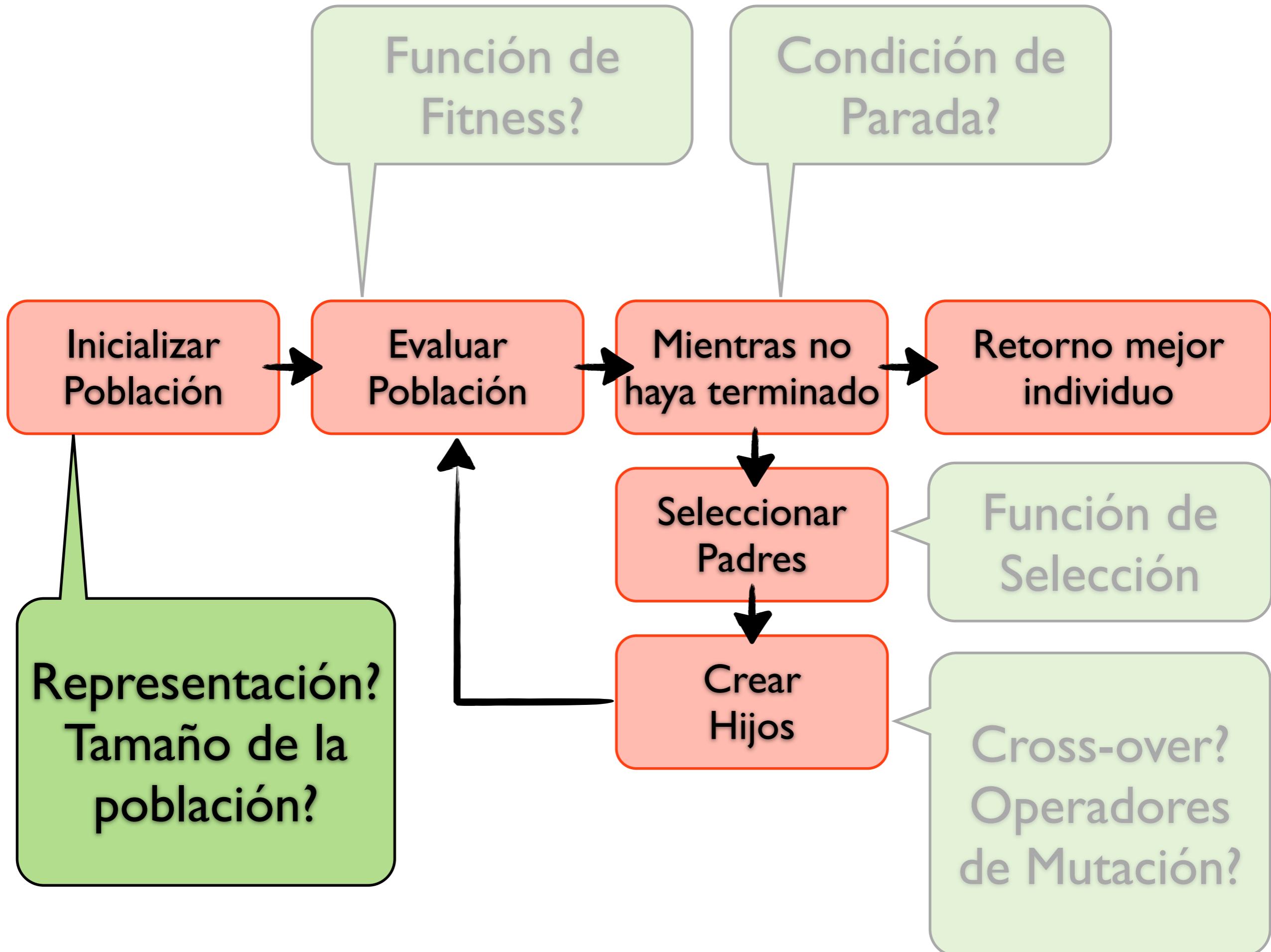






# Fases de los Algoritmos Genéticos





# Ejemplo de Representación (de individuos)

```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True //branch a cubrir  
    else:  
        return False
```

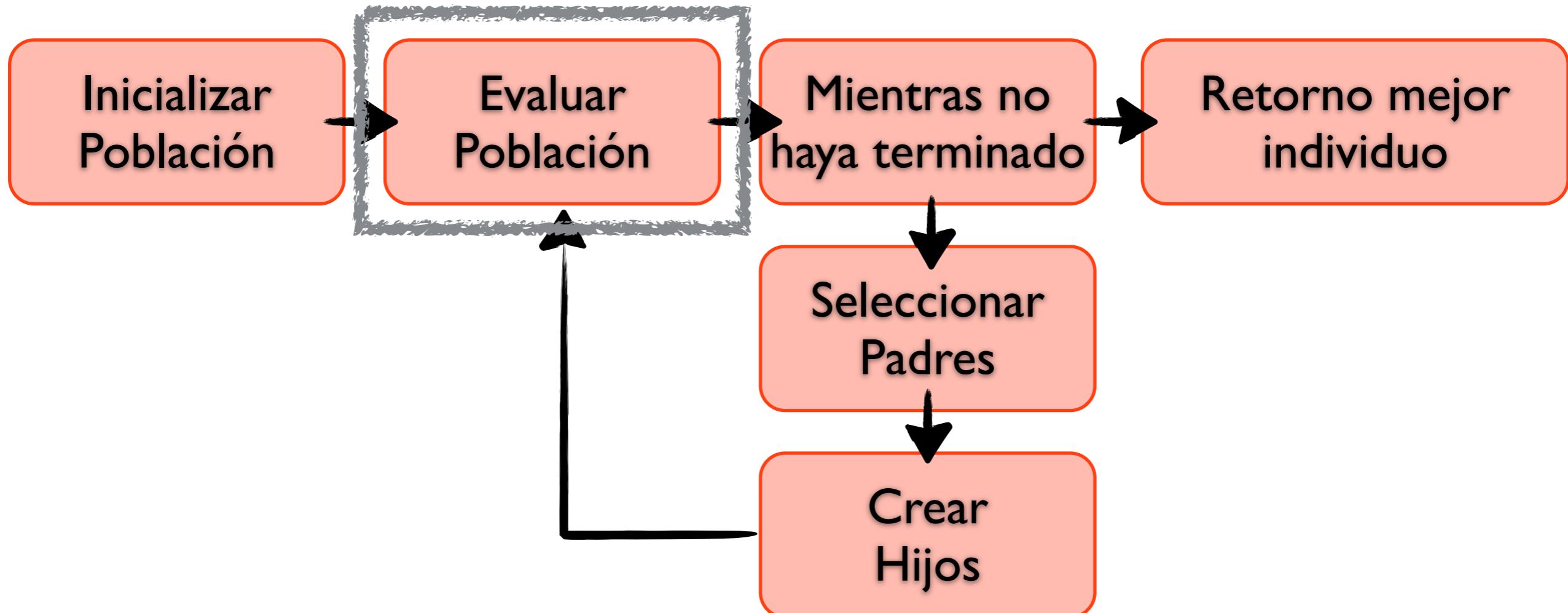
x	y	z
10	10	20

# ¿Cómo generar la Población Inicial ?

- Aleatoriamente
- Usar soluciones existentes
- Usar un Test Suite con cobertura de un criterio de menor potencia
- Manualmente





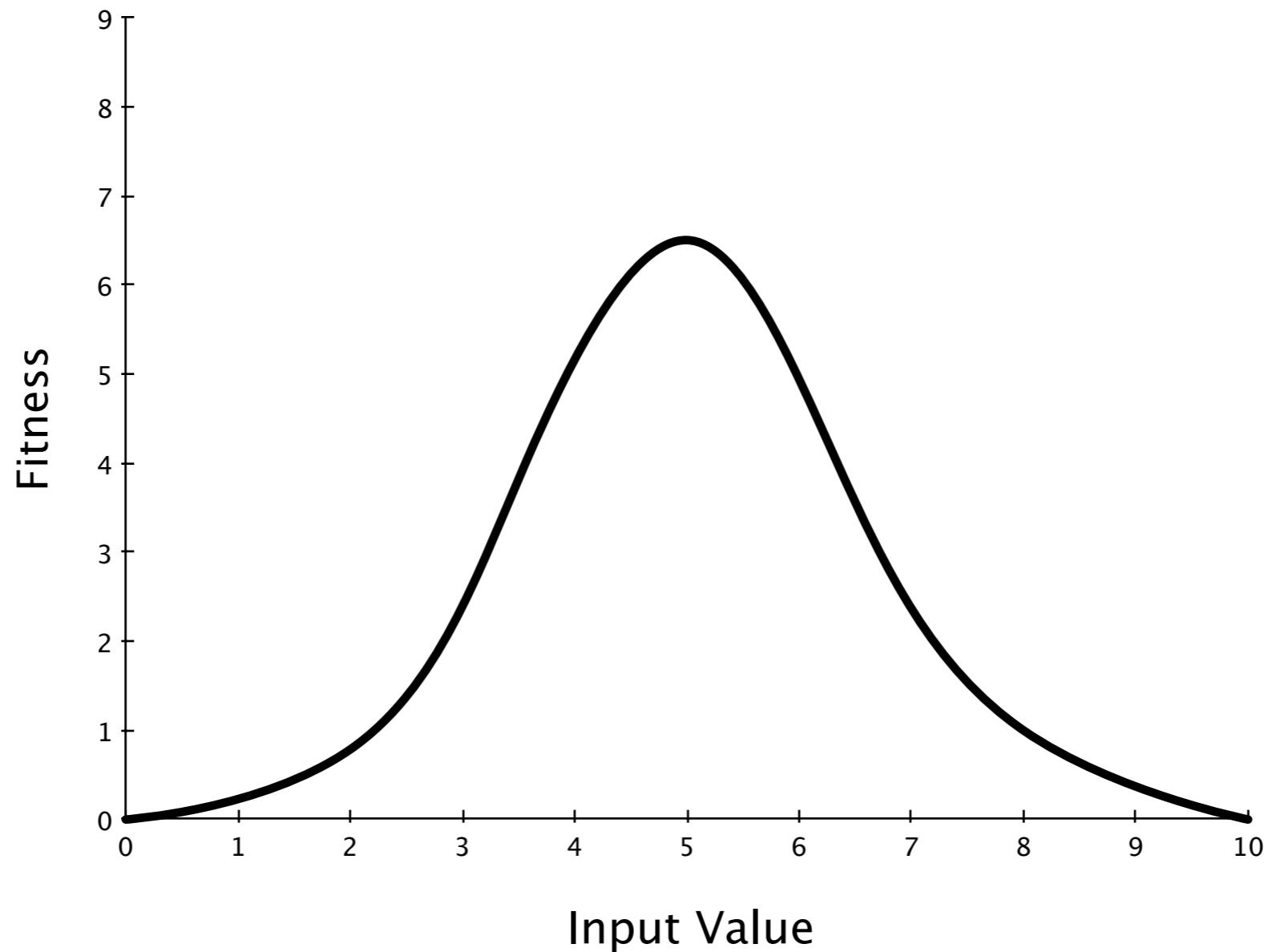


*responde con un número*

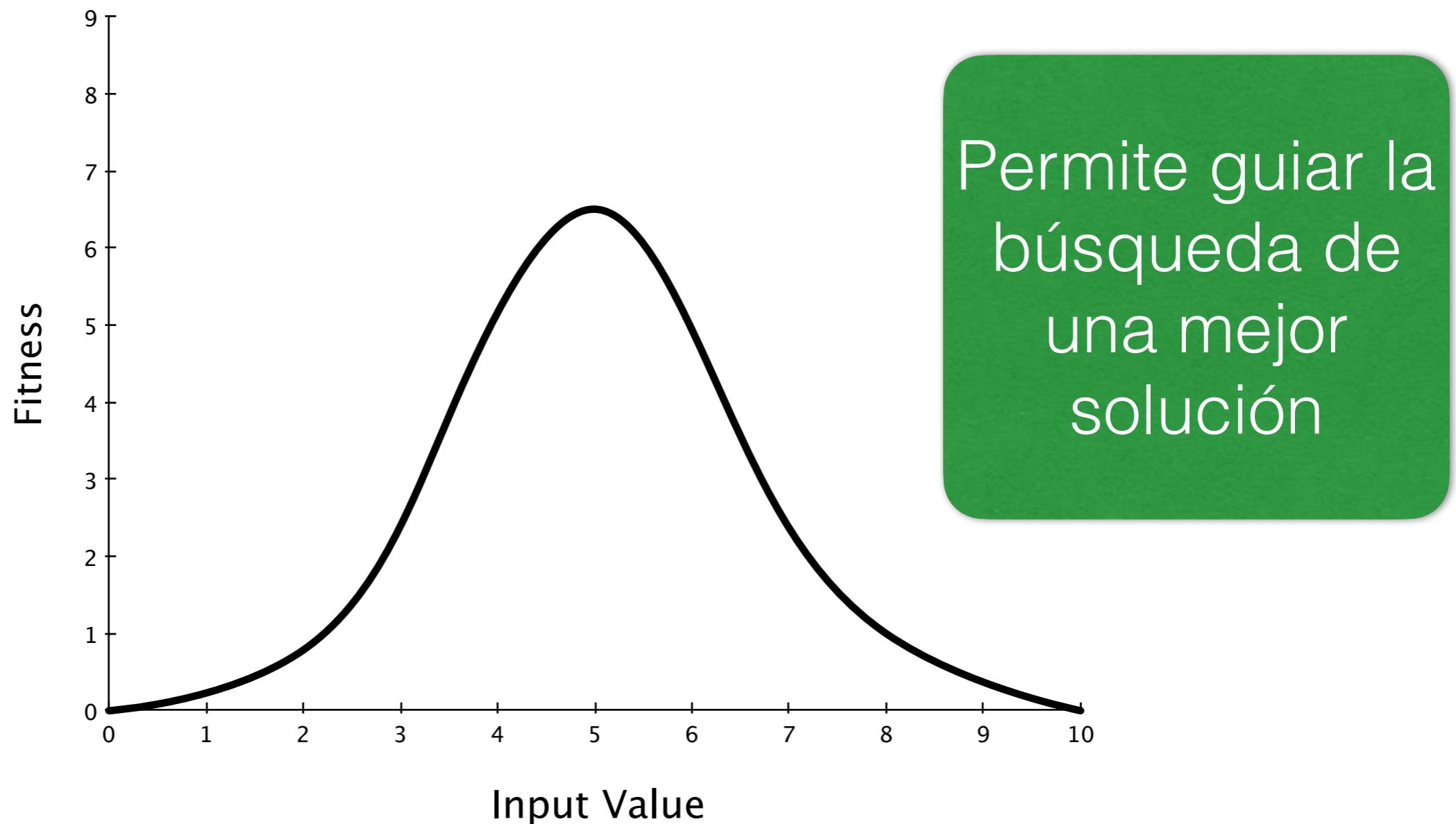
# Fitness Function

- Mide cuán bueno es un individuo como solución
- ¿Es el individuo A mejor que el individuo B?
- Cuantifica la “**bondad**” de un individuo
- La Fitness Function es específica al problema
  - Determina el “paisaje de búsqueda” (fitness landscape)  
*firma del profundo*

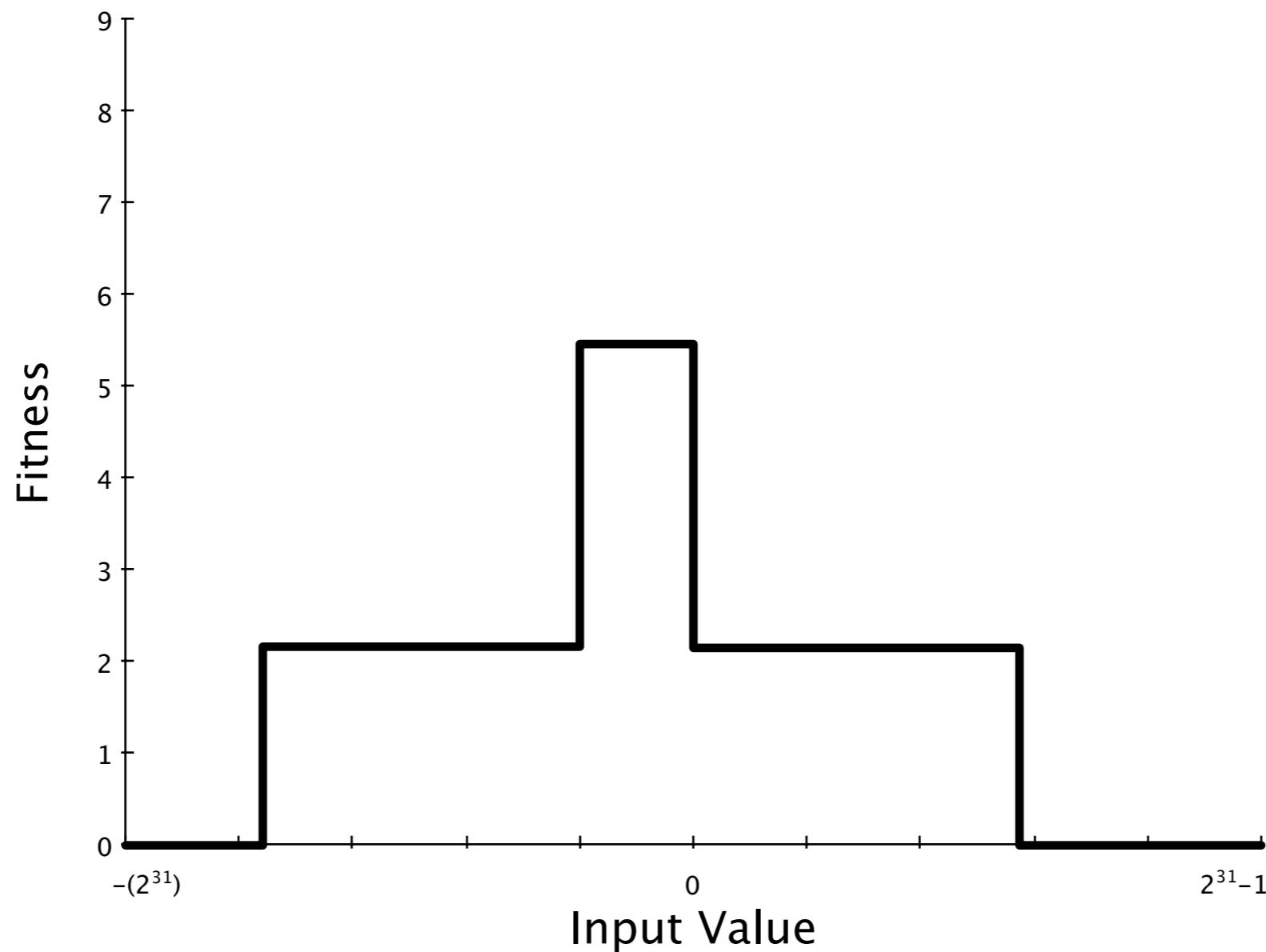
# Smooth Fitness Landscape



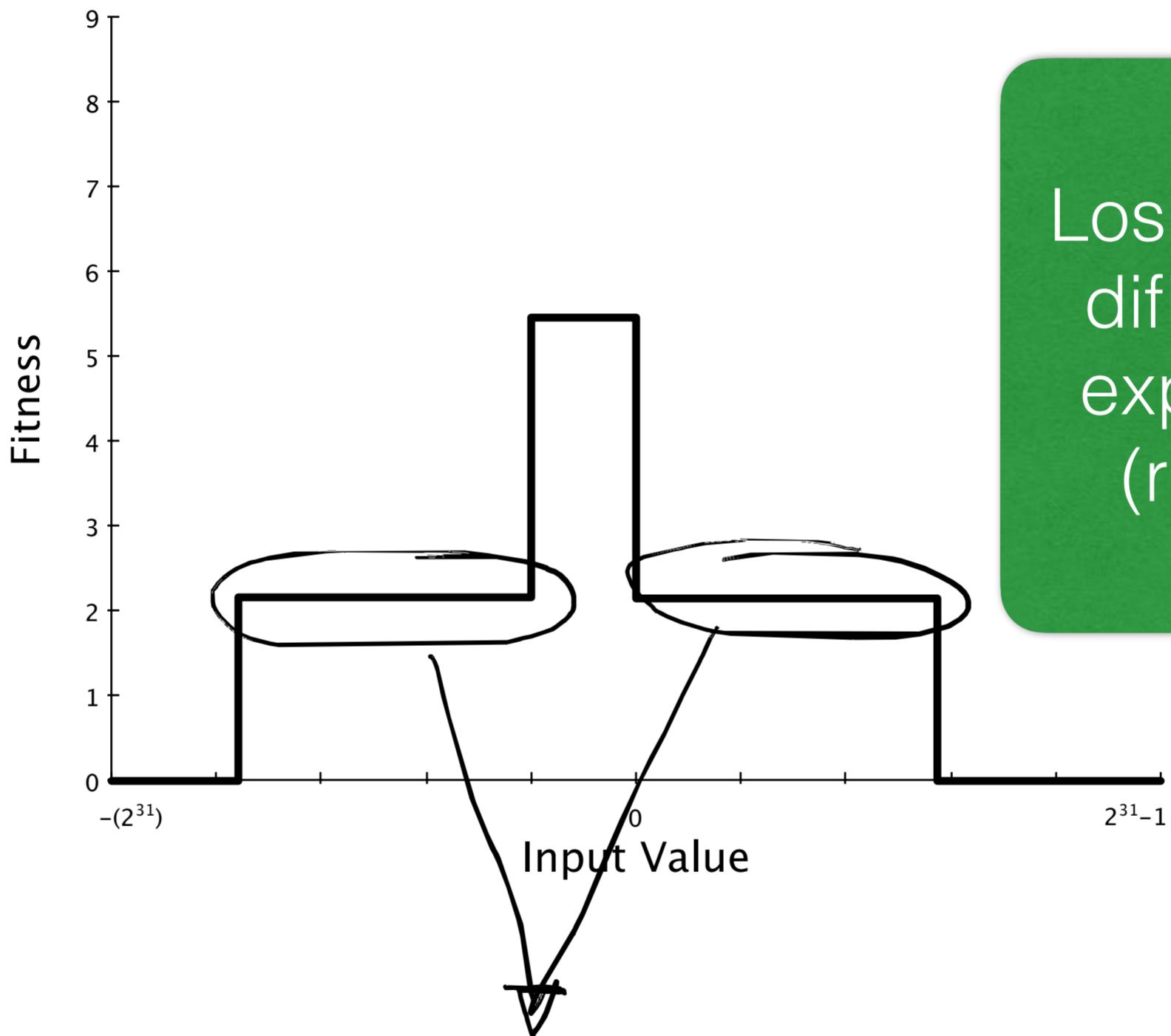
# Smooth Fitness Landscape



# Rugged Fitness Landscape



# Rugged Fitness Landscape



Los **quiebres** dificultan la exploración (random)

mucho del dominio  
con un clima moltíssimas  
→ reaniego o randallendo

# Heurística Branch Distance

- Branch Distance = distancia del predicado a ejercitar el branch (cuantos ceros tenemos de cubrir el branch)
- Puede ser el TRUE o el FALSE

```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True //branch a cubrir  
    else:  
        return False
```

variables de

# Branch Distance

planteamiento cores

$a == b$

$\text{abs}(a-b) = 0 ? 0 : \text{abs}(a-b)$

$a < b$

$a < b ? 0 : (a - b) + K$  ; , <sup>si  $a=b, a-b=0$</sup>  <sub>ejemplo.</sub>

$a <= b$

$a <= b ? 0 : (a - b)$

$a > b$

$a > b ? 0 : (b - a) + K$

$a >= b$

$a >= b ? 0 : (b - a)$

más cosas de

# Branch Distance

$\text{o.m}(v_1, \dots, v_n)$

$a \neq b$

$A \&& B$

$A \parallel B$

$!a$

true ? 0 : K

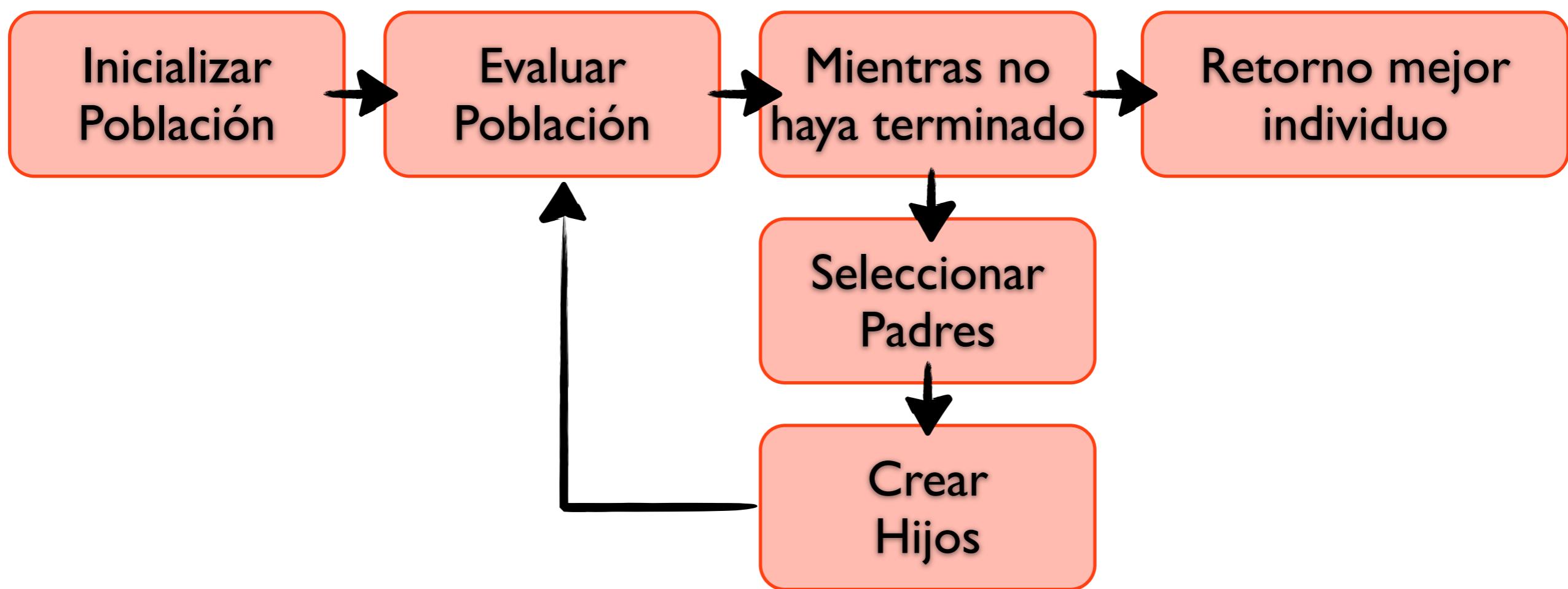
$a \neq b ? 0 \text{ else } K$

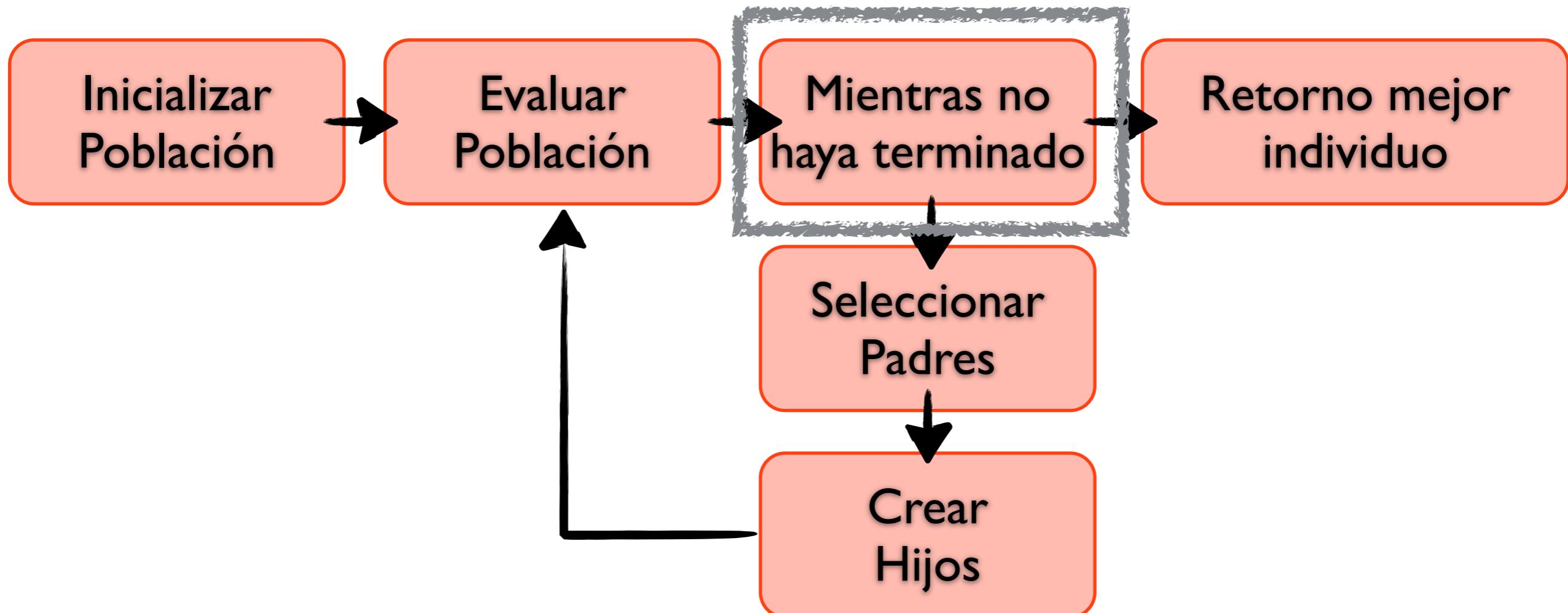
$\text{distance}(A) + \text{distance}(B)$

$\min(\text{distance}(A), \text{distance}(B))$

Aplicar De Morgan

the flow  
problem

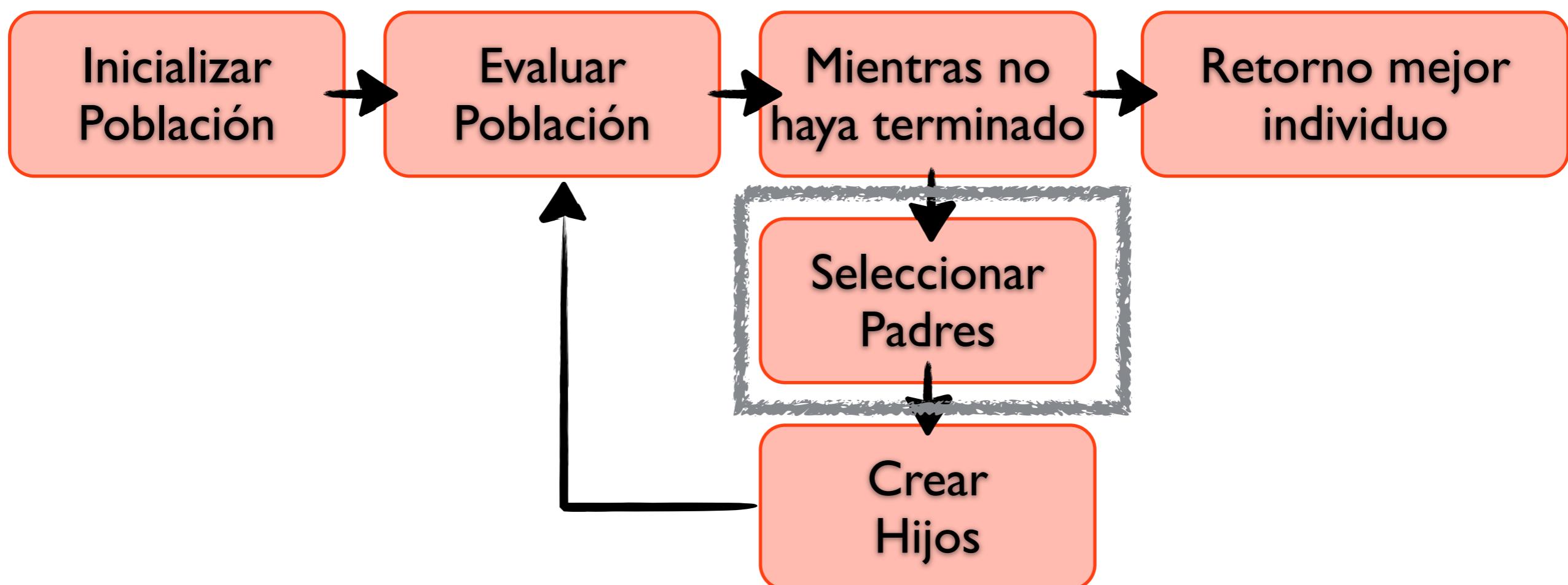




# Opciones de Condición de Parada

- Algunas posibles opciones:
  - Tiempo Máximo
  - Cantidad máxima de iteraciones
  - Cantidad máxima de evaluaciones del fitness de individuos



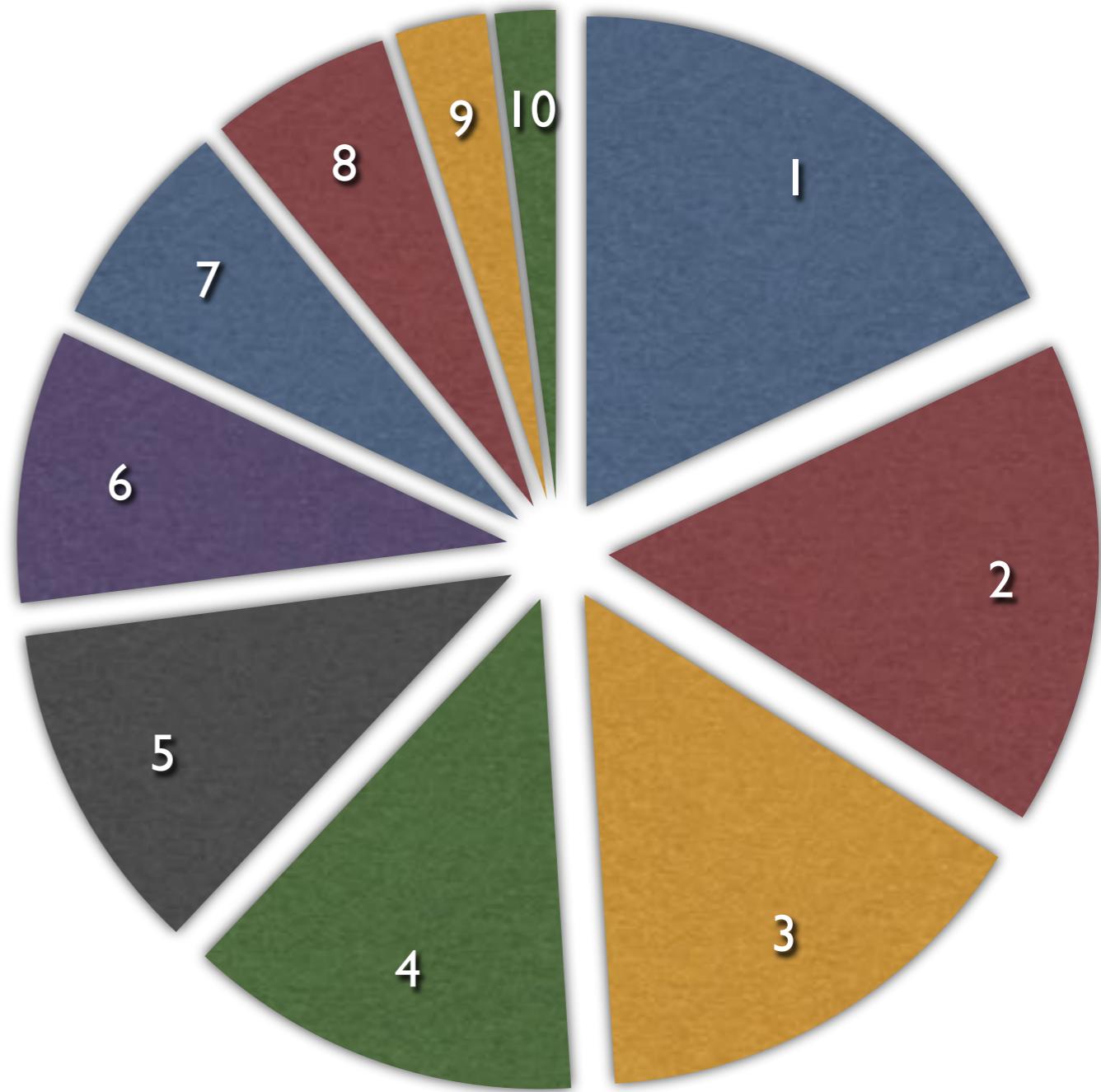


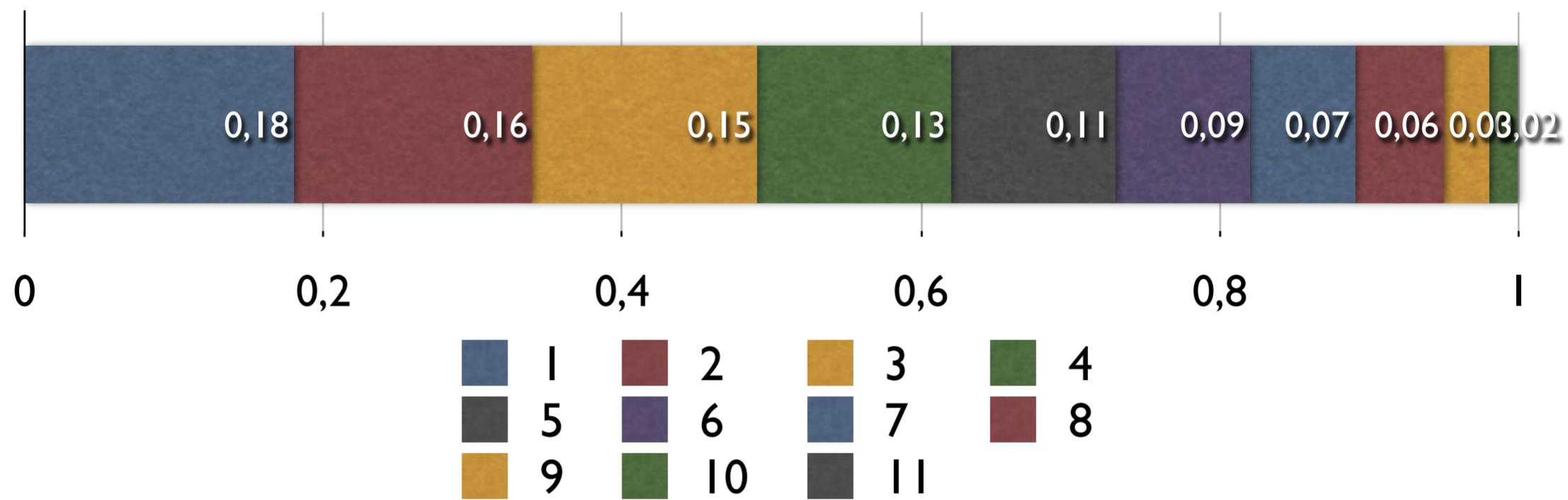
# Fitness Proportionate Selection

- A.k.a. Roulette wheel selection
- La Probabilidad de elegir un individuo es proporcional a su valor de fitness

Individuo	Fitness
1	2
2	1,8
3	1,6
4	1,4
5	1,2
6	1
7	0,8
8	0,6
9	0,4
10	0,2
11	0

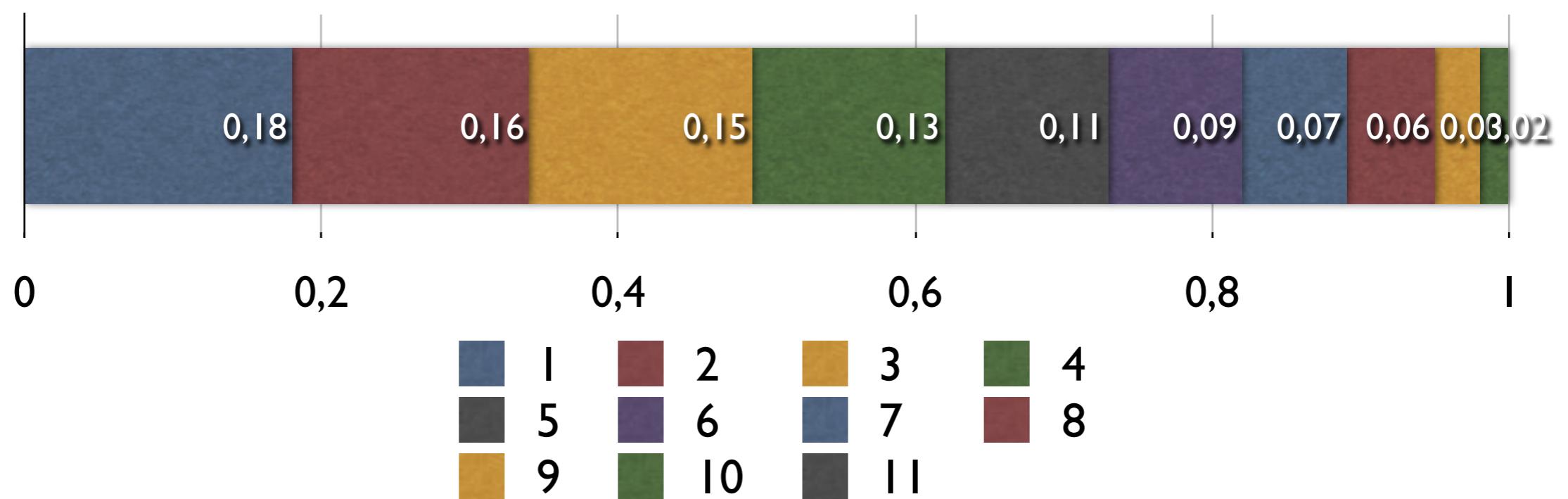
Individuo	Fitness
1	2
2	1,8
3	1,6
4	1,4
5	1,2
6	1
7	0,8
8	0,6
9	0,4
10	0,2
11	0



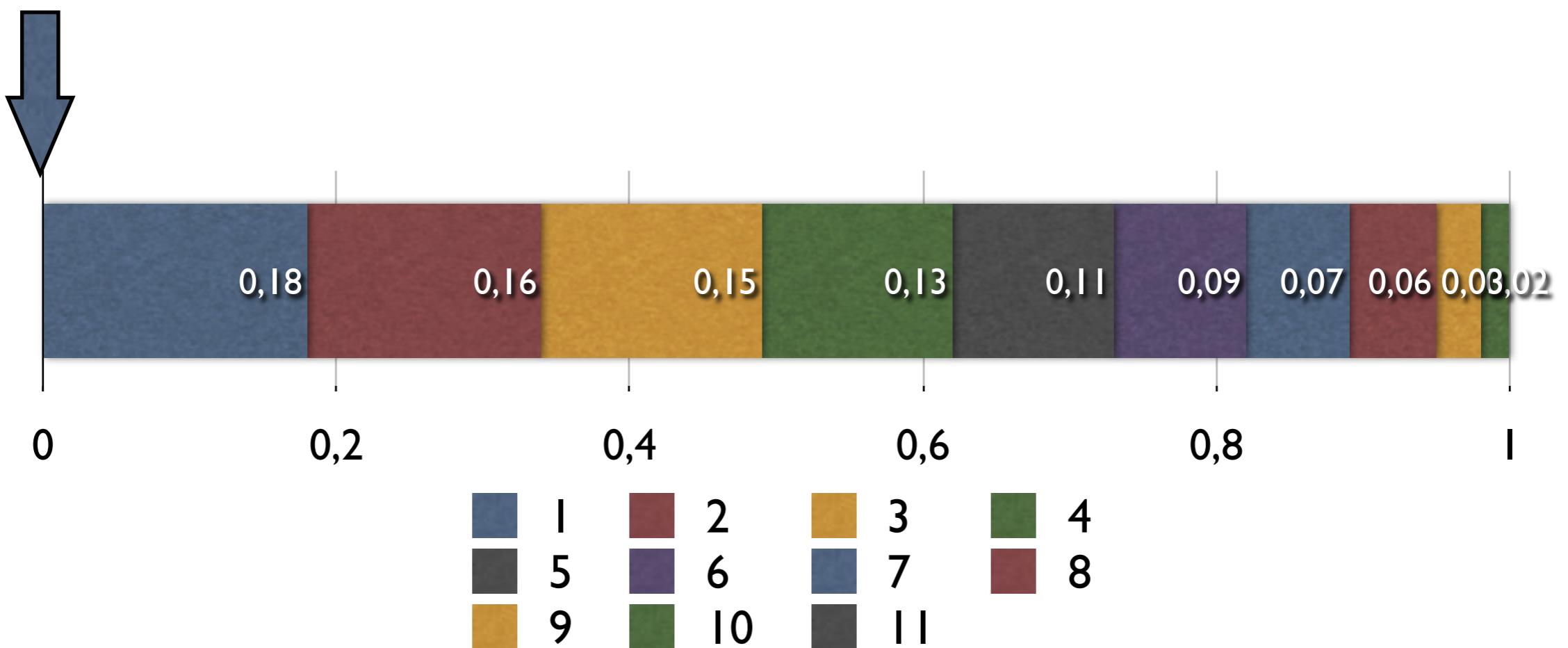


anjoas raupas a cf individuos  
en dependencia de su valor de fitness  
y haga lo mismo o ver  
que raupa es.

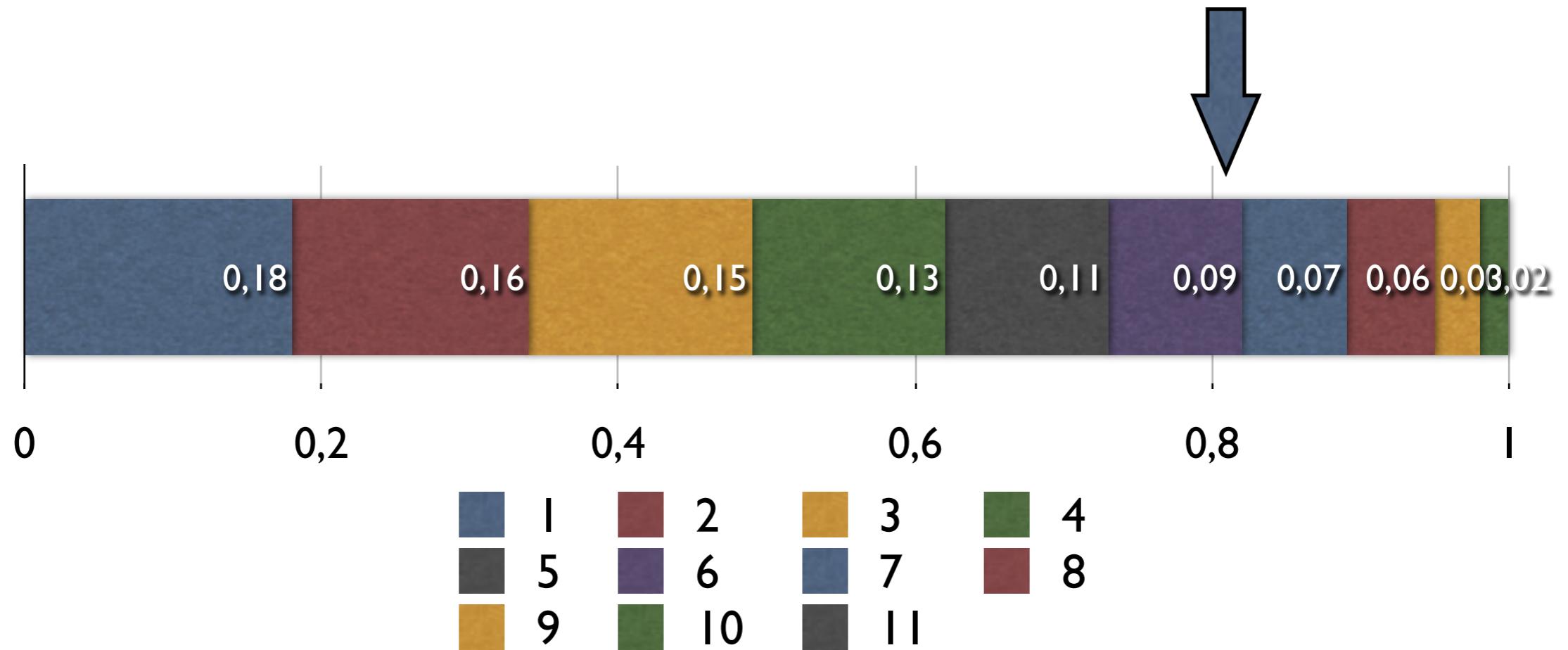
- Valor aleatorio: 0,81



- Valor aleatorio: 0,81

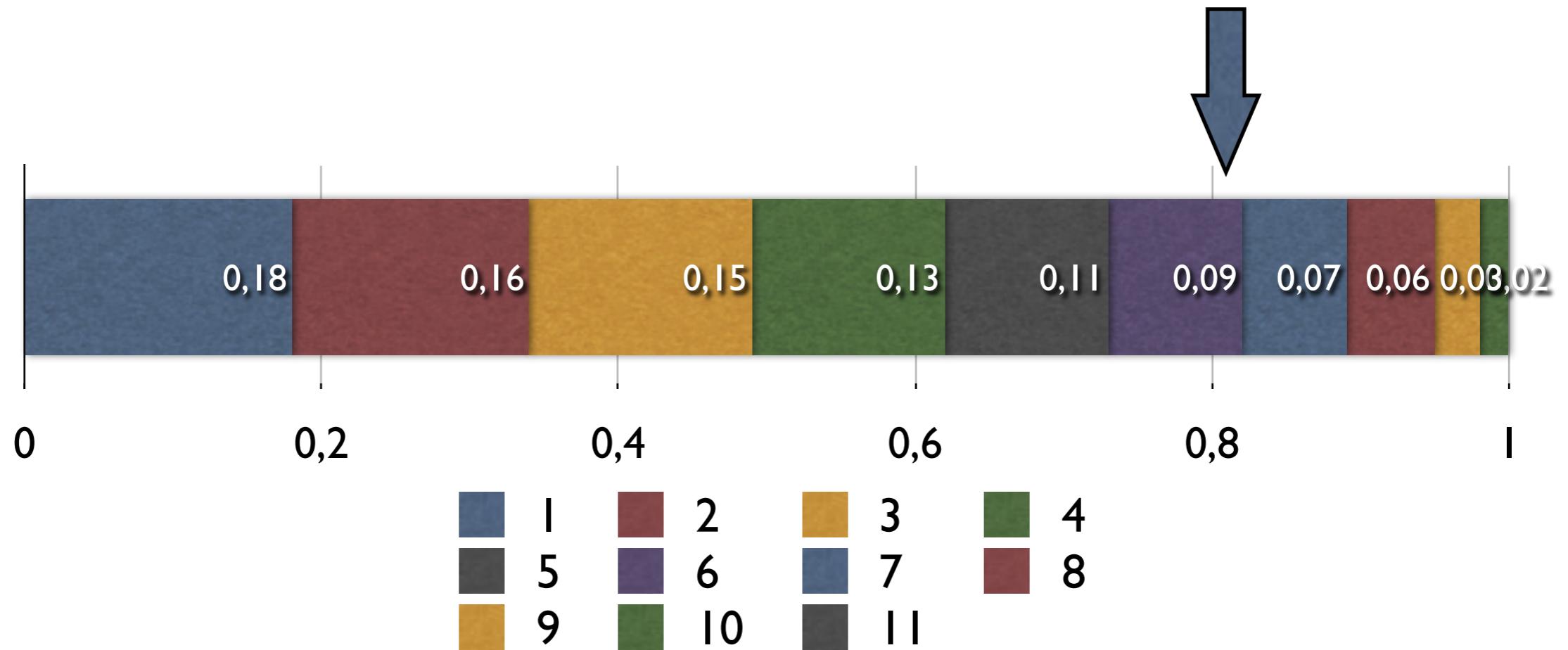


- Valor aleatorio: 0,81



- Valor aleatorio: 0,81

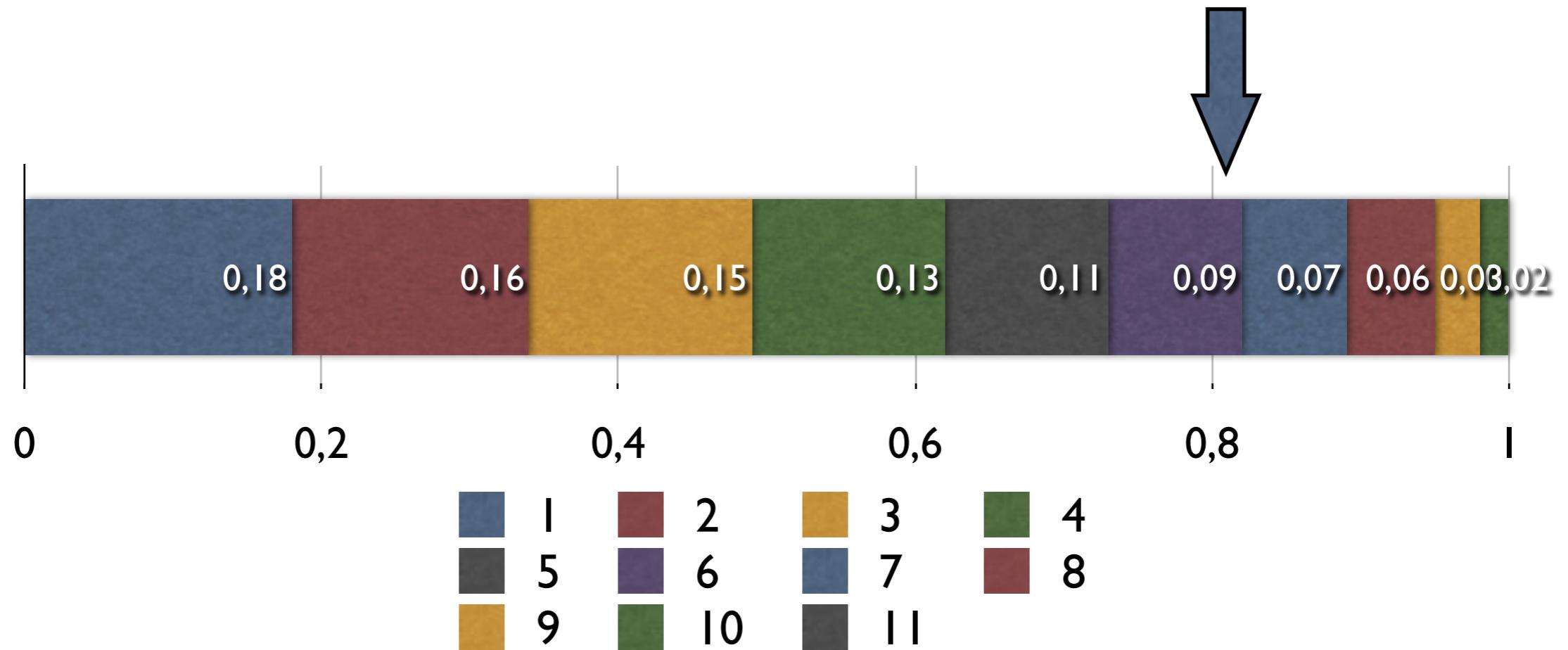
6



6

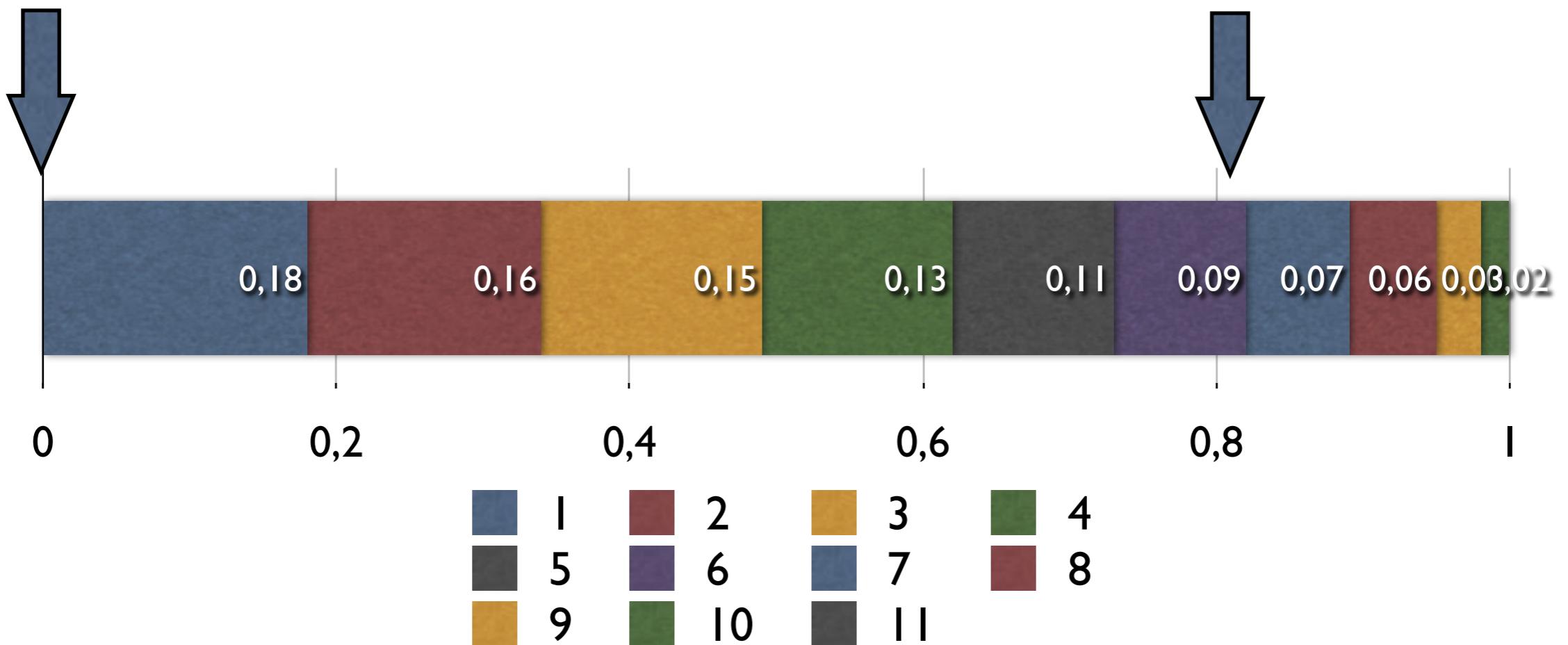
- Valor aleatorio: 0,81

- Valor aleatorio: 0,32



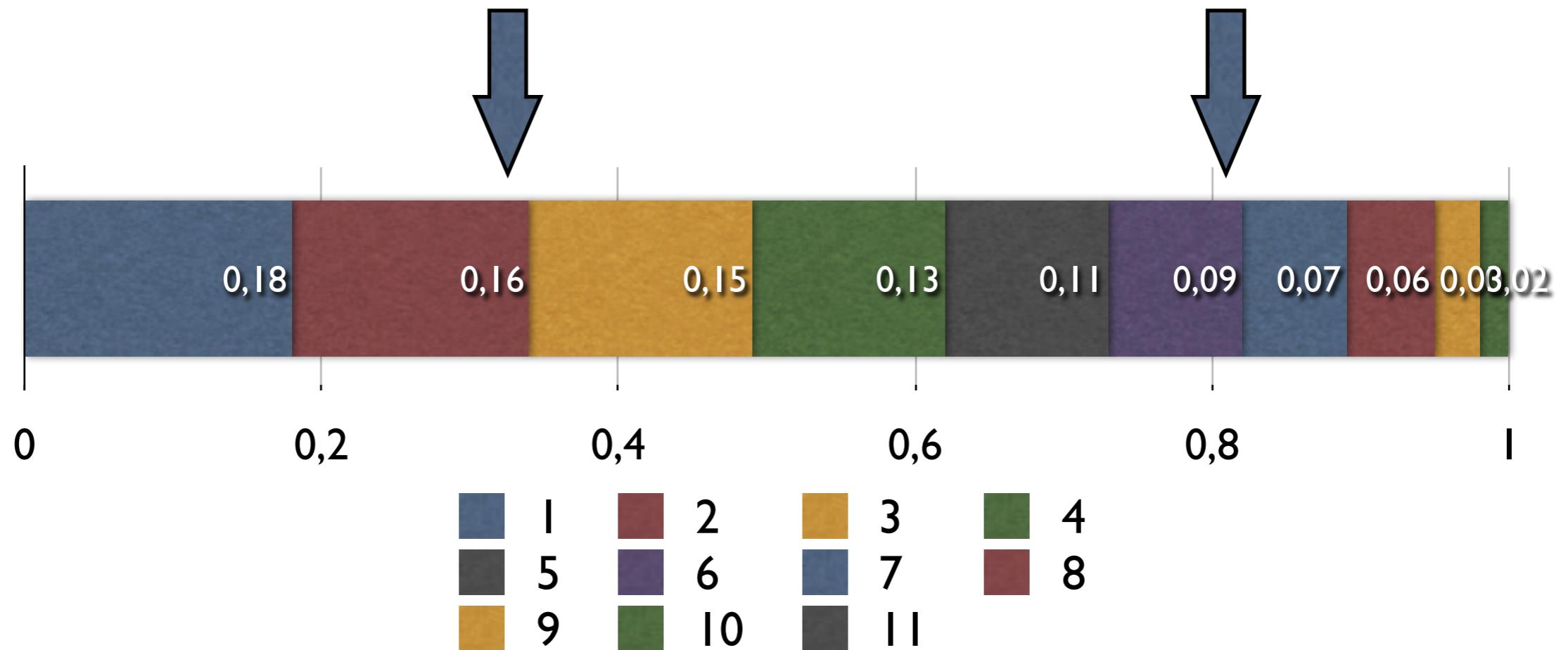
- Valor aleatorio: 0,81

- Valor aleatorio: 0,32

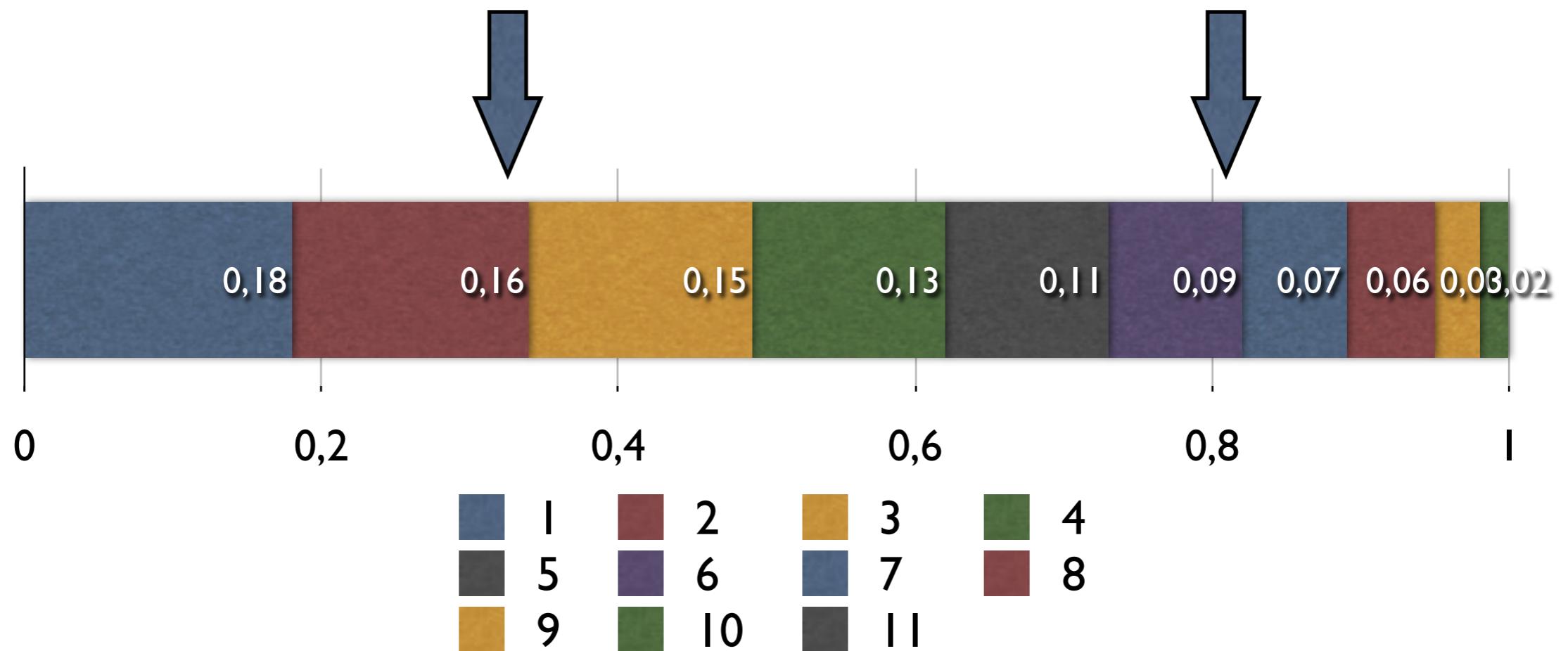


- Valor aleatorio: 0,81

- Valor aleatorio: 0,32



- Valor aleatorio: 0,81
- Valor aleatorio: 0,32

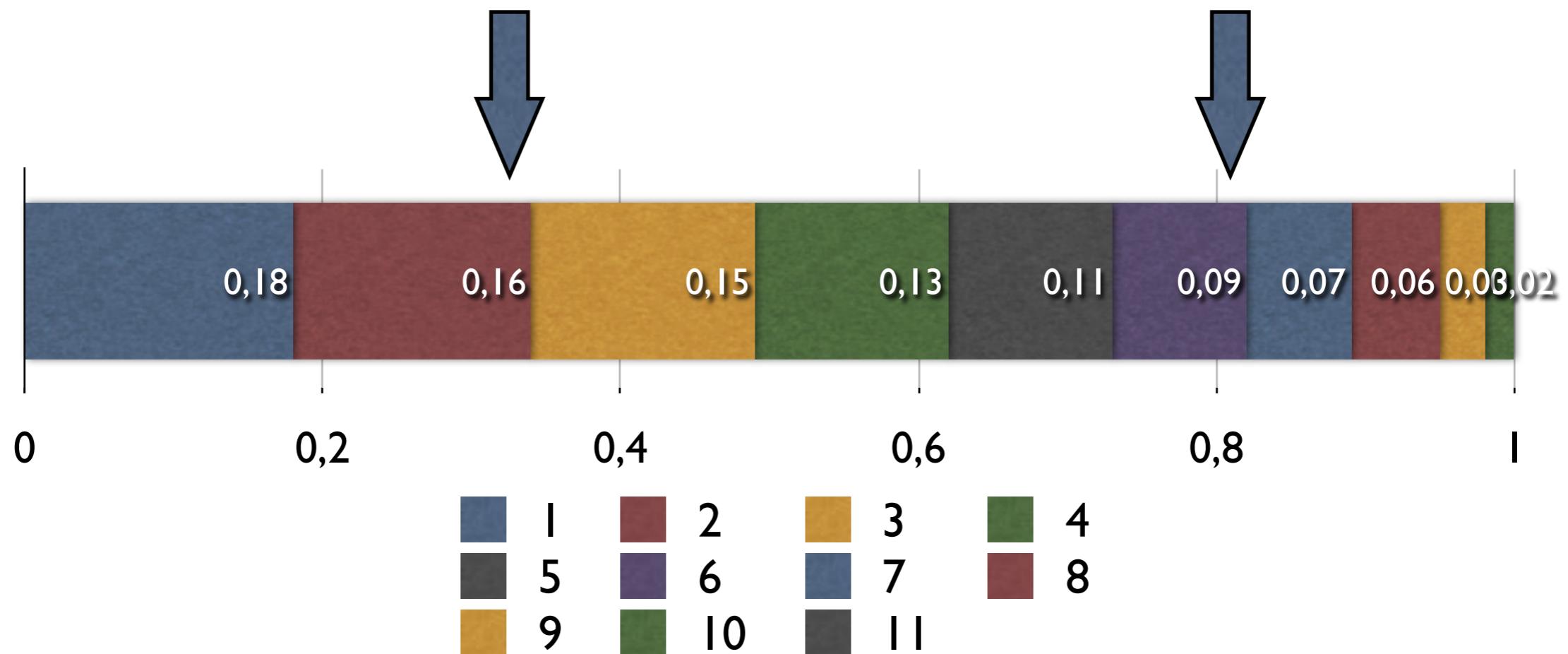


- Valor aleatorio: 0,81



- Valor aleatorio: 0,32

- Valor aleatorio: 0,01

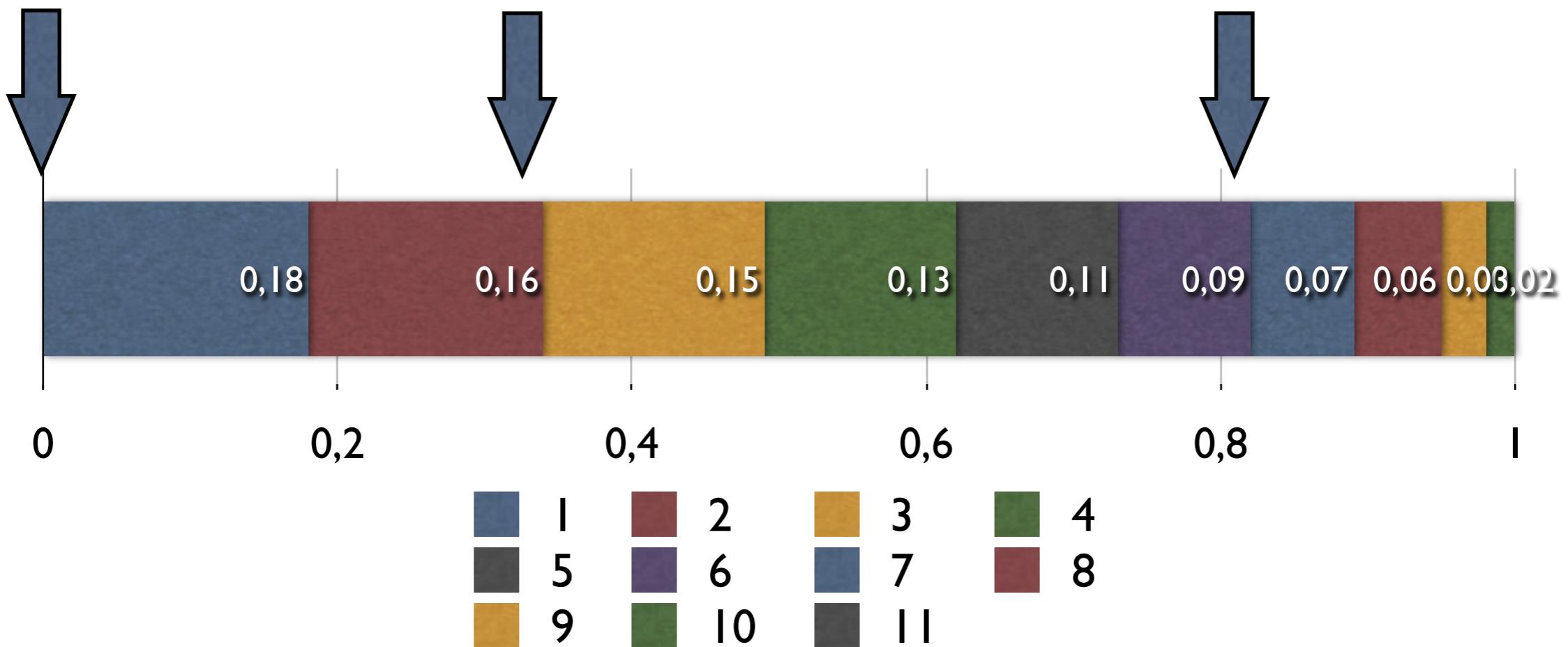


- Valor aleatorio: 0,81



- Valor aleatorio: 0,32

- Valor aleatorio: 0,01

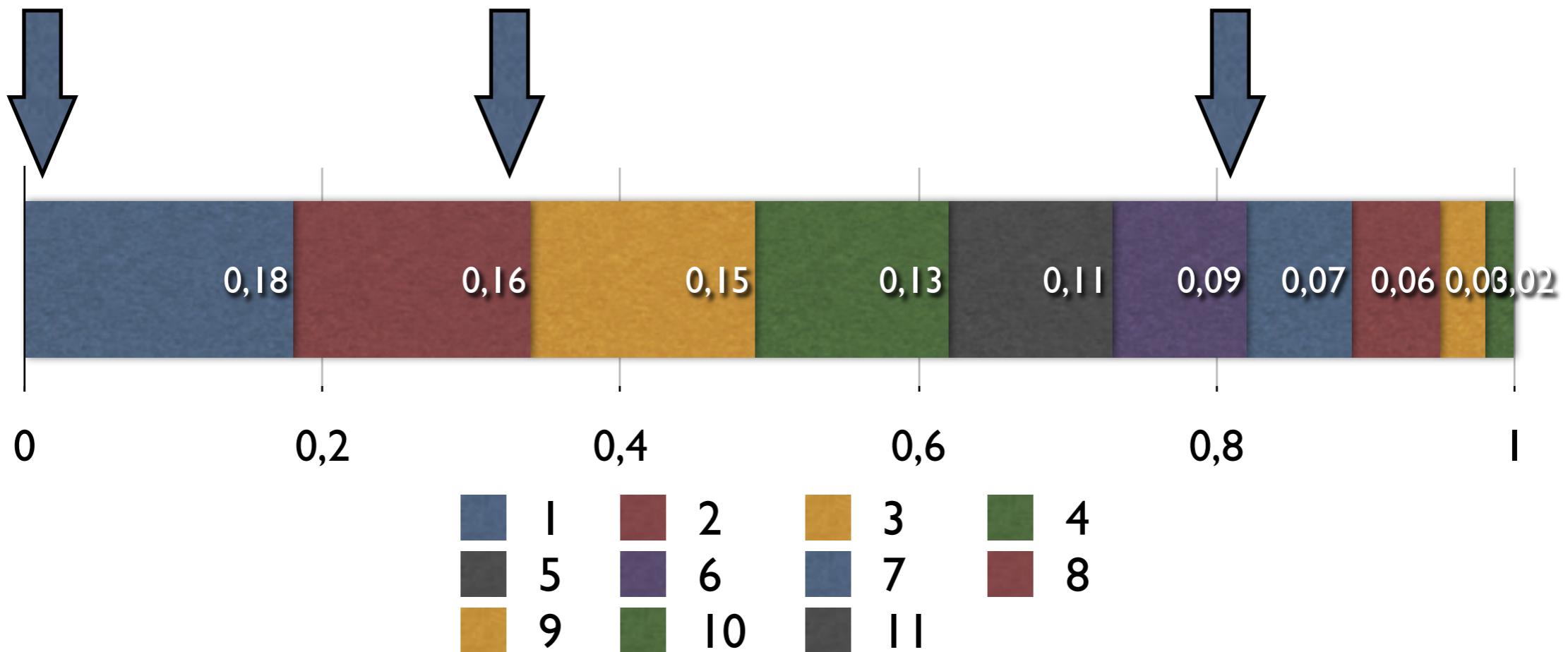


- Valor aleatorio: 0,81

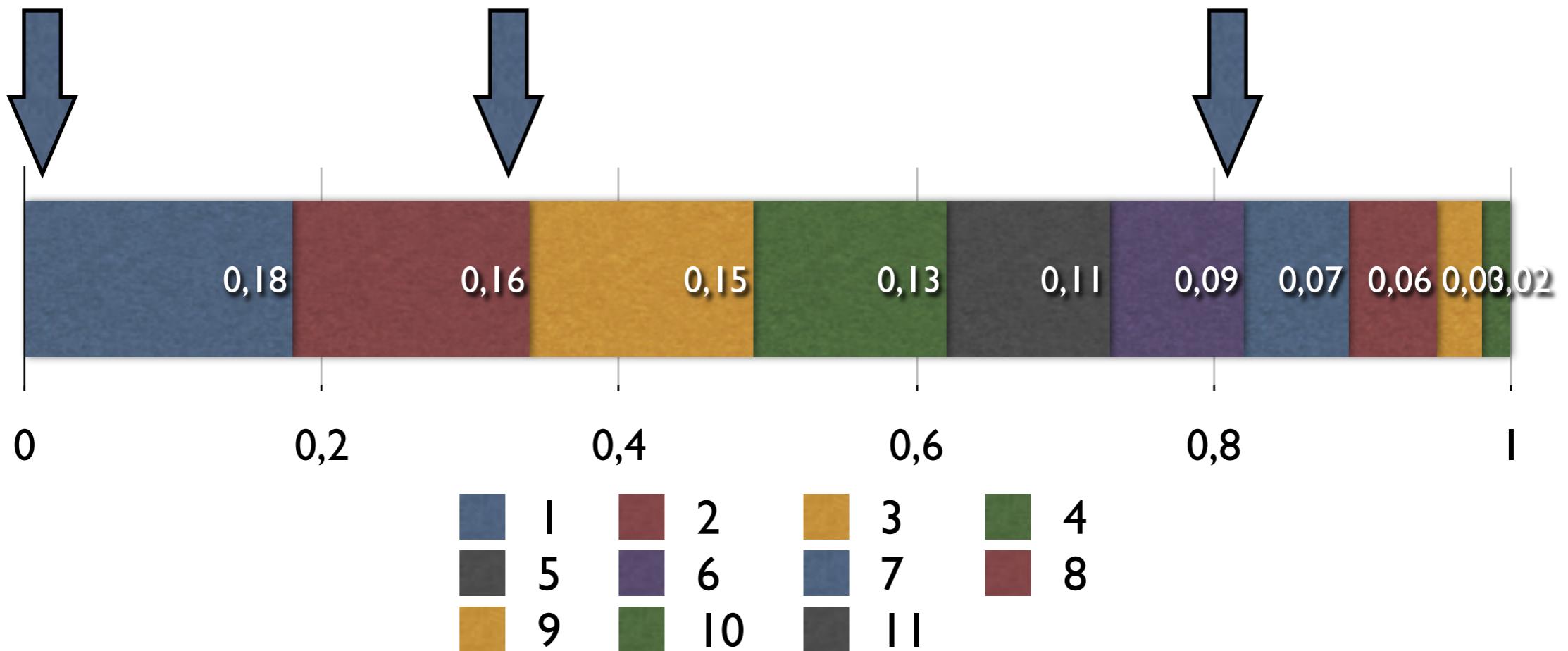


- Valor aleatorio: 0,32

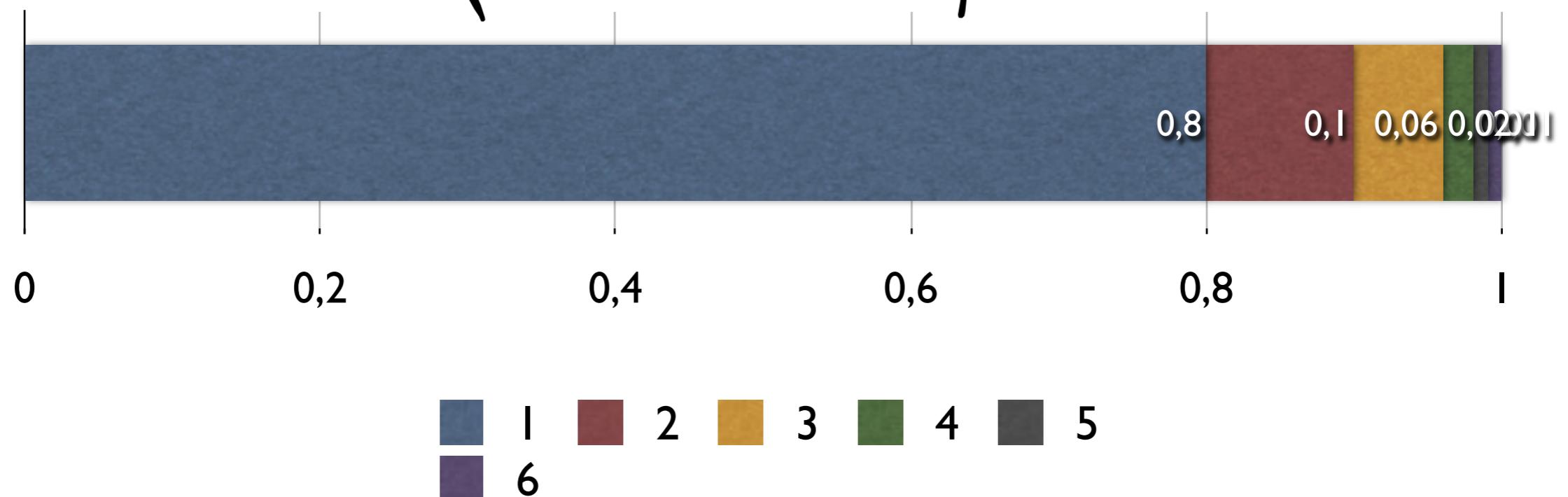
- Valor aleatorio: 0,01

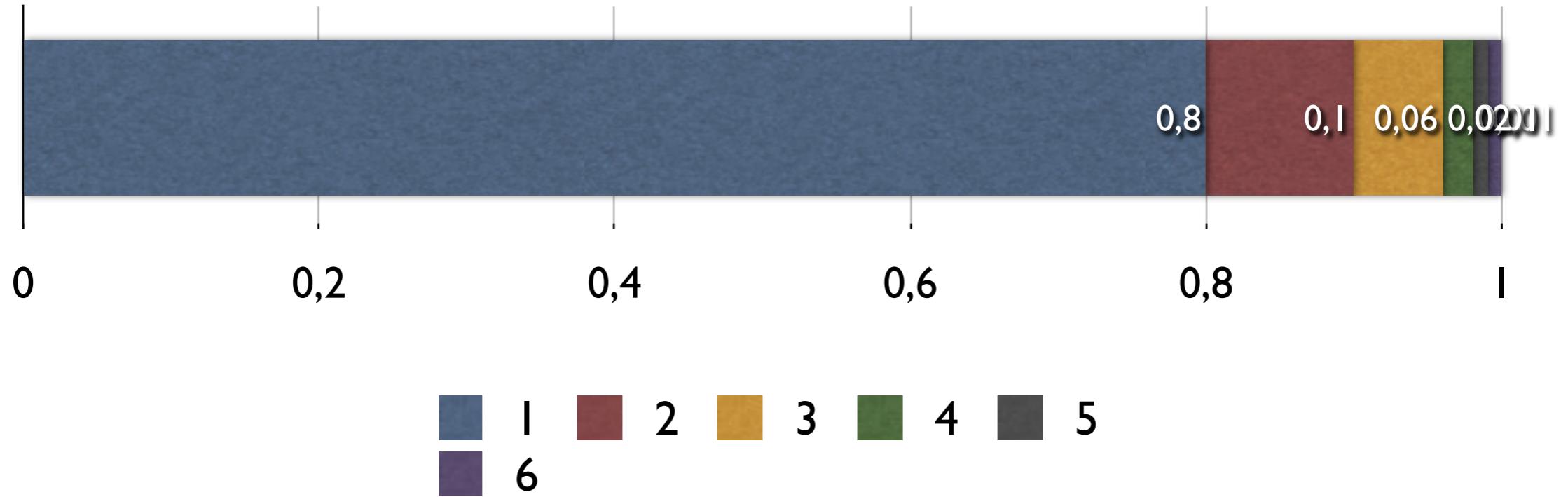


- Valor aleatorio: 0,81
- Valor aleatorio: 0,32
- Valor aleatorio: 0,01

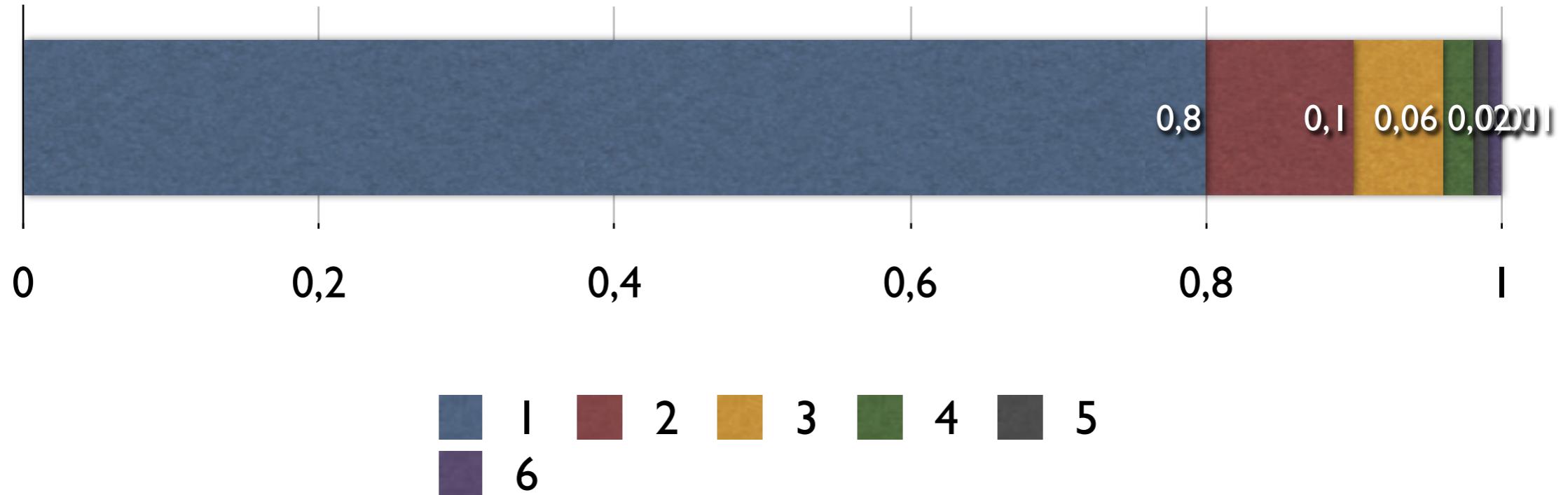


rendido de diversidad genética

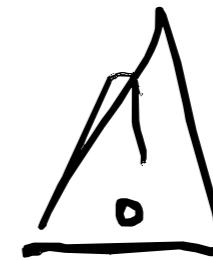




- El Individuo I dominará la selección



- El Individuo I dominará la selección
- **¡No use la rueda de ruleta!**

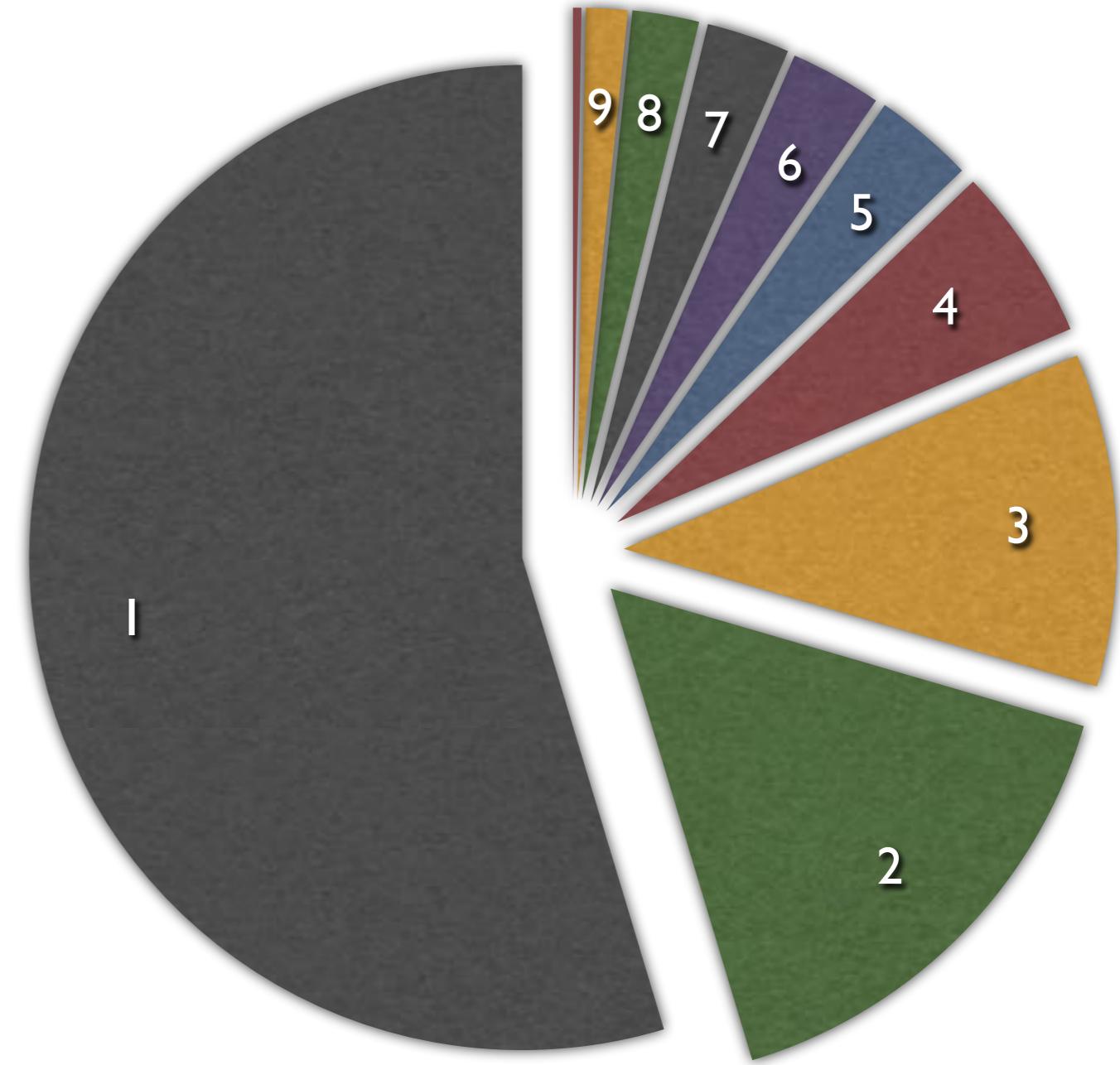


# Rank Selection

- Rankear individuos de acuerdo a su fitness
- No hay diferencia si el individuo con mayor fitness es diez veces mejor que el segundo o únicamente el 0.001% mejor
- Rank Selection es preferible en la práctica

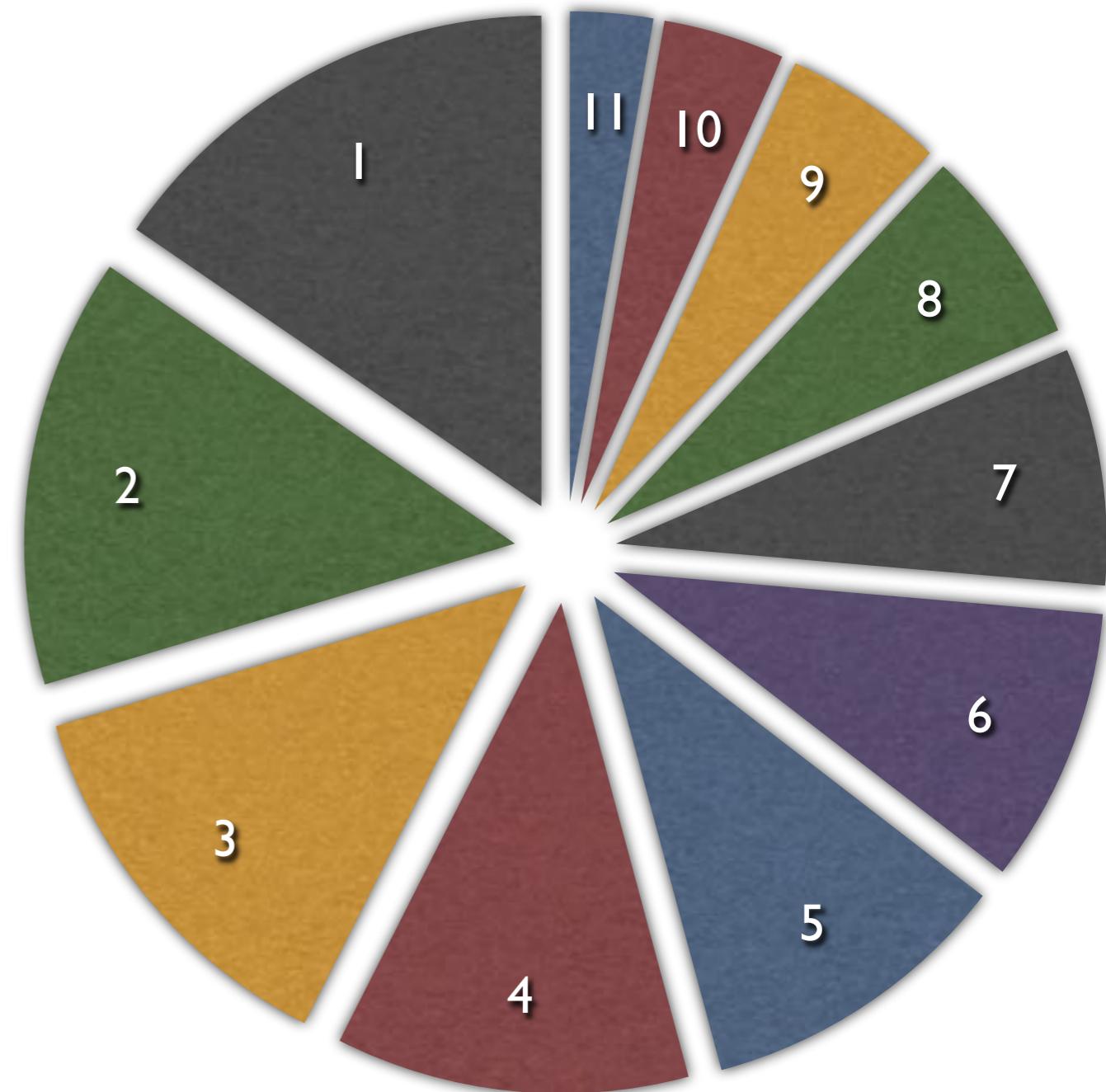


Individuo	Fitness
1	2
2	0,58
3	0,4
4	0,21
5	0,12
6	0,11
7	0,1
8	0,08
9	0,05
10	0,01
11	0



onjunto probabilidades x  
 range o grupo → roulette  
 selection slot

Individuo	Fitness
1	2
2	0,58
3	0,4
4	0,21
5	0,12
6	0,11
7	0,1
8	0,08
9	0,05
10	0,01
11	0



menor probabilidad p/ los  
 del menor grupo

# Tournament Selection

- $N$  = Tamaño del Torneo
- Seleccionar  $N$  individuos aleatoriamente
- El mejor de los  $N$  individuos es seleccionado
- El Tamaño del Torneo define la presión selectiva
- Un individuo con peor fitness aún puede vencer con una cierta probabilidad



Individuo	Fitness
1	2
2	1,8
3	1,6
4	1,4
5	1,2
6	1
7	0,8
8	0,6
9	0,4
10	0,2
11	0

Individuo	Fitness
1	2
2	1,8
3	1,6
4	1,4
5	1,2
6	1
7	0,8
8	0,6
9	0,4
10	0,2
11	0

Tamaño Torneo: 4

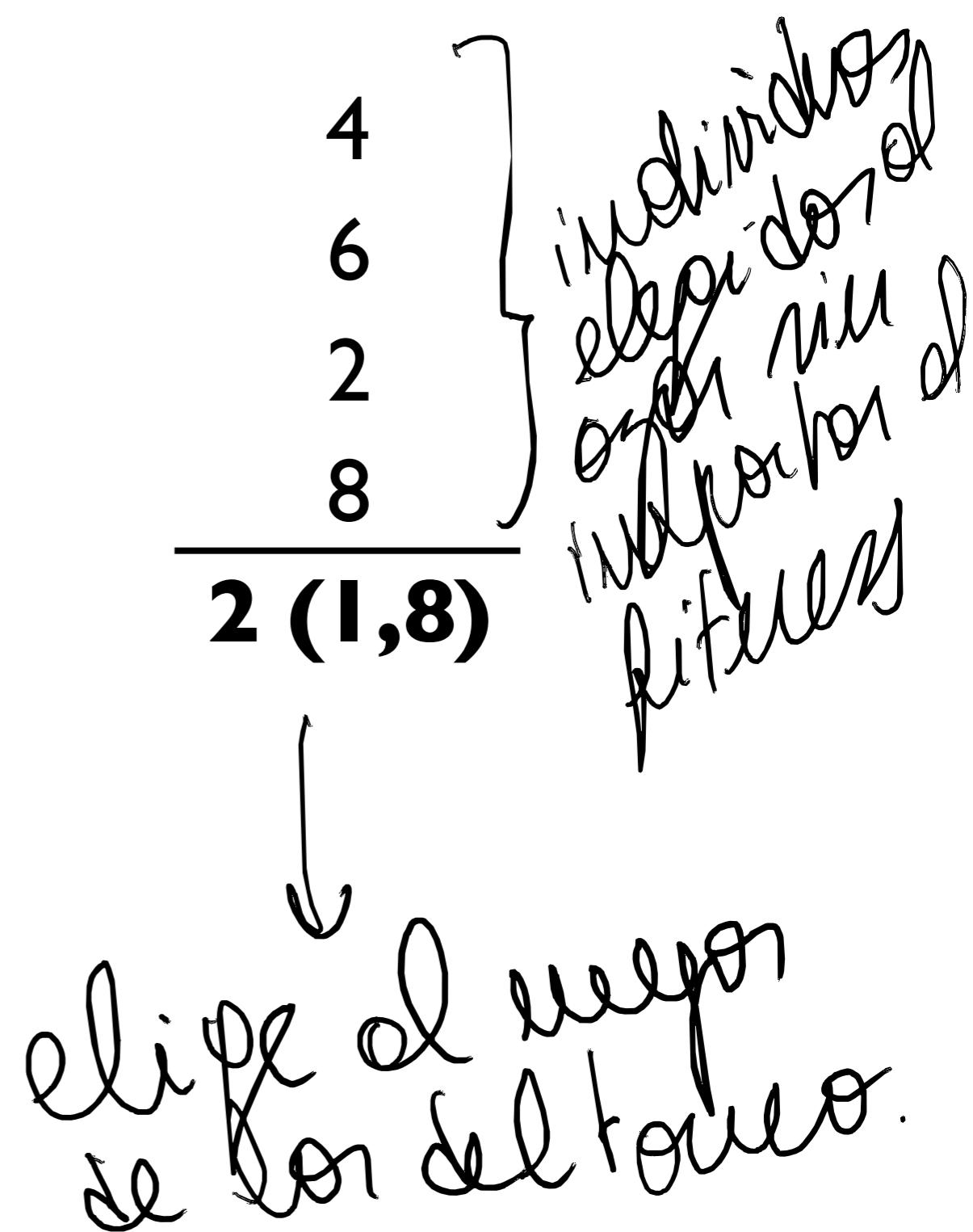
**Tamaño Torneo: 4**

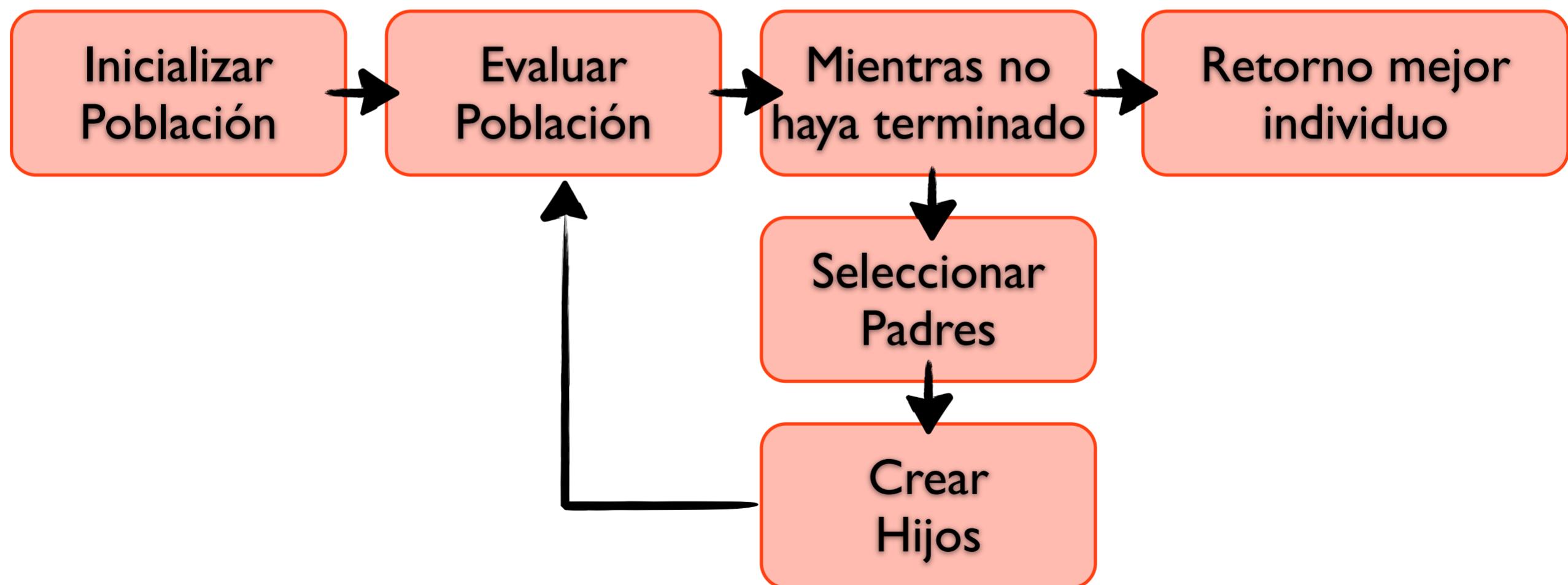
4  
6  
2  
8

Individuo	Fitness
1	2
2	1,8
3	1,6
4	1,4
5	1,2
6	1
7	0,8
8	0,6
9	0,4
10	0,2
11	0

Individuo	Fitness
1	2
2	1,8
3	1,6
4	1,4
5	1,2
6	1
7	0,8
8	0,6
9	0,4
10	0,2
11	0

Tamaño Torneo: 4







# Crossover

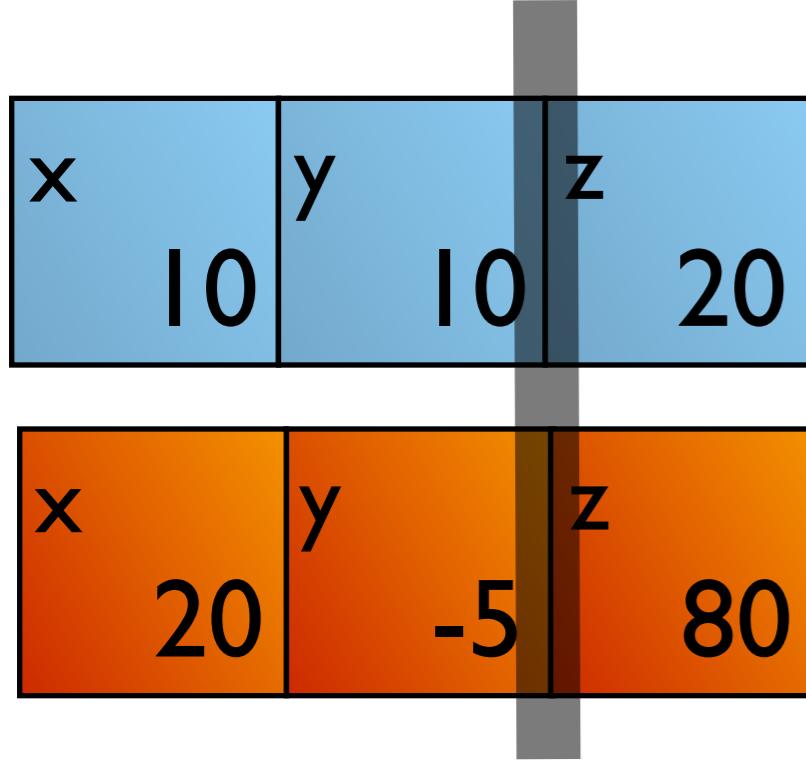
```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True //branch a cubrir  
    else:  
        return False
```

x	y	z
10	10	20

x	y	z
20	-5	80

# Crossover

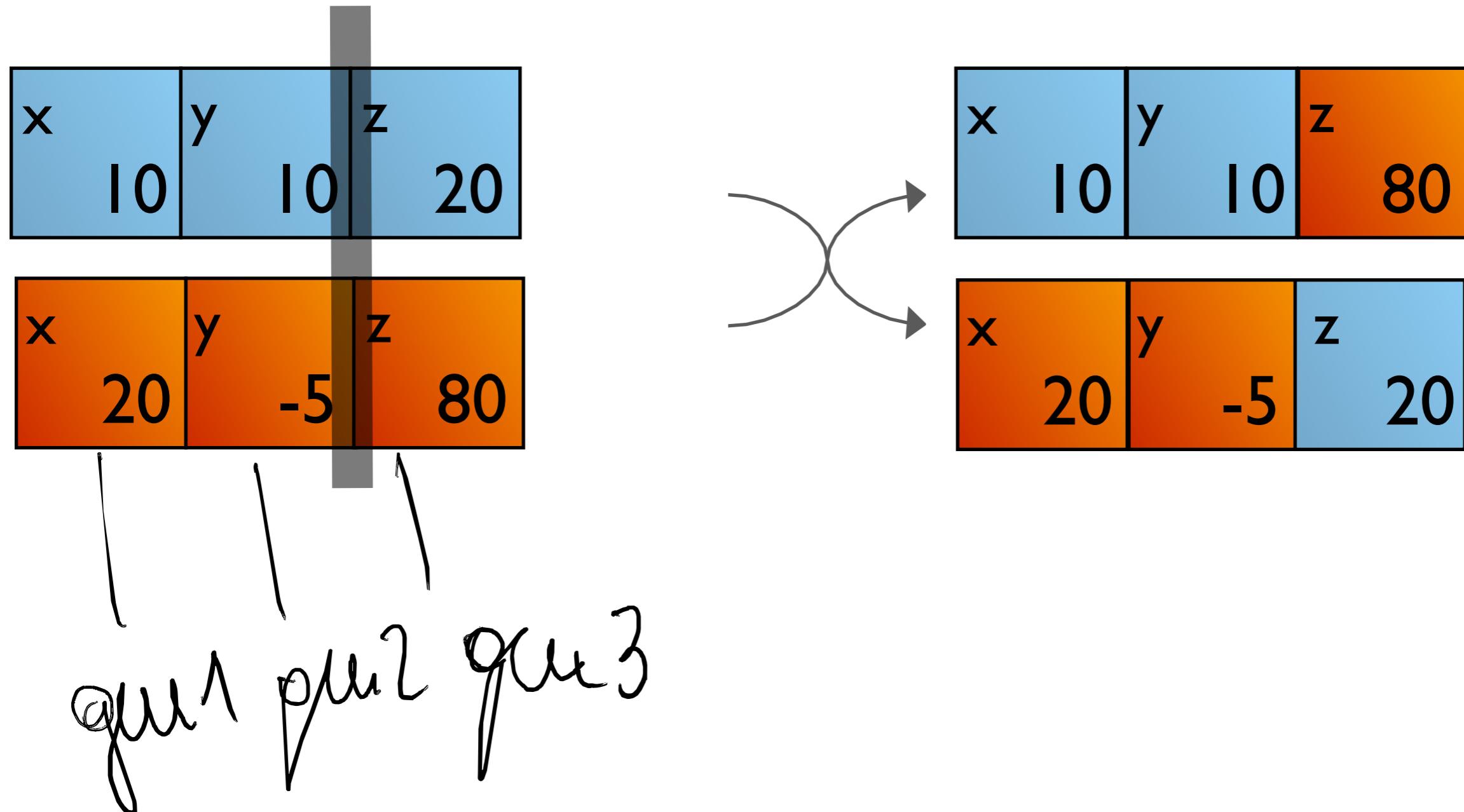
```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True //branch a cubrir  
    else:  
        return False
```



# Crossover

2 padres  
generan 2  
hijos

```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True //branch a cubrir  
    else:  
        return False
```

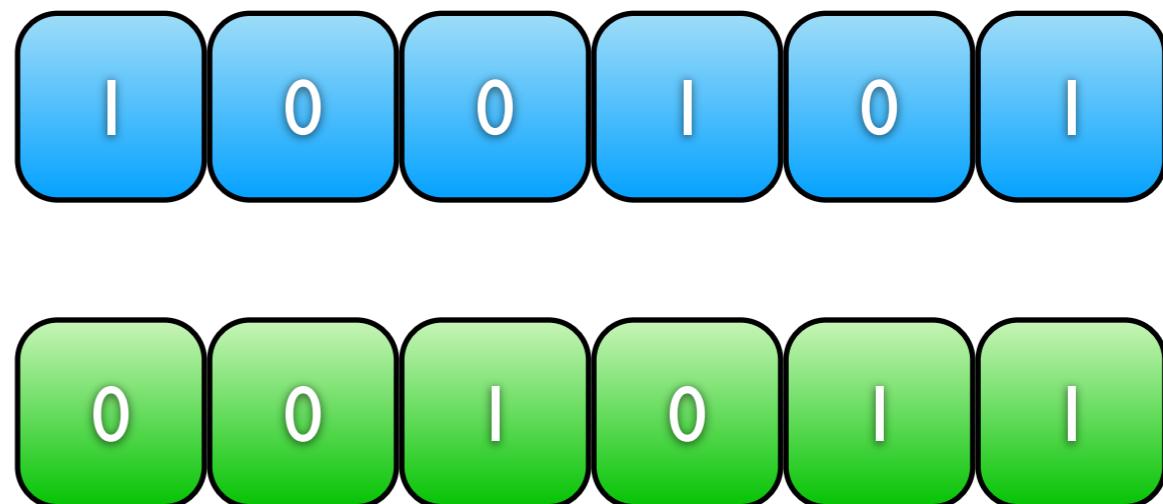


# Fórmulas de Crossover (varios puntos)

- Single-point crossover  
Elegir un único punto en los padres y dividir/unir en ese punto
- Two-point crossover  
Elegir dos puntos en los padres e intercambiar la parte interior
- Fixed vs. variable length  
Igual punto de crossover en ambos padres - descendencia con tamaño constante
- Uniform crossover  
Los Genes son aleatoriamente seleccionados de cada parente

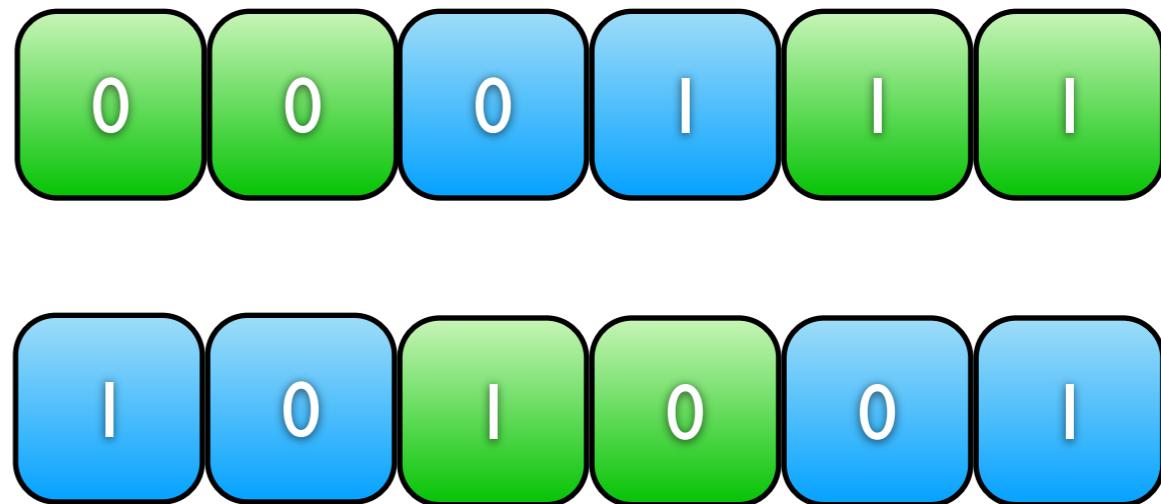
# Crossover

Two-point  
crossover



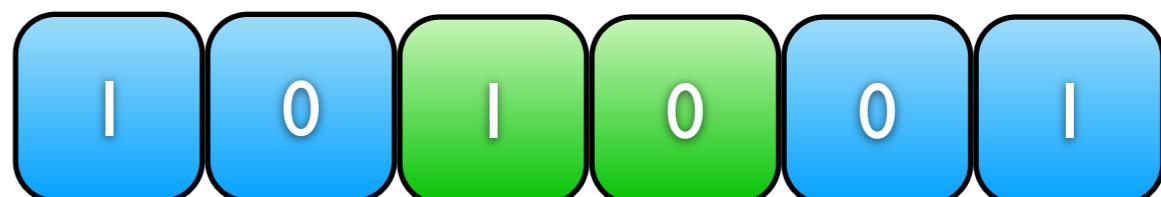
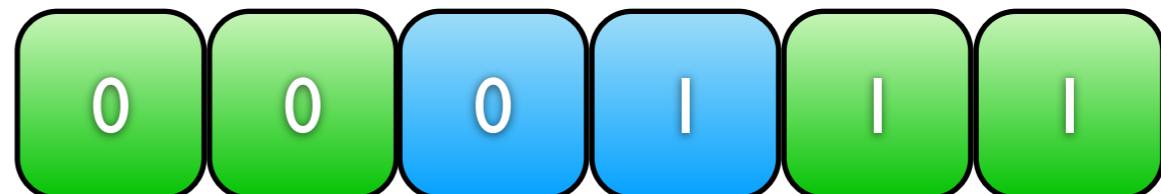
# Crossover

Two-point  
crossover

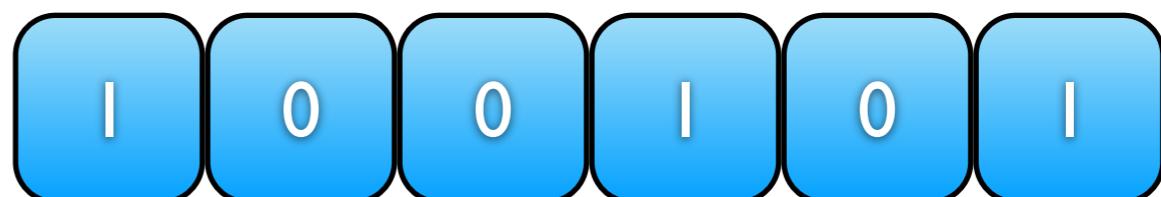


# Crossover

Two-point  
crossover

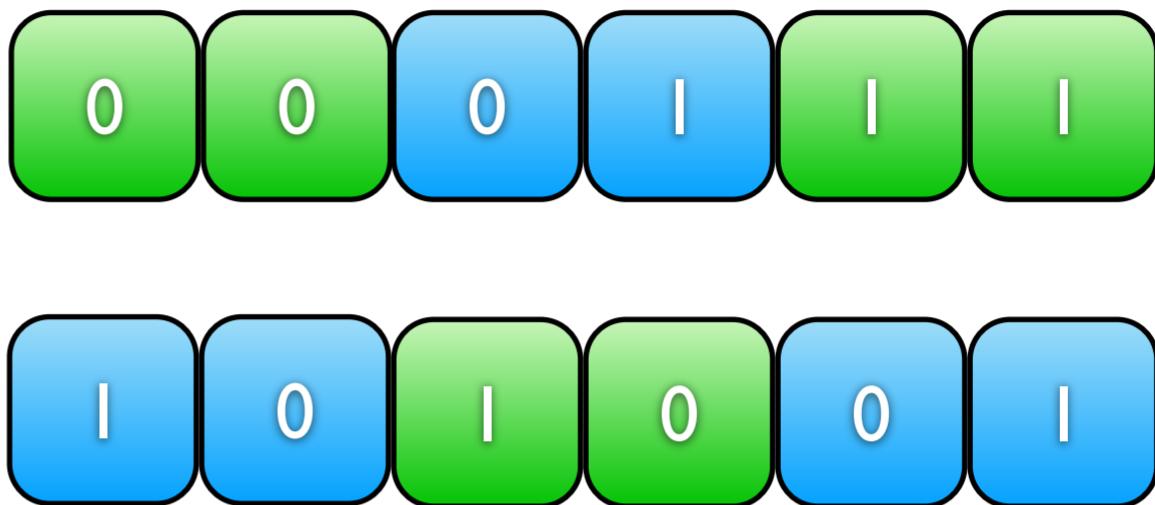


Variable length  
crossover

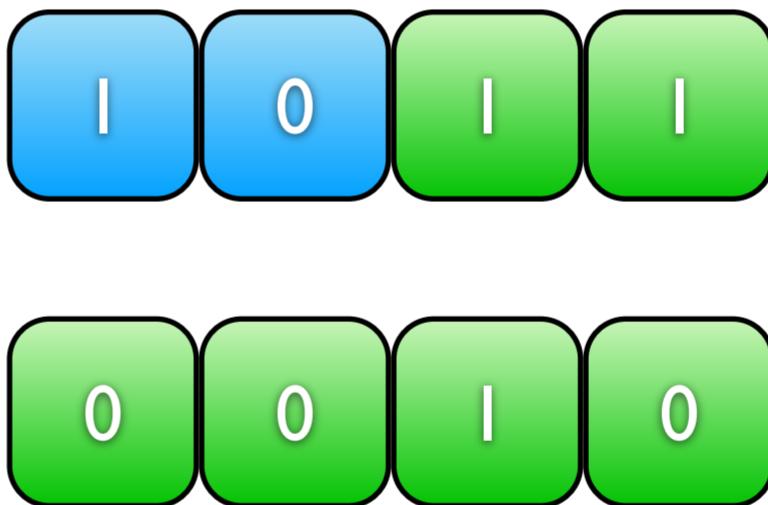


# Crossover

Two-point crossover



Variable length crossover



Necesitas límites para los bloques  
(crecimiento rápido  
de tamaño)

# Mutación

```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True //branch a cubrir  
    else:  
        return False
```

x	y	z
10	10	20

# Mutación

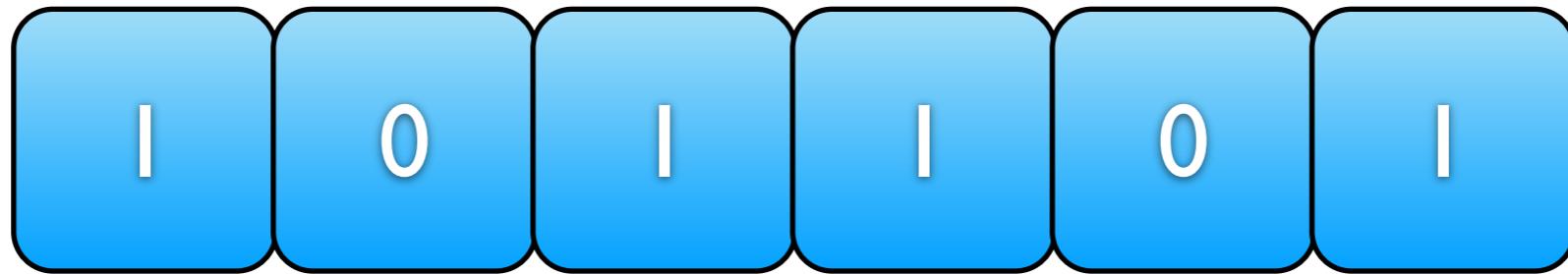
Jepende de  
represa voces  
pl. mutación

```
def testMe(x, y, z):  
    if x * z == 2 * (y + 1):  
        return True //branch a cubrir  
    else:  
        return False
```

↓ invocación, resultado elíctico  
ej: corrijo 1 cero

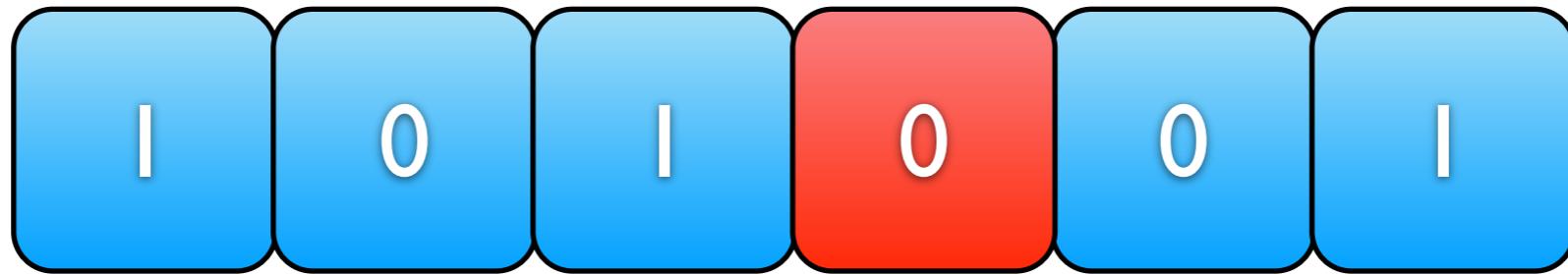
x	y	z
20	10	20

# Mutación



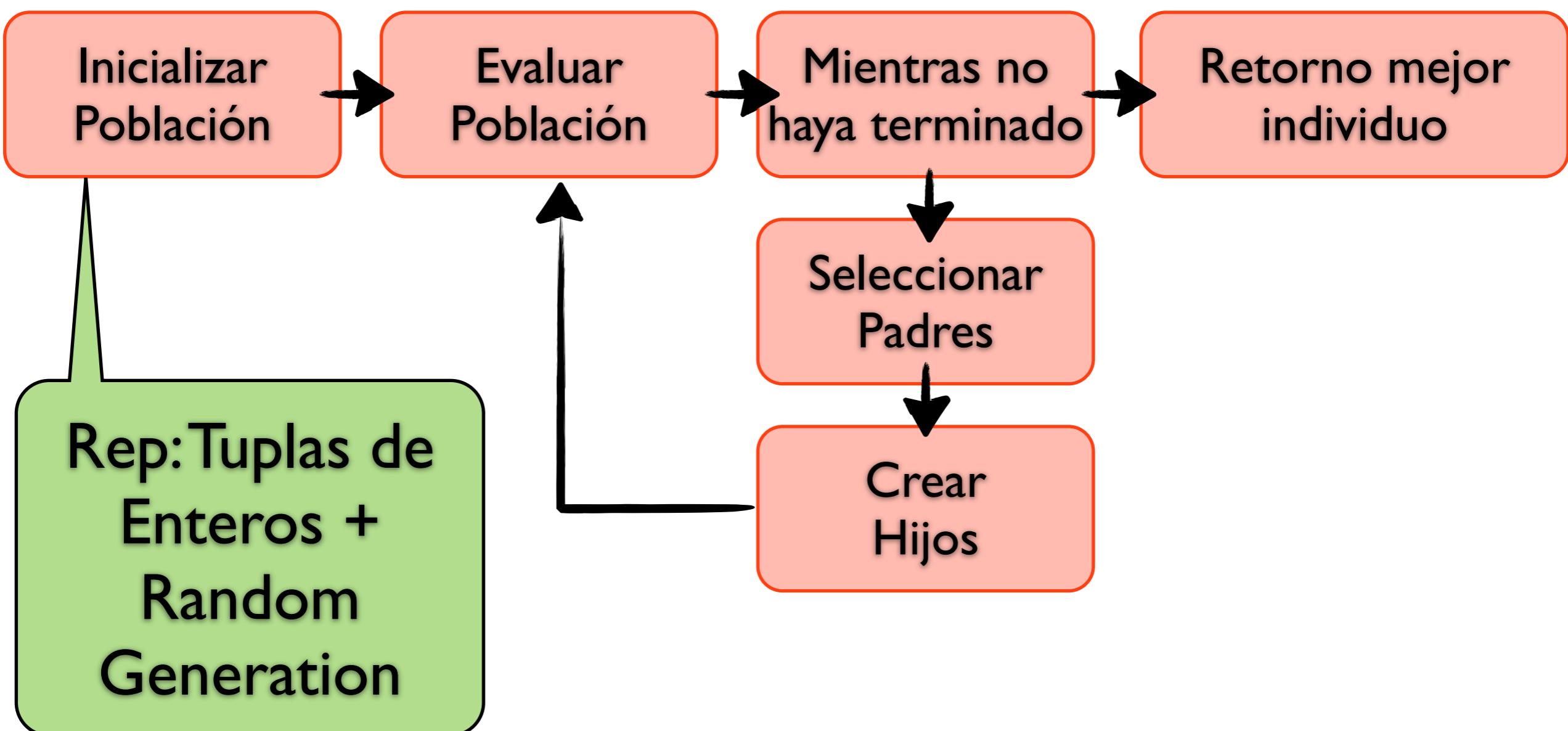
- Cambio en los genes
- La Mutación depende de los operadores genéticos
- Si la representación es binaria - bit flip

# Mutación

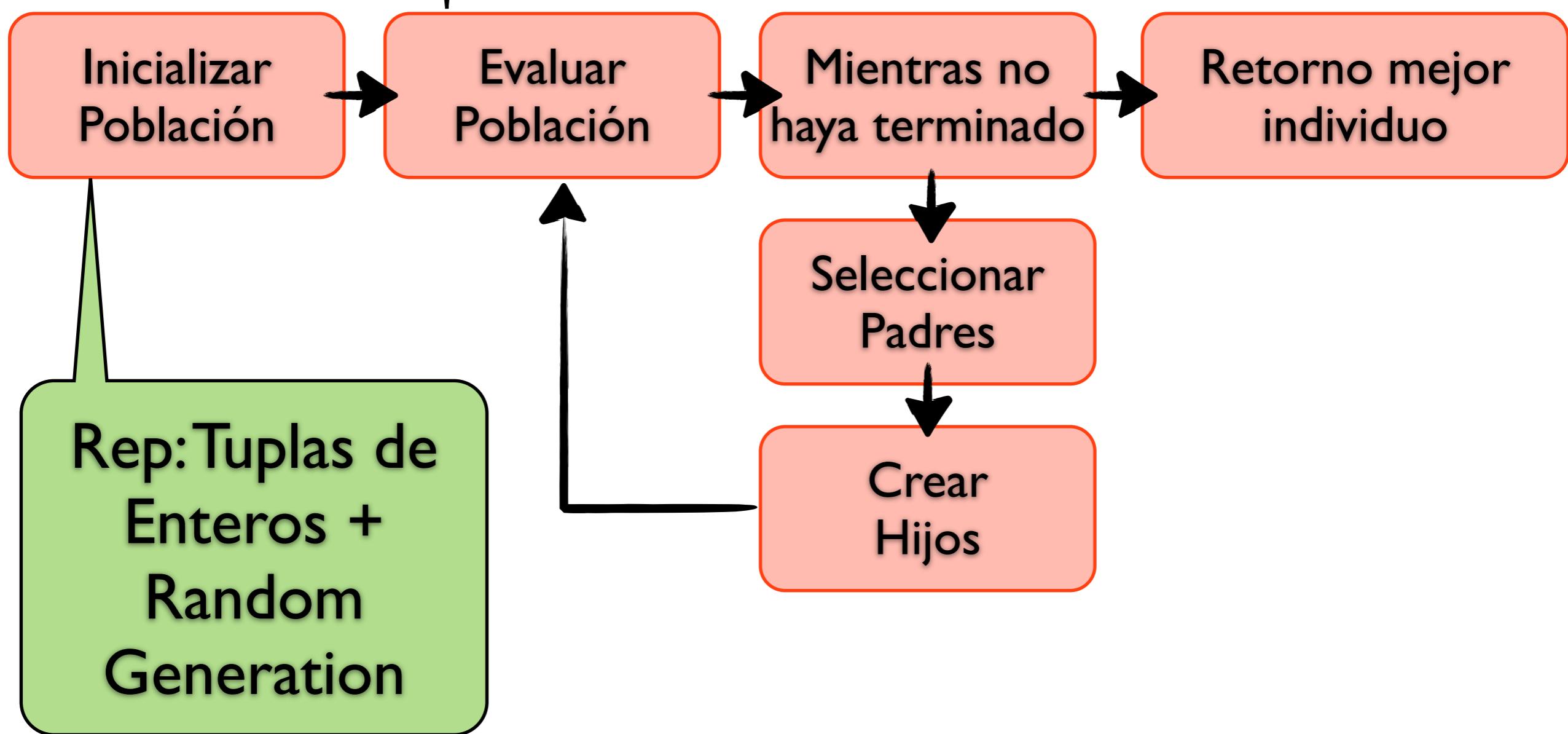


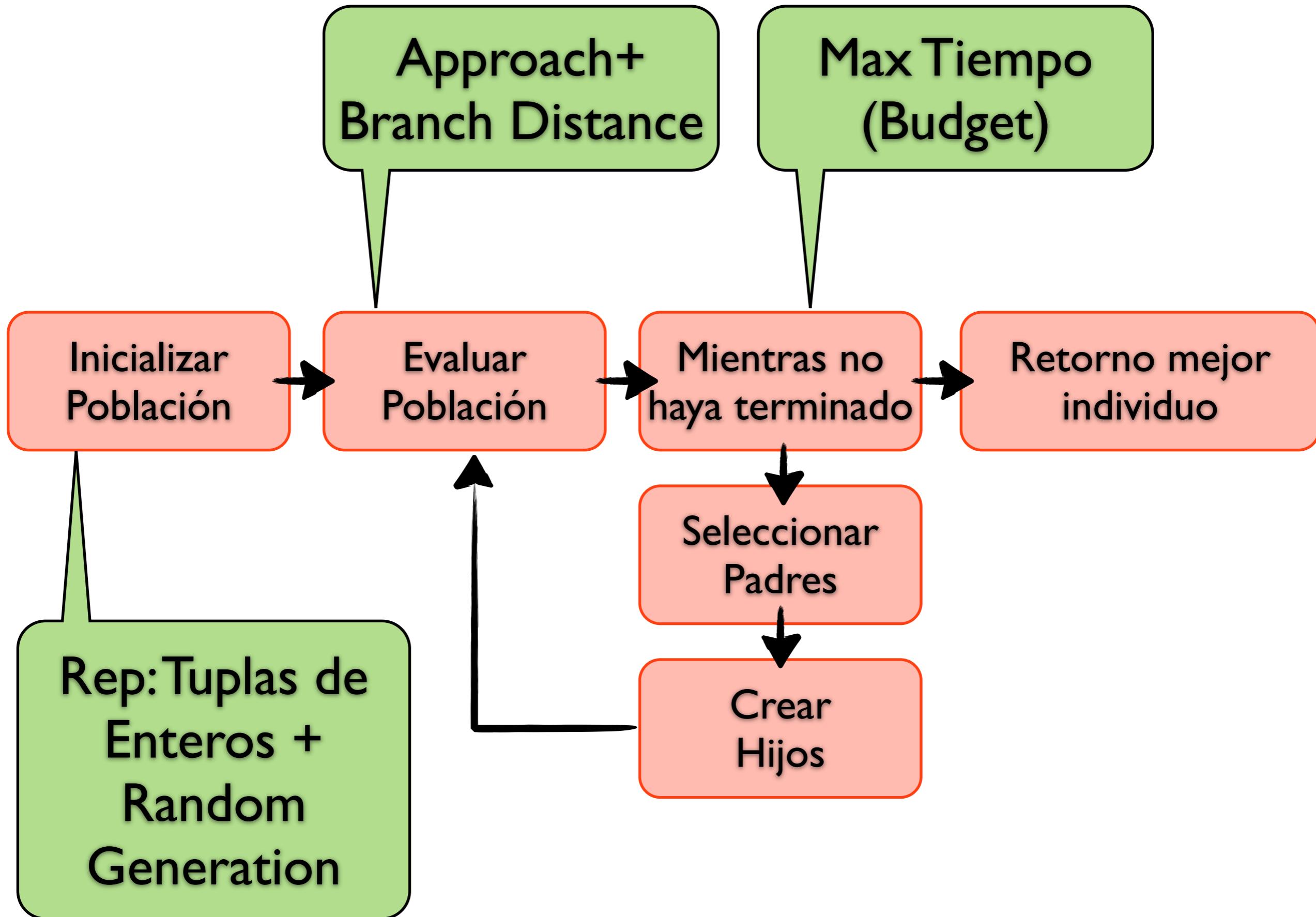
- Cambio en los genes
- La Mutación depende de los operadores genéticos
- Si la representación es binaria - bit flip

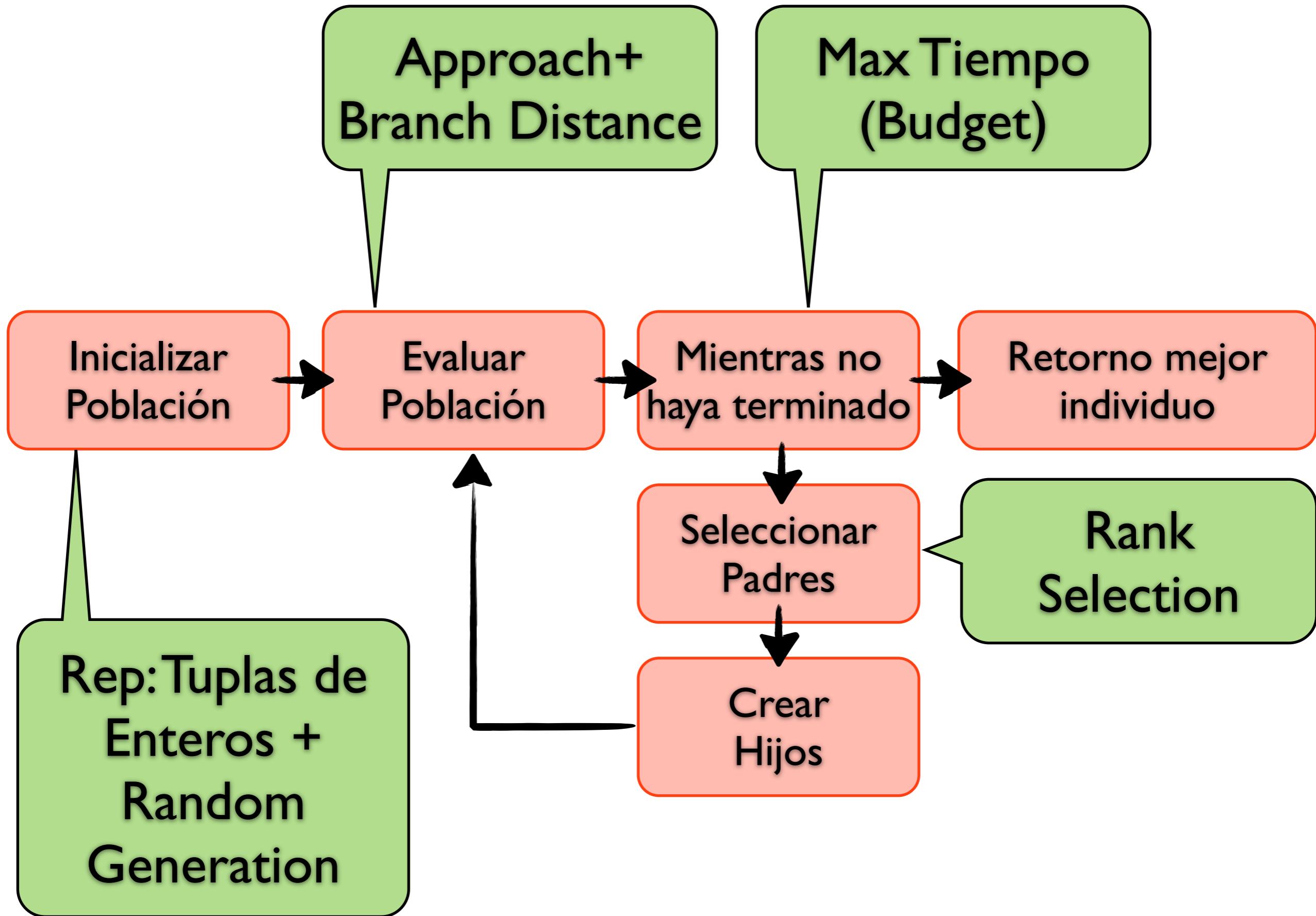




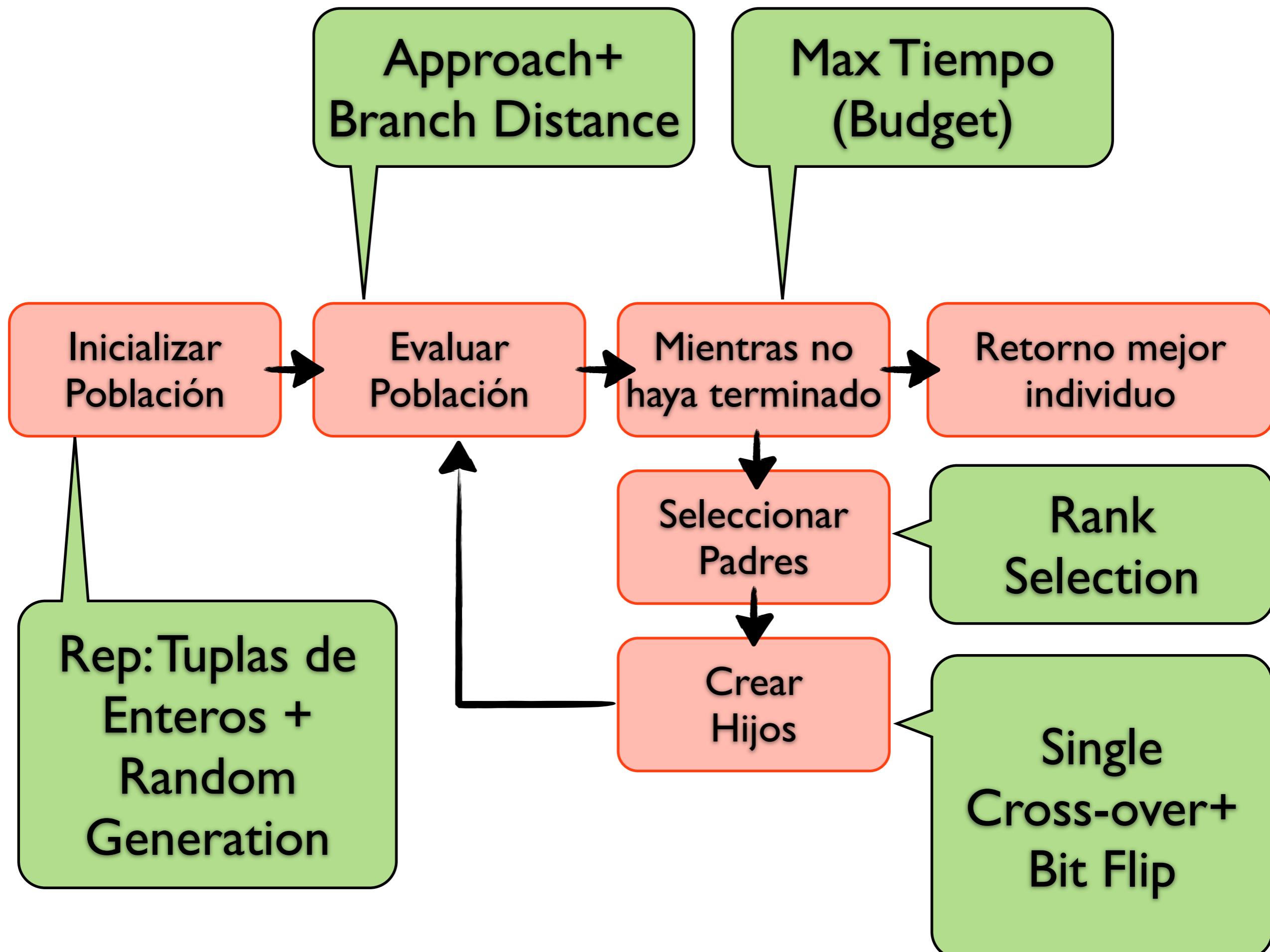
## Approach+ Branch Distance

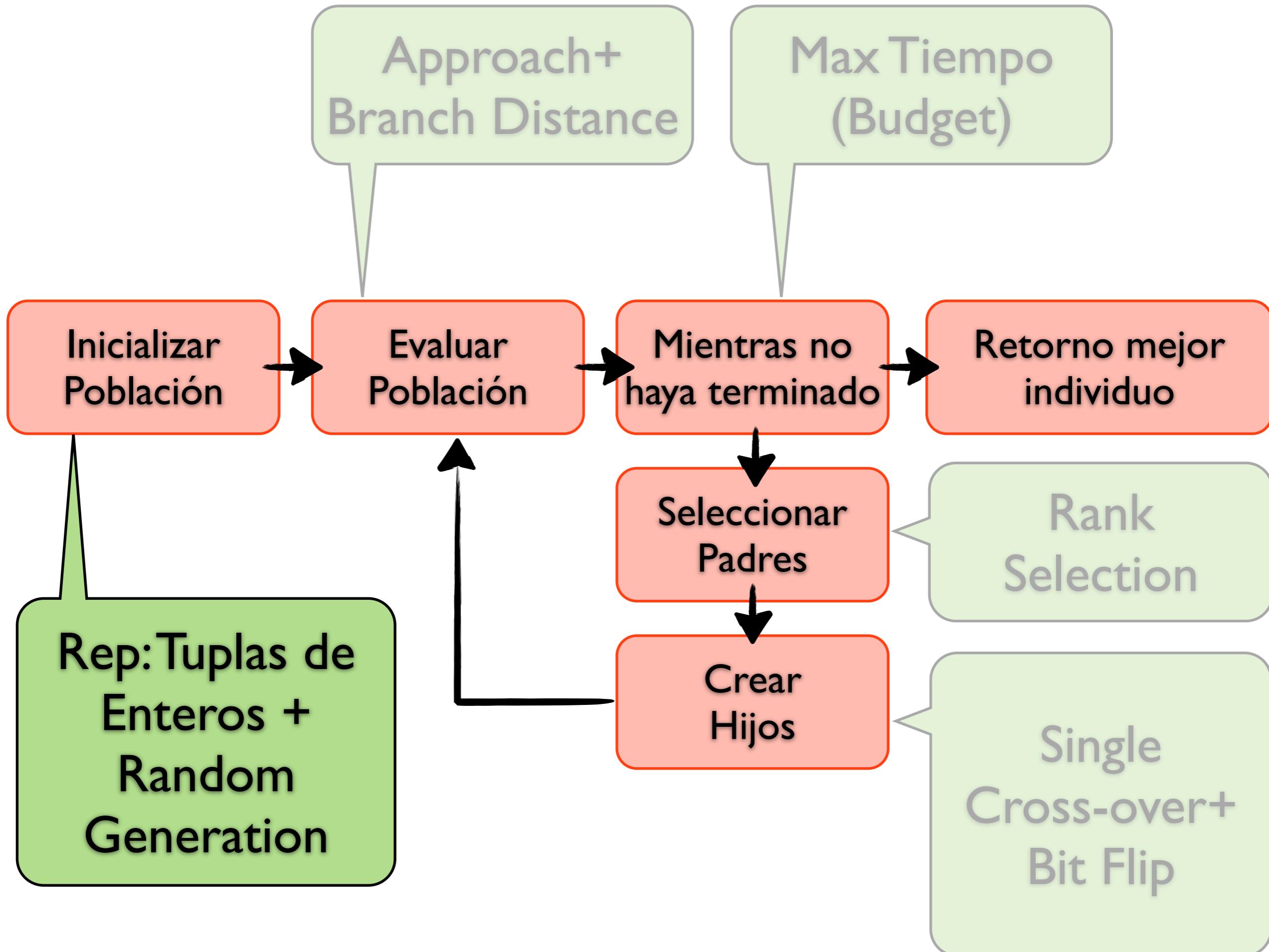






qui uebodos usi plc fose  
U ejemplos





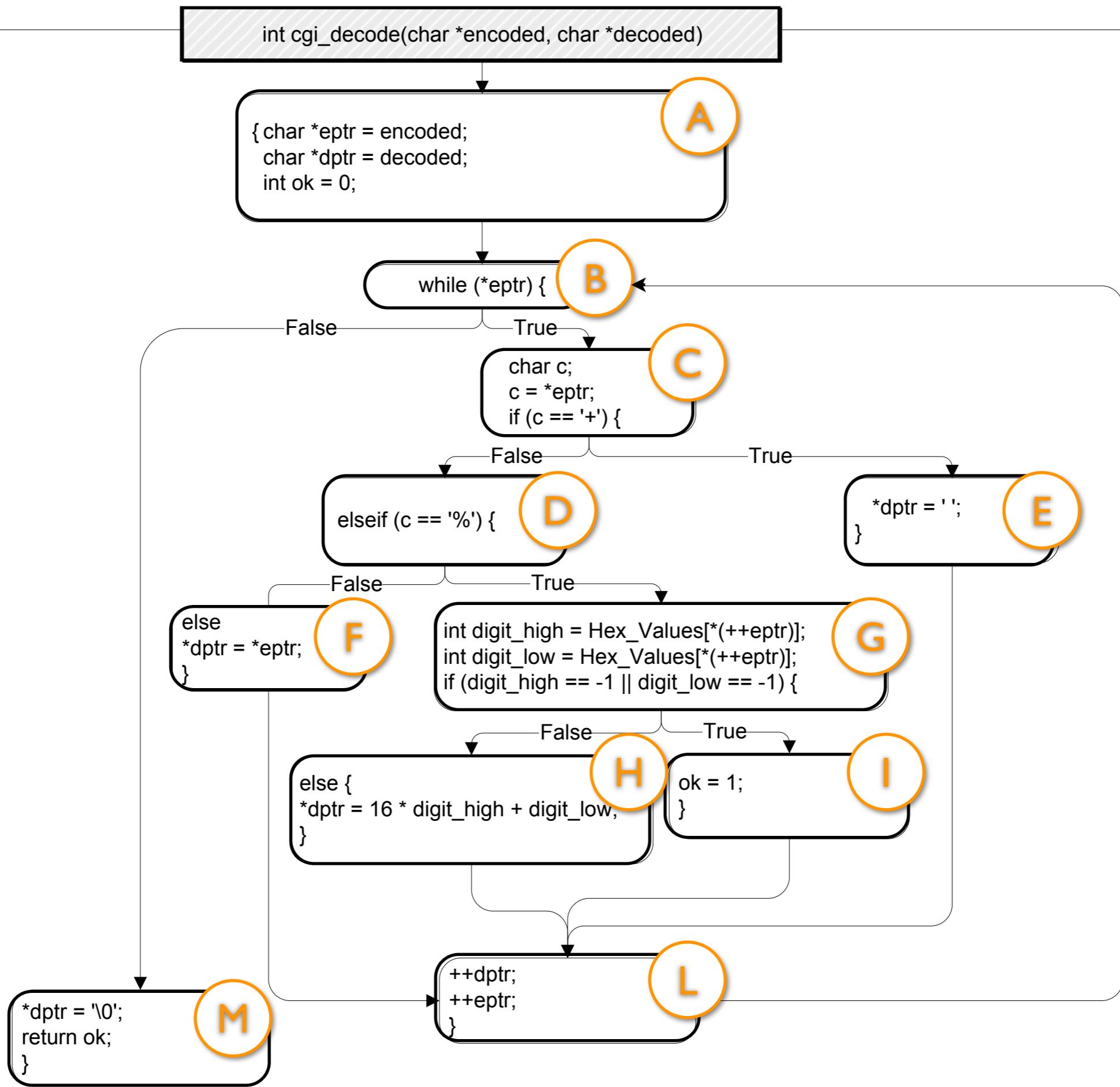
# Limitaciones de Branch Distance

```
def testMe(x, y, z):  
    if x==10:  
        if y==20:  
            if x * z == 2 * (y + 1):  
                return True //branch a cubrir  
    return False
```

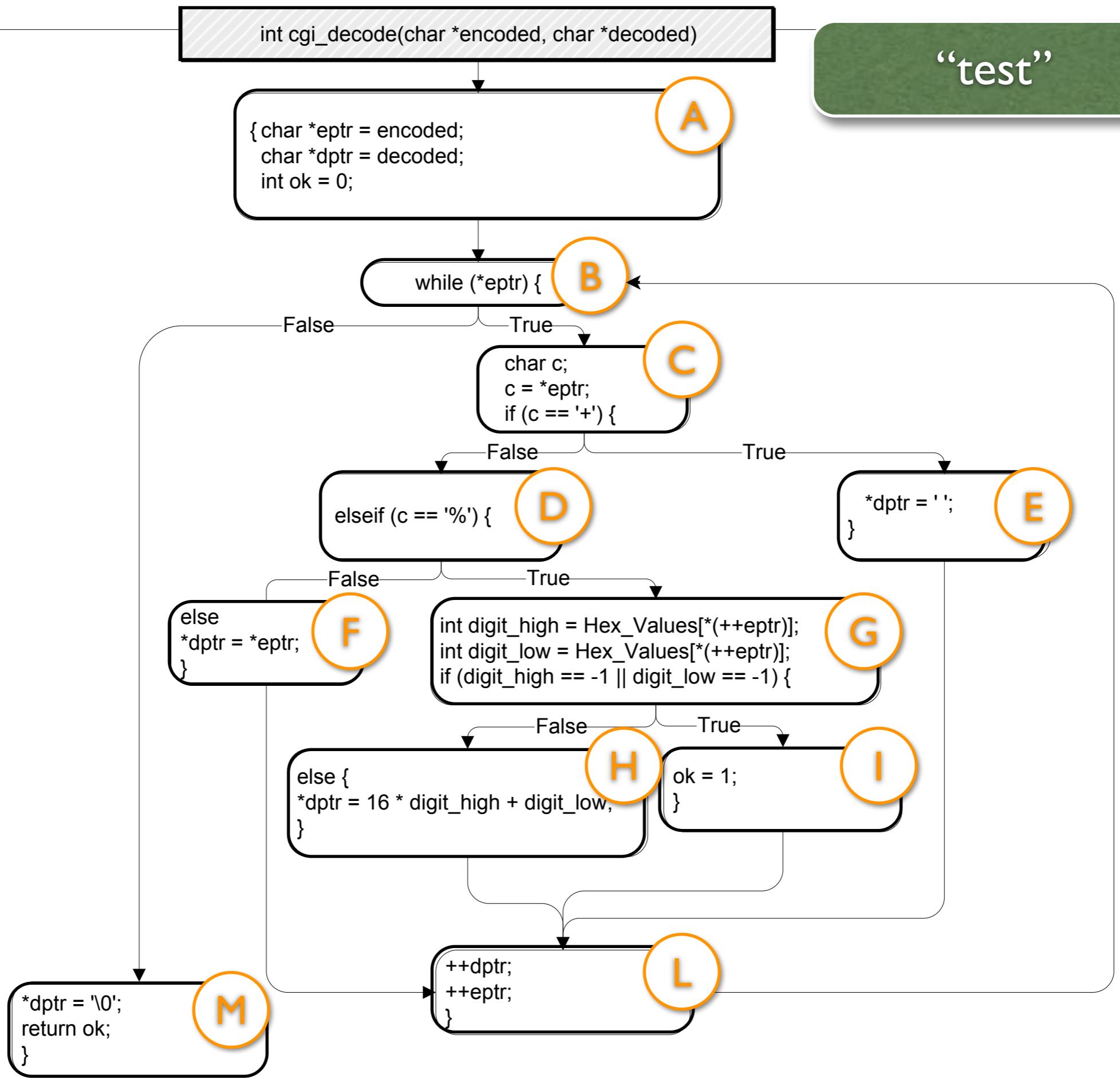
→ Enhance no refleja  
el resto del path p/  
después del branch

- ¿Qué pasa si el predicado no fue alcanzado por el individuo?

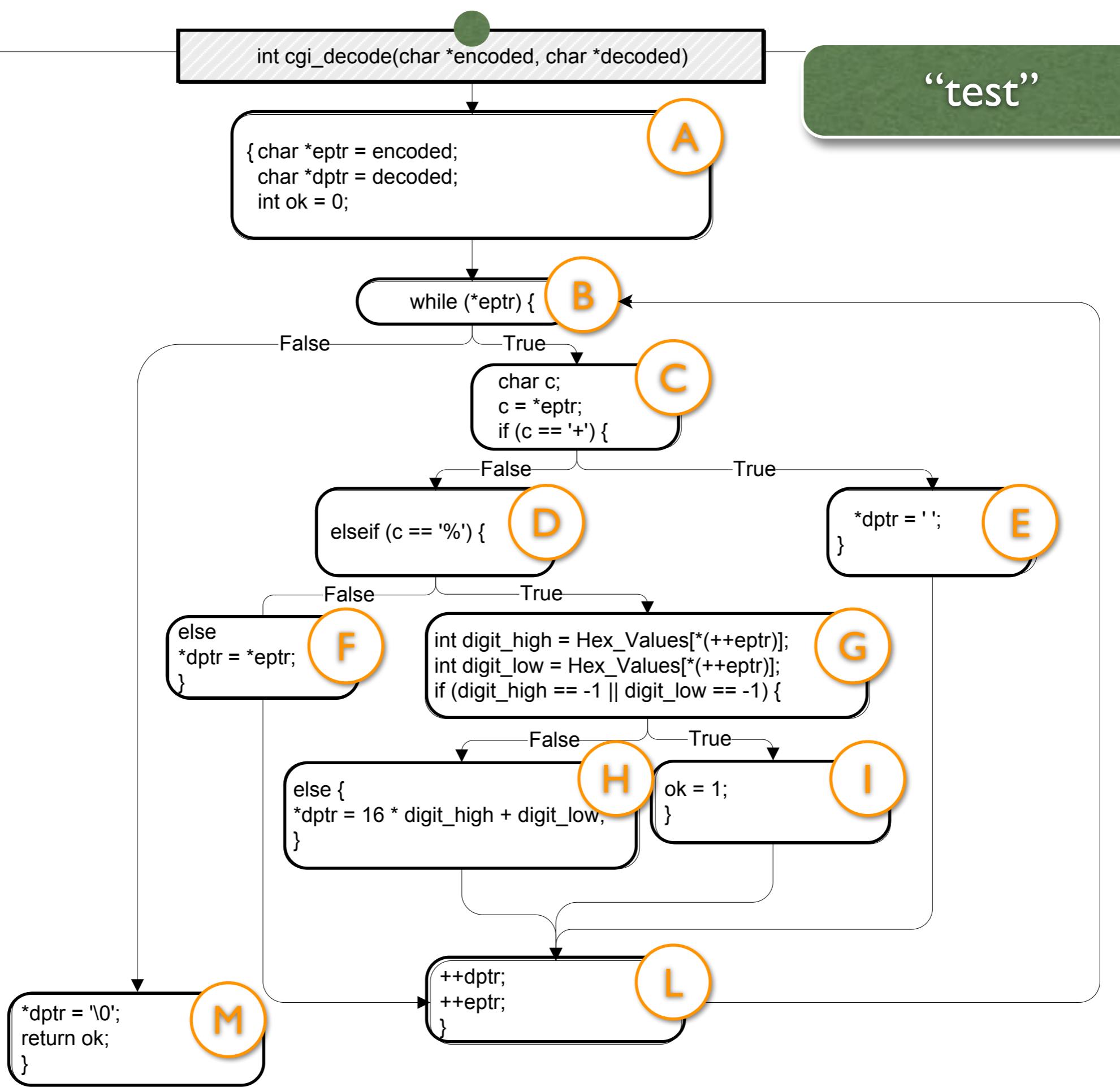
# CFG



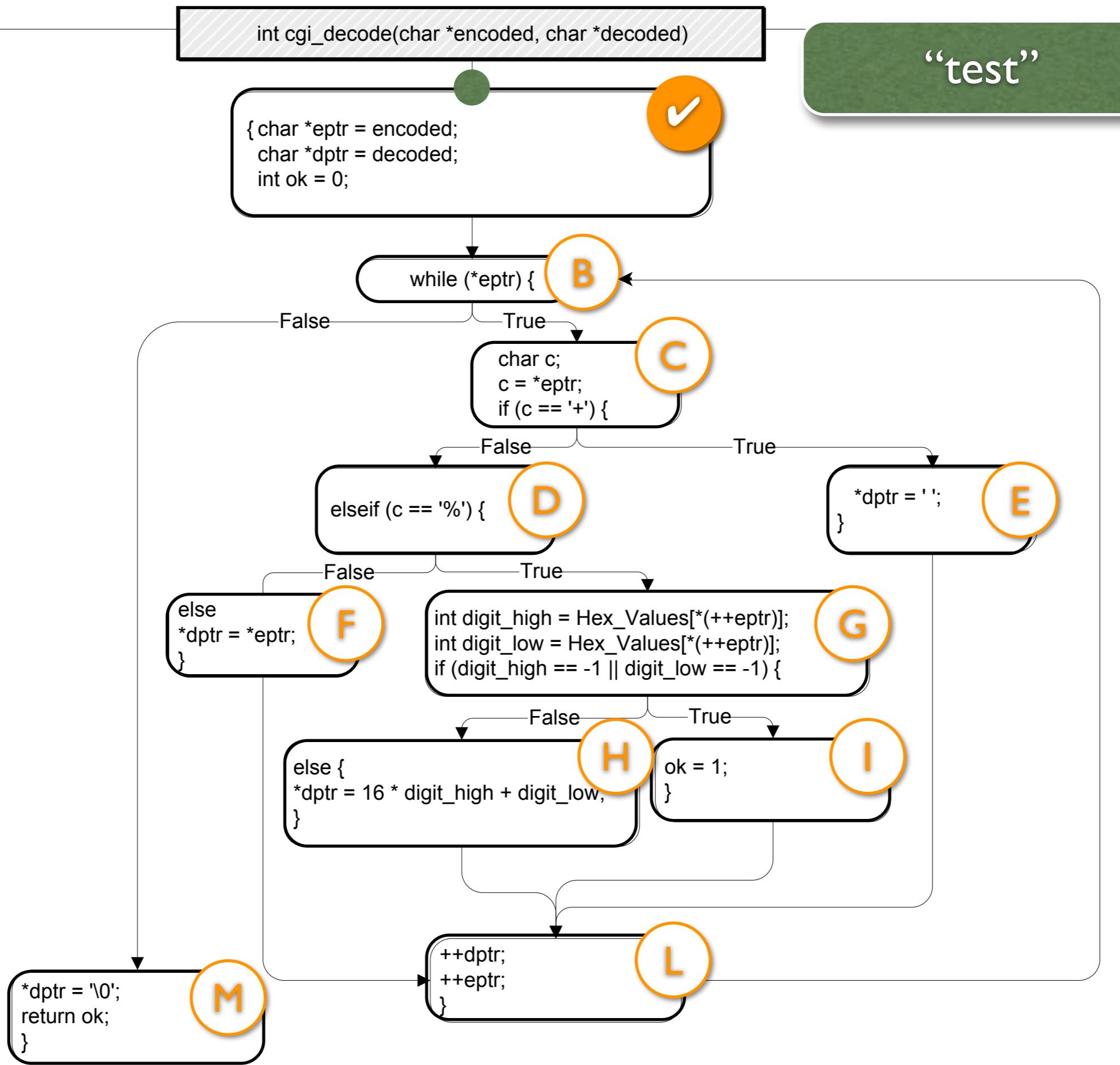
# CFG



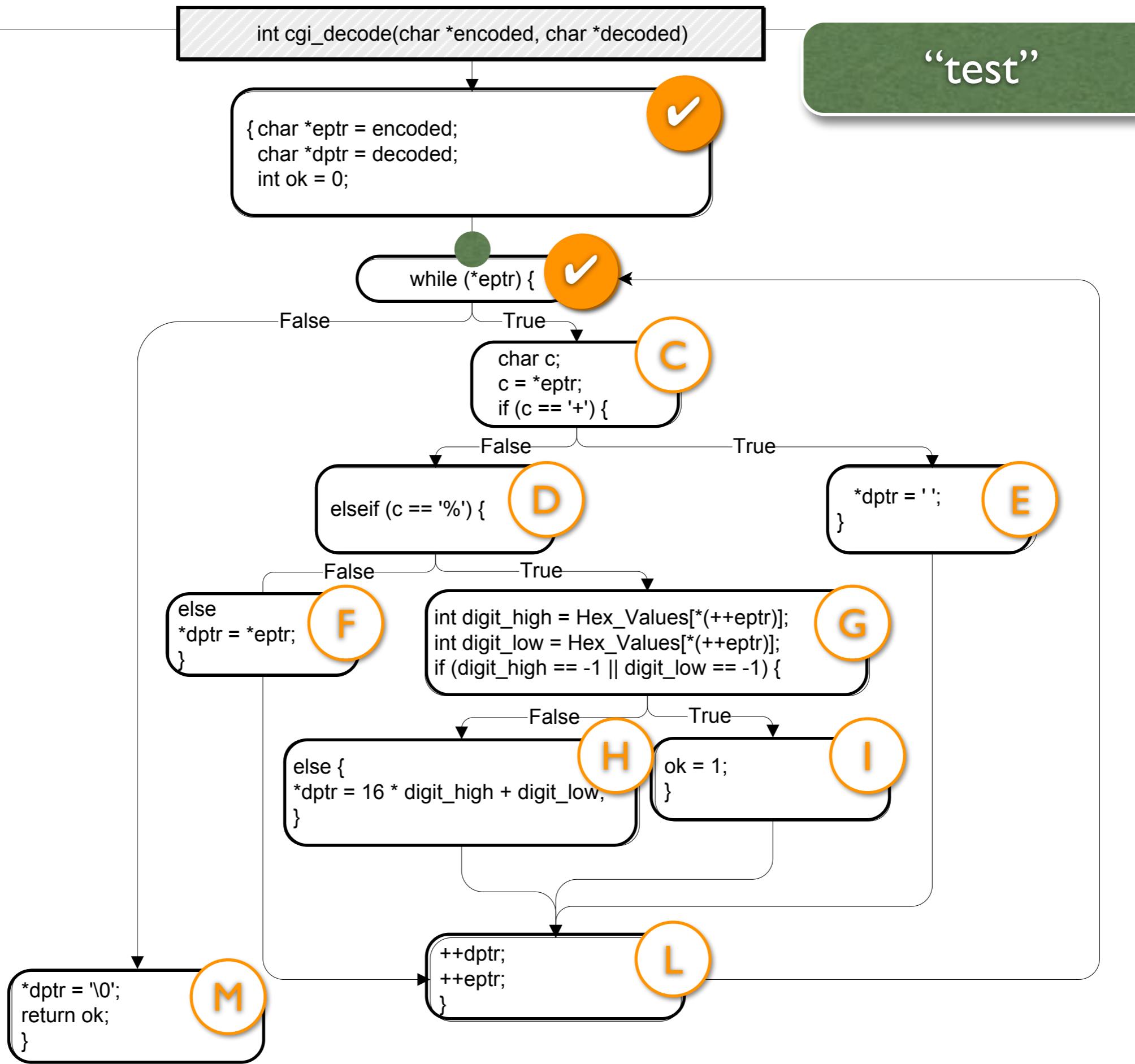
# CFG



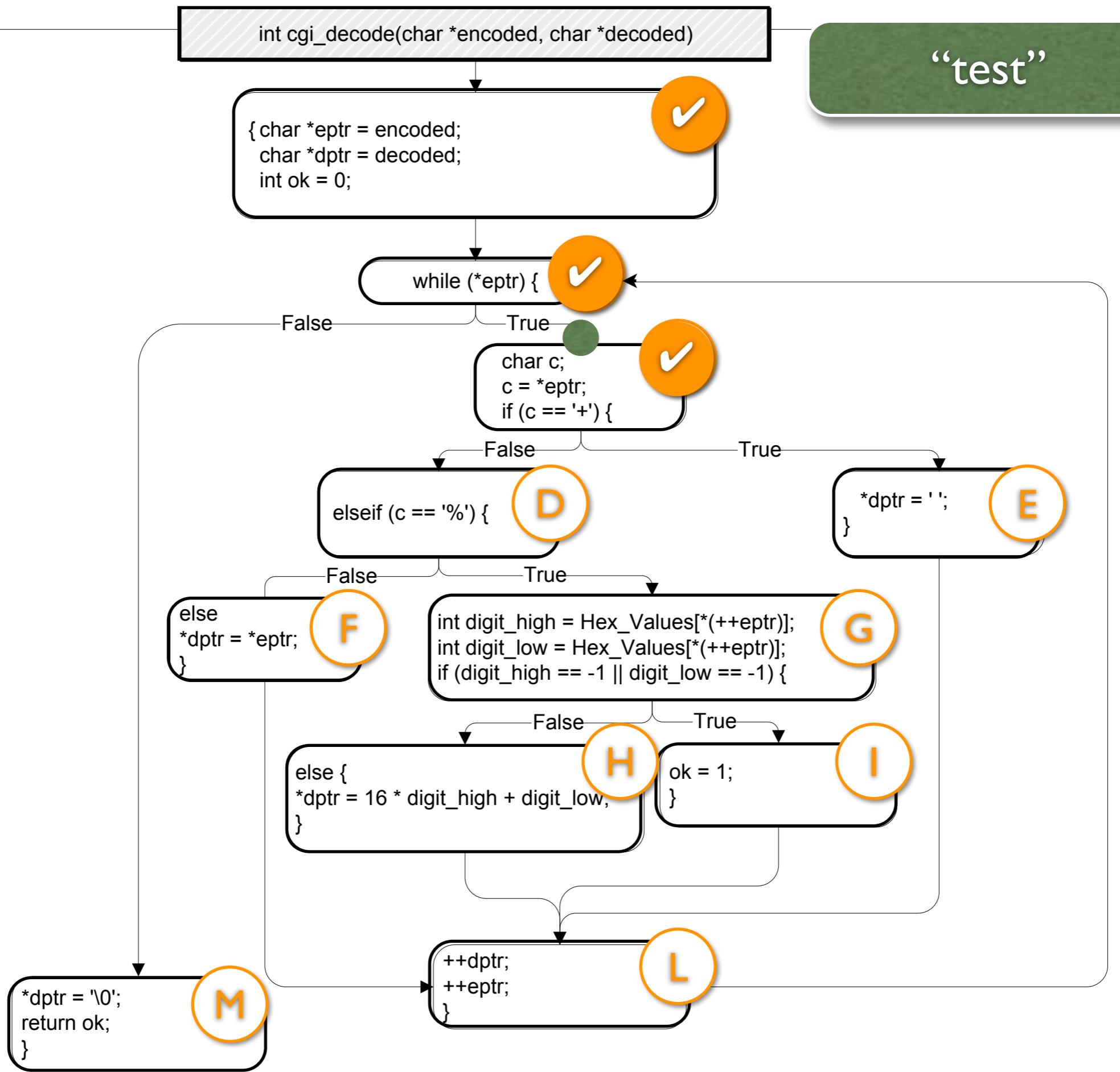
# CFG



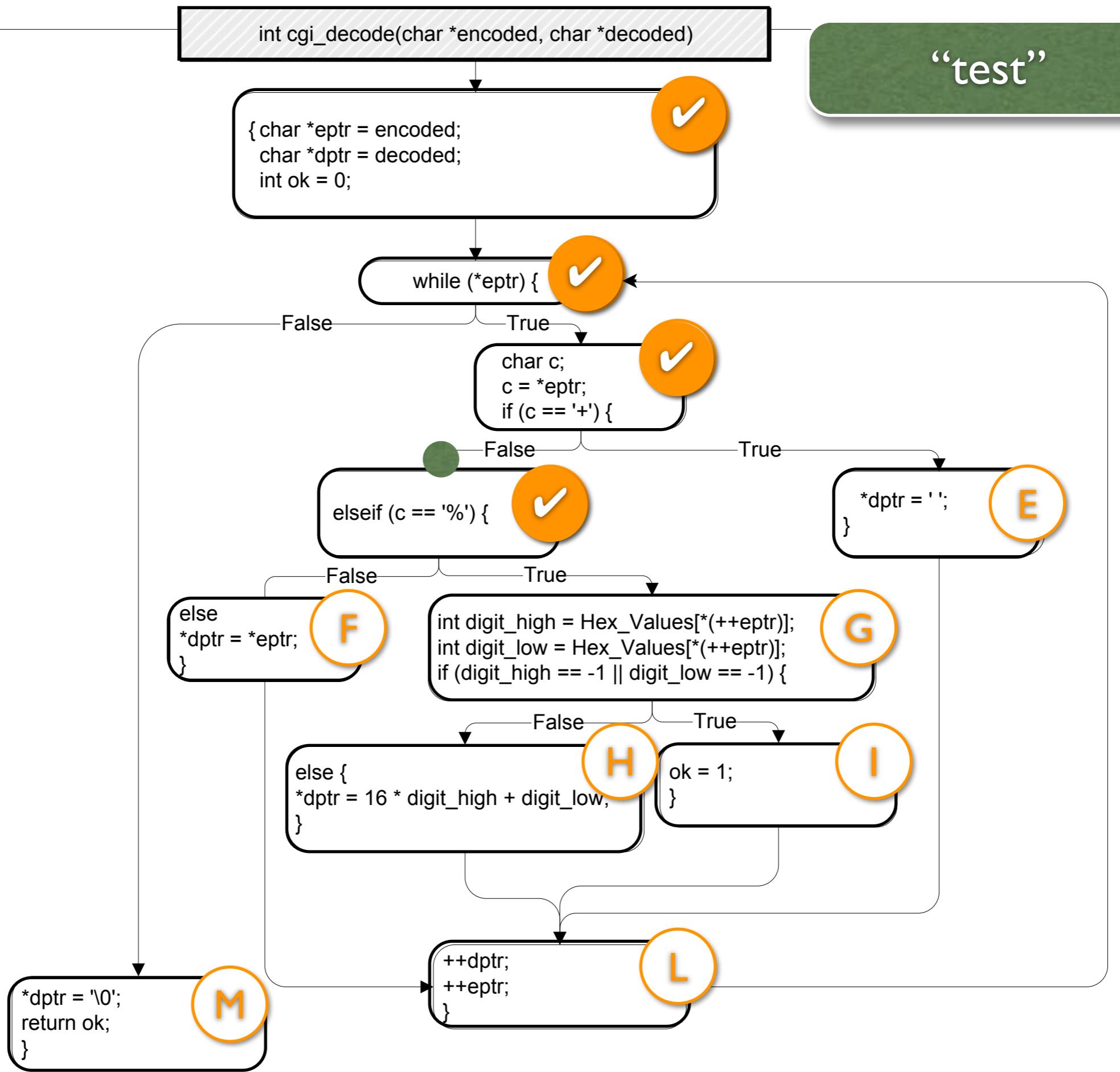
# CFG



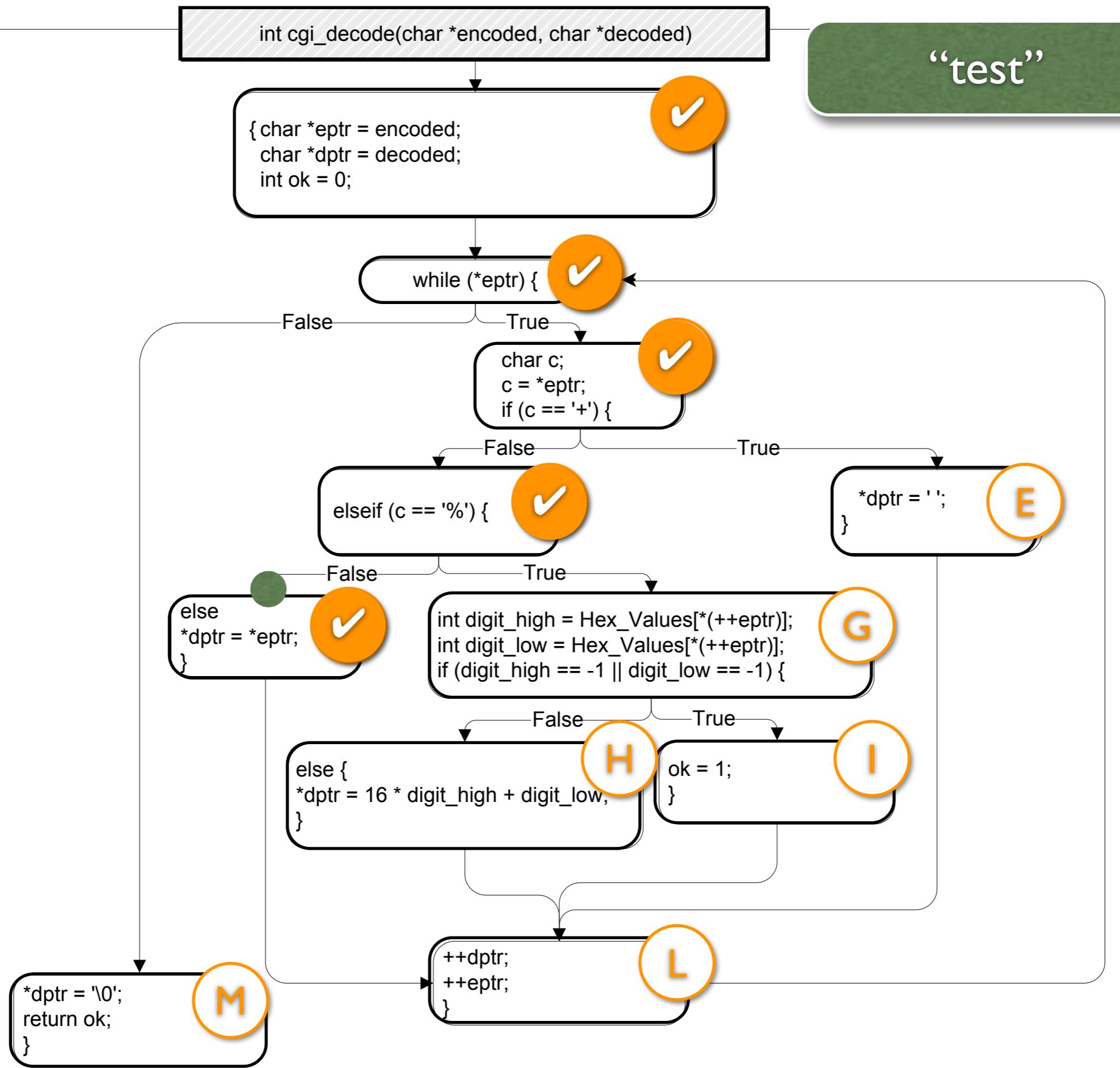
# CFG



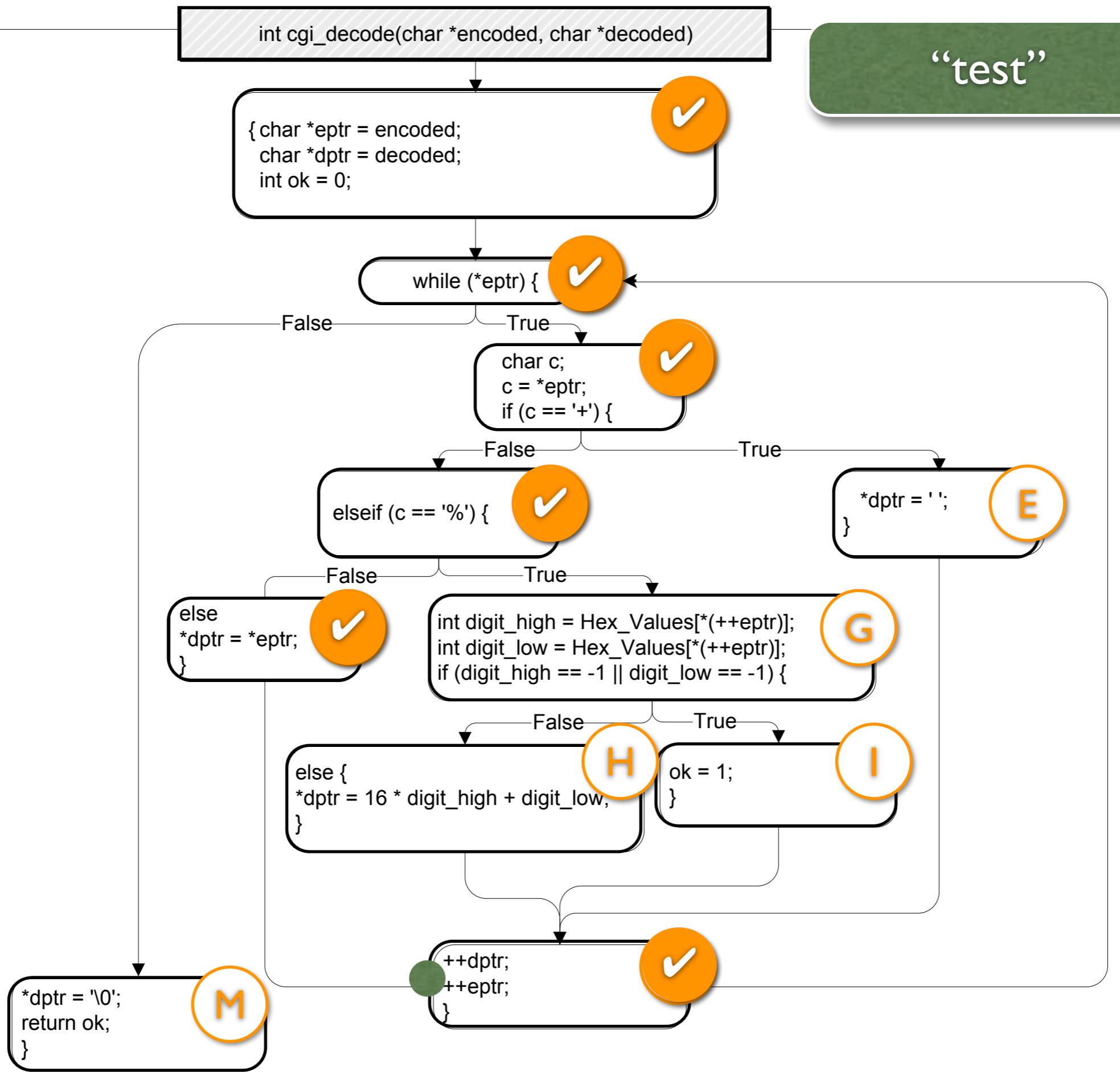
# CFG



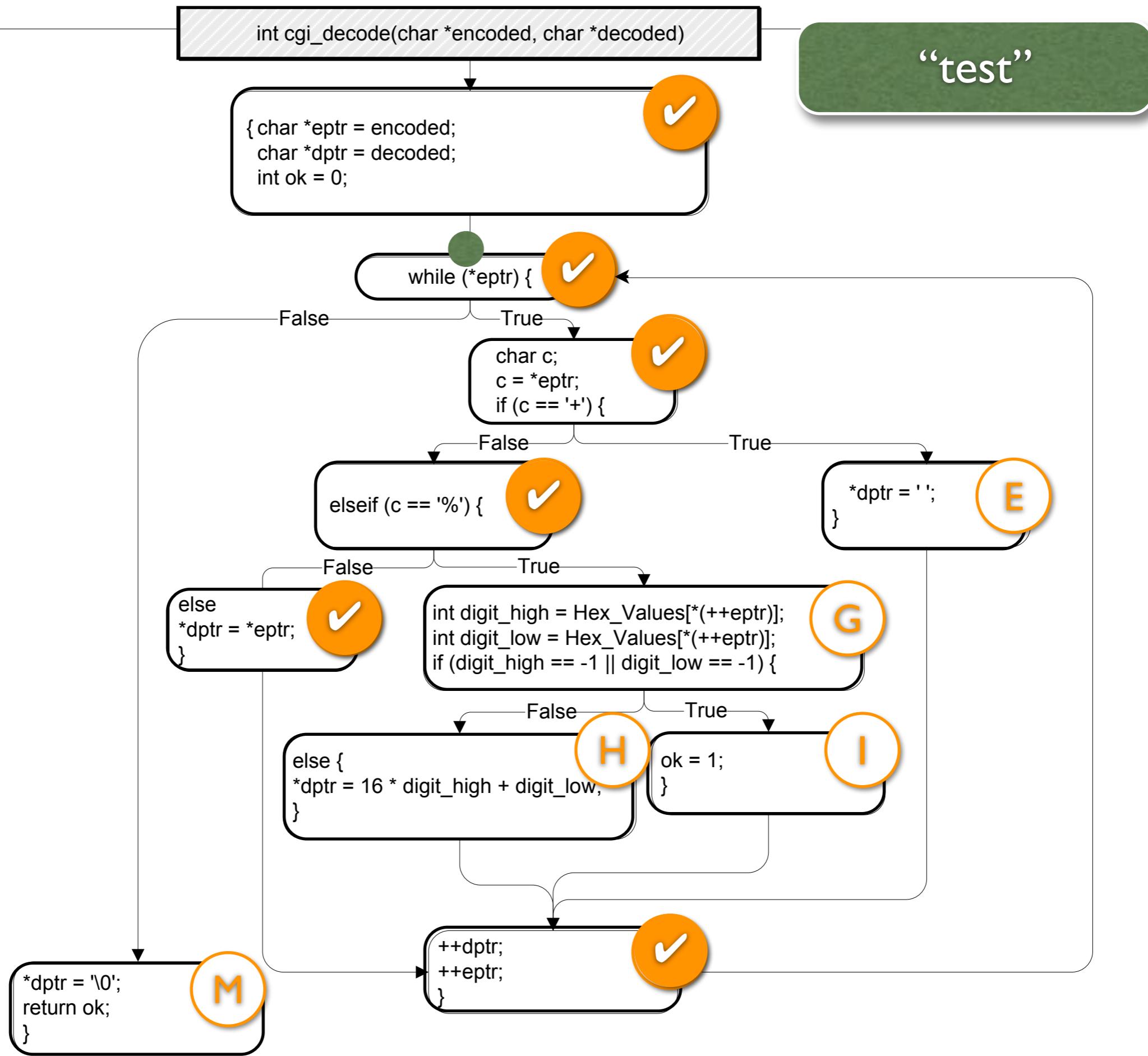
# CFG



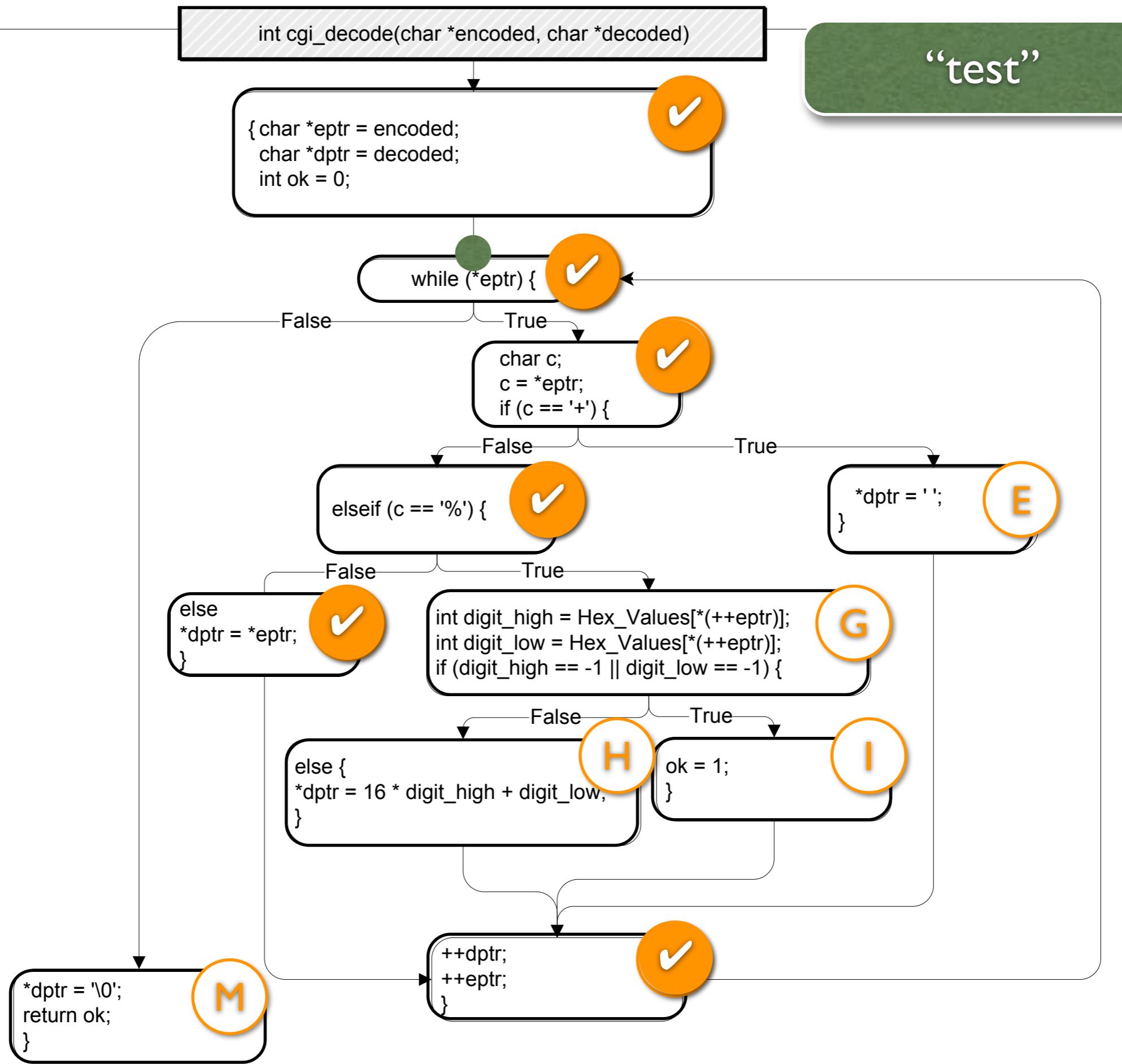
# CFG



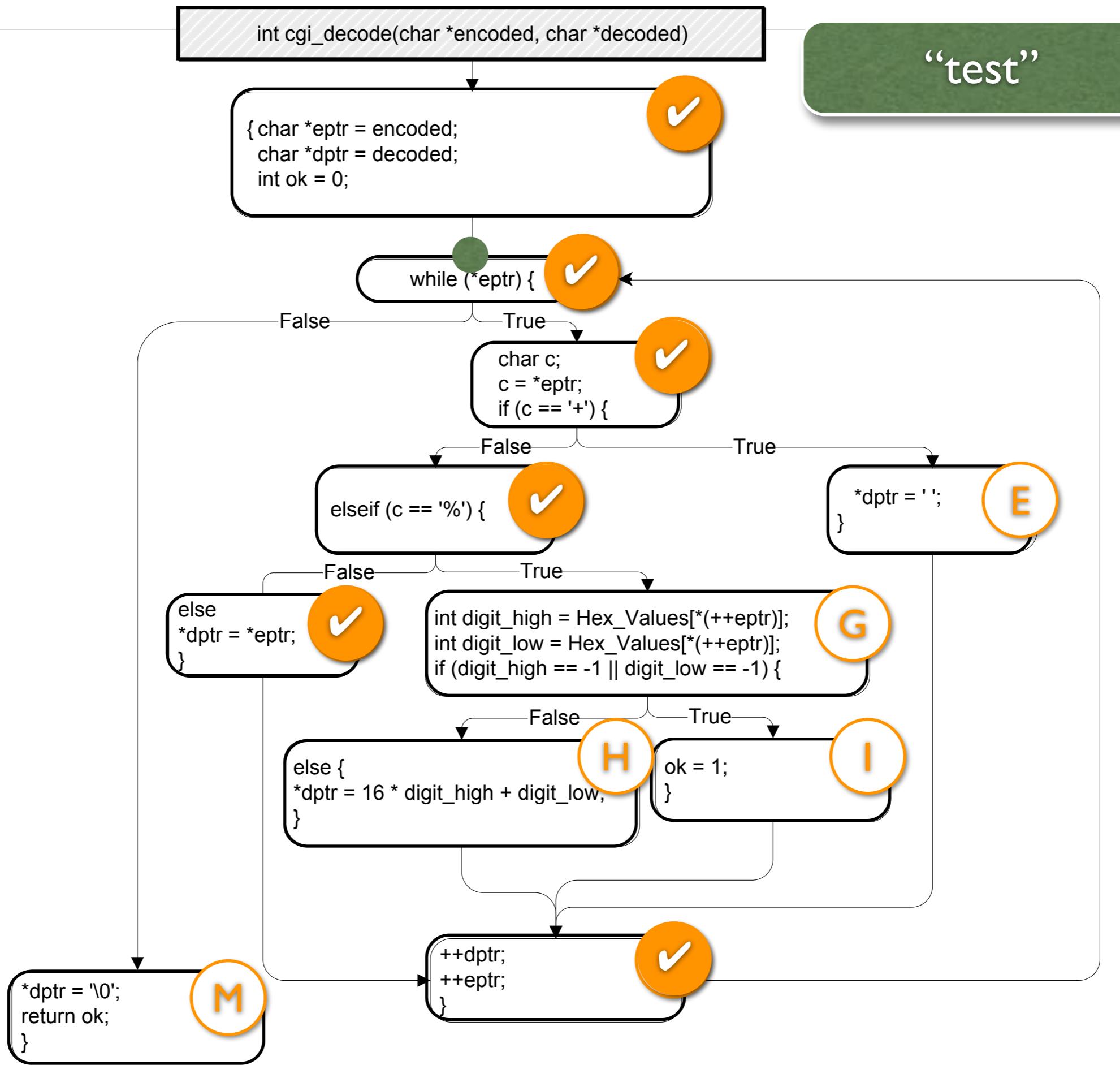
# CFG



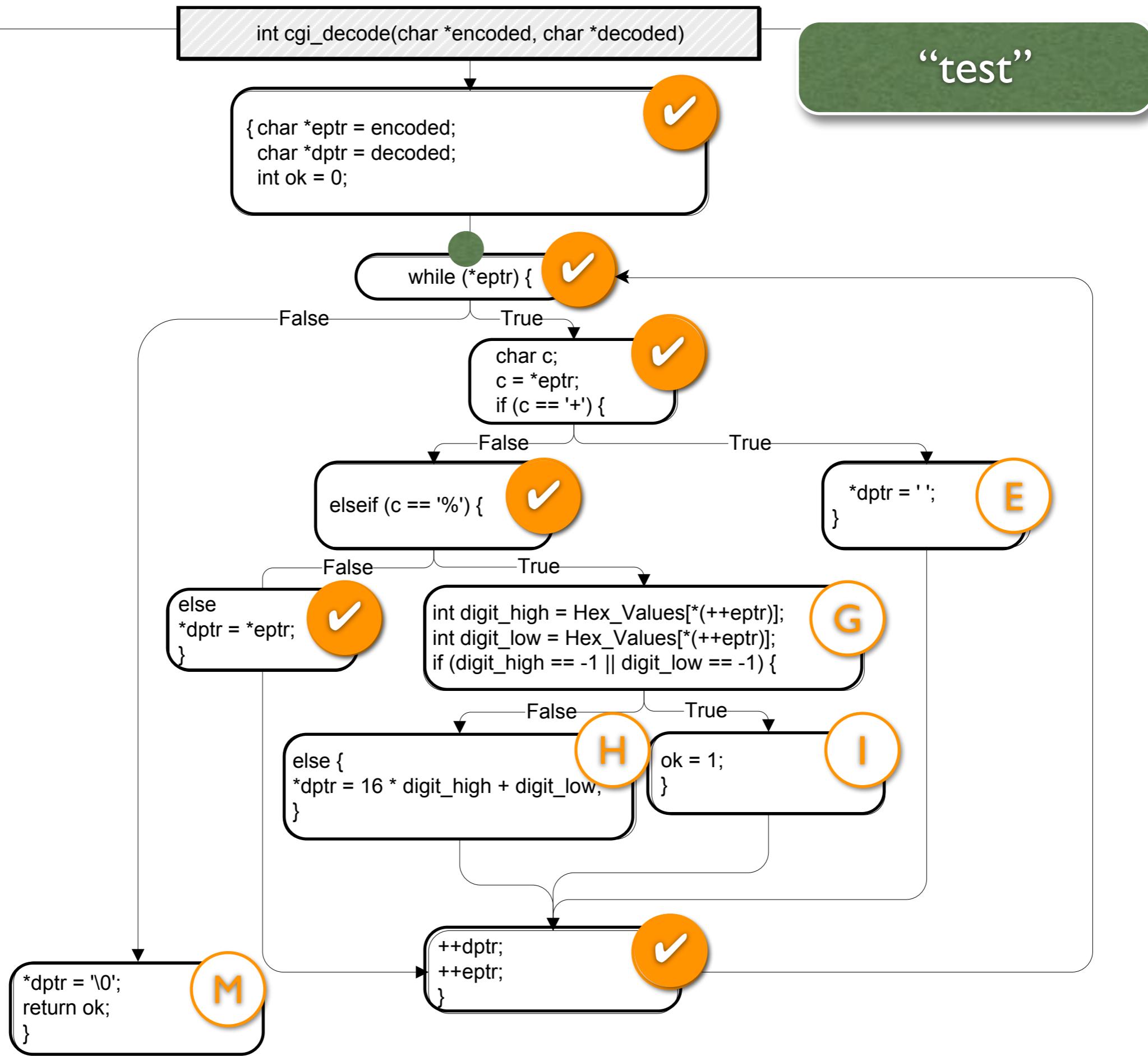
# CFG



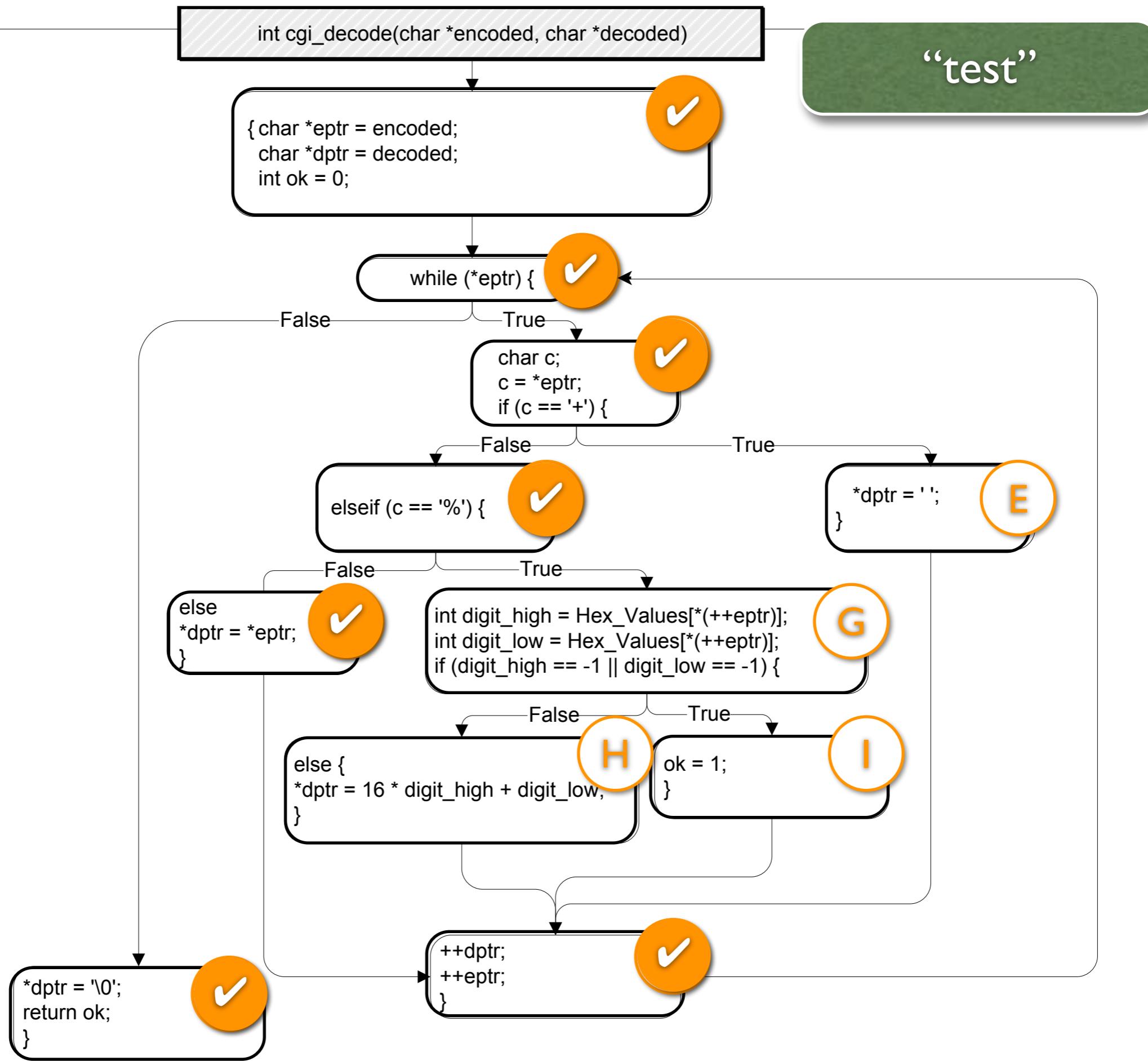
# CFG



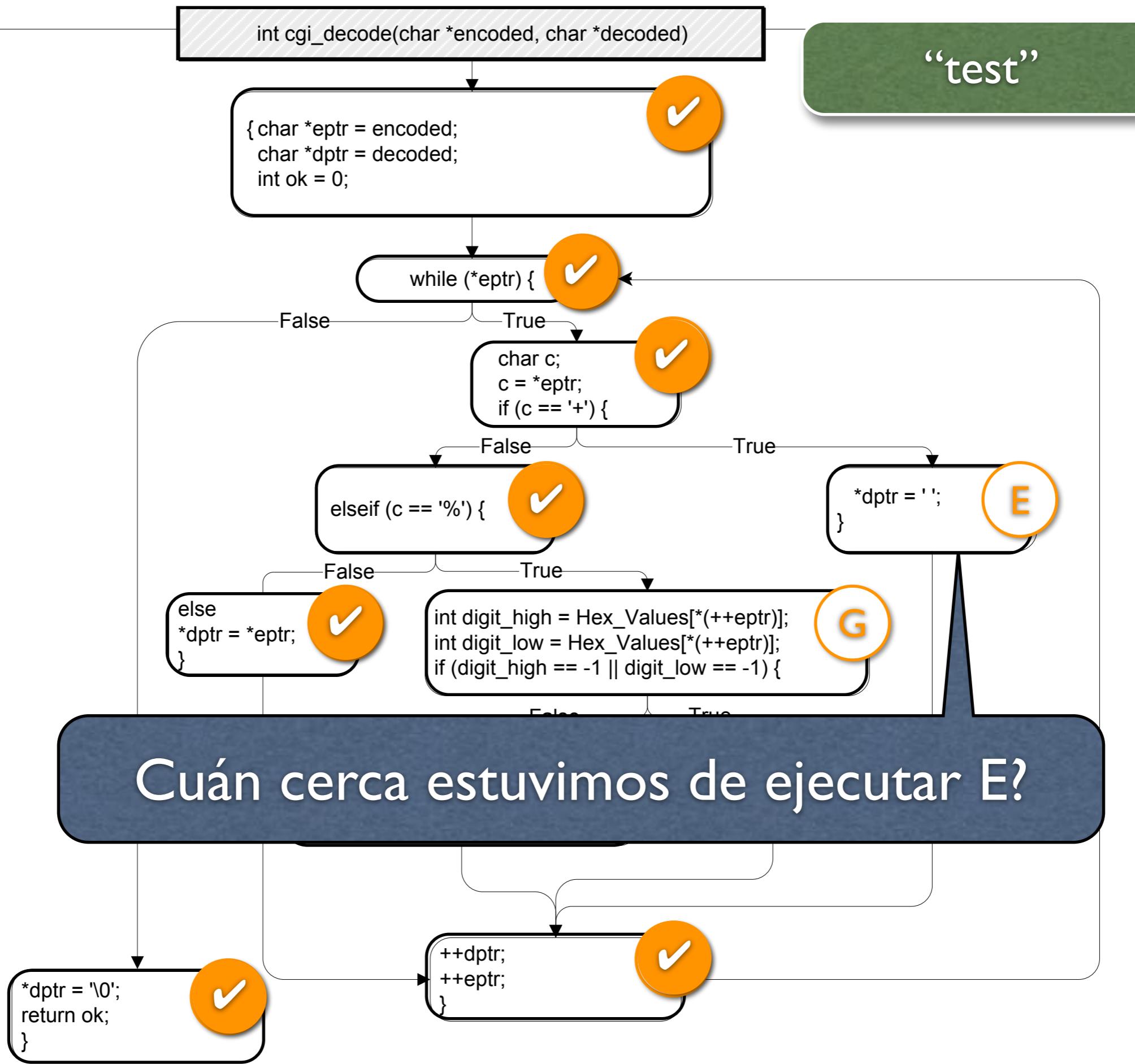
# CFG



# CFG

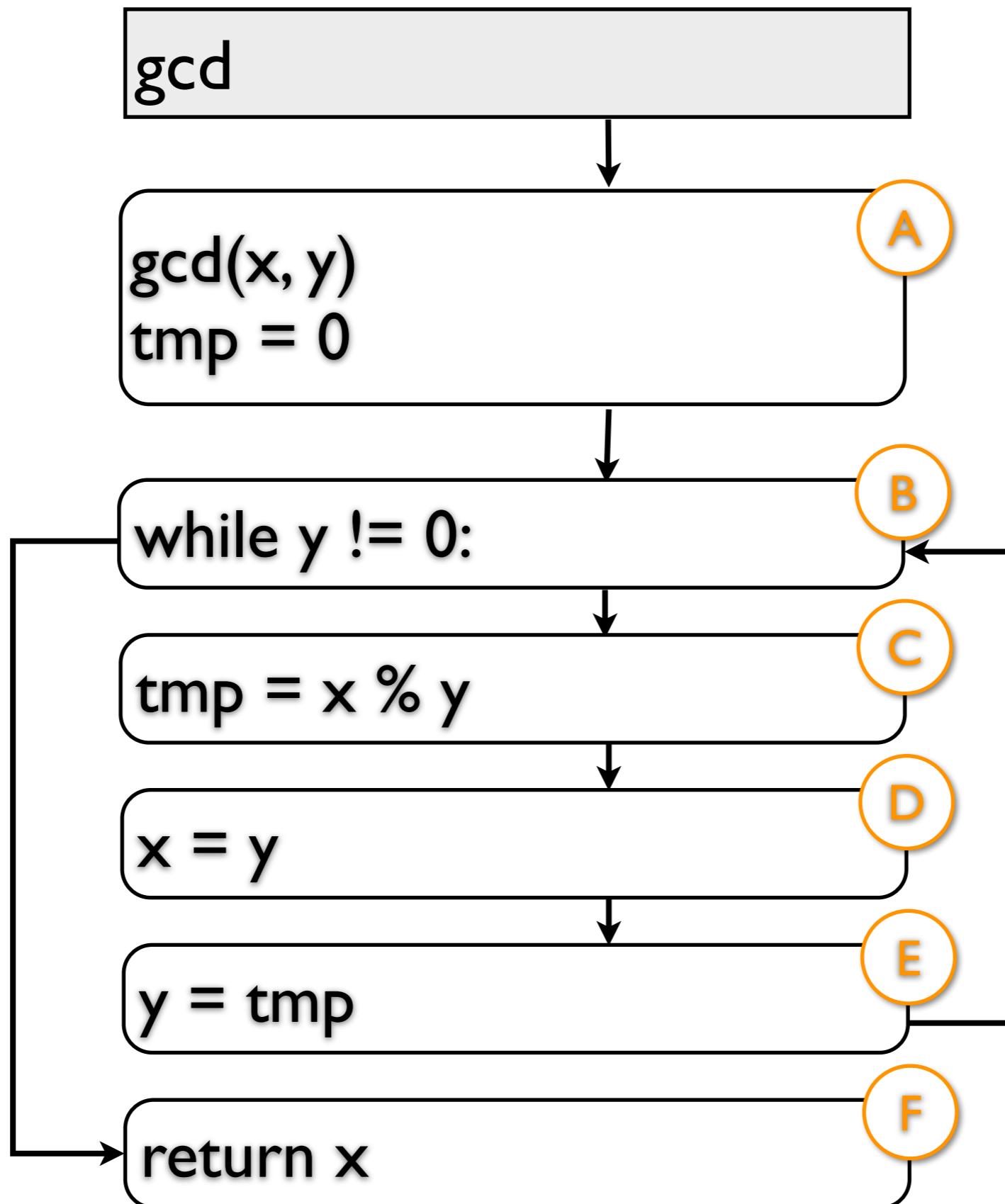


CFG



```
def gcd(x, y):  
    tmp = 0  
    while y != 0:  
        tmp = x % y  
        x = y  
        y = tmp  
  
    return x
```

# CFG

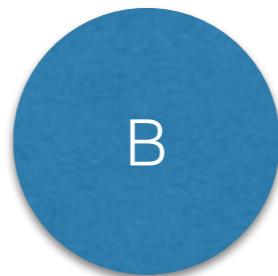


Control dependency graph  
Los nodos dependencias en  
el código

# “Dominator”

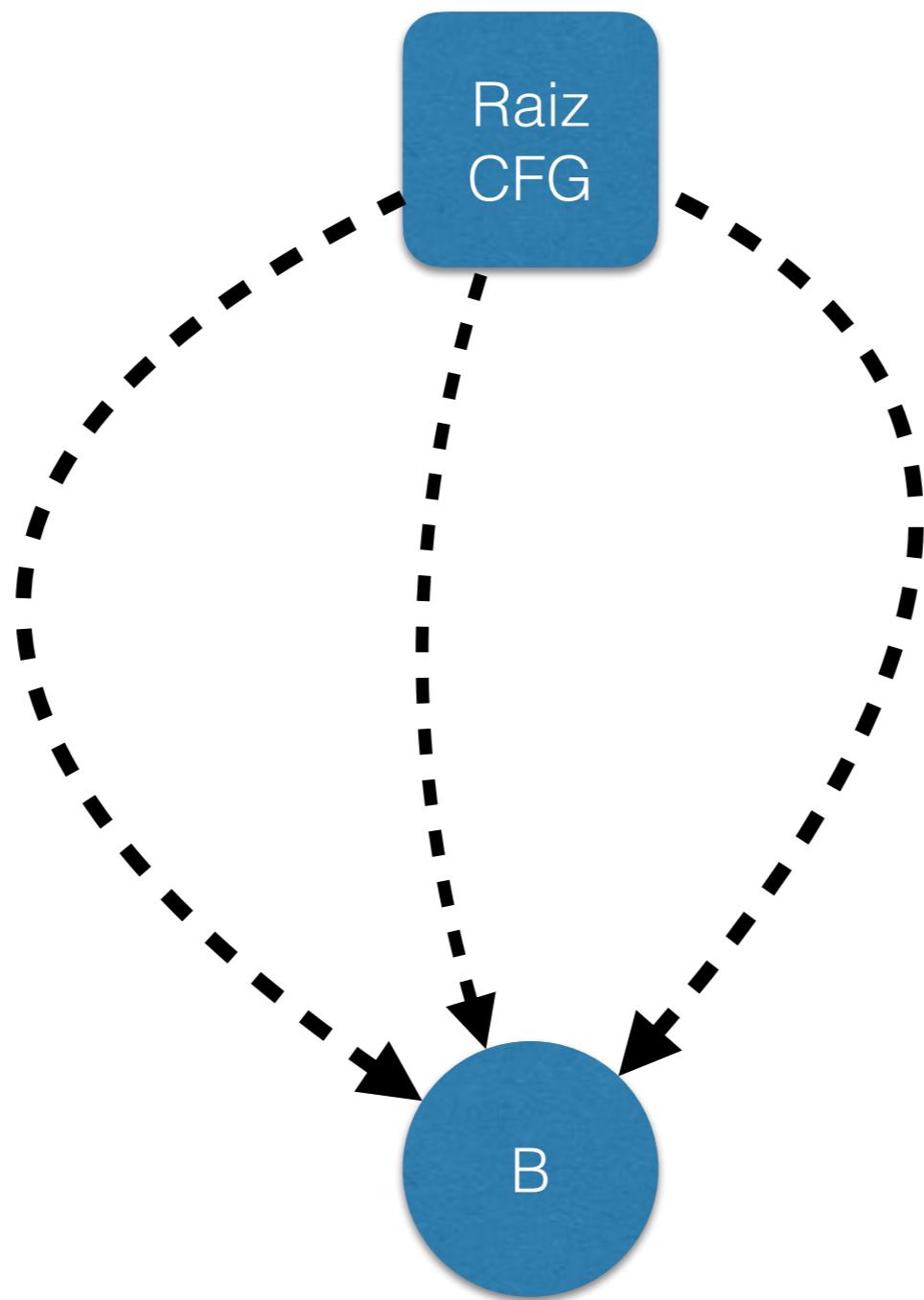
- 
- El Nodo A **domina** al Nodo B si todo camino hacia B debe pasar por A
  - Dominador Inmediato: El dominador mas cercano en todo camino desde la raíz
  - La raíz no tiene dominador inmediato

# A domina a B si:



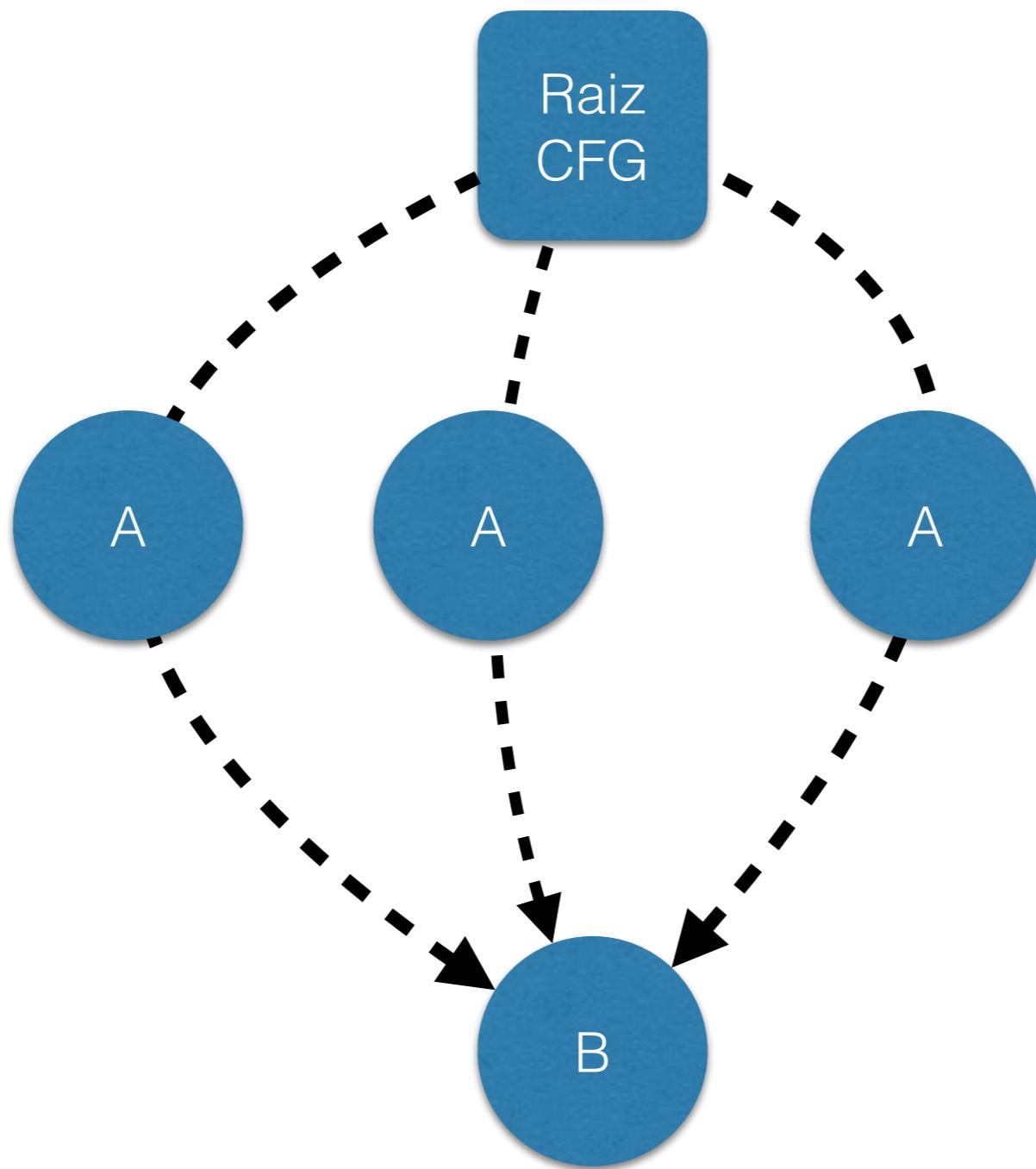
Caminos en el CFG

# A domina a B si:



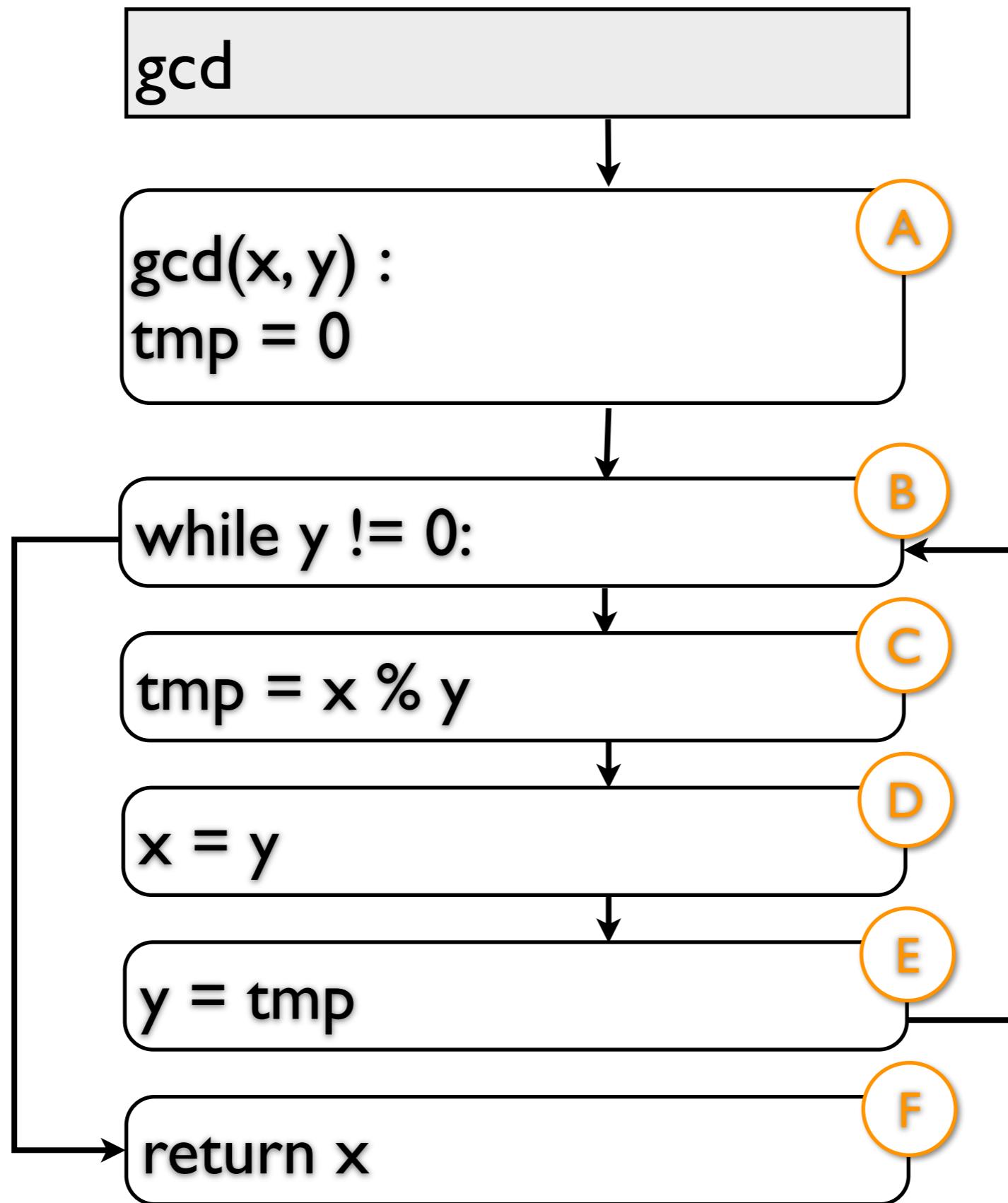
Caminos en el CFG

# A domina a B si:



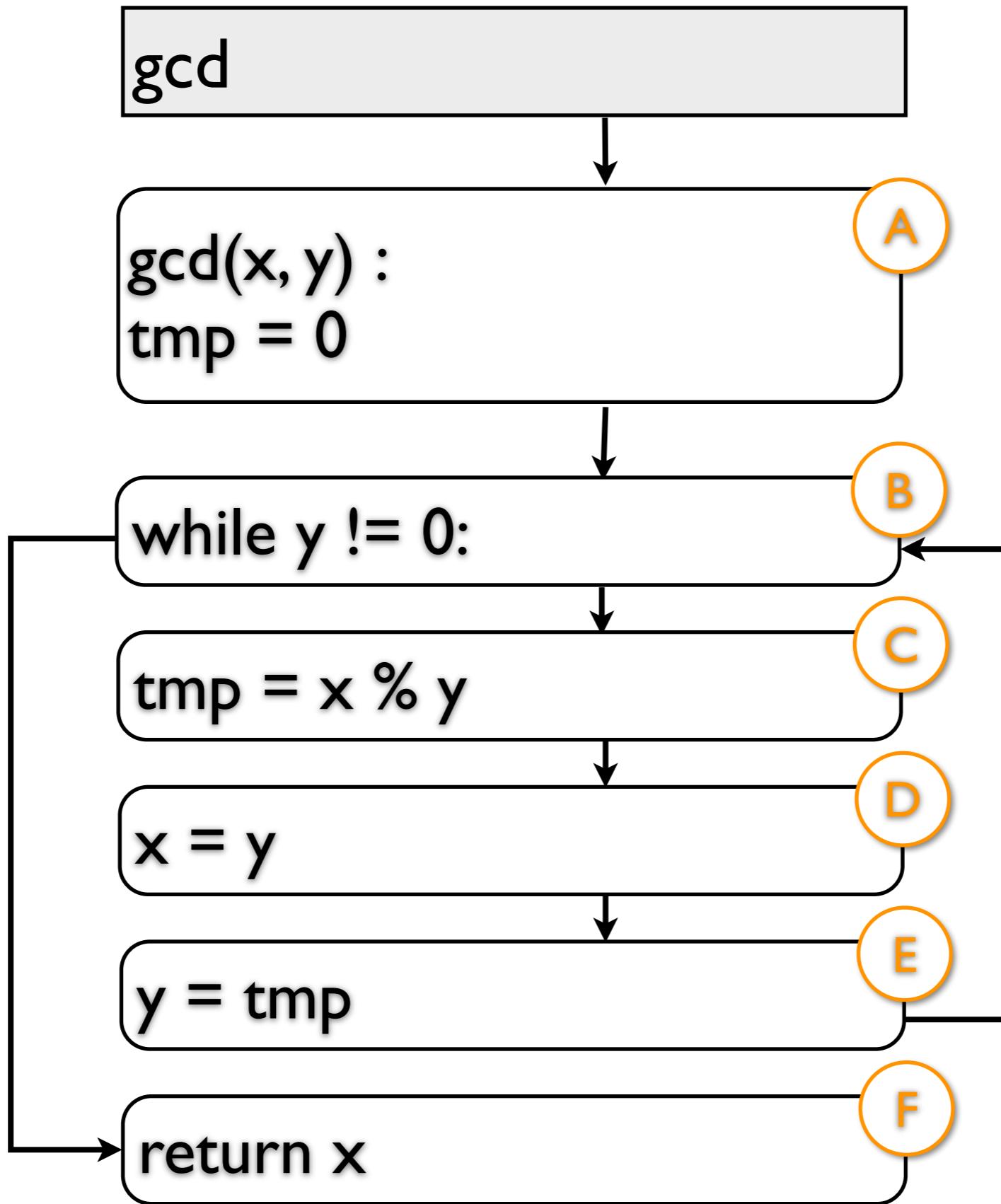
Caminos en el CFG

# CFG

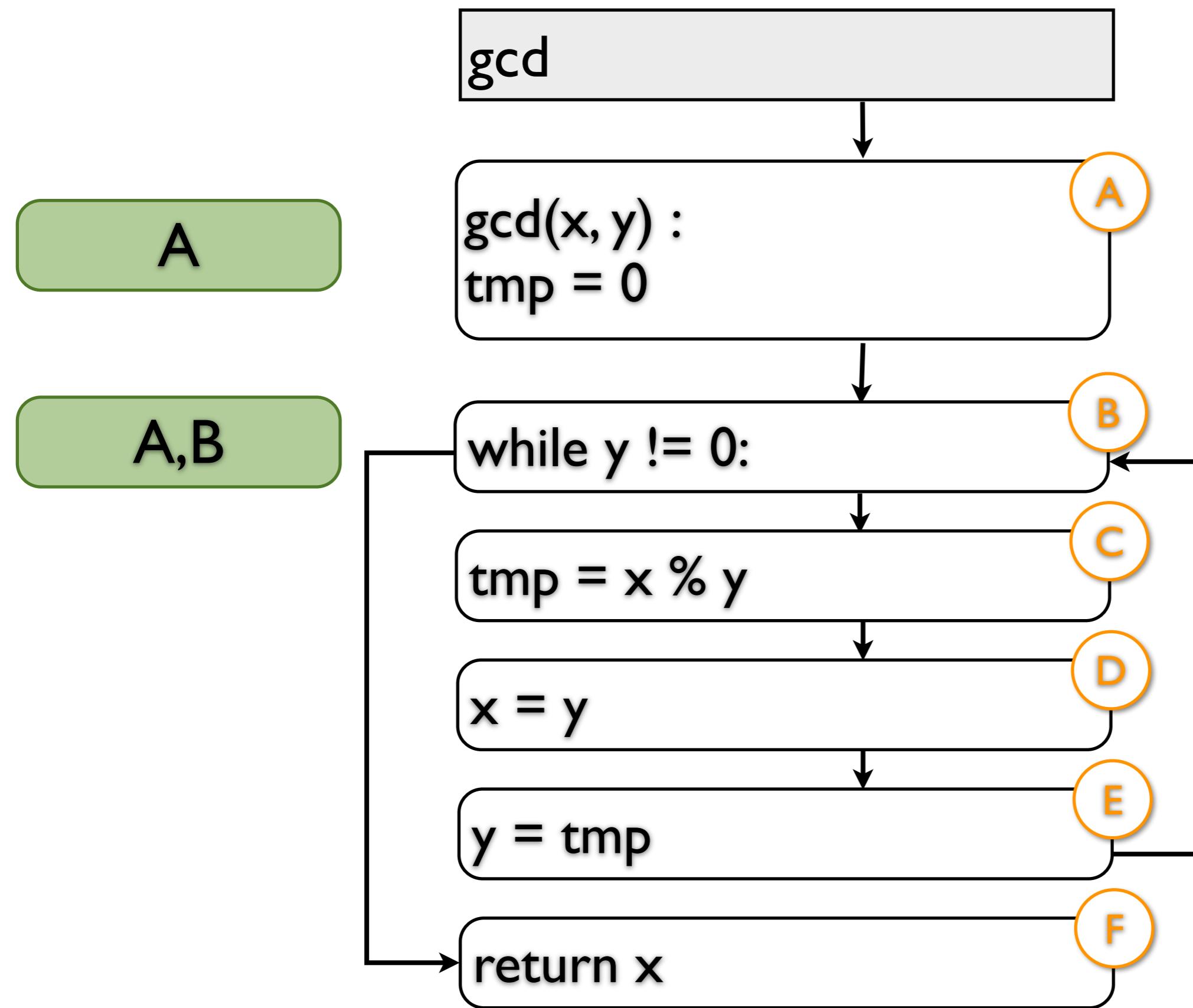


# CFG

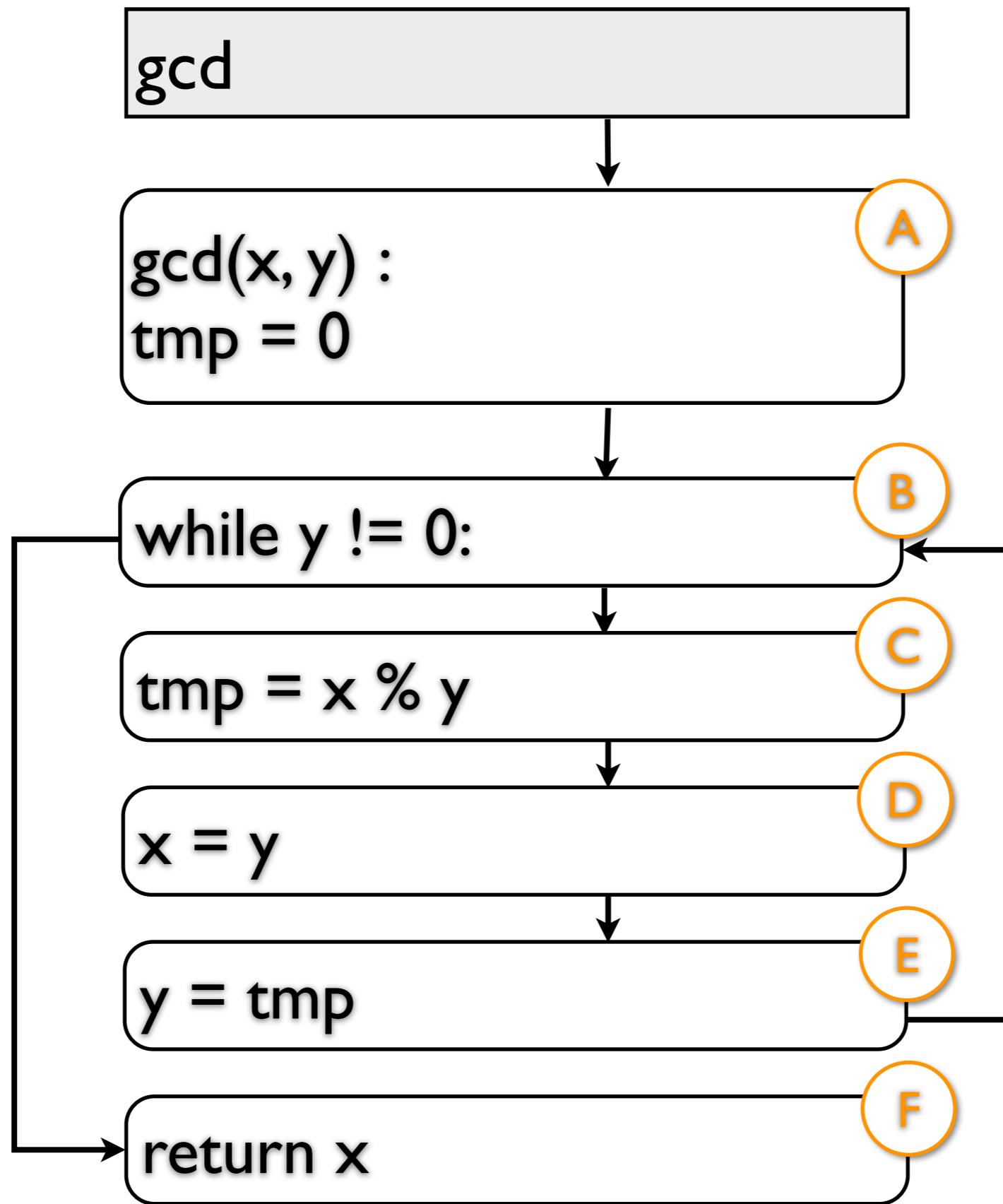
A



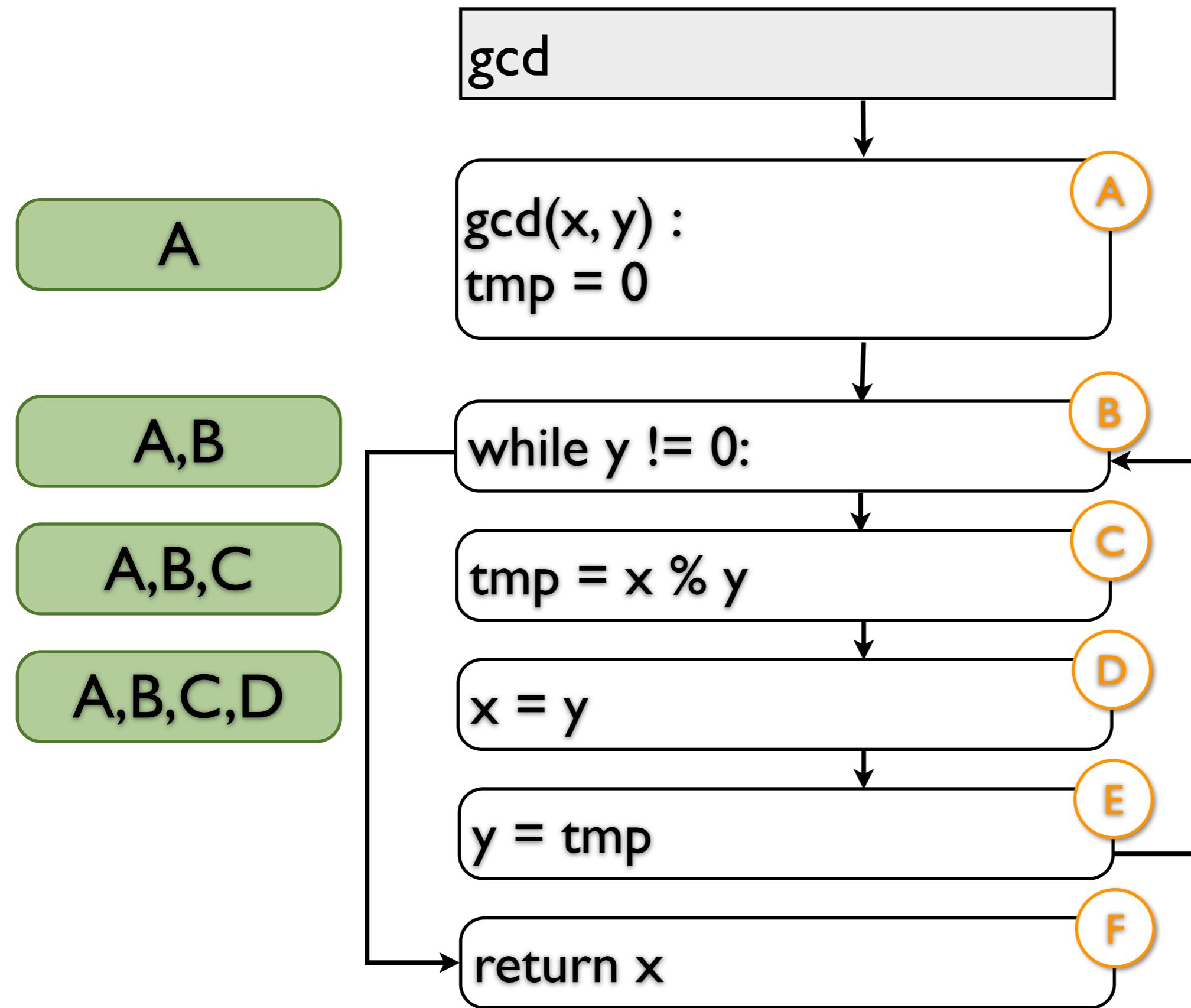
# CFG



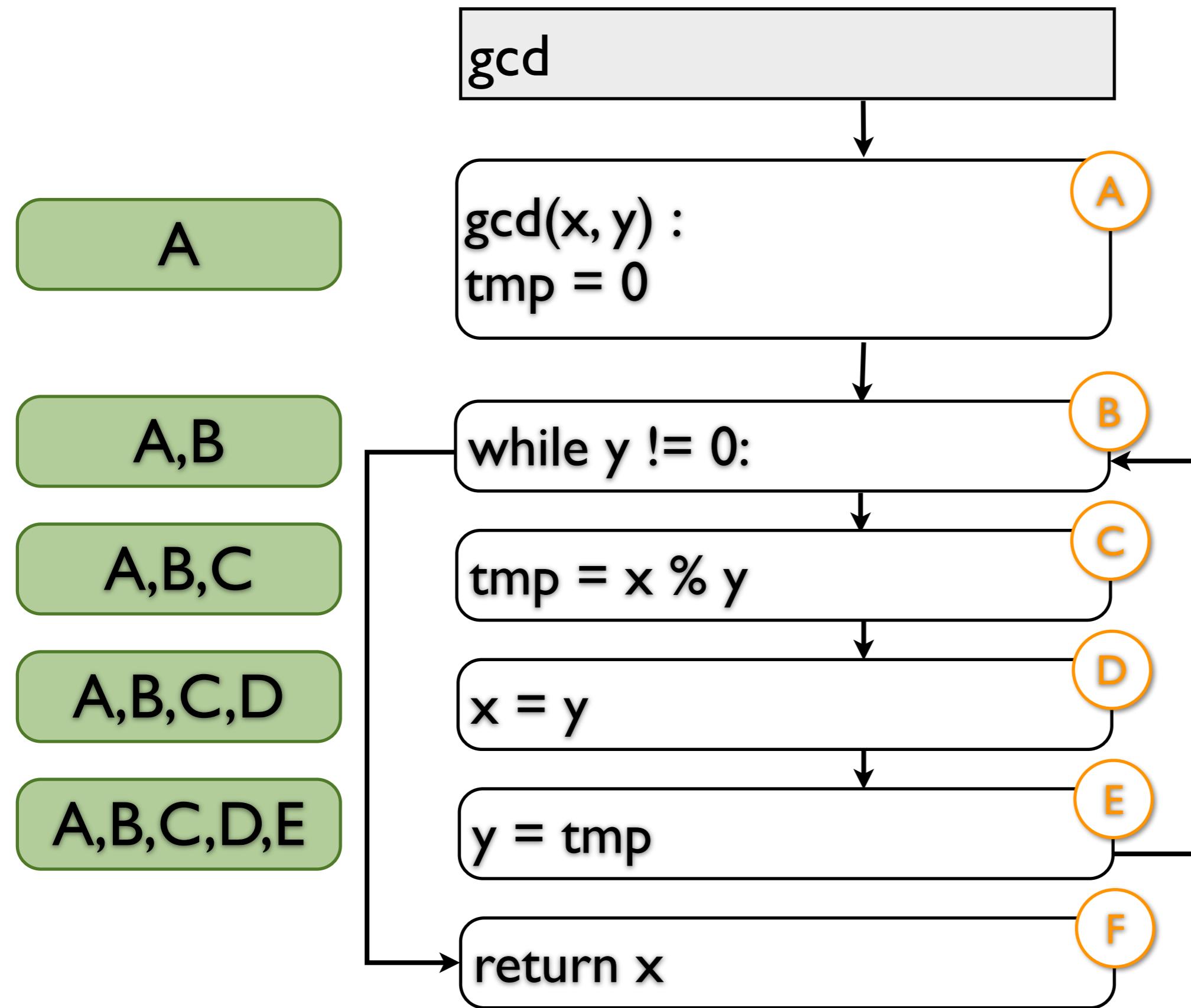
# CFG



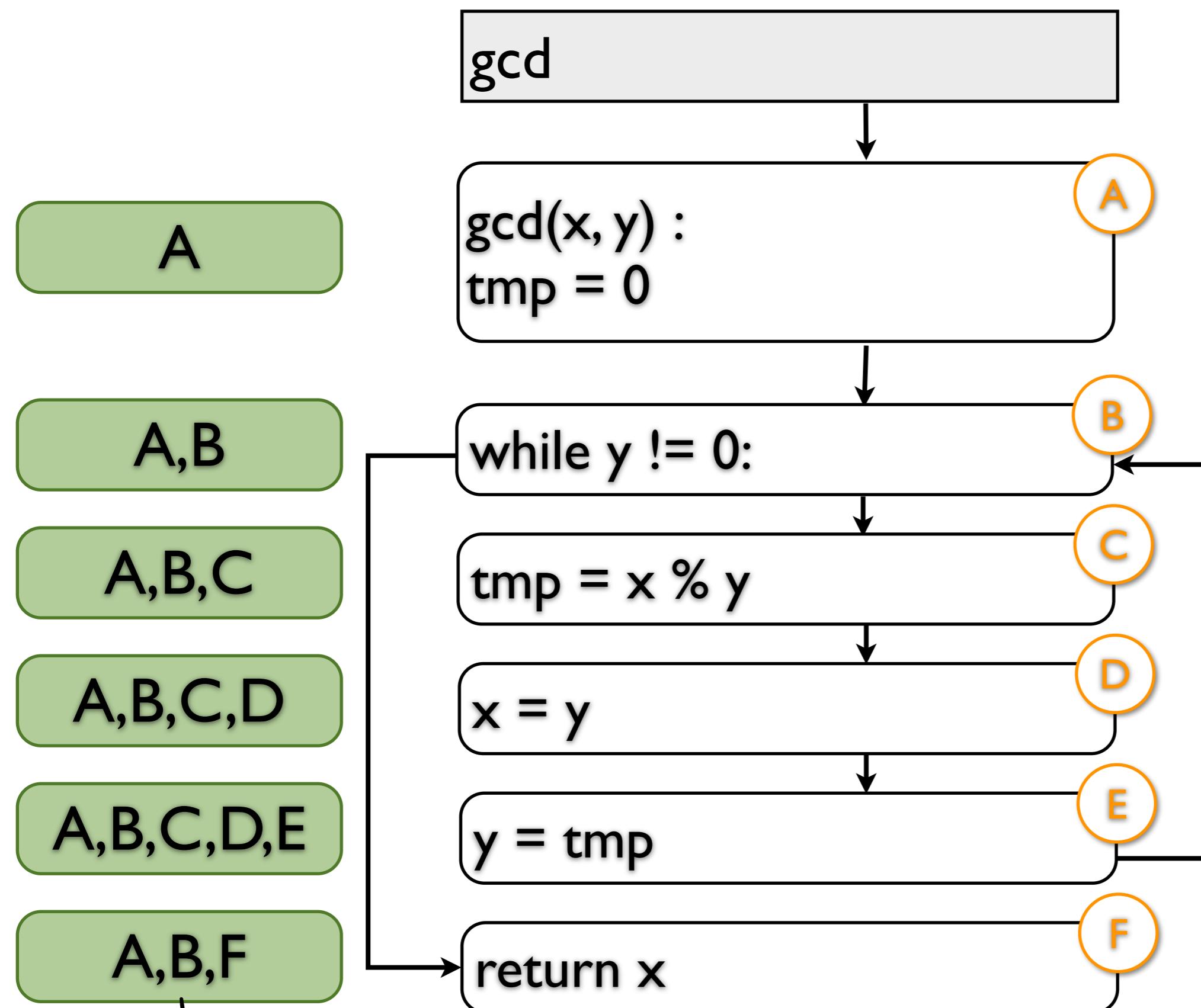
# CFG



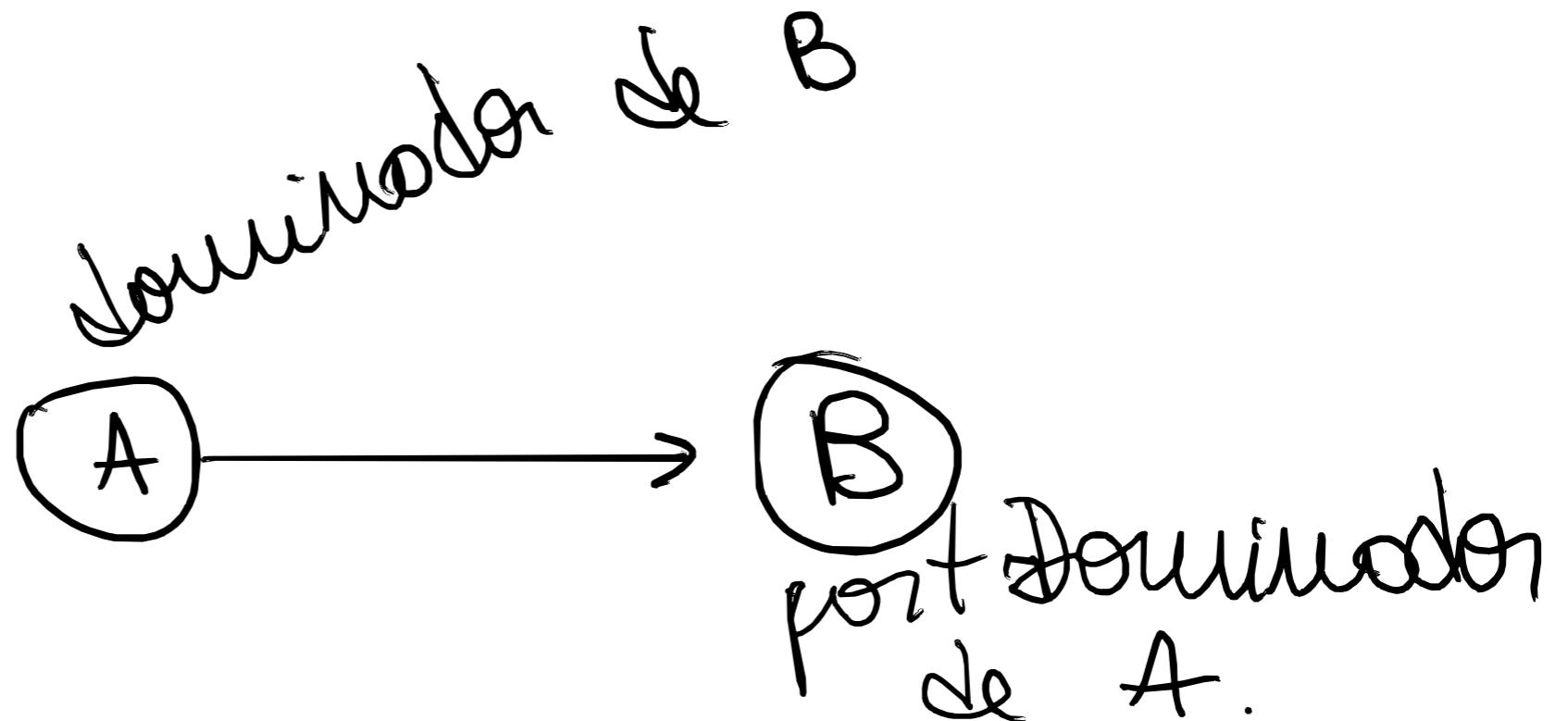
# CFG



# CFG



usos los NECESARIOS p/ llegar al modo



# Post Dominators

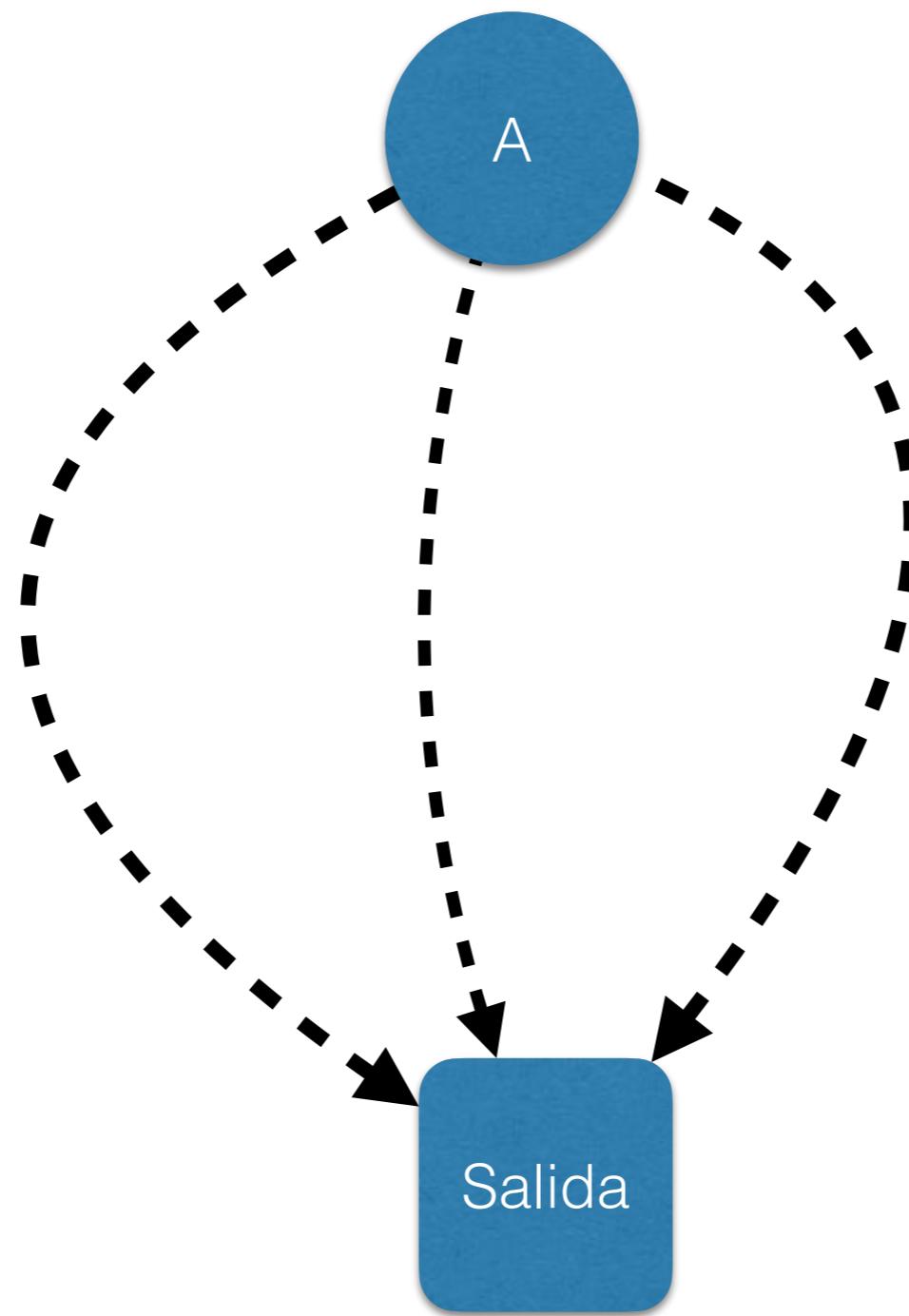
- El Nodo B “post-domina” al Nodo A si todos los caminos desde A a la salida deben pasar por B
- Post dominador Inmediato: El post-dominador más cercano en todo camino hacia el nodo de salida
- Son Dominadores vistos en reversa (caminos desde el nodo de salida)

# B post-domina a A si:



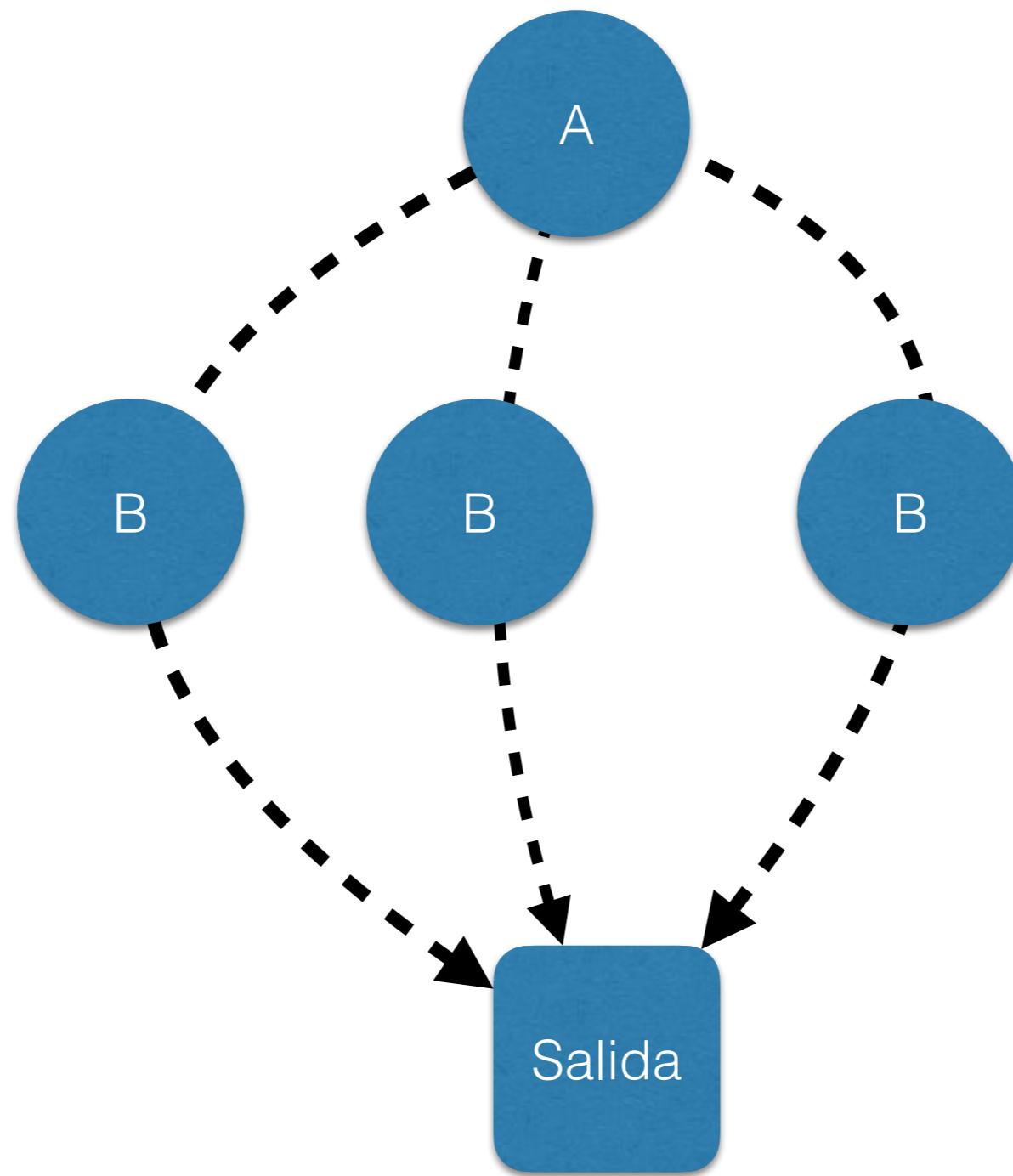
Caminos en el CFG

# B post-domina a A si:



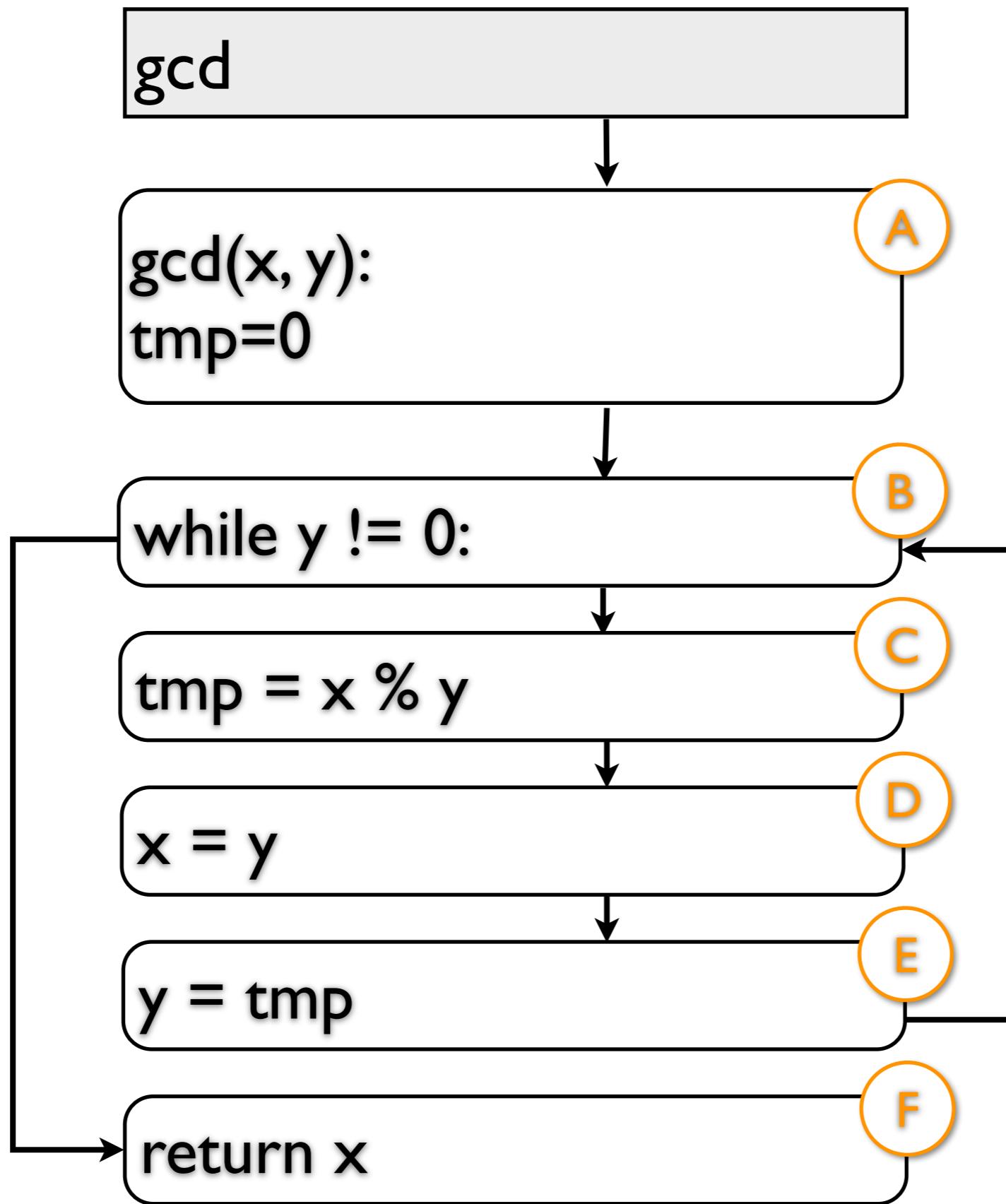
Caminos en el CFG

# B post-domina a A si:

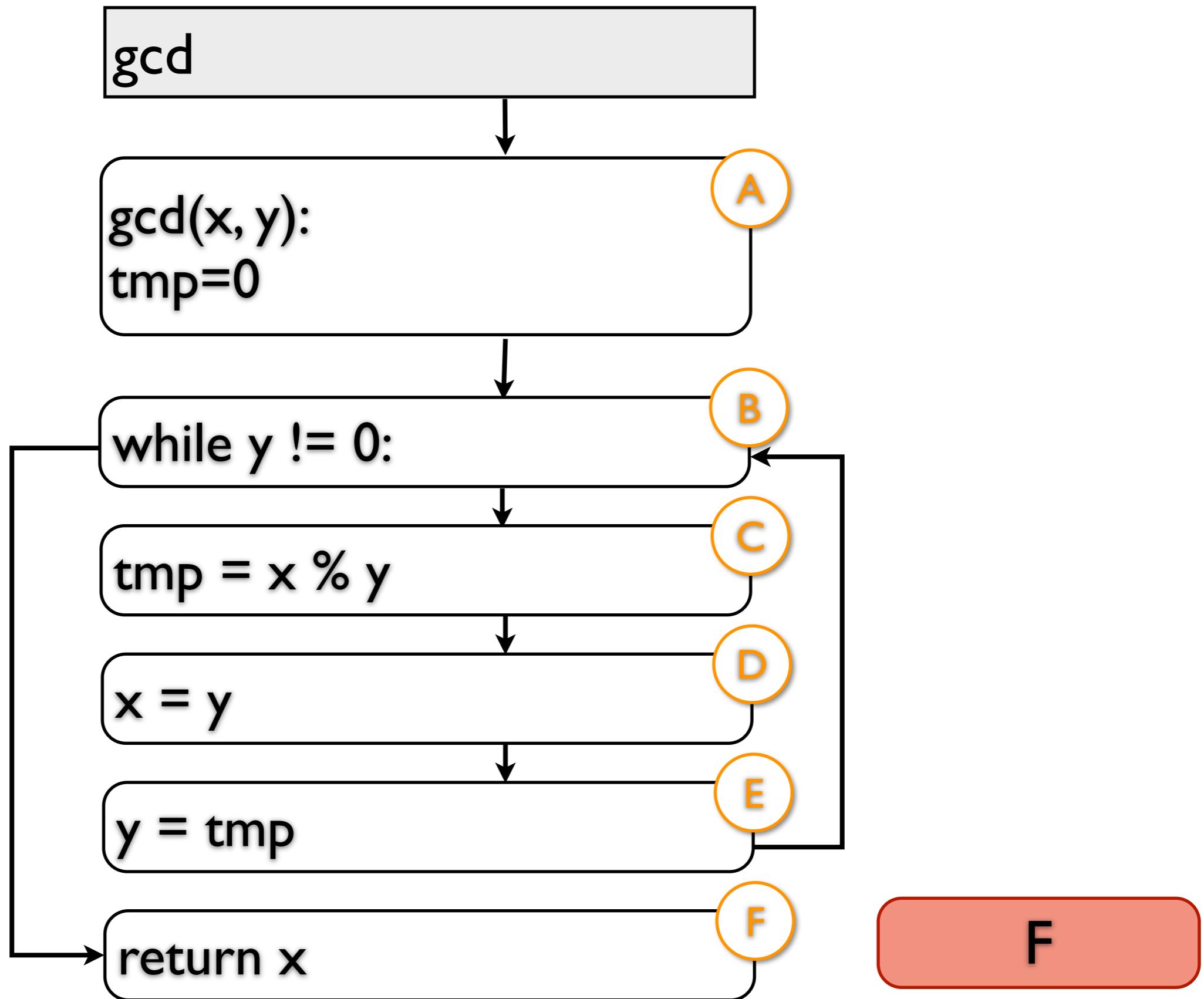


Caminos en el CFG

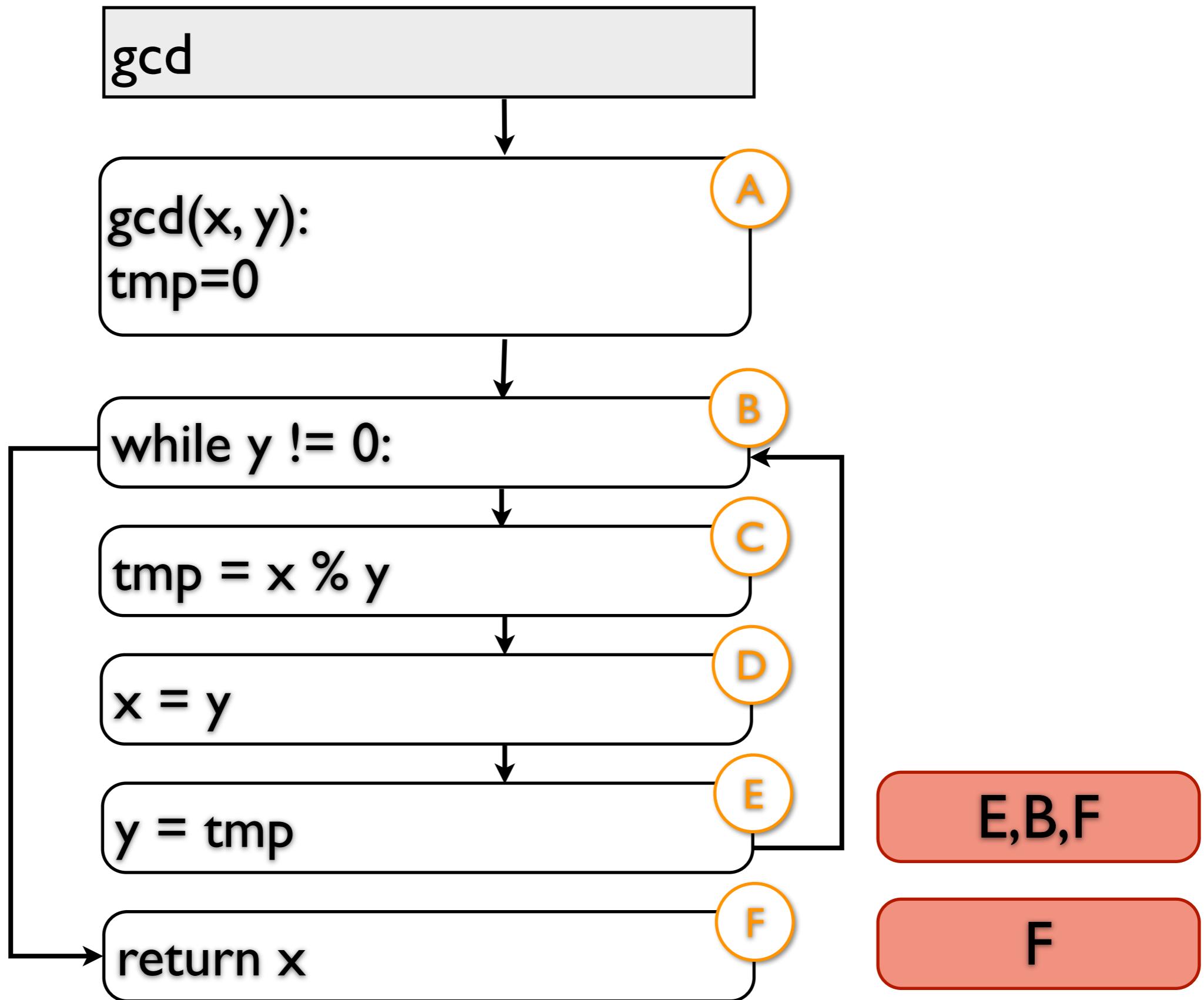
# CFG



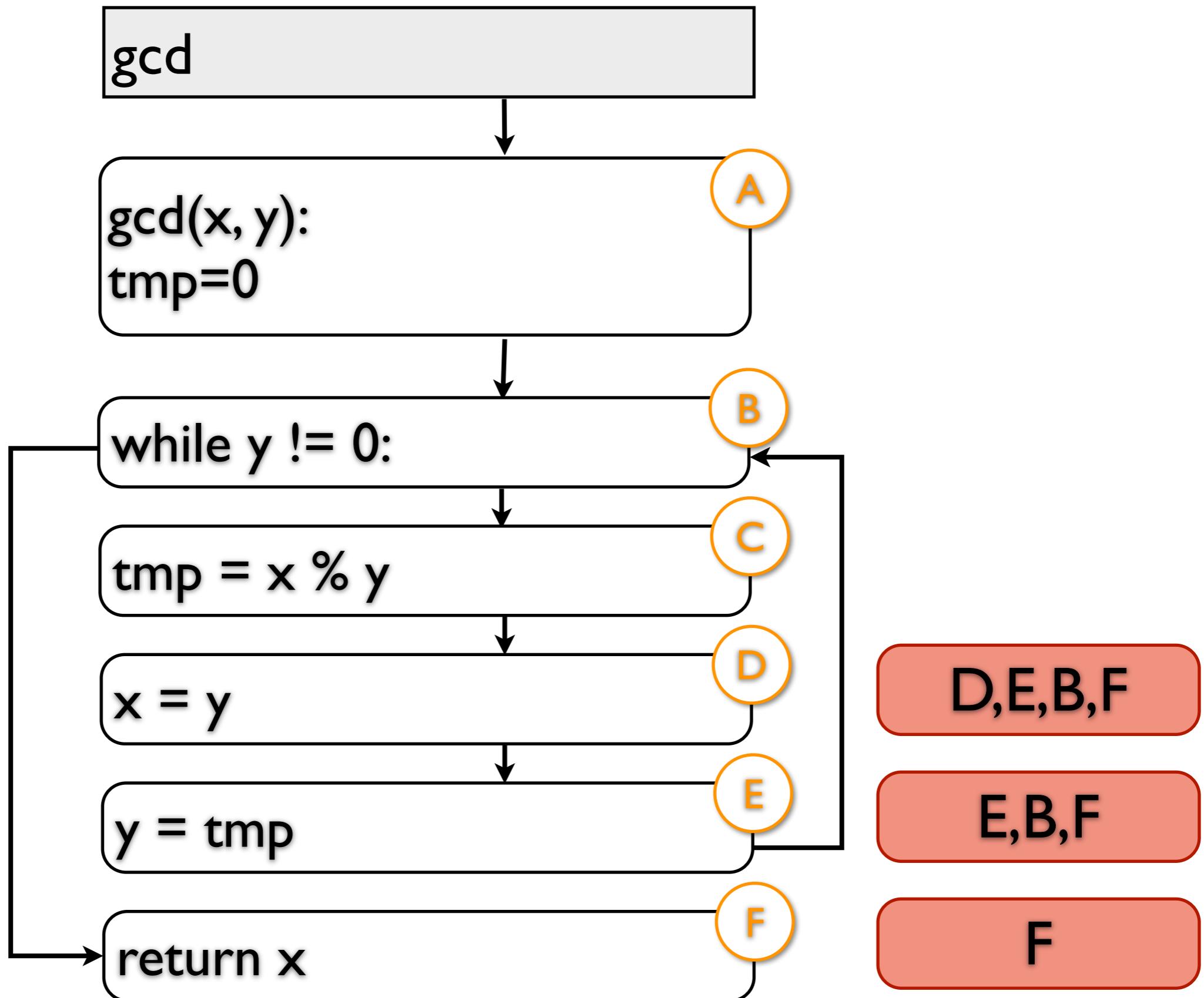
# CFG



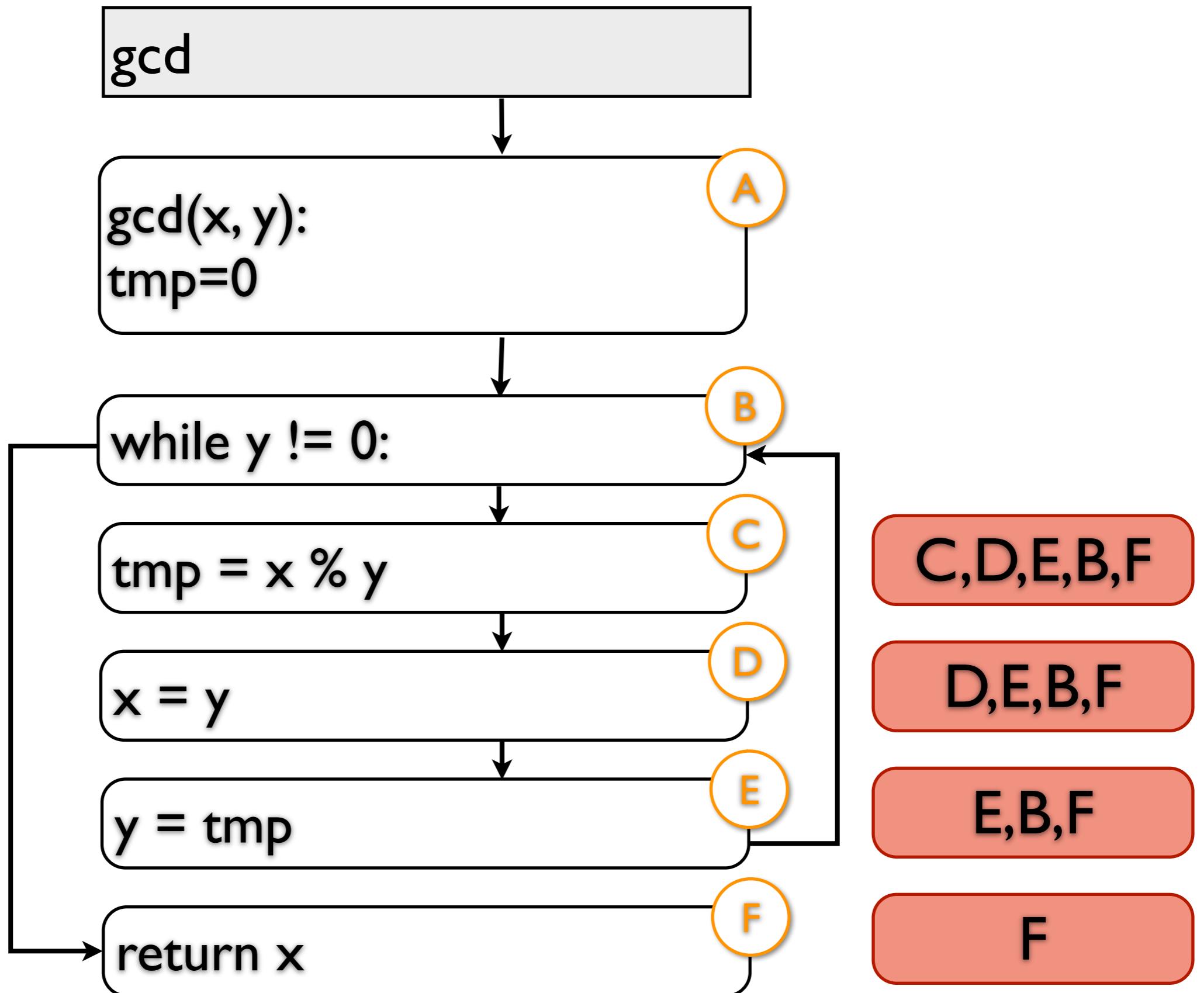
# CFG



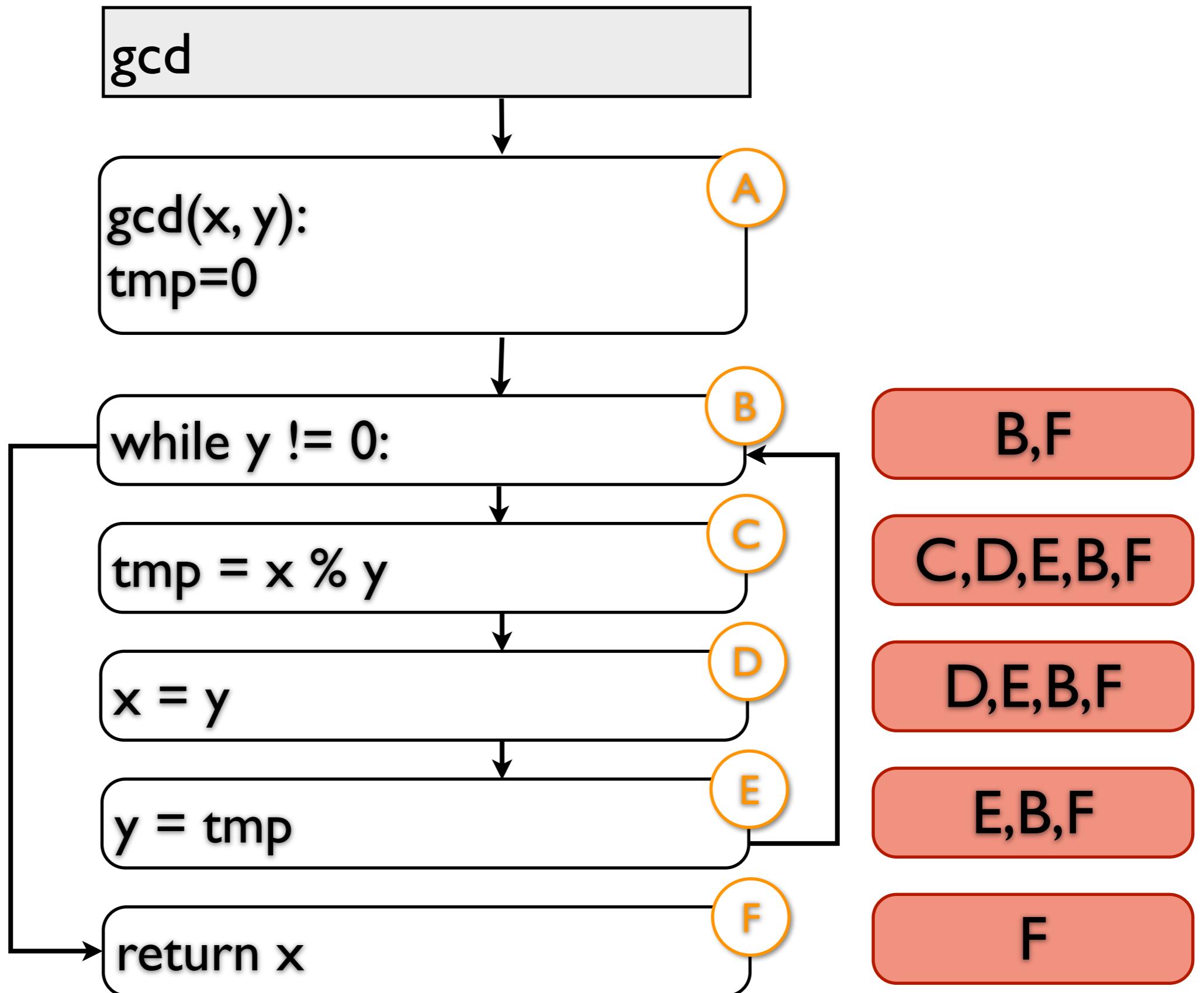
# CFG



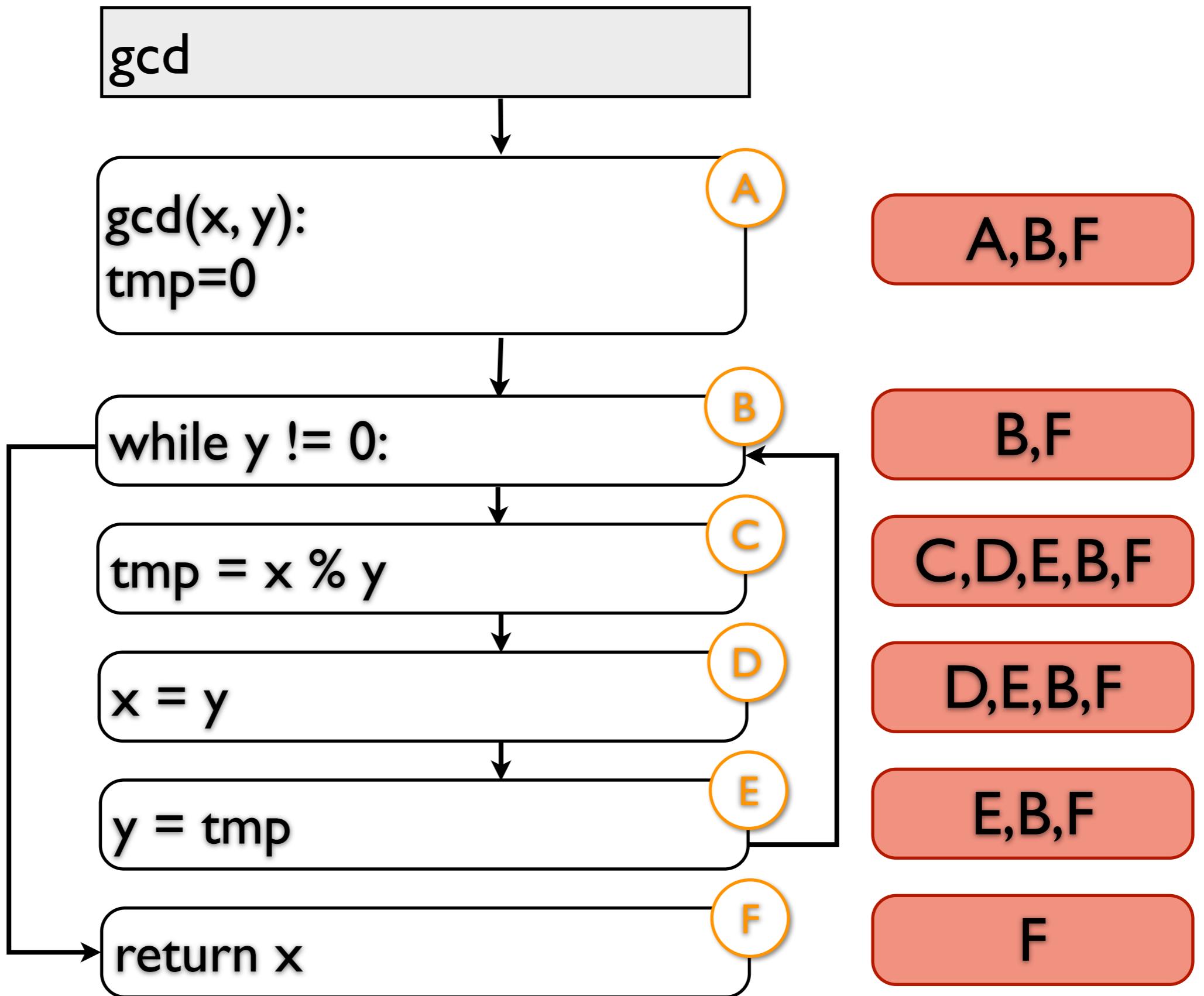
# CFG



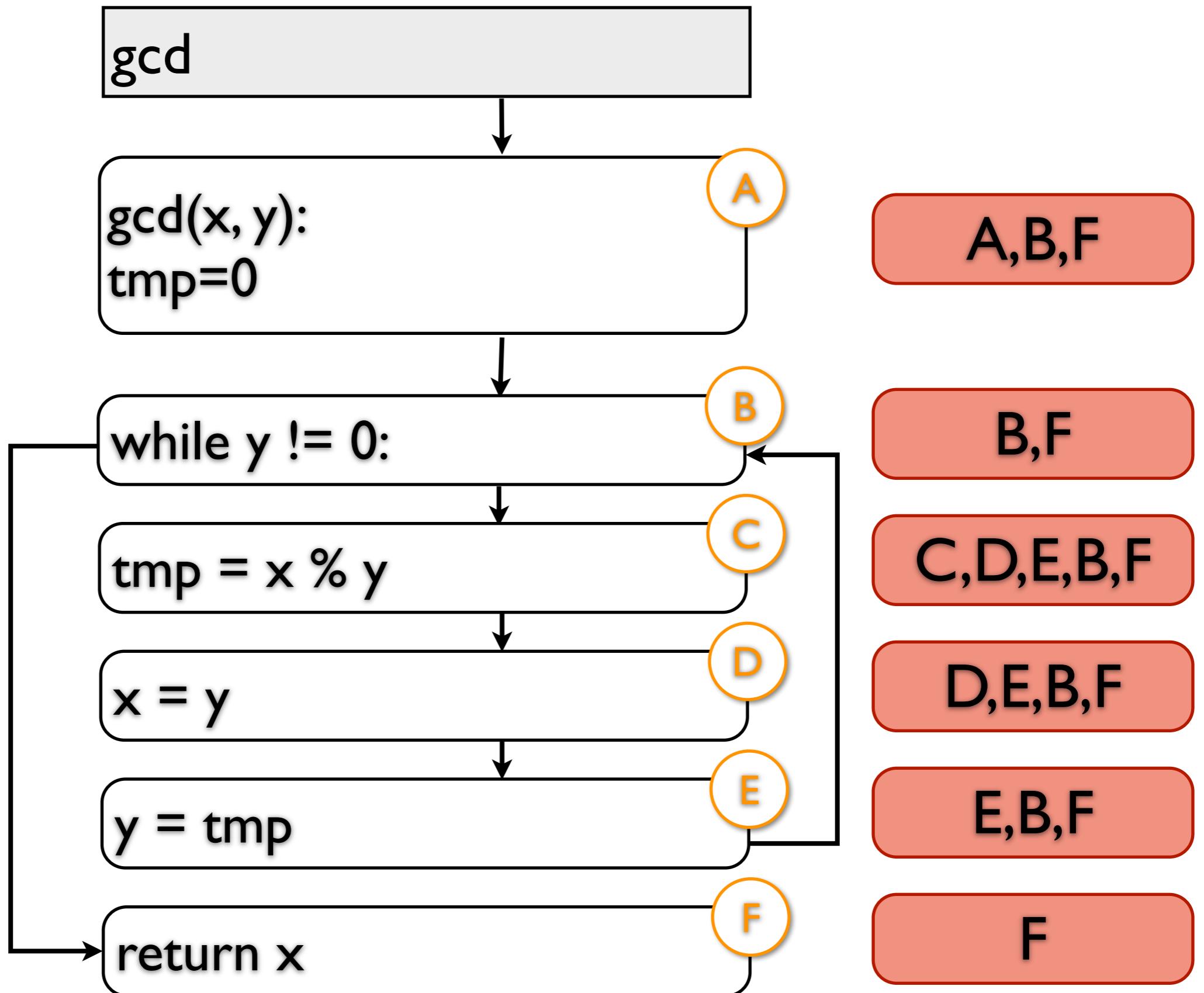
# CFG



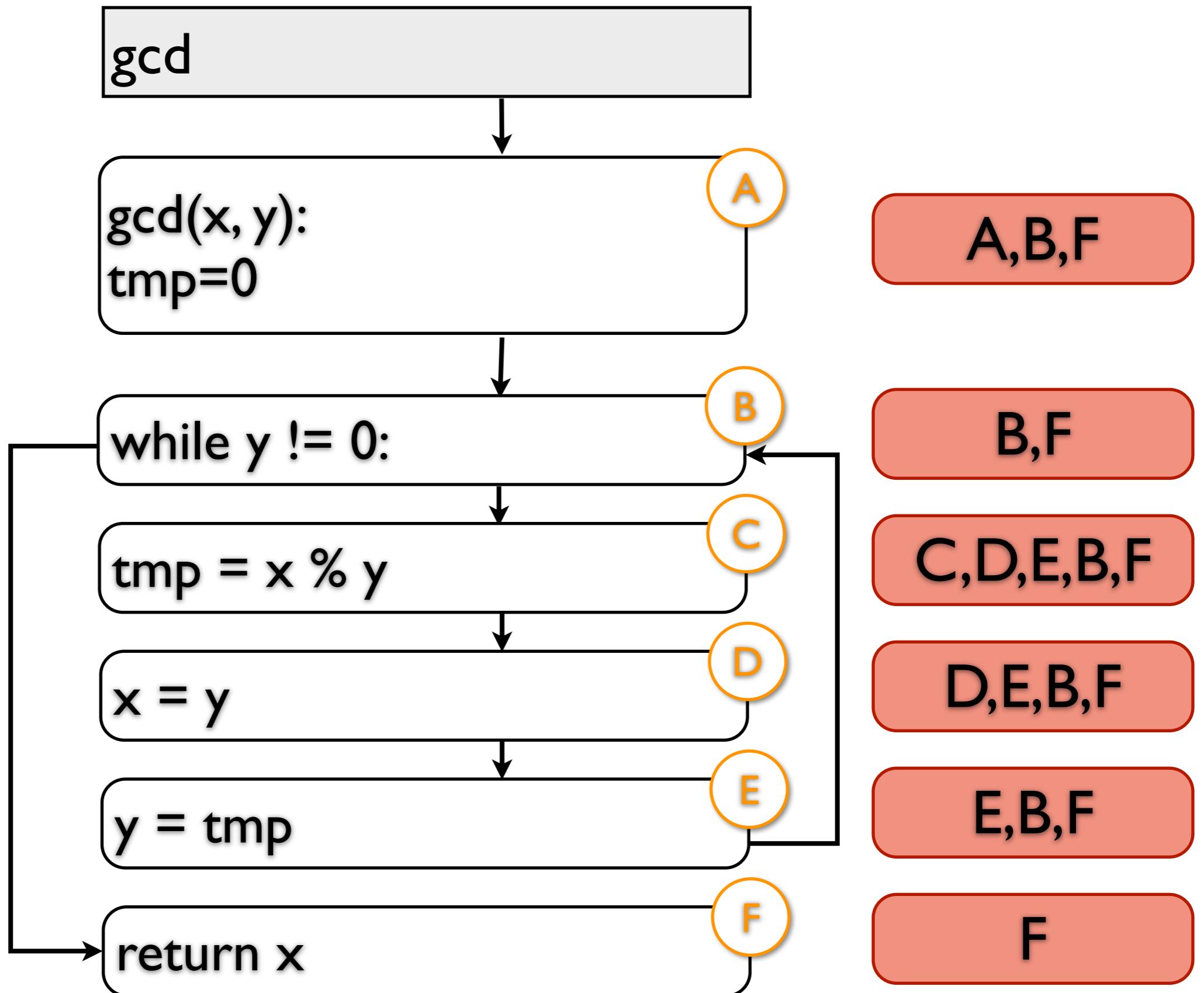
# CFG



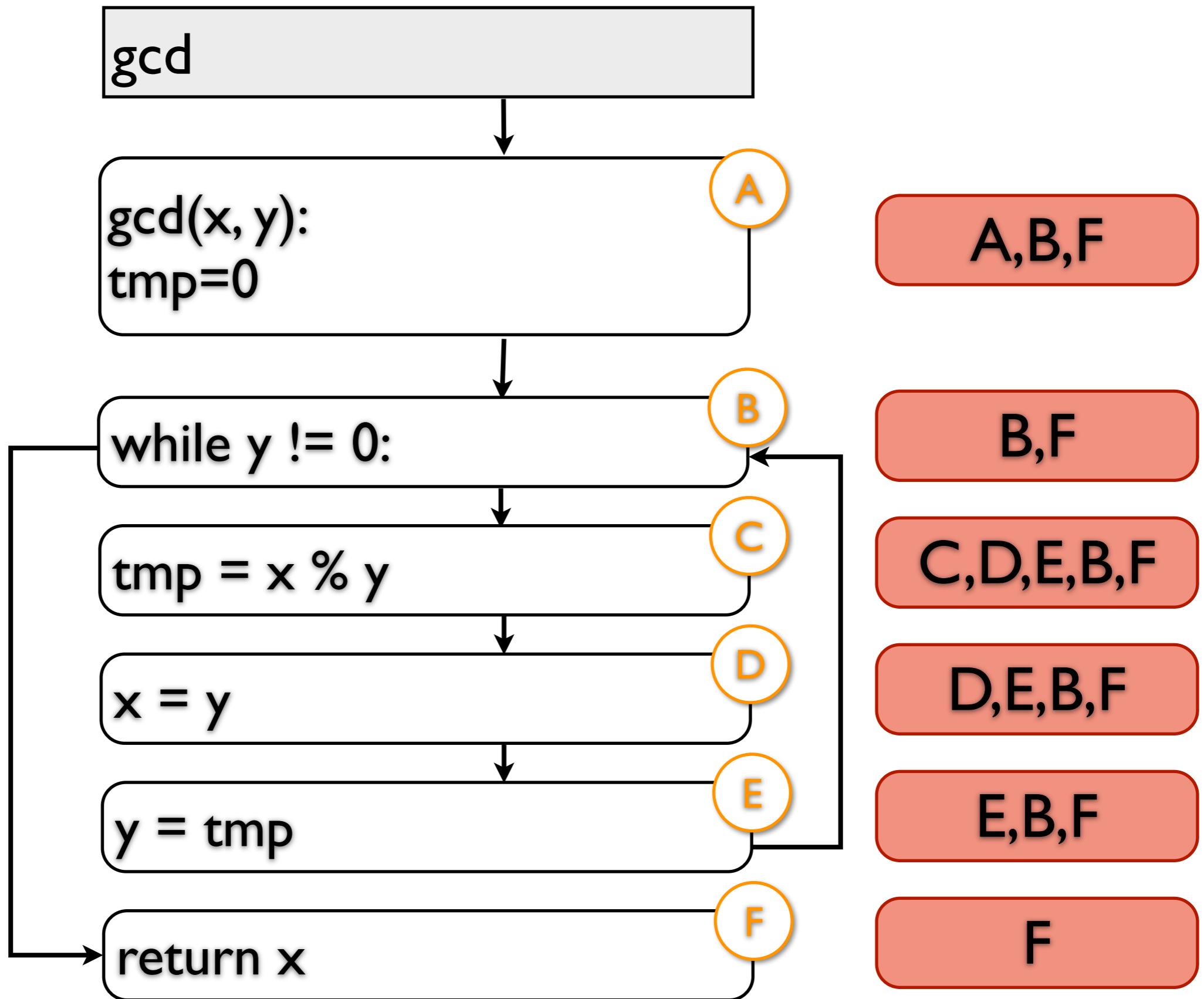
# CFG



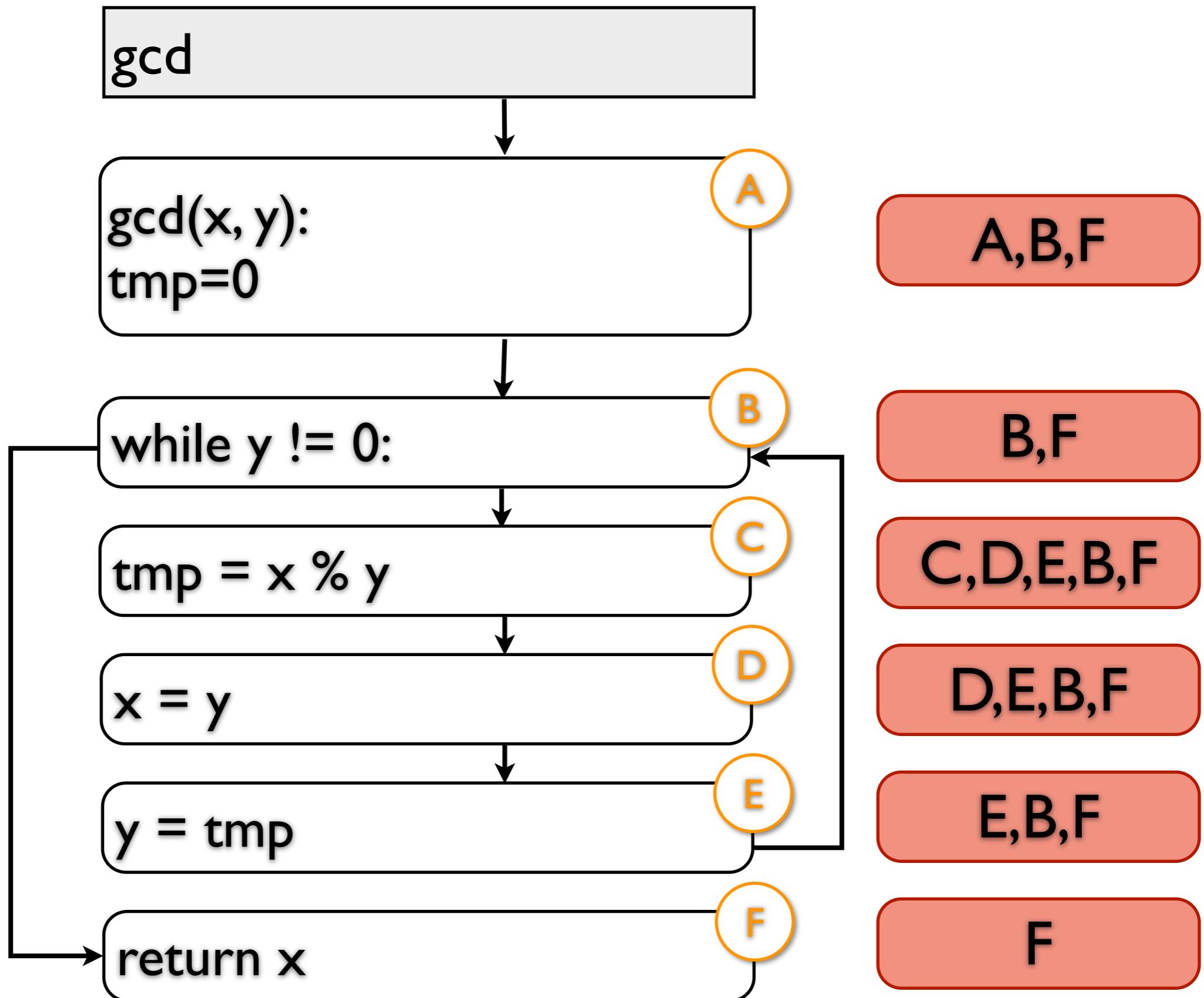
# CFG



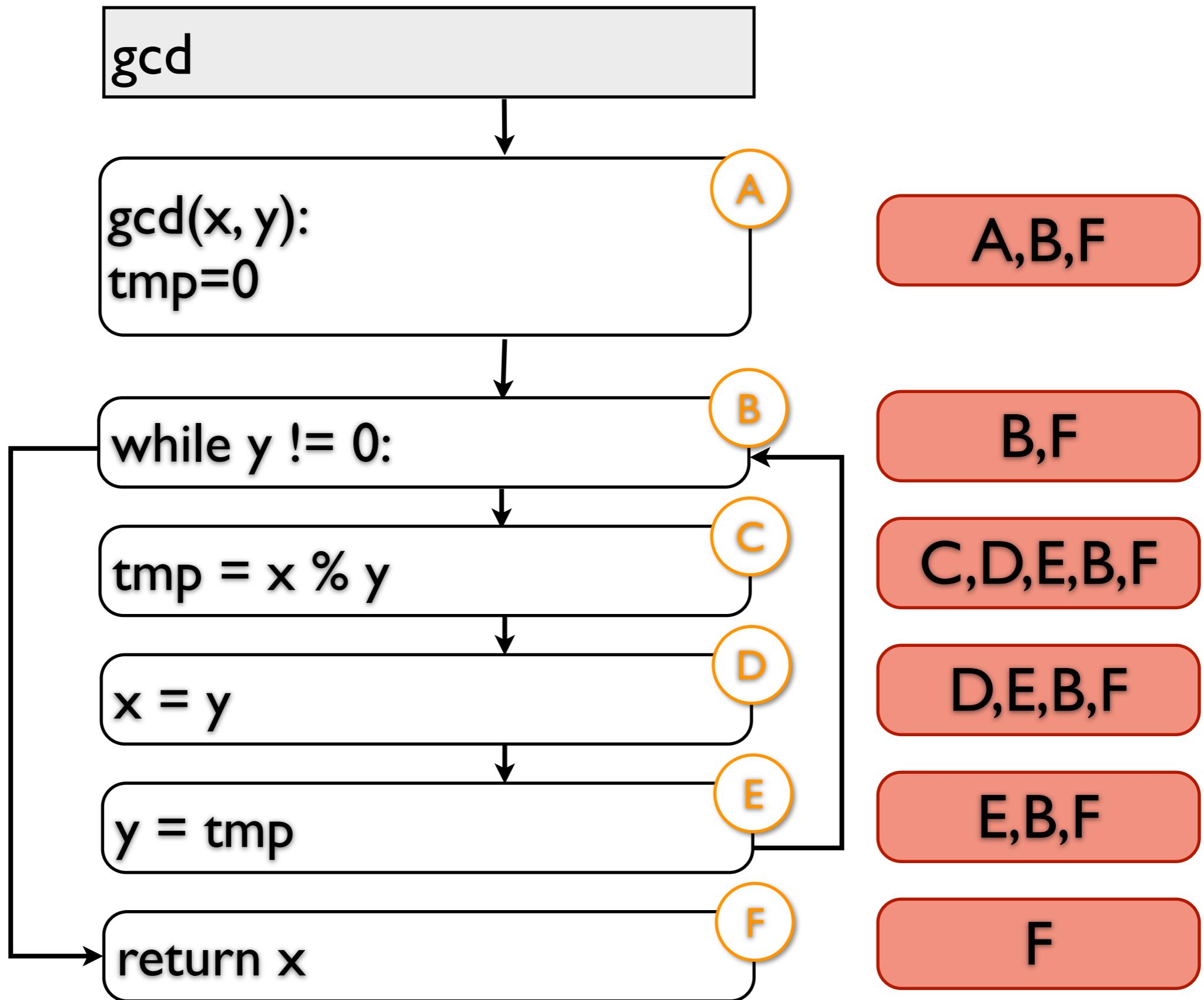
# CFG



# CFG



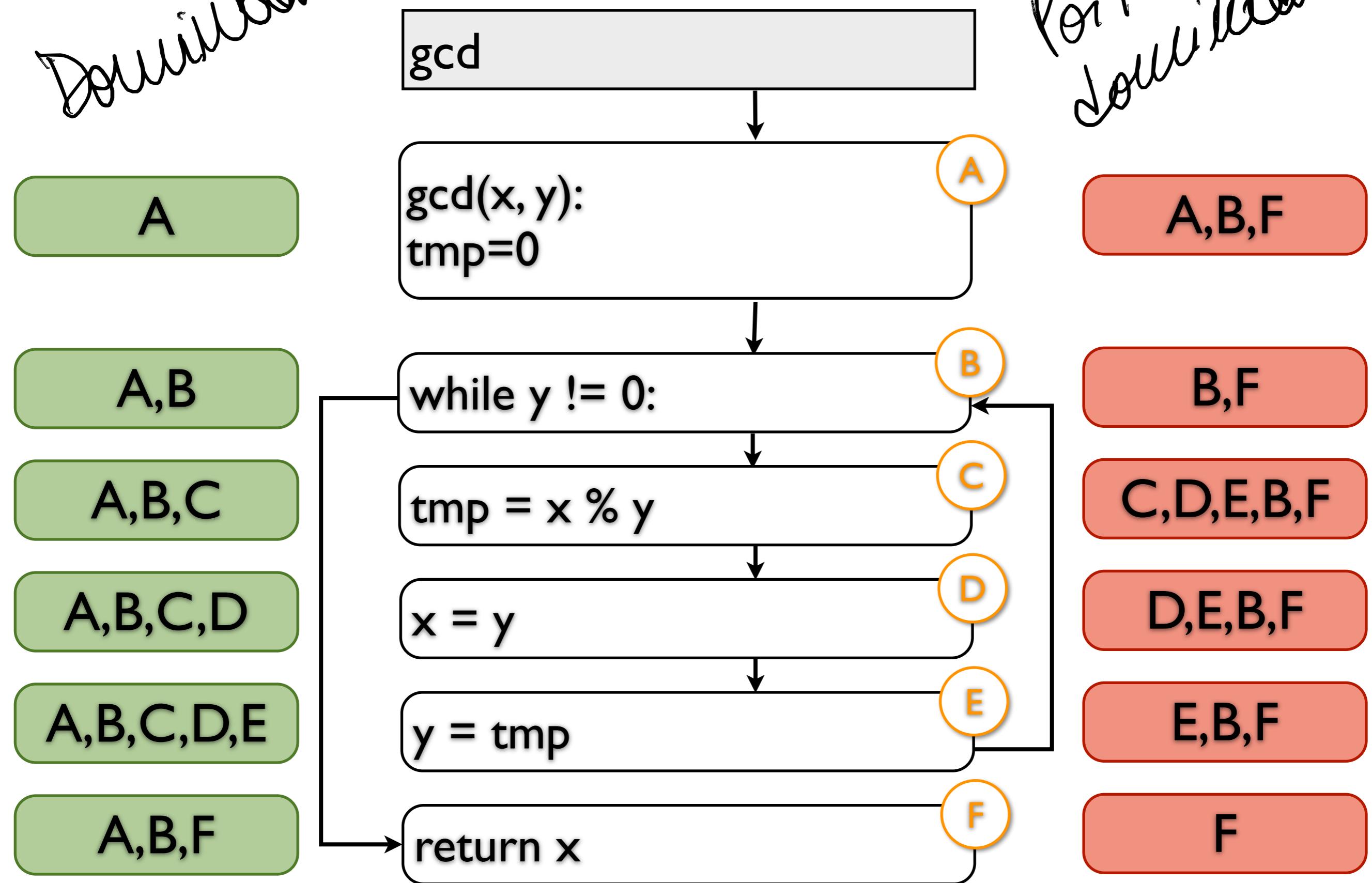
# CFG



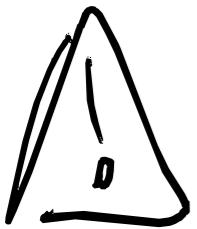
CFG

Dominators

Post dominators



# Control Dependence



- B es control-dependiente de A si:
  - A domina a B
  - B no “post-domina” a A
  - A tiene al menos dos sucesores
  - B post-domina a un sucesor de A



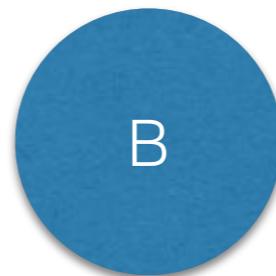
# B es control-dependent de A si:

A es una **decisión**  
(2 sucesores)



**Al menos un**  
camino desde A  
no pasa por B

**Todos** los  
caminos a B  
pasan por A



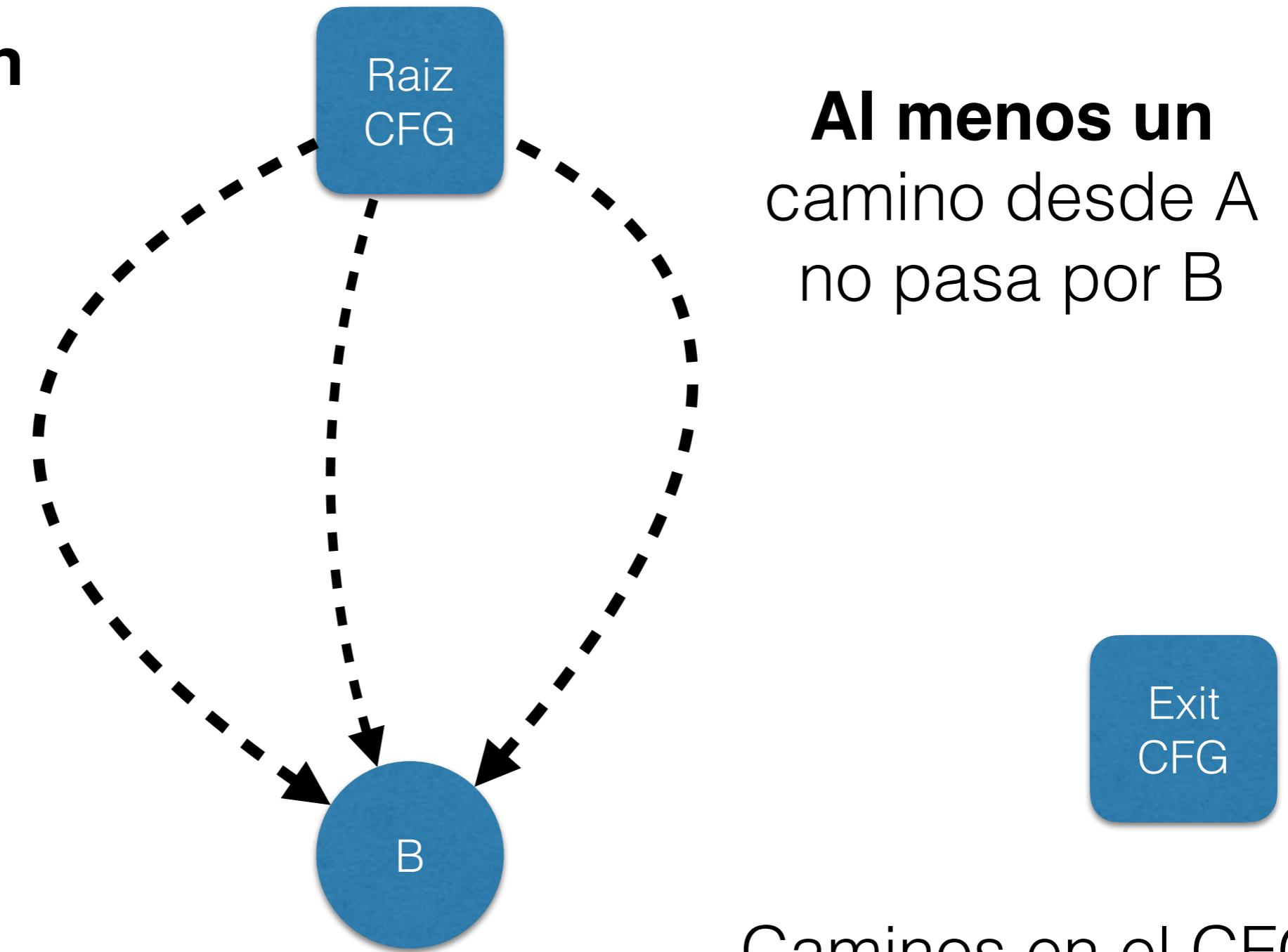
Caminos en el CFG

# B es control-dependent de A si:

A es una **decisión**  
(2 sucesores)

**Al menos un**  
camino desde A  
no pasa por B

**Todos** los  
caminos a B  
pasan por A

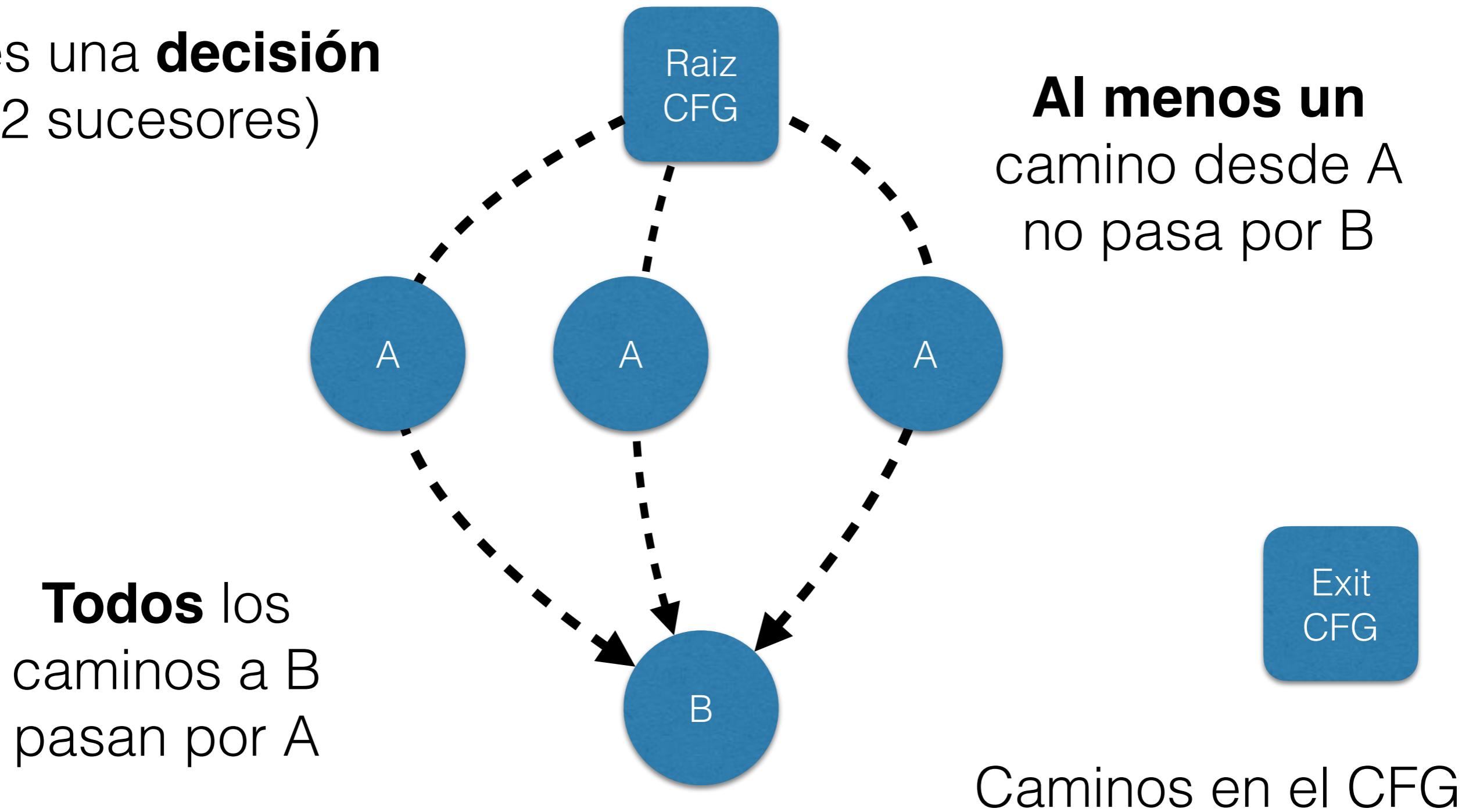


Caminos en el CFG

# B es control-dependent de A si:

A es una **decisión**  
(2 sucesores)

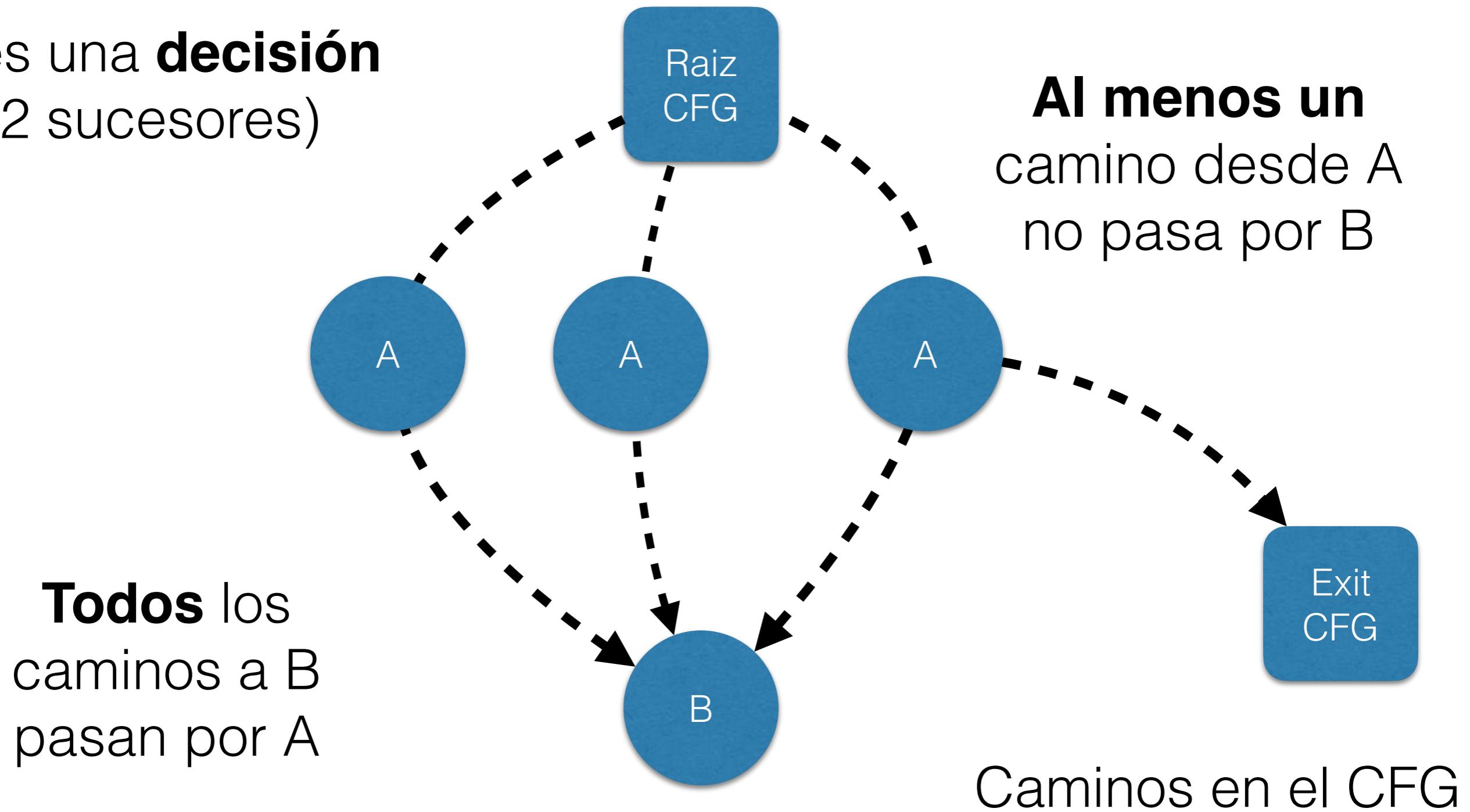
**Al menos un**  
camino desde A  
no pasa por B



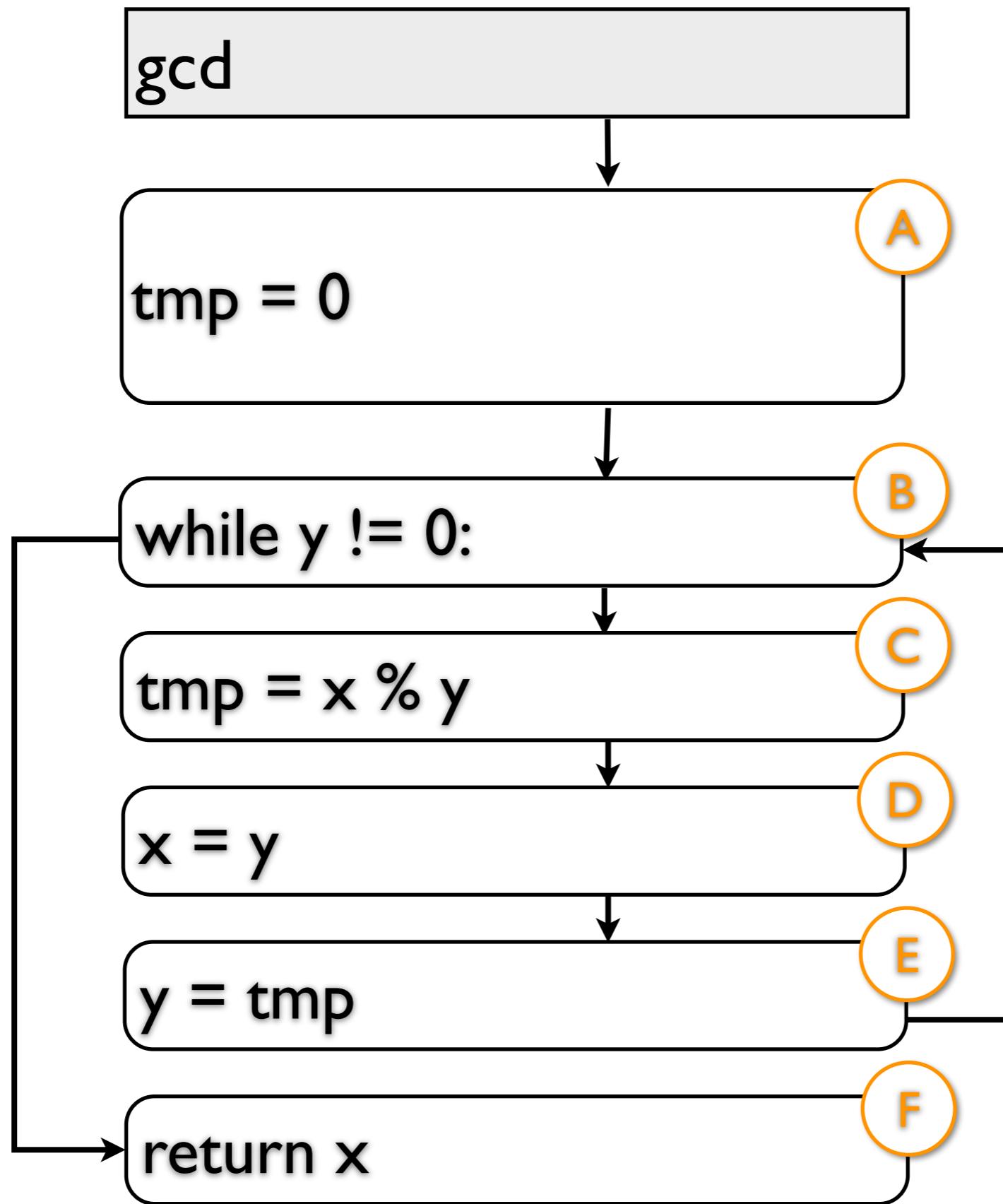
# B es control-dependent de A si:

A es una **decisión**  
(2 sucesores)

**Al menos un**  
camino desde A  
no pasa por B

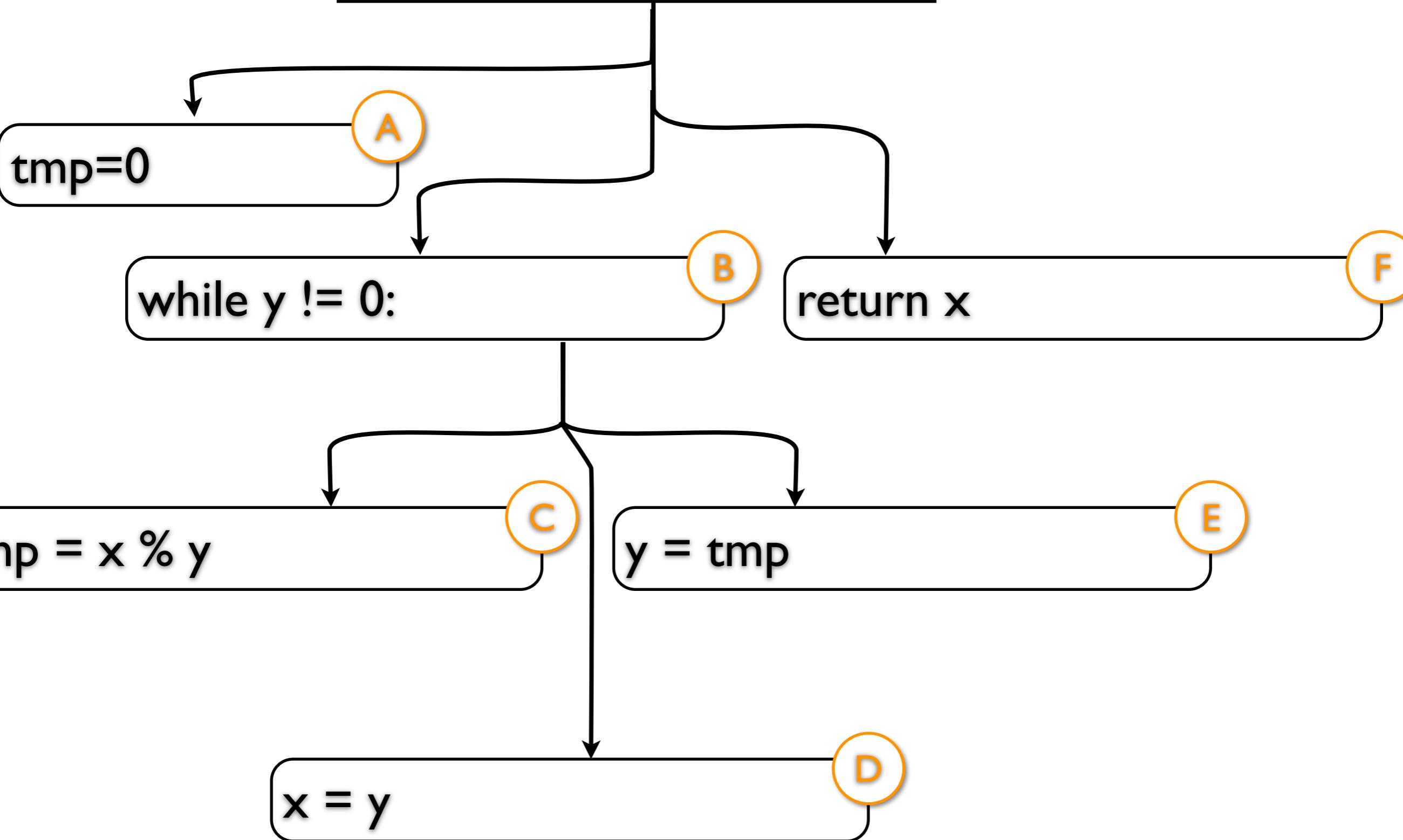


# CFG

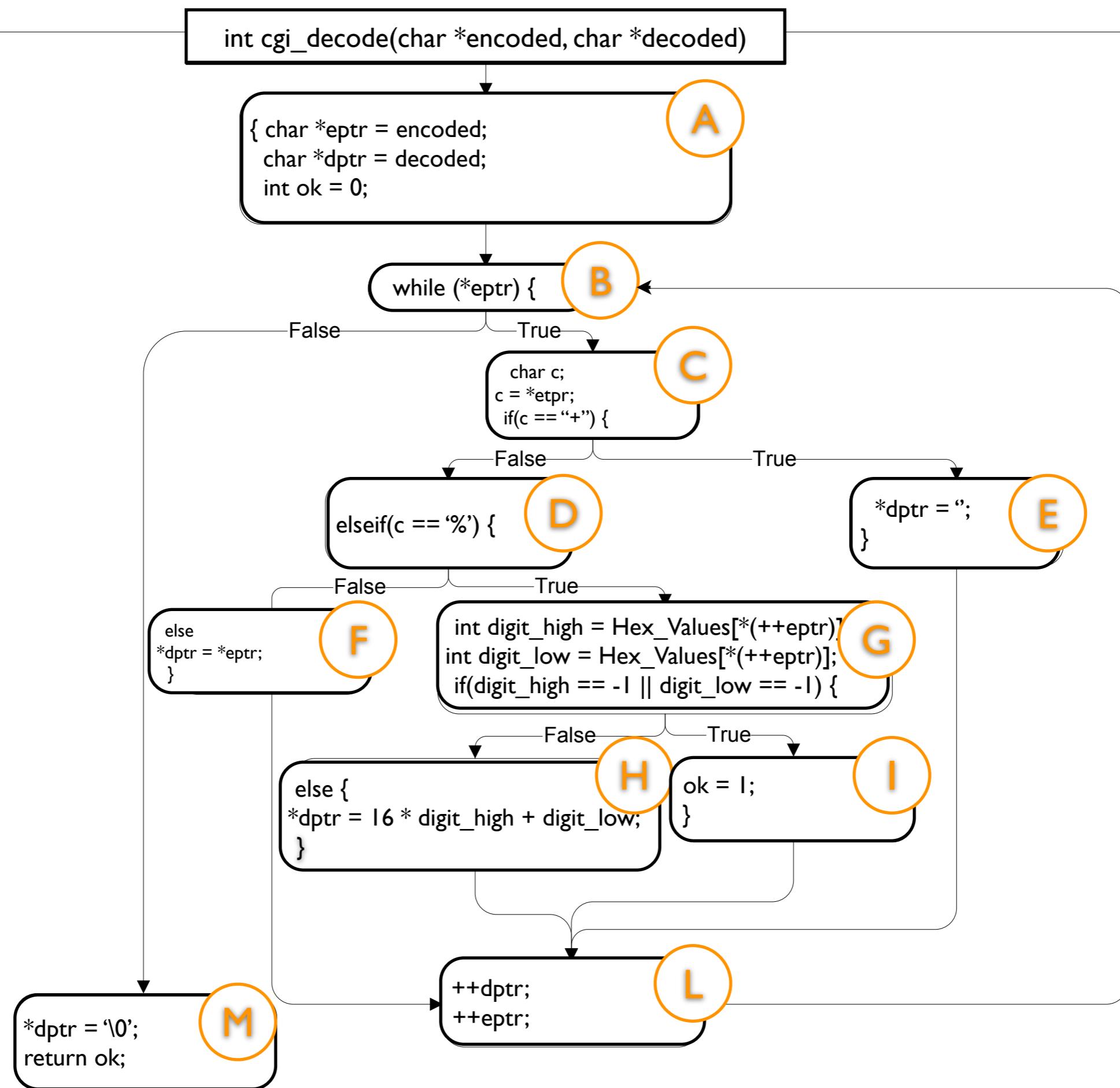


CDG

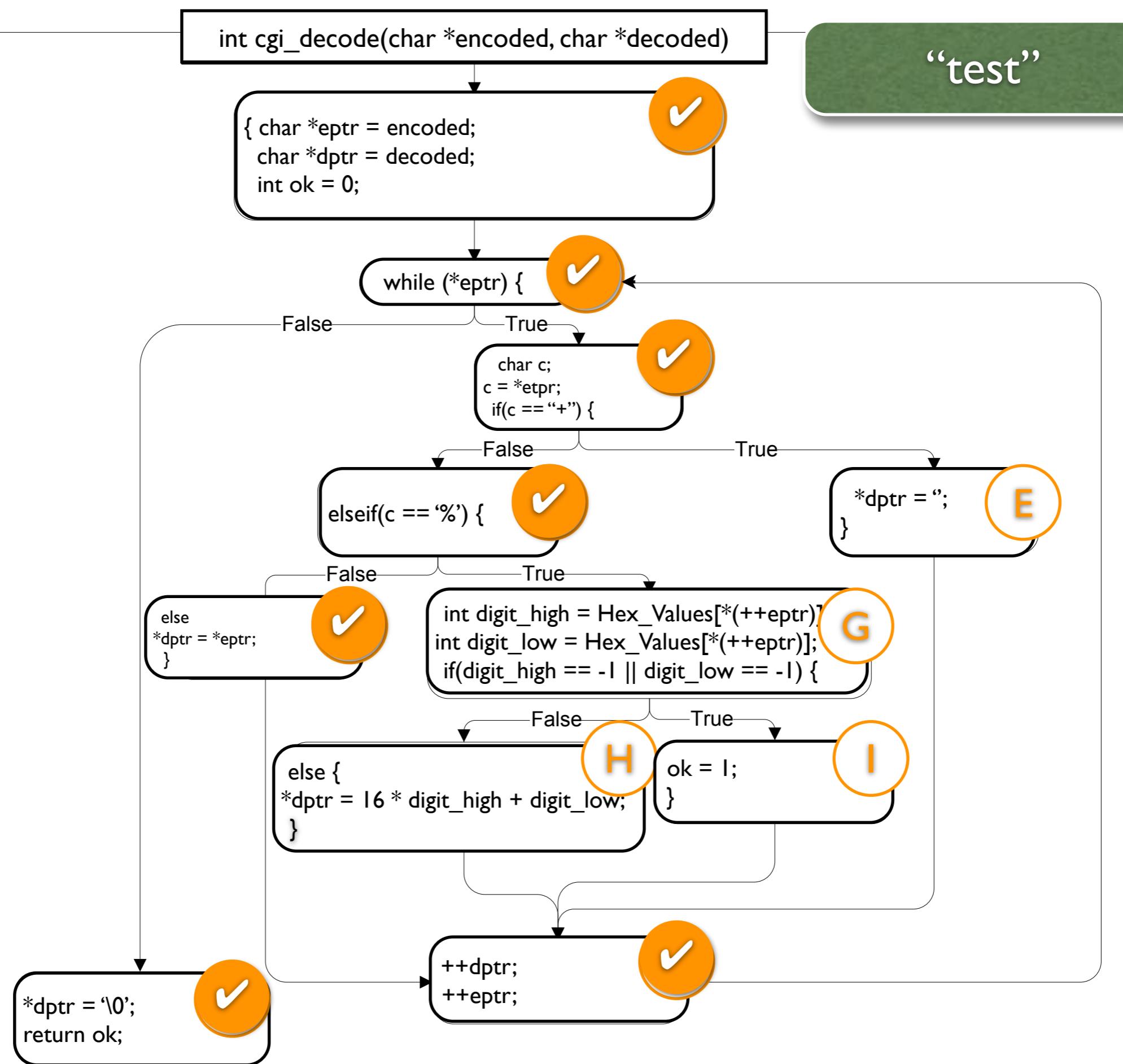
gcd (root)



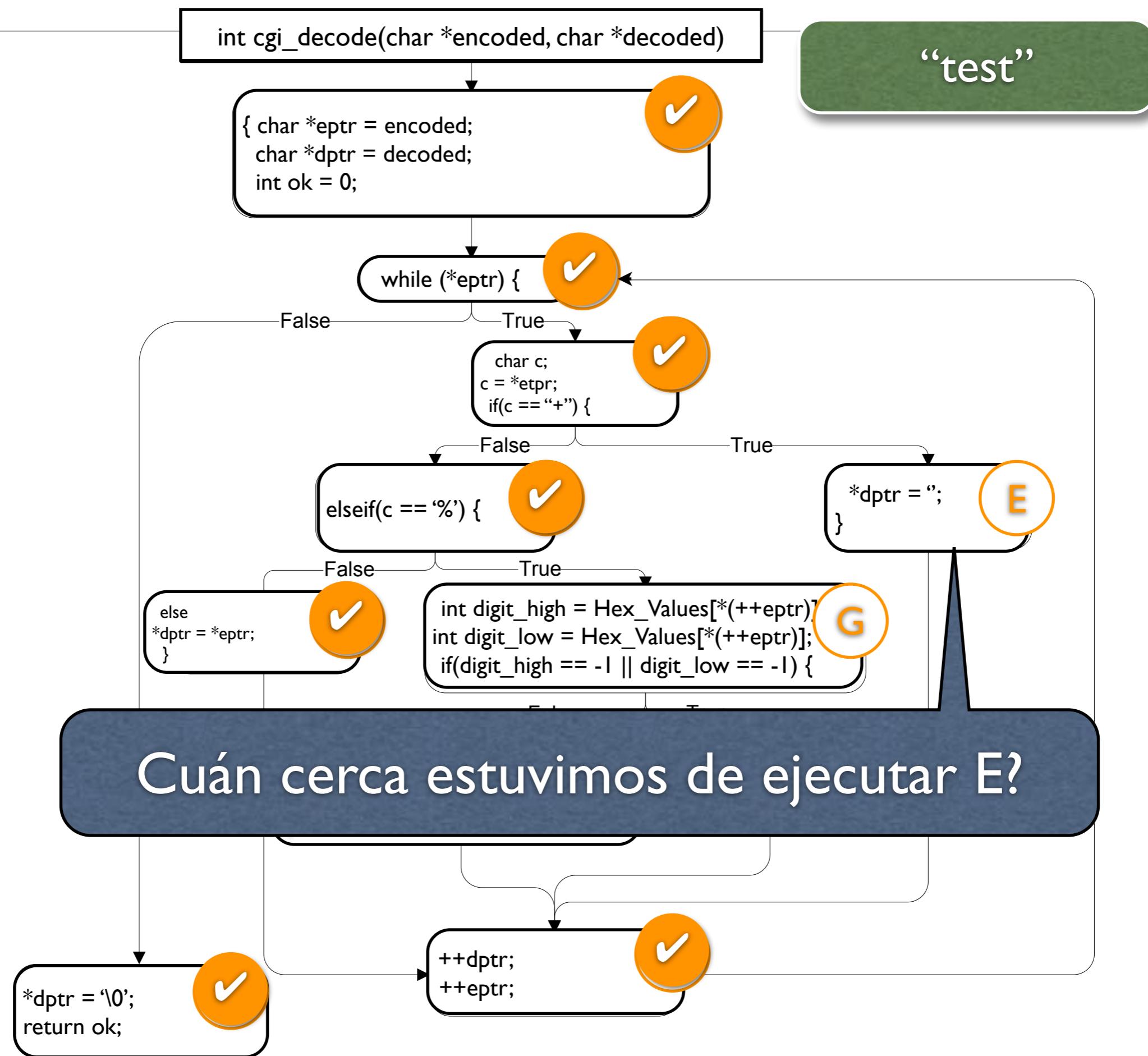
# CFG



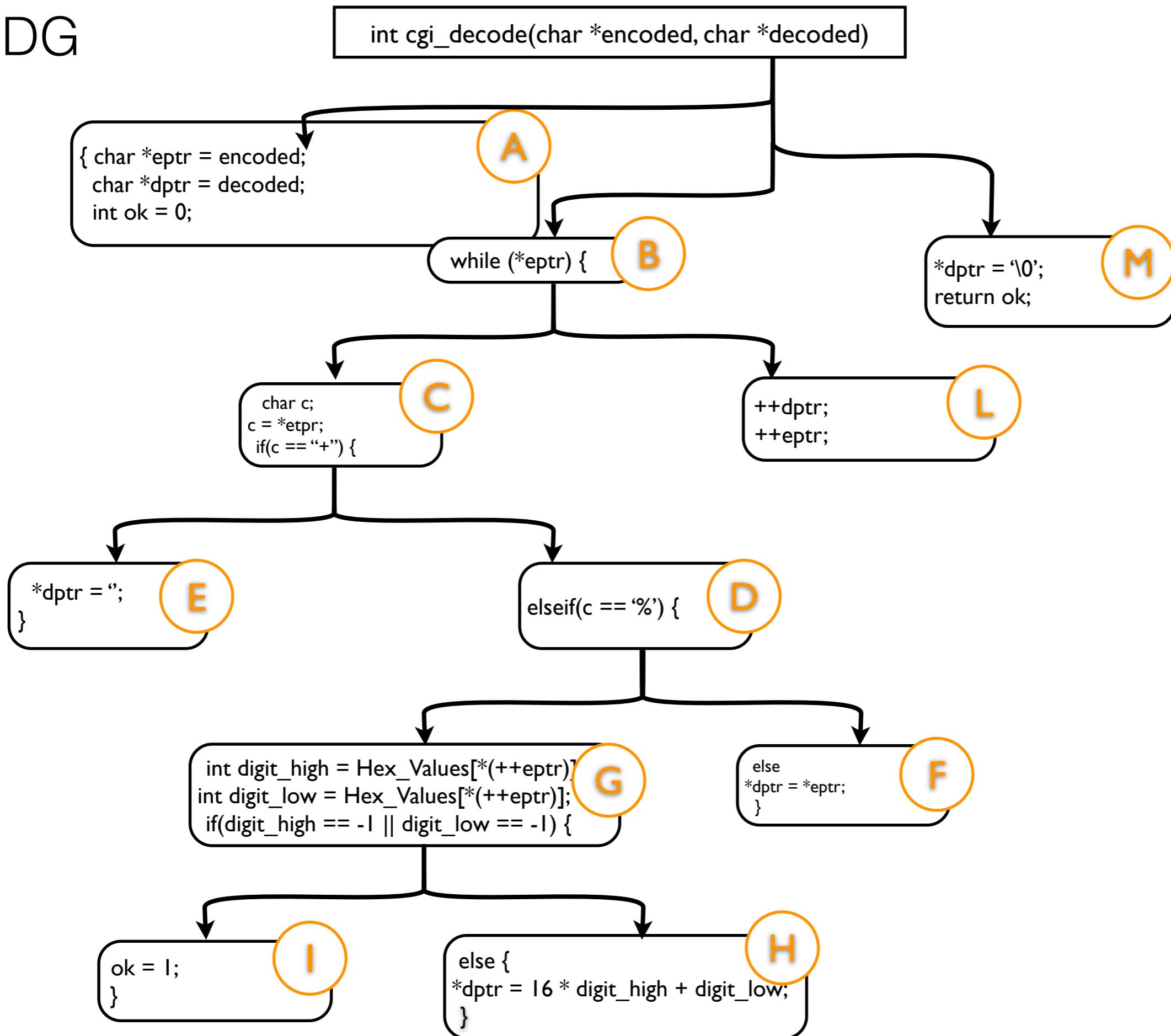
# CFG



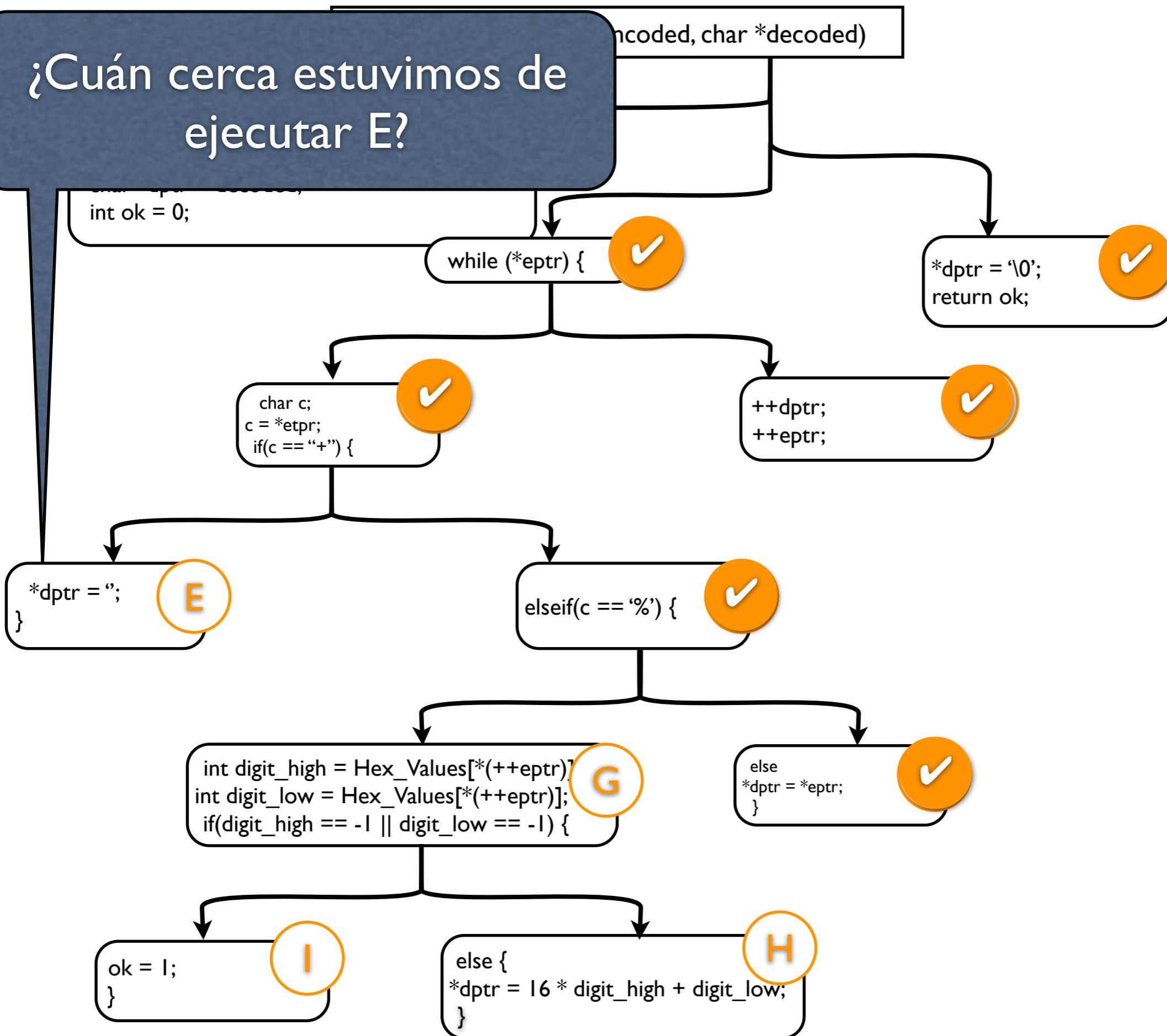
CFG



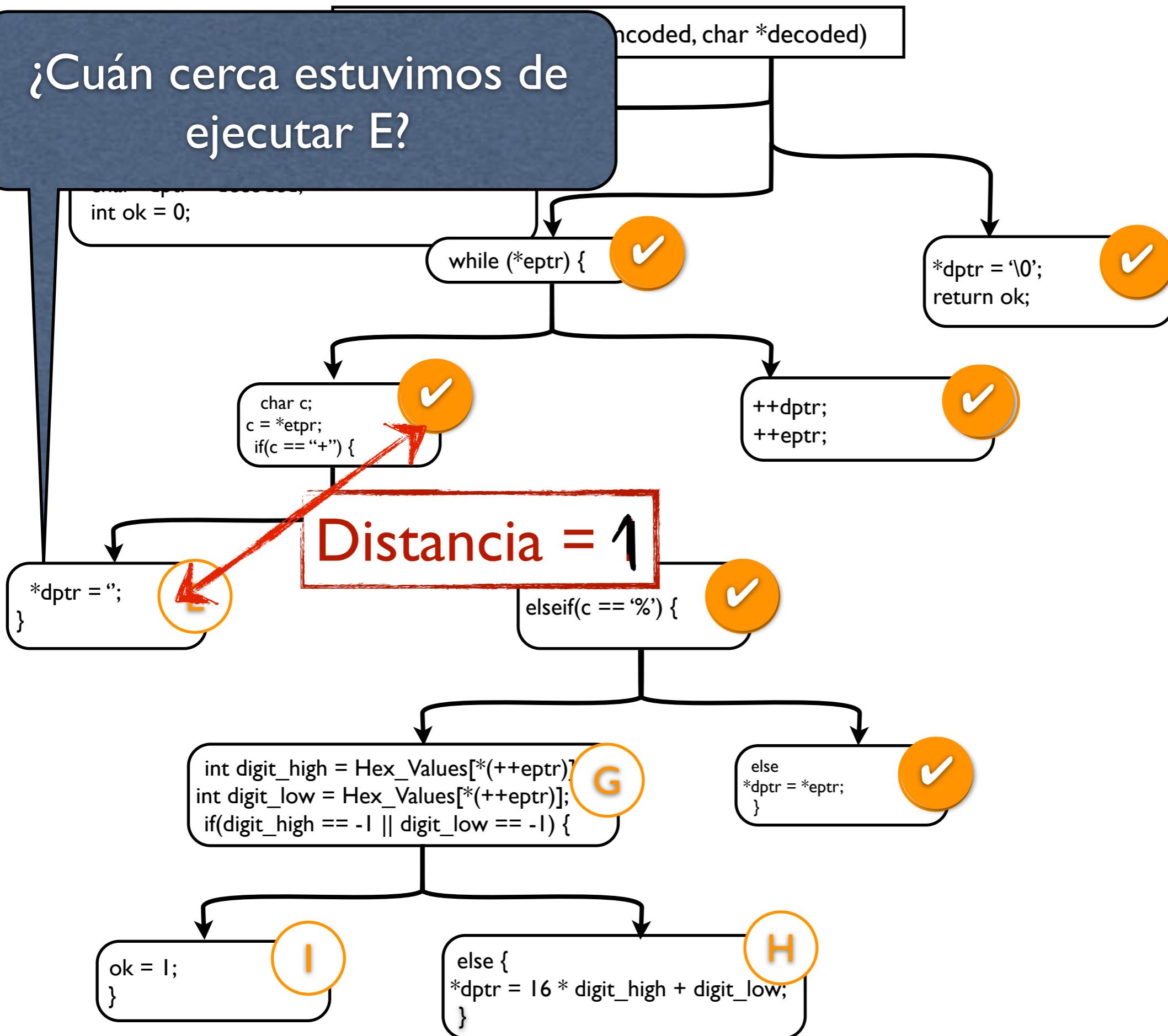
# CDG



# ¿Cuán cerca estuvimos de ejecutar E?



¿Cuán cerca estuvimos de ejecutar E?



# Fitness Function

- Dependent=número de nodos control-dependientes hasta el objetivo (i.e. los ancestros del target)
- Executed=número de nodos control-dependientes (ancestros) cubiertos durante la ejecución del individuo
- $FF = (\text{Dependent} - \text{Executed})$

*cellos en ejecución*

*↓*

*nodos control dependientes*

*fronto blocking*

# Approach Level

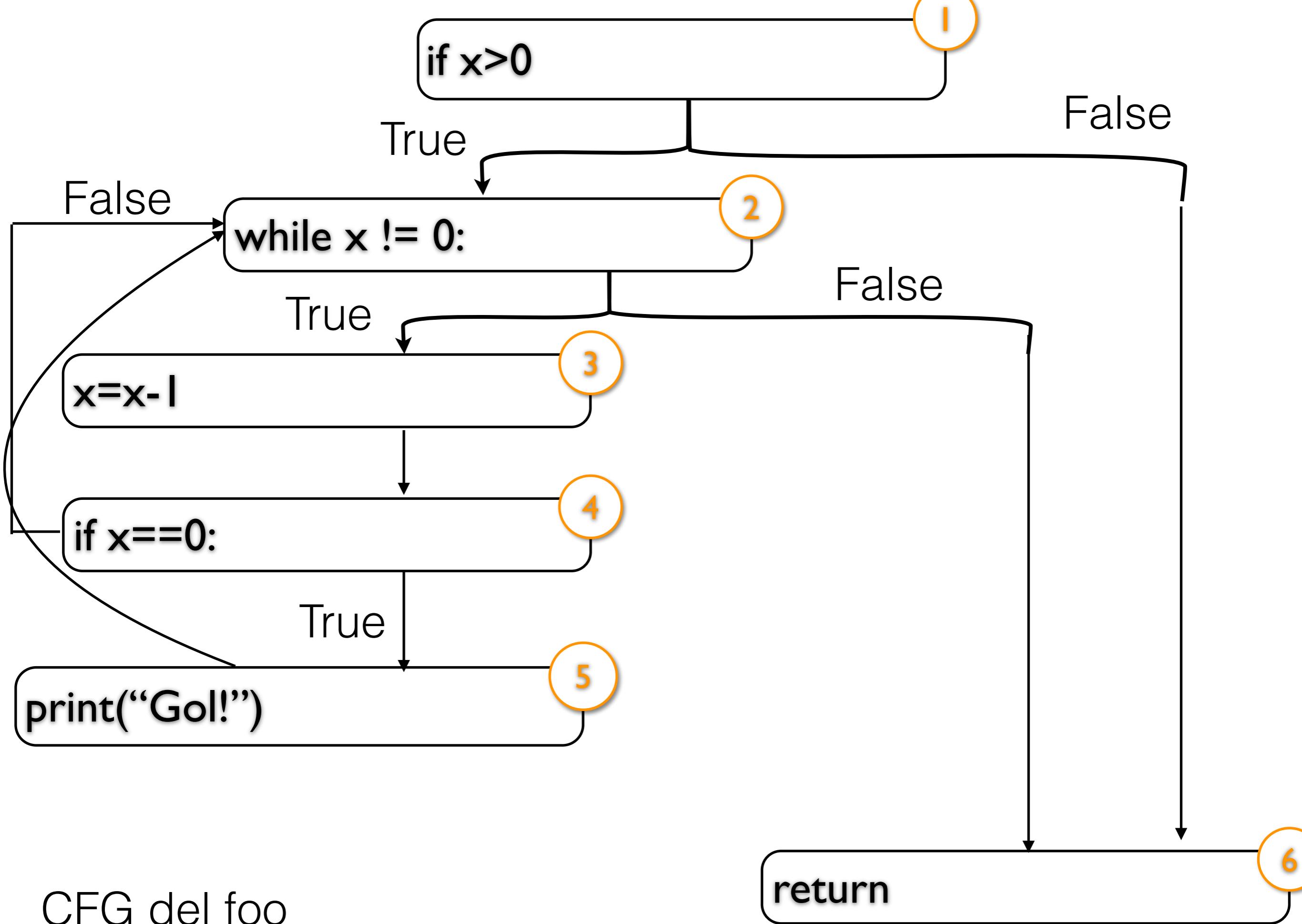


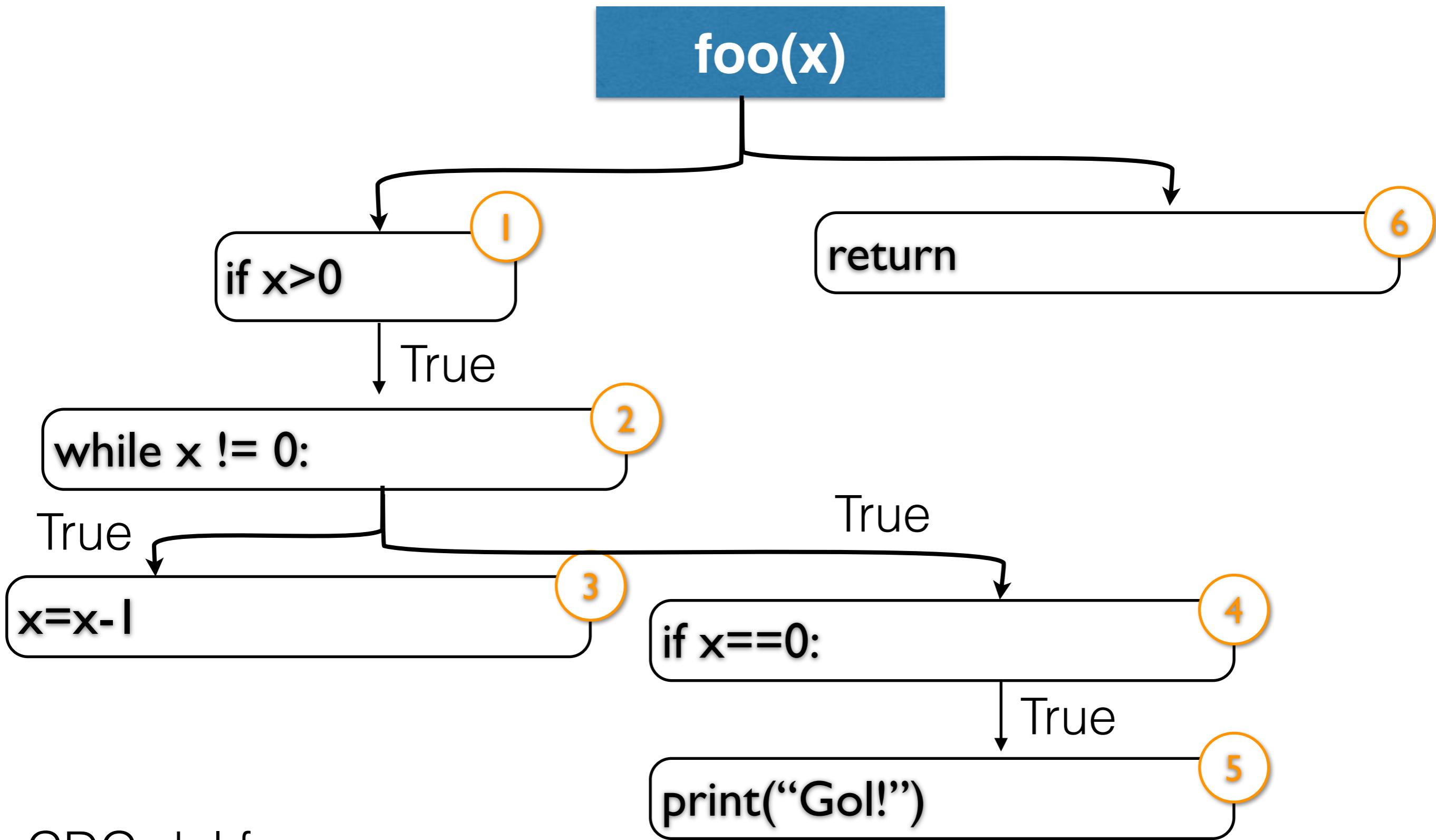
# Fitness Function

- Para **guiar** al algoritmo de búsqueda necesitamos que la fitness function combine:
  - El “approach level” (i.e. cuán cerca estoy del predicado)
  - La “branch distance” (i.e. cuán cerca ejercitar el branch)

# Fitness Function

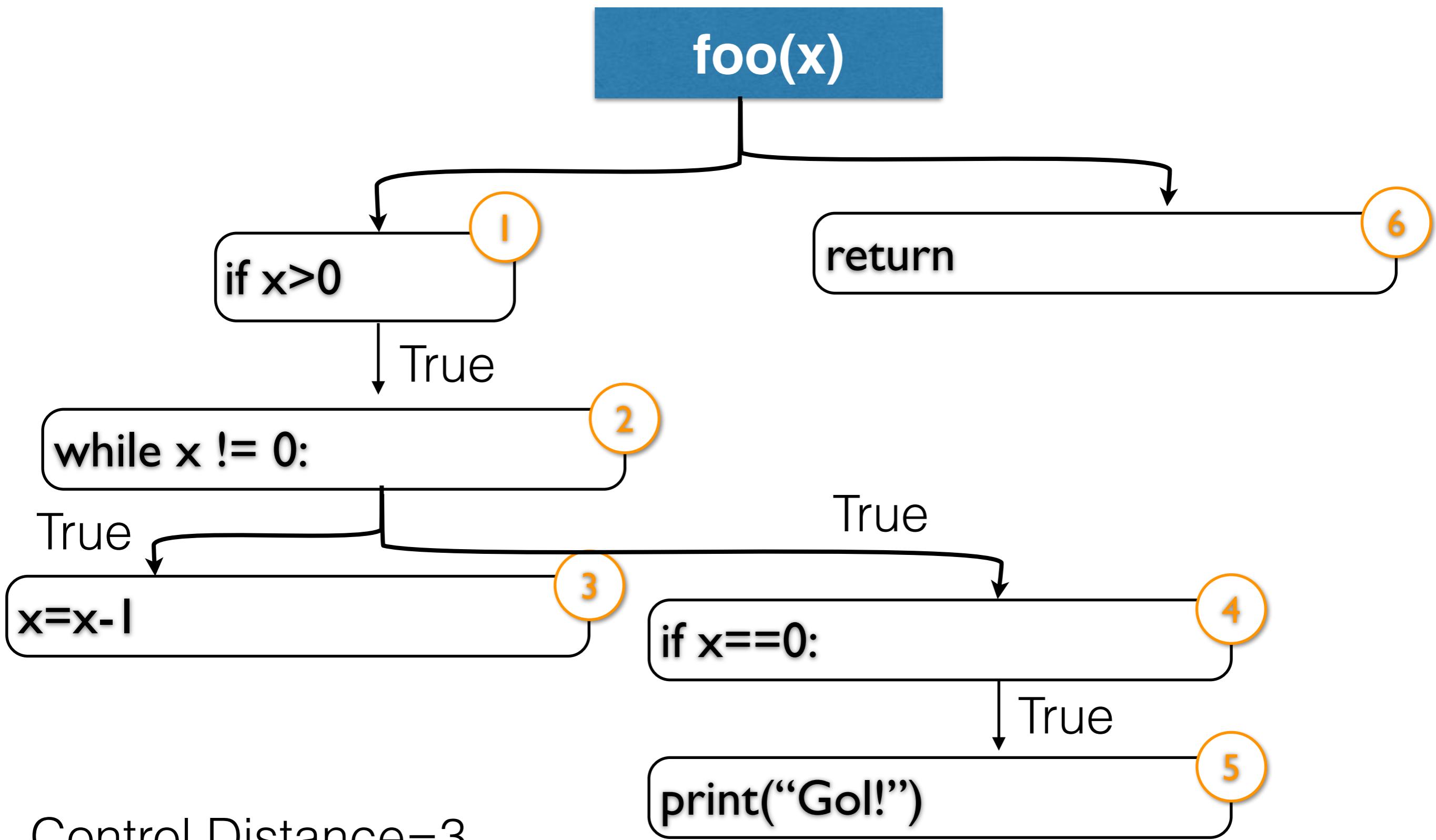
```
def foo(x):
L1: if x>0:
L2:   while x != 0:
L3:     x=x-1
L4:     if x==0:
L5:       print("Gol!")
L6: return
```



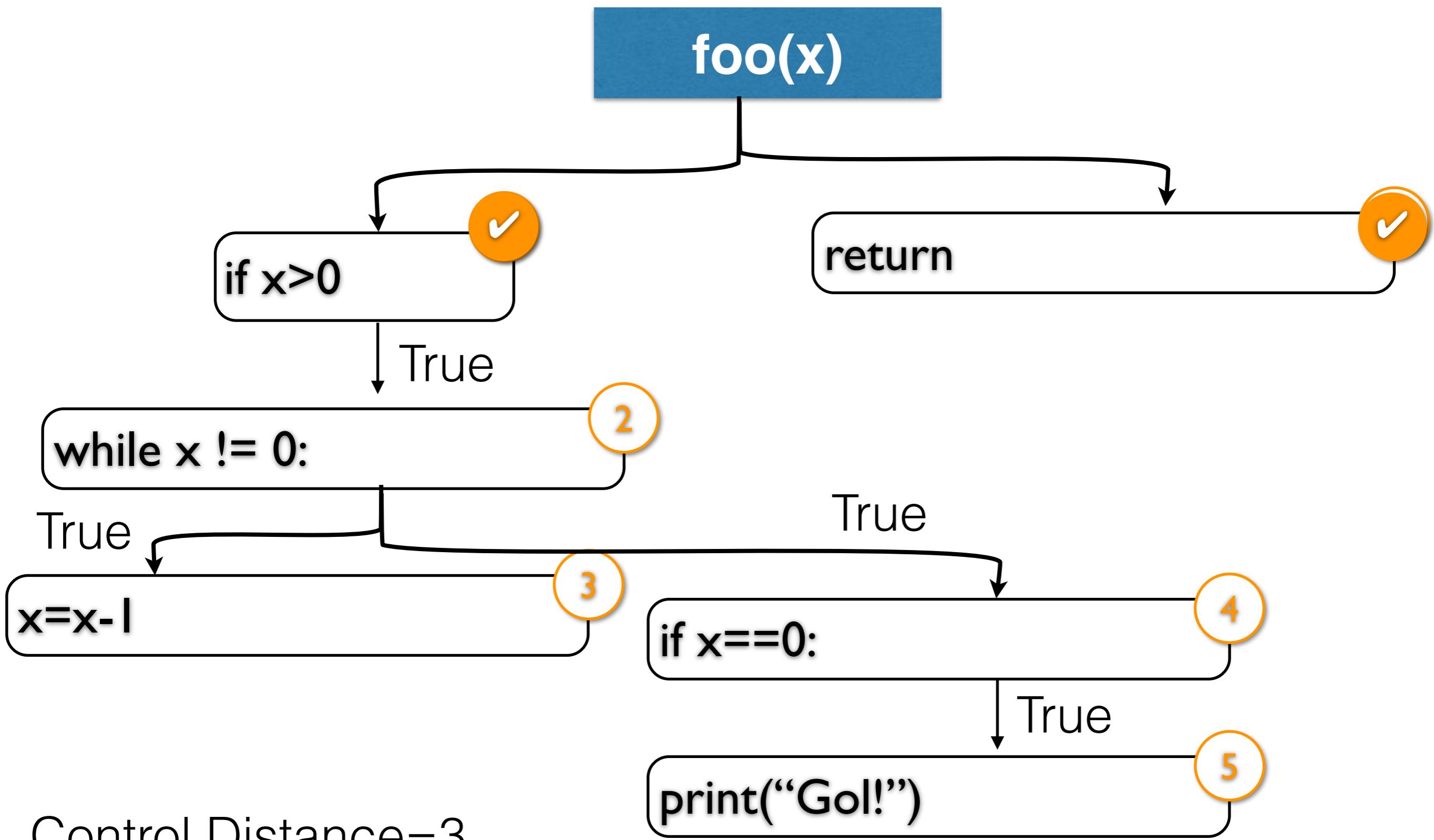


CDG del foo

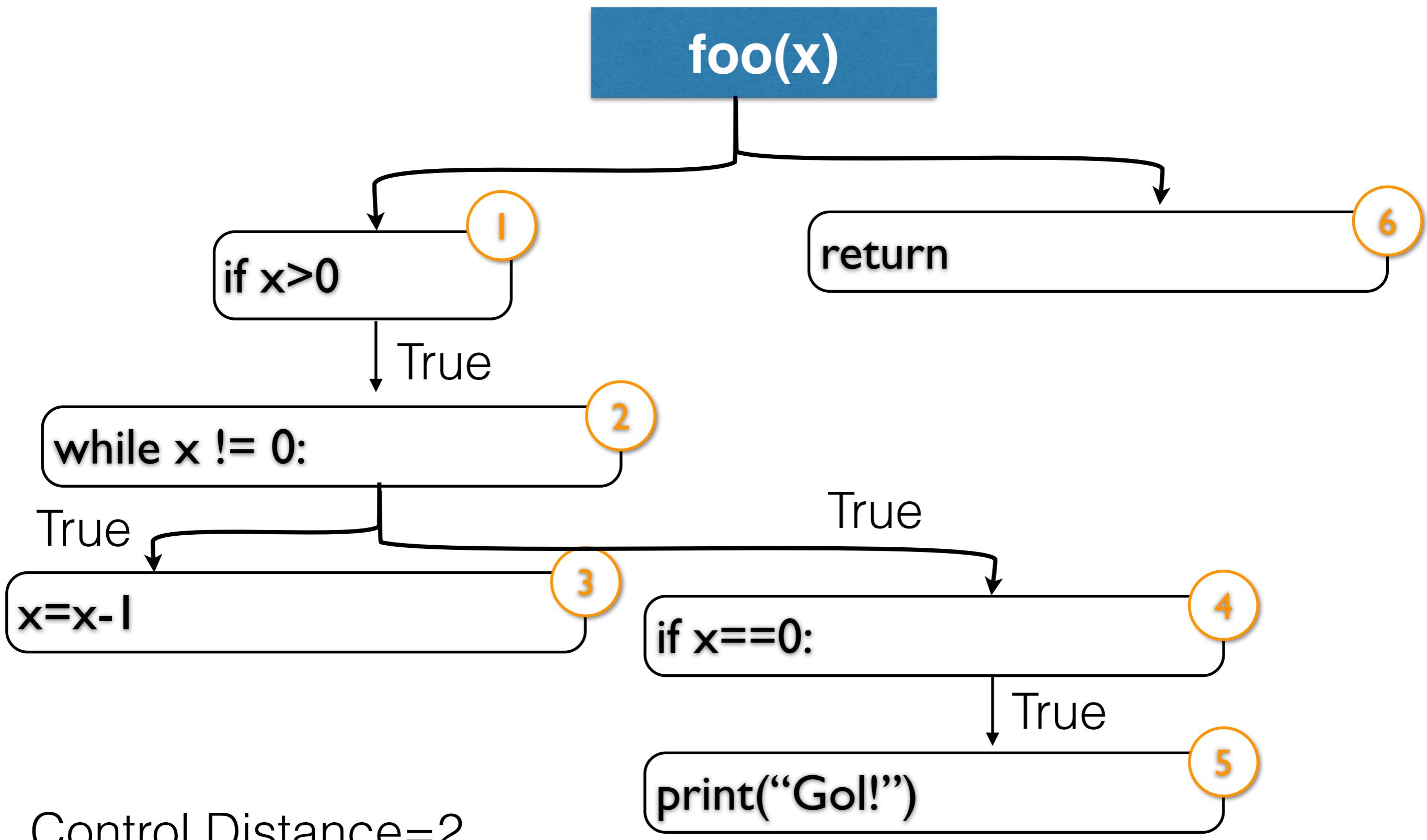
Test1()



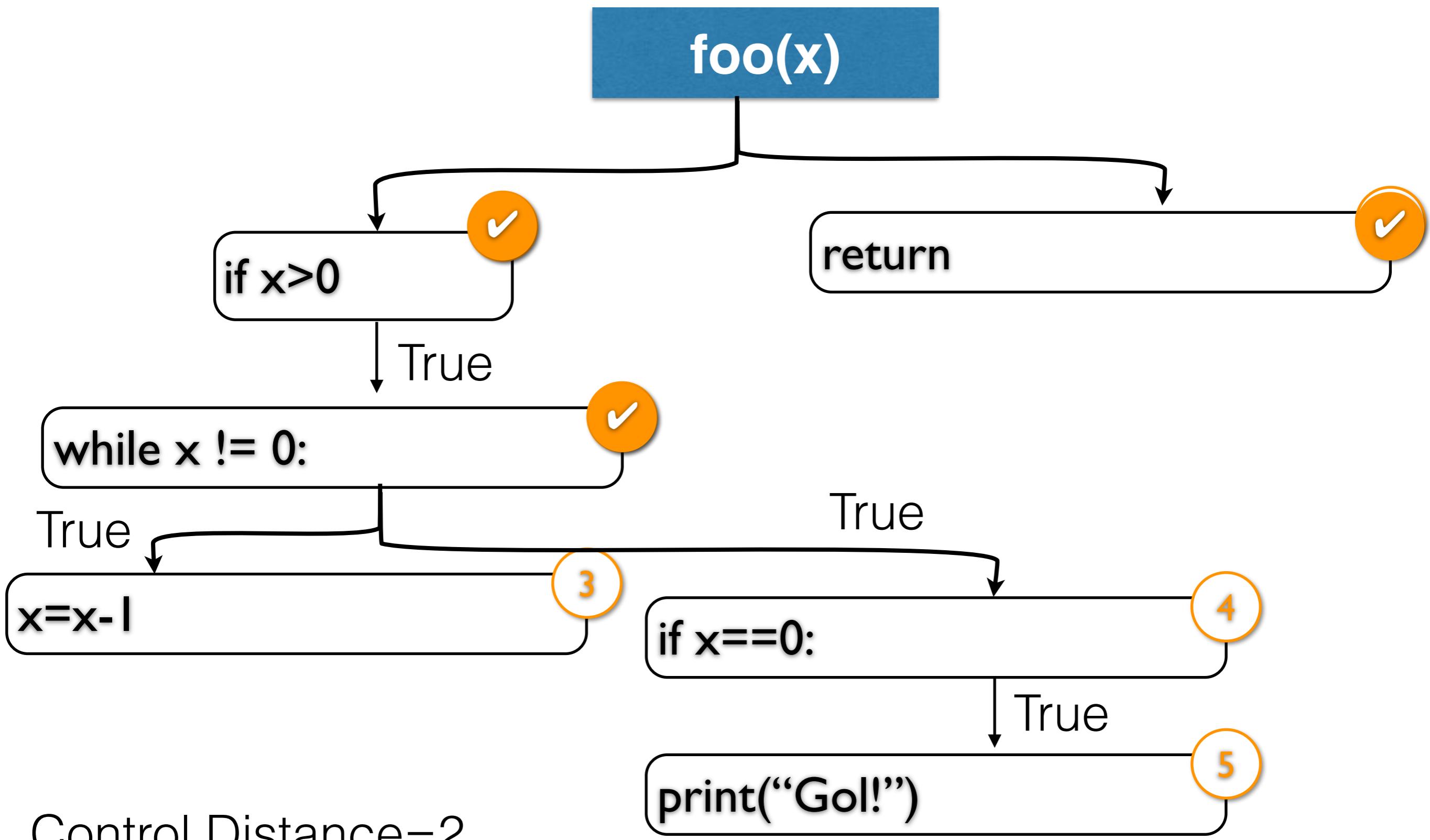
Test1()



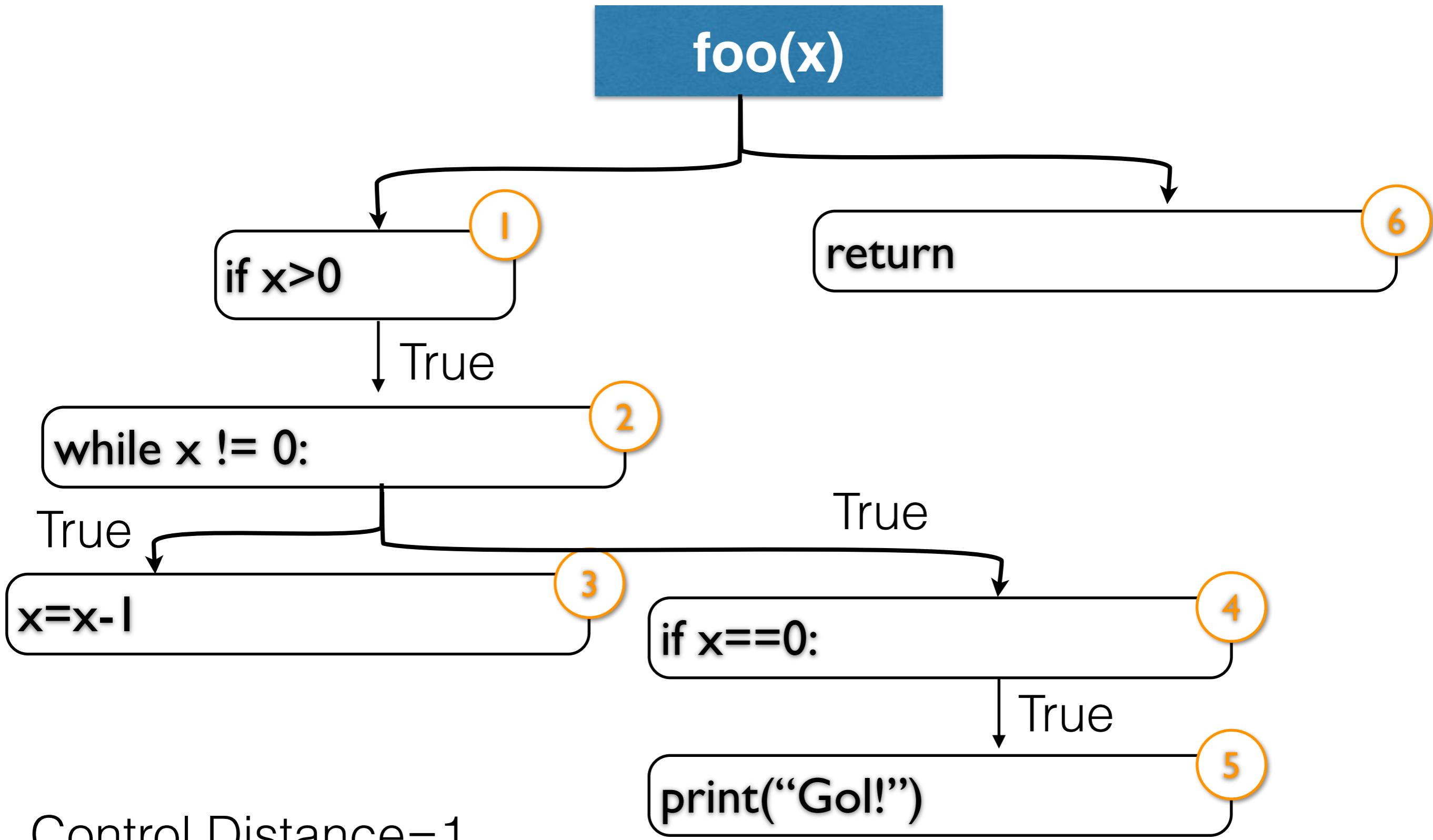
## Test2()



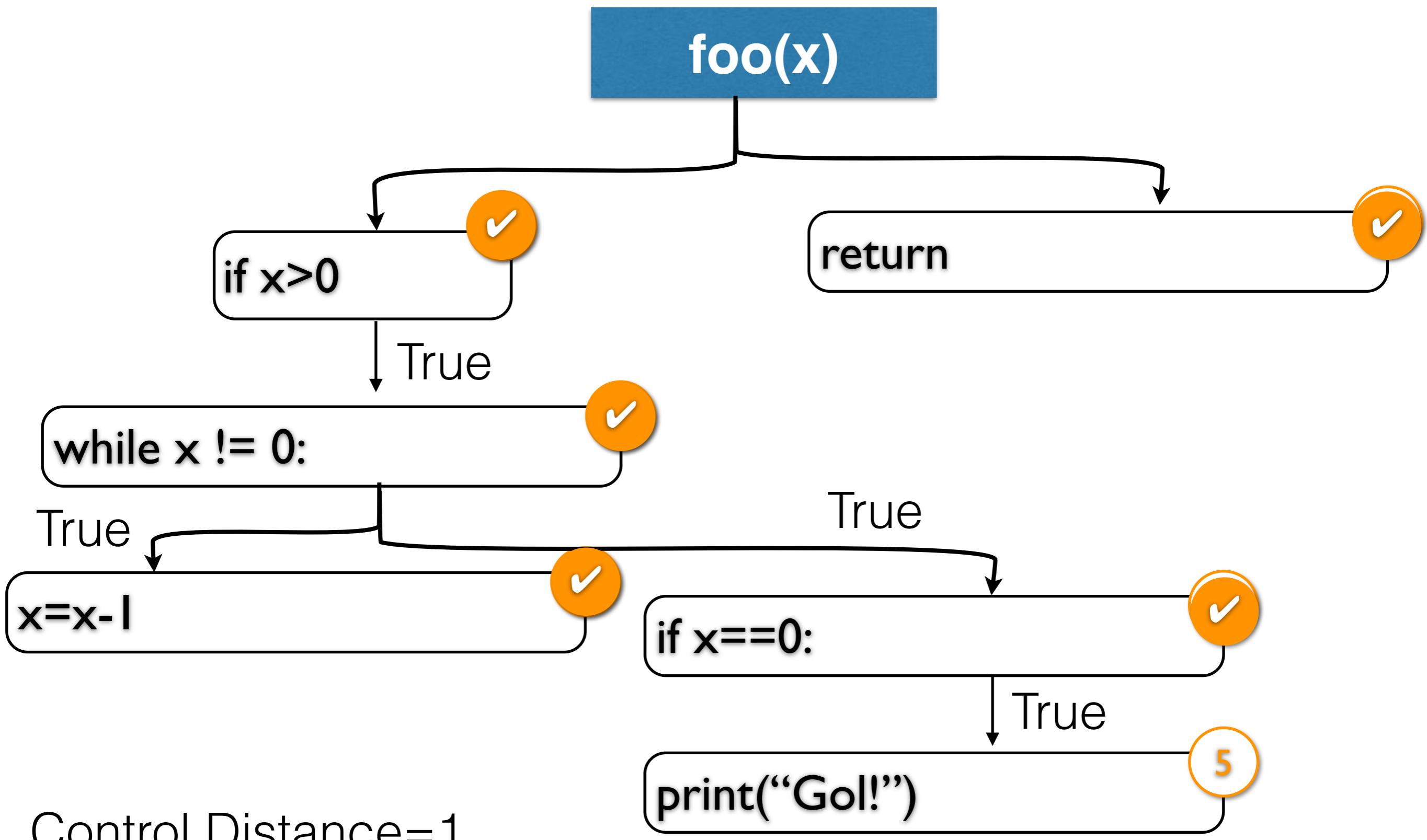
## Test2()



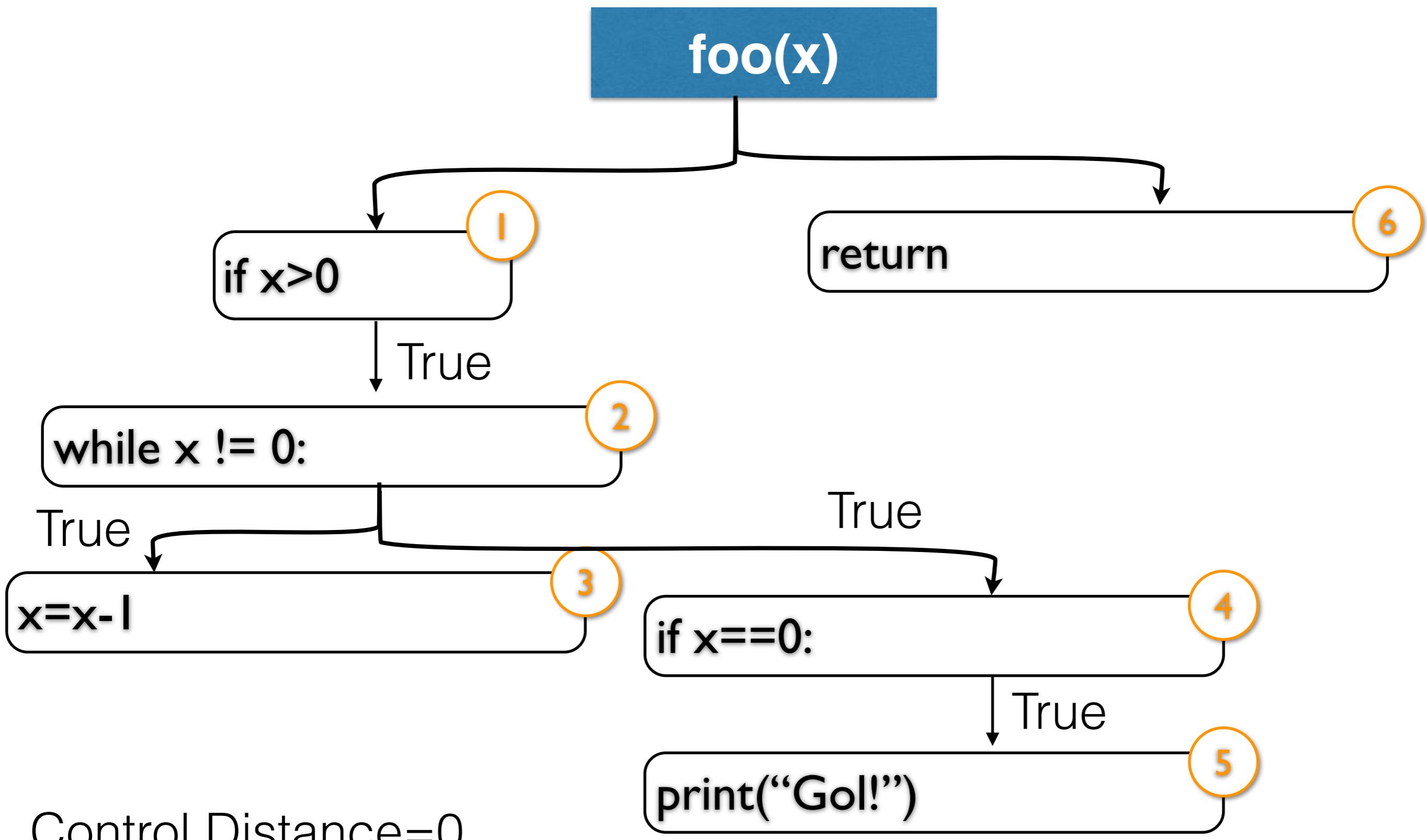
# Test3()



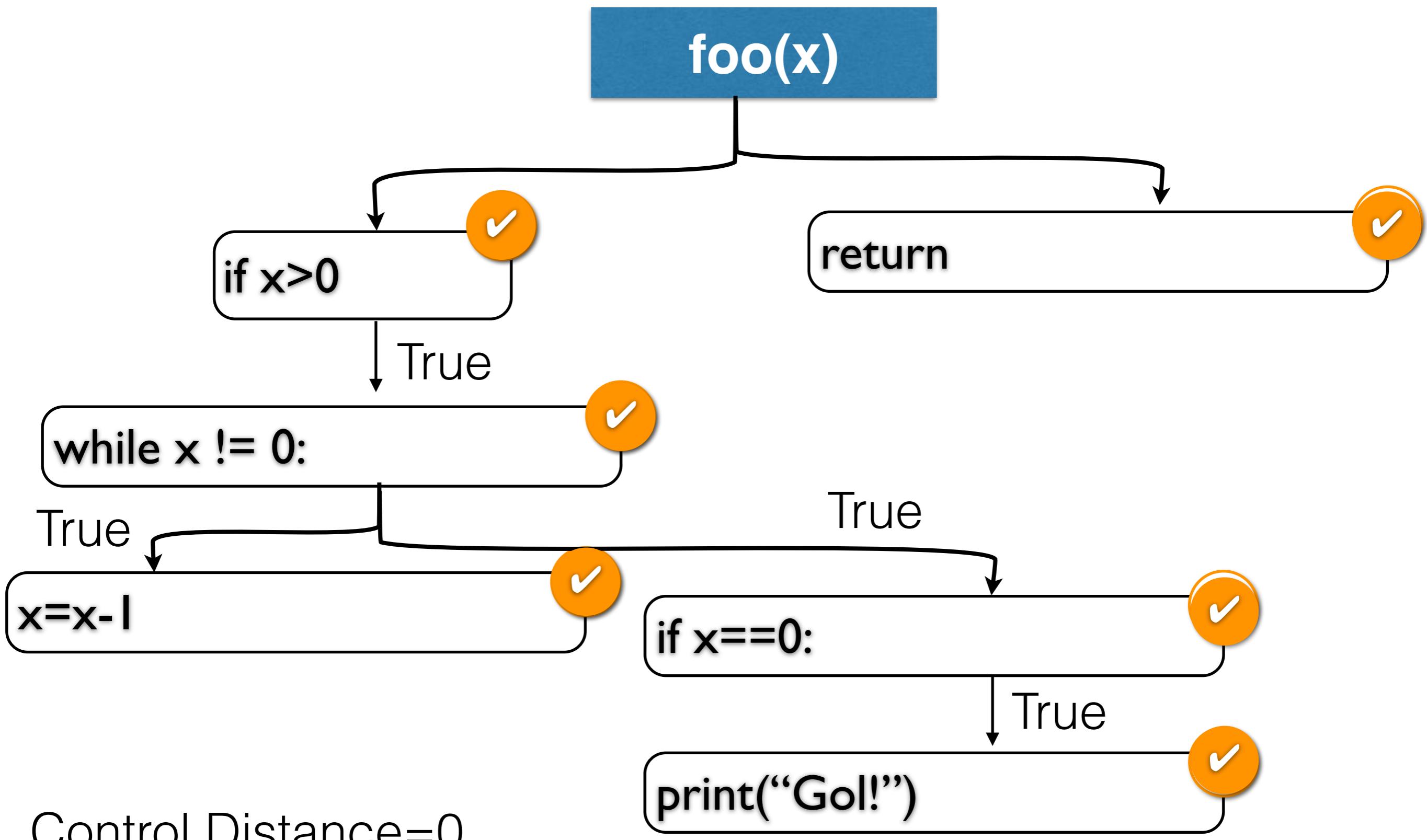
# Test3()



# Test3()



# Test3()

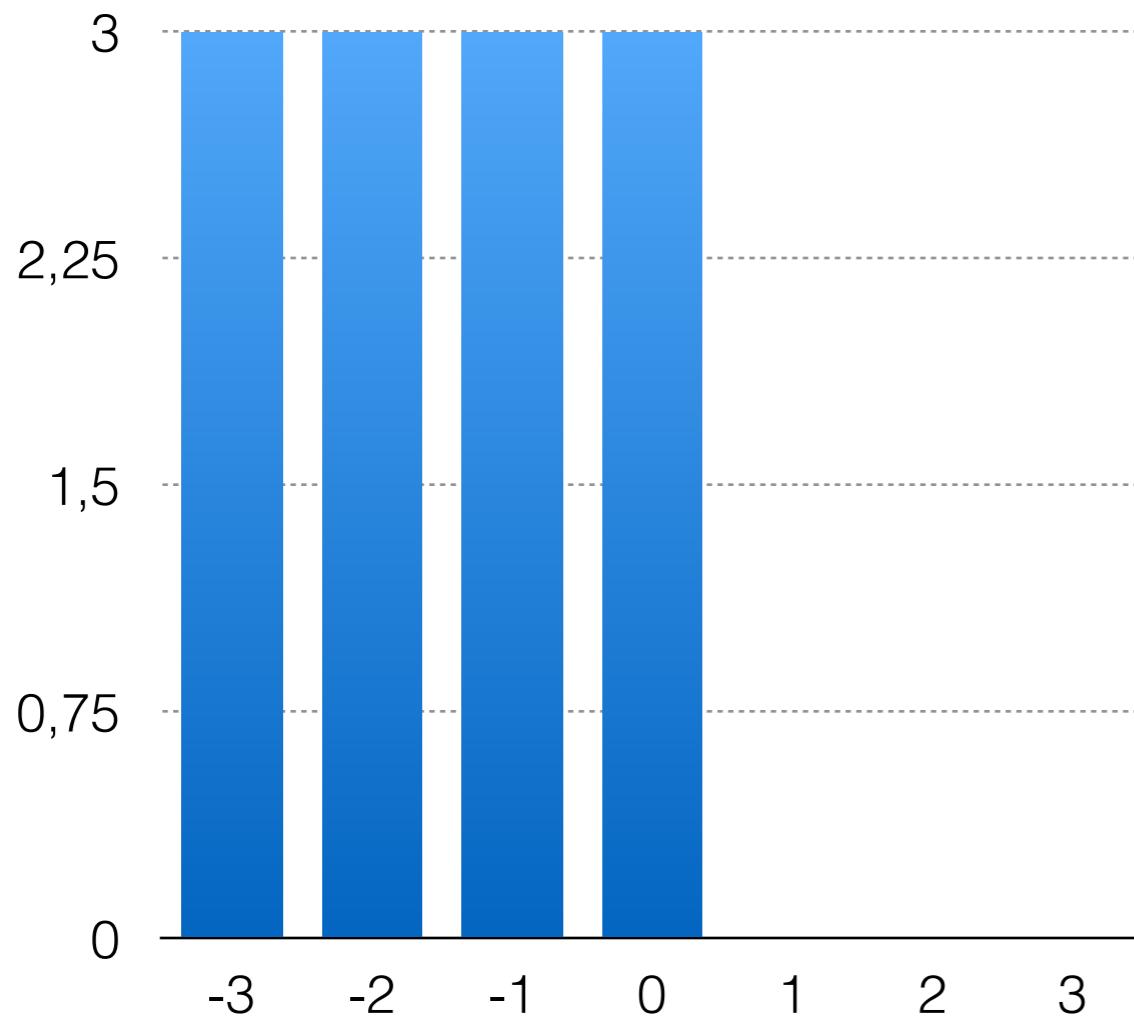


# Control Distance

- Sea  $t$  un camino acíclico desde la raíz del CDG hasta el branch **G**:
  - Computar el “**approach level** hasta G” (i.e. nodos dependientes - nodos ejecutados)
- Fitness Function= el mínimo **approach level** a G usando alguno de los caminos desde la raíz a G
- ¿Cuál será el landscape usando control distance como fitness function?

# Control Distance

- Fitness Function( $x$ ) = Control Distance de ejecutar  $\text{foo}(x)$



- Rugged Fitness Function
- ¿Cómo podemos mejorar la función de fitness para que ayude guiar la búsqueda?

# Control+Branch Distance

- Sea  $t$  un camino acíclico desde la raíz del CDG hasta el branch **G**:
  - Computar el “**approach level** hasta G” (i.e. nodos dependientes - nodos ejecutados)
  - Sumar el **branch distance** del predicado mas cercano alcanzo

# Control+Branch Distance

- Fitness Function= De todos los caminos de la raíz del CDG al branch G, la mínima suma de
  - Approach Level
  - Branch Distance
- ¿Qué pasará si la branch distance es un número muy grande?

# Normalization

- Queremos evitar que la branch distance “**domine**” la función de fitness:
  - Si el último branch es “if  $x==100000$ ” (y además  $x=1$ )
    - Entonces la Branch Distance es 99999
    - Este valor desalienta elegir esta solución

# Normalization Functions

- Nos permiten normalizar todos los valores aunque sean de distinta magnitud
  - Ejemplo:  $v / v+1$
- El rango de normalización es  $[0, 1]$
- La Fitness Function aplica la normalización sobre la Branch Distance

*Calc for fitness function*

# Control+ Normalized Branch Distance

- Sea  $t$  un camino acíclico desde la raíz del CDG hasta el branch **G**:
  - Computar el “**approach level** hasta G” (i.e. nodos dependientes - nodos ejecutados)
  - Sumar la normalización de la **branch distance** del predicado mas cercano alcanzo
- De todos los caminos de la raíz del CDG al branch G, la mínima suma de approach level y branch distance normalizada.

# Control+Normalized Branch Distance

Estoy a una distancia  
de 0,75 de cubrir  
el branch divergente

Fitness  
 $\text{Funcition}(x)=5,75$



Quedan 5 Nodos  
Sin Alcanzar

Goal G

# Control+Normalized Branch Distance

Estoy a una distancia  
de 0,05 de cubrir  
el branch divergente

Fitness  
 $\text{Funcition}(x)=5,05$



Quedan 5 Nodos  
Sin Alcanzar

Goal G

# Control+Normalized Branch Distance

Estoy a una distancia  
de 0,05 de cubrir  
el branch divergente

Fitness  
 $\text{Funcition}(x)=1,05$



Goal G

Quedan solo 1 Nodo  
Sin Alcanzar

Todo élo nivé pluedit  
fitness en cas complexe de  
condiciones aliadas.

# Control+Normalized Branch Distance

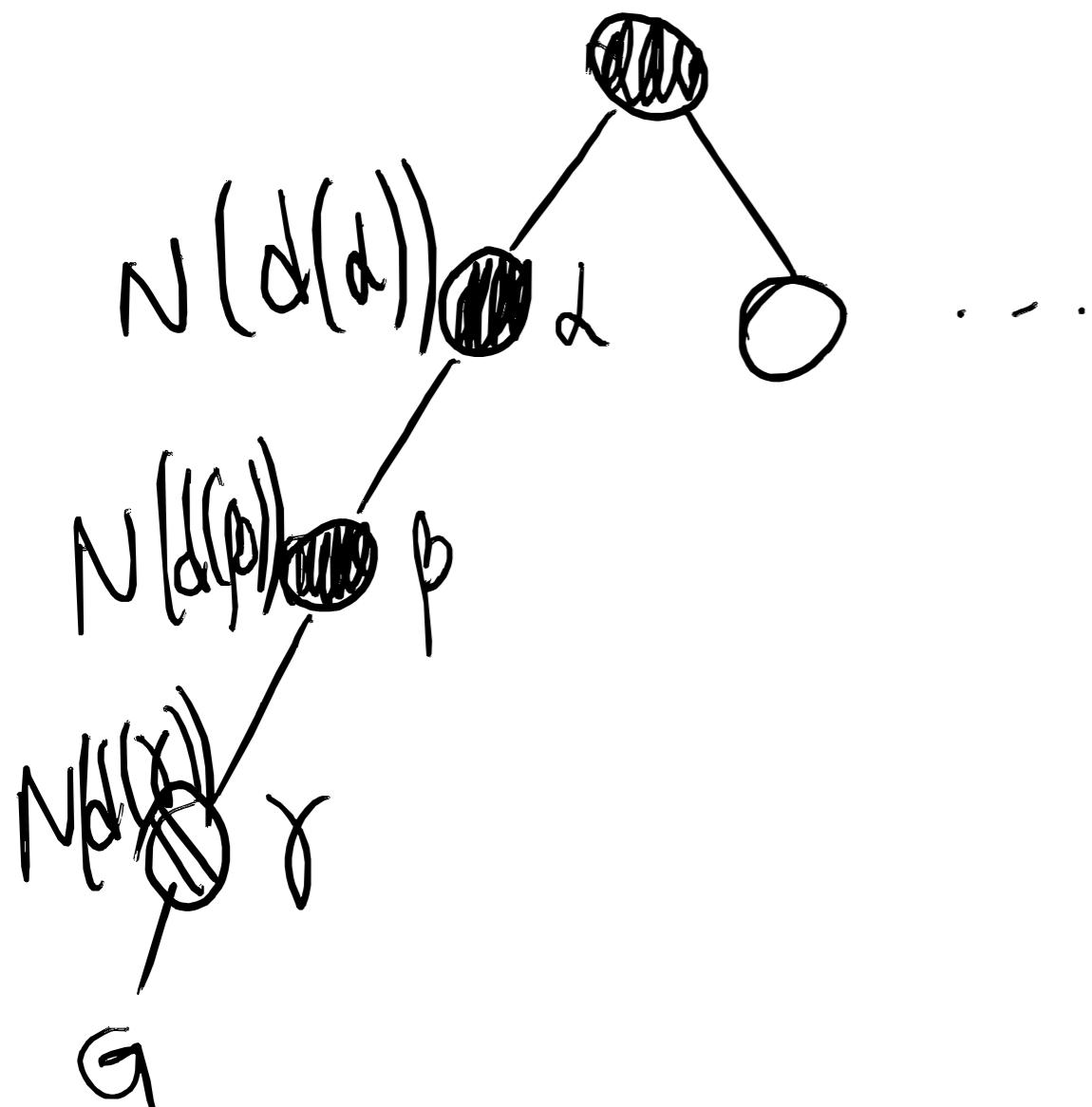
Estoy a una distancia  
de 0,05 de cubrir  
el goal

Fitness  
 $\text{Funciton}(x)=0,05$

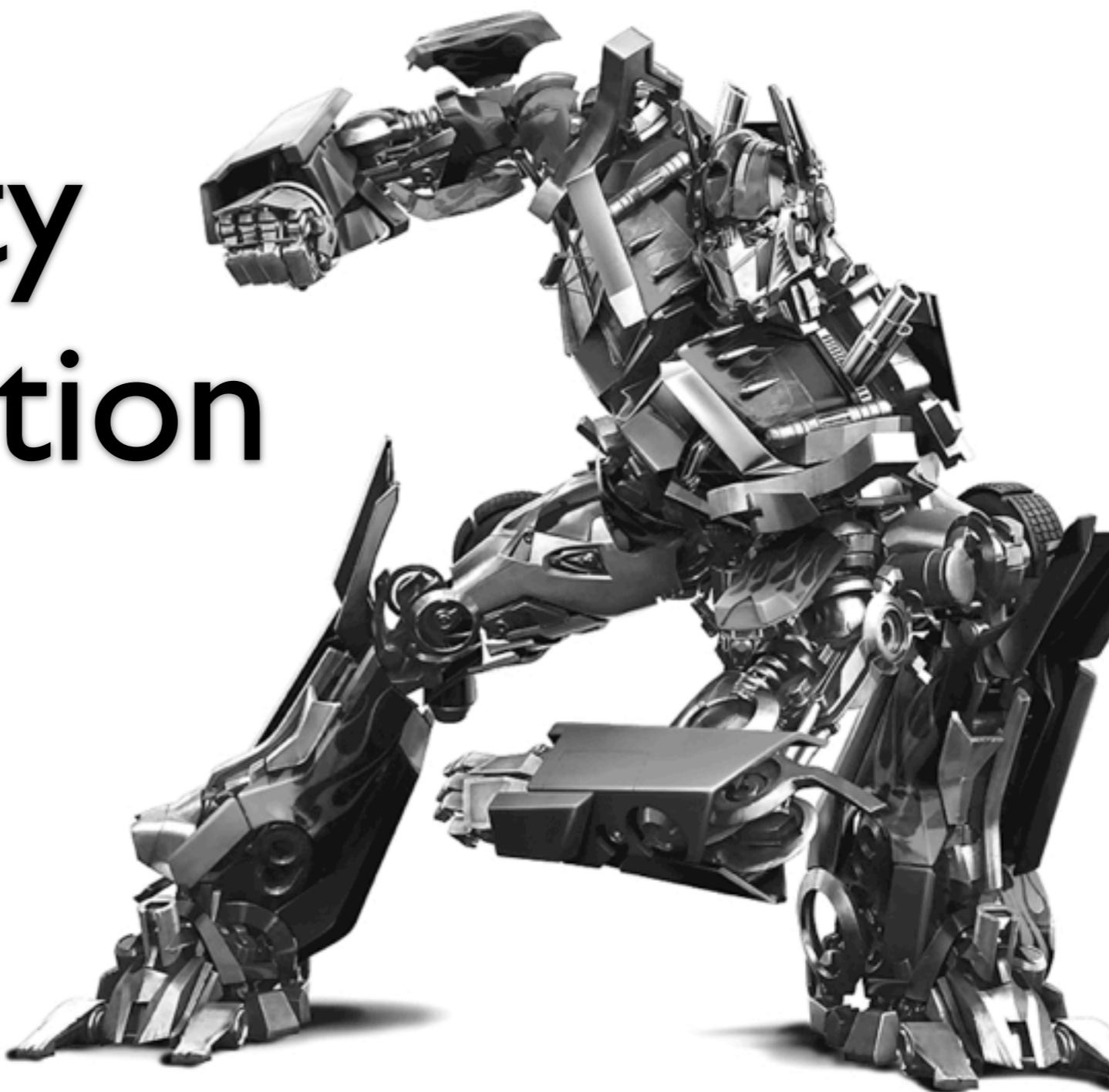


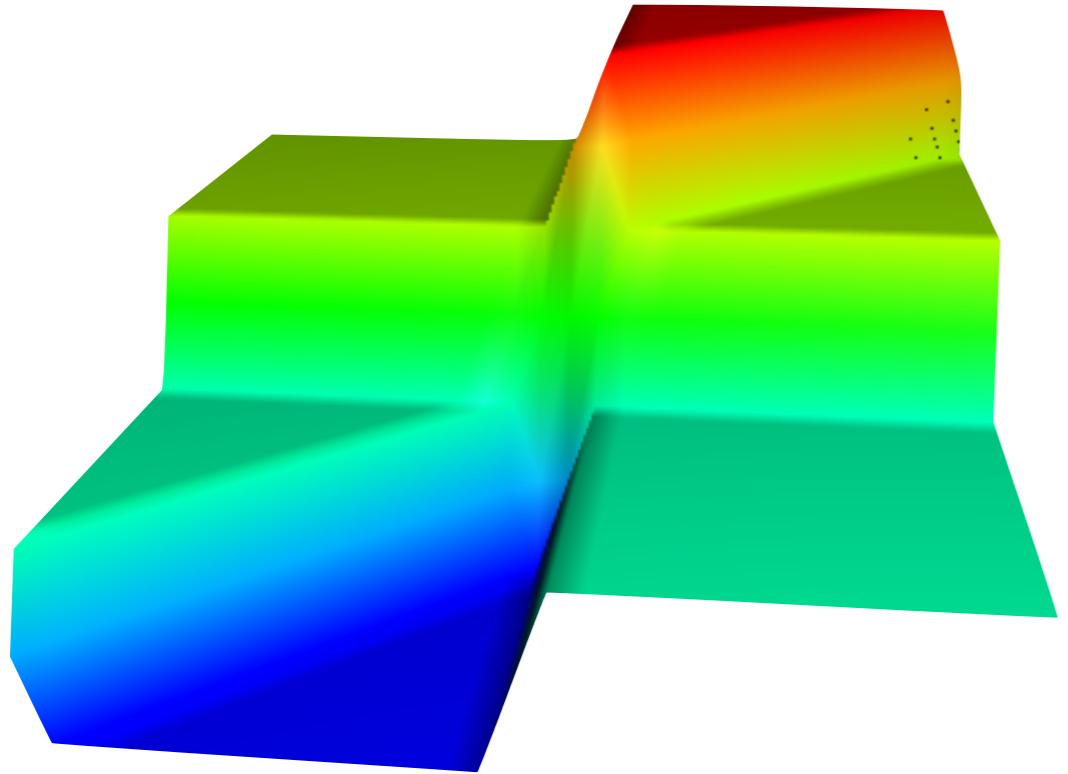
Goal G

Todos los nodos  
han sido alcanzados

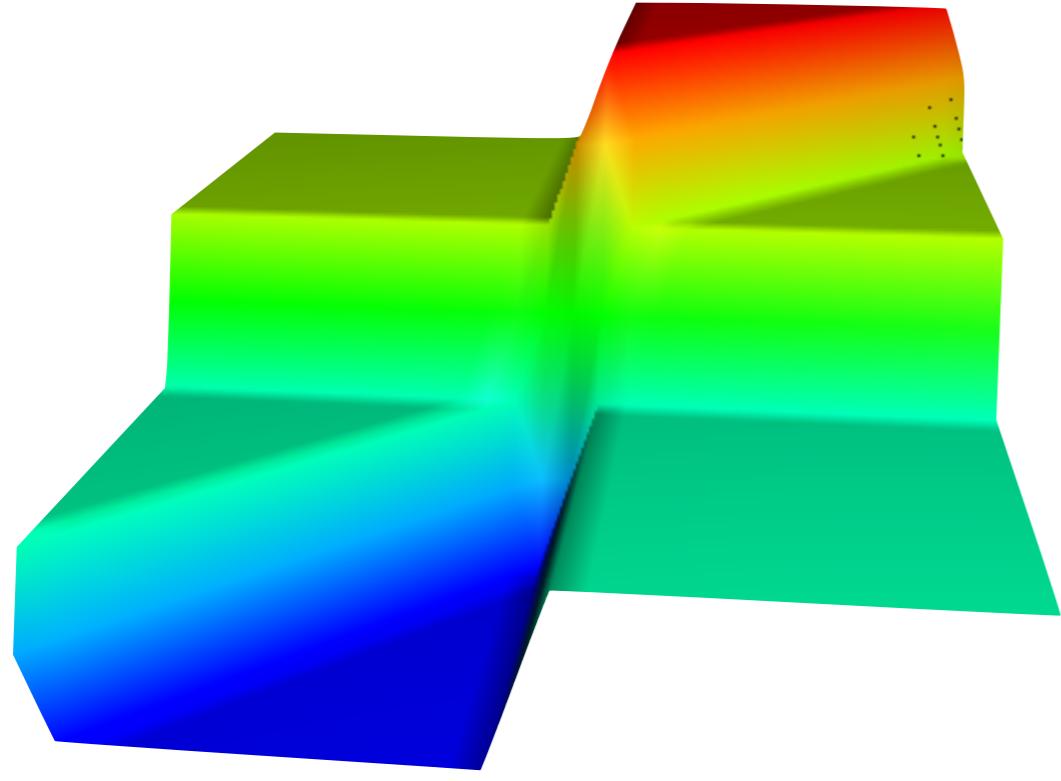


# Testability Transformation

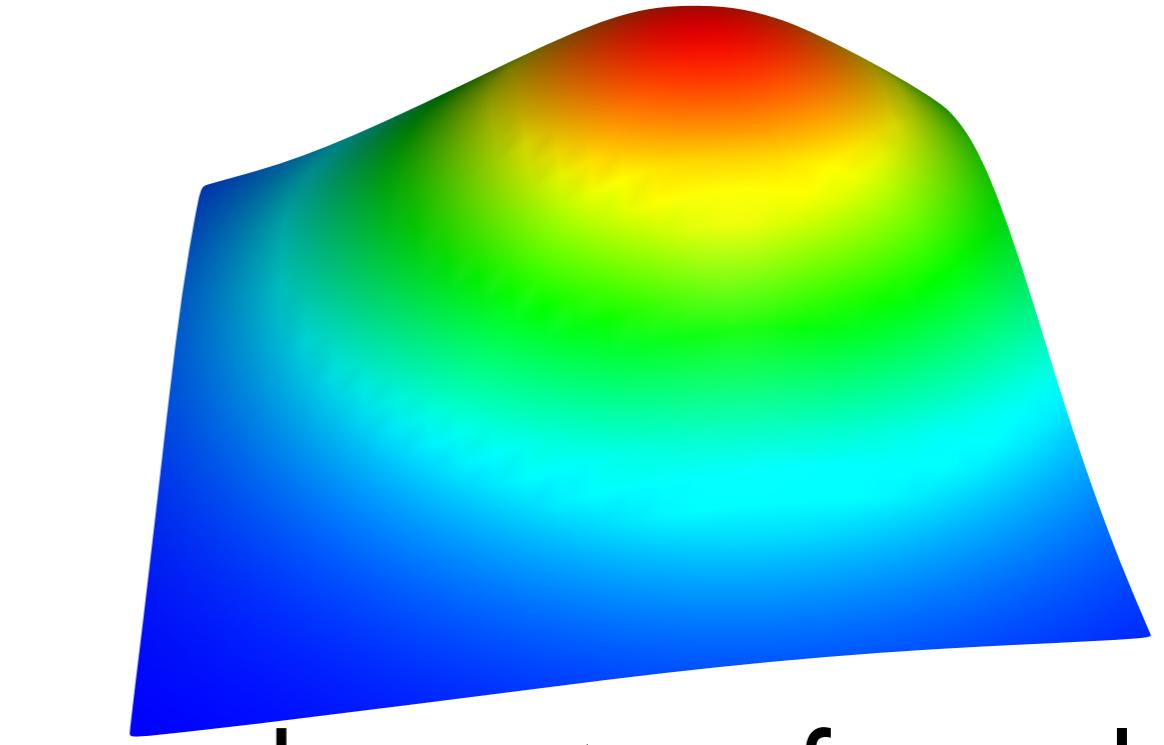




Si un programa es arduo  
para la generación de tests...

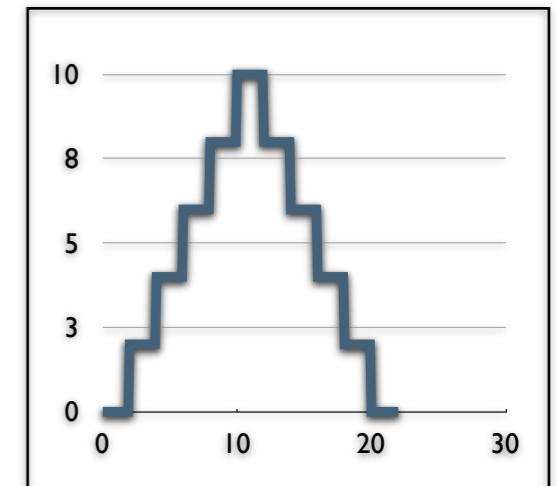
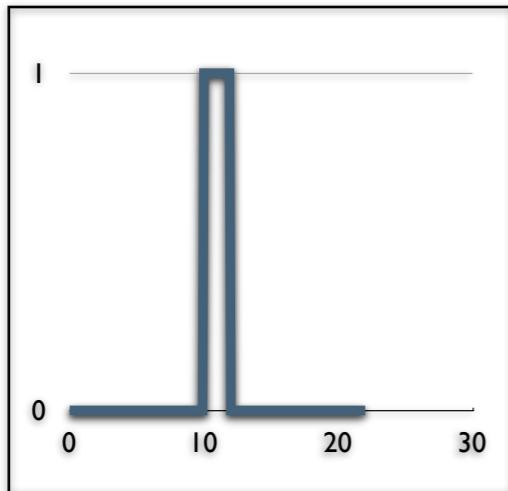


Si un programa es arduo  
para la generación de tests...



...podemos transformarlo a  
una versión más amigable  
para la generación basada en  
búsqueda

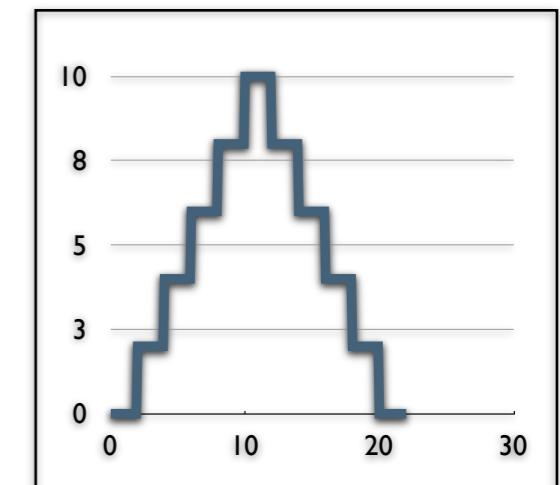
# Testability Transformation



1. Generar tests
2. Descartar programa transformado

# Testability Transformation

```
24 Dim lcount      As Long
25 Dim sChar        As String
26 Dim sPrevChar   As String
27
28 ' Starts with Rem it is a comment
29 sline = Trim(sline)
30 If Left(sline, 3) = "Rem" Then
31     CleanUpLine = ""
32     Exit Function
33 End If
34
35 ' Starts with ' it is a comment
36 If Left(sline, 1) = "'" Then
37     CleanUpLine = ""
38     Exit Function
39 End If
40
41 ' Contains ' may end in a comment, so test if it is a comment or in the
42 ' body of a string
43 If InStr(sline, "'") > 0 Then
44     sPrevChar = "'"
45     lQuoteCount = 0
46
47 For lcount = 1 To Len(sline)
48     sChar = Mid(sline, lcount, 1)
49
50     ' If we found " " then an even number of " characters in front
51     ' means it is the start of a comment, and odd number means it is
52     ' part of a string
53     If sChar = " " And sPrevChar = " " Then
54         If lQuoteCount Mod 2 = 0 Then
55             sline = Trim(Left(sline, lcount - 1))
56             Exit For
57         End If
58     ElseIf sChar = "'" Then
59         lQuoteCount = lQuoteCount + 1
60     End If
61     sPrevChar = sChar
62 Next lcount
63 End If
```



- I. Generar tests
  2. Descartar programa transformado

# Testability Transformation

```
24 Dim lcount As Long
25 Dim sChar As String
26 Dim sPrevChar As String
27
28 ' Starts with Rem it is a comment
29 sLine = Trim(sLine)
30 If Left(sLine, 3) = "Rem" Then
31     CleanUpLine = ""
32     Exit Function
33 End If
34
35 ' Starts with ' it is a comment
36 If Left(sLine, 1) = "'" Then
37     CleanUpLine = ""
38     Exit Function
39 End If
40
41 ' Contains '' may end in a comment, so test if it is a comment or in the
42 ' body of a string
43 If InStr(sLine, "") > 0 Then
44     sPrevChar = ""
45     iQuoteCount = 0
46
47 For lCount = 1 To Len(sLine)
48     sChar = Mid(sLine, lCount, 1)
49
50     ' If we found " " then an even number of " characters in front
51     ' means it is the start of a comment, and odd number means it is
52     ' part of a string
53     If sChar = " " And sPrevChar = " " Then
54         If iQuoteCount Mod 2 = 0 Then
55             sLine = Trim(Left(sLine, lCount - 1))
56             Exit For
57         End If
58     ElseIf sChar = " " Then
59         iQuoteCount = iQuoteCount + 1
60     End If
61     sPrevChar = sChar
62 Next lCount
63
64 End If
```



```
24 Dim lcount As Long
25 Dim sChar As String
26 Dim sPrevChar As String
27
28 ' Starts with Rem it is a comment
29 sLine = Trim(sLine)
30 If Left(sLine, 3) = "Rem" Then
31     CleanUpLine = ""
32     Exit Function
33 End If
34
35 ' Starts with ' it is a comment
36 If Left(sLine, 1) = "'" Then
37     CleanUpLine = ""
38     Exit Function
39 End If
40
41 ' Contains '' may end in a comment, so test if it is a comment or in the
42 ' body of a string
43 If InStr(sLine, "") > 0 Then
44     sPrevChar = ""
45     iQuoteCount = 0
46
47 For lCount = 1 To Len(sLine)
48     sChar = Mid(sLine, lCount, 1)
49
50     ' If we found " " then an even number of " characters in front
51     ' means it is the start of a comment, and odd number means it is
52     ' part of a string
53     If sChar = " " And sPrevChar = " " Then
54         If iQuoteCount Mod 2 = 0 Then
55             sLine = Trim(Left(sLine, lCount - 1))
56             Exit For
57         End If
58     ElseIf sChar = " " Then
59         iQuoteCount = iQuoteCount + 1
60     End If
61     sPrevChar = sChar
62 Next lCount
63
64 End If
```

1. Generar tests
2. Descartar programa transformado

# Testability Transformation

- Transformar el source code puede requerir co-transformar el criterio de generación
- El programa transformado es solamente usado para generar datos de test, y luego es **descartado**
- La Transformación no necesita preservar el comportamiento/signatura

# "The Flag Problem"

...

if(x == 10)

...

MAX -|x-10|

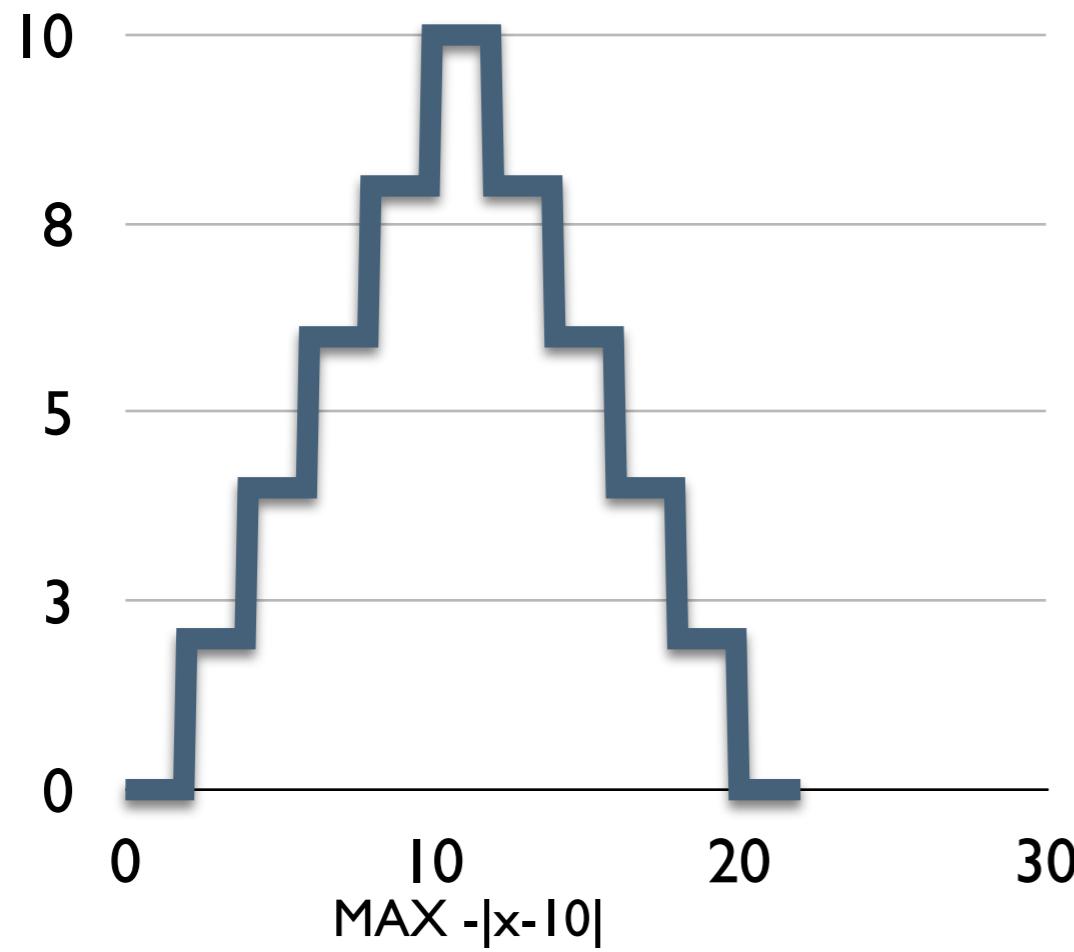
MAX -(1-flag)\*MAX

# "The Flag Problem"

...

if(x == 10)

...



MAX -(1-flag)\*MAX

# "The Flag Problem"

...

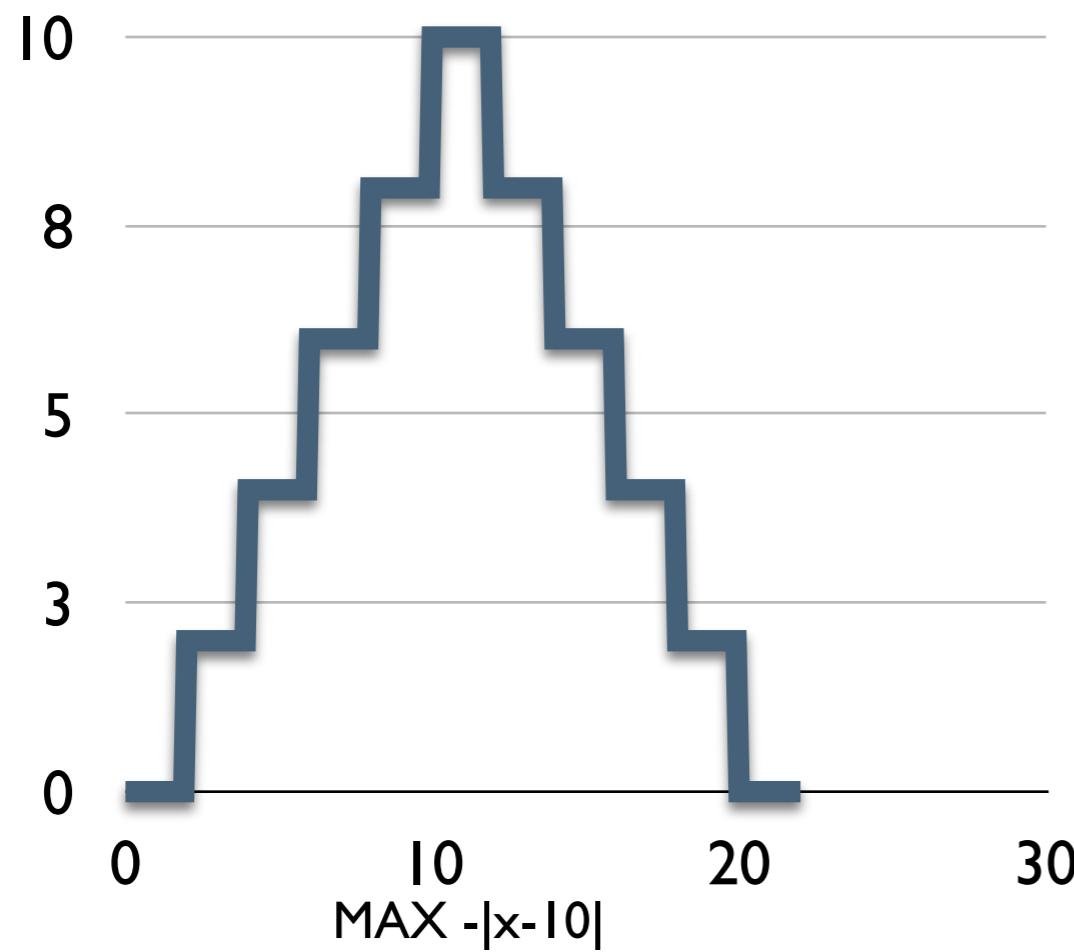
if( $x == 10$ )

...

boolean flag = ( $x == 10$ );

if(flag)

...



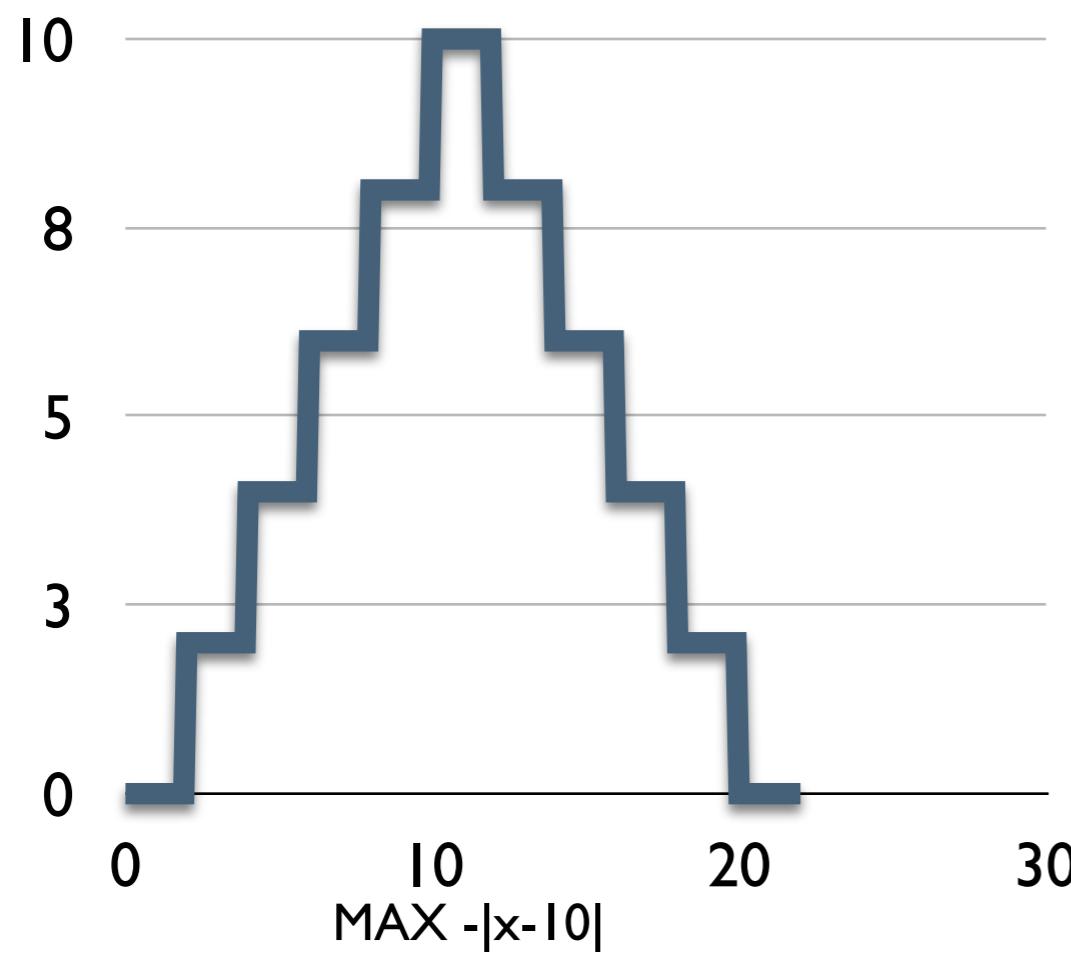
MAX -(1-flag)\*MAX

# "The Flag Problem"

...

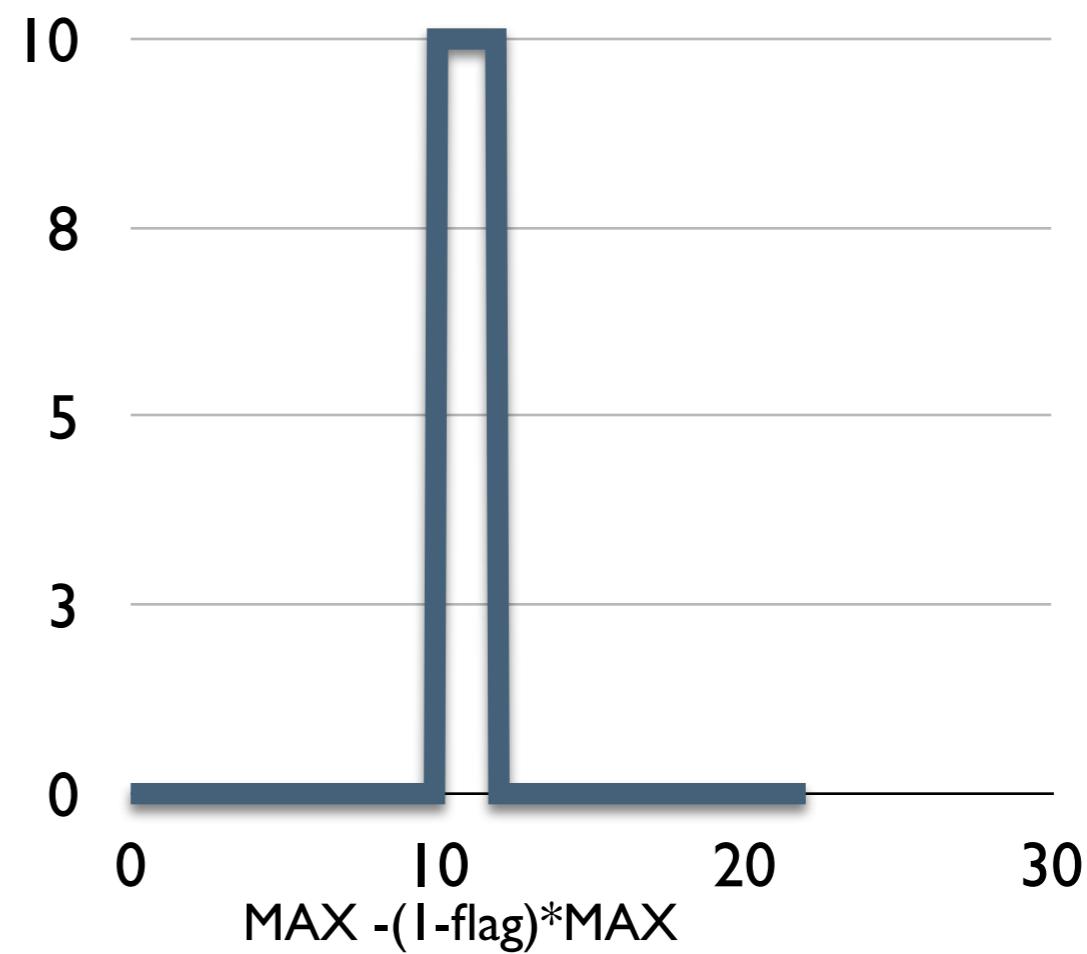
if( $x == 10$ )

...



boolean flag = ( $x == 10$ );  
if(flag)

...



# Flag Level 0

```
if(x == 10) {  
    // ..  
}  
// ..
```

# Flag Level I

```
boolean flag = (x == 10);
// ...
if(flag) {
    // ...
}
// ...
```

# Flag Level I

```
boolean flag = (x == 10);
// ...
if(flag) {
    // ...
}
// ...
```

Ni flag ni x son modificadas antes del predicado, entonces simplemente podemos reemplazar flag con (x == 10) en la if-expression

# Flag Level 2

```
boolean flag = (x >= 10) && (y < z);
```

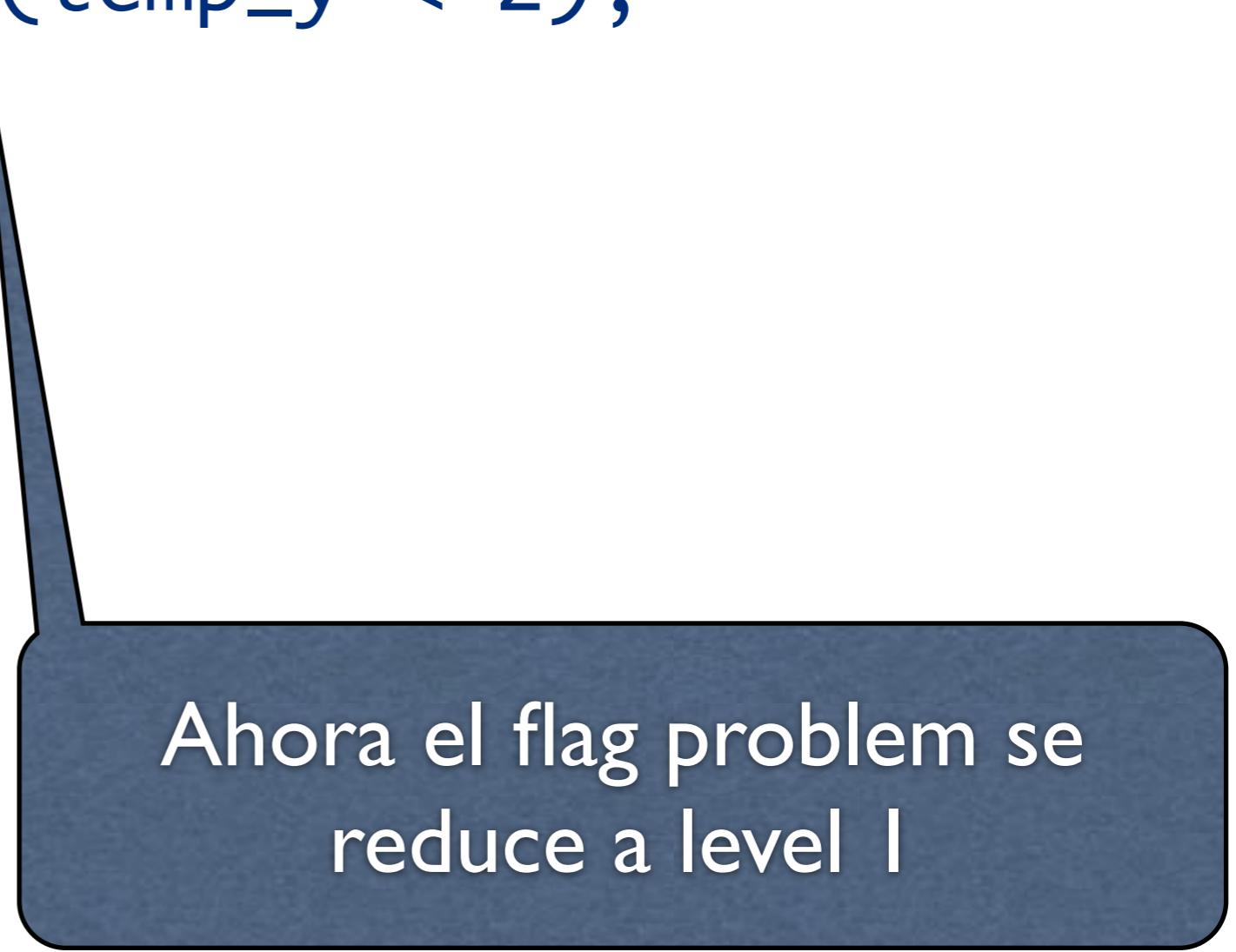
```
// ...
x = 5;
y = x;
// ...
if(flag) {
    // ...
}
// ...
```

# Flag Level 2

```
int temp_x = x; int temp_y = y;  
boolean flag = (temp_x >= 10) &&  
               (temp_y < z);  
  
// ..  
x = 5;  
y = x;  
  
// ..  
if(flag) {  
    // ..  
}  
// ..
```

# Flag Level 2

```
int temp_x = x; int temp_y = y;  
boolean flag = (temp_x >= 10) &&  
               (temp_y < z);  
  
// ..  
x = 5;  
y = x;  
// ..  
if(flag) {  
    // ..  
}  
// ..
```



Ahora el flag problem se  
reduce a level 1

# Flag Level 3

```
x = y + 1;  
y = x * 2;  
flag = x>y;  
y = y + flag;  
flag = flag || y == 0;  
x = y * x;  
if(flag) ...
```

# Flag Level 3

```
x = y + 1;  
y = x * 2;  
flag = x>y;  
  
y = y + flag;  
flag = flag || y == 0;  
x = y * x;  
if(flag) ...
```

```
flag = (y+1)>((y+1)*2) ||  
(((y+1)*2)+(y+1)>((y+1)*2))==0
```

# Flag Problem

- Level 4: Secuencia de flags contiene condicionales
- Level 5: Definición de flags en diferentes loop-bodies, than flag use

# Mejorando Testabilidad

```
if list.isEmpty()  
{...} ???
```

# Mejorando Testabilidad

if list.isEmpty() → branch ?  
{...} ???

if  
(list.size()==0)  
{...}

branch ?  
branch  
instance  
clear

# Mejorando Testabilidad

```
Integer a = ...;
```

```
Integer b = ...;
```

```
if (a.equals(b)) {...} ???
```

# Mejorando Testabilidad

```
Integer a = ...;
```

```
Integer b = ...;
```

```
if (a.equals(b)) {...} ???
```



```
if (b!=null &&  
a.intValue()==b.intValue()) {...}
```

# Mejorando Testabilidad

```
if intSet.contains(k)  
{...} ???
```

# Mejorando Testabilidad

```
if intSet.contains(k)  
{...} ???
```

If  $\text{minDist}(\text{intSet}, k) == 0$  {...}

# Mejorando Testabilidad

```
static int minDist(Set<Integer> aSet, int value) {  
    if (aSet.isEmpty()) return MAX_VALUE;  
    else  
        return intSet.stream()  
            .mapToInt(e -> Math.abs(e-value))  
            .min()  
            .getAsInt();  
}
```

# Mejorando Testabilidad

```
if intMap.containsKey(k)  
{...} ???
```

# Mejorando Testabilidad

```
if intMap.containsKey(k)  
{...} ???
```

```
If minDist(intMap.keySet(), k) == 0  
{...}
```

# Mejorando Testabilidad

```
if str1.equals(str2) {...} ???
```

# Mejorando Testabilidad

```
if str1.equals(str2) {...} ???
```

```
If hammingDistance(str1,str2)==0  
{...}
```

# Distancia de Hamming

```
static int hammingDistance(String str1,  
String str2) {  
  
    int c = 0;  
  
    for (int i=0; i<min(str1.length(),str2.length(),i++) {  
  
        if (str1[i]!=str2[i]) c++;  
  
    }  
  
    c += (max(str1.length(),str2.length()) - min(str1.length(),str2.length()));  
  
    return c;  
}
```

# Distancia de Hamming

## (2)

```
static int hammingDistance(String str1, String str2) {  
    int c = 0;  
  
    for (int i=0; i<min(str1.length(),str2.length()),i++) {  
        if (str1[i]!=str2[i]) c+= abs(str1[i]-str2[i]);  
    }  
  
    c += (max(str1.length(),str2.length()) -  
        min(str1.length(),str2.length())) * MAX_DISTANCE;  
  
    return c;  
}
```

→ Hello World  
changes  
Hello to be gone  
to make new.

# Distancia de Levenshtein

- La distancia de hamming no siempre indica la distancia entre dos strings.
- Ejemplo: “Hello World” y “ello World”.
- En el ejemplo anterior la distancia de hamming será muy alta incluso cuando falta poco para hacer iguales ambos strings
- Distancia de Levenshtein es la cantidad mínima de ediciones de un único character (insersiones, borrados or sustituciones) que se necesitan para convertir str1 en str2.

# Distancia de Levenshtein

- `hamming("Hello_World" , "ello_World_")=11`
- `levenshtein("Hello_World" , "ello_World_")=2`

# Distancia de Levenshtein

- Sea  $a, b$  strings, la distancia de Levenshtein se computa como  $\text{lev}_{\{a,b\}}(|a|, |b|)$  donde  $|a|, |b|$  es la longitud de  $a, b$

$$\text{lev}_{a,b}(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \\ \min \left\{ \begin{array}{l} \text{lev}_{a,b}(i - 1, j) + 1 \\ \text{lev}_{a,b}(i, j - 1) + 1 \\ \text{lev}_{a,b}(i - 1, j - 1) + 1_{(a_i \neq b_j)} \end{array} \right\} & \text{otherwise.} \end{cases}$$

# Otras transformaciones

- `Integer.parseInt(...), Double.parseDouble(...)`
- `DateFormat.parse(...)`
- `Pattern.Matcher.find()`
- `Pattern.Matcher.matches()`



- Automatic Test Suite Generation for Java
- <http://www.evosuite.org>
- <https://github.com/EvoSuite/evosuite>
  - GNU Lesser General Public License (LGPL)
  - Plugins: Eclipse, IntelliJ, Maven, Jenkins



- Optimiza Test-Suites enteros (“Whole Test Suite Generation”)
- Trabaja directamente sobre el Java Bytecode (no hay que proveer source code)
- Demo!

QuickTime Player File Edit View Window Help

Java - Example/src/example/Foo.java - Eclipse Platform

Package Explorer

- Example
  - src
    - example
      - Foo.java
  - JRE System Library [JavaSE-1.6]
  - JUnit 4
  - Referenced Libraries

Foo.java

```
package example;

public class Foo {
    private int x = 0;
    private String str;
    private String str2="bar";
    public Foo(String string) {
        this.str = string;
    }
    public void inc() {
        x++;
    }
    public boolean coverMe() {
        if (x==5)
            if(!str.equals(str2))
                if (str.equalsIgnoreCase(str2))
                    return true;
        return false;
    }
}
```

Coverage

Element	Coverage	Covered Instructions	Missed Instructions

example.Foo.java - Example/src

The screenshot shows the Eclipse IDE interface. The top bar includes the QuickTime Player title, menu items (File, Edit, View, Window, Help), system status icons (battery, signal, etc.), and user information (Mon 20:44, Gordon Fraser). The central workspace displays the Java code for 'Foo.java' in the 'example' package. The 'Coverage' view is open at the bottom, showing a table with four columns: Element, Coverage, Covered Instructions, and Missed Instructions. The table currently has one row with empty cells. On the left, the 'Package Explorer' shows the project structure with 'src/example/Foo.java' selected. The right side features a vertical toolbar with various icons for file operations like cut, copy, paste, and search.

QuickTime Player File Edit View Window Help

Java - Example/src/example/Foo.java - Eclipse Platform

Package Explorer

- Example
  - src
    - example
      - Foo.java
  - JRE System Library [JavaSE-1.6]
  - JUnit 4
  - Referenced Libraries

Foo.java

```
package example;

public class Foo {
    private int x = 0;
    private String str;
    private String str2="bar";
    public Foo(String string) {
        this.str = string;
    }
    public void inc() {
        x++;
    }
    public boolean coverMe() {
        if (x==5)
            if(!str.equals(str2))
                if (str.equalsIgnoreCase(str2))
                    return true;
        return false;
    }
}
```

Coverage

Element	Coverage	Covered Instructions	Missed Instructions

example.Foo.java - Example/src

This screenshot shows the Eclipse IDE interface running on a Mac OS X system. The title bar indicates it's a Java project named 'Example' with 'src/example/Foo.java' open. The left side features the Package Explorer showing the project structure with 'Foo.java' selected. The central area contains the code editor for 'Foo.java' with syntax highlighting. Below the editor is the Coverage view, which is currently empty. The right side has a vertical toolbar with various icons. The status bar at the bottom shows the file path 'example.Foo.java - Example/src'.

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help Mon 20:44 Gordon Fraser Q

Java - Example/evosuite-tests/example/FooEvoSuiteTest.java - Eclipse Platform

Quick Access Resource Java

Package Explorer

Example

src

example

Foo.java

JRE System Library [JavaSE-1.6]

JUnit 4

Referenced Libraries

evoSuite-tests

example

FooEvoSuiteTest.java

evoSuite-report

Foo.java FooEvoSuiteTest.java

```
* This file was automatically generated by EvoSuite

package example;

import org.junit.Test;

public class FooEvoSuiteTest {

    @Test
    public void test0() throws Throwable {
        Foo foo0 = new Foo("bar");
        foo0.inc();
        foo0.inc();
        foo0.inc();
        foo0.inc();
        foo0.inc();
        boolean boolean0 = foo0.coverMe();
        assertEquals(false, boolean0);
    }

    @Test
}
```

Coverage

Element	Coverage	Covered Instructions	Missed Instructions

Writable Smart Insert 2 : 1

S

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help Mon 20:44 Gordon Fraser Q

Java - Example/evosuite-tests/example/FooEvoSuiteTest.java - Eclipse Platform

Quick Access Resource Java

Package Explorer

Example

src

example

Foo.java

JRE System Library [JavaSE-1.6]

JUnit 4

Referenced Libraries

evoSuite-tests

example

FooEvoSuiteTest.java

evoSuite-report

Foo.java FooEvoSuiteTest.java

```
* This file was automatically generated by EvoSuite

package example;

import org.junit.Test;

public class FooEvoSuiteTest {

    @Test
    public void test0() throws Throwable {
        Foo foo0 = new Foo("bar");
        foo0.inc();
        foo0.inc();
        foo0.inc();
        foo0.inc();
        foo0.inc();
        boolean boolean0 = foo0.coverMe();
        assertEquals(false, boolean0);
    }

    @Test
}
```

Coverage

Element	Coverage	Covered Instructions	Missed Instructions

Writable Smart Insert 2 : 1

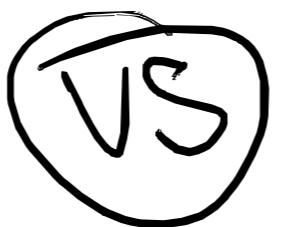
S

Whole Test Suite Generation  
↳ full coverage ( pools )

# Search-Based Test Generation

- Search-based Testing: Transformar la generación de tests en un problema de *optimización combinatoria*.
- Algoritmos Genéticos:
  - Imita el proceso natural de evolución
  - Traditional approach: optimize test case for each objective goal in isolation

WTS G

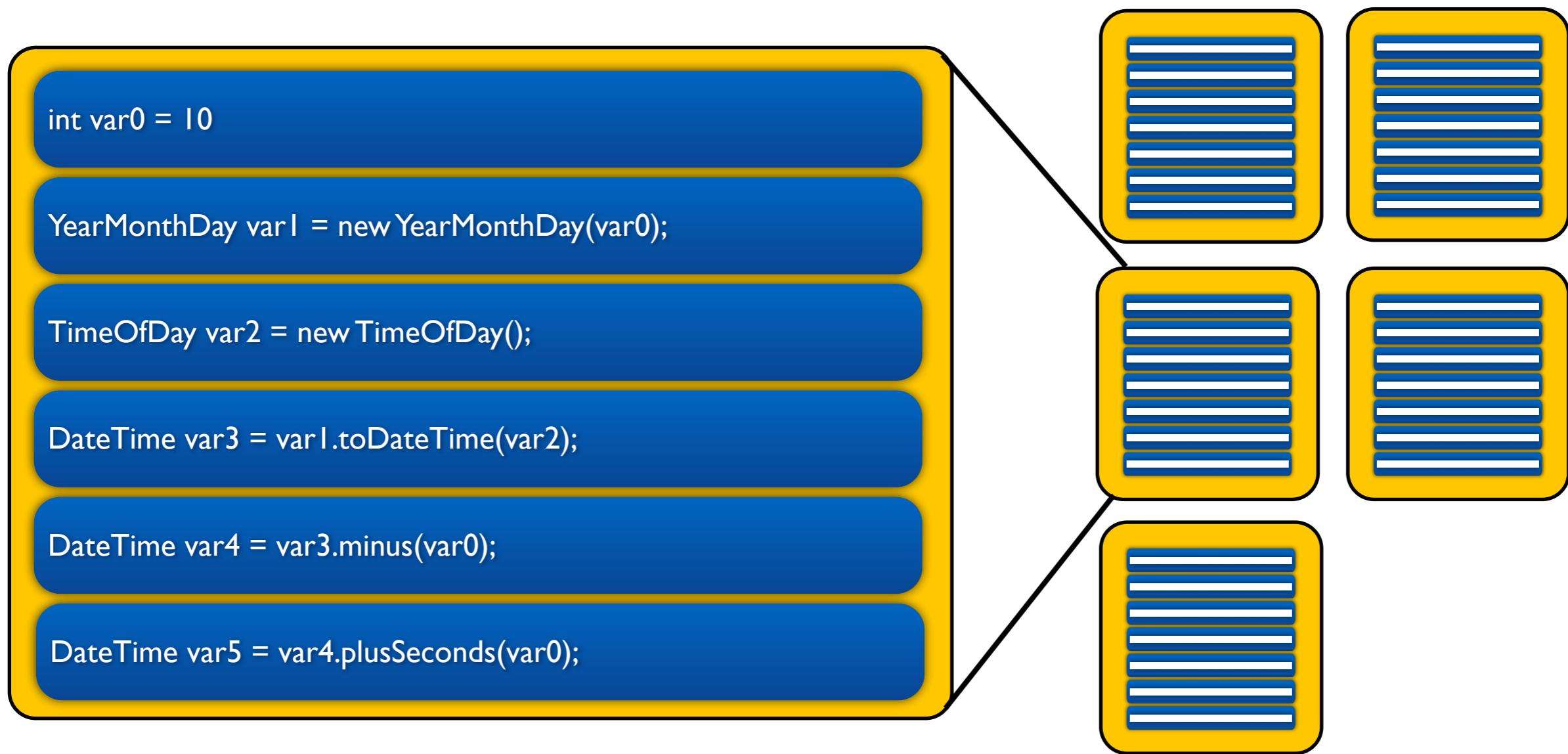


# Estrategia de Goal/ Objetivo único

- Población: conjunto de test cases
- Sean G1, G2, G3 objetivos de test:
  - Cómo podemos distribuir el presupuesto de generación?
    - Qué pasa si G2 es insatisfacible?
    - Qué pasa si G1 es más complejo que G3?

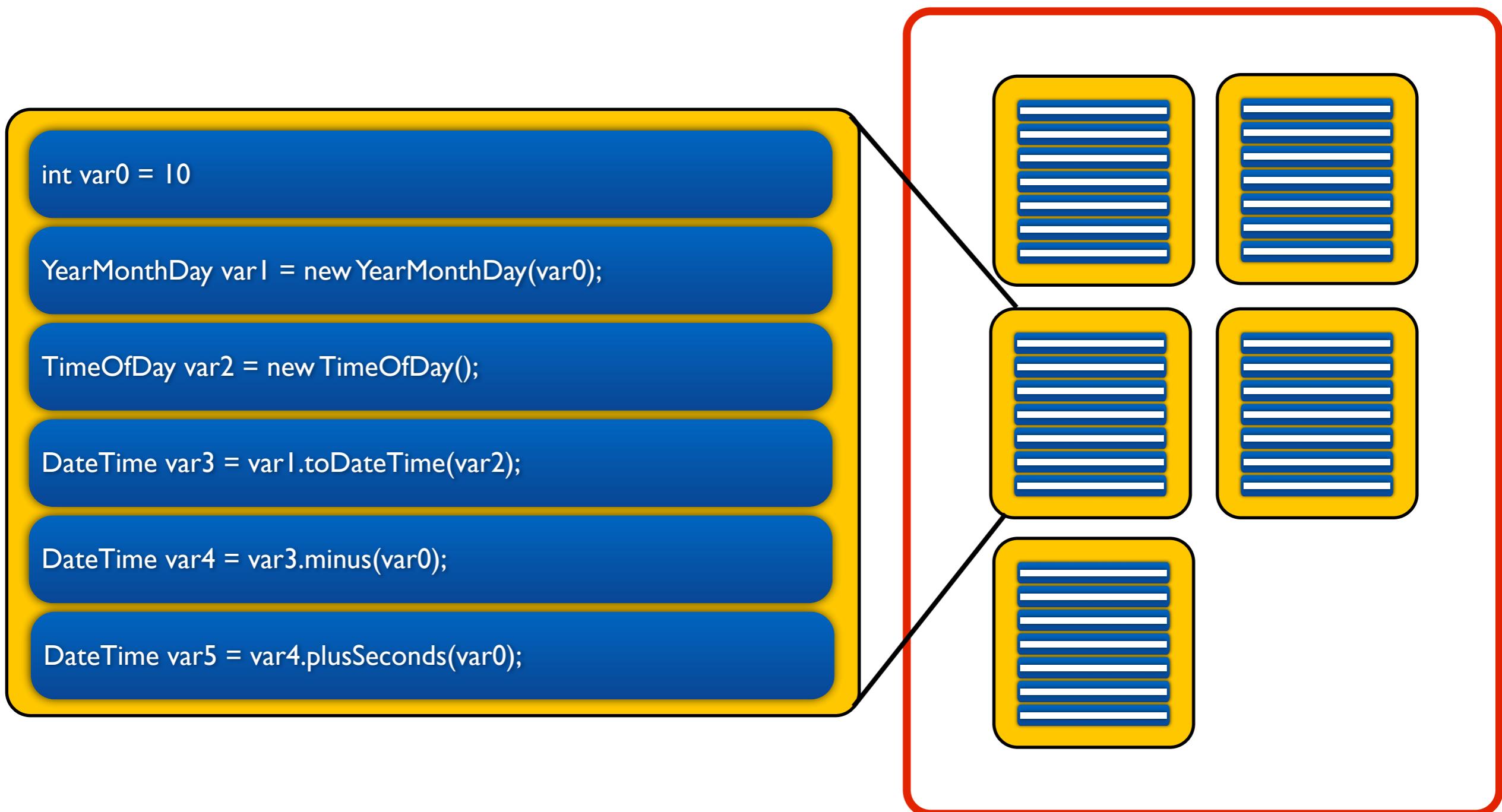
~~H~~  
PRESUPUESTO  
PESA MUCHO

# EvoSuite: Whole-Test Suite Generation



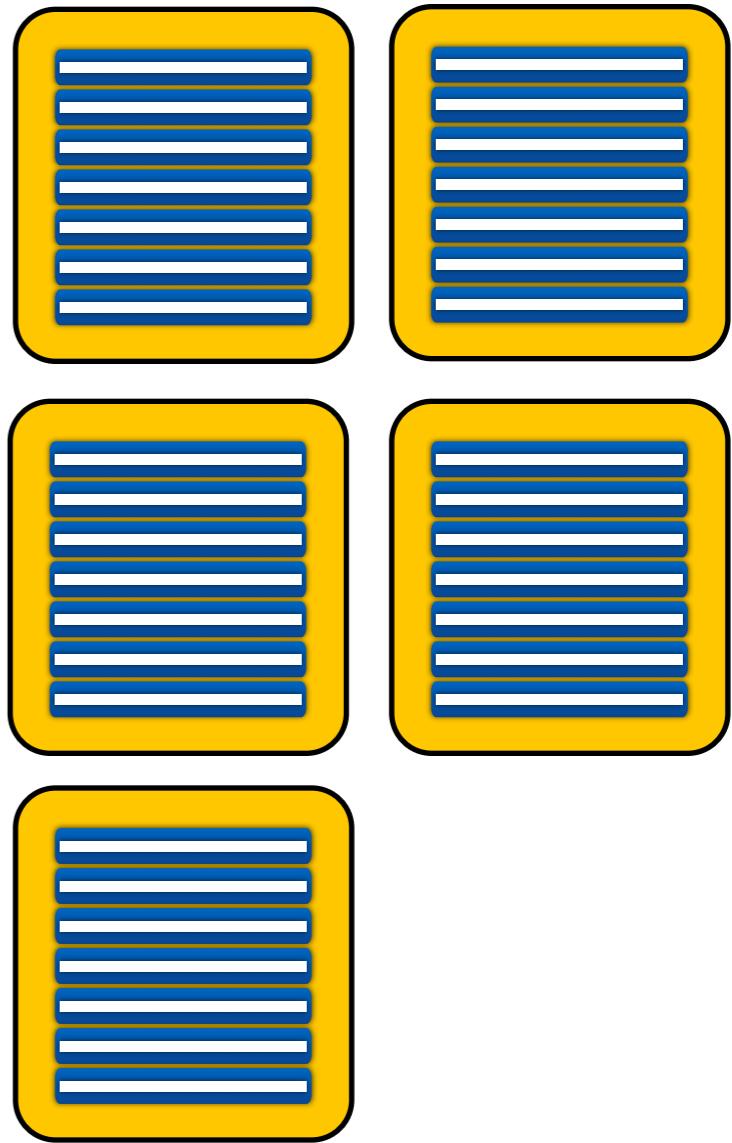
Individuos son copy de casos  
de test. (Test suites)

# EvoSuite: Whole-Test Suite Generation

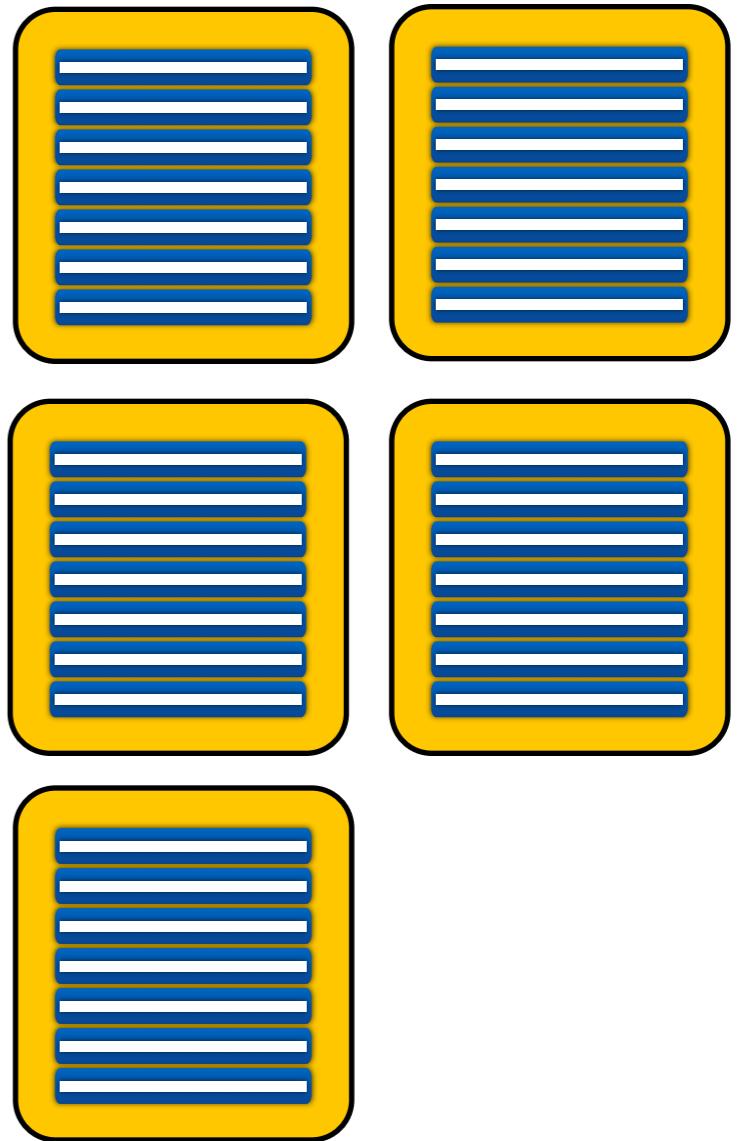


# EvoSuite: Whole-Test Suite Generation

- Optimizamos test suites al mismo tiempo  
*(mejoró fitness de test suite)*
- La distribución de budget entre los goals ya no importa
- La insatisfacibilidad goals individuales no afecta negativamente la búsqueda

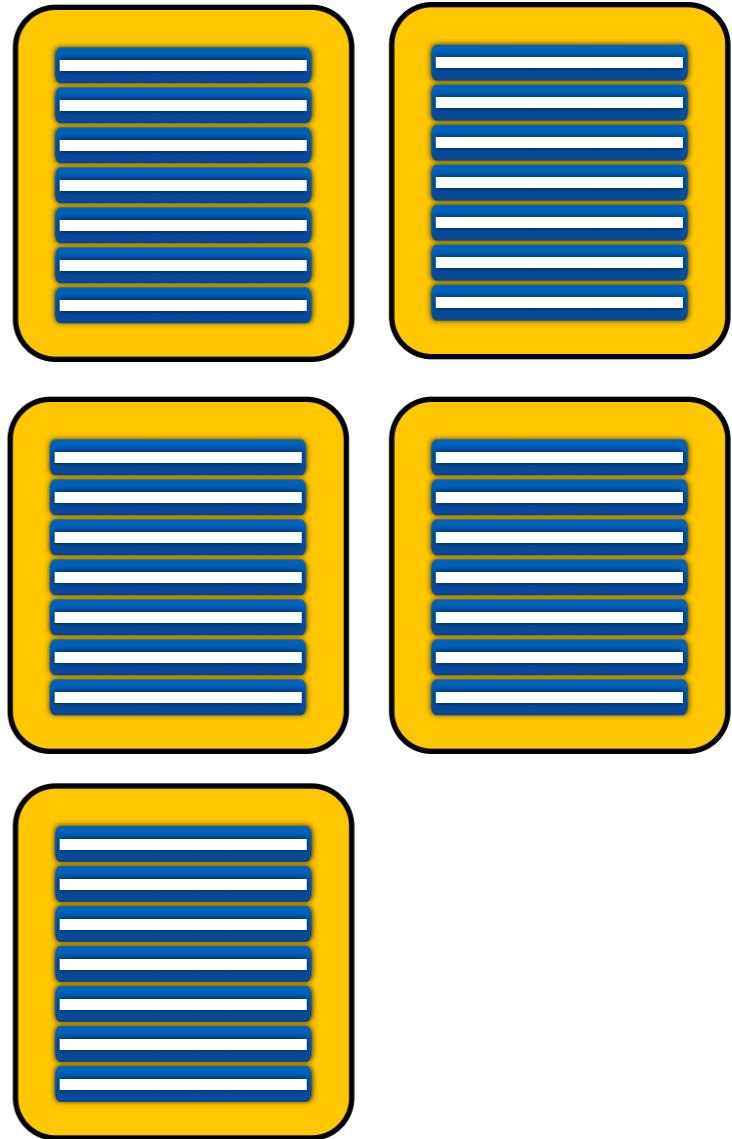


# Algoritmo Genético

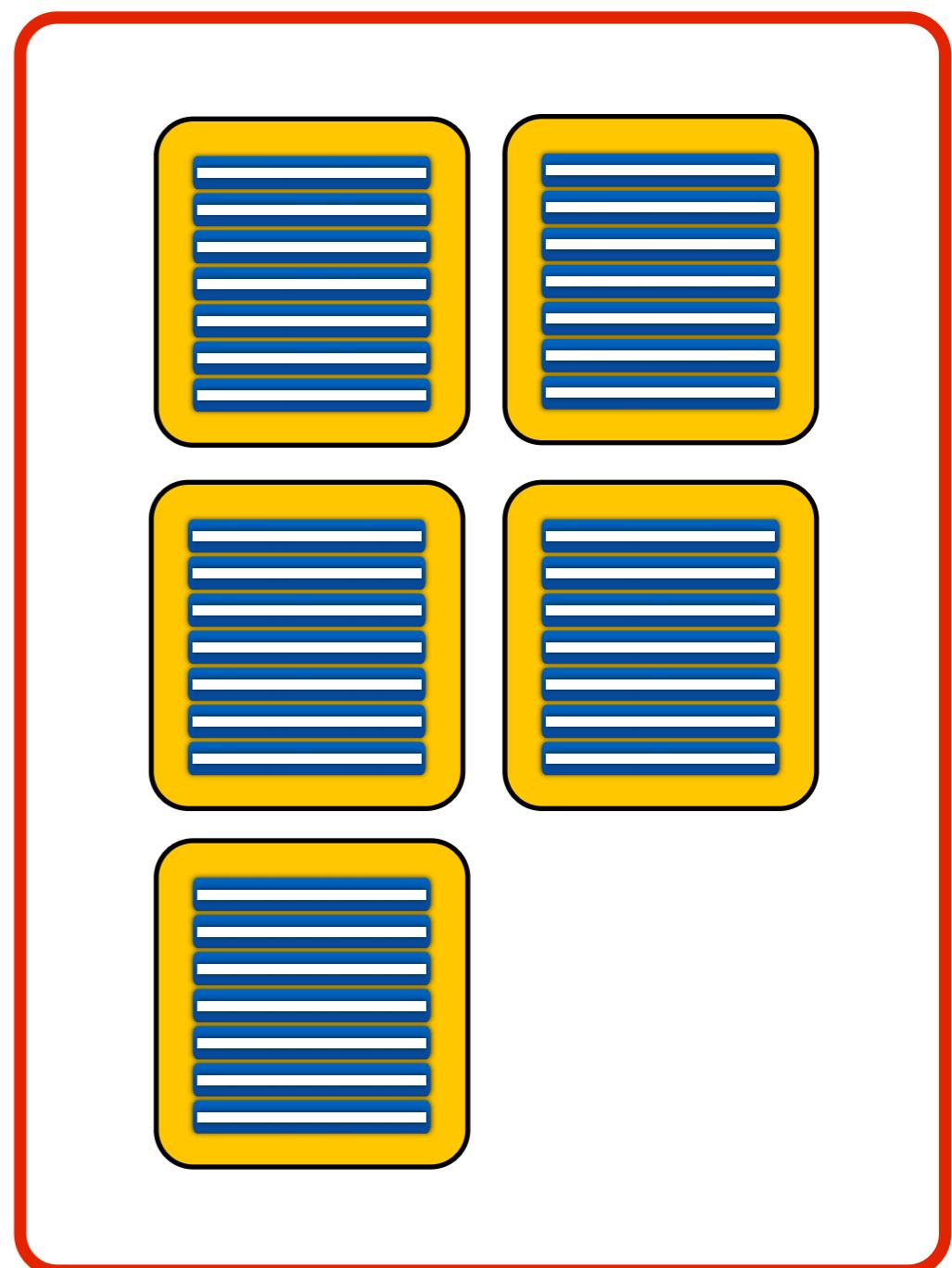
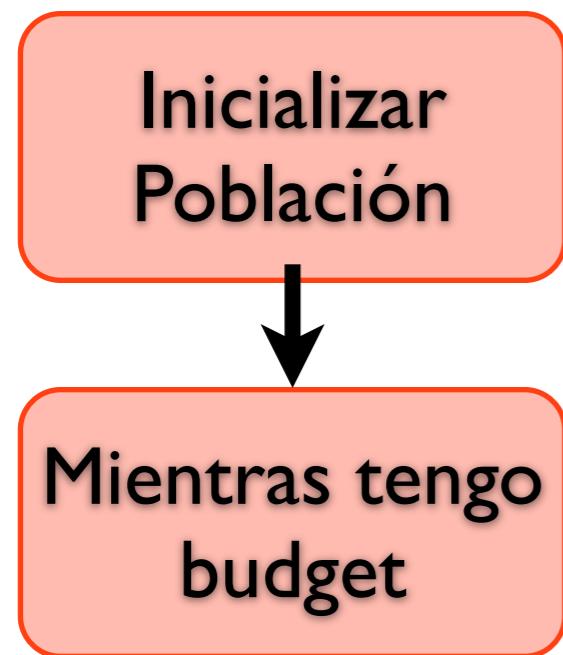


# Algoritmo Genético

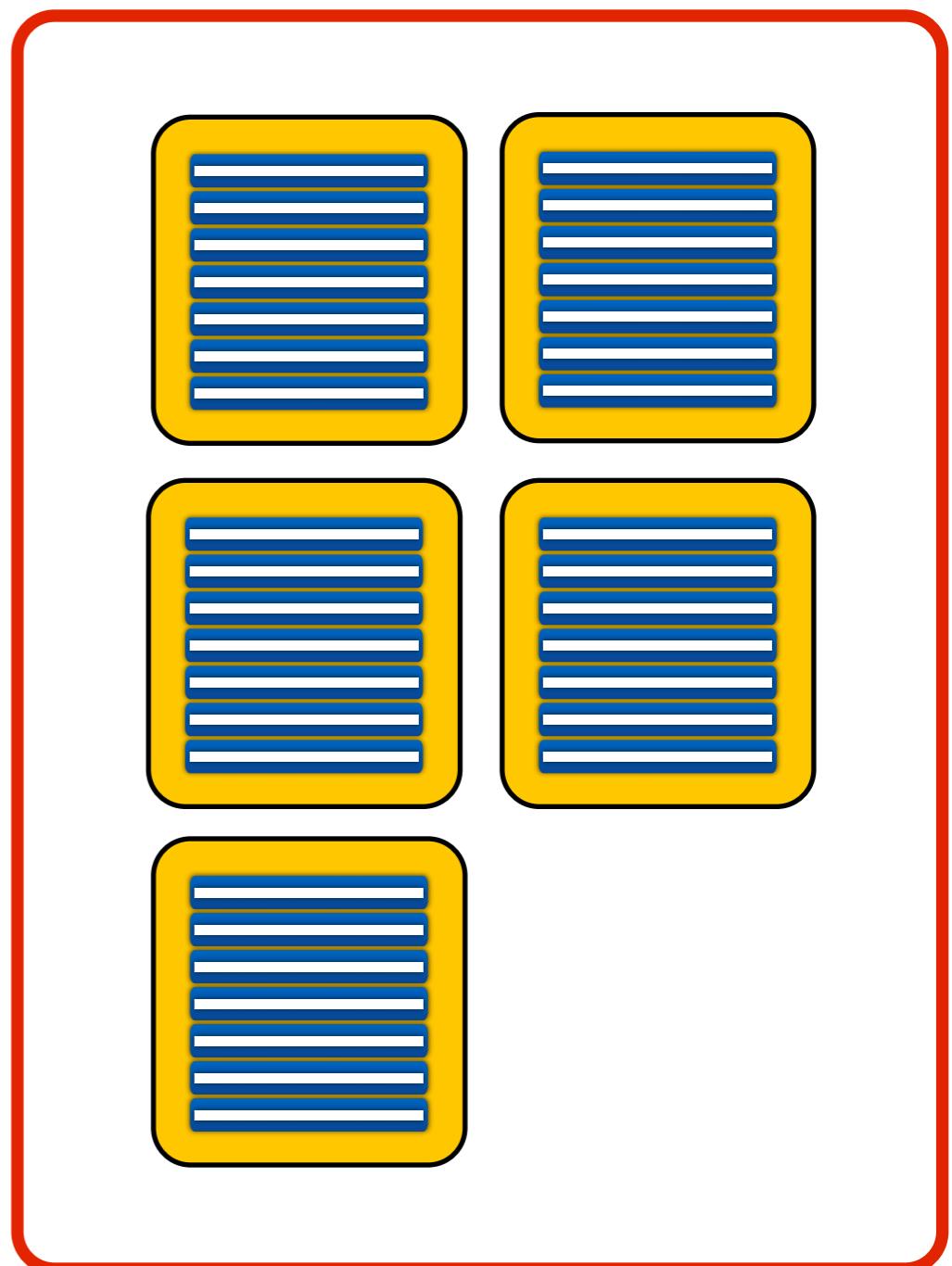
Iniciar  
Población



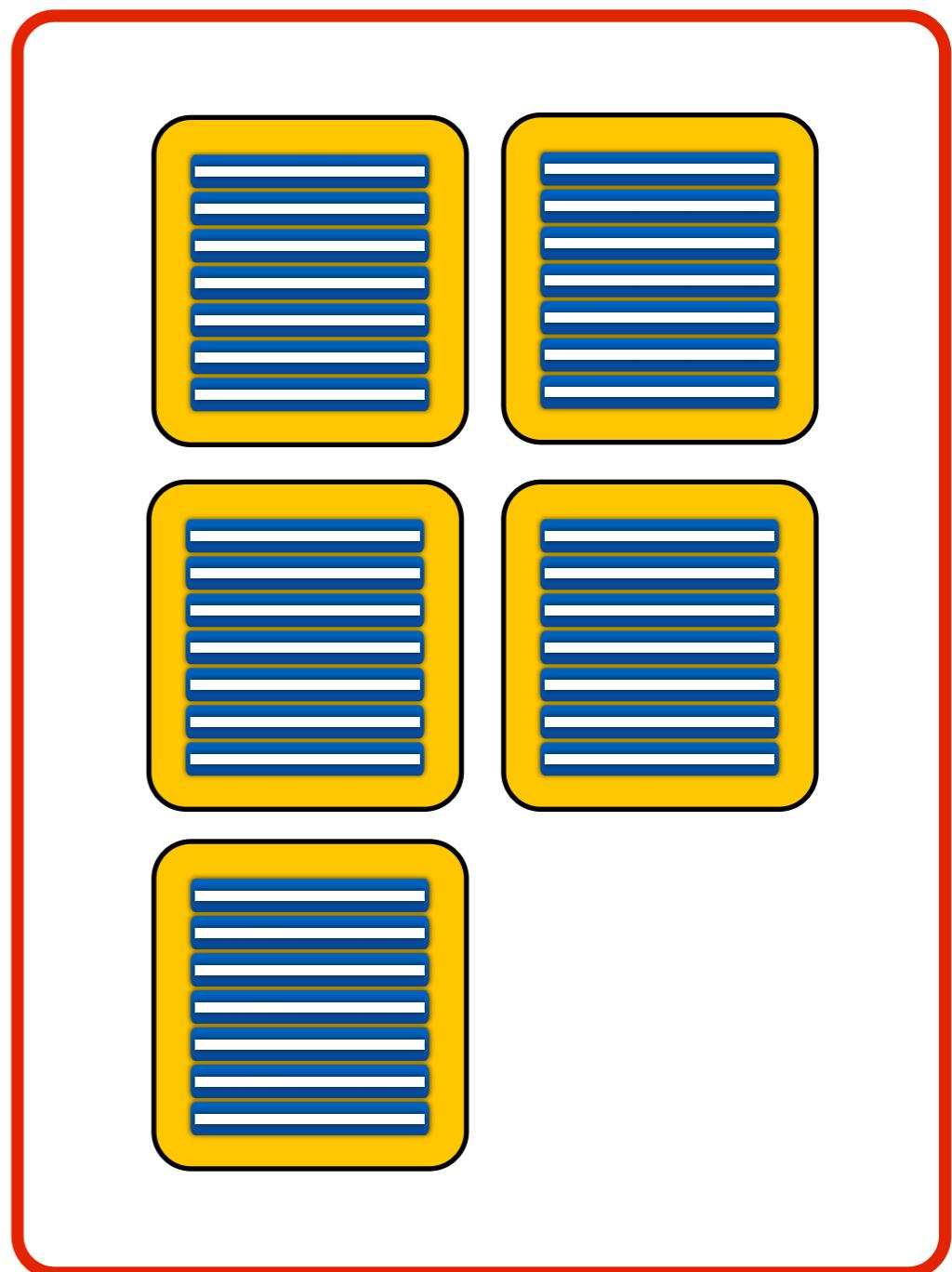
# Algoritmo Genético



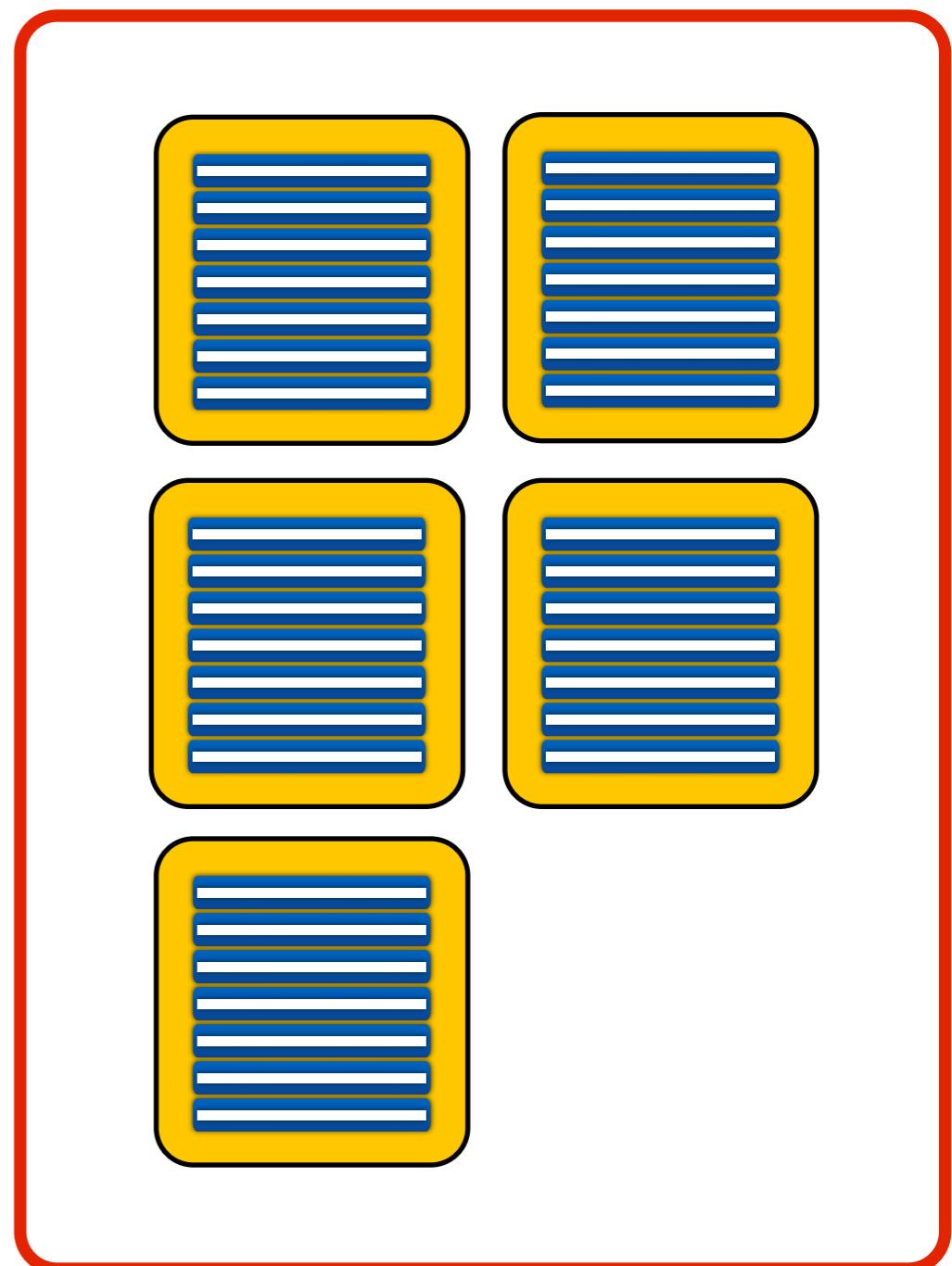
# Algoritmo Genético



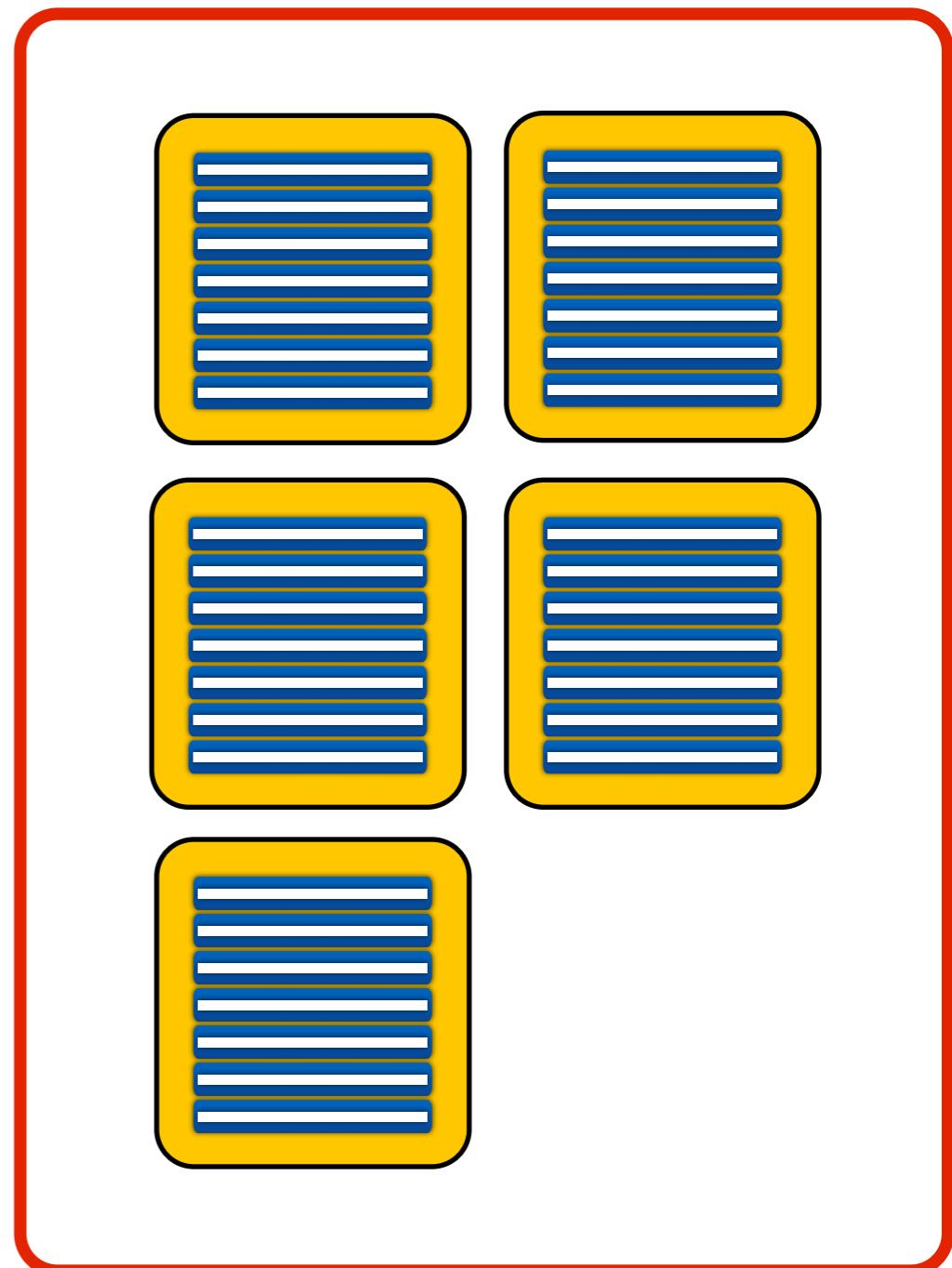
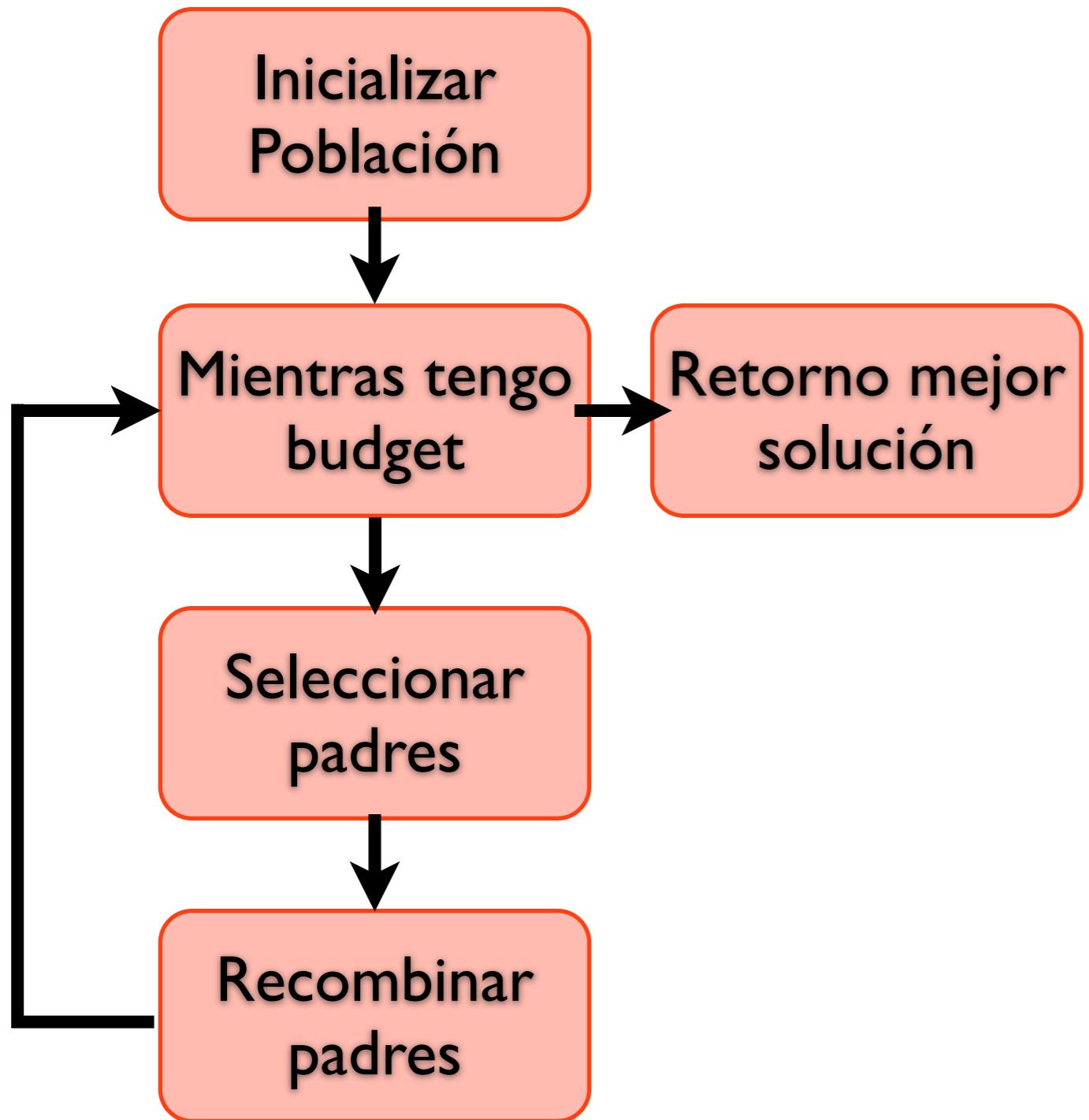
# Algoritmo Genético



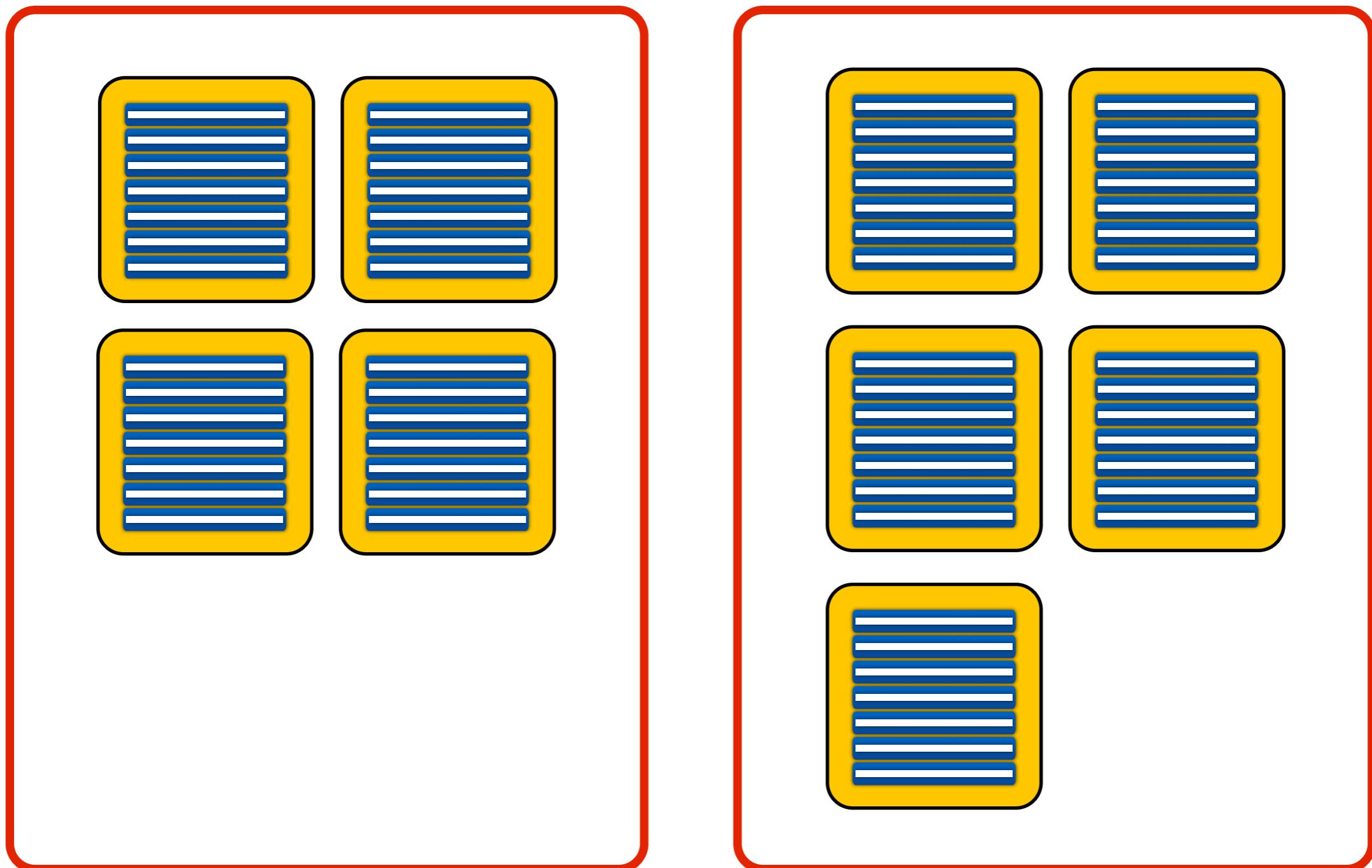
# Algoritmo Genético



# Algoritmo Genético

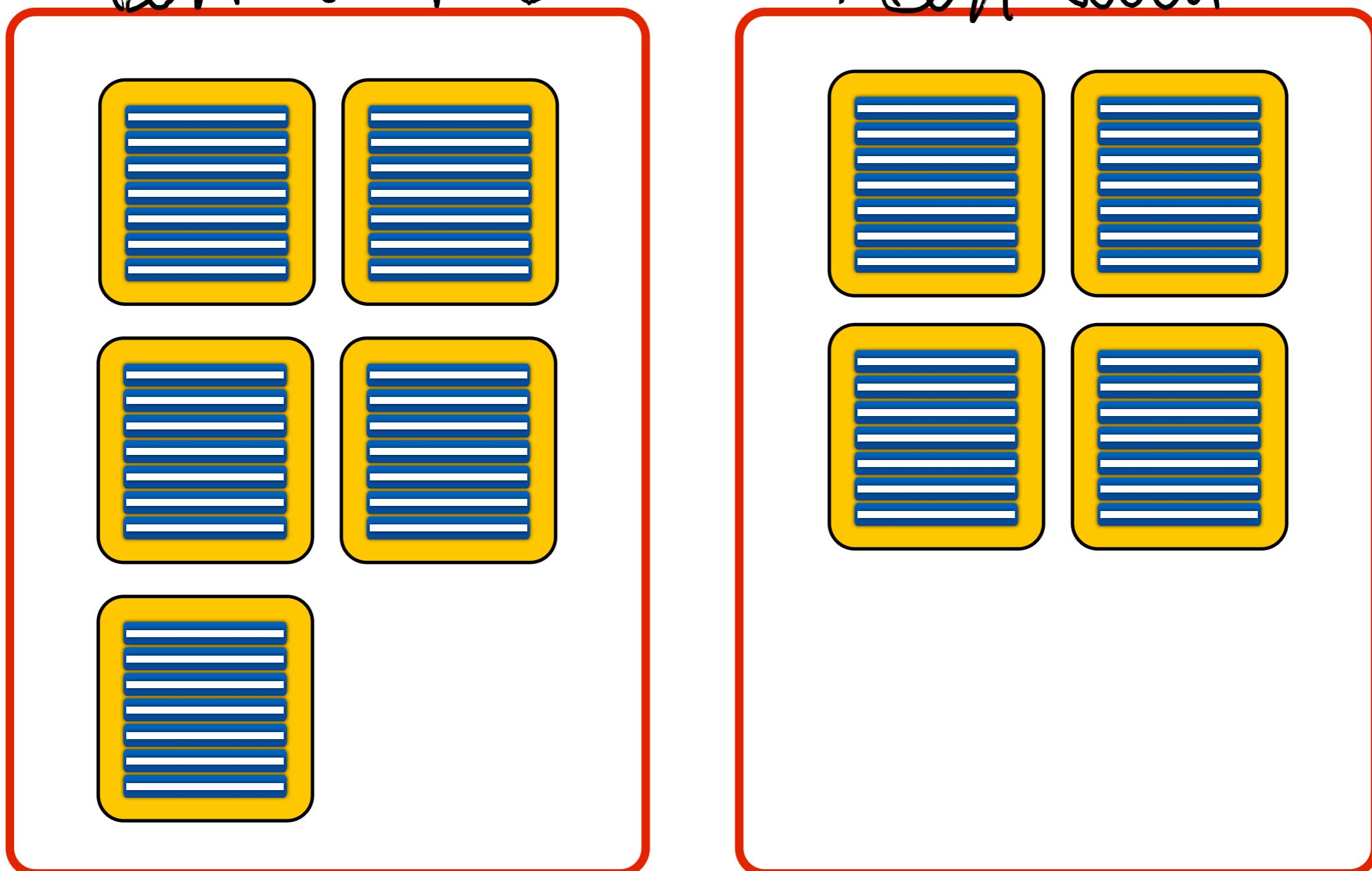


# Whole-Test Suite Generation



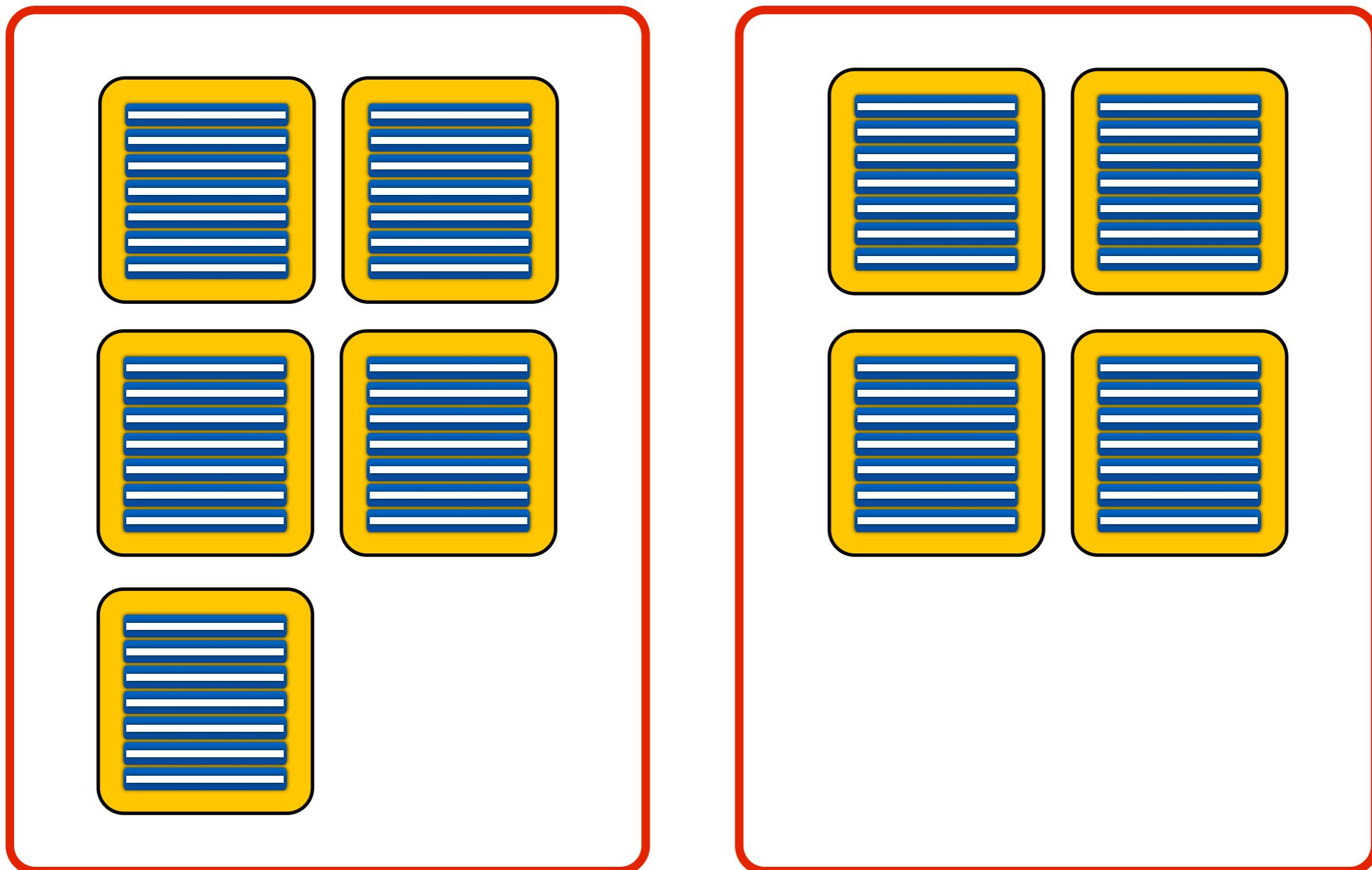
# Crossover

Test Suite (Individual) Test Suite

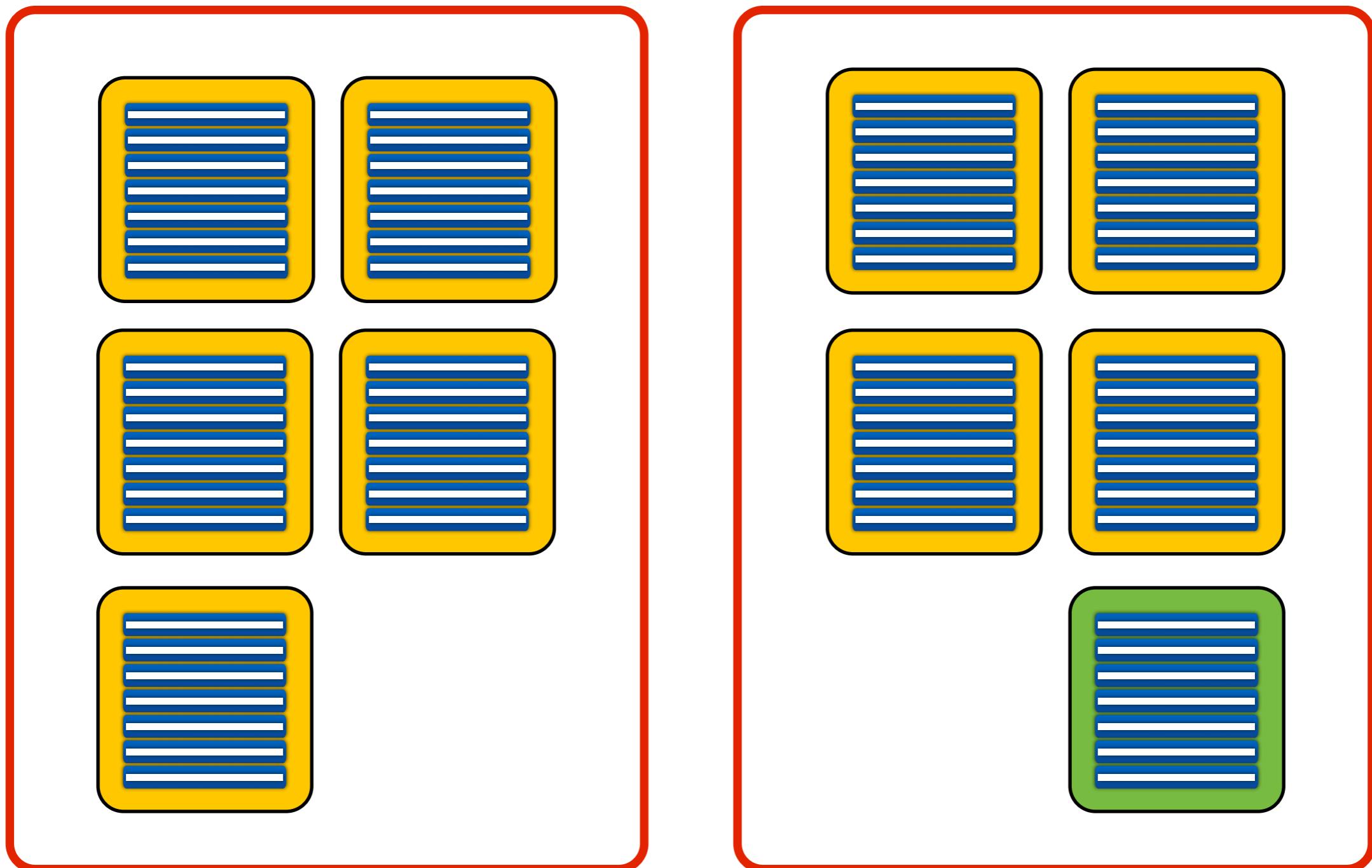


Mutación  
de test cases (genes)

# Mutación

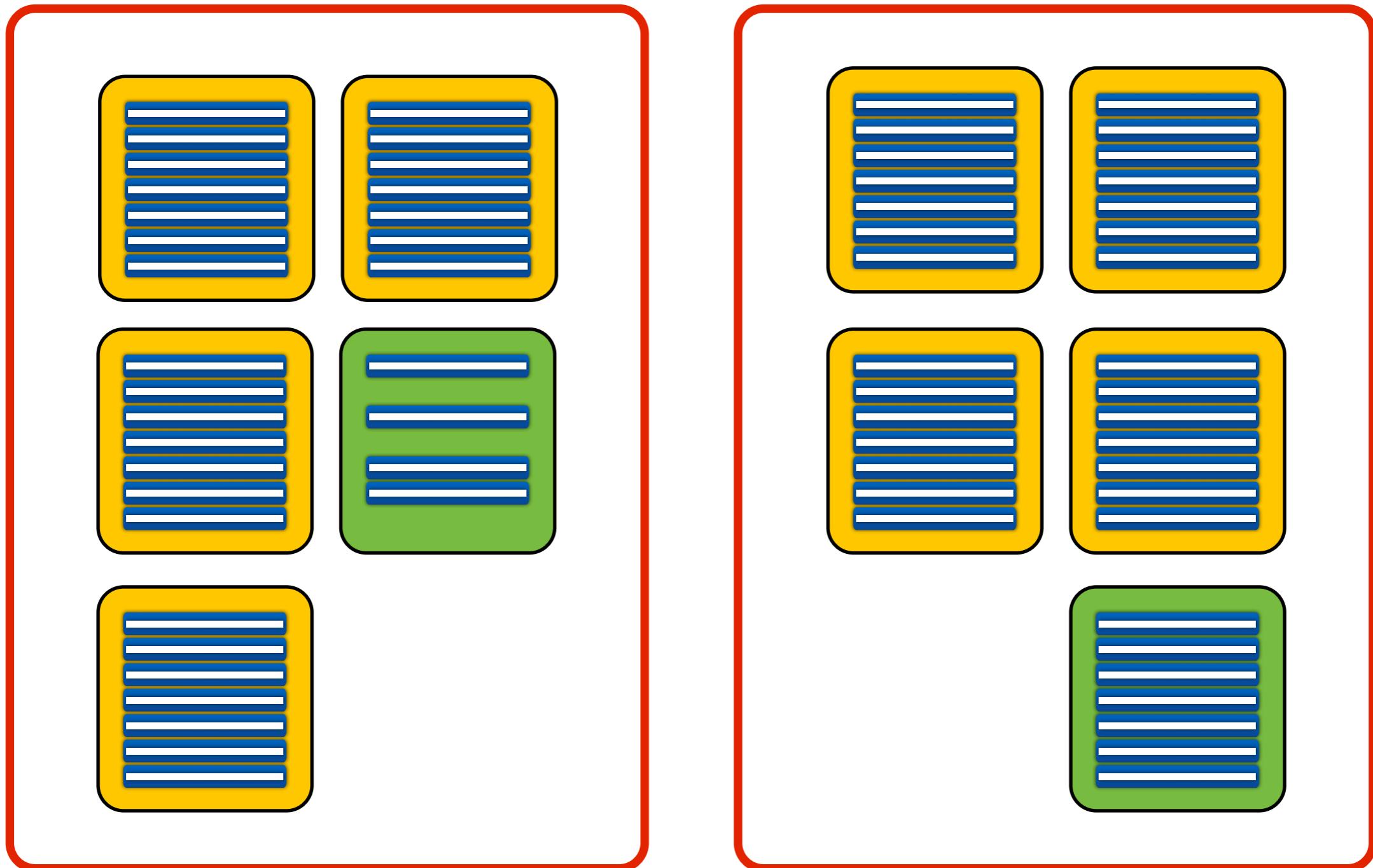


# Mutación



modifica o  
cambia el  
contexto

# Mutación



# Mutación: Borrado

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Borrado

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Borrado

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Borrado

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var5 = var3.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Borrado

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Borrado

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Borrado

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Cambio

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Cambio

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.plus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Cambio

```
int var0 = 20
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.plus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Inserción

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Inserción

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Inserción

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
var3.reset();
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Inserción

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
var3.reset();
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Mutación: Inserción

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
TimeOfDay var2a = var2.minus(var0);
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
var3.reset();
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

Mutations need to TAKE  
CARING OF COMPIRATION  
(if modification makes one invalid  
compilation → needs to fix it)

---

**Algorithm 1** The genetic algorithm applied in EvoSuite

---

```
1 current_population ← generate random population
2 repeat
3    $Z \leftarrow$  elite of current_population
4   while  $|Z| \neq |current\_population|$  do
5      $P_1, P_2 \leftarrow$  select two parents with rank selection
6     if crossover probability then
7        $O_1, O_2 \leftarrow$  crossover  $P_1, P_2$ 
8     else
9        $O_1, O_2 \leftarrow P_1, P_2$ 
10    mutate  $O_1$  and  $O_2$ 
11     $f_P = \min(fitness(P_1), fitness(P_2))$ 
12     $f_O = \min(fitness(O_1), fitness(O_2))$ 
13     $l_P = length(P_1) + length(P_2)$ 
14     $l_O = length(O_1) + length(O_2)$ 
15     $T_B =$  best individual of current_population
16    if  $f_O < f_P \vee (f_O = f_P \wedge l_O \leq l_P)$  then
17      for  $O$  in  $\{O_1, O_2\}$  do
18        if  $length(O) \leq 2 \times length(T_B)$  then
19           $Z \leftarrow Z \cup \{O\}$ 
20        else
21           $Z \leftarrow Z \cup \{P_1 \text{ or } P_2\}$ 
22      else
23         $Z \leftarrow Z \cup \{P_1, P_2\}$ 
24    current_population  $\leftarrow Z$ 
25 until solution found or maximum resources spent
```

# FITNESS FUNCTIONS

(I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



# FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



:



Method Coverage

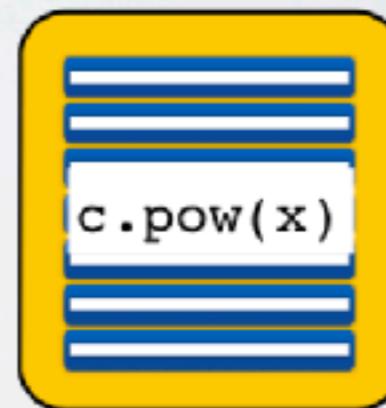
Todos los métodos en la CUT deben ser ejecutados

# FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



:

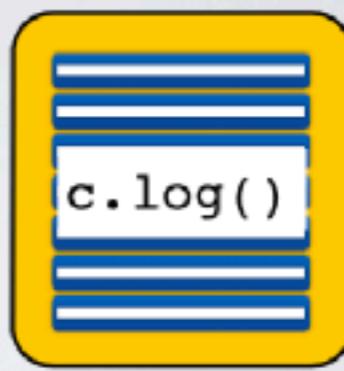


Top-level Method Coverage

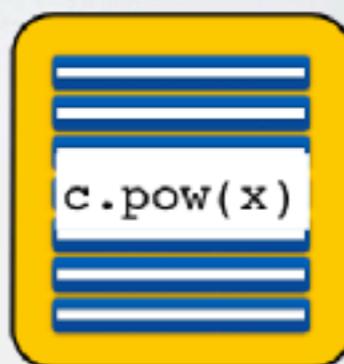
Cada método es invocado **directamente** (desde el test case)

# FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



:



Top-level Method Coverage No exception

todos los métodos son cubiertos con invocaciones directas desde los tests y únicamente se consideran ejecuciones que terminan normalmente (i.e., no hay excepciones)

# FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
    1 if (isNaN) return NaN;
    2 double r = log(abs());
    3 double i;
    4 i = atan2(imaginary, real);
    5 return createComplex(r, i);
}

public Complex pow(double x)
    throws NullArgumentException{
    6 Complex c = this.log();
    7 return c.multiply(x).exp();
}
```



:



Line Coverage

Todos los statements son ejecutados

# FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1 if (isNaN) return NaN;
2 double r = log(abs());
3 double i;
4 i = atan2(imaginary, real);
5 return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6 Complex c = this.log();
7 return c.multiply(x).exp();
}
```



:



Branch Coverage

Todos los predicados de las decisiones se evalúan a verdadero y a falso.

# FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1 if (isNaN) return NaN;
2 double r = log(abs());
3 double i;
4 i = atan2(imaginary, real);
5 return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6 Complex c = this.log();
7 return c.multiply(x).exp();
}
```



:



Direct Branch Coverage

Cada branch en un método accesible es cubierto por una invocación directa desde un test de unidad.

# FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



:



Weak Mutation

Se produce una infección del estado (no necesariamente una propagación) por cada mutante del CUT

# FITNESS FUNCTIONS

(I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
    1 if (isNaN) return NaN;
    2 double r = log(abs());
    3 double i;
    4 i = atan2(imaginary, real); { 0
    5 return createComplex(r, i); +}
}

public Complex pow(double x)
    throws NullArgumentException{
    6 Complex c = this.log();
    7 return c.multiply(x).exp(); { null
}                                         nonnull
```



:



Output Coverage  
(e.g., Alshahwan and Harman, ISSTA'14)

Los métodos deben cubrir particiones de valores de retorno  
(positivos, negativos, dígitos, alfanuméricos, etc.)

# FITNESS FUNCTIONS

(I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



:



Number of exceptions

Cada método maximiza la cantidad de excepciones que emite  
(no podemos definirla con un porcentaje)

# FITNESS FUNCTIONS

## COMBINING MULTIPLE CRITERIA



# FITNESS FUNCTIONS

## COMBINING MULTIPLE CRITERIA

Method

Top-level Method

No-Exception  
Top-level Method

$$\text{fitness}(\text{Suite}) = \sum_{i=1}^n w_i \times f_i$$

nch

Output

Weak Mutation

Exception

# Combinando Múltiples Criterios

- **Criterios no contradictorios** (no necesitamos aplicar algoritmos multi-objetivo)
- Podemos fácilmente combinar todos los criterios para definir nuestra función de fitness ad-hoc. Ejemplo:
  - LINE:BRANCH
  - METHOD:EXCEPTION
  - Todos juntos
- Por default: mismo peso (**sin normalizar**)

# Post-Procesamiento del Test Suite

- Una vez que el Algoritmo Genético finaliza, EvoSuite post-procesa el test suite
  1. **Minimización**: Remueve los statements/tests que no contribuyen al cubrimiento de goals.
  2. **Generación de Aserciones**: Agrega assertions (con un patrón finito) tales que al menos un mutante es matado por la aserción
  3. **JUnit**: Comprueba que el JUnit resultante no es flaky.

# Minimización

While minimization\_budget is not empty:

Select test case from TestSuite

Remove statement from test case

Re-execute test suite

If coverage is the same:

    Update Test case

EndWhile

# Minimización

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Minimización

```
int var0 = 10
```

```
YearMonthDay var1 = new YearMonthDay(var0);
```

```
TimeOfDay var2 = new TimeOfDay();
```

```
DateTime var3 = var1.toDateTime(var2);
```

```
DateTime var4 = var3.minus(var0);
```

```
DateTime var5 = var4.plusSeconds(var0);
```

```
int var6 = var5.getMillis();
```

# Generación de Aserciones

Add All Assertions to Test Suite

While assertion\_budget is not empty:

Select Test Case from TestSuite

Select assertion from test case

Compute mutation score with/without assertion

If mutation score is the same:

Remove assertion from Test case

EndWhile

```
LocalDate date = new LocalDate(2010, 7, 15);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2010);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getYear(), 2010);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2010);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getYear(), 2010);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
assertEquals(date.getDayOfMonth(), ...);  
date.plusYears(1);  
assertEquals(date.getYear(), 2011);  
assertEquals(date.size(), 3);  
assertEquals(date.getValue(YEAR), 2011);  
assertEquals(date.getValue(MONTH_OF_YEAR), 7);  
assertEquals(date.getValue(DAY_OF_MONTH), 15);  
assertEquals(date.getLocalMillis(), ...);  
assertEquals(date, date);  
assertEquals(date.compareTo(date), 0);  
assertEquals(date.getYearOfEra(), ...);  
assertEquals(date.getYearOfCentury(), ...);  
assertEquals(date.getWeekyear(), ...);  
assertEquals(date.getMonthOfYear(), 7);  
assertEquals(date.getWeekOfWeekyear(), ...);  
assertEquals(date.getDayOfWeek(), ...);  
assertEquals(date.getDayOfMonth(), ...);
```

```
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2011);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfEra(), ...);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.getValue(YEAR), 2011);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

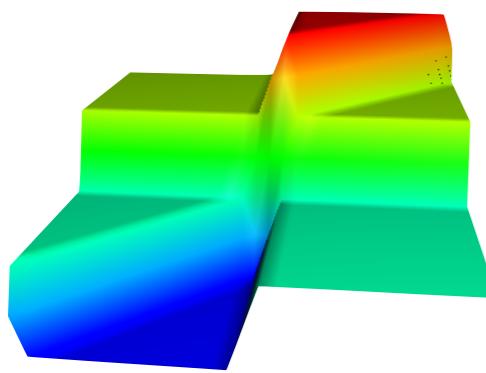
# Archive

- Fitness: mide si un individuo A es mejor que un individuo B
- Escenario:
  - A cubre las líneas L1,L2
  - B cubre las líneas L1,L3
- Cuál es el cubrimiento de líneas de cada individuo?

# Archive

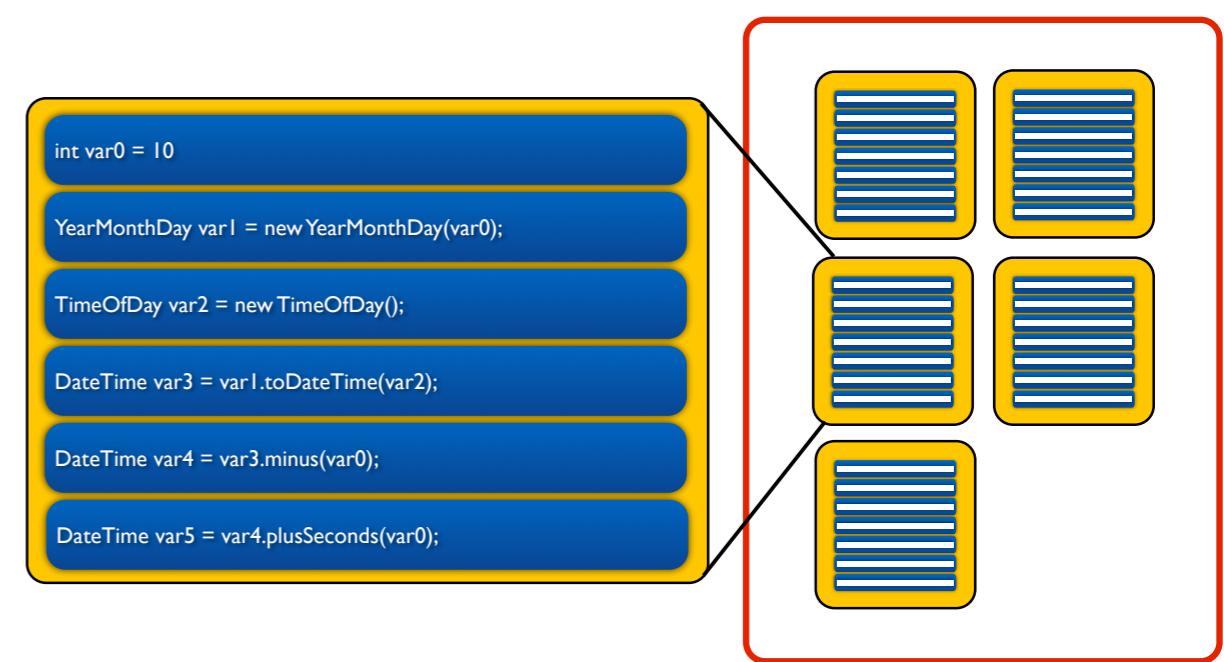
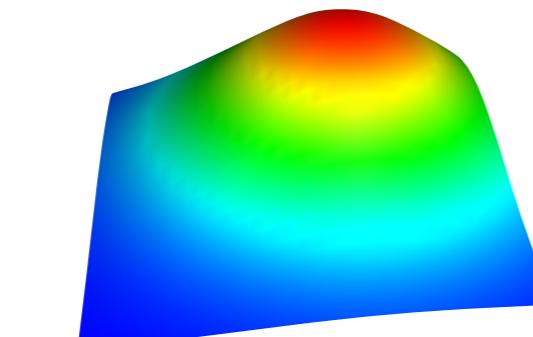
- Solución: En cada evaluación de fitness de un test T monitoreamos los goals de T
- Si cubre un nuevo goal "g"
  - Agregar T al "Archive"
  - Remuevo "g" del conjunto de goals a cubrir
- En lugar del mejor individuo, al finalizar el algoritmo genético devuelve el contenido del Archive

# EvoSuite: Whole-Test Suite Generation

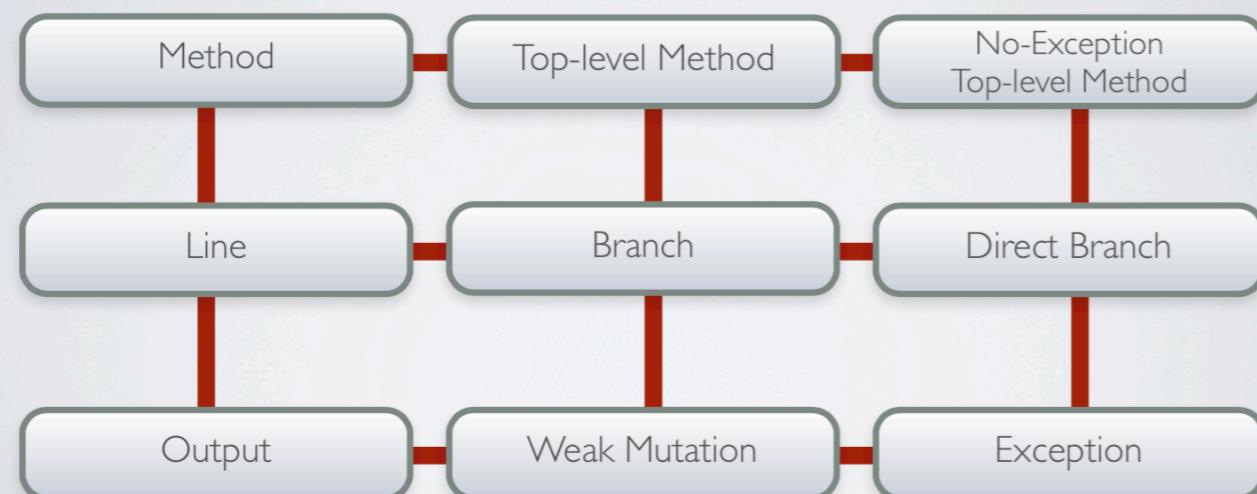


Si un programa es arduo para la generación de tests...

...podemos transformarlo a una versión más amigable para la generación basada en búsqueda



## FITNESS FUNCTIONS COMBINING MULTIPLE CRITERIA



## Post-Procesamiento del Test Suite

- Una vez que el Algoritmo Genético finaliza, EvoSuite post-procesa el test suite
  1. **Minimización:** Remueve los statements/tests que no contribuyen al cumplimiento de goals.
  2. **Generación de Aserciones:** Agrega assertions (con un patrón finito) tales que al menos un mutante es matado por la aserción
  3. **JUnit:** Comprueba que el JUnit resultante no es flaky.