Compiladores II

Generación de Código No Optimizado

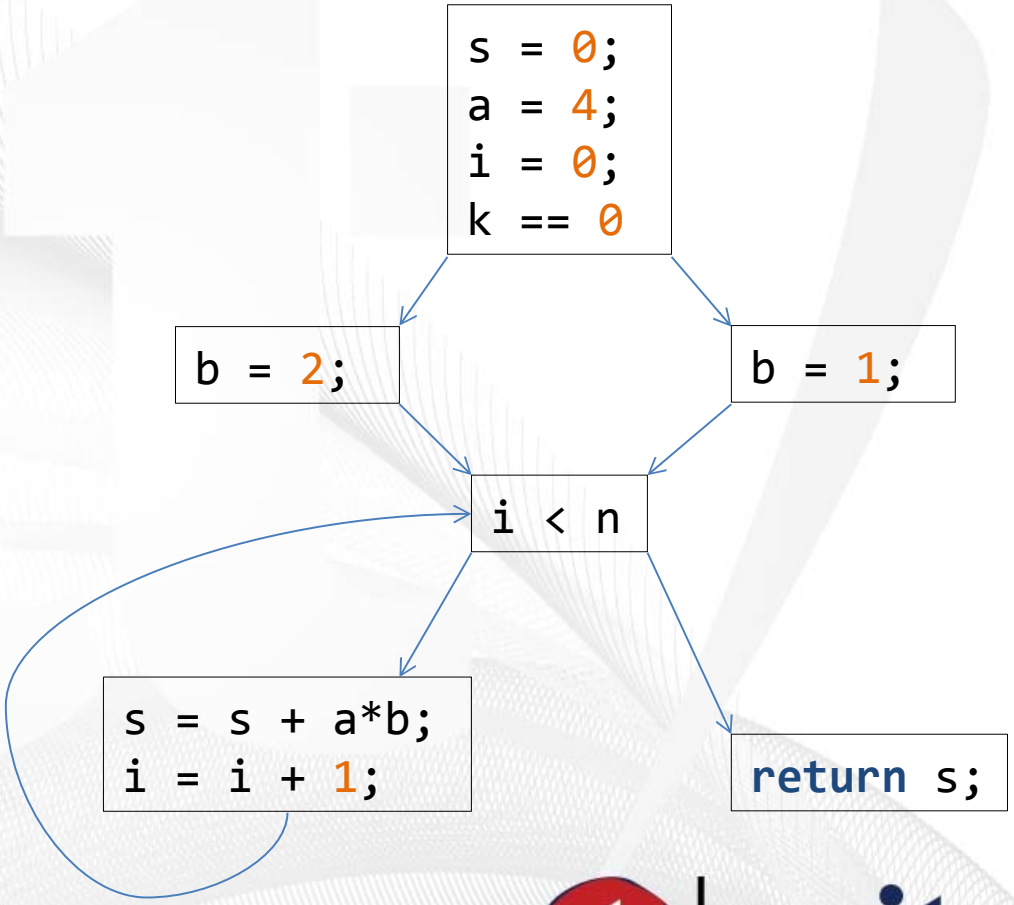
Iván de Jesús Deras Tábora

Outline

- Punto de Inicio: AST
- Punto Intermedio: CFG
- Punto Final: Código Ensamblador

Control Flow Graph (CFG)

```
int add(n, k) {
    s = 0; a = 4; i = 0;
    if (k == 0)
        b = 1;
    else
        b = 2;
    while (i < n) {
        s = s + a*b;
        i = i + 1;
    }
    return s;
}
```



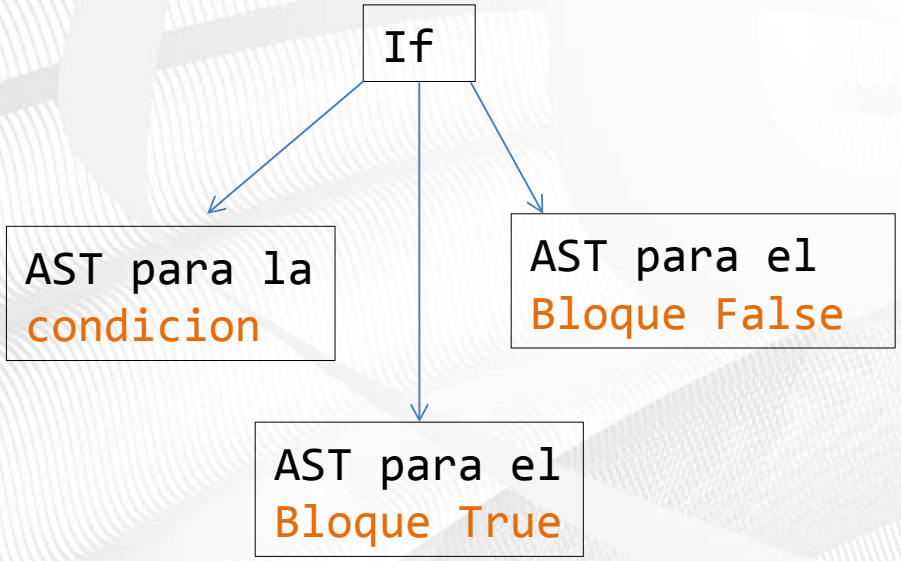
Control Flow Graph (CFG)

- Los Nodos representan cálculos
  - Cada nodo es un bloque básico
  - Un bloque básico es una secuencia de instrucciones
    - No contiene saltos desde la mitad del bloque básico hacia afuera
    - No hay saltos hacia algún punto intermedio del bloque básico
    - Un bloque básico debe ser Maximal
  - La ejecución del bloque básico comienza con la primera instrucción.
- Las aristas representan flujo de control

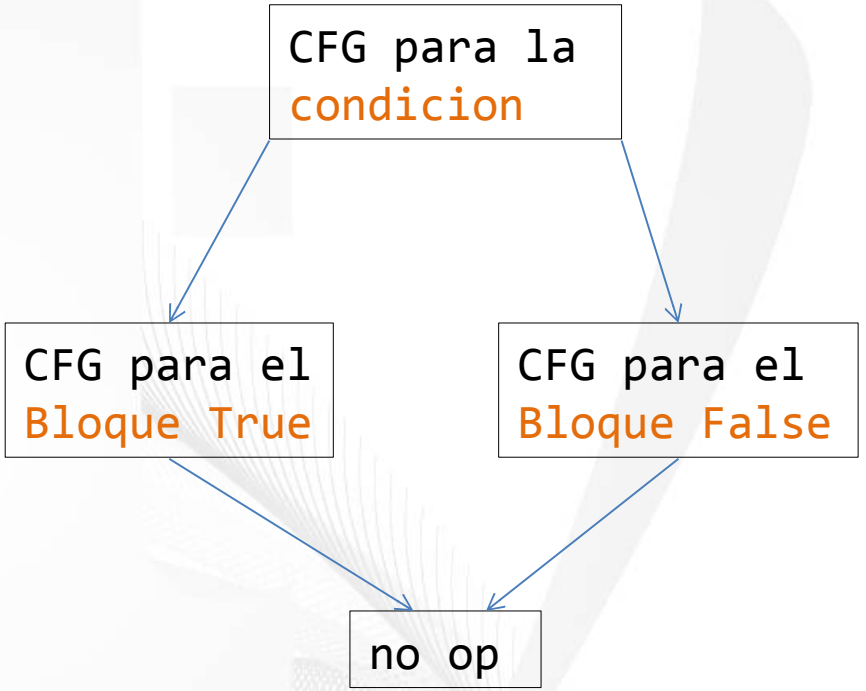
AST a CFG para If Then Else

```
Source Code
if (condicion) {
    Bloque True
} else {
    Bloque False
}
```

AST

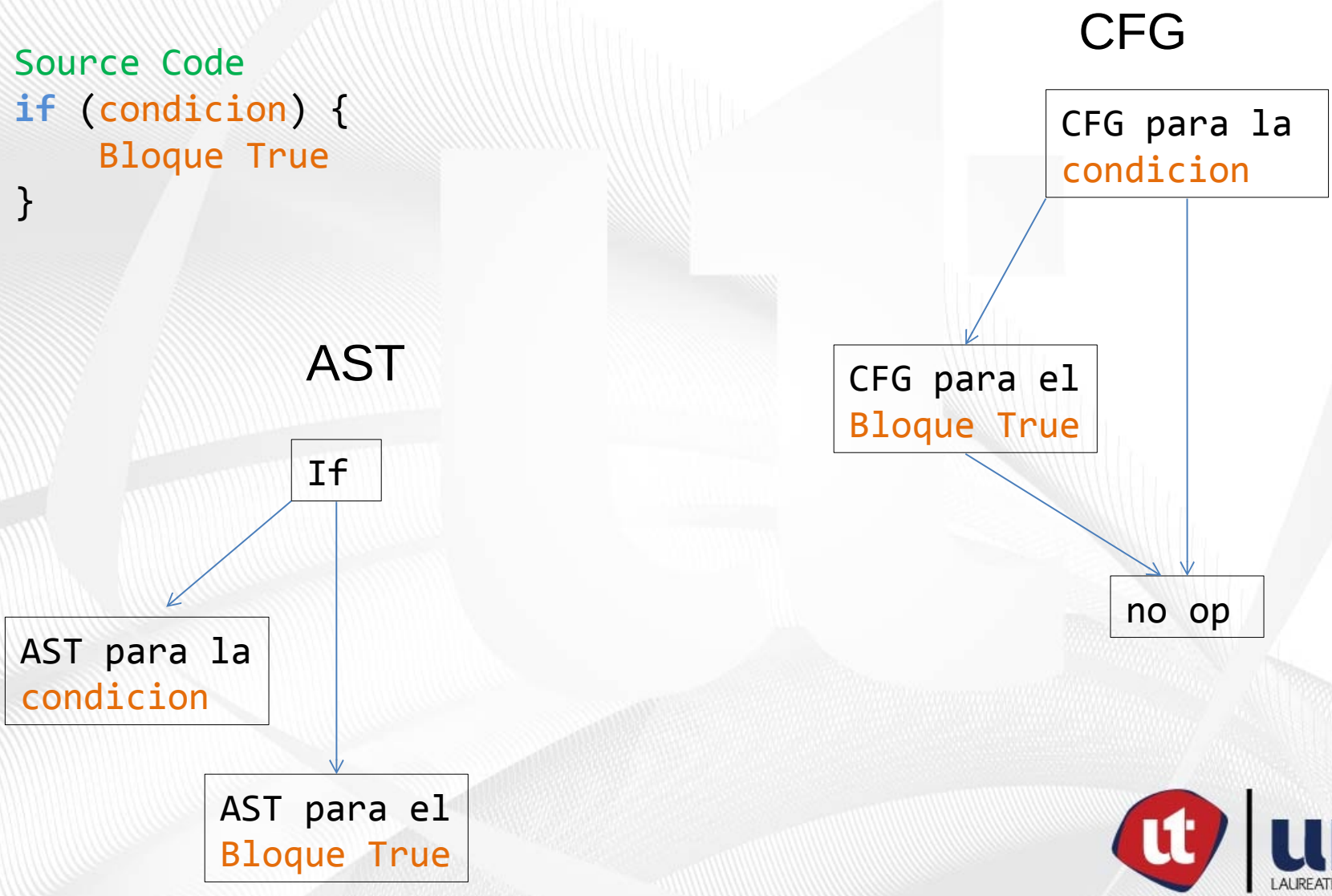


CFG



AST a CFG para If Then

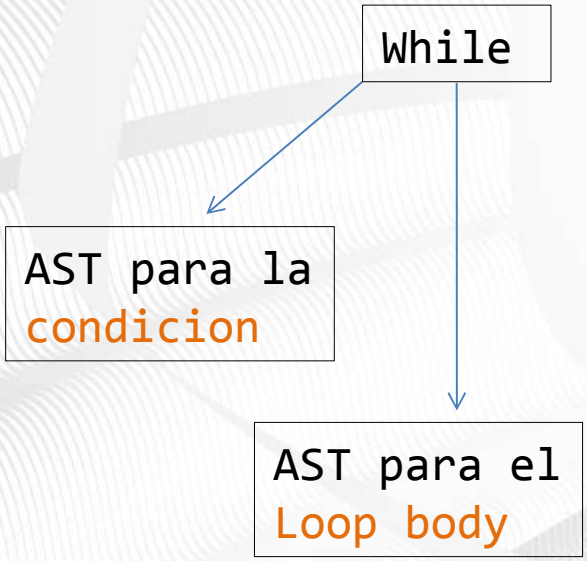
```
Source Code
if (condicion) {
    Bloque True
}
```



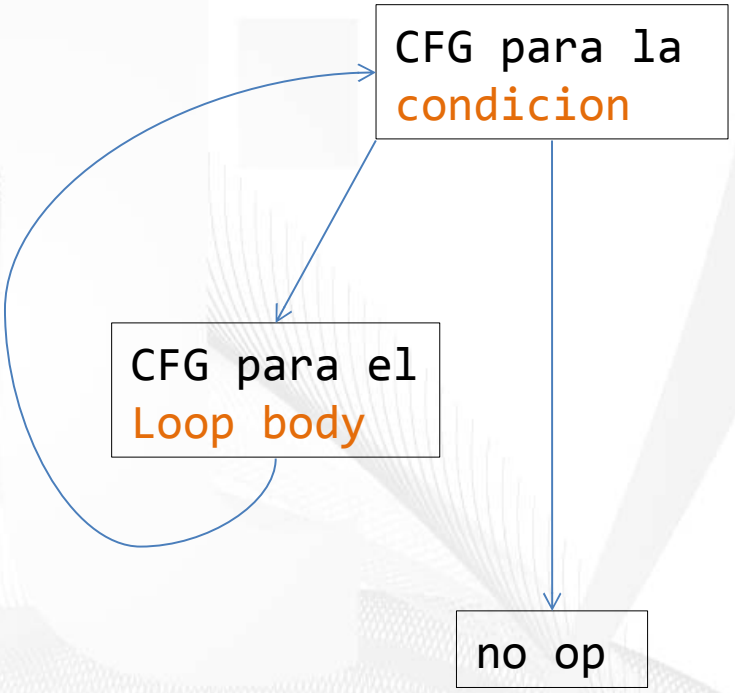
AST a CFG para **While**

```
Source Code
while (condicion) {
    Loop body
}
```

AST

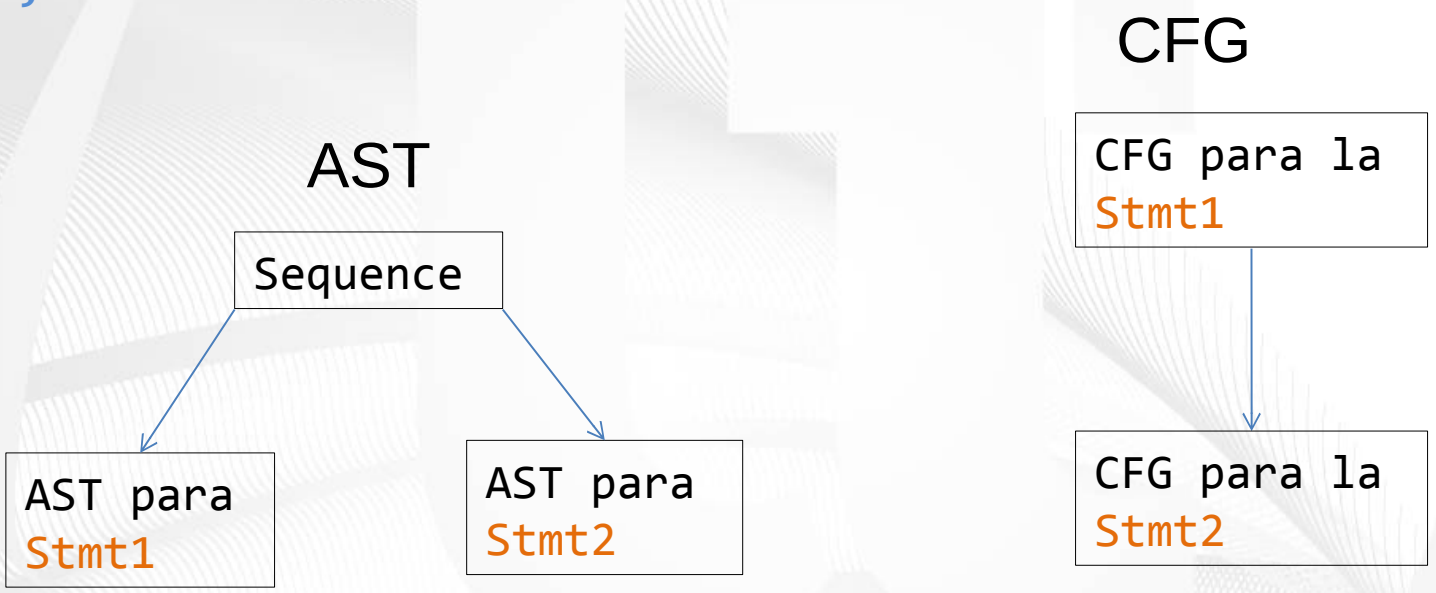


CFG



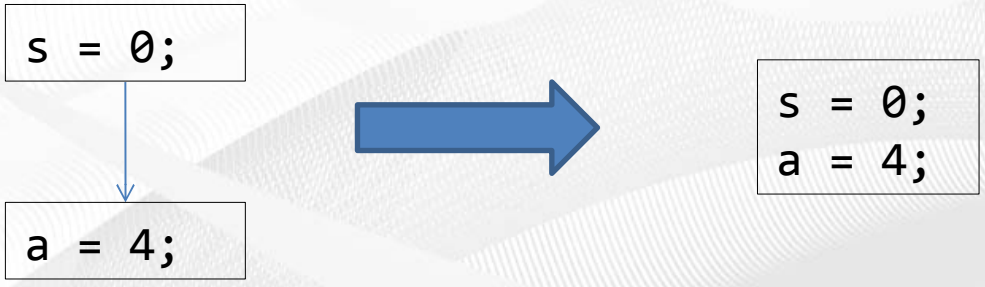
AST a CFG para **Sentencias**

```
Source Code
Stmt1;
Stmt2;
```

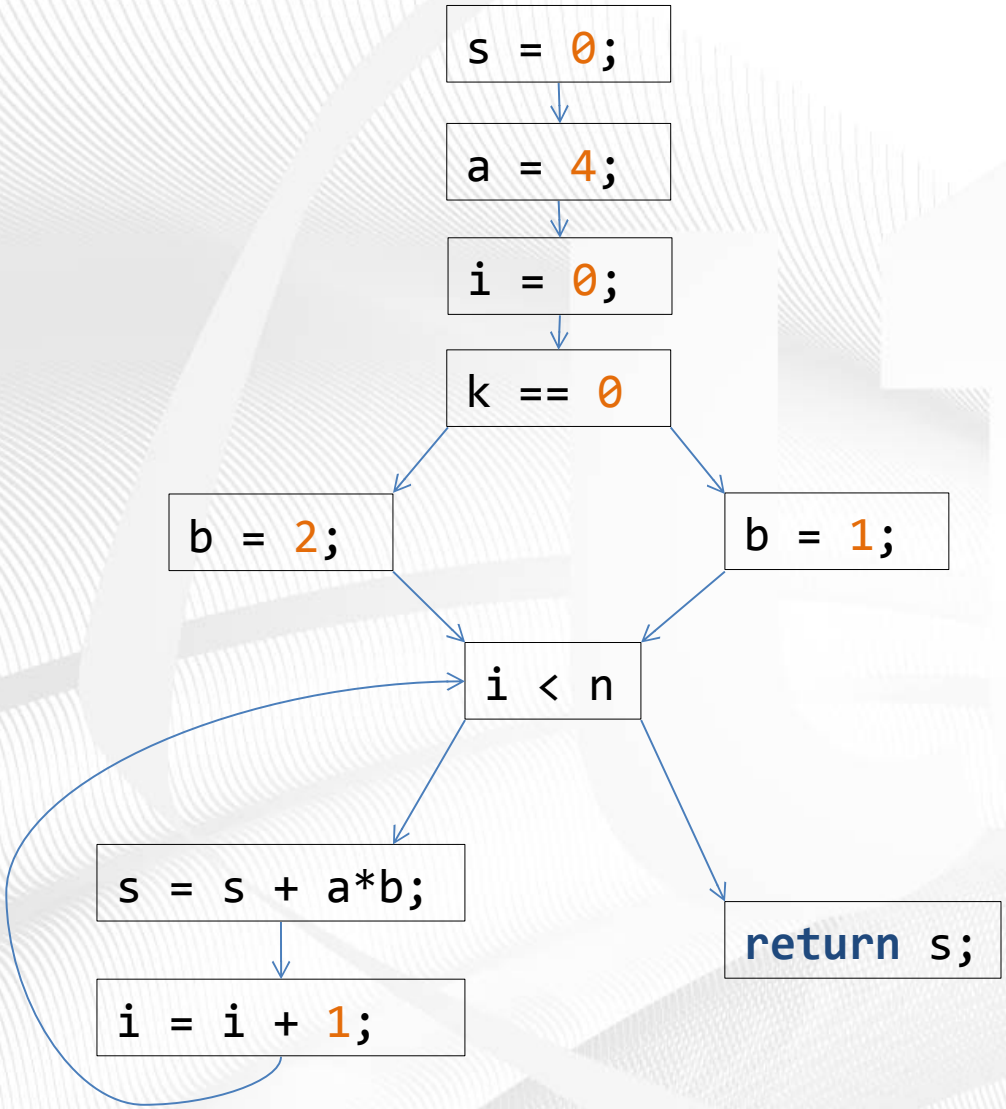


Construcción de Bloques Básicos

- Iniciar con el CFG
  - Visitar todos los nodos en el grafo
  - Combinar nodos adyacentes si:
    - Existe unicamente una arista de salida desde el primer
    - Existe unicamente una arista de llegada al segundo nodo



Construcción de Bloques Básicos: Ejemplo



Puntos de Programa, División y Unión

- Hay un punto de programa antes y después de cada instrucción.
- Un punto de división tiene multiples sucesores. Los saltos condicionales contienen solo puntos de división.
- Los puntos de unión tienen multiples predecesores
- Cada **Bloque Básico**:
  - Comienza con un punto de unión o su predecesor termina en un punto de división
  - Termina con un punto de división o su sucesor comienza con un punto de unión

Evaluación en corto circuito de expresiones condicionales

Considere el siguiente programa

```
int i = 0;
while (i < n && a[i] != 0) {
    i = i + 1;
}
```

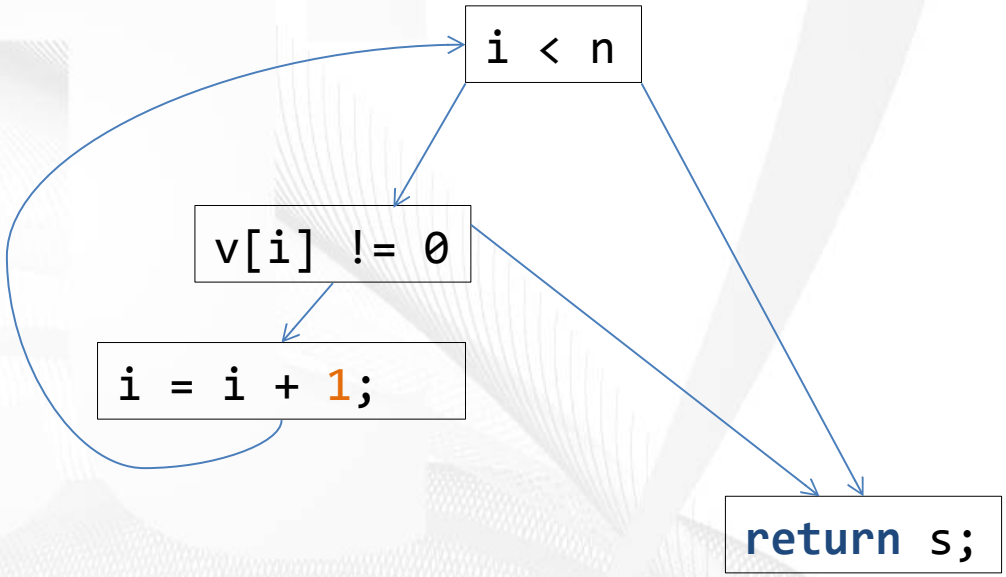
Si i < n es falso, deberiamos evaluar a[i] != 0?

Evaluación en corto circuito de expresiones condicionales

- En un programa, las condiciones se escriben como expresiones booleanas
  - `((i < n) && (v[i] != 0)) || i > k)`
- La semántica dicta que deberiamos evaluar solo lo necesario para determinar el resultado de la condición
  - Evaluate `(v[i] != 0)` si y solo si `(i < n)` es verdadero
  - Evaluate `i > k` si y solo si `((i < n) && (v[i] != 0))` es falso
- Utilizaremos el CFG para representar esta evaluación en corto-circuito.

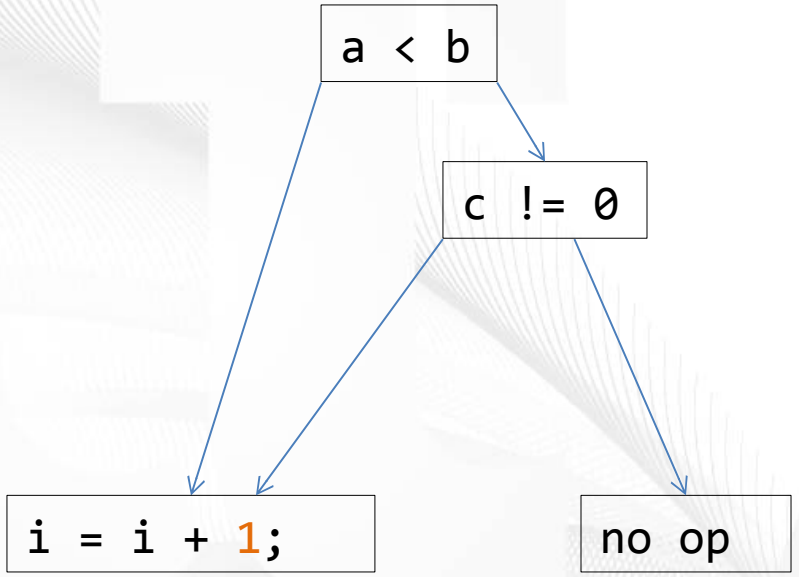
Evaluación en corto circuito de expresiones condicionales

```
while (i < n && v[i] != 0) {
    i = i+1;
}
```



Evaluación en corto circuito de expresiones condicionales

```
if (a < b || c != 0) {
    i = i+1;
}
```



AST a CFG: Implementación

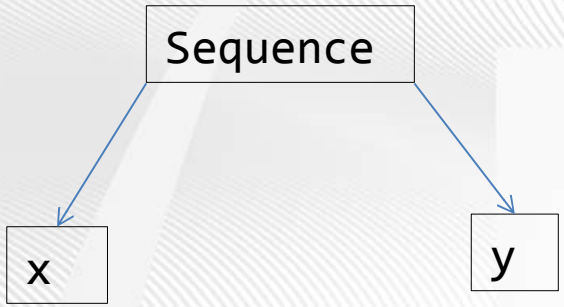
AST2CFG(n)
Genera la representación en CFG del node **n** del AST. Retorna **(b, e). b** es el nodo de inicio y **e** es el nodo final en el CFG. Esta operación aplica a sentencias unicamente.

ShortCircuit(**c**, t, f)
Genera la representación en corto-circuito de **c** en el CFG.
- Si **c** es verdadera, el control se transfiere al nodo **t**
- Si **c** es falsa, el control se transfiere al nodo **f**
- Retorn **b**, el nodo inicial de la condición

CBRNode(**c**, t, f)
- Crea un nodo para saltos condicionales.
- Si **c** es verdadera, el control se transfiere al nodo **t**
- Si **c** es falsa, el control se transfiere al nodo **f**
- Retorn **b**, el nodo inicial de la condición

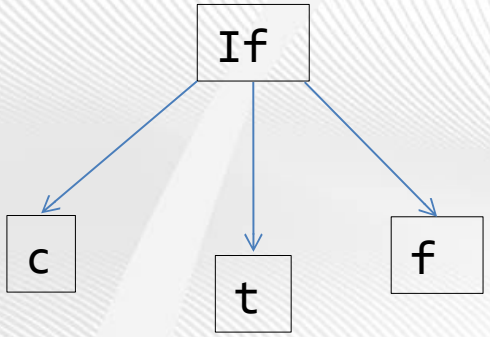
NopNode()
Crear un nodo que representa una operación nula (no operation)

AST a CFG: Secuencias (`seq x y`)



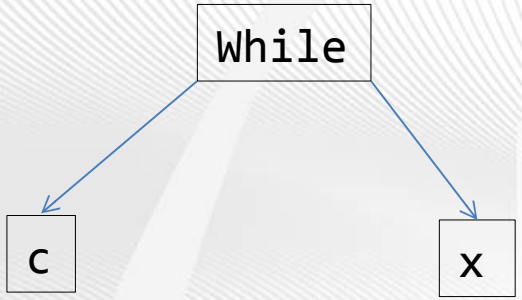

```
(bx, ex) = AST2CFG(x);
(by, ey) = AST2CFG(y);
next(ex) = by;
return (bx, by);
```

AST a CFG: Sentencia **If**



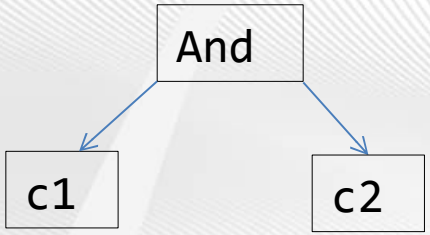

```
(bt, et) = AST2CFG(t);
(bf, ef) = AST2CFG(f);
e = NopNode();
next(et) = e;
next(ef) = e;
bc = ShorCircuit(c, bt, bf);
return (bc, e);
```

AST a CFG: Sentencia **While**



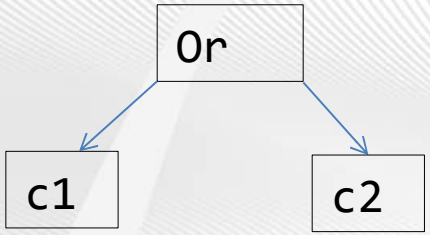

```
(bx, ex) = AST2CFG(x);
e = NopNode();
bc = ShorCircuit(c, bx, e);
next(ex) = bc;
return (bc, e);
```

AST a CFG: Corto Circuito **And**



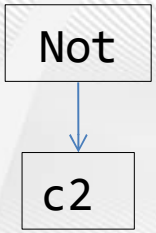

```
ShortCircuit(c, t, f) {
  b2 = ShortCircuit(c2, t, f);
  b1 = ShortCircuit(c1, b2, f);
  return b1;
}
```

AST a CFG: Corto Circuito **Or**



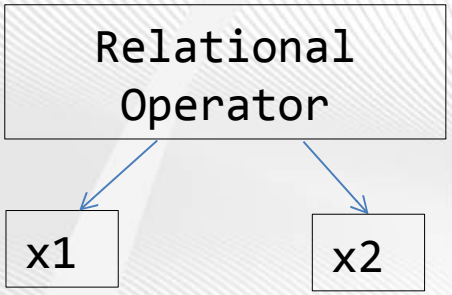

```
ShortCircuit(c, t, f) {
  b2 = ShortCircuit(c2, t, f);
  b1 = ShortCircuit(c1, t, b2);
  return b1;
}
```

AST a CFG: Corto Circuito **Not**



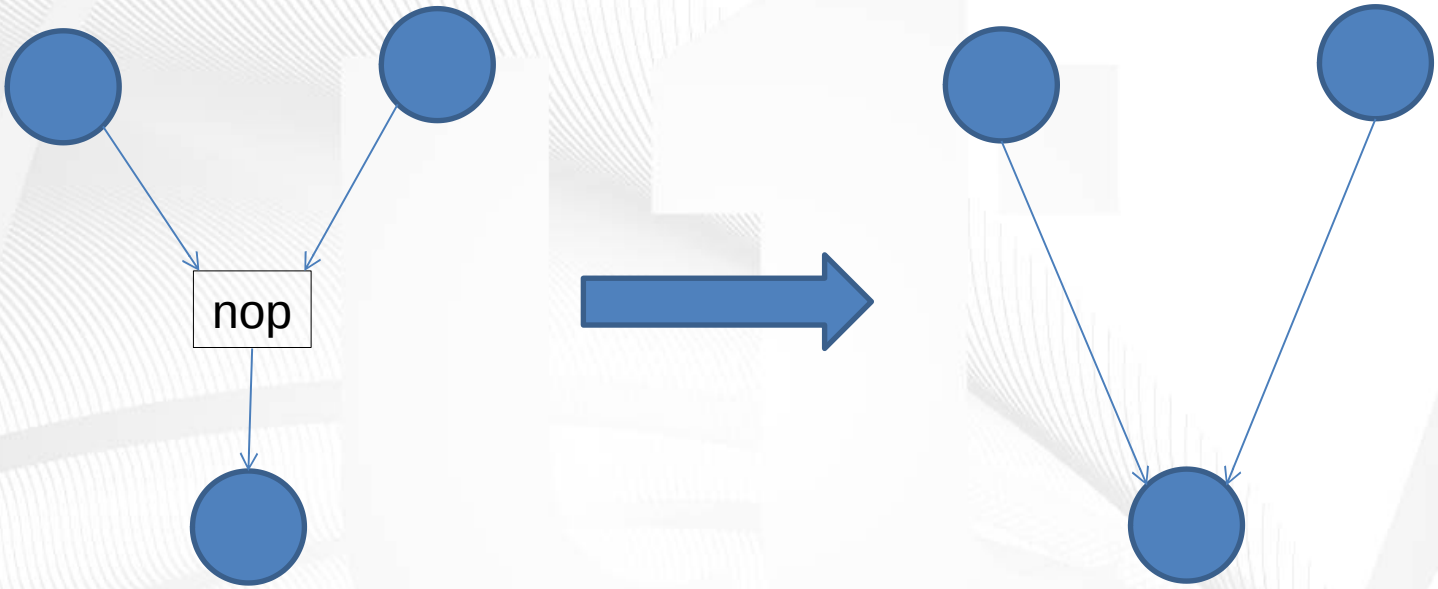

```
ShortCircuit(c, t, f) {
  b = ShortCircuit(c, f, t);

  return b;
}
```

AST a CFG: Corto Circuito **Operaciones Relacionales**




```
ShortCircuit(c, t, f) {
  b = CBRNode(c, f, t);
  return b;
}
```

Eliminando **nops**



Generando ensamblador a partir del CFG

- Generar etiquetas para las aristas destinos en los saltos
- Generar código para el prólogo del procedimiento
- Generar código para los bloques básicos
- Generar código para el epilogo del procedimiento