CÓRDOBA UNIVERSITY

SUPERIOR POLYTECHNIC SCHOOL

DEPARTMENT OF
COMPUTER SCIENCE AND NUMERICAL ANALYSIS

ARTIFICIAL INTELLIGENCE LANGUAGES

TECHNICAL ENGINEERING IN MANAGEMENT COMPUTER SCIENCE

TECHNICAL ENGINEERING IN SYSTEMS COMPUTER SCIENCE

SECOND YEAR

FIRST FOUR-MONTH PERIOD

ACADEMIC YEAR: 2009 - 2010

First part: Scheme

Second part: Prolog

First part: Scheme

Contents

Contents

1. *Fundamental Characteristics of Functional Programming*

- ✓ ***Functional*** *Programming is a* ***subtype*** *of* ***Declarative*** *Programming*

1. Fundamental Characteristics of Functional Programming

- ✓ Declarative Programming (1 / 2)
  - ➢ Objective: Problem description

"What" problem must be resolved?

- ▪ Notice:
  - *It does **not** mind "**how**" the problem is resolved*
  - *It **avoids** the implementation features.*

1. Fundamental Characteristics of Functional Programming

✓ Declarative Programming (2 / 2)

- Features
  - Expressivity
  - Extensible: 10% - 90% rule
  - Protection
  - Mathematic Elegance
- Types:
  - **Functional** or Applicative Programming:
    - Lisp, **Scheme**, Haskell, …
  - **Logic** Programming: **Prolog**

1. *Fundamental Characteristics of Functional Programming*

- ✓ ***Principle*** *of the* ***"Pure" Functional*** *Programming*

  "The ***expression value*** *only* ***depends on*** *its* ***sub-expressions*** *values,*

  if such sub-expressions exist ".

- ✓ ***Non collateral effects***

  The value of "a + b" ***only*** *depends on "a" and "b".*

- ✓ *The* ***function*** *term is used in its* ***mathematical*** *sense.*

- ✓ ***No instructions:*** *programming* ***without*** *assignments*

  - ➢ *The* ***impure*** *Functional programming* ***allows*** *the*

    "assignment instruction"

1. Fundamental Characteristics of Functional Programming

- ✓ Program structure in Functional Programming
  - ➢ The program is a function composed of simpler functions
  - ➢ Function execution:
    - ▪ Receives the input data: functions arguments or parameters
    - ▪ Evaluates the expressions
    - ▪ Returns the Result: computed value of the function

1. Fundamental Characteristics of Functional Programming

- ✓ Type of Functional Languages
  - ➢ Most of them are **interpreted** languages
  - ➢ Some of them have **compiled** versions
- ✓ Memory management
  - ➢ Implicit memory management:
    - ▪ Memory management is a task of the interpreter.
    - ▪ The programmer must **not** worry about memory management.
  - ➢ Garbage collection: task of the interpreter.

In short: the programmer must only worry about the Problem description

Contents

2. *Historic Summary of Scheme*

- ✓ *LISP*
- ✓ *Compilation versus Interpretation*
- ✓ *Lexical (or static) versus dynamical scope*
- ✓ *Origin of Scheme*

2. Historic Summary of Scheme

- ✓ LISP
- ✓ Compilation versus Interpretation
- ✓ Lexical (or static) versus dynamical scope
- ✓ Origin of Scheme

2. Historic Summary of Scheme

- ✓ LISP
  - ➢ **John McCarthy** (MIT)
  - ➢ **"Advice Taker"** program:
    - ▪ Theoretical basis: Logic Mathematics
    - ▪ Objective: Deduction and Inferences
  - ➢ LISP: LISt Processing (1956 – 1958)
    - ▪ Second historic language of **Artificial Intelligence** (after IPL)
    - ▪ At present time, second historic language **in use** (after Fortran)
    - ▪ LISP is based on Lambda Calculus (**Alonzo Church**)
  - ➢ Scheme is a **dialect** of LISP

2. *Historic Summary of Scheme*

- ✓ *LISP*
  - ➢ *Functional Programming Characteristics*
    - *Recursion*
    - *Lists*
    - *Implicit memory management*
    - *Interactive and interpreted programs*
    - ***Symbolic** Programming*
    - *Dynamically **scoped** for **non** local variables*

2. Historic Summary of Scheme

- ✓ LISP
  - ➢ LISP's **contributions:**
    - Built – in functions
    - Garbage collection
    - Definition Formal Language: LISP itself

2. Historic Summary of Scheme

- ✓ LISP
  - ➢ Applications: Artificial Intelligence Programs
    - Theorem verification and testing
    - Symbolic differentiation and integration
    - Search Problems
    - Natural Language Processing
    - Computer Vision
    - Robotics
    - Knowledge Representation Systems
    - Expert Systems
    - And so on

2. Historic Summary of Scheme

- ✓ LISP
  - ➢ Dialects (1 /2)
    - ▪ **Mac LISP** (Man and computer or Machine – aided cognition): **East** Coast Version
    - ▪ **Inter LISP** (Interactive LISP): **West** Coast Version
      - Bolt, Beranek y Newman Company (BBN)
      - Research Center of Xerox at Palo Alto (Texas)
      - **LISP Machine**

2. *Historic Summary of Scheme*

- ✓ *LISP*
  - ➢ ***Dialects*** *(2 / 2)*
    - ▪ ***Mac LISP*** *(Man and computer or Machine – aided cognition): East Coast Version*
      - *C-LISP: Massachusetts University*
      - *Franz LISP: California University (Berkeley).* ***Compiled version.***
      - *NIL (New implementation of LISP): MIT.*
      - *PSL (Portable Standard LISP): Utah University*
      - ***Scheme****: MIT.*
      - *T (True):Yale University.*
      - *Common LISP*

2. Historic Summary of Scheme

- ✓ LISP
- ✓ Compilation versus Interpretation
- ✓ Lexical (or static) versus dynamical scope
- ✓ Origin of Scheme

2. *Historic Summary of Scheme*

- ✓ *Compilation versus interpretation*
  - ➢ *Compilation:*
    - ▪ *The source code (**high level**) is transformed into executable code (**low level**), which can be independently run.*

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Compilation

Source code → $\boxed{\text{Compiler}}$

2. *Historic Summary of Scheme*

- ✓ *Compilation versus interpretation*
  - ➢ *Compilation*

$$\begin{tikzcd}
\text{Source code} \arrow[r] & \boxed{\text{Compiler}} \arrow[d] \\
 & \text{Compilation errors}
\end{tikzcd}$$

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Compilation

$$\text{Source code} \rightarrow \boxed{\text{Compiler}} \rightarrow \text{Executable code}$$

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Compilation

$$\begin{tikzcd}
 & & \text{Input data} \arrow[d] \\
\text{Source code} \arrow[r] & \boxed{\text{Compiler}} \arrow[r] & \text{Executable code}
\end{tikzcd}$$

2. *Historic Summary of Scheme*

- ✓ *Compilation versus interpretation*
  - ➢ *Compilation*

$$\begin{tikzcd}
 & & \text{Input data} \arrow[d] \\
\text{Source code} \arrow[r] & \boxed{\text{Compiler}} \arrow[r] \arrow[d] & \text{Executable code} \arrow[d] \\
 & \text{Execution errors} & \text{Output}
\end{tikzcd}$$

2. Historic Summary of Scheme

✓ Compilation versus interpretation

➢ Compilation

$$\begin{tikzcd}
 & & \text{Input data} \arrow[d] \\
\text{Source code} \arrow[r] & \boxed{\text{Compiler}} \arrow[r] & \text{Executable code} \arrow[d] \\
 & & \text{Output}
\end{tikzcd}$$

2. *Historic Summary of Scheme*

- ✓ *Compilation versus interpretation*
  - ➢ *Interpretation*

2. *Historic Summary of Scheme*

- ✓ *Compilation versus interpretation*
  - ➢ ***Interpretation*** *or* ***simulation****: consists of a cycle of three stages*

2. *Historic Summary of Scheme*

- ✓ *Compilation versus interpretation*
  - ➢ ***Interpretation** or simulation: consists of a cycle of three stages*
    1. *Analysis: the source code is analysed to determine the following correct sentence to be run.*

2. *Historic Summary of Scheme*

- ✓ *Compilation versus interpretation*
  - ➢ ***Interpretation*** *or simulation: consists of a cycle of three stages*
    1. ***Analysis****: the source code is analysed to determine the following correct sentence to be run.*
    2. ***Generation****: the sentence is transformed into executable code.*

2. *Historic Summary of Scheme*

- ✓ *Compilation versus interpretation*
  - ➢ ***Interpretation** or simulation: consists of a cycle of three stages*
    1. ***Analysis**: the source code is analysed to determine the following correct sentence to be run.*
    2. ***Generation**: the sentence is transformed into executable code.*
    3. ***Execution**: the executable code is run.*

2. Historic Summary of Scheme

✓ Compilation versus interpretation

➢ Interpretation

$$\begin{tikzcd} \text{Source code} \arrow[r] & \boxed{\text{Interpreter}} \end{tikzcd}$$

2. Historic Summary of Scheme

✓ Compilation versus interpretation

➢ Interpretation

$$\begin{tikzcd}
 & \text{Input data} \arrow[d] & \text{Interpretation errors} \\
\text{Source code} \arrow[r] & \boxed{\text{Interpreter}} \arrow[ur] &
\end{tikzcd}$$

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Interpretation

$$\begin{tikzcd}
 & \text{Input data} \arrow[d] & \\
\text{Source code} \arrow[r] & \boxed{\text{Interpreter}} \arrow[d] \arrow[dr] & \\
 & \text{Output} & \text{Execution errors}
\end{tikzcd}$$

2. Historic Summary of Scheme

- ✓ Compilation versus interpretation
  - ➢ Interpretation

$$\begin{tikzcd}
 & \text{Input data} \arrow[d] \\
\text{Source code} \arrow[r] & \boxed{\text{Interpreter}} \arrow[d] \\
 & \text{Output}
\end{tikzcd}$$

2. Historic Summary of Scheme

✓ Compilation versus interpretation

▪ Compilation	▪ Interpretation
- Independent	- Dependent
- Memory necessities	- No memory necessities
- Efficient	- Less efficient
- Global	- Local
- No interaction	- Interaction
- Closed code during execution	- Open code during execution

2. Historic Summary of Scheme

- ✓ LISP
- ✓ Compilation versus Interpretation
- ✓ Lexical (or static) versus dynamical scope
- ✓ Origin of Scheme

2. *Historic Summary of Scheme*

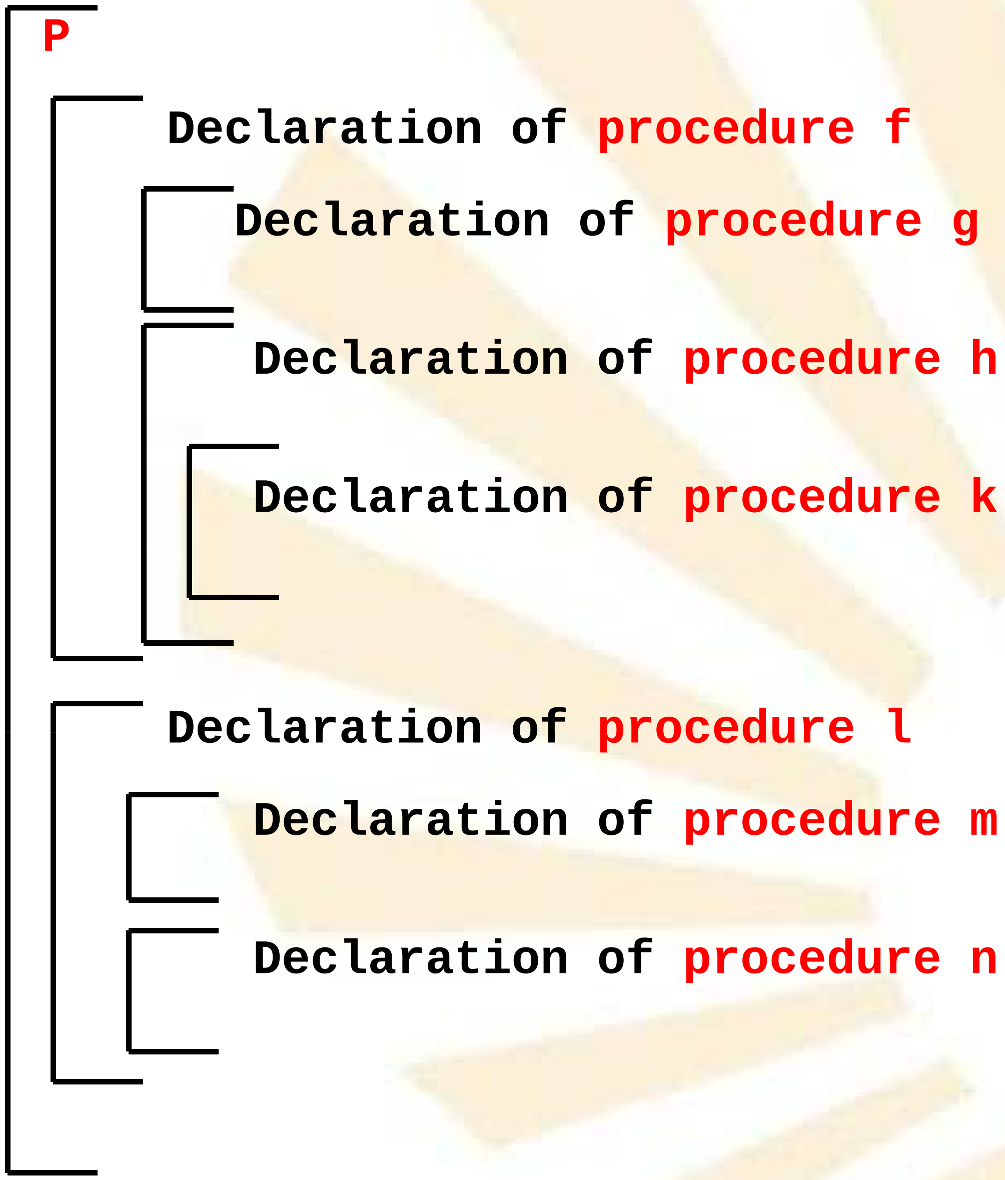
✓ *Lexical (or static) versus dynamical scope*

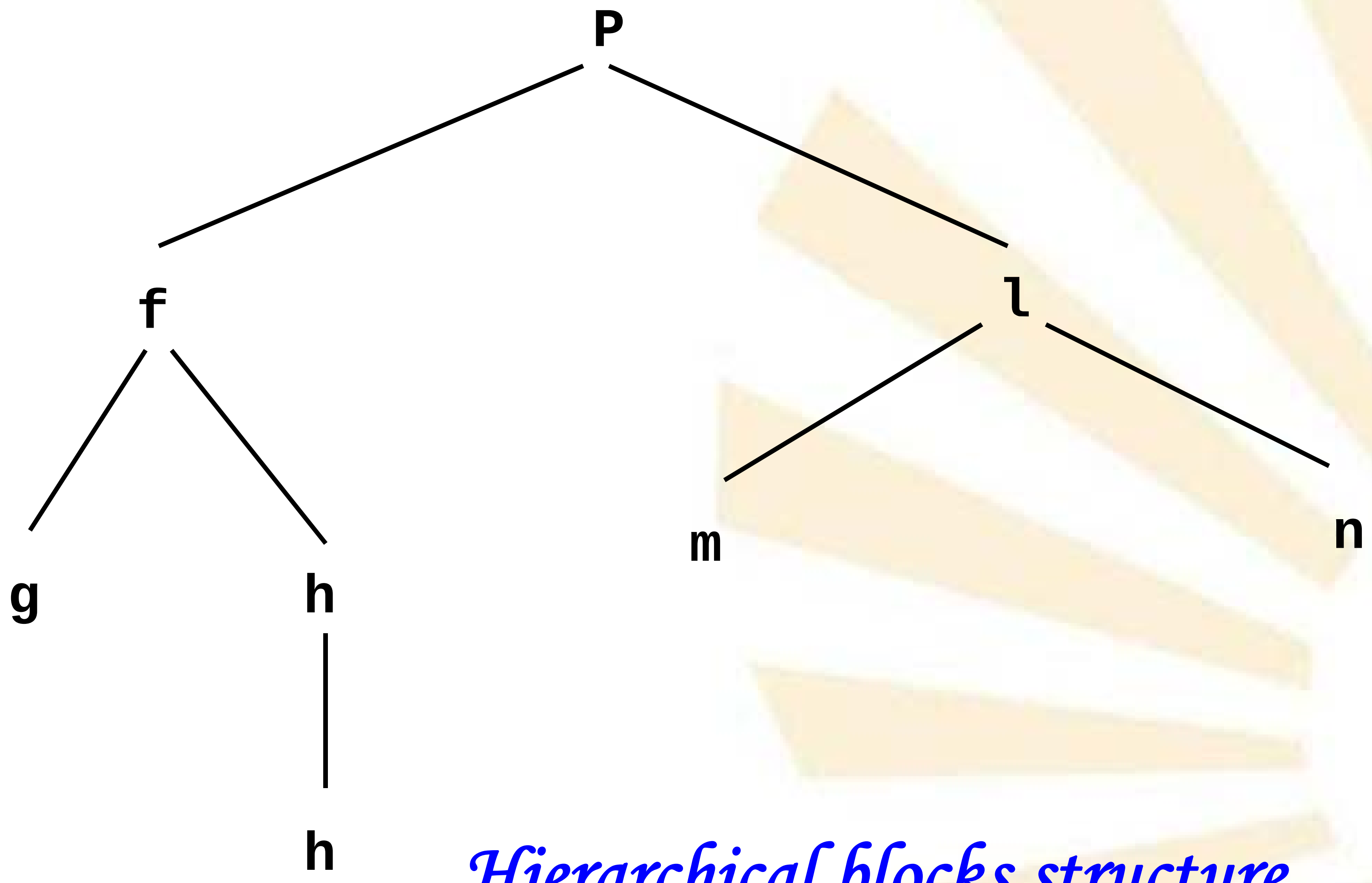
- *The **scope rules** determine the **declaration** of **non** local identifiers*
- ***Non** local identifiers:*
  - ***Variables** or **functions** which can be **used** in a function or procedure but are **not** declared in that function or procedure*
- ***Two types***
  - ***Lexical or static scope***
    - ***With** "blocks structure": Pascal, **Scheme***
    - ***Without** "blocks structure": C, Fortran*
  - ***Dynamical scope:***
    - ***Always with** "blocks structure": Lisp, SNOBOL, APL*

2. Historic Summary of Scheme

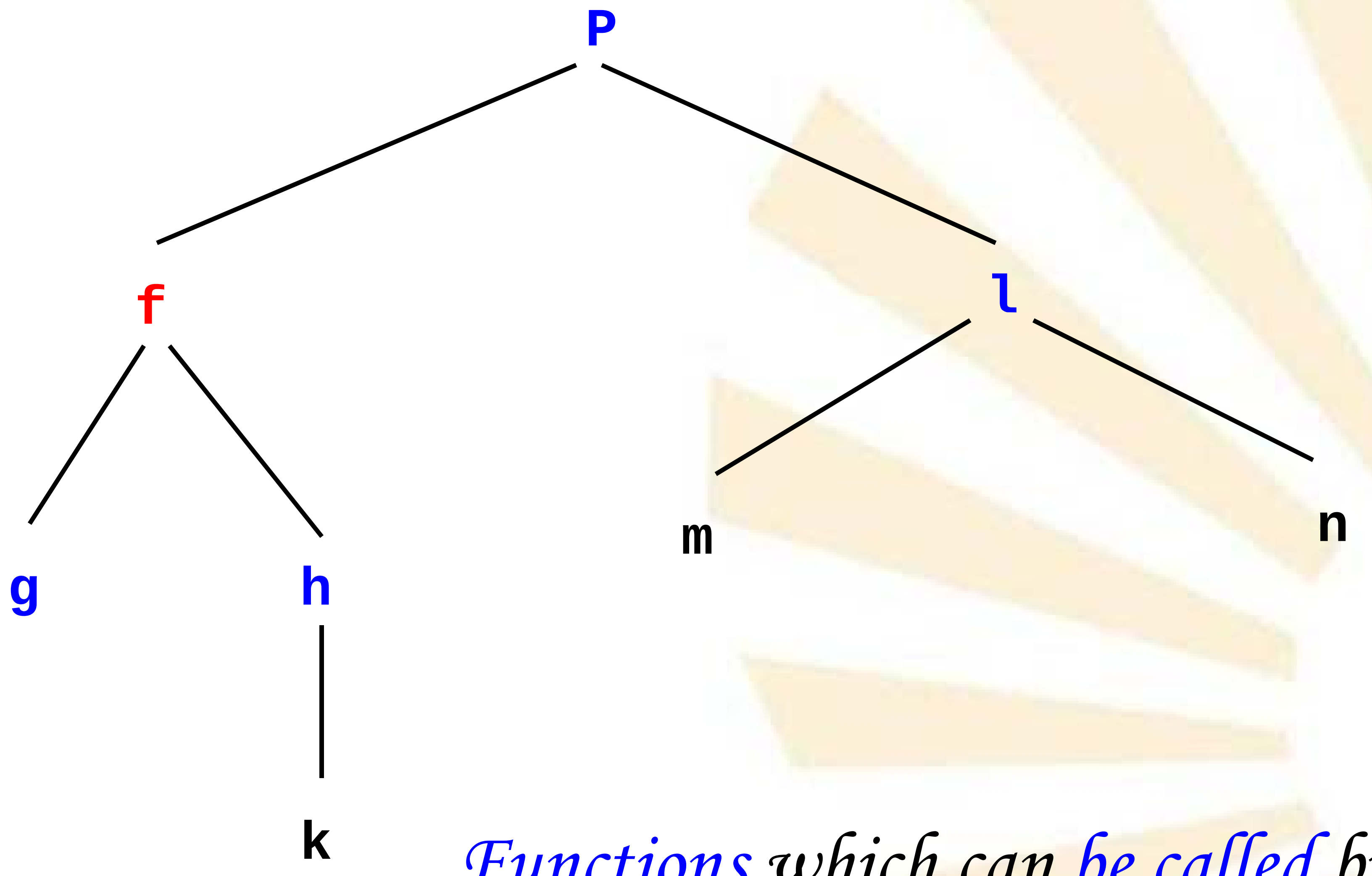
- ✓ Lexical (or static) versus dynamical scope
  - ➢ Block structure
    - ▪ A procedure or function can **call**
      - Itself
      - Its children (but **not** its grandchildren…)
      - Its brothers (but **not** its nephews)
      - Its father, grandfather, great-grandfather, …
      - The brothers of its father, grandfather, …
    - ▪ A procedure or function can **be called** by
      - Itself
      - Its father (but **not** by its grandfather, …)
      - Its children, grandchildren, great-grandchildren, …
      - Its brothers and their children, grandchildren, ...

Example of blocks structure

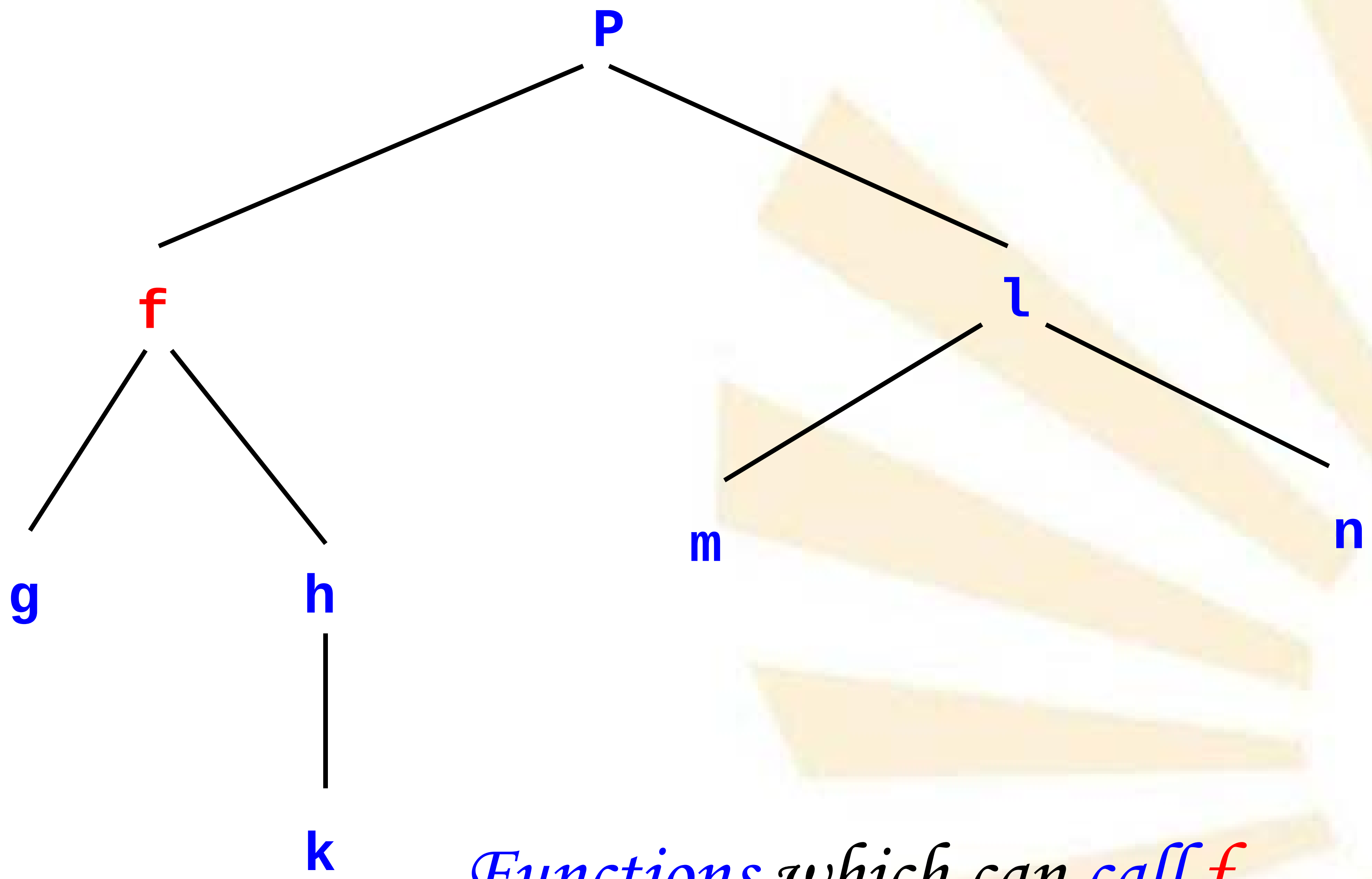

P
Declaration of procedure f
Declaration of procedure g
Declaration of procedure h
Declaration of procedure k
Declaration of procedure l
Declaration of procedure m
Declaration of procedure n



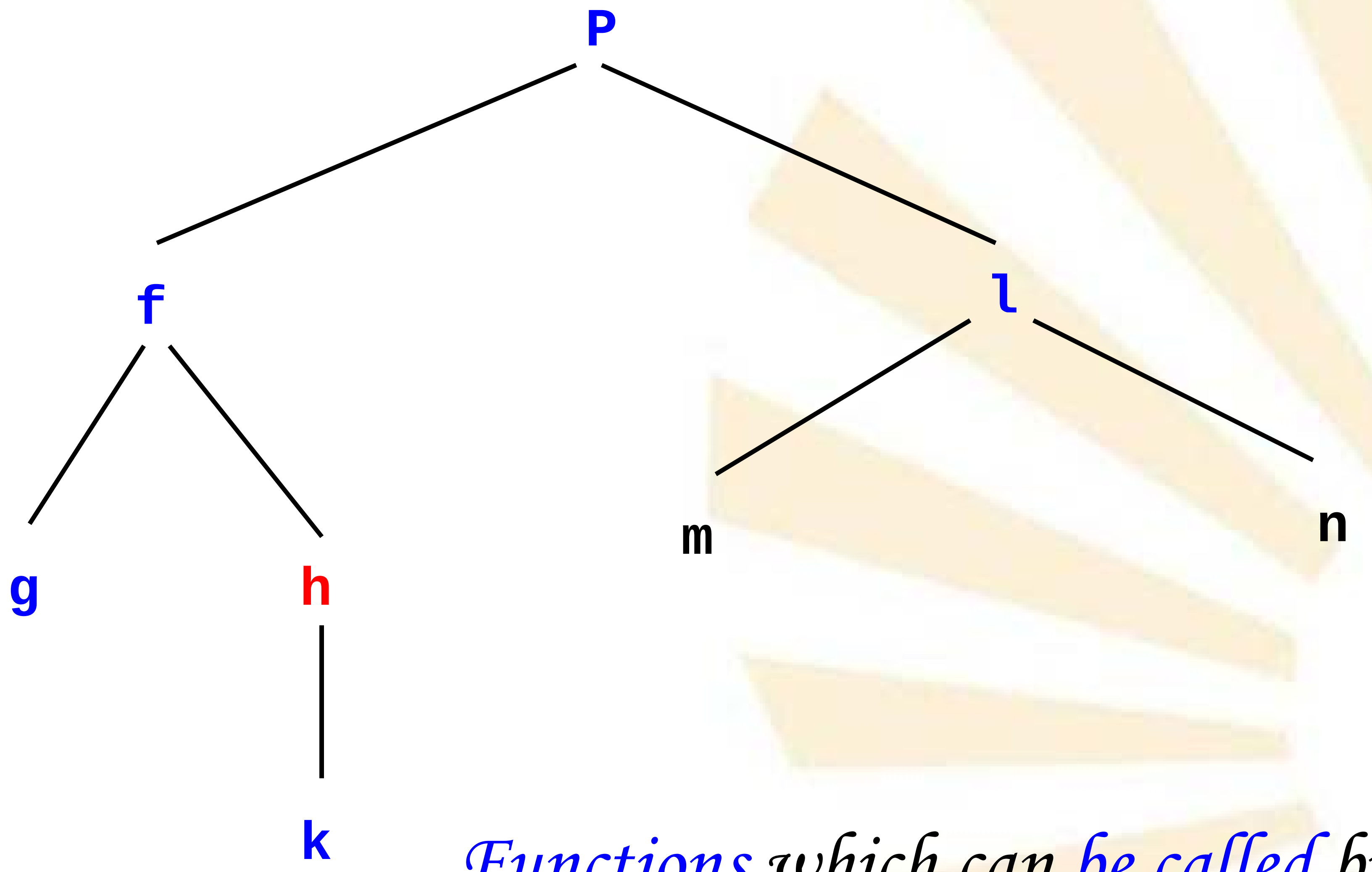
Hierarchical blocks structure



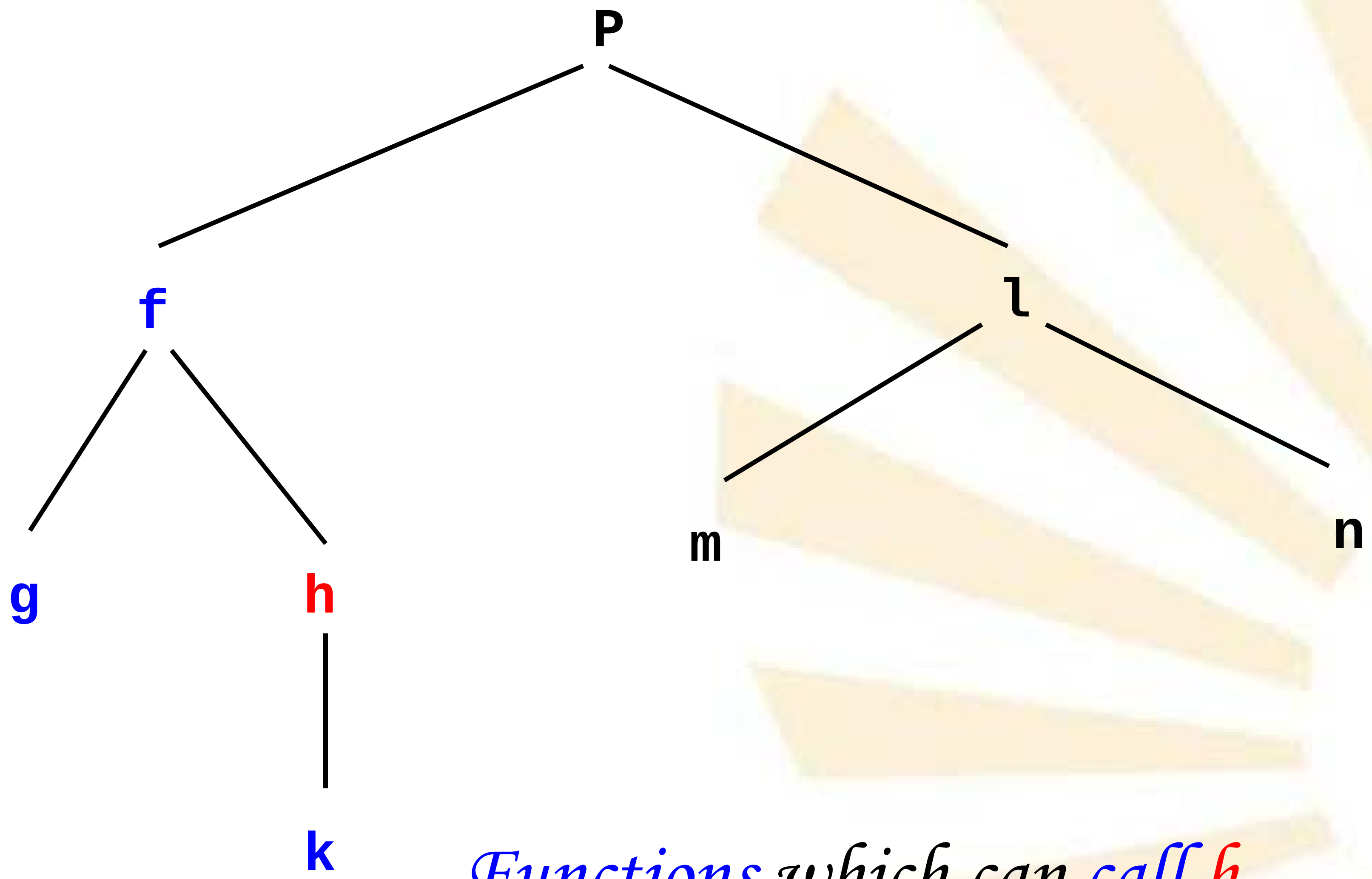
Functions which can be called by f



Functions which can call f



Functions which can be called by h



Functions which can call h

2. Historic Summary of Scheme

✓ Lexical (or static) versus dynamical scope

➢ Lexical or static scope

- The **declaration** of a **non** local identifier depends on the **closest lexical context:**

  You only have to ***read*** the program

  to determine the declaration of an identifier.

- **The closest nesting rules:**
  - The **scope** of a procedure (*) ***f*** includes the procedure ***f***.
  - If a **non** local identifier *x* is used in ***f*** then the declaration of *x* must be found in the **closest** procedure ***g*** which includes ***f***
  - **Notice** (*) : procedure, function or block

Example:

Lexical scope

with "block structure"

```
Declaration of procedure h
 Declaration of variable x   (x1)
 Declaration of variable y   (y1)
 Declaration of variable z   (z1)

 Declaration of procedure g
  Declaration of variable x (x2)
  Declaration of variable y (y2)

  Declaration of procedure f
    Declaration of variable x (x3)

    Use of x   (→ x3)
    Use of y   (→ y2)
    Use of z   (→ z1)

    Use of x   (→ x2)
    Use of y   (→ y2)
    Use of z   (→ z1)
    Call to f

    Use of x   (→ x1)
    Use of y   (→ y1)
    Use of z   (→ z1)
    Call to g
```

2. Historic Summary of Scheme

- ✓ Lexical (or static) versus dynamical scope
  - ➢ Lexical or static scope
    - ▪ Without block structure:
      - If x is **not** local for a **specific** function then it is **not** local for **all** functions

Global variables are not recommended

Example in C:

without

“block structure”

```
int x;   /* x1 */
int y;   /* y1 */
int z;   /* z1 */

main()
        {
    int x;   /* x2 */
    int y;   /* y2 */

    /* Use of x → x2 */
    /* Use of y → y2 */
    /* Use of z → z1 */
    /* Call to f */
    f ();
  }

 f()
  {
   int x;   /* x3 */
    /* Use of x → x3 */
    /* Use of y → y1 */
    /* Use of z → z1 */
    }
```

2. *Historic Summary of Scheme*

✓ *Lexical (or static) versus dynamical scope*

➢ *Dynamical* **scope:**

- *The* ***declaration*** *of an* ***identifier*** *depends on the* ***execution of the program***

  You have to ***run*** *the program*

  to determine the declaration of an identifier

- ***The closest activation rules:***

  - *The* ***scope*** *of a procedure (*)* ***f*** *includes the procedure* ***f****.*
  - *If a* ***non*** *local identifier x is used in the* ***activation*** *of* ***f*** *then the declaration of x must be found in the* ***closest active*** *procedure* ***g*** *with a declaration of x*
  - ***Notice*** *(*) : procedure, function or block*

2. *Historic Summary of Scheme*

- ✓ *Lexical (or static) versus dynamical scope*
  - ➢ *Notice:*
    - ▪ *The **dynamical scope** allows that an **identifier** can be associated to **different declarations** during the program execution*

Example:

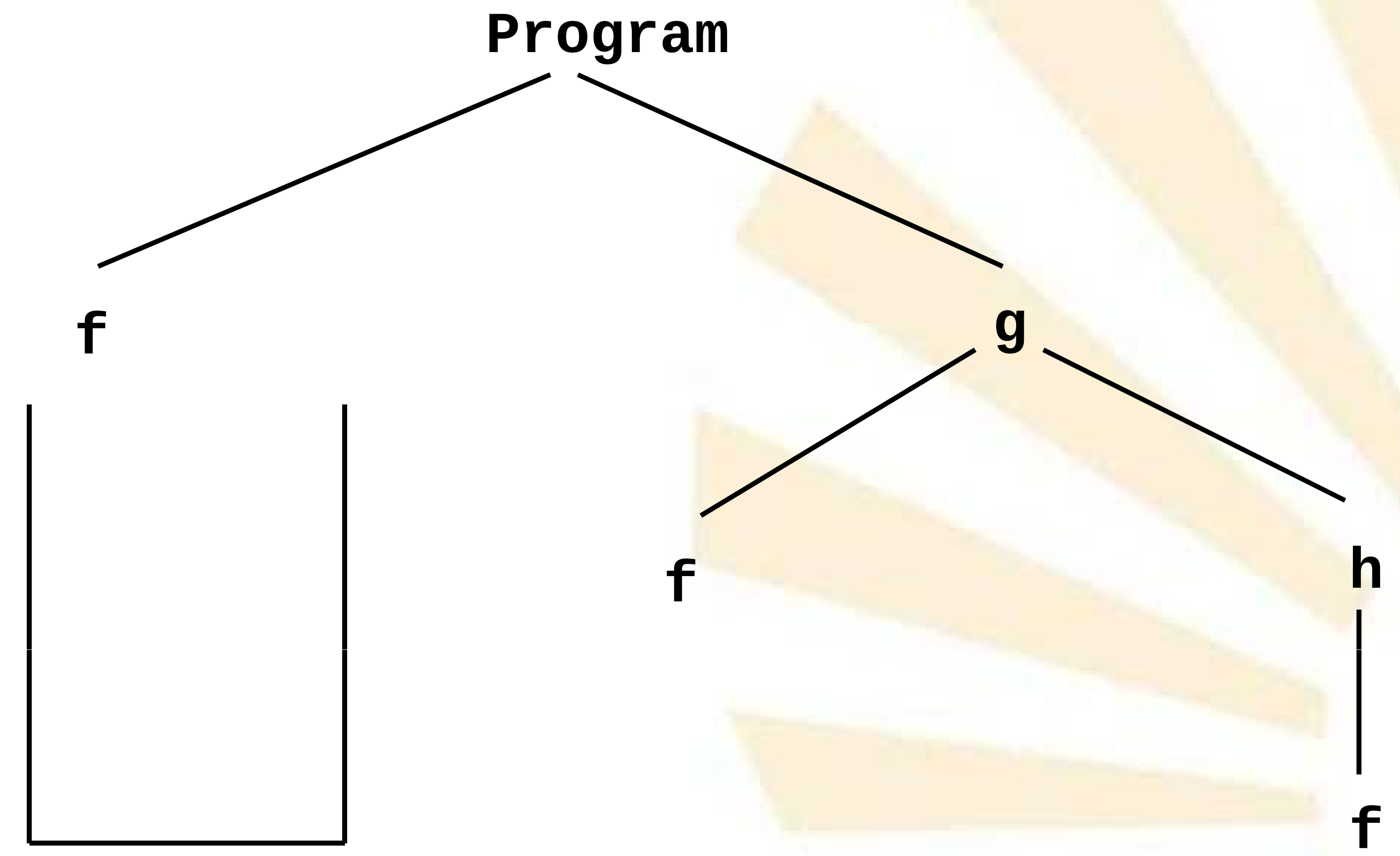
Lexical

versus

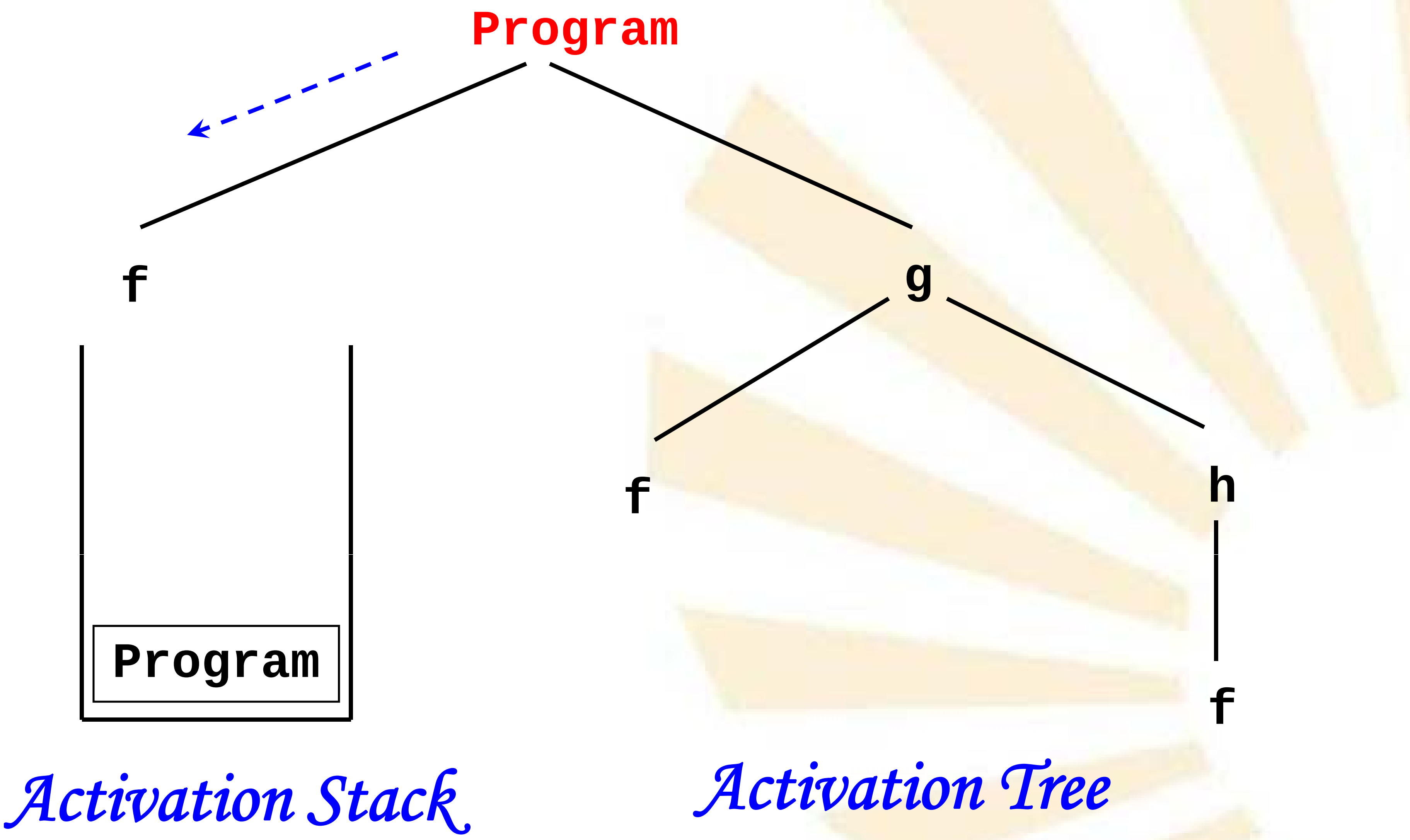
Dynamical

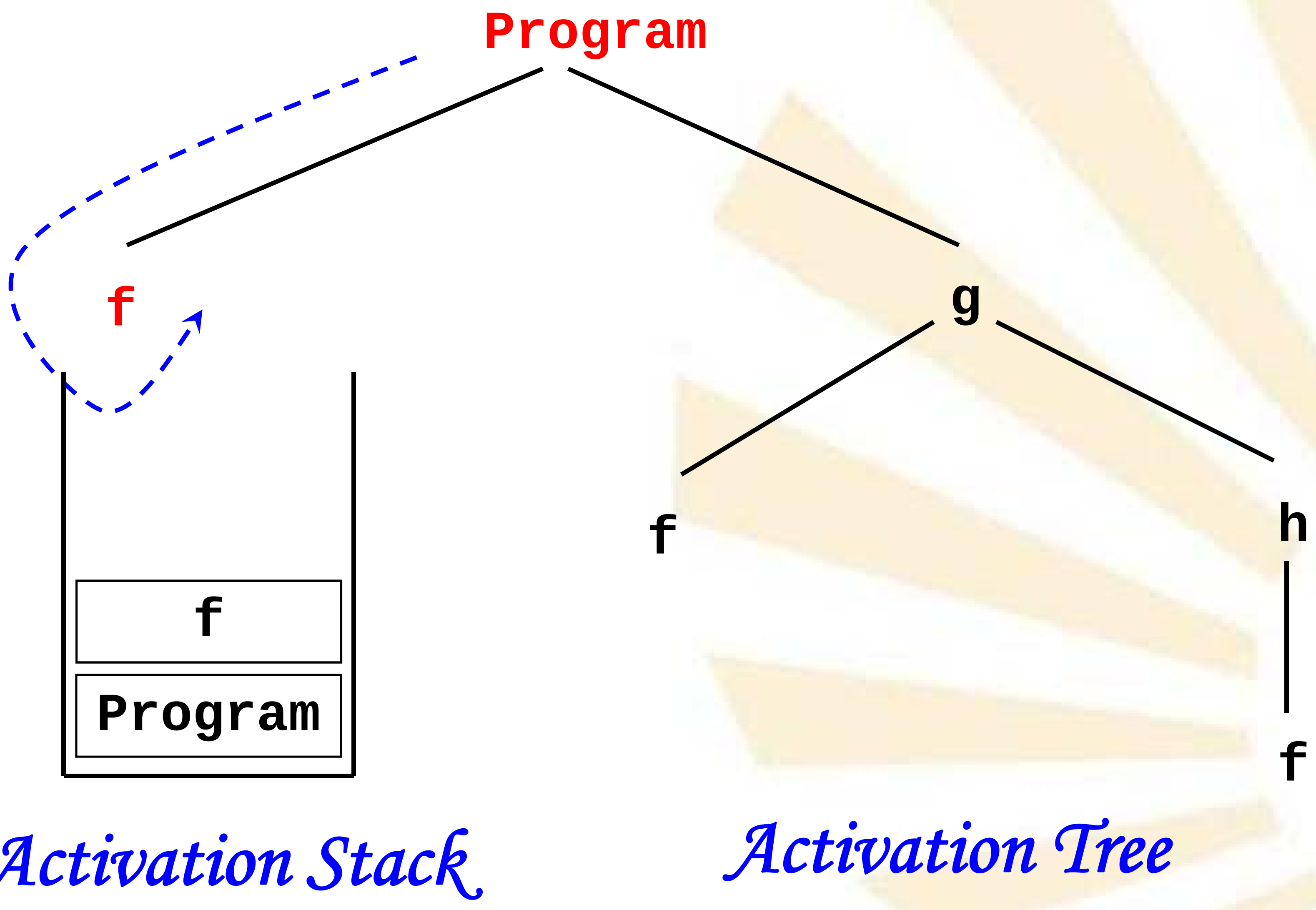
scope

```
Program
    Declaration of variable x

    Declaration of procedure f
     Use of x


    Declaration of procedure g
      Declaration of variable x

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

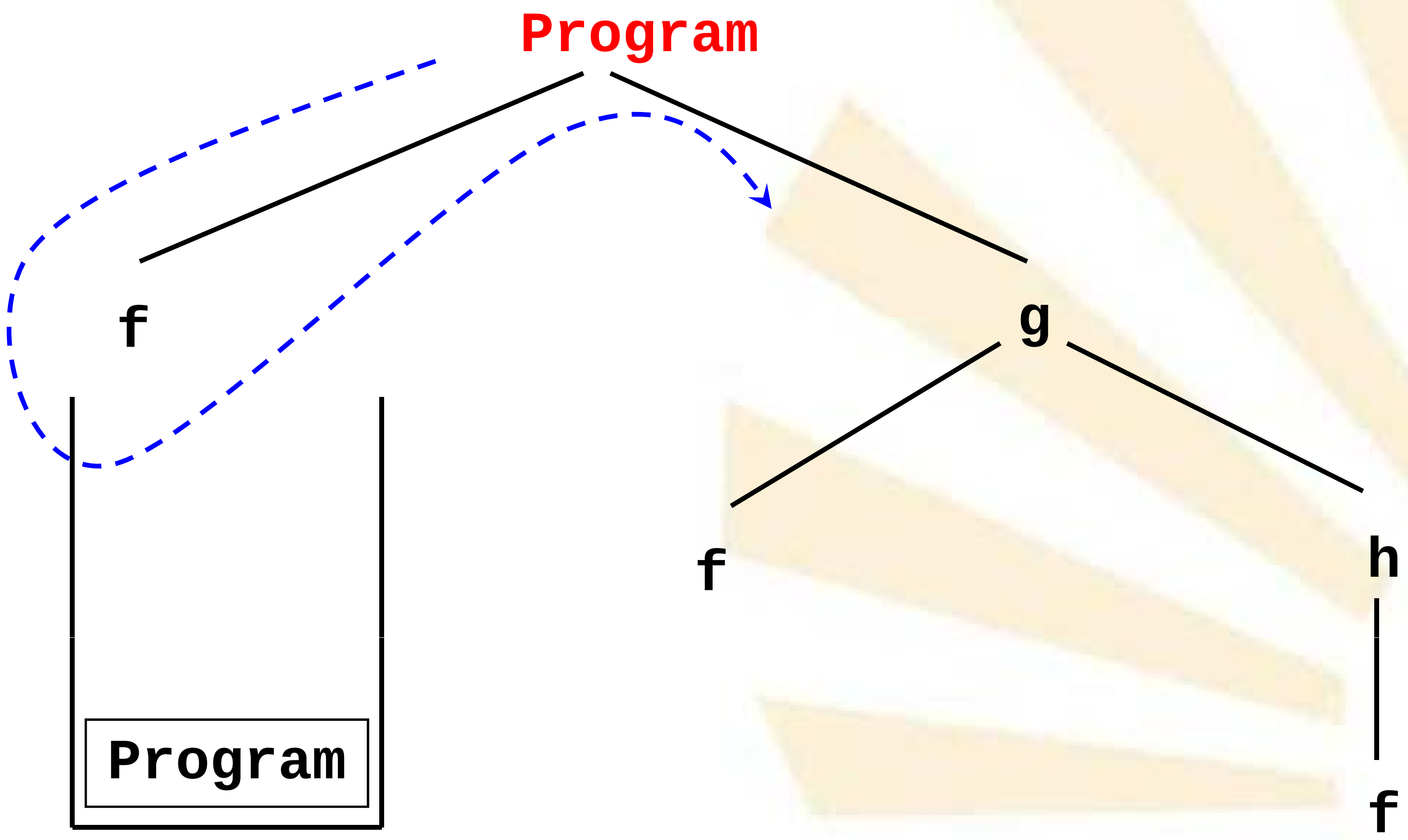


Activation Stack *Activation Tree*

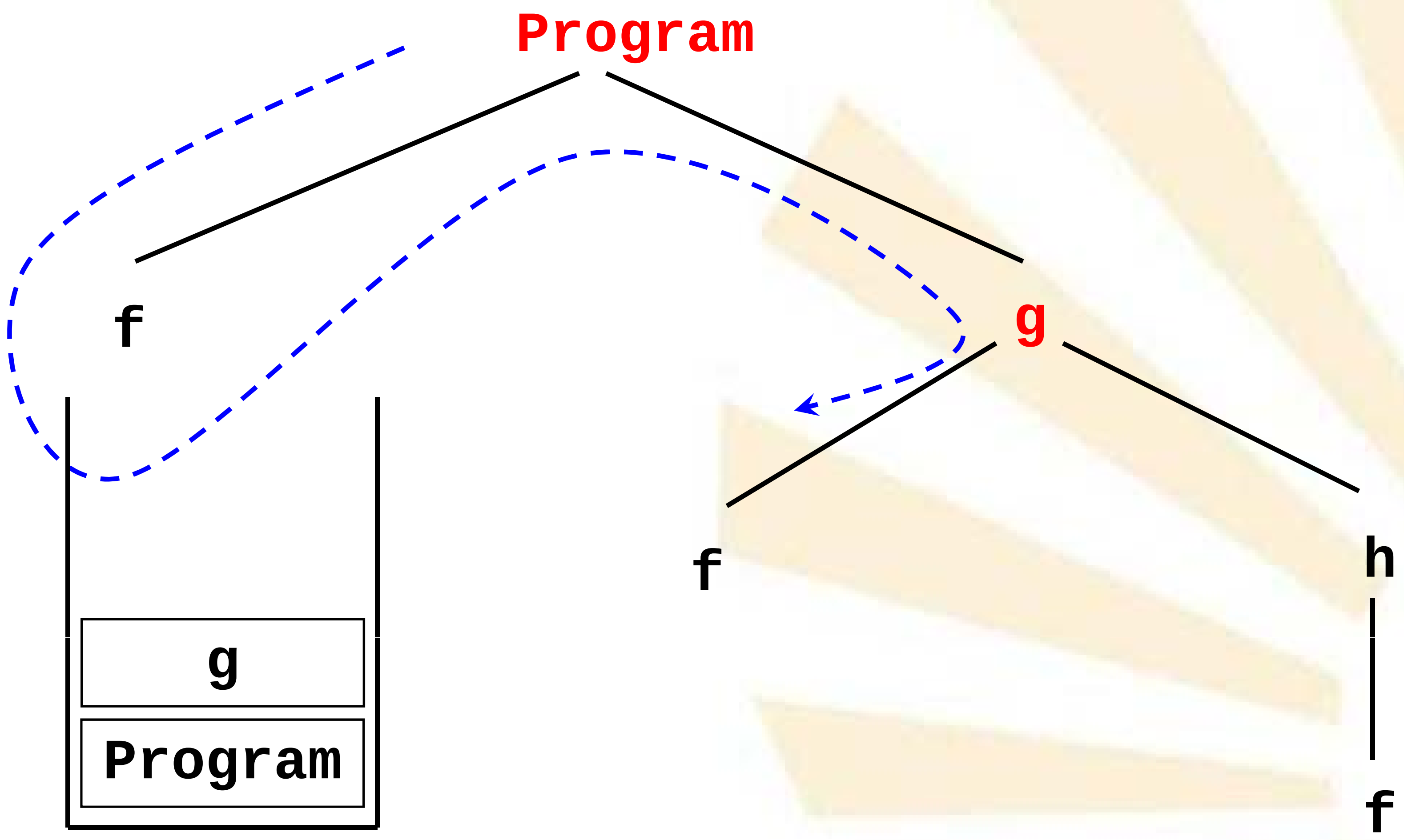

Program
f
g
f
h
f
Program
Activation Stack
Activation Tree



Program
f
g
f
h
f
f
Program
Activation Stack
Activation Tree

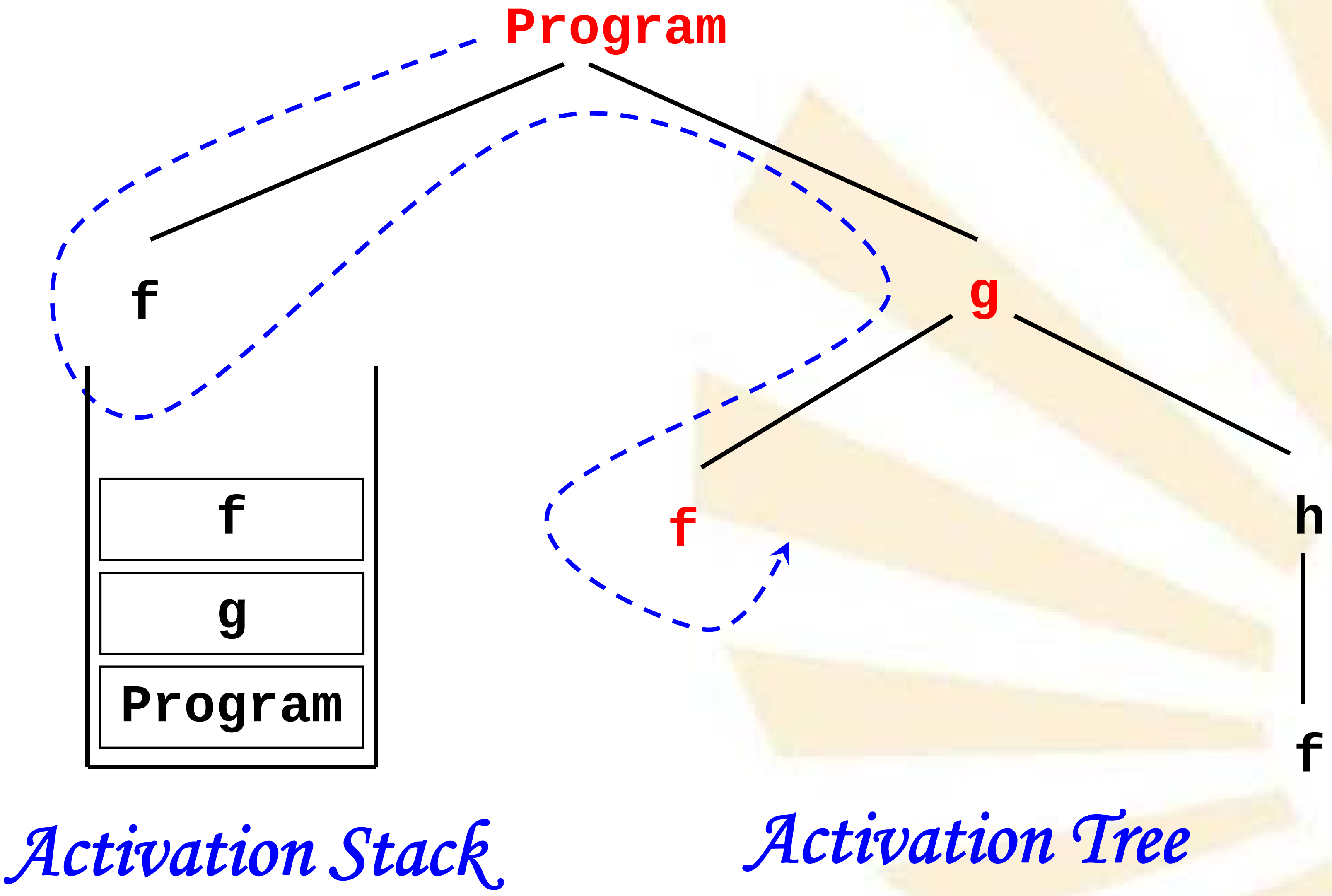


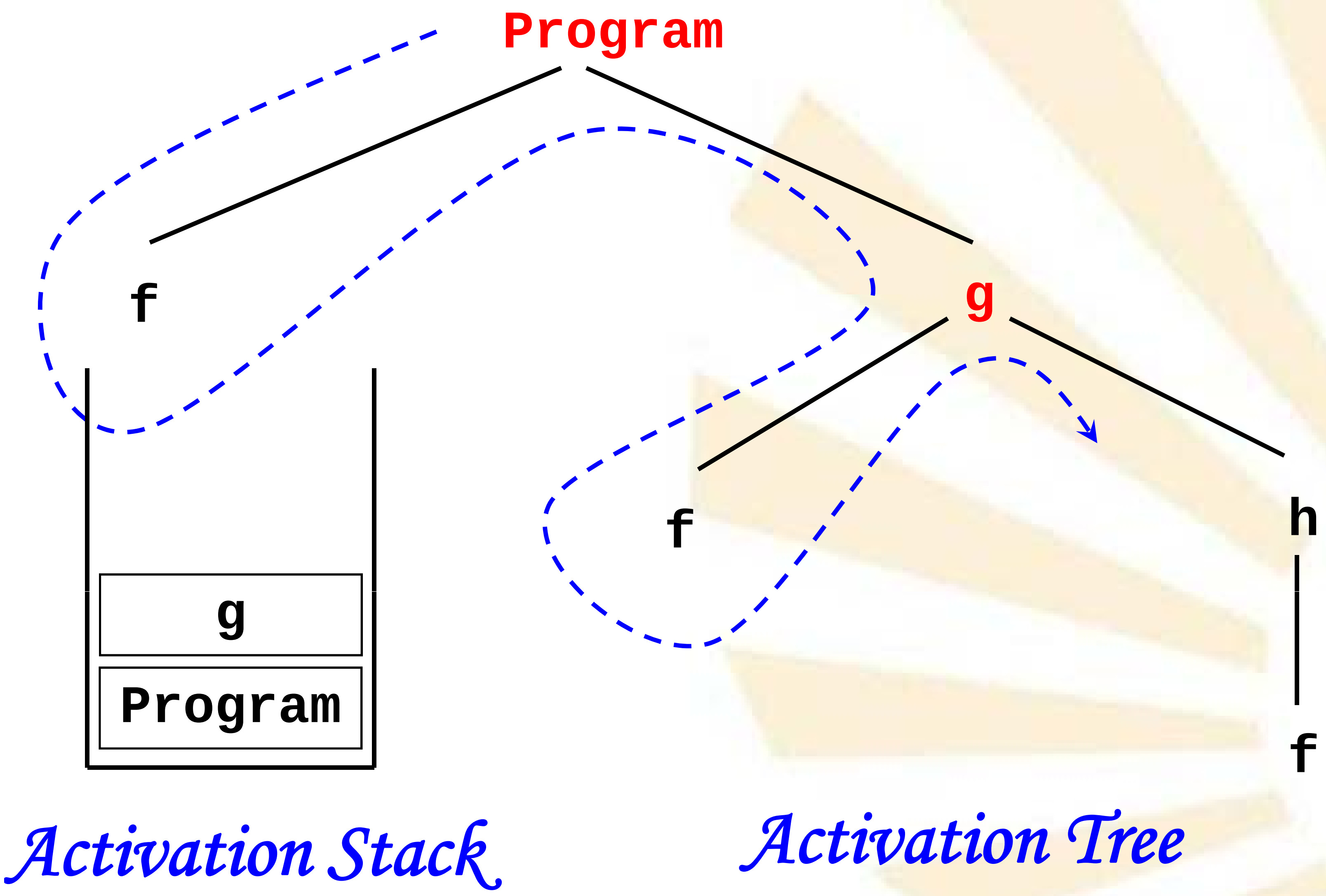
Activation Stack *Activation Tree*

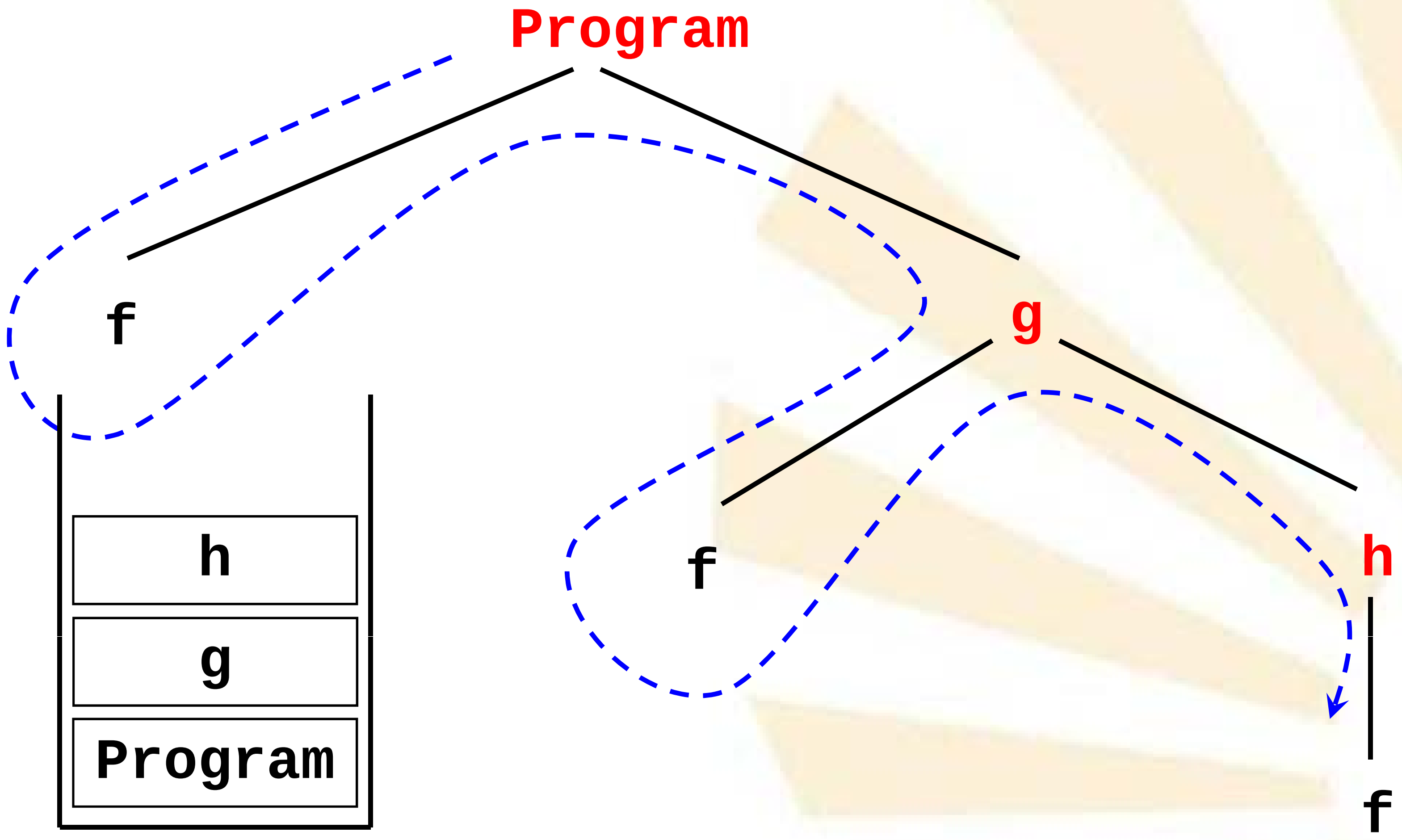


Activation Stack

Activation Tree

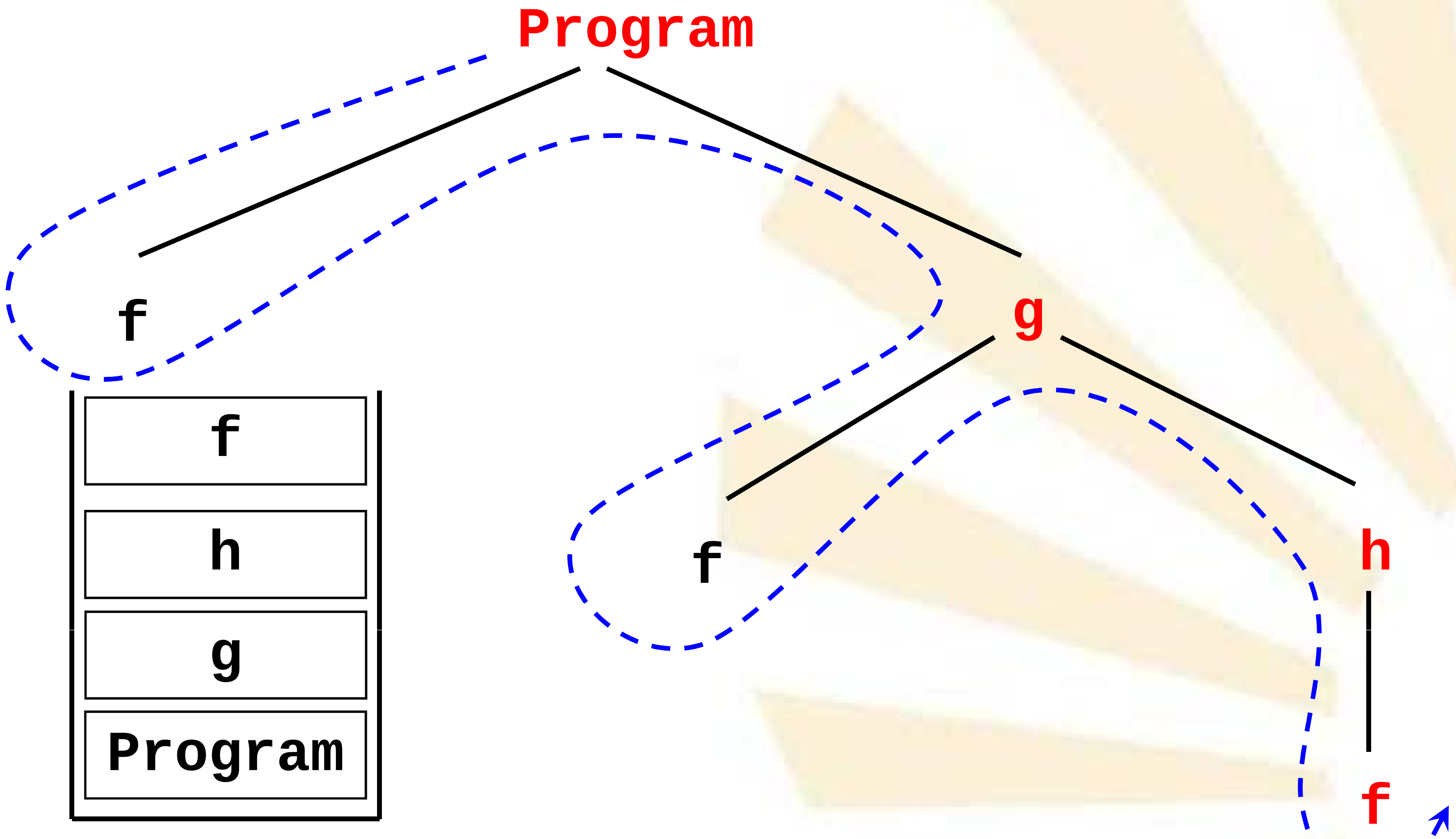

Program
f
g
f
h
f
f
g
Program
Activation Stack
Activation Tree


Program
f
g
f
h
f
g
Program
Activation Stack
Activation Tree



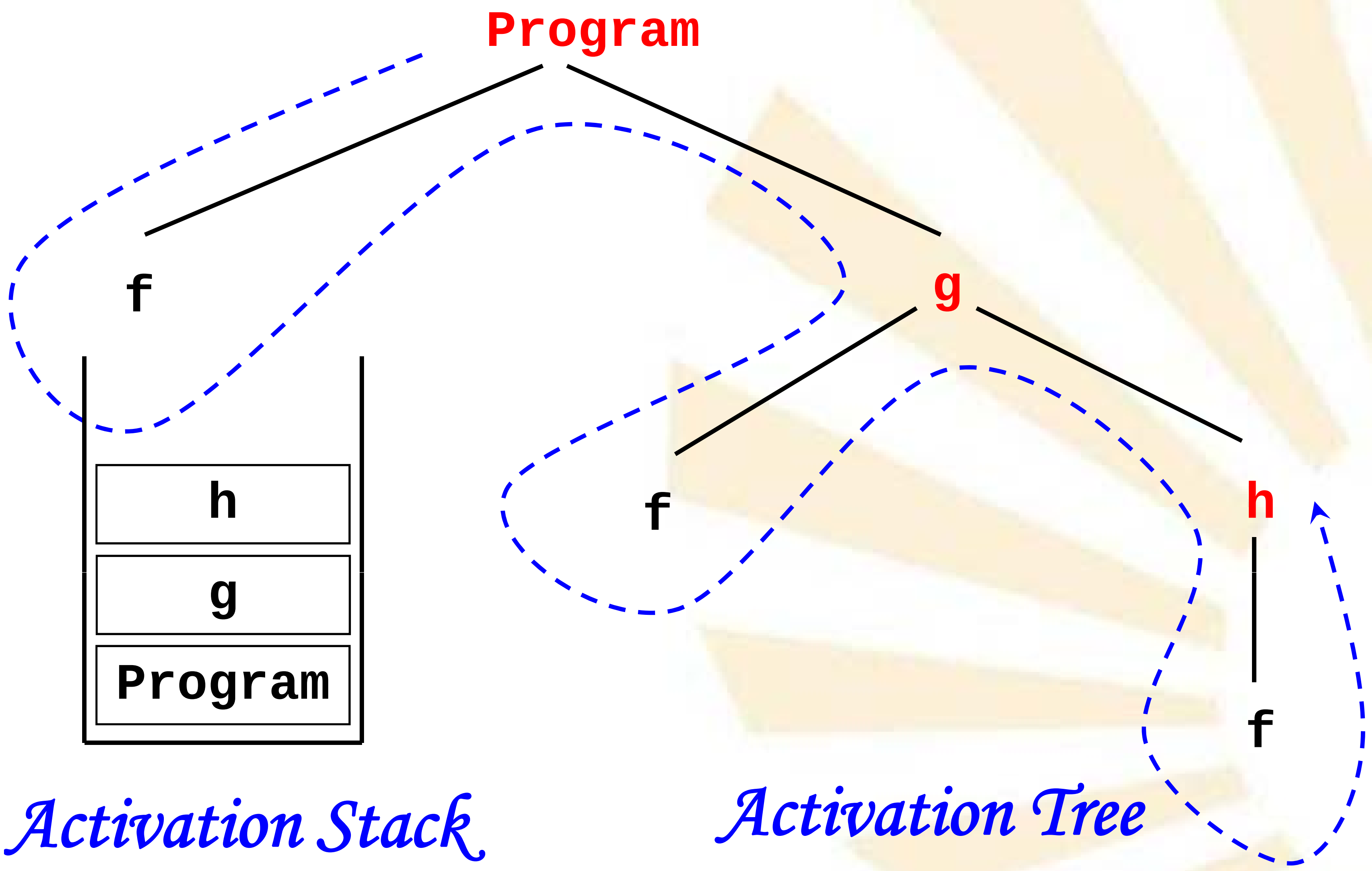
Activation Stack

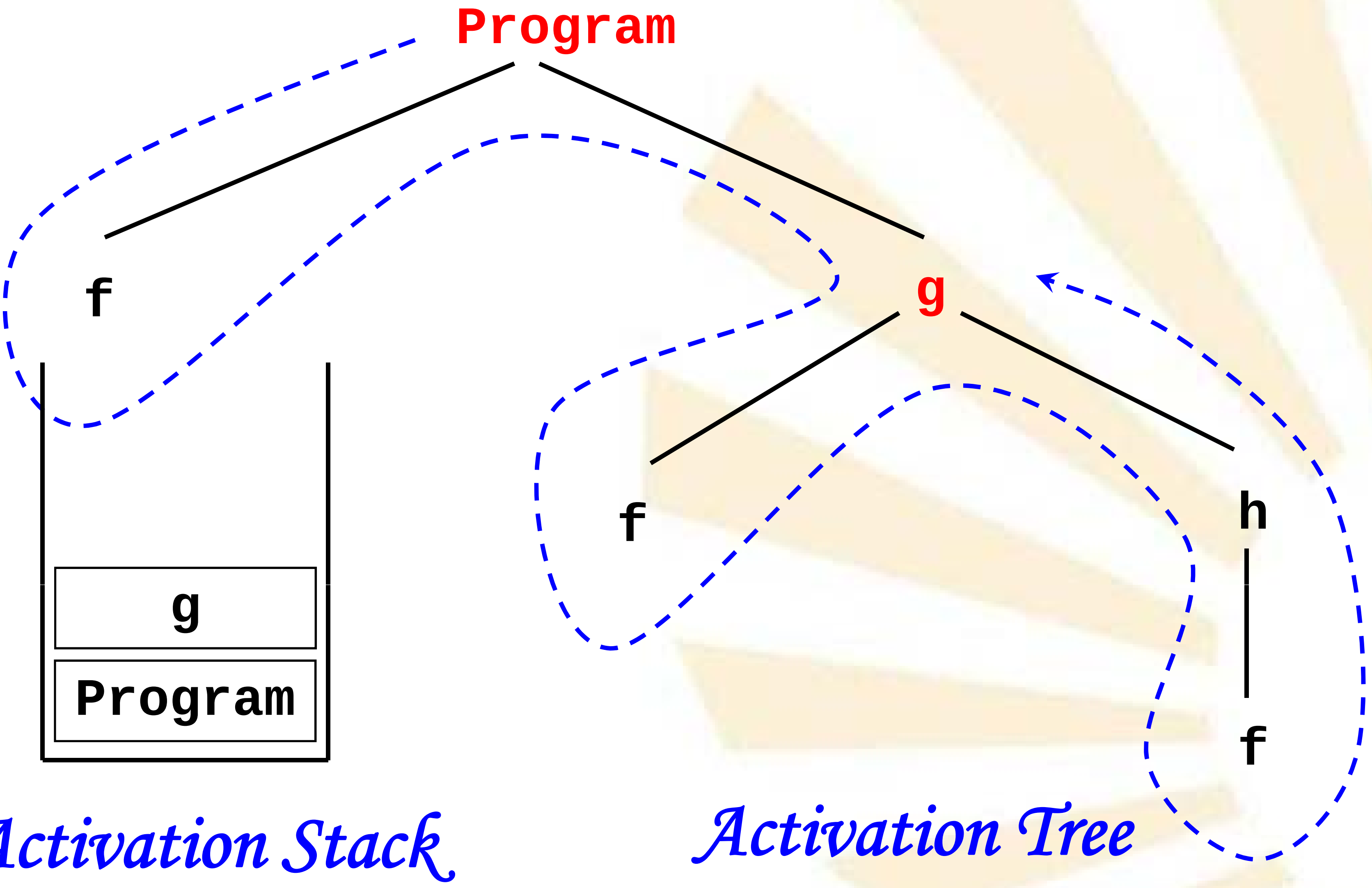
Activation Tree



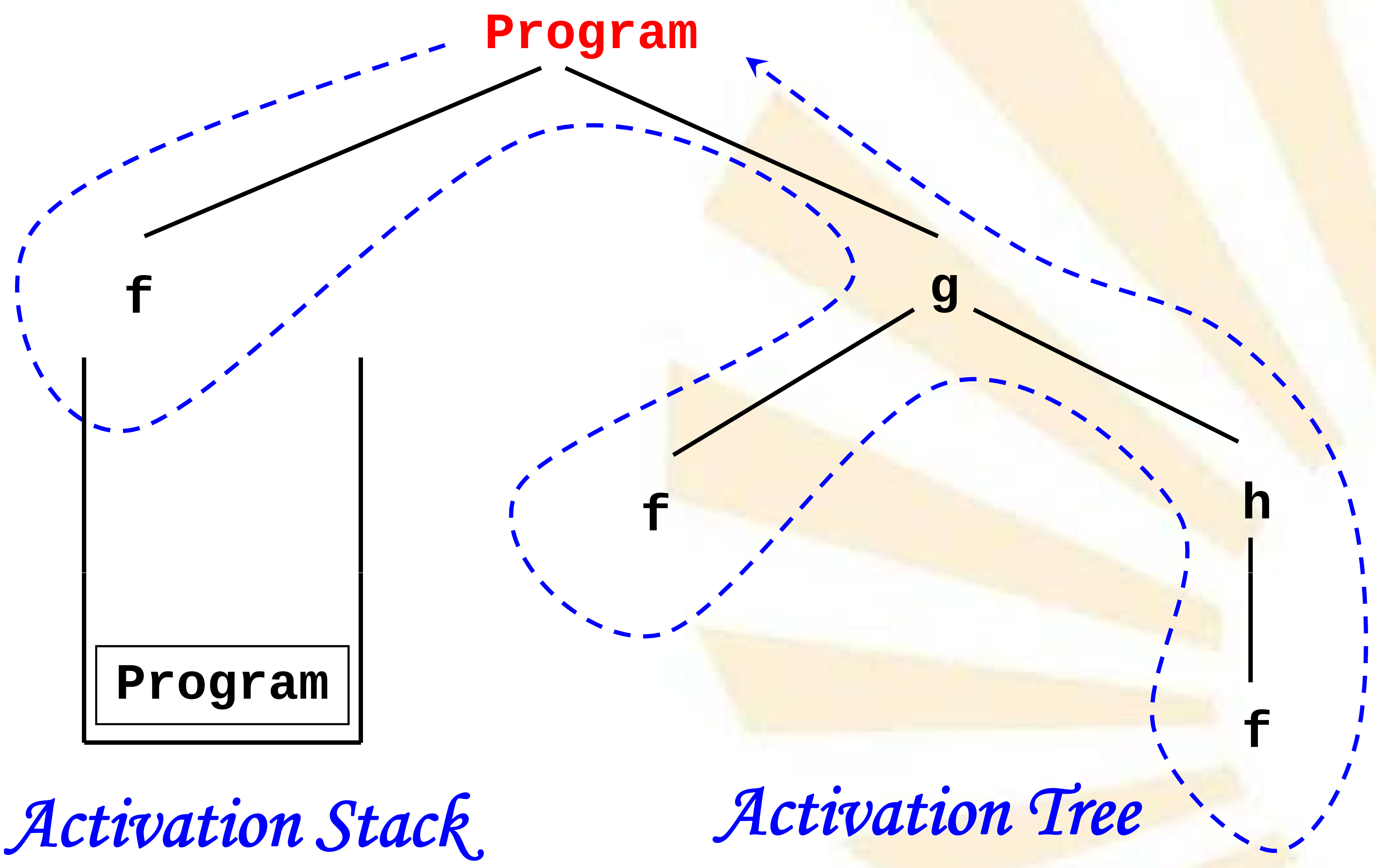
Activation Stack

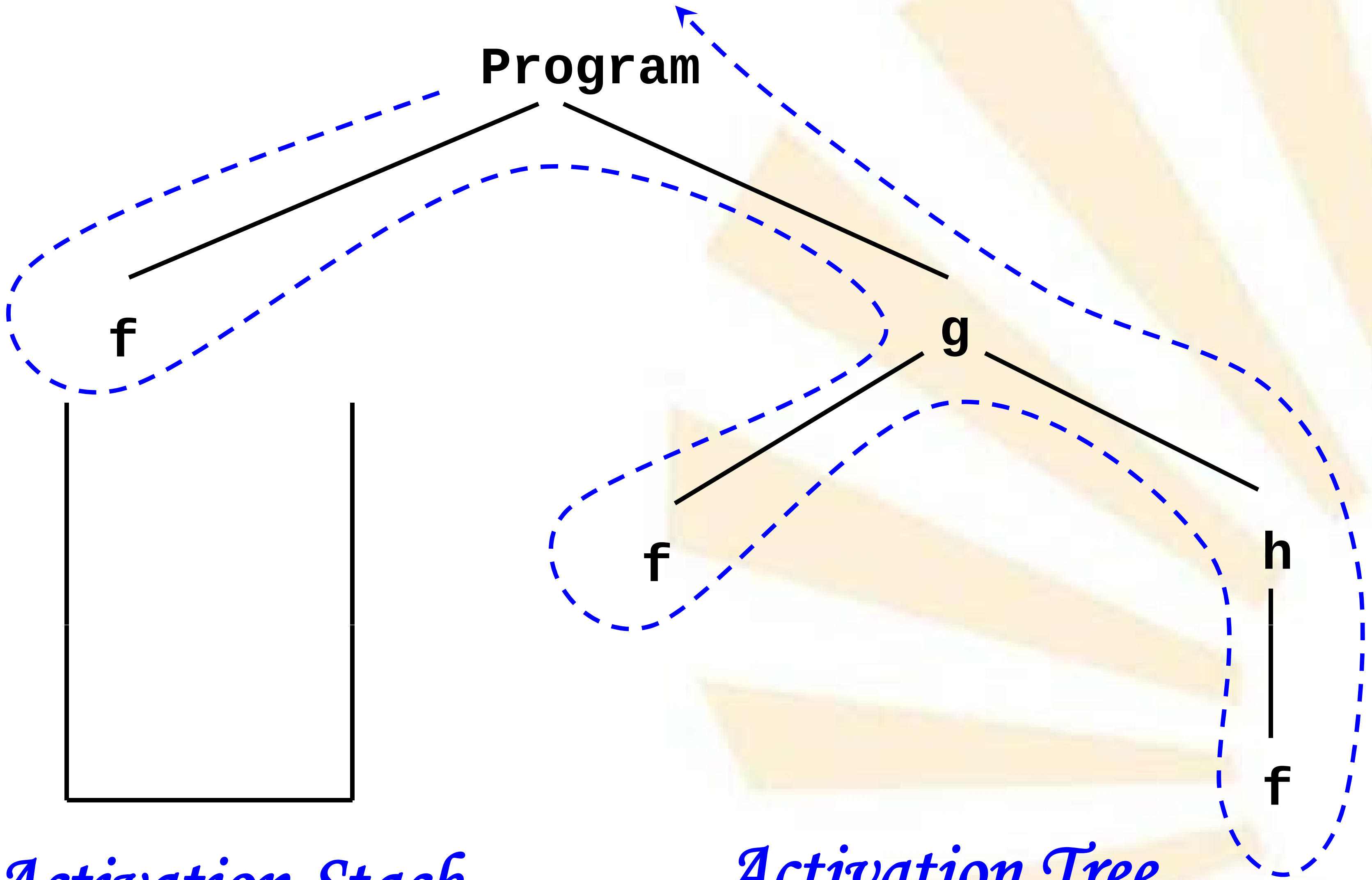
Activation Tree


Program
f
g
f
h
f
h
g
Program
Activation Stack
Activation Tree



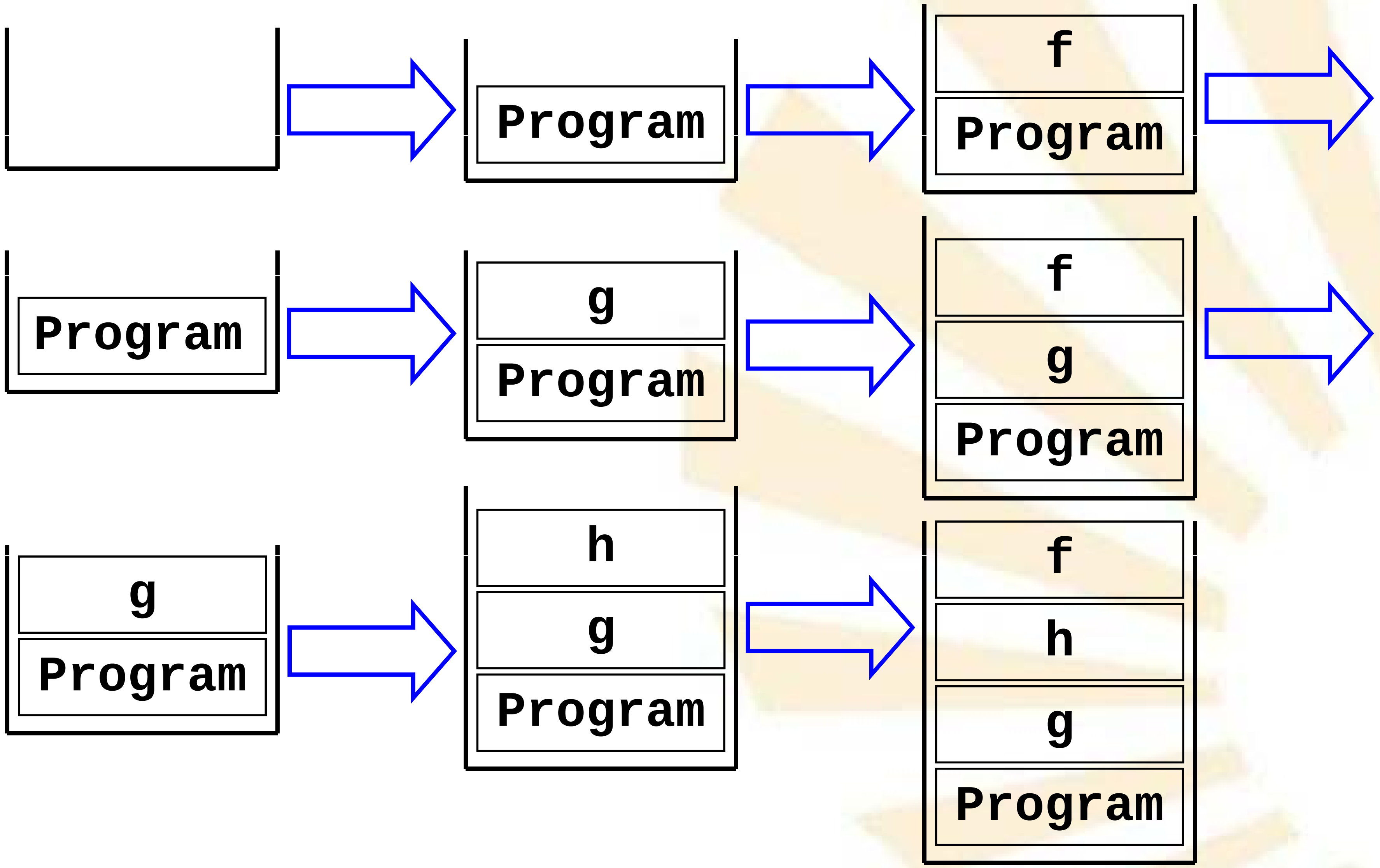
Activation Stack *Activation Tree*


Program
f
g
f
h
f
Program
Activation Stack
Activation Tree

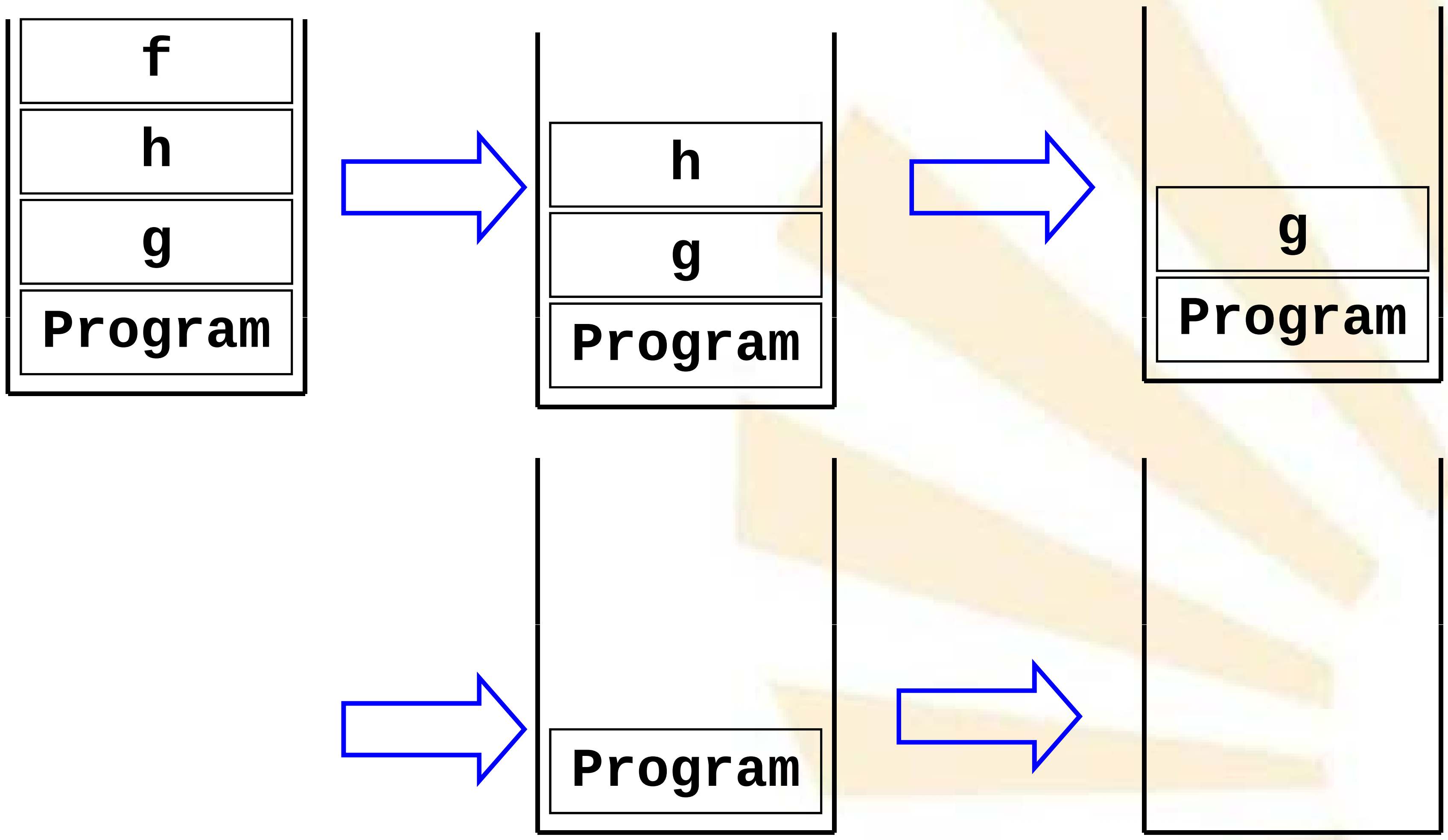


Activation Stack

Activation Tree



Changes in the activation Stack (1 / 2)



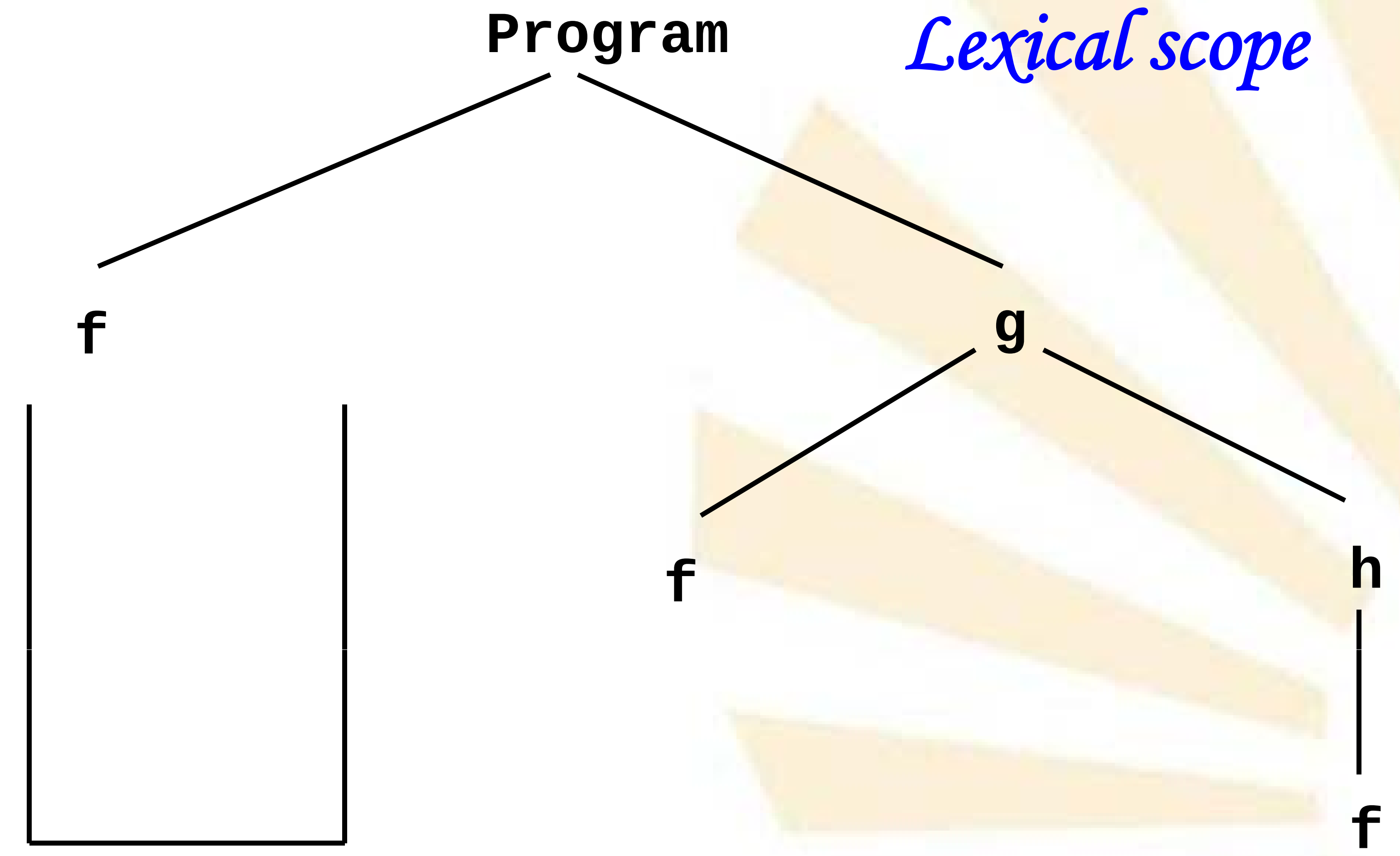
Changes in the activation Stack (2 / 2)

Run with

lexical scope

```
Program
    Declaration of variable x ($x_1$)

    Declaration of procedure f
     Use of x

   Declaration of procedure g
      Declaration of variable x  ($x_2$)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

Lexical scope



Activation Stack

Activation Tree

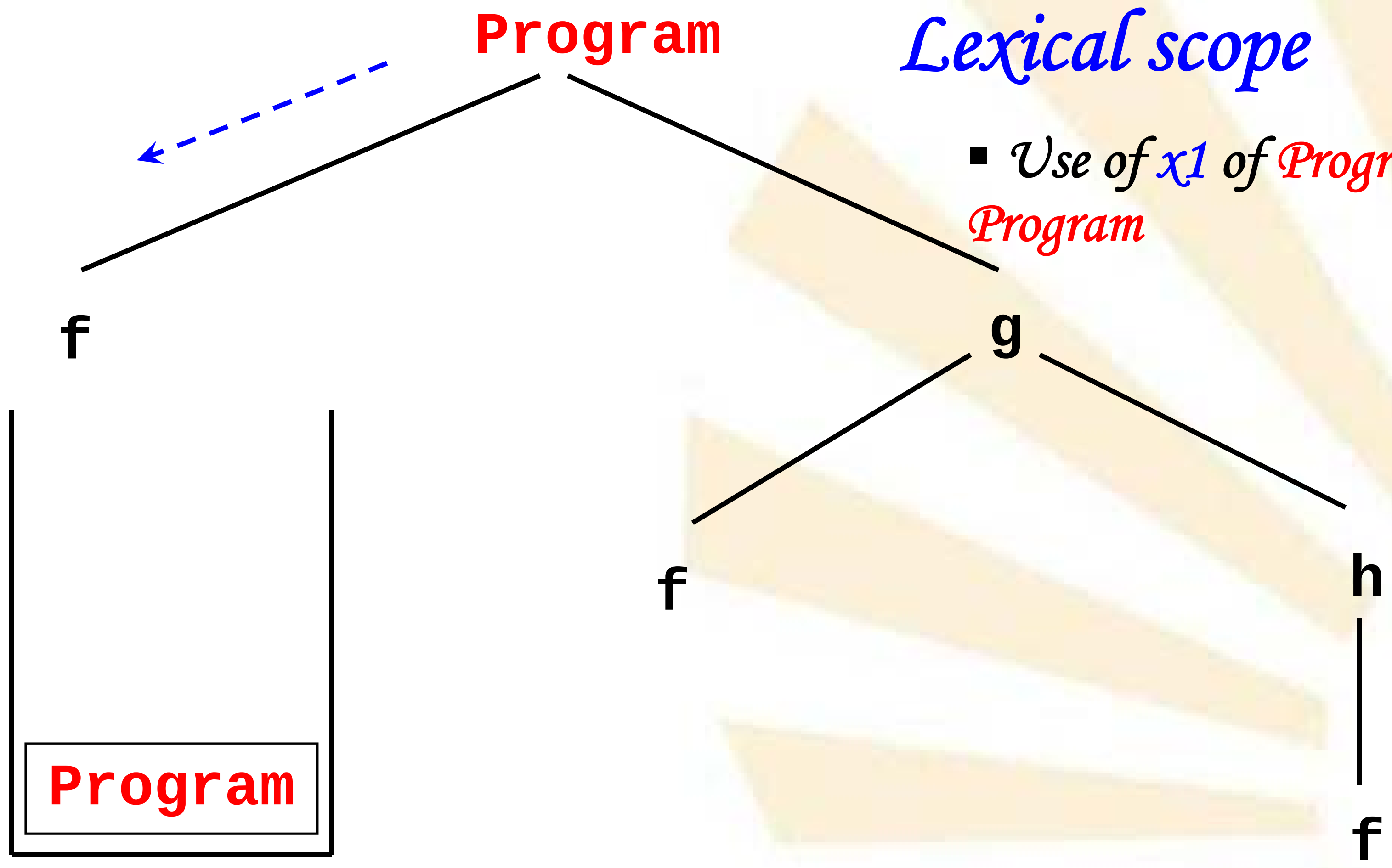
Run with

lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x1
     Call to f
     Call to g
```

Lexical scope

- *Use of x1 of Program in Program*



Activation Stack

Activation Tree

Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x


    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

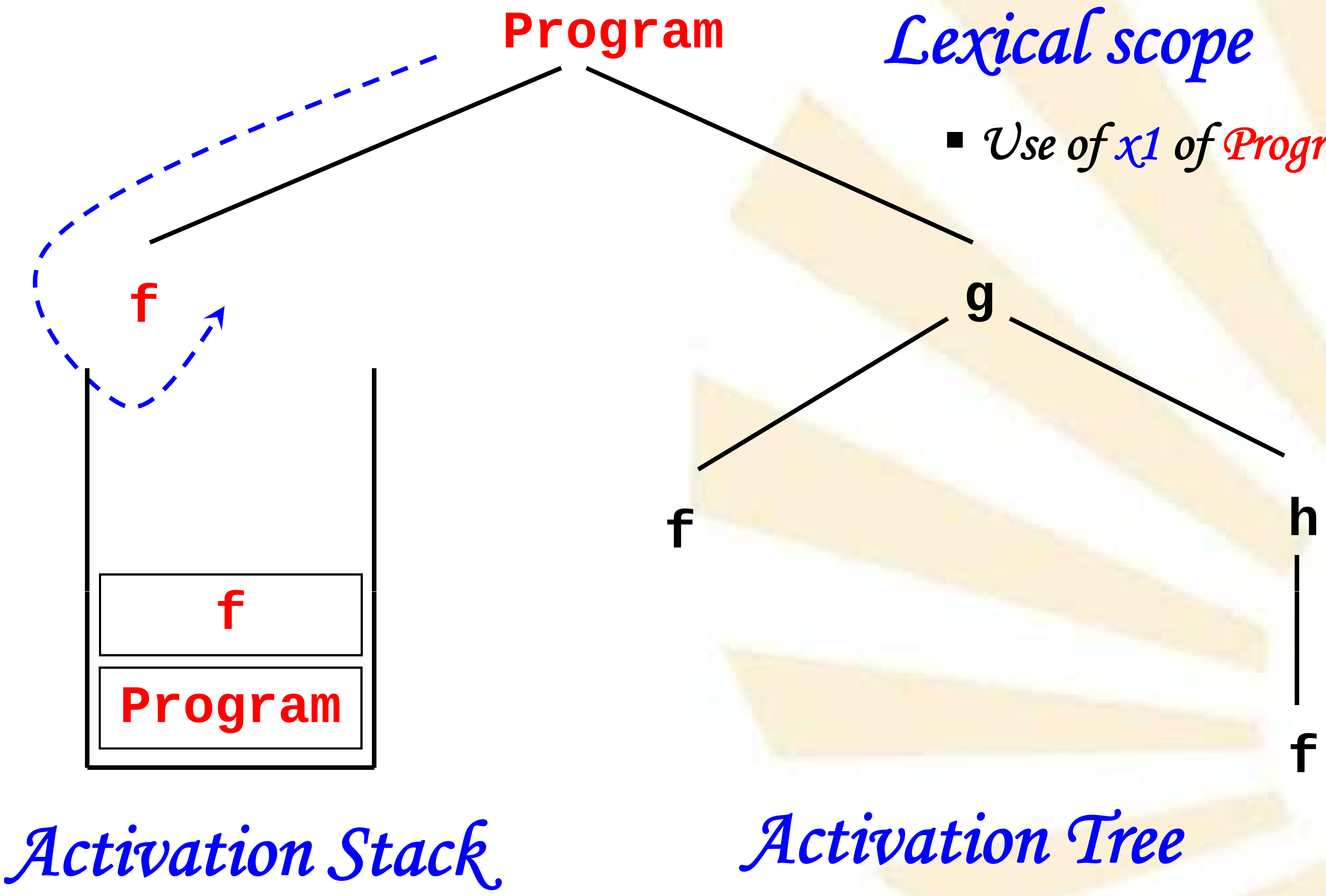
Run with

lexical scope

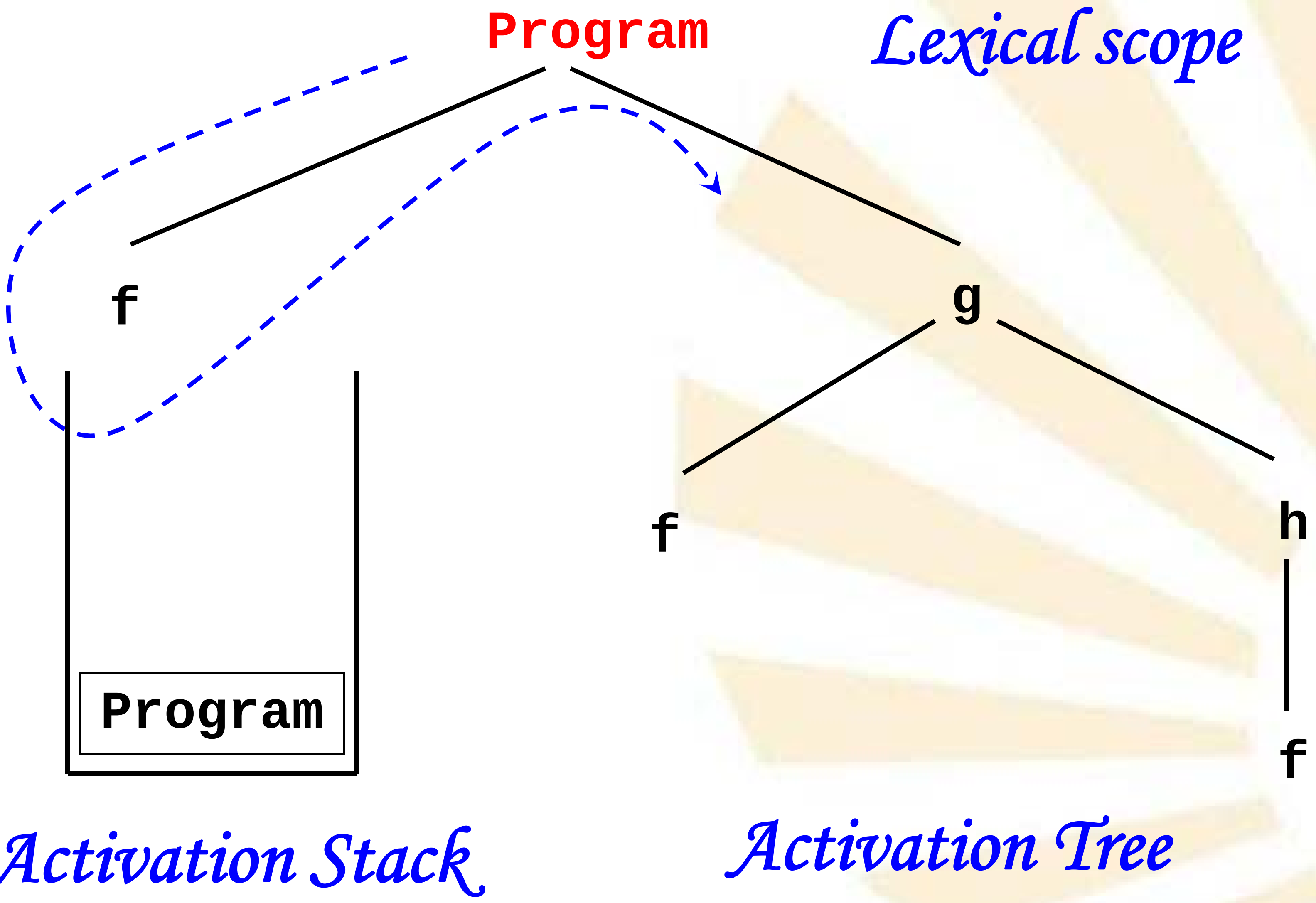
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x1  <=

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f  <=
     Call to g
```

Lexical scope

- Use of x1 of Program in f



Lexical scope

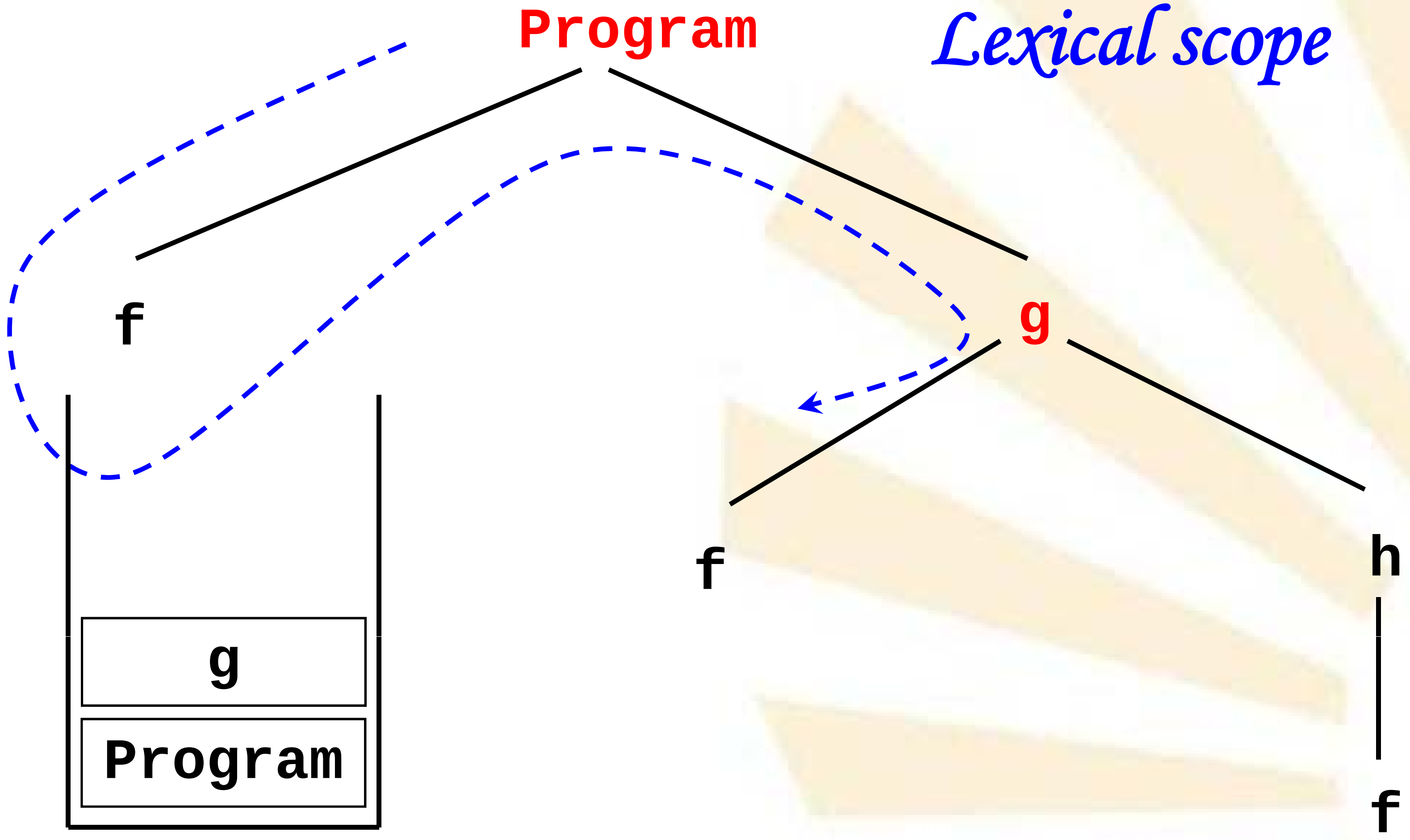


Run with

lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x


    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g  <=
```

Lexical scope



Activation Stack

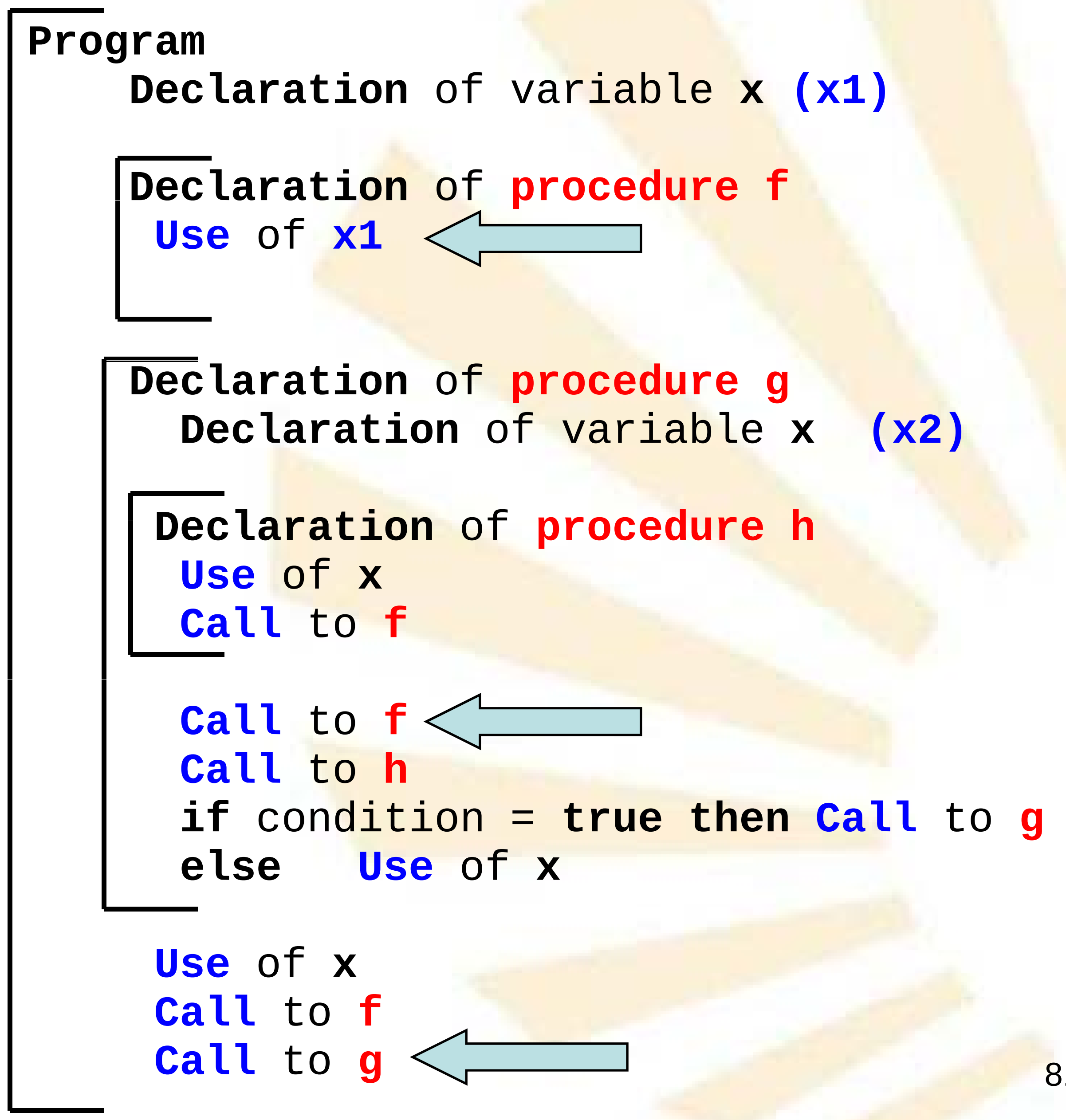
Activation Tree

Run with

lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f ←
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g ←
```

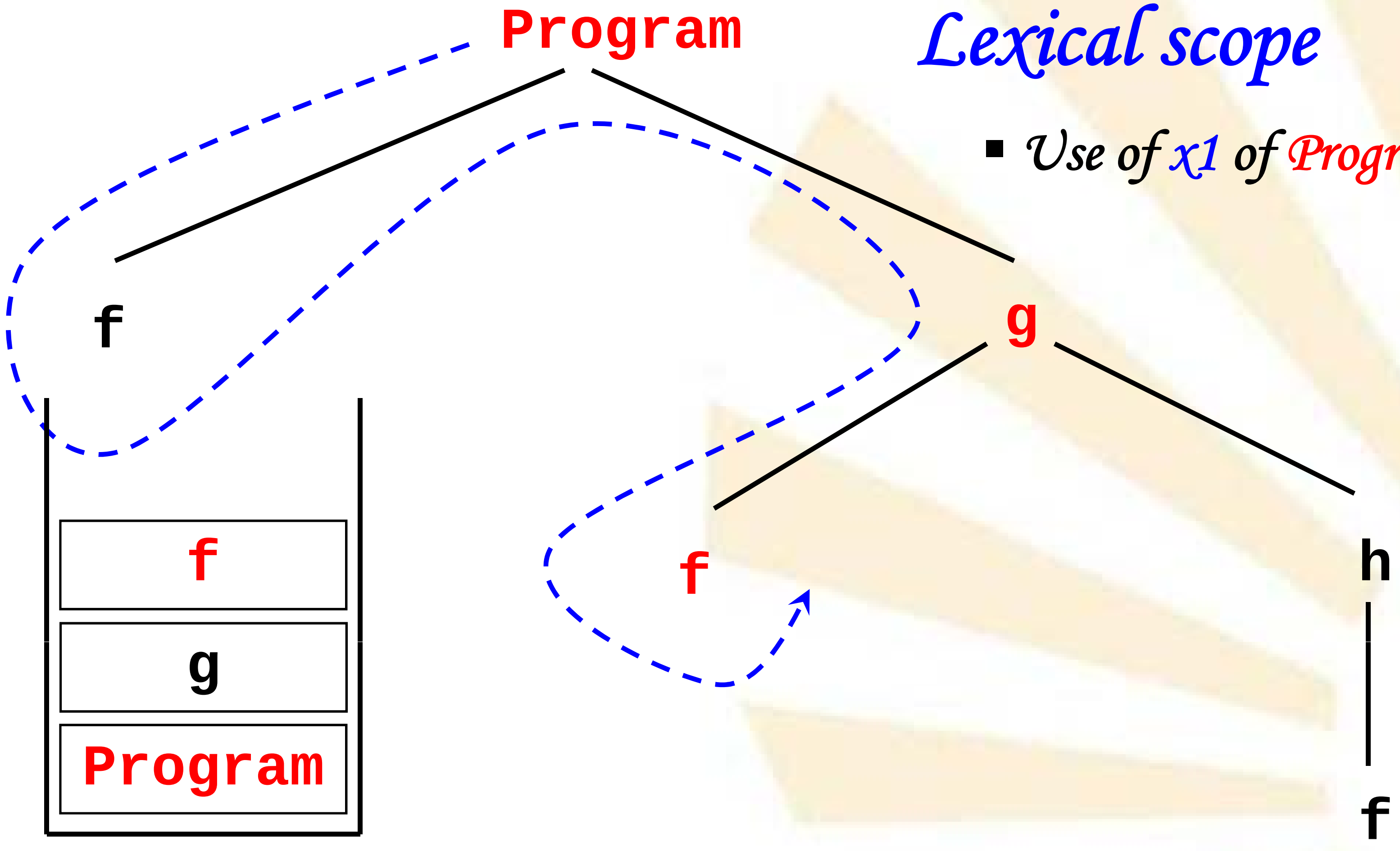
Run with lexical scope



```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x1

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

Lexical scope

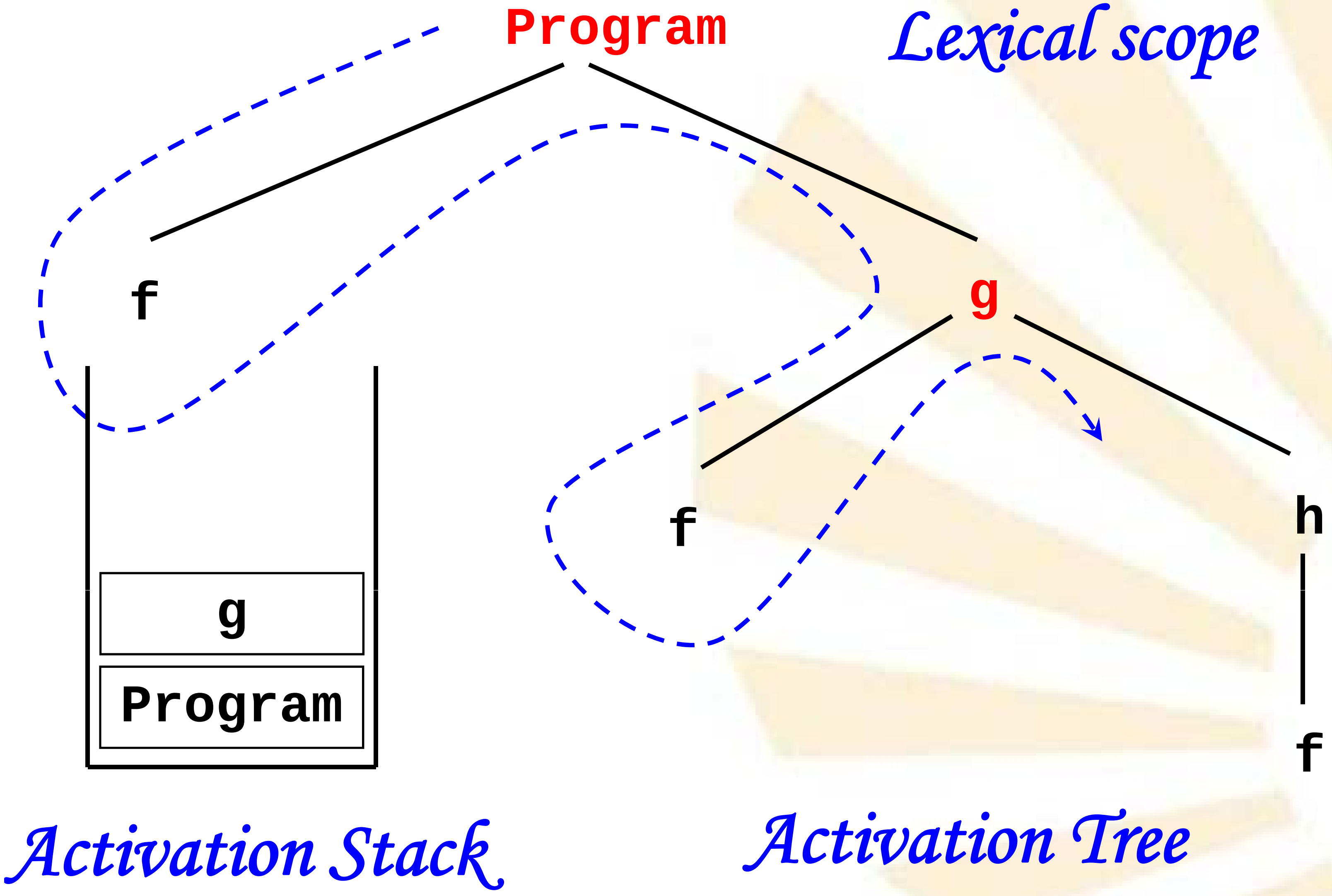
- Use of x1 of Program in f



Activation Stack

Activation Tree

Lexical scope



Run with

lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

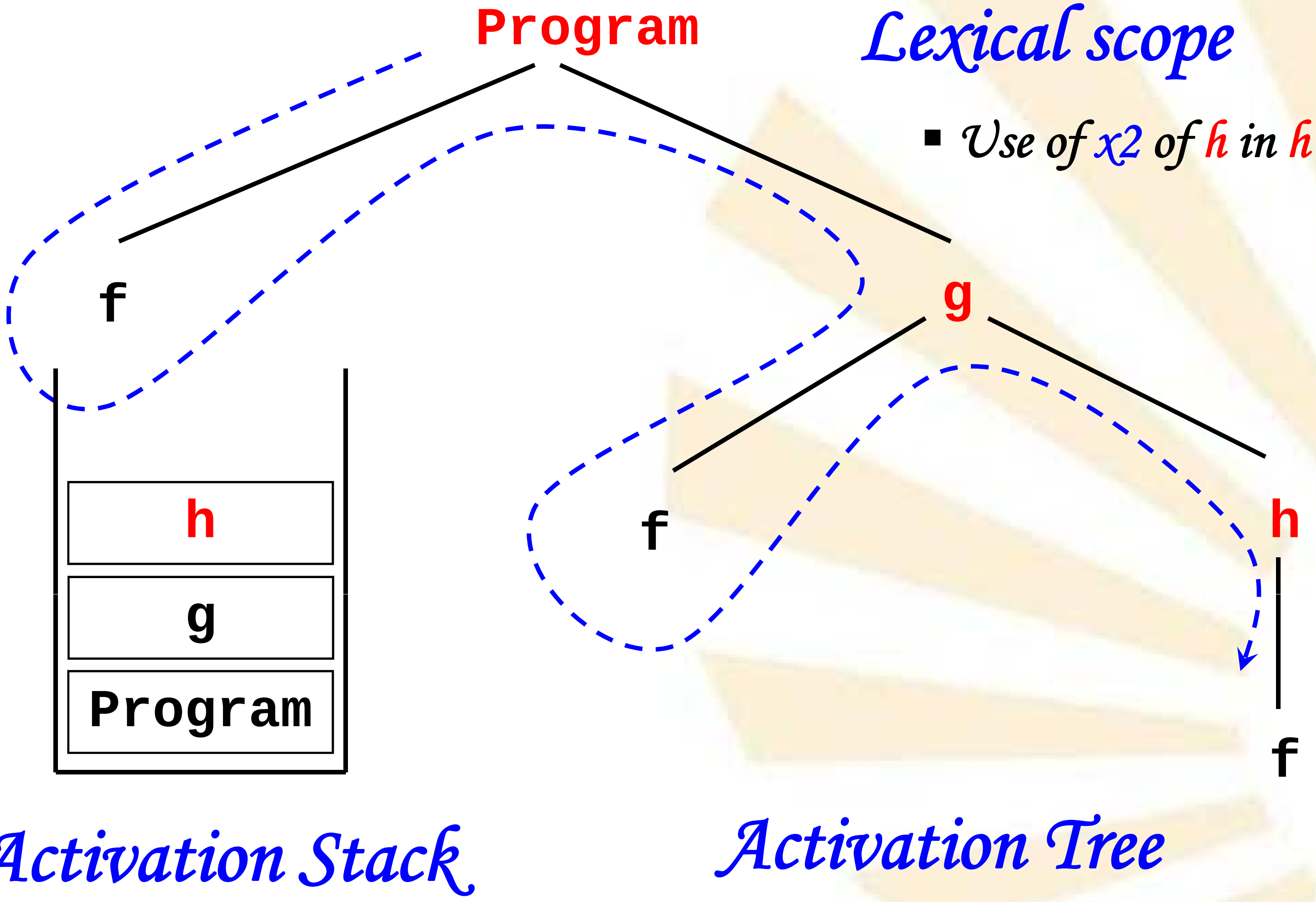
Run with

lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x


    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x2  <==
      Call to f

      Call to f
      Call to h  <==
      if condition = true then Call to g
      else    Use of x

     Use of x
     Call to f
     Call to g  <==
```

Lexical scope

- *Use of x2 of h in h*



Run with lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f ←

      Call to f
      Call to h ←
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g ←
```

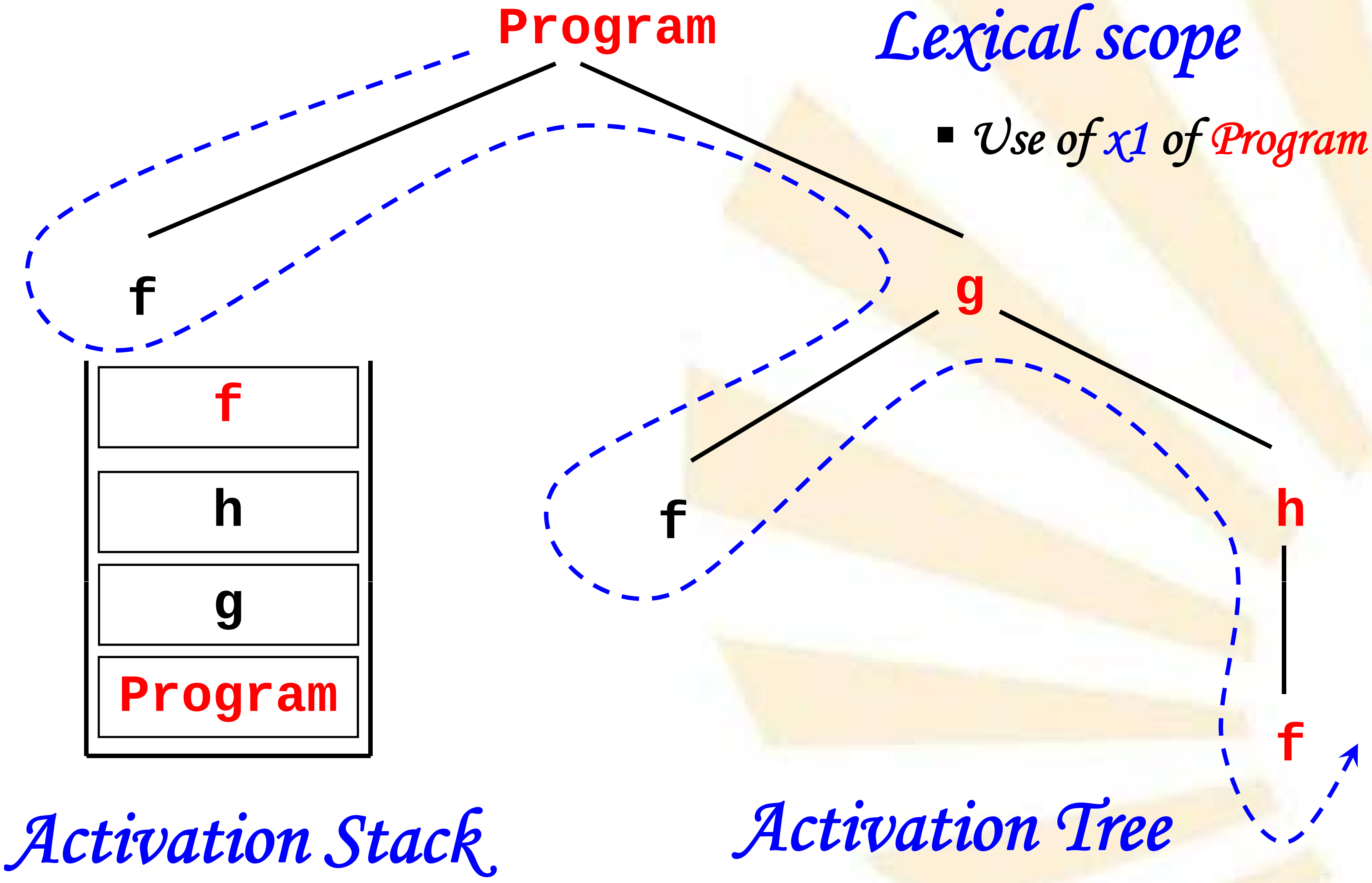
Run with

lexical scope

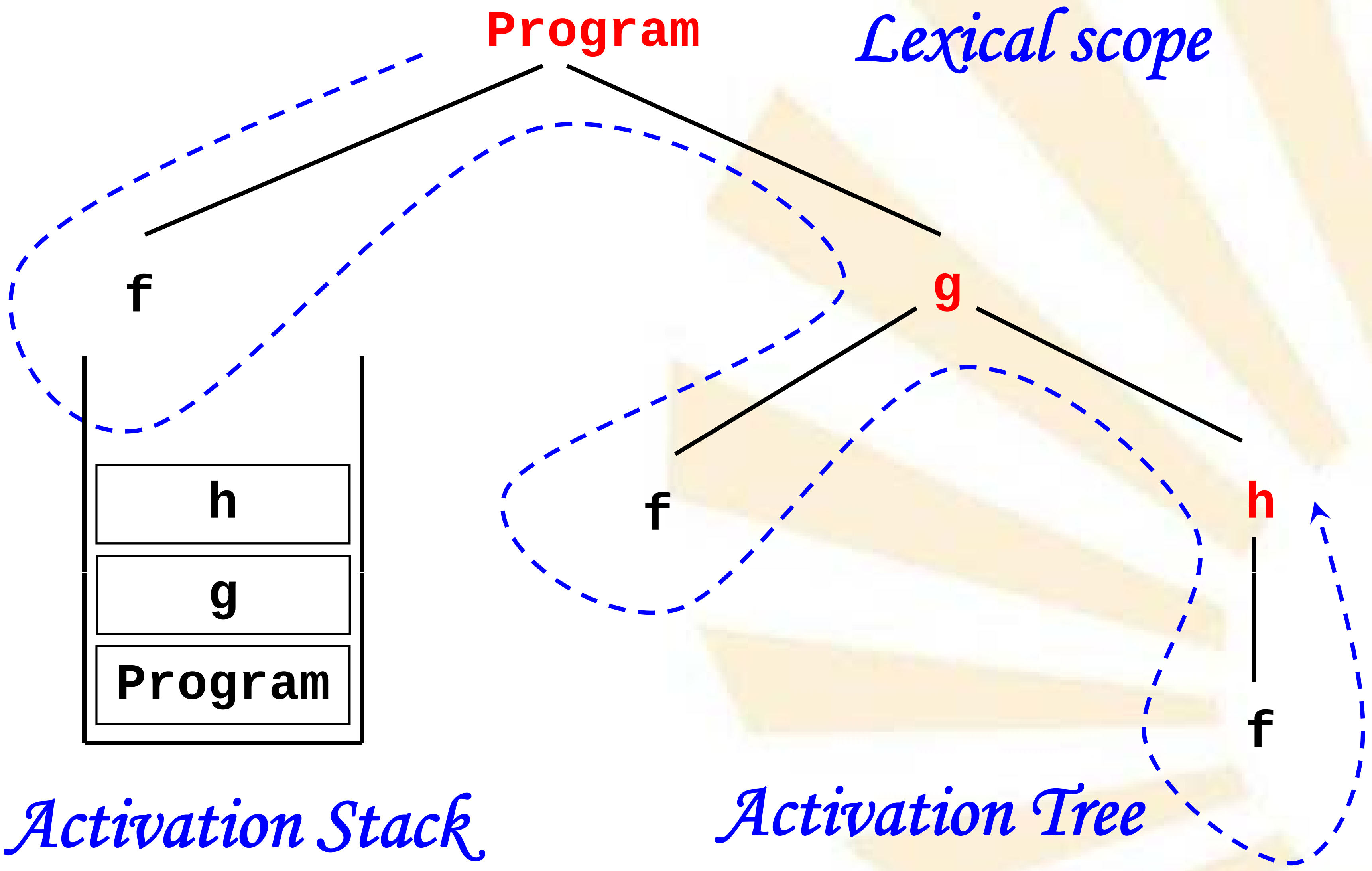
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x1


    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else    Use of x

     Use of x
     Call to f
     Call to g
```

Lexical scope

- *Use of x1 of Program in f*



Lexical scope



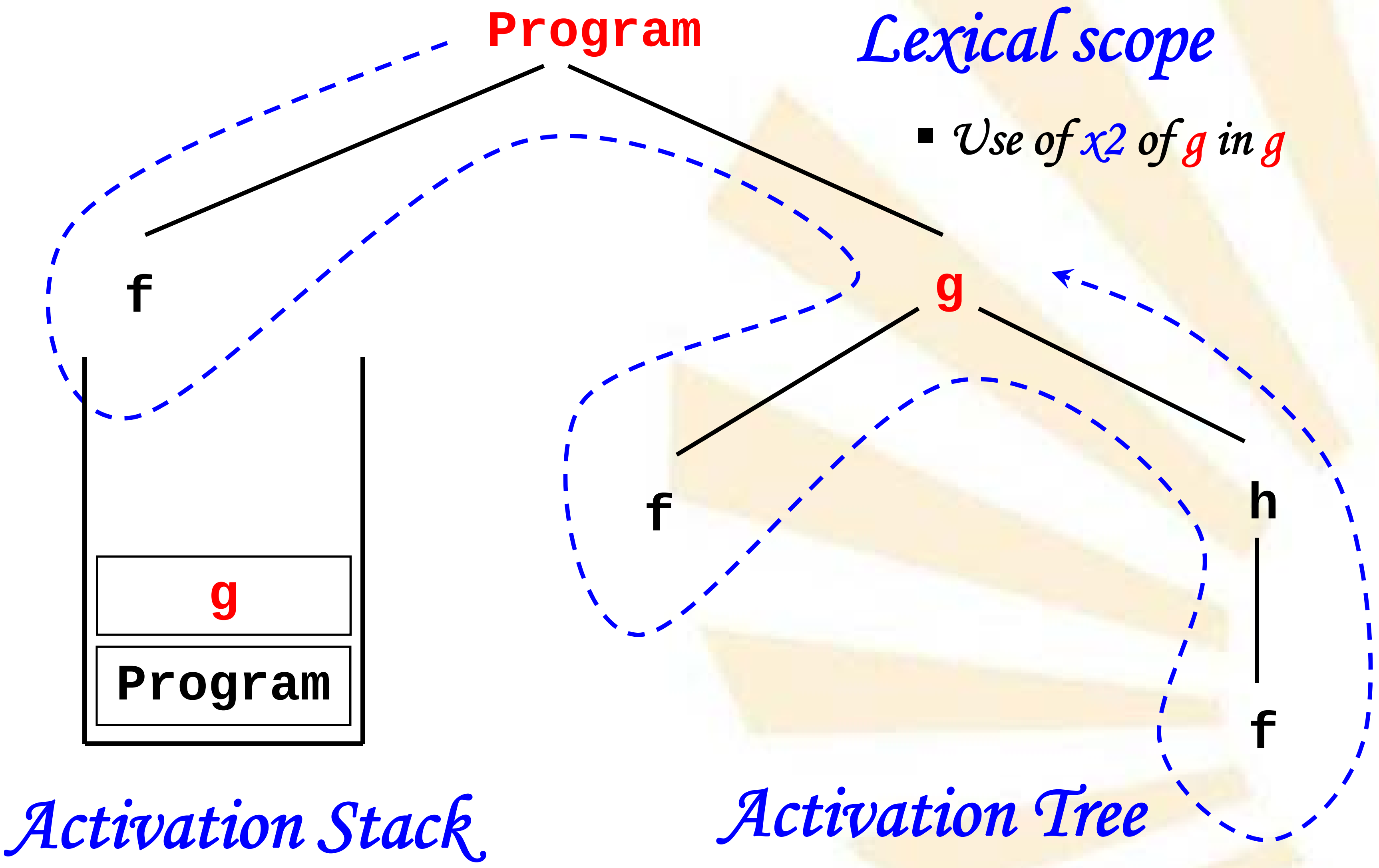
Run with

lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x2


    Declaration of procedure g
      Declaration of variable x   (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else    Use of x2 <=

     Use of x
     Call to f
     Call to g <=
```

Lexical scope

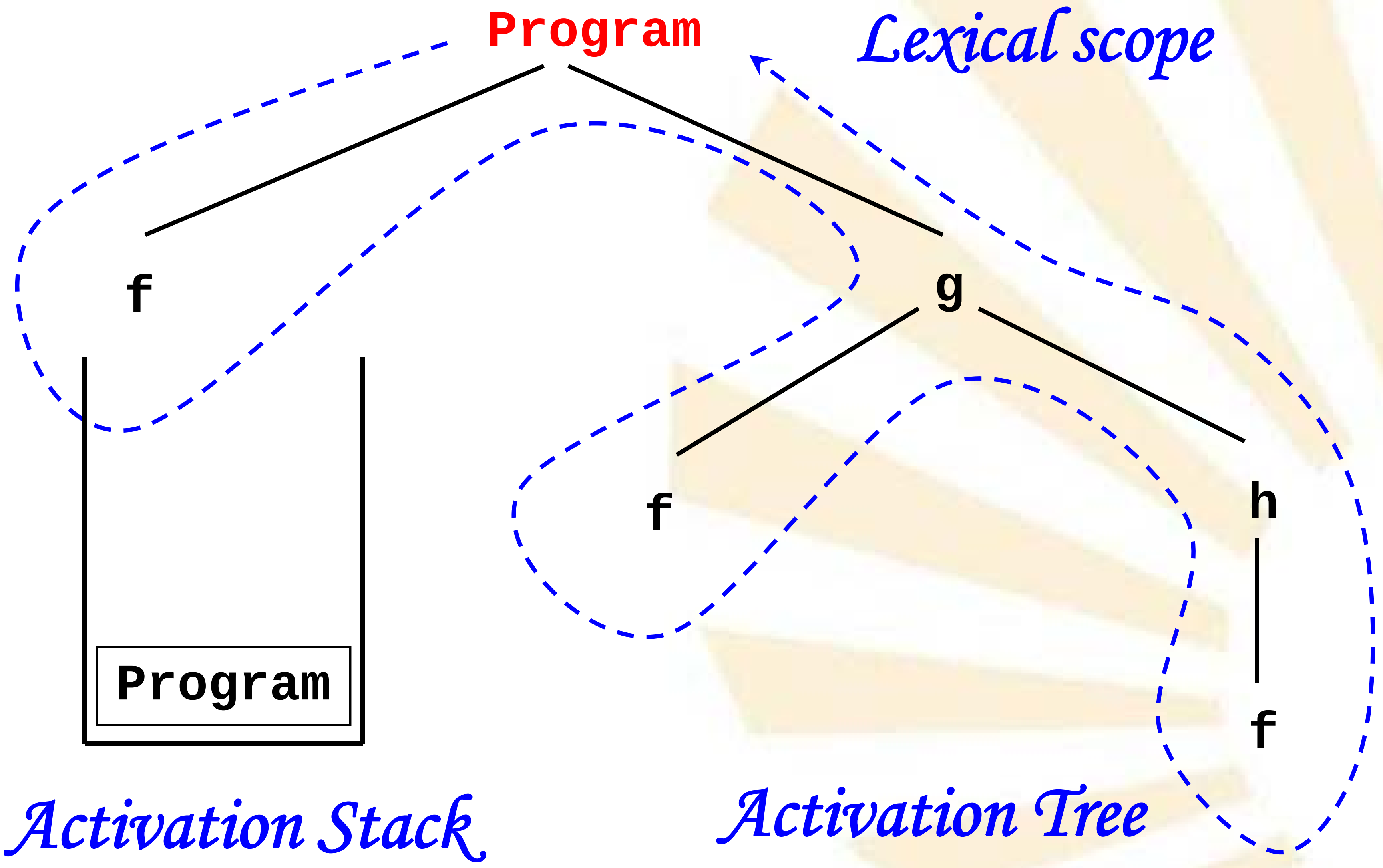
- *Use of x2 of g in g*

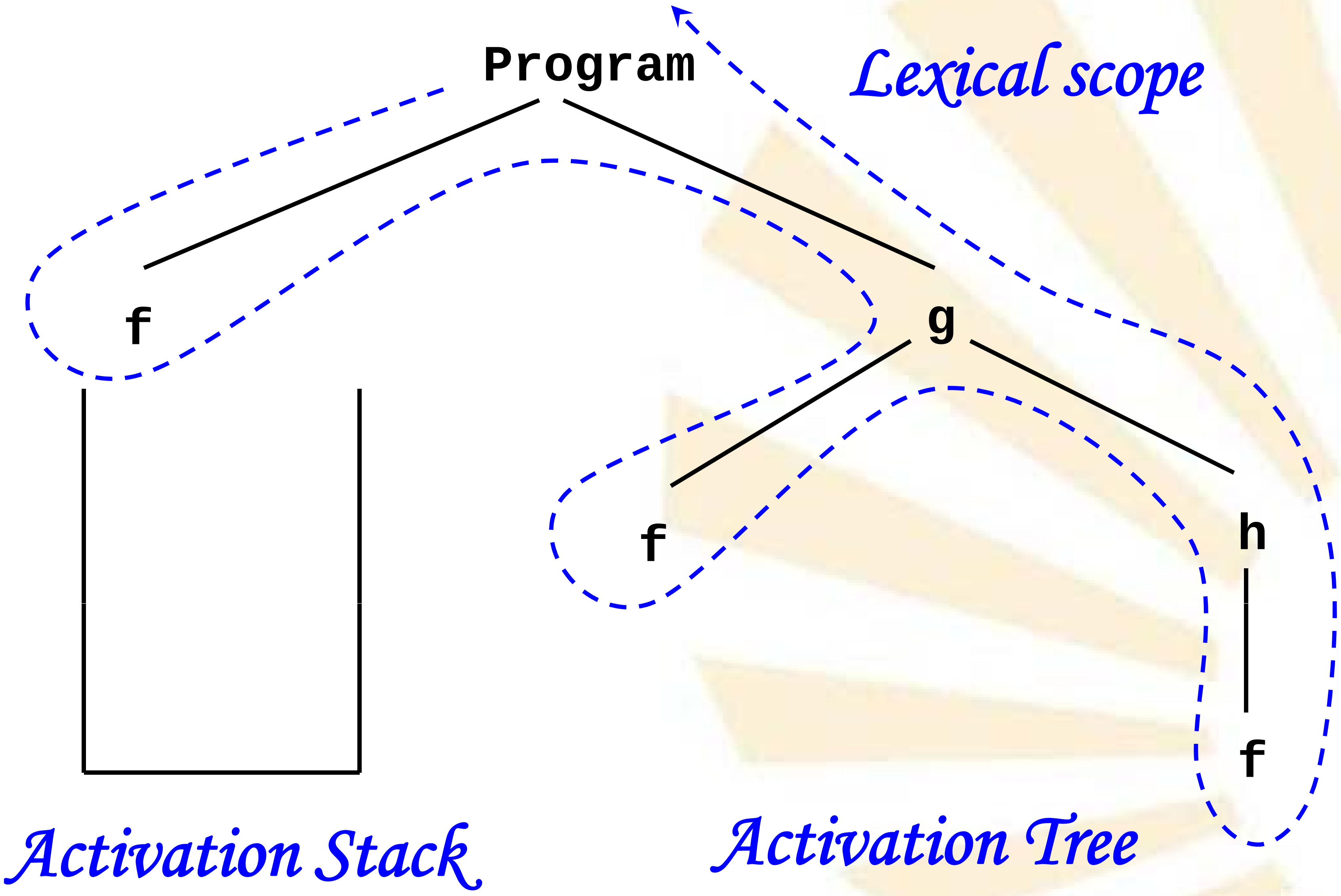


Run with

lexical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x2


    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x2

     Use of x
     Call to f
     Call to g
```


Program
Lexical scope
f
g
f
h
f
Program
Activation Stack
Activation Tree

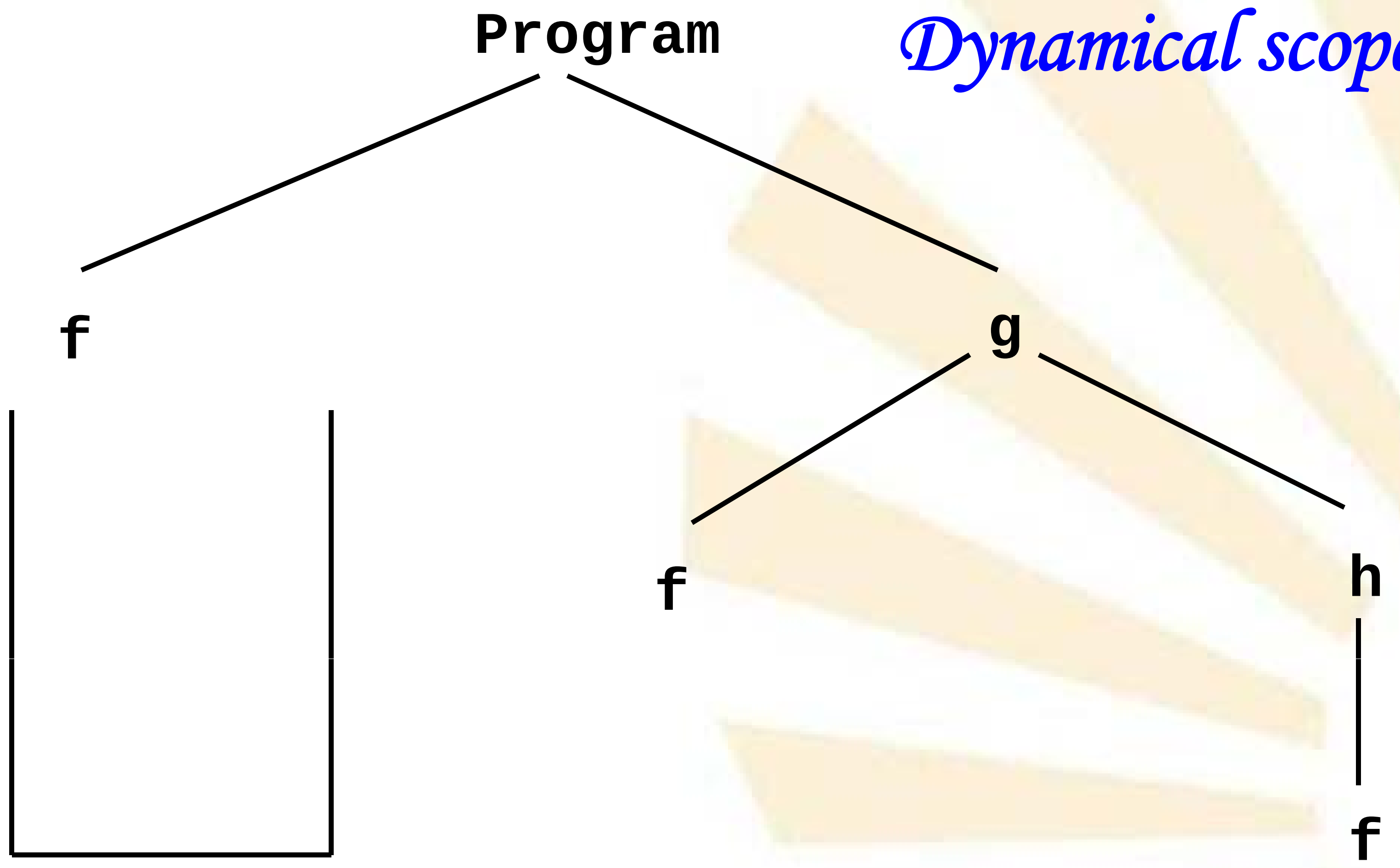

Program
Lexical scope
f
g
f
h
f
Activation Stack
Activation Tree

Run with

dynamical scope

```
Program
    Declaration of variable x (x_1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x_2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

Dynamical scope



Activation Stack *Activation Tree*

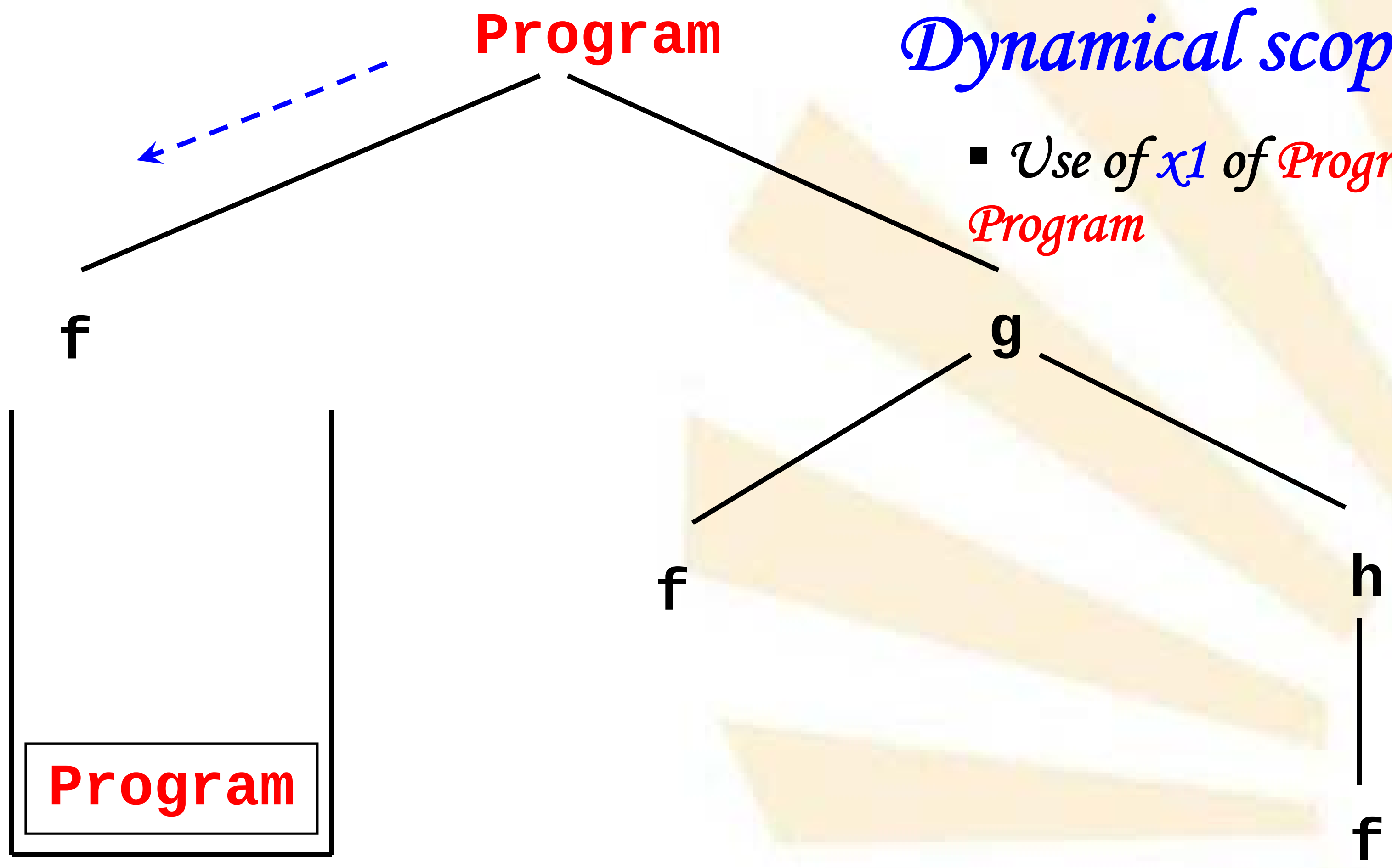
Run with

dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x1  <==
     Call to f
     Call to g
```

Dynamical scope

- *Use of x1 of Program in Program*



Activation Stack *Activation Tree*

Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f  <=
     Call to g
```

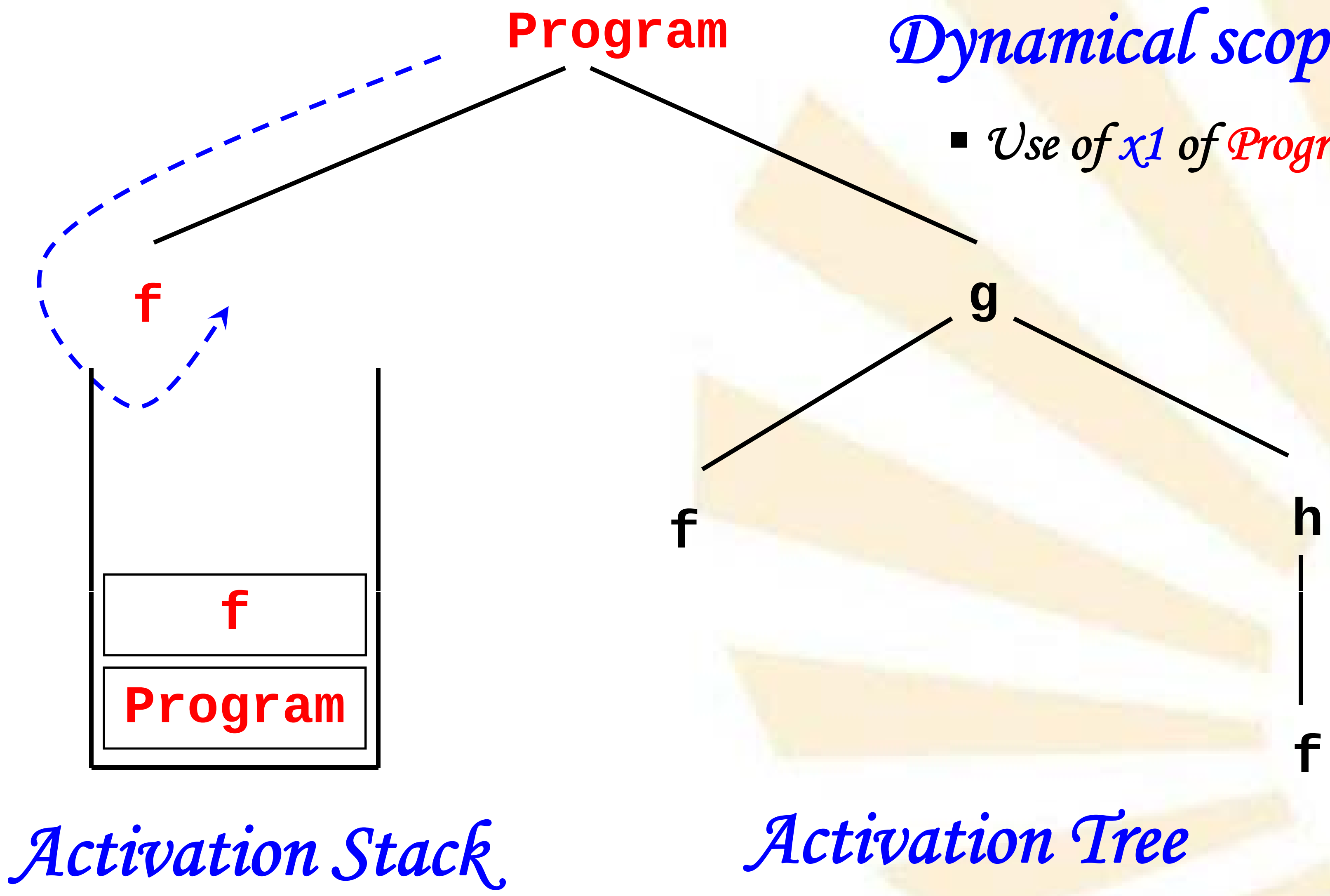
Run with

dynamical scope

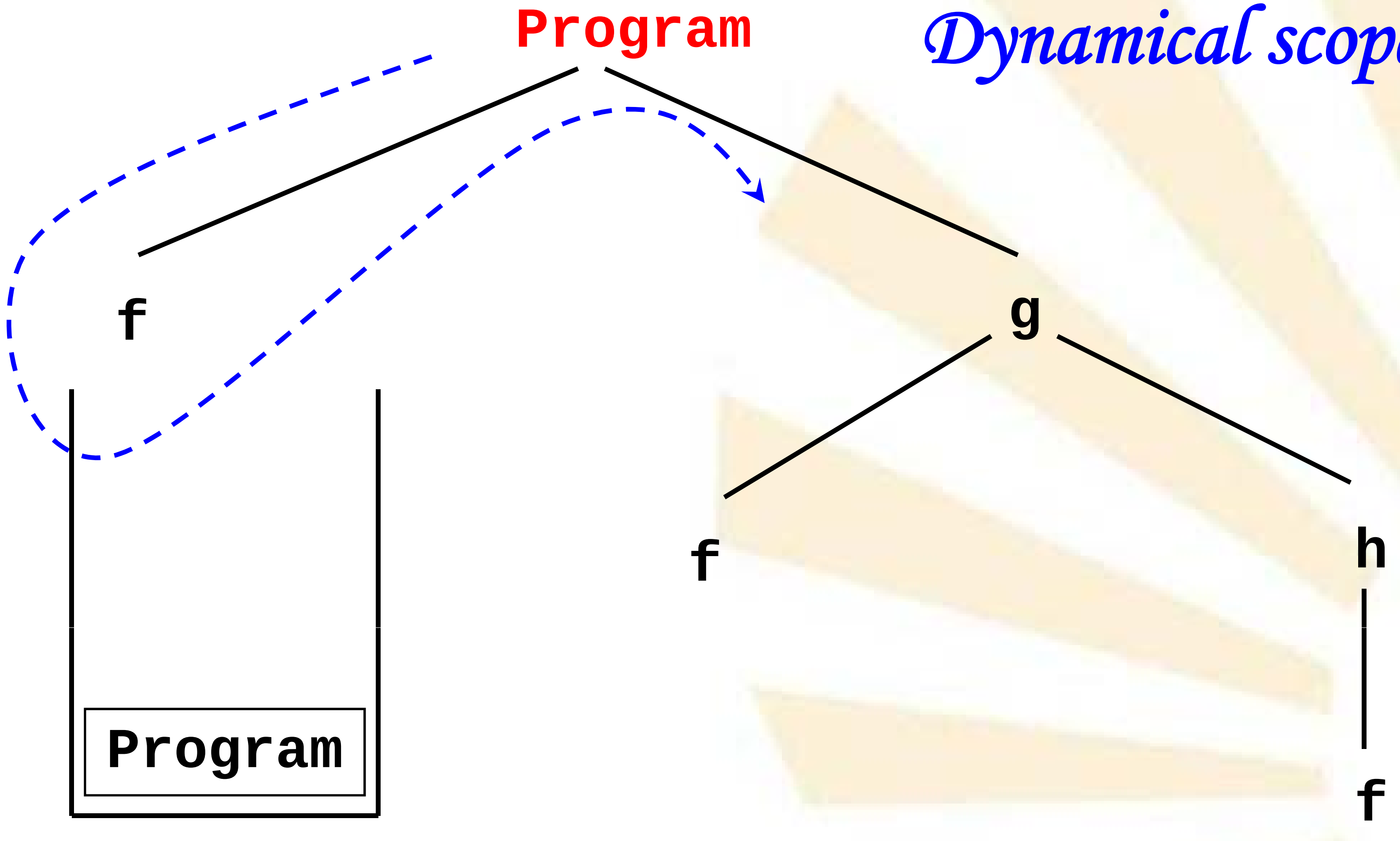
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x1 ⇐

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f ⇐
     Call to g
```

Dynamical scope

- *Use of x1 of Program in f*



Dynamical scope



Activation Stack

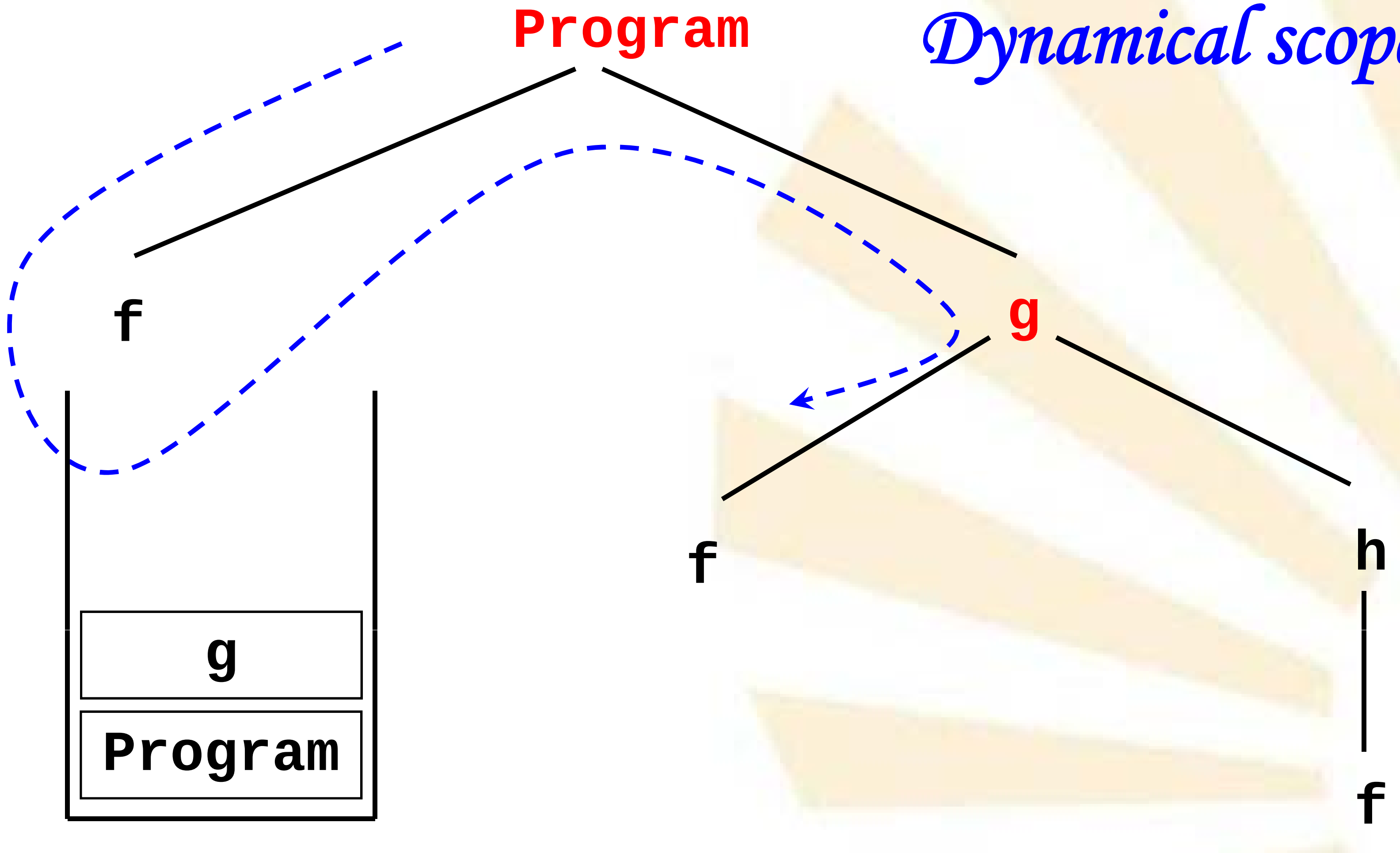
Activation Tree

Run with

dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x


    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g  <=
```

Dynamical scope



Activation Stack *Activation Tree*

Run with

dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else    Use of x

     Use of x
     Call to f
     Call to g
```

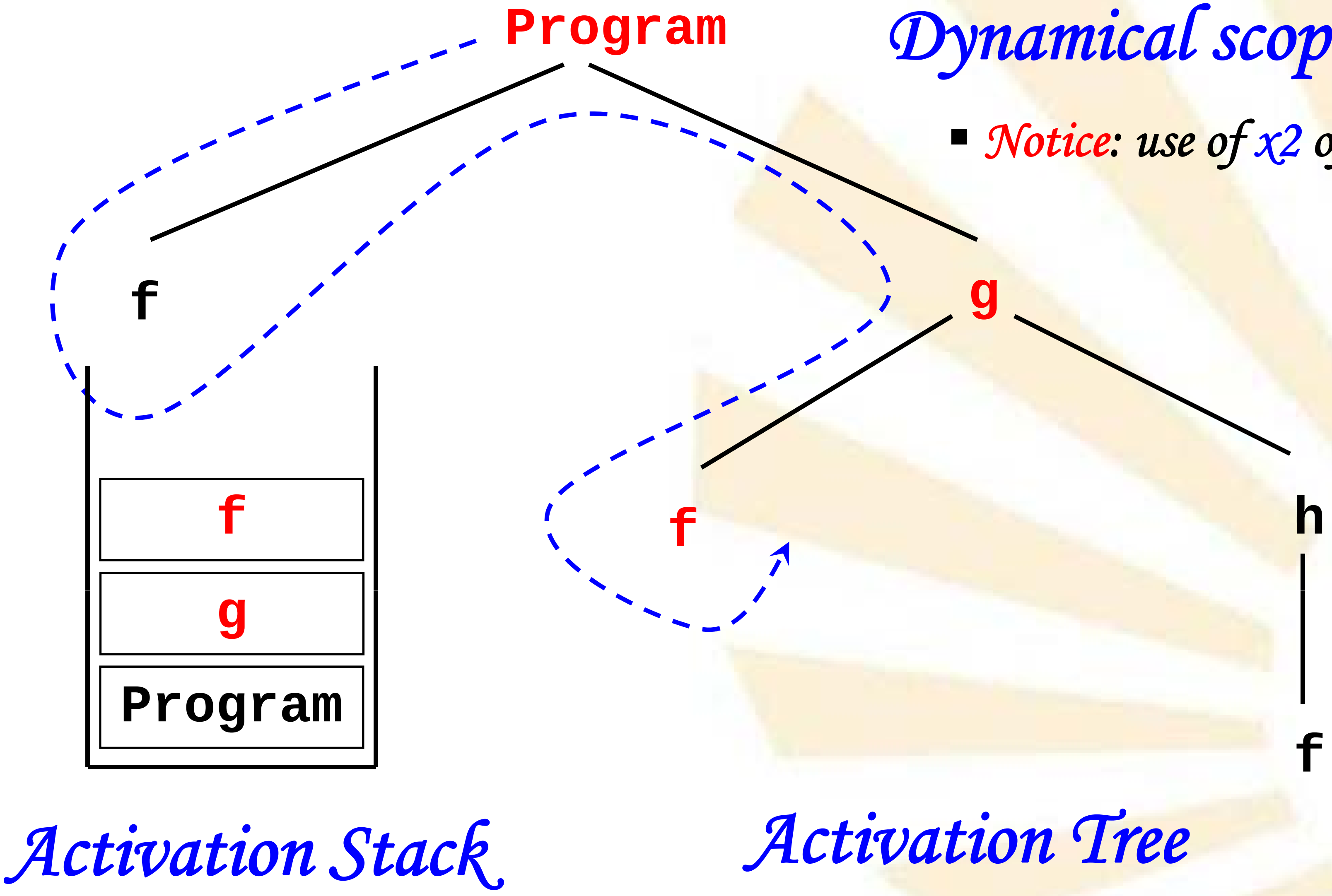
Run with

dynamical scope

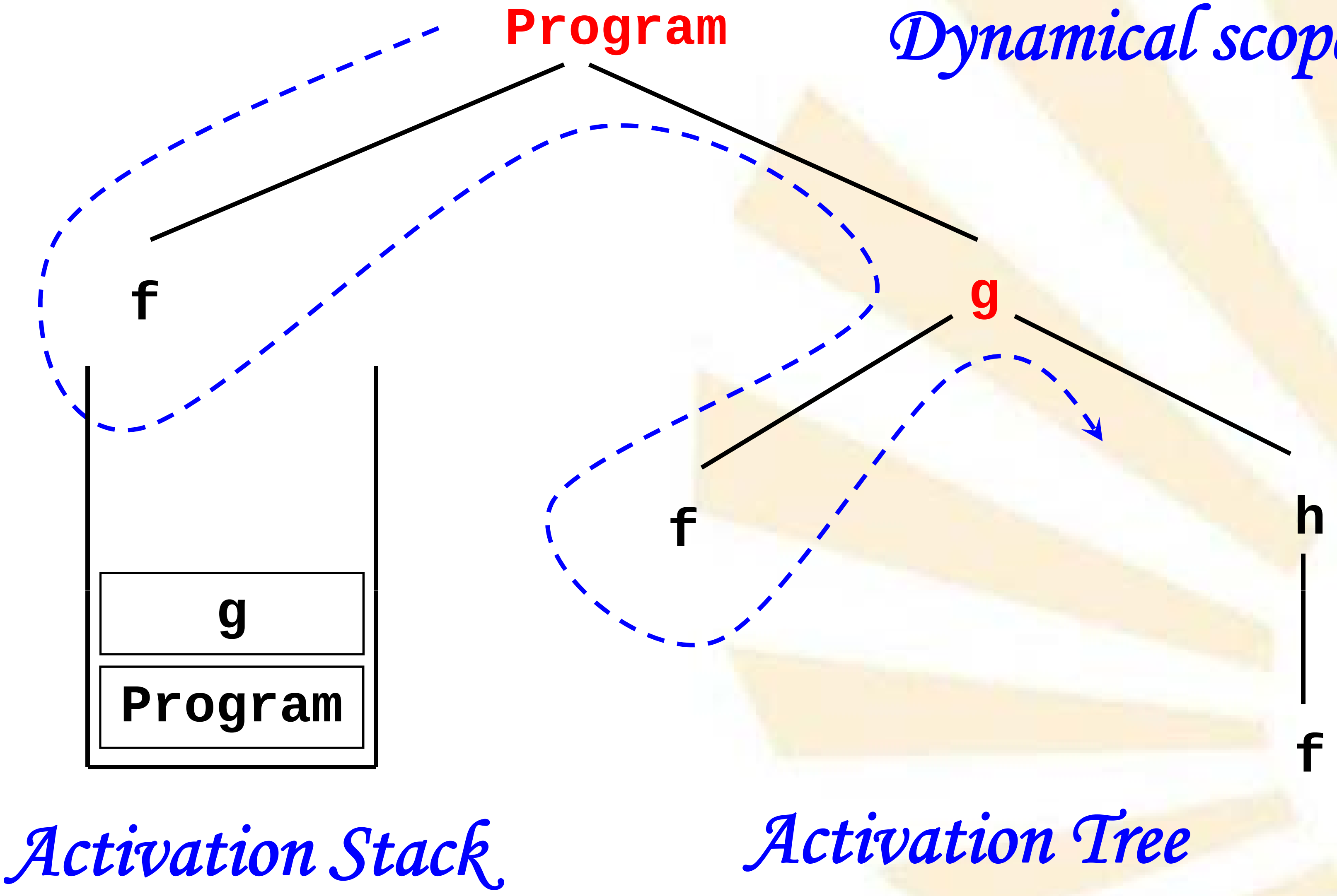
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x2

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

Dynamical scope

- *Notice: use of x2 of g in f*



Dynamical scope



Run with

dynamical scope

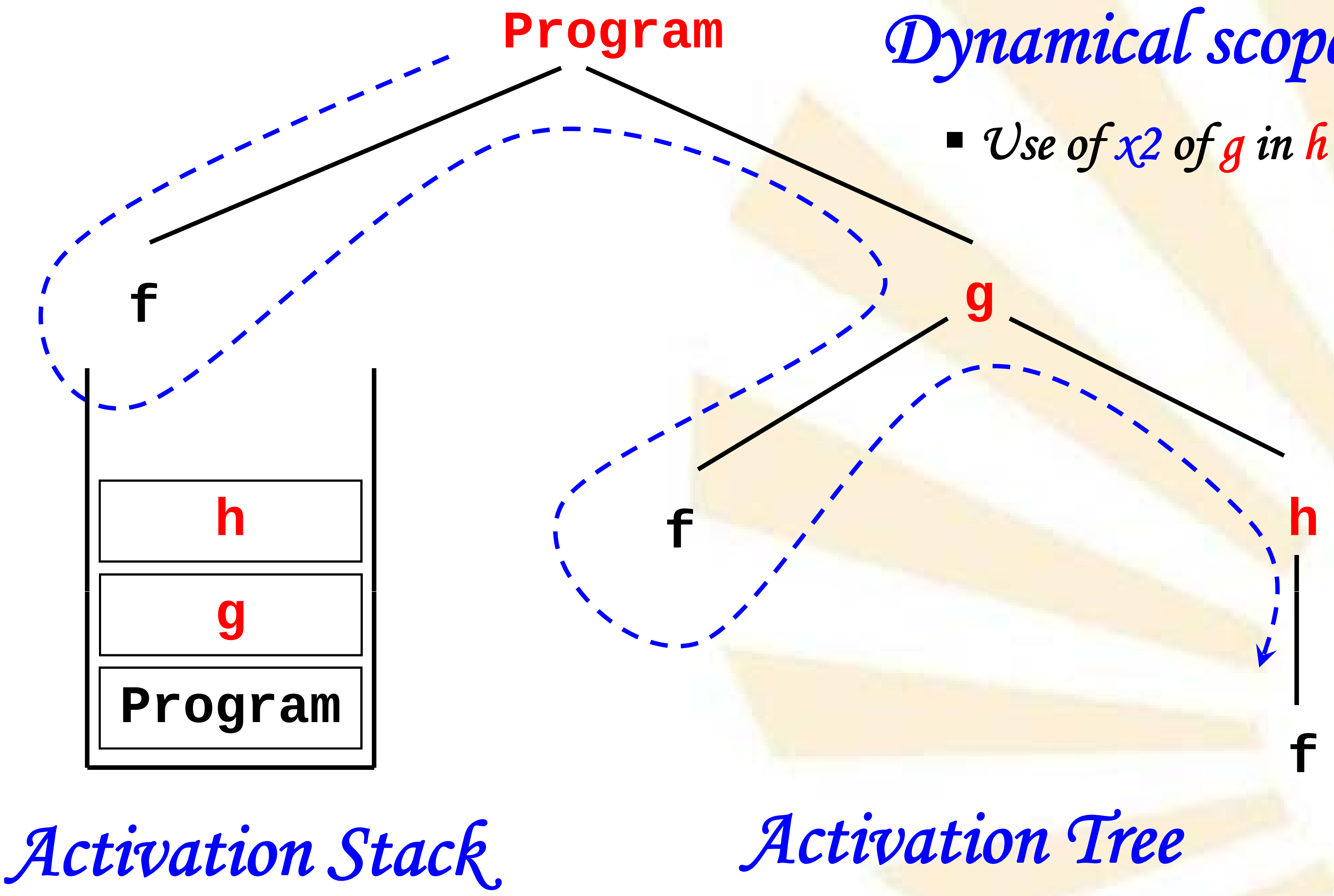
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x   (x2)

     Declaration of procedure h
      Use of x2
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else    Use of x

     Use of x
     Call to f
     Call to g
```

Dynamical scope

- *Use of x2 of g in h*



Run with

dynamical scope

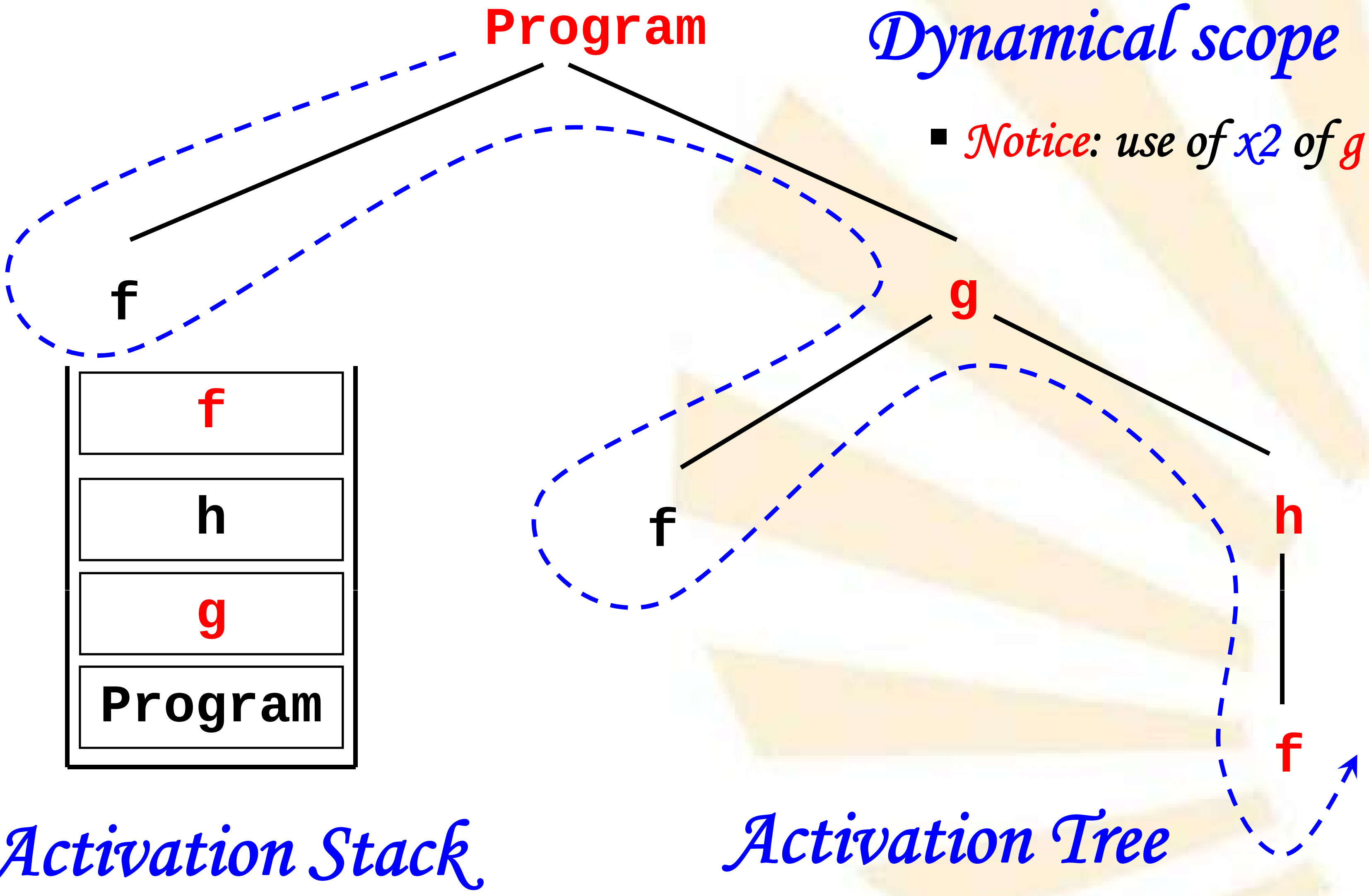
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x


    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x

     Use of x
     Call to f
     Call to g
```

Run with dynamical scope

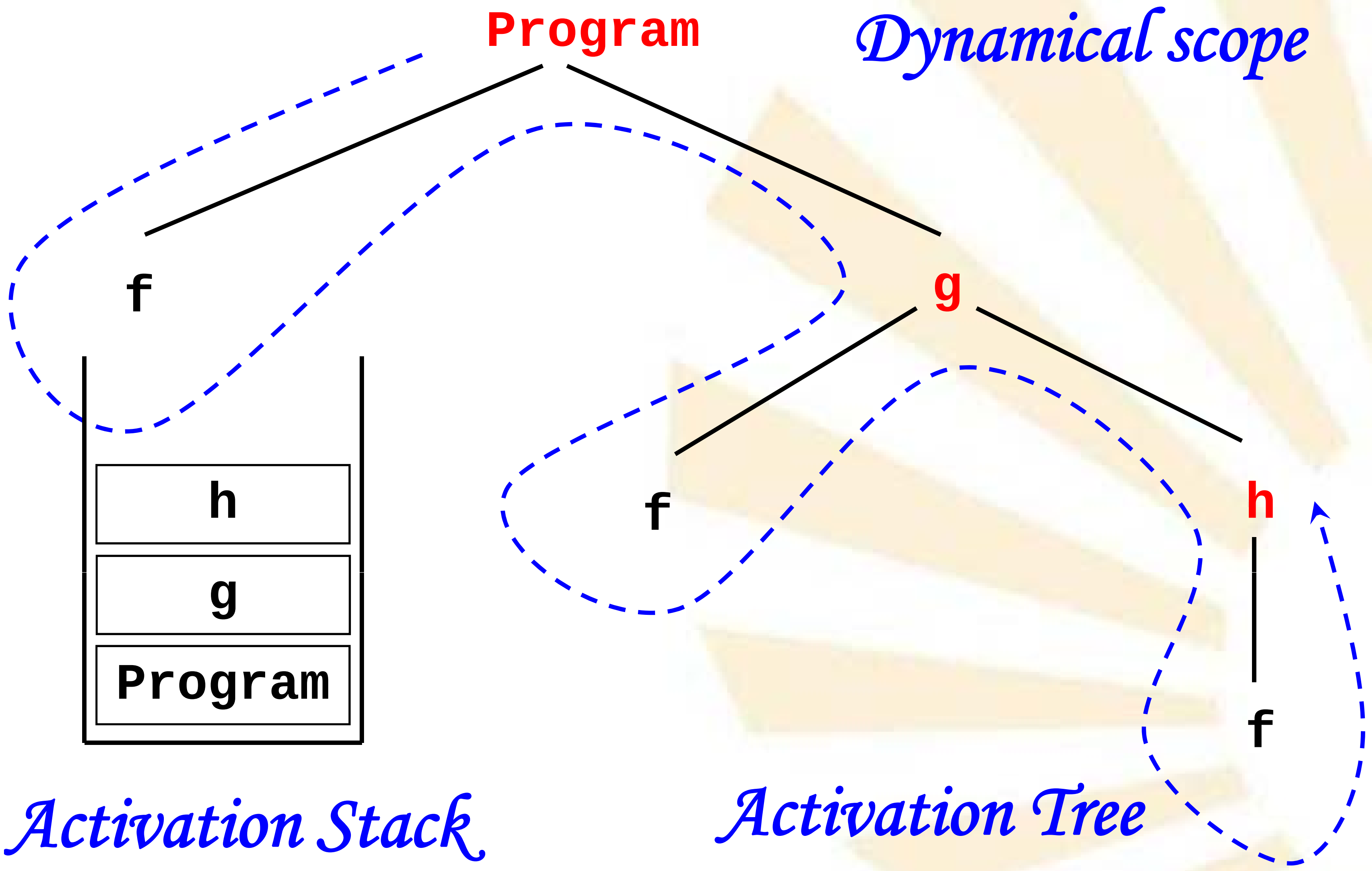
```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x2


    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else    Use of x

     Use of x
     Call to f
     Call to g
```

Dynamical scope

- *Notice: use of x2 of g in f*



Dynamical scope

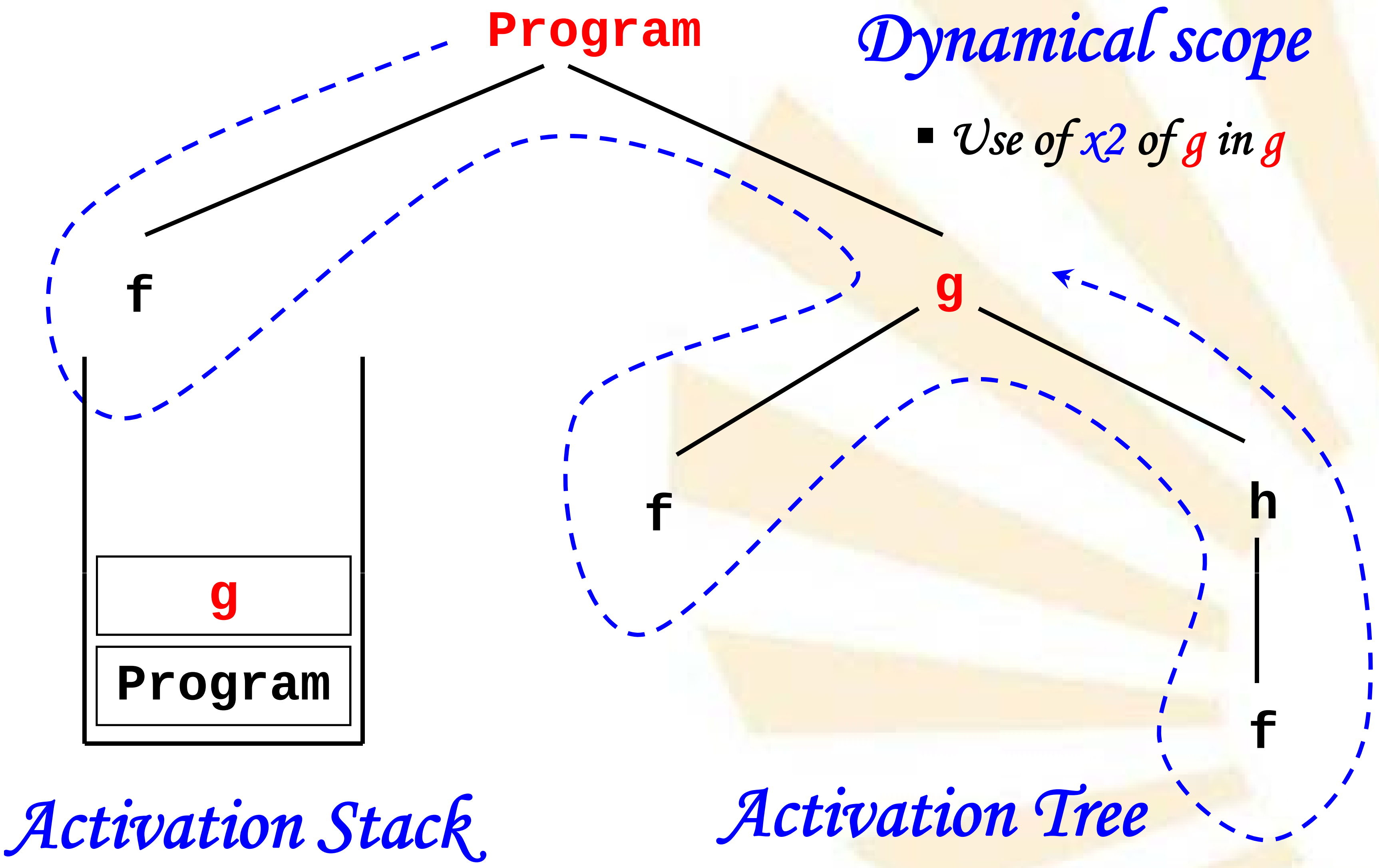


Run with dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x   (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else    Use of x2 <=

     Use of x
     Call to f
     Call to g <=
```

Dynamical scope

- Use of x2 of g in g

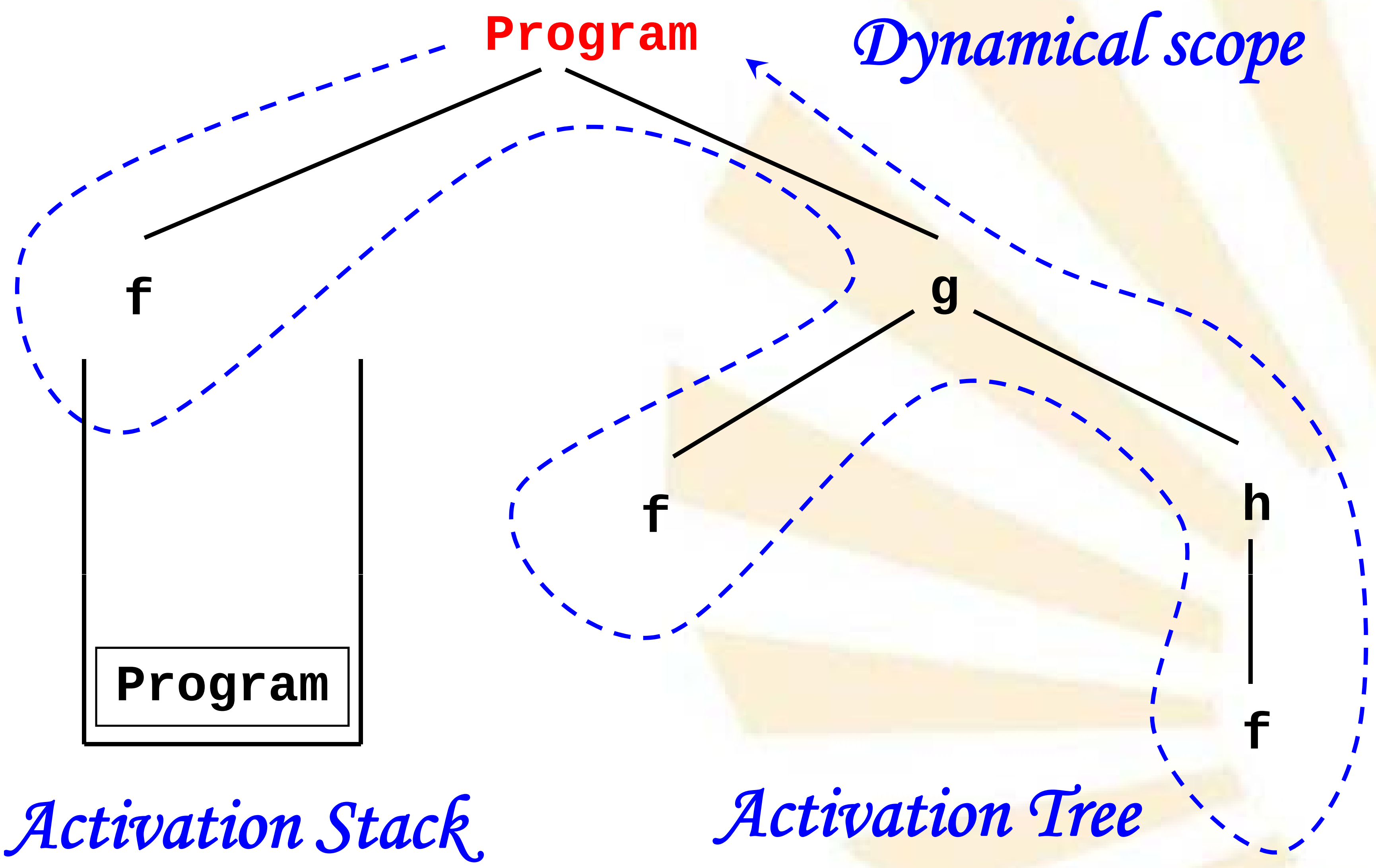


Run with

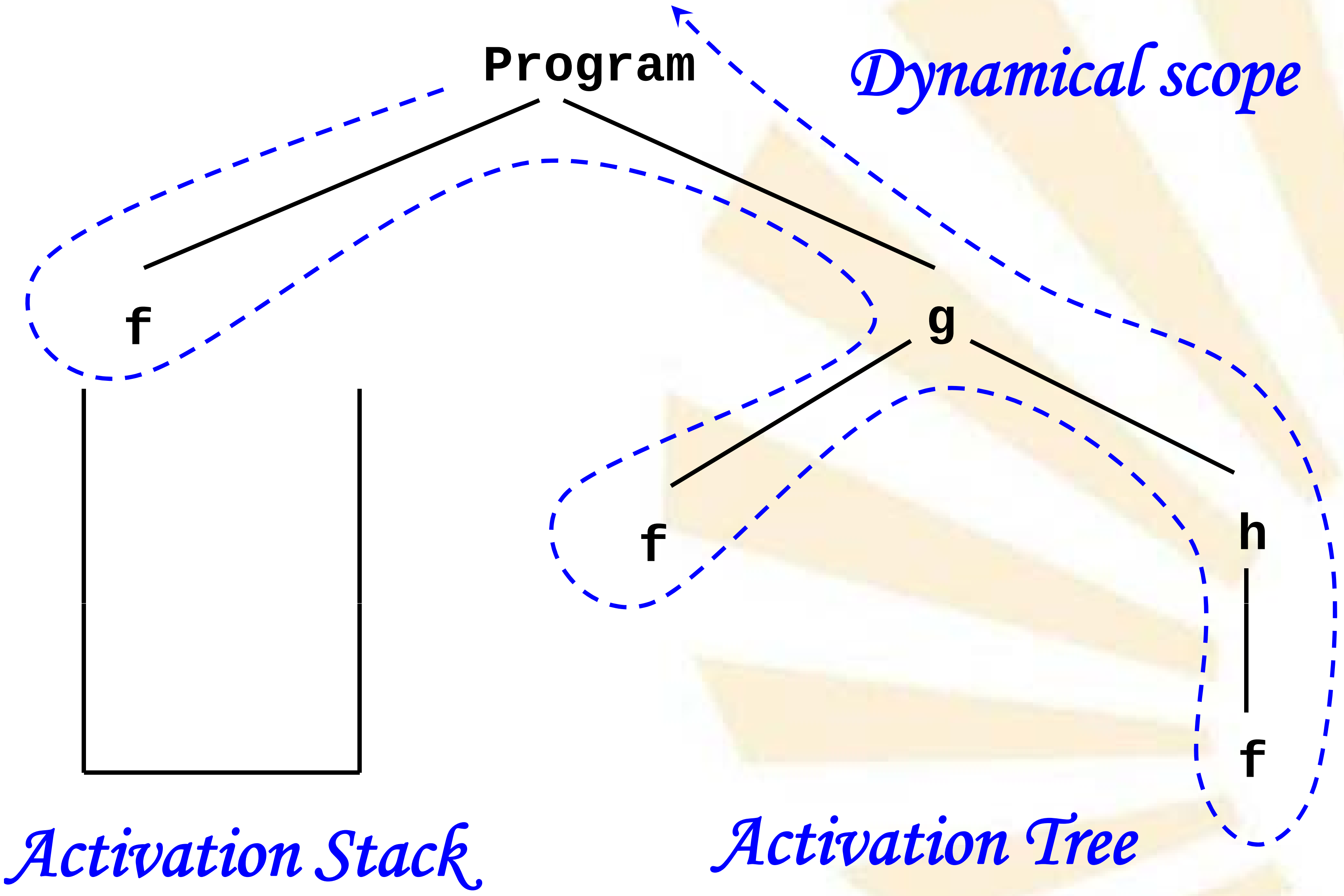
dynamical scope

```
Program
    Declaration of variable x (x1)

    Declaration of procedure f
     Use of x

    Declaration of procedure g
      Declaration of variable x  (x2)

     Declaration of procedure h
      Use of x
      Call to f

      Call to f
      Call to h
      if condition = true then Call to g
      else   Use of x2

     Use of x
     Call to f
     Call to g
```

Dynamical scope



Dynamical scope



2. Historic Summary of Scheme

- ✓ LISP
- ✓ Compilation versus Interpretation
- ✓ Dynamically versus statically scope
- ✓ Origin of Scheme

2. *Historic Summary of Scheme*

✓ *Origin of Scheme:*

- *Gerald Jay **Sussman** (MIT) and Guy Lewis **Steele** Jr.*
- ***Question:***

  *How would **LISP** be with **lexical** or **static scope** rules?*

- ***Answer:** new language → **Scheme***
  - *More **efficient** implementation of **recursion***
  - ***First class functions.***
  - *Rigorous **semantic** rules*
- ***Influence** on Common LISP: lexical scope rules*
- ***Revised^5 Report on the Algorithmic Language Scheme***

2. Historic Summary of Scheme

- ✓ Scheme:
  - ➢ Structure of scheme programs
    - ▪ Sequence of
      - - **definitions** of functions and variables
      - - and **expressions**

CÓRDOBA UNIVERSITY

SUPERIOR POLYTECHNIC SCHOOL

DEPARTAMENT OF
COMPUTER SCIENCE AND NUMERICAL ANALYSIS

ARTIFICIAL INTELLIGENCE LANGUAGES

TECHNICAL ENGINEERING IN MANAGEMENT COMPUTER SCIENCE

TECHNICAL ENGINEERING IN SYSTEMS COMPUTER SCIENCE

SECOND COURSE

FIRST FOUR-MONTH PERIOD

ACADEMIC YEAR: 2009 - 2010

DIAN
DPTO. DE INFORMÁTICA
Y ANÁLISIS NUMÉRICO