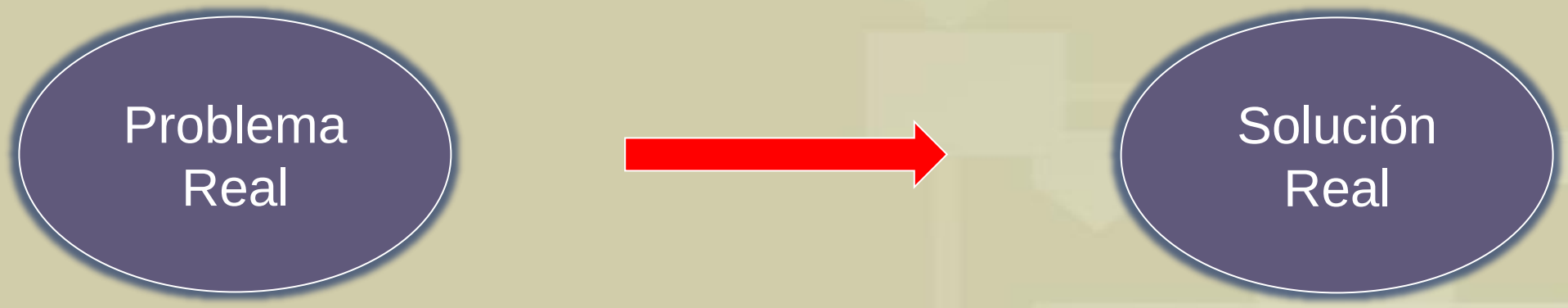
Introducción al Cómputo

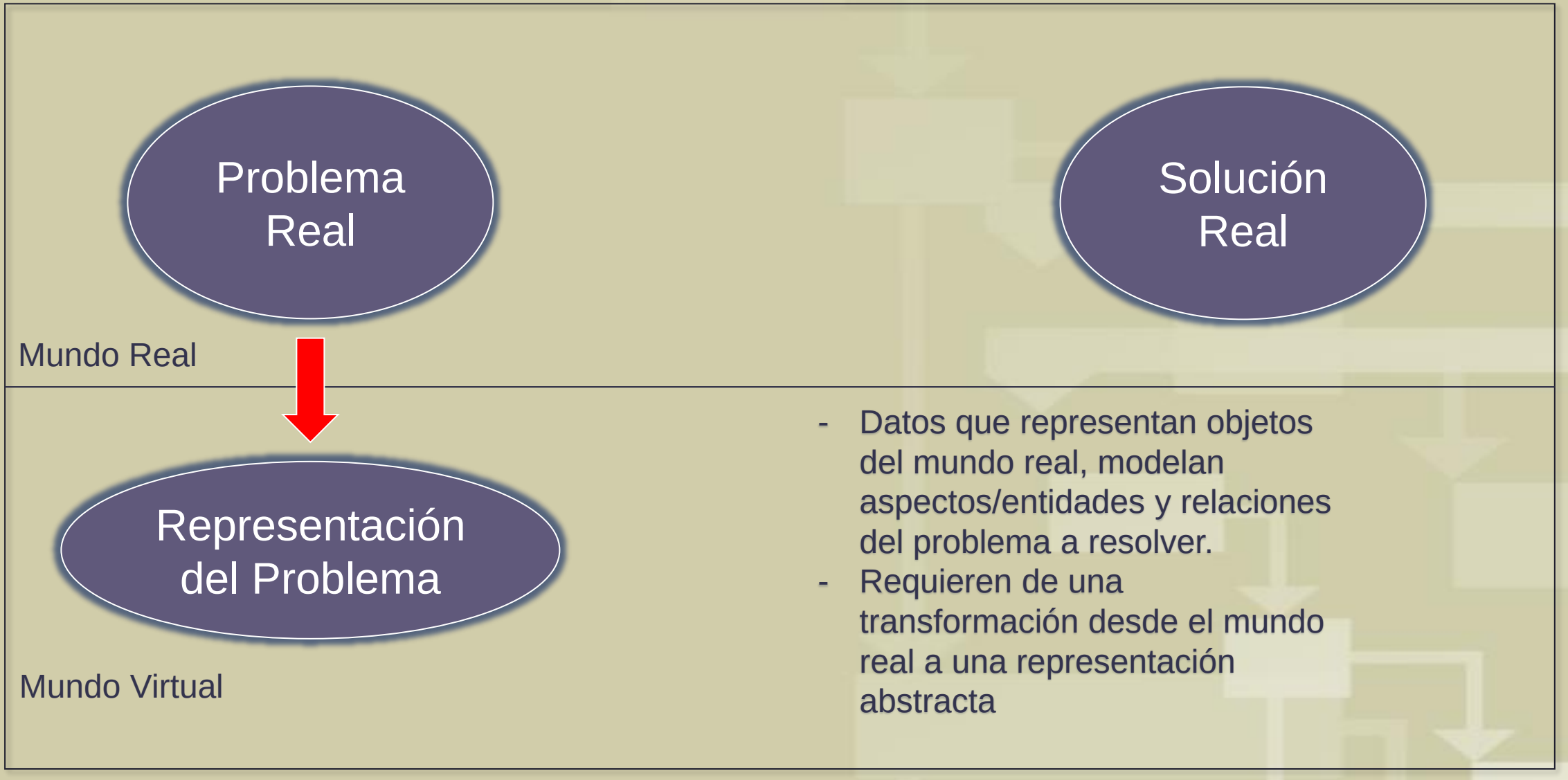
Algoritmos y Resolución de Problemas

Resolución de un Problema



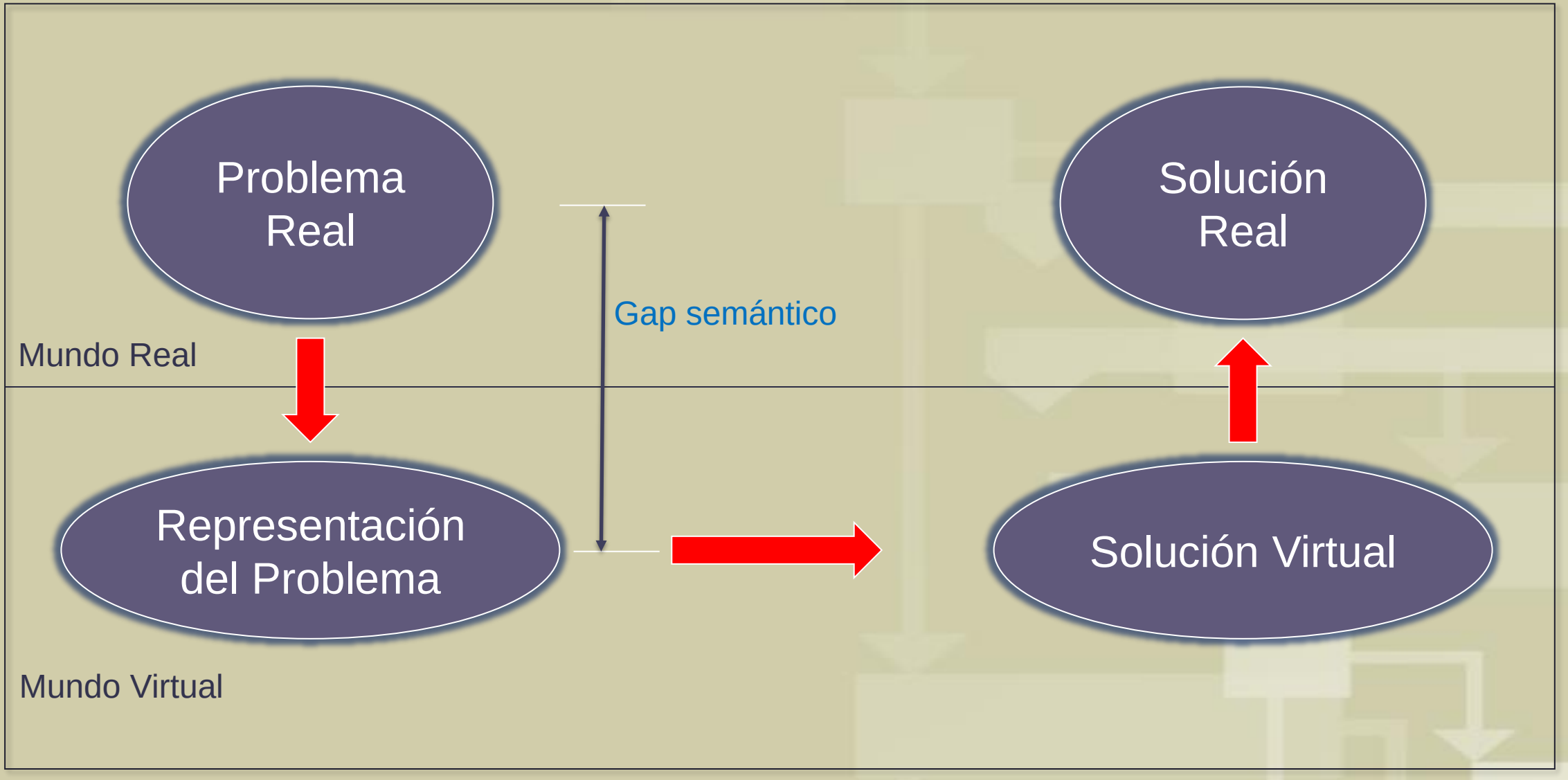
Mundo Real

Resolución de un Problema



- Datos que representan objetos del mundo real, modelan aspectos/entidades y relaciones del problema a resolver.
- Requieren de una transformación desde el mundo real a una representación abstracta

Resolución de un Problema

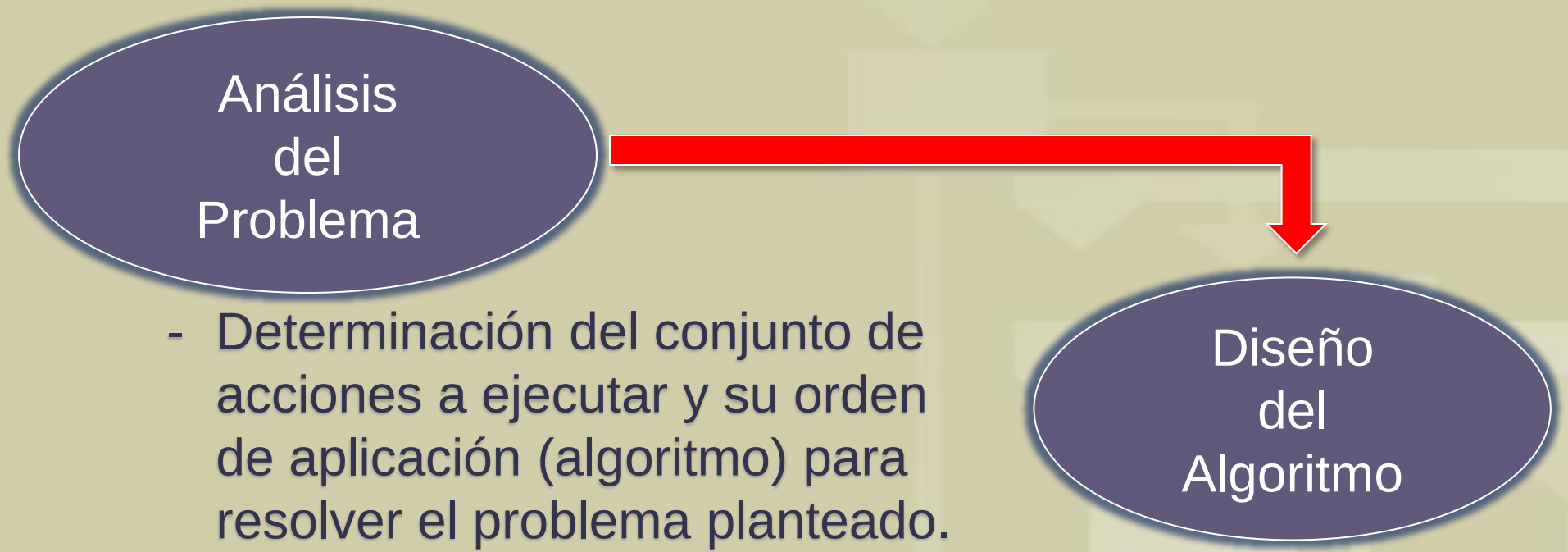


Resolución de un Problema

Análisis del Problema

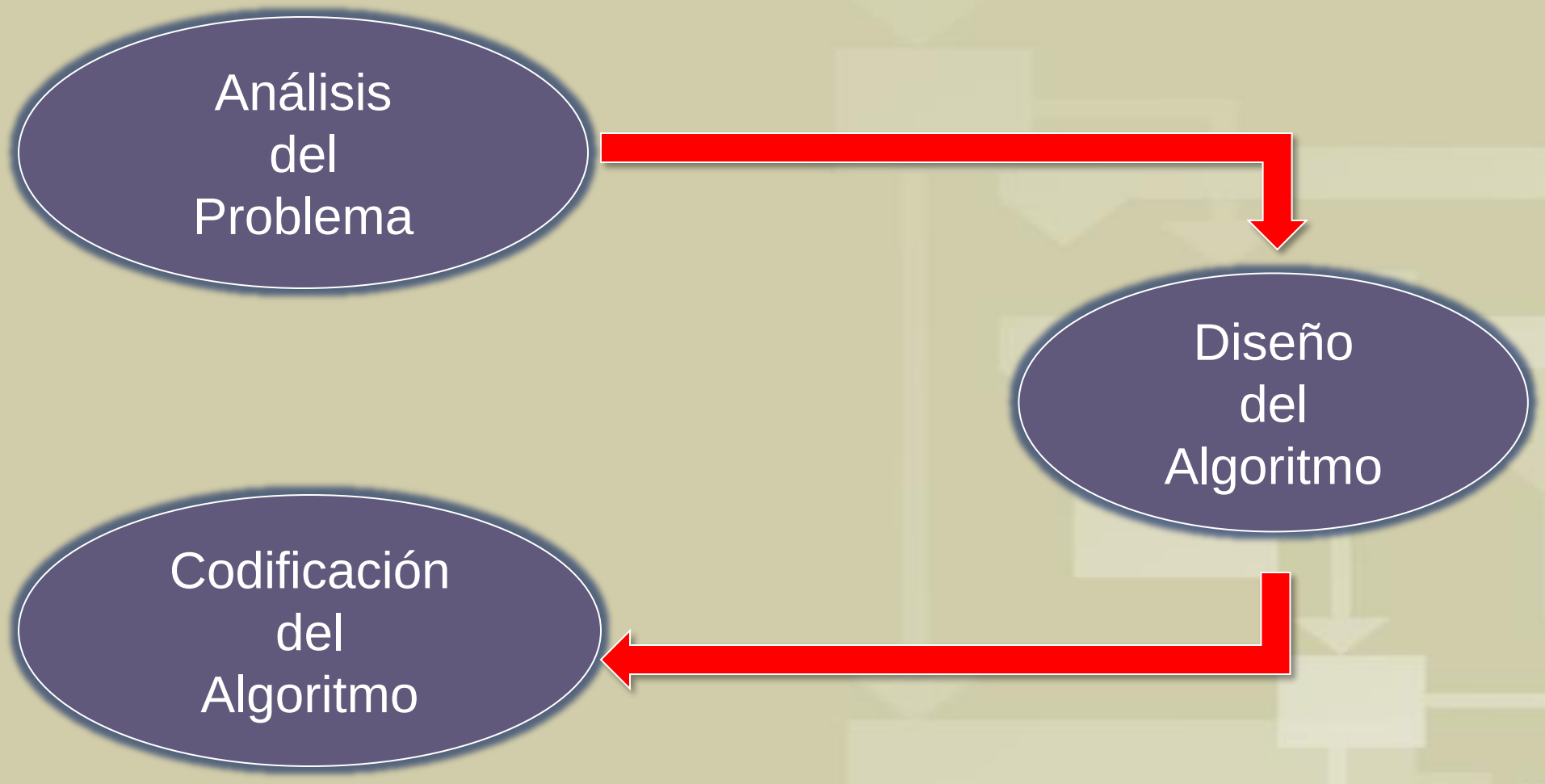
- Estudio detallado del problema.
- Especificación de los requerimientos que debe cumplir la solución.
- En ésta etapa se determina **qué** deberá hacer el programa

Resolución de un Problema



- Determinación del conjunto de acciones a ejecutar y su orden de aplicación (algoritmo) para resolver el problema planteado.
- En ésta etapa se determina **cómo** el programa hará la tarea solicitada

Resolución de un Problema



Algoritmo:

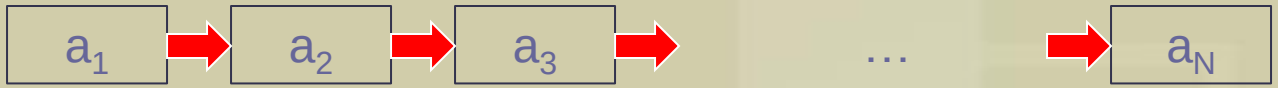
Procedimiento a seguir para resolver un problema definido en términos de:

1) Las acciones a ejecutar
2) El orden de dichas acciones

Características

- Preciso (se indica el orden de realización en cada paso)
- Definido (se repite el resultado)
- Finito (número determinado de pasos)
- Robusto (contempla todas las posibles facetas del problema)
- Eficaz (logra su objetivo)
- Eficiente (minimiza el uso de recursos)

Control de flujo: Secuencial



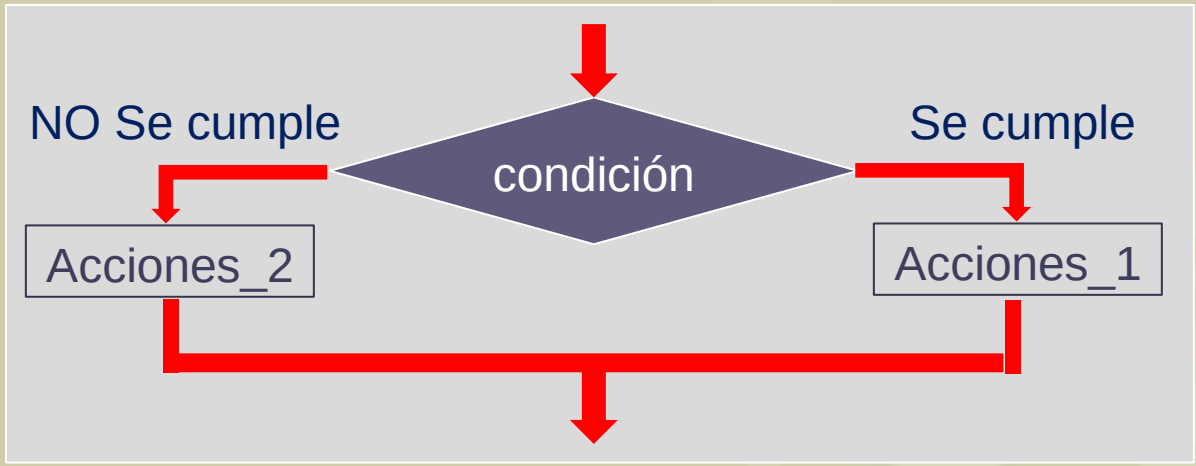
Pseudocódigo:

```
acción_1
acción_2
...
acción_N
```

Ejemplo (promedio de 3 números):

```
mostrar("Ingrese N1: ");
N1 = leeEntero();
mostrar("Ingrese N2: ");
N2 = leeEntero();
mostrar("Ingrese N3: ");
N3 = leeEntero();
Prom = (N1 + N2 + N3)/3;
mostrar("Prom = ", Prom);
```

Control de flujo: Condicional



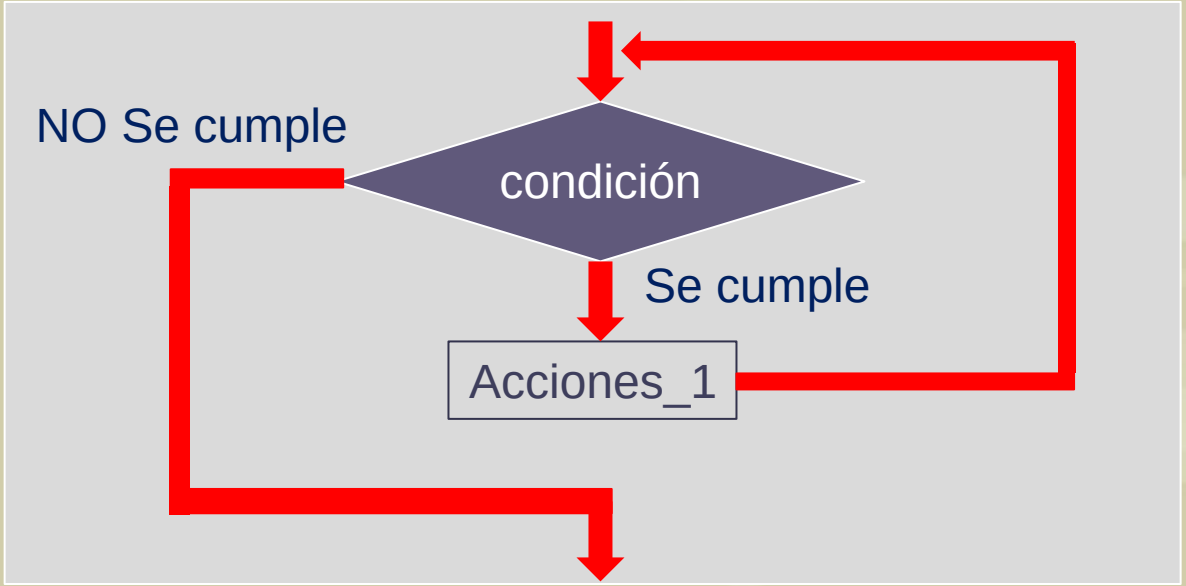
Pseudocódigo:

```
…
si condición hacer
    acciones_1
sino hacer
    acciones_2
fin si
```

Ejemplo:

```
…
si x < 100 hacer
  mostrar("es barato");
sino hacer
  mostrar("es caro");
fin si
```

Control de flujo: Repetitivo




```
Pseudocódigo:
   …
   mientras condición hacer
       acciones_1
   fin mientras
```

```
sum = x = 0;
mientras x < 100 hacer
   sum = sum + x;
   x = x + 1;
fin mientras
mostrar("la suma es", sum);
```

Resolución de un Problema

MODELA,
DIVIDE
Y
CONQUISTA

Modelado de un problema

El modelado de un problema consiste en identificar las entidades involucradas en el problema

- Sus propiedades o atributos (representación).
- El comportamiento que poseen.
- La funcionalidad (interface) que exponen.
- La interacción existente entre esas entidades.

Descomposición de un problema

Dentro del proceso de resolución de problemas, la identificación de subproblemas cumple un rol fundamental (tarea de diseño) y sienta las bases para la descomposición del problema.

La pericia de un programador no está en ser veloz para escribir líneas de programa, sino en saber descubrir, en el proceso de diseño, cuáles son las partes del problema, y luego resolver cada una de ellas abstrayéndose de las otras.

Las funciones definen la abstracción a la solución de cada subproblema individual.

Funciones

Funciones como cajas negras:


```
// función que retorna verdadero si el valor de X es múltiplo de Y,
// o retorna falso en caso contrario

esMultiplo( X, Y )
    si modulo(X, Y) = 0
        retornar verdadero;
    sino
        retornar falso;
    fin si
fin esMultiplo
```

Funciones: Ejemplo

```
// Suma de los primeros N números naturales,
// retorna su resultado

Sumar( N )
      suma = 0;
      índice = 1;
      mientras índice <= N hacer
            suma = suma + índice;
            índice = índice + 1;
      fin mientras
      retornar suma;
fin Sumar
```

Ordenamiento

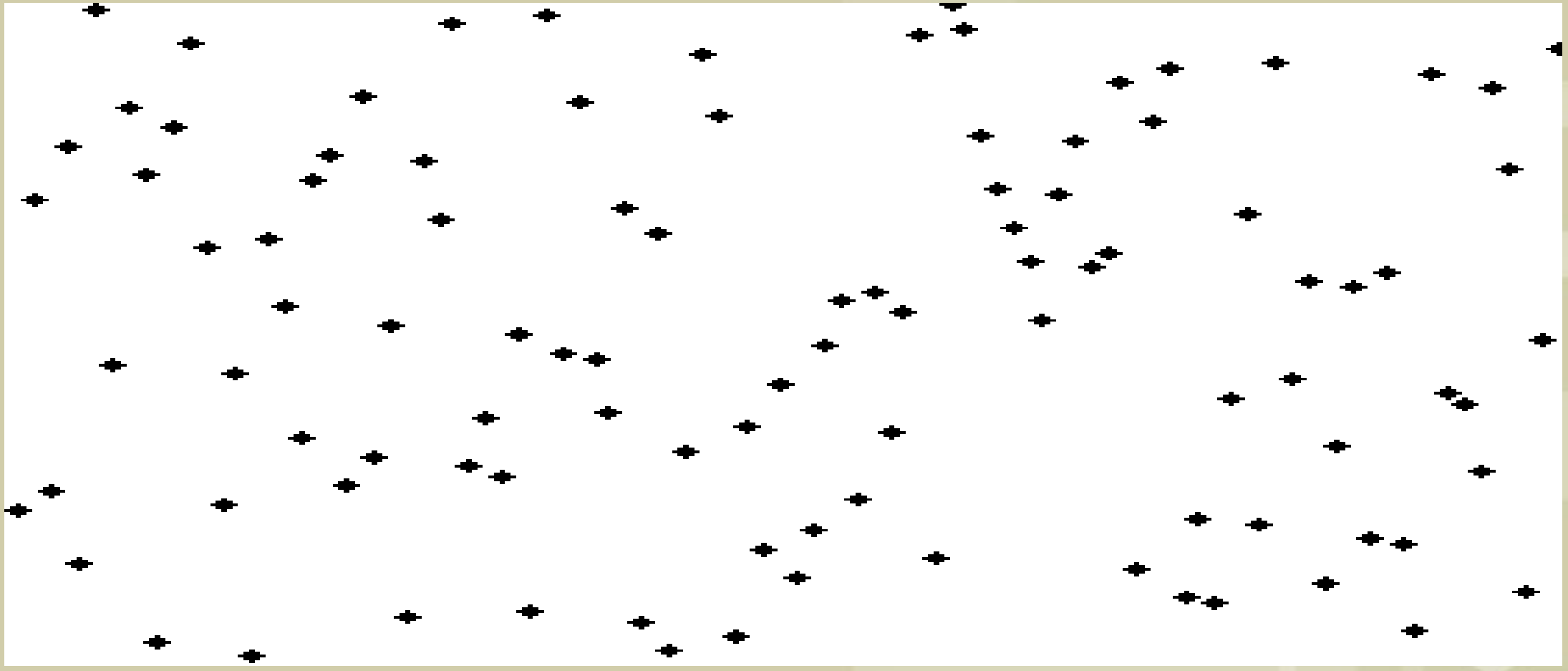
El problema consiste en, dado un conjunto A, ordenar los elementos con algún criterio, por ejemplo que $A_i <= A_{i+1}$

A_0	A_1	A_2	A_3						A_{N-1}

Existen un gran número de algoritmos que resuelven este problema

Ordenamiento

Método de Inserción



Inserción Ordenada

```
// función que ordena el vector A

ordenarPorInsercion(A)
  para i desde 1 hasta tamaño(A)
    insertarOrdenado (A(i), A, i-1);
  fin para
fin ordenarPorInsercion
```

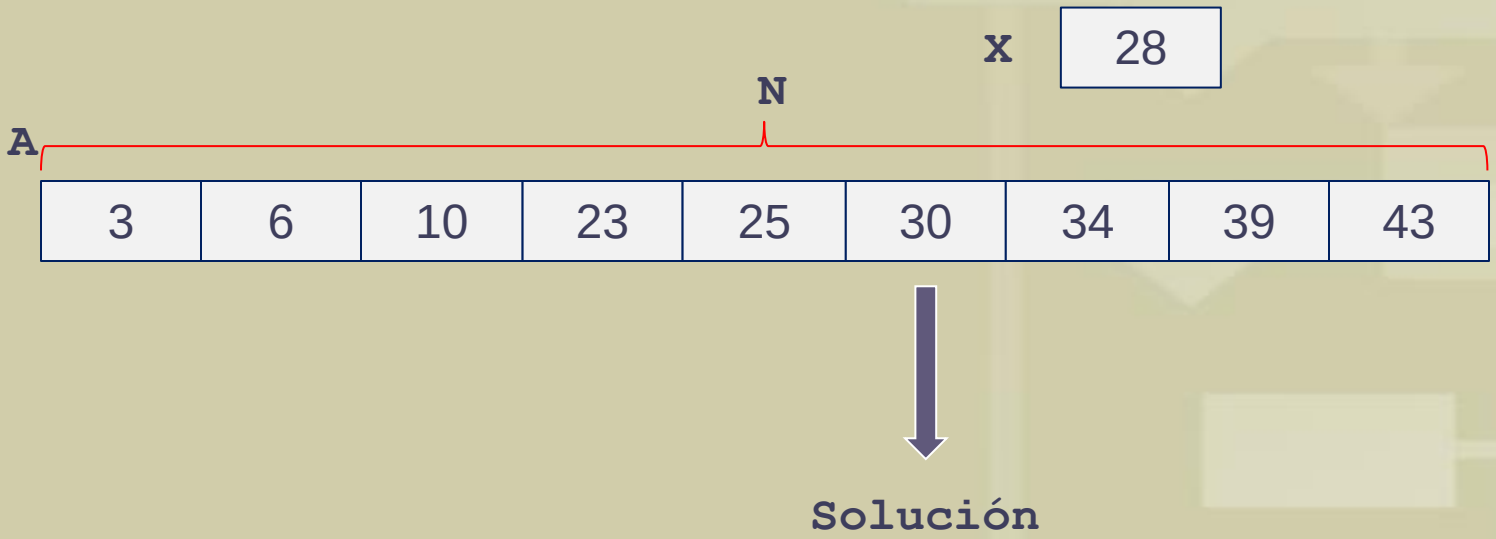
Inserción Ordenada

```
// procedimiento que inserta X en
// forma ordenada dentro del conjunto
// A que tiene N elementos ordenados

insertarOrdenado (X, A, N)
     idx = puntoInsercion(X, A, N);
     desplazar(A, N, idx);
     A(idx) = X;
fin insertarOrdenado
```

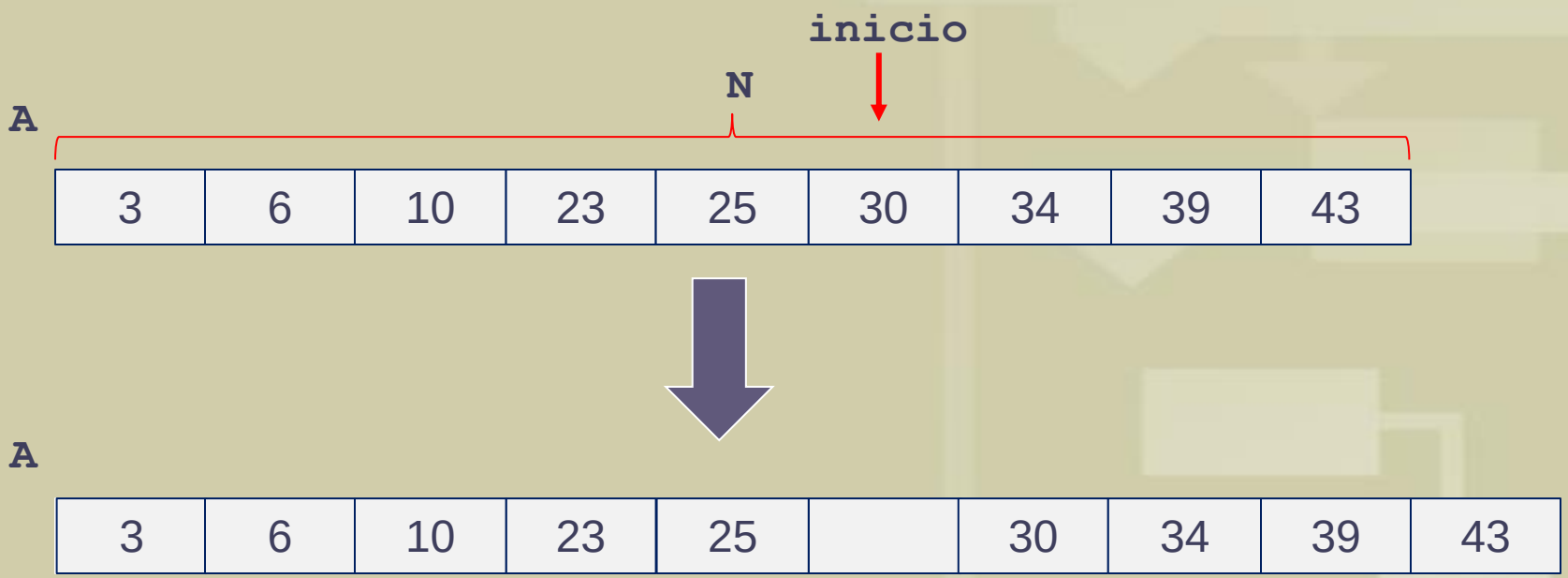
Inserción Ordenada

puntoInsercion(X, A, N)



Inserción Ordenada

desplazar(A, N, inicio)



Inserción Ordenada

```
// procedimiento que busca el punto
// correcto para insertar ordenada-
// mente a X dentro del conjunto
// A que tiene los primeros N
// elementos ordenados
puntoInsercion(X, A, N)
     idx = 1;
     mientras idx <= N Y X > A(idx)
          idx = idx + 1;
     fin mientras
     retornar idx;
fin puntoInsercion
```

Inserción Ordenada

```
// procedimiento que desplaza a todos
// los elementos de A, de tamaño N,
// en una posición a partir de la
// componente inicio

desplazar(A, N, inicio)
     idx = N+1;
     mientras idx >= inicio
          A(idx) = A(idx - 1);
          idx = idx - 1;
     fin mientras
fin desplazar
```

QuickSort

```
quicksort(A)
    si tamaño(A) ≤ 1
        retornar A ;
    fin si
    seleccionar y remover un elemento PIVOT de A
    para cada X en A
        si X ≤ PIVOT
            agregar X a MENORES;
        sino
            agregar X to MAYORES;
        fin si
    fin para
    retornar Unir(quicksort(MENORES),
                  PIVOT,
                  quicksort(MAYORES));
Fin quicksort
```

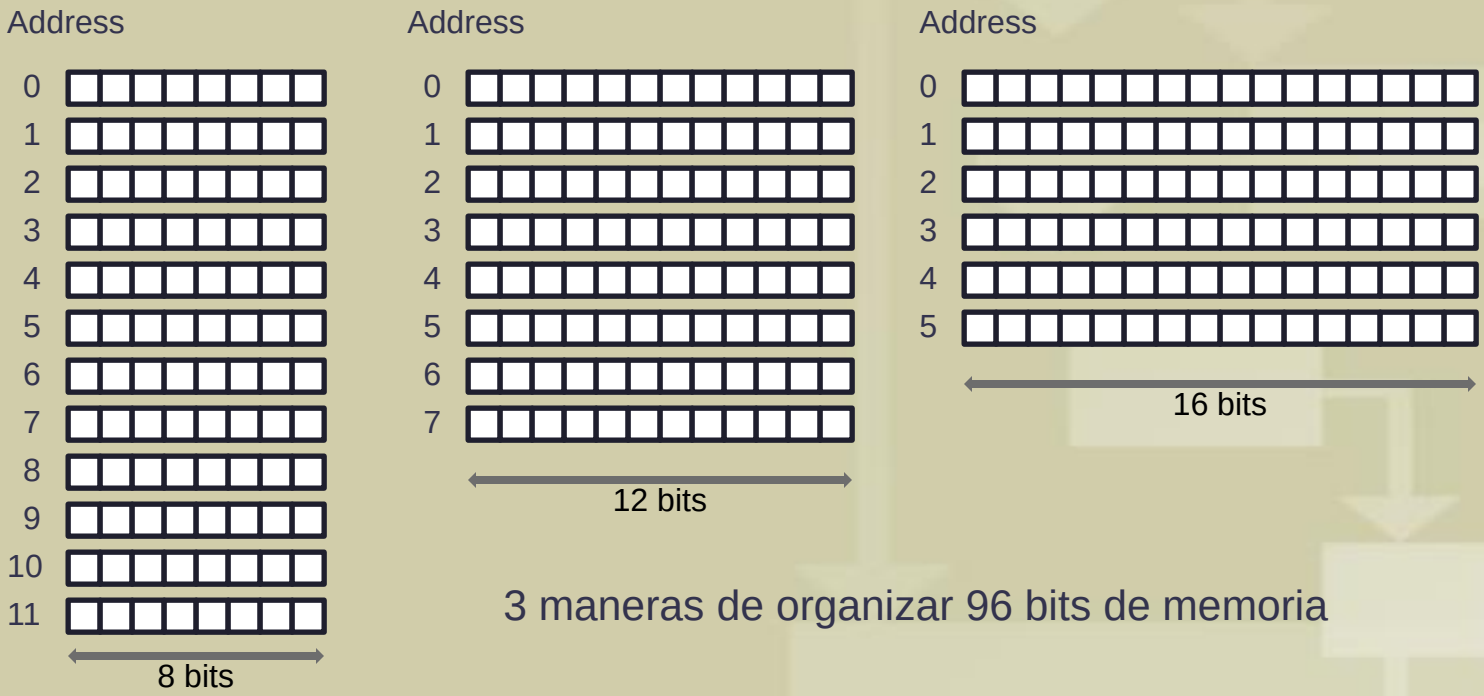
Comparación

tamaño	n2	t [seg]		n.log(n)	t [seg]	
10	100	0.1		33	0.03	
100	10000	10		664	0.66	
1000	1000000	1000		9966	9.97	
1000000	1E+12	1E+09		19931569	19,931.57	
		11574.07	dias		5.54	Horas
		31.70979	años			

Representación

Organización de la memoria RAM

Unidad de memoria: dígito binario, **bit**
1 byte = 8 bits 1 nibble = 4 bits
1 word = 2, 4 u 8 bytes = 16, 32 ó 64 bits



3 maneras de organizar 96 bits de memoria

Números de precisión finita

Enteros positivos de 3 dígitos decimales, sin punto decimal, sin signo.

Exactamente 1000 miembros: *000*, *001*, *002* … *999*

Con estas restricciones, hay números imposibles de representar:

- Número mayores a *999*
- Números negativos
- Fracciones
- Números irracionales
- Números complejos

Números de precisión finita

Se pierde la propiedad aritmética de cierre con respecto a la suma, resta y multiplicación:

600 + 600 = 1200 (muy grande)
003 – 005 = -2 (negativo)
050 x 050 = 2500 (muy grande)
007 / 002 = 3.5 (no es entero)

Violaciones de 2 tipos (mutuamente exclusivas):

- Operaciones cuyo resultado es mayor que el número más grande (error de overflow) o menor que el más chico (error de underflow).

- Operaciones cuyo resultado no es ni muy grande ni muy chico, si no que simplemente no es un número que pertenece al conjunto de los números representables

Números de precisión finita

Ley asociativa:

$a + (b - c) = (a + b) - c$

Para $a = 700$, $b = 400$ y $c = 300$, el primer término da 800 y el segundo da overflow al evaluar $(a + b)$.

El orden de las operaciones es importante.

Ley distributiva:

$a \times (b - c) = a \times b - a \times c$

Para $a = 5$, $b = 210$ y $c = 195$, el primer término da 75 y el segundo da overflow al evaluar $a \times b$.

Parecería por estos ejemplos que las computadoras son inapropiadas para hacer cálculos aritméticos debido a la naturaleza finita de la representación numérica. Esta conclusión es obviamente falsa, pero los ejemplos muestran la importancia de entender como funcionan las computadoras en lo que hace a su representación y sus limitaciones.

Sistemas de numeración

Número decimal, en base 10:

2009.14

Centenas ↓ Decenas ↓ Unidades ↓ Punto decimal Décimas ↓ Centésimas ↓

$d_n \quad \ldots \quad d_2 \quad d_1 \quad d_0 \quad . \quad d_{-1} \quad d_{-2} \quad \ldots \quad d_{-k}$

$$\text{Número} = \sum_{i=-k}^{n} d_i \, 10^i$$

$$2009.14 = 2 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 9 \times 10^0 + 1 \times 10^{-1} + 4 \times 10^{-2}$$

Sistemas de numeración

Utilizando computadoras, es frecuente utilizar otras bases distintas a 10.

Las más utilizadas son 2, 8 y 16. Estos sistemas de numeración son llamados binario, octal y hexadecimal.

Para un sistema de base *k*, se requieren *k* símbolos diferentes para representar los dígitos de *0* a *k-1*.

Sistema decimal:	*0 1 2 3 4 5 6 7 8 9*
Sistema binario:	*0 1*
Sistema hexadecimal:	*0 1 2 3 4 5 6 7 8 9 A B C D E F*

$2009_{10} = 11111011001_2 = 3731_8 = 7D9_{16}$

$11111011001_2 = 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 + 2^0$

$7D9_{16} = 7 \times 16^2 + D \times 16^1 + 9 \times 16^0 = 7 \times 256 + 13 \times 16 + 9$

Números binarios negativos

- Distintas alternativas:
  - Magnitud con signo: el bit de más a la izquierda se utiliza como signo, 0 es positivo y 1 es negativo, el resto de los bits representan el valor absoluto del número.
  - Complemento a 1: el bit de más a la izquierda es el signo, 0 es positivo y 1 es negativo, para negar un número se reemplaza cada 1 por 0 y cada 0 por 1.
  - Complemento a 2: ídem con el bit de signo, para negar un número se reemplaza cada 1 por 0 y cada 0 por 1 (como en complemento a 1) y luego se le suma 1 al resultado.
  - Excess 2^{m-1} (binary offset): para un número con *m* bits, el número se representa como la suma del mismo con 2^{m-1} . Para un número de 8 bits, se le suma *128*,
    ejemplo *-3 → 128 + (-3) = 125*.

Números binarios negativos

00000110	+6
11111001	-6 en complemento a 1
11111010	-6 en complemento a 2
01111010	-6 en excess 128

00010001	+17
11101110	-17 en complemento a 1
11101111	-17 en complemento a 2
01101111	-17 en excess 128

01111111	+127
10000000	-127 en complemento a 1
10000001	-127 en complemento a 2
00000001	-127 en excess 128

No existe	+128
No existe	-128 en complemento a 1
10000000	-128 en complemento a 2
00000000	-128 en excess 128

Problemas:

- dos representaciones distintas para 0
- diferentes cantidades de números positivos y negativos.

Aritmética binaria

- Suma de bits:

	0	0	1	1
	+ 0	+ 1	+ 0	+ 1
Suma	0	1	1	0
Carry	0	0	0	1

- Suma de números binarios:

Se empieza la suma con los bits de más a la derecha y se va llevando el bit de carry, como con los números decimales. Si se usa representación de complemento a 1, el carry del bit de más a la izquierda se vuelve a sumar al resultado, en complemento a 2 se descarta.

Decimal	Compl. 1	Compl. 2
10	00001010	00001010
+ (-3)	+ 11111100	+ 11111101
+7	1 00000110	1 00000111
	+ 1	
	00000111	Se descarta

Operaciones entre bits

- OR – “|” :

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

- XOR – “^” :

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

- AND – “&” :

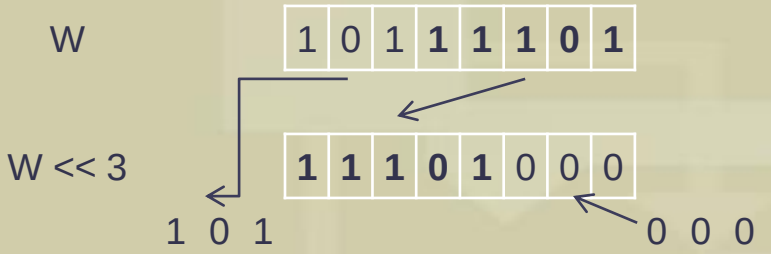
a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

- NOT – “~”

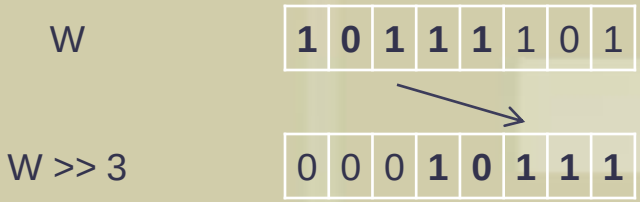
a	~a
0	1
1	0

Operaciones entre bits

- Left shift – “<<” :
(desplazar la representación hacia la izquierda)



- Right shift – “>>” :
(desplazar la representación hacia la derecha)



Números de punto flotante

$$n = f \times 10^{e}$$

f es la fracción o mantisa
e es el exponente

3.14	= 3.14	x 10^{0}	= 0.314	x 10^{1}
0.000001	= 1.0	x 10^{-6}	= 0.1	x 10^{-5}
2009	= 2.009	x 10^{3}	= 0.2009	x 10^{4}

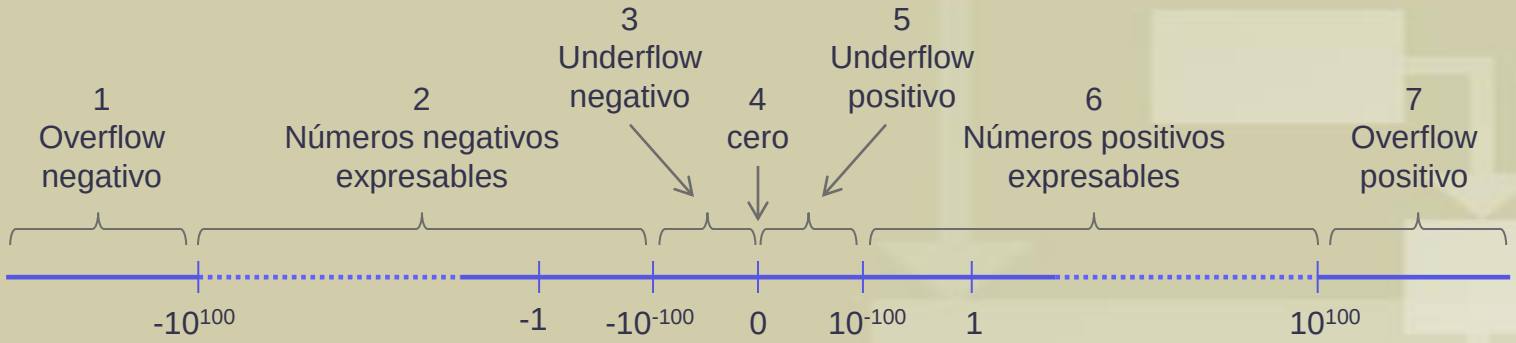
Normalizado: $f = 0$ ó $0.1 \leq |f| < 1$

El *rango* está determinado por la cantidad de dígitos del exponente.
La *precisión* está determinada por la cantidad de dígitos en la fracción.

Números de punto flotante

Representación con 3 dígitos decimales con signo para la fracción y 2 dígitos con signo para el exponente: de 0.100×10^{-99} a $0.999 \times 10^{+99}$.

1. Números negativos grandes, menores a -0.999×10^{99}
2. Números negativos entre a -0.999×10^{99} y -0.100×10^{-99}
3. Números negativos chicos, mayores a -0.100×10^{-99}
4. Cero
5. Números positivos chicos, menores a -0.100×10^{-99}
6. Números positivos entre a $+0.100 \times 10^{-99}$ y $+0.999 \times 10^{99}$
7. Números positivos grandes, mayores a $+0.999 \times 10^{99}$



Error de redondeo

Las operaciones con números de punto flotante pueden llevar a resultados que no se pueden expresar exactamente. El resultado es llevado al número representable más cercano.

Aparece error de redondeo. Ej: $0.100 \times 10^{3} / 3 \rightarrow 0.333 \times 10^{2}$

- El espacio entre números expresables adyacentes no es constante.

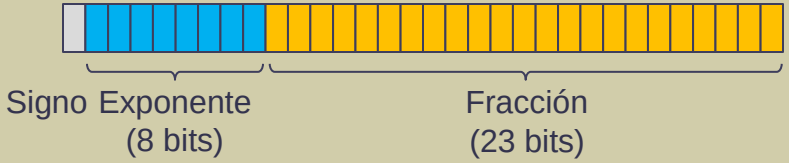
Ej: $0.999 \times 10^{99} - 0.998 \times 10^{99}$ vs. $0.999 \times 10^{-10} - 0.998 \times 10^{-10}$

- El error relativo es prácticamente el mismo.

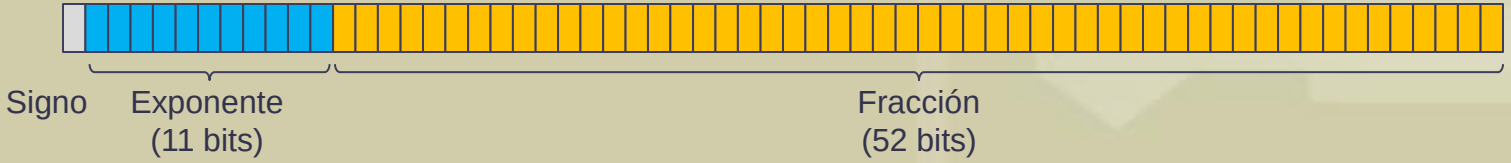
Ej: $0.999 \times 10^{99} / 0.998 \times 10^{99}$ vs. $0.999 \times 10^{-10} / 0.998 \times 10^{-10}$

IEEE 754

Simple precisión (32 bits) 1 bit signo, 8 bits exponente y 24 bits de fracción



Doble precisión (64 bits) 1 bit signo, 11 bits exponente y 53 bits de fracción



En la fracción se omite el primer bit que siempre es 1.
El exponente está codificado con un offset (excess) igual a $(2^{e-1})-1$.

$$Número = (-1)^{Signo} \times 2^{Exponente-Offset} \times (1 + Fracción)$$

Links

- http://en.wikipedia.org/wiki/Computer_architecture
- http://en.wikipedia.org/wiki/Computer_numbering_format
- http://en.wikipedia.org/wiki/Floating_point
- http://en.wikipedia.org/wiki/Kahan_summation_algorithm
- http://en.wikipedia.org/wiki/IEEE_754-1985
- http://en.wikipedia.org/wiki/Single_precision_floating-point_format
- http://en.wikipedia.org/wiki/Double_precision_floating-point_format