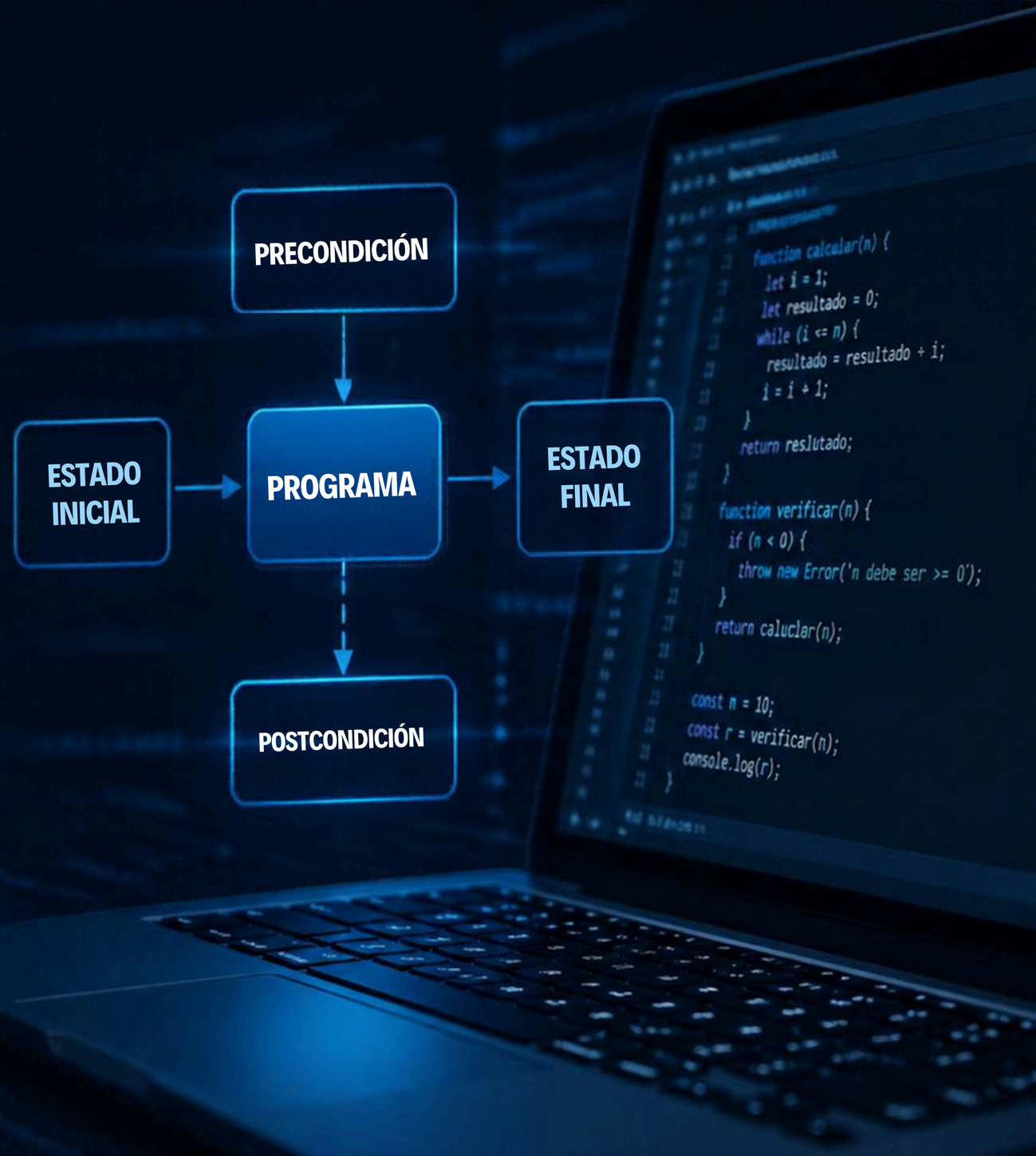
CORRECTITUD PARCIAL

Fundamentos, demostración y aplicación en el desarrollo de software

MATERIA
Desarrollo de Software

CARRERA
Ingeniería de Sistemas

INTEGRANTES
Johan Steinert Riascos Panameño
Jesús David Cortés Alegría Anderson
Andrés Arboleda Rojas



CONCEPTO DE CORRECTITUD PARCIAL

Un programa es parcialmente correcto respecto a una especificación si, iniciando en un estado que cumple la precondición, y si termina su ejecución, el estado final cumple la **poscondición**.

Verifica que los resultados son correctos cuando el programa termina

Relaciona condiciones iniciales con resultados finales

No garantiza terminación

La falta de terminación no viola por sí misma la correctitud parcial

EJEMPLO COTIDIANO

Sumar dos números.
Si el programa termina, el resultado obtenido es la suma correcta.



IMPORTANCIA Y OBJETIVOS

- Permite razonar sobre la lógica interna del programa
- Detecta errores en algoritmos antes de su implementación
- Facilita la comprobación de resultados contra requisitos
- Mejora la confiabilidad del software
- Documenta y justifica el comportamiento esperado
- Es base para pruebas y verificación formal

PRUEBAS DE SOFTWARE

Casos concretos y escenarios específicos

VS

DEMOSTRACIÓN FORMAL

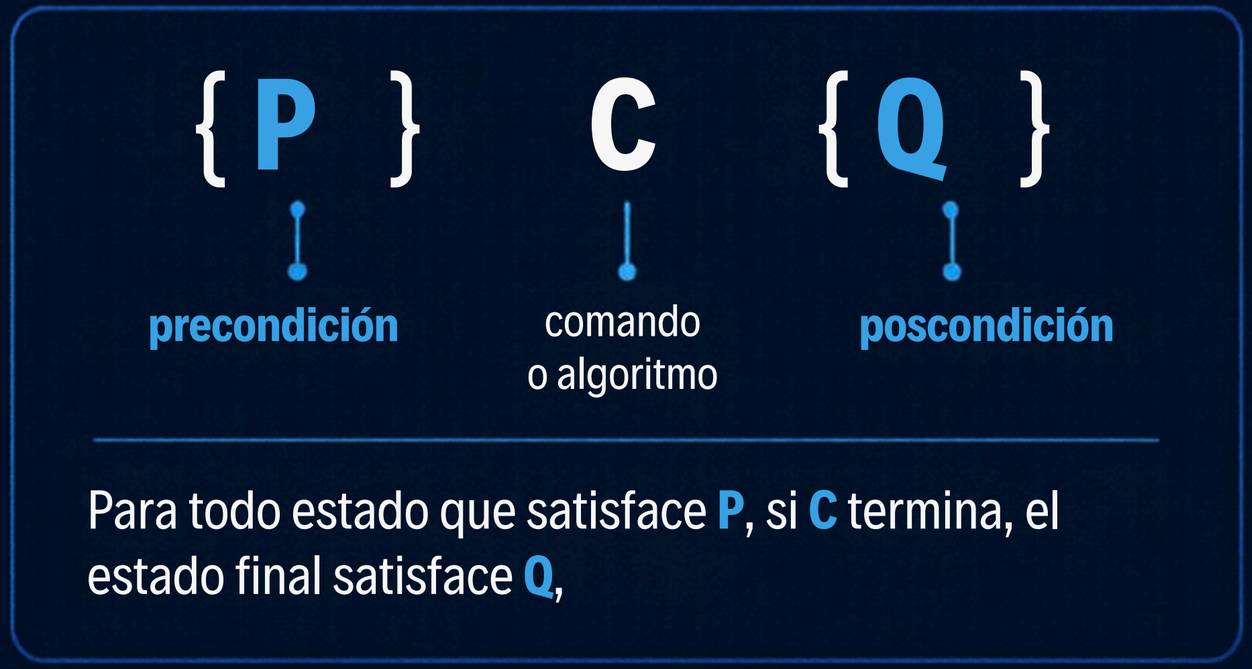
Propiedad válida para todos los estados posibles del programa



PRECONDICIONES Y POSCONDICIONES

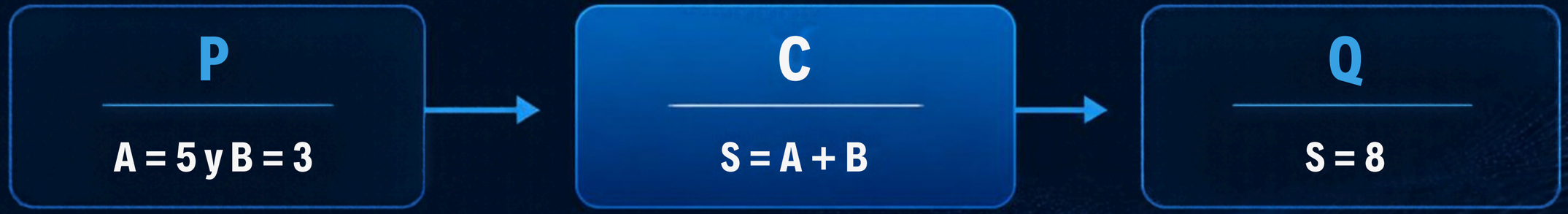
Precondición (P):
condición que debe cumplirse antes de la ejecución

Poscondición (Q):
condición que debe cumplirse al finalizar la ejecución, si termina



Para todo estado que satisface **P**, si **C** termina, el estado final satisface **Q**,

EJEMPLO



LÓGICA DE HOARE

Herramienta formal para razonar sobre la **correctitud parcial** de programas.

$$\{ P \}\ C\ \{ Q \}$$

P representa la **precondición**

C represente el programa

Q representa la **poscondición**

La lógica de Hoare demuestra que si el programa termina, la poscondición es verdadera.

No garantiza que el programa termine.

```
// Programa: suma de dos números
1   { P: A = 5 ∧ B = 3 }
2       int A, B, S;
3       S = A + B;
4       // Resultado de la suma
5       return S;
    { Q: S = 8 }
```

1. Valores iniciales
   A = 5, B = 3 (P)
2. Ejecución del programa (C)
3. Resultado obtenido
   S = 8 (Q)
4. Si el programa termina, **Q** es verdadera

Si se cumple P y, tras ejecutar C, se obtiene **Q**, entonces el programa es **parcialmente correcto** respecto a **{P} C {Q}**.

EJEMPLO PRÁCTICO DESARROLLADO

Algoritmo para calcular el doble de un número.

PSEUDOCÓDIGO

```
01  INICIO
02      LEER NÚMERO
03      doble = número * 2
04      MOSTRAR DOBLE
05  FIN
```

Precondición: número es un entero

Poscondición: doble = 2 × número inicial

Ejemplo: con número=4, resultado esperado doble=8

PASO	OPERACIÓN	RESULTADO
1	Leer número	número = 4
2	doble = número * 2	doble = 4 * 2
3	Mostrar doble	doble = 8
4	Fin	

Si el algoritmo termina, habiendo cumplido la precondición,
entonces se cumple la **poscondición: doble = 2 × número inicial.**

DIFERENCIA ENTRE CORRECTITUD PARCIAL Y TOTAL

ASPECTO	CORRECTITUD PARCIAL	CORRECTITUD TOTAL
Qué asegura	Resultado correcto si el programa **termina**	Resultado correcto y **termina siempre** para entradas válidas
Qué exige	**No exige** demostrar terminación	**Incluye** correctitud parcial y prueba de terminación
Enfoque	Se centra en la **relación P → Q** si el programa termina	Garantiza **P → Q** y terminación para todas las entradas válidas

EJEMPLOS

Correctitud total

Algoritmo suma simple

Siempre produce el resultado correcto y termina para cualquier entrada válida.

```
suma(a, b)
  s ← 0
  s ← a + b
  devolver s
```

Solo correctitud parcial

Algoritmo con ciclo potencialmente infinito
Puede producir el resultado correcto si termina, pero no hay garantía de que termine.

```
proceso(x)
 mientras x ≠ 0 hacer
  x ← x − 1
 devolver x
```

Un **ciclo infinito** impide la correctitud total, pero **no necesariamente** afecta la correctitud parcial.

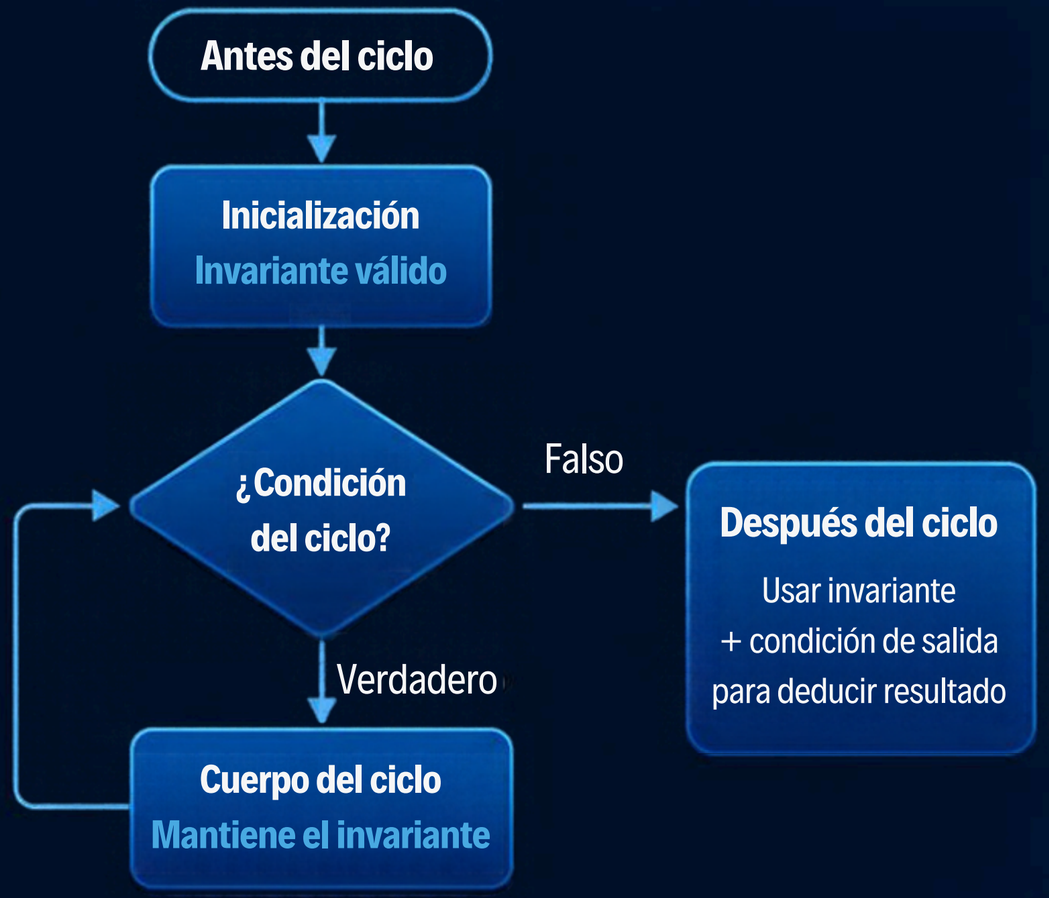
RELACIÓN CON LOS CICLOS Y LA TERMINACIÓN

La **correctitud parcial** analiza que el resultado obtenido es correcto si el programa termina.

Demostrar terminación es una **propiedad distinta.**

TRES PASOS PARA RAZONAR CON INVARIANTES

1. **Inicialización**
   El invariante es válido antes de entrar al ciclo.
2. **Mantenimiento**
   Si el invariante vale el inicio de una iteración, también vale al finalizarla.
3. **Finalización**
   Al salir del ciclo, el invariante junto con la condición de salida permite deducir el resultado.



CORRECTITUD PARCIAL

Si el programa termina, el resultado es correcto.

TERMINACIÓN

Requiere demostrar que el ciclo eventualmente finaliza.

VENTAJAS, LIMITACIONES Y APLICACIONES



VENTAJAS

- Razonamiento formal de algoritmos
- Identificación de errores lógicos
- Verificación de requisitos
- Software más confiable
- Base para demostraciones formales



LIMITACIONES

- No garantiza terminación por sí sola
- Depende de definiciones claras de pre/postcondiciones
- Requiere conocimientos de lógica y matemáticas
- Verificación compleja en programas grandes



APLICACIONES

- Algoritmos matemáticos
- Sistemas de procesamiento
- Software con alta confiabilidad
- Verificación formal
- Análisis de ciclos y condiciones

CONCLUSIONES Y BIBLIOGRAFÍA

La **correctitud parcial** asegura resultados correctos si el programa termina

Las **precondiciones** y **poscondiciones** describen el comportamiento esperado

La **lógica de Hoare** formaliza el razonamiento sobre programas

La **correctitud total** añade la prueba de terminación

Es una **herramienta clave** para desarrollar software confiable

REFERENCIAS ACADÉMICAS

C. A. R. Hoare
"An Axiomatic Basis for Computer Programming"
(1969)

Michael Huth y Mark Ryan
"Logic in Computer Science: Modelling and Reasoning about Systems"

Gracias por su atención. **¿Preguntas?**