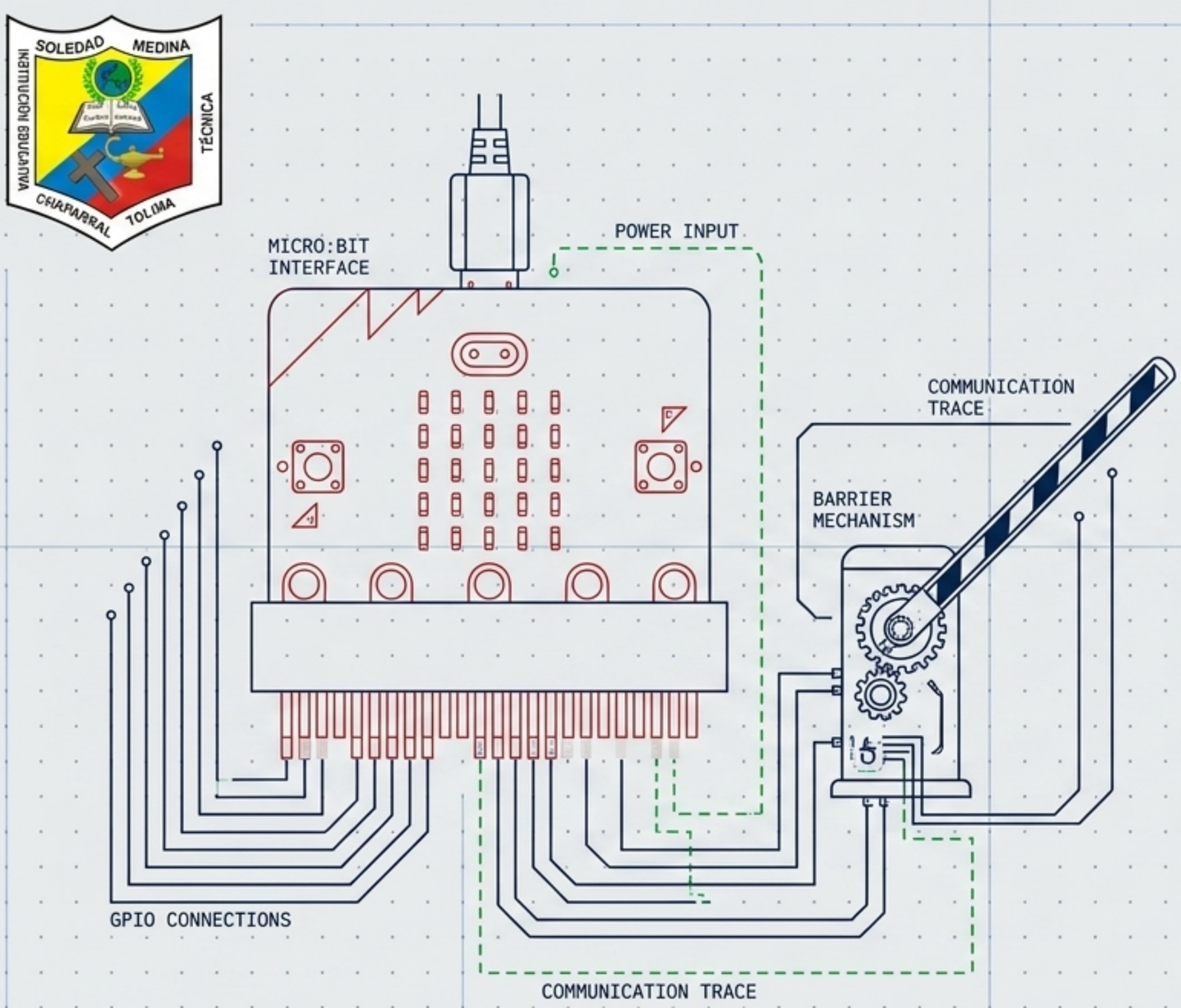
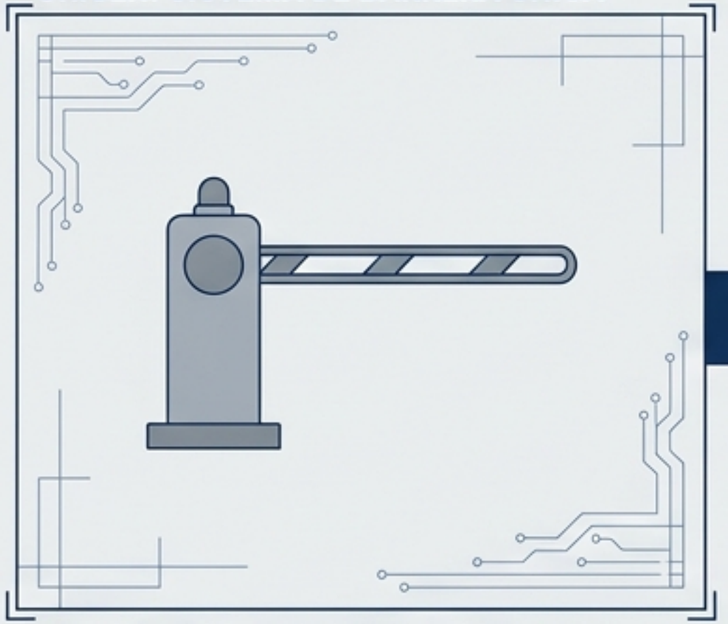


Parqueadero Inteligente 2.0

Optimización, Alarmas Visuales y Escalabilidad en micro:bit.

Laboratorio de Prototipado / Sesión 4 - Grado 11°

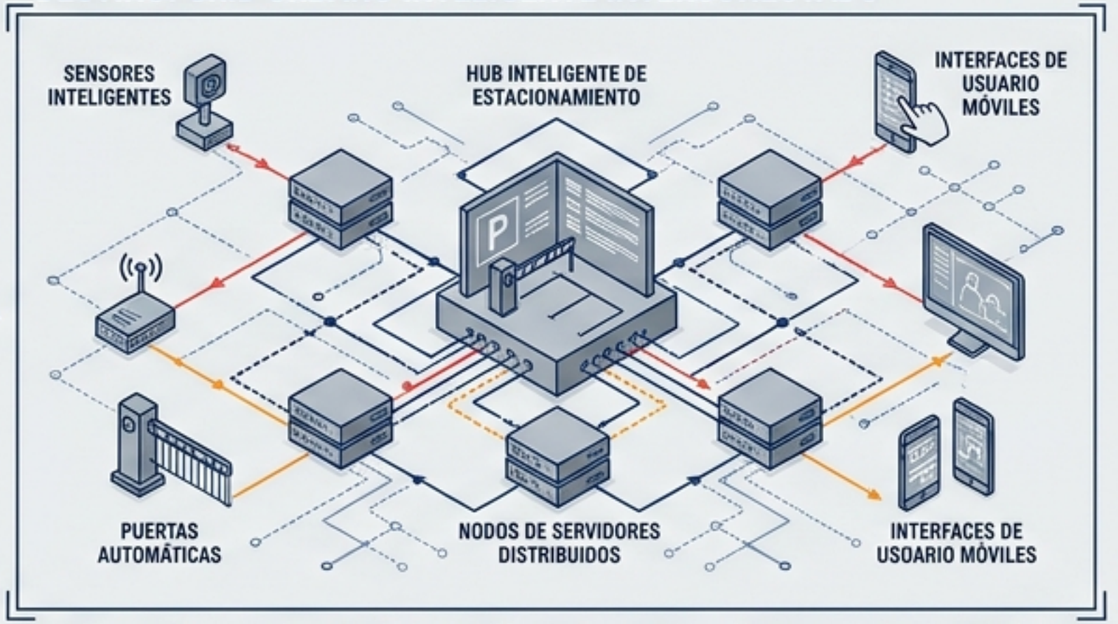
ORIGEN: SISTEMA DE BARRERA ÚNICA



Mecánica básica, control simple, sin integración de datos.

20% de avance del proyecto final

DESTINO: GRID URBANO INTELIGENTE INTERCONECTADO



Sistemas integrados, análisis de datos, comunicación activa y escalabilidad.

MÓDULOS

TECHNICAL

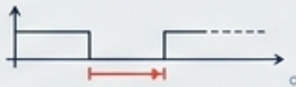
MÓDULO 1: Optimizar código existente.

```
REF: OPT-CODE-01;
// Refactorización de rutinas de control;
REDUCE_LATENCY(system_core);
```

MÓDULO 2: Implementar alarmas visuales basadas en disponibilidad.

```
IF (occupancy > THRESHOLD) THEN (
  ACTIVATE_VISUAL_ALARM(LED_ARRAY_01, WARNING_ORANGE);
)
```

MÓDULO 3: Integrar control de tiempo (simulación de cobro).



```
TIME_START = GET_TIMESTAMP();
// Simulación de entrada;
...
CALC_FEE(TIME_END - TIME_START, RATE_TABLE);
```

MÓDULO 4: Evaluar escalabilidad.

```
TEST: SCALABILITY_AUDIT;
// Pruebas de carga multi-nodo;
VERIFY_THROUGHPUT(data_stream_XS);
```

TAKEAWAY: Un sistema real no solo funciona de manera mecánica; analiza datos, previene errores y se comunica activamente con el usuario.

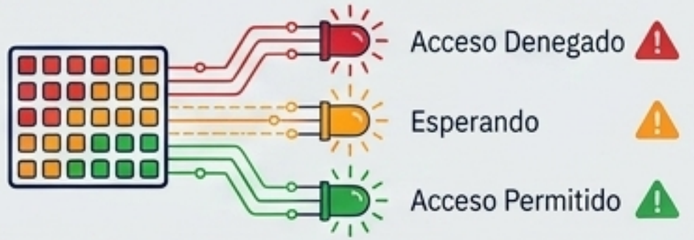
Arquitectura de la Solución

Para llevar el proyecto al siguiente nivel, añadimos una capa de comunicación visual y control de tiempo que informa al usuario antes de interactuar con la barrera.

Salidas de Estado (Alertas Visuales LEDs)

Matriz de LEDs

Indicadores Estado




```
// Visual Feedback
if (accessDenied) {
  basic.showIcon(IconNames.No);
  pins.digitalNritePin(DigitalPin.P1, 1);
}
```

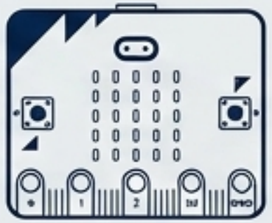
Entradas

Sensores
Proximidad
Luz

Botones
Activación
Reset

```
// Input Readings
input.onButtonPressed(Button.A, function () {
   readSensor();
});
```

CEREBRO (micro:bit)



Controlador Central

Salidas de Acceso

Barreras / Servomotores
Acción Mecánica
Control Posición

```
// Actuatlon Logic
pins.servoWrltePin(AnalogPin.P0, 90); // Open barrier
```

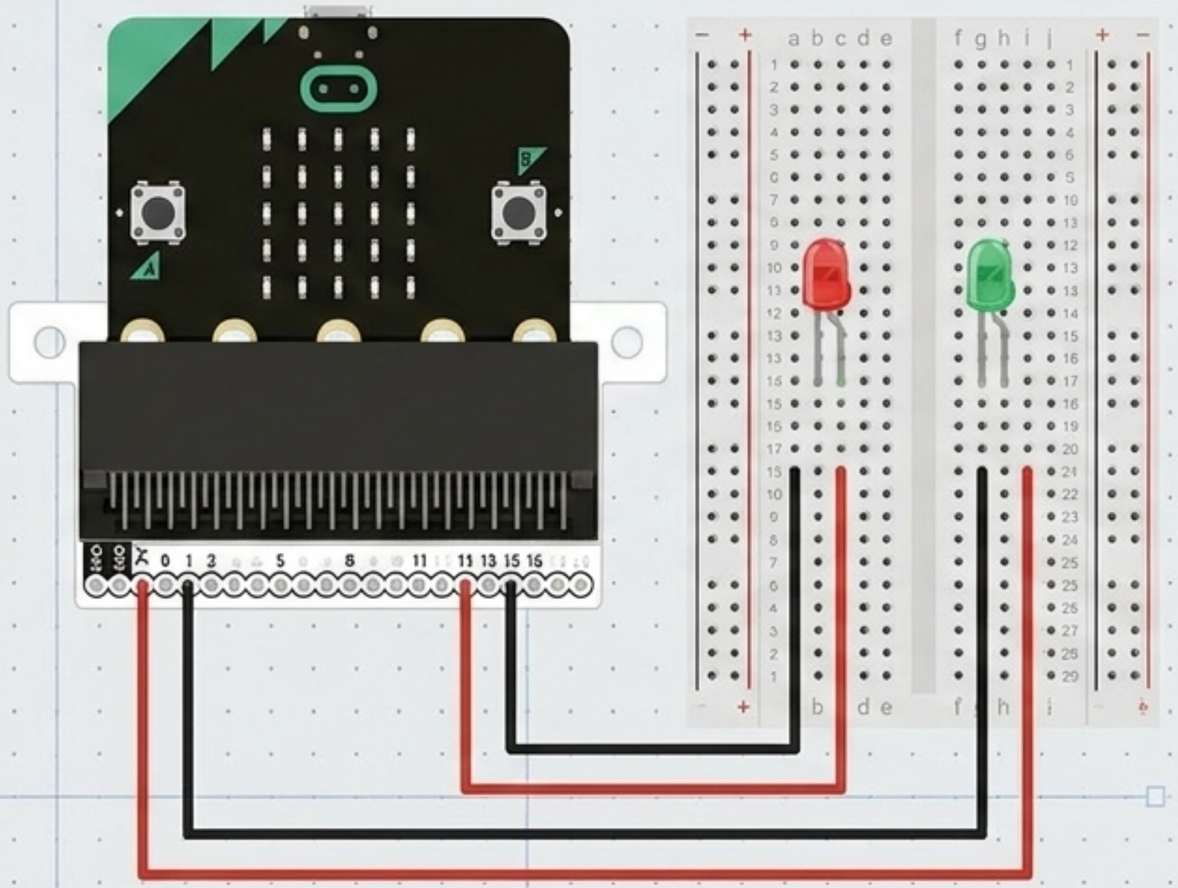
DIAGNÓSTICO Y LÓGICA DE HARDWARE

Condición Matemática	Estado del Sistema	Hardware Asignado	Señal del Microcontrolador
`puestosDisponibles = 0`	Lleno	LED Rojo (Pin P1) 	ALTA (Parpadeo)
`puestosDisponibles: 1 a 3`	Casi Lleno	LED Naranja (Pin P2) 	ALTA (Parpadeo)
`puestosDisponibles > 3`	Disponible	Ambos LEDs 	BAJA (Apagado)

El diseño de hardware comienza definiendo el comportamiento lógico.

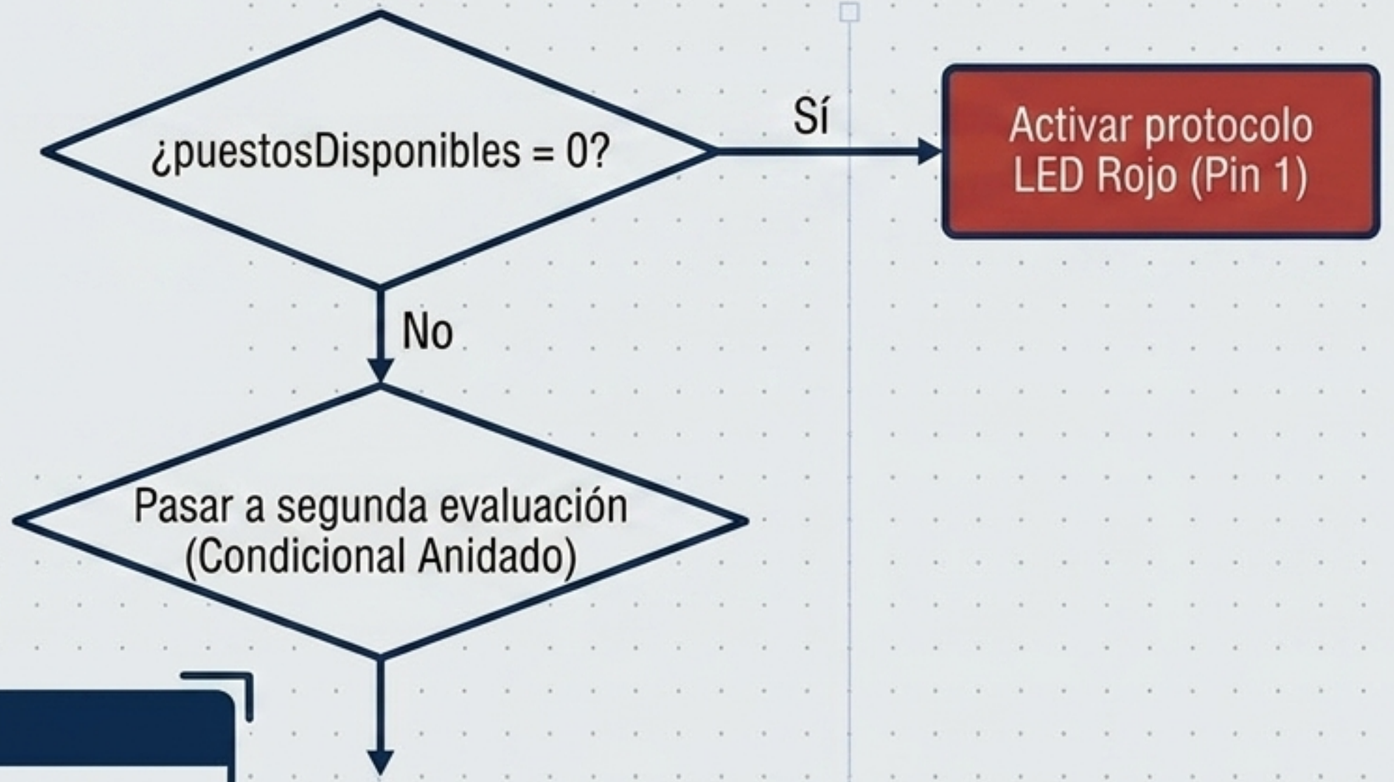
Enrutamiento de Hardware

Tabla de Conexiones		
LED Rojo	Pin de Señal P1 (Cable Rojo)	Tierra GND (Cable Negro)
LED Naranja	Pin de Señal P2 (Cable Rojo)	Tierra GND (Cable Negro)



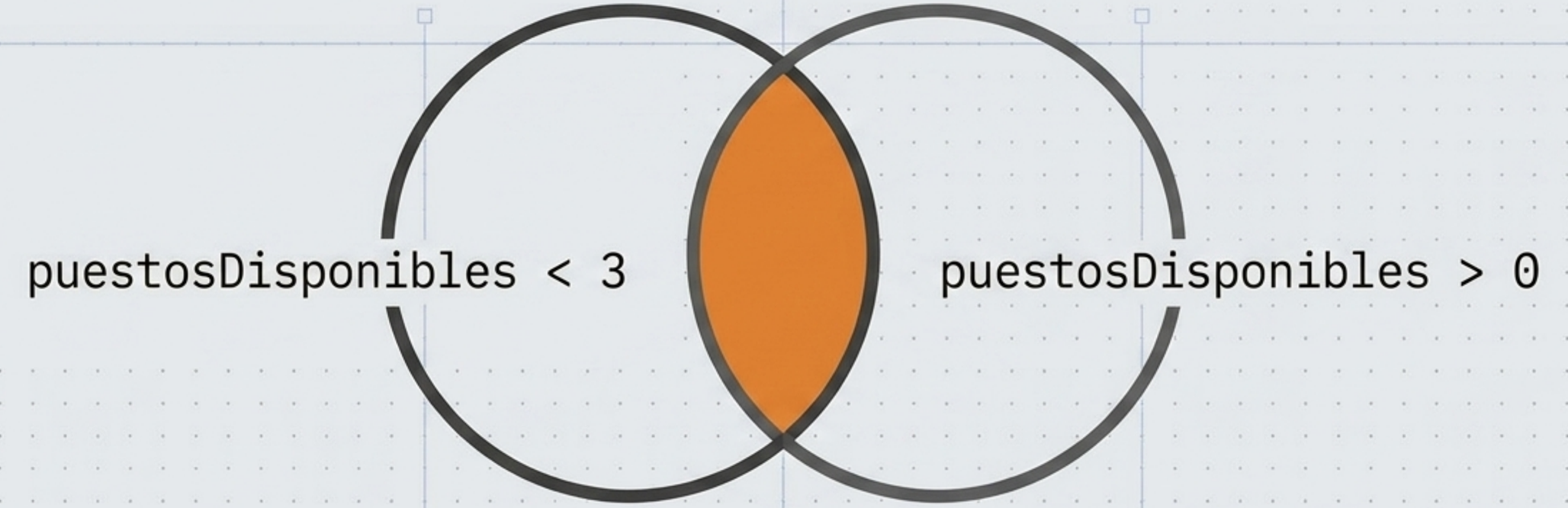
⚠ Mantener la convención de colores (Rojo = Positivo, Negro = Negativo) previene cortocircuitos y permite que cualquier ingeniero pueda leer el plano al instante.

DIAGRAMA DE FLUJO DE DECISIÓN



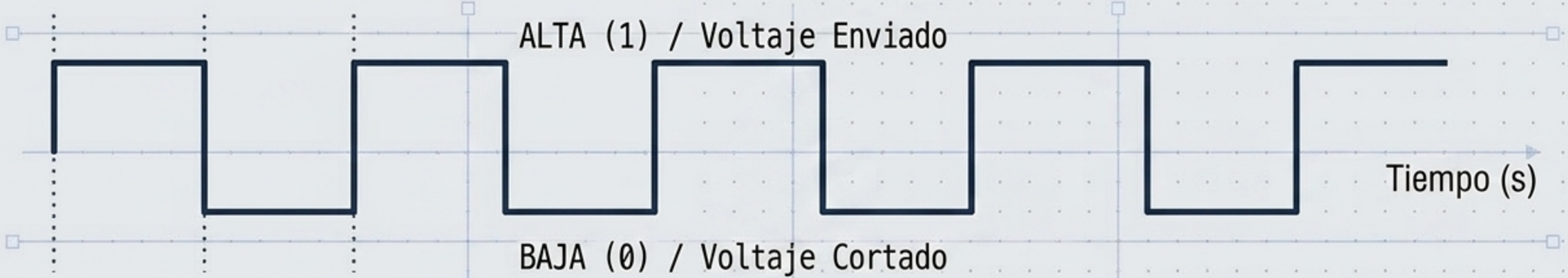
El bloque ‘SI’ actúa como un filtro. Si la primera condición no se cumple, el sistema verifica sub-condiciones antes de tomar una decisión.

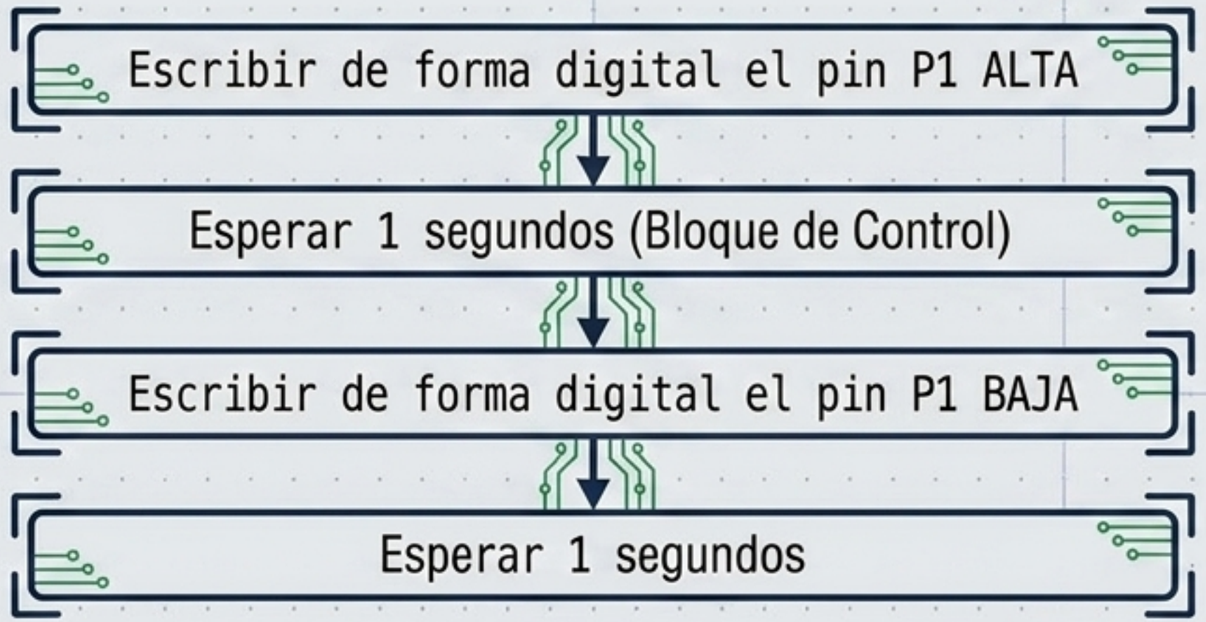
El Operador Lógico ‘Y’



Para el estado ‘Casi Lleno’, el sistema debe verificar un rango específico. El operador ‘Y’ (AND) asegura que el LED naranja solo se encienda si ambas condiciones son matemáticamente verdaderas simultáneamente, evitando falsos positivos cuando el parqueadero está completamente lleno (0).

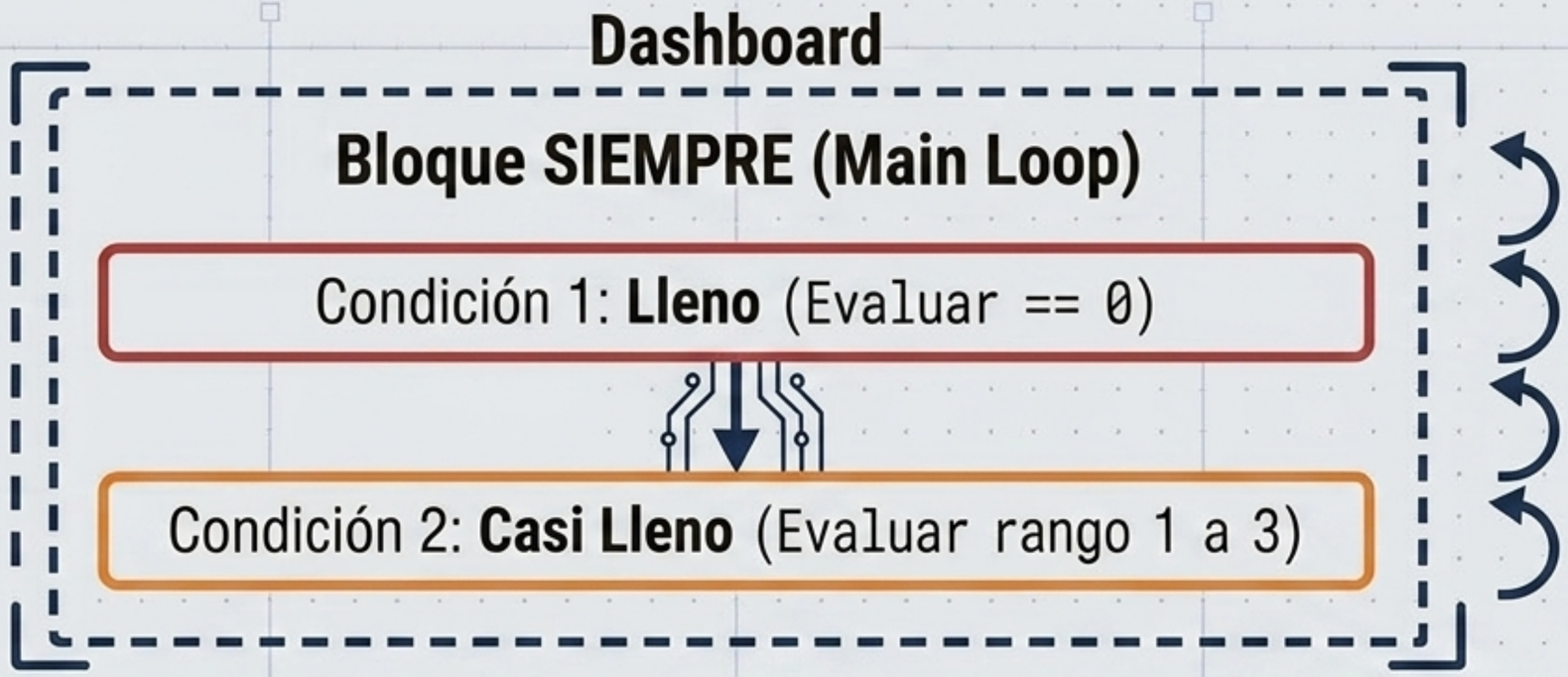
Señales Digitales y el Efecto de Parpadeo





El parpadeo no es una función mágica; es la manipulación precisa de la corriente eléctrica a través del tiempo.

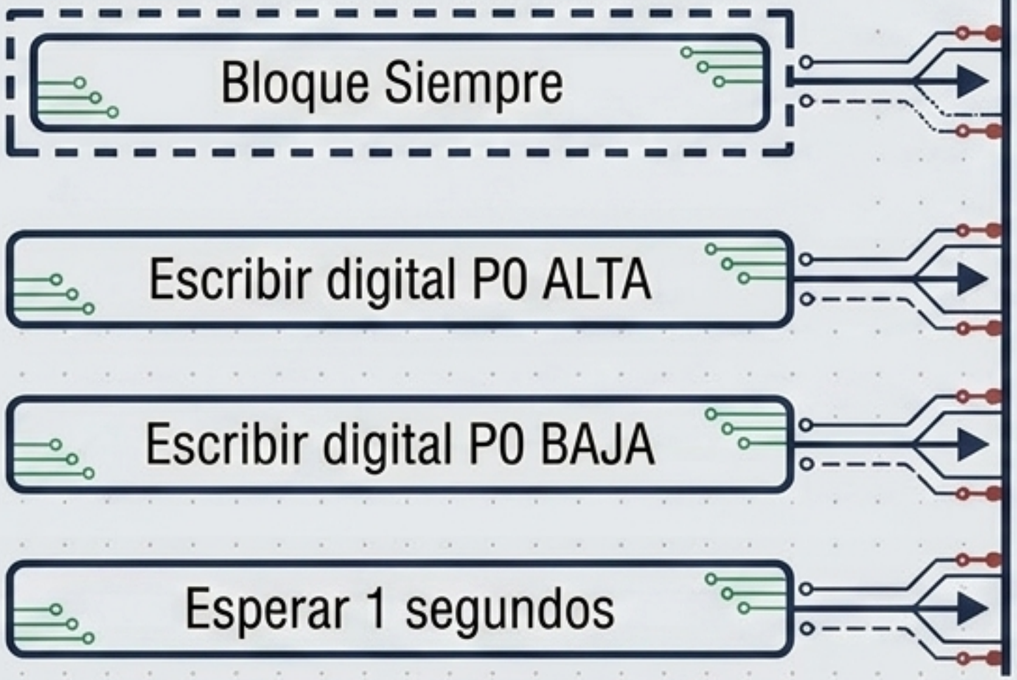
Monitoreo en Tiempo Real



El bloque ‘Siempre’ actúa como un supervisor constante. Evalúa la variable `puestosDisponibles` miles de veces por segundo y reacciona instantáneamente según el flujo programado. Es el corazón operativo del sistema.

Comparación de Lenguaje: Bloques vs. Python

Representación Gráfica (Bloques)

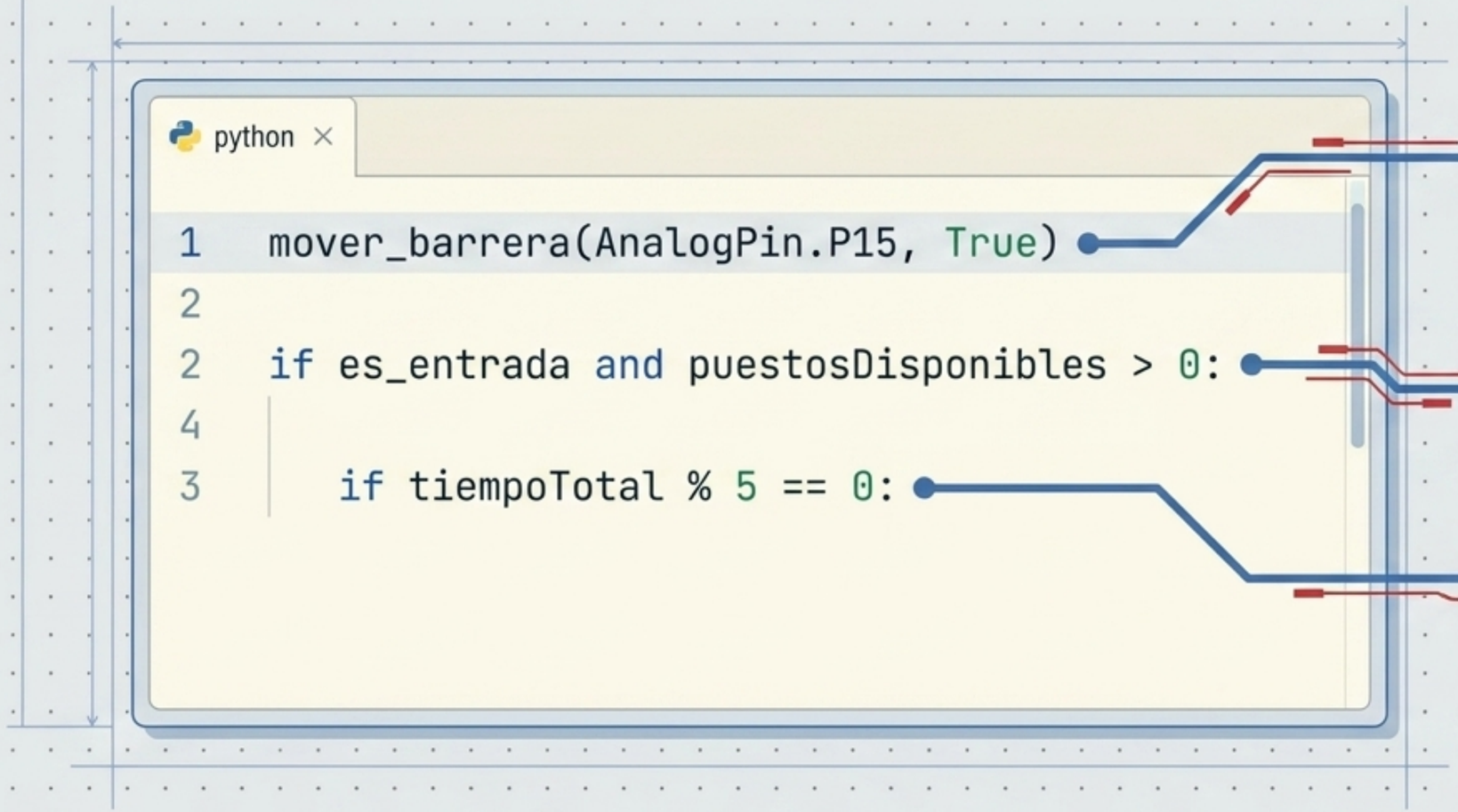


Sintaxis Profesional (Python)

```python
def on_forever():
    basic.forever(on_forever)

pins.digital_write_pin(DigitalPin.P0, 1)

pins.digital_write_pin(DigitalPin.P0, 0)

basic.pause(1000)
```

La lógica algorítmica es idéntica; solo evoluciona el lenguaje.

Análisis de Código Profesional: Lógica y Control Físico

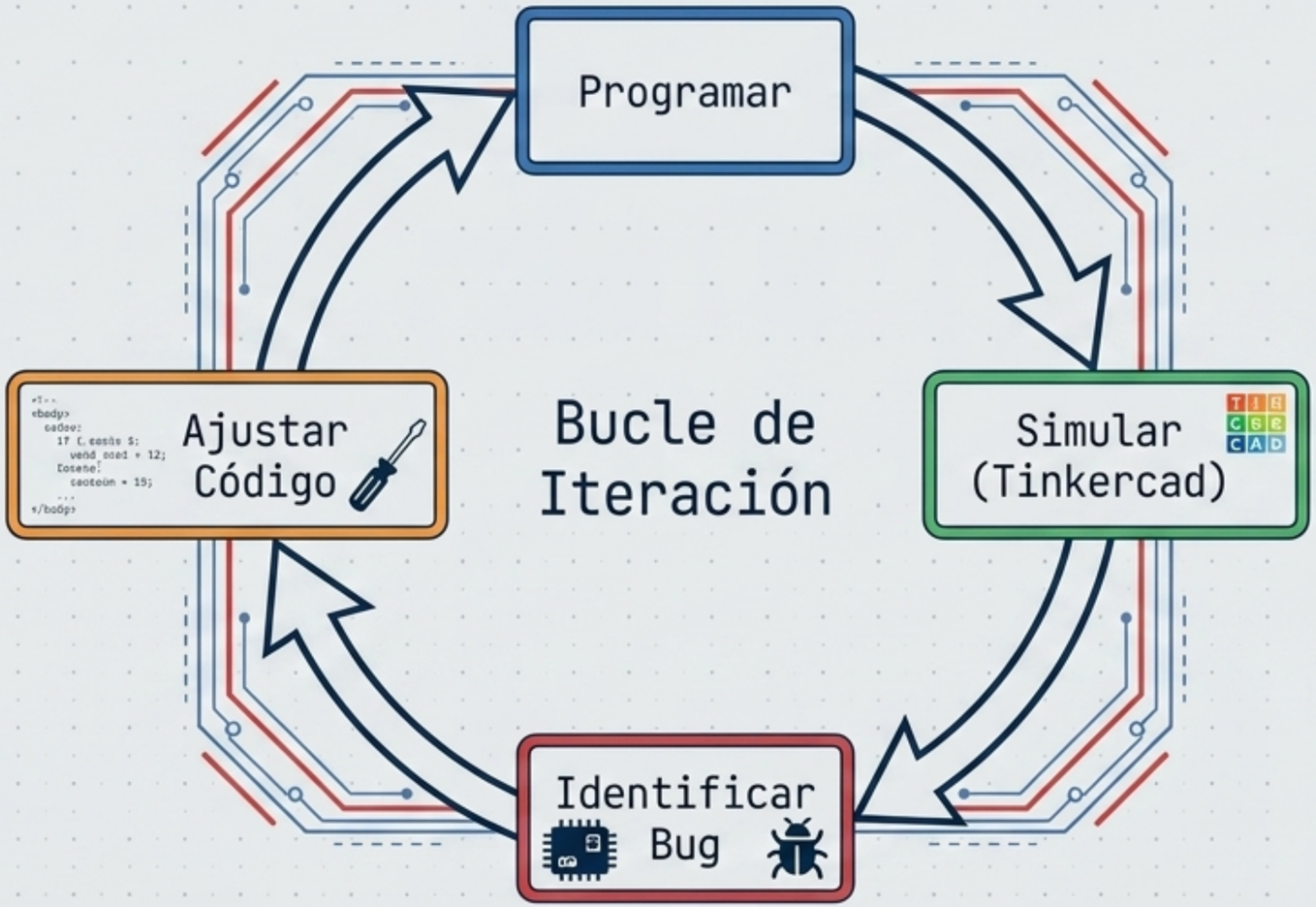


Control Físico: Ajuste dinámico de los servomotores (90 grados).

Validación Lógica: Bloquea la entrada si no hay cupo en el sistema.

Módulo Económico: Uso de matemáticas modulares para simular el cobro de estacionamiento cada 5 minutos exactos.

Testing y Refinamiento (QA)

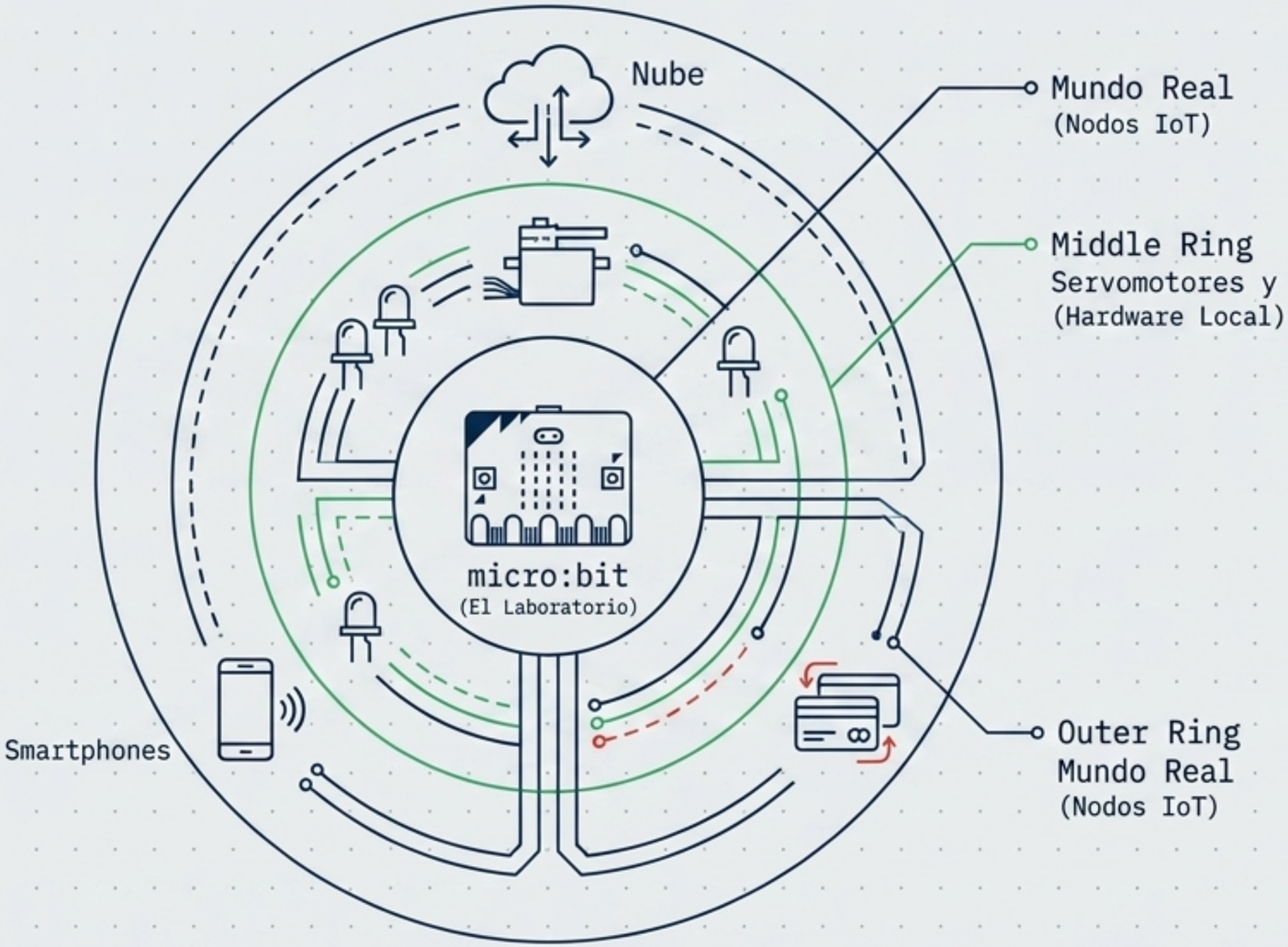


Preguntas de Diagnóstico:

- ☑ ¿Se encienden los LEDs correctos según el número de vehículos?
- ☑ ¿Se detiene el contador de disponibilidad al llegar a 0?
- ☑ ¿El cobro por tiempo (tiempoTotal) se actualiza correctamente?

El desarrollo de hardware real requiere pruebas exhaustivas virtuales antes de conectar un solo cable.

Más Allá del Laboratorio: Escalabilidad



El código creado hoy es la base matemática para sistemas masivos. En el mundo real, este prototipo escala hacia:

- análisis de patrones de uso
- predicción de disponibilidad vía Machine Learning
- integración automatizada de pagos.

Evaluación del Sistema

☑ Optimización

Capacidad para extender y refinar código lógico existente.

☑ Implementación

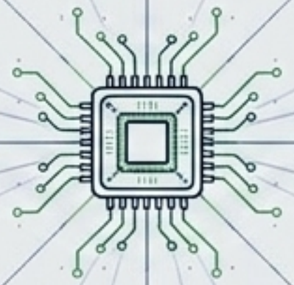
Integración exitosa de características avanzadas (Alarmas visuales y control de cobro).

☑ Escalabilidad

Evaluación clara de cómo el modelo simulado se traduce a un entorno físico y comercial real.

Pensando en el reto central…

¿Cómo te preparan estas habilidades algorítmicas y de diseño de hardware para los grandes proyectos tecnológicos del futuro?



La ingeniería no se trata de conectar piezas, sino de diseñar soluciones inteligentes que se anticipen al mundo real.