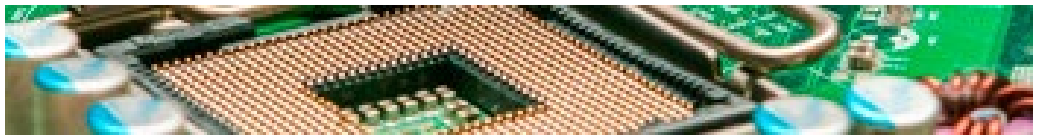
# Tema 2: Planificación inteligente

# Seminario 2. Modelado de problemas de planificación con Pyhop

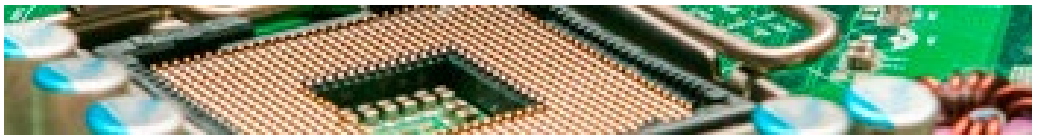




# Contenido

1. Planificación HTN
2. Conocimiento experto
3. Ejemplo
4. Tipos de métodos
   1. Secuenciales
   2. Recursivos
   3. Iterativos



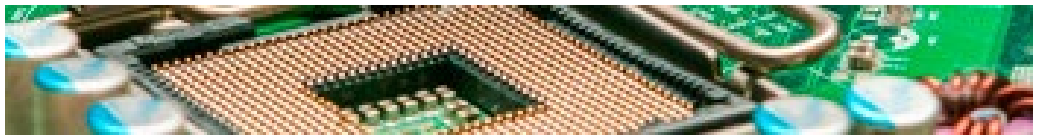


# Planificación HTN

El objetivo es obtener el conjunto de operadores a ejecutar (plan) para resolver el problema.

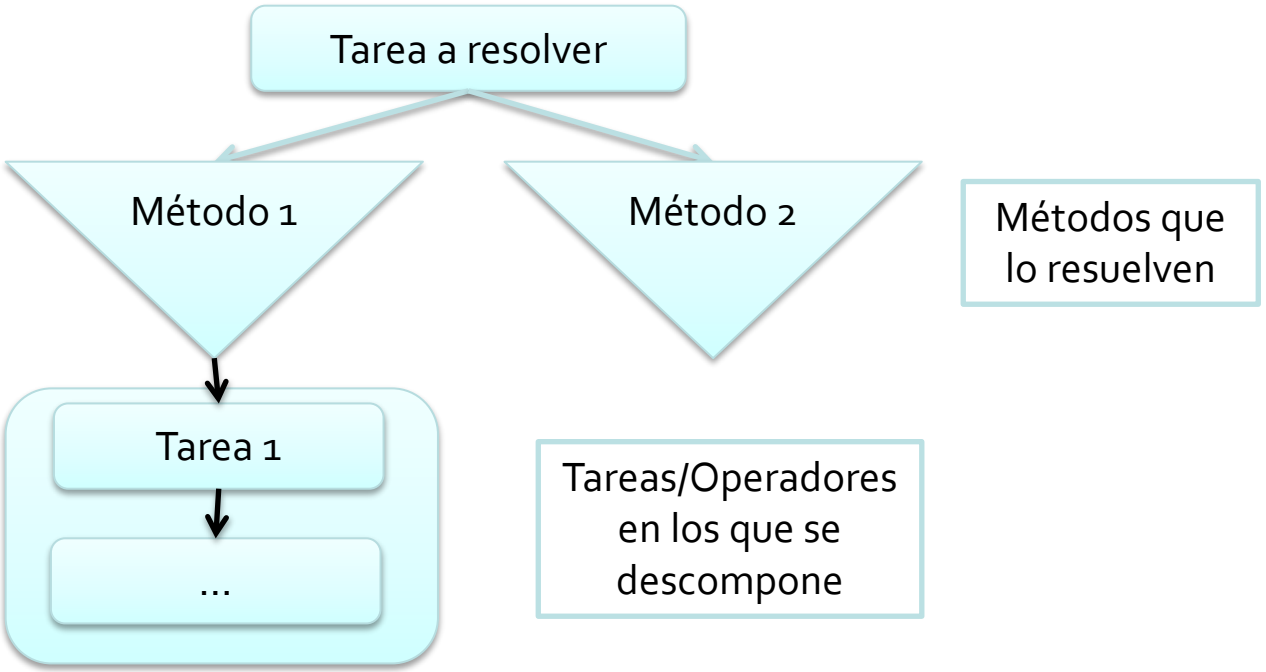
- Input:
  - Conjunto de operadores (ejecutables en el entorno).
  - Conjunto de métodos: procedimientos de actuación para descomponer tareas complejas en tareas más primitivas.
    - Método: definido en un nivel de abstracción mayor.
    - Operador: representa a la tarea ejecutable de menor nivel de abstracción.
- Proceso de planificación:
  - Descomponer tareas no-primitivas de forma recursiva hasta que se alcanza el nivel de operador.





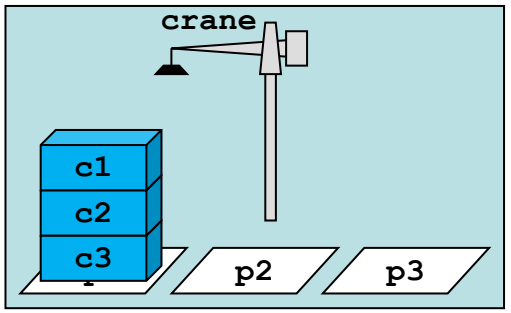
# Conocimiento experto

- Para representar el conocimiento experto, este debe estar orientado a objetivos.
- El conjunto de estos objetivos definirá un procedimiento de actuación ante problemas.



# Ejemplo

## Movimiento de contenedores del puerto




- Objetivo: mover la pila de contenedores desde el pallet p1 al pallet p3 preservando el orden inicial.

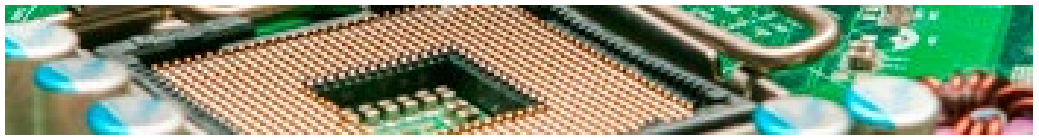
- Métodos (informales):

concepto abstracto

  - Mover a través de un pallet intermedio: mover la **pila** a un pallet intermedio en orden inverso y después al destino final en orden inverso otra vez.
  - Mover una pila: mover repetidamente el contenedor en el tope de la pila hasta que la pila esté vacía.
  - Mover el contenedor en el tope: coger el contenedor y dejarlo en su destino.

operadores ejecutables





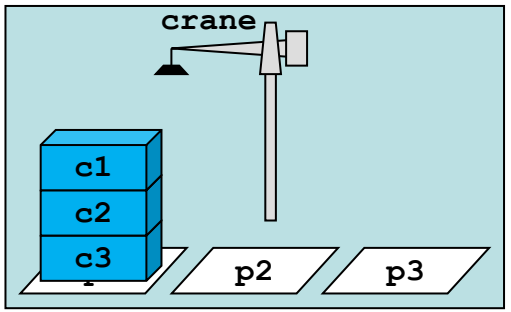
# Ejemplo

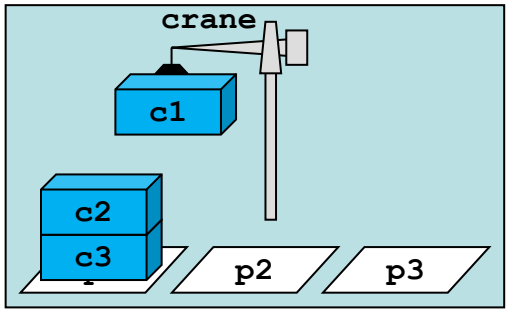
Operador *take(crane, container, container_below, pallet)*

- Instanciación:
  - *take(crane,c1,c2,p1)*

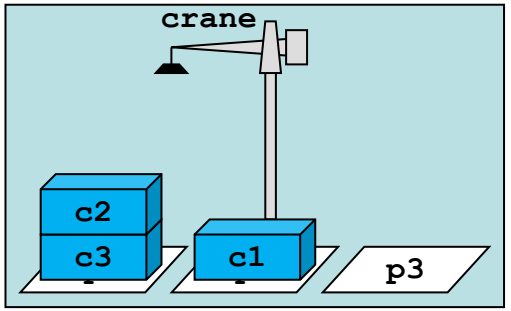
Operador *put(crane, container, container_below, pallet)*

- Instanciación:
  - *put(crane,c1,p2,p2)*

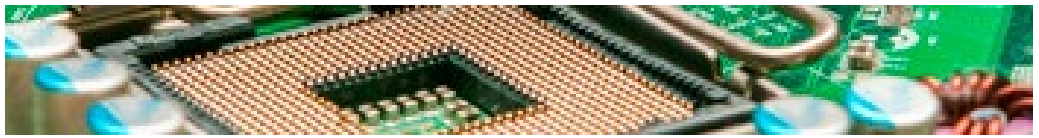



Tras la ejecución de ambos operadores

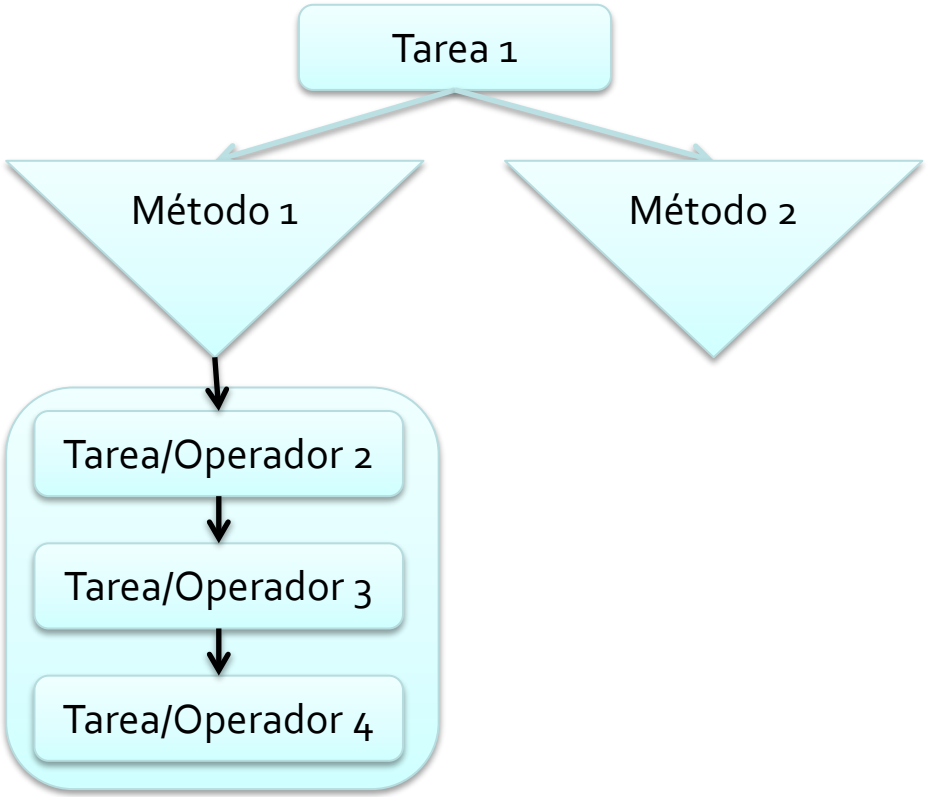




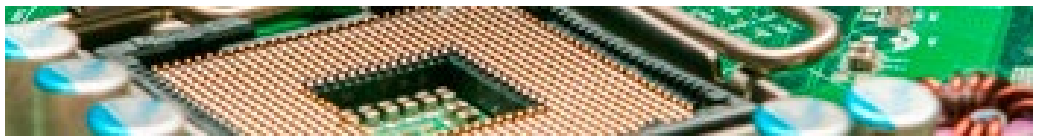



# Tipo de métodos: secuencial

Método secuencial: se descompone en una secuencia ordenada de operadores y/o tareas





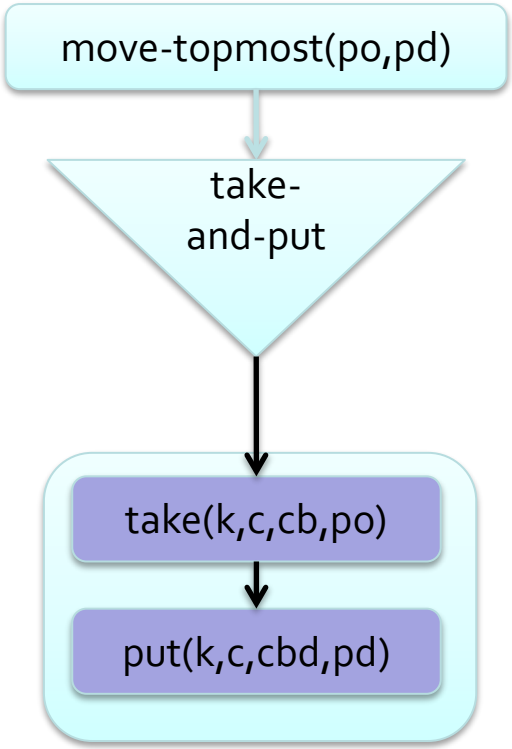



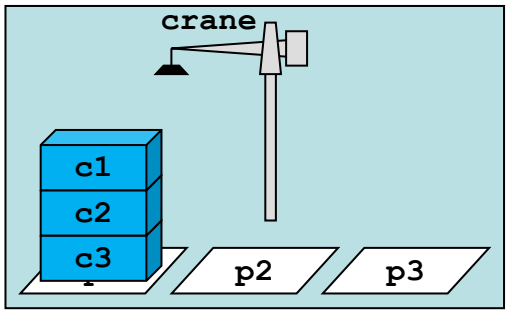
# Ejemplo

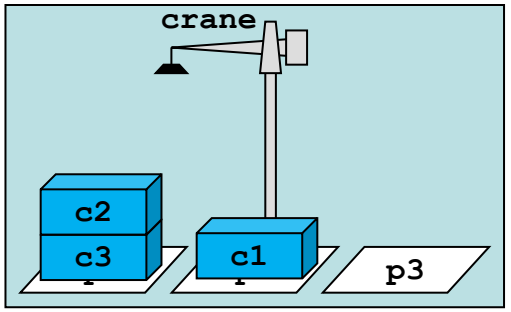
## Tarea *move-topmost(po,pd)*

*take(k - crane, c - container, cb - container_below, po - pallet)*

*put(k - crane, c - container, cbd - container_below, pd - pallet)*



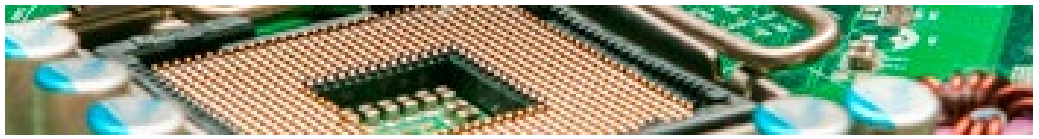




Parámetros:
*k,c,cb,cbd,po,pd*

Condiciones:
*top(c,po) and on(c,cb) and top(cbd,pd) and free(k)*



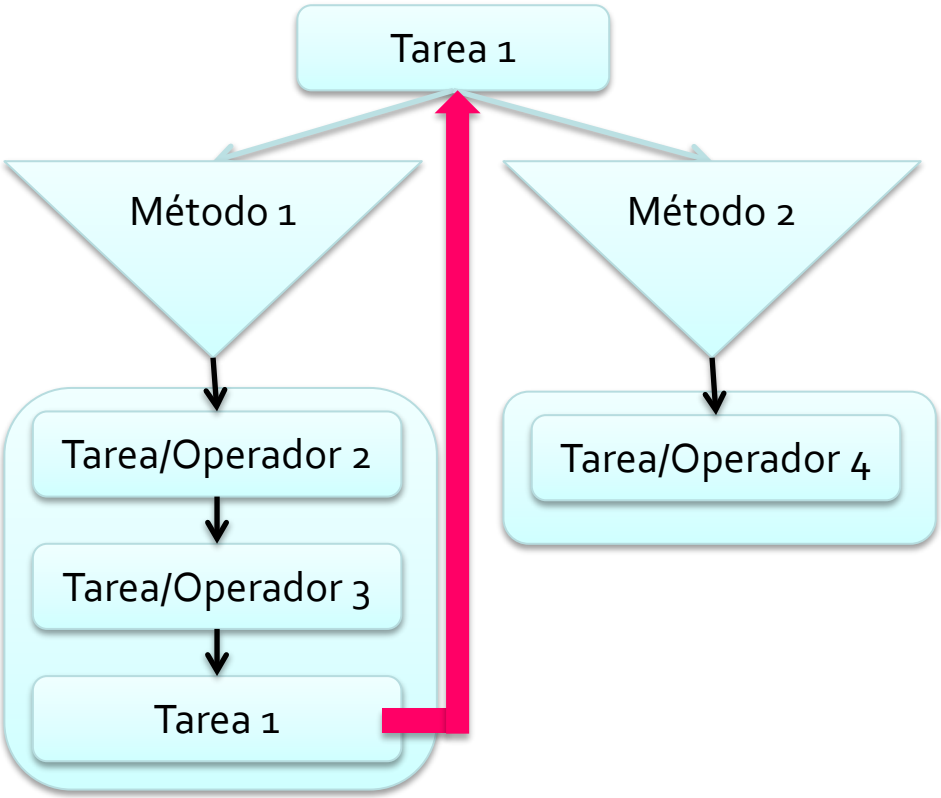


# Tipo de métodos: recursivo

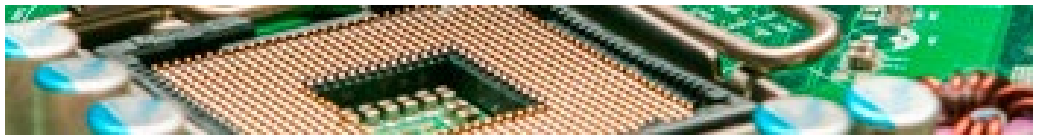
Método **recursivo**: se descompone en una secuencia ordenada de operadores y/o tareas e incluye una llamada recursiva a la tarea que resuelve ese método.

Es **necesario** definir un método como **caso base** que acompañe al método recursivo (que debería ser el primero en comprobarse).

Se utilizan para resolver tareas que pueden repetirse un número de veces indeterminado (por ejemplo, acciones de navegación o desplazamiento).

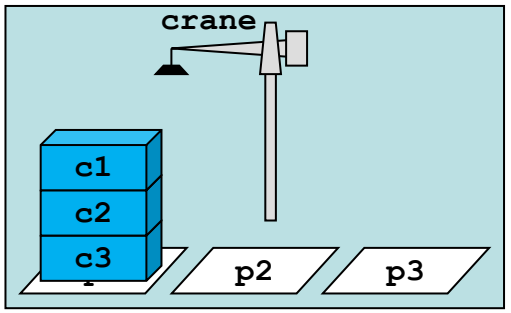


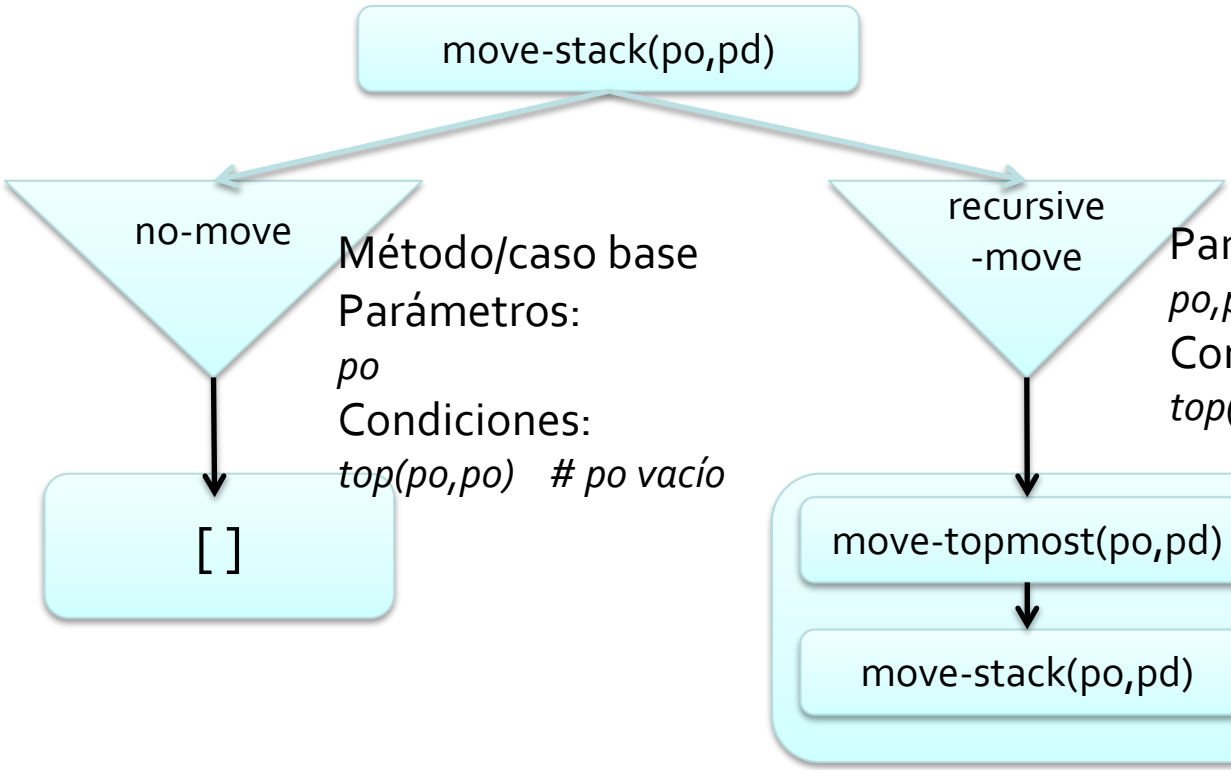


# Ejemplo

Tarea *move-stack(po,pd)* que vacía pallet *po*

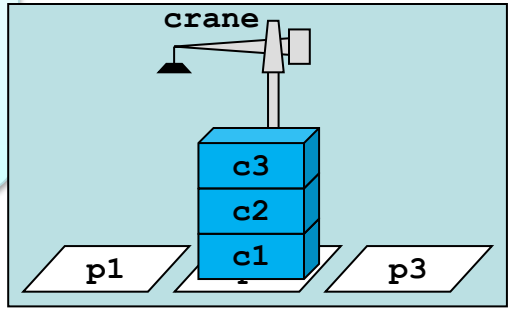


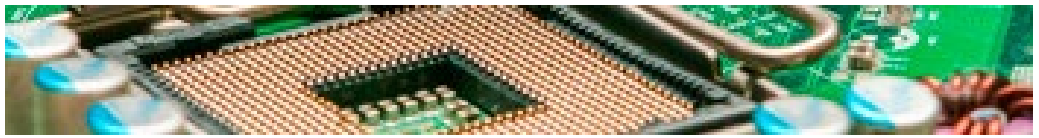

Método/caso base
Parámetros:
*po*
Condiciones:
*top(po,po)  # po vacío*

Parámetros:
*po,pd,c,cbd*
Condiciones:
*top(c,po) and top(cbd,pd)*

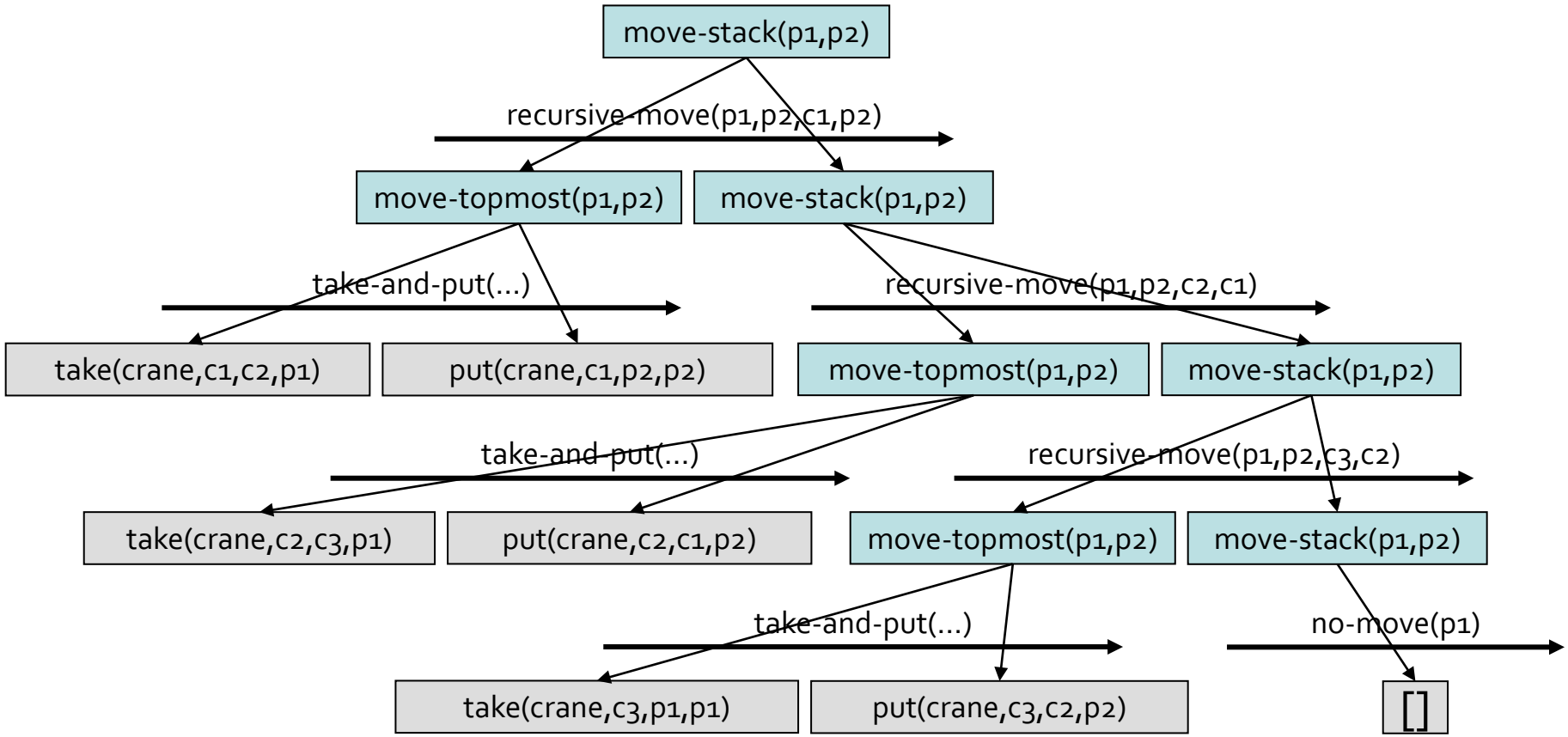
Se procesa cada uno de los contenedores de la pila



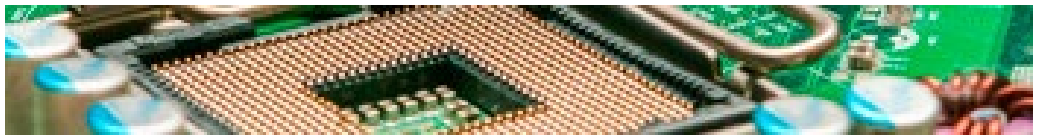


# Resultado de la descomposición

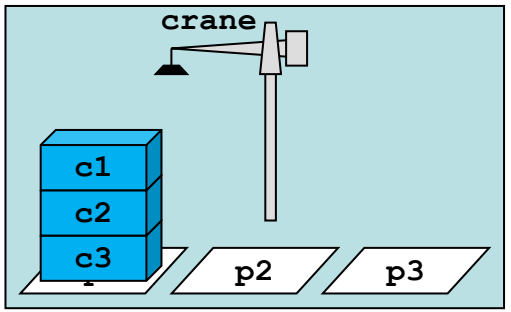




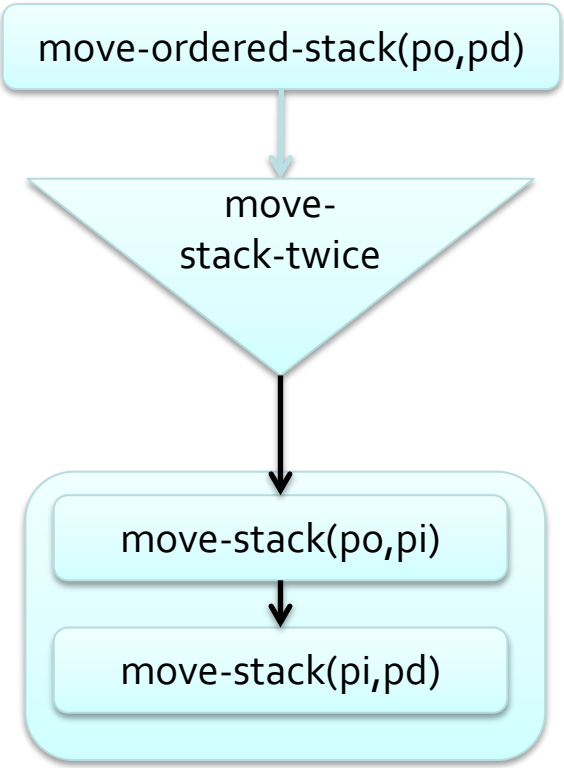



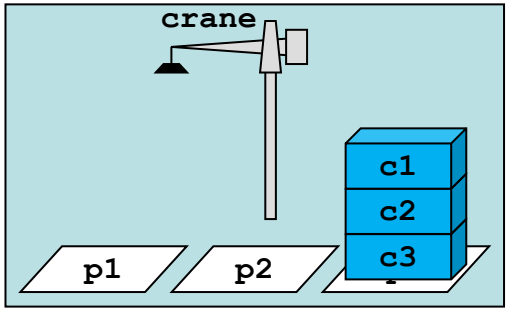
# Ejemplo

Tarea *move-ordered-stack(po,pd)*

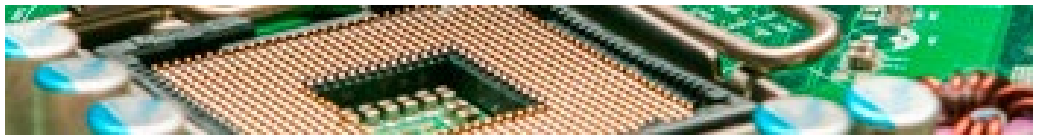



Parámetros:
*po,pi,pd*
Condiciones:



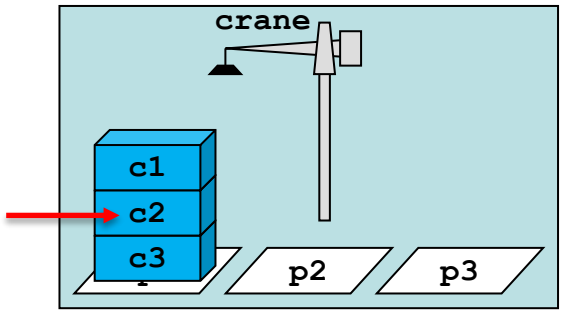


# Ejercicio

¿Y si solo quisiéramos mover **media** pila?

Un número de contenedores o hasta uno dado




Operadores:

- *take(crane,c1,c2,p1)*
- *put(crane,c1,p2,p2)*

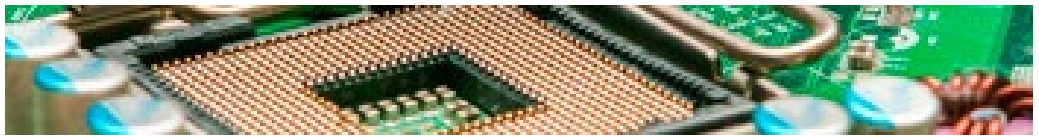
Tareas:

- *move-topmost(po,pd)* con método *take-and-put*
- *move-stack(po,pd)* con métodos *no-move* y *recursive-move*
- *move-ordered-stack(po,pd)* con método *move-stack-twice*

¿Cambios necesarios para mover hasta el contenedor c2?

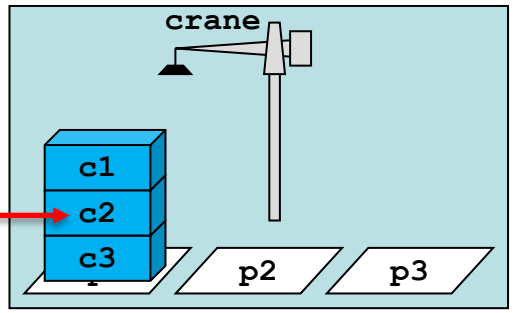
¿Y si quisiéramos mover muchas pilas?





# Ejercicio

## Mover un número de contenedores o hasta uno dado




- Añadir un nuevo parámetro en move-ordered-stack que fuera el contenedor dado hasta el que hay que mover o el número total de contenedores a mover. A partir de aquí hay dos posibilidades:
  1. Modificar el caso base en no-move y hacer que se detenga cuando se detecte ese contenedor.
  2. En las condiciones de move-topmost se comprobaría cuando se detecte ese contenedor y se dejaría de mover ese top cuando se hubiera llegado a ese contenedor.
  - En ambos casos, eso solo serviría para el primer movimiento de contenedores al pallet intermedio. El segundo movimiento sí debería hacerse contando la pila entera.

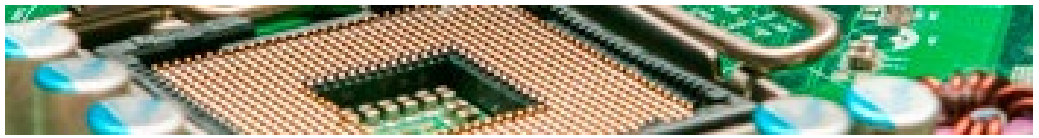
¿Cambios necesarios para mover hasta el contenedor c2?

- **Para ambas soluciones** es necesario añadir un nuevo parámetro (pero no en move-ordered-stack) que diga si estamos en el primer movimiento (primer move-stack) o segundo. De esta forma, si es el primer movimiento sí nos detenemos cuando se detecte el contenedor. Si es el segundo movimiento hay que trabajar con la pila entera.
- Esto indica que **podemos necesitar parámetros** que no estén en los niveles superiores más abstractos, pues a medida que vamos profundizando y detallando más lo que hay que hacer pueden aparecer parámetros que arriba en la jerarquía no necesitábamos.

¿Y si quisiéramos mover muchas pilas?

**Habría que manejarlas de forma iterativa**



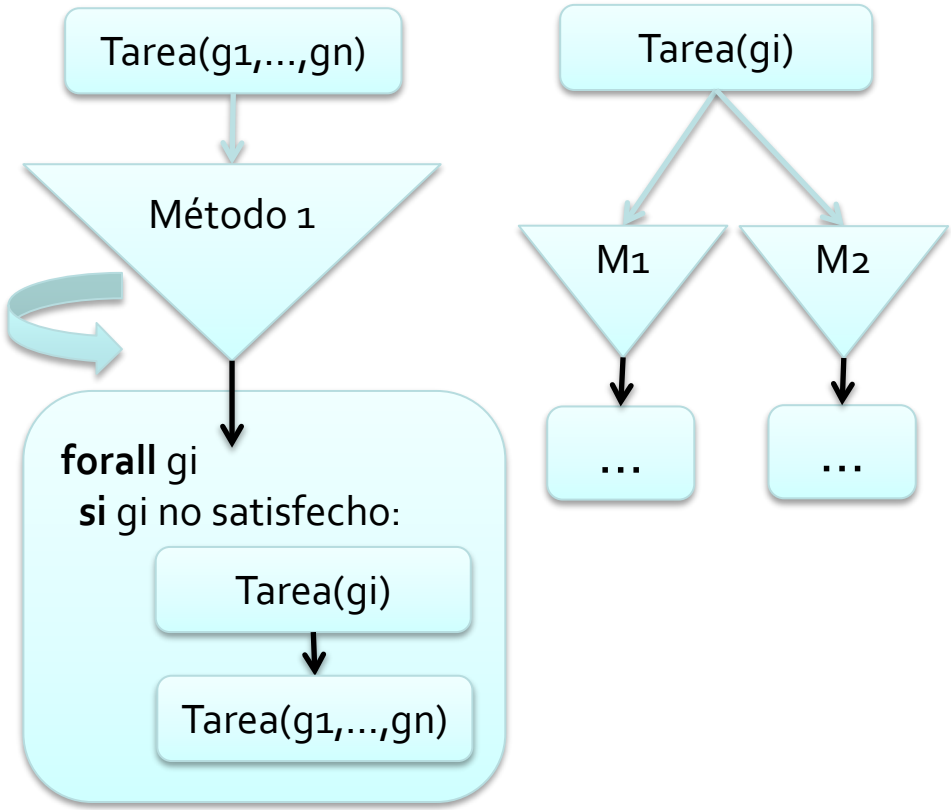


# Tipo de métodos: iterativo

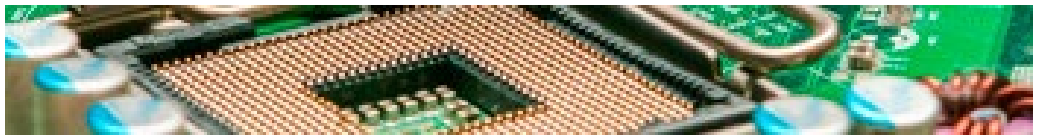
Método **iterativo**: normalmente se utiliza para procesar una lista de objetivos.

Para cada uno de estos objetivos, se comprueba si el **objetivo ya se satisface** en el estado actual. Si no es así, se descompone en una tarea que resuelve un objetivo individual y en una llamada recursiva a la tarea que resuelve ese método con el conjunto de objetivos inicial.

Cuando finaliza el recorrido de todos los objetivos, se retorna una lista de tareas vacía.



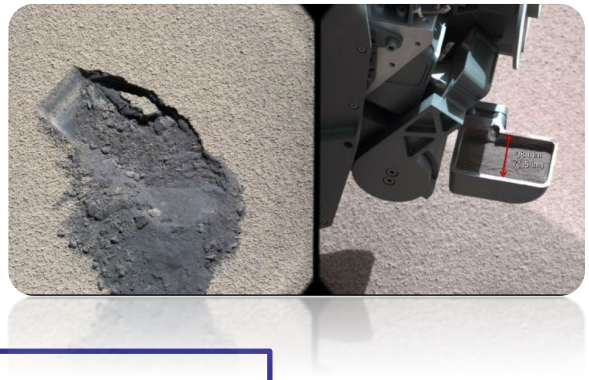


# Ejemplo

Ejemplo con pyhop sobre el dominio Mars rovers

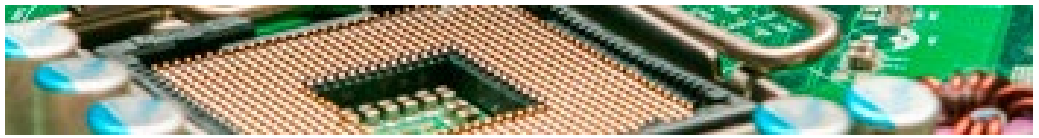
(https://mars.nasa.gov/)



```
goal1=pyhop.Goal('goal1')
goal1.data=[['soil','w2'], ['rock','w3'], ['image','o1','high-res']]
```

```
def rovers_iterative_goal_m(state, goal):
    for data in goal.data:
        if data[0]=='soil':
            if data[1] not in state.communicated_soil_data:
                return [('communicate_soil_data', data[1]), ('rovers_iterative_goal', goal)]
        if data[0]=='rock':
            if data[1] not in state.communicated_rock_data:
                return [('communicate_rock_data', data[1]), ('rovers_iterative_goal', goal)]
        if data[0]=='image':
            if {data[1],data[2]} not in state.communicated_image_data:
                return [('communicate_image_data', data[1], data[2]),
                                                    ('rovers_iterative_goal', goal)]
    return []
```





# Otros ejemplos

Se proporcionan **varios ejemplos**:

- *simple_travel_example.py*. Ejemplo de transporte en taxi o a pie para una única persona.
- *simple_travel_modified.py*. Versión extendida para controlar el número de veces que se llama a un taxi, si se desea llamar a un taxi, tiempo de espera y si se desea leer un libro mediante la espera.
- *simple_travel_with_goals_recursive.py*. Versión extendida para trabajar de forma recursiva con múltiples objetivos (transporte de varias personas).
- *simple_travel_with_goals_iterative.py*. Misma versión que la anterior, pero procesando los múltiples objetivos de forma iterativa.