Organización del Procesador

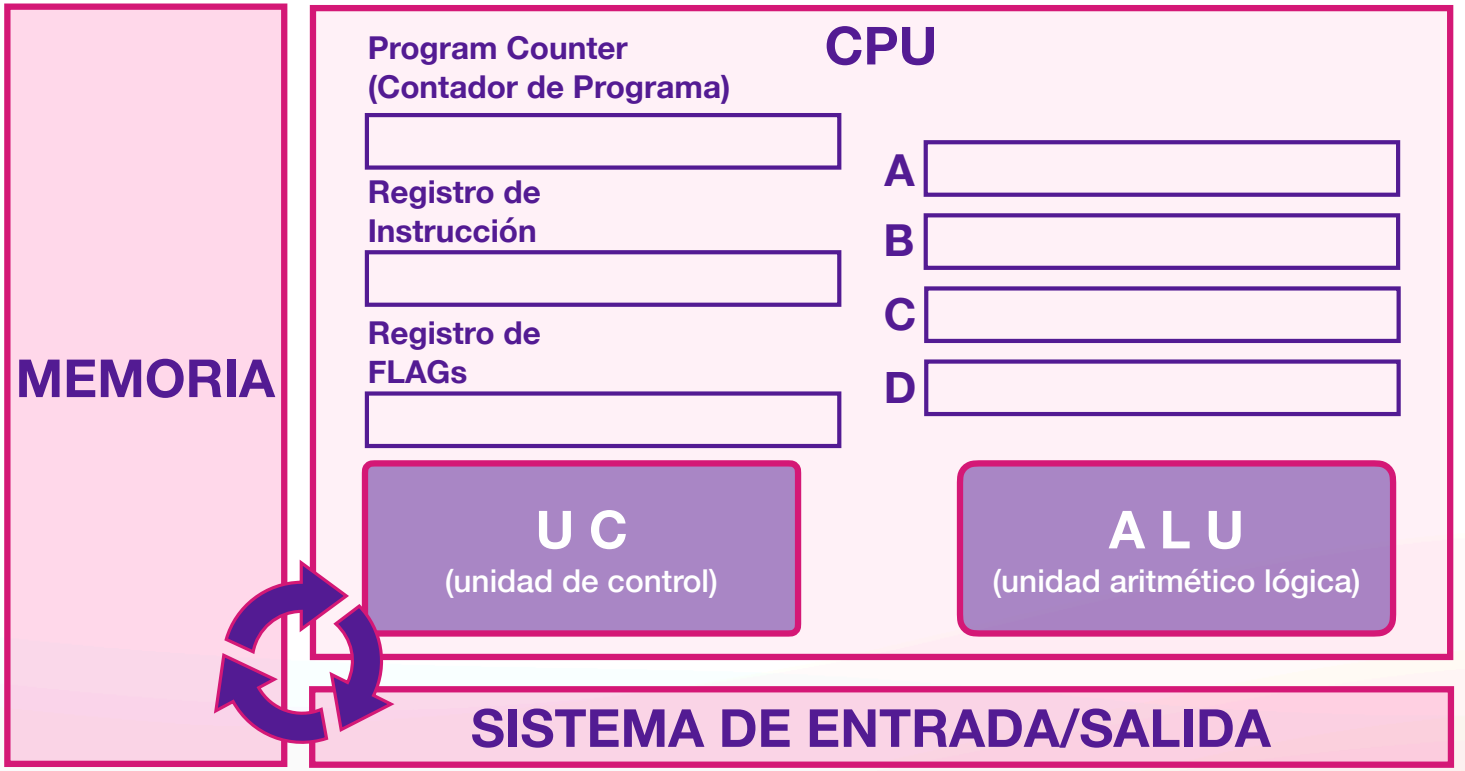
Pipeline

Departamento de Computación - UNRC

El camino a recorrer

- Un poco de Historia y Sistemas Numéricos
- Introducción a la Electrónica
- Representación de Información
- Cómo computar utilizando la electricidad
- Evolución y funcionamiento abstracto de una computadora
- Assembly X86
- Micro-programación (cómo fabricar un procesador)
- Eficiencia
  - **Pipelines**
  - Memoria Caché
  - Memoria Virtual

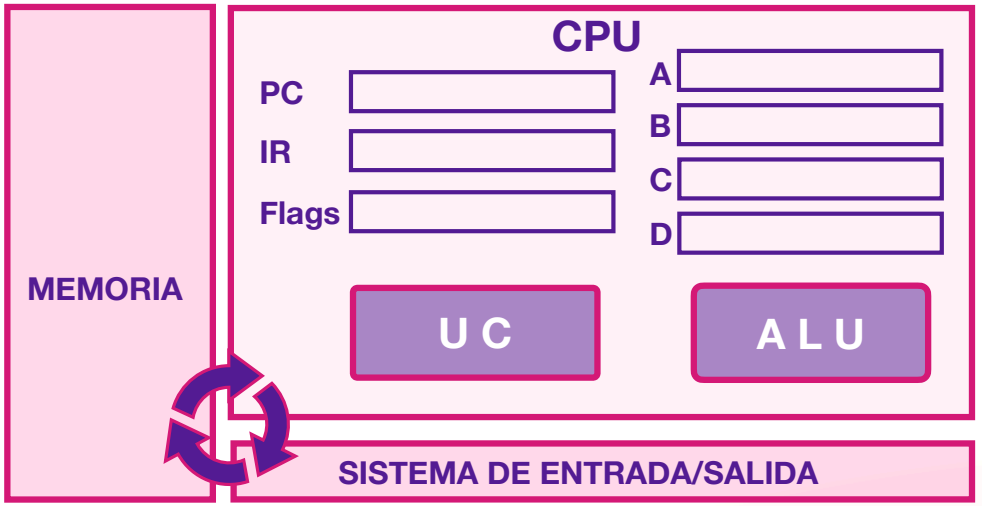
Arquitectura von Neumann



John von Neumann: (1903-1957) Físico Matemático Húngaro, Estadounidense. Participó activamente en el proyecto ENIAC y en el desarrollo de la EDVAC, donde propuso el concepto de almacenamiento de programas y datos en la memoria de la computadora, sentando las bases de la arquitectura de von Neumann, que se convirtió en el modelo predominante en las computadoras modernas.

Arquitectura von Neumann

Fetch: La Unidad de control obtiene de la memoria la próxima instrucción que indica el *contador de programa* (**PC**) y la almacena en el Registro de Instrucción (**IR**). Finalmente actualiza el **PC** indicando la dirección de la próxima instrucción.



Decode: La Unidad de control decodifica la instrucción y obtiene de la memoria (si fuere necesario) la información que involucra dicha instrucción.

Execute: La ALU ejecuta (calcula) el resultado de la operación y lo almacena en un registro o memoria.

Ejemplo de tareas asociadas a la ejecución de una instrucción

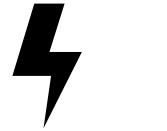
```
ADD   eax, [L1]
```

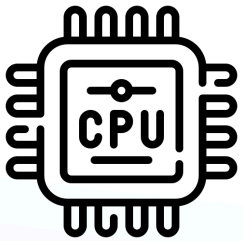

1) Recuperar la instrucción de la memoria (al IR)
2) Decodificar cuál es la instrucción
3) Calcular los operandos (memoria efectiva)
4) Recuperar los operandos (en registros)
5) Ejecutar la instrucción
6) Guardar el resultado

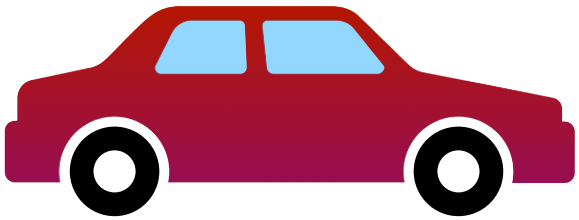
¿Cómo podemos hacer de manera más eficiente esta tarea?

Ejecución de instrucción

```
ADD    eax, [L1]
```



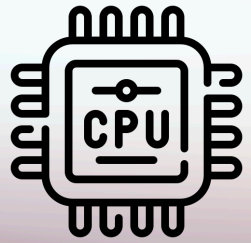




Ejecución de instrucción

```
ADD    eax, [L1]
```

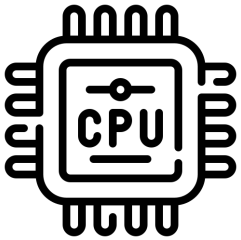
- **R** 1) Recuperar la instrucción de la memoria (al IR)
- **D** 2) Decodificar cuál es la instrucción
- **C** 3) Calcular los operandos (memoria efectiva)
- **M** 4) Recuperar los operandos (en registros)
- **E** 5) Ejecutar la instrucción
- **G** 6) Guardar el resultado



1) Carroceria
2) Paragolpes
3) Puertas
4) Neumáticos
5) Cristales
6) Pintura

Ejecución de instrucción

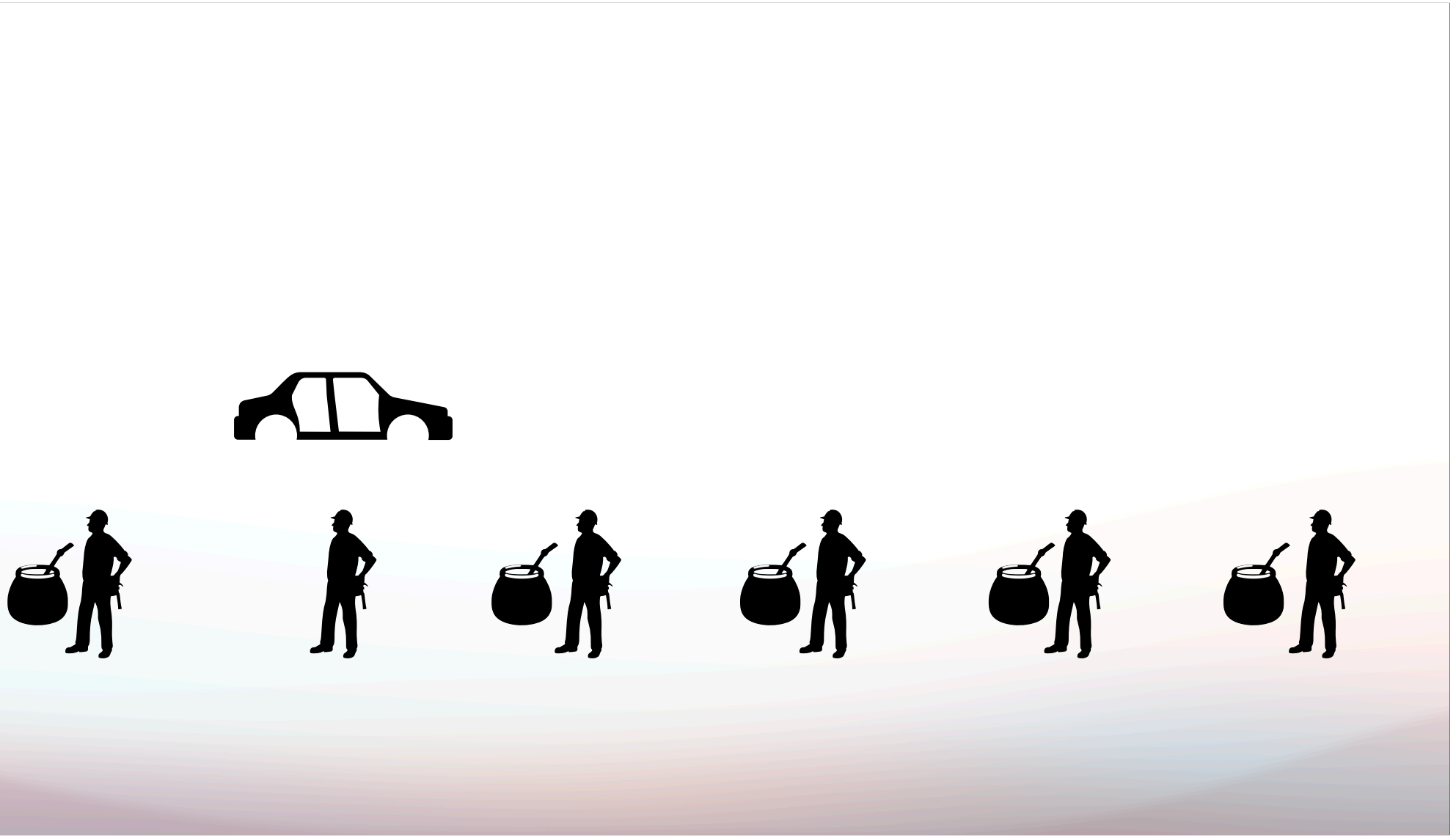
Supongamos que cada Sub Tarea consume **1 Unidad de Tiempo**

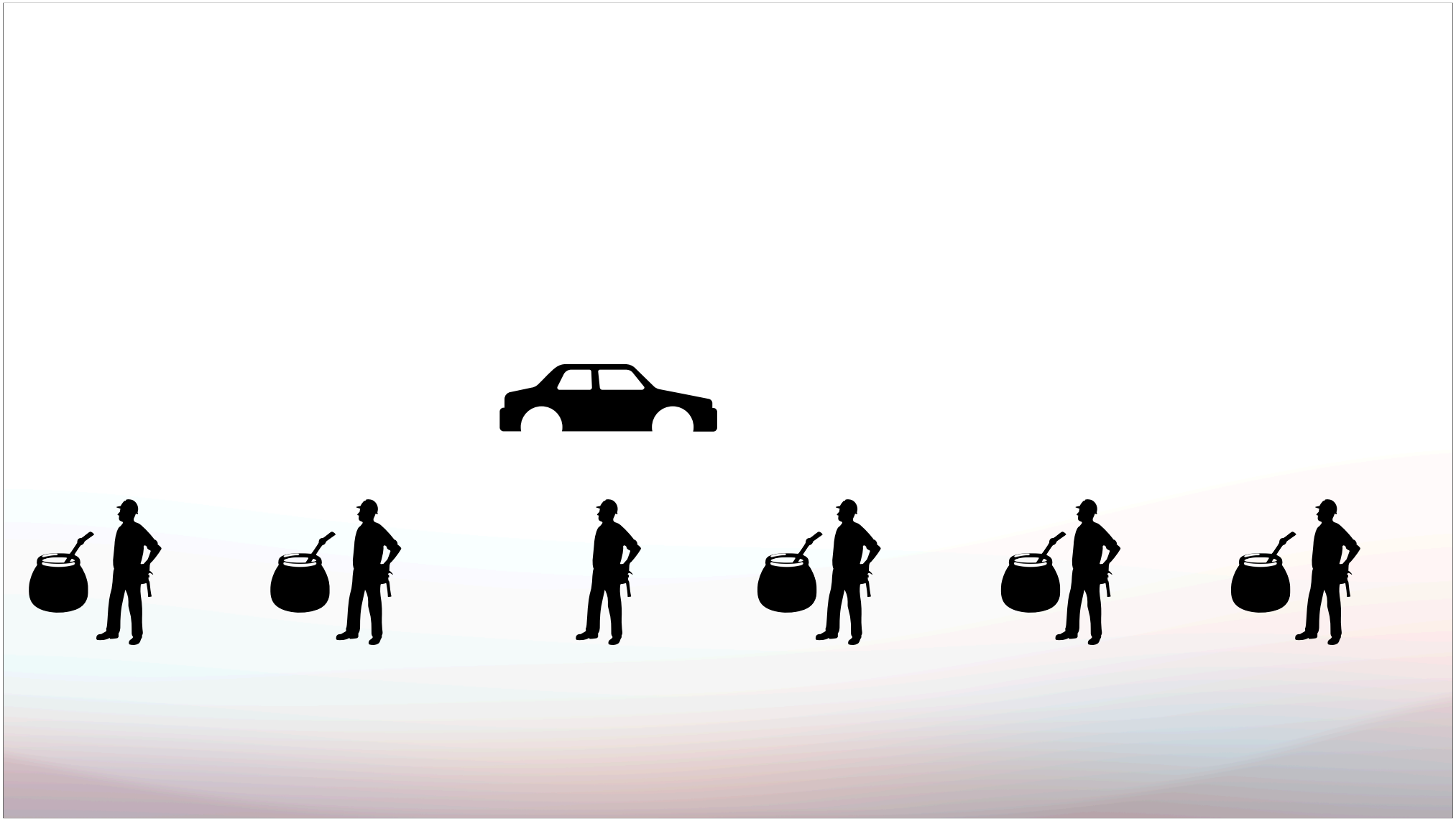


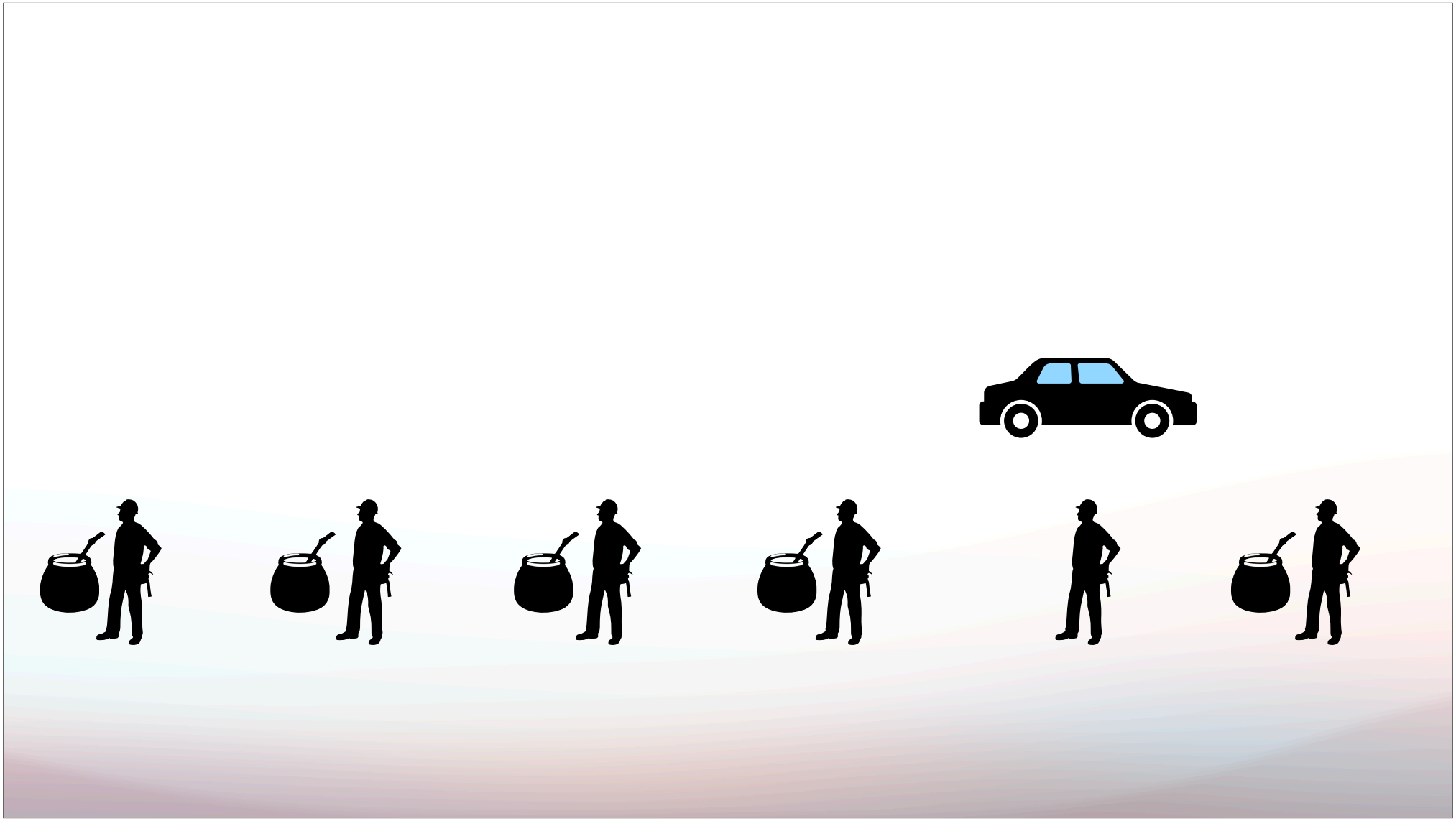
R D C M E G

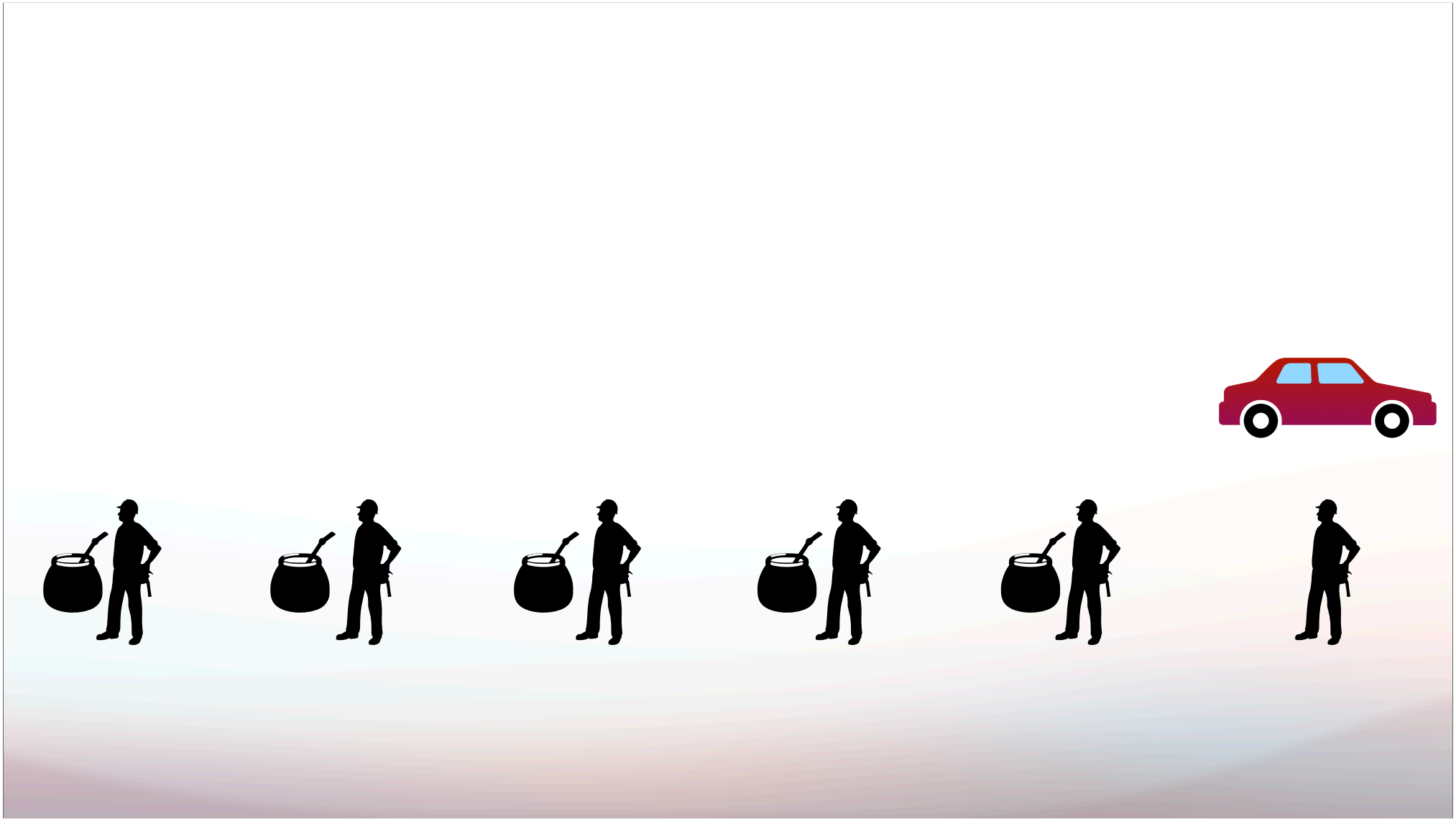
Tiempo

¿**Cuándo demora** ejecutar **una** instrucción / ensamblar **un** automóvil?
¿…y para **100** ?



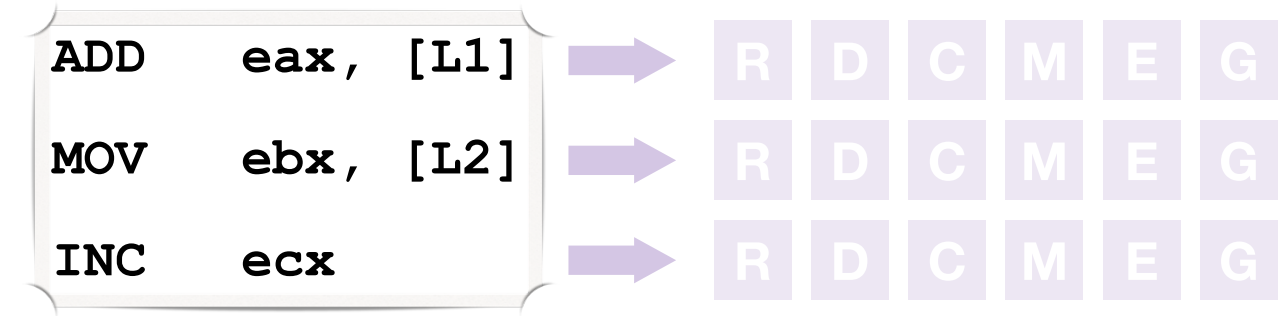


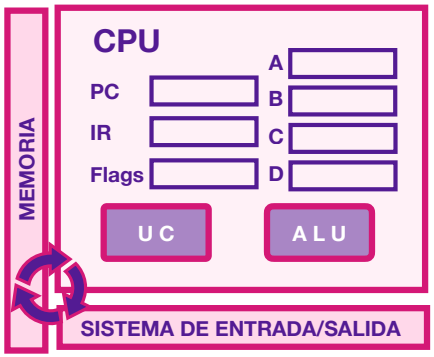




Ejecución estándar

```
ADD     eax, [L1]
MOV     ebx, [L2]
INC     ecx
```





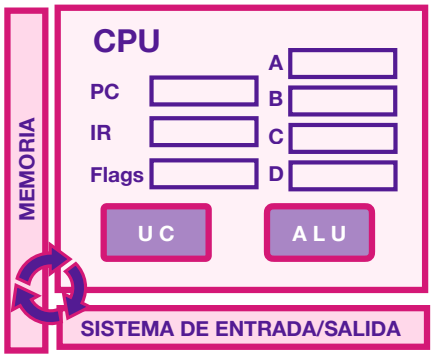
Tiempo

Si tenemos **3** (***n***) **instrucciones** de **6** (***k***) **etapas** cada una, y cada etapa demora **1**(***t***) unidad de **tiempo**, ejecutar nuestro código demora 18 unidades de tiempo.

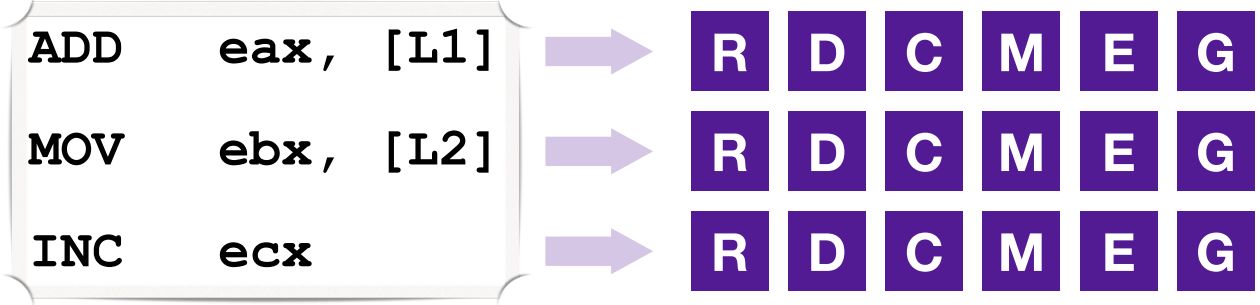
Tiempo Total (*n* instrucciones, *k* etapas, *t*) =

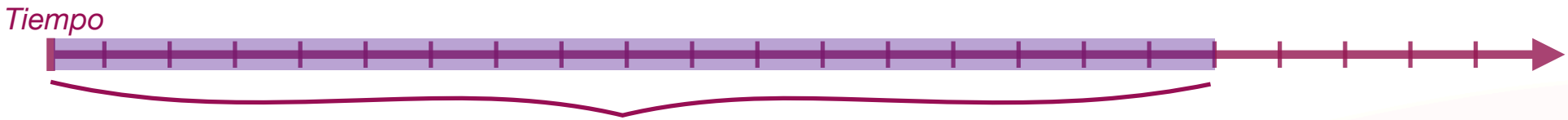
$$n * k * t$$

Ejecución estándar




```
ADD     eax, [L1]
MOV     ebx, [L2]
INC     ecx
```





Si tenemos **3** (***n***) **instrucciones** de **6** (***k***) **etapas** cada una, y cada etapa demora **1**(***t***) unidad de **tiempo**, ejecutar nuestro código demora 18 unidades de tiempo.

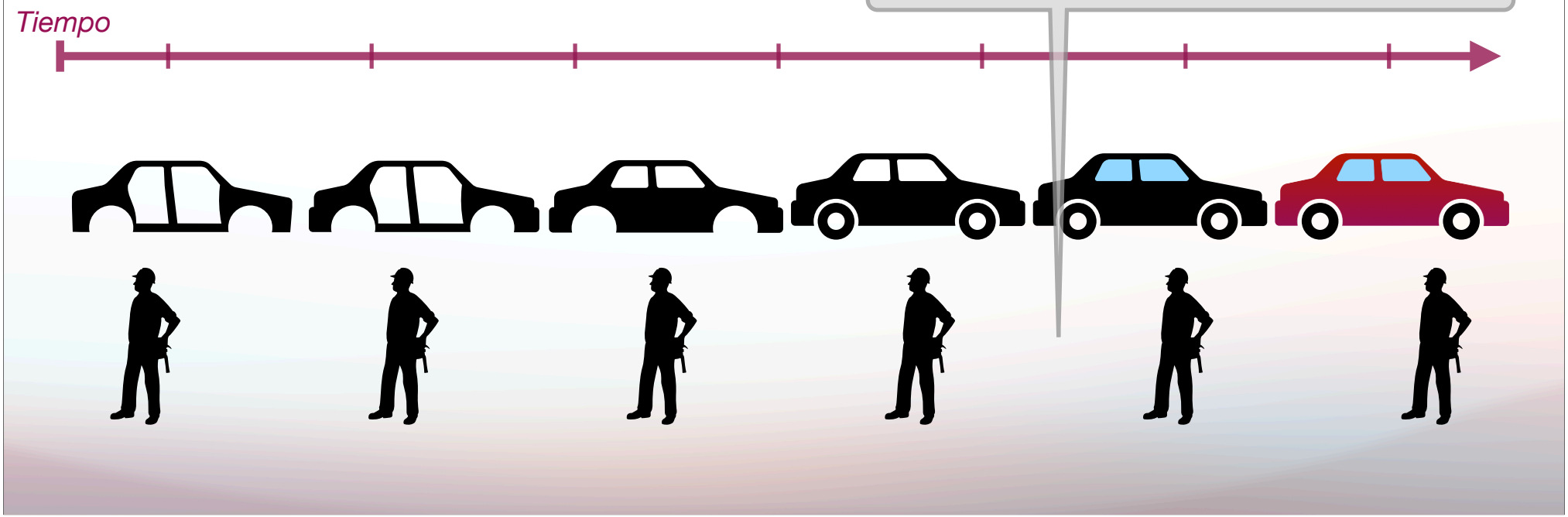
Tiempo Total (*n* instrucciones, *k* etapas, *t*) =

$$n * k * t$$

Pipelines

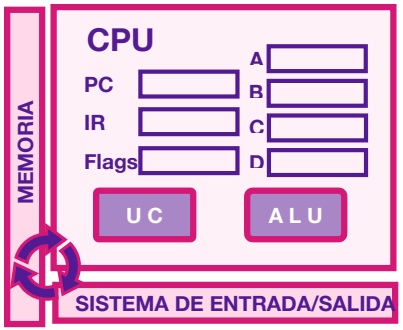
¿ Cuántas unidades de tiempo necesito para obtener el primer auto, cuántas para el segundo, tercero, … ?

¿qué condición necesaria necesitamos para poder hacer lo mismo con la ejecución de instrucciones?



Pipeline

```
ADD   eax, [L1]
MOV   ebx, [L2]
INC   ecx
```



R D C M E G R D C M E G R D C M E G

Ejecución estándar

Tiempo

R D C M E G
R D C M E G
R D C M E G

Ejecución en un **pipeline ideal**. Para ello cada etapa debe realizarse de manera autónoma e independiente de las otras.

Tiempo Total (*n* instrucciones, *k* etapas, *t*) =

$$(k * t) + (n-1) * t$$

La ganancia en la eficiencia en la medida que crecen la cantidad de instrucciones es de **k veces mejor**.

Conflictos en la ejecución de Pipelines

- **Conflictos con recursos**: Sucede cuando dos instrucciones necesitan utilizar el mismo recurso, en general la memoria.
- **Dependencia de Datos**: Sucede cuando una instrucción utiliza datos de la/s instrucciones (inmediatas) precedentes.
- **Saltos incondicionales**: Cuando cambiamos el flujo de ejecución de las instrucciones, independientemente de contexto (siempre cambia).
- **Saltos condicionales**: Cuando cambiamos el flujo de ejecución de las instrucciones dependiendo del contexto en el que se encuentra la ejecución del programa (estado).

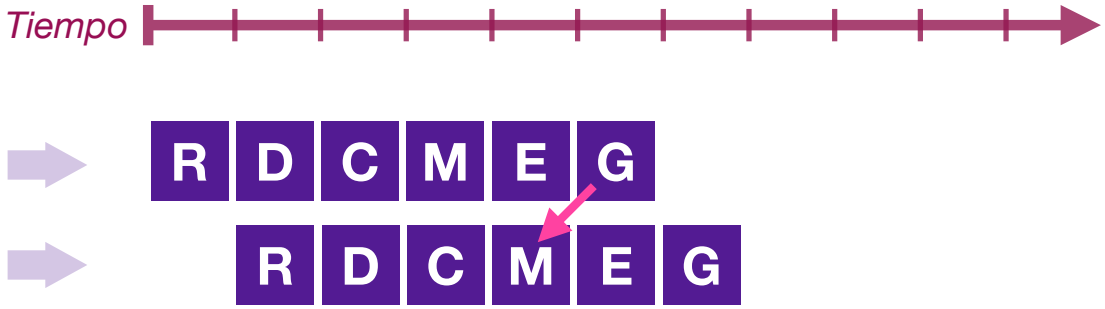
Conflictos en la ejecución de Pipelines

- **Conflictos con recursos**: Sucede cuando dos instrucciones necesitan utilizar el mismo recurso, en general la memoria.
- **Dependencia de Datos**: Sucede cuando una instrucción utiliza datos de la/s instrucciones (inmediatas) precedentes.
- **Saltos incondicionales**: Cuando cambiamos el flujo de ejecución de las instrucciones, independientemente de contexto (siempre cambia).
- **Saltos condicionales**: Cuando cambiamos el flujo de ejecución de las instrucciones dependiendo del contexto en el que se encuentra la ejecución del programa (estado).

¿Cómo podemos abordar estas situaciones?

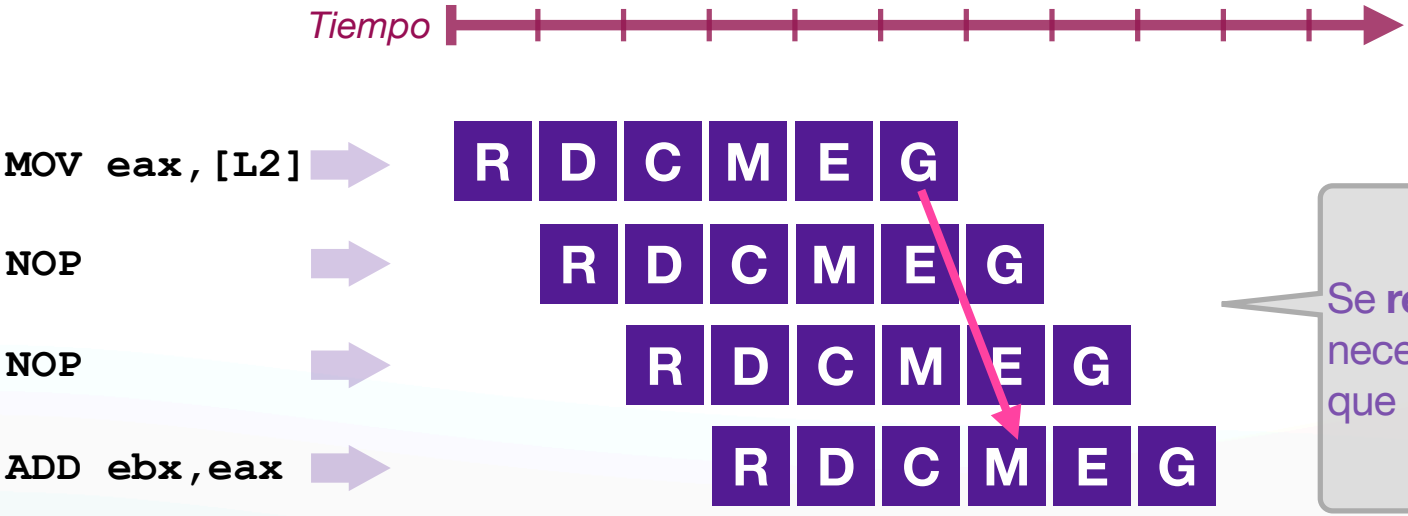
Conflictos en la ejecución de Pipelines - Dependencia de Datos

```
MOV   eax, [L2]
ADD   ebx, eax
```



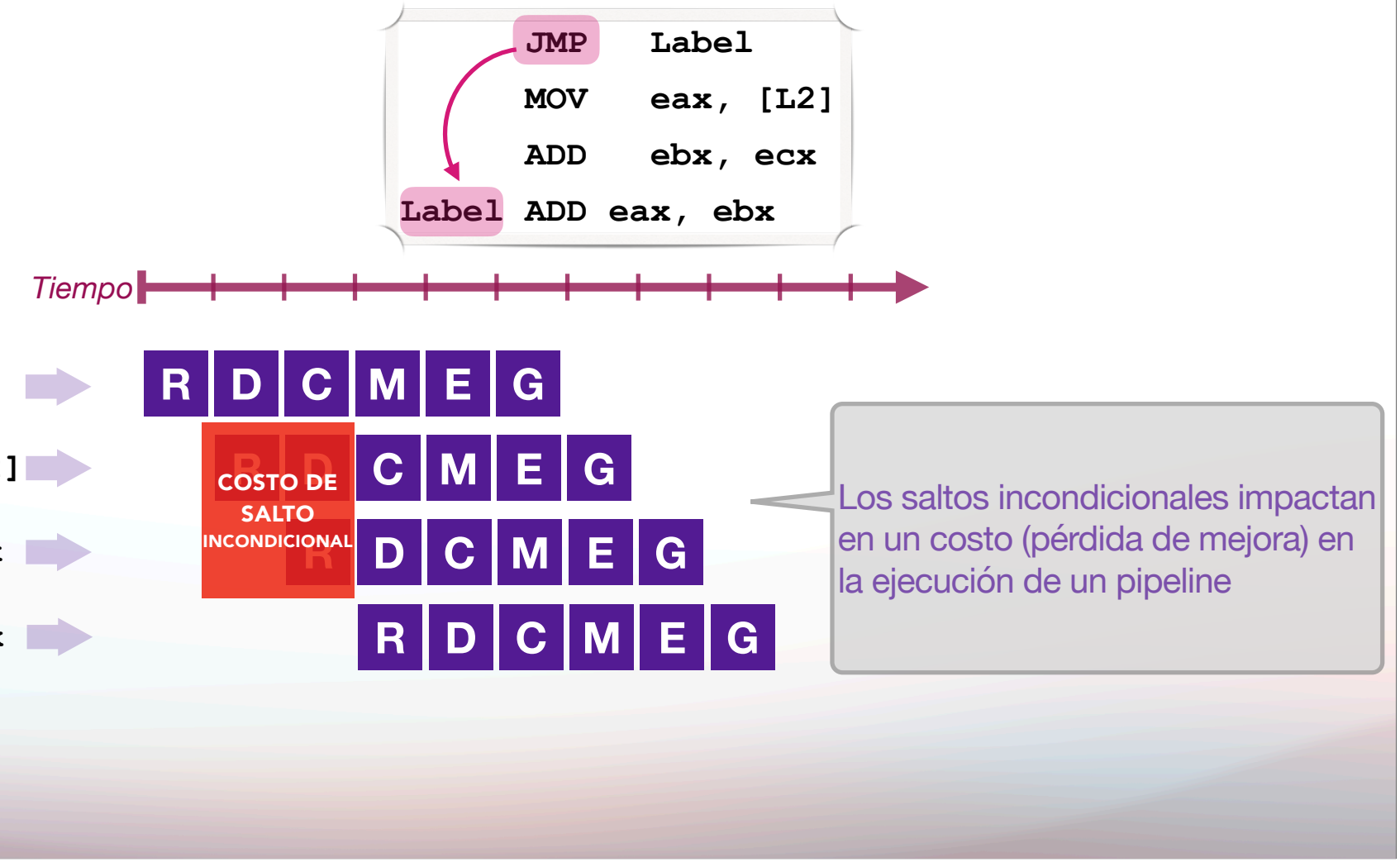
Conflictos en la ejecución de Pipelines - Dependencia de Datos

```
MOV   eax, [L2]
ADD   ebx, eax
```

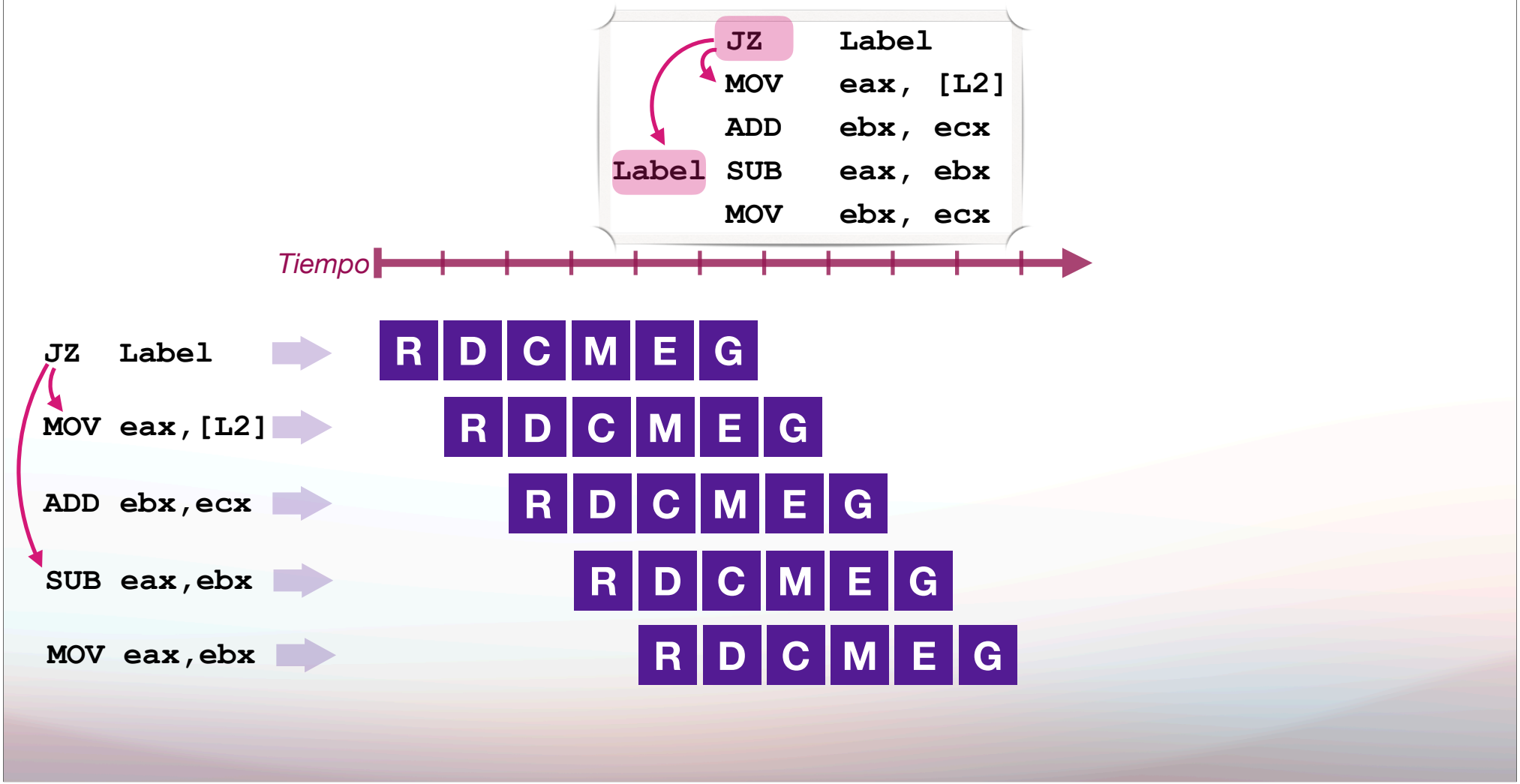


Se **retrasa** la ejecución las etapas necesarias mediante operaciones que no hacen NADA (**NOP**).

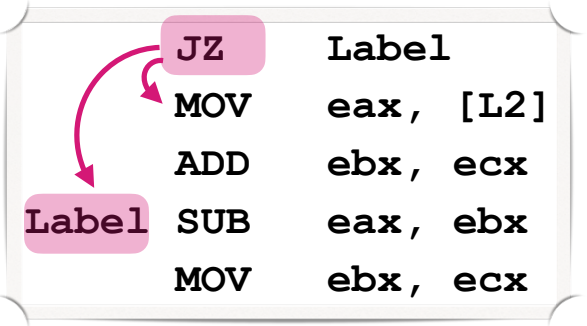
Conflictos en la ejecución de Pipelines - Saltos incondicionales



Conflictos en la ejecución de Pipelines - Saltos Condicionales



Conflictos en la ejecución de Pipelines - Saltos Condicionales




```
JZ   Label
MOV  eax,[L2]
ADD  ebx,ecx
SUB  eax,ebx
MOV  eax,ebx
```

- **Predicción estática:** Por ejemplo, asume que nunca va a saltar
- **Predicción dinámica:** Toman información de ejecuciones previas, por ejemplo, asume que la próxima vez va a tomar la misma decisión que la anterior.
- **Proceso paralelo**: en multiprocesadores, se procesan ambos branches en paralelo y con un flag se indica cuál fue tomado.