



75-08 Sistemas Operativos  
Lic. Ing. Osvaldo Clúa  
Lic. Adrián Muccio

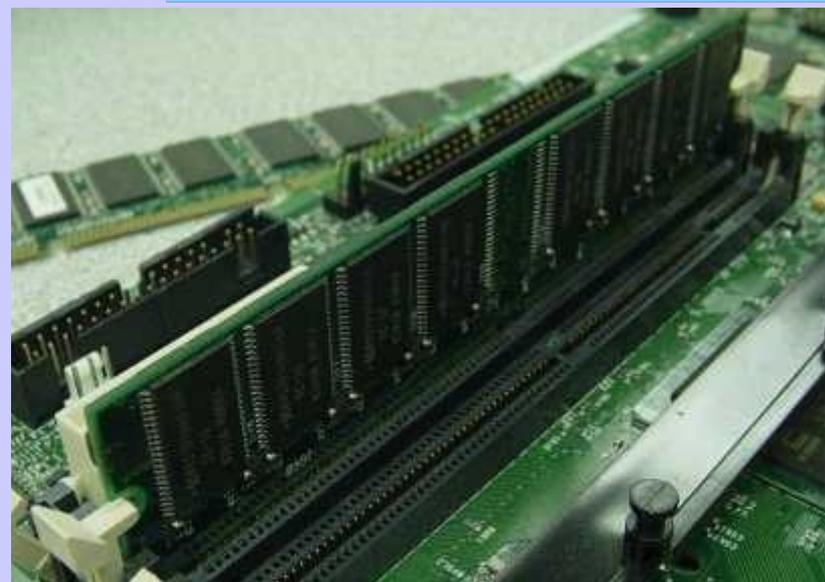
*Facultad de Ingeniería*  
*Universidad de Buenos Aires*

Administración de Memoria

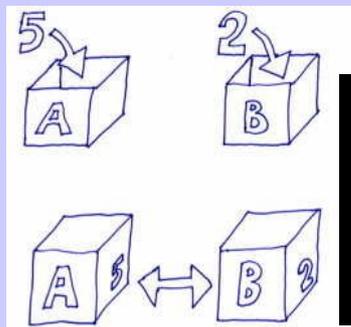
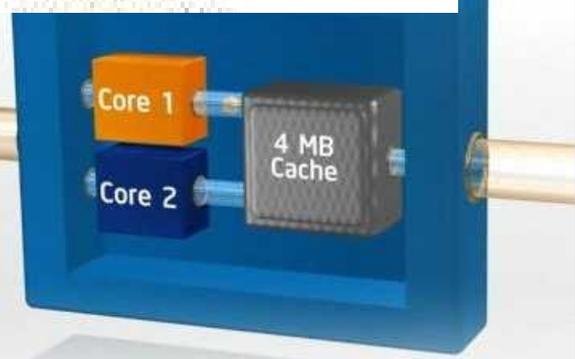
# Contexto

Agrupar las siguientes imágenes y explicar el criterio de agrupamiento.

# Contexto



```
int entradaMotor1 = 2;
int entradaMotor2 = 3;
int pwm = 9;
```



# Áreas de Administración de Memoria

## Aplicación

Memoria del programa.

Recolector de residuos.

## Hardware

RAM y Cache.

## Sistema Operativo

Traducción de direcciones.

Memoria Virtual.

# Administración de la memoria de una Aplicación

Suministrar la memoria necesaria para el código de un programa y sus estructuras de datos.

Asignación de memoria.

Reciclado del almacenamiento.

Está relacionado con el esquema de manejo de memoria soportado por el **lenguaje**.

El compilador se encarga de intercalar el código correspondiente.

# Vocabulario de manejo de memoria en los lenguajes

Declarar una variable:

Establecer nomenclatura (introducir el identificador) sin asignar memoria.

Definir una variable:

Asignarle memoria y posiblemente un valor inicial

Ambiente:

Porción de código durante el cual una variable está declarada.

# Vocabulario de manejo de memoria en los lenguajes (2)

Vida.

Intervalo de la ejecución en el cual una variable tiene memoria asignada.

Ámbito (scope).

Cuando una variable está en su ambiente y en su tiempo de vida.

# Tipos de Variables según su manejo de memoria

## Externa.

Se encuentra definida pero no declarada en el bloque.

## Estática.

Su vida se extiende a toda la duración del programa.

## Dinámica Automática.

La memoria se asigna y libera automáticamente cuando la ejecución ingresa en el ámbito de la variable

# Tipos de Variables según su manejo de memoria (2)

## Dinámica Controlada.

La ejecución del programa controla explícitamente cuando se asigna memoria a la variable.

## Garbage Collection.

Cuando la recuperación de memoria no referenciada es automática.

## Liberación manual

Cuando la recuperación se hace en el código del programa

# Estrategias de implementación

Las variables dinámicas automáticas (o *lexically scoped*) se manejan por medio de un *stack*.

Las variables dinámicas controladas se manejan por medio de una estructura llamada *heap*.

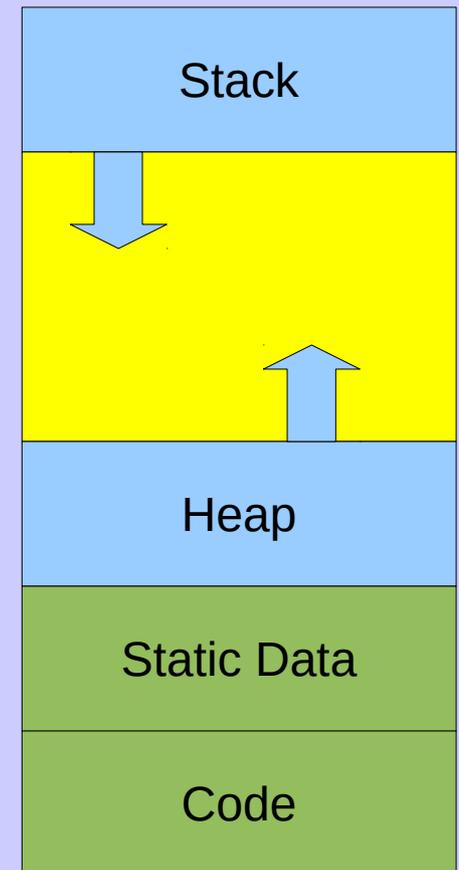
Que se puede implementar de varias formas  
(Listas o Buddies)

# Run Time Storage Organization

El compilador genera un esquema de uso de la memoria.

La memoria controlada se obtiene del heap (`new`, `malloc` ...)

El Stack mantiene el **Registro de Activación** o frame.



# Los Objetos y la Administración de Memoria

El Modelo de Objetos cambia los conceptos de Alojamiento en memoria.

Aparecen las Clases, Variables de Clase e Instancia y Ambientes de evaluación no ligados al ámbito léxico.

Cada Lenguaje OO tiene un modelo propio de memoria

Que se traduce a bibliotecas, se emula o se integra al Sistema Operativo.

# Los Objetos y la Administración de Memoria (2)

Hay una **patente** de un método (!!?)

**C++** mapea los objetos a un modelo procedural de ejecución usando bibliotecas.

En **Java**, la JVM brinda un ambiente mas propicio a los objetos.

En **.net** la **CLR** es la encargada de manejar la memoria.

# La ejecución y la memoria

El modelo teórico es el de memoria **PRAM**. En el mundo real:

En multithreading los resultados dependen del **modelo de memoria** de cada lenguaje.

En multiprocesadores el modelo de memoria incluye la resolución de **NUMA**.

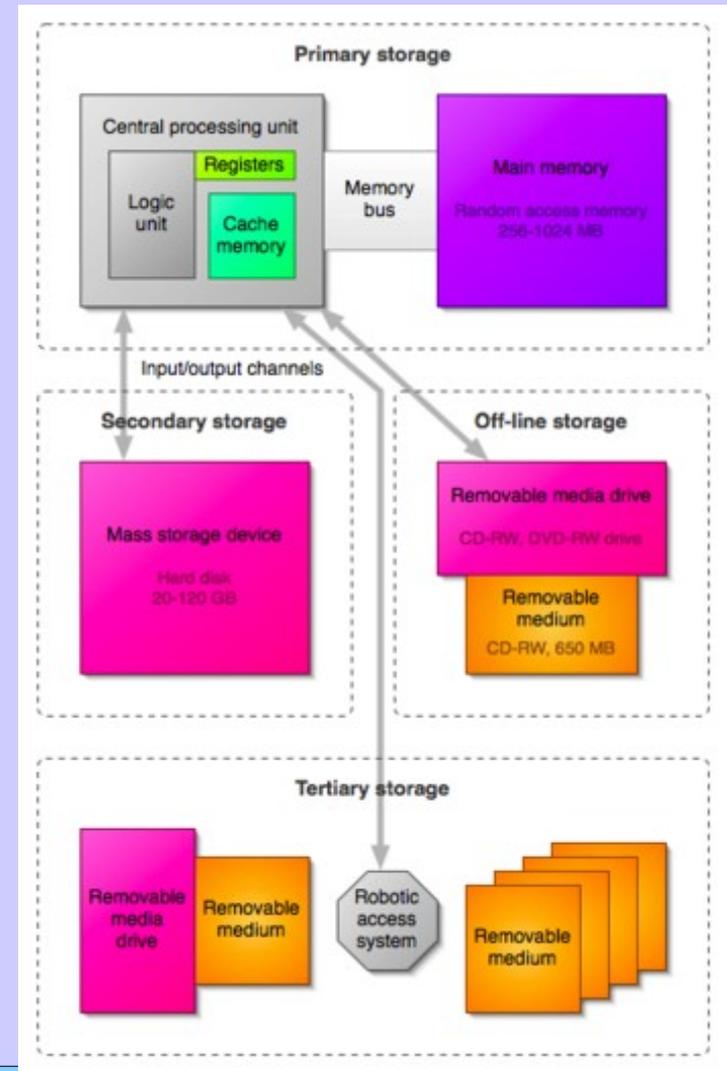
En la arquitectura el modelo especifica la coherencia de la **memoria cache**.

# Direccionamiento en Hardware

En el **ciclo de instrucción** hay mas de una referencia a memoria:

Estas son referencias a la Memoria Principal.

La **Memoria Cache** no es administrada por el Sistema Operativo.



# Modos de direccionamiento para código

El compilador genera las direcciones de las instrucciones:

Puede ser una dirección absoluta.

Puede ser relativa al Program Counter (  
**position independent**)

Puede generar las direcciones en cada instrucción como **SECD** para cálculo

# Modos de direccionamiento para datos

Dirección Absoluta

Operador Inmediato (literal)

Dirección indirecta (\*ptr)

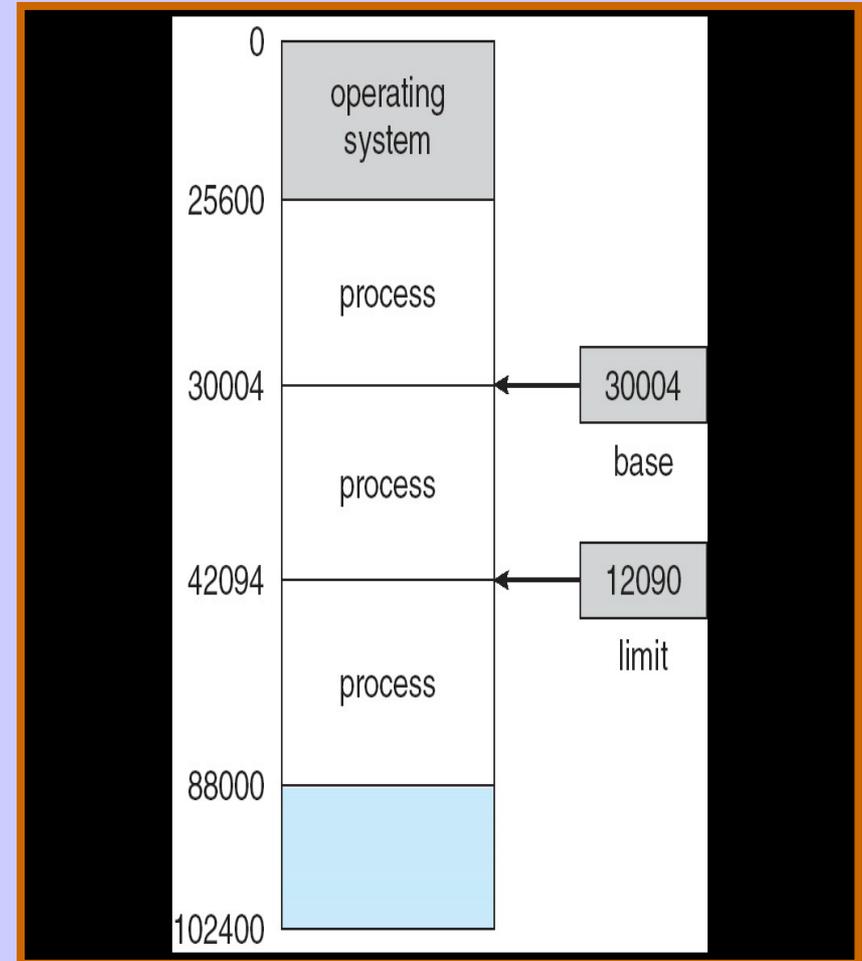
Dirección relativa a Program Counter (\*+ptr)

Base/Índice/Offset.

Base + Desplazamiento.

# Protección de Memoria (1)

El Sistema Operativo debe impedir a un proceso invadir la memoria de otro. Una forma de hacerlo es por medio de un registro base y uno límite.

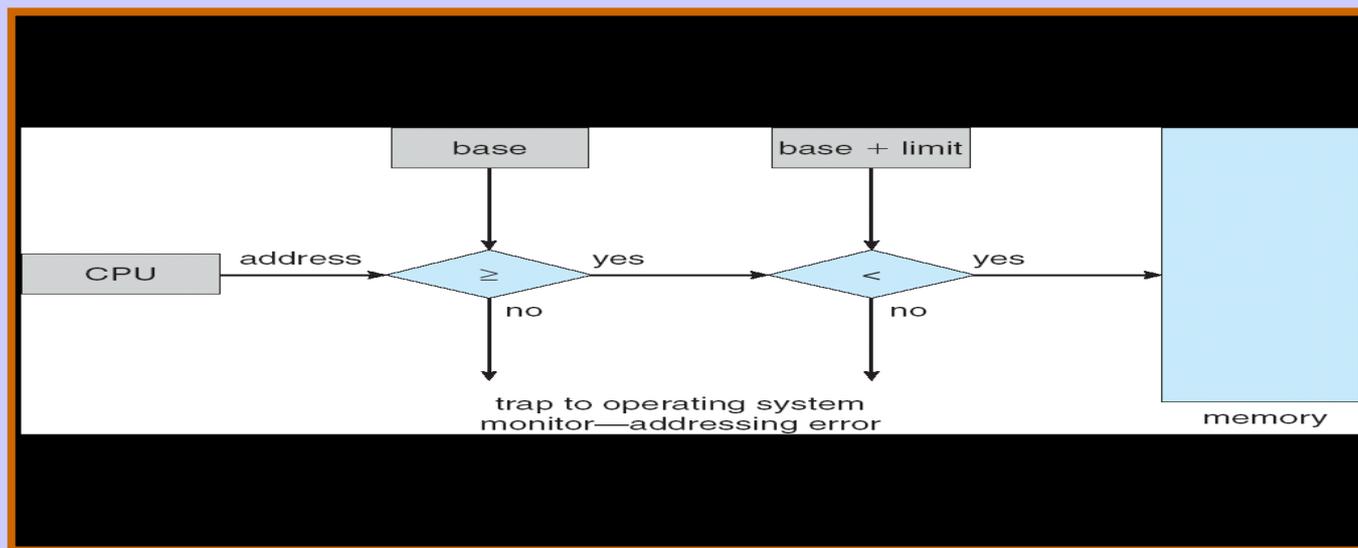


# Protección de Memoria (2)

El registro base puede usarse para direccionamiento indirecto.

Cada dirección se compara con ambos límites.

El Sistema Operativo no tiene restricciones al operar en Modo Kernel



# Administración de Memoria por el Sistema Operativo

El Modelo de Procesos añade un área especial para la administración del proceso.

La *U\_Area*

El Sistema Operativo debe proveer alojamiento para la ejecución del proceso

En inglés: *Memory Allocation*.

En multiprocesamiento, más de un proceso hace sus requerimientos de memoria.

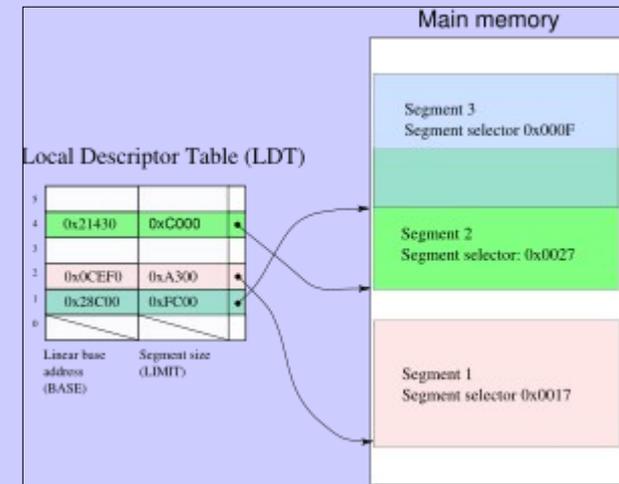
# Segmentación

Es una forma de proveer de más de un espacio de direcciones.

El programador (o el compilador) es quien lo usa.

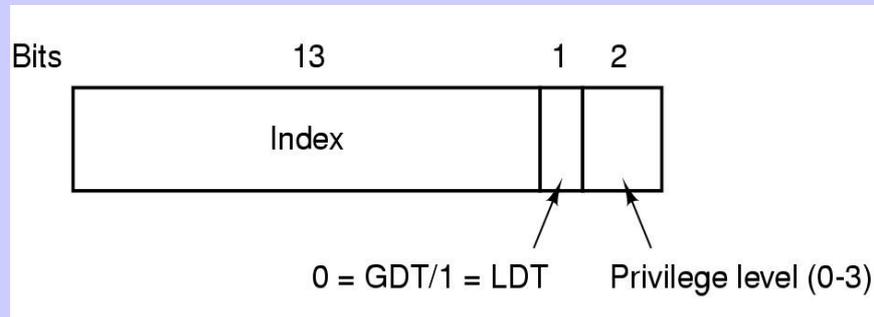
Su uso puede superponerse con el del paginado.

La protección puede asociarse a segmentos.

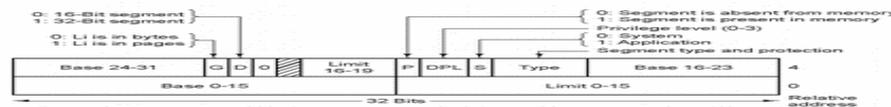


En Intel los segmentos solo pueden accederse desde un grupo de registros especiales: CS, DS, SS, ES, FS, GS (CodeSegment, DataSegment, StackSegment, ExtraSegment. Su valor se llama selector.

# Segmentación en INTEL



El valor del registro base se conoce como selector



El selector apunta a una entrada de la tabla de descriptores. Se obtiene la dirección base.

# Segmentos locales y globales

Algunos sistemas separan las tablas de segmentos (ej. **Intel**):

global (para el Sistema Operativo)

local (para los procesos)

o hasta una por proceso.

Esto permite que los procesos no compitan con el Sistema Operativo por memoria.

# Administrando el espacio libre/ocupado

Pueden usarse bitmaps o listas encadenadas.

Aparecen distintos Algoritmos de alojamiento:

Best fit (buscar el hueco mas ajustado)

Worst fit (buscar el hueco mas holgado)

First fit (buscar el primer hueco en que quepa)

# Buddy System

Usado en los dispositivos modernos que no tienen memoria virtual.

La memoria se asigna en cantidades potencias de dos.

Permite una recuperación rápida de huecos grandes.

Sufre de fragmentación interna.

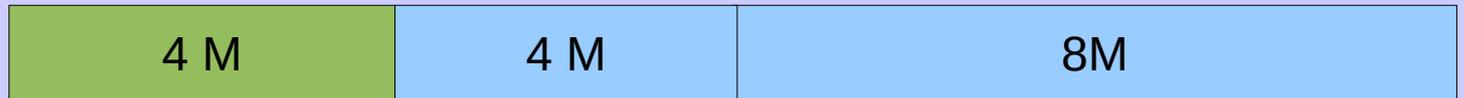
Su implementación es muy sencilla y rápida.

# Buddy System

Se piden 3 MB



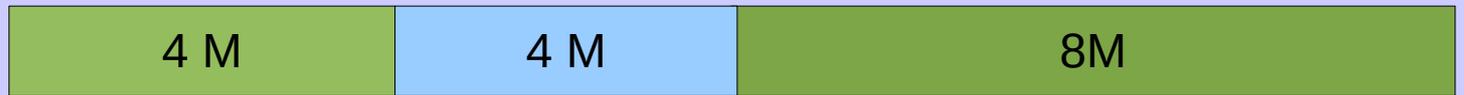
Se usan 4 MB



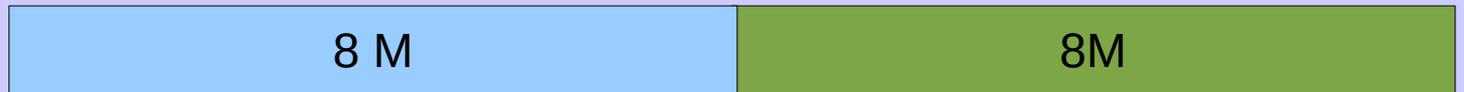
Se piden 5 MB

Se usan 8 MB

Se liberan 3 MB



Se obtiene un  
hueco de 8 MB



# Swapping

Consiste en pasar a Memoria Secundaria un proceso que no está corriendo.

y que hace mucho que no es activado (ej. esperando intervención humana).

Debe tenerse en cuenta la interacción con la I/O.

Se desplaza *todo* el proceso y se marca en el PCB.

# Swapping

Al hacer swap-in, un proceso puede volver a una dirección distinta.

El direccionamiento indirecto resuelve la reubicación.

