# Administración de memoria

Sergio Yovine

Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Sistemas Operativos, segundo cuatrimestre de 2015

# (2) Roles del SO

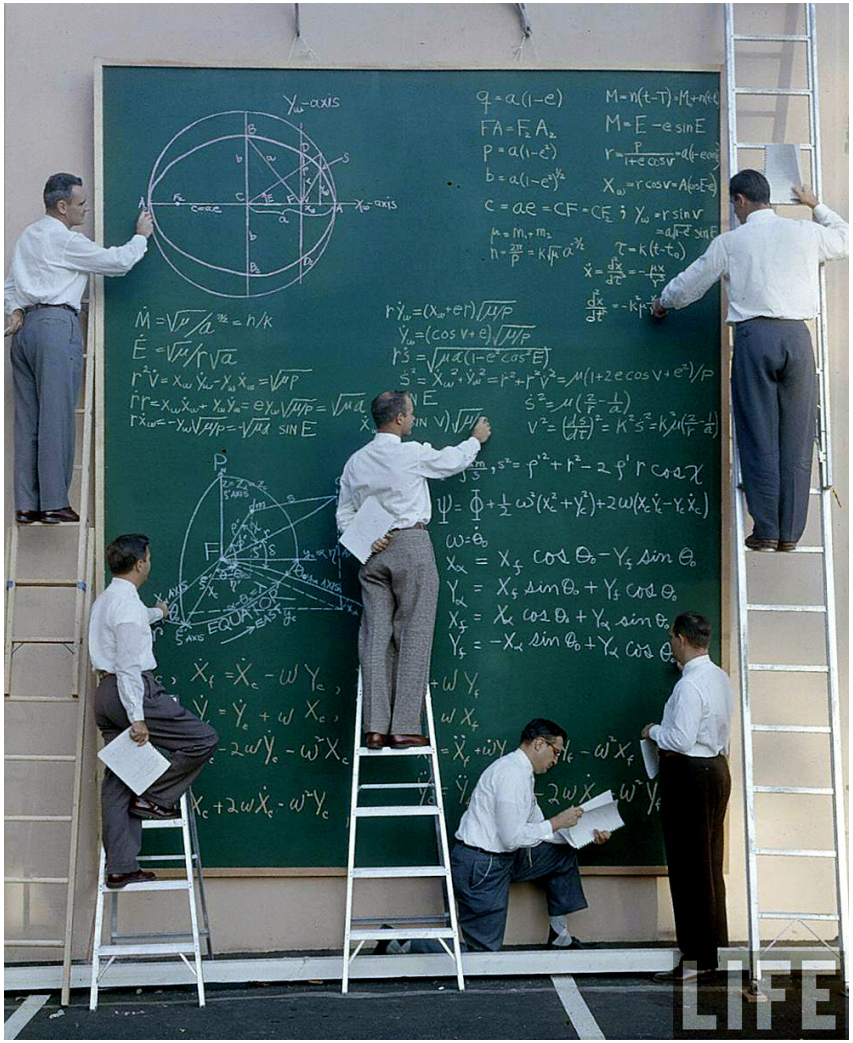
- Abstracción
  - Usuario, proceso, ...
  - Comunicación entre procesos (IPC)
  - Sistema de archivos
- Virtualización
  - Proceso, hilo de ejecución, ...
  - Memoria compartida o distribuida
  - *Memoria virtual*
- Proveedor de servicios
  - syscalls
- Administrador
  - Scheduler = cómo repartir el uso de (de los) procesador(es) entre los procesos
  - *Memory manager* = administrador de la memoria

# (3) Roles del SO en el manejo de la memoria

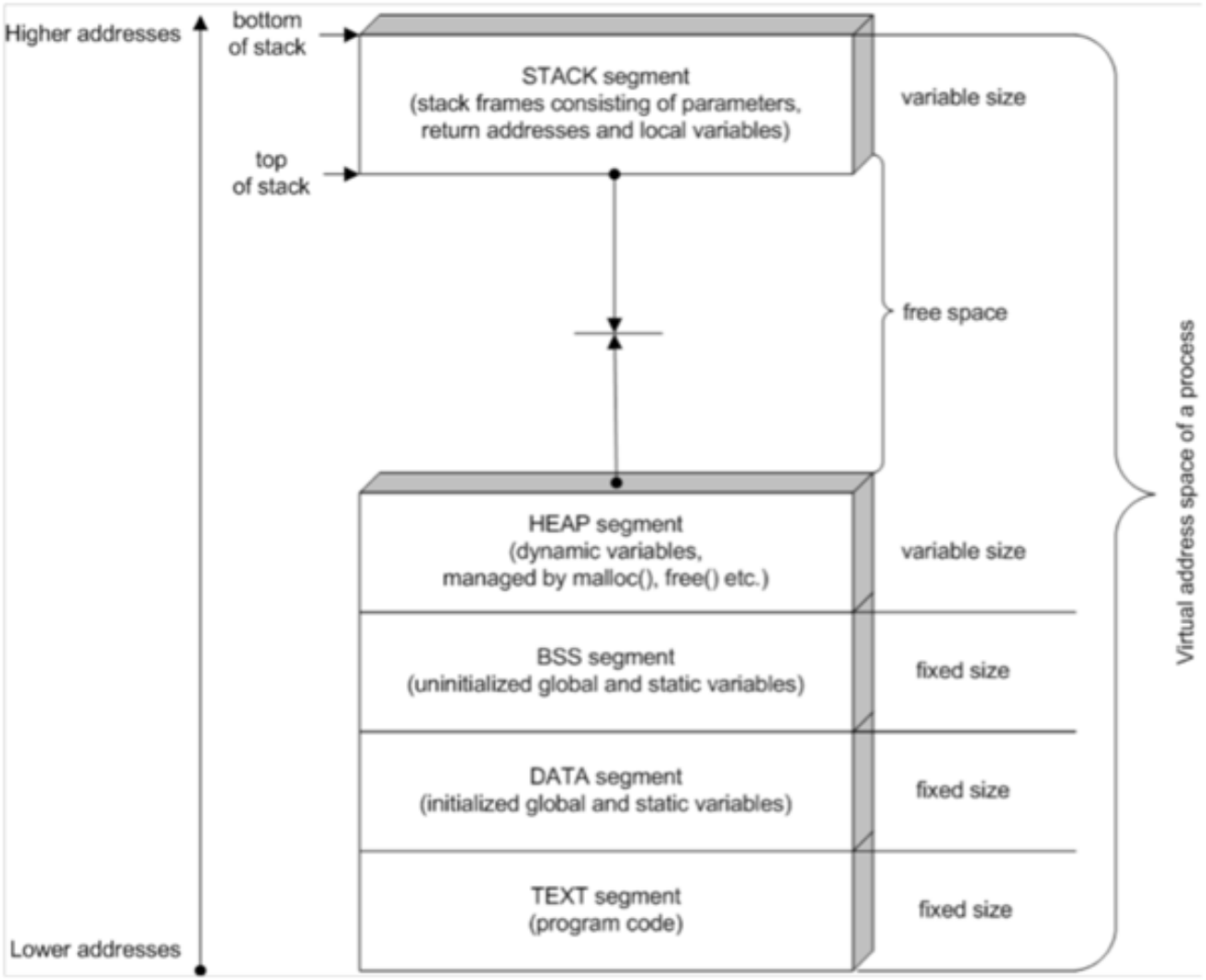
- Abstracción
  - Direcciones *lógicas* vs *físicas*
  - Memoria *compartida* o *privada*
- Virtualización
  - Más direcciones lógicas que físicas
  - Ayuda del HW: *Memory Management Unit* (MMU).
- Proveedor de servicios
  - API para reclamar y liberar memoria
- Administrador
  - Políticas de asignación de memoria a los procesos
  - Políticas de organización *espacial* de la memoria
  - Políticas de *modificación del contenido* de la memoria
    - Interacción lenguaje-SO-HW

# (4) Asignación/Organización de la memoria

- *Protección*:
  Memoria privada de los procesos
- *Reubicación*:
  Context-switch, swapping
- Manejo del espacio ocupado y libre.



# (5) Organización de la memoria de un proceso (layout)



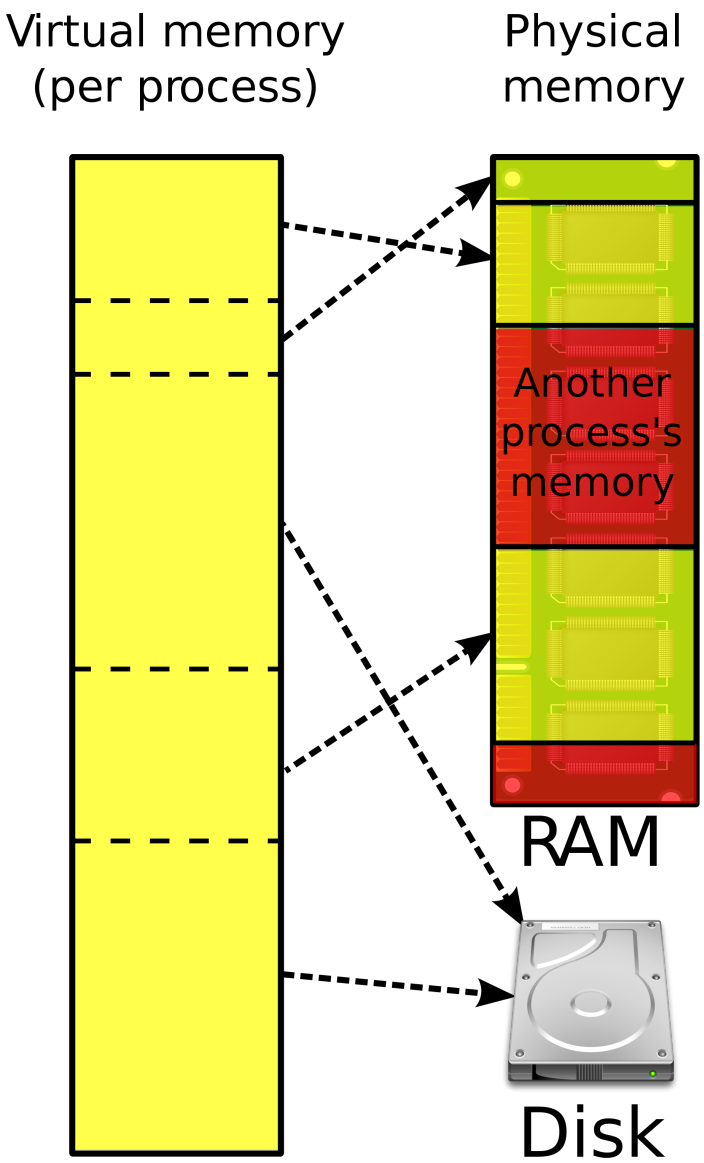
# (6) Memoria virtual y scheduling: Desalojo y reubicación

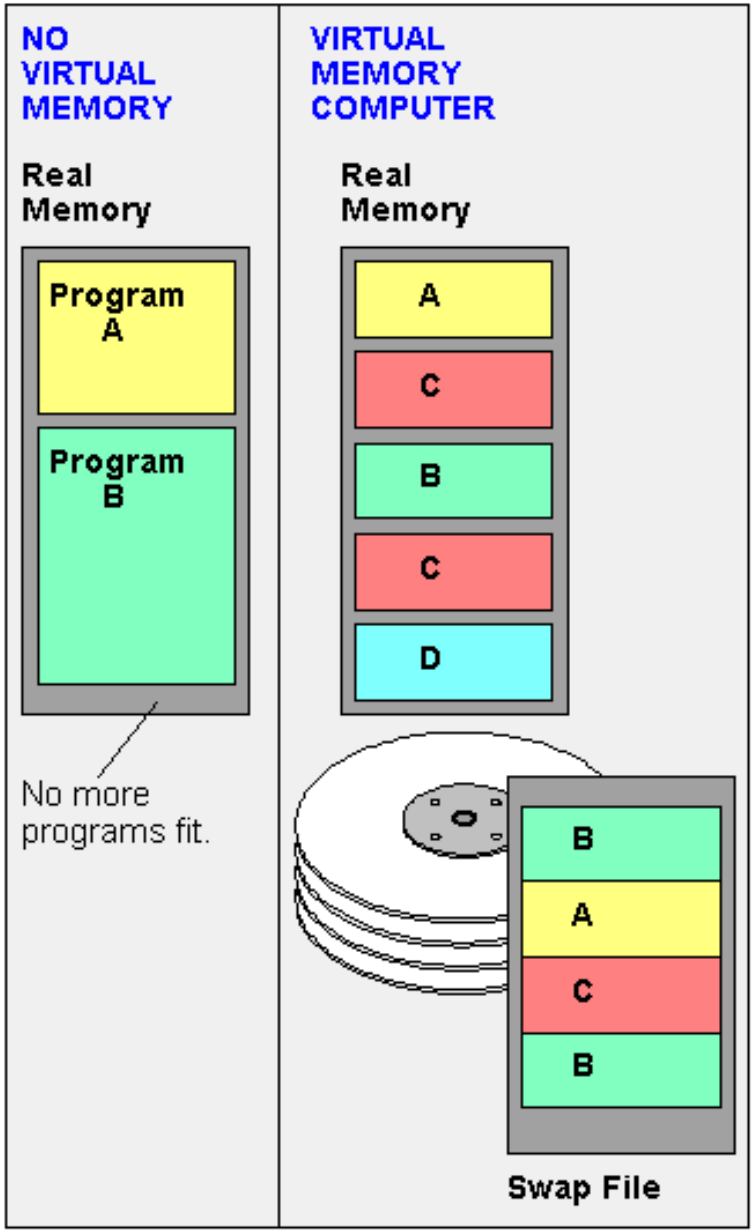
- Problema: No hay más memoria para el proceso en ejecución
  - Solución: ?
- Problema: La memoria del proceso seleccionado para ejecutarse por el scheduler está en disco
  - Solución: ?
- Problema: La dirección (o página) de memoria accedida por un proceso está en disco
  - Solución: ?
- Problema: La *frecuencia* de acceso a direcciones de memoria en disco es alta (*thrashing*)
  - Solución: ?

# (7) Memoria virtual y scheduling: Desalojo y reubicación

- Problema: No hay más memoria para el proceso en ejecución
  - Solución: (parte de) la memoria de algún proceso en espera o bloqueado se guarda en disco
  - Típicamente: parte de = página
- Problema: La memoria del proceso seleccionado para ejecutarse por el scheduler está en disco
  - Las direcciones de memoria son *relativas*
- Problema: La dirección (o página) de memoria accedida por un proceso está en disco
  - Solución: reemplazo de página (*swap*)
- Problema: La *frecuencia* de acceso a direcciones de memoria en disco es alta (*thrashing*)
  - Solución: predicción + política de scheduling + control de acceso + matar procesos

# (8) Memoria virtual



# (9) Administración de la memoria

- Complejidad espacial vs temporal
- Organización espacial
  - Partición de la memoria en *bloques*
  - Problema:
    *Fragmentación interna* =
    Memoria asignada pero no usada/usable
  - Solución: ?

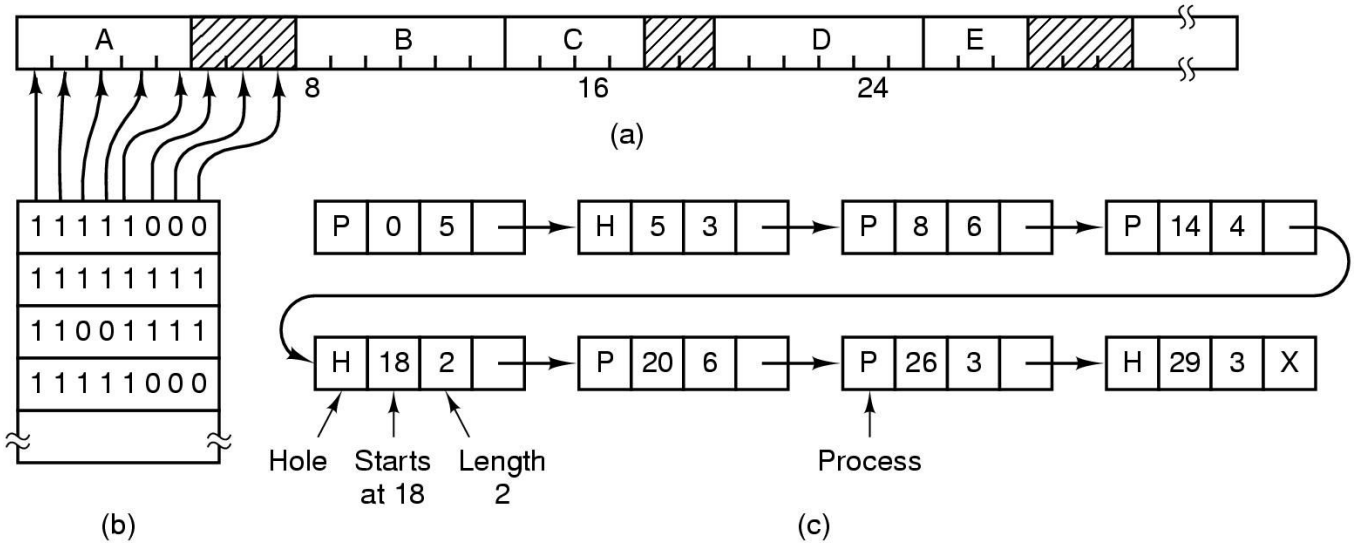
- Políticas de asignación
  - Problema:
    *Fragmentación externa* =
    Memoria libre pero imposible de asignar porque no es *contigua*
  - Solución: ?

# (10) Administración de la memoria

- Complejidad espacial vs temporal
- Organización espacial
  - Partición de la memoria en *bloques*
  - Problema:
    *Fragmentación interna* =
    Memoria asignada pero no usada/usable
  - Solución: *splitting* y *coalescing*

- Políticas de asignación
  - Problema:
    *Fragmentación externa* =
    Memoria libre pero imposible de asignar porque no es *contigua*
  - Solución: *asignar* el bloque más adecuado
  - Solución: *compactar* moviendo bloques de lugar

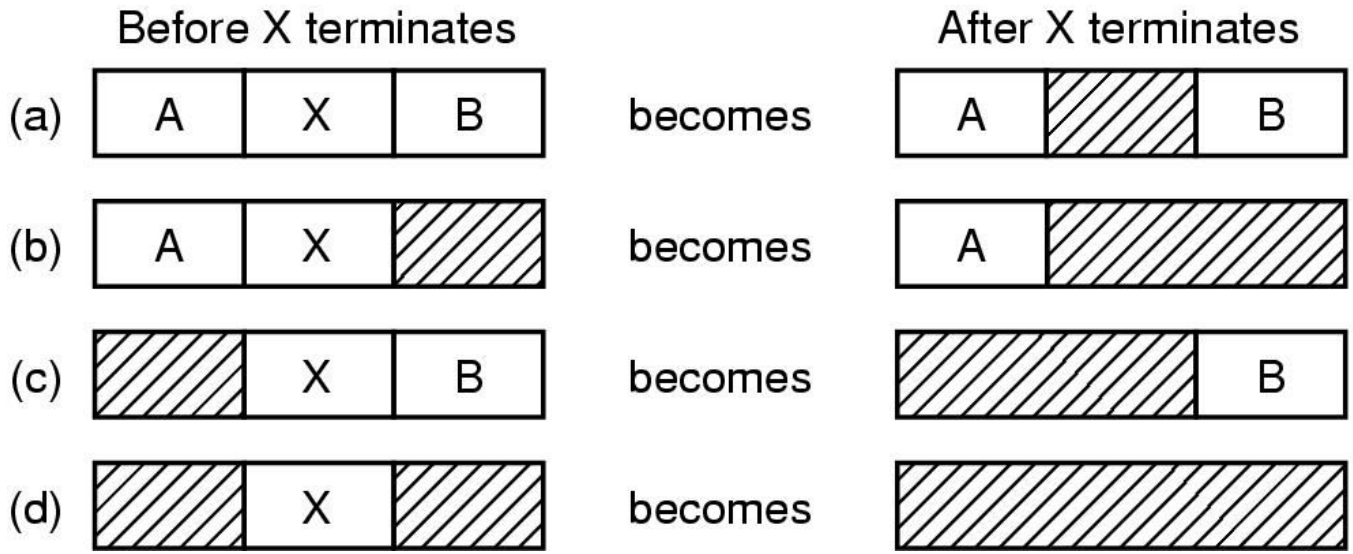
Bitmaps vs Listas enlazadas




Porción de memoria con 5 procesos y 3 espacios libres

# (12) Organización espacial: Listas Enlazadas

¿Qué pasa cuando el proceso X libera memoria?



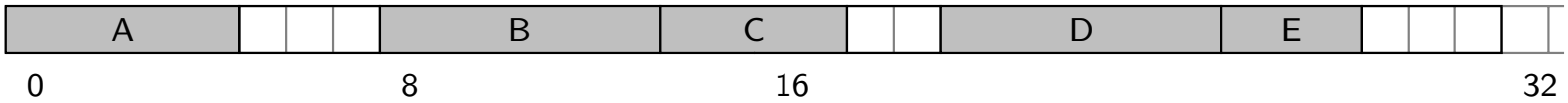

Cuatro combinaciones de vecinos para el proceso X
*Coalescing*: (b), (c) y (d)

# (13) Políticas de asignación

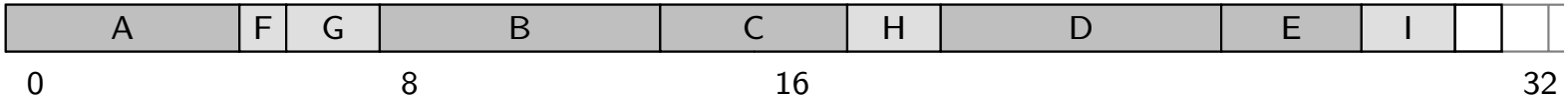
- *First fit*: asignar en el primer espacio libre donde entra.
- *Next fit*: Igual a *first fit* pero buscando desde la posición de la última asignación.
- *Best fit*: asignar dónde entra más justo.
- *Worst fit*: asignar dónde entra más holgado.
- *Quick fit*: se mantiene una lista de los bloques libres de los tamaños más frecuentemente solicitados.
- *Buddy system*: usa *splitting* de bloques.

# (14) Políticas de asignación: análisis

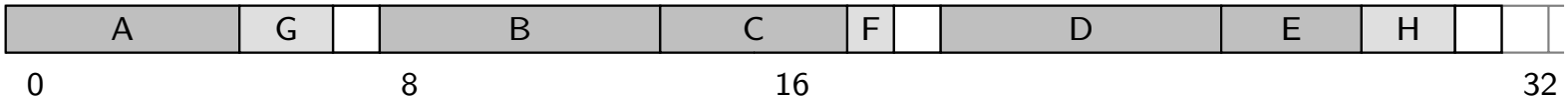
- Fragmentación ¿Best fit es mejor que First fit?
- Consideremos el siguiente ejemplo.
- Estado inicial (el mismo que vimos antes)




- Secuencia de solicitudes: (F, 1), (G, 2), (H, 2), (I, 2)
- First fit




- Best fit: ¡no hay lugar para I!



# (15) Políticas de asignación: análisis

- Análisis de First fit (D. Knuth citado por A. Tanenbaum)
- Asumiendo:
  - Solicitudes/liberaciones aleatorias con distribución uniforme en tiempo, espacio y tamaño
- *La regla del 50 %*
  - el *largo* de la lista de bloques libres tiende a la mitad de los bloques usados
- *La regla de la memoria no usada*
  - $m$ es la memoria *usada* en promedio por los procesos
  - $k > 0$ una cierta constante
  - en promedio, la memoria *total ocupada* es

$$m \cdot (1 + k/2)$$

- Inconvenientes de este análisis:
  - Hipótesis *no realistas*.
  - Conclusiones *poco relevantes* en la práctica.

# (16) Políticas de asignación: análisis

- $M =$ máx. cantidad de memoria usada en cualquier momento
  - $M > m$
- $B =$ tamaño máximo de un bloque
  - En la práctica, $\log B > 1 + k/2$
- El espacio requerido por *first-fit* en el *peor caso* está acotado *inferiormente* por

$$M \cdot \log B$$

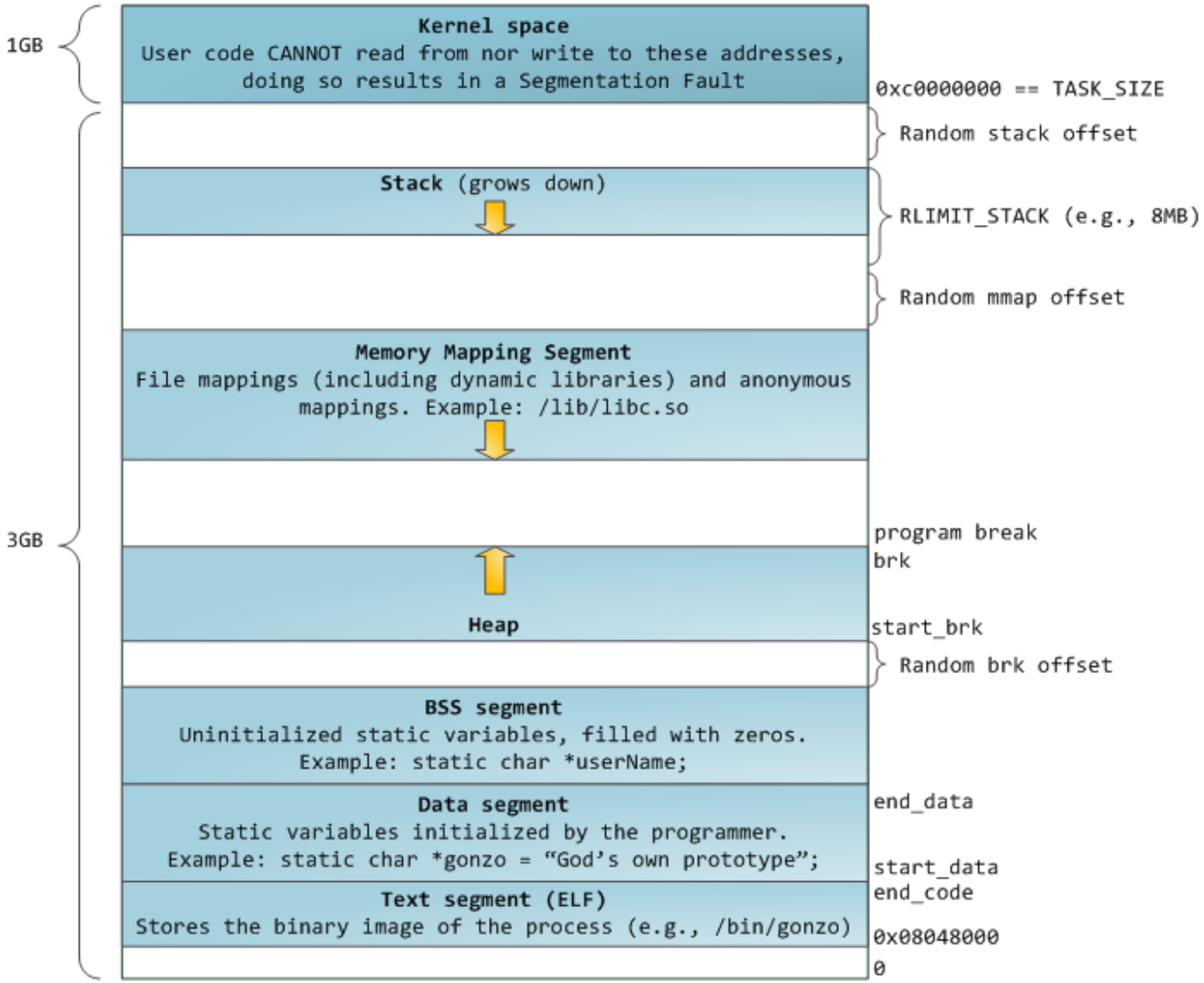
- La cota inferior es *peor* para *best-fit*

$$M \cdot B$$

# (17) Administración de la memoria: API

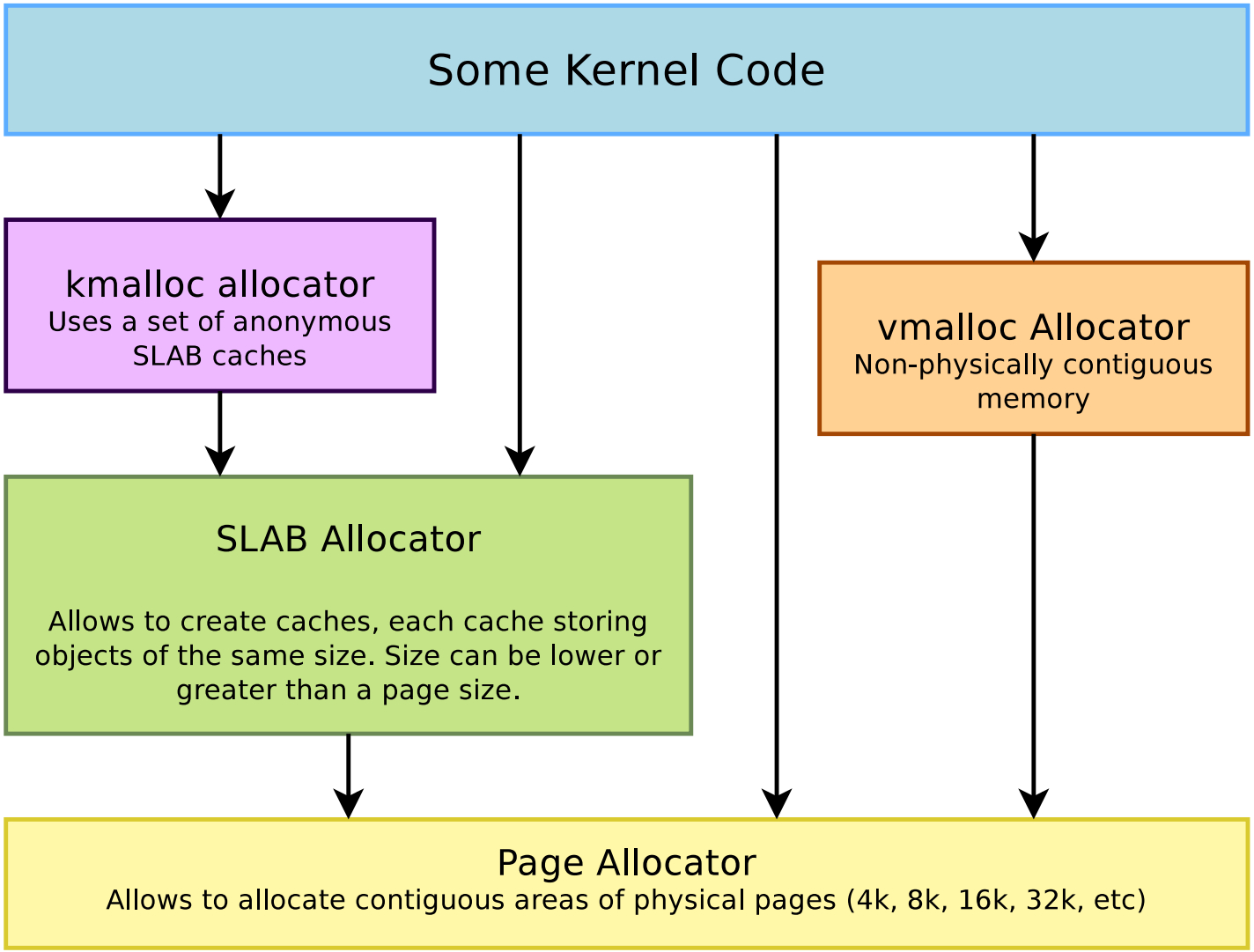
- Sistema operativo
  - Memoria del kernel
  - Memoria de los procesos
- Lenguaje de programación
  - Explícita
    - Librerías: e.g., `malloc()` y free() en `libc`
    - Primitivas: e.g., `new` y `delete` en `C++`
  - Implícita
    - No controlada por el programa: e.g.,`Haskell`
    - Parcialmente controlada/guiada: e.g., `Swift`
  - Híbrida
    - Creación explícita y liberación implícita: e.g., `Java`
    - Librerías y primitivas: e.g., `C++`
    - Librerías y análisis a la compilación: e.g., `RTSJ`
- Integrada: e.g., `Android`, `iOS`

# (18) Administración de la memoria en Linux: layout

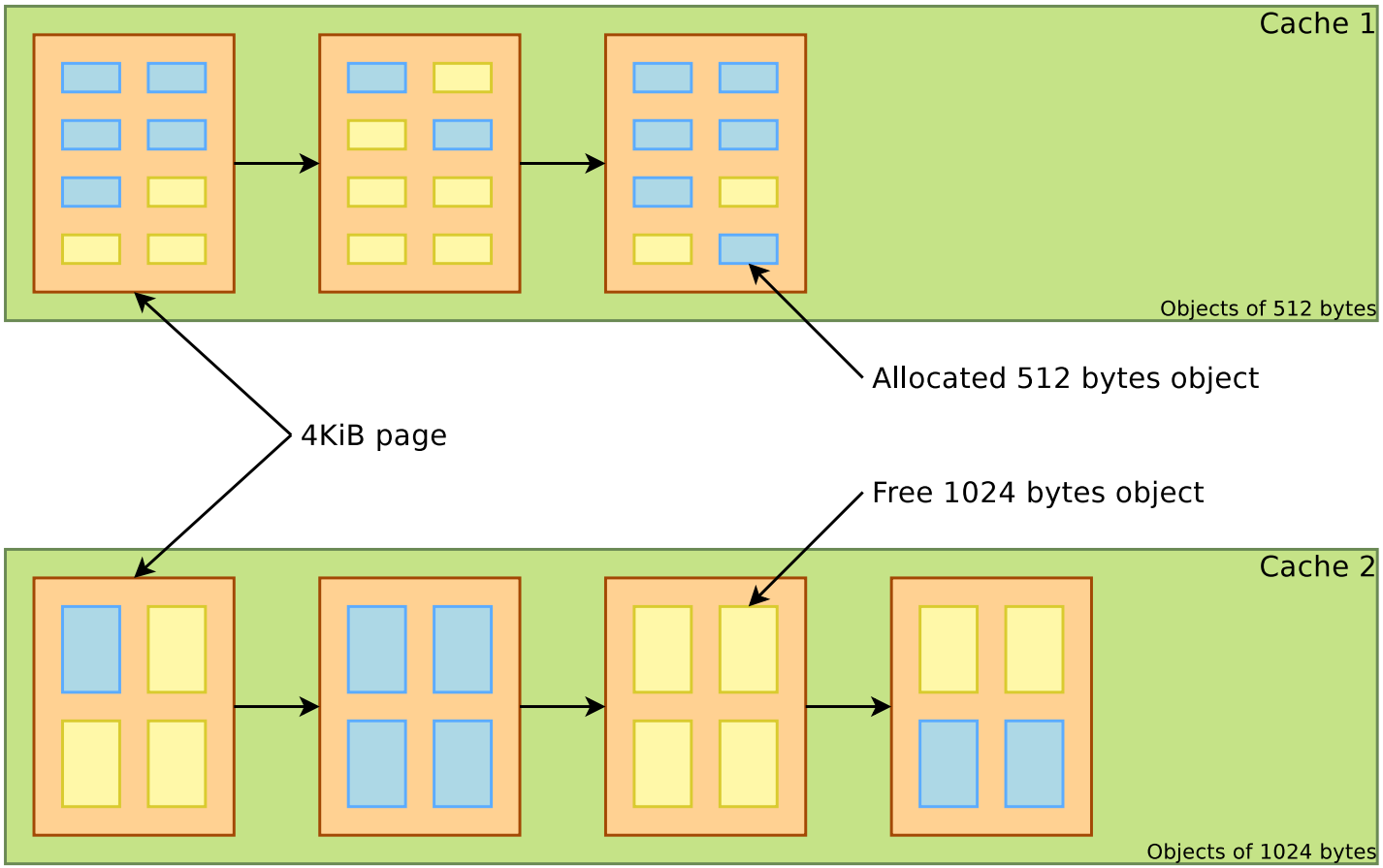



©Anatomy of a Program in Memory. `http://goo.gl/YQHk2B`

# (19) Administración de la memoria en Linux: kernel



# (20) Administración de la memoria en Linux: slab



# (21) Administración de la memoria en Linux: proceso

- Heap
  - No portable: incrementar/decrementar el *program break*

```
#include <unistd.h>
int brk(void *addr);
void *sbrk(intptr_t increment);
```

  - Portable

```
#include <stdlib.h>
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

# (22) Administración de la memoria en Linux: proceso

- Stack
  - Típicamente usada por el compilador (e.g., gcc)

```
#include <alloca.h>
void *alloca(size_t size);
```

- File mapping
  - Mapear un archivo directamente a memoria
  - Usos: segmento de programa, librerías dinámicas
  - Anónimo o no, privado o compartido

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot,
           int flags, int fd, off_t offset);
int munmap(void *addr, size_t length);
```

# (23) Administración de la memoria: en la práctica

- Se usan esquemas que conocen un poco más sobre la distribución de los pedidos y realizan manejos más sofisticados.
- Integración/cooperación lenguaje/compilador/SO
- Cómo asignar memoria de manera rápida y eficiente sigue siendo un problema interesante.
- Ejemplo: Doug Lea allocator: `http://goo.gl/JQqFCD`
  - Política de asignación:
    - Best-fit con Most-Recently-Used en caso de empate
    - Para pedidos de menos de 256 bytes no usa best-fit
  - Concurrencia: No *thread-safe* por default
  - Seguridad:
    - Garantiza inmutabilidad de direcciones menores que la base del heap
    - *Check word*: evita liberaciones no hechas por malloc

# (24) Administración de la memoria: bibliografía adicional


- J.M. Robson. Worst Case Fragmentation of First Fit and Best Fit Storage Allocation Strategies. Computer J. 20, 1977. http://goo.gl/lhfGxG
- P. Wilson, M. Johnstone, M. Neely, D. Boles. Dynamic Storage Allocation: A Survey and Critical Review. http://goo.gl/Uf6T17
- S. Craciunas, C. Kirsch, H. Payer, A. Sokolova, H. Stadler, R. Staudinger: A Compacting R-T Memory Mgmnt. System. http://goo.gl/z9SCwt
- M. K. McKusick, J. Karels. Design of a General Purpose Memory Allocator for the 4.3BSD UNIX Kernel. http://goo.gl/RC8Vtp
- J. Bonwick. The slab allocator: An object-caching kernel memory allocator. USENIX Summer, 87-98, 1994. http://goo.gl/V4Vgur
- Valgrind: an instrumentation framework for building dynamic analysis tools. http://valgrind.org/
- F. Siebert. Eliminating external fragmentation in a non-moving garbage collector for Java. http://goo.gl/Bi8QHB
- D Garbervetsky, C Nakhli, S Yovine, H Zorgati. Program instrumentation and run-time analysis of scoped memory in java. http://goo.gl/t5g0RP