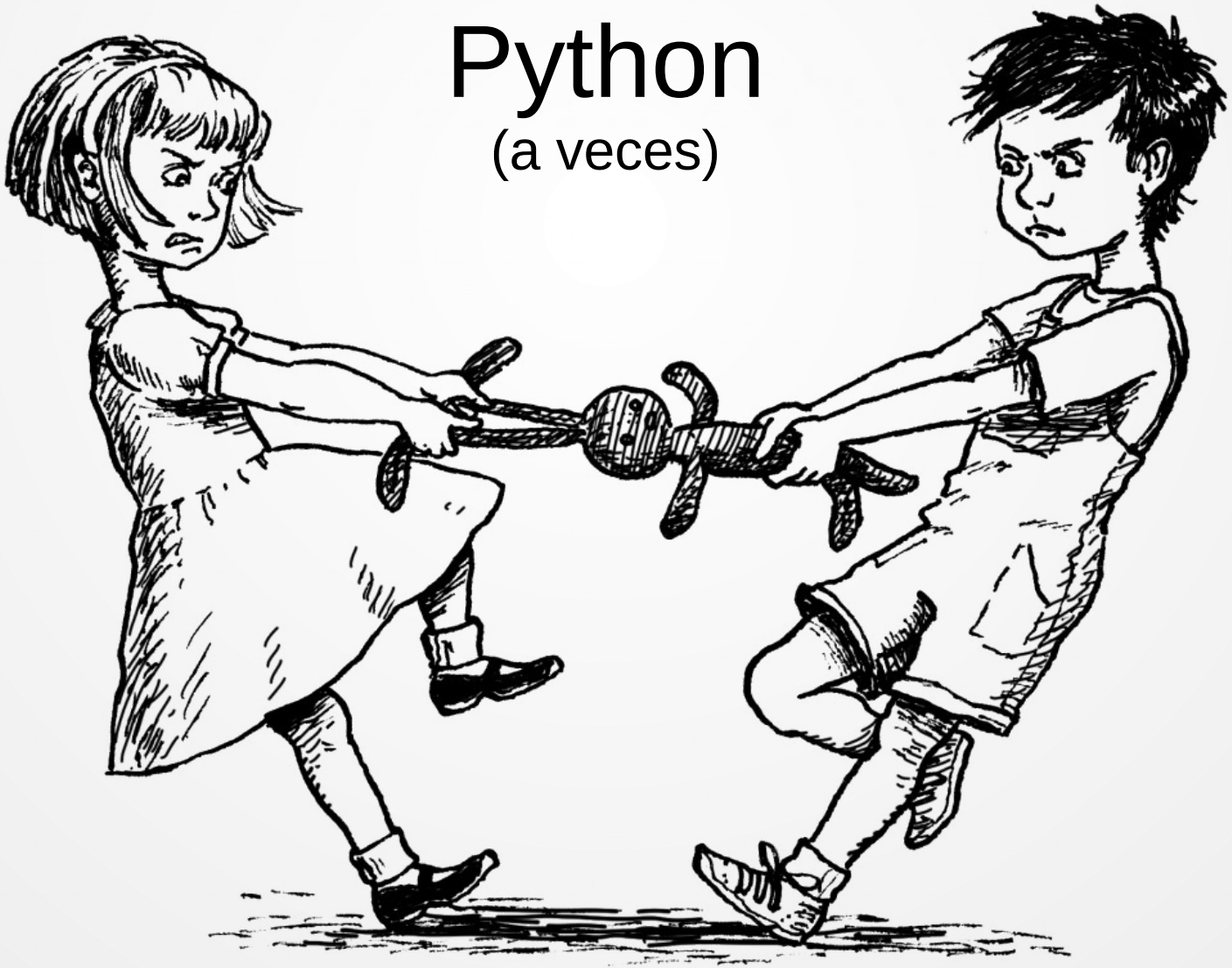
Compartiendo memoria.
Eficientemente.

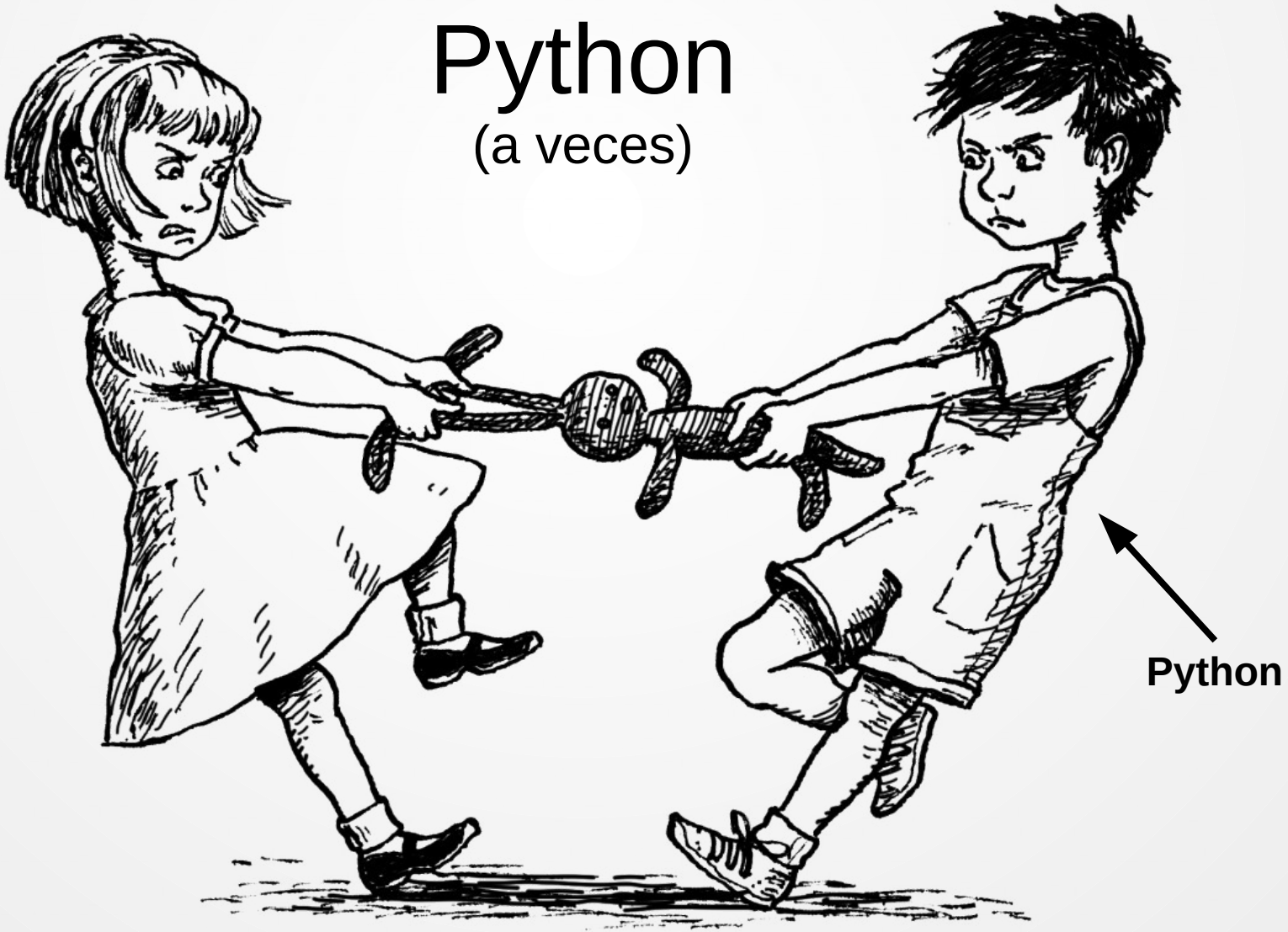
Claudio Daniel Freire

PyConAr Bahía Blanca 2016

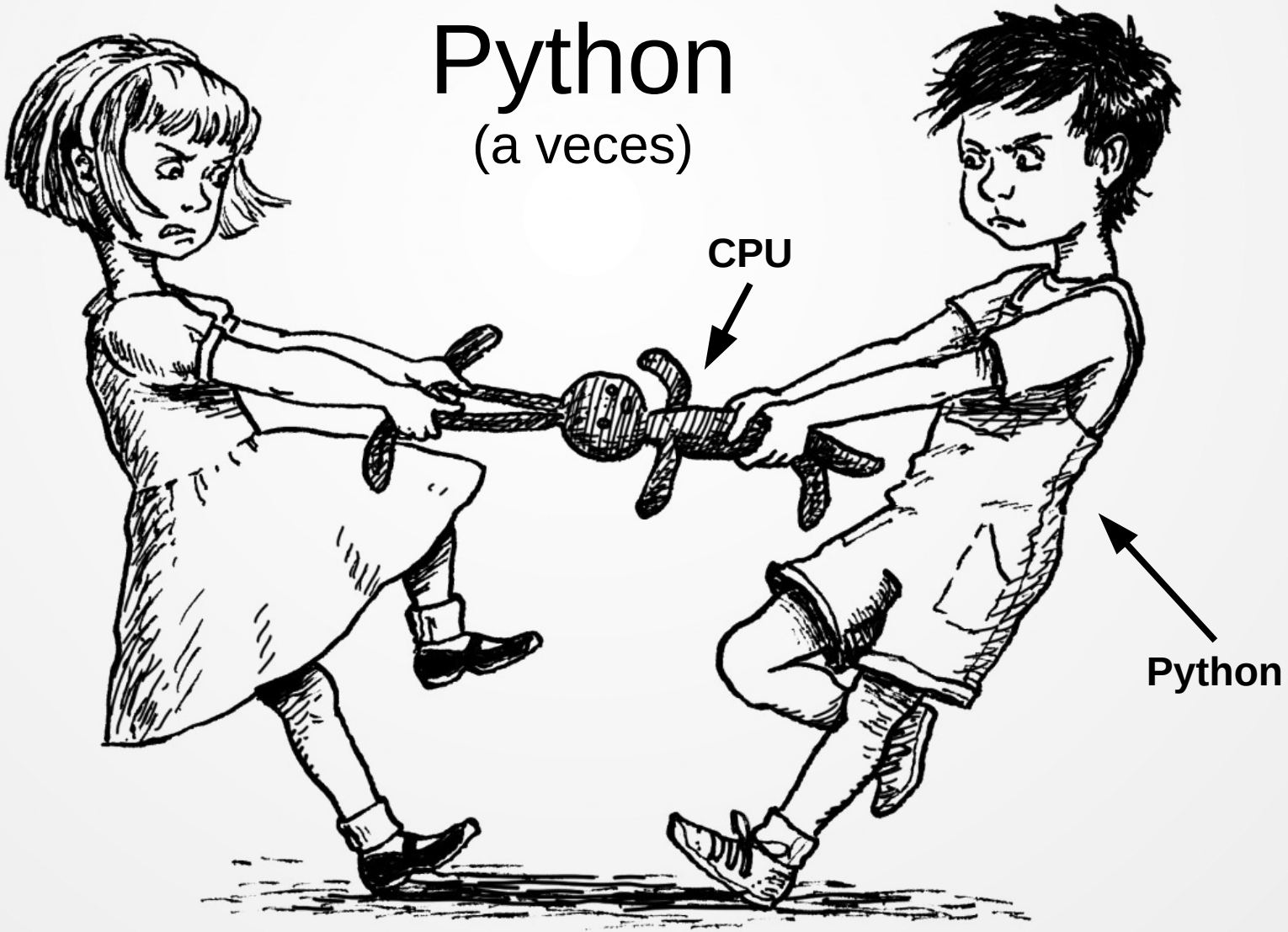
Python
(a veces)



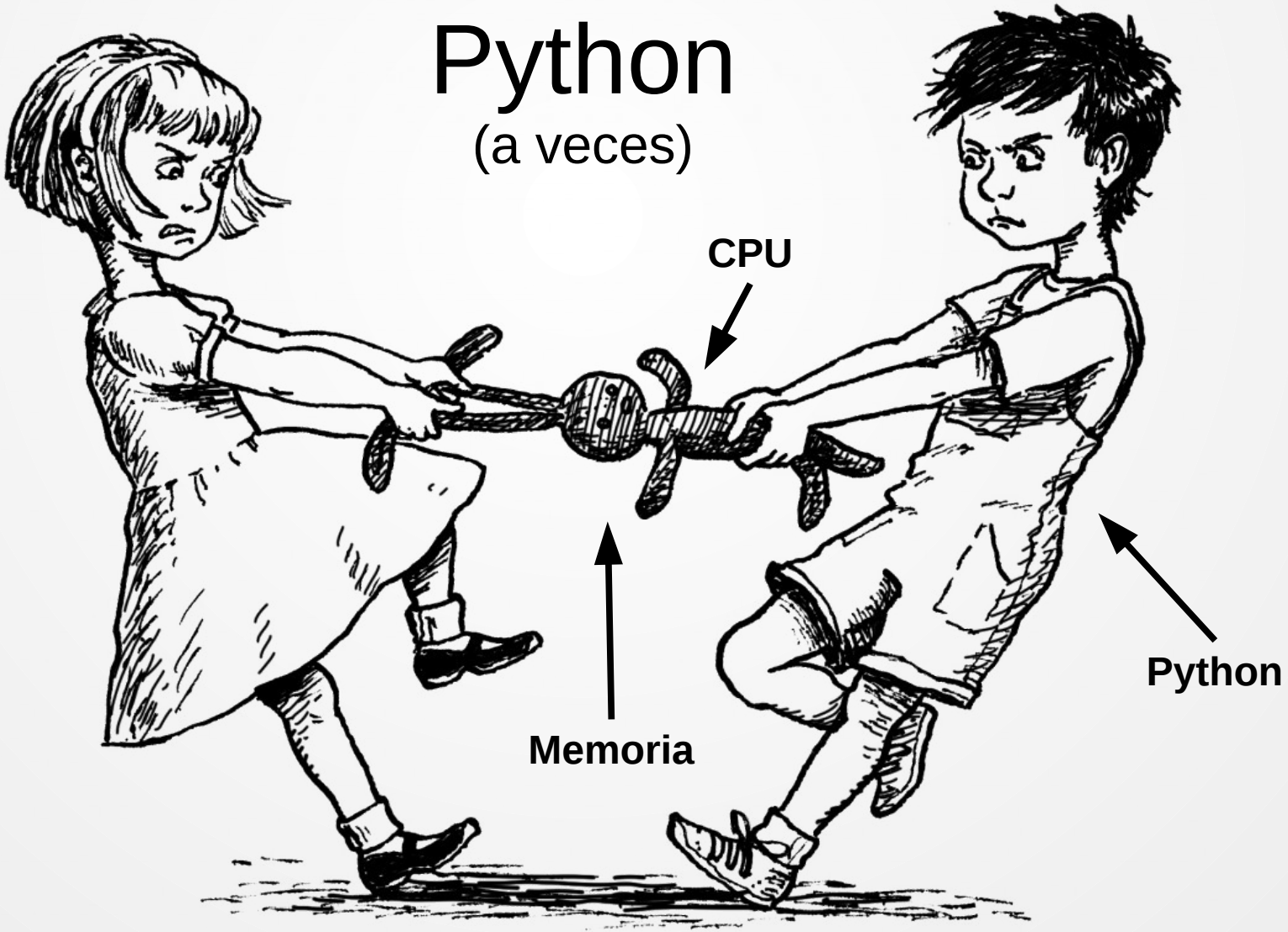
Python
(a veces)



Python
(a veces)



Python
(a veces)



Compartir memoria... ¿para qué?

- Caches demasiado grandes para entrar en memoria...
  - N veces con N procesadores
    - Multiprocessing: datos de entrada
    - Tornado / mod_wsgi: caches de evolución lenta
  - Cuando sólo un pequeño porcentaje es accedido frecuentemente
    - El "set caliente" entra en memoria, pero no la totalidad
    - Pagar un acceso a disco para obtener el valor infrecuente es aceptable

Compartir memoria... ¿para qué?

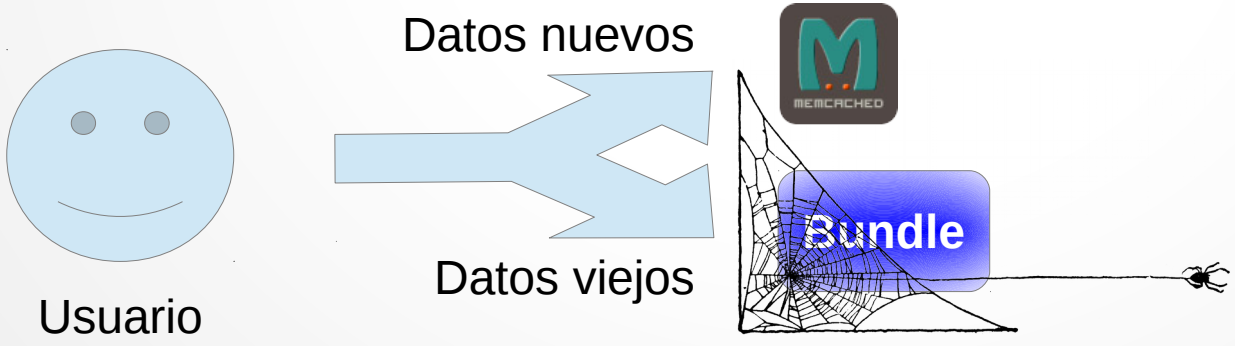
- Cuando el costo de serializar se vuelve prohibitivo
  - Estructuras grandes y complejas en cache:
    se pierde mucho tiempo serializando y deserializando muchos objetos
  - Objetos muy pesados de serializar para cachear:
    - Ej: SQLAlchemy

Compartir memoria... ¿para qué?

- La transición a un "bundle" compartido
  - Separamos el cache en dos capas superpuestas:
    - Una de lenta evolución, sólo lectura
    - Una con la diferencia, de actualización contínua pero infrecuente
  - La mayor parte de los objetos van a estar en la capa estática
  - Lo que falte (o esté viejo) en esa capa, se accede como siempre

El bundle compartido

- Síntesis de lo mejor de ambos mundos
  - Estructura compacta, eficiente y compartida para datos estáticos de base
  - Estructura dinámica, de actualización rápida, para el resto



¿DEMO?

¿cómo?

¿cómo?

Tan simple como:

```
fileobj = open("buf", "r+")
buf = mmap.mmap(
    fileobj.fileno(), 0,
    access = mmap.ACCESS_READ)
```

¿cómo uso eso para meter objetos?

Más complicado

- Hay que definir un esquema
  - Que sea fácilmente manipulable sin serialización
  - Que sea eficiente en espacio y tiempo de acceso
- Hay que armar maquinaria que permita acceder a esos datos
  - Como si fueran objetos
  - Sin copiarlos a la memoria privada de cada proceso

¿cómo uso eso para meter objetos?

Más complicado

- Hay que definir un esquema **–struct—**
  - Que sea fácilmente manipulable sin serialización
  - Que sea eficiente en espacio y tiempo de acceso
- Hay que armar maquinaria que permita acceder a esos datos
  - Como si fueran objetos
  - Sin copiarlos a la memoria privada de cada proceso **–proxies—**

Structs

En C:

```
struct {
    int a;
    float b;
    bool c;
}
```

En Python

```
import struct
struct.pack(
   "if?", 1, 2.0, True)
```

Structs

- ¿Por qué meter C en esto?
  - El código nativo puede acceder nativamente a los structs
  - Son multi-lenguaje (es posible interpretarlos en otros lenguajes)
  - Cython

Proxies

- Clases que saben dónde en un buffer buscar un struct
- Convierten acceso a atributos a lectura del struct:

```
x = Proxy(buf, offset=10)
x.a  # lee el int
x.b  # lee el float
x.c  # lee el bool
```

Proxies

- No requieren serialización
  - Con conocer dónde está el struct basta
- Se pueden fácilmente "reapuntar"
  - Cambiar el offset cambia qué objeto están mostrando eficientemente
  - Evita el costo de crear objetos python todo el tiempo
- Son relativamente transparentes
  - Se ven casi como el objeto original

Proxies – agregando complejidad

```
struct ComplexProxy {
    int value;
    int child_left_offset;
    int child_right_offset;
}
```

```
class ComplexObj:
   def __init__(self,
          l = None, r = None):
      self.value = 3
      self.left = l
      self.right = r
```

Proxies – agregando complejidad

```
class ComplexProxy:
   def __init__(self, buf, pos):
      self.buf = buf
      self.pos = pos

   a = IntProperty(offset=0)
   b = ProxyProperty(ComplexProxy,
       offset=4)
   c = ProxyProperty(ComplexProxy,
       offset=8)
```

```
class IntProperty:
  def __get__(self, obj, kls):
    return unpack("i",
        obj.buf, obj.pos+self.offset)
class ProxyProperty:
  def __get__(self, obj, kls):
    voffset = unpack("i", obj.buf,
        obj.pos+self.offset)
    return ComplexProxy(
      obj.buf, voffset)
```

Proxies – referencias cíclicas – OOPS!

- Se complica empaquetar objetos con referencias cíclicas
  - Hay que reconocerlas al armar el archivo
  - Requieren cuidado, como siempre
- Hay dos formas de manejarlas:
  - Prohibirlas
  - Soportarlas

Proxies – referencias cíclicas – OOPS!

Mapa de identidad

- Mapea id(objeto) → offset
- Al empaquetar un objeto, se actualiza el mapa de identidad
  - Y chequearlo, para detectar objetos ya empaquetados
- Comprime el archivo
  - Unifica referencias repetidas a un mismo objeto
- Rompe ciclos

Proxies – referencias cíclicas – OOPS!

Mapa de identidad

- Cuidados a tener
  - Si se está iterando un generador para construir el "bundle", es posible que hayan dos objetos con el mismo id()
    - Hay que mantener sincronizado el mapa de identidad con los objetos realmente en memoria. Si un objeto es destruido, tiene que limpiarse su entrada del mapa
  - El mapa de identidad puede crecer considerablemente
    - En particular cuando se generan bundles grandes con millones de objetos

pará un cacho

Dijiste sin serializar

Manipulación sin serialización

- Generar un bundle cuesta
  - Similar a serializar, sí

Manipulación sin serialización

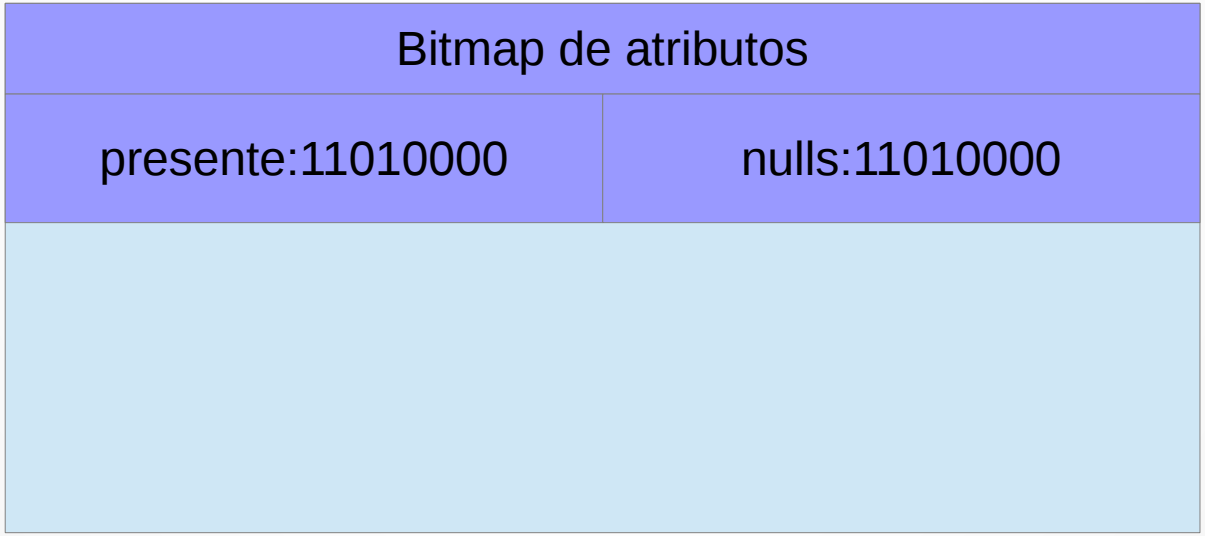
- Generar un bundle cuesta
  - Similar a serializar, sí
- Pero... **usar**, no
  - Abrir
  - Leer
  - Buscar
  - Hasta escribir (con ciertos límites)

Manipulación sin serialización

Estructura de un objeto

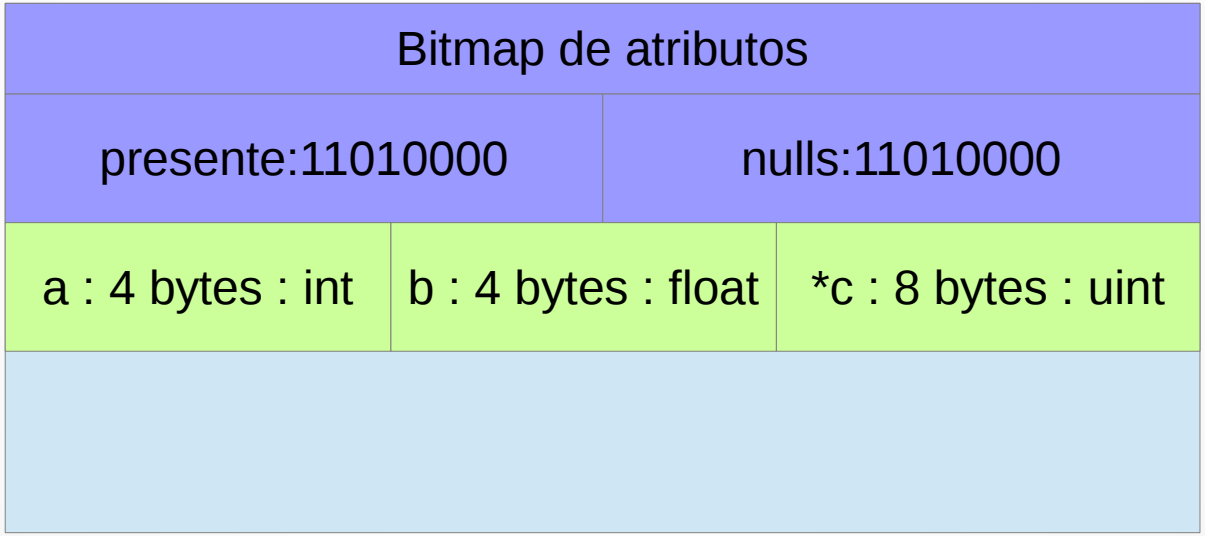
Manipulación sin serialización

Estructura de un objeto



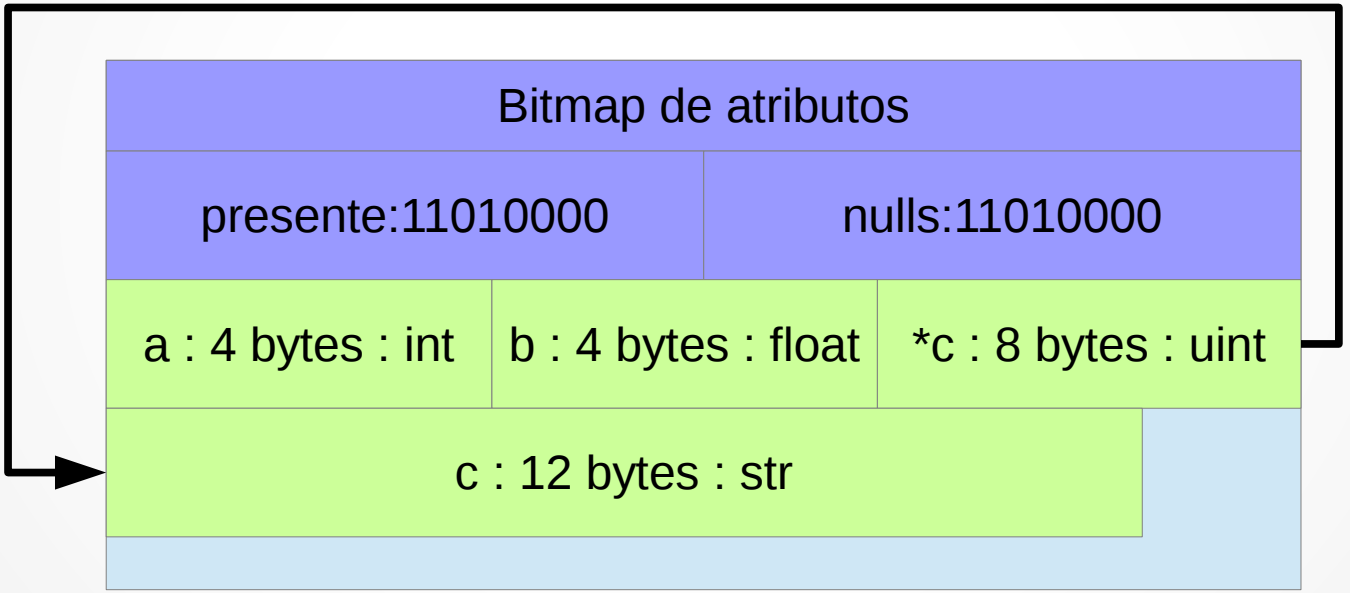
Manipulación sin serialización

Estructura de un objeto



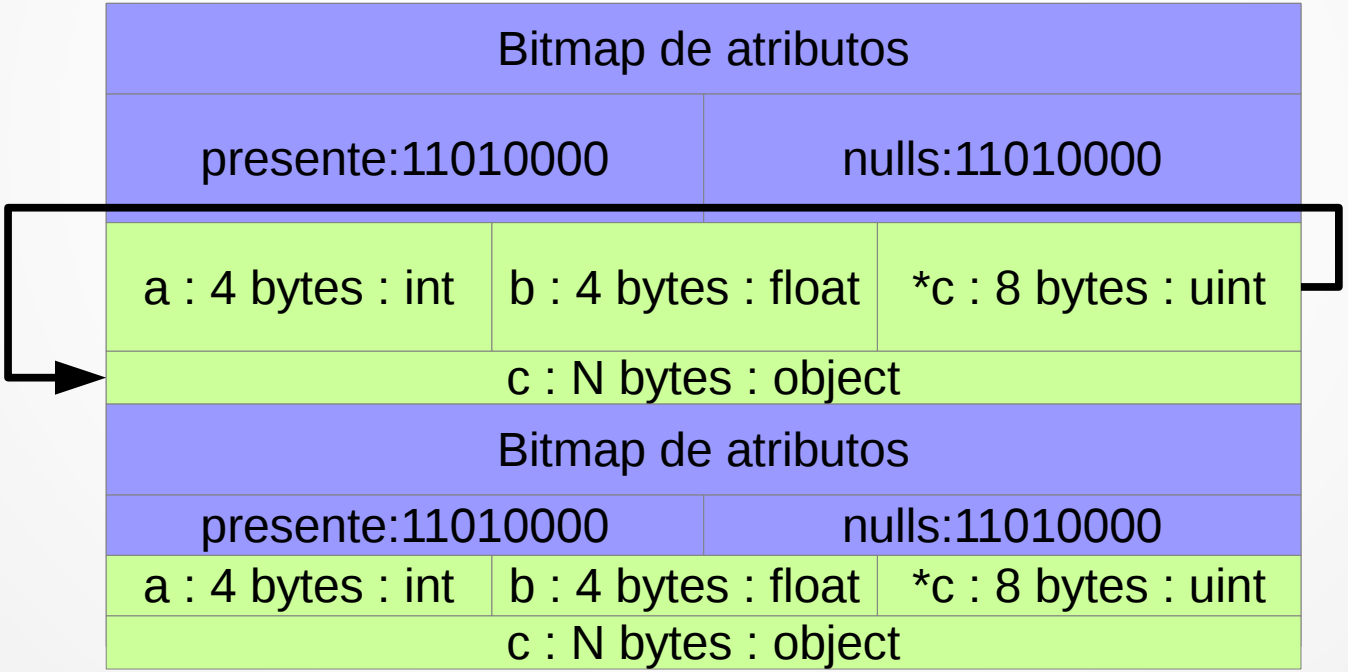
Manipulación sin serialización

Estructura de un objeto



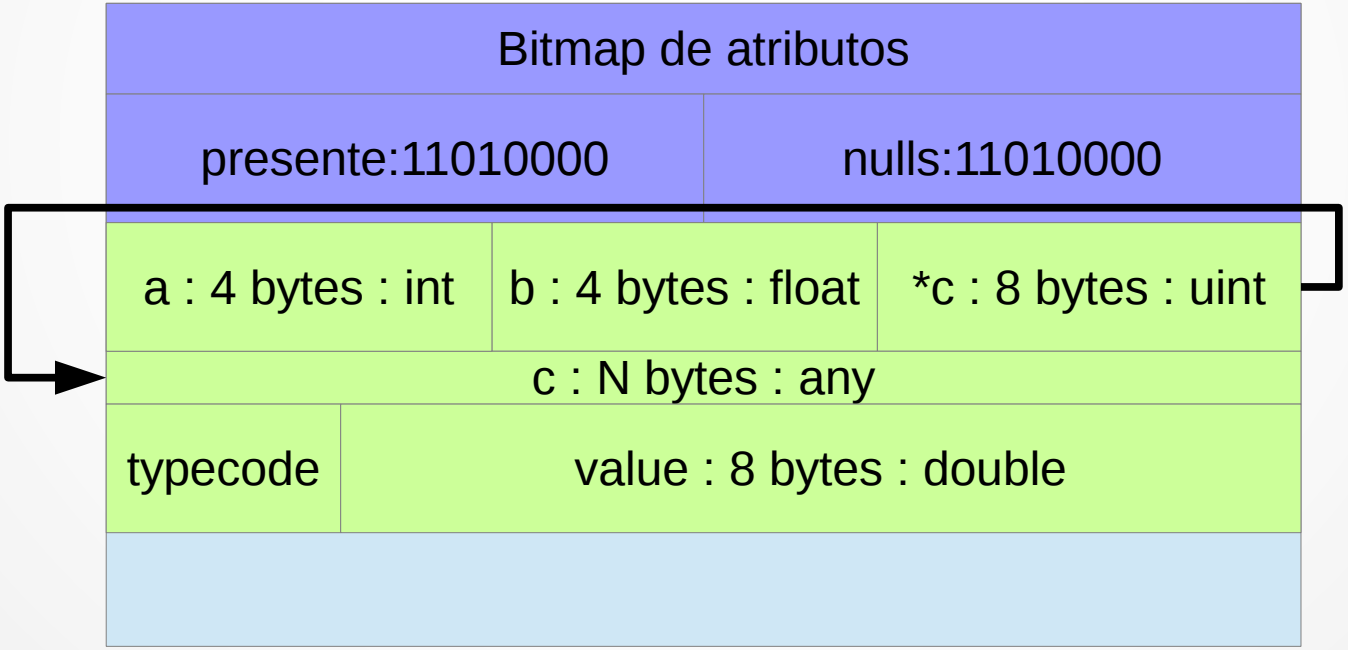
Manipulación sin serialización

Anidando objetos



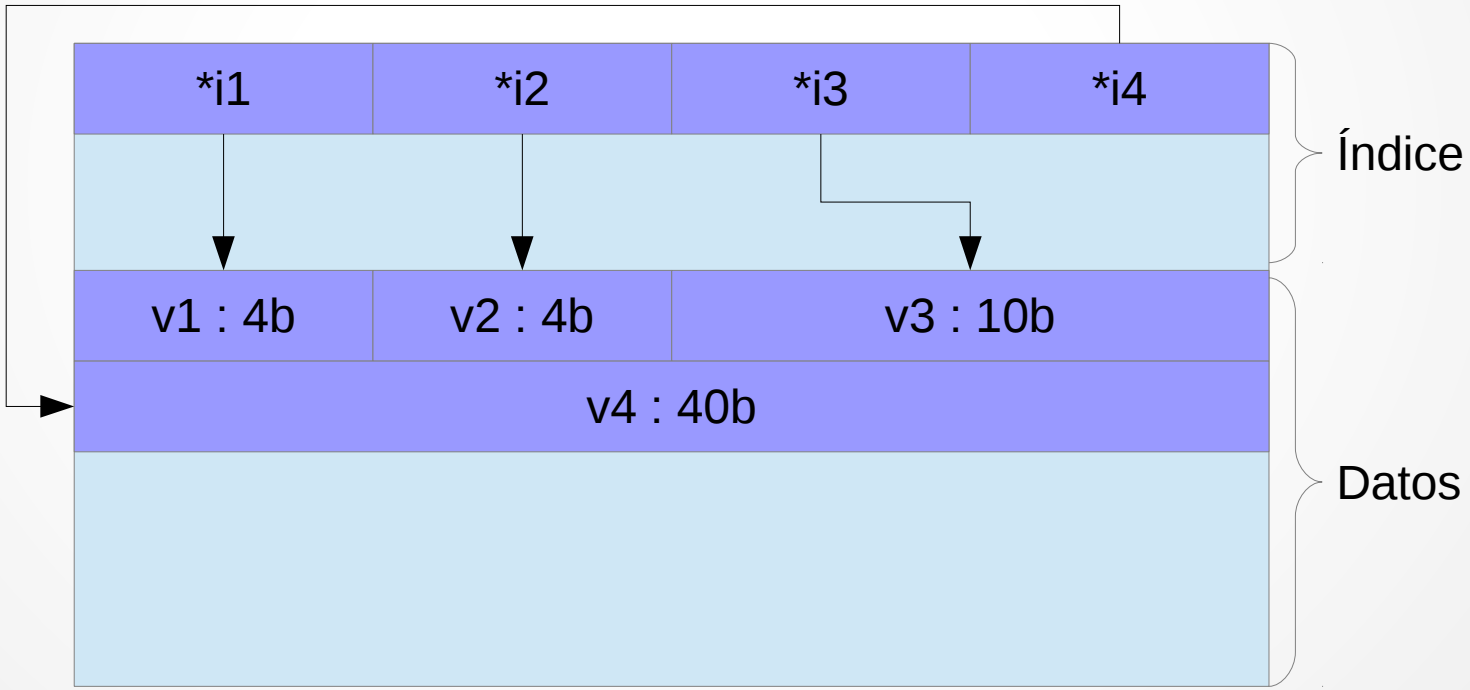
Manipulación sin serialización

Tipos dinámicos

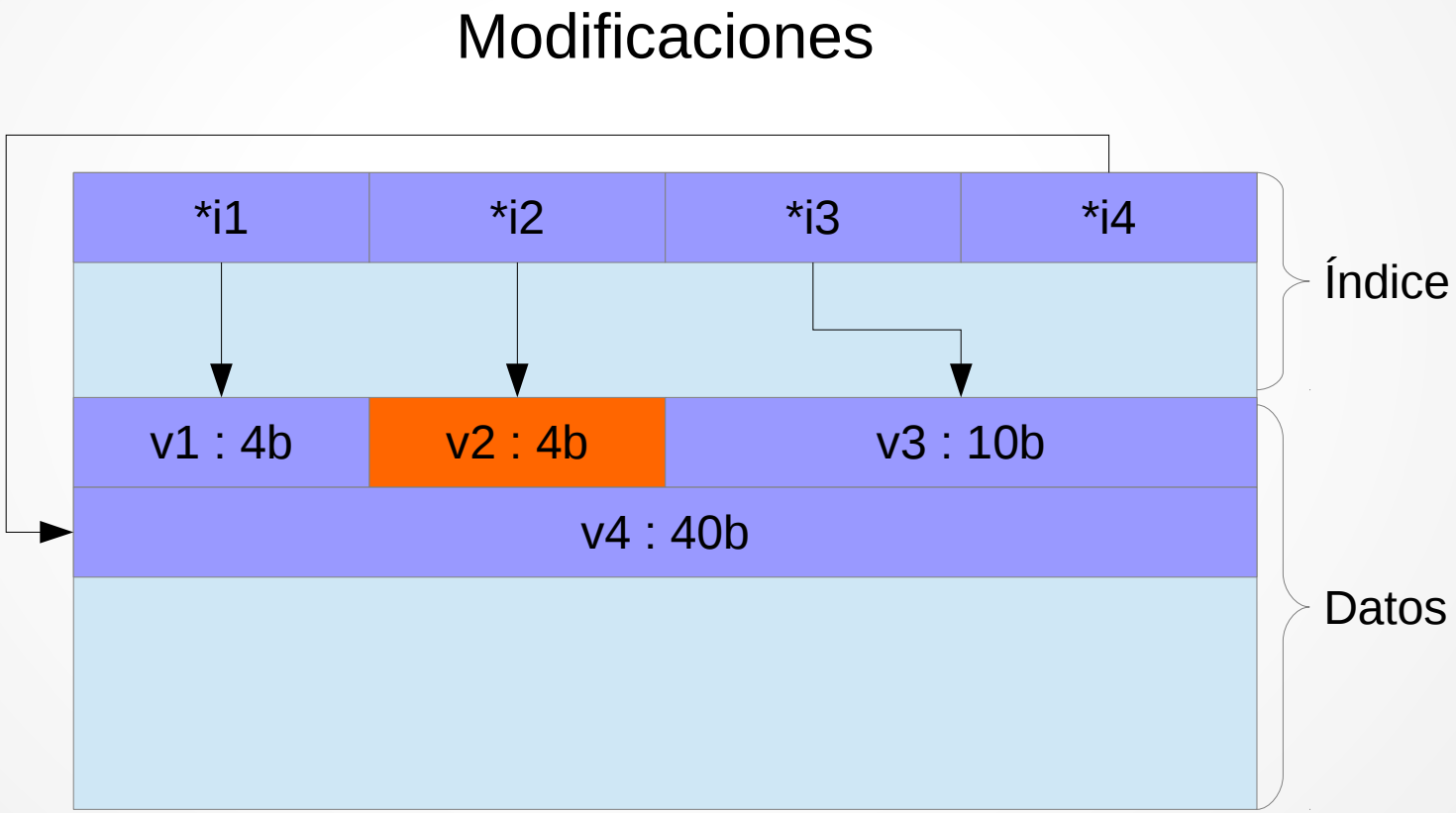


Manipulación sin serialización

Secuencias lineales

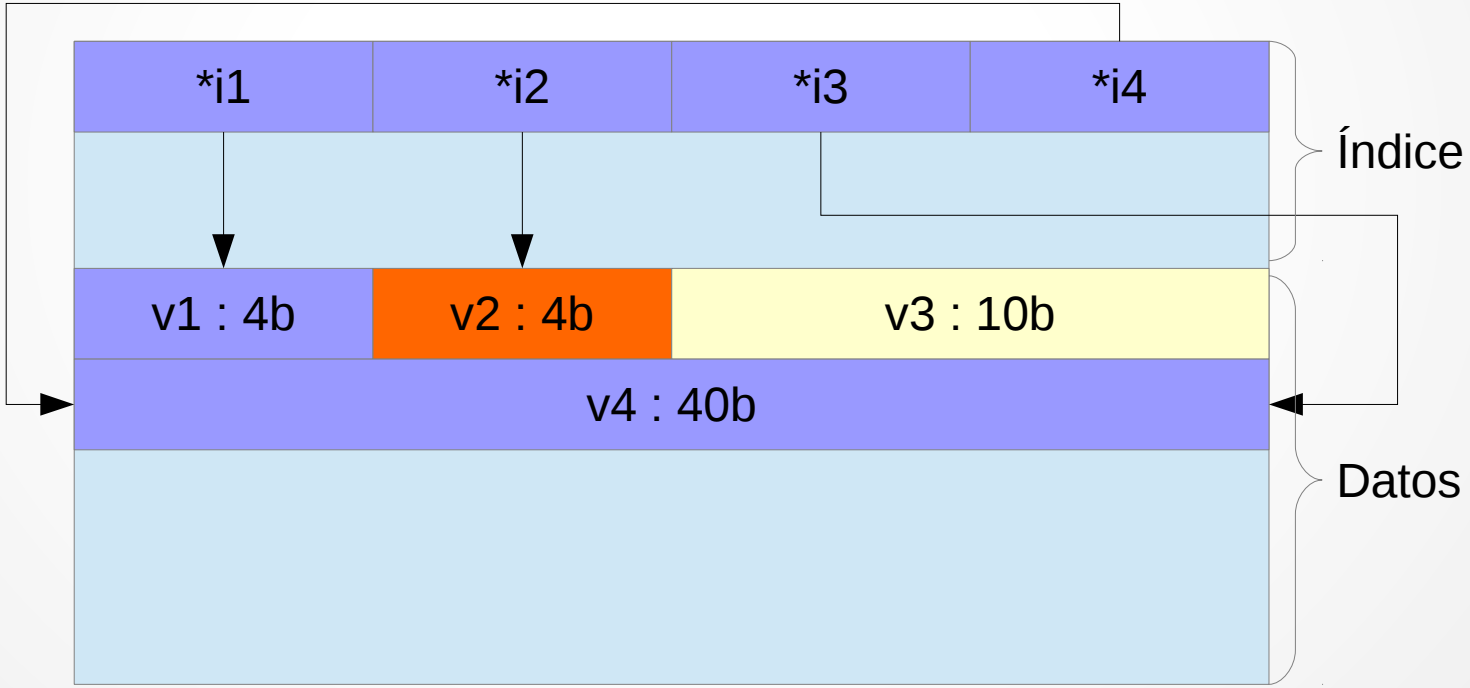


Manipulación sin serialización



Manipulación sin serialización

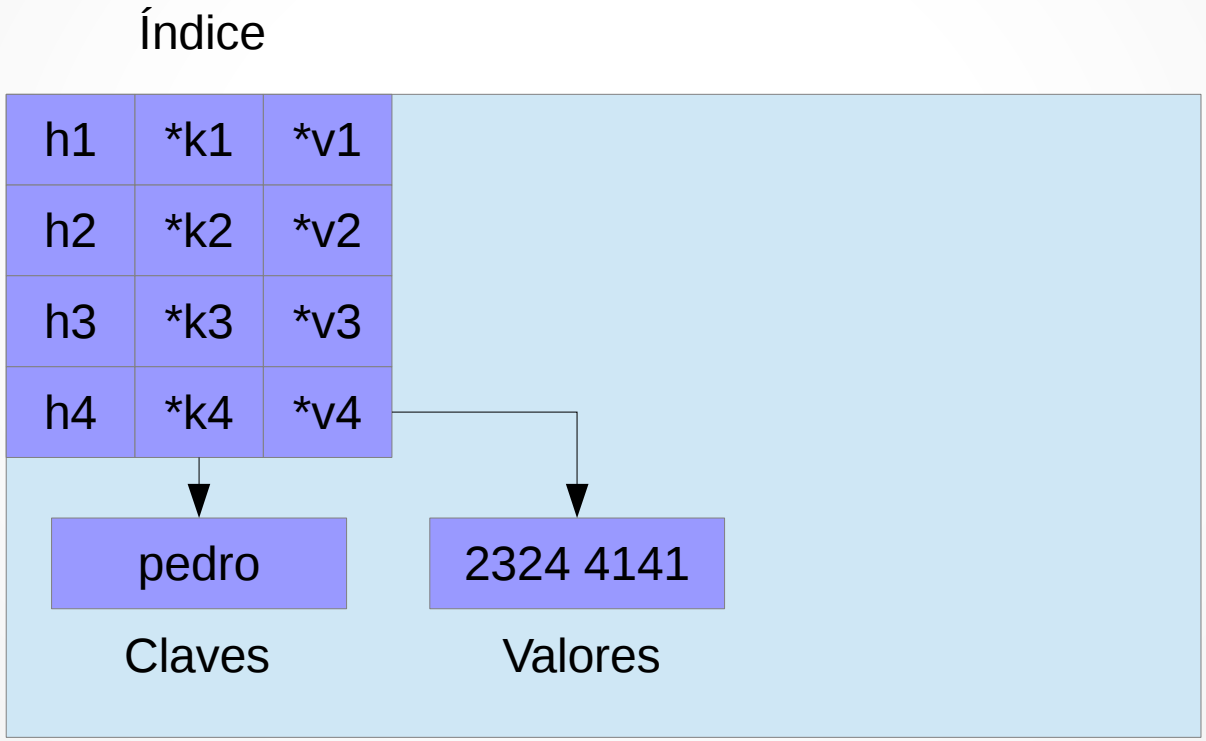
Modificaciones



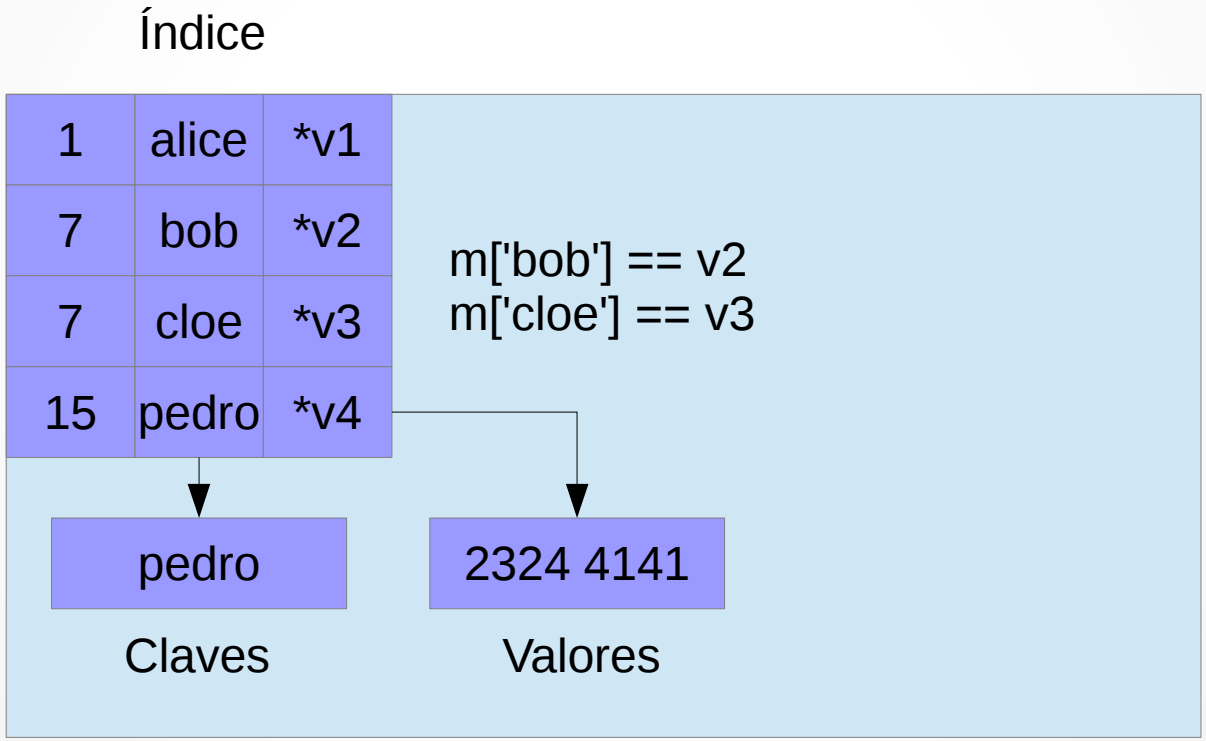
Mapas asociativos

- Tabla de hash compacta:
  - Array ordenado de tuplas <hash, clave, valor>
  - Búsqueda binaria optimizada para distribuciones uniformes
    - Una predicción sabiendo la distribución de las claves (hash)
    - Una iteración de búsqueda exponencial para ajustar la predicción
    - Finalizar con una búsqueda binaria regular
- Tabla de hash aproximada:
  - Tirar la clave, asumir las colisiones como error aceptable
  - Particularmente eficiente para claves textuales largas

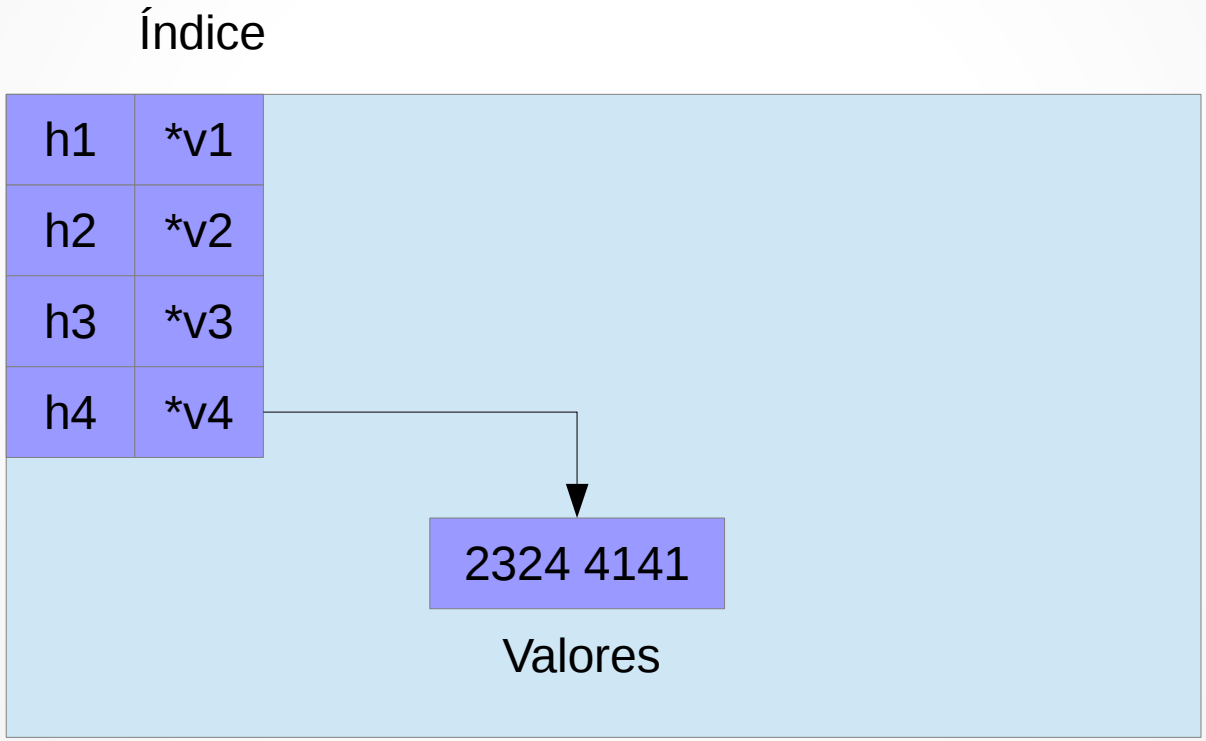
Mapas asociativos



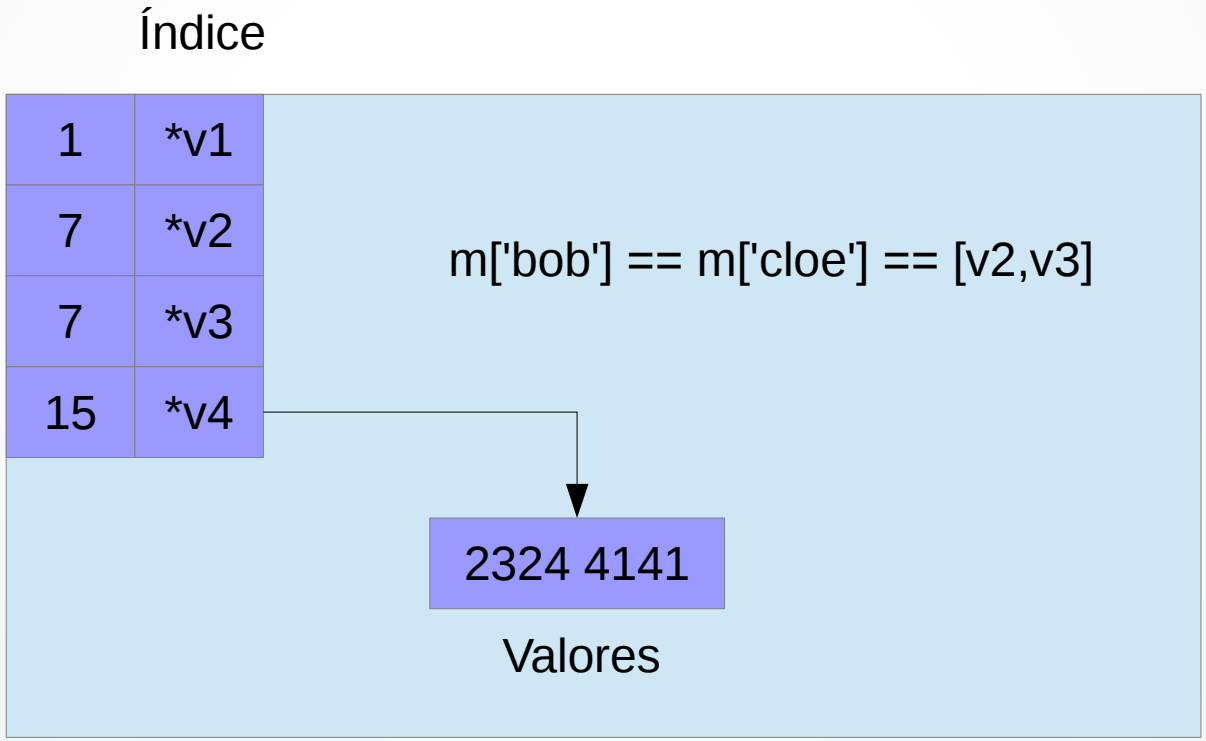
Mapas asociativos



Mapas asociativos aproximados



Mapas asociativos aproximados

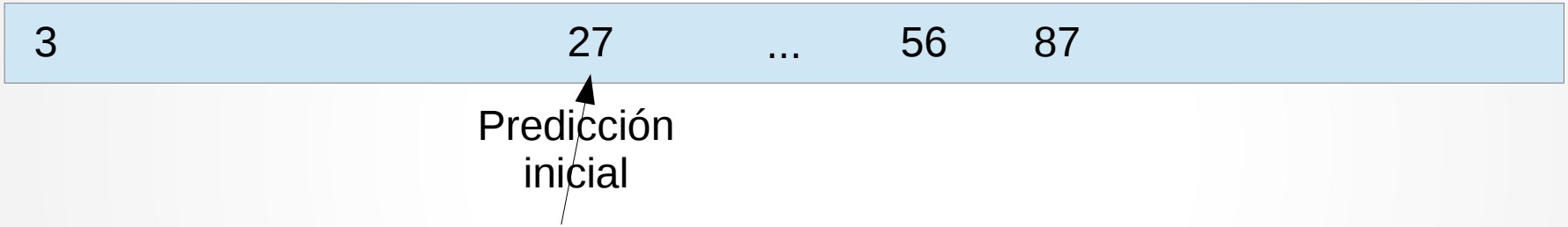


Mapas asociativos aproximados

Búsqueda binaria optimizada

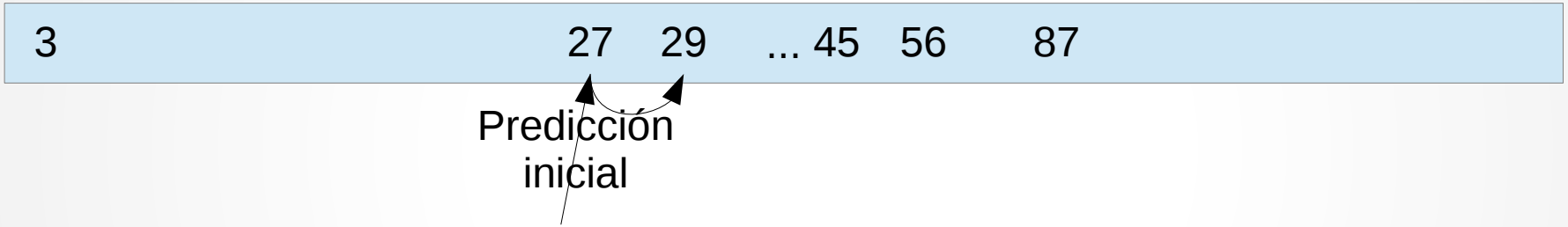
Mapas asociativos aproximados

Búsqueda binaria optimizada



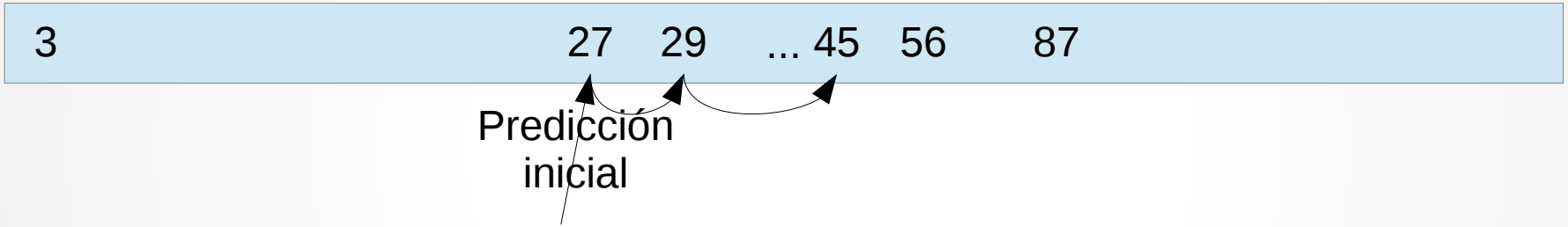
Mapas asociativos aproximados

Búsqueda binaria optimizada



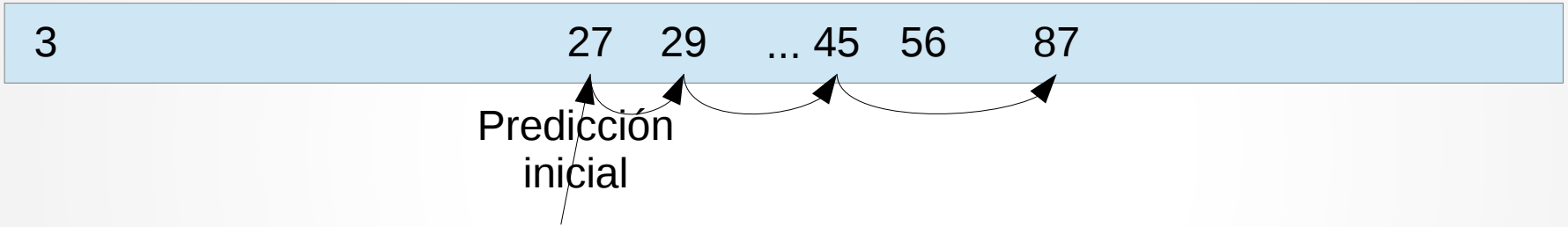
Mapas asociativos aproximados

Búsqueda binaria optimizada



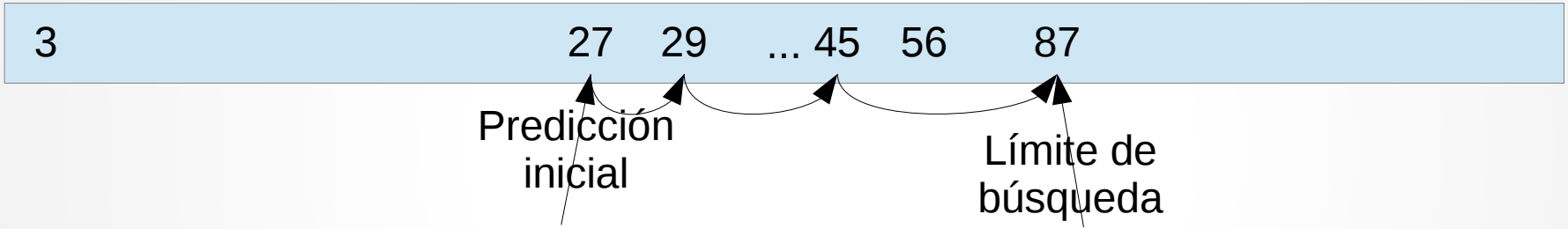
Mapas asociativos aproximados

Búsqueda binaria optimizada



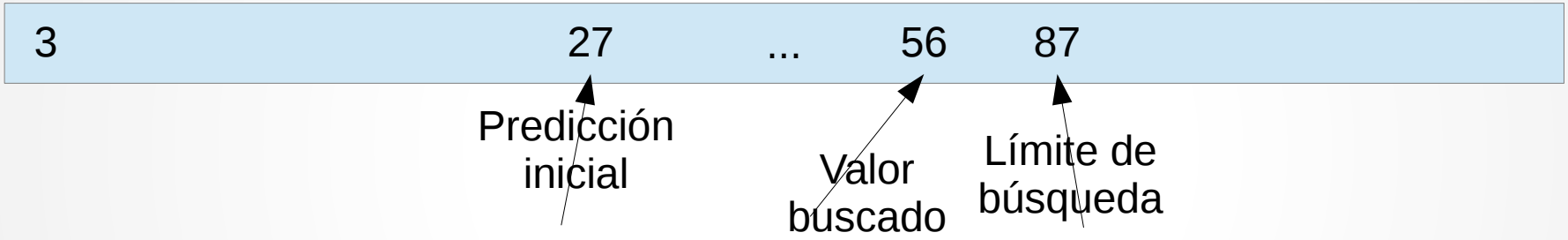
Mapas asociativos aproximados

Búsqueda binaria optimizada



Mapas asociativos aproximados

Búsqueda binaria optimizada



Exprimiendo velocidad

- Performance:
  - Sólo el set de datos "caliente" (más utilizado) necesita entrar en RAM
  - Búsqueda binaria optimizada en *2 e log(e)*
    - *e* siendo el error entre la predicción y la posición efectiva
    - $e < n$
- Tabla de hash aproximada:
  - Tamaño fijo incluso con claves grandes (strings largos)
  - Acceso incluso más eficiente (no hace falta verificar claves)

Exprimiendo velocidad

- Performance:
  - Buen patrón de acceso a disco si no entra en memoria:
    - La búsqueda exponencial resulta en acceso quasi-secuencial
    - Buena localidad de referencia con buenas predicciones
      - O(1) seeks en promedio
    - Posiblidad de precargar el índice
      - mucho más compacto que los valores o las keys

Exprimiendo velocidad

- Magia Cython:
  - En vez de usar struct por todos lados
  - Evita generar objetos python para operaciones temporales
- Reuso de proxies:
  - En vez de crearlos todo el tiempo, reapuntar proxies existentes
  - Transmutación de tipos para cuando cambia la forma del objeto
    - `proxy.__class__ = new_cls`


U MAD
BRO?
quickmeme.com

No lo escribas desde cero

```
pip install sharedbuffers
```