Implementación

¿Cómo se responde una búsqueda a escala Web en una fracción de segundo?

¿Cómo se estructuran, construyen y utilizan índices del orden de petabytes?

Implementación

- La implementación de modelos de IR (y en definitiva de un motor de búsqueda) se centra principalmente en la creación de **índices**
  - Selección de términos (conversión de los documentos en bags of words)
  - Estructura de los índices
  - Construcción y actualización de los índices
- Además hay que diseñar **algoritmos** concretos para él cálculo de las funciones de **ránking** de los modelos IR usando estos índices
  - Acceso a y recorrido eficiente de las estructuras de índices
  - Algún elemento de ránking adicional: búsqueda proximal
  - Otros detalles necesarios en una aplicación: tolerancia a erratas, etc.

Eficacia vs. eficiencia

- **Eficacia:** teoría, modelos, evaluación… de IR → acierto en las respuestas
- **Eficiencia y escalabilidad:** necesarios para la viabilidad un sistema IR real con PBs de datos, miles de consultas por segundo
  - Tiempo de respuesta
  - Consumo de recursos computacionales
- La eficiencia se consigue con el uso de **índices**
  - Elemento central de la arquitectura de un buscador (para texto)

Índices de búsqueda

- Síntesis de la información de los documentos necesaria para el cálculo eficiente y escalable de funciones de ránking
- Inversión de la asociación documento-palabra
  - Término $\rightarrow$ lista de documentos con frecuencia o lista de posiciones
- Medidas de eficiencia de un índice y el sistema IR en conjunto
  - Tiempo de indexado
  - Espacio para indexado: durante la construcción y en almacenamiento final
  - Latencia de consulta: tiempo entre input y output
  - Throughput de consulta: promedio de consultas procesadas por segundo

El proceso de indexación

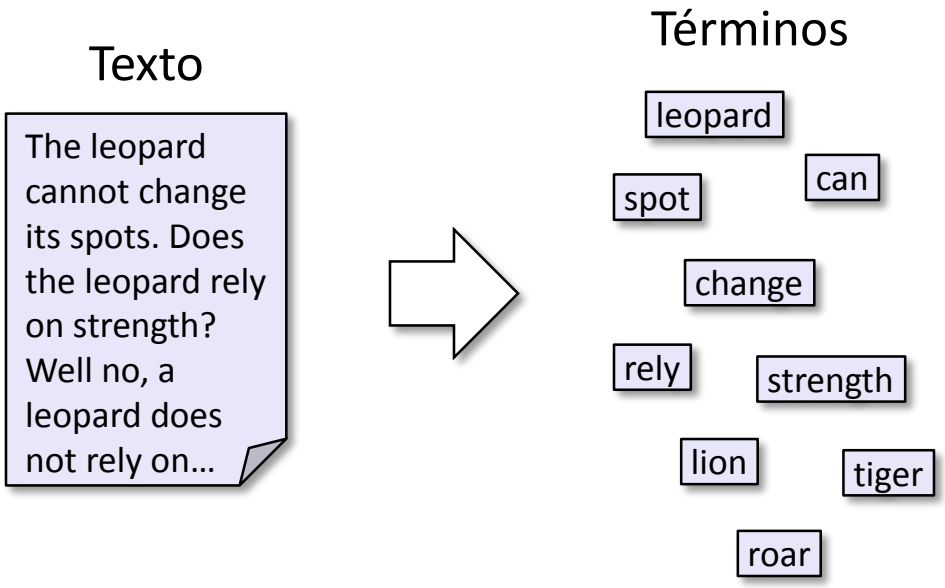
Preprocesamiento de la colección, previo a la respuesta de consultas

1. Tratamiento preliminar del contenido de los documentos
2. Extracción de términos (y formación del vocabulario)
   - Selección de palabras útiles
   - Procesamiento de las palabras $\rightarrow$ términos
3. Construcción de las estructuras de índices
   - Diseño de estructuras $\longrightarrow$ Implementación efectiva de modelos IR
   - Compresión

Pasos preliminares

- Establecimiento de la unidad de documento (granularidad), por ejemplo…
  - Un archivo → un documento
  - Archivos que contienen varios documentos, p.e. buzón de email
  - Segmentos de un archivo: capítulos, secciones, etc.
  - Agrupación de varios archivos, p.e. pdfs de cada diapositiva de una presentación
- Obtención de la secuencia de caracteres
  - Secuencia de bytes → secuencia de caracteres
  - Trivial si la codificación es p.e. ASCII inglés, pero muchos más esquemas (Unicode UTF-8, etc.), estándares regionales y específicos a productos (pdf, PostScript, MS Office, etc.)
  - Decodificación de texto: caracteres html (p.e. &), eliminación de marcas que no se vayan a tener en cuenta (tags html, XML, LaTeX, etc.)
  - Tratamientos de idioma (p.e. orden bidireccional de los caracteres en árabe o hebreo)

Extracción de términos



Extracción de términos – ejemplo

O'Neill Criticizes Europe on Grants

Treasury Secretary Paul O'Neill expressed irritation Wednesday that European countries have refused to go along with a U.S. proposal to boost the amount of direct grants rich nations offer poor countries. The Bush administration is pushing a plan to increase the amount of direct grants the World Bank provides the poorest nations to 50 percent of assistance, reducing use of loans to these nations.

http://www.seobook.com

Normalización

o'neill criticizes europe on grants treasury secretary paul o'neill expressed irritation wednesday that european countries have refused to go along with a us proposal to boost the amount of direct grants rich nations offer poor countries the bush administration is pushing a plan to increase the amount of direct grants the world bank provides the poorest nations to 50 percent of assistance reducing use of loans to these nations

Stopwords

o'neill criticizes europe on **grants treasury secretary paul o'neill expressed irritation** wednesday that **european countries** have **refused** to go along with a **us proposal** to **boost** the amount of **direct grants rich nations** offer **poor countries** the **bush administration** is **pushing** a **plan** to **increase** the **amount** of **direct grants** the **world bank** provides the **poorest nations** to 50 percent of **assistance** reducing use of **loans** to these **nations**

Stopwords

o'neill criticizes europe grants treasury secretary paul o'neill expressed irritation european countries refused us proposal boost direct grants rich nations poor countries bush administration pushing plan increase amount direct grants world bank poorest nations assistance loans nations

Stemming

```
information -> inform

presidency -> presid

presiding -> presid

happiness -> happi

happily -> happi

discouragement -> discourag

battles -> battl
```

Stemming

o'neill criticizes europe grants treasury secretary

paul o'neill expressed irritation european countries

refused us proposal boost direct grants rich nations

poor countries bush administration pushing plan

increase amount direct grants world bank poorest

nations assistance loans nations

Stemming

o'neill criticizes europe grants treasury secretary

paul o'neill expressed irritation european countries

refused us proposal boost direct grants rich nations

poor countries bush administration pushing plan

increase amount direct grants world bank poorest

nations assistance loans nations

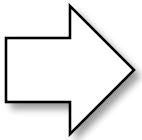
Términos extraídos

administrat	express	poor (2)
amount	grant (2)	propos
assist	increas	push
bank	irritat	refus
boost	loan	rich
bush	nation (3)	secretar
countri (2)	o'neill	treasuri
direct	paul	us
europ	plan	world

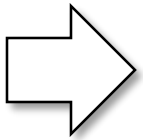
Extracción de términos

Texto

The leopard cannot change its spots. Does the leopard rely on strength? Well no, a leopard does not rely on…



- Tokenización
- Normalización
- Filtrado de stopwords
- Stemming
- Grupos nominales
- Tolerancia a erratas



Términos

leopard
can
spot
change
rely
strength
lion
tiger
roar

Tokenización

- Separación del texto en palabras
  - **Token type**: secuencia de caracteres
  - **Token**: aparición (instancia) de un token type en un documento
  - **Término**: token type procesado (la relación token $\rightarrow$ término es n-1)
- Definición tentativa de token: secuencia de caracteres alfanuméricos de longitud ≥ 3 terminada por un espacio u otro carácter especial
  - Ejemplo: Bigcorp's 2007 bi-annual report showed profits rose 10%
    $\rightarrow$ Bigcorp 2007 annual report showed profits rose
- Esta definición descarta demasiada información en general
  - Palabras de longitud 1-2 pueden ser significativas, solas o en grupos nominales, p.e. C, C#, Formula 1, etc.
  - Guiones, puntos, apóstrofes y otros caracteres especiales son a veces parte de algunas palabras
  - Los números pueden ser importantes
- La casuística se suele tratar con reglas ad-hoc, diccionarios, etc.
  - Puesta a punto mayormente por prueba y error

Normalización

- Forma canónica en la que se eliminan diferencias superficiales
  - Acentos
  - Mayúsculas
  - Variantes del idioma (p.e. British/US English)
  - …
- Puede producir pérdida de información (p.e. Bush vs. bush)
  - A pesar de ello los motores comerciales generalmente aplican sistemáticamente varias reglas de normalización (no todas)

Stopwords

- Partículas funcionales con poco significado propio
  - Artículos, preposiciones, conjunciones, adjetivos demostrativos…
- Palabras muy frecuentes
  - P.e. the → 6%, of → 2%, ambas presentes en ~100% documentos en inglés
- Se eliminan del vocabulario
  - Propósito: eliminar ruido y reducir el tamaño del índice
- Listas estándar
  - P.e. lucene default: a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the, their, then, there, these, they, this, to, was, will, with
  - Lemur incluye una lista de 418 términos
  - Otras de diversas longitudes: 300, 200, 7-12 términos…
  - Otros idiomas
- Inconveniente: se pierden grupos nominales (~~the~~ ~~who~~), títulos/frases de obras (~~to~~ ~~be~~ ~~or~~ ~~not~~ ~~to~~ ~~be~~ ~~that~~ ~~is~~ ~~the~~ question; ~~do~~ ~~as~~ ~~you~~ ~~would~~ ~~be~~ ~~done~~ ~~by~~), etc.
- Los motores Web tienden a no eliminar stopwords y utilizar en su lugar técnicas para el tratamiento de términos de altas frecuencias
- También se hacen ajustes ad-hoc, por campos (“click here”), etc.

Hablando de palabras y frecuencias…

¿Qué naturaleza tiene el espacio de las palabras?

¿Cuántos términos maneja un buscador?

¿Cómo se distribuyen las palabras?

¿Cuántas palabras tiene un idioma?

- La pregunta no tiene sentido ☺
  - Neologismos, argot, desaparición de palabras…
  - Más las inflexiones, más los tecnicismos, más los nombres propios! Más…
  - Los idiomas están vivos, los límites de un idioma son indefinidos
- Algunas cifras
  - DRAE ~100.000 entradas, la mitad arcaicas
  - TLF ~100.000 entradas
  - OED ~600.000 entradas – confluencia de germánico + norse + latín (2/3!) + francés + apertura contemporánea a otros idiomas + extensión geográfica del uso

    Estudio Harvard/Google 2010 – ~1M palabras en inglés, expansión a 8.500 palabras al año
  - × más de 6.000 idiomas en el mundo (top 18 ≥ 50% población mundial)
- Vocabulario funcional (datos inglés)
  - Idioma nativo: 18 meses ~150 palabras, 5-6 años ~5.000, adulto ~20.000
  - Como 2ª lengua: p.e. ~5.000 palabras dan comprensión del 95%
  - Tomar las cifras con un grano de sal: ¿qué entendemos aquí por palabra?
- Vocabulario Web: muchos millones…

Propiedades estadísticas de los términos en IR: palabras más frecuentes

1	the	21	this	41	so	61	people	81	back
2	be	22	but	42	up	62	into	82	after
3	to	23	his	43	out	63	year	83	use
4	of	24	by	44	if	64	your	84	two
5	and	25	from	45	about	65	good	85	how
6	a	26	they	46	who	66	some	86	our
7	in	27	we	47	get	67	could	87	work
8	that	28	say	48	which	68	them	88	first
9	have	29	her				see	89	well
10	I	30	she				other	90	way
11	it	31	or				than	91	even
12	for	32	an	52	make	72	then	92	new
13	not	33	will	53	can	73	now	93	want
14	on	34	my	54	like	74	look	94	because
15	with	35	one	55	time	75	only	95	any
16	he	36	all	56	no	76	come	96	these
17	as	37	would	57	just	77	its	97	give
18	you	38	there	58	him	78	over	98	day
19	do	39	their	59	know	79	think	99	most
20	at	40	what	60	take	80	also	100	us
								101	. . .

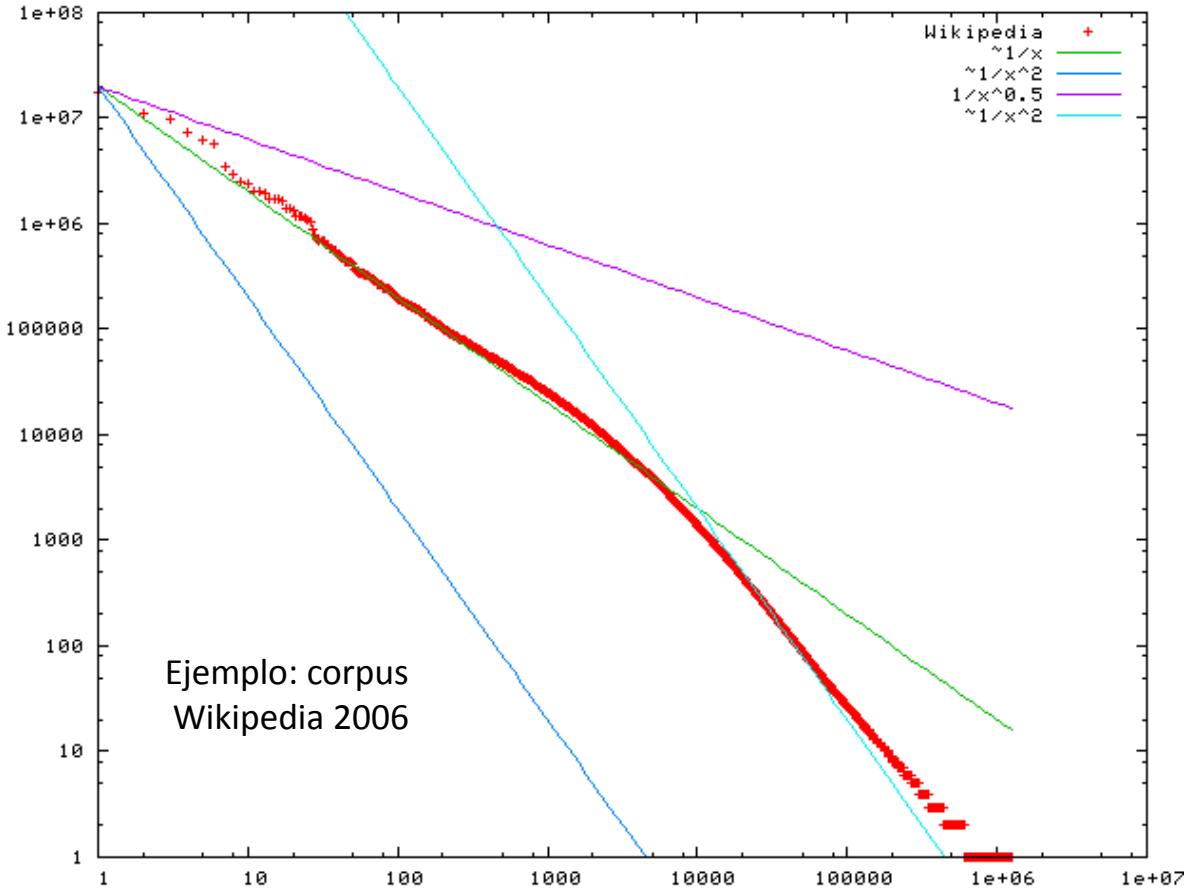
50% del Oxford English Corpus

Frecuencia de las palabras

Ley de Zipf: $cf(t_k) = Ck^{-\alpha}, \quad C = \frac{\sum_t cf(t)}{\sum_k k^{-\alpha}}$

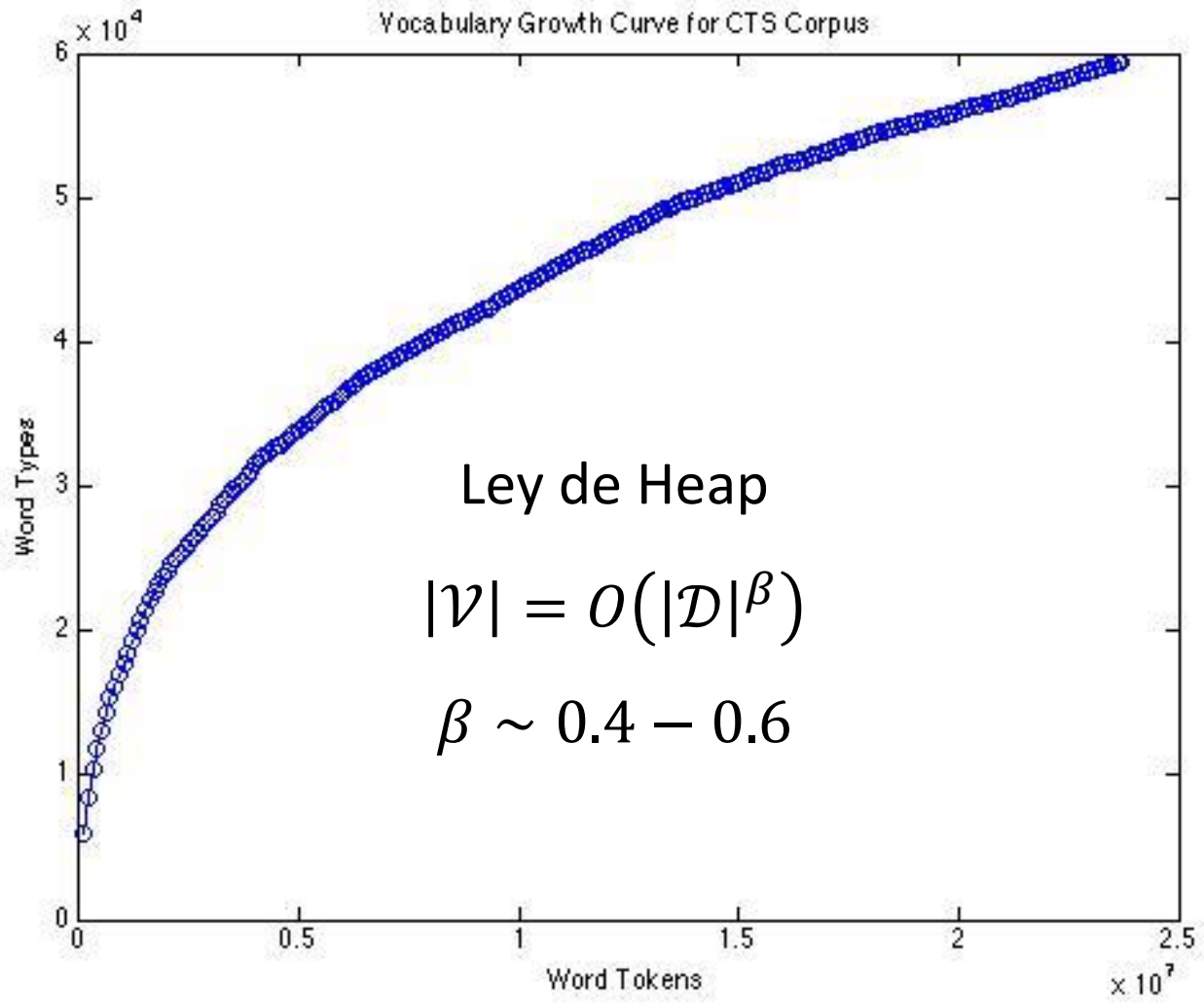
En inglés, $\alpha \sim 1$

George K. Zipf
1902-1950



http://en.wikipedia.org/wiki/Zipf's_law

Crecimiento del vocabulario



Ley de Heap

$$|\mathcal{V}| = O\left(|\mathcal{D}|^{\beta}\right)$$

$$\beta \sim 0.4 - 0.6$$

Stemming

- Reducir inflexiones y derivaciones a un lema común: la idea es quedarnos con significados e ignorar diferencias irrelevantes (ruidosas)
  - Número, género
  - Conjugación de verbos
  - Palabras derivadas: democracia, democrático, democratización…
  - Ejemplo: am, are, is, was → be

    car, cars, car's, cars' → car

    the boy's cars are different colors → the boy car be differ color

Principalmente se ciñe a sufijos (y algún verbo irregular común)

- Existen algoritmos estándar, p.e. Porter's stemmer
  - Reglas heurísticas para la eliminación de sufijos (plurales, etc.)
- Stemmer vs. lematizador
  - Stemming: truncado de palabras, más simple, no siempre correcto (p.e. is → i)
  - Lematización: reducción a lema mediante análisis morfológico

    Utilizan un diccionario y reglas morfológicas

    Necesarios para algo tan básico como conjugaciones irregulares (am, are, is, was → be)

    Limitación: palabras nuevas, evolución del lenguaje

Stemming – ejemplos

Original

Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer

such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer

such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer

such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Porter's stemmer

Martin F. Porter

- Propuesto en los 70, uno de los más populares (para inglés)
- Eliminación de sufijos
- Varios pasos que constan de un conjunto de reglas
- En cada paso se da preferencia a la regla que elimina el sufijo más largo
- Ejemplo paso 1 (plurales, participio, gerundio)
  - sses → ss — stresses → stress
  - consonante + s → consonante — gaps → gap, gas → gas
  - ied, ies → i, ie — ties →tie, cries → cri
  - vocal + cons + eed, eedly → vocal + cons + ee — agreed → agree, feed → feed
  - vocal + ed, edly, ing, ingly → vocal (+ e si termina en at, bl ó iz ó la palabra es corta, eliminar última letra si termina en doble letra que no sea ll, ss, zz)

Porter's stemmer – ejemplos de errores

Falsos positivos	*Falsos negativos*
university ≡ universe	cylindrical ≢ cylinder
paste ≡ past	matrices ≢ matrix
policy ≡ police	urgency ≢ urgent
negligible ≡ negligent	creation ≢ create
generalization ≡ generic	analyses ≢ analysis
numerical ≡ numerous	usefully ≢ useful
specialized ≡ special	noisy ≢ noise
executive ≡ execute	triangular ≢ triangle
…	…

Mejoras más recientes, ver http://snowball.tartarus.org

Stemming

- Mejora recall, empeora precisión
- En algunos idiomas la mejora es algo mayor que en inglés
- En general no se puede hablar de una solución infalible
  - Ni en coherencia lingüística, ni en efectividad práctica para la búsqueda
- Los buscadores Web tienden a prescindir de stemming
- Pero puede ser útil en muchos casos y aplicaciones

Procesamiento del texto de las consultas

- La extracción de términos debe ser **coherente** con la que se haya hecho en los documentos
  - Mismo stemming, normalización, stopwords, etc.
- Además se suelen realizar operaciones adicionales
  - Tolerancia a erratas
  - Grupos nominales
  - Búsqueda literal } Búsqueda posicional
  - Sinónimos } Expansión de la consulta
  - Relevance feedback
  - Polisemia } Diversificación de resultados
  - Sugerencia de palabras de consulta y consultas relacionadas

Tolerancia a erratas

- Detección de erratas
  - Términos que no están en el diccionario
  - Términos de ínfima frecuencia con *alternativa* mucho más frecuente
- Selección de alternativas
  - Basada en medidas de *distancia* entre palabras
  - Y frecuencia de los términos, i.e. priorizar la alternativa más frecuente
  - Frecuente en la colección o, mejor, en el log de búsqueda
- Selección de alternativas en contexto
  - Combinar las alternativas de las palabras de la consulta

    P.e. “fly form london” $\rightarrow$ “fly fore london” “fly from london” …
  - Tener en cuenta la frecuencia (nº resultados o log) de cada combinación
- Subsanación de erratas
  - Directamente, y/o sugiriendo al usuario la posible corrección

Distancia entre palabras

- Distancia Levenshtein (edit distance)
  - Nº mínimo de cambios (inserción, eliminación, sustitución de un caracter) en una palabra para obtener la otra, p.e. $d(\text{cat}, \text{dog}) = 3$
  - También se puede ponderar de forma distinta cada cambio, p.e. una sustitución de caracteres próximos en el teclado tendría menor peso
- Otras medidas
  - Nº de k-gramas en común
  - Distancia fonética: diccionario de asociación de palabras fonéticamente similares (algoritmos soundex)
- Alto coste de las comparaciones
  - $O(l_1 \cdot l_2)$ entre cada palabra de la consulta y el resto del vocabulario!
  - Diversas heurísticas, p.e. restringir las alternativas candidatas a las palabras que empiezan con la misma letra (y otras más sofisticadas)

Frases y grupos nominales

- Son palabras que querríamos tratar como un solo término
  - Muy comunes en las búsquedas
  - Nombre + apellido de personas
  - Nombres compuestos de instituciones, empresas, clubs deportivos, ciudades, países, fiestas, obras, personajes, enfermedades, conceptos…
  - Búsquedas literales (phrases), p.e. título de una obra, pasaje en un texto, expresiones comunes… o lo que interese al usuario
- Soluciones

Extracción del vocabulario

  - Utilizar un diccionario de grupos nominales e incluirlos en el vocabulario
  - Crear un índice de bigramas o k-gramas para palabras consecutivas en el corpus (coste: índice adicional de diccionario $\ll$ simple^k; limitación: sólo k palabras)
  - Utilizar análisis morfosintáctico para seleccionar grupos nominales tipo NX*N, p.e. Quijote (N) de (X) la (X) Mancha (N)
  - Utilizar **búsqueda posicional**: en la intersección de postings, añadir como condición que las posiciones sean consecutivas

Índices de búsqueda

- Índices invertidos
  - Denominación redundante, se utiliza por razones históricas
- Estructura: términos ordenados (**diccionario**) + lista de IDs de documentos (**postings**)
  - Los IDs se suelen generar automáticamente al indexar
  - Acompañados por diversa información en los postings, típicamente se distinguen tres casos: frecuencia (p.e. VSM), posiciones (búsqueda posicional), ∅ (p.e. booleano)
- Estructura del diccionario
  - Índice simple (longitud fija o variable) o árbol B, orden lexicográfico
  - Tabla hash, acceso más rápido, ocupa más espacio
- Ordenación de postings
  - Por ID de documento (p.e. para modelo booleano)
  - Por frecuencia del término en el documento (orientado a ránking/poda)
- Se almacenan además otros valores específicos a modelos IR
  - Módulo de documentos para VSM, PageRank, estadísticas para otros modelos, etc.
- Adicionalmente, document map
  - Nombre, snippet y otros datos de los documentos (fecha creación/actualización, etc.)
  - Índice forward documento → término, p.e. para relevance feedback

Ejemplo

$d_0 =$ “Shipment of gold damaged in a fire”

$d_1 =$ “Delivery of silver arrived in a silver truck”

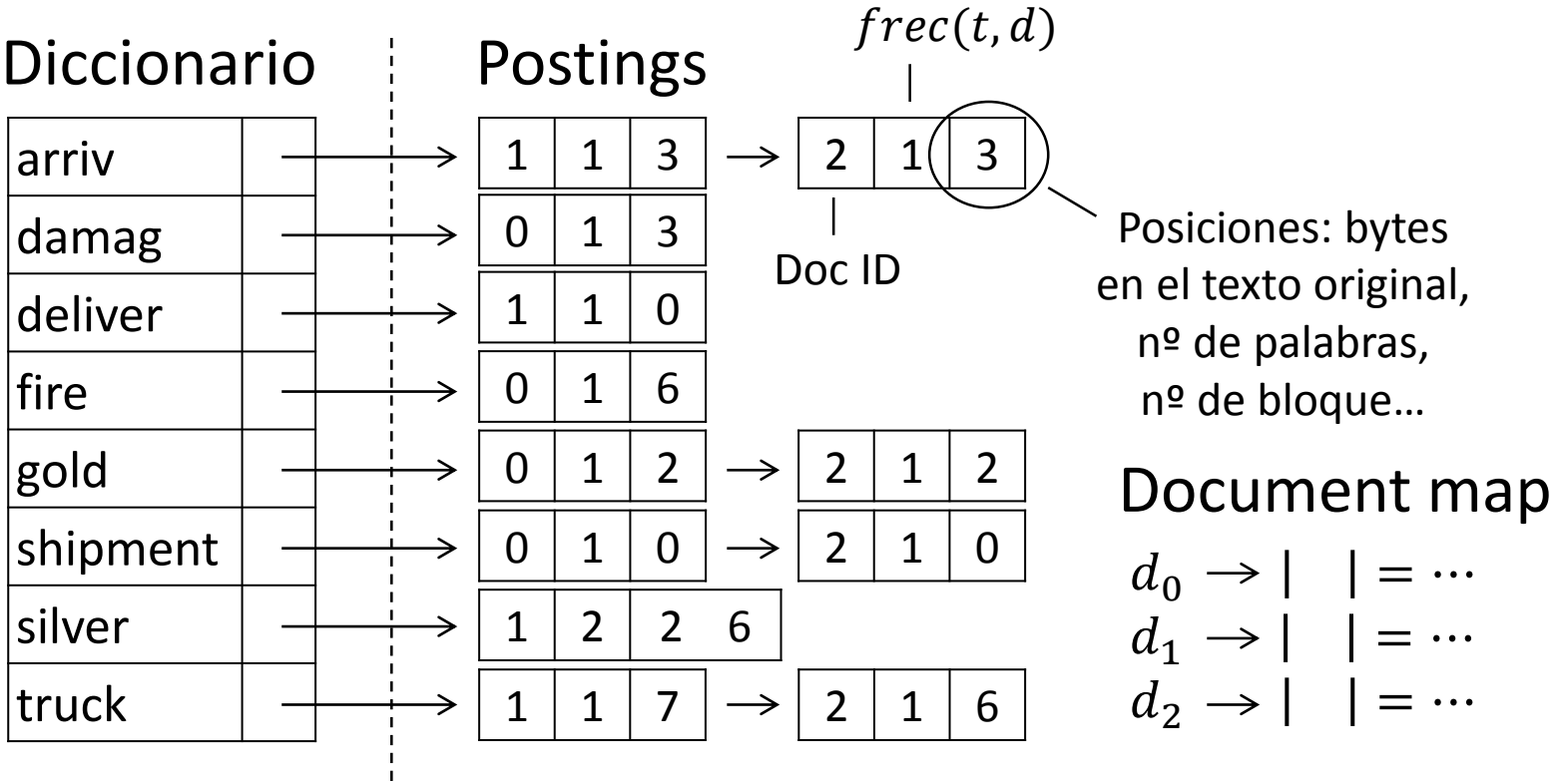
$d_2 =$ “Shipment of gold arrived in a truck”

Ejemplo

$d_0 =$ shipment gold damag fire

$d_1 =$ deliver silver arriv silver truck

$d_2 =$ shipment gold arriv truck

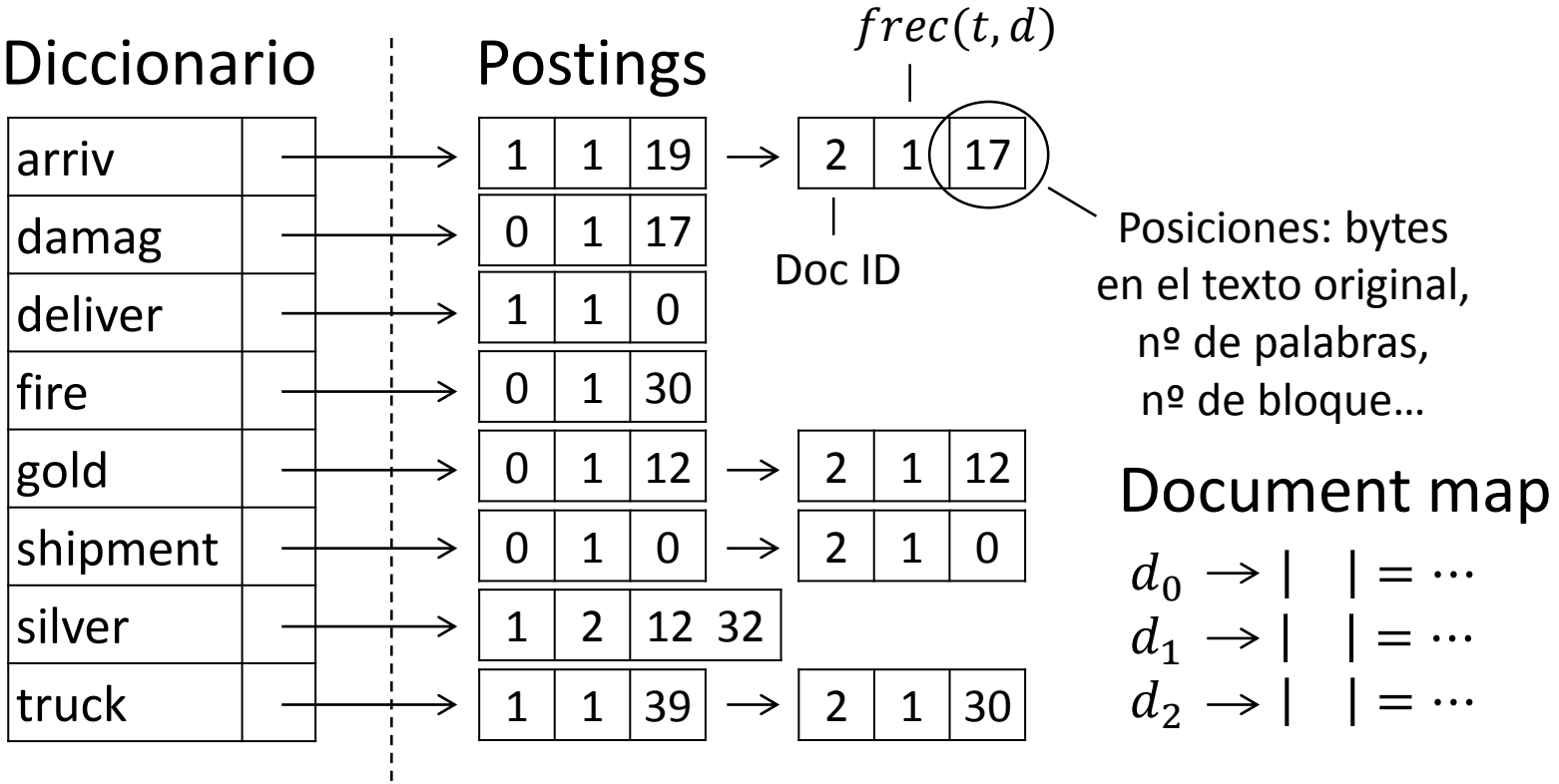


Ejemplo

$d_0 =$ shipment gold damag fire

$d_1 =$ deliver silver arriv silver truck

$d_2 =$ shipment gold arriv truck



Almacenamiento de los índices

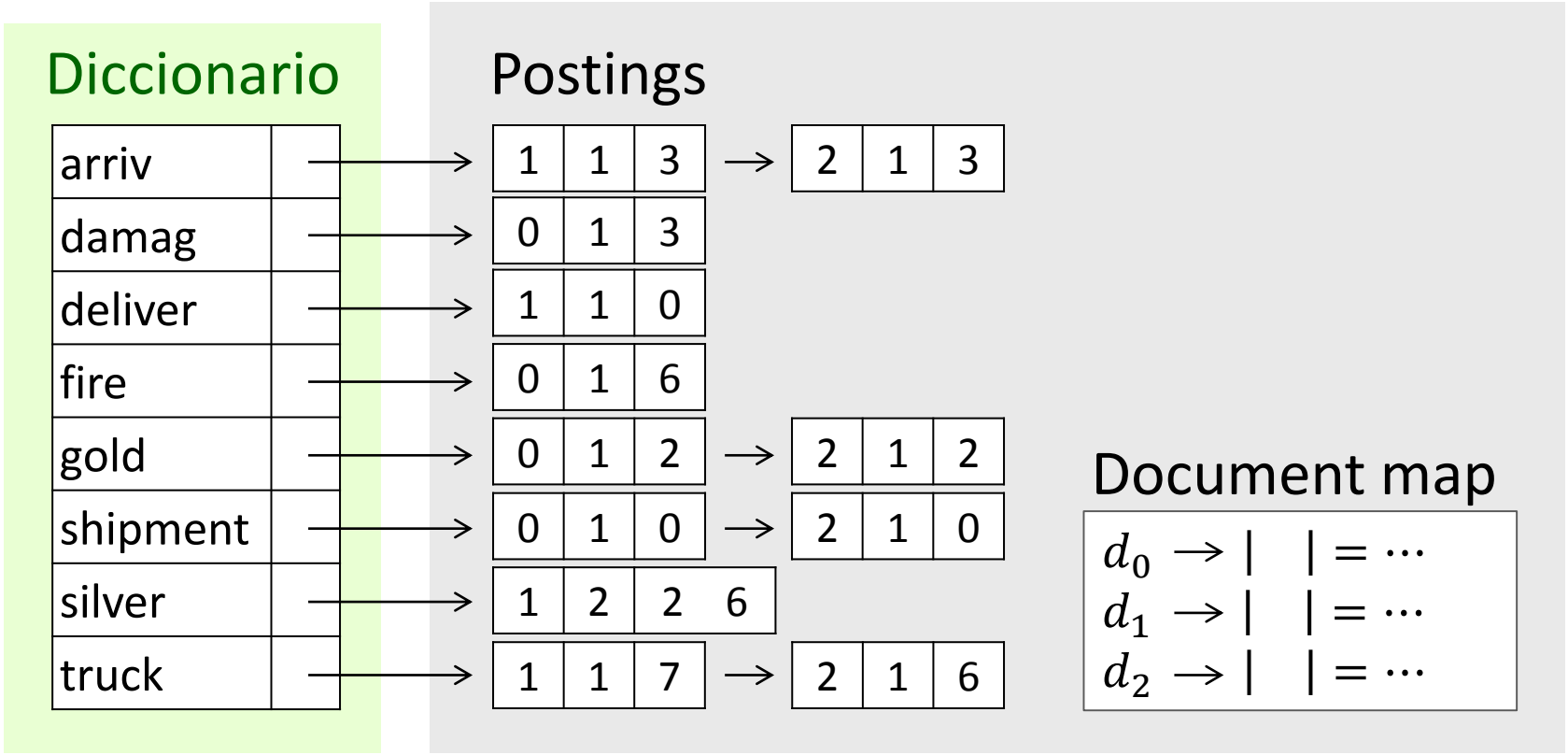
- Todo en RAM
  - Tablas hash para el diccionario, arrays para postings
  - Muy fácil de construir, búsquedas muy rápidas
  - Generalmente poco escalable: postings de millones de docs; en total los postings de un índice posicional ocupan $O\left(\sum_{d\in\mathcal{D},t\in\mathcal{V}} frec(t,d)\right)$ bytes
- Diccionario en RAM, postings en disco
  - Tabla hash o índice simple para diccionario
  - Postings ≡ registros de longitud variable
- Diccionario parcialmente en disco, postings en disco
  - P.e. árbol B para el diccionario, otras variantes
- El diccionario se archiva en cualquier caso en disco, aunque se mantenga en RAM

Ejemplo

$d_0 =$ shipment gold damag fire

$d_1 =$ deliver silver arriv silver truck

$d_2 =$ shipment gold arriv truck

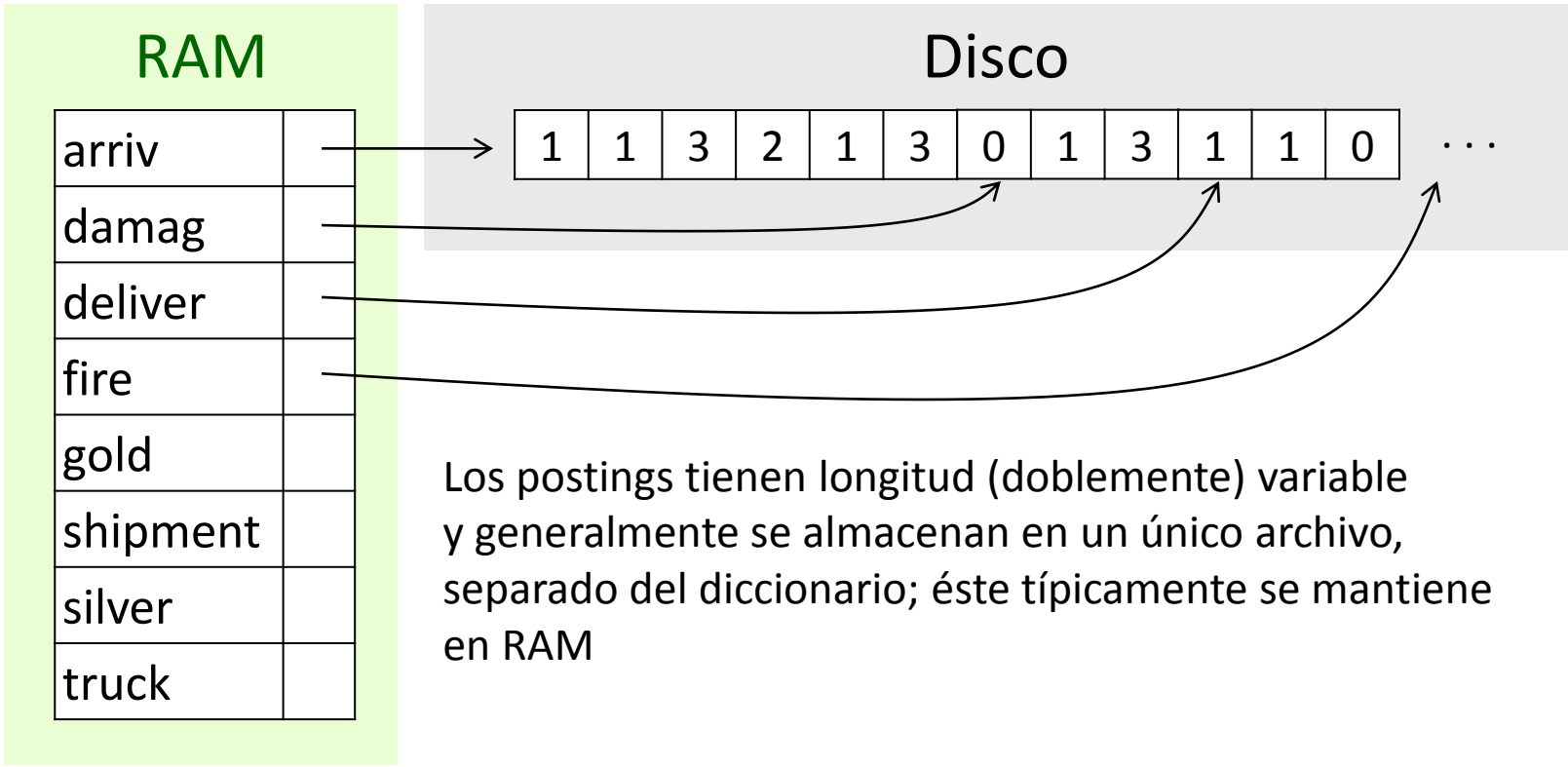


Ejemplo

$d_0 =$ shipment gold damag fire

$d_1 =$ deliver silver arriv silver truck

$d_2 =$ shipment gold arriv truck



Los postings tienen longitud (doblemente) variable y generalmente se almacenan en un único archivo, separado del diccionario; éste típicamente se mantiene en RAM

Ejemplo

$d_0 =$ shipment gold damag fire

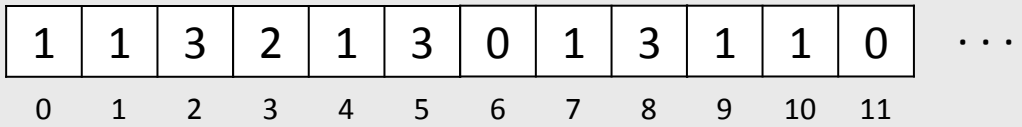
$d_1 =$ deliver silver arriv silver truck

$d_2 =$ shipment gold arriv truck

RAM

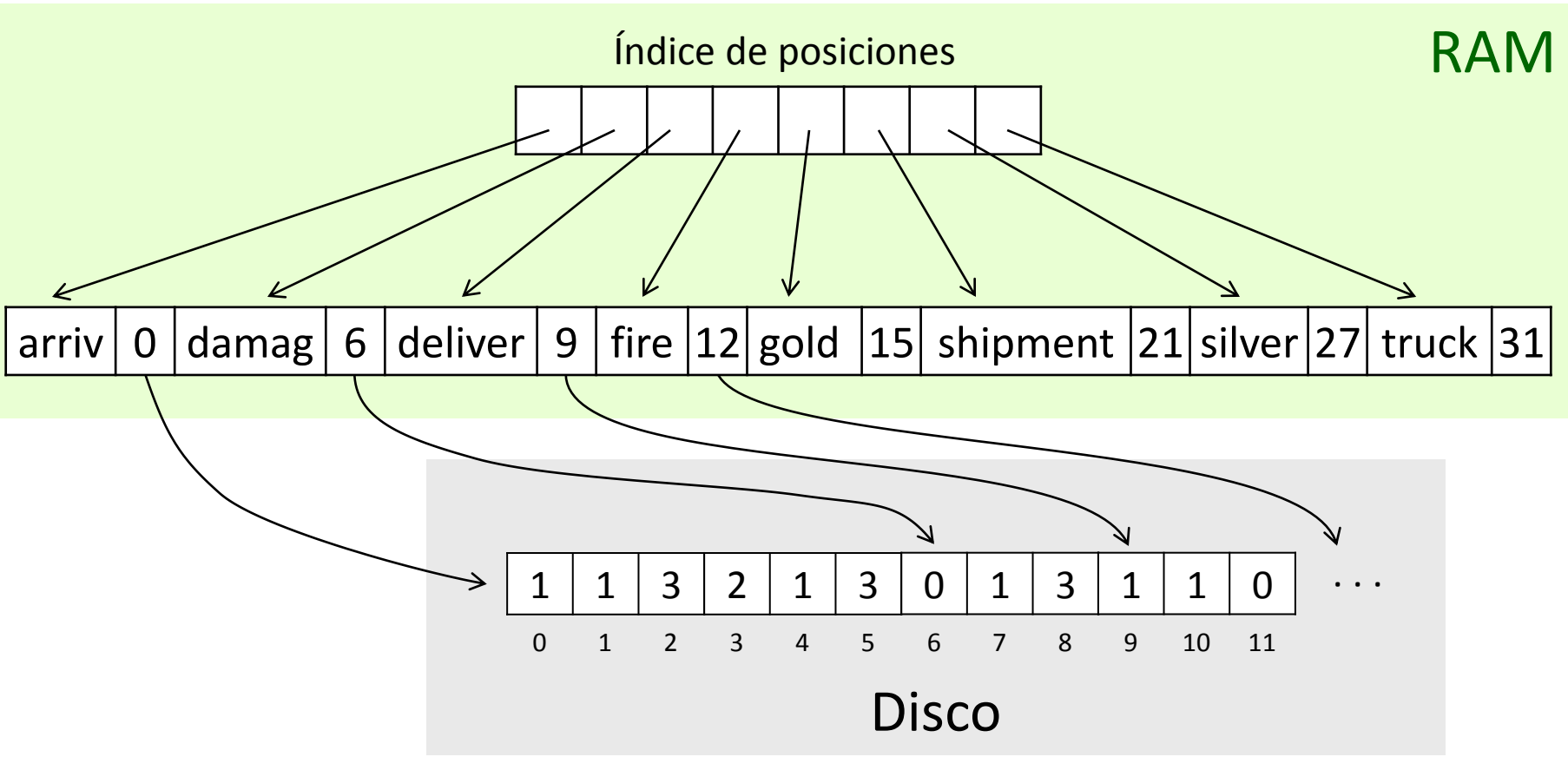
arriv	0
damag	6
deliver	9
fire	12
gold	15
shipment	21
silver	27
truck	31

Disco



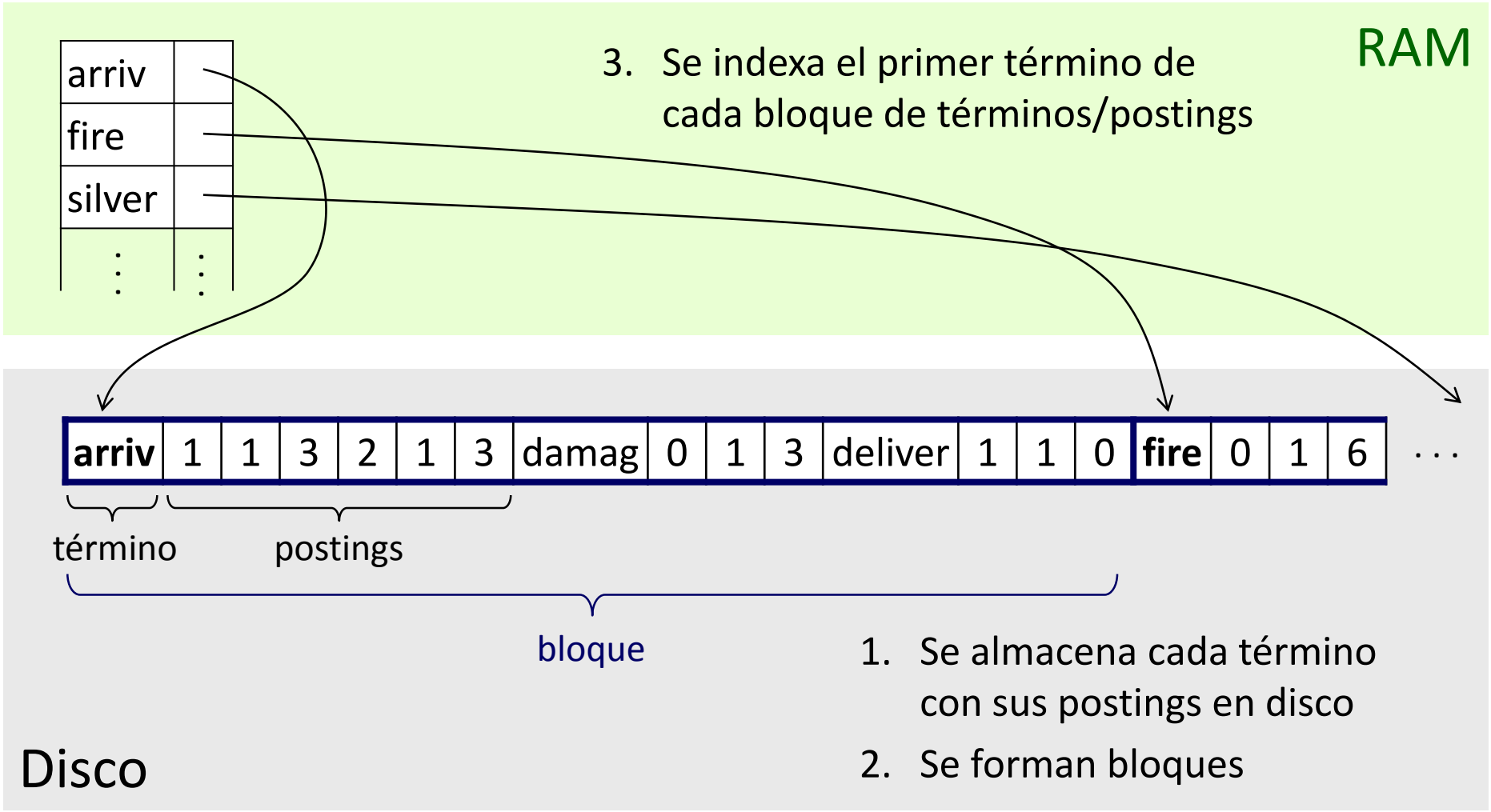
Los postings tienen longitud (doblemente) variable y generalmente se almacenan en un único archivo, separado del diccionario; éste típicamente se mantiene en RAM

Términos de longitud variable



Los términos suelen tener longitud muy diversa, por ello se suelen almacenar con longitud variable y un índice de posiciones

Más variaciones: diccionario intercalado



Y otras variaciones: árboles B, B+…

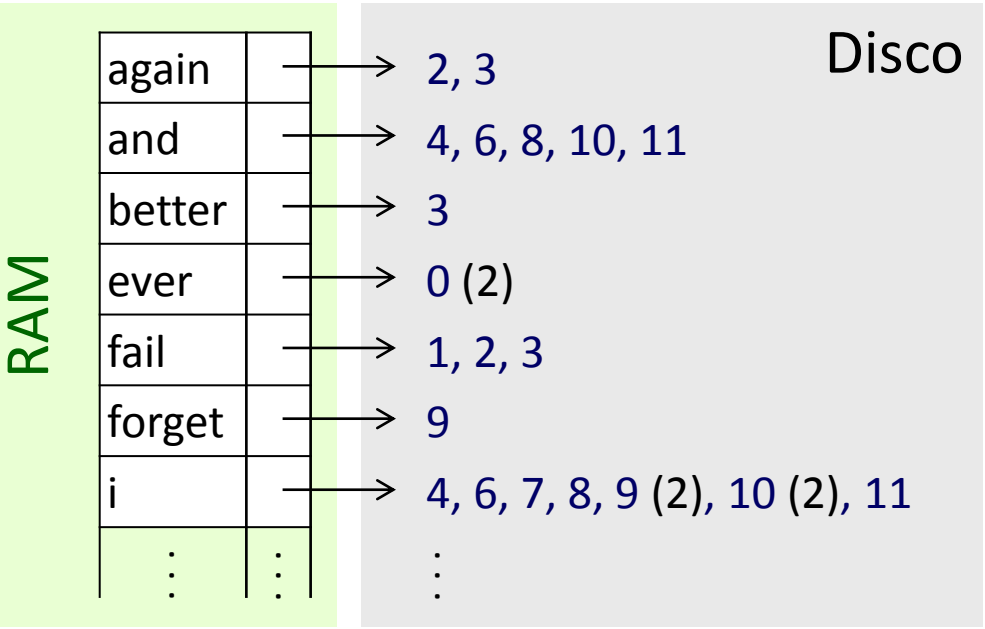
Direccionamiento por bloques de documento

- Permite reducir el espacio que ocupan las postings lists de los índices posicionales
- Se dividen los documentos en bloques
- Los términos se posicionan por bloques en lugar de documento y posición (el resto es todo igual que antes)
- Para recuperar la posición exacta se parsean sobre la marcha los bloques seleccionados, en tiempo de consulta

Direccionamiento por bloques

Disco

0	1	2	3
Ever tried. Ever	failed. No matter.	Try again. Fail	again. Fail better.
4	**5**	**6**	**7**
Tell me and I	forget. Teach me	and I remember.	Involve me and I
8	**9**	**10**	**11**
learn. I hear and	I forget. I see	and I remember. I	do and I understand



Si se necesitan las posiciones,
se obtienen en tiempo de consulta
procesando el bloque en RAM

Direccionamiento por bloques

Porcentaje del tamaño índice respecto a la colección original (sin / con stopwords)

Granularidad de direccionamiento ↓	1 doc 1MB		Pequeña colección 200MB		Colección media 2GB	
Palabras	45%	73%	36%	64%	35%	63%
Bloques 64K	27%	41%	18%	32%	5%	9%
Bloques 256K	18%	25%	1.7%	2.4%	0.55%	0.7%

R. Baeza and B. Ribeiro. Modern Information Retrieval 2nd ed. Addison-Wesley 2012

Diccionario hash

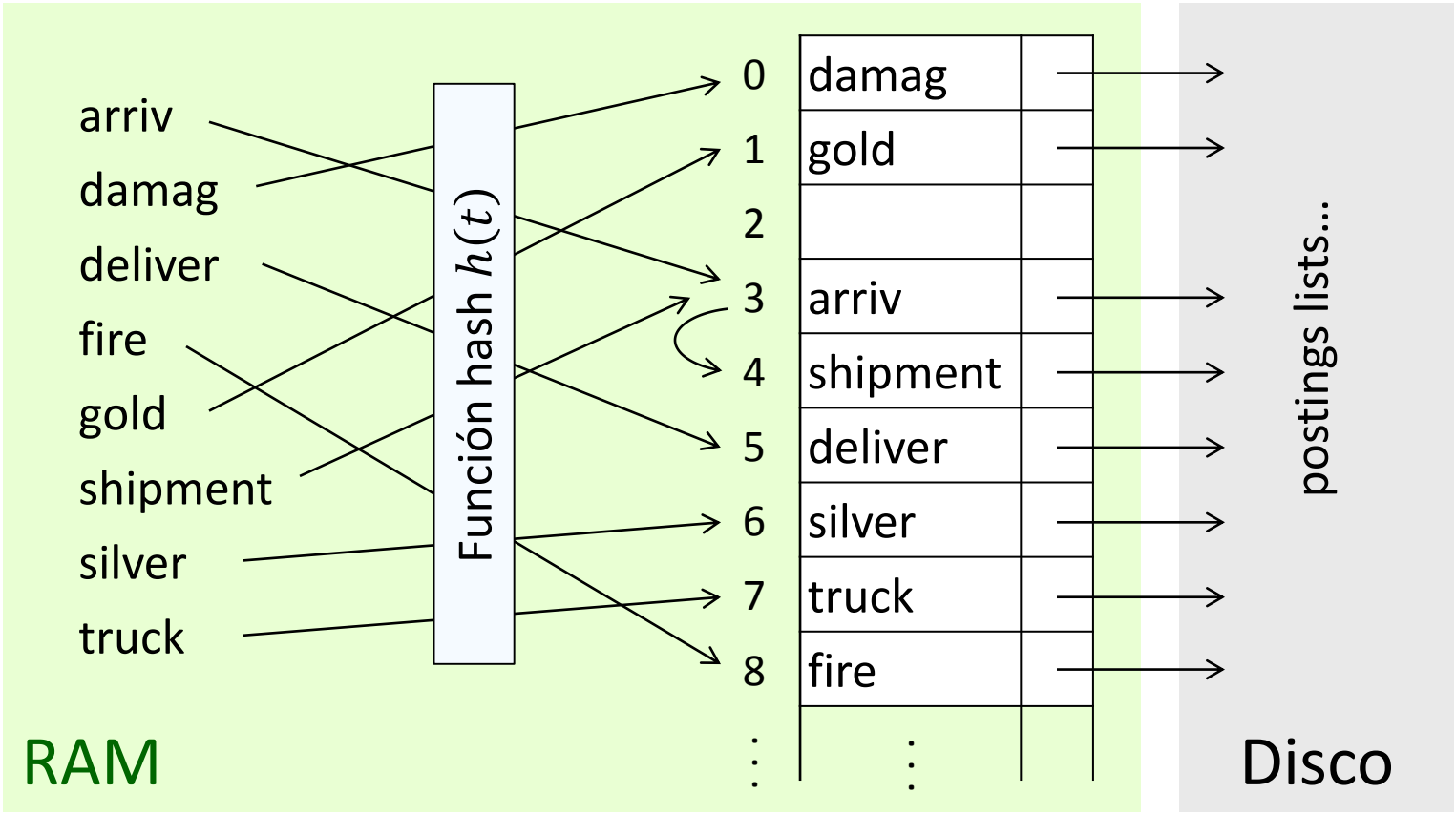
- Los términos se indexan por una función hash, en lugar de por orden
  - Todo lo demás es idéntico a los índices ordenados
  - Colisiones por desbordamiento, o listas enlazadas / arrays dinámicos
  - Prioridad a términos frecuentes en las colisiones (frecuentes en la colección para tiempo de indexado vs. en consultas para tiempo de consulta)
    - ☞ Aprovechar la ley de Zipf!!
- Tiempo de acceso más rápido que con índice ordenado
- Contrapartida: mayor tamaño de la estructura
  - Nº espacios hash >> nº términos para tasa de colisiones moderada
- Es común usar una tabla hash en tiempo de construcción del índice
  - Ya que se necesitan procesar *todas las palabras* de la colección
- Y un diccionario ordenado en tiempo de consulta
  - El tiempo de acceso a términos no es un cuello de botella

Ejemplo

$d_0 =$ shipment gold damag fire

$d_1 =$ deliver silver arriv silver truck

$d_2 =$ shipment gold arriv truck

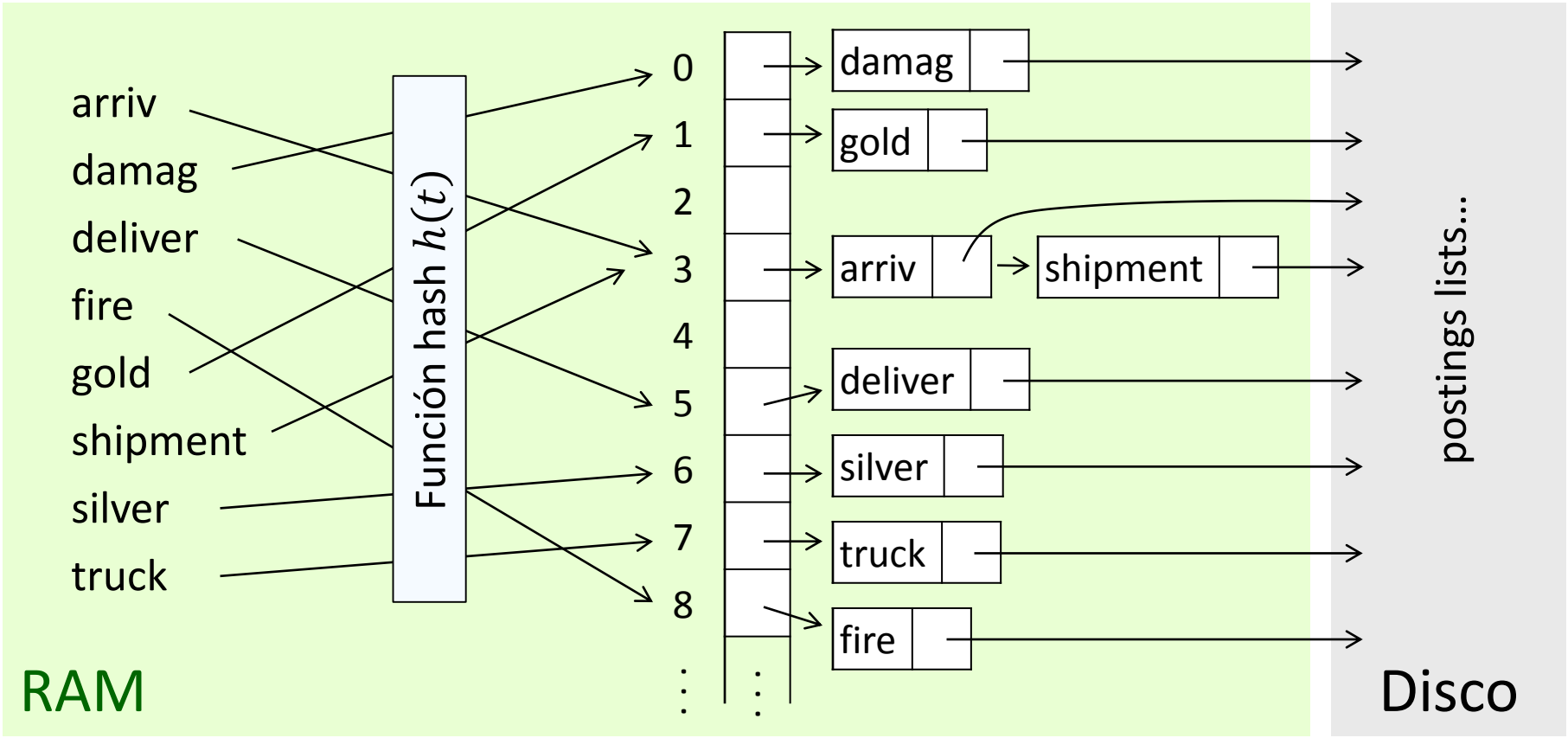


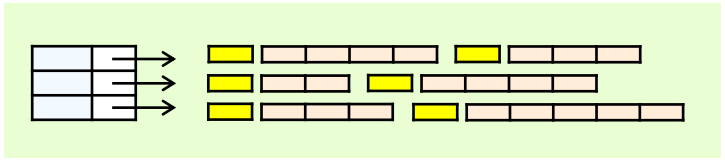
Ejemplo

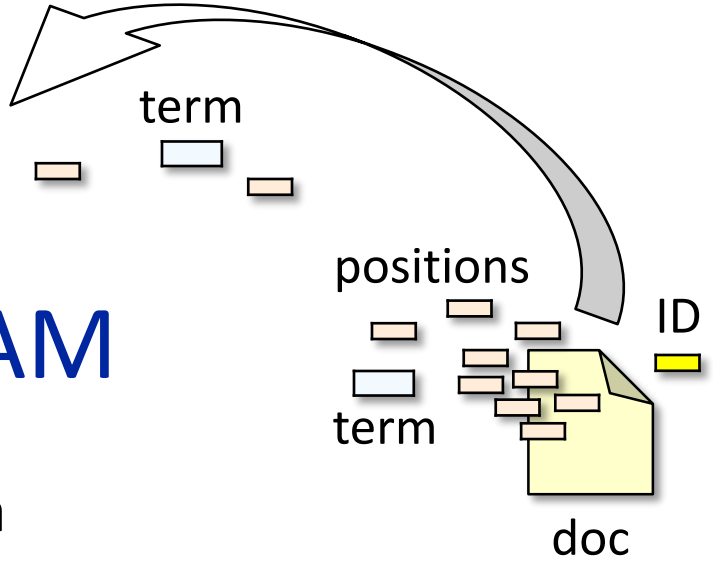
$d_0 =$ shipment gold damag fire

$d_1 =$ deliver silver arriv silver truck

$d_2 =$ shipment gold arriv truck







Construcción de índice RAM

- Es la solución más rápida con diferencia
  - Pero requiere muchísima RAM para una colección típica!
- Puede construirse en una sola pasada a la colección o en dos
  a) Dos pasadas:
     1. Calcular el tamaño de las listas de postings
     2. Añadir los valores
  b) Una sola pasada con arrays dinámicos para ir insertando los postings

En general es mejor b, más rápido a cambio de gasto de RAM

Ejemplo


0
ship
gold damag
fire
1
ship
gold arriv
truck
2
deliver
silver arriv
silver truck

Construcción de índice disco

Dos métodos

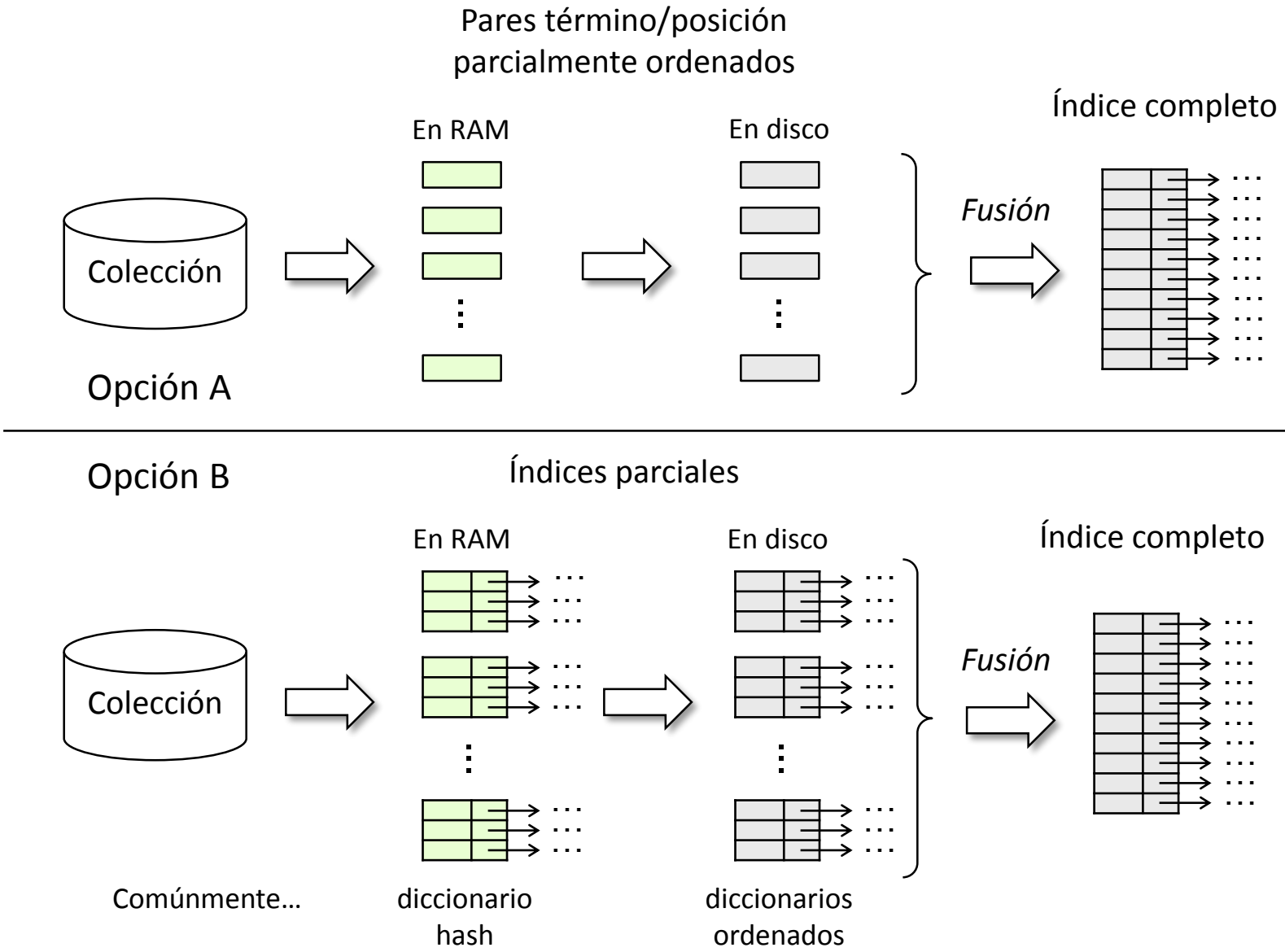
- Opción A: Orden total de pares término / posición
  1. Leer a RAM los pares término / posición
  2. Cuando no caben más pares en RAM, ordenar por término / posición
  3. Escribir a archivo auxiliar y volver a 1

  } Heapsort para solapar lectura/escritura

  4. Cuando se ha procesado toda la colección, fusión de los archivos auxiliares
     a) k-merge
     b) Merge multipaso
  5. Recorrer los pares ordenados y generar los postings y estructuras finales de índice (esto se puede hacer directamente en el último paso de merge)
- Opción B: Utilizar un método para índices en RAM
  1. Crear índice en RAM (no hace falta que use la estructura definitiva que tendrá en disco)
  2. Cuando se llena la RAM, generar postings y estructuras en disco (índices parciales)
  3. Cuando se ha procesado toda la colección, fusión de los índices de disco

Los índices finales no tienen por qué tener igual estructura que los temporales, pues interesa que éstos sean eficientes en tiempo de construcción

Construcción de índice disco (cont)

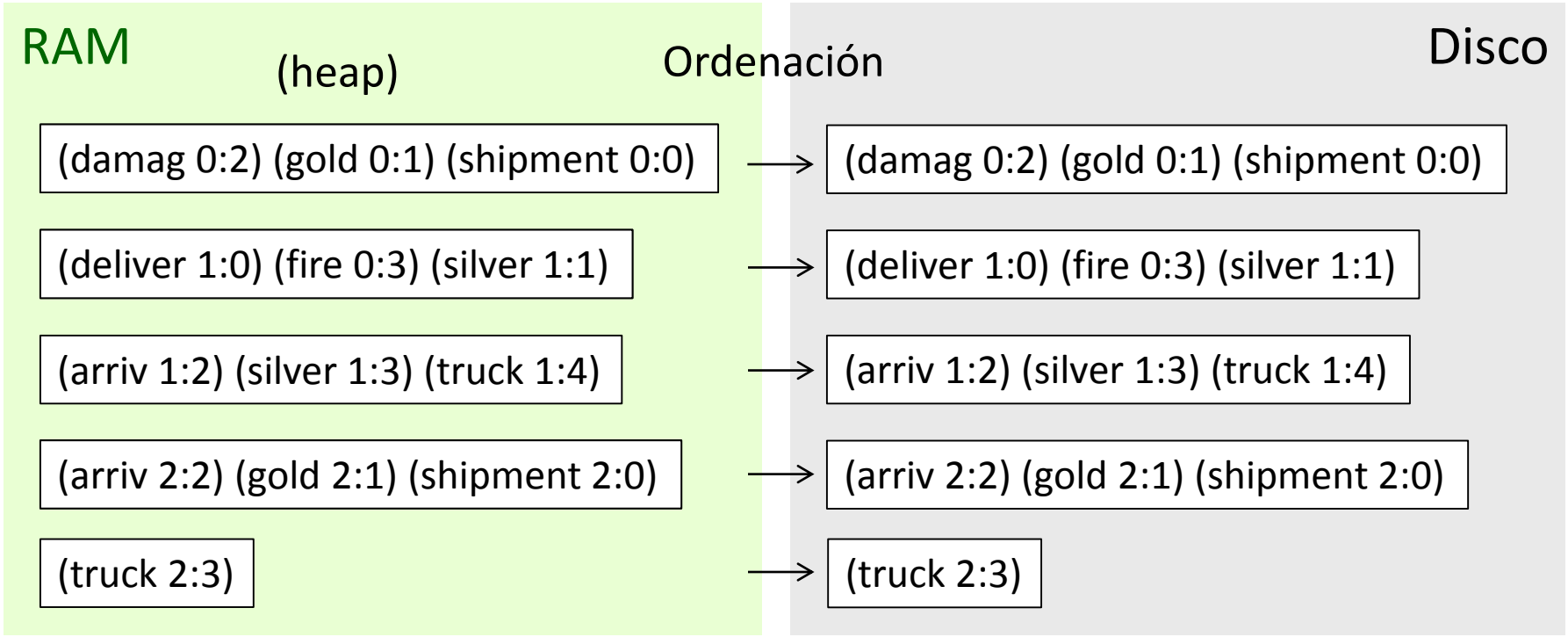


Ejemplo

$d_0 =$ shipment gold damag fire

$d_1 =$ deliver silver arriv silver truck

$d_2 =$ shipment gold arriv truck

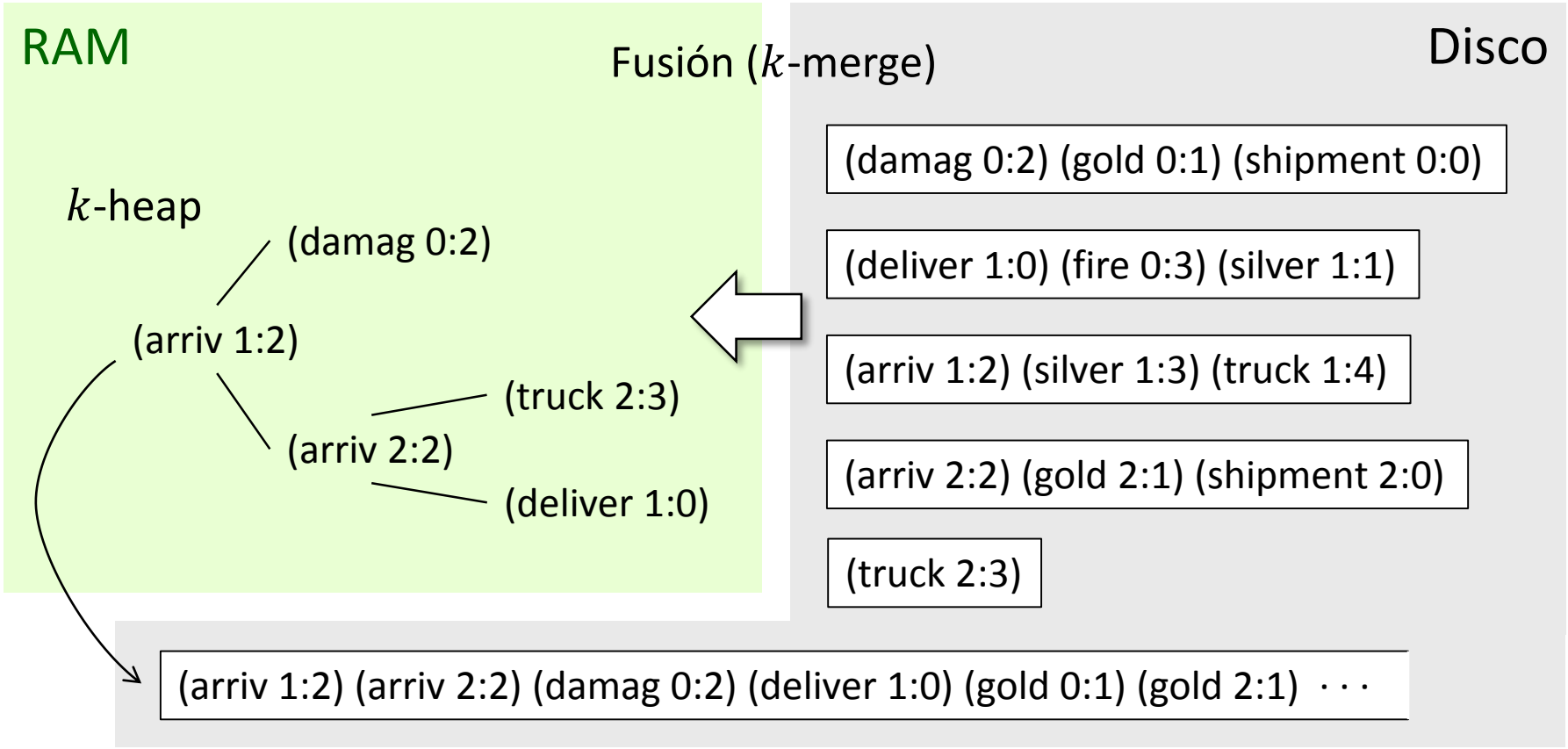


Ejemplo

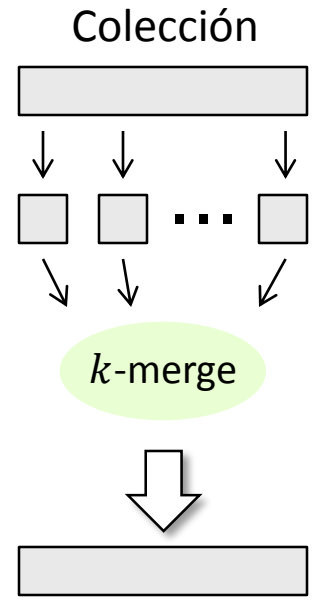
$d_0 =$ shipment gold damag fire

$d_1 =$ deliver silver arriv silver truck

$d_2 =$ shipment gold arriv truck



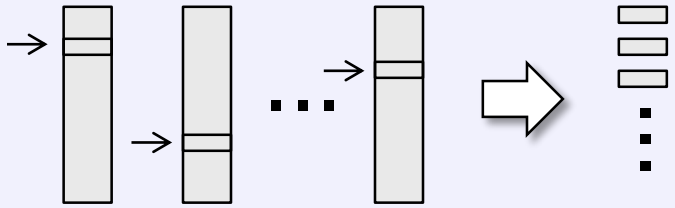
Merge simple



Pares término/posición
o índices parciales

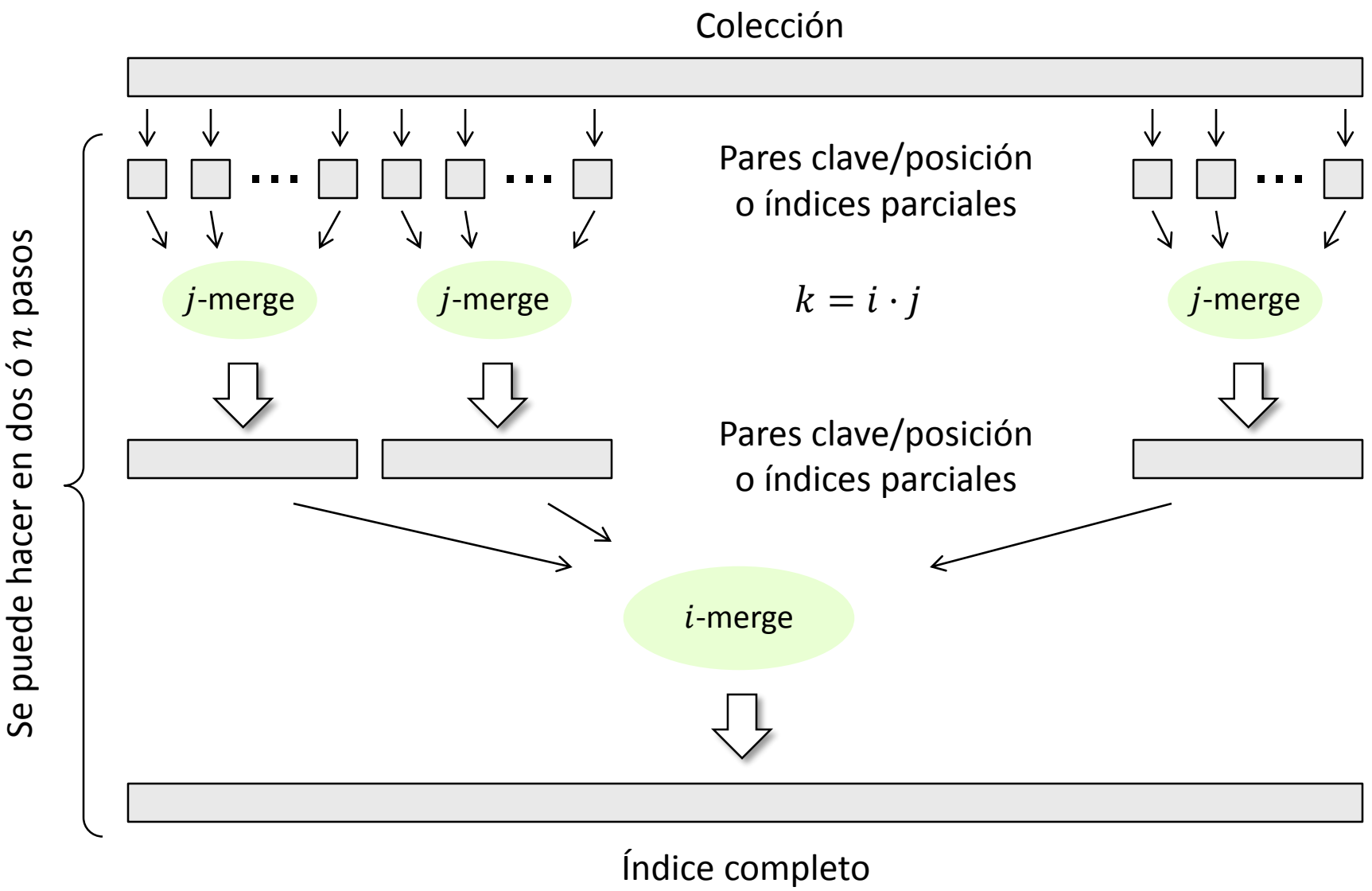
Procesamiento cosecuencial

- Iteración coordinada en varias listas



- Unión, intersección y variaciones (en este caso unión)
- Para k listas se suele utilizar un k-heap

Merge multipaso

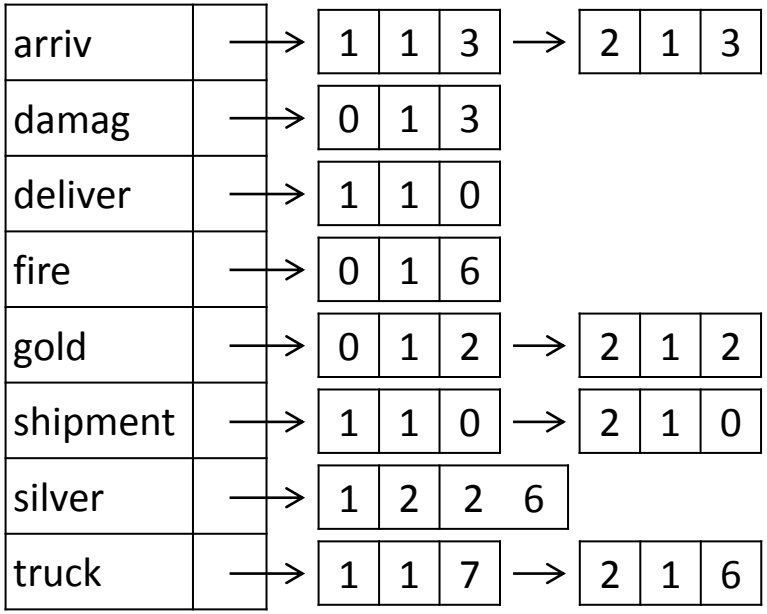


Fusión de índices

- Iterar cosecuencialmente t_1, t_2 sobre los diccionarios
- Si $t_1 \neq t_2$, añadir $\min(t_1, t_2)$ con sus postings al índice fusionado
- Si $t_1 = t_2$, añadir t_1 al índice fusionado, concatenando los postings de t_1 y t_2 (merge si el orden no fuese por docID)
- Se generaliza a k-merge igual que la ordenación de pares término/posición

Fusión de índices: ejemplo

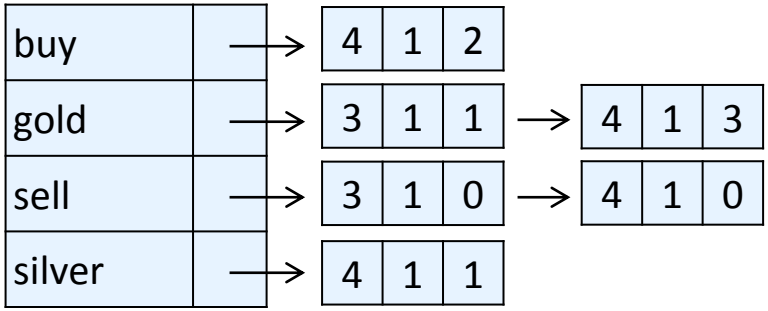
Índice principal



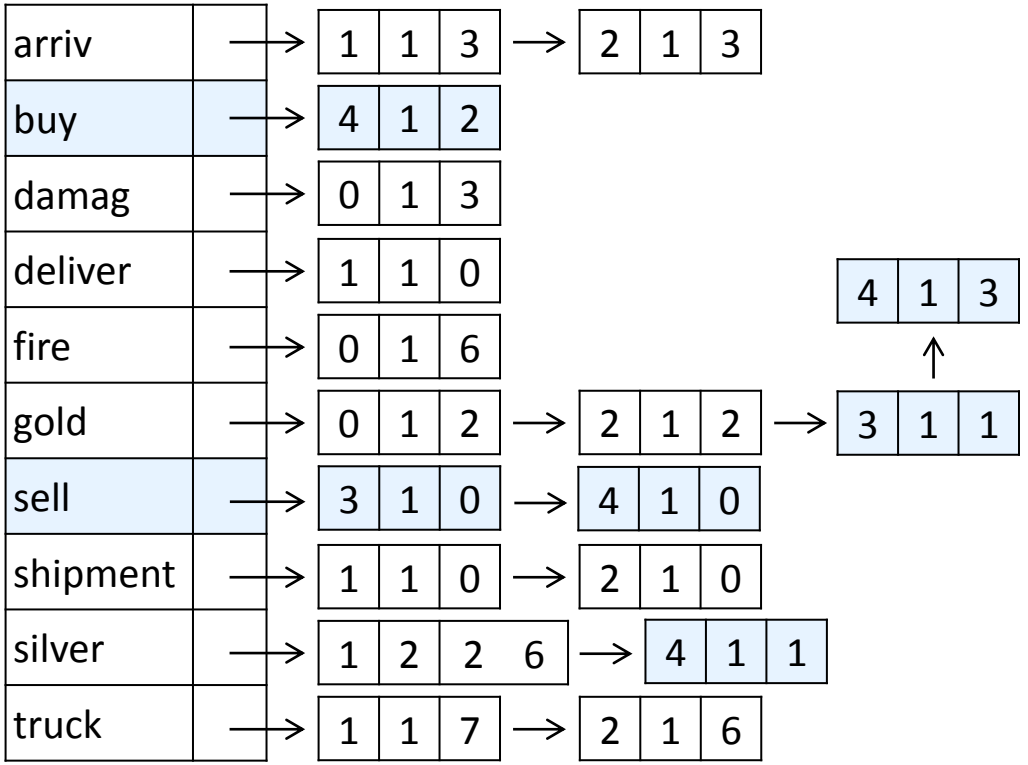
Añadir

d_3 = "sell gold"

d_4 = "sell silver buy gold"



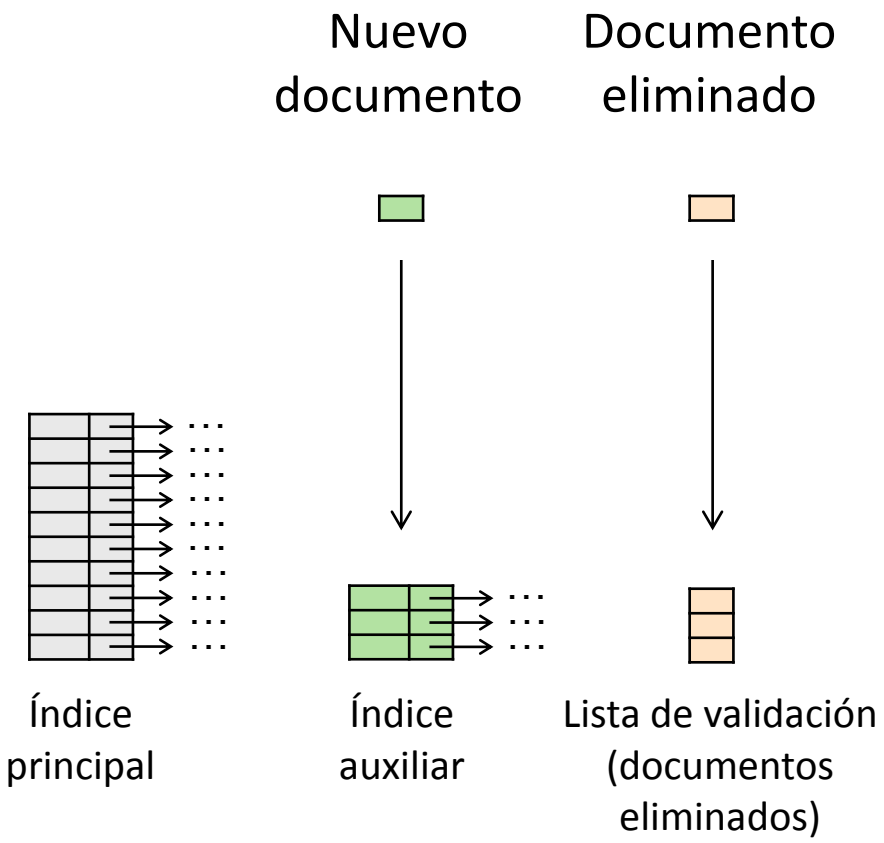
Índice resultante



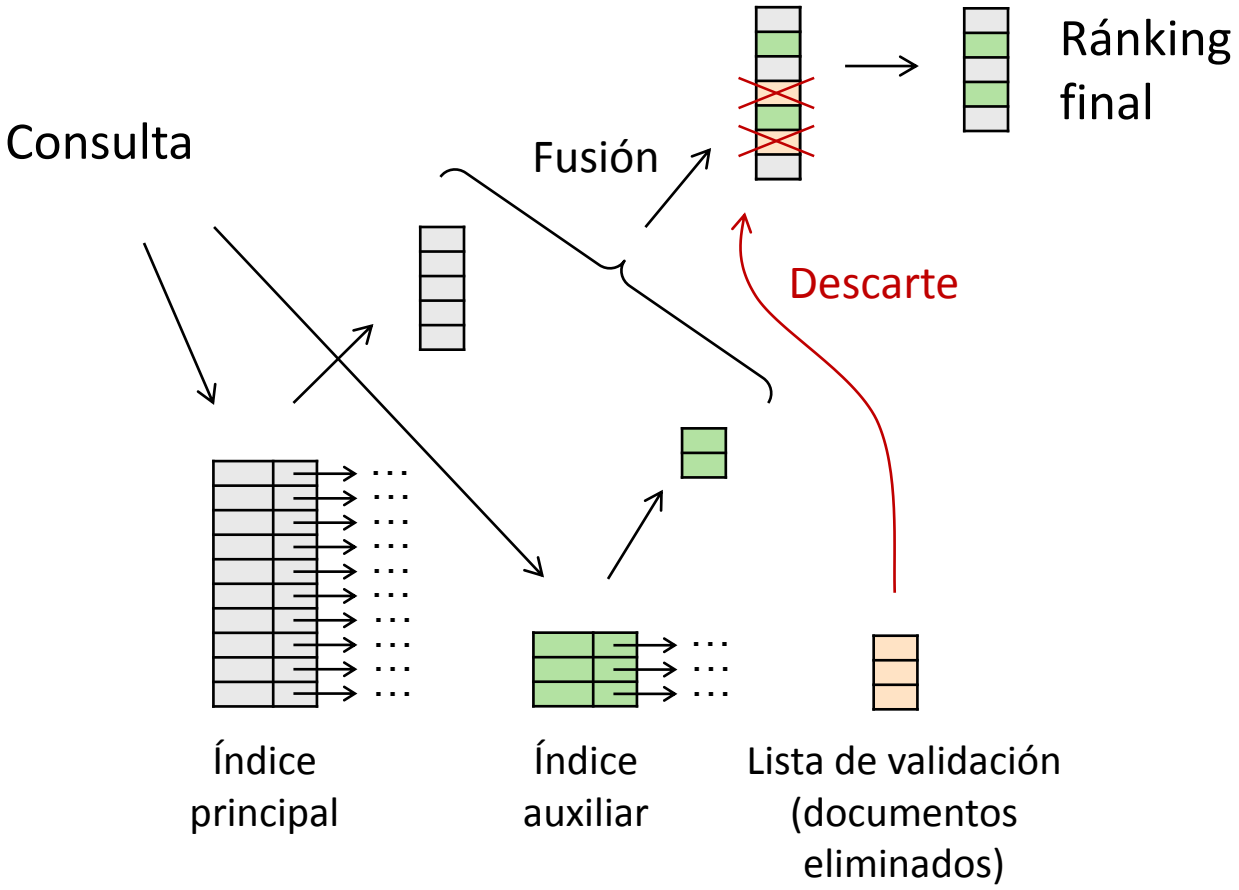
Actualización del índice

- Inserción de nuevos documentos, eliminación de documentos
- La reconstrucción desde cero es viable si la espera de actualización es aceptable
- En otro caso, actualización incremental mediante un pequeño índice auxiliar en RAM
  - Nuevos documentos: las nuevas entradas se añaden al índice auxiliar
  - Búsqueda: se manejan dos índices, uniendo (merge) la salida de ambos
  - Eliminación: se mantiene una lista de documentos eliminados, donde se comprueban los documentos antes de incluirlos en el resultado
  - Modificación de documentos: eliminación + inserción
- Cuando el índice auxiliar crece demasiado, se fusiona con el principal
  - Los documentos eliminados se purgan en ese momento
- Hay otras soluciones para una actualización instantánea de índice único
  - P.e. para modelos IR donde los cálculos no son separables por índices parciales

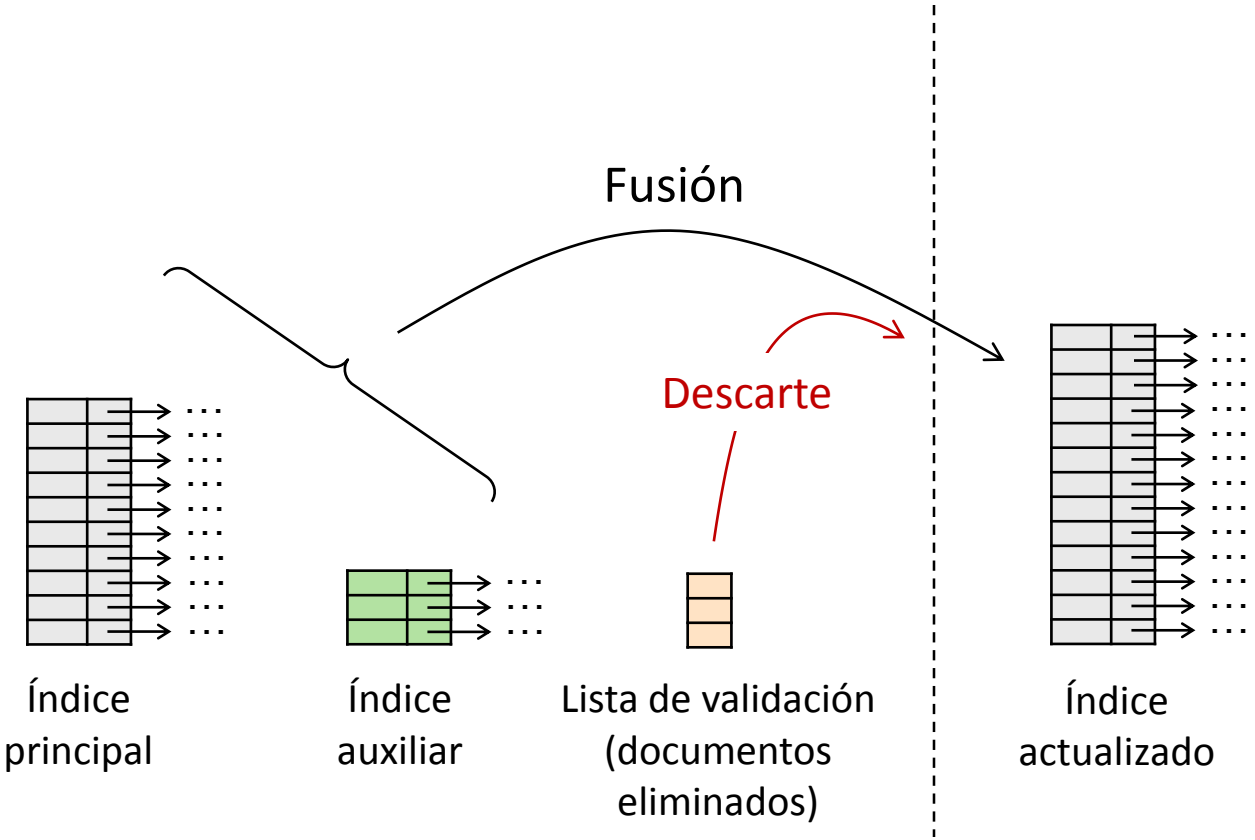
Actualización incremental del índice



Actualización incremental del índice



Actualización incremental del índice



Actualización del índice (cont)

- En los motores comerciales, la actualización de índices es frecuente, por ello se tiende a la opción incremental que es más eficiente en esta situación
- La actualización incremental es ineficiente en la eliminación de documentos
  - Si la tasa de eliminación es muy alta, compensa reconstruir

Implementación del modelo vectorial

Recordemos la función de ránking

$$f(d,q) = \cos(d,q) = \frac{d \cdot q}{|d||q|} \propto \frac{1}{|d|}\sum_{t \in \mathcal{V}} d_t q_t$$

Donde podemos utilizar una versión de *tf-idf* tipo:

$$d_t = tf\text{–}idf(t,d) = \begin{cases} (1 + \log_2 frec(t,d)) \log \dfrac{|\mathcal{D}|}{|\mathcal{D}_t|} & \text{si } t \in d \\ 0 \quad \text{en otro caso} \end{cases}$$

$$q_t = tf(t,q) = \begin{cases} 1 + \log_2 frec(t,q) = 1 & \text{si } t \in q \\ 0 \quad \text{en otro caso} \end{cases}$$

Con lo cual: $$f(d,q) = \frac{1}{|d|}\sum_{q_i \in q} tf\text{–}idf(q_i,d)$$

Lo que sigue vale también para otros modelos de IR con estructura similar

$$f(d,q) = \gamma(d) + \eta(d)\sum_{q_i \in q} \omega(q_i,d)$$ p.e. BM25, etc.

Implementación del modelo vectorial

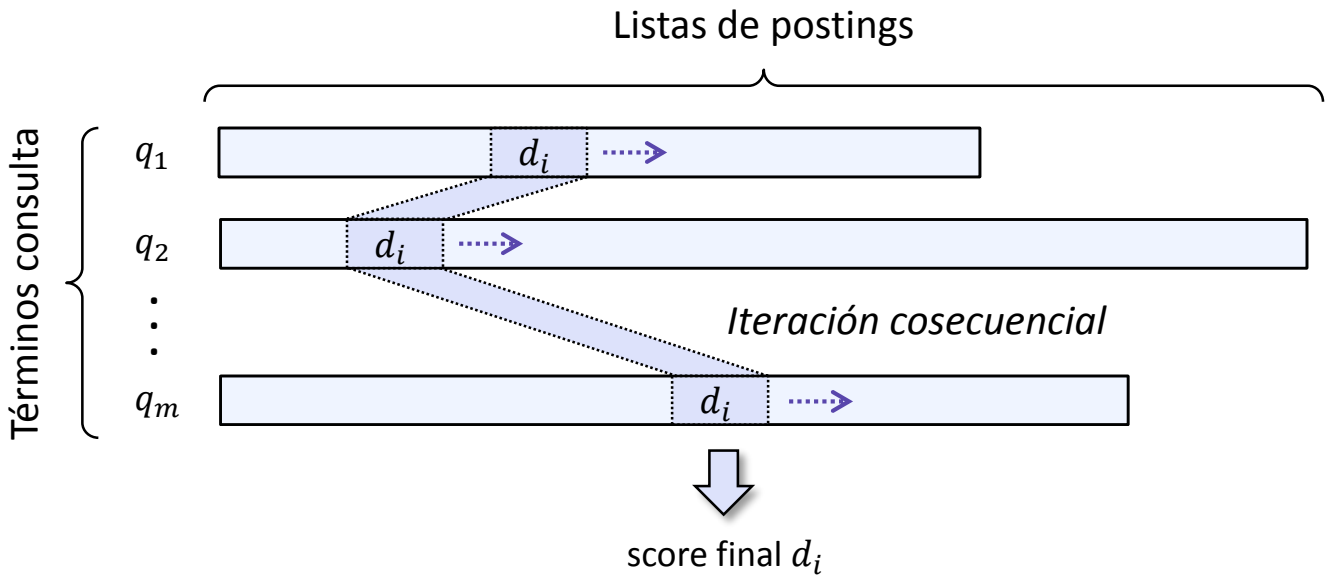
- Implementar el cómputo de una función como $\frac{1}{|d|}\sum_{q_i \in q} tf\text{-}idf(q_i, d)$ puede parecer muy trivial
  - Por ejemplo, recorrer todos los documentos d y para cada documento iterar en $q_i \in q$ y sumar $tf\text{-}idf(q_i, d)$
  - Además, tener precalculado $|d|$ para todos los documentos al indexar
- Pero…
  - Iterar sobre todos los documentos es un coste enorme e innecesario
  - El índice no está organizado por documentos, sino por términos
  - Las dimensiones del índice hacen que la viabilidad de cualquier implementación sea un reto
- Así pues hay que hacer algoritmos…

  …que tengan en cuenta la estructura del índice

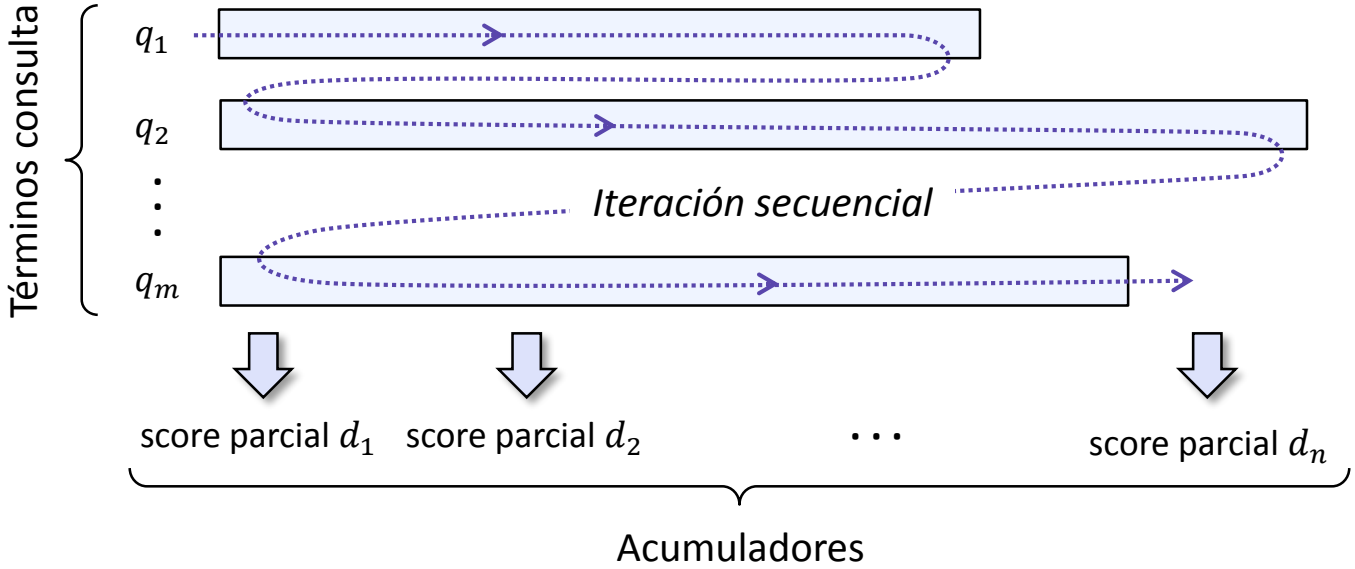
  …que optimicen el coste en tiempo y gasto de RAM

Implementación del modelo vectorial

A. Orientado a documentos



B. Orientado a términos



Implementación del modelo vectorial Método orientado a documentos

- Postings **ordenados por docID**
- Iterar cosecuencialmente en las postings lists de los términos de la consulta
  - Usar un heap (tamaño = longitud de la consulta) para iterar por orden de docID
- Para cada documento, sumar *tf-idf* de cada posting que contiene al documento
  - Avanzar en la iteración cosecuencial a cada posting procesado
  - Si se quiere implementar un *and* implícito en la consulta, desechar los documentos que no aparecen en todas las listas de postings (procesamiento cosecuencial tipo "intersección")
- Normalizar la suma por el módulo del documento (este dato conviene guardarlo en el –o junto al– índice)
- Añadir el par (doc, score) a un **min-heap** de tamaño k
  - Sólo si el score es mayor que el que está en la cima del heap

Ahorro drástico de RAM y tiempo de ordenación

- Una vez agotados los postings de todos los términos, el ránking se obtiene extrayendo los documentos del heap
- Otros modelos IR (BM25, etc.) se implementan de forma parecida

Ejemplo

Postings

tf-idf (para simplificar el ejemplo)

q1 (d1, 1) (d2, 2) (d4, 3)

q2 (d2, 3) (d3, 1) (d4, 1)

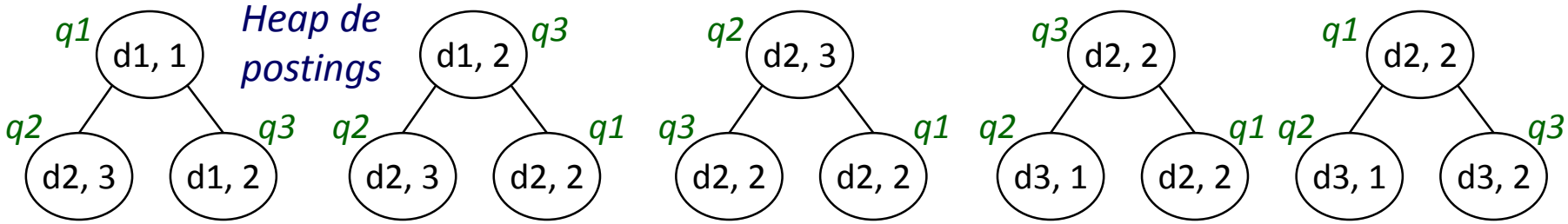
q3 (d1, 2) (d2, 2) (d3, 2) (d4, 2)

q = (q1, q2, q3)

Devolver top 3

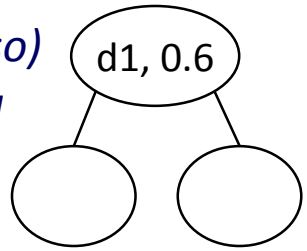
Supongamos |d1| = 5, |d2| = 10, |d3| = 15, |d4| = 15

Heap de postings

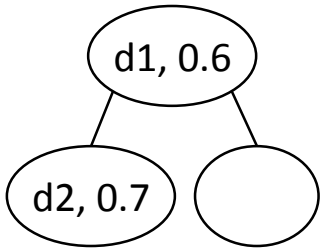


q1	(d2, 2) (d4, 3)	(d4, 3)	(d4, 3)	(d4, 3)	(d4, 3)
q2	(d3, 1) (d4, 1)	(d3, 1) (d4, 1)	(d3, 1) (d4, 1)	(d4, 1)	(d4, 1)
q3	(d2, 2) (d3, 2) (d4, 2)	(d2, 2) (d3, 2) (d4, 2)	(d3, 2) (d4, 2)	(d3, 2) (d4, 2)	(d4, 2)
	score(d1) = 1	score(d1) = 1+2	score(d2) = 3	score(d2) = 3+2	score(d2) = 3+2+2

Heap (inverso) de ránking



tamaño heap postings = long consulta
tamaño heap ránking = long top k



Ejemplo

Postings

tf-idf (para simplificar el ejemplo)

q1 (d1, 1) (d2, 2) (d4, 3)

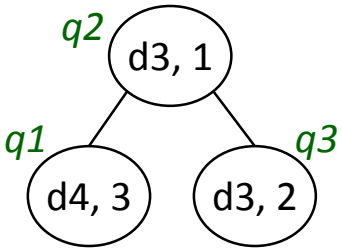
q2 (d2, 3) (d3, 1) (d4, 1)

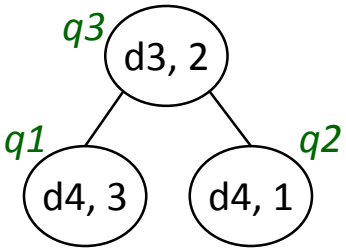
q3 (d1, 2) (d2, 2) (d3, 2) (d4, 2)

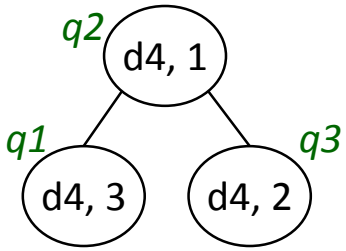
q = (q1, q2, q3)

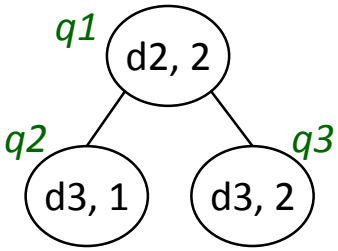
Devolver top 3

Supongamos |d1| = 5, |d2| = 10, |d3| = 15, |d4| = 15

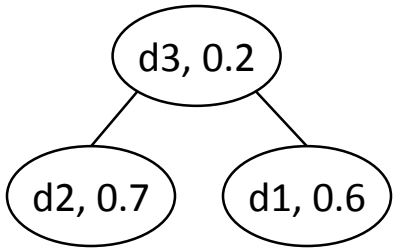


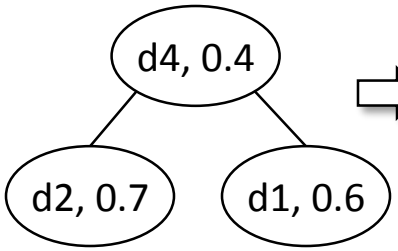






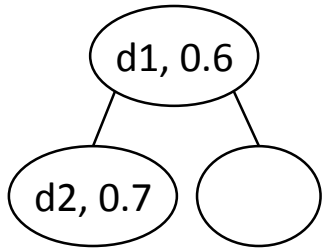
q1	∅	∅	(d4, 3)
q2	(d4, 1)	∅	(d4, 1)
q3	(d4, 2)	(d4, 2)	(d4, 2)
	score(d3) = 1	score(d3) = 1+2	score(d4) = 1+2+3	score(d2) = 3+2+2





Ránking final

d2	**0.7**
d1	**0.6**
d4	**0.4**



Implementación del modelo vectorial Método orientado a documentos

- Nótese que estamos aplicando una forma de OR de las palabras de la consulta
  - Así actúa el VSM
- Podríamos querer hacer un AND
  - Esa opción resulta computacionalmente más eficiente (¿por qué?)
  - De hecho no necesitaríamos heaps de postings (¿por qué?)
  - Los resultados son obviamente distintos (un subconjunto) del resultado OR

Implementación del modelo vectorial Método orientado a términos

- Iterar sobre las postings lists completas de cada término de la consulta, en secuencia una detrás de otra
  - Puede dar ventajas en consultas largas, poda, y/o paralelización
- Mantener una lista de pares documento/acumulador de score
  - Ordenado igual que las postings lists (p.e. por doc ID)
  - Al iterar sobre cada lista de postings, se itera cosecuencialmente en los acumuladores
- Para cada posting…
  - Se suma el *tf-idf* al score del documento correspondiente
  - Se añade un nuevo acumulador si el documento no estaba en la lista
- Finalmente los acumuladores contienen los scores de ránking
  - Ordenar esa lista por scores $\rightarrow$ ránking final
- Ventaja cuando los postings no caben en RAM: se procesan secuencialmente
  - Mientras que en la orientada a documentos se procesan cosecuencialmente

Ejemplo

Postings

q1 (d1, 1) (d2, 2) (d4, 3)

q2 (d2, 3) (d3, 1) (d4, 1)

q3 (d1, 2) (d2, 2) (d3, 2) (d4, 2)

q = (q1, q2, q3)

Devolver top 3

Supongamos |d1| = 5, |d2| = 10, |d3| = 15, |d4| = 15

Acumuladores

1. Procesamos q1 → [**d1 1**] [**d2 2**] [**d4 3**]
2. Procesamos q2 → [d1 1] [d2 **5**] [**d3 1**] [d4 **4**]
3. Procesamos q3 → [d1 **3**] [d2 **7**] [d3 **3**] [d4 **6**]

Scores finales

Dividir por |d| → [d1 0.6] [d2 0.7] [d3 0.2] [d4 0.4]

Ránking final

Ordenar

Top 3

d2	**0.7**
d1	**0.6**
d4	**0.4**
d3	**0.2**

Implementación del modelo vectorial

- El método orientado a documentos tiende a ser más rápido que el orientado a términos
- Pero puede tener contrapartidas en consumo de RAM y/o acceso a disco, o incluso ser más lento
  - Depende de las estadísticas de la colección, la consulta, y los recursos RAM/disco disponibles
- El método orientado a términos facilita por otra parte la poda de resultados

Poda de resultados

- En la práctica los buscadores de gran escala no devuelven todos los resultados
  - No merece la pena cuando son cientos de miles
  - Y además es un cuello de botella generar listas tan largas (millones de docs en Web)
  - P.e. Google no devuelve más de 1.000 resultados (probar ?q=⋯&start=⋯)
  - Se informa al usuario de una estimación del nº total de resultados
- Otras veces simplemente interesa un ránking top k
- Poda de acumuladores (implementación orientada a términos)
  - Iterar sobre los términos de la consulta por orden de (mayor a menor) *idf* del término
  - Ordenar los postings en el índice por *tf* (o el score del modelo IR a utilizar)
    El orden por docID ya no importa tanto (soluciones híbridas si se quiere conservar)
  - Limitar la longitud de la lista de acumuladores (límite $\gg k$) y no crear más llegado ese límite (sólo actualizar los acumuladores ya creados)
- Ránking inexacto: se podrían perder resultados en la cola del ránking
  - Pero compensa en costes, más considerando la imprecisión inherente a los modelos

Estimación del nº de resultados

Longitud de la lista de postings de t

Notación:

$$df(t) \equiv |\mathcal{D}_t|$$

- Para informar al usuario cuando no se devuelven todos
- Suponiendo independencia de términos

$$\frac{df(q_1, q_2, q_3)}{|\mathcal{D}|} \sim \frac{df(q_1)}{|\mathcal{D}|} \frac{df(q_2)}{|\mathcal{D}|} \frac{df(q_3)}{|\mathcal{D}|} \quad /* \text{ Hip. } q_1 \perp q_2 \perp q_3 \ */$$

$$\Rightarrow df(q_1, q_2, q_3) \sim df(q_1) df(q_2) df(q_3) / |\mathcal{D}|^2$$

- Con hipótesis de independencia más suaves, conociendo frecuencias conjuntas

$$df(q_1, q_2, q_3) \sim df(q_1, q_3) df(q_1, q_2) / df(q_1) \quad /* \text{ Hip. } q_2 \perp q_3 \mid q_1 \ */$$

- Si admitimos resultados con *algunos* términos…

$$df(q_1 \vee q_2) \sim df(q_1) + df(q_2) - df(q_1, q_2)$$

Estimación del tamaño de la colección

- Cuando no tenemos acceso directo al índice ni la colección
- Interesante p.e. con buscadores Web
- A partir de la frecuencia de palabras:

$$\frac{df(t_1,t_2)}{|\mathcal{D}|} \sim \frac{df(t_1)}{|\mathcal{D}|}\frac{df(t_2)}{|\mathcal{D}|} \quad \Rightarrow \quad |\mathcal{D}| \sim \frac{df(t_1)df(t_2)}{df(t_1,t_2)}$$

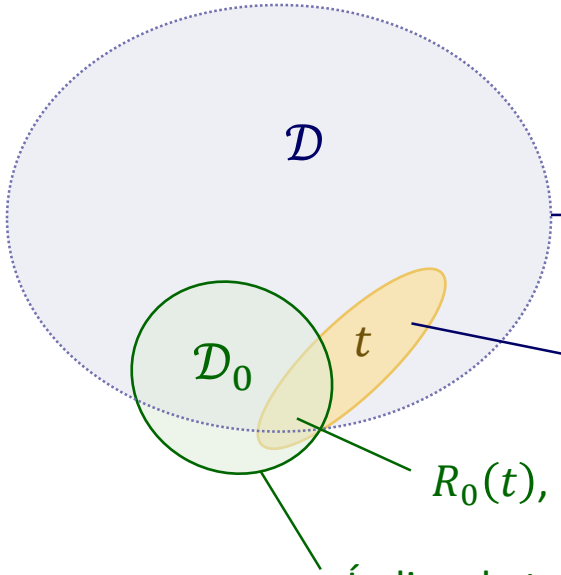
(utilizar t_1, t_2 lo más independientes posible)

- O simplemente extrapolando la frecuencia en un corpus de muestra (p.e. $df_{ODP}(\text{"}the\text{"})/|\text{ODP}| \sim 0.67$) al corpus global (p.e. la Web)

$$\frac{df_0(t)}{|\mathcal{D}_0|} \sim \frac{df(t)}{|\mathcal{D}|} \quad \Rightarrow \quad |\mathcal{D}| \sim \frac{df(t)}{df_0(t)}|\mathcal{D}_0|$$

- La estimación de $df(t)$ la podemos obtener ejecutando t como consulta en el buscador

Estimación del tamaño de la colección



Índice de tamaño a estimar

$R(t)$, resultados buscando t como consulta en $\mathcal{D}$

$R_0(t)$, resultados buscando t como consulta en $\mathcal{D}_0$

Índice de tamaño conocido (p.e. ODP)

$$|\mathcal{D}| \sim \frac{df(t)}{df_0(t)}|\mathcal{D}_0| \sim \frac{|R(t)|}{|R_0(t)|}|\mathcal{D}_0|$$

Implementación del modelo booleano

- Obtener los postings (docIDs) de los términos de la consulta
  - No se necesitan frecuencias ni posiciones en los postings
- Efectuar las operaciones lógicas
  - Crítico ejecutar AND antes que OR: forma normal disyuntiva
  - AND $\rightarrow$ Match (intersección) cosecuencial de postings
  - AND NOT $\rightarrow$ Resta cosecuencial de postings
  - OR $\rightarrow$ Merge (unión) cosecuencial de postings
  - OR NOT $\rightarrow$ Problemático: hay que devolver todos los docIDs de la colección (!) menos la lista de postings del término negado
- Optimización en las conjunciones múltiples
  - Usar conmutatividad para empezar por las listas de postings más cortas

Optimizaciones de las operaciones cosecuenciales de tipo “intersección”

- De postings en ránking por documentos, de postings y acumuladores, etc.
  - Especialmente efectivas en operaciones tipo “match” (intersección, and, etc.)
  - También cabe aplicarlas a la fusión en la construcción/actualización de índices, para hacer lecturas secuenciales más largas y menos seeks (el ahorro no es tanto)
- En operaciones compuestas, operar primero las listas cortas
- Saltarse elementos avanzando más rápido en la iteración cosecuencial
  - Alternativa al uso de heaps

1. Skip pointers
   - Se avanza en los postings de *n* en *n* (se necesitan skip pointers en la lista de postings)

para $k = 2$ listas

2. Búsqueda binaria
   - Puede compensar buscar los elementos de una lista (postings o índice parcial) en la otra, en lugar del avance cosecuencial, si una de las listas es mucho más pequeña que la otra
3. Galloping search
   - Avanzar en saltos de potencias de 2 en la lista más larga (equivale a un skip exponencial)
   - Después búsqueda binaria en el último segmento

API de los índices

- Diccionario
  - getPostings(término)
- Postings list
  - first, last, next/prev (para iterar), size (para *df*), insert (en tiempo de indexado)
- Posting
  - getDocID, getFrequency, getPositions…
- Información adicional
  - Dependiente del modelo, p.e. getDocModule para VSM
  - Dependiente de la UI, p.e. snippets, índice directo (términos de documento) para resaltar palabras, etc.

Búsqueda proximal

- Se trata de favorecer las apariciones de términos de la consulta cercanas entre sí dentro del documento, no sólo su frecuencia
- Función de ránking $score(d, q)$
  - Encontrar el conjunto $\mathcal{M}(d, q)$ de intervalos mínimos de secuencias de palabras en d que contienen todas las palabras de q
  - Utilizar el número y longitud de estos intervalos en la función de ránking
  - Por ejemplo, calcular un score de proximidad como:

$$score(d, q) = \sum_{[a,b] \in \mathcal{M}(d,q)} \frac{1}{b - a}$$

  - En la práctica matizar p.e. $b - a - |q| + 2$ para que no dependa de $|q|$
  - Combinar el score proximal con el score del modelo IR (implícitamente el proximal ya captura algo de la frecuencia, salvo que se tome el promedio por nº de intervalos)
- ¿Cómo determinar estos intervalos $\mathcal{M}(d, q)$?

Ejemplo

And by opposing end them: to die, to sleep q = to sleep

No more; and by a sleep, to say we end

The Heart-ache, and the thousand Natural shocks

That Flesh is heir to? 'Tis a consummation

Devoutly to be wished. To die to sleep,

To sleep, perchance to Dream; Aye, there' s the rub,

For in that sleep of death, what dreams may come,

When we have shuffled off this mortal coil,

Must give us pause.

Ejemplo

0 1 2 3 4 5 6 7 8
And by opposing end them: to die, to sleep

9 10 11 12 13 14 15 16 17 18
No more; and by a sleep, to say we end

19 20 21 22 23 24 25 26
The Heart-ache, and the thousand Natural shocks

27 28 29 30 31 32 33 34
That Flesh is heir to? 'Tis a consummation

35 36 37 38 39 40 41 42
Devoutly to be wished. To die to sleep,

43 44 45 46 47 48 49 50 51 52
To sleep, perchance to Dream; Aye, there' s the rub,

53 54 55 56 57 58 59 60 61 62
For in that sleep of death, what dreams may come,

63 64 65 66 67 68 69 70
When we have shuffled off this mortal coil,

71 72 73 74
Must give us pause.

q = to sleep

$$score(d,q) = \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{1}{2} + \frac{1}{10} = 5.6$$

a	b
$-\infty$	
7	8
14	15
41	42
42	43
43	44
44	46
46	56
	∞

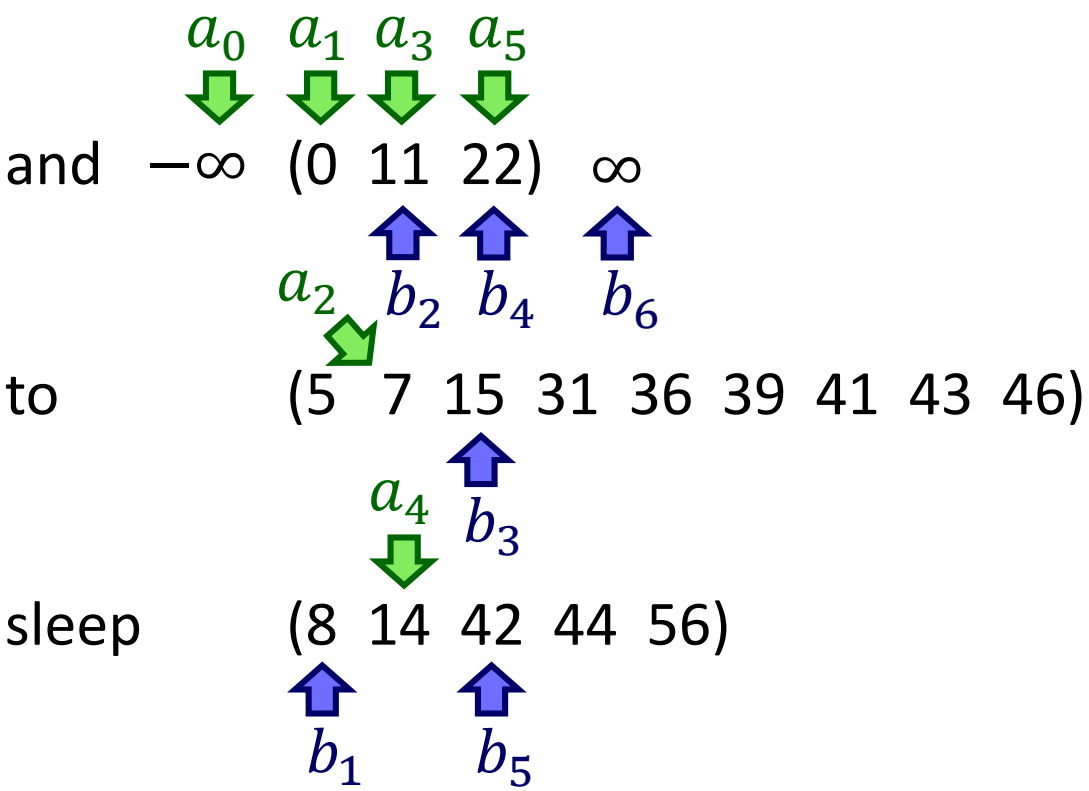
Cómo determinar los intervalos

- Usando las estructuras de índice posicional (postings con posiciones)
- Para cada documento que aparece en las listas de postings de <u>todos</u> los términos de la consulta

1. Obtener $\mathcal{P}_d \equiv$ listas (una por término de consulta) de posiciones para d (Interesa por tanto listas de postings ordenadas por docID, ¿por qué?)
2. $a \leftarrow -\infty$
3. $b \leftarrow \max \left\{ \min\left(l \cap (a, \infty)\right) \,\middle|\, l \in \mathcal{P}_d \right\}$
4. Si "$b = \infty$" fin // O bien antes del paso 3, $a = \min \{ \text{long}(l) \mid l \in \mathcal{P}_d \}$
5. $a \leftarrow \min \{ \max(l \cap (0, b]) \mid l \in \mathcal{P}_d \}$
6. Devolver $[a, b]$, volver a 3

Ejemplo

q = and to sleep



a	b
$-\infty$	
0	8
7	11
11	15
14	22
22	42
	∞

$\mathcal{P}_d$, postings de q_i con d

Intervalos mínimos: $[0,8]$ $[7,11]$ $[11,15]$ $[14,22]$ $[22,42]$

$$score(d,q) = \frac{1}{7} + \frac{1}{3} + \frac{1}{3} + \frac{1}{7} + \frac{1}{19} = 1.005$$

// Parámetros de entrada:
// Longitud de la consulta: $|q|$
// Listas (arrays) de posiciones, array de longitud $|q|$: int posList[$|q|$][]
// Nota: expresamos como posList[j][p] $= \infty$ cuando p excede la longitud de posList[j]
// Variables locales:
// Punteros de iteración, uno por lista de posiciones: int p[$|q|$]
// Extremos de los intervalos: a, b

score ← 0
for j=1 to $|q|$ do p[j] ← 0
$b \leftarrow \max_j$posList[j][p[j]]

while $b < \infty$ do
    $i \leftarrow 0$
    for j=1 to $|q|$ do
        // Avanzar todas las listas hasta posición b
        while posList[j][p[j]+1] $\leq b$ do p[j]++
        // Nos queremos quedar en i con la posición más baja de las $|q|$ listas
        if posList[j][p[j]] < posList[i][p[i]] then $i \leftarrow j$
    $a \leftarrow$ posList[i][p[i]]
    score ← score + 1/(b - a - $|q|$ + 2)
    $b \leftarrow$ posList[i][++p[i]]
return score

Otras señales de relevancia

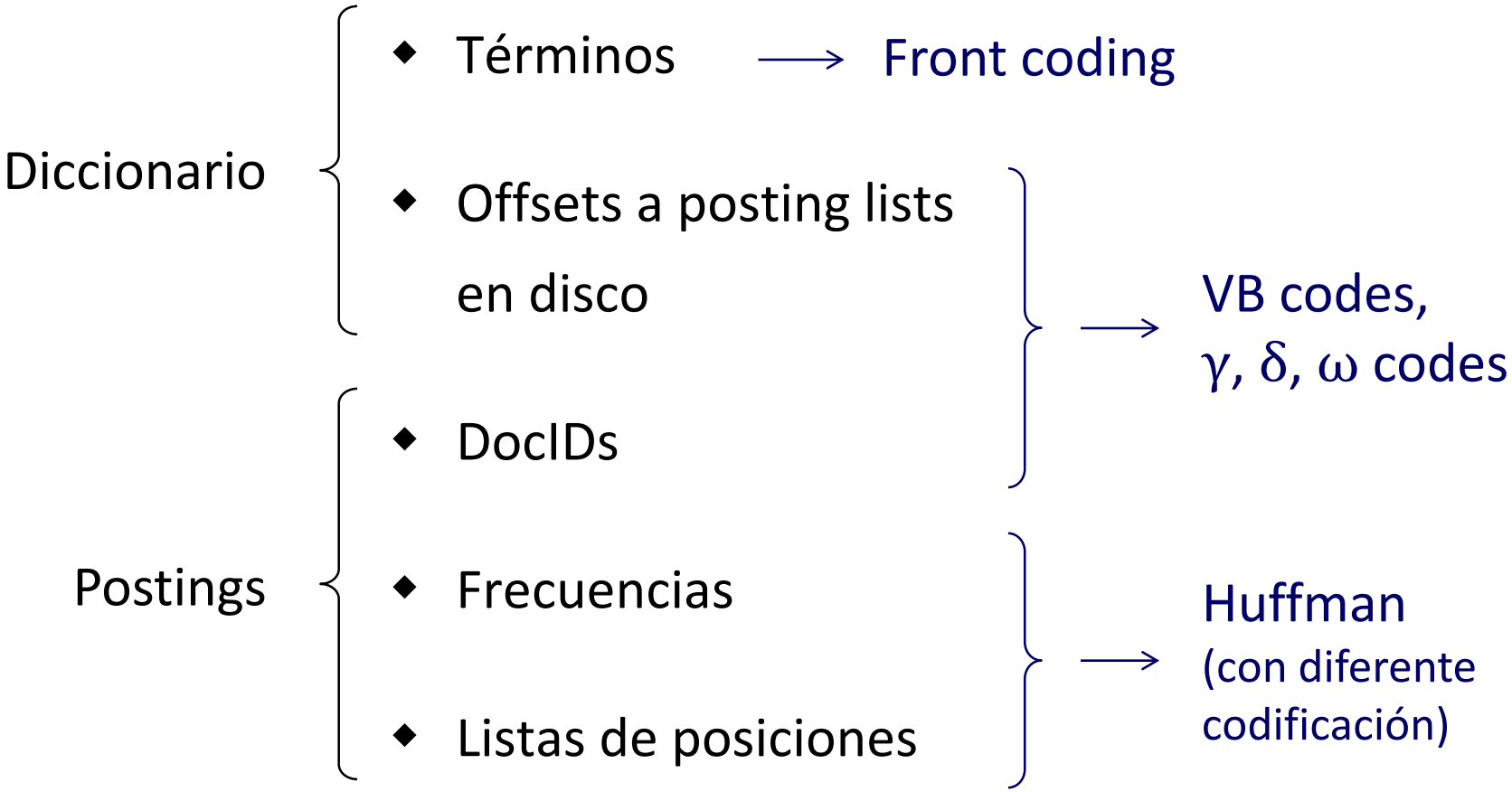
- En la práctica los motores combinan el score del modelo IR con otros scores
  - Varios modelos IR: VSM, probabilísticos, etc.
  - Calidad genérica del documento, p.e. PageRank, freshness, tiempo de respuesta del servidor, etc.
  - Propiedades de la aparición de los términos: proximidad, énfasis tipográfico (tamaño de letra, negrita, títulos… p.e. en html), etc.
- Campos
  - P.e. título del documento, anchor text, autor, metadatos, secciones, etc.
  - Se puede ponderar de forma diferente la aparición de palabras en cada campo (p.e. más significativo en el título que en el cuerpo)
  - Índices por campos, o id de campo en la posición
- Técnicas de machine learning para combinar scores
- Clicks!!
- P.e. Google habla de “cientos de señales”

  (ver https://www.google.com/search/howsearchworks/algorithms)

Compresión de índices

- Imprescindible para hacer viable la búsqueda a gran escala
  - P.e. Google menciona índice > 100PB
- La compresión permite ahorrar espacio de disco
  - Y en su caso también de RAM
  - Son comunes tasas de compresión del orden de 4:1
- Pero además acelera las operaciones de búsqueda en índice
  - El tiempo de decodificación (si es razonablemente eficiente) se compensa con la reducción del coste de lectura en disco (se leen menos bytes)
  - O incluso hay técnicas de recorrido de datos sin descomprimir

¿Qué se puede comprimir en un índice?



Breve introducción a compresión

- Comprimir: re-codificar información en una representación que ocupe menos bits
- ¿Cómo? En esencia, eliminando redundancia
- Ejemplo:

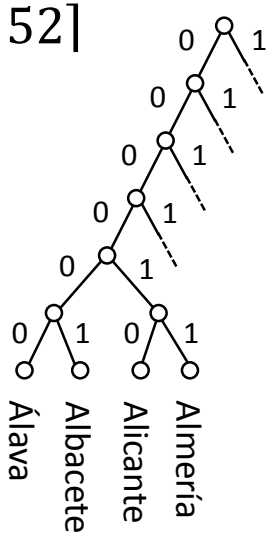
	Varios bytes		1 *byte*
52 símbolos	Álava	→	1
	Albacete	→	2
	Alicante	→	3
	Almería	→	4
	…		

Breve introducción a compresión

- Comprimir: re-codificar información en una representación que ocupe menos bits
- ¿Cómo? En esencia, eliminando redundancia
- Ejemplo:

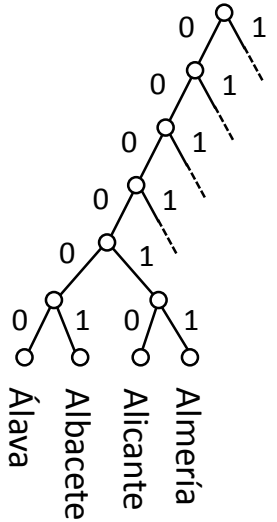
52 símbolos

Varios bytes		1 *byte*		6 *bits*	$\rightarrow$ $2^6 = 64$ valores diferentes
Álava	$\rightarrow$	1	$\rightarrow$	000000	$6 = \lceil \log_2 52 \rceil$
Albacete	$\rightarrow$	2	$\rightarrow$	000001	
Alicante	$\rightarrow$	3	$\rightarrow$	000010	
Almería	$\rightarrow$	4	$\rightarrow$	000011	
…					

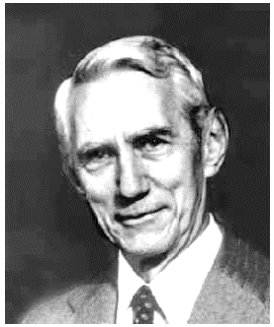


Compresión orientada a símbolos

- Alfabeto $\mathcal{S}$ de símbolos
  - Caracteres (bytes), números enteros, cadenas de bytes (palabras, n-gramas), pixels de color, vocabulario controlado (p.e. provincias)…
- Codificación
  - Mapping del alfabeto a secuencias de bits $c : \mathcal{S} \rightarrow \bigcup_{n=1}^{\infty} \{0,1\}^n$
  - Se puede ver como un árbol binario a cuyos nodos se asignan símbolos
- Mensaje: información a comprimir
  - Secuencia de símbolos $\mathcal{M} \in \bigcup_{n=1}^{\infty} \mathcal{S}^n$



Principios de compresión



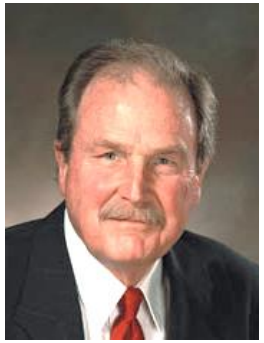
Claude E. Shannon
(1916-2001)

- Idea *naïf*: códigos de longitud fija, la mínima suficiente
  - k bits son suficientes para $2^{k-1} < |\mathcal{S}| \leq 2^k$
  - Es decir $k = \lceil \log_2 |\mathcal{S}| \rceil$
- Pero no hace falta que todos los códigos tengan la misma longitud
  - Usar menos bits para los códigos más frecuentes, i.e. interesa que la codificación cumpla $p(s_1) > p(s_2) \Rightarrow \text{long}\big(c(s_1)\big) \leq \text{long}\big(c(s_2)\big)$
  - La eficacia de la compresión se puede expresar por:
    - Tasa de compresión: bits de $\mathcal{M}$ comprimido / bits de $\mathcal{M}$ sin comprimir
    - Longitud promedio de código: $\text{avg}_{s \in \mathcal{M}} \text{long}\big(c(s)\big)$
  - Teorema de Shannon: $\text{avg}_{s \in \mathcal{M}} \text{long}\big(c(s)\big) \geq -\sum_{s \in \mathcal{S}} p(s) \log_2 p(s) = H_p(\mathcal{S})$
  - Cuanto más sesgada la distribución, más margen de compresión

Métodos de compresión: tipos

- Con/sin pérdida de información (lossy vs. non-lossy)
- Codificación estática vs. semi-estática vs. adaptativa
  - Estática: la codificación es independiente de los datos a comprimir
  - Semi-estática: la codificación depende de los datos
    - Se escanean los datos (p.e. para calcular frecuencias de los símbolos), se generan los códigos, y se vuelven a leer los datos para comprimir
    - El mapping se guarda junto con los datos comprimidos
  - Adaptativa: los códigos se modifican a medida que se comprime
    - P.e. se actualizan las frecuencias de los símbolos leídos y se ajustan los códigos sobre la marcha
- Codificaciones libres de prefijo
  - Ningún código es prefijo de otro
  - Permiten decodificar sin ambigüedad

Métodos: códigos de Huffman



David A. Huffman
(1925-1999)

- Semi-estático
- Libre de prefijos
- Propiedades de optimalidad para distribuciones arbitrarias
  - Mejor cuanto más sesgadas
- Ejemplo…

Métodos: códigos de Huffman (cont)

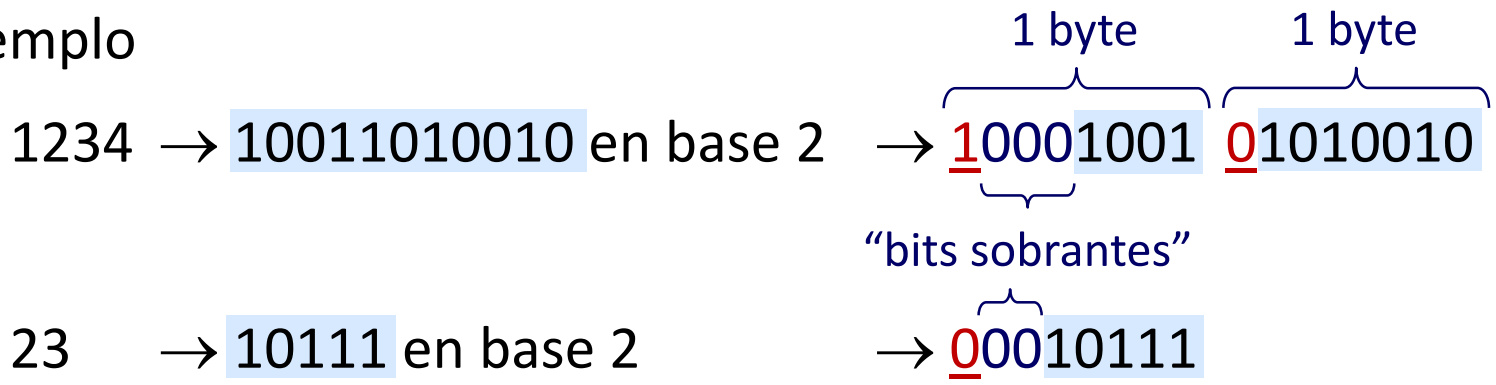
Adecuado para comprimir…

- Caracteres de los términos del diccionario
  - Aunque hay otra alternativa mejor: front coding
- Postings lists
  - Frecuencias (si se almacenan como *tf*, no se pueden comprimir)
  - Posiciones: aplicar códigos diferencia (relativos en cada lista)
  - Pero no los docIDs, pues su distribución es poco sesgada,

    y con códigos diferencia no se consiguen muchas más repeticiones

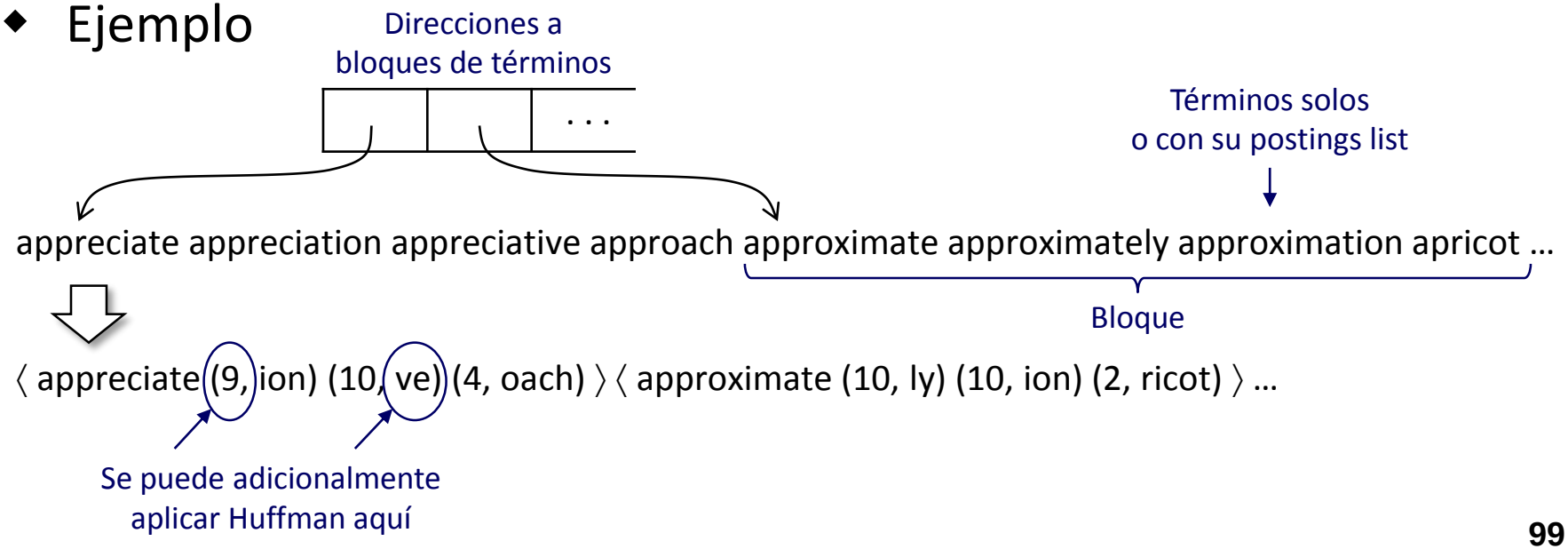
Métodos: código byte variable (VBC)

- Adecuado para los docIDs, que varían en un rango muy alto (p.e. en Web hasta 5 bytes)
- Números almacenados en "longitud variable" (en nº de bytes)
  - Se utiliza el primer bit de cada byte para indicar si es o no el último byte
    Los otros siete bits forman parte de la codificación del número
  - El "sobrante de bits" se sitúa en el primer byte de la cadena de bytes
- Produce mayor compresión cuando los valores son generalmente bajos pero aparecen valores altos ocasionalmente
- Ejemplo



Métodos: front coding

- Para compresión de diccionario (si está ordenado por término)
- Utilizar una estructura de diccionario con direccionamiento por bloques de términos (o términos + postings)
- Partiendo del primer término del bloque, se almacena la diferencia en sufijo para cada término respecto del anterior
- Ejemplo



Front coding: ejemplo en detalle

Diccionario sin comprimir

Términos

…	…
combine	234
commerce	5
commercial	76
commercialization	102
commercialize	173
communicate	34
communication	150
community	215
…	…

Offsets a postings lists

(P.e. bloques de 4 términos)

Diccionario comprimido

```
0 7 combine 234 3 5 merce 5 7 3 ial 76 10 7 ization 102 |
0 13 commercialize 173 4 7 unicate 34 10 3 ion 150 7 2 ty 215
```

Longitud prefijo *Sufijo* *Offset postings list*

Métodos de compresión

- VBC es libre de prefijo y libre de parámetros (no tiene en cuenta las características de los datos ni necesita guardar la codificación)
- Hay alternativas más elaboradas a VBC: códigos γ, δ, y ω
  - Ajustan a nivel de bit (métodos "desalineados"), codifican la longitud de las cadenas de bits mediante cadenas de 0's
  - VBC se codifica y decodifica muy rápido, los métodos desalineados son más costosos: las iteraciones por bits son más largas que por bytes
- Se suele usar renumeración de los docIDs p.e. para que las diferencias pequeñas sean más frecuentes en las postings lists
  - Asignar docIDs a documentos "similares" (en tamaño, fecha, URL…) bajo el supuesto de que es probable que compartan muchos términos
  - Esos términos tendrán así diferencias más pequeñas entre docIDs

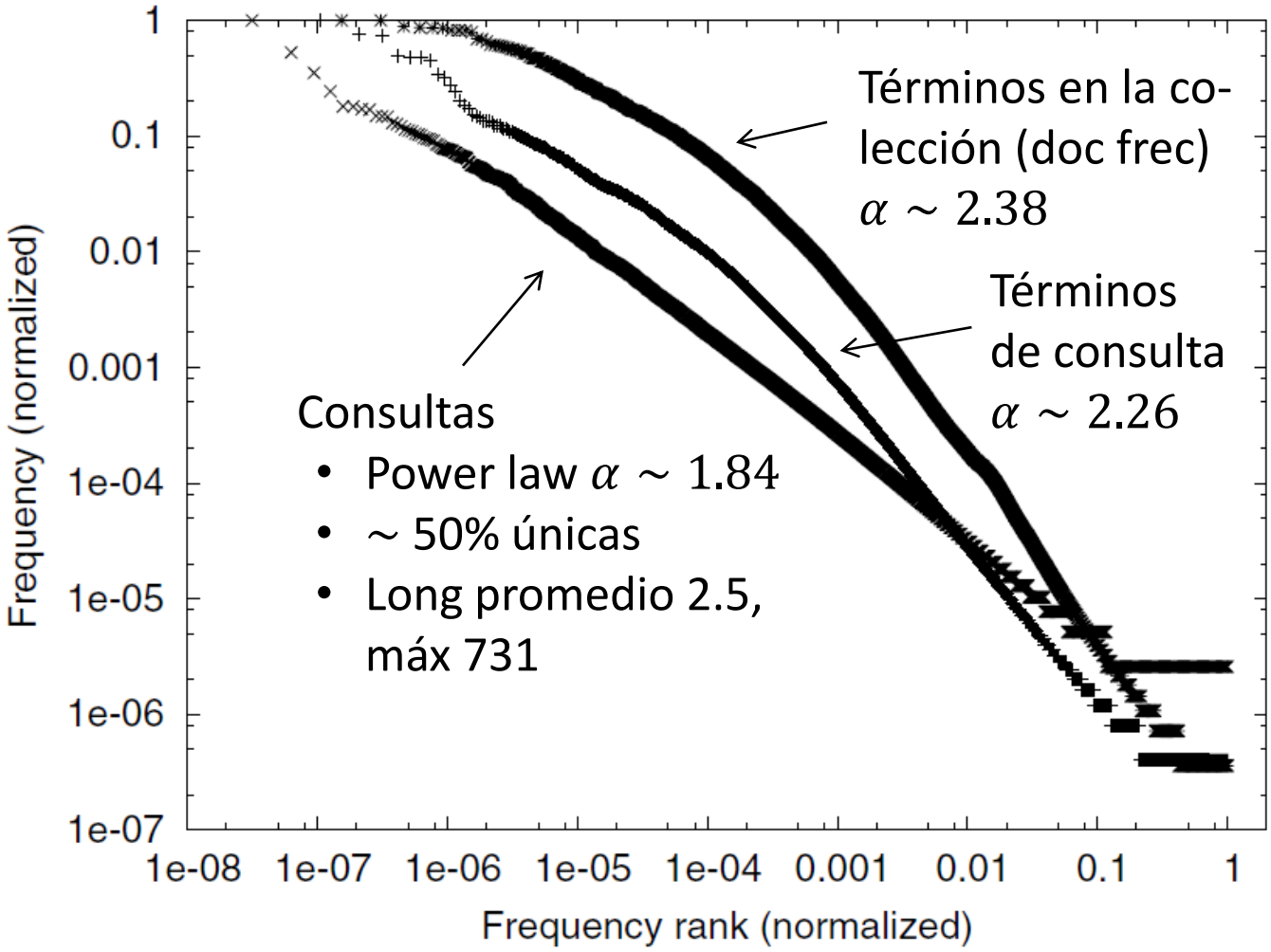
Otras optimizaciones: paralelización

- Los buscadores Web modernos necesitan paralelización masiva e índices distribuidos para su viabilidad
- Índices distribuidos
  - Cada índice indexa una partición de los documentos de la colección
  - Las consultas se envían a todos los índices y se fusiona el resultado
  - Se necesitan centralizar algunas estadísticas: *idf*, PageRank…
  - Es posible también particionar por palabras, de modo que cada nodo procesa un subconjunto de las listas de postings para una consulta
- Procesamiento paralelo
  - Muy diversas fórmulas de paralelización de operaciones completas, parciales, con datos distribuidos, compartidos y/o replicados (p.e. MapReduce)
  - El balanceo de carga es comúnmente uno de los puntos a resolver

Otras optimizaciones: cache

- Cache de resultados finales o intermedios
  - Guardar en memoria los resultados de consultas frecuentes
  - Y/o intersecciones de postings utilizadas frecuentemente
  - Y/o listas de postings de términos frecuentes
- Estrategias cache
  - Substituir el menos reciente
  - Substituir el menos frecuente
  - Pre-fetching: anticipar la necesidad de caching p.e. con un log de consultas, teniendo en cuenta incluso las fluctuaciones horarias de actividad
- Los mecanismos de caching se usan ampliamente en los buscadores Web y son básicos para su eficiencia
- De igual forma que se desarrollan optimizaciones teniendo en cuenta las estadísticas de la colección particular, se analiza el espacio de consultas, sus estadísticas, distribución, tendencias, etc.

Frecuencia de las consultas



Ejemplo: 1 año de consultas yahoo.co.uk

- 2.8M documentos
- 6.5M términos
- 2.100M tokens

Ver Google trends

R. Baeza-Yates et al. The impact of caching on search engines. SIGIR 2007

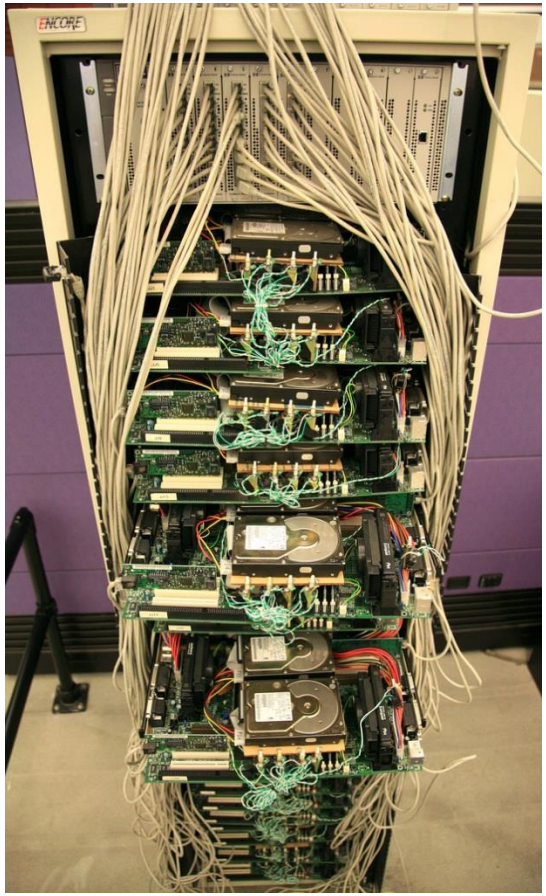
Frecuencia de las consultas (cont)

- Responder a las consultas de la cabeza de la distribución (las más frecuentes) es generalmente más fácil
  - Casi todos los buscadores responden bien a estas consultas
- Es más difícil acertar con las consultas infrecuentes (long-tail)
  - La calidad de respuesta en la cola de consultas puede hacer que un buscador se distinga de la competencia
- Los grandes logs de consultas son un reflejo de actividad humana, una gran mina de información
  - Más allá del propósito de la búsqueda
  - Investigación sociológica, criminológica, etc.

Escala

Primer servidor producción Google 1998

(26 millones de páginas Web)



- 2 x Pentiums II + Sun + RS 6000
- 8 procesadores 200-300MHz
- 1.3GB RAM
- ~380GB disco

Escala Web…

Google Data Centers

(Cientos de miles de millones de documentos, imágenes, vídeo, logs, user data…)