DATA20021

University of Helsinki, Department of Computer Science

Information Retrieval

Lecture 4: Index Construction

Simon J. Puglisi

puglisi@cs.helsinki.fi

Spring 2020

Today…

16.1: Introduction to Indexing

- Boolean Retrieval model
- Inverted Indexes

21.1: Index Compression

- unary, gamma, variable-byte coding
- (Partitioned) Elias-Fano coding (used by Google, facebook)

23.1: Index Construction

- preprocessing documents prior to search
- building the index efficiently

28.1: Web Crawling

- getting documents off the web at scale
- architecture of a large scale web search engine

30.1: Query Processing

- scoring and ranking search results
- Vector-Space model

MORGAN & CLAYPOOL PUBLISHERS

Scalability Challenges in Web Search Engines

B. Barla Cambazoglu
Ricardo Baeza-Yates

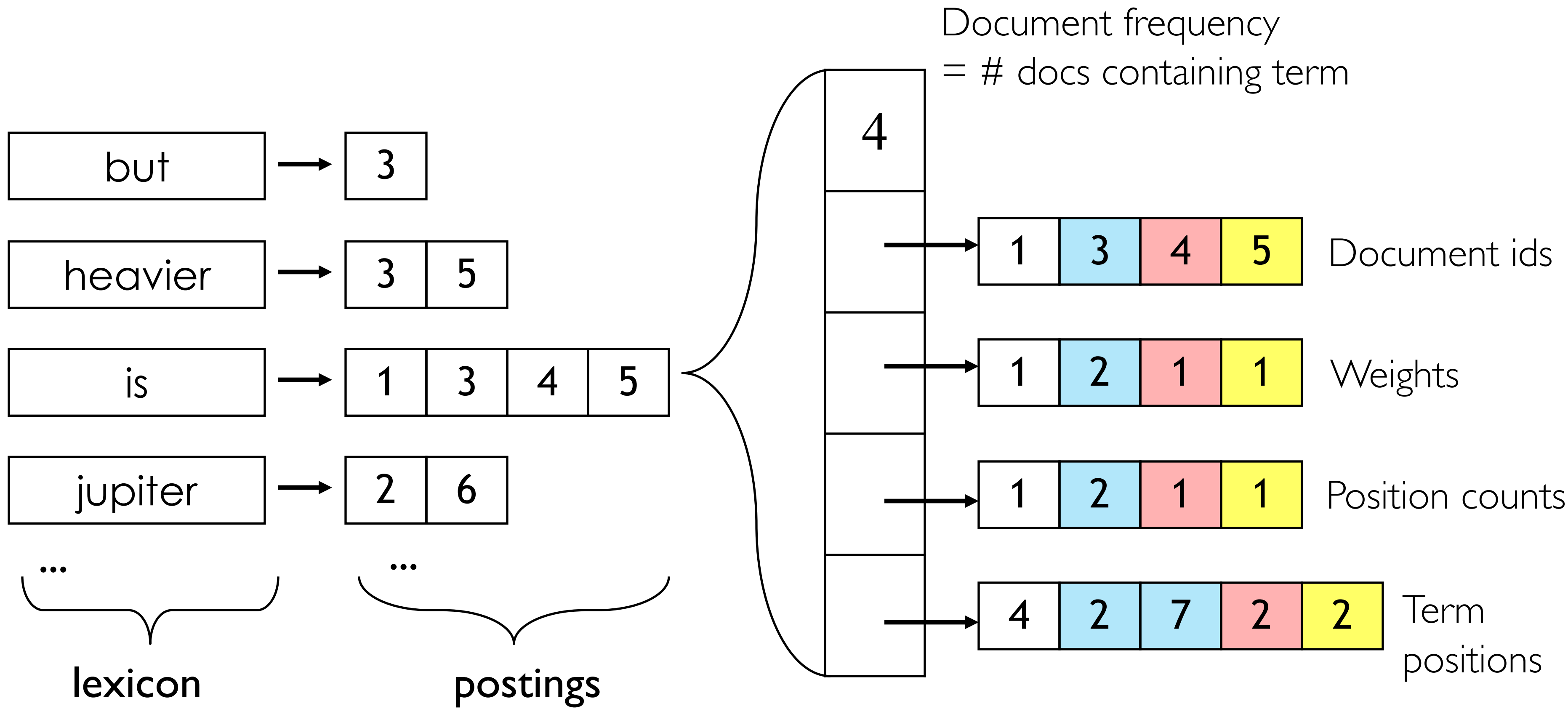
Synthesis Lectures on Information Concepts, Retrieval, and Services

Gary Marchionini, *Series Editor*

Research Search Engines…

- Terrier: http://terrier.org/
  - Lots of features, Java, built and maintained by U. Glasgow
- Anserini: https://github.com/castorini/Anserini
  - Lots of features, focus on reproducible experimentation, built on top of Apache's Lucene, built and maintained by U. Waterloo
  - Heavy use in research due to excellent documentation for replicating experiments
- Indri/Galago: https://www.lemurproject.org/indri/ and https://www.lemurproject.org/galago.php
  - Indri is the C++ predecessor to Galago (Java), maintained by U. Mass
- PISA: https://github.com/pisa-engine/pisa
  - Efficiency focus, C++, NYU and RMIT, but origins at the U. Pisa

Inverted Index



Terms, Parsing, Stemming, …

Terms and Documents

- We’ve been looking at a simple Boolean retrieval system
- Our assumptions have been many, including:
  - We know what a document is
  - We know what a term is
- Both issues can be complex in reality…

Decisions, decisions…

- We have pulled a page off the web…
  - What format is the page in? Is there text in it?
  - What language is the text in? (if there is text)
  - What constitutes a document?
    - A page, a paragraph, a sentence?
    - Very long documents (books) pose problems

- Once we have worked all that out…

- What constitutes a term?
  - How do we define and process the vocabulary of terms of a collection?

Jargon

- Word – A delimited string of characters as it appears in the text
- Term – A “normalized” word
  - case
  - morphology
  - spelling, et c.
  - an equivalence class of words
- Token – An instance of a word or term occurring in a document

Inverted Index Construction

- Input:

Friends, Romans, contrymen.	So let it be with Caesar.	...

- Output:

friend	roman	countryman	so	...

- Each token is a candidate for a postings entry.
- What are valid tokens to emit?

One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333

- Older IR systems didn’t index numbers, but generally it’s a useful feature.

Chinese: No whitespace

莎拉波娃现在居住在美国东南部的佛罗里达。今年4月9日，莎拉波娃在美国第一大城市纽约度过了18岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

Ambiguous Segmentation in Chinese

These two characters can be treated as one word meaning ‘monk’ or as a sequence of two words meaning ‘and’ and ‘still’.

Other cases of no whitespace

- Compounds in Dutch and German
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can't hear very well)
- Finnish, Swedish, Greek, Urdu, many other languages

Arabic Script : Bidirectionality

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

$\leftarrow$ $\rightarrow$ $\leftarrow$ $\rightarrow$ $\leftarrow$ START

'Algeria achieved its independence in 1962 after 132 years of French occupation.'

Arabic Script : Bidirectionality

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

← → ← → ← START

'Algeria achieved its independence in 1962 after 132 years of French occupation.'

Unicode allows direction to be encoded

Back to English…

Normalization

- Normalization is the process of canonicalizing tokens so that matches occur despite superficial differences in character sequences
  - E.g., We want to match U.S.A. and USA
- Need to normalize terms in indexed text as well as query terms into the same form.
- Most commonly: implicitly define equivalence classes of terms
  - window, windows, Windows → window
- Alternatively: do asymmetric expansion
  - window → window, windows
  - windows → Windows, windows
  - Windows (no expansion)

Case folding

- Convert all letters to lower case
- Possible exceptions: capitalized words in mid-sentence
  - MIT vs. mit
  - Fed vs. fed
- It's often best to lowercase everything since users will use lowercase regardless of correct capitalization

Normalization: other languages

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence "ae")
- Most important criterion: How are users likely to write their queries for these words?
- Normalization and language detection interact
  - PETER WILL NICHT MIT. → MIT = mit
  - He got his PhD from MIT. → MIT ≠ **mit**

More equivalence classing

- Soundex: phonetic equivalence
  - Tchebyshev = Chebysheff
- Thesauri: semantic equivalence
  - car = automobile

Stop words

- stop words are extremely common words seemingly of little value in helping select documents matching a user need
- Examples:
  - a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for some phrase queries, e.g. “The Smiths”; “Romeo and Juliet”
- Most web search engines index stop words.

Lemmatization

- Reduce inflectional/variant forms to base form
  - Example: am, are, is → be
  - Example: car, cars, car's, cars' → car
  - Example: the boy's cars are different colors → the boy car be different color
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).
- Inflectional morphology (cutting → cut) vs. derivational morphology (destruction → destroy)
- Lemmatization can be hard to do well, so…

Stemming

- Stemming is a (crude) heuristic process that chops off the ends of words in the hope of achieving what "principled" lemmatization attempts to do with a lot of linguistic knowledge.

- Language dependent

- Example: automate, automatic, automation all reduce to automat

Porter Algorithm

- Most common algorithm for stemming English
  - Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
  - Sample command: Delete final ement if what remains is longer than 1 character
  - replacement → replac
  - cement → cement
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

Porter stemmer: a few rules

Rule	Example
SSES → SS	caresses → caress
IES → I	ponies → poni
SS → SS	caress → caress
S →	cats → cat

Three stemmers: a comparison

- Sample text: 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

- 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

- 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

- 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

Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others
- Porter Stemmer equivalence class oper contains all of: *operate operating operates operation operative operatives operational*
- Queries where stemming hurts: “operations AND research”, “operating AND system”

What does Google do?

- No stop words (they index every term)
- Normalization
- Tokenization
- Lowercasing
- No Stemming (very probably)
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers

Inverted Index Construction…

- In most commercial indexing systems, the inverted index is constructed and deployed periodically
  - Mostly due to practical issues involved in keeping the index constantly right up to date (see later)

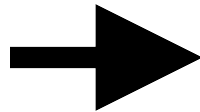
- Queries continue to be evaluated over an older version of the index until a new index is deployed

- Important to keep index deployment cycle short: a stale index may harm search quality
  - E.g., results containing deleted pages or missing recent but relevant ones

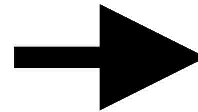
- Therefore we need efficient construction methods

Earlier...

term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i’	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2



term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i’	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



term	doc freq	inv list
ambitious	1	2
be	1	2
brutus	2	1,2
capitol	1	1
caesar	2	1,2
did	1	1
enact	1	1
hath	1	1
I	2	1
in	1	1
it	1	2
julius	1	1
killed	2	1
let	1	2
me	1	1
noble	1	2
so	1	2
the	2	1,2
told	1	2
you	1	2
was	2	1,2
with	1	2

Sort-based Index Construction

- As we build index, we parse docs one at a time
- The final postings for any term are incomplete until the end
- At ~12+ bytes per postings entry, demands a lot of memory for large collections
- 5.5 billion postings in GOV2 collection (a tiny part of the .gov domain) → 60GB of RAM
  - (We could actually do GOV2 in memory on a decent server)
- Thus: we need to store intermediate results on disk

Same algorithm for disk?

- Can we use the same index construction algorithm for larger collections, but by using disk instead of memory?

- Not directly: Sorting T = billions of records on disk is too slow if not done carefully – too many disk seeks
  - Random accesses to disk are at least 10 times slower than random accesses to RAM
  - Different model of computation:
    - scans are OK (the hardware prefetches the next chunk of data)
    - transferring one large chunk of data from disk to memory is faster than transferring many small chunks

- We need an external memory sorting algorithm
  - We'll adapt merge sort for this purpose…

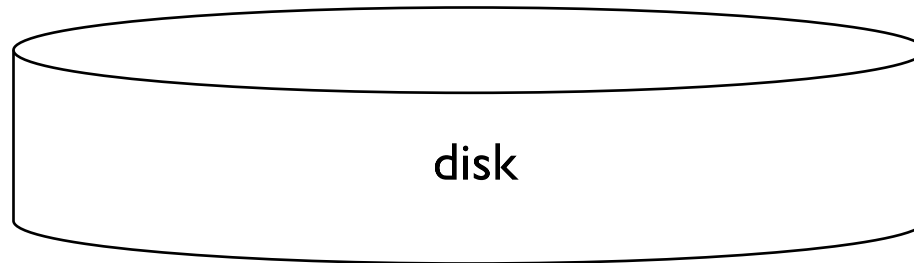
"External" sorting algorithm (using few disk seeks)

- For each term create a (termID, docID) pair
- Define a *block* to consist of 500,000,000 such postings
  - We can easily fit that many postings into memory (~6GB)
  - We will have ~10 such blocks for GOV2.
- Basic idea of algorithm:
  1. Make sorted runs: read in each block, sort it, write it to disk
  2. Repeatedly merge adjacent pairs of sorted blocks until only one remains
  3. Collect like terms as before to arrive at final postings lists

Setup

1. All is quiet…

Doc1	Doc2	Doc3	Doc4	Doc5	…
Julius Caesar	Brutus killed	Noble Brutus	Caesar with	…	…



2. Parse collection (+ stem, etc), create (term,docID) pairs

julius d1 caesar d1 brutus d2 killed d2 noble d3 brutus d3

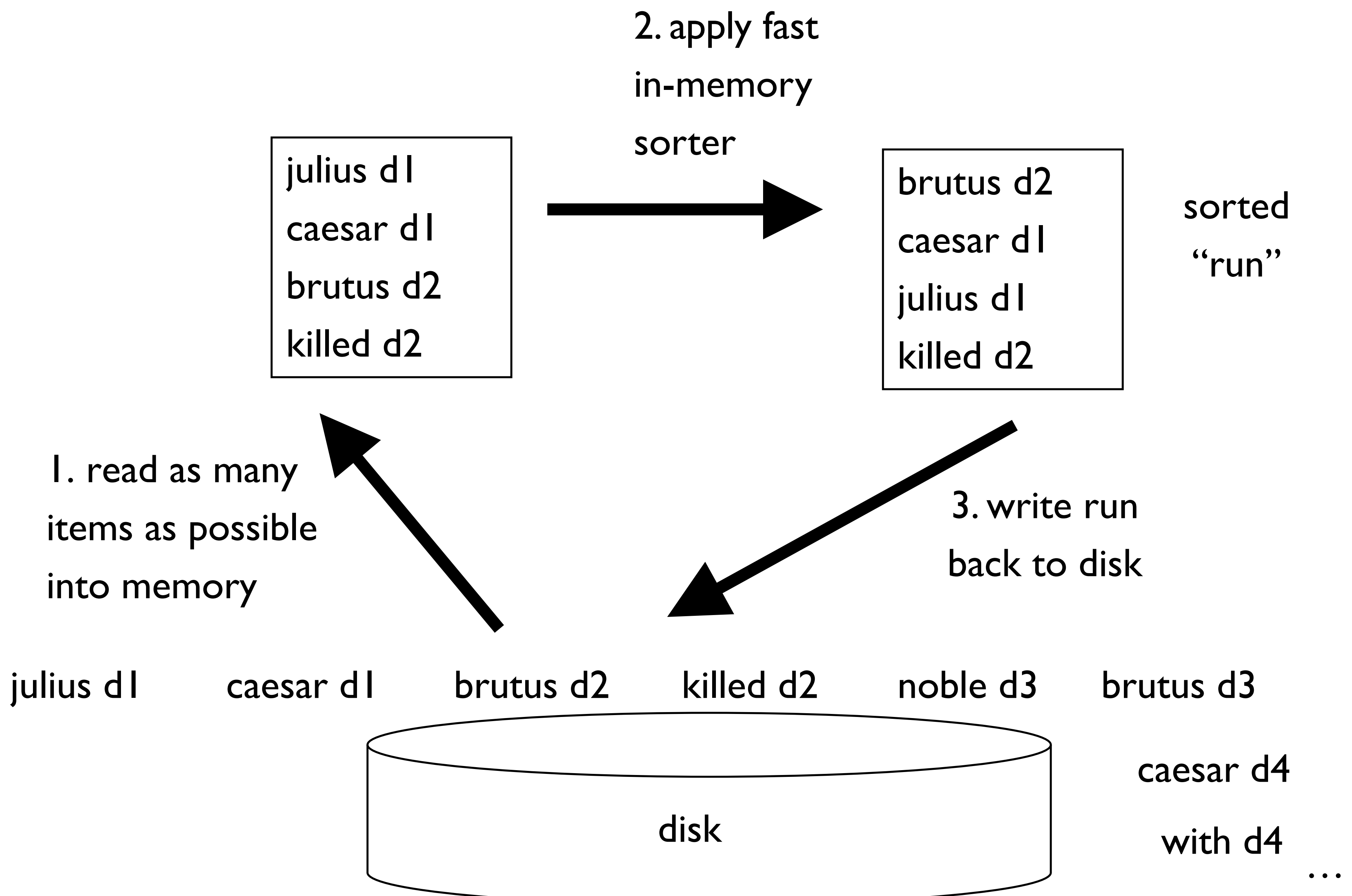
disk

caesar d4

with d4

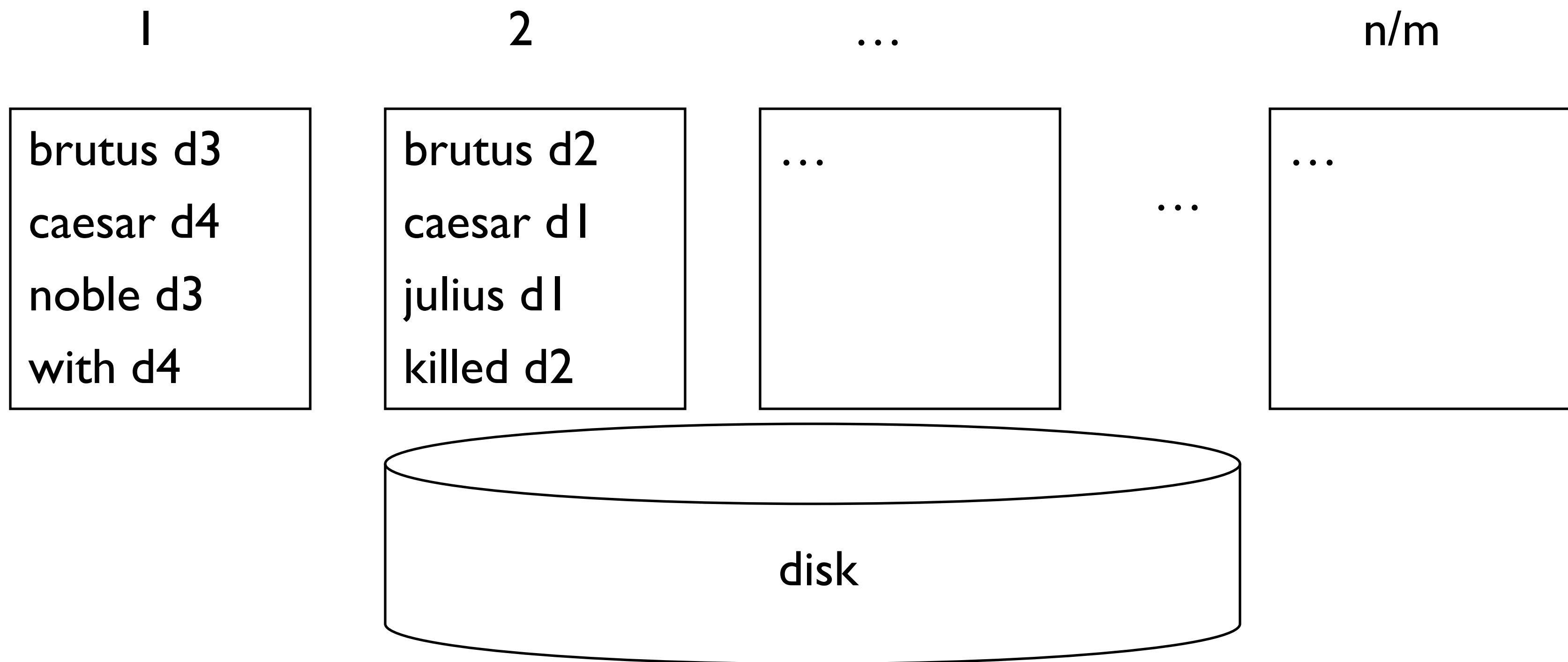
…

Make sorted runs

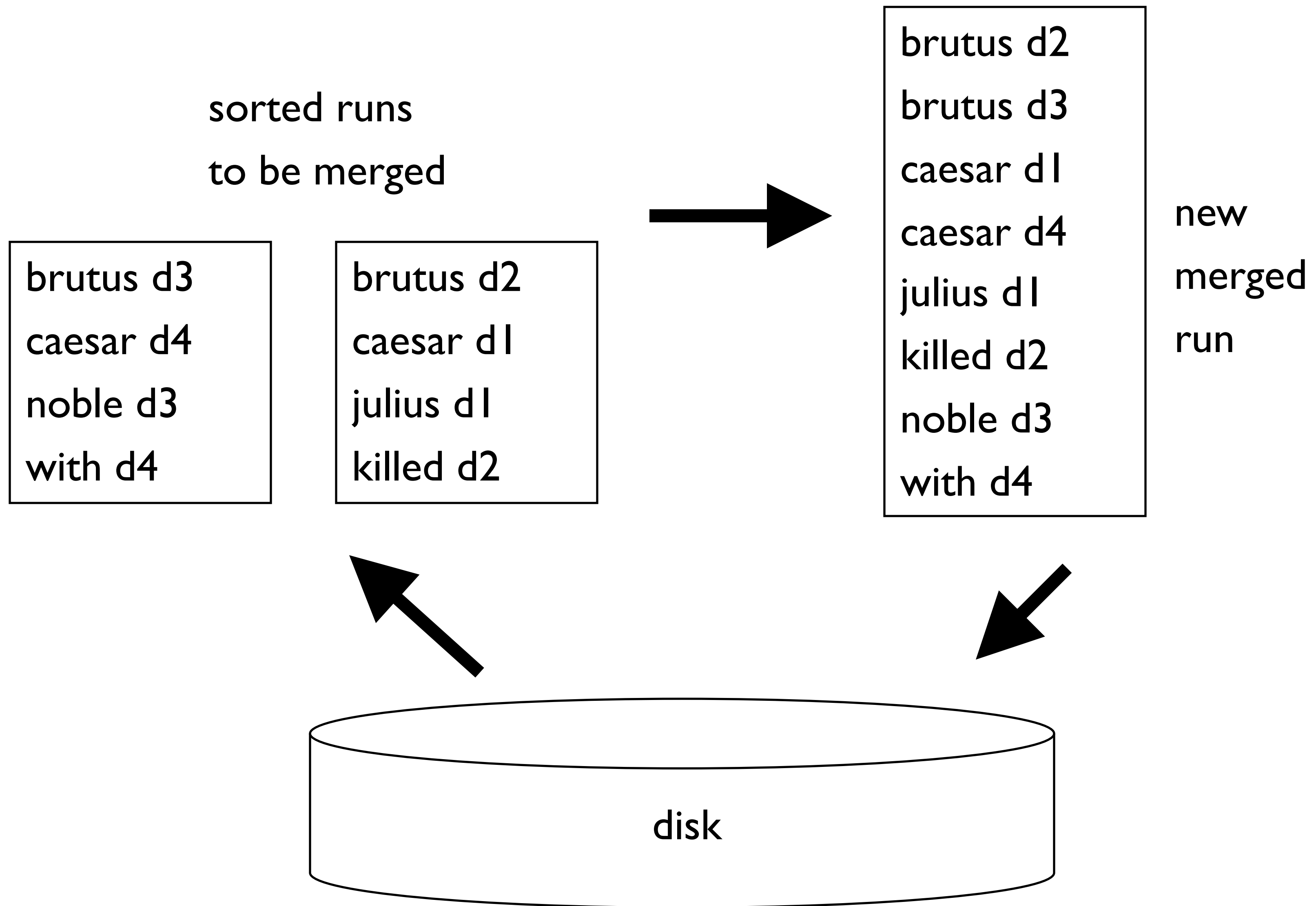


End of run creation phase…

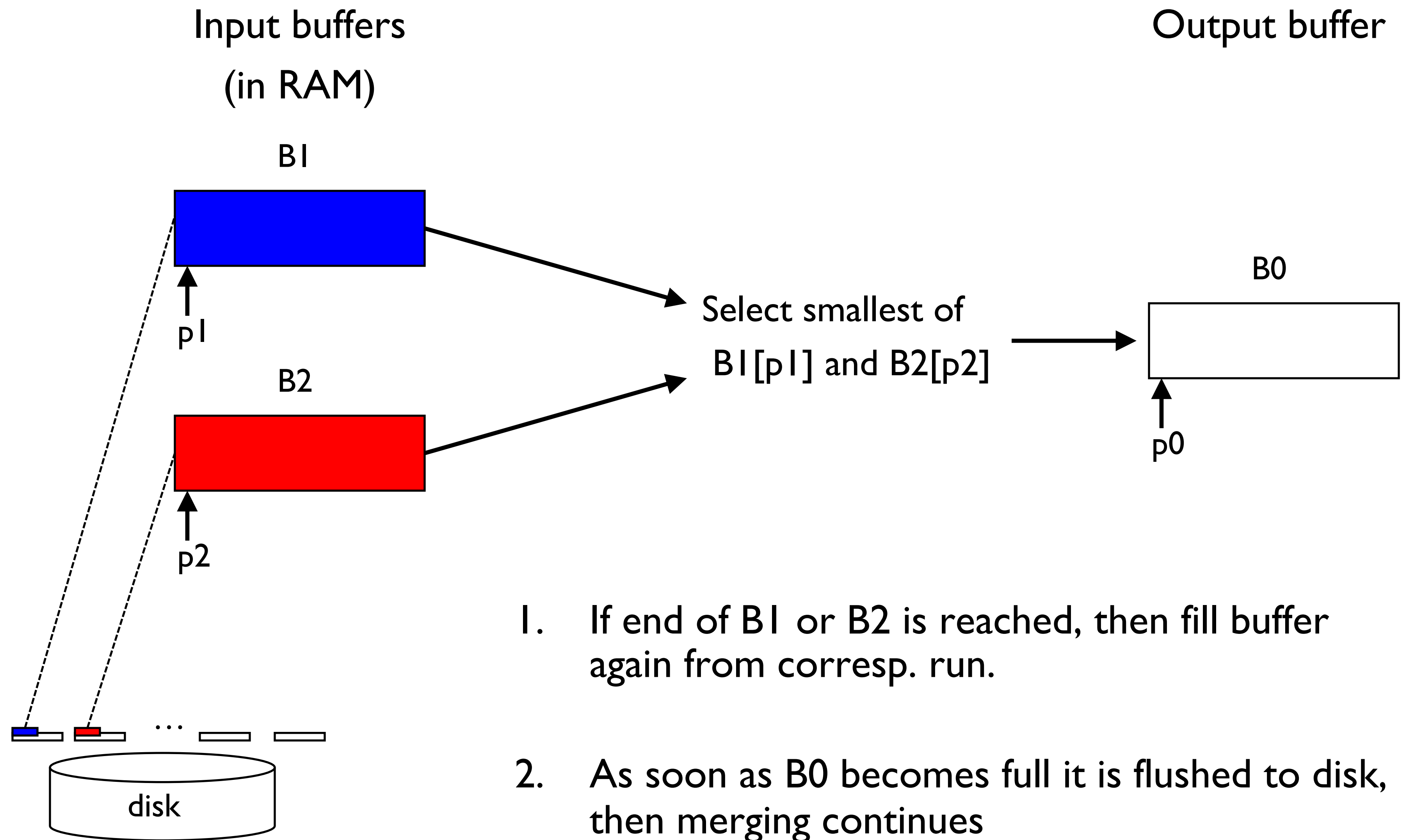
- We now have n/m sorted runs, where n is the number of terms in the collection, and m is the number of term,docID pairs we can hold in memory



Merging two runs



Two way merge



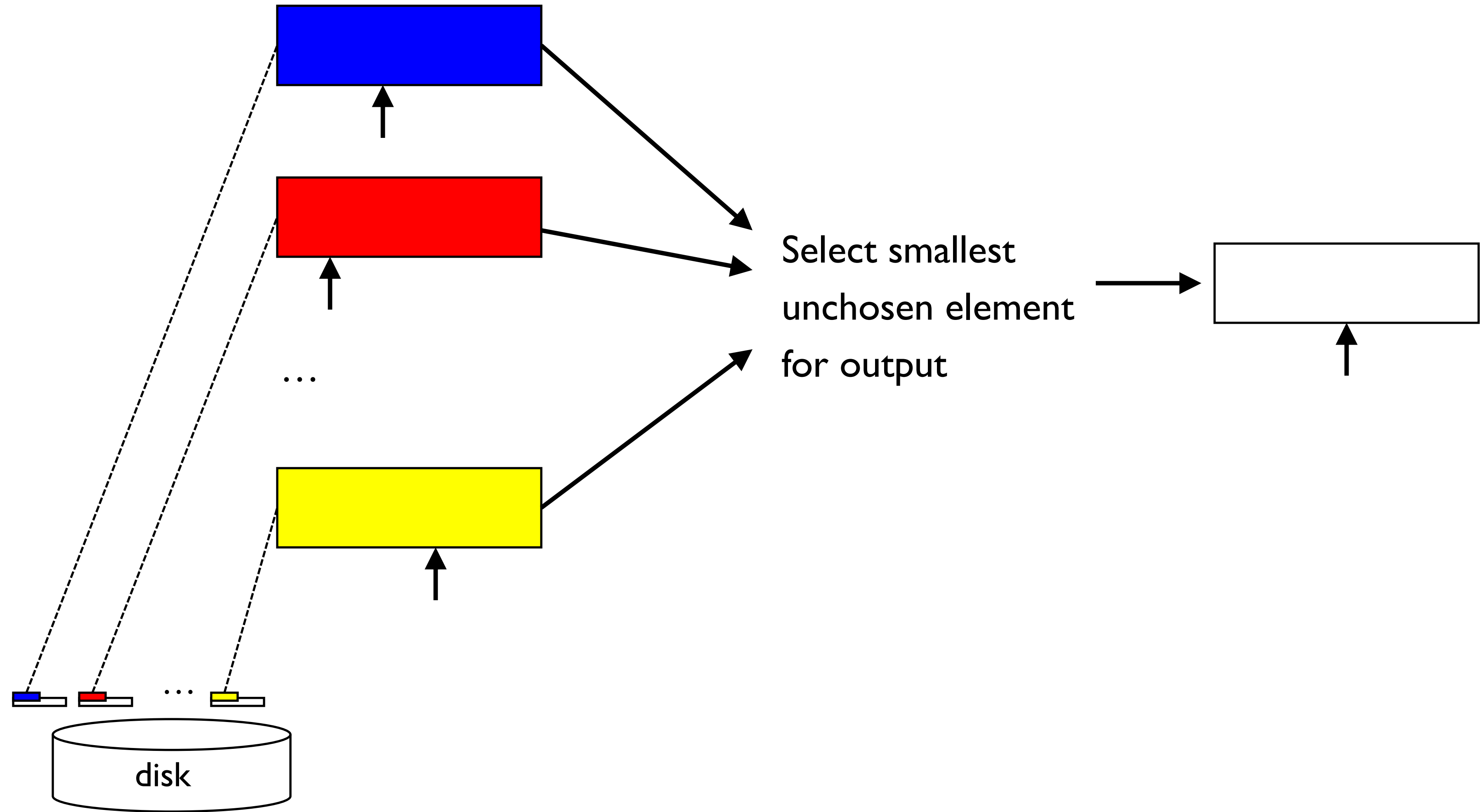
1. If end of B1 or B2 is reached, then fill buffer again from corresp. run.

2. As soon as B0 becomes full it is flushed to disk, then merging continues

Two-way merge

- Two way merge reduces the number of runs we have by a factor of two (ie. half) each round
  - For n items, we thus require $\log_2 n$ rounds
  - Each round we push n items through memory
- There are a ways to speed up this basic idea
  - We'll briefly look at two of them

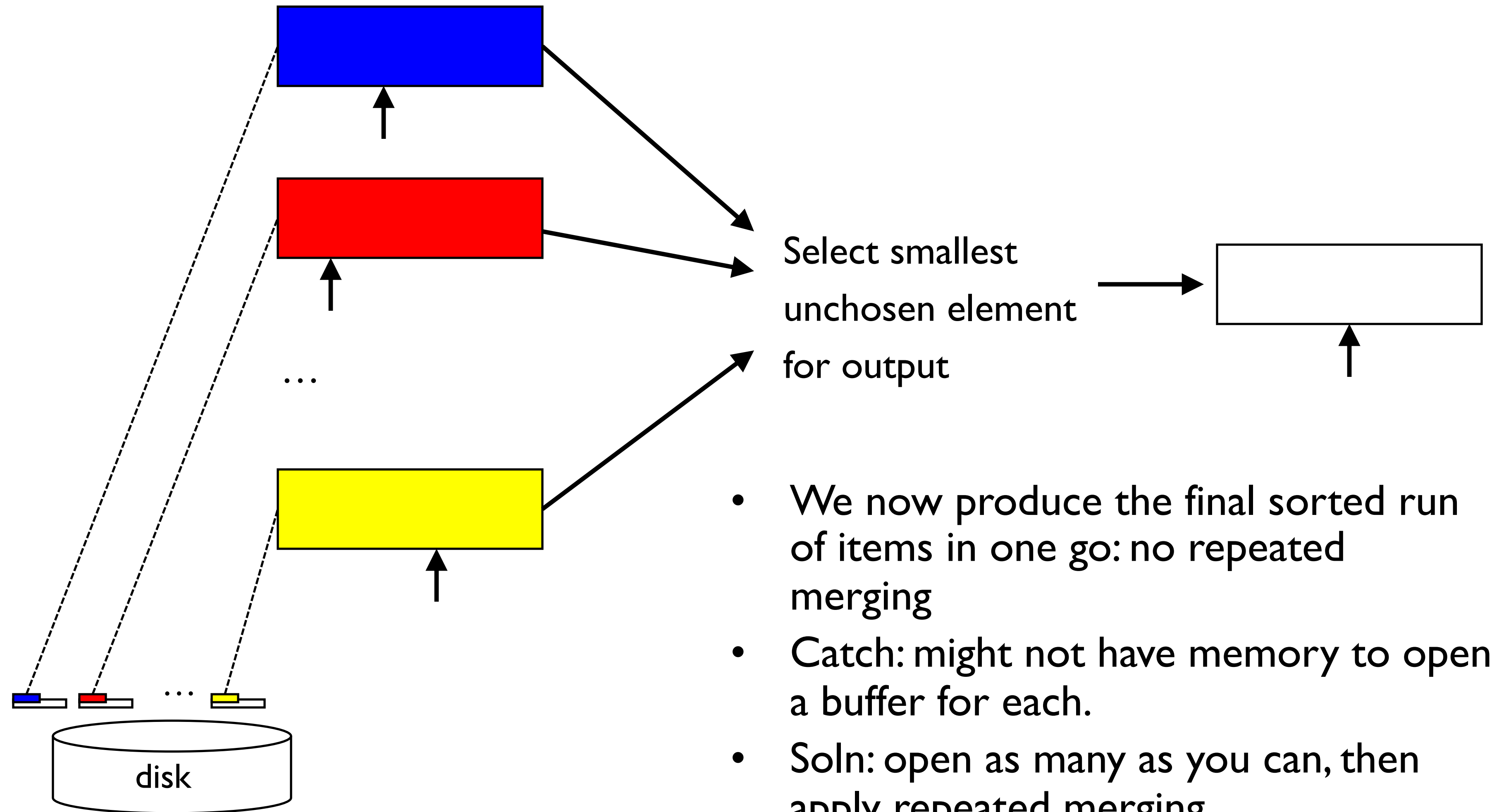
1. Multi-way merge



1. Multi-way merge

Input buffer for each run

Output buffer



- We now produce the final sorted run of items in one go: no repeated merging
- Catch: might not have memory to open a buffer for each.
- Soln: open as many as you can, then apply repeated merging

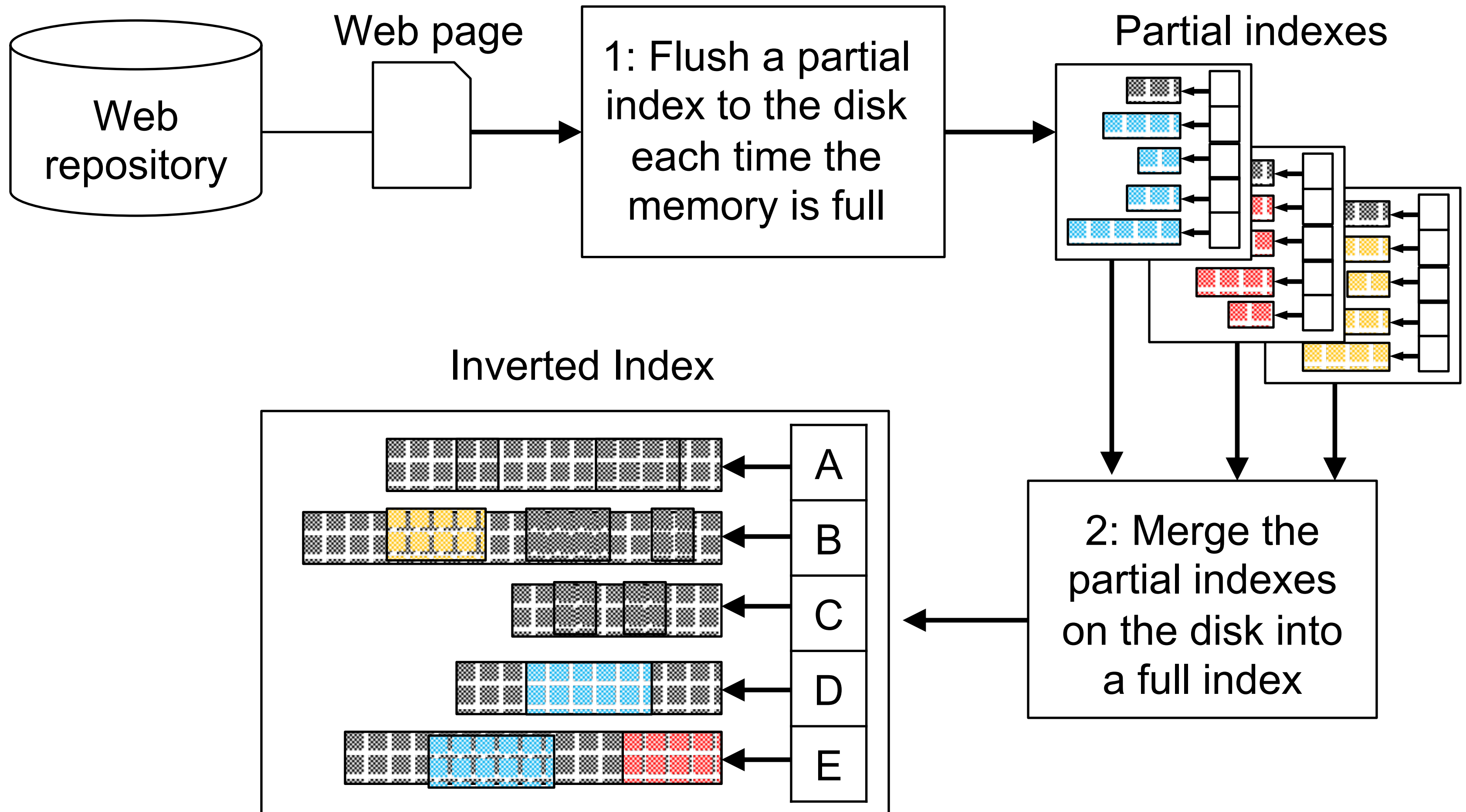
2. Make longer initial runs

- Before we have collected like terms (currently the last thing we do) if a term t occurs k times in the input, it is repeated k times in the runs:
  - Initial: (the,1)(cat,1)(the,1)(bat,1)(the,2)(cat,2)(bat,3)(cat,3)(the,4)(the,4)(the,5)(cat,5)(cat,5)
  - Sorted: (bat,1)(bat,3)(cat,1)(cat,2)(cat,3)(cat,5)(cat,5)(the,1)(the,1)(the,2)(the,4)(the,4)(the,5)
- But in the final output it occupies only d integers, or one integer for each document it appears in:
  - Final: (bat: 1,3)(cat: 1,2,3,5)(the:1,2,4,5)

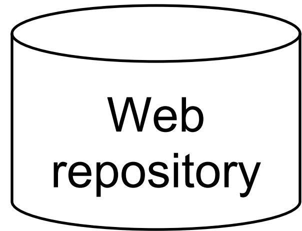
Collect like terms during initial run creation

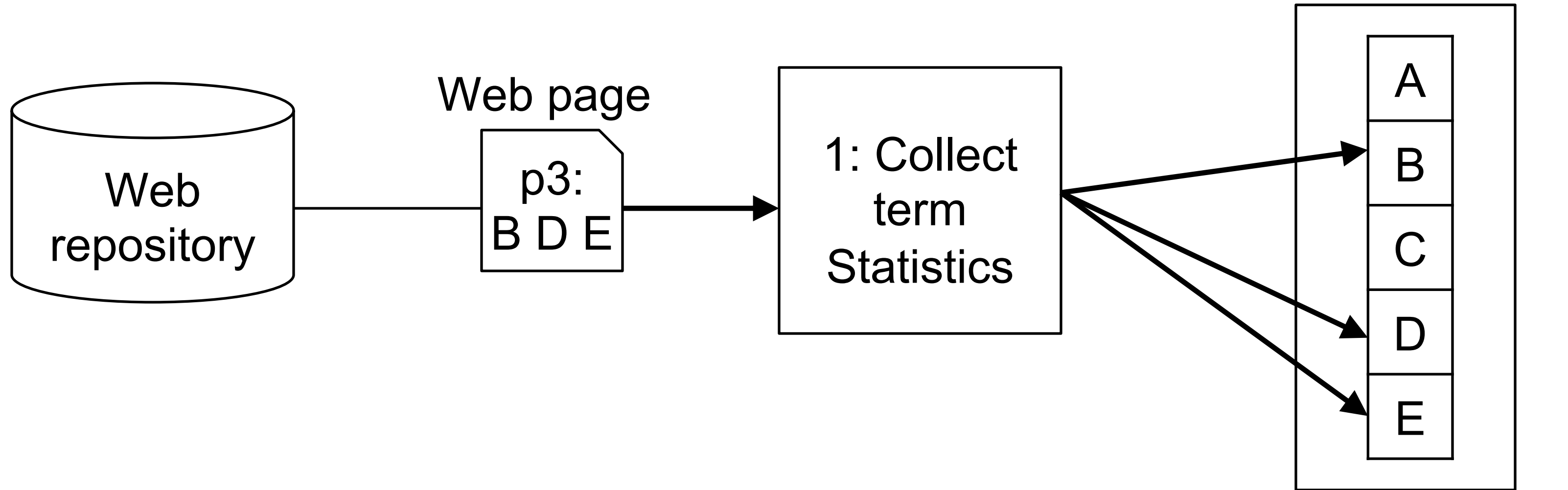
- To exploit this observation, we collect like terms not at the end, but during the initial run creation phase.
  - A bit like applying insertion sort on postings as they are read in
- Because less space is used per term, we can process more of the collection before being forced to write a run out to disk
  - Runs get longer
  - Fewer runs
  - Less merging to do
- Single-Pass In-Memory Indexing (SPIMI)
  - Heinz and Zobel (2003)
  - Made even more effective by compression.

One-pass index construction

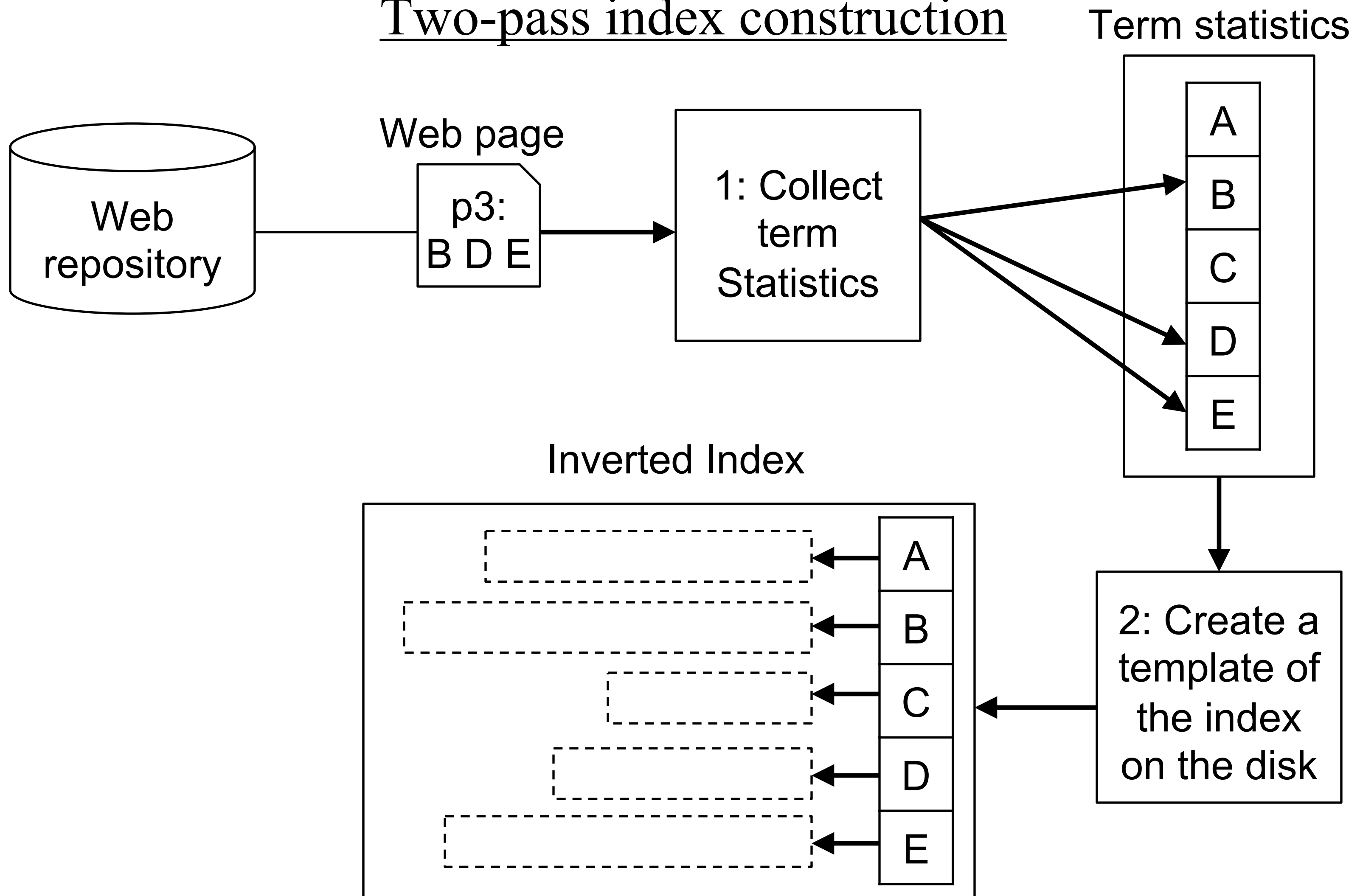


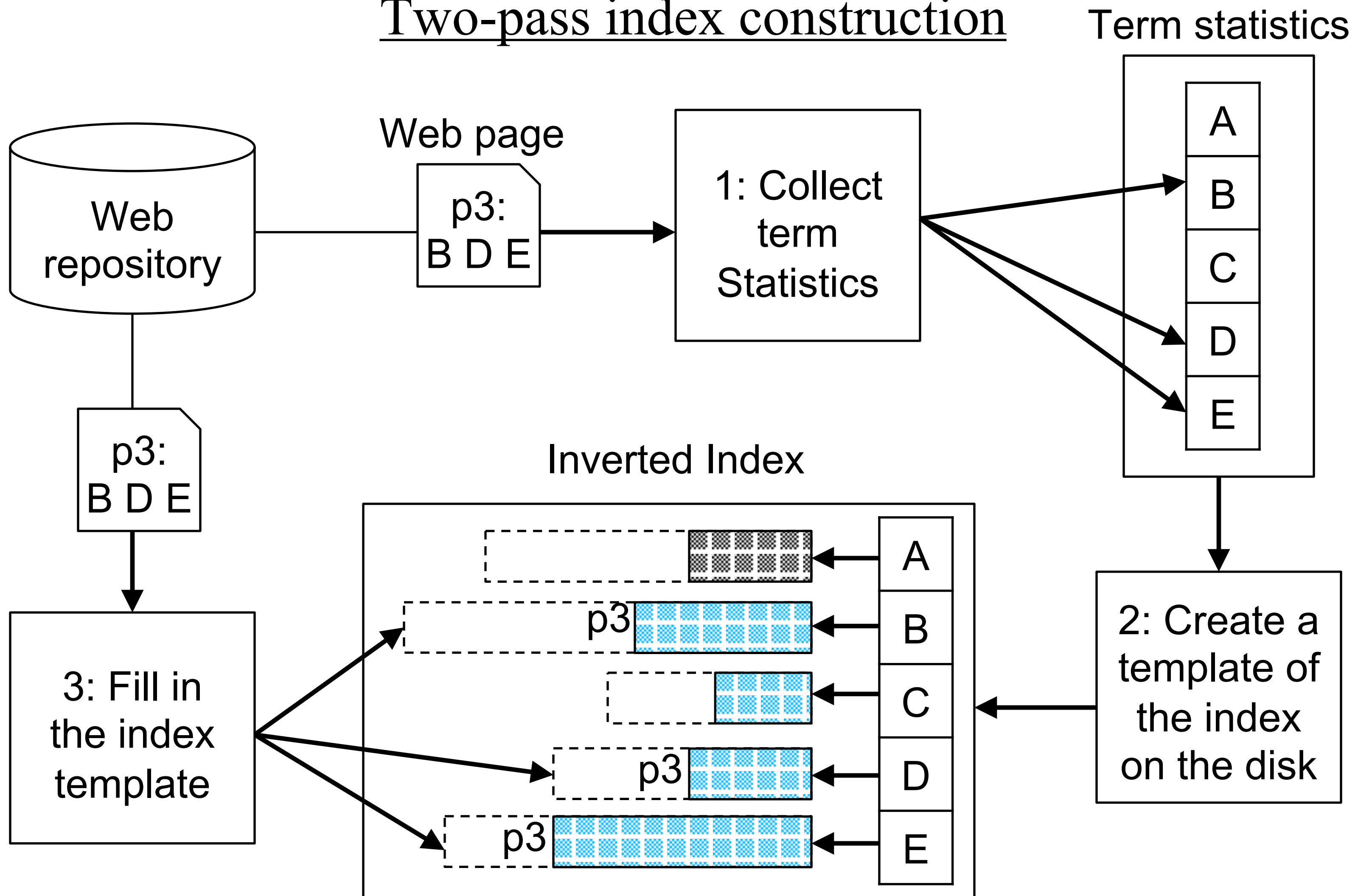
Two-pass index construction




Two-pass index construction
Term statistics
Web page
Web
repository
p3:
B D E
1: Collect
term
Statistics
A
B
C
D
E

Two-pass index construction





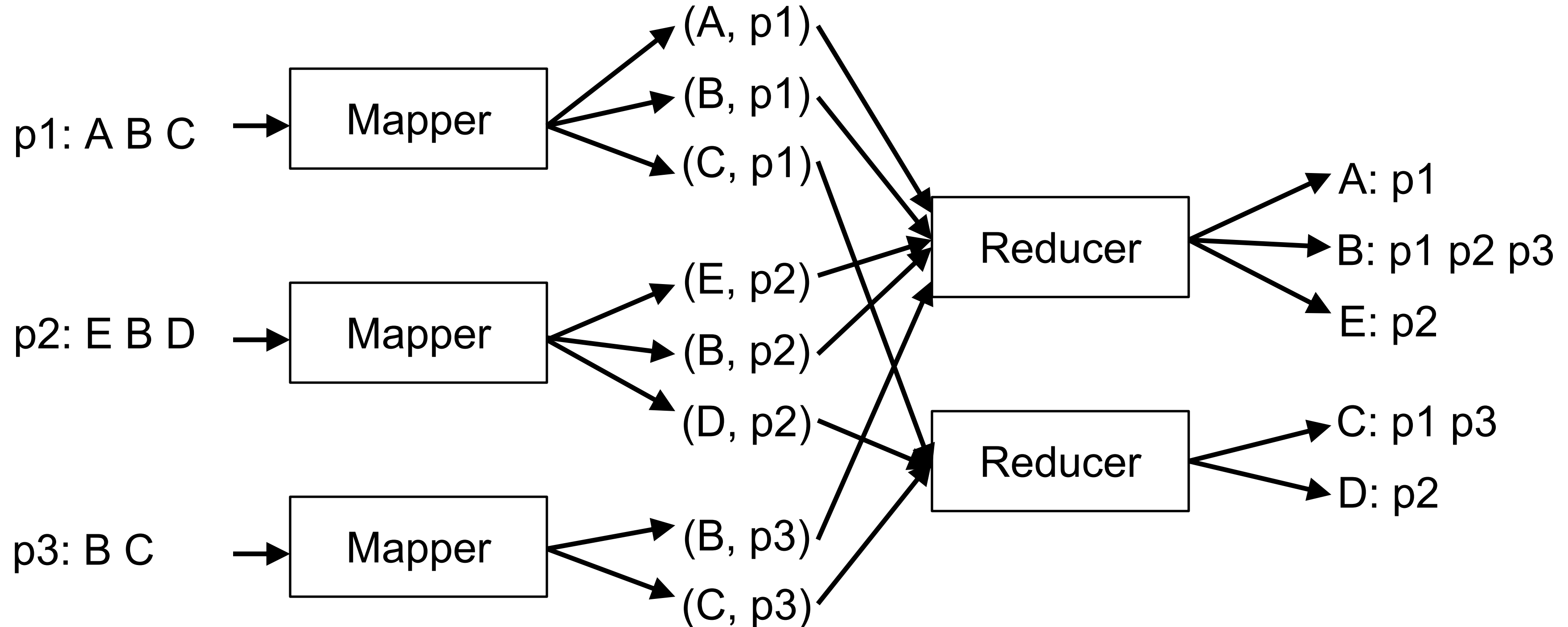
Two-pass index construction
Term statistics
Web page
Web repository
p3:
B D E
1: Collect term Statistics
A
B
C
D
E
p3:
B D E
Inverted Index
3: Fill in the index template
p3
p3
p3
A
B
C
D
E
2: Create a template of the index on the disk

- In large-scale indexing systems, the index is built and deployed on thousands of nodes
  - Besides being scalable and efficient, the indexing system needs to tolerate hardware, software and network failures

- Hadoop, a distributed storage and processing framework for very large data sets, can be used to create inverted indexes in a scalable, fast, reliable way

- Converting a large document collection into an inverted index becomes a conceptually simple task…

Construction via Map/Reduce



- *Mappers* read web pages and emit (term, doc id) pairs
  - terms are keys, doc ids are values
- Pairs are partitioned across *Reducers* according to terms
  - doc ids associated with each term are grouped at different reducers
- Reducers output the list of doc ids lists, one term at a time

Updating an Inverted Index…

- Periodically rebuilding the index guarantees a certain level of freshness

- For the main web search index, a deployment cycle of a few hours may be considered reasonable

- However, time-sensitive search *verticals* (e.g. news or tweet search) have even stricter freshness reqs
  - Modifications to page repository (insertions, deletions, updates) should be reflected in the inverted index as quickly as possible

- Incorporating term-related info from new web pages into an inverted index is tricky
  - Lists are compressed and packed into disk blocks

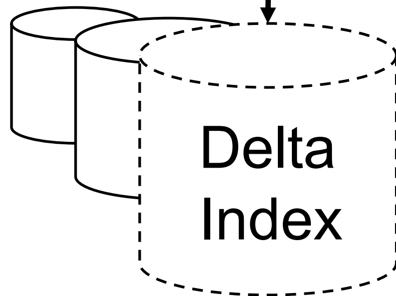
- There are several solutions, all relying on keeping a so-called *delta index* in memory
  - Delta index is small inverted index that keeps information about the most recently added documents
  - Once the delta index is grows to big it is written to disk and a new delta index starts to be built
  - Available solutions differ in the way the delta is merged with the main inverted index…

No merge

Memory

Delta Index

Disk



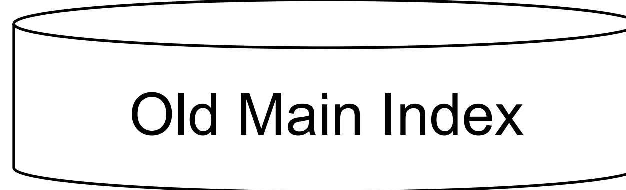
- Simplest approach: store each delta index as a separate index on disk beside the main index
  - Low index maintenance: just write the delta to disk and start a new one
  - Major drawback: inverted lists become fragmented over many small indexes
  - Query processing increases since all deltas + main index must be accessed

Incremental update

Memory

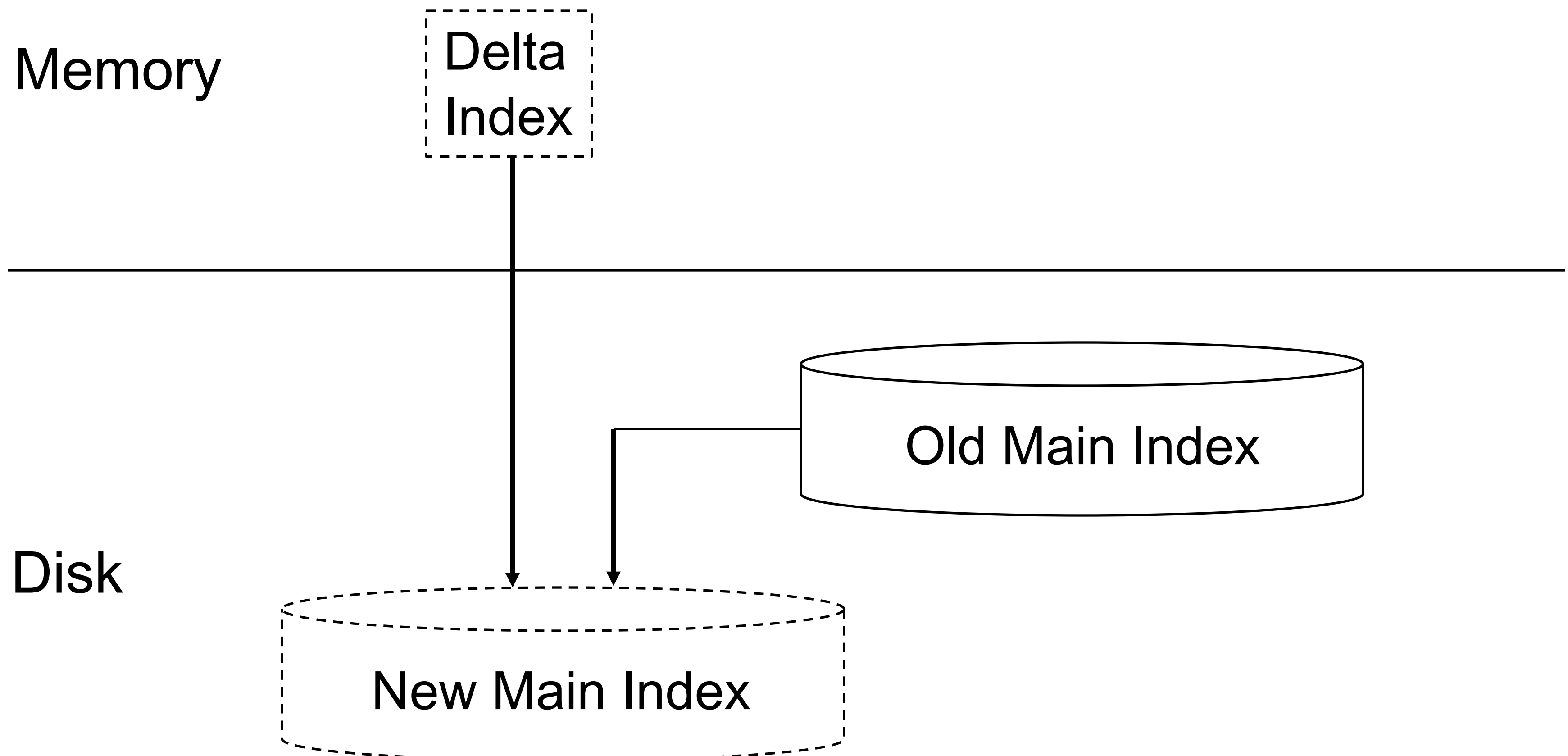
Delta
Index

Disk



- Update main index *in situ*
  - Requires allocating buffer space at end of every inverted list at construction
  - During merge delta entries are appended to these buffer spaces
  - Only one delta is maintained, preexisting parts of main index not modified
  - Drawbacks: queries/updates compete for resources; buffers eventually fill up

Immediate merge



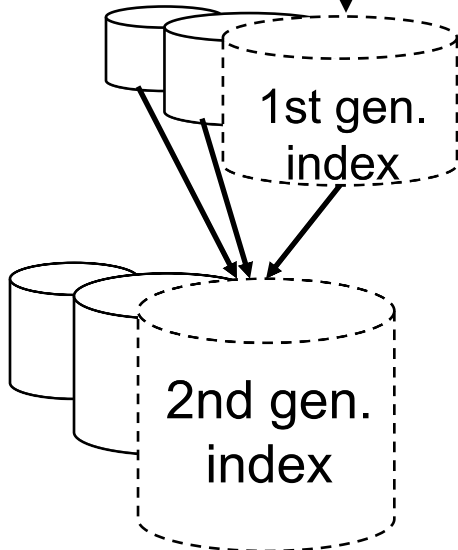
- Merge delta with main index in memory and write an up-to-date main index to disk
  - Single active idx maintained on disk at all times → query processing unaffected
  - However every merge operation requires reading and writing entire index

Lazy merge

Memory

Delta Index

Disk



Old Main Index

- Variant: maintain multiple generations of deltas
  - Delta indexes are lazily merged over time to create larger indexes on disk
  - Creates tradeoff between index maintenance cost and query processing cost

- Besides handling page insertions, the idx maintenance technique should also handle page deletions

- Naïve option is to maintain ids of deleted pages in an in-memory structure and filter them from results

- Better approach is to have some kind of garbage collection mechanism and regularly remove deleted pages from in-memory and on-disk indexes
  - Garbage collection can be performed on-the-fly during merge operations

Next week…

16.1: Introduction to Indexing
- Boolean Retrieval model
- Inverted Indexes

21.1: Index Compression
- unary, gamma, variable-byte coding
- (Partitioned) Elias-Fano coding (used by Google, facebook)

23.1: Index Construction
- preprocessing documents prior to search
- building the index efficiently

28.1: Web Crawling
- getting documents off the web at scale
- architecture of a large scale web search engine

30.1: Query Processing
- scoring and ranking search results
- Vector-Space model