The index merge scheduling problem

John MacCormick, Frank McSherry

Microsoft Research Silicon Valley

Motivation: full text indexing of dynamic content

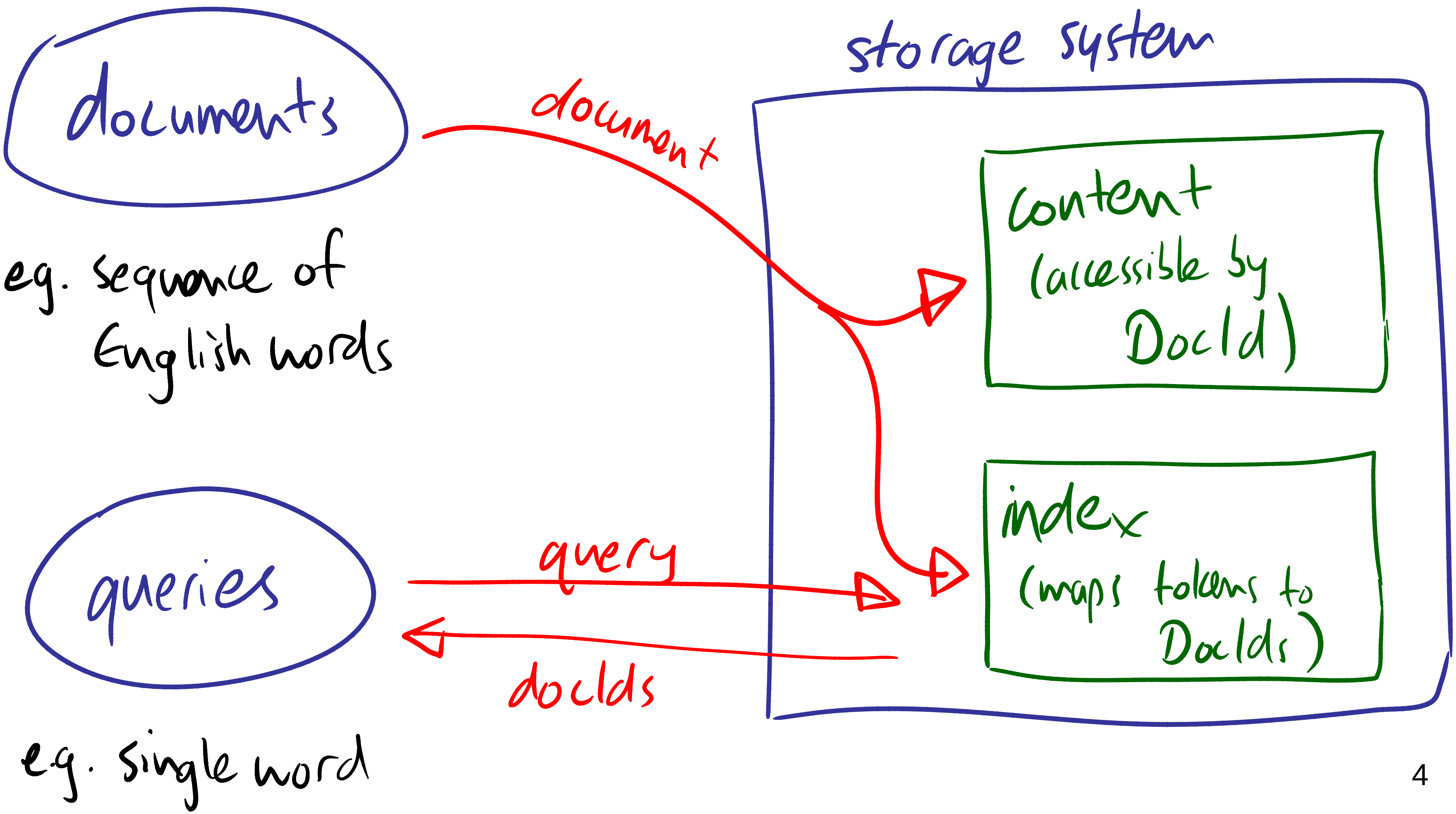
- data arrives continuously
- queries must reflect latest arrivals
- examples:
  - webmail (Yahoo Mail, Gmail, Hotmail,…)
  - blogs (MSN spaces)
  - news stories (Google News)
  - desktop search
  - Web search

Motivation: full text indexing of dynamic content

- data arrives continuously
- queries must reflect latest arrivals
- examples:
  - webmail (Yahoo Mail, Gmail, Hotmail,…)
  - blogs (MSN spaces)
  - news stories (Google News)
  - desktop search
  - Web search

focus on this as main example

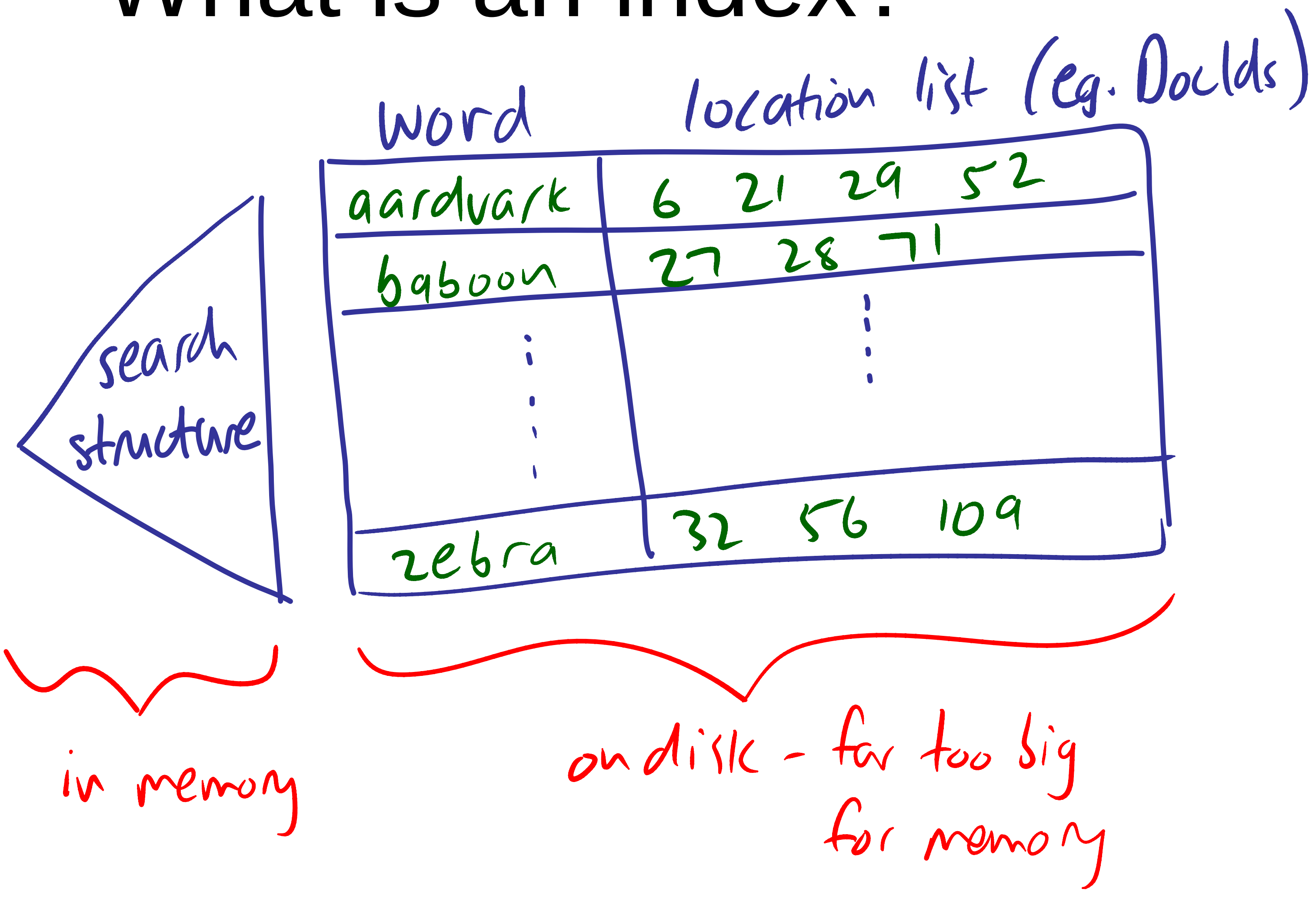
High -level overview of dynamic content indexing



the index merge scheduling problem: roadmap

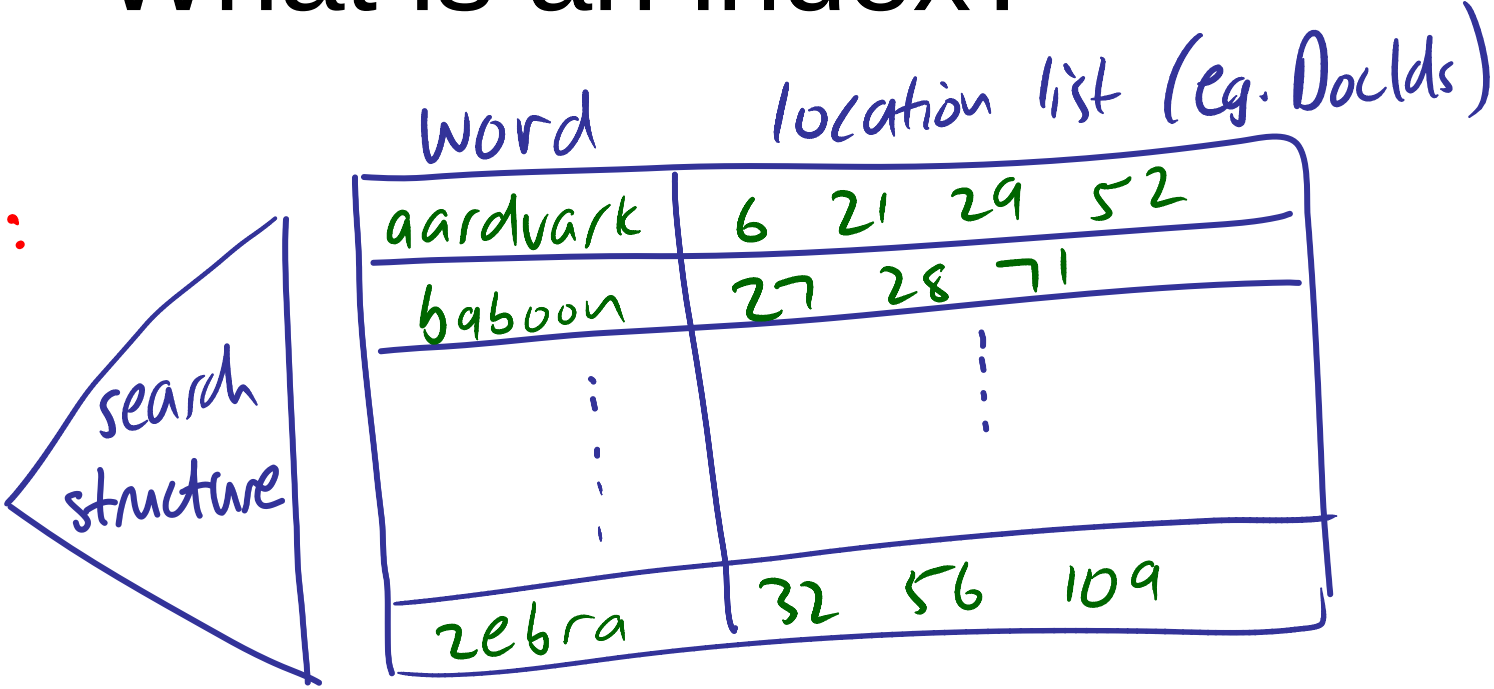
1. single index is insufficient for dynamic content – need multiple indexes, and therefore need occasional index merges

What is an index?



What is an index?

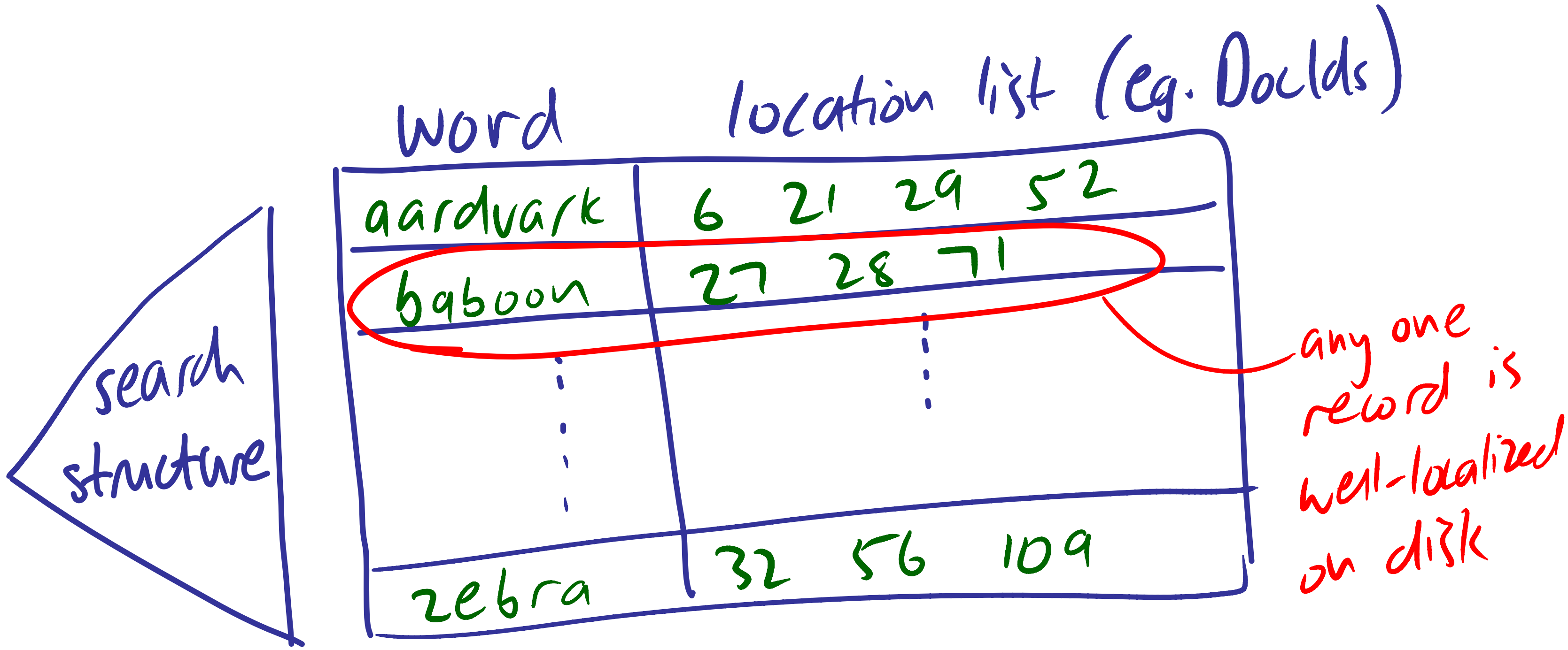
Logically:



Physically: B-tree, vanilla inverted file, fancy inverted file

(Zobel et al 93, Lomet 88, Carey et al 89)

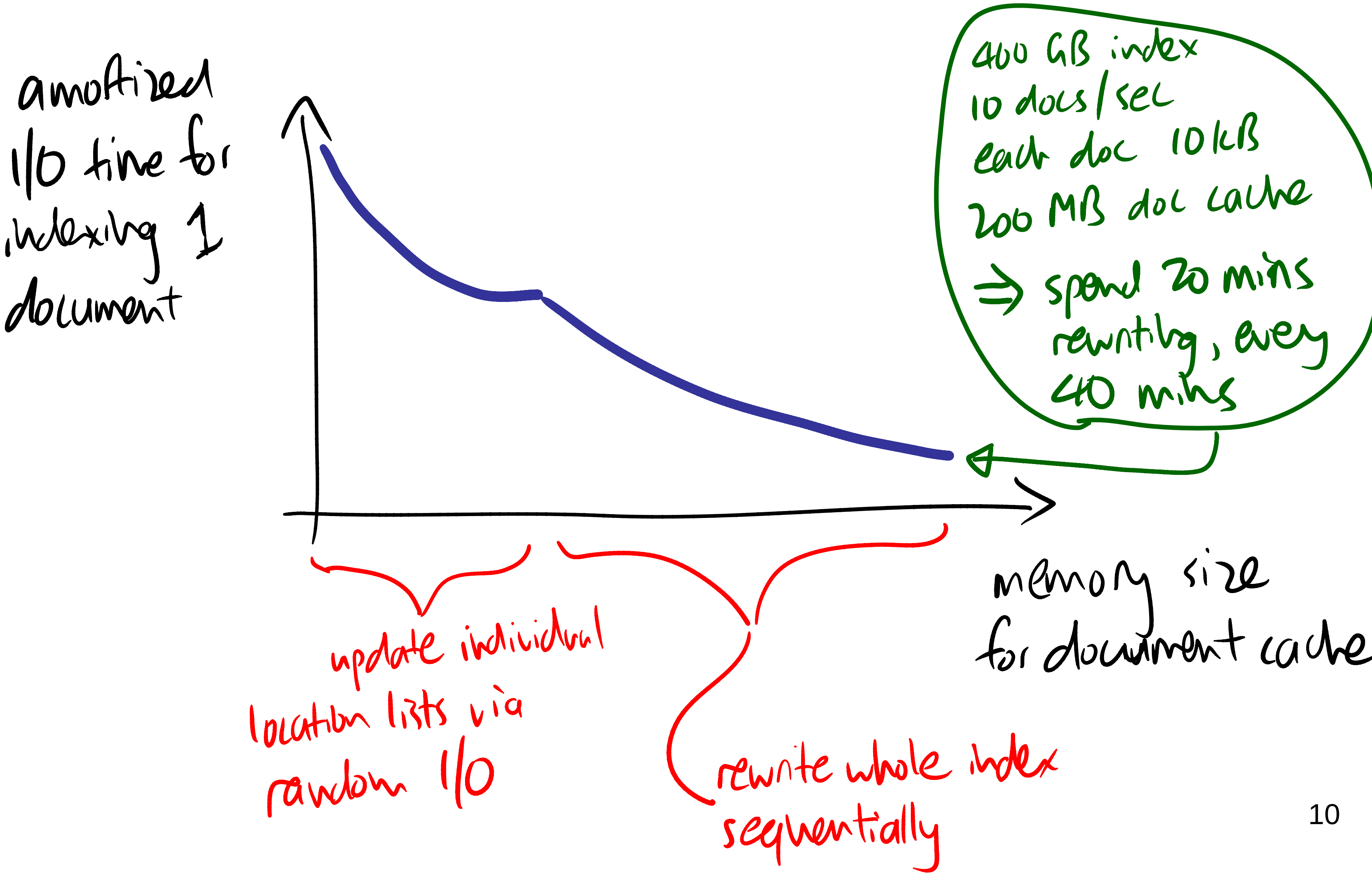
Definition: a *single index* can retrieve the entire location list for a word in a single I/O



Naïvely indexing dynamic content is far too slow

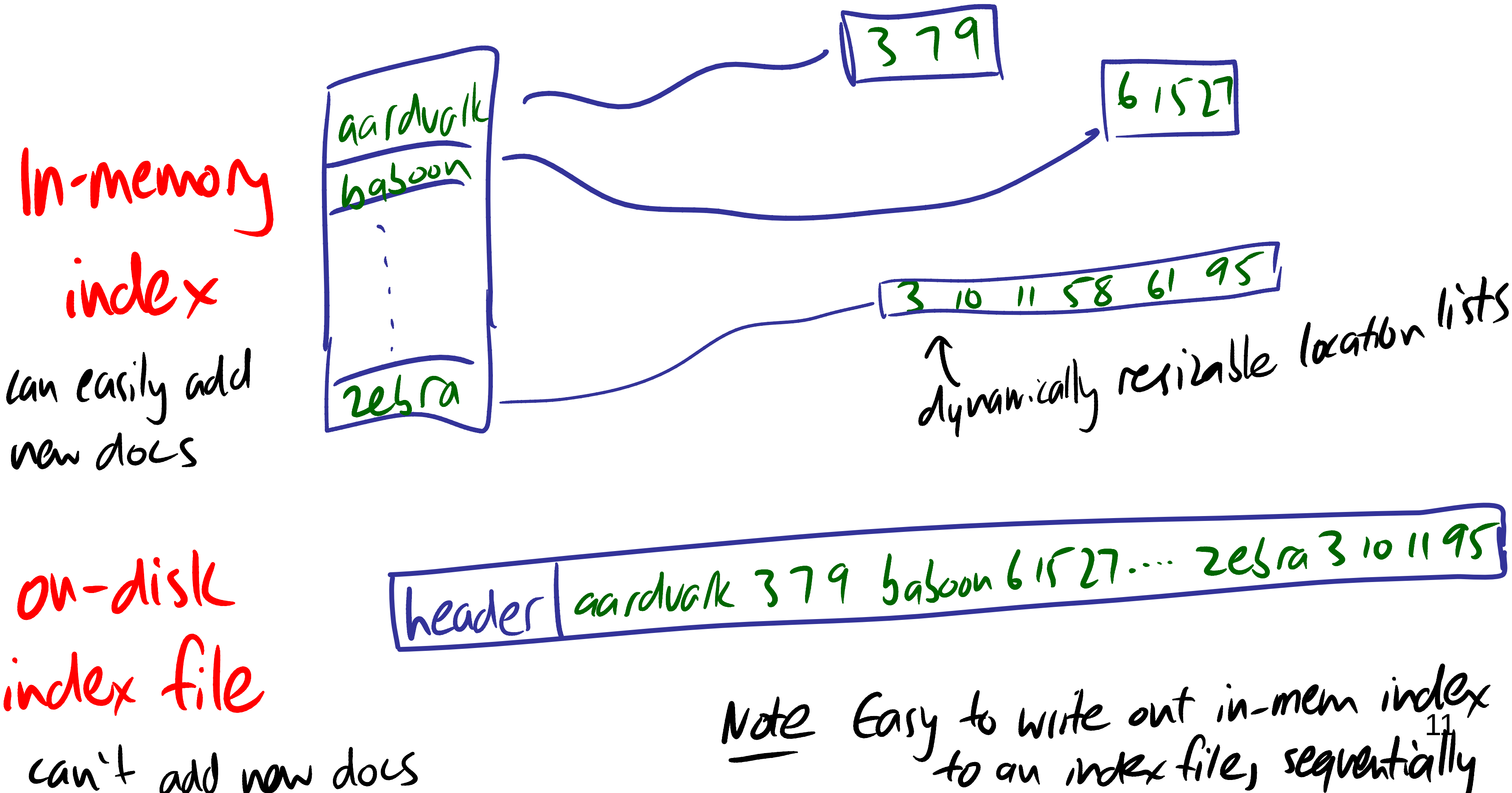
- Recall:
  - random I/O is slow (10 ms seek time per I/O)
  - sequential I/O is fast (1 seek + 100 MB/s)
- Example of naïve indexing: scientific paper
  - 100 kB as text file
  - contains 2500 unique indexable words
  - naïvely updating each index entry would take 2500 times longer than writing the file sequentially!

Claim: a single index is too slow

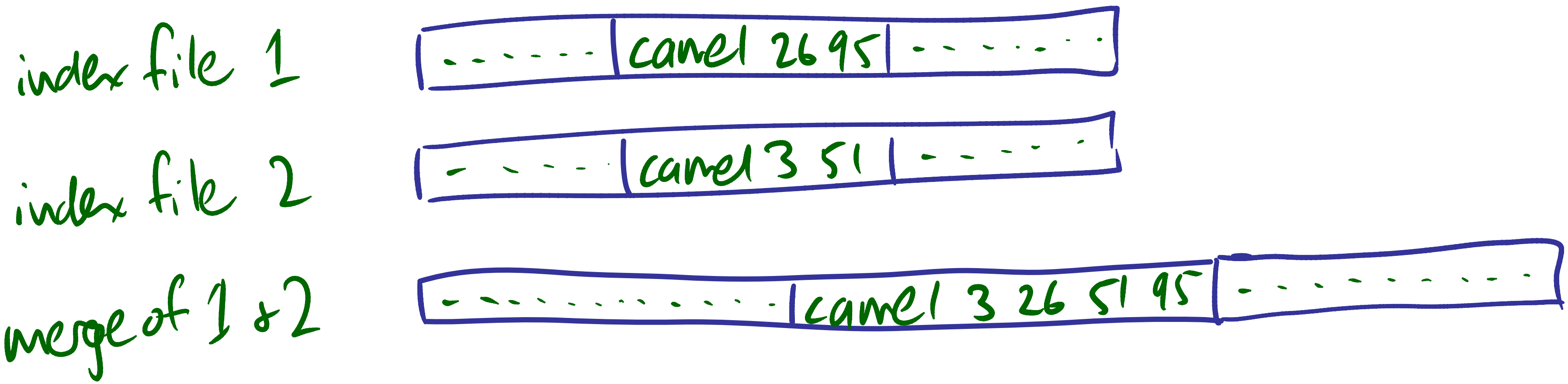


Therefore, we need multiple indexes

- 2 basic types: in-memory and on-disk



Index files can be merged using only sequential I/O and little memory



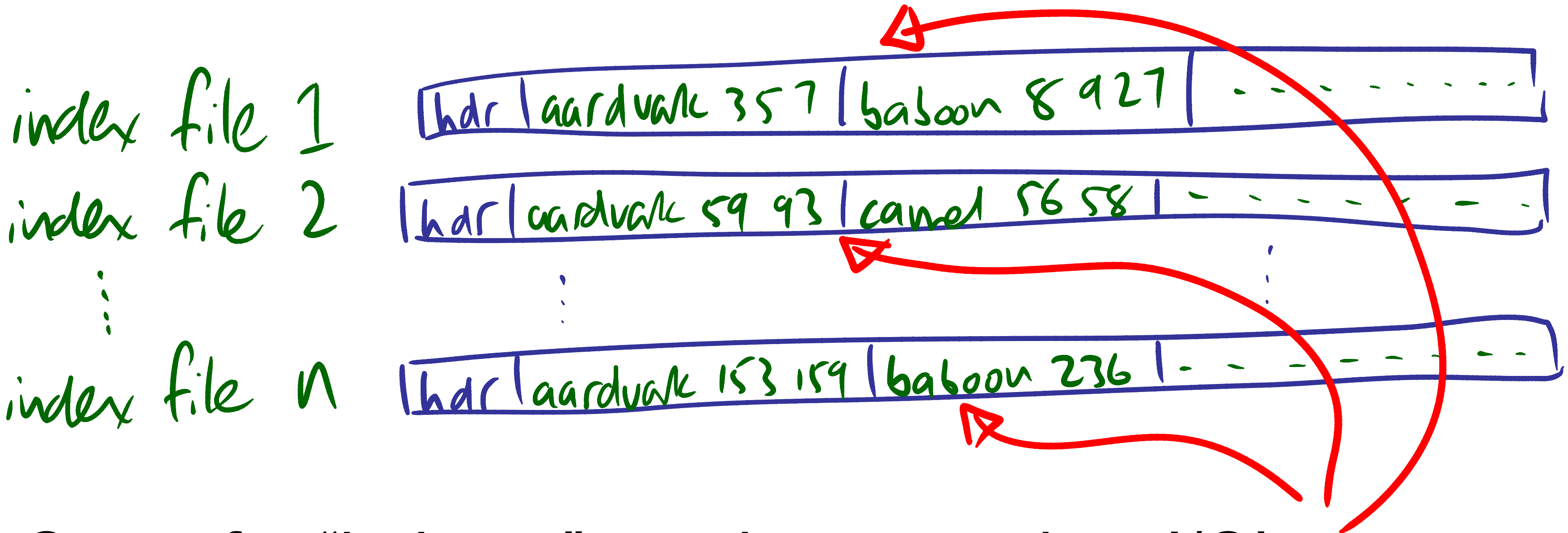
I/O cost of merge = cost of sequential read of 1+2
+ sequential write of 1+2

- can merge many indexes into 1 simultaneously
- in-memory indexes can be merged with on-disk index files
- writing out in-memory index to disk by itself can be regarded as a trivial merge

Basic strategy for dynamic indexing: accumulate and merge

- Repeat:
  - accumulate as many documents as possible in an in-memory index
  - merge in-memory index with zero or more index files
- Optionally, in parallel, repeat:
  - merge some index files

Why merge? Because query cost is proportional to number of indexes



Query for "baboon" requires n random I/O's

Exception: if index files are sorted by relevance, some queries require less I/O

the index merge scheduling problem: roadmap

1. single index is insufficient for dynamic content – need multiple indexes, and therefore need occasional index merges
2. scheduling merges is related to the cost-distance problem in network construction

Formal definition of index merge scheduling problem

given finite sequence of events $(e_1, e_2, \ldots e_T) \in \{D, Q\}^T$

D ← data arrival; Q ← query arrival

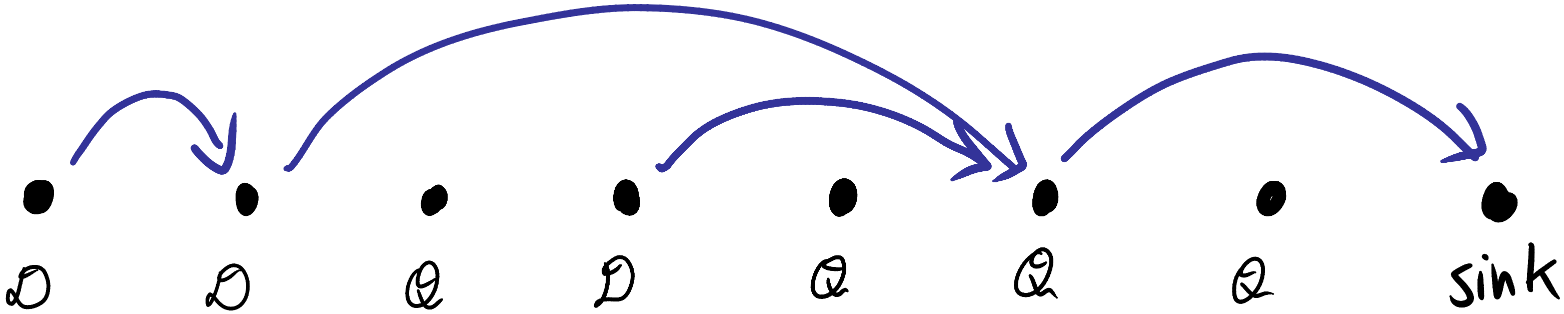
construct merge schedule

$m : \{1, 2, \ldots T\} \rightarrow$ set of indexes to be merged at time t

to minimize total cost

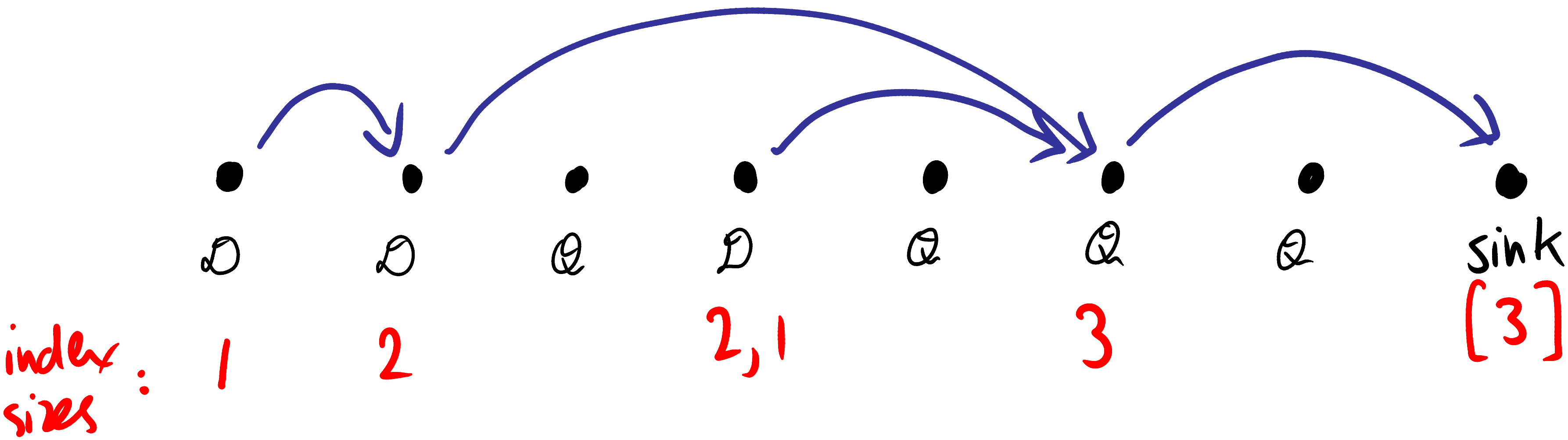
$$C = C_{merge}(T) + C_{query}(T)$$

merge scheduling can be regarded as constructing minimum-cost network on a certain graph



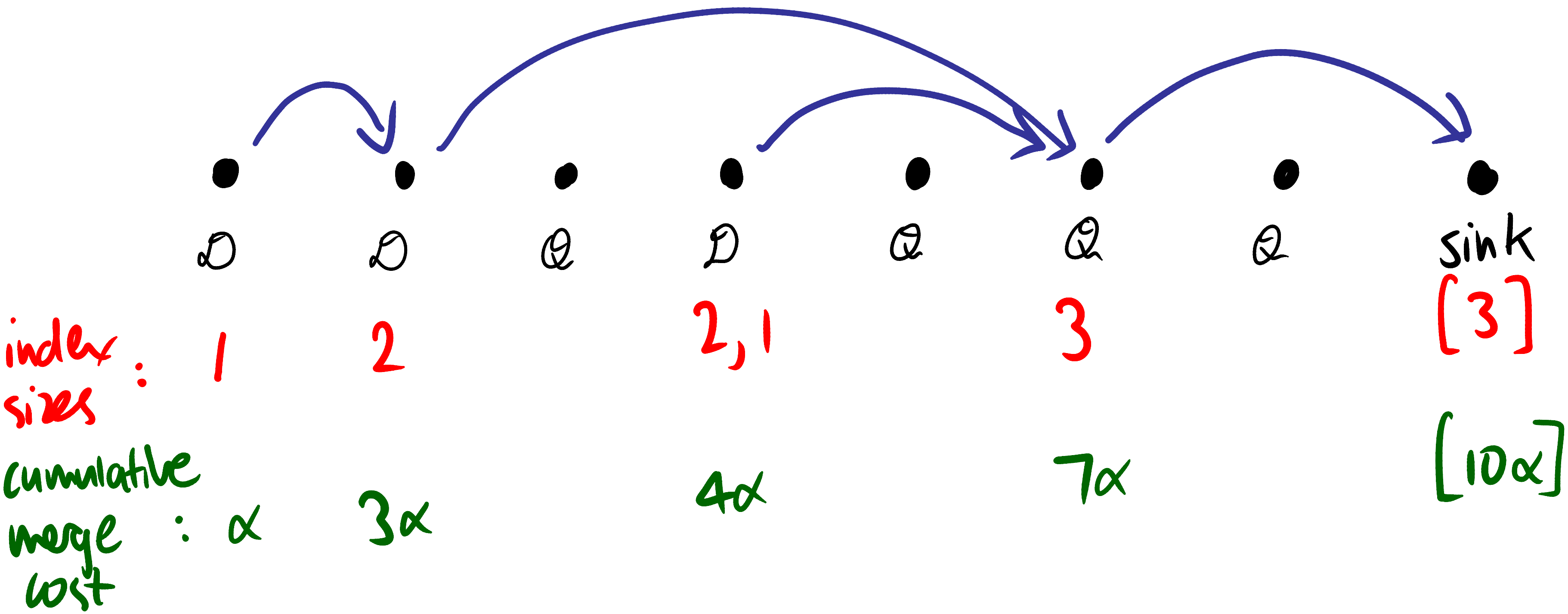
edge from A to B means “take all data written immediately after event A and merge it immediately after event B”

merge scheduling can be regarded as constructing minimum-cost network on a certain graph

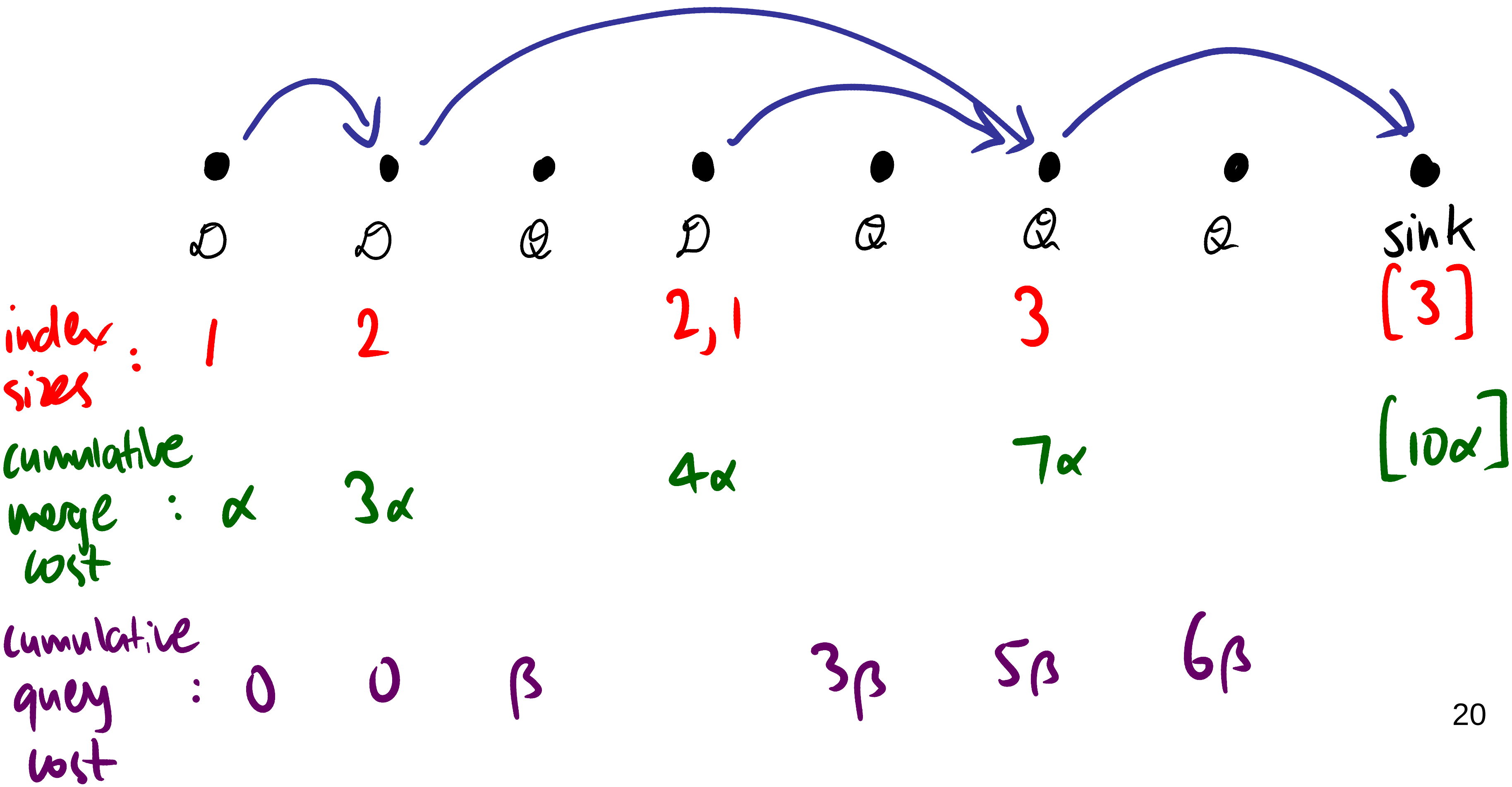


edge from A to B means "take all data written immediately after event A and merge it immediately after event B"

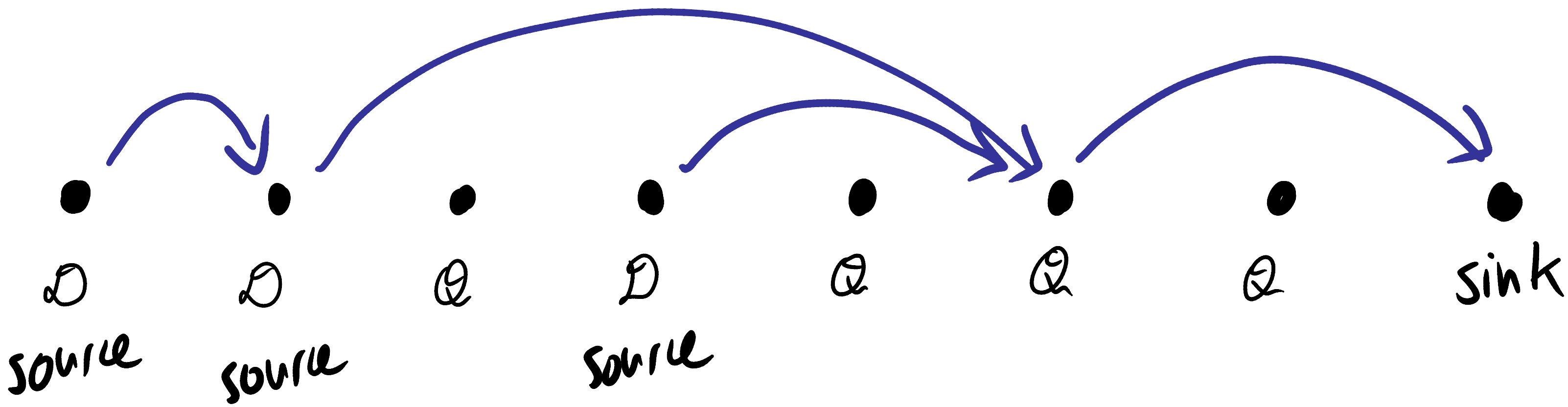
merge scheduling can be regarded as constructing minimum-cost network on a certain graph



merge scheduling can be regarded as constructing minimum-cost network on a certain graph



merge cost is sum of path lengths from sources to sink

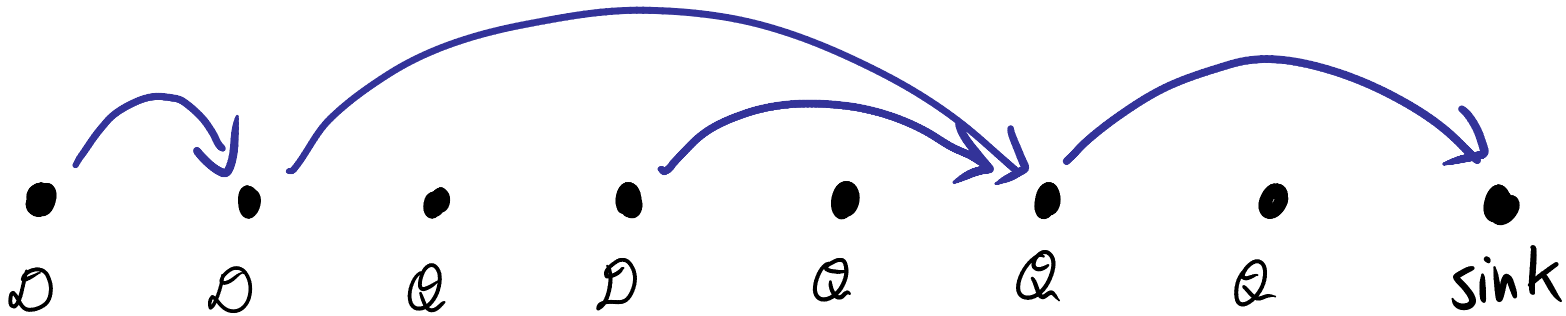


$$C_{merge} = \alpha \sum_{\text{sources } s} \text{len}(s, \text{sink})$$

↑ merge price

↑ path length i.e. number of hops

query cost is sum of edge costs



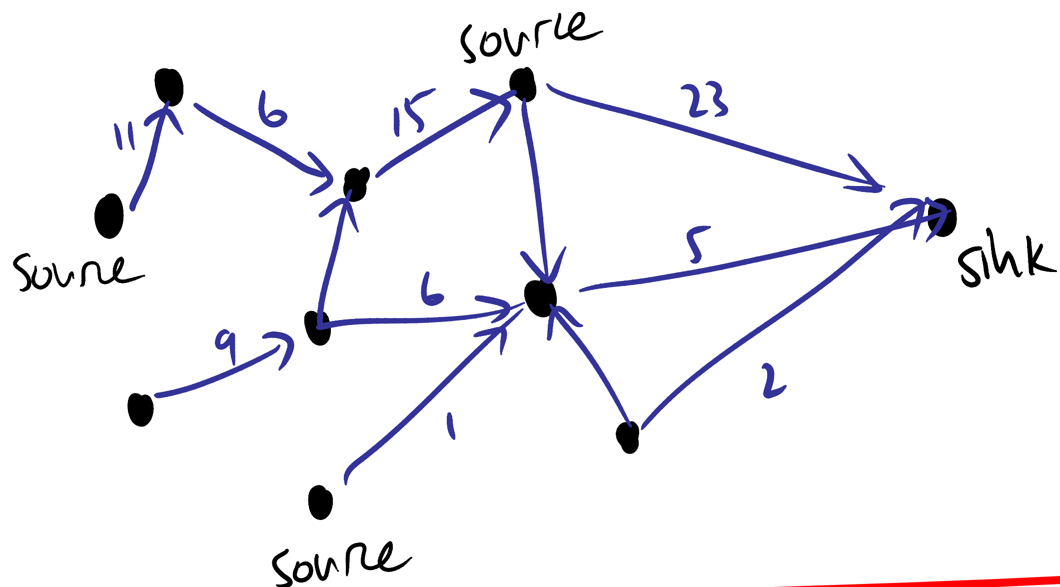
$c(e)$ = cost of edge e

= number of queries skipped by e

$$C_{query} = \beta \sum_{\text{edges } e} c(e)$$

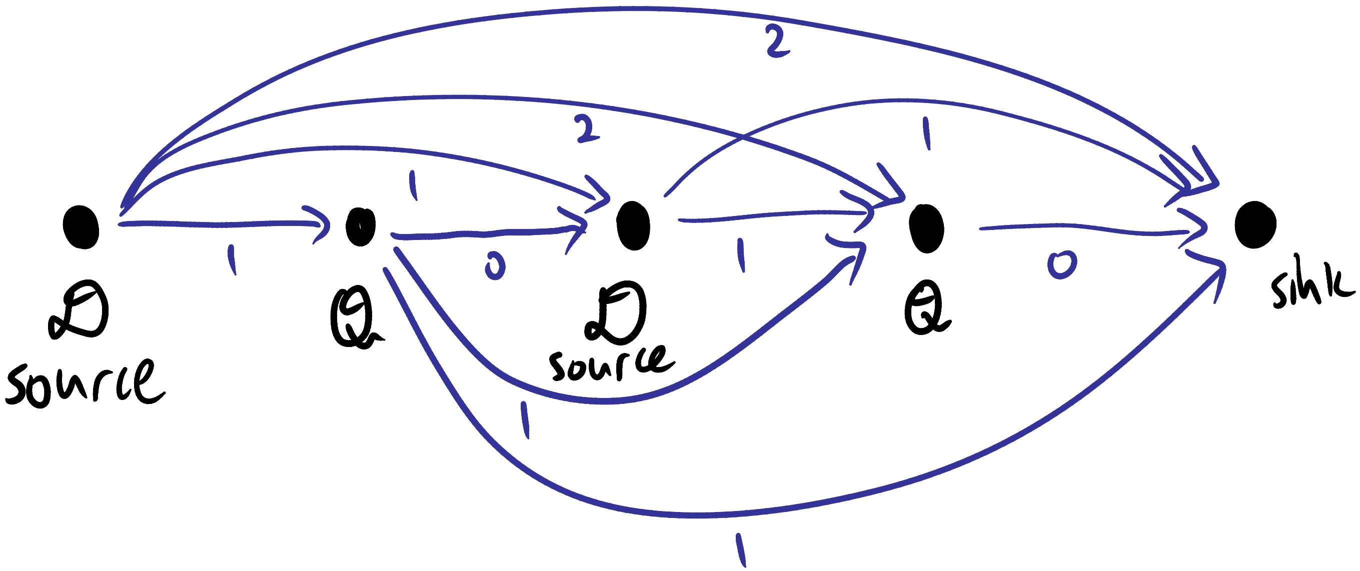
↑ query price

Index merge scheduling is a special case of "directed cost-distance" problem



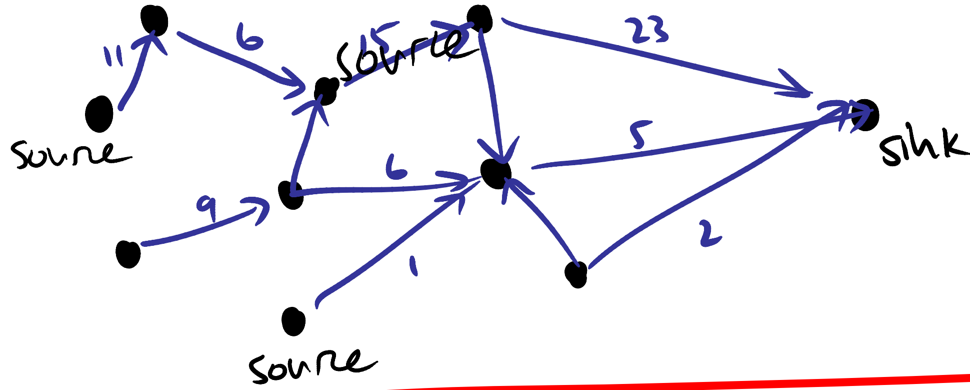
choose a subgraph minimizing $\sum_{\text{edges}} \text{edge cost} + \sum_{\text{sources}} \text{path length}$

Index merge scheduling is a special case of "directed cost-distance" problem



choose a subgraph minimizing $\sum_{\text{edges}} \text{edge cost} + \sum_{\text{sources}} \text{path length}$

"Cost-distance problem"



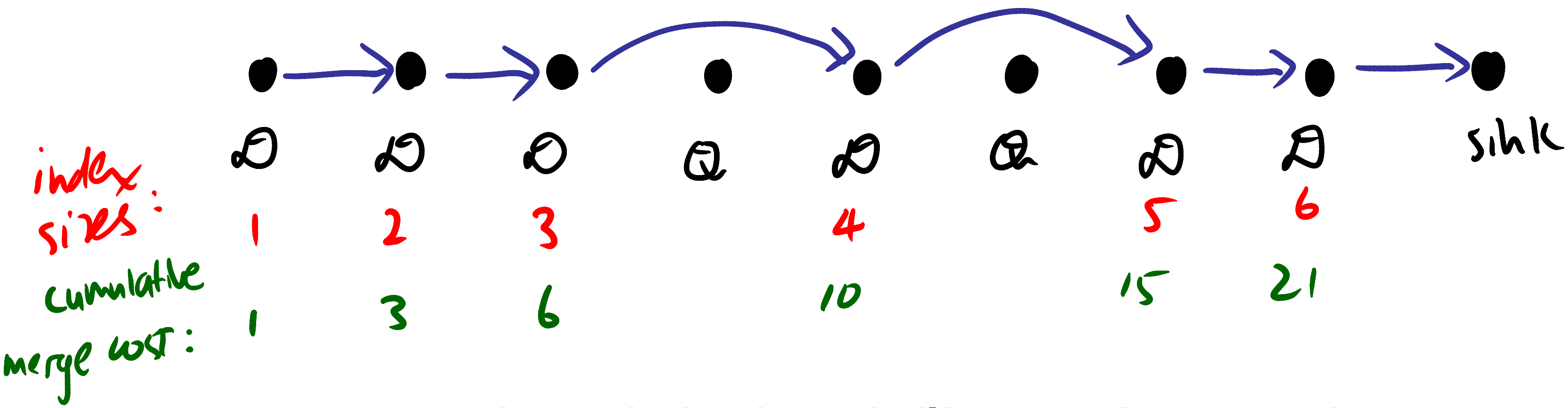
choose a subgraph minimizing $\sum_{\text{edges}} \text{edge cost} + \sum_{\text{sources}} \text{path length}$

- *undirected* case studied by Meyerson-Munagala-Plotkin 2000
- NP-complete (Steiner tree is special case)
- they give an efficient O(log(number of sources)) approximation
- *directed* case seems much harder
- fortunately, the graph for index merge scheduling has very special structure

the index merge scheduling problem: roadmap

1. single index is insufficient for dynamic content – need multiple indexes, and therefore need occasional index merges
2. scheduling merges is related to the cost-distance problem in network construction
3. imposing geometrically decreasing index sizes gives good performance – O(n log n)

merging as often as possible has quadratic cost

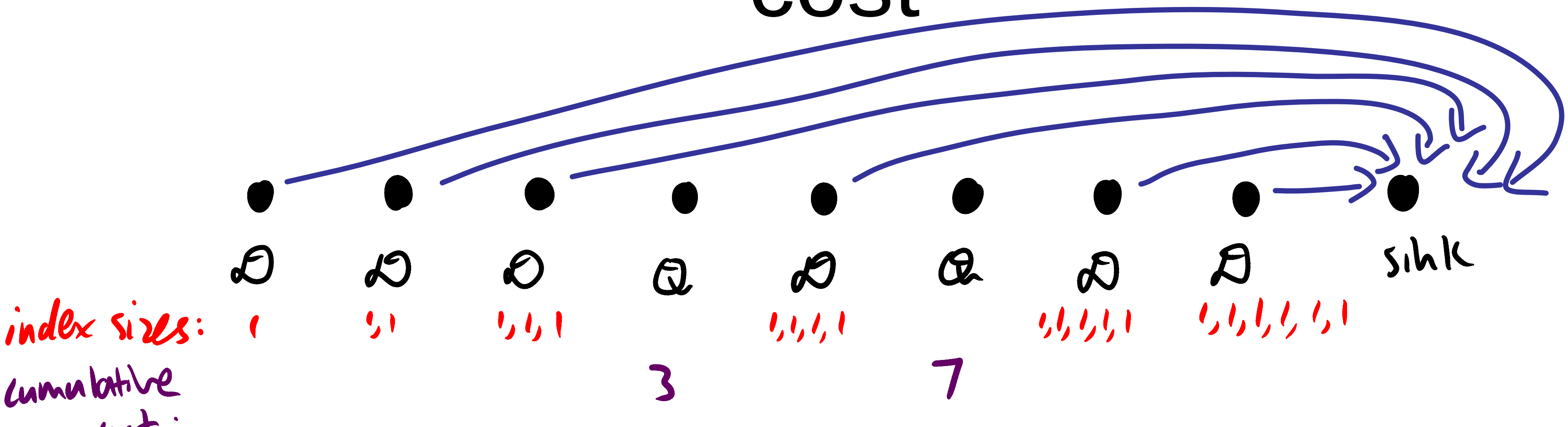


- query cost is minimized (linear in number of queries)
- but merge cost is quadratic in the number of data arrivals:

nth arrival costs αn,

so after n data arrivals, $C_{merge} = \frac{1}{2}\alpha n(n-1) = O(n^2)$

never merging also has quadratic cost



- merge cost is minimized (linear in number of data arrivals)
- but query cost can be quadratic:

n repititions

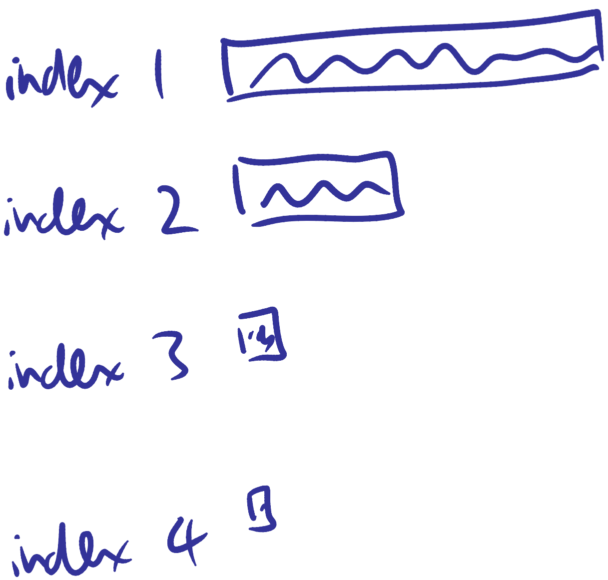
e.g. if data and queries alternate (D, Q, D, Q, ... D, Q)

then $C_{query} = \frac{1}{2}\beta n(n-1) = O(n^2)$

Maintaining geometrically decreasing index sizes guarantees total I/O cost is O(T log T)

Algorithm:

- Fix $K > 1$ (e.g. $K = 2$)



- Maintain invariant that if index sizes are sorted so that $s_1 > s_2 > \ldots > s_n$, then

$$s_i > K s_{i+1} \quad \text{for each } i$$

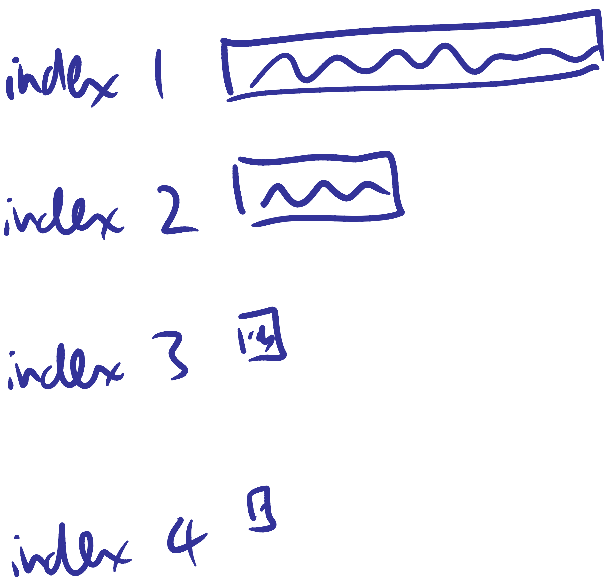
- Always merge the r smallest indexes, where r is minimal to keep the invariant

Maintaining geometrically decreasing index sizes guarantees total I/O cost is O(T log T)

Algorithm:

- Fix $K > 1$ (e.g. $K = 2$)
- Maintain invariant that if index sizes are sorted so that $S_1 > S_2 > \ldots > S_n$, then $S_i > K S_{i+1}$ for each i
- Always merge the r smallest indexes, where r is minimal to keep the invariant

similar alg used in Altavista, SQL Server, MS Desktop search etc



query cost is logarithmic in number of data arrivals

N_D = number of data arrivals

N_Q = number of queries

number of index files never exceeds $1 + \log_K N_D$

$$\Rightarrow \quad C_{query} \leq \beta N_Q (1 + \log_K N_D)$$

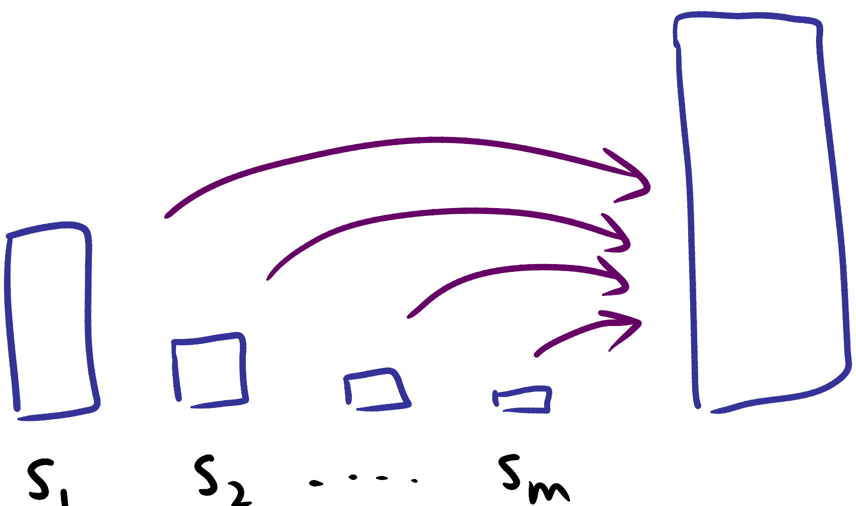
merge cost is also logarithmic in number of data arrivals

Lemma If we merge indexes with sizes $s_1 > s_2 > \ldots > s_m$, obtaining single new index file of size $s^* = s_1 + s_2 + \ldots + s_m$

then each $s_i \leq \frac{K}{K+1} s^*$

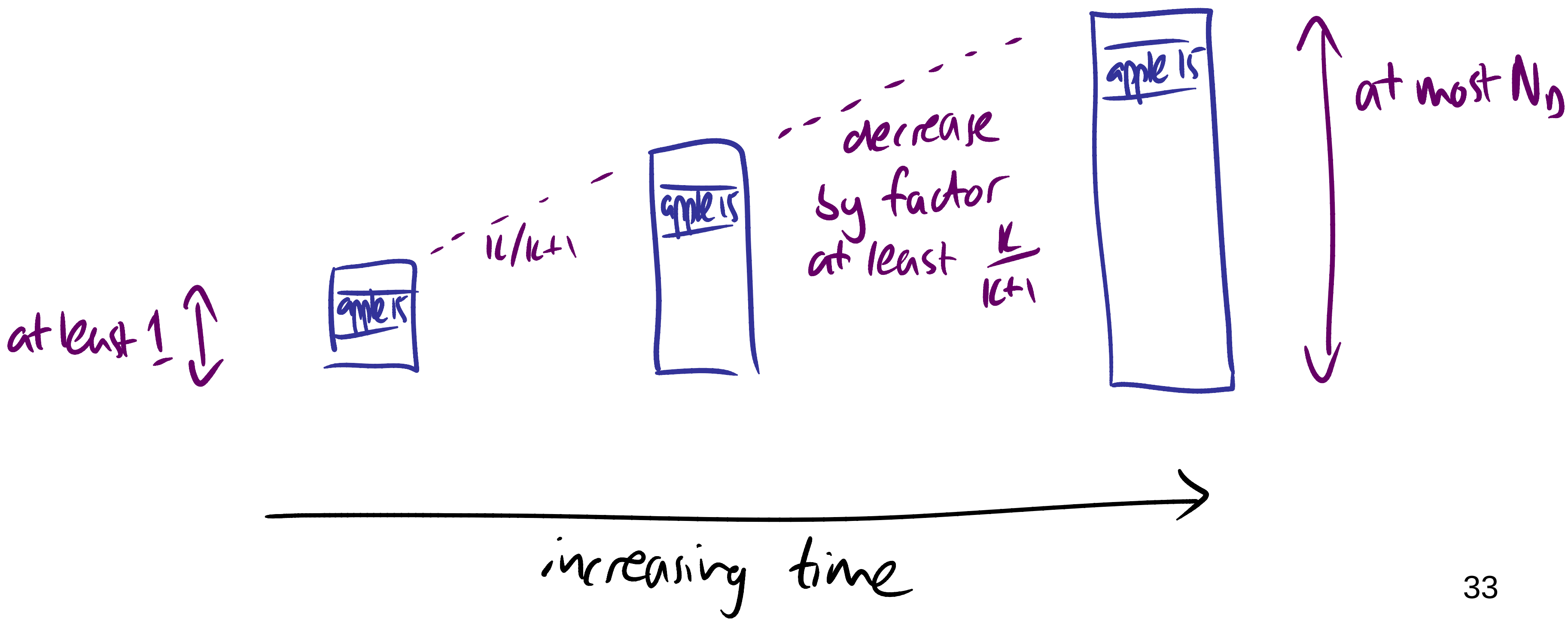
Proof: $s_2 + \ldots + s_m > s_1/K$

$\Rightarrow s^* > s_1 + s_1/K$ $\square$



merge cost is also logarithmic in number of data arrivals

Claim Each word-location pair is written out at most $1 + \log_{\frac{k+1}{k}} N_D$ times



merge cost is also logarithmic in number of data arrivals

Thus,

$$C_{merge} \leq \alpha N_D \left(1 + \log_{\frac{K+1}{K}} N_D \right)$$

merge cost is also logarithmic in number of data arrivals

Thus,

$$C_{merge} \leq \alpha N_D \left(1 + \log_{\frac{K+1}{K}} N_D\right)$$

tradeoff in choice of K

[Recall, $C_{query} \leq \beta N_Q \left(1 + \log_K N_D\right)$]

merge cost is also logarithmic in number of data arrivals

Thus,

$$C_{merge} \leq \alpha N_D \left(1 + \log_{\frac{K+1}{K}} N_D\right)$$

↕ tradeoff in choice of K

[Recall, $C_{query} \leq \beta N_Q \left(1 + \log_K N_D\right)$]

<u>NB</u> For $K=2$, $C_{merge} \approx \alpha N_D \log_2 N_D$

the index merge scheduling problem: roadmap

1. single index is insufficient for dynamic content – need multiple indexes, and therefore need occasional index merges
2. scheduling merges is related to the cost-distance problem in network construction
3. imposing geometrically decreasing index sizes gives good performance – O(n log n)
4. O(n log n) is optimal, in general

O(T log T) is optimal

Consider the input sequence $(D, Q, D, Q, \ldots, D, Q)$ — n pairs of D, Q

Claim If n is a power of 2, the optimal merge schedule costs at least

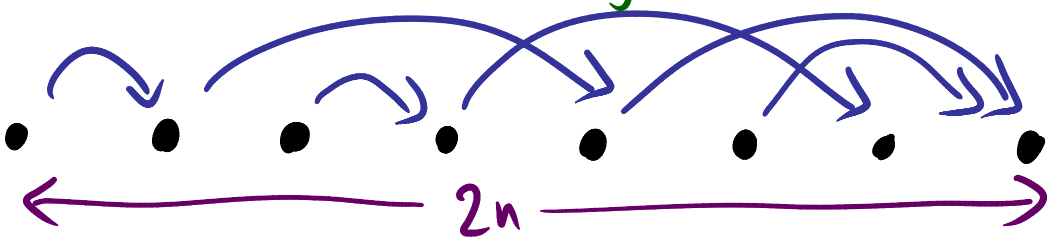
$$\min(\alpha, \beta) \frac{n}{2} \log_2 n$$

Claim If n is a power of 2, the optimal merge schedule costs at least

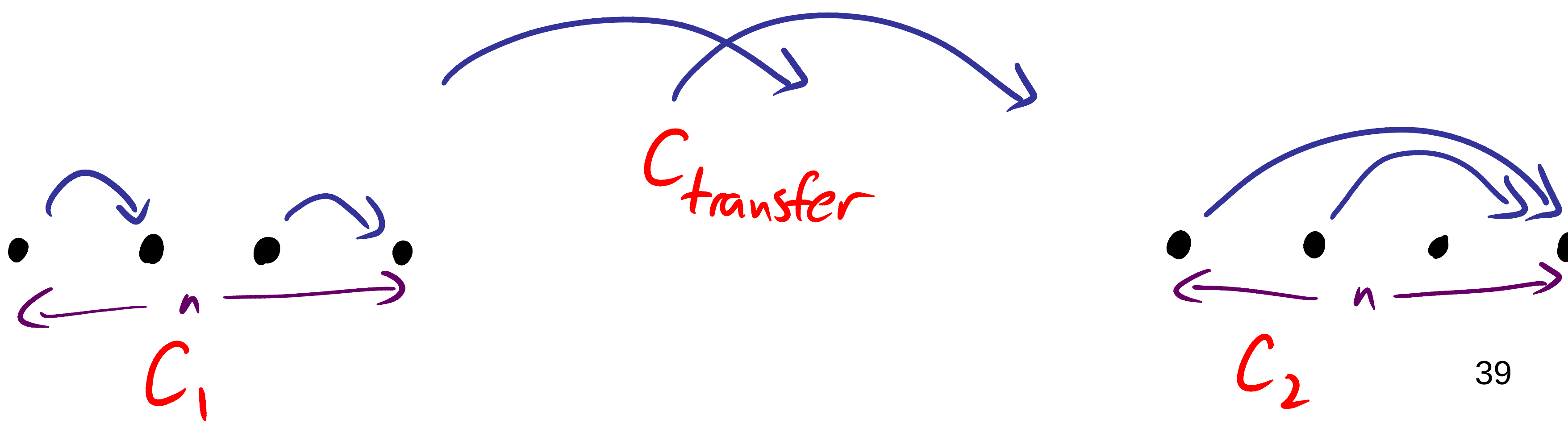
$$\min(\alpha, \beta)\ \frac{n}{2} \log_2 n$$

Proof: Induction on n. Assume for n, prove for $2n$.

Consider optimal strategy for $2n$:



Break cost into 3 parts:

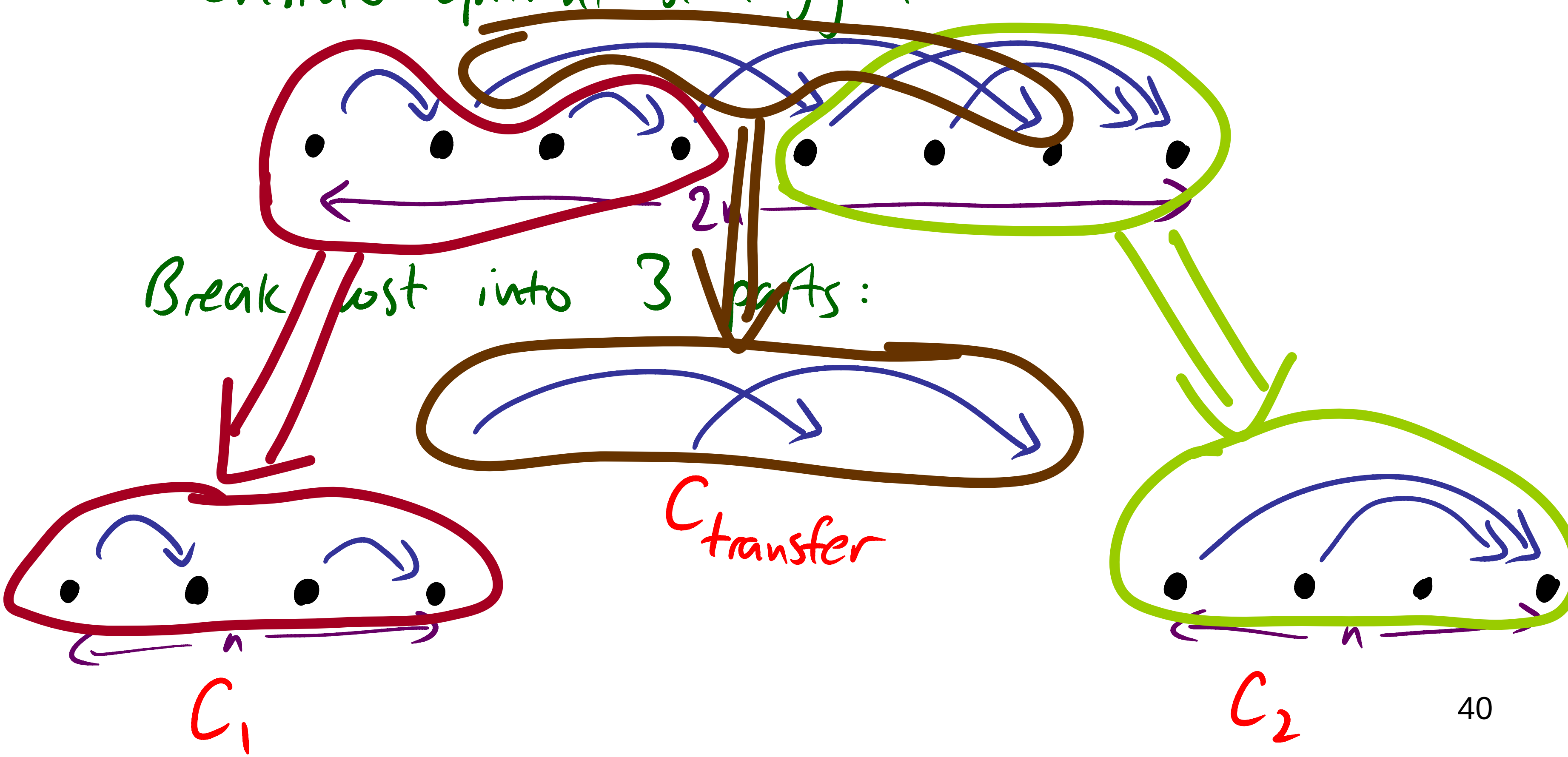


Claim If n is a power of 2, the optimal merge schedule costs at least

$$\min(\alpha, \beta)\, \frac{n}{2} \log_2 n$$

Proof: Induction on n. Assume for n, prove for $2n$.

Consider optimal strategy for $2n$:



Break cost into 3 parts:

Claim: $C_{transfer}$ is at least $\min(\alpha, \beta)\, n$

proof: Either every index file created in 1st half gets rewritten in 2nd half

$\Rightarrow$ additional merging cost of at least αn

or some index file created in 1st half is not rewritten during 2nd half

$\Rightarrow$ additional query cost of at least βn

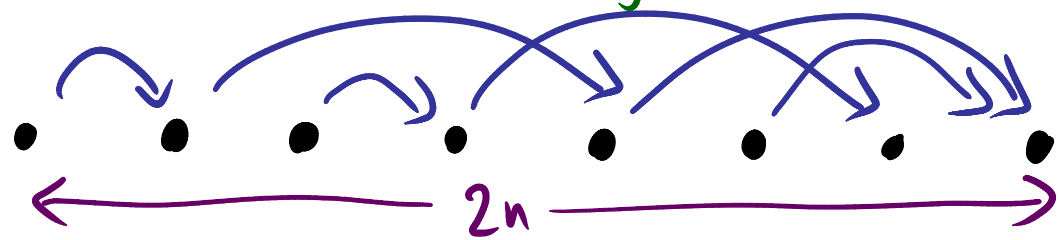
$\square$

Claim If n is a power of 2, the optimal merge schedule costs at least

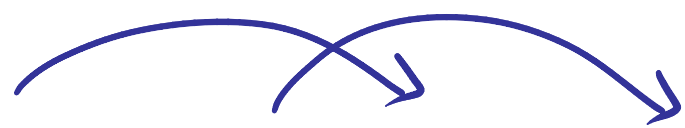
$$\min(\alpha,\beta)\,\frac{n}{2}\log_2 n$$

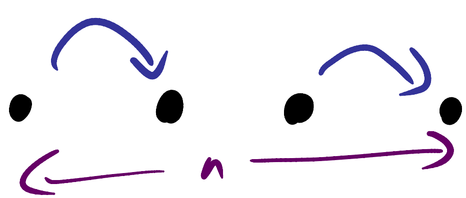
Proof: Induction on n. Assume for n, prove for $2n$.

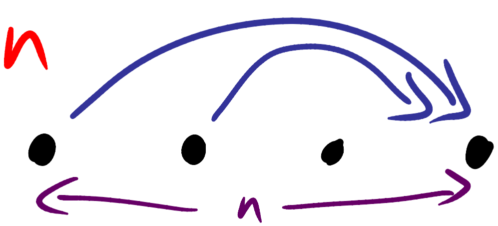
Consider optimal strategy for $2n$:



Break cost into 3 parts:







$$C_{transfer} \geq \min(\alpha,\beta)\,n$$

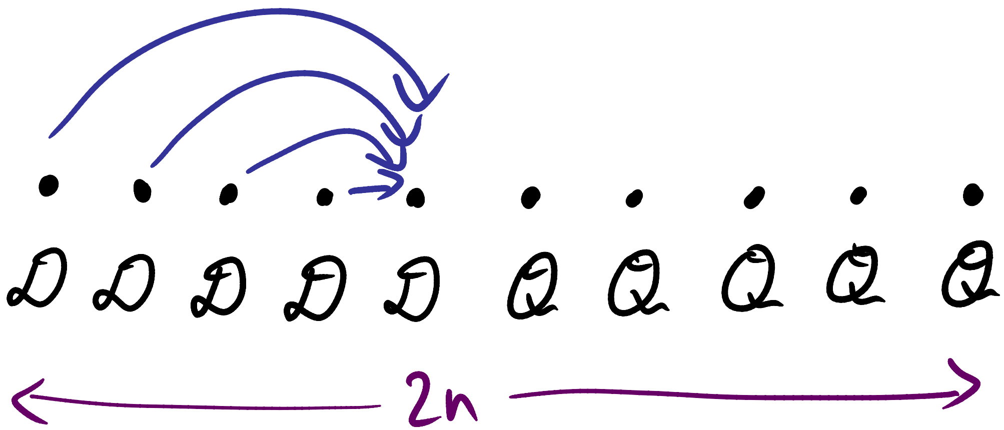
$$C_1 \geq \min(\alpha,\beta)\,\frac{n}{2}\log_2 n$$

$$C_2 \geq \min(\alpha,\beta)\,\frac{n}{2}\log_2 n$$

the index merge scheduling problem: roadmap

1. single index is insufficient for dynamic content – need multiple indexes, and therefore need occasional index merges
2. scheduling merges is related to the cost-distance problem in network construction
3. imposing geometrically decreasing index sizes gives good performance – O(n log n)
4. O(n log n) is optimal, in general
5. cost-balancing approach is more flexible and may be superior – O(n) at times

Some input sequences can be processed in linear cost



total cost: $(2\alpha+\beta)n$

cost-balancing approach is promising

- For every index file, store historic merge & query costs

  i.e. index file described by (s, m, q)

  - s ← size
  - m ← merge cost
  - q ← query cost

- When merging $(s_1, m_1, q_1), (s_2, m_2, q_2), \ldots (s_r, m_r, q_r)$, obtain

$$\left(\sum s_i \, , \, \sum (m_i + \alpha s_i) \, , \, \sum q_i \right)$$

- On each query, $q_i \longmapsto q_i + \beta$

cost-balancing approach is promising

invariant: $m_i \leq q_i$ [vaguely analogous to ski rental]

"balance"

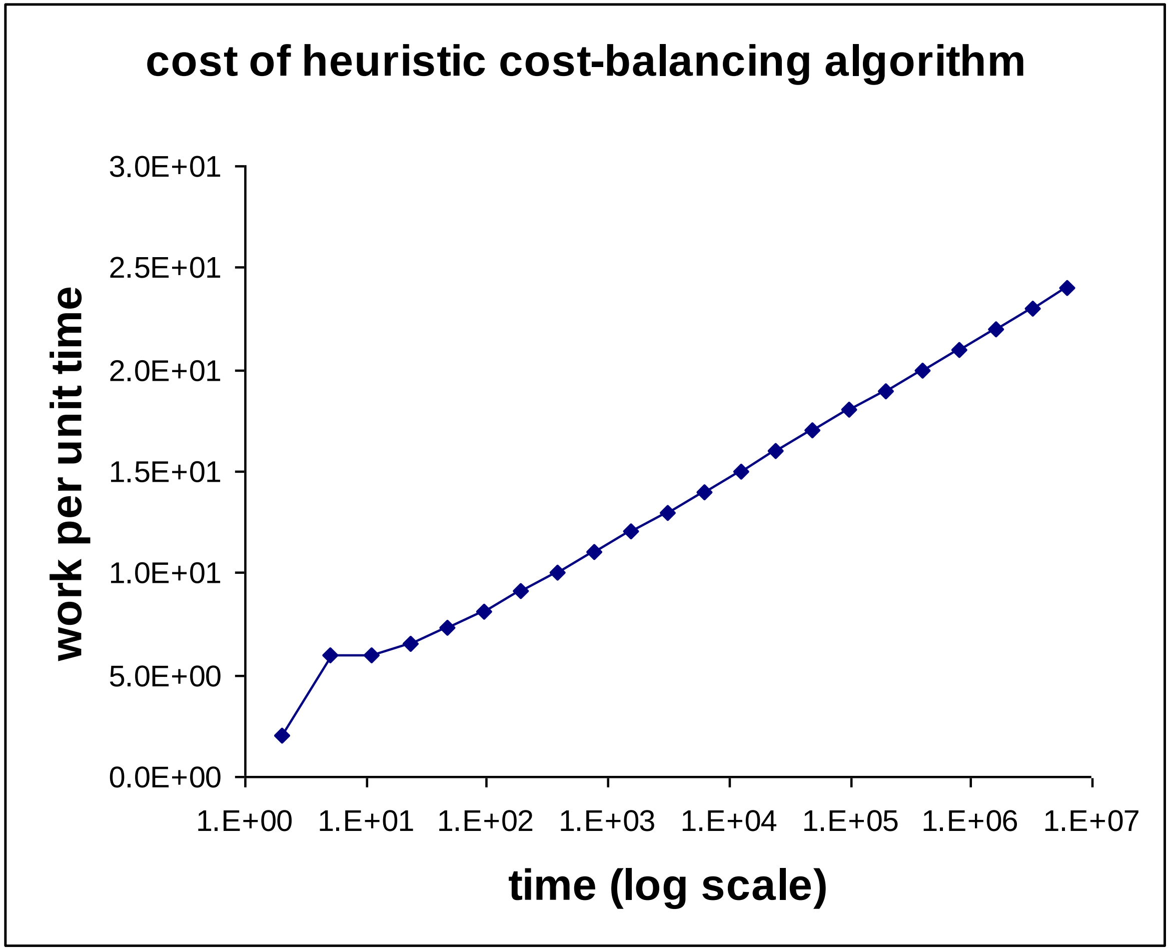
strategy:

- always merge the r smallest index files
- always merge as many as possible without violating balance

cost-balancing approach is promising

- $O(n \log n)$ cost on alternating DQDQ.... input?
- $O(n)$ cost on DDD...DQQ...Q input.
- much more complex/realistic cost modelling is possible

empirical performance of cost-balancing on DQDQ… input is O(n log n)



the index merge scheduling problem: summary

1. single index is insufficient for dynamic content – need multiple indexes, and therefore need occasional index merges
2. scheduling merges is related to the cost-distance problem in network construction
3. imposing geometrically decreasing index sizes gives good performance – O(n log n)
4. O(n log n) is optimal, in general
5. cost-balancing approach is more flexible and may be superior – O(n) at times