Question 1

Consider a B-tree subject to uniformly randomly distributed updates. There are 100 entries per page. The b-tree occupies 70% of the SSD, while the rest is over-provisioned. What write-amplification would you expect?

Model: $$B \cdot (1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P})$$

Where L = logical data size

P = physical SSD capacity

Under what kind of workload would write-amplification for a B-tree be significantly lower?

Question 1

Consider a B-tree subject to uniformly randomly distributed updates. There are 100 entries per page. The b-tree occupies 70% of the SSD, while the rest is over-provisioned. What write-amplification would you expect?

Model: $B \cdot (1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P})$ = $100 \cdot (1 + \frac{1}{2} \cdot \frac{0.7}{1 - 0.7})$ = **167**

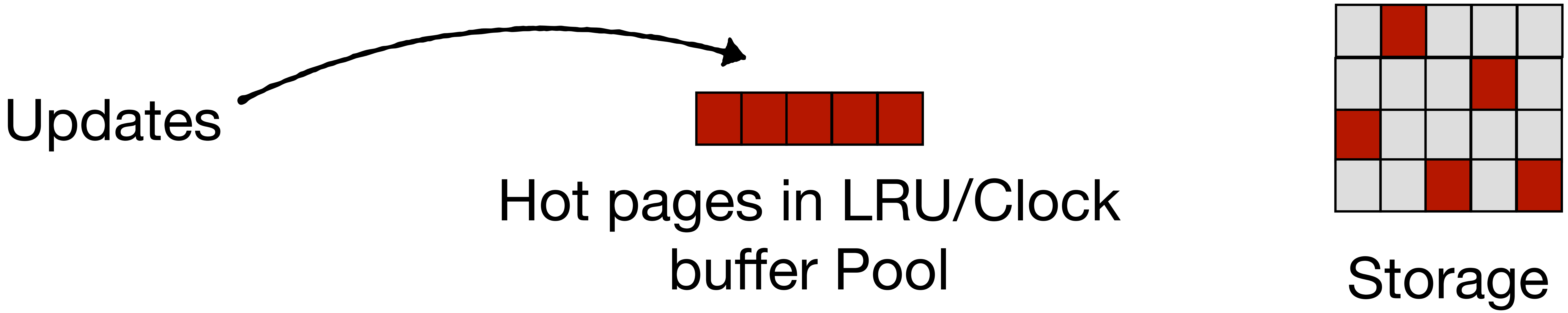
Under what kind of workload would write-amplification for a B-tree be significantly lower?

Question 1

Consider a B-tree subject to uniformly randomly distributed updates. There are 100 entries per page. The b-tree occupies 70% of the SSD, while the rest is over-provisioned. What write-amplification would you expect?

Under what kind of workload would write-amplification for a B-tree be significantly lower?

When a workload exhibits spatial and/or temporal locality: some areas of the logical address space are hot



Question 1

Consider a B-tree subject to uniformly randomly distributed updates. There are 100 entries per page. The b-tree occupies 70% of the SSD, while the rest is over-provisioned. What write-amplification would you expect?

Under what kind of workload would write-amplification for a B-tree be significantly lower?

When a workload exhibits spatial and/or temporal locality: some areas of the logical address space are hot

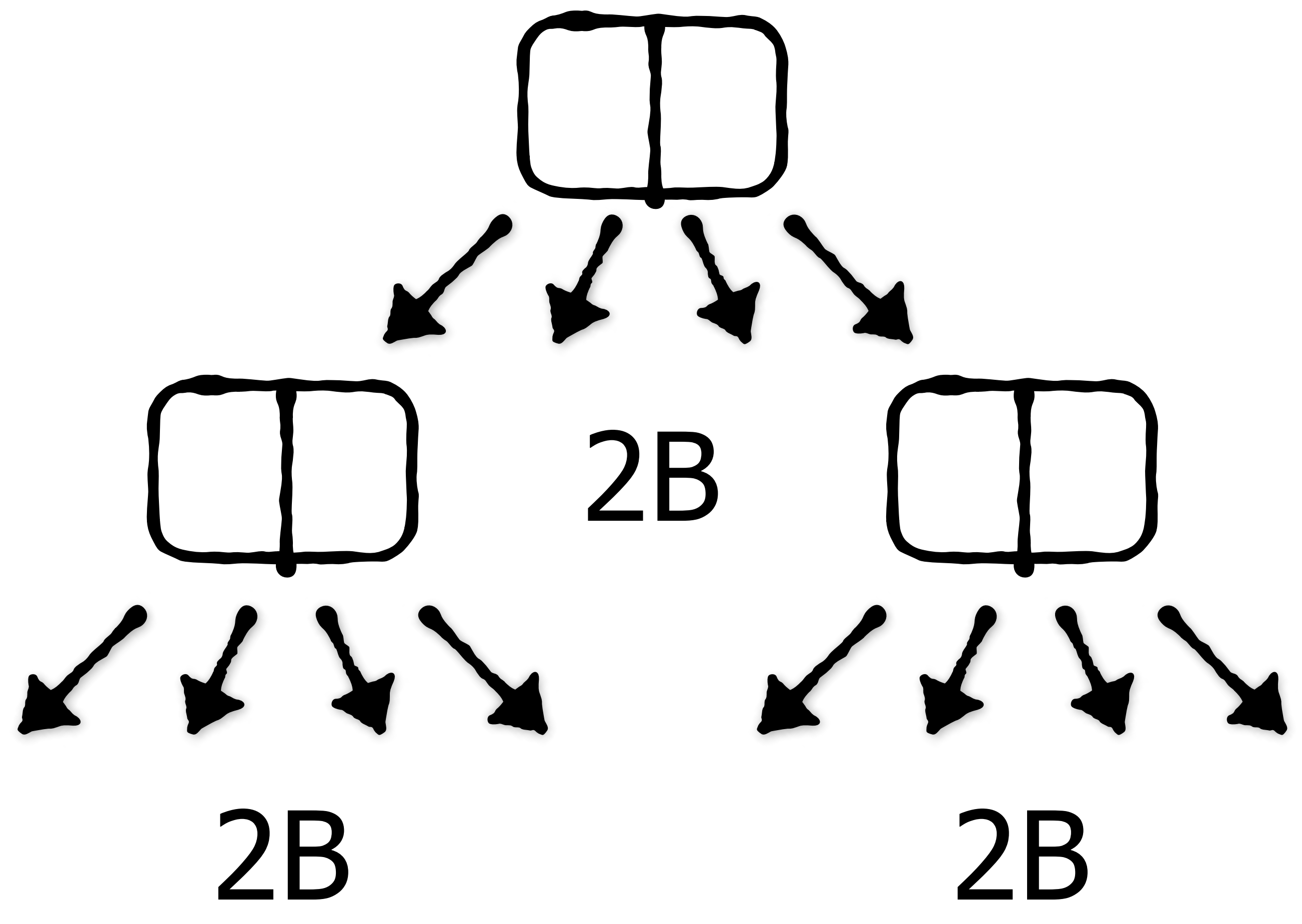
This argument does not hold for extendible hashing as entries that are adjacent logically are distributed randomly in the hash table! This is a disadvantage of hash vs tree indexes.

Question 2

Consider the possibility of making each B-tree node take up two rather than just one flash pages. This can make the tree shallower. Is this a good idea? How about on Disk?

On SSD, cost is measured as # pages accessed

$$= 2*\log_{2B}(N)$$



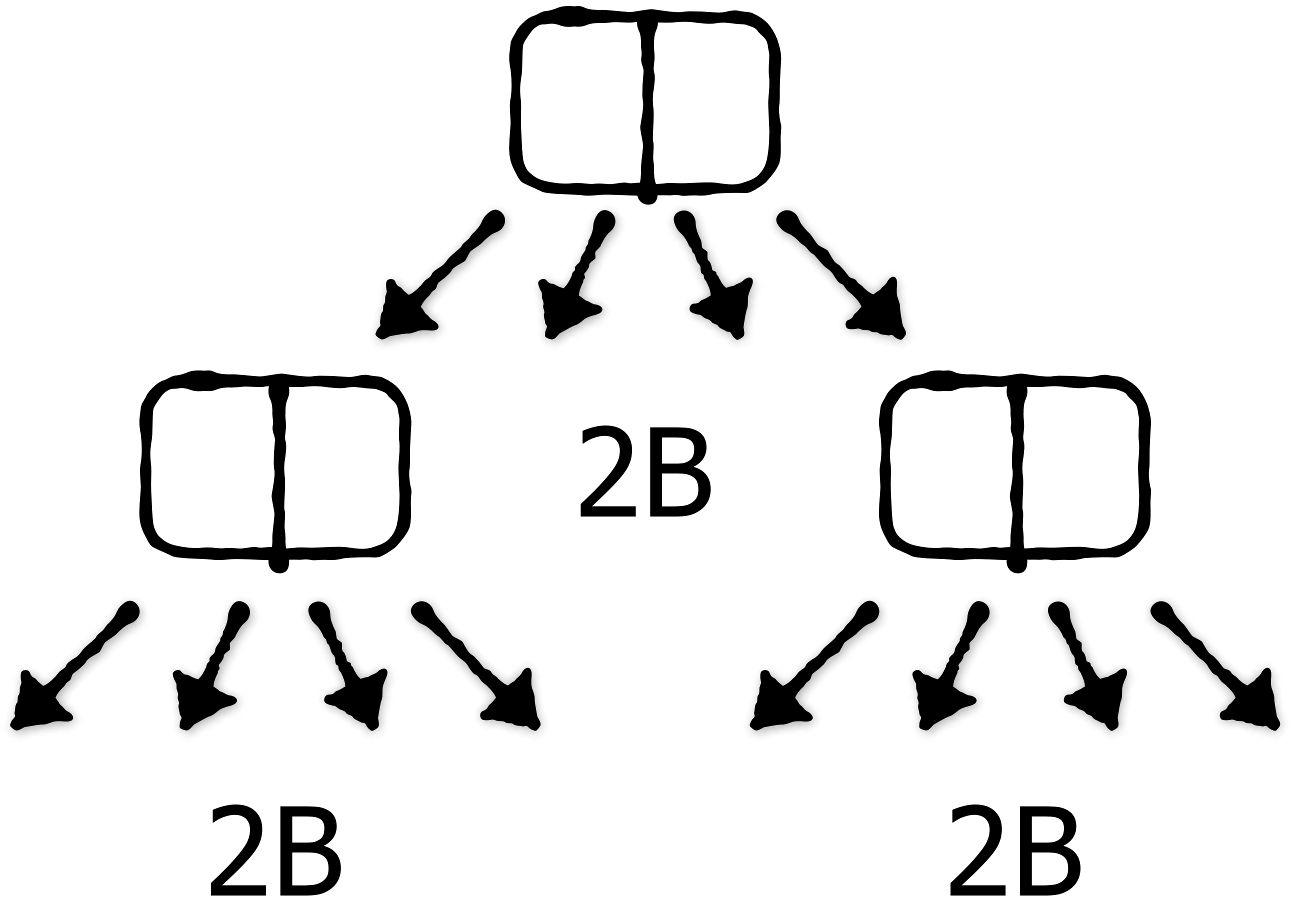
Question 2

Consider the possibility of making each B-tree node take up two rather than just one flash pages. This can make the tree shallower. Is this a good idea? How about on Disk?

On SSD, cost is measured as # pages accessed

$$2 * \log_{2B}(N) \leq \log_{B}(N)$$

Condition for being cheaper than standard B-tree



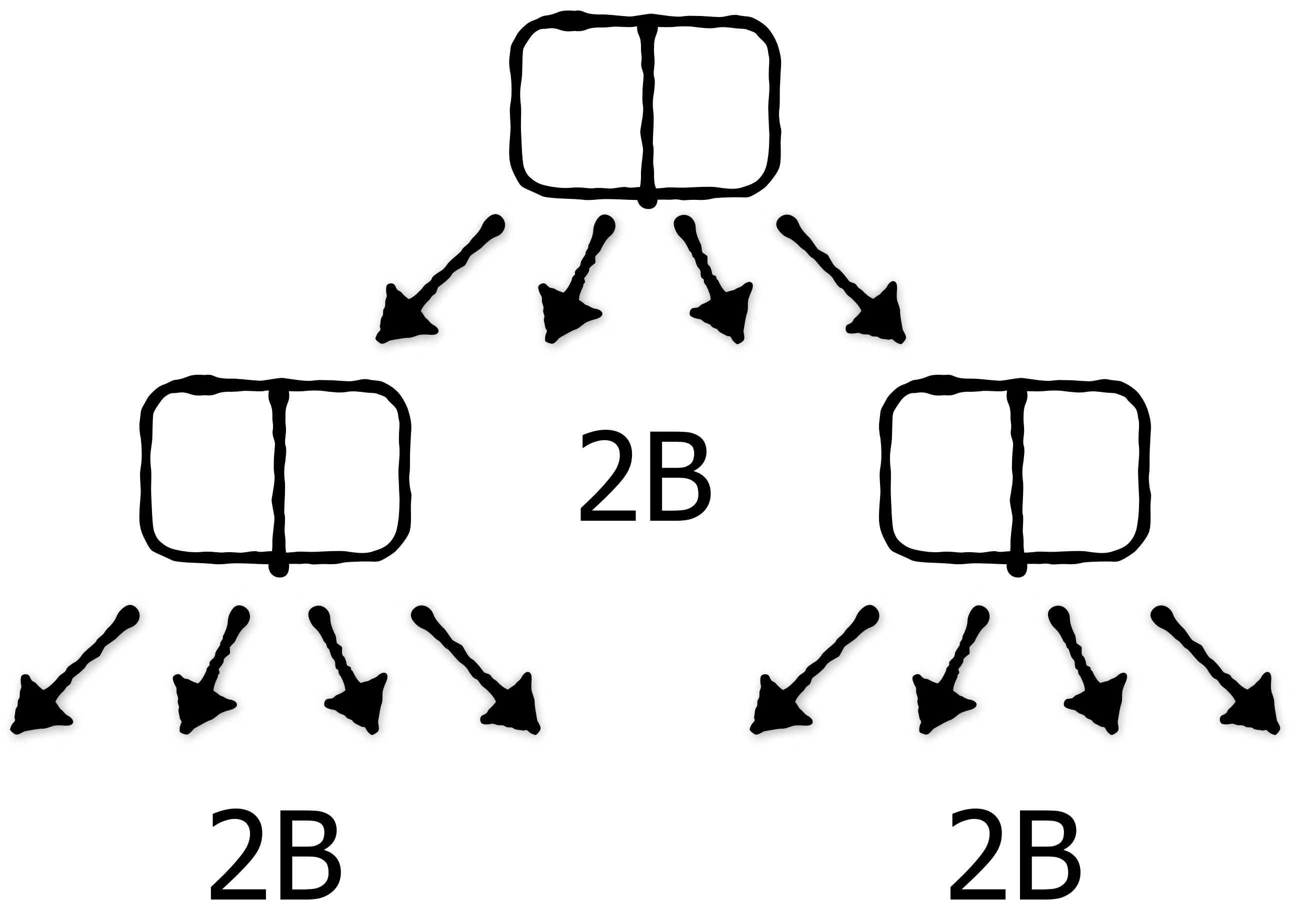
Question 2

Consider the possibility of making each B-tree node take up two rather than just one flash pages. This can make the tree shallower. Is this a good idea? How about on Disk?

On SSD, cost is measured as # pages accessed

$$2 * \log_{2B}(N) \leq \log_{B}(N)$$

Simplifies to: $\mathbf{B \leq 2}$



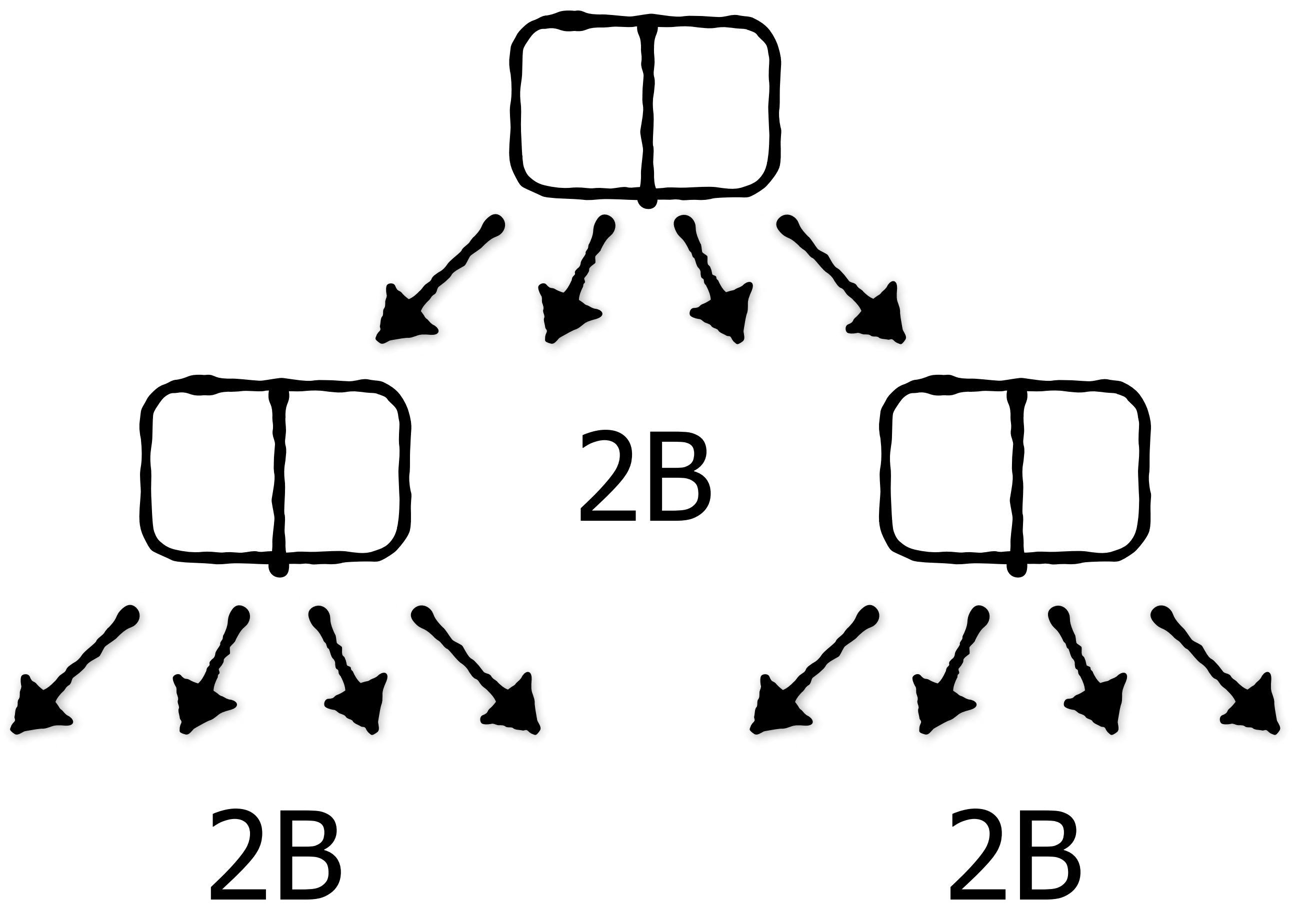
Question 2

Consider the possibility of making each B-tree node take up two rather than just one flash pages. This can make the tree shallower. Is this a good idea? How about on Disk?

On SSD, cost is measured as # pages accessed

$$2 * \log_{2B}(N) \leq \log_{B}(N)$$

Simplifies to: $B \leq 2$



But B is typically larger. So it's not generally a good idea.

Question 2

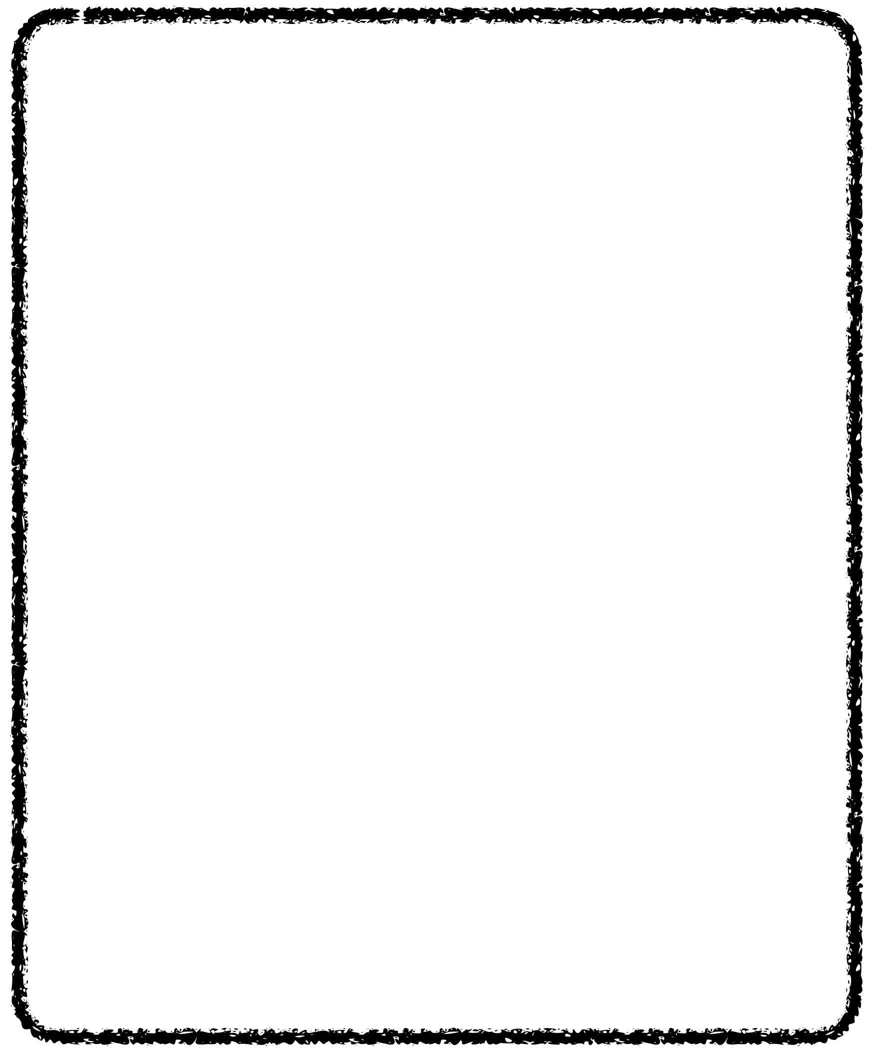
Consider the possibility of making each B-tree node take up two rather than just one flash pages. This can make the tree shallower. Is this a good idea? How about on Disk?

On disk, seek & rotational delay dominate, while data transfer is negligible. So this is a good idea (enlarging the node size by a multiplicative factor of B will approx. halve the depth). Likely incur diminishing returns beyond that.

Question 3

Consider a table with columns A, B and C. Suppose we employ buffered inserts at a cost of $O(1/B)$ each.

A B C



50%	Select * from table where A = "…"	Return 1 row each
50%	Insert (, ,)	

Should we employ a B-tree index on any of the columns? Estimate the overall I/O cost of both queries with and without out it.

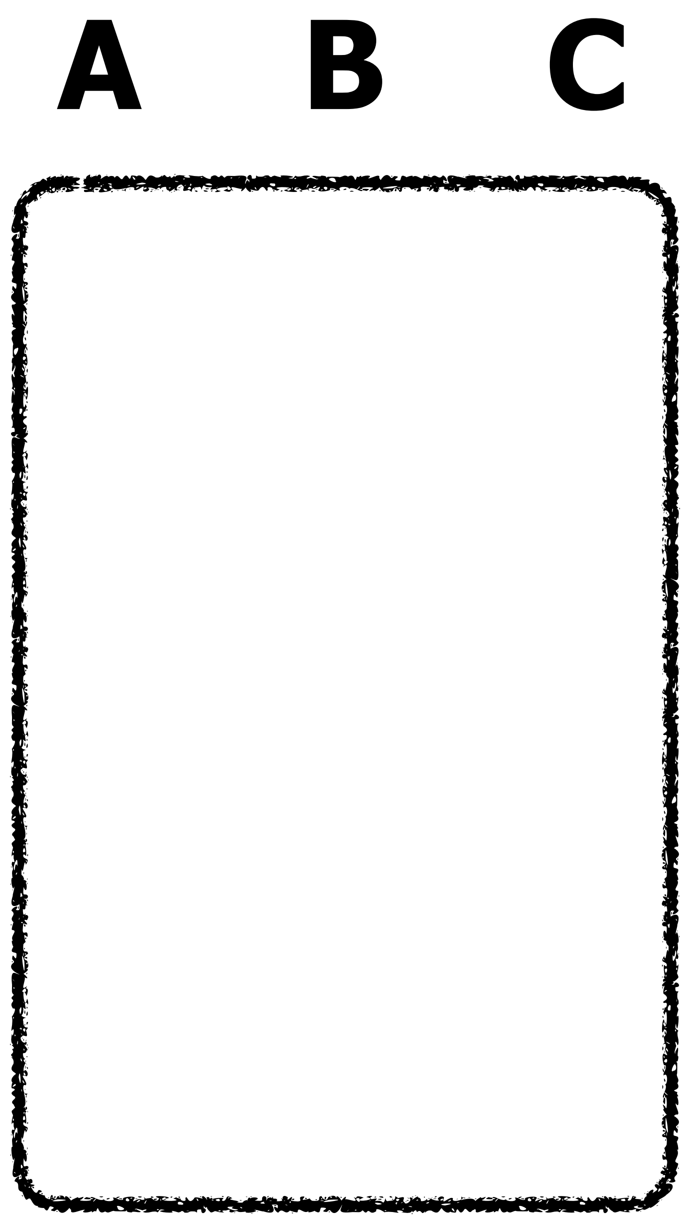
Indexing A significantly reduces overall costs.

I/O cost without index: $0.5 * N/B + 0.5 * 1/B \approx N/B$

I/O cost with index: $0.5 * \log_B N + 0.5 * \log_B N \approx \log_B N$

Question 4

Consider a table with columns A, B and C.

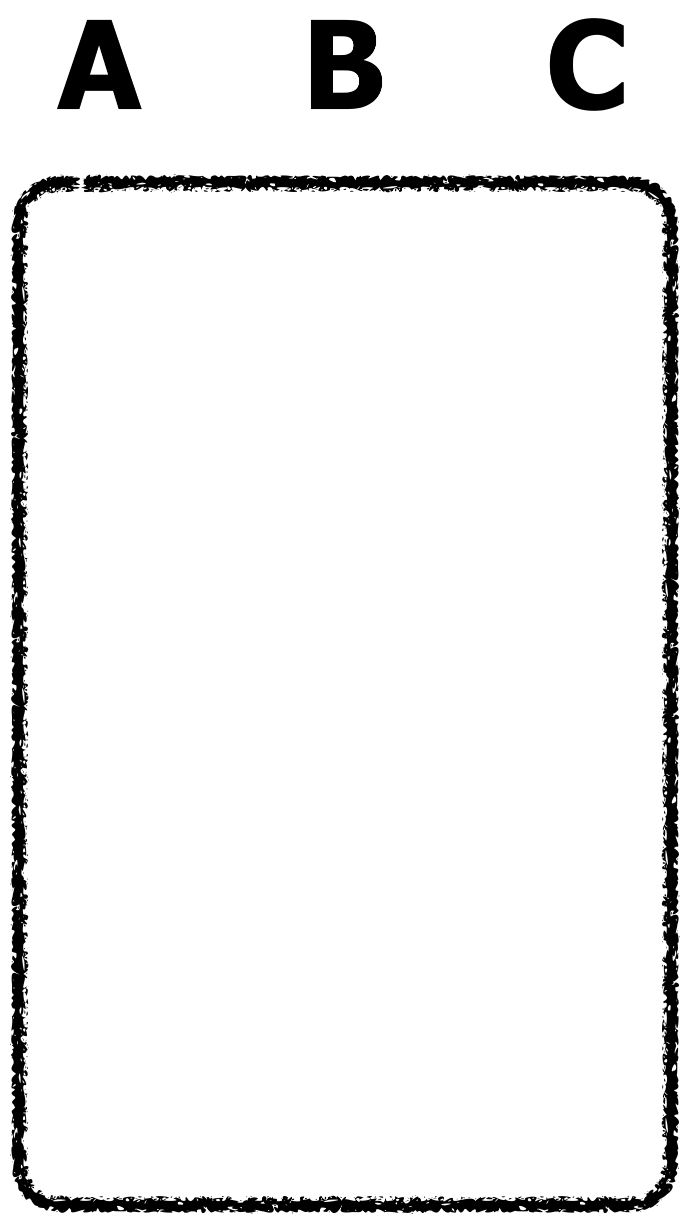


50%	Select A from table where A = "…"	Returns 1 row each
50%	Select * from table where B > x and B < y	Returns avg. S=10 rows

How should we index this table? B-tree or extendible hashing? Clustered vs. unclustered? Estimate worst-case I/O cost with your plan for each query with these indexes assuming $N=2^{40}$ and $B=2^{10}$

Question 4

Consider a table with columns A, B and C.



50%	Select A from table where A = "…"	Returns 1 row each
50%	Select * from table where B > x and B < y	Returns avg. S=10 rows

How should we index this table? B-tree or extendible hashing? Clustered vs. unclustered? Estimate worst-case I/O cost with your plan for each query with these indexes assuming $N=2^{40}$ and $B=2^{10}$

Clustered B-tree on B

log_(N) + S/B

4 + 1

Extendible Hash table on A

1-2, assuming directory is in memory and data is evenly distributed