Tutorial 3

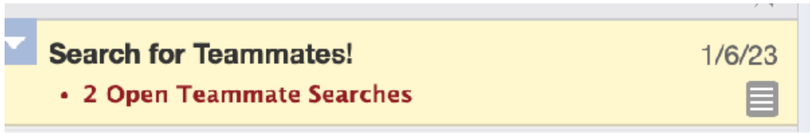
Database System Technology - Indexing

Niv Dayan - 25 Jan, 2023

Groups

Most of you are grouped up.

One new person in the course is looking for a team!



Agenda

More on SSD write-amplification

4 exercises on indexing

Introduce step 2 of the project

Modeling SSD Write-Amp More Precisely

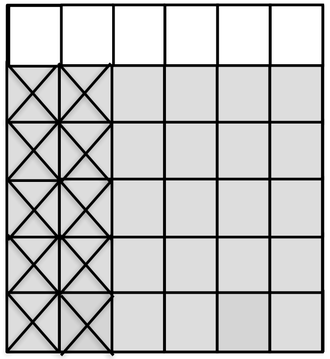
Consider an SSD for which the logical address space size is a fraction of x of the physical capacity.

Modeling SSD Write-Amp More Precisely

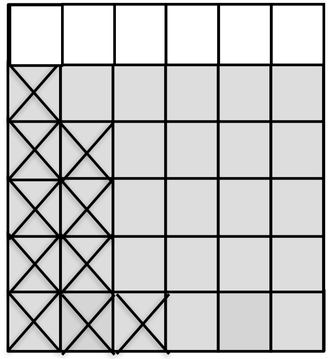
Consider an SSD for which the logical address space size is a fraction of x of the physical capacity.

Worst case write-amplification (WA) occurs when each erase-unit has same number of invalid pages.

Worst case



Non-Worst Case



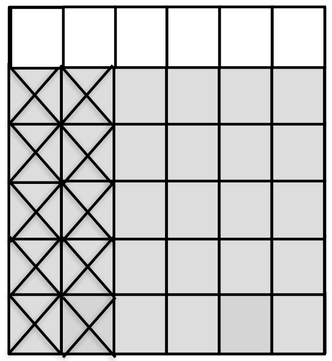
← Best target

Modeling SSD Write-Amp More Precisely

Consider an SSD for which the logical address space size is a fraction of x of the physical capacity.

Worst case write-amplification (WA) occurs when each erase-unit has same number of invalid pages.

Worst case



SSD WA Approx.
from last week

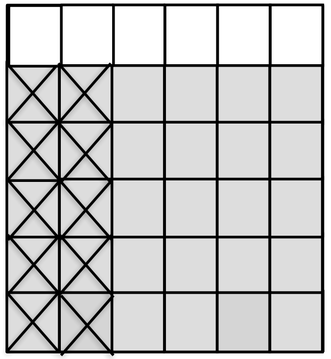
$$\frac{1}{1-x}$$

Modeling SSD Write-Amp More Precisely

Consider an SSD for which the logical address space size is a fraction of x of the physical capacity.

Worst case write-amplification (WA) occurs when each erase-unit has same number of invalid pages.

Worst case



Tutorial 2 Question 1 Fri

Two questions regarding the garbage collection multiplicative factor:1. Can we get more intuition on how we arrived at t

SSD WA Approx. from last week

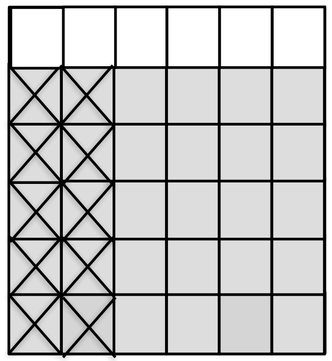
$$\frac{1}{1-x}$$

Modeling SSD Write-Amp More Precisely

Consider an SSD for which the logical address space size is a fraction of x of the physical capacity.

Worst case write-amplification (WA) occurs when each erase-unit has same number of invalid pages.

Worst case



More precise model

$$1 + \frac{x}{1 - x}$$

SSD WA Approx. from last week

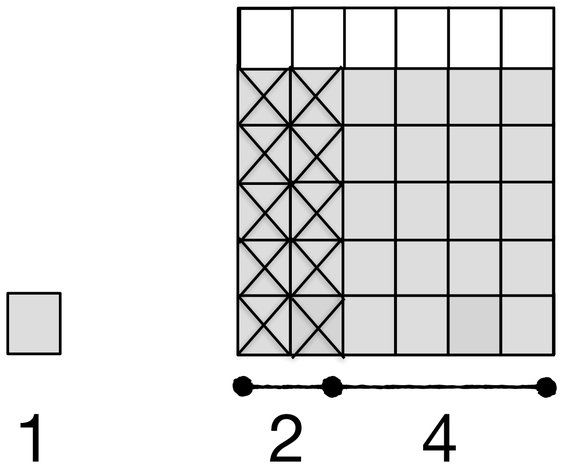
$$\frac{1}{1 - x}$$

Modeling SSD Write-Amp More Precisely

Consider an SSD for which the logical address space size is a fraction of x of the physical capacity.

Worst case write-amplification (WA) occurs when each erase-unit has same number of invalid pages.

Worst case



More precise model

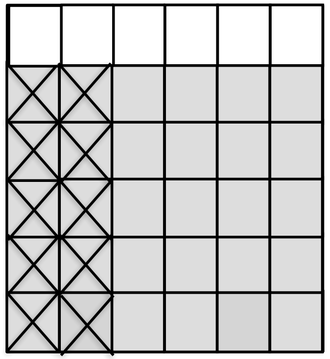
$$1 + \frac{x}{1 - x}$$

Modeling SSD Write-Amp More Precisely

Consider an SSD for which the logical address space size is a fraction of x of the physical capacity.

Worst case write-amplification (WA) occurs when each erase-unit has same number of invalid pages.

Worst case



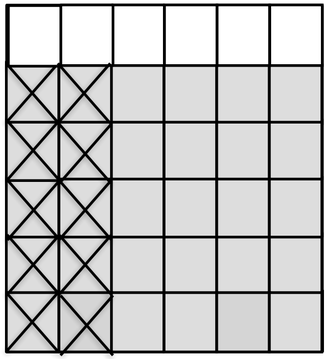
$$1 + \frac{4}{2} = 3$$

More precise model

$$1 + \frac{x}{1 - x}$$

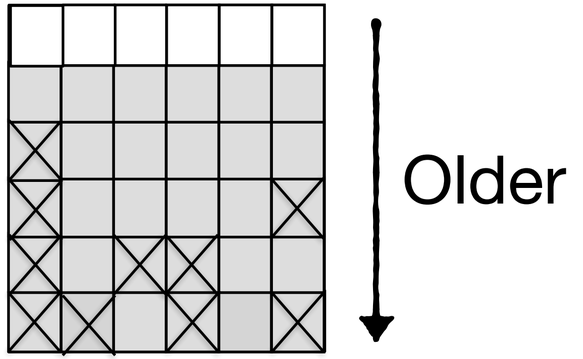
Modeling SSD Write-Amp More Precisely

But the worst-case hardly ever happens in practice



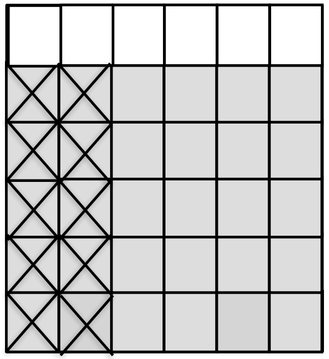
$$1 + \frac{x}{1 - x}$$

In practice, erase units written longer ago have more invalid pages, so GC is cheaper

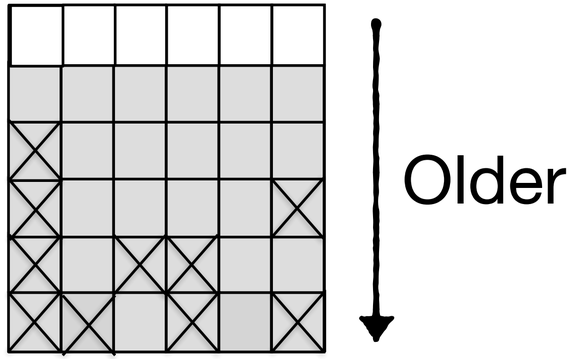


Modeling SSD Write-Amp More Precisely

Cost model assuming uniformly randomly distributed writes?

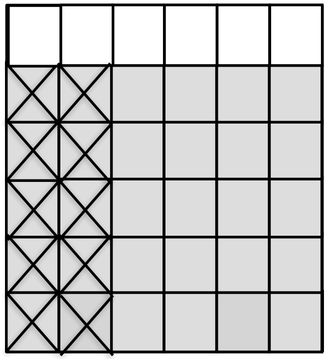


$$1 + \frac{x}{1 - x}$$



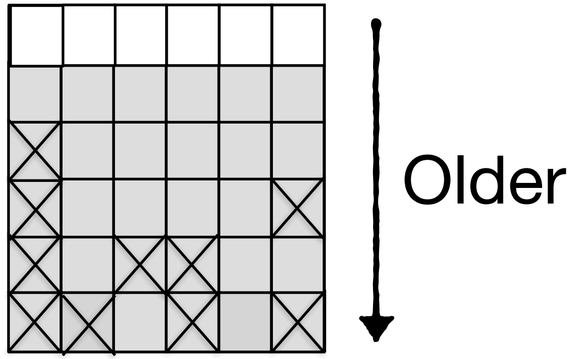
Modeling SSD Write-Amp More Precisely

Cost model assuming uniformly randomly distributed writes?

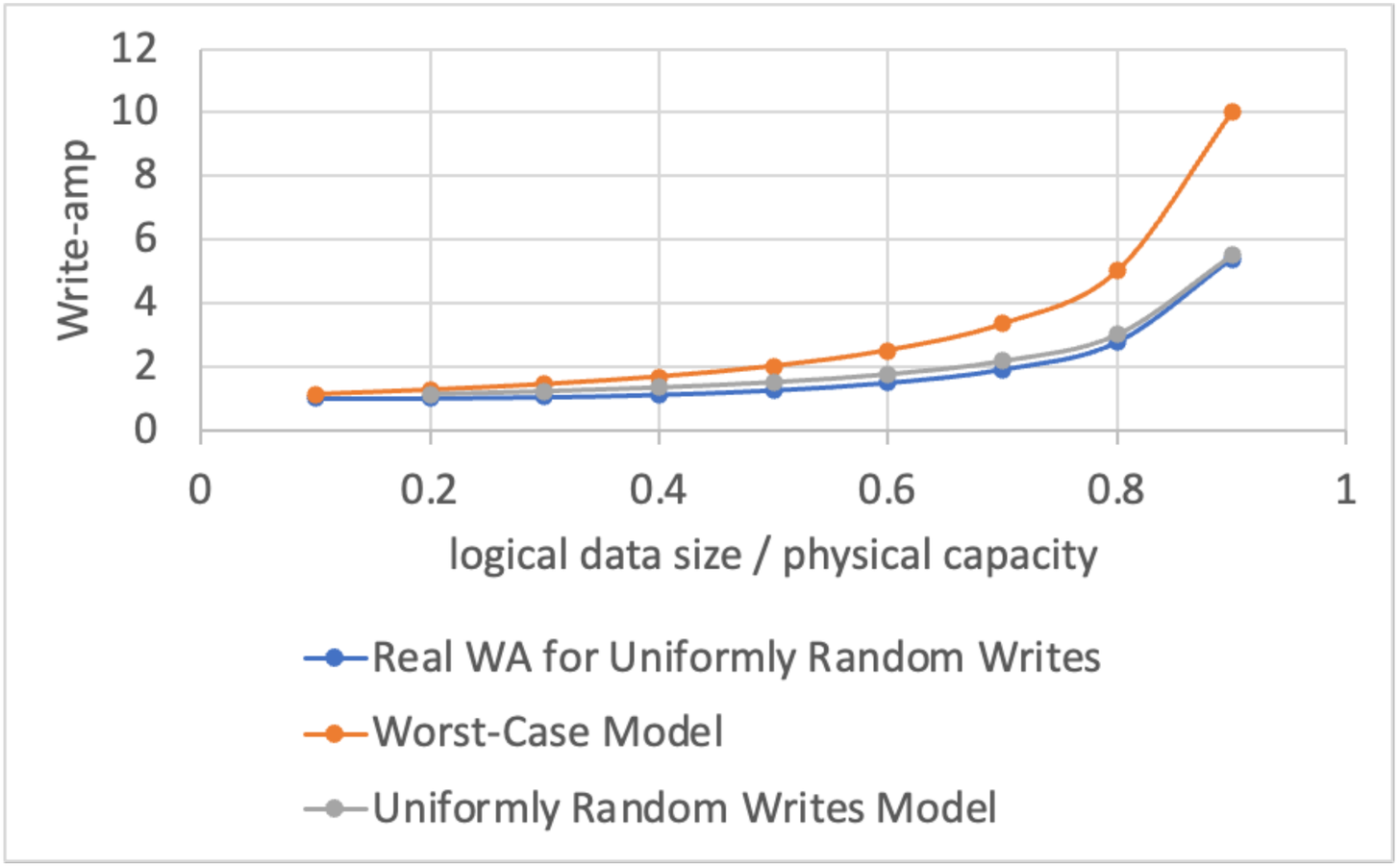


$$1 + \frac{x}{1 - x}$$

$$1 + \frac{1}{2} \cdot \frac{x}{1 - x}$$



Modeling SSD Write-Amp More Precisely

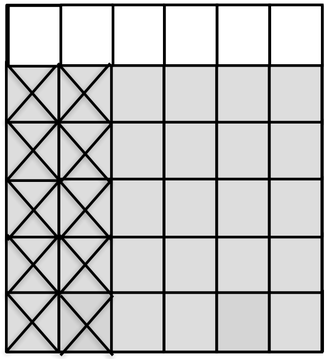


$$1 + \frac{x}{1 - x}$$

$$1 + \frac{1}{2} \cdot \frac{x}{1 - x}$$

How does this model change assuming there are B entries per page, and each update modifies B entries?

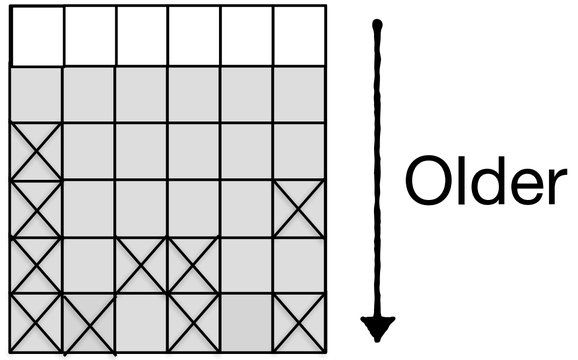
Worst-case



$$1 + \frac{x}{1-x}$$

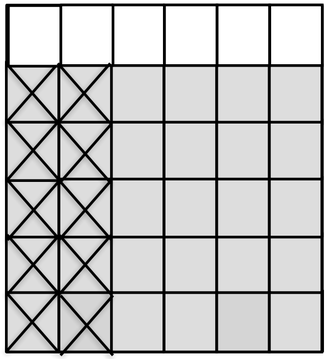
Uniformly Random Case

$$1 + \frac{1}{2} \cdot \frac{x}{1-x}$$



How does this model change assuming there are B entries per page, and each update modifies B entries?

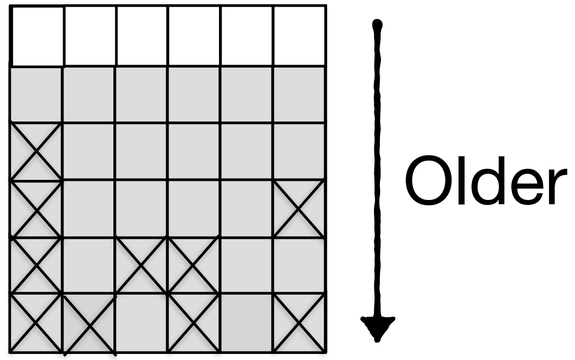
Worst-case



$$B \cdot (1 + \frac{x}{1-x})$$

Uniformly Random Case

$$B \cdot (1 + \frac{1}{2} \cdot \frac{x}{1-x})$$



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 empty. What write-amplification would you expect?

Model: $$B \cdot (1 + \frac{1}{2} \cdot \frac{x}{1-x})$$

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 empty. What write-amplification would you expect?

Model: $B \cdot (1 + \frac{1}{2} \cdot \frac{x}{1-x})$ **= 216**

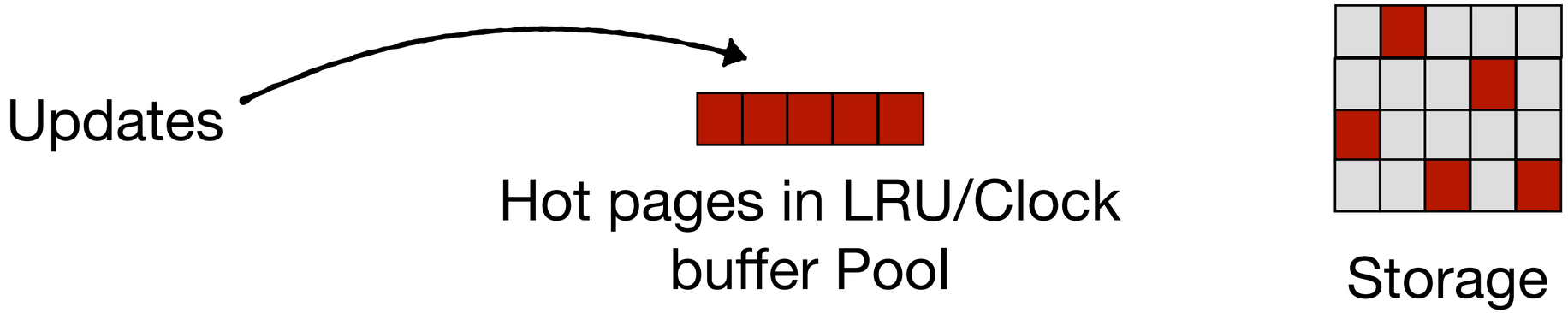
Under what kind of workload would write-amplification for a B-tree be significantly lower?

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 empty. 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 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 empty. What write-amplification would you expect?

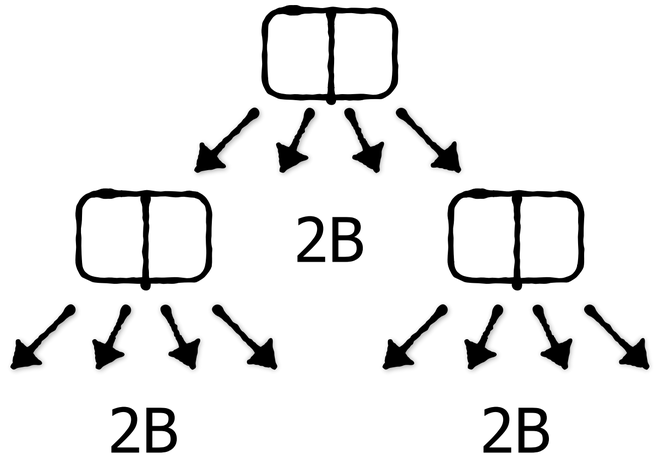
Under what kind of workload would write-amplification for a B-tree be significantly lower?

When a workload exhibits spatial 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 (i.e., 8KB rather than 4KB). This can make the tree shallower. Is this a good idea for flash? How about on Disk?

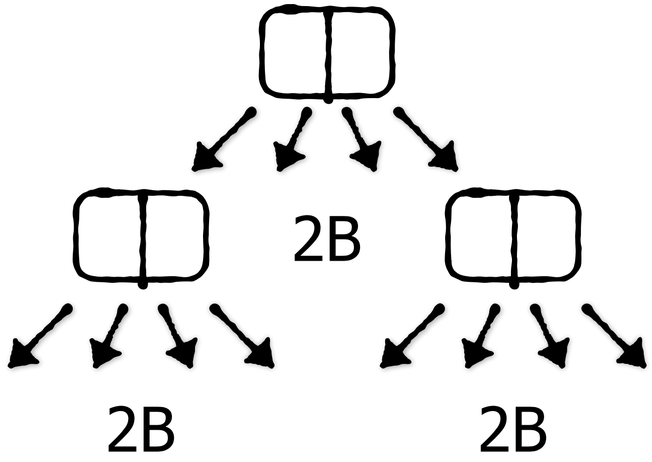


Question 2

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

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



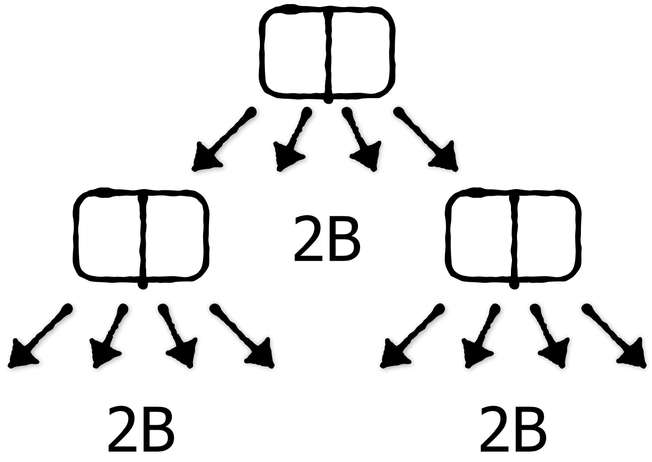
Question 2

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



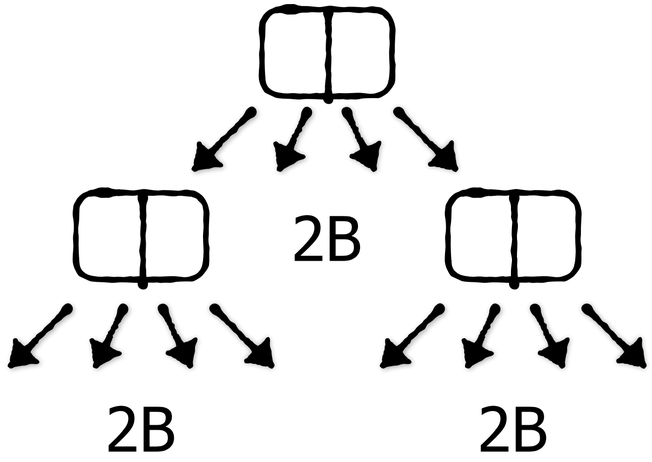
Question 2

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: $B \leq 2$



Question 2

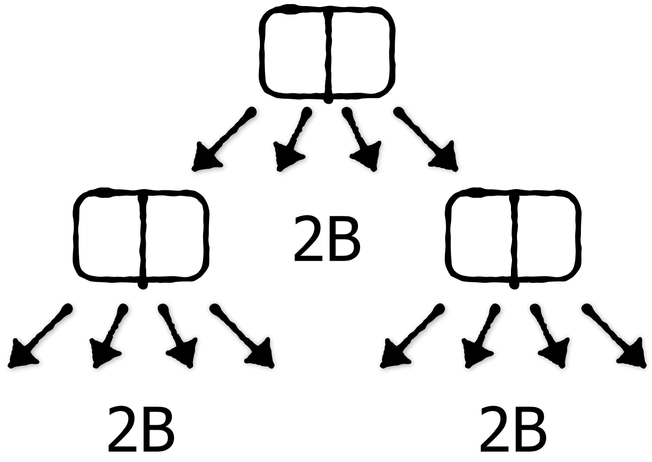
Consider the possibility of making each B-tree node take up two rather than just one flash pages. This can make the tree shallower. Is this a good idea? How about on Disk?

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.



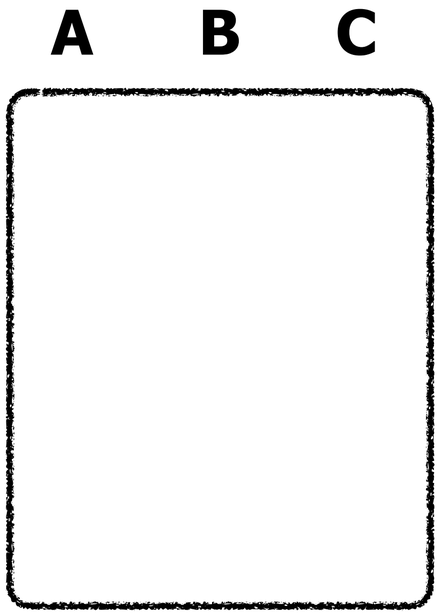
Question 2

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 factor of B will reduce depth by one. 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.



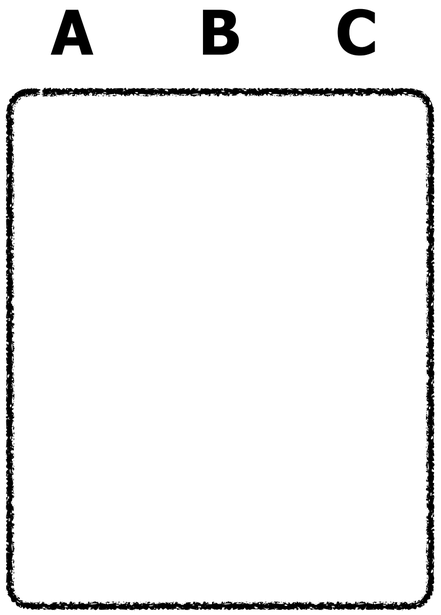
The workload consists of:

50% Select * from table where A = "…" Return 1 row each

50% Insert (, ,)

Question 3

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



The workload consists of:

50% Select * from table where A = "…" Return 1 row each

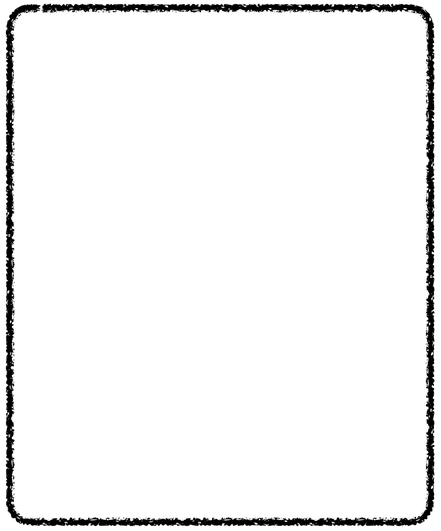
50% Insert (, ,)

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.

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.

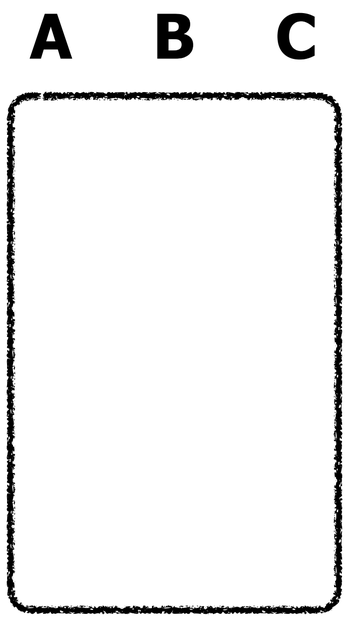
Indexing A significantly reduces overall costs.

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.



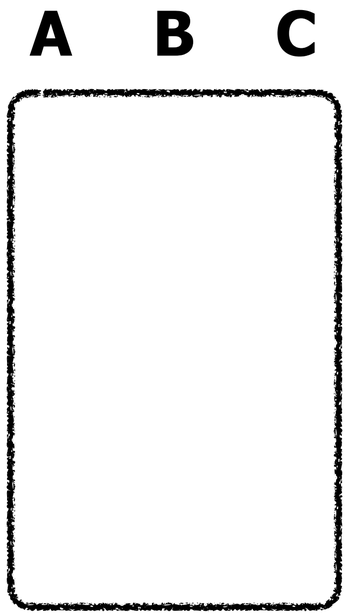
50% Select A from table where A = "…" Returns 1 row each

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