Indexing Tutorial

Database System Technology

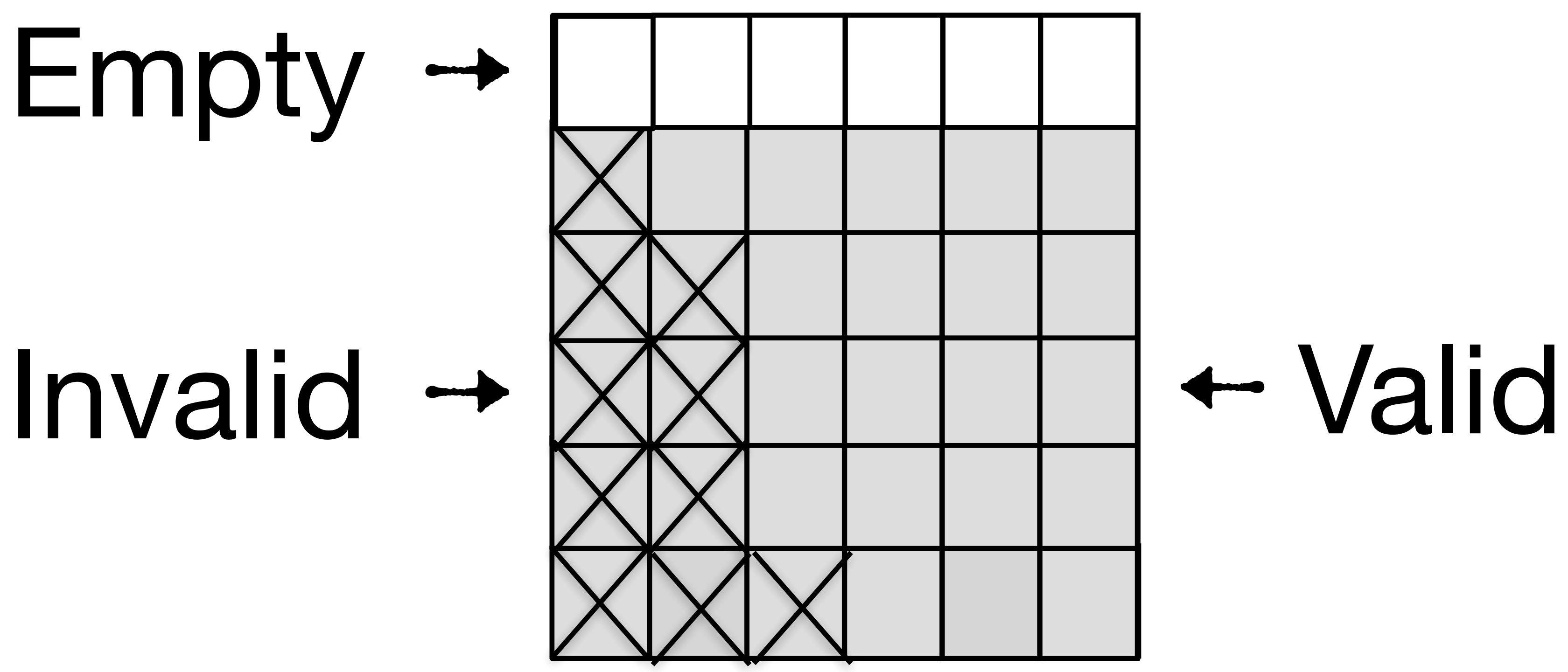
Niv Dayan

Modeling SSD Write-Amp

Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.

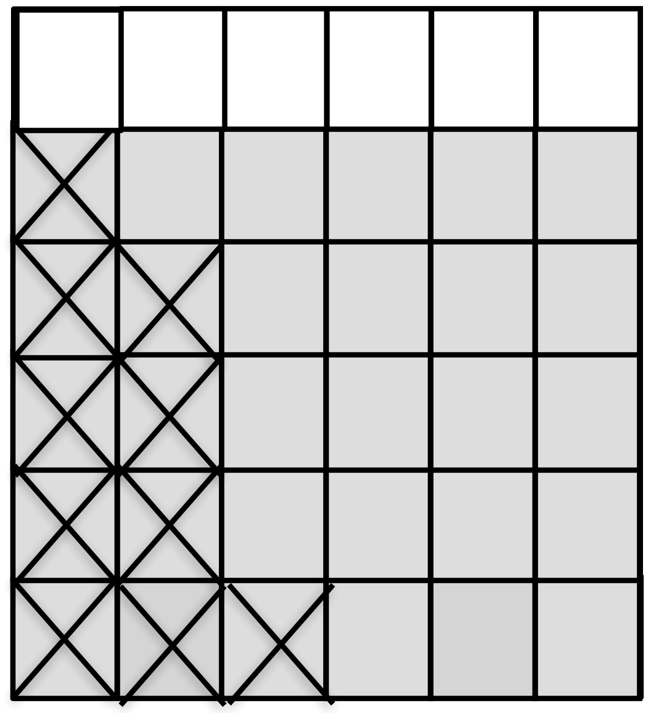
Modeling SSD Write-Amp

Consider an SSD with logical address space size L and physical capacity P.
The usable capacity is a fraction of L/P of the overall capacity.



Modeling SSD Write-Amp

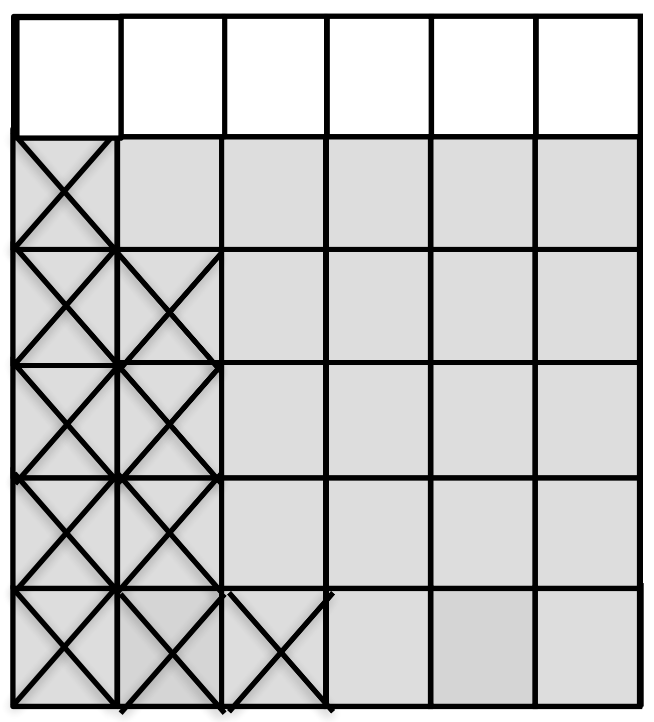
Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.



$$L/P = \frac{\%Valid}{\%Empty + \%Valid + \%Invalid}$$

Modeling SSD Write-Amp

Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.



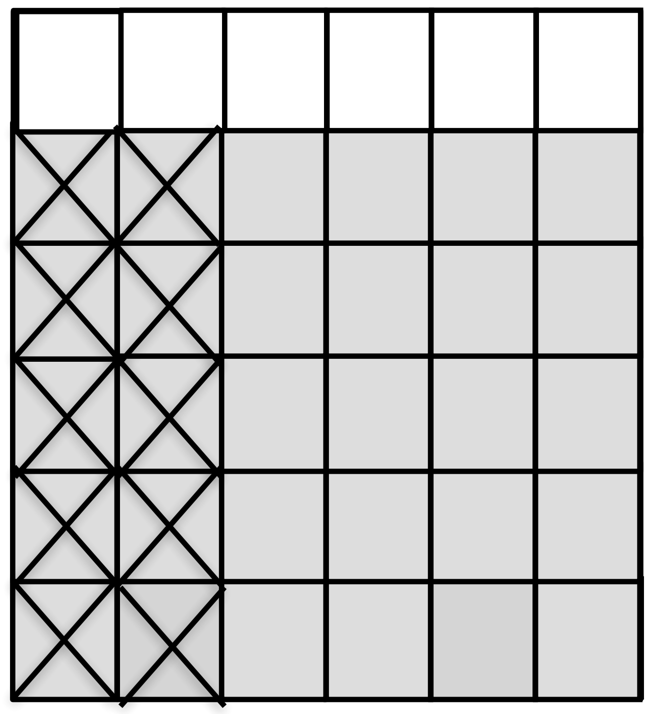
$$L/P = \frac{\%Valid}{\%Valid + \%Invalid}$$

Modeling SSD Write-Amp

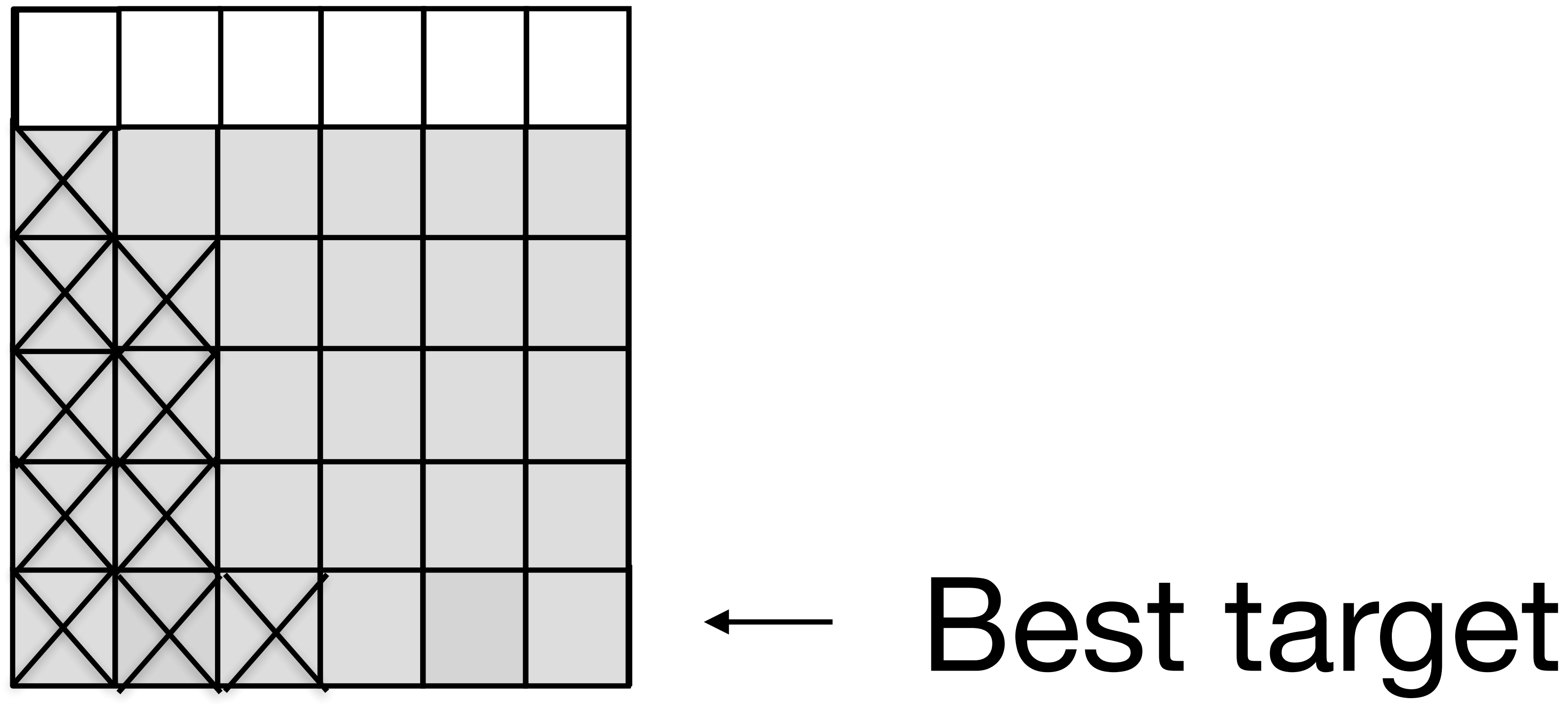
Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.

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

Worst case



Non-Worst Case

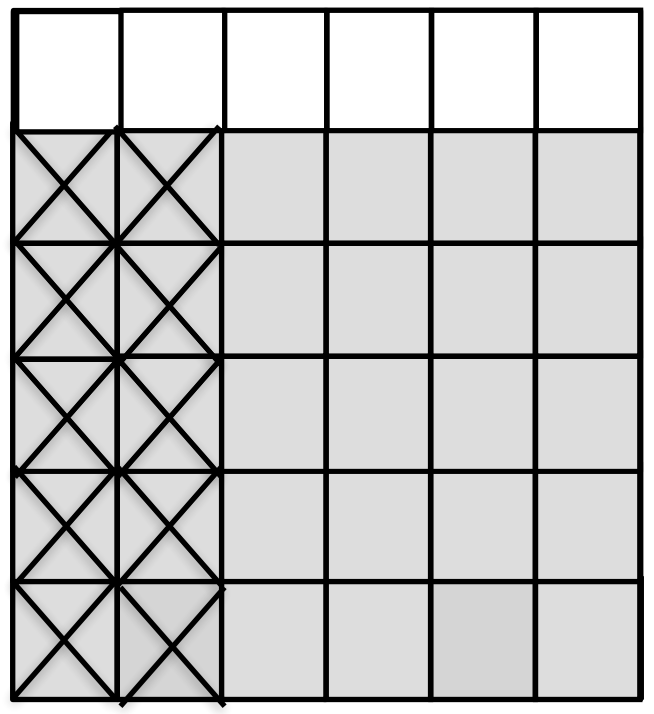


Modeling SSD Write-Amp

Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.

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

Worst case



SSD WA Approx

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

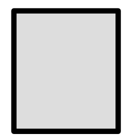
x is avg. fraction of valid pages in the garbage-collected block

Modeling SSD Write-Amp

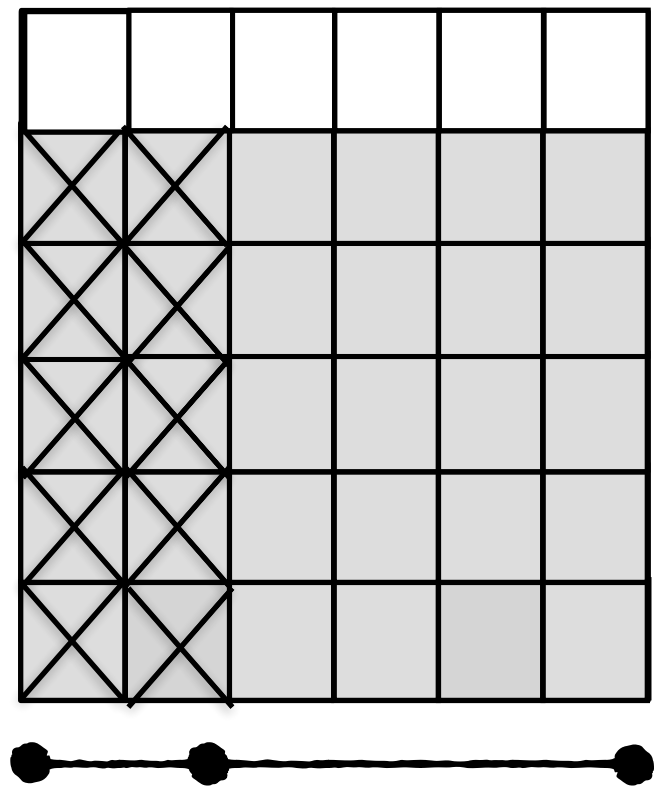
Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.

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

Worst case



1



2/6 4/6

SSD WA Approx

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

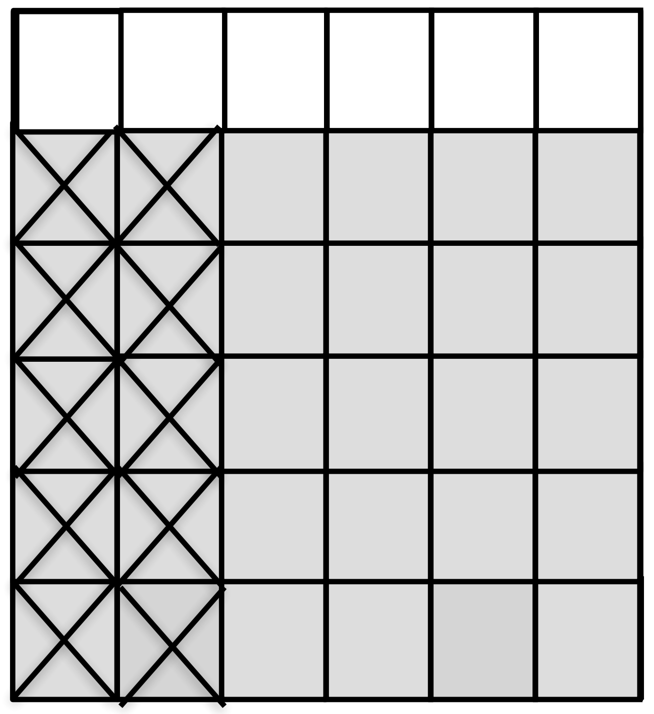
x is avg. fraction of valid pages in the garbage-collected block

Modeling SSD Write-Amp

Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.

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

Worst case



$$1 + \frac{4/6}{2/6}$$

SSD WA Approx

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

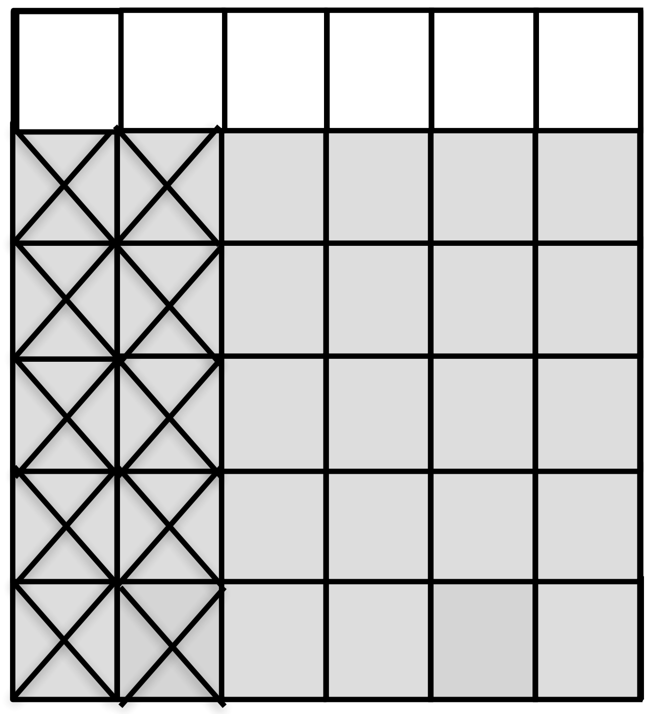
x is avg. fraction of valid pages in the garbage-collected block

Modeling SSD Write-Amp

Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.

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

Worst case



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

SSD WA Approx

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

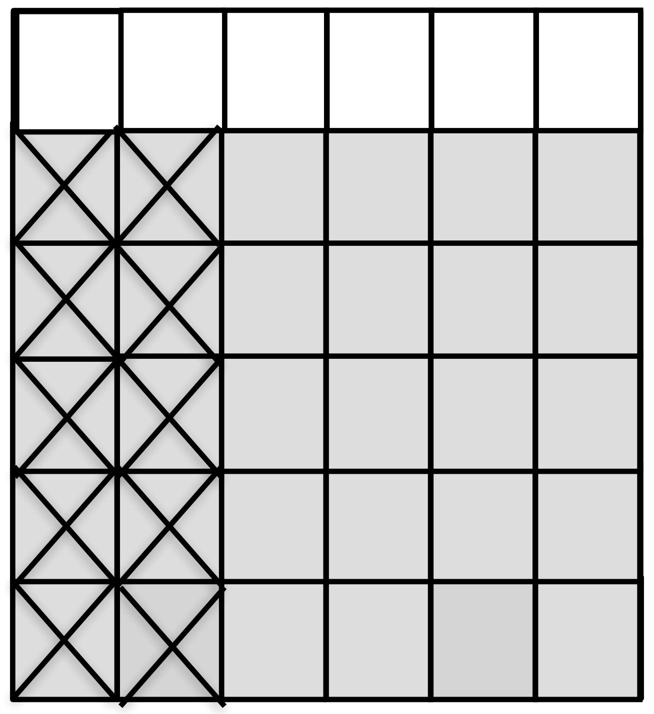
x is avg. fraction of valid pages in the garbage-collected block

Modeling SSD Write-Amp

Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.

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

Worst case



SSD WA Approx

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

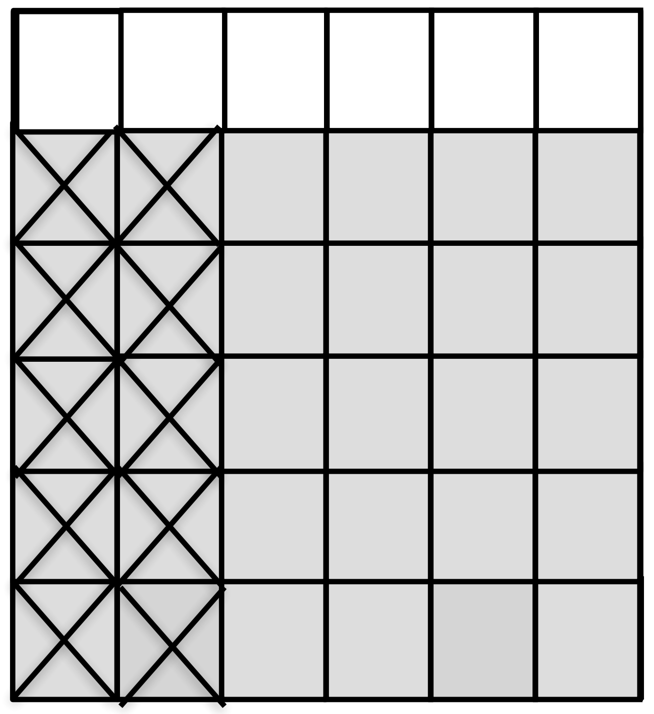
In worst case: x = valid / total = 4/6 = L/P

Modeling SSD Write-Amp

Consider an SSD with logical address space size L and physical capacity P. The usable capacity is a fraction of L/P of the overall capacity.

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

Worst case



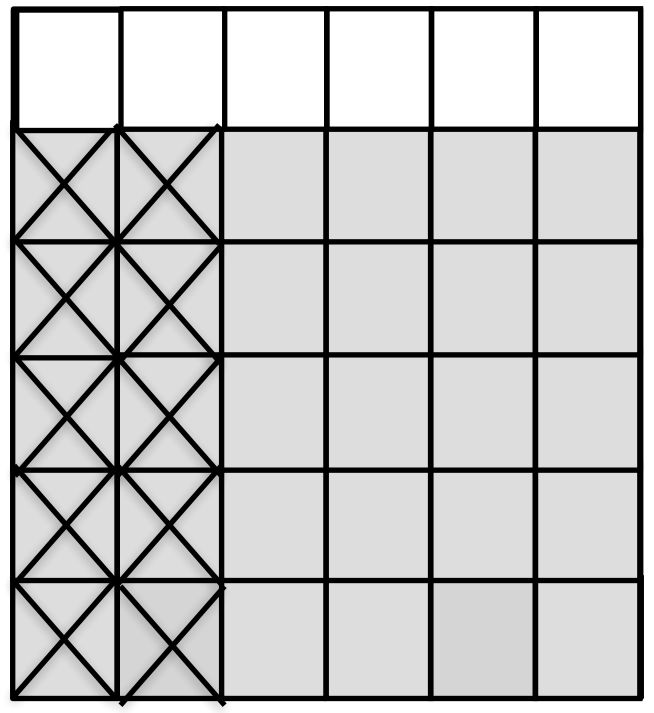
SSD WA Approx

$$1 + \frac{L/P}{1 - L/P}$$

In worst case: x = valid / total = 4/6 = L/P

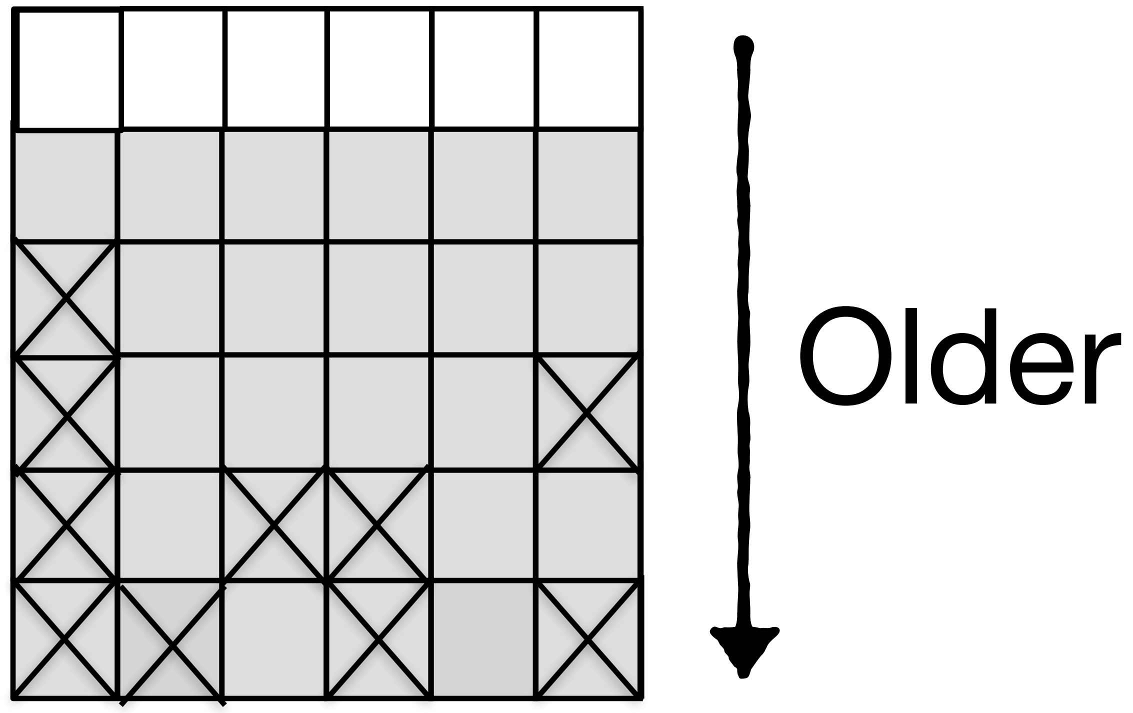
Modeling SSD Write-Amp

But the worst-case hardly ever happens in practice



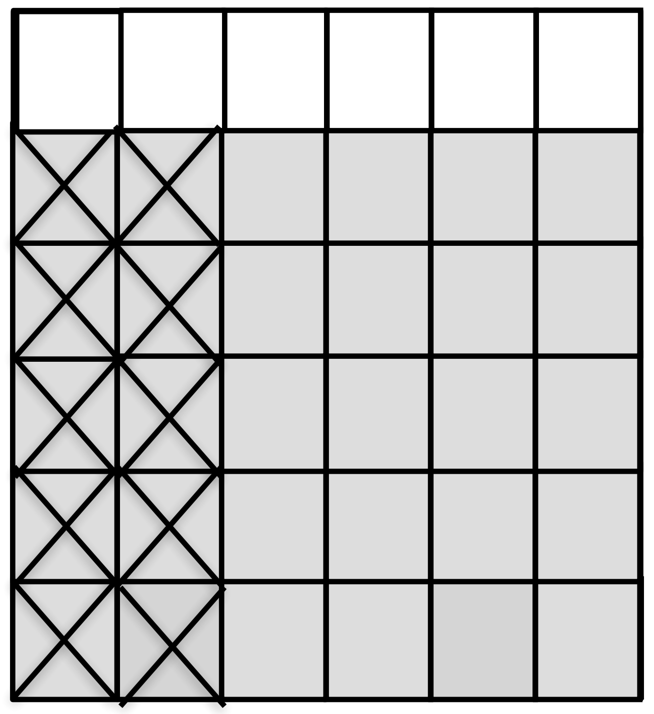
$$1 + \frac{L/P}{1 - L/P}$$

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

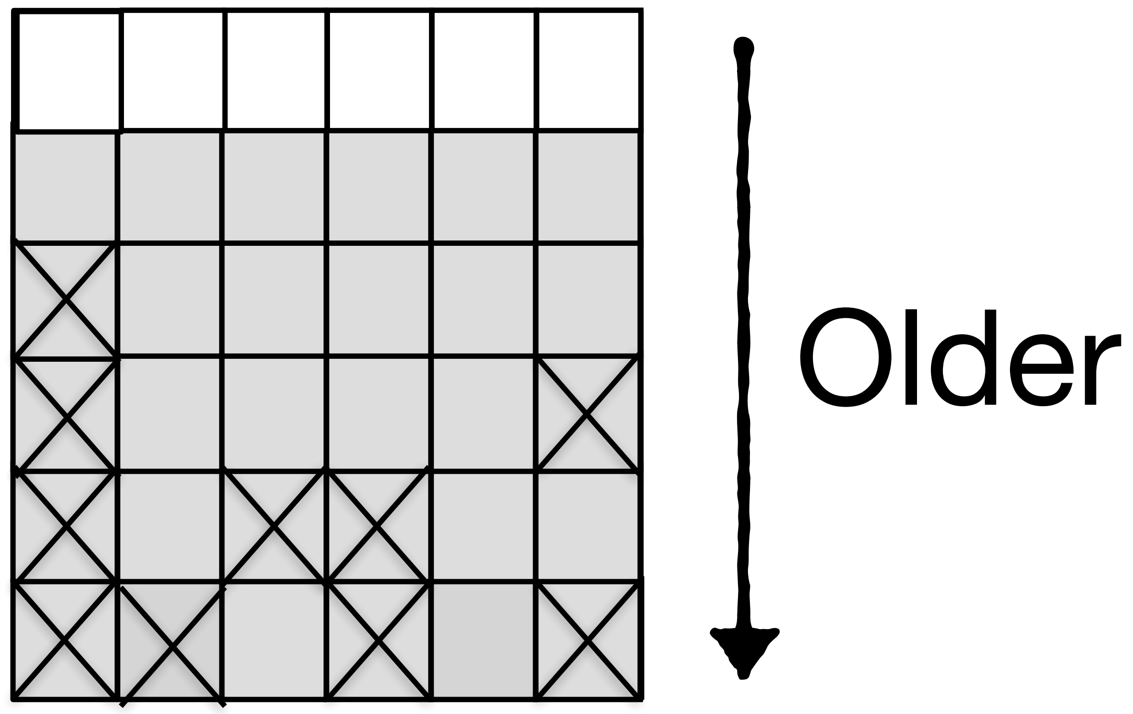


Modeling SSD Write-Amp

Cost model assuming uniformly randomly distributed writes?

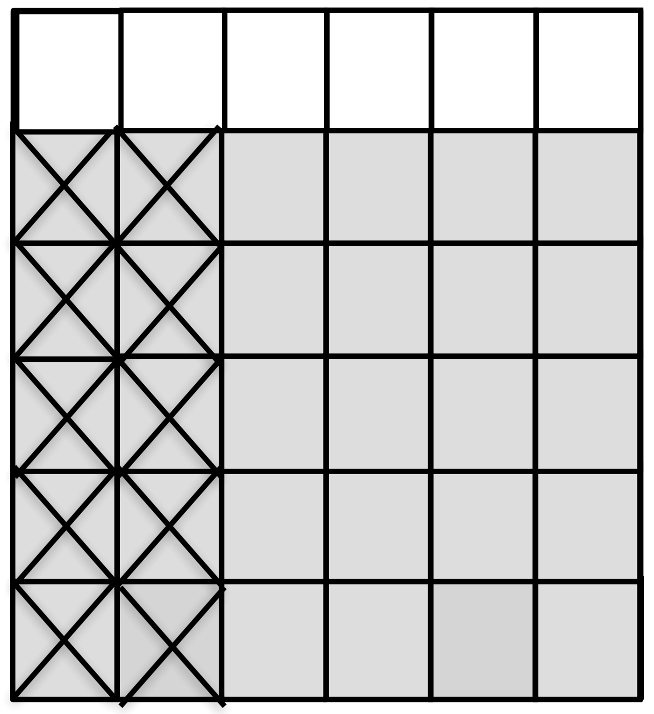


$$1 + \frac{L/P}{1 - L/P}$$



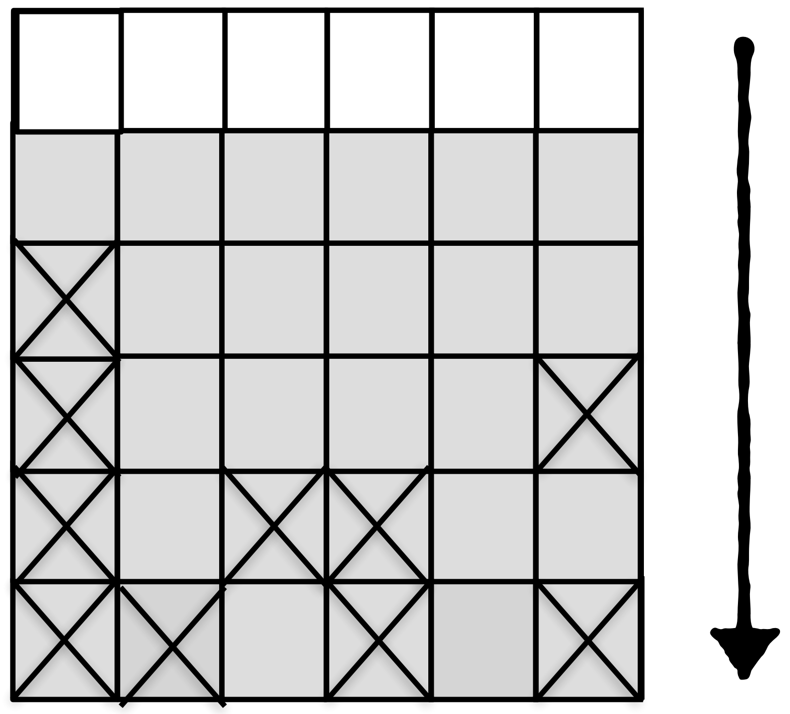
Modeling SSD Write-Amp

Cost model assuming uniformly randomly distributed writes?



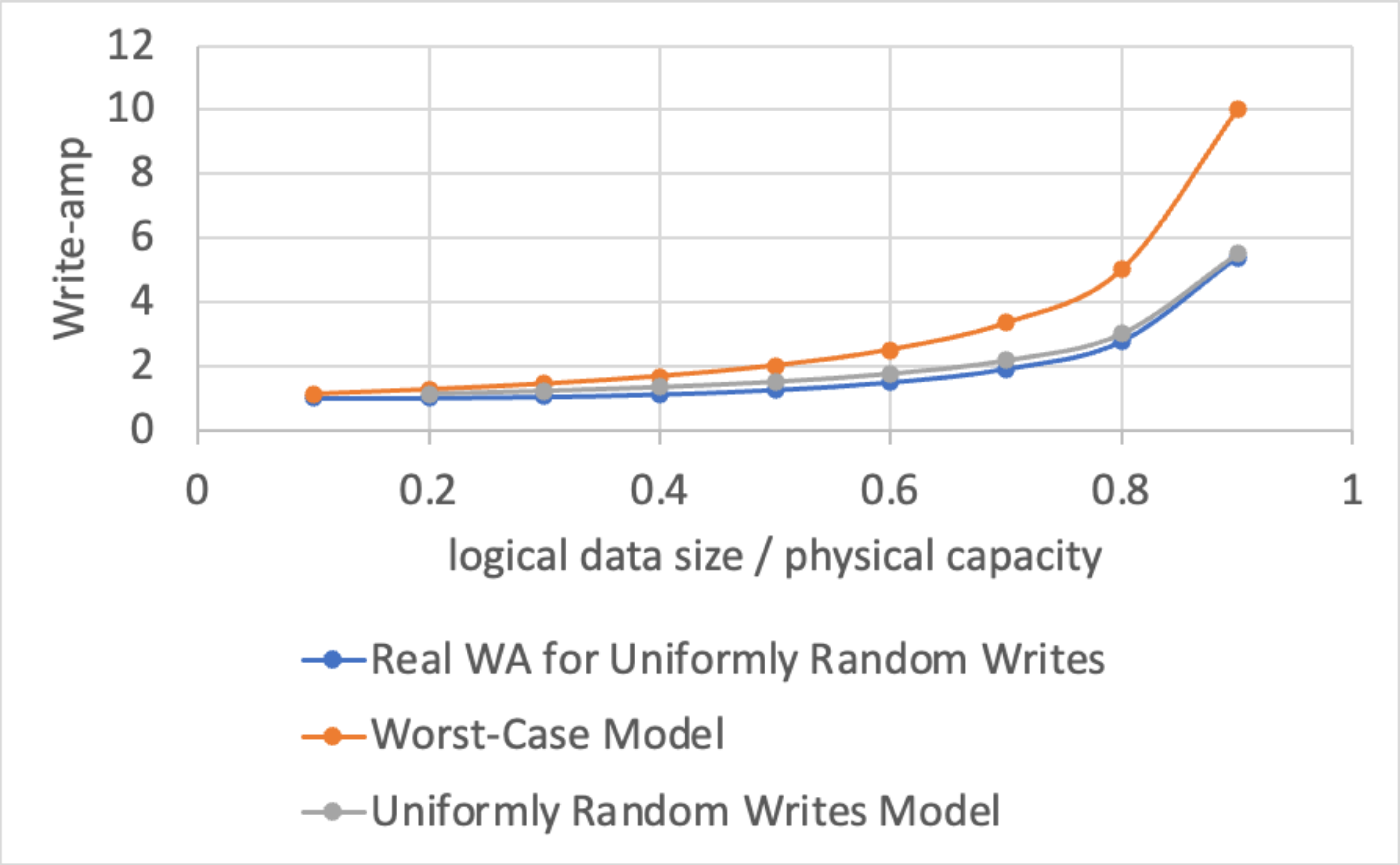
$$1 + \frac{L/P}{1 - L/P}$$

$$1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P}$$



Older

Modeling SSD Write-Amp

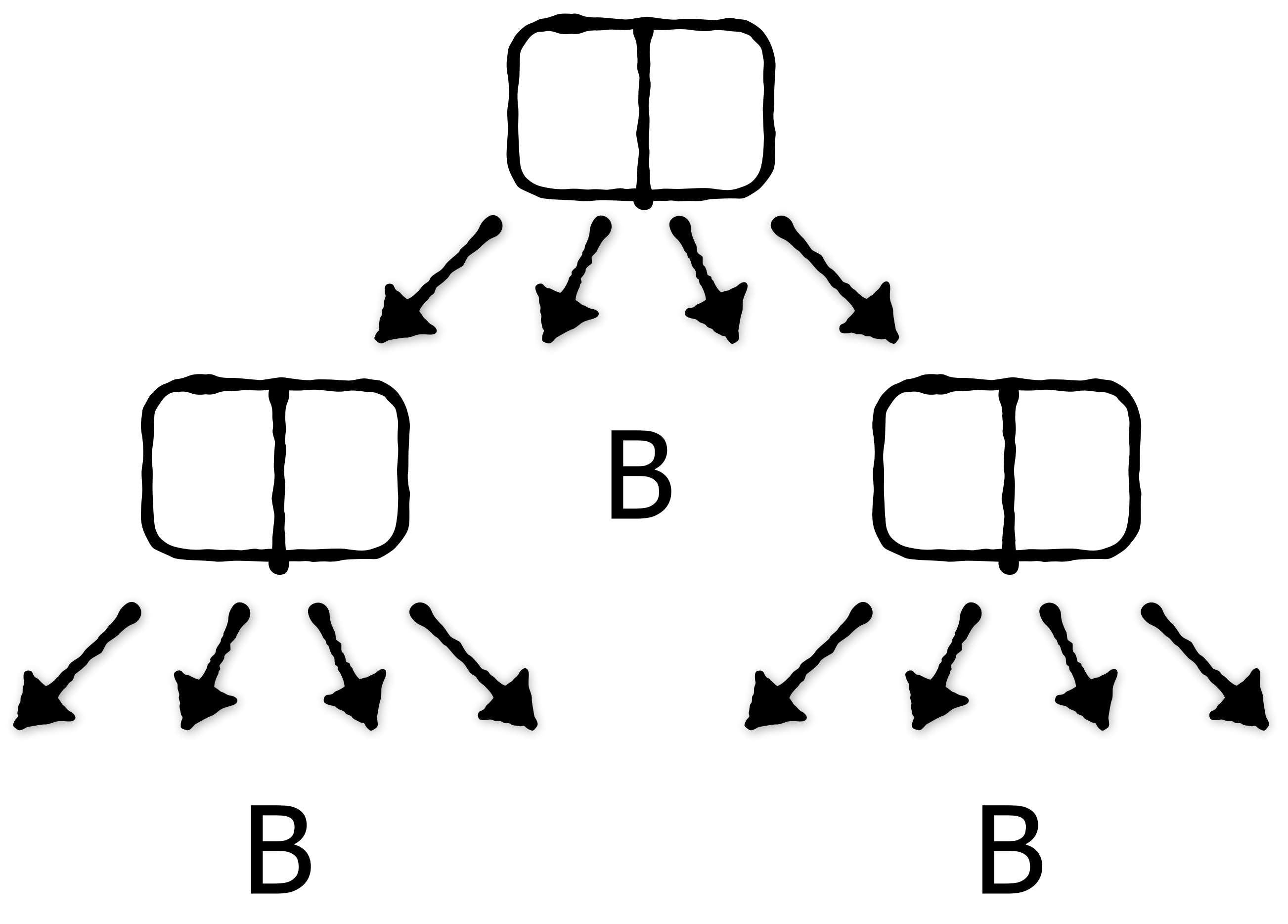


$$1 + \frac{L/P}{1 - L/P}$$

$$1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P}$$

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 physical SSD capacity, while the rest is over-provisioned. What write-amplification would you expect? A back-of-the-envelope calculation is enough.



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 physical SSD capacity, while the rest is over-provisioned. What write-amplification would you expect? A back-of-the-envelope calculation is enough.

SSD WA Model with uniformly random updates: $$1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P}$$

B-tree update cost: ≈ 1 write I/O

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 physical SSD capacity, while the rest is over-provisioned. What write-amplification would you expect? A back-of-the-envelope calculation is enough.

SSD WA Model with uniformly random updates: $$1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P}$$

B-tree update cost: ≈ 1 write I/O

B-tree WA: ≈ B

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 physical SSD capacity, while the rest is over-provisioned. What write-amplification would you expect? A back-of-the-envelope calculation is enough.

SSD WA Model with uniformly random updates: $$1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P}$$

B-tree update cost: ≈ 1 write I/O

B-tree WA: ≈ B

Total write-amp: $100 \cdot (1 + \frac{1}{2} \cdot \frac{0.7}{1 - 0.7})$ = **216.67**

Question 1

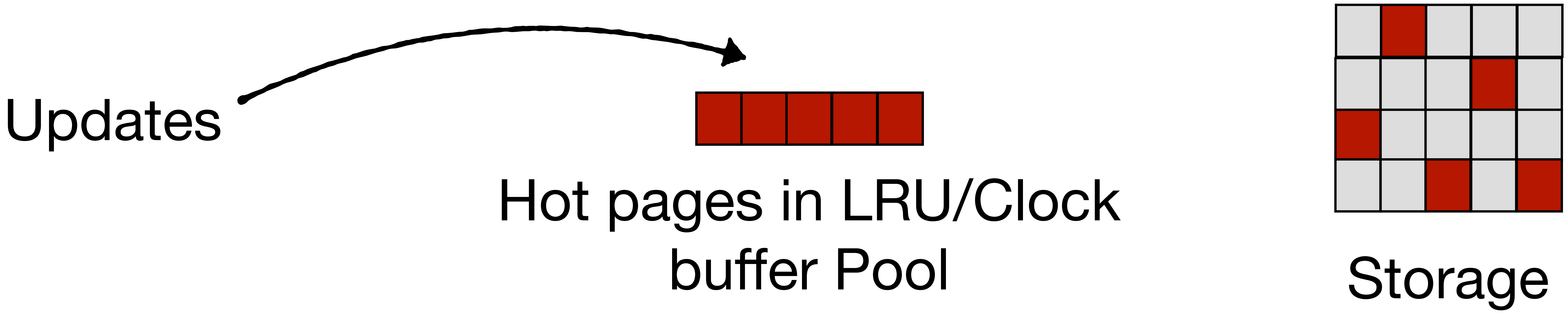
Consider a B-tree subject to uniformly randomly distributed updates. There are 100 entries per page. The b-tree occupies 70% of the physical SSD capacity, while the rest is over-provisioned. What write-amplification would you expect? A back-of-the-envelope calculation is enough.

Now suppose the workload exhibits skew (some entries are more likely to be updated). Which mechanism of a database allows us to reduce write-amplification in this case, and why?

Question 1

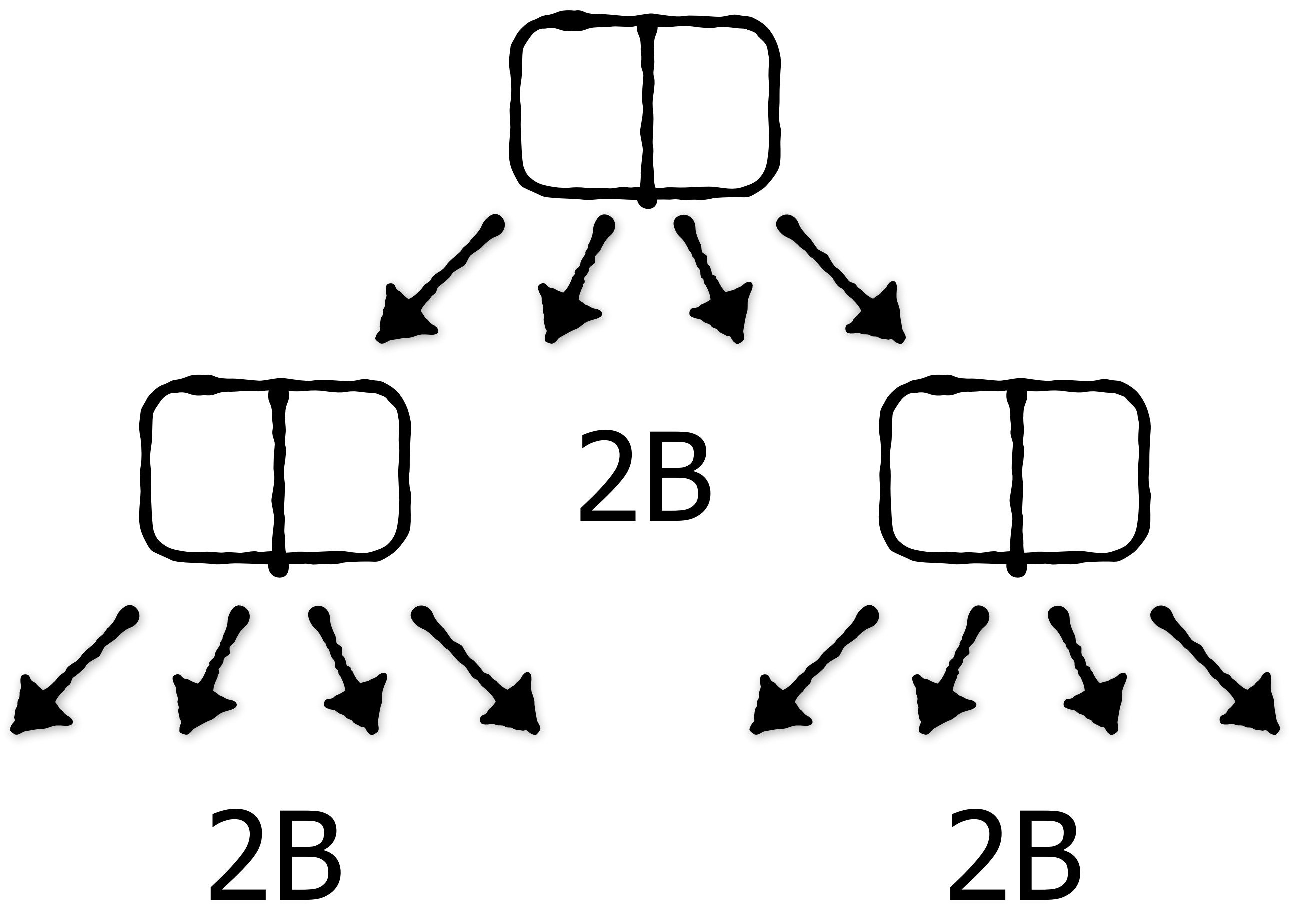
Consider a B-tree subject to uniformly randomly distributed updates. There are 100 entries per page. The b-tree occupies 70% of the physical SSD capacity, while the rest is over-provisioned. What write-amplification would you expect? A back-of-the-envelope calculation is enough.

Now suppose the workload exhibits skew (some entries are more likely to be updated). Which mechanism of a database allows us to reduce write-amplification in this case, and why?



Question 2

We have a B-tree on an SSD. the workload is read-intensive, so we only care about read performance. Each node and each SSD page are 4 KB. Consider making each B-tree node take up two rather than one flash pages (i.e., 8KB rather than 4KB). This can make the tree shallower. Is this a good idea?

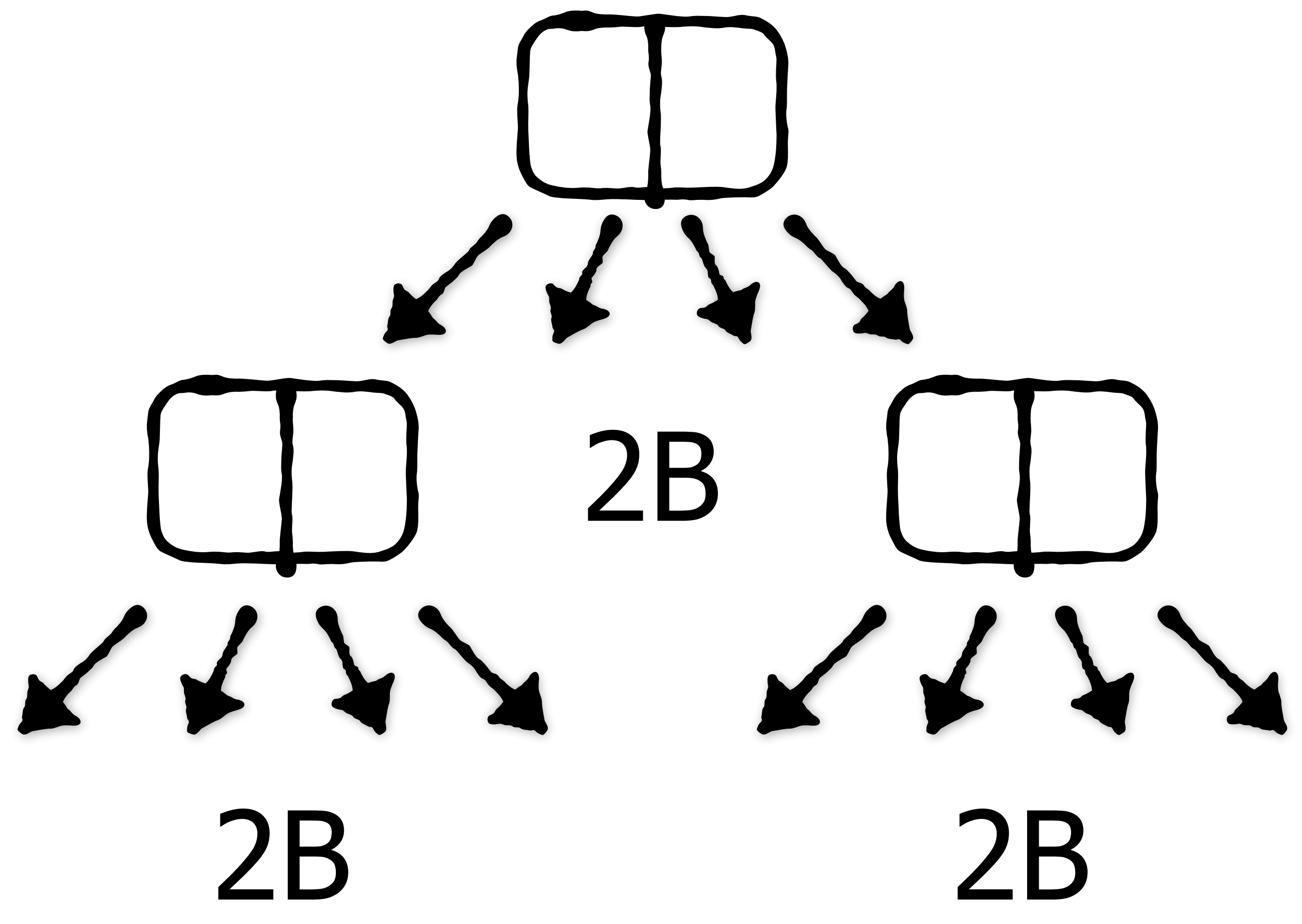


Question 2

We have a B-tree on an SSD. the workload is read-intensive, so we only care about read performance. Each node and each SSD page are 4 KB. Consider making each B-tree node take up two rather than one flash pages (i.e., 8KB rather than 4KB). This can make the tree shallower. Is this a good idea?

On SSD, cost is measured as # pages accessed

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



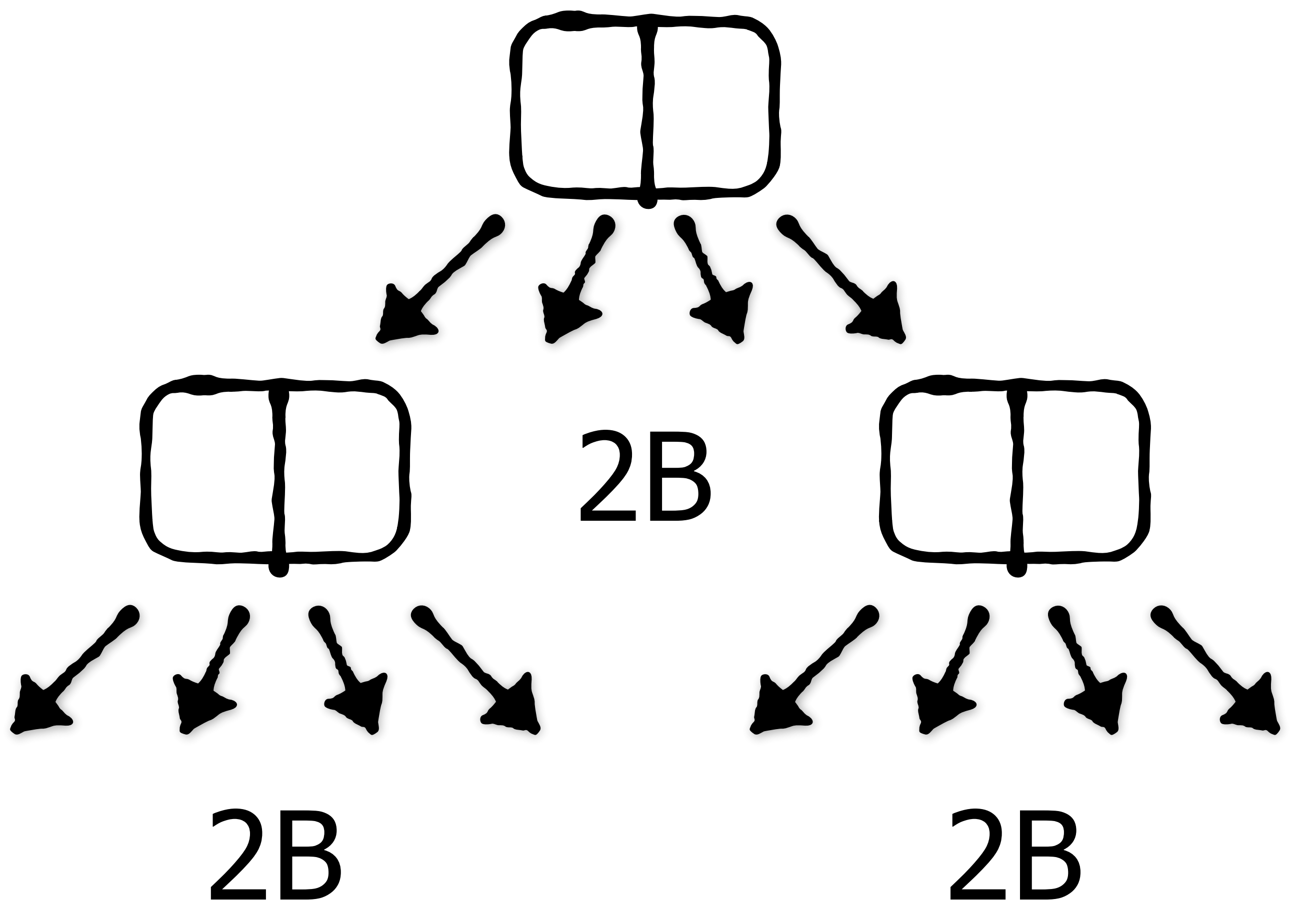
Question 2

We have a B-tree on an SSD. the workload is read-intensive, so we only care about read performance. Each node and each SSD page are 4 KB. Consider making each B-tree node take up two rather than one flash pages (i.e., 8KB rather than 4KB). This can make the tree shallower. Is this a good idea?

On SSD, cost is measured as # pages accessed

$$2 \cdot \log_{2B}(N) \leq \mathbf{\log_B(N)}$$

Condition for being cheaper than standard B-tree



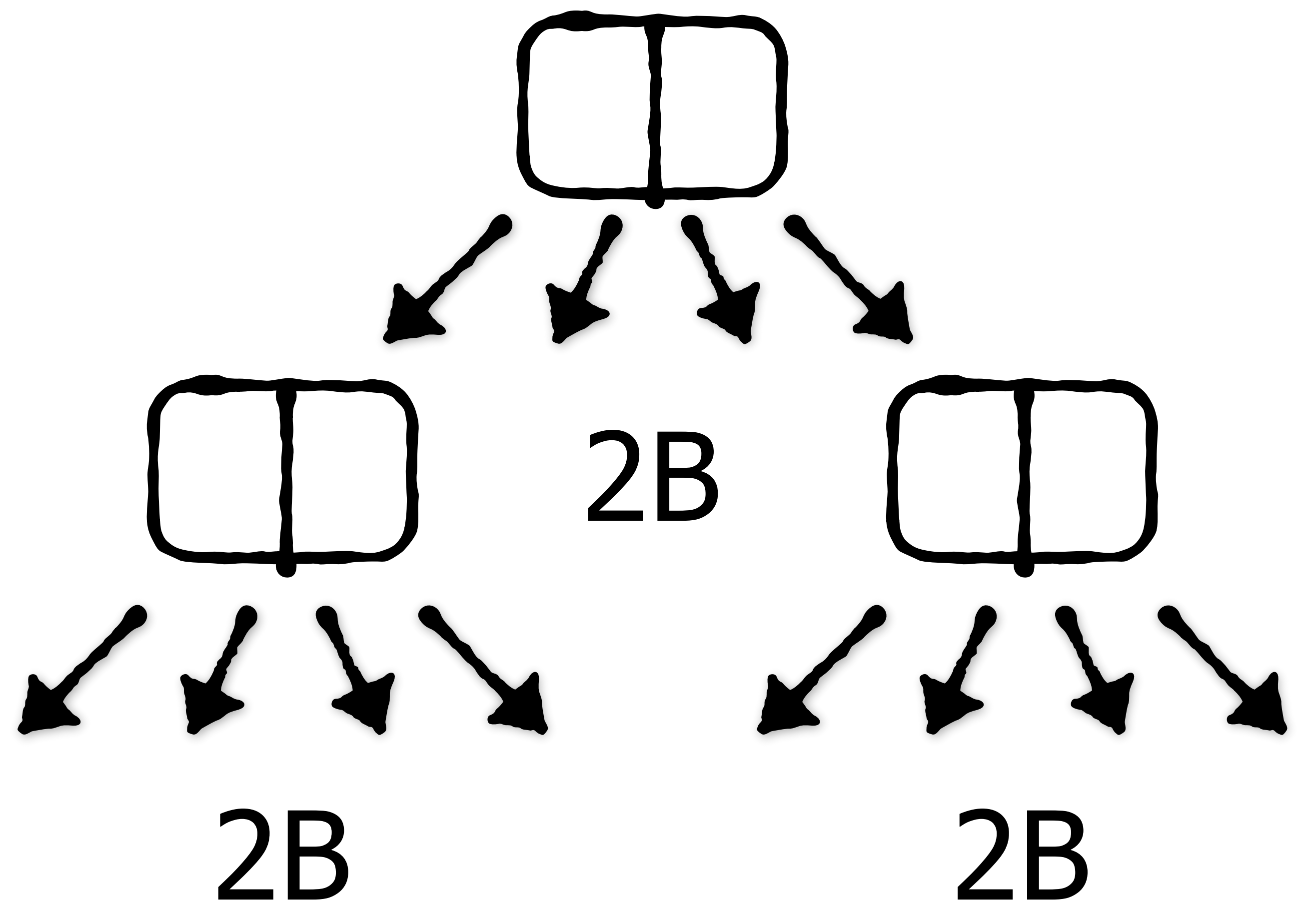
Question 2

We have a B-tree on an SSD. the workload is read-intensive, so we only care about read performance. Each node and each SSD page are 4 KB. Consider making each B-tree node take up two rather than one flash pages (i.e., 8KB rather than 4KB). This can make the tree shallower. Is this a good idea?

On SSD, cost is measured as # pages accessed

$$2 \cdot \log_{2B}(N) \quad \leq \quad \log_{B}(N)$$

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



Question 2

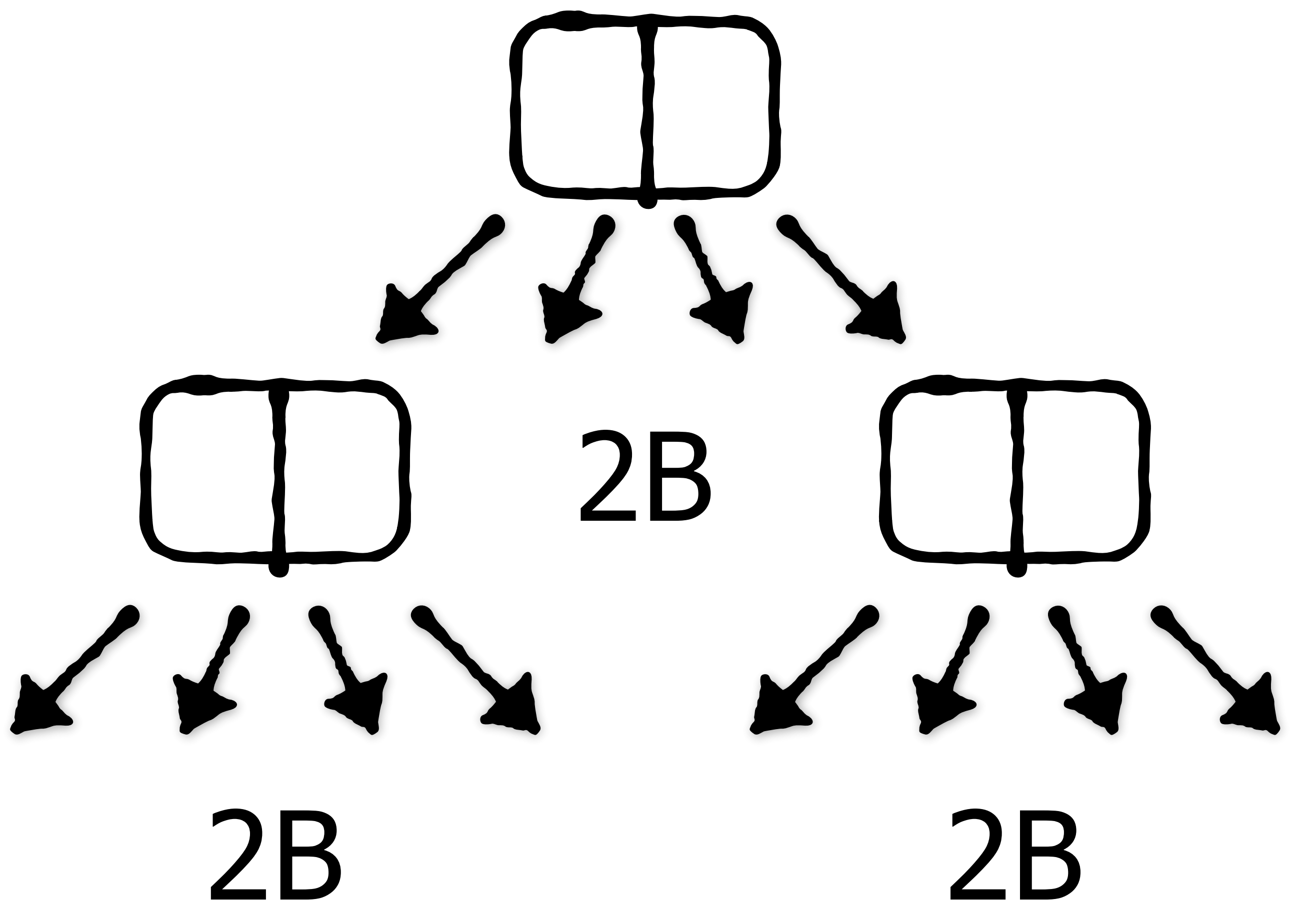
We have a B-tree on an SSD. the workload is read-intensive, so we only care about read performance. Each node and each SSD page are 4 KB. Consider making each B-tree node take up two rather than one flash pages (i.e., 8KB rather than 4KB). This can make the tree shallower. Is this a good idea?

On SSD, cost is measured as # pages accessed

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

Simplifies to: $B \leq 2$

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



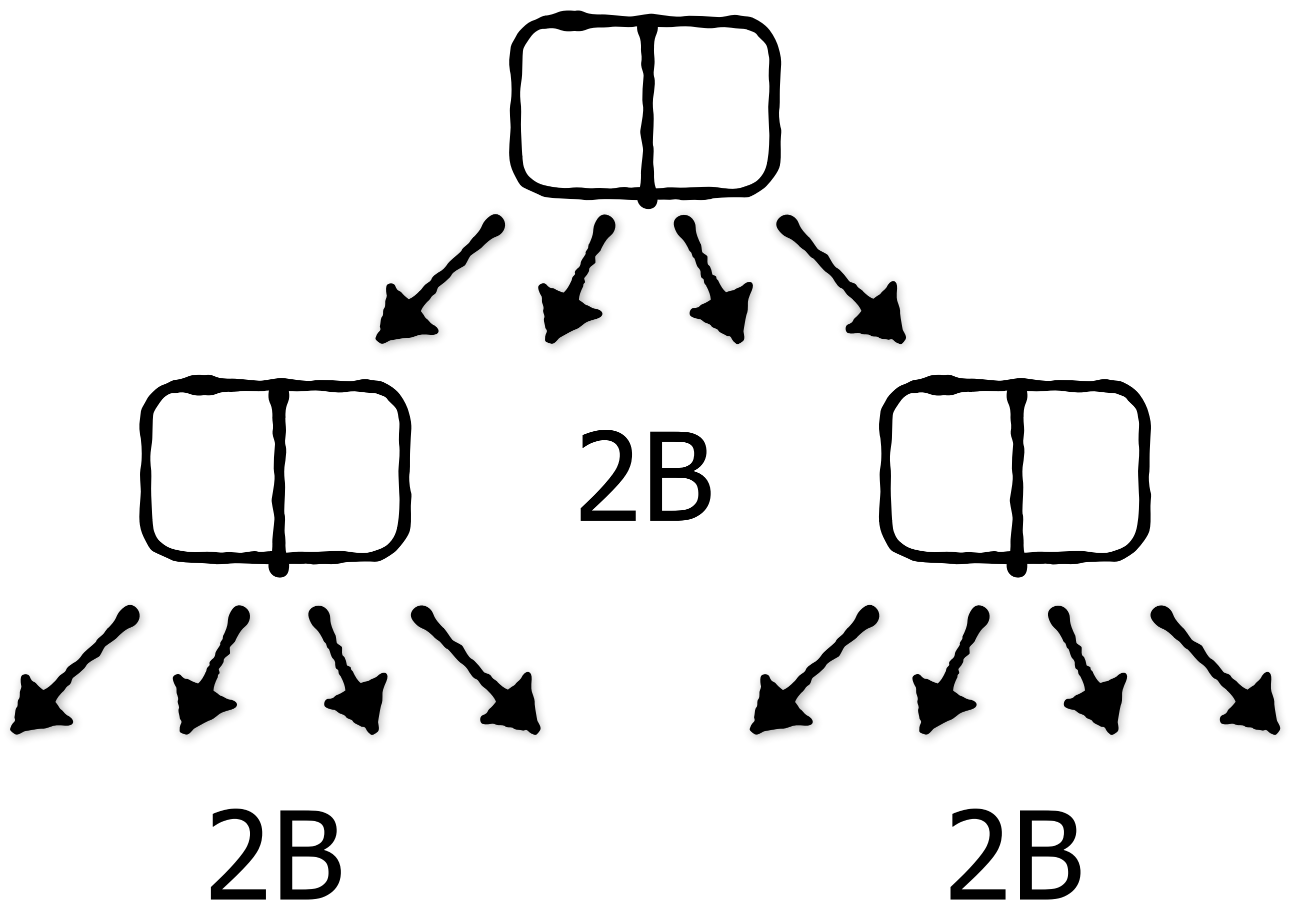
Question 2

We have a B-tree on an SSD. the workload is read-intensive, so we only care about read performance. Each node and each SSD page are 4 KB. Consider making each B-tree node take up two rather than one flash pages (i.e., 8KB rather than 4KB). This can make the tree shallower. Is this a good idea?

On SSD, cost is measured as # pages accessed

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

Simplifies to: $B \leq 2$



What if we use disk instead of SSD?

Question 2

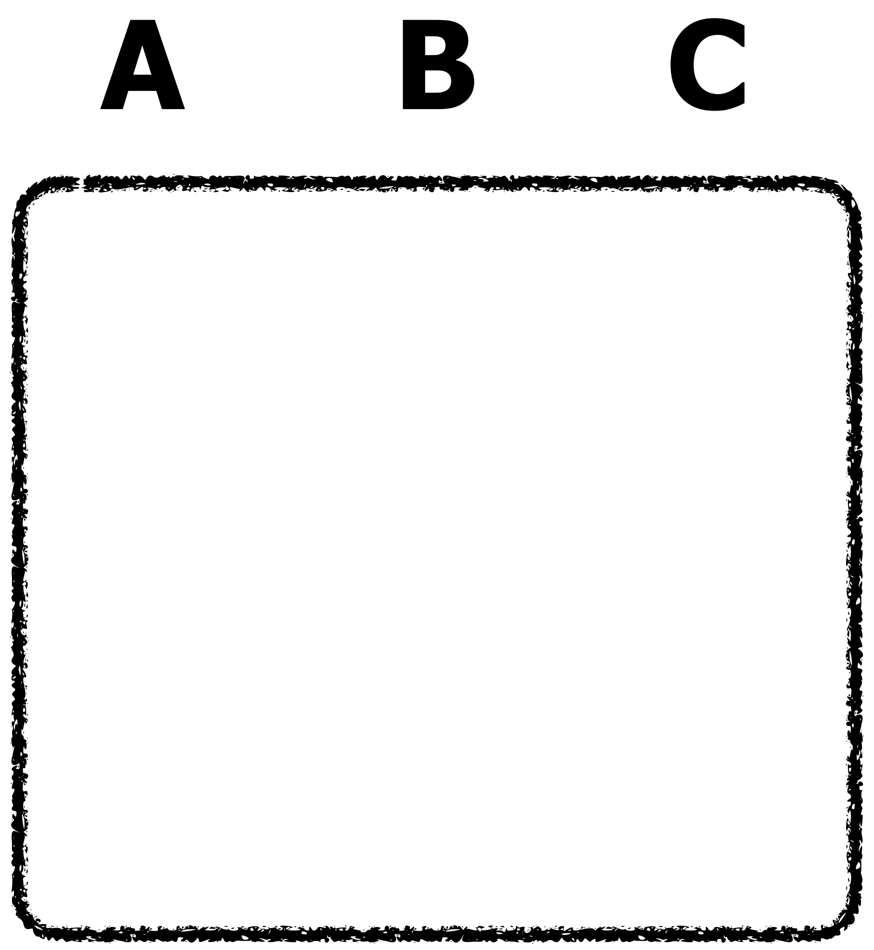
We have a B-tree on an SSD. the workload is read-intensive, so we only care about read performance. Each node and each SSD page are 4 KB. Consider making each B-tree node take up two rather than one flash pages (i.e., 8KB rather than 4KB). This can make the tree shallower. Is this a good idea?

What if we use disk instead of SSD?

On disk, seek & rotational delay dominates, while data transfer is negligible. So this is a good idea. A node size in the area of a few MB is common for disk. Beyond that, we begin to incur a penalty for long sequential accesses.

Question 3

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

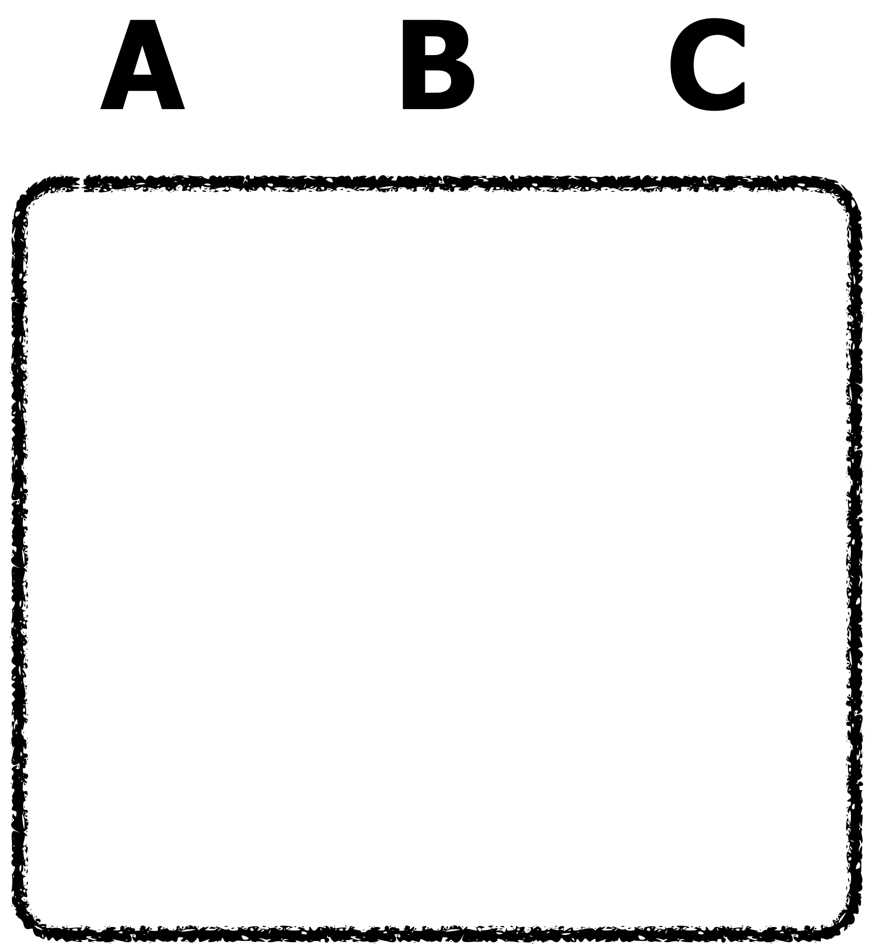


Workload:

50%	Select * from table where A = "…"	Returns one row
50%	Insert (, ,)	

Question 3

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



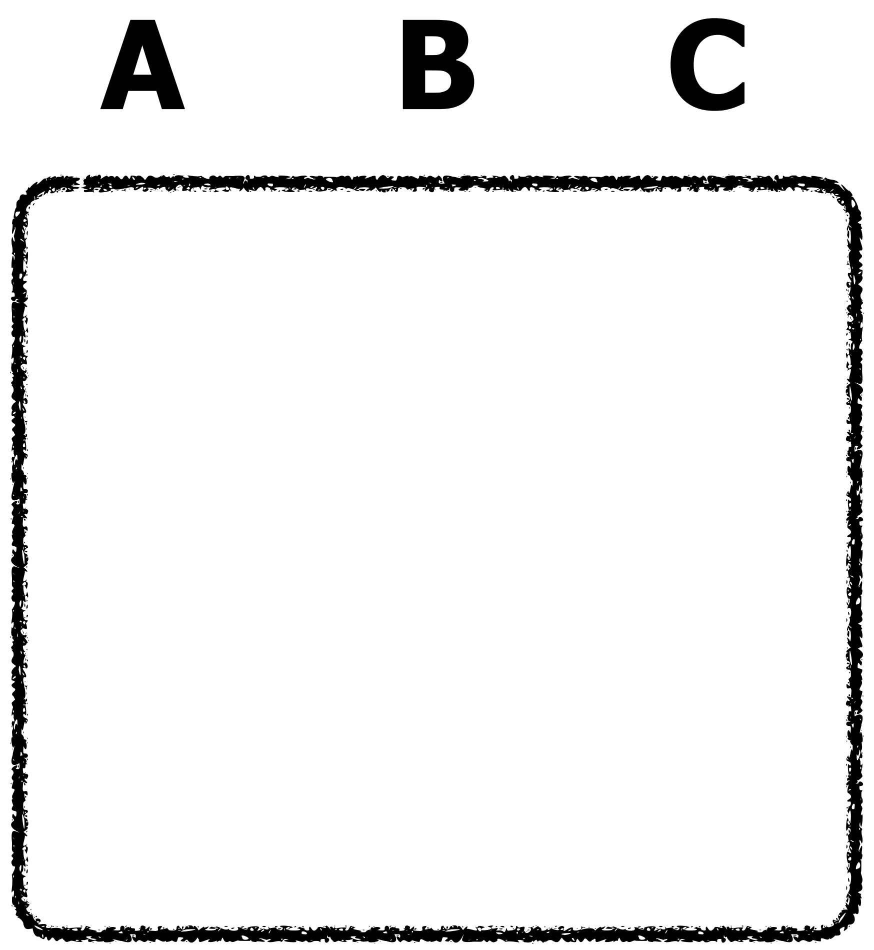
Workload:

50%	Select * from table where A = "…"	Returns one row
50%	Insert (, ,)	

Should we employ a B-tree index on any of the columns?
Estimate the overall I/O cost of both operations with & without it.
(Back-of-the-envelope reasoning is sufficient)

Question 3

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



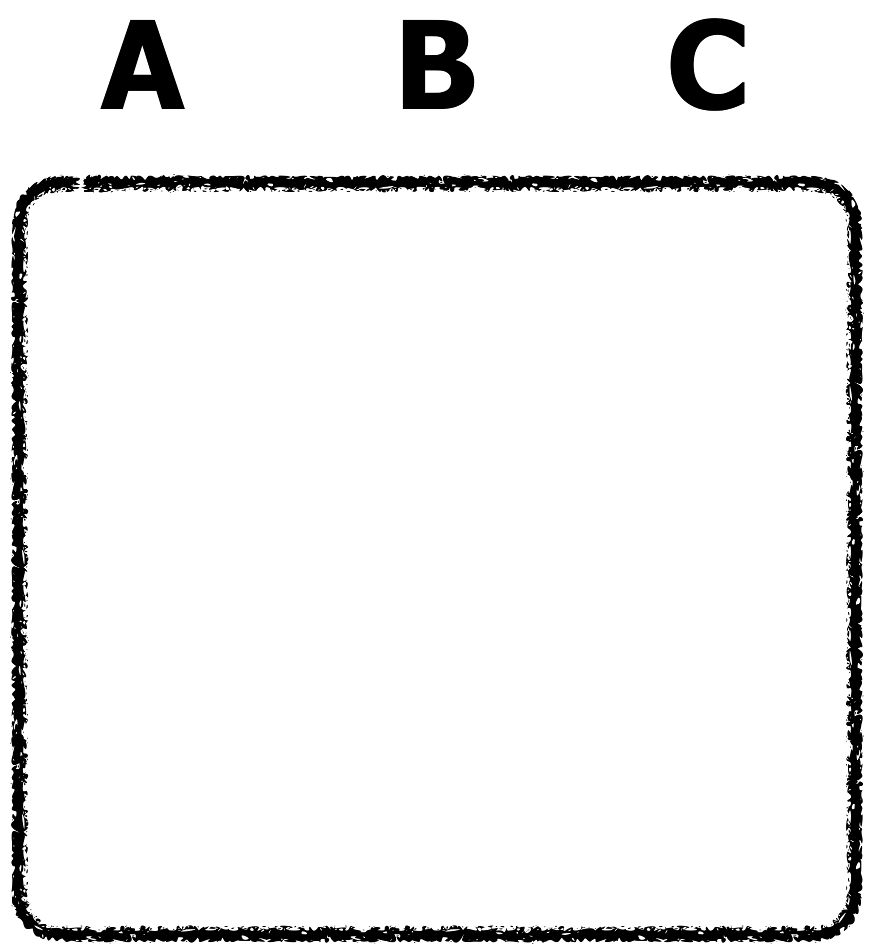
50%	Select * from table where A = "…"	Returns one row
50%	Insert (, ,)	

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

	Query		**Insert**		
I/O cost without index	$0.5 \cdot N/B$	$+$	$0.5 \cdot 1/B$	$\approx$	$0.5 \cdot N/B$

Question 3

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



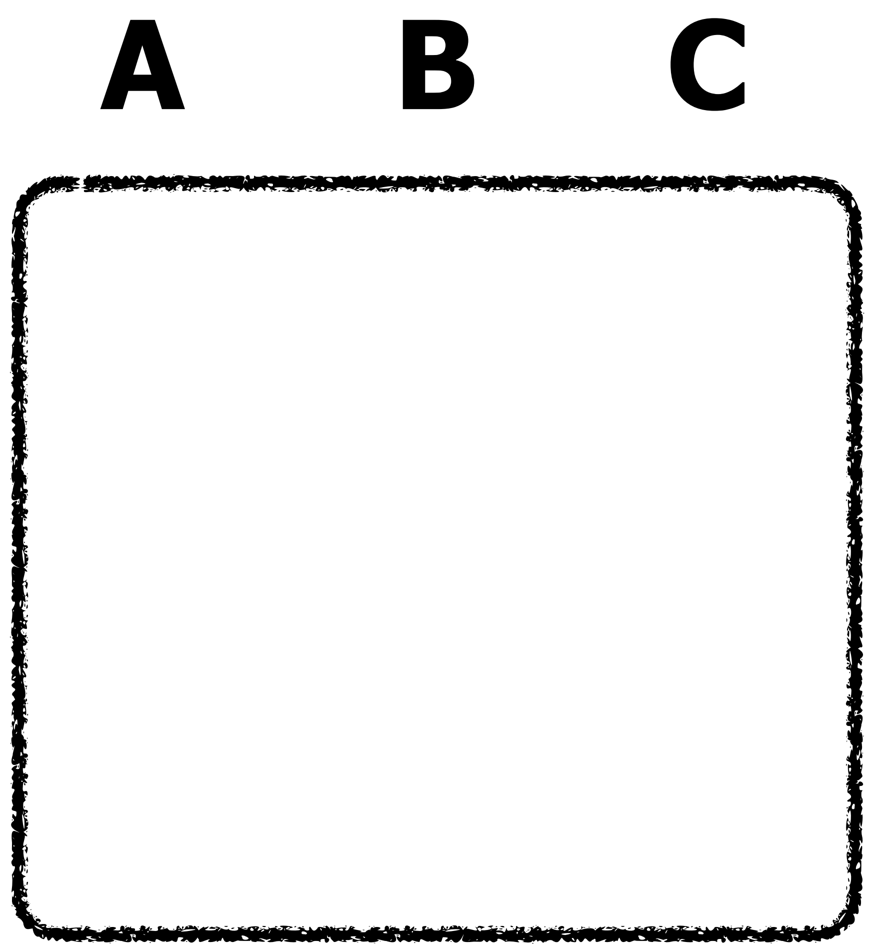
50%	Select * from table where A = "…" Returns one row
50%	Insert (, ,)

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

	Query		**Insert**		
I/O cost without index	$0.5 \cdot N/B$	+	$0.5 \cdot 1/B$	≈	$0.5 \cdot N/B$
I/O cost with index on A	$0.5 \cdot \log_B N$	+	$0.5 \cdot (1/B + \log_B N)$	≈	$\log_B N$

Question 3

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



50%	Select * from table where A = "…" Returns one row
50%	Insert (, ,)

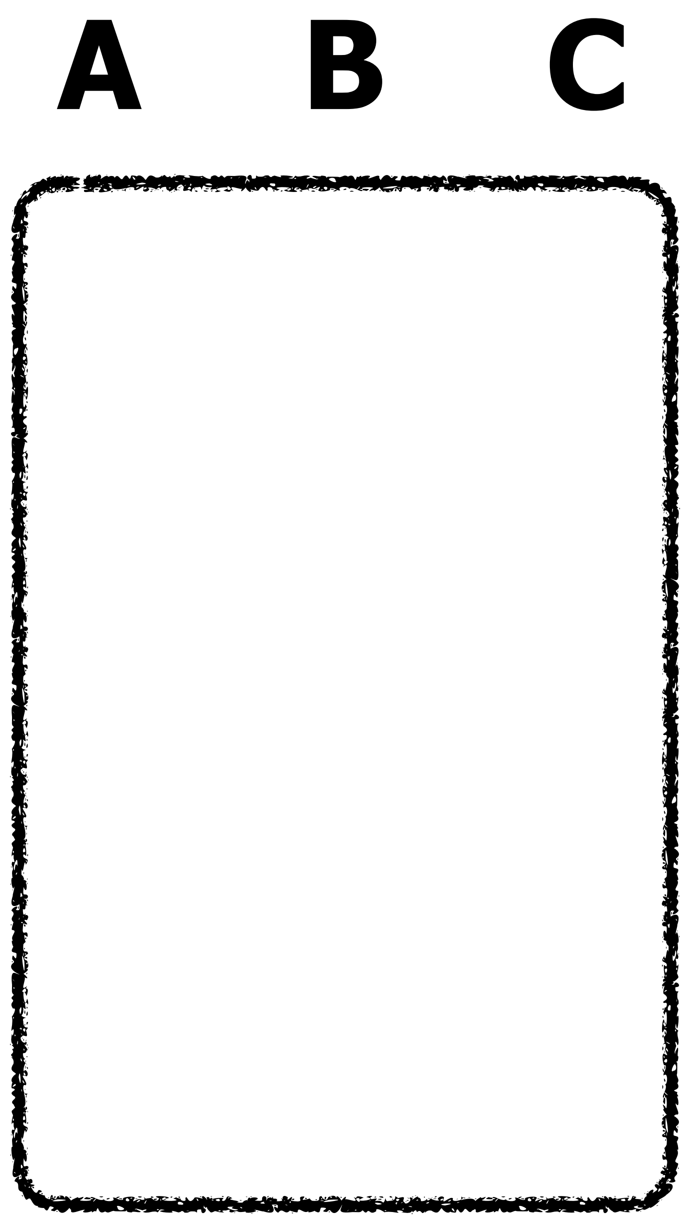
Should we employ a B-tree index on any of the columns?
Estimate the overall I/O cost of both operations with & without it.

	Query		Insert		
I/O cost without index	$0.5 \cdot N/B$	+	$0.5 \cdot 1/B$	$\approx$	$0.5 \cdot N/B$
I/O cost with index on A	$0.5 \cdot \log_B N$	+	$0.5 \cdot (1/B + \log_B N)$	$\approx$	$\log_B N$

Indexing column A significantly reduces overall costs.

Question 4

A table has columns A, B and C. The workload consists of two query types:



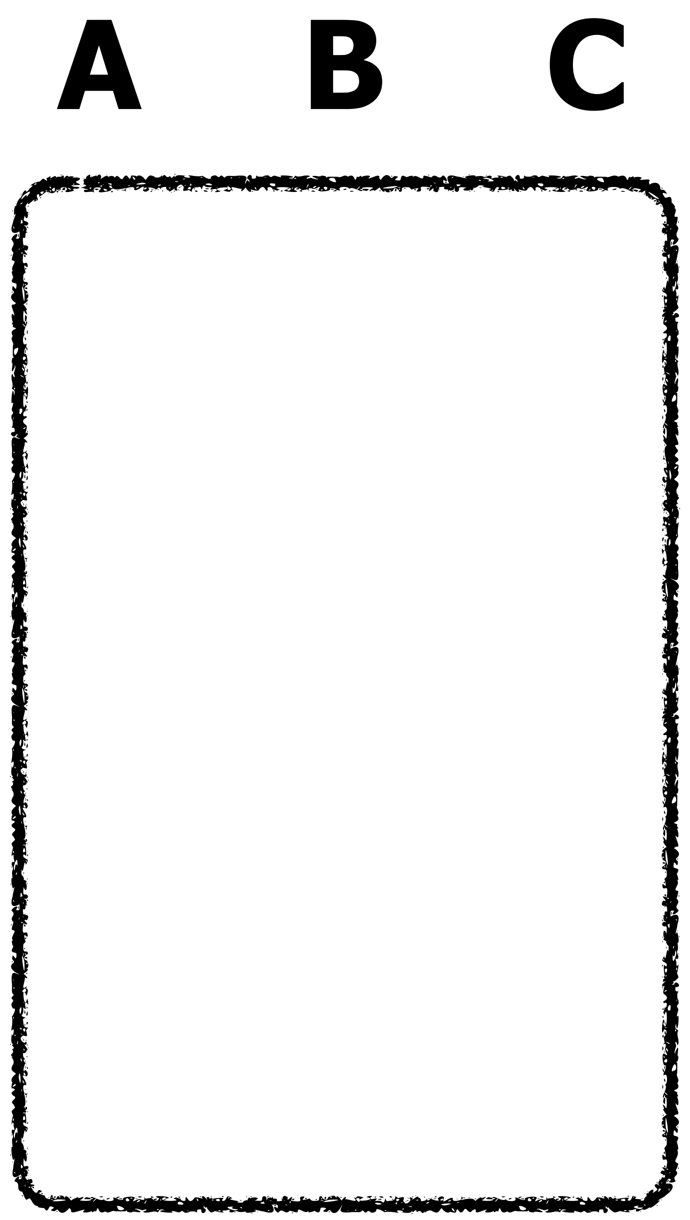
50% Select A from table where A = “…” Returns one row

50% Select * from table where B > x and B < y Returns avg. 10 rows

How should we index this table? B-tree or hash table? 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

A table has columns A, B and C. The workload consists of two query types:



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

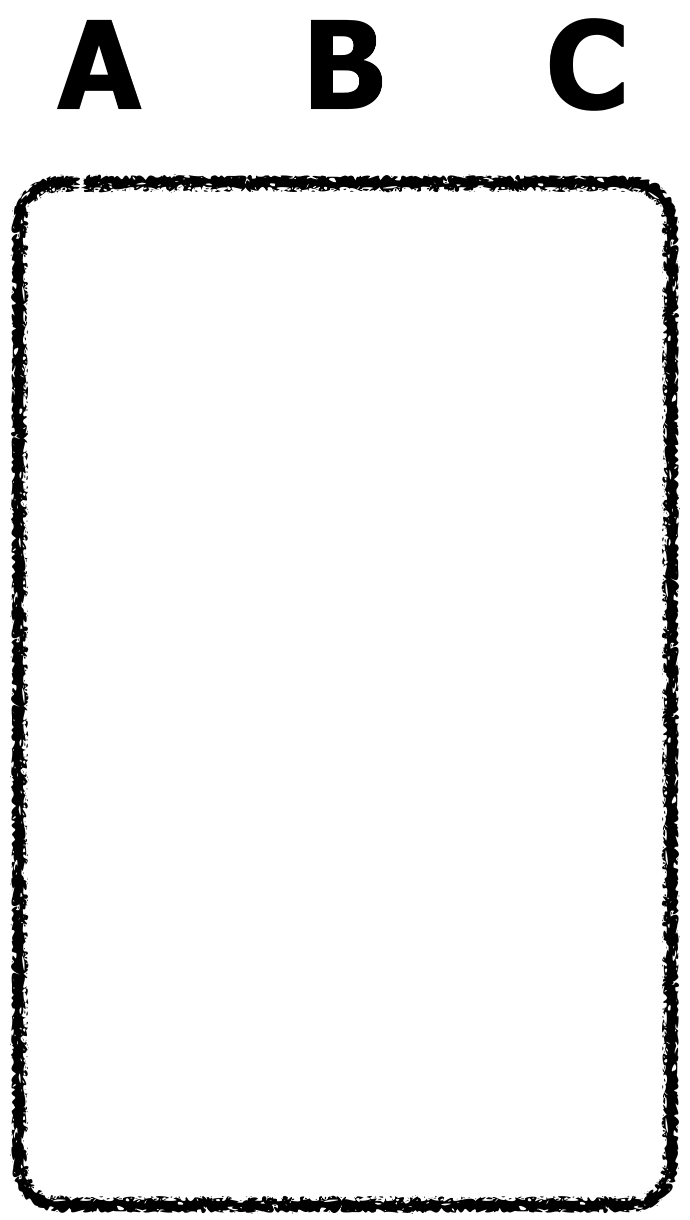
50% Select * from table where B > x and B < y Returns avg. 10 rows

How should we index this table? B-tree or hash table? 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 **Unclustered Hash table on A**

Question 4

A table has columns A, B and C. The workload consists of two query types:



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

How should we index this table? B-tree or hash table? 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

$$\lceil \log_B(N) \rceil + \lceil S/B \rceil$$

$$4 + 1$$

Unclustered Hash table on A

$\approx$1 I/O per query