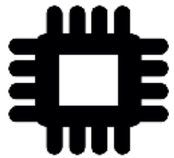
Chucky: A Succinct Cuckoo Filter for LSM-Tree

Niv Dayan, Moshe Twitto

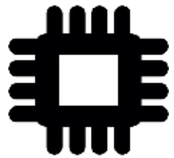
PLiOPS

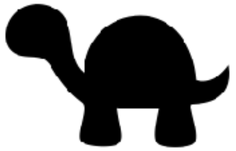
EXTREME DATA PROCESSOR

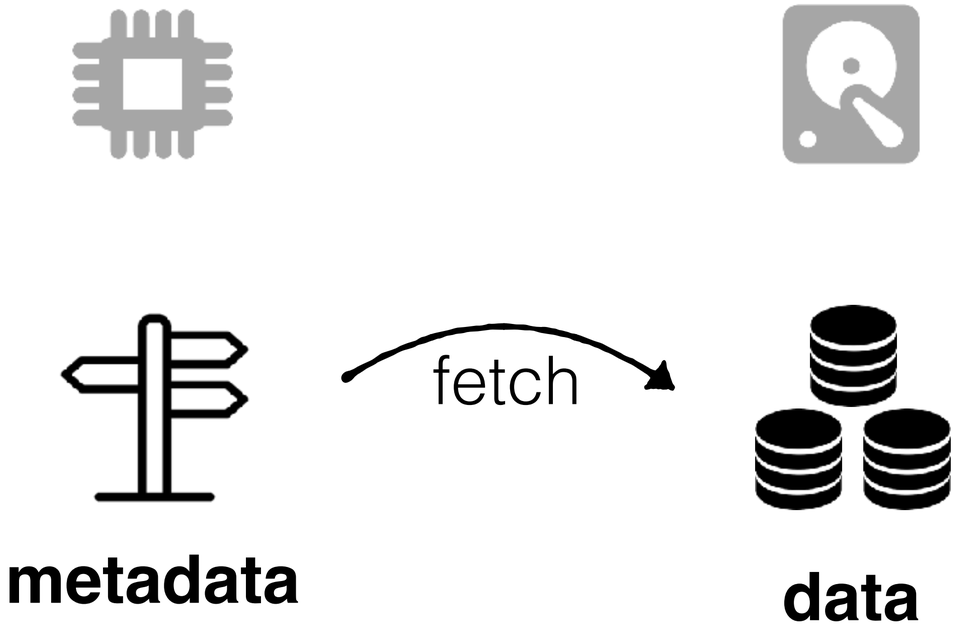


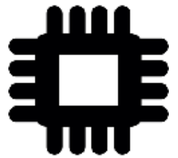
memory

storage

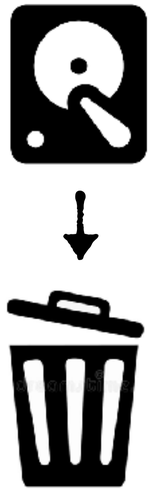





fetch
metadata
data

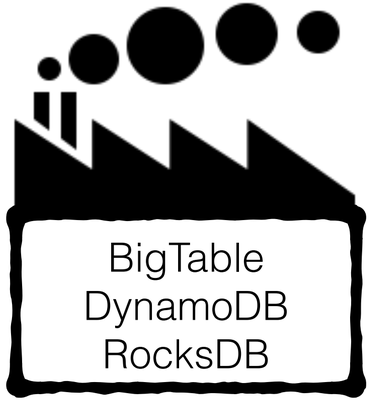



SSD



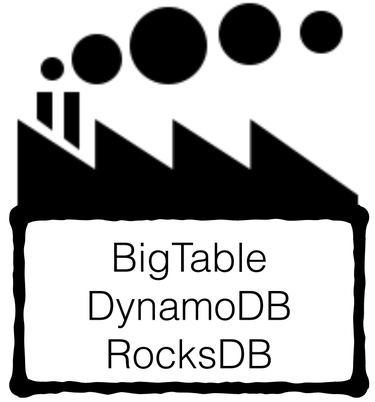
noticeable overhead

LSM-tree



Bloom filters

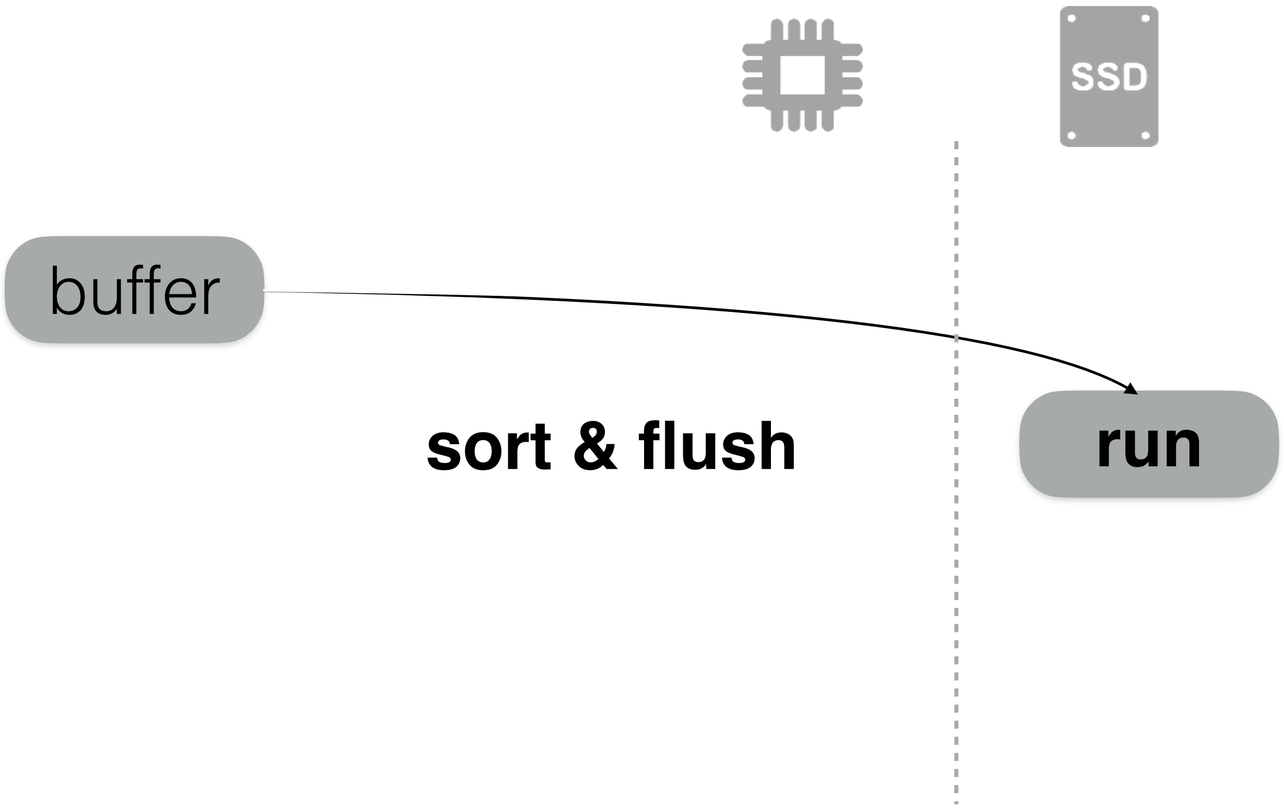
LSM-tree

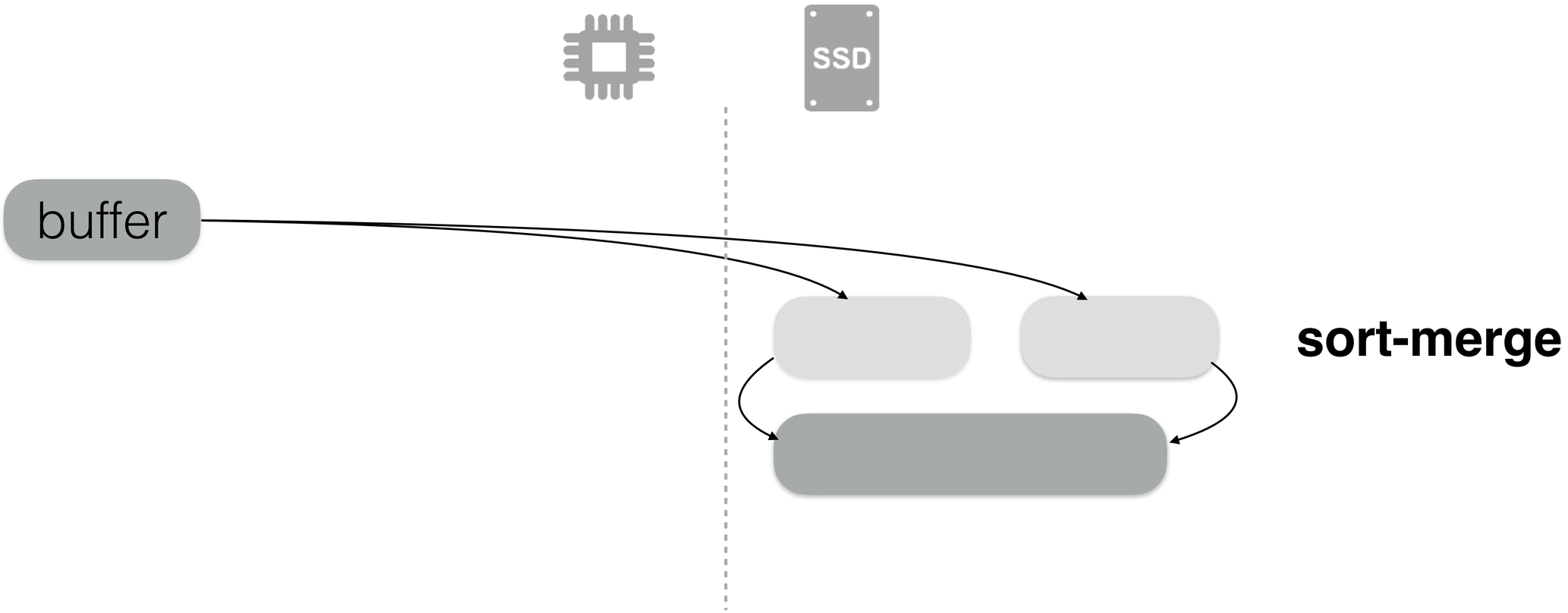


LSM-tree

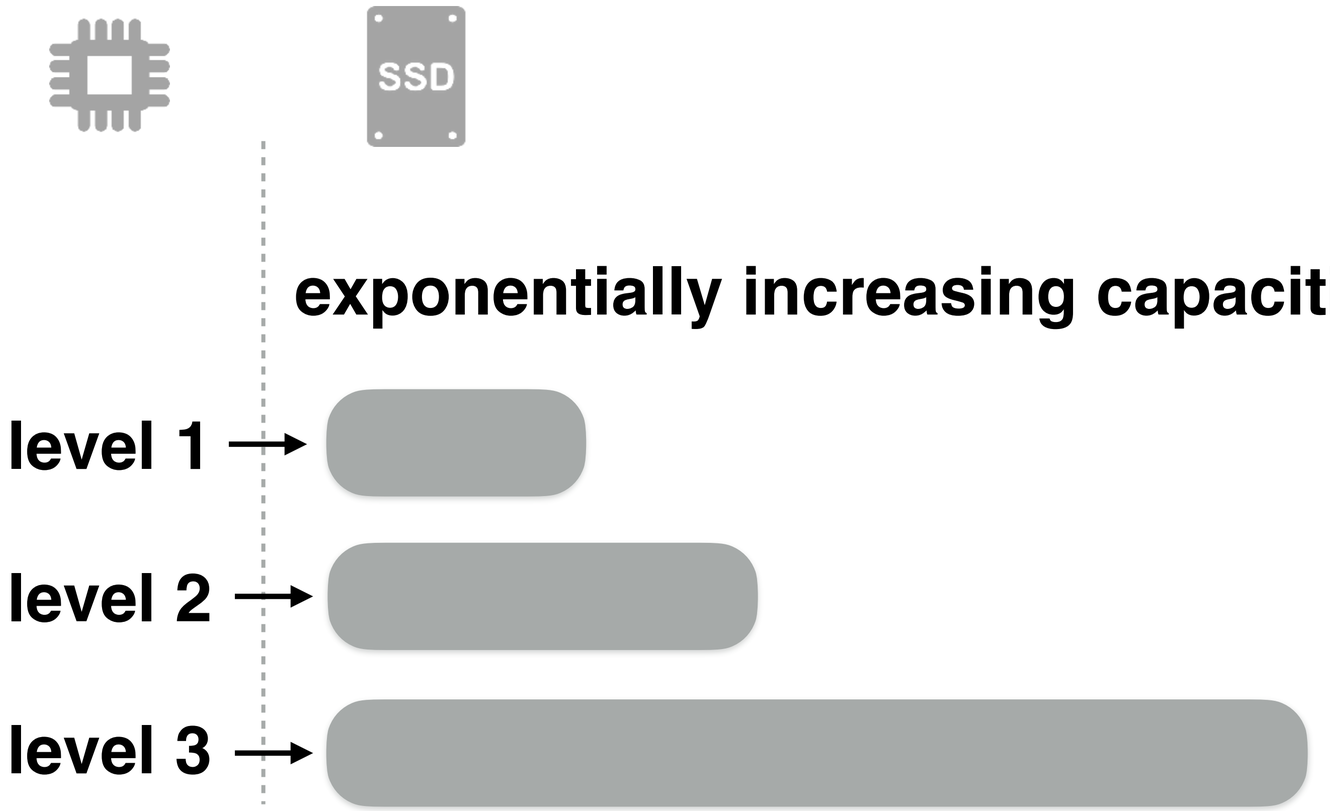
key-value pairs

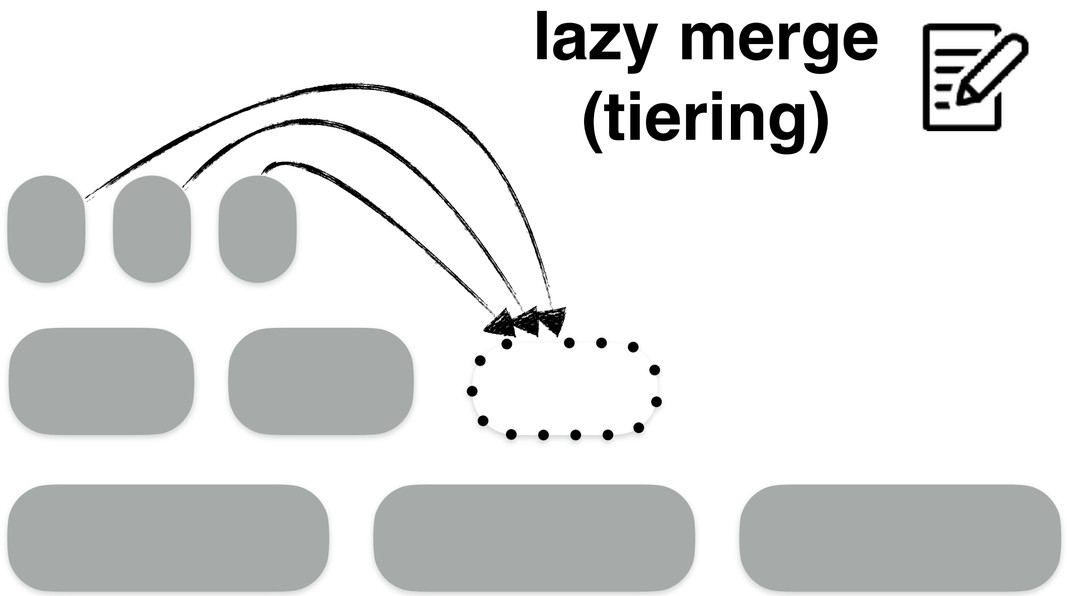
buffer


SSD
buffer
sort & flush
run

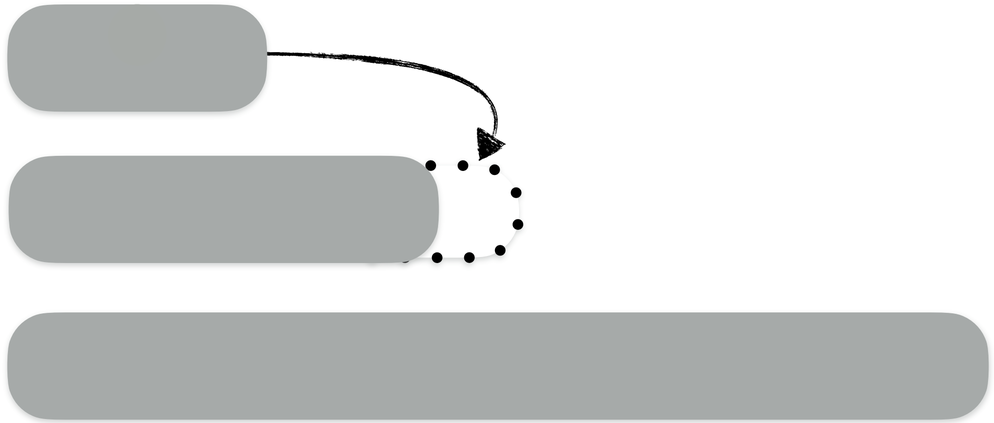

SSD
buffer
sort-merge

buffer

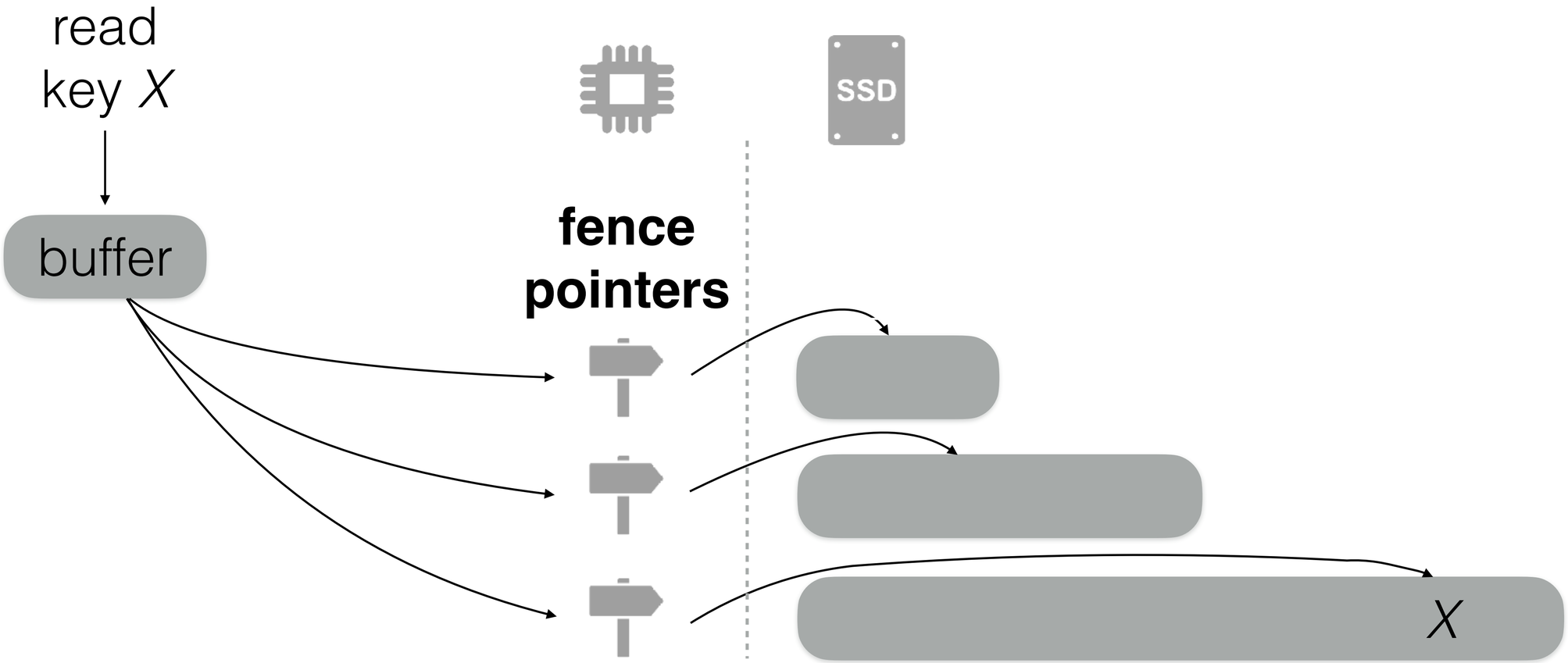

SSD
exponentially increasing capacities
level 1
level 2
level 3

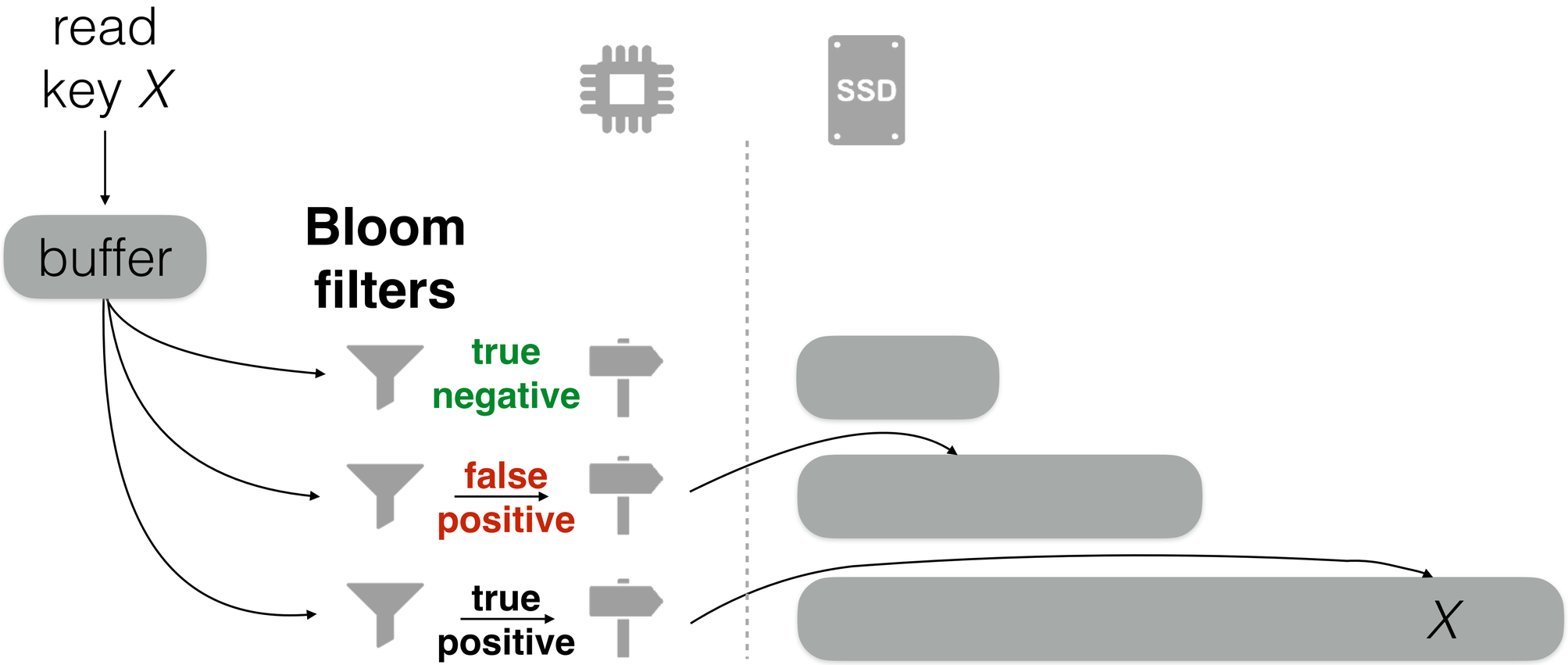

lazy merge
(tiering)

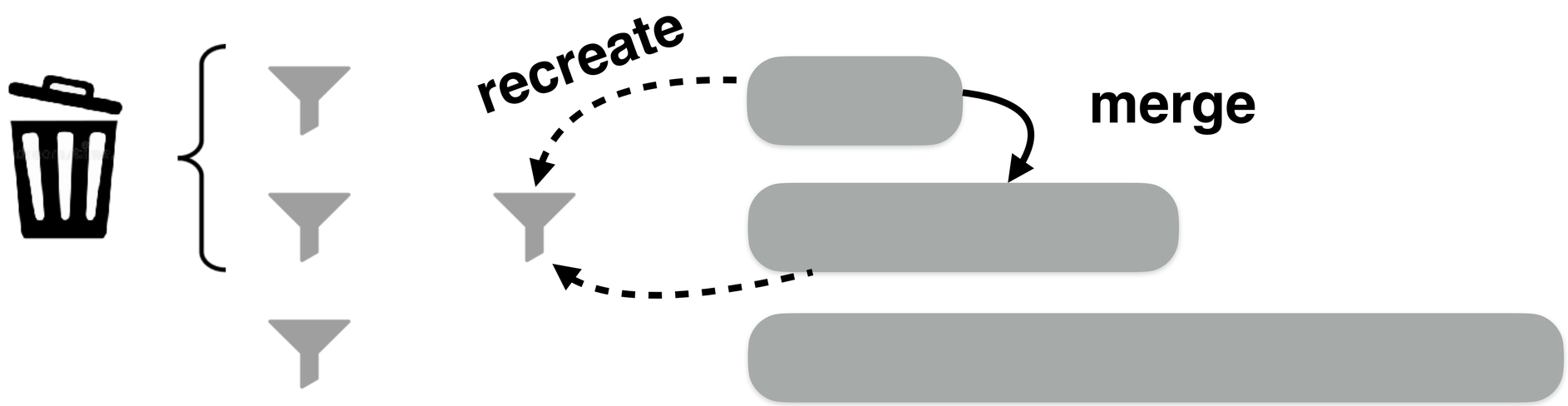
greedy merge (leveling)



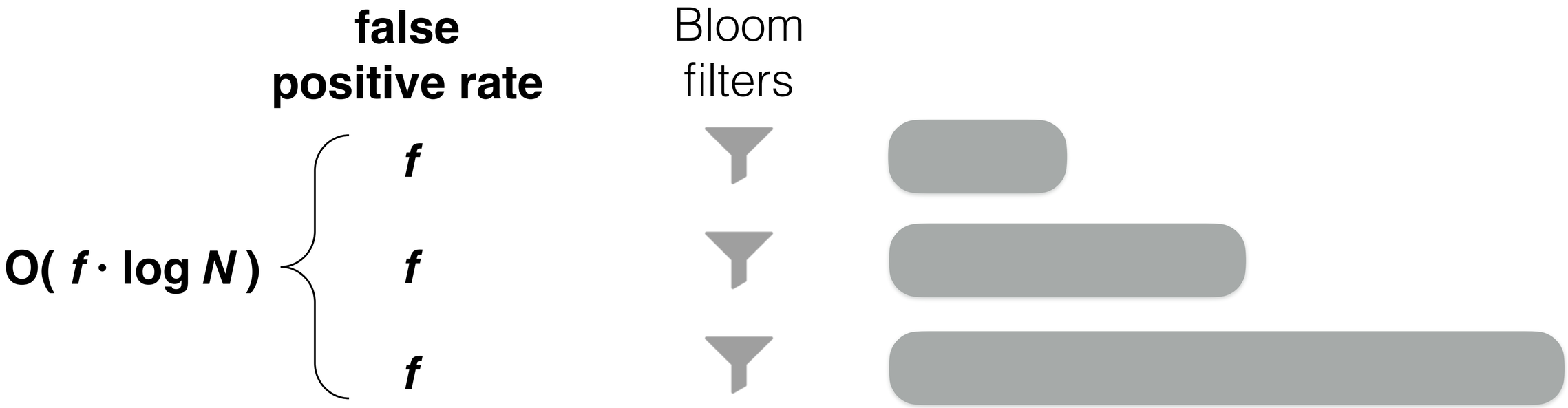
Levels $L = \log(M)$

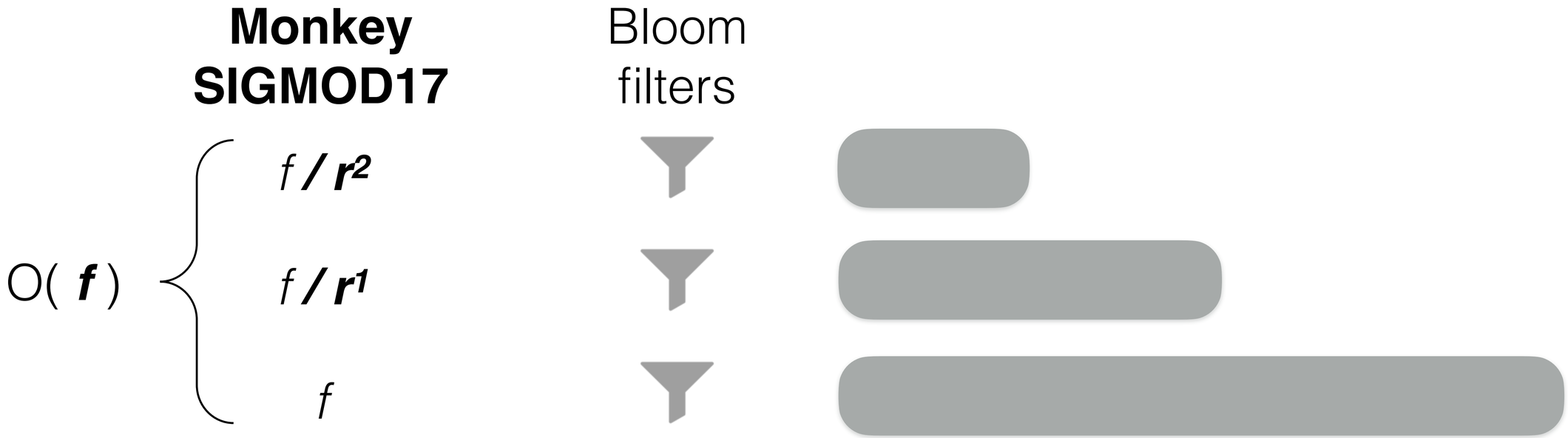

read
key X
buffer
SSD
fence
pointers
X


read
key X
buffer
Bloom
filters
SSD
true
negative
false
positive
true
positive
X


recreate
merge

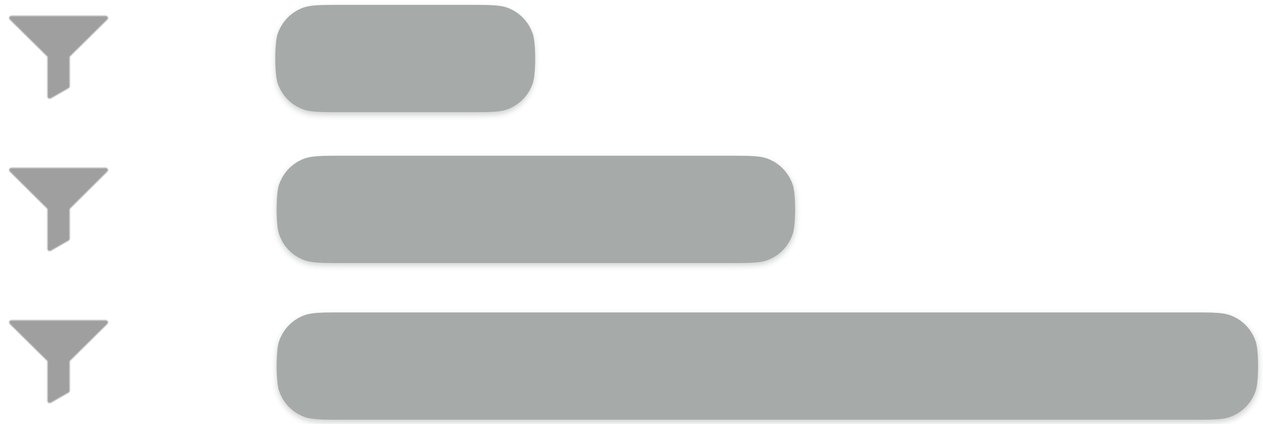
Bloom Filters' WA = LSM-tree's WA


false
positive rate
Bloom
filters
f
f
f
O(f · log N)

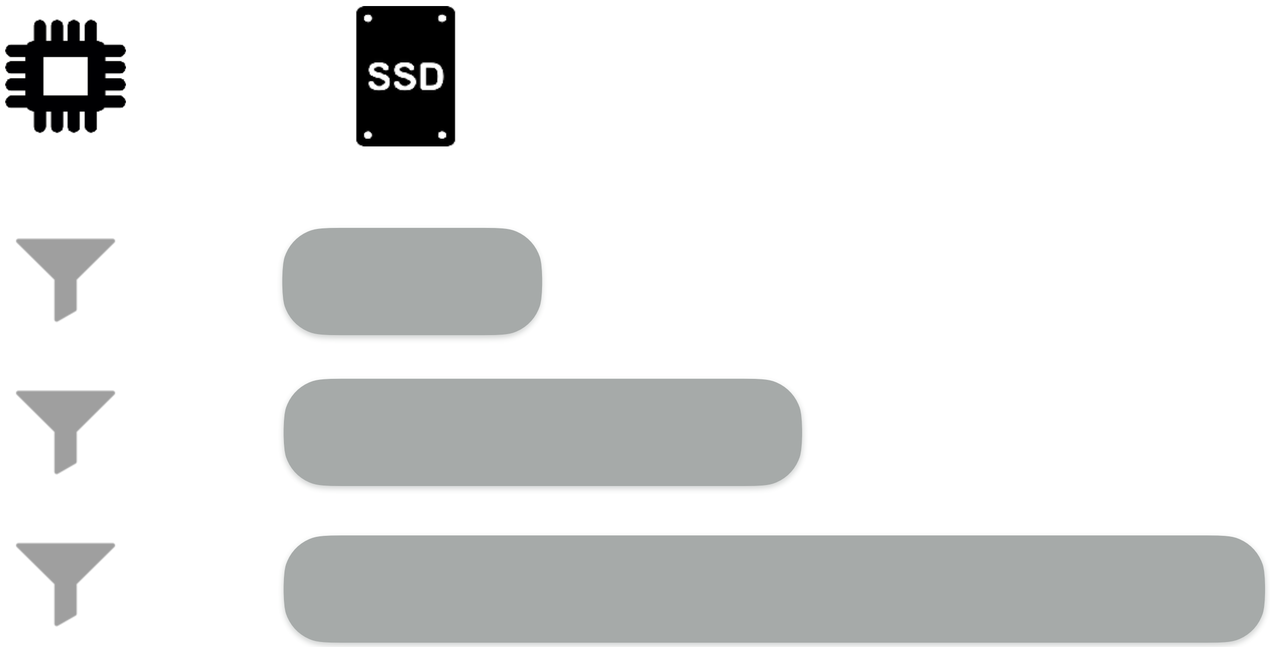

Monkey
SIGMOD17
Bloom
filters
O(f)
f / r²
f / r¹
f

$O(e^{-M \cdot \ln(2)})$

Bloom filters

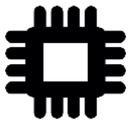


Problem: shrinking memory/storage speed gap



Problem: shrinking memory/storage speed gap

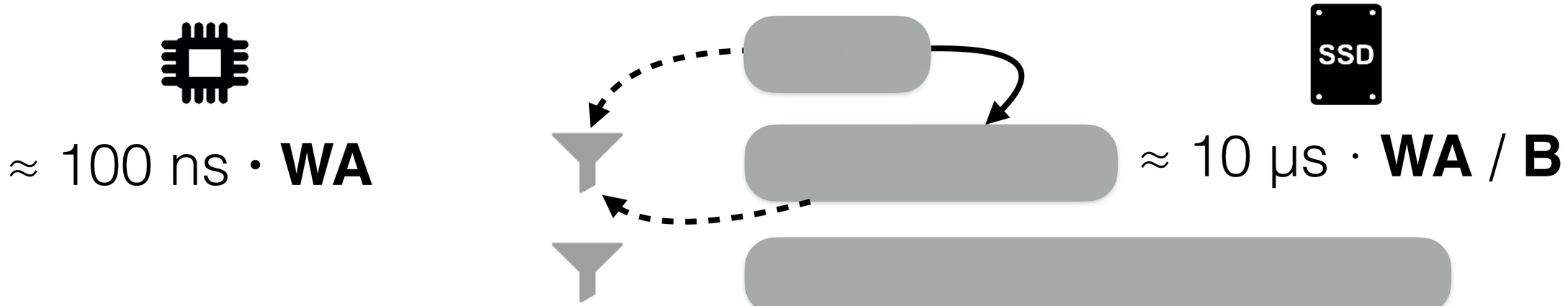
≈ 100 ns



≈ 10 µs (optane)

Problem: shrinking memory/storage speed gap

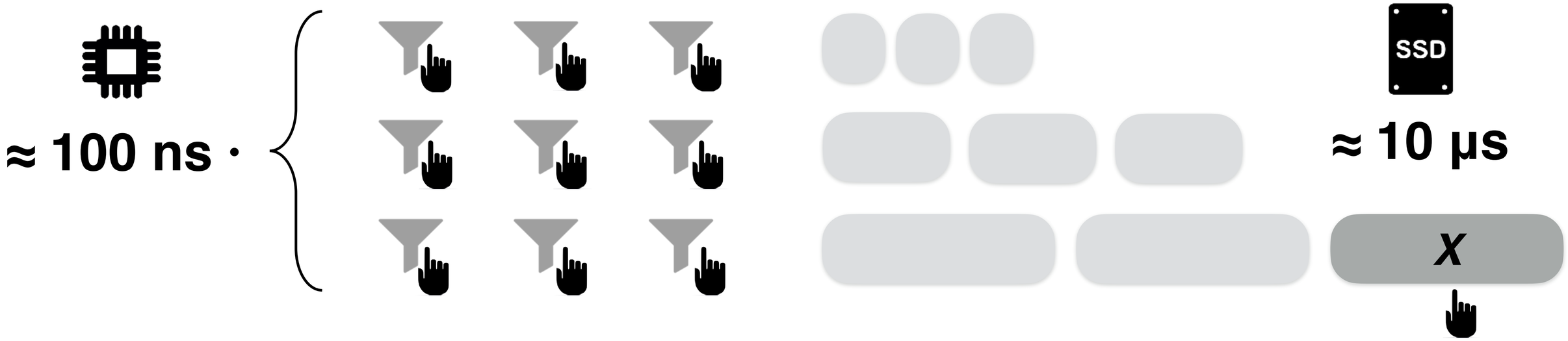
(1) write amplification



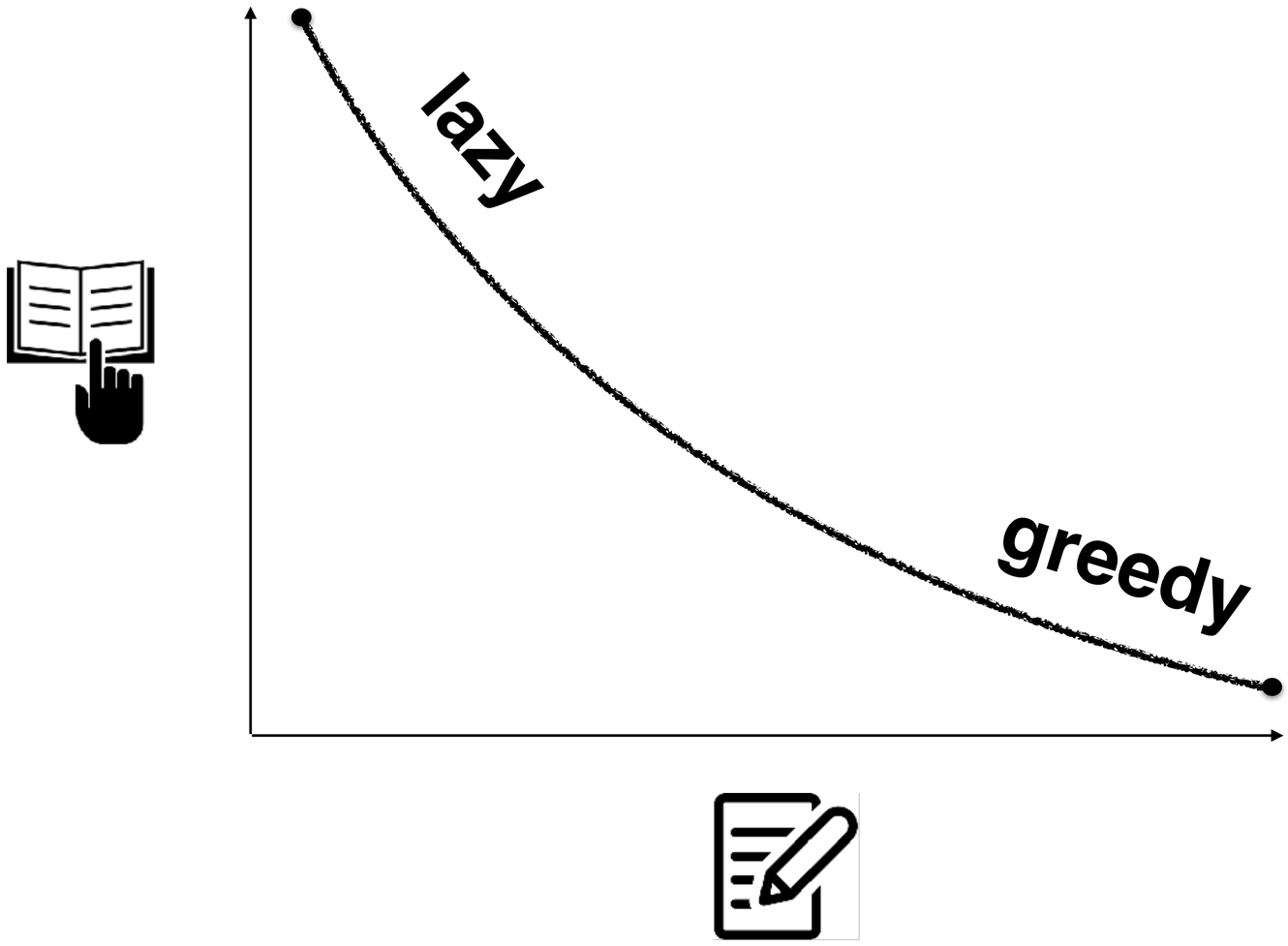
Problem: shrinking memory/storage speed gap

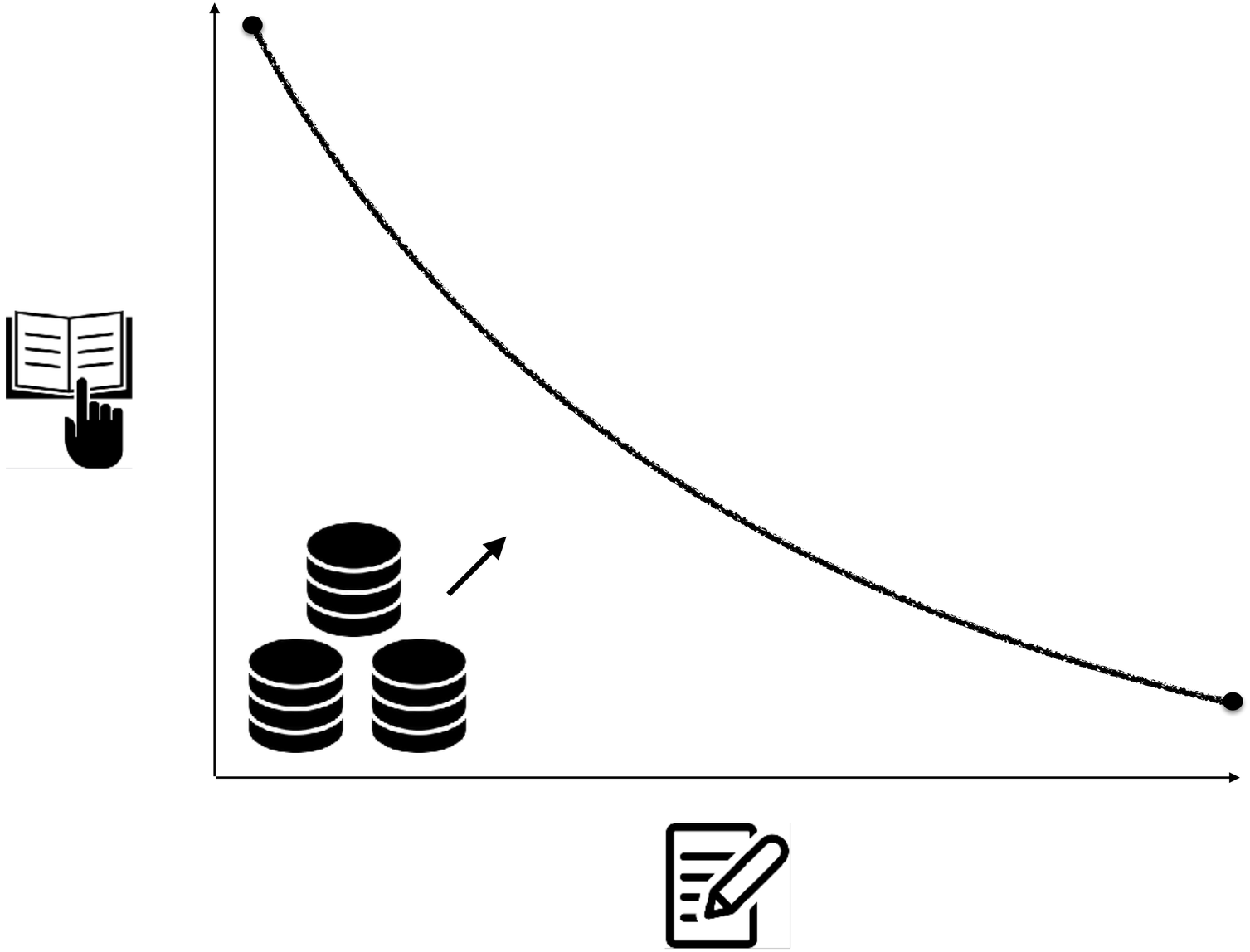
(1) write amplification

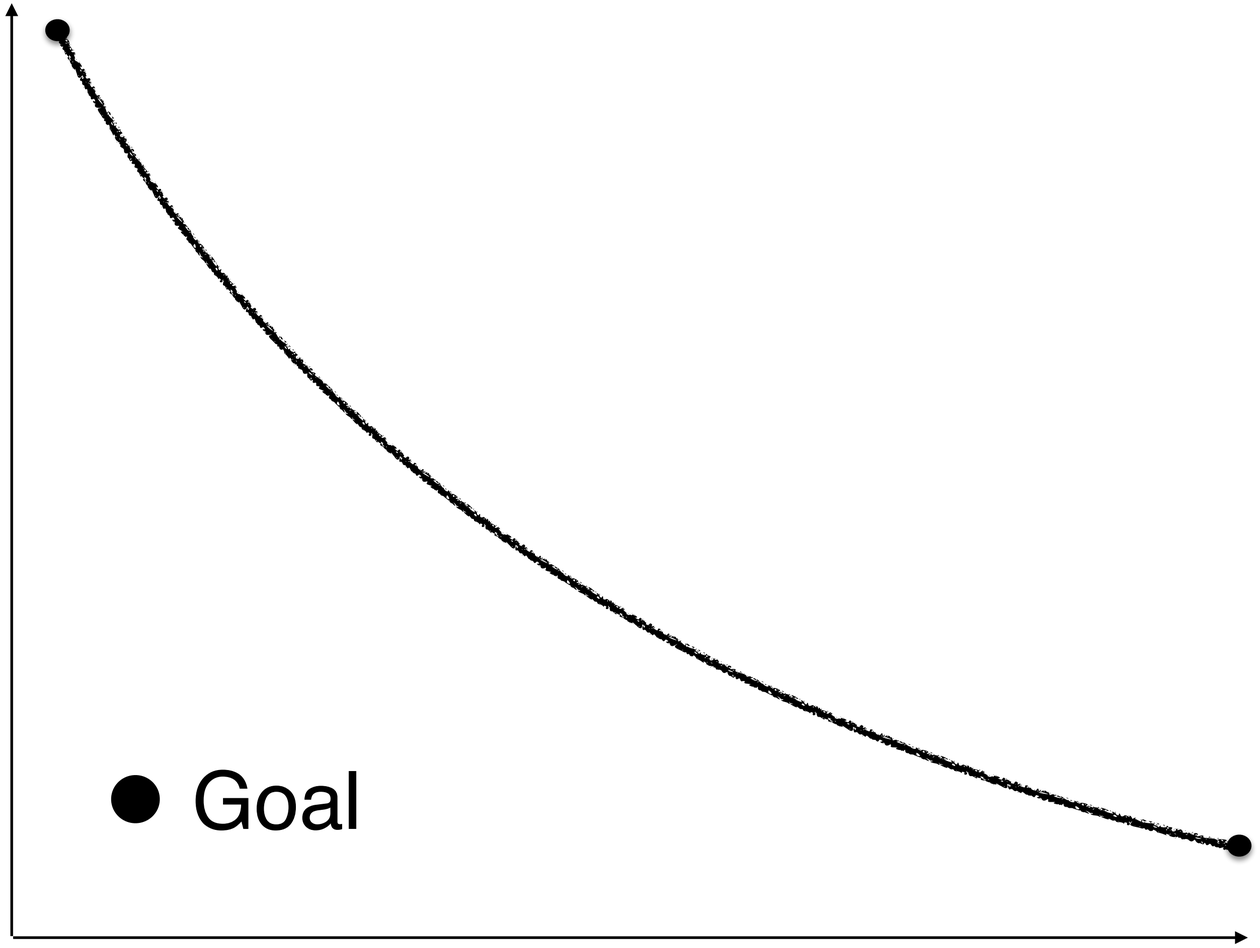
(2) read amplification

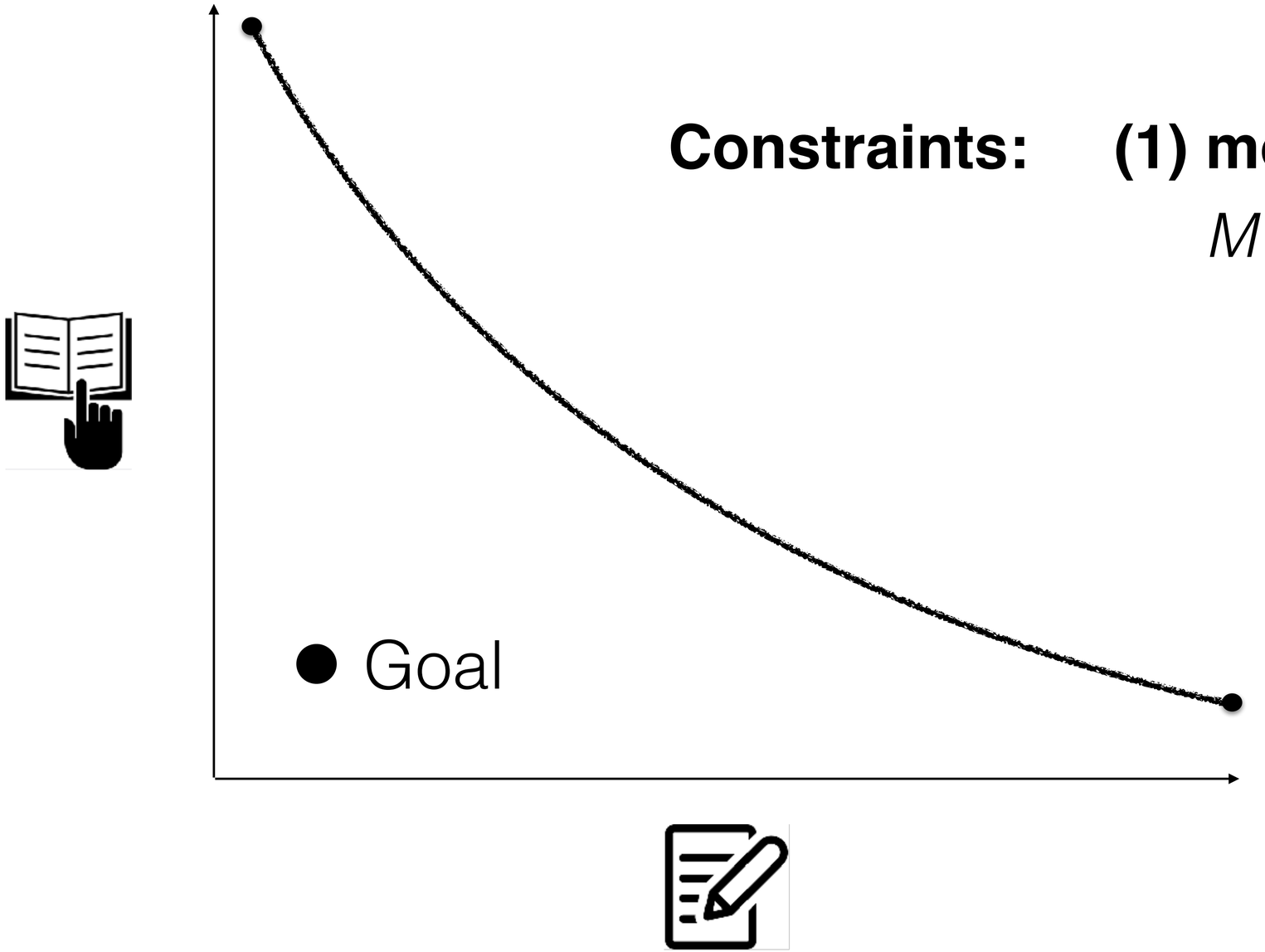


read / write contention



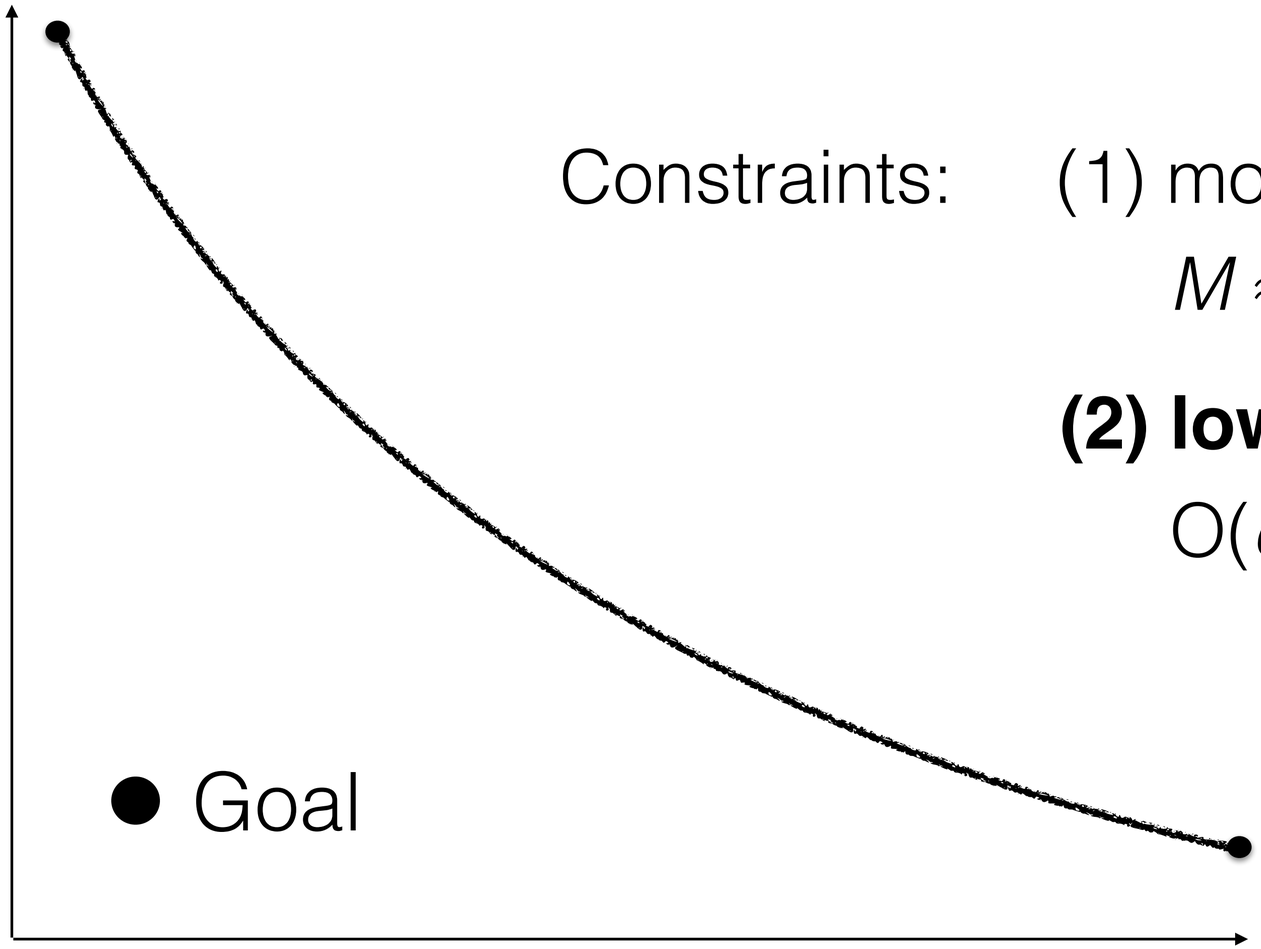



Goal



Constraints: **(1) modest memory footprint**

$M \approx 10$ bits / entry



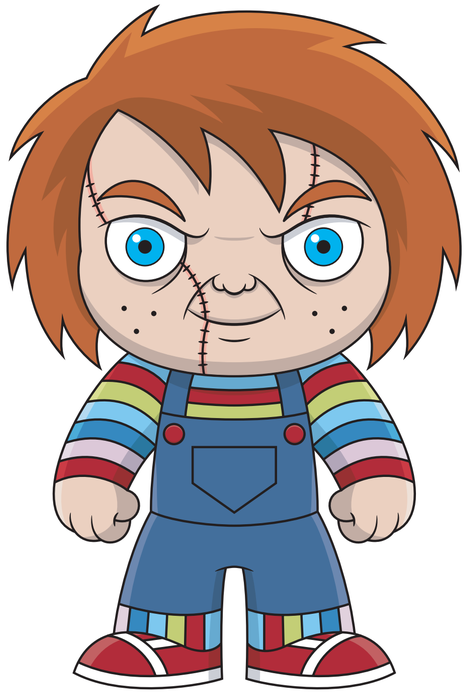
Constraints: (1) modest memory footprint

$M \approx 10$ bits / entry

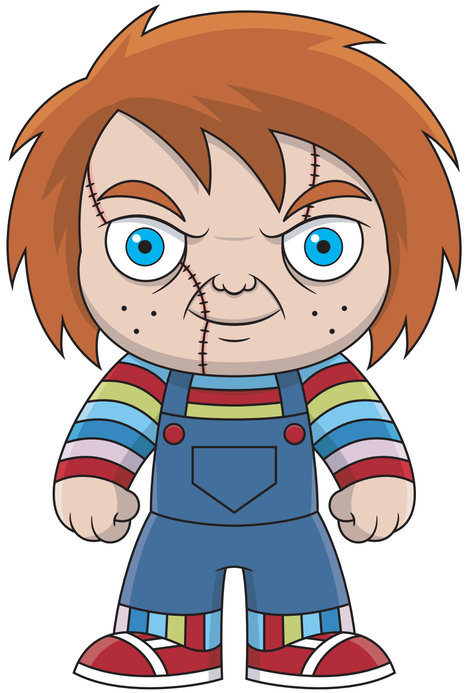
(2) low false positive rate

$O(e^{-M \cdot \ln(2)})$

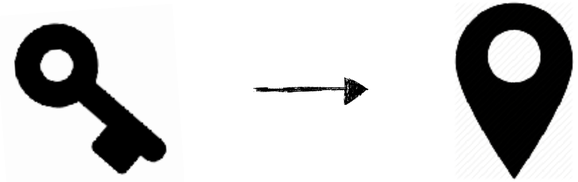
Chucky



Chucky: **hu**ffman **c**oded **k**e**y**-value store



hash table



key **level**

level ID

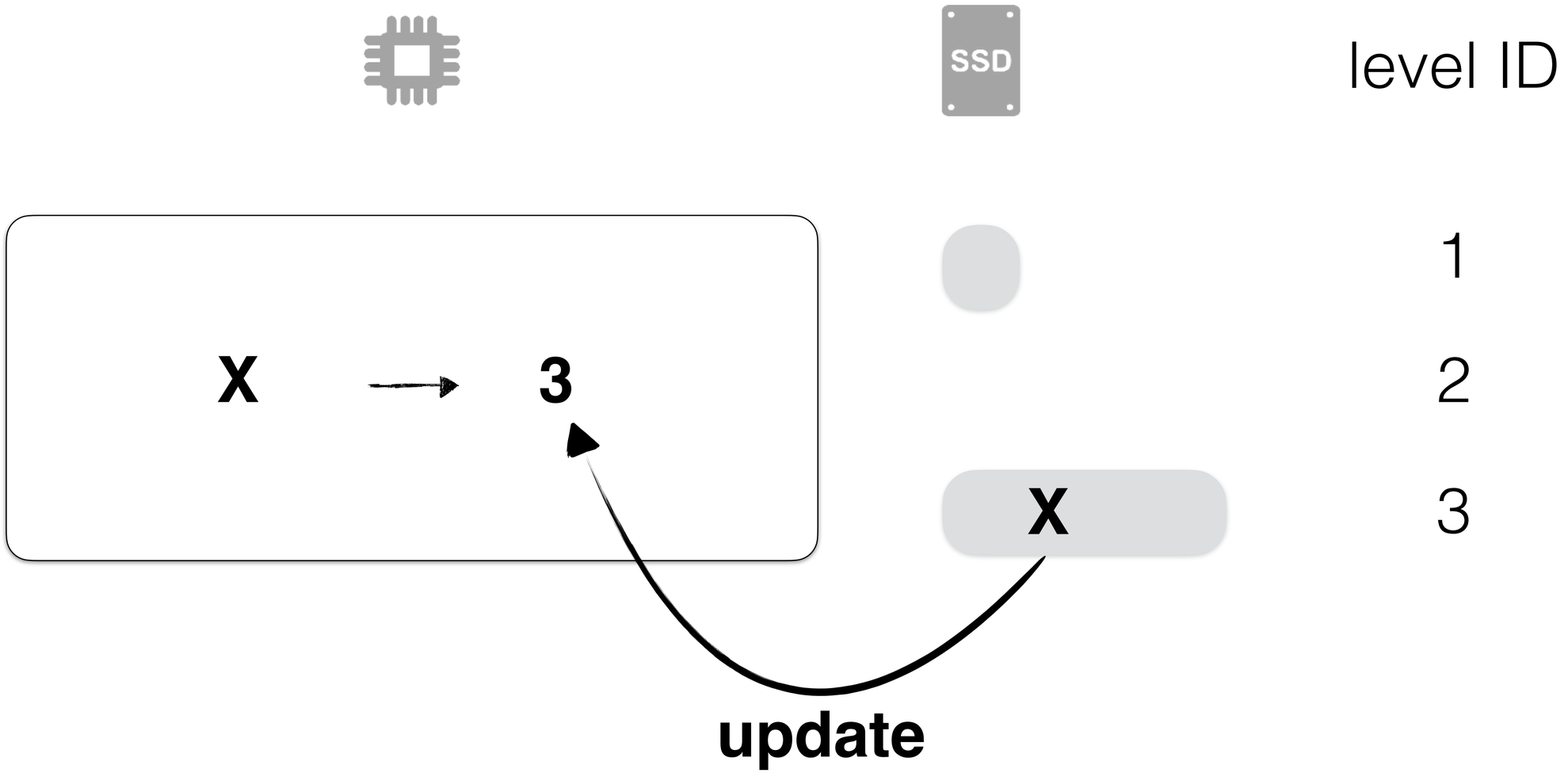
X → 2

X

1

2

3


SSD
level ID
X
3
1
2
X
3
update

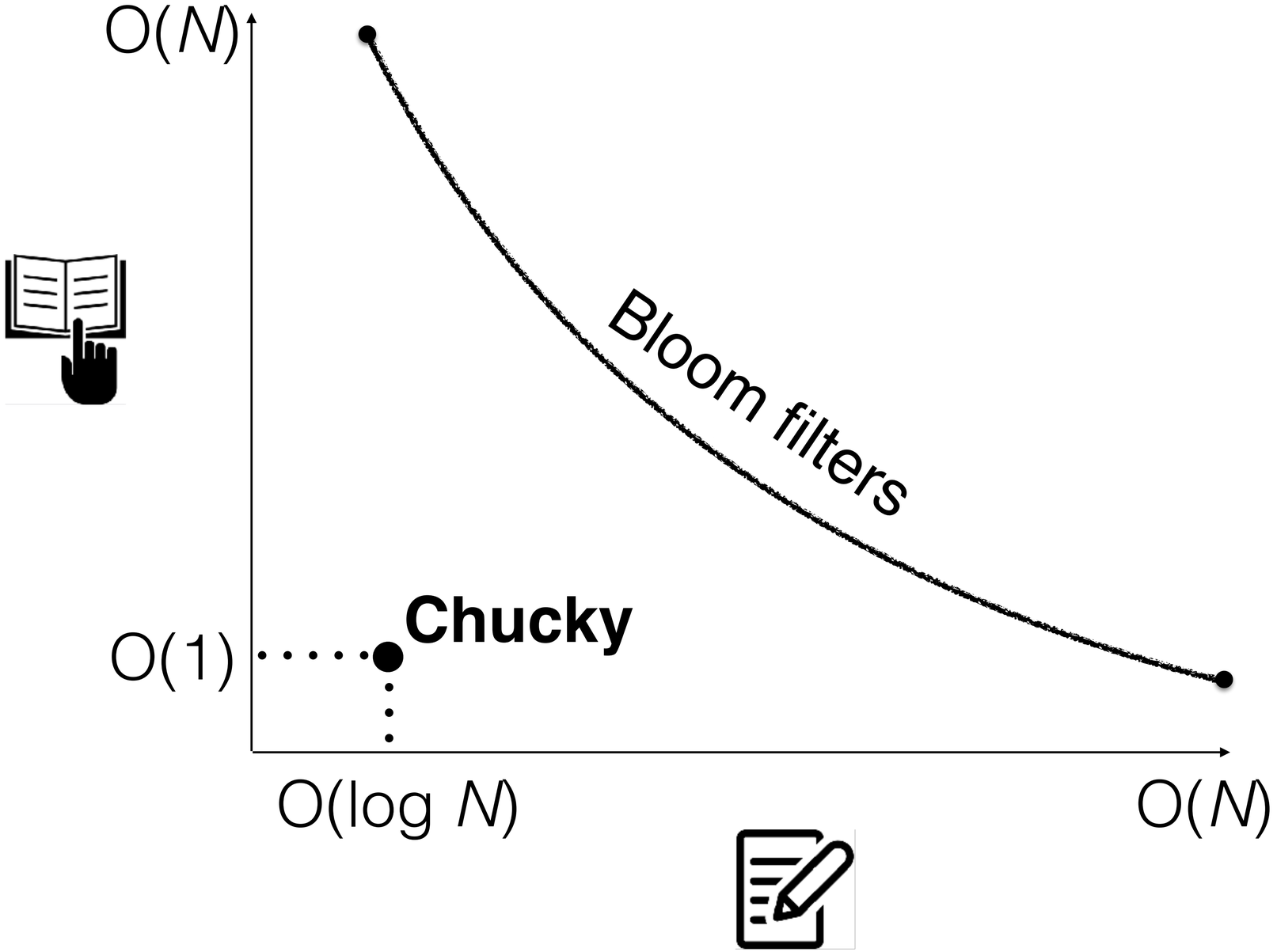
run

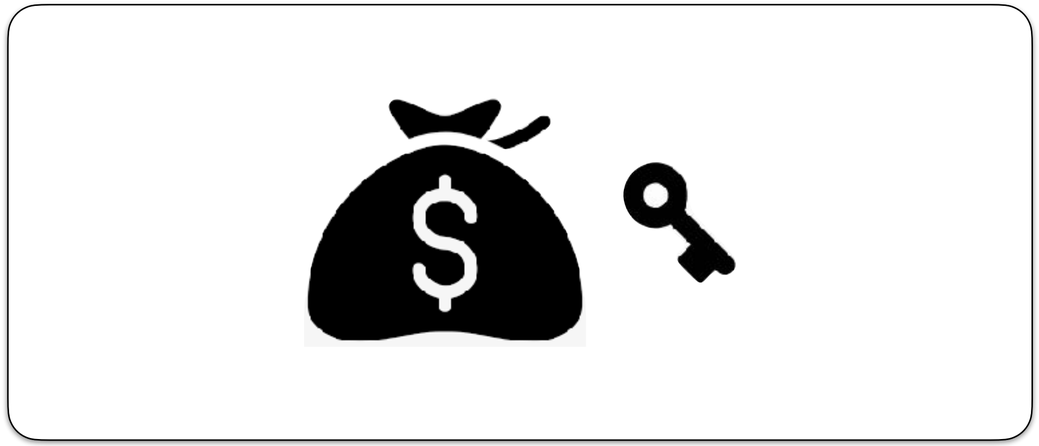
lazy

O(1) reads

O(1) reads 

O(log *N*) updates 


O(*N*)
Bloom filters
Chucky
O(1)
O(log *N*)
O(*N*)



memory ≫ 10 bits / entry

hash() = 

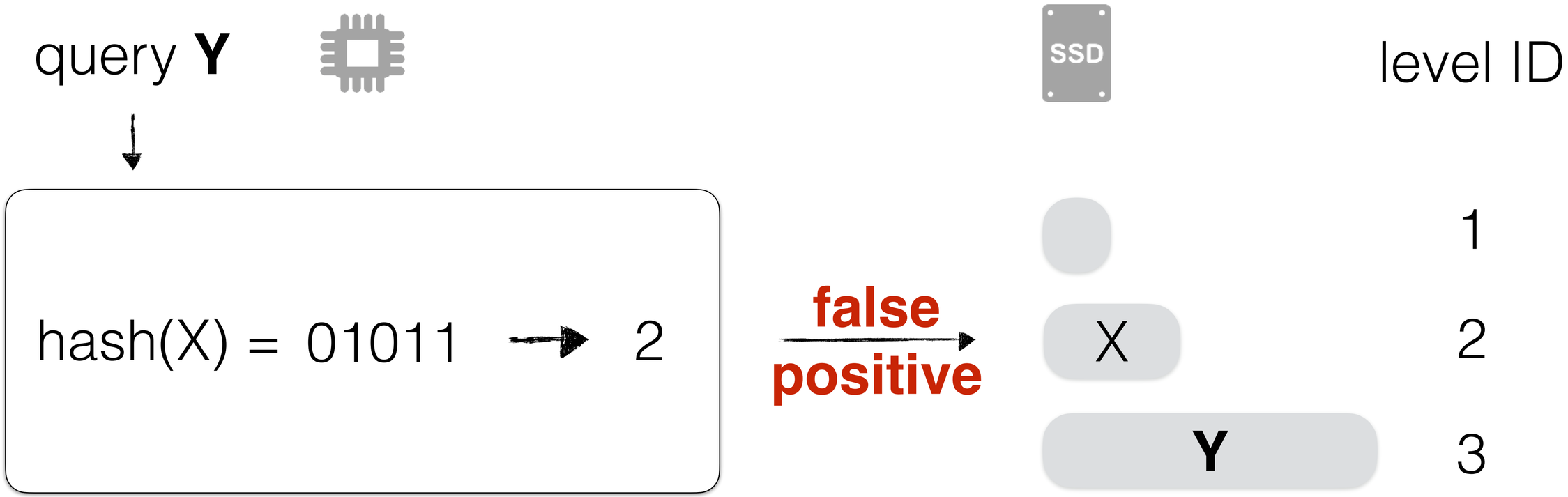
level ID

hash(**X**) = 01011 → **2**

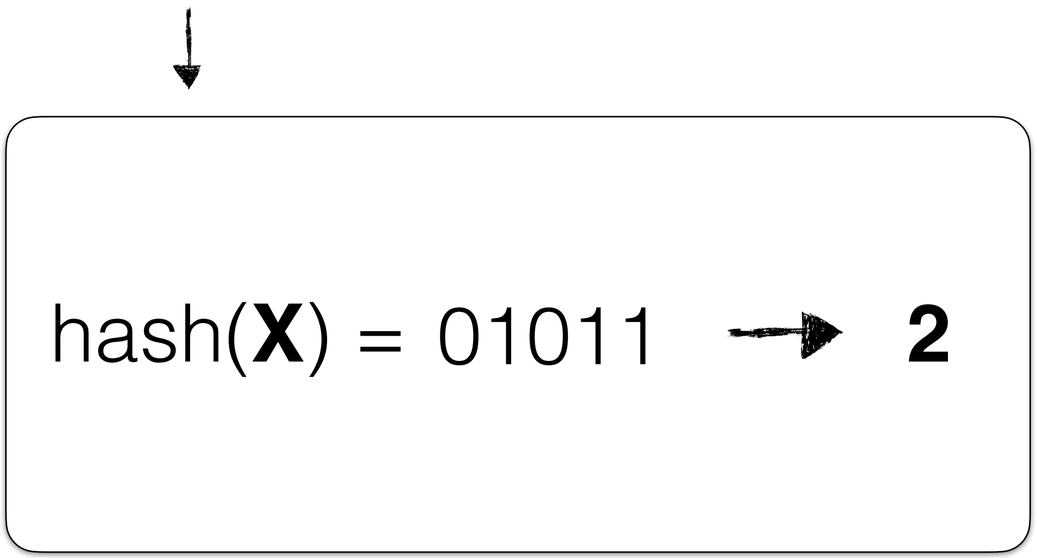
1

X **2**

3

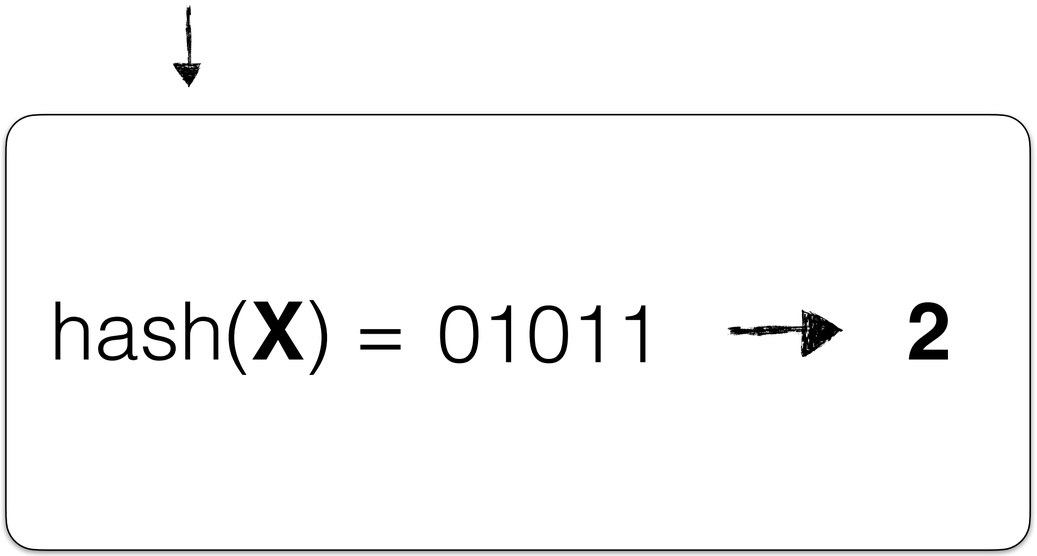

query **Y**
SSD
level ID
hash(X) = 01011
2
false positive
1
X
2
Y
3

query **Y**



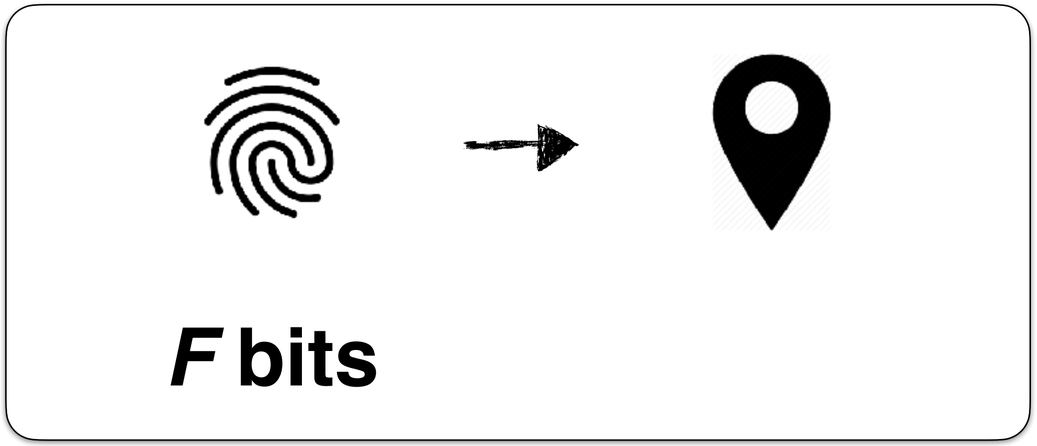
false positive rate?

query **Y**



false
positive rate?

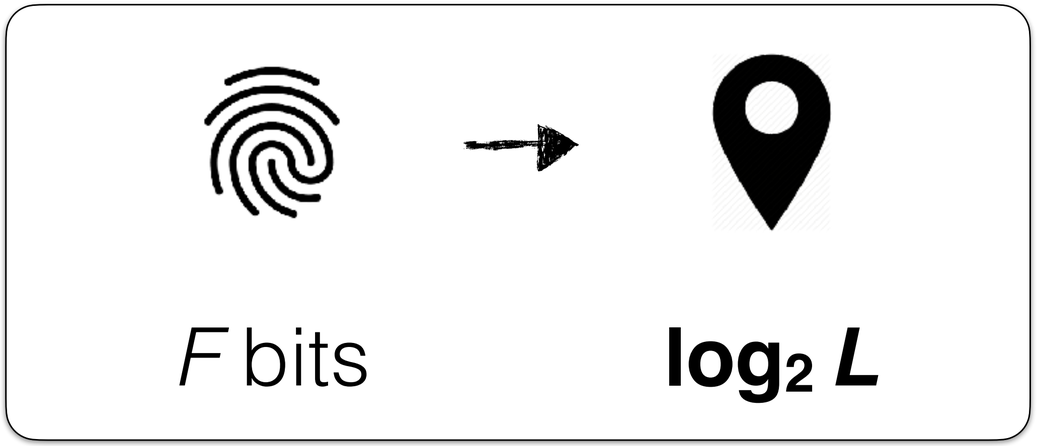
(Assume minimal perfect hashing)



false
positive rate?

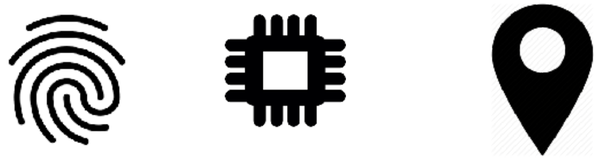
2^{-F}

_M_ bits



false
positive rate?

2^{-F}



$$F = M - \log_2 L$$

false
positive rate?

$$2^{-F}$$

Bloom filters

$\approx e^{-M \cdot \ln(2)}$

<

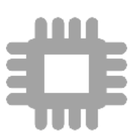
Chucky (so far)

$\mathbf{2^{-M} \cdot \log N}$

level IDs grow,
taking bits from fingerprints

$\approx e^{-M \cdot \ln(2)}$ **<** $2^{-M} \cdot \log N$

encode larger levels with fewer bits



1
2
2
3
3
3
3

most entries

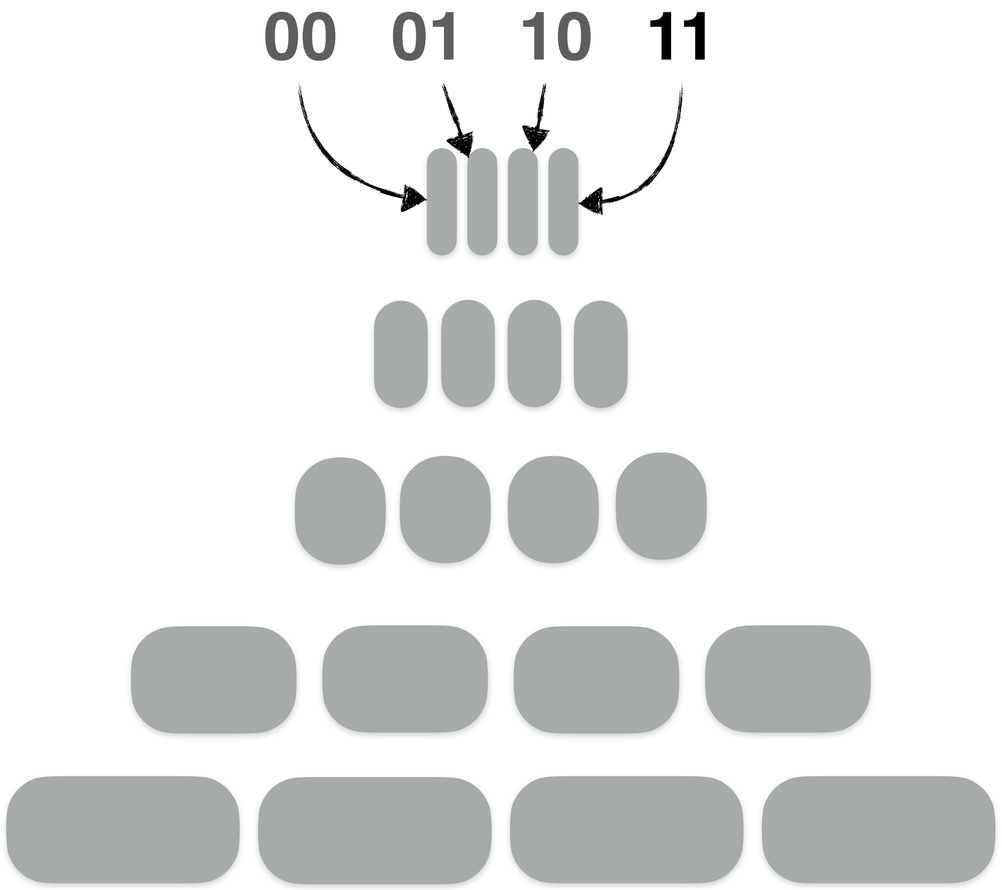
Level IDs

1

2

3

unary encoding	level ID
00001	1
0001	2
001	3
01	4
1	5



unary encoding	level ID
00001___	1
0001___	2
001___	3
01___	4
1___	5

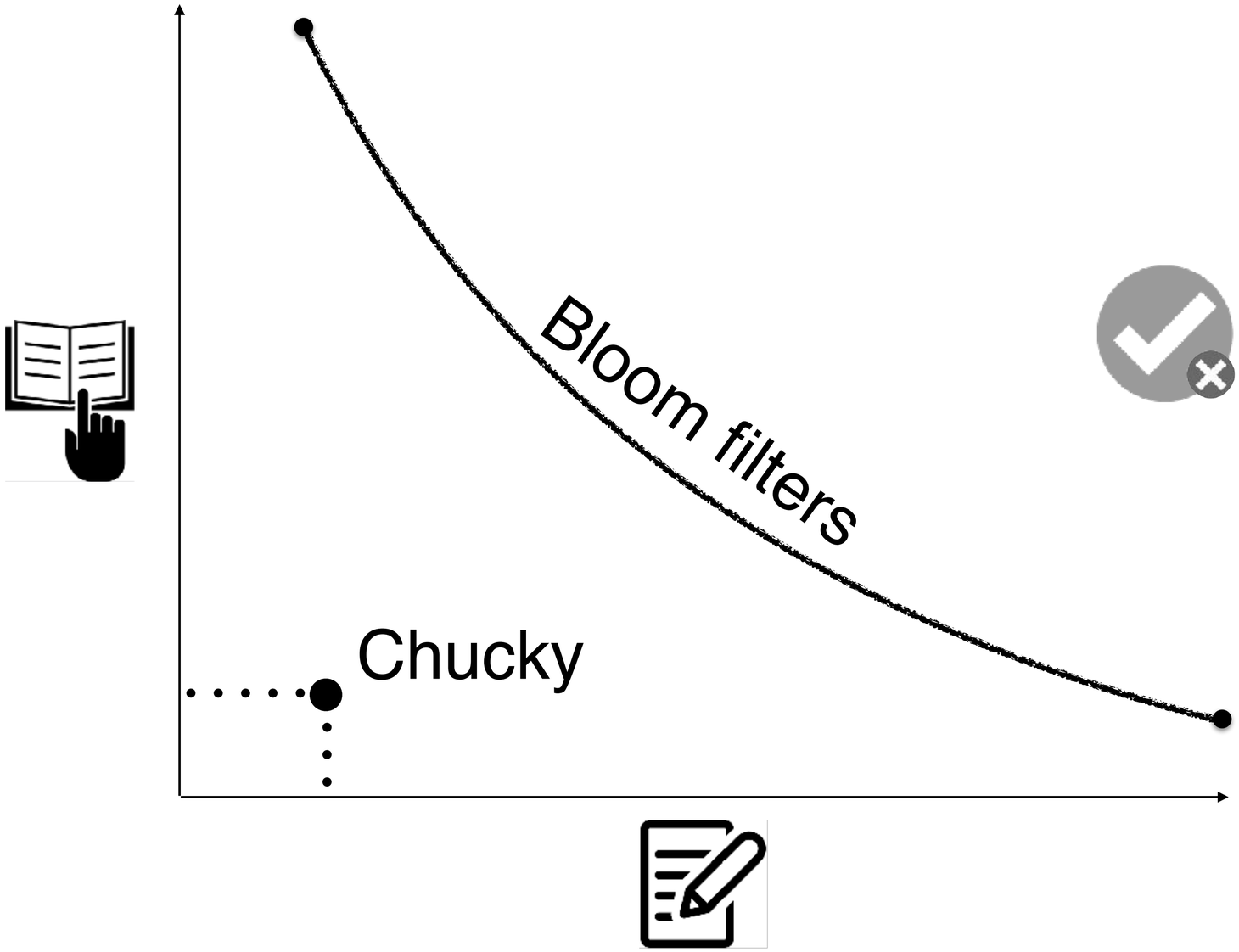
unary encoding	level ID
00001	1
0001	2
001	3
01	4
1	5

less common

average level ID length < 2 bits < log *N*

unary encoding	level ID
00001	1
0001	2
001	3
01	4
1	5

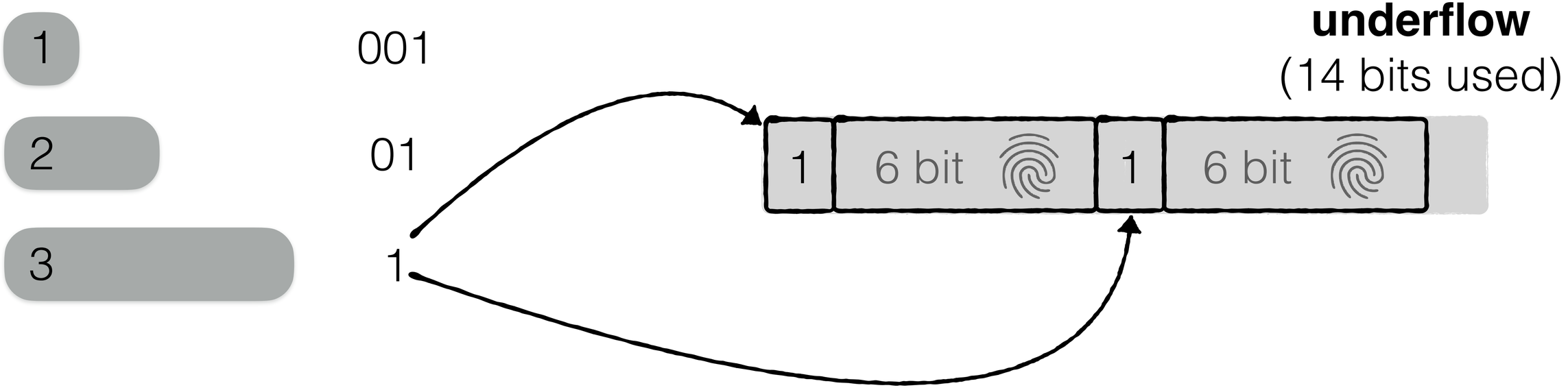
$$F = M - 2$$

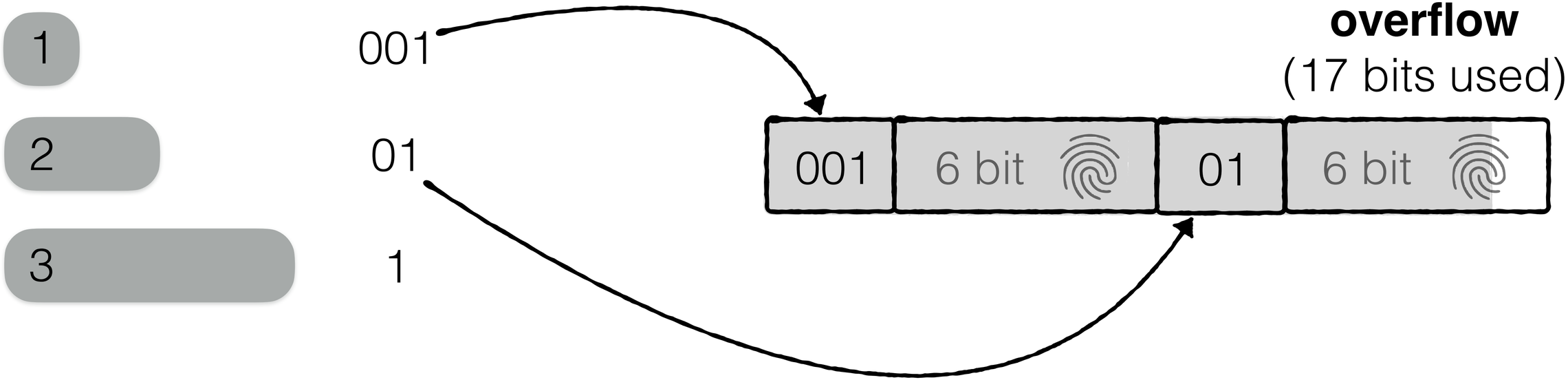


Bloom filters Chucky

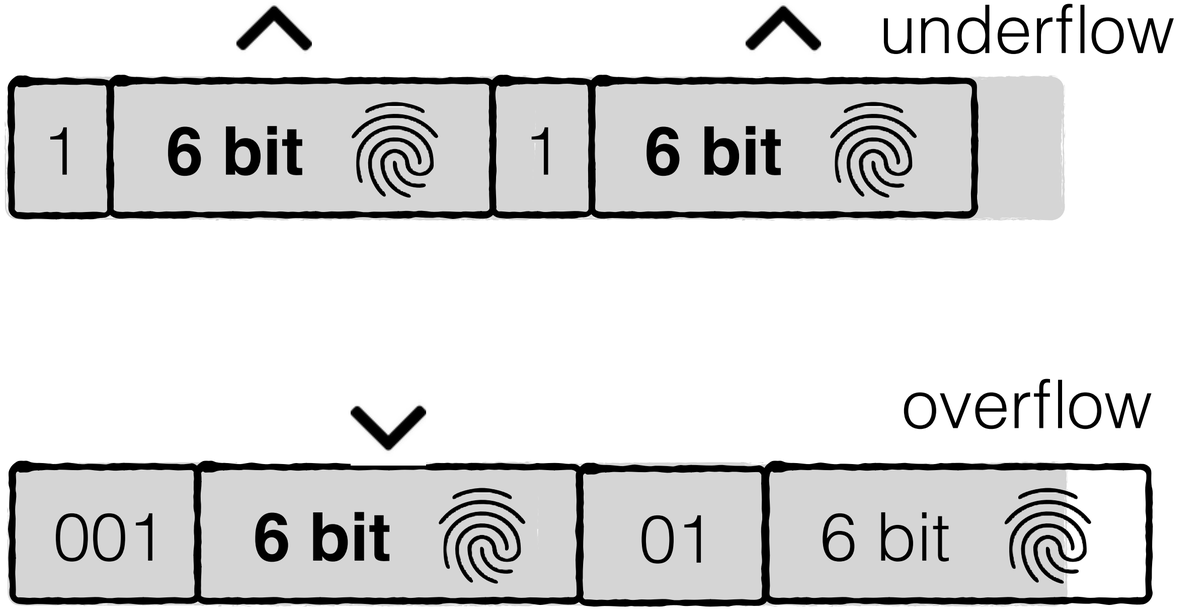
$O(e^{-M \cdot ln(2)}) \approx O(2^{-M})$

Level	code	some bucket
1	001	
2	01	16 bit, 2 entries
3	1	

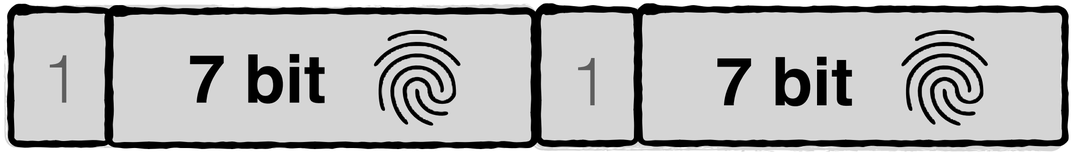

1
2
3
001
01
1
1
6 bit
1
6 bit
underflow
(14 bits used)

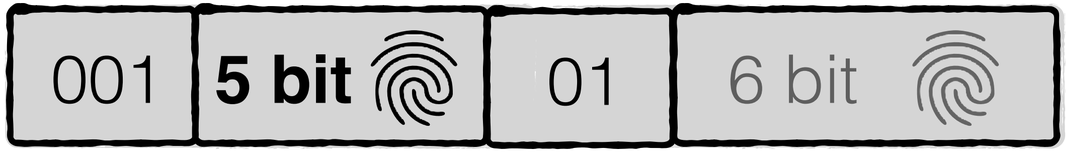

1
2
3
001
01
1
overflow
(17 bits used)
001
6 bit
01
6 bit

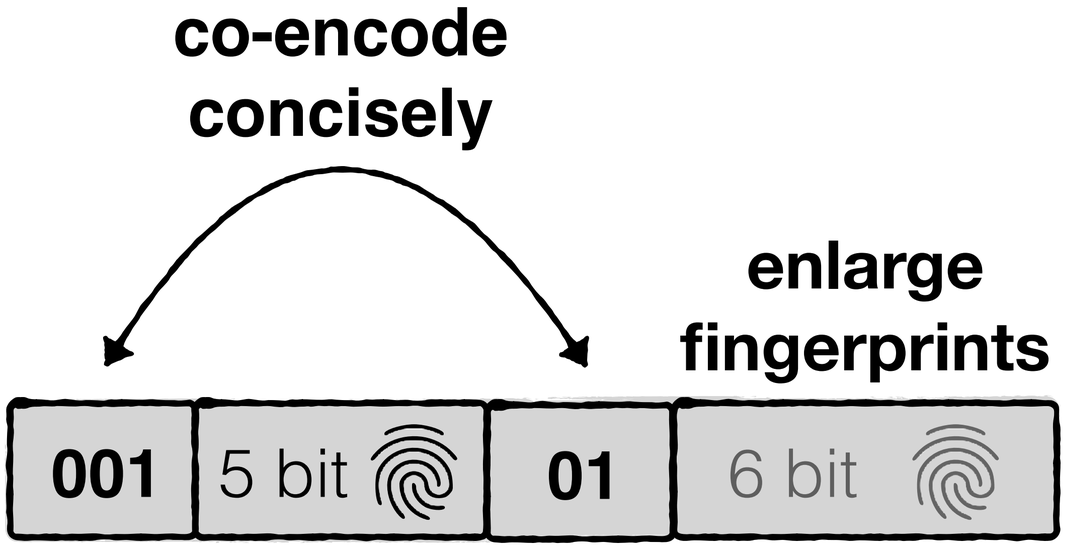
larger fingerprints for larger levels



larger fingerprints
for larger levels






co-encode
concisely
enlarge
fingerprints
001
5 bit
01
6 bit


0011
5 bit
7 bit

Encoding Level IDs as combinations

levels **3 & 3**

levels **2 & 2**

levels **3 & 2**

levels **2 & 1**

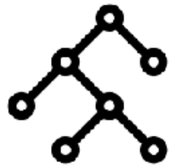
levels **3 & 1**

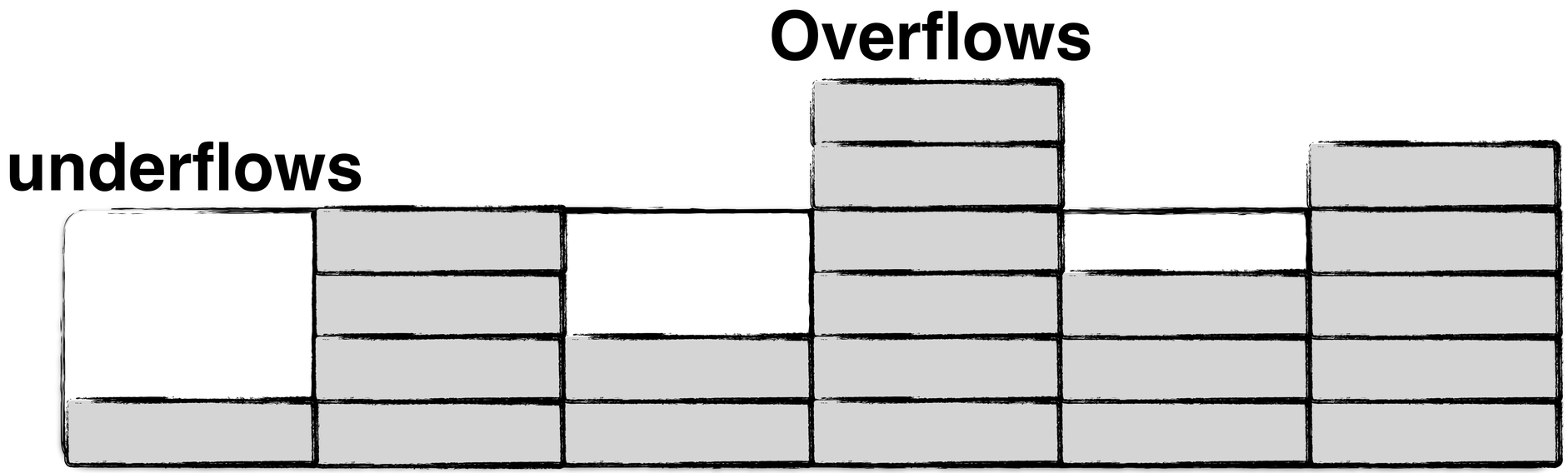
levels **1 & 1**

	task	technique
Step 1		
Step 2		

	task	technique
Step 1	**max. avg.**  **size** **unique decodability** 	
Step 2		

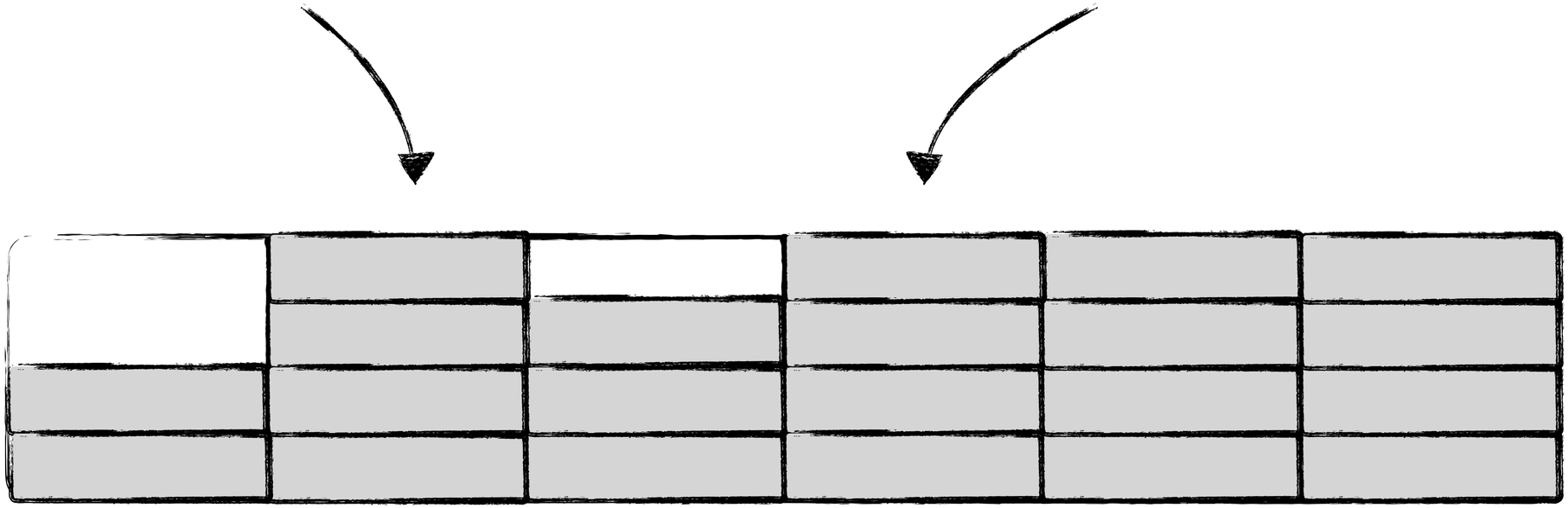
	task	technique
Step 1	max. avg.  size unique decodability 	
Step 2	**generate codes** 	

	task	technique
Step 1	max. avg.  size unique decodability 	**hill-climbing**  **Kraft–McMillan** ≠
Step 2	generate codes 	**Huffman Tree** 

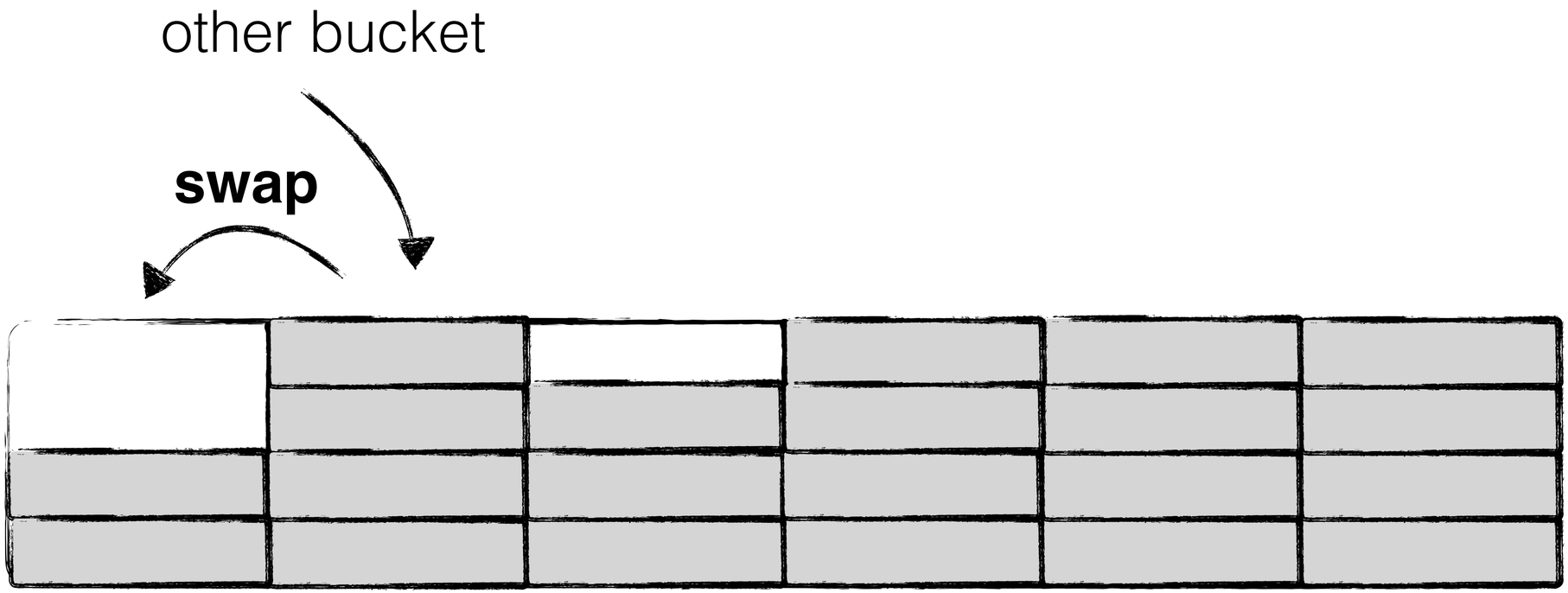

Overflows
underflows

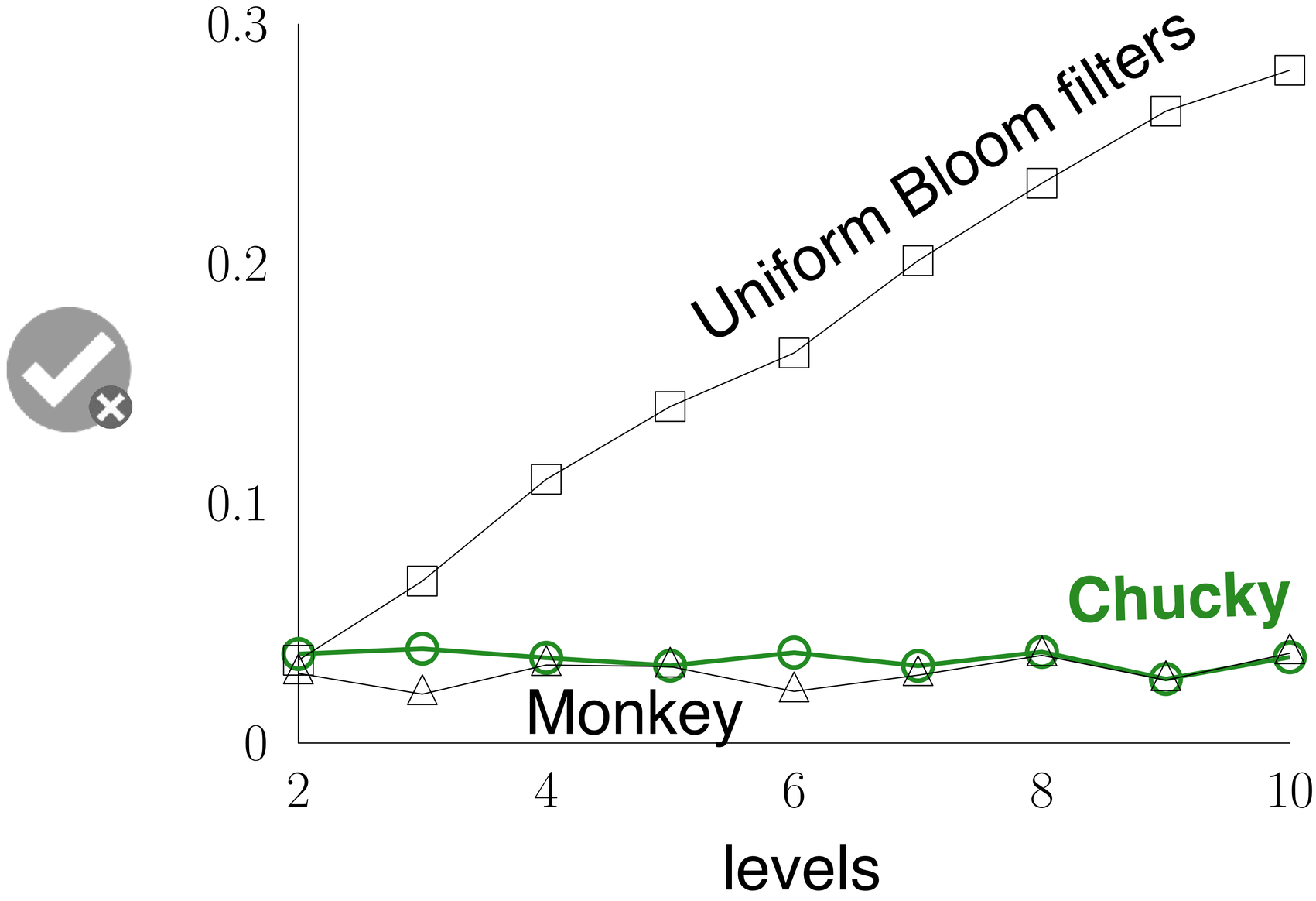
Cuckoo Filter

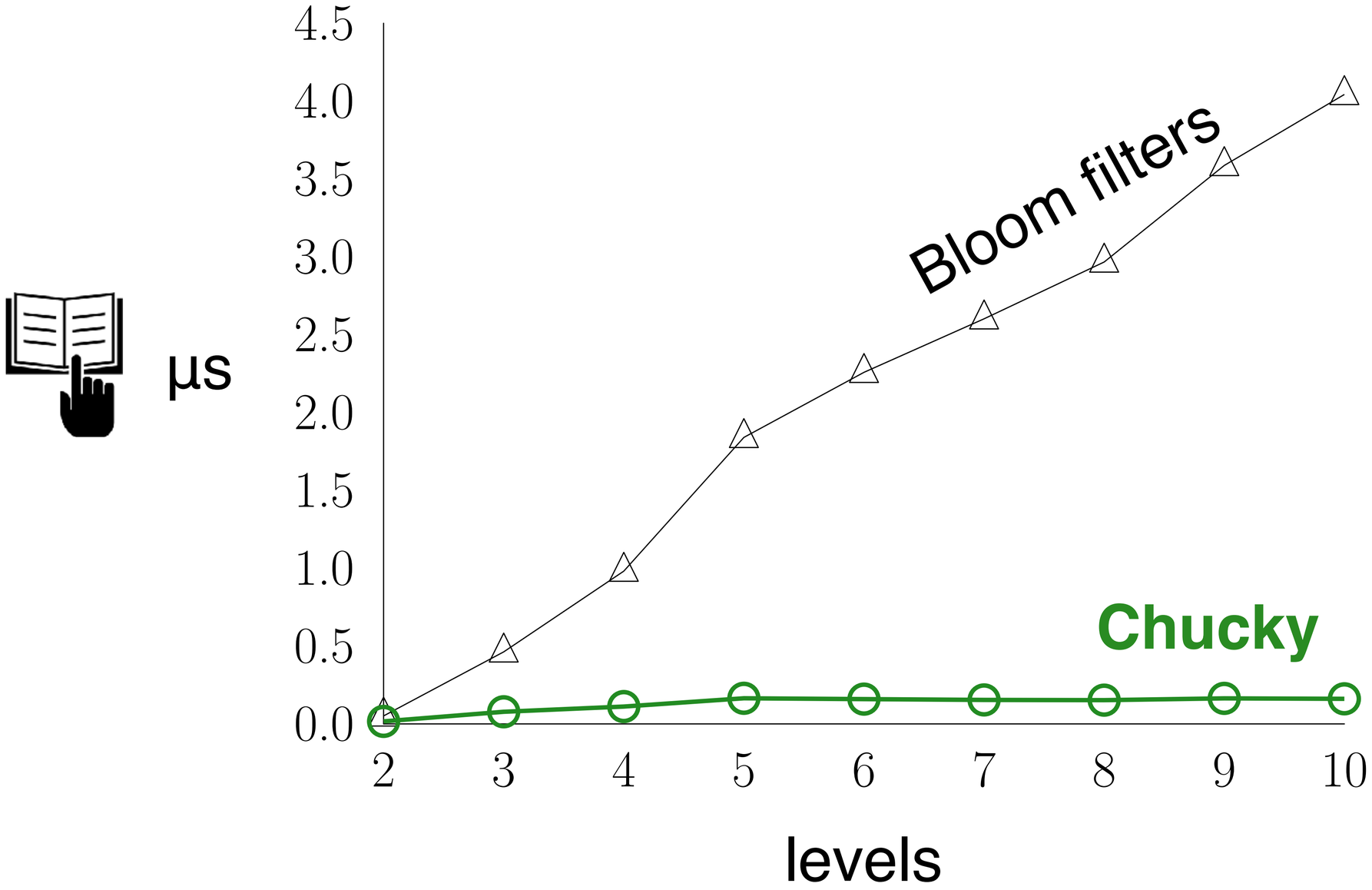
other bucket = hash() ⊕ current bucket

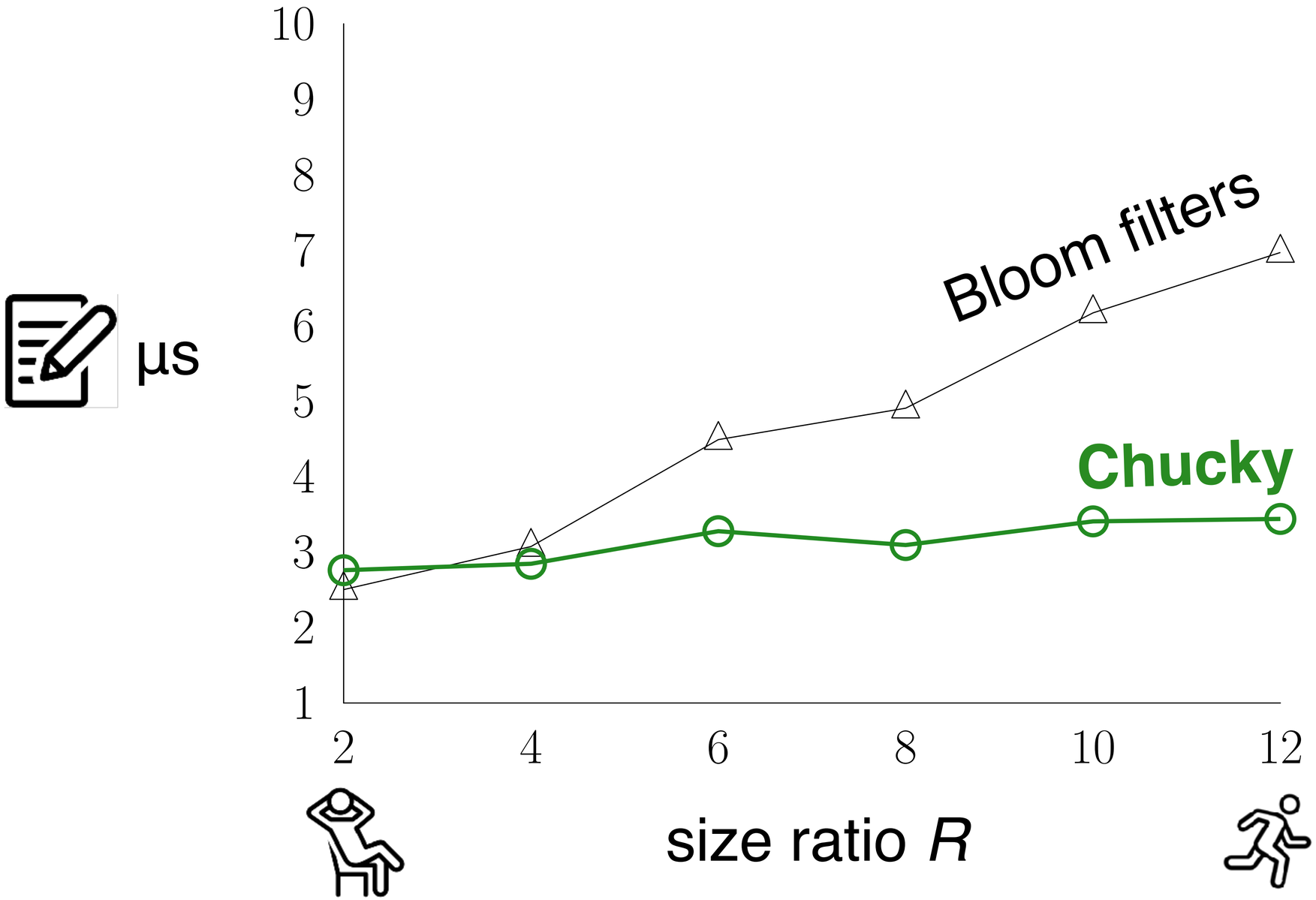


Cuckoo Filter

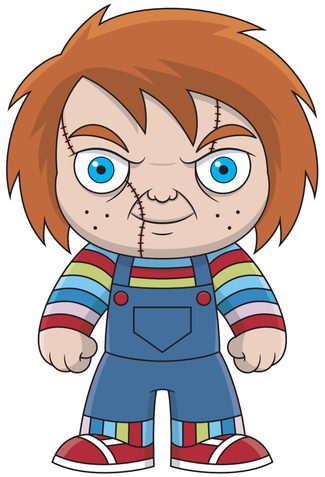


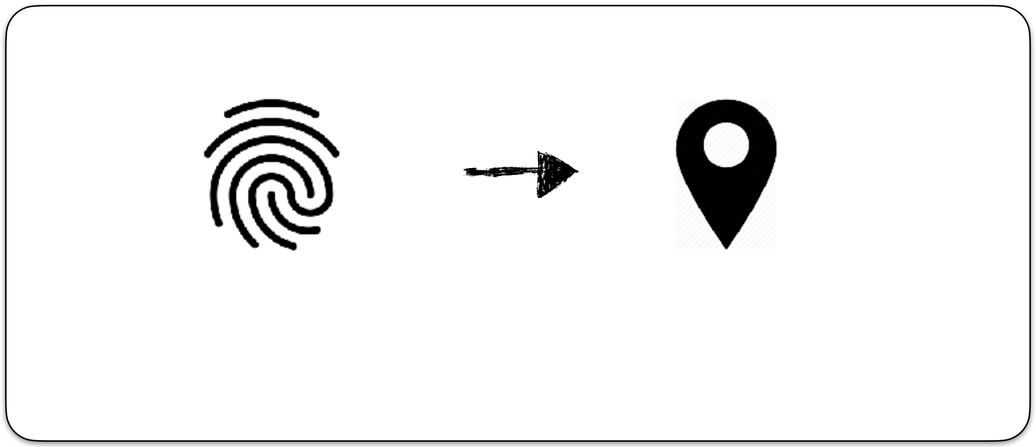

0.3
0.2
0.1
0
2
4
6
8
10
levels
Uniform Bloom filters
Chucky
Monkey


µs
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2
3
4
5
6
7
8
9
10
levels
Bloom filters
Chucky

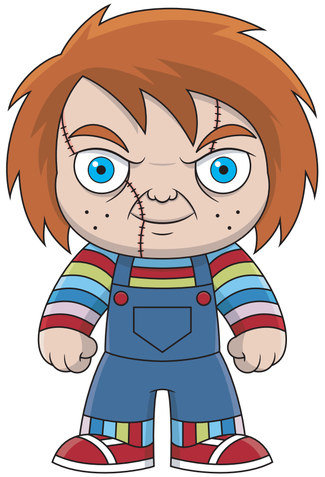

µs
10
9
8
7
6
5
4
3
2
1
2
4
6
8
10
12
Bloom filters
Chucky
size ratio R

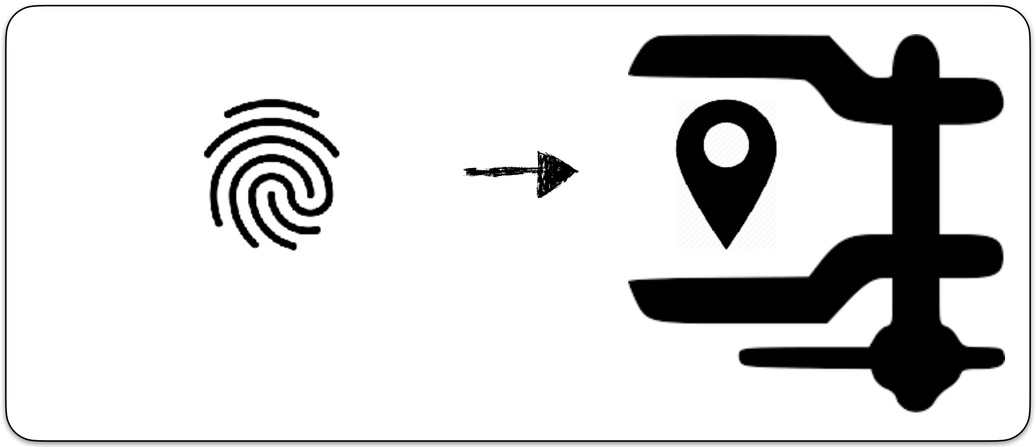
Chucky



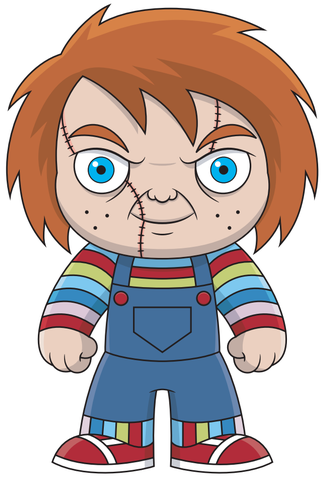


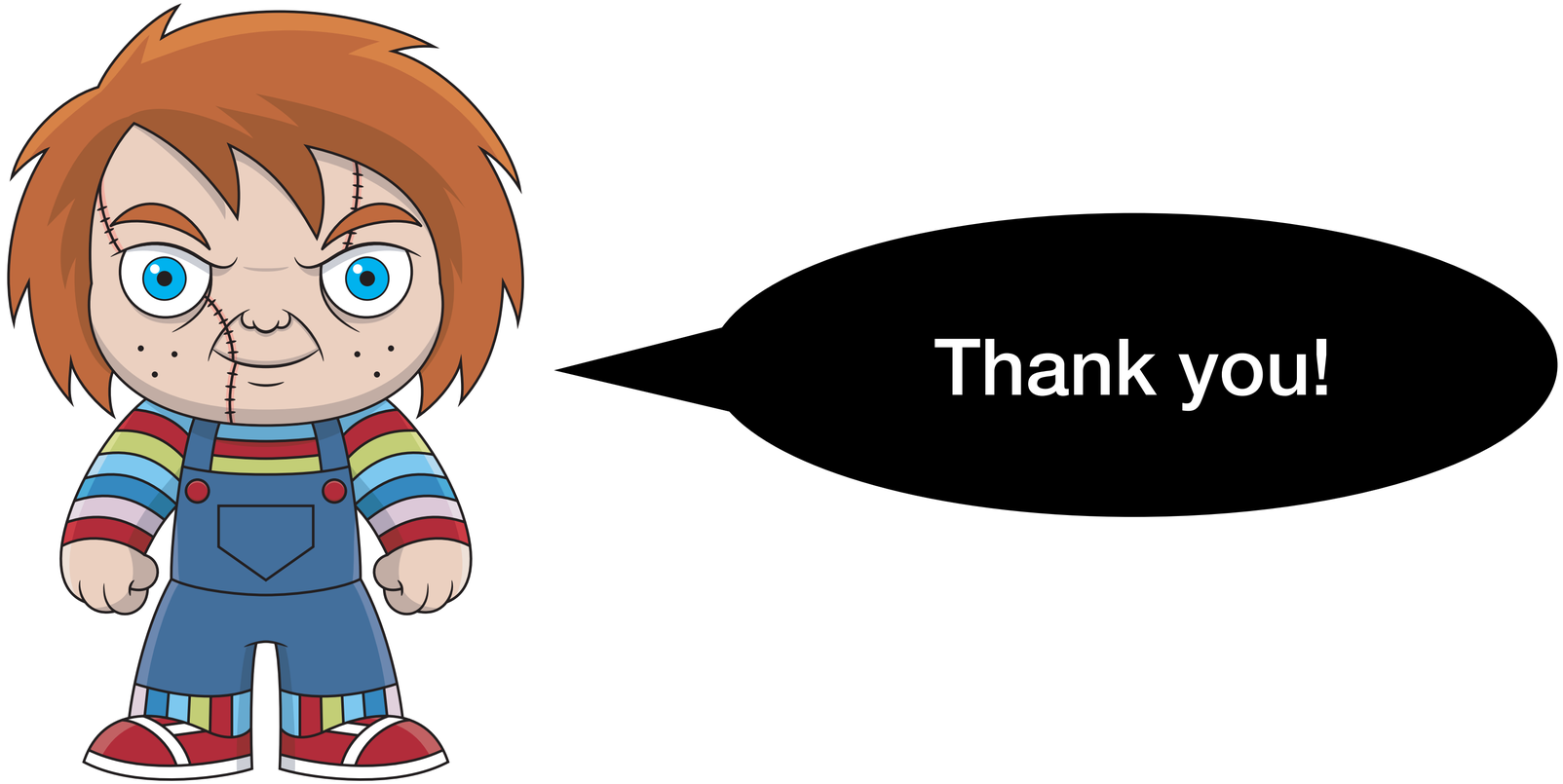
Chucky





Chucky




Thank you!