

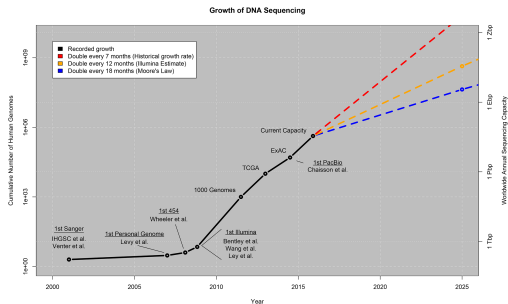
HowDeSBT and Simka wedding:

What has been done and what we plan to do?

Téo Lemane, Paul Medvedev, Rayan Chikhi, Pierre Peterlongo

DSB meeting
4 February 2020

High production of sequencing data



- ▶ Non-indexed datasets
- ▶ Query only the metadata, not the sequences content

ATCGAAGCACCAAAAATTACAGACGGGG

All Maps Images Videos News More Settings

Your search - **ATCGAAGCACCAAAAATTACAGACGGGG** - did not match any documents.

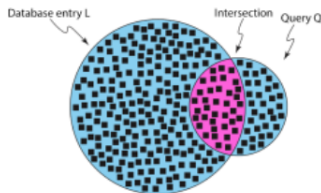
Stephens *et al.*

Query a sequence of interest?

Given experiment sets, and a sequence of interest, which dataset contains this sequence?

In terms of k-mers:

A query Q matches an experiment L if at least a fraction θ of Q 's k-mers are present in L .



k-mers based indexing: general processing

- ▶ k-mers counting for each dataset
 - ▶ dealing with sequencing errors
- ▶ indexing each set of k-mers (using AMQ for example)
 - ▶ with or without the counting information, depending on the data structure used
- ▶ merging each index in one
 - ▶ with possible adjustments to optimize storage and query

Example: SBT, HowDeSBT, BIGSI ...

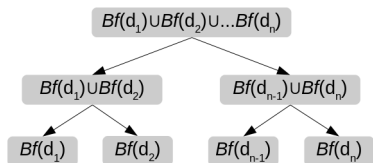
Common to all methods: k-mers counting

Dealing with sequencing errors

- ▶ Usually, k-mer counters use simple error correction (throw out k-mers whose abundance are below a hard threshold)
- ▶ low abundance k-mer: not necessarily an error (under-represented sequence)
- ▶ metagenomics, transcriptomics, cancer genomics (Griffith *et al.*)

HowDeSBT¹: main idea

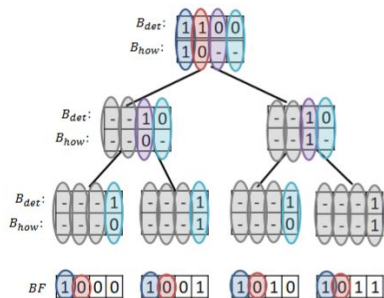
- ▶ Indexing datasets using their k-mers content
- ▶ Bloom filters as a basic structure
- ▶ Final index based on Sequence Bloom Tree (Solomon and Kingsford)



- ▶ One leaf = one experiment.
- ▶ Internal node represents k-mers in its subtree.
- ▶ Root node is the union of each BF in the tree.

¹ Harris and Medvedev

HowDeSBT: data structure

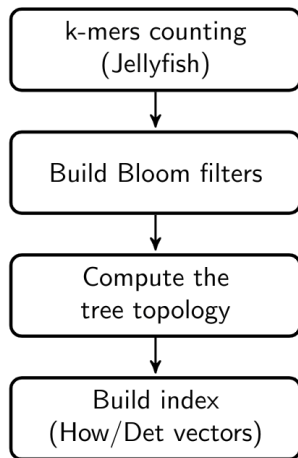


Two binary vectors by node:

- B_{det} : Bit is **determined** in the subtree?
- B_{how} : If **det**, **how** is it determined? (0/1)

Medvedev, DSB 2019

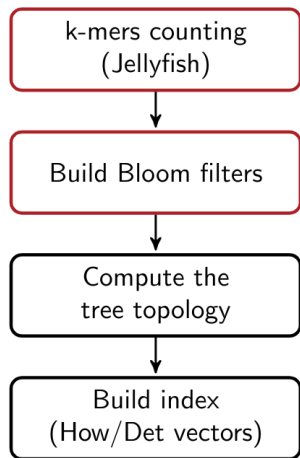
HowDeSBT: pipeline



What we want to do?

- ▶ Include particular error handling
- ▶ At the same time: can computation times and memory footprint be improved?

HowDeSBT: pipeline

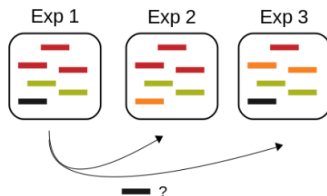


What we want to do?

- ▶ Include particular error handling
- ▶ At the same time: can computation times and memory footprint be improved?
- ▶ Focus on the two first steps

How to improve errors handling?

- ▶ Leverage information across samples
- ▶ e.g. For a k-mer seen less than N times, check the count in the other datasets.



- ▶ Verification only in datasets with compatible metadata

How? Help from Simka (Benoit *et al.*)

- ▶ *de novo* comparative metagenomics tool.
- ▶ Compute ecological distances between samples using k-mers decomposition.

	A	B	C	D
A	0	0.2	0.1	0.4
B	0.2	0	0.6	0.3
C	0.1	0.6	0	0.5
D	0.4	0.3	0.5	0

k-mers matrix

- After modifications, we can obtain a count table with datasets in columns and k-mers in lines.

	<u>A</u>	<u>B</u>	<u>C</u>	<u>D</u>
ACGT	12	16	7	1
ACTG	1	0	0	0
CTGA	8	1	8	0
GATA	21	10	21	20
TCGA	12	0	0	4
TCGT	0	1	0	1

k-mers matrix

- ▶ Leverage information across samples:
 - ▶ easily applicable with this kind of matrix
- ▶ Add this new errors handling feature in Simka
- ▶ Many ways to do it

Count table

	<u>A</u>	<u>B</u>	<u>C</u>	<u>D</u>
ACGT	12	16	7	1
ACTG	<u>1</u>	0	0	0
CTGA	8	<u>1</u>	8	0
GATA	21	10	21	20
TCGA	12	0	0	4
TCGT	0	<u>1</u>	0	<u>1</u>

Binary matrix

	<u>A</u>	<u>B</u>	<u>C</u>	<u>D</u>
ACGT	1	1	1	1
ACTG				
CTGA	1	<u>1</u>	1	0
GATA	1	1	1	1
TCGA	1	0	0	1
TCGT	0	<u>1</u>	0	<u>1</u>

Simka: How it works ? - Reminder

- Minimizers & super-k-mers (KMC 2, Deorowicz *et al.*)

```

Read  CTCATGCACGTTC
k-mers CTCATG
      TCATGC
      CATGCA
      ATGCAC
      TGCACG
      GCACGT
      CACGTT
      ACGTTTC
  
```

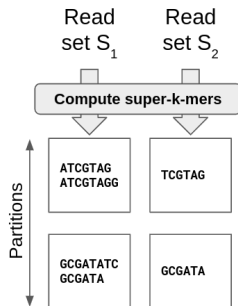
- super-k-mers: merged overlapping k-mers that share the same minimizer.

2 minimizers, 2 super-k-mers with $k = 6$ **and** $m = 3$

CTCATCGAC, TGCACGTTC

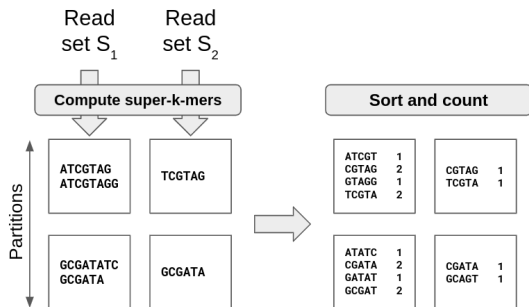
How it works: Counting (GATB-DSK)

- Step 1: Compute super-k-mers from each dataset and store them into partitions according their minimizers.



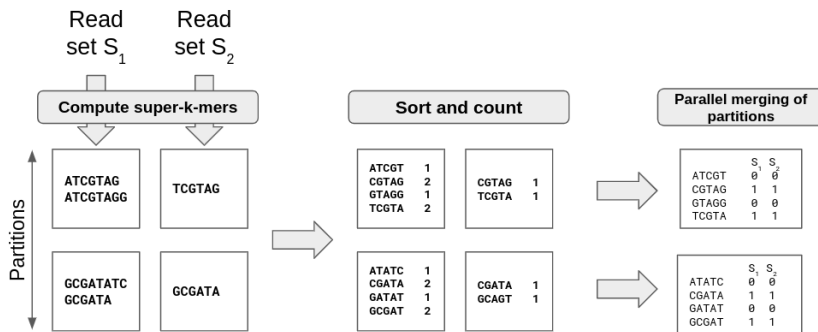
How it works: Counting (GATB-DSK)

- Step 2: In each partition, split super-k-mers into k-mers and sort them. The count is given by identical consecutive k-mers.



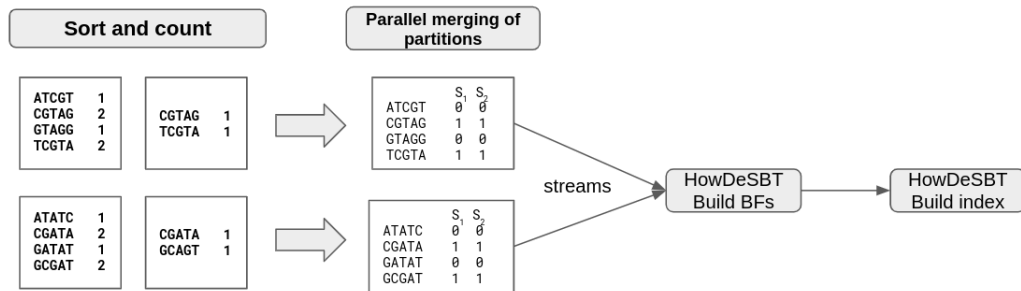
How it works: Counting (GATB-DSK)

- Step 3: Merge equivalent partitions between datasets to obtain sub-matrices.



Simka-HowDeSBT: Current implementation

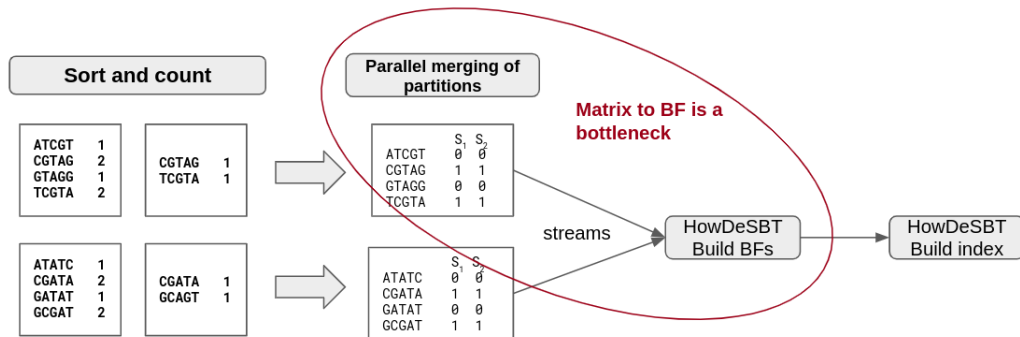
- Merge between datasets and stream each lines to build Bloom filters with HowDeSBT.



- <https://github.com/tleman/Simka-HowDeSBT>

Simka-HowDeSBT: Current implementation

- Merge partitions and stream each lines to build Bloom filters with HowDeSBT.

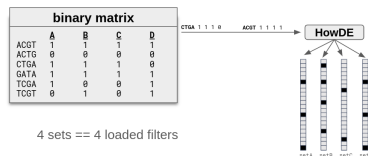


- <https://github.com/tlemane/Simka-HowDeSBT>

Matrix to Bloom filters

- ▶ Currently two possibilities:
 - ▶ Stream each line to HowDeSBT
 - ▶ Binary matrix on disk

In both cases: requires to have as many open filters in memory as there are data sets



Solutions ?

- ▶ Store matrix on disk:
 - ▶ Multiple passes
 - ▶ Transposition
- ▶ Build Bloom filters on files:
 - ▶ too many files open at the same time with large number of datasets

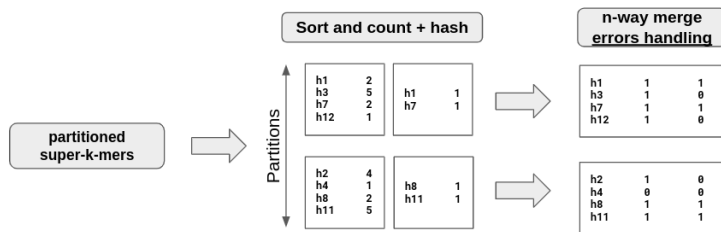
Can we do better?

- ▶ Currently, Simka gives textual k-mers counted to HowDeSBT which builds Bloom filters
- ▶ Modify early steps of Simka to build directly Bloom filters.
 - ▶ One possibility: **hash instead of k-mers**

Hash instead of k-mers (not yet implemented)

Hash instead of k-mers

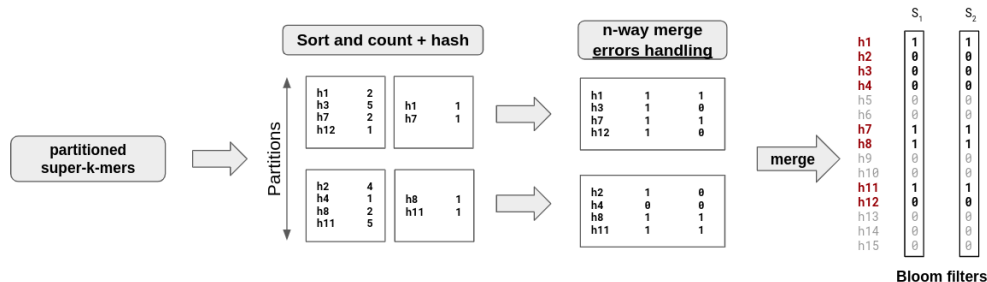
- hash k-mers at step 2 → obtain sorted hash values in partitions



Hash instead of k-mers (not yet implemented)

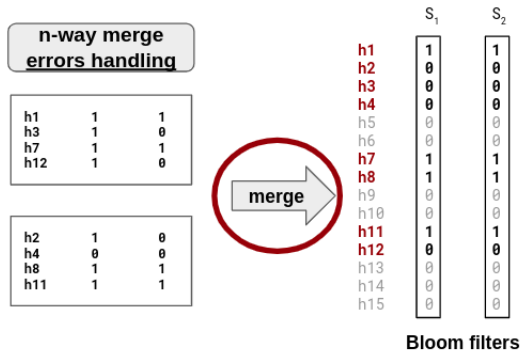
Merging sub-matrices

Set missing hash values to 0 → Bloom filters



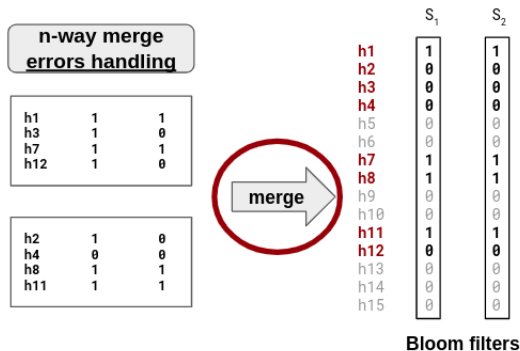
Hash instead of k-mers (not yet implemented)

- Obtain directly Bloom filters but with one more merge



Hash instead of k-mers (not yet implemented)

- Obtain directly Bloom filters but with one more merge

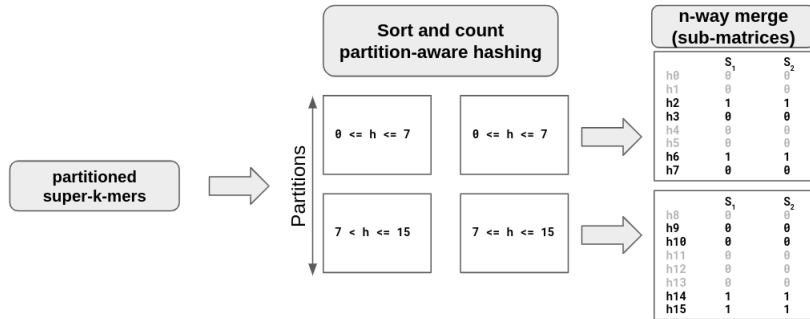


- Can we avoid this second merge?

Partition-aware hashing

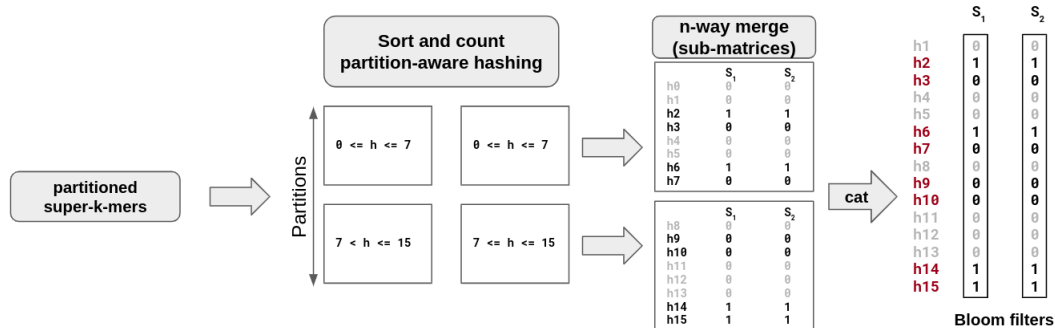
Define hash space according to the partition

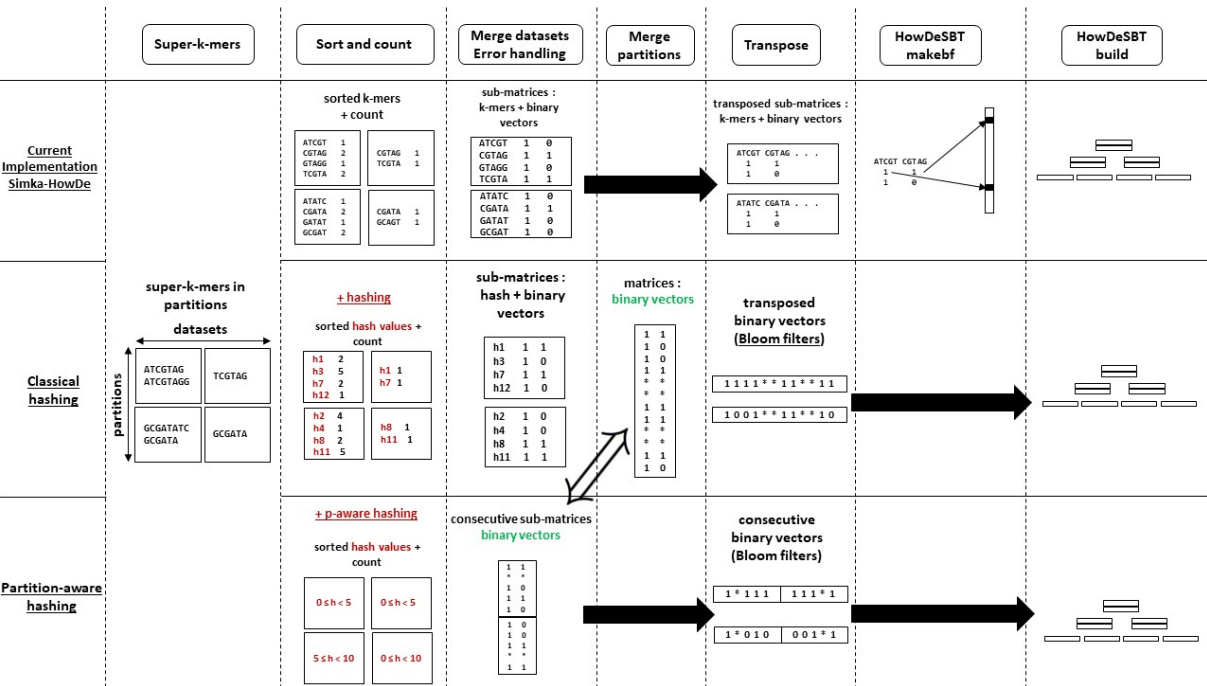
Requires to compute minimizers at query to select the right hash space

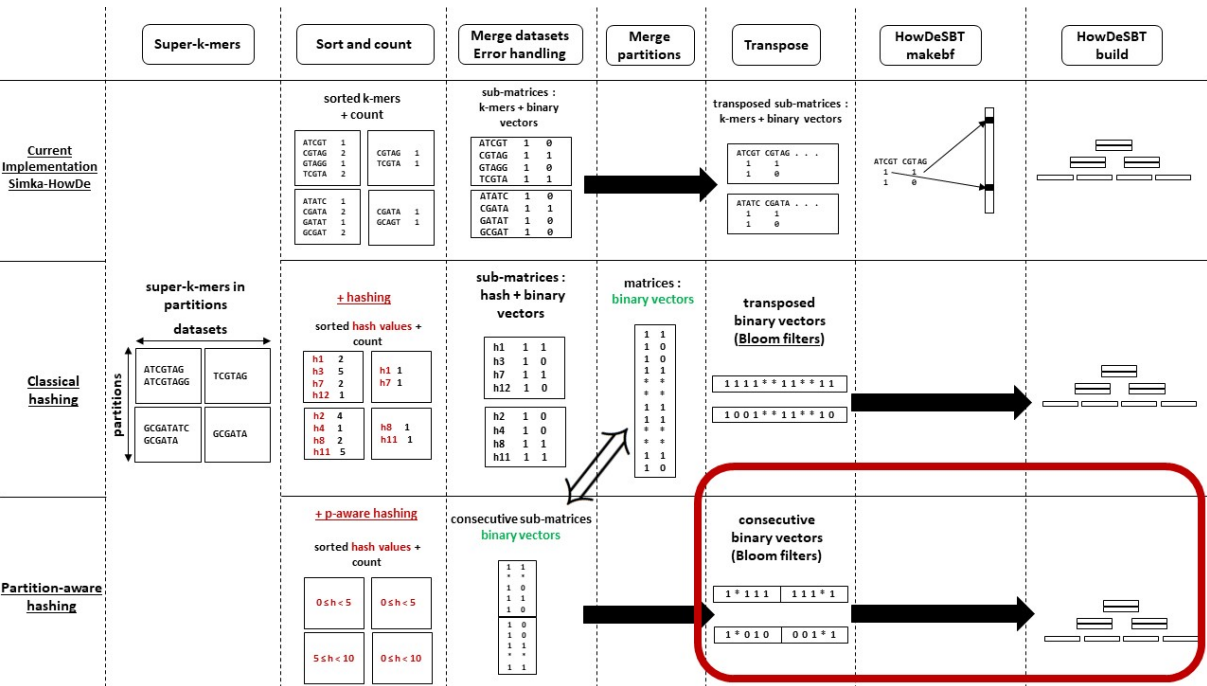


Partition-aware hashing

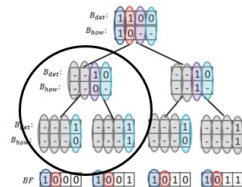
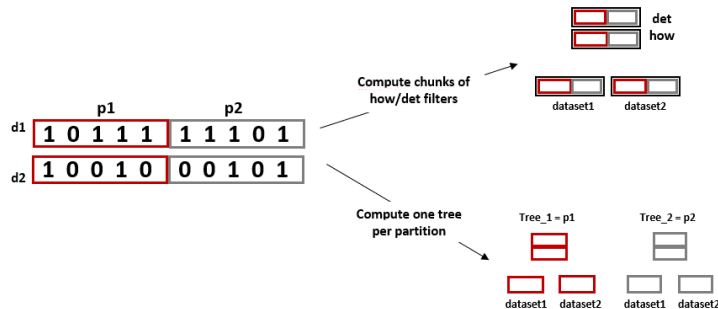
- Merging step becomes concatenation







Can the tree computation be parallelized?



Simka-HowDeSBT: What we plan to do?

- ▶ **Build one global tree vs one tree per partition?**
 - ▶ Are there any issues with building one tree per partition?
 - ▶ Storage?
 - ▶ Query efficiency?
 - ▶ In the case of tree per partition: classical hashing \Leftrightarrow partition-aware hashing

Simka-HowDeSBT: What we plan to do?

- ▶ **Build one global tree vs one tree per partition?**
 - ▶ Are there any issues with building one tree per partition?
 - ▶ Storage?
 - ▶ Query efficiency?
 - ▶ In the case of tree per partition: classical hashing \Leftrightarrow partition-aware hashing
- ▶ **Build one global tree?**
 - ▶ Partition-aware seems to be better than classical hashing
 - ▶ How to define efficient hash spaces?
 - ▶ Consideration relative to the tree topology computation

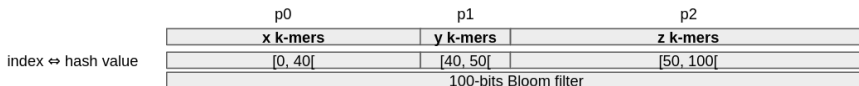
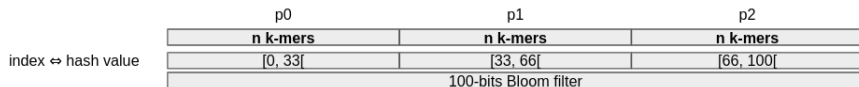
Simka-HowDeSBT: What we plan to do?

- ▶ **Build one global tree vs one tree per partition?**
 - ▶ Are there any issues with building one tree per partition?
 - ▶ Storage?
 - ▶ Query efficiency?
 - ▶ In the case of tree per partition: classical hashing \Leftrightarrow partition-aware hashing
- ▶ **Build one global tree?**
 - ▶ Partition-aware seems to be better than classical hashing
 - ▶ How to define efficient hash spaces?
 - ▶ Consideration relative to the tree topology computation
- ▶ **What about collisions?**

Simka-HowDeSBT: What we plan to do?

- ▶ **Build one global tree vs one tree per partition?**
 - ▶ Are there any issues with building one tree per partition?
 - ▶ Storage?
 - ▶ Query efficiency?
 - ▶ In the case of tree per partition: classical hashing \Leftrightarrow partition-aware hashing
- ▶ **Build one global tree?**
 - ▶ Partition-aware seems to be better than classical hashing
 - ▶ How to define efficient hash spaces?
 - ▶ Consideration relative to the tree topology computation
- ▶ **What about collisions?**
- ▶ **Implementation**

S1



S2

Maybe it corresponds to:
(3 different k-mers)

hash_x s_1 s_2 s_n
1 1 1 1

but we can't know that

		s_1	s_2	s_n
kmer1	hash_x	1	0	0
kmer2	hash_x	0	1	0
kmer3	hash_x	0	0	1

Solution: using oversized hash function

kmer1	hash_x + hash_a	s_1	s_2	s_n
kmer2	hash_x + hash_b	0	1	0
kmer3	hash_x + hash_c	0	0	1

hash_x s_1 s_2 s_n
0 0 0 0