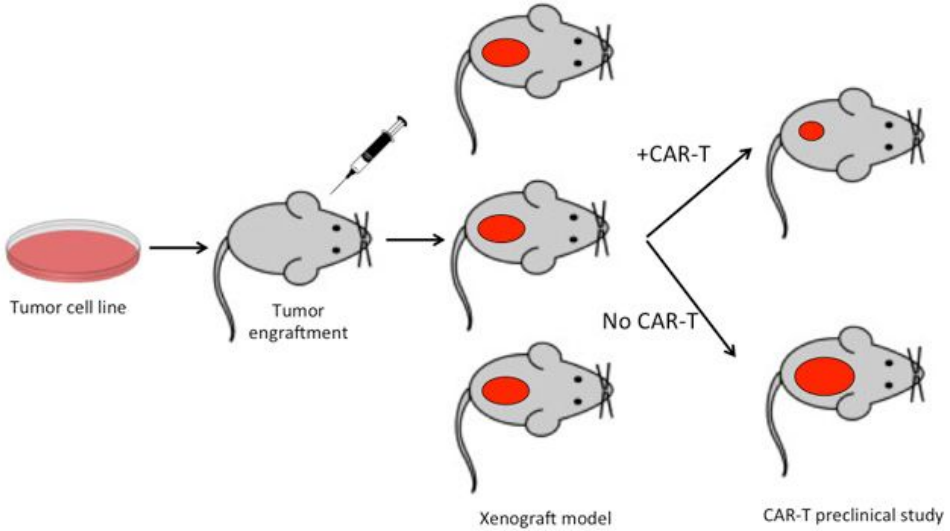
Fast lightweight accurate xenograft sorting

(Faster xenograft sorting with 3-way bucketed Cuckoo hashing of *k*-mers)

Sven Rahmann & Jens Zentgraf
Genome Informatics, Institute of Human Genetics
University of Duisburg-Essen, Essen, Germany

DSB 2020, Rennes, 04.02.2020

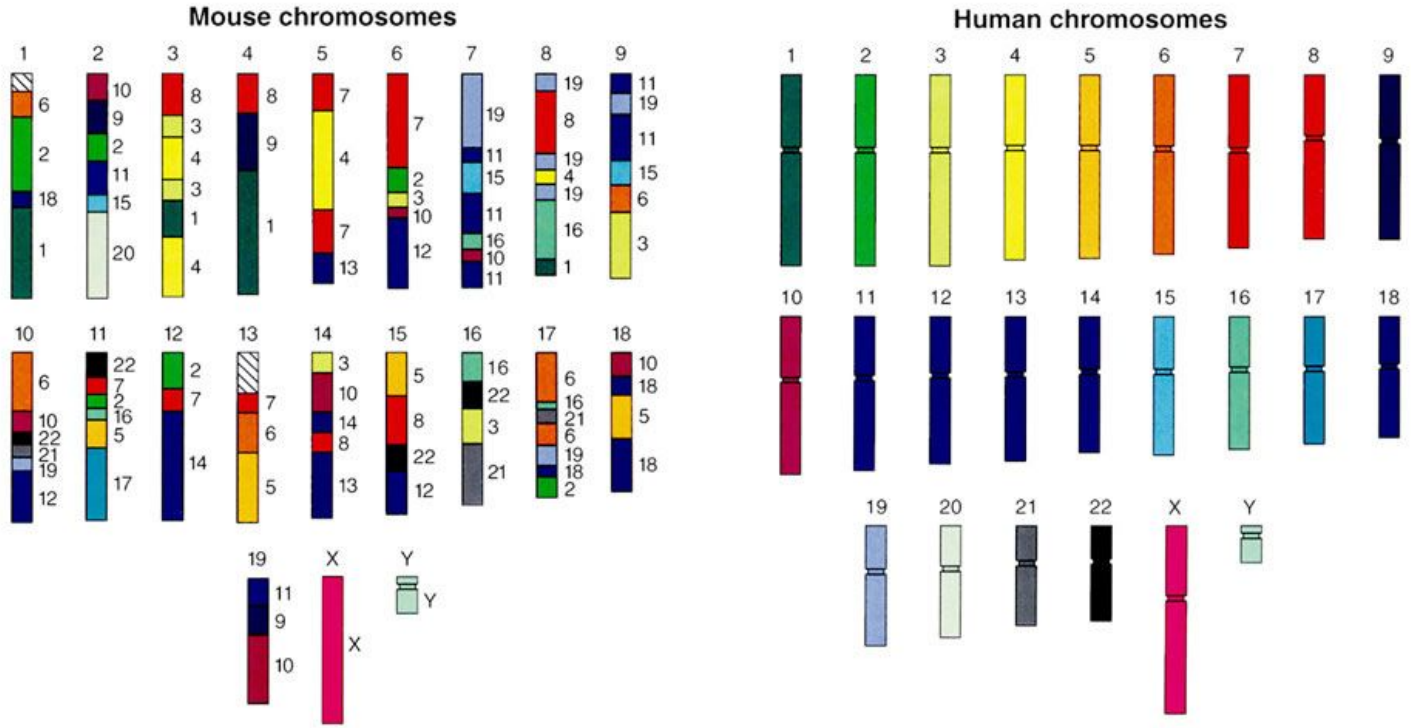
Patient-derived xenografts (PDXs)



- tumor cell lines
  or patient tumor samples
  implanted in mice
- study tumor heterogeneity,
  evolution
- sequencing of samples
- mixture of human+mouse DNA
- First task: separate/sort reads
  ("xenograft sorting"), or:
  extract graft (human) reads

Source: Creative AniModel,
https://www.creative-animodel.com/Featured-Service/Human-Tumor-Xenograft-Model.html

Mouse and Human Genetic Similarities



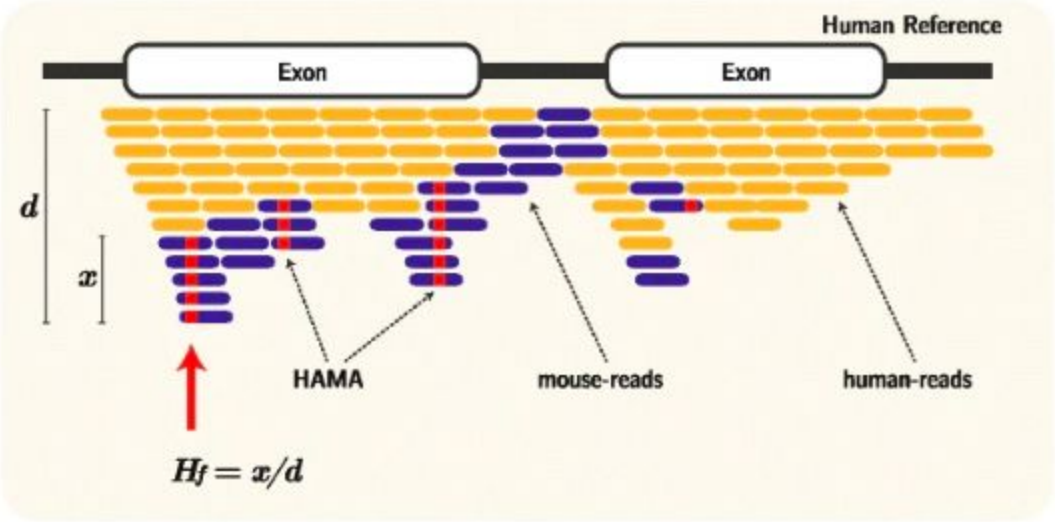
Source: https://public.ornl.gov/site/gallery/originals/Mouse_and_Human_Genetic_Similarities_-_original.jpg

YGA 98-075R2

Courtesy Lisa Stubbs
Oak Ridge National Laboratory

Problem: Human-Aligned Mouse Alleles (HAMAs)

- mouse reads may align to human genome
- may lead to false human (tumor) variant calls
- oncogenes particularly prone to this effect



S. Y. Jo, E. Kim, and S. Kim. Impact of mouse contamination in genomic profiling of patient-derived models and best practice for robust analysis. *Genome Biology*, 20(1):Article 231, Nov 2019.

Genome-scale xenograft sorting

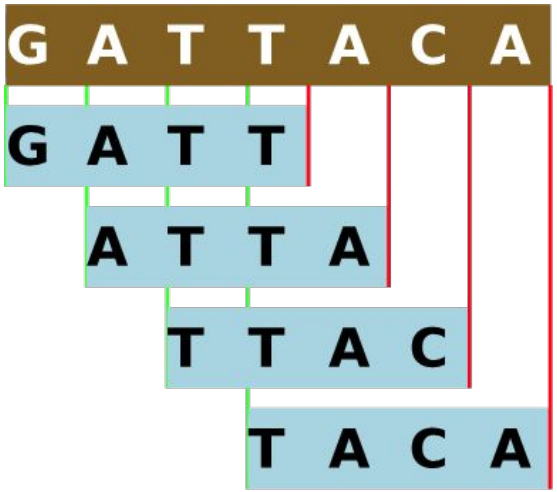
Classical approach	Alignment-free ("k-mer") approach
compute-intensive, slow	lightweight, fast
1. Map reads to reference genomes (read mappers: bwa-mem, bowtie2; based on "**FM-index**").	1. Partition reads into k-mers, look up species information for each k-mer, aggregate information per read to classify read.
2. Sort aligned reads (BAM files) by chromosome and position.	2. Perhaps: Use classical approach for difficult (ambiguous) reads.
3. Scan BAM files to find better match (species of origin) for each read	

Table 1. Tools for xenograft sorting and read filtering with key properties. See text for definition of operations; Lang.: programming language.

Tool	Input	Operations	Lang.
XenofilteR	aligned BAM	filter	R
Xenosplit	aligned BAM	filter, count	Python
Bamcmp	aligned BAM	partial sort	C++
Disambiguate	aligned BAM	partial sort	Python or C++
BBsplit	raw FASTQ	partial sort	Java
xenome	raw FASTQ	count, sort	C++
xengsort	raw FASTQ	count, sort	Python

k-mer methods for xenograft sorting

- Partition each read into its *k*-mers
- Look up information on each *k*-mer in a table [*k*-mer ↦ human | mouse | both]
- Absent k-mers occur in neither species.
- Aggregate *k*-mer information into a statement about the read [e.g., majority vote]

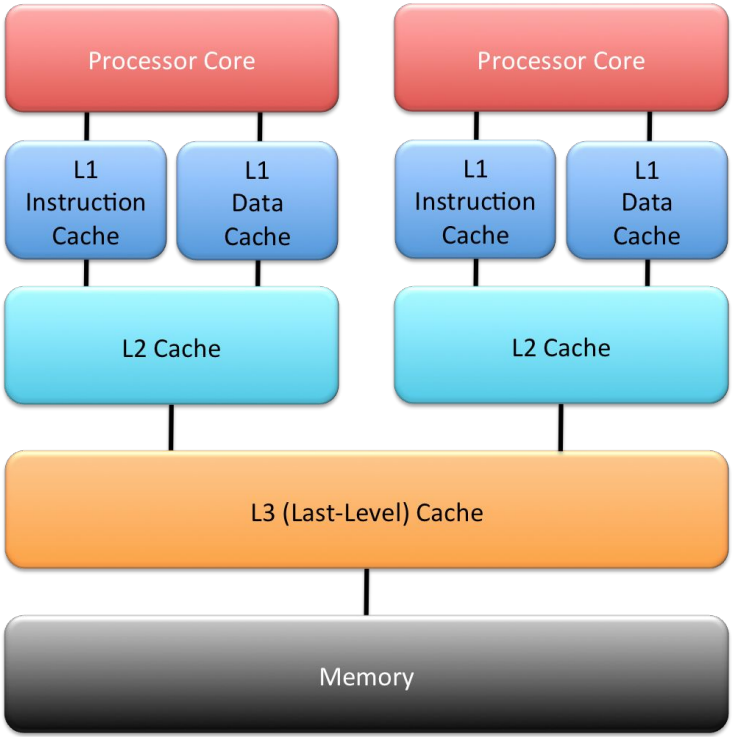


Goals: Be both fast and small

Time bottleneck
random memory lookup
(~400 times slower than arithmetic ops)

Therefore:
Try to achieve **a single** lookup,
avoid indirection !

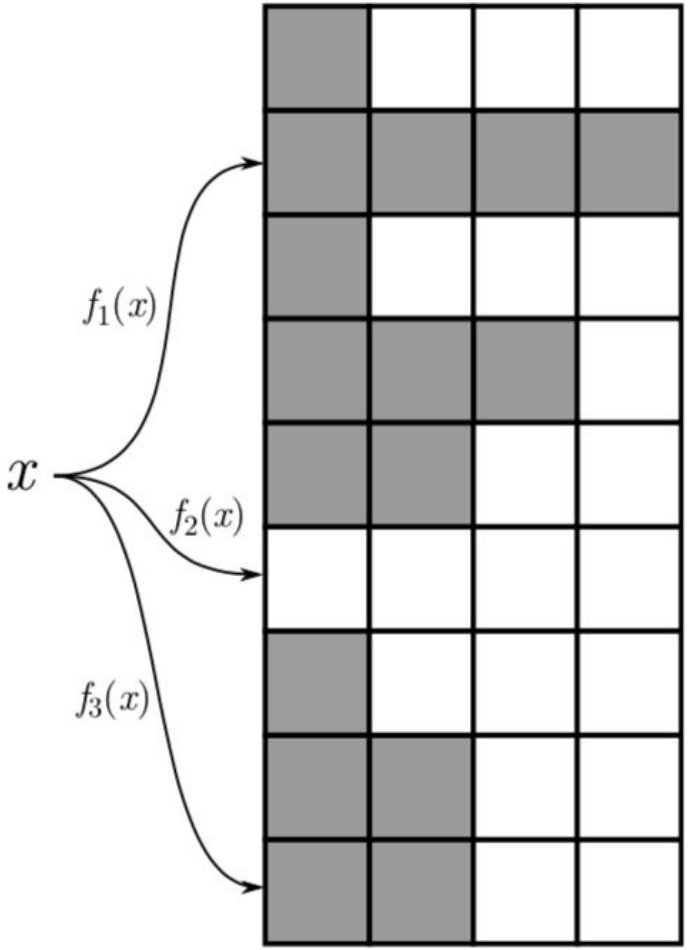
Space bottleneck
Bits for *k*-mers (50 for 25-mers)



(3,4) Cuckoo hashing

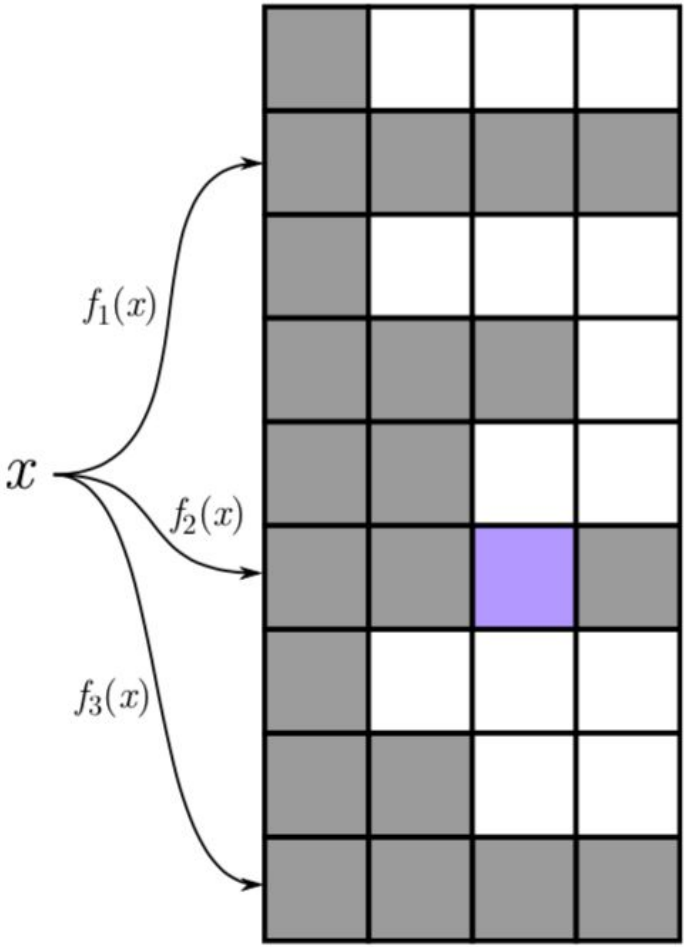
- We use 3 hash functions.
- Each maps a key (*k*-mer) to a bucket.
- Each bucket can store up to 4 elements.
- Idea: bucket fits within a cache line

- 12 possible locations for each element.
- At worst 3 memory lookups
  with cache misses.



Insertion by random walk

- Insert: try three buckets in order.
- Insert into first bucket with space available.
- If all full, evict a random element,
  place current element into now free slot.
- Re-insert evicted element
  into different slot.
- May cause another eviction…
- Random walk through table.
- Limit length of walk (e.g. 500 steps).
  Fail if limit reached.



Why (*h*,*b*) = (3,4) ?

More hash functions (*h*), larger buckets (*b*) have ⊕ and ⊖ effects:

⊕ higher load limit

[only 50% for standard (2,1)]
[over 99.9% for (3,4),
less w/ random walk]

⊖ more worst case cache misses (*h*)

⊖ more search effort per bucket (*b*)

- (3,4) is a good compromise (maybe also (2,8)).

2	3
0.5	0.9179352767
0.8970118682	0.9882014140
0.9591542686	0.9972857393
0.9803697743	0.9992531564

S. Walzer. Load thresholds for cuckoo hashing with overlapping blocks. ICALP 2018, LIPIcs 107:102.

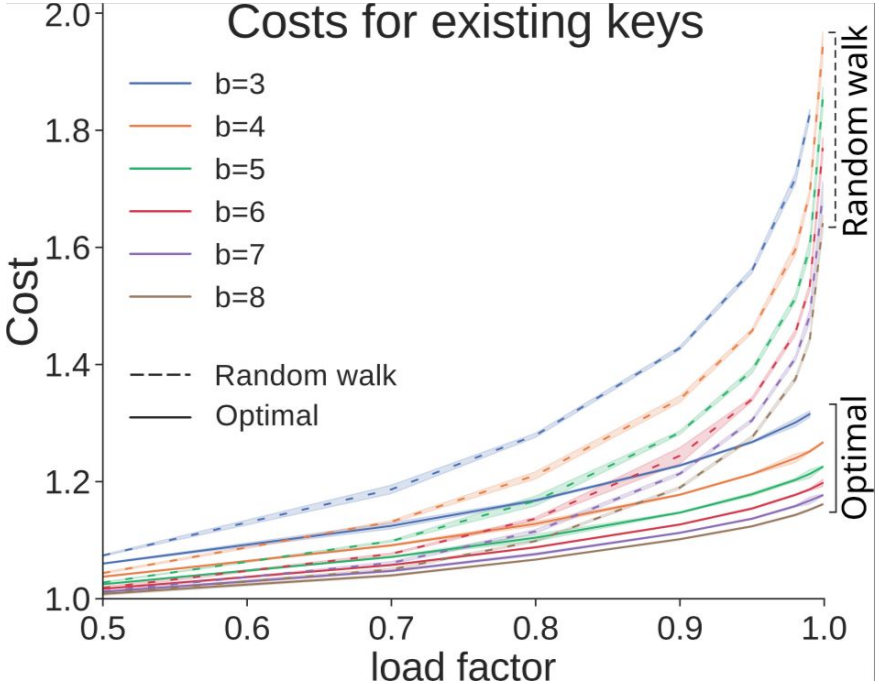
Speed vs. space: High vs. very high loads

So (*h*,*b*) = (3,4) allows loads up to 99.9%, but should we use it?

Leaving slots empty gives better choice distribution: more elements at their first hash bucket choice, lower average cost (cache misses).

Can be optimized exactly (Jens Zentgraf's talk; ALENEX 2020).

Random walk degrades near 100%. We use 88%, random walk performs fine.



Weak *k*-mers

Host or graft *k*-mers with a close neighbor (Hamming distance 1)
in the other species are not as reliable ("**weak**"):
A single nucleotide variation suffices to switch species.

After building the hash table, we mark weak *k*-mers.

Value set of size 5: host, weak host, graft, weak graft, both.
Each *k*-mer in the table has exactly one of these values.

[Xenome: similar concept with 4 values: host, graft, both, "**marginal**"]

Marking weak *k*-mers

Naive (and slow) method:
For each k-mer, query all 3k neighbors, adjust values accordingly.

Our method:
Create sorted list of k-mers and their reverse complements.
[Processed in 16 chunks according to first two letters].

Observation 1. *For $k = 2\ell + 1$, two k-mers x, y with Hamming distance 1 have their differing base either in their ℓ-suffix, in the ℓ-suffix of their reverse complement or in their middle base.*

Marking weak *k*-mers

k-mer: ℓ-prefix (`P`), 1 middle base (`M`), ℓ-suffix (`S`): 25 = 12 + 1 + 12.
Consider each block with common first 12 + 1 = 13 letters (`P + M`).

Compare all pairs of suffixes in such a block.
We catch all canonical k-mers with HD 1 unless they differ in the middle base.

```
PPPPPPPPPPPP|M|SSSSSSSSSSSS
...
SSSSSSSSSSSS|M|PPPPPPPPPPPP
```

For those, re-code *k*-mers in each block and re-sort block:

```
PPPPPPPPPPPP|SSSSSSSSSSSS|M
```

Space considerations

Keys: 25-mers (50 bits, 49 because canonical, but hard to encode)
Values: species (5 classes: host, graft, weak host, weak graft, both; 3 bits)

Store human and mouse reference, alternative alleles, cDNA transcripts:
~ 4.5 billion *k*-mers * 53 bits
at load 0.88:
33.88 GB for hash table 😩

- Graft (human) genome (1100 MB; 61 Gbp ↦ 885 MB; 3.15 Gbp): `ftp://ftp.ensembl.org/pub/release-98/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.toplevel.fa.gz`
- Graft (human) transcriptome (67 MB; 0.37 Gbp): `ftp://ftp.ensembl.org/pub/release-98/fasta/homo_sapiens/cdna/Homo_sapiens.GRCh38.cdna.all.fa.gz`
- Host (mouse) genome (804 MB; 12 Gbp → 776 MB; 2.72 Gbp): `ftp://ftp.ensembl.org/pub/release-98/fasta/mus_musculus/dna/Mus_musculus.GRCm38.dna.toplevel.fa.gz`
- Host (mouse) transcriptome (50 MB; 0.25 Gbp): `ftp://ftp.ensembl.org/pub/release-98/fasta/mus_musculus/cdna/Mus_musculus.GRCm38.cdna.all.fa.gz`

Saving Space with Quotienting

Keys are encoded canonical k-mers (half of set $[4^k] := \{0, .., 4^k\text{-}1\}$).

Step 1: Bijective randomizing function $[4^k] \to [4^k]$ with a odd

$$g_{a,b}(x) := [a \cdot (\mathrm{rot}_k(x) \text{ xor } b)] \bmod 4^k$$

Step 2: Map to buckets (simply mod p: number of buckets). Define

$$f(x) := g_{a,b}(x) \bmod p \quad \text{and} \quad q(x) := g_{a,b}(x) \mathbin{//} p \ .$$

Then x can be uniquely reconstructed
from $f(x)$ ("hash value, "bucket number") and $q(x)$ ("fingerprint", "quotient").
Sufficient to store $q(x)$ in bucket $f(x)$ (and which hash function was chosen).

Saving Space with Quotienting

Keys: 4.5 billion 25-mers (50 bits) at load 0.88 in 1 278 409 091 buckets (size 4)
Quotients: 19.75 → 20 bits
Hash choices: 4 (empty, 1, 2, 3): 2 bits
Values: 5 classes (host, graft, weak host, weak graft, both): 3 bits

~ 4.5 billion key value pairs * 25 bits / load 0.88:
15.98 GB for the hash table 😀

Values and/or choices could be further compressed,
saving a little more space (at the expense of CPU time).

Properties of the *k*-mer-species stores

k-mers	$k = 23$	(%)	$k = 25$	(%)	$k = 27$	(%)
total	4 396 323 491	(100)	4 496 607 845	(100)	4 576 953 994	(100)
host	1 924 087 512	(43.8)	2 050 845 757	(45.6)	2 105 520 461	(46.0)
graft	2 173 923 063	(49.4)	2 323 880 612	(51.7)	2 395 147 724	(52.3)
both	18 701 862	(0.4)	12 579 160	(0.3)	9 627 252	(0.2)
wk host	132 469 231	(3.0)	52 063 110	(1.2)	32 445 717	(0.7)
wk graft	147 141 823	(3.4)	57 239 206	(1.3)	34 212 840	(0.7)

Properties of the *k*-mer-species stores

tool	*k*	build CPU	build wall	mark CPU	mark wall	total CPU	total wall	mem final	size peak
xengsort	23	50	50	591	176	641	226	12.8	17.3
xengsort	25	53	53	437	158	490	211	15.9	20.4
xengsort	27	51	51	495	214	546	265	17.3	21.8
xenome	25	992	151	2338	356	3626	552	31.2	57.1
XenofilteR	–	528	658	–	–	528	658	13.0	22.0

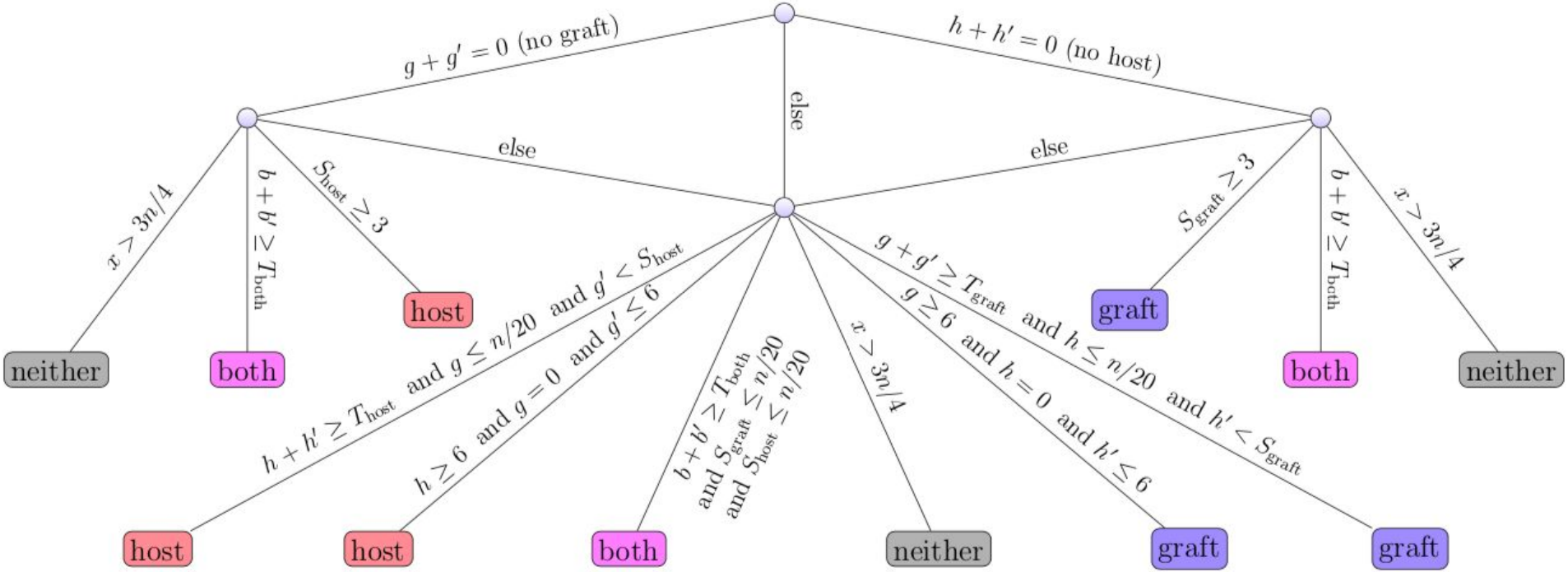
xengsort: 1 thread for build, 8 for mark
xenome: 8 (9) threads for build and mark
XenofilteR: 8 threads (bwa index)

Read classification using (*h*, *h*', *g*, *g*', *b*, *x*; *n*)

- Partition read into its valid k-mers (no Ns)
  (number: *n*)

- Look up class of each *k*-mer and count:
  - *h*, *h*': *k*-mers in read belonging to "host", "weak host"
  - *g*, *g*': *k*-mers in read belonging to "graft", "weak graft"
  - *b*: *k*-mers in read belonging to both species
  - *x*: *k*-mers in read belonging to neither species

Read classification using (*h*, *h*', *g*, *g*', *b*, *x*; *n*)



Quick mode

(inspired by a similar shortcut in kallisto)

- Examine 3rd and 3rd-last *k*-mer in read and look up classes.
- If classes agree, classify read accordingly.
- Otherwise, count all k-mers and use decision rule tree.

Results: Mouse exome, captured with human kit

BALB/c	*xengsort*		*xenome*		XfR	
time	68 Cm	15 Wm	392 Cm	45 Wm	61 Cm	61 Wm
mouse	62 235 960	(98.99)	61 274 277	(97.46)		
both	118 541	(0.19)	68 949	(0.11)		
human	342 908	(0.55)	348 154	(0.55)	285 556	(0.45)
ambgs.	45 063	(0.07)	1 098 036	(1.65)		
neither	127 035	(0.20)	80 091	(0.13)		

Sequences from Kim et al.: "Impact of mouse contamination in genomic profiling of patient-derived models and best practice for robust analysis." Genome Biology, 20(1):231 (Nov 2019).

- xengsort: ⅛ of CPU work, ⅓ of the time of xenome.
- XenofilteR only compares already aligned sequences, same CPU work
- xenome often reports "ambiguous" when xengsort does not.

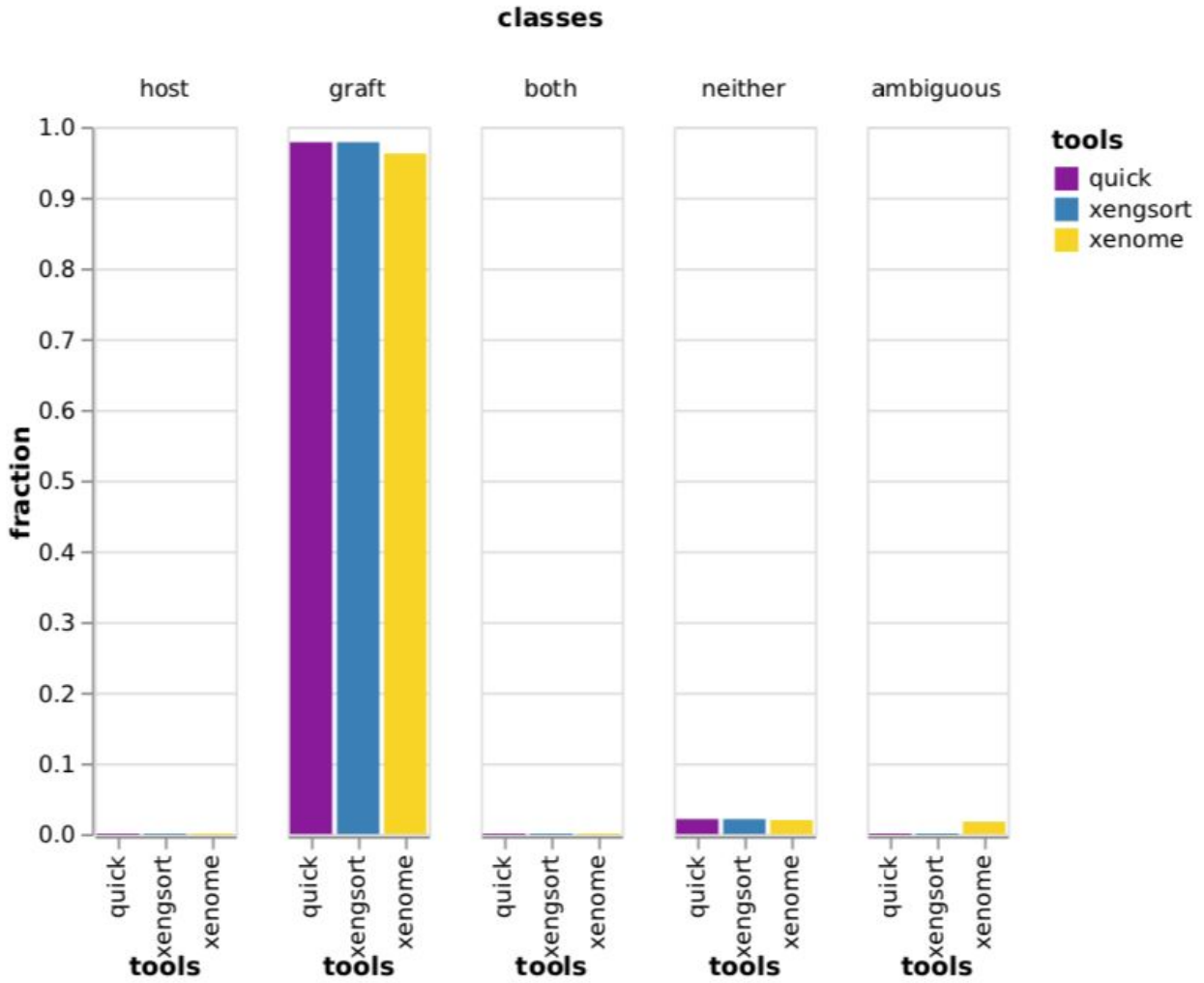
Human

GIAB human matepair dataset (Ashkenazim trio; 1258 million read pairs).

Almost all graft (correct).

Quick mode gives almost identical results.

Xenome sometimes bails out ("ambiguous").



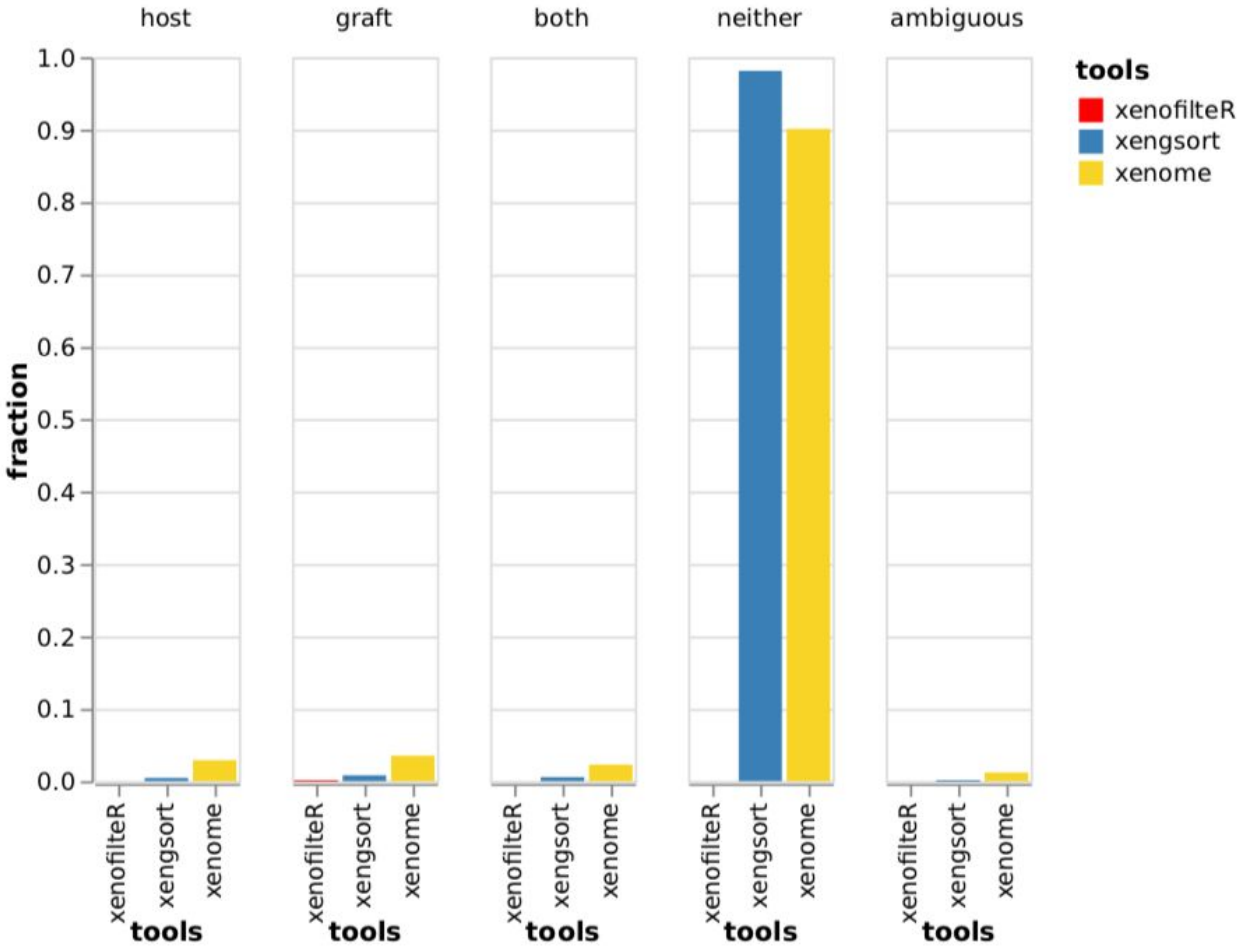
Chicken

A sequenced chicken genome.

XenofilteR only extracts graft (human) reads, remainder not classified. Finds none (correct).

xengsort: Almost all neither (correct).

xenome: 10% host, graft, and both (not ideal).

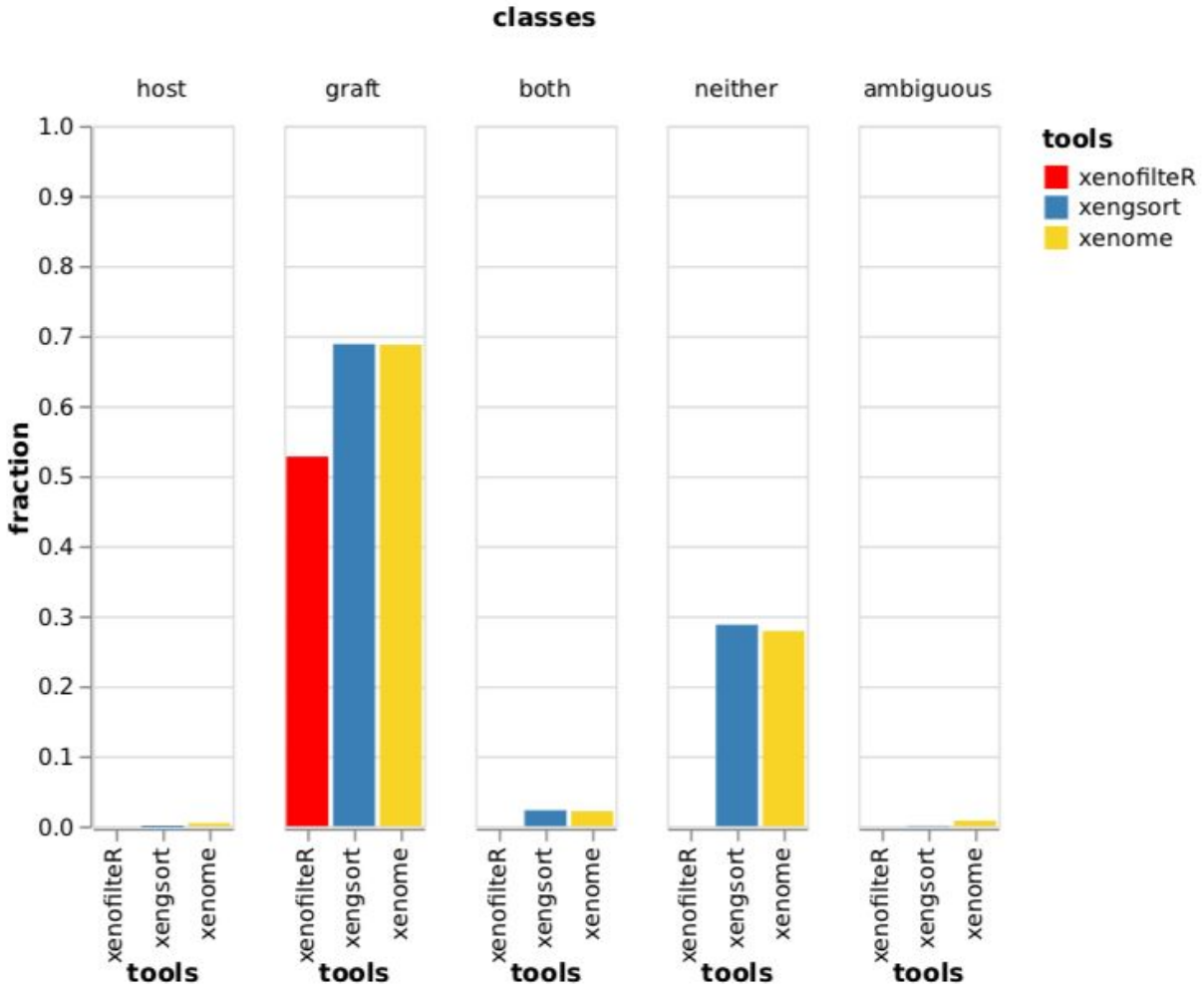


Human

Human lymphocytic leukemia RNA-seq (single-end).

XenofilteR finds less human reads than both k-mer based tools (only ~50 %; remainder?)

Still, ~30% "neither" ?
Technical library issues?

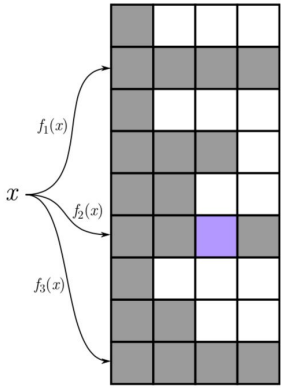


Summary: Fast lightweight xenograft sorting

- Xenograft sorting is necessary to avoid false variant calls.
- Alignment-free approach using 25-mers and decision rules works well, lightweight on CPU resources, using 3-way bucketed Cuckoo hashing
- Our implementation xengsort outperforms xenome;
  ⅛ of the CPU work, ⅓ wall clock time (both 8 threads)
- Same time just to scan the BAM files (XenofilteR)
- Quick mode on very large human data sets reduces time by ⅓,
  giving almost same results (needs further testing).
- 25-mer table fits in 16 GB RAM, could be made smaller
  (higher load, compacted values and choice indicators).

Our Data Structure in Bioinformatics 2020

- Hash table
- 3-way bucketed Cuckoo hashing
  (with bucket size 4)
- Keys reduced using quotienting
  (part of key stored in bucket number)
- Interesting trade offs:
  Small buckets = small quotients,
  but lower load possible,
  and fewer keys at first hash choice.
- Several further engineering opportunities



$$g_{a,b}(x) := [a \cdot (\mathrm{rot}_k(x) \text{ xor } b)] \bmod 4^k$$