Expediting MRSH-v2 Approximate Matching with Hierarchical Bloom Filter Trees ICDF2C October 10th 2017

DAVID LILLIS, FRANK BREITINGER AND MARK SCANLON

UCD Forensics and Security Research Group

Approximate Matching

- **Scenario**: Collection of “known illegal” files. Want to search for these on a seized device.
- Finding exact matches is easy (hashing).
- Approximate matching (a.k.a. “fuzzy hashing”) aims to find similar files on the byte level, e.g.
  - Files that have been extended/truncated.
  - Files within files.
  - Partial files.

MRSH-v2

- Initially proposed by Breitinger & Baier (2012).
- Generates a **similarity digest** for each file.
  - Consists of one or more Bloom Filters: probabilistic data structure that can say whether it probably contains an item, or definitely does not contain it.
  - These can be compared to calculate a similarity score.
- File divided into “chunks”: file read byte-by-byte and a rolling hash identifies the end of a chunk.
- Each chunk is hashed using FNV (a fast, noncryptographic hashing function).
- Hash used to set 5 bits of the Bloom Filter.
- When Bloom Filter is full, a new, empty Bloom Filter is added to the digest, and further inserts are in this.

Motivations

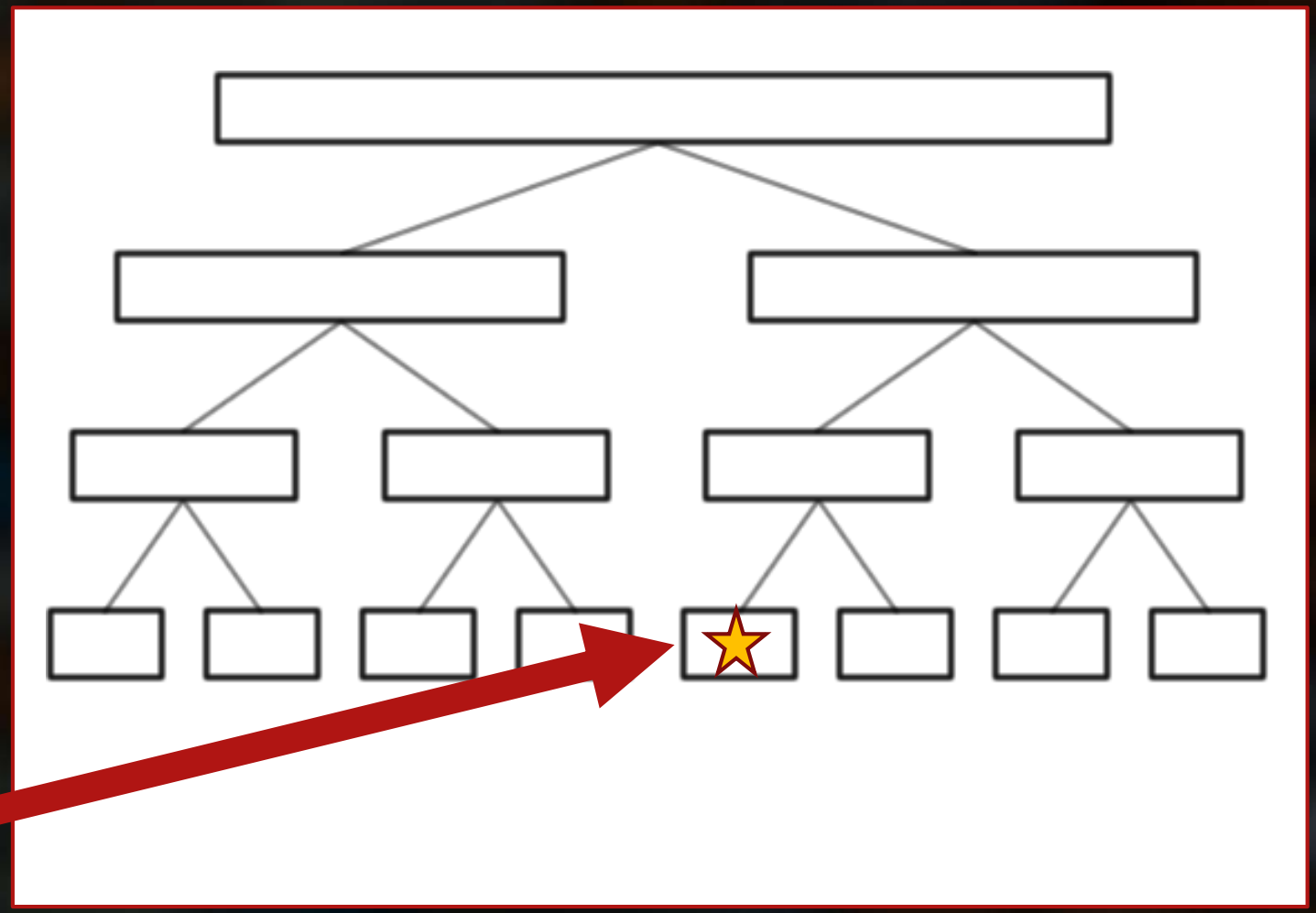
- **Problem:** Similarity score comes from a pairwise comparison of two similarity digests. Not scalable.
- Aim to explore alternative data structures that can achieve the same results in less time.
- Hierarchical Bloom Filter Trees initially proposed theoretically by Breitinger et al. (2014).
- This work gathers some empirical data on the performance of this approach.
  - i.e. Can we do the same thing, but faster?

Hierarchical Bloom Filter Trees (HBFTs): Building

- Binary tree of Bloom Filters.
- Each parent is twice the size of its children.
- Files allocated to leaf nodes: round robin.

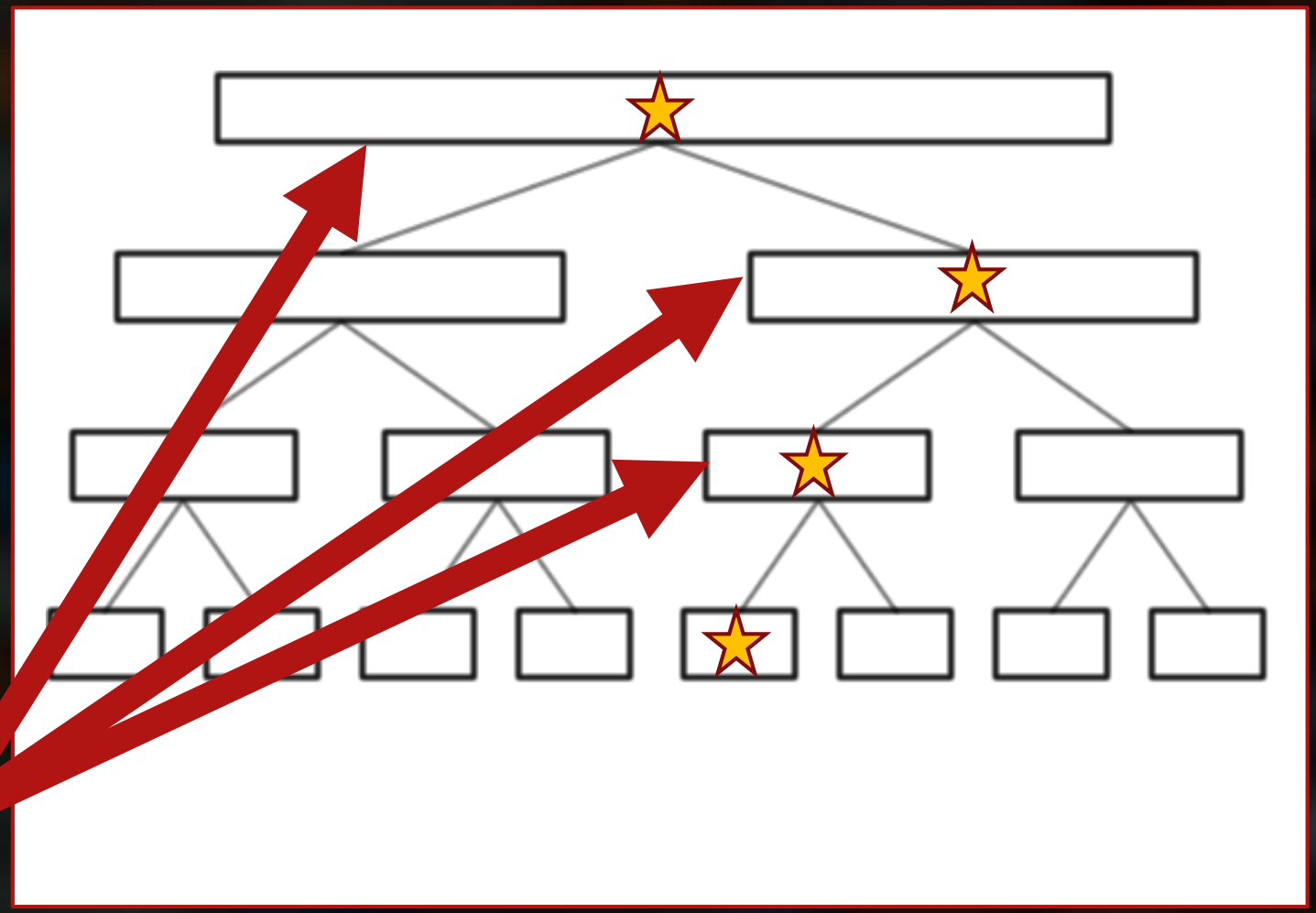
File is processed in the same way as for MRSH-v2.

When each chunk is hashed, this is used to set bits in the relevant leaf node.



Hierarchical Bloom Filter Trees (HBFTs): Building

- Binary tree of Bloom Filters.
- Each parent is twice the size of its children.
- Files allocated to leaf nodes: round robin.



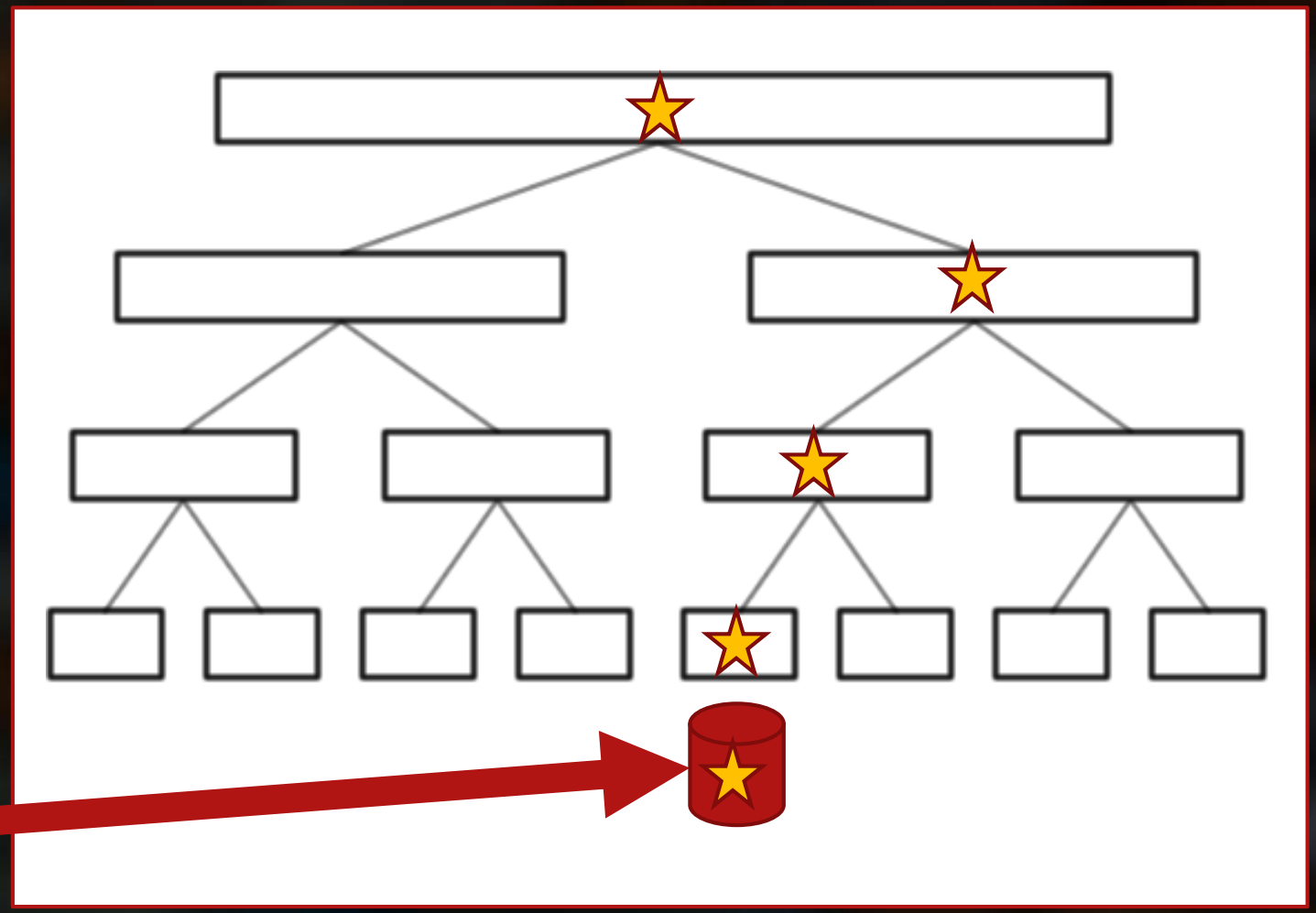
The same hash values are used to set bits in the parent node also.

A similar process is followed for all ancestor nodes.

Hierarchical Bloom Filter Trees (HBFTs): Building

- Binary tree of Bloom Filters.
- Each parent is twice the size of its children.
- Files allocated to leaf nodes: round robin.

The file's MRSH-v2 similarity digest is stored in association with the appropriate leaf node.

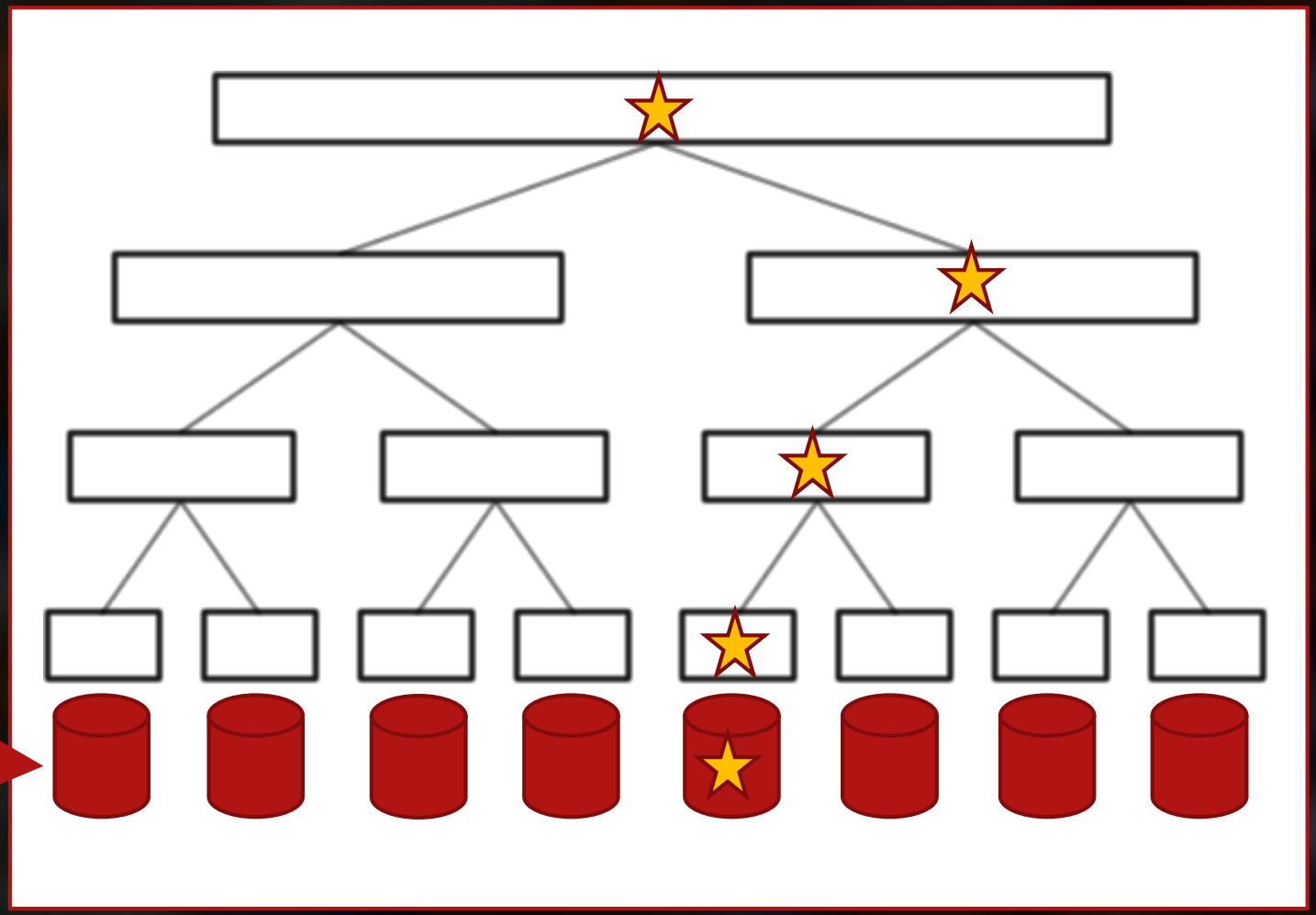


Hierarchical Bloom Filter Trees (HBFTs): Building

- Binary tree of Bloom Filters.
- Each parent is twice the size of its children.
- Files allocated to leaf nodes: round robin.

Every leaf node has a set of similarity digests associated with it.

Each represents1/L of the collection, where L is the number of leaf nodes



HBFTs: Searching

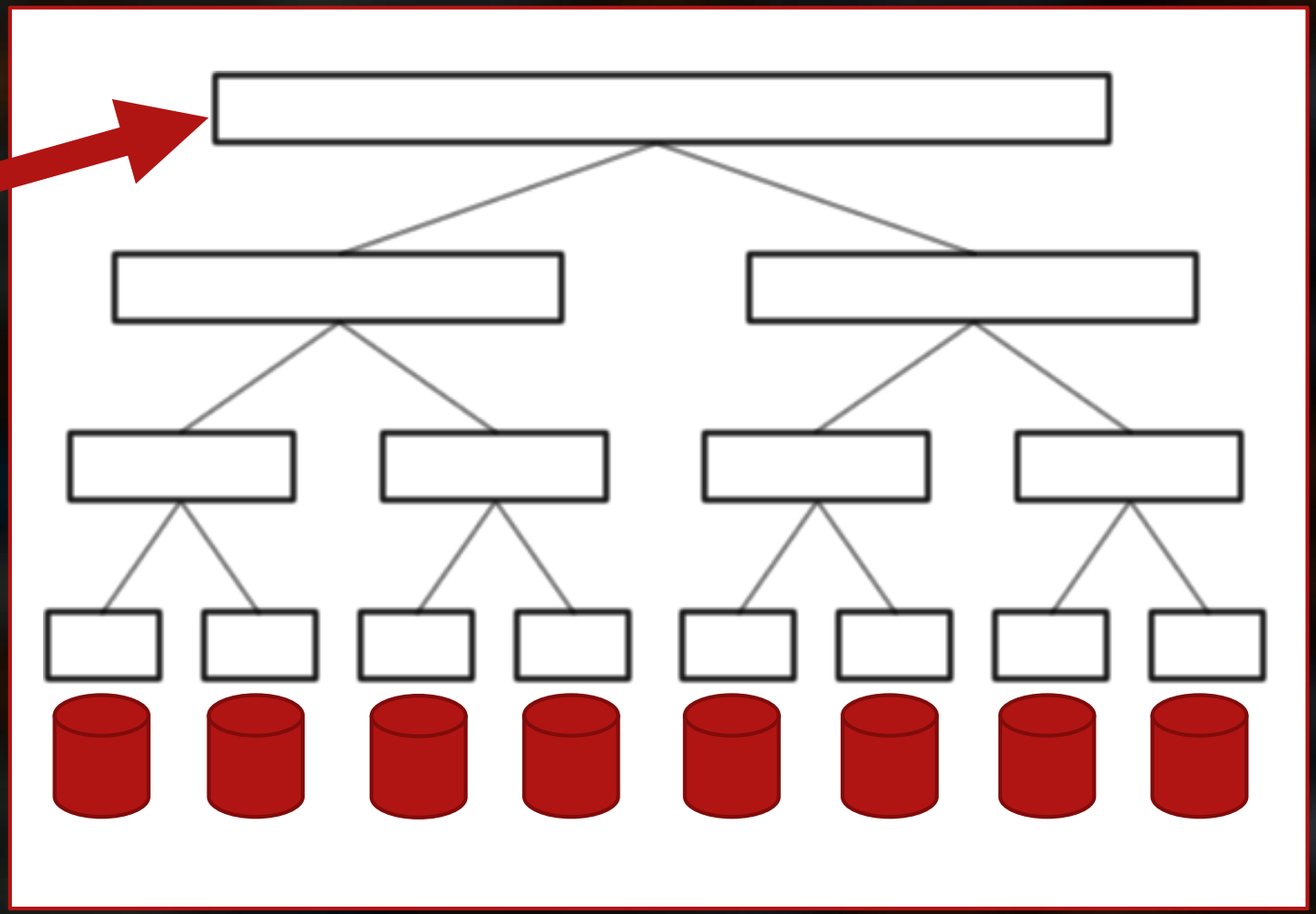
- To search for a file, it is also processed in a similar way.

Initially search at the root node.

For each hash of a file chunk, we check if it is contained in the root Bloom Filter.

If the number of consecutive matches exceeds a threshold, it is considered to be a successful match.

We call this threshold `min_run`.

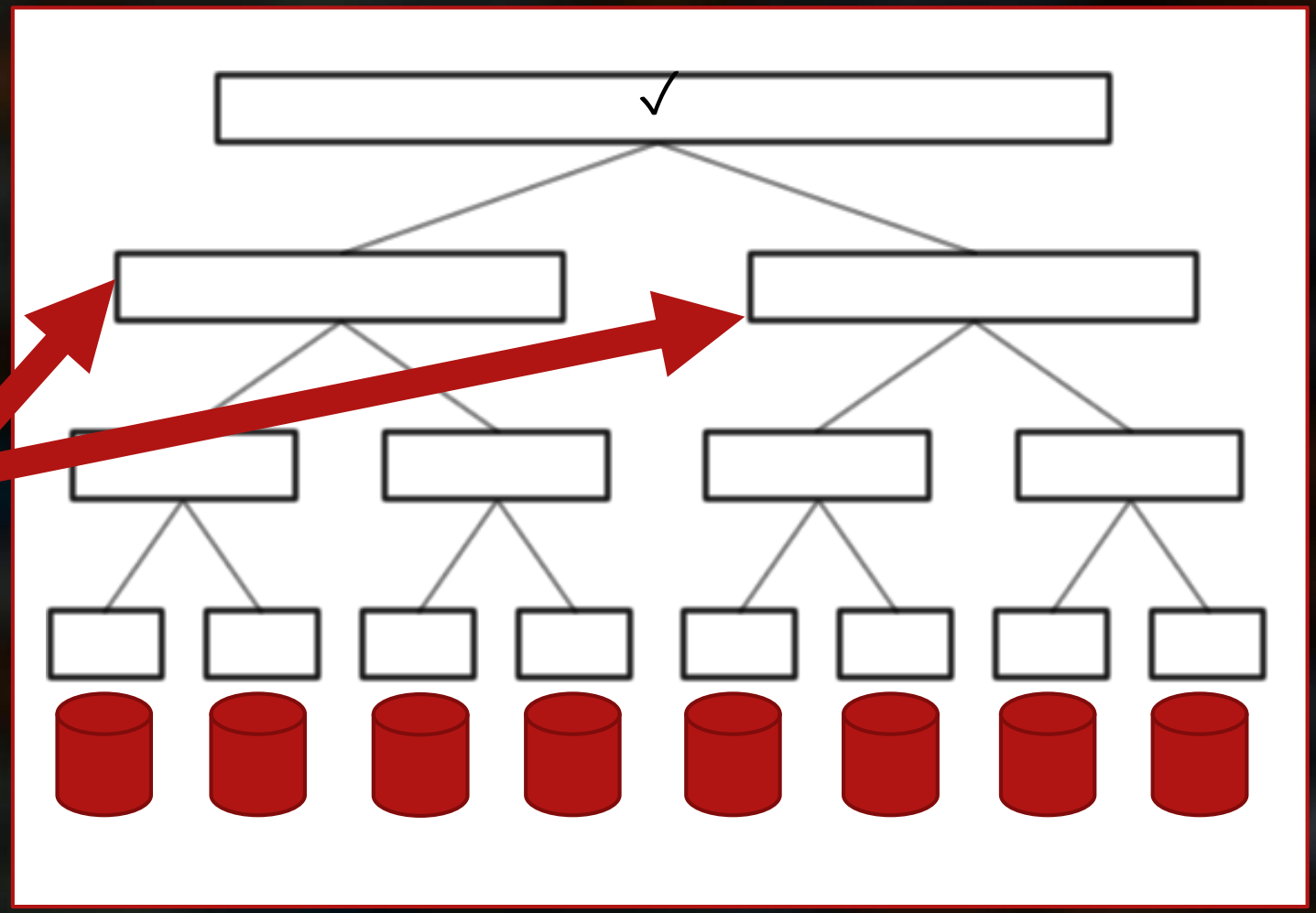


HBFTs: Searching

- To search for a file, it is also processed in a similar way.

If a match is found, the search continues at the next level.

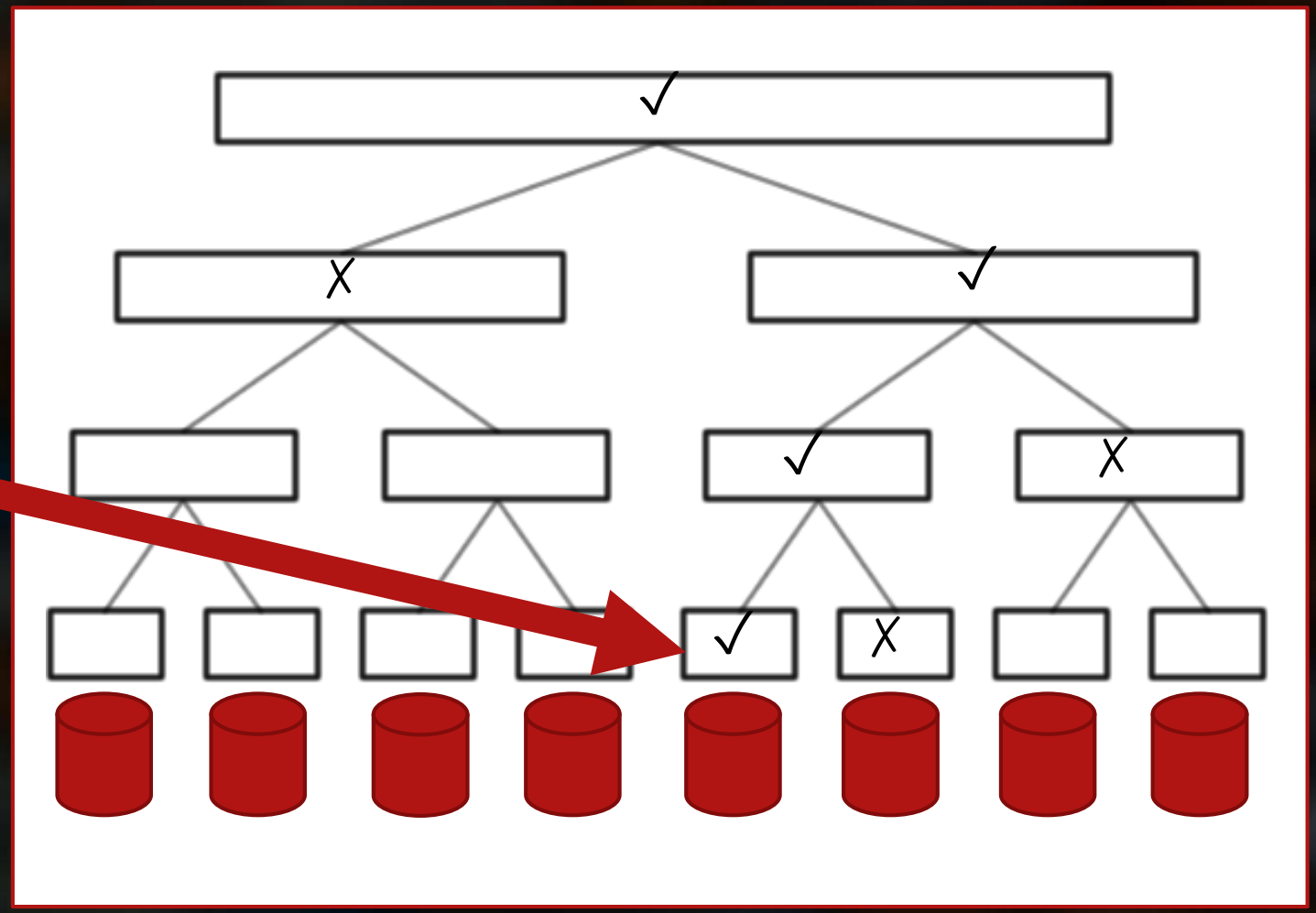
Both child nodes must be searched.



HBFTs: Searching

- To search for a file, it is also processed in a similar way.

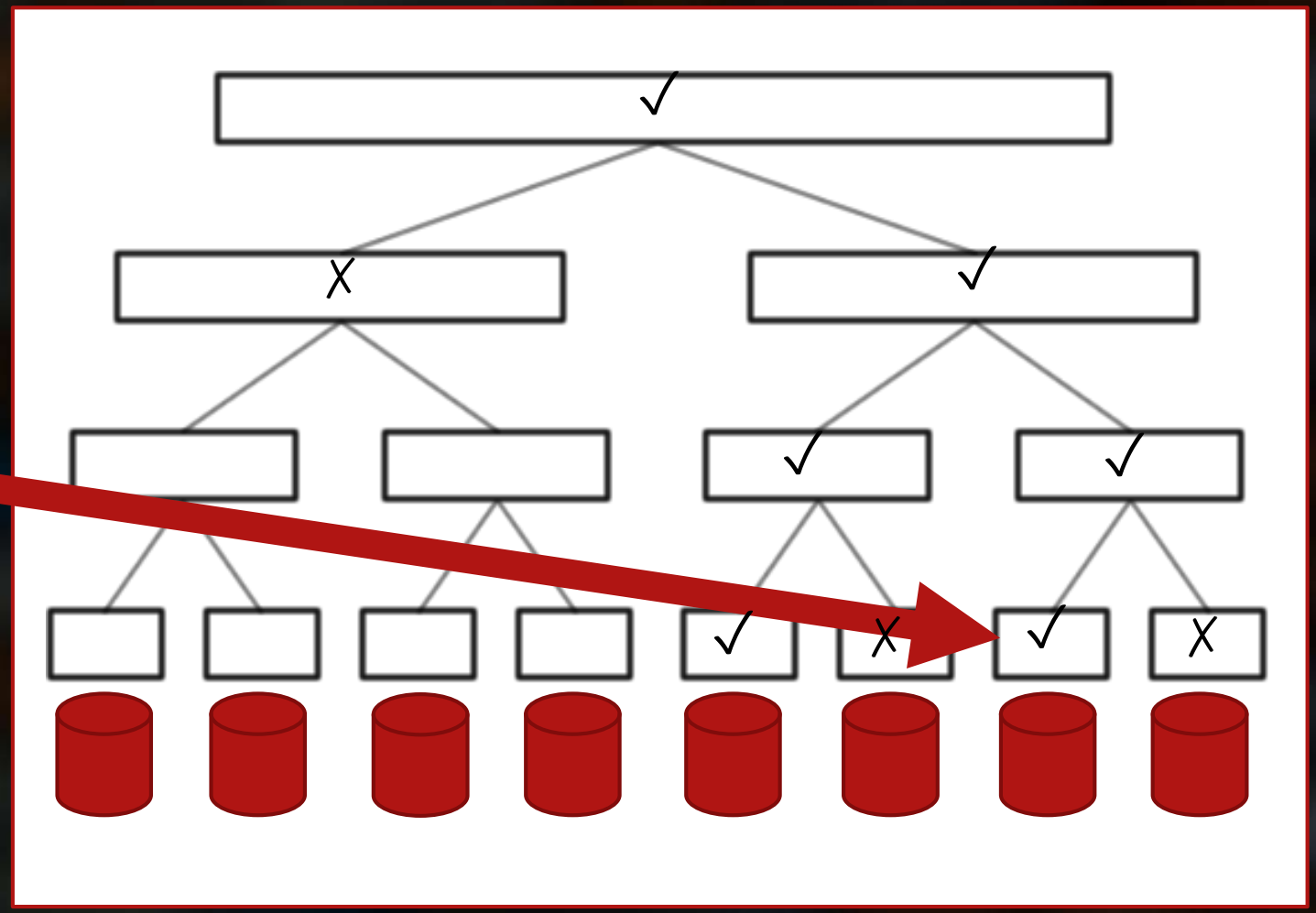
The search continues until one or more leaf nodes are reached.



HBFTs: Searching

- To search for a file, it is also processed in a similar way.

Bloom Filters can give false positive results, so it is possible for searches to reach leaves even where there are no similar results.



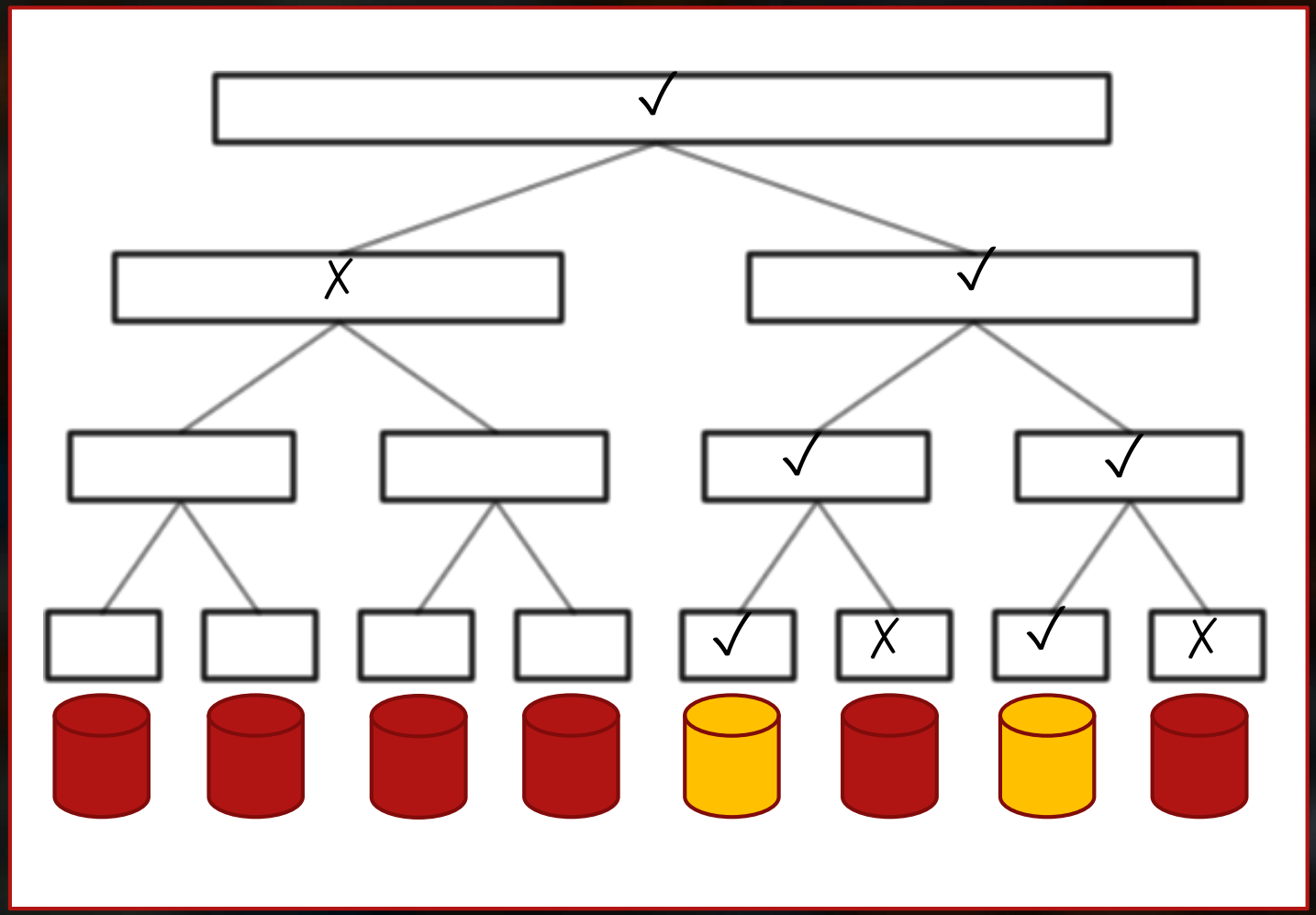
HBFTs: Searching

- To search for a file, it is also processed in a similar way.

To calculate the similarity scores, the existing MRSH-v2 algorithm is used to make pairwise comparisons.

A similarity digest is created for the file that we are searching for.

This must be compared with all the digest stored at any leaf that the search reaches.



HBFT: Some Questions

- How many nodes in the tree?
  - More nodes: fewer pairwise comparisons.
  - Fewer nodes: larger Bloom Filters (fewer false positives).
- What constitutes a positive match for a node in the tree?
  - i.e. what threshold should be used for `min_run`?
- When comparing two datasets, which should the tree represent?

Datasets

- **t5*:** 4,457 files (~1.8GiB)
  - Gathered from US government websites, often used for approximate matching.
  - Plain text, HTML, PDF, Images, MS Office documents.
- **win7:** 48,384 files excluding empty files and symlinks (~10GiB)
  - Fresh install of Windows 7.
  - Varied file types.

* Obtainable from http://roussev.net/t5

Experiment #1

- Datasets: Tree represents t5, search for t5.
- Goals:
  - Measure effectiveness for exact matching.
  - Identify appropriate value for `min_run` parameter.
  - Investigate relationship between size of tree and time to build & search tree.
  - Investigate relationship between size of tree and number of pairwise comparisons required to calculate similarity scores.

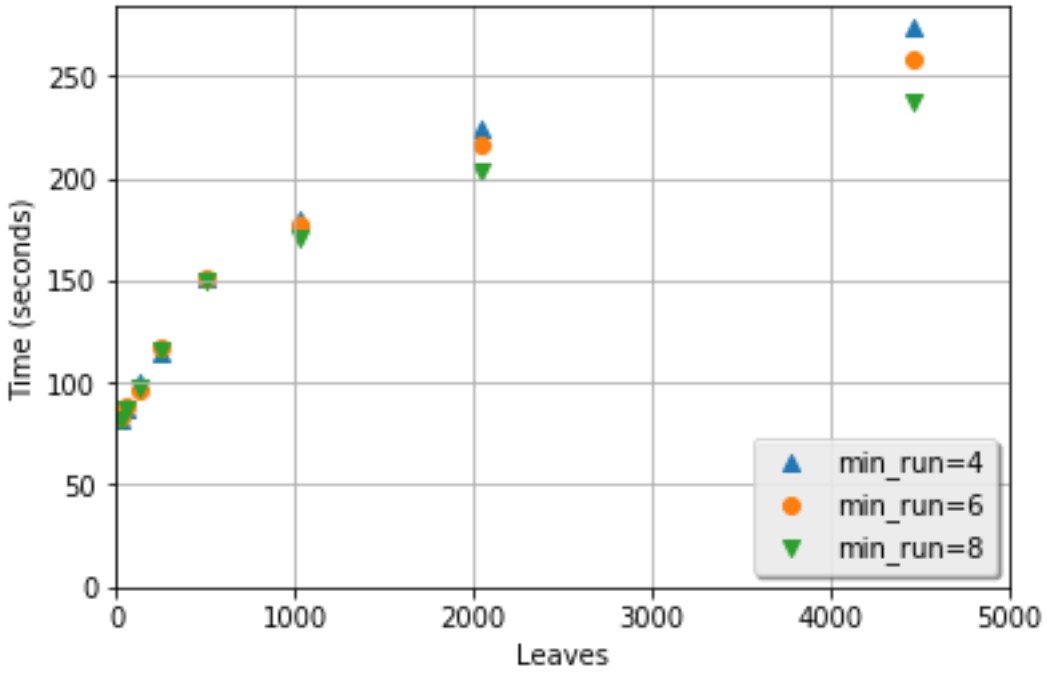
Experiment #1: Results

- Exact matching:

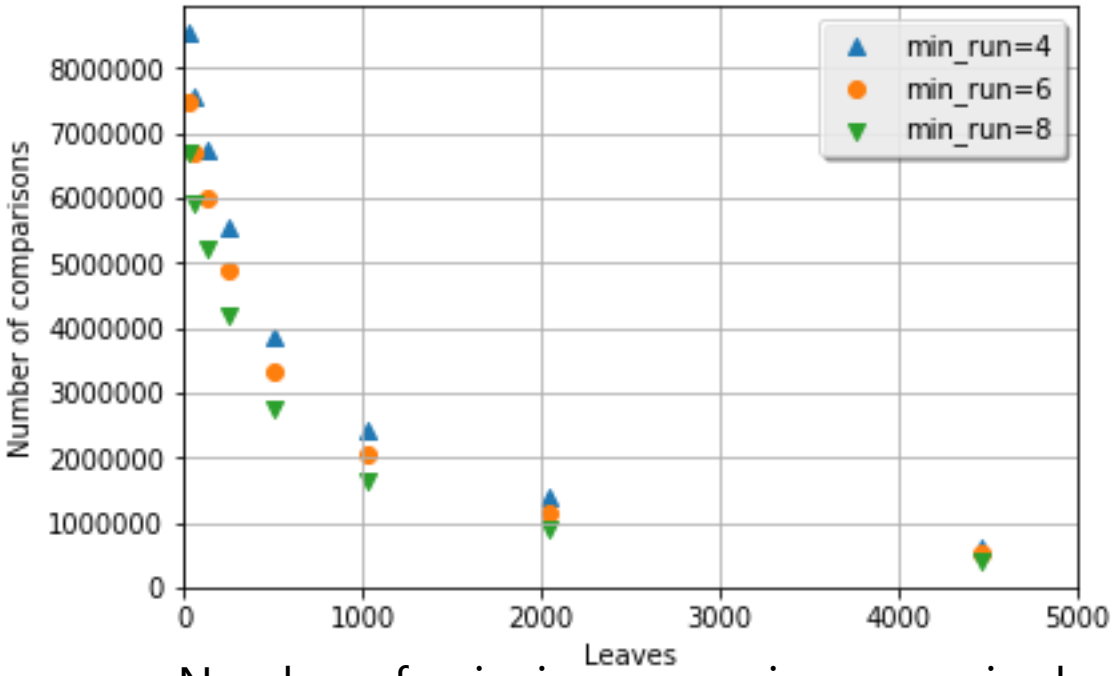
min_run	Recall
4	100%
6	99.98%
8	99.93%

- When min_run = 4, all identical files are found.
  - With higher values, some files are missed.

Experiment #1: Results



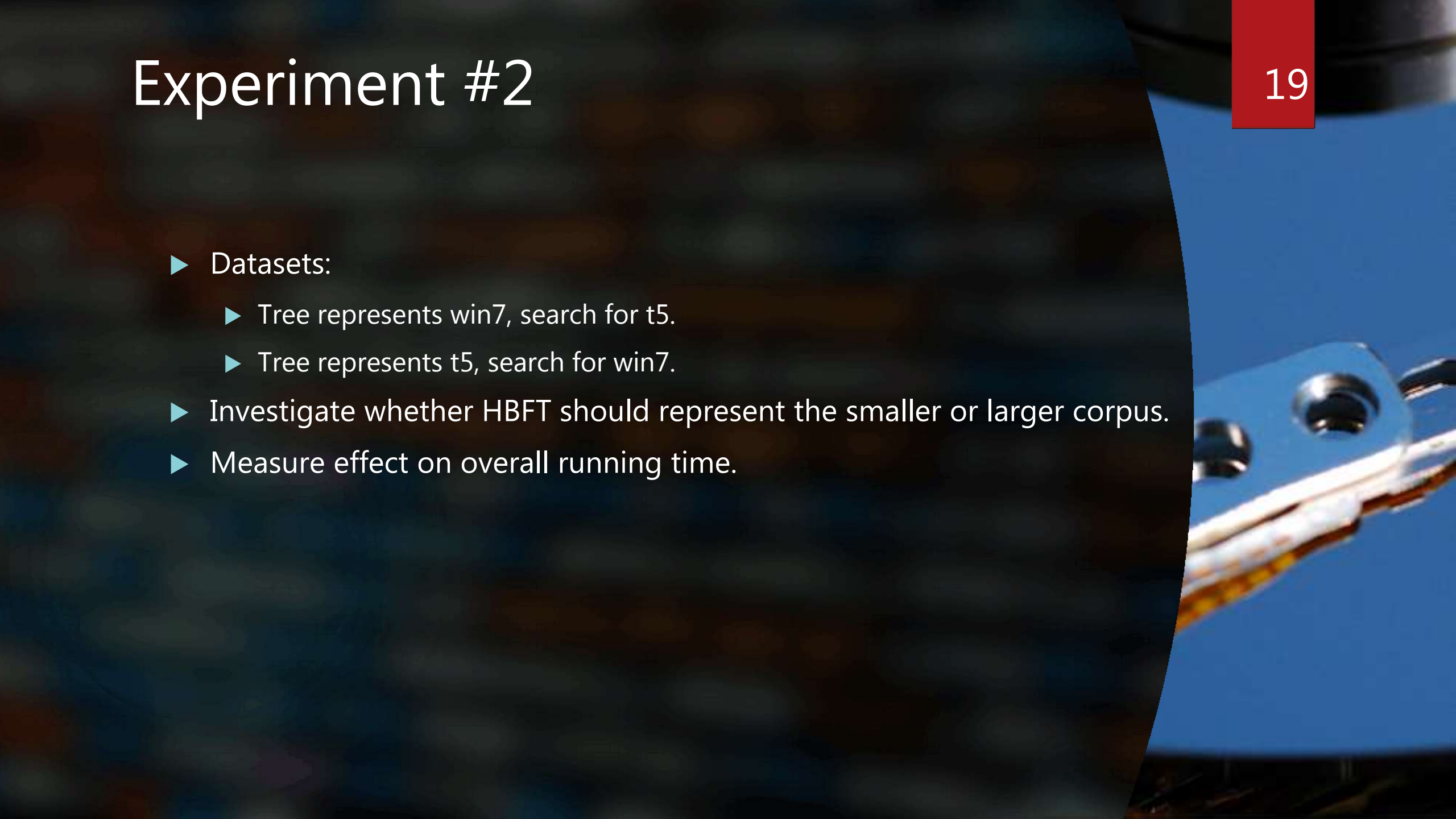
Time to build tree and search for all files.
(excluding pairwise comparisons)



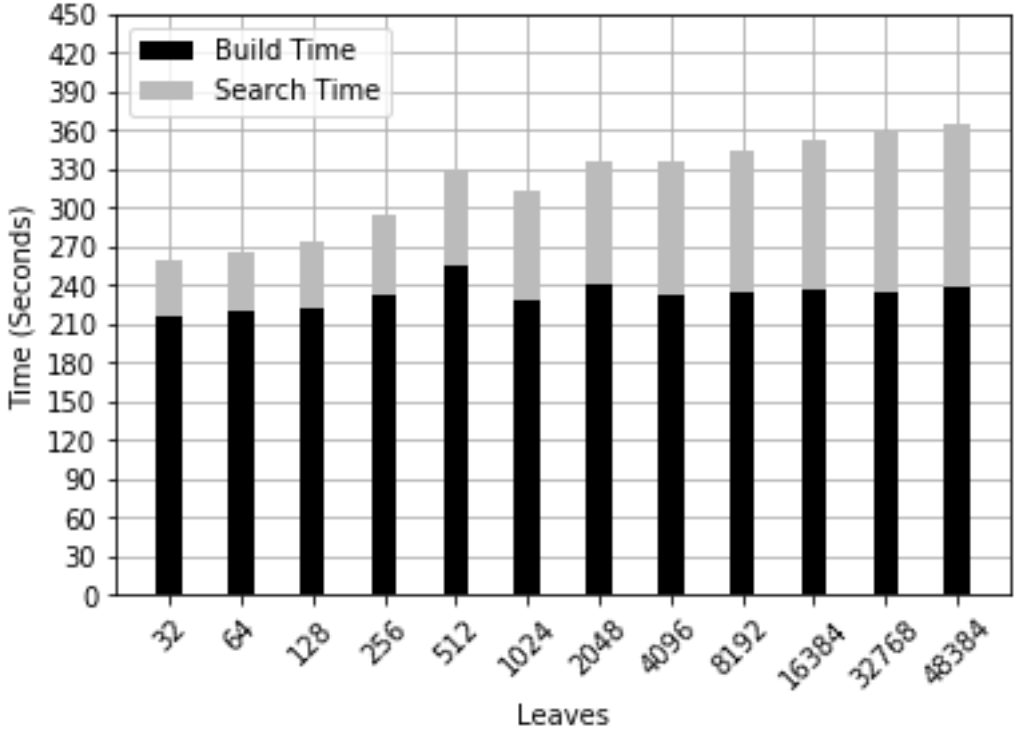
Number of pairwise comparisons required at leaves.

Experiment #2

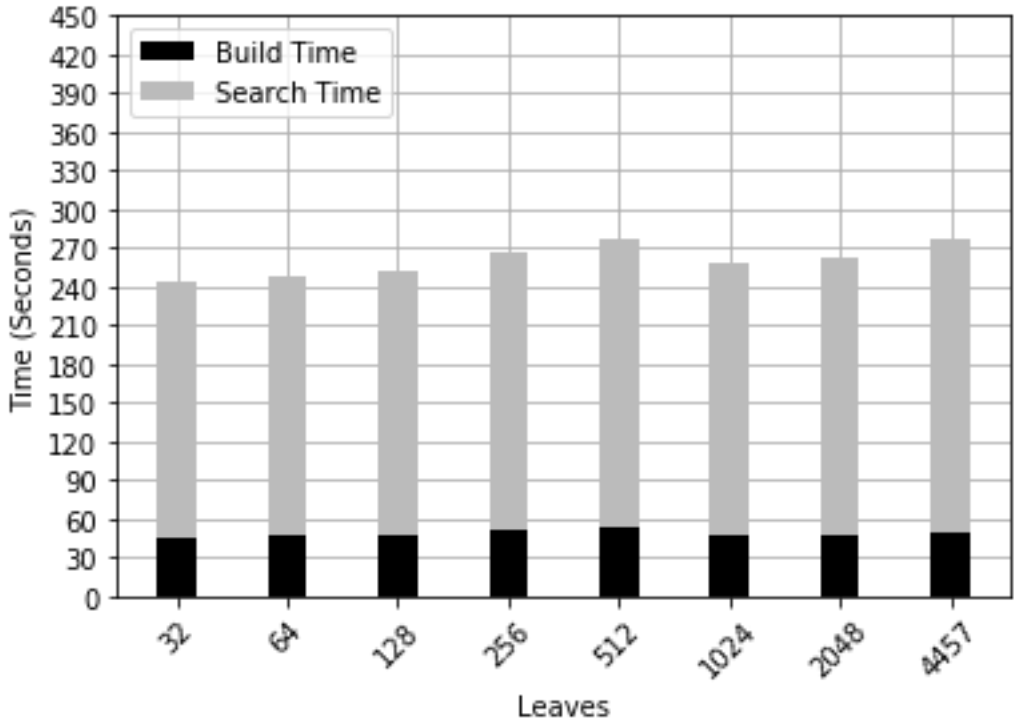
- Datasets:
  - Tree represents win7, search for t5.
  - Tree represents t5, search for win7.
- Investigate whether HBFT should represent the smaller or larger corpus.
- Measure effect on overall running time.



Experiment #2: Results



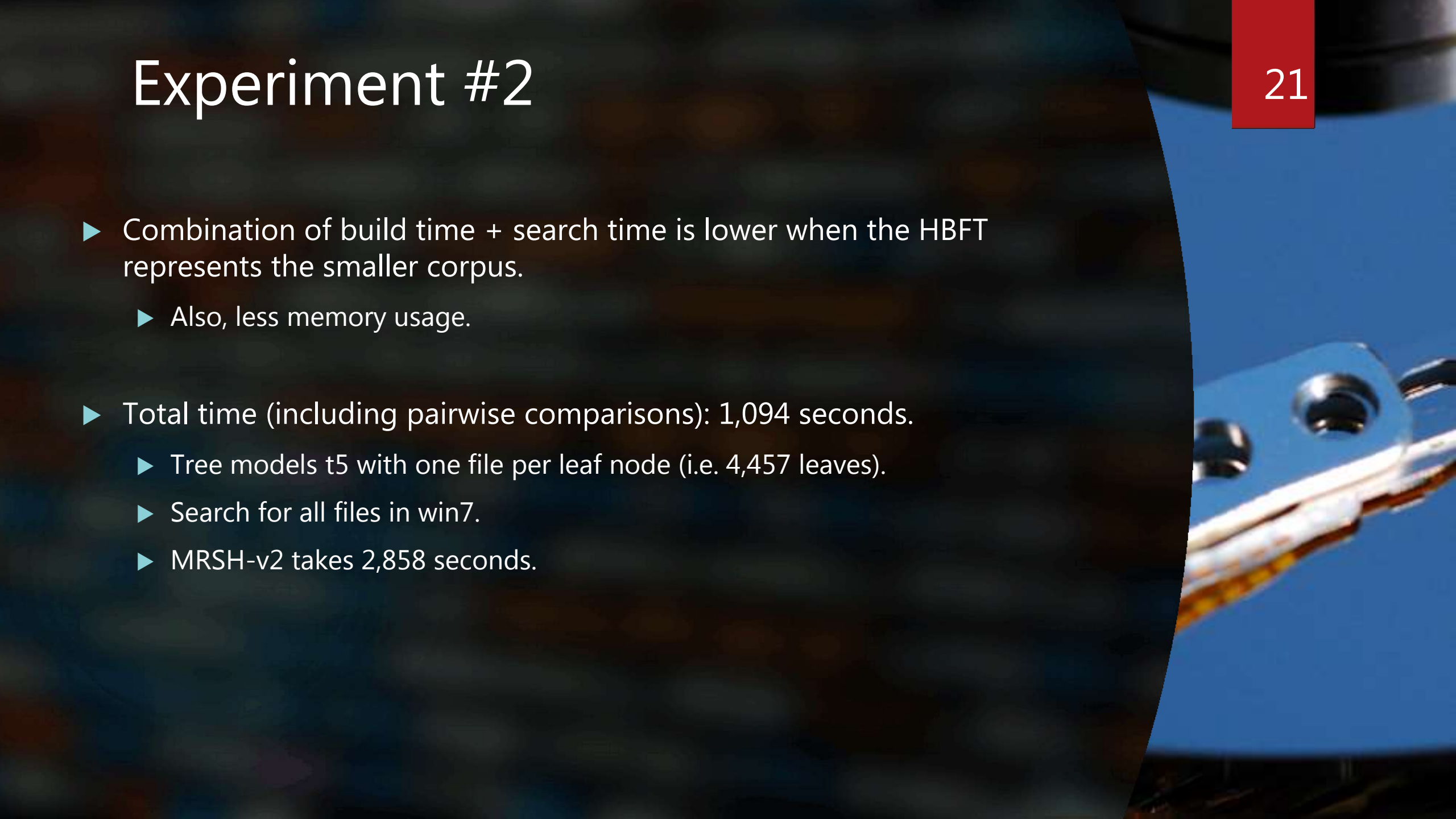
Time to search for t5 in a win7 tree.
(excluding pairwise comparisons)



Time to search for win7 in a t5 tree.
(excluding pairwise comparisons)

Experiment #2

- Combination of build time + search time is lower when the HBFT represents the smaller corpus.
  - Also, less memory usage.
- Total time (including pairwise comparisons): 1,094 seconds.
  - Tree models t5 with one file per leaf node (i.e. 4,457 leaves).
  - Search for all files in win7.
  - MRSH-v2 takes 2,858 seconds.



Experiment #3

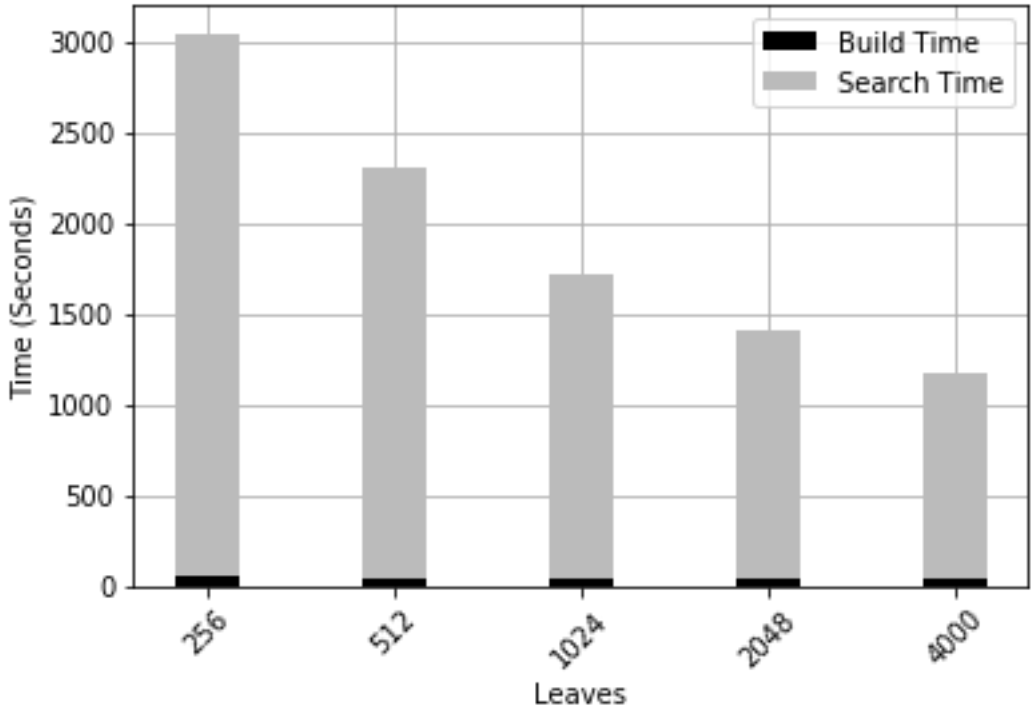
- Datasets:
  - 4,000 files from t5 represent set of “known-illegal” files.
  - win7 represents seized disk image, with 140 “planted” files from t5 added:
    - 100 files that are also in the “known-illegal” set.
    - 40 files with high similarity to files in the “known-illegal” set:
      - 10 that have ≥ 80% similarity.
      - 10 that have ≥ 60% and < 80% similarity.
      - 10 that have ≥ 40% and < 60% similarity.
      - 10 that have ≥ 20% and < 40% similarity.
- Aims:
  - Compare time to MRSH-v2
  - Evaluate effectiveness of finding planted files.

Experiment #3: Results

MRSH-v2 similarity	Files planted	Files found	Similar recall
80%-100%	10	10	100%
60%-79%	10	10	100%
40%-59%	10	10	100%
20%-39%	10	8	80%
Overall	**40**	**38**	**95%**

Running time (4,000 leaves):

- **MRSH-v2:** 2,592 seconds.
- **HBFT:** 1,182 seconds.



Time to search for planted evidence.
(including pairwise comparisons)

Conclusions

- More leaf nodes lead to fewer pairwise comparisons.
- min_run of 4 looks like a reasonable value.
- If corpora are different sizes, use the tree to represent the smaller one.
- Final experiment: all files with ≥ 20% similarity were found, with time reduction of 54%.
- Likely to scale better than existing approach using pairwise comparisons.

DAVID.LILLIS@UCD.IE

WWW.FORENSICSANDSECURITY.COM

@FORSECRESEARCH

UCD Forensics and
Security Research Group