Scalable Data Correlation

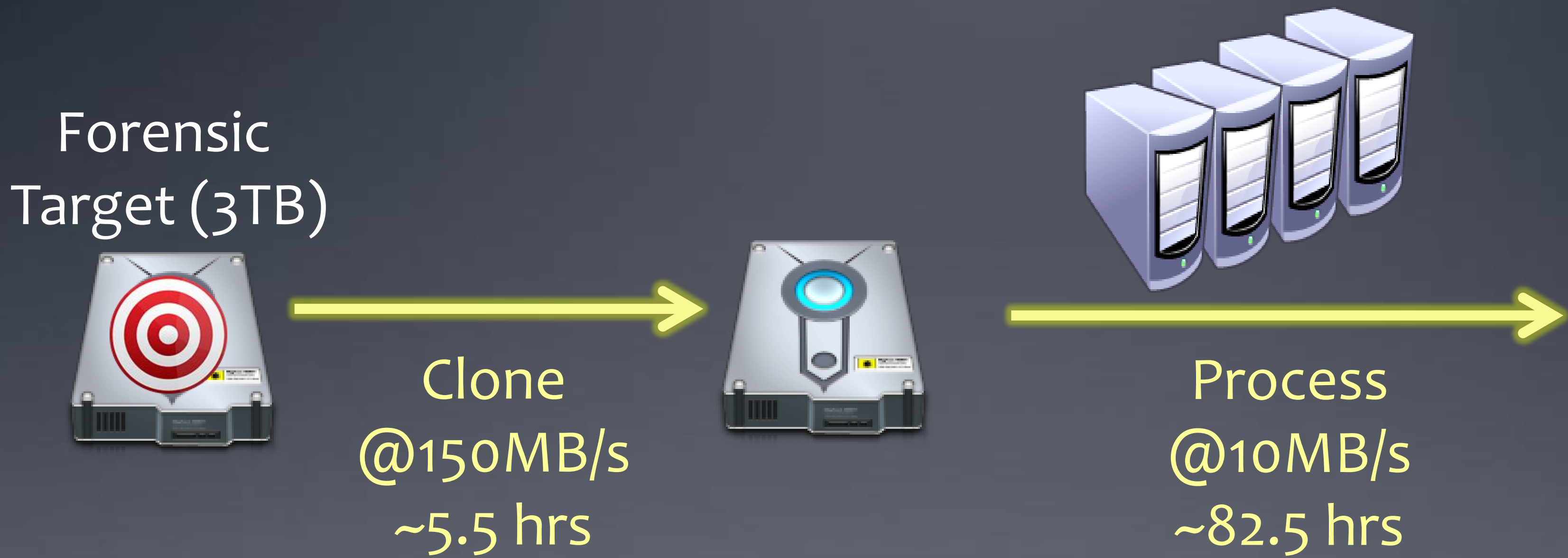
Managing TB-scale investigations with similarity digests

THE UNIVERSITY of NEW ORLEANS

Vassil Roussev

vassil@roussev.net

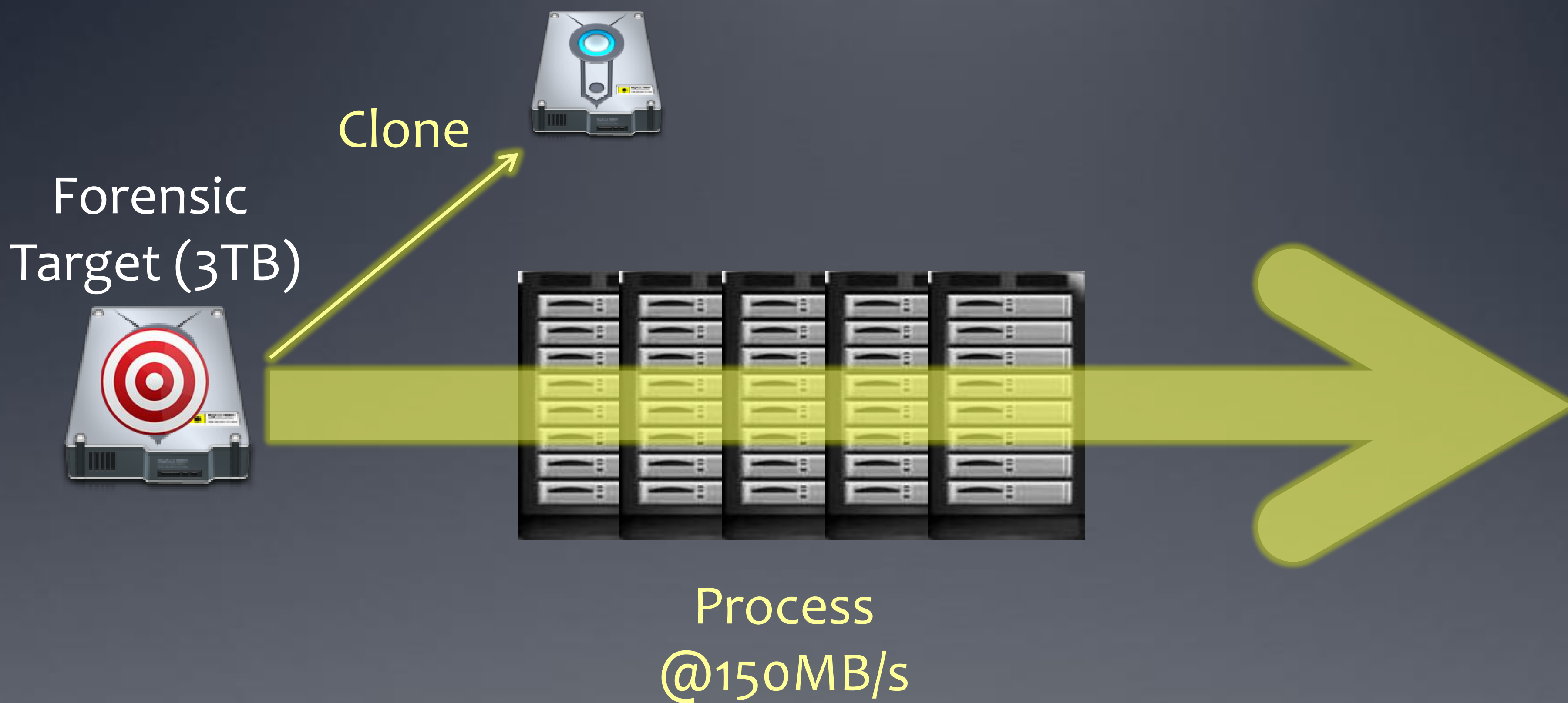
The Current Forensic Workflow



➔ We can ***start*** working on the case after ~~88~~ 129* hours.

* http://accessdata.com/distributed-processing

Scalable Forensic Workflow



→ We can start working on the case ***immediately***.

Current Forensic Processing

- Hashing/filtering/correlation
- File carving/reconstruction
- Indexing

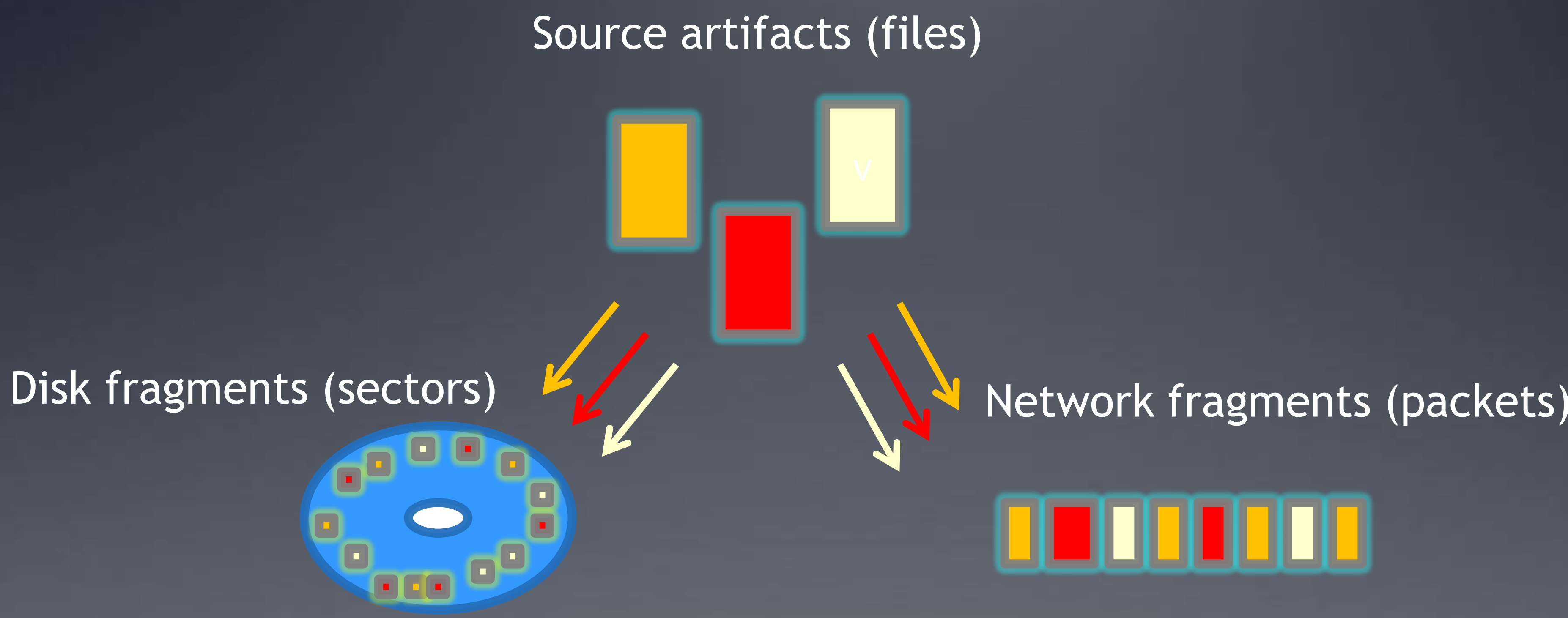
The ultimate goal of this work is to make similarity hash-based correlation scalable & I/O-bound.

Motivation for similarity approach:
Traditional hash filtering is failing

- *Known file filtering:*
  - Crypto-hash known files, store in library (e.g. NSRL)
  - Hash files on target
  - Filter in/out depending on interest
- Challenges
  - Static libraries are falling behind
    - Dynamic software updates, trivial artifact transformations

    → We need **version** correlation
  - Need to find embedded objects
    - Block/file in file/volume/network trace
  - Need higher-level correlations
    - Disk-to-RAM
    - Disk-to-network

Scenario #1: Fragment Identification



- Given a fragment, identify source
  - Fragments of interest are 1-4KB in size
  - Fragment ***alignment is arbitrary***

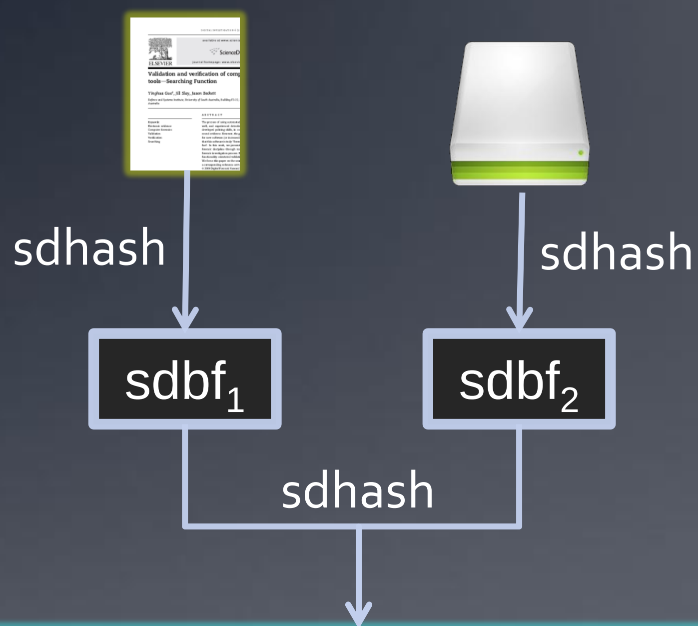
Scenario #2: Artifact Similarity

Similar files
(shared content/format)

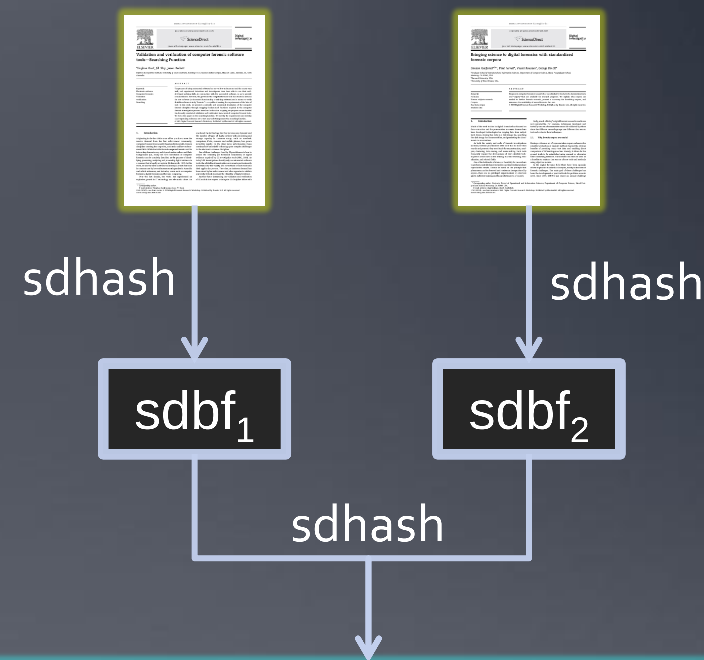
Similar drives
(shared blocks/files)

- Given two binary objects, detect similarity/versioning
  - Similarity here is purely syntactic;
  - Relies on commonality of the binary representations.

Solution: Similarity Digests



Is this fragment present on the drive?
→ 0 .. 100



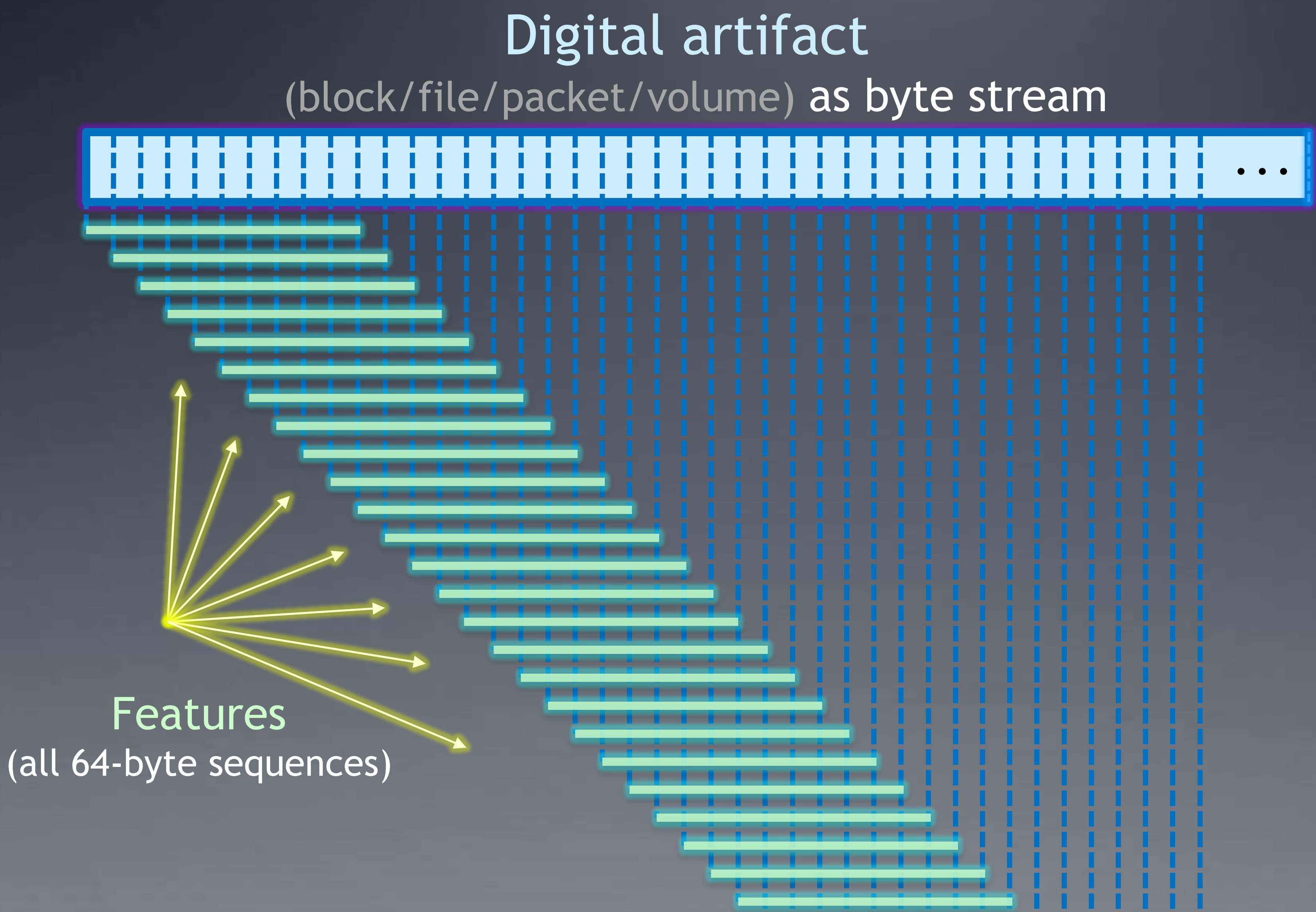
Are these artifacts correlated?
→ 0 .. 100

All correlations based on bitstream commonality

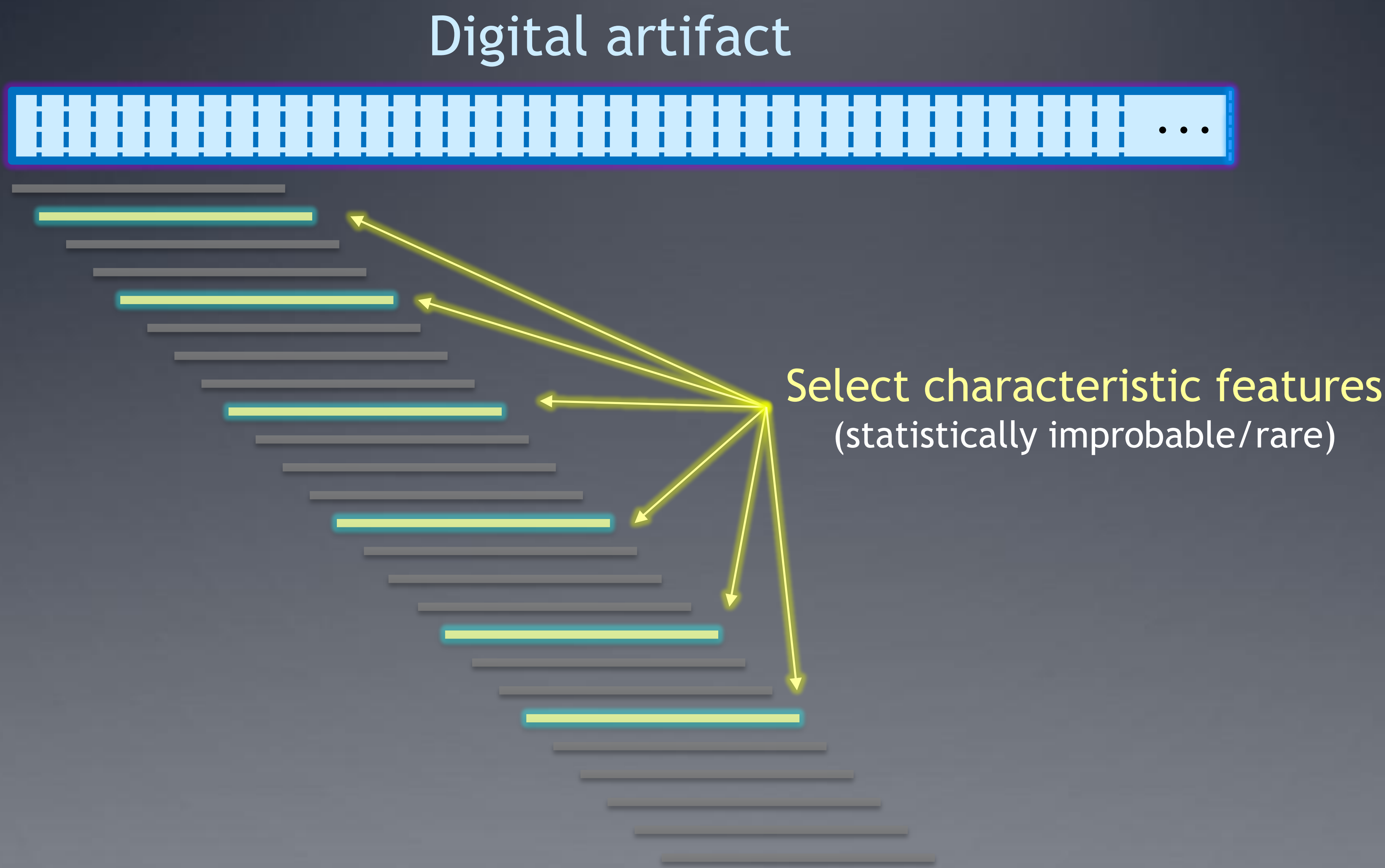
Quick Review:

Similarity digests & sdhash

Generating sdhash fingerprints (1)

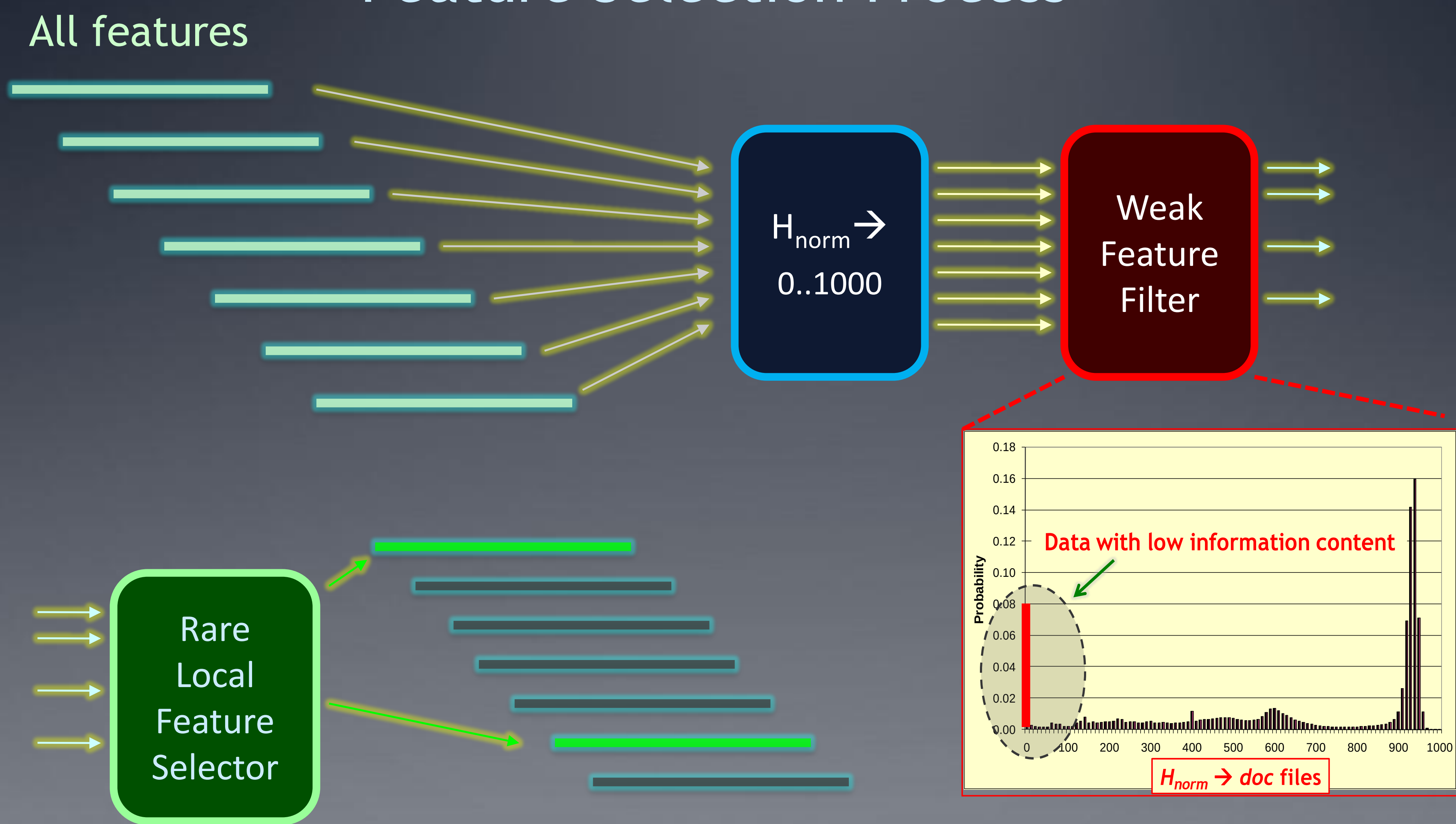


Generating sdhash fingerprints (2)

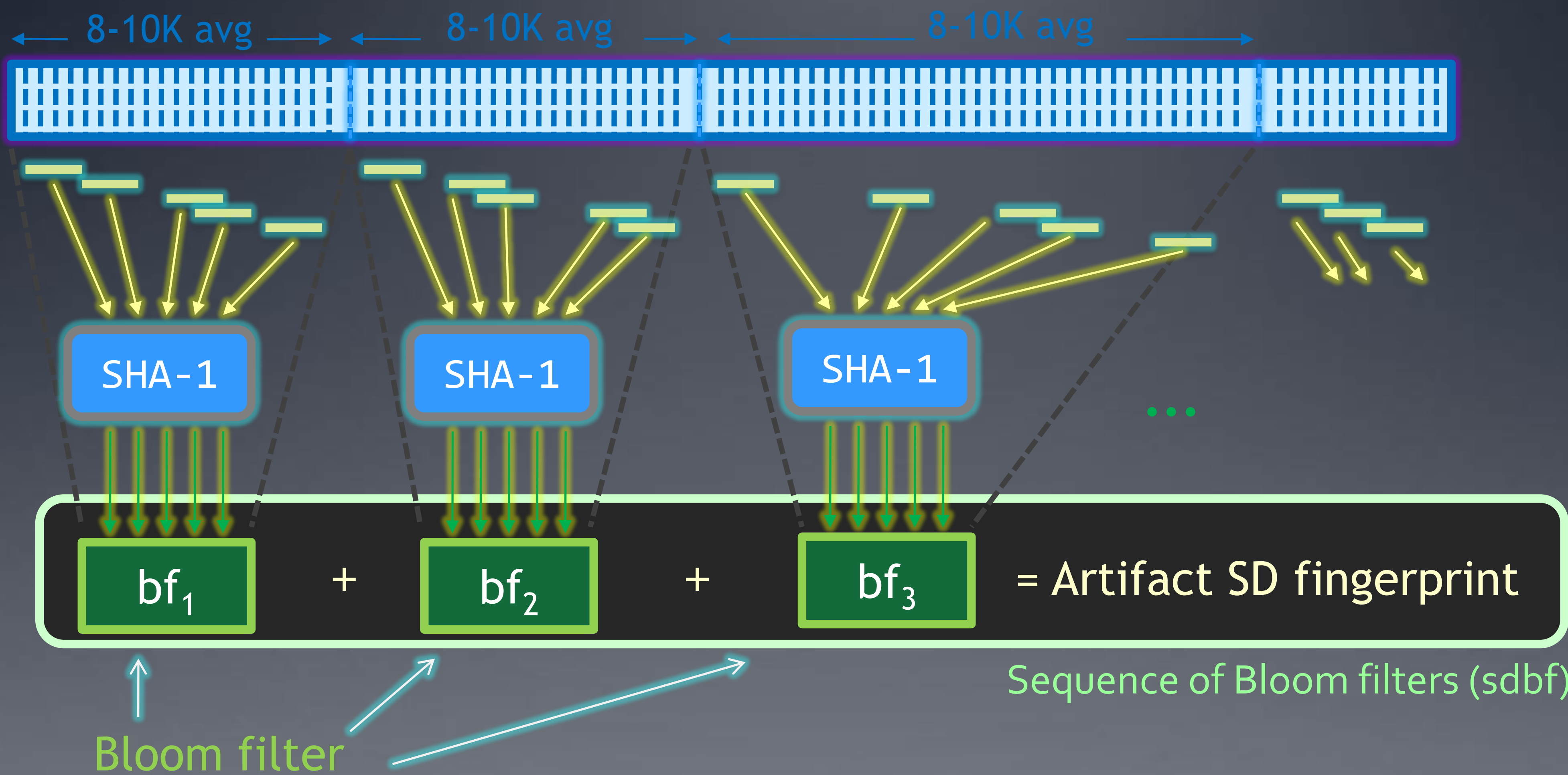


Generating sdhash fingerprints (3)

Feature Selection Process

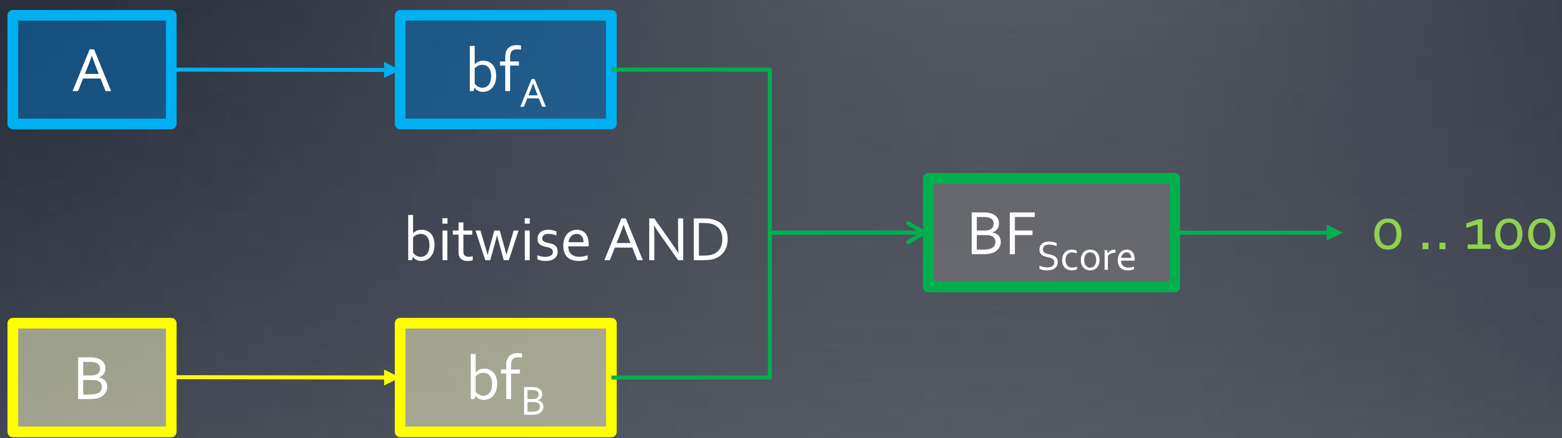


Generating sdhash fingerprints (4)



- local SD fingerprint
- 256 bytes
- up to 128/160 features

Bloom filter (BF) comparison

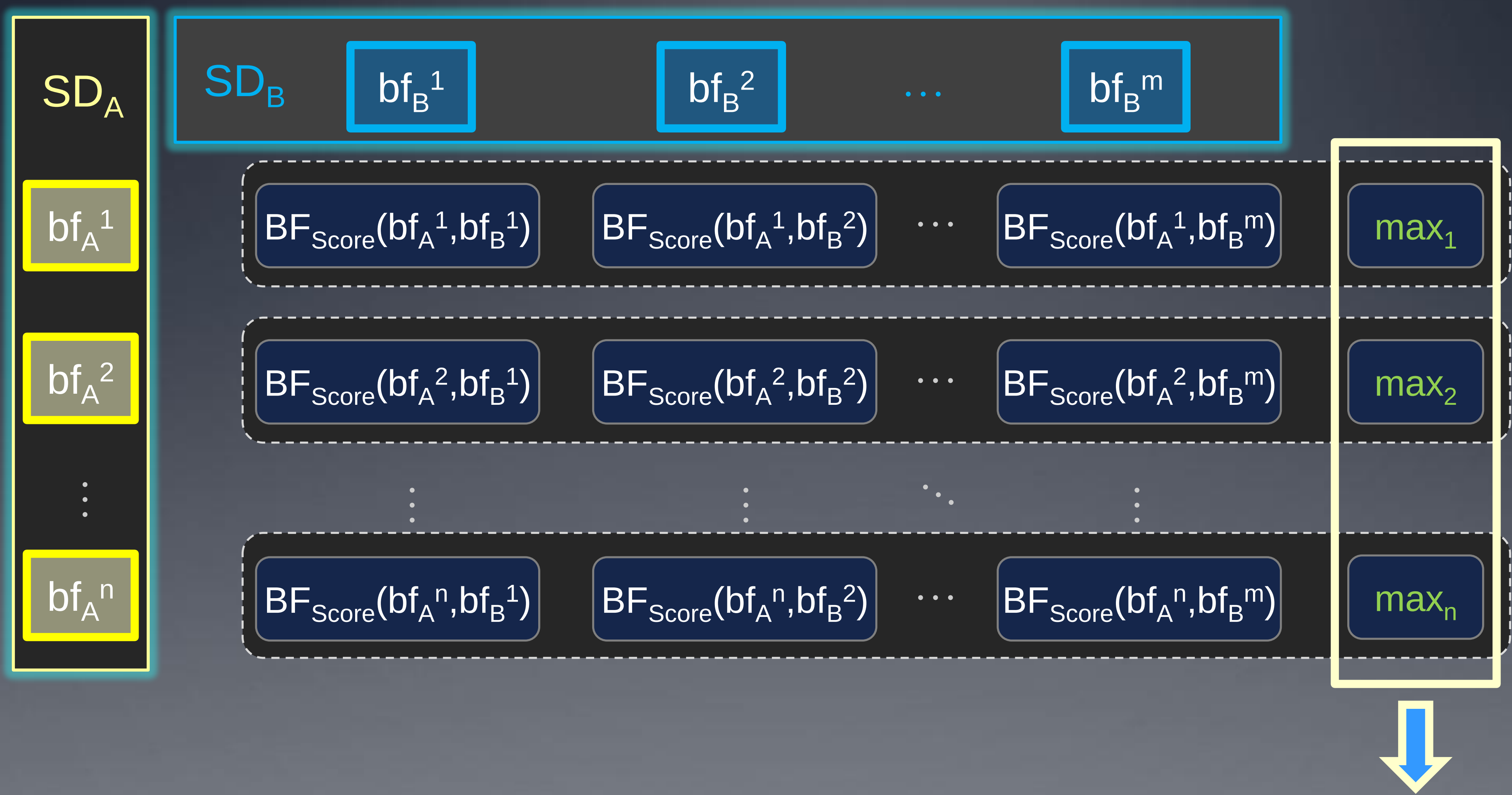


Based on BF theory,
overlap due to chance is analytically predictable.

Additional BF overlap is proportional to overlap in features.

BF_{Score} is tuned such that $BF_{Score}(A_{random}, B_{random}) = 0$.

SDBF fingerprint comparison

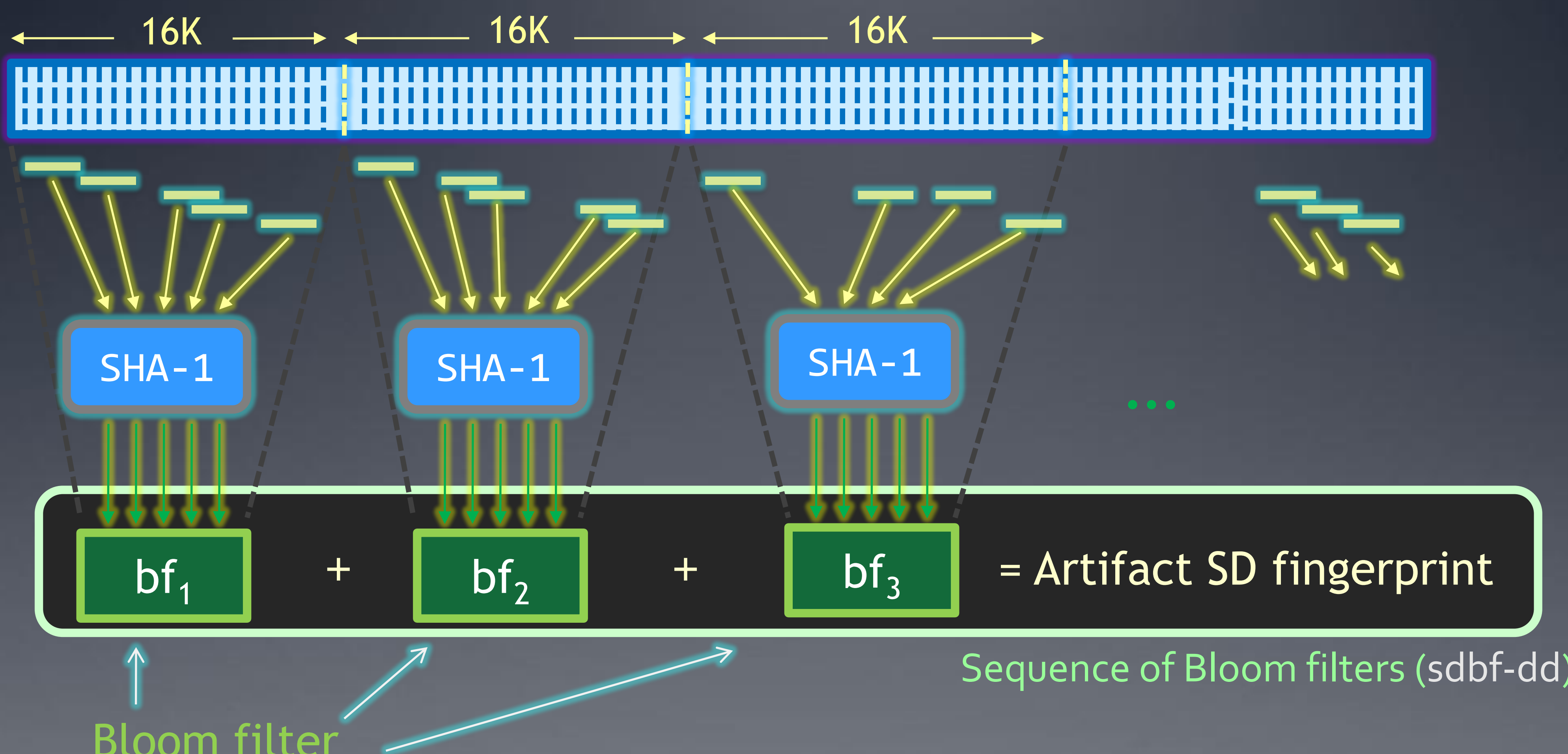


$$SD_{Score}(A,B) = \text{Average}(max_1, max_2, \ldots, max_n)$$

Scaling up:

Block-aligned digests & parallelization

Block-aligned similarity digests (sdbf-dd)



Bloom filter

- local SD fingerprint
- 256 bytes
- up to 192 features

Advantages & challenges for block-aligned similarity digests (sdbf-dd)

- Advantages
  - Parallelizable computation
  - Direct mapping to source data
  - Shorter (1.6% vs 2.6% of source)
  - ➔ Faster comparisons (fewer BFs)
- Challenges
  - Less reliable for smaller files
  - Sparse data
  - Compatibility with sdbf digests
- Solution
  - Increase features for sdbf filters: 128➔ 160
  - Use 192 features per BF for sdbf-dd filters
  - Use compatible BF parameters to allow sdbf ⇔ sdbf-dd comparisons

sdhash 1.6: sdbf vs. sdbf-dd accuracy

Query size	FP rate	TP rate	Query size	FP rate	TP rate
1,000	0.1906	1.000	2,000	0.0006	0.997
1,100	0.0964	1.000	2,200	0.0005	1.000
1,200	0.0465	1.000	2,400	0.0001	1.000
1,300	0.0190	1.000	2,600	0.0001	0.997
1,400	0.0098	1.000	2,800	0.0000	1.000
1,500	0.0058	1.000	3,000	0.0000	0.999
1,600	0.0029	0.999	3,200	0.0000	0.998
1,700	0.0023	0.999	3,400	0.0000	0.998
1,800	0.0013	0.999	3,600	0.0000	1.000
1,900	0.0010	0.998	3,800	0.0000	0.998

Sequential throughput: sdhash 1.3

- Hash generation rate
  - Six-core Intel Xeon X5670 @ 2.93GHz
    ~27MB/s per core
  - Quad-Core Intel Xeon @ 2.8 GHz
    ~20MB/s per core
- Hash comparison
  - 1MB vs. 1MB: 0.5ms
- T5 corpus (4,457 files, all pairs)
  - 10mln file comparisons in ~ 15min
    - 667K file comps per second
    - Single core

sdhash 1.6: File-parallel generation rates on 27GB real data (in RAM)

Threads	Time (sec)	Throughput (MB/s)	Speedup
1	920	29.08	1.00
4	277	96.57	3.32
8	187	143.05	4.92
12	144	185.76	6.39
24	129	207.36	7.13

sdhash 1.6: Optimal file-parallel generation: 5GB synthetic target (RAM)

Threads	Time (sec)	Throughput (MB/s)	Speedup
1	177	28.25	1.00
4	50	100.00	3.54
8	30	166.67	5.90
12	24	208.33	7.38
24	19	263.16	9.32

sdhash-dd: Hash generation rates 10GB in-RAM target (RAM)

Threads	Time (sec)	Throughput (MB/s)	Speedup
1	374.0	26.74	1.00
4	93.0	107.53	4.02
8	53.0	188.68	7.06
12	44.5	224.72	8.40
24	27.0	370.37	13.85

Throughput summary: sdhash 1.6

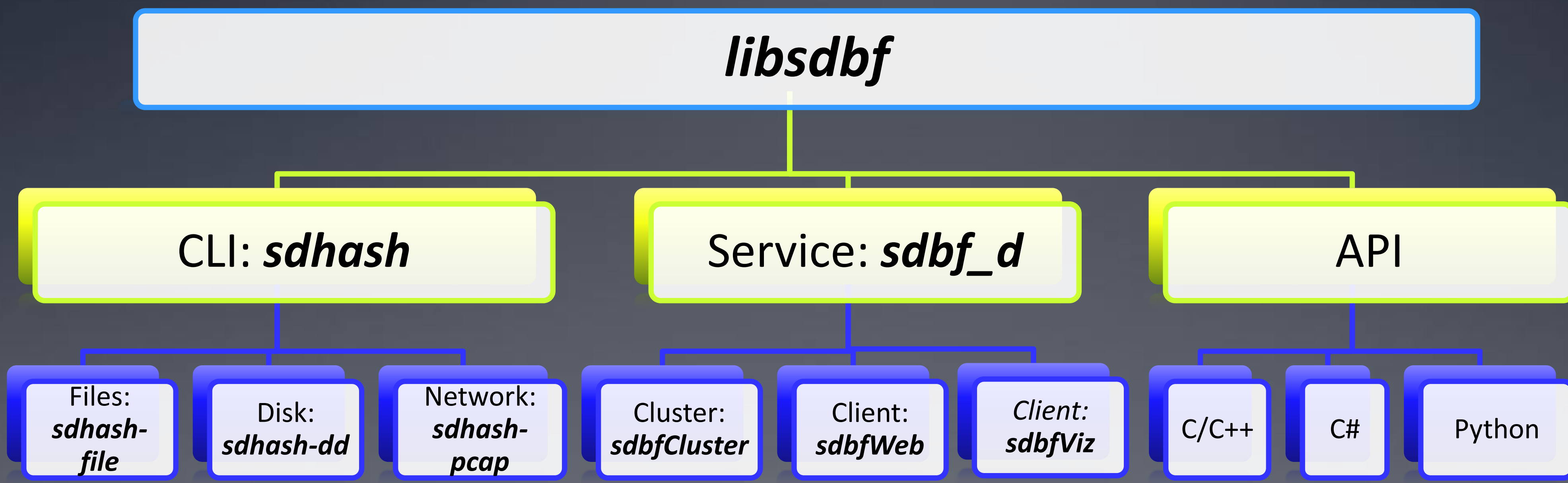
- Parallel hash generation
  - sdbf: file-parallel execution
    - 260 MB/s on 12-core/24-threaded machine
  - sdbf-dd: block-parallel execution
    - 370 MB/s (SHA1 —> 330MB/s)

- Optimized hash comparison rates
  - 24 threads: 86.6 mln BF/s

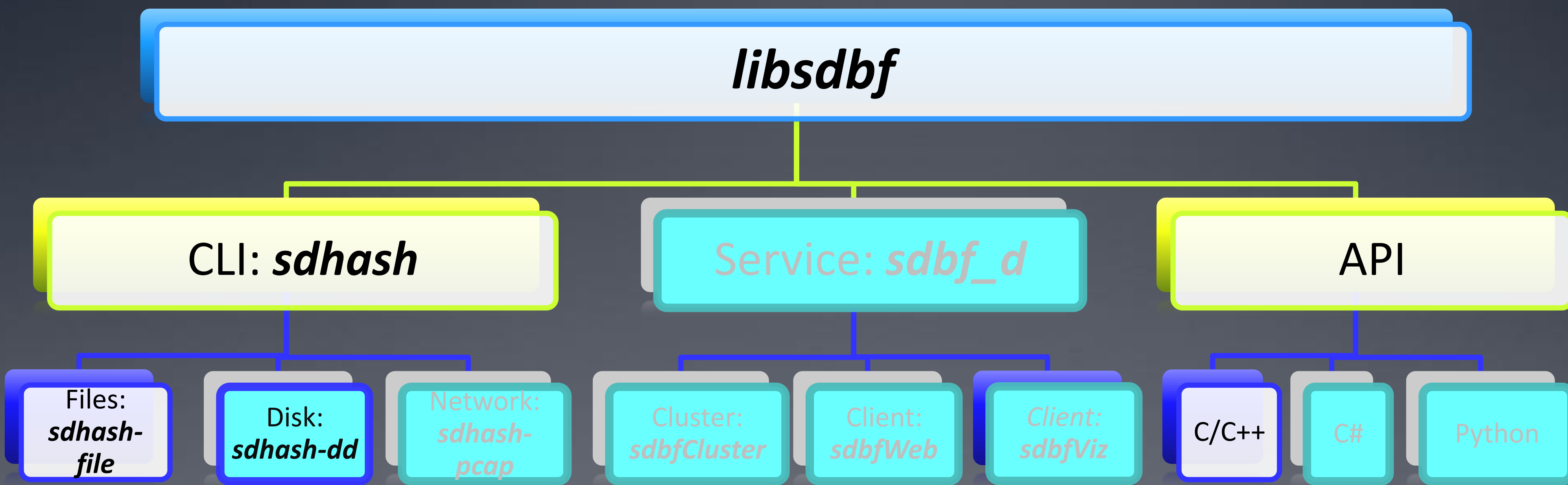
  ➔ 1.4 TB/s for small file comparison (<16KB)

  I.e., we can search for a small file in a reference set of 1.4TB in 1s

The Envisioned Architecture



The Current State



Todo List (1)

- ***libsdbf***
  - C++ rewrite (v2.0)
  - TBB parallelization
- ***sdhash-file***
  - More command line options/compatibility w/ssdeep
  - Service-based processing (w/ ***sdbf_d***)
- ***GPU acceleration***
- ***sdhash-pcap***
  - Pcap-aware processing:
    - payload extraction, file discovery, timelining

Todo List (2)

- ***sdbf_d***
  - Persistance: XML
  - Service interface: JSON
  - Server clustering
- ***sdbfWeb***
  - Browser-based management/query
- ***sdbfViz***
  - Large-scale visualization & clustering

Further Development

- Integration w/ RDS
  - ***sdhash-set***: construct *SDBF*s from existing SHA1 sets
    - Compare/identify whole folders, distributions, etc.
- Structural feature selection
  - E.g., exe/dll, pdf, zip, …
- Optimizations
  - Sampling
  - Skipping
    - Under **min** continuous block assumption
  - Cluster "core" extraction/comparison
- Representation
  - Multi-resolution digests
  - New crypto hashes
  - Data offsets

Thank you!

- http://roussev.net/sdhash
  - wget http://roussev.net/sdhash/sdhash-1.6.zip
  - make
  - ./sdhash
- Contact:

Vassil Roussev

vassil@roussev.net

- Reminder

DFRWS'12: Washington DC, Aug 6-8

Paper deadline: Feb 20, 2012

Data sniffing challenge to be released shortly