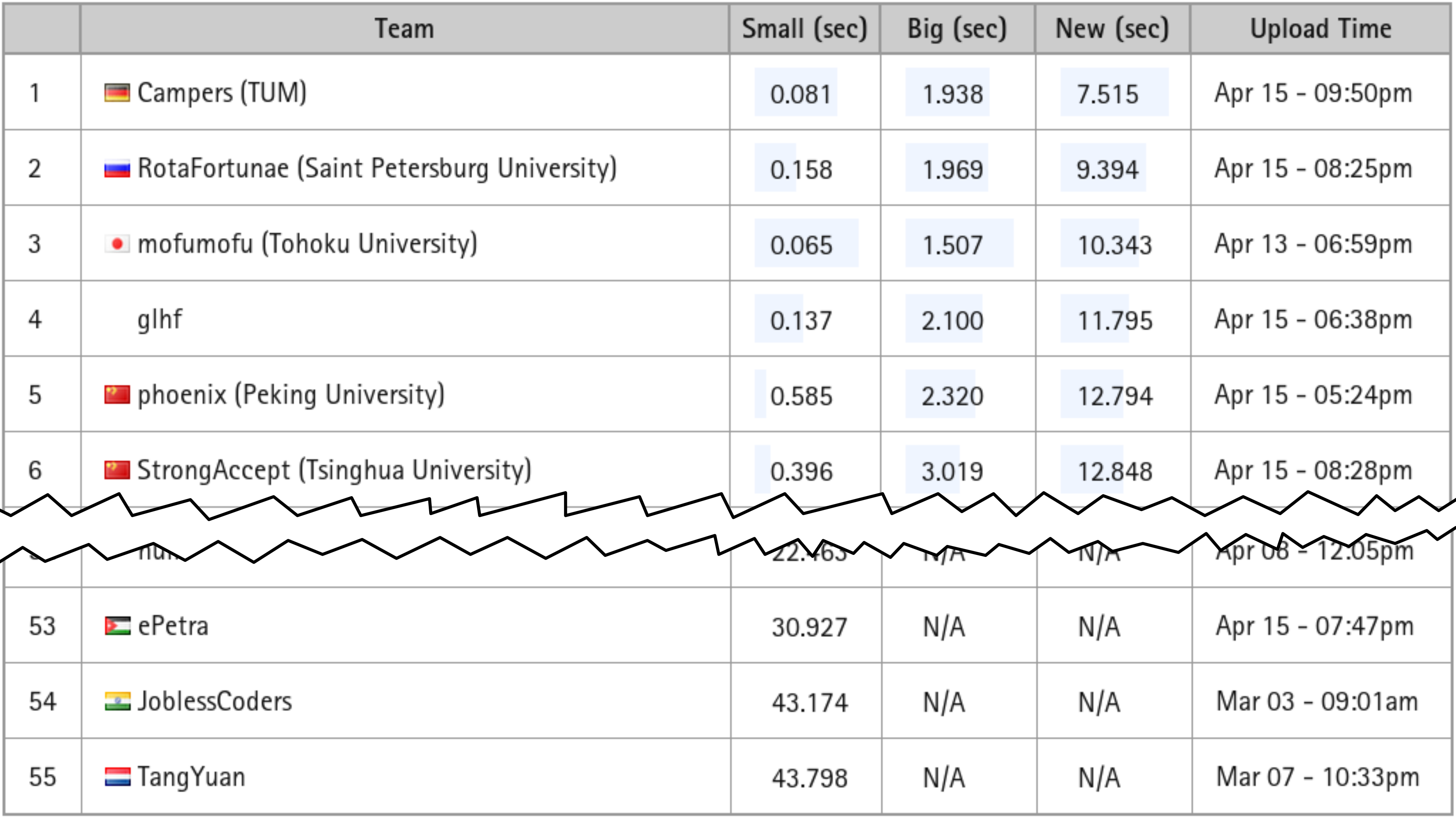
Winning* the SIGMOD 2013 programming contest

Henrik Mühe henrik.muehe@in.tum.de
Florian Funke florian.funke@in.tum.de

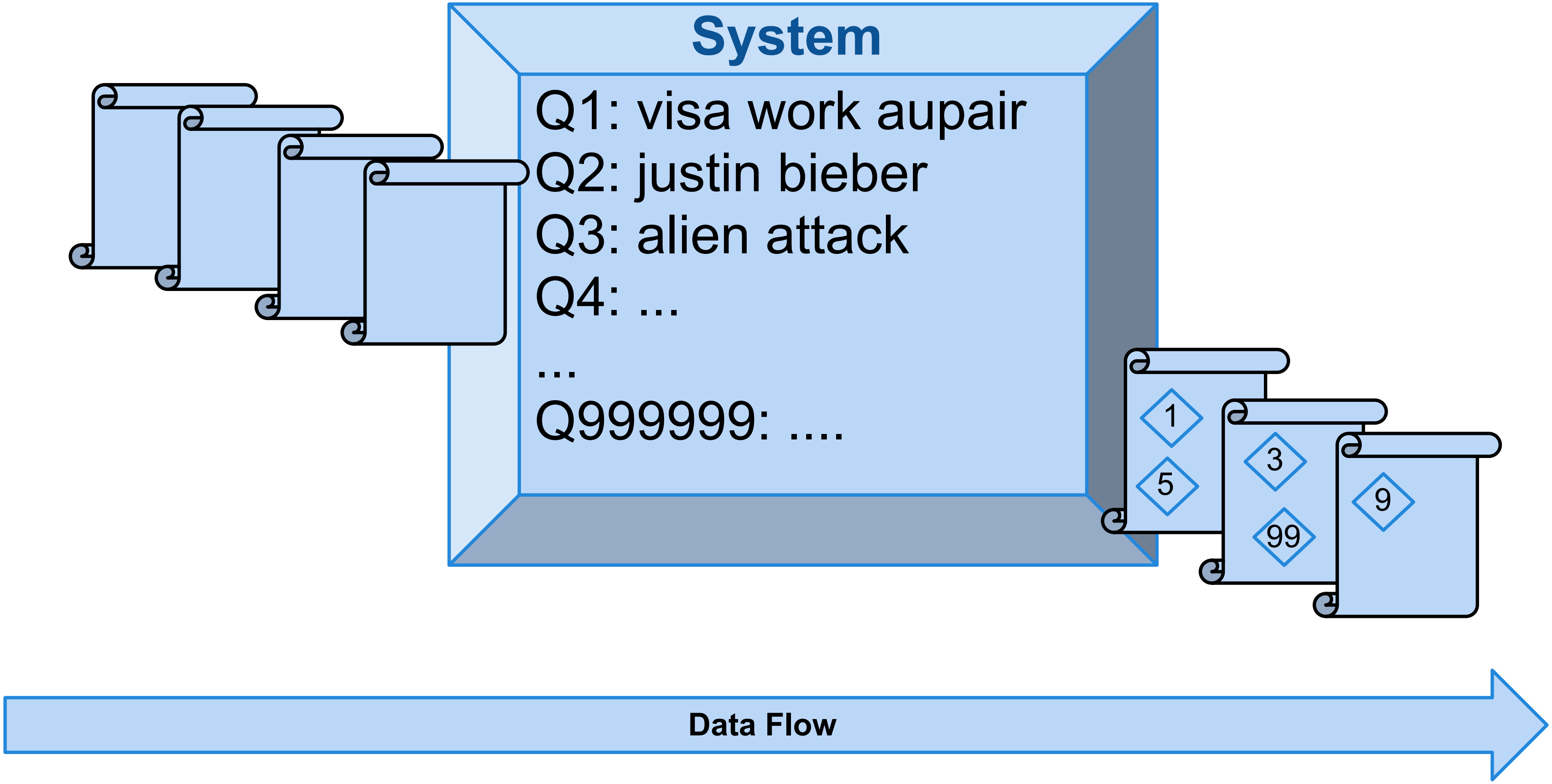
SIGMOD Conference and Challenge

Leaderboard



	Team	Small (sec)	Big (sec)	New (sec)	Upload Time
1	Campers (TUM)	0.081	1.938	7.515	Apr 15 - 09:50pm
2	RotaFortunae (Saint Petersburg University)	0.158	1.969	9.394	Apr 15 - 08:25pm
3	mofumofu (Tohoku University)	0.065	1.507	10.343	Apr 13 - 06:59pm
4	glhf	0.137	2.100	11.795	Apr 15 - 06:38pm
5	phoenix (Peking University)	0.585	2.320	12.794	Apr 15 - 05:24pm
6	StrongAccept (Tsinghua University)	0.396	3.019	12.848	Apr 15 - 08:28pm
[illegible]	nu	22.463	N/A	N/A	Apr 08 - 12:05pm
53	ePetra	30.927	N/A	N/A	Apr 15 - 07:47pm
54	JoblessCoders	43.174	N/A	N/A	Mar 03 - 09:01am
55	TangYuan	43.798	N/A	N/A	Mar 07 - 10:33pm

The Challenge



System

Q1: visa work aupair
Q2: justin bieber
Q3: alien attack
Q4: ...
...
Q999999: ....

The Metrics:
Exact Match

Query matches a document iff all query words are contained in the document.

The Metrics:
Hamming Distance

Query matches a document iff all query words are within hamming distance d of at least on word inside the document.

Hamming?

Jamming?

1 position differs ⇨ HD=1

The Metrics:
Levenshtein Distance

Query match

words are wit

at least on v

ent iff all query

ein distance d of

he document.

Lev

tein?

Levenshtein Examples

```
levenshtein
   henrik
   jenrik
= 1 (= hd())
```

```
levenshtein
   alfons
   fonts
= 3
```

```
levenshtein
   abc
   abcdef
= 3
```

Levenshtein Definition

levenshtein(a,b) :=

Lowest number of

- Replace
- Insert
- Remove

to change a into b

O(|a|*|b|) ⇦ terrible

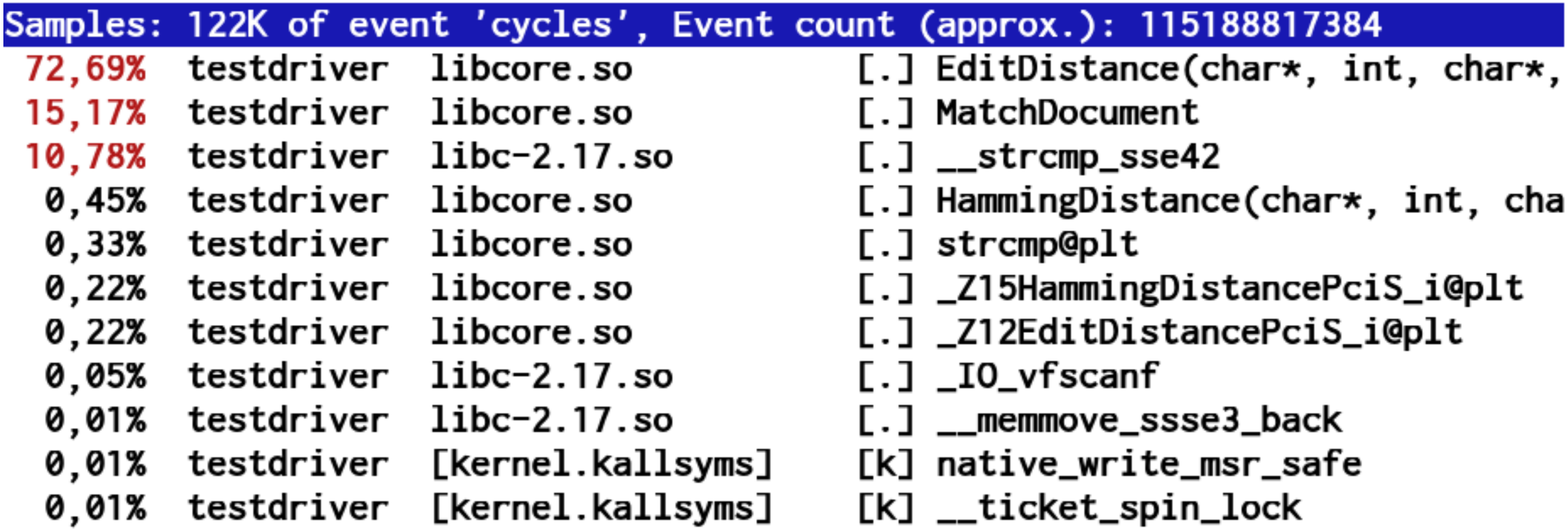
Baseline

- tar.gz download, fully functional
- Naive 'nested-loop' style
- Unbearably slow
- Horrible, horrible code

```
int cur=0;
ia=0;
for(ib=0;ib<=nb;ib++)
    T[cur][ib]=ib;
cur=1-cur;
```

Baseline Analysis

```
$ ./testdriver
Start Test ...
Your program has passed all tests.
Time=30704[30s:704ms]
$ perf record ./testdriver && perf report
```



```
Samples: 122K of event 'cycles', Event count (approx.): 115188817384
 72,69%  testdriver  libcore.so          [.] EditDistance(char*, int, char*,
 15,17%  testdriver  libcore.so          [.] MatchDocument
 10,78%  testdriver  libc-2.17.so        [.] __strcmp_sse42
  0,45%  testdriver  libcore.so          [.] HammingDistance(char*, int, cha
  0,33%  testdriver  libcore.so          [.] strcmp@plt
  0,22%  testdriver  libcore.so          [.] _Z15HammingDistancePciS_i@plt
  0,22%  testdriver  libcore.so          [.] _Z12EditDistancePciS_i@plt
  0,05%  testdriver  libc-2.17.so        [.] _IO_vfscanf
  0,01%  testdriver  libc-2.17.so        [.] __memmove_ssse3_back
  0,01%  testdriver  [kernel.kallsyms]   [k] native_write_msr_safe
  0,01%  testdriver  [kernel.kallsyms]   [k] __ticket_spin_lock
```

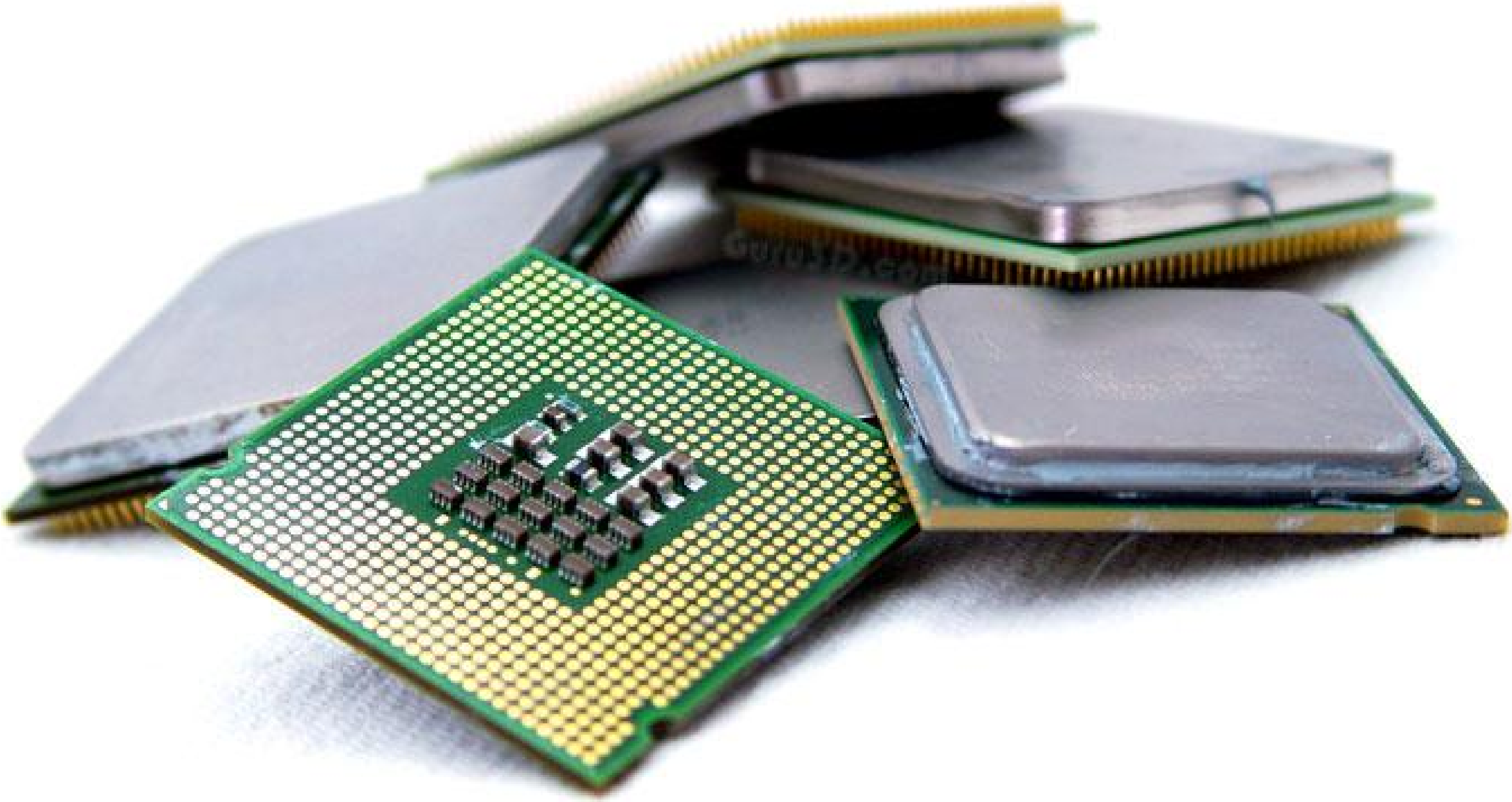
API

- `StartQuery`
- `EndQuery`
- `MatchDocument`
- `GetNextAvailRes`

The Magic Sauce

1. Massive parallelism
2. Architecture-aware optimizations
3. Efficient computation of metrics
4. Filtering
5. Indexing
6. Caching

1. Parallelism & Concurrency



1. Parallelism & Concurrency

MatchDocument

- Spawn async task with subtasks for each match type
- Parallelize Hamming & Levensthein distance
- Avoid sync points

Inherent Optimization Potentials

Deduplication

- Remove all duplicates in document

- Match every query word only once (even if it is in multiple queries)

Caveats

Q1: henrik mühe
Q2: henrik database
Q3: henrik funfacts

QueryWords: henrik, mühe, database, funfacts

Caveats

Q1: henrik mühe
Q2: henrik database
Q3: henrik funfacts

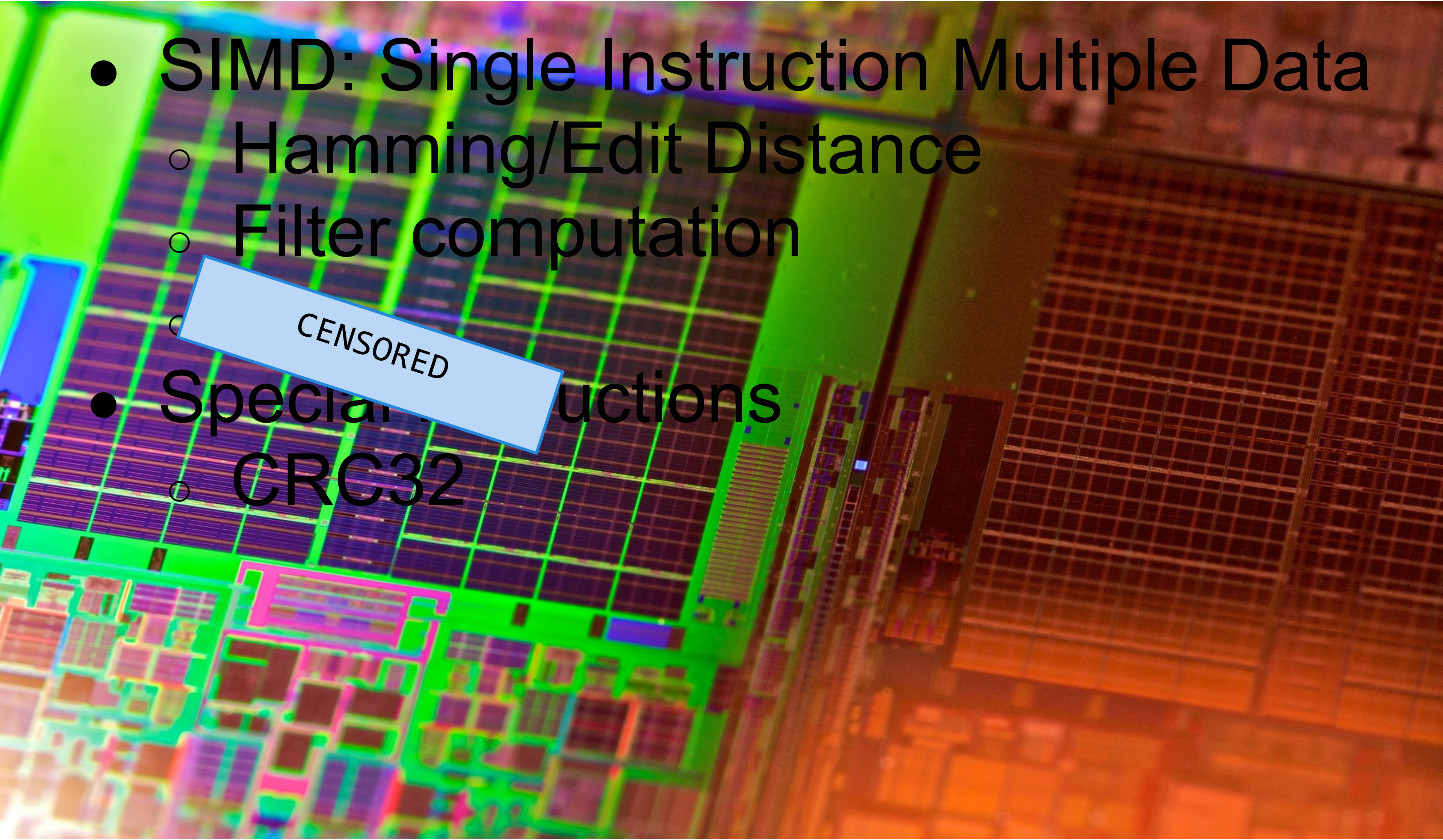
QueryWords: henrik, mühe, database, funfacts

Document.probe(henrik) -> false
What about: mühe, database, funfacts

Cover Pruning

- For every word, determine which words can be skipped.
  - Full computation too expensive
  - When a query is added, remove word from invalidated dependency sets
  - Do not re-add
  - Recompute when queries have changed substantially
- Skip vector in hot loop
- Harmless race condition

2. Architecture-Aware Optimizations



- SIMD: Single Instruction Multiple Data
  - Hamming/Edit Distance
  - Filter computation
  - CENSORED
- Specia uctions
  - CRC32

3. Efficient computation of metrics

Improving Exact Match

Insert all query words into Hashmap
Signature: `hash<QueryWord,vector<Query>>`

1. Probe each document word &
   Mark QueryWord as matched
2. Count matching words per query
3. Generate result

Improving Hamming Distance

Materialize all and add to Exact Matcher?

Improving Hamming Distance

Materialize all and add to Exact Matcher?

For word with length 10 and distance 3 roughly

d=1	10 * 25
d=2	+ (10 * 25)^2
d=3	+ (10 * 25)^3

>> 15 000 000

Improving Hamming Distance

Hamming is essentially the sum of bytewise XOR

```
   x=   aaaabbbb
   y=   bbaaabbb
sum  ( 11001000 ) = 3 = hamming(x,y)
```

Improving Hamming Distance

SIMD easy solution:

```
POPCNT(PCMPESTRM)
```

SIMD fastest solution:

CENSORED

Improving Edit Distance: Naive Algorithm

```
/// Compute levenshtein distance recursively
inline uint32_t levenshtein_rec(StringRef a,StringRef b) {
        // If one of the strings is empty, return the number of characters left
        if (a.length()==0) return b.length();
        if (b.length()==0) return a.length();

        // If the first two characters are equal, the edit distance is the edit
        // distance between the two suffixes
        if (a[0]==b[0]) return levenshtein_rec(a.substring(1),b.substring(1));

        // If they are not equal, try insert,remove and substitution
        // Pretend a is b with an extra letter in front
        uint32_t dInsert=levenshtein_rec(a.substring(1),b);
        // Pretend a is b with the first letter removed
        uint32_t dRemove=levenshtein_rec(a,b.substring(1));
        uint32_t dSubst= levenshtein_rec(a.substring(1),b.substring(1));

        // Return the best of the three possibilities above and add one for the
        // insert/remove/substitution we did
        return std::min(dInsert,std::min(dRemove,dSubst)) + 1;
}
```

Improving Edit Distance

- Superset of Hamming Operations

- Literature Research
  - Validation:
    - Levenshtein Automata

  - Improved Algorithms
    - Memoization (matrix)
    - Less memoization (column)
    - Bit-parallel Levenshtein

Levenshtein Automaton Example

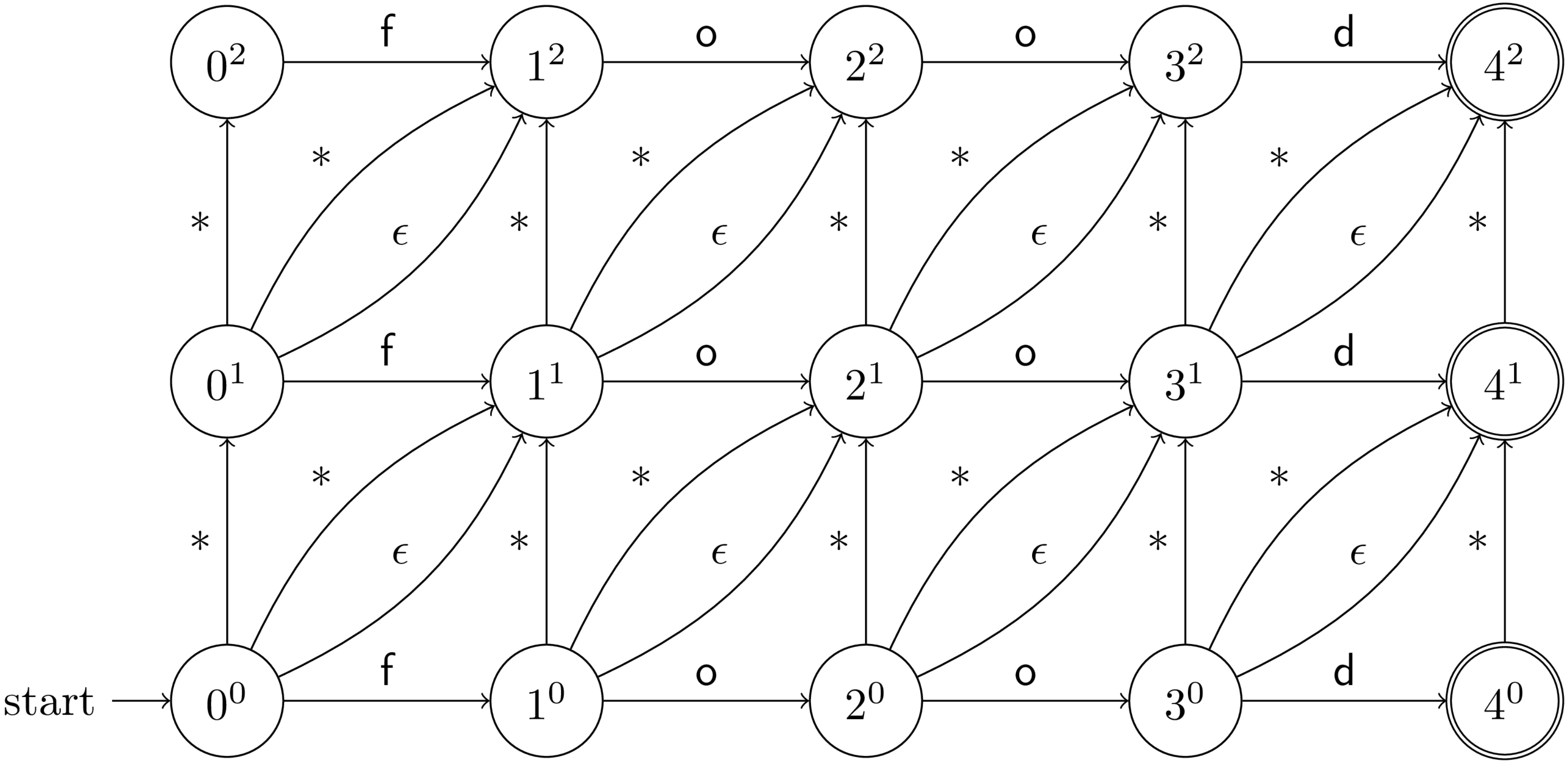


Figure 2: A finite automaton accepting strings less than three edits from “food”

CENSORED

4. Filtering

4. Filtering

- Determine if two words **can** be within edit/hamming distance
- Filter computation should be cheaper than metric invokation...
- Filters
  - Length
  - QGram
  - ...
  - Frequency

$$\overbrace{|qg(a,q) \cap qg(b,q)|}^{\text{Number of shared qgrams}} < (max(|a|,|b|) - q + 1) - q * d$$

Frequency Filter

Looking at the histograms of two words:

x= aaabbb

y= aacbba

H_x	H_y
a=3	a=3
b=3	b=2
c=0	c=1
d=0	d=0
...	...
z=0	z=0

Define *delta* operation

Max possible *delta*: 2d-lengthdiff

CENSORED

5. Indexing

SCHLAGWORTKATALOG
BAY - BEE
34
SCHLAGWORTKATALOG
BIBLIOGRAPH:N - BIBLIOTHEKEN:DD
40
SCHLAGWORTKATALOG
BISM - BODEN:E
SCHLAGWORTKATALOG
BEF - BERF
35
SCHLAGWORTKATALOG
BIBLIOTHEKEN:DE - BIBLIOTHEK:N
Bibliotheken:
Kataloge: Schlagwortkatalog
I
206 919

5. Indexing

- Physically reorganize words by some order relation
- Limit search space to a collocated subset
- Orders
  - Length
  - CENSORED
- Build column store
- Additive pointer arithmetics in hot loop

6. Caching

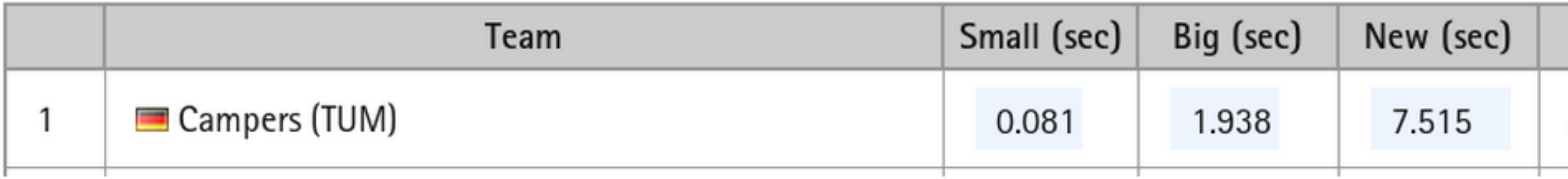
6. Caching

- Observation: People make the same mistakes again and again
- Remember last match
  - for each query word
  - for each distance
- Probing a (good!) hashtable is a lot cheaper than finding an edit distance match in an entire doc

Conclusion

```
$ ./testdriver
Start Test ...
Your program has passed all tests.
Time=30704[30s:704ms]
```

VS.



	Team	Small (sec)	Big (sec)	New (sec)
1	Campers (TUM)	0.081	1.938	7.515

Questions?