SearchWorkings

Challenges in maintaing a high-performance Search-Engine written in Java

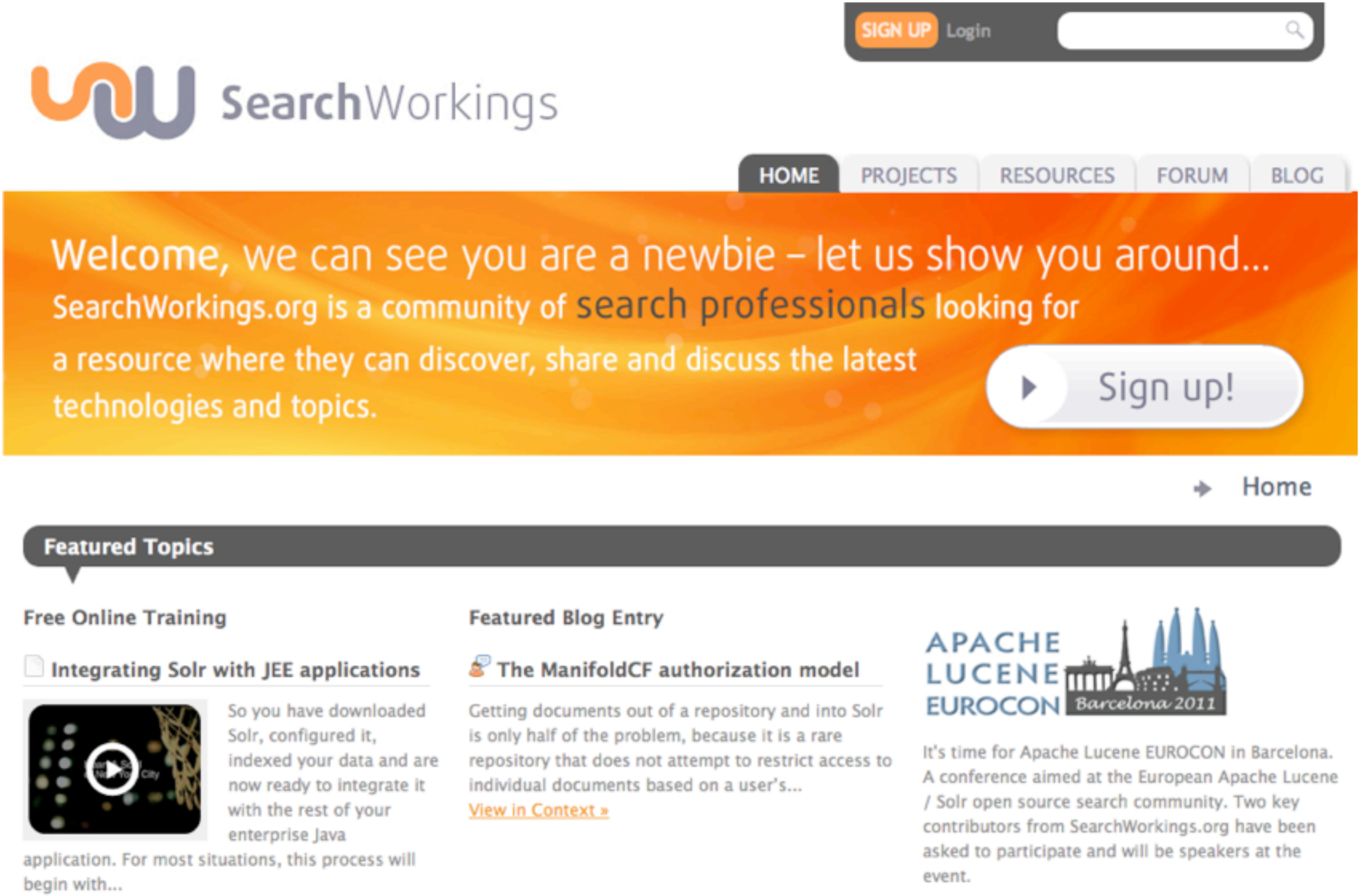
Simon Willnauer
Apache Lucene Core Committer & PMC Chair
simonw@apache.org / simon.willnauer@searchworkings.com

Who am I?

- Lucene Core Committer
- Project Management Committee Chair (PMC)
- Apache Member
- Co-Founder **BerlinBuzzwords**
- Working on **Searchworkings.org** / **Searchworkings.com**

http://www.searchworkings.org

- Community Portal targeting OpenSource Search



Welcome, we can see you are a newbie – let us show you around...

SearchWorkings.org is a community of search professionals looking for a resource where they can discover, share and discuss the latest technologies and topics.

Featured Topics

Free Online Training

Integrating Solr with JEE applications

So you have downloaded Solr, configured it, indexed your data and are now ready to integrate it with the rest of your enterprise Java application. For most situations, this process will begin with...

Featured Blog Entry

The ManifoldCF authorization model

Getting documents out of a repository and into Solr is only half of the problem, because it is a rare repository that does not attempt to restrict access to individual documents based on a user's...

View in Context »

It's time for Apache Lucene EUROCON in Barcelona. A conference aimed at the European Apache Lucene / Solr open source search community. Two key contributors from SearchWorkings.org have been asked to participate and will be speakers at the event.

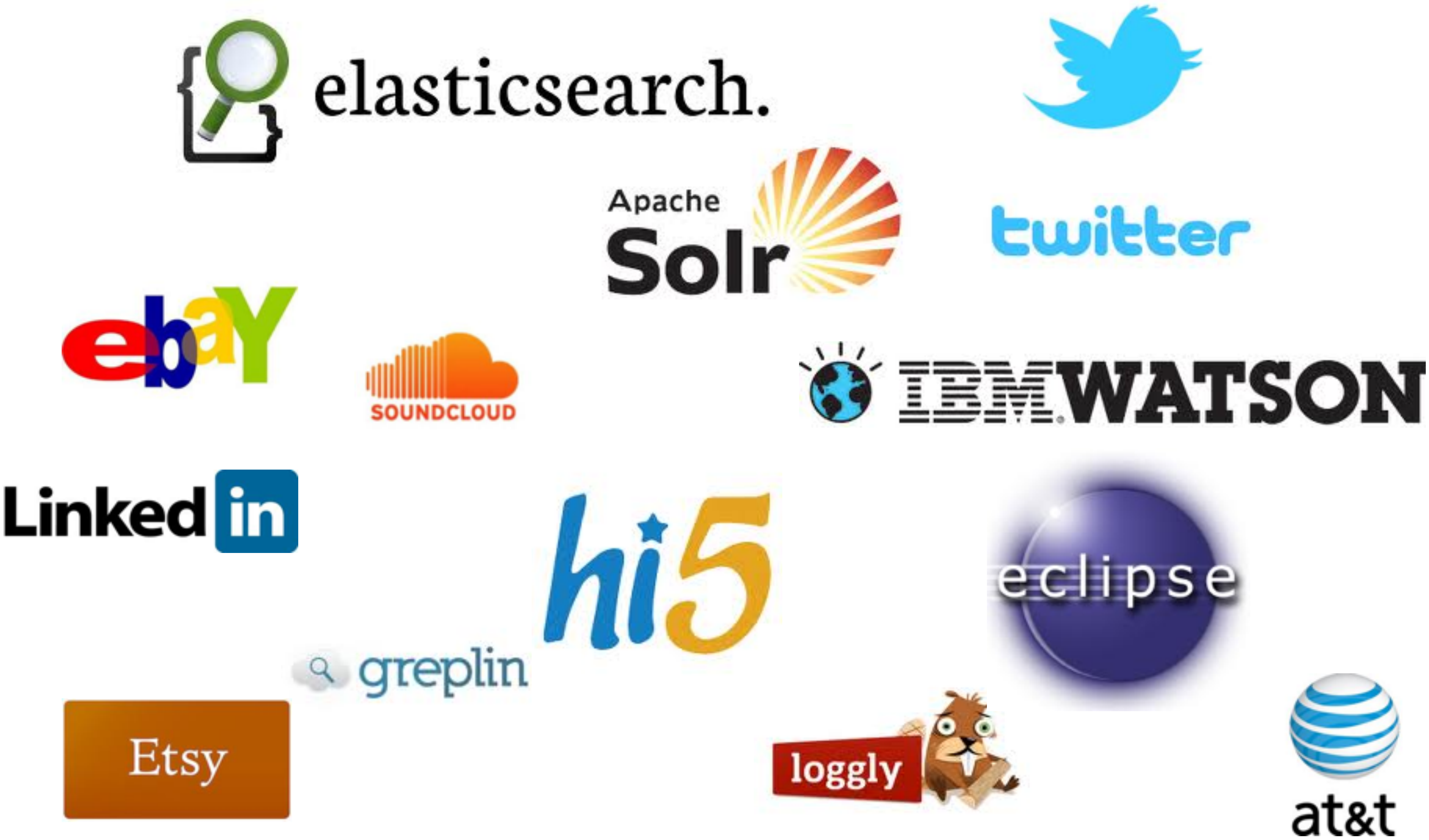
Agenda

- What search engine are you talking about?
- Its all about performance ...eerrr community
- It’s Java so its fast?
- Challenges we faced and solved in the last years
    - Testing, Performance, Concurrency and Resource Utilization
- Questions

Lets talk about Lucene

- Apache TLP since 2001
- Grandfather of projects like Mahout, Hadoop, Nutch, Tika
- Used by thousands of applications world wide
- Apache 2.0 licensed
- Core has Zero-Dependency
- Developed and Maintained by Volunteers

Who uses it?



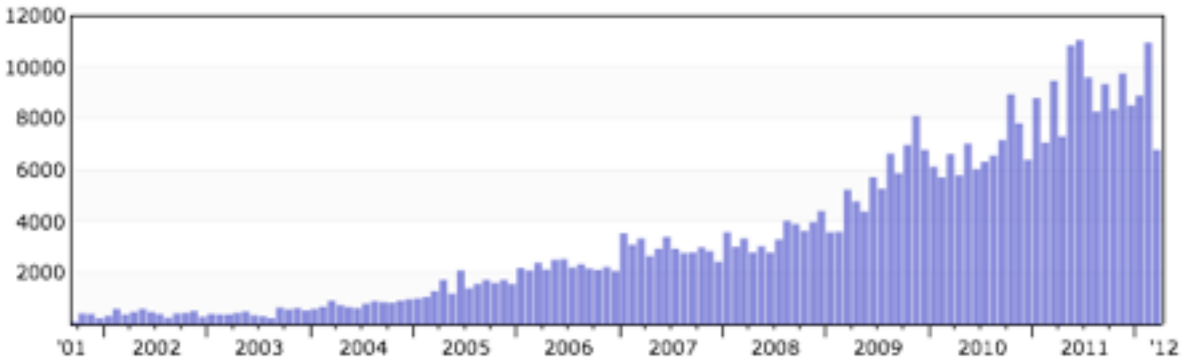
Notice: All information above are taken from publicly available resources mentioning the use of Apache Lucene, ElasticSearch or Apache Solr, The mentioned companies or products are randomly selected without any particular importance. All marks mentioned may be trademarks or registered trademarks of their respective owners.

Just a search engine - so what’s the big deal?

- True - Just software!
- Massive community - with big expectations
- Mission critical for lots of companies
- End-user expects instant results independent of the request complexity
- New features often require major changes
- Our contract is trust - we need to maintain trust!

Trust & Passion

- ~ 30 committers (~ 10 active, some are payed to work on Lucene)
- All technical communication are public (JIRA, Mailinglist, IRC)
- Consensus is king!
- No lead developer or architect
- No stand-ups, meetings or roadmap
- Up to 10k mails per month
- No passion, no progress!
- The Apache way: **Community over Code**



Enough about Community - lets talk about code!

Java
Lucene
ORACLE
Java
Lucene
ORACLE

yesterday today tomorrow

We are working in Java so....

- No need to know the machine & your environment
- Use JDK Collections, they are fast
- Short Lived Objects are Free
- Great Data-Structures are Mutable
- Concurrency means *Speed*
- IO is trivial
- Method Calls are fast - there is a JIT, no?
- Unicode is there and it works

Mechanical Sympathy (Jackie Steward / Martin Thompson)

“The most amazing achievement of the computer software industry is its continuing cancellation of the steady and staggering gains made by the computer hardware industry.” Henry Peteroski

Know your environment at scale - an example

- For Lucene **Term→ Posting-List** lookups are crucial
- Speed is everything, we can do up to **600k** key/value lookups per second (single box)
- We deal with Strings mainly (full Unicode Support)
- The main data-structure is a Sorted-Dictionary
- No internal caches anymore
- Large amount of concurrent reads

The upper bound - not uncommon to reach!

- 274 Billion Unique Terms (Java Strings - 2 byte per Char)
- One entry (term, postingpointer, docFreq)
- At least one additional object per entry
- (numTerms * (objectHeader + postingspointer + docFreq + objectHeader + reference + *average num Chars per String*))
  - (10^9 * (8byte + 8byte + 4byte + 8byte + 8byte + 10 bytes)) = 10^9 * 46byte ~ 44GB
- You might have enough Heap Space, but how is your **GC** gonna like that? -> Remember 2 *10^9 Objects

Where to focus on?

Cost of a Monitor / CAS

CPU Cache Utilization

Space/CPU Utilization

Concurrency

Need of mutability

Cost & Need of a Multiple Writers Model

JVM memory allocation

Can we allow stack allocation?

Impact on GC

Can we specialized a data-strucutres

Amount of Objects (Long & Short Living)

Compression

Any exploitable data properties

Do we need 2 bytes per Character?

What we focus on...

Materialized Data structures for Java HEAP

Write, Commit, Merge

Prevent False Sharing

Space/CPU Utilization

Concurrency

Single Writer - Multiple Readers

Write Once & Read - Only

No Java Collections where scale is an issue

Guarantee continuous memory allocation

Impact on GC

Finite State Transducers / Machines

Data Structures with Constant number of objects

MemoryMap | NIO

Compression

Strings can share prefix & suffix

Exploit FS / OS Caches

Materialize strings to bytes

UTF-8 by default or custom encoding

Is all this necessary?

- Yes & No - it all depends on finding the hotspots
- Measure & Optimize for you use-case.
    - Data-structures are not general purpose (like the don’t support deletes)
- Follow the 80 / 20 rule
- Enforce Efficiency by design
    - **Java Iterators** are a good example of how not to do it!
- Remember you OS is highly optimized, make use of it!

Enough high level - concrete problems please!

- Challenge: Idle is no-good!
- Challenge: One Data-Structure to rule them all?
- Challenge: How how to test a library
- Challenge: What’s needed for a 20000% performance improvement

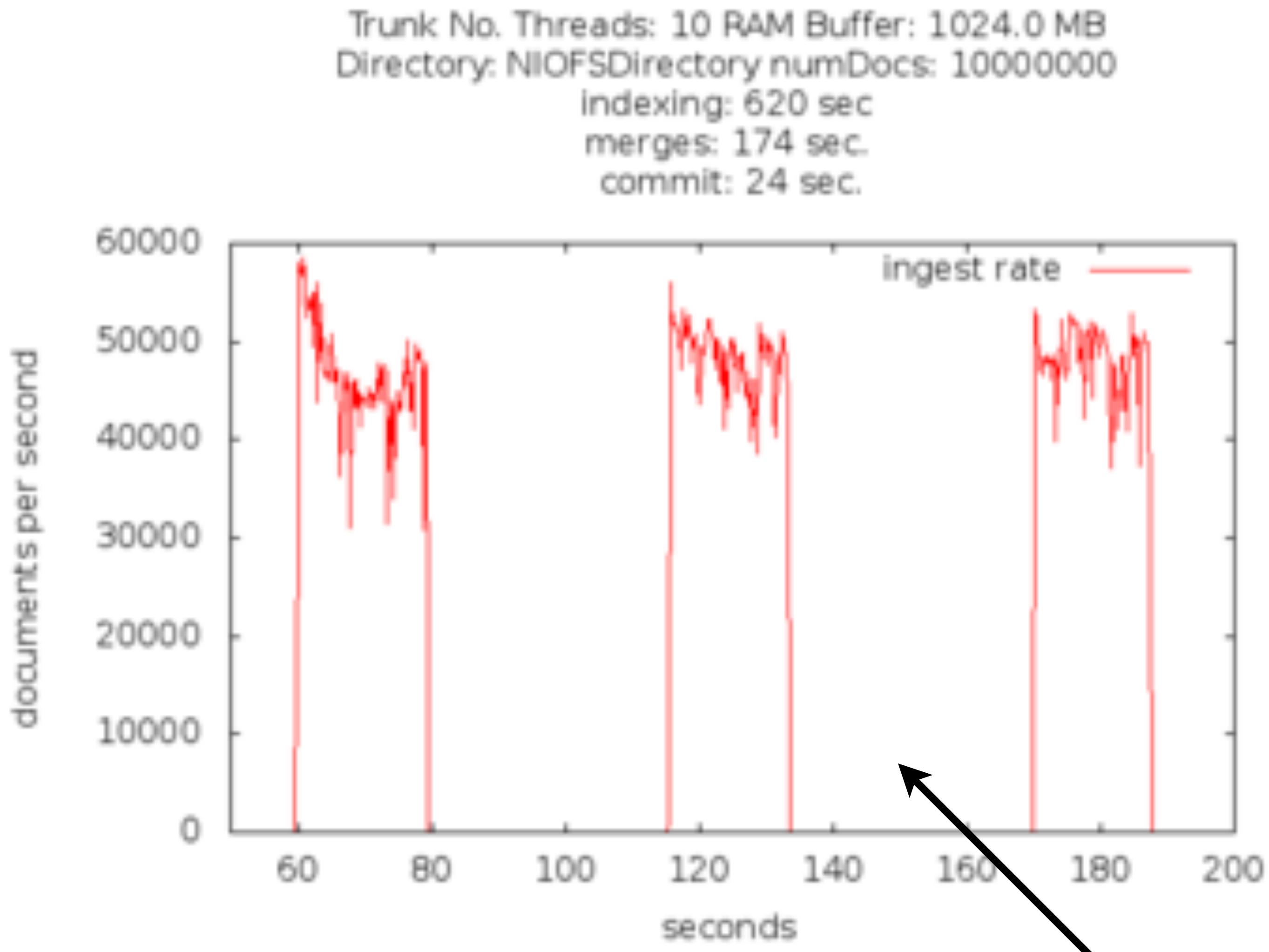
Challenge: Idle is no-good

- Building an index is a CPU & IO intensive task
- Lucene is full of indexes (thats basically all it does)
- Ultimate Goal is to scale up with CPUs and saturate IO at the same time

Don’t go crazy!

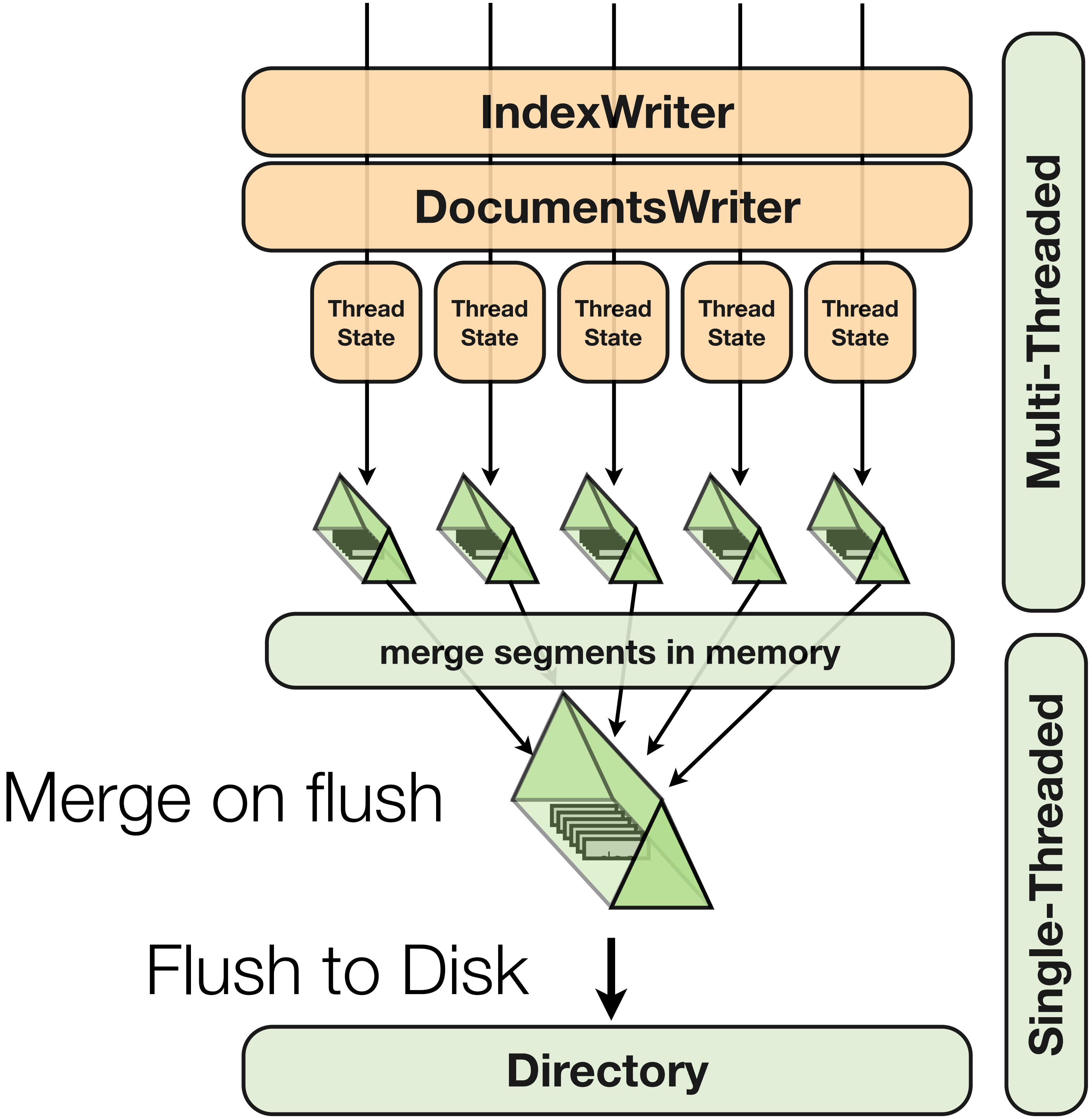
- Keep your code complexity in mind
    - Other people might need to maintain / extend this

Here is the problem



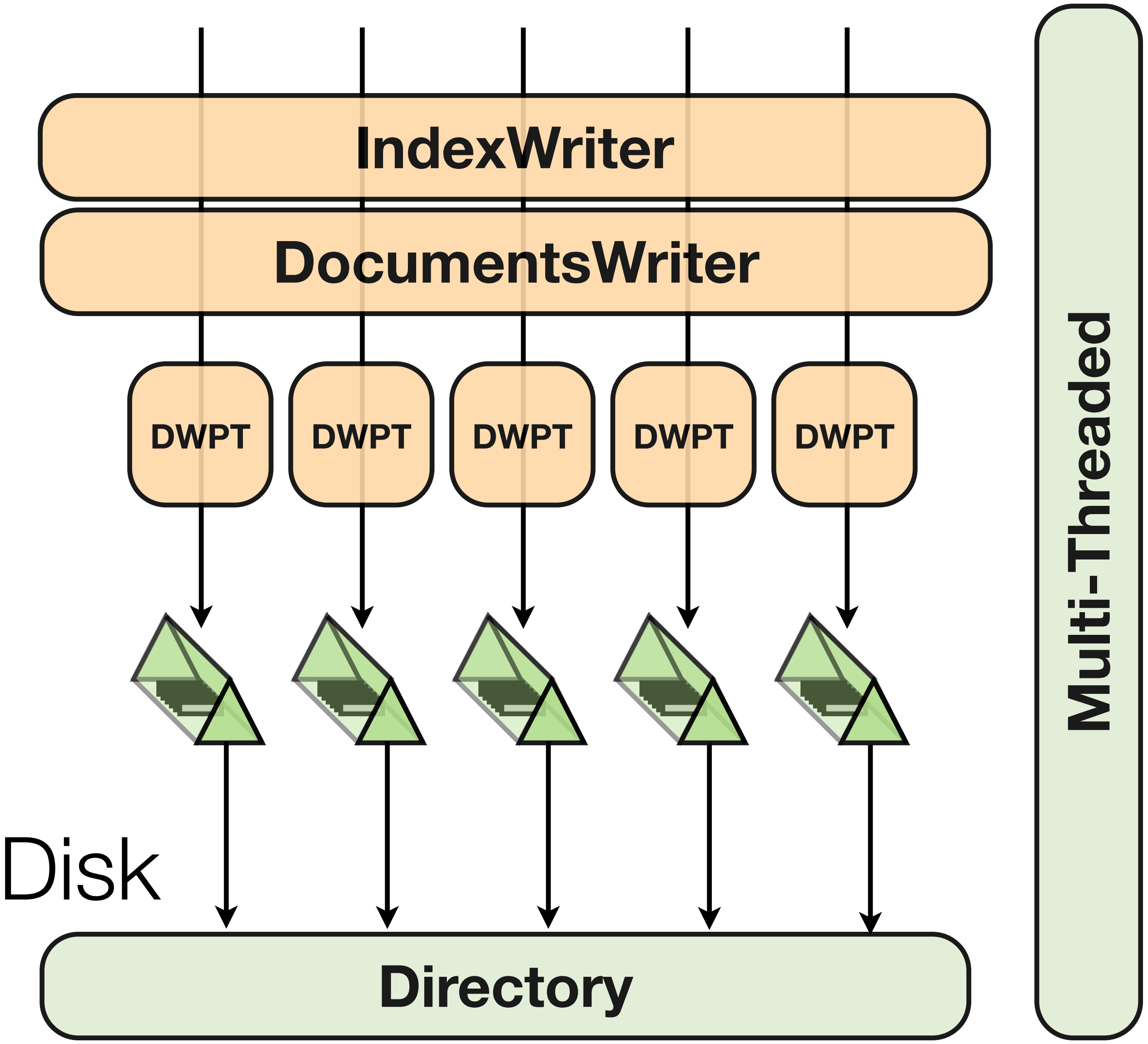
WTF?

A closer look...



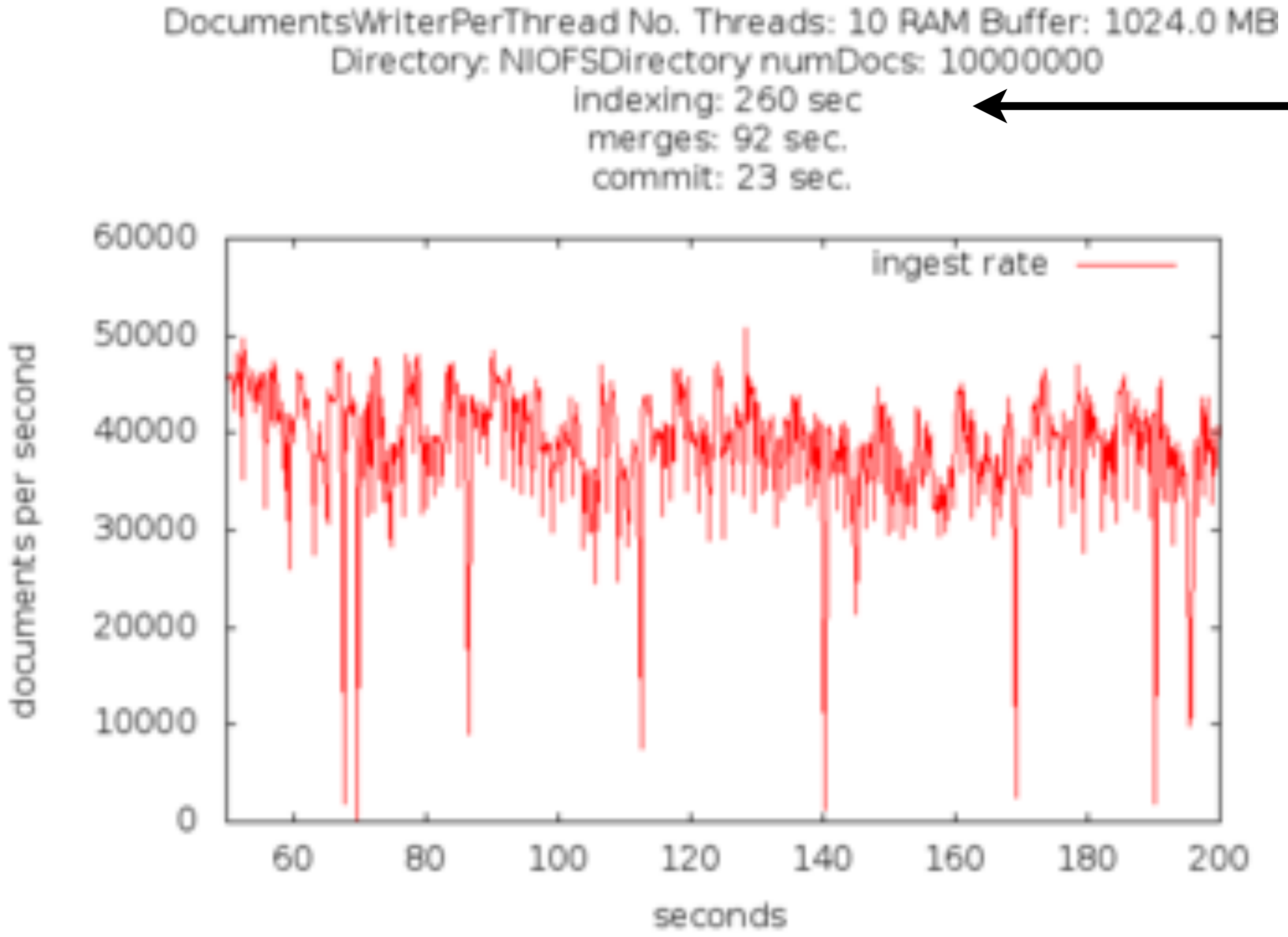
Answer: it gives you threads a break and it's having a drink with your slow-as-s**t IO System

Our Solution



Flush to Disk

The Result



DocumentsWriterPerThread No. Threads: 10 RAM Buffer: 1024.0 MB
Directory: NIOFSDirectory numDocs: 10000000
indexing: 260 sec
merges: 92 sec.
commit: 23 sec.

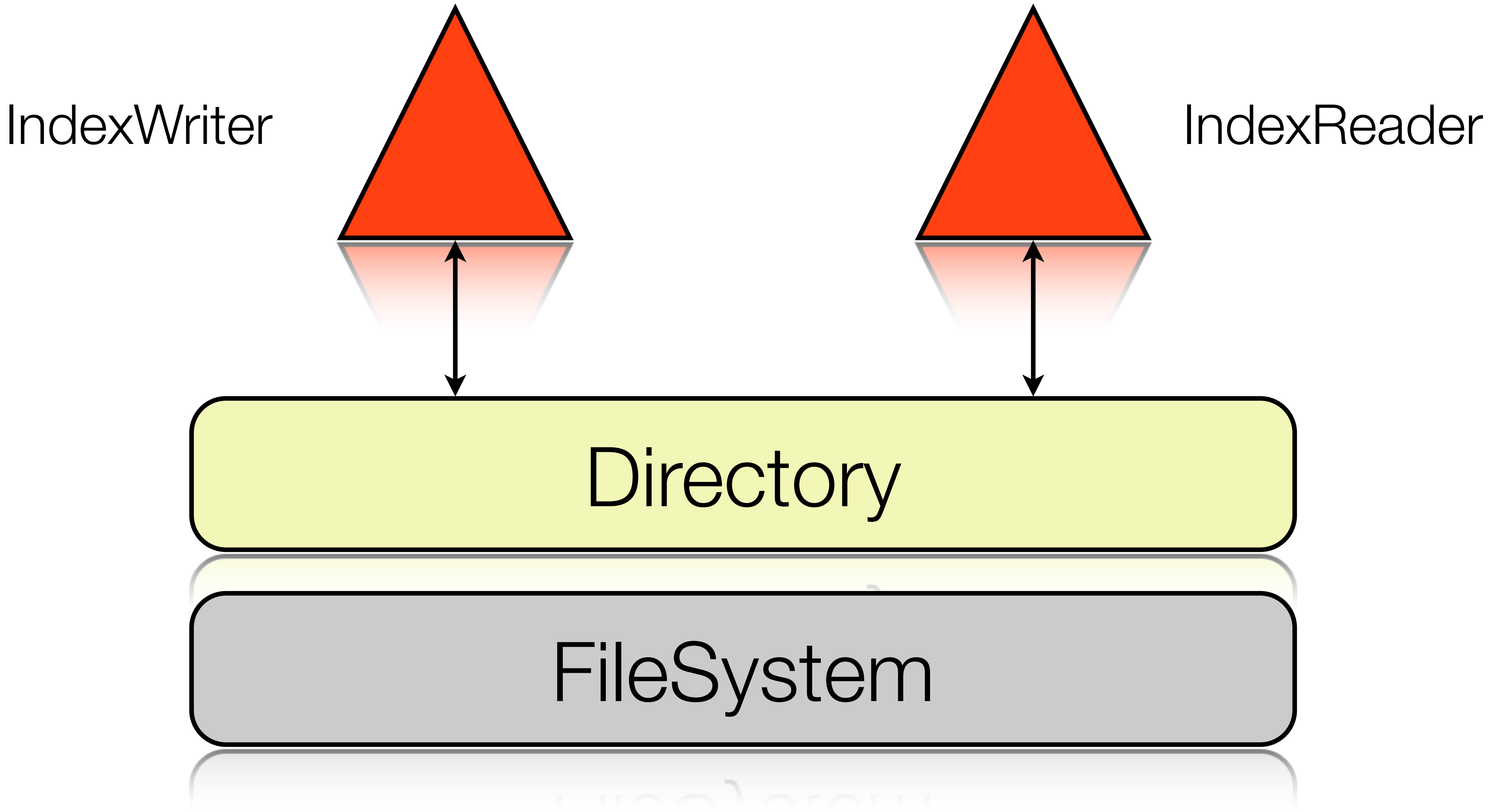
vs. 620 sec on 3.x

Indexing Ingest Rate over time with Lucene 4.0 & DWPT Indexing 7 Million 4kb wikipedia documents

Challenge: One Data-Structure to Rule them all?

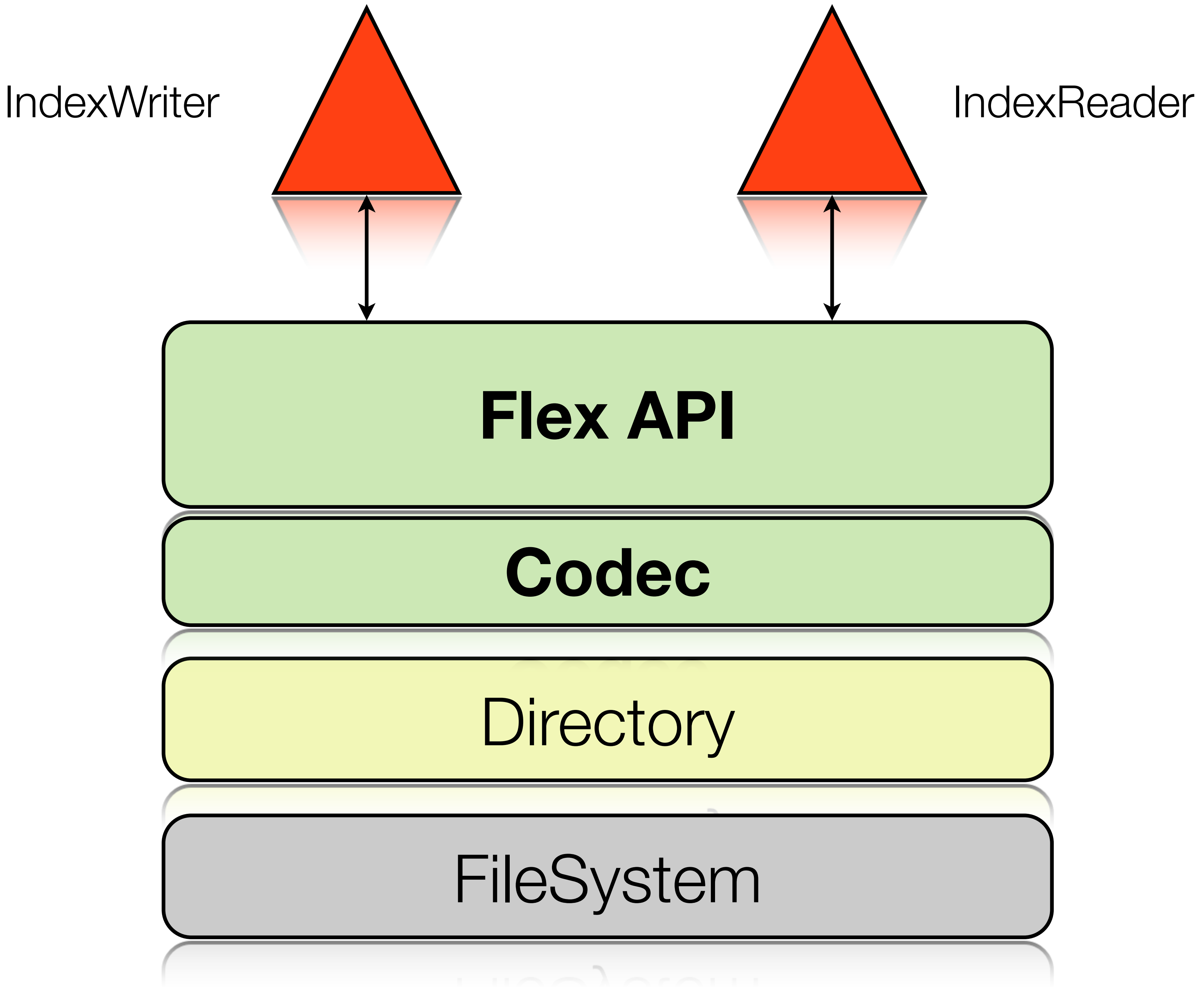
- Like most other systems writing datastructures to disk Lucene didn't expose it for extension
- Major problem for researchers, engineers who know what they are doing
- Special use-cases need special solutions
  - Unique ID Field usually is a 1 to 1 key to document mapping
  - Holding a posting list pointer is a wasteful
  - Term lookup + disk seek vs. Term lookup + read
- Research is active in this area (integer encoding for instance)

10000 ft view

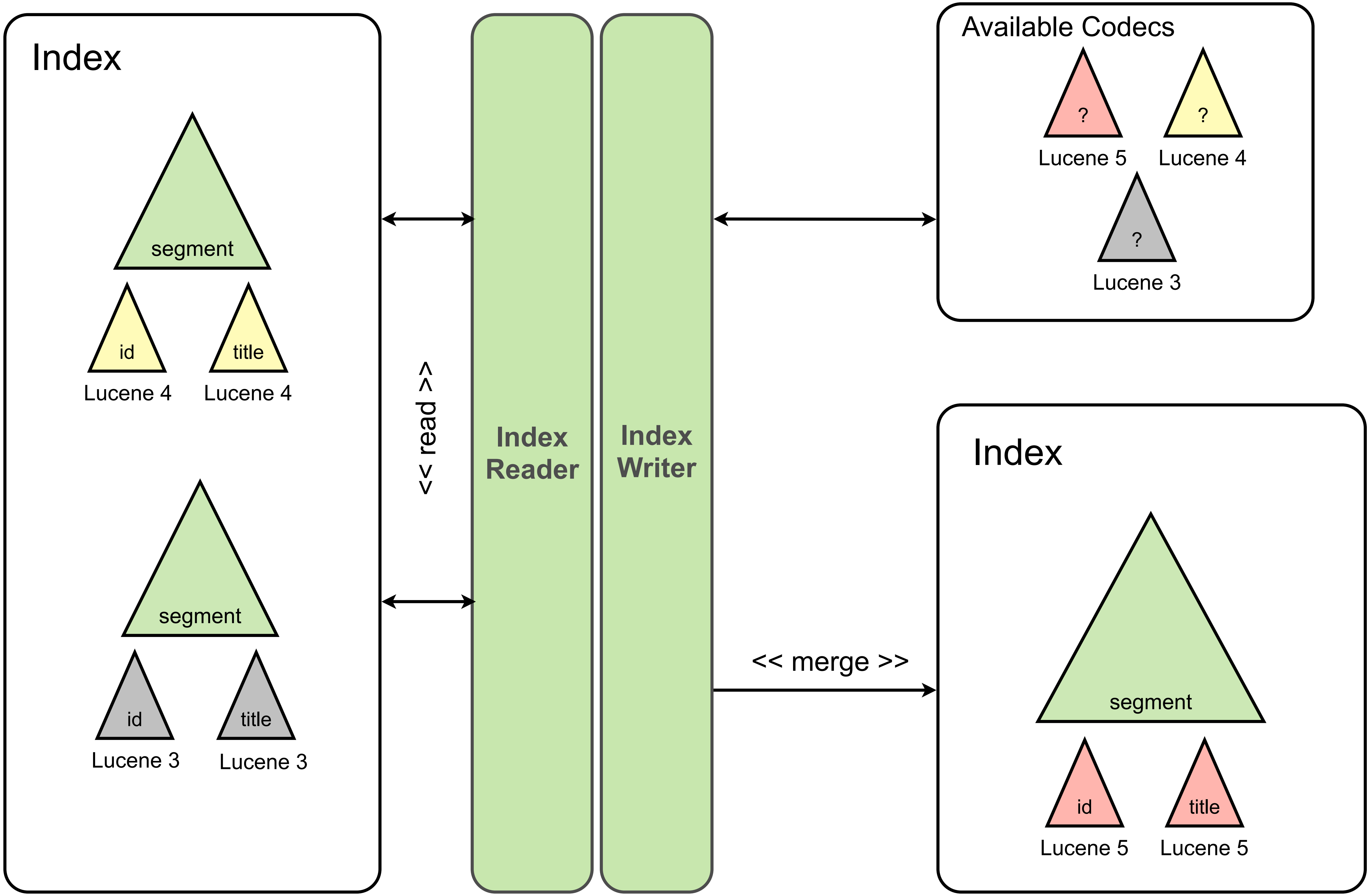


24

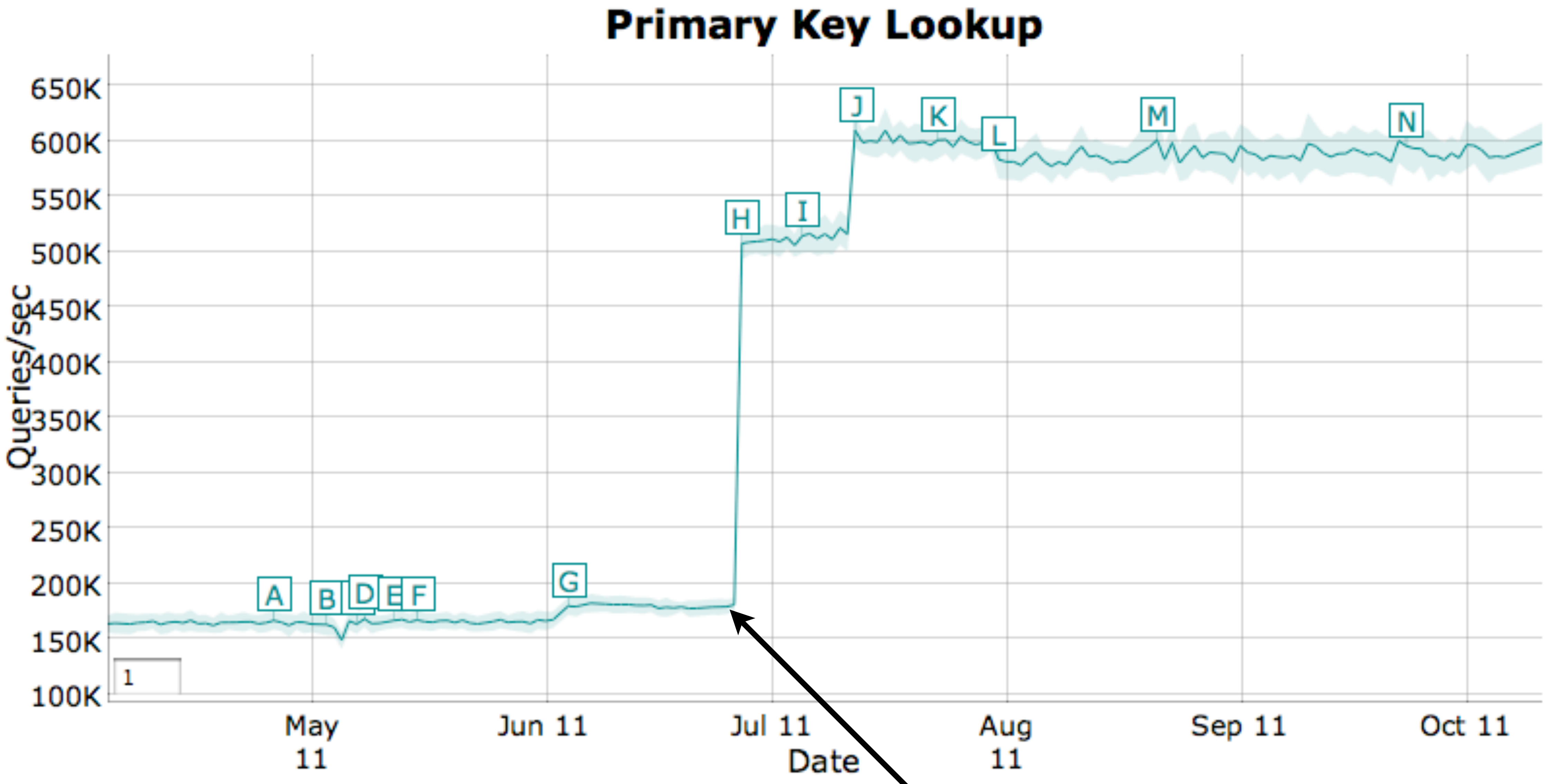
Introducing an extra layer



For Backwards Compatibility you know?

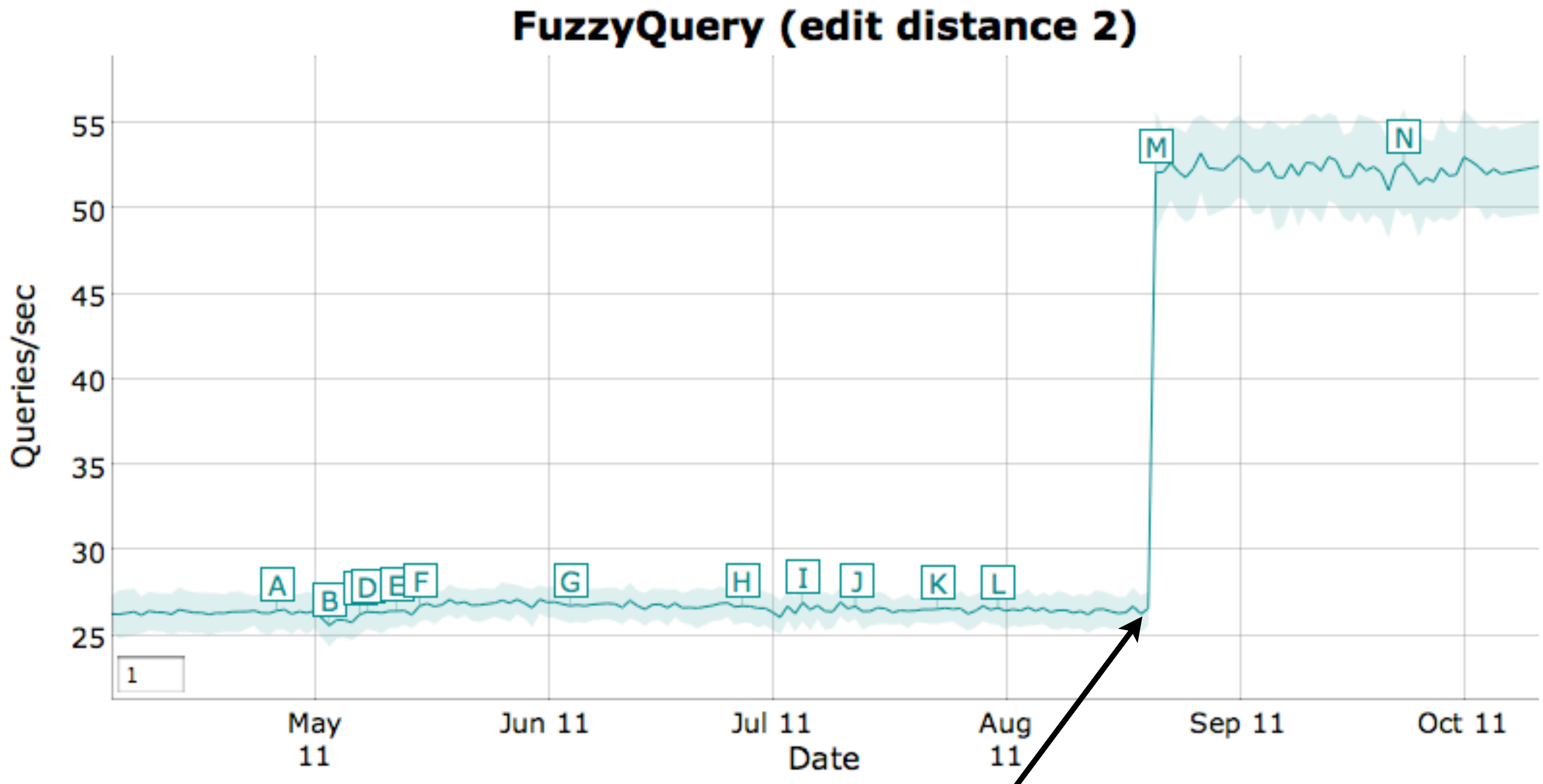


Using the right tool for the job..



Switching to Memory PostingsFormat

Using the right tool for the job..



Switching to BlockTreeTermIndex

Challenge: How to test a library

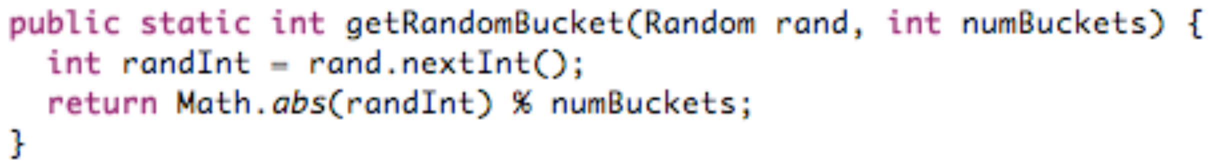
- A library typically has:
  - lots of interfaces & abstract classes
  - tons of parameters
  - needs to handle user input gracefully
- Ideally we test all combinations of Interfaces, parameters and user inputs?
- Yeah - right!

What’s wrong with Unit-Test

- Short answer: Nothing!
- But...
  - 1 Run == 1000 Runs? (only cover regression?)
  - Boundaries are rarely reached
  - Waste of CPU cycles
  - Test usually run against a single implementation
  - How to test against the full Unicode-Range?

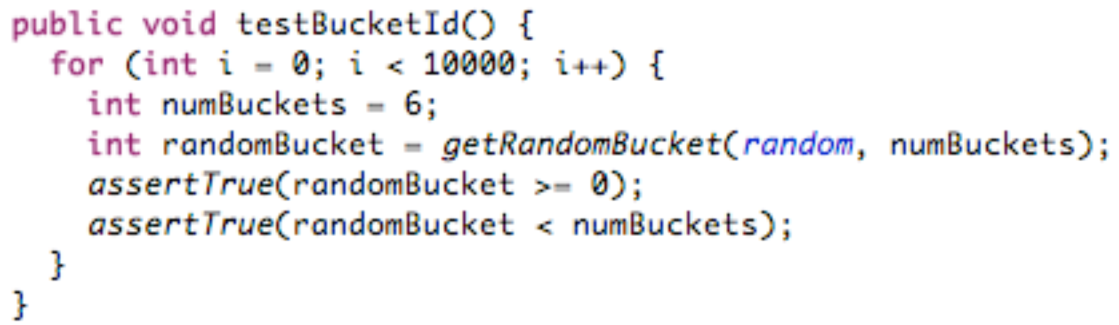
An Example

The method to test:



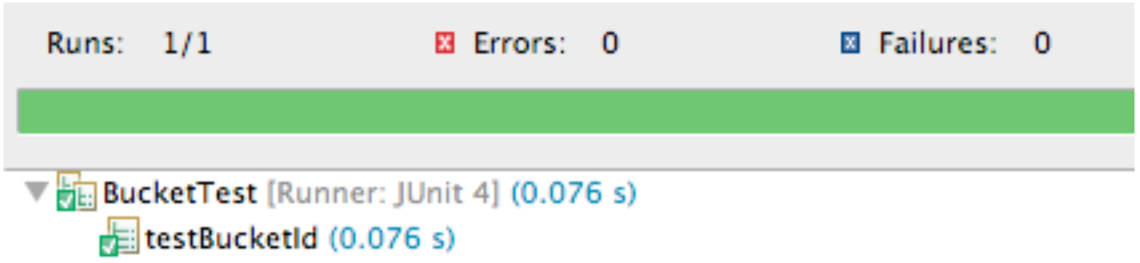
```
public static int getRandomBucket(Random rand, int numBuckets) {
  int randInt = rand.nextInt();
  return Math.abs(randInt) % numBuckets;
}
```

The test:



```
public void testBucketId() {
  for (int i = 0; i < 10000; i++) {
    int numBuckets = 6;
    int randomBucket = getRandomBucket(random, numBuckets);
    assertTrue(randomBucket >= 0);
    assertTrue(randomBucket < numBuckets);
  }
}
```

The result:

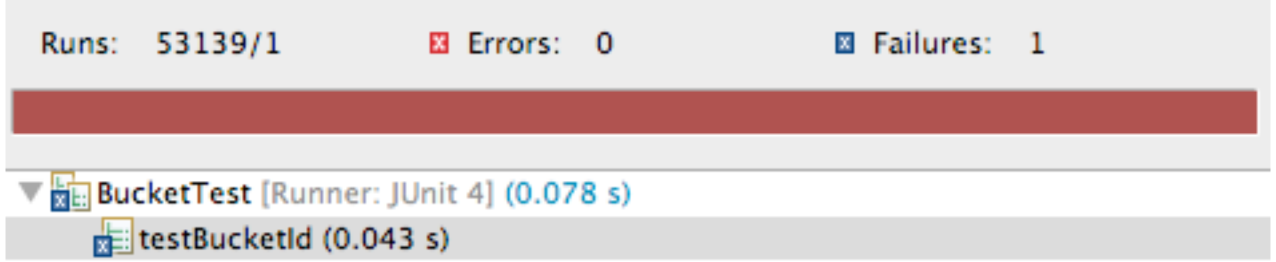


```
Runs:   1/1          ☒ Errors:   0          ☒ Failures:   0

▼ BucketTest [Runner: JUnit 4] (0.076 s)
    testBucketId (0.076 s)
```

Can it fail?

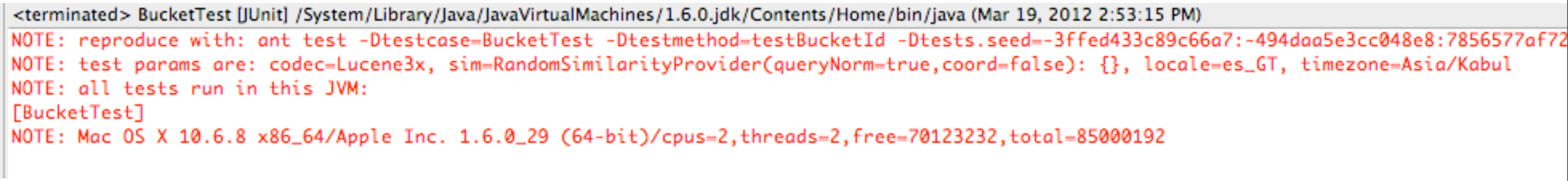
It can! ...after 53139 Runs



```
Runs:   53139/1        ☒ Errors:   0        ☒ Failures:   1

▼ BucketTest [Runner: JUnit 4] (0.078 s)
      testBucketId (0.043 s)
```

- Boundaries are everywhere
- There is no positive value for Integer.MIN
- But how to repeat / debug?



```
<terminated> BucketTest [JUnit] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Mar 19, 2012 2:53:15 PM)
NOTE: reproduce with: ant test -Dtestcase=BucketTest -Dtestmethod=testBucketId -Dtests.seed=-3ffed433c89c66a7:-494daa5e3cc048e8:7856577af72
NOTE: test params are: codec=Lucene3x, sim=RandomSimilarityProvider(queryNorm=true,coord=false): {}, locale=es_GT, timezone=Asia/Kabul
NOTE: all tests run in this JVM:
[BucketTest]
NOTE: Mac OS X 10.6.8 x86_64/Apple Inc. 1.6.0_29 (64-bit)/cpus=2,threads=2,free=70123232,total=85000192
```

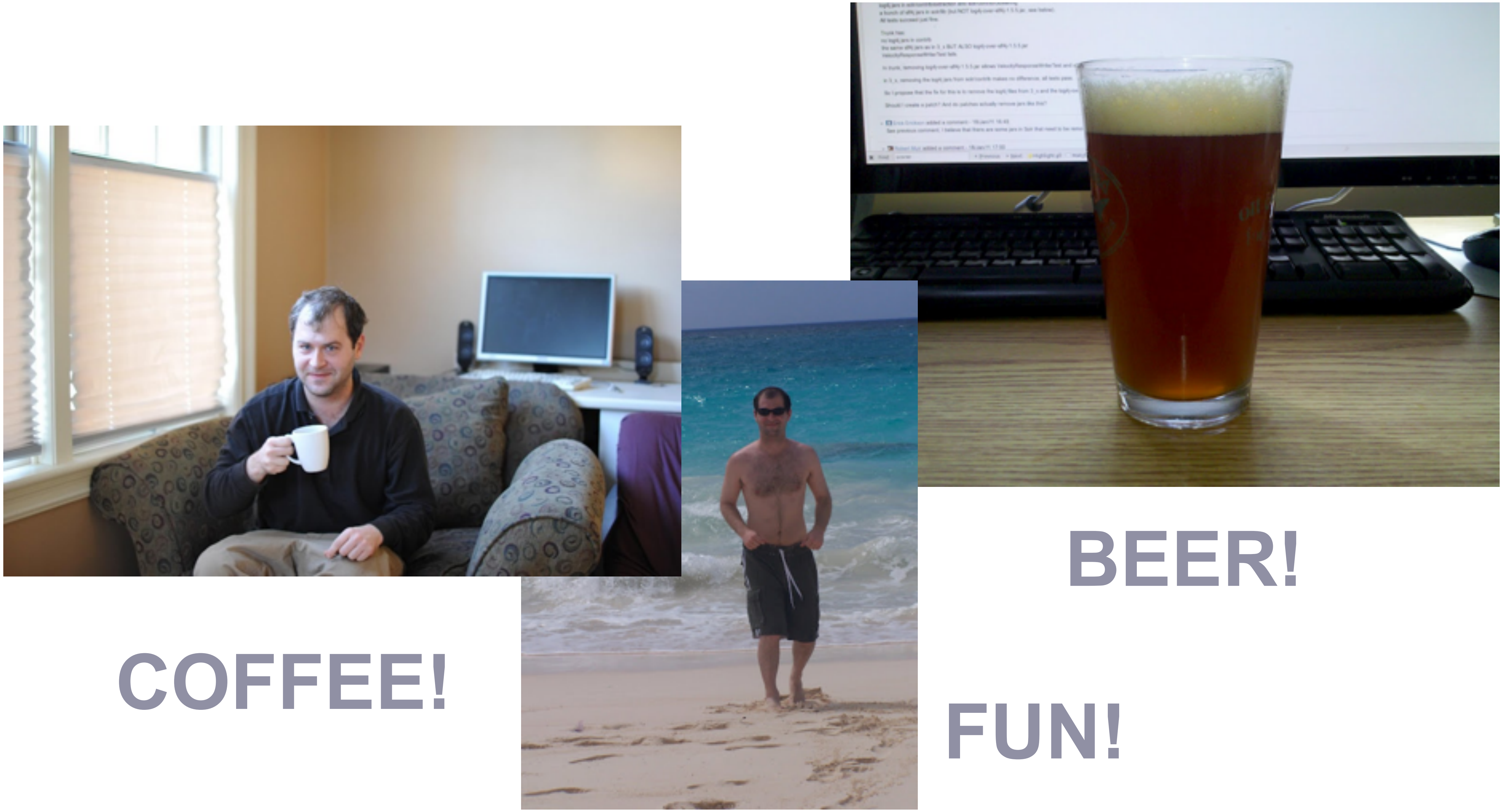
Solution: A Randomized UnitTest Framework

- Disclaimer: this stuff has been around for ages - not our invention!
- Random selection of:
  - Interface Implementations
  - Input Parameters like # iterations, # threads, # cache sizes, intervals, ...
  - Random Valid Unicode Strings (Breaking JVM for fun and profit)
  - Throttling IO
  - Random Low Level Data-Strucutures
  - And many more...

Make sure your unit tests fail - eventually!

- Framework is build for Lucene
  - Currently factored out into a general purpose framework
  - Check it out on: https://github.com/carrotsearch/randomizedtesting
- Wanna help the Lucene Project?
  - Run our tests and report the failure!

Challenge: What’s needed for a 20k% Performance improvement.



BEER!

COFFEE!

FUN!

The Problem: Fuzzy Search

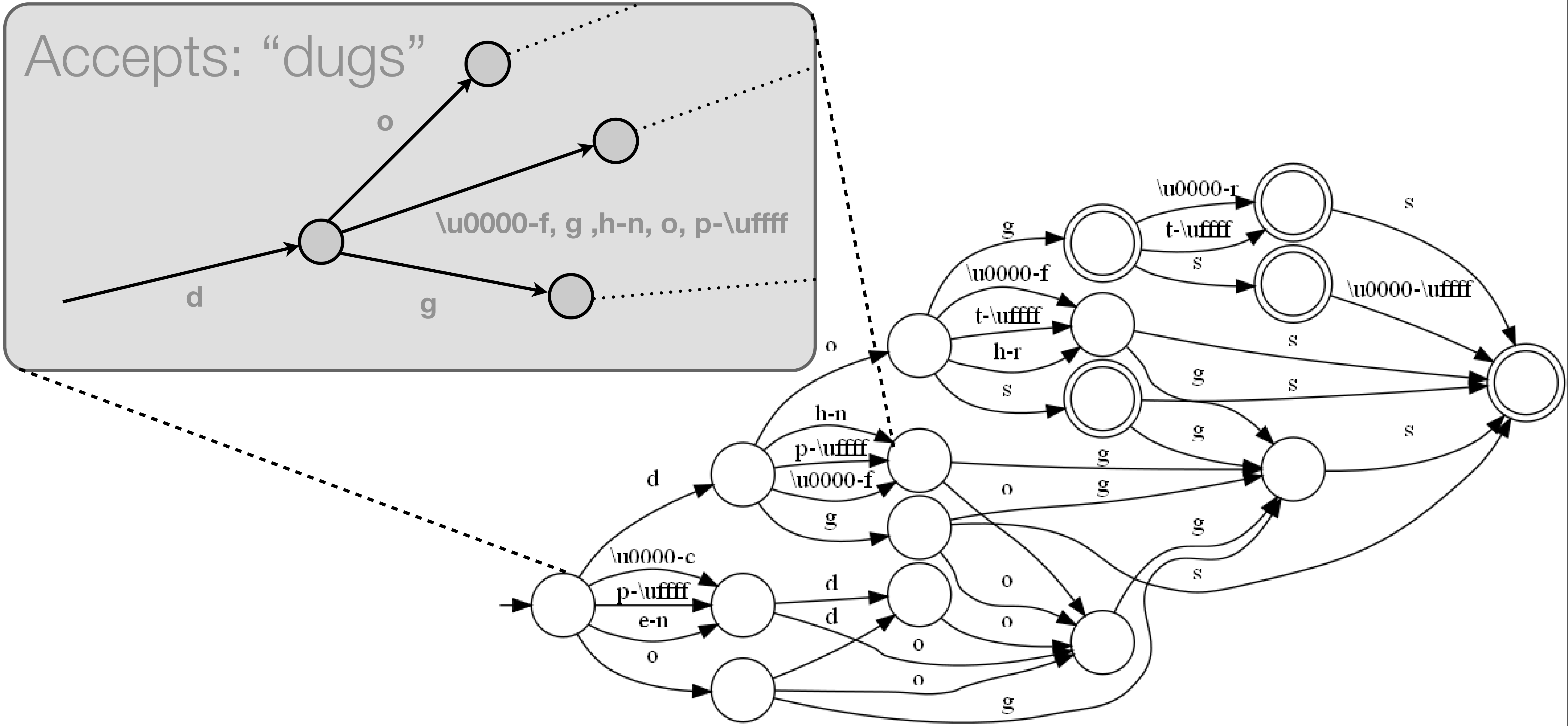
- Retrieve all documents containing a given term within a Levenshtein Distance of <= 2
- **Given**: a sorted dictionary of terms
- **Trivial Solution**: Brute Force - filter(terms, LD(2, queryTerm))
- **Problem**: it's damn slow!
  - O(t) terms examined, t=number of terms in all docs for that field. Exhaustively compares each term. We would prefer $O(\log_2 t)$ instead.
  - $O(n^2)$ comparison function, n=length of term. Levenshtein dynamic programming. We would prefer O(n) instead.

Solution: Turn Queries into Automatons

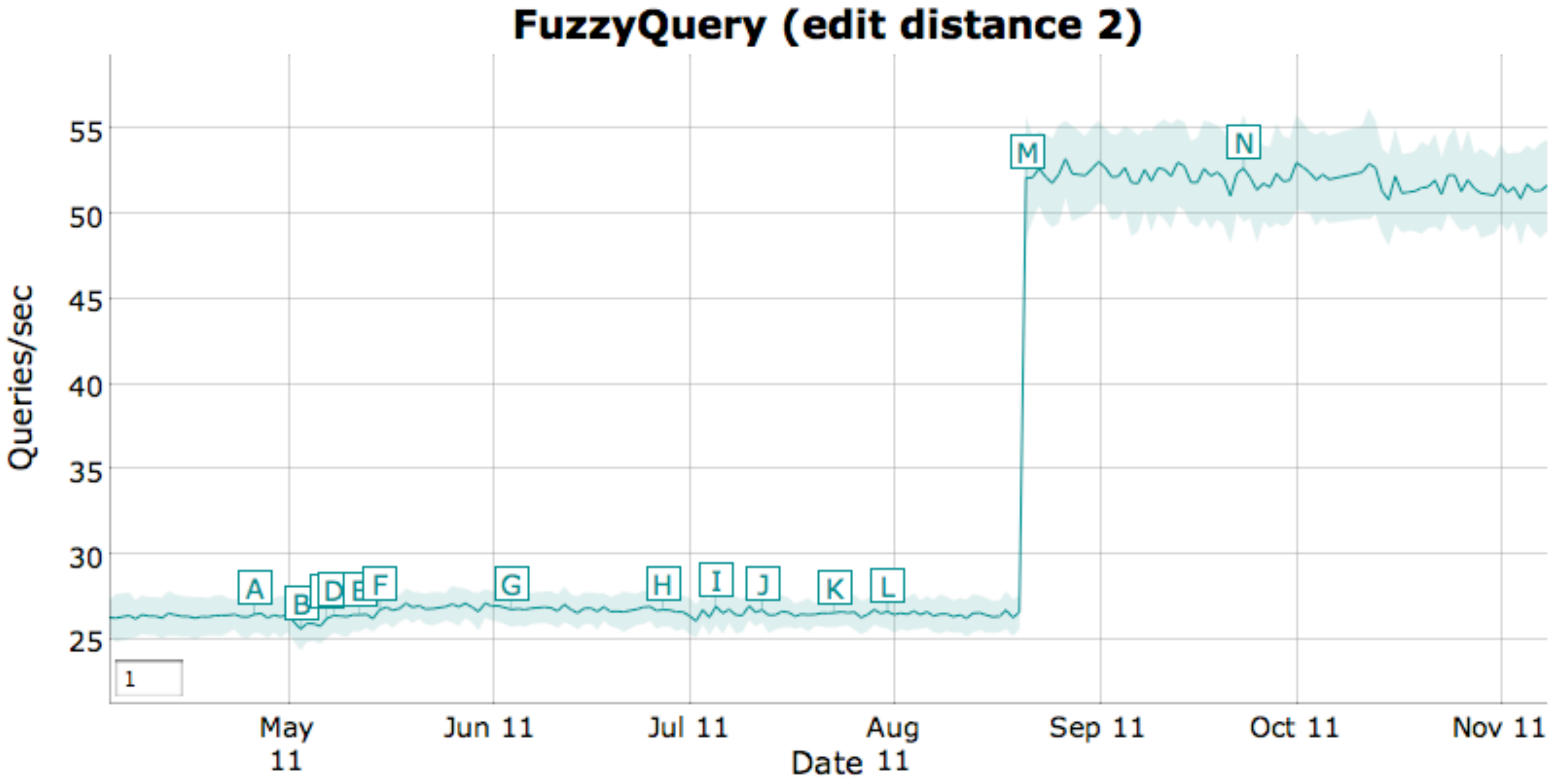
- Read a crazy Paper about building Levenshtein Automaton and implement it. (sounds easy - right?)
- Only explore subtrees that can lead to an accept state of some finite state machine.
- AutomatonQuery traverses the term dictionary and the state machine in parallel
- Imagine the index as a state machine that recognizes Terms and transduces matching Documents.
  - AutomatonQuery represents a user’s search needs as a FSM.
  - The intersection of the two emits search results

Solution: Turn Queries into Finite State Machines

Example DFA for "dogs" Levenshtein Distance 1


Accepts: "dugs"

Turns out to be a massive improvement!



In Lucene 3 this is about 0.1 - 0.2 QPS

Berlin Buzzwords 2012

- Conference on High-Scalability, NoSQL and Search
- 600+ Attendees, 50 Sessions, Trainings etc.
- Discount Code: 10% with “**BB12-33DG” valid March 21st - March 25th**

Questions anybody?

?