Sonntag, 30. Mai 2010

Before I begin, let me just … get something off my chest.

My friends and I have BEEN here for two days now. And…

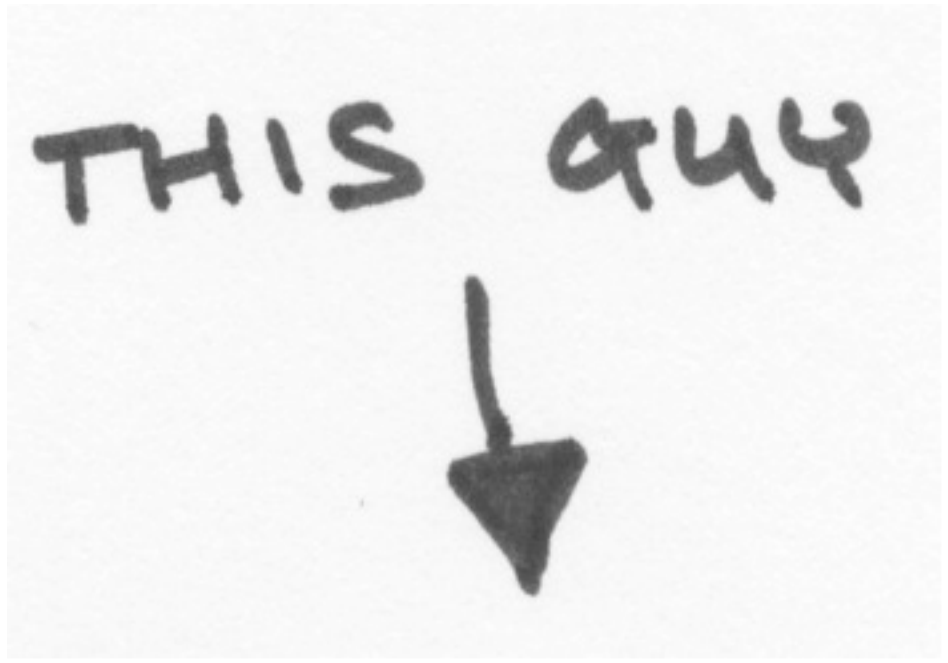
You polish guys… you have it great.

Your food… it is sooo.

Sonntag, 30. Mai 2010

Hi all.

Florian “Flöre” Hanke



My name is Florian Hanke and I’m going to tell you a little story on
how I wrote a search engine.
In Ruby.

And also tell you about a few surprising discoveries and creative solutions I needed to apply.
And creative in this context means … dirty.

Once upon a time, I was looking for a job.

Almost the first thing that I found was this:

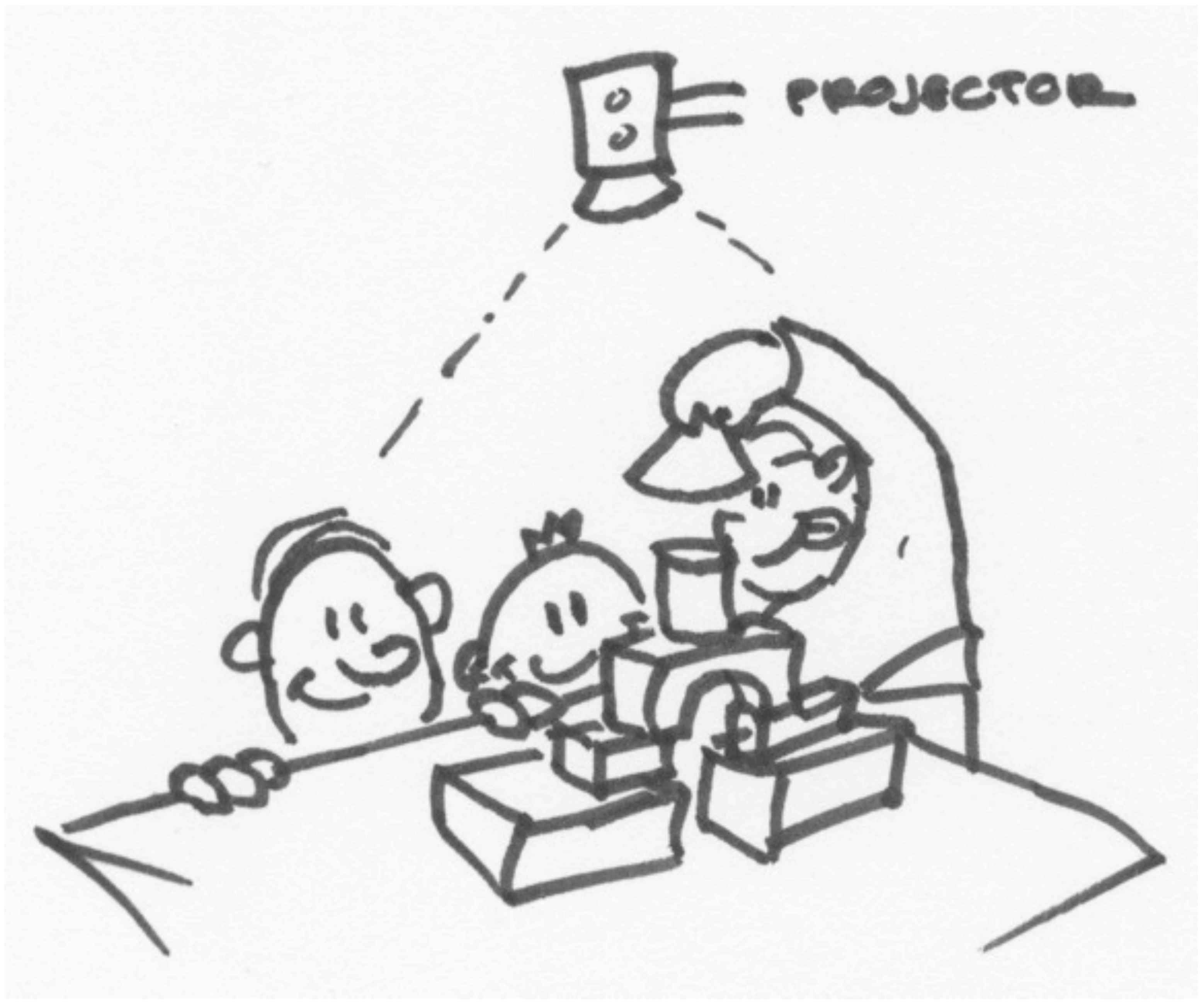
That was the actual wording, “Search Wizard needed”.

It was incredibly intriguing, and as soon as I saw it I knew:
I needed to do it.

But first, let me tell you a little bit about myself.

I’ve done quite a few computer sciency things in my life.

For example



Sonntag, 30. Mai 2010

A multiuser 3D building table.

I worked on the brasilian space program, writing code for microcontrollers.

And as a diploma project I created a visual framework builder,
where one could click together frameworks using a visual editor.

Sonntag, 30. Mai 2010

And some other projects.

But why am I telling you all this?

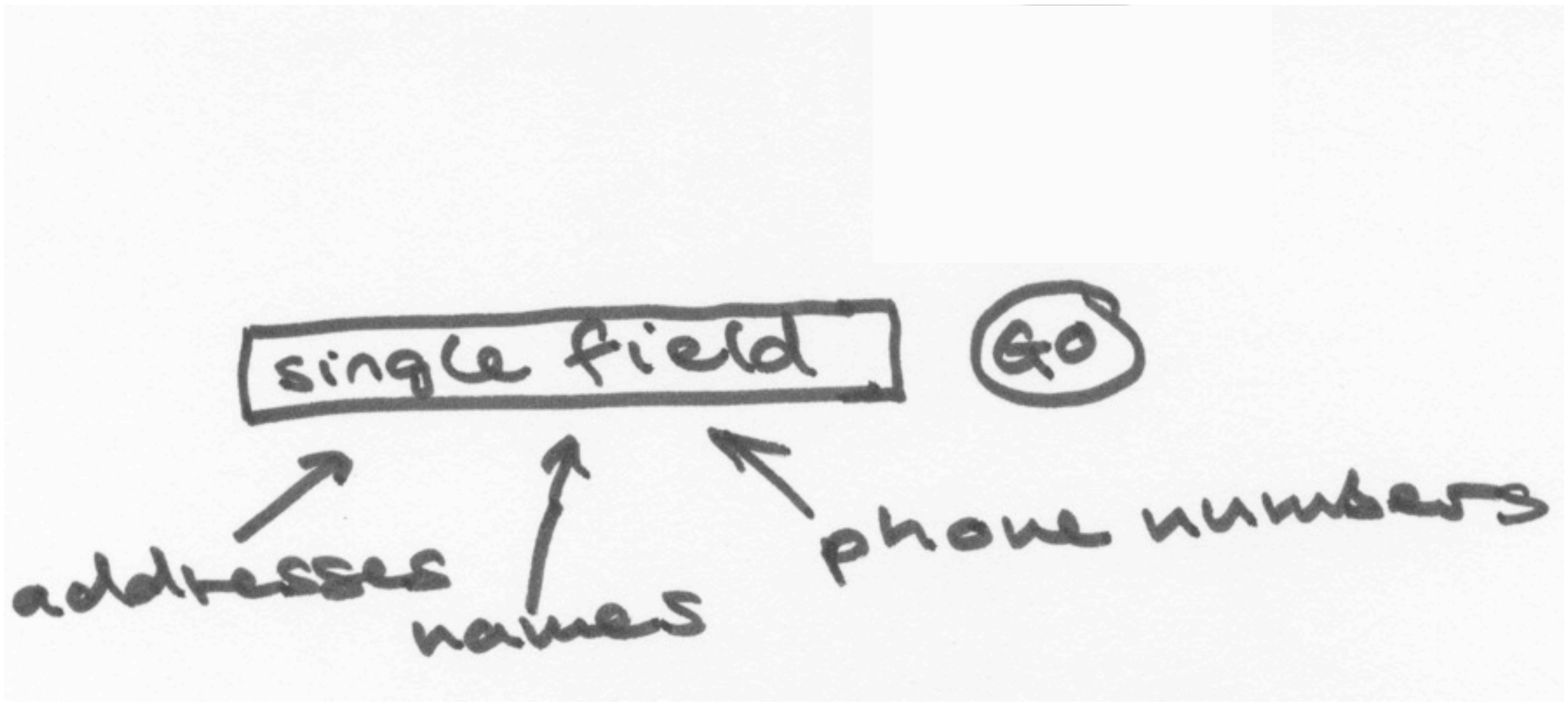
I’m telling you this, because none of my projects had anything to do with speed.

So as far as magical search skills go:
I am Harry Potter, living NOT in Hogwarts, but under the staircase.

But hey, a search engine in Ruby just proved to be too cool to pass up.

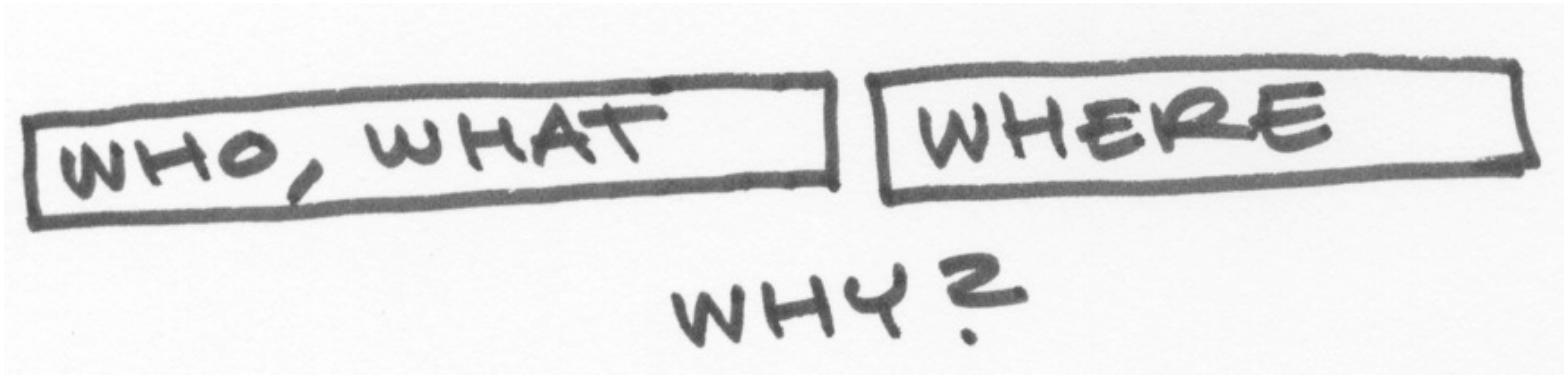
Sonntag, 30. Mai 2010

So I got the job.



What THEY wanted to do was create a single field address/phone number search.

WHY is this special at all? It is special because…



Sonntag, 30. Mai 2010

ALL the phone searches in the world are structured like this:

Two fields:
Who, what – and – where. But why?
I was really wondering about that.

I was wondering about it BECAUSE…

Google in Norway for example had already introduced a single field address search.

I think it's not working anymore, though.

So I went to two phone search specialists to get an answer.
One in Germany and one in Switzerland, where I come from.

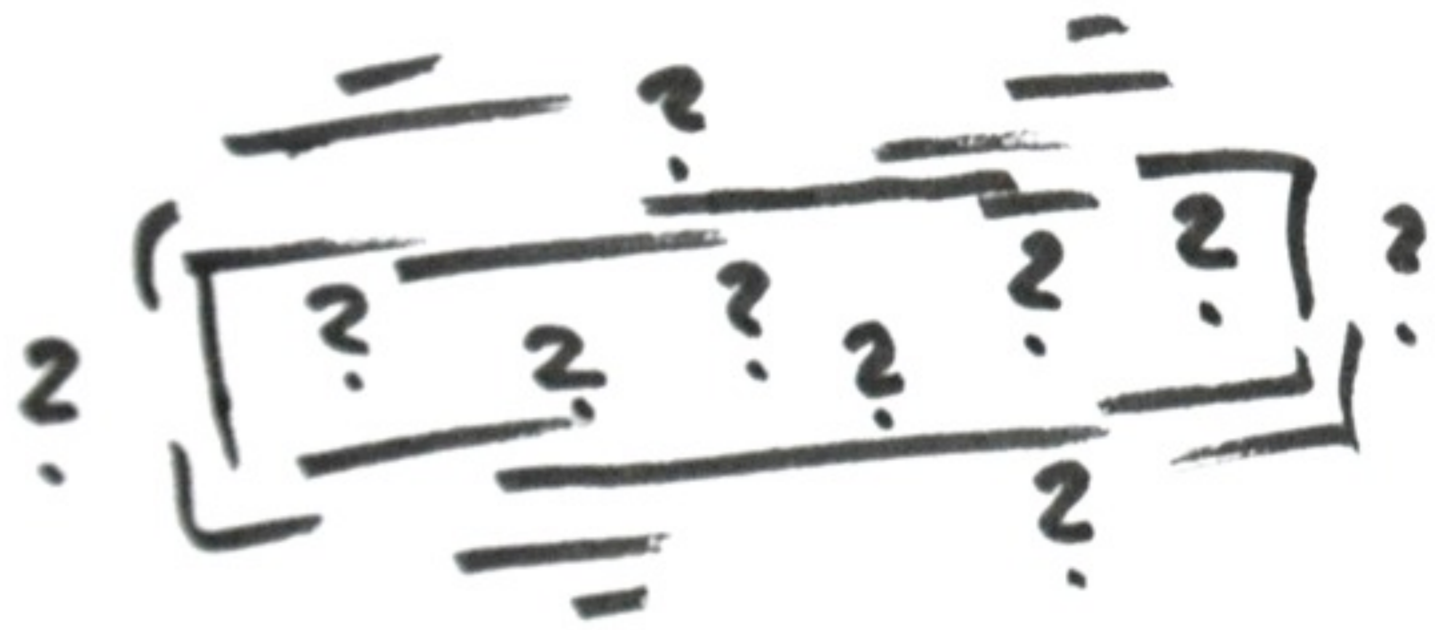
They both told me the reason is because everyone else does it like that and people are used to the two fields.

Sonntag, 30. Mai 2010

That’s not a bad answer, but I thought it could be done with just one field.

So I asked them on their opinion on a single field search.

And what they both told me outright: It’s not possible.



Sonntag, 30. Mai 2010

Why should something like this not be possible?

IF you have just a single field instead of two, you are missing lots of information.

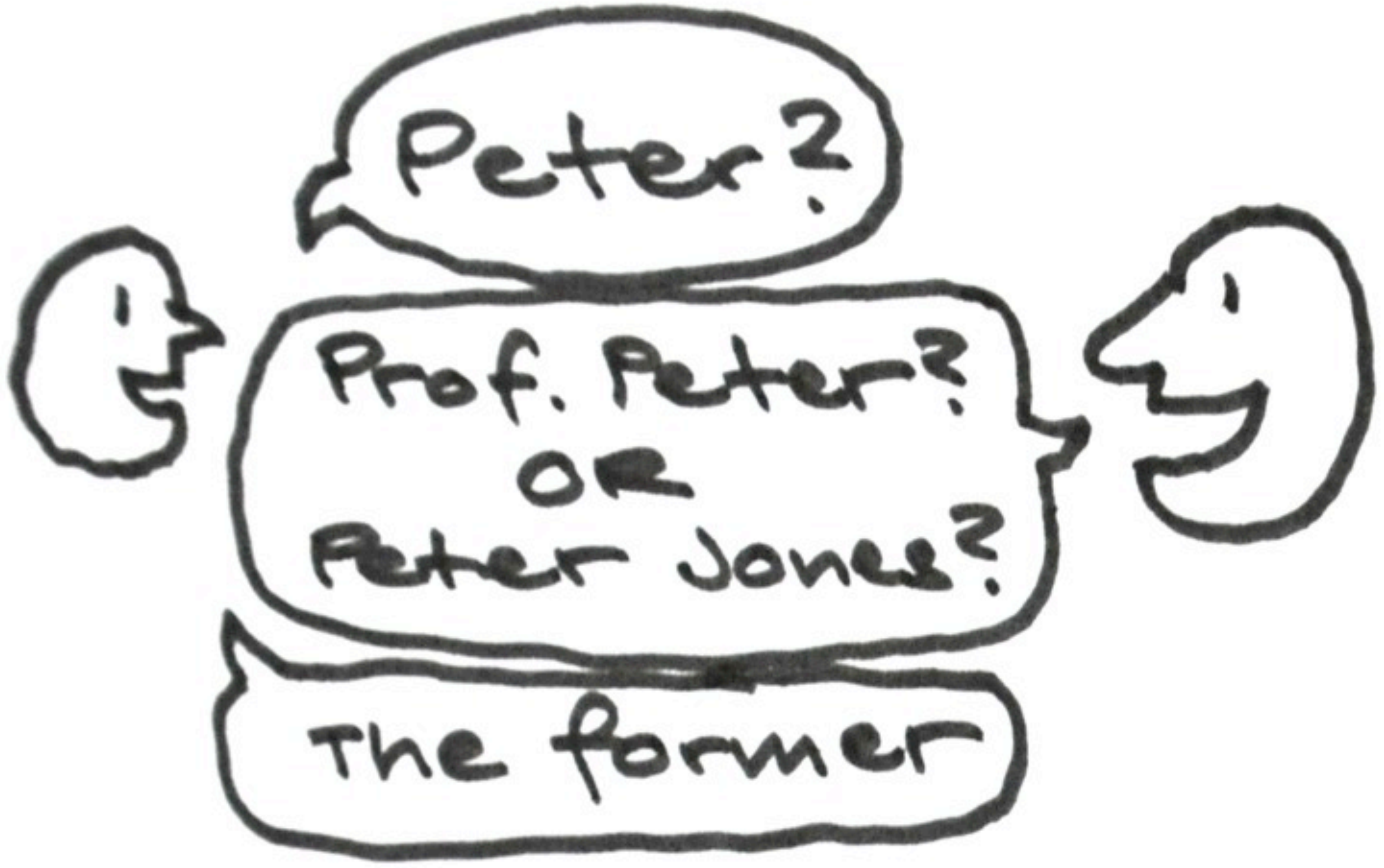
The only information you get … is BASICALLY the order of the words. What can we do?

Sonntag, 30. Mai 2010

Usually it is a good idea to mimic real world behaviour.

I had the following idea: If SOMEBODY asks me for an address with a name, somehow this works JUST like a single field query.

Let me give you an example:



Sonntag, 30. Mai 2010

You are on the left. I'm the guy on the right: Big nose, big mouth.

If you ask me about the phone number of Peter,
I might not know WHAT Peter you are referring to, and ask back for details.

I might know just two, but our search engine for example knows 80'000 Peters.

Sonntag, 30. Mai 2010

This is mimicked in the search engine.

Let me show it to you live.

Demo

Peter
Hans Peter
F* L* O*

Sonntag, 30. Mai 2010

As you saw, devising a good single field interface is quite complex.

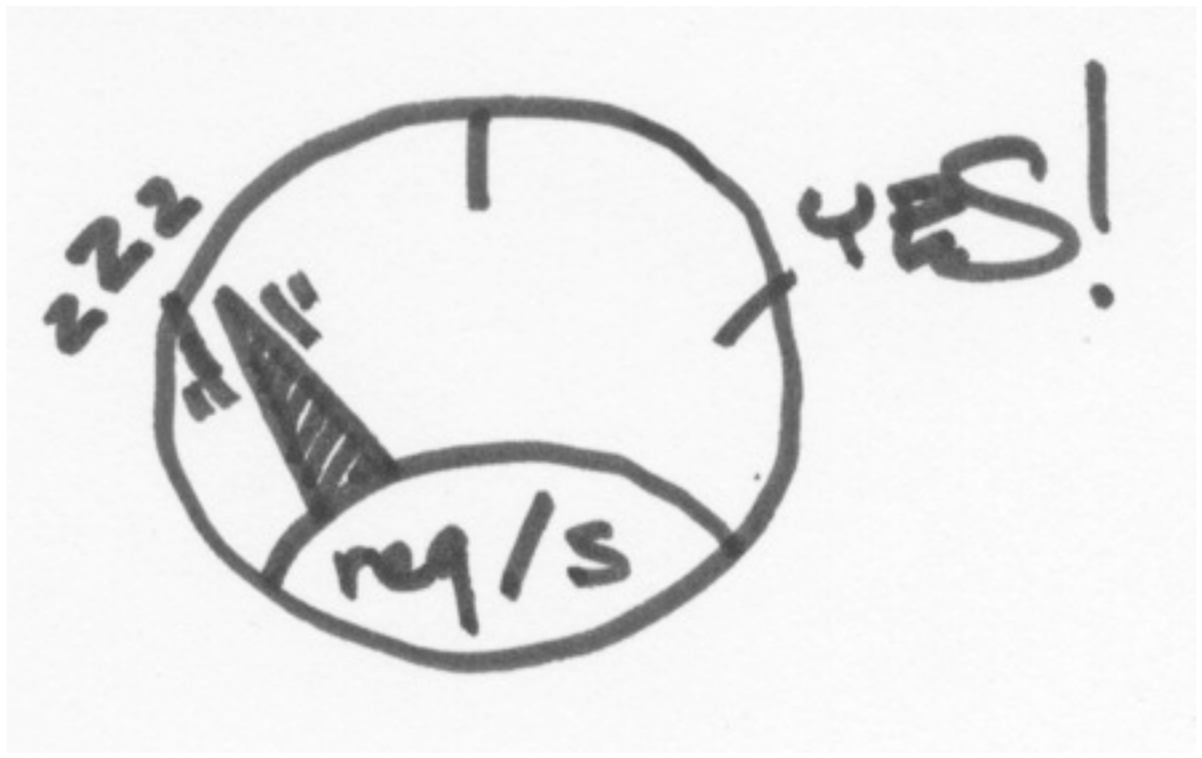
But the really hard part proved to be in the backend, driving this user interface.

There’s a lot of natural language processing, combinatorics and Bayesian inference involved, for example. And as we ALL know: Ruby is perfect for a job like that.

Sonntag, 30. Mai 2010

So we soon had a somewhat working prototype.

It had a little problem, though.



It was a bit slow, just 1 or 2 requests per second.

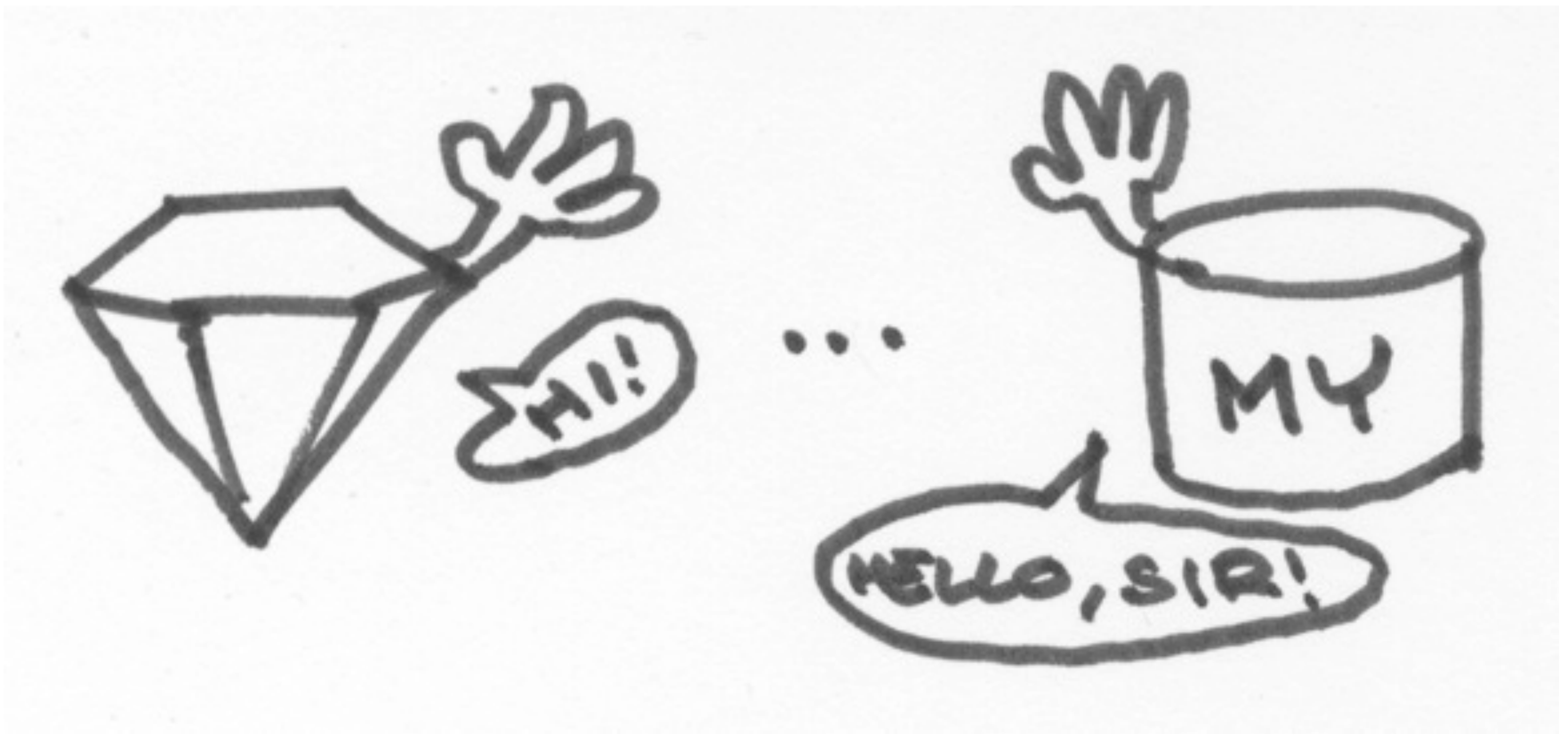
But my boss was SO optimistic. HE told me that if it is TOO slow, we can just buy 50 more servers. Normally it is ME telling that to MY bosses.

Sonntag, 30. Mai 2010

So I needed to get Ruby a bit faster.

So how did I make it go fast?

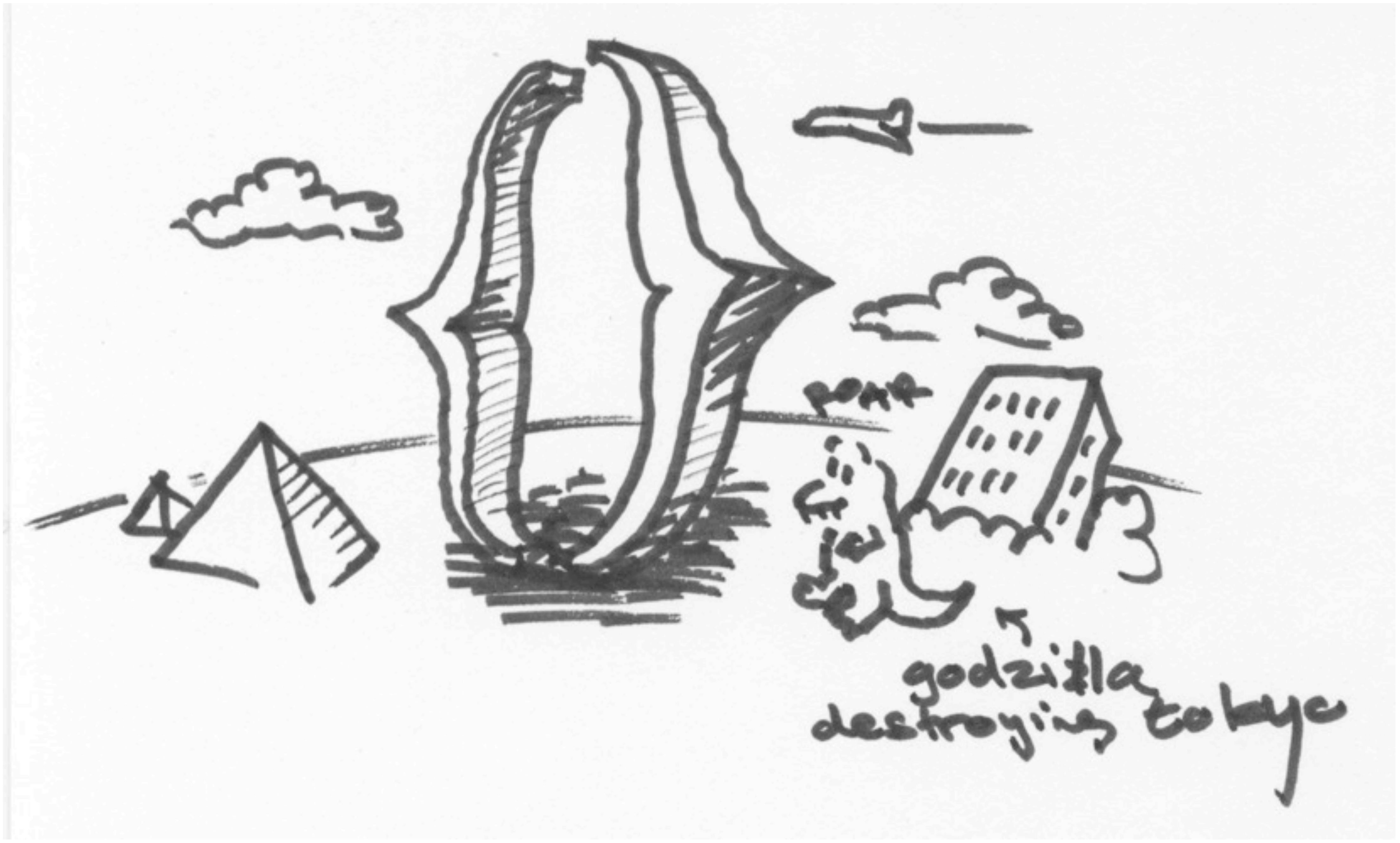
Quite a few things. But let me tell you shortly about three techniques that I used.



Sonntag, 30. Mai 2010

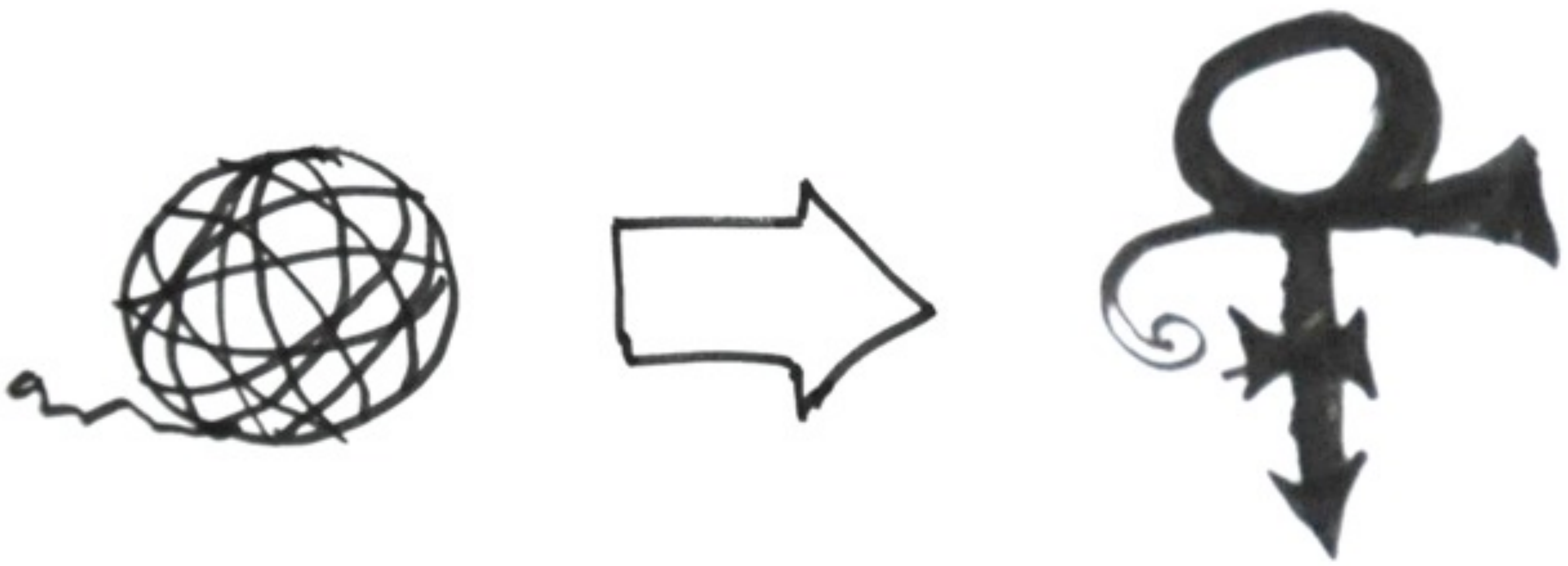
The backend and the indexes in the prototype were on MySQL. And really optimized.

But Ruby talking to the database just wasn’t fast enough.



I exchanged that for in-memory hashes.

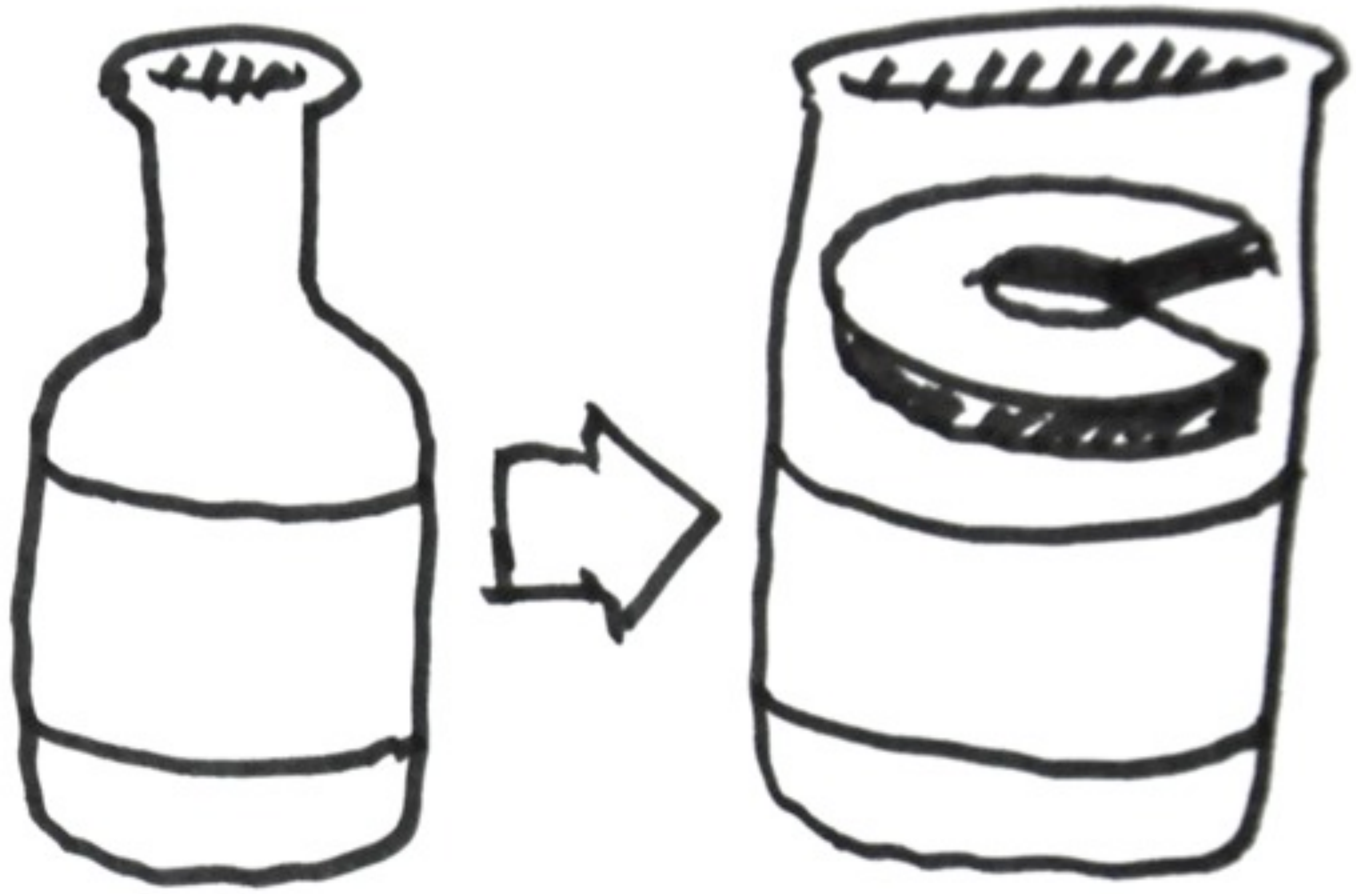
Massive in-memory hashes. In total, they have a size of about 10 GB.



Sonntag, 30. Mai 2010

The prototype used a lot of strings, which I then replaced with symbols.

If you want speed, and no trouble with the garbage collector, I recommend you use symbols.



Sonntag, 30. Mai 2010

Also, I replaced a single bottleneck – intersecting multiple large arrays – with 20 lines of Ruby C code.

Pjotr has already covered this part very nicely, so I refer to his presentation.

So that’s how I made this speCIFIC search engine fast.

What if YOU need to speed up Ruby? What do you need to look out for?

First of all, Ruby is programmed in a rather general way.
It needs to be fast or ok working for 95% percent of the cases. If you’re doing something like THIS, then YOUR case is probably not one of them.

Also there are a few surPRISES along the way. Let me show you one example.

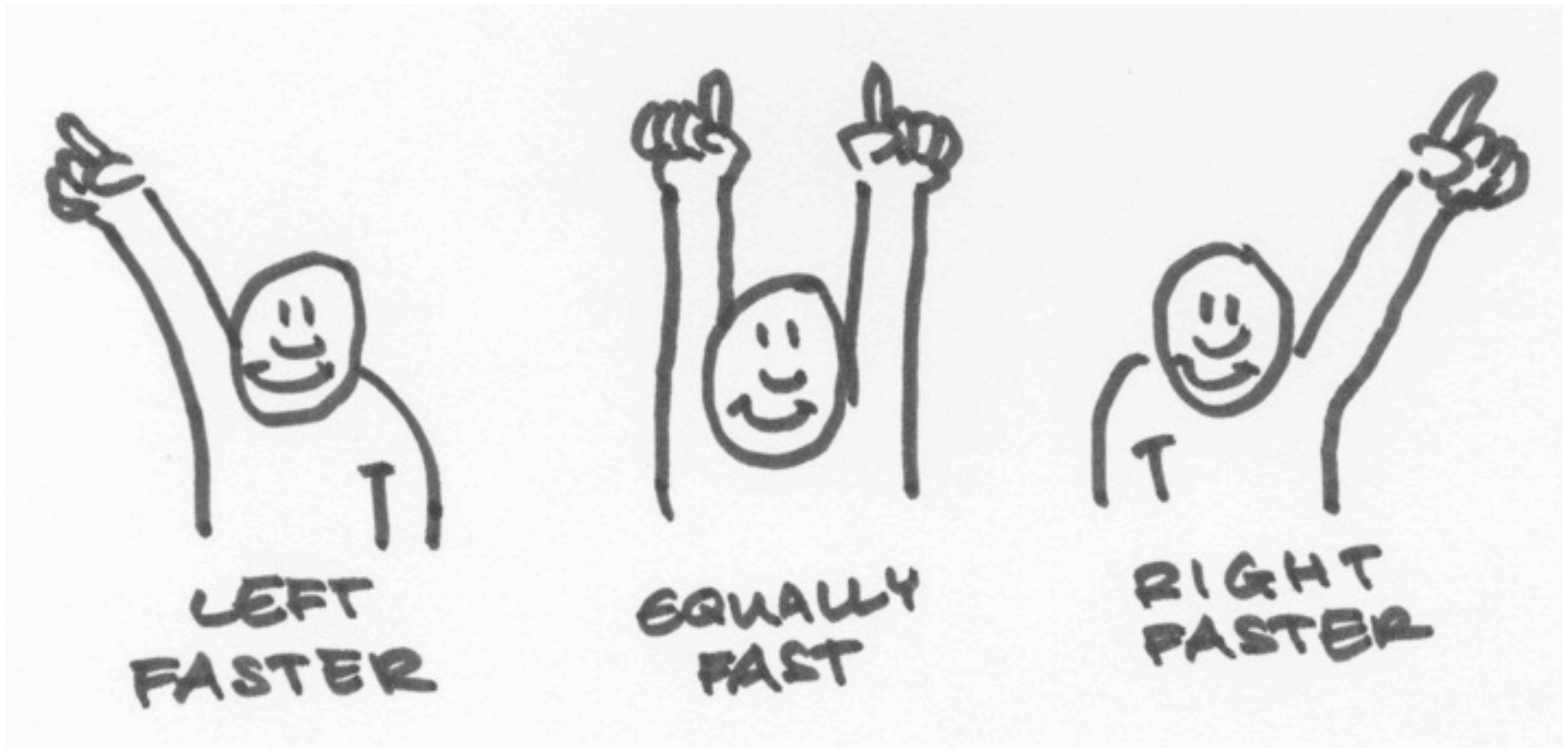
Sonntag, 30. Mai 2010

THIS calls for some AUUdience participation.

What I needed was large – and I mean laarge – arrays to intersect fast.

```
long  = (1..100_000).to_a
short = (1..10).to_a
```

```
short & long                  long & short
```



We have a long array with 100’000 elements. And we have a short array, with 10 elements.

We need the intersection, the ampersand.

Show me your hands! Are they equally fast? If not, which one is faster?
long & short is faster, by about 5 times … Why are they not the same speed?

Sonntag, 30. Mai 2010

I won’t give you the answer, but instead encourage you to look under Ruby’s hood to see why this is so.

Most of us never look how the Ruby code actually works. Most of us prefer to dance with the unicorns, run with the mongrels, ride the rails. And that’s ok.
<CLICK>
But what I encourage you to do, is click “show source” once in a while. You will learn a LOT of stuff about Ruby.

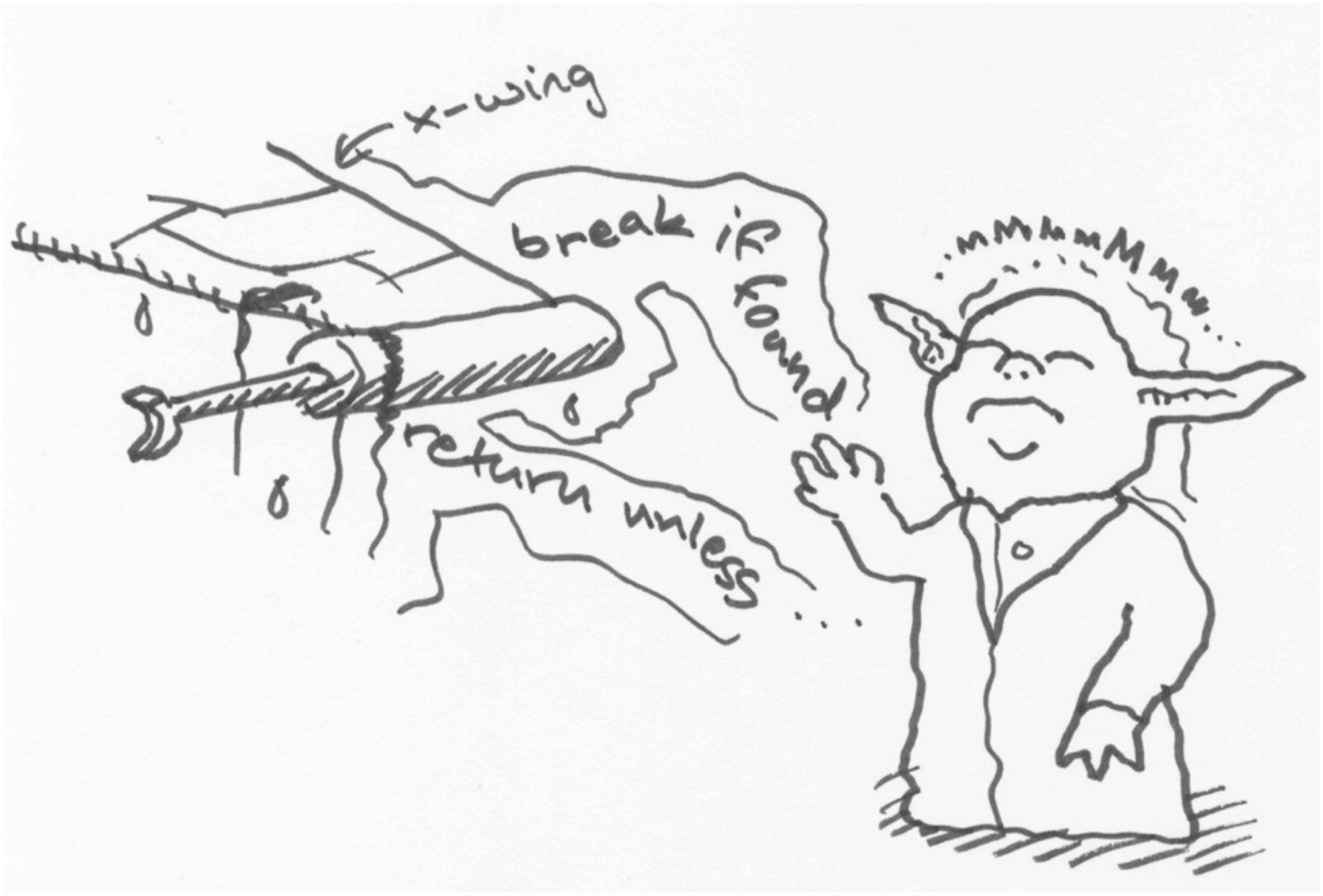
Most of us never look how the Ruby code actually works. Most of us prefer to dance with the unicorns, run with the mongrels, ride the rails. And that’s ok.
<CLICK>
But what I encourage you to do, is click “show source” once in a while. You will learn a LOT of stuff about Ruby.

Sonntag, 30. Mai 2010

Use the source, luke.

Sonntag, 30. Mai 2010

Use the source, luke.



Use the source, luke.

Yodacode:
confront_father if force_with_you?
instead of
if force_with_you? confront_father end

Mindset

And that’s part of the mind set I’d like to give you as a guide for writing faster applications.
Read the source.

But BEFORE you do start writing: The first thing you should ask yourself has NOTHING to do with speed, but with COMPLEXITY:
Ask yourself. Is what I want to do trivial? Or a common problem? That needs a fast solution.

Sonntag, 30. Mai 2010

It’s already been done or solved. And faster than you can ever write it in Ruby.

Good examples are Sphinx, or Varnish.

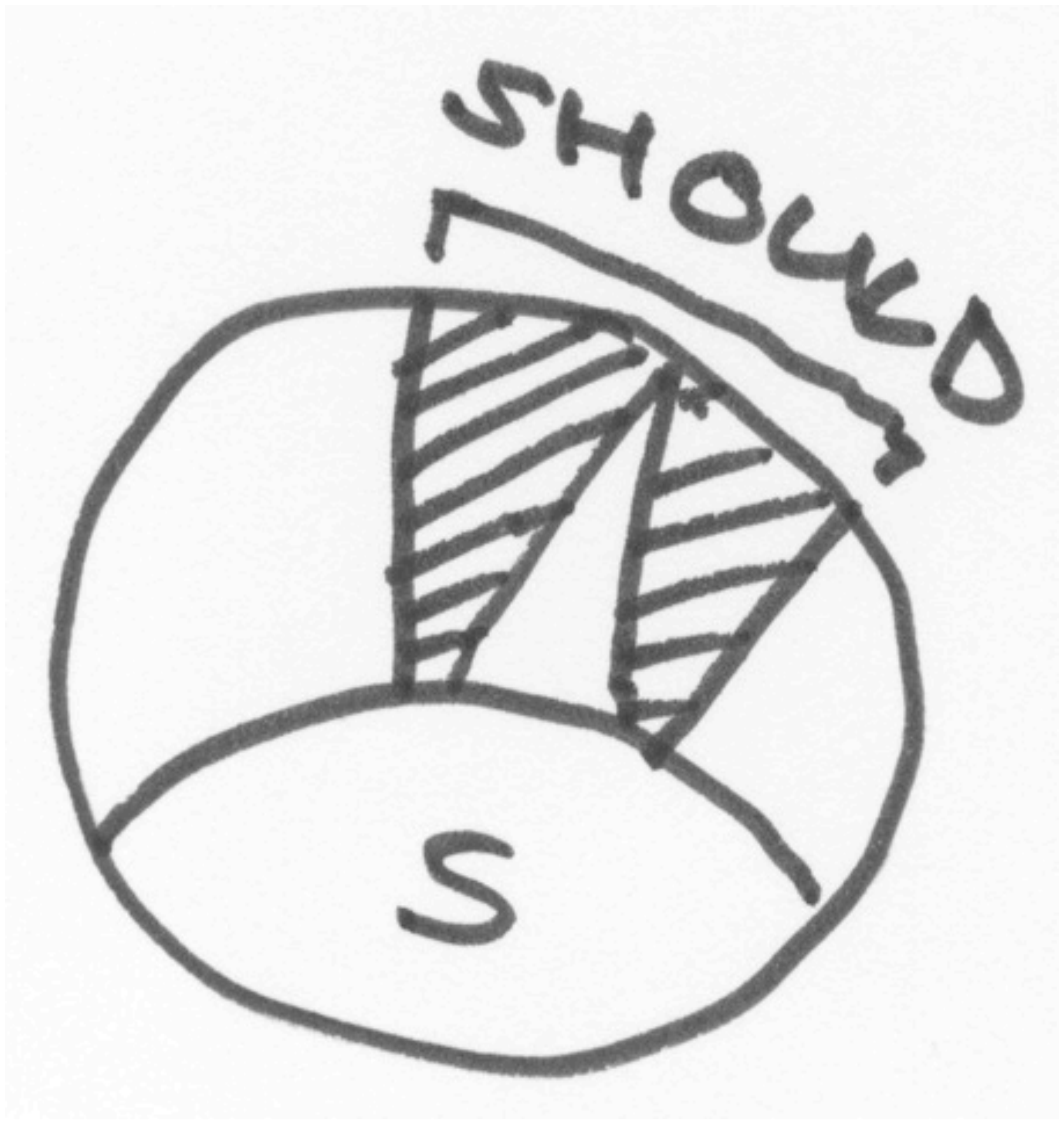
In a nutshell, ask yourself: Can I stand on the shoulders of a giant?

If you can't, go ahead and do it in Ruby!

Sonntag, 30. Mai 2010

If you go ahead and decide to do it in Ruby, consider to write your own special speed Ruby.
What I mean to say is not just extend Ruby, like by using ActiveSupport.

So if you need a few parts of Ruby to be a bit faster. Like I did for intersecting multiple monstrous arrays. Do it.
Just remember that sometimes you might be giving a bit of generality.



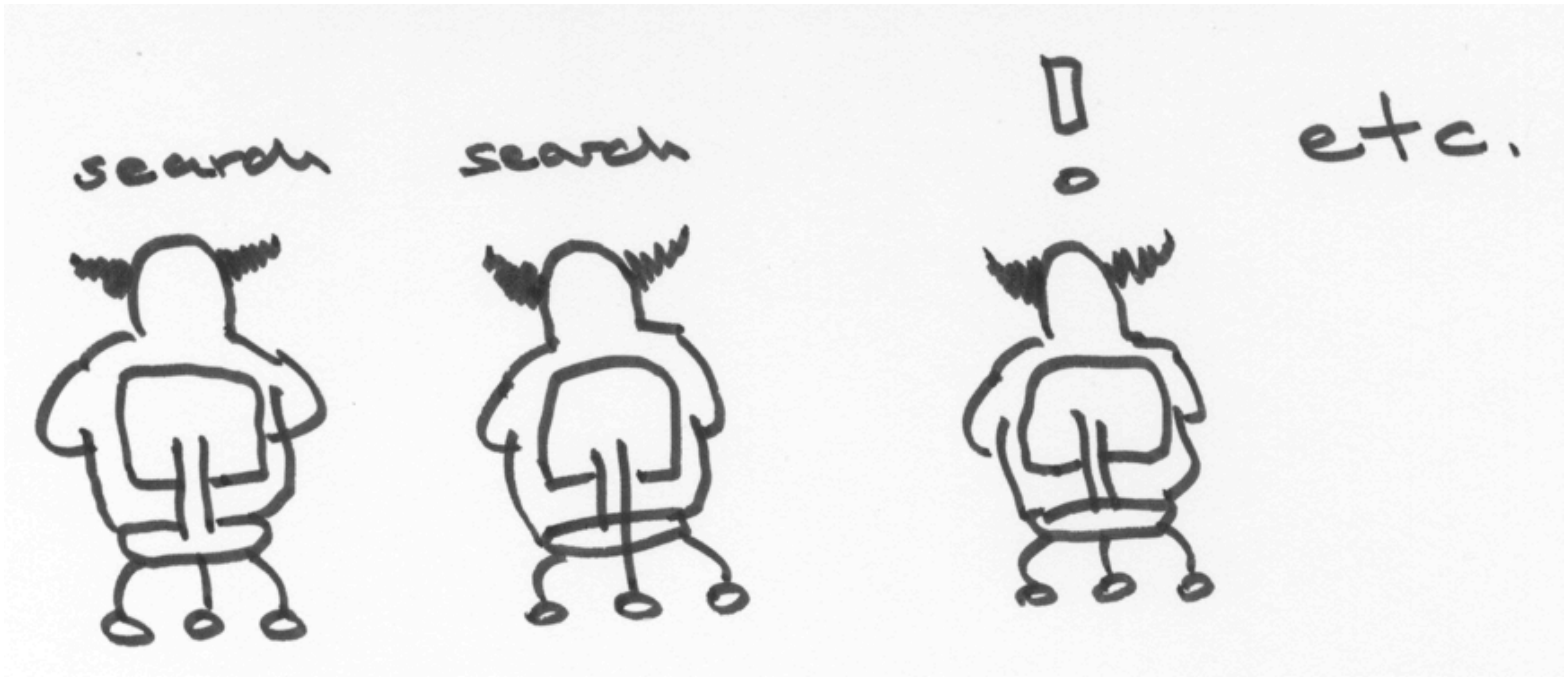
One thing you absolutely MUST do, is to write speed tests. So you don't just check for correctness, but also for speediness. That it stays fast.

By the way, writing tests for a fast thing is incredibly rewarding. I have about 1000 integration tests which run in 1.2 seconds. So you have NO reason not to test.

Sonntag, 30. Mai 2010

One more thing: Try to avoid speed craziness.

What do I MEAN by speed craziness? Let me close by giving you an example.



So that’s my boss. One day he came to me and told me that the search is really fast normally, but that sometimes, inexplicably, it takes 9 seconds for an answer. Why?

Sonntag, 30. Mai 2010

Of course, it was the garbage collector. A garbage collector is not very well suited for a search engine.
Because a garbage collector normally springs into action when something is happening.
And something is happening in a search engine when somebody searches. And I have huge hashes to garbage collect on.

Sonntag, 30. Mai 2010

And that’s when I got a little speed crazy.

Rewriting the code. Installing object pools, using 3 or less instance variables per object.
Stuff that made the code HARD to understand. Don’t do this.

It’s just NEVER worth it. Keep the code clean and nicely structured.

Sonntag, 30. Mai 2010

Instead, take a step back and look for any possible solution.

What I thought about first, but immediately dismissed as a dirty and “non-computer sciency” solution was simply switching the Garbage Collector off.

You’re probably all thinking: Is he crazy? What about the garbage? Will it just accumulate. Like in Naples?

Instead, take a step back and look for any possible solution.

What I thought about first, but immediately dismissed as a dirty and “non-computer sciency” solution was simply switching the Garbage Collector off.

You’re probably all thinking: Is he crazy? What about the garbage? Will it just accumulate. Like in Naples?

Let me explain. I’m using Unicorn. Unicorn is a webserver that uses a master process to fork off worker processes.

Initially they share the memory. As the workers start to handle the requests, they use up memory at about 1MB per request.

So let me show you some code.

```
# Honorable kill technique.

# Switch off the Garbage Collector.
#
after_fork { GC.disable }

RackResponder = lambda do |env|
  possibly_commit_harakiri
  …
end

# Commit Harakiri/Seppuku, kill yourself.
#
def possibly_commit_harakiri
  return unless commit_harakiri?
  Process.kill “KILL”, Process.pid
end
```

First, I switch off the garbage collector, just after forking the worker.
On each request, the child Unicorn checks if it time to retire honorably.
If yes, it commits suicide. It will be killed AFTER the request and its memory will be freed by the OS.
The Unicorn MASTER process then (extremely quickly) forks a new Unicorn worker, ready for the following requests.

Sonntag, 30. Mai 2010

Ok…
So Ruby is a programmer’s best friend.
It helps you a LOT with complexity.
But if YOU invest a little time and get to know it EVEN a bit better.
It might just break the principle of least surprise, and be fast just where you need it to be.
Thank you and Dschinquia.

Thank you

And thanks to:
Andi, Johanna, Kaspar, Niko, Severin, Stefan

Btw, if a Unicorn commits suicide, there is no blood, only rainbows.

Q&A

Watch this repository for news:

http://github.com/floere/search-engine-notification