



# Clojure in Practice

Karl Krukow, Trifork,  
kkr@trifork.com.  
Mar. 7th, 2012

# About me



- Working at Trifork for about 6 years on Web, JavaScript, Java/JEE, Ruby/Rails, iOS, Conferences and Training.
- Last two years on iOS as part Trifork teams for some of the larger Danish banks.
- Clojure and Rich Hickey fan-boy!
- Founder of Danish Clojure Users' Group (<http://clojure.higher-order.net>).
- Recently part of a start-up doing mobile automated testing: LessPainful (<http://www.lesspainful.com>).

# About me



- Working at Trifork for about 6 years on Web, JavaScript, Java/JEE, Ruby/Rails, iOS, Conferences and Training.
- Last two years on iOS as part of Trifork team for some of the larger Danish banks.
- Clojure and Rich Hickey fan-boy
- Founder of Danish Clojure Users' Group (<http://clojure.higher-order.net>).
- Recently part of a start-up doing mobile automated testing: LessPainful (<http://www.lesspainful.com>).



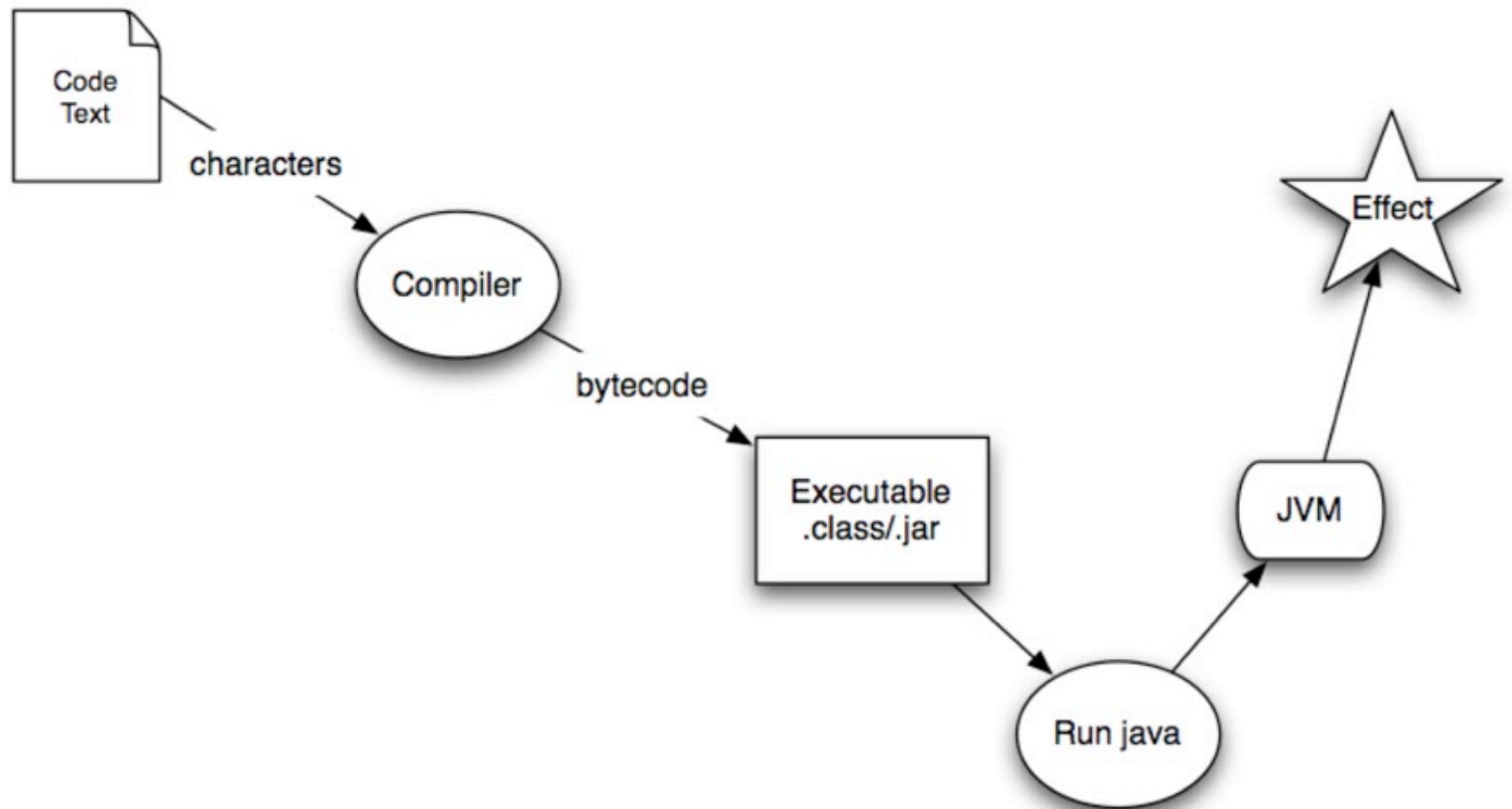
# About you :)

# About you :)

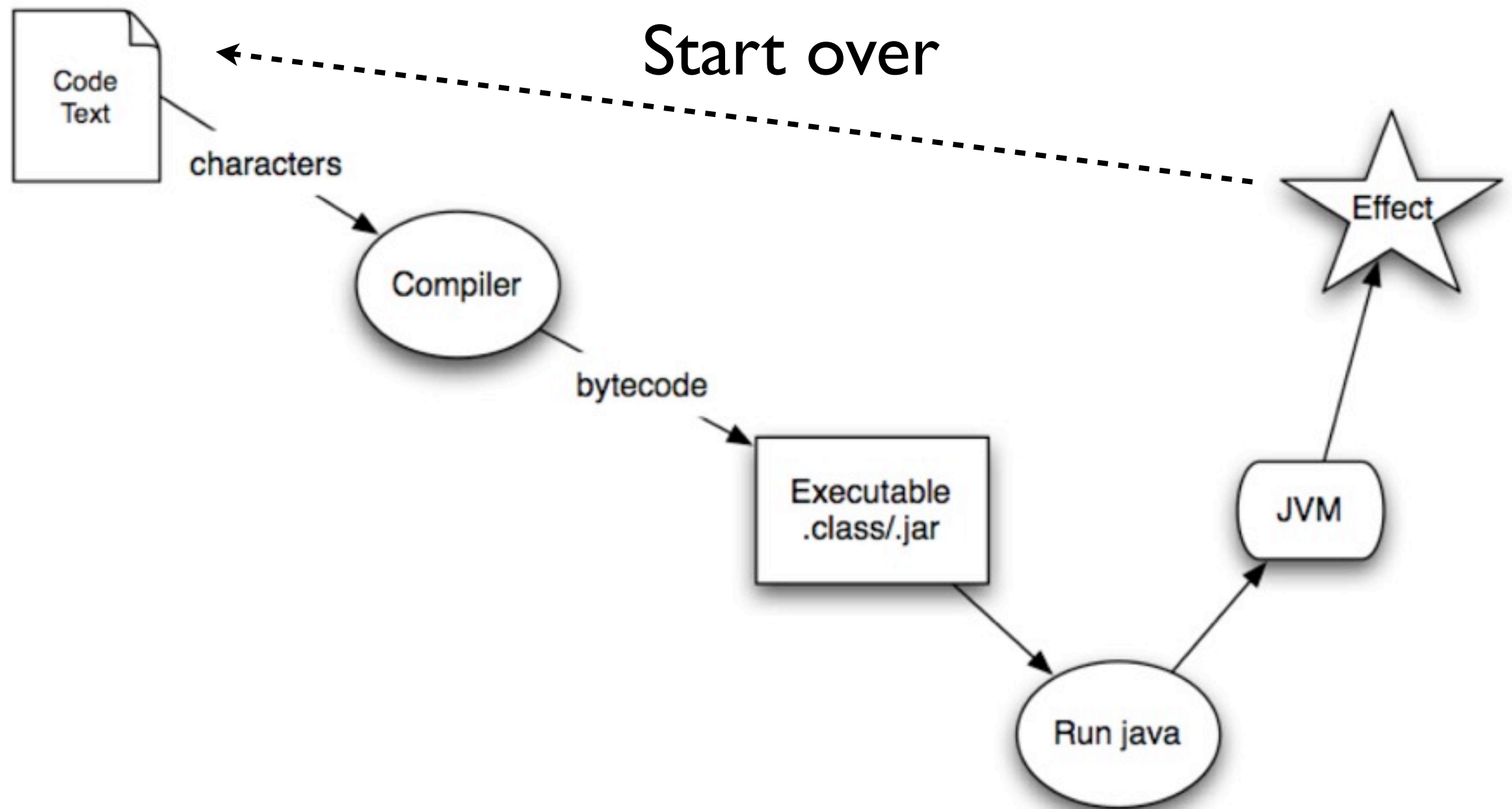
- What kind of technologies are you working with?
- What are the pain points in your everyday software development life?

# Traditional Development Experience

# Traditional Development Experience

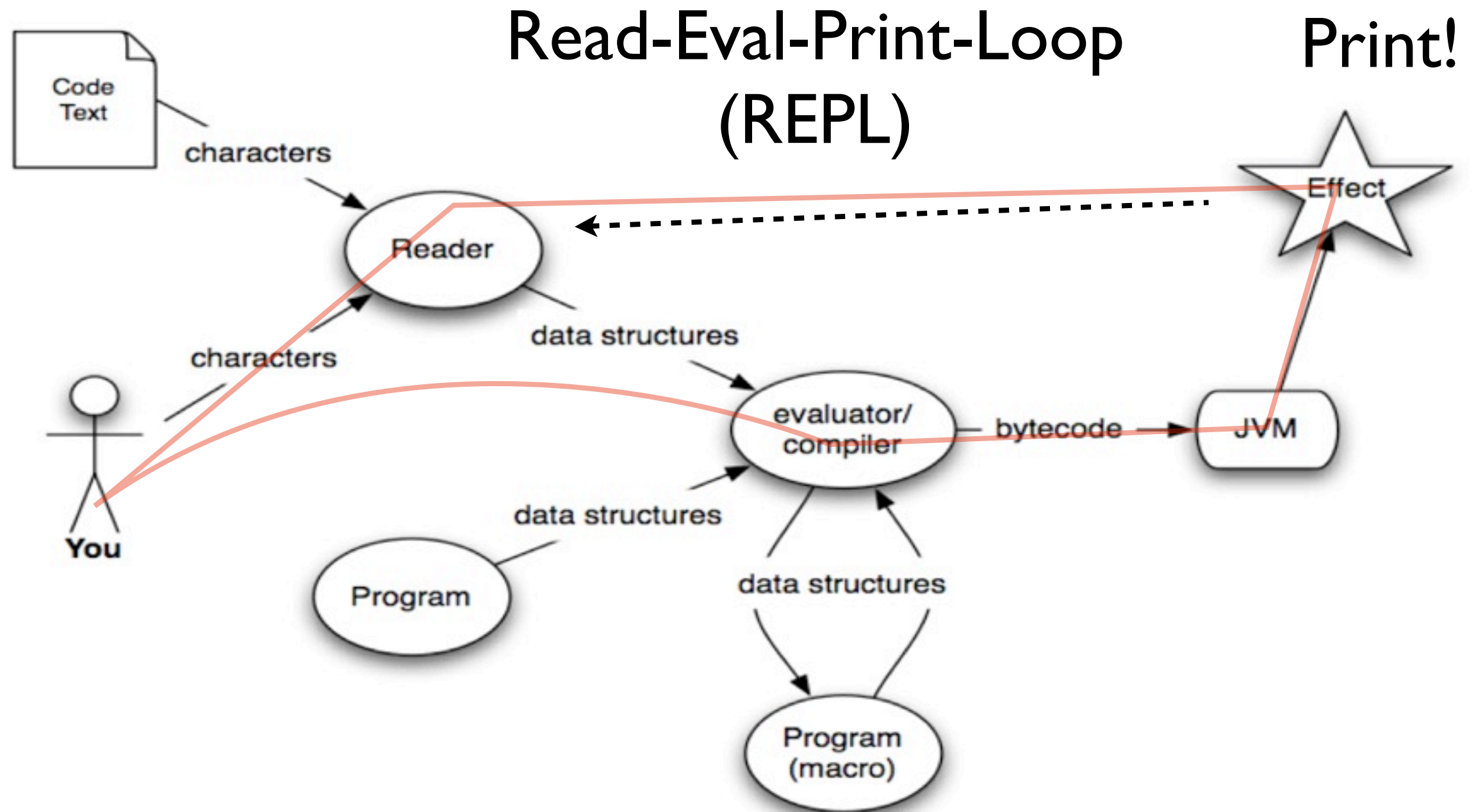


# Traditional Development Experience





# The Clojure Experience



# Agenda

- Very Short and very high-level intro to Clojure
- Clojure in Practice
  - Pointers: How to get started? Installation and tools.
  - Pointers: Editing, running, debugging.
- A simple mini app:
  - sucking data off AWS.
  - data manipulation
  - storing it in a Lucene index

# Clojure in one slide

- Functional dynamic language
  - Persistent data structures, pure functions, sequence library
- A unique programming & concurrency model
  - State management constructs: var, ref, atom, agent
- On-the-fly & AOT compilation: JVM bytecode
  - Deep two-way JVM interop.; embraces host
- A LISP family member
  - Meta programming, closures, interactivity

# More Clojure topics

- Records, protocols and polymorphism
- Multi methods
- Laziness
- Concurrency
- Macros and meta programming
- Numeric support and type hints
- JVM language interop
- Datomic
- Sequence library
- DSLs with Clojure
- Logic programming
- Meta data
- Parallelism
- Persistent data structures
- Testing
- Clojure CLR
- ClojureScript

# Note!

# Note!

- Please note that there is so much more to Clojure than what I'll present.
  - In this talk, we focus on the practical stuff: installing, development experience and tools.
- So do checkout the references at the end of these slides, particularly the videos by Rich Hickey: <http://clojure.blip.tv>
- Come to Goto Copenhagen 2012, <http://gotocon.com/cph-2012/>
  - Meet Rich Hickey and Stuart Halloway
  - Join the free Danish Clojure Users Group event ([clojure.higher-order.net](http://clojure.higher-order.net))

# Data structures

- Lists - singly linked, grow at front
  - `(1 2 3 4 5)`, `(fred ethel lucy)`, `(list 1 2 3)`
- Vectors - indexed access, grow at end
  - `[1 2 3 4 5]`, `[fred ethel lucy]`
- Maps - key/value associations
  - `{:a 1, :b 2, :c 3}`, `{1 "ethel" 2 "fred"}`
- Sets `#{fred ethel lucy}`
- Everything Nests

# Syntax



# Syntax

- Syntax defined by data structures!

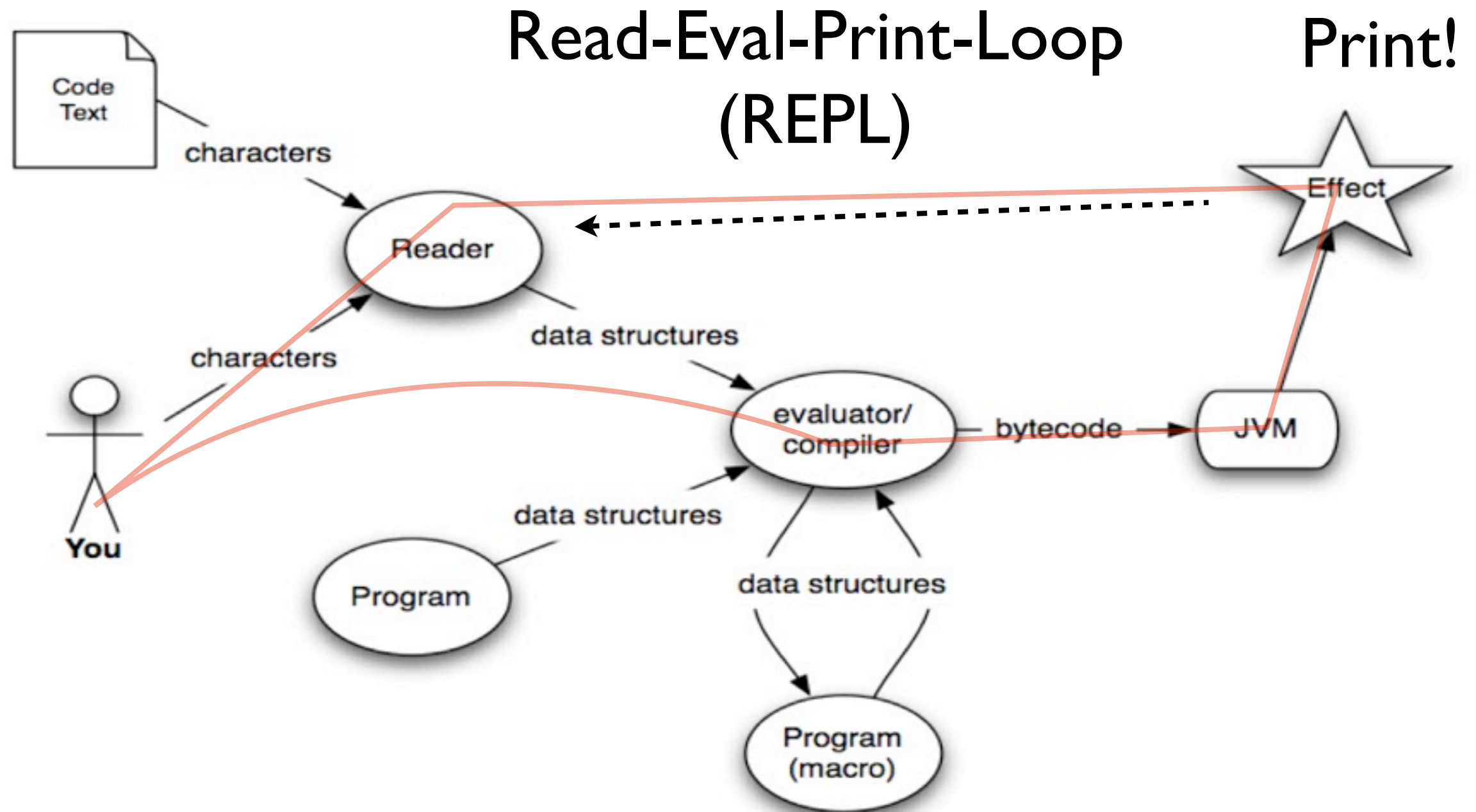
# Syntax

- Syntax defined by data structures!
- You use parens to describe data: `() [] {} #{}.`
  - Java/C# also has parens, they're just in a different place!
  - `obj.method(a,b)` vs `(method object a b)` -- that's not hard :)

# Syntax

- Syntax defined by data structures!
- You use parens to describe data: `() [] {} #{}.`
  - Java/C# also has parens, they're just in a different place!
  - `obj.method(a,b)` vs `(method object a b)` -- that's not hard :)
- Please, please, please do not judge the language by your initial impressions of its syntax!
  - There is so much beauty inside the `()`
  - Does not take long to get used to, and then you appreciate it.

# The Clojure Experience



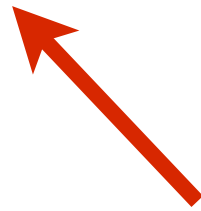
What is the most  
powerful program on  
my machine?

# What is the most powerful program on my machine?

```
krukow:~/cip/demo$ ls
README      lib          resources    test
classes     project.clj  src
krukow:~/cip/demo$ ps aux | grep -v grep | grep \"/#{@test.id}.logcat\" | awk '{print $2}' | xargs kill -9
krukow:~/cip/demo$
```

# What is the most powerful program on my machine?

```
user>
```



The REPL - isn't it cool ? :)

# Syntax and REPL examples def, fn, defn, call

- <http://tryclj.com/>



# Data structures

# Data structures

- In Clojure, all collections are *persistent* data structures.

# Data structures

- In Clojure, all collections are *persistent* data structures.
- There are efficient operations for creating variants of a data structure, for example:

# Data structures

- In Clojure, all collections are *persistent* data structures.
- There are efficient operations for creating variants of a data structure, for example:
- If *M* is a hash map, then **assoc**(*M*,*k*,*v*) is a map which is like *M* except that it maps key **k** to value **v** (almost like '*M*.add(*k*,*v*)').
  - (within 1-4x their mutable counterparts, or faster :)

# Data structures

- In Clojure, all collections are *persistent* data structures.
- There are efficient operations for creating variants of a data structure, for example:
- If  $M$  is a hash map, then **assoc**( $M, k, v$ ) is a map which is like  $M$  except that it maps key **k** to value **v** (almost like ' $M.add(k, v)$ ').
  - (within 1-4x their mutable counterparts, or faster :)
- Further: they are persistent meaning that the operations are non-destructive, e.g.,
  - Both  $M$  and **assoc**( $M, k, v$ ) are usable and preserve their performance guarantees.

# NO!

# NO!

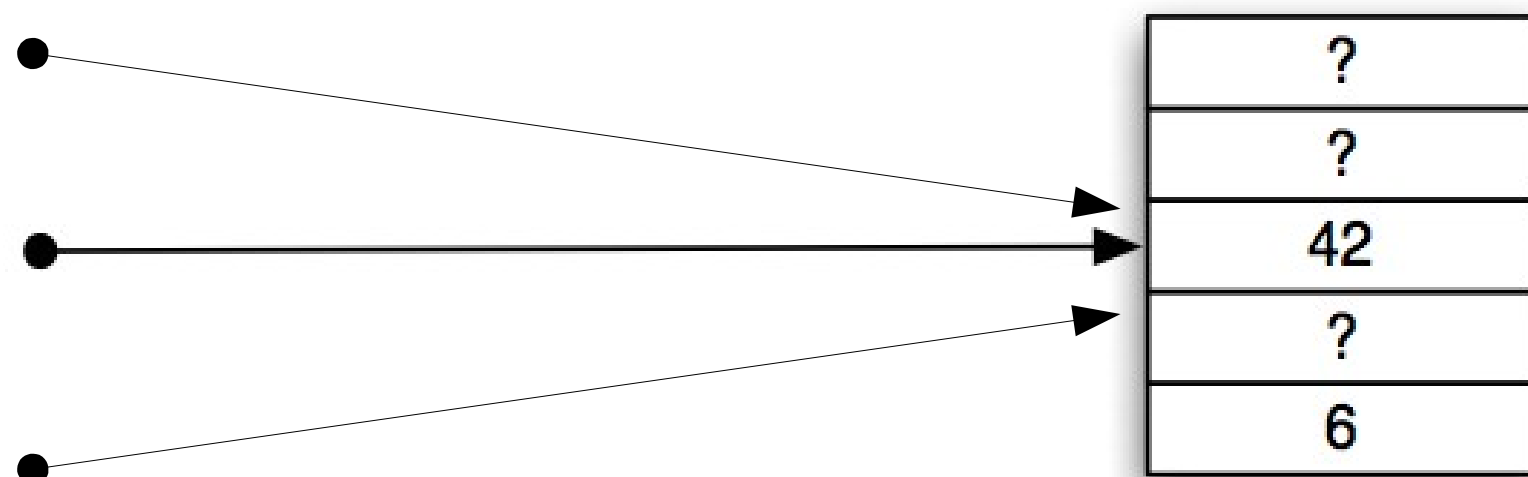
- ... that doesn't mean copying the entire structure(!)
- See my blog for explanation:
- <http://blog.higher-order.net/2009/02/01/understanding-clojures-persistentvector-implementation/>
- <http://blog.higher-order.net/2009/09/08/understanding-clojures-persistenthashmap-deftwice/>
- <http://blog.higher-order.net/2010/08/16/assoc-and-clojures-persistenthashmap-part-ii/>
- <http://blog.higher-order.net/2010/06/11/clj-ds-clojures-persistent-data-structures-for-java/>

# Clojure Philosophy

- Most programs could have dramatically less state than they do - we tend to introduce state just because it is the language default.
- In Clojure
  - We rarely use mutable objects, instead immutable data and pure functions.
  - Explicitly mark the parts of the program that have state!
  - State change is managed by Clojure.
  - References atomically change from referring to immutable objects.



# Direct references to Mutable Objects



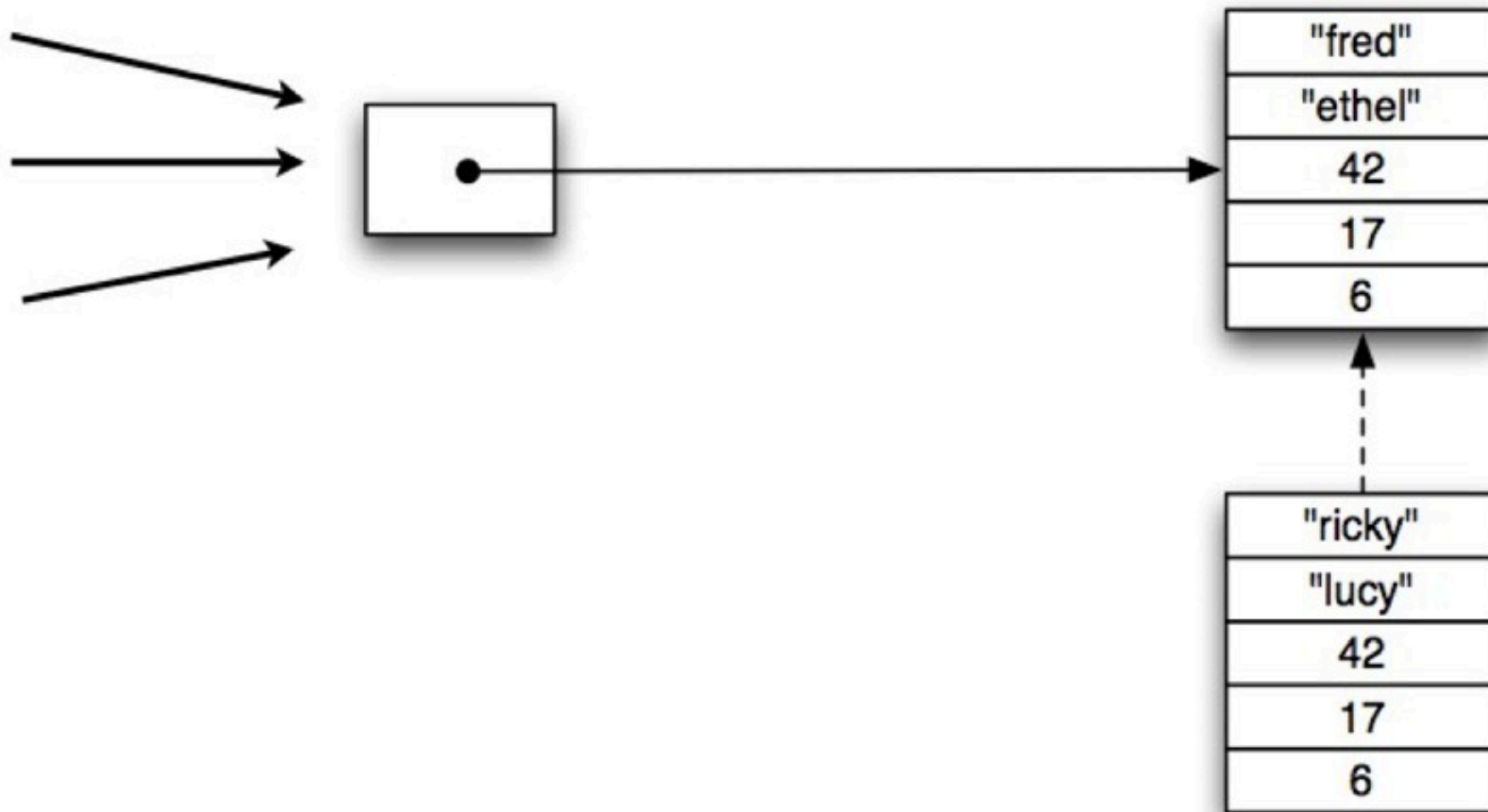
Ensuring a consistent Object is on your head

# Indirect references to Immutable Objects

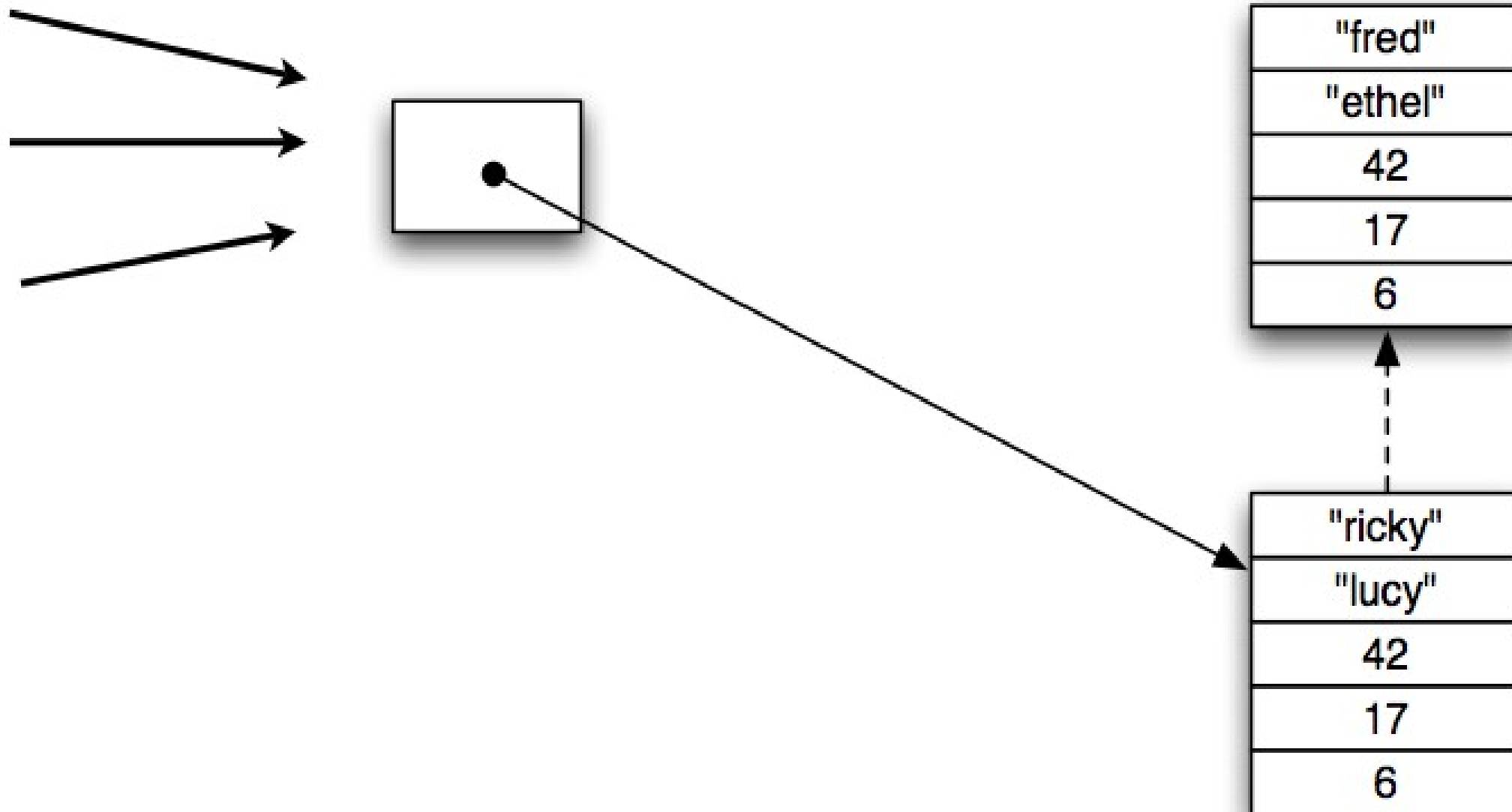


Never an inconsistent Object

# Persistent 'Edit'



# Atomic Update



# Atom

- Simplest reference type in Clojure
- `(def a (atom ["fred" "ethel" 42 17 6]))`

# Atom

- Simplest reference type in Clojure
- `(def a (atom ["fred" "ethel" 42 17 6]))`



# Atom

- Simplest reference type in Clojure
- `(def a (atom ["fred" "ethel" 42 17 6]))`



- `@a` dereferences `a`
- `(swap! a fun)` atomically swaps the value of `a` to the result of applying `fun` to the current value
- `AtomicReference` for Java people)

# Clojure in one slide

- Functional dynamic language
  - Persistent data structures, pure functions, sequence library
- A unique programming & concurrency model
  - State management constructs: var, ref, atom, agent
- On-the-fly & AOT compilation: JVM bytecode
  - Deep two-way JVM interop.; embraces host
- A LISP family member
  - Meta programming, closures, interactivity



# Closure in Practice

# Getting started

- <http://tryclj.com/> is just to play with Clojure.
- To run locally you need Java 1.5 and Clojure
  - Just a jar file ([http://clojure.org/getting\\_started](http://clojure.org/getting_started))
    - (for non-Java gues this is like a bundled java app)
- This is enough to try simple examples.
  - It does not show you the full development experience.
- To get more serious you need an IDE and some tools...
  - This takes some effort to install.
  - <http://dev.clojure.org/display/doc/Clojure+Tools>

# My preferred tools

- There are several options available for IDEs
  - Emacs, Eclipse (CounterClockwise), IntelliJ(La Clojure), Netbeans (Enclojure)..
  - Probably CounterClockwise is the easiest to use right now at least if you're used to Eclipse.
- My preferred tools are
  - Emacs + Sam Aaron's Live Coding Emacs Setup for Overtone
  - Leiningen (project management, dependencies, Emacs support)
  - slime+swank-clojure and CDT for REPL and debugging

# (Not-so) Easy Guide

- Download Emacs
  - <http://dev.clojure.org/display/doc/Getting+Started+with+Emacs>, see detailed comment by: [Muthu Bhuvana Sundaram T](#)
  - MacOS: <http://emacsformacosx.com>
- Install <https://github.com/overtone/live-coding-emacs>
- Add: `(setq slime-net-coding-system 'utf-8-unix)`
  - to `.emacs.d/config/core.el`
- Install Leiningen <https://github.com/technomancy/leiningen>
- Swank Clojure: <https://github.com/technomancy/swank-clojure>
  - `lein plugin install swank-clojure 1.4.0`
- Optional: lab Repl exercises <https://github.com/relevance/labrepl>

# Tools Demo

Emacs,

Lein

Live Emacs Coding

Swank Clojure

CDT

REPL tools

# A small sample app!

# References

- <http://clojure.org/>
- <http://clojure.blip.tv/>
- Stuart Halloway: Programming Clojure
- <http://www.4clojure.com/>
- Chris Houser, Michael Fogus: The Joy of Clojure
- Rich Hickey
- <http://www.infoq.com/presentations/hickey-clojure>
- <http://www.infoq.com/presentations/Simple-Made-Easy>
- <http://www.infoq.com/presentations/Are-We-There-Yet-Rich-Hickey>
- <http://www.infoq.com/presentations/Value-Identity-State-Rich-Hickey>
- DCUG: <http://clojure.higher-order.net/>