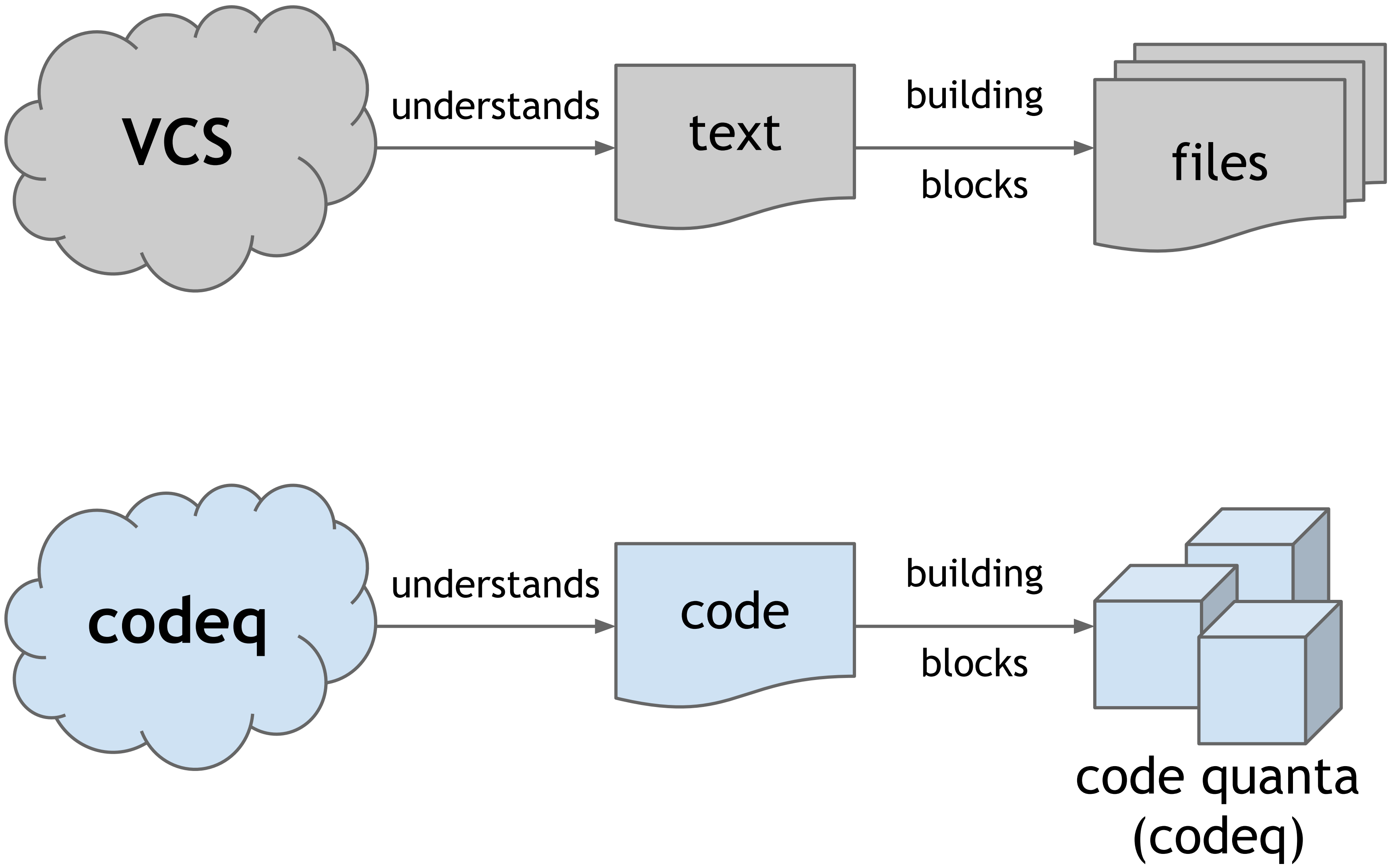
codeq
making a VCS code-sensitive

an evaluation

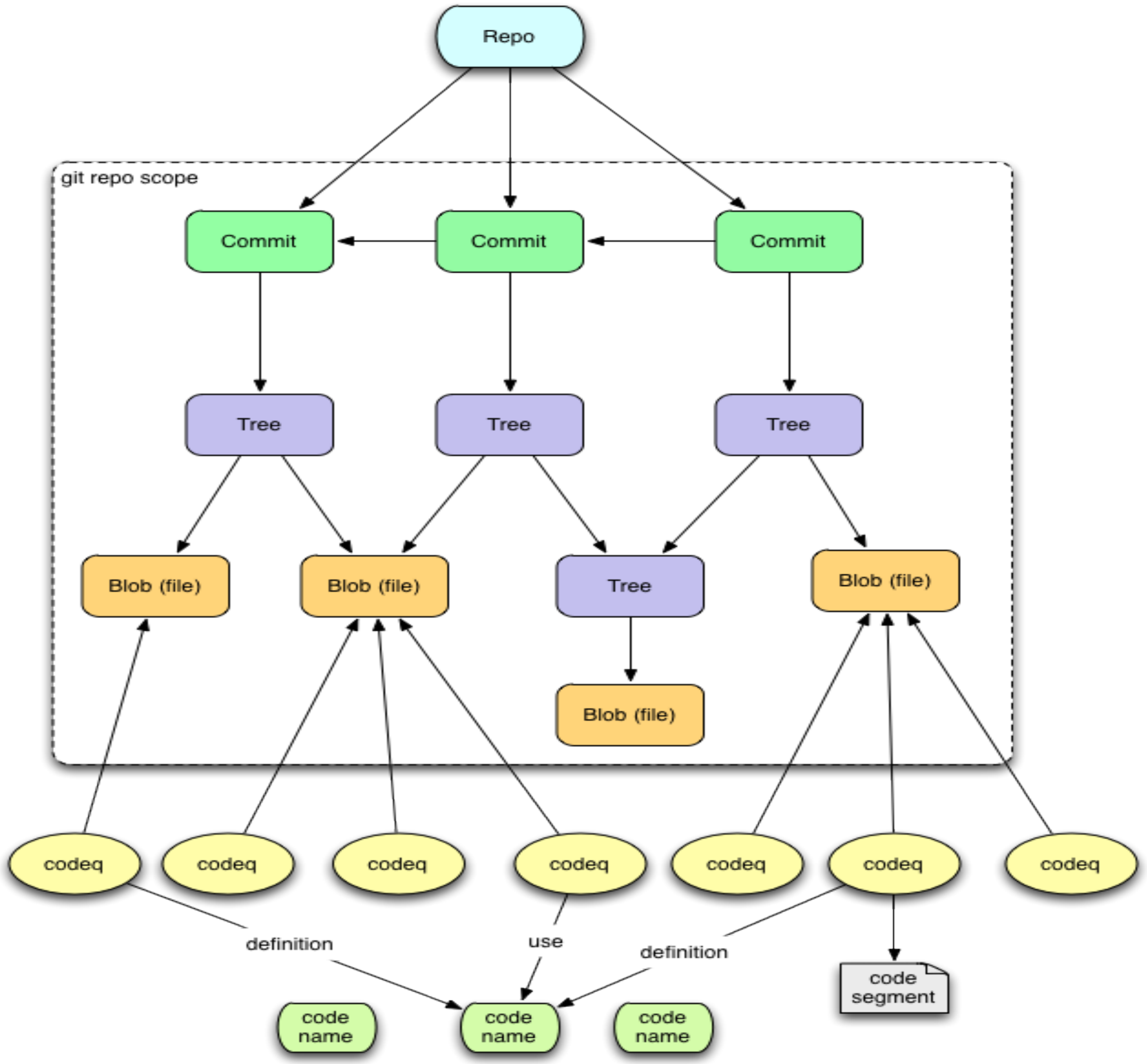
George {Kastrinis, Kollias}

Marketing Idea

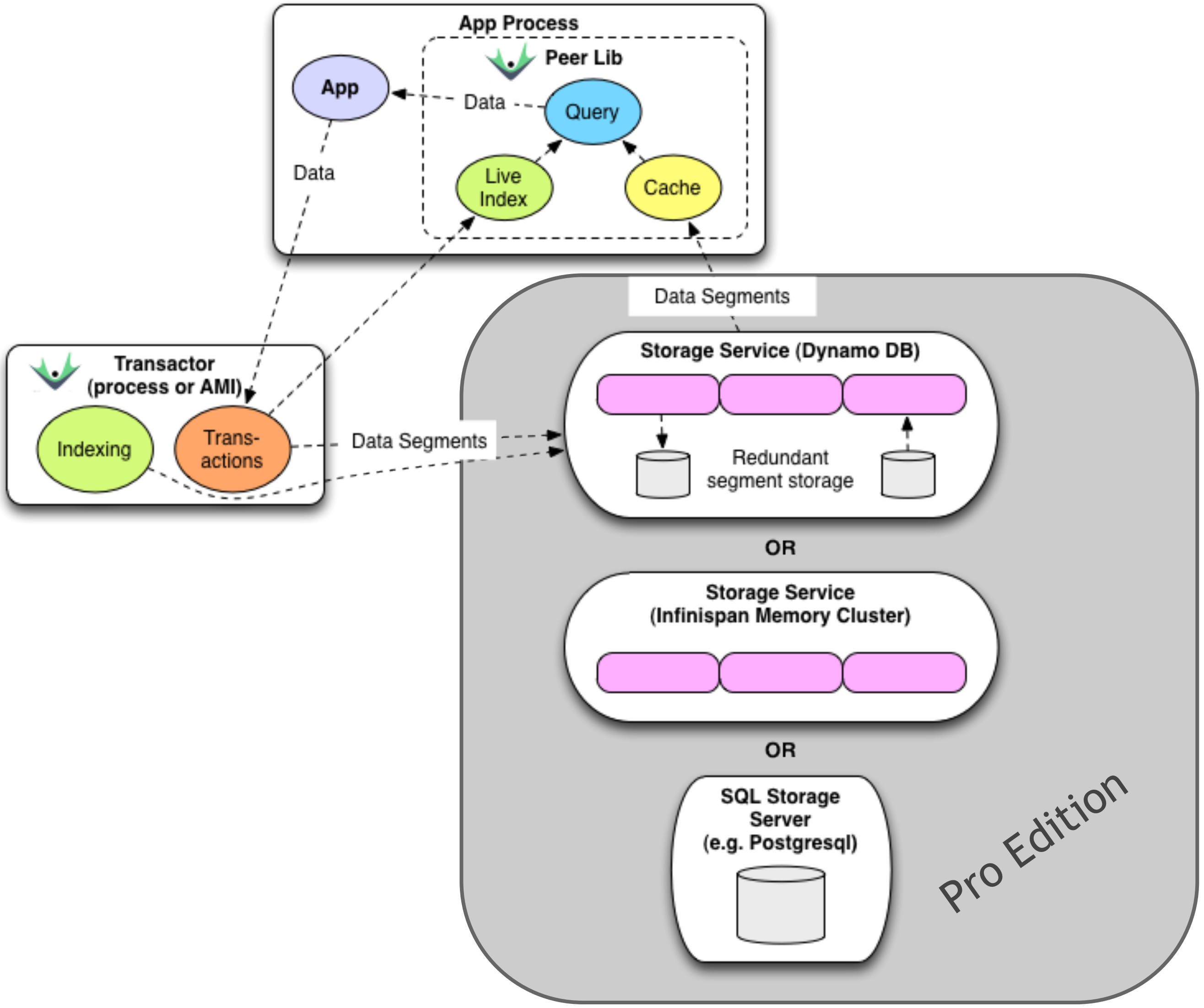
- **Import** a code repo in a DB
  - DB = Datomic (uses a Datalog variant)
- **Analyze** the source files
- Issue **queries** on code history


VCS
understands
text
building
blocks
files
codeq
understands
code
building
blocks
code quanta
(codeq)

Codeq Overview



Storing in Datomic


App Process
Peer Lib
App
Data
Query
Data
Live Index
Cache
Data Segments
Transactor (process or AMI)
Indexing
Trans-actions
Data Segments
Storage Service (Dynamo DB)
Redundant segment storage
OR
Storage Service (Infinispan Memory Cluster)
OR
SQL Storage Server (e.g. Postgresql)
Pro Edition

The long story

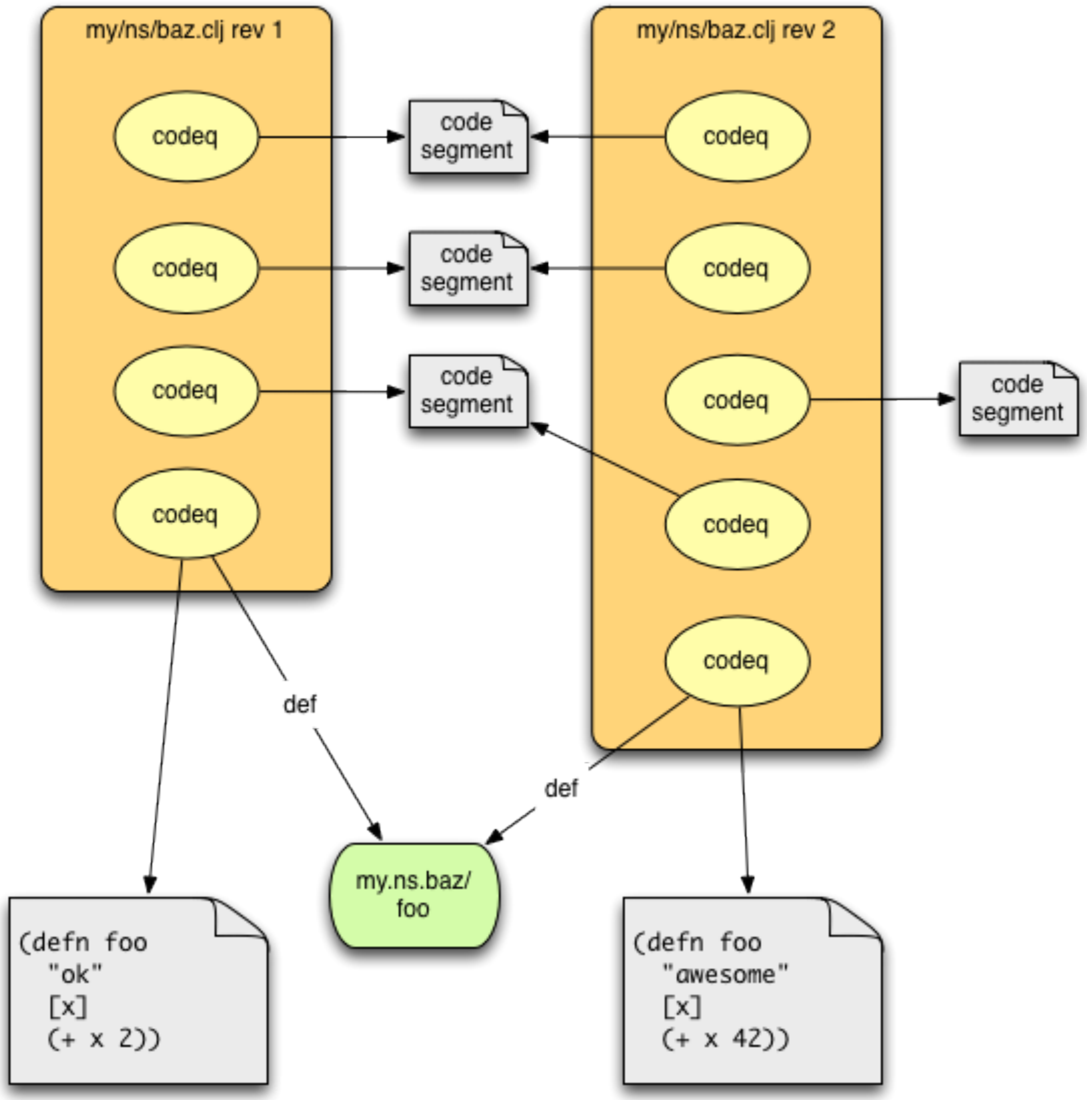
Importing git repos

- Parser for git repos structure
  - Finds the general file hierarchy
  - Imports all commits sequentially
  - 1:1 mapping of **commits** to **Datomic transactions**
    - Each Datomic transaction **annotated** with codeq commit's info

Analyzing repo code

- **Language-specific** analyzers
    - Break each (source) file to **codeqs**
    - Determine level of **granularity**
    - Label each codeq with **semantic** information
    - 1:1 mapping of **analyses/modified files** to **Datomic transactions**
    - Written in Clojure

Analyzing source file



```
(defn foo
  "ok"
  [x]
  (+ x 2))
```

```
(defn foo
  "awesome"
  [x]
  (+ x 42))
```

Codeq

- **Code** Quantum (segment)
- Encodes the **name**, **code, location** & **semantic** information
  - The semantics depend on analyzer
- Must be uniquely identifiable
  - e.g. Global namespacing

Datomic

- The underlying db
- Rules in a **Datalog** variant
- **Universal** Schema
    - **immutable** facts (*Datoms*) stored over time in the form:

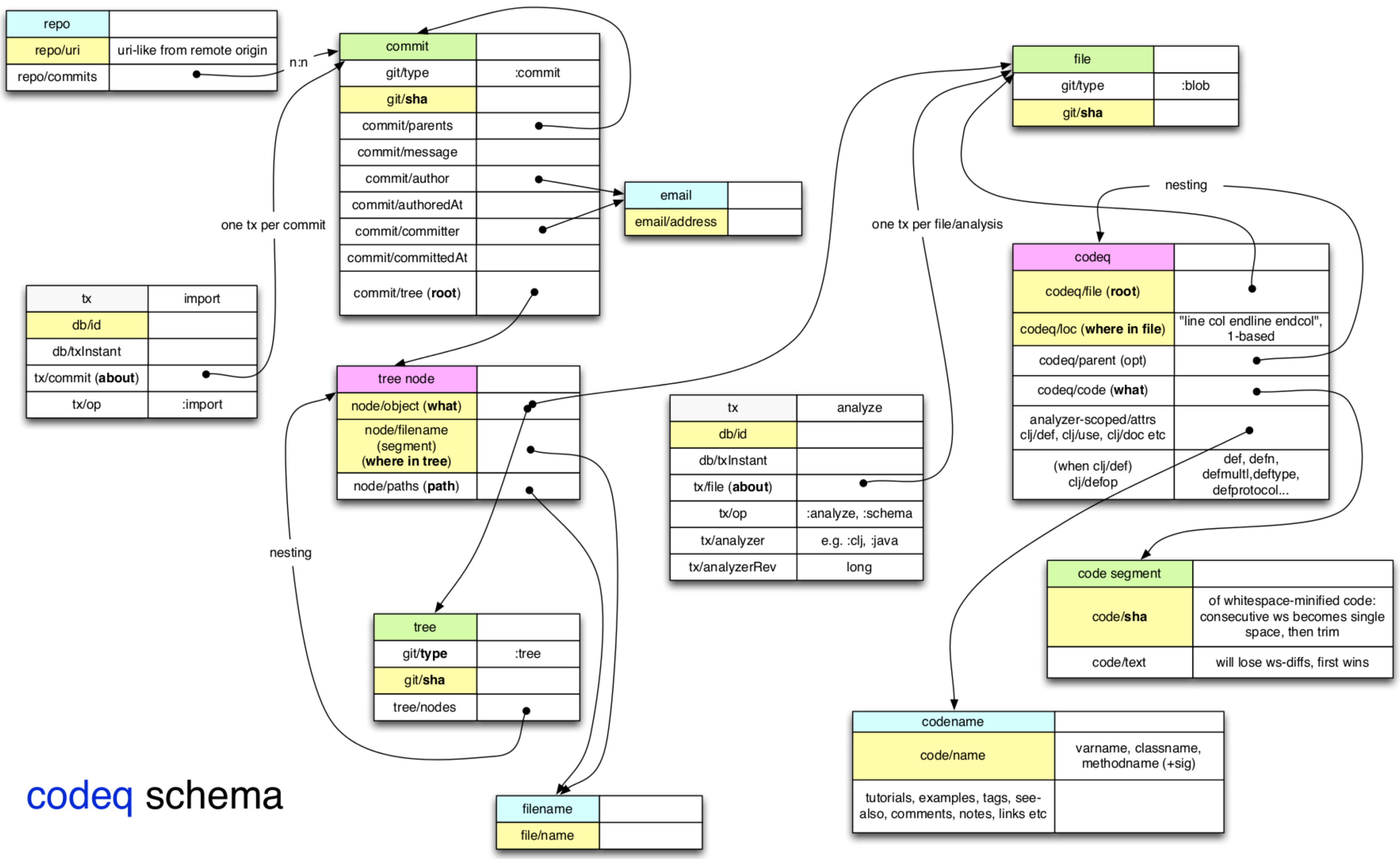
<entityID, attribute, value, transactionID, add/retract>

Datomic Versioning

```
> conn.transact(list(":db/add", 1, ":intern/firstName", "George"))
    <1, :intern/firstname, “George”, 2, add>
    <2, :db/txInstant, 2013-06-14 14:05:04.0000, add>
> conn.transact(list(":db/add", 1, ":intern/firstName", "Kostas"))
    <1, :intern/firstname, “George”, 3, retract>
    <1, :intern/firstname, “Kostas”, 3, add>
    <3, db/txInstant, 2013-06-14 14:06:04.0000, add>
> conn.transact(list(":db/retract", 1, ":intern/firstName",
"Kostas"))
    <1, :intern/firstname, “Kostas”, 4, retract>
    <4, :db/txInstant, 2013-06-14 14:07:04.0000, add>
```

We can get a DB value as of any specific point in time!
(using a specific transaction's timestamp)

Codeq Schema



repo	
repo/uri	uri-like from remote origin
repo/commits	●

commit	
git/type	:commit
git/**sha**	
commit/parents	●
commit/message	
commit/author	●
commit/authoredAt	
commit/committer	●
commit/committedAt	
commit/tree (**root**)	●

email	
email/address	

file	
git/type	:blob
git/**sha**	

tx	import
db/id	
db/txInstant	
tx/commit (**about**)	●
tx/op	:import

tree node	
node/object (**what**)	●
node/filename (segment) (**where in tree**)	●
node/paths (**path**)	●

tx	analyze
db/id	
db/txInstant	
tx/file (**about**)	●
tx/op	:analyze, :schema
tx/analyzer	e.g. :clj, :java
tx/analyzerRev	long

codeq	
codeq/file (**root**)	●
codeq/loc (**where in file**)	"line col endline endcol", 1-based
codeq/parent (opt)	●
codeq/code (**what**)	●
analyzer-scoped/attrs clj/def, clj/use, clj/doc etc	●
(when clj/def) clj/defop	def, defn, defmultl,deftype, defprotocol...

tree	
git/**type**	:tree
git/**sha**	
tree/nodes	●

code segment	
code/**sha**	of whitespace-minified code: consecutive ws becomes single space, then trim
code/text	will lose ws-diffs, first wins

codename	
code/name	varname, classname, methodname (+sig)
tutorials, examples, tags, see-also, comments, notes, links etc	

filename	
file/name	

codeq schema

Codeq Versioning

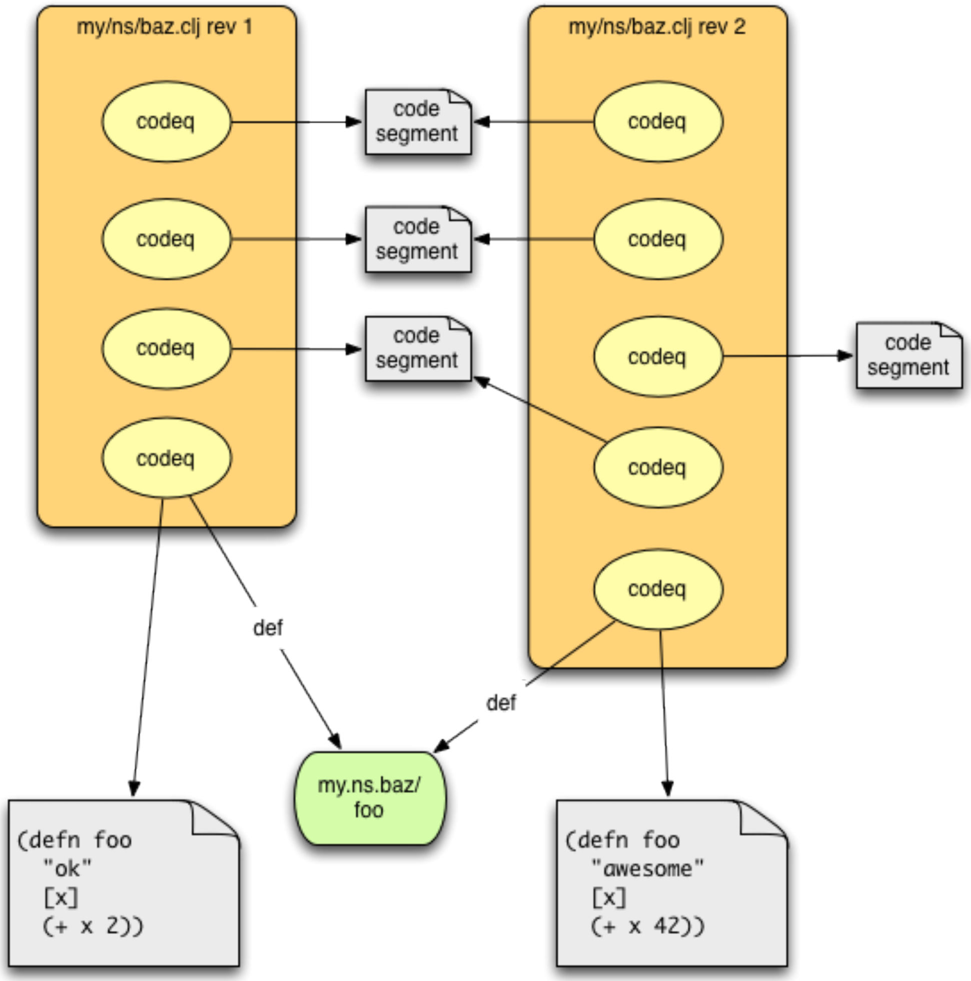
- Reflects git's structure
  - Keeps **parent** commits, **date**, **authors**,...
- Needs a single total order
  - In effect, import **only the active** branch
  - Datomic supports linear history
- Must import **in order**
- Ignore any missing **dependencies**
  - User must import them manually

Example Queries

- Find all the files referenced by a commit
- Find all the codeqs referenced by a commit
- Find all the commits including a file
- Find all the commits including a codeq
- Find all commits including a specific string
- Find all codeqs including a specific string
- ...

Real Example Query

Find all the different definitions of the function *datomic.codeq.core/test* when **firstly introduced** (each def)



```
(defn foo
  "ok"
  [x]
  (+ x 2))
```

```
(defn foo
  "awesome"
  [x]
  (+ x 42))
```

Real Example Query (LogiQL)

Find all the different definitions of the function *datomic.codeq.core/test* when **firstly introduced** (each def)

```
_[src] = mindate <-
agg<<mindate = min(date)>>(
    name[n] = "datomic.codeq.core/test",
    clj:def[n] = cq,
    codeq:codeSegm[cq] = cs, code:text[cs] = src,
    codeq:file[cq] = f, file:commits(f, c),
    commit:date[c] = date
).
```

Real Example Query (Datomic)

Find all the different definitions of the function
datomic.codeq.core/test when **firstly introduced** (each def)

```
(d/q '[:find ?src (min ?date)
       :in $ % ?name
       :where
       [?n :code/name ?name]
       [?cq :clj/def ?n]
       [?cq :codeq/code ?cs]
       [?cs :code/text ?src]
       [?cq :codeq/file ?f]
       (file-commits ?f ?c)
       (?c :commit/authoredAt ?date)]
   db  rules "datomic.codeq.core/test")
```

```
[["(defn test\n  ..."
  #inst "2012-10-06T18:07:38.00"]
 ["(defn test\n  ..."
  #inst "2012-10-03T02:29:19.00"]
 ["(defn test\n ..."
  #inst "2012-10-03T19:09:39.00"]
 ["(defn test\n ..."
  #inst "2012-09-28T21:55:25.00"]
 ["(defn test\n ..."
  #inst "2012-10-02T21:38:38.00"]]
```

Evaluation

Language Support

- Must write an **analyzer** for each language
  - Only the **Clojure** one is provided
    - Handles only **top-level** Clojure structures (functions, namespace definitions, etc)
    - Doesn't parse top-level Call-Sites (totally **ignored**)
    - Very **simple** (~40 LOC)
  - There is a half-baked **3rd party** Java Analyzer

Language Support - Thoughts

level of code understanding

- text
  (simply another CVS)
- syntax
  (what codeq currently does)
- semantics
  (might need expensive analysis of whole program)

Activity

- Main support only from their team
- Sporadic commits
  - Last useful commit, months old
  - Not a single issue (out of 8) addressed
- Weak user community

Installation

- Download **Datomic** free edition
- Start Datomic's **Transactor**
- Build **codeq** with Clojure specific **Leiningen** build tool
- Run codeq over a **git repo**
  - sends the code segments to the Transactor
- Start querying!

Usage Experience - Pros

- It is **simple**
- It just works
- Quite **fast**
  - Uses directly git capabilities
- **Incremental** support
- Makes both **git repos** & the actual **code history** queryable

Usage Experience - Cons

- It is **simple**
- More of a **proof-of-concept** than an actual tool
- Naive Clojure Analyzer
- Tightly tied to Datomic
    - On an implementation level
    - On a concept level, any db would do
- Codeq redundancy even if unchanged
- Codeq name-changes end its history

Mining Software Repositories

commits & metadata -> co-changing files -> frequent patterns between files

commits -> **AST (XML)** -> evolutionary change data

file structure & config files (filtering) -> detection of clones (similar code segments)

source files -> code duplication, code quality

git API -> queryable database (like codeq)

crash reports & **bug reports** & **commits** -> crash correlations, bug localization

bytecode -> class / packages graph & dependencies

source code & AST -> usage of dynamic features

commits & AST & **POS system** -> documentation from identifiers

bug tracking system & **IR** (information retrieval) **techniques** -> bug localization

binaries -> infer usage of libraries and other binaries