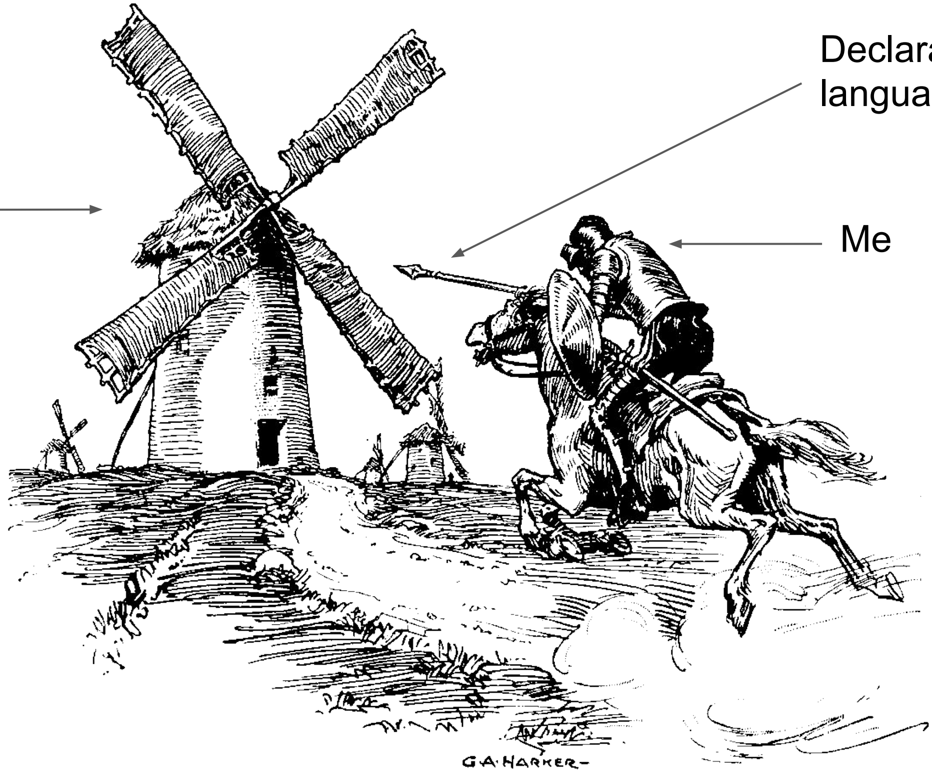
3 things I wish I knew when I started designing languages

About “poor me”



About the real me

Flunked trig, flunked chem, never took calculus or physics.

Graduated HS with a 2.8 GPA

Bachelor of Arts in English Literature

3 years as an editor; 2 as a DBA; 5 as a software engineer before grad school

I am not nor was I ever a PL researcher

This talk is about me (designing a language)

1. **Misgivings**: how I almost never began
2. **Lucky guesses**: things I got right
3. **Discoveries**: stuff I learned along the way

Prelude: misapprehensions and misgivings

how we FUD ourselves out of language design

1: The Look

An audacious new language should look unique!

```
2PC

{request, Xact, Client} ->
        if (valid(request)) then
                multicast(Members, {prepare, Xact, Myaddress});
        end if
{prepare, Xact, Coordinator} ->
        if (exec(Xact)) then
                send(Coordinator, {vote, Xact, y, Myaddress});
        else
                send(Coordinator, {vote, Xact, n, Myaddress});
        end if

{vote, Xact, Vote, Agent} ->
        // voting...
        if (Vote == n) then
           send(Members, {status, Xact, abort});
        else
                // local state, etc.
           if () then

                end
        end if

{status, Xact, Status, Coordinator} ->
        if (Status == abort) then
                abort(Xact);
        else
                commit(Xact);
        end if
        send(Coordinator, {ack, Xact, Myaddress});

{ack, Xact, Agent} ->
```

1: The Need

some bicyclist[sic]
@palvaro

Replying to @palvaro @cmeik

P: you're peter alvaro?
Me:
P: yo
distr
Me:

5:24 PM

1 Retwe

1 1 7

Tweet your reply

some bicyclist[sic] @palvaro · 16 Sep 2016
Replying to @palvaro @cmeik
P: *pointing derisively* HAHAHAHAHAHAHAHA!

5 9

some bicyclist[sic] @palvaro · 16 Sep 2016
(HPTS 2009)

the heart rears wings bold and ...
@palvaro

last year carl hewitt honored me by walking

11:45 PM - 27 Apr 2013

▲ jerf on Apr 8, 2011 [-]

I get where you are coming from, and it's a good plan, as long as the plan is to eventually fully detach from Ruby. Being even two or three times as fast as Ruby, which seems to be an optimistic interpretation of JRuby's performance, is still starting from a terrible position in so many ways.

I don't get the idea that some people seem to have that performance doesn't matter for distributed systems, when the truth is the exact opposite. Desktops and even cell phones, we see a great deal of sloppiness around performance, because it doesn't really matter that much. Small servers or small clusters, we still say throw more hardware at it and just hack some stuff together for clustering. But when you're serious about distributed systems is also when you are counting every one of something; maybe disk hits, maybe CPU cycles, maybe bytes of RAM, but there is something you are obsessing over. And maybe you're obsessing over more than one of these at once, all with an intensity that would credit an Atari 2600 programmer. (Facebook apparently published the specs for their machines today. Tell me they aren't too concerned about performance.) I'm not sure leaving performance for later is a good idea, they may well iterate their way into a cool abstraction that will *never* perform. Designing a distributed system abstraction without worrying about performance strikes me as about as sensible as designing a new 3D framework without worrying about performance... not necessarily a fatal flaw but I sure hope you have a good plan.

the heart rears wings bold and ...
@palvaro

(roughly transcribed) OH: "in racket, everything is parenthesis. what is the thing in your language that is everything and that I don't buy?

11:12 AM - 8 Apr 2015

a library?

▲ chewzerita 5 months ago [-]

How is this different from elixir?

▲ CoffeeDregs on Apr 8, 2011 [-]

I like it! umm... but what the hell is it? I STFA (skimmed-the-f*ing-article) and don't know what's going on here. Quick! What does this mean?

1: The Impact

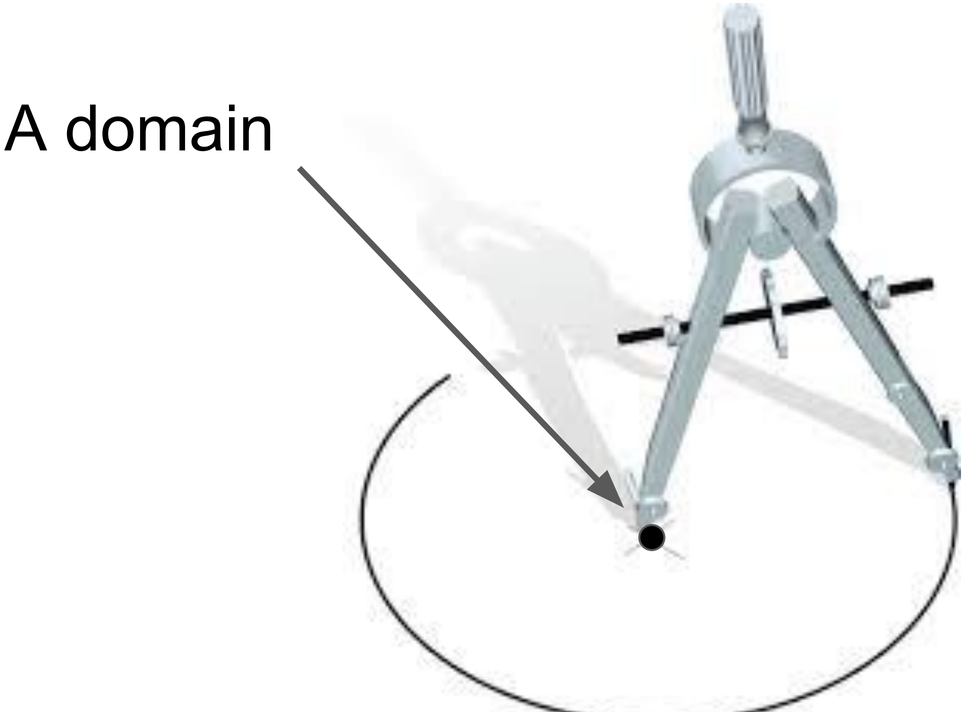
Lucky guesses: things I got right

Lucky guess 1:

context context context context context context
context context context context context context
context context context context context context
context context context context context context
context context context context context context
context context context context context context
context context context context context context
context context context context context context

Every language is a DSL

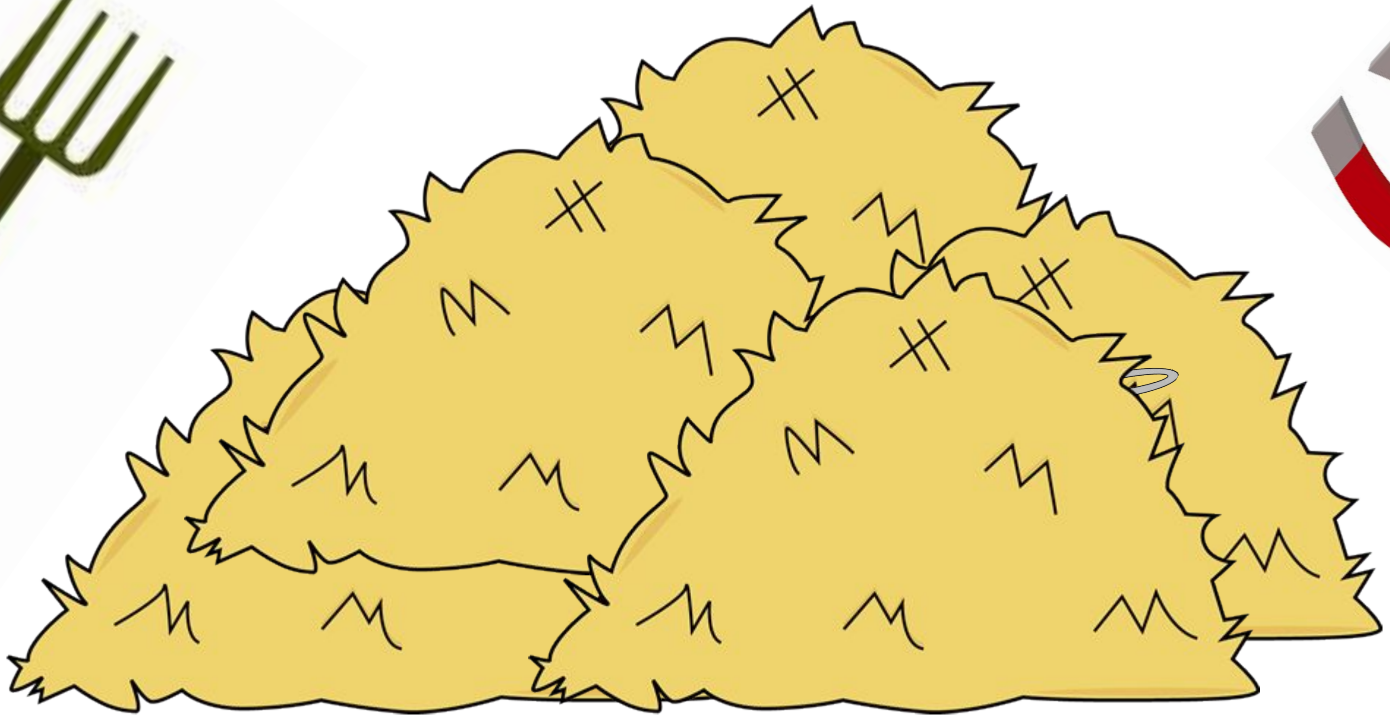
Thy firmness makes my circle just



More lucky guesses: a damn problematic domain

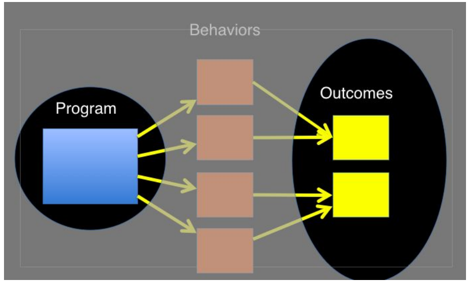
$$P_{real}(success) = F\left(\frac{perceived\ crisis}{perceived\ pain\ of\ adoption}\right)$$

Hiding and illuminating

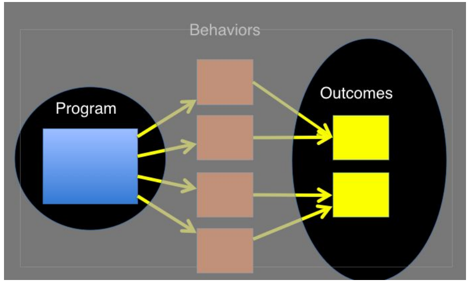
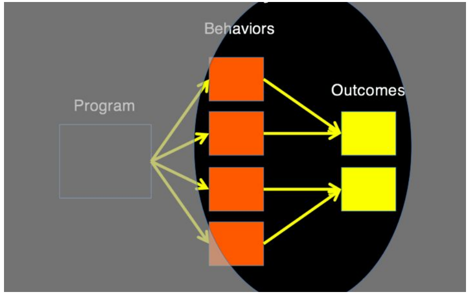


What is **damn hard** about this domain?

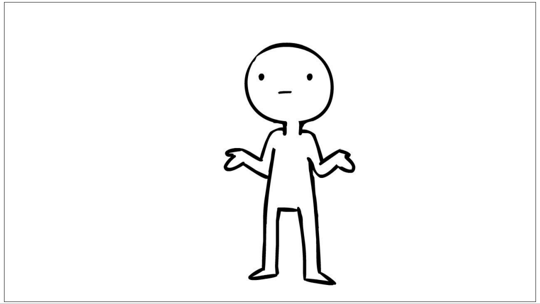
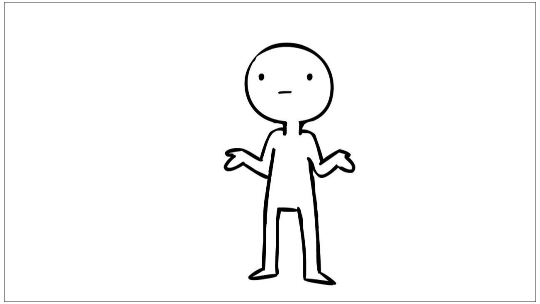
Program correctness



What is **damn hard** about this domain?

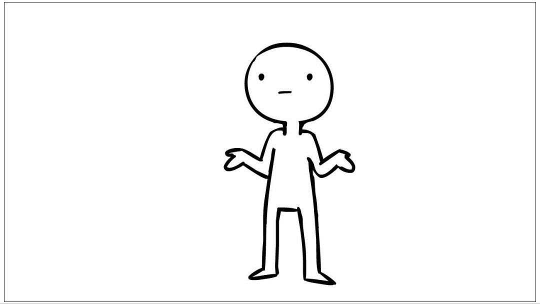
<table>
<tr><td>Program correctness

</td><td>Debugging

</td></tr>
<tr><td></td><td></td></tr>
</table>

What is **damn hard** about this domain?

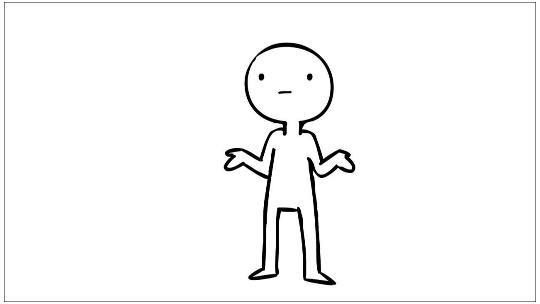
Program correctness	Debugging
	

What is **damn hard** about this domain?

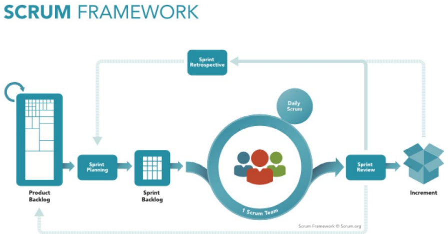
Program correctness

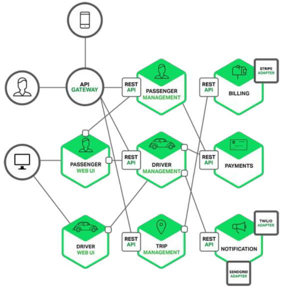


Debugging

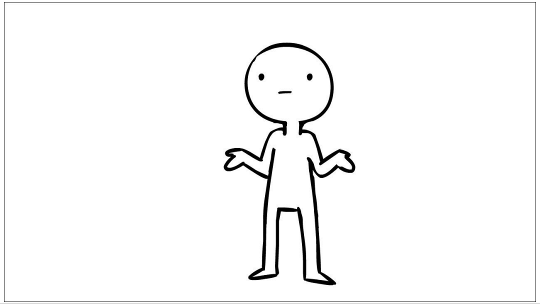
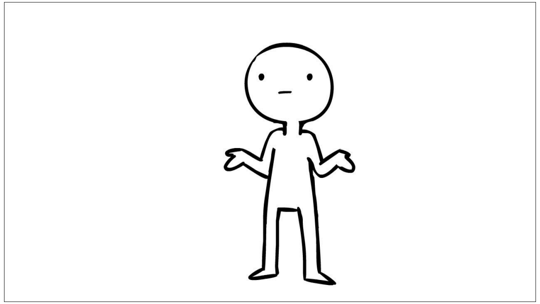


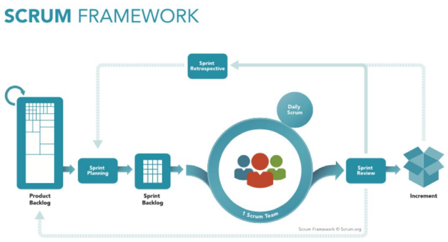
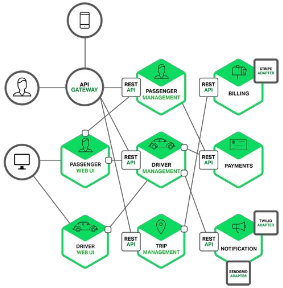
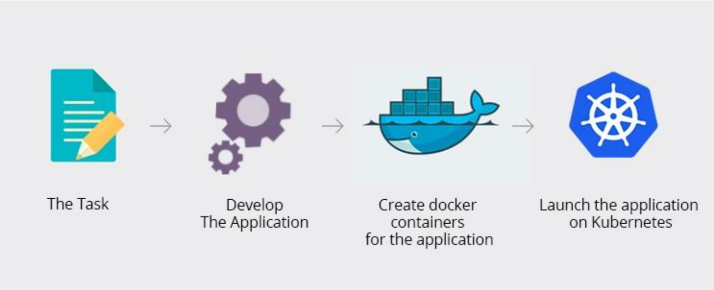
Maintenance and extensibility



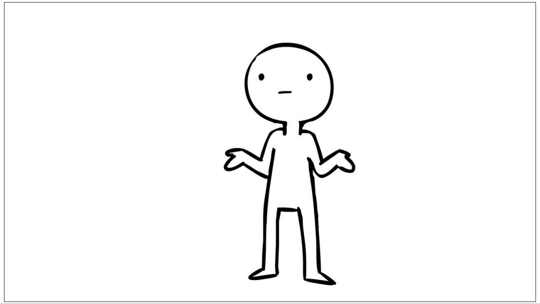
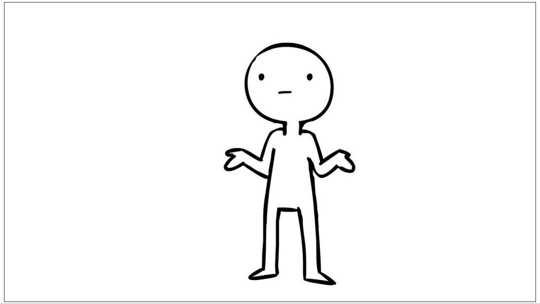


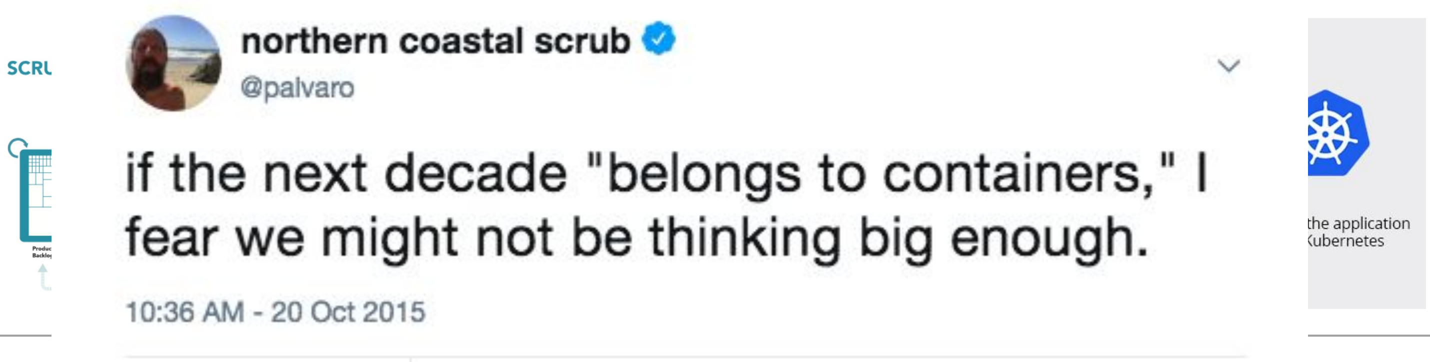
What is **damn hard** about this domain?

Program correctness	Debugging
	

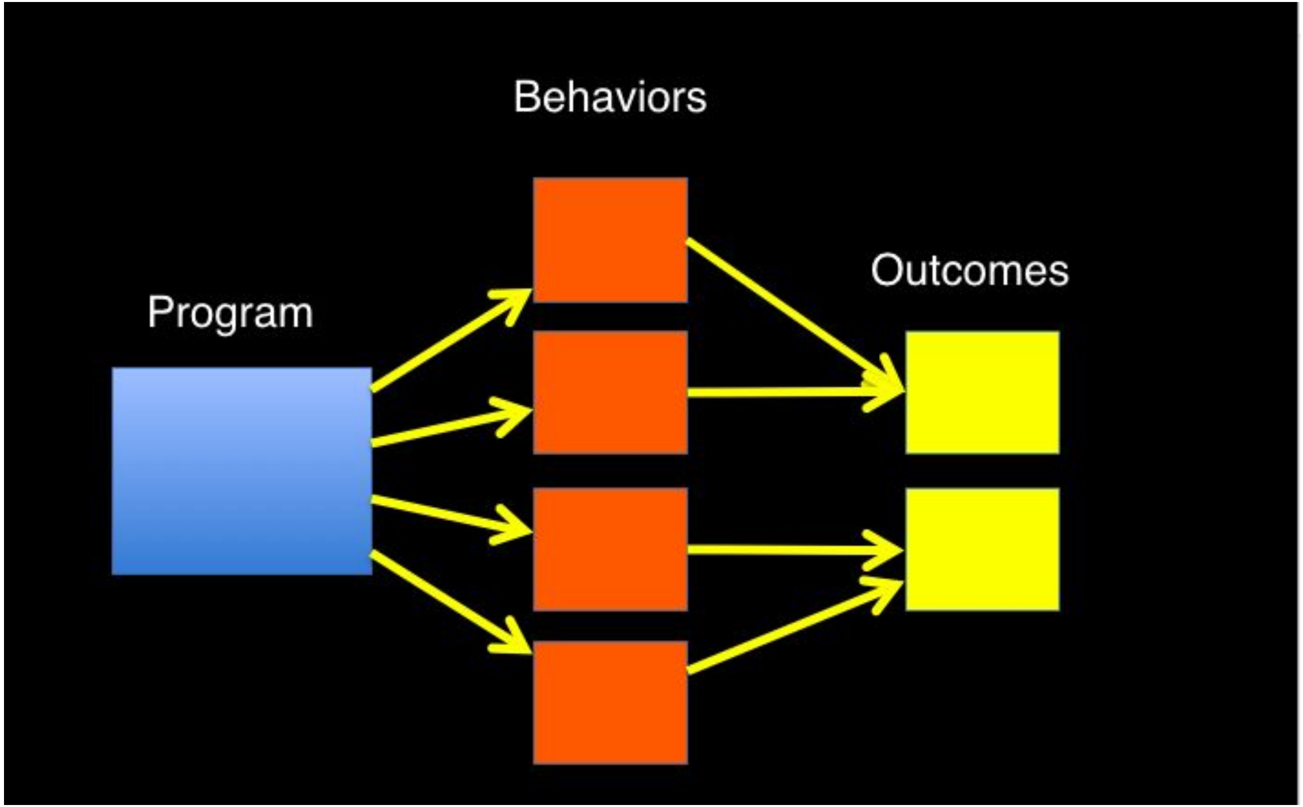
Maintenance and extensibility	Heterogeneity and portability
 	

Rearranging the deckchairs...

Program correctness	Debugging
	



Why so damn hard?



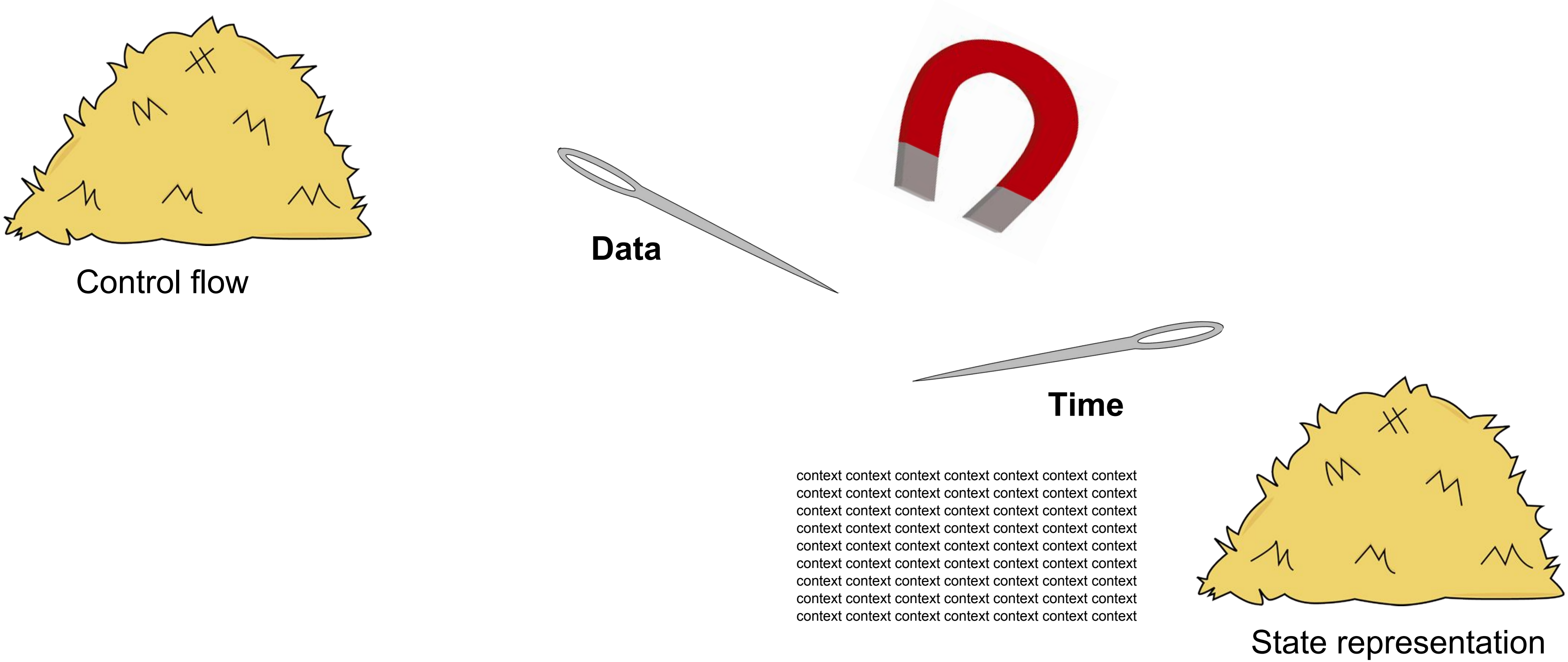
The right language would focus our attention on

How **data** flows through the system;

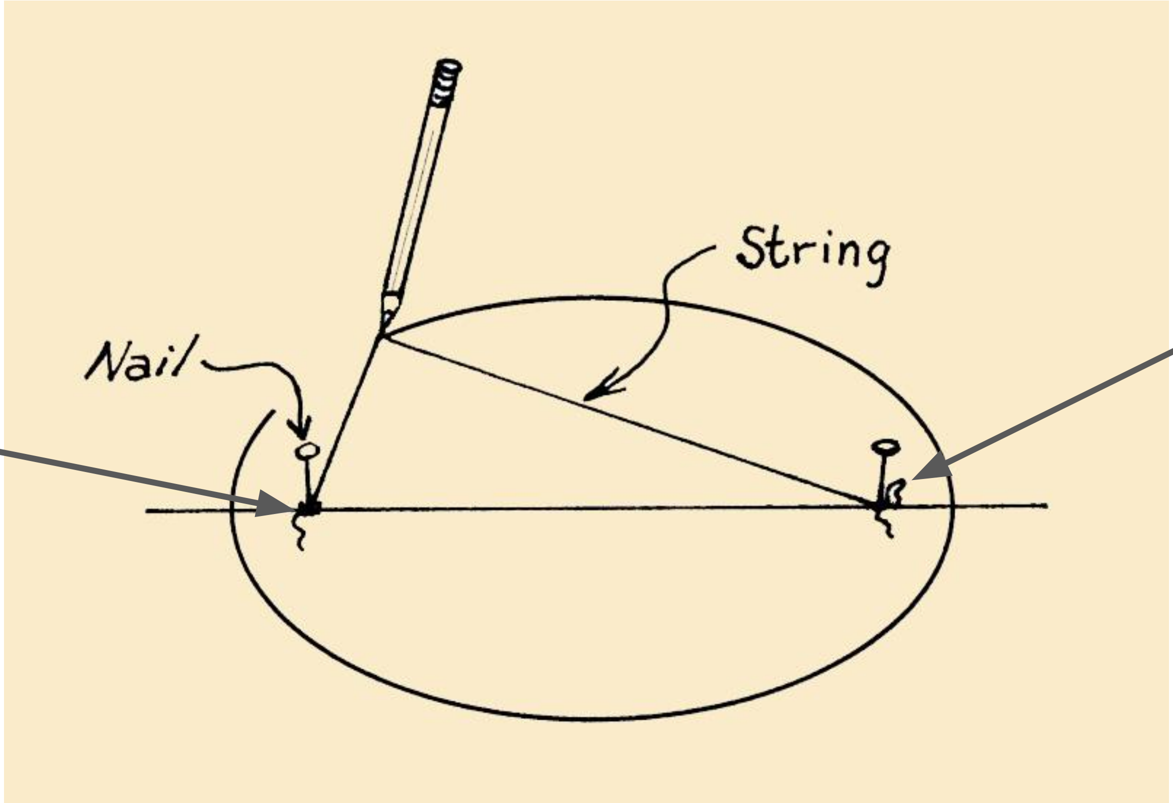
How it is allowed to change over **time**;

Where and when we can control how it changes and when we can't.

Everything else, arguably, is a distraction


Control flow
Data
Time
context context context context context context context
context context context context context context context
context context context context context context context
context context context context context context context
context context context context context context context
context context context context context context context
context context context context context context context
context context context context context context context
context context context context context context context
State representation

Thy firmness makes my circle just



A domain

An opinion

Many moons passed

so I'm sitting in the bath reading lamport, like I do most nights. and it occurs to me...

are we thinking about this wrong, trying to figure out how to map the doughface language onto an overwriting implementation? it occurs to me that we should have this:

1. all facts (instantaneous events) are stored (almost) eternally by appending to a log. this log is timestamp order, obviously, but we could imagine indexes into it.
2. all a program does is define what is true when, based on what is already true then.   a predicate p is true at a given time N if it was a fact given at that time (cheap to look up since our log is sorted), or if it can be proven that a tuple has carried over from a previous time (i.e., if (\exists a tuple A \in p@M s.t. M < N) and (~\exists a tuple B \in del_p@O s.t. M < O < N).

statement structure:

```
head(Arg1, Arg2, [...]) :-
        Op1 {
                clause1(ArgN, [...]),
                ArgN > X;

                clause2(ArgN, [...]),
        },

        Op2 {


        };
```

backing up. what are reasonable state primitives for a programming language?

Vol1, p. 23

"encapsulation [...] appears antithetical to declarativeness.

talk about the data: name it, talk about how it changes.

This is all very well and good in a datalog program, which is evaluated over a static set of ground facts until there are no more conclusions to be drawn. But when we introduce the notions of time and communication, from pure logic to asynchronous distributed systems, we feel uncomfortable (understandably) with the idea of rules firing in no particular order.

Inspiration

Descriptive complexity (Immerman'99)

In the beginning, there were two measures of computational complexity: time and space. From an engineering standpoint, these were very natural measures, quantifying the amount of physical resources needed to perform a computation. From a mathematical viewpoint, time and space were somewhat less satisfying, since neither appeared to be tied to the inherent mathematical complexity of the computational problem.

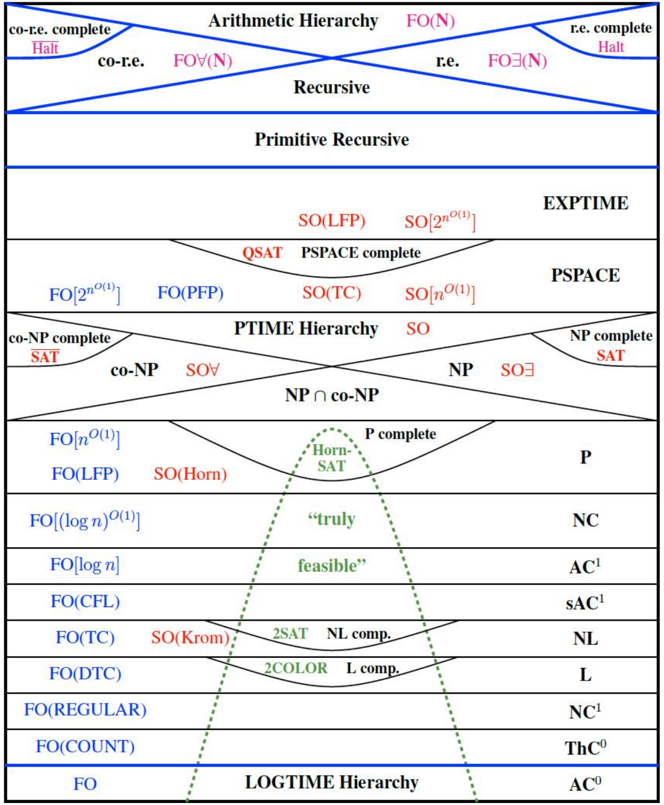
In 1974, Ron Fagin changed this. He showed that the complexity class NP — those problems computable in nondeterministic polynomial time — is exactly the set of problems describable in second-order existential logic. This was a remarkable insight, for it demonstrated that the computational complexity of a problem can be understood as the richness of a language needed to specify the problem. Time and space are not model-dependent engineering concepts, they are more fundamental.

Descriptive complexity (Immerman'99)

$$\Phi_{\text{3-color}} \equiv (\exists R^1)(\exists Y^1)(\exists B^1)(\forall x)\Big[\big(R(x) \vee Y(x) \vee B(x)\big) \wedge (\forall y)\Big(E(x,y) \rightarrow \neg\big(R(x) \wedge R(y)\big) \wedge \neg\big(Y(x) \wedge Y(y)\big) \wedge \neg\big(B(x) \wedge B(y)\big)\Big)\Big]$$

$$\Phi_{\mathrm{SAT}} \equiv (\exists S)(\forall x)(\exists y)((P(x,y) \wedge S(y)) \vee (N(x,y) \wedge \neg S(y))) .$$

Descriptive complexity (Immerman'99)



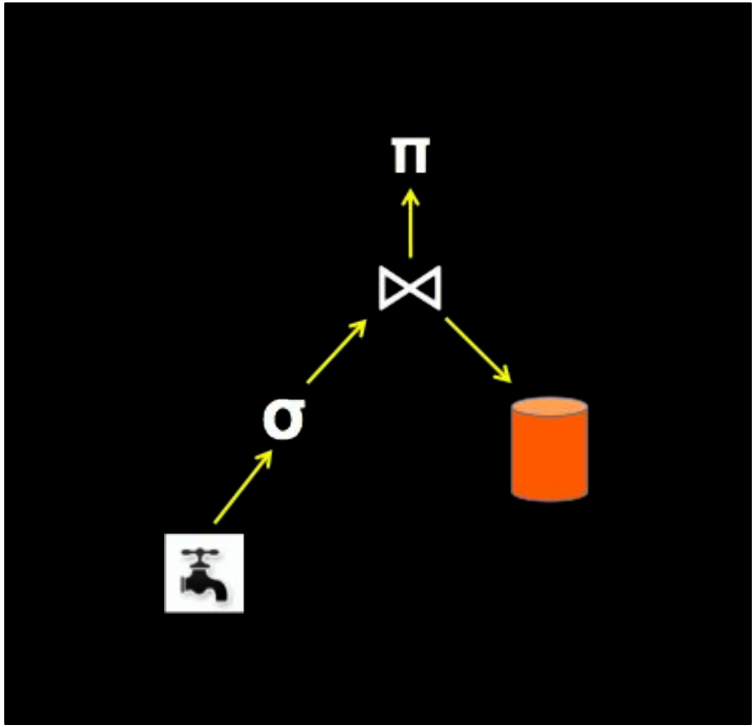
Descriptive complexity (Immerman'99)

Although few programmers consider their work in this way, a computer program is a completely precise description of a mapping from inputs to outputs. In this book we follow database terminology and call such a map a *query* from input structures to output structures. Typically a program describes a precise sequence of steps that compute a given query. However, we may choose to describe the query in some other precise way. For example, we may describe queries in variants of first- and second-order mathematical logic.

Fagin's Theorem gave the first such connection. Using first-order languages, this approach, commonly called descriptive complexity, demonstrated that virtually all measures of complexity can be mirrored in logic. Furthermore, as we will see, the most important classes have especially elegant and clean descriptive characterizations.

Queries made a neat lens...

```
create view response as
  select client, server, code, document
         from request r, page p
               where r.server  = p.server
               and r.URI = p.URI;
```



Maybe languages are really lenses

fragments

Conjunctive queries

$\neg$ **σ π ⋈** **LFP**

fragments

SQL

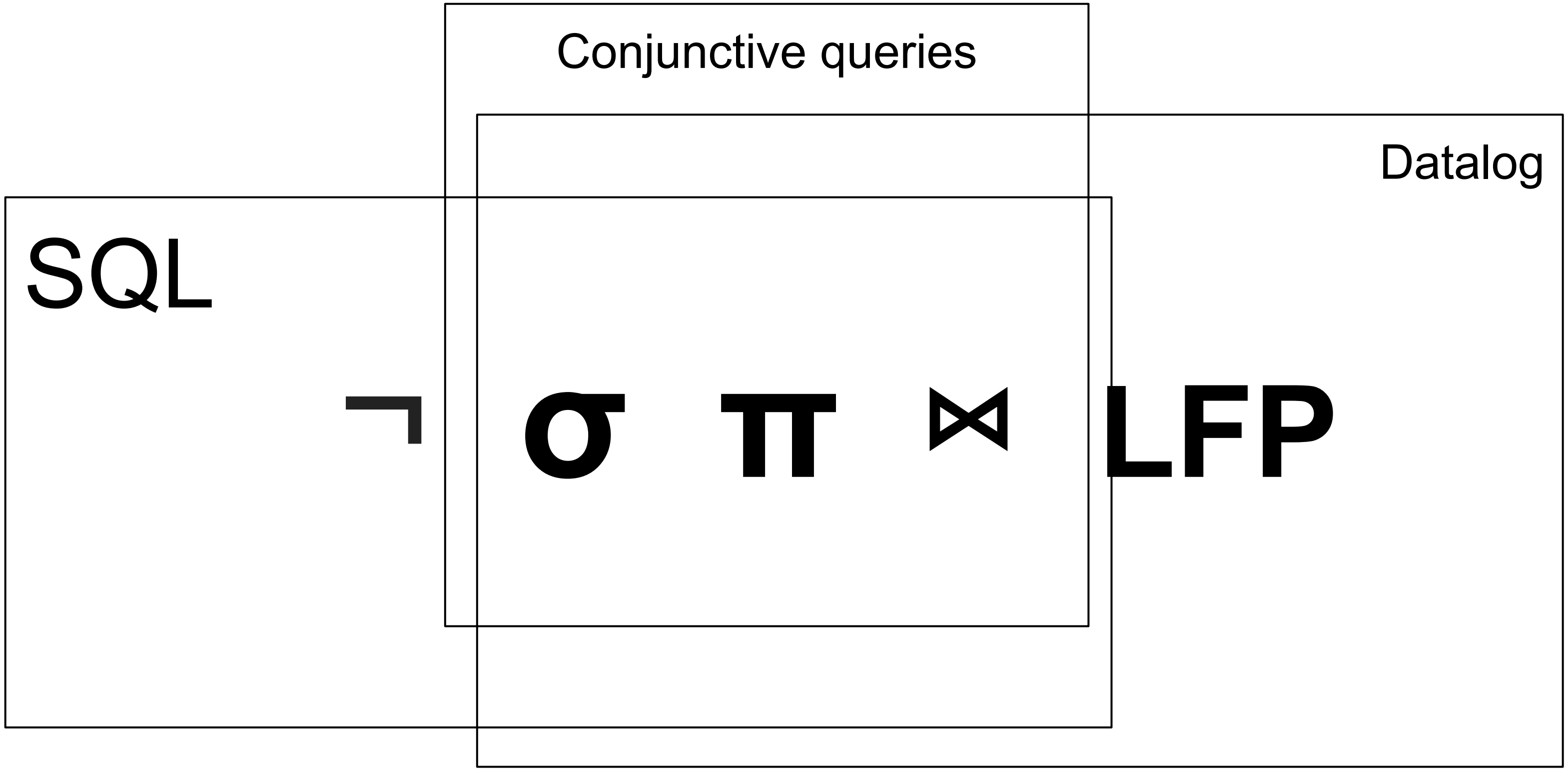
$\neg$ σ π $\ltimes$ LFP

fragments

Datalog

¬ σ π ⋈ LFP

Or maybe they are lassos



context context context context context context
context context context context context context
context context context context context context
context context context context context context
context context context context context context
context context context context context context
context context context context context context
context context context context context context

Datacentrism

knowledge("details")

Datacentrism

knowledge(host1, “details”)

Contexualized by location (space)

Datalog cannot express

Mutable state

Uncertainty

Datacentrism

knowledge(host1, “details”, 27)

Contexualized by relative order (time)

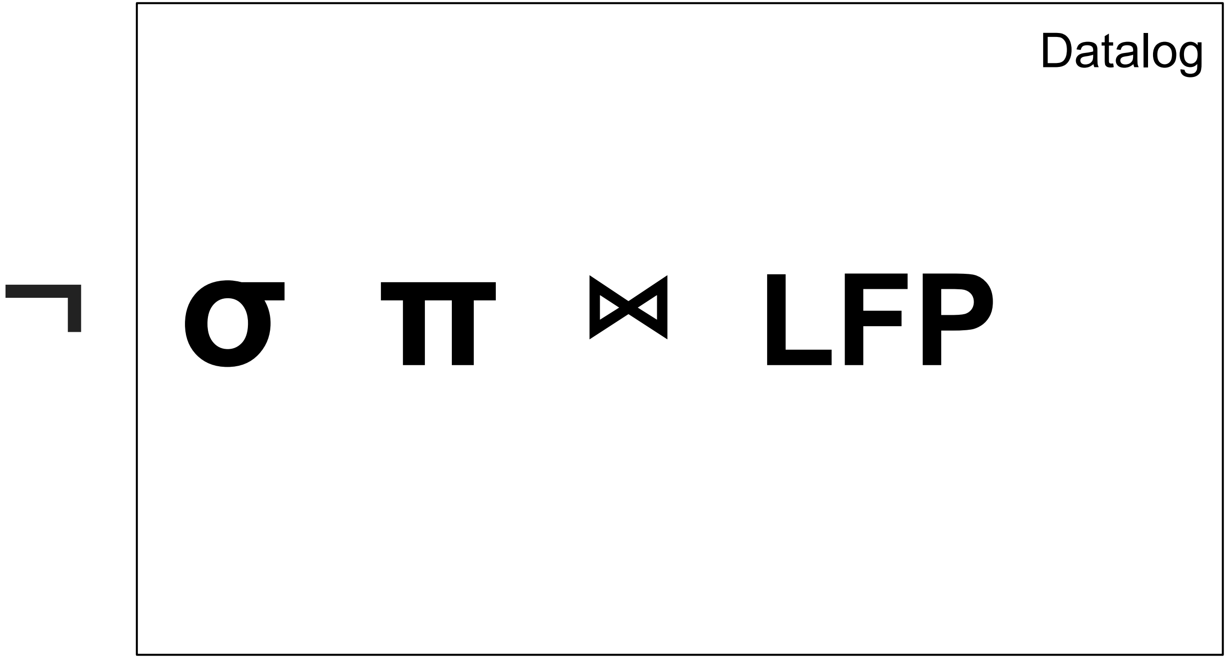
Datacentrism

register(host1, “current value”, 27)

Datacentrism

kvs(host1, key, “current value”, 27)

Or maybe they are lassos



Or maybe they are lassos

"Statelog"

¬ σ π ⋈ LFP

+1

Dedalus can express it all. but...

Dedalus

¬ σ π ⋈ LFP

+1 ND choice

Dedalus al… on't want to say; requi… her not use

Paxos

```
nodes(A, N, I)@next :- nodes(A, N, I);
seed(A, S)@next :- seed(A, S), notin update_seed(A);
seed(A, S+C)@next :- seed(A, S), update_seed(A), agent_cnt(A, C);

prepare(B, A, S, M)@async :- proposal(A, M), seed(A, S), nodes(A, B, _);
update_seed(A) :- proposal(A, _);

redo(A, M) :- timeout(A, M), notin accepted(A, _, M);
prepare(B, A, S, M)@async :- redo(A, M), seed(A, S), nodes(A, B, _);
timer_svc(A,M,3) :- redo(A, M);
update_seed(A) :- redo(A, M);

response_log(C, A, S, O, Os) :- prepare_response(C, A, S, O, Os);
response_log(C, A, S, O, M)@next :- response_log(C, A, S, O, M);

// workaround for the fact that c4 can't count strings!
//response_cnt(C, S, count<A>) :- response_log(C, A, S, O, Os);
response_cnt(C, S, count<I>) :- response_log(C, A, S, O, Os), nodes(C, A, I);

best(C, S, max<Os>) :- response_log(C, A, S, O, Os);
what(C, I) :- nodes(C, _, I);
//agent_cnt(C, count<I>) :- nodes(C, _, I);
agent_cnt(C, count<I>) :- what(C, I);

accept(A, S, O)@async :- agent_cnt(C, Cnt1), response_cnt(C, S, Cnt2),
                         response_log(C, _, S, O, Os), best(C, S, Os), nodes(C, A, _), Os != 1, Cnt2 > Cnt1 / 2;
accept(A, S, P)@async :- agent_cnt(C, Cnt1), response_cnt(C, S, Cnt2), response_log(C, _, S, O, Os),
                         best(C, S, Os), my_proposal(C, P), nodes(C, A, _), Os == 1, Cnt2 > Cnt1 / 2;


// acceptor
dominated(A, S) :- prepare(A, _, S, _), prepare_log(A, S2, _), S2 > S;
can_respond(A, C, S, M) :- prepare(A, C, S, M), notin dominated(A, S);
prepare_response(C, A, S, O, Os)@async :- can_respond(A, C, S, M), accepted(A, Os, O), highest_accepted(A, Os);
prepare_response(C, A, S, "anything", 1)@async :- can_respond(A, C, S, M), notin accepted(A, _, _);


highest_accepted(A, max<S>) :- accepted(A, S, _);
accepted(A, S, M) :- accept(A, S, M);
accepted(A, S, M)@next :- accepted(A, S, M);


prepare_log(A, S, M) :- prepare(A, _, S, M);
prepare_log(A, S, M)@next :- prepare_log(A, S, M);
```

2PC

```
cancommit(Agent, Coord, Xact)@async :- begin(Coord, Xact), agent(Coord, Age
vote_msg(Coord, Agent, Xact, "Y")@async :- cancommit(Agent, Coord, Xact), c
vote(C,A,X,S) :- vote_msg(C,A,X,S);

timer_svc(A, X, 4) :- cancommit(A, _, X);

// the coordinator is the distinguished node that is not an agent...
abort(A, X)@next :- timeout(A, X), notin coordinator(A, A), notin precommit


authorized(C, X) :- vote_msg(C, _, X, "Y"), notin missing_vote(C, X), notin

precommit(A, C, X)@async :- authorized(C, X), agent(C, A);
ack(C, A, X)@async :- precommit(A, C, X), prepared(A, C, X, "Y");
timer_cancel(A, X) :- precommit(A, _, X), prepared(A, _, X, "Y");
timer_svc(A, X, 4) :- precommit(A, C, X), prepared(A, C, X, "Y");

//commit(A, X)@next :- timeout(A, X), precommitted(A, C, X), notin abort(A,
commit(A, X) :- timeout(A, X), precommitted(A, C, X), notin abort(A, X);

precommitted(A, C, X) :- precommit(A, C, X);
precommitted(A, C, X)@next :- precommitted(A, C, X);

abort(C, X)@next :- vote(C, _, X, "N");
commit(C, X)@next :- ack(C, _, X), notin missing_ack(C, X), notin abort(C,

missing_ack(C, X) :- agent(C, A), running(C, X), notin ack(C, A, X);
missing_vote(C, X) :- agent(C, A), running(C, X), notin vote(C, A, X, "Y");

prepared(A, C, X, "Y") :- cancommit(A,C,X), can(A,X);
prepared(A, C, X, Y)@next :- prepared(A,C,X,Y);


timer_svc(C, X, 5) :- begin(C, X);
abort(C, X)@next :- timeout(C, X), coordinator(C, C), missing_ack(C, X), no


commit(A, X)@async :- commit(C, X), agent(C, A), notin abort(C, A);
abort(A, X)@async :- abort(C, X), agent(C, A);
```

3PC

```
d(A);
, agent_cnt(A, C);

ed(A, S), nodes(A, B, _);

_, M);
, S), nodes(A, B, _);

e(C, A, S, O, Os);
g(C, A, S, O, M);

strings!
C, A, S, O, Os);
 A, S, O, Os), nodes(C, A, I);

 Os);

sponse_cnt(C, S, Cnt2), response_log(C, _, S, O, Os), best(C, S, Os), nodes(C,
sponse_cnt(C, S, Cnt2), response_log(C, _, S, O, Os), best(C, S, Os), my_propo

_log(A, S2, _), S2 > S;
 notin dominated(A, S);
espond(A, C, S, M), accepted(A, Os, O), highest_accepted(A, Os);
:- can_respond(A, C, S, M), notin accepted(A, _, _);

);

M);

my_proposal(A, P) :- proposal(A, P);
```

Waiting requires counting

Counting requires waiting

(Joe Hellerstein)

¬

Waiting requires counting

Nonmontonicity required to express coordination

Counting requires waiting

Coordination required to tolerate nonmonotonicity

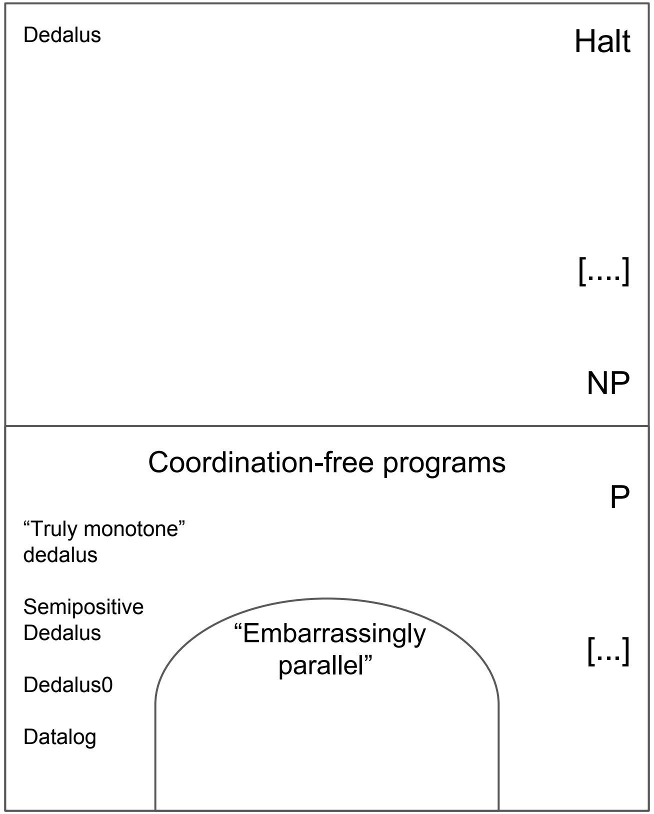
Or maybe they are lassos

CALM Dedalus

¬ σ π ⋈ LFP

+1 ND choice

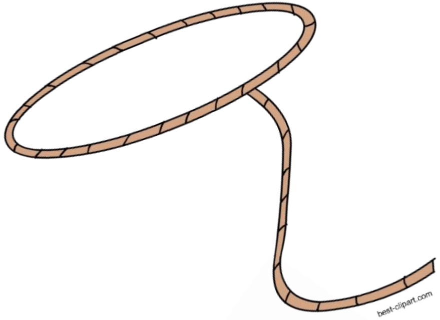
Pop descriptive complexity

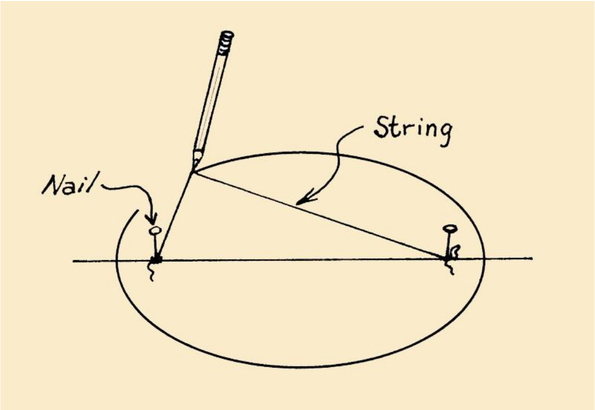


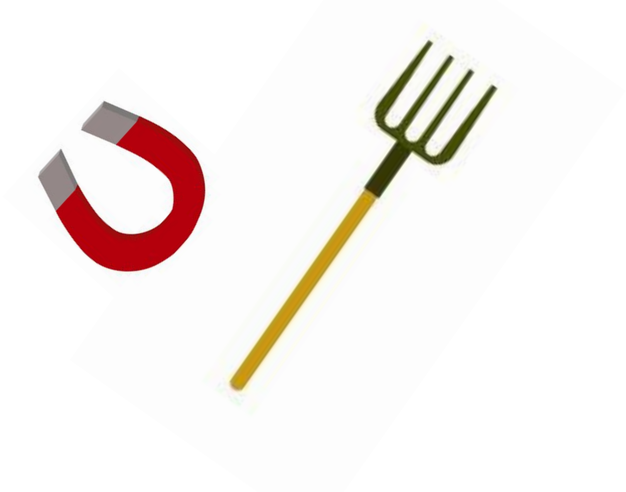
Discoveries: stuff I learned along the way

Languages and the design process

context context context context context context context context context context
context context context context context context context context context context
context context context context context context context context context context
context context context context context context context context context context
context context context context context context context context context context
context context context context context context context context context context
context context context

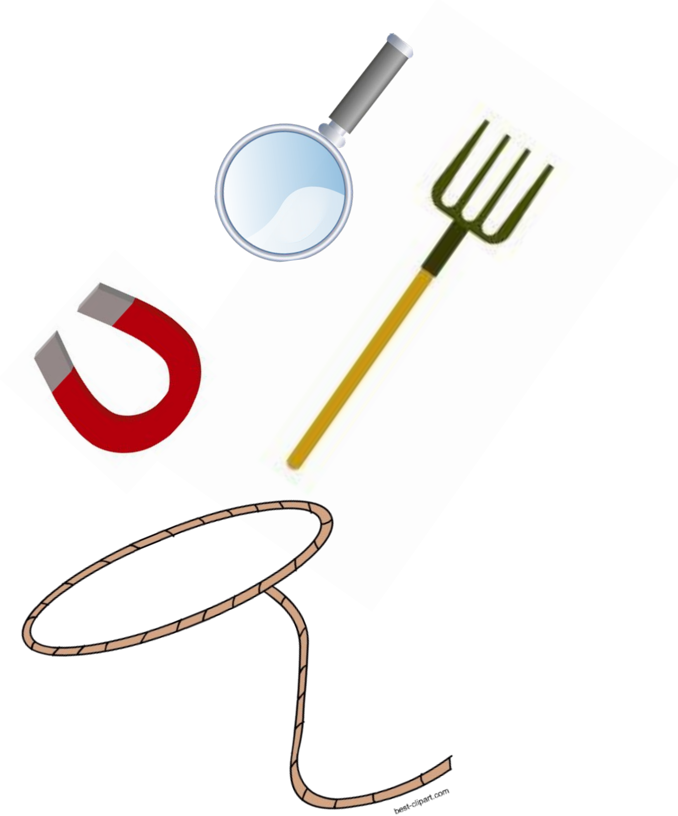






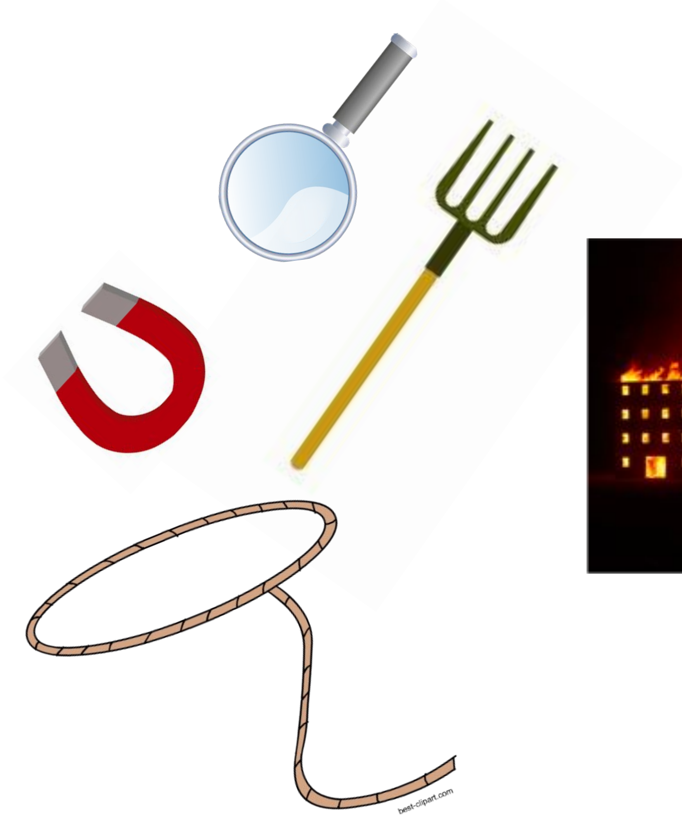
Languages and the design process

Subject



Object

Languages and the design process



Discoveries: stuff I learned along the way

The look

Discoveries: stuff I learned along the way

~~The look~~ it's about the *fit*

Discoveries: stuff I learned along the way

~~**The look**~~ it's about the *fit*

The need

Discoveries: stuff I learned along the way

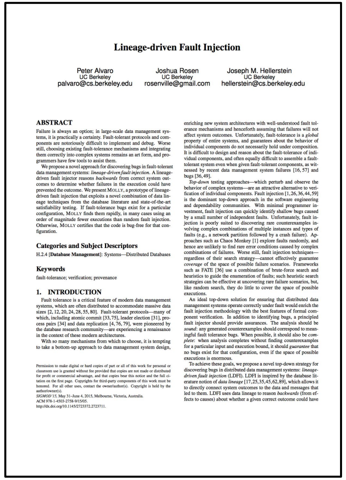
~~The look~~ it's about the *fit*

~~The need~~ it's about *our* need

Discoveries: stuff I learned along the way

~~The look~~	it's about the *fit*
~~The need~~	it's about *our* need
The impact	well….


DEDALUS: Datalog in Time and Space
Peter Alvaro William R. Marczak Neil Conway Joseph M. Hellerstein David Maier Russell Sears
University of California, Berkeley Portland State University
ABSTRACT
1. INTRODUCTION

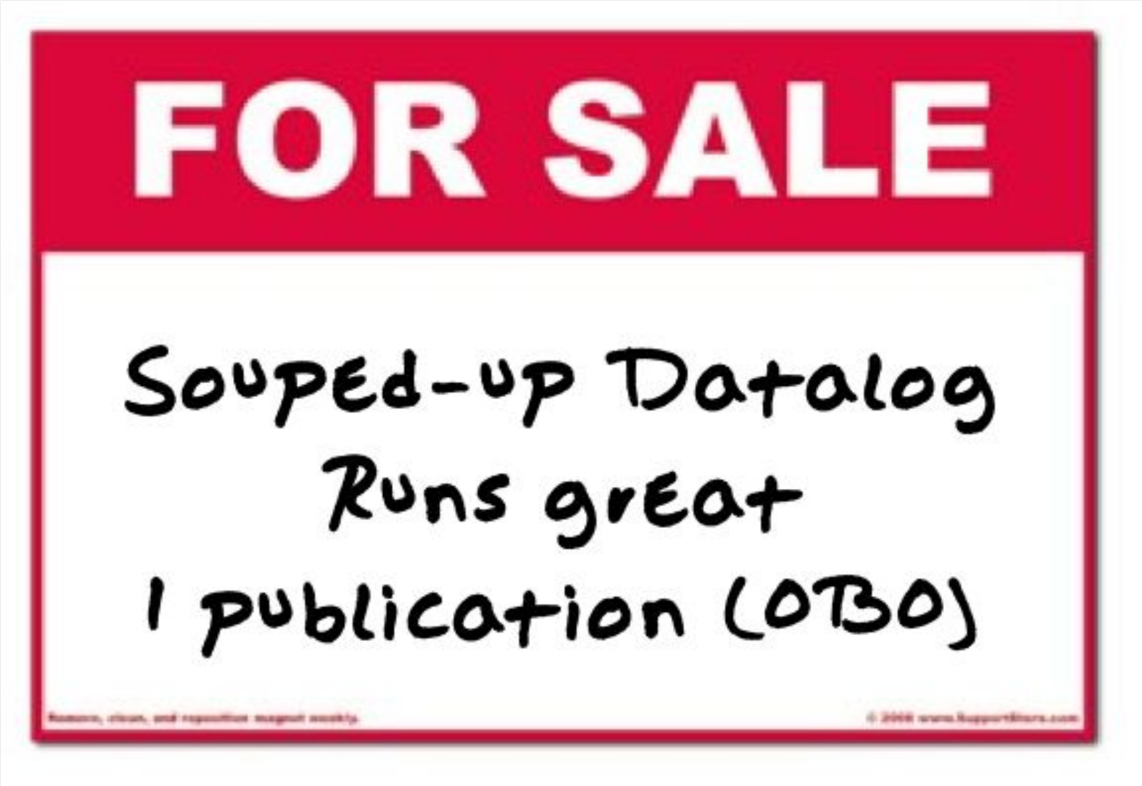

Lineage-driven Fault Injection
Peter Alvaro UC Berkeley palvaro@cs.berkeley.edu
Joshua Rosen UC Berkeley rosenville@gmail.com
Joseph M. Hellerstein UC Berkeley hellerstein@cs.berkeley.edu
ABSTRACT
Categories and Subject Descriptors
Keywords
1. INTRODUCTION


Growing a protocol
Kamala Ramasubramanian, Kathryn Dahlgren, Asha Karim, Sanjana Maiya, Sarah Borland, Boaz Leskes, Peter Alvaro
University of California, Santa Cruz
Elastic
Abstract
1 Introduction


Consistency Analysis in Bloom: a CALM and Collected Approach
Peter Alvaro, Neil Conway, Joseph M. Hellerstein, William R. Marczak
{palvaro, nrc, hellerstein, wrm}@cs.berkeley.edu
University of California, Berkeley
ABSTRACT
1. INTRODUCTION
2. CONSISTENCY AND LOGICAL MONOTONICITY (CALM)


Fixed It For You:
Protocol Repair Using Lineage Graphs
Lennart Oldenburg, Xiangfeng Zhu, Kamala Ramasubramanian, and Peter Alvaro
University of California, Santa Cruz Technische Universität Berlin
Abstract
1. INTRODUCTION

Poor lucky me



BTW: Where’s the lie?