Rustling up predictive sports-betting models on the BEAM

Hello!

I am Dave Lucia

VP, Engineering at SimpleBet

Running Elixir in production since 2016

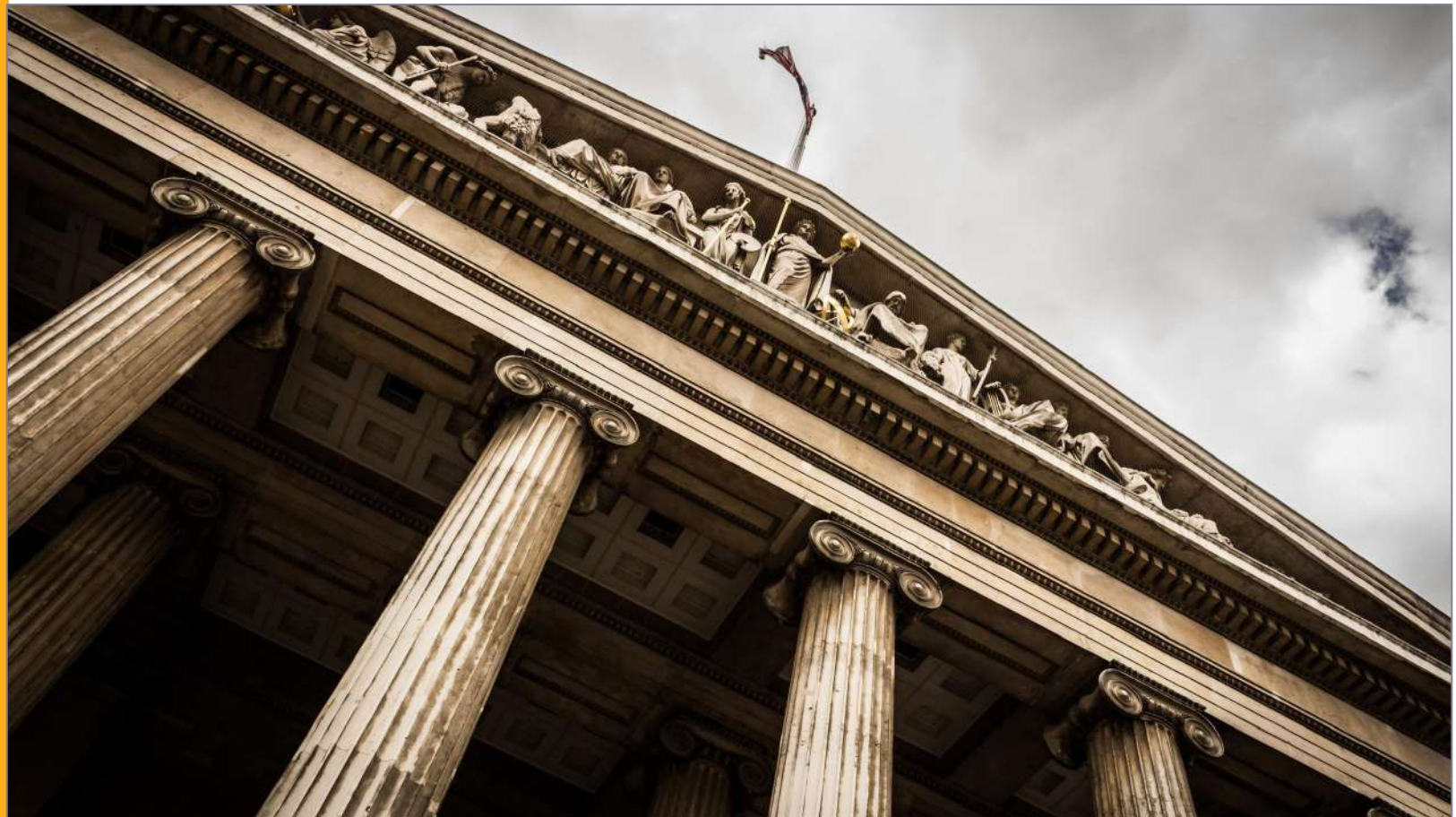
Topics

- Sports Betting
- Machine Learning on the BEAM
- One really big NIF
- Organizational design

Sports Betting Today

Where do the odds come from?

United States - PASPA



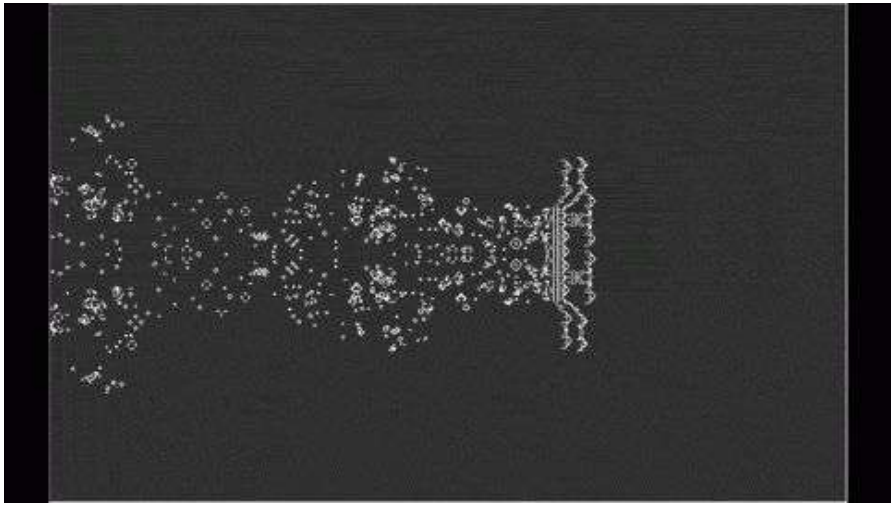
Terminology

- Market – An opportunity to bet, e.g. Yankees - Red Sox moneyline
- Bet / Wager – Taking a stance on one side of a market with monetary stake
- Stake – Amount of $$$ placed on a bet
- Selections – The options to bet on for a market, e.g. under or over
- Odds – Potential $$$ of the bet wins
- Sportsbook – Company that accepts bets/wagers

Vegas Line - Set by “some dude” in Vegas

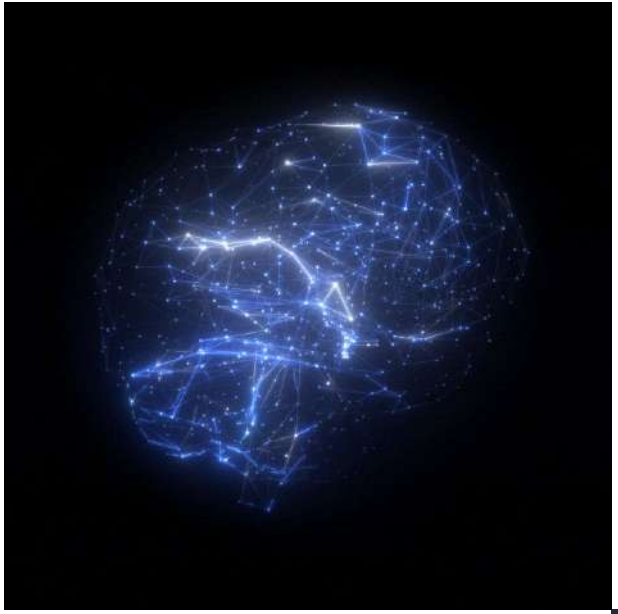
Simulation approach

- Monte Carlo simulation algorithms will simulate the outcome of the game
- Simulation runs for 10-100k iterations
- Count statistics for each iteration
- Statistic / # of iterations is the probability



Machine Learning approach

- Design a model with “features” and a prediction function
- Train on historical data
- Predict using the trained model parameters combined with the live data



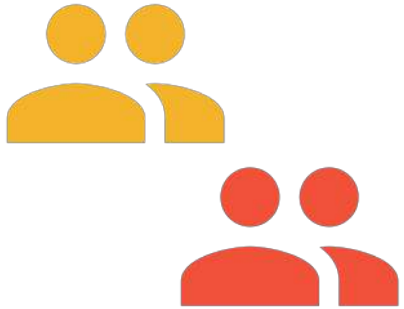
Sports Betting

Traditional markets

Moneyline - Who will win, team A or team B?

Spread - Will team A beat team B by X points?

Over/Under - Will team A and B score more or less than X total points?



11

10

9

The “line”

8

7

6

5

Moneyline - Odds

Selection	Implied Odds	Decimal Odds	American Odds
Team A	40%	2.5	150
Team B	60%	1.667	-150

Sorry, but the house gotta make $$$

Moneyline - Odds with applied overround

Selection	Implied Odds	Decimal Odds	American Odds
Team A	40%  45%	2.5  2.222	150  122
Team B	60%  65%	1.667  1.538	-150  -186

- ★ 10% applied overround
- ★ 9.09% vig

SimpleBet

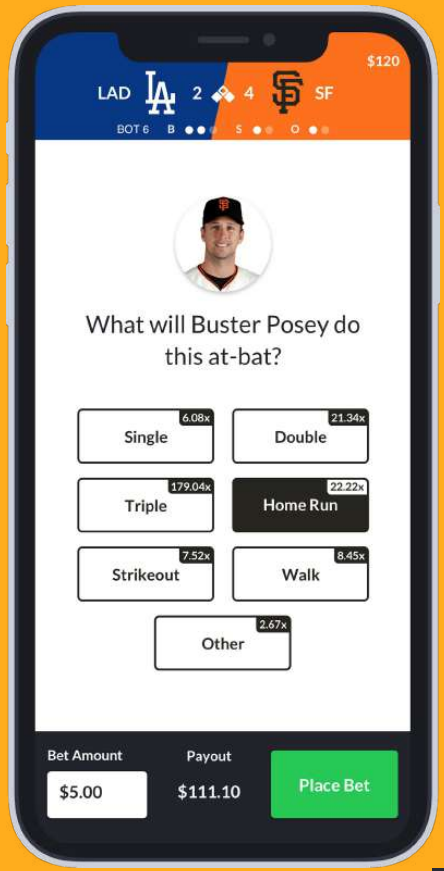
Turning every moment into a betting opportunity

“

SimpleBet makes in-play, discrete occurrence markets possible by combining machine learning and automation

Moment-based markets

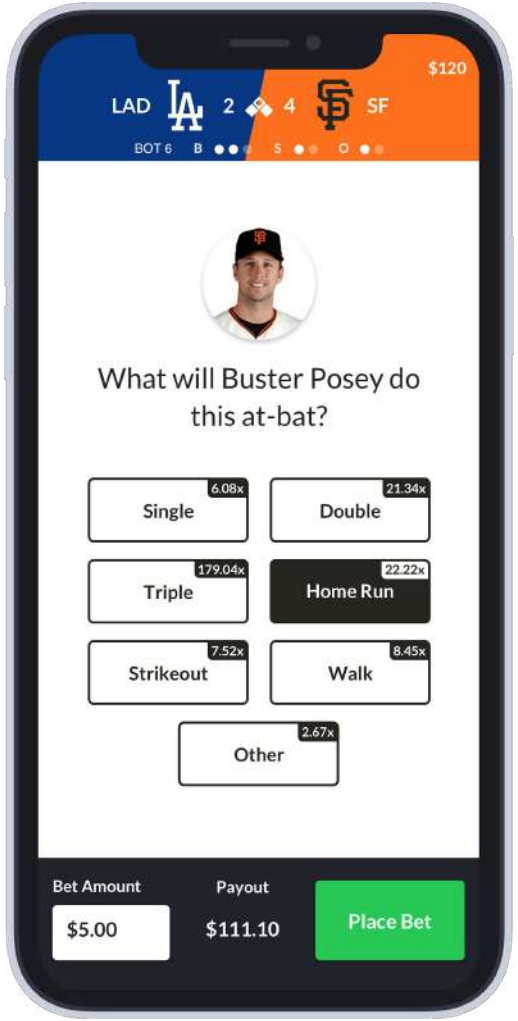
- In-play, discrete occurrences
- Bet while the game happens
- Know the outcome within minutes or seconds of placing the bet



Baseball

Plate Appearance Exact

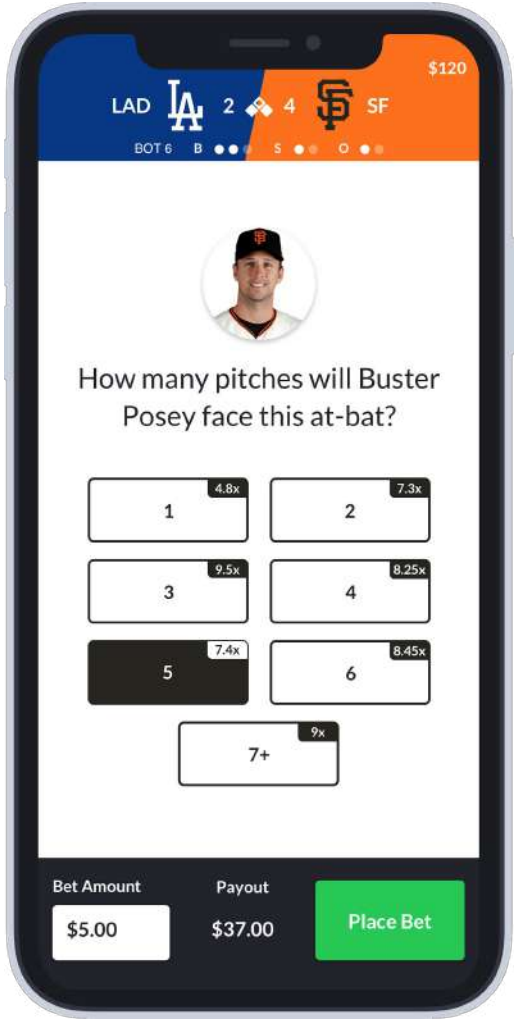
What will be the outcome of this plate appearance?



Baseball

Pitch Count

How many pitches will be in this plate appearance?



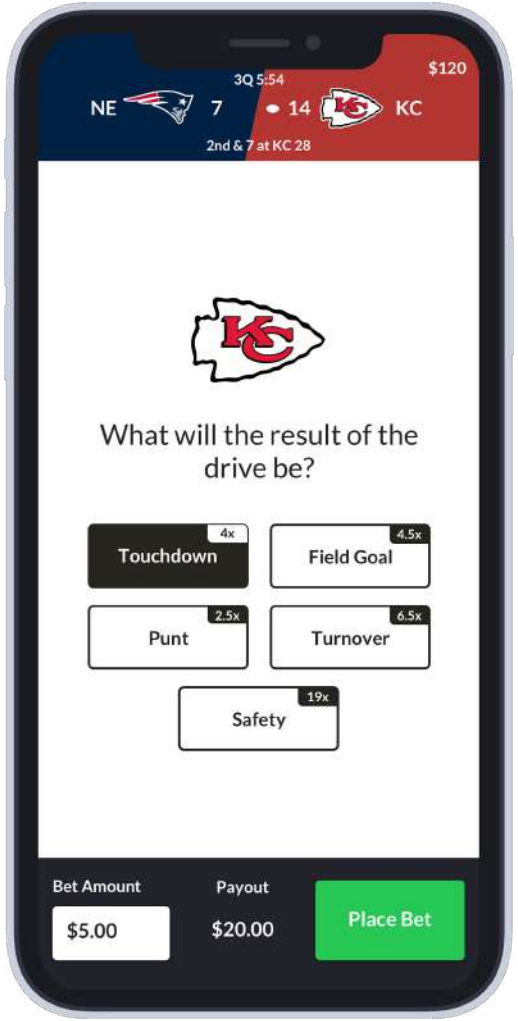
Baseball

Pitch Result

What will be the result of the next pitch?

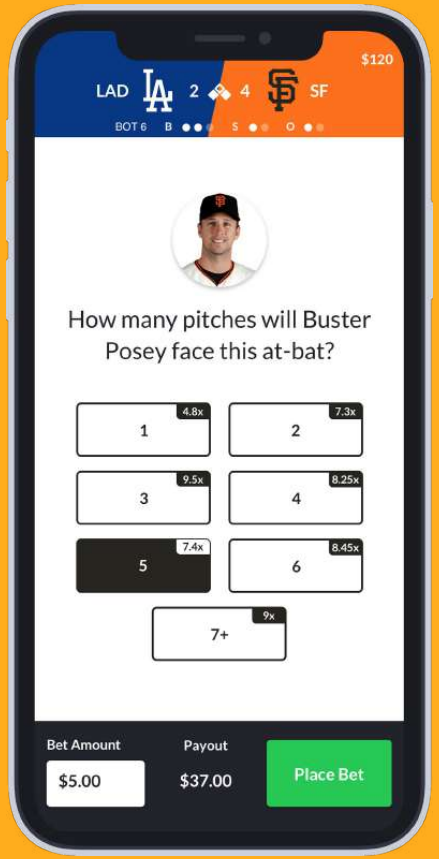
American Football
Next Drive

What will be the result of the next drive?



What does SimpleBet do?

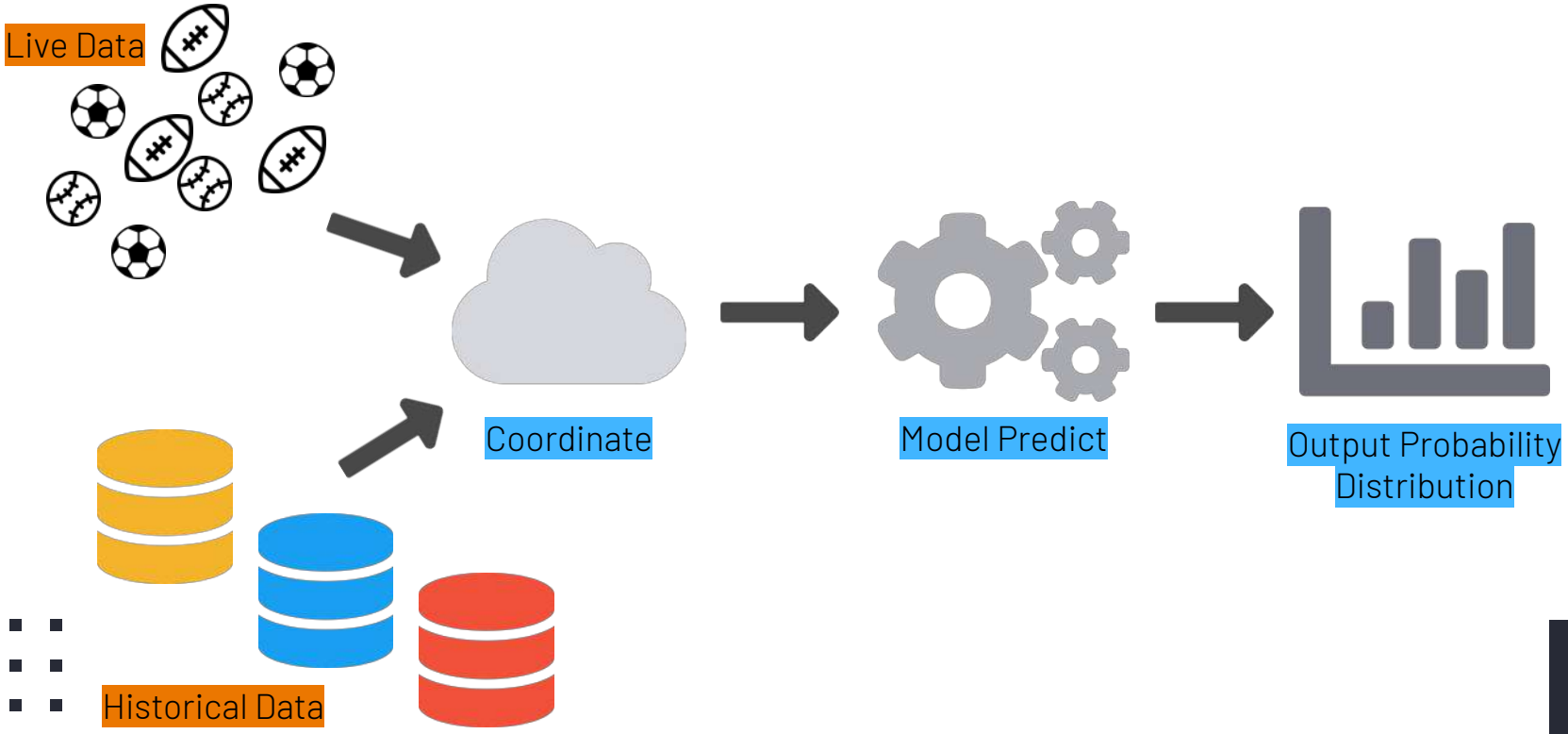
- Sell our odds as a service to enable in-play markets
- B2B customers can use our odds to offer in-play markets on their Sportsbooks



Machine Learning on the BEAM

A cautionary tale

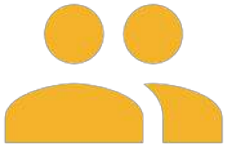
Data Flow


Live Data
Coordinate
Model Predict
Output Probability Distribution
Historical Data

In-Play Odds Feed - Engineering Problems

- Coordinating data ingestion from multiple sources
- Data issues in one match should not affect another
- Predictions must be fast enough to ensure viability of the market

Team Structure

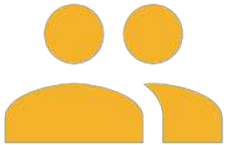


Data Science
Research

Machine
Learning
Engineering

Platform

Team Structure



Data Science Research

- Math and statistics background
- Focused on modeling and feature engineering
- Some experience in software engineering

Team Structure

Machine Learning Engineering

- Low-level systems programmers
- Minimal to no ML experience
- Excited about Rust!

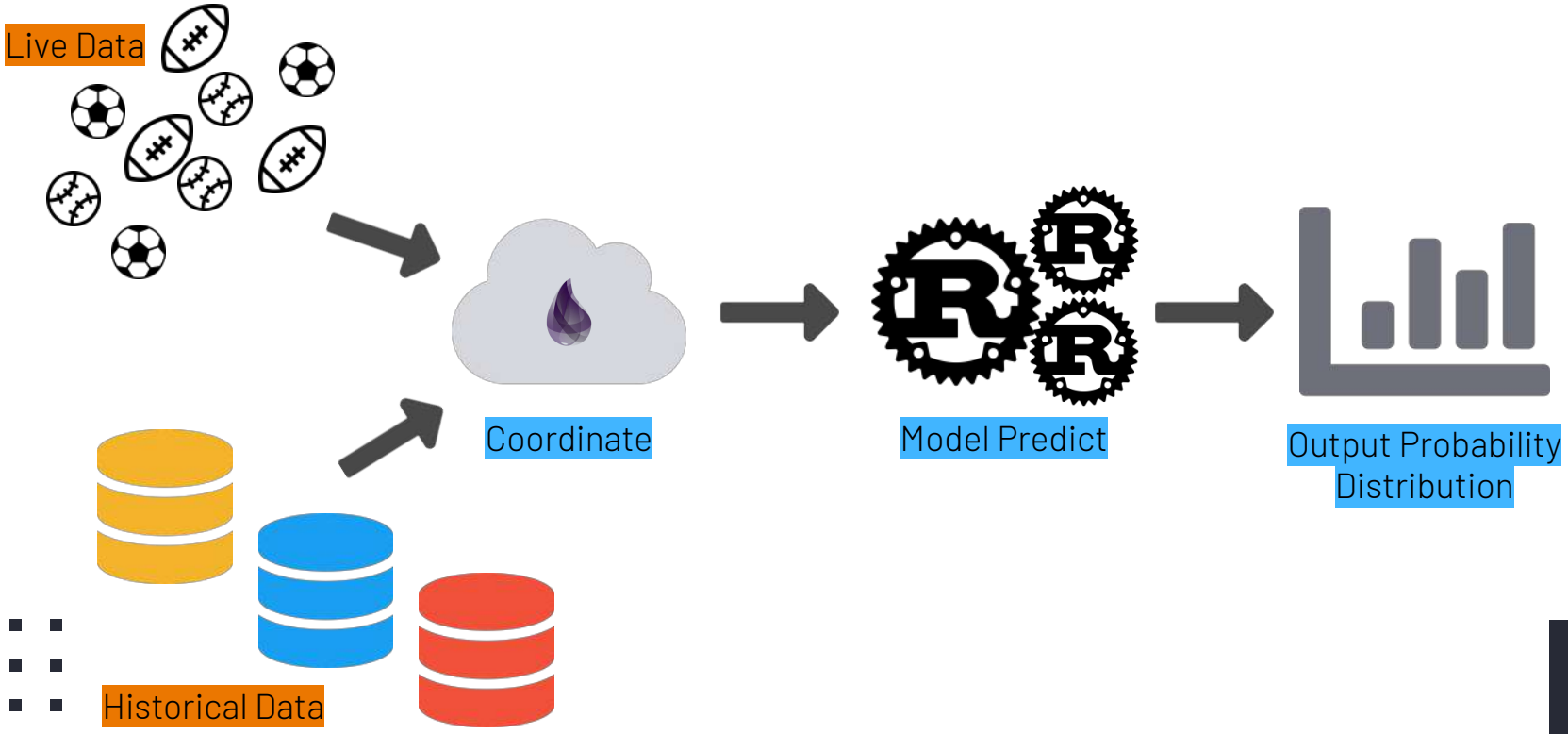
Team Structure

Platform

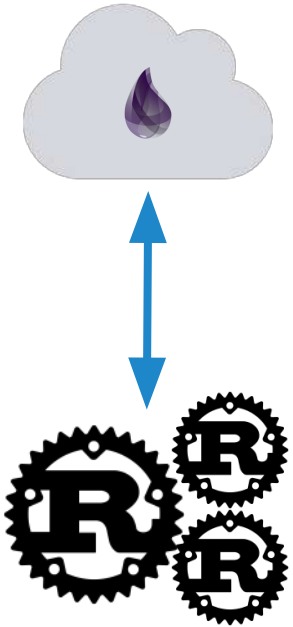
- Java, Scala, Node and Elixir backgrounds
- Experience with actor-based systems like Akka
- Responsible for coordinating the lifecycle of markets

Why use Elixir for building an odds feed?

Data Flow - NIF



Deploy Rust as a NIF



The Dream Stack

An Elixir and Rust love story

What is a NIF?

> It is a simpler and more efficient way of calling C-code than using port drivers. NIFs are most suitable for synchronous functions...that do some relatively short calculations without side effects and return the result.
>
> - Erlang documentation

Use NIFs when...

Native

- Enacl - Libsodium bindings (crypto)
- Floki - HTML5 parser using the html5ever Rust NIF from Servo
- Sass.ex - Sass compiler

Speed

- sorted_set - Sorted Set Data structure
- nifsy - Faster File System Access
- no-way-jose - JWT Signing

Using Rust to Scale Elixir for 11 Million Concurrent Users

Matt Nowack Follow
May 17 · 8 min read

Erlang

```erlang
-module(complex6).
-export([foo/1, bar/1]).
-on_load(init/0).

init() ->
    ok = erlang:load_nif("./complex6_nif", 0).

foo(_X) ->
    exit(nif_library_not_loaded).
bar(_Y) ->
    exit(nif_library_not_loaded).
```

```c
#include <erl_nif.h>

extern int foo(int x);
extern int bar(int y);

static ERL_NIF_TERM foo_nif(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
    int x, ret;
    if (!enif_get_int(env, argv[0], &x)) {
        return enif_make_badarg(env);
    }
    ret = foo(x);
    return enif_make_int(env, ret);
}

static ERL_NIF_TERM bar_nif(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
    int y, ret;
    if (!enif_get_int(env, argv[0], &y)) {
        return enif_make_badarg(env);
    }
    ret = bar(y);
    return enif_make_int(env, ret);
}

static ErlNifFunc nif_funcs[] = {
    {"foo", 1, foo_nif},
    {"bar", 1, bar_nif}
};

ERL_NIF_INIT(complex6, nif_funcs, NULL, NULL, NULL, NULL)
```

C

NIFs are compiled as .so files (Shared Objects)

```
gcc -o complex6_nif.so -fpic -shared complex.c
complex6_nif.c
```

Don’t let it crash!!!!

Rustler

Rustler Features

Safety

The code you write in a Rust NIF should never be able to crash the BEAM

Interop

Decoding and encoding Rust values into Erlang terms is as easy as a function call

Type Composition

Making a Rust struct encodable and decodable to Erlang or Elixir can be done with a single attribute

Resource Objects

Enables you to safely pass a reference to a Rust struct into Erlang code. The struct will be automatically dropped when it's no longer referenced

Let’s build a “model” in Elixir and Rust!

https://github.com/davydog187/baseball

Rustler

```
defmodule Baseball do
  @moduledoc """
  Machine Learning on the BEAM
  """
  use Rustler, otp_app: :baseball, crate: "baseball"

  def prepare(_nums), do: :erlang.nif_error(:nif_not_loaded)
  def update(_ref, _incident), do: :erlang.nif_error(:nif_not_loaded)
  def get_scores(_ref), do: :erlang.nif_error(:nif_not_loaded)
  def predict(_ref, _multiplier), do: :erlang.nif_error(:nif_not_loaded)
end
```

API - Parts

- prepare - Scores static data in the NIF for later reference
- update - Updates the state of the game e.g. home_score, away_score
- get_scores - Get the current score
- predict - Run the model!

Rustler

```
defmodule BaseballTest do
  use ExUnit.Case

  test "test" do
    nums = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]

    assert {:ok, ref} = Baseball.prepare(nums)
    assert is_reference(ref)

    assert :ok = Baseball.update(ref, :home_score)
    assert :ok = Baseball.update(ref, :away_score)
    assert :ok = Baseball.update(ref, :home_score)
    assert :ok = Baseball.update(ref, :home_score)

    assert {:ok, 3, 1} = Baseball.get_scores(ref)

    assert {:ok, 0.118} = Baseball.predict(ref, 2.0)
  end
end
```

Rustler - Function Exporting

```
pub fn on_load<'a>(env: Env, _load_info: Term<'a>) -> bool {
    rustler::resource_struct_init!(GameState, env);
    true
}

rustler::rustler_export_nifs! {
    "Elixir.Baseball",
    [
        ("prepare", 1, prepare),
        ("update", 2, update),
        ("get_scores", 1, get_scores),
        ("predict", 2, predict)
    ],
    Some(on_load)
}
```

Rustler - Initialization

```
fn prepare<'a>(env: Env<'a>, args: &[Term<'a>]) -> Result<Term<'a>, Error> {
    let data: Vec<f64> = args[0].decode()?;

    let state = GameState {
        home_score: RwLock::new(0.0),
        away_score: RwLock::new(0.0),
        model_data: data
    };

    let resource = ResourceArc::new(state);

    Ok((ok(), resource).encode(env))
}
```

Rustler - Updating and Getting State

```
fn update<'a>(env: Env<'a>, args: &[Term<'a>]) -> Result<Term<'a>, Error> {
    let state: ResourceArc<GameState> = args[0].decode()?;
    let incident: Incident = args[1].decode()?;

    state.update(incident);

    Ok((ok()).encode(env))
}

fn get_scores<'a>(env: Env<'a>, args: &[Term<'a>]) -> Result<Term<'a>, Error> {
    let state: ResourceArc<GameState> = args[0].decode()?;

    Ok((ok(), state.home_score() as i64, state.away_score() as i64).encode(env))
}
```

Rustler - Updating Game State

```
impl GameState {
    pub fn update(&self, incident: Incident) {
        use Incident::*;

        match incident {
            HomeScore => {
                let mut score = self.home_score.write().unwrap();
                *score += 1.0;
            },
            AwayScore => {
                let mut score = self.away_score.write().unwrap();
                *score += 1.0;
            }
        };
    }
}
```

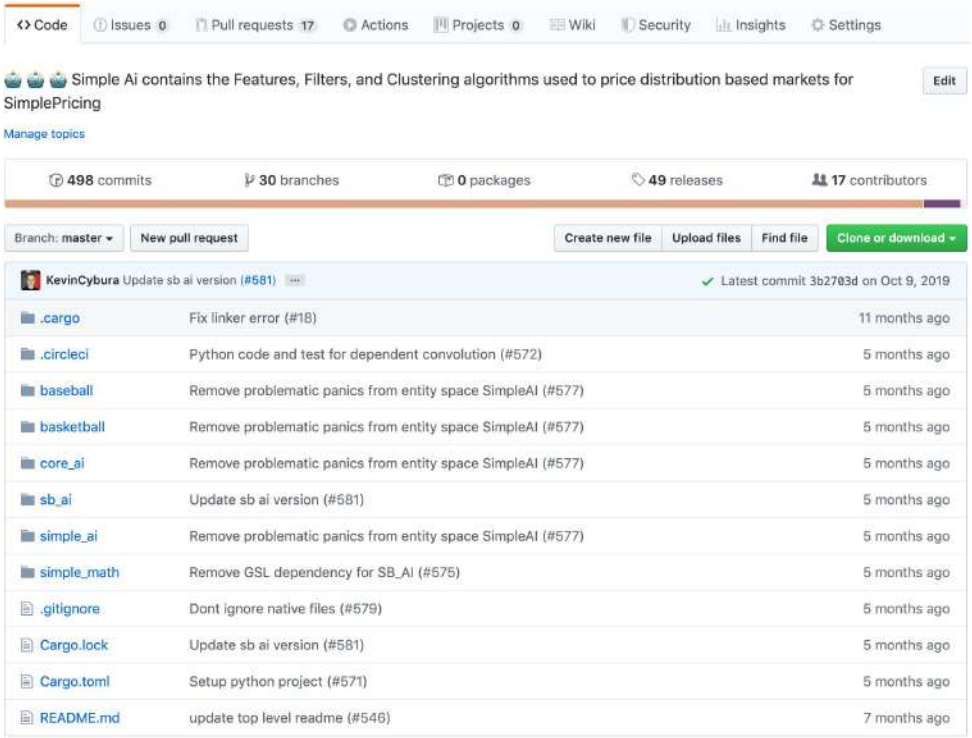
Rustler - Predict

```
fn predict<'a>(env: Env<'a>, args: &[Term<'a>]) -> Result<Term<'a>, Error> {
    let state: ResourceArc<GameState> = args[0].decode()?;
    let multiplier: f64 = args[1].decode()?;

    let sum: f64 = state.model_data.iter().sum();

    let result = (sum + state.home_score() + state.away_score()) * multiplier / 1000.0;

    Ok((ok(), result).encode(env))
}
```

Rustler

```
defmodule BaseballTest do
  use ExUnit.Case

  test "test" do
    nums = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]

    assert {:ok, ref} = Baseball.prepare(nums)
    assert is_reference(ref)

    assert :ok = Baseball.update(ref, :home_score)
    assert :ok = Baseball.update(ref, :away_score)
    assert :ok = Baseball.update(ref, :home_score)
    assert :ok = Baseball.update(ref, :home_score)

    assert {:ok, 3, 1} = Baseball.get_scores(ref)

    assert {:ok, 0.118} = Baseball.predict(ref, 2.0)
  end
end
```

SimpleAI - Rust-based Machine Learning Framework



SimpleAI Components

- core_ai - ML Framework
- simple_math - ML Algorithms and data structures
- simple_ai - NIF wrapper
- baseball - Baseball-specific model implementations for sports betting markets

Model Implementation Process

1. Model Research and feature development
2. Model is documented in Markdown in SimpleAI
3. Machine Learning Engineer implements the features into the model
4. SimpleAI is tagged for release, updated in consuming codebase

Rust NIF Approach - Challenges

- Model execution time
- Transferring data between Elixir and Rust
- Encoding and decoding external Rust structs
- Argument Error!

SimpleAI success!


ElixirTalk
Episode 153 feat. Dave Lucia- The Dream Stack with Rust & Elixir
6 months ago
Technology
0:08
1:16:06
E
T

Going too fast

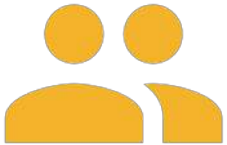
Building a ferrari when all you needed was a go-kart


DANGER
HOT PIPES

Survivorship Bias

- Just because it worked, doesn't mean it was right.
- Always challenge your assumptions

Team Structure



Data Science
Research

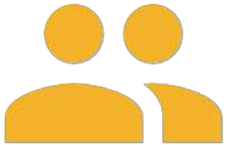
Machine
Learning
Engineering

Platform

> An organization who designs a system will produce a design whose structure is a copy of the organization's communication structure

- Conway's Law

Inverse Conway Maneuver



Data Science
Research

Machine
Learning
Engineering

Platform

Inverse Conway Maneuver

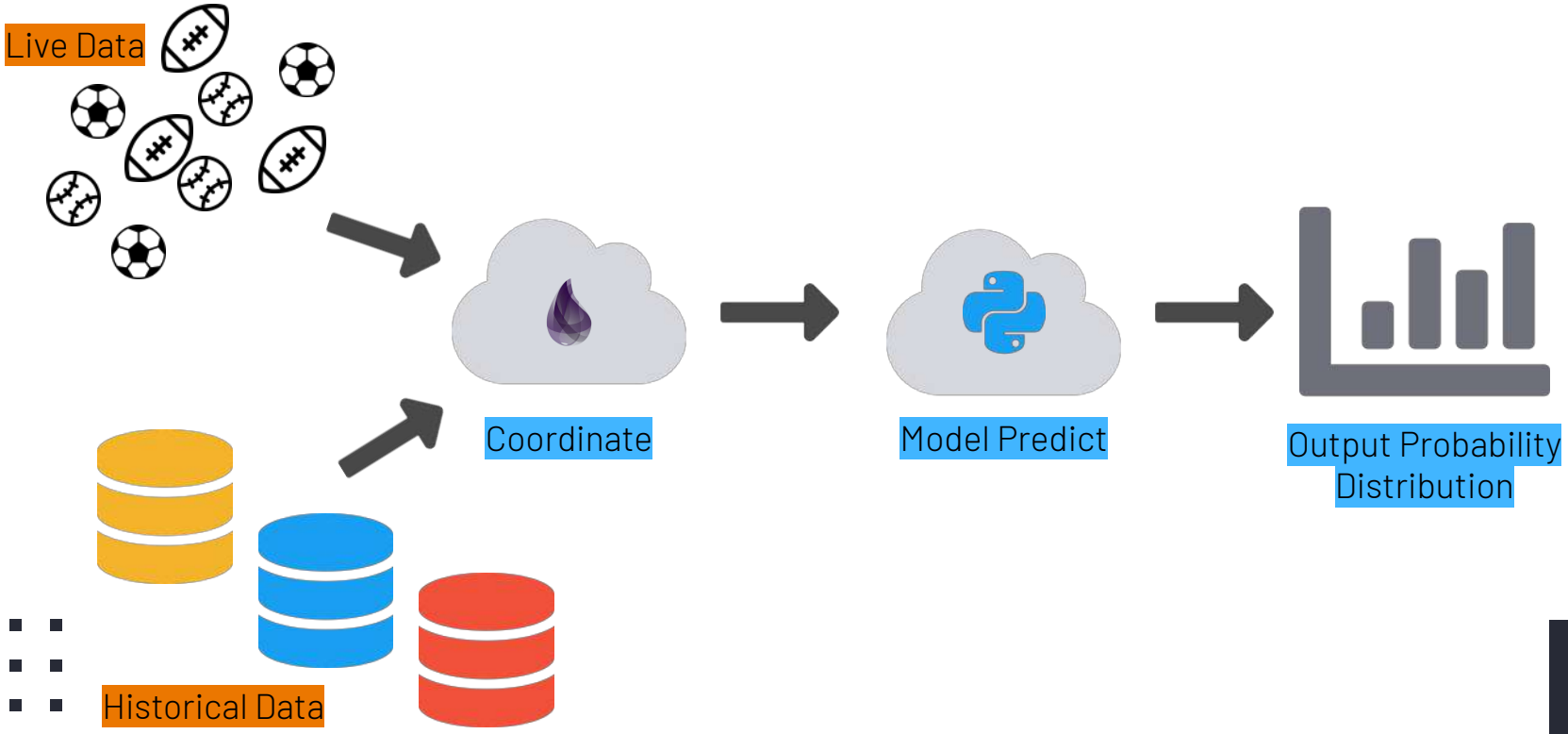
Machine
Learning
Engineering

Platform

“

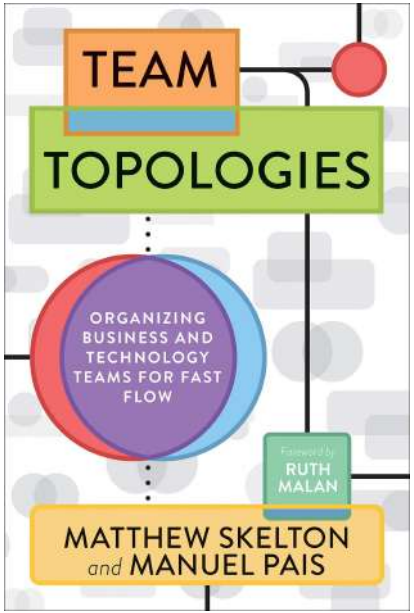
What if we were successful, despite the obstacles we created?

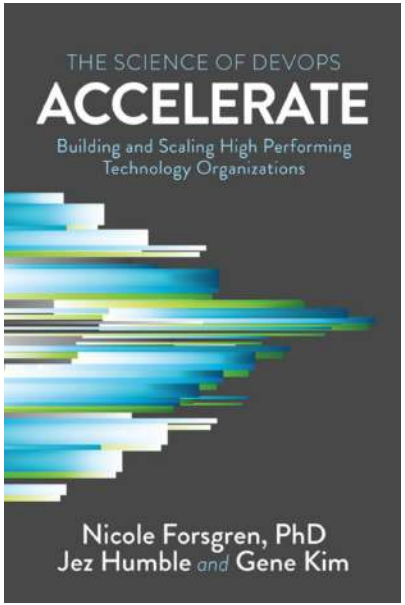
Data Flow - Service-oriented models

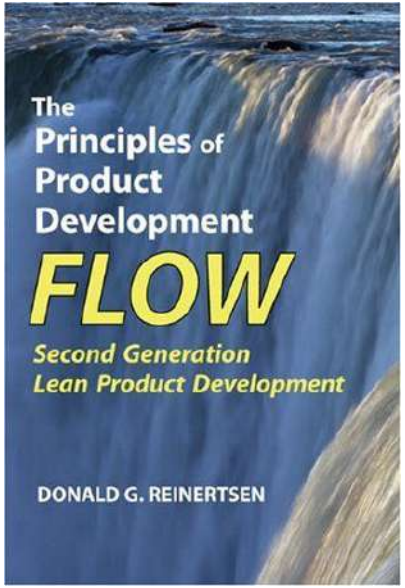


Rustling up predictive sports-betting models on the BEAM

Books


TEAM
TOPOLOGIES
ORGANIZING BUSINESS AND TECHNOLOGY TEAMS FOR FAST FLOW
Foreword by
RUTH MALAN
MATTHEW SKELTON
and MANUEL PAIS


THE SCIENCE OF DEVOPS
ACCELERATE
Building and Scaling High Performing Technology Organizations
Nicole Forsgren, PhD
Jez Humble and Gene Kim


The
Principles of
Product
Development
FLOW
Second Generation
Lean Product Development
DONALD G. REINERTSEN

Other Resources

- What happens with you hire a Data Scientist without a Data Engineer
- The rise of the term MLOps
- What is the most effective way to structure a data science team
- Thoughtworks: Inverse Conway Maneuver