ForceRuler

Overview

Bill.Anderson@salesforce.com

January 27, 2021

Agenda

1. Overview
2. Benefits
3. High-Level Design
4. Rule Anatomy
5. Example Usage
6. Demo

Questions

Overview

What is ForceRuler and Purpose

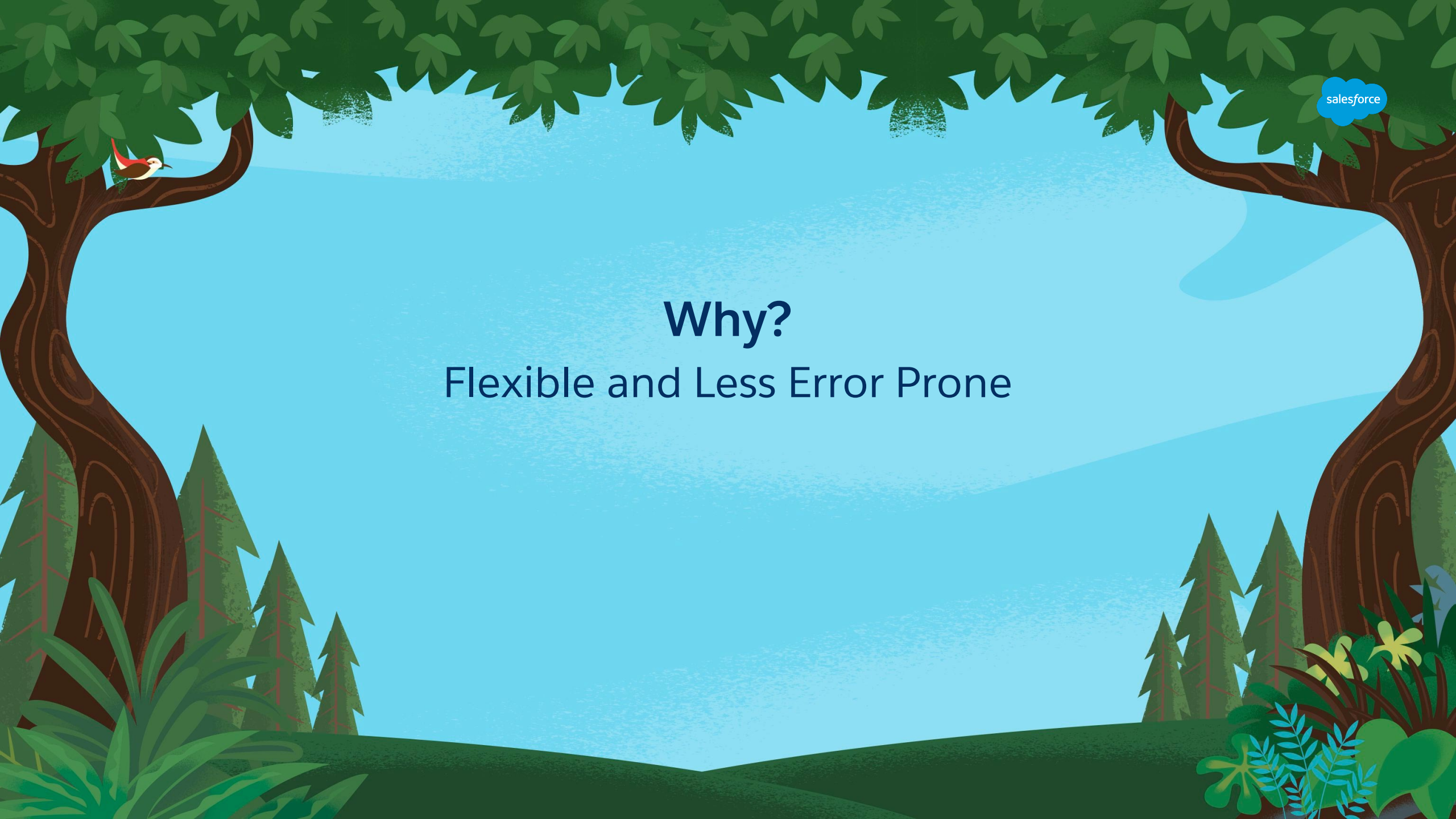
ForceRuler is an On-Platform (Apex) Simple Rules Engine modeled after Drools, where Drools keywords are **rule**, **when, then**, and **end**.

ForceRuler defines the *language* first which allows for the *dynamic* creation of rules as well as simple UI base tools for non-technical users.

ForceRuler purpose is to provide developers a simple and performant tool without mundane coding; and provides the non-technical users the ability to create rules without heavy coding skills. This helps provide small reusable content for users.

```
rule "Sample Rule Name"
  Global($lastname="Chopra", $firstname="wes", $email = "wes.chopra@gmail.com" );
  Context(sync=true,debug=true);
when
  Contact( lastname == $lastname && firstname == $firstname);
then
  Contact(email = $email);
end
```

The slides that follow cover more of the ***ForceRuler*** functionality.



Why?

Flexible and Less Error Prone

Why ForceRuler ?

Consulting at various clients it was clear to me that they needed something easier than Apex code. Many of the developers were not very good which was reflected in the number of issues, including performance. Why did I write ***ForceRuler***?

- Flow was not dynamic (i.e., create on the fly).
- Wanted something easily crafted from an editor without a UI.
- Allows Apex, but controlled and measurable.
- Provide a performant (safe) alternative to Apex.
- Build in functionality to avoid common mistakes (***pro-active*** instead of ***re-active***)
- Had written a Parser before (in C# and C++) for clients

Benefits

Rethinking the “Code” Approach

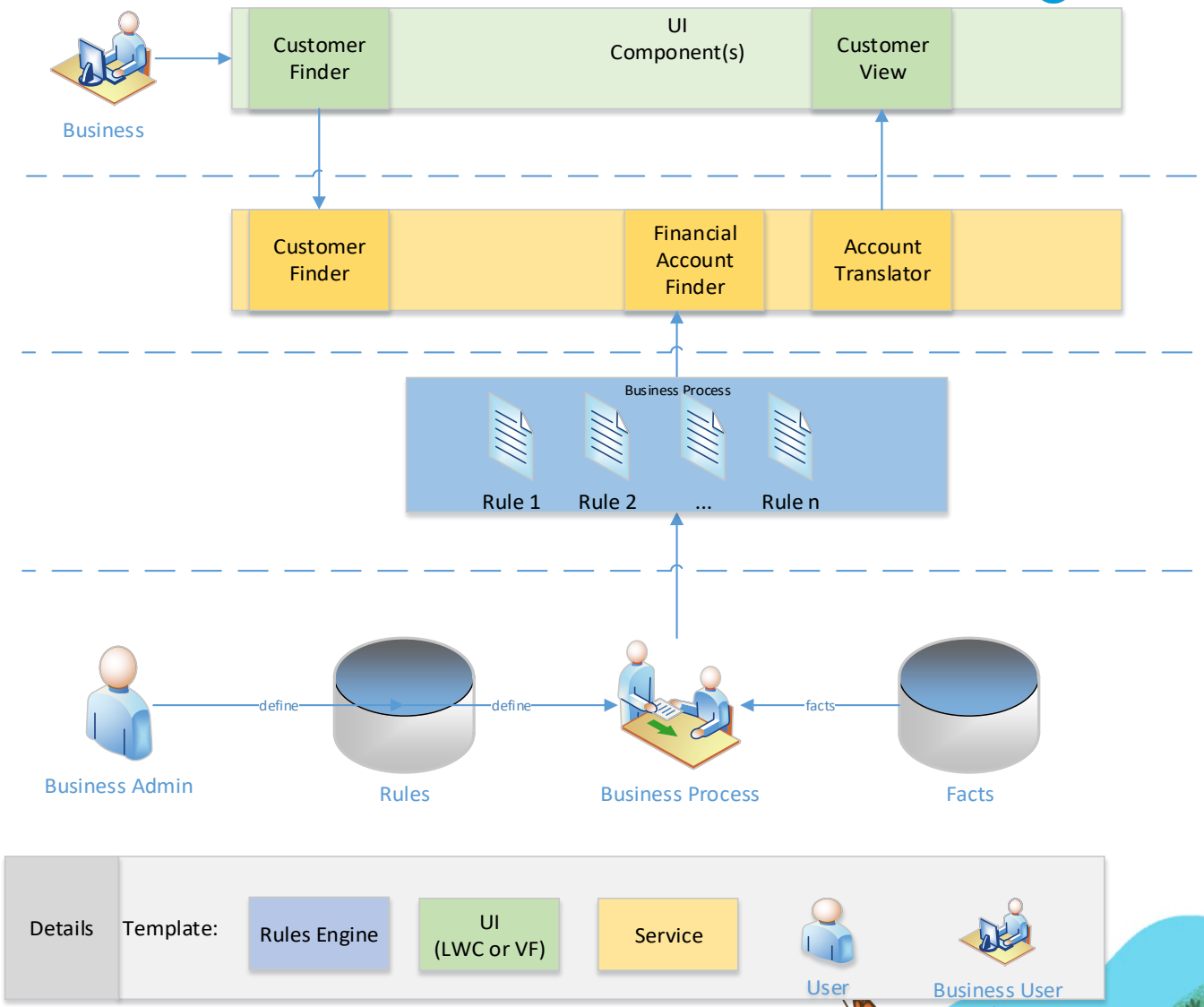
Benefits

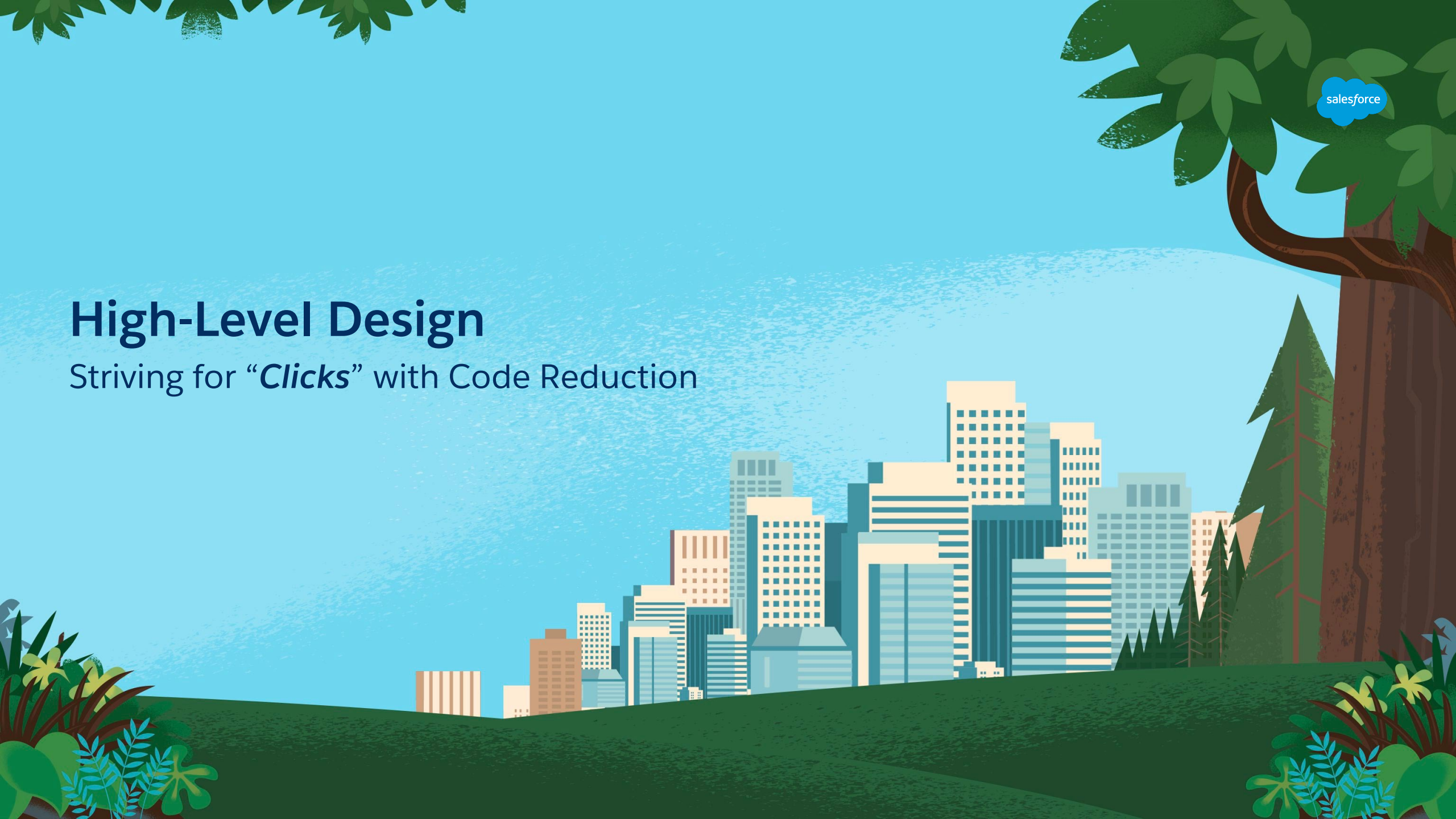
- ✓ Rules provide a reusable component
- ✓ Rules are dynamic and easy to change *(Apex Language Utility)*
- ✓ Rules can run synchronously or asynchronously
- ✓ Rules are compiled and cached (i.e., Performant, avg ~200-400ms, but can run <20ms)
- ✓ Rules work in Triggers, Apex Code, CDC & Platform Events
- ✓ Rules ***Then*** functionality can be augmented
- ✓ Rules ***When*** functionality can be used just to filter (sub-set of Facts)
- ✓ Rules can be changed by Developer Or Business Analyst via UI
- ✓ Rules reduce the APEX footprint (lightweight, terse functionality)
- ✓ Rules can use aliased methods to avoid revision (i.e., ***FR.save***())
- ✓ Rules Callable(s) can be augmented.
- ✓ Rules do not require Unit Tests *(except, Callable Apex method(s) you may write)*

Rules to Accelerate the Business

ForceRuler can take either Business or Technical Rules and accelerate the Business time to market. Rules can be *chained* together and augmented as the Business Process Changes.

In addition, rules could be created dynamically, whereby, Business Rules could create **new** Business Rules at runtime.





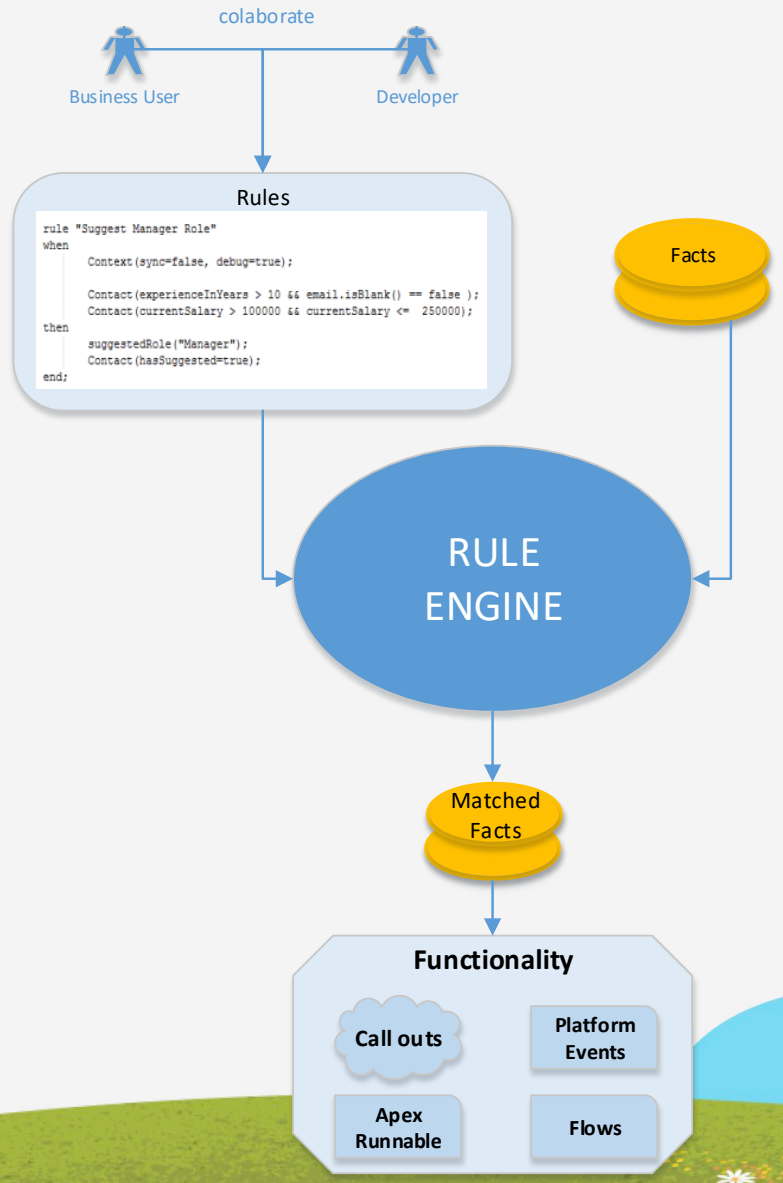
High-Level Design

Striving for "*Clicks*" with Code Reduction

High-Level Flow

ForceRuler uses Rules to evaluate and process Facts (i.e., SObjects). Facts that match the ***WHEN*** criteria are pass along to the Process Section (***THEN***). The Process Section supports (sync and async):

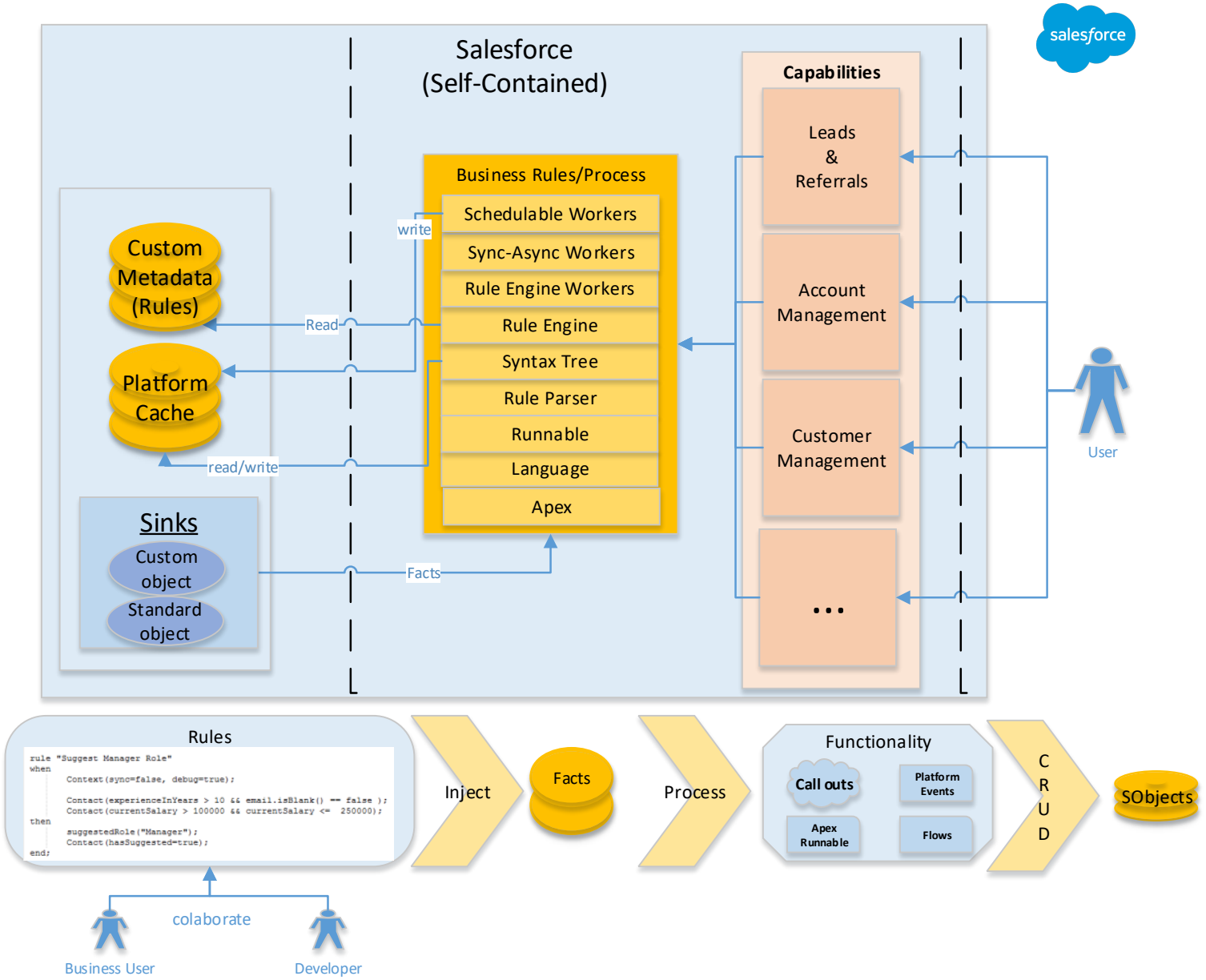
- Callouts
- Platform Events / CDC
- Flows
- Apex



High-level Architecture

ForceRuler contains various elements to allow extension (via abstraction). Some of the benefits include, but not limited to:

- Caching
- Syntax/Parse Tree
- Parser
- Runnable Interface (***Then*** section)
- Language Utility
- Sync/Async Workers
- Schedulable (runtime cache)
- Aliasing (Custom Metadata)
- Runs completely On-Platform; however, was designed to utilized *Salesforce Functions* (future).



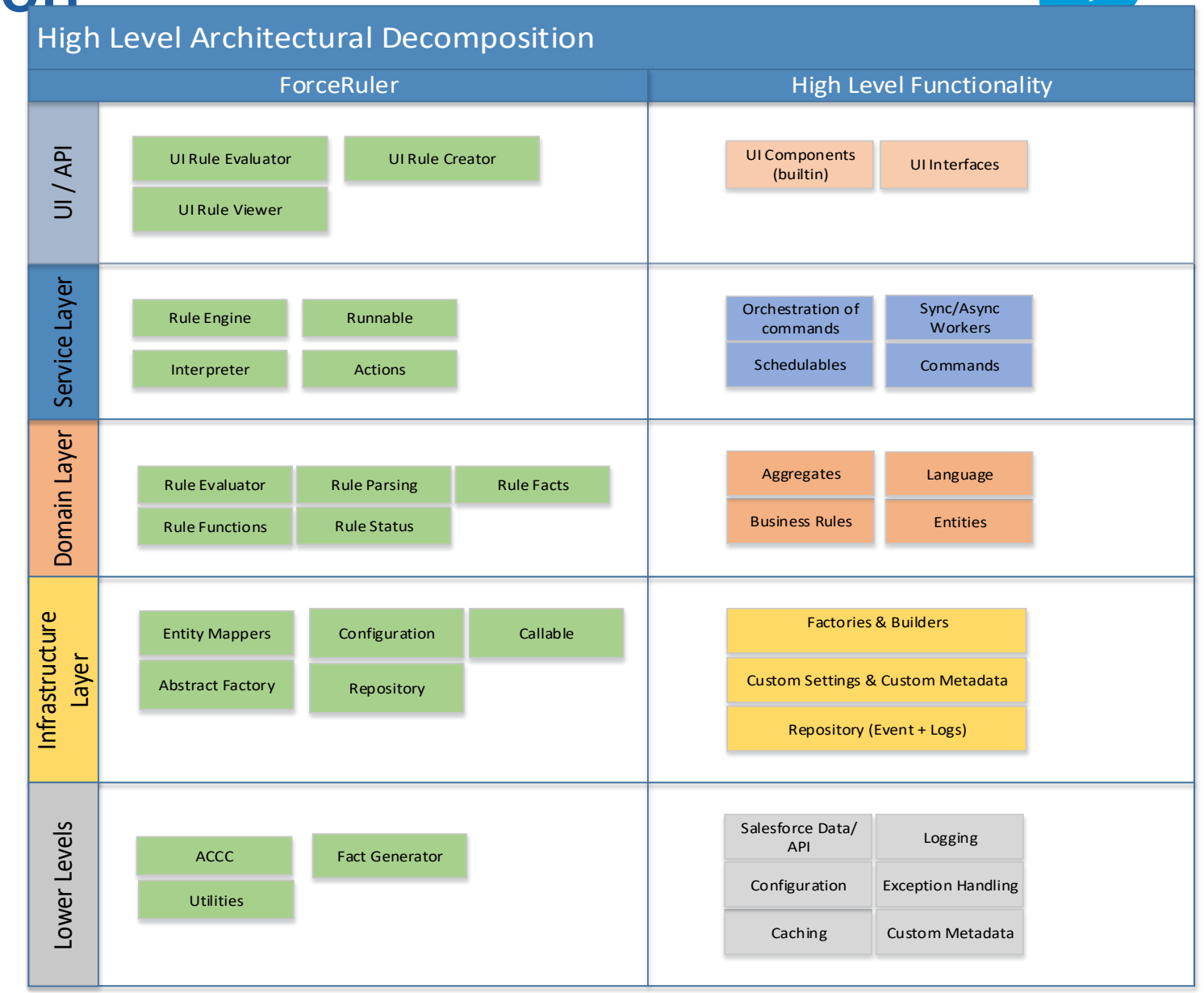
High-Level Decomposition

ForceRuler is decomposed into layers for extension and management of concerns.

This is still a work in progress. While there are various abstractions / interfaces injected via Factories and Builders there is still refactoring that will take place.

The goal was to allow extensions, not modifications (**OCP**). Some aspects have a Point-Of-View (POV) but allows the POV to be extended.

***Note: With the advent of Salesforce Functions, parsing may be offload**



ForceRuler Anatomy

Rule Basics

ForceRuler Anatomy

Basics

ForceRuler follows a simple Drools-like format,

Rule "<RULE-NAME>"

Global($<characters>+ <assignment> <text> | <numeric> | <function>);

Context(sync=<Bool>, debug=<Bool>, performance=<bool>);

When
<DOMAIN>(<FIELDS> [operator] <VALUE> [|| &&] ...);
:
Then
<DOMAIN>(<FIELDS> = <VALUE> , ...);
:
<FUNCTION>(<parameters>);
:
End

```
rule "Sample Rule Name"
  Global($lastname="Chopra",  $email = "wes.chopra@gmail.com" );
  Context(sync=true,debug=true);
when
  Contact( lastname == $lastname );
then
  Contact(email = $email);
  fr.save("sample rule name");
end
```

TEMPLATE:

<DOMAIN>

- SObject Name (Generally)
  - **'Account', 'Contact', 'Custom__c'**

<FUNCTION>

- Can be an aliased function **(i.e. fr.run())**
- Or any class that inherits from **sfr_Runnable**

Example Rule Usage

Apex Code and UI

Example

Used in a Trigger (just for presentation; not recommended this way)

```
/** Note     : This is Just a Sample! We would normally use a Trigger Framework to call into!**/
trigger Contacts on Contact (before insert, before update) {
   // our contact rule name
   final string RULE_NAME = util_Constants.CONTACT_RULE_NAME;

   if (Trigger.isInsert) {
      if (Trigger.isBefore) {
         eval_status status = (new sf_RuleTriggerRunner(RULE_NAME,Trigger.New)).run();
         Application.tracer('+++++ [Bef Ins] FACTS     :' + status.evaluatedSuccessfulFacts);
         Application.tracer('+++++ [Bef Ins] PROC FACTS:' + status.processedFacts);
         Application.tracer('+++++ [Bef Ins] SUCCESS   :' + status.success());
      } // end of before update
   } else if ( Trigger.isUpdate ) {
      if (Trigger.isBefore) {
         // Process before update
         eval_status status = (new sf_RuleTriggerRunner(RULE_NAME,Trigger.New)).run();
         Application.tracer('+++++ [Bef Upd] FACTS     :' + status.evaluatedSuccessfulFacts);
         Application.tracer('+++++ [Bef Upd] PROC FACTS:' + status.processedFacts);
         Application.tracer('+++++ [Bef Upd] SUCCESS   :' + status.success());
      }
   } // end of update
}// end of trigger
```

Example

Callable

```
rule “Sample Rule Using  Callables”
  Global( $name=Userinfo.getname(), $cout= Callable.test() );
  Context(sync=true,debug=false);
when
  Contact( email==Userinfo.getuseremail() && $cout.contains("test") );
then
  Contact( firstname = $name );
end
```

The above …

- uses two callouts, ***UserInfo*** and ***Callable,*** these two callouts are alias and can use any (properly inherited) Apex class to handle various functions (ICallable).
- The ***WHEN*** section, at runtime, makes a callout to get the user’s email and takes the output from test and runs contains.
- Will set the Contact *firstname* to the contents found in *$name*.

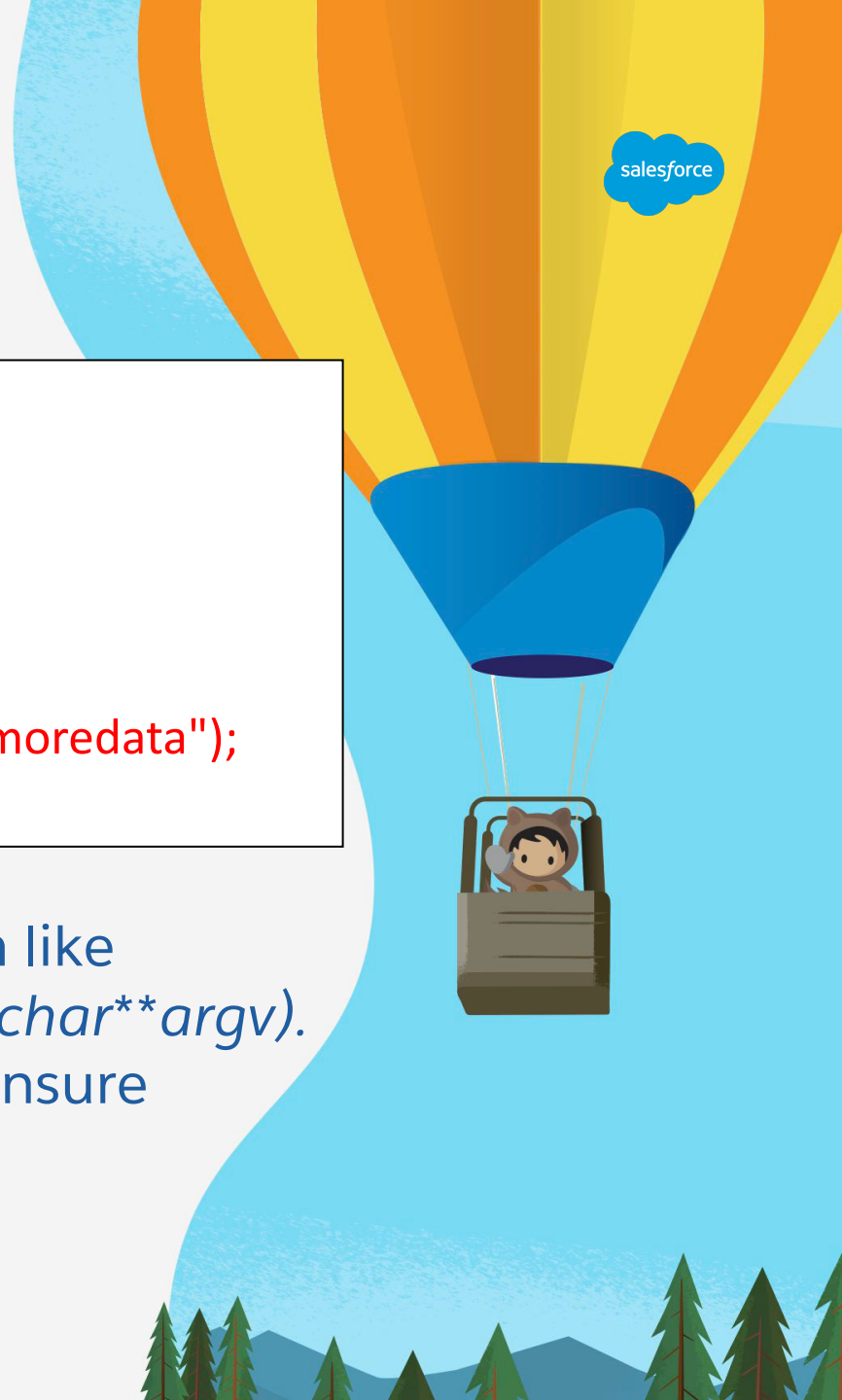
- ***PS: Context is optional, only added for edification.***

Example

Account Rule calling a Flow

```
rule "Account Check Name is Testing"
        Context(sync=true, debug=false);
when
        Account(name==“sample account flow"  );
then
        flow.run("","THE_FLOW_API_Name", "arg1:somedata", "arg2:moredata");
end
```

Parameters can be passed into any aliased function, much like command line arguments passed into C++ *main(int argc, char**argv).* The above uses the demarcation (*arg1, arg2, … argN*) to ensure expected order



Example

Changed Data Capture (CDC) or Platform Event

```
trigger AccountChangeEventTrigger on AccountChangeEvent (after insert) {
    // Normally, one pulls the Rule from the Sink
    String PE_Rule ='rule "AccountChangeEvent Rule"  ' +
                    '     Context(sync=true, debug=true); ' +
                    ' when ' +
                    '      AccountChangeEvent("update" == changetype &&  ' +
                    '      true == changedfields.contains("industry") &&  industry!="banking" ); ' +
                    ' then  ' +
                    '      fr.out(); ' +
                    ' end';
    // this processes the CDC rule ... fr.out() will write output out as well!
    eval_status status = (new sfr_RuleTriggerRunner(PE_Rule,Trigger.New,false)).run();
    Application.tracer('+++++ [CDC] FACTS COUNT  :' + status.evaluatedSuccessfulFacts);
    Application.tracer('+++++ [CDC] PROC FACTS   :' + status.processedFacts);
    Application.tracer('+++++ [CDC] SUCCESS      :' + status.success());
    Application.tracer('+++++ [CDC] FACTS        :' + status.theFacts);
    Application.tracer('+++++ [CDC] STATUS       :' + status);
}
```

CDC events require a specialization. In the above sample, the domain is *Change Data Capture Event*. This allows the parser to treat the components within the parenthesis as CDC Events. The other three special values are:

- **changetype** – indicates if this is a **CREATE**, **UPDATE**, **DELETE**, **UNDELETE**, **GAP_CREATE**, etc.
- **objectName** – indicates the type of event (*AccountChangeEvent*)
- **changedfields.contains** – indicates whether a field (i.e. *Industry*) has changed.

Special Functions

ForceRuler aliases methods to allow behavior variations [THEN Section]

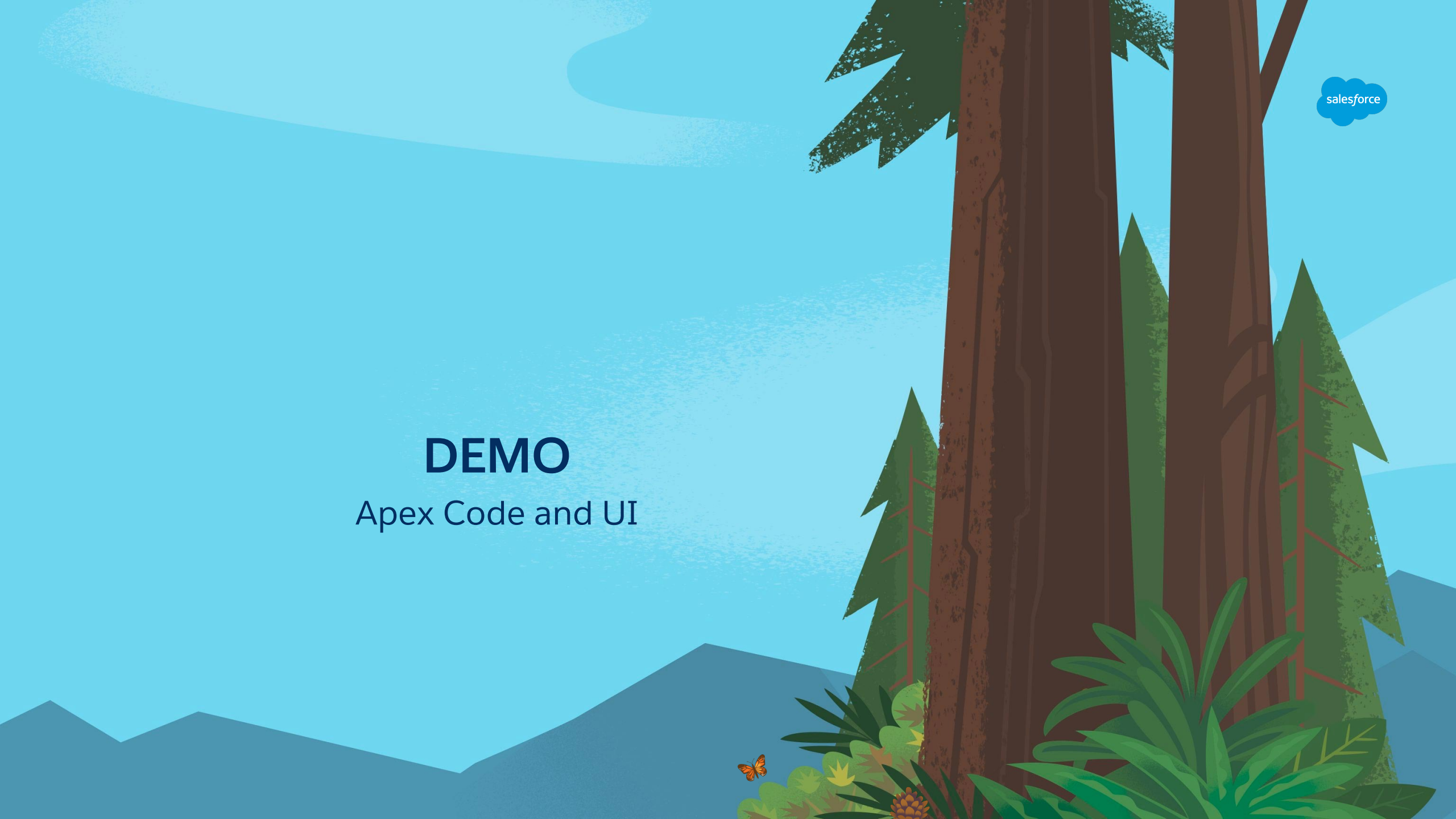
Alias	Class Name (Built-in Functions)	Description / Comment
fr.save()	*sfr_RunnableSave*	Apex Class which supports the *sf_RERunnable* Interface. Will save the data to Salesforce
fr.update()	*sfr_RunnableSave*	Apex Class which supports the *sf_ RERunnable* Interface. Will update the data to Salesforce
flow.run()	*sfr_RunnableFlowCaller*	Apex Class which supports the *sf_ RERunnable* Interface. Will invoke a specific flow, passing in user-defined arguments and Facts
fr.out()	*sfr_RunnableOut*	Apex Class which supports the *sf_RERunnable* Interface. Will output ONLY the data (Facts, Parameters, etc.)
UserInfo	*Util_UserInfo*	Apex class used to get ***UserInfo***
Callable	*Util_Callable*	Apex class which supports ICallable

ForceRuler allows functionality to be alias to Apex classes. This allows the rule function to remain the same, yet the underlying implementation can be changed internally. There are three built-in functions, but one has the ability (via custom metadata) to change the underlying class called. As well as augment.

Function or class names are **NOT** case-sensitive.

DEMO

Apex Code and UI



Summary

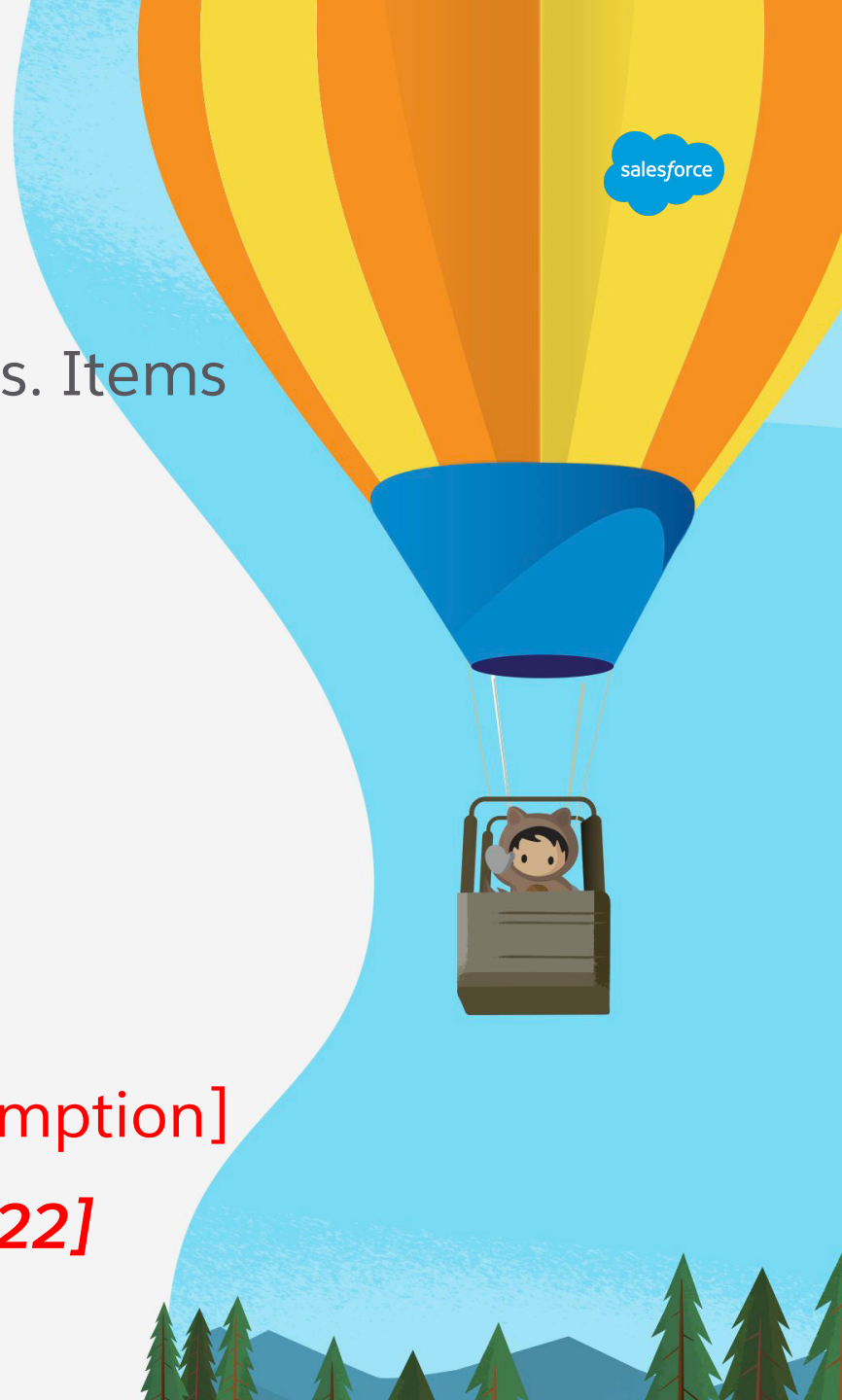
Summary

Overview

ForceRuler performs well but is still a work in progress. Items still on the table (no specific order):

- Use *Salesforce Functions*
- Improve the User Interface
- Import Functionality
- Improve Debugging
- Refactor parts (SOC)
- Create Managed (Locked) Package [for user consumption]

[Items in red represent targeted items for 2022]




QUESTIONS
?


Thank
You