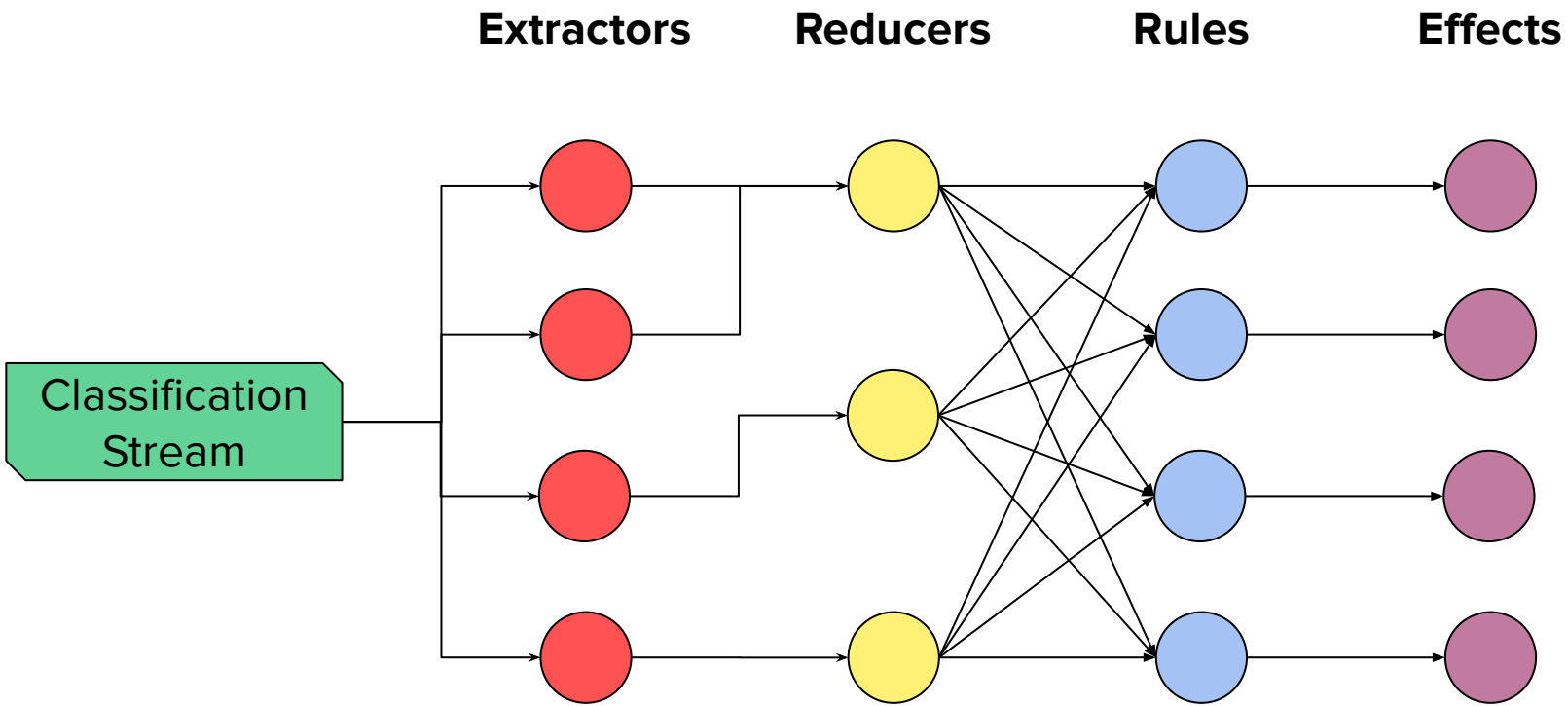
Introducing Caesar

An Advanced Subject Retirement Engine

The Basic Idea



Extractors

- Raw *Zooniverse* classifications contain **a lot** of data and most of those data are **not scientifically interesting**.
- Caesar's **built-in extractors** are designed to extract just the information that you're interested in.
- Some are designed for specific tasks like **question** tasks, **drawing** tasks or **survey** tasks.
- The *Pluck Field* extractor can be used to **select a specific field** from the raw classification data.
- The *Blank* extractor can be used to **detect null responses** to tasks.
- Finally, you can use the *External* extractor to **define your own functionality**.
- When you create a Caesar extractor you must assign it a **unique name**.

Reducers

- Reducers are designed perform **aggregation tasks**.
- When you create a Caesar reducer you must assign it a **unique name**.
- They can consider the **outputs from *specific* extractors** and some ***specific* metadata** associated with a subject.
- They can aggregate extracts associated with a **particular subject *or*** with a **particular volunteer**.
- Caesar's **built in reducers** compute **simple statistics** like the majority consensus for a particular subject, or the number of classifications it has received.
- You can extend Caesar's built-in functionality using **external reducers**, or the **SQS Reducer** (see machine learning tutorial).

Rules

- Rules are **logical and mathematical operations** that are applied in order to **make decisions** about specific **subjects** or **volunteers**.
- Rules consider the **outputs from reducers** and ***any*** **metadata** associated with a subject.
- Rules are defined using a **special syntax** (see later slides).
- It is possible to specify that Caesar evaluates rules in a **specific order**.
- In that case, Caesar will stop evaluating rules for a particular subject as soon as one of them evaluates to ***true***. So **some later rules may not be considered**.
- When rules evaluate to ***true*** they trigger **effects.**

Effects

- When Caesar's rules evaluate to **true**, they trigger **one or more** effects.
- Like rules, effects can apply to **particular subjects** or **volunteers**.
- For **subjects** possible effects are:
  - **Retire the subject**, specifying a **retirement reason**.
  - **Add** the subject to a ***Collection***.
  - **Add** the subject to another **subject set**.
  - **Send** the subject to an **external URL**.
- For **volunteers** one type of effect is **currently in development**. It will allow volunteers to be ***promoted*** to a specific workflow.
- The intended use of this effect is promoting skilled volunteers to more difficult workflows and tasks.

Example

- Here's an example that follows on from our **project building tutorial**.
- We don't want to waste our volunteers' time classifying junk lightcurves, so we'll set up a rule that will **immediately retire any subject that receives three "Junk" classifications**.
- To do this, we'll use a ***QuestionExtractor*** to isolate the answer to the first task and then a ***StatsReducer*** to count the number of votes for "Junk".
- Then we'll set up a rule that evaluates to True if the *StatsReducer* counts 3 or more "Junk" votes.
- Finally we'll associate a **retire subject effect** with our rule.
- This should **save lots of volunteer clicks**!

Caesar’s Rule Syntax

- Caesar rules are defined using a **special syntax**.
- At first glance they can look daunting, but **complex rules** are actually composed of **several simple conditional units** that all have the **same basic structure**.

```
[ “operation”, operand1, operand2, … , operandN ]
```

- **Complex rules** are possible because any of the **operands may themselves be conditional units**. Conditions can be **nested** inside each other.
- The operands can ***also*** refer to the **results of reducers**, **subject metadata**, or they may represent **constant numerical values**.

Caesar's Rule Syntax - Operations

- The result of a **conditional unit** is a Boolean value - *True* or *False*.
- Accordingly, the operations that Caesar conditions support all perform **logical reductions and comparisons** between the operands.
- The supported operations that operate on **numerical values** are:
  - `"lt"` - ***Less than*** - returns **True** if `operand1 < operand2 < … < operandN` and **False** otherwise.
  - `"gt"` - ***Greater than*** - returns **True** if `operand1 > operand2 > … > operandN` and **False** otherwise.
  - `"lte"` - ***Less than or equal*** - returns **True** if `operand1 <= operand2 <= … <= operandN` and **False** otherwise.
  - `"gte"` - ***Greater than or equal*** - returns **True** if `operand1 >= operand2 >= … >= operandN` and **False** otherwise.

Caesar's Rule Syntax - Operations

- "eq" - ***Less than*** - returns **True** if `operand1 == operand2 == … == operandN` and **False** otherwise.

- The supported operations that operate on **boolean values** are:
  - "and" - ***Greater than*** - returns **True** if `operand1 and operand2 and … and operandN` and **False** otherwise.
  - "or" - ***Less than or equal*** - returns **True** if `operand1 or operand2 or … or operandN` and **False** otherwise.
- There are **two other important components** in Caesar rules.
  - **Constants**, specified as `[ "const", value ]`, where *value* is a number e.g. 3. Constants are used as operands for numerical operations.
  - **Lookups**, specified as `[ "lookup", reference, default ]`, where *reference* may refer to results of reducers or subject metadata, and *default* is a value that should be used if the specified reference does not exist.

Caesar's Rule Syntax - Reducer Result Lookup

- To use the **result of a particular reducer** as the operand of a condition, use a lookup component like this:

  `[ "lookup", reducer_name.reducer_attribute, default ]`

- *`reducer_name`* refers to the **unique name** you assigned to the reducer when you created it.
- *`reducer_attribute`* refers to a particular element of the reducer's output.
- For example, the **Consensus reducer** returns data with three attributes: `most_likely`, `num_votes` and `agreement`. To select the **`num_votes`** attribute (with a default of zero) for a reducer ***named*** **`cons`** you would use

  `[ "lookup", cons.num_votes, 0 ]`

Caesar's Rule Syntax - Subject Metadata Lookup

- To use the **value of a subject metadatum** as the operand of a condition, use a lookup component like this:

  `[ "lookup", subject.metadatum_name, default ]`

- **`subject`** is literally the word "subject".
- *`metadatum_name`* is the name you gave to your subject metadata when you uploaded it.
- For example, if you wanted to Caesar to ignore a subset of your subjects you could define a hidden metadatum called `#ignoreme`. You could then look up the value of `#ignoreme` like this

  `[ "lookup", subject.#ignoreme, False ]`

The Caesar User Interface

- The Zooniverse team have developed a **web user interface** for Caesar
- Using the web UI, you can set up **extractors**, **reducers**, **rules** and **effects** for your workflows.
- To **access** this user interface, visit

https://caesar.zooniverse.org

- You can **log in** using your normal **Zooniverse username and password**.

Extending Caesar

- Sometimes Caesar's built-in functionality doesn't do exactly what you want.
- If you find that to be the case and you don't mind a bit of coding you have some options.
  - For simple operations you could consider adding a new reducer or extractor to the `aggregation_for_caesar` GitHub repository.
  - `aggregation_for_caesar` is also the name of a Zooniverse-run application for custom extractors and reducers. If you specify a URL to that repository when you set up an ***external*** reducer or extractor, then your code will be executed by and **you won't need to provide any compute resources**.
  - With more effort, you can define external extractors and reducers that can perform **arbitrarily complex** aggregation and reduction logic **asynchronously using compute resources you manage**, and then pass the results back to Caesar.