How To Write Complex Data Pipeline In Ruby

2016.12.02

RubyConf Taiwan 2016

Kazuyuki Honda

HELLO!

I am Kazuyuki Honda

DevOps Engineer at Quipper

You can find me at:
https://github.com/hakobera on Github
@hakobera on Twitter

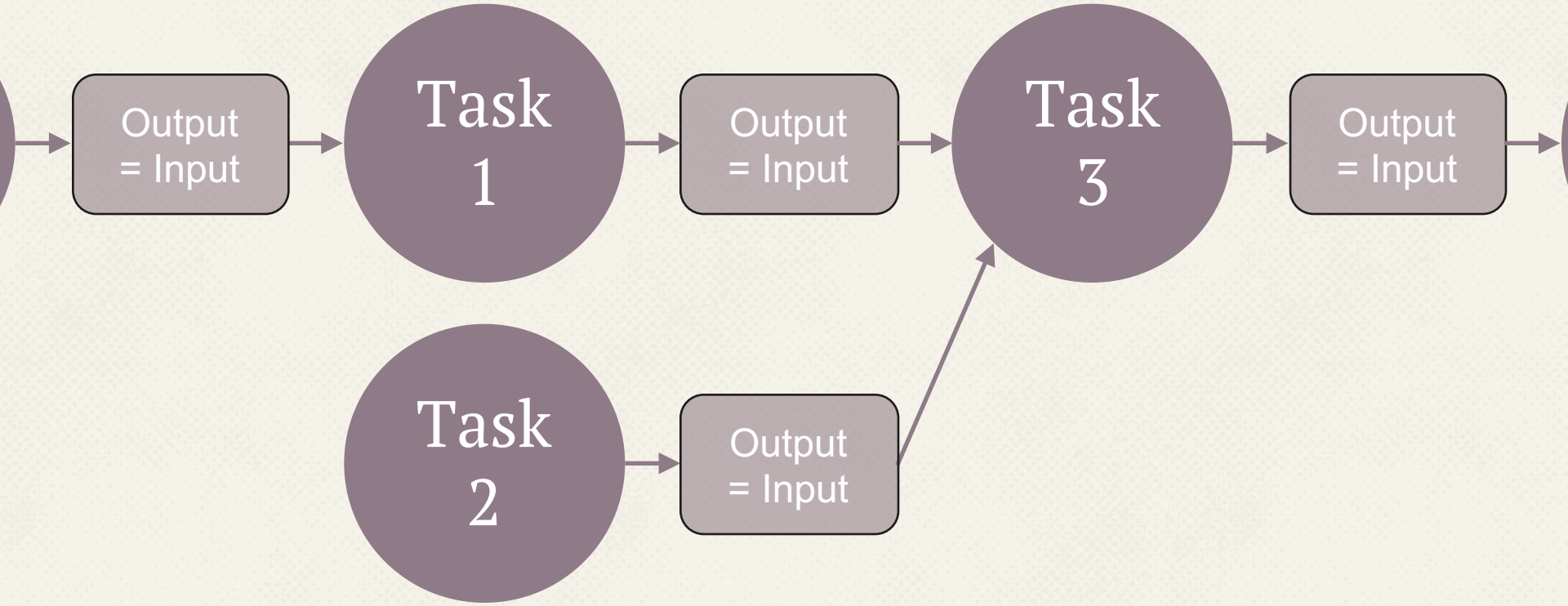
Quipper

- EdTech Company
  - Our service support both teachers and students
- Services are launched in 6 countries
  - Indonesia, Philippines, Mexico and Japan
  - Trials starts in 2 other countries

What is Data Pipeline?

Let’s dive into data and pipes

What is Data Pipeline?

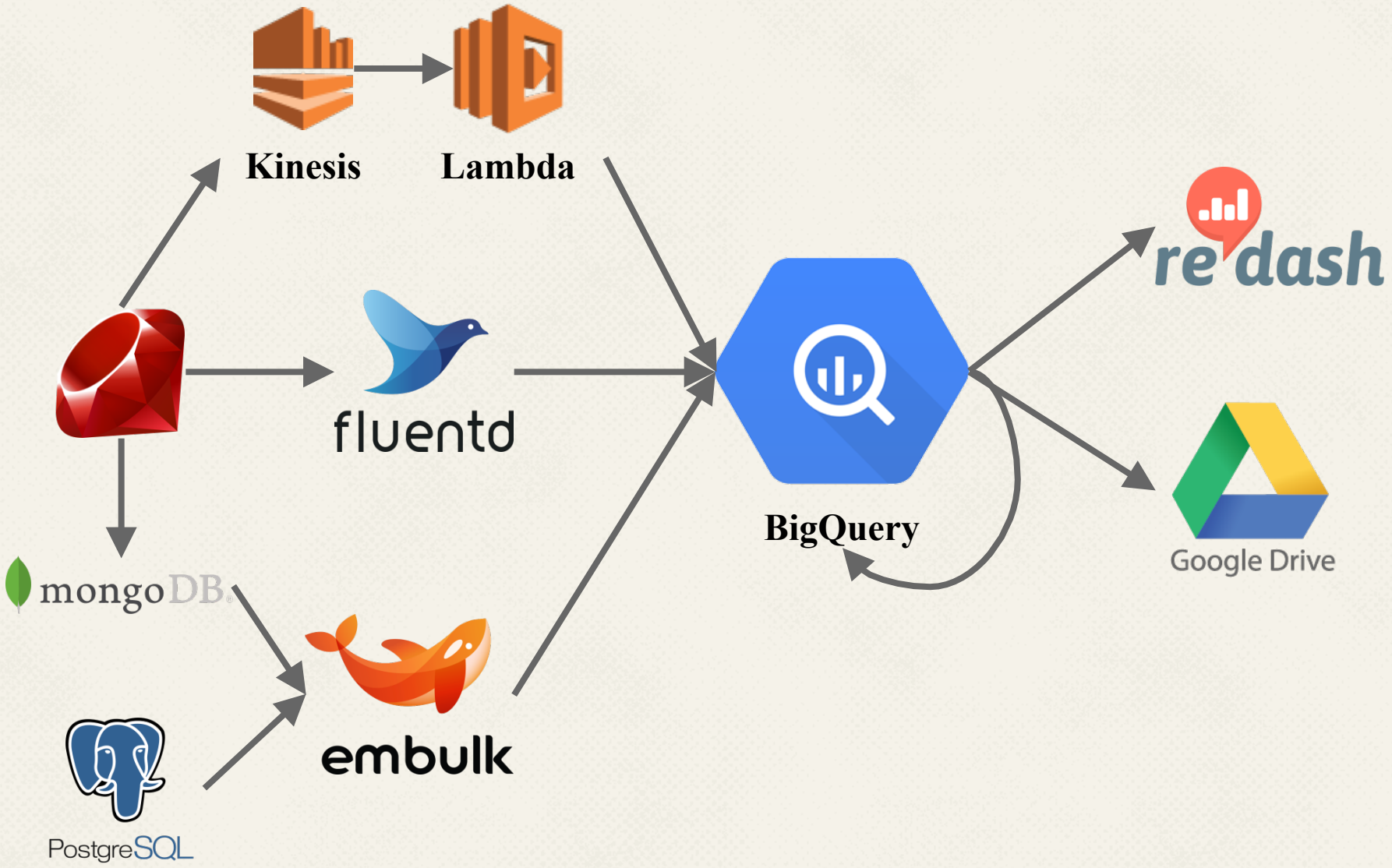


- Data Pipeline is sequence of data and task
- Represented as DAG (Directed Acyclic Graph)
- Task can read data which generated by previous tasks

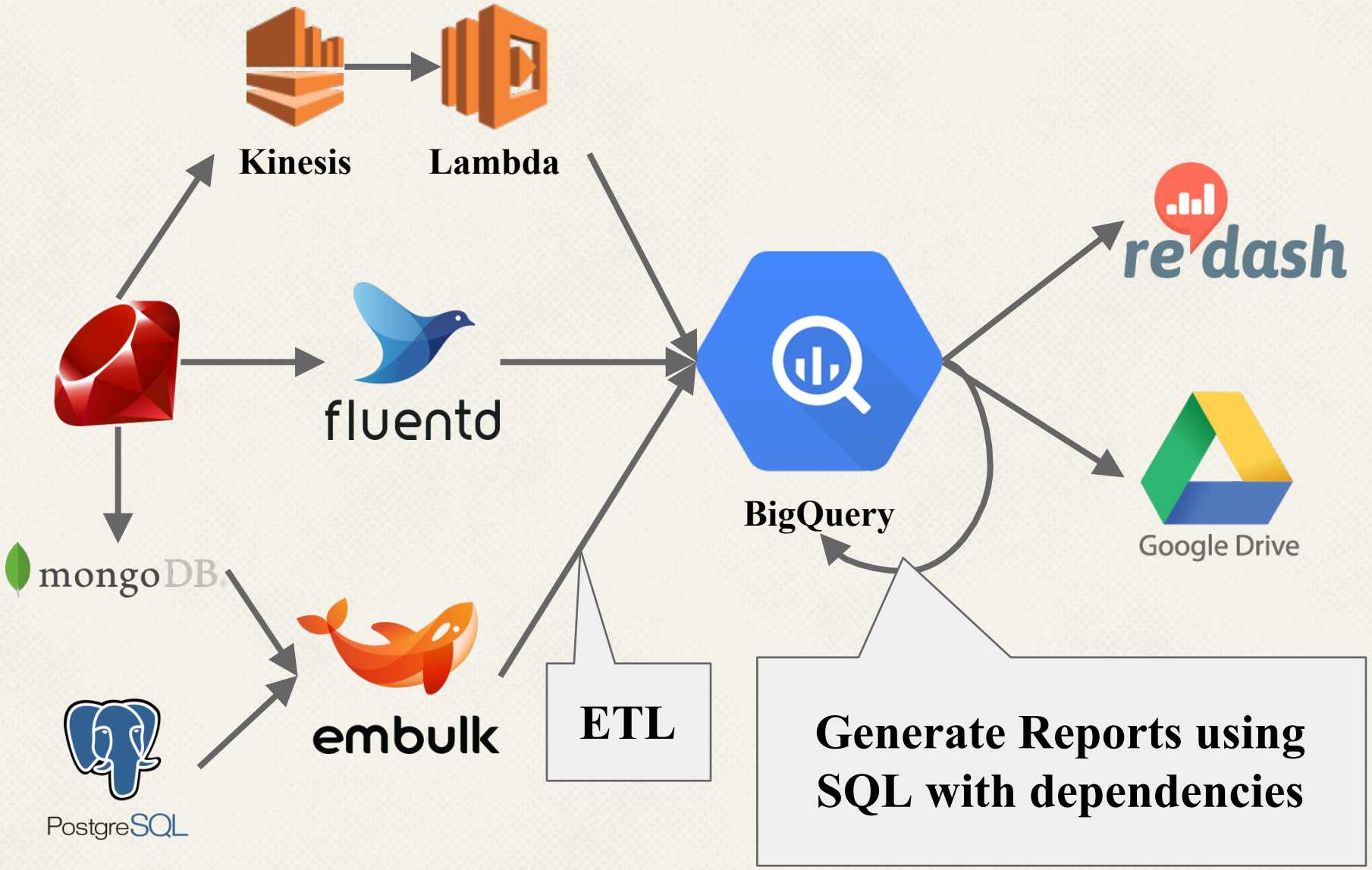
Examples of Data Pipeline

- ETL (Extract-Transform-Load)
  - Extract data from production database
  - Transform, cleaning, pre-processing data
  - Load it into report database
- Generate reports using SQL which has dependencies

Examples of Data Pipeline



Examples of Data Pipeline



How much data does Quipper process?

Daily Activity

100GB

Events and Logs are inserted

4TB

Scans

40k queries

Executed

This is not so big, but not so small.

So we have to consider about how to write complex data pipeline more easy, and run it stable.

Why writing complex data pipeline so difficult?

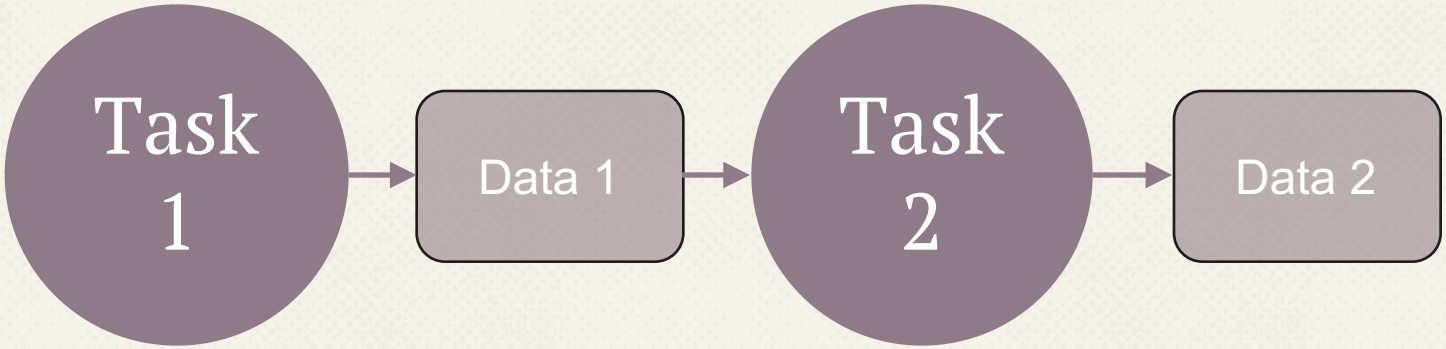
Why writing complex data pipeline so difficult?

Because we have to consider many things **which is not directly related to data and task** such as

- Dependency resolution
- Error handling
- Idempotence
- Retry
- Logging
- Parallel execution

Write data pipeline in Ruby (Early Days)

Early Days Example



Let’s write this simple data pipeline in Ruby.

Early Days Example (1)

```
do_task1()
do_task2()
```

- ✓ Dependency resolution
- Error handling
- Idempotence
- Logging
- Retry

Early Days Example (2)

```
begin
  do_task1()
  do_task2()
resque => e
  exit 1
```

- ✓ Dependency resolution
- ✓ Error handling
- Idempotence
- Logging
- Retry

Early Days Example (3)

```
begin
  do_task1() unless data1_exist?
  do_task2() unless data2_exist?
resque => e
  exit 1
```

- ✓ Dependency resolution
- ✓ Error handling
- ✓ Idempotence
- Logging
- Retry

Early Days Example (4)

```
begin
  log.info(“start task1”)
  do_task1() unless data1_exist?
  log.info(“success task1”)
  log.info(“start task2”)
  do_task2() unless data2_exist?
  log.info(“success task2”)
resque => e
  log.error(e.message)
  exit 1
```

- ✓ Dependency resolution
- ✓ Error handling
- ✓ Idempotence
- ✓ Logging

  Retry

Early Days Example (5)

```
Retriable.configure do … end
begin
  Retriable.retriable do
    log.info(“start task1”)
    do_task1() unless data1_exist?
    log.info(“success task1”)
    log.info(“start task2”)
    do_task2() unless data2_exist?
    log.info(“success task2”)
  end
resque => e
  log.error(e.message)
  exit 1
```

- ✓ Dependency resolution
- ✓ Error handling
- ✓ Idempotence
- ✓ Logging
- ✓ Retry

Too many boilerplate!💢

I need a solution!!

Solution

Workflow Engine can solve this problem.

There are famous Open Source Workflow Engine

- Luigi by Spotify
- Airflow by AirBnB
  - Now become Apache Incubator

They are awesome, but
they are for Python🐍,
not for Ruby💎!!

So I made it myself

tumugi

A open source plugin based ruby library to build, run and manage complex workflows

Document: https://tumugi.github.io/

Source: https://github.com/tumugi/tumugi

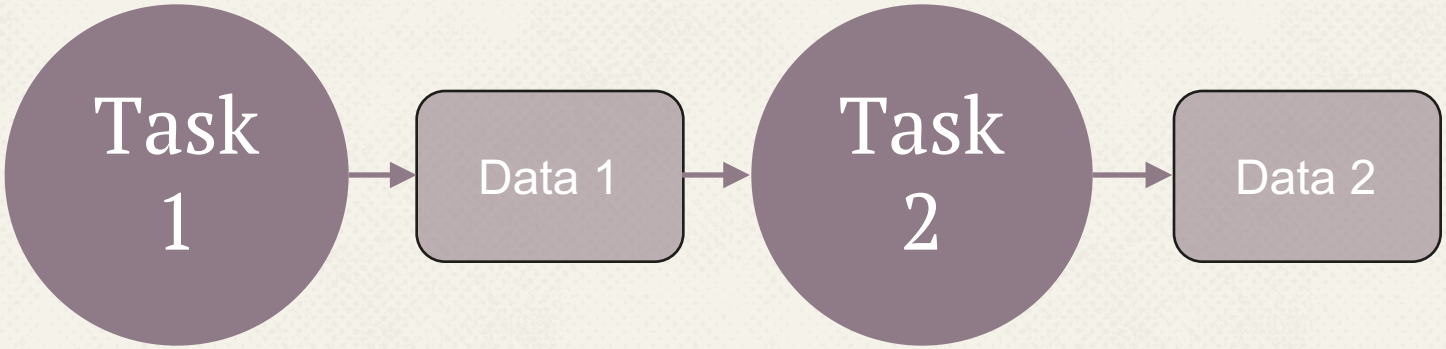
What tumugi provides

- Define workflow using internal DSL
- Task dependency resolution
- Error handling and retry
- Support build idempotence data pipeline
- Centralized logging
- Parallel task execution using threads
- Visualize
- Plugin architecture

What tumugi does not provide

- Scheduler (Event Trigger)
  - Use external tools like cron, Jenkins
- Executor for multiple machines
  - One data pipeline can only in one machine
    - Control cloud distributed resources
      - Eg. BigQuery, Cloud Storage
    - Sync multiple data pipeline using external task

Data Pipeline with tumugi



Let’s write this very simple data pipeline using tumugi.

Data Pipeline with tumugi

data pipeline with tumugi (1)

```
task :task1 do
  output target(:data1)
  run { “do something” }
end

task :task2 do
  requires :task1
  output target(:data2)
  run { output.write(“do something using #{input.value}”) }
end
```

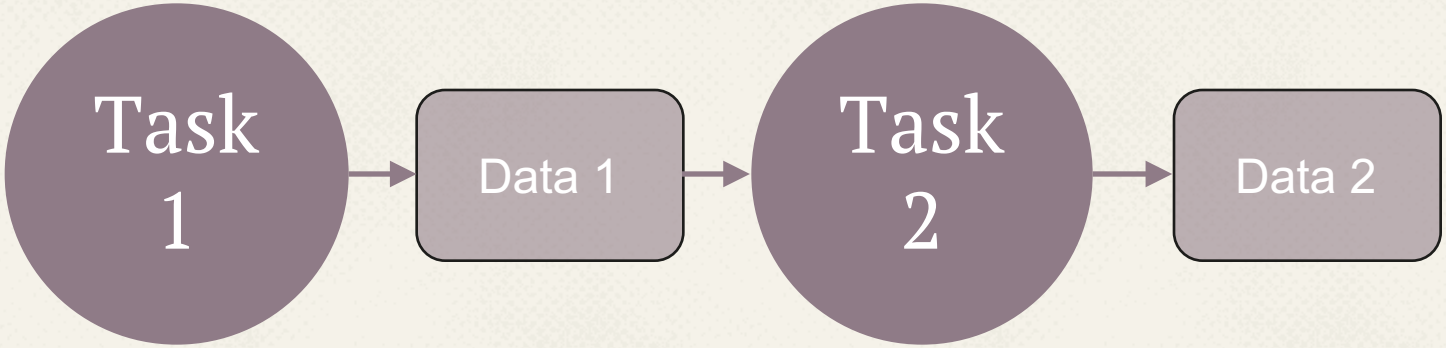
Before and After

Ruby only

```
Retriable.configure do … end
begin
  Retriable.retriable do
    log.info(“start task1”)
    do_task1() unless data1_exist?
    log.info(“success task1”)
    log.info(“start task2”)
    do_task2() unless data2_exist?
    log.info(“success task2”)
  end
resque => e
  log.error(e.message)
  exit 1
```

tumugi DSL

```
task :task1 do
  output target(:data1)
  run { output.write(“do something”) }
end

task :task2 do
  requires :task1
  output target(:data2)
  run {
    output.write(
      “do something using #{input.value}”) }
  }
end
```




```
task :task1 do
  output target(:data1)
  run { output.write("do something") }
end

task :task2 do
  requires :task1
  output target(:data2)
  run { output.write("do something using #{input.value}") }
end
```

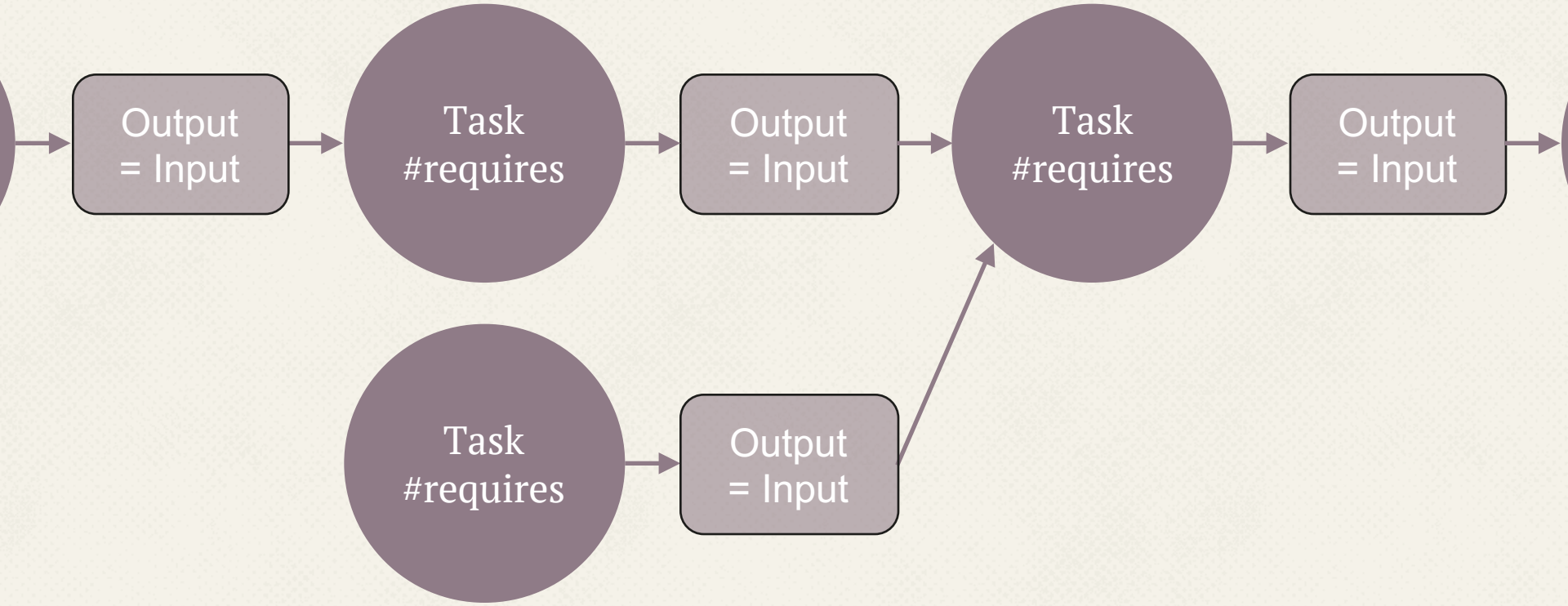
How to abstract data pipeline in Ruby

Core Components of tumugi

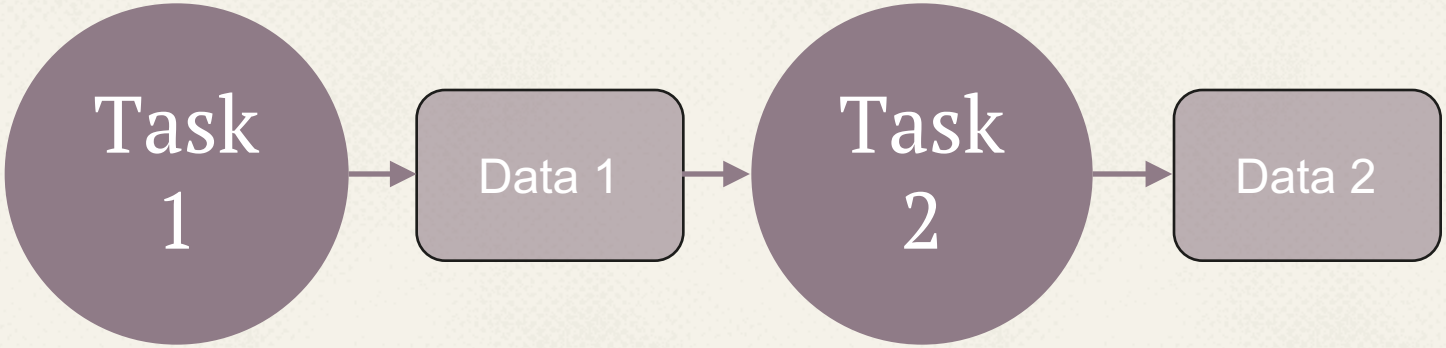
What is Task?

Task is represents a `task` in data pipeline.
Write your own data process logic in #run method using input data and generate output data.

Task Dependency



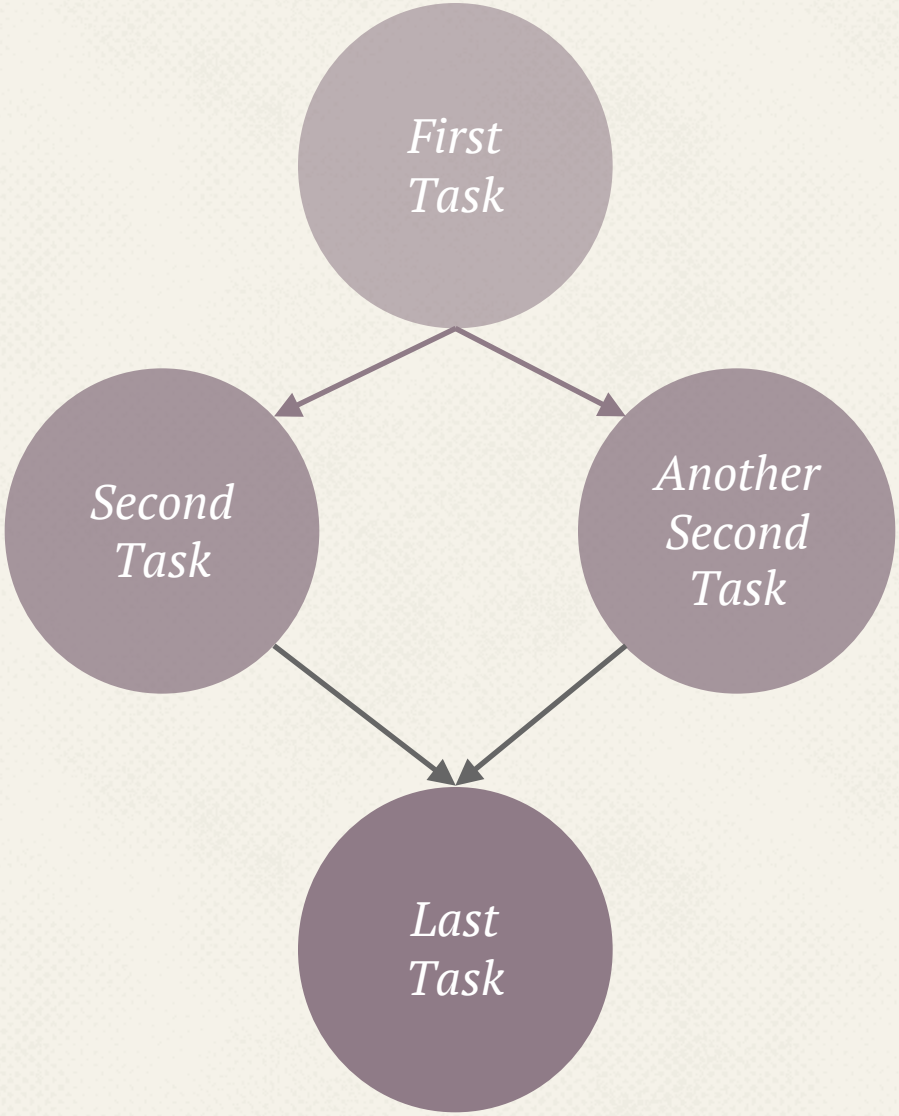
Task also has `#requires` method, which write dependency of tasks.
This information used by tumugi to build DAG.




```
task :task1 do
  output target(:data1)
  run { output.write("do something") }
end

task :task2 do
  requires :task1
  output target(:data2)
  run { output.write("do something using #{input.value}") }
end
```

Requires multiple tasks

```
task :first_task do
end

task :second_task do
  requires :first_task
end

task :another_second_task do
  requires :first_task
end

task :last_task do
  requires [:second_task,
            :another_second_task]
end
```



What is Target?

Target abstract `data` in data pipeline.
E.g. Local File, Remote Object, Datbase Table, etc
Target must implements #exist? method

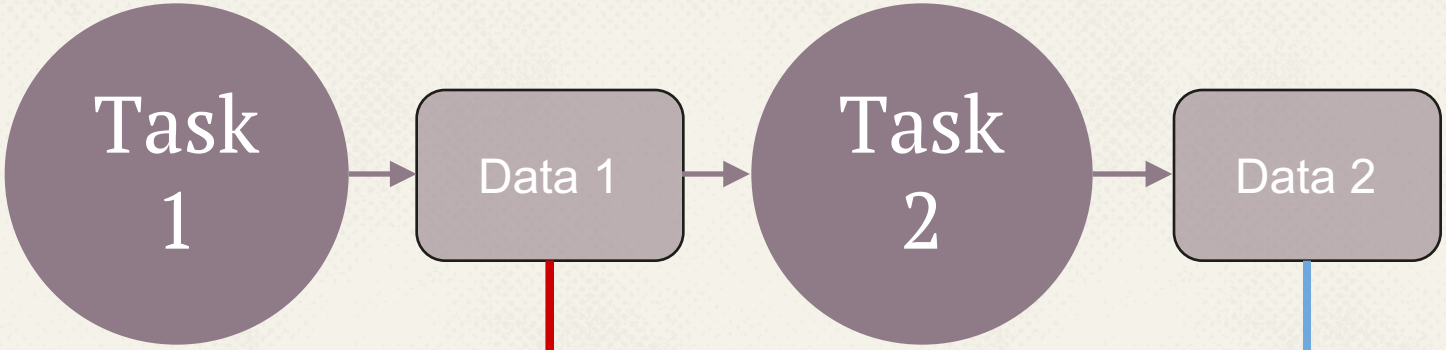
Example of Target

```
class LocalFileTarget < Tumugi::Target
  attr_reader :path

  def initialize(path)
    @path = path
  end

  def exist?
    File.exist?(@path)
  end
end
```

Target#exist? method is for Idempotence

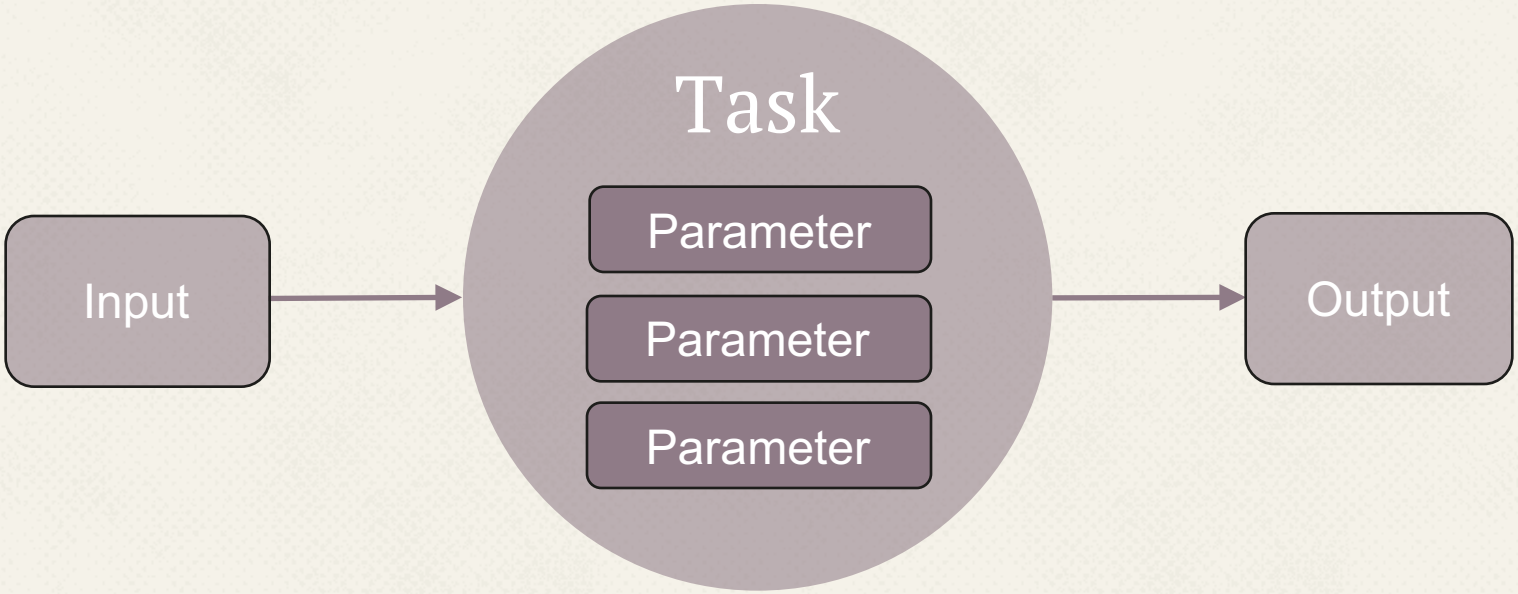
```
while !dag.complete?
  task = dag.next()
  if task.output.exist? #<= HERE!
    task.trigger_event(:skip)
  elsif task.ready?
    task.run && task.trigger_event(:complete)
  else
    dag.push(task)
  end
end
```

Target#exist? used by tumugi to build idempotence data pipeline.




```
task :task1 do
  output target(:data1)
  run { output.write("do something") }
end

task :task2 do
  requires :task1
  output target(:data2)
  run { output.write("do something using #{input.value}") }
end
```

What is Parameter?



Parameter is another input data of Task.
You can read parameter in Task#run method.

Example of Parameters

```
task :some_task do
  param :key1, type: :string, default: ‘value1’
end

task :another_task, type: :awesome_plugin do
  key2 ‘another param name’
  run do
    puts key2 #=> ‘another param name`
  end
end
```

- You can define parameter using #param
- You can read parameter in #run method
- You can override parameters of plugin in DSL

Parameter Auto Binding

```
task :some_task do
  param :param1, type: :string, auto_bind: true
  run { puts param1 } #=> “hello”
end
```

```
$ tumugi run -f workflow.rb -p param1:hello
```

If a parameter auto_binding is enabled,
tumugi bind parameter value from CLI options.

Plugin Architecture

Everything is plugin

All tasks and targets are plugin

Task Plugin

Definition

```
module Tumugi
  module Plugin
    class AwesomeTask < Tumugi::Task
      Plugin.register_task('awesome', self)
      param :key
      def run
        puts key
      end
    end
  end
end
```

Usage

```
task :task1, type: :awesome do
  key "value"
end
```

Target Plugin

Definition

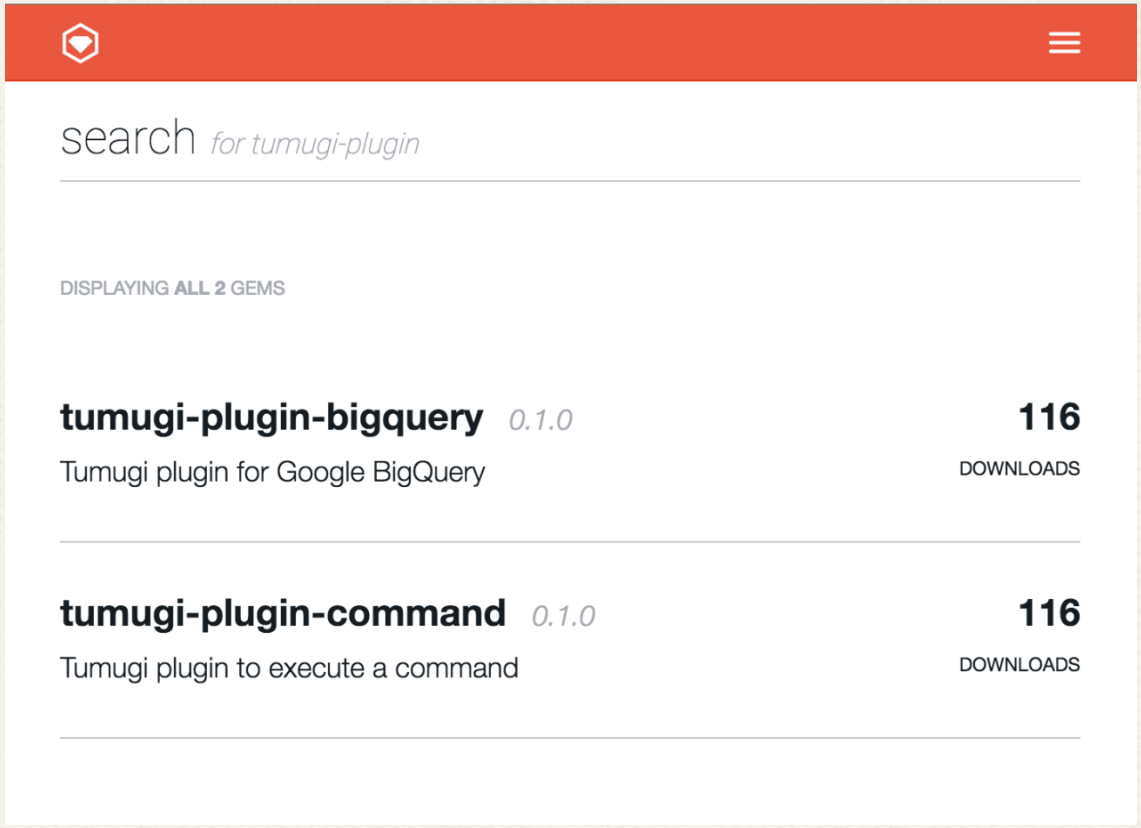
```
module Tumugi
  module Plugin
    class AwesomeTarget < Tumugi::Target
      Plugin.register_target('awesome', self)
      def exist?
        # check awesome resource is exist or not
      end
    end
  end
end
```

Usage

```
task :task1 do
  output target :awesome
end
```

Distribute tumugi plugin

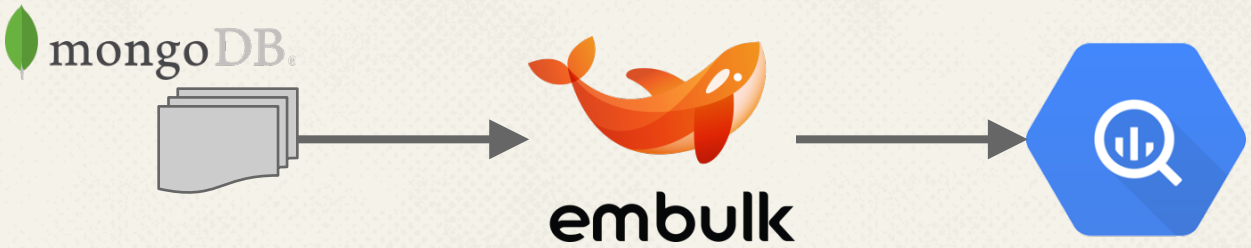
You can find and register tumugi plugins in RubyGems.org
Each gem has useful tasks and targets.
https://rubygems.org/search?query=tumugi-plugin



Real world example

Dynamic Workflow

Export multiple MongoDB collections using embulk




```
configs = Dir.glob("schema/*.json")

task :main do
  requires configs.map {|config|
    "#{File.basename(config, ".*")}_export"
  }
  run { log "done" }
end
```

```
configs.each do |config|
  collection = File.basename(config, ".*")
  task "#{collection}_export", type: :command do
    param :day, auto_bind: true, required: true
    command {
      "embulk_wrapper --day=#{day} --collection=#{collection}"
    }
    output {
      target(:bigquery_table,
             project_id: "xxx", dataset_id: "yyy",
             table_id: "#{collection}_#{day}") }
  end
end
```

Run query, export and notify

DEMO:
Export BigQuery query result to Google Drive and notify URL to Slack.

http://tumugi.github.io/recipe2/

THANKS!

Any questions?

You can find me at:
https://github.com/hakobera on Github
@hakobera on Twitter