Simple Ray Tracer with the Apache Beam Go SDK

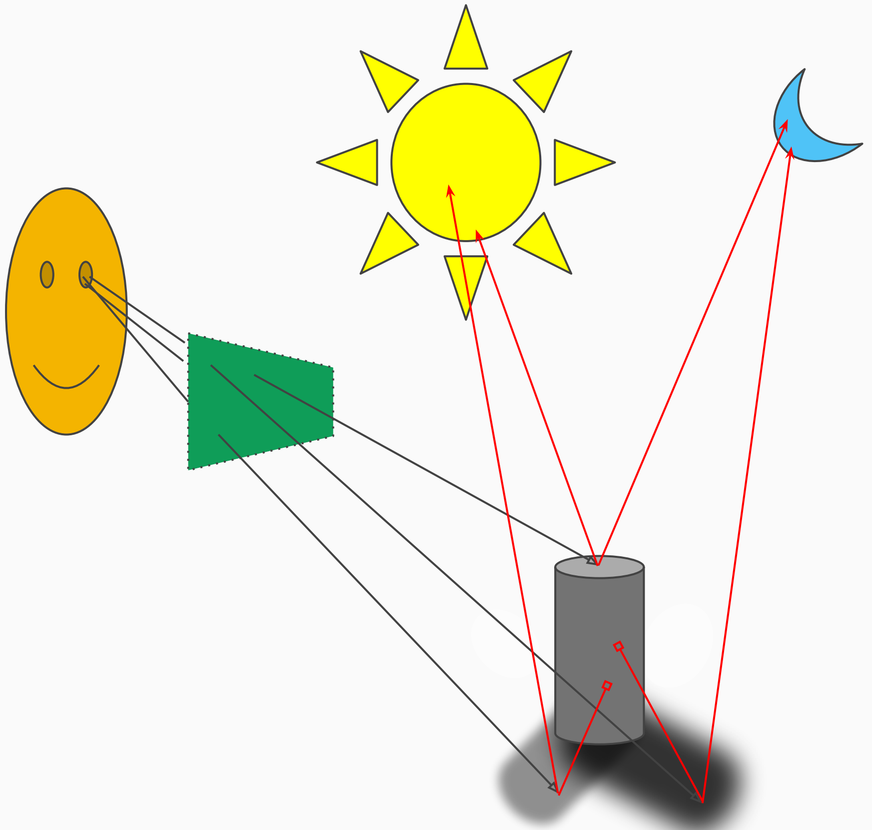
Robert Burke (@lostluck) Beam Summit 2021

Learning Goals

- How do Ray Tracers work
- How to write one with the Beam Go SDK
- How to use SpliitableDoFns with it
- Debugging Beam Go
- Executing on a Distributed Runner

What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects

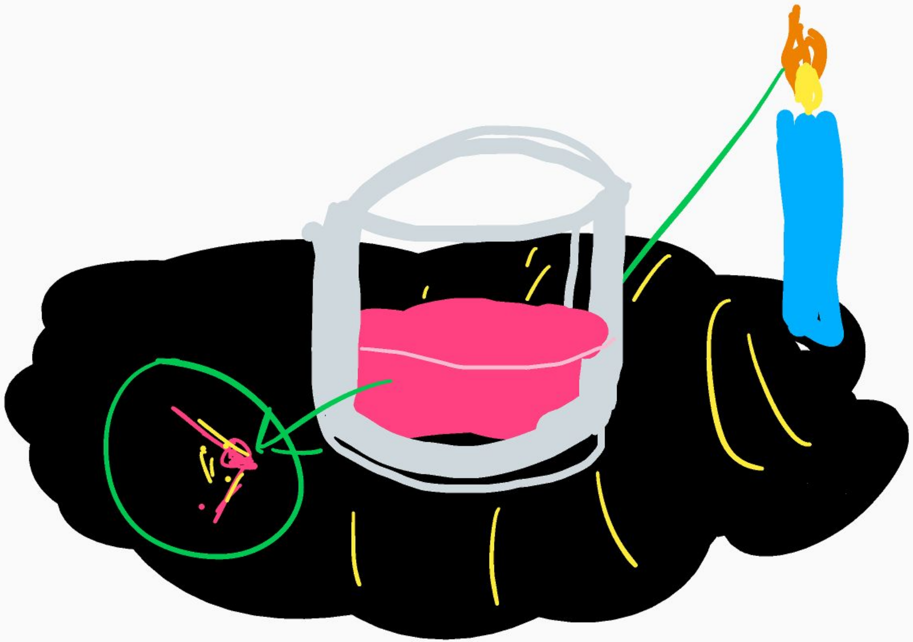


What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects

What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects

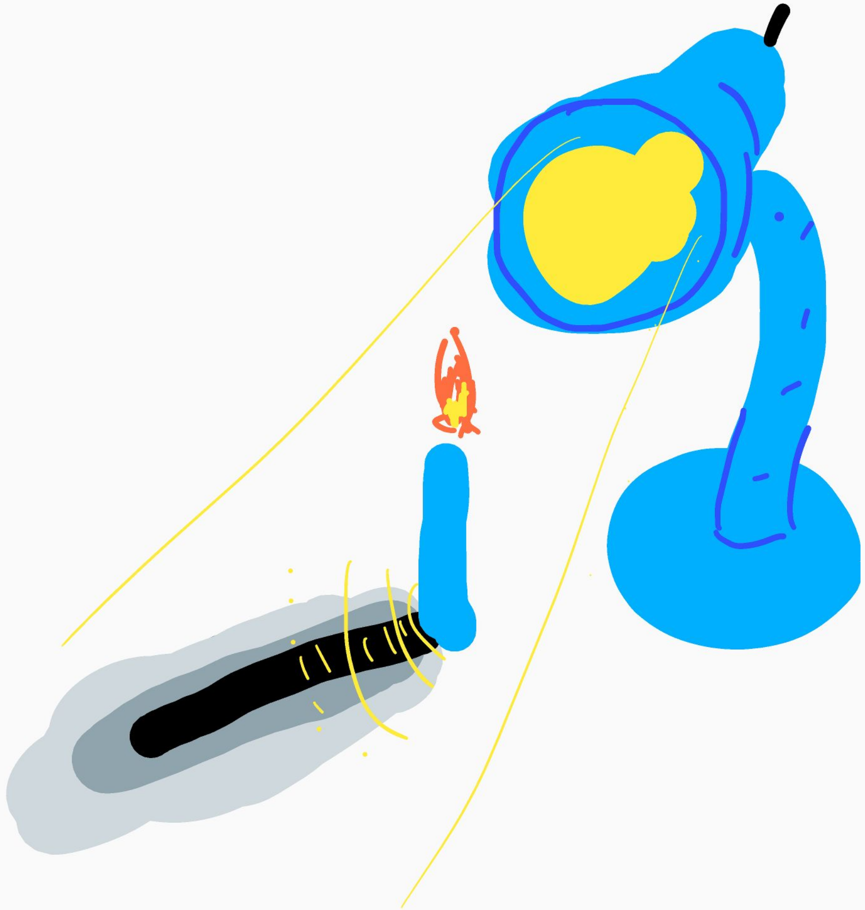


What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects

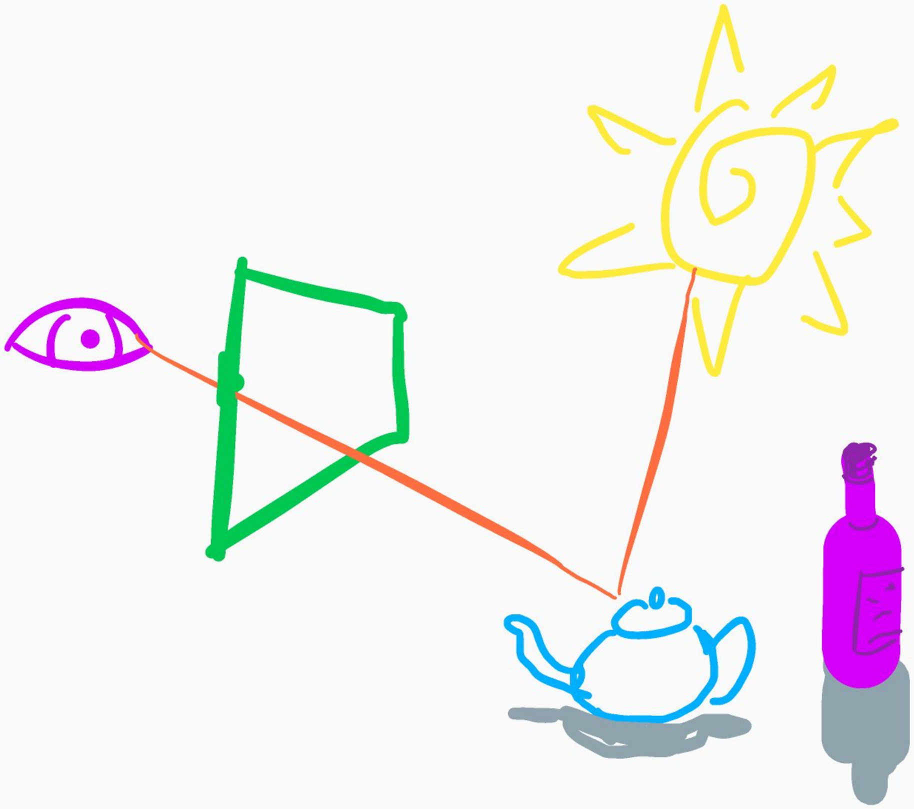
What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects

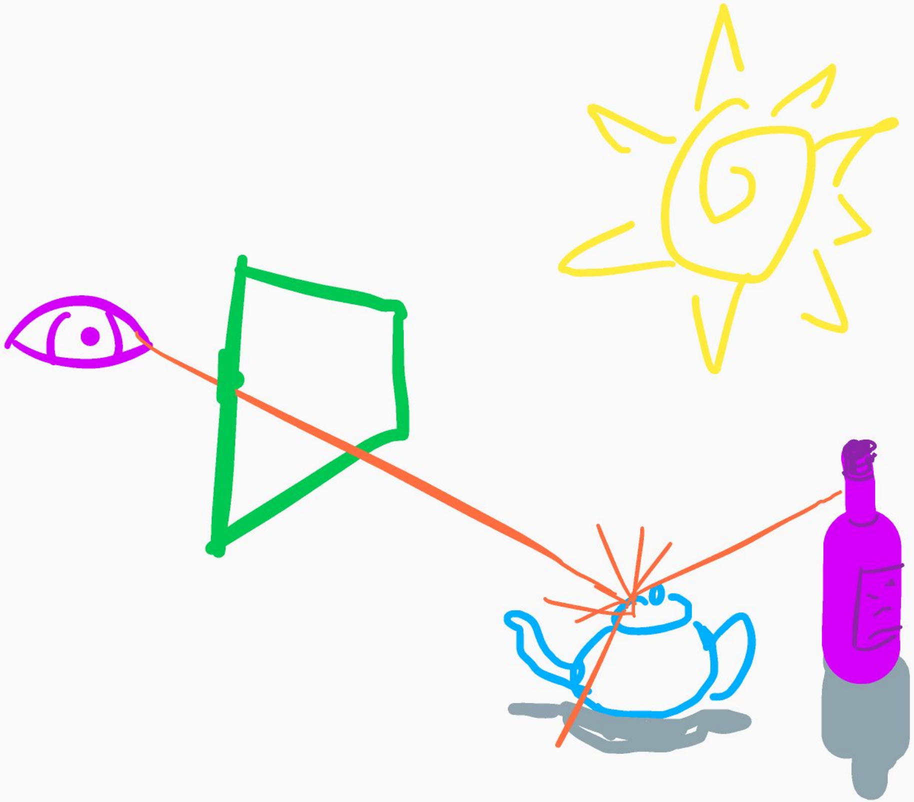


What is a Ray Tracer?

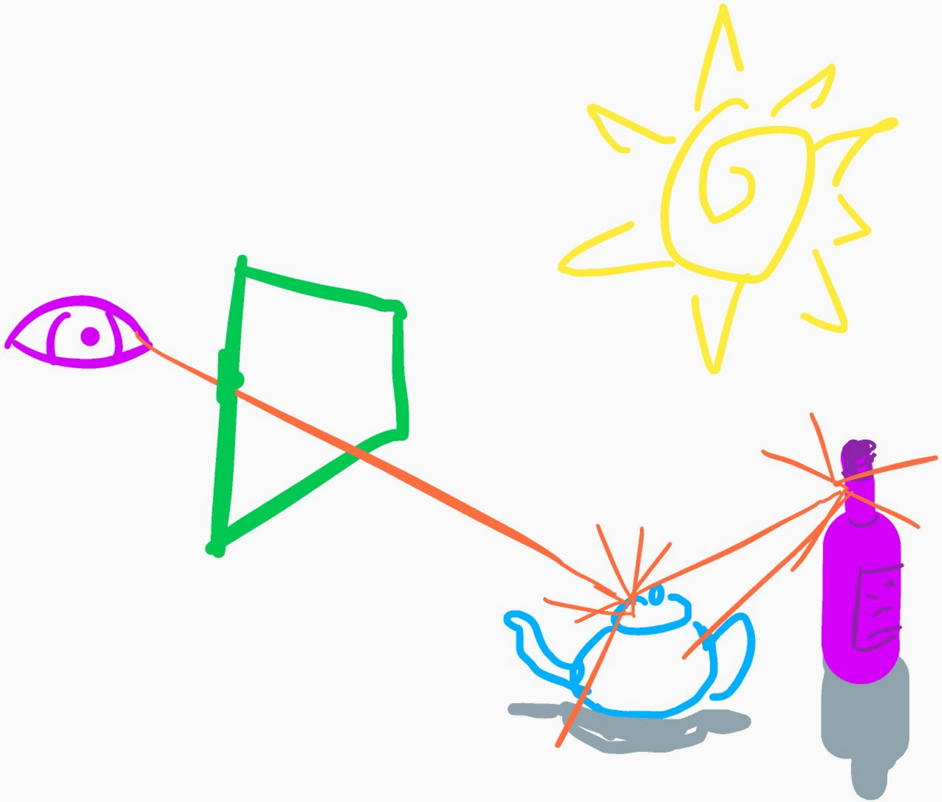
- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects



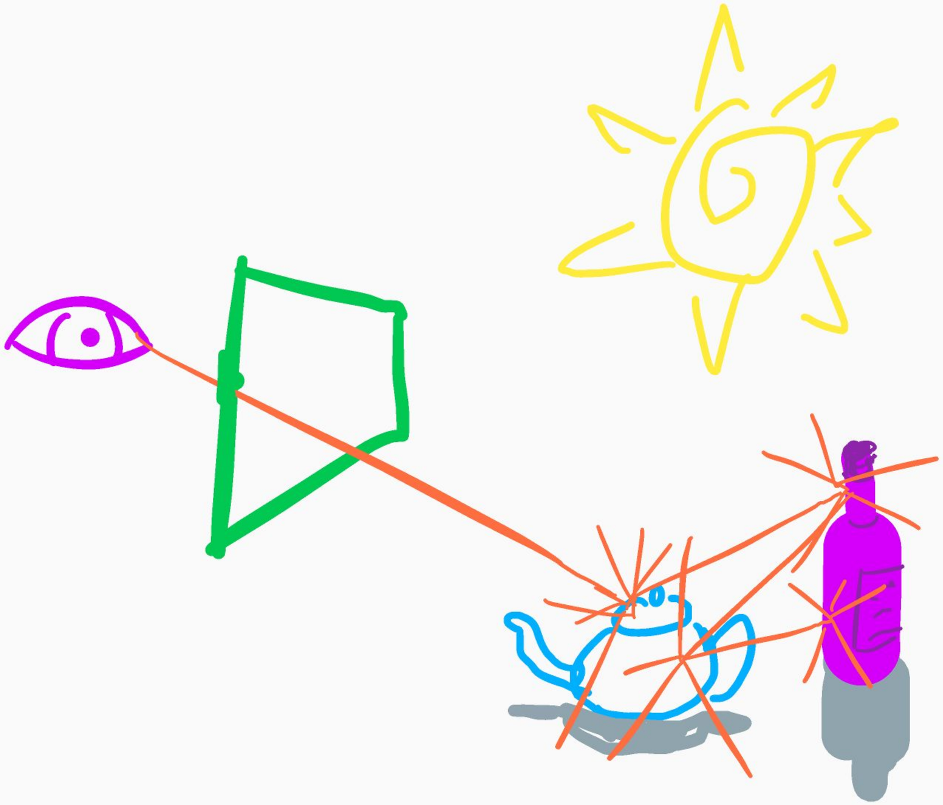
A Ray is cast



Additional rays are cast



Further Additional rays are cast



Further Additional rays are cast

- Read in the scene and it’s configuration options
- Set up the camera
- For each pixel:
    - Cast sampling rays from the camera to the scene
    - Find the object in the scene the ray intersects with
    - Depending on the properties of the object
        - Cast additional sampling rays to determine the color of the object
            - These can be called “bounces”
        - Stop when we hit the bounce limit
    - Accumulate the contribution from all sampling rays
    - Set the pixel color
- Save the image

https://github.com/lostluck/experimental/

Dividing Work

Splittable DoFn

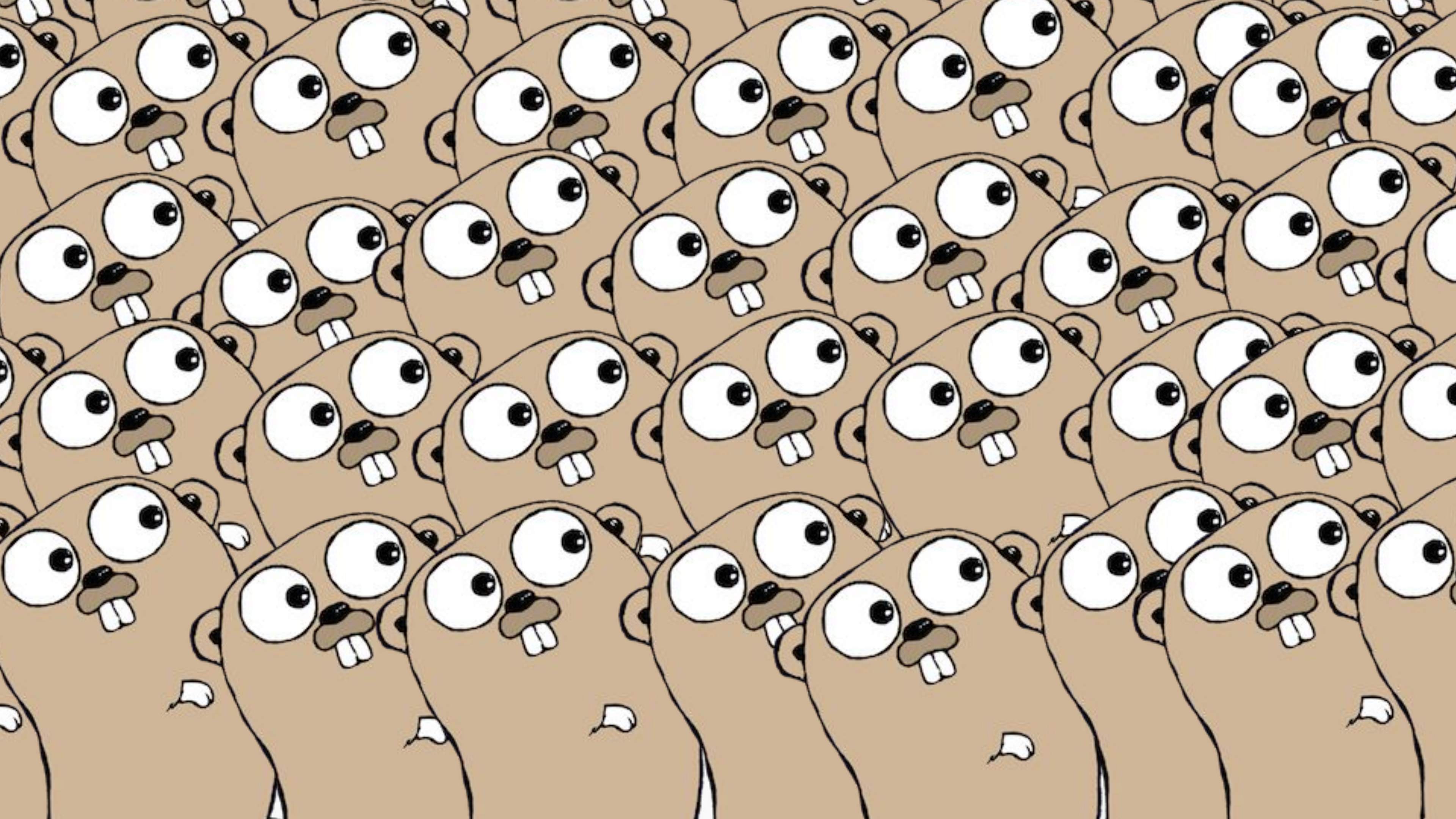
- Create a Restriction for an element
- Split a Restriction for a given element appropriately.
- Create Trackers for a restriction.
- Process the element with respect to the given restriction tracker.

Splittable DoFn

Element: The image being produced.

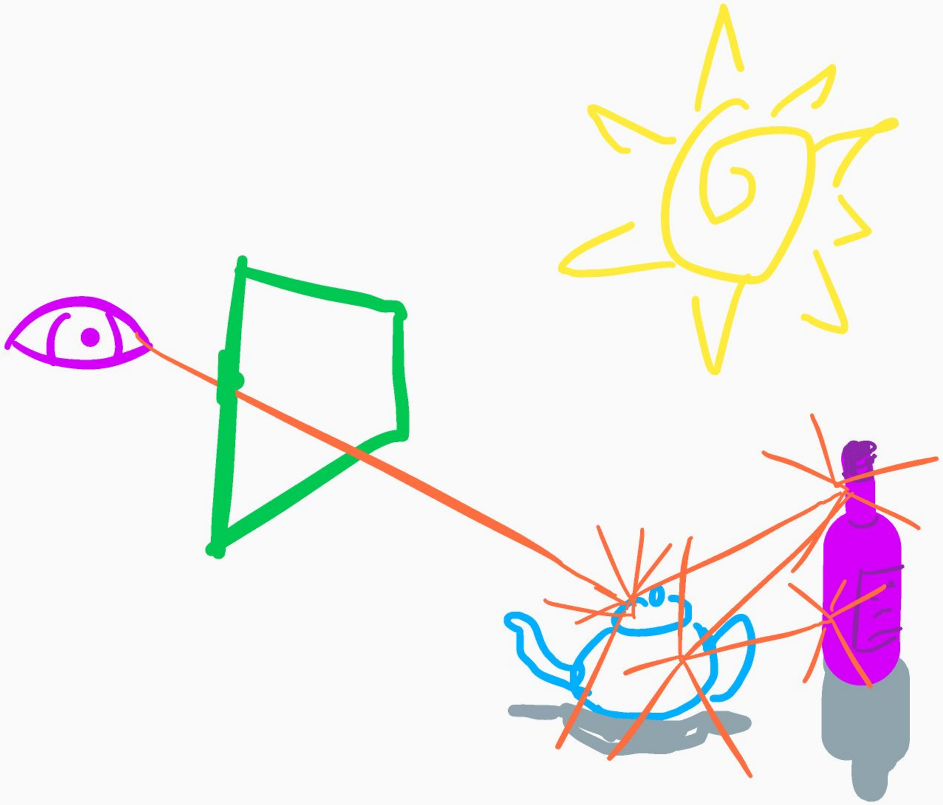
Restriction: Offset Ranges

- Enumerate each Sample from 1 to Width*Height*Samples.
- Decompose from numbers back to individual pixel coordinates.
- Easy to Split, built into Beam



```
type Ray struct {
  Xp,Yp,Zp float64 // Position
  Xv,Yv,Zv float64 // Vector
  Rc,Gc,Bc float64 // Color

  Xpx,Ypx int32    // Pixel
  Bounce, ID int16 // SampleID
}
```

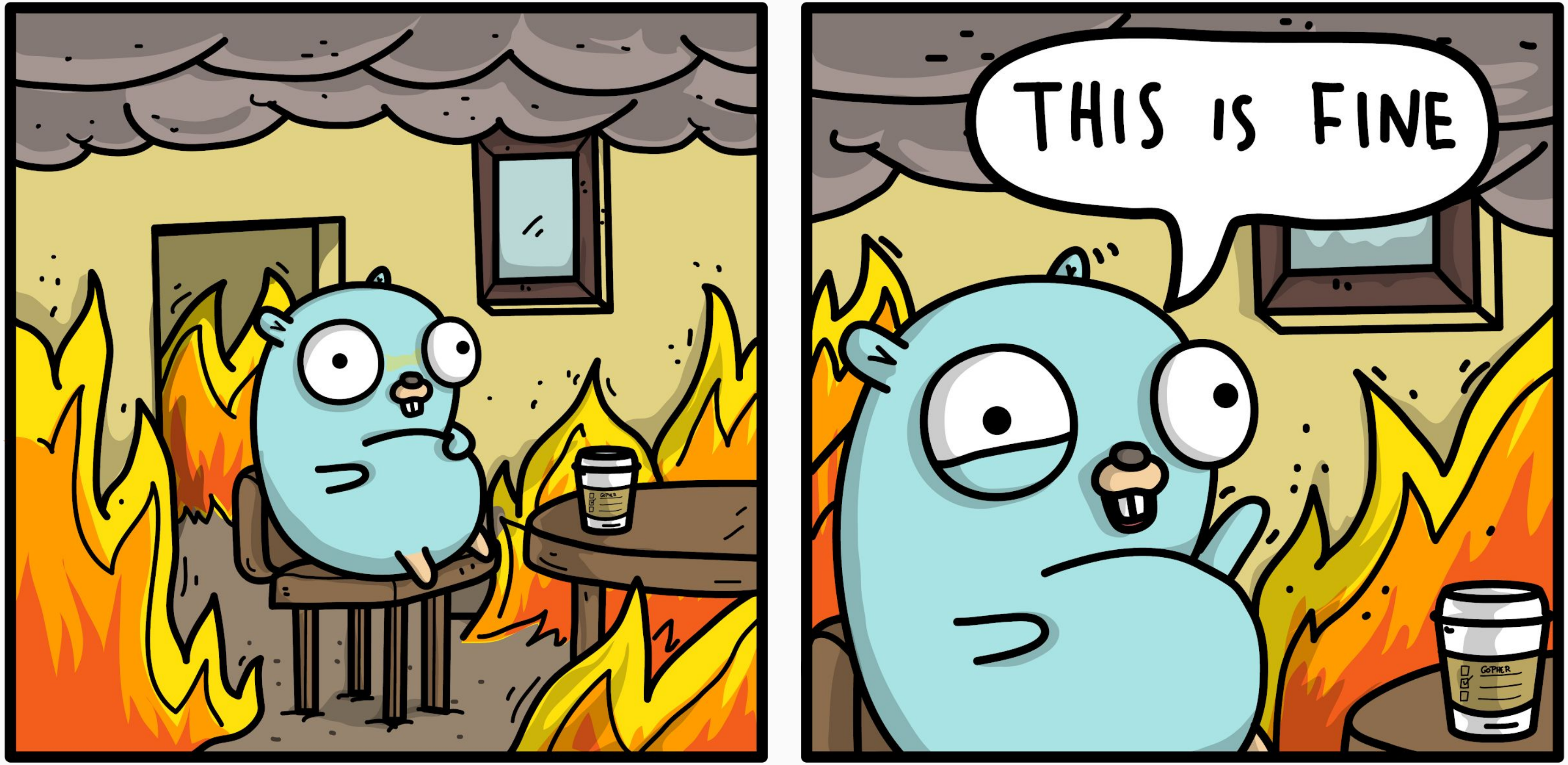


The Problem

8M px

× 4096 rays/px

× 88 bytes/ray $\simeq$ 3 Tb



Art by @ashleymcnamara, “this is fine” by KC Greene, Gophers by Renee French

Debugging Beam Go Pipelines

Local Debugging

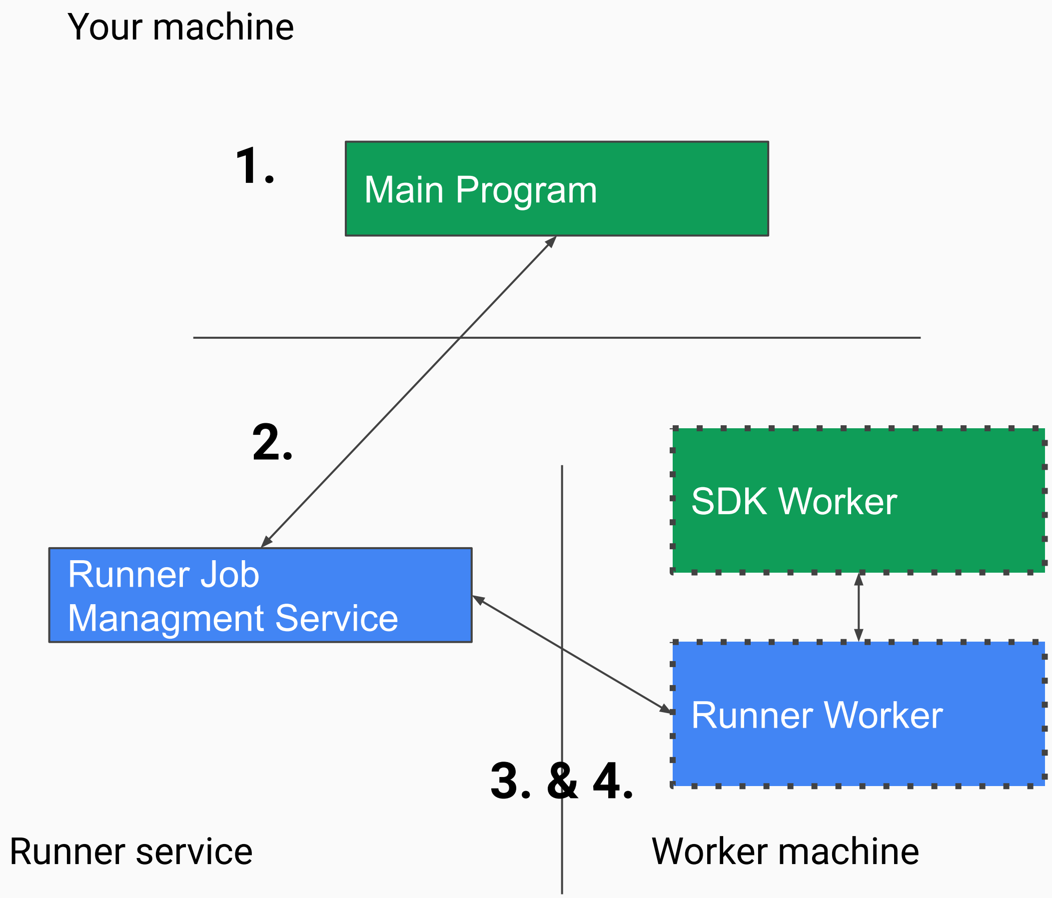
- Unit test your code
- Counters
- Local portable runners and LOOPBACK mode
- Profile your code

Counters

Local Runners and LOOPBACK Mode

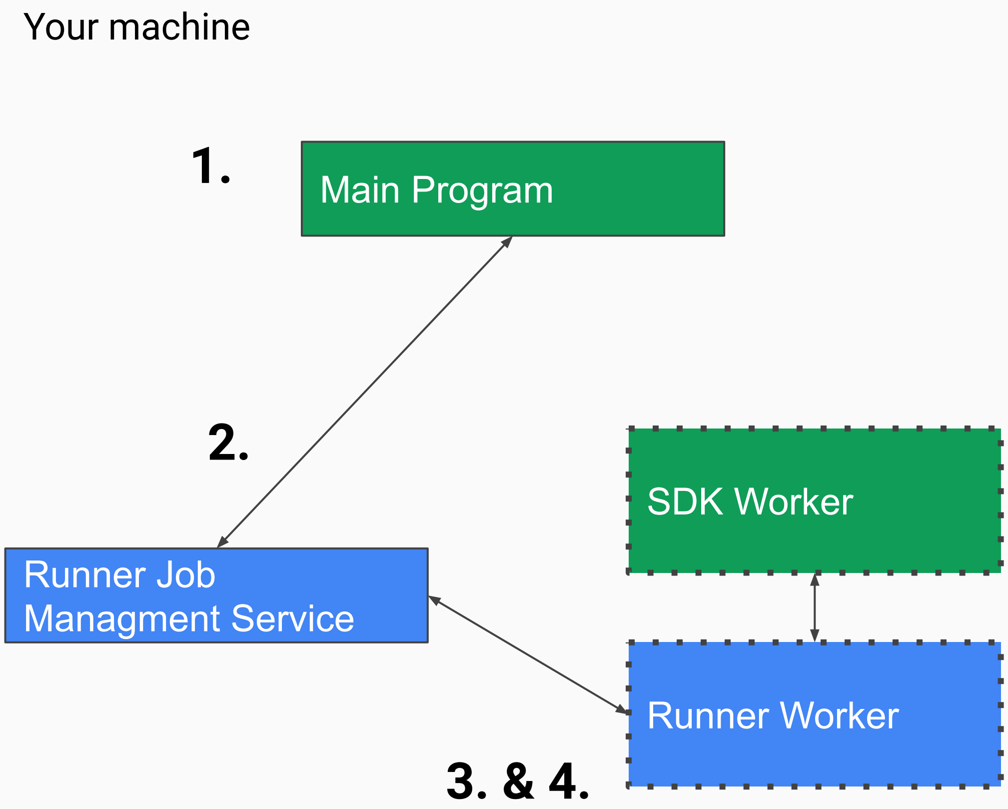
Distributed Execution

1. Main Program starts up and constructs the pipeline object.
2. Sends the worker artifact etc to the runner.
3. As needed, Runner starts SDK Worker and Runner Worker containers, which fetch the worker artifact.
4. Runner assigns workers bundles to execute until termination



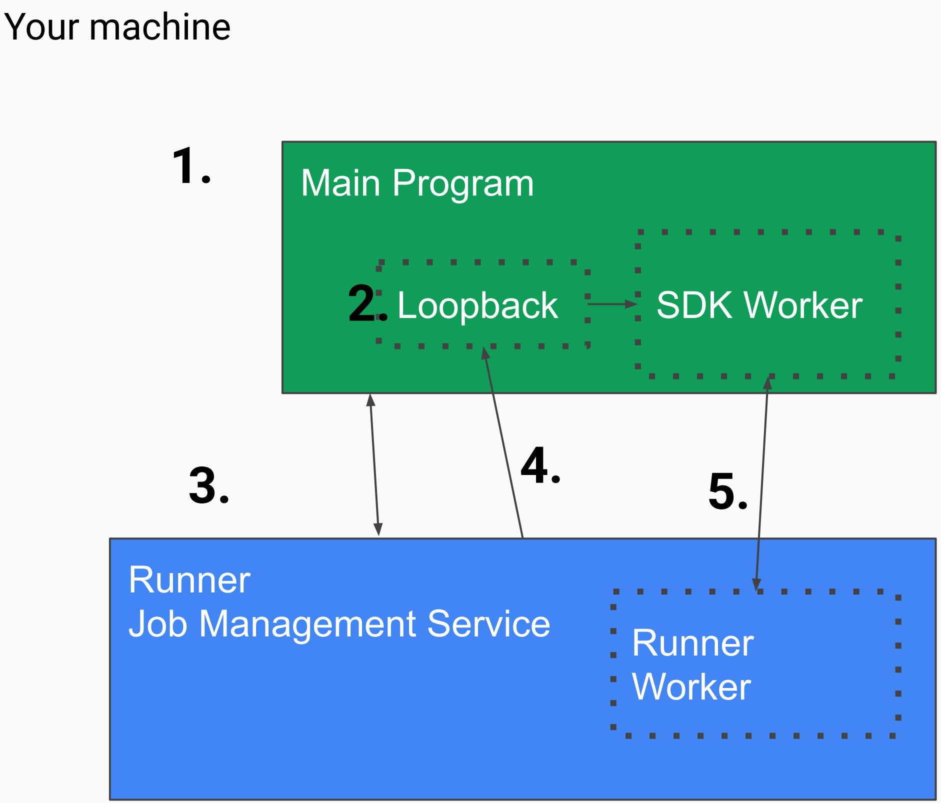
~~Distributed~~
Local Execution

1. Main Program starts up and constructs the pipeline object.
2. Sends the worker artifact etc to the runner.
3. As needed, Runner starts SDK Worker and Runner Worker containers, which fetch the worker artifact.
4. Runner assigns workers bundles to execute until termination



LOOPBACK Execution

1. Main Program starts up and constructs the pipeline object.
2. Runner tells Main program to start a LOOPBACK server, to create SDK Workers
3. Sends the worker artifact etc to the runner.
4. Runner spins up workers in the main program process via Loopback.
5. Runner assigns workers bundles to execute until termination



```
➜  gbrt git:(master) ✗ go run . --use_beam=true --word="FOO" --runner=universal --endpoint=localhost:8099
--environment_type=LOOPBACK --output_dir=/home/rebo/experimental/gbrt/images --samples=16 --cpu_profile=gb
rt.cpu.pprof

2021/08/02 08:14:59 bounces3.samples16.FOO.
2021/08/02 08:14:59 starting Loopback server at 127.0.0.1:44123
2021/08/02 08:14:59 components: <
  transforms: <
    key: "e1"
    value: <
      unique_name: "GenerateRays/Impulse"
      spec: <
        urn: "beam:transform:impulse:v1"
      >
      outputs: <
        key: "i0"
        value: "n1"
      >
    >
  >
  transforms: <
    key: "e10"
    value: <
      unique_name: "ToImage/lib.MakeImageFn"
```

Loopback mode enabled

Starting a job in LOOPBACK mode

Profiling

Profiling w/PProf

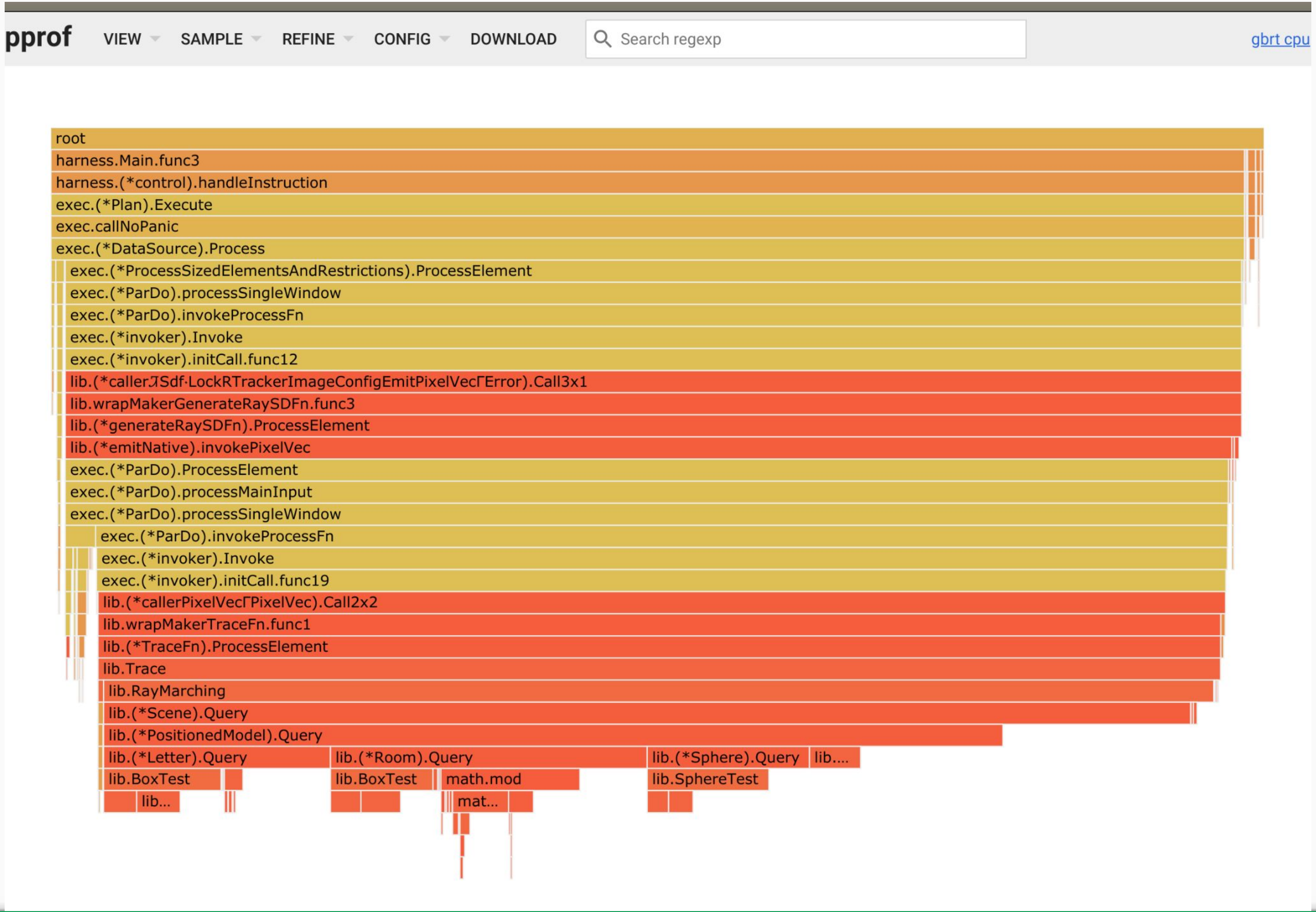
Add calls to `pprof.StartCPUProfile(f)` and `defer pprof.StopCPUProfile()` to your main()

$ <execute job locally, in LOOPBACK with profiling>

$ go install github.com/google/pprof

$ sudo apt-get install graphviz

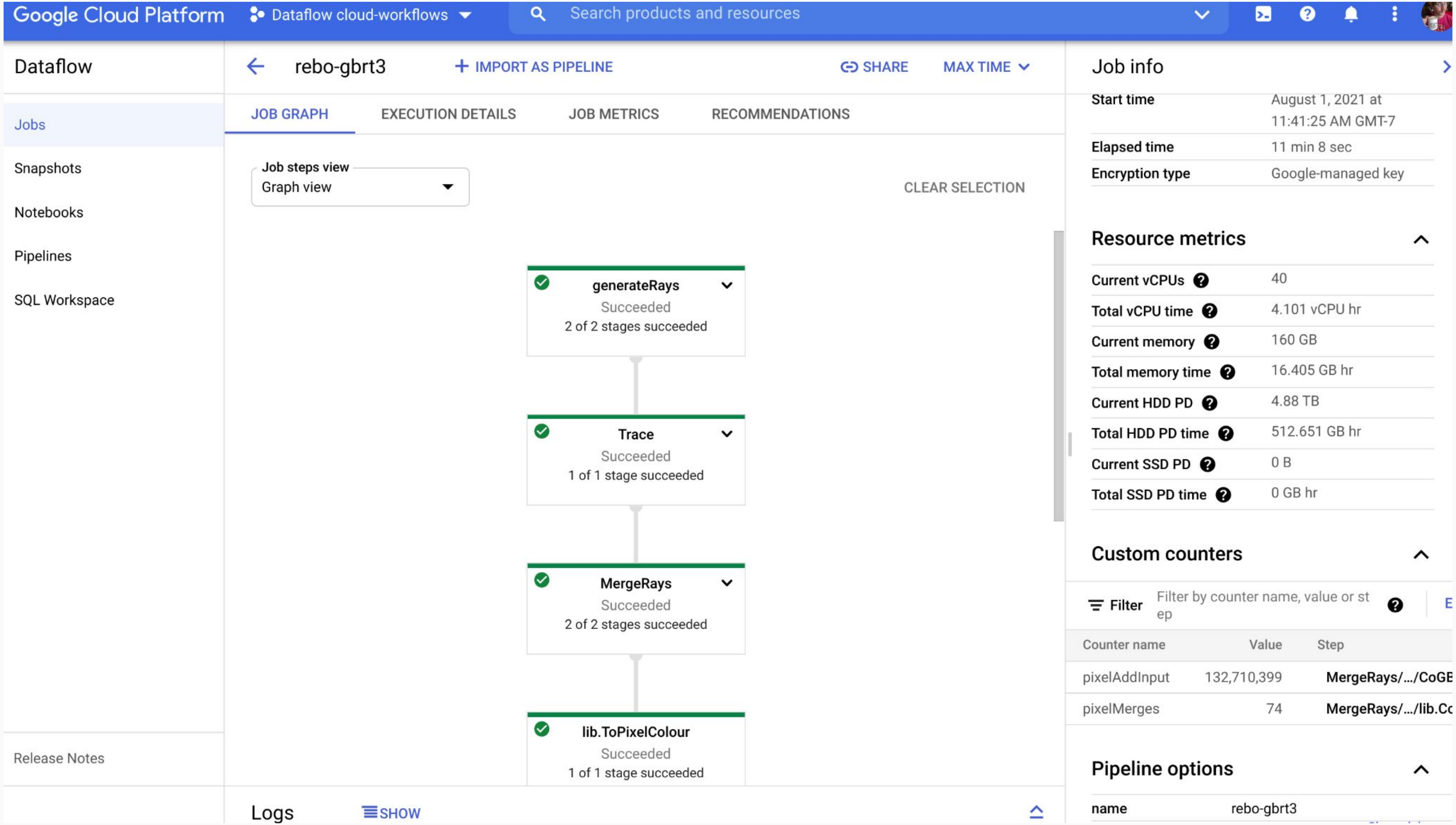
$ pprof --http=: <binary name> <profile file name>



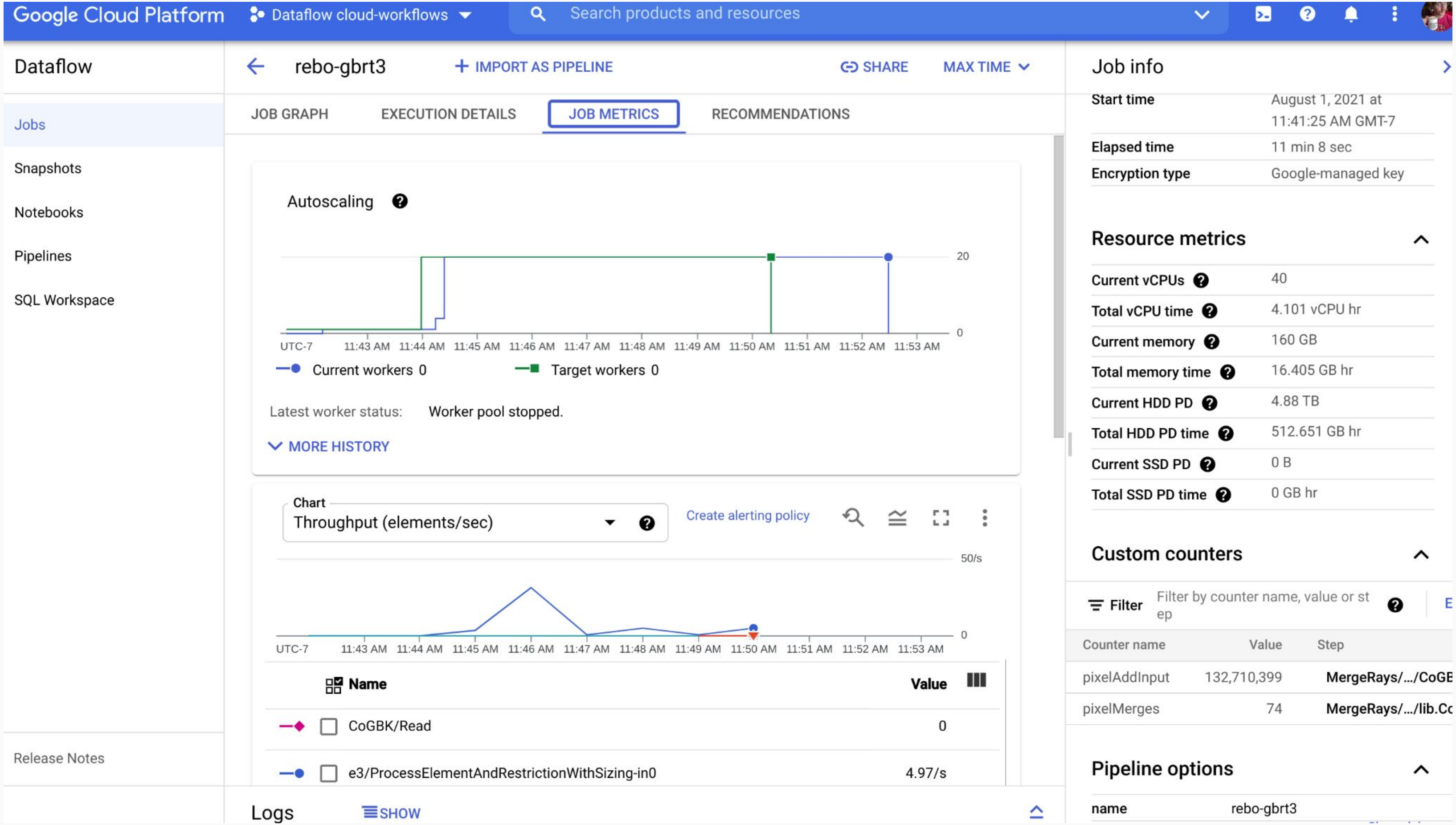
PProf Flame Graph

Distributed Runners

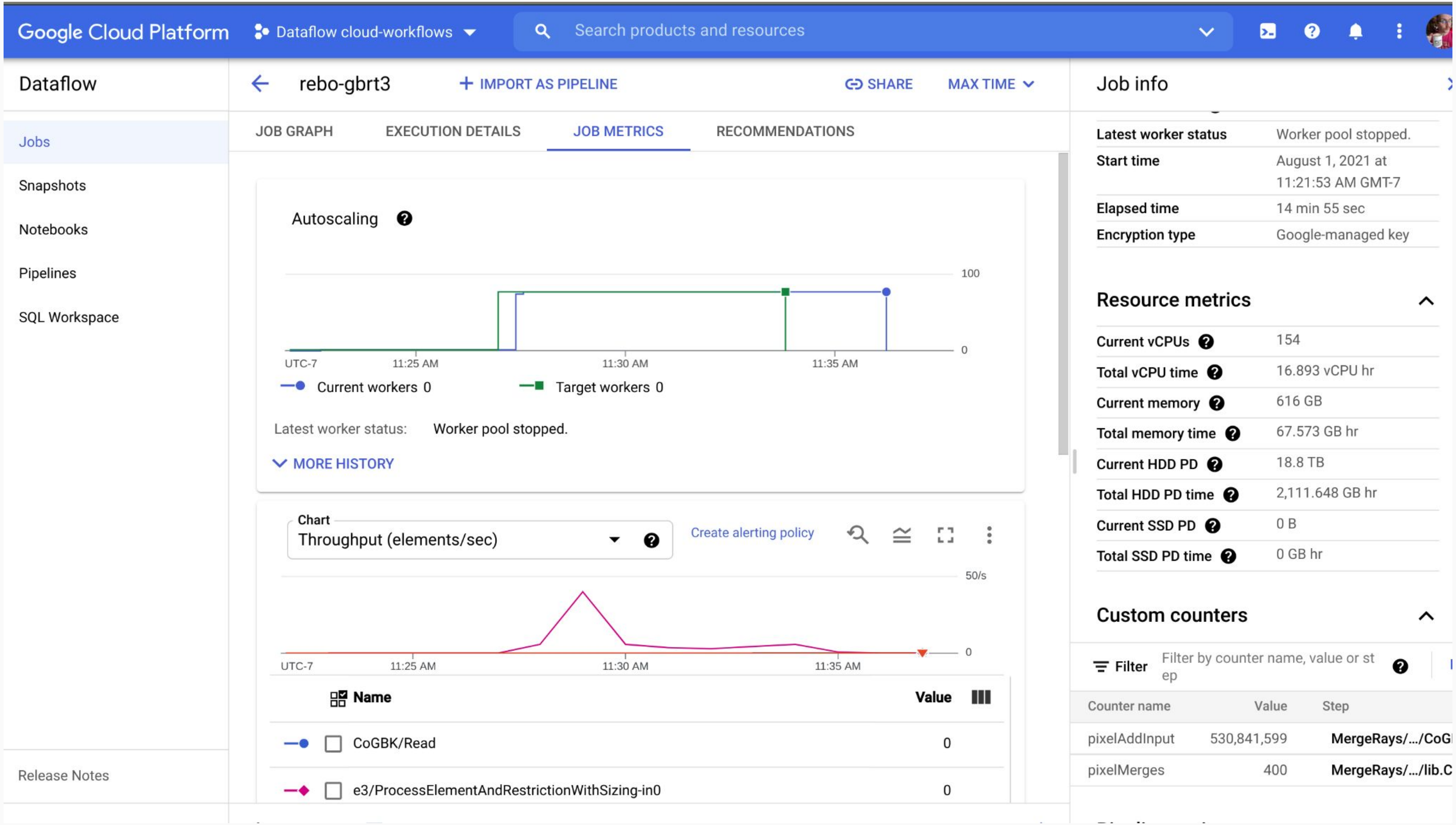
```
$  go run . --use_beam=true --word=GOOGLE \
--samples=1024 --bounces=5 \
--runner=dataflow \
--project=$PROJECT \
--staging_location=$STAGING_GCS \
--region=us-central1 \
--job_name=rebo-gbrt3 \
--environment_config=$SDK_CONTAINER
```



Job run on Google Cloud Dataflow



Job run on Google Cloud Dataflow



Google Cloud Platform
Dataflow cloud-workflows
Search products and resources
Dataflow
Jobs
Snapshots
Notebooks
Pipelines
SQL Workspace
Release Notes
rebo-gbrt3
IMPORT AS PIPELINE
SHARE
MAX TIME
JOB GRAPH
EXECUTION DETAILS
JOB METRICS
RECOMMENDATIONS
Autoscaling
100
0
UTC-7
11:25 AM
11:30 AM
11:35 AM
Current workers 0
Target workers 0
Latest worker status: Worker pool stopped.
MORE HISTORY
Chart
Throughput (elements/sec)
Create alerting policy
50/s
0
UTC-7
11:25 AM
11:30 AM
11:35 AM
Name
Value
CoGBK/Read 0
e3/ProcessElementAndRestrictionWithSizing-in0 0
Job info
Latest worker status Worker pool stopped.
Start time August 1, 2021 at 11:21:53 AM GMT-7
Elapsed time 14 min 55 sec
Encryption type Google-managed key
Resource metrics
Current vCPUs 154
Total vCPU time 16.893 vCPU hr
Current memory 616 GB
Total memory time 67.573 GB hr
Current HDD PD 18.8 TB
Total HDD PD time 2,111.648 GB hr
Current SSD PD 0 B
Total SSD PD time 0 GB hr
Custom counters
Filter
Filter by counter name, value or step
Counter name
Value
Step
pixelAddInput 530,841,599 MergeRays/.../CoG
pixelMerges 400 MergeRays/.../lib.C


THANKS

Learning Goals

- How do Ray Tracers work
- How to write one with the Beam Go SDK
- How to use SpliitableDoFns with it
- Debugging Beam Go
- Executing on a Distributed Runner

Fabien Sanglard and Andrew Kensler
https://fabiensanglard.net/postcard_pathtracer/

The Beam Summit organizers

Viewers like you.

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t)\, L_i(\mathbf{x}, \omega_i, \lambda, t)\, (\omega_i \cdot \mathbf{n})\, \mathrm{d}\,\omega_i$$