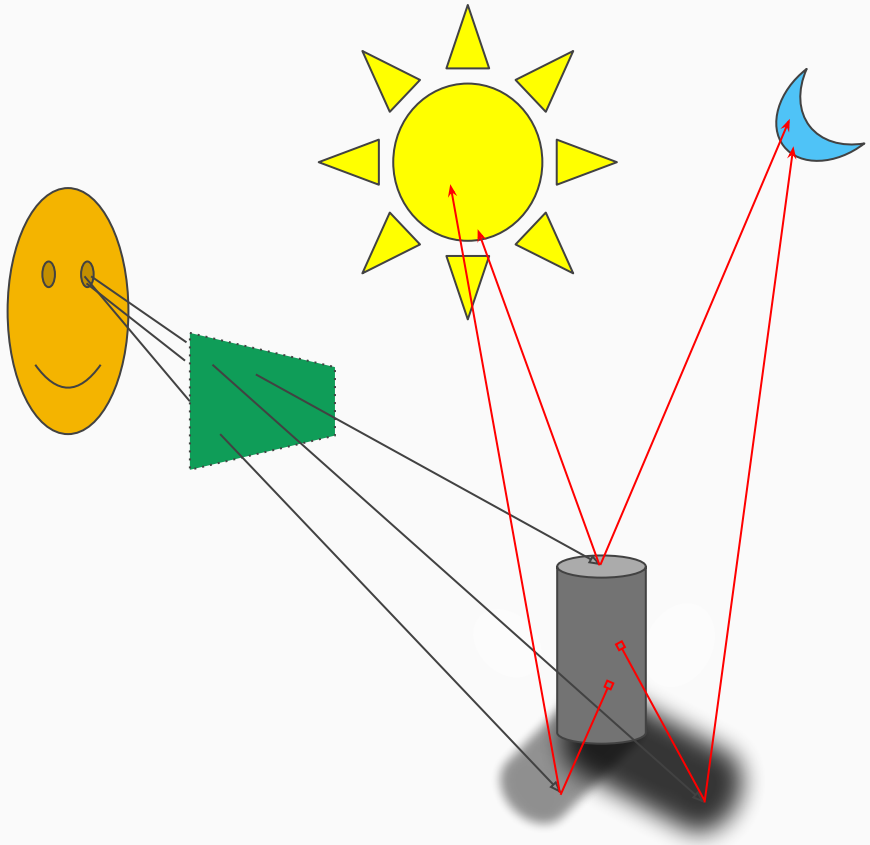
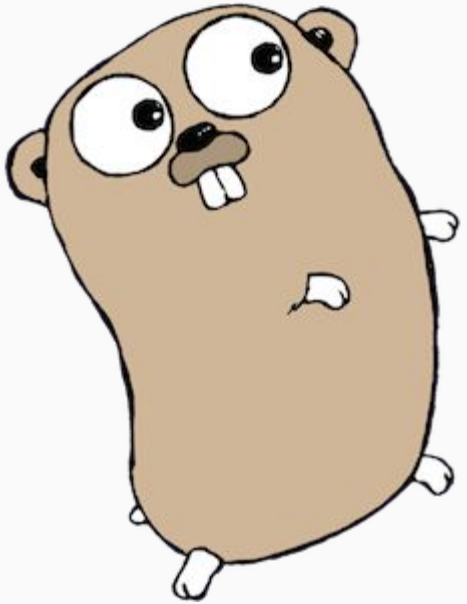
Writing a Distributed Ray Tracer with Apache Beam, Abridged

Robert Burke (@lostluck) Berlin Buzzwords 2019



beam


GO

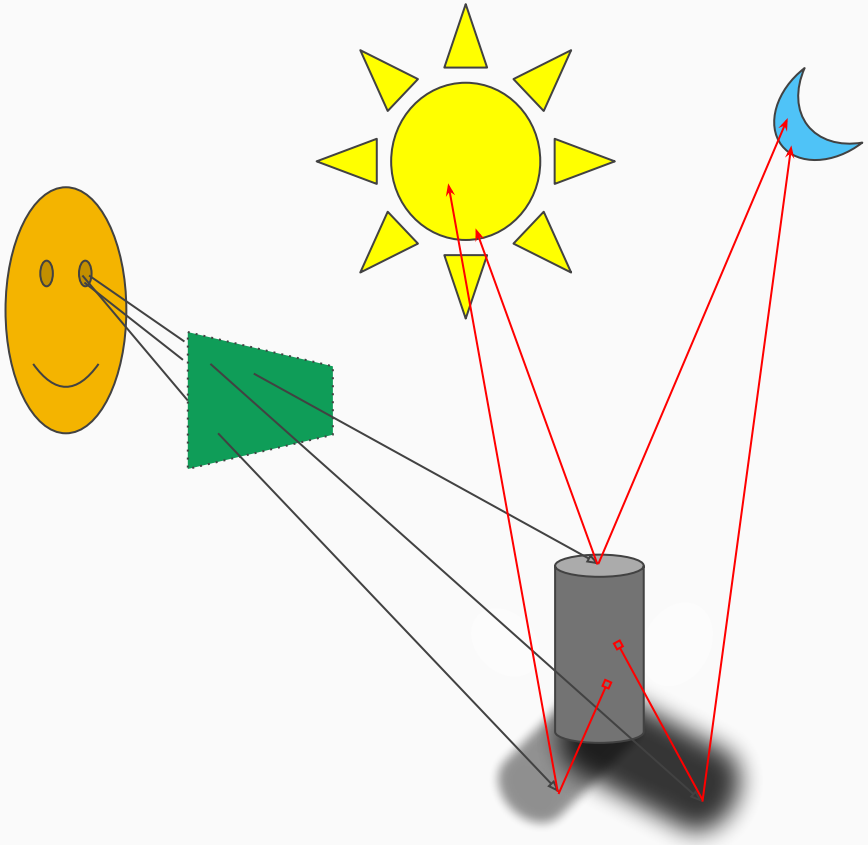


Learning Goals

- What is…
  - a Ray Tracer?
  - Apache Beam?
- Why you'd want to write one with the other

What is a Ray Tracer?

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

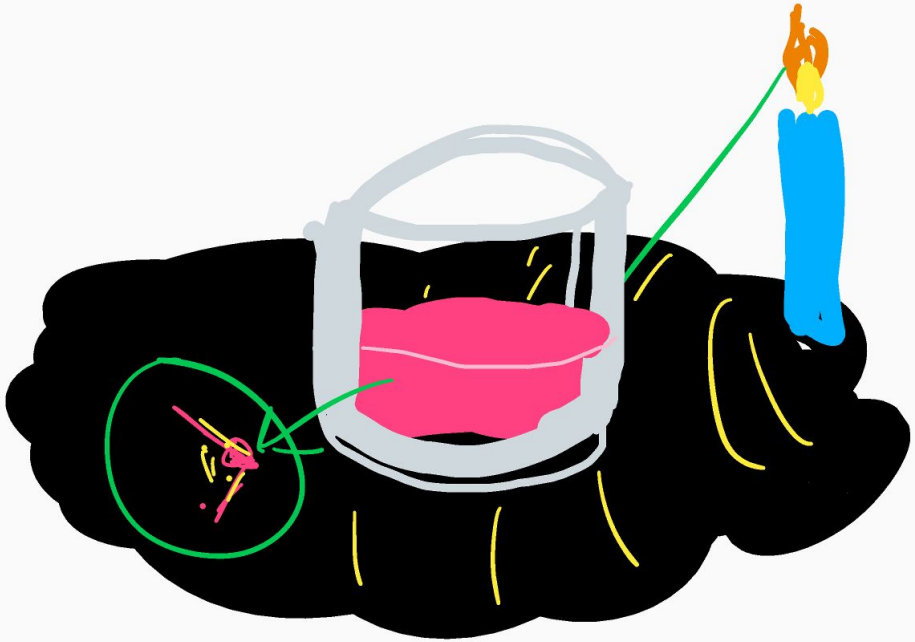


What is a Ray Tracer?

- 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

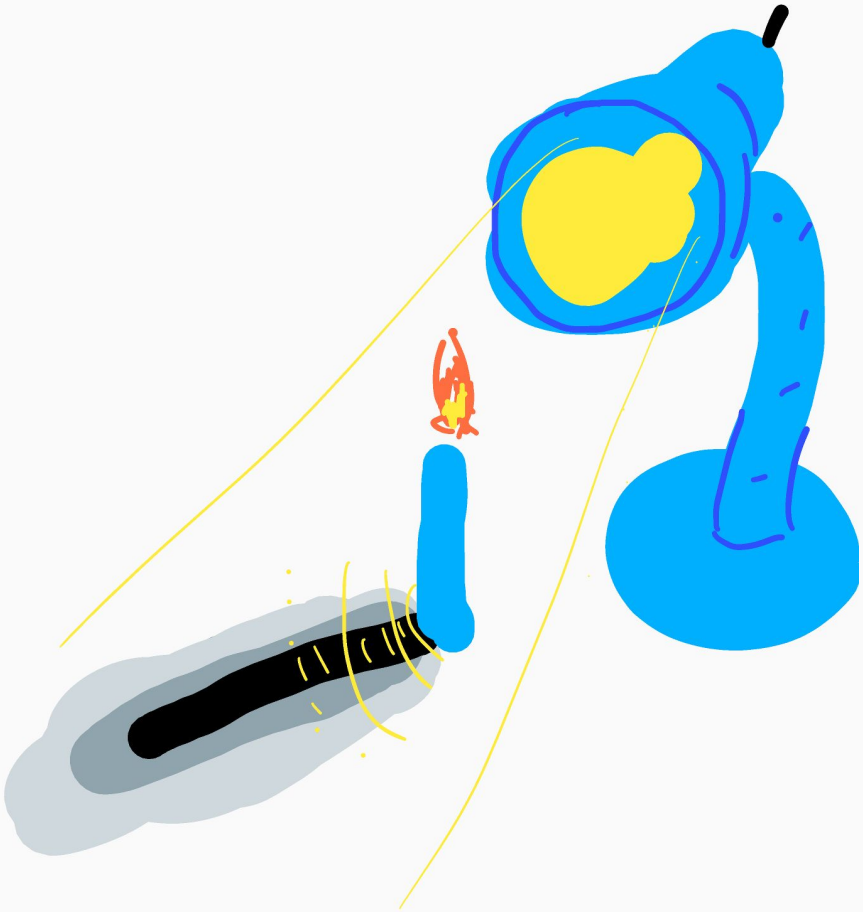


What is a Ray Tracer?

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

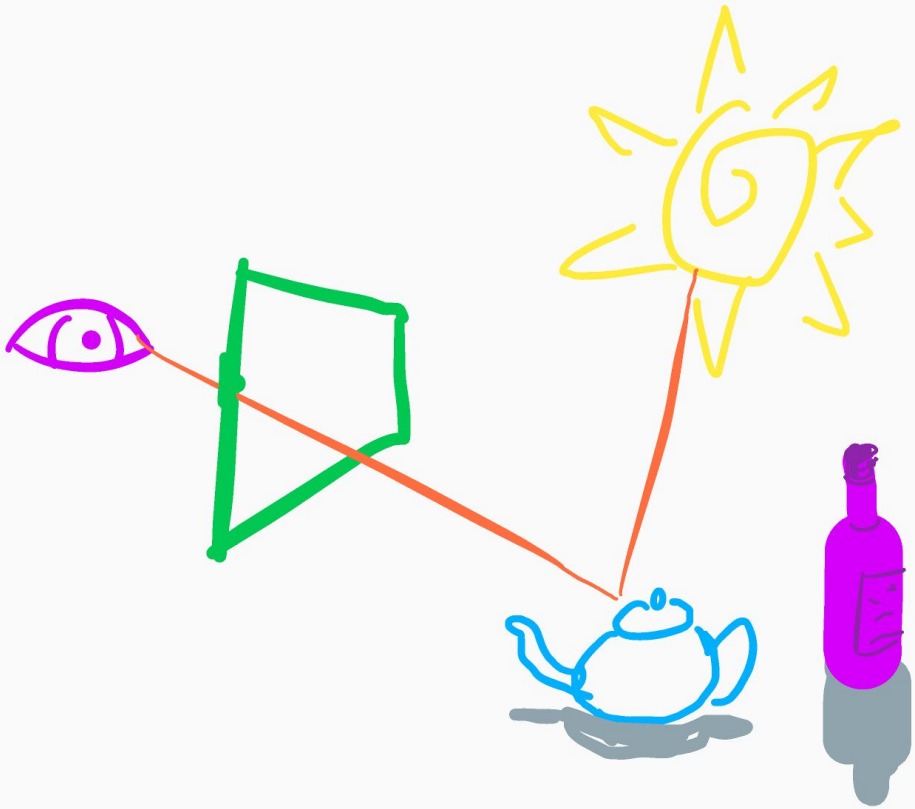
What is a Ray Tracer?

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

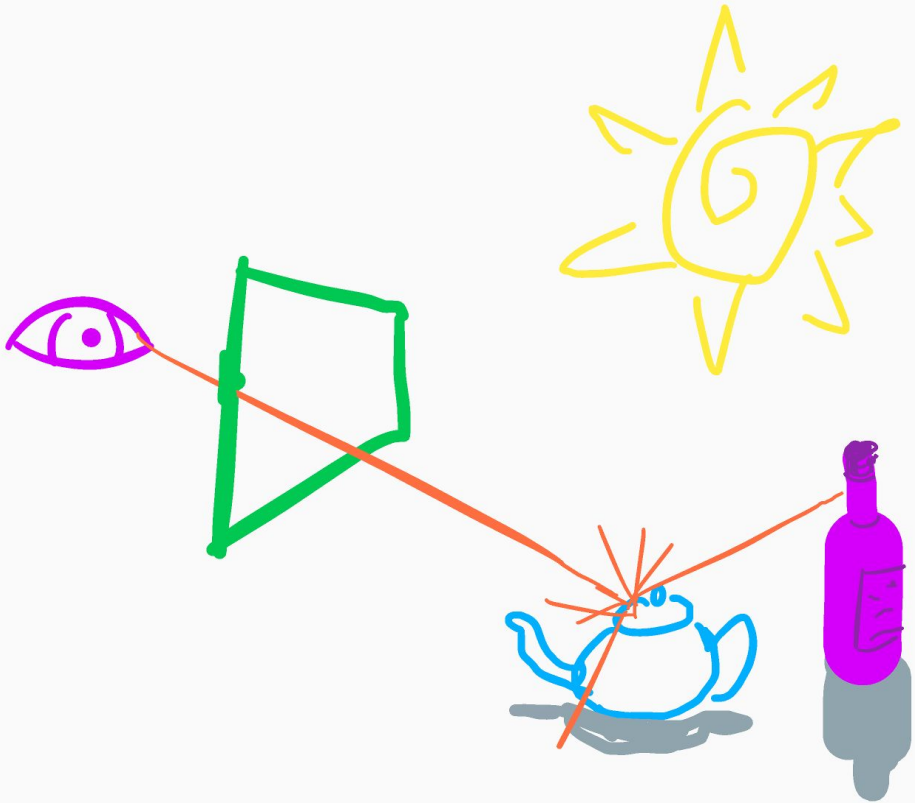


What is a Ray Tracer?

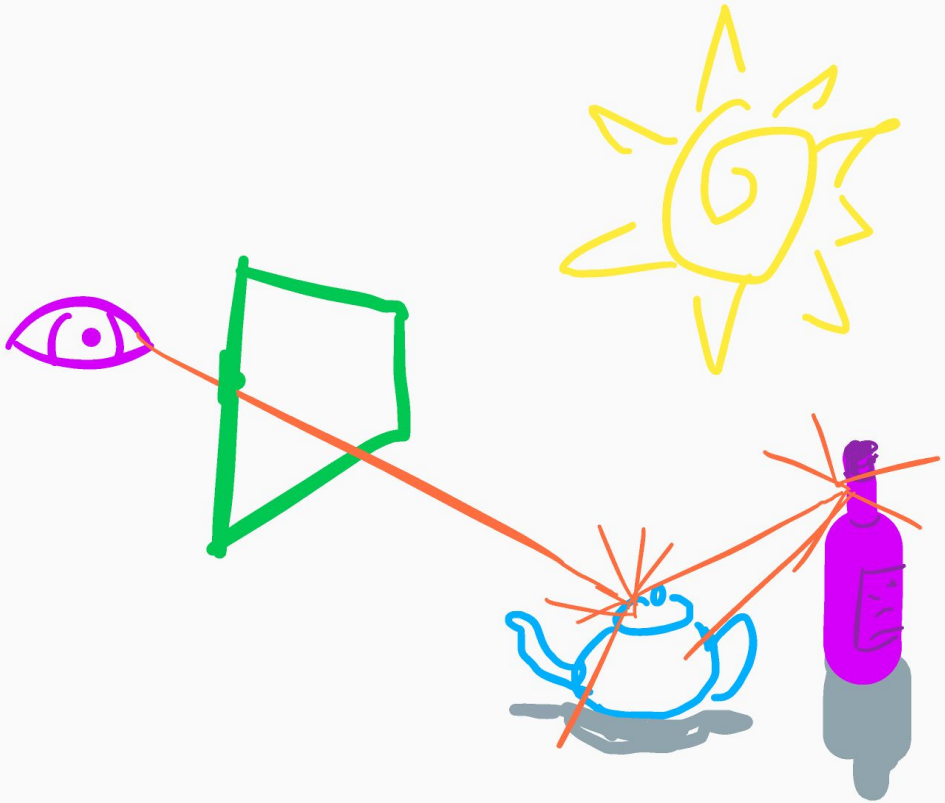
- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects



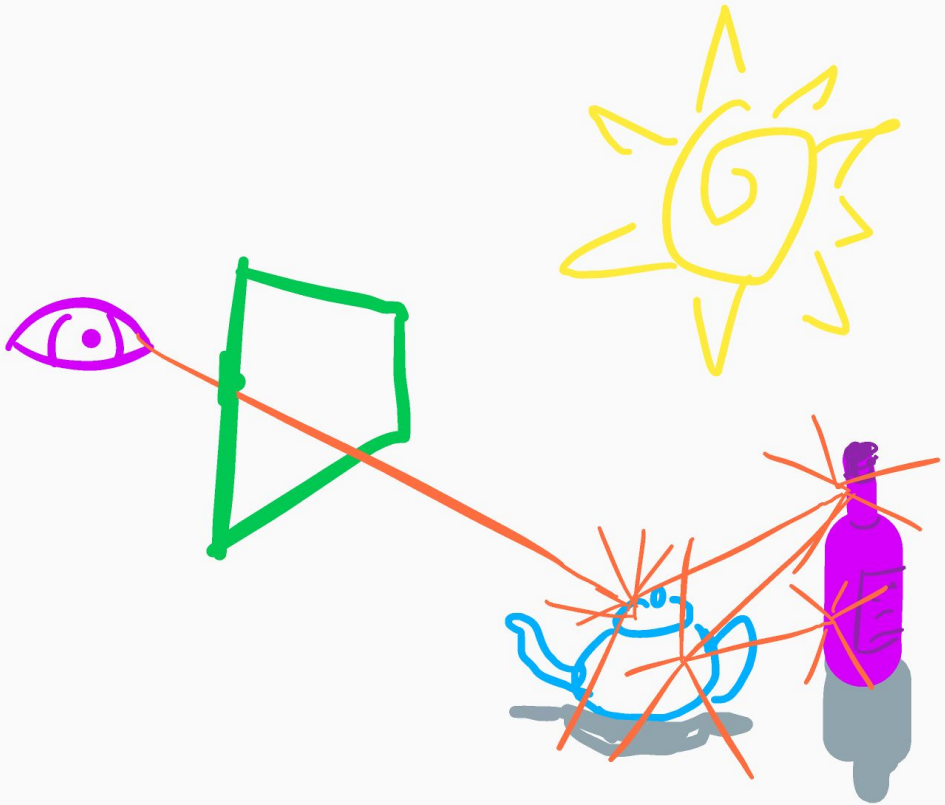
A Ray is cast



Additional rays are cast

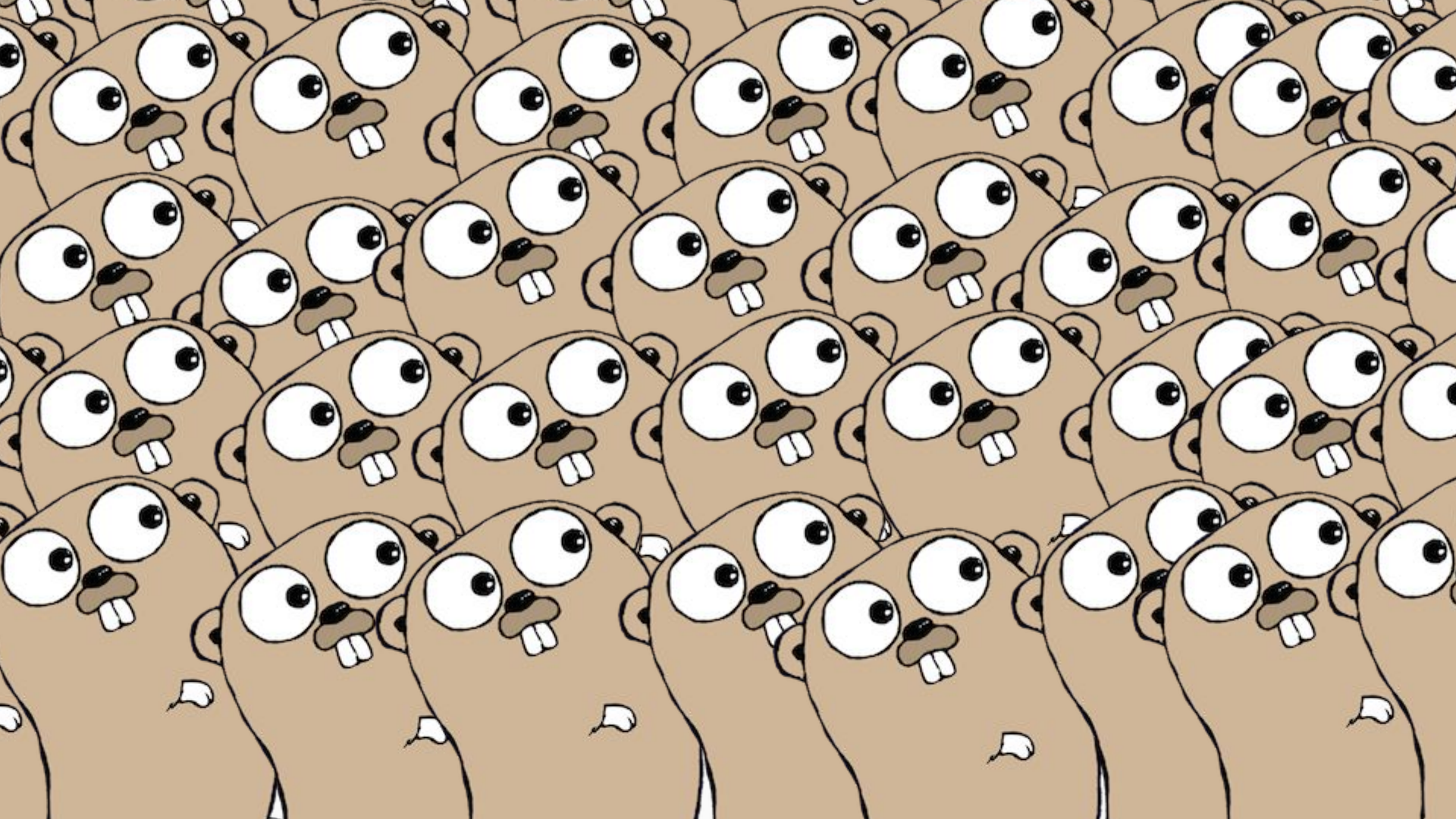


Further Additional rays are cast



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

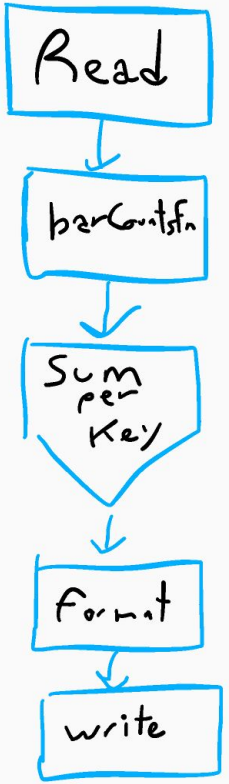


Apache Beam

beam

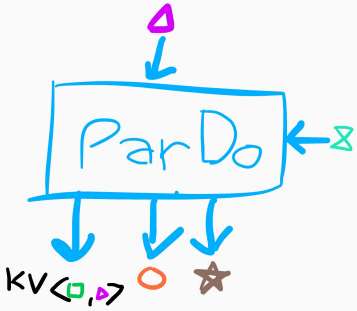
Apache Beam Go SDK

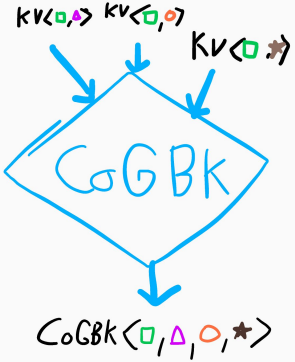
```
func main() {
  beam.Init()

  p, s := beam.NewPipelineWithRoot()
  foos := foo.Source(s, foo.DefaultConfig())
  barCounts := beam.ParDo(s, barCountsFn, foos)
  barTotals := stats.SumPerKey(s, barCounts)
  barOutput := beam.ParDo(s, &formatFn{}, barTotals)
  textio.Write(s, *output, barOutput)

  if err:= beamx.Run(context.Background(), p); err != nil {
    log.Exitf("pipeline failed: %v", err)
  }
}
```

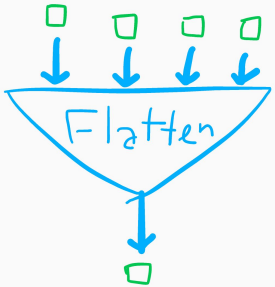


Model

```
beam.NewPipeline()

beam.ParDo(...)

beam.CoGroupByKey(...)

beam.Combine(...)

beam.Flatten(...)
```

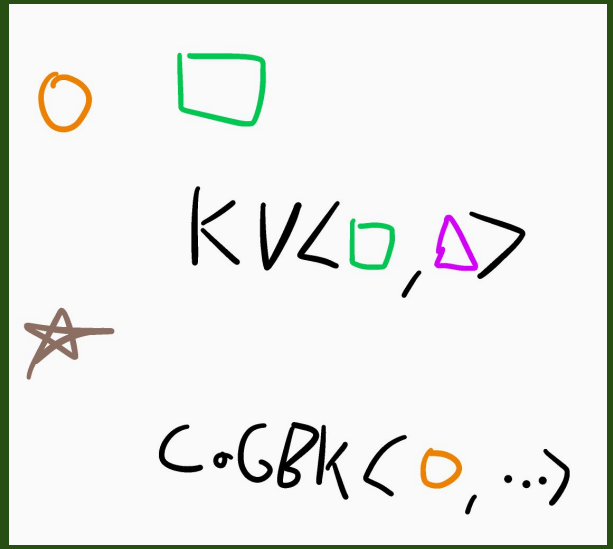






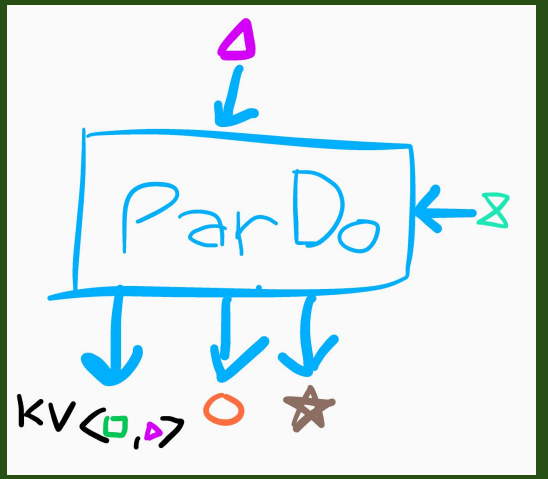
PCollections and Elements

```
var myPCol beam.PCollection

type Threshold float64

type Pixel struct {
  X, Y int
}

type Vec struct {
  X, Y, Z float64
}

type Ray struct {
  Px Pixel
  Id int
  Dir, Origin Vec
}
```



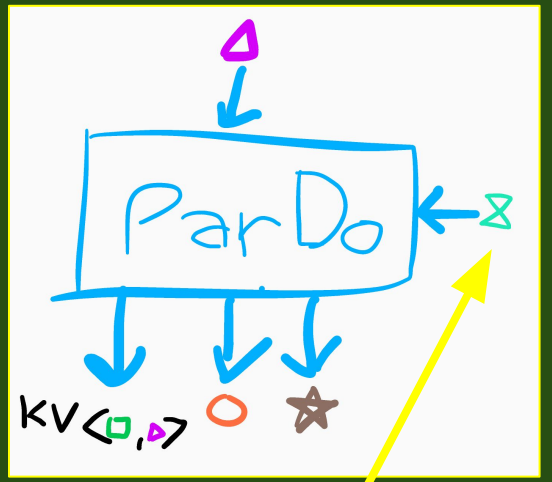
ParDo & DoFns

```
func getBarCountsFn(in Foo) (string, int) {
  return in.Key, len(in.B)
}

barCounts := beam.ParDo(s, getBarCountsFn, foos)
Titles := beam.ParDo(s, strings.Title, names)

type filterFn struct {
  Min int
}

func (fn *filterFn) ProcessElement(in Foo, emit func(string, int)) {
  l := len(in.B)
  if l < fn.Min {
   emit(in.Key, l)
  )
}

filterCounts := beam.ParDo(s, &filterFn{Min: 42}, foos)
```



Side Inputs

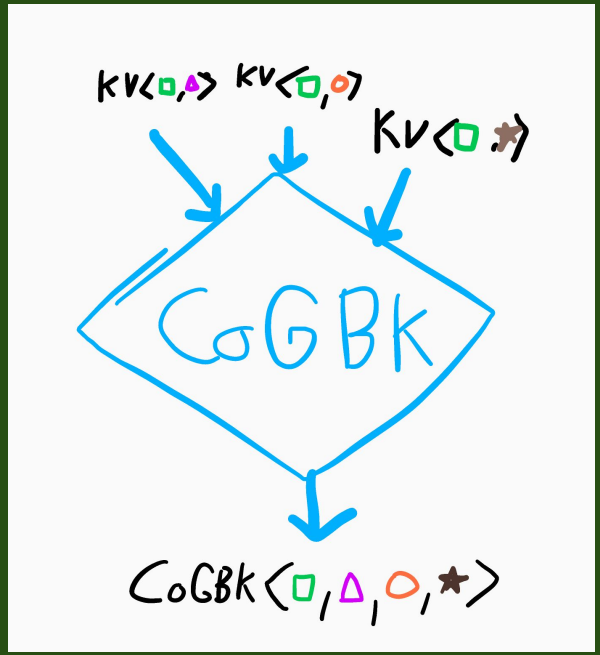
```
func exclude(v int, bounds func(*int) bool, high, mid, low func(string)) {
  ...
}

func exclude(v int, bounds []int, high, mid, low func(string)) {
  ...
}
```




```
highs, lows, mids := beam.ParDo3(s, exclude, importantValuesPCol, beam.SideInput{boundsPCol})
```

CoGroupByKey

```
func joinFooBar(k string, fooIter func(*Foo) bool, barIter func(*Bar) bool) (string, int) {
  ...
}


grouped := beam.CoGroupByKey(s, keyedFoos, keyedBars)
summed := beam.ParDo(s, joinFooBar, grouped)
```



Combines

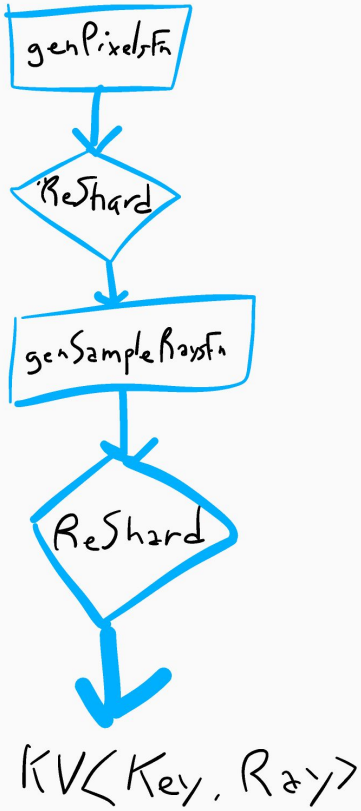
```
func sum(a,b int) int {
  return a + b
}

summed := beam.CombinePerKey(s, sum, myKeyedInts)


type cbnFn struct {
  ...
}

func (fn *cbnFn) AddInput(a Accum, i Foo) Accum { ... }

func (fn *cbnFn) MergeAccumulators(a Accum, b Accum)
  Accum { ... }

func (fn *cbnFn) ExtractOutput(a Accum) Bar { ... }

combinedBar := beam.Combine(s, &cbnFn{...}, myKeyedInts)
```

The Ray Tracing Algorithm

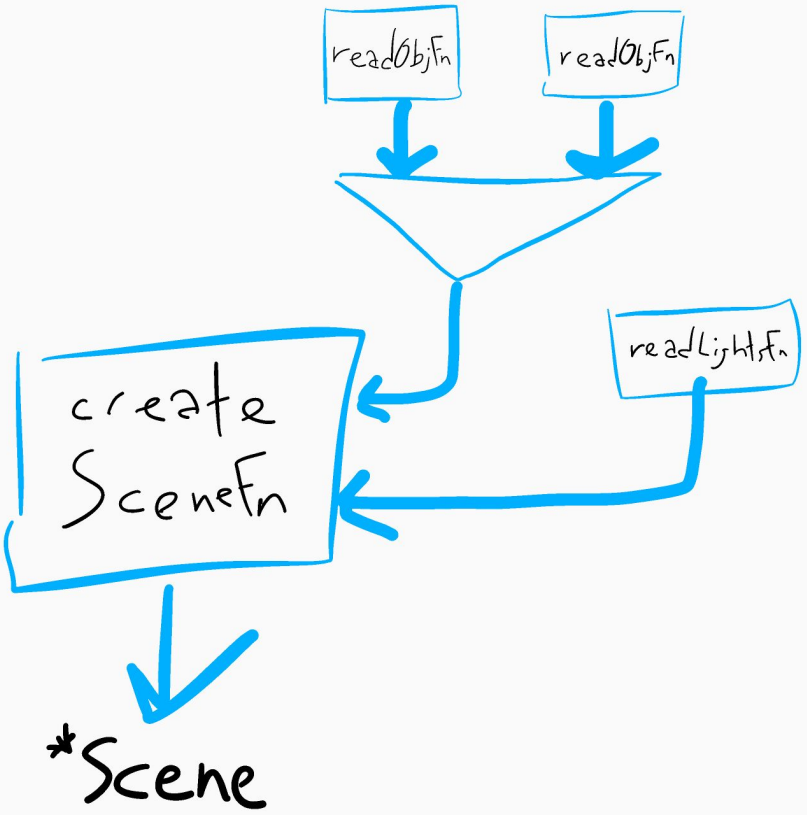
- Read Scene files & assemble Scene
- Set up camera
- For each pixel:
  - Trace sample rays
    - Intersect objects with ray
    - Trace “bounce” rays if needed
  - Accumulate color from rays
- Set pixel color
- Save Image

Saving an Image

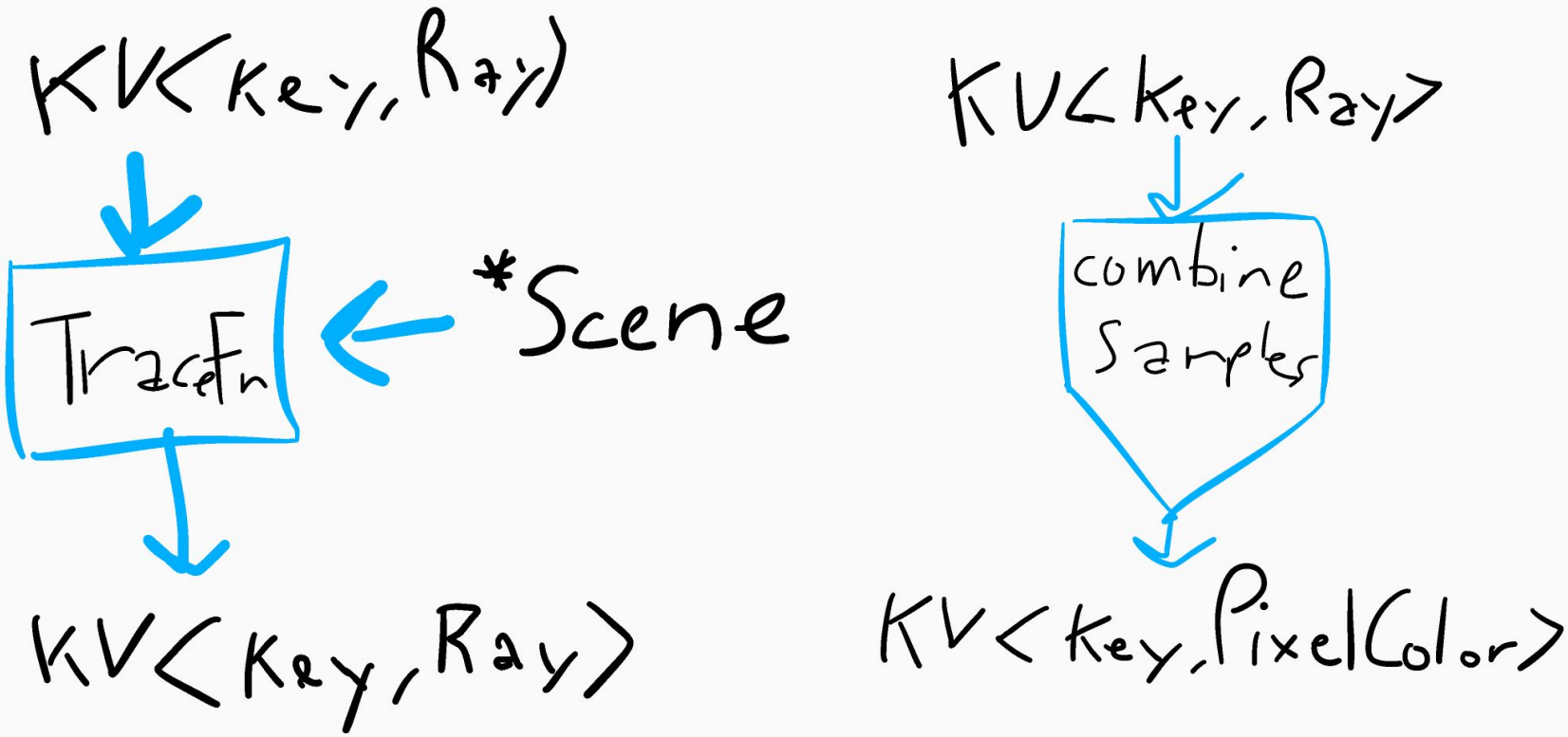
Generating Rays



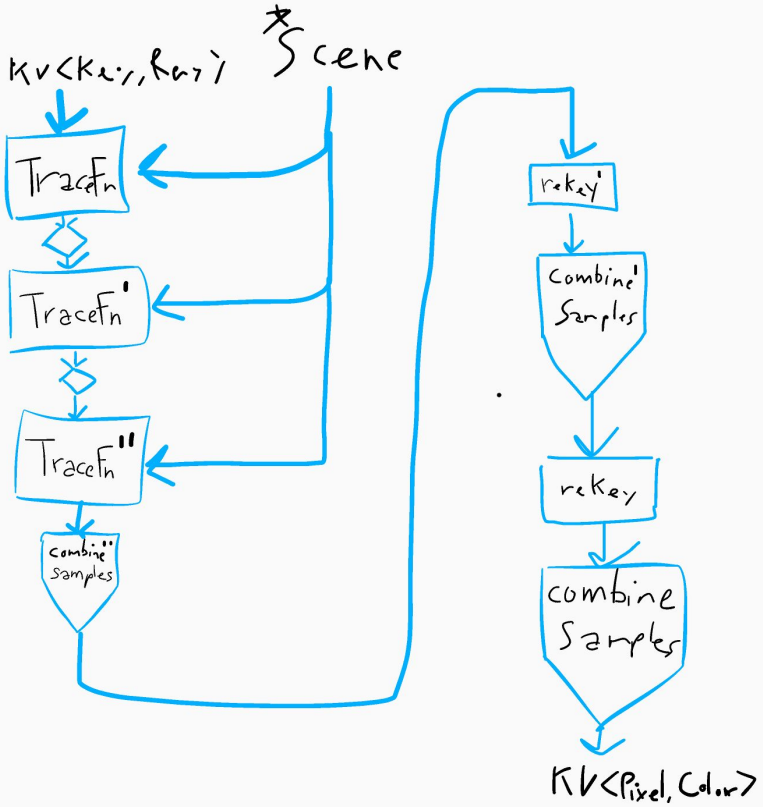
Generating the Scene

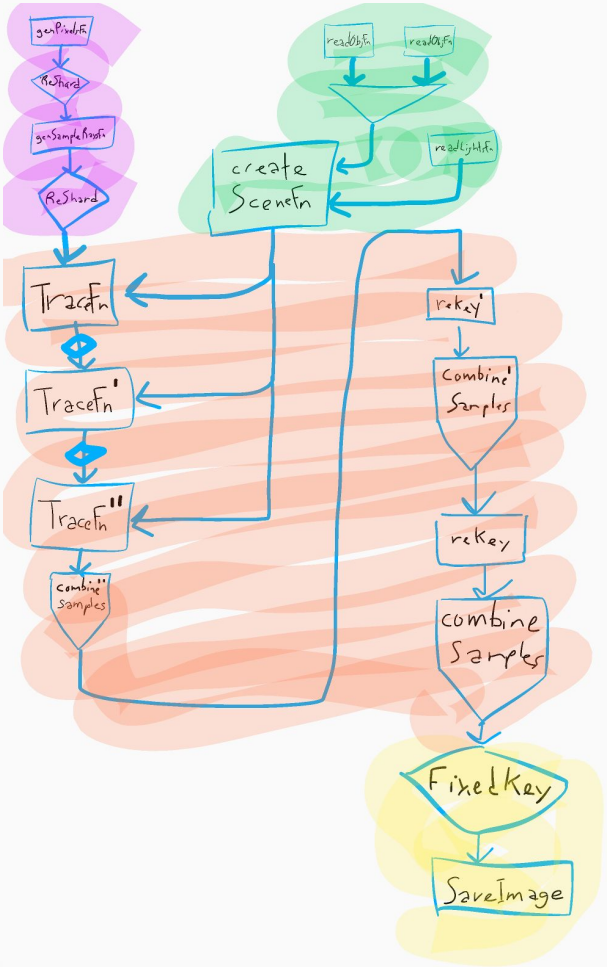


Tracing Rays



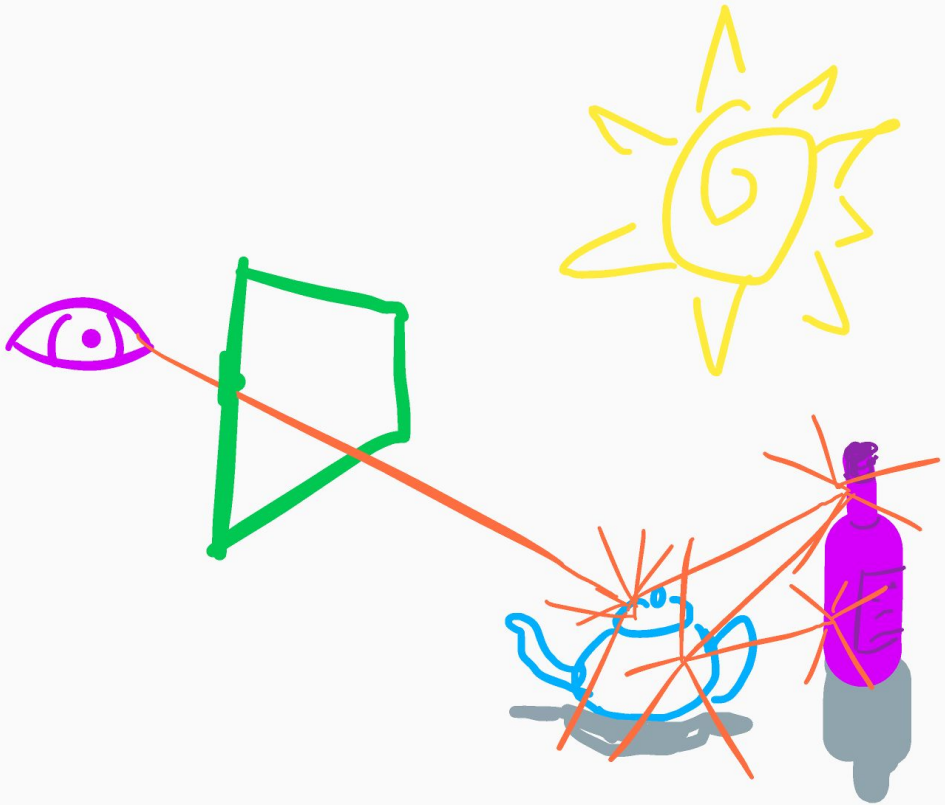
Tracing Rays



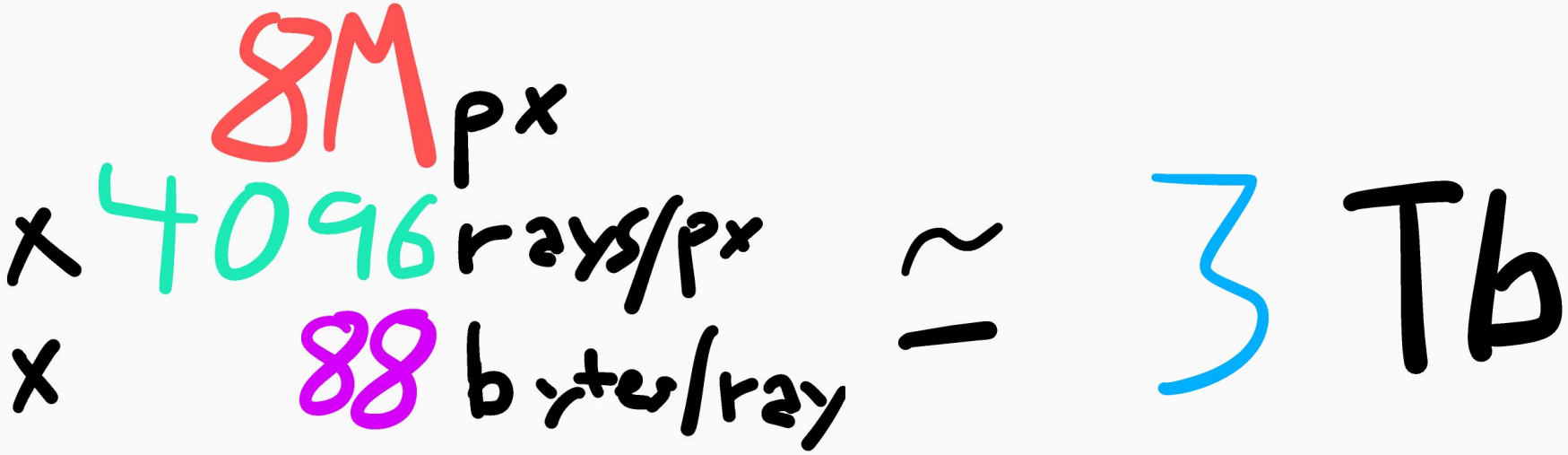


The Resulting Pipeline

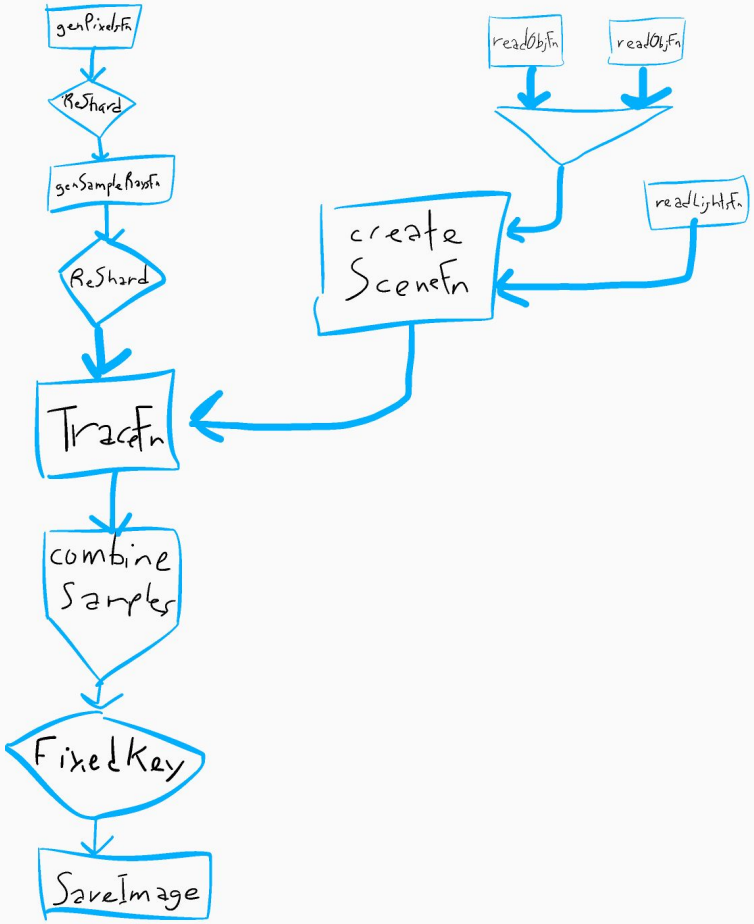
```
type Ray struct {
  Xp,Yp,Zp float64 // Position
  Xv,Yv,Zv float64 // Vector
  Rc,Gc,Bc float64 // Color

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



The Problem

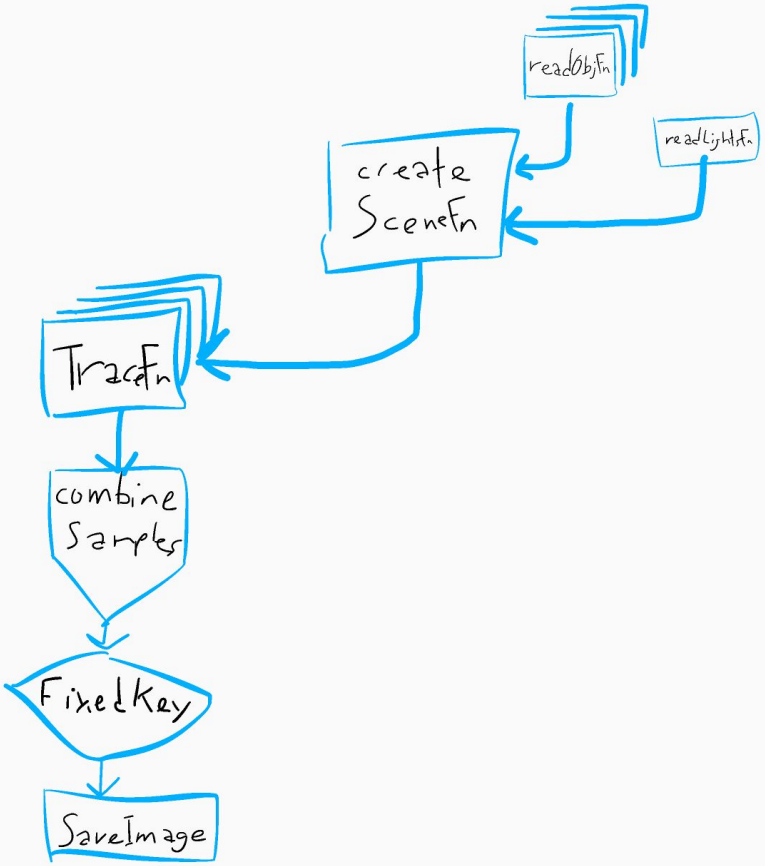

8M px
x 4096 rays/px
x 88 bytes/ray
≃ 3 Tb

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


genPixelFn
ReShard
genSampleRayFn
ReShard
TraceFn
combine Samples
FixedKey
SaveImage
readObjFn
readObjFn
readLightFn
create SceneFn

A Solution

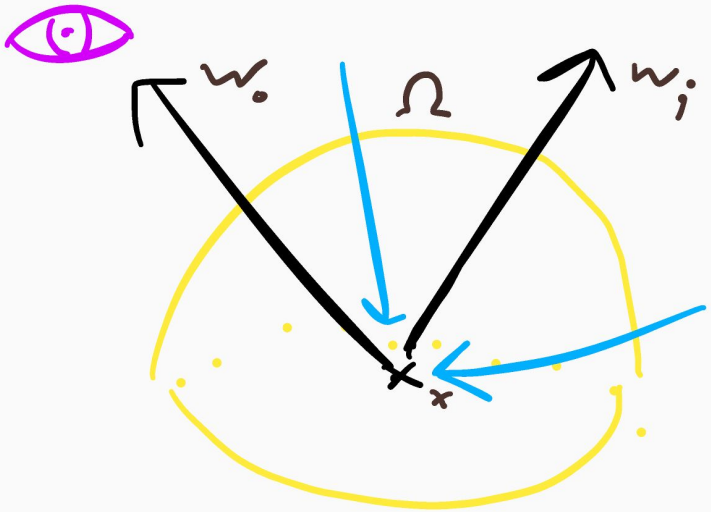
Future Work



A Better Solution

Danke!

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



Mathematics!