Oops, I Wrote a Portable Runner in Go

Robert Burke
@lostluck

BEAM SUMMIT Austin, 2022

Didn't Finish Writing

Oops, I ~~Wrote~~ a Portable Runner in Go

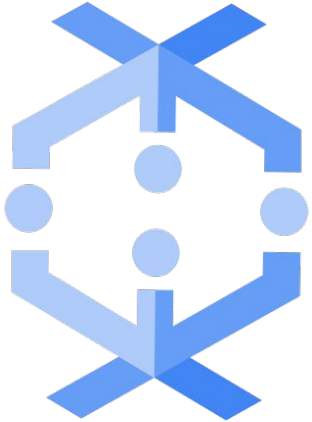
Robert Burke
@lostluck

BEAM SUMMIT Austin, 2022

How to Implement an SDK Feature?

1. Look at how other SDKs do it.
2. Implement.
3. Unit test.
4. Integration testing.

samza



Spark

Capability Matrix

What is being computed?

	Google Cloud Dataflow	Apache Flink	Apache Spa (RDD/DStream
ParDo	✓	✓	✓
GroupByKey	✓	✓	~
Flatten	✓	✓	✓
Combine	✓	✓	✓
Composite Transforms	~	~	~

https://beam.apache.org/documentation/runners/capability-matrix/

Map Side Inputs

Declare the side input

```
ProcessElement(..., lookup func(K) func(*V) bool,...){
...

vals := lookup(key)
var val V

for vals(&val) { ... }
```

Look up an iterator for a key

Iterate over associated values

Map Side Inputs

```
ProcessElement(..., lookup fu                    ...){
...

vals := look
var va

for v                 { ... }
```

Declare the sid

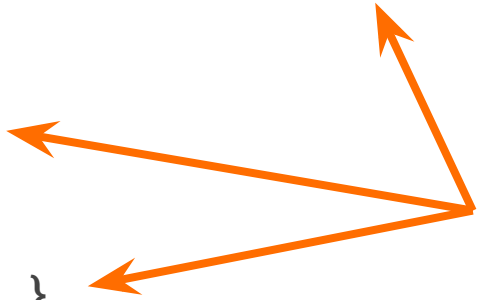
Look up an iterator for a key

Iterate over associated values

FEATURE IMPLEMENTED!

Cross Bundle Cache

```
ProcessElement(..., lookup func(K) func(*V) bool,...){
...

vals := lookup(key)
var val V

for vals(&val) { ... }
```



Caching Observed?

Cross Bundle Cache

```
import “github.com/apache/beam/sdks/v2/go/pkg/beam/util/harnessopts”

func main() {
    flags.Parse()

    harnessopts.SideInputCacheCapacity(*cacheKeyCount)

    beam.Init()
    . . .
}
```

Cross Bundle Cache

```
ProcessElement(..., lookup func(K) func(*V) bool,...){
...

timestamps.Update(ctx,time.Now().UnixNano())
vals := lookup(key)
var val V

for vals(&val) { ... }
timestamps.Update(ctx,time.Now().UnixNano())
```

Cross Bundle Cache

```
ProcessElement(..., lookup fu                    ...){
...

timestamps.U                    ().UnixNano())
vals :=
v

fo      s(&val) { ... }
timestamps.Update(ctx,time.Now().UnixNano())
```

FEATURE IMPLEMENTED ...?

Caching Observed?

Capability Matrix

What is being computed?

	Google Cloud Dataflow	Apache Flink	Apache Spa (RDD/DStream
ParDo	✓	✓	✓
GroupByKey	✓	✓	~
Flatten	✓	✓	✓
Combine	✓	✓	✓
Composite Transforms	~	~	~

https://beam.apache.org/documentation/runners/capability-matrix/

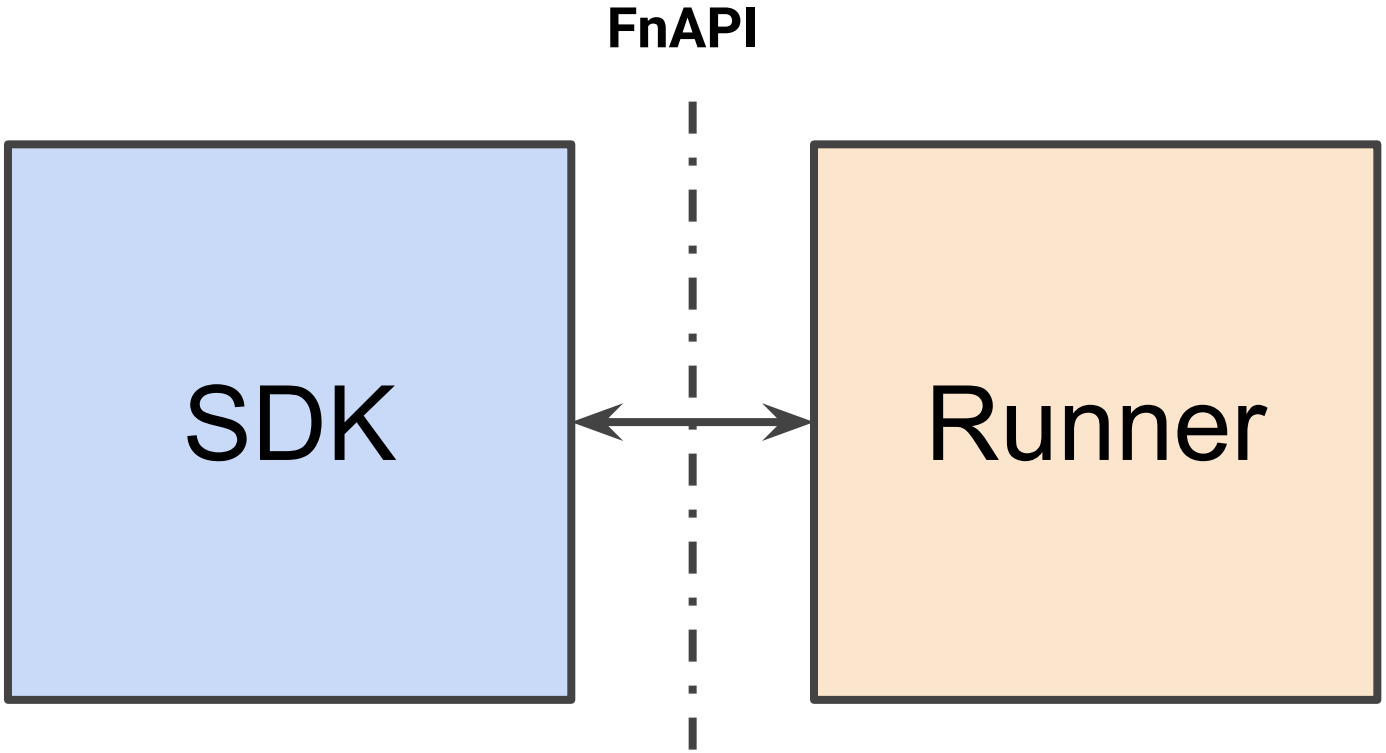
Cross Bundle Cache

```
ProcessElement(..., lookup fu                    ,...){
...

timestamps.U                    ().UnixNano())
vals :=
v

fo      s(&val) { ... }
timestamps.Update(ctx,time.Now().UnixNano())
```

FEATURE IMPLEMENTED ...?

Caching Observed?

Portable Worker



BEAM SUMMIT

Beam has a Testing Problem

My Runner's Goals

- SDK Testing focused
- Low Bar: Supercede Go Direct Runner
- Local & Single Machine
- In Memory
- Per-Pipeline Configurable
- High Bar: Run Java and Python pipelines

- Contribute it to the repo

Names are hard

Name possibilities

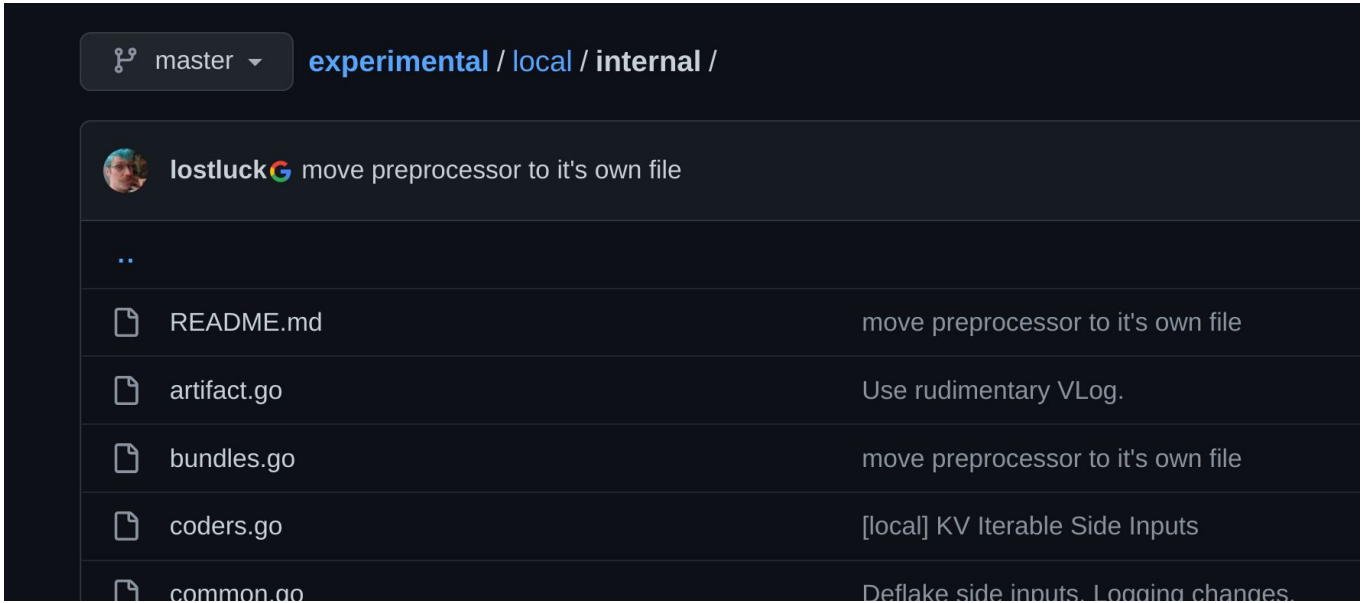
local: emphasises it's not a distributed computation, but limits growth in that direction.
fake: tongue-in-cheek reference to test fakes which represent using a full implementation but may take some shortcuts that aren't appropriate in production
beam: It is what it is, but confusing to all future runners.
model: Clear, unambiguous that it represents implementation of the beam model. Might be confusing when discussion the model though (does model implement foo…)
unit: Emphasises its suitability for unit testing, in unoptimized mode.
portable: Might get conflated with the Python portable runner.
universal: Might get confused with the decommissioned Java universal runner.
teach: emphasises the customizability and pedagogical benefits of the runner.
comp: emphasises the ability to validate different components.
collab: emphasises how the runner collaborates with SDKs to make testing things easier. But easily mistaken for Google Colab.
sdk: improved reference point runner -> each sdk has a Direct runner, so the term is confusing. Calling the the sdk runner or the gosdk runner avoids the confusion, and makes it rather clear about the implementation of it is. Bad point: then the packaging can get confusion: is the problem in the Go sdk or the gosdk?

<some not generic name>: care needs to be taken against being confused with other products and features in the data processing space.

handlebar: it gives control to pipelines for how it will execute (might be confused with the related moustache templates extension)
prism: It splits the Beam into its component parts, makes it visible.
lens: It focuses Beam into a pinpoint for consistent testing.

What I've got so far

https://github.com/lostluck/experimental/tree/master/local/internal



Loopback

https://github.com/lostluck/experimental/tree/master/local/internal

```
42  func Execute(ctx context.Context, p *beam.Pipeline) (beam.PipelineResult, error) {
43      if *jobopts.Endpoint == "" {
44          // One hasn't been selected, so lets start one up and set the address.
45          // Conveniently, this means that if multiple pipelines are executed against
46          // the local runner, they will all use the same server.
47          s := internal.NewServer(0)
48          *jobopts.Endpoint = s.Endpoint()
49          go s.Serve()
50      }
51      if !jobopts.IsLoopback() {
52          //  log.Infof(ctx, "Environment type: %v, forcing loopback, as the local runner c
53          *jobopts.EnvironmentType = "loopback"
54      }
55      return universal.Execute(ctx, p)
56  }
```

Single PTransform Execution

https://github.com/lostluck/experimental/tree/master/local/internal

```
115
116     // Goroutine for executing bundles on the worker.
117     go func() {
118         // Send nil to start, Impulses won't require parental translation.
119         processed <- nil
120         for b := range toProcess {
121             b.ProcessOn(wk) // Blocks until finished.
122             // Metrics?
123             j.metrics.contributeMetrics(<-b.Resp)
124             // Send back for dependency handling afterwards.
125             processed <- b
126         }
127         close(processed)
128     }()
129
130     // prevs is a map from current transform ID to a map from local ids to b
131     prevs := map[string]map[string]*bundle{}
132
133     for _, tid := range topo {
134         // Block until the previous bundle is done.
135         <-processed
```

```
    coders := map[string]*pipepb.Coder{}
    transforms := map[string]*pipepb.PTransform{
        tid: t, // The Transform to Execute!
    }

    pardo := &pipepb.ParDoPayload{}
```

```
246                 reconcileCoders(coders, pipeline.GetComp
247
248                 desc := &fnpb.ProcessBundleDescriptor{
249                     Id:                  bundID,
250                     Transforms:          transforms,
251                     WindowingStrategies: pipeline.GetCom
252                     Pcollections:        pipeline.GetCom
253                     Coders:              coders,
254                     StateApiServiceDescriptor: &pipepb.A
255                         Url: wk.Endpoint(),
```

GBKs

https://github.com/lostluck/experimental/tree/master/local/internal

```
457  func gbkBytes(ws *pipepb.WindowingStrategy, wc exec.WindowDecoder, kc, vc *pipepb.Coder,
458      var outputTime func(typex.Window, mtime.Time) mtime.Time
459      switch ws.GetOutputTime() {
460      case pipepb.OutputTime_END_OF_WINDOW:
461          outputTime = func(w typex.Window, et mtime.Time) mtime.Time {
462              return w.MaxTimestamp()
463          }
464      default:
465          logger.Fatalf("unsupported OutputTime behavior: %v", ws.GetOutputTime())
466      }
467
468      type keyTime struct {
469          key    []byte
470          w      typex.Window
471          time   mtime.Time
472          values [][]byte
473      }
474      // Map windows to a map of keys to a map of keys to time.
475      // We ultimately emit the window, the key, the time, and the iterable of elements,
476      // all contained in the final value.
477      windows := map[typex.Window]map[string]keyTime{}
478
479      kd := pullDecoder(kc, coders)
480      vd := pullDecoder(vc, coders)
481
482      // Right, need to get the key coder, and the element coder.
```

What I've got so far

https://github.com/lostluck/experimental/tree/master/local/internal

- One PTransform at at time.
- ParDos
- GBKs
- Unlifted Combines
- Metrics

Example: CombineFns

BEAM SUMMIT Austin, 2022

CombineFn

Input **I** -> Accumulator **A** -> Output **O**

- CreateAccumulator **() -> A**
- Add Input **(I, A) -> A**
- Merge Accumulators **(A, A) -> A**
- Extract Output **(A) -> O**

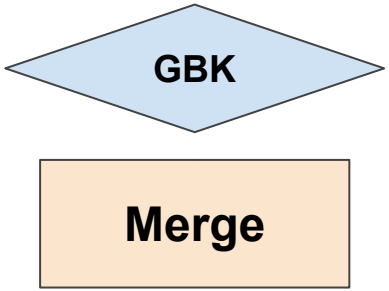
CombineFn

Input **I** == Accumulator **A** == Output **O**

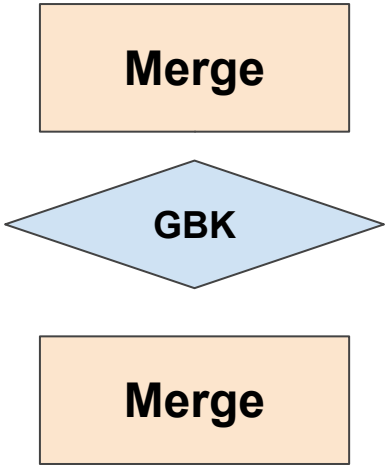
- CreateAccumulator **() -> A**
- Add Input **(I, A) -> A**
- Merge Accumulators **(A, A) -> A**
- Extract Output **(A) -> O**

Combiner Lifting

Unlifted



Lifted



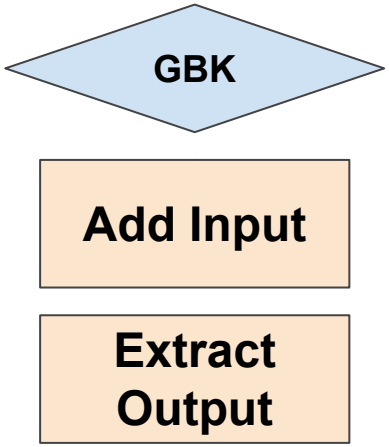
CombineFn

Input **I** != Accumulator **A** == Output **O**

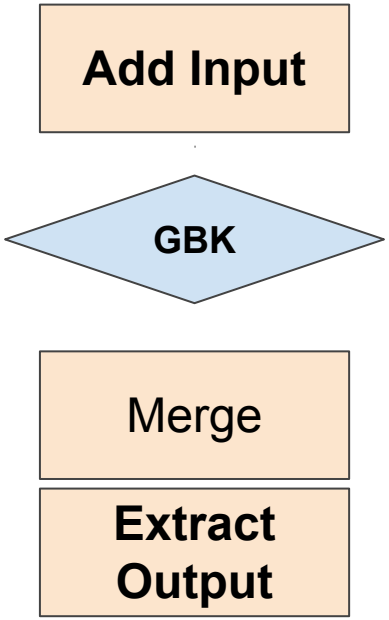
- CreateAccumulator **() -> A**
- Add Input **(I, A) -> A**
- Merge Accumulators **(A, A) -> A**
- Extract Output **(A) -> O**

Combiner Lifting

Unlifted



Lifted



Combiner Lifting

- Used in Batch execution
  - Bigger bundles, more benefit
- Not used in Streaming execution
  - Small bundles, negligible benefit

Example: SplittableDoFns

BEAM SUMMIT Austin, 2022

Splittable DoFns

- CreateInitialRestriction
- CreateTracker
- SplitRestriction
- RestrictionSize
- …and more

My Runner's "Secret" Goals

Production Ready?

- Modular
- Production features
  - Optimized Execution Graph
  - Disk Spillover
- Automation of Testing SDK Features

Oops, I Wrote a Portable Runner in Go

Robert Burke
@lostluck

BEAM SUMMIT Austin, 2022

Related Talks

Writing a native Go streaming pipeline

Tuesday 16:15-16:40 CDT, Room 203
with Danny McCormick and Jack McCluskey
https://2022.beamsummit.org/sessions/native-go-pipeline/

Direct Runners are Bad

BEAM SUMMIT

Direct Runners are Bad