PaSh: Light-Touch Data-Parallel Shell Processing

Nikos Vasilakis*
MIT

Konstantinos Kallas*
University of Pennsylvania

Konstantinos Mamouras
Rice University

Achilles Benetopoulos
(Unaffiliated)

Lazar Cvetković
University of Belgrade

pash-discuss@googlegroups.com
github.com/andromeda/pash

* equal contribution

Shell Scripts are Everywhere

Default/scriptable system interface even in the lightest containers Kubernetes, Docker

```
# Check all possible clusters, as your .KUBECONFIG may have multiple contexts:
kubectl config view -o jsonpath='{"Cluster name\tServer\n"}{range .clusters[*]}{

# Select name of cluster you want to interact with from above output:
export CLUSTER_NAME="gke_ps-dev-201405_us-east1_acaternberg"
#export SERVICE_ACCOUNT=cjoc
#export SERVICE_ACCOUNT=cloudbees-core-nginx-ingress
export SERVICE_ACCOUNT=default

# Point to the API server referring the cluster name
APISERVER=$(kubectl config view -o jsonpath="{.clusters[?(@.name==\"$CLUSTER NAM
```

Universal composition environment
Commands (programs) can be written in C, C++, Rust, JS, Python, Ruby, Haskell...

```
echo "Building parser..."
cd compiler/parser
echo "|-- making libdash..."
make libdash &> $LOG_DIR/make_libdash.log
```

Succinct data processing: download/extraction/ preprocessing/querying

```
seq $FROM $TO | sed "s;^;$IN;" | sed 's;$;/;' | xargs -r -n 1 $fetch | grep gz |
tr -s ' \n' | cut -d ' ' -f9 | sed 's;^\(.*\)\(20[0-9][0-9]\).gz;\2/\1\2\.gz;' |
sed "s;^;$IN;" | xargs -n1 curl -s | gunzip |                    # end-preprocssing
cut -c 89-92 | grep -v 999 | sort -rn | head -n1                 # actual processing
```

A Classic Shell Script

Bentley: A word-counting challenge

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair.

```
10 was
10 the
10 of
10 it
 2 times
```

Knuth: 100s of lines of literate WEB

McIlroy: Unix one-liner

```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```

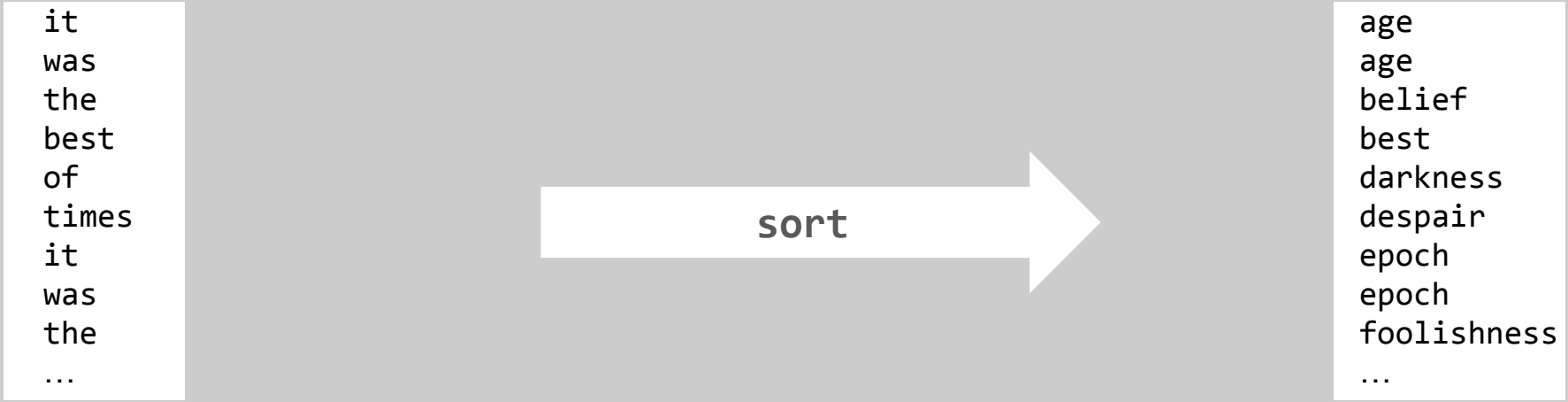
```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```

```
It was the best of times, it was the
worst of times, it was the age of
wisdom, it was the age of
foolishness, it was the epoch of
belief, it was the epoch of
incredulity, it was the season of
Light, it was the season of
Darkness, it was the spring of
hope, it was the winter of despair.
```


```
It
was
the
best
of
times
it
was
the
…
```

```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```


```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```




```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```

```
age
age
belief
best
darkness
despair
epoch
epoch
foolishness
…
```


```
 2 age
 1 belief
 1 best
 1 darkness
 1 despair
 2 epoch
 1 foolishness
 1 hope
10 it
 …
```

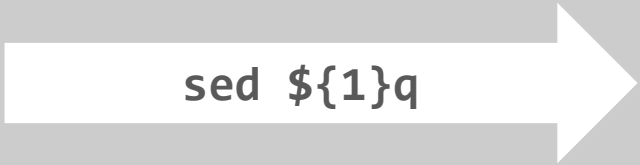
```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```

```
 2 age
 1 belief
 1 best
 1 darkness
 1 despair
 2 epoch
 1 foolishness
 1 hope
 1 incredulity
10 it
```


```
10 was
10 the
10 of
10 it
 2 times
 2 season
 2 epoch
 2 age
 1 worst
 1 wisdom
```

```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```

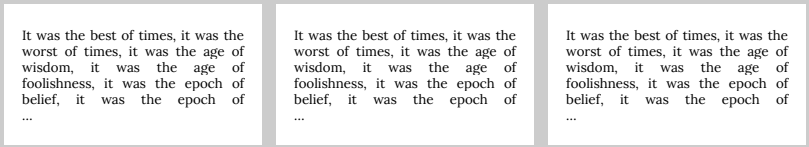
```
10 was
10 the
10 of
10 it
 2 times
 2 season
 2 epoch
 2 age
 1 worst
 1 wisdom
 …
```

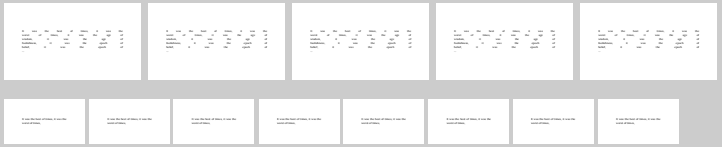



```
10 was
10 the
10 of
10 it
 2 times
```

```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```

```
It was the best of times, it was the
worst of times, it was the age of
wisdom, it was the age of
foolishness, it was the epoch of
belief, it was the epoch of
...
```





```
10 was
10 the
10 of
10 it
 2 times
```

How to parallelize?

```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```

Shell scripts are mostly sequential

Their parallelization requires considerable effort:

- Command-specific flags (e.g., `sort -p`, `make -jN`)
- Mostly-manual, restricted parallelization tools (e.g., GNU `parallel`)
- Full rewrites in parallel frameworks (e.g., MapReduce)

```java
import java.io.*;
import java.util.*;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;

public class top_10_Movies_Mapper extends Mapper<Object,
 Text, Text, LongWritable> {

   private TreeMap<Long, String> tmap;

   @Override
   public void setup(Context context) throws IOException,
   InterruptedException
   {
     tmap = new TreeMap<Long, String>();
   }

   @Override
   public void map(Object key, Text value,
     Context context) throws IOException,
   InterruptedException
   {

     // no_of_views  (tab seperated)
     // we split the input data
     String[] tokens = value.toString().split("\t");

     String movie_name = tokens[0];
     long no_of_views = Long.parseLong(tokens[1]);

     tmap.put(no_of_views, movie_name);

     if (tmap.size() > 10)
     {
       tmap.remove(tmap.firstKey());
     }
   }


   @Override
   public void cleanup(Context context) throws IOException,
   InterruptedException
   {
     for (Map.Entry<Long, String> entry : tmap.entrySet())
     {
       long count = entry.getKey();
       String name = entry.getValue();
       context.write(new Text(name), new LongWritable(count));
     }
   }
}
```

```java
import java.io.IOException;
import java.util.Map;
import java.util.TreeMap;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class top_10_Movies_Reducer extends Reducer<Text,
 LongWritable, LongWritable, Text> {
   private TreeMap<Long, String> tmap2;

   @Override
   public void setup(Context context) throws IOException,
   InterruptedException
   {
     tmap2 = new TreeMap<Long, String>();
   }

   @Override
   public void reduce(Text key, Iterable<LongWritable> values,
     Context context) throws IOException, InterruptedException
   {

     String name = key.toString();
     long count = 0;

     for (LongWritable val : values) {
       count = val.get();
     }

     tmap2.put(count, name);

     if (tmap2.size() > 10) {
       tmap2.remove(tmap2.firstKey());
     }
   }

   @Override
   public void cleanup(Context context) throws IOException,
   InterruptedException {

     for (Map.Entry<Long, String> entry : tmap2.entrySet()) {
       long count = entry.getKey();
       String name = entry.getValue();
       context.write(new LongWritable(count), new Text(name));
     }
   }
}
```

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class Driver {
 public static void main(String[] args) throws Exception
 {
   Configuration conf = new Configuration();
   String[] otherArgs = new GenericOptionsParser(conf,
     args).getRemainingArgs();

   if (otherArgs.length < 2)
   {
     System.err.println("Error: please provide two paths");
     System.exit(2);
   }

   Job job = Job.getInstance(conf, "top 10");
   job.setJarByClass(Driver.class);

   job.setMapperClass(top_10_Movies_Mapper.class);
   job.setReducerClass(top_10_Movies_Reducer.class);

   job.setMapOutputKeyClass(Text.class);
   job.setMapOutputValueClass(LongWritable.class);

   job.setOutputKeyClass(LongWritable.class);
   job.setOutputValueClass(Text.class);

   FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
   FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

   System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Big-Data Version of McIlroy's Pipeline

150-line Hadoop Program

Mostly sequential by default — how to parallelize?

Parallelization requires considerable effort:

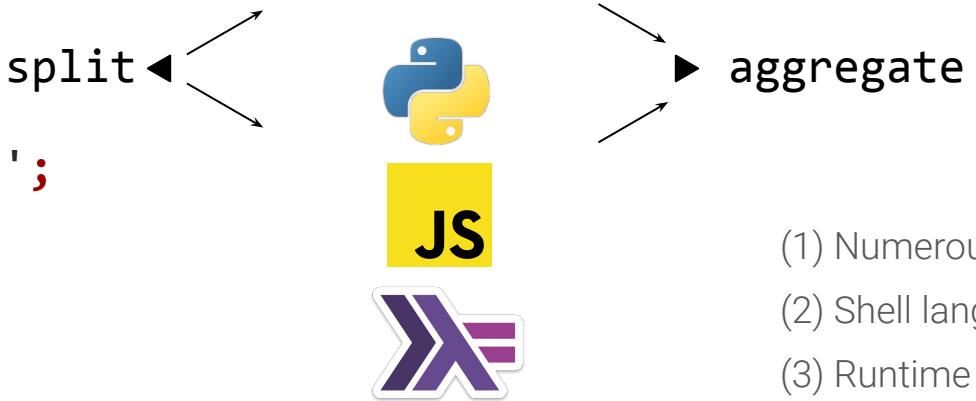
- Command-specific flags (e.g., `sort -p`, `make -jN`)
- Mostly-manual, restricted parallelization tools (e.g., GNU `parallel`)
- Full rewrites in parallel frameworks (e.g., MapReduce)

Challenges of Automating Shell-Script Parallelization

```
for directory in /project/gutenberg/*/; do
 ls $directory | grep 'txt' | wc -l > index.txt
done

cat f1 f2 |
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q

echo 'Done';
```

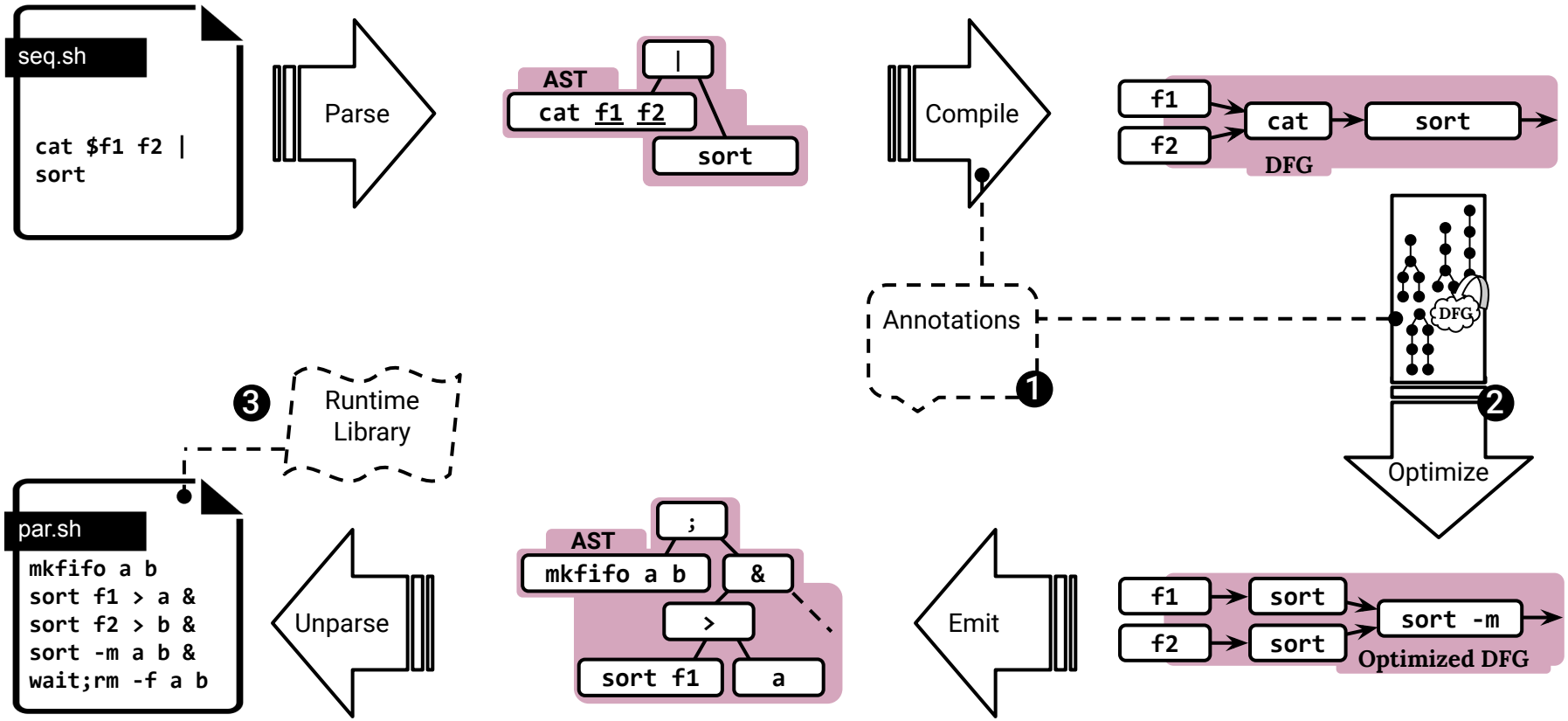


(1) Numerous and opaque Unix commands

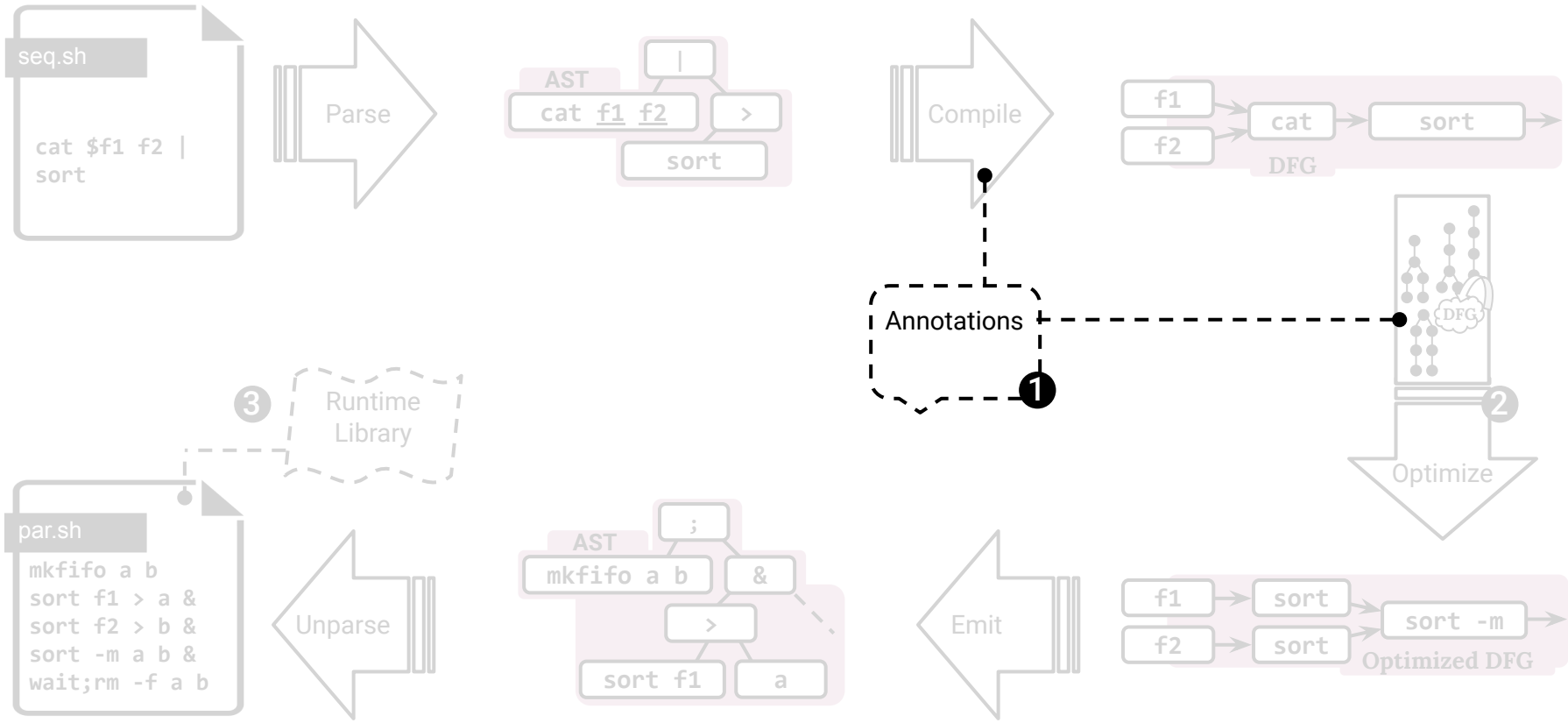
(2) Shell language enforced dependencies

(3) Runtime support for Unix parallelization

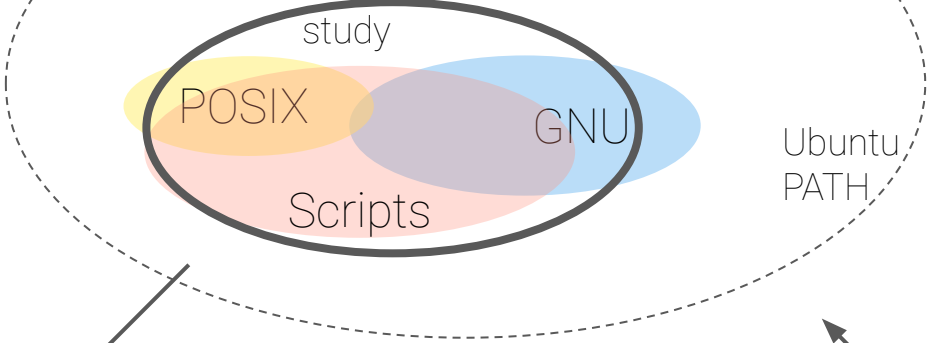
PaSh Overview



PaSh Overview



1. Unix Parallelizability Study & Annotations



POSIX
GNU

Parallelizability properties:

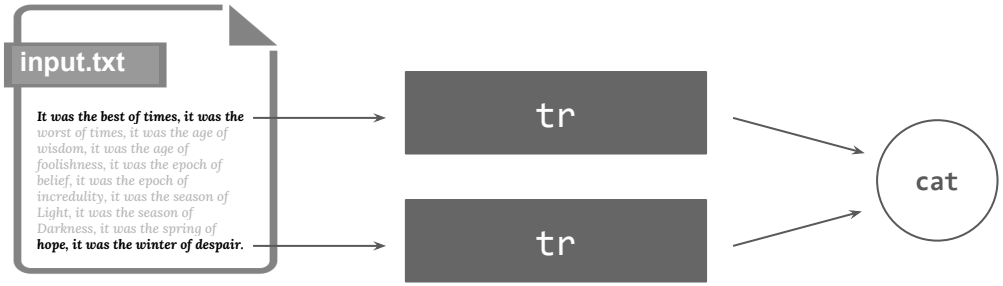
- 4 broad classes
- Flags and options
- Input consumption

Parallelizability DSL:

`(cmd, flg, [in])` → DFG node

4 command parallelizability classes

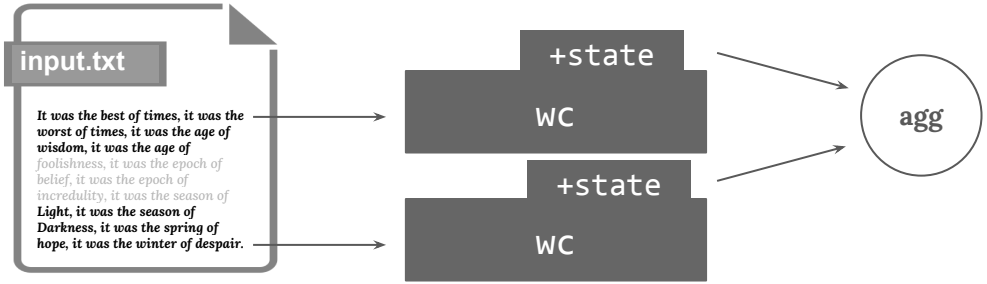
12.7% stateless



4 command parallelizability classes

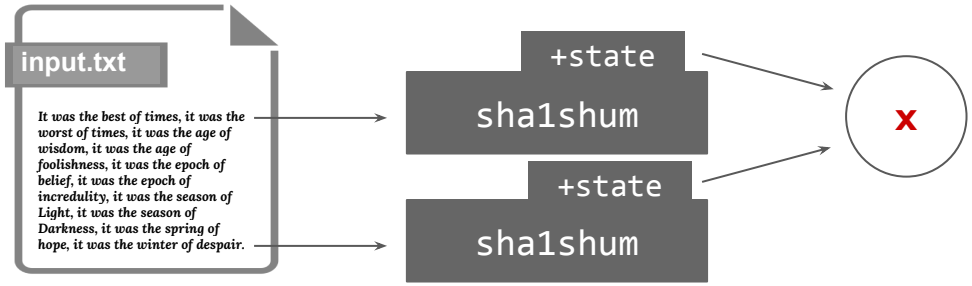
12.7% stateless

8.7% parallelizable pure



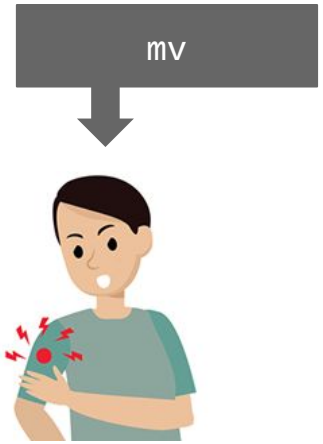
4 command parallelizability classes

12.7% stateless
8.7% parallelizable pure
8.2% non-parallelizable pure

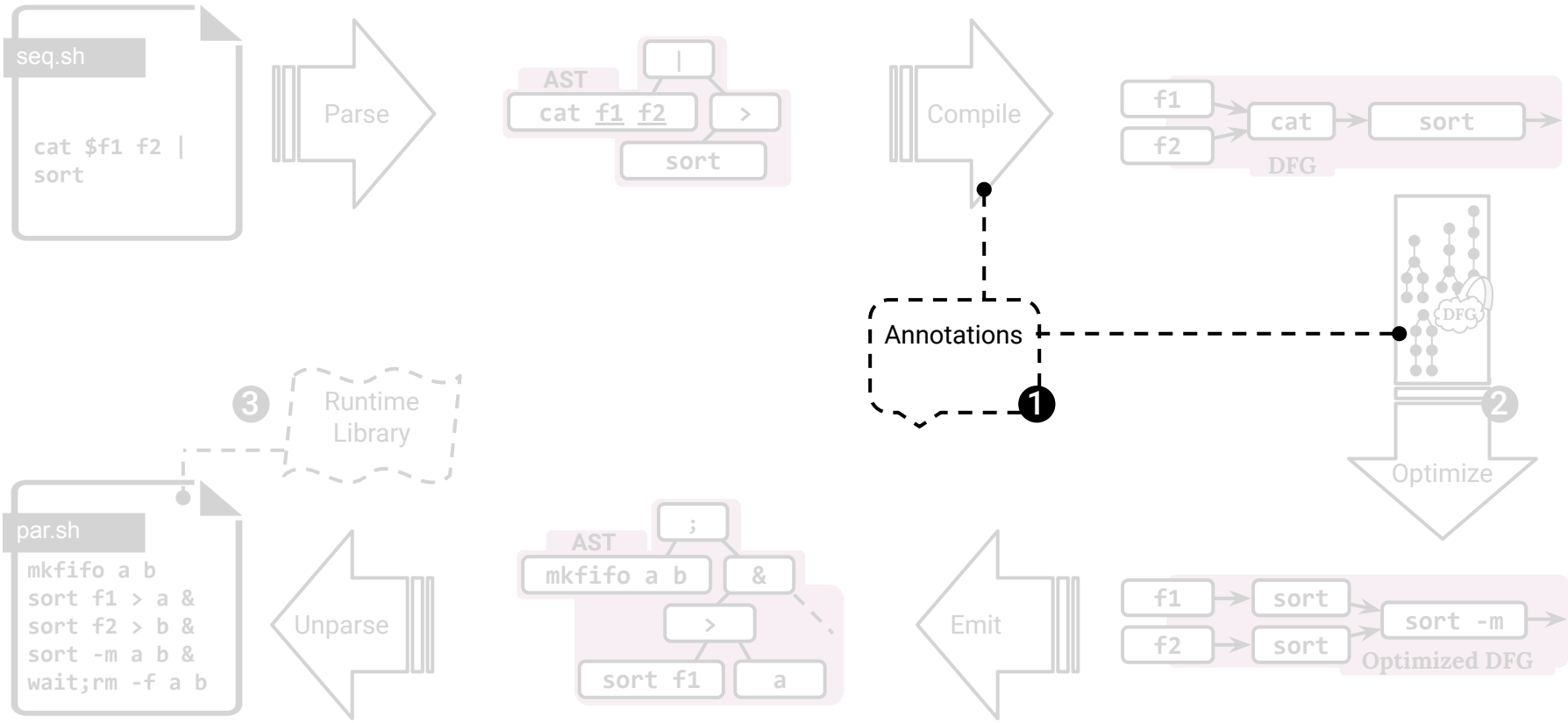


4 command parallelizability classes

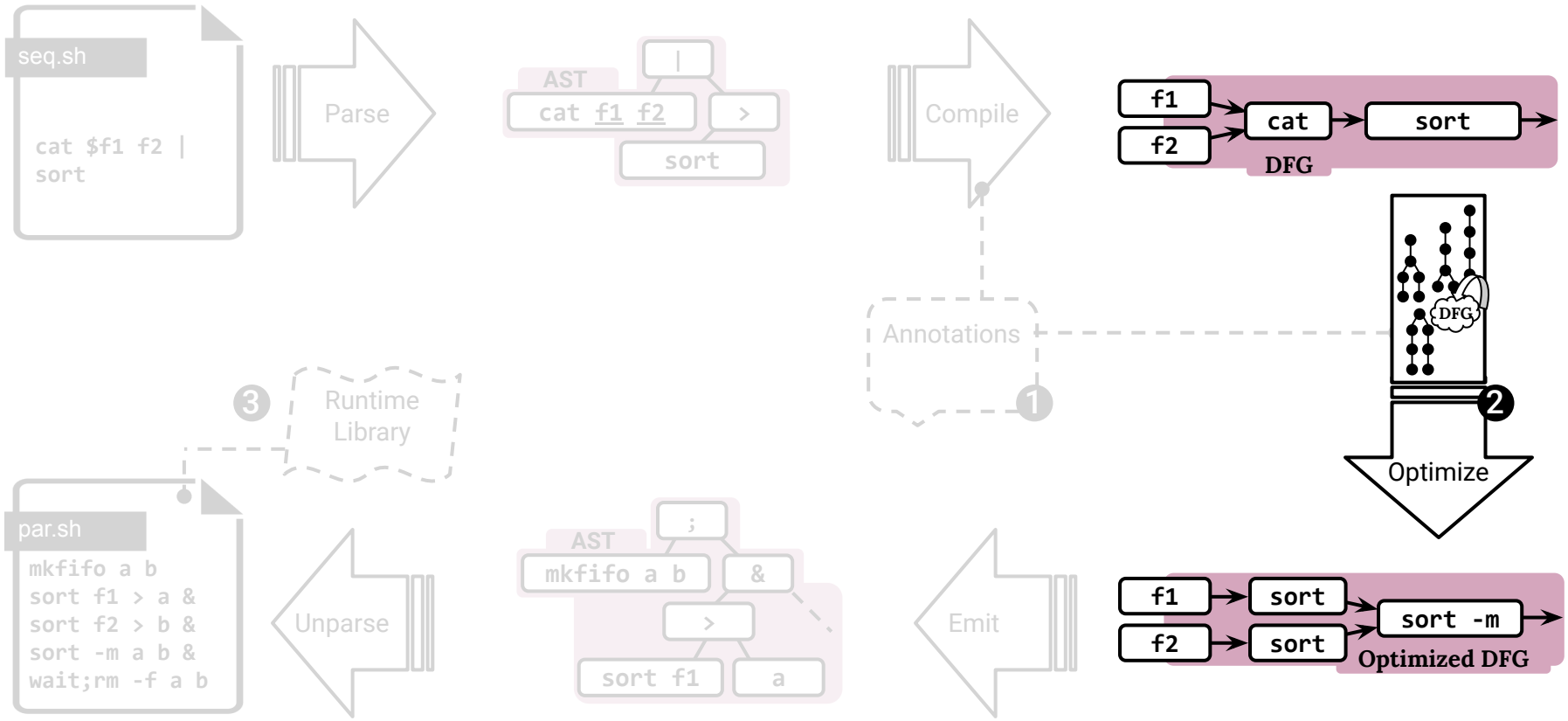
12.7%	stateless
8.7%	parallelizable pure
8.2%	non-parallelizable pure
70.4%	side-effectful



PaSh Overview



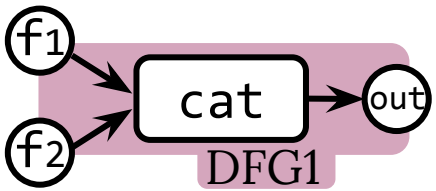
PaSh Overview



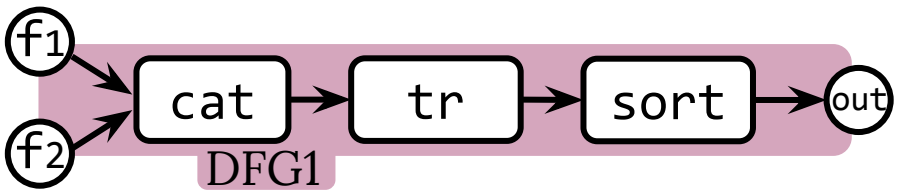
2. Dataflow Model & Transformations

Scheduling constraint

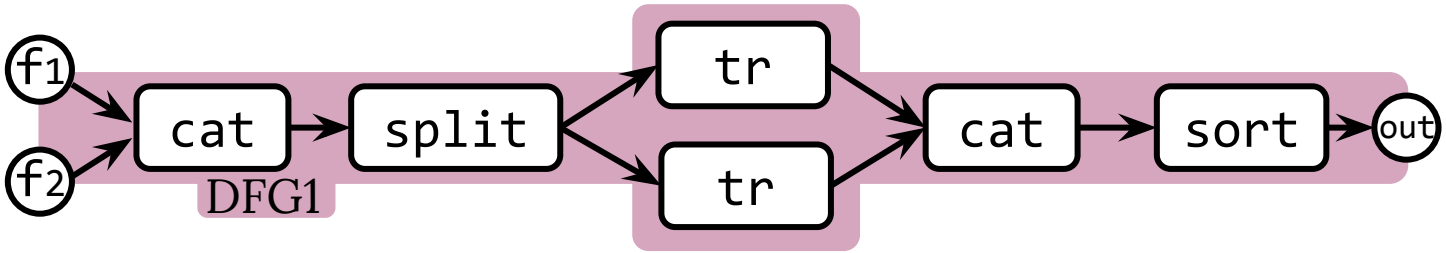
```
cat f1 f2 > out.txt; cat out.txt
```



```
cat f1 f2 | tr A-Z a-z | sort > out.txt
```

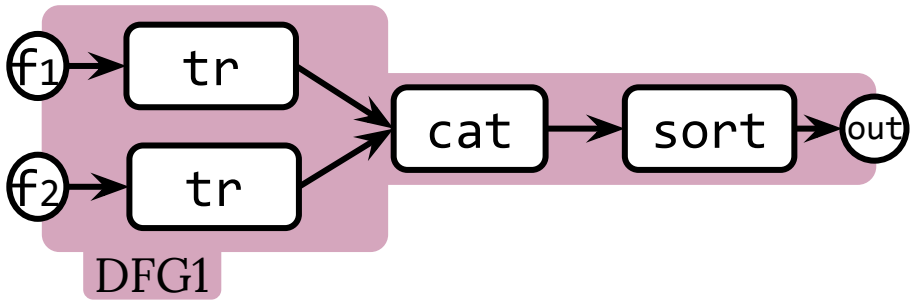


```
cat f1 f2 | tr A-Z a-z | sort > out.txt
```



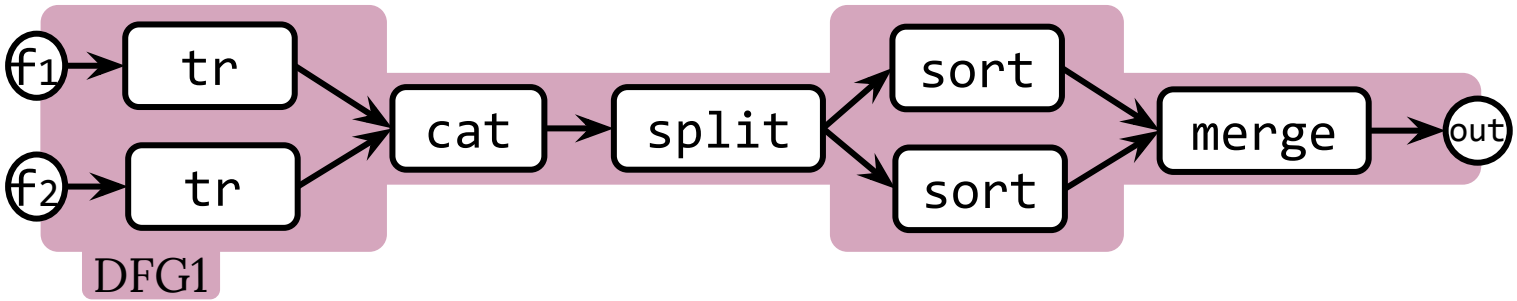
Transformation condition: **`tr`** is stateless

```
cat f1 f2 | tr A-Z a-z | sort > out.txt
```



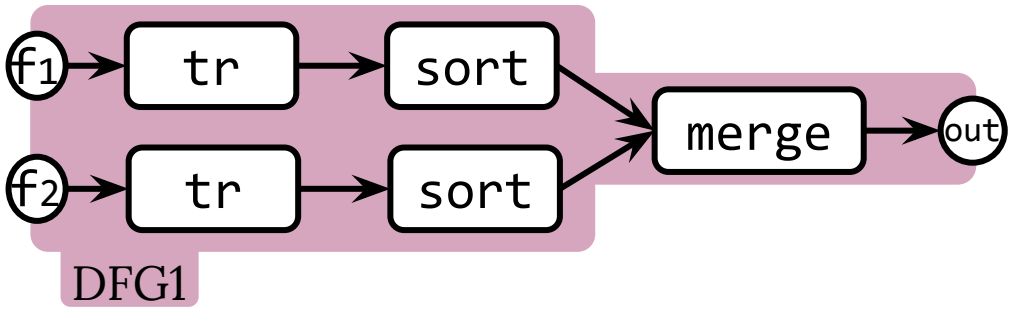
Transformation condition: **`cat`** followed by **`split`**

```
cat f1 f2 | tr A-Z a-z | sort > out.txt
```



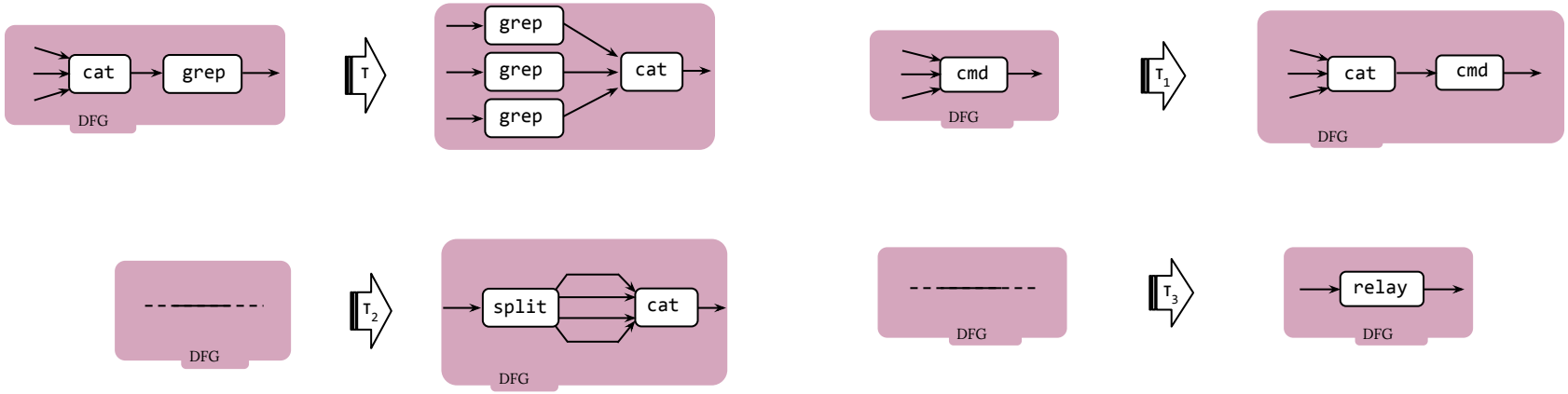
Transformation condition: **`sort`** is parallellizable pure

```
cat f1 f2 | tr A-Z a-z | sort > out.txt
```

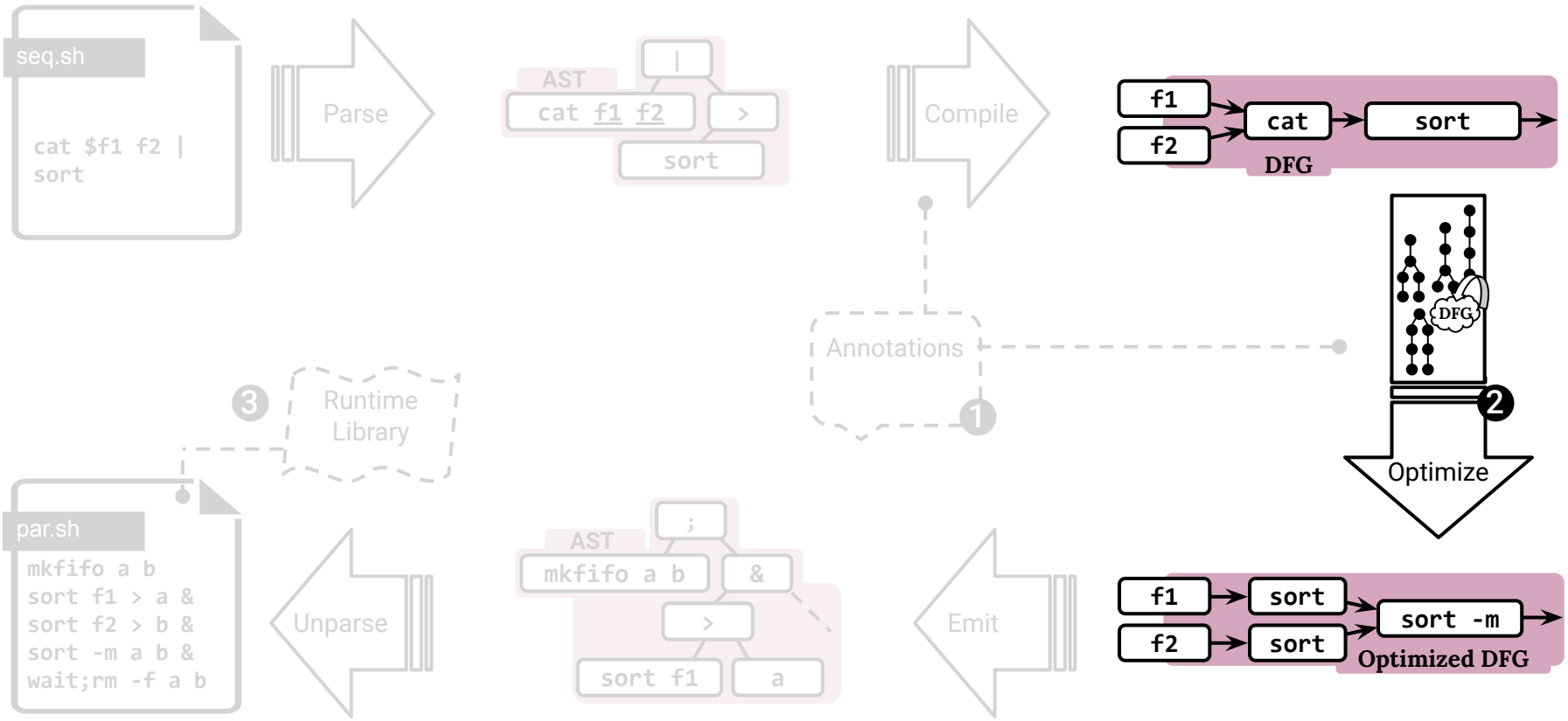


Transformation condition: **`cat`** followed by **`split`**

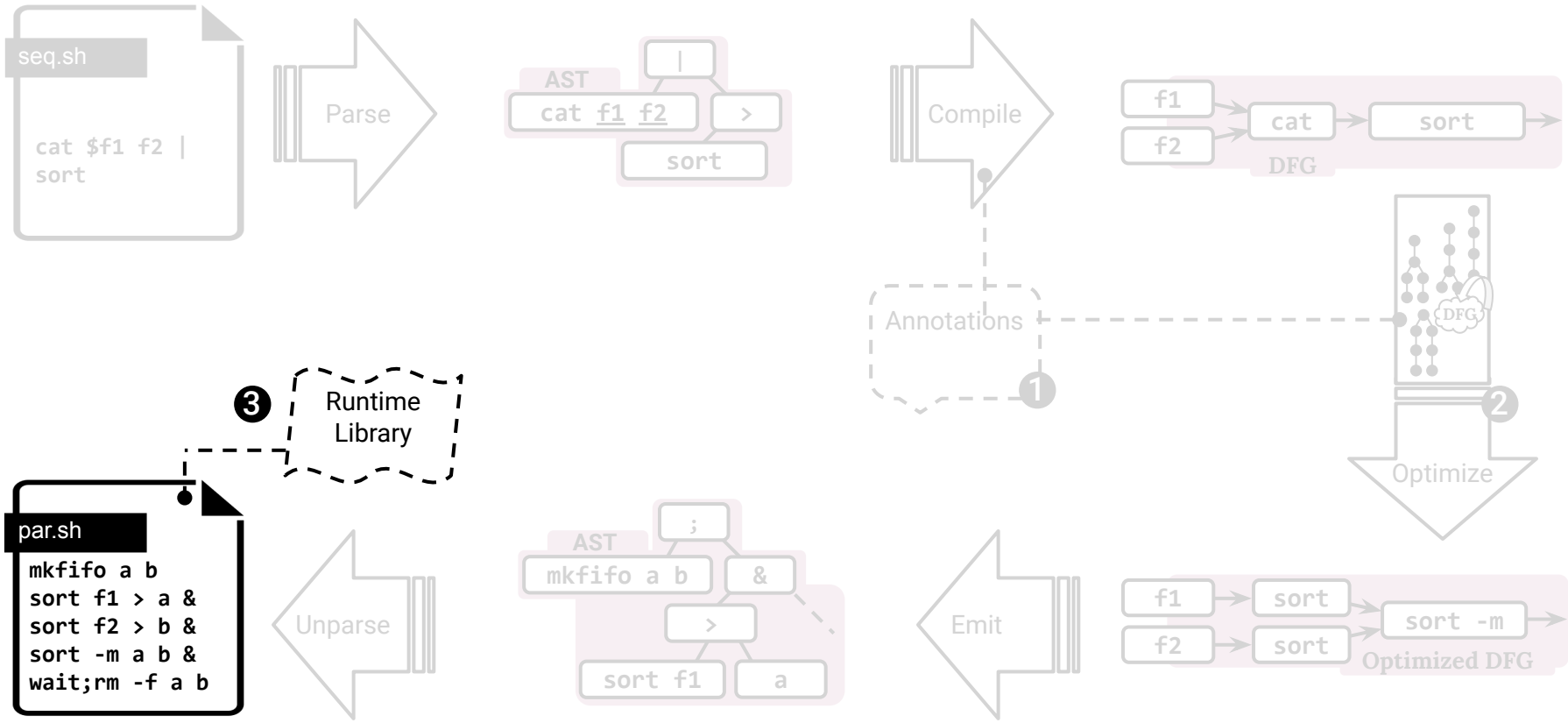
1 + 3 Transformations



PaSh Overview



PaSh Overview



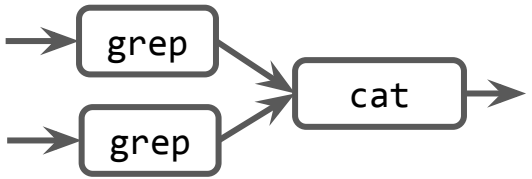
3. Runtime Support

Runtime Support: Performance & Correctness

- Unix pipes are lazy, *i.e.*, inadequate buffering (and for a good reason)
- Dataflow graph termination is tricky
- P*arallelizable-pure* commands require careful aggregation

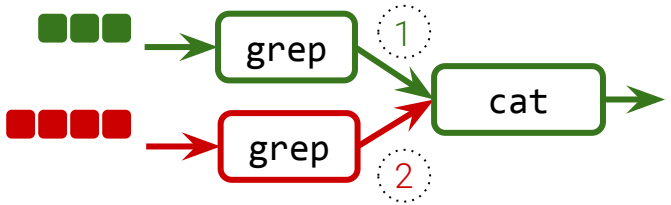
Runtime Challenge: Unix's Lazy Semantics

```
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```



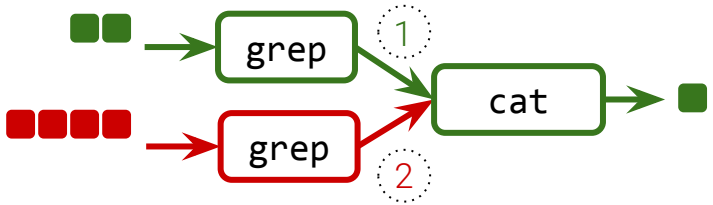
Runtime Challenge: Unix's Lazy Semantics

```
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```



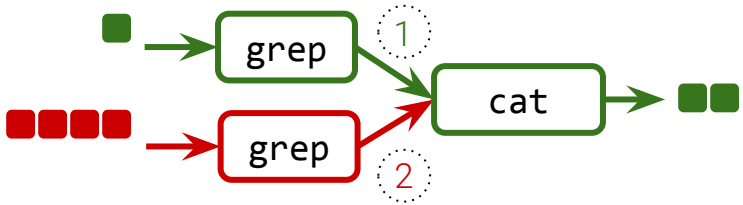
Runtime Challenge: Unix's Lazy Semantics

```
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```



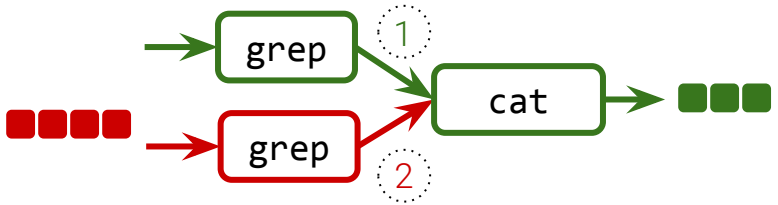
Runtime Challenge: Unix's Lazy Semantics

```
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```



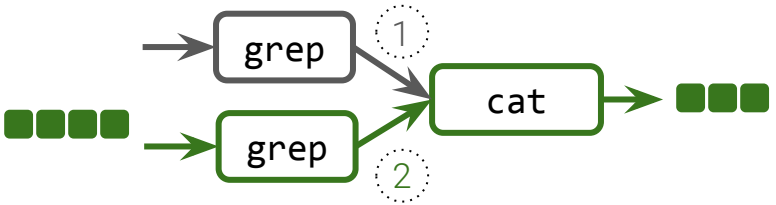
Runtime Challenge: Unix's Lazy Semantics

```
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```



Runtime Challenge: Unix's Lazy Semantics

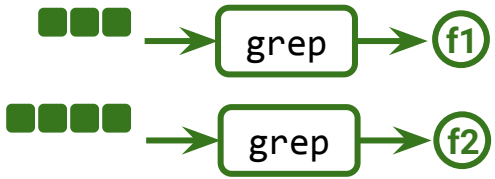
```
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```

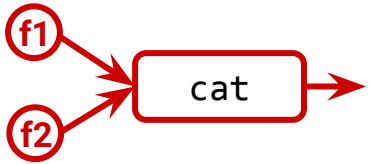


Execution proceeds in steps!

A non-solution: Use intermediary files...

```
touch f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f1 &
wait
cat f1 f1
```

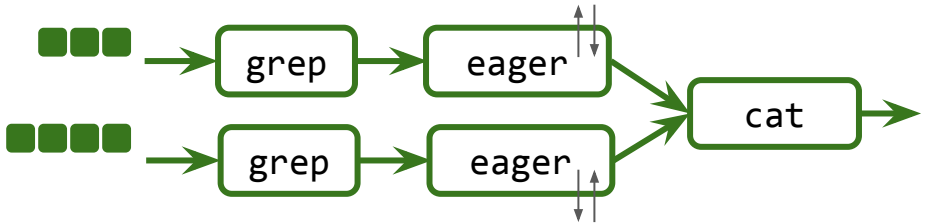




Among other problems, this "solution" prevents pipeline parallelism (more on that later)

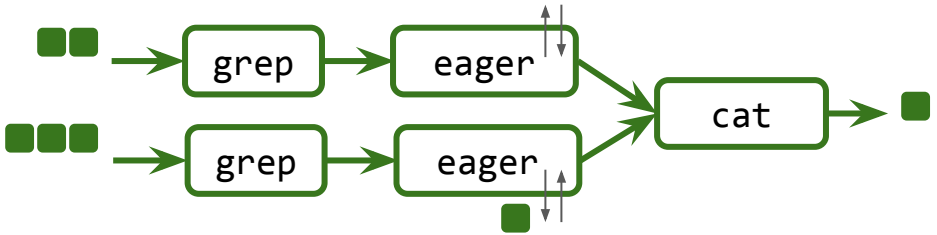
The PaSh Solution: Eager Buffers

```
mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
```



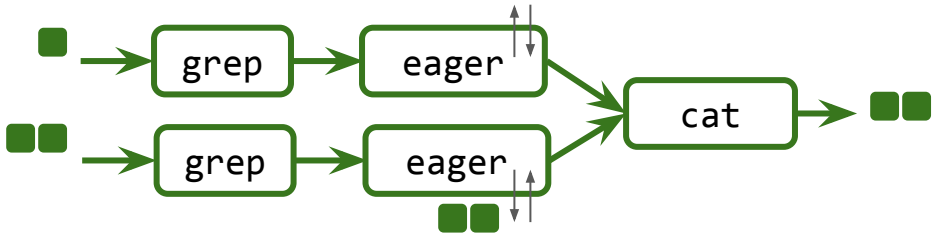
The PaSh Solution: Eager Buffers

```
mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
```



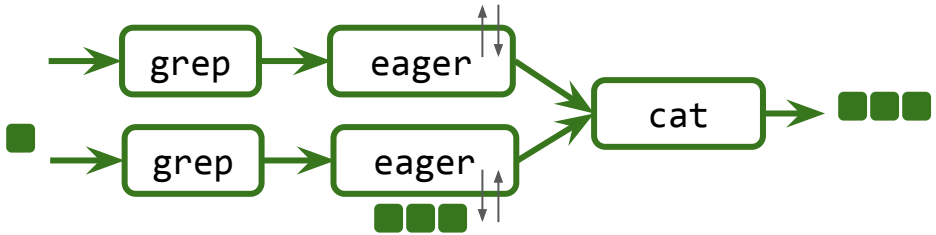
The PaSh Solution: Eager Buffers

```
mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
```



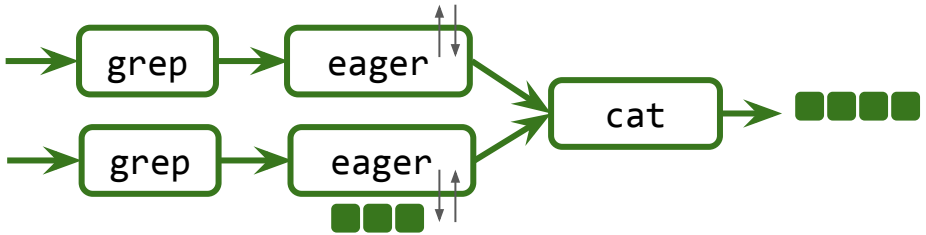
The PaSh Solution: Eager Buffers

```
mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
```



The PaSh Solution: Eager Buffers

```
mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
```



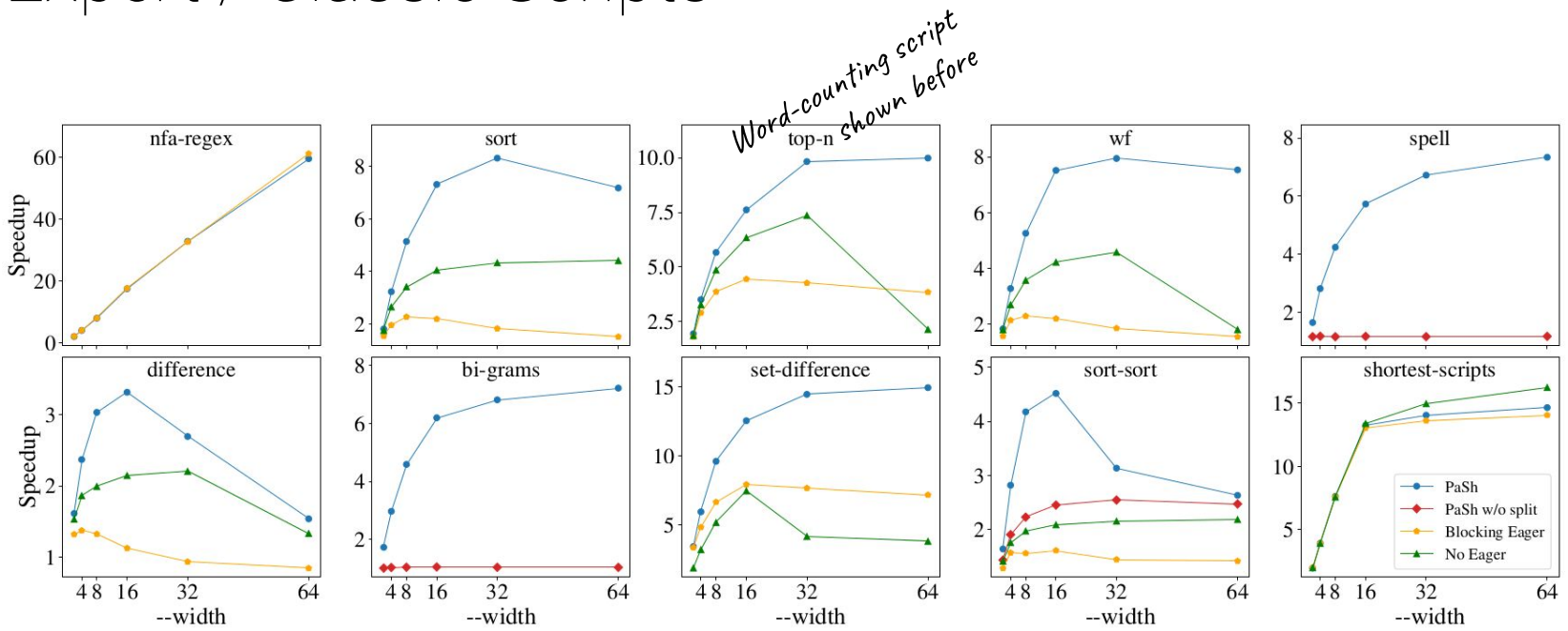
`/pash/runtime/eager`

- Unix command, usable outside PaSh too
- Buffers input eagerly — can spill to disk
- Keeps fragment in DFG model

Demo Time!

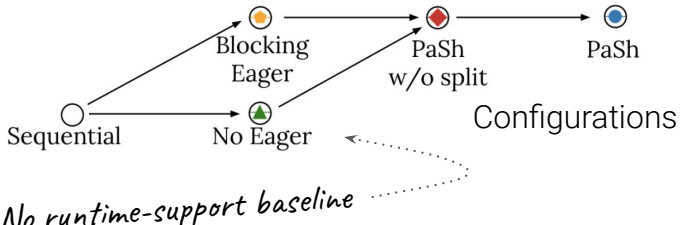
Evaluation

1. Expert / Classic Scripts

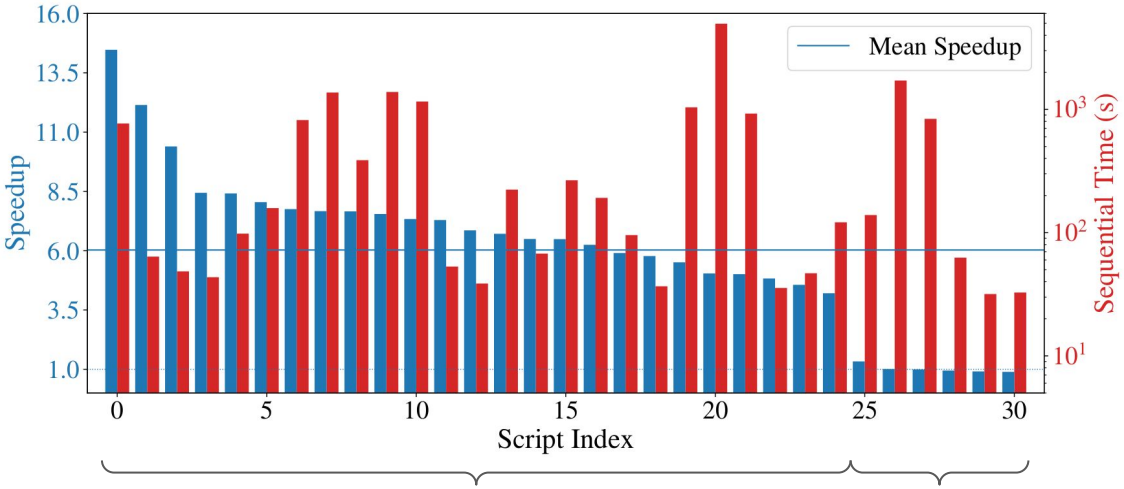


Speedups against bash baseline for `pash --width=16`:

5.93× *vs.* 8.83×



2. Pipelines in the wild



Parallelizable Non parallelizable

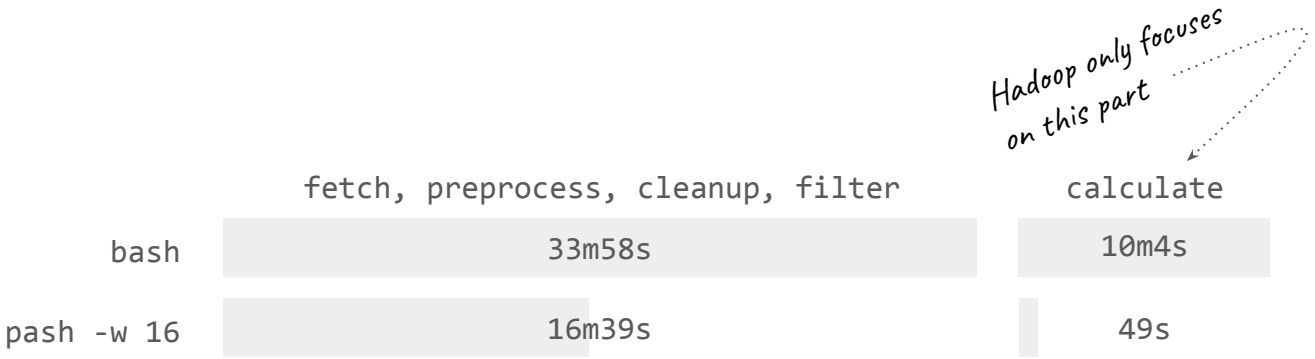
+ PaSh awareness goes a long way!

e.g. #26

```
cat $IN6 | awk '{print $2, $0}' | sort -nr | cut -d ' ' -f 2    (1.01×)
cat $IN6 |                        sort -nr -k2 | cut -d ' ' -f 1    (8.1× !!1!1)
```

Configuration:
- Full PaSh `--width=16`

3. Case Study no.1: NOAA Weather Analysis



2.04×
speedup for preprocessing

12.31×
speedup for preprocessing

2.52×
combined speedup for the full program

This part is not the focus of traditional parallelization frameworks but parallelizing it has the biggest impact

Configuration:
- Full PaSh `--width=16`
- 82GB (5y data)

Conclusion

Conclusion

- Parallelize unix shell scripts (POSIX -> POSIX)
- Annotations address extensibility issues
- Open source — 12+ contributors
- Lots of recent excitement — let's rehabilitate the shell!

pash-discuss@googlegroups.com
github.com/andromeda/pash