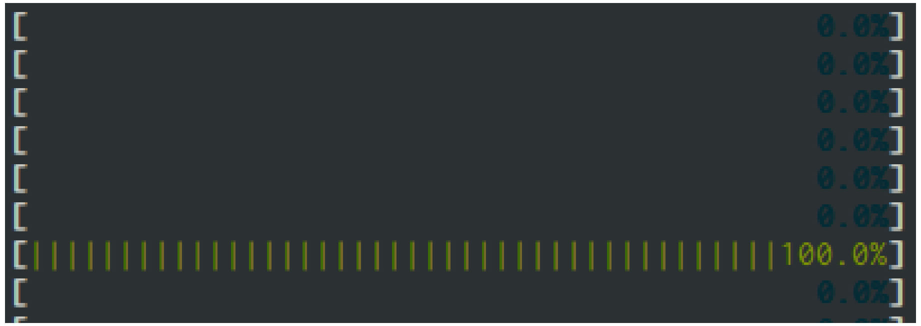
# PaSh: A parallelizing shell

# PaSh: A parallelizing shell

or how to get from this:

# PaSh: A parallelizing shell

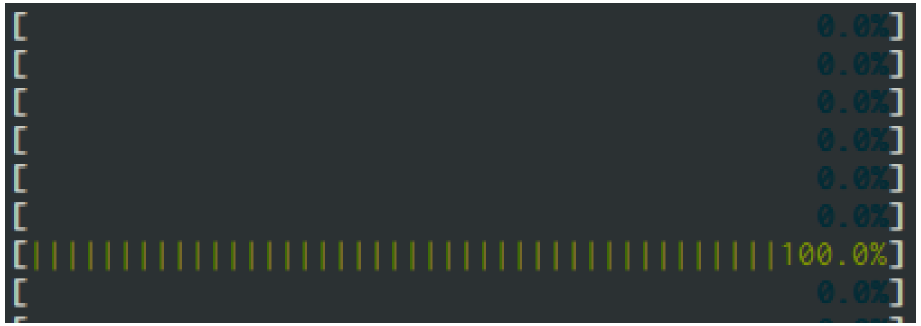
or how to get from this:



```
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
[                                            0.0%]
```

# PaSh: A parallelizing shell

or how to get from this:

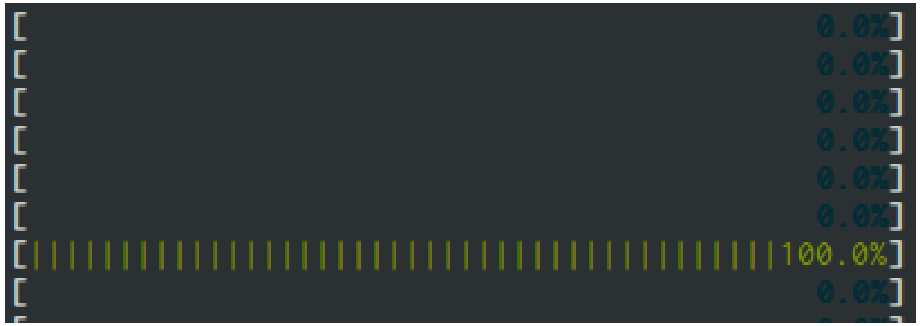


```
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
[                                            0.0%]
```

to this:

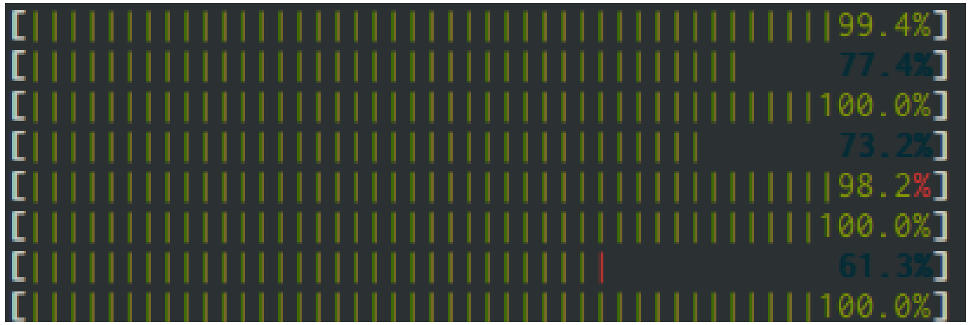
# PaSh: A parallelizing shell

or how to get from this:



```
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
[                                            0.0%]
```

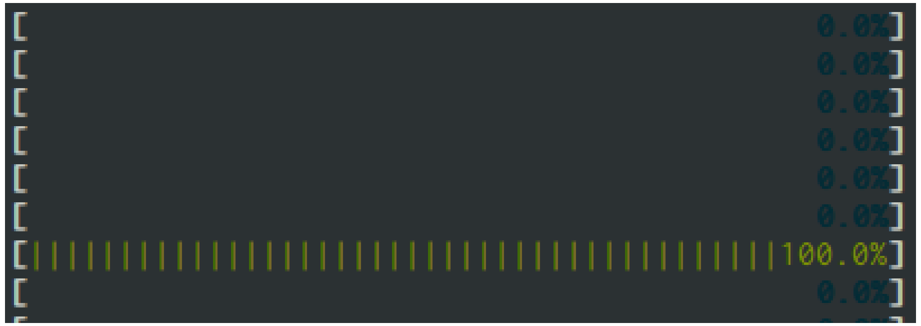
to this:



```
[|||||||||||||||||||||||||||||||||||||||||||99.4%]
[||||||||||||||||||||||||||||||||||||||     77.4%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
[||||||||||||||||||||||||||||||||||||       73.2%]
[|||||||||||||||||||||||||||||||||||||||||||98.2%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
[|||||||||||||||||||||||||||||||            61.3%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
```

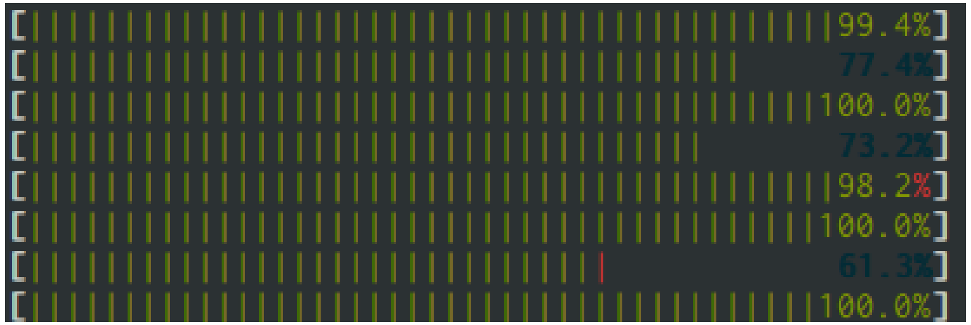
# PaSh: A parallelizing shell

or how to get from this:



```
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[                                            0.0%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
[                                            0.0%]
```

to this:



```
[|||||||||||||||||||||||||||||||||||||||||||99.4%]
[||||||||||||||||||||||||||||||||||||||     77.4%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
[||||||||||||||||||||||||||||||||||||       73.2%]
[|||||||||||||||||||||||||||||||||||||||||||98.2%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
[|||||||||||||||||||||||||||||||            61.3%]
[||||||||||||||||||||||||||||||||||||||||||100.0%]
```

# Joint work with:

Nikos Vasilakis

And many others (in alphabetical order):

Achilles Benetopoulos

Lazar Cvetkovic

Thurston Dang

Michael Greenberg

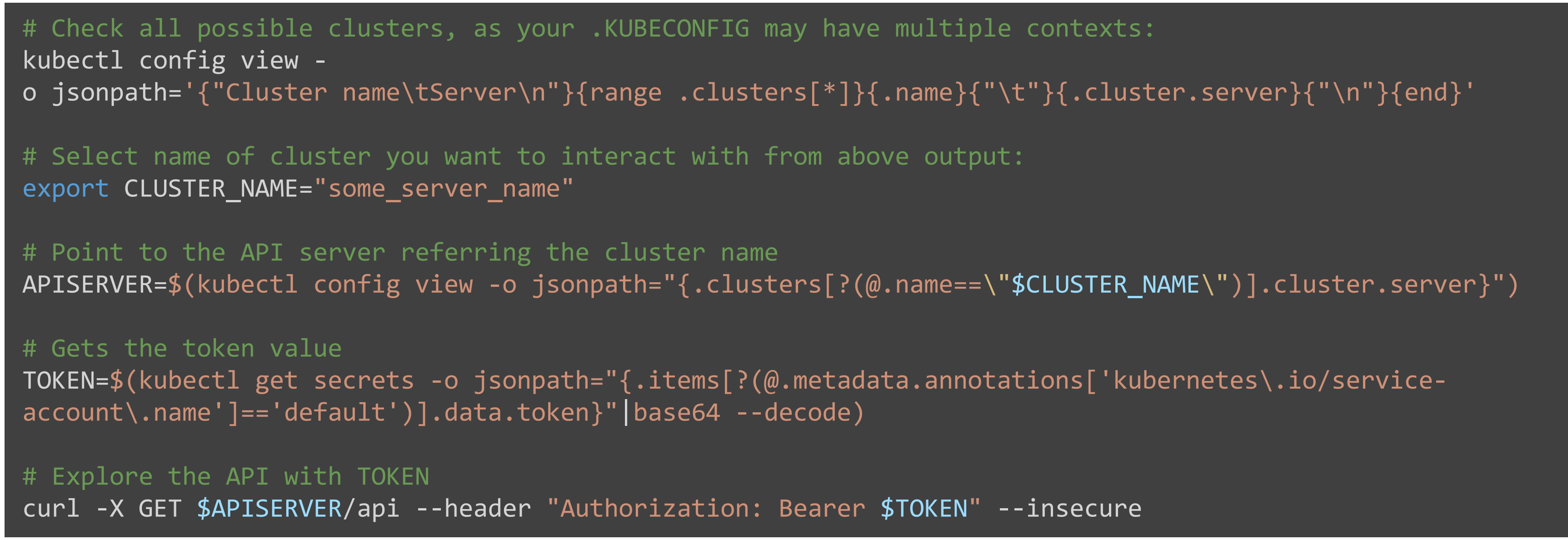
Shivam Handa

Konstantinos Mamouras

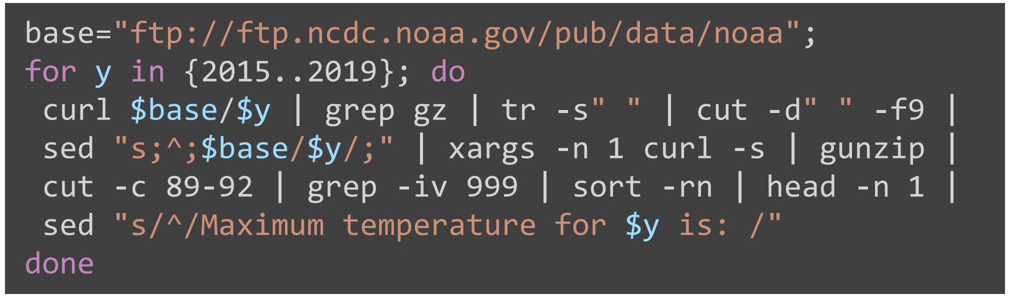
Martin Rinard

# Used by everyone!

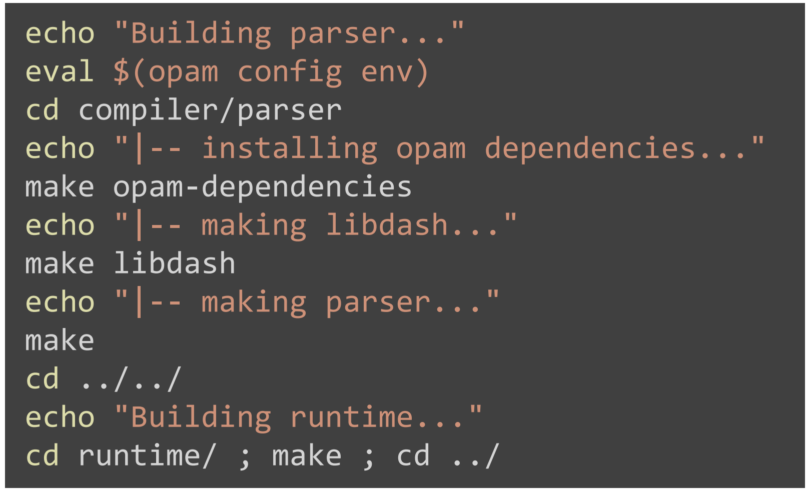
- Orchestration
  - Kubernetes deployment
  - Docket containers …
- Data processing:
  - Downloading
  - Extracting
  - Preprocessing
  - Querying
- Automation Tasks
  - Configuration
  - Installation



```
# Check all possible clusters, as your .KUBECONFIG may have multiple contexts:
kubectl config view -
o jsonpath='{"Cluster name\tServer\n"}{range .clusters[*]}{.name}{"\t"}{.cluster.server}{"\n"}{end}'

# Select name of cluster you want to interact with from above output:
export CLUSTER_NAME="some_server_name"

# Point to the API server referring the cluster name
APISERVER=$(kubectl config view -o jsonpath="{.clusters[?(@.name==\"$CLUSTER_NAME\")].cluster.server}")

# Gets the token value
TOKEN=$(kubectl get secrets -o jsonpath="{.items[?(@.metadata.annotations['kubernetes\.io/service-
account\.name']=='default')].data.token}"|base64 --decode)

# Explore the API with TOKEN
curl -X GET $APISERVER/api --header "Authorization: Bearer $TOKEN" --insecure
```



```
base="ftp://ftp.ncdc.noaa.gov/pub/data/noaa";
for y in {2015..2019}; do
 curl $base/$y | grep gz | tr -s" " | cut -d" " -f9 |
 sed "s;^;$base/$y/;" | xargs -n 1 curl -s | gunzip |
 cut -c 89-92 | grep -iv 999 | sort -rn | head -n 1 |
 sed "s/^/Maximum temperature for $y is: /"
done
```



```
echo "Building parser..."
eval $(opam config env)
cd compiler/parser
echo "|-- installing opam dependencies..."
make opam-dependencies
echo "|-- making libdash..."
make libdash
echo "|-- making parser..."
make
cd ../../
echo "Building runtime..."
cd runtime/ ; make ; cd ../
```

# The Problem

Shell scripts are mostly sequential! :'(

# Parallelism could help

# Parallelism could help

But it requires manual effort:

# Parallelism could help

But it requires manual effort:

- Using specific command flags (e.g., sort -p, make -jN)

# Parallelism could help

But it requires manual effort:

- Using specific command flags (e.g., sort -p, make -jN)
- Using semi-automatic restricted parallelization tools (e.g., GNU parallel)

# Parallelism could help

But it requires manual effort:

- Using specific command flags (e.g., sort -p, make -jN)
- Using semi-automatic restricted parallelization tools (e.g., GNU parallel)
- Manually parallelizing using the background (&) operator

# Parallelism could help

But it requires manual effort:

- Using specific command flags (e.g., sort -p, make -jN)
- Using semi-automatic restricted parallelization tools (e.g., GNU parallel)
- Manually parallelizing using the background (&) operator
- Manually parallelizing by rewriting parts of a script in parallel frameworks (e.g., MR)

# Challenges

# Challenges

1. Lack of static information:

# Challenges

1. Lack of static information:
   - Shell execution depends on dynamic components (file system, environment variables, etc)

# Challenges

1. Lack of static information:
   - Shell execution depends on dynamic components (file system, environment variables, etc)
   - This makes a static parallelization procedure that is always sound and effective impossible

# Challenges

1. Lack of static information:
   - Shell execution depends on dynamic components (file system, environment variables, etc)
   - This makes a static parallelization procedure that is always sound and effective impossible
2. Subtle parallelism:

# Challenges

1. Lack of static information:
   - Shell execution depends on dynamic components (file system, environment variables, etc)
   - This makes a static parallelization procedure that is always sound and effective impossible
2. Subtle parallelism:
   - Properties such as commutativity, or independence by key are not satisfied by many commands

# Challenges

1. Lack of static information:
    - Shell execution depends on dynamic components (file system, environment variables, etc)
    - This makes a static parallelization procedure that is always sound and effective impossible
2. Subtle parallelism:
    - Properties such as commutativity, or independence by key are not satisfied by many commands
    - This requires a finer parallelism model that captures the properties of shell commands

# Challenges

1. Lack of static information:
   - Shell execution depends on dynamic components (file system, environment variables, etc)
   - This makes a static parallelization procedure that is always sound and effective impossible
2. Subtle parallelism:
   - Properties such as commutativity, or independence by key are not satisfied by many commands
   - This requires a finer parallelism model that captures the properties of shell commands
3. Arbitrary black-box commands:

# Challenges

1. Lack of static information:
   - Shell execution depends on dynamic components (file system, environment variables, etc)
   - This makes a static parallelization procedure that is always sound and effective impossible
2. Subtle parallelism:
   - Properties such as commutativity, or independence by key are not satisfied by many commands
   - This requires a finer parallelism model that captures the properties of shell commands
3. Arbitrary black-box commands:
   - Shell commands are written in arbitrary languages and are constantly updated or modified

# Challenges

1. Lack of static information:
   - Shell execution depends on dynamic components (file system, environment variables, etc)
   - This makes a static parallelization procedure that is always sound and effective impossible
2. Subtle parallelism:
   - Properties such as commutativity, or independence by key are not satisfied by many commands
   - This requires a finer parallelism model that captures the properties of shell commands
3. Arbitrary black-box commands:
   - Shell commands are written in arbitrary languages and are constantly updated or modified
   - This makes an automated command analysis infeasible and a one-time manual analysis useless

# PaSh

# PaSh

A tool that:

# PaSh

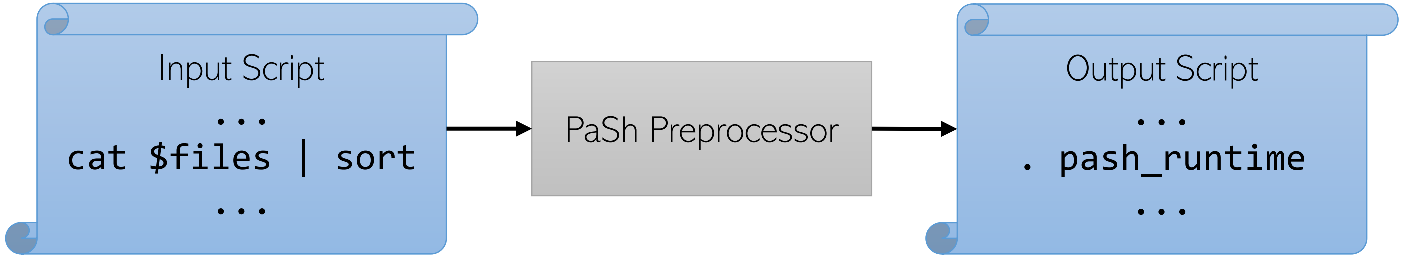
A tool that:

- exposes latent data parallelism in shell scripts

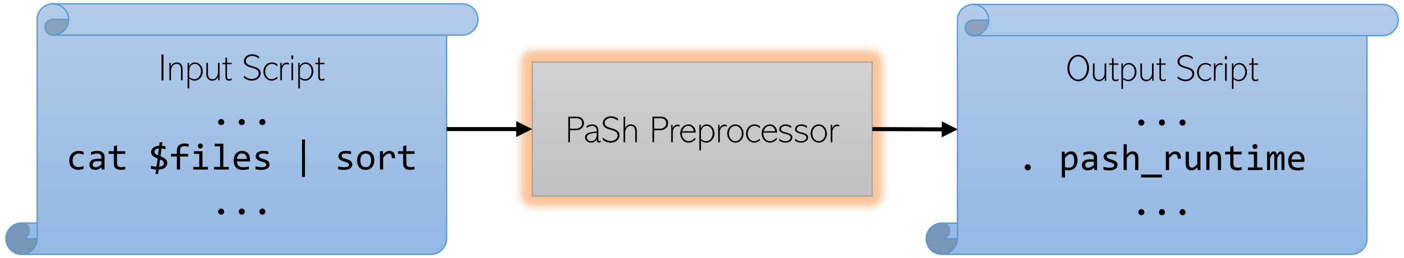
# PaSh

A tool that:

- exposes latent data parallelism in shell scripts
- is a lightweight layer on top of bash



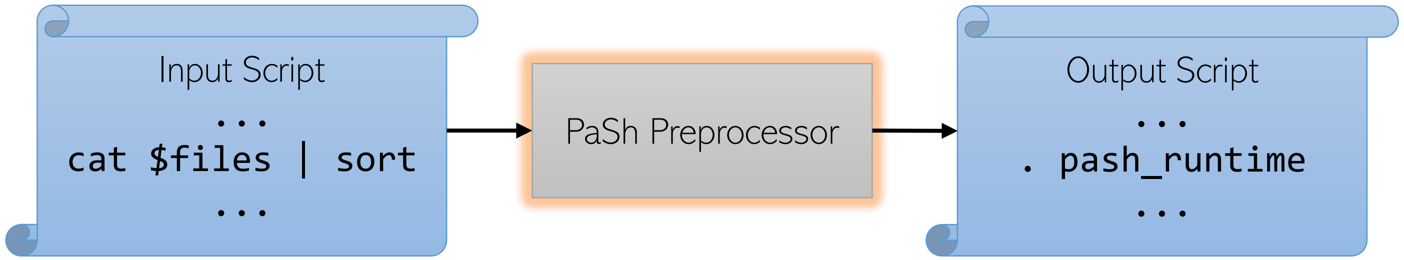
# PaSh



```
...
cat $files | sort
...
```

```
...
. pash_runtime
...
```

# PaSh



Input Script

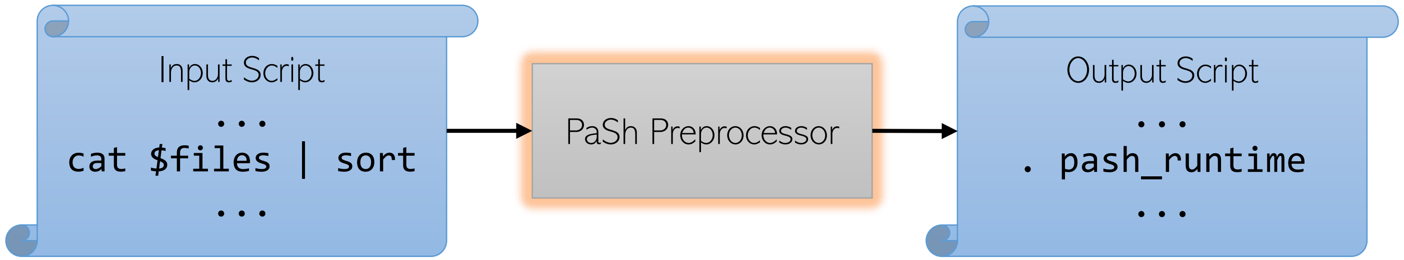
```
...
cat $files | sort
...
```

Output Script

```
...
. pash_runtime
...
```

pash_runtime

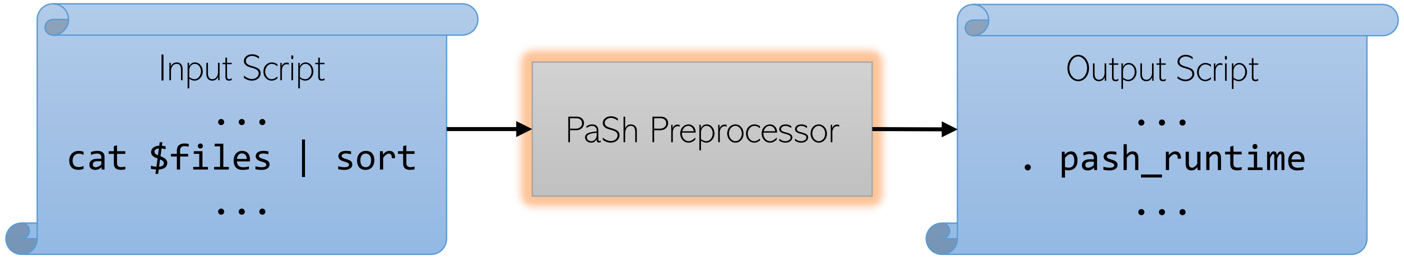
# PaSh



pash_runtime

```
cat $files |
    sort
```

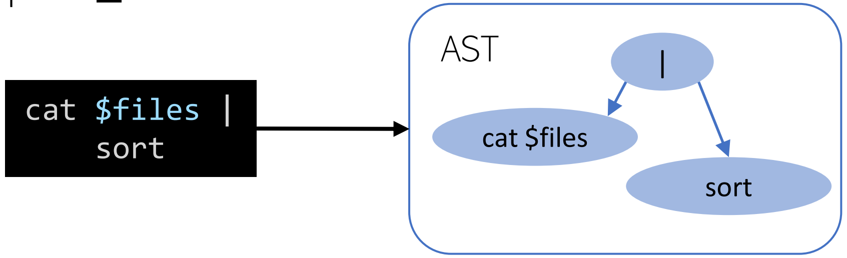
# PaSh



```
...
cat $files | sort
...
```

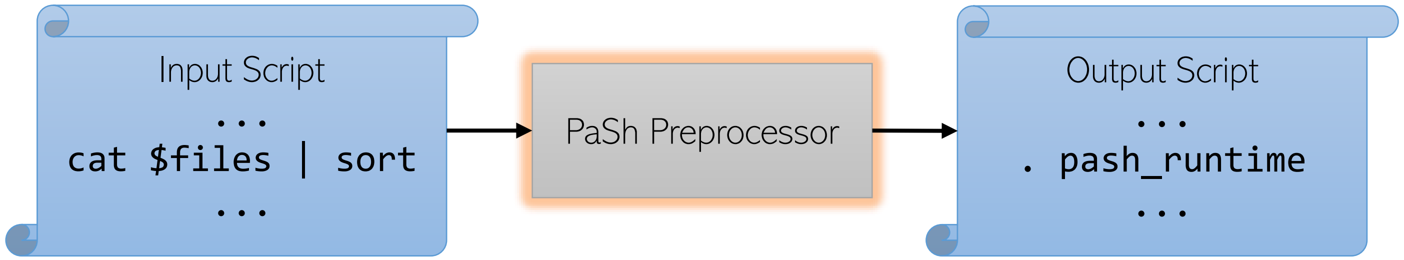
```
...
. pash_runtime
...
```

pash_runtime



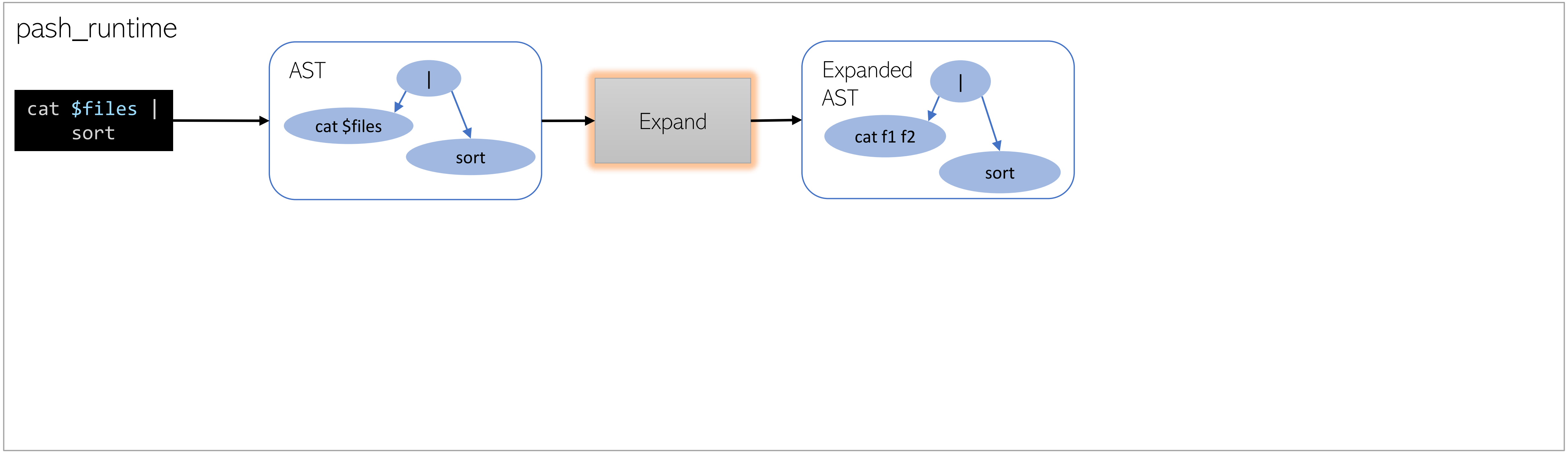
```
cat $files |
    sort
```

# PaSh



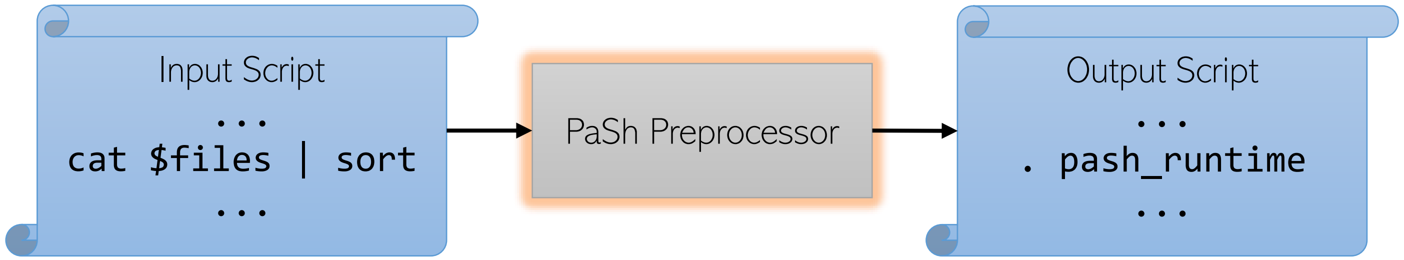
```
...
cat $files | sort
...
```

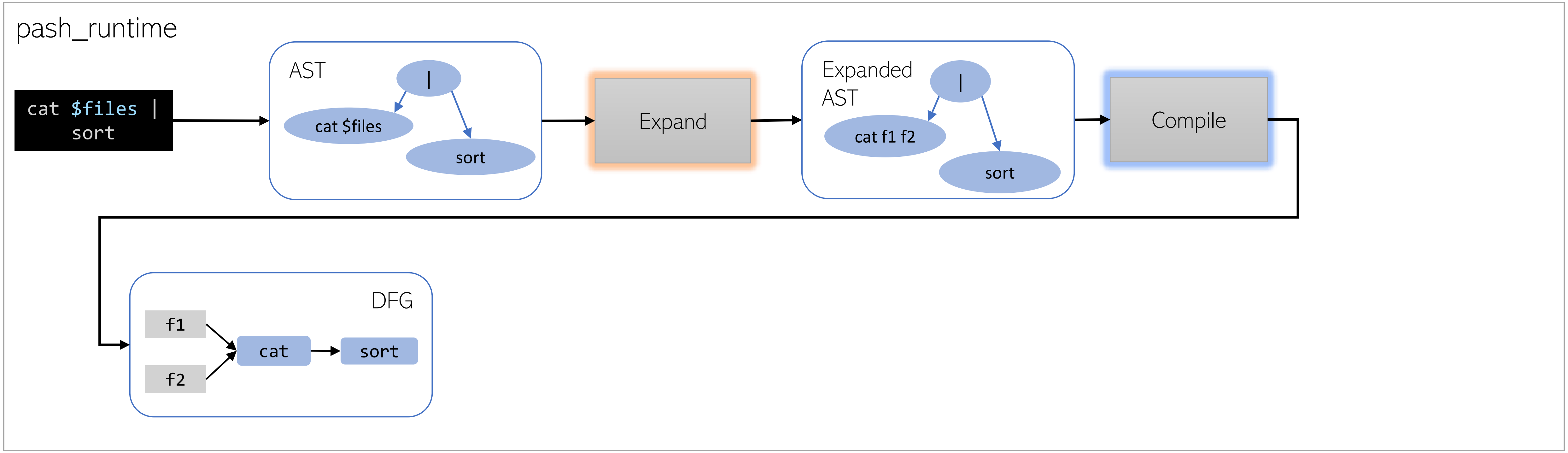
```
...
. pash_runtime
...
```



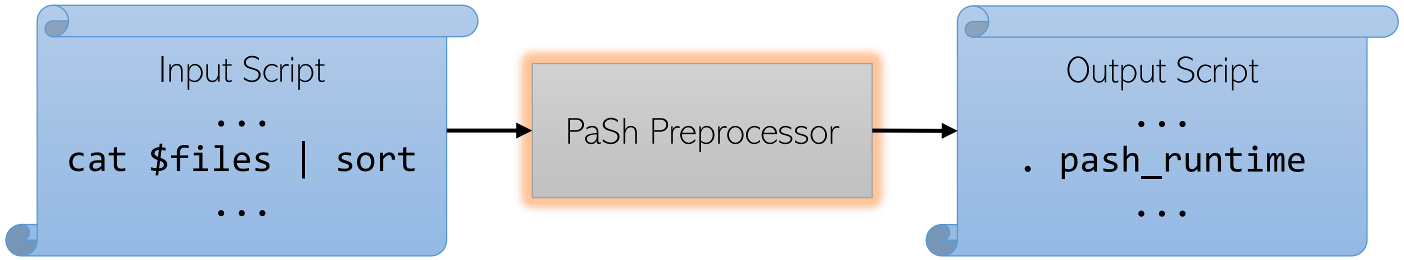
```
cat $files |
   sort
```

# PaSh


Input Script
...
cat $files | sort
...
PaSh Preprocessor
Output Script
...
. pash_runtime
...

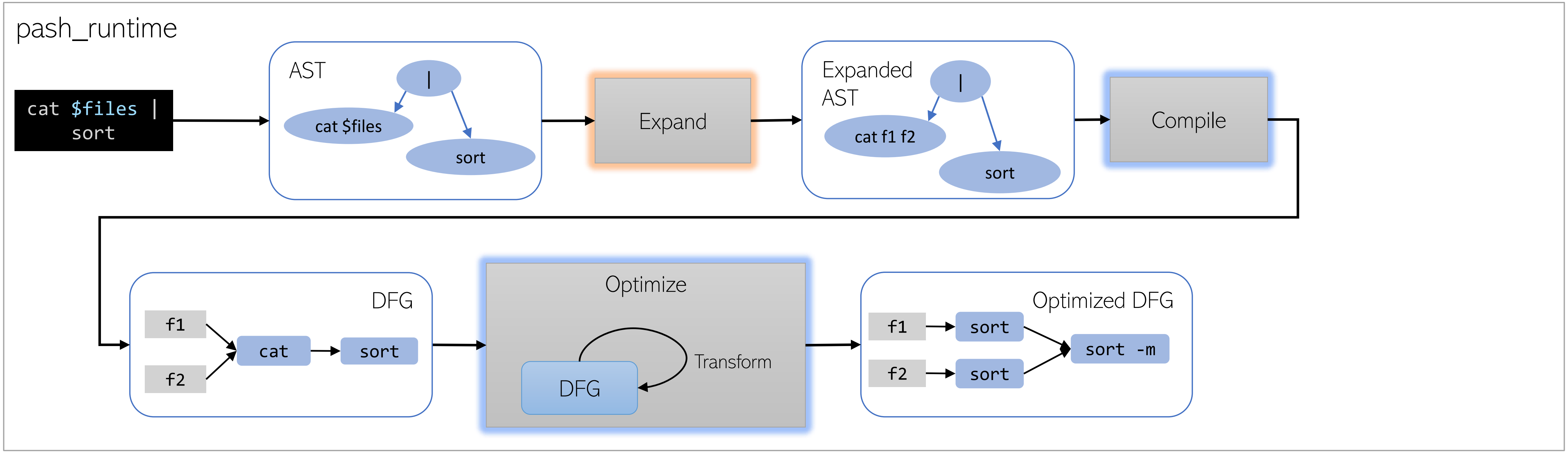

pash_runtime
cat $files |
sort
AST
|
cat $files
sort
Expand
Expanded
AST
|
cat f1 f2
sort
Compile
DFG
f1
f2
cat
sort

# PaSh



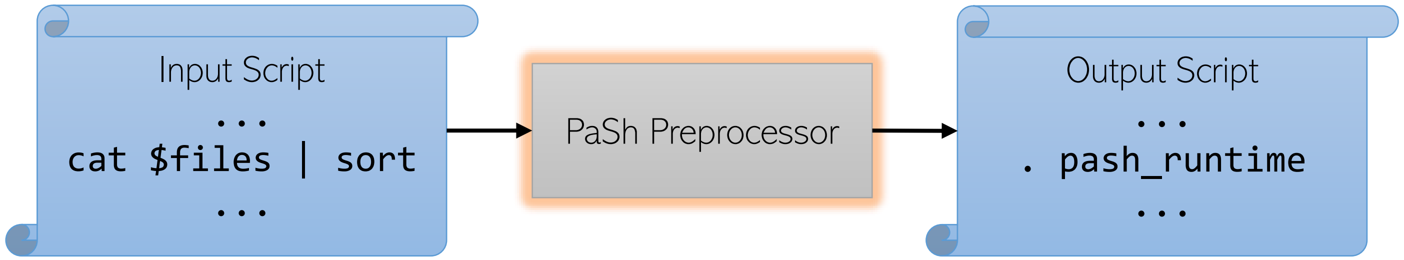
```
...
cat $files | sort
...
```

```
...
. pash_runtime
...
```



```
cat $files |
  sort
```

# PaSh



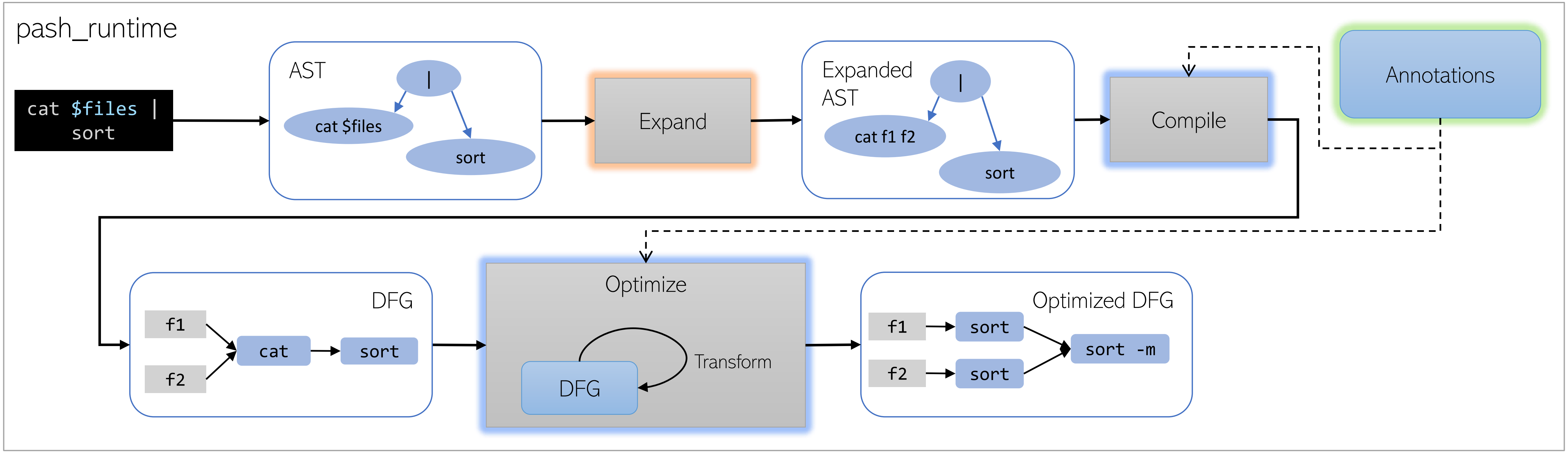
Input Script

```
...
cat $files | sort
...
```

PaSh Preprocessor

Output Script

```
...
. pash_runtime
...
```



pash_runtime

```
cat $files |
sort
```

AST

|

cat $files

sort

Expand

Expanded AST

|

cat f1 f2

sort

Compile

Annotations

DFG

f1

f2

cat

sort

Optimize

DFG

Transform

Optimized DFG

f1

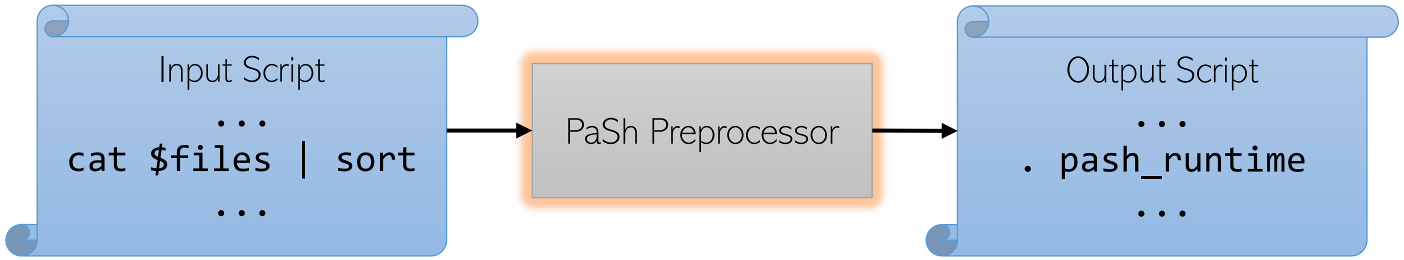
f2

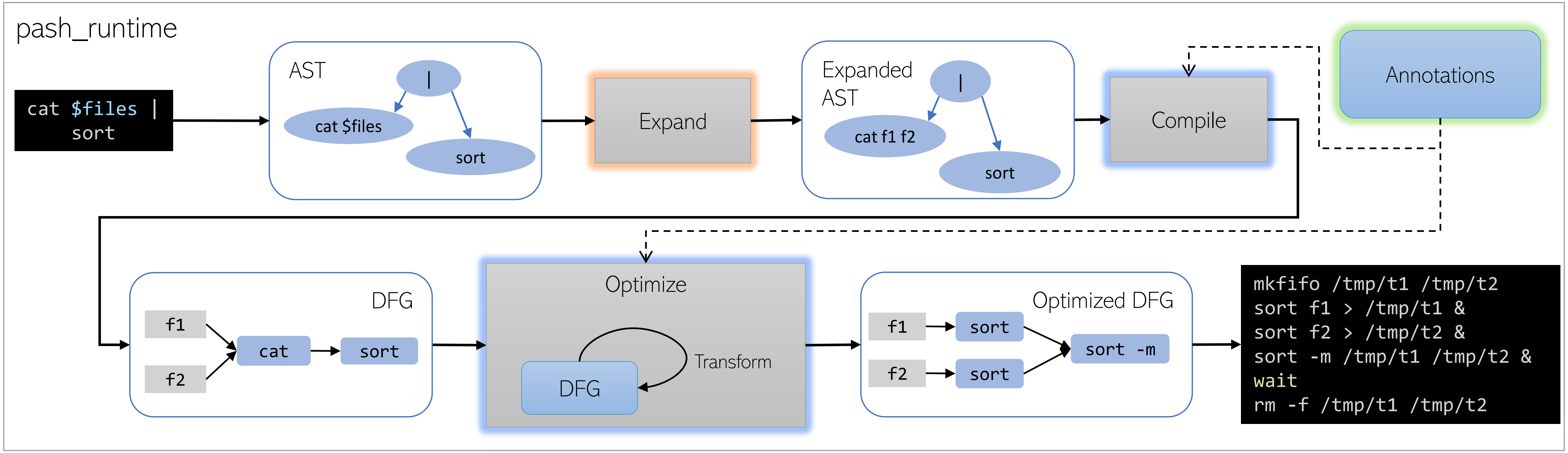
sort

sort

sort -m

# PaSh

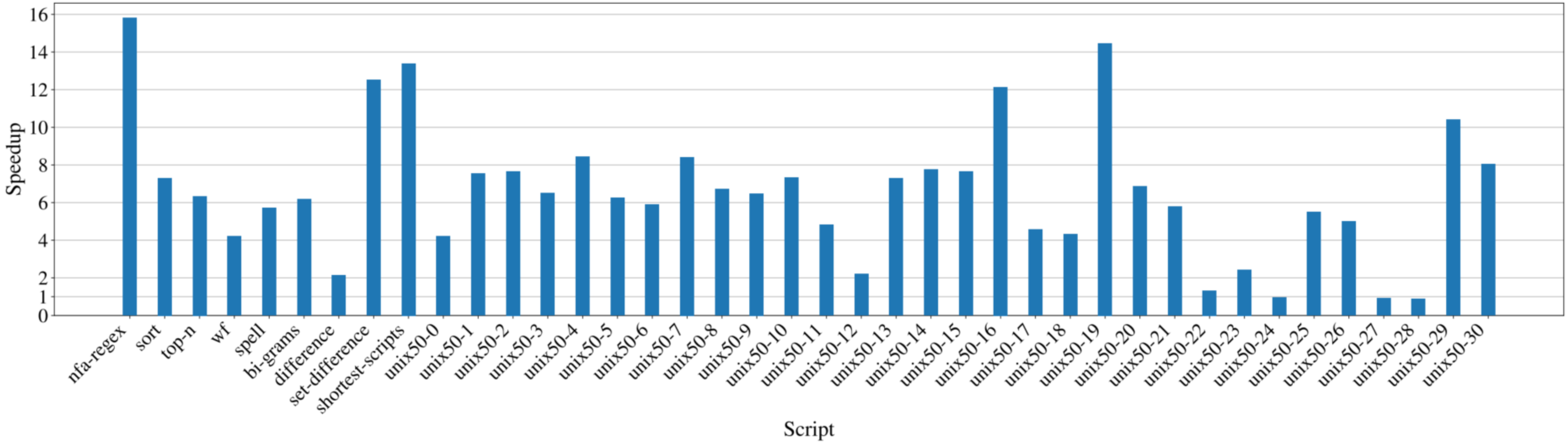




```
cat $files |
  sort
```

```
mkfifo /tmp/t1 /tmp/t2
sort f1 > /tmp/t1 &
sort f2 > /tmp/t2 &
sort -m /tmp/t1 /tmp/t2 &
wait
rm -f /tmp/t1 /tmp/t2
```

# High speedups!!!



Average: 6.56x, Maximum: 15.81x, Minimum: 0.89x

# Come chat ☺

- If you want to learn more
- If you are interested in trying out PaSh
- If you have long running scripts that might benefit from parallelism
- If you would like to collaborate

Come and chat in the poster session ☺