Practically Correct, Just-in-Time Shell Script Parallelization

OSDI 2022

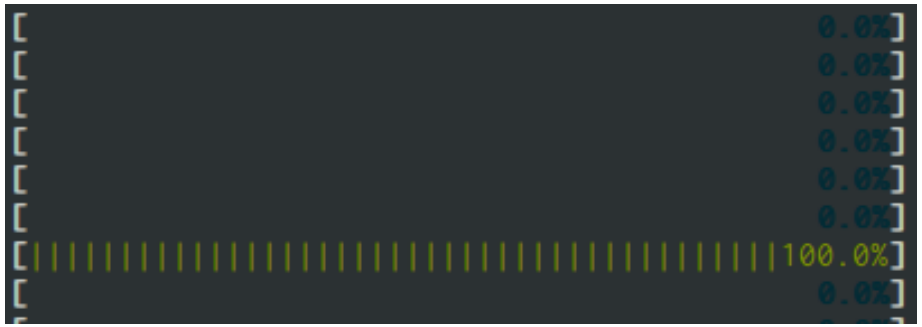
binpa.sh github.com/binpash/pash

THE LINUX FOUNDATION

Practically Correct, Just-in-Time Shell Script Parallelization

OSDI 2022

or how to get from this:



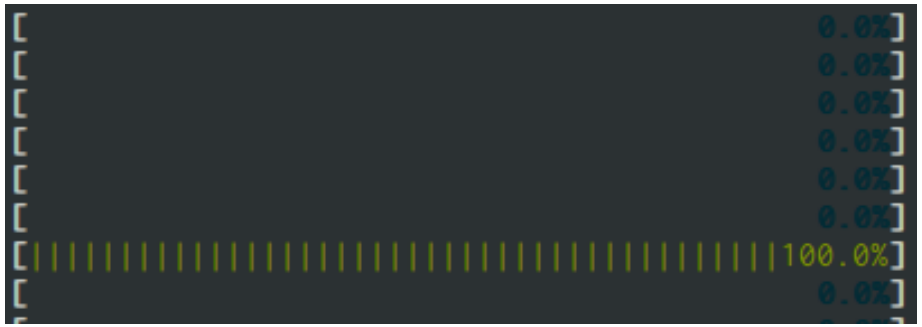
binpa.sh github.com/binpash/pash

THE LINUX FOUNDATION

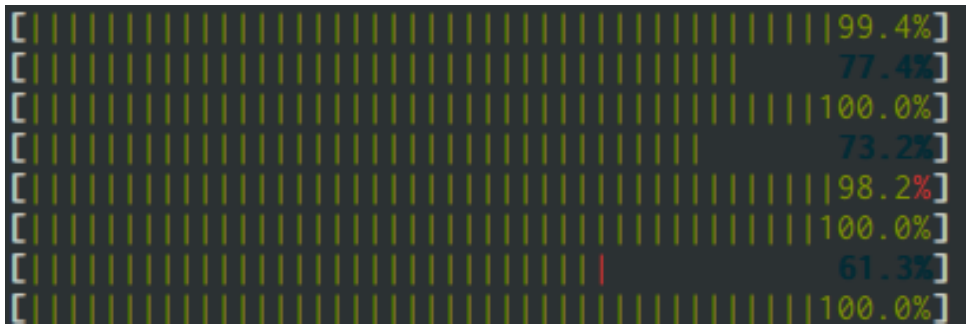
Practically Correct, Just-in-Time Shell Script Parallelization

OSDI 2022

or how to get from this:



to this:



ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

binpa.sh github.com/binpash/pash

THE LINUX FOUNDATION

Joint work with:


Tammam Mustafa


Jan Bielak


Dimitris Karnikis


Thurston Dang


Michael Greenberg


Nikos Vasilakis

Building on work by many others (in alphabetical order):


Achilles Benetopoulos


Lazar Cvetkovic


Shivam Handa


Kostas Mamouras


Radha Patel


Martin Rinard

shell

Used by everyone!

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

Used by everyone!

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

```
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
```

Used by everyone!

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

```
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 ../
```

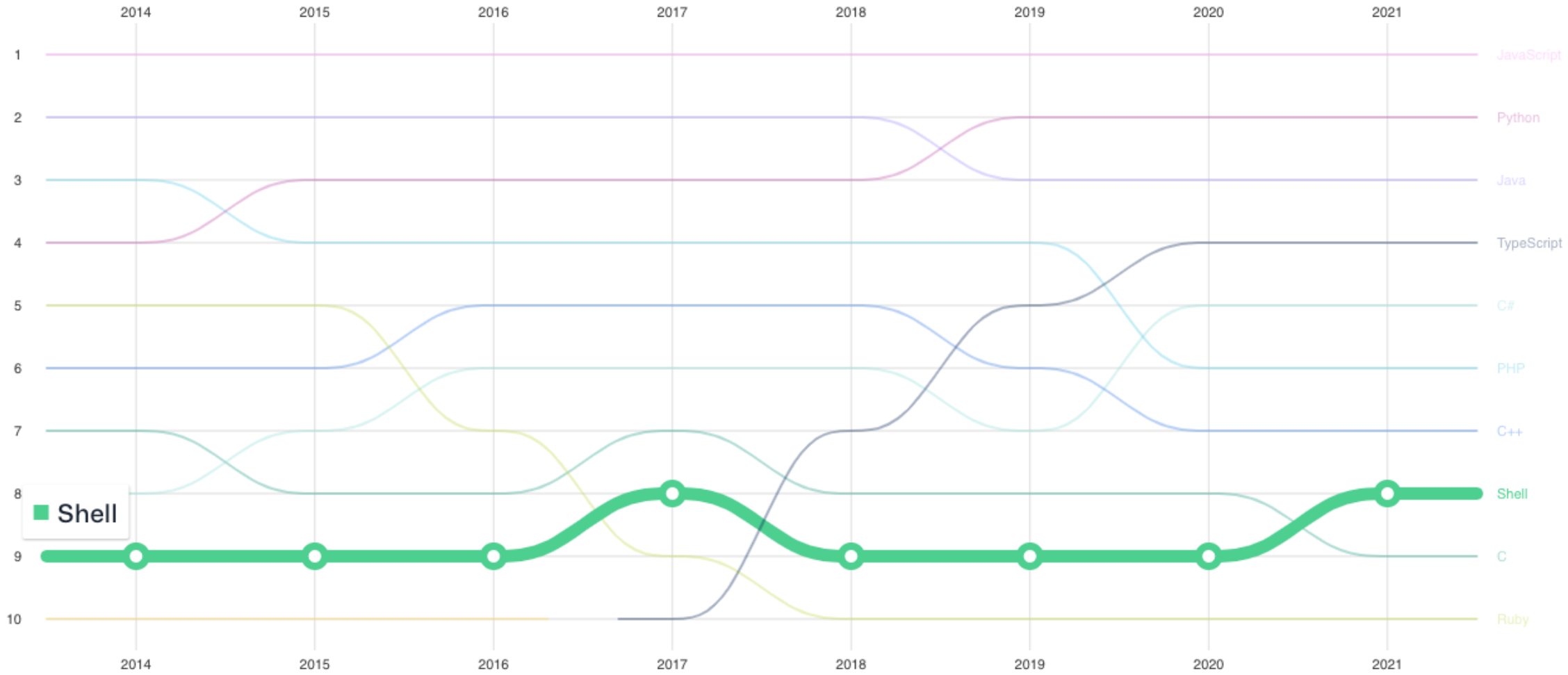
Used by everyo

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

Used by

- D
  -
  -
  -
- Autom
  - Co
  - Insta

. . . for real



from the 2021 state of the octoverse: https://octoverse.github.com

Why? … well, the shell is great

- Universal Composition
  - Composing arbitrary commands using files and pipes
  - Allows users to create powerful but succinct scripts
- Unix native
  - It is well suited to the Unix abstractions (files, strings, etc)
  - Offers great control and management of the file system
- Interactive
  - The complete system environment is accessible
  - Short commands and flags allows for quick experimentation

An example: Temperature Analysis

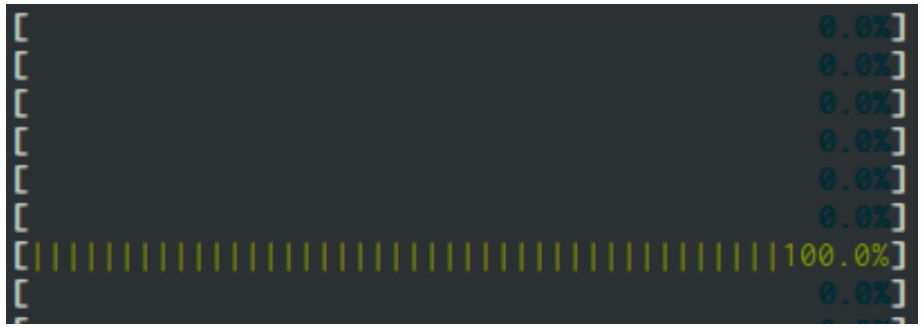
- This script computes the max temp in the US for the years 2015-2019
- To do so it:
  - Preprocessing
    - Fetches the indexes of temperature data archives
    - Downloads the archived temp data
    - Extracts the raw data
  - Processing
    - Cleans it
    - Computes the maximum

```
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
```

- The preprocessing part is taken from the Hadoop book
  - Until the gunzip
- The final two lines replace the MapReduce program from Hadoop book
  - The MapReduce equivalent in Java is 150 lines of code :')

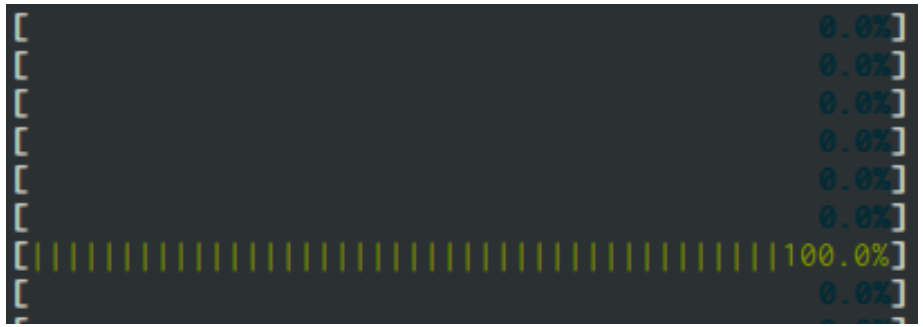
The shell is great but …

Shell scripts are mostly sequential!*



*Actually they have a ton more issues but we will come to that in the end

The shell is great but …



Shell scripts are mostly sequential!*

Parallelizing requires a lot of manual effort:

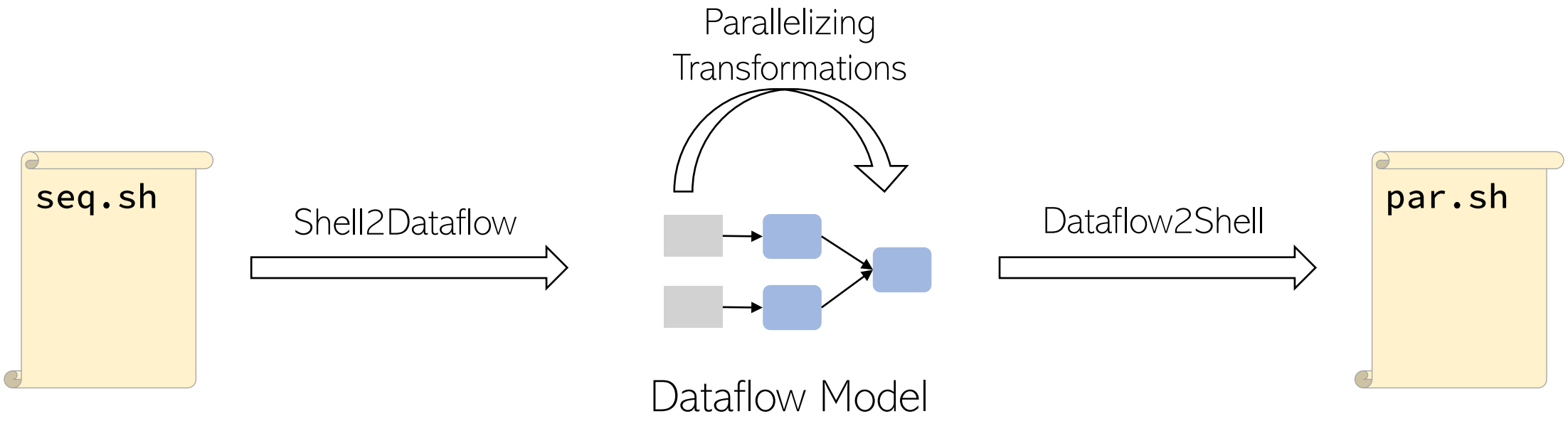
- Using specific command flags (e.g., sort -p, make -jN)
- Using parallelization tools (e.g., GNU parallel)
- Rewriting script in parallel languages (e.g. Erlang)

What did we do to deserve this??? :'(

*Actually they have a ton more issues but we will come to that in the end

PaSh

PaSh



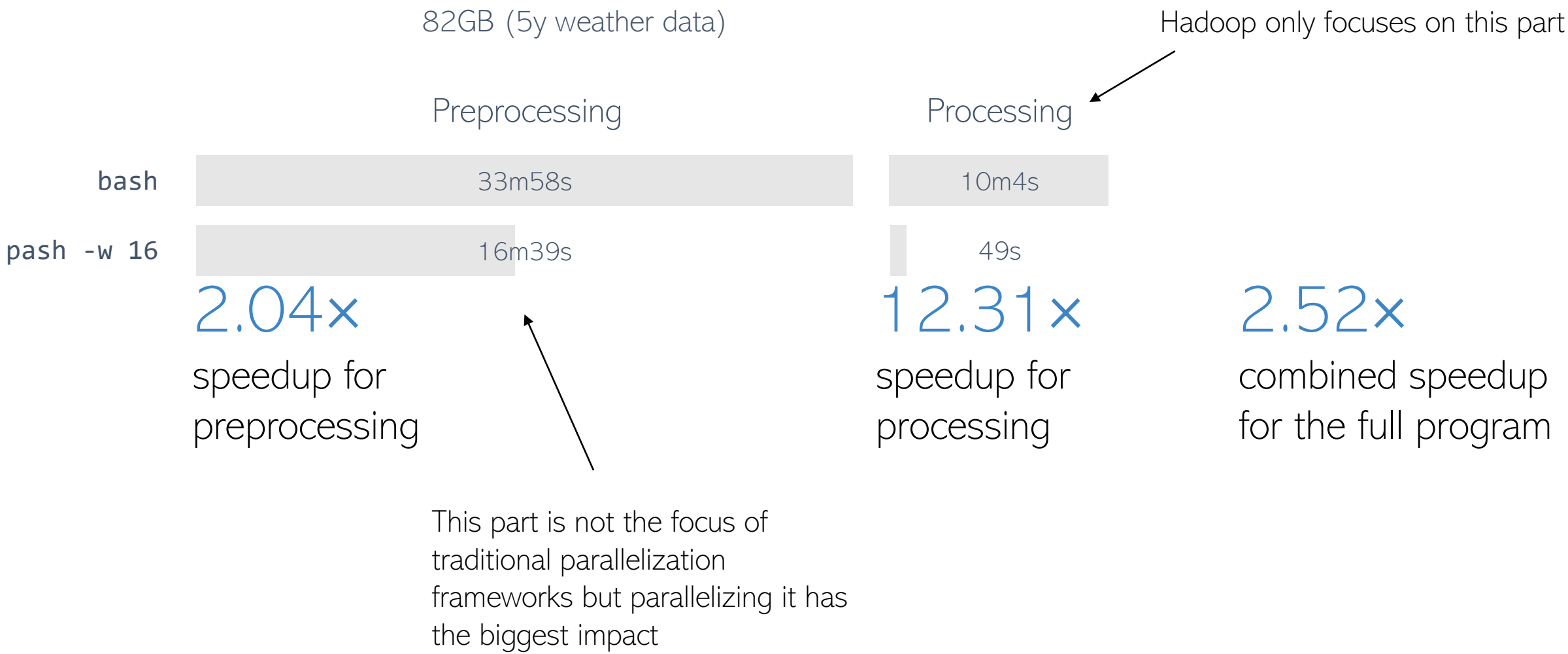
No tight coupling: Could work on top of any shell!

PaSh on Temperature Analysis

82GB (5y weather data)

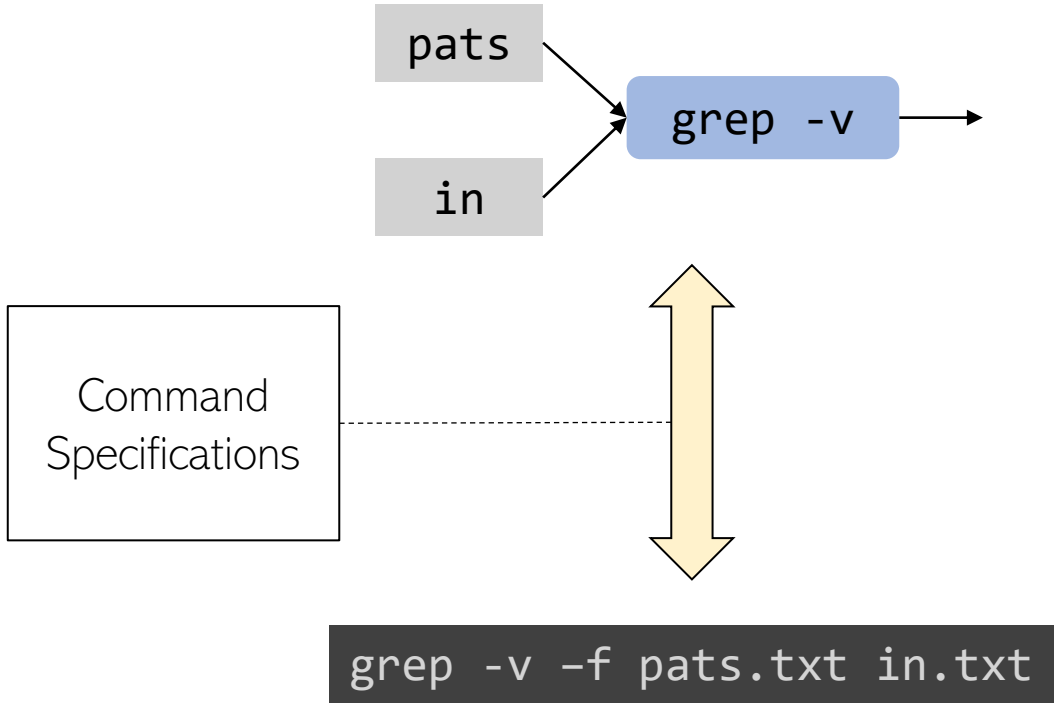
	Preprocessing	Processing
bash	33m58s	10m4s

PaSh on Temperature Analysis



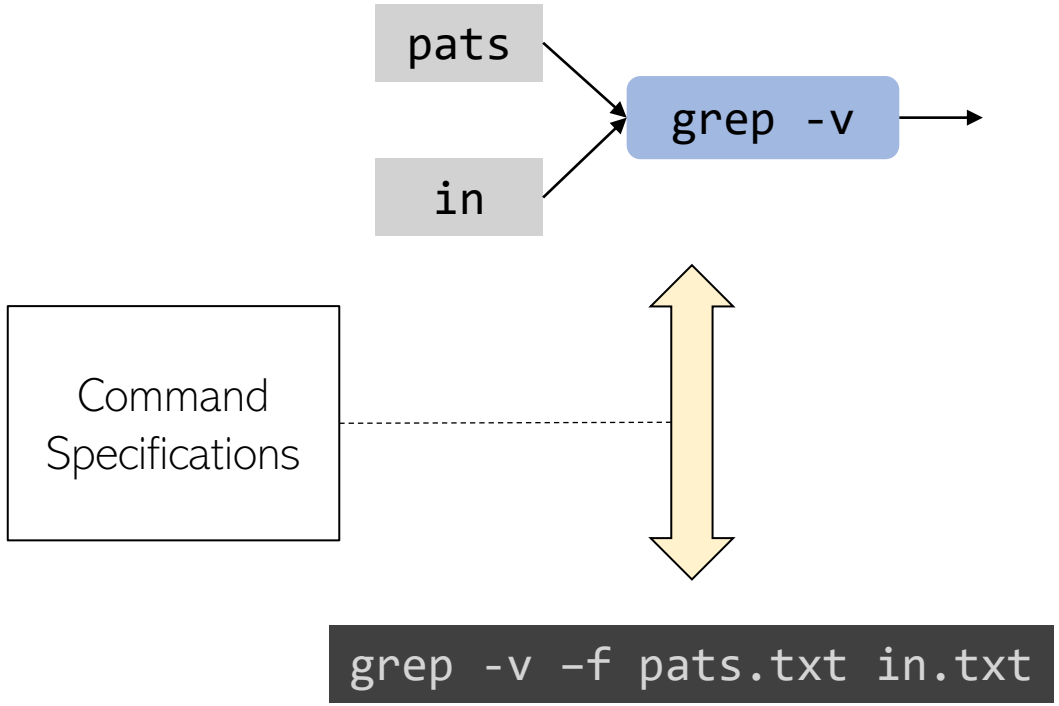
PaSh Insights

Command Specification Framework



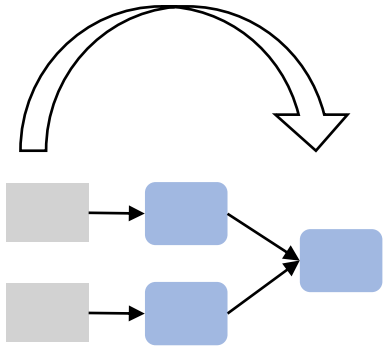
PaSh Insights

Command Specification Framework



Formalized Order Aware Dataflow Model

Parallelizing Transformations



Transformations proven correct

Read the PaSh papers at EuroSys 21 and ICFP 21 for more!

PaSh -- The static way

That should be OK, right?

That should be OK, right?

Conservative or unsound – Choose one

- The shell is dynamic:
  - Current directory
  - Environment variables
  - Unexpanded strings
  - File system

```
IN=${IN:-$TOP/pg}
mkdir $IN
cd $IN
echo 'Downloading, be patient...'
wget $SOURCE/data/pg.tar.xz
if [ $? -ne 0 ]; then
  echo "Download failed!"
  exit 1
fi
cat pg.tar.xz | tar -xJ

cd $TOP
OUT=${OUT:-$TOP/output}
mkdir -p "$OUT"
for input in $(ls ${IN}); do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```

Conservative or unsound – Choose one

- The shell is dynamic:
  - Current directory
  - Environment variables
  - Unexpanded strings
  - File system

- Static parallelization has to choose:
  - Sound but **conservative**
  - **Unsound** and optimistic

```
IN=${IN:-$TOP/pg}
mkdir $IN
cd $IN
echo 'Downloading, be patient...'
wget $SOURCE/data/pg.tar.xz
if [ $? -ne 0 ]; then
  echo "Download failed!"
  exit 1
fi
cat pg.tar.xz | tar -xJ

cd $TOP
OUT=${OUT:-$TOP/output}
mkdir -p "$OUT"
for input in $(ls ${IN}); do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```

PaSh-JIT

Just in time parallelization

- PaSh-JIT tries to parallelize as-late-as-possible™
- Provides critical information to the compiler:
  - State of shell, Variables, Directory, Files
- Not only correct, but also **faster!!!**
- How?
  - By constantly switching between evaluation and parallelization

Just in time parallelization

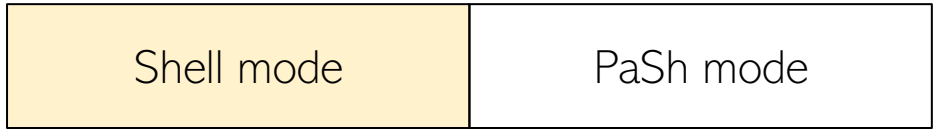
```
OUT=${OUT:-$TOP/out}
for input in $(ls ${IN}); do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```

Shell mode	PaSh mode

```
TOP=/pash
IN=/pash/in
```

Just in time parallelization

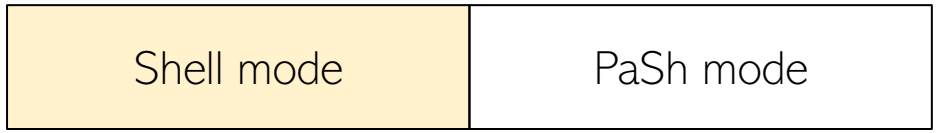
```
OUT=${OUT:-$TOP/out}
for input in $(ls ${IN}); do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```




```
TOP=/pash
IN=/pash/in
```

Just in time parallelization

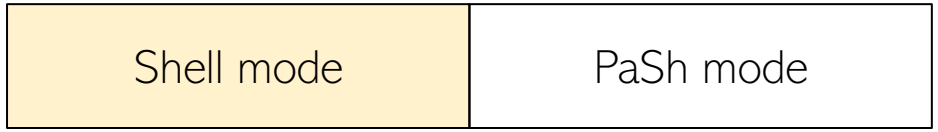
```
OUT=/pash/out
for input in $(ls ${IN}); do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```




```
TOP=/pash
IN=/pash/in
```

Just in time parallelization

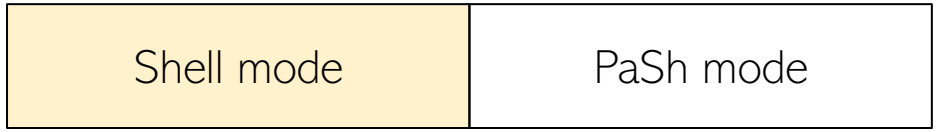
```
OUT=/pash/out
for input in $(ls ${IN}); do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```




```
TOP=/pash
IN=/pash/in
OUT=/pash/out
```

Just in time parallelization

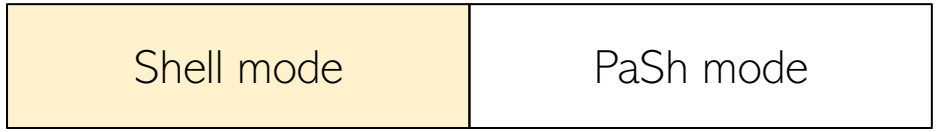
```
OUT=/pash/out
for input in in1 in2; do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```




```
TOP=/pash
IN=/pash/in
OUT=/pash/out
```

Just in time parallelization

```
OUT=/pash/out
for input in in1 in2; do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```




```
TOP=/pash
IN=/pash/in
OUT=/pash/out
input=in1
```

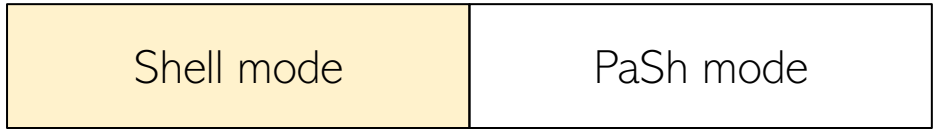
Just in time parallelization

```
OUT=/pash/out
for input in in1 in2; do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```

Shell mode	PaSh mode
TOP=/pash IN=/pash/in OUT=/pash/out input=in1	Expanding

Just in time parallelization

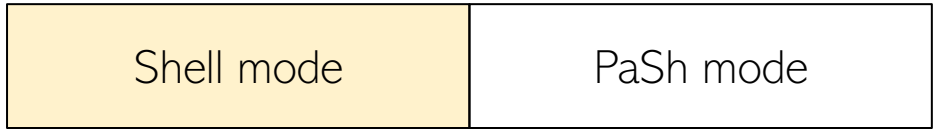
```
OUT=/pash/out
for input in in1 in2; do
  cat "$IN/$input" |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > "${OUT}/${input}.out"
done
```




```
TOP=/pash
IN=/pash/in
OUT=/pash/out
input=in1
```

Just in time parallelization

```
OUT=/pash/out
for input in in1 in2; do
  cat /pash/in/in1 |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > /pash/out/in1.out
done
```




```
TOP=/pash
IN=/pash/in
OUT=/pash/out
input=in1
```

Just in time parallelization

```
OUT=/pash/out
for input in in1 in2; do
  cat /pash/in/in1 |
    tr -sc '[A-Z][a-z]' '[\012*]’ |
    sort > /pash/out/in1.out
done
```

Shell mode	PaSh mode

`TOP=/pash`
`IN=/pash/in`
`OUT=/pash/out`
`input=in1`
	Parallelize?

Just in time parallelization

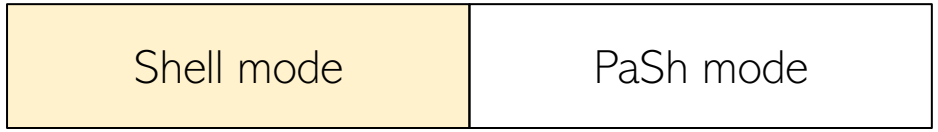
```
OUT=/pash/out
for input in in1 in2; do
  mkfifo f1 f2 f3 f4
  cat /pash/in/in1 | split f1 f2 &
  ... &
  sort < f1 > f3 &
  sort < f3 > f4 &
  sort –m f3 f4 > /pash/out/in1.out
  rm f1 f2 f3 f4
done
```

Shell mode	PaSh mode

`TOP=/pash`
`IN=/pash/in`
`OUT=/pash/out`
`input=in1`
	Parallelize?
	Success!

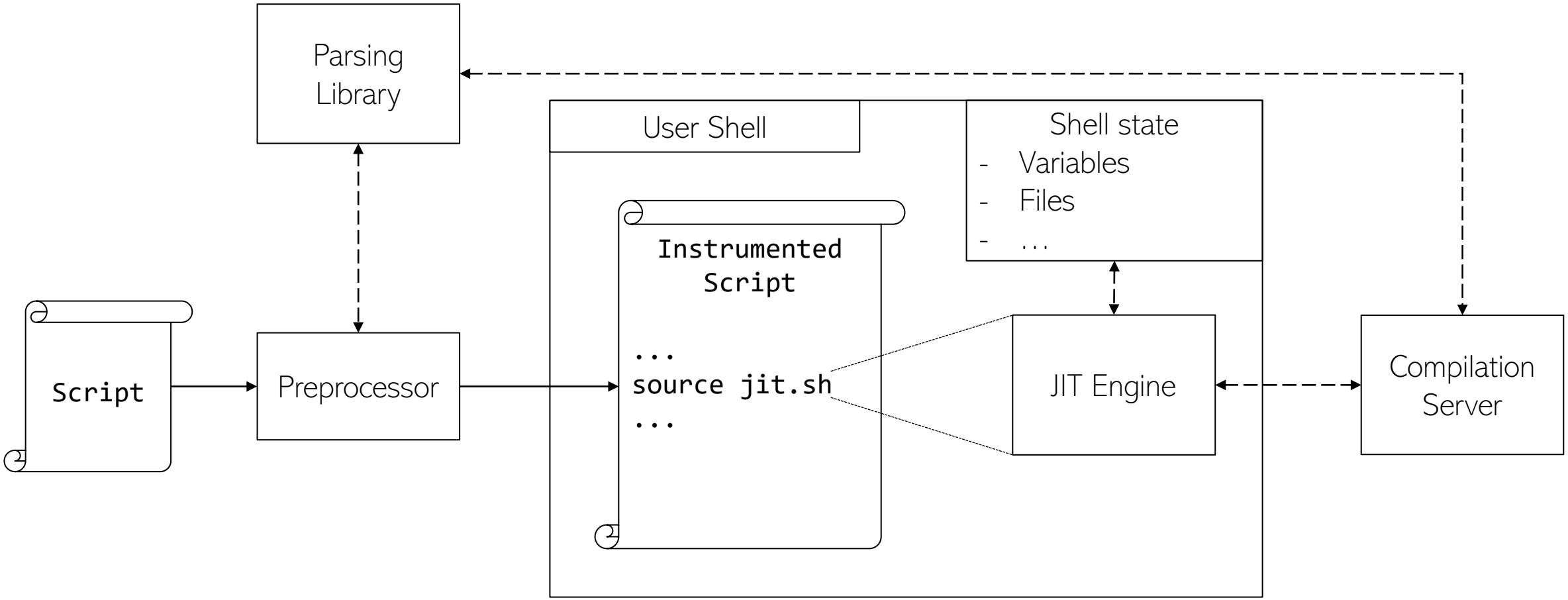
Just in time parallelization

```
OUT=/pash/out
for input in in1 in2; do
  mkfifo f1 f2 f3 f4
  cat /pash/in/in1 | split f1 f2 &
  ... &
  sort < f1 > f3 &
  sort < f3 > f4 &
  sort –m f3 f4 > /pash/out/in1.out
  rm f1 f2 f3 f4
done
```

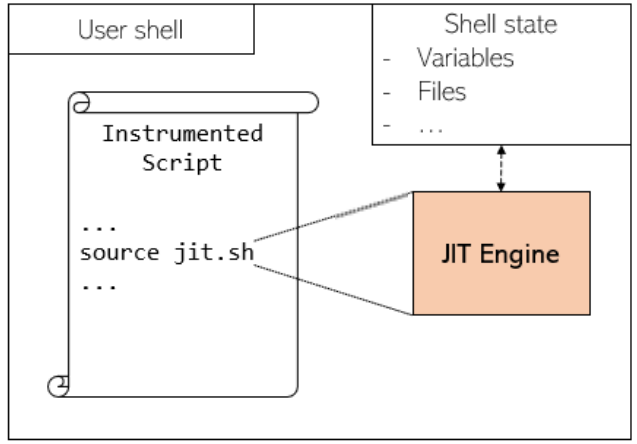



```
TOP=/pash
IN=/pash/in
OUT=/pash/out
input=in1
```

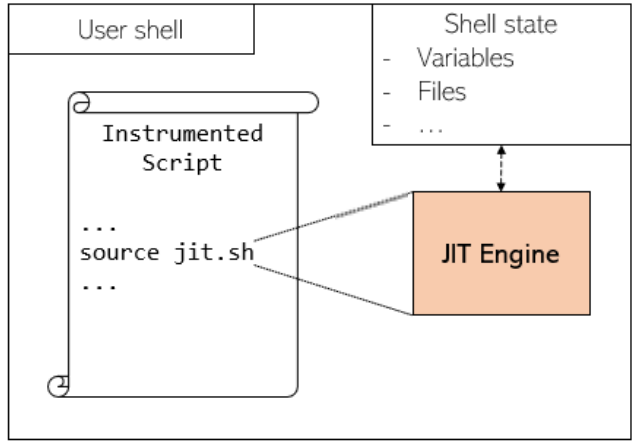
PaSh-JIT overview



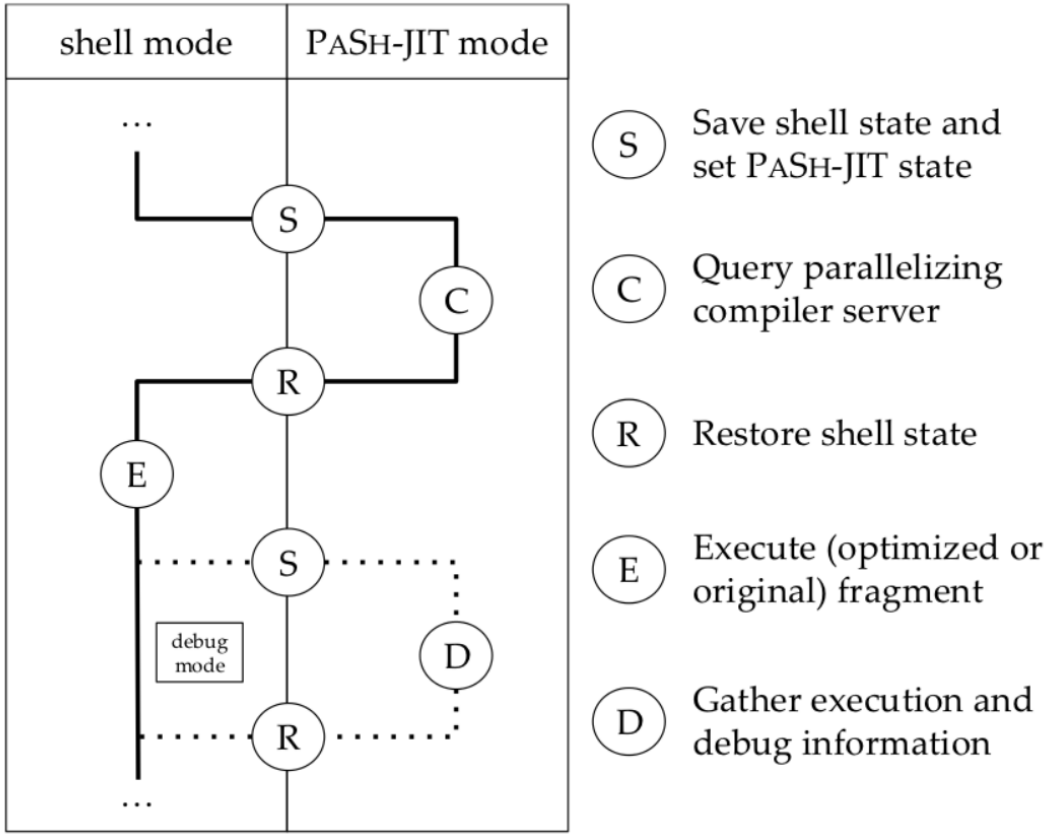
JIT Engine


User shell
Shell state
- Variables
- Files
- ...
Instrumented
Script
...
source jit.sh
...
JIT Engine

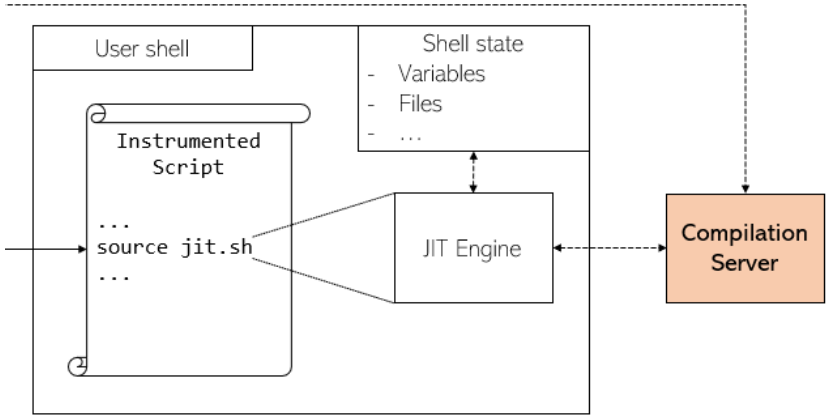
JIT Engine



Hides compilation from the perspective of the shell



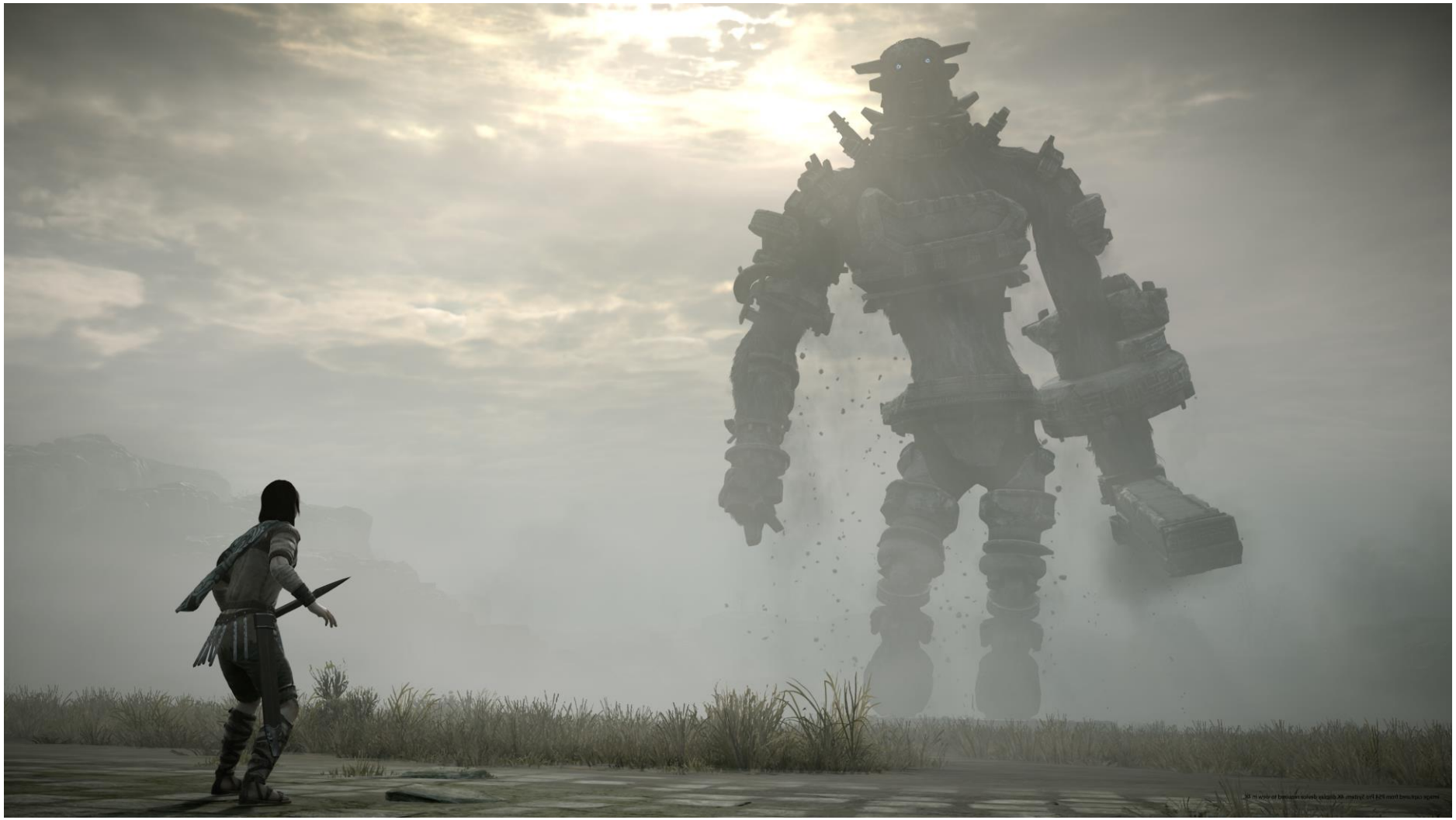
Compilation Server



- The compilation server reduces latency!
  - Doesn't require initializations and keeps state in memory
  - Necessary for feasibility in practice (in tight loops every ms counts)
- Also enables additional optimizations
  - Parallelization of independent fragments (e.g., iterations that touch different files)
  - Profile-guided optimizations (e.g., configuring parallelization width)
- For more, check out our paper

Evaluation

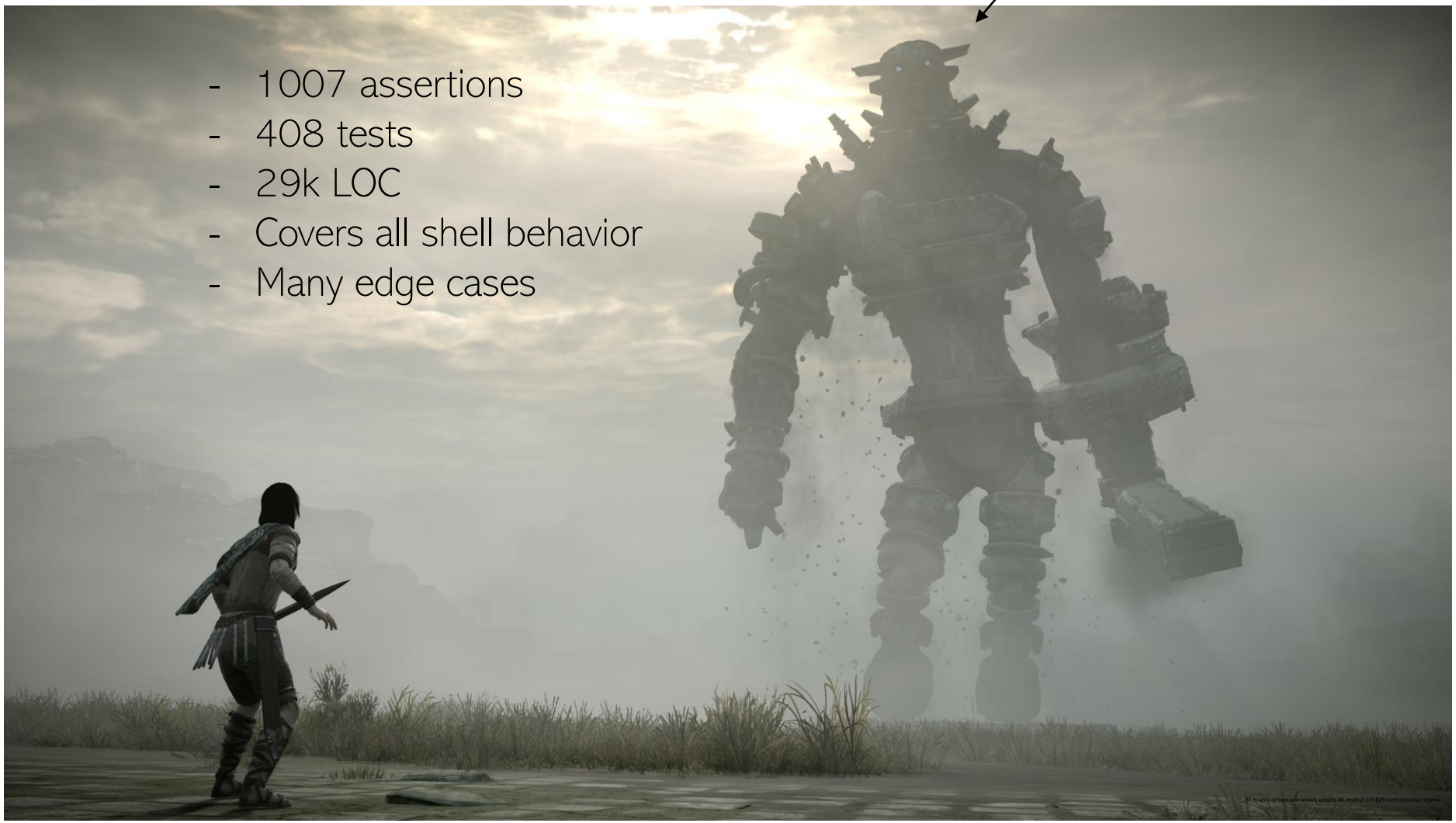
Evaluation: Correctness



Evaluation: Correctness

Evaluation: Correctness

POSIX Shell Test Suite



Evaluation: POSIX test suite

- Out of the 408 tests
  - Bash passes 376 and fails 32 tests
  - PaSh-JIT passes 374 and fails 34 tests
- Divergence in these two tests is only in the exit status
  - Both return with an error, though different code

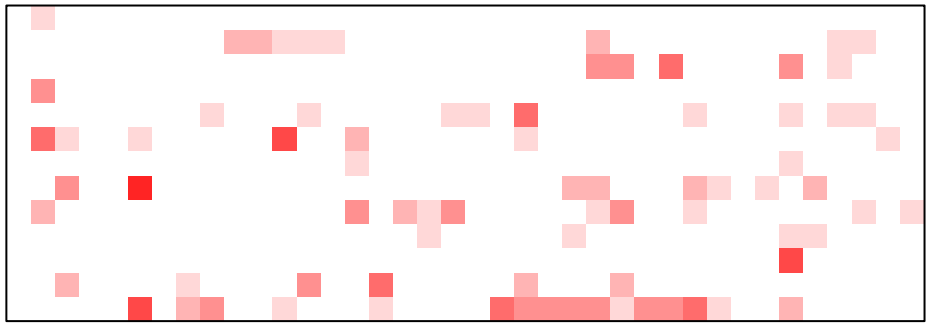
Evaluation: POSIX test suite

- Out of the 408 tests
  - Bash passes 376 and fails 32 tests
  - PaSh-JIT passes 374 and fails 34 tests
- Divergence in these two tests is only in the exit status
  - Both return with an error, though different code
- Other shells compared to bash:

	Both Bash and X fail	Bash succeeds X fails
dash	1	20
ksh	2	22
mksh	1	29
posh	4	52
yash	1	20

Evaluation: POSIX test suite

- Out of the 408 tests
  - Bash passes 376 and fails 32 tests
  - PaSh-JIT passes 374 and fails 34 tests
- Divergence in these two tests is only in the exit status
  - Both return with an error, though different code
- Other shells compared to bash:
- Various shell failures on POSIX tests:

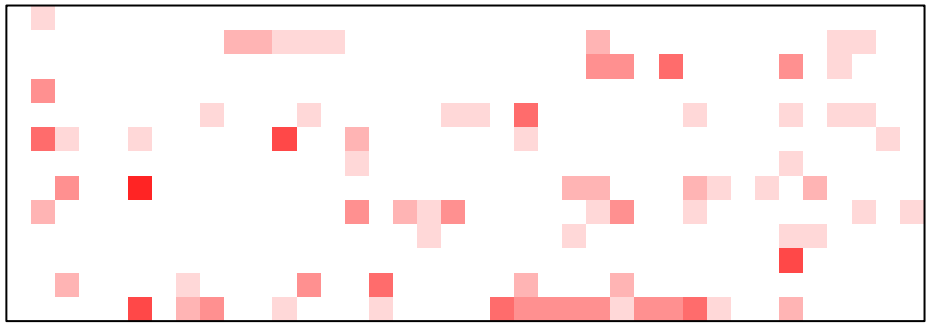


	Both Bash and X fail	Bash succeeds X fails
dash	1	20
ksh	2	22
mksh	1	29
posh	4	52
yash	1	20

Evaluation: POSIX test suite

- Out of the 408 tests
  - Bash passes 376 and fails 32 tests
  - PaSh-JIT passes 374 and fails 34 tests
- Divergence in these two tests is only in the exit status
  - Both return with an error, though different code
- Other shells compared to bash:
- Various shell failures on POSIX tests:

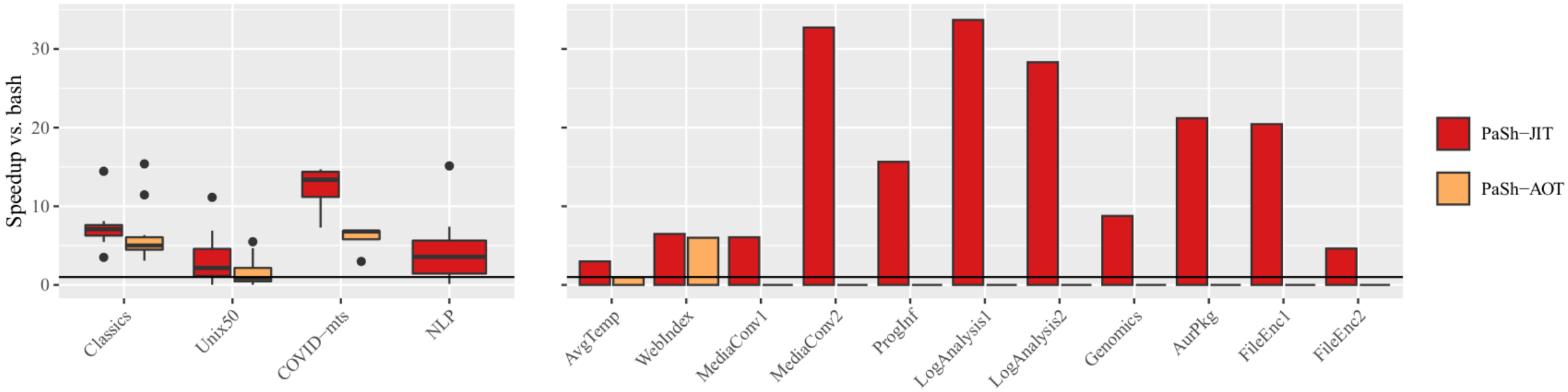
By following a lightweight shim approach (instead of reimplementing) we achieve very high compatibility with bash ✨



	Both Bash and X fail	Bash succeeds X fails
dash	1	20
ksh	2	22
mksh	1	29
posh	4	52
yash	1	20

Evaluation: Performance

- Evaluating on 82 shell scripts (4 suites and 11 standalone scripts)



Avg speedups: PaSh-JIT (x5.8) – PaSh-AOT (x2.9)

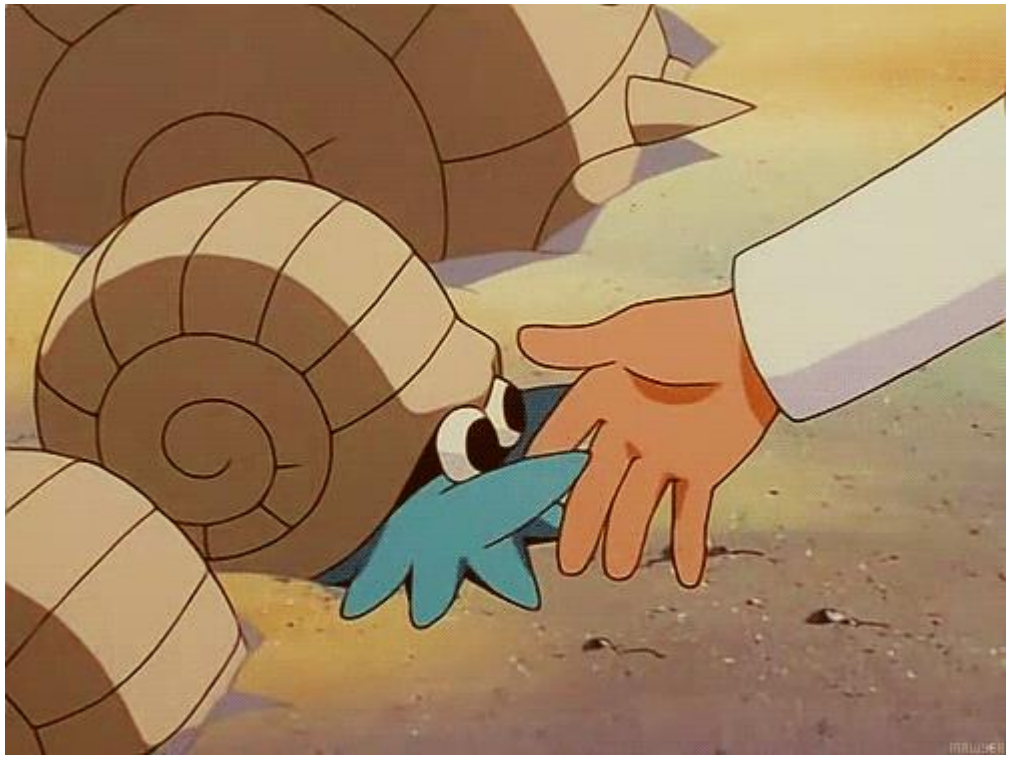
Conclusion

Conclusion

- Shells were angry that we tried to parallelize statically

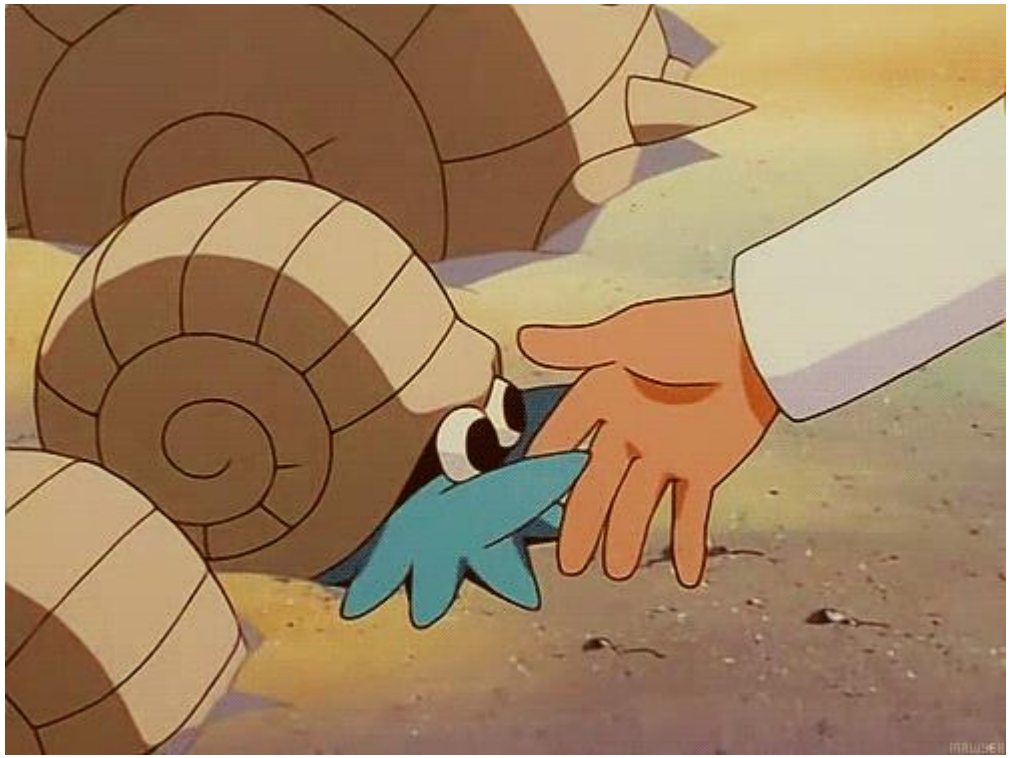
Conclusion

- Shells were angry that we tried to parallelize statically
- We can make them happy by being dynamic



Conclusion

- Shells were angry that we tried to parallelize statically
- We can make them happy by being dynamic
- Correct
- And fast!



The shell has more problems…

- Error-proneness
  - accidentally `rm -rf /` 
- Hard to learn
  - still googling for if-then-else shell syntax
- Redundant recomputation
  - we have to use Makefiles etc
- Lack of support for contemporary deployments
  - managing a distributed cluster

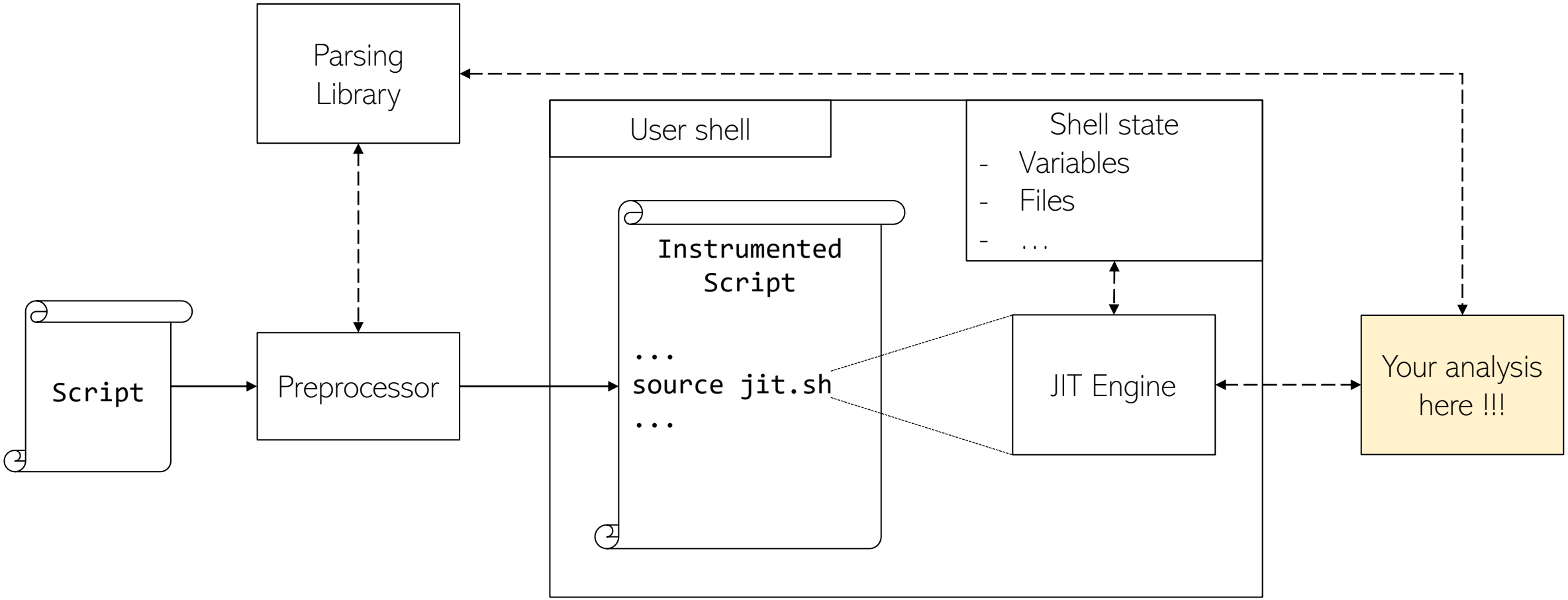
Recent exceptions: Rattle [1] and Riker [2]

[1] Sarah Spall, Neil Mitchell, and Sam Tobin-Hochstadt. “Build scripts with Perfect Dependencies.“OOPSLA. 2020.

[2] Charlie Curtsinger, and Daniel W. Barowy. “Riker: Always-Correct and Fast Incremental Builds from Simple Specifications “ATC. 2022.

The JIT part of PaSh-JIT is an enabler

- The JIT structure of PaSh-JIT enables additional analyses/solutions

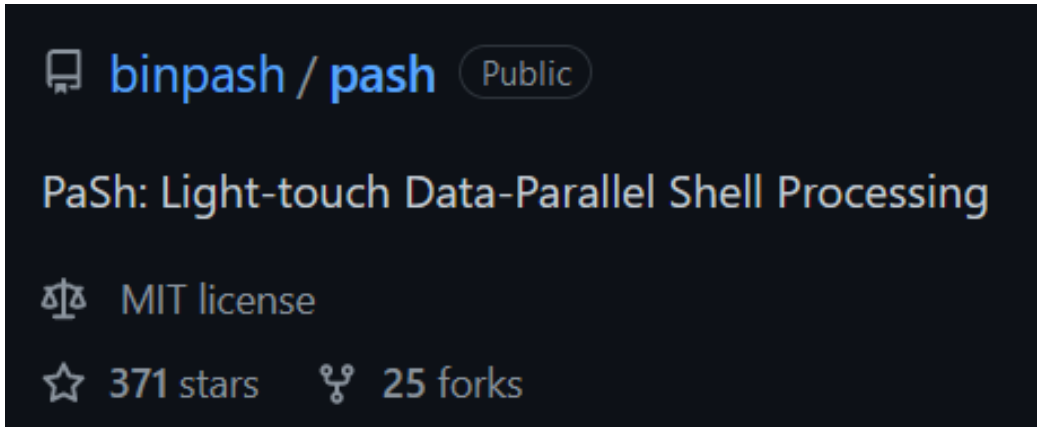


Some exciting future directions

- A shell monitor that ensures that safety/security props are not violated
- A fully distributed shell 
- An incremental execution shell
- Talk to us if you have ideas!
- Michael and Nikos are hiring!

Practical impact and availability

- PaSh is open source and hosted by the Linux Foundation



- It is virtually indistinguishable from bash (406/408 POSIX tests)
  - And requires no modifications/reimplementation
- OSDI artifact badges

- Download it and play binpa.sh github.com/binpash/pash