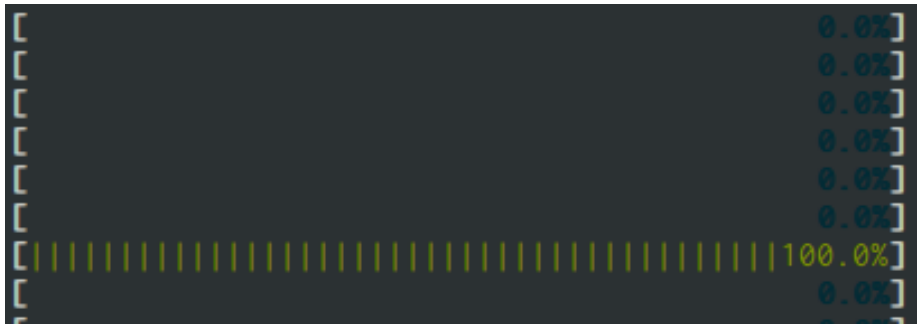
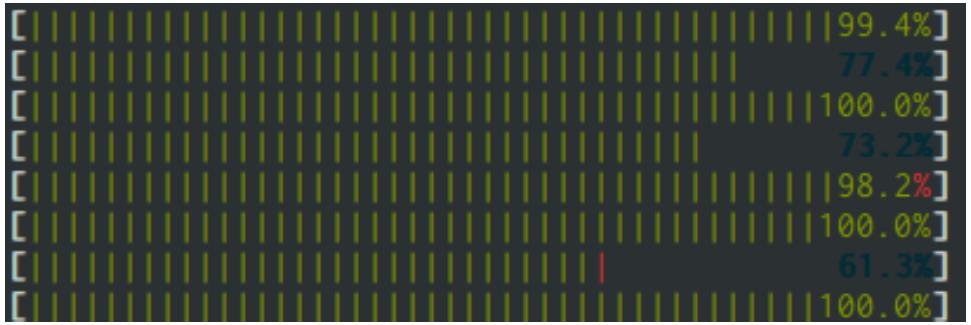
PaSh: Light-touch Parallelization for Shell Scripts

or how to get from this:



to this:



binpa.sh github.com/binpash/pash

Joint work with:

And many others (in alphabetical order):

Nikos Vasilakis

Michael Greenberg

Achilles Benetopoulos

Jan Bielak

Lazar Cvetkovic

Thurston Dang

Shivam Handa

Dimitris Karnikis

Kostas Mamouras

Tammam Mustafa

Radha Patel

Martin Rinard

Just-in-Time

PaSh: Light-touch Parallelization for Shell Scripts

shell

Usec

```
loop as the one in execute_evaluation_scripts
_config in "${microbenchmark_configs[@]}"; do
ead -r -a flags <<< "${microbenchmark_config}"
robenchmark=${flags[0]}
echo "Executing test: $microbenchmark"
# Execute the sequential script on the first run only
exec_seq="-s"
for n_in in "${n_inputs[@]}"; do
    echo "Number of inputs: ${n_in}"

    ## Generate the intermediary script
    python3 generate_microbenchmark_intermediary_scripts.py \
        $microbenchmarks_dir $microbenchmark $n_in $intermediary_dir "test"
    for flag in "${flags[@]:1}"; do
        echo "Flag: ${flag}"

        ## Execute the intermediary script
        ./execute_compile_evaluation_script.sh $assert_correctness $exec_seq $flag \
            "${microbenchmark}" "${n_in}" "test_results" "test_" > /dev/null 2>&1
        rm -f /tmp/eager*

        ## Only run the sequential the first time around
        exec_seq=""
    done
done
done
done
```

```
#!/usr/bi
# TODO
# w
p
```

```
execute_tests "" "${script_microbenchmarks[@]}"
execute_tests "-c" "${pipeline_microbenchmarks[@]}"

echo "Below follow the identical outputs:"
grep --files-with-match "are identical" ../evaluation/results/test_result

echo "Below follow the non-identical outputs:"
grep -L "are identical" ../evaluation/results/test_res

TOTAL_TESTS=$(ls -la ../evaluation/results
PASSED_TESTS=$(grep --files-with-ma
echo "Summary: ${PASSED_TESTS
```

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: Max Temp

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

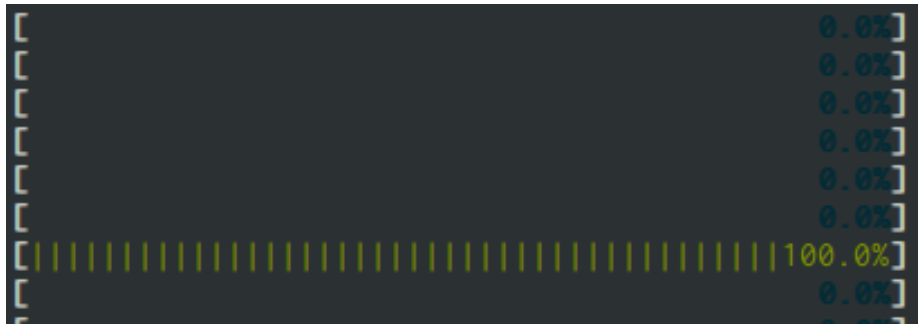
Preprocessing

Processing

```
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!*

Parallelizing requires a lot of manual effort:

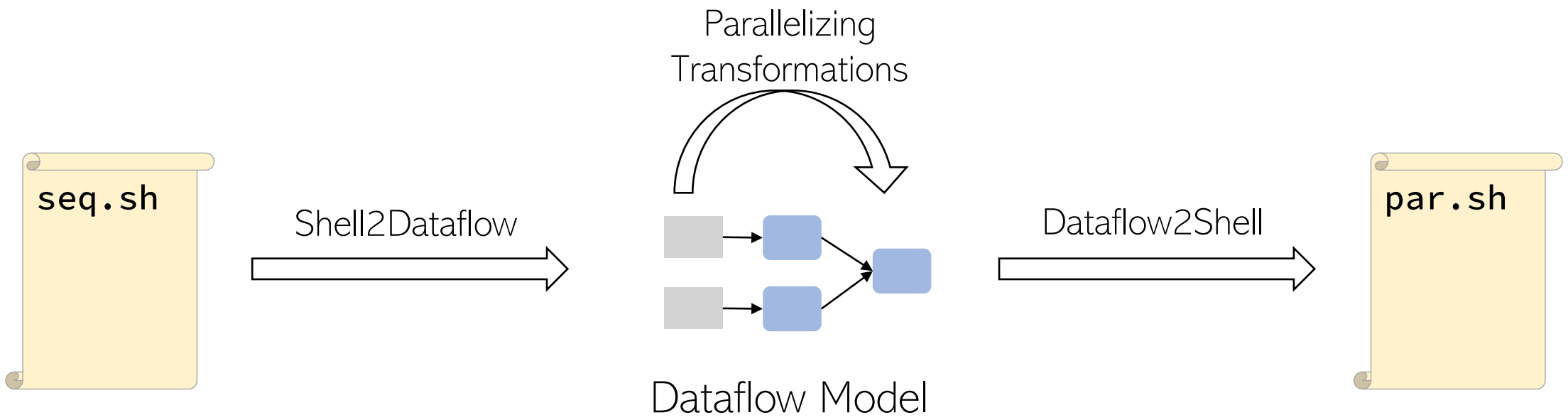
- Using specific command flags (e.g., sort -p, make -jN)
- Using parallelization tools (e.g., GNU parallel)
- Rewriting script in parallel languages (e.g. Erlang)

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 Max Temp script

82GB (5y weather data)

Hadoop only focuses on this part

	Preprocessing	Processing
`bash`	33m58s	10m4s
`pash -w 16`	16m39s	49s

2.04× speedup for preprocessing

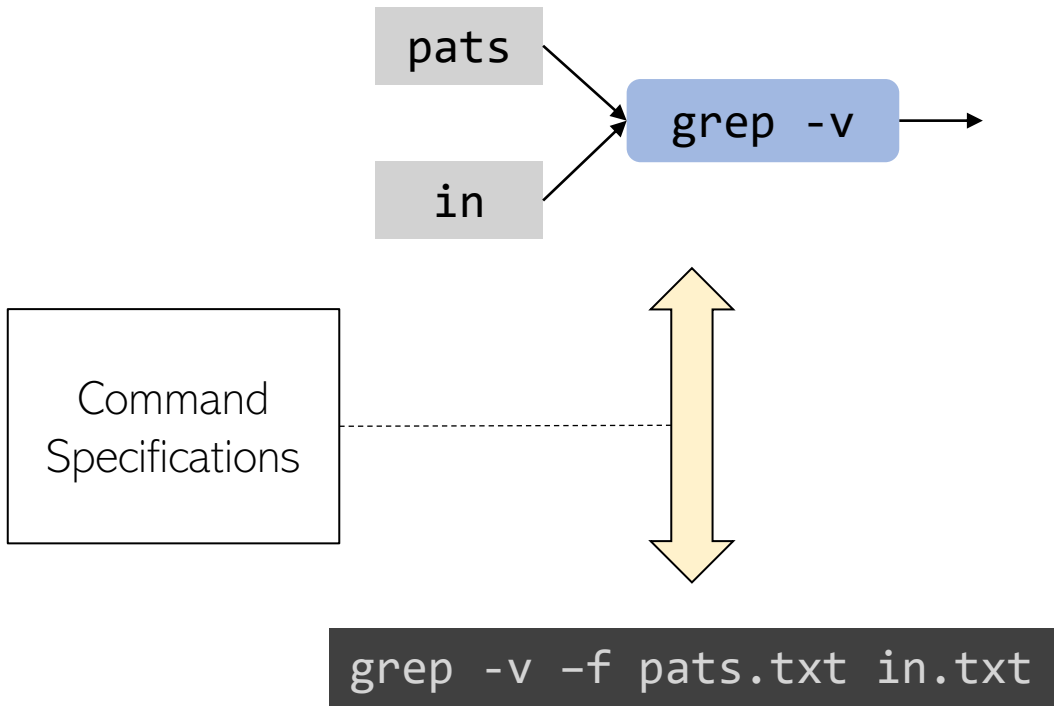
12.31× speedup for preprocessing

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

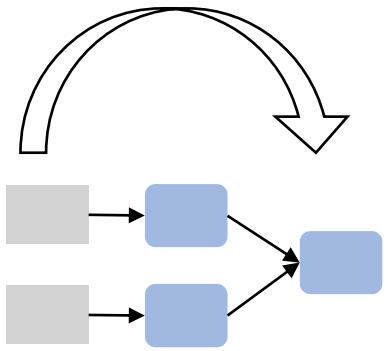
PaSh Insights

Command Specification Framework



Formalized Order Aware Dataflow Model

Parallelizing Transformations



Transformations proven correct

Read our EuroSys 21 and ICFP 21 papers for more!

PaSh -- The static way

That should be OK, right?

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 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
```

Shell mode	PaSh mode

```
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
```

Shell mode	PaSh mode

```
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
```

Shell mode	PaSh mode

```
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
```

Shell mode	PaSh mode

```
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
```

Shell mode	PaSh mode

```
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
```

Shell mode	PaSh mode

```
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
```

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!

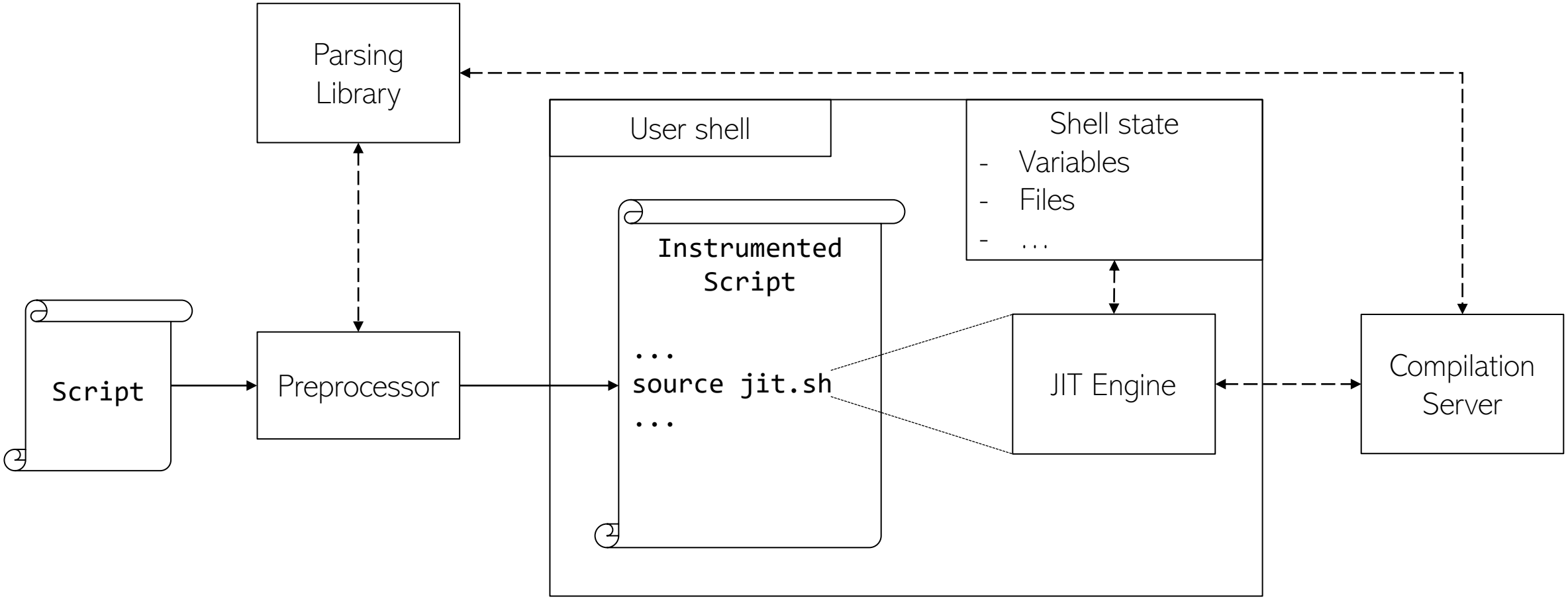
Just in time parallelization

```
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
```

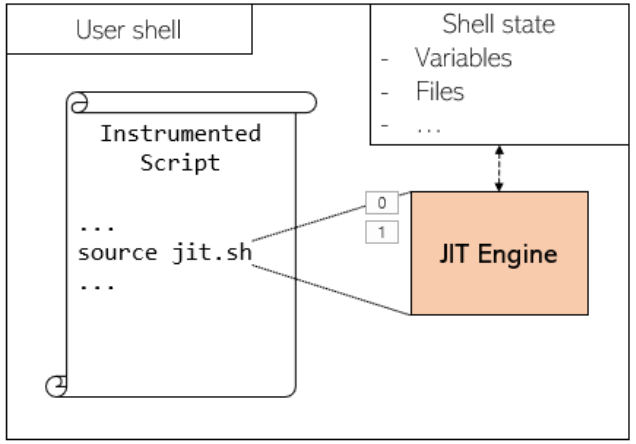
Shell mode	PaSh mode

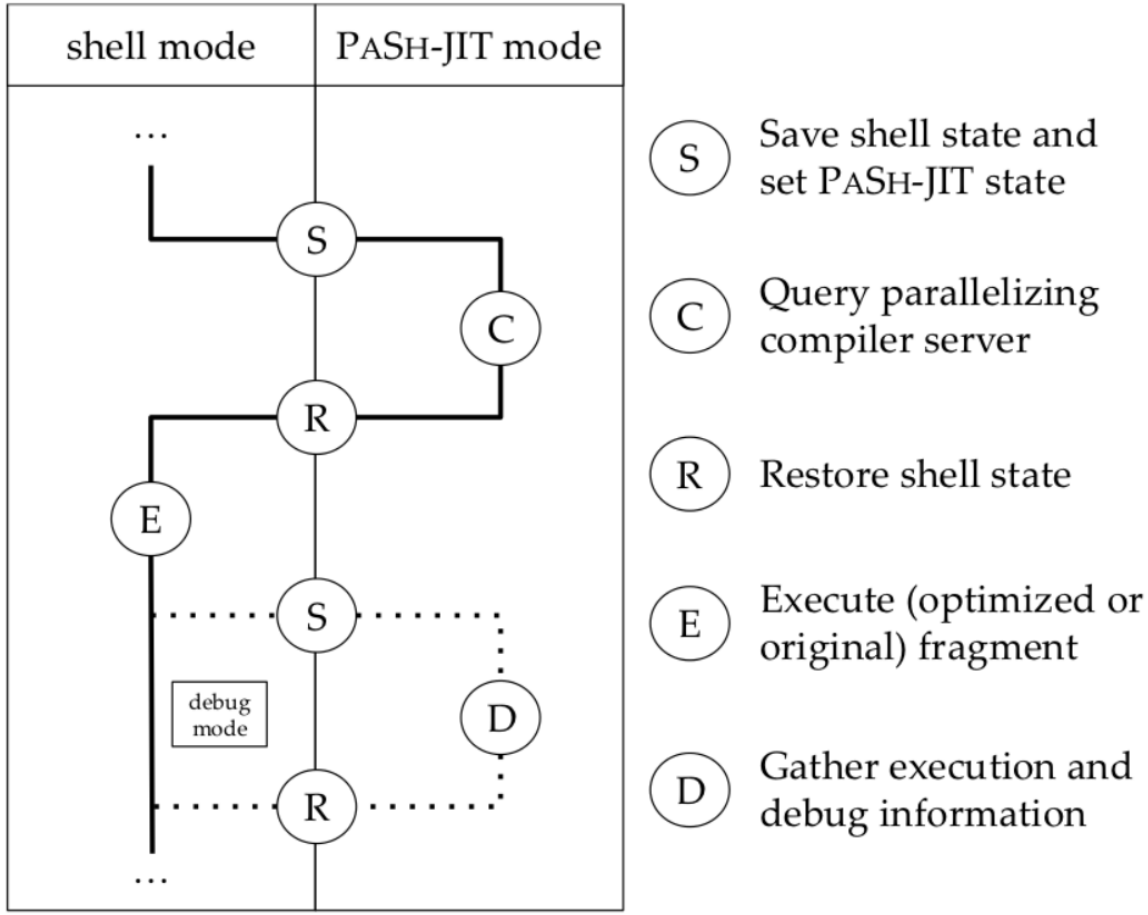
```
TOP=/pash
IN=/pash/in
OUT=/pash/out
input=in1
```

PaSh-JIT overview

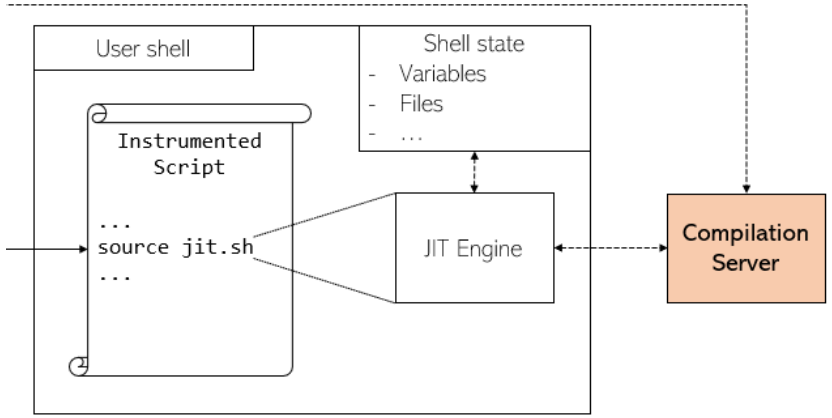


JIT Engine


User shell
Shell state
- Variables
- Files
- ...
Instrumented Script
...
source jit.sh
...
0
1
JIT Engine


shell mode
PASH-JIT mode
...
S
C
R
E
S
debug mode
D
R
...
S Save shell state and set PASH-JIT state
C Query parallelizing compiler server
R Restore shell state
E Execute (optimized or original) fragment
D Gather execution and debug information

Compilation Server

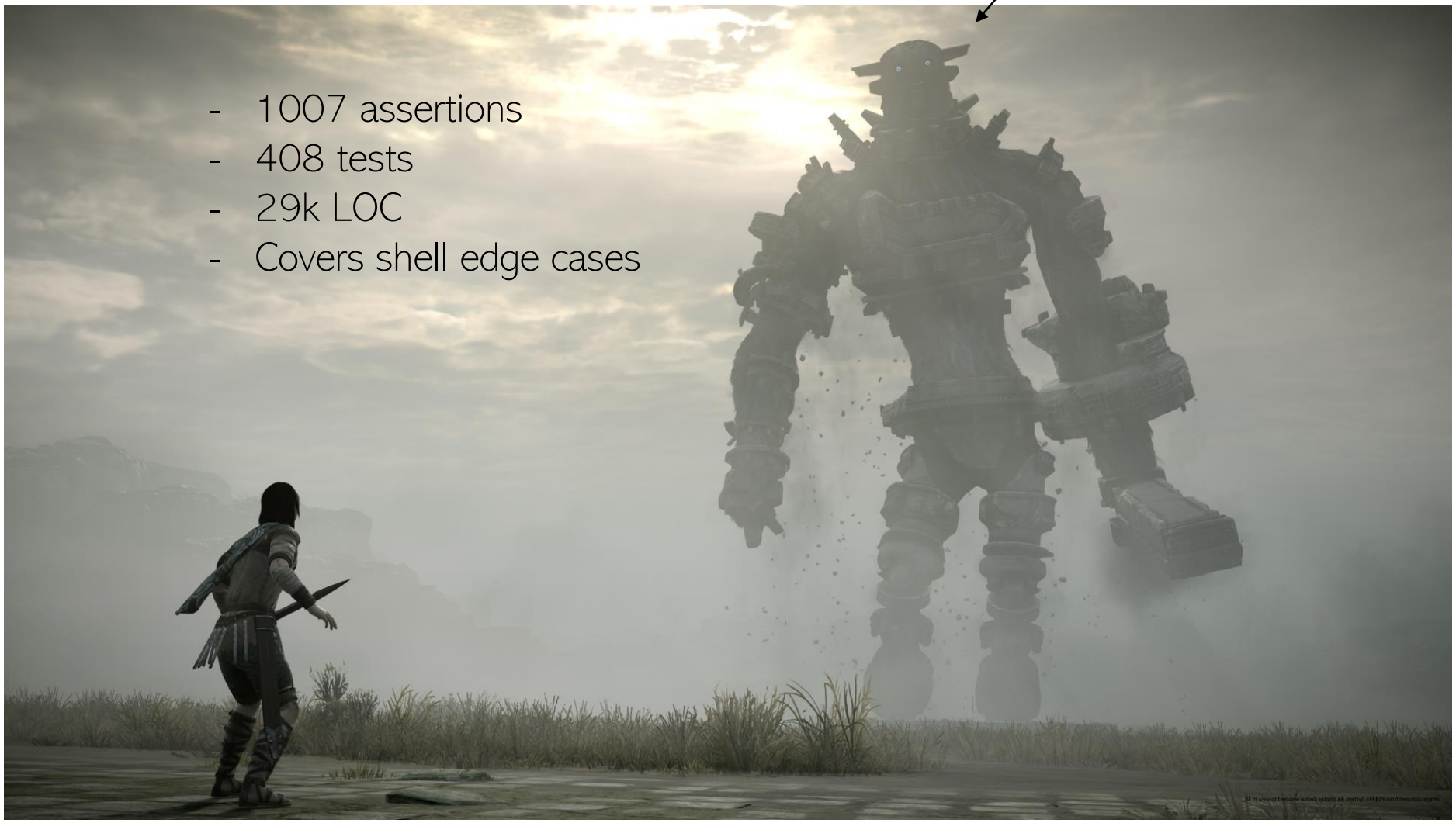


- The compilation server reduces latency!
  - Doesn't require initializations and keeps state in memory
  - Necessary for feasibility in practice (e.g., tight loops)
- Also enables additional optimizations
  - Parallelization of independent fragments (e.g., iterations that touch different files)
  - Profile-guided optimizations (e.g., configuring parallelization width)
- For more, check our OSDI 22 paper

Evaluation

Evaluation: Correctness

POSIX Shell Test Suite



- 1007 assertions
- 408 tests
- 29k LOC
- Covers shell edge cases

Evaluation: POSIX test suite

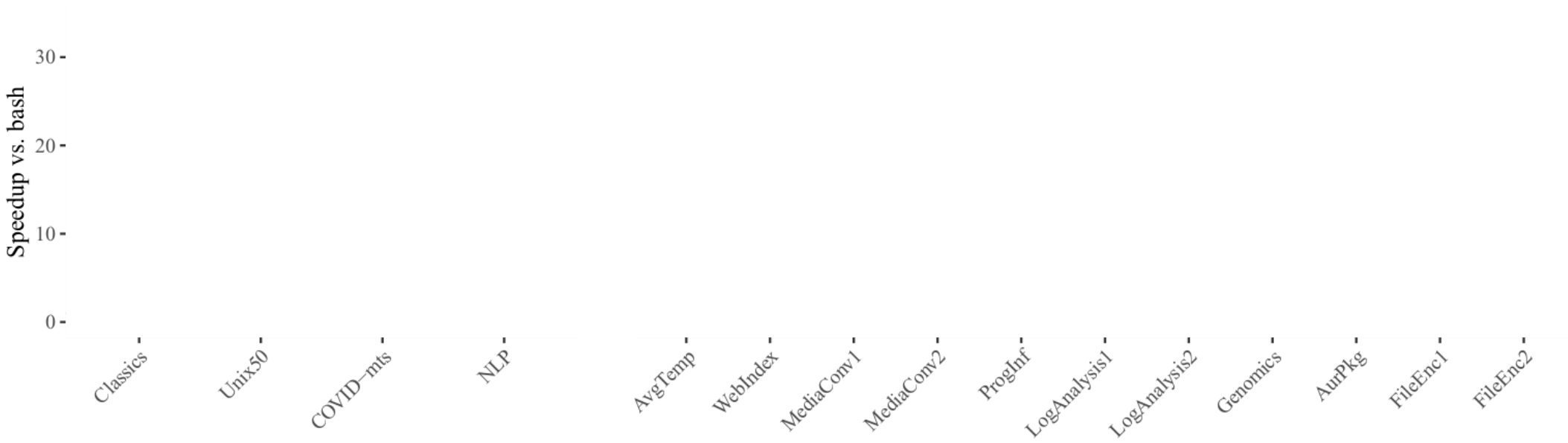
By following a lightweight shim approach (instead of reimplementing) we achieve very high compatibility with bash ✨

- 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:

	Bash succeeds X fails	Bash fails X succeeds
dash	20	3
ksh	20	2

Evaluation: Performance

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

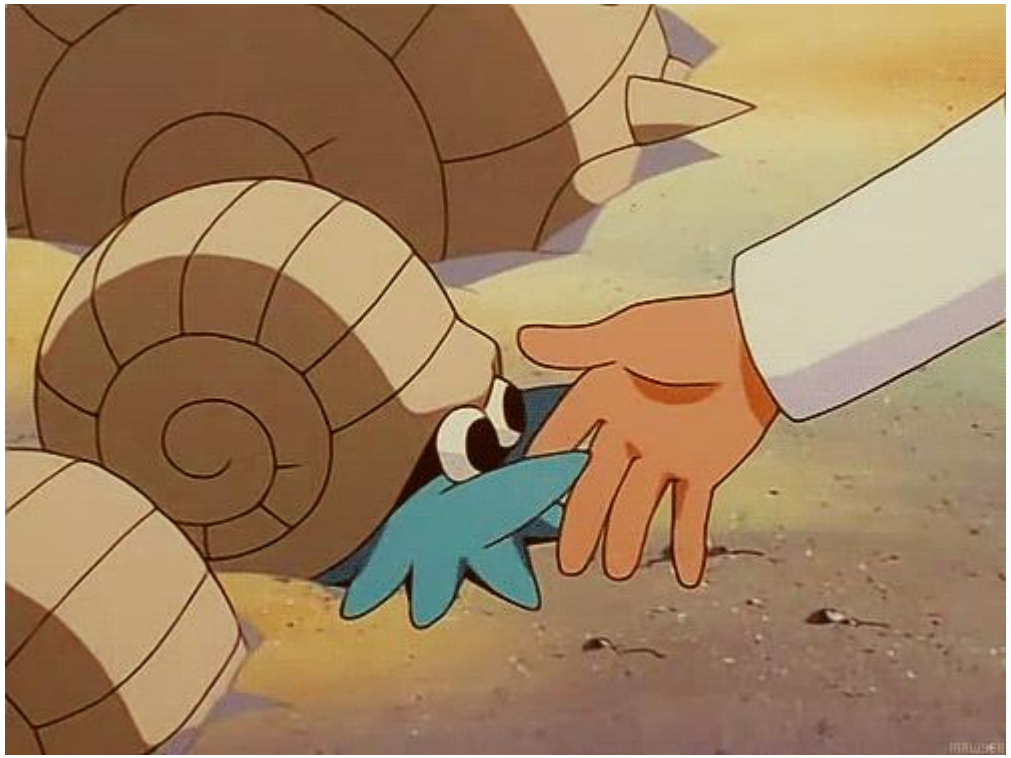


Avg speedups: PaSh-JIT (x5.8) – PaSh-AOT (x2.9)

Conclusion

Conclusion

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



The shell has more problems…

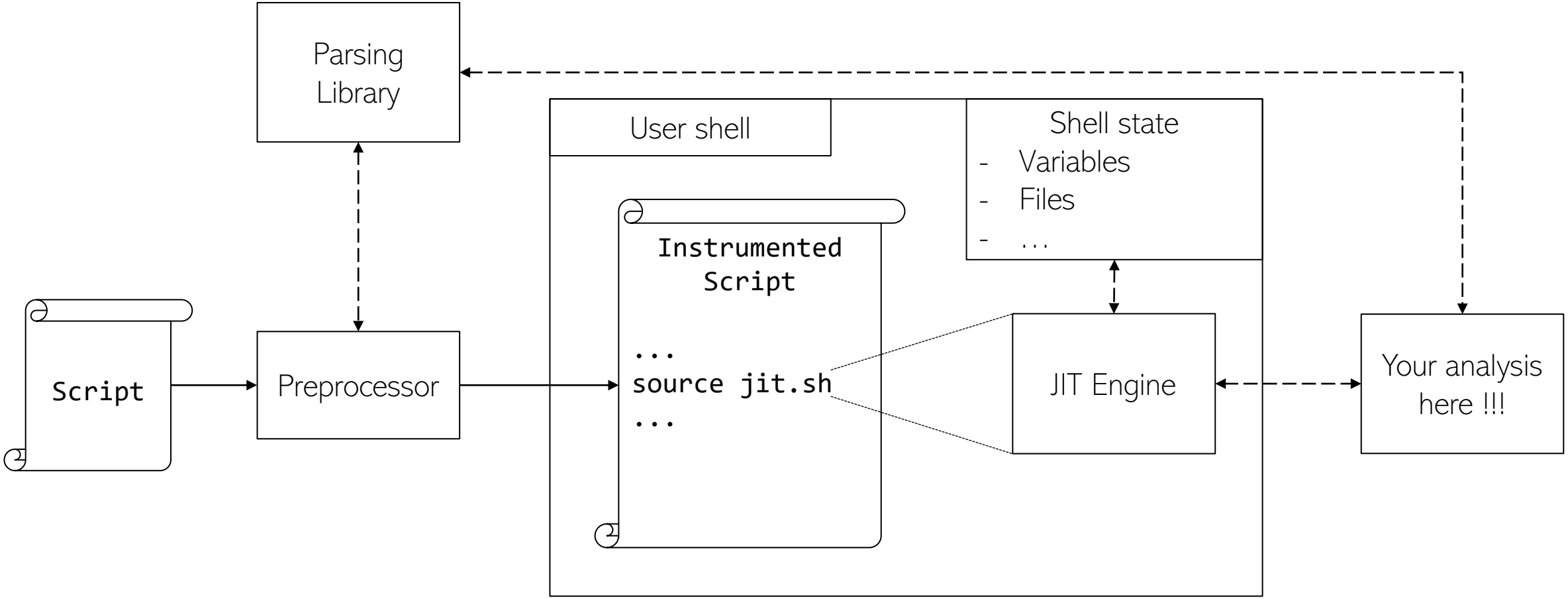
- Error-proneness
  - accidentally `rm -rf /` 
- Hard to learn
  - still googling for if-then-else shell syntax
- Redundant recomputation
  - we have to use Makefiles etc
- Lack of support for contemporary deployments
  - managing a distributed cluster

Recent exception: Rattle [1]

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

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!

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