

Advanced

T

pics

in

C

mpilers



NOELLEGym

Simone Campanoni
simone.campanoni@northwestern.edu



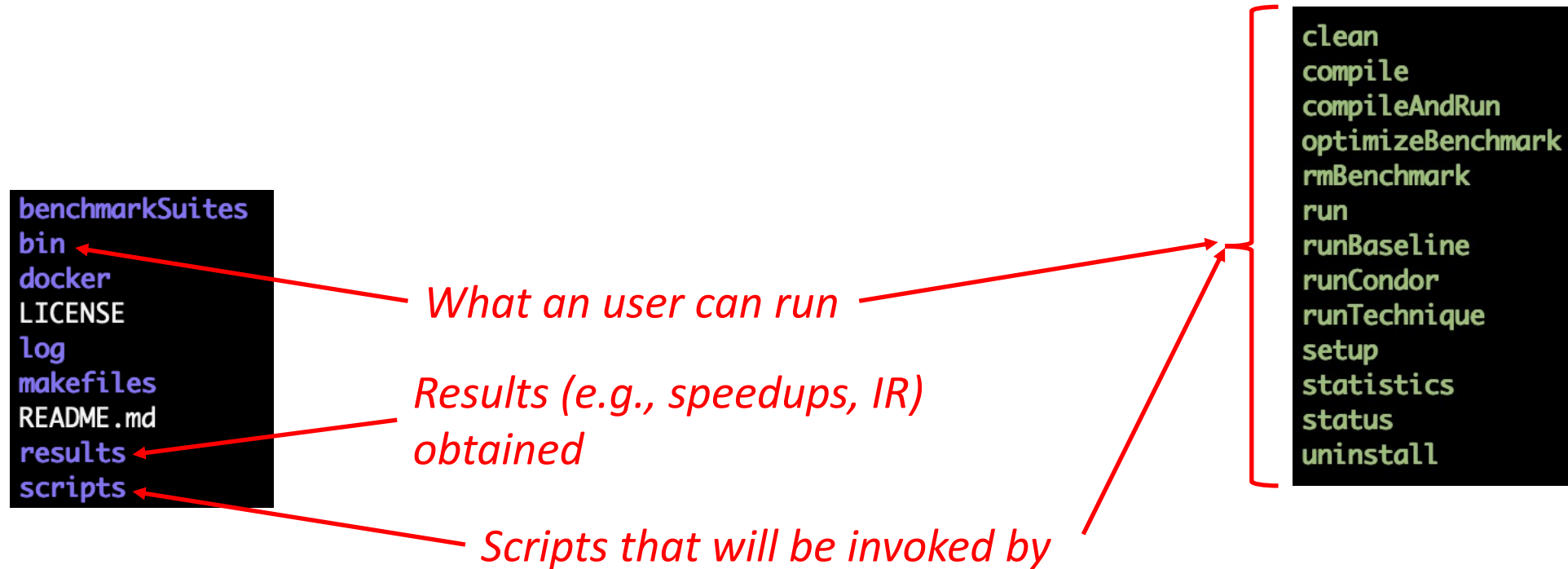
Outline

- Introduction
- Compile and optimize benchmarks
- Run benchmarks
- Inspect and modify the sources of a benchmark

NOELLEGym: introduction

- Infrastructure to test NOELLE-based optimizations on benchmarks typically used in research venues [link](#)
- Not particularly well designed
 - Started as a quick “put-together” infrastructure to quickly collect results
 - We are *slowly* improving its design
 - Feel free to make changes and do pull-requests (we’ll all appreciate it!)

NOELLEGym: structure



Software dependences

Your environment needs to have the following installation:

- GO (we tested with 1.13.7 as well as 1.17.13)
- LLVM 9.0.0
- Bash
- Python 3

Software dependences on hanlon and alike

- `export PATH=/home/software/go1.17.13/bin/:$PATH`
- `export PATH=/home/software/llvm-9.0.0/bin/:$PATH`

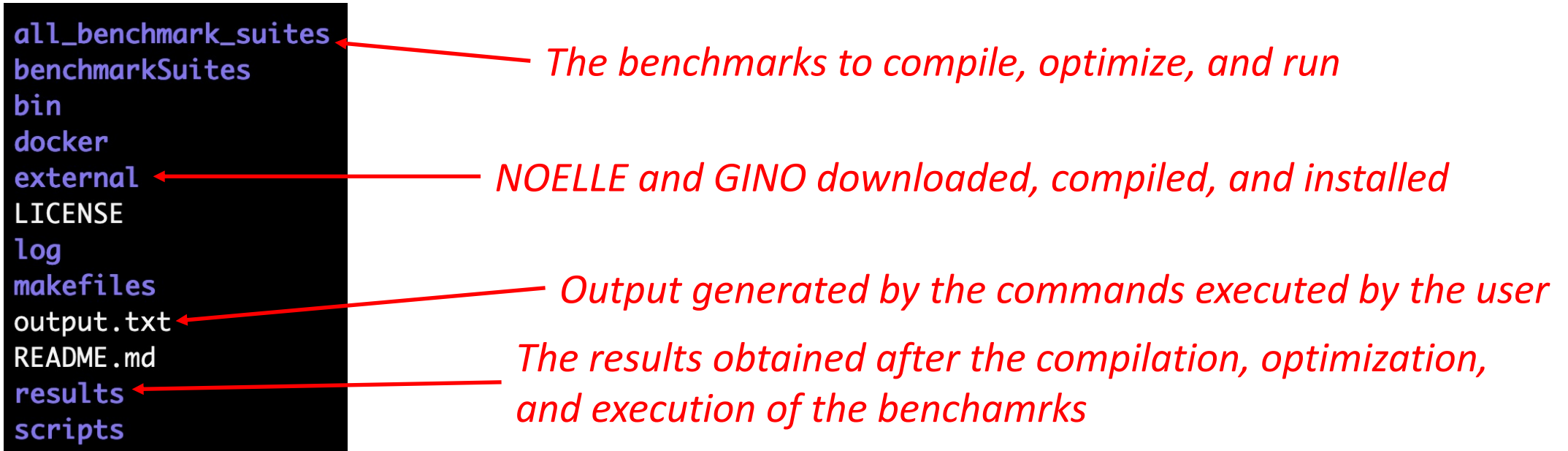
Software dependences on Zythos

- source /project/go/go_1.13.7/enable
- source /project/extra/llvm/9.0.0/enable
- source /project/gllvm/enable

Compiling NOELLEGym

- NOELLEGym relies on software that needs to be compiled
- To do so, run:
(after your environment is set as described in previous slides)
`./bin/setup`

NOELLEGym: structure after setup



Outline

- Introduction
- Compile and optimize benchmarks
- Run benchmarks
- Inspect and modify the sources of a benchmark

Parallelize all benchmarks with GINO

`./bin/clean`

`./bin/compile`

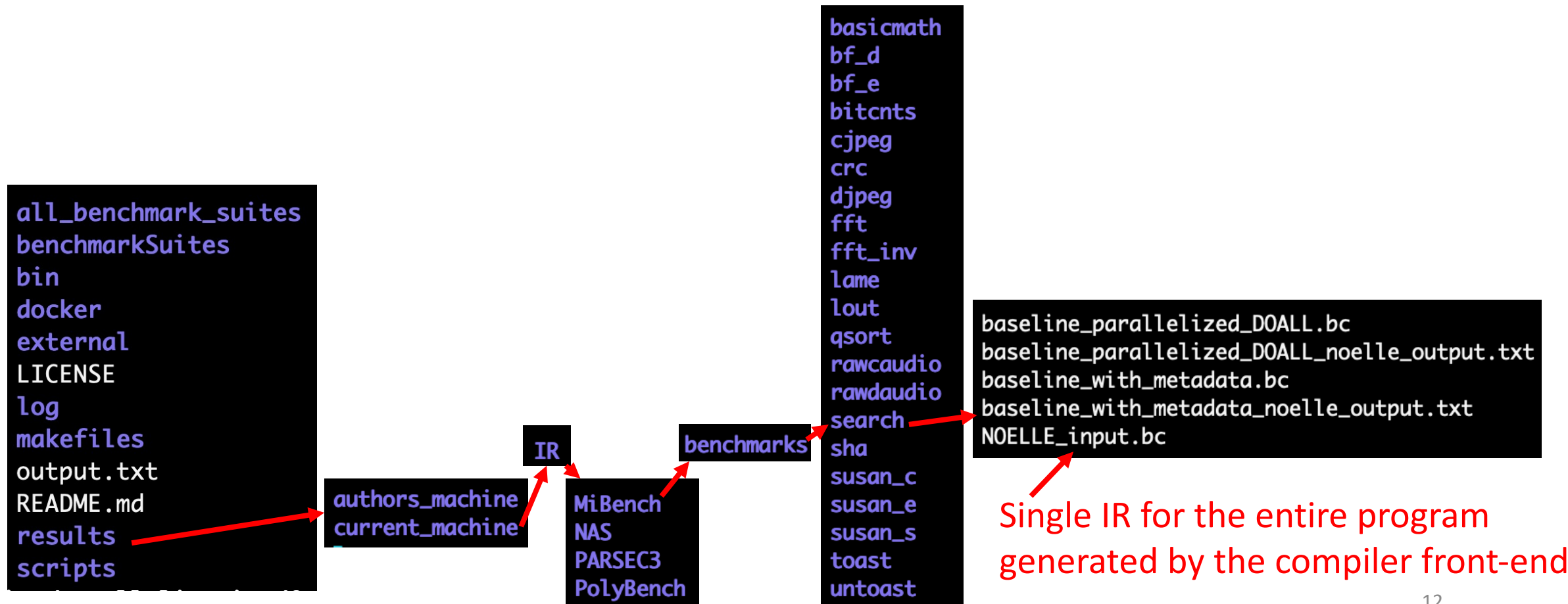
```
all_benchmark_suites
benchmarkSuites
bin
docker
external
LICENSE
log
makefiles
output.txt
README.md
results
scripts
```

The first time this command executes, it performs the following:

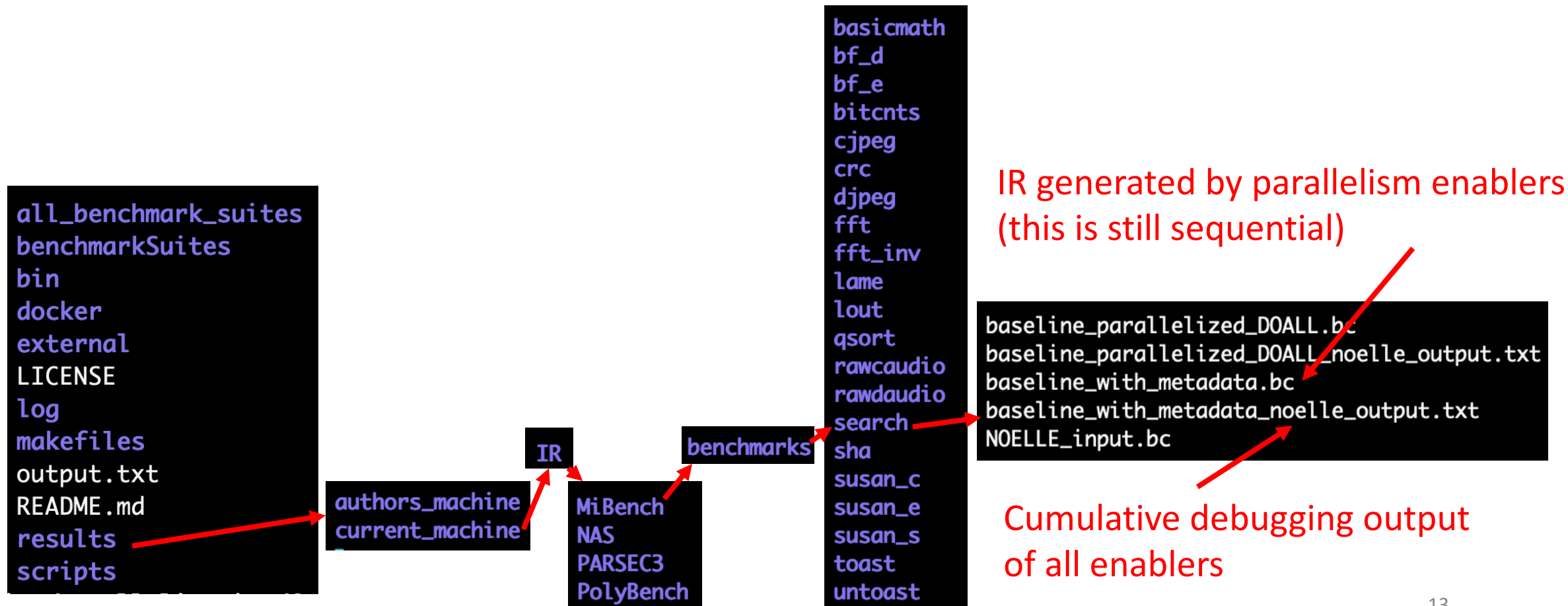
1. It generates the single IR file for an entire benchmark, for all benchmarks, in all benchmark suites
2. It runs the optimization/parallelization for all benchmarks, in all benchmark suites

Sub-sequent invocations of the same command will only perform 2.

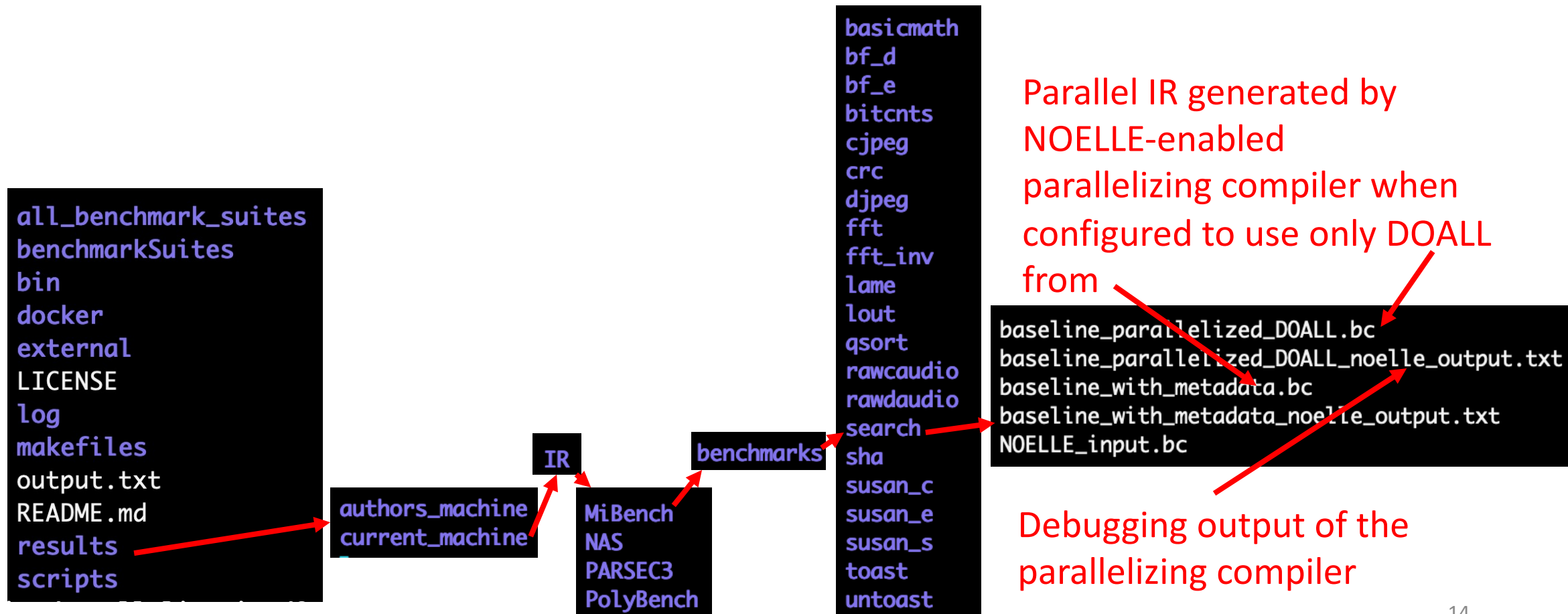
Parallelize all benchmarks with GINO: Checking the output



Parallelize all benchmarks with GINO: Checking the output



Parallelize all benchmarks with GINO: Checking the output



Check the status

`./bin/status`

It checks the status of results/current_machine of:

1. IR generated
2. Statistics about dependences in IR, parallelization performed
3. Execution times of the different IRs

It prints what is missing

```
$ ./bin/status
Next we list the results/code that are currently missing in "results/current_machine"

== IR
The suite "MiBench" has only 19 (over 21) baselines
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with NONE benchmarks
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with DOALL benchmarks
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with HELIX benchmarks
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with DSWP benchmarks
The suite "NAS" has only 7 (over 8) benchmarks parallelized with DOALL benchmarks
The suite "NAS" has only 7 (over 8) benchmarks parallelized with HELIX benchmarks
The suite "NAS" has only 6 (over 8) benchmarks parallelized with DSWP benchmarks
The suite "PARSEC3" has only 5 (over 8) baselines
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with NONE benchmarks
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with DOALL benchmarks
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with HELIX benchmarks
The suite "PARSEC3" has only 4 (over 8) benchmarks parallelized with DSWP benchmarks

== Dependences
The suite "MiBench" has only 19 (over 21) benchmarks with LLVM dependence information
The suite "MiBench" has only 19 (over 21) benchmarks with NOELLE dependence information
The suite "PARSEC3" has only 5 (over 8) benchmarks with LLVM dependence information
The suite "PARSEC3" has only 5 (over 8) benchmarks with NOELLE dependence information

== Parallelization
The suite "MiBench" has only 19 (over 21) benchmarks with parallelization statistics for DOALL
The suite "MiBench" has only 19 (over 21) benchmarks with parallelization statistics for DSWP
The suite "MiBench" has only 19 (over 21) benchmarks with parallelization statistics for HELIX
The suite "MiBench" has only 19 (over 21) benchmarks with parallelization statistics for NONE
The suite "NAS" has only 7 (over 8) benchmarks with parallelization statistics for DOALL
The suite "NAS" has only 6 (over 8) benchmarks with parallelization statistics for DSWP
The suite "NAS" has only 7 (over 8) benchmarks with parallelization statistics for HELIX
The suite "PARSEC3" has only 5 (over 8) benchmarks with parallelization statistics for DOALL
The suite "PARSEC3" has only 4 (over 8) benchmarks with parallelization statistics for DSWP
The suite "PARSEC3" has only 5 (over 8) benchmarks with parallelization statistics for HELIX
The suite "PARSEC3" has only 5 (over 8) benchmarks with parallelization statistics for NONE

== Execution time
The suite "MiBench" has only 10 (over 21) baselines with execution times
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with NONE with execution times
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with DOALL with execution times
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with HELIX with execution times
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with DSWP with execution times
The suite "NAS" has only 7 (over 8) benchmarks parallelized with DOALL with execution times
The suite "NAS" has only 7 (over 8) benchmarks parallelized with HELIX with execution times
The suite "NAS" has only 6 (over 8) benchmarks parallelized with DSWP with execution times
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with NONE with execution times
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with DOALL with execution times
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with HELIX with execution times
The suite "PARSEC3" has only 4 (over 8) benchmarks parallelized with DSWP with execution times
```

Parallelize a benchmark with NOELLE

`./bin/clean`

`./bin/optimizeBenchmark MiBench/search DOALL`

```
all_benchmark_suites
benchmarkSuites
bin
docker
external
LICENSE
log
makefiles
output.txt
README.md
results
scripts
```

The first time this command executes, it performs the following:

1. It generates the single IR file for an entire benchmark, for all benchmarks, in all benchmark suites
2. It runs the optimization/parallelization for only the benchmark specified as input

Sub-sequent invocations of the same command will only perform 2.

Output generated by the commands executed by the user

Outline

- Introduction
- Compile and optimize benchmarks
- Run benchmarks
- Inspect and modify the sources of a benchmark

Run benchmarks

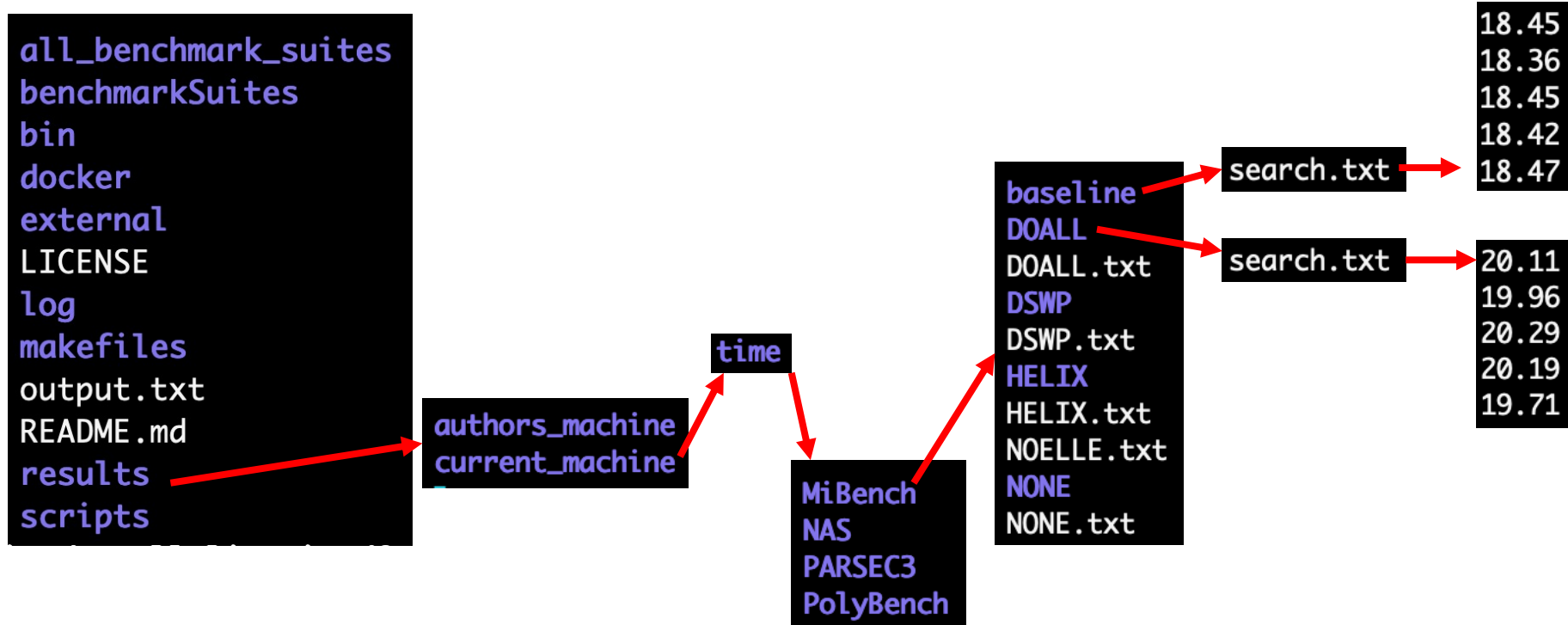
`./bin/clean`

`./bin/run`

It performs the following for every benchmark that has an IR:

1. If the baseline time of benchmark X is not available in `results/current_machine/time`, then X is optimized using `clang -O3 -march=native` w/o using NOELLE, and the so-generated binary runs Y times
2. If the IR of an optimization (DOALL) is available and its execution time isn't available in `results/current_machine/time`, then it generates the binary from the optimized IR (e.g., `baseline_parallelized_DOALL.bc`), and it runs that binary Y times

Checking the times



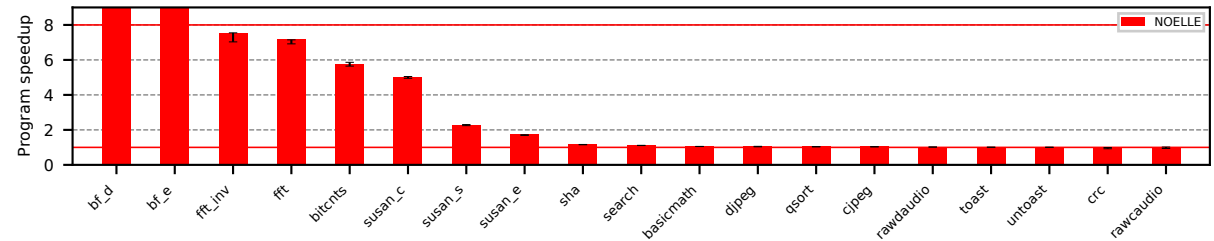
Checking the speedups

```
all_benchmark_suites
benchmarkSuites
bin
docker
external
LICENSE
log
makefiles
output.txt
README.md
results
scripts
```

```
authors_machine
current_machine
```

plots

```
MiBench_NOELLE.pdf
MiBench.pdf
NAS_NOELLE.pdf
NAS.pdf
PARSEC3_NOELLE.pdf
PARSEC3.pdf
PolyBench_NOELLE.pdf
PolyBench.pdf
```



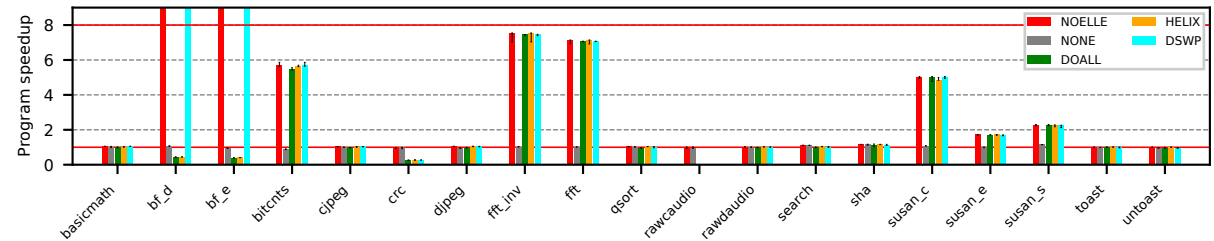
Checking the speedups

```
all_benchmark_suites
benchmarkSuites
bin
docker
external
LICENSE
log
makefiles
output.txt
README.md
results
scripts
```

```
authors_machine
current_machine
```

plots

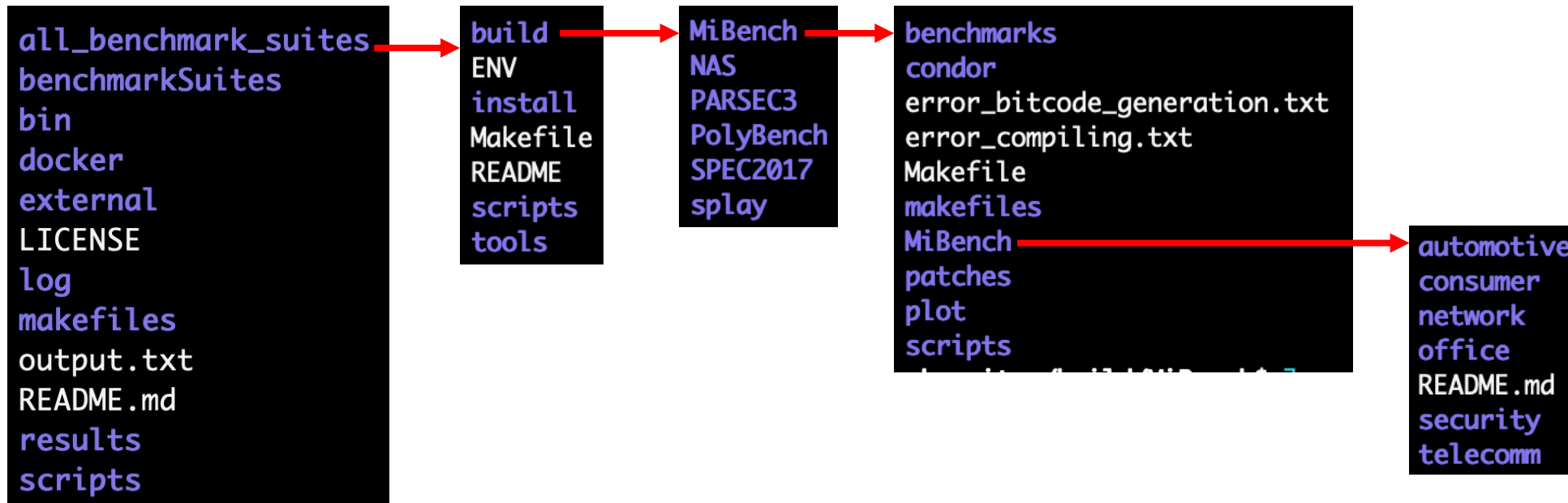
```
MiBench_NOELLE.pdf
MiBench.pdf
NAS_NOELLE.pdf
NAS.pdf
PARSEC3_NOELLE.pdf
PARSEC3.pdf
PolyBench_NOELLE.pdf
PolyBench.pdf
```



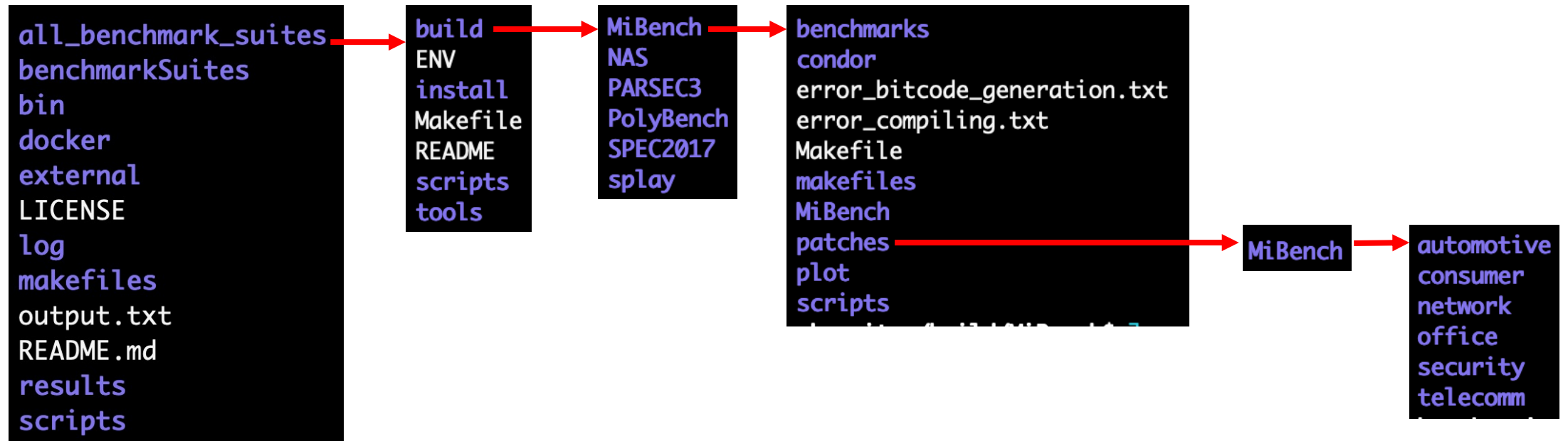
Outline

- Introduction
- Compile and optimize benchmarks
- Run benchmarks
- **Inspect and modify the sources of a benchmark**

Checking the sources of a benchmark



Changing the sources of a benchmark



After it, you need to delete `results/current_machine` and re-run your optimization

Always have faith in your ability

Success will come your way eventually

Best of luck!