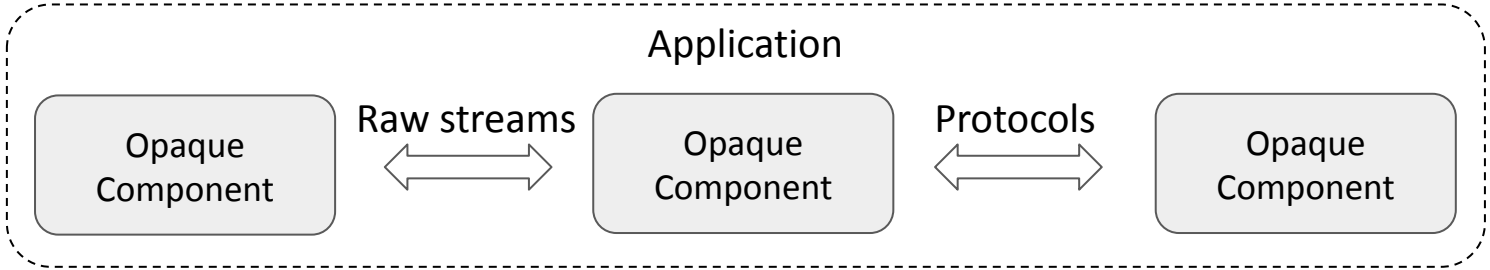
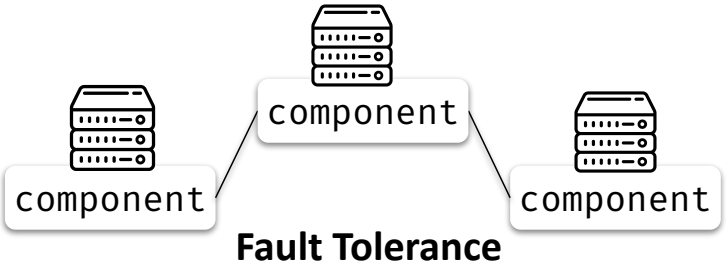
Fractal: Fault-Tolerant Shell-Script Distribution

Zhicheng Huang*, Ramiz Dundar*, **Yizheng Xie**, Konstantinos Kallas[UCLA], Nikos Vasilakis

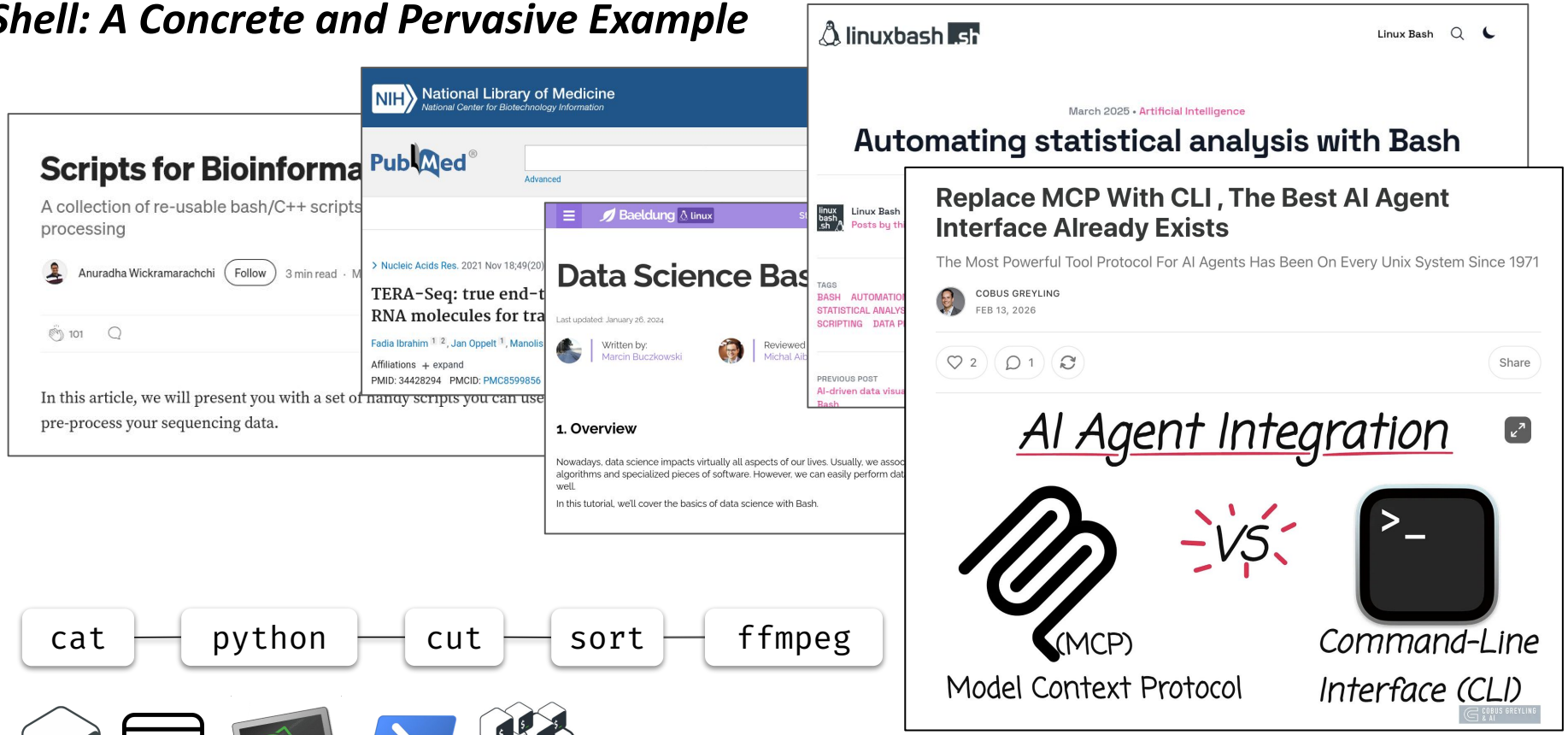
github.com/binpash/fractal

Opaque Software Components are Pervasive





Shell: A Concrete and Pervasive Example

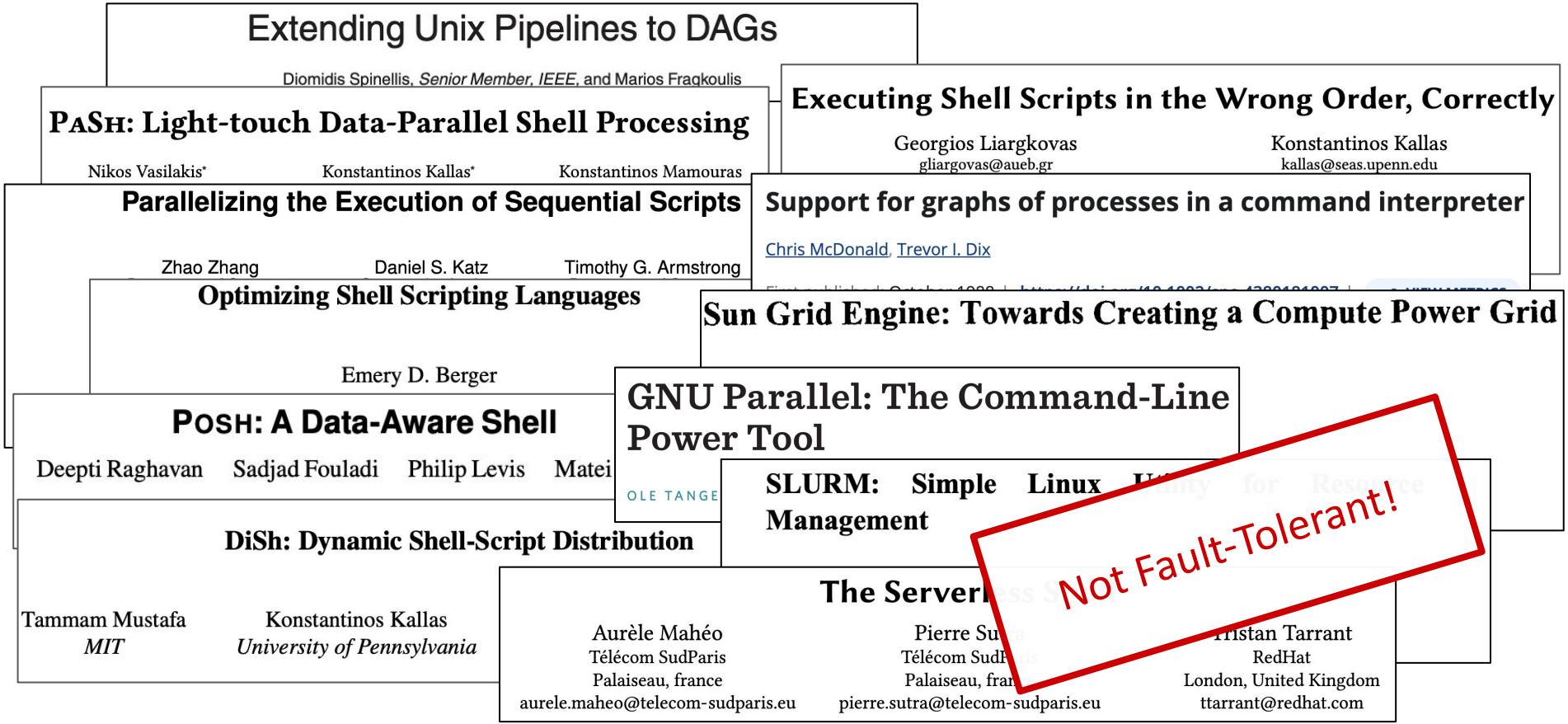


`cat` — `python` — `cut` — `sort` — `ffmpeg`

... **Shell Programs**

+more!

Existing Shell Programs Distribution Systems Lack Fault Tolerance

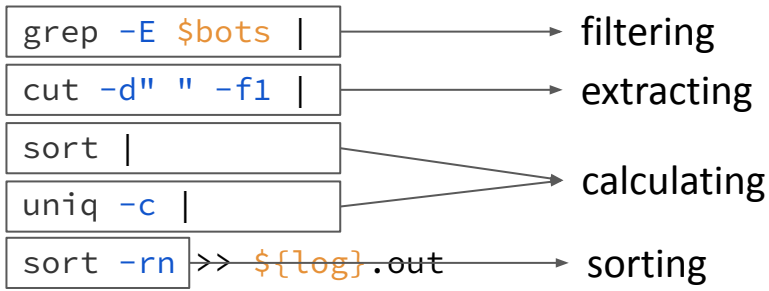


An Example Pipeline

```
#!/bin/bash

bots='Googlebot|Bingbot|Baiduspider'

for log in $(hdfs dfs -ls -C $in); do
  # Identify bot IPs by visit frequency
  hdfs dfs -cat $log |
    grep -E $bots |
    cut -d" " -f1 |
    sort |
    uniq -c |
    sort -rn >> ${log}.out
  # ...
done
```

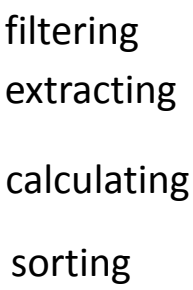


Rewriting the Pipeline Using Fault-Tolerant Systems

```
#!/bin/bash

bots='Googlebot|Bingbot|Baiduspider'

for log in $(hdfs dfs -ls -C $in); do
  # Identify bot IPs by visit frequency
  hdfs dfs -cat $log |
    grep -E $bots |
    cut -d" " -f1 |
    sort |
    uniq -c |
    sort -rn >> ${log}.out
  # ...
done
```




```
sc.textFile(log)
  .filter(lambda l: p.search(l)...)
  .map(lambda l: l.split(" ")[0])
  .map(lambda key: (key, 1))
  .reduceByKey(lambda a, b: a + b)
  .sortBy(lambda kv: kv[1]...)
```

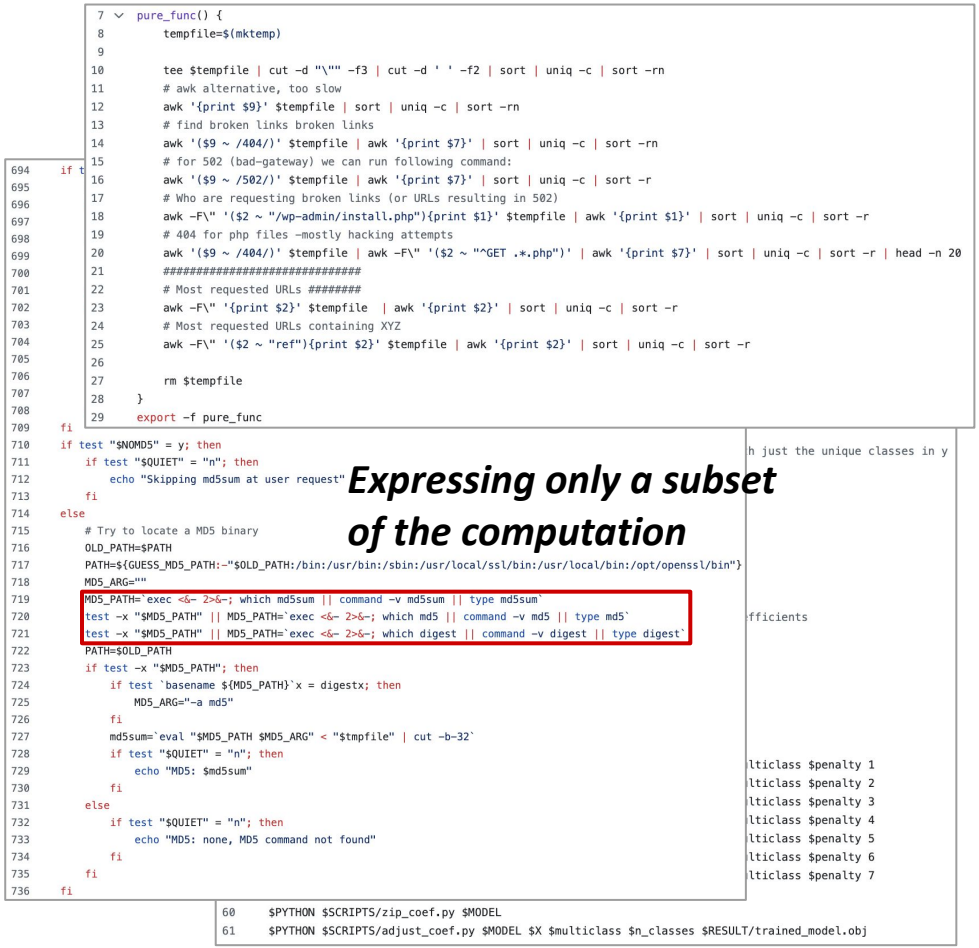
Spark

```
# mapper.py
for l in stdin:
  if bots.search(l)...
```

```
# reducer.py
for l in sys.stdin:
  k, v = l.split(...)
```

APACHE hadoop

Manual Rewriting is Inefficient and Error-Prone



Expressing only a subset of the computation

Error-prone!
E.g., env variables, conditionals…

```
sc.textFile(log)
  .filter(lambda l: p.search(l)...)
  .map(lambda l: l.split(" ")[0])
  .map(lambda key: (key, 1))
  .reduceByKey(lambda a, b: a + b)
  .sortBy(lambda kv: kv[1]...)
```

Spark

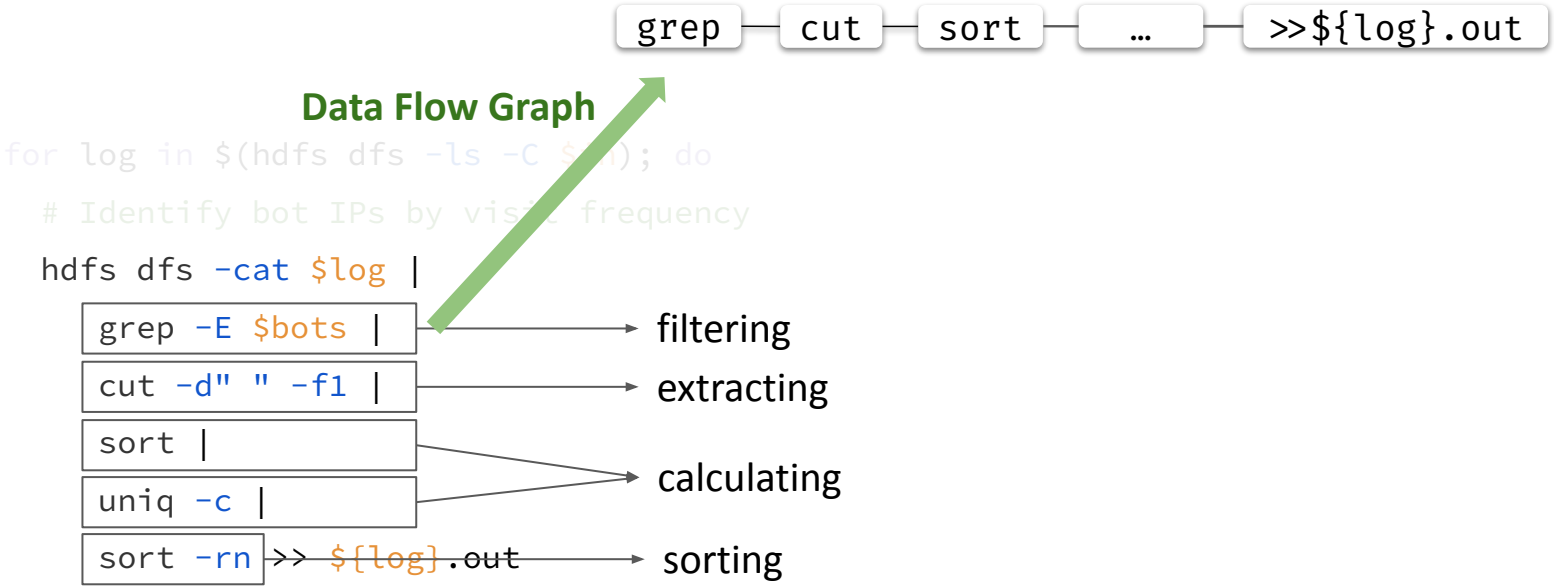
```
# mapper.py
for l in stdin:
  if bots.search(l)...
```

```
# reducer.py
for l in sys.stdin:
  k, v = l.split(...)
```

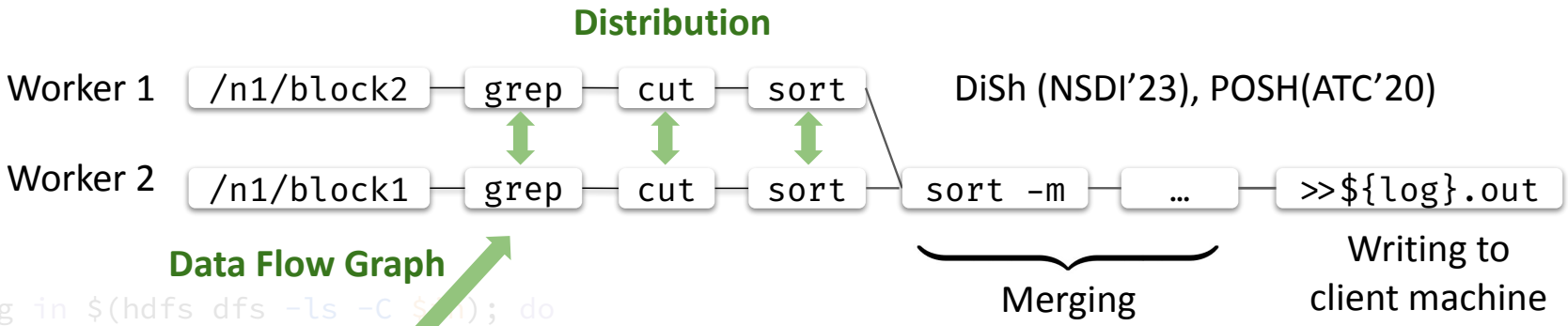
APACHE hadoop

Distributing the Example Pipeline

DiSh (NSDI'23), POSH(ATC'20)

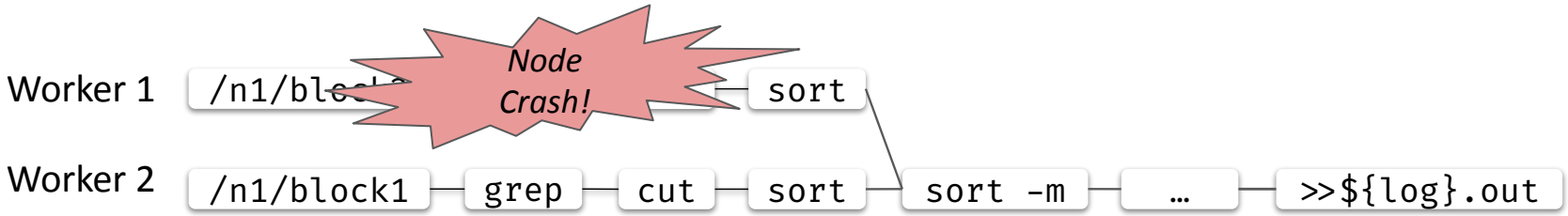


Distributing the Example Pipeline

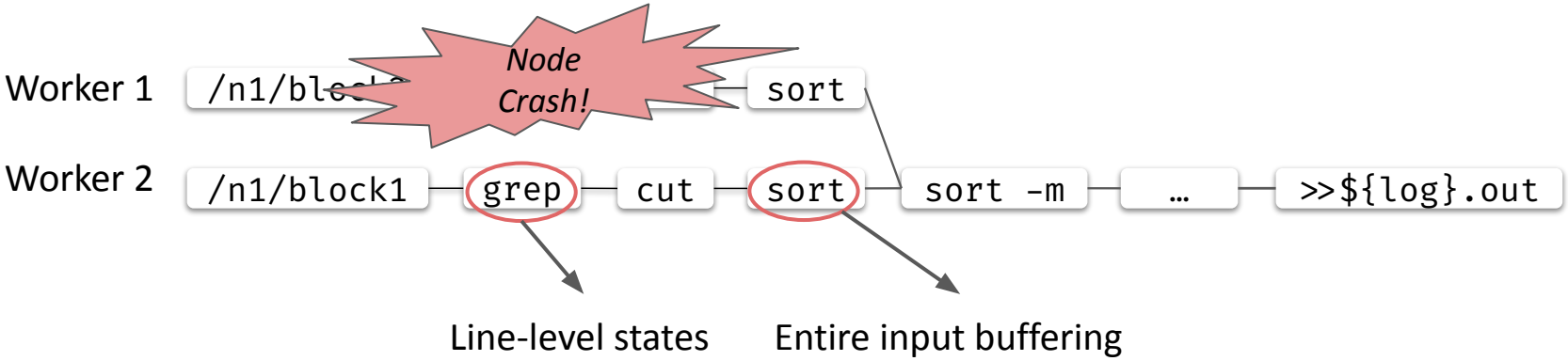



```
for log in $(hdfs dfs -ls -C $(); do
  # Identify bot IPs by visit frequency
  hdfs dfs -cat $log |
    grep -E $bots |
    cut -d" " -f1 |
    sort |
    uniq -c |
    sort -rn >> ${log}.out
  # ...
done
```

Tolerating Faults in Distribution is Challenging



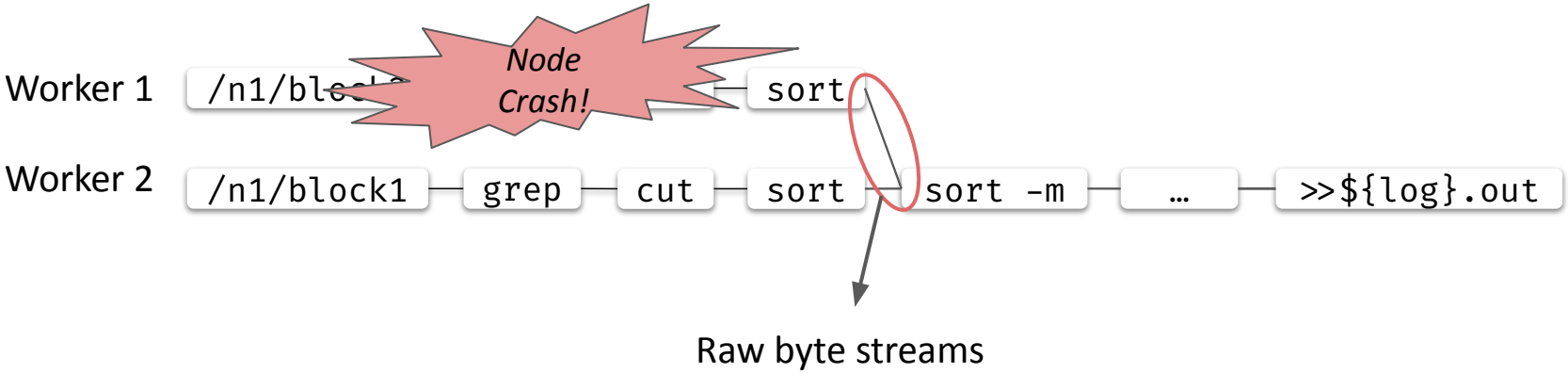
Checkpointing Opaque States is Hard



Difficult to checkpoint and recover from all states correctly

Idea 1: Checkpointing? E.g., Flink (VLDB'17), Naiad (SOSP'13)

No Barrier, No Stages, No Structured Data



Idea 2: Barrier-based? E.g., MapReduce, Hadoop Streaming

Idea 3: Lineage-based? E.g., Spark (NSDI'12), Ray (OSDI'18)

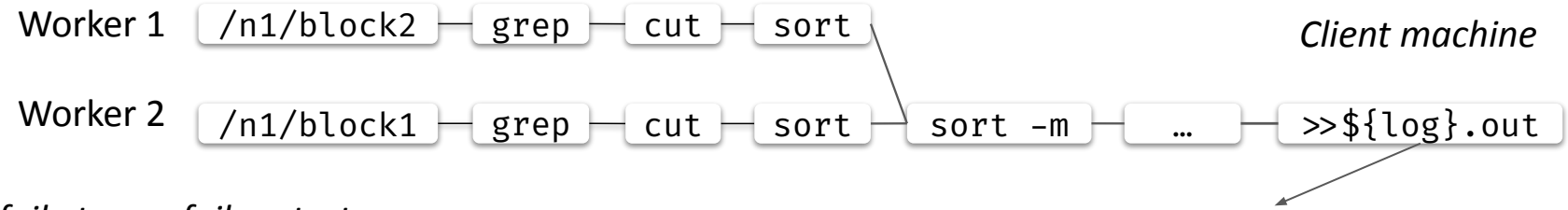
Requirement: either

(1) barrier or stage, e.g., mapper, reducer

(2) structured task or data, e.g., objects

Fractal *is the first system to provide fault tolerance for opaque components in shell-program distribution.*

Fault Model and Goal



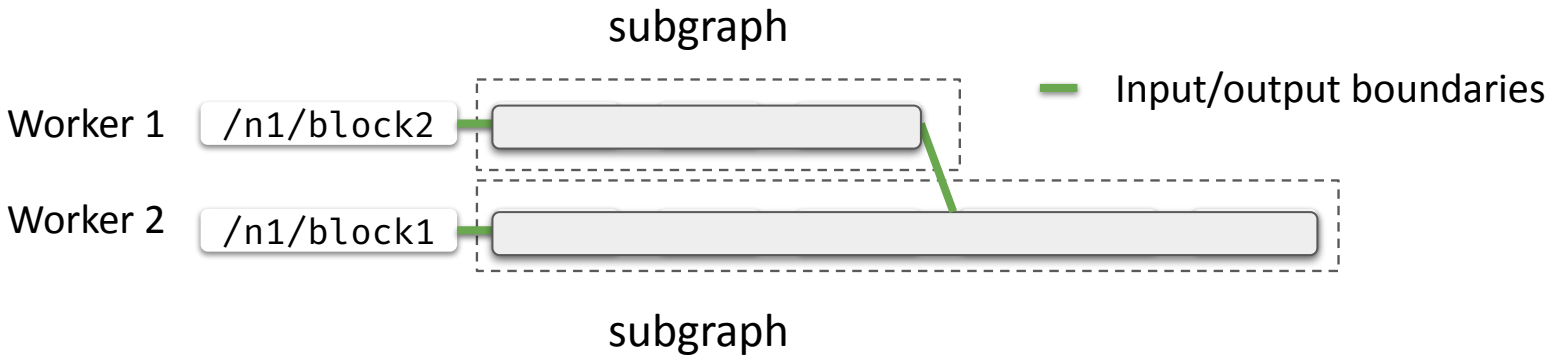
Target: fail-stop or fail-restart crashes of ***worker nodes***

Side-effectful, non-idempotent (e.g., interacting with client file system)…

Distributed workload: deterministic and idempotent computation

Goal: the exact same behaviors (e.g., exit codes) and output (e.g., no duplicates or loss)

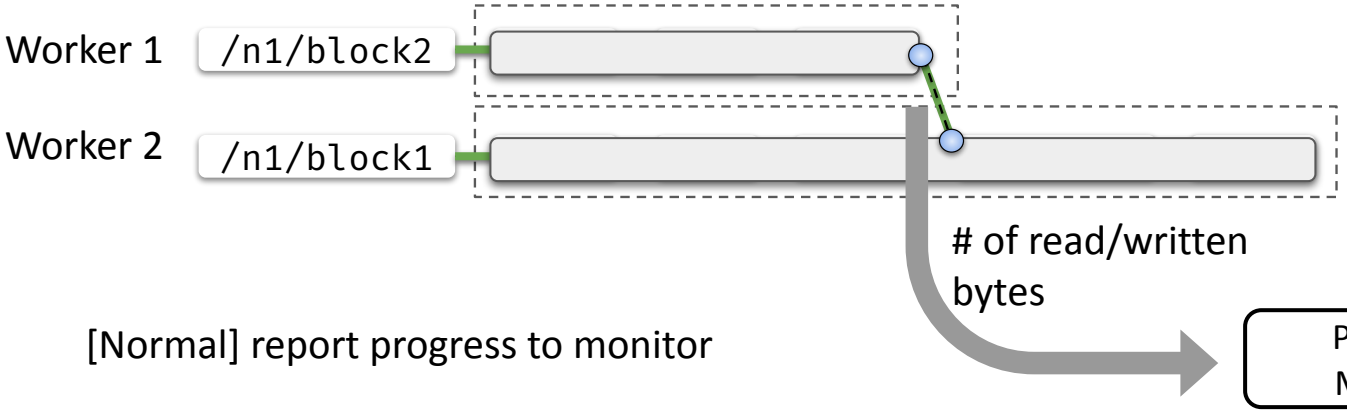
Key Insight: Opaque Subgraph Granularity



Fractal only tracks input and output for each *subgraph.*

Bolt-on Progress Tracking via Raw Bytes

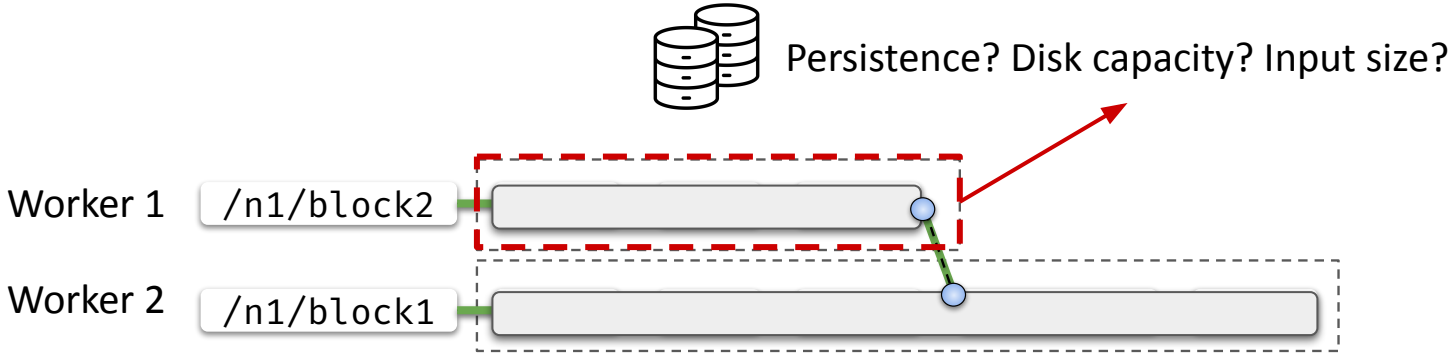
Remote pipes: drop-in replacement for Unix pipes



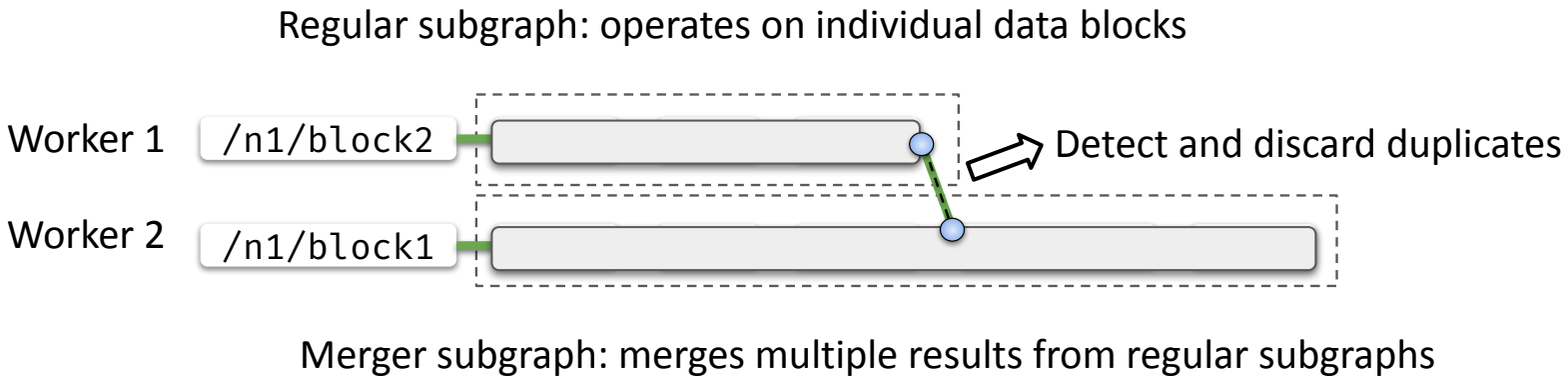
[Normal] report progress to monitor

[Recovery] detect and discard duplicates

Fast Recovery via Dynamic Output Persistence



Recovering from Faults with Different Strategies



Input for recovering regular faults: the same input data blocks

Input for recovering merger faults: persisted results or re-execution of upstream subgraphs

Evaluation

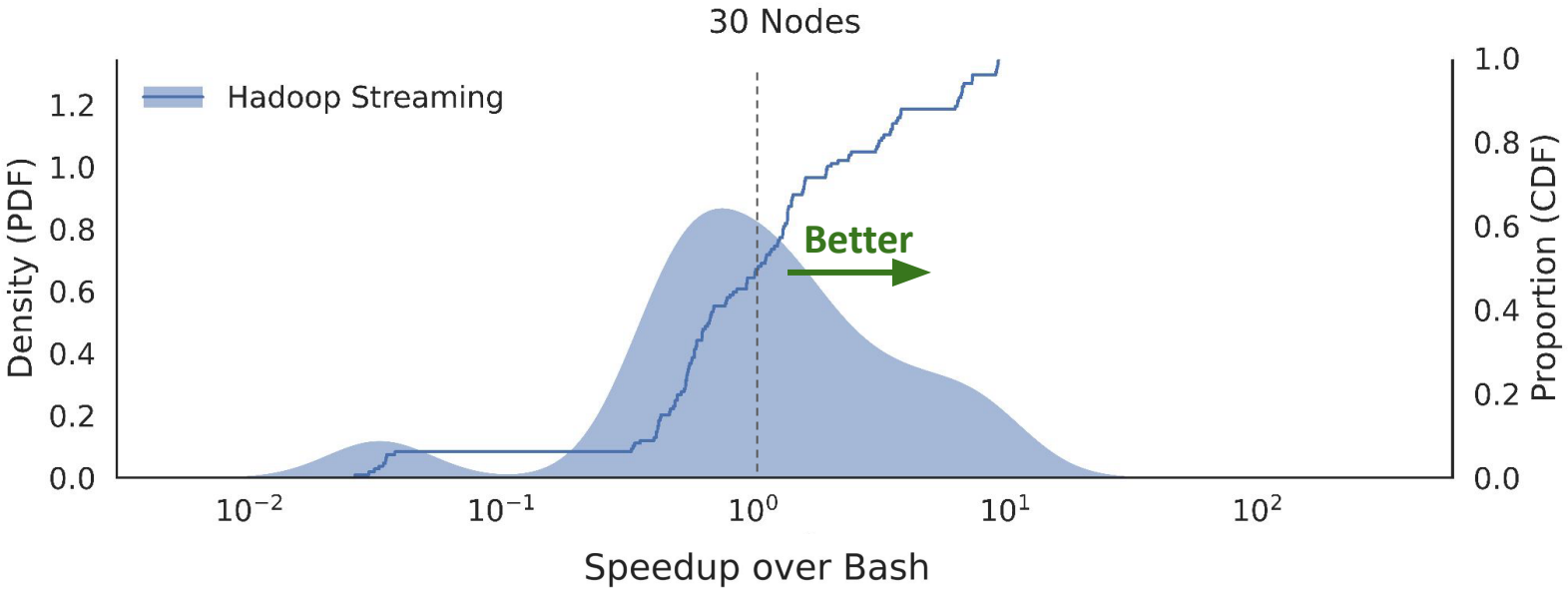
How fast is Fractal under fault-free execution and fault recovery?

Evaluation

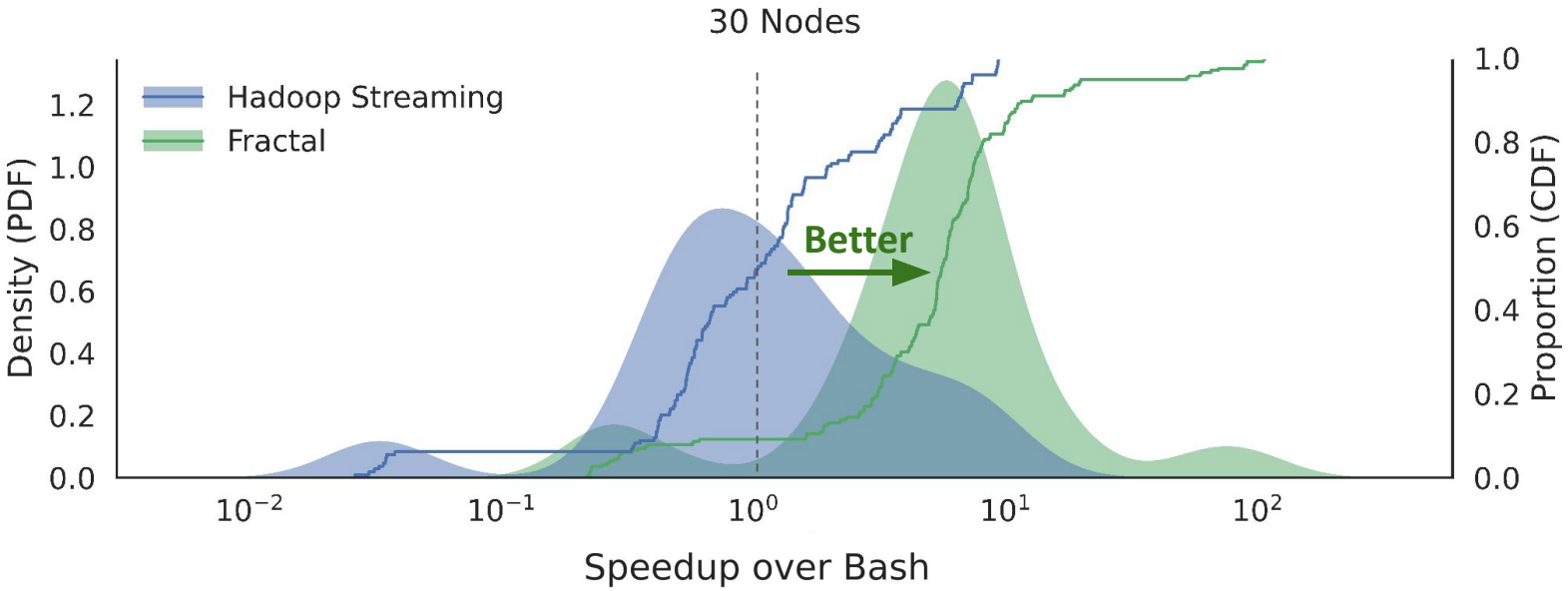
- Benchmarks
  - 5 Benchmarks from the Koala benchmark suite[1]
  - Totaling 77 scripts and up to 33.4 GB input.
- Baselines
  - Bash: the standard shell interpreter
  - DiSh (NSDI'23): a state-of-the-art shell distribution system
  - Apache Hadoop Streaming (AHS): a MapReduce system that offers fault-tolerant distributed execution of arbitrary binaries
- *frac:* fault injection subsystem

[1] Evangelos Lamprou, Ethan Williams, Georgios Kaoukis, Zhuoxuan Zhang, Michael Greenberg, Konstantinos Kallas, Lukas Lazarek, and Nikos Vasilakis. "The Koala benchmarks for the shell: characterization and implications." USENIX ATC. 2025.

Evaluation — Fault-free Performance

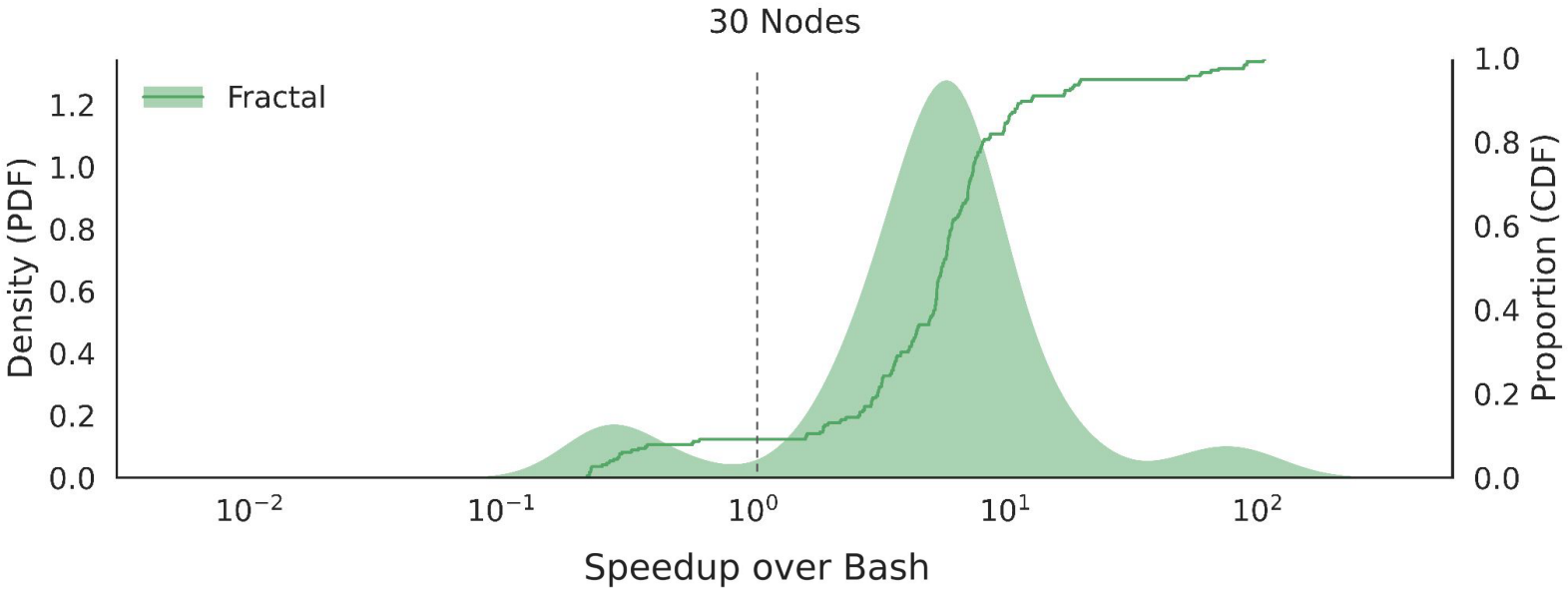


Evaluation — Fault-free Performance

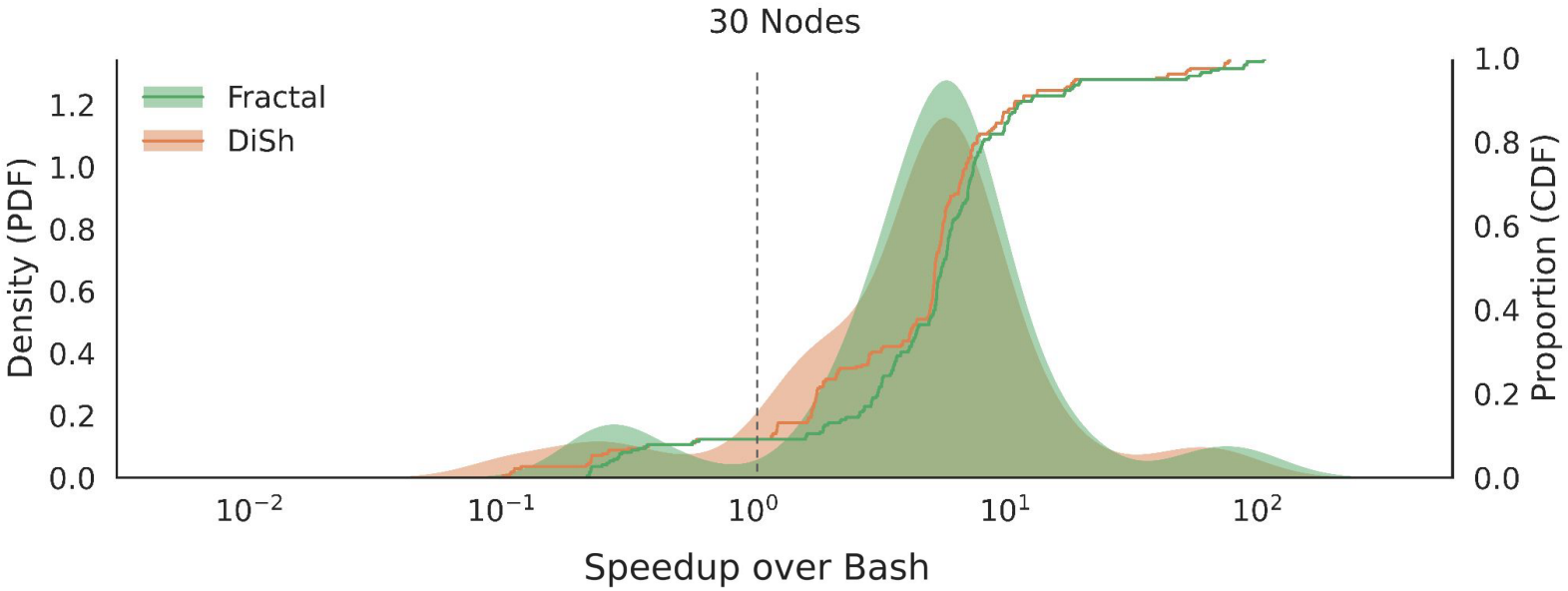


Fractal: max — 107.84× Hadoop Streaming: max — 9.48×

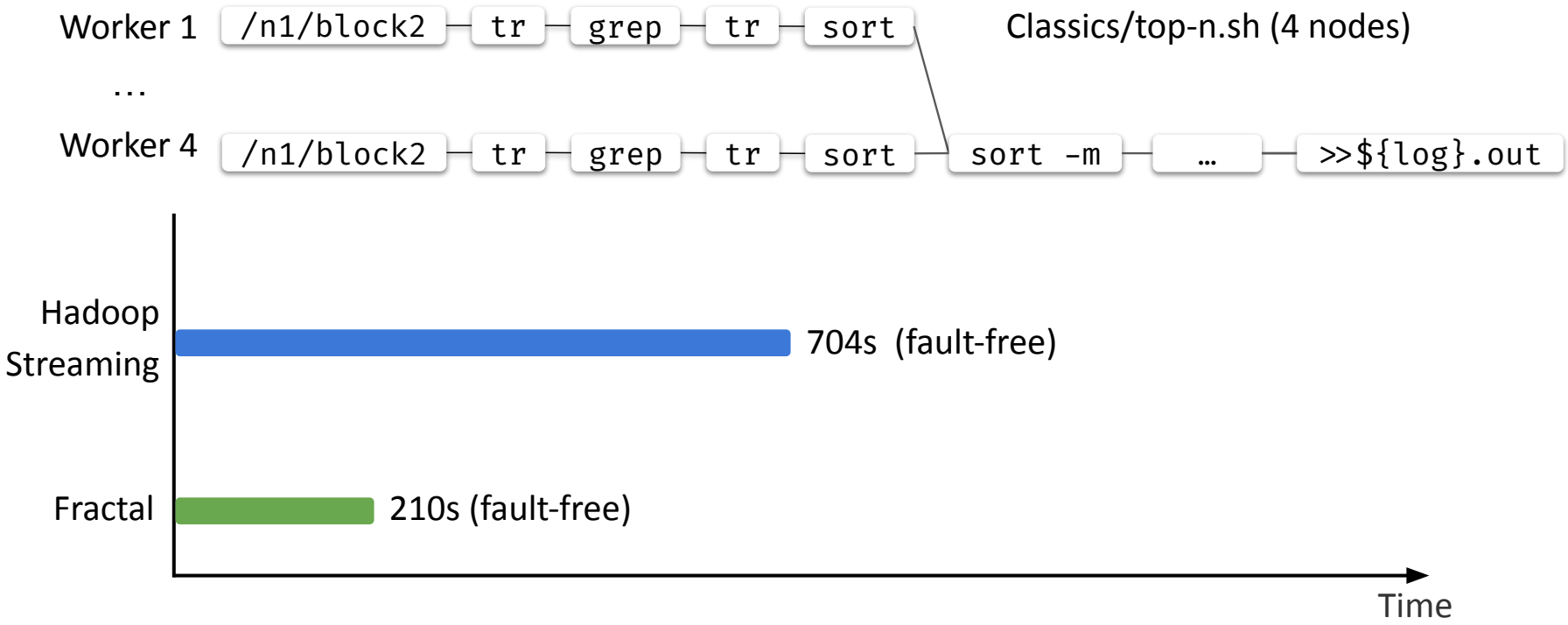
Evaluation — Fault-free Performance



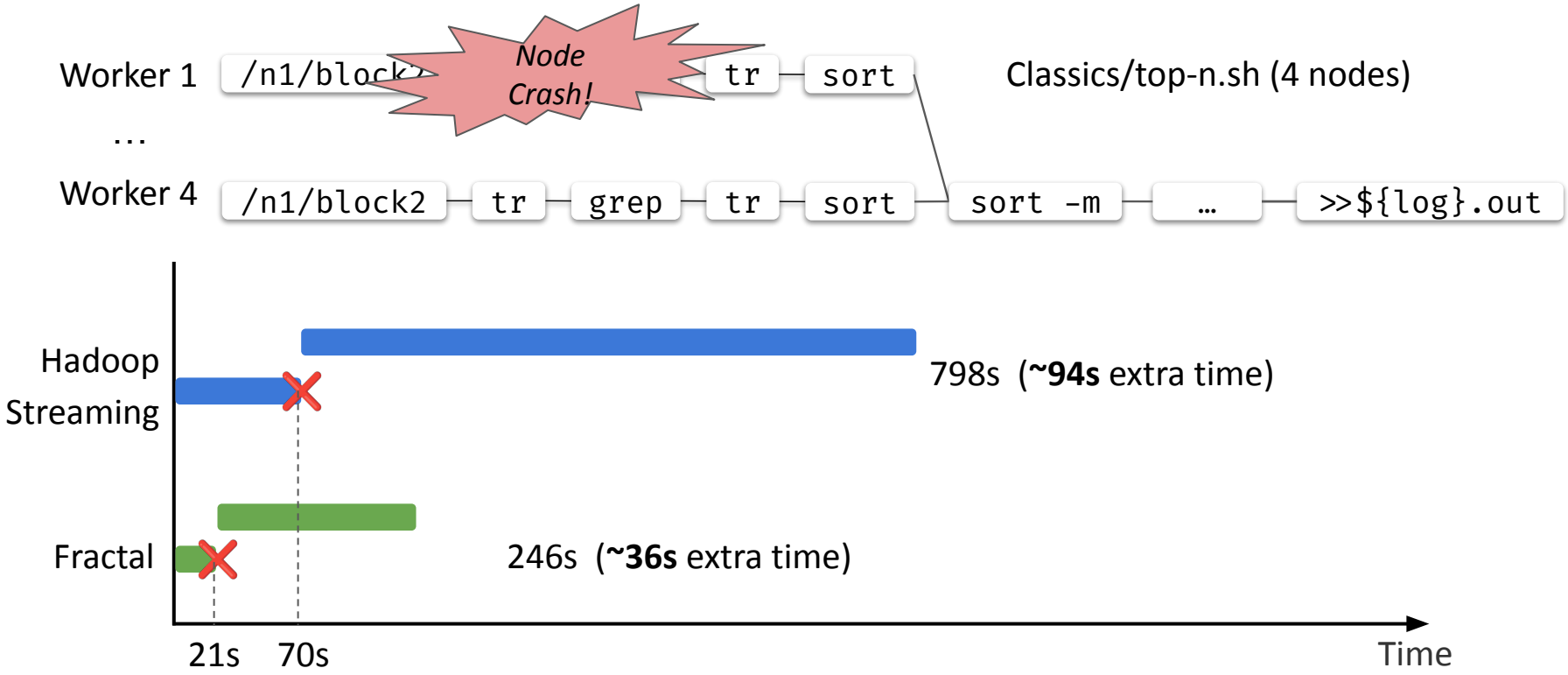
Evaluation — Fault-free Performance



Evaluation — Fault Recovery Performance

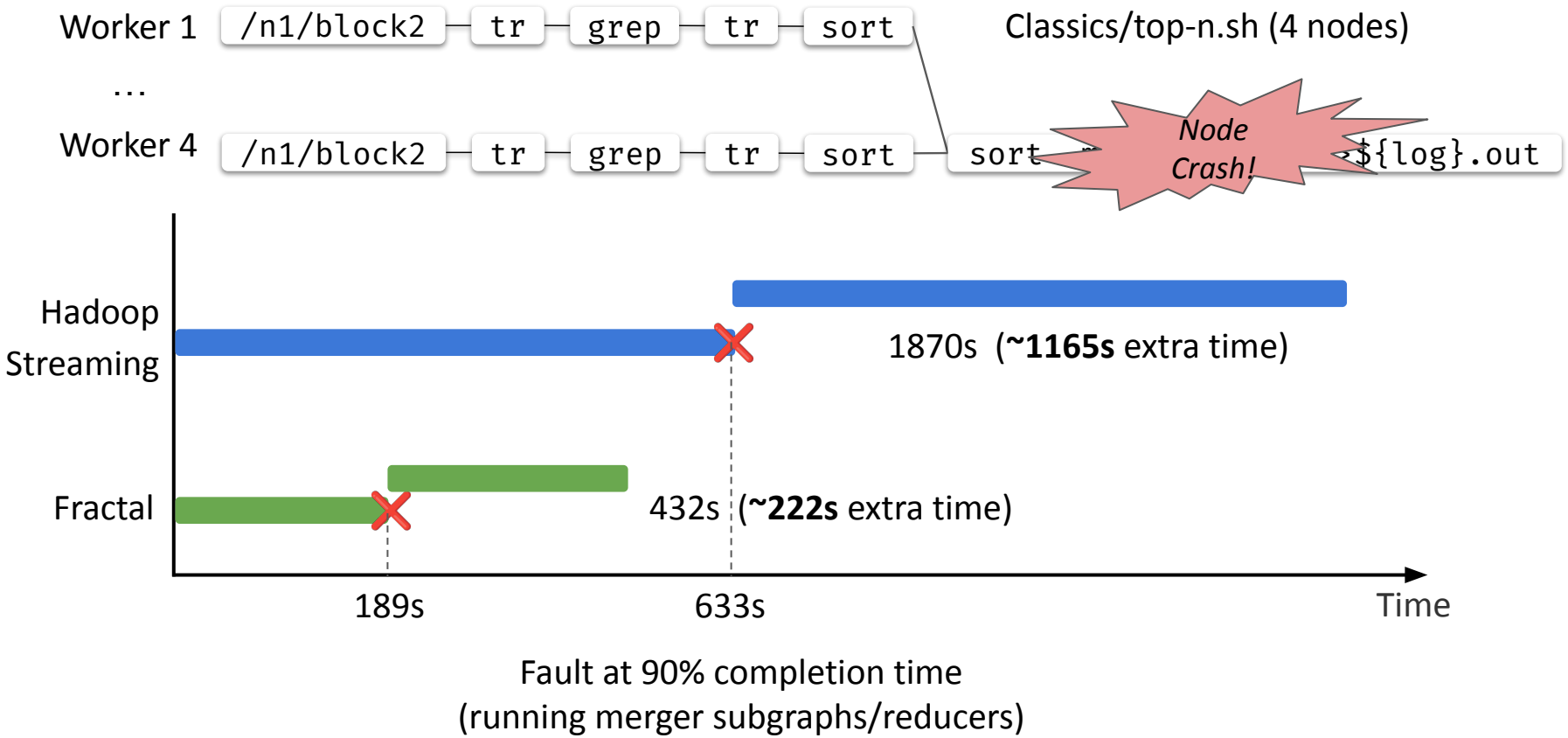


Evaluation — Fault Recovery Performance



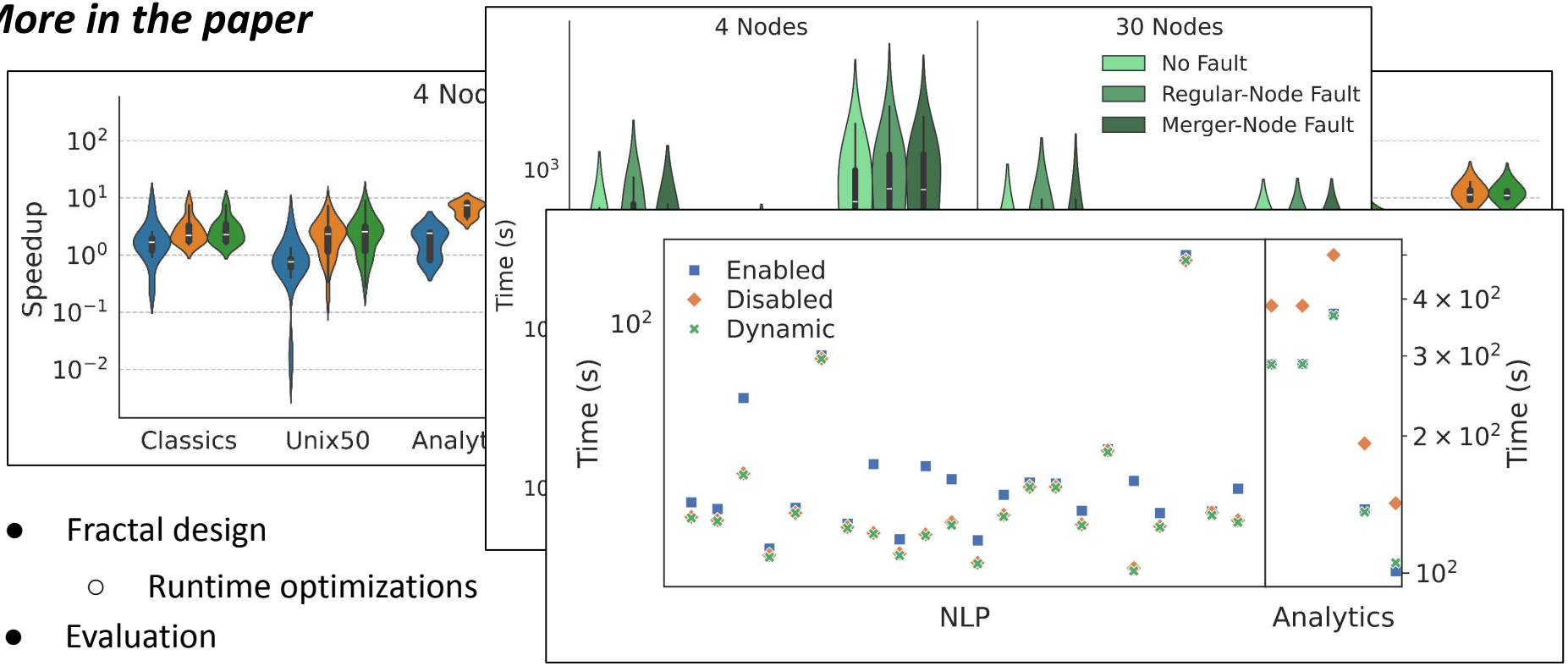
Fault at 10% completion time
(running regular subgraphs/mappers)

Evaluation — Fault Recovery Performance



Fault at 90% completion time
(running merger subgraphs/reducers)

More in the paper



- Fractal design
  - Runtime optimizations
- Evaluation
  - Fine-grained performance and recovery analysis
  - …and more experiments!

Fractal: Fault-Tolerant Shell-Script Distribution

It offers:

- An execution engine to distribute unmodified Shell scripts with fault tolerance.
- Performance optimizations to improve efficiency at runtime.
- A fault injection subsystem to enable large-scale characterization of fault recovery.

Zhicheng Huang*, Ramiz Dundar*, **Yizheng Xie**,
Konstantinos Kallas[UCLA], Nikos Vasilakis

BROWN UCLA

github.com/binpash/fractal

ATLAS
atlas.cs.brown.edu