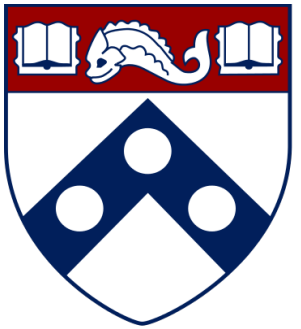
Monadic Composition for Deterministic, Parallel Batch Processing

Ryan Scott[1]
Ryan Newton[1]
Omar Navarro Leija[2]
Joe Devietti[2]

[1]Indiana University
[2]University of Pennsylvania




rgscott@indiana.edu

github.com/RyanGlScott

Unintended nondeterminism sucks.

Same commit, different results #770

Closed **amitaibu** opened this issue on Nov 19, 2012 · 2 comments

amitaibu commented on Nov 19, 2012

I have a build that was successful, but after "rebuilding" it fails.

Success VS Fail -- so I assume it's related to the environment?

can't reproduce examples #21

Open **casanovarodrigo** opened this issue on Jul 3 · 8 comments

Can't reproduce nested grid method from Examples #621

Closed **shorty2240** opened this issue on Jul 7 · 3 comments

Can't Reproduce the Accuracy for "Pre-trained word embeddings in Keras" Example #5826

Closed **MadLily** opened this issue on Mar 16 · 4 comments

Assignees

No one assign

Labels

None yet

Projects

None yet

Same commit, different results #770

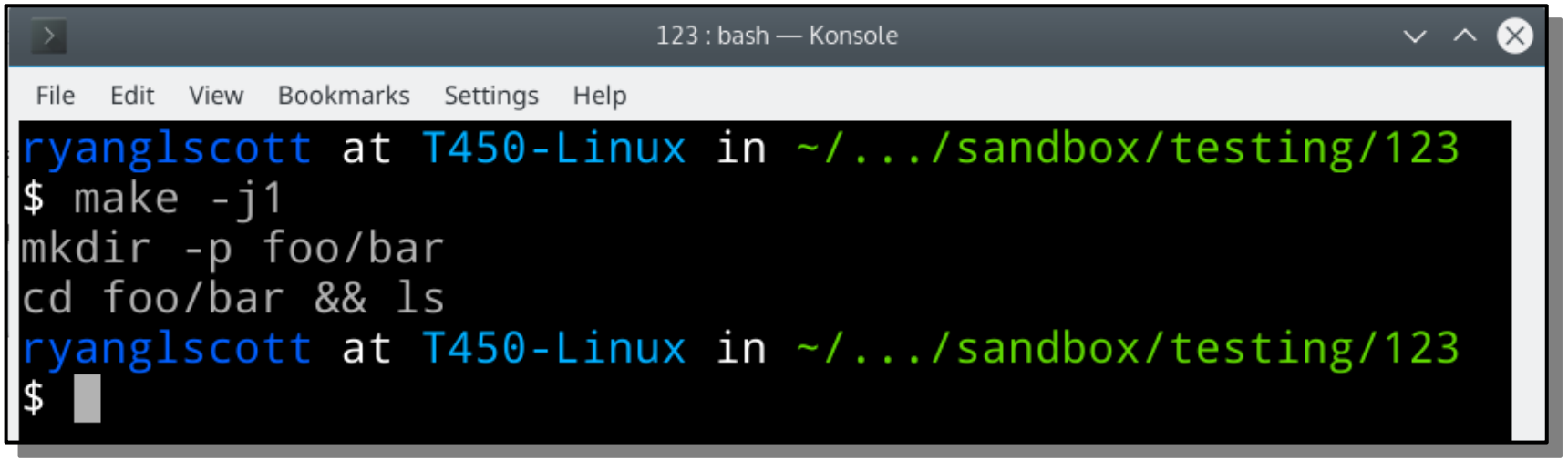
Closed **amitaibu** opened this issue on Nov 19, 2012 · 2 comments

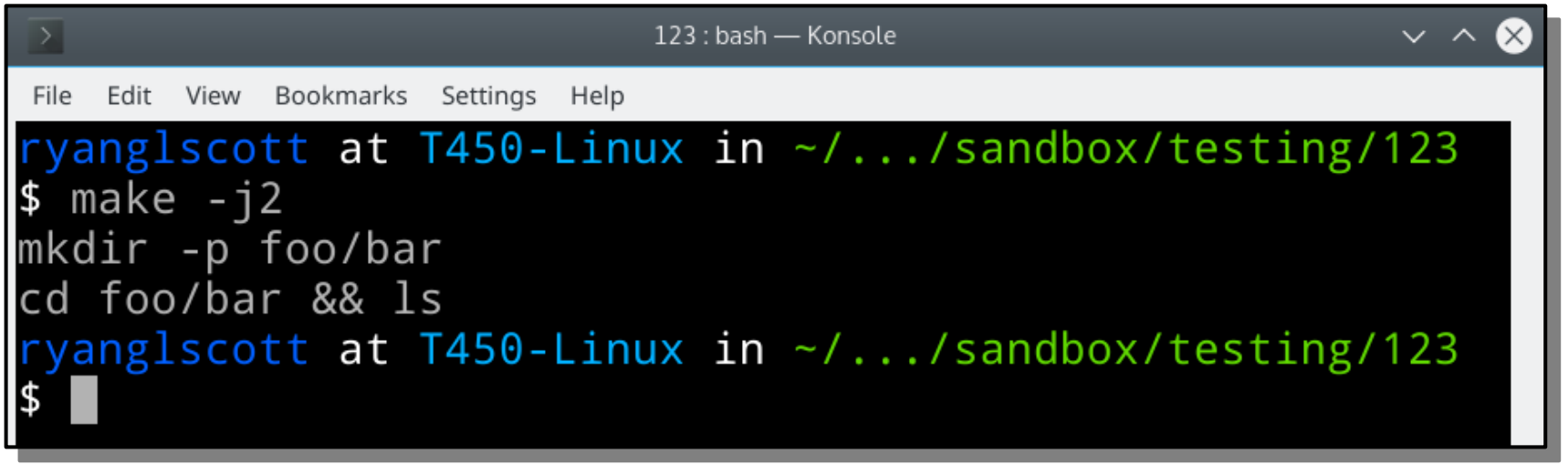
amitaibu commented on Nov 19, 2012

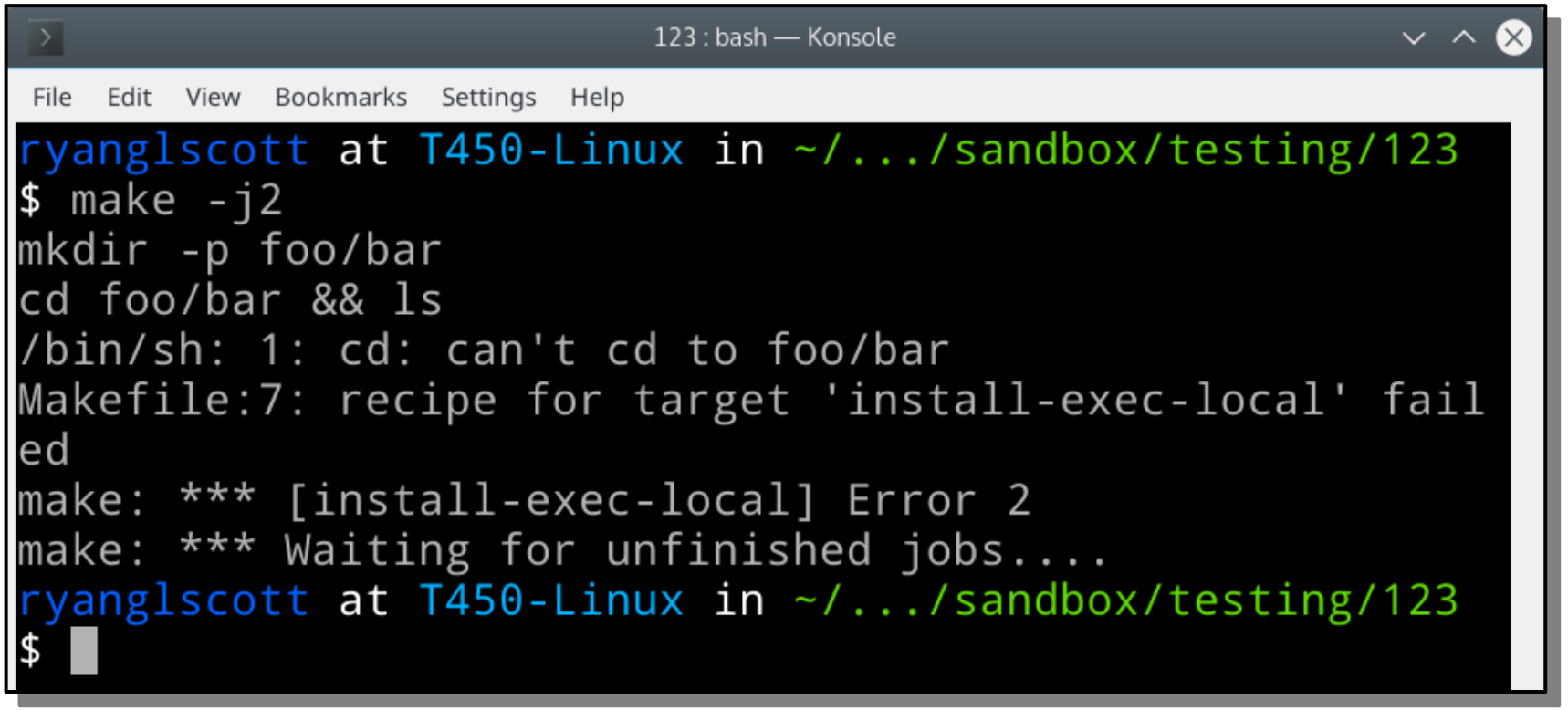
I have a build that was successful, but after "rebuilding" it fails.

Success VS Fail -- so I assume it's related to the environment?

```
all: create-bindir install-exec-local

DESTDIR=foo
bindir=bar

install-exec-local:
	cd $(DESTDIR)/$(bindir) && ls
create-bindir:
	mkdir -p $(DESTDIR)/$(bindir)
```

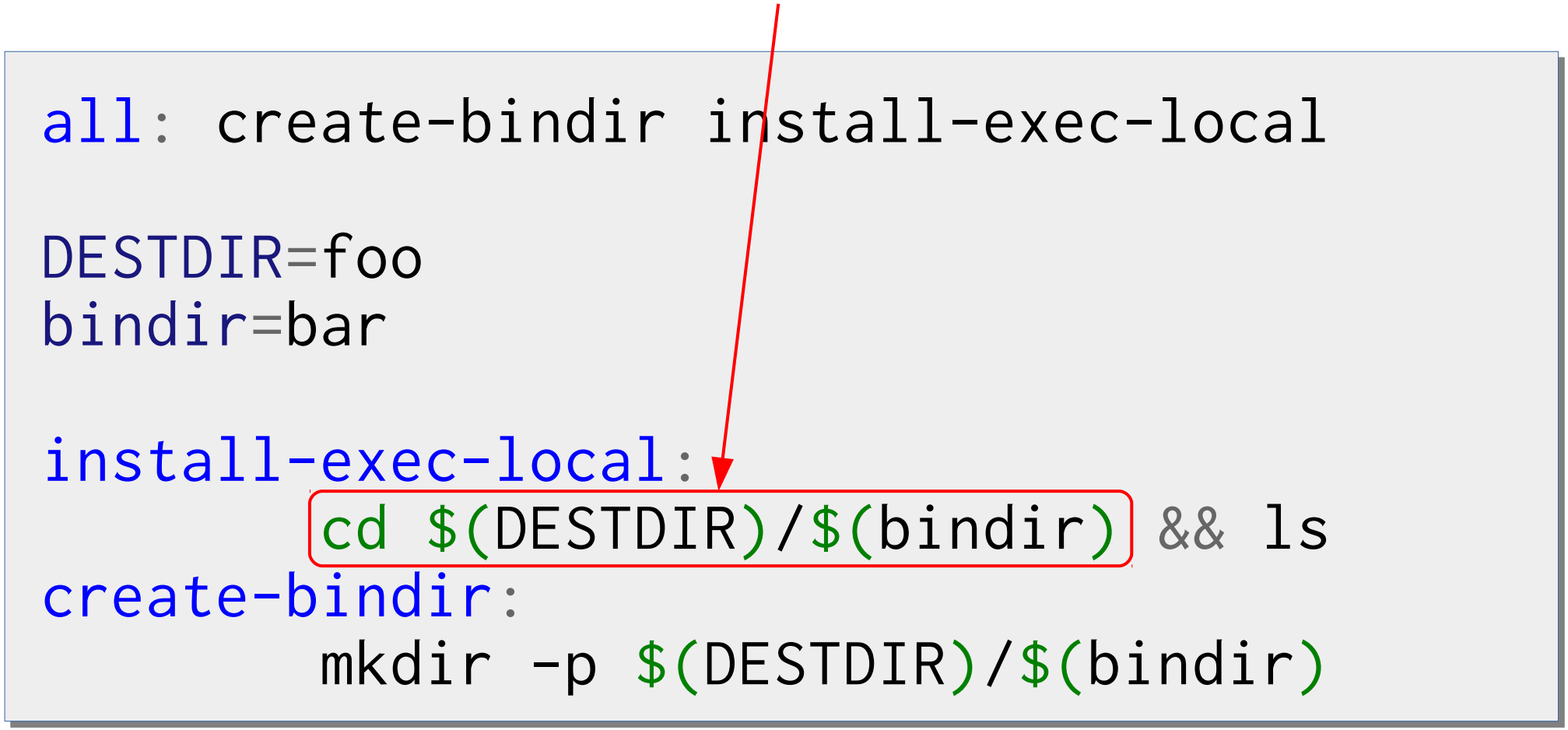

123 : bash — Konsole
File Edit View Bookmarks Settings Help
ryanglscott at T450-Linux in ~/.../sandbox/testing/123
$ make -j1
mkdir -p foo/bar
cd foo/bar && ls
ryanglscott at T450-Linux in ~/.../sandbox/testing/123
$


123 : bash — Konsole
File Edit View Bookmarks Settings Help
ryanglscott at T450-Linux in ~/.../sandbox/testing/123
$ make -j2
mkdir -p foo/bar
cd foo/bar && ls
ryanglscott at T450-Linux in ~/.../sandbox/testing/123
$


123 : bash — Konsole
File Edit View Bookmarks Settings Help
ryanglscott at T450-Linux in ~/.../sandbox/testing/123
$ make -j2
mkdir -p foo/bar
cd foo/bar && ls
/bin/sh: 1: cd: can't cd to foo/bar
Makefile:7: recipe for target 'install-exec-local' fail
ed
make: *** [install-exec-local] Error 2
make: *** Waiting for unfinished jobs....
ryanglscott at T450-Linux in ~/.../sandbox/testing/123
$

```
all: create-bindir install-exec-local

DESTDIR=foo
bindir=bar

install-exec-local:
	cd $(DESTDIR)/$(bindir) && ls
create-bindir:
	mkdir -p $(DESTDIR)/$(bindir)
```

Race condition!

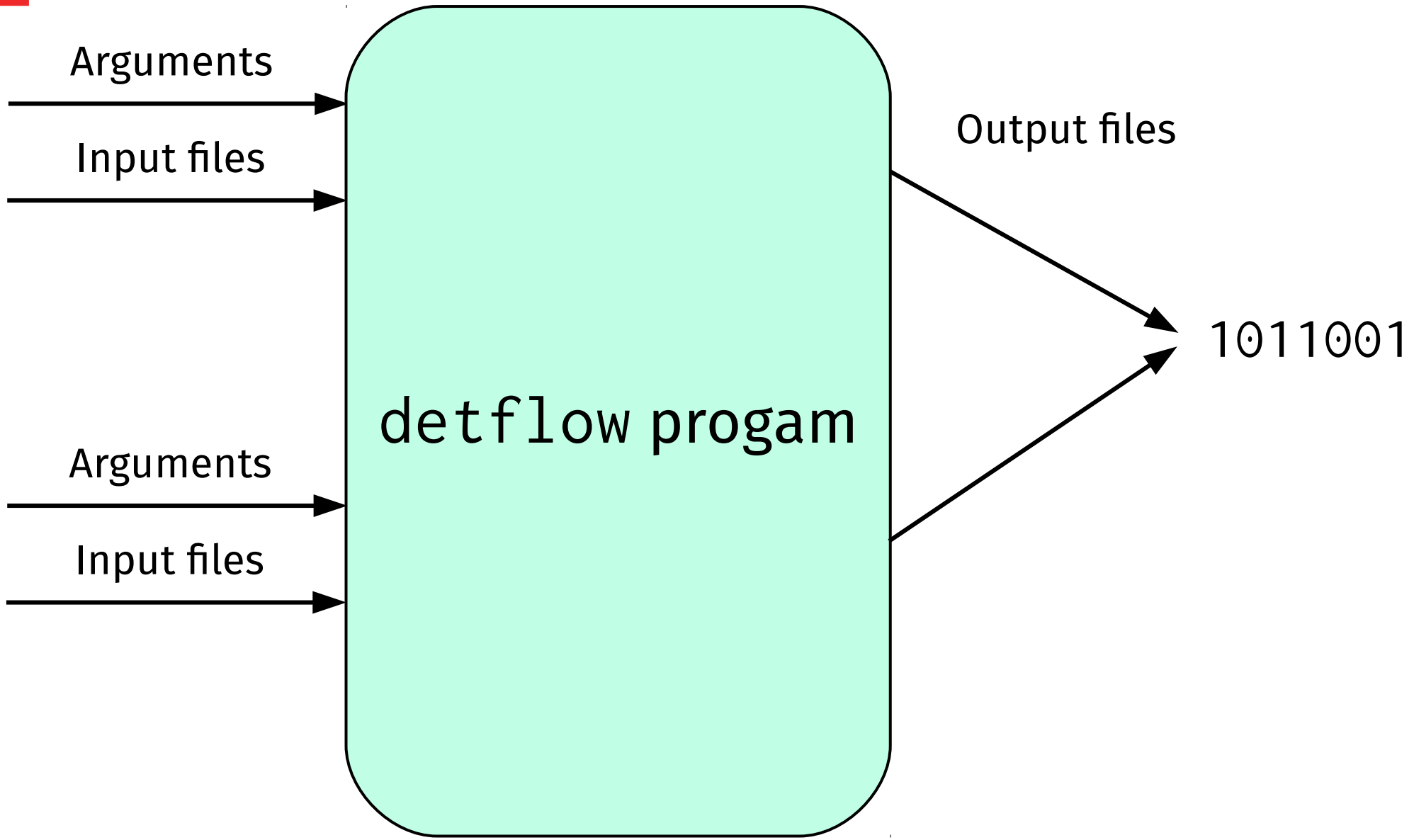


```
all: create-bindir install-exec-local

DESTDIR=foo
bindir=bar

install-exec-local:
        cd $(DESTDIR)/$(bindir) && ls
create-bindir:
        mkdir -p $(DESTDIR)/$(bindir)
```

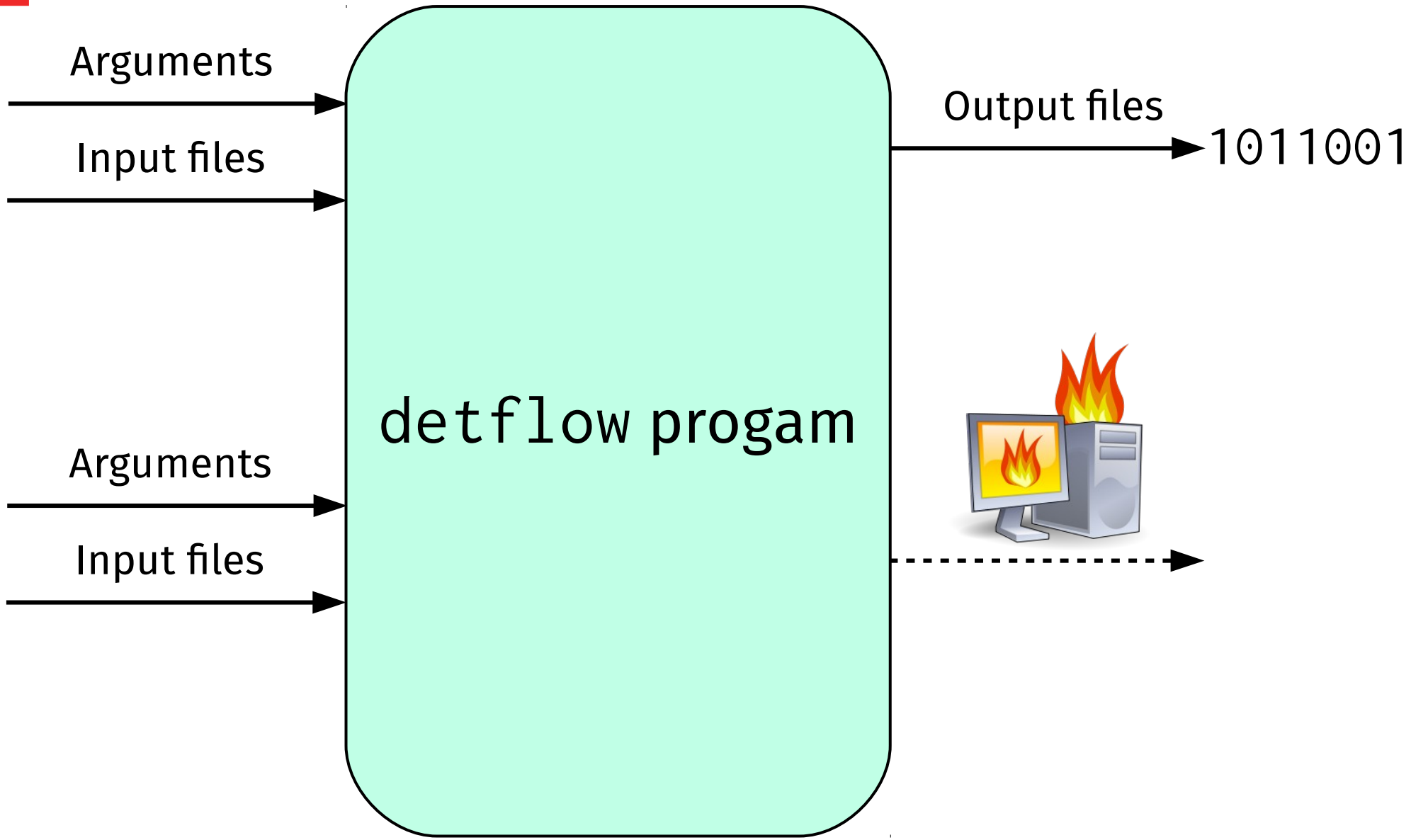
The `detflow` guarantee

If a program is invoked under detflow twice with identical inputs, and given sufficient machine resources to complete, then **both invocations will produce the same output**.

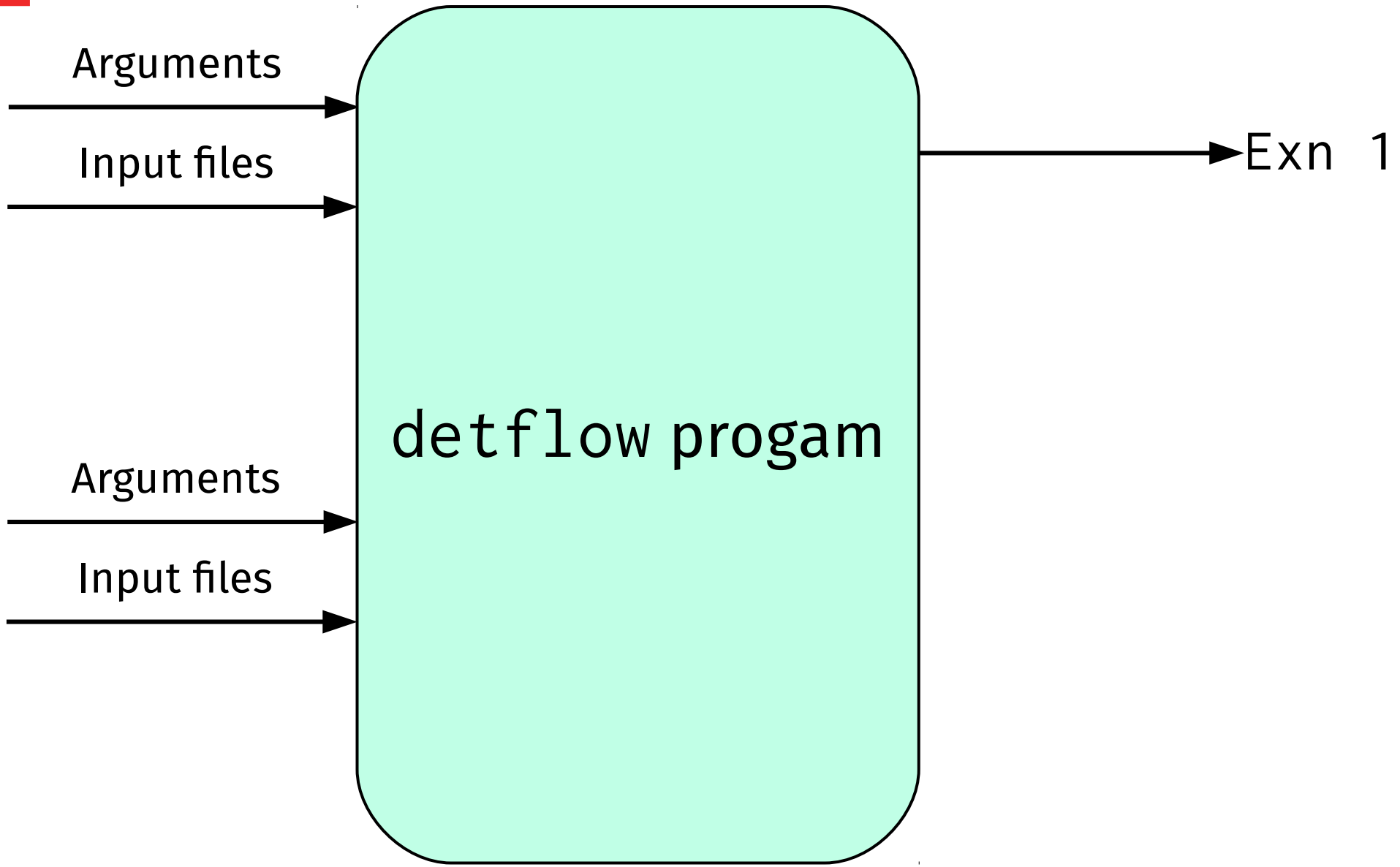
Quasideterminism



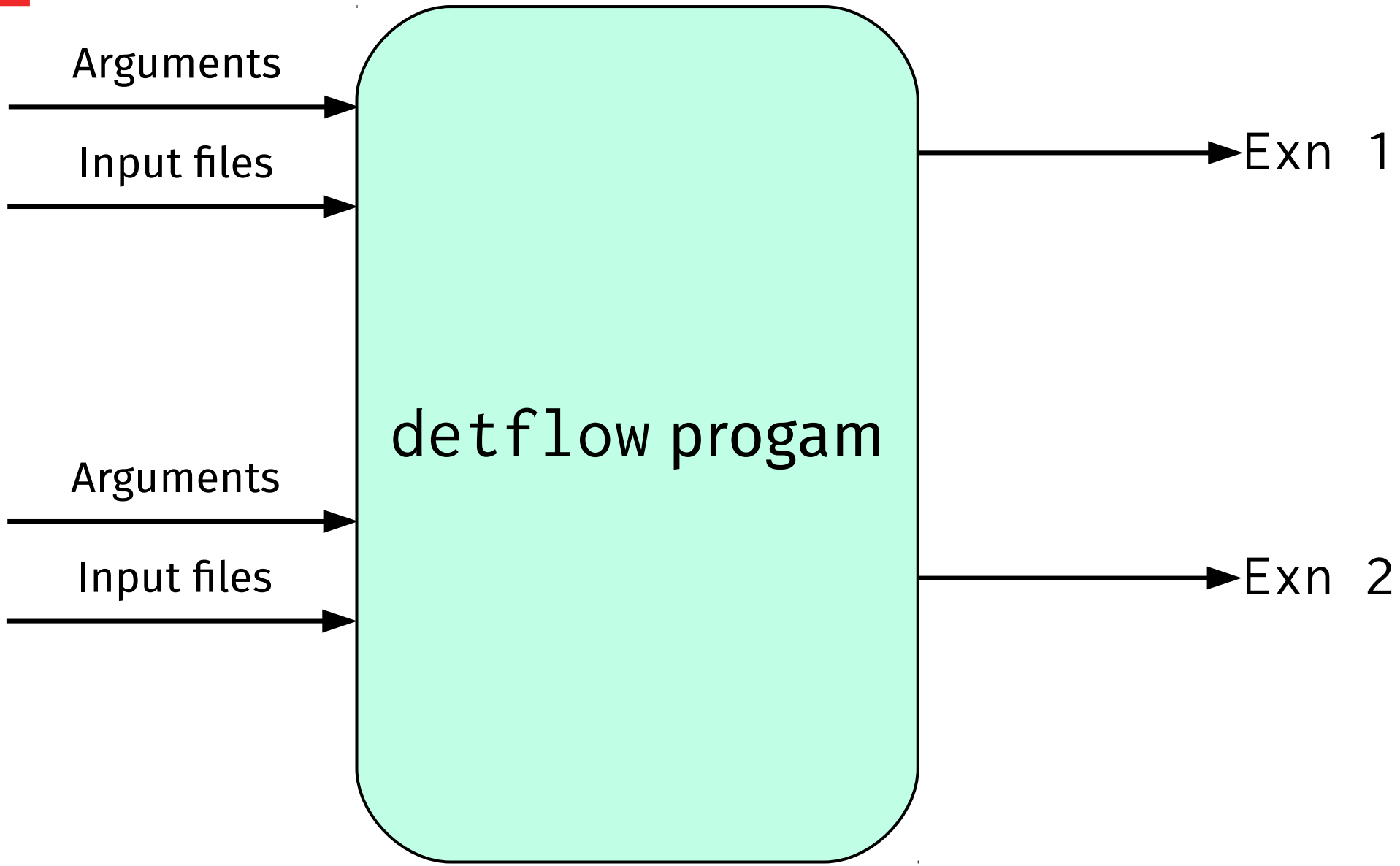
Quasideterminism



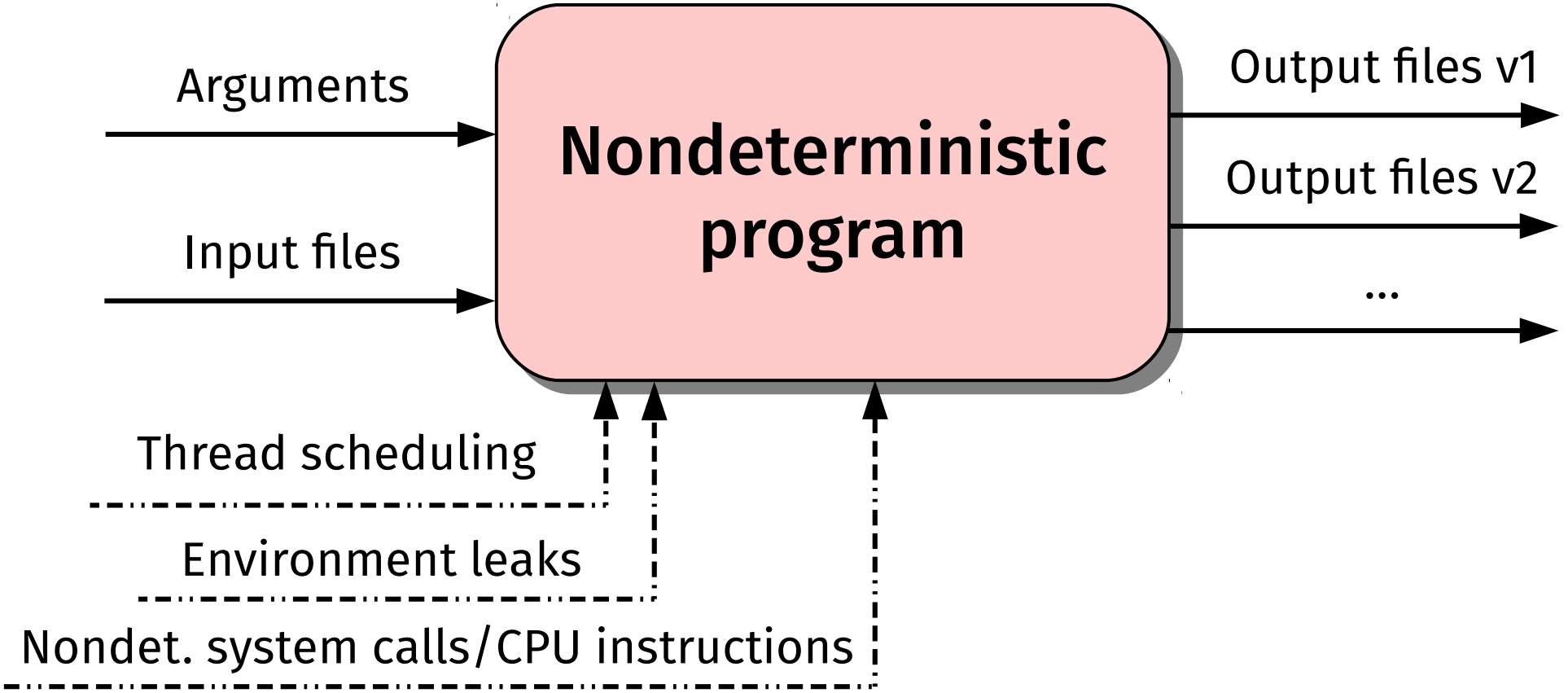
Quasideterminism

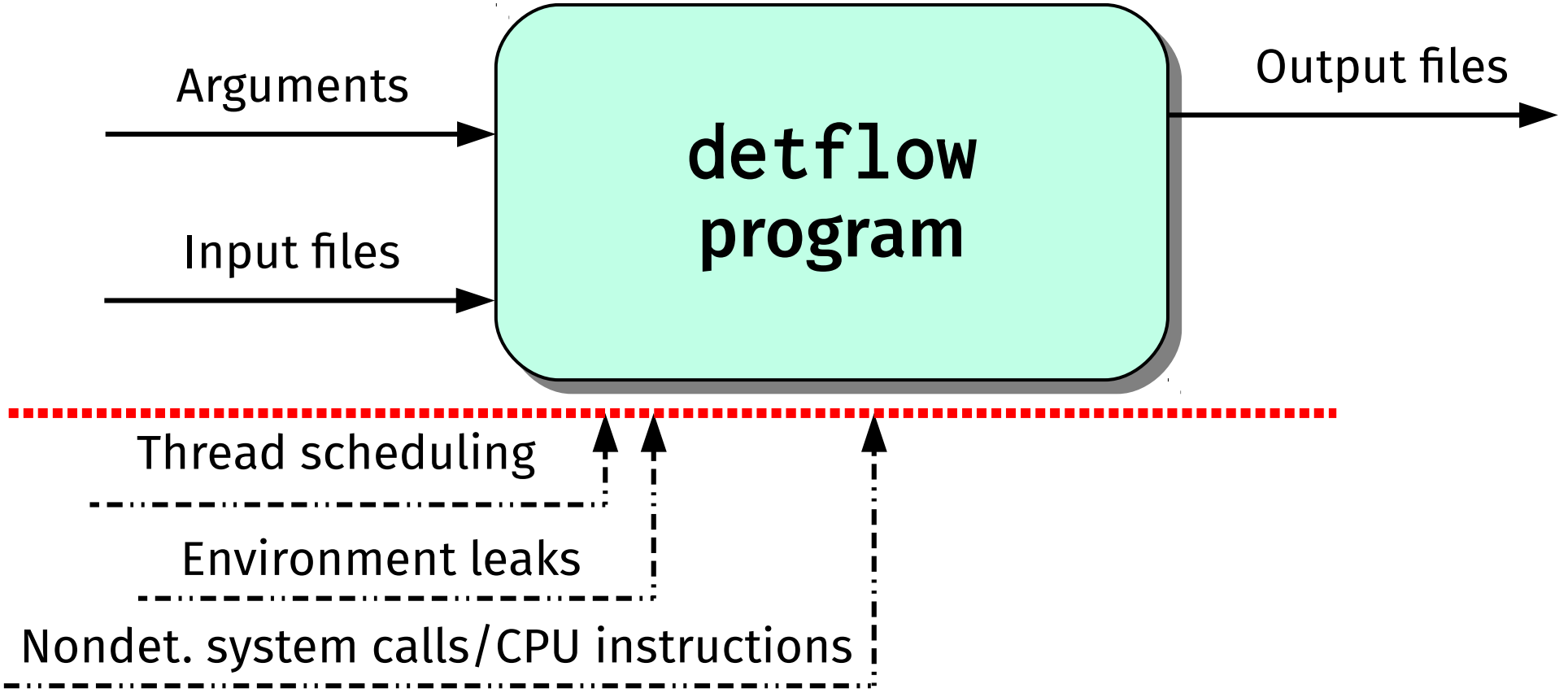


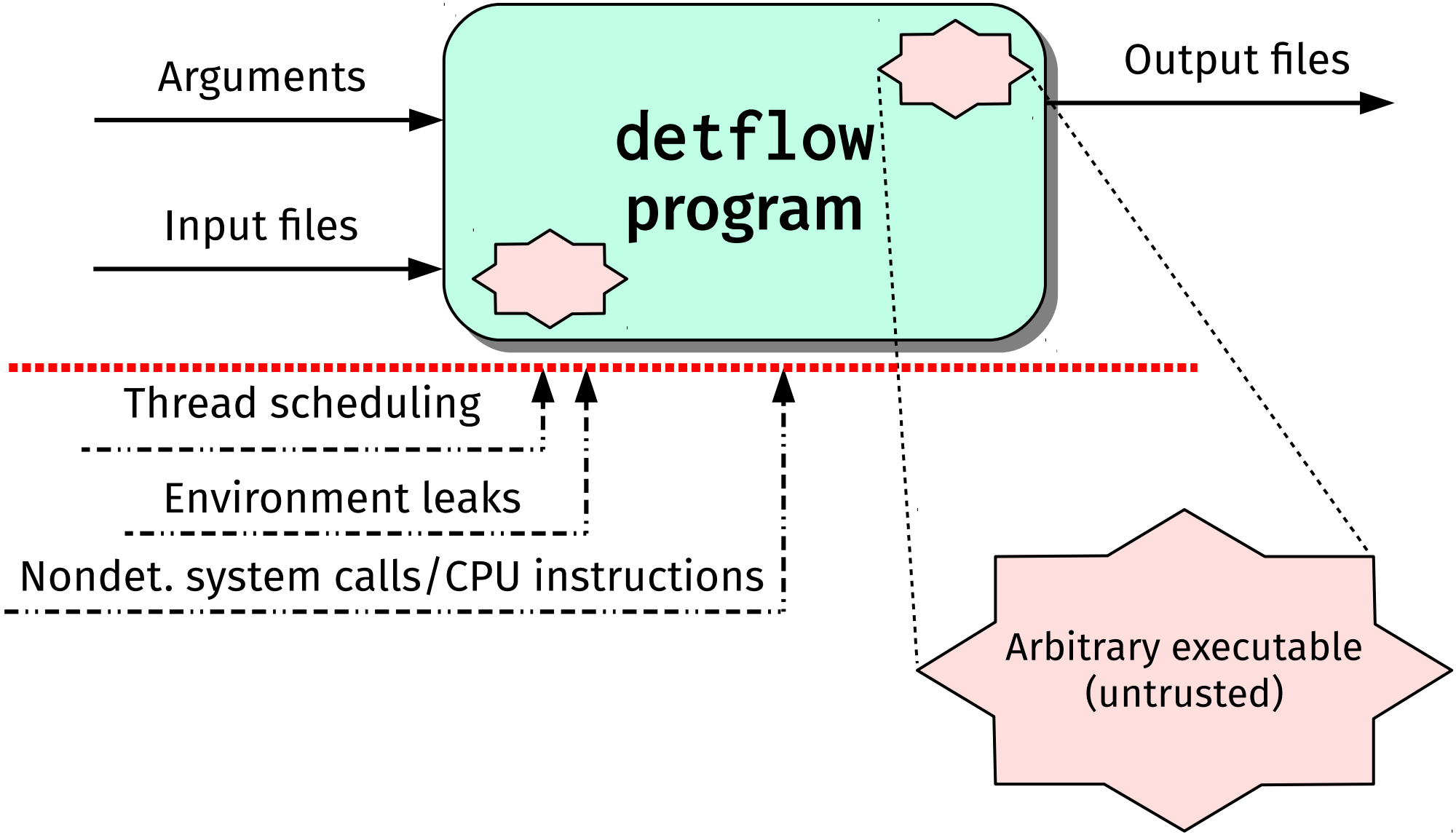
Quasideterminism

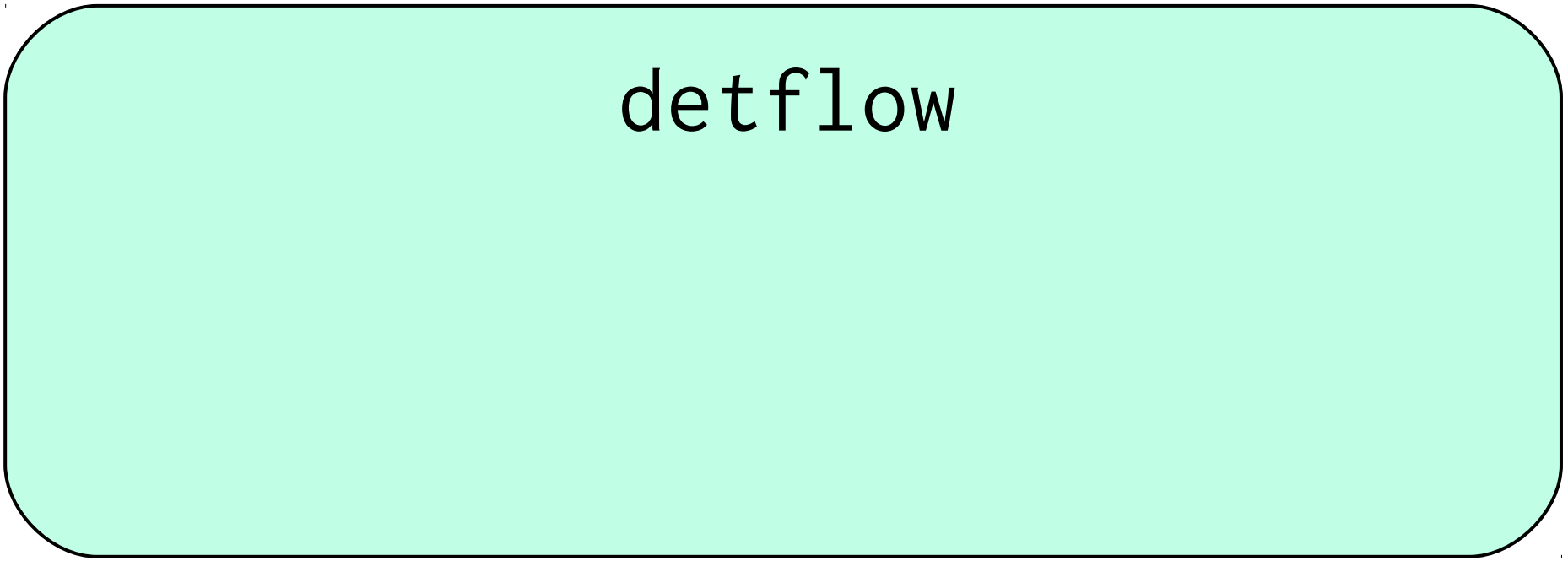



Arguments
Input files
Nondeterministic program
Output files v1
Output files v2
...

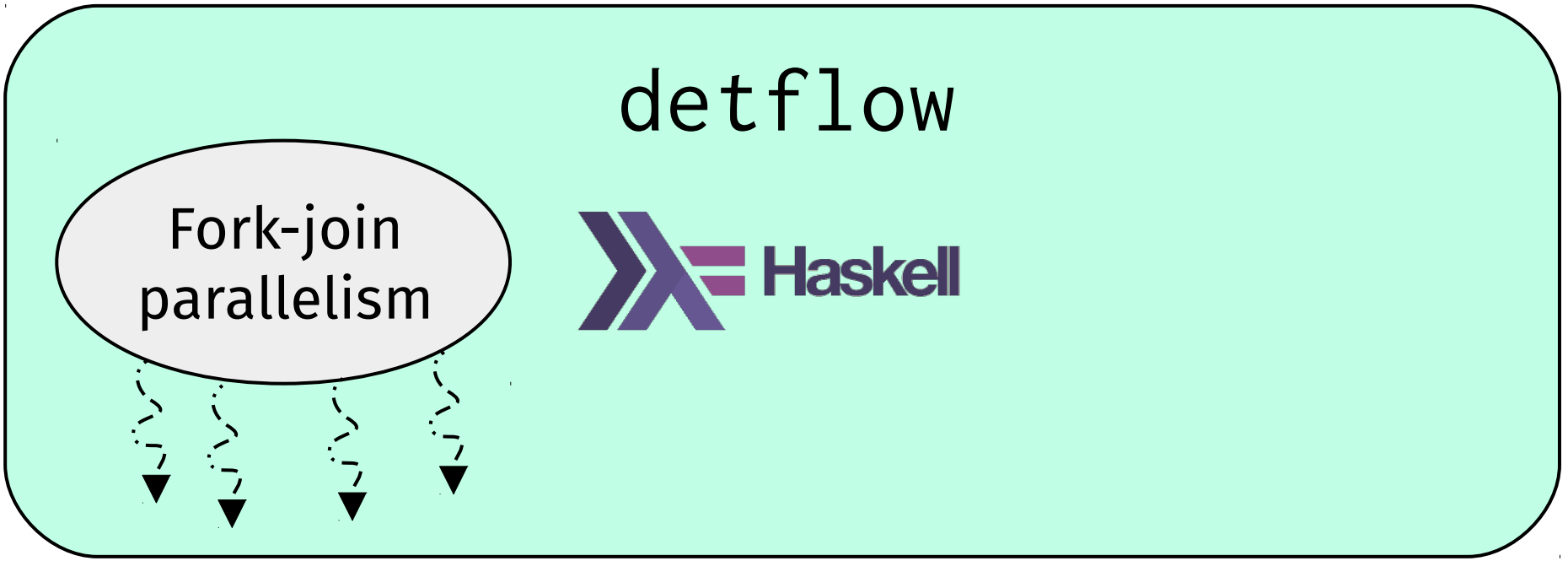

Arguments
Input files
Nondeterministic program
Output files v1
Output files v2
...
Thread scheduling
Environment leaks
Nondet. system calls/CPU instructions

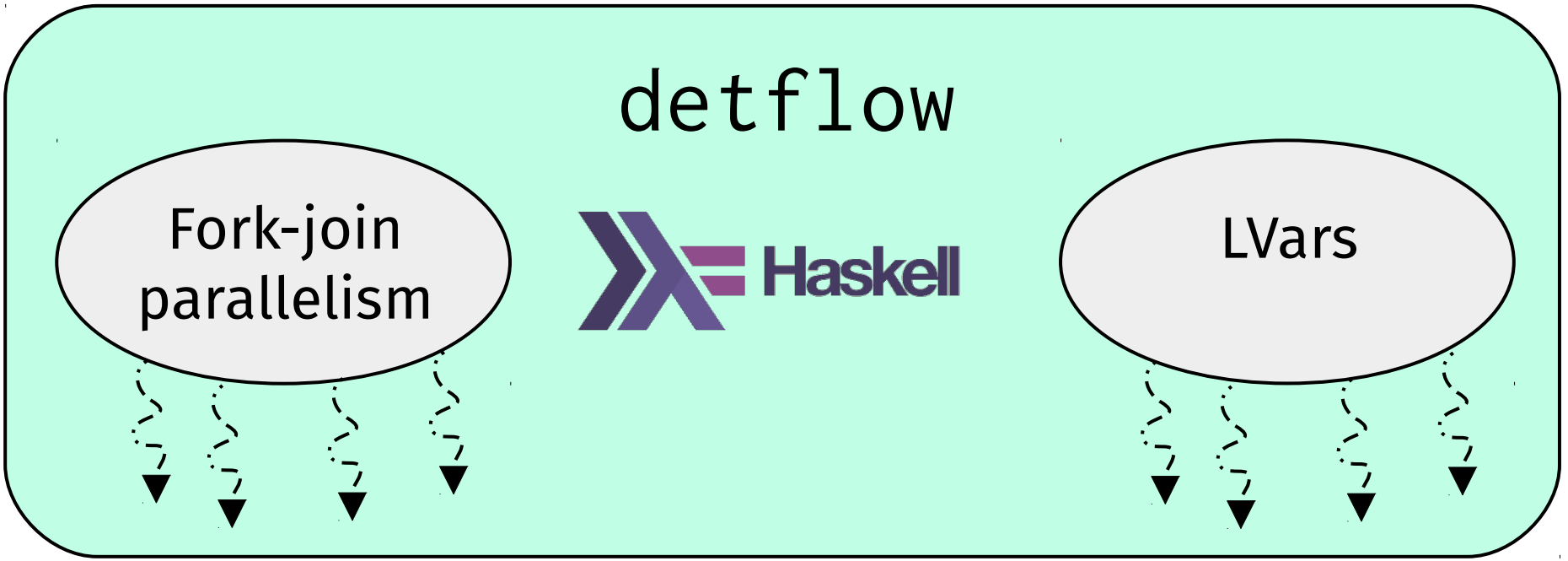

Arguments
Input files
detflow
program
Output files
Thread scheduling
Environment leaks
Nondet. system calls/CPU instructions

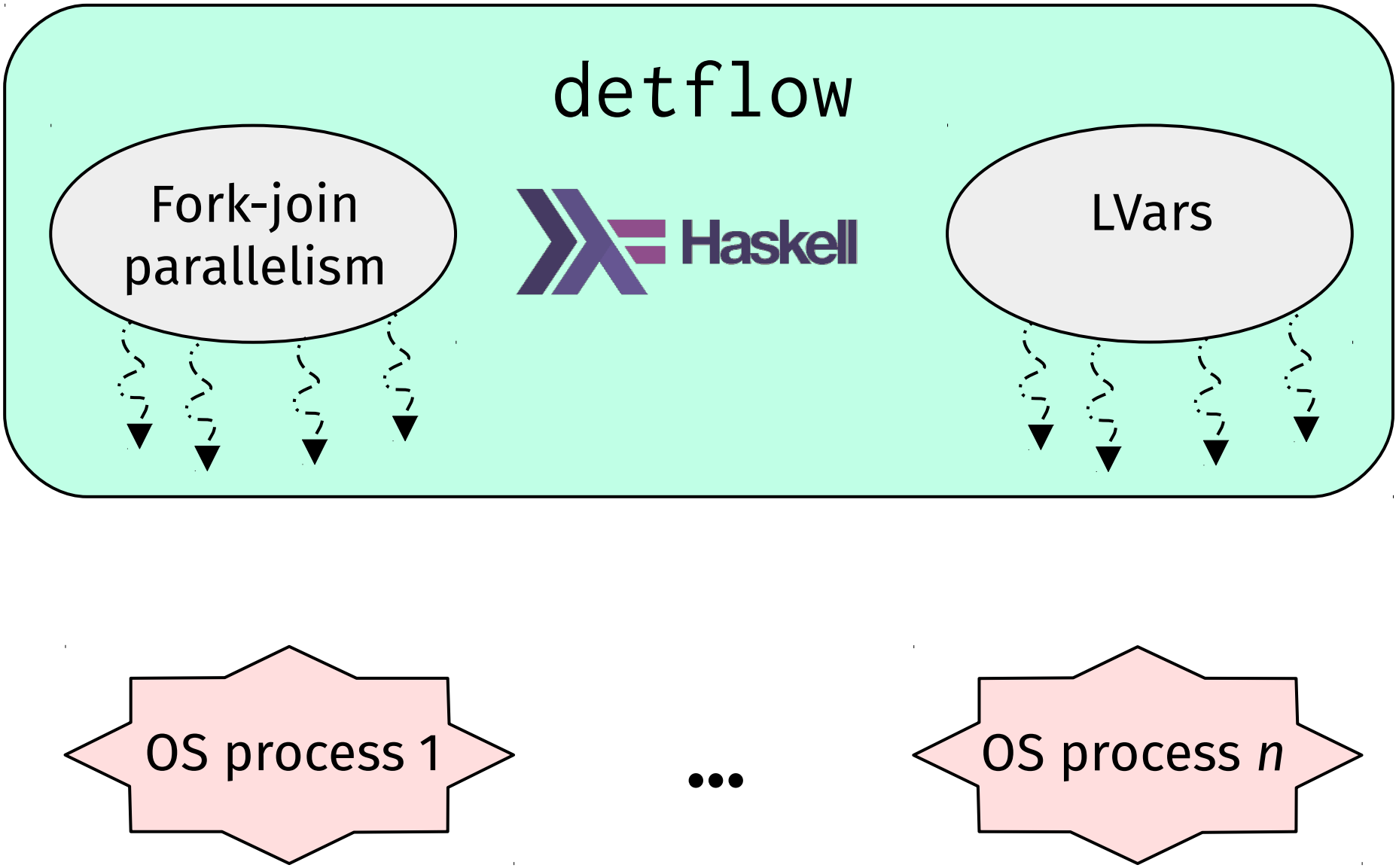

Arguments
Input files
detflow
program
Output files
Thread scheduling
Environment leaks
Nondet. system calls/CPU instructions
Arbitrary executable
(untrusted)

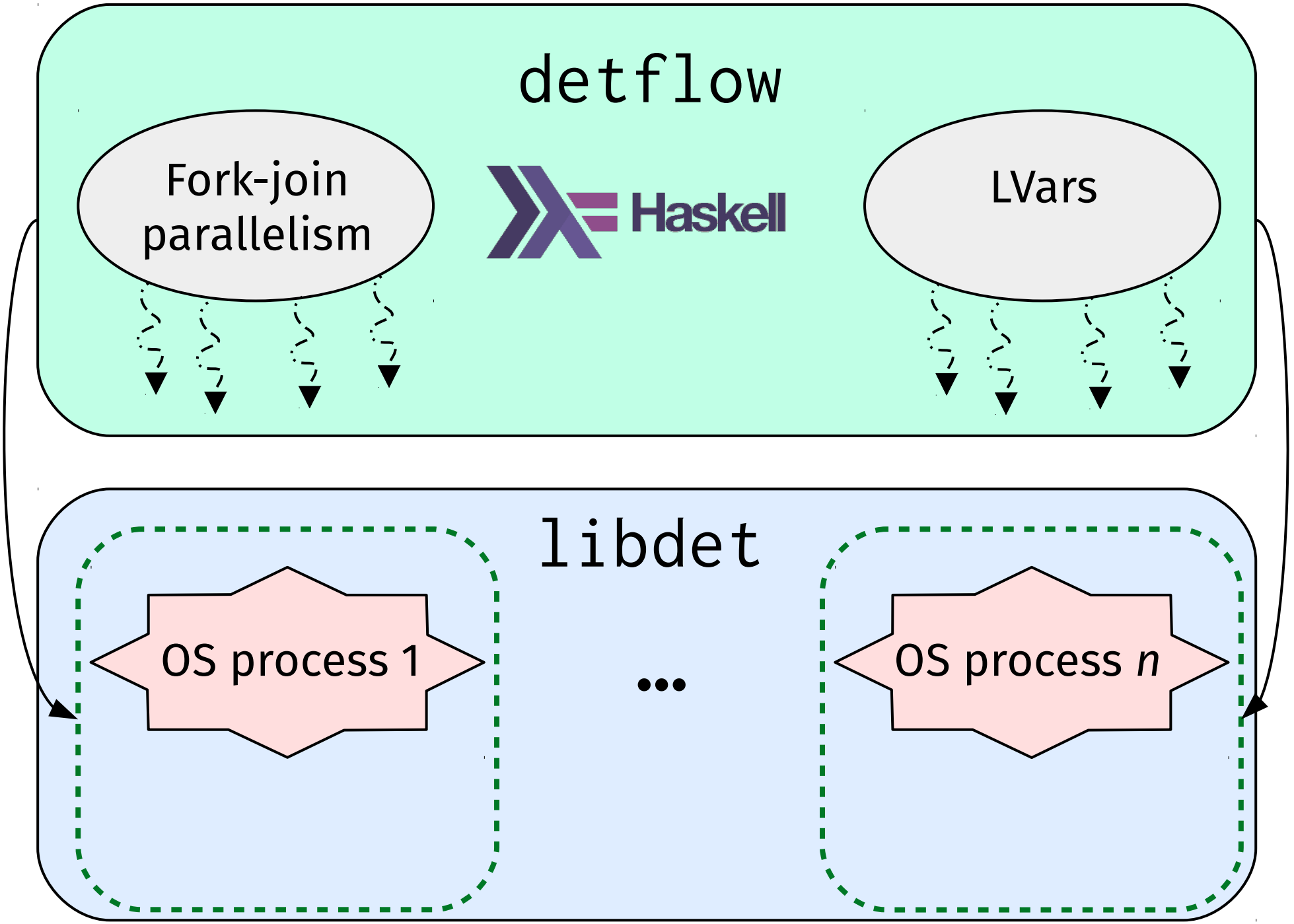

detflow

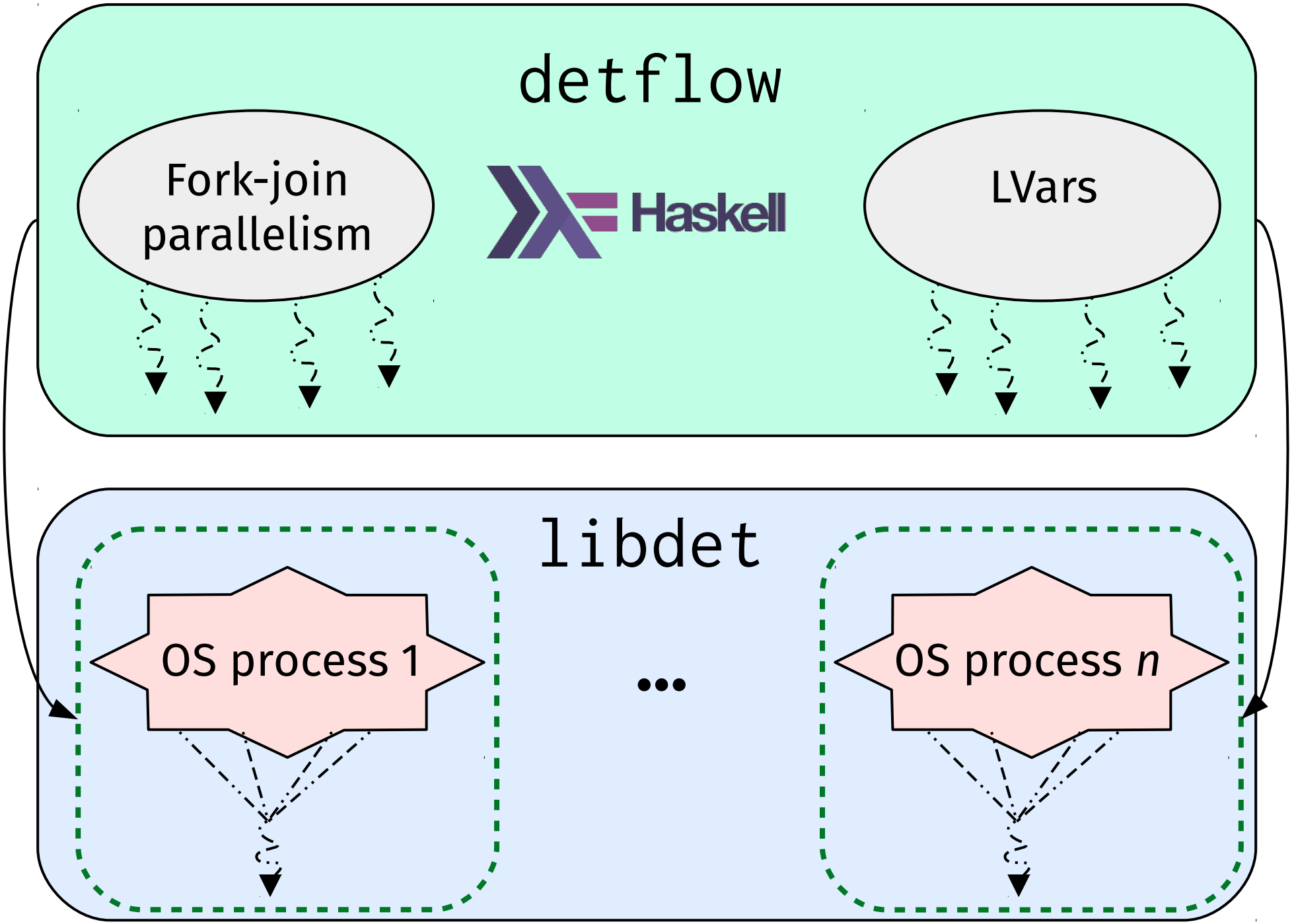
detflow

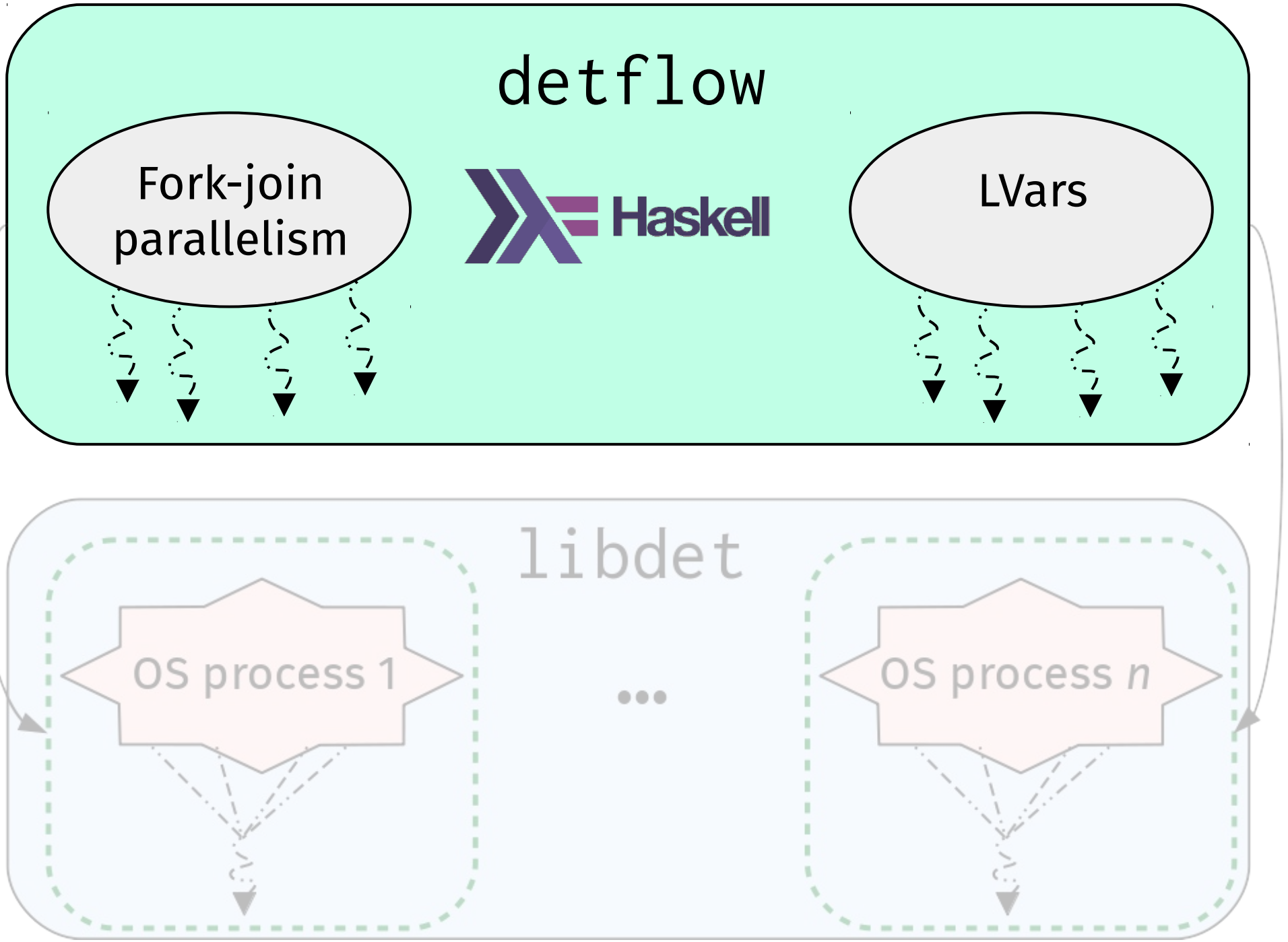

detflow
Fork-join
parallelism
Haskell


detflow
Fork-join
parallelism
Haskell
LVars


detflow
Fork-join parallelism
Haskell
LVars
OS process 1
...
OS process *n*


detflow
Fork-join parallelism
Haskell
LVars
libdet
OS process 1
...
OS process n


detflow
Fork-join parallelism
Haskell
LVars
libdet
OS process 1
...
OS process n


detflow
Fork-join parallelism
Haskell
LVars
libdet
OS process 1
...
OS process n

Traditional Haskell program

```
main :: IO ()
```

Traditional Haskell program

```haskell
main :: IO ()
     -- ^ Lots of ways to sneak in
     --   nondeterminism!
```

`detflow` Haskell programs

```
main :: DetIO ()
```

DetIO

```
data DetIO a -- Abstract
```

DetIO

```
data DetIO a -- Abstract
-- Expose only deterministic API calls
getLine  :: DetIO String
putStrLn :: String -> DetIO ()
-- etc.
```

DetIO

```
data DetIO a -- Abstract
-- Expose only deterministic API calls
getLine  :: DetIO String
putStrLn :: String -> DetIO ()
-- etc.
```

Key idea: Only expose deterministic operations that can be *composed* in a deterministic fashion

DetIO

```
data DetIO a -- Abstract
-- Expose only deterministic API calls
getLine  :: DetIO String
putStrLn :: String -> DetIO ()
-- etc.
```

```
main :: DetIO ()
main = do x <- getLine
          system (“gcc -c ” ++ x)
          putStrLn x
```

Parallel file access

- `detflow` uses the filesystem as a mutable, shared store
- Should this be allowed?

```
readFile  :: FilePath -> DetIO String
writeFile :: FilePath -> String
          -> DetIO String
fork      :: DetIO a -> DetIO (Thread a)
```

Problem: racing file access

Thread 1

```
do writeFile “foo.txt”
     “Hello, World”
```

Thread 2

```
do foo <- readFile “foo.txt”
   if foo == “Hello, World”
      then ...
      else ...
```

Problem: racing file access

Thread 1

```
do writeFile “foo.txt”
     “Hello, World”
```

Thread 2

```
do foo <- readFile “foo.txt”
   if foo == “Hello, World”
      then ...
      else ...
```

Solution: permissions

- Every thread holds separate permissions on system filepaths

Solution: permissions

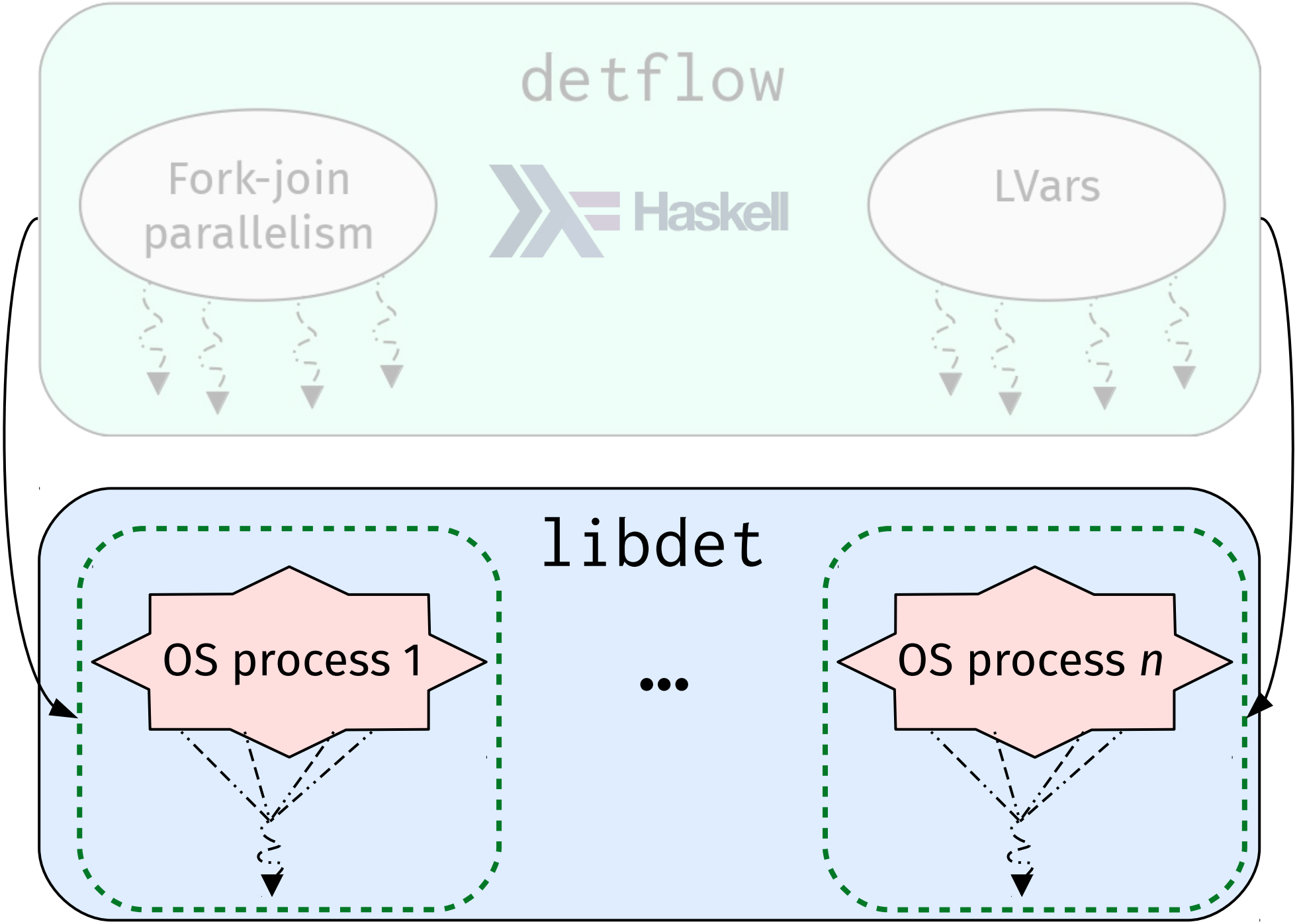
- Every thread holds separate permissions on system filepaths

	/abcdef/	ghijkl/	mnopqr
Thread 1	R	R	RW
Thread 2	R	R	

Parallel file access, revisited

```
data Perm -- (R/RW) + path

forkWPerms :: [PathPerm] -> DetIO a
           -> DetIO (Thread a)
```

- `readFile` and `writeFile` must respect the permissions in a thread's local state


detflow
Fork-join parallelism
Haskell
LVars
libdet
OS process 1
...
OS process *n*

`system` shell calls

```
system :: String -> DetIO ()

main :: DetIO ()
main = system “gcc foo.c -o foo”
```

libdet

`libdet` must intercept potential sources of nondeterminism at runtime.

libdet

libdet must intercept potential sources of nondeterminism at runtime.

Reading from “banned” directories

- /dev/urandom
- /proc
- etc.

libdet

libdet must intercept potential sources of nondeterminism at runtime.

Reading from “banned” directories

- /dev/urandom
- /proc
- etc.

Solution

- Intercept calls to fopen() (with LD_PRELOAD), error if they read anything blacklisted

libdet

libdet must intercept potential sources of nondeterminism at runtime.

Uncontrolled concurrency

- e.g., with pthreads

libdet

`libdet` must intercept potential sources of nondeterminism at runtime.

Uncontrolled concurrency

- e.g., with `pthreads`

Solution

- Intercept calls to `pthread_create()` (with `LD_PRELOAD`) to run everything sequentially

libdet

libdet must intercept potential sources of nondeterminism at runtime.

Nondeterministic OS properties

- e.g., reading addresses returned by `mmap()`

libdet

`libdet` must intercept potential sources of nondeterminism at runtime.

Nondeterministic OS properties

- e.g., reading addresses returned by `mmap()`

Solution

- Disable address-space layout randomization (ASLR)

libdet

`libdet` must intercept potential sources of nondeterminism at runtime.

Path operations with insufficient permissions

- e.g., reading `/foo` without read permissions on `/foo`

libdet

`libdet` must intercept potential sources of nondeterminism at runtime.

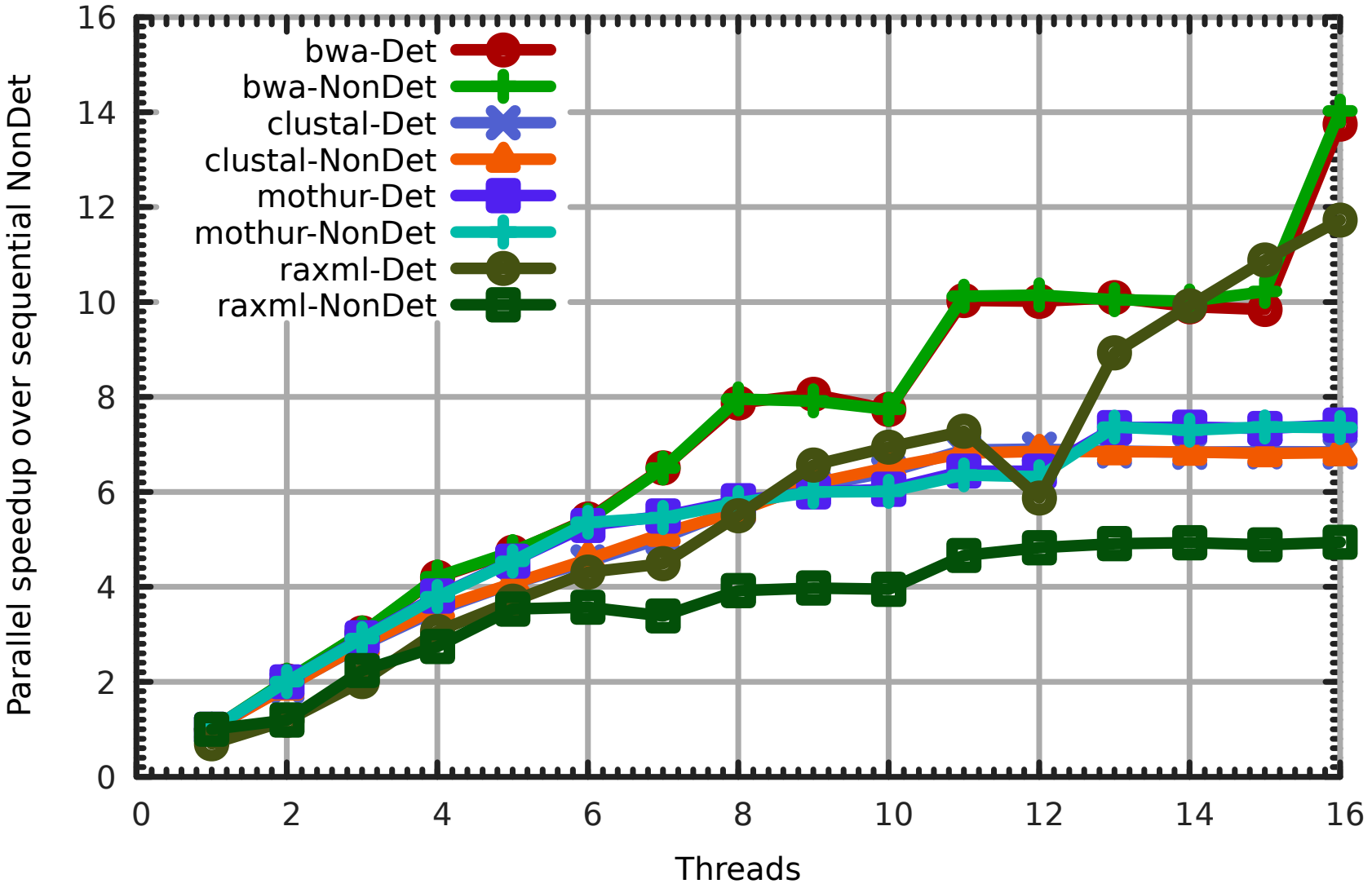
Path operations with insufficient permissions

- e.g., reading `/foo` without read permissions on `/foo`

Solution

- Inherit permissions from `detflow`!

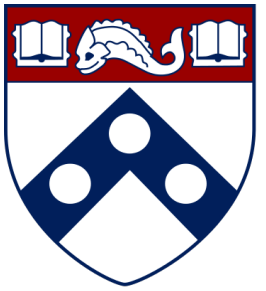
Bioinfo. apps, parallel speedup



(Higher is better)

Future work

- Reach closer to catching *all* sources of nondeterminism in runtime
- Dynamic (at-runtime) checkout of permissions
- Make more programs feasible to determinize



Penn

UNIVERSITY of PENNSYLVANIA

`detflow` can be used to construct and run parallel batch processing jobs deterministically (including legacy binaries) with less than 5% overhead.

Approach:

- Statically-typed root process: allows multithreading
- Each thread may shell out to legacy binaries: internally sequentialized by sandbox
- Legacy binaries can create subprocesses: also sequentialized
- Each thread and subprocess holds distinct file system permissions to prevent races

`https://github.com/iu-parfunc/detmonad`