2016.08.23

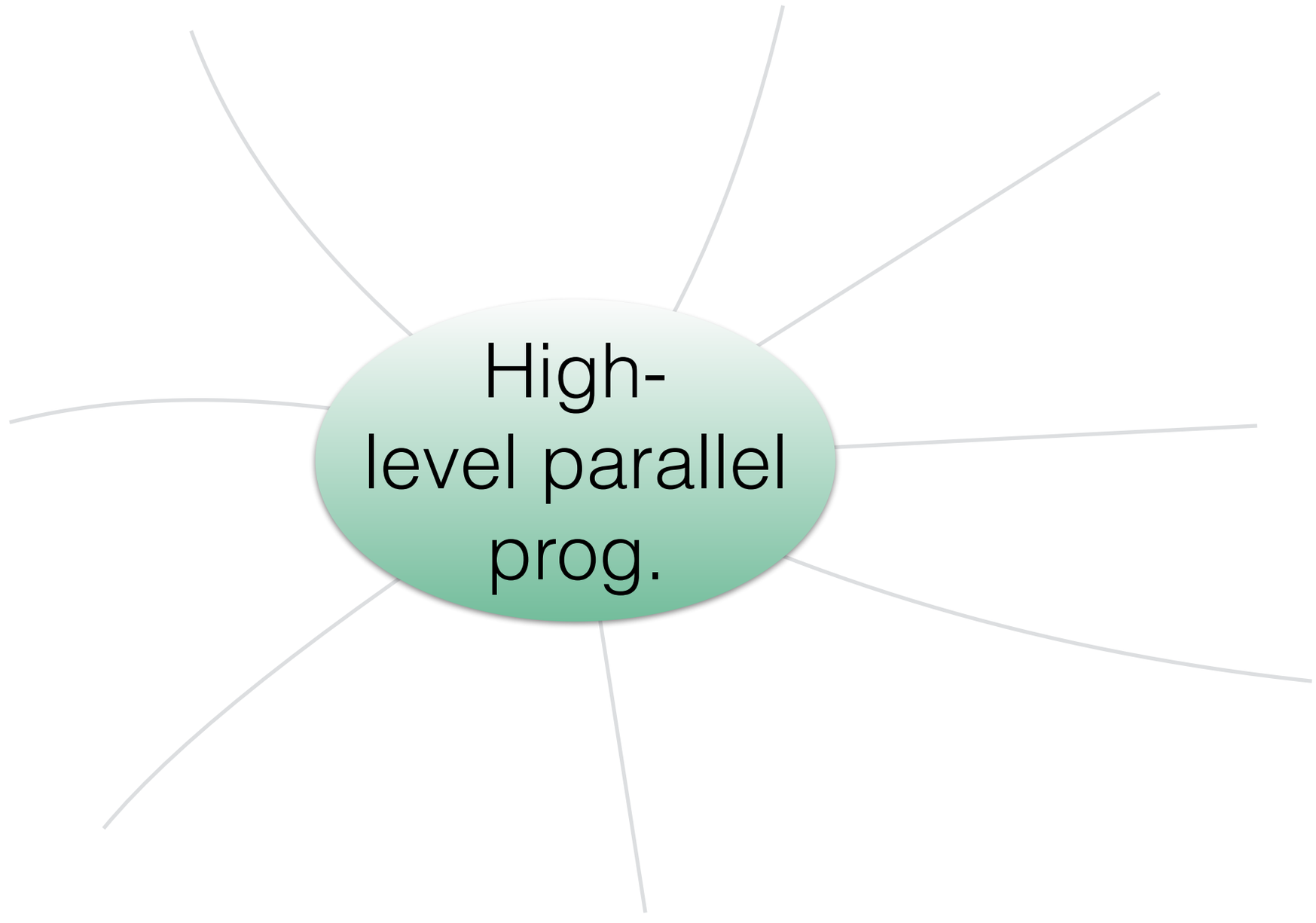
Open Problems from my perspective

INDIANA UNIVERSITY

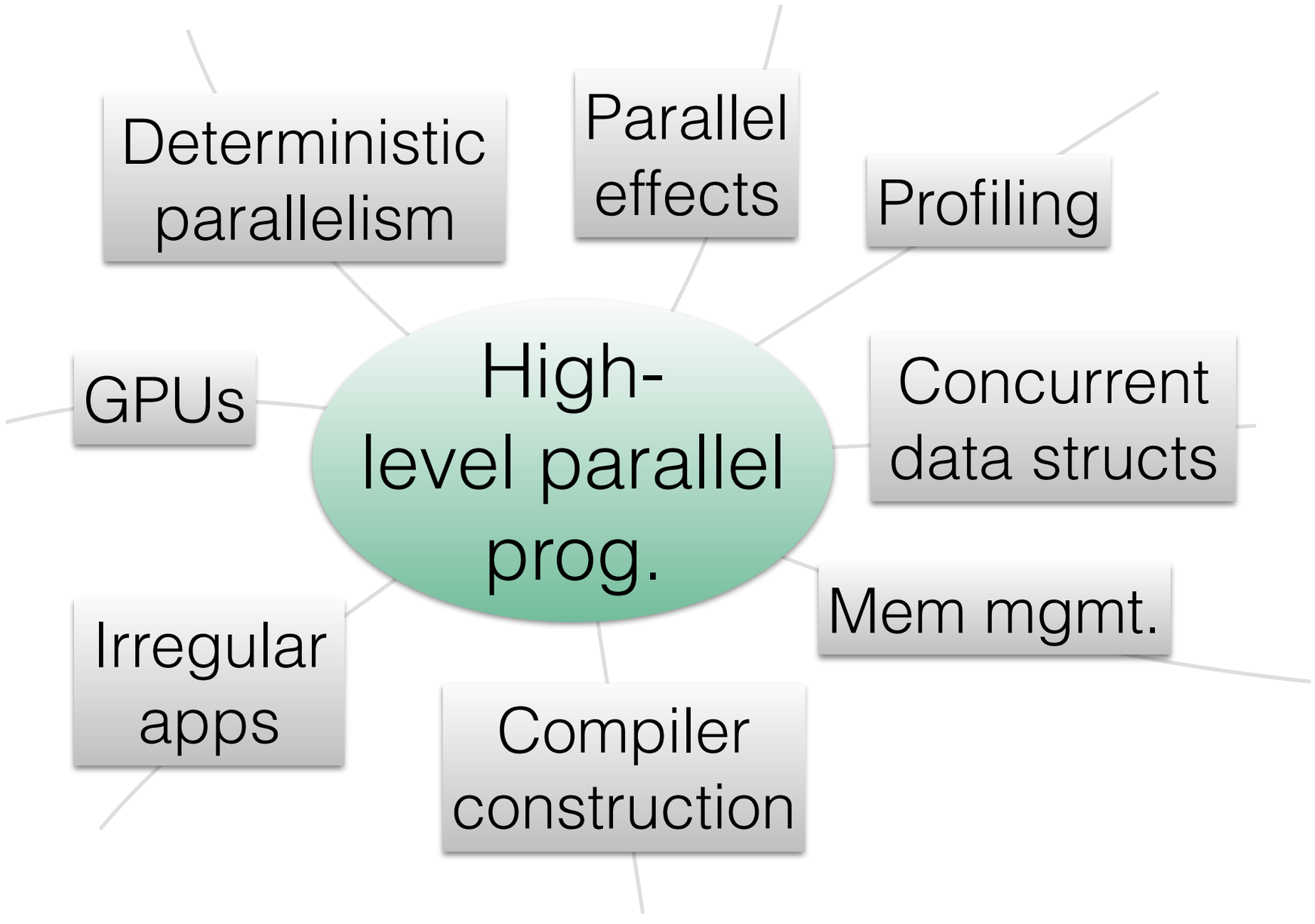
Our group (last 3 years)

Our group (last 3 years)

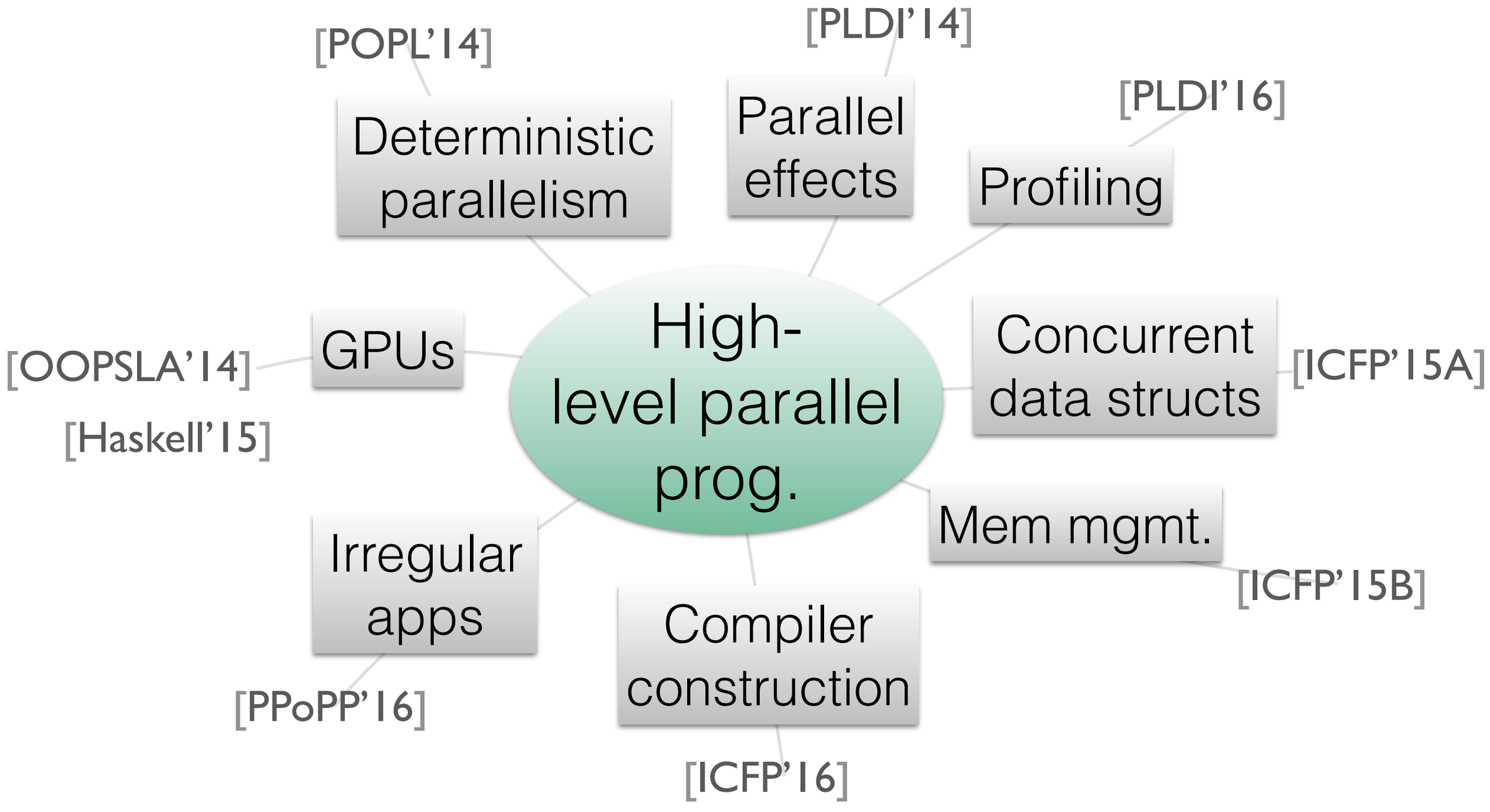
Our group (last 3 years)



Our group (last 3 years)

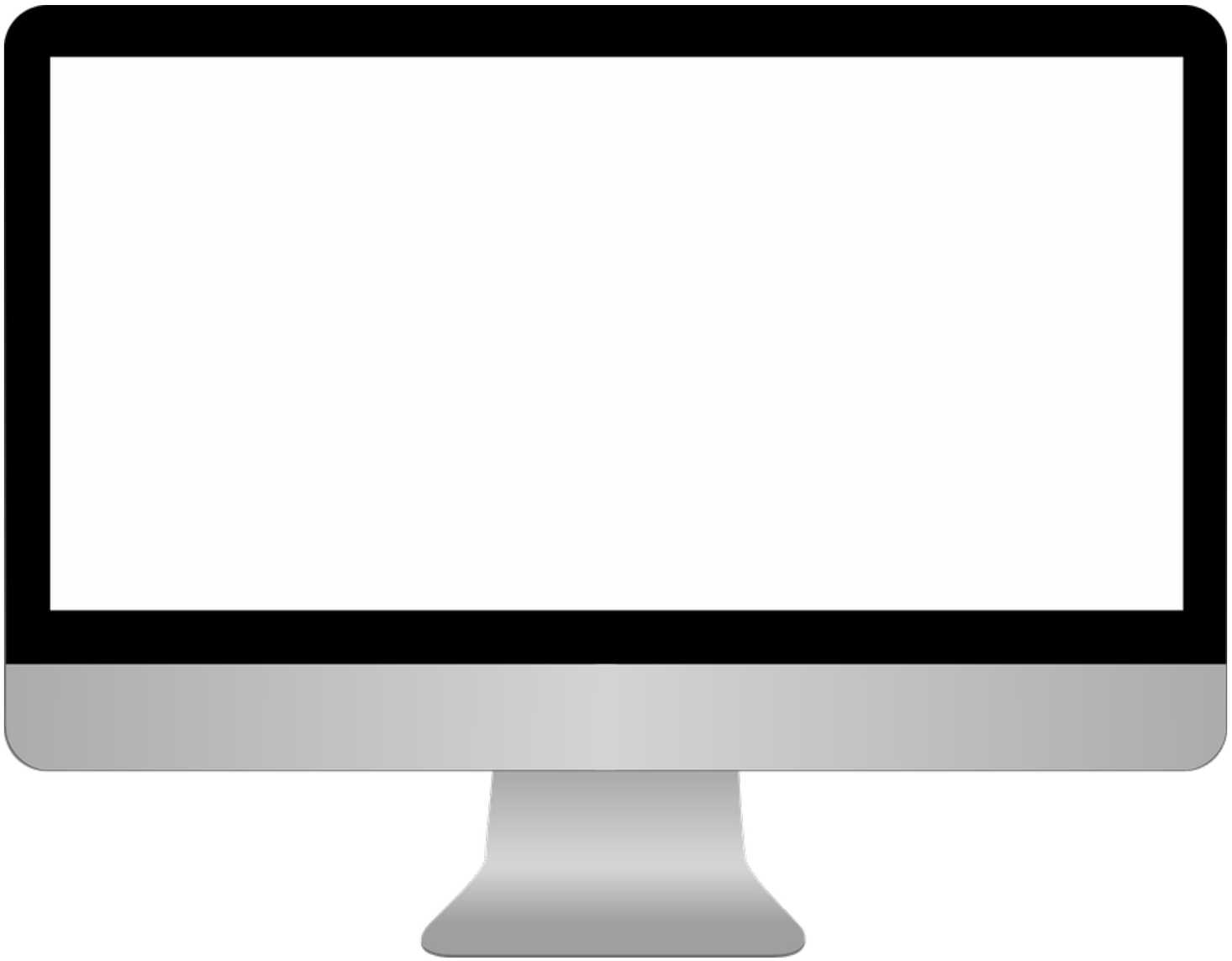


Our group (last 3 years)



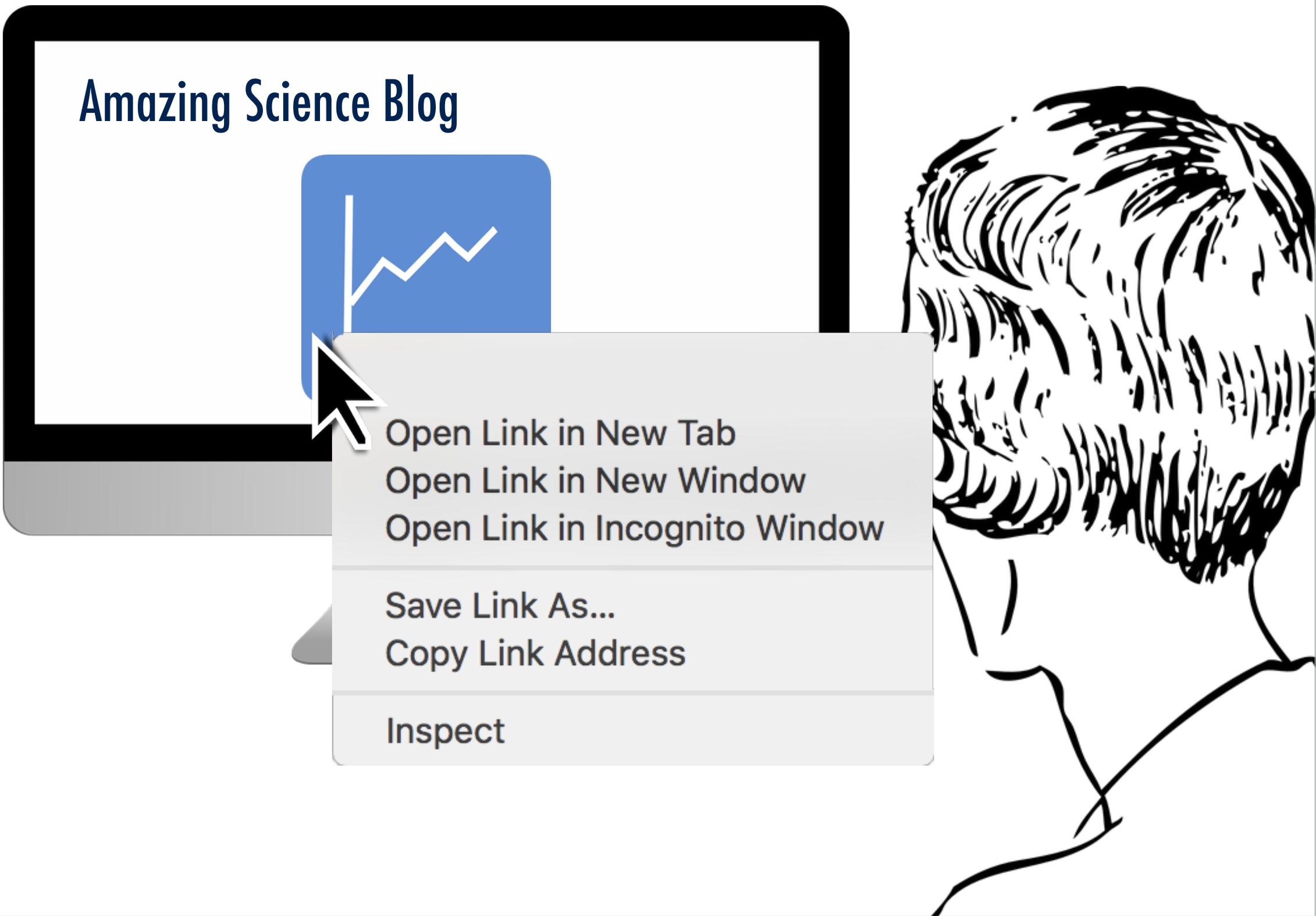
1. Archival-quality computation

1. Archival-quality computation

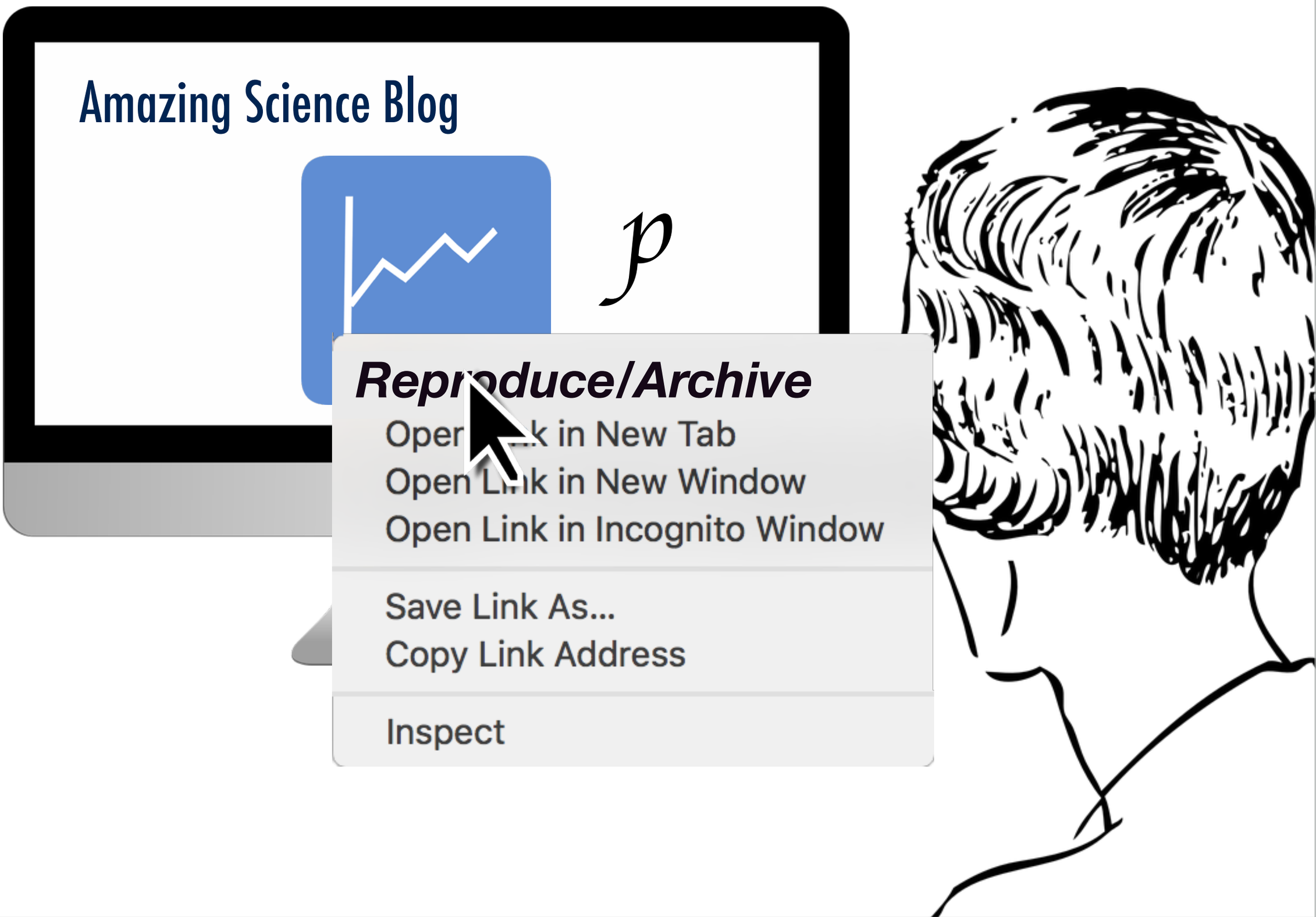


1. Archival-quality computation

1. Archival-quality computation

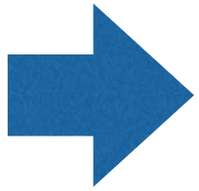


1. Archival-quality computation

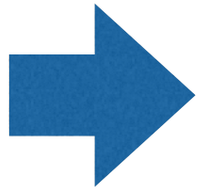


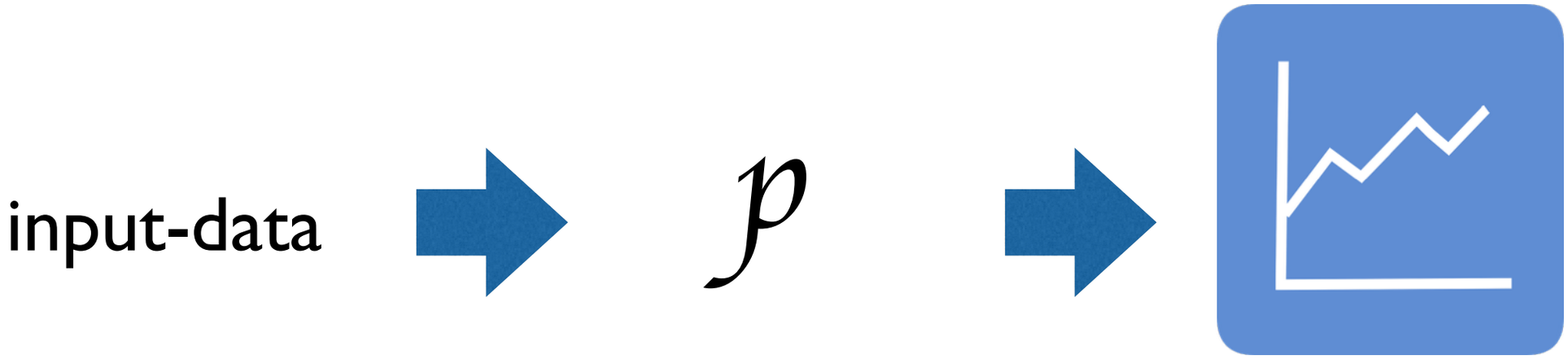
$$\mathcal{p}$$

input-data

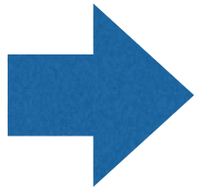
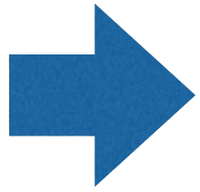


$\mathcal{P}$





Determinism Enforcement

input-data  $\mathcal{P}$  

Determinism Enforcement

<u>control</u>

- Environment

input-data → $\mathcal{P}$ → 

Determinism Enforcement

control

- Environment
- Execution

input-data → $\mathcal{P}$ → 

Determinism Enforcement

control

- Environment
- Execution
  - dynamic

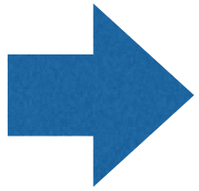
input-data → $\mathcal{P}$ → 

Determinism Enforcement

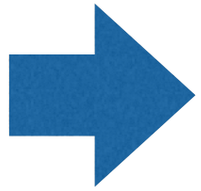
<u>control</u>

- Environment
- Execution
  - dynamic
  - static

input-data

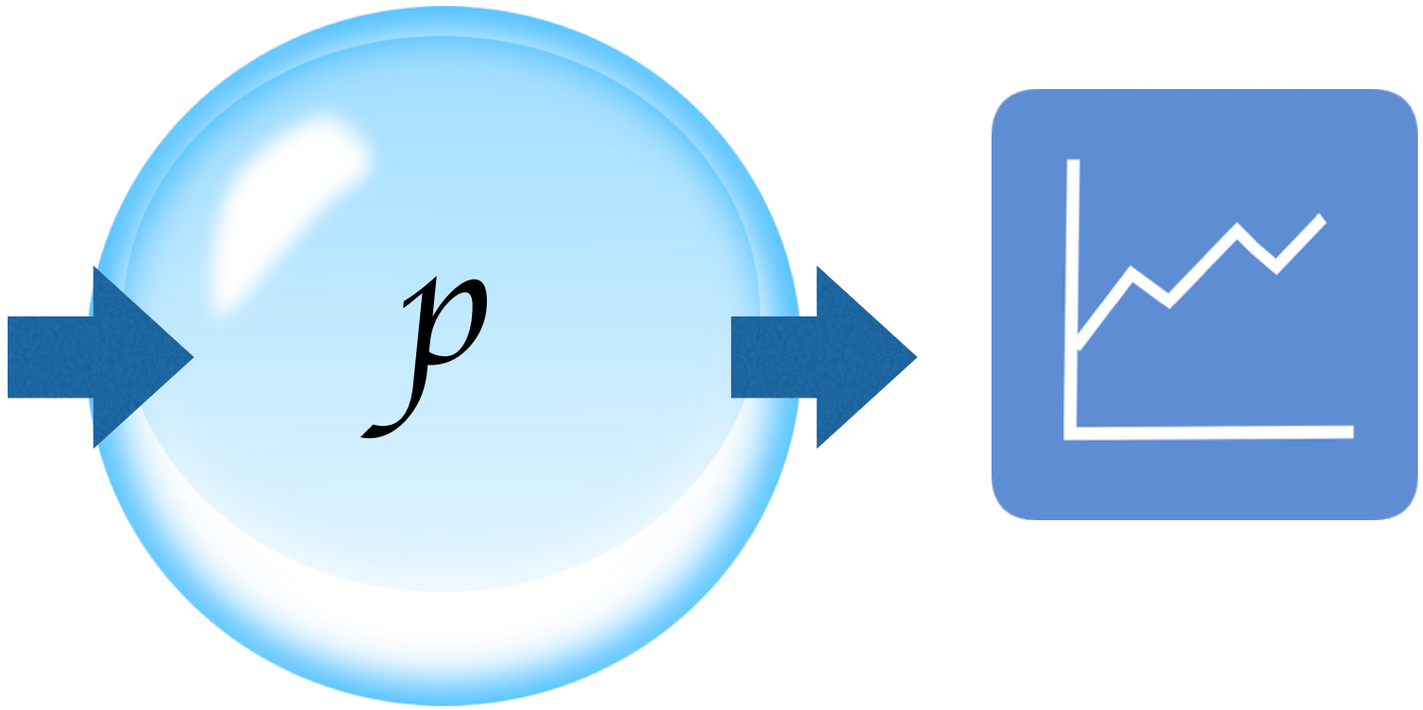


$\mathcal{P}$



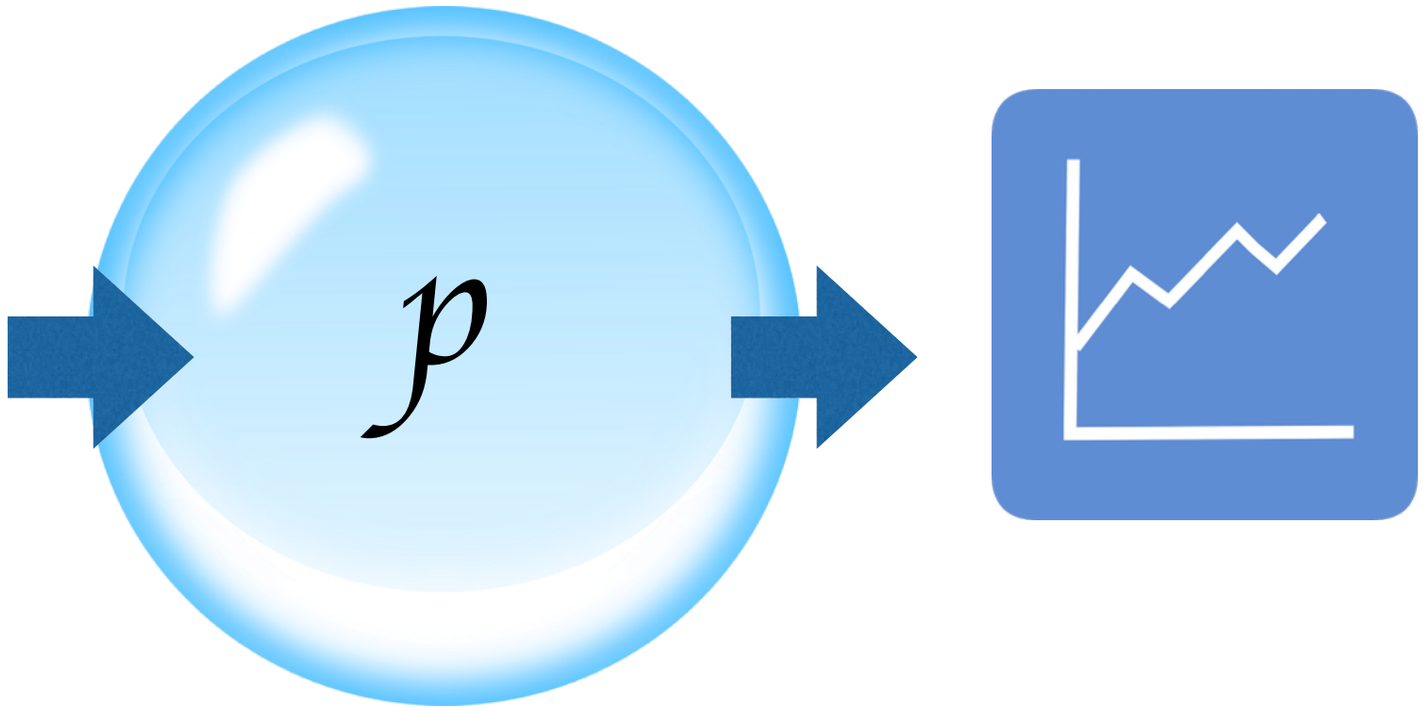
Environment:

input-data



Environment:

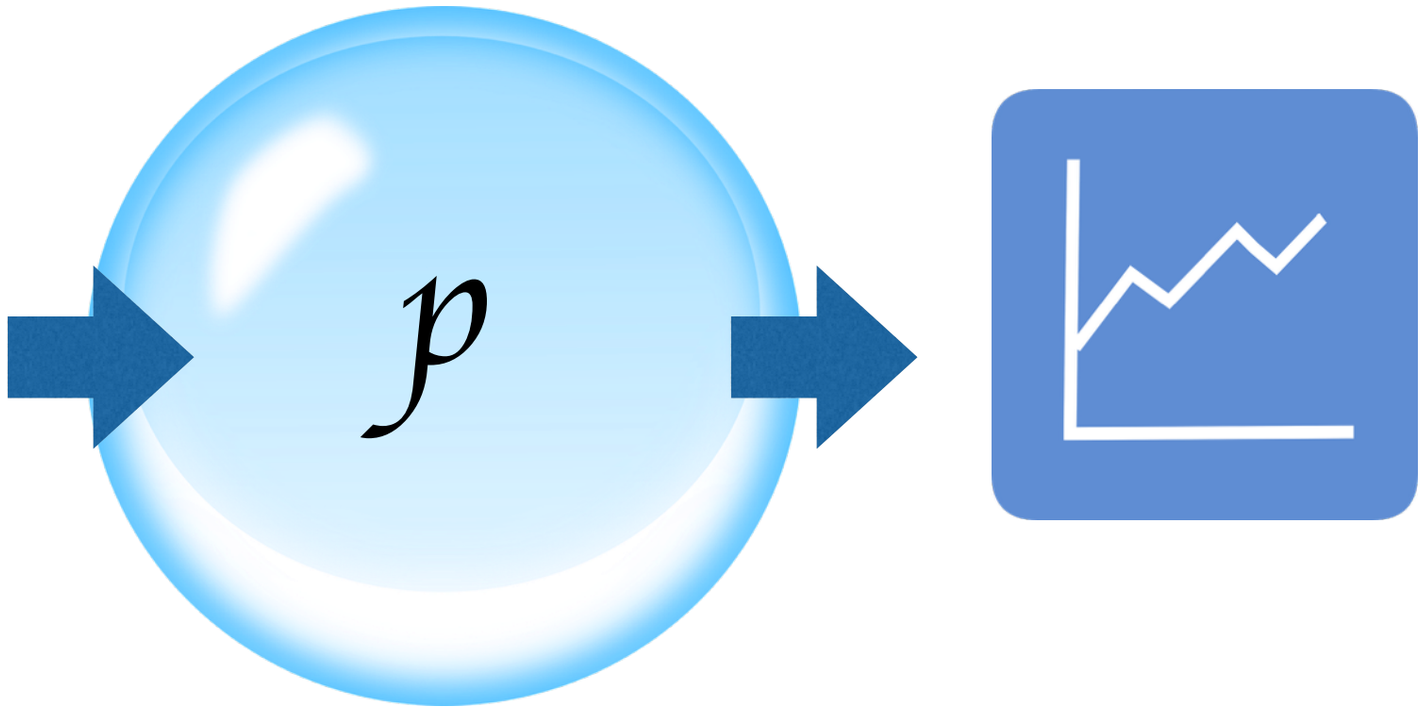
input-data



Deterministic base image

Environment:

input-data



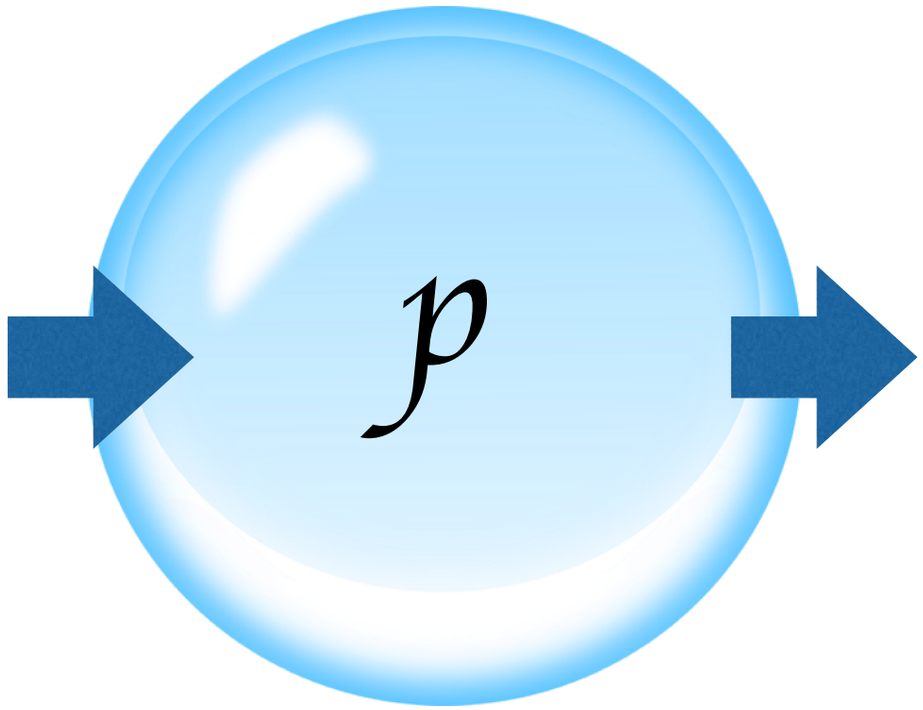
Deterministic base image

docker

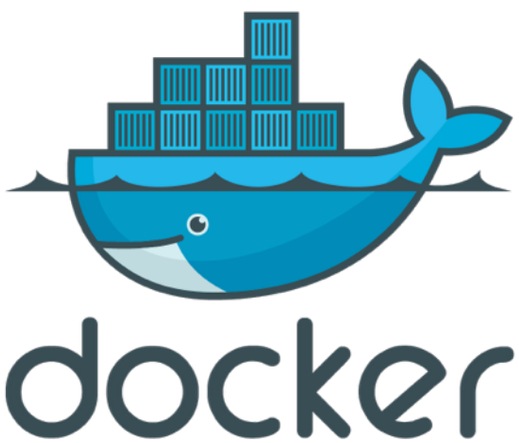
NixOS

Execution:

input-data



Deterministic base image



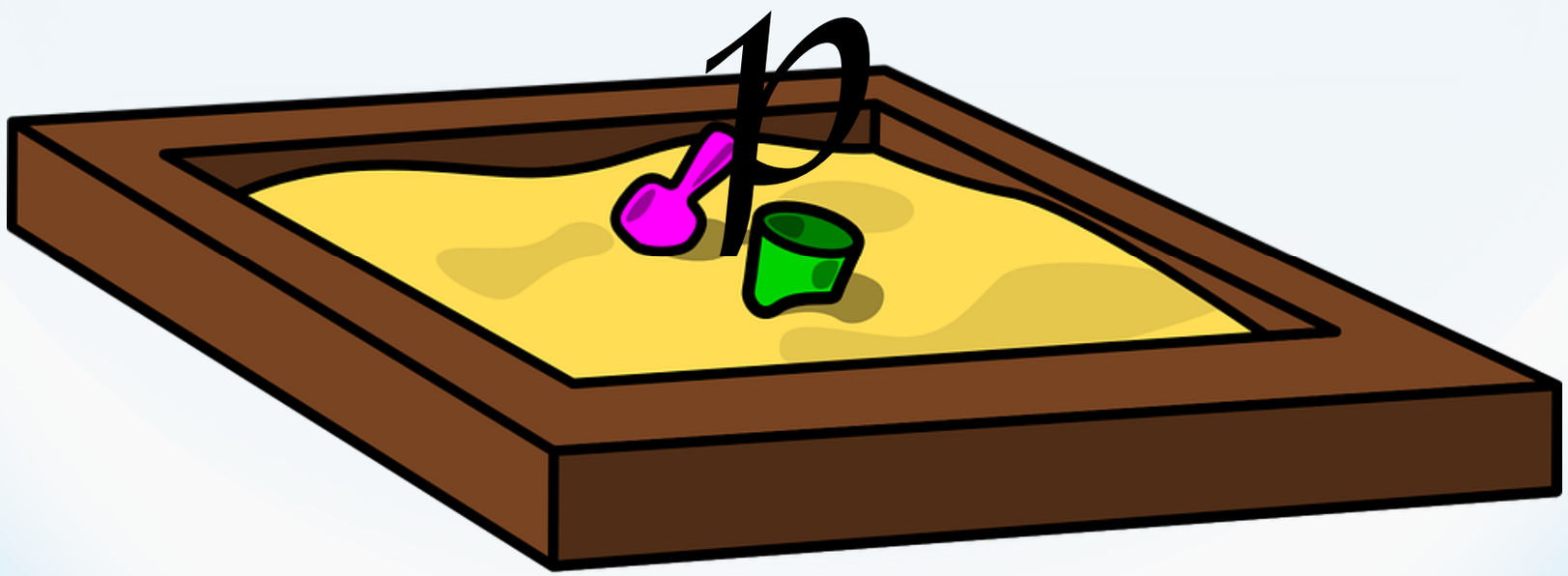
Execution:

p

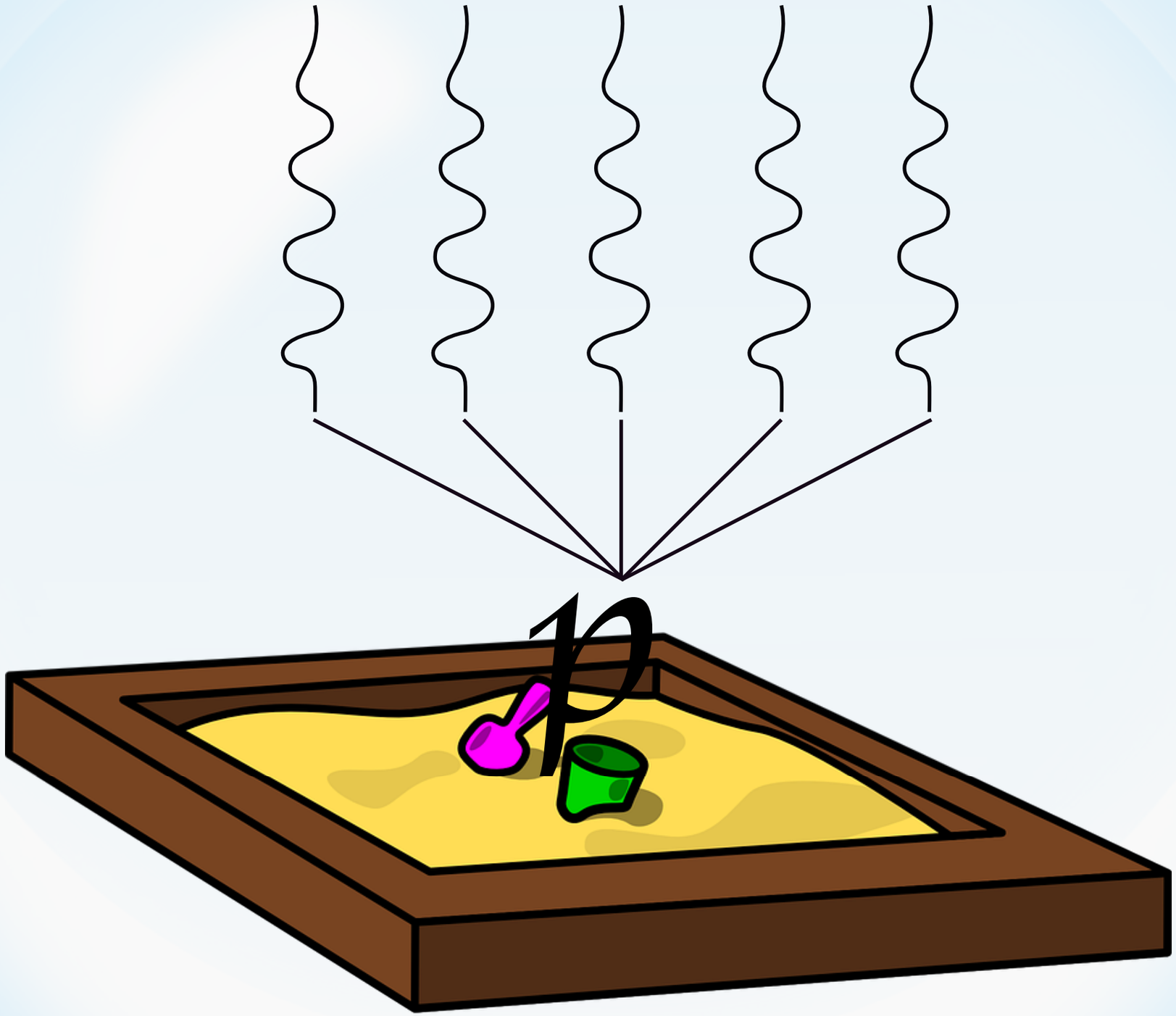
Execution:

p

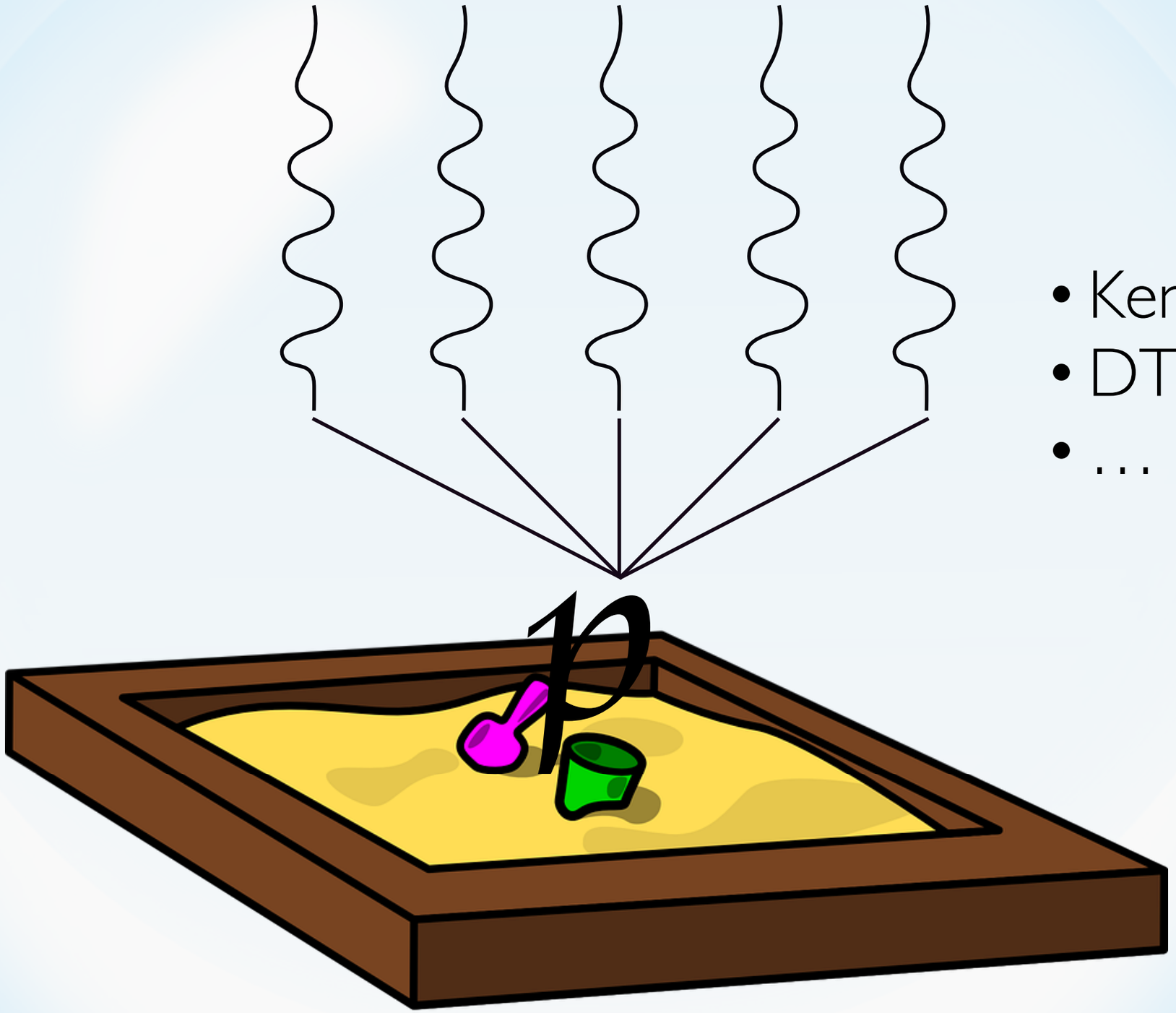
Dynamic determinism enforcement



Dynamic determinism enforcement

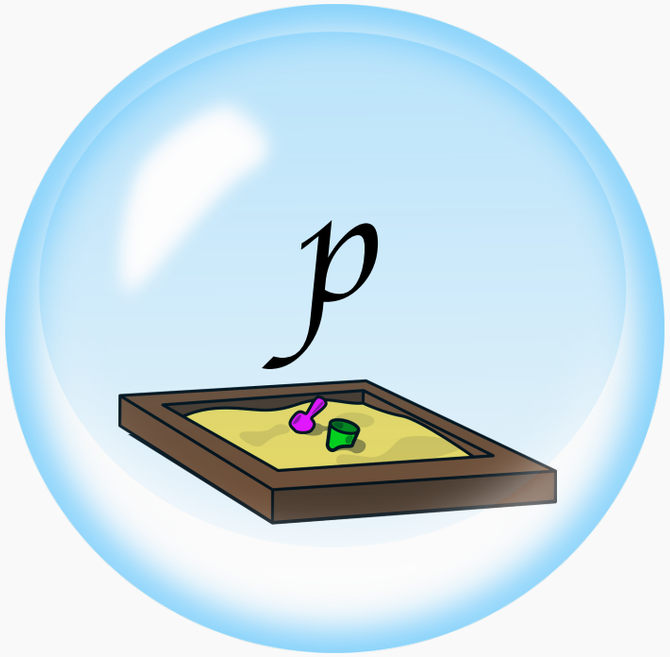


Dynamic determinism enforcement

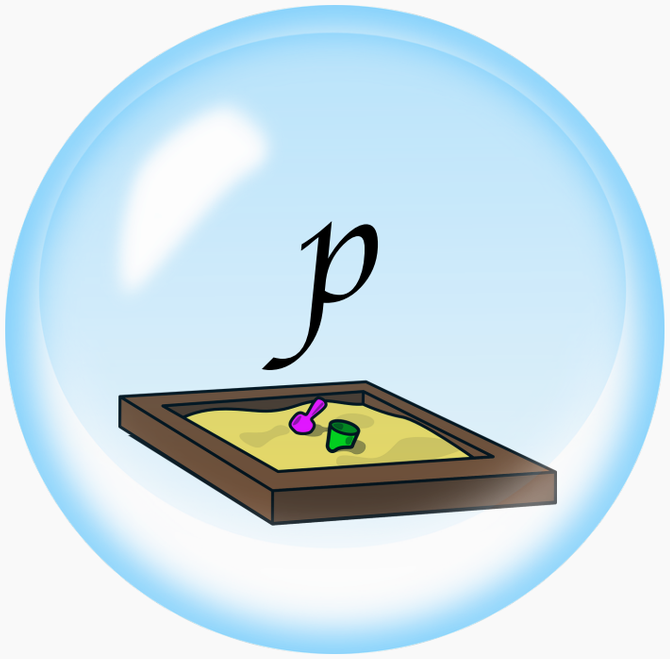


- Kendo
- DThreads
- …

Dynamic determinism enforcement

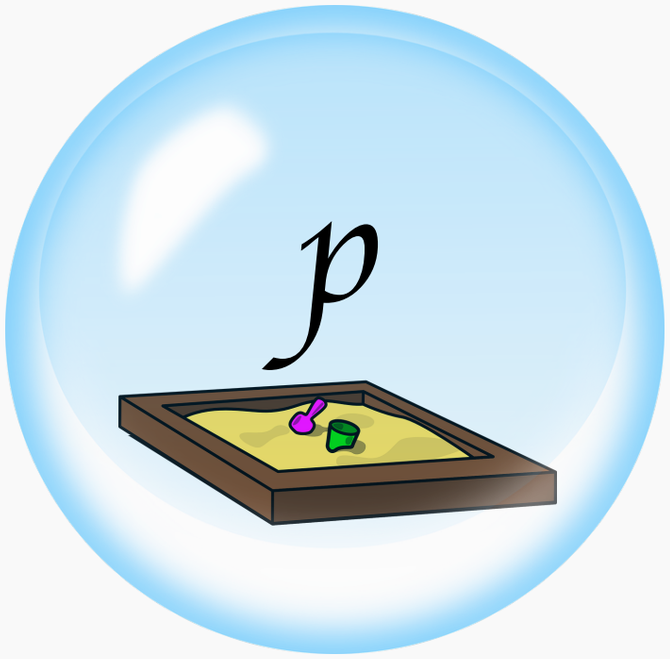


: (Inputs , NumThreads) → Outputs



: (Inputs , NumThreads) → Outputs

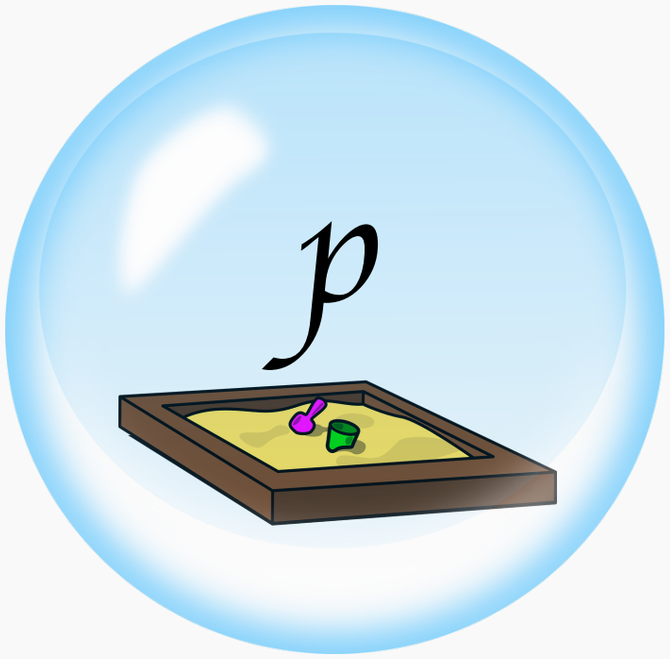
Reproducibility



: (Inputs , NumThreads) → Outputs

Reproducibility

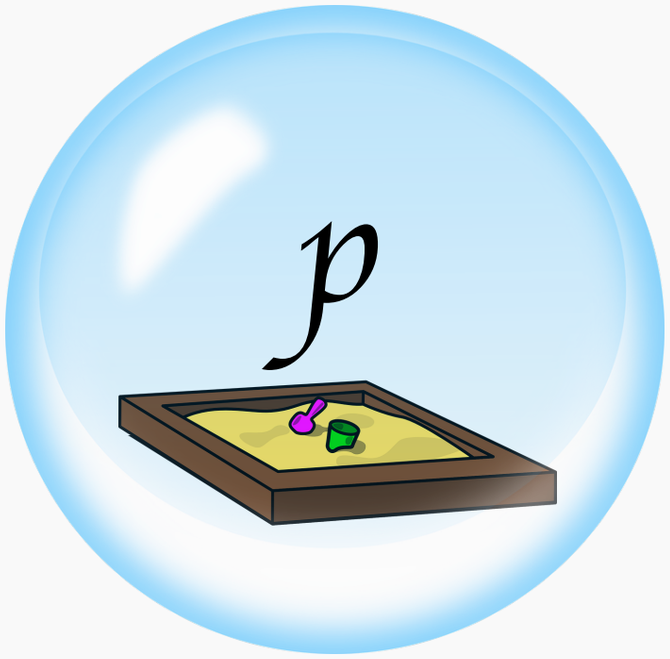
Portability



: (Inputs , GPU model) → Outputs

Reproducibility

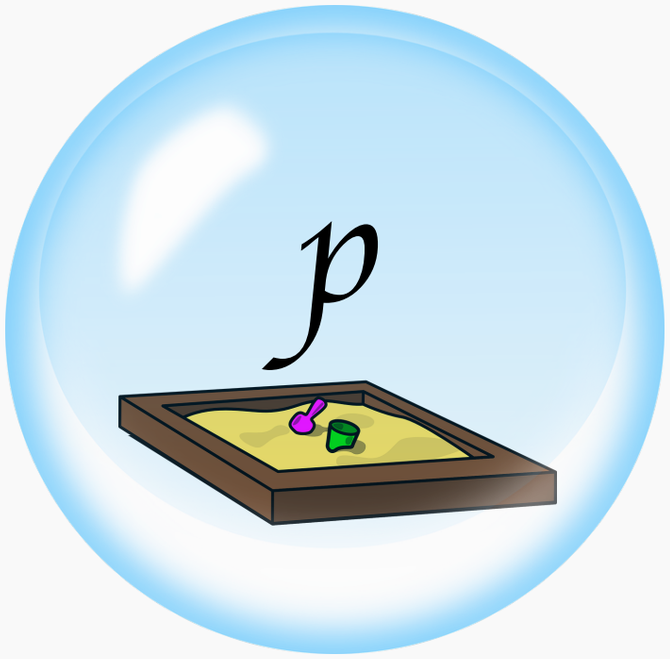
Portability



: (Inputs , OS Version) → Outputs

Reproducibility

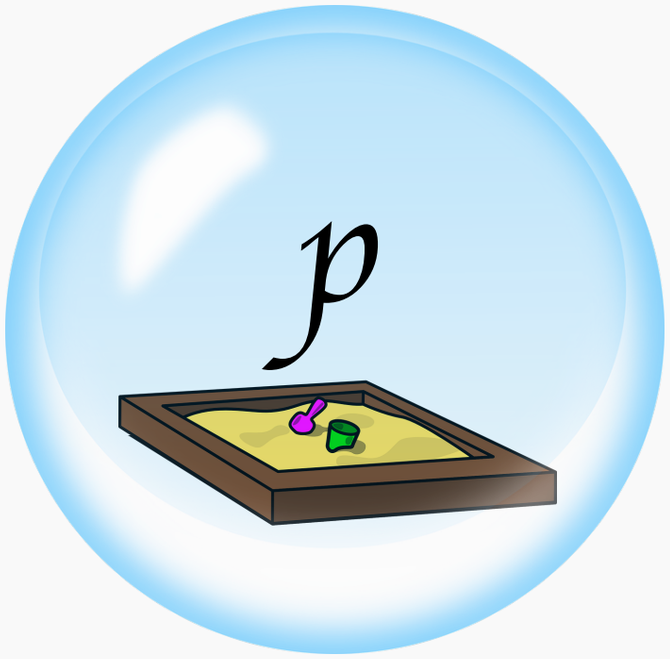
Portability



: (Inputs , Sched.Trace) → Outputs

Reproducibility

Portability



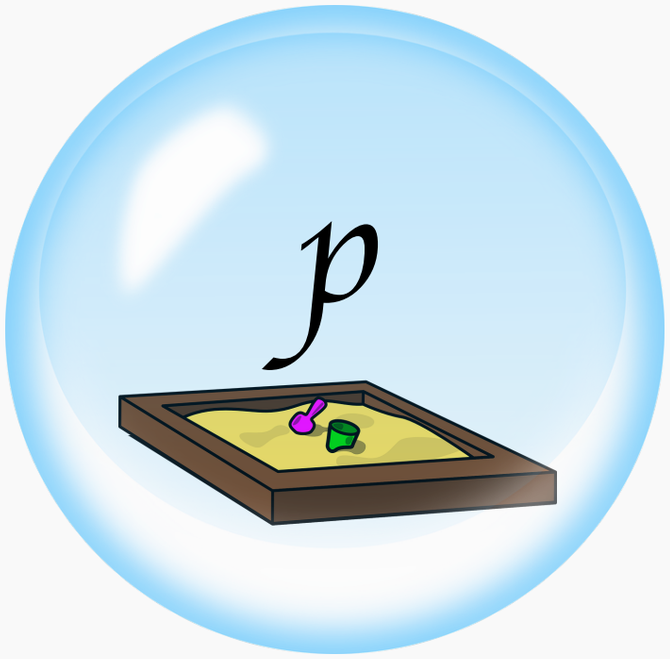
: (Inputs , Sched.Trace) → Outputs

rr

mozilla

Reproducibility

Portability



: (Inputs , Sched.Trace) → Outputs

rr

mozilla

Reproducibility

Portability

- Still open problems for dynamic determinism enforcment: **user-space process tree determinism, file systems**

?

Reproducibility

Portability

Static ? checking

Reproducibility

Portability

Tony Hoare, 1971, on the prospect of **Static checking**

5

TOWARDS A THEORY OF PARALLEL PROGRAMMING

C. A. R. HOARE

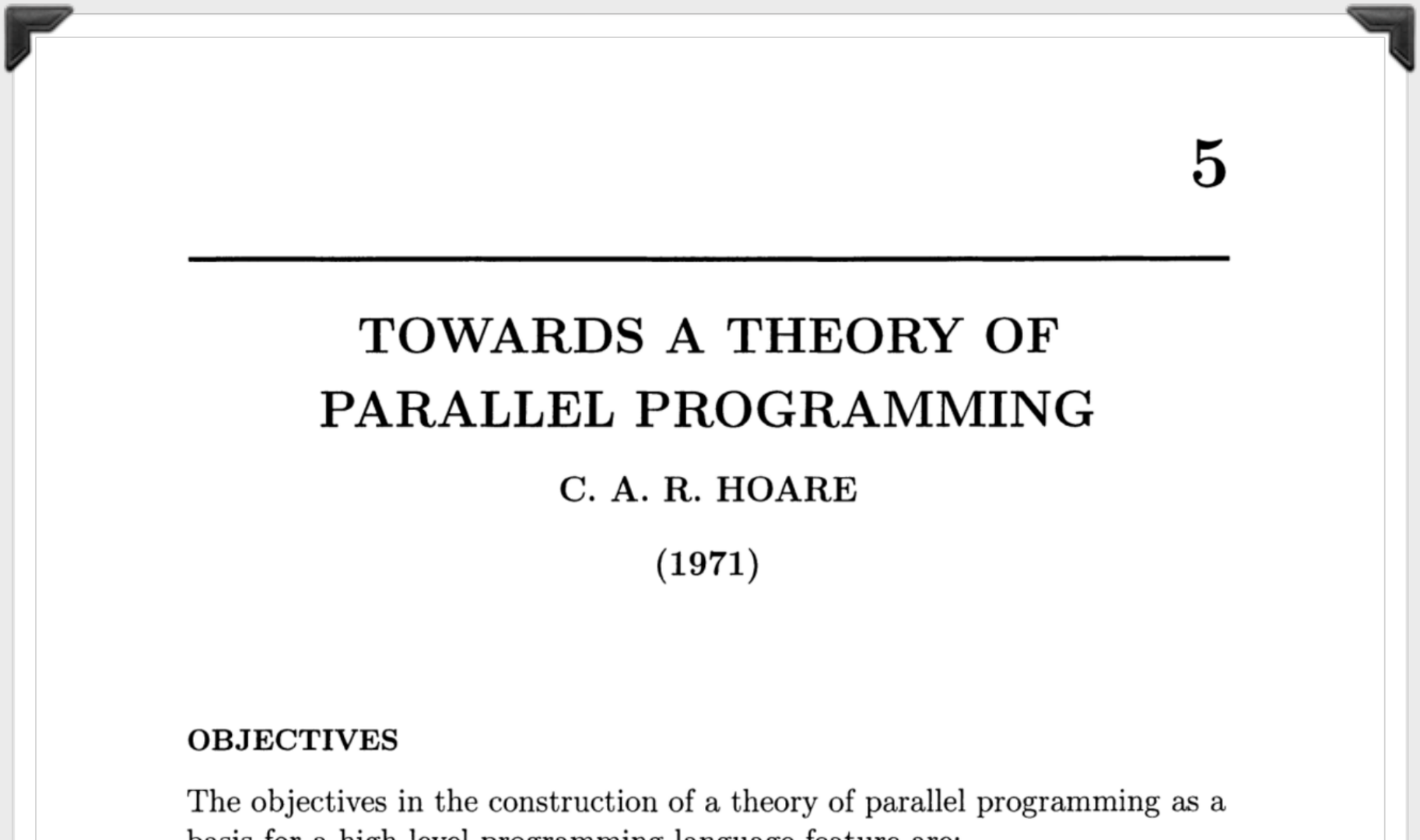
(1971)

OBJECTIVES

The objectives in the construction of a theory of parallel programming as a

Tony Hoare, 1971, on the prospect of **Static checking**

"It is therefore very important that a high-level language designed for [parallel programming] should provide complete security against time-dependent errors by means of a ***compile-time check****."*



5

TOWARDS A THEORY OF PARALLEL PROGRAMMING

C. A. R. HOARE

(1971)

OBJECTIVES

The objectives in the construction of a theory of parallel programming as a

Determinism as a Safety property

```
{-# LANGUAGE Safe #-}
import Control.LVish
x = <your untrusted code>
main :: IO ()
main = print x
```

Determinism as a Safety property

```
{-# LANGUAGE Safe #-}
import Control.LVish
x = <… runPar _ …>
main :: IO ()
main = print x
```

Determinism as a Safety property

```
{-# LANGUAGE Safe #-}
import Control.LVish
x = <… runPar _ …>
main :: IO ()
main = print x
```

```
{-# LANGUAGE Safe #-}
…
main :: Det ()
main = print x
```

Relationship to program generation

- Archiving programs in source form requires precise representations of the:
  - code
  - compiler
  - environment
- (A “perfect name” or hash for a program.)

2. Composable Autotuning

- Typically:
  - Global parameters
  - Hacky Scripts
  - Fixed K-dimensional search spaces

```
$ cat “1,2,3” > params.txt
$ ./run_stuff.sh
```

2. Composable Autotuning

- Typically:
  - Global parameters
  - Hacky Scripts
  - Fixed K-dimensional search spaces

```
$ cat “1,2,3” > params.txt
$ ./run_stuff.sh
```

Ideal

```
import A (a)
import B (b)
…
   do  a  — may use auto-tuning
       b  — may use auto-tuning
       …
```

One idea

One idea

- Use an Applicative transformer (Tune m a)

One idea

- Use an Applicative transformer (Tune m a)
- Gather params before running `m`

One idea

- Use an Applicative transformer (Tune m a)
- Gather params before running `m`
- But individual runs are still *anonymous*

One idea

- Use an Applicative transformer (Tune m a)
- Gather params before running `m`
- But individual runs are still *anonymous*

Avoid:

```
x = runSearch $
    setParam (Proxy::Proxy "a") (0,10) $
    setParam (Proxy::Proxy "b") (10,20) $
    go
```

Problems:

- *Persistence* is awkward
  - Store learned results
  - Comes back to naming programs:
    - What is the *same* program?
  - And naming *choices*:
    - choice structure is a (un)labeled tree?

3. Fusion for nested, irregular data

3. Fusion for nested, irregular data

```
fold1(setUnion,
      map(nbrs,states))
```

3. Fusion for nested, irregular data

```
fold1(setUnion,
      map(nbrs,states))
```

```
do acc ← newEmptySet()
   forEach x ∈ states:
     forEach n ∈ nbrs(x):
       insert(n,acc)
```

4. Data structure representation rewrites

4. Data structure representation rewrites

```
Set (Maybe a) ⇒ (Bool, Set a)
```

4. Data structure representation rewrites

```
Set (Maybe a) ⇒ (Bool, Set a)
```

```
from s = (Nothing ∈ s,
          map fromJust
            (filter isJust s))
```

4. Data structure representation rewrites

```
Set (Maybe a) ⇒ (Bool, Set a)
```

```
from s = (Nothing ∈ s,
          map fromJust
           (filter isJust s))
```

- (Closed type families are sufficient for this example.)

5. User-friendly metaprogramming

5. User-friendly metaprogramming

- Should I need to know whether a particular expression happens at a particular stage?

5. User-friendly metaprogramming

- Should I need to know whether a particular expression happens at a particular stage?
- Staging *goals*

5. User-friendly metaprogramming

- Should I need to know whether a particular expression happens at a particular stage?
- Staging *goals*
  - this function isn't called at runtime (INLINE)

5. User-friendly metaprogramming

- Should I need to know whether a particular expression happens at a particular stage?
- Staging *goals*
  - this function isn't called at runtime (INLINE)
  - this datatype doesn't appear at runtime

5. User-friendly metaprogramming

- Should I need to know whether a particular expression happens at a particular stage?
- Staging *goals*
  - this function isn't called at runtime (INLINE)
  - this datatype doesn't appear at runtime
  - this type class creates no dictionaries at runtime

6. Deterministic par. without loose ends

6. Deterministic par. without loose ends

- Don’t assume associativity, commutativity

6. Deterministic par. without loose ends

- Don’t assume associativity, commutativity
  - Accelerate, DPJ, etc…

6. Deterministic par. without loose ends

- Don’t assume associativity, commutativity
  - Accelerate, DPJ, etc…
  - *Prove it!*

6. Deterministic par. without loose ends

- Don’t assume associativity, commutativity
  - Accelerate, DPJ, etc…
- *Prove it!*
  - (… and integrate with static compiler checks like -XSafe)

6. Deterministic par. without loose ends

- Don’t assume associativity, commutativity
  - Accelerate, DPJ, etc…
- *Prove it!*
  - (… and integrate with static compiler checks like -XSafe)
- *(WIP: w/ Ranjit Jhala)*