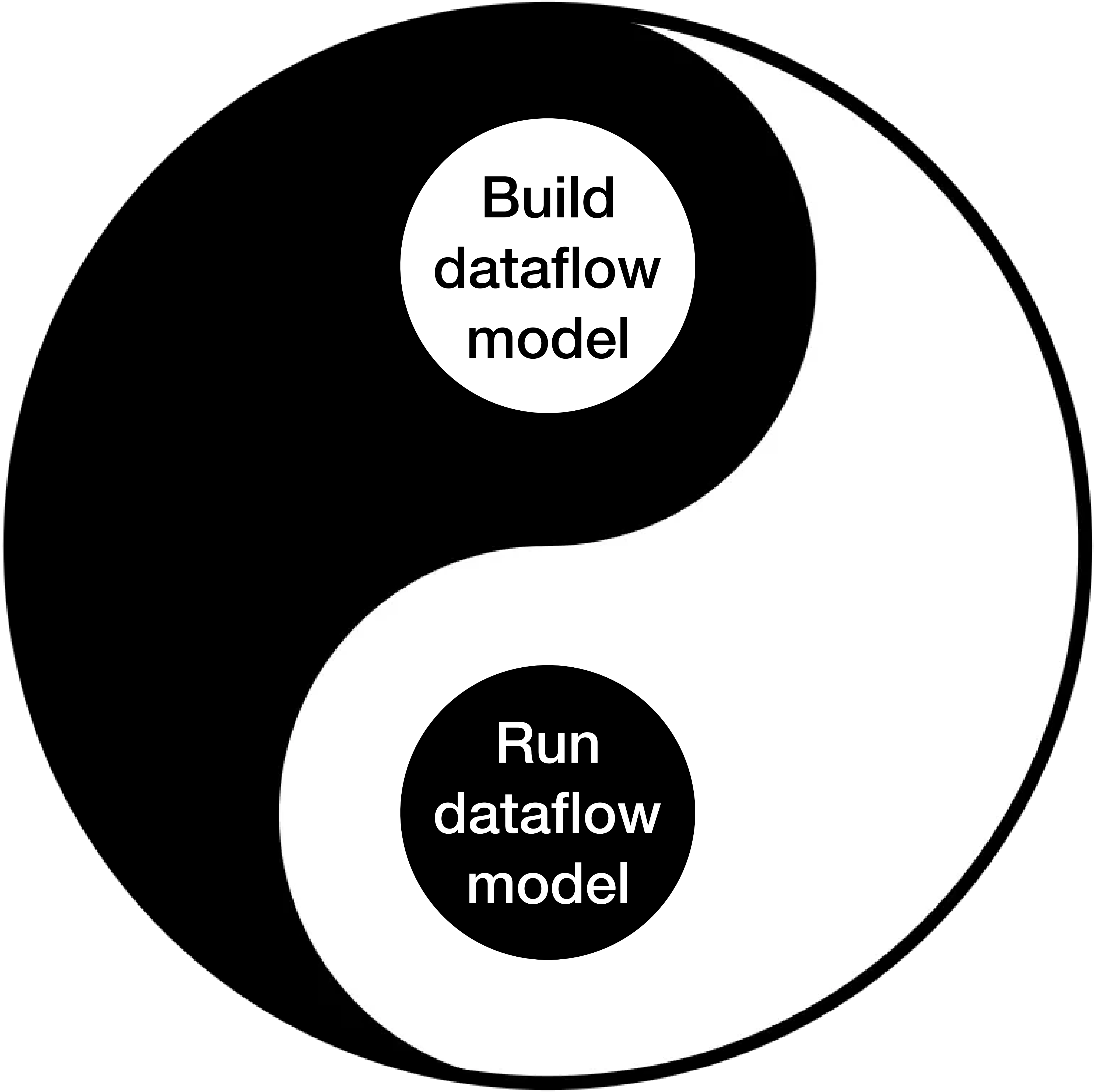
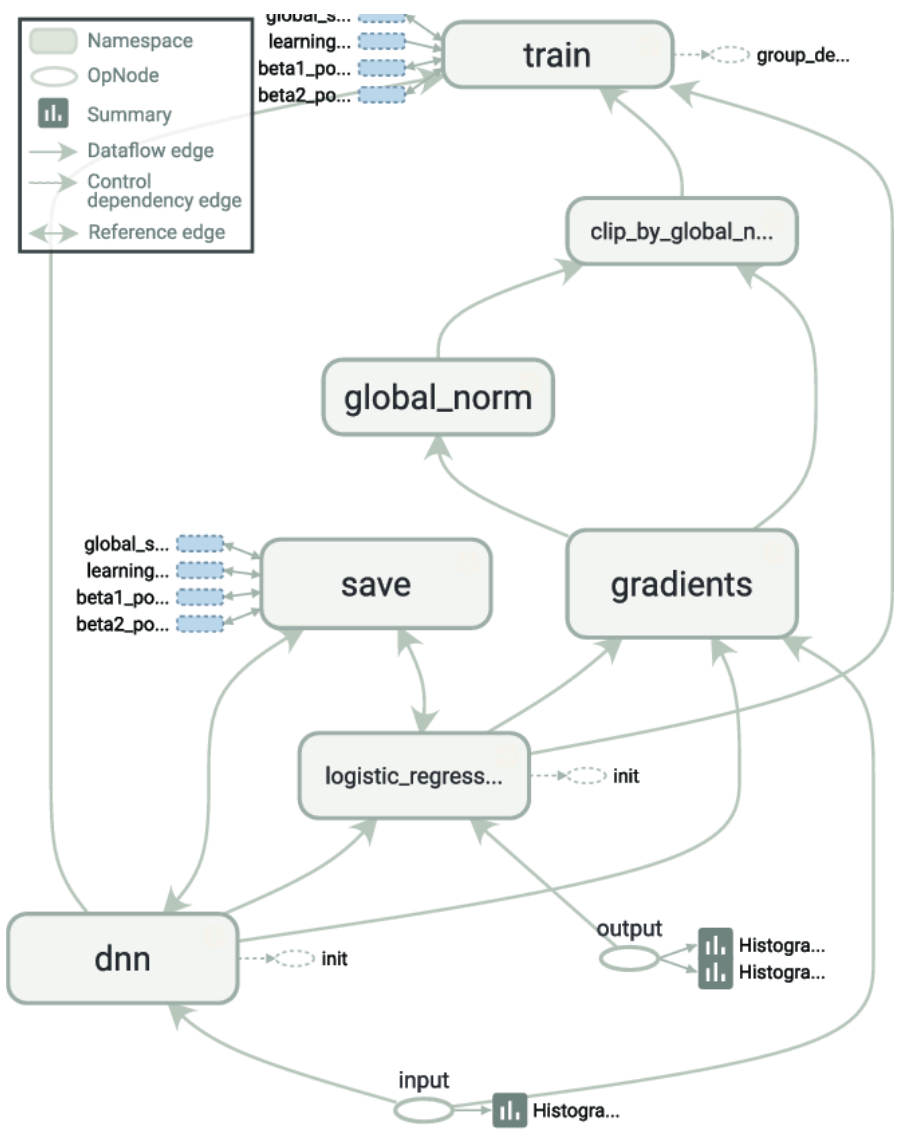
Operational semantics of higher-order transparent synchronous dataflow

Dan R. Ghica // Steven Cheung // Koko Muroya

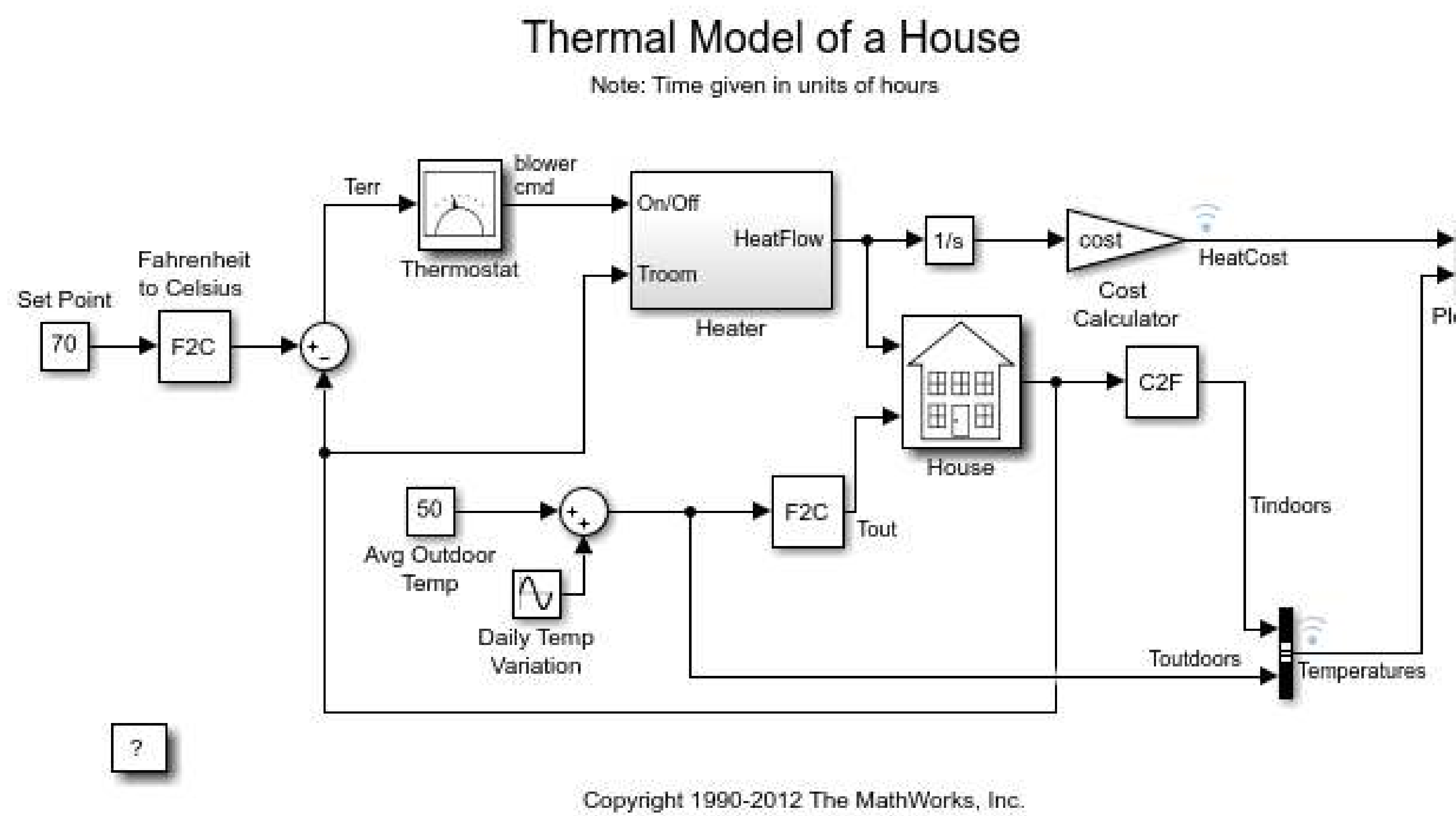
University of Birmingham


Build
dataflow
model
Run
dataflow
model

Bi-modal languages

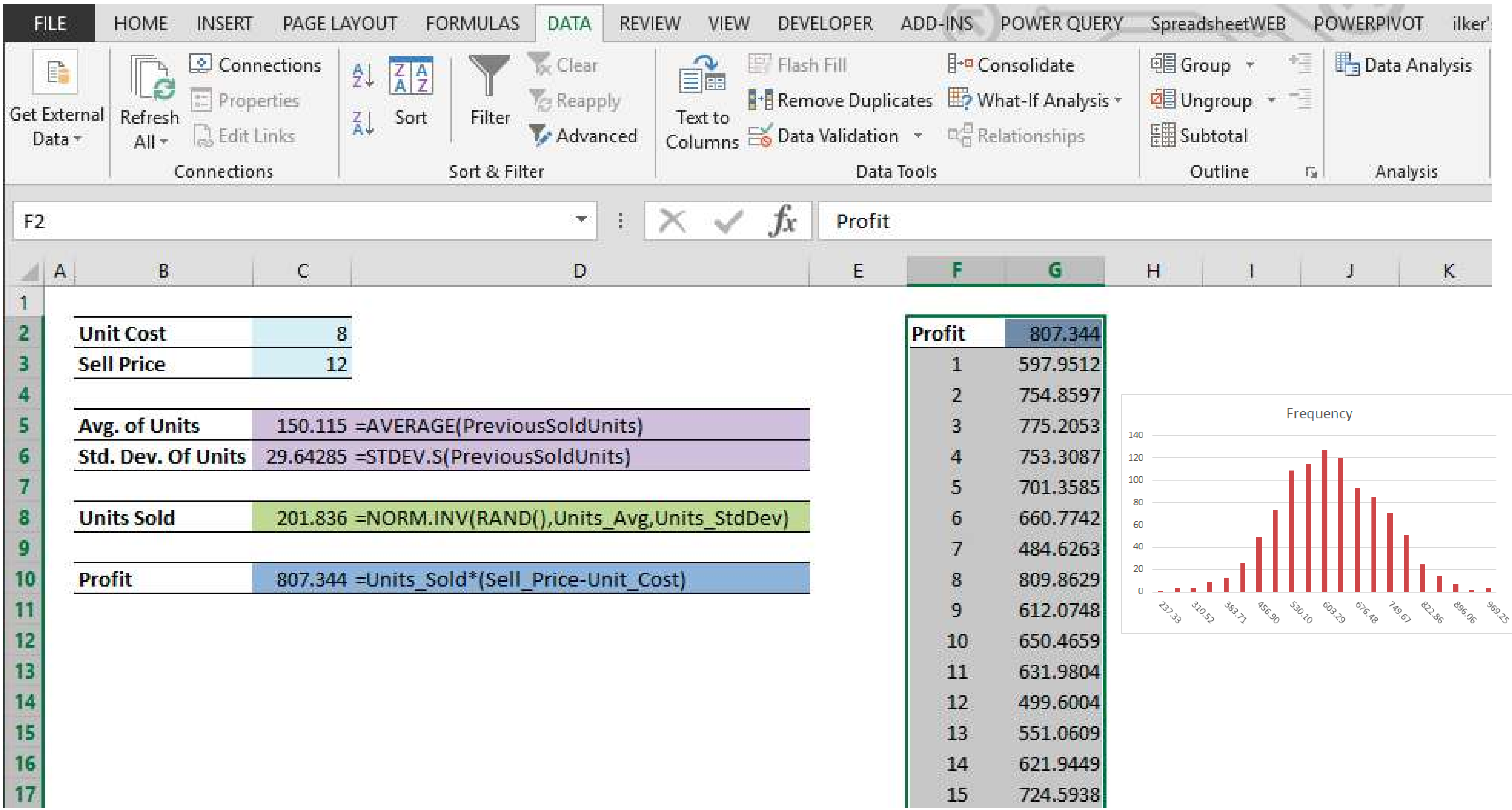


Tensorflow 1.x
(machine learning)

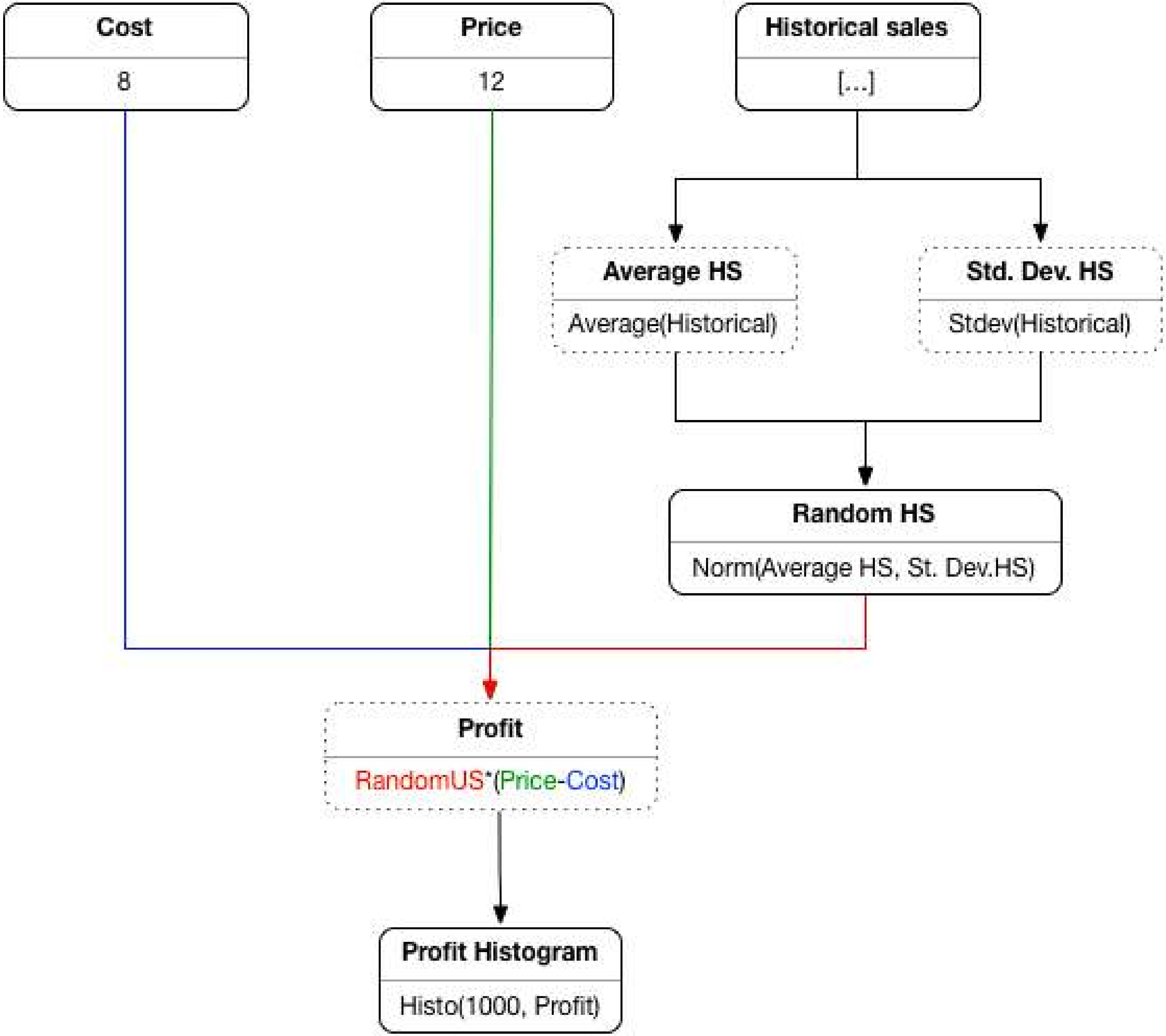


Simulink
(simulation & optimisation)

Excel



FILE HOME INSERT PAGE LAYOUT FORMULAS DATA REVIEW VIEW DEVELOPER ADD-INS POWER QUERY SpreadsheetWEB POWERPIVOT
Get External Data
Refresh All
Connections
Properties
Edit Links
Connections
Sort
Filter
Clear
Reapply
Advanced
Sort & Filter
Text to Columns
Flash Fill
Remove Duplicates
Data Validation
Consolidate
What-If Analysis
Relationships
Data Tools
Group
Ungroup
Subtotal
Outline
Data Analysis
Analysis
F2
Profit
Unit Cost 8
Sell Price 12
Avg. of Units 150.115 =AVERAGE(PreviousSoldUnits)
Std. Dev. Of Units 29.64285 =STDEV.S(PreviousSoldUnits)
Units Sold 201.836 =NORM.INV(RAND(),Units_Avg,Units_StdDev)
Profit 807.344 =Units_Sold*(Sell_Price-Unit_Cost)
Profit 807.344
1 597.9512
2 754.8597
3 775.2053
4 753.3087
5 701.3585
6 660.7742
7 484.6263
8 809.8629
9 612.0748
10 650.4659
11 631.9804
12 499.6004
13 551.0609
14 621.9449
15 724.5938
Frequency



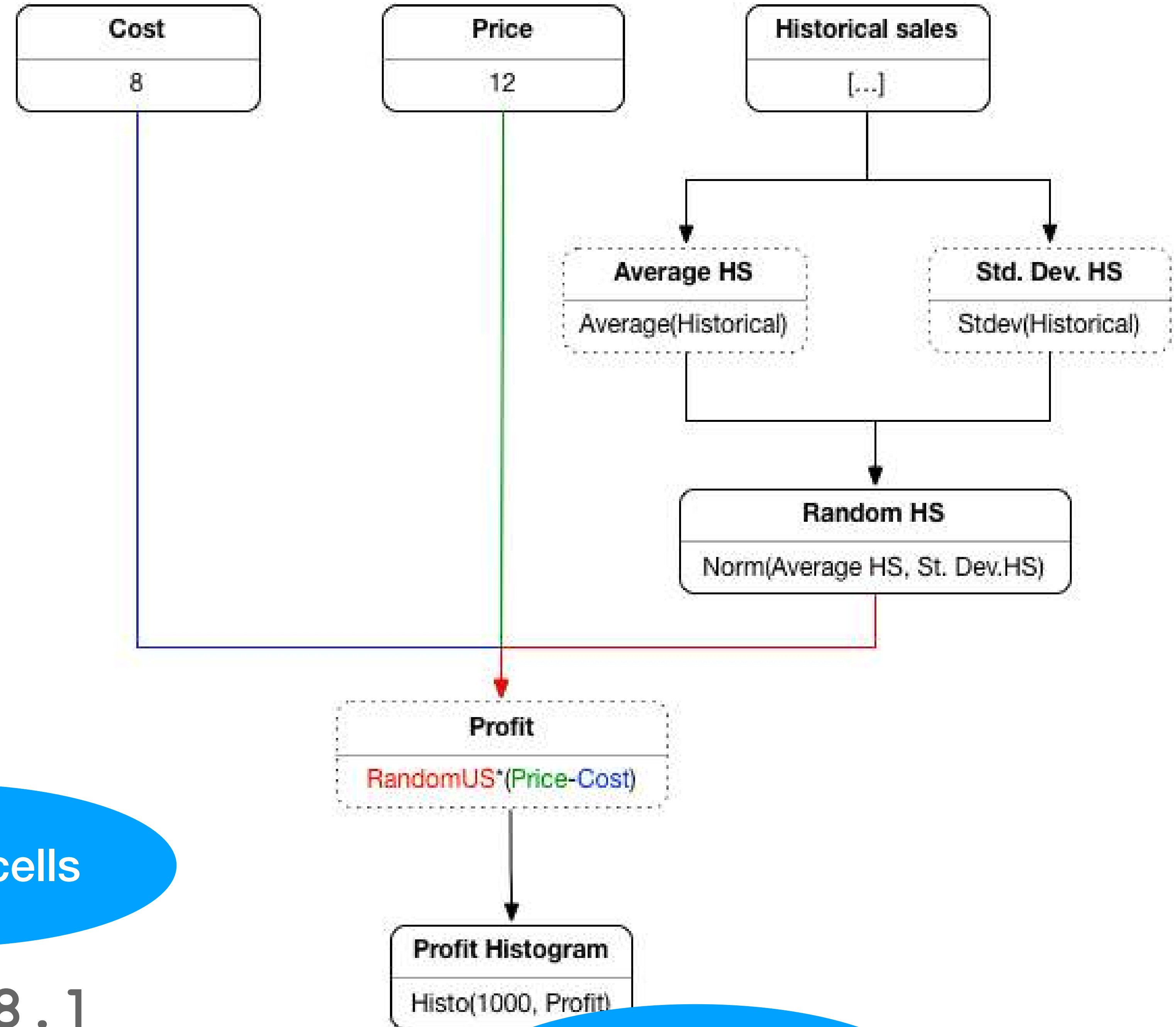
Cost
8
Price
12
Historical sales
[...]
Average HS
Average(Historical)
Std. Dev. HS
Stdev(Historical)
Random HS
Norm(Average HS, St. Dev.HS)
Profit
RandomUS*(Price-Cost)
Profit Histogram
Histo(1000, Profit)

Transparent **S**ynchronous **D**ataflow

an OCaml PPX extension

https://github.com/cwtsteven/TSD

Model construction



(named) cells

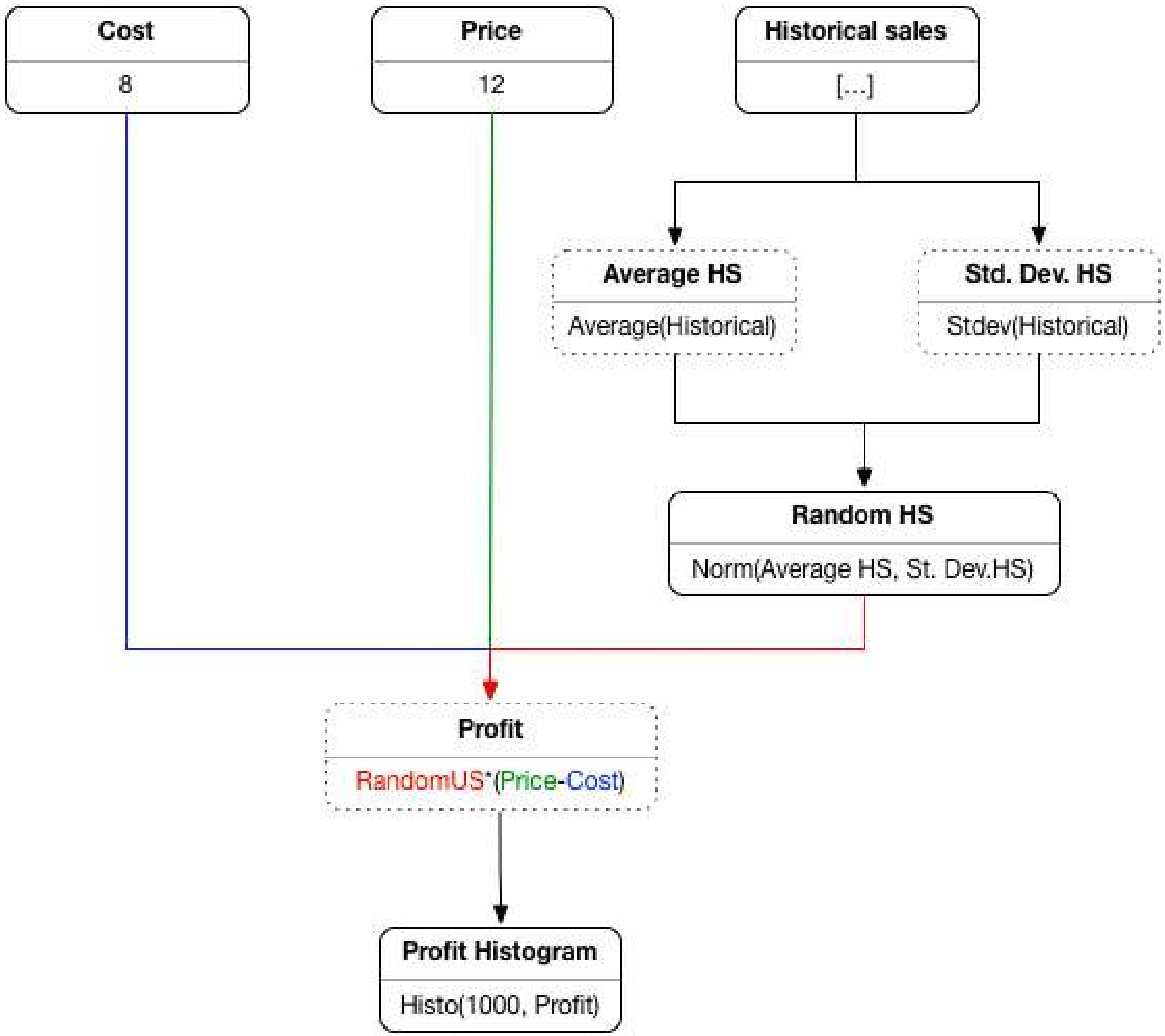
lists in cells

graph dependencies

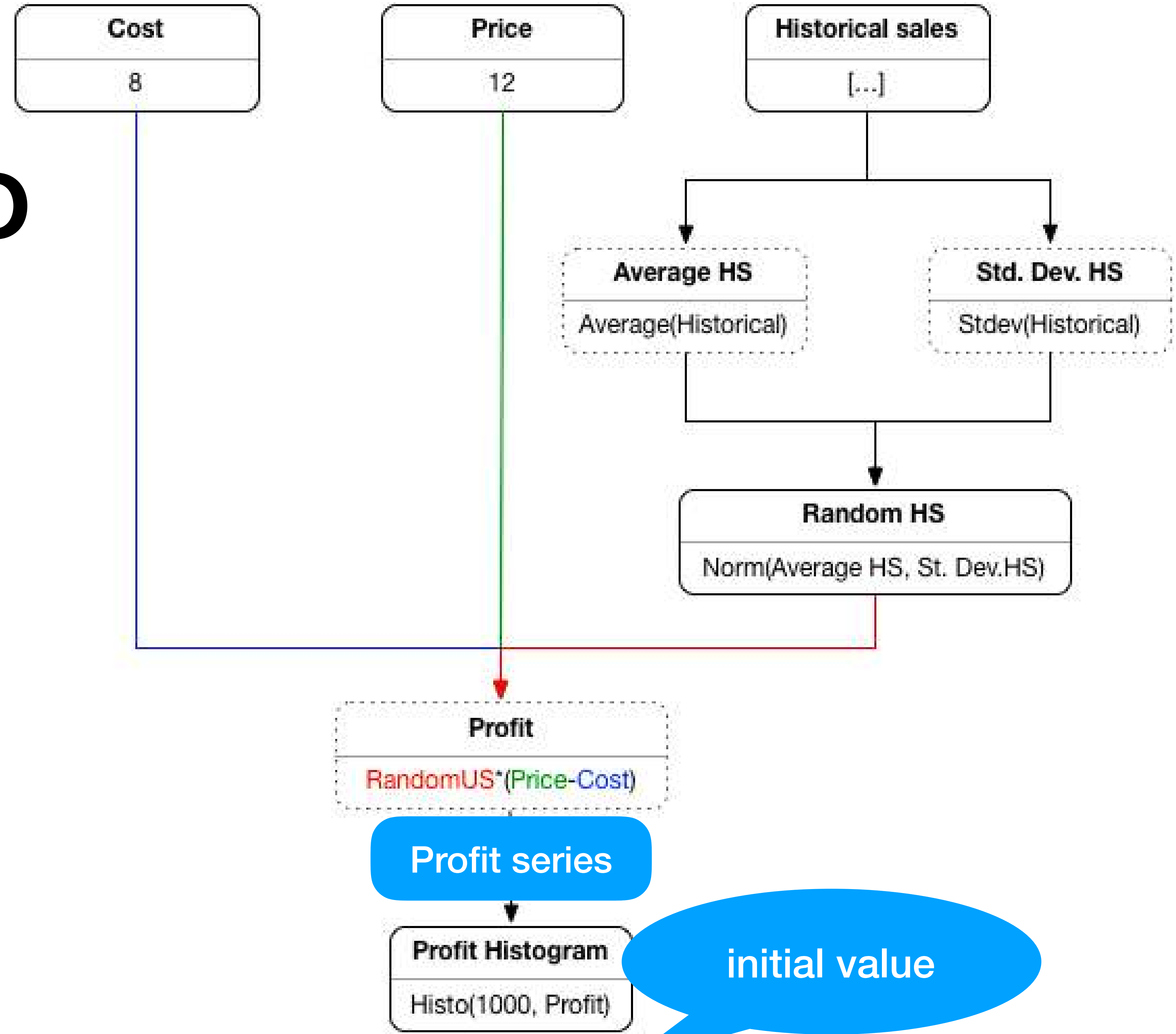
```
let cost  = cell [%dfg  8.]
let price = cell [%dfg 12.]
let hs    = lift [122.; 138.; 163.; 16
let historical_sales = cell [%dfg hs]
let average_hs = [%dfg average historical_s
let std_dev_hs = [%dfg std_dev historical_sales]
let random_hs  = cell [%dfg gauss average_hs std_dev_hs]
let profit     = [%dfg random_hs * (price - cost)]
```

Run model

```
#  peek average_hs;;
- : float = 148.95
# peek std_dev_hs;;
- : float = 25.4430245843
# peek random_hs;;
- : float = 137.839327531
# peek profit;;
- : float = 551.357310127
# step ();;
- : bool = true
# peek average_hs;;
- : float = 148.95
# peek std_dev_hs;;
- : float = 25.44302458435175
# peek random_hs;;
- : float = 154.58015036351415
# peek profit;;
- : float = 618.3206014540566
```



Build
Monte Carlo



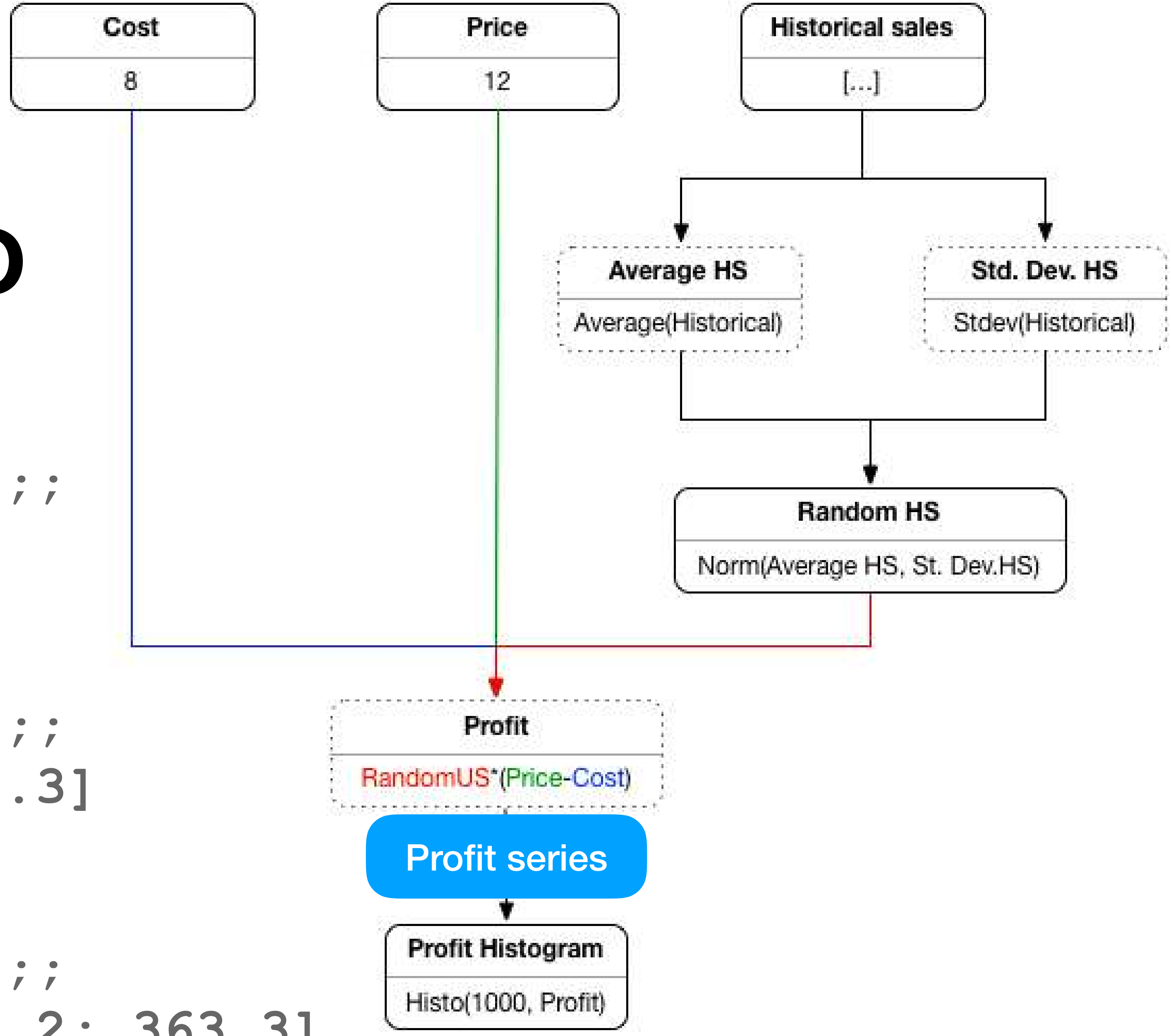

```
let profit_series = let s = cell [%dfg []] in
                    link s [%dfg cons profit s]; s
```

change dependency

feedback loop

Run Monte Carlo

```
# peek profit_series ;;
- : float list = []
# step();;
- : bool = true
# peek profit_series ;;
- : float list = [363.3]
# step();;
- : bool = true
# peek profit_series ;;
- : float list = [206.2; 363.3]
# step();;
- : bool = true
# peek profit_series ;;
- : float list = [335.8; 206.2; 363.3]
```



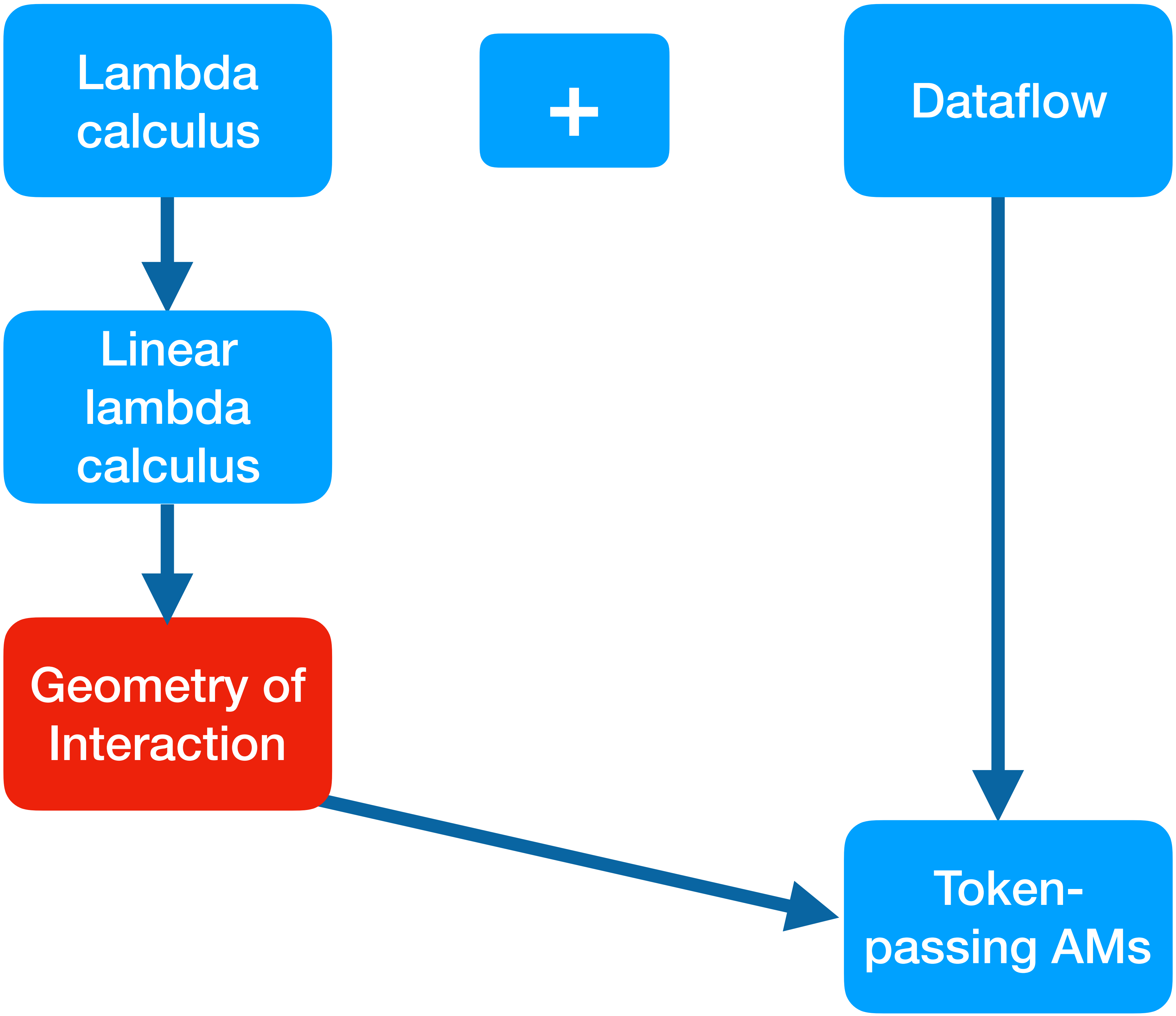
The TSD Calculus

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{\Gamma, x : \tau \vdash t : \tau'}{\Gamma \vdash \lambda x.t : \tau \to \tau'} \qquad \frac{\Gamma \vdash t' : \tau \to \tau' \quad \Gamma \vdash t : \tau}{\Gamma \vdash t'\ t : \tau'} \qquad \frac{}{\Gamma \vdash \underline{n} : Int}$$

$$\frac{}{\Gamma \vdash op : \tau} \qquad \frac{\Gamma \vdash t : Int \quad \Gamma \vdash t_1 : \gamma \quad \Gamma \vdash t_1 : \gamma}{\Gamma \vdash \text{if } t \text{ then } t_1 \text{ else } t_2 : \gamma} \qquad \frac{\Gamma, f : \tau \vdash t : \tau}{\Gamma \vdash \text{rec } f.t : \tau}$$

$+, -, \times, \div : Int \to Int \to Int$	(arithmetic operations)
$\{\_\} : Int \to Cell$	(cell creation)
$\text{link} : Cell \to Int \to Unit$	(linking)
$\text{assign} : Cell \to Int \to Unit$	(assignment of cells)
$\text{peek} : \gamma \to \gamma$	(peeking)
$\text{deref} : Cell \to Int$	(dereferencing of cells)
$\text{root} : Cell \to Int$	(dataflow graph of cells)
$\text{step} : Int$	(step propagtion)

operational semantics



operational semantics

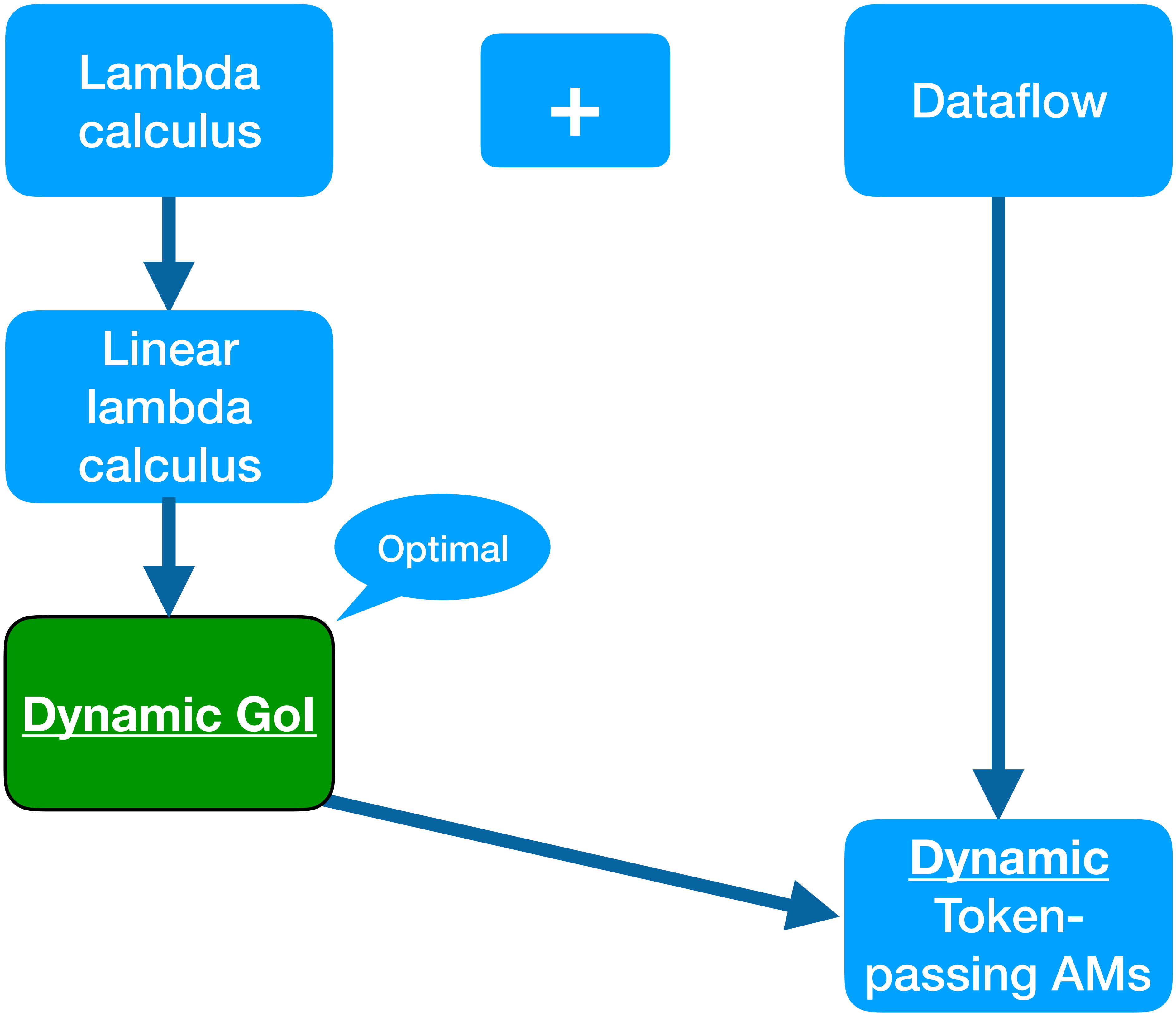
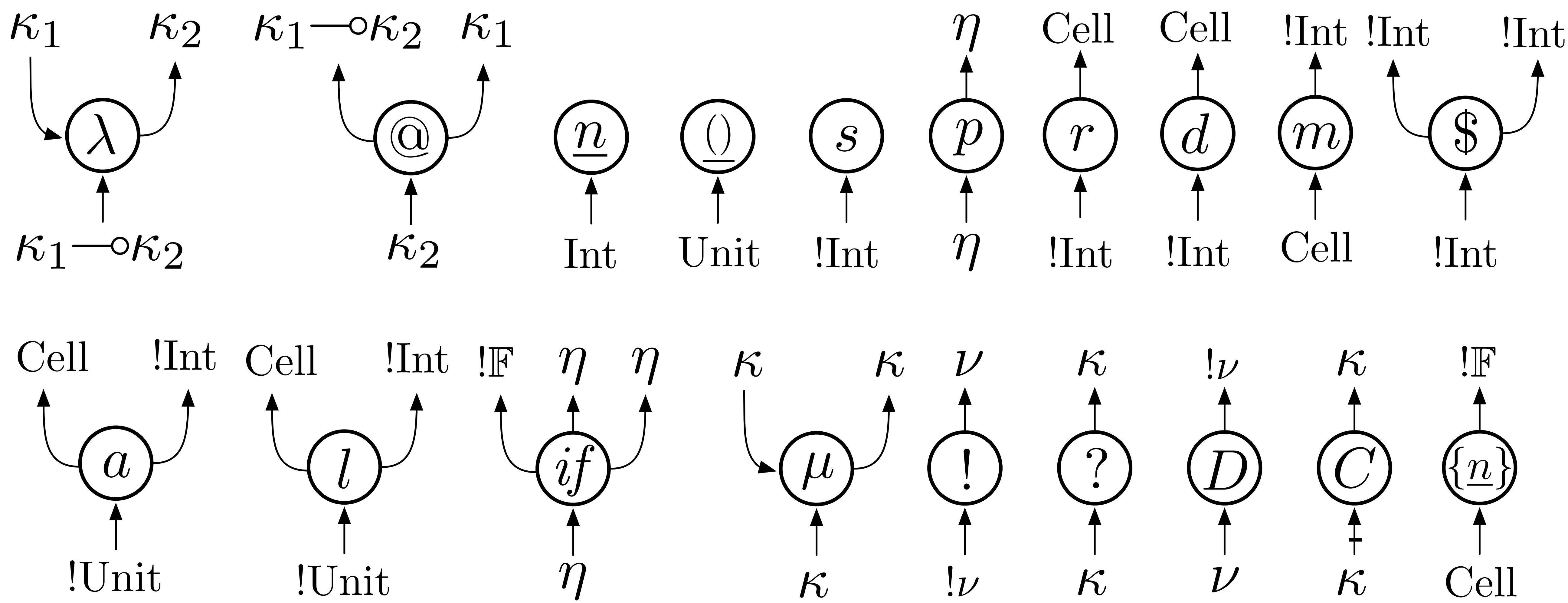


illustration by examples

```
(1+2)+(3+4)
```

```
(λf.f f) (λx.x)
```

```
let x = { 0 } in
let y = { (deref x) + 1 } in
step;
peek y
```



κ_1 κ_2 λ $\kappa_1 \multimap \kappa_2$
$\kappa_1 \multimap \kappa_2$ κ_1 $@$ κ_2
$\underline{n}$ Int
$\underline{()}$ Unit
s !Int
η p η
Cell r !Int
Cell d !Int
!Int m Cell
!Int !Int \$!Int
Cell !Int a !Unit
Cell !Int l !Unit
$!\mathbb{F}$ η η if η
κ κ μ κ
ν ! $!\nu$
κ ? κ
$!\nu$ D ν
κ C κ
$!\mathbb{F}$ $\{\underline{n}\}$ Cell

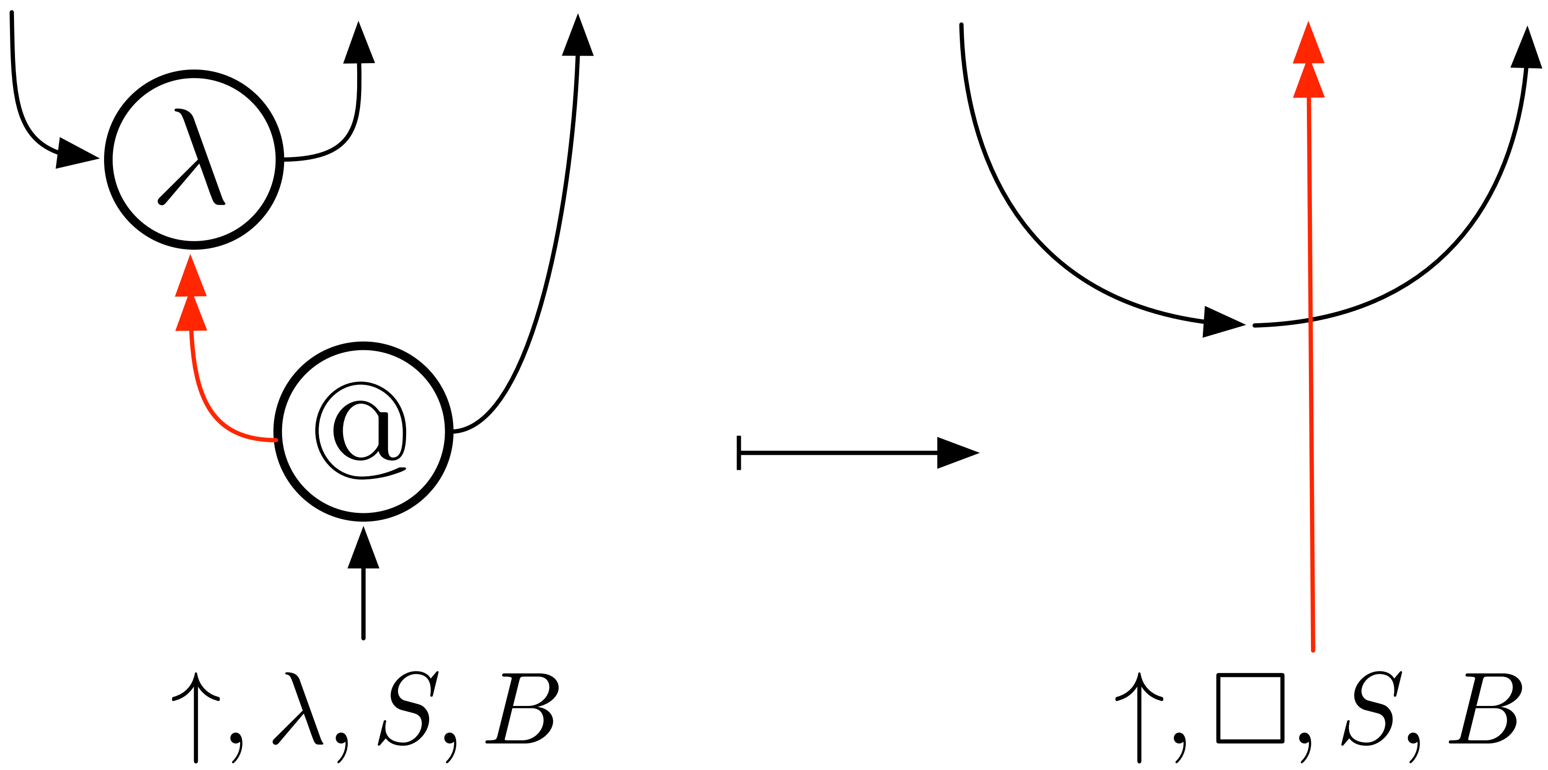
Definition 3.1 (Evaluation tokens)**.** For any graph G, the state of an evaluation token (e, d, f, S, B) consists of a *link* $e \in Link_G$ indicating the token position, a *direction* d, a *rewrite flag* f, a *computation stack* S and a *box stack* B with

$$
\begin{aligned}
d &::= \uparrow \mid \downarrow \\
f &::= \Box \mid \lambda \mid \mathrm{if} \mid C \mid ! \mid \mu \mid m \mid p \mid l(i) \mid a(b,i) \mid r(i) \mid sp \mid s \\
S &::= \Box \mid \star : S \mid (\lambda, -) : S \mid (\underline{n}, -) : S \mid (\underline{n}, g) : S \mid (\underline{n}, i) : S \mid (\underline{()}, -) : S \mid \mathrm{if}_0 : S \mid \mathrm{if}_1 : S \\
B &::= \Box \mid r : B
\end{aligned}
$$

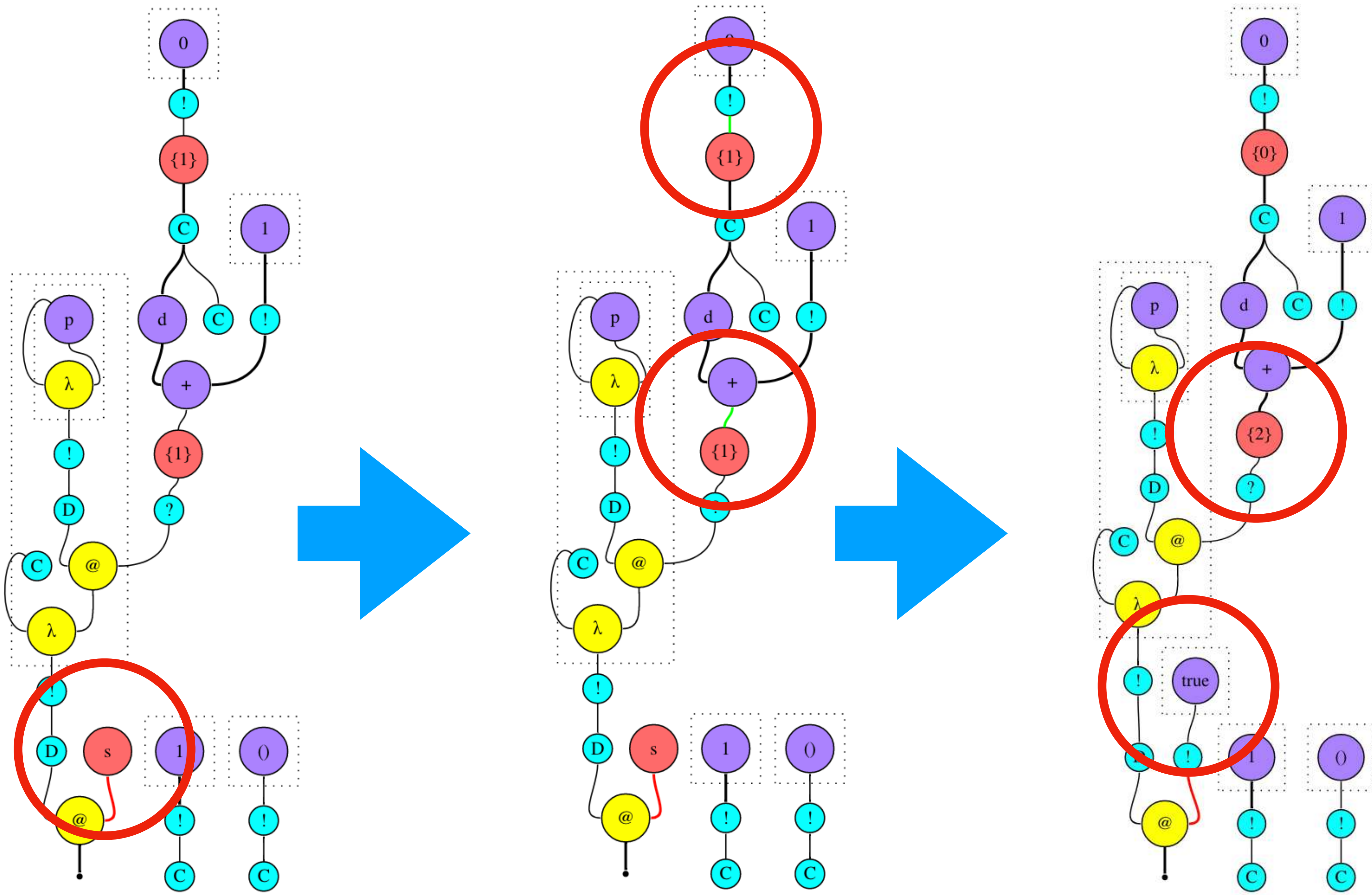
Pass transitions

$$(\mathscr{G}[\mu], (i_1, \uparrow, \square, S, B), \emptyset) \longmapsto (\mathscr{G}[\mu], (i_1, \uparrow, \mu, S, B), \emptyset)$$

Rewrite transitions



Bi-modal execution



Results

Proposition 3.1 (Determinism). *The transitions $\mapsto$ is deterministic up to equivalence of propagation sequences.*

Theorem 3.2 (Type soundness). *Let $G : \{r : [\![\tau]\!]\} \to \emptyset = [\![\vdash t : \tau]\!]$ for some closed well-typed term $\vdash t : \tau$. If t is recursion-free then any execution from Init(G, r) terminates. Otherwise, the execution of t is at least safe.*

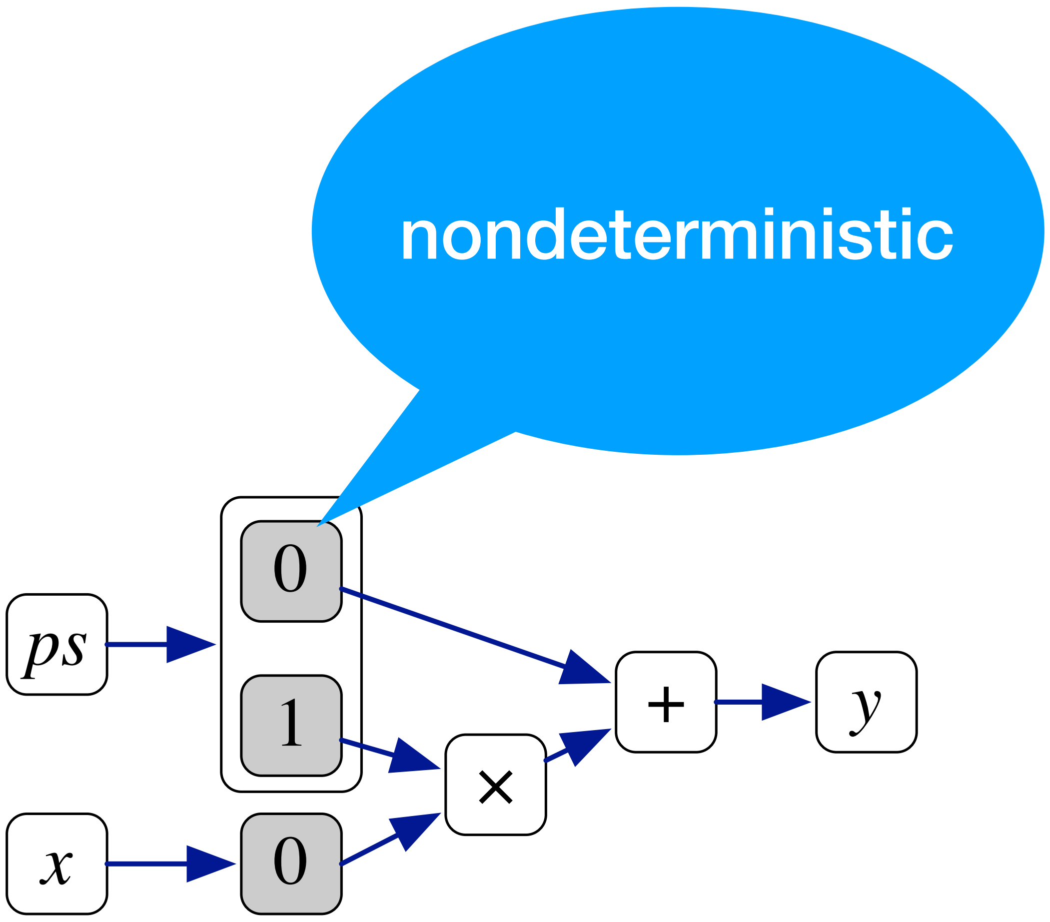
Theorem 3.3 (Efficiency). *The following operations can be executed in linear time on the depth of the dataflow graph: cell creation ($\{t\}$), dereferencing, peeking, linking, assignment, root. The* step *operation can be executed in linear time on the depth of the dataflow graph and on the number of cells.*

Optimisation a la TF

```
W = tf.Variable(np.random.randn(), name = "W")
b = tf.Variable(np.random.randn(), name = "b")
y_pred = tf.add(tf.multiply(X, W), b)
cost = tf.reduce_sum(tf.pow(y_pred-Y, 2)) / (2 * n)
optimizer = tf.train.GradientDescentOptimizer(cost)
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    sess.run(optimizer, feed_dict = {X : _x, Y : _y})
    c = sess.run(cost, feed_dict = {X : x, Y : y})
    weight = sess.run(W)
    bias = sess.run(b)
```

Optimisation a la TSD

```
let x = cell (lift 0)
let y = [%syf x * pc 1 + pc 0 ]
let ps = fusion y
optimise (x, y) ps loss data
```



Taming TF

- fuse variables opaquely
- only use symmetric operations
- prevent mixing using generative typing

$$\frac{A \vdash \Gamma, \tau}{A \mid \Gamma, x : \tau \vdash x : \tau} \qquad \frac{}{A \mid \Gamma \vdash \underline{n} : \mathbb{F}} \qquad \frac{}{A \mid \Gamma \vdash \text{step} : \mathbb{F}} \qquad \frac{A \mid \Gamma, x : \tau \vdash t : \tau'}{A \mid \Gamma \vdash \lambda x.t : \tau \longrightarrow \tau'}$$

$$\frac{A \mid \Gamma \vdash t' : \tau \longrightarrow \tau' \quad A \mid \Gamma \vdash t : \tau}{A \mid \Gamma \vdash t' \; t : \tau'} \qquad \frac{A \mid \Gamma, f : \tau \vdash t : \tau}{A \mid \Gamma \vdash \text{rec } f.t : \tau} \qquad \frac{A \mid \Gamma \vdash t_i : \tau_i \quad i = 1,2 \quad \$: \tau_1 \longrightarrow \tau_2 \longrightarrow \tau}{A \mid \Gamma \vdash t_1 \;\; \$ \;\; t_2 : \tau}$$

$$\frac{A \mid \Gamma \vdash t : \mathbb{F} \quad A \mid \Gamma \vdash t_i : \gamma \quad i = 1,2}{A \mid \Gamma \vdash \text{if } t \text{ then } t_1 \text{ else } t_2 : \gamma} \qquad \frac{A \mid \Gamma \vdash t : \mathit{Cell} \quad A \mid \Gamma \vdash t' : \mathbb{F}}{A \mid \Gamma \vdash \text{link } t \text{ to } t' : \mathit{Unit}}$$

$$\frac{A \mid \Gamma \vdash t : \mathit{Prm}_\alpha \quad A \mid \Gamma \vdash t' : \mathit{Vec}_\alpha}{A \mid \Gamma \vdash \text{assign } t \text{ to } t' : \mathit{Unit}} \qquad \frac{A \mid \Gamma \vdash t : \mathit{Cell} \quad A \mid \Gamma \vdash t' : \mathbb{F}}{A \mid \Gamma \vdash \text{assign } t \text{ to } t' : \mathit{Unit}} \qquad \frac{A \mid \Gamma \vdash t : \mathbb{F}}{A \mid \Gamma \vdash \{t\} : \mathit{Cell}}$$

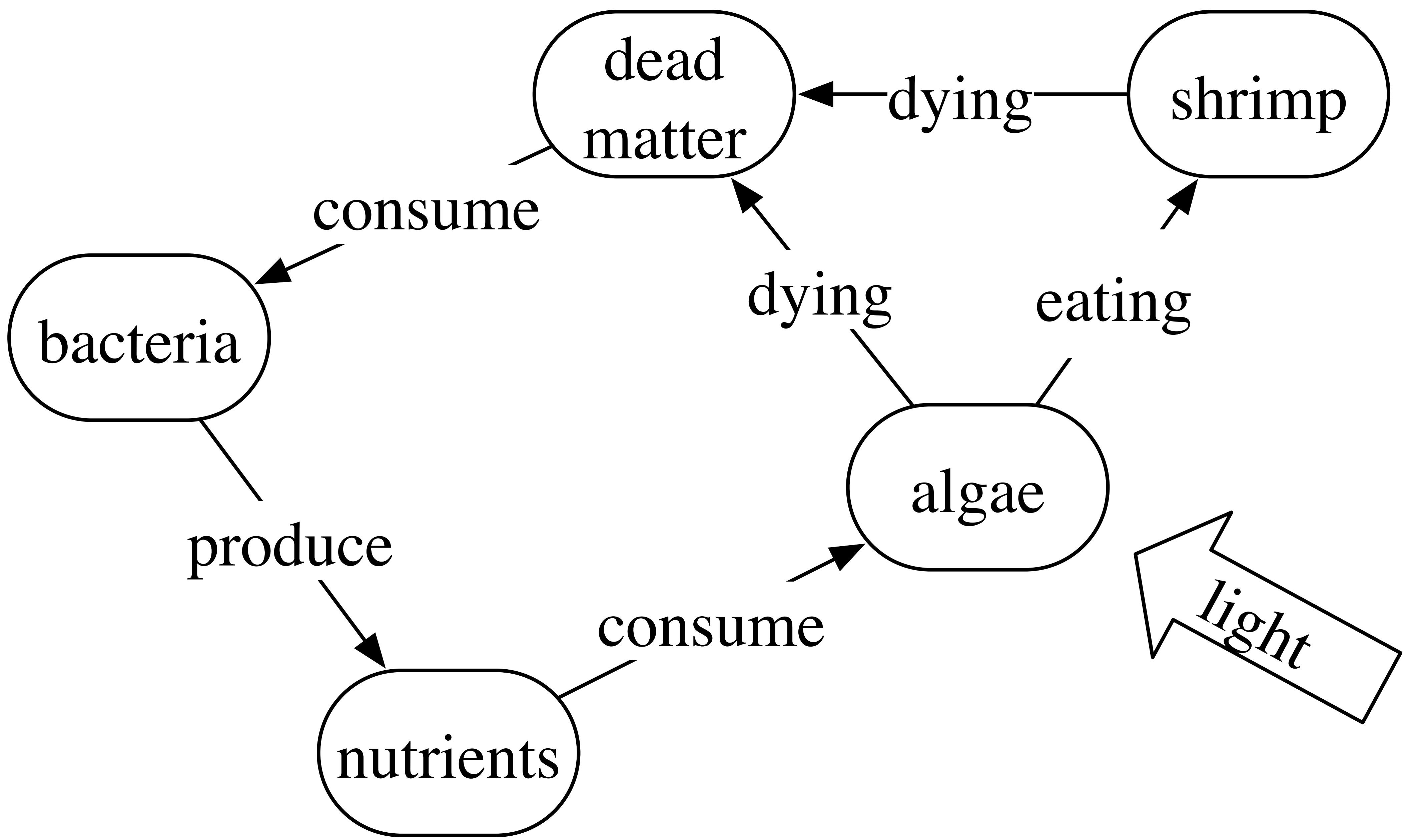
$$\frac{A \mid \Gamma \vdash t : \gamma}{A \mid \Gamma \vdash \text{peek } t : \gamma} \qquad \frac{A \mid \Gamma \vdash t : \mathit{Prm}_\alpha}{A \mid \Gamma \vdash \text{deref } t : \mathit{Vec}_\alpha} \qquad \frac{A \mid \Gamma \vdash t : \mathit{Cell}}{A \mid \Gamma \vdash \text{deref } t : \mathbb{F}} \qquad \frac{}{A \mid \Gamma \vdash \text{pc } \underline{n} : \mathbb{F}}$$

$$\frac{A \mid \Gamma \vdash t : \tau \quad \alpha \notin A}{\alpha, A \mid \Gamma \vdash \text{fusion } t : \mathit{Prm}_\alpha}.$$

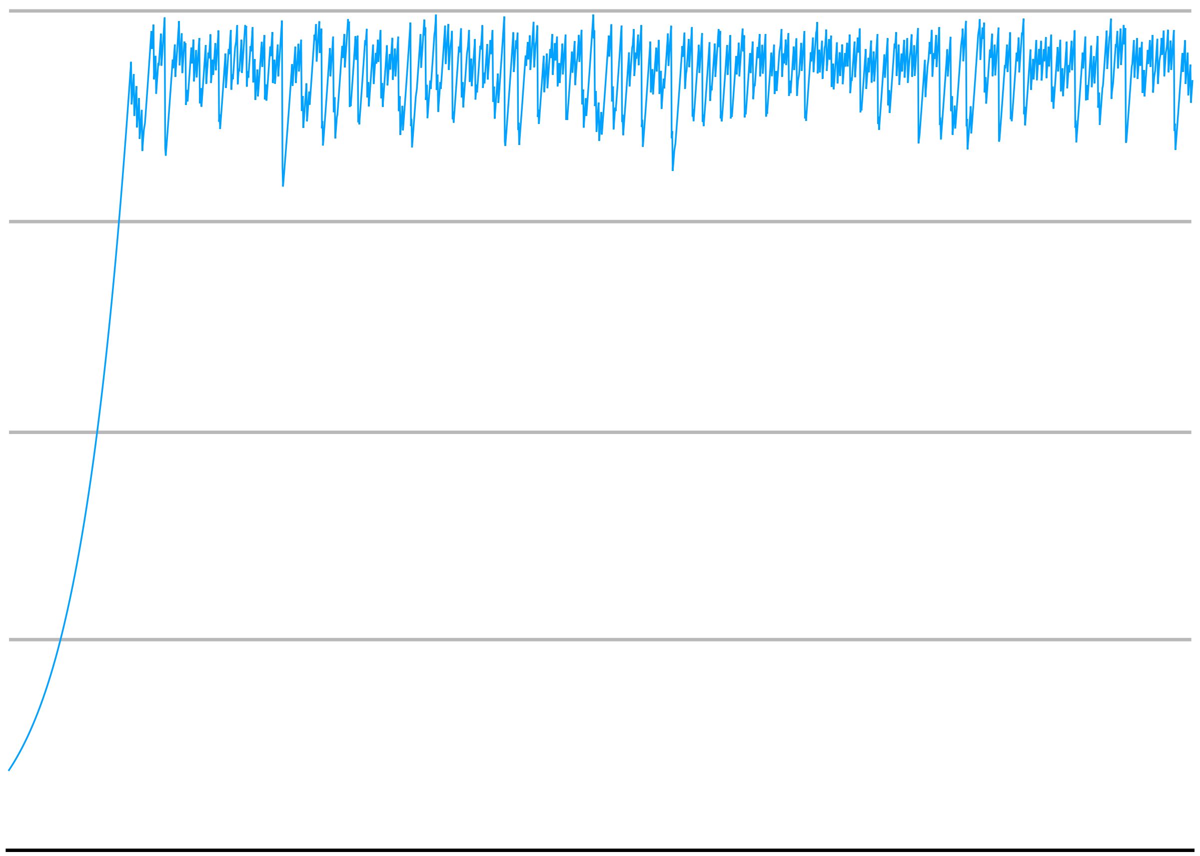
Typing via GADTs

```
type z = Z
type 'n s = S : 'n -> 'n s
type 'n symtensor = 'n * Dictionary.t
val fusion : 'a graph -> 'n symtensor
```

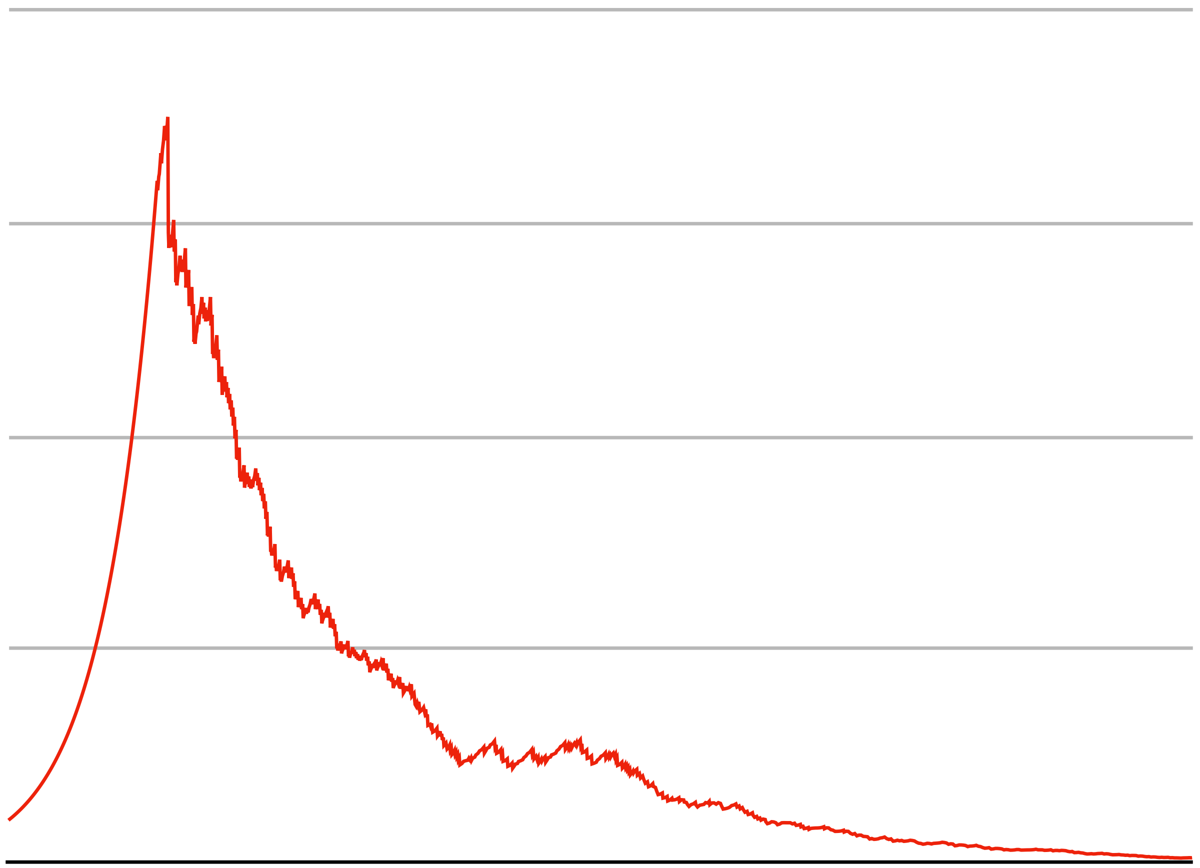
Example: Ecosphere



Example: Ecosphere



Shrimp population oscillates
(*loss* ⬇)



Shrimp die-off
(*loss* ⬆)

Example: Ecosphere

```
let createPrms _ = pc 1000.0, pc 500.0, pc 10.0, pc 100.0, pc 1000.0

let createCells _ =
        let nut = cell (lift 0.0) in
        let alg = cell (lift 0.0) in
        let shr = cell (lift 0.0) in
        let bac = cell (lift 0.0) in
        let ded = cell (lift 0.0) in
        (nut, alg, shr, bac, ded)

let createModel _ =
        let (init_n, init_a, init_s, init_b, init_d) = createPrms () in
        let (nut, alg, shr, bac, ded) = createCells () in
        link nut [%syf nut_eqn init_n nut alg bac ded ];
        link alg [%syf alg_eqn init_a alg nut shr ];
        link shr [%syf shr_eqn init_s shr alg ];
        link bac [%syf bac_eqn init_b ded bac ];
        link ded [%syf ded_eqn init_d ded bac alg nut shr ];
        (init_n, init_a, init_s, init_b, init_d, nut, alg, shr, bac, ded)

let optimise _ =
        let model = createModel () in
        let (_,_,_,_,_,_,_,shr,_,_) = model in
        let ps = fusion shr in
        gradient_descent model ps loss 1000
```

Conclusions

- operational semantics for specification
- reason about type safety
- reason about (in-principle) efficiency
- extremely flexible -- exotic operations
    - including dataflow primitives, graph abstraction, etc.
- reason about observational equivalence

References

- *Transparent Synchronous Dataflow.* Steven W. T. Cheung, Dan R. Ghica, Koko Muroya. arXiv:1910.09579.
- *Local Reasoning for Robust Observational Equivalence.* Dan R. Ghica, Koko Muroya, Todd Waugh Ambridge. arXiv:1907.01257
- *The Dynamic Geometry of Interaction Machine: A Token-Guided Graph Rewriter*. Koko Muroya, Dan R. Ghica. arXiv:1803.00427 (and CSL'17 and LMCS'19)
- Abductive functional programming, a semantic approach. Koko Muroya, Steven Cheung, Dan R. Ghica. arXiv:1710.03984 (also LICS'18 and FLOPS'18)