

EVENT CORRELATION SYSTEMS À LA CARTE

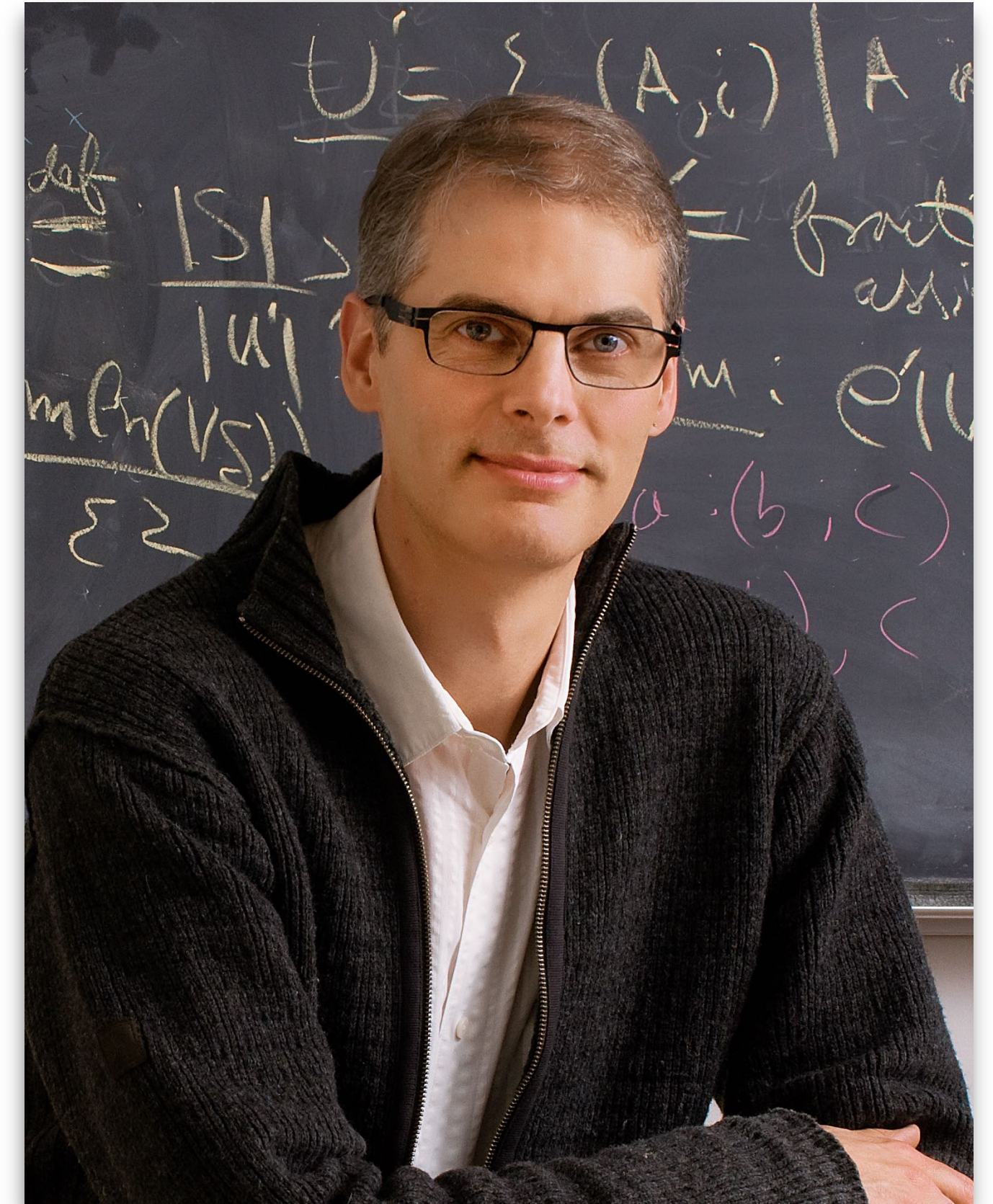
Oliver Bračevac

Software Technology Group @ TU Darmstadt, Germany

PurPL Seminar 2019-11-15

“PL people are good at taking **complicated domains** where people are creating new ways of doing things, but where it's too messy and **not well understood**, and **pulling out the abstractions and the core concepts.**”

— Benjamin C. Pierce

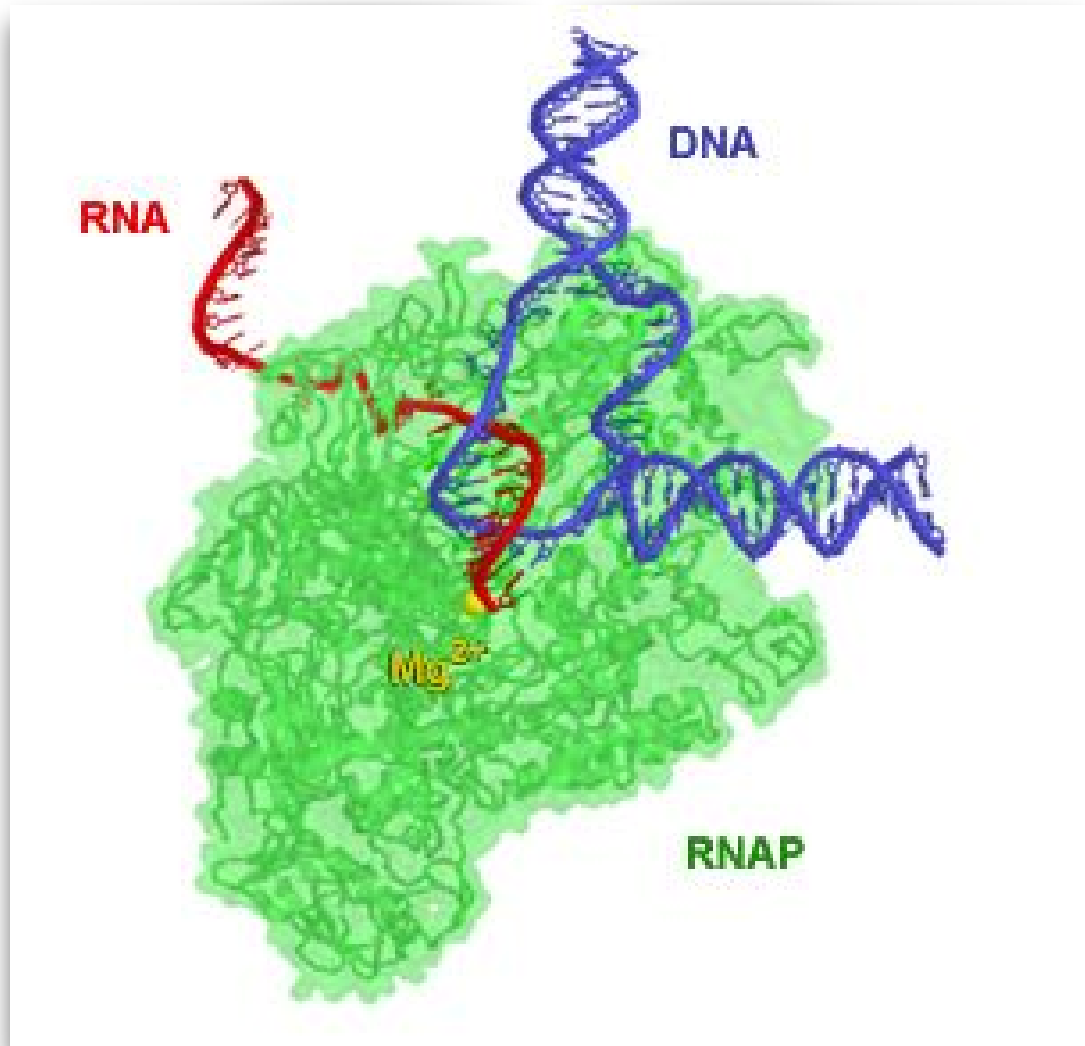


CONTEXT: EVENT CORRELATION

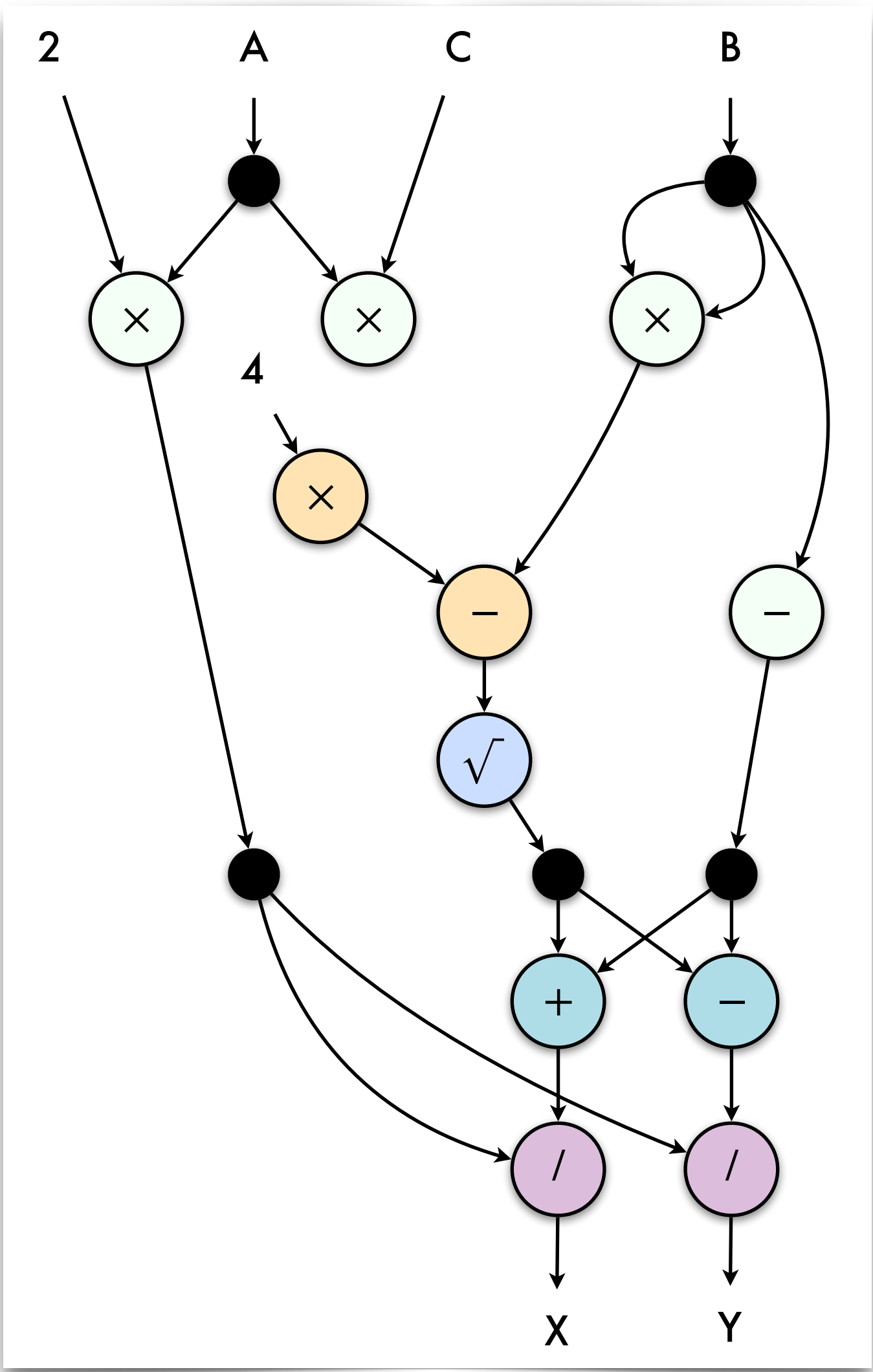
JOINING/MERGING OF INFORMATION FLOWS...



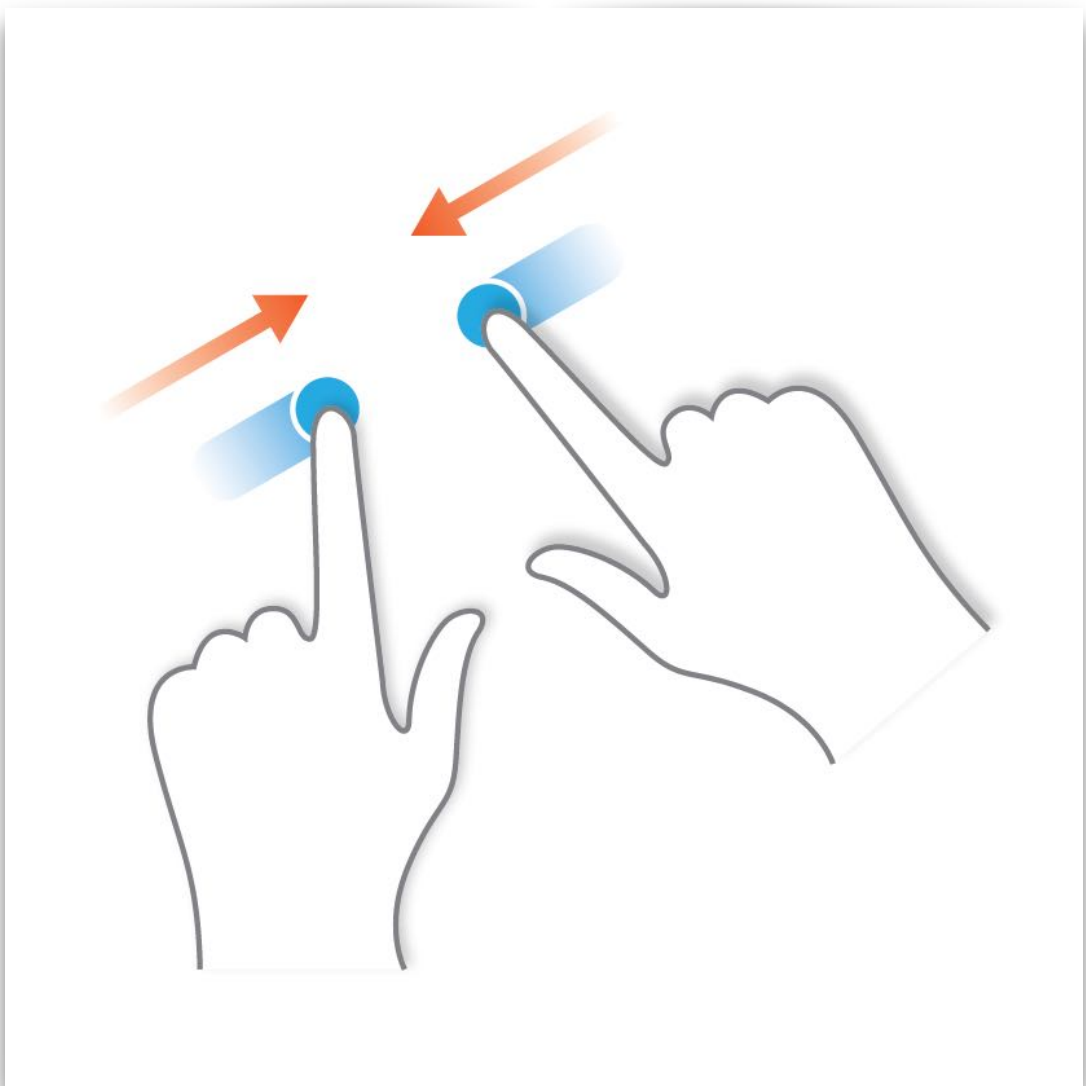
<https://pxhere.com/en/photo/138038>



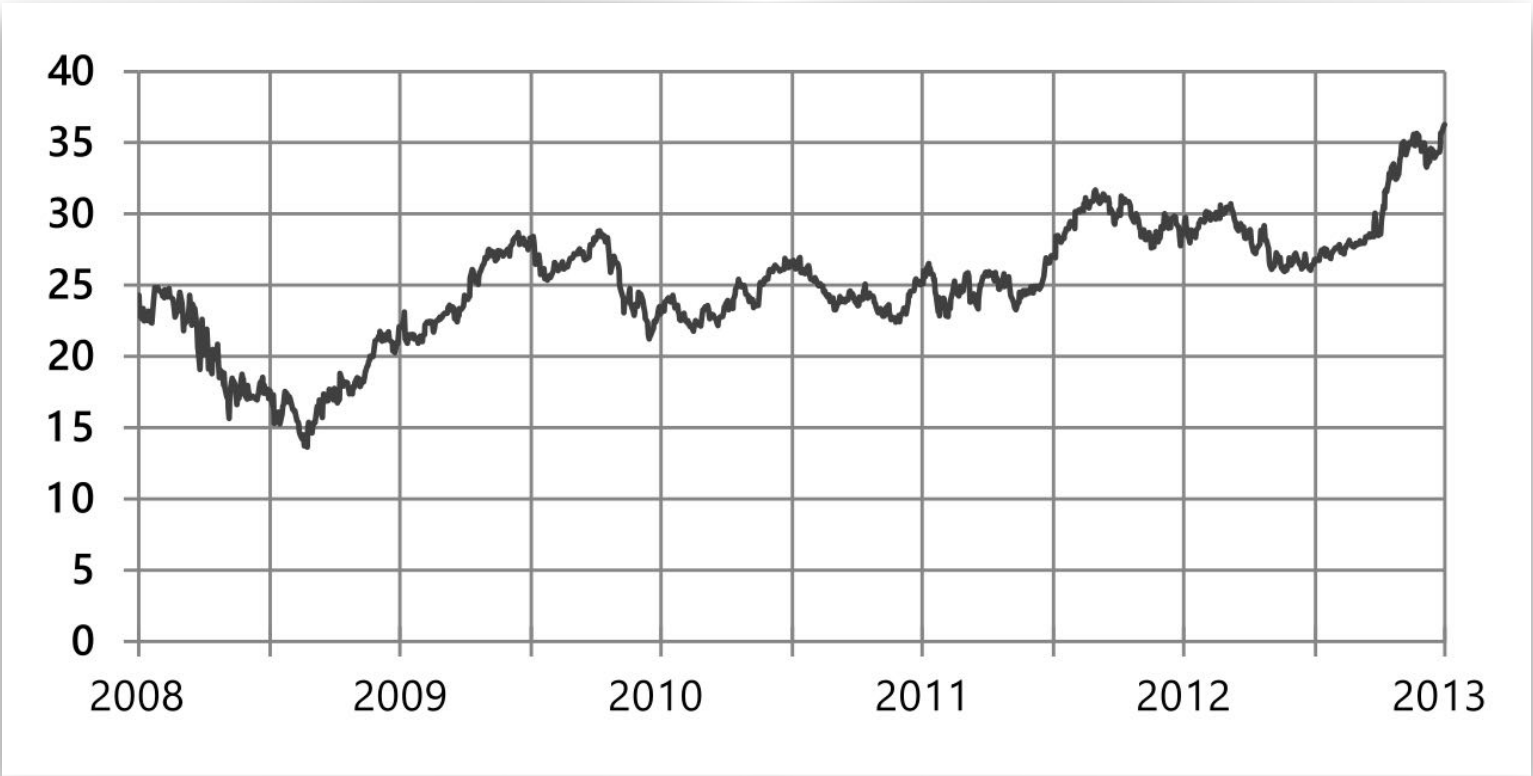
https://commons.wikimedia.org/wiki/File:RNAP_TEC_small.jpg



3



https://commons.wikimedia.org/wiki/File:Gestures_Two_Hand_Pinch.png



https://en.wikipedia.org/wiki/File:Microsoft_5-Year_Stock_History.svg

CONTEXT: EVENT CORRELATION

...HAS BEEN REINVENTED SEVERAL TIMES OVER

Complex Event Processing

```
PATTERN SEQ(Stock+ a[], Stock b)
WHERE skip_till_next_match(a[], b) {
  [symbol]
  and a[1].volume > 1000
  and a[i].price > avg(a[..i-1].price)
  and b.volume < 0.8*a[a.LEN].volume }
WITHIN 1 hour
SASE+ [Agrawal et al. '08]
```

Stream Processing

```
Select Istream(Close.item_id)
From Close[Now], Open[Range 5 Hours]
Where Close.item_id = Open.item_id
CQL [Arasu et al. '04]
```

Reactive Programming

```
val force = signal {
  mass() * accel()
}
```

ReScala [Salvaneschi et al. '14]

Concurrent Programming

```
def wait() & finished(r) =
  reply (Some r) to wait
or wait() & timeout() =
  reply None to wait
```

JoCaml [Conchon et al. '99]

CONTEXT: EVENT CORRELATION

...HAS BEEN REINVENTED SEVERAL TIMES OVER

Complex Event Processing

```
PATTERN SEQ(Stock+ a[], Stock b)
WHERE skip_till_next_match(a[], b) {
  [symbol]
  and a[1].volume > 1000
  and a[i].price > avg(a[..i-1].price)
  and b.volume < 0.8*a[a.LEN].volume }
WITHIN 1 hour
SASE+ [Agrawal et al. '08]
```

Reactive Programming

```
val force = signal {
  mass() * accel()
}
ReScala [Salvaneschi et al. '14]
```

Stream Processing

```
Select Istream(Close.item_id)
From Close[Now], Open[Range 5 Hours]
Where Close.item_id = Open.item_id
CQL [Arasu et al. '04]
```

Concurrent Programming

```
def wait() & finished(r) =
  reply (Some r) to wait
or wait() & timeout() =
  reply None to wait
JoCaml [Conchon et al. '99]
```

EVENT CORRELATION

MY THESIS IN A NUTSHELL

HOW DO WE OBTAIN EVENT CORRELATION SYSTEMS “À LA CARTE”?

”Build your own

We have made sure that the right items for a well-balanced meal are always present – but you are the Master Builder and so get to construct your own meal.
Adults choose **4 dishes**. Children choose **3 dishes + the special yellow brick**.

The Wild



Fantastic fish with a lemon cream, chervil and roasted buckwheat.



Glad gris (Happy pig!) with honey/mustard and tarragon.



Crispy chicken with basil mayonnaise (ketchup for the kids).



Al dente potatoes with lovage, truffle oil and almonds.



Powerful meatballs in tomato sauce.

The Cozy



Thin yum-yum-fries with saffron mayonnaise.



Tossede noodler (Crazy noodles!) with sesame sauce.



First class potato compote with leeks and cold pressed rapeseed oil.



Happy sour dough bread with butter and smoked salt.

The Crispy



Cool cabbage salad with parsley, beans and vinaigrette.



Crispy and raw vegetables
Raw vegetables – carrot sticks, cucumber and tomatoes.



Tossed green crunchy salad with vinaigrette.



One twomato salad with basil, onion and mozzarella.

The Grown up



Baked butternut with pumpkin seeds and hoisin sauce.



Grilled and wonderful cauliflower with grated cheese, browned butter and lemon.



Pan-fried and tasteful carrots with sea buckthorn.

1



2



3



Freely composable correlation features

```
graph TD; A[Red 1x2, Blue 1x2] --> B[Blue 1x3, Red 1x2, Orange 1x2]; C[Yellow 2x3, Orange 1x2, Green 1x2] --> D[Blue 1x2, Red 1x2, Orange 1x2]; E[Blue 1x2, Red 1x2, Orange 1x2] --> F[Blue 1x2, Red 1x2, Orange 1x2, Yellow 1x2, Green 1x2]; G[Blue 1x2, Red 1x2, Orange 1x2, Yellow 1x2, Green 1x2] --> H[Blue 1x2, Red 1x2, Orange 1x2, Yellow 1x2, Green 1x2, Blue 1x2]; I[Blue 1x2, Red 1x2, Orange 1x2, Yellow 1x2, Green 1x2, Blue 1x2] --> J[Blue 1x2, Red 1x2, Orange 1x2, Yellow 1x2, Green 1x2, Blue 1x2, Yellow 1x2];
```

An extensible “feature menu”

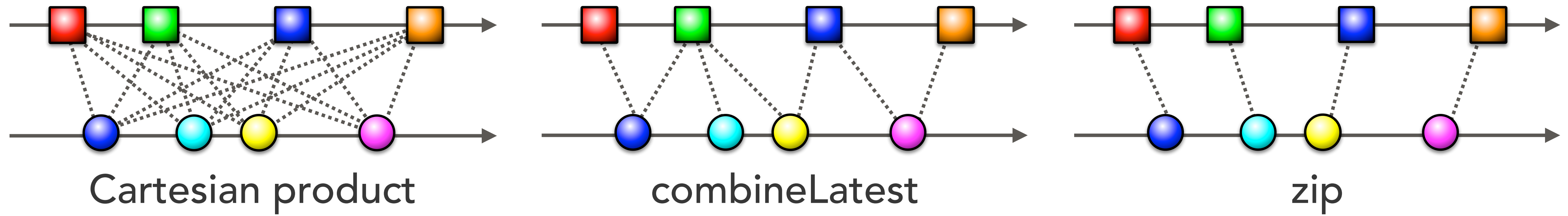
5

I

EXTENSIBLE CORRELATION SEMANTICS

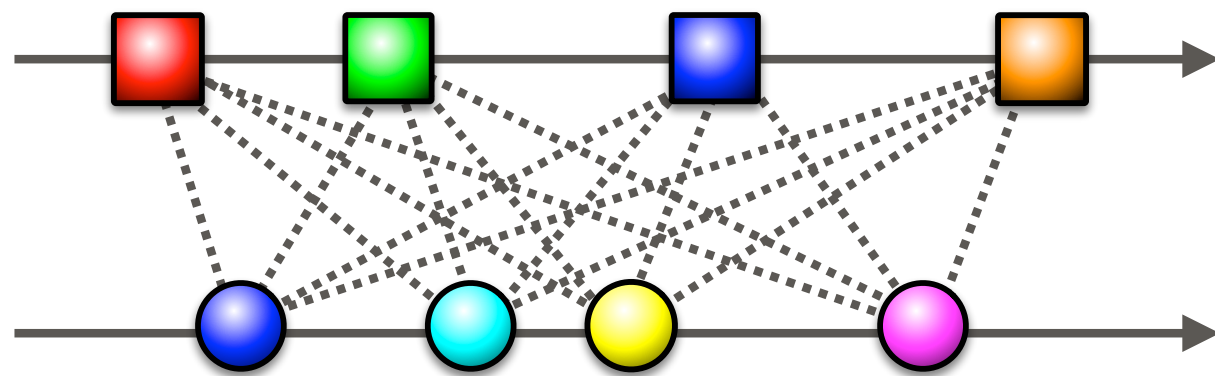
THE ESSENCE OF JOINING

COMPUTATIONAL INTERPRETATION



Enumerate the cartesian product, where (user-defined) side effects influence how the computation proceeds.

CARTESIUS: EXTENSIBLE SEMANTICS



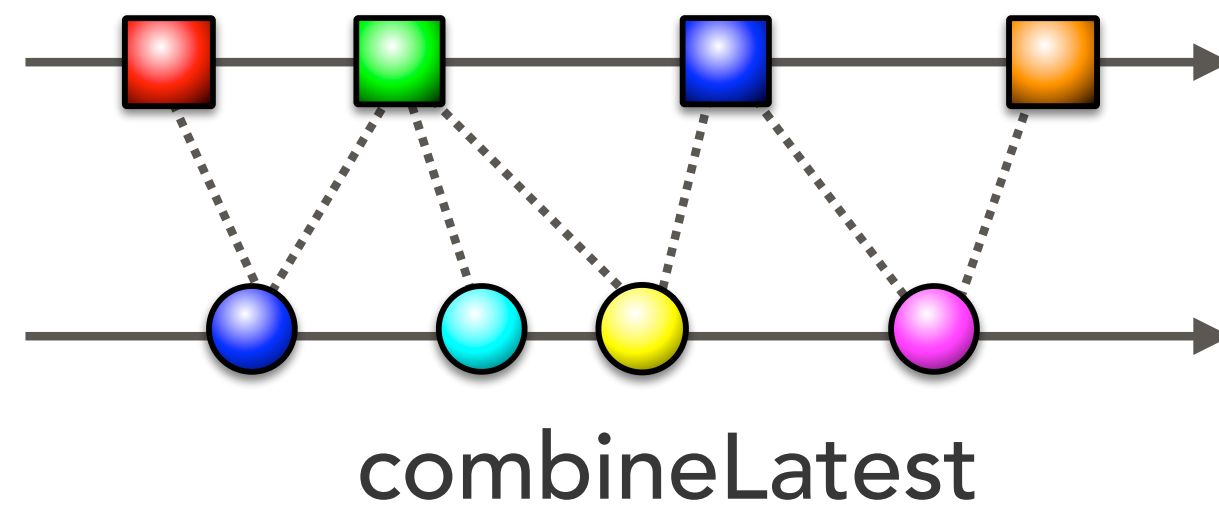
Cartesian product

```
let cart: R[A] -> R[B] -> R[(A,B)]  
  =  $\lambda$  r1 r2. correlate {  
    x from r1  
    y from r2  
    yield (x,y) }
```

We support

- Direct-style join specifications

CARTESIUS: EXTENSIBLE SEMANTICS



```
correlate {
```

```
  with (mostRecently r1)
```

```
    ⊞ (mostRecently r2)
```

```
  x from r1
```

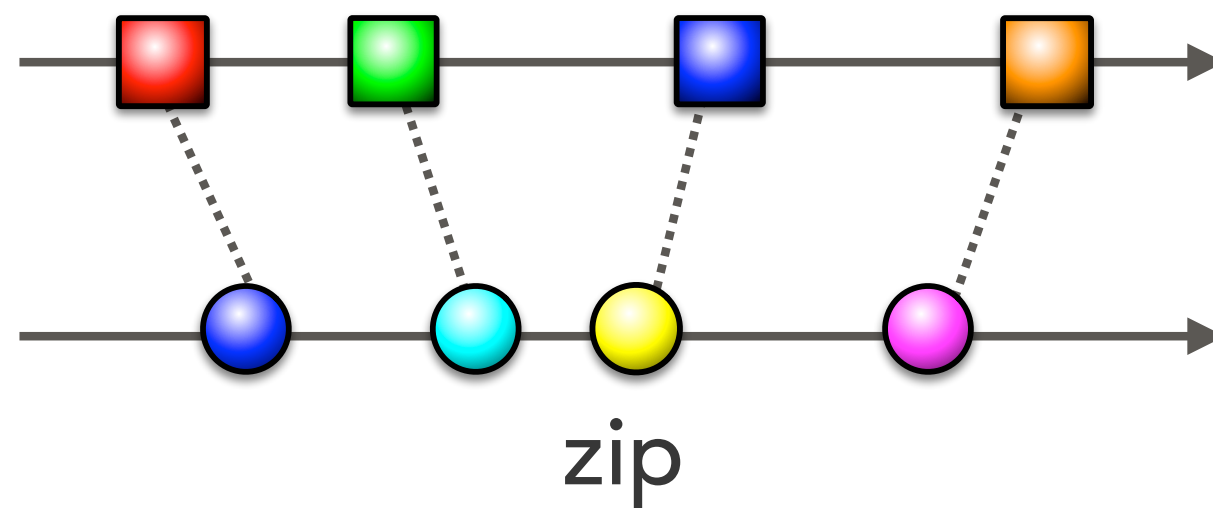
```
  y from r2
```

```
  yield (x,y)}
```

We support

- Direct-style join specifications
- Controllable matching behavior

CARTESIUS: EXTENSIBLE SEMANTICS



```
correlate {  
  with (mostRecently r1)  
    ⊞ (mostRecently r2)  
    ⊞ (aligning r1 r2)  
  x from r1  
  y from r2  
  yield (x,y) }
```

We support

- Direct-style join specifications
- Controllable matching behavior
- Mix-and-match composition of features
- Uniform programming model & extensibility

CARTESIUS: EXTENSIBLE SEMANTICS

More in Thesis: Timing and Time Windows

```
correlate {  
  x from r1  
  y from r2  
  where  
    abs(?timex - ?timey) < 5s  
  yield (x,y)}
```

```
x from r1  
y from r2  
yield (x,y)}
```

```
correlate {  
  with slidingWindow(1h)  
  x from r1  
  y from r2  
  yield (x,y)}
```

- Uniform programming model & extensibility

ALGEBRAIC EFFECTS AND HANDLERS

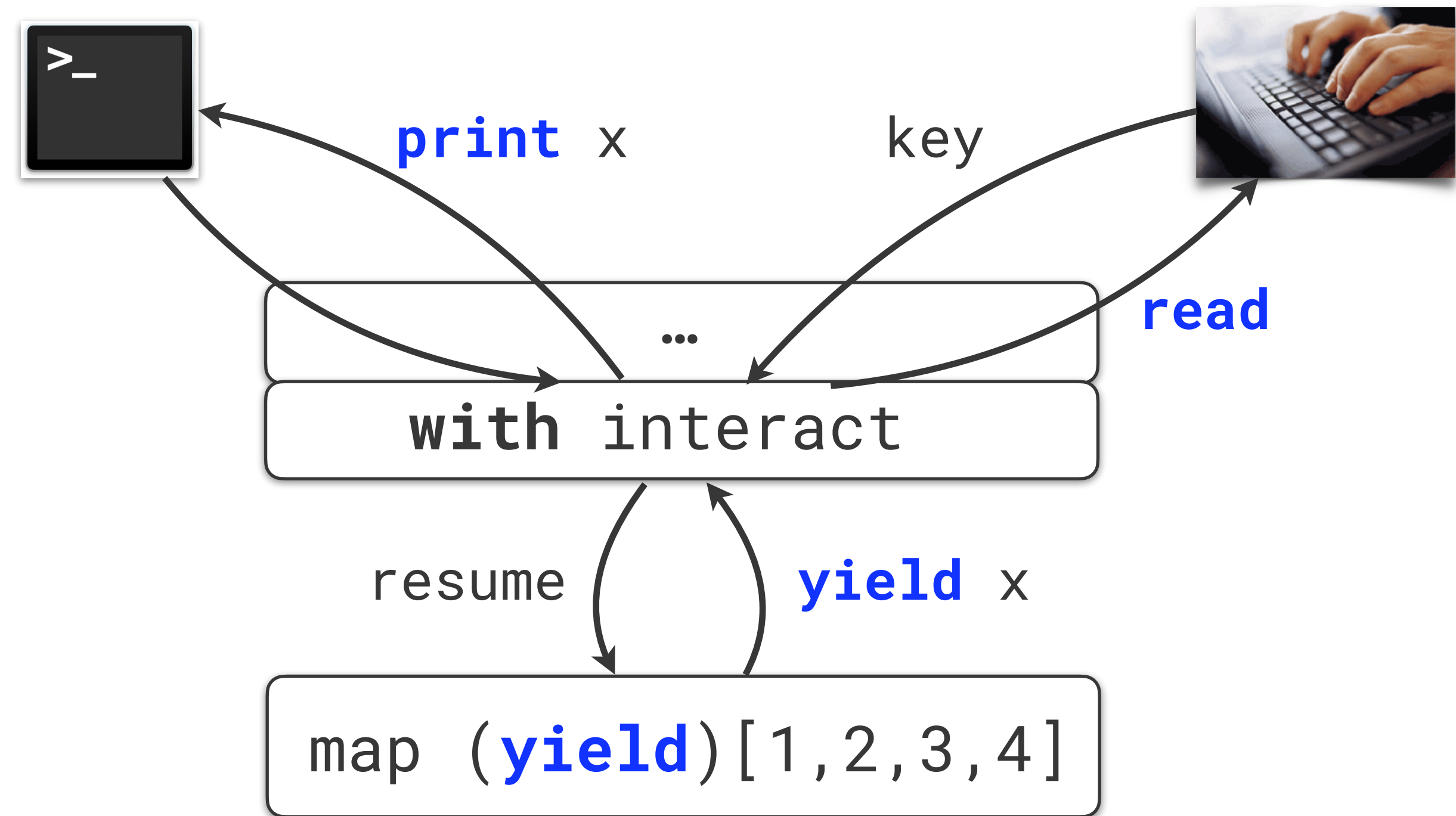
[PLOTKIN & POWER '03, PLOTKIN & PRETNAR '09]

Effect Interfaces:

yield: $\text{Int} \rightarrow \text{Unit}$

Effect Handlers:

```
let interact = handler
| yield val resume ->
  print val;
match read <>
| \cr -> resume <>
| _ -> <>
```



Effect Types:

$\text{interact}: \forall \alpha \mu. \{\alpha\}^{\langle \text{yield}, \mu \rangle} \rightarrow \langle \text{print}, \text{read}, \mu \rangle \text{Unit}$

Handler composition:

$h_1 \boxplus h_2 := \lambda tn. \text{with } h_1 (\text{with } h_2 (tn \langle \rangle))$

CORRELATE BY HANDLING

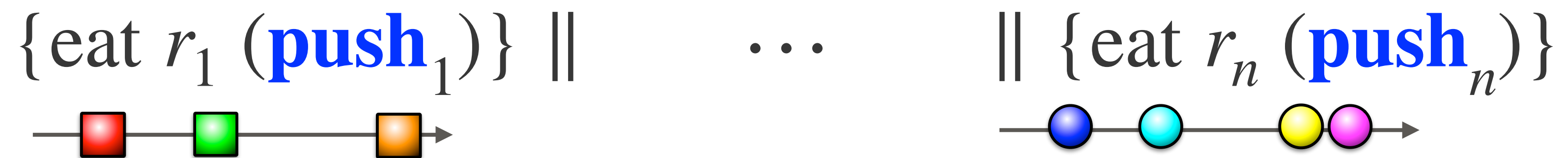
DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS [BRACEVAC ET AL. 2018]

Idea: Events = Effects, Correlating = Handling

CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS [BRACEVAC ET AL. 2018]

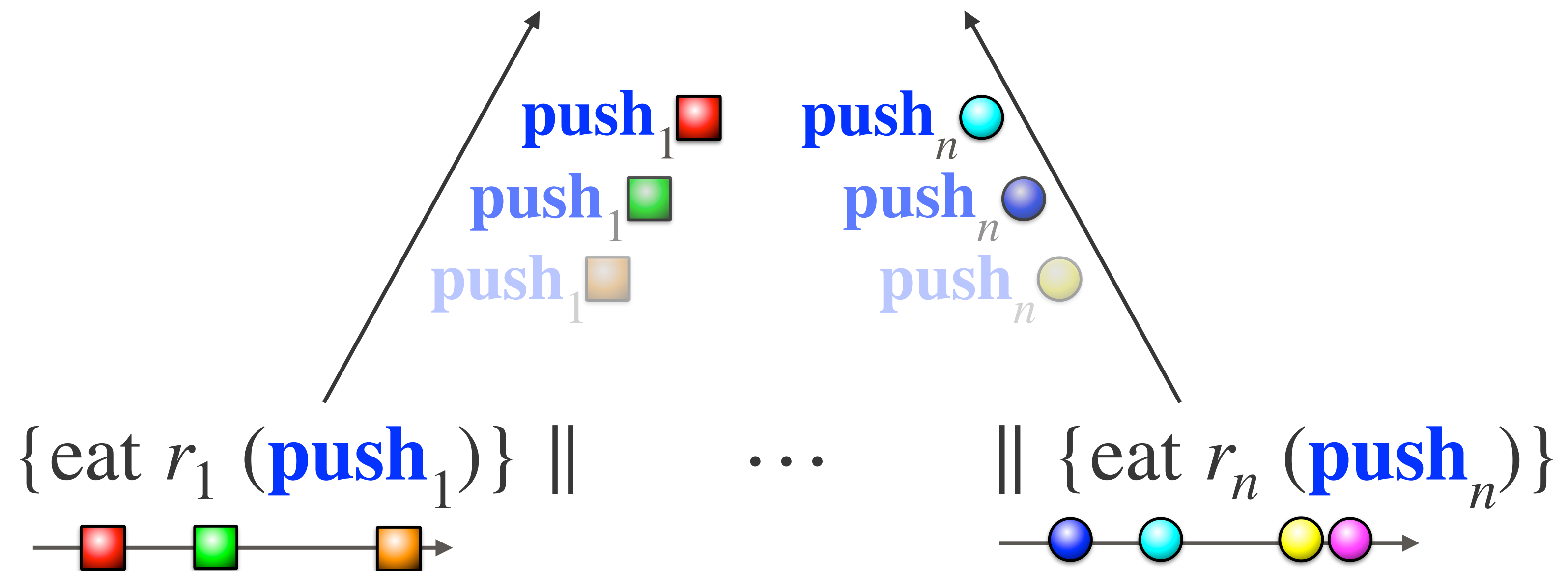
Idea: Events = Effects, Correlating = Handling



CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS [BRACEVAC ET AL. 2018]

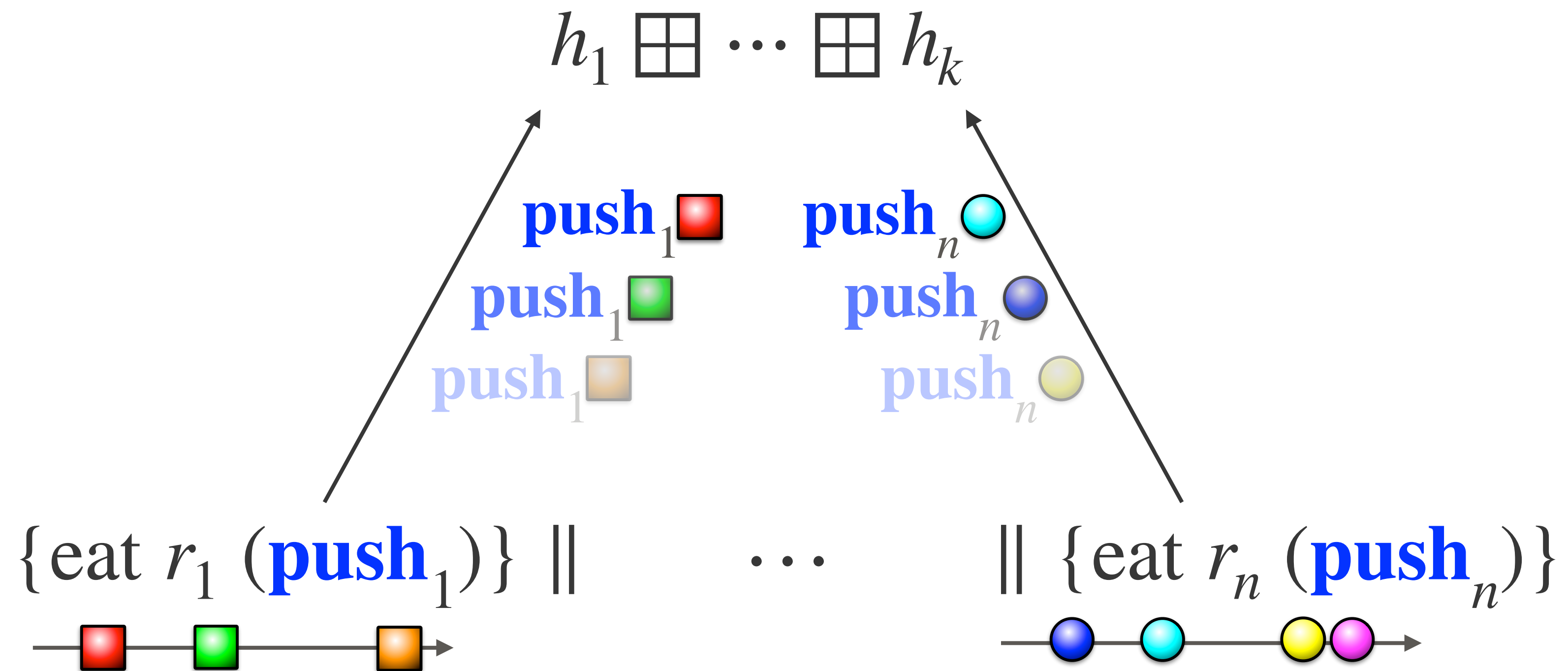
Idea: Events = Effects, Correlating = Handling



CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS [BRACEVAC ET AL. 2018]

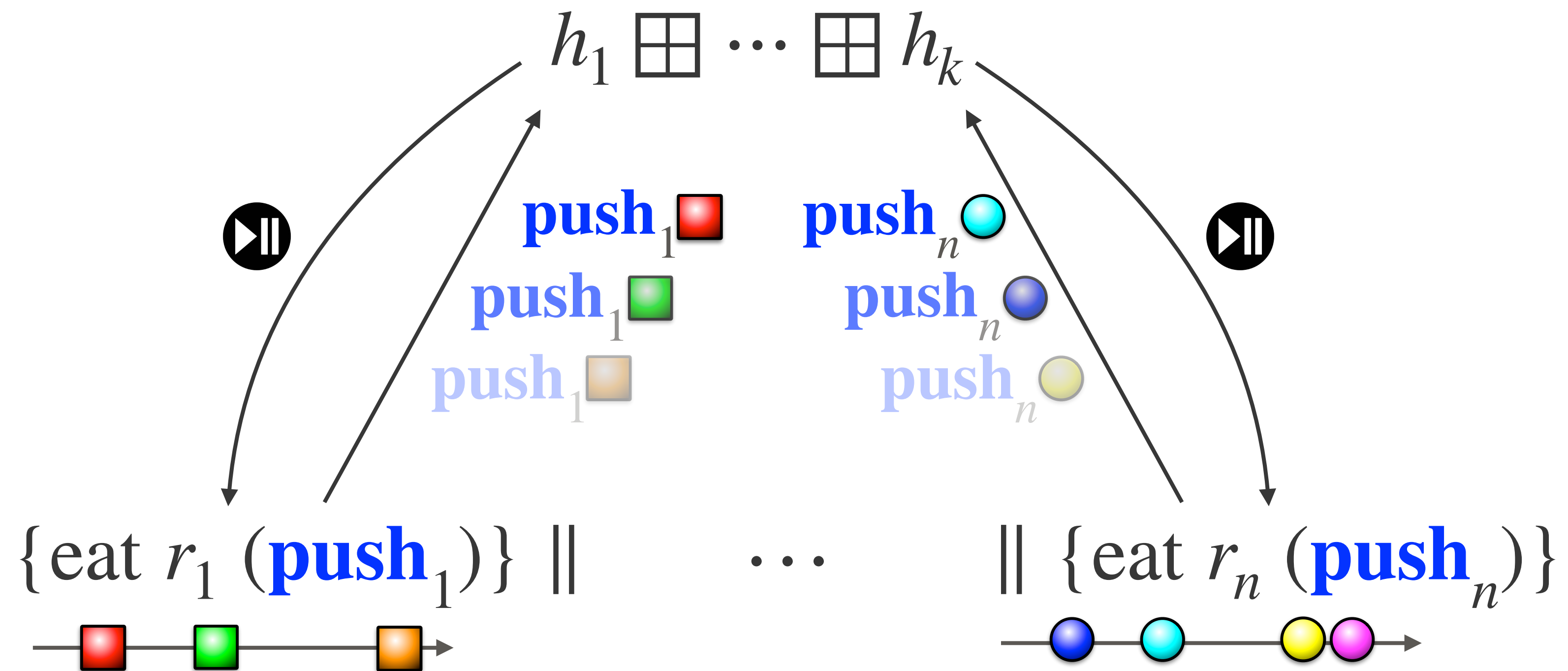
Idea: Events = Effects, Correlating = Handling



CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS [BRACEVAC ET AL. 2018]

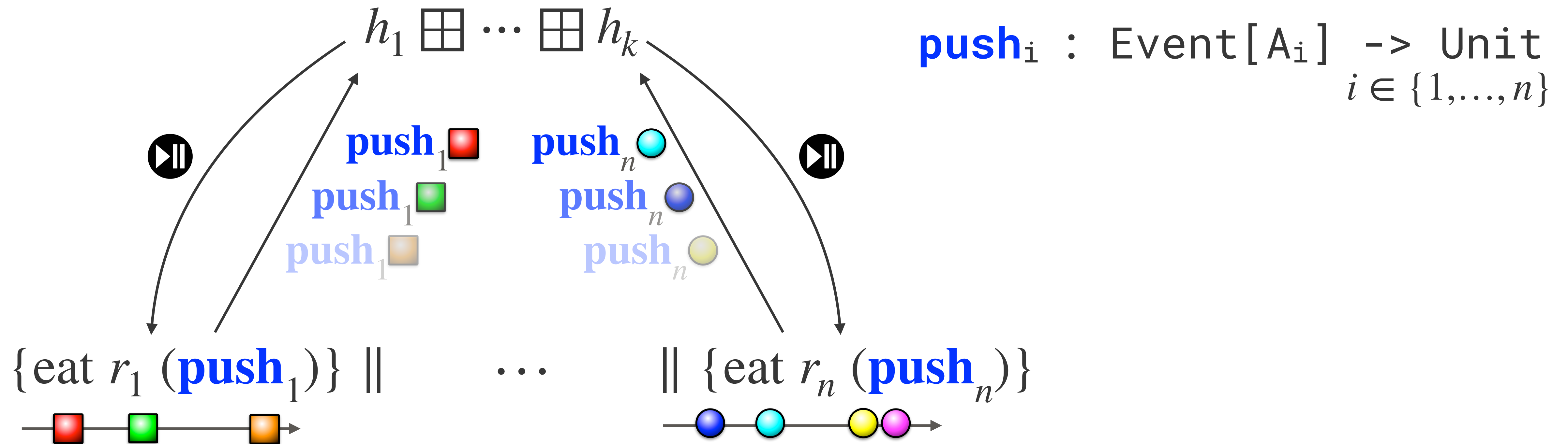
Idea: Events = Effects, Correlating = Handling



CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS [BRACEVAC ET AL. 2018]

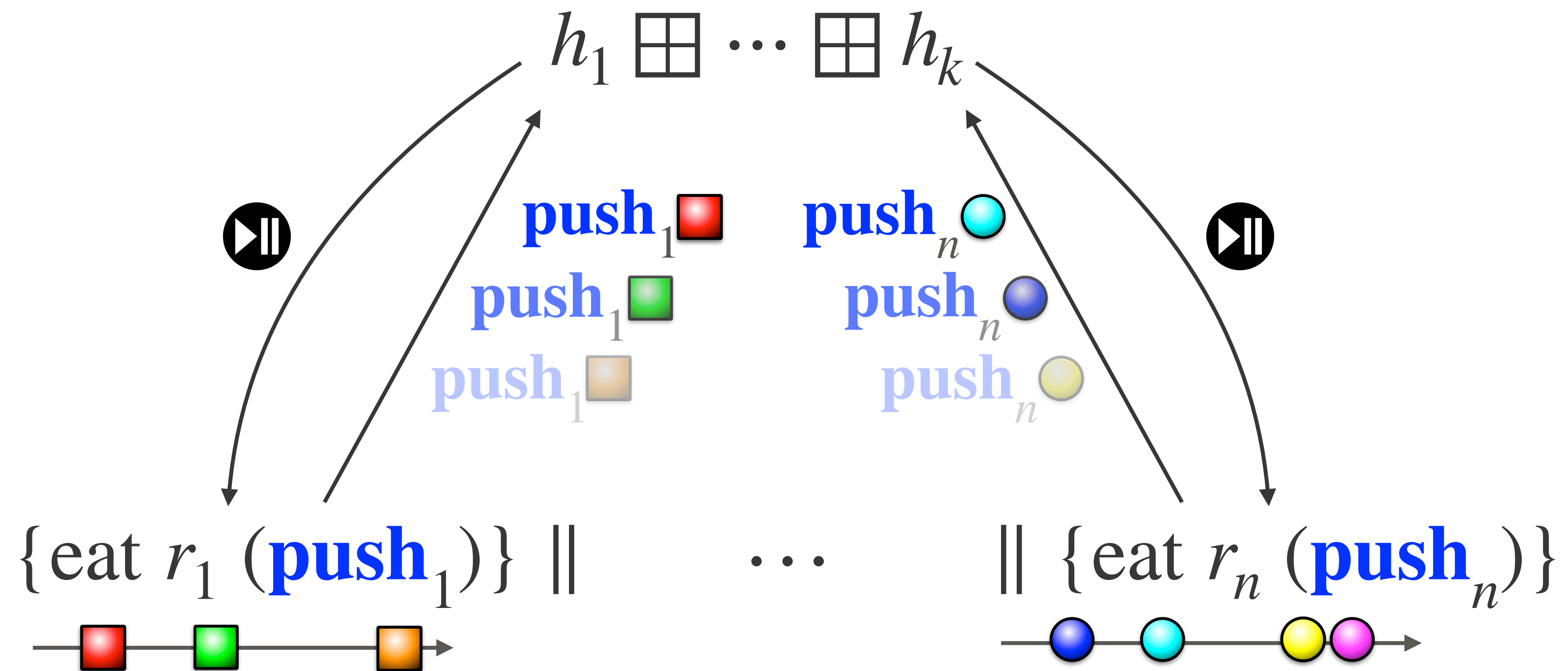
Idea: Events = Effects, Correlating = Handling



CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS [BRACEVAC ET AL. 2018]

Idea: Events = Effects, Correlating = Handling



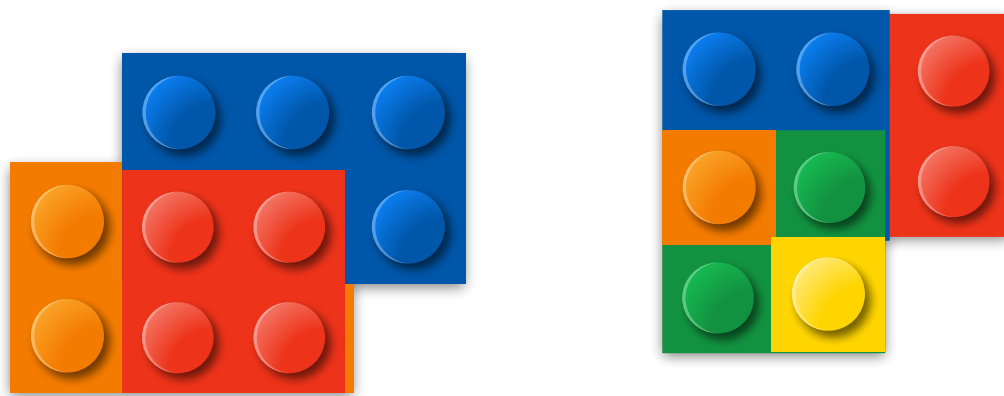
https://commons.wikimedia.org/wiki/File:Scoubidou_6_strands.jpg

CORRELATE BY HANDLING

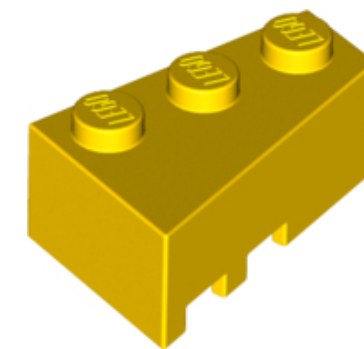
DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS [BRACEVAC ET AL. 2018]

$$h_1 \boxplus \cdots \boxplus h_k : \{ \text{Unit} \} \langle \text{push}_1, \dots, \text{push}_n, \text{async} \boxed{\varepsilon} \rangle \rightarrow \langle \text{async} \boxed{\varepsilon} \rangle \text{Unit}$$

Legos!



New brick forms!



THE BIG PICTURE

```
let  $r_{\text{out}}$  =  
  correlate {  
    with  $h_1 \boxplus \dots \boxplus h_k$   
     $x_1$  from  $r_1$   
    ...  
     $x_n$  from  $r_n$   
    where p  
    yield e }
```


THE BIG PICTURE

```
let rout =  
  correlate {  
    with h1 ⊞ ... ⊞ hk  
    x1 from r1  
    ...  
    xn from rn  
    where p  
    yield e }
```

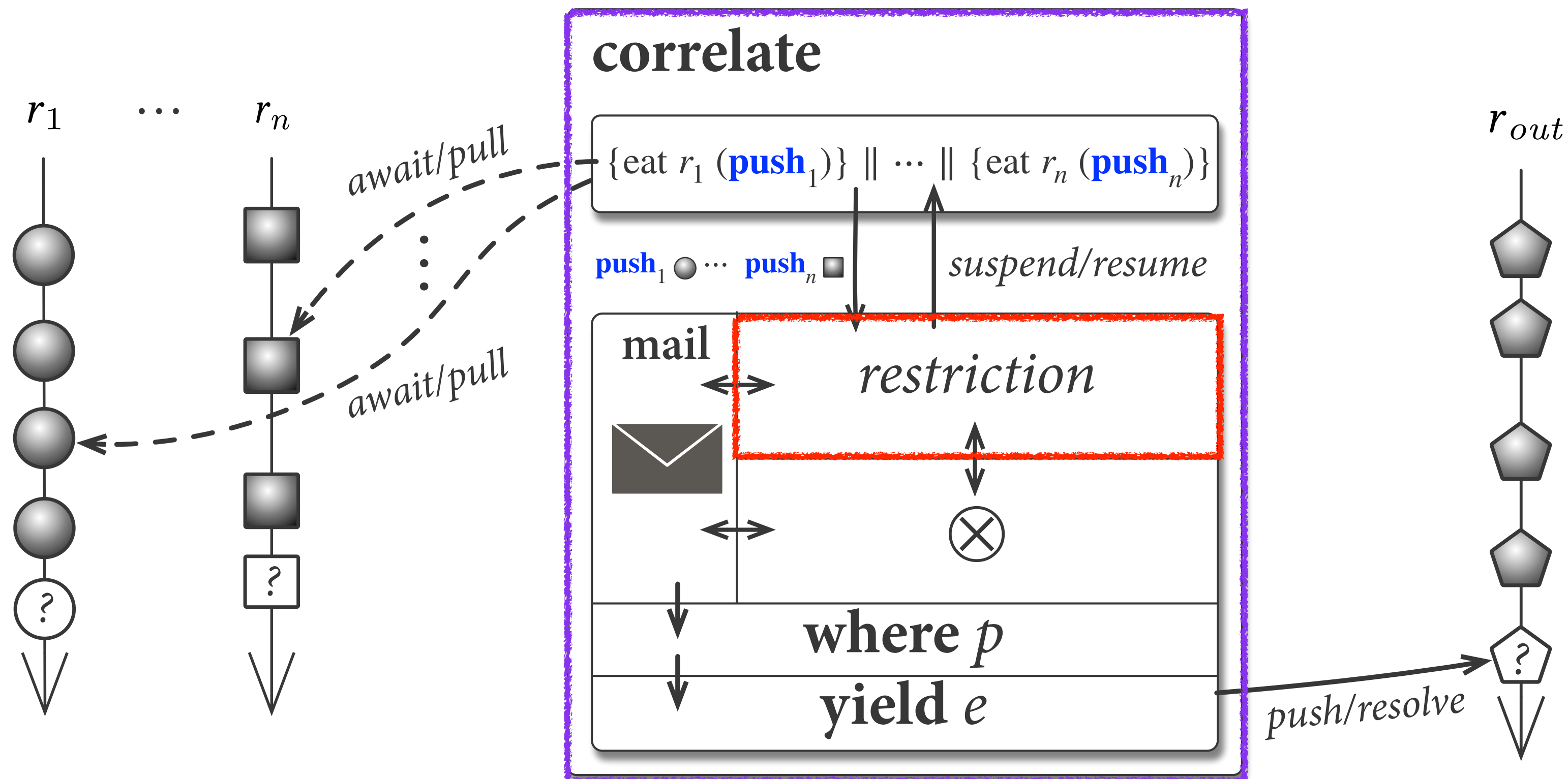
$$\text{let } r_{\text{out}} = \text{with } (h_{\otimes} \boxplus h_1 \boxplus \cdots \boxplus h_k) \\ \{ \text{eat } r_1 (\text{push}_1) \} \parallel \cdots \parallel \{ \text{eat } r_n (\text{push}_n) \}$$

THE BIG PICTURE

```

let rout =
  correlate {
    with h1 ⊞ ... ⊞ hk
    x1 from r1
    ...
    xn from rn
    where p
    yield e }

let rout = with (h⊗ ⊞ h1 ⊞ ... ⊞ hk)
  {eat r1 (push1)} || ... || {eat rn (pushn)}
  
```



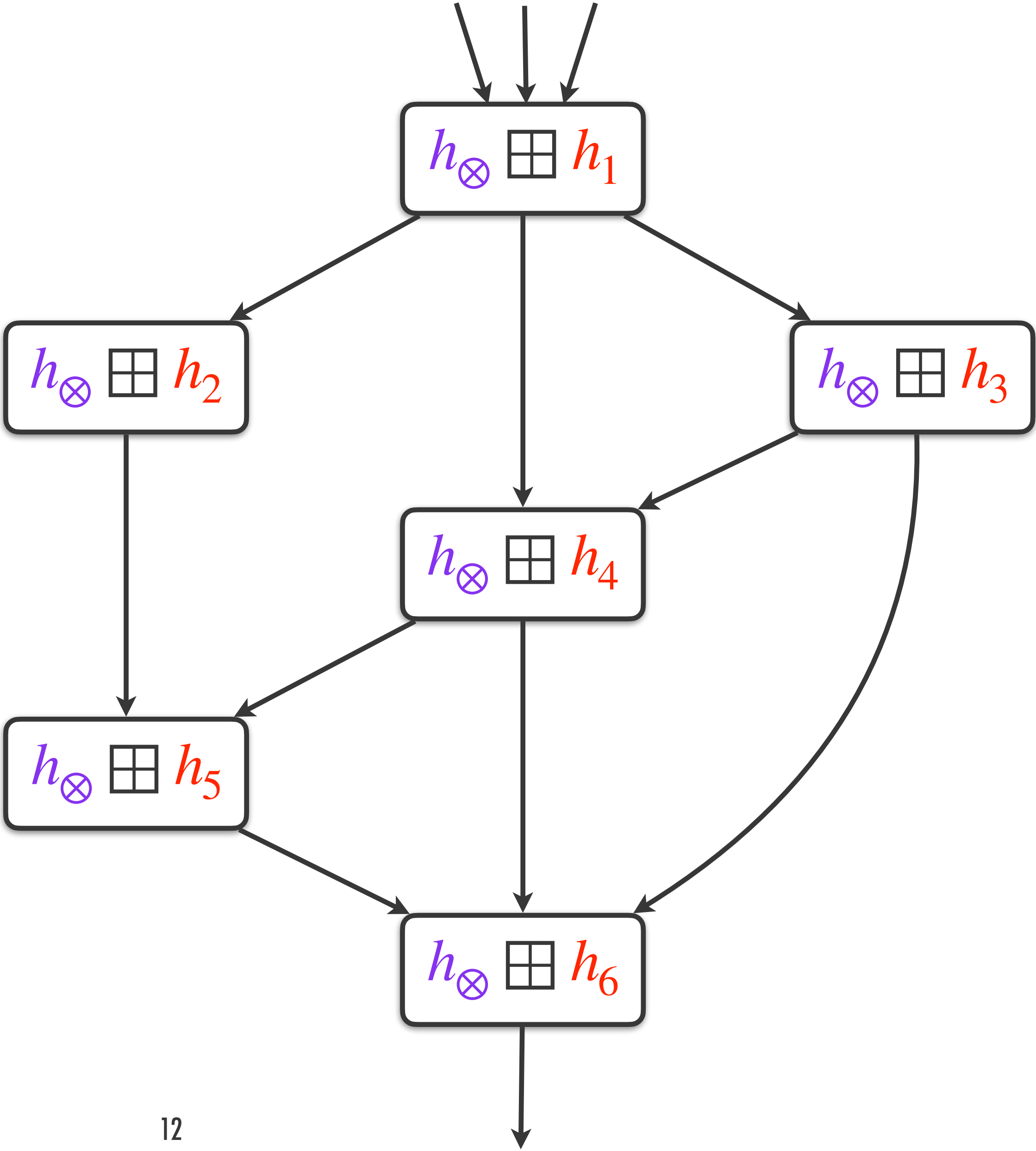
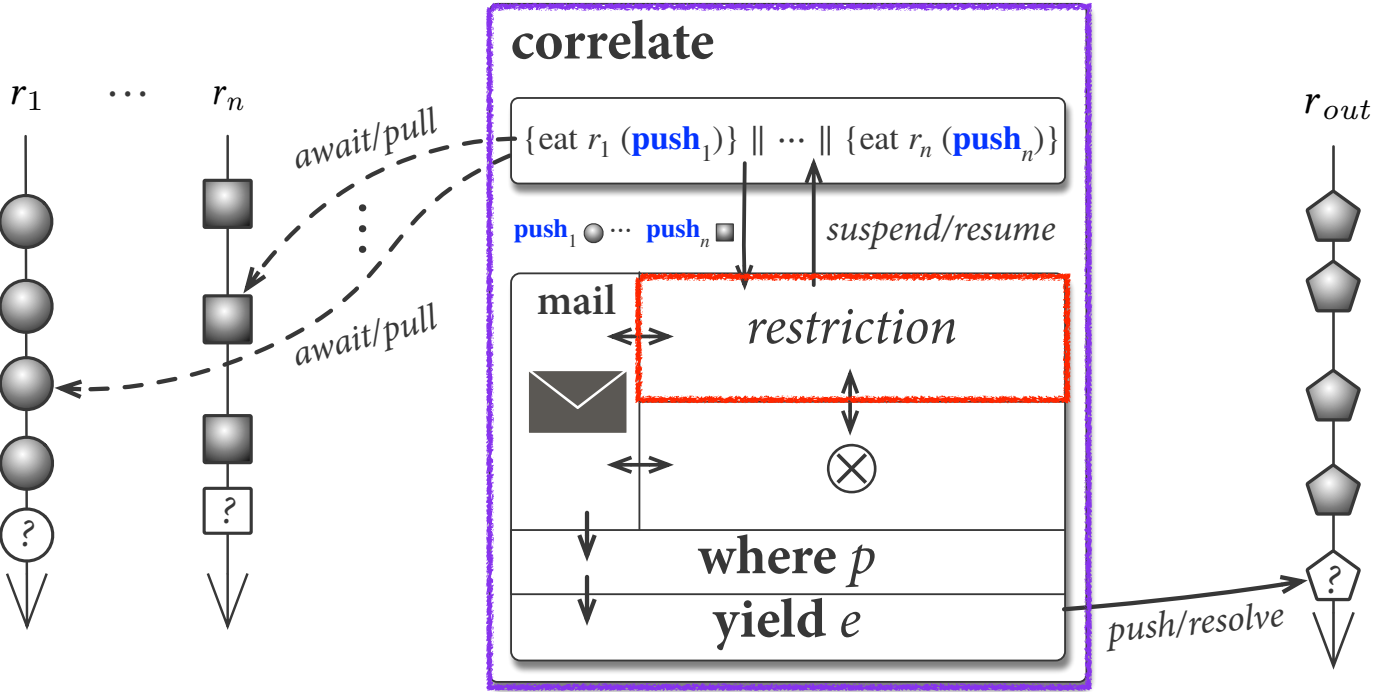
THE BIG PICTURE

```

let rout =
  correlate {
    with h1 ⊞ ... ⊞ hk
    x1 from r1
    ...
    xn from rn
    where p
    yield e }
  
```

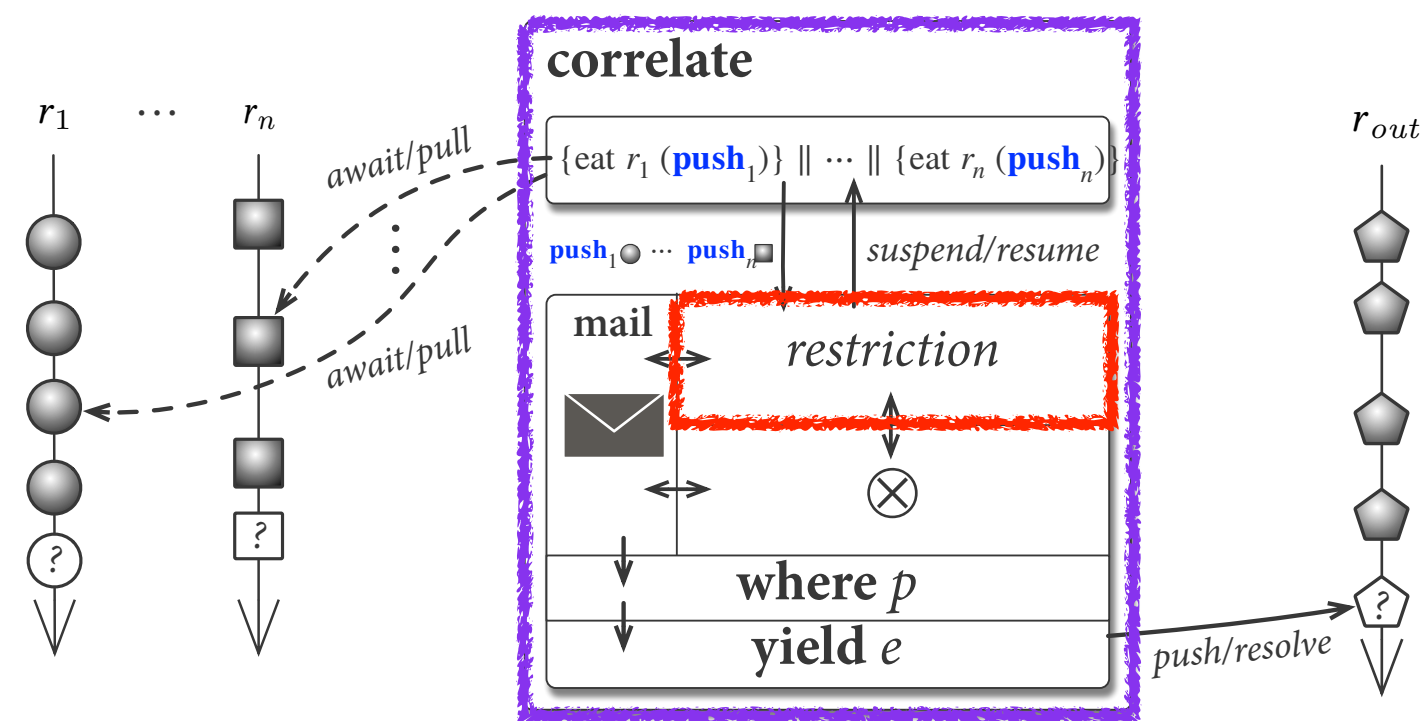
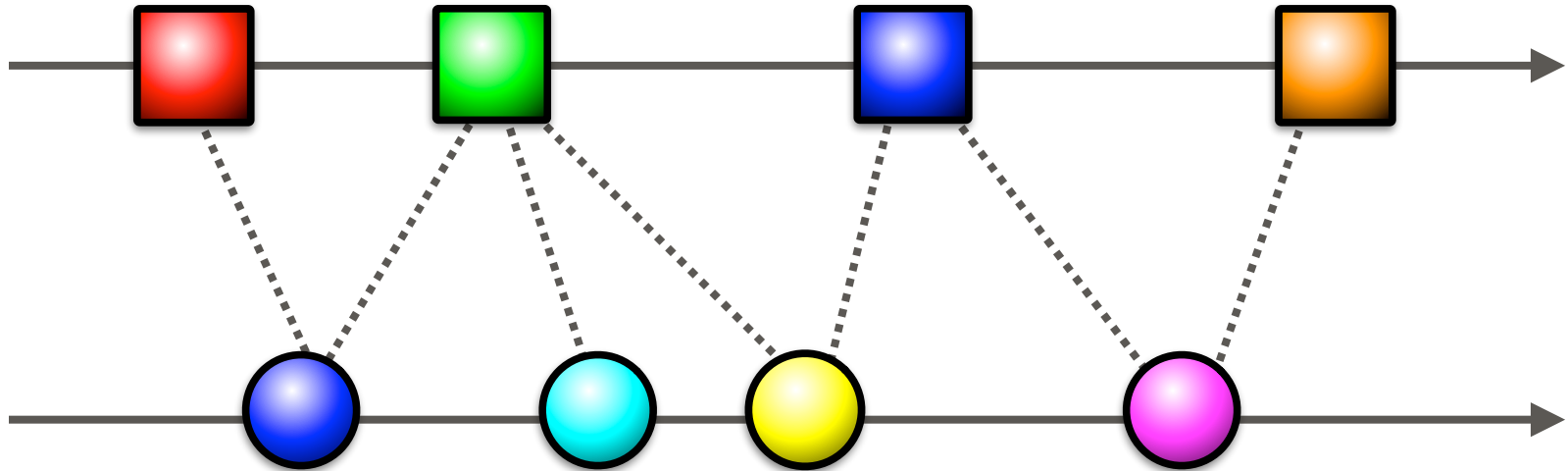
```

let rout = with (h⊗ ⊞ h1 ⊞ ... ⊞ hk)
  {eat r1 (push1)} || ... || {eat rn (pushn)}
  
```



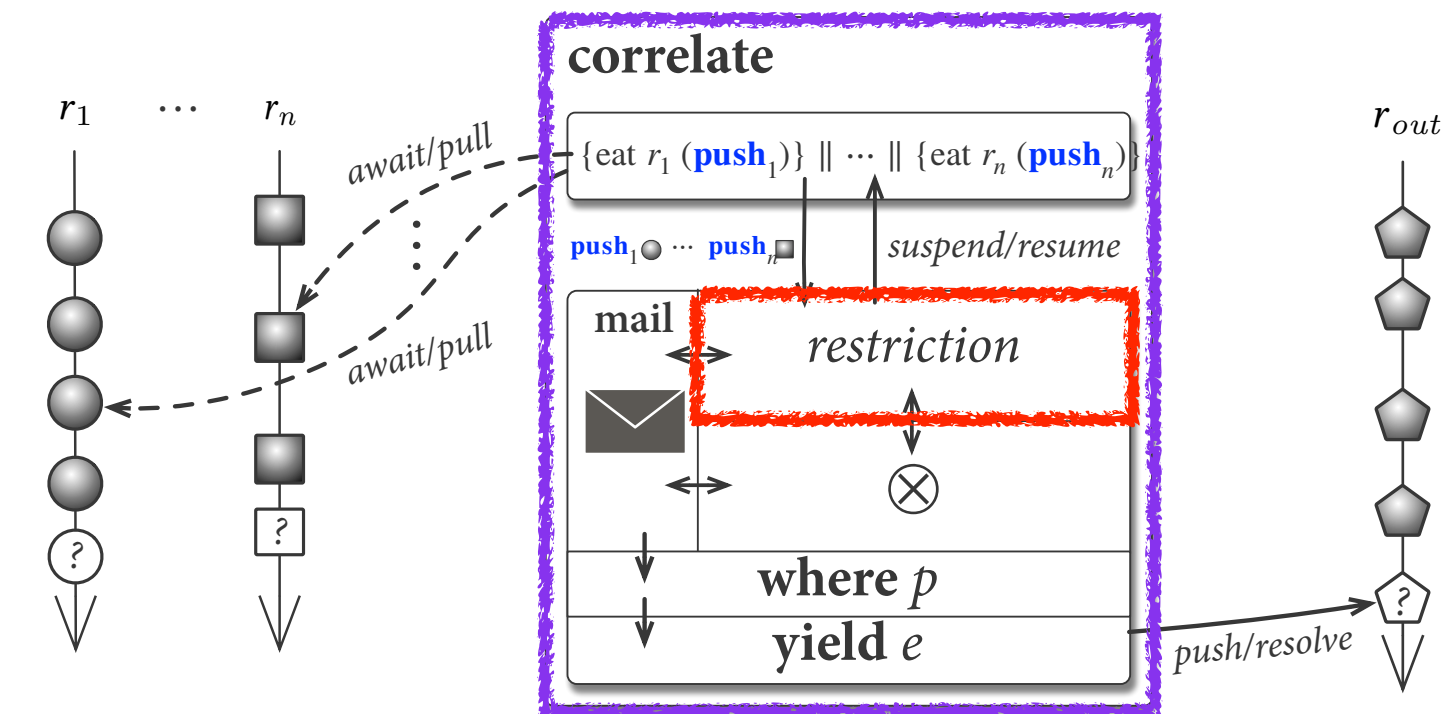
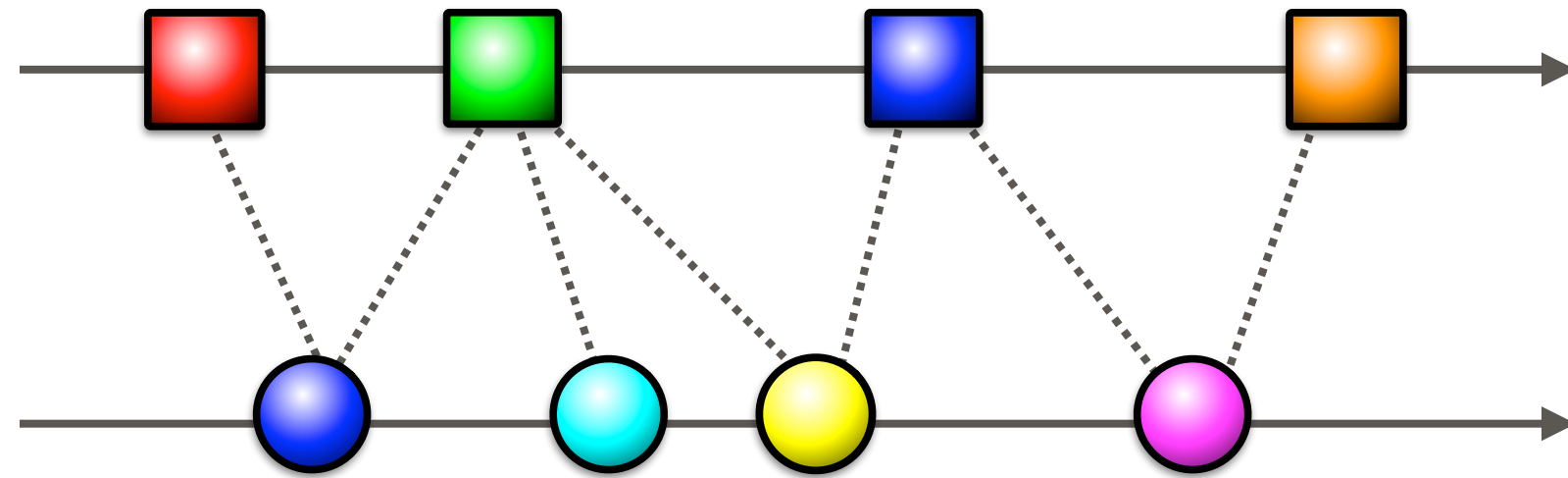
RESTRICTION HANDLERS

SPECIALISE THE NAIVE CARTESIAN PRODUCT



RESTRICTION HANDLERS

SPECIALISE THE NAIVE CARTESIAN PRODUCT

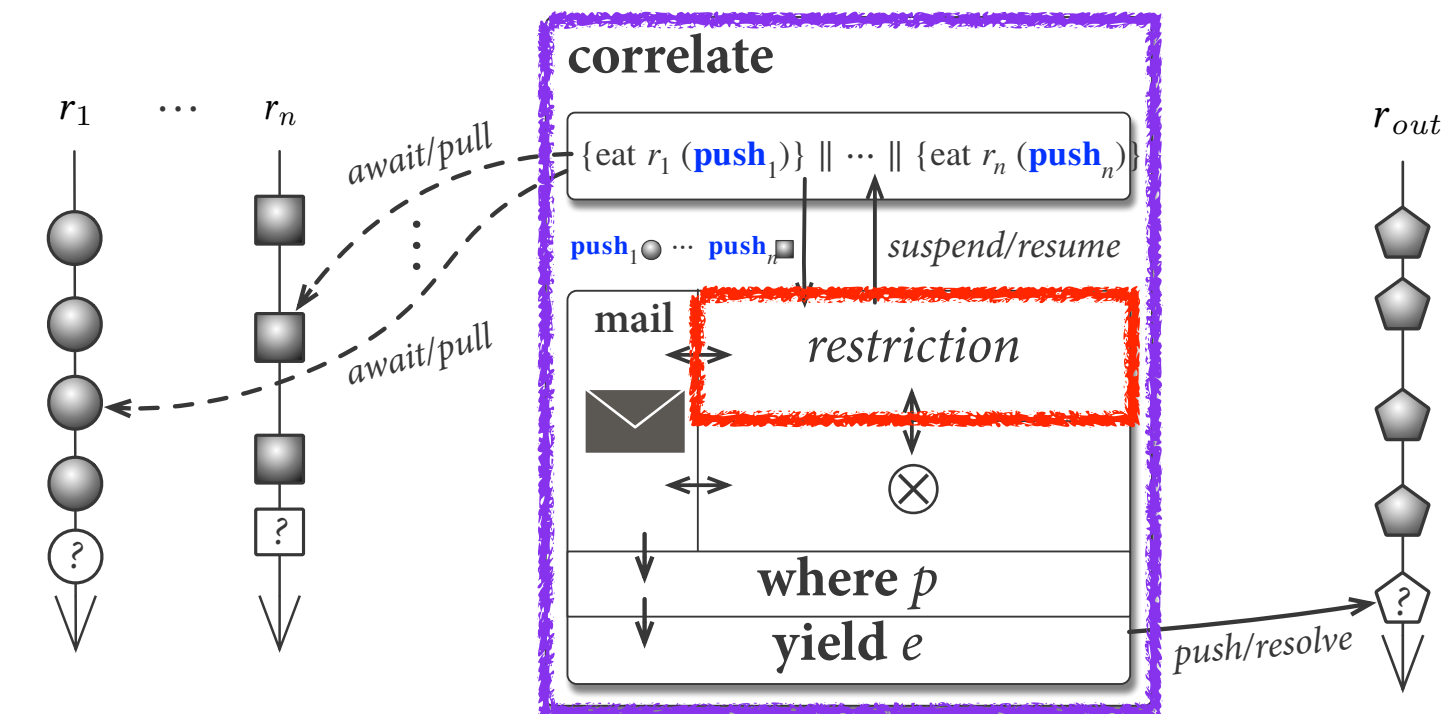
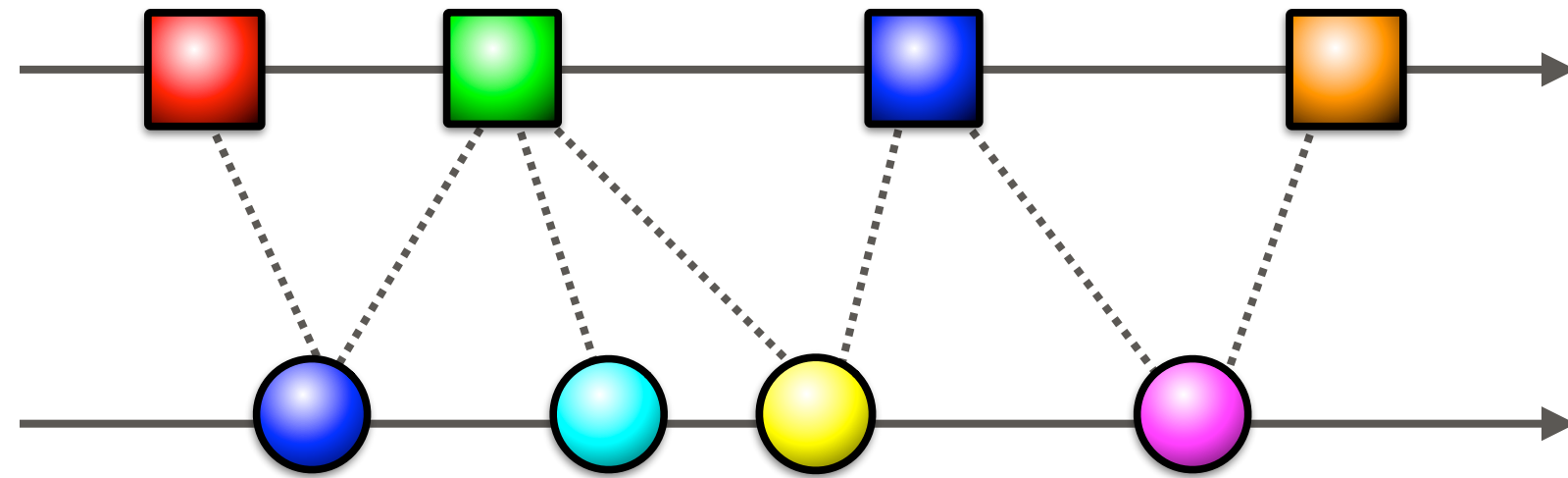


```

let mostRecentlyi = handler
| pushi ev resume ->
    seti [ev];
    resume (pushi ev)
  
```

RESTRICTION HANDLERS

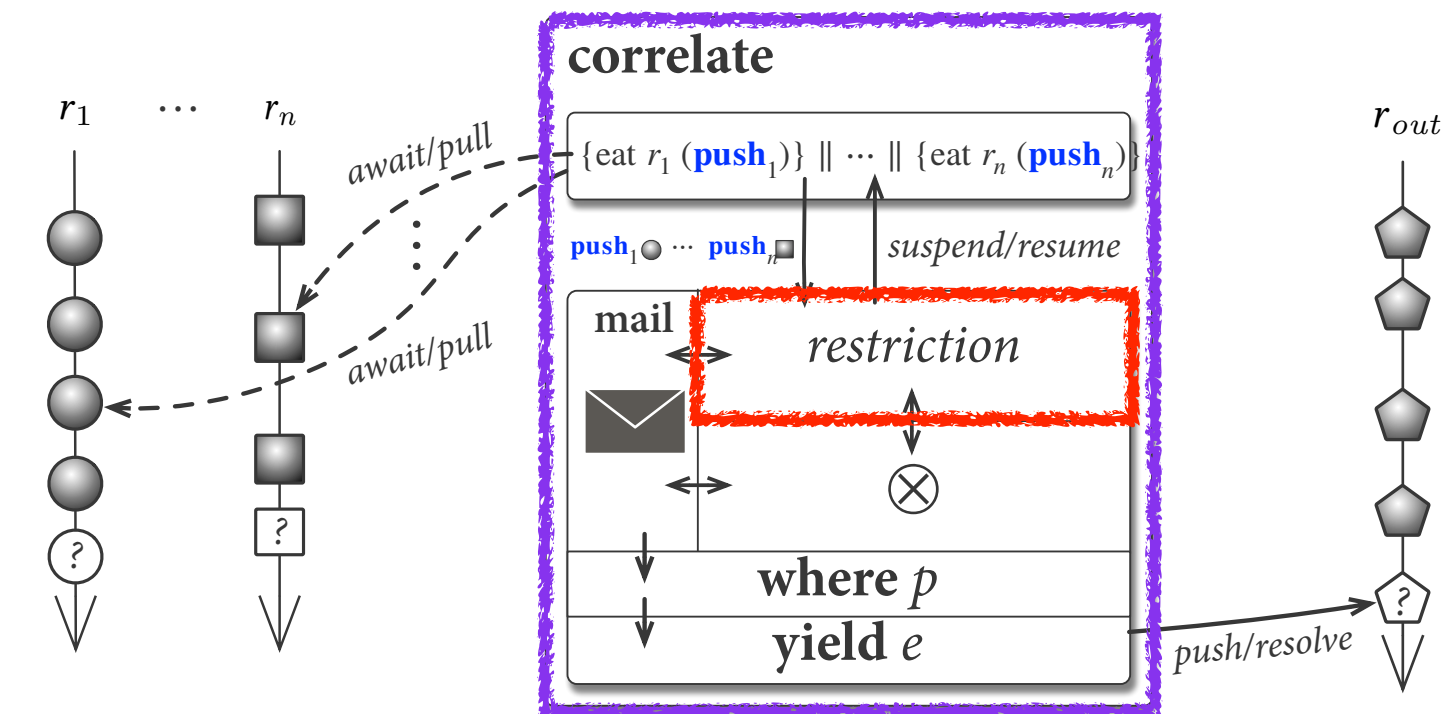
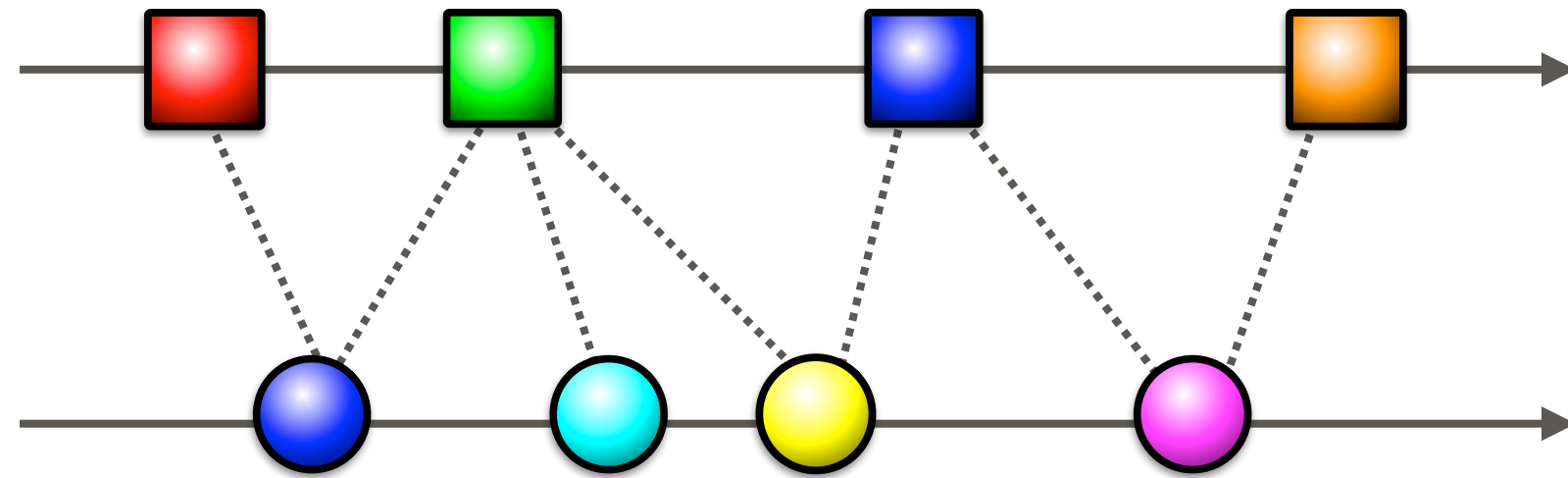
SPECIALISE THE NAIVE CARTESIAN PRODUCT



```
let mostRecentlyi = handler
| pushi ev resume ->
  seti [ev];
  resume (pushi ev)
```

RESTRICTION HANDLERS

SPECIALISE THE NAIVE CARTESIAN PRODUCT



```

let mostRecentlyi = handler
| pushi ev resume ->
  seti [ev];
  resume (pushi ev)
  
```


FROM THEORY TO PRACTICE

IMPLEMENTATION CHALLENGE: HOW CAN WE CORRELATE ANY NUMBER OF EVENT SOURCES?

FROM THEORY TO PRACTICE

IMPLEMENTATION CHALLENGE: HOW CAN WE CORRELATE ANY NUMBER OF EVENT SOURCES?

Declarative Pattern syntax

```
correlate {  
  with h  
  x1 from r1 ... xn from rn  
  where p  
  yield e }
```

FROM THEORY TO PRACTICE

IMPLEMENTATION CHALLENGE: HOW CAN WE CORRELATE ANY NUMBER OF EVENT SOURCES?

Declarative Pattern syntax

```
correlate {  
  with h  
  x1 from r1 ... xn from rn  
  where p  
  yield e }
```

$n \in \mathbb{N}$

FROM THEORY TO PRACTICE

IMPLEMENTATION CHALLENGE: HOW CAN WE CORRELATE ANY NUMBER OF EVENT SOURCES?

Declarative Pattern syntax

```
correlate {  
  with h  
  x1 from r1 ... xn from rn  
  where p  
  yield e }
```

$n \in \mathbb{N}$

Vs. Comprehensions? LINQ [Meijer et al. '06]?

```
from x1 in r1 ... from xn in rn  
where p  
select e
```


FROM THEORY TO PRACTICE

IMPLEMENTATION CHALLENGE: HOW CAN WE CORRELATE ANY NUMBER OF EVENT SOURCES?

Declarative Pattern syntax

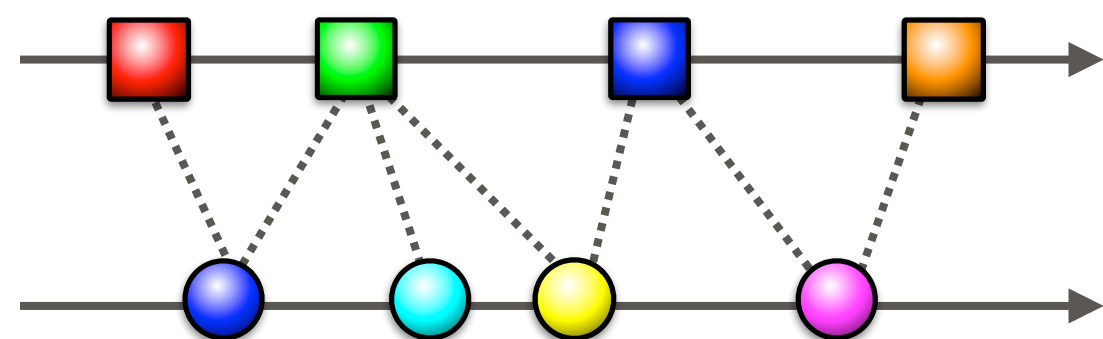
```
correlate {  
  with h  
  x1 from r1 ... xn from rn  
  where p  
  yield e }
```

$n \in \mathbb{N}$

Vs. Comprehensions? LINQ [Meijer et al. '06]?

```
from x1 in r1 ... from xn in rn  
where p  
select e
```

Cannot express dataflow!



FROM THEORY TO PRACTICE

IMPLEMENTATION CHALLENGE: HOW CAN WE CORRELATE ANY NUMBER OF EVENT SOURCES?

Declarative Pattern syntax

```
correlate {  
  with h  
  x1 from r1 ... xn from rn  
  where p  
  yield e }  
n ∈ ℕ
```

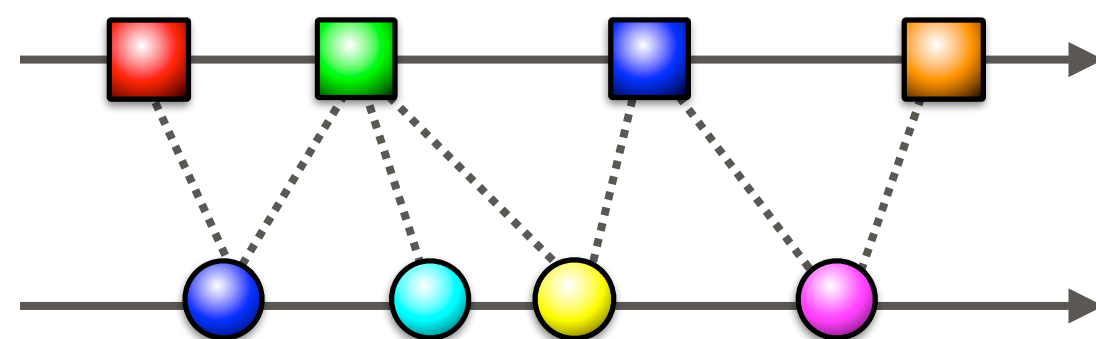
Generative effect declarations...

```
effect pushi : Event[Ai] -> Unit  
i ∈ {1, ..., n}
```

Vs. Comprehensions? LINQ [Meijer et al. '06]?

```
from x1 in r1 ... from xn in rn  
where p  
select e
```

Cannot express dataflow!



FROM THEORY TO PRACTICE

IMPLEMENTATION CHALLENGE: HOW CAN WE CORRELATE ANY NUMBER OF EVENT SOURCES?

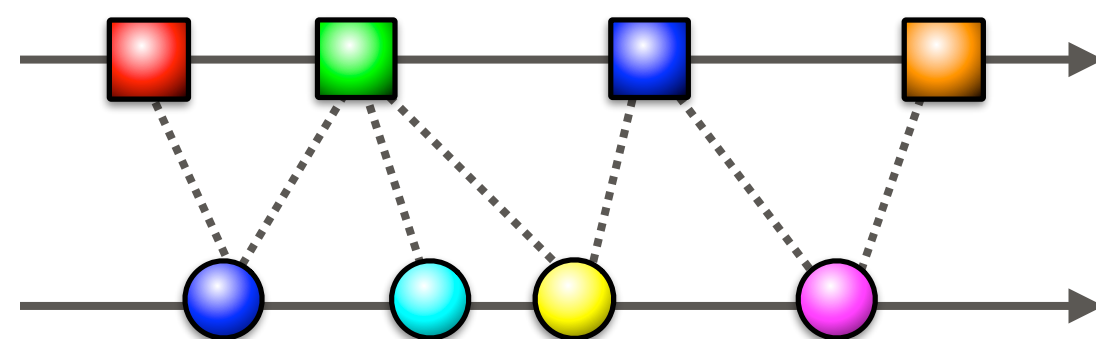
Declarative Pattern syntax

```
correlate {  
  with h  
  x1 from r1 ... xn from rn  
  where p  
  yield e }  
n ∈ ℕ
```

Vs. Comprehensions? LINQ [Meijer et al. '06]?

```
from x1 in r1 ... from xn in rn  
where p  
select e
```

Cannot express dataflow!



Generative effect declarations...

```
effect pushi : Event[Ai] -> Unit  
i ∈ {1, ..., n}
```

...and their handlers...

```
let mostRecentlyi = handler  
| pushi ev resume ->  
  seti [ev];  
  resume (pushi ev)  
i ∈ {1, ..., n}
```

FROM THEORY TO PRACTICE

IMPLEMENTATION CHALLENGE: HOW CAN WE CORRELATE ANY NUMBER OF EVENT SOURCES?

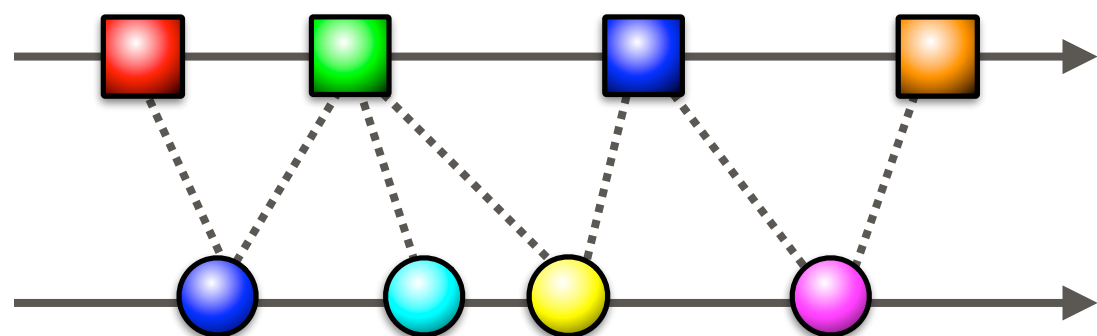
Declarative Pattern syntax

```
correlate {  
  with h  
  x1 from r1 ... xn from rn  
  where p  
  yield e }  
n ∈ ℕ
```

Vs. Comprehensions? LINQ [Meijer et al. '06]?

```
from x1 in r1 ... from xn in rn  
where p  
select e
```

Cannot express dataflow!



Generative effect declarations...

```
effect pushi : Event[Ai] -> Unit  
i ∈ {1, ..., n}
```

...and their handlers...

```
let mostRecentlyi = handler  
| pushi ev resume ->  
  seti [ev];  
  resume (pushi ev)  
i ∈ {1, ..., n}
```

Informal, lack linguistic support!

II

EXTENSIBLE PATTERN SYNTAX

POLYJOIN: PL EMBEDDING OF CORRELATION PATTERNS

A BETTER LINQ, WITHOUT COMPILER EXTENSIONS [BRACEVAC ET AL. 2020]

```
correlate ((from temp_sensor) @. (from smoke_sensor))  
    (fun ((temp, t1), ((smoke, t2), ())) ->  
        where smoke  
            && (temp >= 50.0)  
        (yield (format "Fire %f" temp)))
```

POLYJOIN: PL EMBEDDING OF CORRELATION PATTERNS

A BETTER LINQ, WITHOUT COMPILER EXTENSIONS [BRACEVAC ET AL. 2020]

```
correlate ((from temp_sensor) @. (from smoke_sensor))  
  (fun ((temp, t1), ((smoke, t2), ()))) ->  
    where (within t1 t2 (minutes 5.0))  
      && smoke  
      && (temp >= 50.0)  
    (yield (format "Fire %f" temp)))
```

POLYJOIN: PL EMBEDDING OF CORRELATION PATTERNS

A BETTER LINQ, WITHOUT COMPILER EXTENSIONS [BRACEVAC ET AL. 2020]

```
correlate (from temp_sensor)  
  (fun ((temp, t1), ((smoke, t2), ())) ->  
    where (within t1 t2 (minutes 5.0))  
          && smoke  
          && (temp >= 50.0)  
    (yield (format "Fire %f" temp)))
```


POLYJOIN: PL EMBEDDING OF CORRELATION PATTERNS

A BETTER LINQ, WITHOUT COMPILER EXTENSIONS [BRACEVAC ET AL. 2020]

```
correlate ((from temp_sensor) @. (from smoke_sensor))  
  (fun ((temp,t1), ((smoke,t2), ())) ->  
    where (within t1 t2 (minutes 5.0))  
      && smoke  
      && (temp >= 50.0)  
    (yield (format "Fire %f" temp)))
```

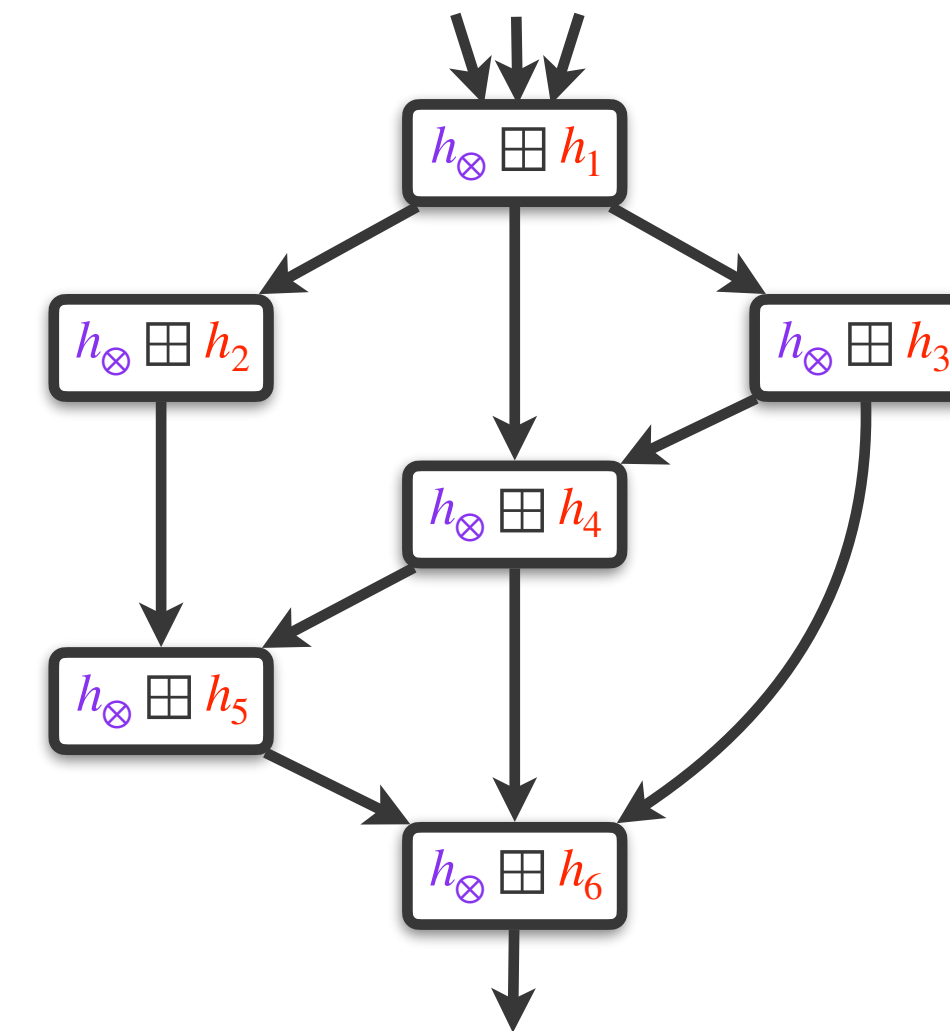
POLYJOIN: PL EMBEDDING OF CORRELATION PATTERNS

A BETTER LINQ, WITHOUT COMPILER EXTENSIONS [BRACEVAC ET AL. 2020]

```
correlate ((from temp_sensor) @. (from smoke_sensor))  
  (fun ((temp,t1), ((smoke,t2), ())) ->  
    where (within t1 t2 (minutes 5.0))  
      && smoke  
      && (temp >= 50.0)  
    (yield (format "Fire %f" temp)))
```

Portable!

 **Scala**



 **EsperTech**

...

POLYVARIADICITY

ARITY GENERICITY + HETEROGENEITY

Pattern Variables in Higher-Order Abstract Syntax (HOAS)

```
correlate: ('a, 'b) ctx -> ('b -> 'c pat) -> 'c shape repr
```

End-user query determines the type-level instantiations, i.e.,

```
correlate ((from temp_sensor) @. (from smoke_sensor))
```

Compiler calculates number and types of variables automatically:

```
((float repr * meta repr) * (bool repr * meta repr) -> 'c pat) -> 'c shape repr
```

POLYVARIADICITY

ARITY GENERICITY + HETEROGENEITY

Pattern Variables in Higher-Order Abstract Syntax (HOAS)

```
correlate: ('a, 'b) ctx -> ('b -> 'c pat) -> 'c shape repr
```

Indexing by type vectors (contexts): $'a \equiv \vec{A} \in \text{Type}^*$

End-user query determines the type-level instantiations, i.e.,

```
correlate ((from temp_sensor) @. (from smoke_sensor))
```

Compiler calculates number and types of variables automatically:

```
((float repr * meta repr) * (bool repr * meta repr) -> 'c pat) -> 'c shape repr
```


POLYVARIADIC EFFECTS AND HANDLERS

VIA INDEXING BY CONTEXT

```
correlate: ( 'a, 'b) ctx -> ( 'b -> 'c pat) -> 'c shape repr
```

POLYVARIADIC EFFECTS AND HANDLERS

VIA INDEXING BY CONTEXT

```
correlate: ('a, 'b) ctx -> ('b -> 'c pat) -> 'c shape repr
```

```
let mostRecentlyi = handler  
  | pushi ev resume ->  
    seti [ev];  
    resume (pushi ev)
```

POLYVARIADIC EFFECTS AND HANDLERS

VIA INDEXING BY CONTEXT

```
correlate: ('a, 'b) ctx -> ('b -> 'c pat) -> 'c shape repr
```

```
let mostRecently: type a i. (i, a) ptr -> a ext  
  fun ptr ctx ->  
    let module I = project (ptr ()) ctx in  
    handler  
      | I.push ev resume ->  
        I.set [ev];  
        resume (I.push ev)
```

POLYVARIADIC EFFECTS AND HANDLERS

VIA INDEXING BY CONTEXT

```
correlate: ('a, 'b) ctx -> ('b -> 'c pat) -> 'c shape repr
```

Context Polymorphism

```
let mostRecently: type a i. (i, a) ptr -> a ext
```

```
fun ptr ctx ->
```

```
let module I = project (ptr ()) ctx in
```

```
handler
```

```
| I.push ev resume ->
```

```
  I.set [ev];
```

```
  resume (I.push ev)
```

POLYVARIADIC EFFECTS AND HANDLERS

VIA INDEXING BY CONTEXT

```
correlate: ('a, 'b) ctx -> ('b -> 'c pat) -> 'c shape repr
```

```
let mostRecently: type a i. (i, a) ptr -> a ext  
  fun ptr ctx ->  
    let module I = project (ptr ()) ctx in  
    handler  
      | I.push ev resume ->  
        I.set [ev];  
        resume (I.push ev)
```

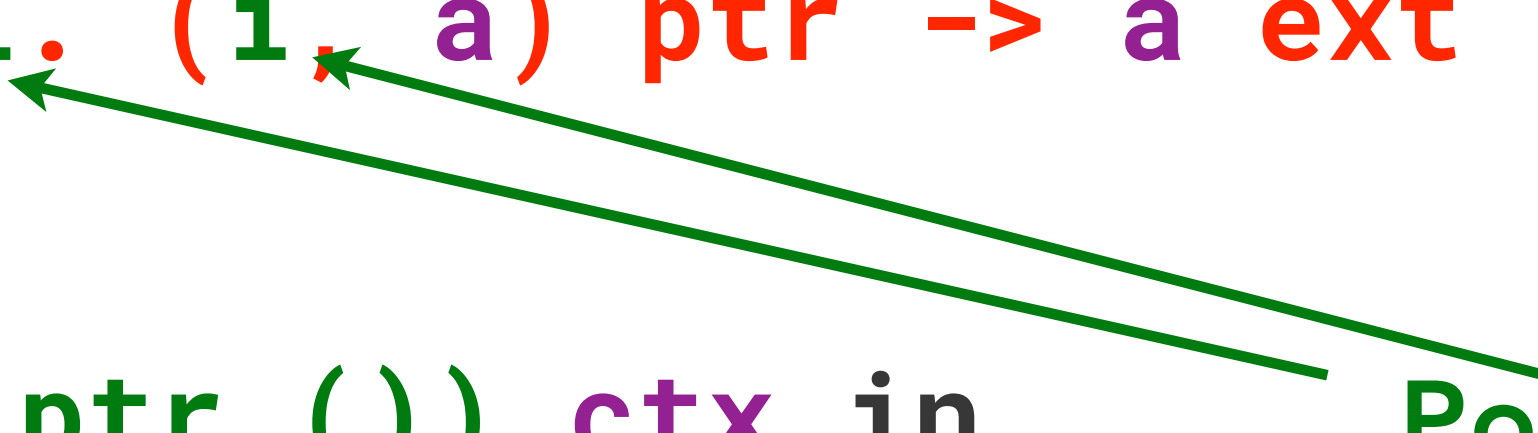

POLYVARIADIC EFFECTS AND HANDLERS

VIA INDEXING BY CONTEXT

```
correlate: ('a, 'b) ctx -> ('b -> 'c pat) -> 'c shape repr
```

```
let mostRecently: type a i. (i. a) ptr -> a ext  
  fun ptr ctx ->  
    let module I = project (ptr ()) ctx in  
    handler  
      | I.push ev resume ->  
        I.set [ev];  
        resume (I.push ev)
```

Position Polymorphism



POLYVARIADIC EFFECTS AND HANDLERS

VIA INDEXING BY CONTEXT

```
correlate: ('a, 'b) ctx -> ('b -> 'c pat) -> 'c shape repr
```

```
let mostRecently: type a i. (i, a) ptr -> a ext
  fun ptr ctx ->
    let module I = project (ptr ()) ctx in
    handler
    | I.push ev resume ->
      I.set [ev];
      resume (I.push ev)
```

POLYVARIADIC EFFECTS AND HANDLERS

VIA INDEXING BY CONTEXT

```
correlate: ('a, 'b) ctx -> ('b -> 'c pat) -> 'c shape repr
```

```
let mostRecently: type a i. (i, a) ptr -> a ext
fun ptr ctx ->
  let module I = project (ptr ()) ctx in
  handler
  | I.push ev resume ->
    I.set [ev];
    resume (I.push ev)
```

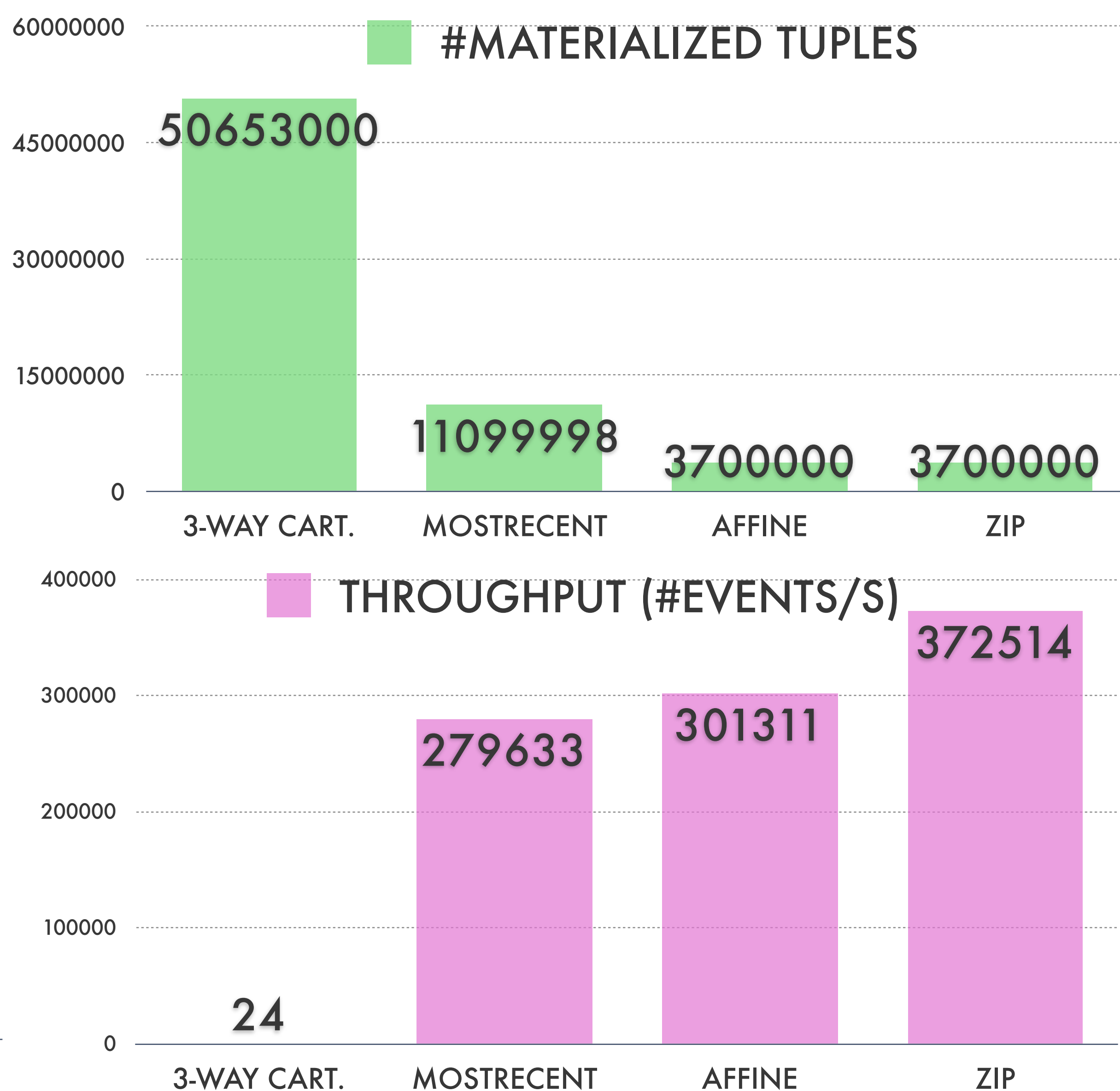
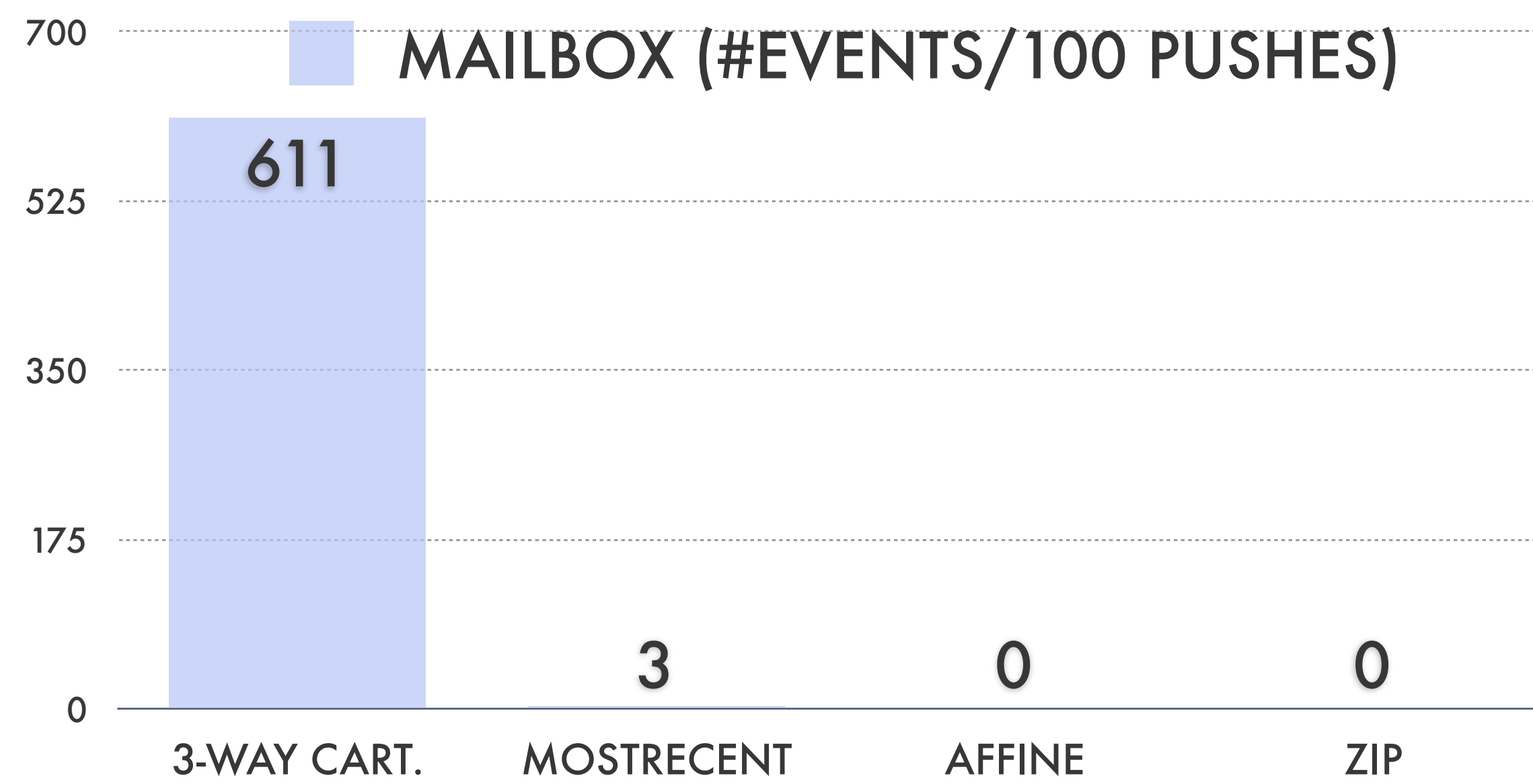
```
correlate ((from temp_sensor) @. (from smoke_sensor))
  (mostRecently 0) ⊞ (mostRecently 8)
  (fun ((temp, t1), ((smoke, t2), ())) ->
    where p
      (yield e))
```

III

EVALUATION

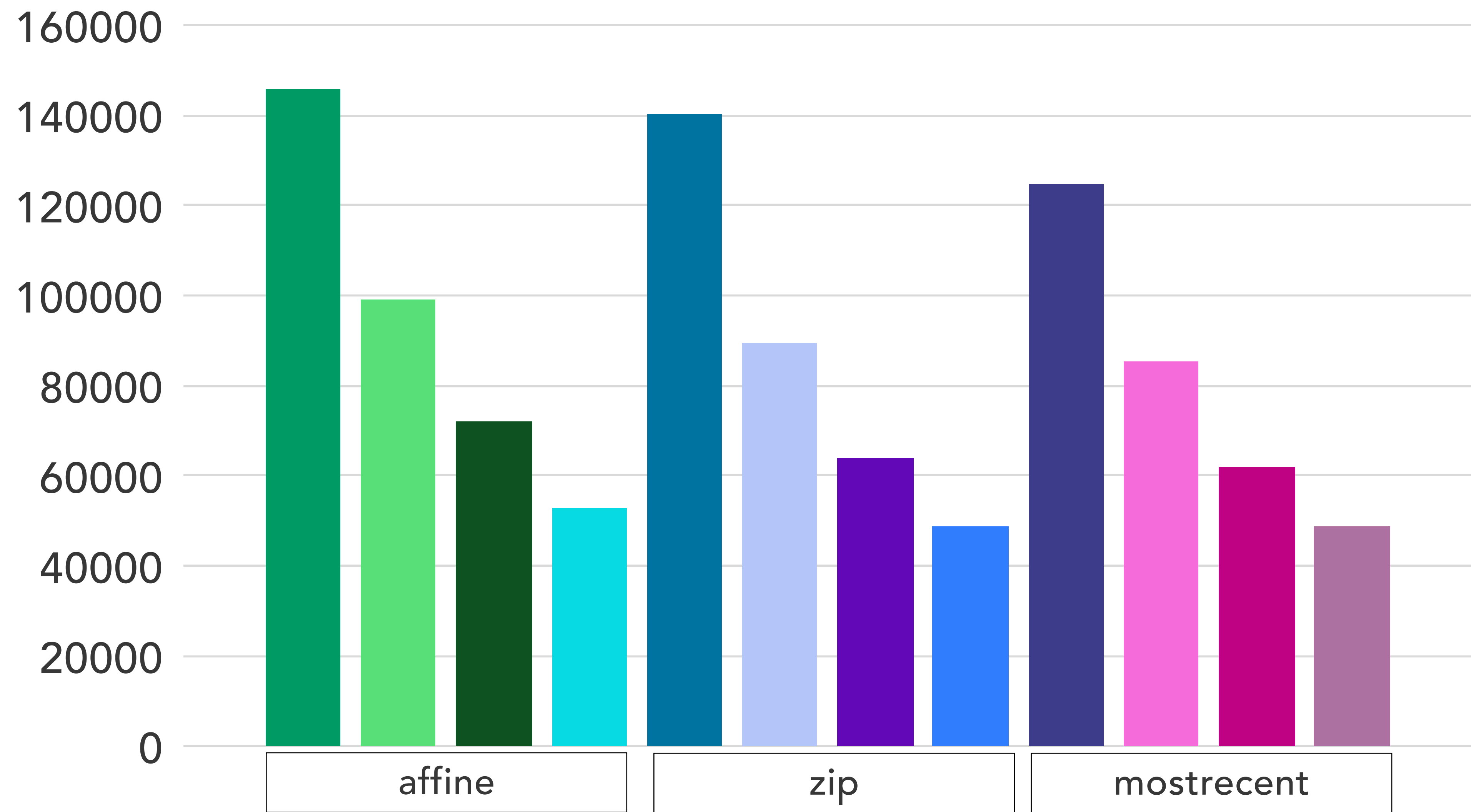
HOW “EFFECTIVE” ARE RESTRICTION HANDLERS?

- Experiment: Uniformly distribute 10^7 events over 3 push streams.
- Decomposition into naive cartesian product and restrictions is practical.



EFFECT OF ARITY ON PERFORMANCE

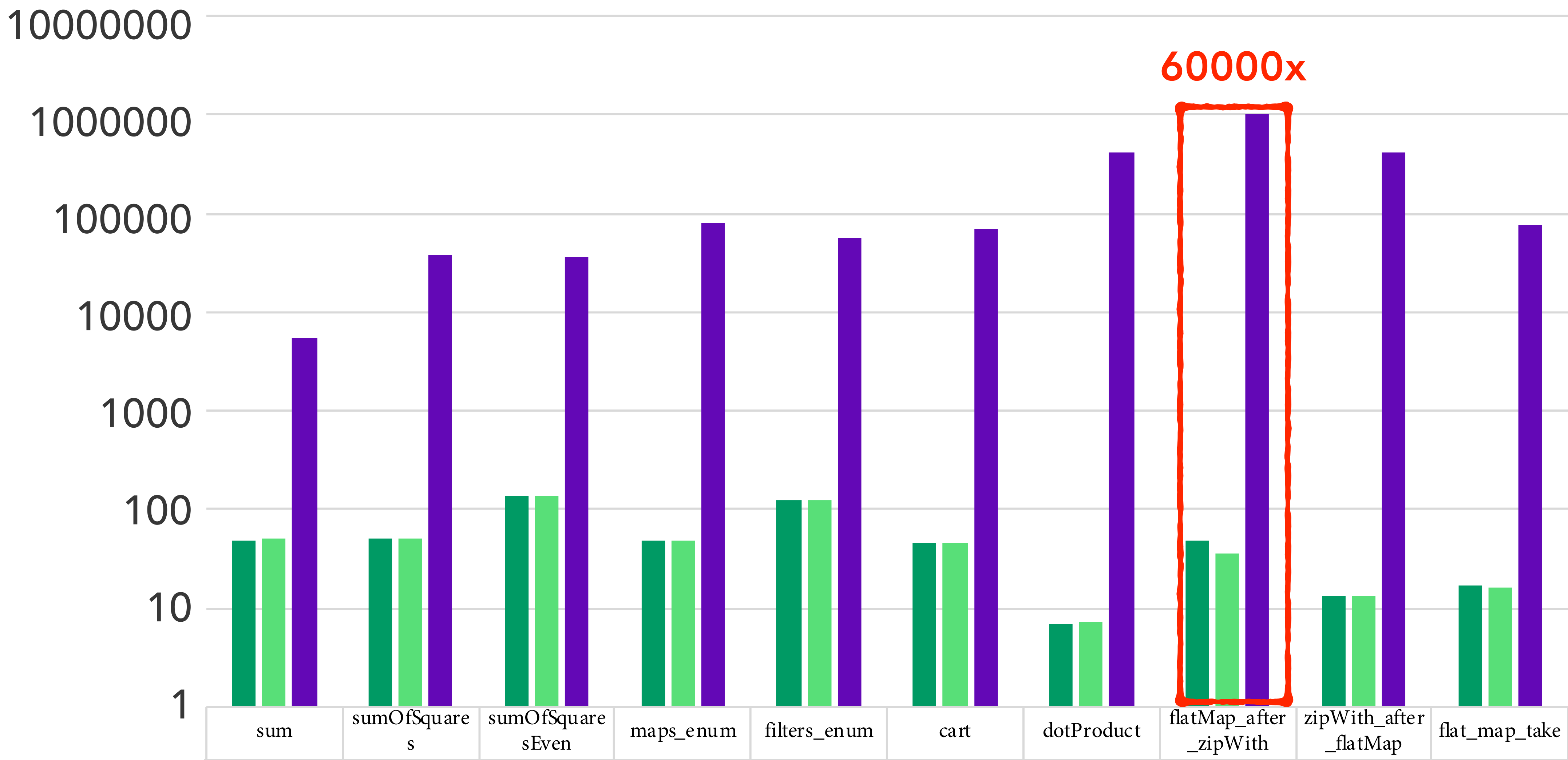
Total Throughput (#Events/sec), Even Distribution of 10^7 Events, Arity $n = 8, 12, 16, 20$



COMPARISON TO HAND-WRITTEN CODE

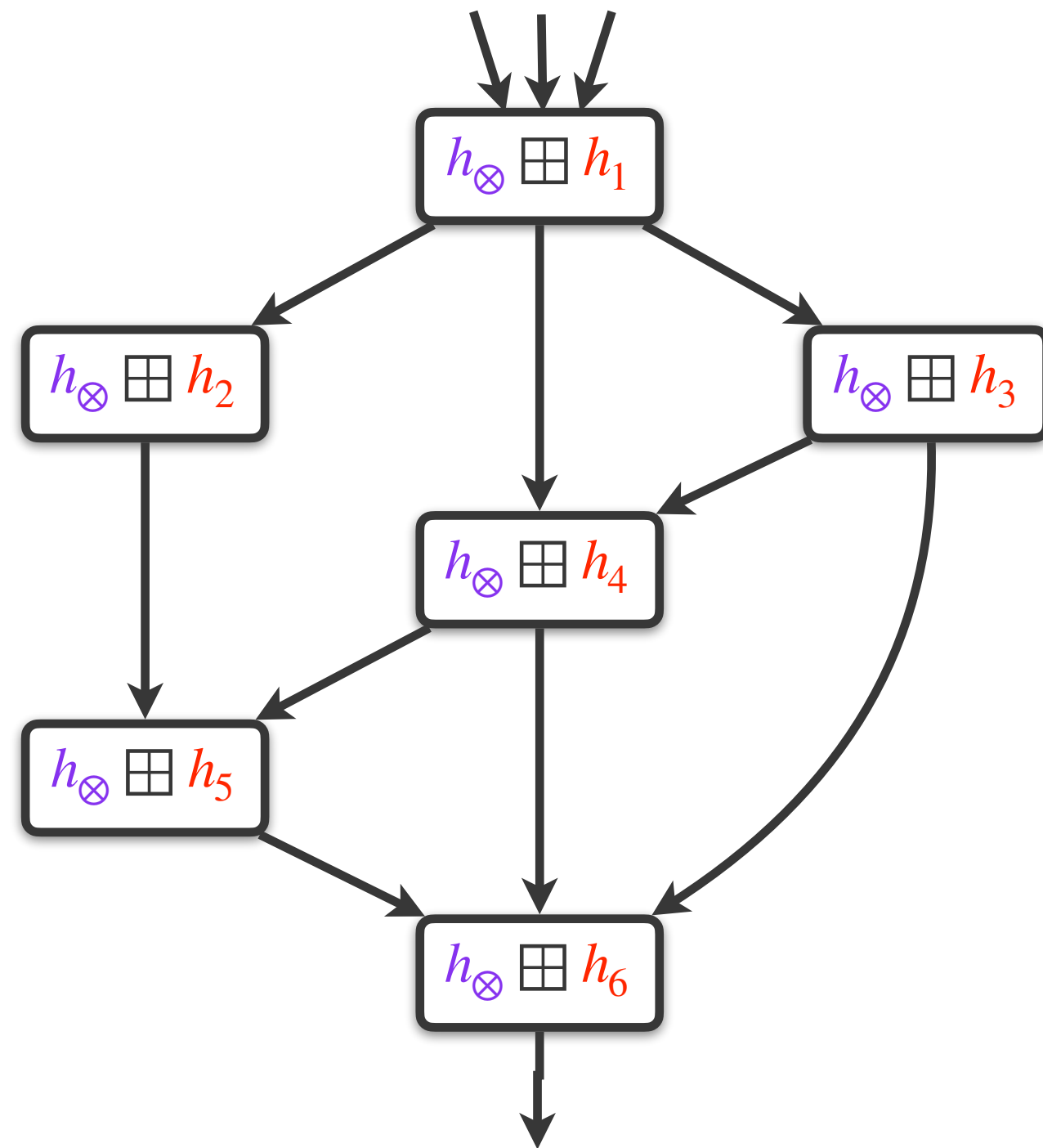
STRYMONAS LIBRARY [KISELYOV ET AL. 2017]

Execution Time (ms), log scale

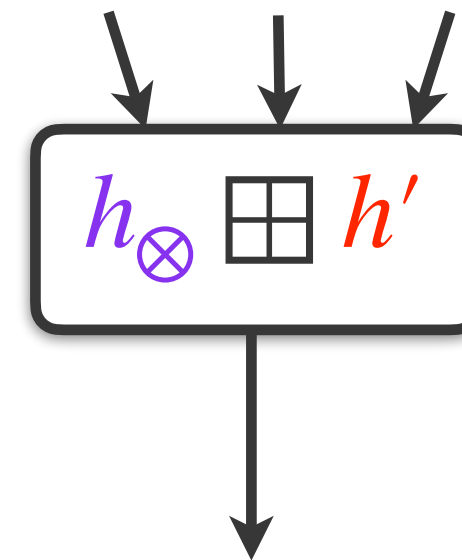


FUTURE WORK

Make it fast!



- Staging
- Deforestation
- Handler Fusion



Logical Interpretation

Propositions as Types, Proofs as Programs

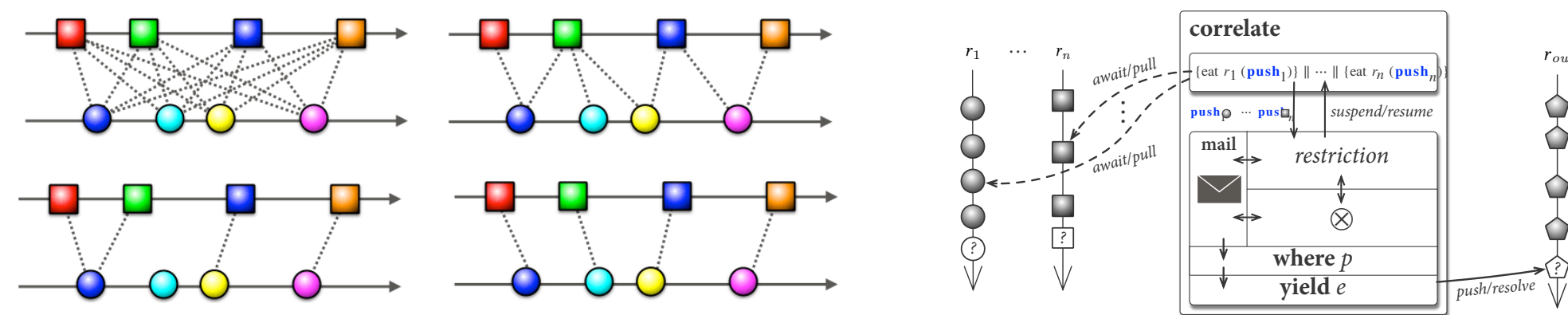
$$R[T] = \Box (T \times \Diamond [R[T]])$$

- Strong Normalization
- (Co)Induction
- Liveness & Productivity

MY CONTRIBUTIONS

AN À LA CARTE SYNTAX AND SEMANTICS FOR EVENT CORRELATION

CARTESIUS: SEMANTIC MODEL



- The **first** work that **uniformly** expresses join variants.
- Programming of **"interleaved coroutines"**.
- Effects make naïve enumerations **efficient**.

POLYJOIN: PORTABLE PL SYNTAX EMBEDDING

```
correlate ((from s1) @. ... @. (from sn))  
  (fun (x1, (... (xn, ()))...) ->  
    (where p  
      (yield e)))
```

- **A better LINQ:** Practical, portable, extensible, type-safe correlation patterns.
- Join syntax and semantics are **polyvariadic**.

PROCESSING MODEL

INTERLEAVING CONCURRENCY

interleave: $\forall \mu . \text{List}[\{\text{Unit}\}^{\langle \text{async}, \mu \rangle}] \rightarrow \langle \text{async}, \mu \rangle \text{Unit}$
[Leijen '17a]

$$e_1 \parallel \cdots \parallel e_n \quad \rightsquigarrow \quad \{\text{interleave } [e_1, \dots, e_n]\}$$

PROCESSING MODEL

INTERLEAVING CONCURRENCY

interleave: $\forall \mu . \text{List}[\{\text{Unit}\}^{\langle \text{async}, \mu \rangle}] \rightarrow \langle \text{async}, \mu \rangle \text{Unit}$
[Leijen '17a]

$$e_1 \parallel \cdots \parallel e_n \rightsquigarrow \{\text{interleave } [e_1, \dots, e_n]\}$$

The calling context observes effects of the strands!

$\{\text{await } x\} \parallel \{\text{println "foo"}\} \parallel \{\text{yield } 1\} : \{\text{Unit}\}^{\langle \text{yield}, \text{println}, \text{async} \rangle}$

THE BIG PICTURE

```
let  $r_{\text{out}}$  =  
  correlate {  
    with  $h$   
     $x_1$  from  $r_1$   
    ...  
     $x_n$  from  $r_n$   
    where  $p$   
    yield  $e$  }
```

THE BIG PICTURE

```
let  $r_{\text{out}}$  =  
  correlate {  
    with  $h$   
     $x_1$  from  $r_1$   
    ...  
     $x_n$  from  $r_n$   
    where  $p$   
    yield  $e$  }
```

let $r_{\text{out}} = \text{with } (h_{\otimes} \boxplus h)$
 $\{\text{eat } r_1 (\text{push}_1)\} \parallel \cdots \parallel \{\text{eat } r_n (\text{push}_n)\}$

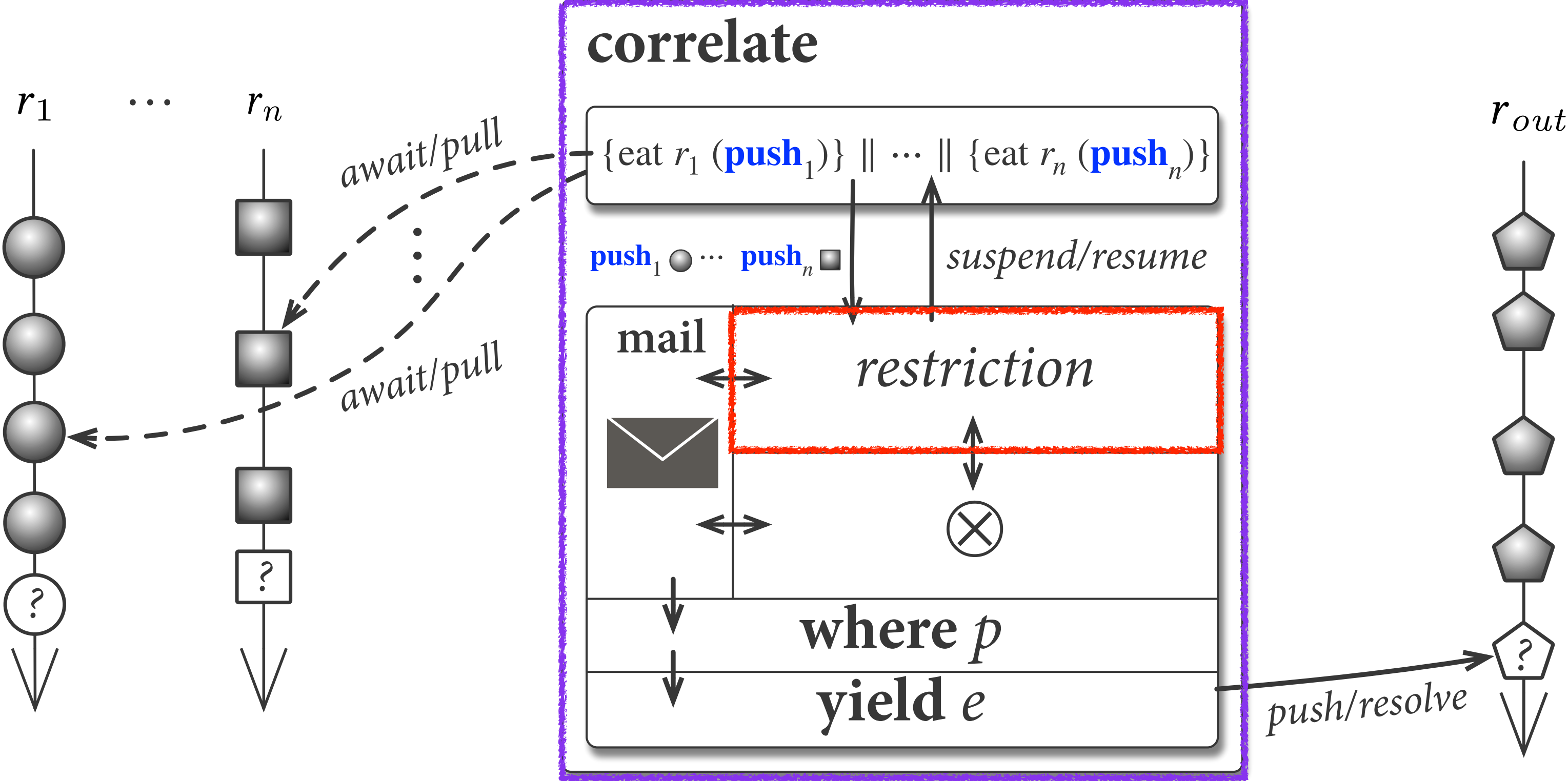
THE BIG PICTURE

```

let rout =
  correlate {
    with h
    x1 from r1
    ...
    xn from rn
    where p
    yield e }

let rout = with (h⊗ ⊞ h)
  {eat r1 (push1)} || ... || {eat rn (pushn)}

```

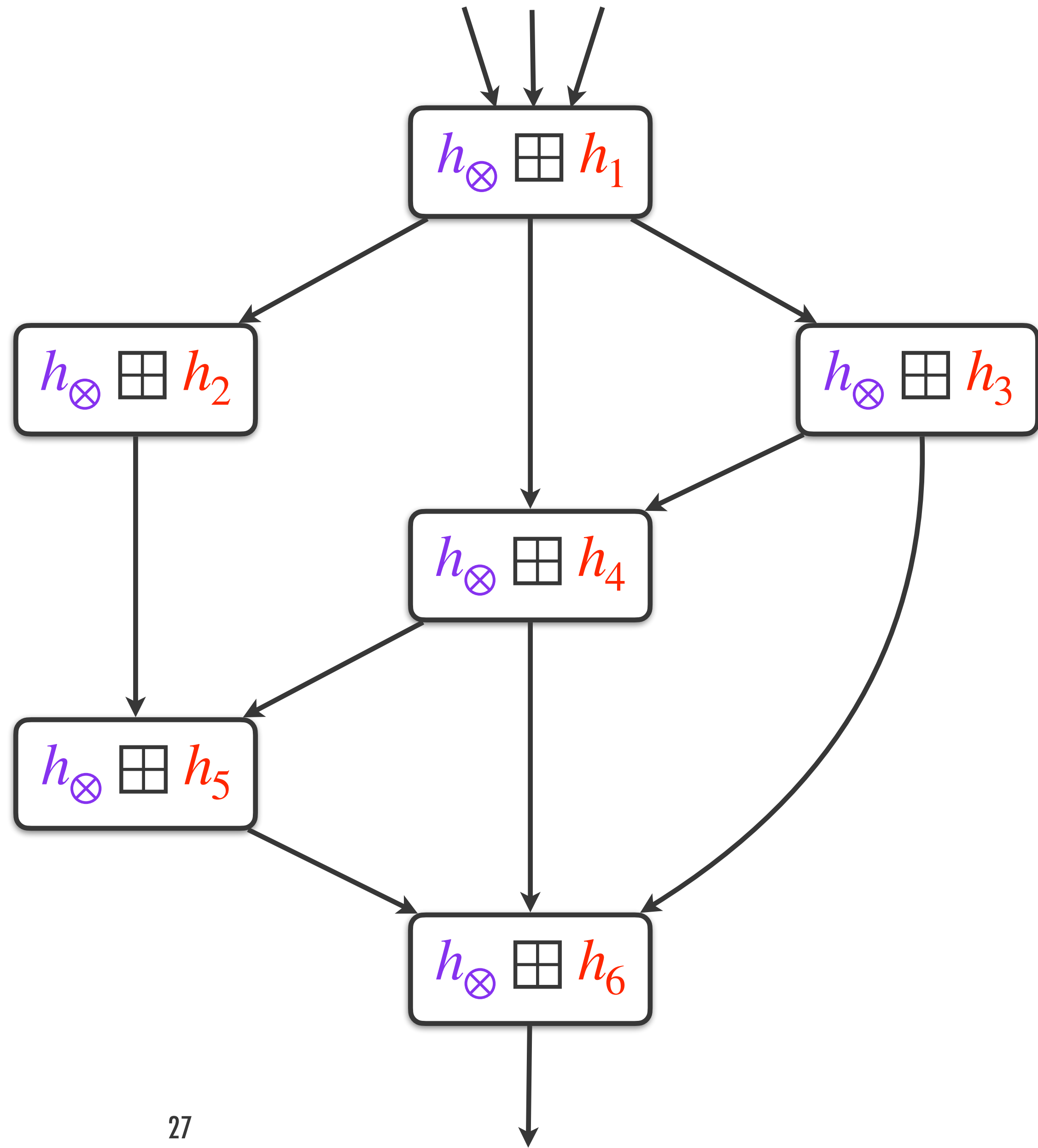
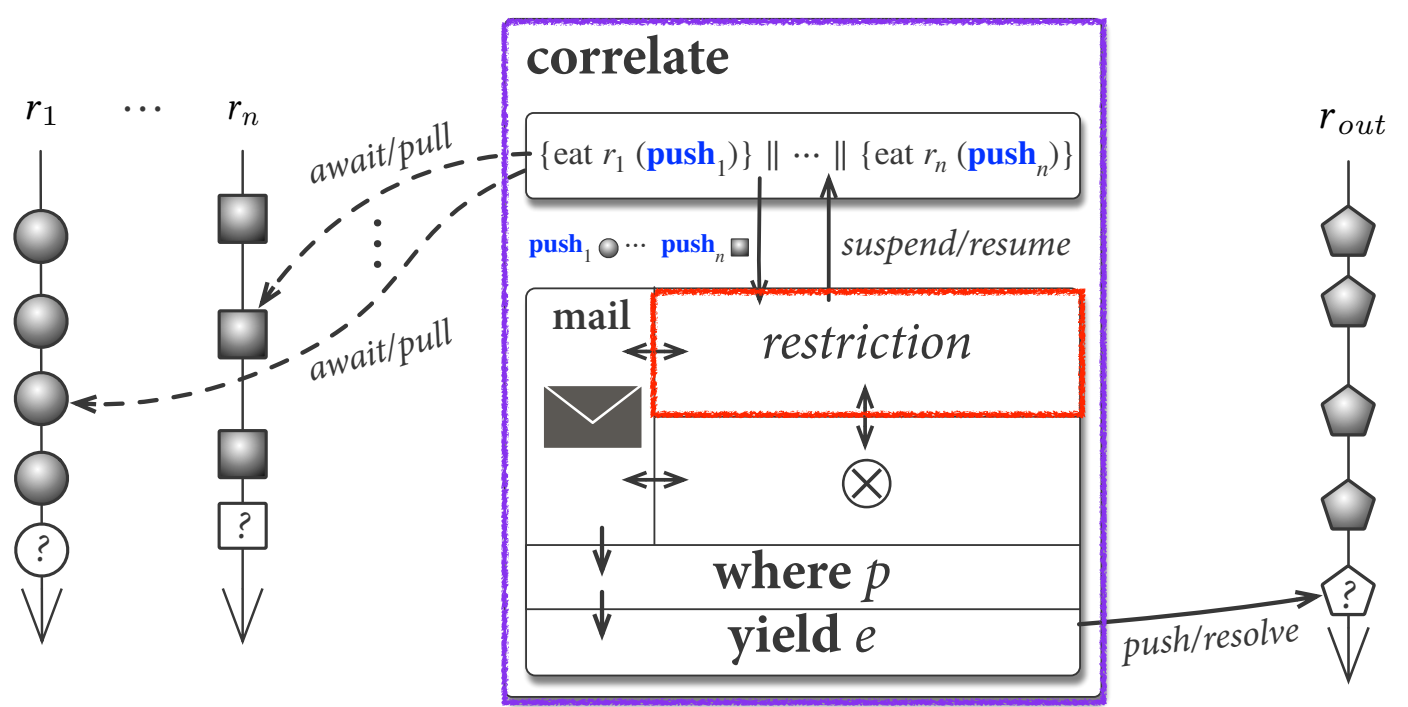


THE BIG PICTURE

```

let rout =
  correlate {
    with h
    x1 from r1
    ...
    xn from rn
    where p
    yield e }

let rout = with (h⊗ ⊞ h)
             {eat r1 (push1)} || ... || {eat rn (pushn)}
```

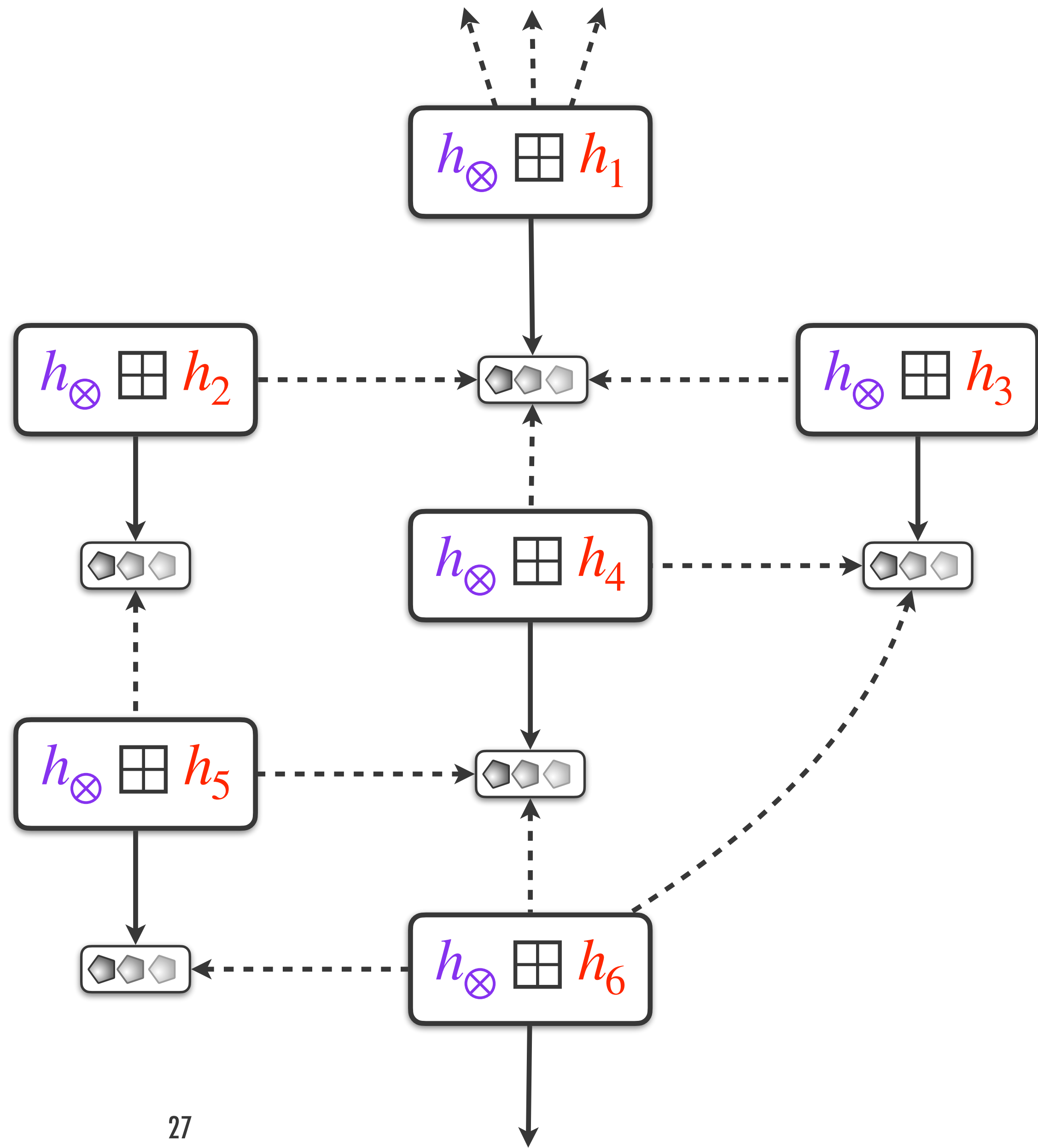
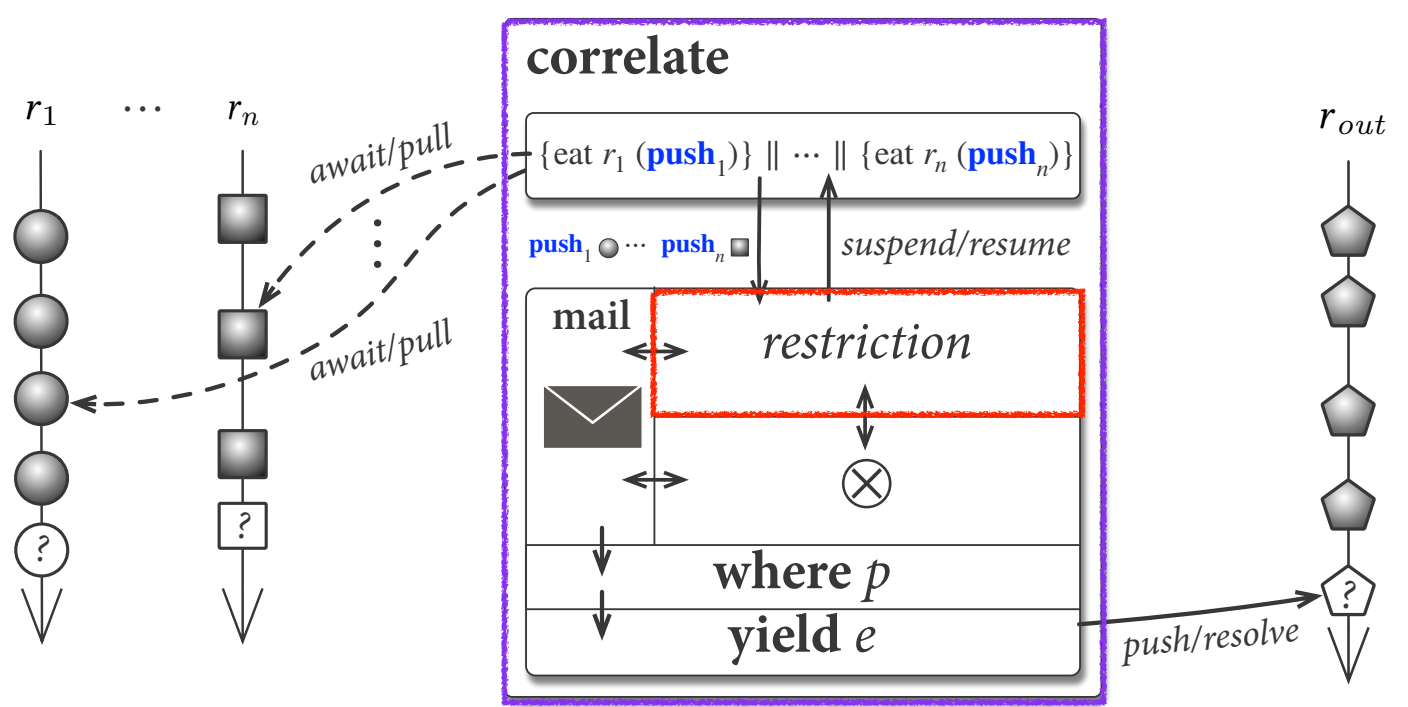


THE BIG PICTURE

```

let rout =
  correlate {
    with h
    x1 from r1
    ...
    xn from rn
    where p
    yield e }

let rout = with (h⊗ ⊞ h)
  {eat r1 (push1)} || ... || {eat rn (pushn)}
  
```



NESTING IS BAD!

Could define joins as

$$M[A_1] \rightarrow \dots \rightarrow M[A_n] \rightarrow M[A_1 \times \dots \times A_n]$$
$$\text{return} : A \rightarrow M[A]$$
$$\text{bind} : M[A] \rightarrow (A \rightarrow M[B]) \rightarrow M[B]$$

the basis for comprehensions [Wadler '92],
LINQ [Meijer '06]:

from x_1 in m_1

\vdots

from x_n in m_n

select $\langle x_1, \dots, x_n \rangle$


$$\text{bind } m_1 (\lambda x_1:A_1.$$
$$\text{bind } m_2 (\lambda x_2:A_2.$$
$$\dots$$
$$\text{bind } m_n (\lambda x_n:A_n. \text{return } \langle x_1, \dots, x_n \rangle) \dots))$$

- **Problem:** sequential binding is inadequate for asynchrony
- We are not in control of event sources
- Join should be able to proceed on any event, **need non-sequential binding**

POLYVARIADICITY

ARITY GENERICITY + HETEROGENEITY, INDEXING BY VARIABLE CONTEXT SHAPE

Pattern Variables in Higher-Order Abstract Syntax (HOAS)

```
correlate: ( 'a, 'b) ctx -> ( 'b -> 'c pat) -> 'c shape repr
```

Encodes an indexed function family

```
correlate:  $\left( \overrightarrow{\text{Shape}[A]} \rightarrow (\overrightarrow{A \times \text{Meta}} \rightarrow C \times \text{Meta}) \rightarrow \text{Shape}[C] \right)_{\overrightarrow{A} \in \text{Type}^*}$ 
```

Payoff: Permits variable binding semantics suitable for data flow patterns, which LINQ and monad comprehensions *cannot* express!

POLYJOIN: A BETTER LINQ

SUMMARY

TAGLESS INTERPRETERS [CARETTE ET AL. 2009]

- Modular, extensible abstract syntax & interface!
- Polyvariadic binding forms?

POLYVARIADIC FUNCTIONS/PATTERN COMBINATORS [DANVY 1998, RHIGER 2009]

- Separate interface and implementation?
- Polyvariadic binding forms!

POLYJOIN [BRACEVAC ET AL. 2020]

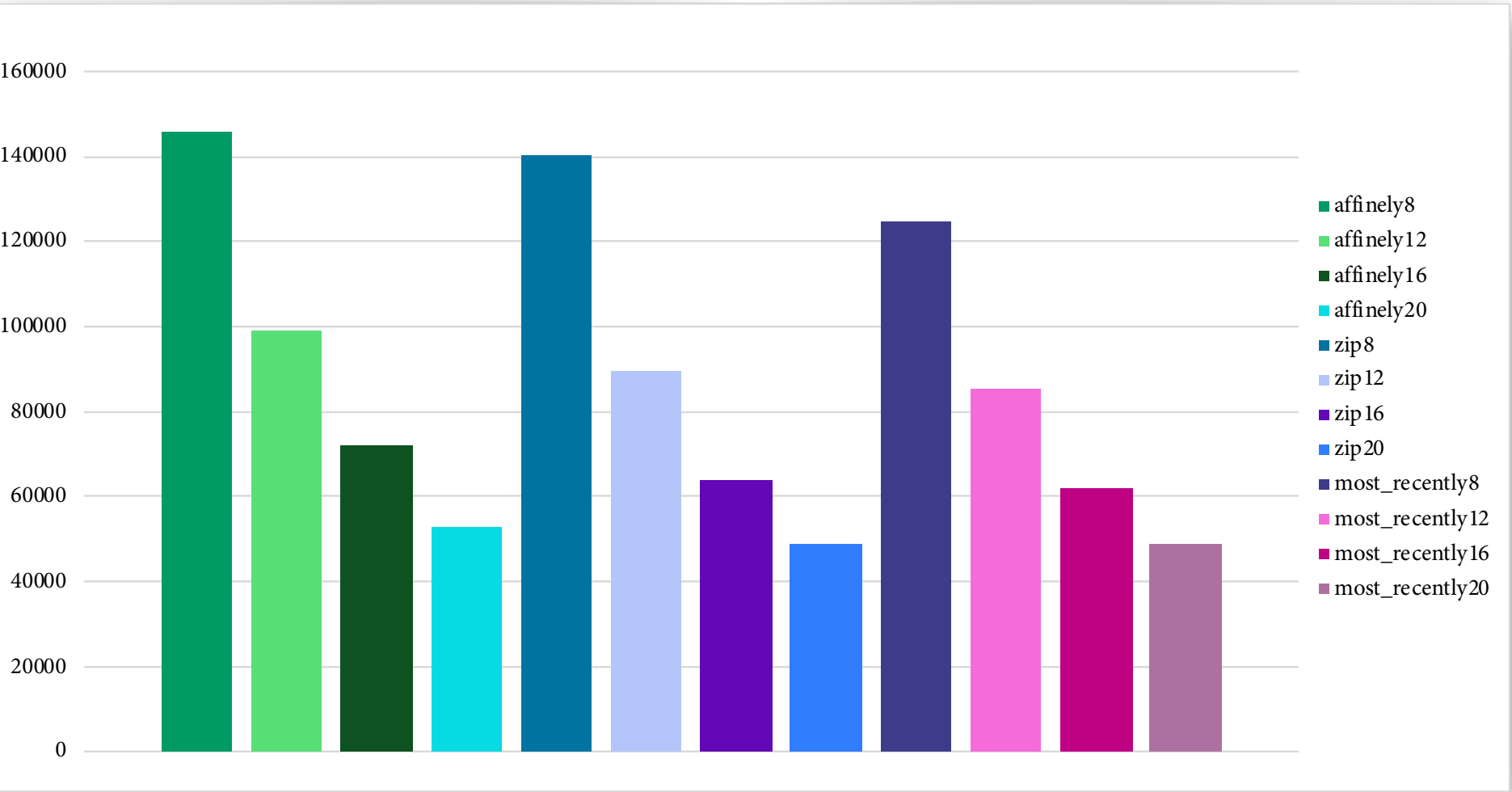
- Modular, extensible abstract syntax & interface!
- Polyvariadic binding forms!

BIBLIOGRAPHY

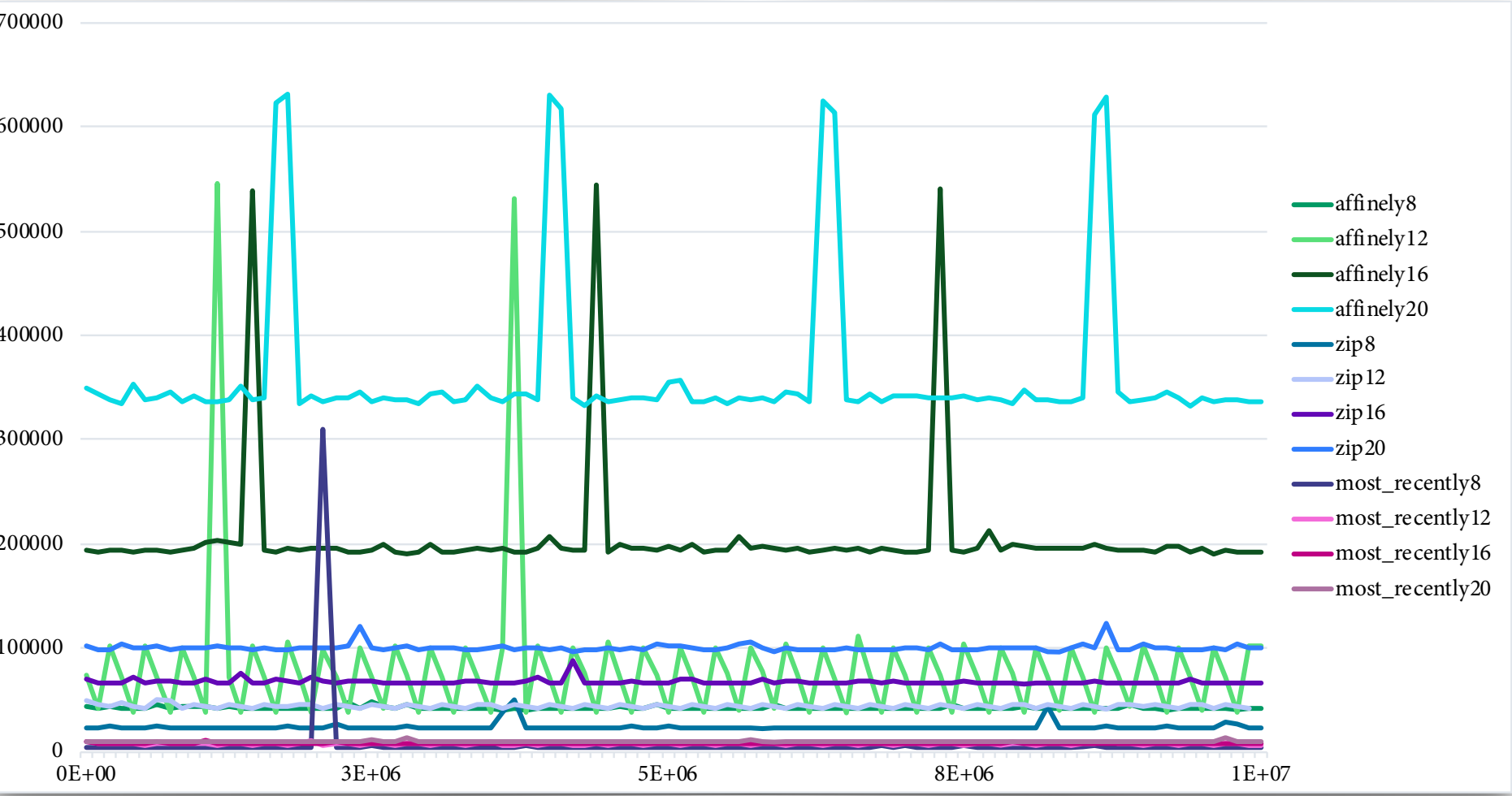
- Agrawal, J., Diao, Y., Gyllstrom, D., and Immerman, N. (2008). Efficient pattern matching over event streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.
- Arasu, A., Babu, S., and Widom, J. (2004). CQL: A language for continuous queries over streams and relations. In *Database Programming Languages*. Springer Berlin Heidelberg.
- Conchon, S. and Le Fessant, F. (1999). JoCaml: mobile agents for Objective-Caml. In *First and Third International Symposium on Agent Systems Applications, and Mobile Agents (ASAMA)*.
- Cugola, G. and Margara, A. (2012). Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Computing Surveys*, 44(3):15:1-15:62.
- Czaplicki, E. and Chong, S. (2013). Asynchronous Functional Reactive Programming for GUIs. In *Proceedings of Conference on Programming Language Design and Implementation (PLDI)*.
- Edwards, J. (2009). Coherent reaction. In *Proceedings of Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*.
- Fournet, C. and Gonthier, G. (1996). The reflexive CHAM and the Join-calculus. In *Proceedings of Symposium on Principles of Programming Languages (POPL)*.
- Leijen, D. (2017a). Structured asynchrony with algebraic effects. In *Proceedings of the International Workshop on Type-Driven Development (TyDe)*.
- Leijen, D. (2017b). Type directed compilation of row-typed algebraic effects. In *Proceedings of Symposium on Principles of Programming Languages (POPL)*.
- Lindley, S., McBride, C., and McLaughlin, C. (2017). Do be do be do. In *Proceedings of Symposium on Principles of Programming Languages (POPL)*.
- Meijer, E., Beckman, B., and Bierman, G. (2006). LINQ: Reconciling object, relations and XML in the .NET framework. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.
- Plotkin, G. D. and Power, J. (2003). Algebraic operations and generic effects. *Applied Categorical Structures*.
- Plotkin, G. D. and Pretnar, M. (2009). Handlers of algebraic effects. In *Proceedings of the European Conference on Programming Languages and Systems (ESOP)*.
- Wadler, P. (1992). Comprehending monads. *Mathematical Structures in Computer Science*, 2(4):461-493.

EFFECT OF ARITY ON PERFORMANCE

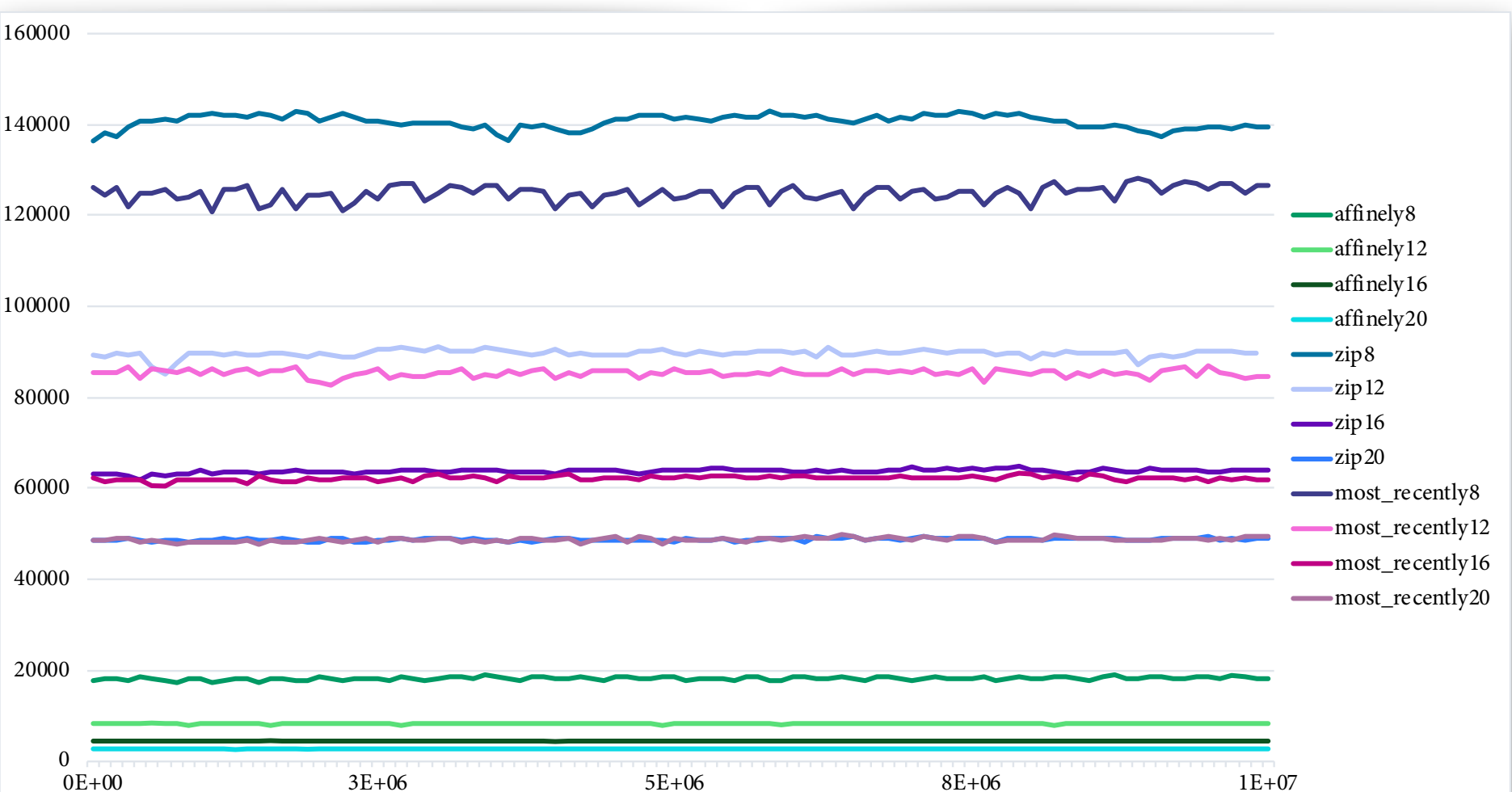
Total Throughput (#Events/sec)



Latency (ns)



Production Rate (#Events/sec)



- Experiment: Uniformly distribute 10^7 events over n inputs, where $n = 8, 12, 16, 20$.
- Throughput declines with n .
- Culprit: Effects Dispatch/Stacking of handlers.