香 港 大 學
THE UNIVERSITY OF HONG KONG

Algebraic Effects, Evidently

Ningning Xie The University of Hong Kong

Joint Work with Jonathan Brachthäuser, Daniel Hillerström, Philipp Schuster, Daan Leijen

Microsoft Research

Introduction

algebraic effects → evidence passing → polymorphic lambda calculus

a theoretical basis for languages such as Haskell and ML

Algebraic Effects

Composable and modular computational effects

Algebraic effects

define a family of operations

operation — effect

```
effect reader {
  ask : () -> int
}
```

Effect handlers

give semantics to operations

effect handler — implementation

```
handler {
  ask -> \x.\k. k 1
} (\_.
  perform ask () + perform ask () //2
)
```

raise an effect

Drawbacks

Runtime Support

Difficult to integrate into existing languages

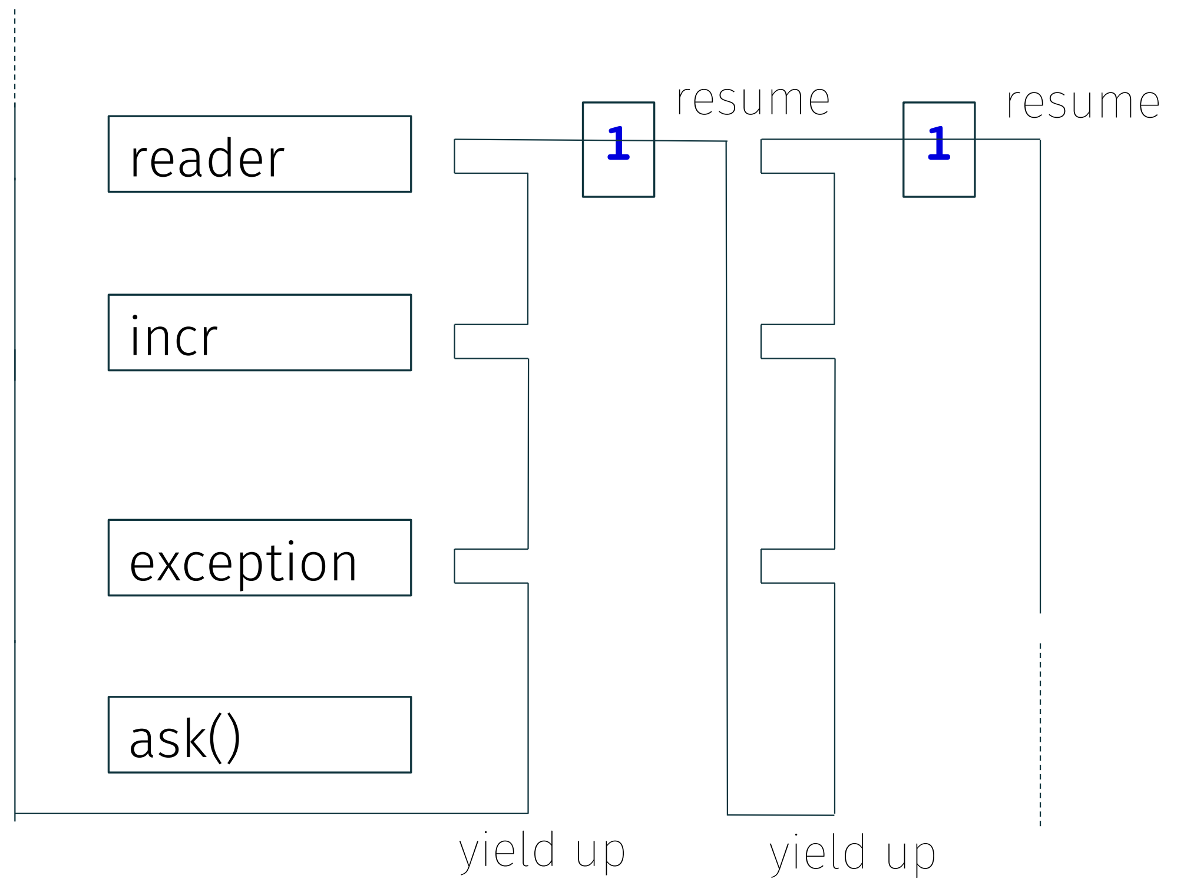
Inefficiency

Dynamically yield up to search for the handler

Evaluation Under Traditional Semantics

```
handler {
  ask -> \x.\k. k 1
} (\_.
handler {
  incr -> \x.\k. 1 + k ()
} (\_.
handler {
  fail -> \x.\k. 3
} (\_.
  perform ask () + perform ask () // 2
)))
```

tail-resumptive
(\x.\k. k e with k∉ fv(e))



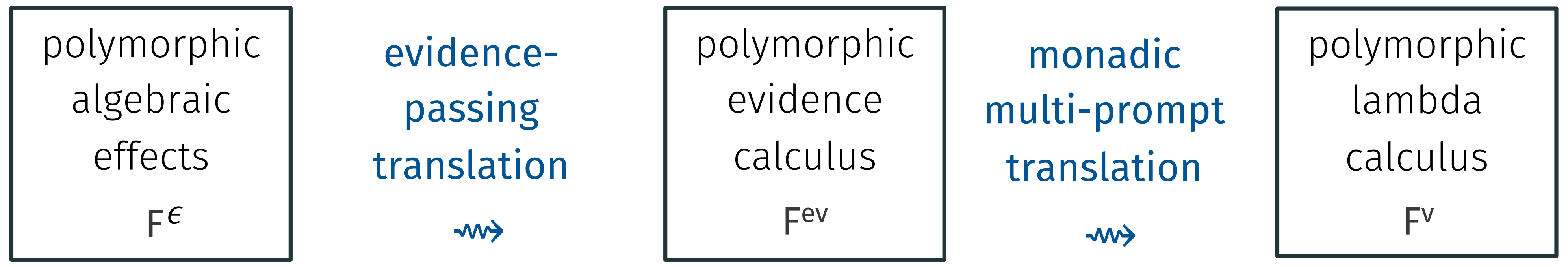
Evaluation Under Evidence Semantics

Q.E.D

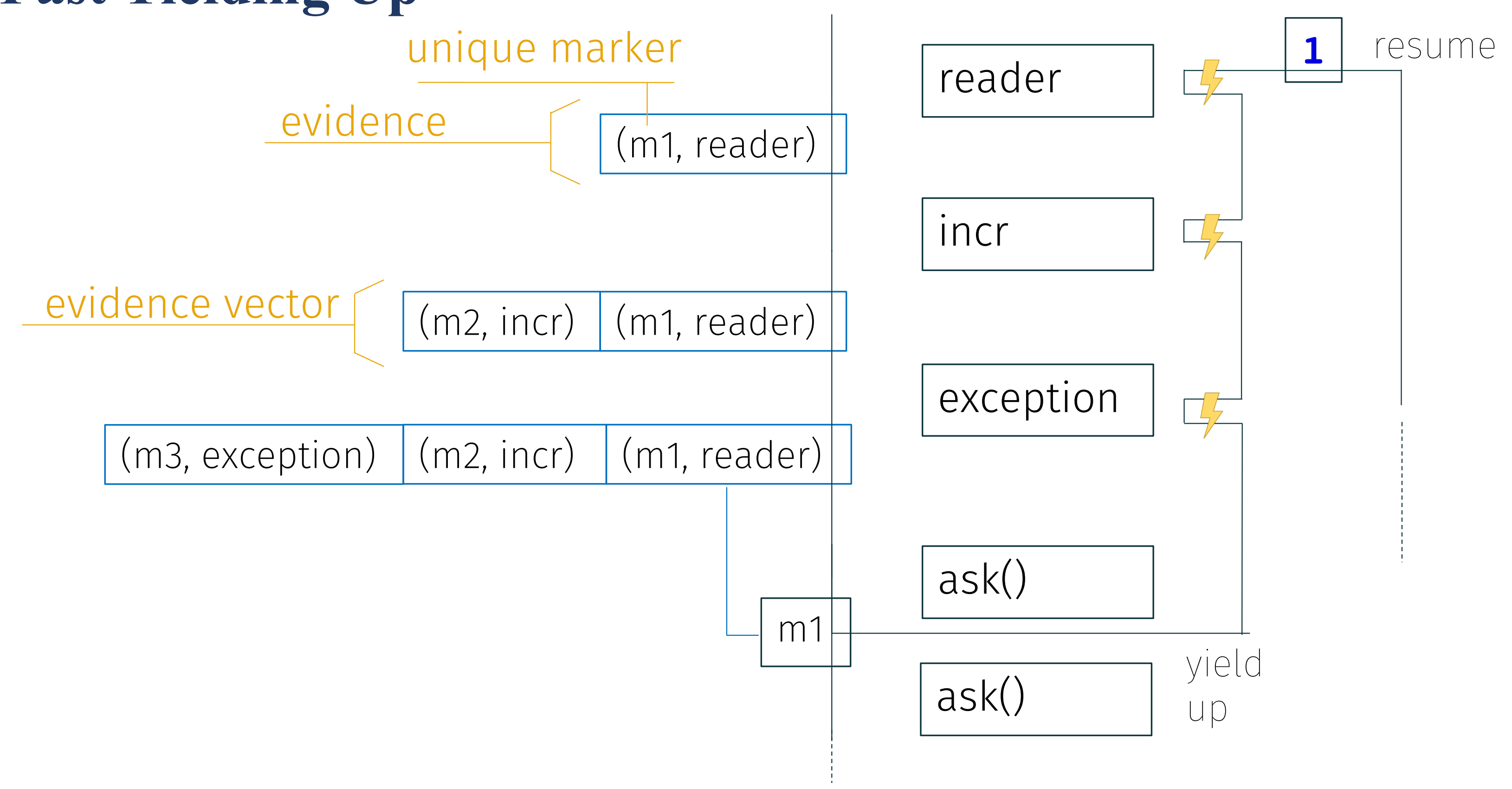
composable, modular, efficient, and easy-to-implement computational effects

Key Idea

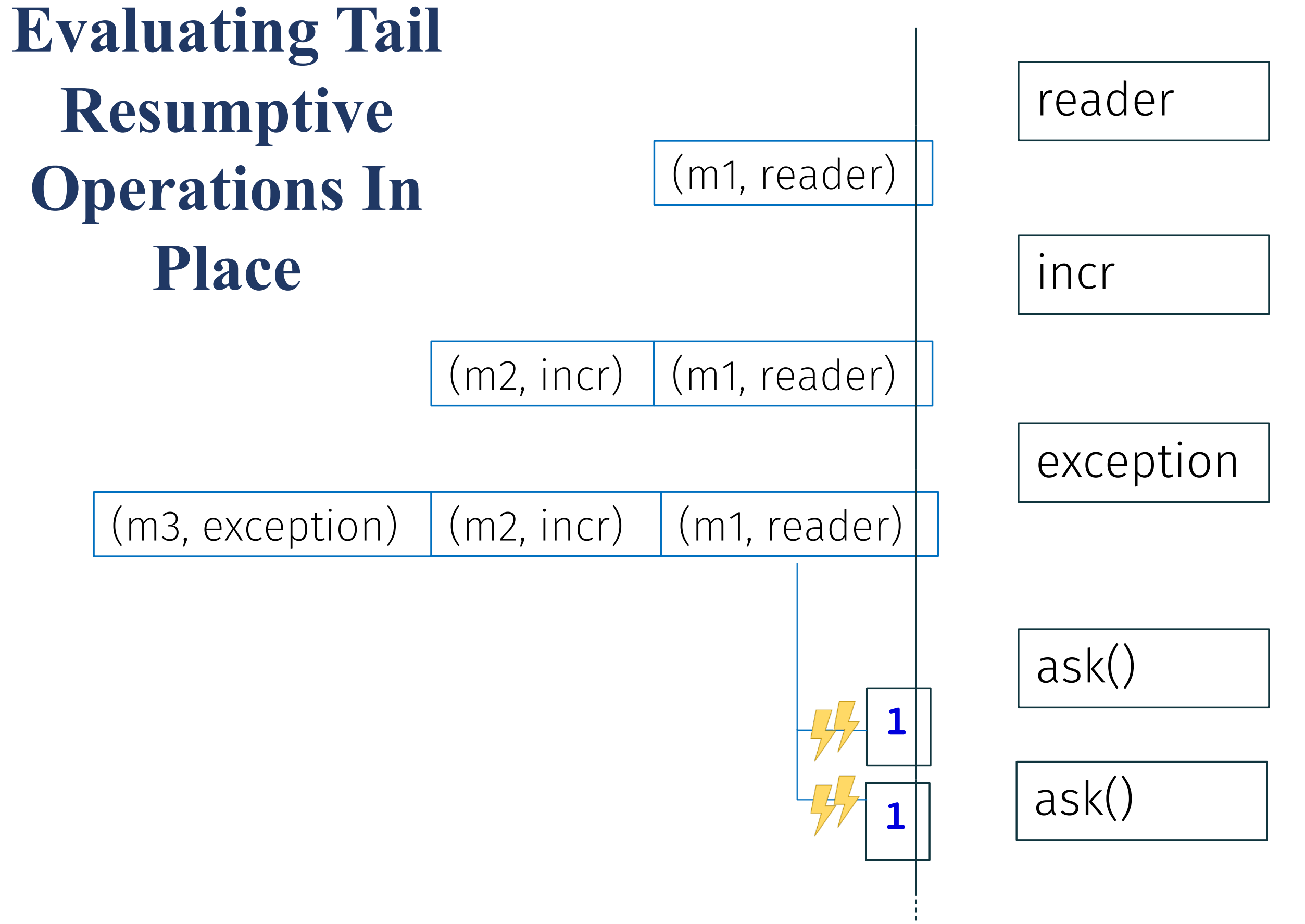
A vector of handlers is passed down as an implicit parameter to all operation invocations


polymorphic algebraic effects
F^{ϵ}
evidence-passing translation
⇝
polymorphic evidence calculus
F^{ev}
monadic multi-prompt translation
⇝
polymorphic lambda calculus
F^{v}

Fast Yielding Up



Evaluating Tail Resumptive Operations In Place



Challenge

(1) A well-typed source program translates to a well-typed target program; and

(2) their evaluation semantics coincides

Solution

Restriction: Scoped Resumption

Resumptions can only be applied in the very scope of their original handler context

Expressiveness All important effect handlers in practice

✓ No special runtime support needed

✓ Advanced compilation strategies can be used (e.g., reference counting)

Implementation

https://github.com/koka-lang/koka

Koka is a strongly typed, strict functional language which tracks the (side) *effects* of every function in its type.

Benchmarks

	runtime	evidence	direct
counter	1.00×	1.94×	2.14×
count-mod5	1.00×	1.28×	0.83×
layered	1.00×	2.23×	2.34×
nqueens	1.00×	46.09×	76.20×

nnxie@cs.hku.hk