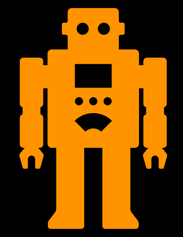
Pure Borrow

Linear Haskell Meets Rust-Style Borrowing

Yusuke Matsushita Kyoto University
Hiromi Ishii JIJ Inc.

June 19, 2026 — PLDI ’26 @ Boulder, USA

Big picture

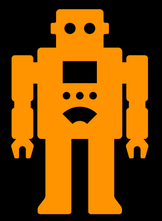
Computers 

Imperative

Software ❤

Functional

Humans 🧠



C / C++

Ownership types *Safe & Robust*

Rust

Borrows Mutation w/o ownership passing

Our work

Pure Borrow

- **Pure** & **Mutate** & **Parallelize** *Parallel quicksort*
- Just **APIs**
- **Metatheory** *WIP* *Purity by history*

Borrows !

Linear Haskell

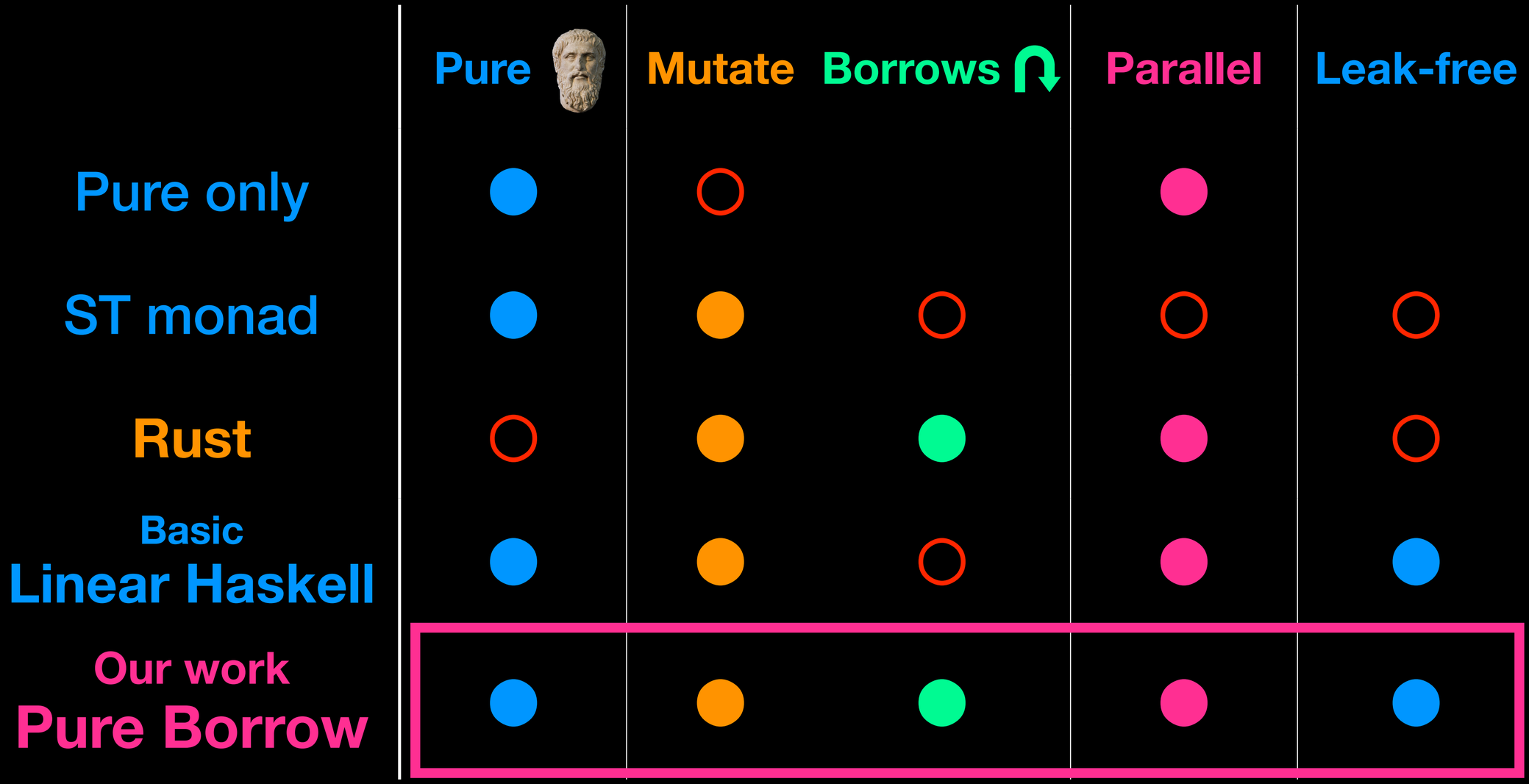
Linear types *Mutate*

Purity!

Same result regardless of timing

Parallelism
Lazy evaluation

High-level comparison

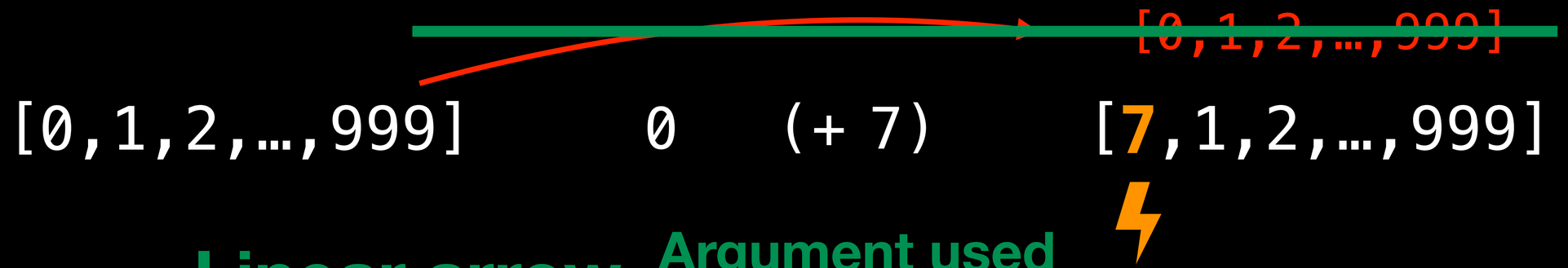


More technically

Background Linear Haskell [Bernardy+ '18]

Original Haskell Persistent data → Copy

$$\text{Vec a} \to \text{Int} \to (\text{a} \to \text{a}) \to \text{Vec a}$$



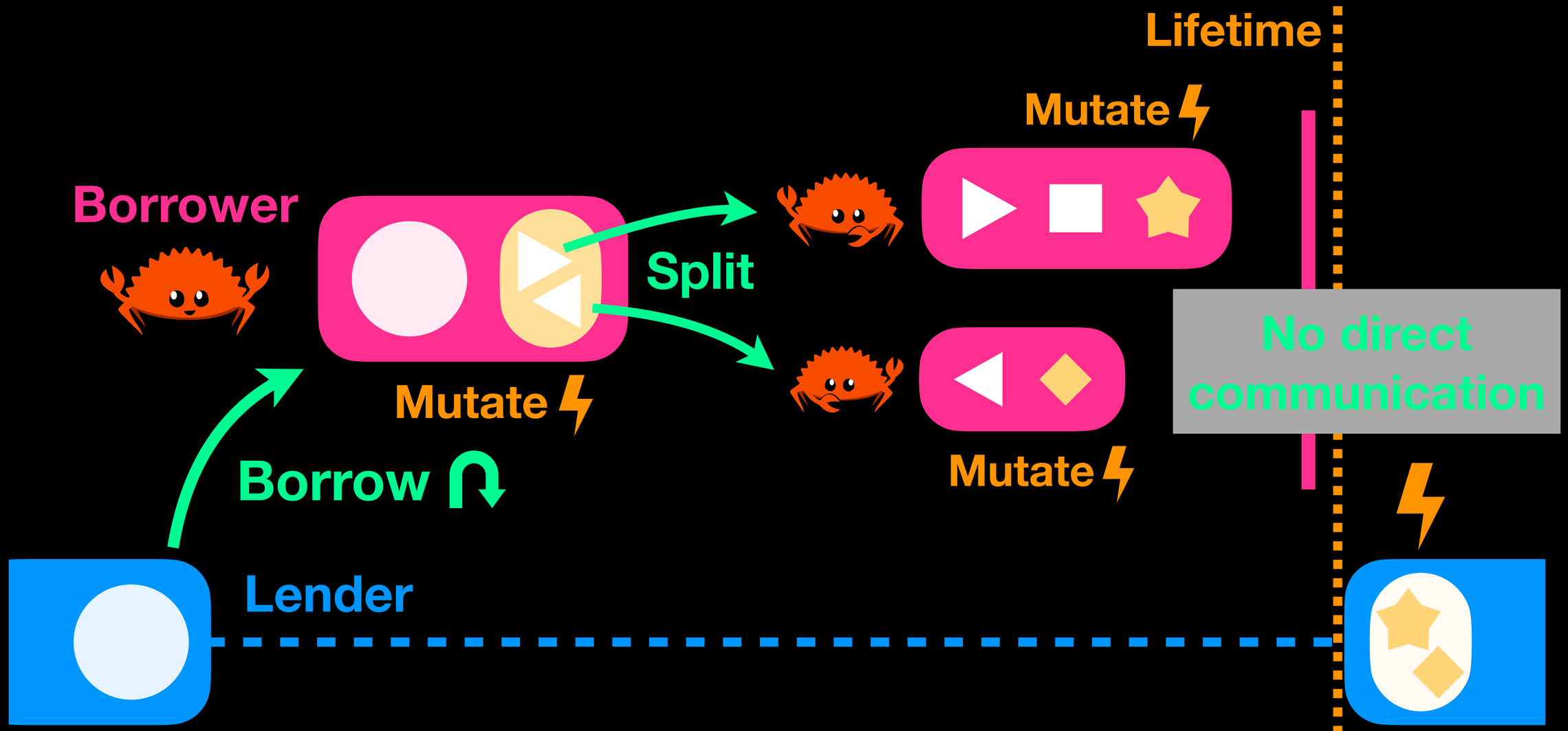
Linear arrow **Argument used exactly once**

$$\underline{\text{Vec a}} \multimap \text{Int} \to (\text{a} \multimap \text{a}) \multimap \underline{\text{Vec a}}$$

Ownership passing

Linear Haskell Linear → Mutate in place!

Background Rust-style borrowing [Matsakis+ ’14]



Our work Pure Borrow  Just APIs!

Borrower **Lender**

Borrow $\mathrm{borrow} :: \mathrm{Lin} \multimap a \multimap (\mathrm{Mut}^{\alpha}\ a, \mathrm{Lend}^{\alpha}\ a)$

Linearity witness [Spiwack+ '22]

BO Monad

Mutate $\mathrm{modifyAt} :: \mathrm{Mut}^{\alpha}\ (\mathrm{Vec}\ a) \multimap \mathrm{Int} \to (a \multimap a) \multimap \mathrm{BO}^{\alpha}\ (\mathrm{Mut}^{\alpha}\ (\mathrm{Vec}\ a))$

Split $\mathrm{splitAt} :: \mathrm{Mut}^{\alpha}\ (\mathrm{Vec}\ a) \multimap \mathrm{Int} \to (\mathrm{Mut}^{\alpha}\ (\mathrm{Vec}\ a), \mathrm{Mut}^{\alpha}\ (\mathrm{Vec}\ a))$

Drop $\mathrm{drop} :: \mathrm{Mut}^{\alpha}\ a \multimap ()$

No direct communication from borrowers to the lender

α **has ended**

Reclaim $\mathrm{reclaim} :: \mathrm{End}^{\alpha} \Rightarrow \mathrm{Lend}^{\alpha}\ a \multimap a$ 

cf. **RustBelt** [Jung+ '18]

Key idea BO Monad

$\mathrm{BO}^{\alpha}\ \mathrm{a}$ = **Computation** of a allowed during the **lifetime** α

Timing constraint t(**Borrower access**) < t(**Lender reclaim**)

Pure 

$$\mathrm{runBO} :: \mathrm{Lin} \multimap (\forall \alpha.\ \mathrm{BO}^{\alpha}\ (\mathrm{End}^{\alpha} \Rightarrow \mathrm{a})) \multimap \mathrm{a}$$

Rank-2 poly

cf. **ST Monad** [Launchbury+ '94] $\mathrm{runST} :: (\forall \mathrm{s}.\ \mathrm{ST}^{\mathrm{s}}\ \mathrm{a}) \to \mathrm{a}$

Parallel $\mathrm{parBO} :: \mathrm{BO}^{\alpha}\ \mathrm{a} \multimap \mathrm{BO}^{\alpha}\ \mathrm{b} \multimap \mathrm{BO}^{\alpha}\ (\mathrm{a}, \mathrm{b})$

Pure ← **Linearity** guarantees **disjointness**

Share & Reborrow

Temporarily share

Shared borrower

Non-linear

$$\text{sharing} :: \text{Mut}^{\alpha}\ a \multimap (\forall \beta.\ \text{Share}^{\beta \wedge \alpha}\ a \to \text{BO}^{\beta \wedge \alpha'}\ b) \multimap \text{BO}^{\alpha'}\ (b, \text{Mut}^{\alpha}\ a)$$

Lifetime intersection

$$\uparrow\ \text{share} :: \text{Mut}^{\alpha}\ a \multimap \text{Ur}\ (\text{Share}^{\alpha}\ a)$$ **+ reborrowing**

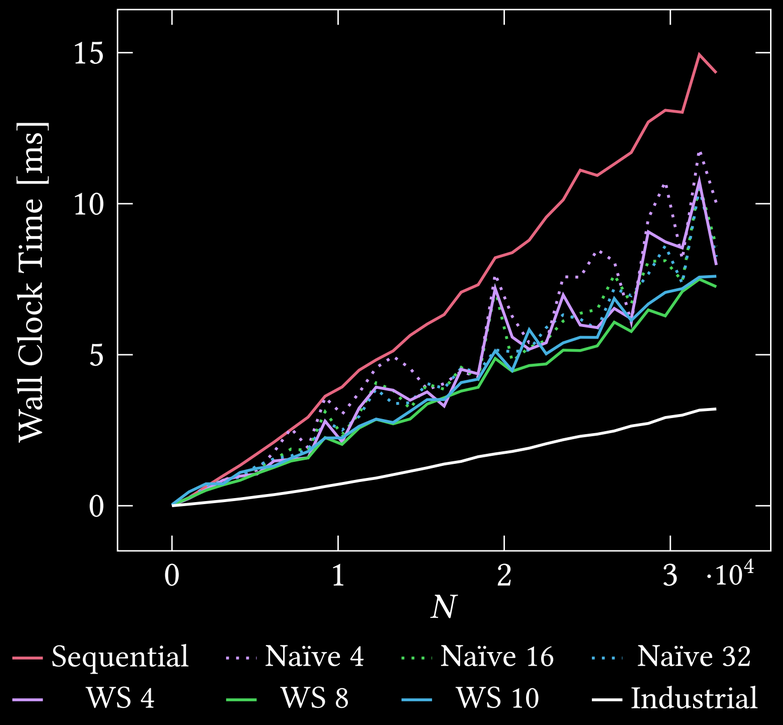
Unrestricted = Non-linear

Reborrow

$$\text{reborrowing} :: \text{Mut}^{\alpha}\ a \multimap (\forall \beta.\ \text{Mut}^{\beta \wedge \alpha}\ a \multimap \text{BO}^{\beta \wedge \alpha'}\ b) \multimap \text{BO}^{\alpha'}\ (b, \text{Mut}^{\alpha}\ a)$$

Case study Parallel quicksort

```
qsort :: Mut^α (Vec Int) ⊸ BO^α ()
qsort v = let (Ur n, v) = size v in
  if n ≤ 1 then pure (drop v) else do
    (Ur a, v) ← readAt (n // 2) v
    (i, v) ← pivotSwap a v   Mutate!
    let (lo, hi) = splitAt v i   Split!
    drop <$> parBO (qsort lo) (qsort hi)
                Parallelize!
```



Also: Work-stealing scheduler

Safe parallelized mutation by Pure Borrow!

Metatheory Purity of borrowing

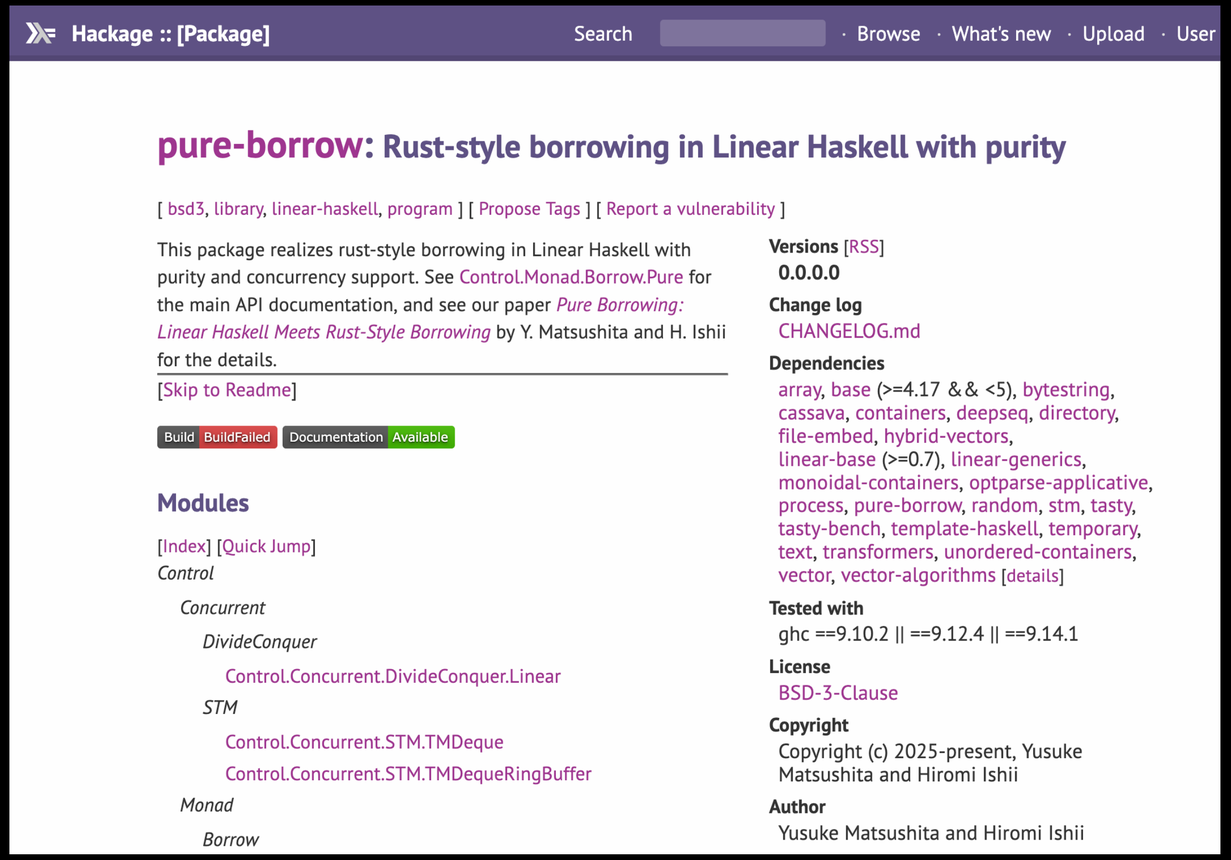
- Challenge: Justify the lender reclaim?
- Key idea: History-based operational semantics
  - BO monad can record the past mutation histories! → End^{α} a
- Theorem: History-based op. sem. is strongly confluent
  - Referential transparency: Future work, due to borrow id generation
- WIP: Correspondence with standard op. sem.
  - Roughly sketched progress & bisimulation for a dynamic, relational linear type system

Related work OxCaml [Lorenzen+ ’24, Georges ’25]

- OCaml + Ownership etc. → Safe low-level controls
- Lightweight mode annotation
  - Coarse lifetime modes: local vs global
- No purity
- Linear vs Unique [Marshall+ ’22]

comonad **many**	≤	**once**	Ur a ⊸ a	Ur a ⊸ Ur (Ur a)
unique	≤	**aliased** monad	a ⊸ (Mut$^{\alpha}$ a, Lend$^{\alpha}$ a)	Mut$^{\alpha}$ (Mut$^{\beta}$ a) ⊸ Mut$^{\alpha \wedge \beta}$ a

Pure Borrow is already there!



Hackage :: [Package] Search · Browse · What's new · Upload · User

pure-borrow: Rust-style borrowing in Linear Haskell with purity

[bsd3, library, linear-haskell, program] [Propose Tags] [Report a vulnerability]

This package realizes rust-style borrowing in Linear Haskell with purity and concurrency support. See Control.Monad.Borrow.Pure for the main API documentation, and see our paper *Pure Borrowing: Linear Haskell Meets Rust-Style Borrowing* by Y. Matsushita and H. Ishii for the details.

[Skip to Readme]

Build BuildFailed Documentation Available

Modules

[Index] [Quick Jump]

Control

Concurrent

DivideConquer

Control.Concurrent.DivideConquer.Linear

STM

Control.Concurrent.STM.TMDeque

Control.Concurrent.STM.TMDequeRingBuffer

Monad

Borrow

Versions [RSS]
0.0.0.0

Change log
CHANGELOG.md

Dependencies
array, base (>=4.17 && <5), bytestring, cassava, containers, deepseq, directory, file-embed, hybrid-vectors, linear-base (>=0.7), linear-generics, monoidal-containers, optparse-applicative, process, pure-borrow, random, stm, tasty, tasty-bench, template-haskell, temporary, text, transformers, unordered-containers, vector, vector-algorithms [details]

Tested with
ghc ==9.10.2 || ==9.12.4 || ==9.14.1

License
BSD-3-Clause

Copyright
Copyright (c) 2025-present, Yusuke Matsushita and Hiromi Ishii

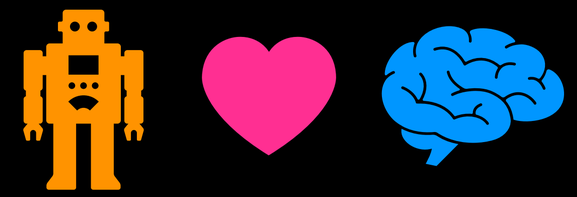
Author
Yusuke Matsushita and Hiromi Ishii

Pure Borrow works on

GHC w/o tweaks

Give it a try!

Pure Borrow — Summary



- Linear Haskell + Rust-style borrowing
  - Pure & Mutate w/o ownership passing & Parallelize
- BO monad — Lifetime constraint for borrows
  - Like ST monad, but with linearity & parallelization
- Case study — Parallel quicksort
- Metatheory — Purity by history-based borrow model

Thank you!