# The Wide World of Actors, or, Can I Have an Erlang Pony?

Code BEAM San Francisco
Friday, March 16, 2018

Scott Lystig Fritchie
Wallaroo Labs

# Introducing Myself


- I am Scott Lystig Fritchie
- Currently at Wallaroo Labs
- Formerly of VMware Research, Basho, Gemini Mobile, Caspian Networks, Sendmail, and UNIX sysadmin prior
- Former co-chair of ACM ICFP Erlang Workshop for 4 years
  - International Conference on Functional Programming
- @slfritchie at GitHub and Twitter
- I eat and cook a lot of Japanese food

# One Slide About Wallaroo


- Wallaroo = data streaming processor
  - Easy scaling of Python & Go processing logic
- Targets apps not well-served by Storm, Flink, Spark, …
  - Very low latency = efficiency
  - Very low jitter = predictable tail latencies
  - Fast interface to foreign language interpreters (in C!)
- Wallaroo as great Erlang app?
  - Sure, but …
  - … it's written in Pony

# Outline of the Talk


- BEAM =:= Actors
  - False!
- A Brief & Biased History of Programming
- Definition of the Actor Model
- 20+ Extra Dimensions to the Actor Model
- Actor implementations: BEAM languages vs. Pony
- Cool Pony Stuff Outside of the Actor Model

# My Goals


- Better understanding of where the Actor Model came from.
- Many dimensions to design & build an Actor Model system.
  - BEAM is an opinionated implementation.
- BEAM & Pony are quite similar
  - … but the exceptions are **big exceptions**.
- Pony's implementation of Actor Model might be better that BEAM's in some cases.
- Pony is interesting enough to learn more about.
- Type systems are amazing tools. Don't ignore Dialyzer!

# Programming History in 1 Bad Slide


- Programming is COOL!
- Writing & debugging programs is NOT EASY
- (Industry introduces timesharing & concurrency)
  - (… programming languages invented …)
  - (… structured programming invented …)
- Managing concurrency is DIFFICULT
- Managing concurrency + actual simultaneous execution is WICKED HAHD


We must manage complexity or else go insane.


# BEAM =:= Actors

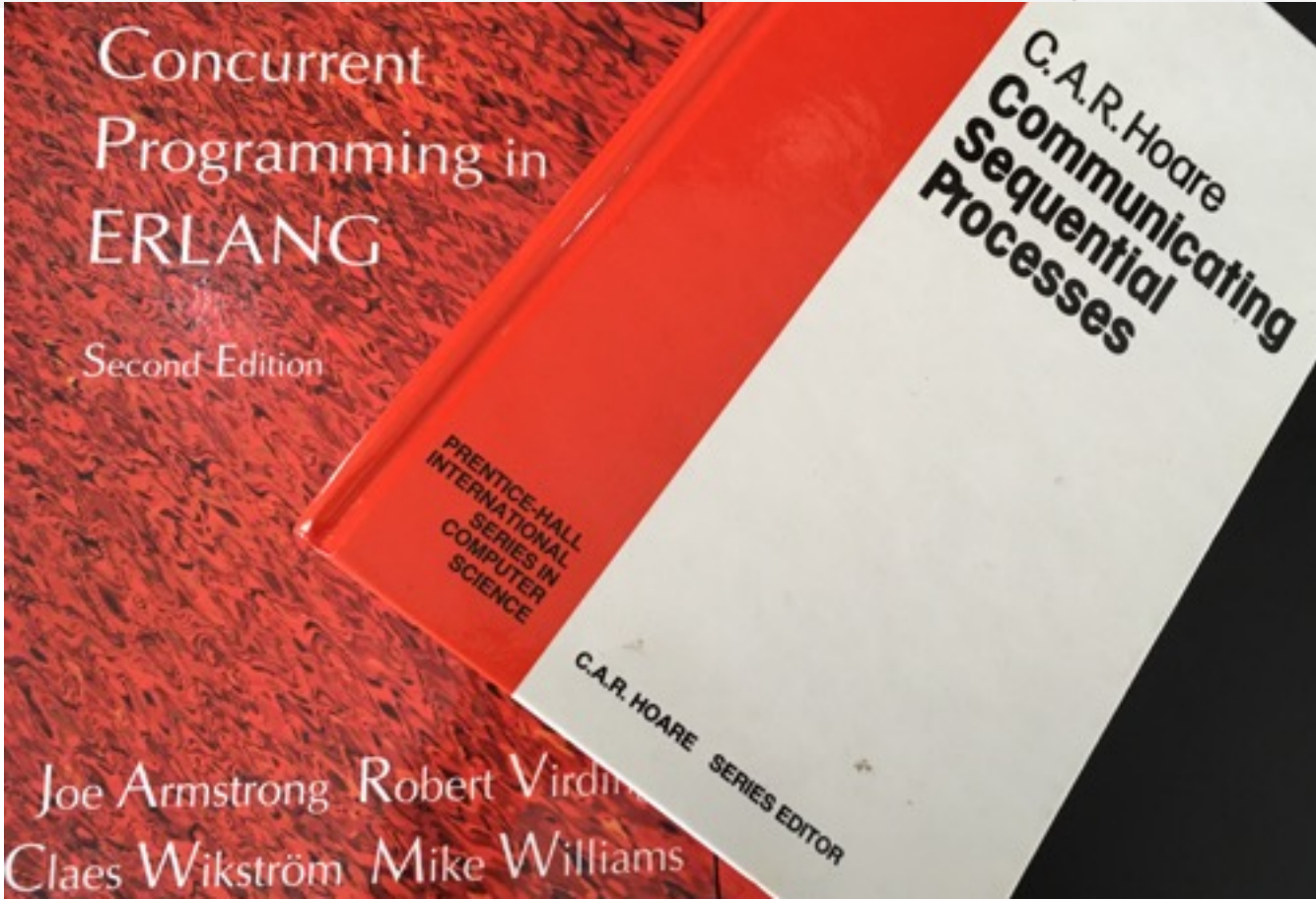
false

# The Actor Model


1. The actor is the fundamental unit of computation
2. An actor has its own private state: registers, memory, etc.
3. An actor can read & modify only its own private state
    - It is **private state**: no other actor has any access
4. An actor can send a message to another actor
5. An actor can react to a message that was sent to it
    - Message passing is the only communication mechanism between actors
6. An actor can create a new actor

# Communicating Sequential Processes (CSP)

- Hoare (c) 1985
- Armstrong, Virding, Wikström, Williams (c) 1993




# CSP + opinionated telecom giant + research lab

equals Erlang


# Let's get more specific about what an actor implementation might really need

# Actor Model Details


- Message sending:
  - sync vs. async message sending
  - named vs. unnamed processes
  - message destination: process vs. channel
    - 1:1?
    - broadcast?
    - other?
  - typed vs. untyped messages

# Actor Model Details


- Message receiving
  - Reliable vs. unreliable delivery
  - First in first out (FIFO) vs. causal vs. another order
  - Blocking semantics?
    - Block waiting for a message?
  - Time-aware vs. time-ignorant

# Actor Model Details


- How actors are scheduled to execute?
  - When to start running an actor?
  - When to stop running an actor
    - Cooperative vs. preemptive scheduling
  - Work stealing?

# Actor Model Details


- Memory limits
  - Channel/ambient/mailbox limits?
    - “In transit” messages?
    - “At rest” messages?
- Back-pressure vs. buffering only
- Actor memory limits?
- All of the above: penalty for violating limits?

# Actor Model Details


- Message delivery order
  - Causal vs. FIFO vs. no guarantee vs. other?
  - Messages duplication allowed?
  - Message loss allowed?

# Actor Model Details


- Actor lifetime
  - Do actors exist forever?
  - Can actors crash?
- Can actors interact with non-actor computations?
- Byzantine/malicious actor behavior?


# Enough!?

it is a good start. but there is more.


# BEAM languages vs. Pony

## 20+ Dimensions of the Actor Model

# Message Sending


Synchronous vs. Asynchronous message sending

- BEAM: async
- Pony: async

SAME

# Message Sending


Named Processes vs. Unnamed Processes

- BEAM: named
- Pony: named

SIMILAR

# Message Sending


## Message Destination

- BEAM: process
- Pony: actor

SAME

# Message Sending


## Typed vs. Untyped Messages

- BEAM: untyped
- Pony: typed

WHOA!

# Message Receiving


Reliable vs. Unreliable Delivery

- BEAM: reliable'ish
- Pony: reliable

SIMILAR

# Message Receiving

Message delivery order

- BEAM: any order
- Pony: FIFO only

WHOA!

# Message Receiving


Causal message order guarantee

- BEAM: yes or no
- Pony: yes always

SIMILAR

# Message Receiving

Blocking vs. Non-Blocking message receive

- BEAM: yes
- Pony: no

WHOA!

# Message Receiving


Time-Aware vs. Time-Ignorant

- BEAM: yes
- Pony: no

WHOA!

# Scheduling


What schedules actors?

- BEAM: custom scheduler
  - 1 scheduler/CPU core
- Pony: custom scheduler
  - 1 scheduler/CPU core

SAME

# Scheduling


## Scheduler Overhead

- BEAM: {100's} bytes/process, {few} usec to create & destroy
- Pony: 240 bytes/actor, {few} usec to create & destroy
- Scheduling millions is fine
- Actors are cheap

SAME

# Scheduling

Preemptive vs. Cooperative Scheduling

- BEAM: Preemptive
- Pony: Cooperative

WHOA!

# Scheduling

Actor priority schemes?

- BEAM: Yes, 4 levels
- Pony: No

WHOA!

# Scheduling


Work stealing?

- BEAM: Yes
- Pony: Yes

SAME

# Scheduling


Energy Conservation by Idle Schedulers?

- BEAM: Yes
- Pony: Yes

SAME

# Scheduling


Mailbox size limits?

- BEAM: No
- Pony: No

SAME

# Scheduling


Maximum Heap Size?

- BEAM: No
- Pony: No

SAME

# Scheduling

- Actor Lifecycle
  - Cheap vs. Cheap *SAME*
- Actor Crash?
  - Yes vs. No

WHOA!

# Scheduling

Back-pressure to reduce workload of overloaded actors?

- BEAM: Yes -> No
- Pony: Yes

WHOA!

# Theoretical Message Delivery Properties


- Causal order: Yes
  - *SIMILAR*
- - Message loss: 0%
  - *SAME*
- - Message duplication: 0%
  - *SAME*
- - Message reordering:
  *WHOA!*

WHOA!

# Actors & the Outside World


Actor interaction with non-actors

- BEAM: yes
- Pony: yes, but…

SIMILAR

# Byzantine Actors


Incorrect/Malicious Actors Tolerated?

- BEAM: No
- Pony: No

SAME

# Review of Similarities by Category


- SAME
  - 13
- SIMILAR
  - 5
- WHOA!
  - 8

# WHOA! Summary


- Msg Receiving: message reordering
- Msg Receiving: blocking vs. non-blocking receive
- Msg Receiving: time-aware vs. time-ignorant
- Scheduling: preemptive vs. cooperative scheduling
- Msg Sending: untyped vs. typed messages
- Scheduling: actor priority schemes?
- Lifecycle: actors crash?
- Back-pressure for "overloaded" actors?

# In Pony, one does not simply call() a gen_server

ever.


# In Pony, one does not simply call() a gen_server

you cannot block awaiting for the reply.


# In Pony, all messaging is cast()-style


# Good Stuff Not in the "Actor Model" Basket

# Pony language & runtime safety guarantees


- Type safe
- Memory safe
- Exception safe
- Data-race free
  - All messaging is pass-by-reference
  - Sharing data between actors is guaranteed safe
- Deadlock free
- Type system is fully aware of actors & concurrency

# If the Compiler Isn't Happy, Nobody's Happy


- Pony's compiler is **FUSSY**
  - Far more than Erlang's or Elixir's compiler
- But it’s always right(*)
- So is the Dialyzer
  - Type systems are powerful tools
  - Dialyzer finds bugs in BEAM code
  - Use Dialyzer to fix your bugs
  - Put Dialyzer into your workflow so you can’t ignore it

# Pony compiles to target hardware CPU


- Erlang, Elixir, LFE, etc.
  - Runs on BEAM VM with optional compilation to native code via HiPE
- Pony
  - Compiles to target CPU instructions via LLVM toolchain
  - JIT is available via LLVM
  - DWARF symbols, “looks like C++” to debuggers and profilers

# Side-effect of safety: actors don't crash


- All errors must be handled explicitly
  - “?” syntax used to mark a "partial function”
    - "partial" = "may raise an error"
- Compiler enforced, of course
- No actor crashes => no need for BEAM’s links & monitors to help manage failure

# Per-Actor Heaps + Distributed GC


- Distributed GC across all actor heaps
  - No "stop the world" GC
  - Fully concurrent, sync-free, lock-free, and barrier-free
- Message passing maintains ref counts on shared objects
  - Dead objects are reaped by creating/allocating object
- GC and Type System **Co-Designed** with ORCA protocol
  - Actors are 1st class, GC'ed objects in the system
  - Runtime halts when all actors are GC'ed/GC'able.
  - ORCA works (on paper) across multiple machines

# ORCA GC Comparison on μB'marks



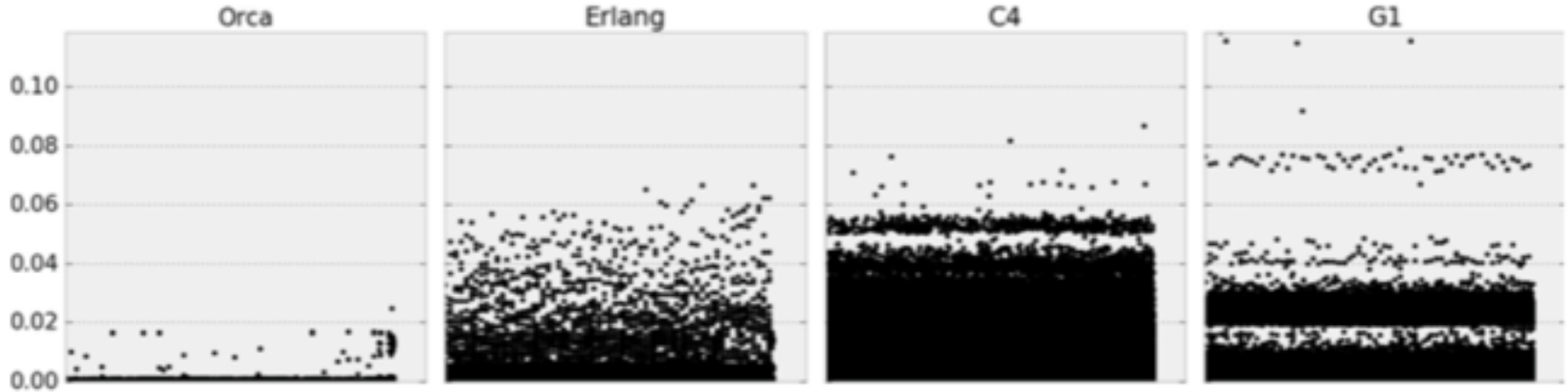




Fig. 17. Responsiveness. X-axis: request ID, Y-axis: Jitter/difference between finishing time (seconds) of subsequent requests. Java measurements are from a warmed-up VM and does not include JIT'ing.

# ORCA GC Comparison on µB'marks

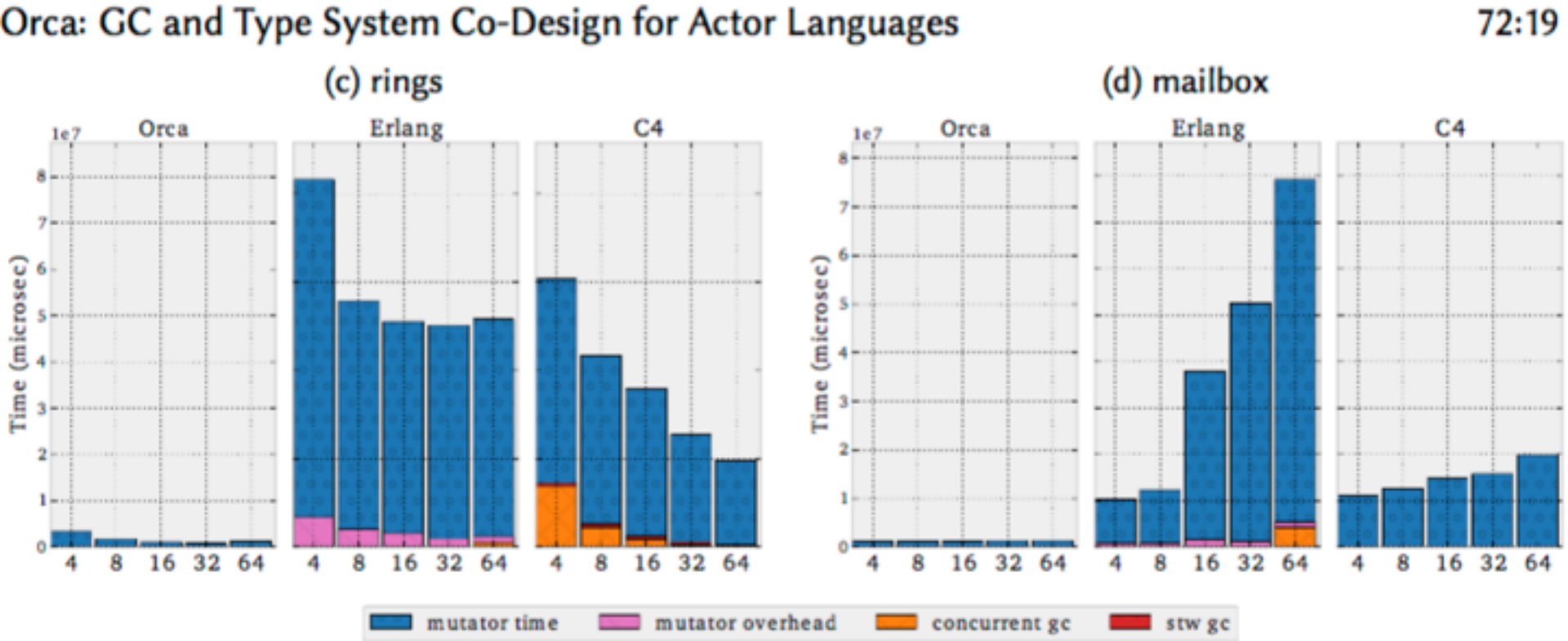



Fig. 16. Strong scalability on 4–64 cores. (stw=stop-the-world.)

# ORCA GC Comparison on µB'marks

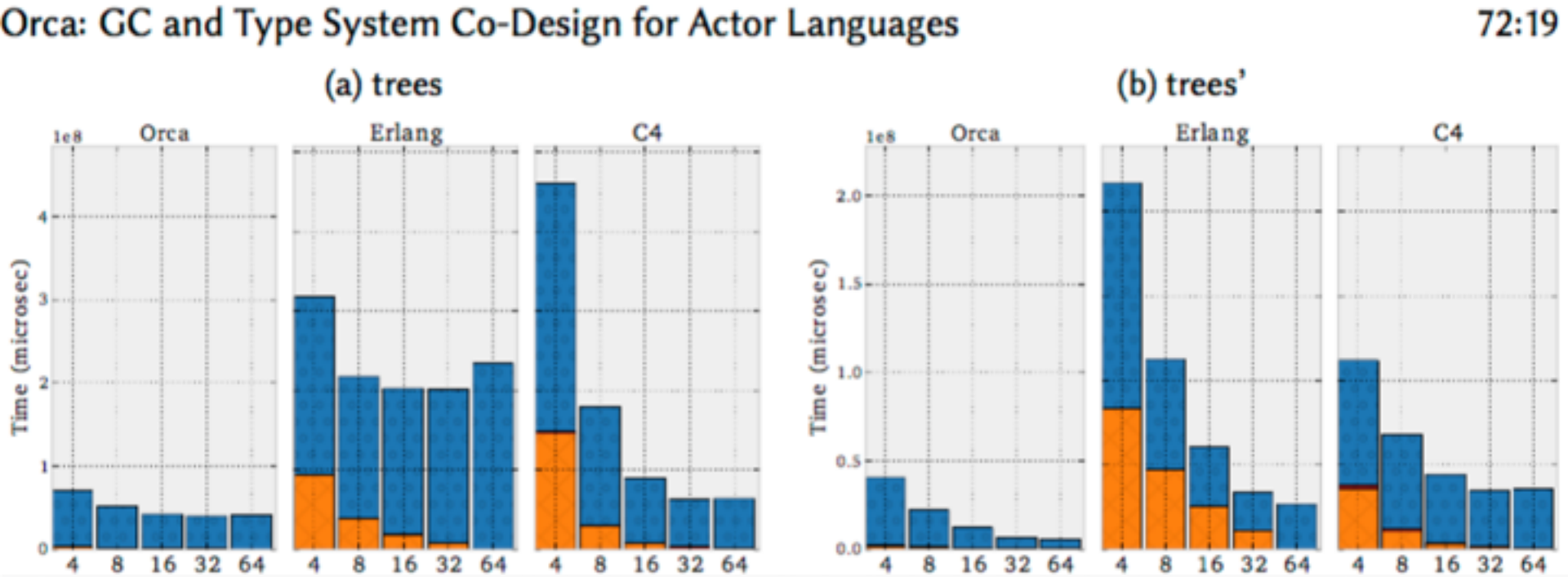



Fig. 16. Strong scalability on 4–64 cores. (stw=stop-the-world.)

# Pony Is Not a Functional Language


- Pony is very imperative
- … but the type system provides lovely safety properties

# Pony Has Lambdas & More


- lambdas / unnamed functions
- `map()` & friends, hooray
- persistent data structures in the standard library

# Pony Is Object-Oriented


- … but not Java-style
- Not **everything** is an object
  - You control the class hierarchy
- Has both structural & nominal subtyping
  - Pony's `interface` = structural typing

# Pony Has Generic Types


```
// map over a List[A] to
// create a List[B]

fun box map[B: B](
  f: {(this->A!): B^}[A, B] box)
: List[B] ref^
```

# Pony Has Pattern Matching!


- `match` statement to match:
  - basic data types
  - sub-/super-types in class hierarchy
  - tuple element destructuring
- Function head matching is gone
  - … but will return again soon (I hope)

# Pony Is Open Source


- BSD-style license
- https://github.com/ponylang/ponyc/
- Target CPUs
  - x86_64, ARM
- Target operating systems:
  - Linux, Windows, OS X
  - FreeBSD & DragonflyBSD (limited support)

# Pony Is Young


- The standard library is small
- The open source community is small
- Ecosystem of Pony language libraries & apps is small
- "Get Stuff Done" development model
  - Correctness > Performance > Simplicity > Consistency > Completeness

# Pony's FFI to C/C++ ABI


- Easily interface to C & C++ ABI functions
- Runtime's requirements for memory & threads are modest
- Many Pony primitive data types map directly to target CPU
  - `I8, I16, I32, I64, I128`
  - `U8, U16, U32, U64, U128`
  - `Array[U8]` for contiguous unstructured bytes

# Pony's Reference Capabilities


- Strong, static type checker is the price to pay for safety
- It’s a big mind shift to adjust to both:
  - Mutable data (even if it is safe!)
  - Pony’s type system (based on affine types)
- The end advantages:
  - Zero runtime cost for safety
  - Very quick GC

# Get Involved!


- Web: http://ponylang.org
- GitHub: https://github.com/ponylang/ponyc/
- Twitter: `@ponylang`
- Freenode IRC: `#ponylang`
- Mailing list info: https://pony.groups.io/g/user
- Pester me about Erlang, Pony, and/or Wallaroo:
  - Anytime here at the conference
  - `@slfritchie` on Twitter
  - `slfritchie@` on gmail.com

# Sources & Where to Look For More


On the Actor Model:

- https://en.wikipedia.org/wiki/Actor_model
- https://en.wikipedia.org/wiki/Process_calculus
- https://en.wikipedia.org/wiki/Actor_model_and_process_calculi
- https://en.wikipedia.org/wiki/Communicating_sequential_processes

On Pony:

- http://blog.acolyer.org/2016/02/17/deny-capabilities/
- https://blog.acolyer.org/2016/02/18/ownership-and-reference-counting-based-garbage-collection-in-the-actor-world/
- https://www.youtube.com/watch?v=e0197aoljGQ
- https://github.com/ponylang/ponyc/
- http://ponylang.org (also Pony logo source)

Source of microbenchmark graphs:
S Clebsch, J Franco, S Drossopoulou, AM Yang, T Wrigstad, J Vitek
“Orca: GC and type system co-design for actor languages”. Proceedings of the ACM on Programming Languages 1 (OOPSLA), 72
https://uu.diva-portal.org/smash/get/diva2:1160319/FULLTEXT01.pdf

Sean Bean image:
New Line Cinema, The Fellowship of the Ring, 2001
http://knowyourmeme.com/memes/one-does-not-simply-walk-into-mordor
https://memegenerator.net/Does-Not-Simply-Walk-Into-Mordor-Boromir