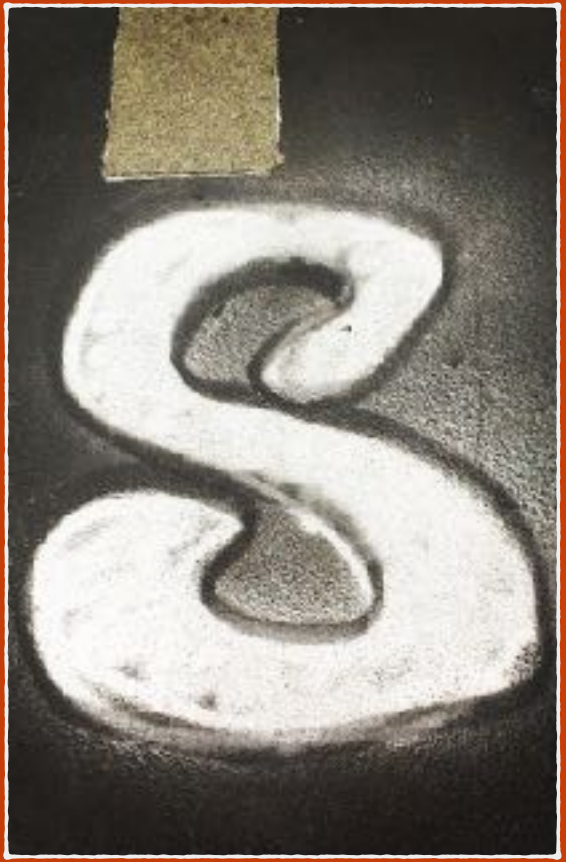
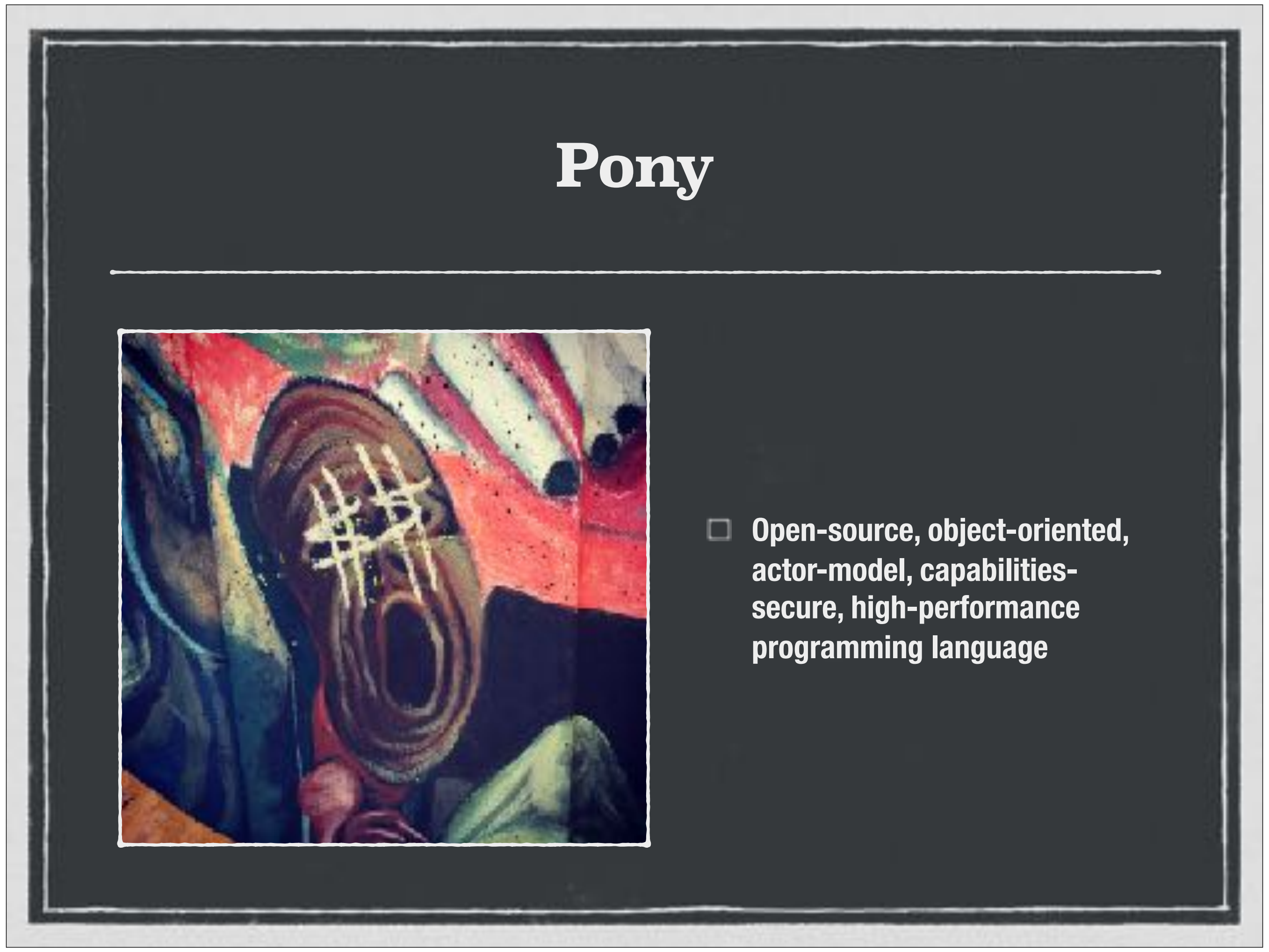
Pony

How I learned to stop worrying and embrace an unproven technology

Sean T Allen

Author of Storm Applied
Member of the Pony core team
VP of Engineering at Sendence





Pony

- Open-source, object-oriented, actor-model, capabilities-secure, high-performance programming language

google: “pony for fintech”
type safe
memory safe - no dangling pointers or buffer overruns
exception safe - no uncaught runtime exceptions
data race free
deadlock free
native code - ahead of time compiled. llvm
c compatible

This talk is...

The story of Pony told through the story of Wallaroo; the product we built with it...

Wallaroo

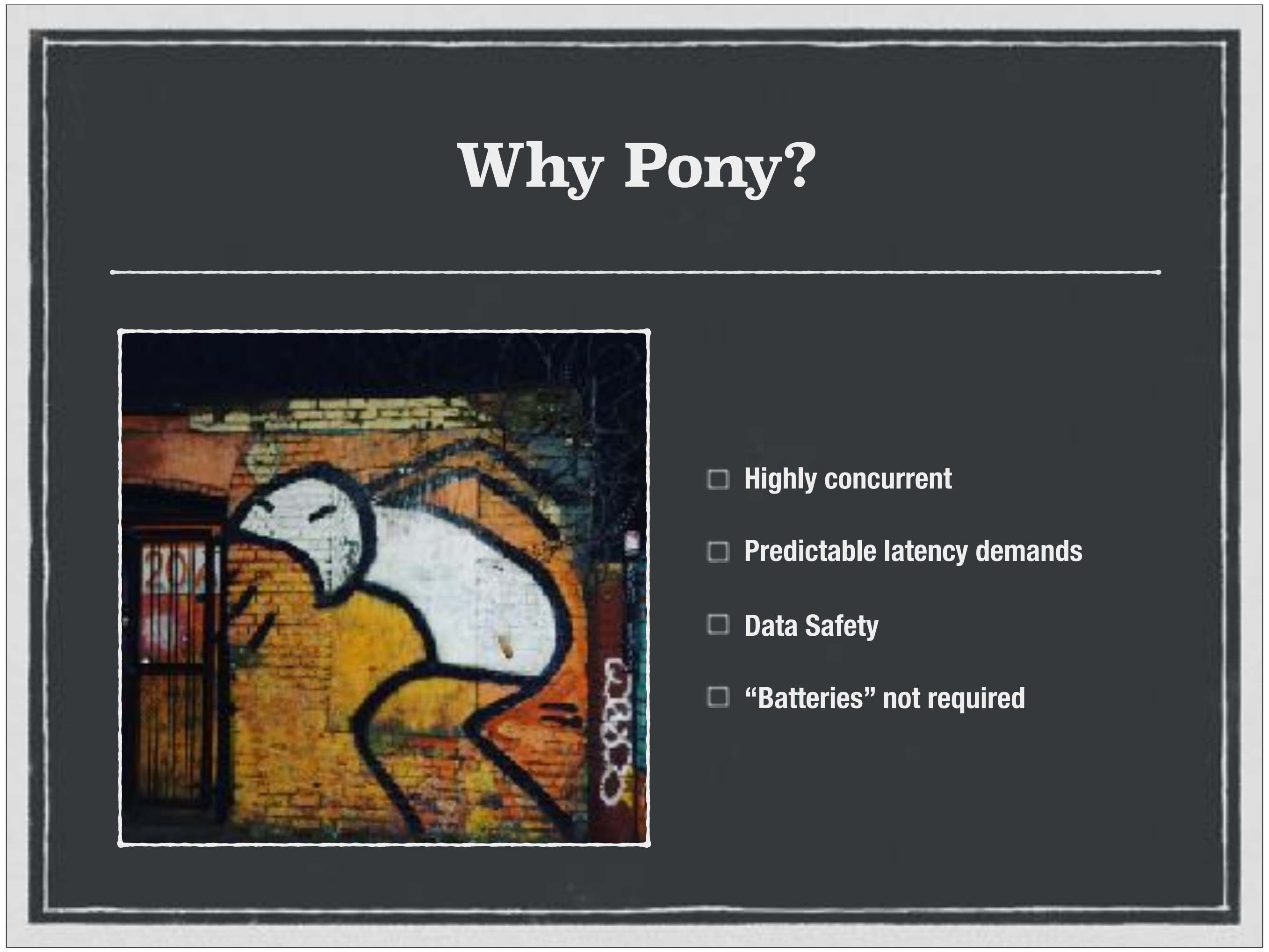
- High-performance, distributed stream processor
- High-throughput
- Low-Latency
- Consistent performance
- Resource efficient

discuss risk management, position keeping etc type systems.
lower cost of building these system
microsecond latency targets
we've talked to plenty of financial institutions that regularly miss their "targets"

“A programming language is just another tool. It’s not about syntax. It’s not about expressiveness. It’s not about paradigms or models. It’s about managing hard problems.”

-Sylvan Clebsch

Why Pony?



Why Pony?

- Highly concurrent
- Predictable latency demands
- Data Safety
- “Batteries” not required

We can expand “data safety” out to mean “memory safety”

Highly Concurrent

Highly Concurrent

- Actor model
- Async messaging
- Work stealing scheduler
- Mechanical sympathy

actor-model is a model of concurrency. consider it “managed threads”.
concurrency aware type system.
lock contention, deadlocks. not a problem thanks to the actor model.

if objects are state + synchronous methods then actors are state + async.
message queue per actor.
actors have their own heap. actors are individually garbage colllected.

actor model people think “erlang” or “akka” but there are differences. don’t assume that if its in erlang that is what it means to be actor model.
type system. “let it crash”.

Predictable latencies

Predictable Latencies

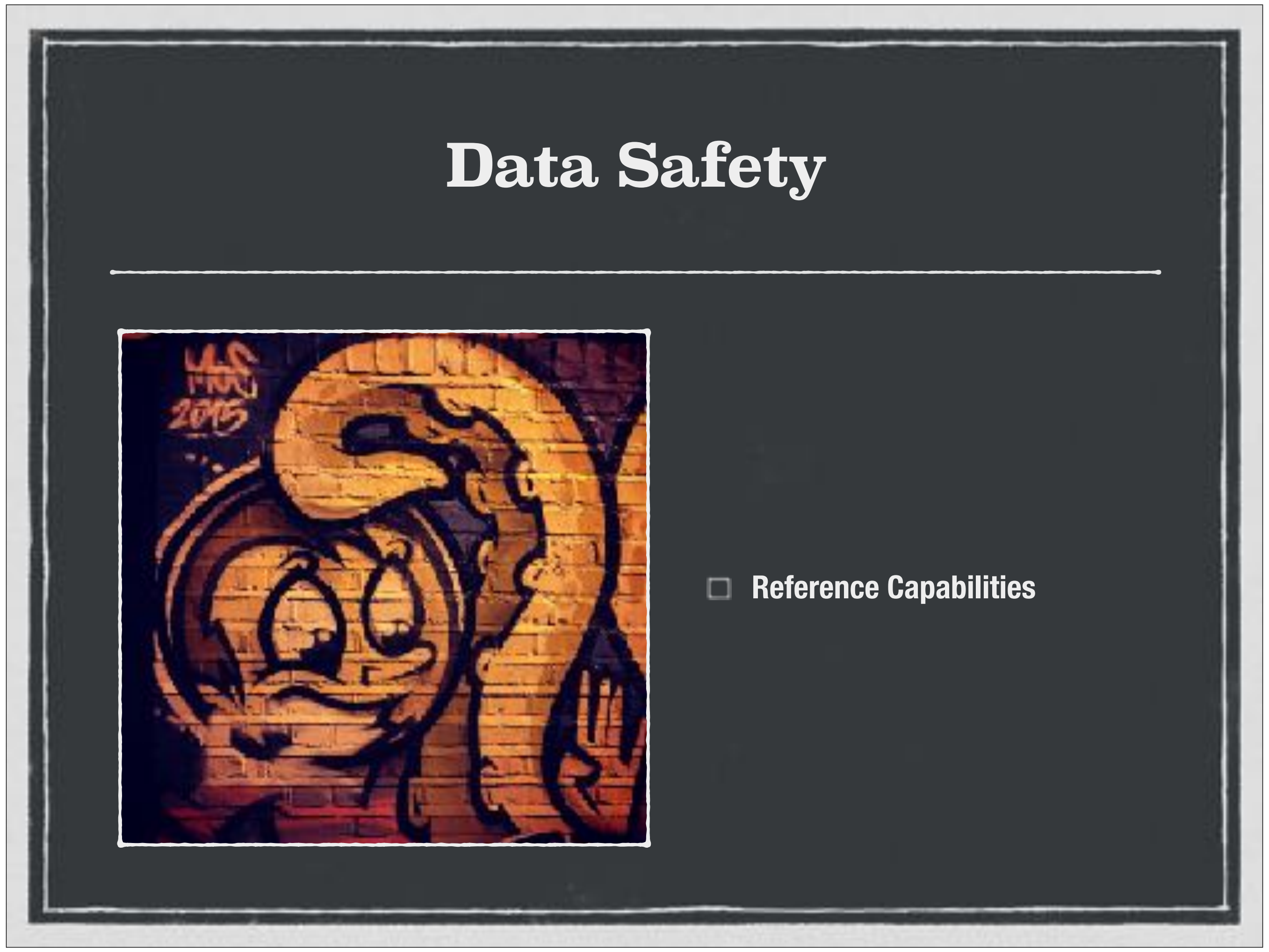
- No “stop the world” GC
- Per-actor heaps
- Better clustering

per actor heaps (like erlang, unlike akka)
erlang generally has "copy on message pass" semantics. copying can be slow. we want to go fast, but safely-- reference capabilities

because we have per actor heaps, there's concurrent garbage collection. the entire runtime does not stop to collect garbage.
trash day. cluster.

ORCA - there's a paper and a proof. however, we recently found a hole in the proof that is being worked on.

Data Safety



Data Safety

- Reference Capabilities

reference capabilities explained: what are the types? what are the patterns they codify? : ref, iso, val, tag (there are a couple others but let's not worry about them)

anecdote about how I came to "functional" programming in the 90s. "how i learned to stop segfaulting everything"

anecdote about early struggles with reference capabilities and the "aha moment" of o god that bug would have taken so long to find.

do fast unsafe things because in this situation we can prove that its safe. that said, the compiler could be smarter in places. ¯_(ツ)_/¯. software its a highway.

Batteries not required

Batteries not required

- Small Standard Library
- C-FFI

small standard library.
c-ffi is good but... some c libraries like to do things with globals and threads that can be very bad. you can't really use those particular libraries.

Results

Results

- 18 month jumpstart for “free”
- Excellent performance
- More confident refactoring
- Some pain

Pony runtime has been huge win for us. Excellent performance. Didn’t need to write our own work scheduler or garbage collection.

Almost no-segfault (c-ffi is a dangerous thing)

Pain

Pain

- Pre-1.0
- Compiler bugs
- Breaking changes
- Limited tooling
- Runtime knowledge required

pre-1.0, regular breaking changes (but work needed to keep up is small)

lack of tooling sometimes hurts, particularly wish there was a quickcheck for Pony

runtime knowledge (how does GC work) required to get wicked fast code

Is Pony right for you?

Is Pony right for you?

Yes, if…

- You have a hard concurrency problem
- You aren't reliant on a lot of existing libraries
- You are willing to write "most everything" from scratch

building to interoperate with other systems.
taking data off of real systems into real systems
you don't have to replace everything to use pony, just the parts that need to do what pony is good at

“Fintech” Questions

“Fintech” Questions

- How do I shard my data?
- How do I do priority queues?
- How do I only process the most recent messages?

“You might have more, I’m happy to take them in general after the talk”

How do I shard my data?

Priority Queues

Only process the last message (the NBBO/market data question)

Takeaways…

Pony has a powerful, data-race free, concurrency-aware type system

The Pony runtime can help you solve hard concurrency problems

You might be able to use Pony for fintech in production now

Trash Day paper:
https://www.usenix.org/system/files/conference/hotos15/hotos15-paper-maas.pdf

ORCA paper:
https://www.ponylang.org/media/papers/OGC.pdf

@seantallen

www.monkeysnatchbanana.com

@SendenceEng

www.sendence.com

@ponylang

www.ponylang.org

#ponylang on freenode