PONY

Sylvan Clebsch, Sebastian Blessing, Sophia Drossopoulou, Andrew Mc Neil



Causality Ltd and Imperial College London

Pony – a programming language

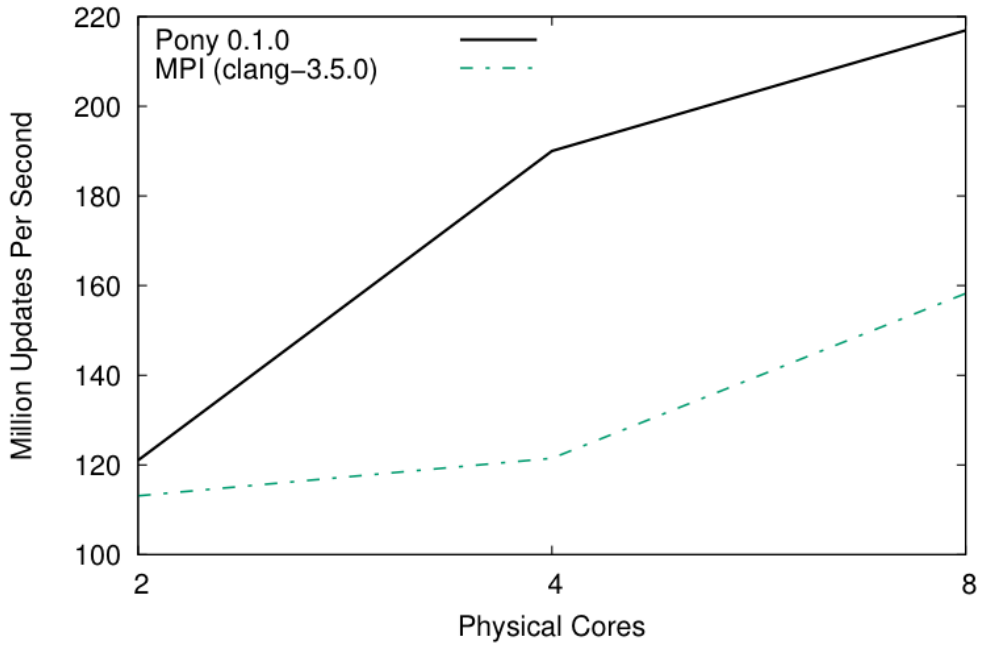
aiming

- for concurrent (distributed) programming,
- very fast,
- easy to learn, easy to use,
- data-race free and atomicity.

Very fast?

GUPS benchmark (from Linpack suite)

- Measures rate of random updates of memory
- First Pony implementation (~4 days of work)

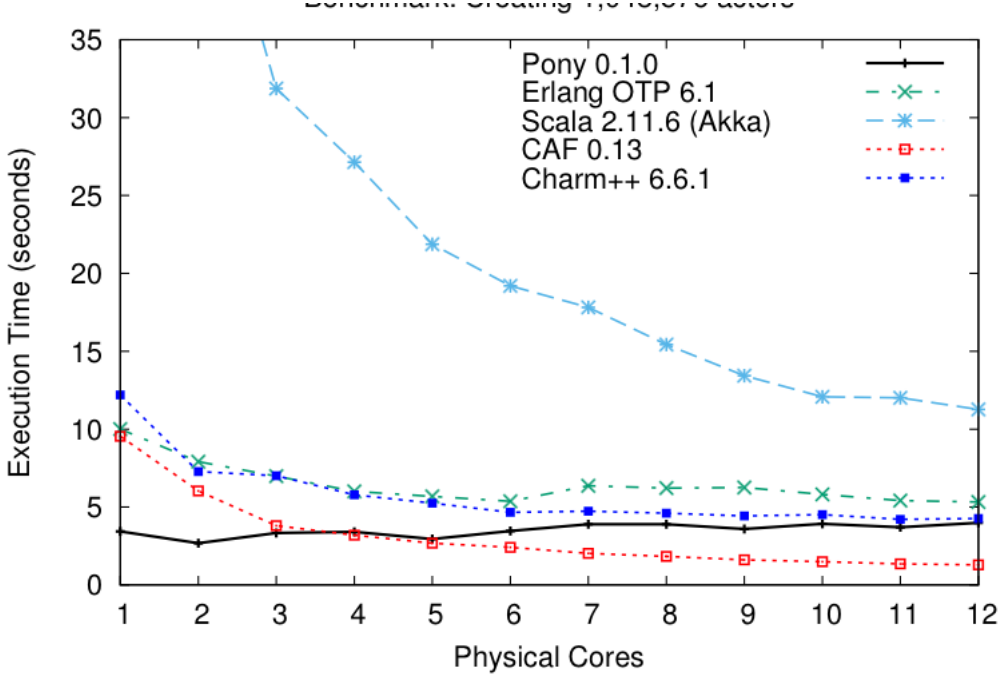


- Pony outperforms heavily optimized MPI version
- 44% source code reduction

Very fast?

Actor creation - (from CAF suite)

- Create ~1M actors
- Run on 12 core, 2.3 GHz Opteron, 64 GByte

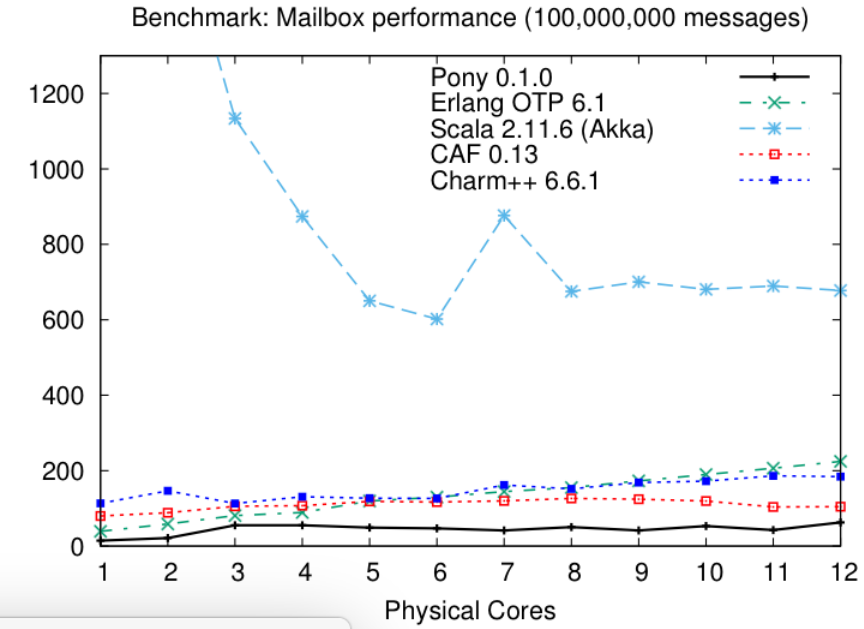


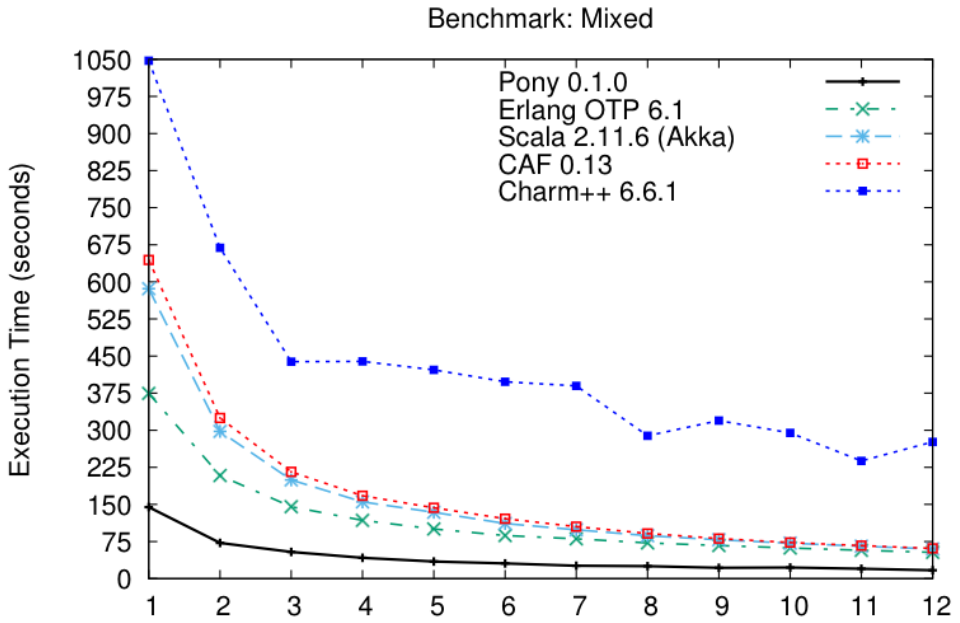
- Pony outperforms all except of CAF
- CAF does no gc

Very fast?

Mailbox and Mixed - (from CAF suite)

- Highly contended mailbox, and mixed case (factorizarion)
- Run on 12 core, 2.3 GHz Opteron, 64 GByte





- Pony outperforms all in both

easy to learn, easy to use?

- Pony API adopted as the back-end for the EU project UPSCALE; API currently used by 10 programmers
- Pony was taken up by a PhD student after a 2 hour introduction
- Pony has been unofficially taken up by another 8 programmers in the financial sector
- We have developed a 10K lines Pony standard library (incl. file handling, collections, networking, http client and server, SSL/TLS, timers and timing, random numbers)
- We have developed several “real world” analytics programs
- *Only 10.7% of types require annotations (later)*

Remaining talk – Pony language design

- Pony concurrency model
- Pony Types to prevent data races
- Pony_ORCA: Garbage Collection Objects
- Pony_AGC: Garbage Collection Actors
- Language Design Observations

Pony Concurrency Model

- Based on actor paradigm
- Actor ~ active object
- Actors/objects may send synchronous messages to actors/objects, and asynchronous messages to actors
- Asynchronous messages stored in queue; executed one at a time
- No other synchronization primitive
- Imperative Features
- Objects may be passed in messages, no copying

Pony Types to prevent data races

- capability annotations for each type;
- capabilities ~ points in a matrix of actions denied to local/global aliases

	deny global read alias	deny global write alias	Deny nothing to glob.alias

Pony Types to prevent data races

- capability annotations for each type;
- capabilities ~ points in a matrix of actions denied to local/global aliases

	deny global read alias	deny global write alias	Deny nothing to glob.alias
deny local read alias			
deny local write alias			
Deny nothing to locl.alias			

Pony Types to prevent data races

- capability annotations for each type;
- capabilities ~ points in a matrix of actions denied to local/global aliases

	deny global read alias	deny global write alias	Deny nothing to glob.alias
deny local read alias		✗	✗
deny local write alias			✗
Deny nothing to locl.alias			

Pony Types to prevent data races

	deny global read alias	deny global write alias	Deny nothing to glob.alias
deny local read alias	iso	✗	✗
deny local write alias	trn	val	✗
Deny nothing to locl.alias	ref	box	tag

Pony Types to prevent data races – what the holder may do

	deny global read alias	deny global write alias	Deny nothing to glob.alias
deny local read alias	iso	✗	✗
deny local write alias	trn	val	✗
Deny nothing to locl.alias	ref	box	tag
	Write	**Read**	**Ident**

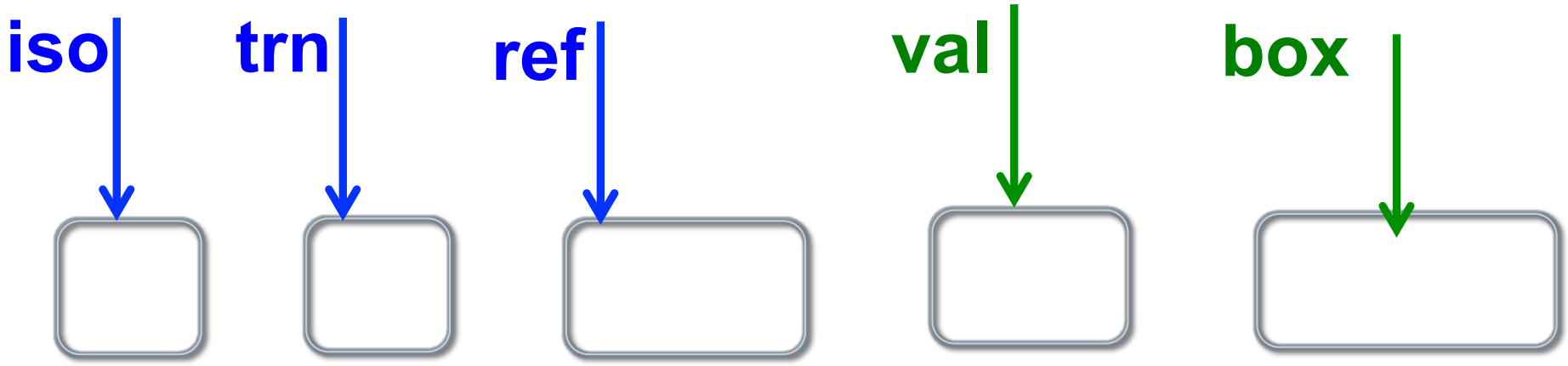
Sendable references

- iso can be sent to other actor after being consumed
- val and tag can be sent to other actors
- no copying required

	deny global read alias	deny global write alias	Deny nothing to glob.alias
deny local read alias	***iso***	✗	✗
deny local write alias	trn	***val***	✗
Deny nothing to locl.alias	ref	box	***tag***
	Writeable	Readable	Opaque

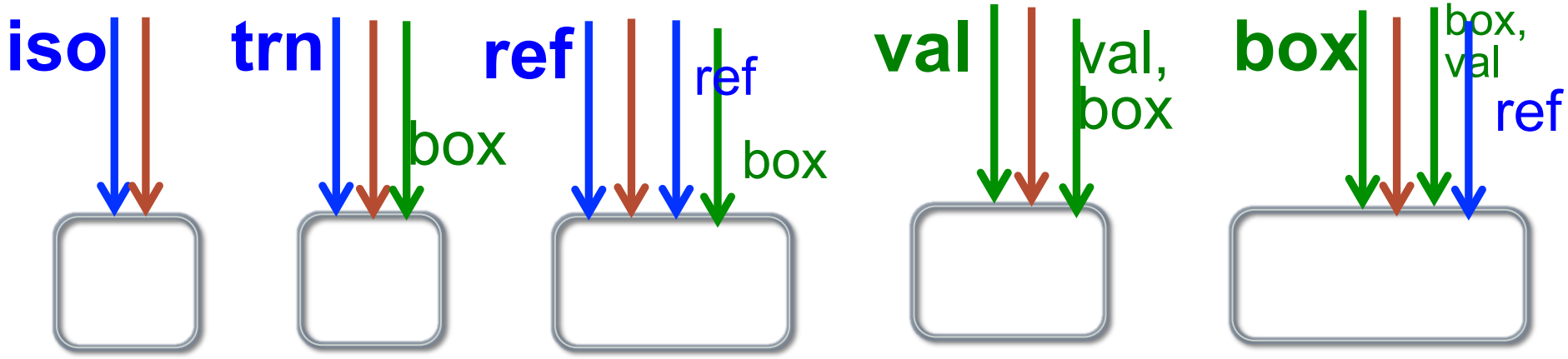
Valid local aliases

	deny global read alias	deny global write alias	Deny nothing
deny local read alias	*iso*	✗	✗
deny local write alias	trn	*val*	✗
Deny nothing	ref	box	*tag*
	Writeable	Readable	Opaque



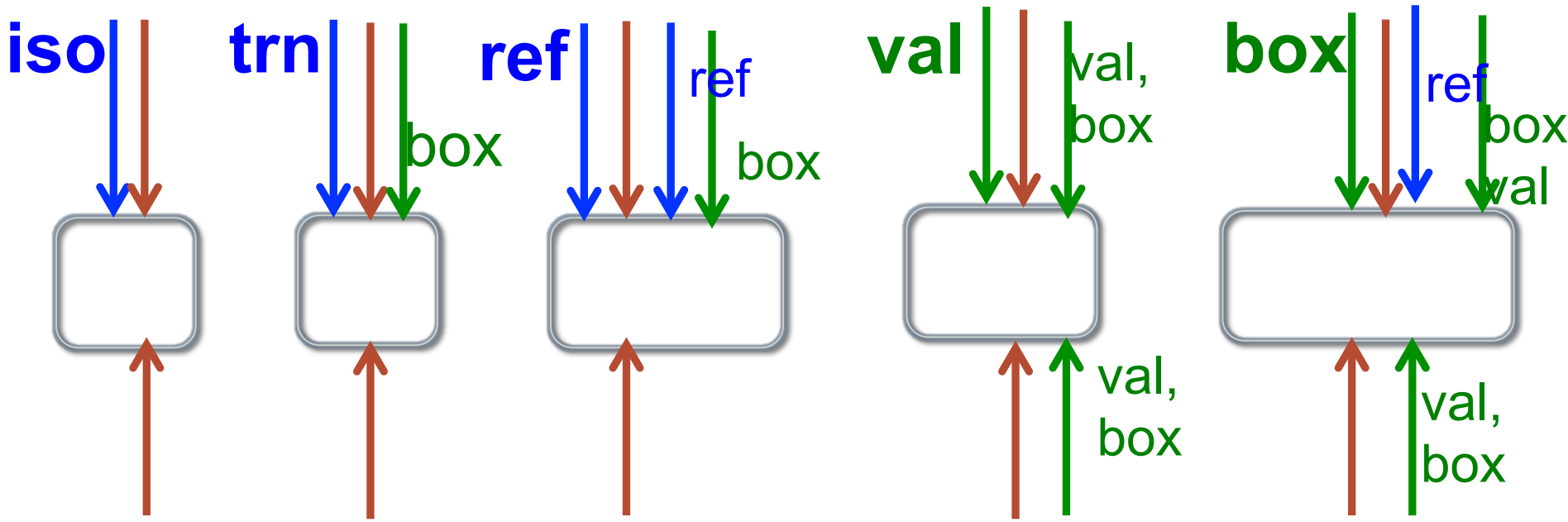
Valid local aliases

	deny global read alias	deny global write alias	Deny nothing
deny local read alias	*iso*	✗	✗
deny local write alias	trn	*val*	✗
Deny nothing	ref	box	*tag*
	Writeable	Readable	Opaque

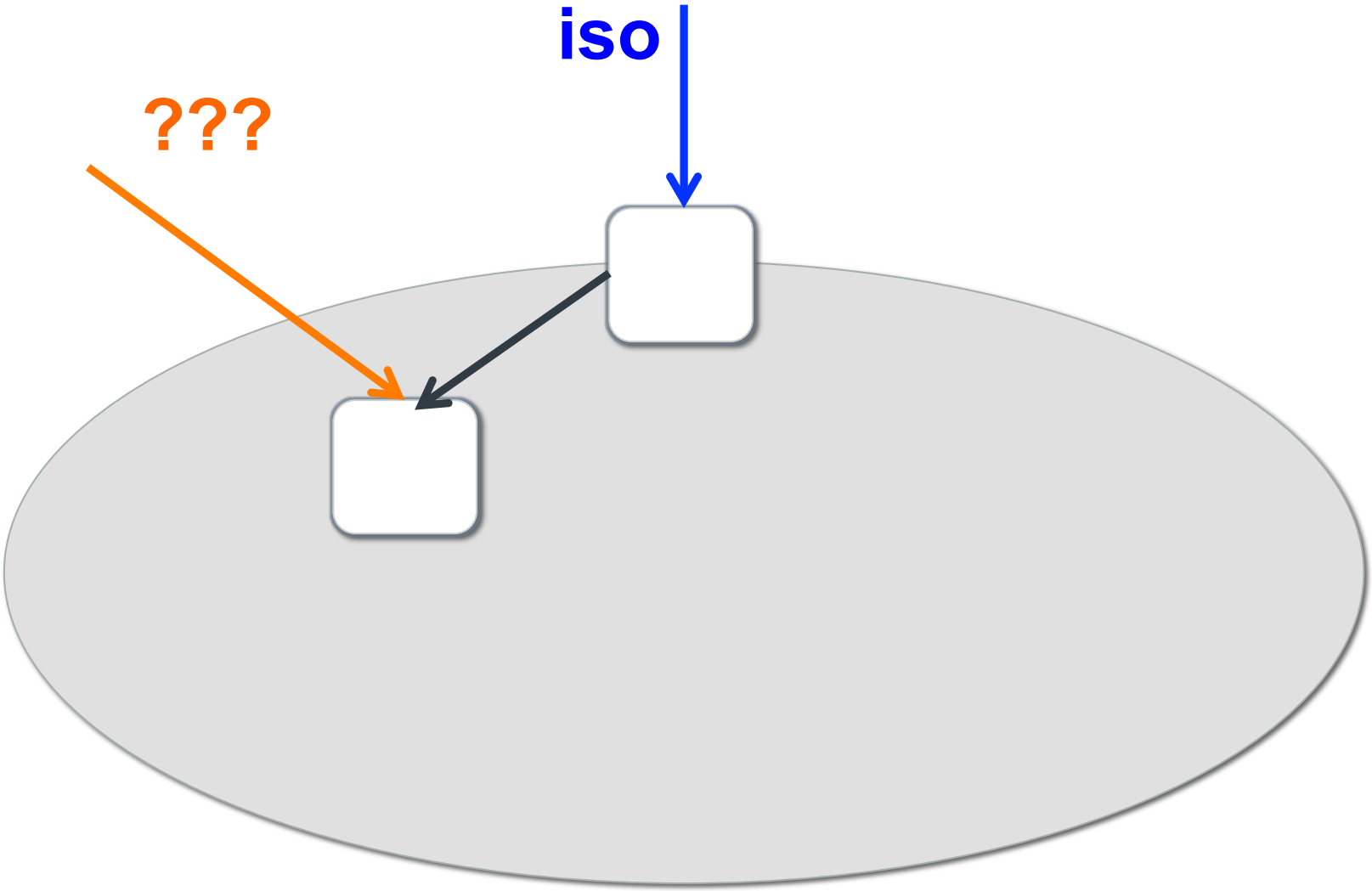


Valid local, and **global** aliases

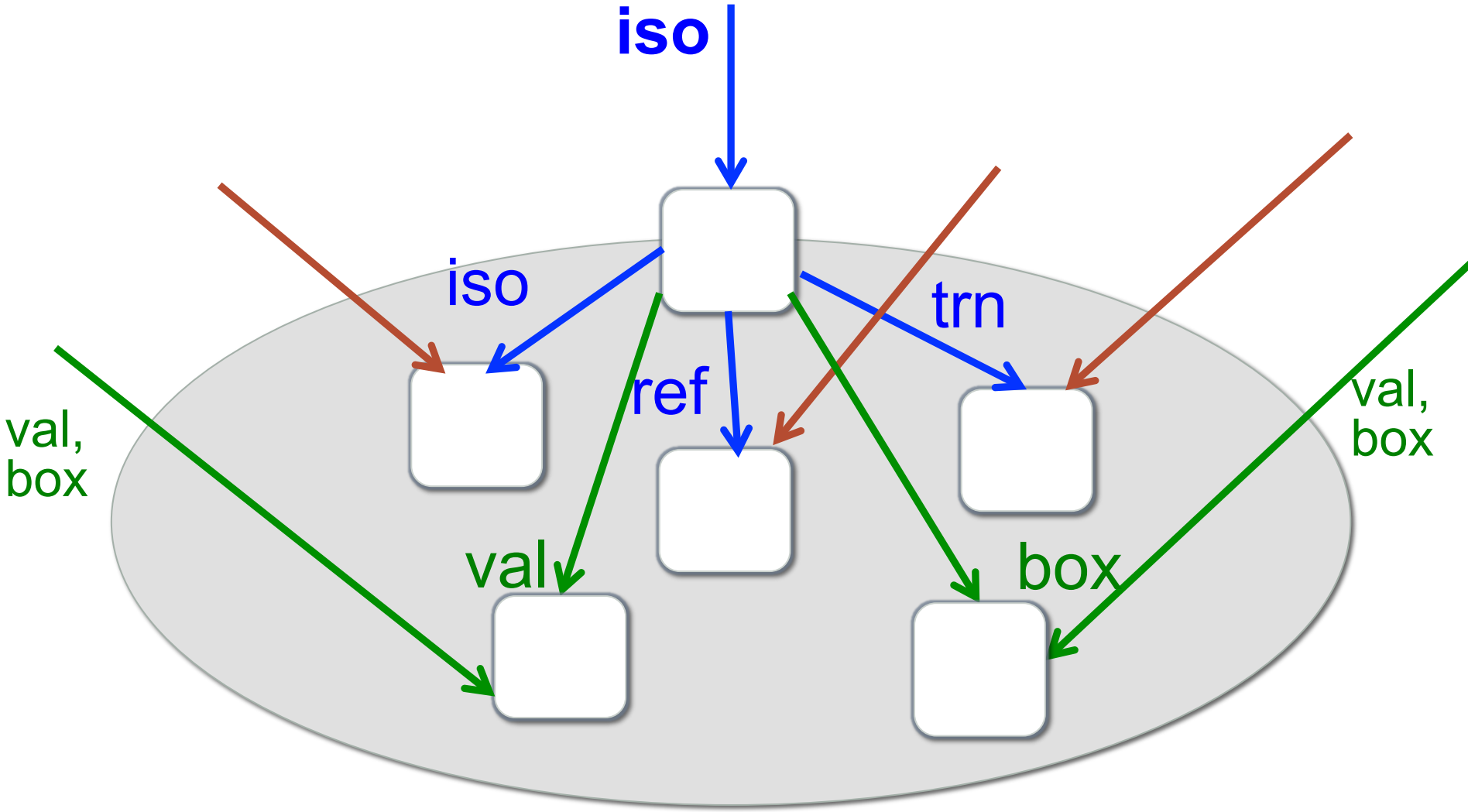
	deny global read alias	deny global write alias	Deny nothing
deny local read alias	*iso*	✗	✗
deny local write alias	trn	*val*	✗
Deny nothing	ref	box	*tag*
	Writeable	Readable	Opaque



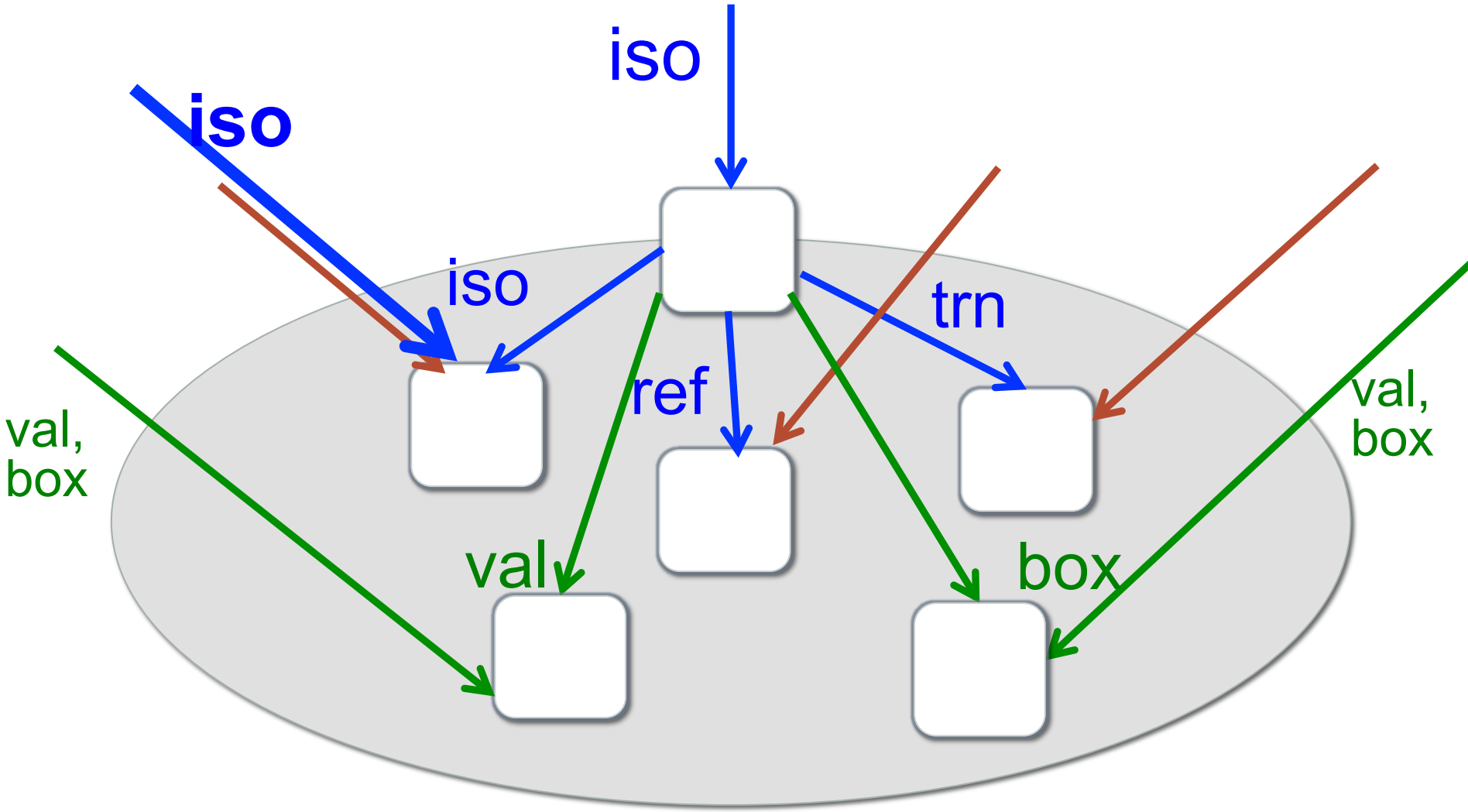
Permanent alias inside an **iso**-bubble



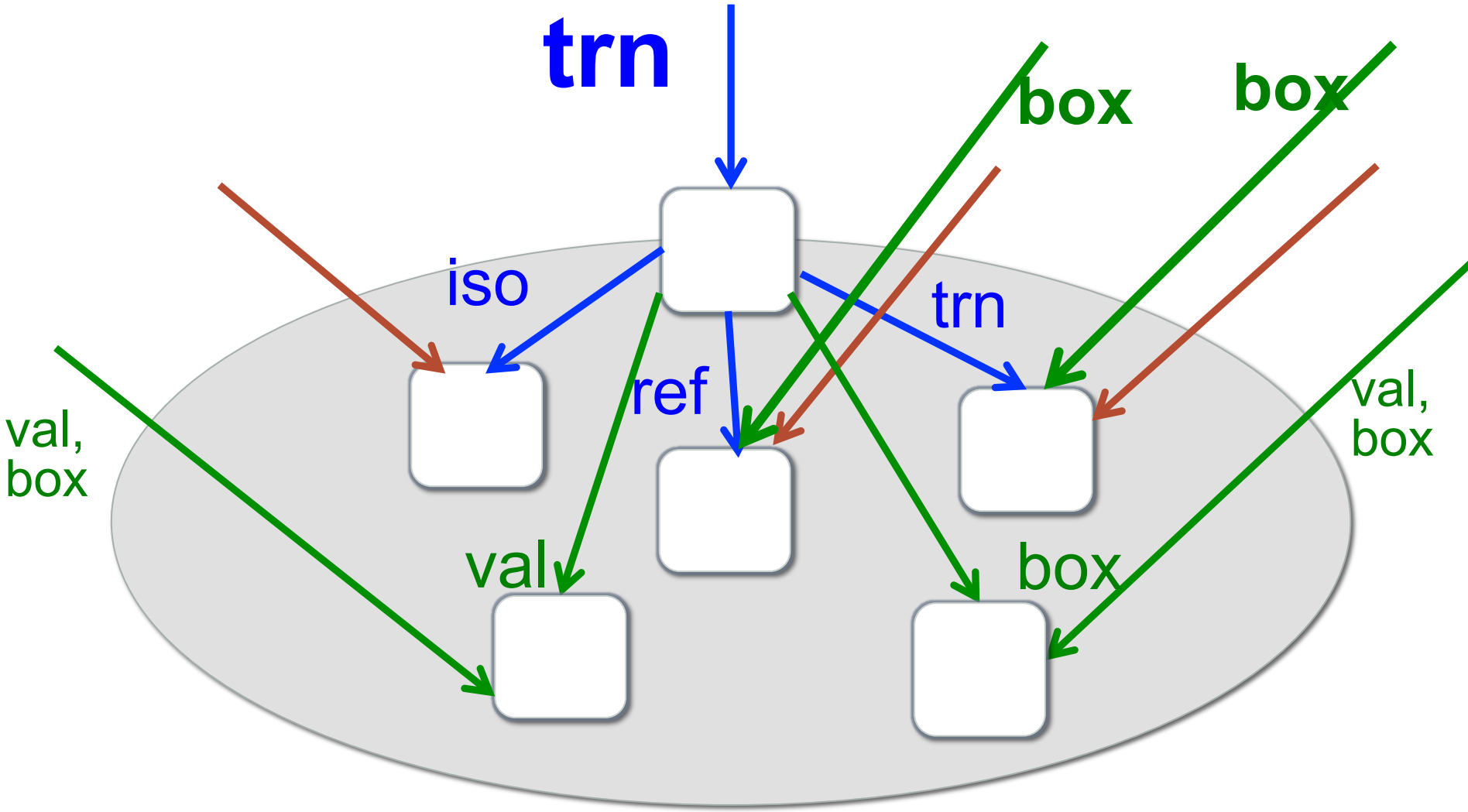
Permanent alias inside an **iso**-bubble



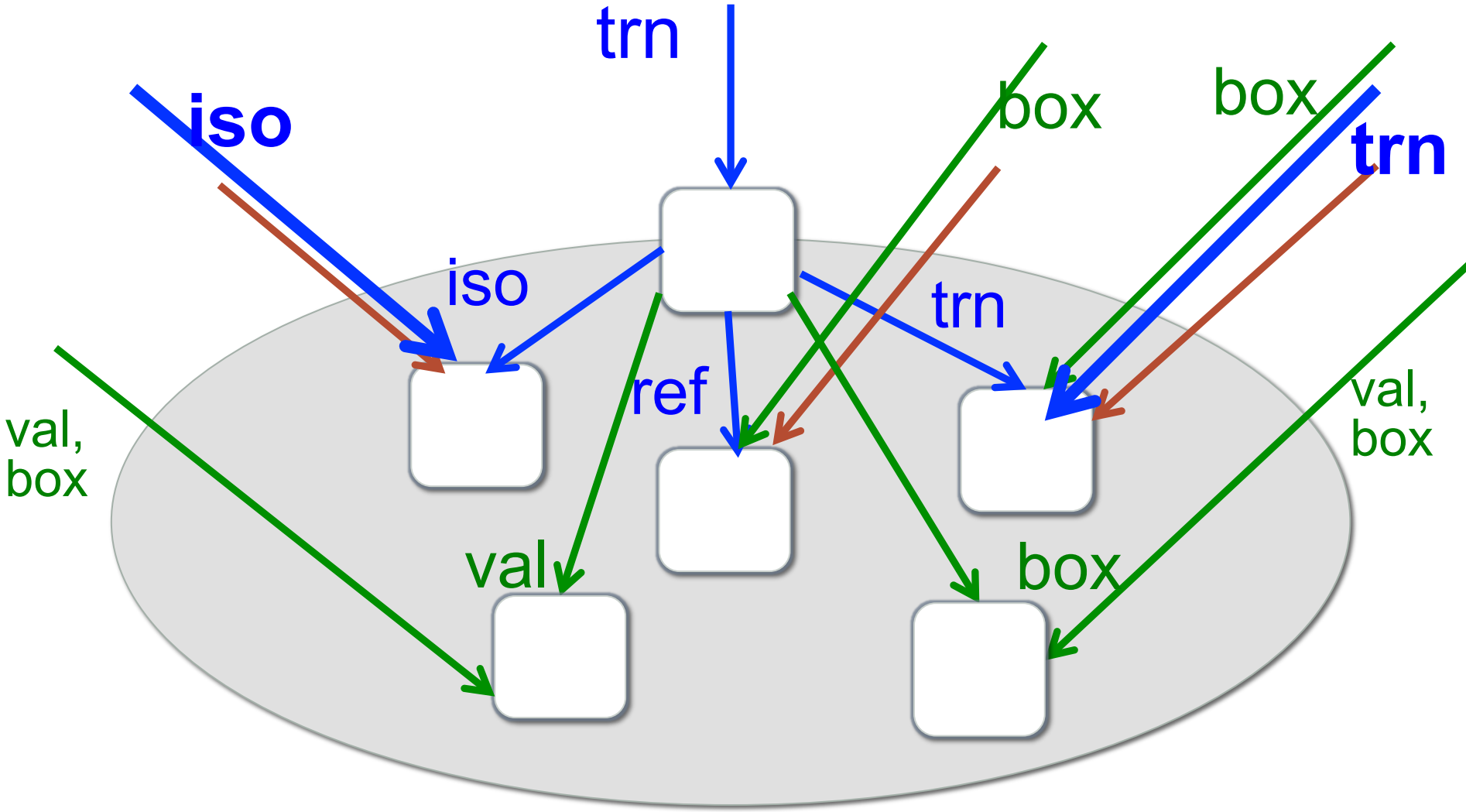
Permanent and **temporary** alias inside a iso-bubble



Permanent alias inside a **trn**-bubble



Permanent and **temporary** alias inside a trn-bubble



Pony type system

- Further issues
  - Recover expressions
  - Type if field read (viewpoint adaptation)
  - Type of method call
  - Type of field write
  - Proofs
- More at
  http://www.doc.ic.ac.uk/~scd/fast-cheap.pdf
- More features – implemented
  - *Algebraic types and Pattern Matching*
  - *Generics*
  - *Non-null*
- More features – perhaps
  - *Reflection*

Pony ORCA: Object Garbage Collection

- Actors perform GC fully concurrenty.
- Objects are owned by the actor which created them.
- Actor garbage collect owned objects.
- Challenge: objects may be accessible from other actors
- Approach: Weighted deferred reference counts: actor keep reference count for a) owned objects reachable from other actors, b) objects owned by other actors and reachable from the actor.

Pony ORCA: Object Garbage Collection.2

- Actors (α) , Object (o), owning actor (Own)
- Actor keep reference count for a) owned objects reachable from other actors, b) objects owned by other actors and reachable from the actor.
- RC(Own(o),o)

  $= \sum_{\alpha \neq Own(o)} RC(\alpha,o) +\# \{(\alpha,i) \mid Mssg(a,i) \text{ reaches } o \}$
- When sending/receiving messages, trace, and for all reachable objects adjust RC accordingly.

- When tracing, and non-owned object becomes inaccessible

Pony ORCA: Object Garbage Collection.2

- Actors (α) , Object (o), owning actor (Own)
- Actor keep reference count for a) owned objects reachable from other actors, b) objects owned by other actors and reachable from the actor.
- RC(Own(o),o) + Σm_i where Mssg(Own(o),i)=INC(m,o)
  $= \Sigma_{\alpha \neq Own(o)} RC(\alpha,o) +\# \{(\alpha,i) \mid Mssg(a,i) \text{ reaches } o \}$
- When sending/receiving messages, trace, and for all reachable objects o adjust RC accordingly. If RC gets to 0 then add k to own, and send to INC(k+1,o) to Onw(o).
- When tracing, and non-owned object becomes inaccessible, set own RC to 0, and send INC(-old(RC),o) to Onw(o).

Pony ORCA: Object Garbage Collection.3

- Tracing functions using the type system, and therefore no race conditions
- Used the actor paradigm (messages) to implement garbage collection.
- Used causal message delivery for Garbage Collection.
- Object Reference cycles not an issue.

- More at

  http://www.doc.ic.ac.uk/~scd/ogc.pdf

Pony Actor Collection

- Actor collection requires detection of Actor Cycles.
- This is done by a separate actor – the cycle detector.
- Actors keep a count of all external referring actors, and the set of actors they refer to.
- When blocking, they send this information to the cycle detector.
- This information is sufficient for the cycles detector to identify, and remove cycles.

Pony Actor Collection.2

- This is done by a separate actor – the cycle detector.
- Actors keep a count of all external referring actors, and the set of actors they refer to.
- When blocking, they send this information to the cycle detector.
- But, actors' views of themselves may be out of sync with "real" state (another actor drops/gains reference to them).
- Also, cycle detector's view of actor may be out of sync with actor's view (a blocked actor may wake up).

Pony Actor Collection.3

- This is done by a separate actor – the cycle detector.
- Actors keep a count of all external referring actors, and the set of actors they refer to.
- When blocking, they send this information to the cycle detector.
- But, actors' views of themselves may be out of sync with "real" state (another actor drops/gains reference to them). In this case, send appropriate INC/DEC messages.
- Also, cycle detector's view of actor may be out of sync with actor's view (a blocked actor may wake up). The cycle detector sends ACK, and awaits CONF messages from blocked actors.

Pony Actor Collection.4

- Adaptation of reference counting protocol.
- Used the actor paradigm.
- Causal Message Delivery a prerequisite.
- Fun Proof.
- More at

  S. Clebsch and S. Drossopoulou.
  Fully concurrent garbage collection of actors on many-core machines. OOPLSA 2013.

More on Pony

- Native code compiler
- Pony actor runtime, including scheduler, memory allocator, message queues, and garbage collector
- Debugger
- Several applications developed
- Implementation fast; language relatively easy to learn/use
- Even though model is subtle, language feels simple
- Good defaults and good error messages are crucial
- Tutorial at http://causalityltd.github.io/pony-tutorial/
- Sandbox implementation at http://sandbox.ponylang.org
- **Cheers, Jeers, Help, Collaboration, Feedback, welcome**

Language Design Conclusions

- Actor Paradigm applied to the GC problem; messages sent when an actor's view of world out of sync with the state of world
- Data Race freedom, Causal Message Delivery prerequisites for Garbage Collection – and distribution.

- Starting from Principles Delivers
- Formal Models Deliver
- Finding the Invariant is key
- Finding ways not to think of time is key