# Type Systems For Distributed Data Structures

Ben Liblit & Alexander Aiken

University of California, Berkeley

# Underlying Memory Model

- Multiple machines, each with local memory
- Global memory is union of local memories
- Distinguish two types of pointers:
  - **Local** points to local memory only
  - **Global** points anywhere: ⟨machine, address⟩
  - Different representations & operations

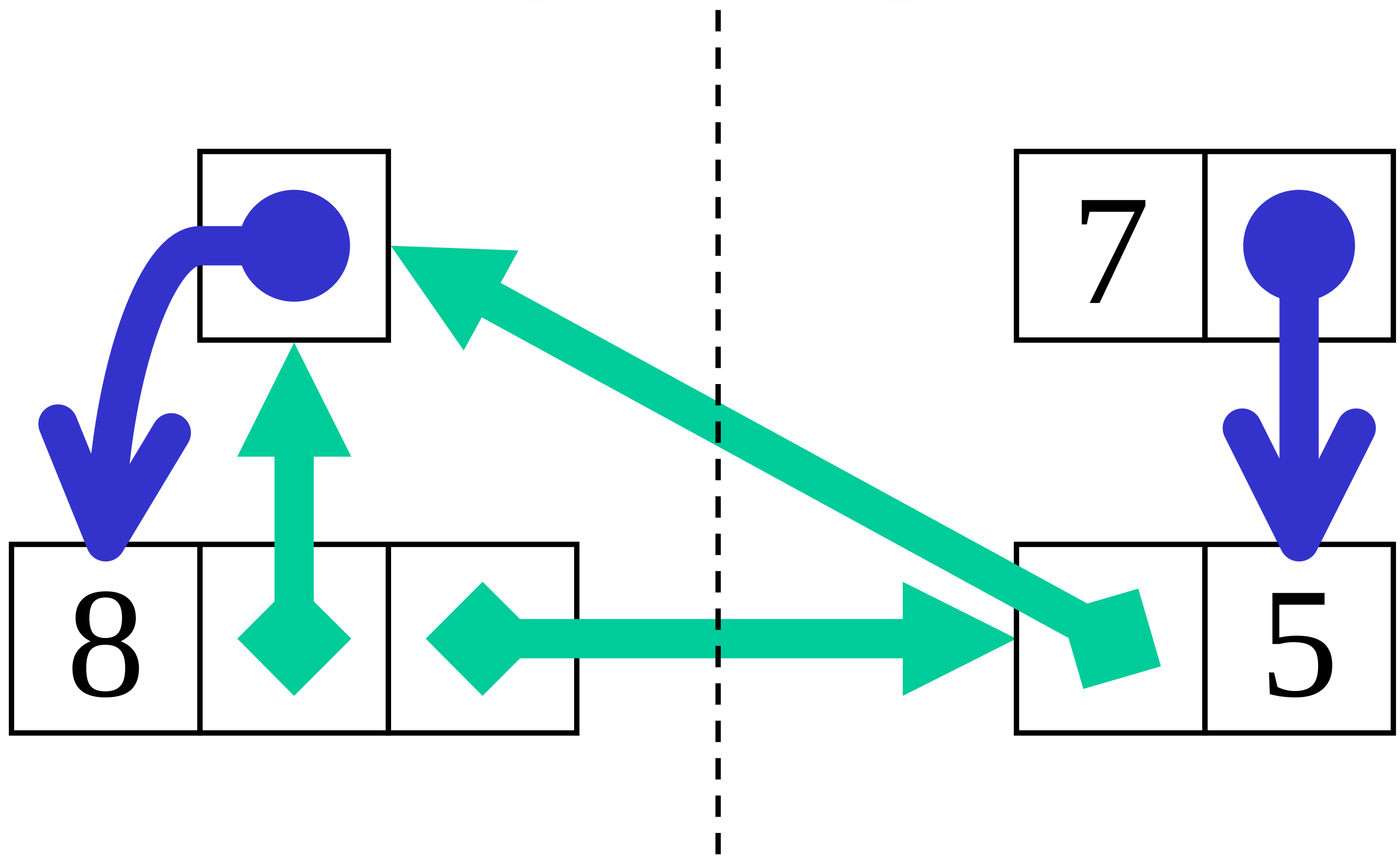
# Language Design Options

- Make everything global?
  - ✓ Conservatively sound
  - ✓ Easy to use
  - ✗ Hides program structure
  - ✗ Needlessly slow

# Language Design Options

- Expose local/global to programmer?
  - ✓Explicit cost model
  - ✓Faster execution
  - ✗ Naïve designs are unsound (as we will show)
  - ✗ Code becomes difficult to write and maintain
  - ✗ Conversion of sequential code is problematic

# A (Possibly) Gratuitous Global
# A (Potentially) Unsound Local



# Understand It First, Then Fix It

- Study local/global in a simpler context
  - Design a *sound* type system for a tiny language
- Move from type checking to type inference
  - Programmers see as much detail as they want
- Apply findings to design of real languages
  - Type system detects & forbids “bad things”
  - Local qualification inference as optimization

# Type Grammar

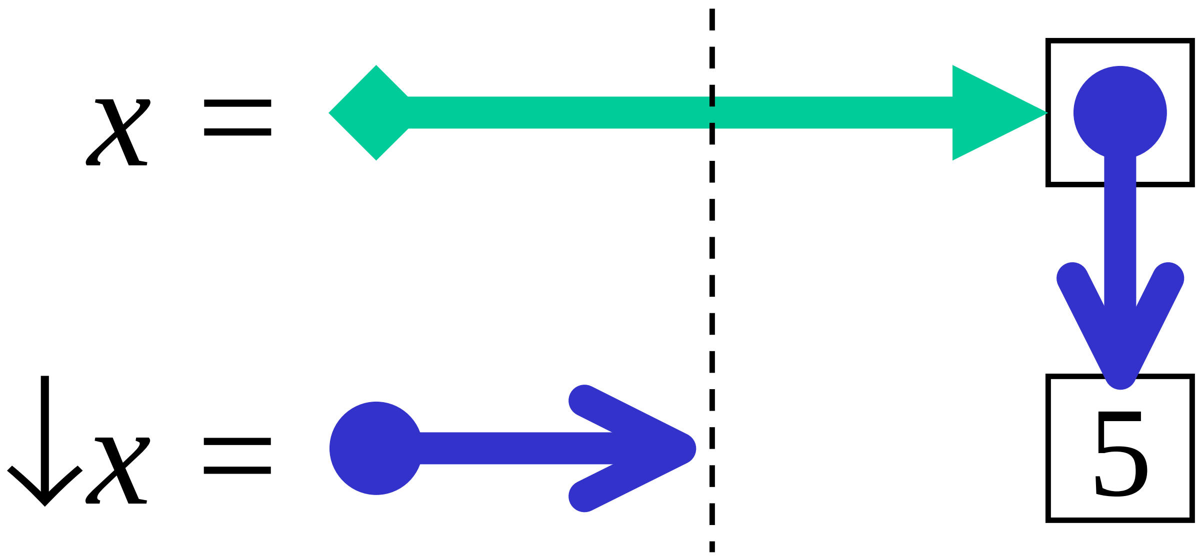
$$\omega ::= \text{local} \mid \text{global}$$

$$\tau ::= \text{int} \mid \text{boxed}\,\omega\,\tau \mid \tau \times \tau$$

- Boxed and unboxed values
- Integers, pointers, and pairs
  - Pairs are *not* assumed boxed
- References to boxes are either local or global

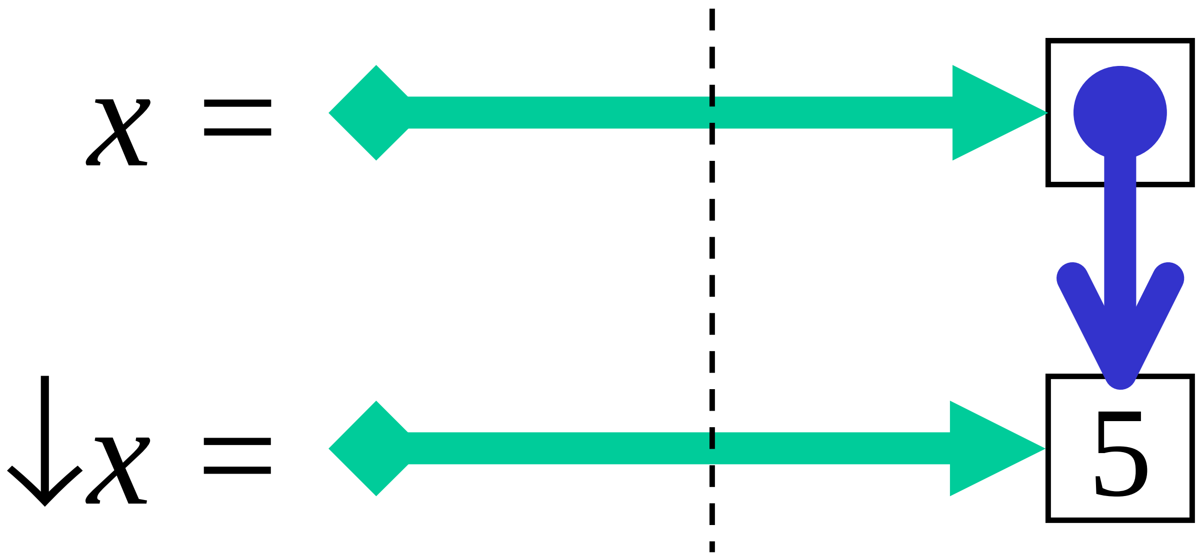
# Global Dereferencing: Standard Approach Unsound

$$\frac{x : \text{boxed global } \tau}{\downarrow x : \tau}, \text{ where } \tau = \text{boxed local int}$$



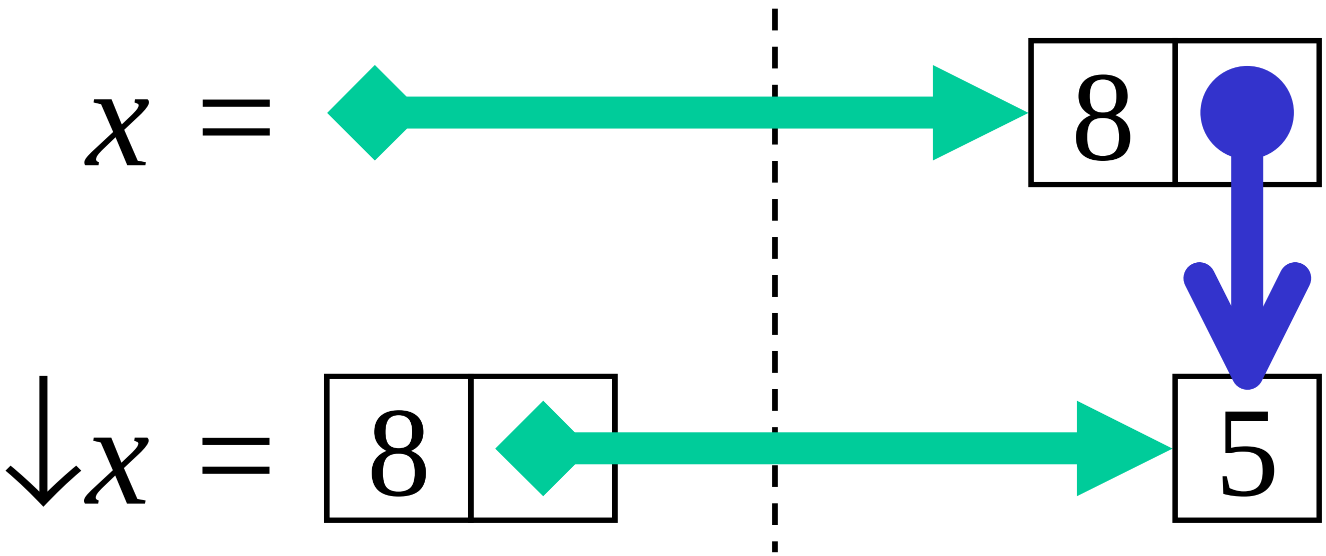
# Global Dereferencing: Sound With Type Expansion

$$\frac{x : \text{boxed global } \tau}{\downarrow x : \text{expand}(\tau)}$$



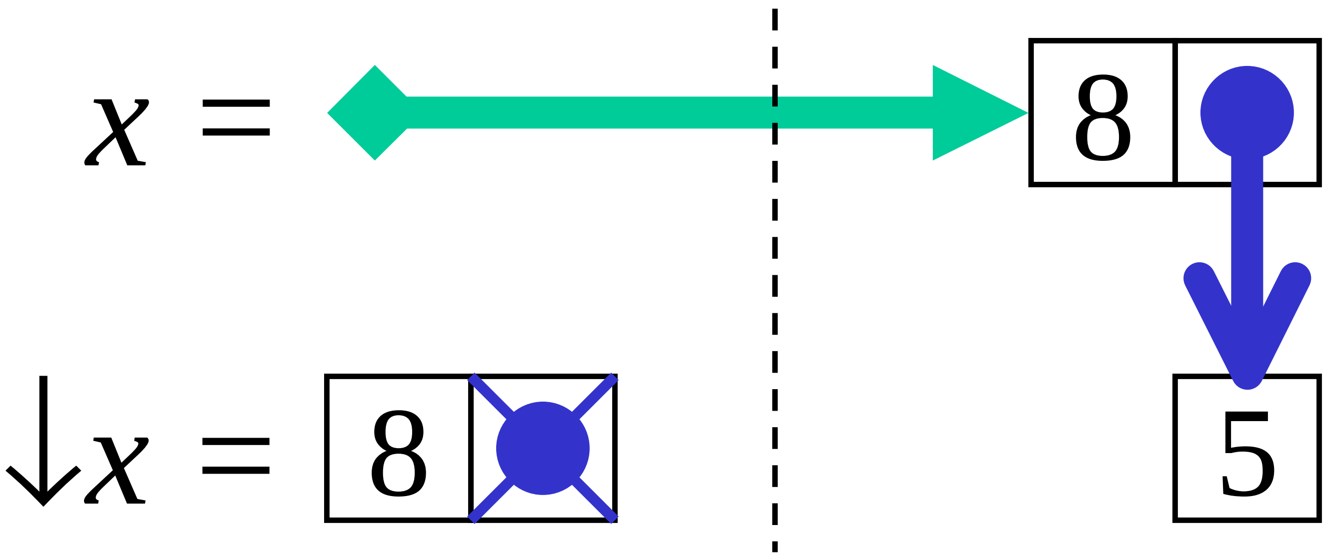
# Global Dereferencing: Tuple Expansion

- Type expansion for tuple components?
- No: would change representation of tuple



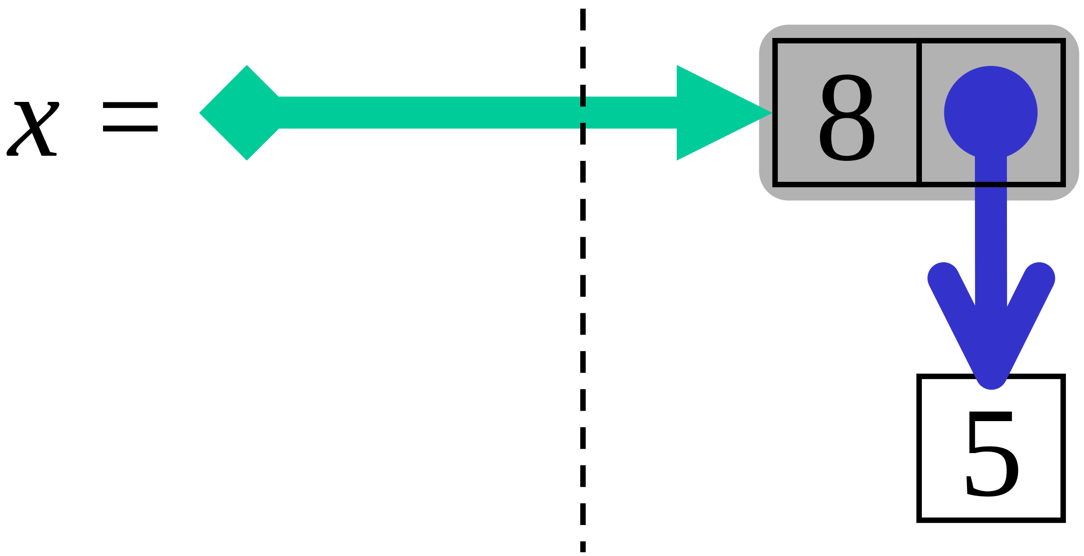
# Global Dereferencing: Tuple Expansion

- Solution: Invalidate local pointers in tuples
- Other components remain valid, usable



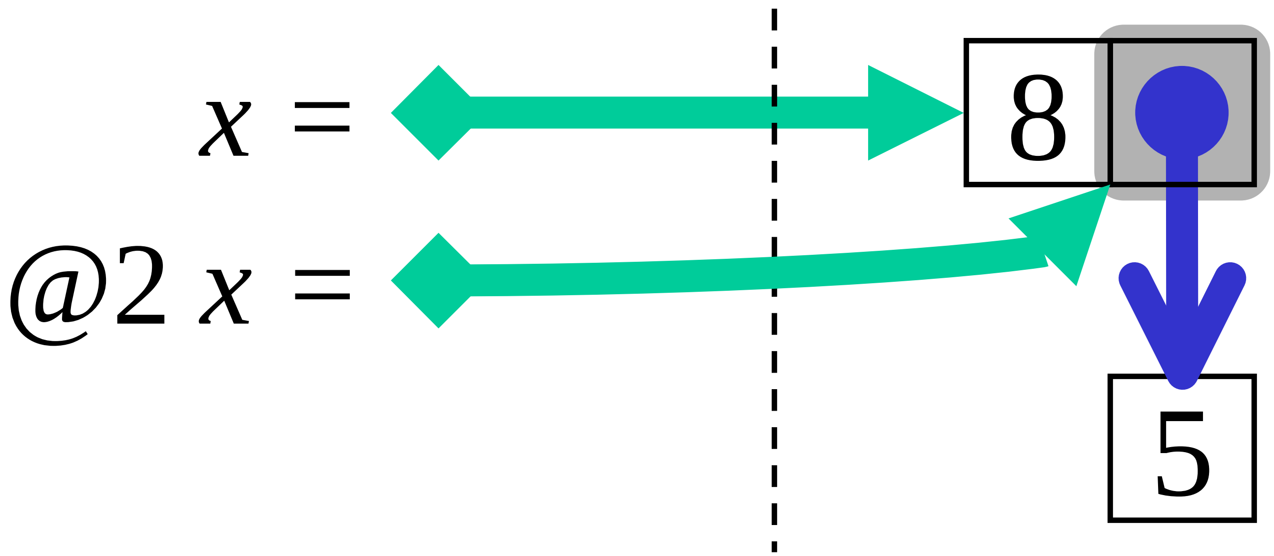
# Global Tuple Selection

- Starting at *x*, can we reach 5?
- Yes, with a proper selection operator



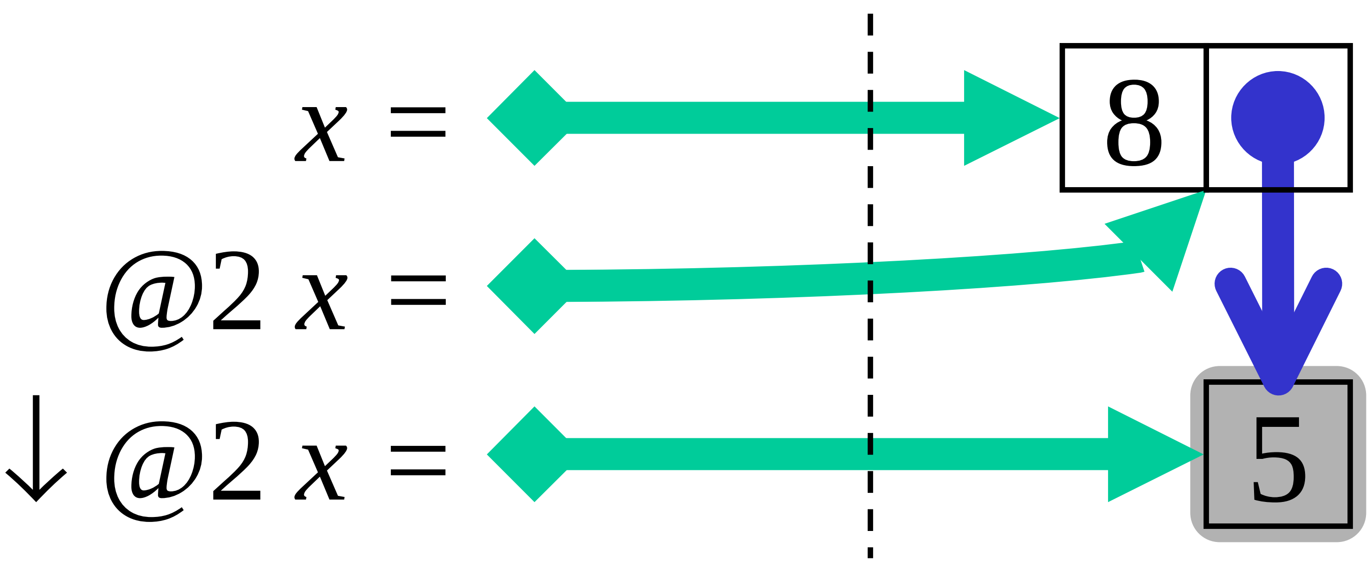
# Global Tuple Selection

- Selection offsets pointer *within* tuple



# Global Tuple Selection

- Selection offsets pointer *within* tuple
- Global-to-local pointer works just as before



# Extended Type Grammar

$$\omega ::= \text{local} \mid \text{global}$$

$$\rho ::= \text{valid} \mid \text{invalid}$$

$$\tau ::= \text{int} \mid \text{boxed}\,\omega\,\rho\,\tau \mid \tau \times \tau$$

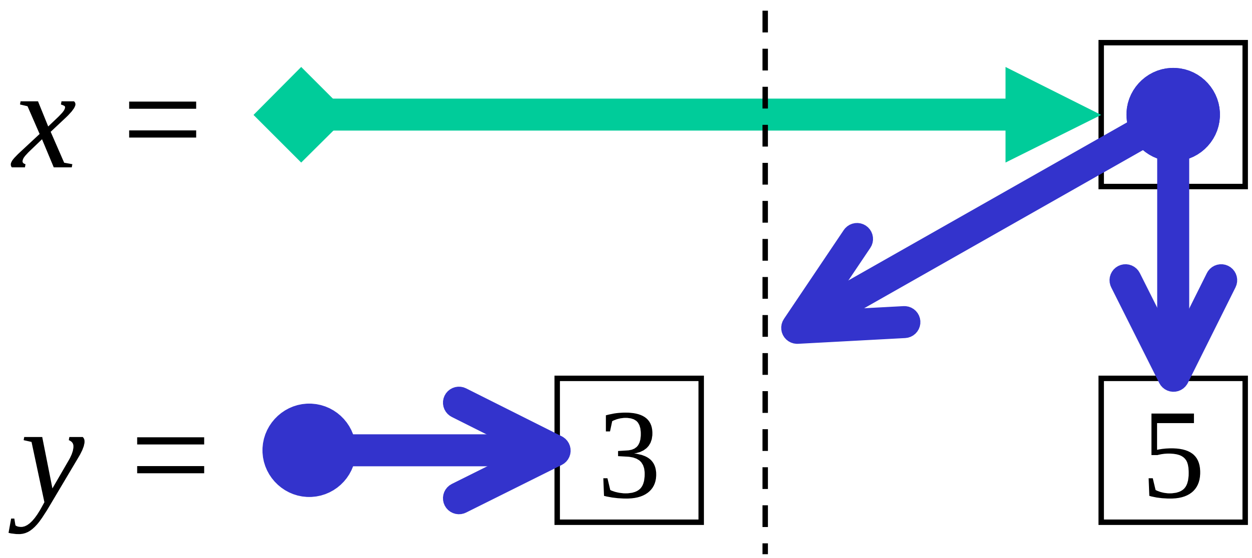
- Allow subtyping on validity qualifiers

$$\text{boxed}\,\omega\,\text{valid}\,\tau \le \text{boxed}\,\omega\,\text{invalid}\,\tau$$

$$\tau_1 \times \tau_2 \le \tau_3 \times \tau_4 \quad \Leftrightarrow \quad \tau_1 \le \tau_3 \ \wedge \ \tau_2 \le \tau_4$$

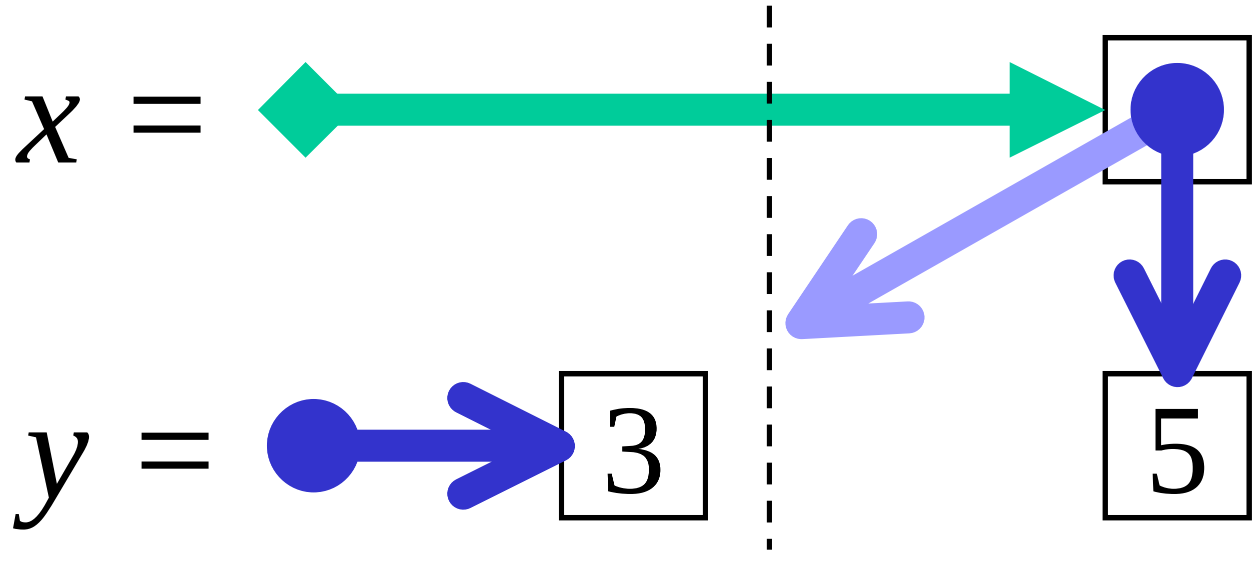
# Global Assignment

$$\frac{x : \text{boxed valid global } \tau \qquad y : \tau}{x := y : \tau}$$



# Global Assignment

$$\frac{\begin{array}{c} x : \text{boxed valid global } \tau \\ y : \tau \qquad \text{expand}(\tau) = \tau \end{array}}{x := y : \tau}$$



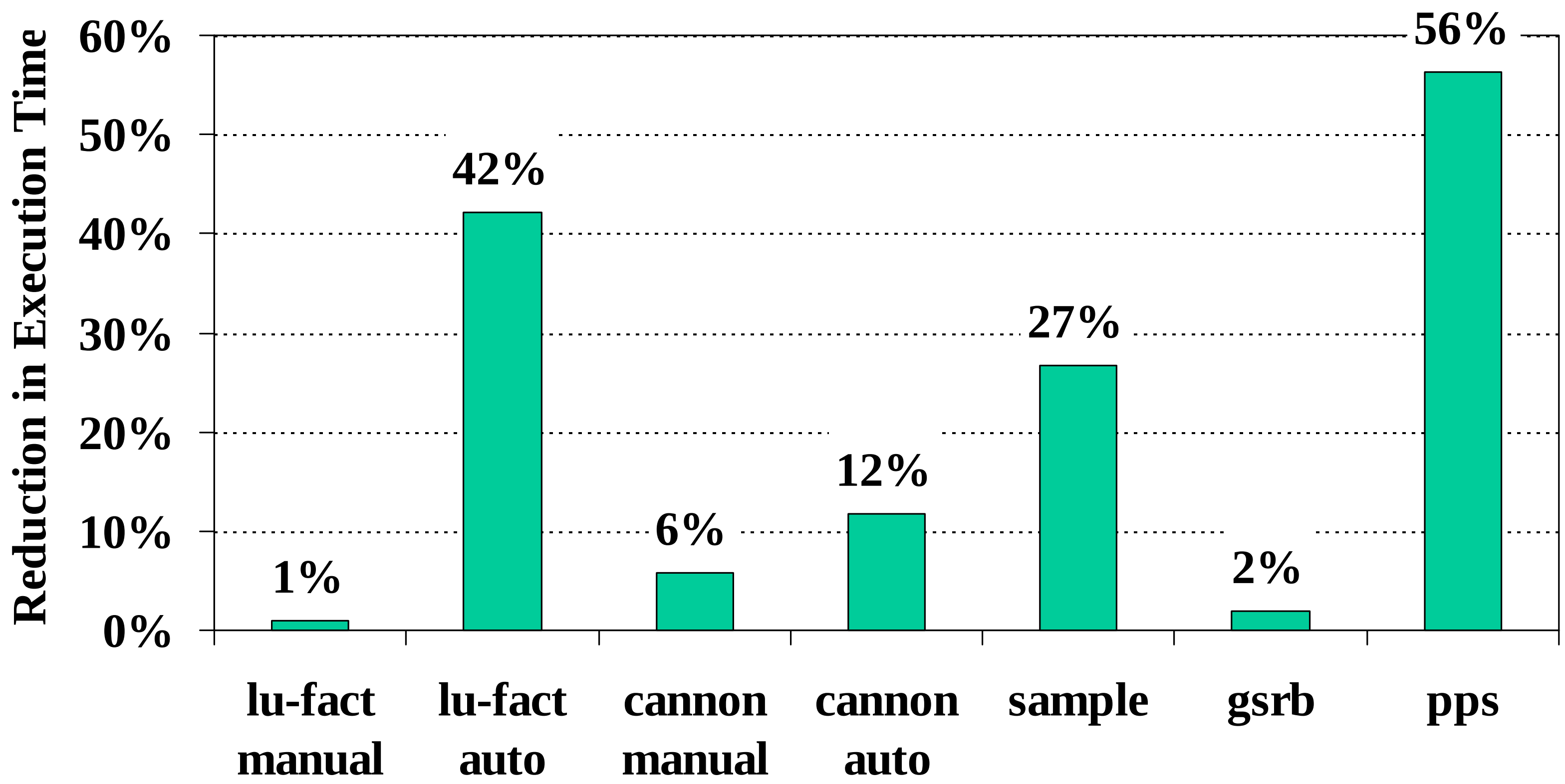
# Type Qualifier Inference

- Efficiently infer qualifiers in two passes:
  1. Maximize number of “invalid” qualifiers
  2. Maximize number of “local” qualifiers
- Allows for a range of language designs
  – Complete inference
  – Allow explicit declarations as needed
- On large codes, does better than humans!

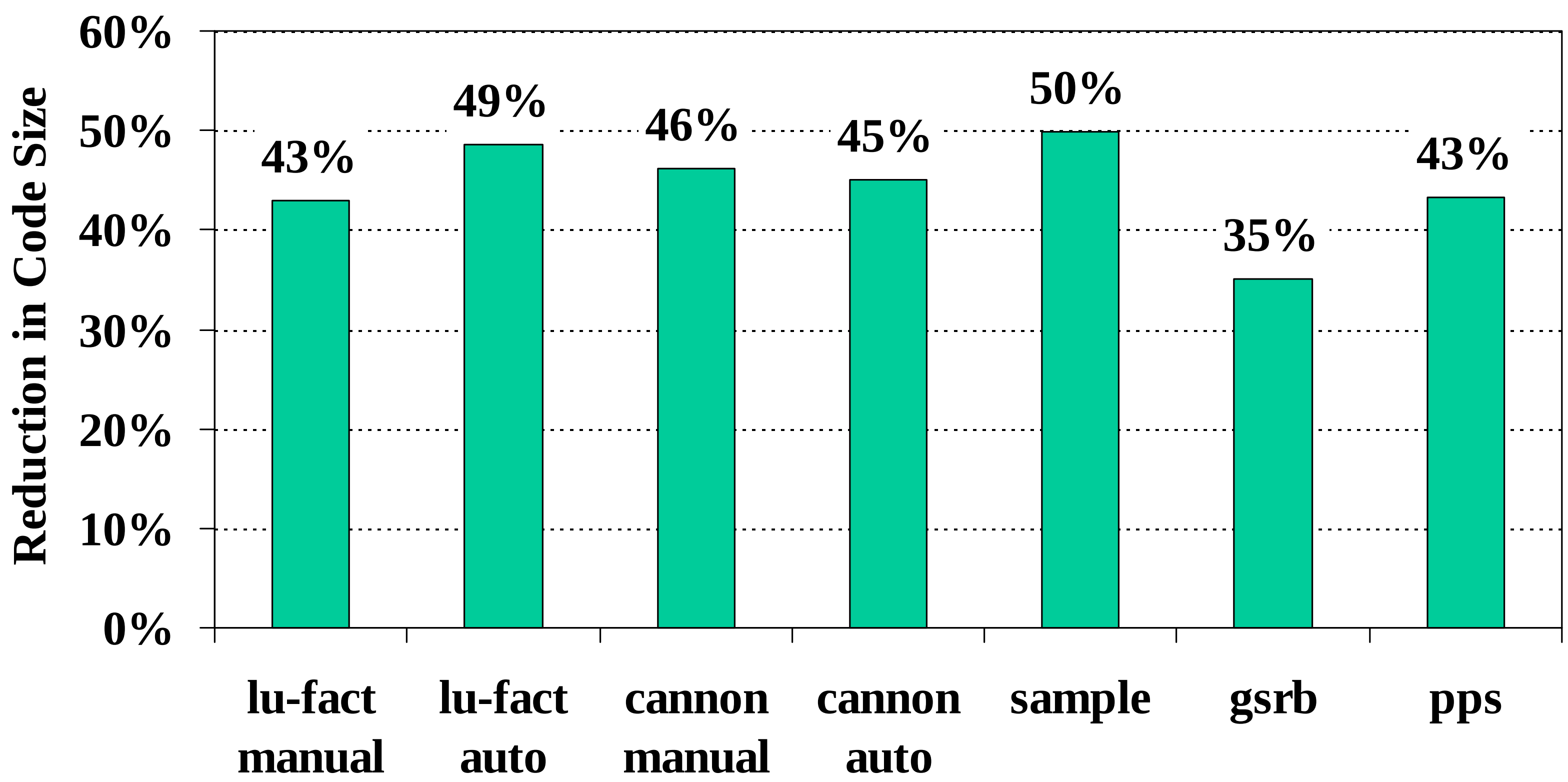
# Titanium Implementation

- Titanium = Java + SPMD parallelism
  - Focus is on scientific codes
- Global is assumed; local is explicit
  - E.g., “Object local” or “double [] local [] local”
- Local qualification inference in compiler
  - Conservative for valid/invalid qualifiers
  - Monomorphic

# Titanium Benchmarks: Speed



# Titanium Benchmarks: Code Size



# Summary and Conclusions

- For top performance, local/global *must* be dealt with
- Soundness issues are subtle, but tractable
  - Analysis core is surprisingly simple
- Type qualifier inference is a double win:
  - Programming is easier
  - Optimized code is faster, smaller