From POPL to the Jungle and back

Peter Sewell

University of Cambridge

PLMW: the SIGPLAN Programming Languages Mentoring Workshop

San Diego, January 2014

POPL 2014

41th ACM SIGPLAN-SIGACT Symposium

on

Principles of Programming Languages

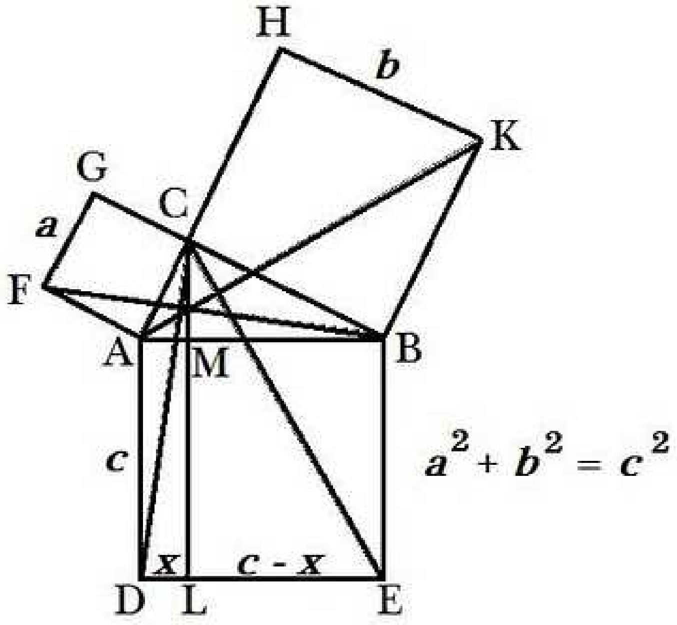
How can we reconcile the two?

Four styles of research — each with their pros and cons
(all can be good or bad...)

Option 1: *Principles* of Programming Languages

Ignore that legacy infrastructure altogether

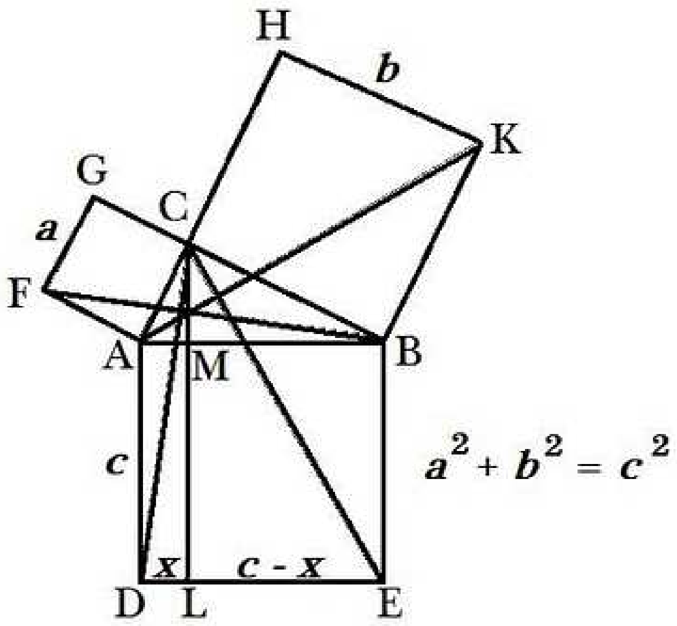
Do some beautiful mathematics



Option 1: *Principles* of Programming Languages

Ignore that legacy infrastructure altogether

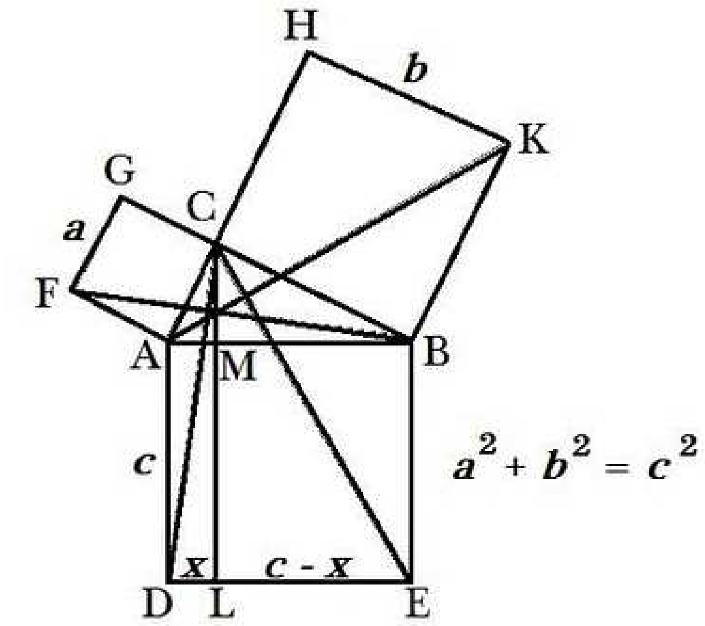
Do some beautiful $\left\{ \begin{matrix} \text{Fundamental} \\ \text{Irrelevant} \end{matrix} \right\}$ mathematics



Option 1: *Principles* of Programming Languages

Ignore that legacy infrastructure altogether

Do some beautiful $\left\{\begin{array}{l}\text{Foundational?}\\ \text{Irrelevant}\end{array}\right\}$ mathematics



Option 2: *Principled* Programming Languages

How would we rebuild that infrastructure *right?*

Principled Programming

Option 2: *Principled* Programming Languages

How would we rebuild that infrastructure *right?*

Principled Programming

(ignoring constraints of engineering effort and currently available skills...)

Option 3: Principles of *Fragments of Hypothetical* Programming Languages (or Analysis Tools)

1. Pick some specific issue from practice
2. Propose a solution
3. Work it out in the context of a small calculus

Option 3: Principles of *Fragments of Hypothetical* Programming Languages (or Analysis Tools)

1. Pick some specific issue from practice
2. Propose a solution
3. Work it out in the context of a small calculus
4. Build a prototype implementation with no formal connection to that calculus

Option 3: Principles of *Fragments of Hypothetical* Programming Languages (or Analysis Tools)

1. Pick some specific issue *arguably* from practice
2. Propose a solution
3. Work it out in the context of a small calculus
4. Build a prototype implementation with no formal connection to that calculus

(hope it will catch on in some future full-scale language design)

Option 4: *As-principled-as-you-can-manage* Approach to *Mainstream* Programming Languages

Take (some aspect of) the legacy infrastructure seriously.
Figure out how to do *something* rigorous+useful with it.
Accept it may not be beautiful

Option 4: *As-principled-as-you-can-manage* Approach to *Mainstream* Programming Languages

Take (some aspect of) the legacy infrastructure seriously.
Figure out how to do *something* rigorous+useful with it.

~~Accept it may not be beautiful~~

p. 10

(hope you manage to get somewhere before going mad)

Example: Type-safe Distributed FP

(example of Option 2/3: Principles of Fragments of Hypothetical Programming Languages)

POPL 2001: Module system for distributed abstract types

Example: Type-safe Distributed FP

(example of Option 2/3: Principles of Fragments of Hypothetical Programming Languages)

POPL 2001: Module system for distributed abstract types

ICFP 2003: λ-Calculi with marshalling and dynamic update

Leifer, Peskine, Wansbrough; Bierman, Hicks, Stoyle

Example: Type-safe Distributed FP

(example of Option 2/3: Principles of Fragments of Hypothetical Programming Languages)

POPL 2001: Module system for distributed abstract types

ICFP 2003: λ-Calculi with marshalling and dynamic update

Leifer, Peskine, Wansbrough; Bierman, Hicks, Stoyle

ICFP 2005: Acute Language

Leifer, Wansbrough, Zappa Nardelli, Vafeiadis, ...

Example: Type-safe Distributed FP

(example of Option 2/3: Principles of Fragments of Hypothetical Programming Languages)

POPL 2001: Module system for distributed abstract types

ICFP 2003: λ-Calculi with marshalling and dynamic update

Leifer, Peskine, Wansbrough; Bierman, Hicks, Stoyle

ICFP 2005: Acute Language

Leifer, Wansbrough, Zappa Nardelli, Vafeiadis, ...

ML Workshop 2006: HashCaml

Shinwell, Billings, Strniša

Example: Type-safe Distributed FP

(example of Option 2/3: Principles of Fragments of Hypothetical Programming Languages)

POPL 2001: Module system for distributed abstract types

ICFP 2003: λ-Calculi with marshalling and dynamic update
Leifer, Peskine, Wansbrough; Bierman, Hicks, Stoyle

ICFP 2005: Acute Language
Leifer, Wansbrough, Zappa Nardelli, Vafeiadis, ...

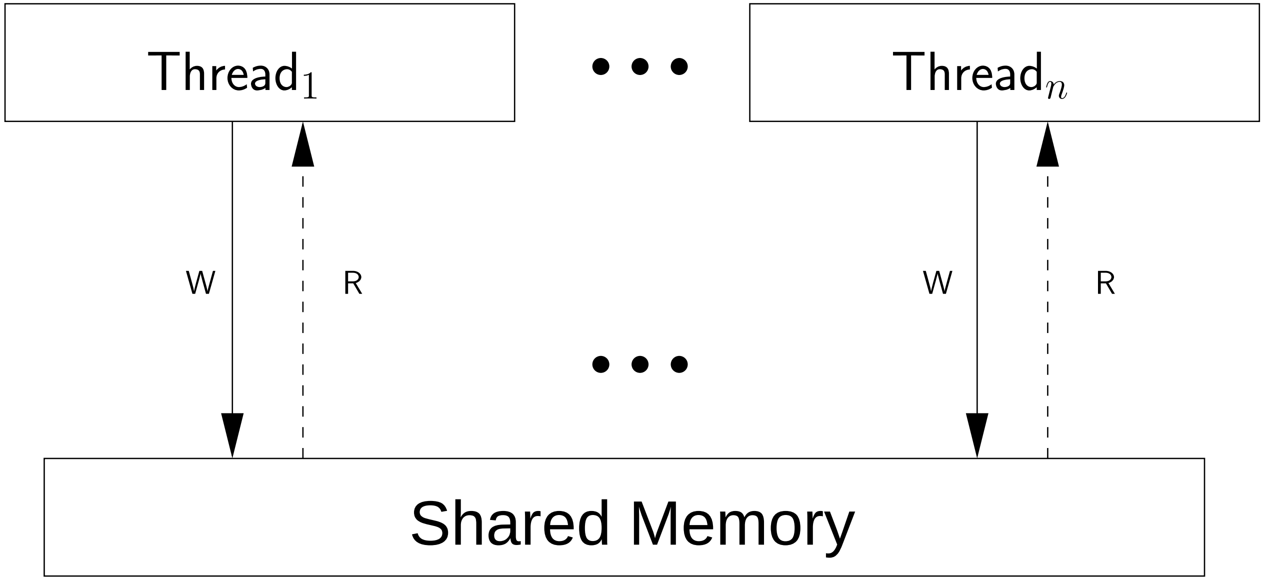
ML Workshop 2006: HashCaml
Shinwell, Billings, Strniša

$\longrightarrow$ POPLmark challenge (with Pierce, Weirich, Zdancewic, ...)
$\longrightarrow$ Ott (with Zappa Nardelli, Owens,...)

Example: Relaxed Shared-Memory Concurrency

(Example of Option 4: As-principled-as-you-can-manage Approach to Mainstream Programming Languages)

What You Would Naturally Expect



Multiple hardware threads operating on the *same* memory

Asynchronously...

The Ghoſt of Multiproceſſors Paſt

BURROUGHS D825, 1962

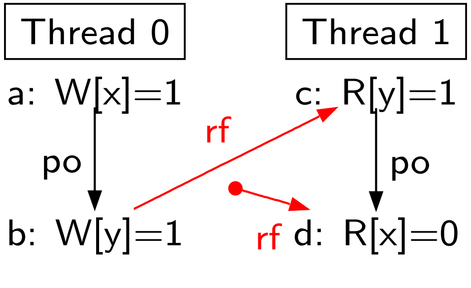
‘‘Outstanding features include truly modular hardware with parallel processing throughout’’

FUTURE PLANS
The complement of compiling languages is to be expanded.

Multiprocessor Concurrency

Simple Message-Passing Example:

Thread 0	Thread 1
x=1	r1=y if this reads 1...
y=1	r2=x ...will this definitely read 1?
Initial state: x=0 y=0	

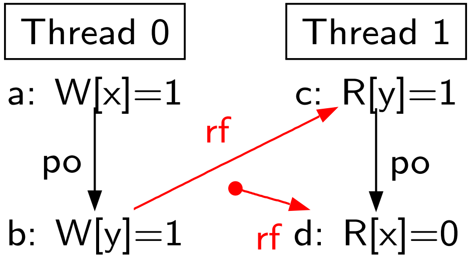


Test MP

Multiprocessor Concurrency

Simple Message-Passing Example:

Thread 0	Thread 1
x=1	r1=y if this reads 1...
y=1	r2=x ...will this definitely read 1?
Initial state: x=0	y=0



Test MP

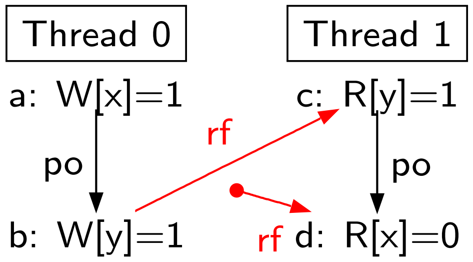
x86: yes

		POWER			ARM			
	Kind	PowerG5	Power6	Power7	Tegra2	Tegra3	APQ8060	A5X
MP	Allow	10M/4.9G	6.5M/29G	1.7G/167G	40M/3.8G	138k/16M	61k/552M	437k/185M

Multiprocessor Concurrency

Simple Message-Passing Example:

Thread 0	Thread 1
x=1	r1=y if this reads 1...
y=1	r2=x ...will this definitely read 1?
Initial state: x=0 y=0	

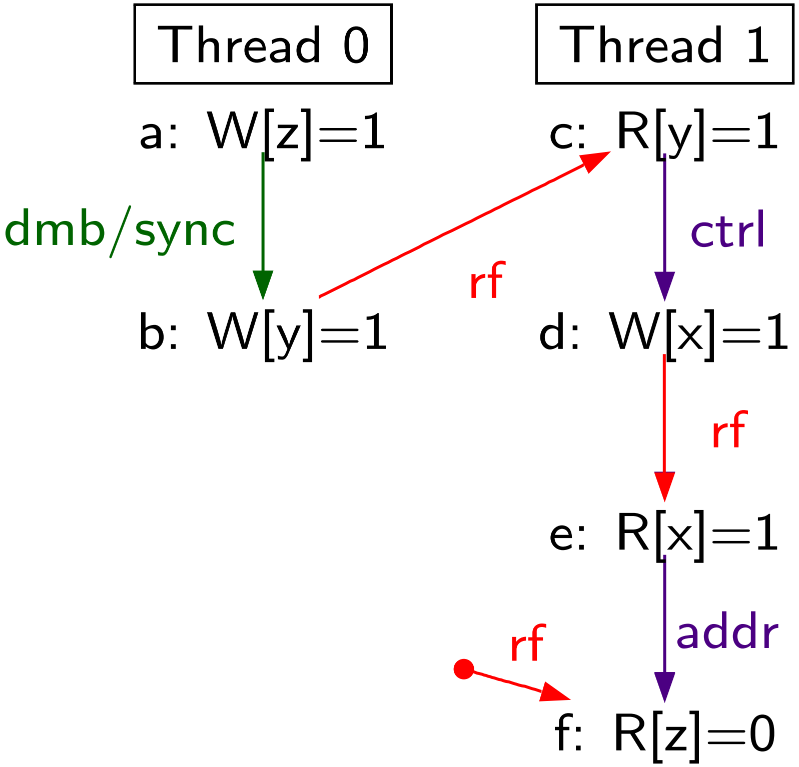


Test MP

Microarchitecturally: writes committed, writes propagated, and/or reads satisfied out-of-order

Compilers: common subexpression elimination

Less Simple Example



Test PPOCA: Allowed

		POWER			ARM			
	Kind	PowerG5	Power6	Power7	Tegra2	Tegra3	APQ8060	A5X
PPOCA	Allow	1.1k/3.4G	0/49G U	175k/157G	0/24G U	0/39G U	233/743M	0/2.2G U
PPOAA	Forbid	0/3.4G	0/46G	0/209G	0/24G	0/39G	0/26G	0/2.2G

What do the vendor architecture specs say?

“all that horrible horribly incomprehensible and confusing [...] text that no-one can parse or reason with — not even the people who wrote it”

Anonymous Processor Architect, 2011

How to Make Sense of This?

1. figure out how to talk to systems people
2. test generation (manual and systematic)
3. test harness (pre-silicon and production - found many surprising phenomena plus some serious bugs)
4. write model in math (4000 lines)
5. generation of exhaustive simulator from model
6. auto-comparison between tests and model
7. English version of model, in sync with maths (few pages)
8. discussion with architects
9. `goto 1`

[Sarkar, Maranget, Alglave, Williams, Sewell]

...since 2007

- clarify concurrency model for x86, IBM POWER, ARM (Sarkar, Owens, Zappa Nardelli, Alglave, Maranget,...)
- clarify concurrency model for C11/C++11 (Batty, ...)

Industry Impact:

- x86 consensus spec
- in-depth discussion with IBM and ARM architects
- found processor bugs
- fixed C/C++ standards
- compilation of C/C++11 concurrency to x86, Power, ARM
- compiler concurrency testing (Zappa Nardelli)

Using those models for s/w verification: CompCertTSO, C/C++11, take-up by others

Executable Semantics

Must be able to:

- explore the semantics interactively
- decide whether an experimentally observed result is allowed by the semantics
- compute the set of *all* semantics-allowed behaviours of small test programs
- reason about the semantics

QUICK CPPMEM DEMO

Principles?

The Importance of Opportunism

Research

1. Identify problem worth solving

2. Guess how hard it’s going to be

3. Solve it

4. Explain problem + solution to people

Research

1. Identify problem worth solving
2. Guess how hard it's going to be
3. Solve it
4. Explain problem + solution to people
5. ...get a job
6. GOTO 1

(btw, for Option 4 people... we'll be recruiting over the next years)