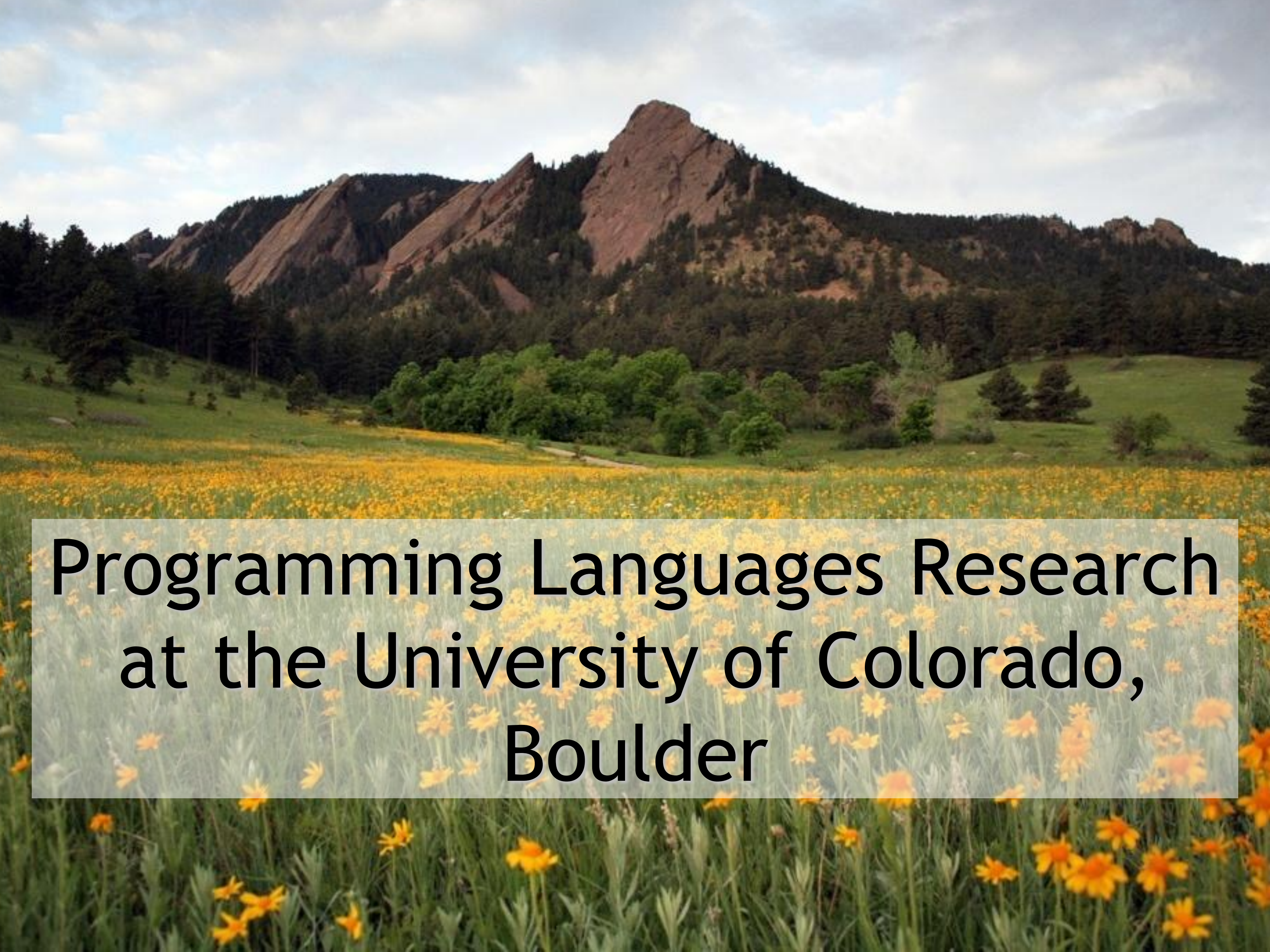
Calling Context Abstraction with Shapes

Xavier Rival
INRIA/ENS Paris

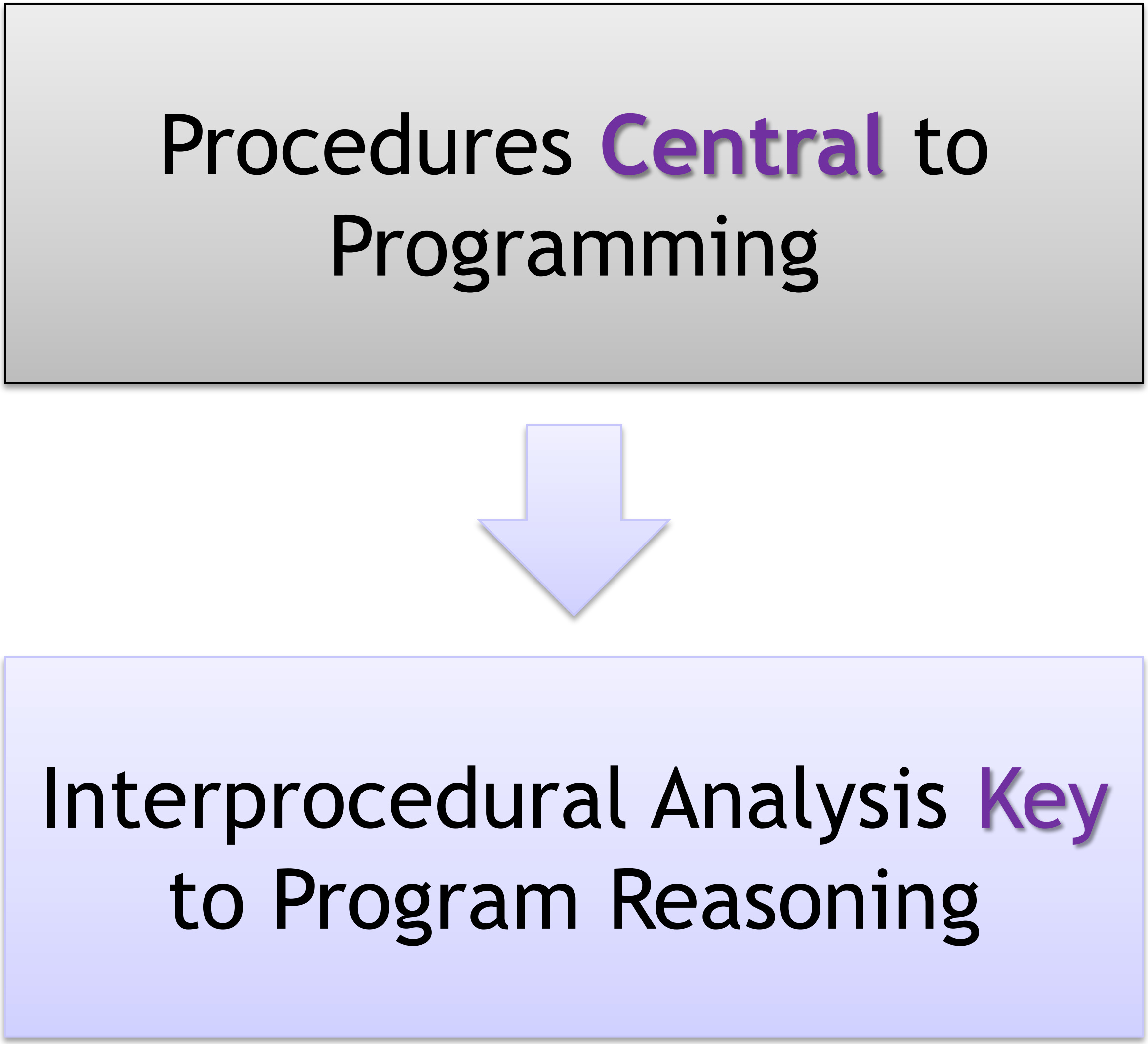
Bor-Yuh Evan Chang 張博聿
U of Colorado, Boulder

National Taiwan University – December 17, 2010
Work to Appear in POPL 2011



Programming Languages Research at the University of Colorado, Boulder

Interprocedural analysis is important



Procedures **Central** to Programming

Interprocedural Analysis **Key** to Program Reasoning

```
int f(int x) { … }

let f x = …

function f(x) { … }
```

Two approaches to interprocedural analysis

Each procedure separately

“build procedure summaries”

+ m

+ r

pre/post invariants

- ab

- c

s

Whole program simulation

“simulate inlined procedures”

t

each call site

analyze

each definition vs. each call

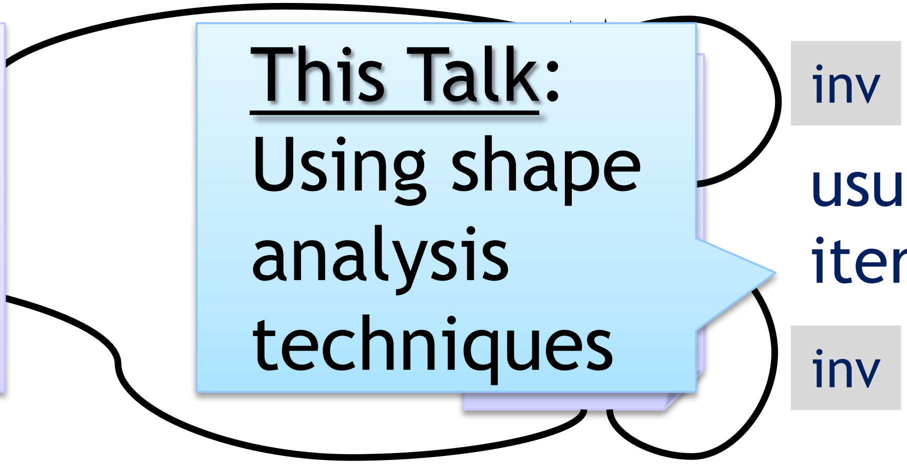
infer abstractions of

pairs vs. individual

states

Two approaches to interprocedural analysis

State of Practice:
Almost all interprocedural analyses today



This Talk:
Using shape analysis techniques

tabulation	pre_1	$post_1$

pre_1	$post_1$
pre_2	$post_2$

Challenge: Frame Inference

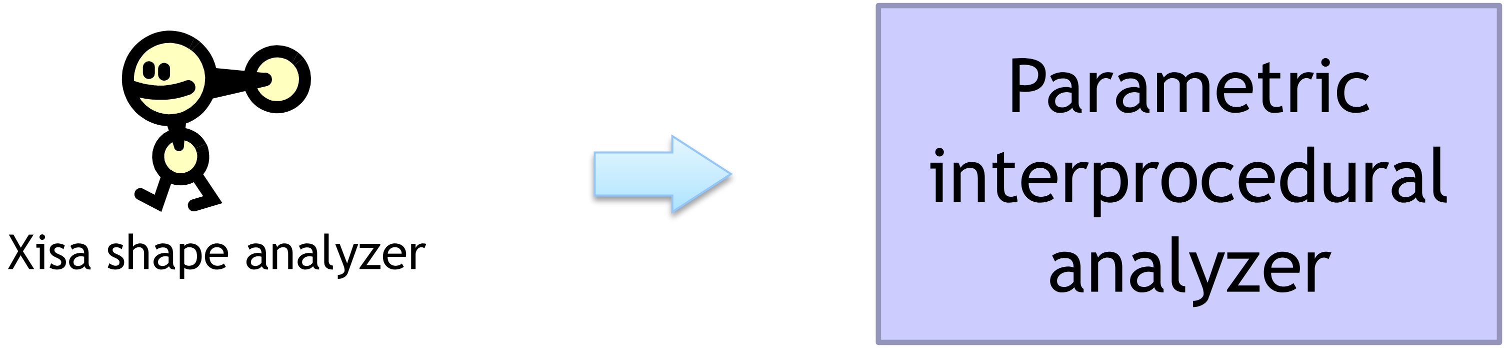
caller state

Challenge: Unbounded Calls

“infinite inlining”

Our approach is to …

Apply inductive shape analysis to summarize unbounded calling contexts in a whole program, state-based interprocedural analysis.



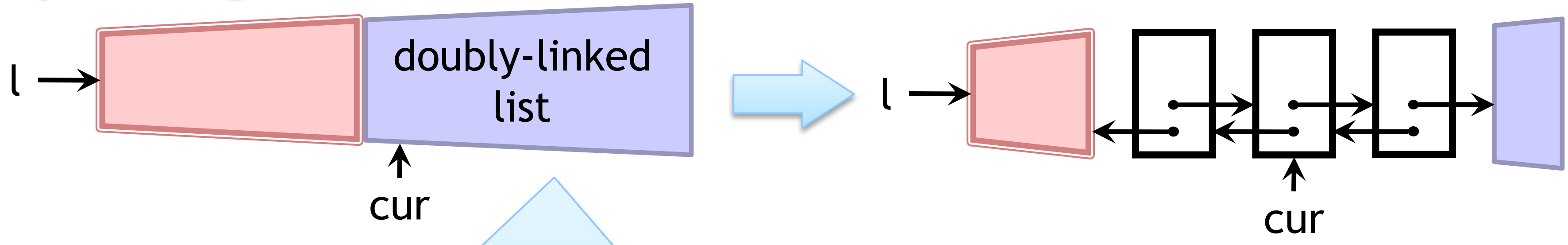
Xisa shape analyzer

Parametric interprocedural analyzer

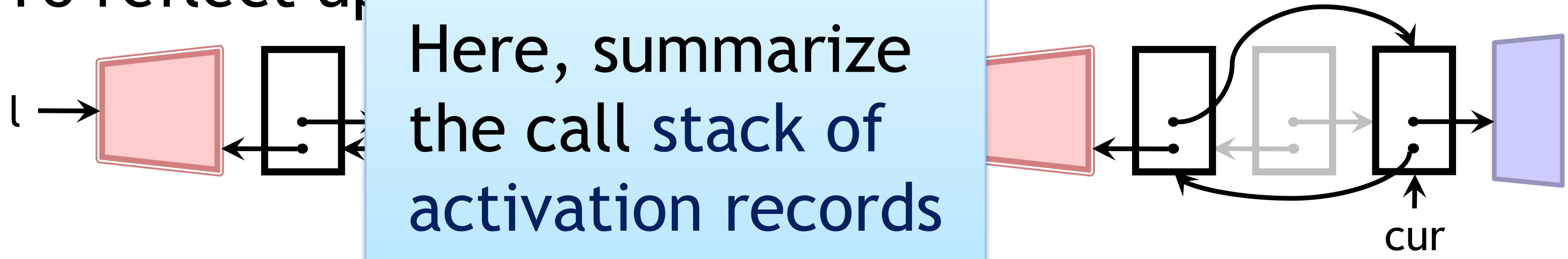
- "Very" context-sensitive
  - Simultaneous summarization of the stack and heap
- Use simpler base domains with precision
  - Need only abstract sets of states not relations

Shape analysis is an abstract interpretation on abstract memory descriptions with …

Splitting of summaries

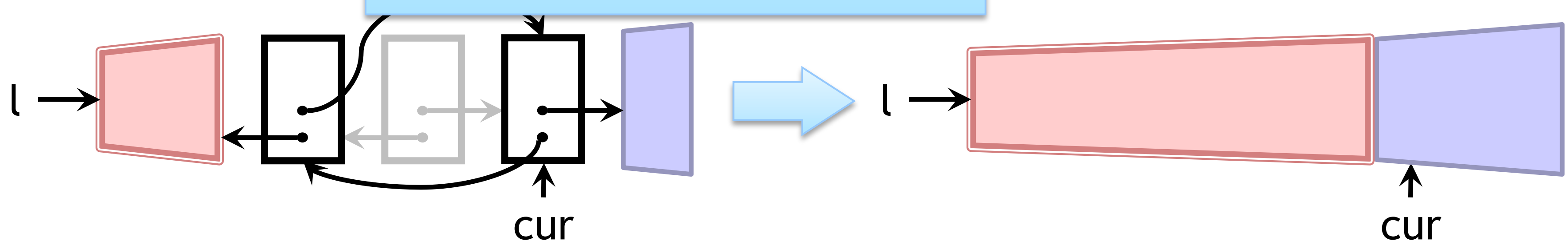


To reflect up



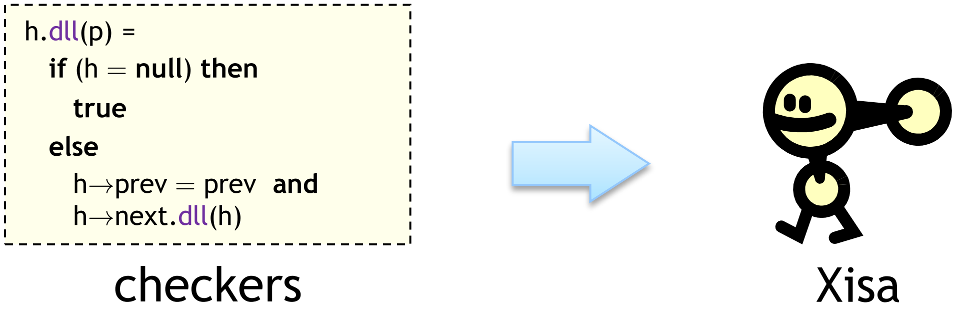
And summar

Here, summarize the call stack of activation records with “shapes”



Challenge: Obtain stack inductives

Xisa is a shape analysis with an precise abstraction based around user-supplied invariant checkers.



```
h.dll(p) =
  if (h = null) then
    true
  else
    h→prev = prev  and
    h→next.dll(h)
```

checkers

Xisa

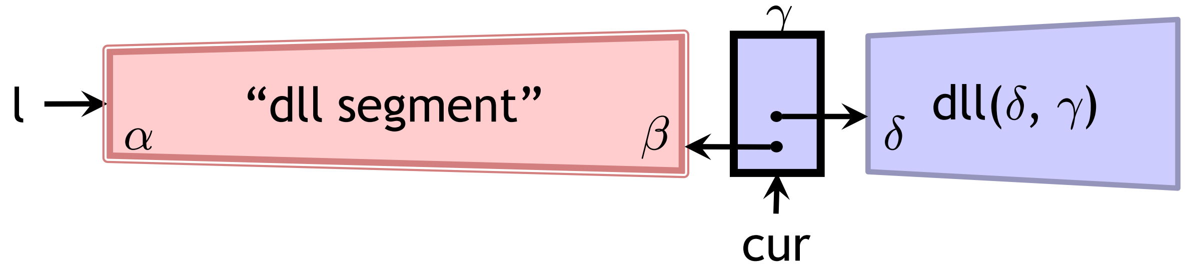
- Reasonable to expect user-supplied inductive definitions for user-defined heap structures
- Unreasonable to expect inductive definitions describing possible call stacks.
  - **Contribution**: derived automatically

Roadmap

- Background: Memory as graphs
- Abstracting calling contexts
- Deriving inductive cases for calling context summarization

Memory as separating shape graphs

Analogous to separation logic formulas



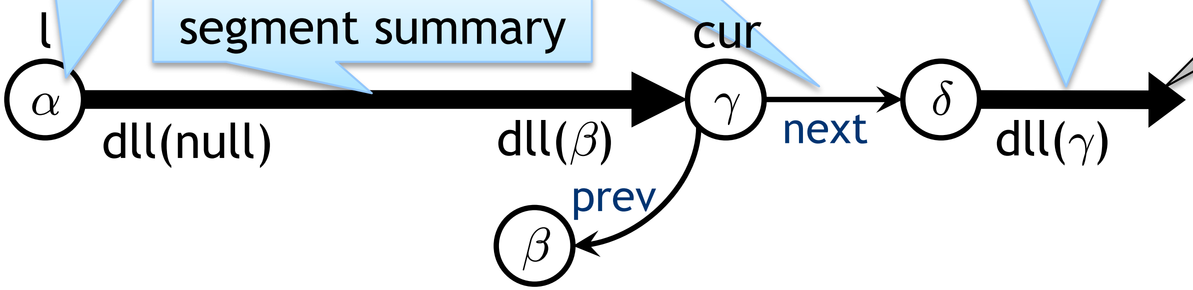
memory address (value, not cell)

memory cell (points-to: γ→next $= \delta$)

checker summary (inductive pred)

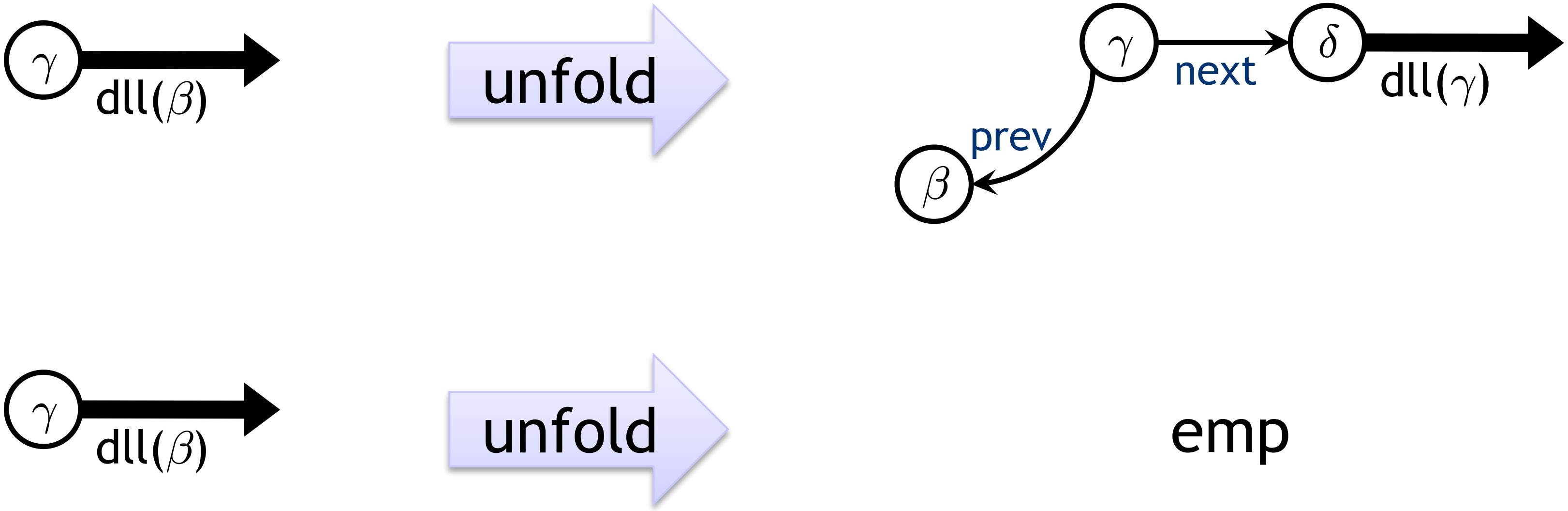
Some number of memory cells (thin edges)

segment summary



```
h.dll(p) =
  if (h = null) then
    true
  else
    h→prev = p  and
    h→next.dll(h)
```

Unfolding inductive summaries



Possible unfoldings give an inductive definition

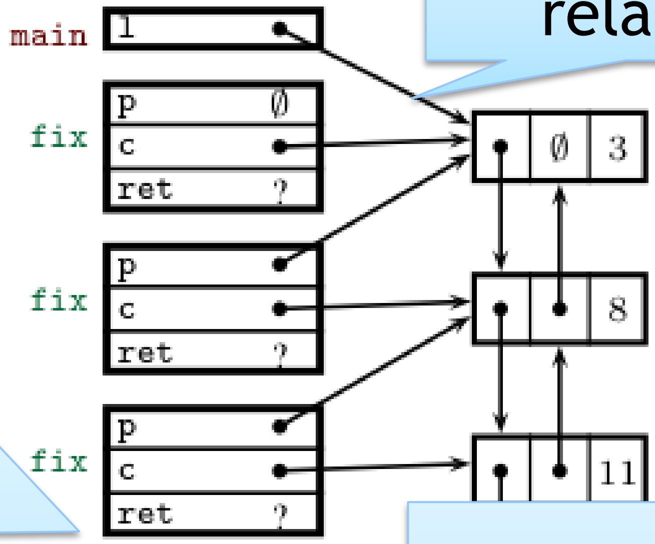
Roadmap

- Background: Memory as graphs
- Abstracting calling contexts
- Deriving inductive cases for calling context summarization

Concrete view of a recursive example

```
void main() {
  …
  l = fix(l, NULL);
}
// c is a singly-linked list
dll* fix(dll* c, dll* p) {
  dll* ret;
  if (c != NULL) {
    c->prev = p
    c->next = f
    if (check(c
      ret = c->r
      remove(c)
    }
    else { ret = c; }
  }
  return c;
}
```

stack unbounded, needs summarization

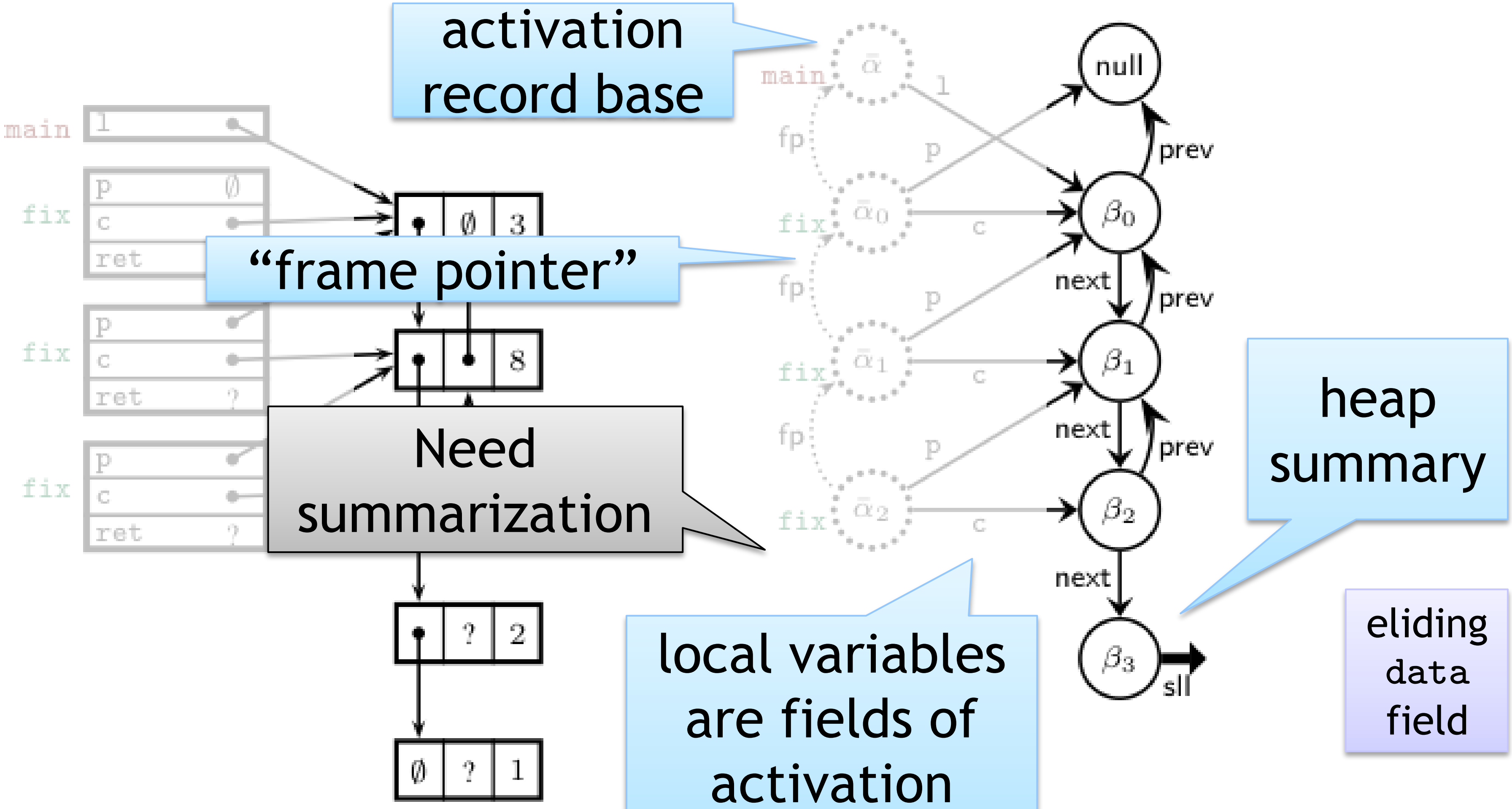


complex relations

heap unbounded, needs summarization

stack heap

Putting calling contexts into shape graphs



activation record base

“frame pointer”

Need summarization

heap summary

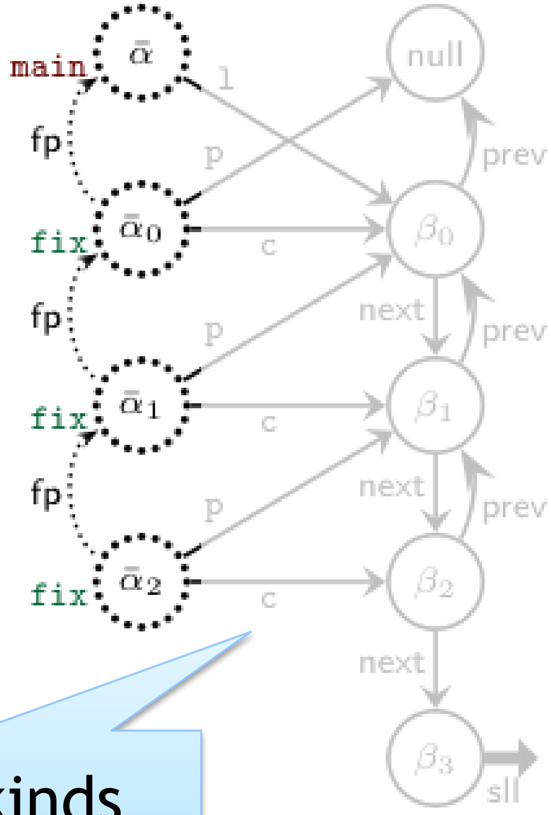
eliding data field

local variables are fields of activation records

Calling context is a list

Inductive structure always a "list"

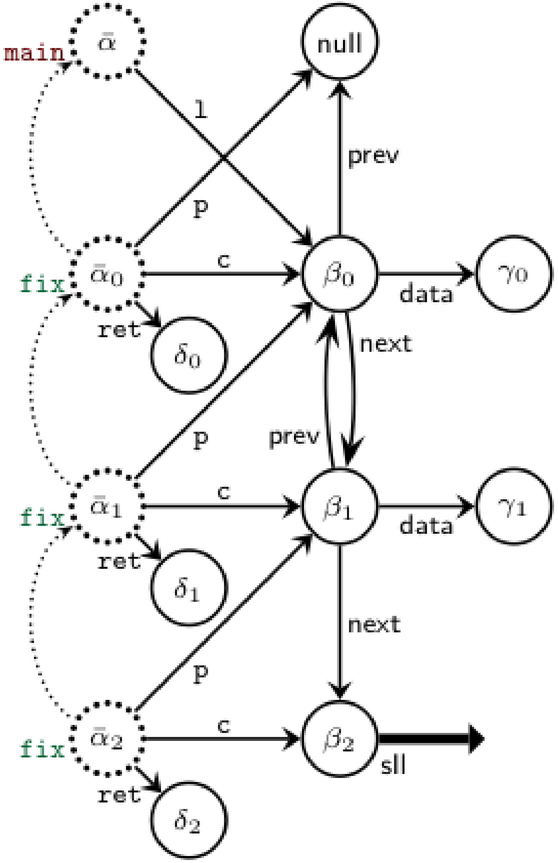
Summarize calling context using an inductive predicate stack whose definition is derived "on-the-fly"



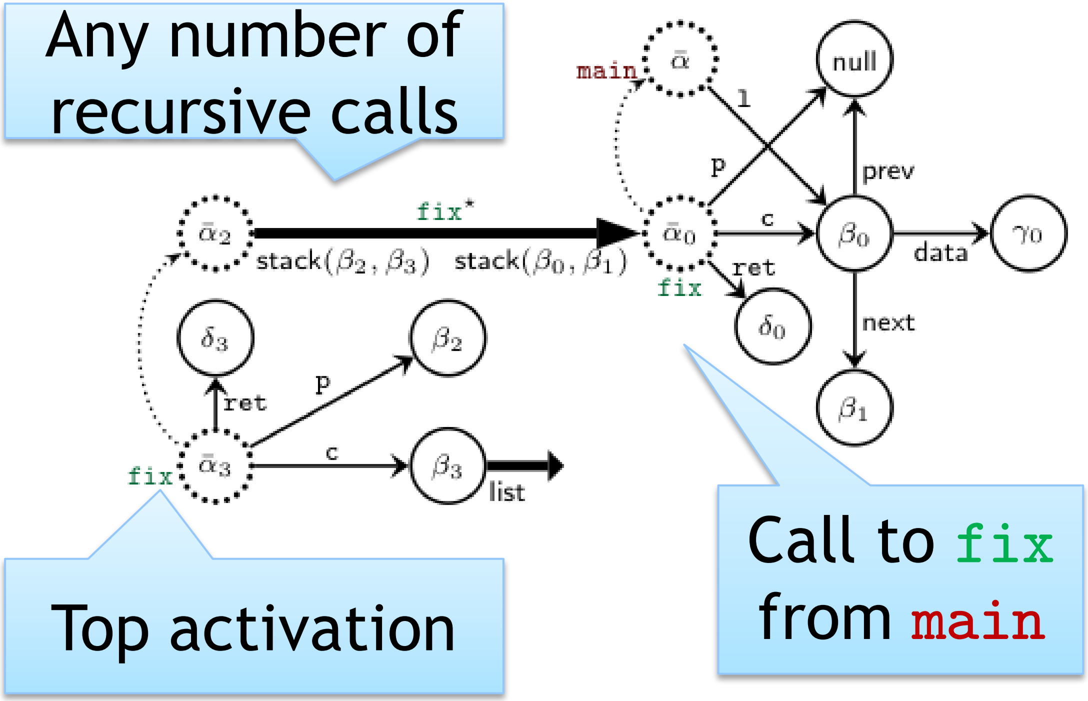
"Node" kinds program-specific

Calling context summarization

Example instance (with all fields)



A call stack summary



Any number of recursive calls

Top activation

Call to fix from main

Roadmap

- Background: Memory as graphs
- Abstracting calling contexts
- Deriving inductive cases for calling context summarization

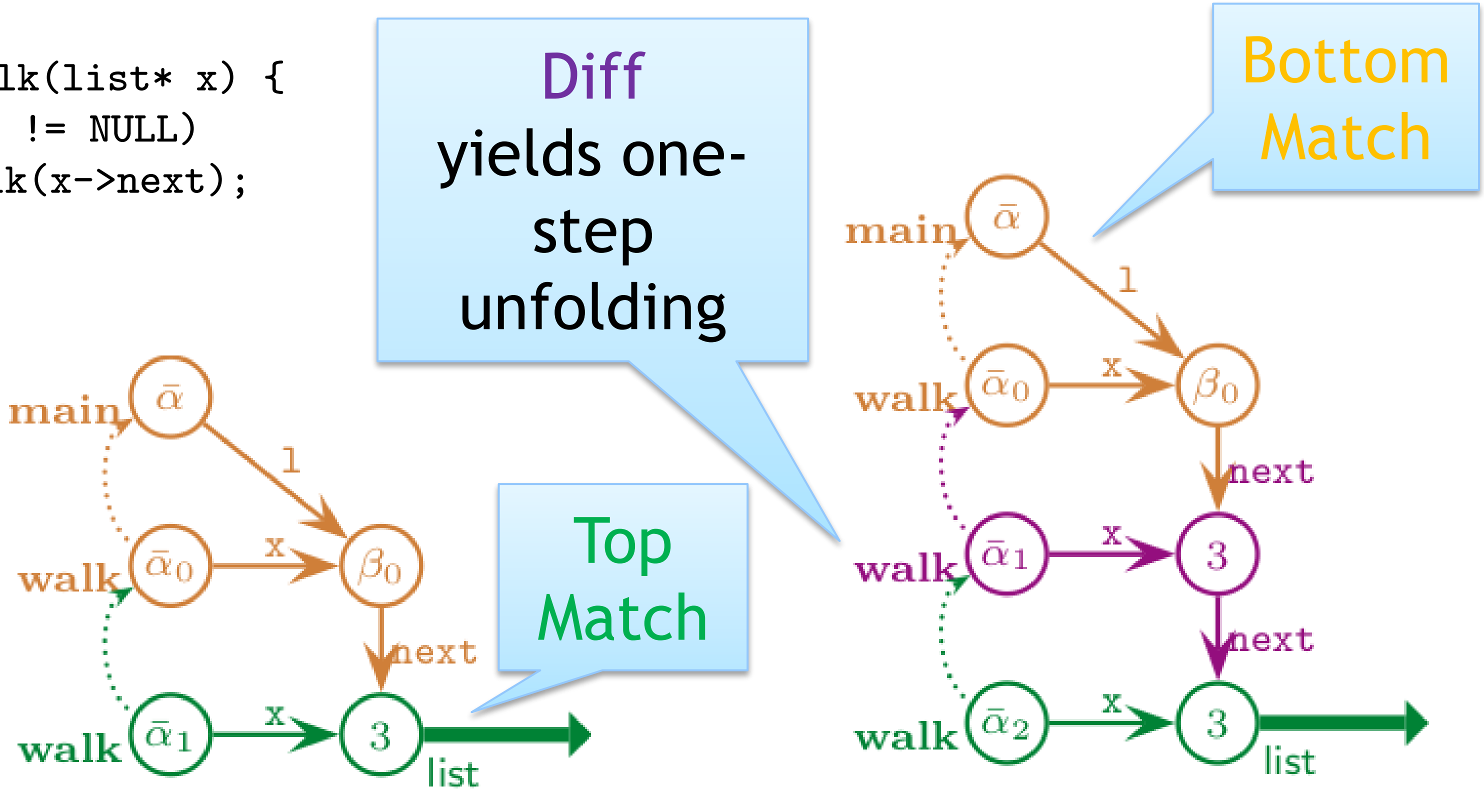
Deriving the stack inductive definition

Intuition

- At a call, new activation record added
- Need to widen to obtain summaries with stack instances (but need the definition of stack)
- Compare a few iterations to augment the definition of stack, then apply widening.
  - Subtraction

Subtraction

```
void main() {
   ...
   walk(l);
}
void walk(list* x) {
   if (x != NULL)
      walk(x->next);
}
```



Diff yields one-step unfolding

Bottom Match

Top Match

Iteration 1

Iteration 2

Preliminary Experience

Benchmark	Recursive (ms)	Iterative (ms)
lis		4
lis		4
lis		6
lis		1
lis		4
lis		3
lis		5

Case Study:

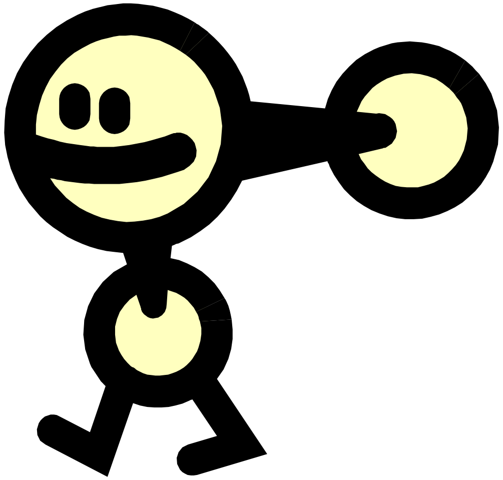
Discussing precision with this approach versus the modular approach.

[see paper]

-
- points per recursion (call site and return site)

Conclusion

- Xisa applied straightforwardly
  - folding at call sites
  - unfolding at return sites
  - widening applied on recursion
  - core analysis algorithms remain
  - evidence for flexibility of the framework
- New option for interprocedural analysis
  - “very” context-sensitive
  - no need to abstract relations ⇒ simpler base domains



http://www.cs.colorado.edu/~bec/

Programming Languages Research at the University of Colorado, Boulder

Amer Diwan

Jeremy Siek

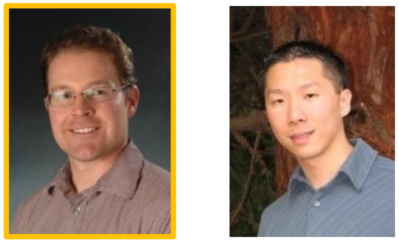
Bor-Yuh Evan Chang

Sriram Sankaranarayanan

PL research at CU has ***breadth***!

How do we effectively express computation?

language design, type systems, logic

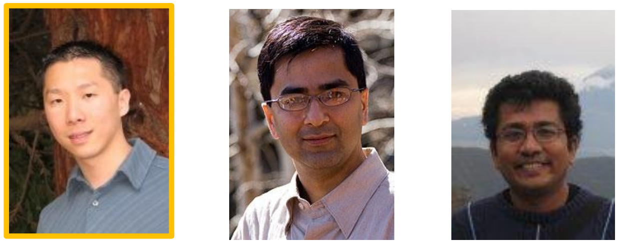


How do we make programs run efficiently?

performance analysis, compilation

How do we assist reasoning about programs?

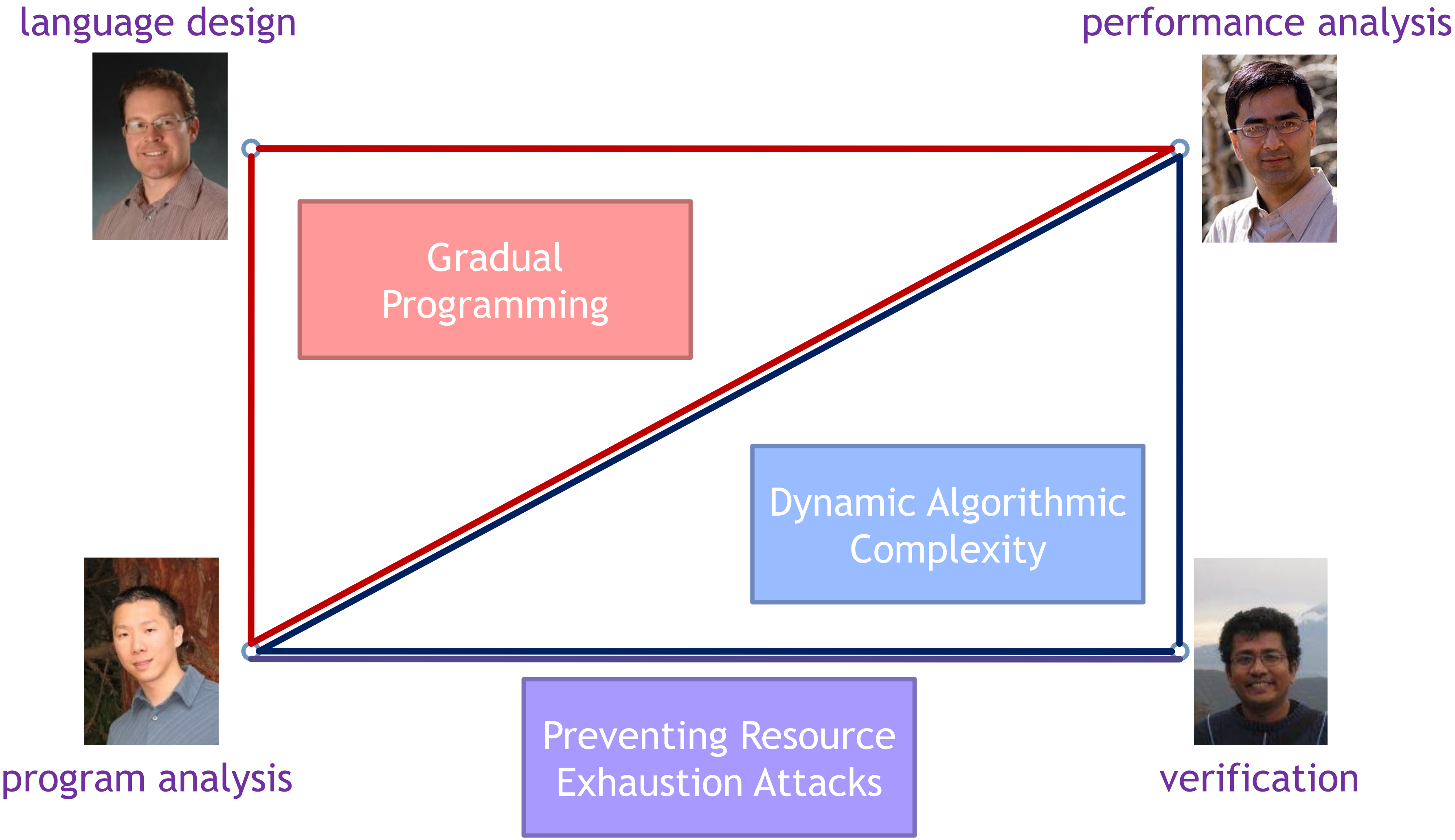
program analysis, development tools



How do we get reliable, secure software?

verification, model checking

PL researchers at CU *collaborate*!



language design

performance analysis

Gradual Programming

Dynamic Algorithmic Complexity

program analysis

Preventing Resource Exhaustion Attacks

verification

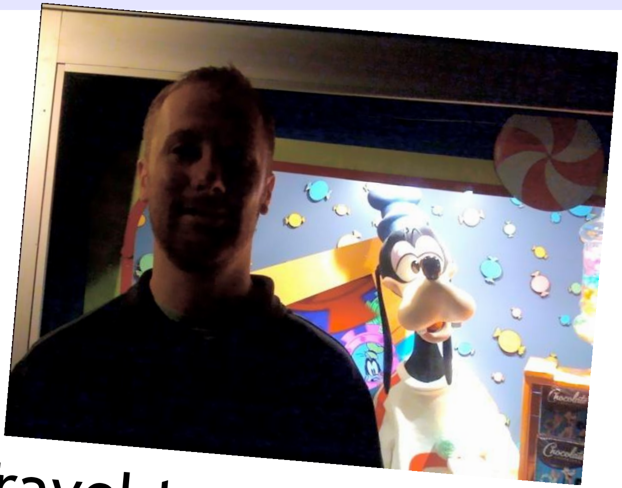
Formal methods *connections*

Prof. Aaron Bradley (ECEE)

Prof. Fabio Somenzi (ECEE)

The PL group has *fun* together!

Group meetings at the Boulder Tea House once/twice a month

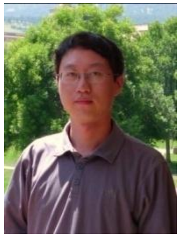


Travel to conferences (Todd at OOPSLA'09)

Successes: 2 papers at each of POPL'11, PLDI'10, and POPL'10

Our group

Devin


Weiyu

PhD

Sam

Jonathan

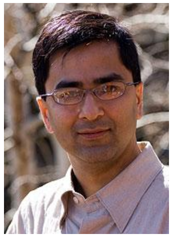
Aleks

Huck

MS

BS

James

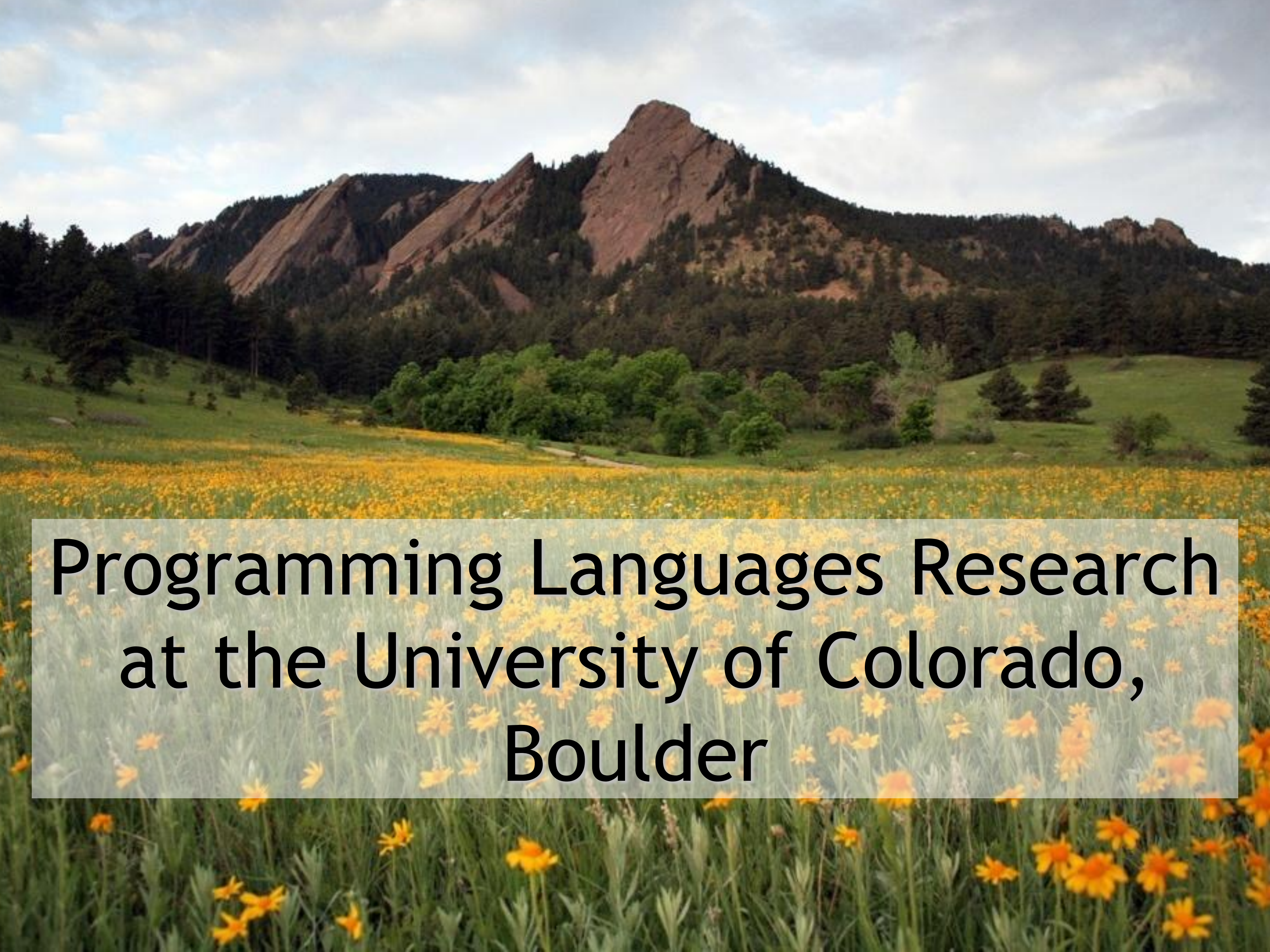

Amer

Jeremy

Faculty

Evan

Sriram



Programming Languages Research at the University of Colorado, Boulder

Applying to Colorado

- Computer Science Department information
  - http://www.cs.colorado.edu/grad/admission/
- Deadlines
  - Jan 2 for Fall (Oct 1 for Spring)
- Graduate Advisor: Jackie DeBoard
  - jacqueline.deboard@colorado.edu
- Talk to me about application fee waiver
  - http://www.cs.colorado.edu/~bec/