Pointer Analysis: The Big Picture View

Uday Khedker

(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,

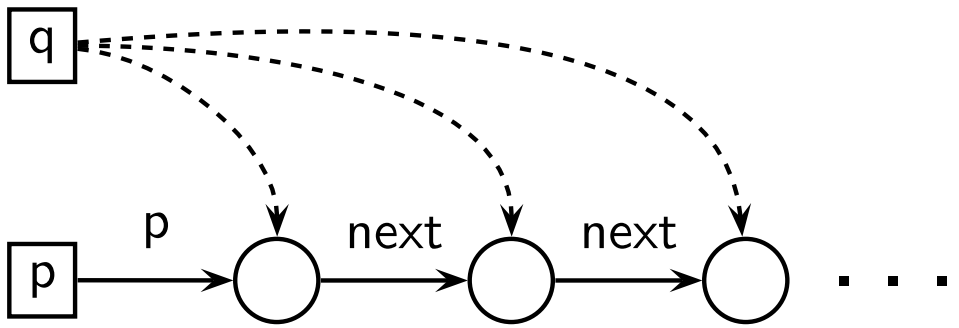
Indian Institute of Technology, Bombay

Dec 2017

Outline

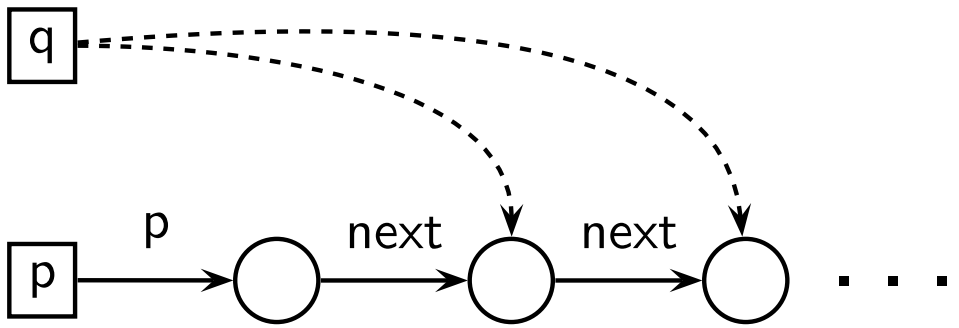
- The What and Why of pointer analysis
- Abstactions vs. approximations in pointer analysis
- An engineering landscape for pointer analysis
- Our Holy Grail in pointer analysis

Code Optimization In Presence of Pointers

Program	*Memory graph at statement 5*
1. q = p; 2. while (. . .) { 3. q = q→next; 4. } 5. p→data = r1; 6. print (q→data); 7. p→data = r2;	

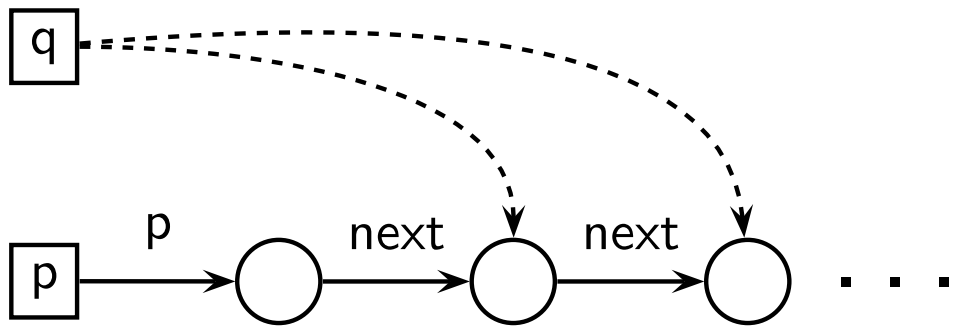
- Is p→data live at the exit of line 5? Can we delete line 5?

Code Optimization In Presence of Pointers

Program	*Memory graph at statement 5*
1. q = p; 2. do { 3. q = q→next; 4. } while (. . .) 5. p→data = r1; 6. print (q→data); 7. p→data = r2;	

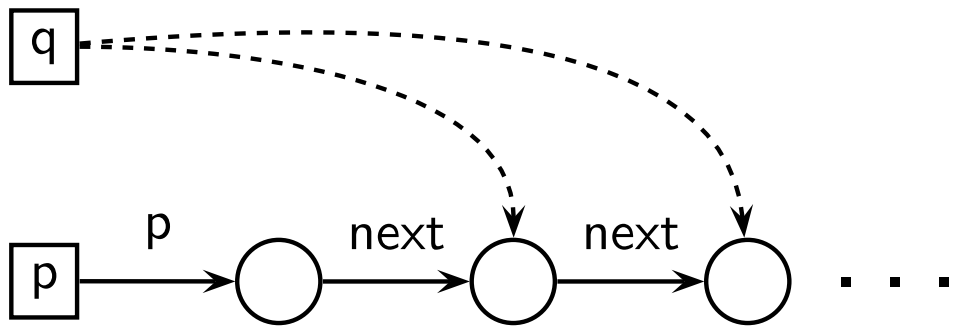
- Is p→data live at the exit of line 5? Can we delete line 5?

Code Optimization In Presence of Pointers

Program	*Memory graph at statement 5*
1. q = p; 2. do { 3. q = q→next; 4. } while (. . .) 5. p→data = r1; 6. print (q→data); 7. p→data = r2;	

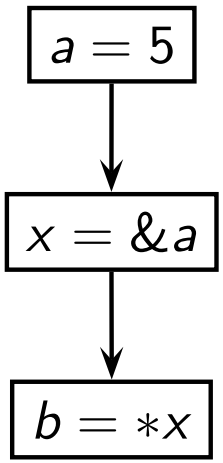
- Is p→data live at the exit of line 5? Can we delete line 5?
- We cannot delete line 5 if p and q can be possibly aliased (while loop or do-while loop with a circular list)

Code Optimization In Presence of Pointers

Program	*Memory graph at statement 5*
1. q = p; 2. do { 3. q = q→next; 4. } while (. . .) 5. p→data = r1; 6. print (q→data); 7. p→data = r2;	

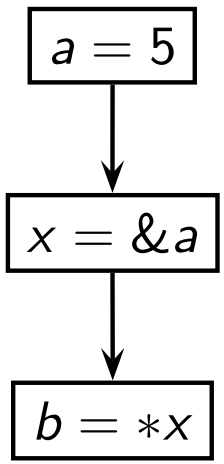
- Is p→data live at the exit of line 5? Can we delete line 5?
- We cannot delete line 5 if p and q can be possibly aliased (while loop or do-while loop with a circular list)
- We can delete line 5 if p and q are definitely not aliased (do-while loop without a circular list)

Code Optimization In Presence of Pointers

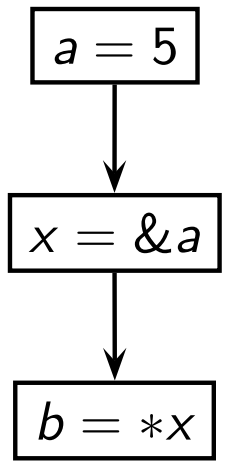


Original program

Code Optimization In Presence of Pointers

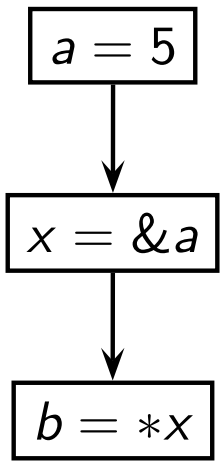


Original program

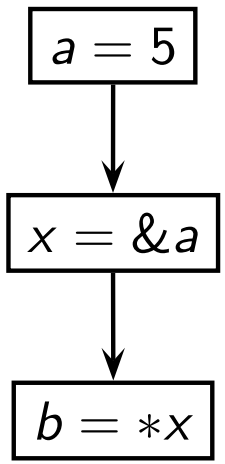


Constant propagation
without pointer analysis

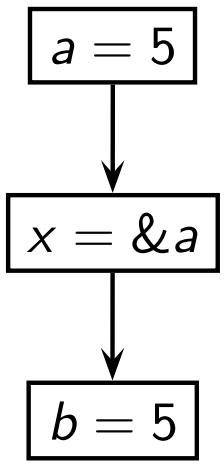
Code Optimization In Presence of Pointers



Original program

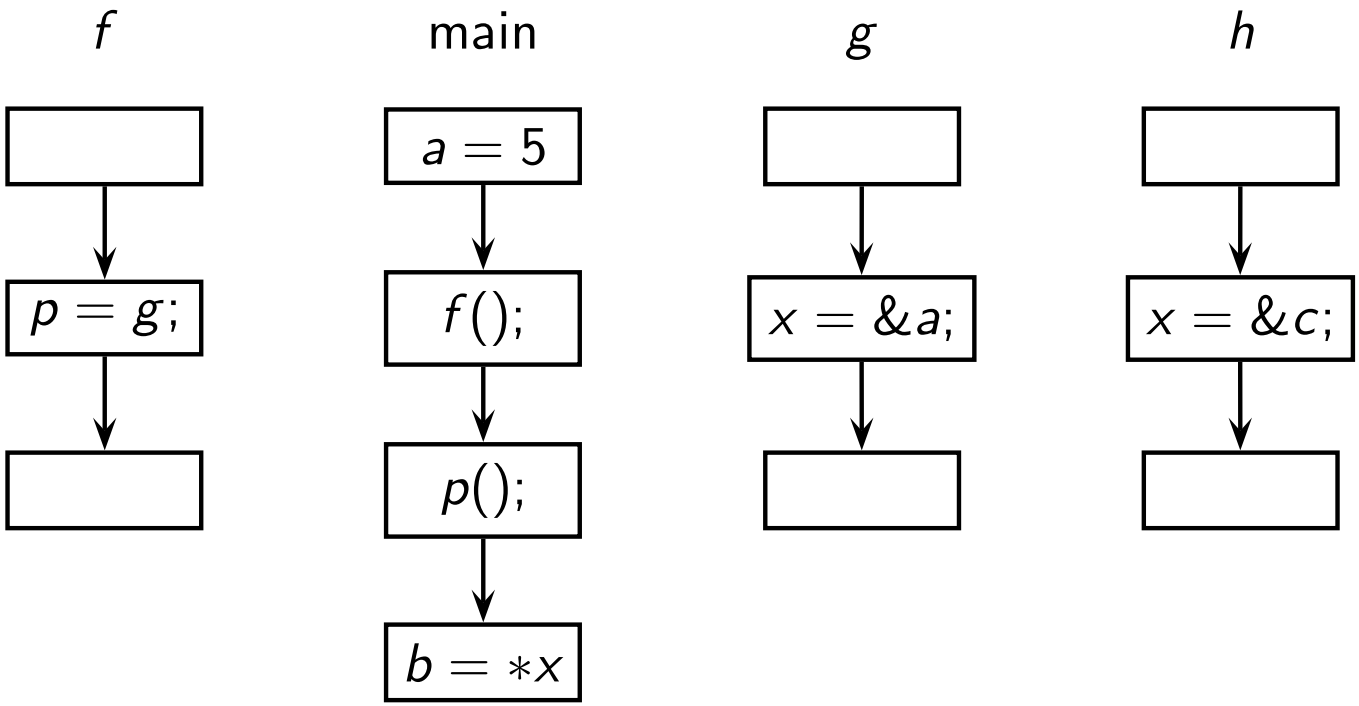


Constant propagation without pointer analysis

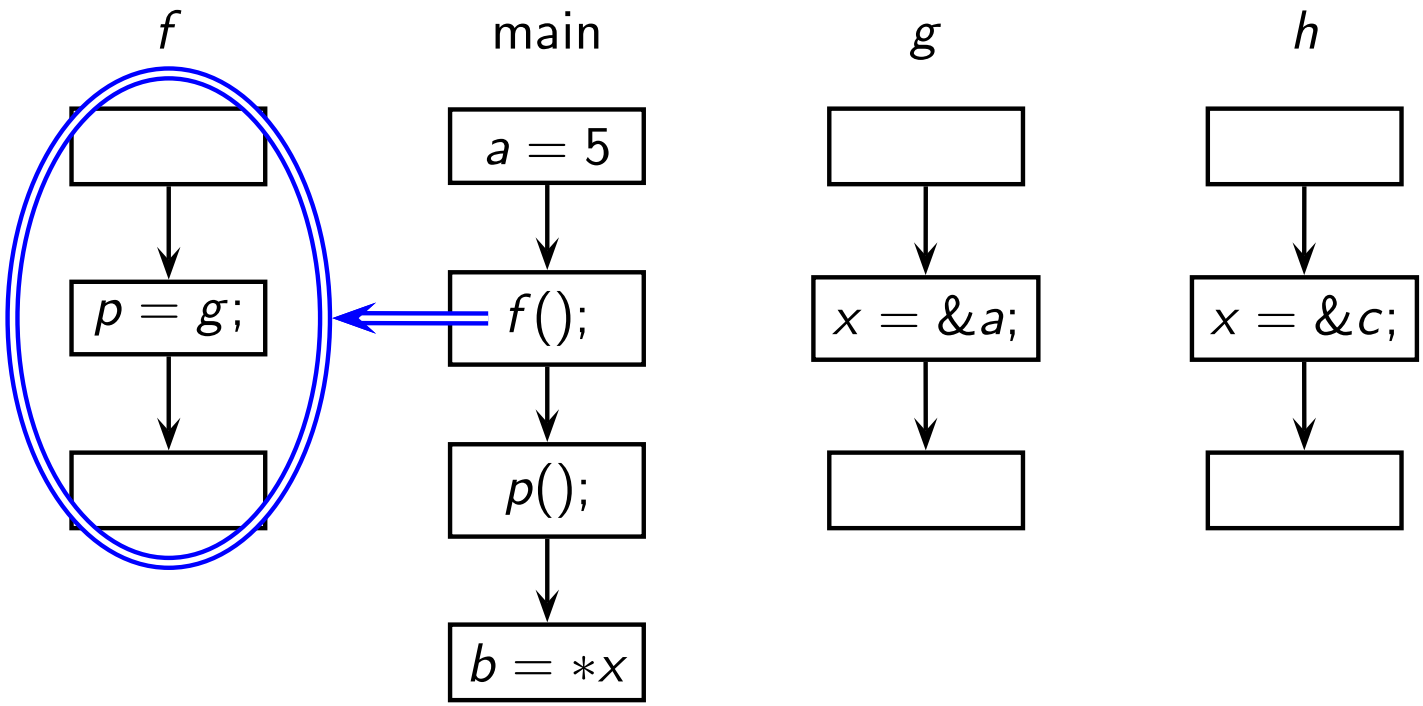


Constant propagation with pointer analysis

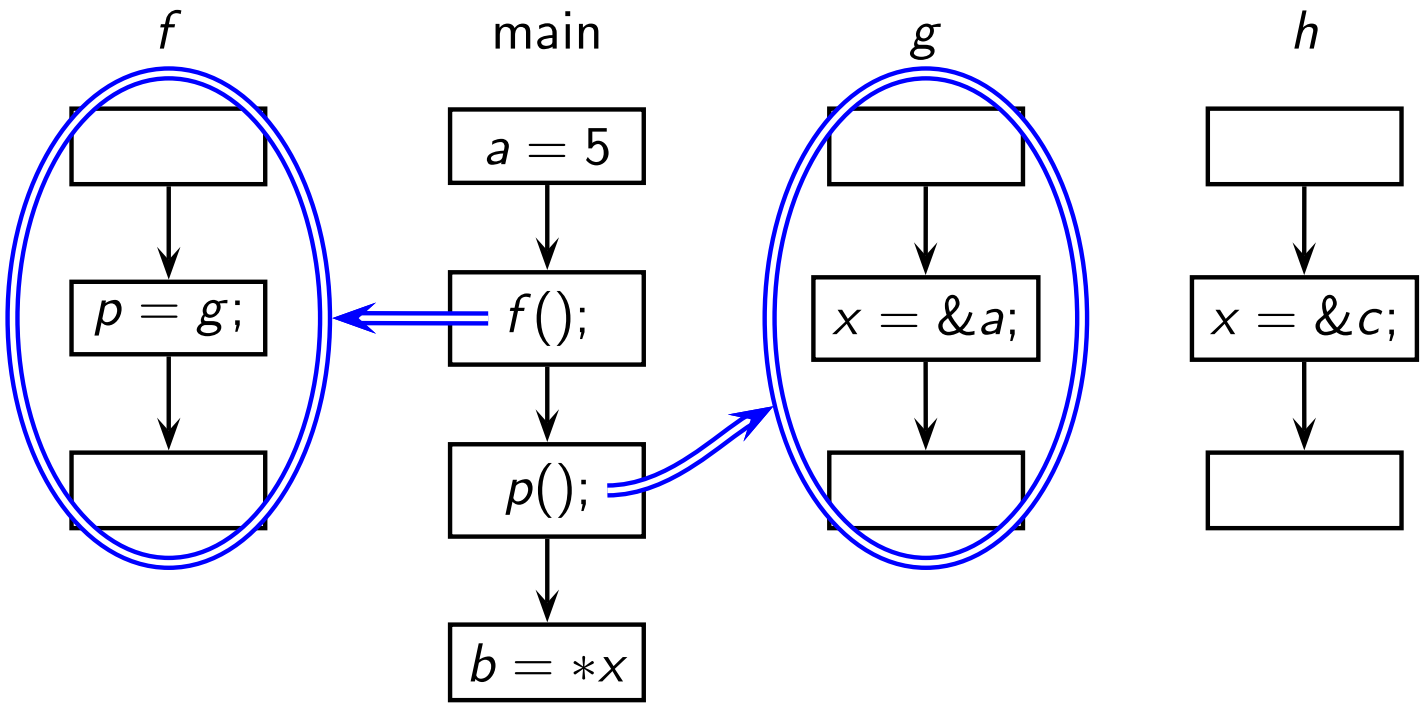
Code Optimization In Presence of Pointers



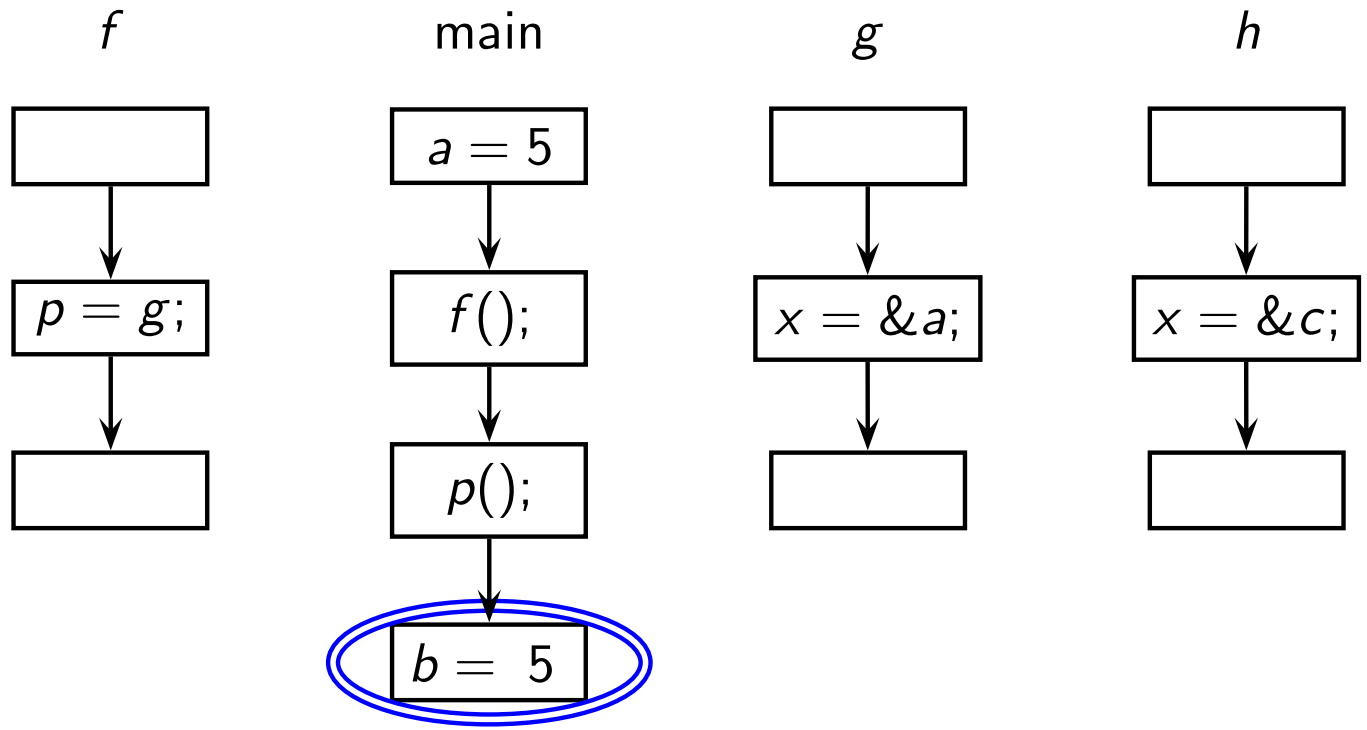
Code Optimization In Presence of Pointers



Code Optimization In Presence of Pointers



Code Optimization In Presence of Pointers

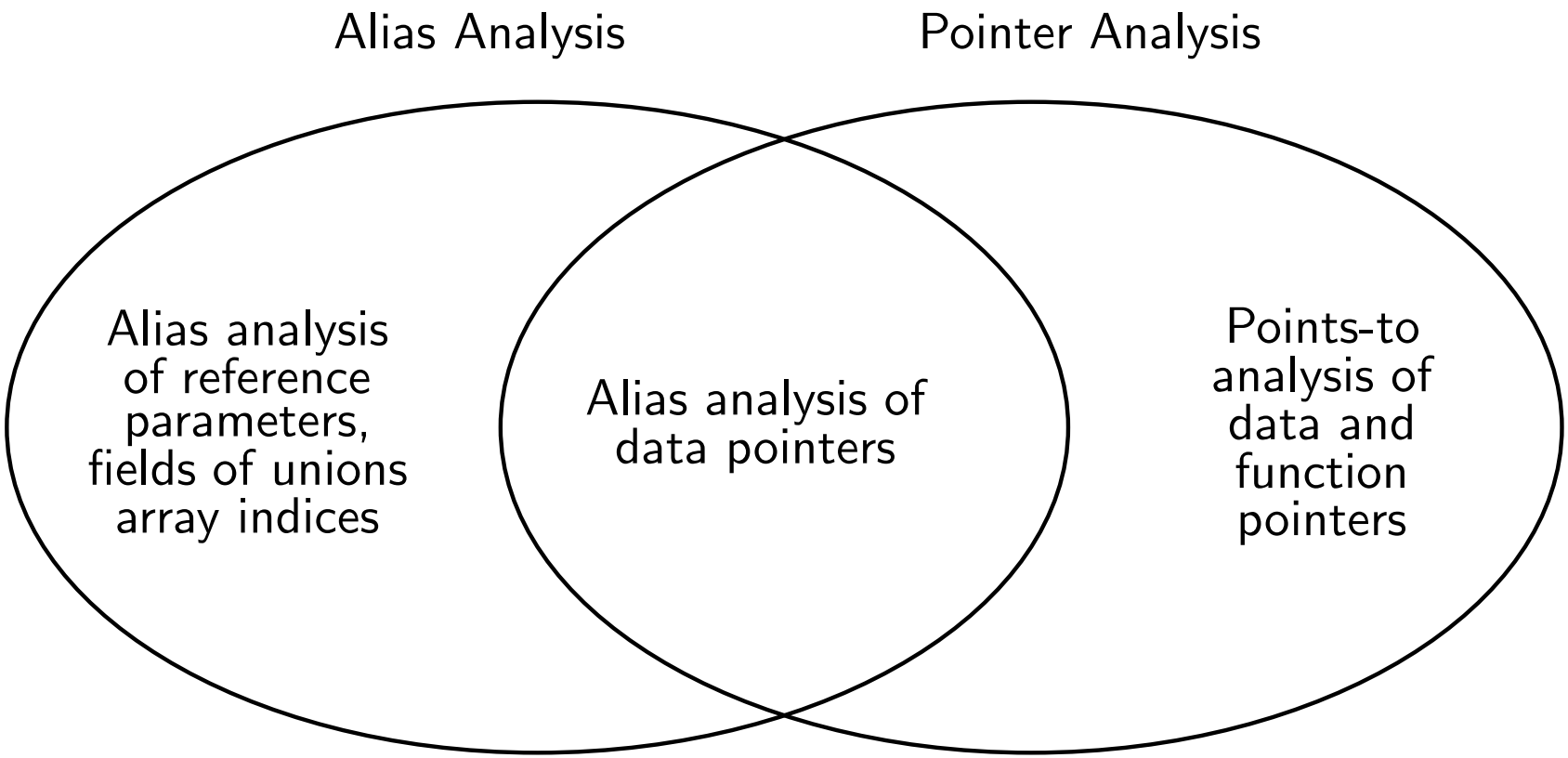


Pointer Analysis

- Answers the following questions for indirect accesses:
  - Which data is read? $x = *y$
  - Which data is written? $*x = y$
  - Which procedure is called? $p()$ or $x \rightarrow f()$
- Enables precise data flow and interprocedural control flow analysis
- Computationally intensive analyses are ineffective when supplied with imprecise points-to analysis,

  (e.g., model checking, interprocedural analyses)
- Needs to scale to large programs

The World of Pointer Analysis



Pointer Analysis Musings

- A keynote address:

  "The worst thing that has happened to Computer Science is C, because it brought pointers with it . . ."

  - Frances Allen, IITK Workshop (2007)

- A couple of influential papers
  - Which Pointer Analysis should I Use?

    Michael Hind and Anthony Pioli. ISTAA 2000
  - Pointer Analysis: Haven't we solved this problem yet ?

    Michael Hind PASTE 2001

Pointer Analysis Musings

- A keynote address:

  "The worst thing that has happened to Computer Science is C, because it brought pointers with it . . ."

  - Frances Allen, IITK Workshop (2007)

- A couple of influential papers
  - Which Pointer Analysis should I Use?

    Michael Hind and Anthony Pioli. ISTAA 2000

  - Pointer Analysis: Haven't we solved this problem yet ?

    Michael Hind PASTE 2001

Pointer Analysis Musings

- A keynote address:

  "The worst thing that has happened to Computer Science is C, because it brought pointers with it . . ."

  - Frances Allen, IITK Workshop (2007)

- A couple of influential papers
  - Which Pointer Analysis should I Use?

    Michael Hind and Anthony Pioli. ISTAA 2000
  - Pointer Analysis: Haven't we solved this problem yet ?

    Michael Hind PASTE 2001

    
  - 2017 . . 

The Mathematics of Pointer Analysis

In the most general situation

- Alias analysis is undecidable.

  Landi-Ryder [POPL 1991], Landi [LOPLAS 1992], Ramalingam [TOPLAS 1994]

- Flow insensitive alias analysis is NP-hard

  Horwitz [TOPLAS 1997]

- Points-to analysis is undecidable

  Chakravarty [POPL 2003]

Adjust your expectations suitably to avoid disappointments!

So what should we expect?

To quote Hind [PASTE 2001]

So what should we expect?

To quote Hind [PASTE 2001]

- "Fortunately many approximations exist"

So what should we expect?

To quote Hind [PASTE 2001]

- "Fortunately many approximations exist"
- "Unfortunately too many approximations exist!"

So what should we expect?

To quote Hind [PASTE 2001]

- "Fortunately many approximations exist"
- "Unfortunately too many approximations exist!"

Engineering of pointer analysis is much more dominant than its science

Pointer Analysis: Engineering or Science?

- Engineering view
  - Build quick approximations
  - The tyranny of (exclusive) OR
    Precision OR Efficiency?
- Science view
  - Build clean abstractions
  - Can we harness the Genius of AND?
    Precision AND Efficiency?

Pointer Analysis: Engineering or Science?

- Engineering view
  - Build quick approximations
  - The tyranny of (exclusive) OR
    Precision OR Efficiency?
- Science view
  - Build clean abstractions
  - Can we harness the Genius of AND?
    Precision AND Efficiency?

- Most common trend as evidenced by publications
  - Build acceptable approximations guided by empirical observations
  - The notion of acceptability is often constrained by beliefs rather than possibilities

Abstraction Vs. Approximation in Static Analysis

- Static analysis needs to create abstract values that represent many concrete values
- Mapping concrete values to abstract values
  - *Abstraction*.

    Deciding which properties of the concrete values are essential — *What*

    Ease of understanding, reasoning, modelling etc. — *Why*
  - *Approximation*.

    Deciding which properties of the concrete values cannot be represented accurately and should be summarised — *What*

    Decidability, tractability, or efficiency and scalability — *Why*

Abstraction Vs. Approximation in Static Analysis

- Abstractions
  - focus on precision and conciseness of modelling
  - tell us what we can ignore without being imprecise
- Approximations
  - focus on efficiency and scalability
  - tell us the imprecision that we have to tolerate

Abstraction Vs. Approximation in Static Analysis

- Abstractions
  - focus on precision and conciseness of modelling
  - tell us what we can ignore without being imprecise
- Approximations
  - focus on efficiency and scalability
  - tell us the imprecision that we have to tolerate
- *Build clean abstractions before surrendering to the approximations*

The Hope of Clean Abstractions in Pointer Analysis

- Common belief
- However,
- Because

The Hope of Clean Abstractions in Pointer Analysis

- Common belief

  Pointer information is very large

- However,

- Because

The Hope of Clean Abstractions in Pointer Analysis

- Common belief

  Pointer information is very large

- However,

  Precision can reduce the size of pointer information to make it far more manageable

- Because

The Hope of Clean Abstractions in Pointer Analysis

- Common belief

  Pointer information is very large

- However,

  Precision can reduce the size of pointer information to make it far more manageable

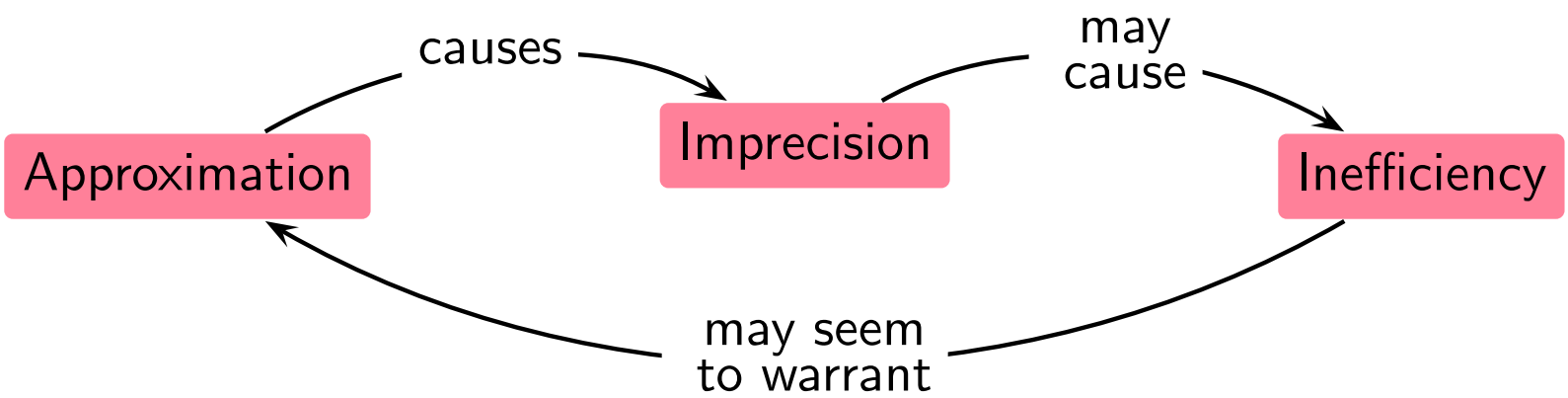
- Because

  At any program point, the usable pointer information is much smaller than the total pointer information

  Current methods perform many repeated and possibly avoidable computations

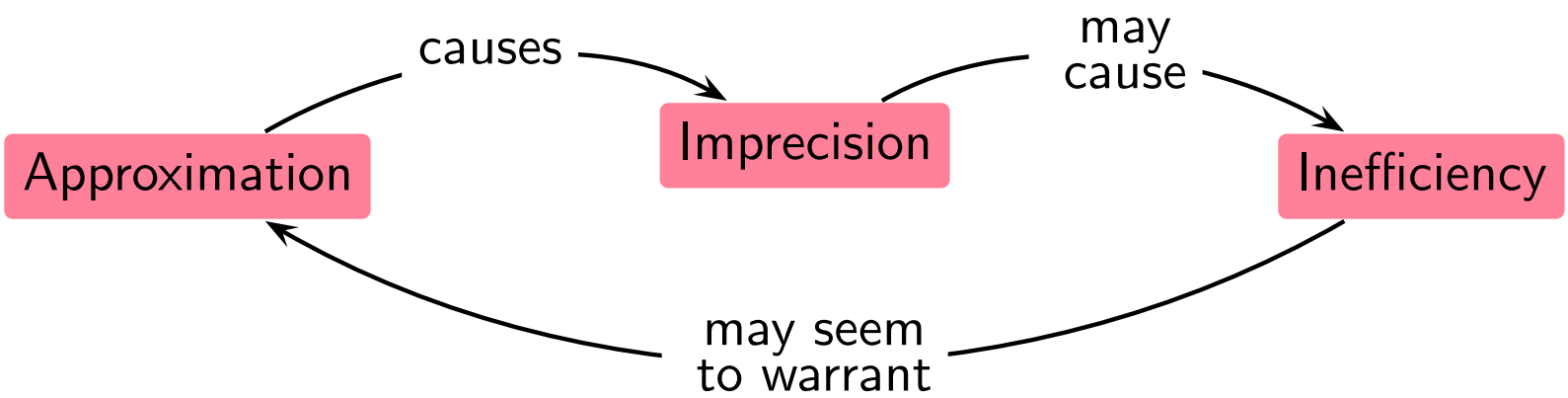
Why Avoid Approximations?

- Approximations may create a vicious cycle



Why Avoid Approximations?

- Approximations may create a vicious cycle



- Two examples of inefficiency cause by approximations
  - k-limited call strings may create "butterfly cycles" causing spurious fixed point computations [Hakjoo, 2010]
  - Imprecision in function pointer analysis overapproximates calls

    may create spurious recursion in call graphs

Which Approximations Should We Avoid?

Approximation	Admits
Flow insensitivity	
Context insensitivity (or partial context sensitivity)	
Imprecision in call graphs	
Allocation site based heap abstraction	

Which Approximations Should We Avoid?

Approximation	Admits
Flow insensitivity	Spurious intraprocedural paths
Context insensitivity (or partial context sensitivity)	Spurious interprocedural paths
Imprecision in call graphs	Spurious call sequences
Allocation site based heap abstraction	Spurious paths in memory graph

Flow Insensitivity in Data Flow Analysis

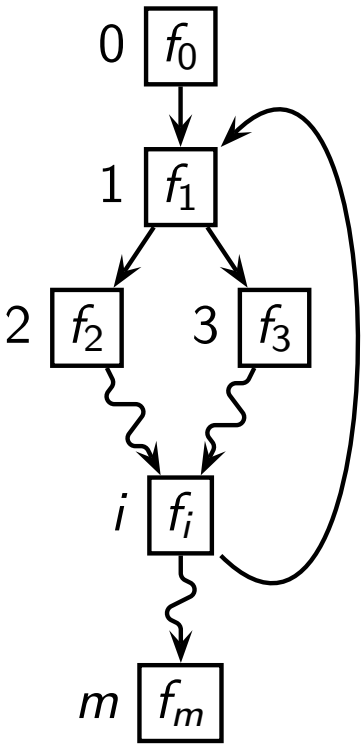
- Assumption: Statements can be executed in any order.
- Instead of computing point-specific data flow information, summary data flow information is computed.

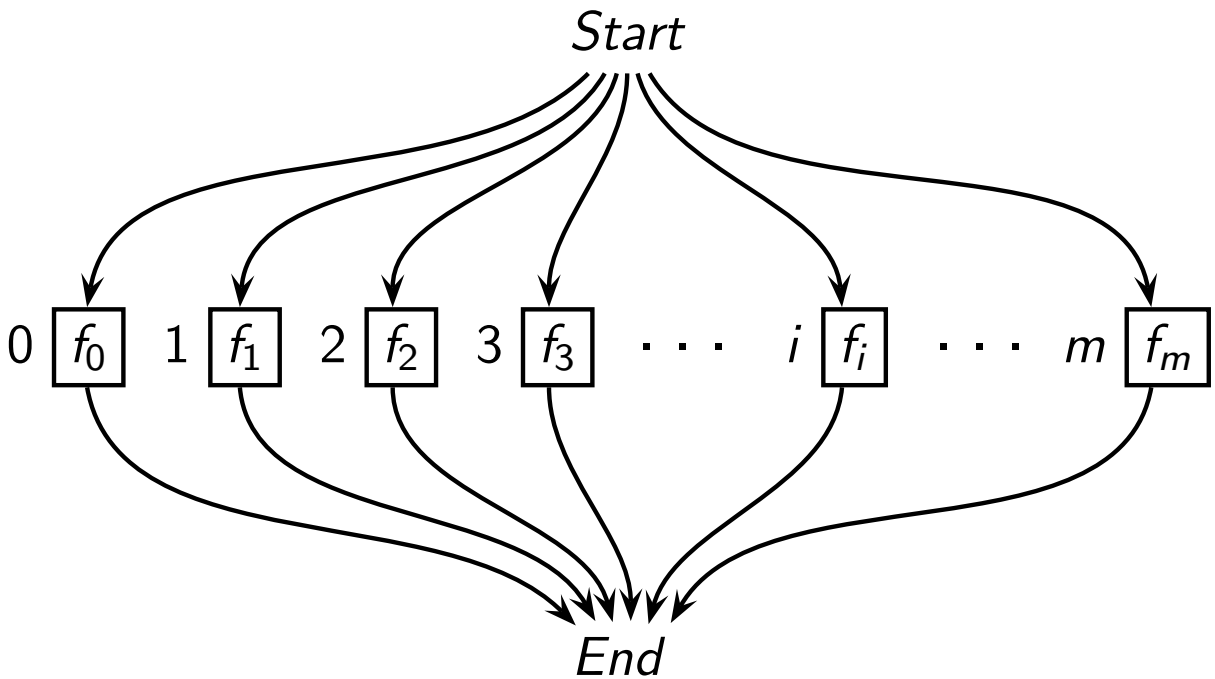
  The summary information is required to be a safe approximation of point-specific information for each point.
- No data flow information is killed

  If a statement kills data flow information, there is an alternate path that excludes the statement.

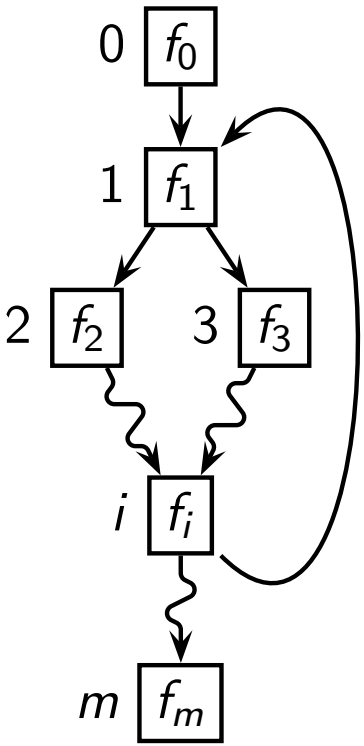
The control flow graph viewed as a complete graph (except for the Start and End nodes)

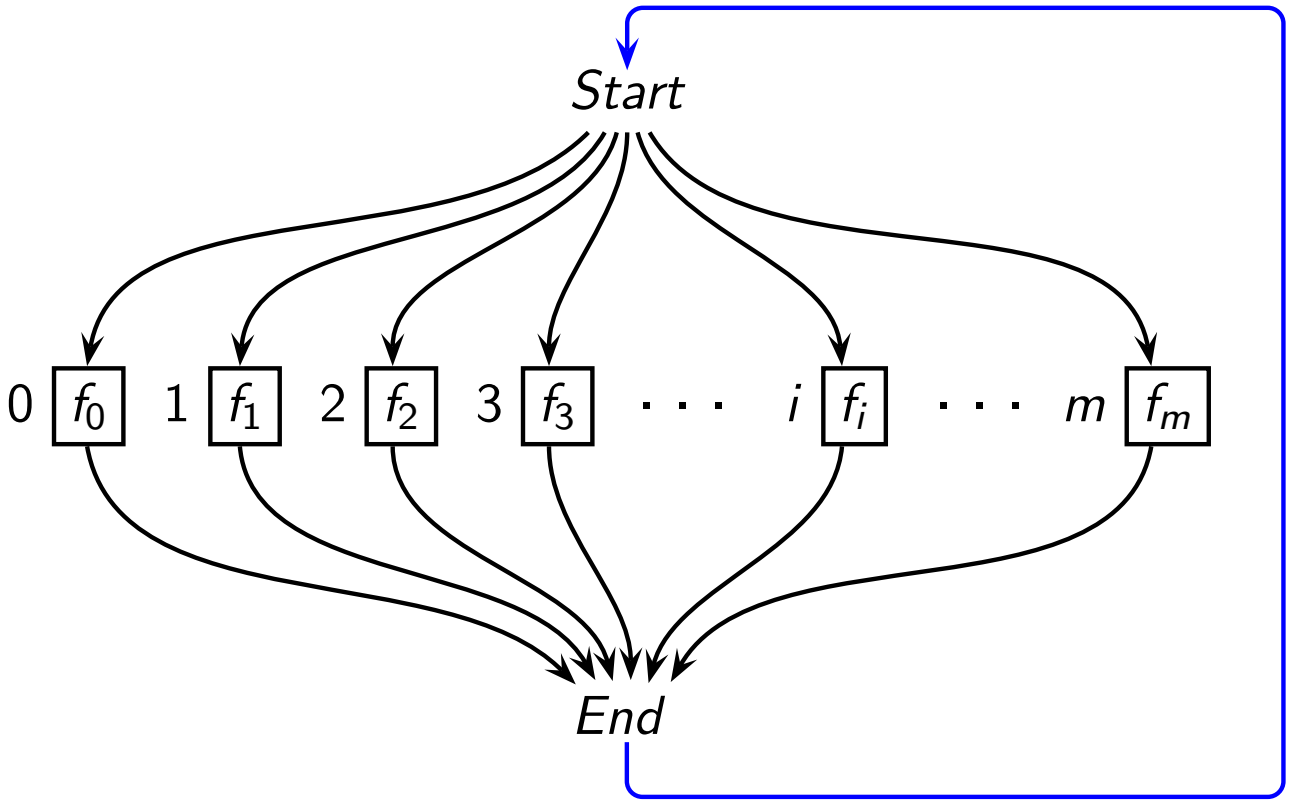
Flow Insensitivity in Data Flow Analysis



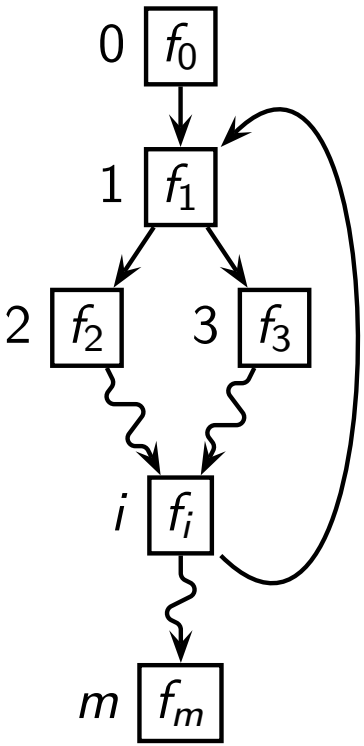


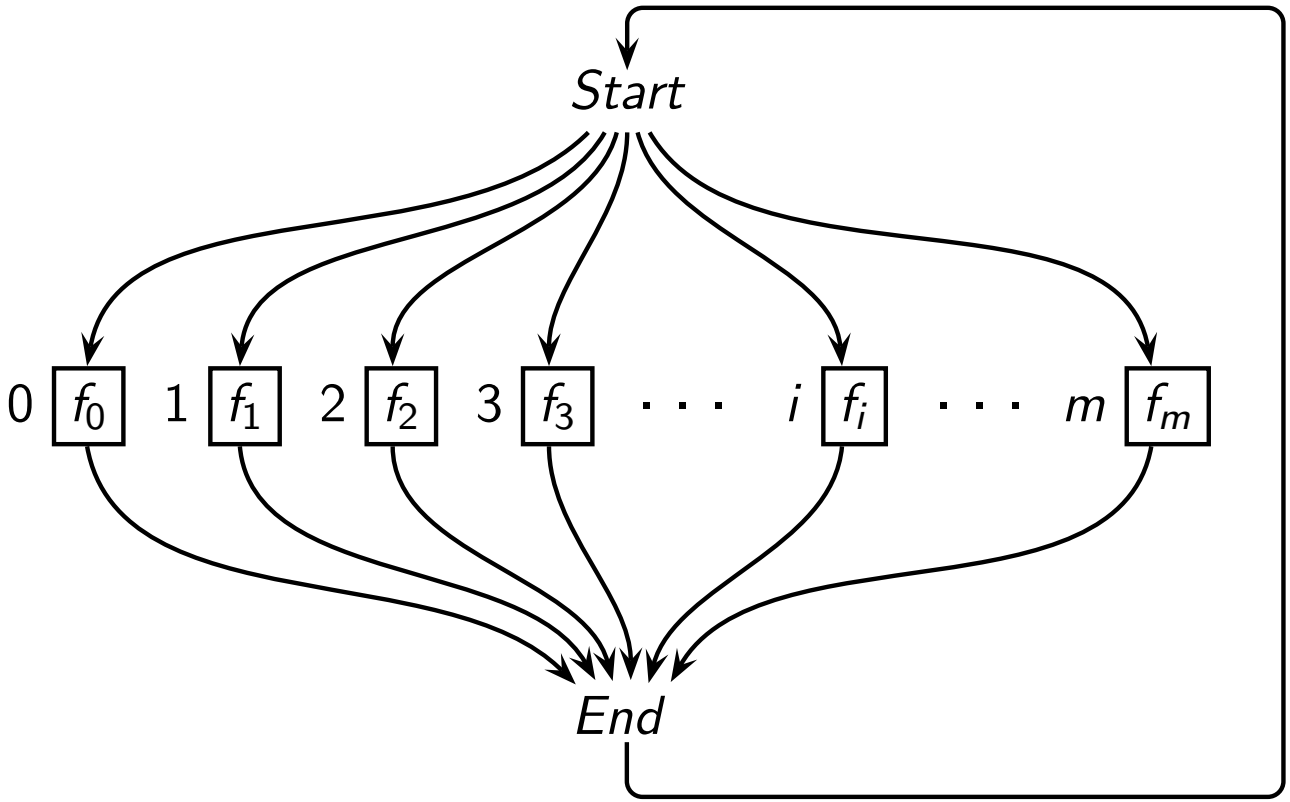
Flow Insensitivity in Data Flow Analysis





Flow Insensitivity in Data Flow Analysis

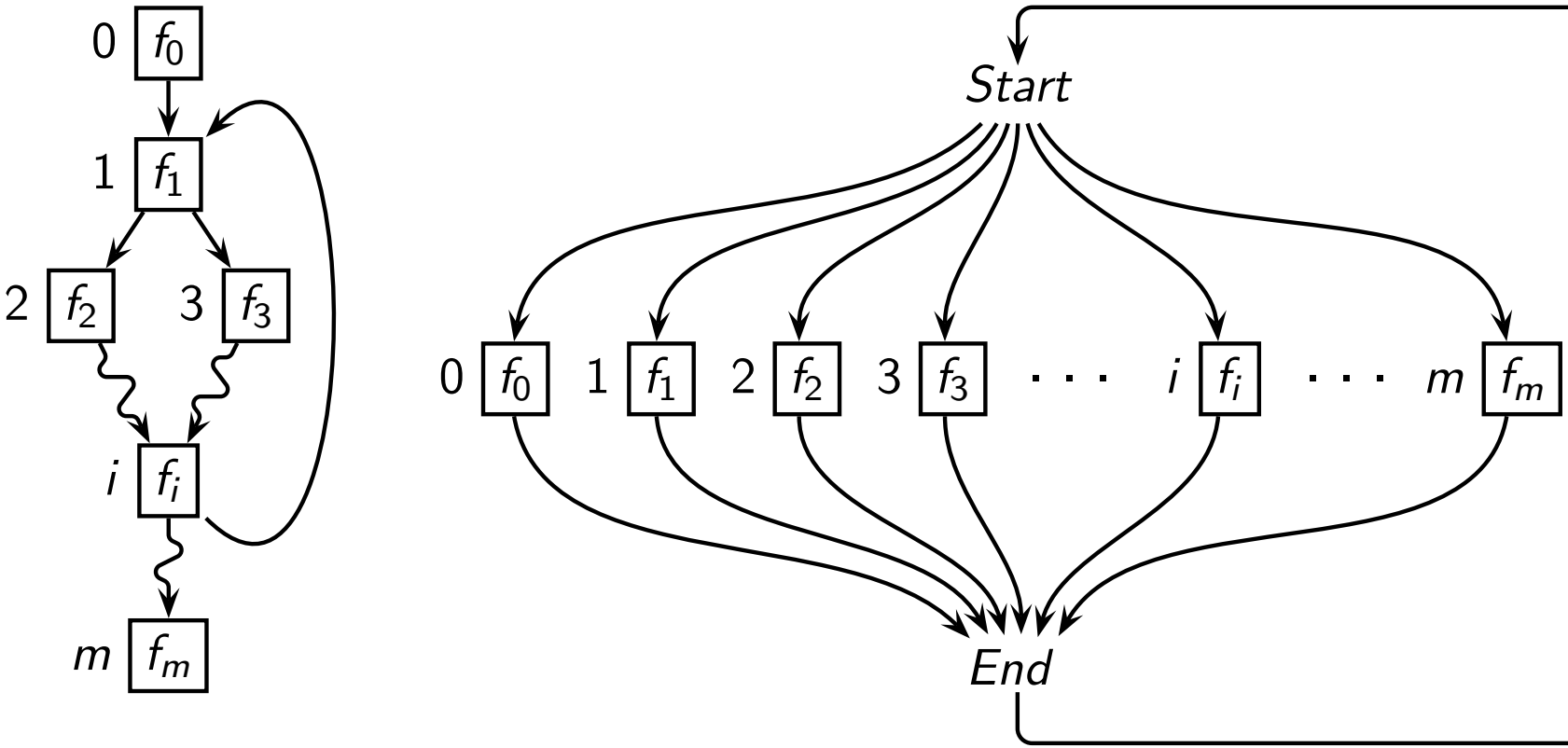




Allows arbitrary compositions of flow functions in any order

$\Rightarrow$ *Flow insensitivity*

Flow Insensitivity in Data Flow Analysis



In practice, dependent constraints are collected in a global repository in one pass and then are solved independently

Examples of Flow Insensitive Analyses

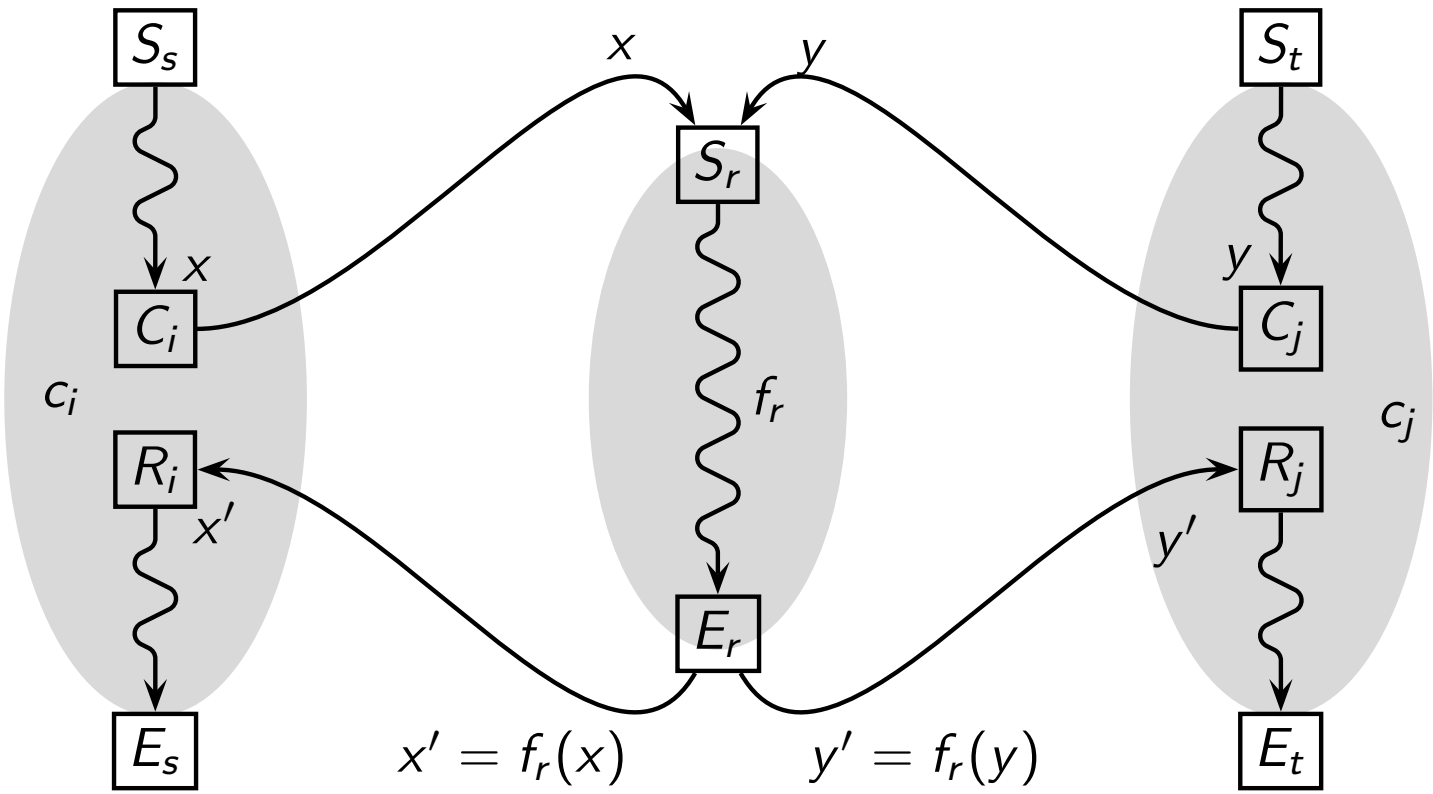
- Type checking/inferencing

  (What about interpreted languages?)
- Address taken analysis

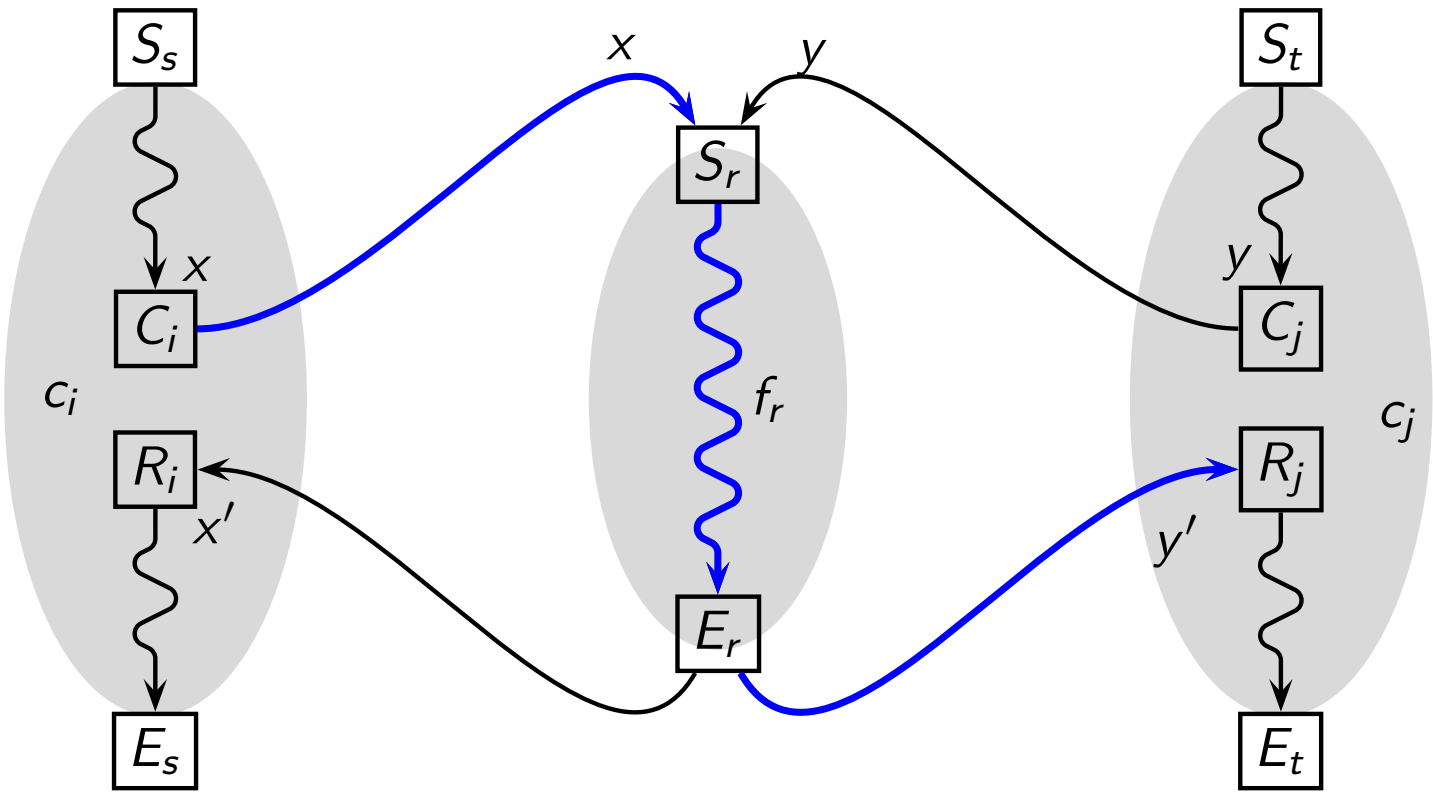
  Which variables have their addresses taken?
- Side effects analysis

  Does a procedure modify a global variable? Reference Parameter?

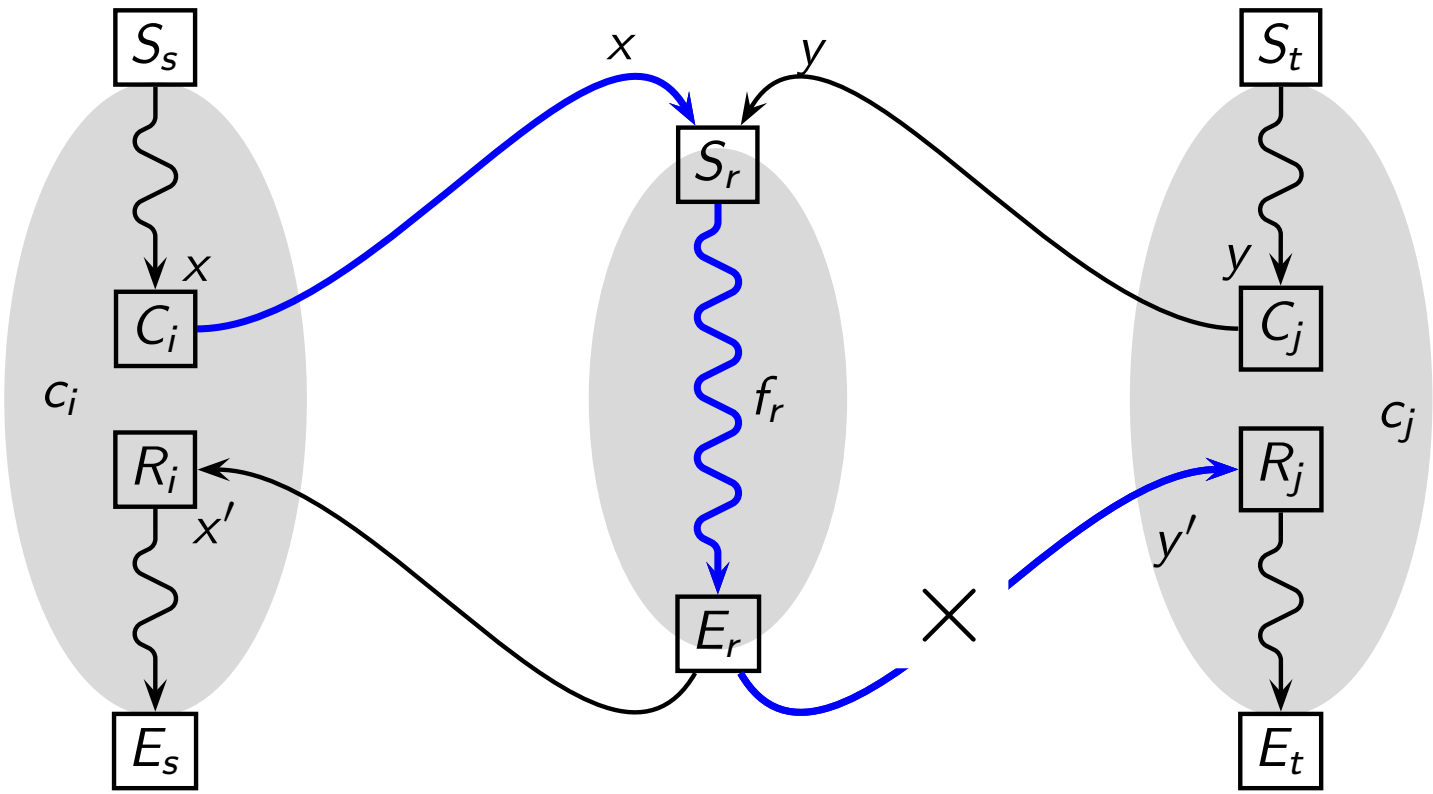
Context Sensitivity in Interprocedural Analysis



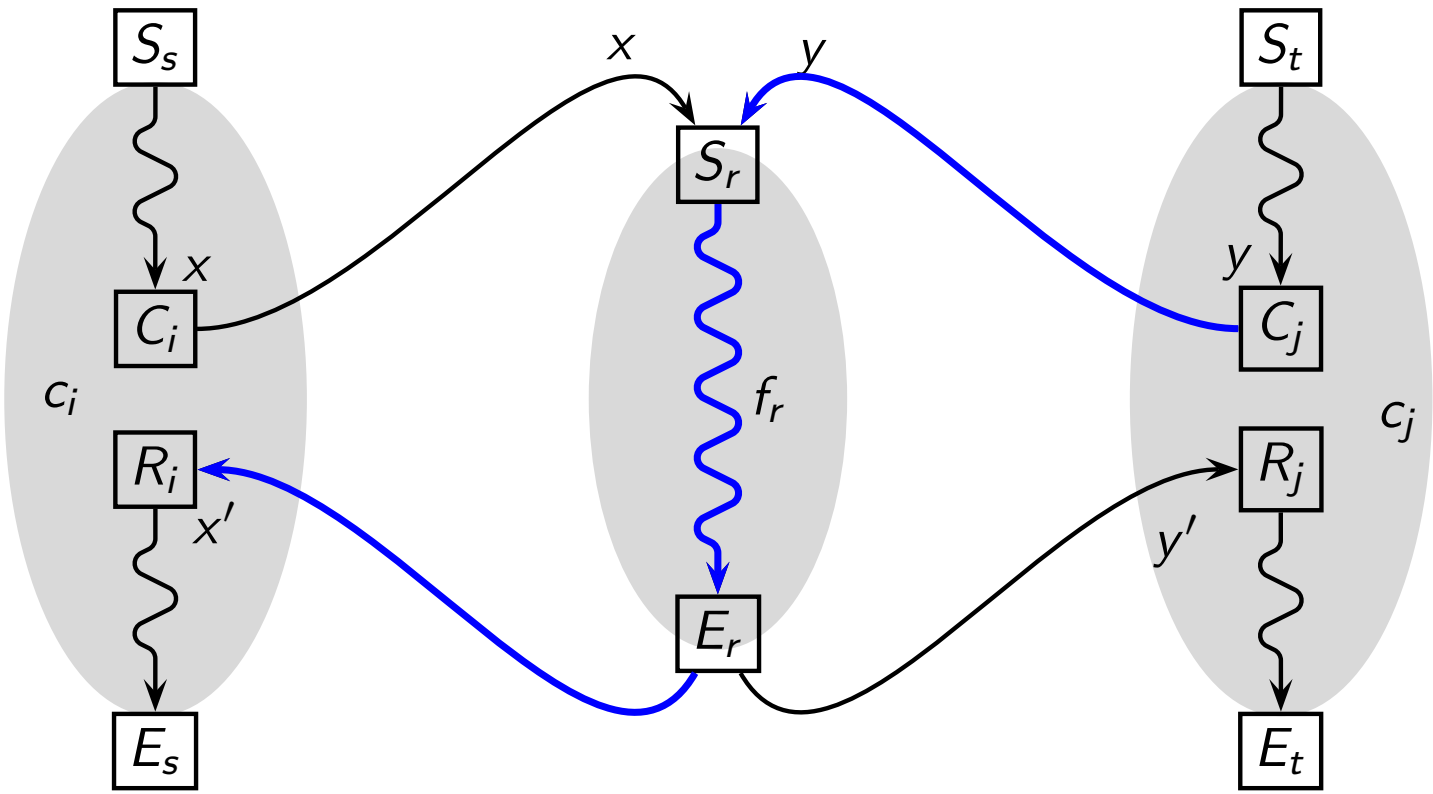
Context Sensitivity in Interprocedural Analysis



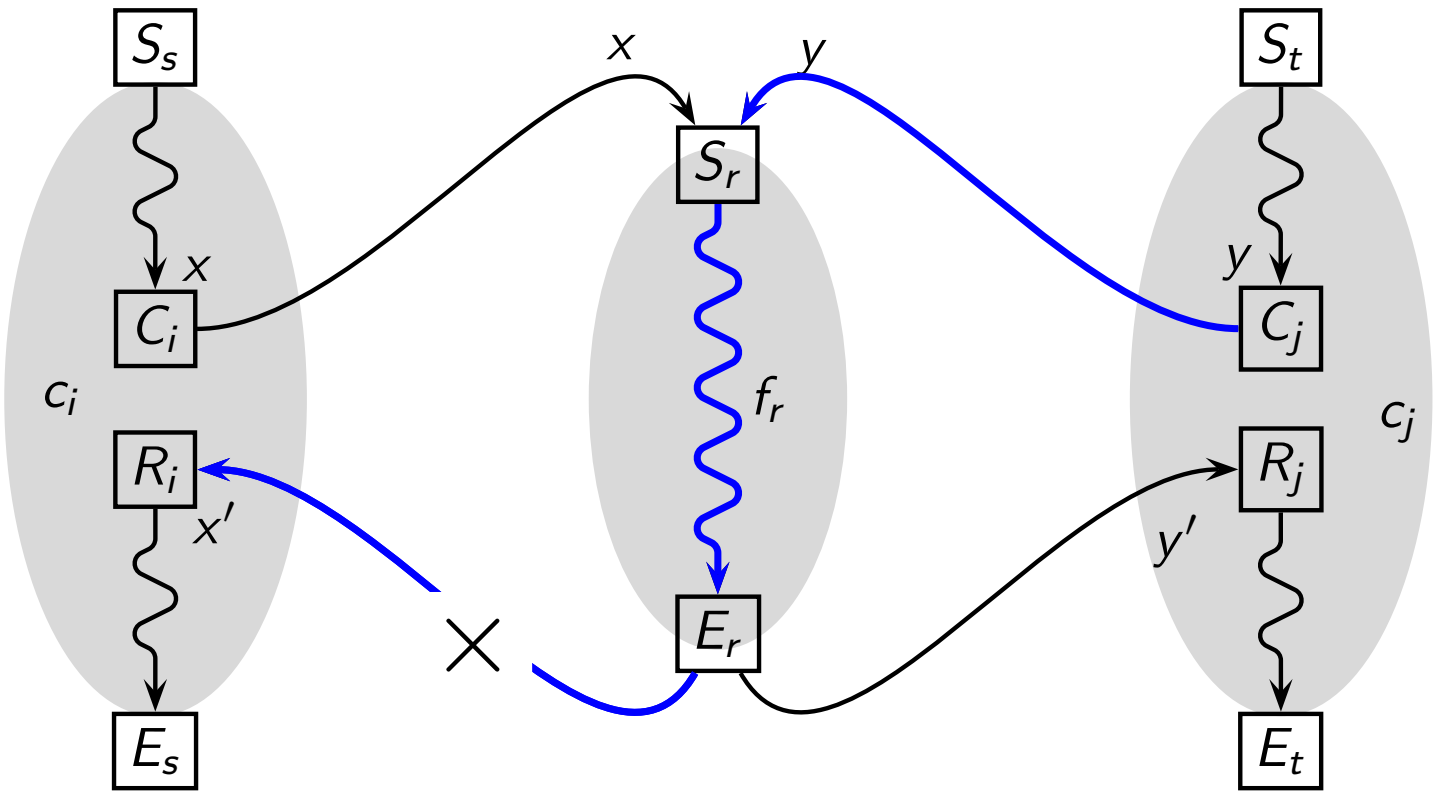
Context Sensitivity in Interprocedural Analysis



Context Sensitivity in Interprocedural Analysis



Context Sensitivity in Interprocedural Analysis



The Classical Precision-Efficiency Dilemma

Abstraction	Role in precision	Cause of inefficiency
	Distinguishes between	Needs to consider
Flow sensitivity		
Context sensitivity		
Precise heap abstraction		
Precise call structure		

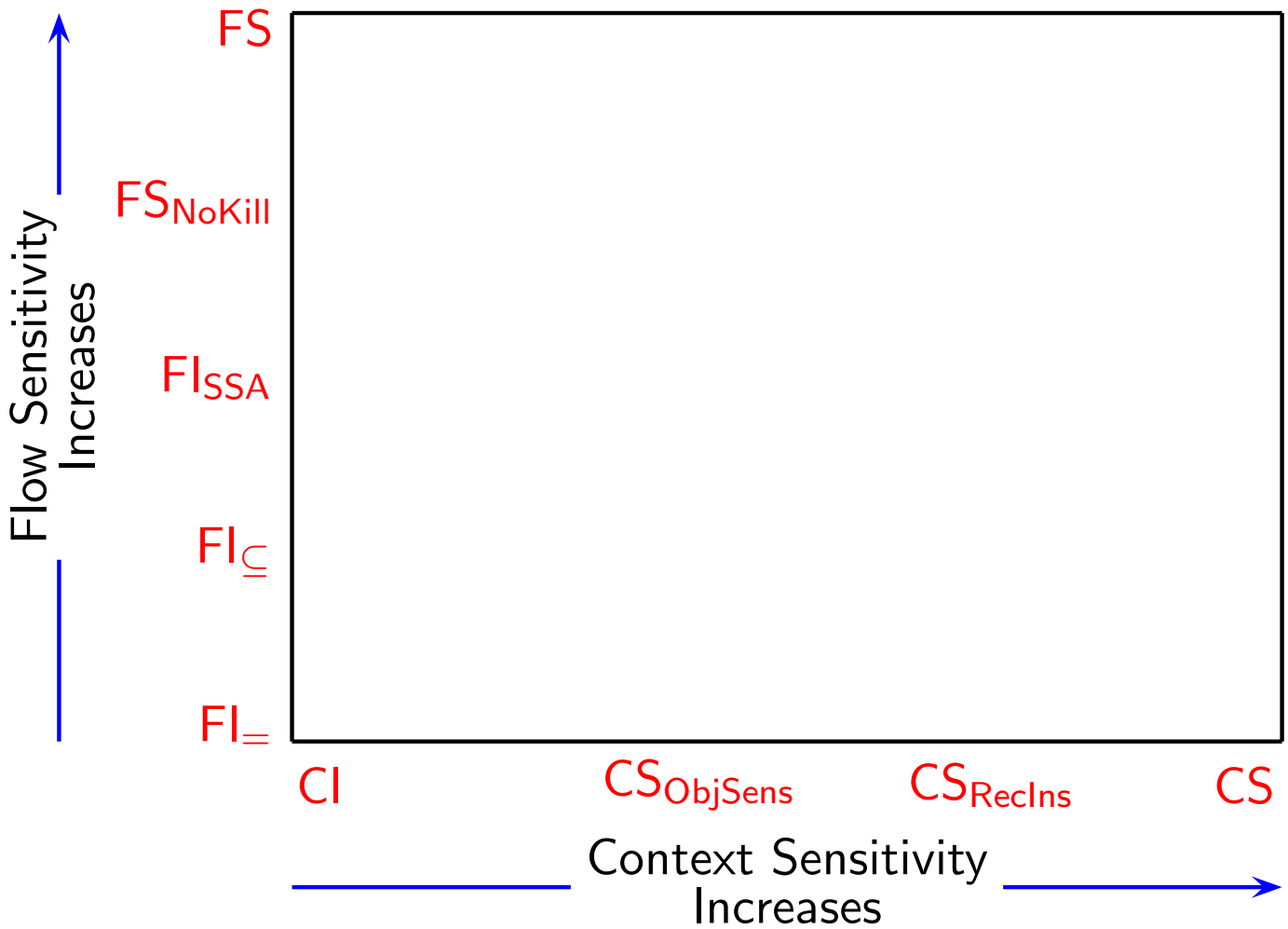
The Classical Precision-Efficiency Dilemma

Abstraction	Role in precision	Cause of inefficiency
	Distinguishes between	Needs to consider
Flow sensitivity	Information at different program points	
Context sensitivity	Information in different contexts	
Precise heap abstraction	Different heap locations	
Precise call structure	Indirect calls made to different callees from the same program point	

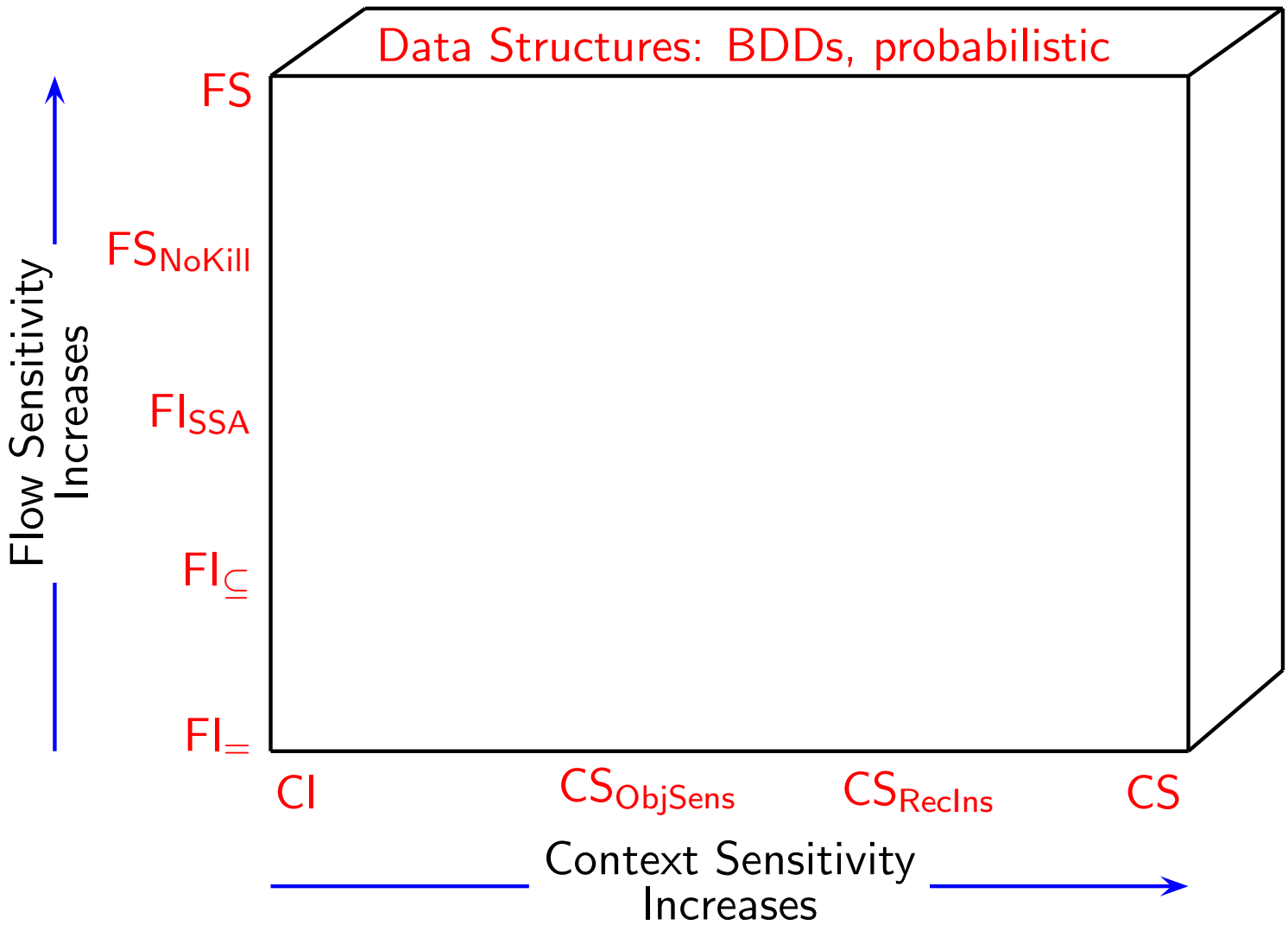
The Classical Precision-Efficiency Dilemma

Abstraction	Role in precision	Cause of inefficiency
	Distinguishes between	Needs to consider
Flow sensitivity	Information at different program points	A large number of program points
Context sensitivity	Information in different contexts	Exponentially large number of contexts
Precise heap abstraction	Different heap locations	Unbounded number of heap locations
Precise call structure	Indirect calls made to different callees from the same program point	Precise points-to information

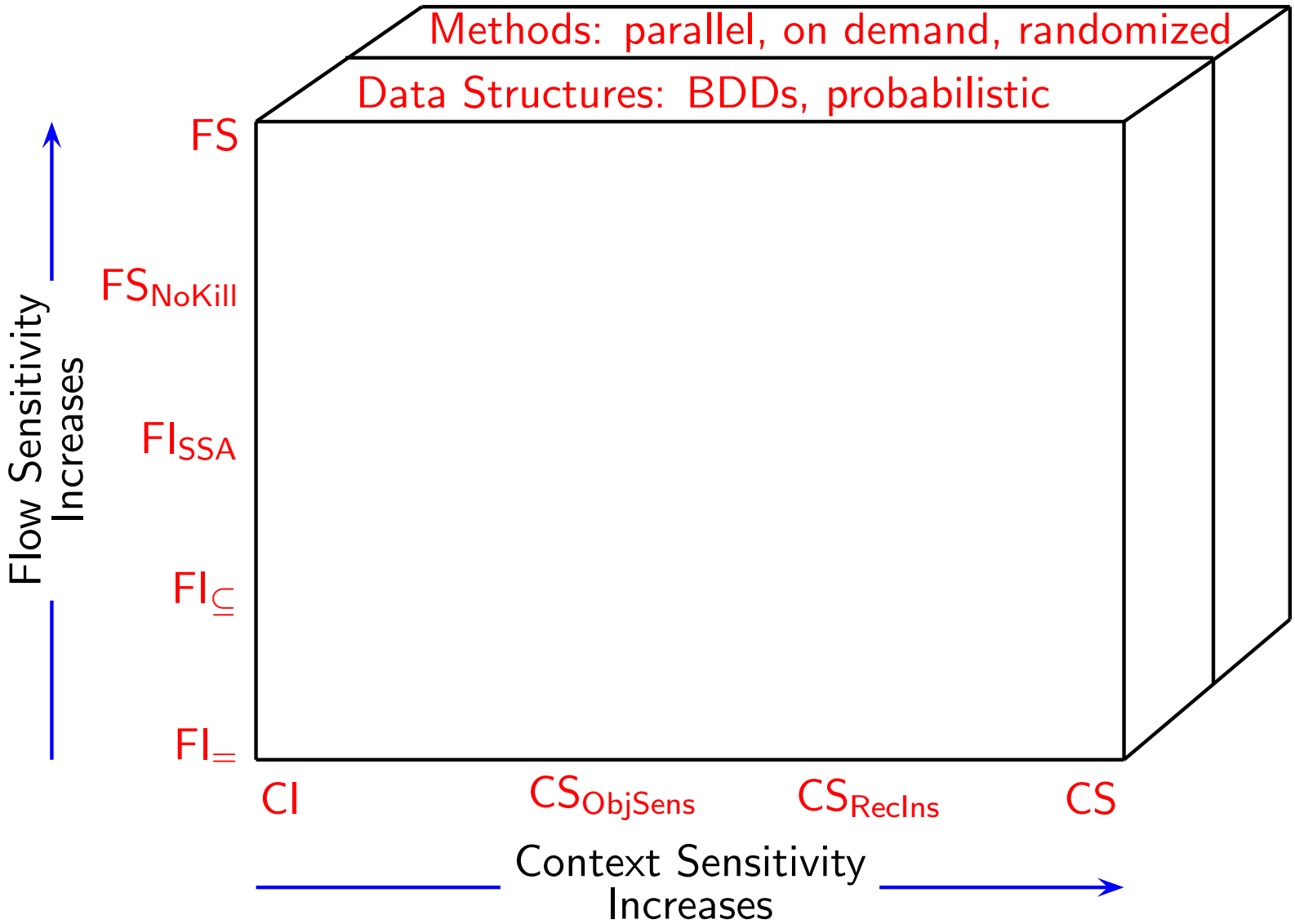
Pointer Analysis: An Engineer's Landscape



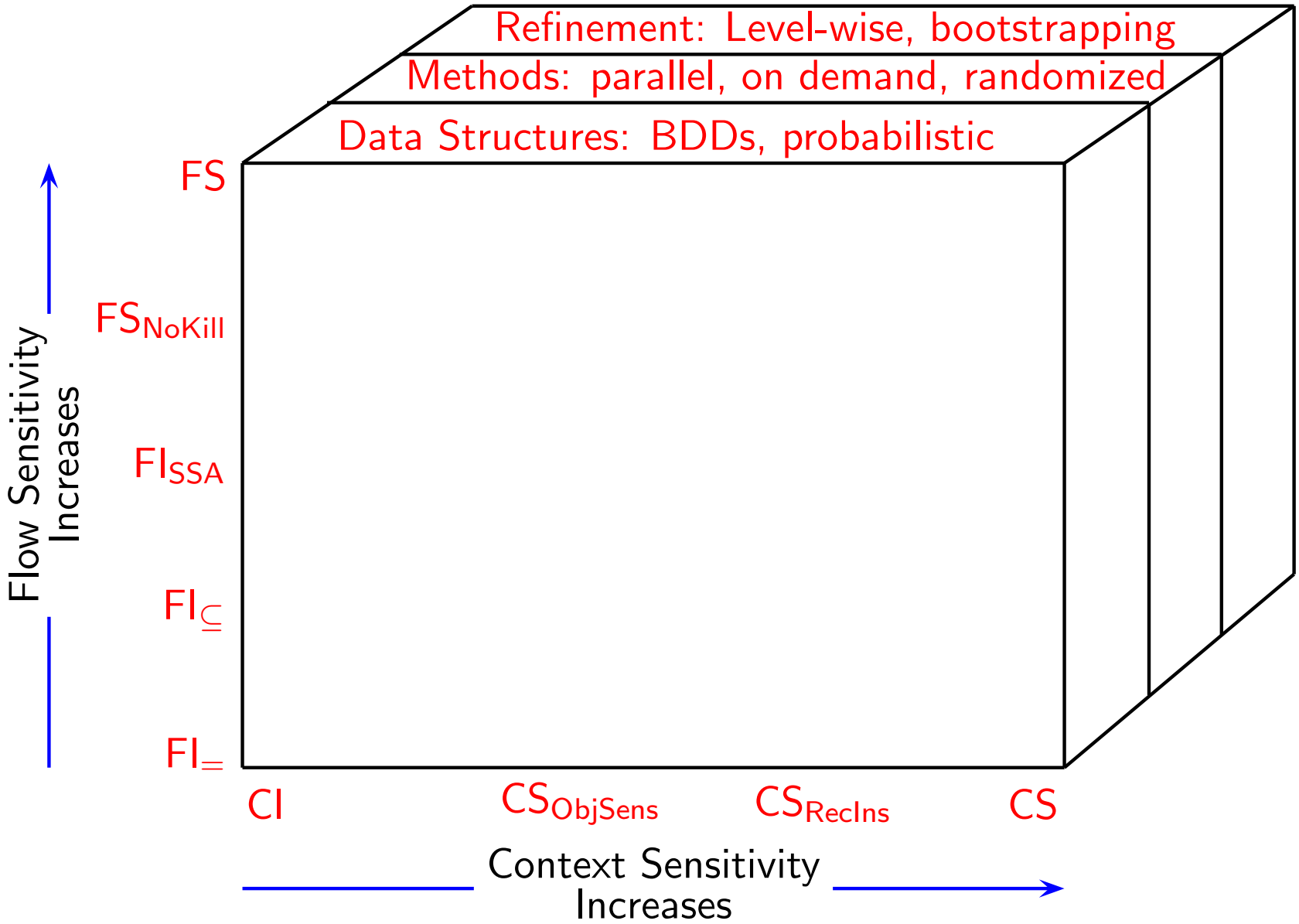
Pointer Analysis: An Engineer's Landscape



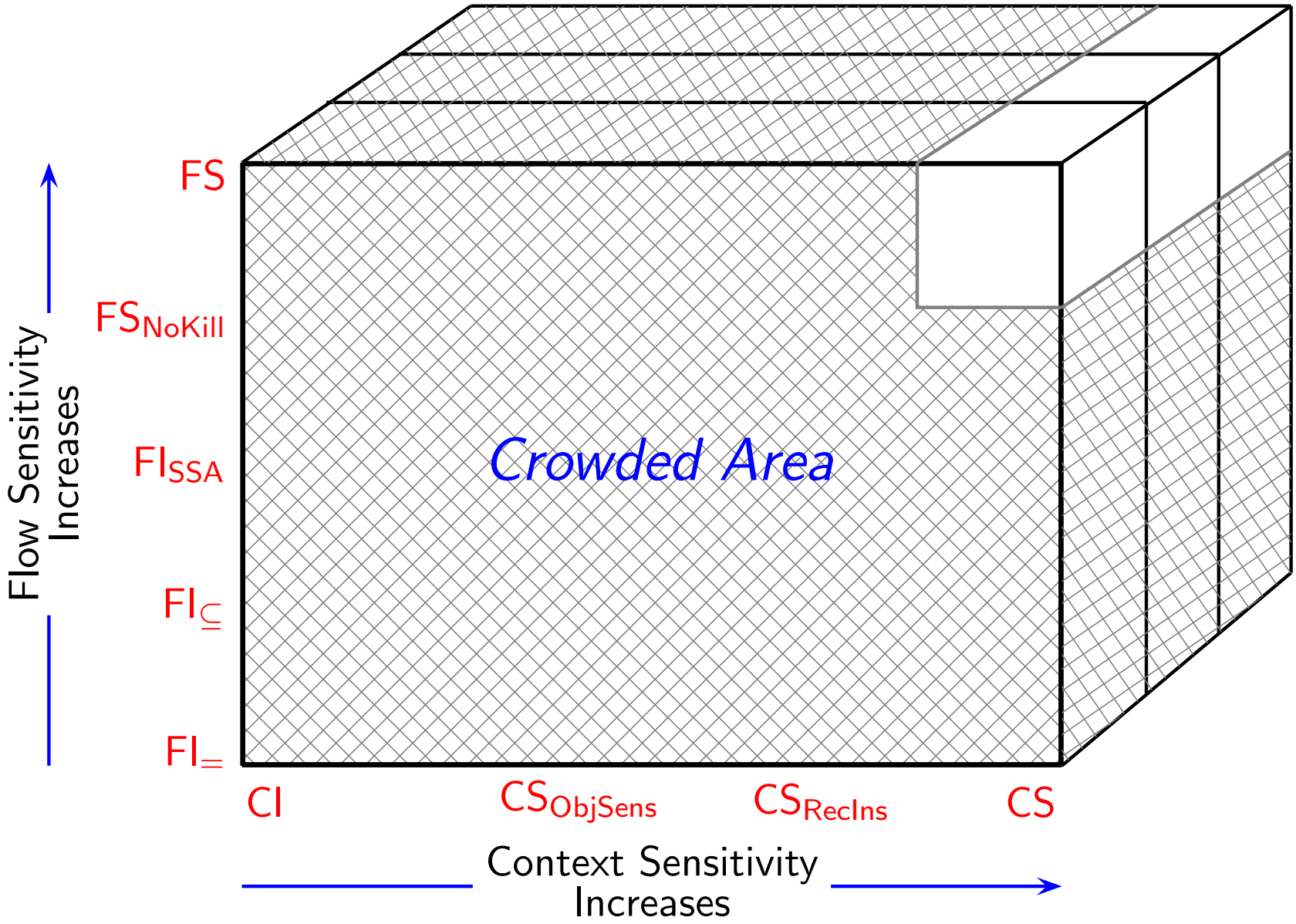
Pointer Analysis: An Engineer's Landscape



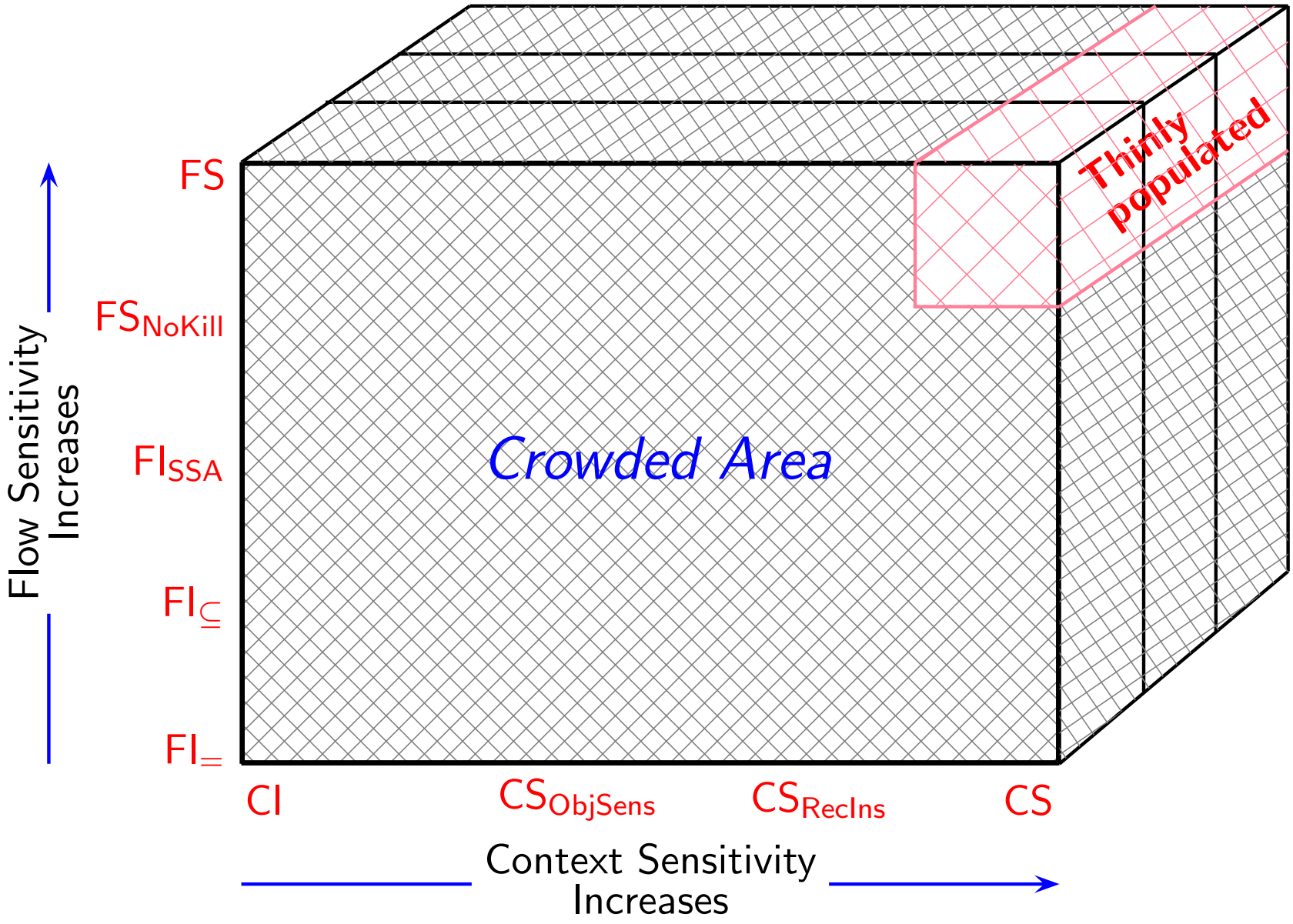
Pointer Analysis: An Engineer's Landscape



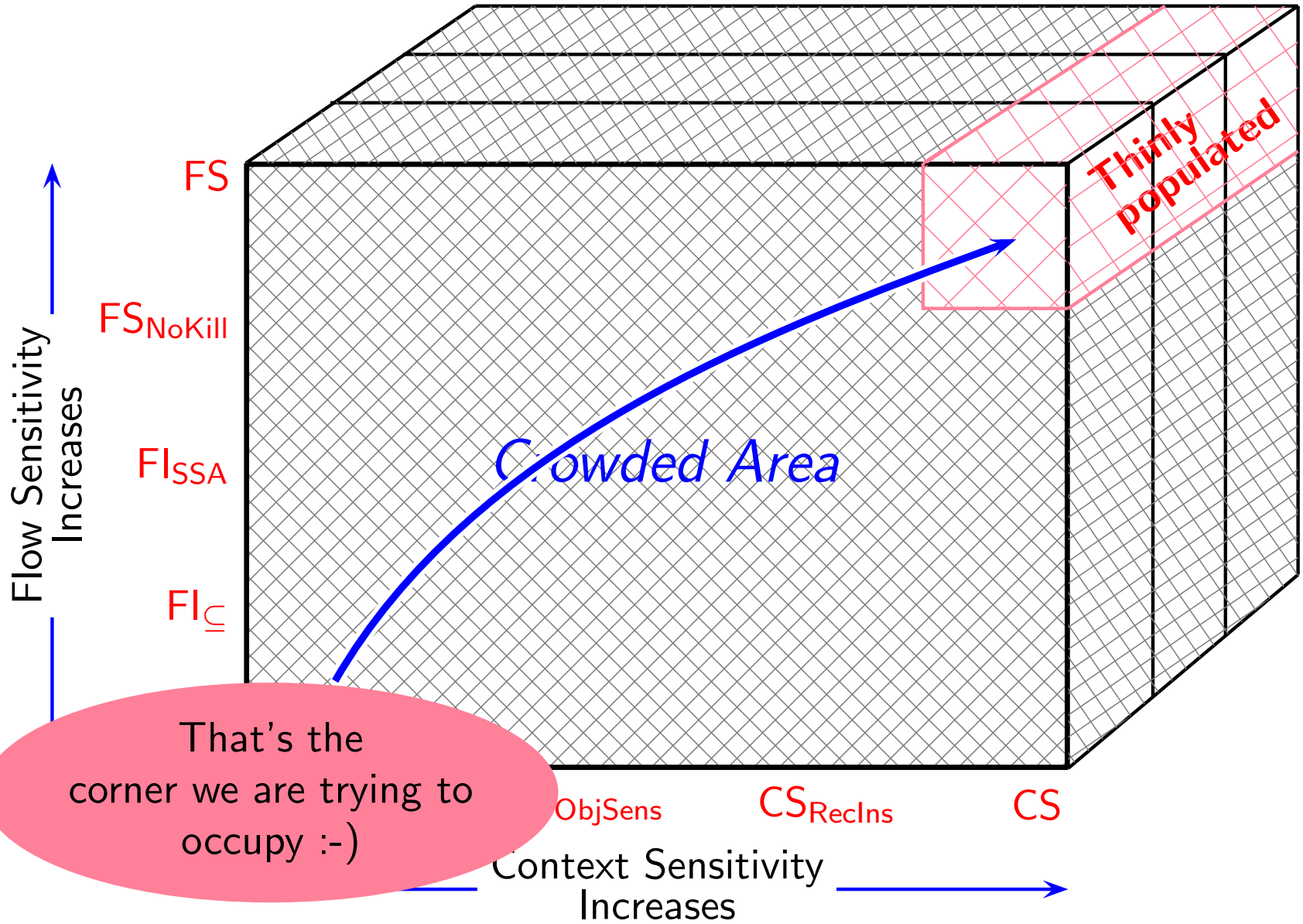
Pointer Analysis: An Engineer's Landscape



Pointer Analysis: An Engineer's Landscape



Pointer Analysis: An Engineer's Landscape



In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity		
Context sensitivity (Caller sensitivity)		
Precise heap abstraction		
Precise call structure		

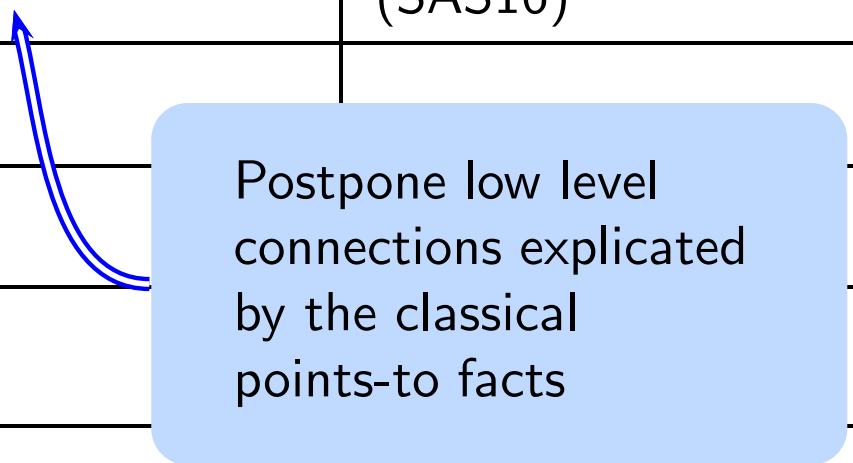
In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
Context sensitivity (Caller sensitivity)		
Precise heap abstraction		
Precise call structure		

Restrict the computation only to the usable data. Weave liveness discovery into the analysis

In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	High level abstraction of memory	Partial accomplishment (SAS16)
Context sensitivity (Caller sensitivity)		
Precise heap abstraction		
Precise call structure		



In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	High level abstraction of memory	Partial accomplishment (SAS16)
Context sensitivity (Caller sensitivity)	Value contexts	Mature accomplishment (CC08, SAS12, SOAP13)
Precise heap abstraction		
Precise call structure		

Distinguish between contexts by their data flow values and not their call chains

In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity	Joint liveness and points-to analysis	Partial accomplishment
	High level abstra of memory	ent
Context sensitivity (Caller sensitivity)	Value contexts	ent P13)
	GPG based bottom-up summary flow functions	Mature accomplishment (SAS16)
Precise heap abstraction		
Precise call structure		

Avoid recomputations for each context. Use a higher level abstraction of memory.

In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	High level abstraction of memory	Partial accomplishment (SAS16)
Context sensitivity (Caller sensitivity)	Value contexts	ent P13)
	GPG based bc summary flow	ent
Precise heap abstraction	Liveness access graphs	Partial accomplishment (TOPLAS07)
Precise call structure		

Identify the part of heap actually accessed in terms of patterns of accesses

In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	High level abstraction of memory	Partial accomplishment (SAS16)
Context sensitivity (Caller sensitivity)	Value contexts	Mature accomplishment (PLDI13)
	GPG based bo summary flow	ent
Precise heap abstraction	Liveness acces graphs	ent
	Access based abstraction	Mature accomplishment (ISMM17)
Precise call structure		

Distinguish between heap locations based on how they are accessed and not how they are allocated

In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	High level abstraction of memory	Partial accomplishment (SAS16)
Context sensitivity (Caller sensitivity)	Value contexts	Mature accomplishment (CC08, SAS12, SOAP13)
	GPG based bottom-up summary flow functions	Mature accomplishment (SAS16)
Precise heap abstraction	Liveness access graphs	ent
	Access based abstraction	ent
Precise call structure	Callee sensitivity	Work in progress

Call strings record call *history*. We need to record call *future* also.

In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	High level abstraction of memory	Partial accomplishment (SAS16)
Context sensitivity (Caller sensitivity)	Value contexts	Mature accomplishment (CC08, SAS12, SOAP13)
	GPG based bottom-up summary flow functions	Mature accomplishment (SAS16)
Precise heap abstraction	Liveness acces graphs	ent
	Access based abstraction	ent
Precise call structure	Callee sensitivity	Work in progress
	Virtual call resolution	Work in progress

Make the call graph more precise by computing a more precise set of callees

In Search of Abstractions for Precision Without Inefficiency

Desired Abstraction	Enabling Abstraction	Status of our work
Flow sensitivity	Joint liveness and points-to analysis	Partial accomplishment (SAS12)
	High level abstraction of	Partial accomplishment (SAS16)
Context sensitivi (Caller sensitivi		ure accomplishment , SAS12, SOAP13)
		accomplishment)
Precise heap abstraction		l accomplishment OPLAS07)
	Acces abstraction	Mature accomplishment (ISMM17)
Precise call structure	Callee sensitivity	Work in progress
	Virtual call resolution	Work in progress

We are destined to a long haul with no guarantees :-)