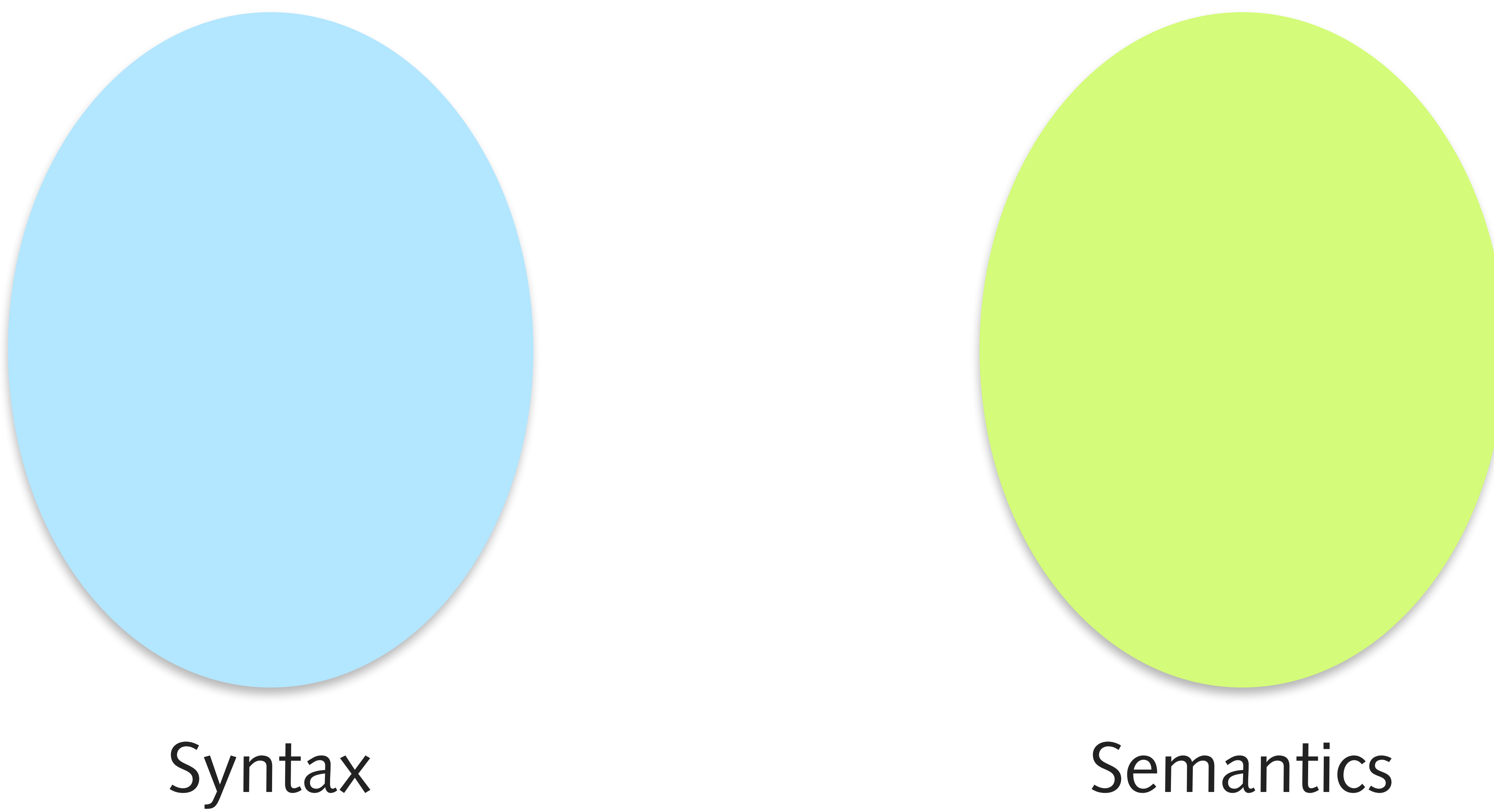
Scaling Program Synthesis by Exploiting Existing Code

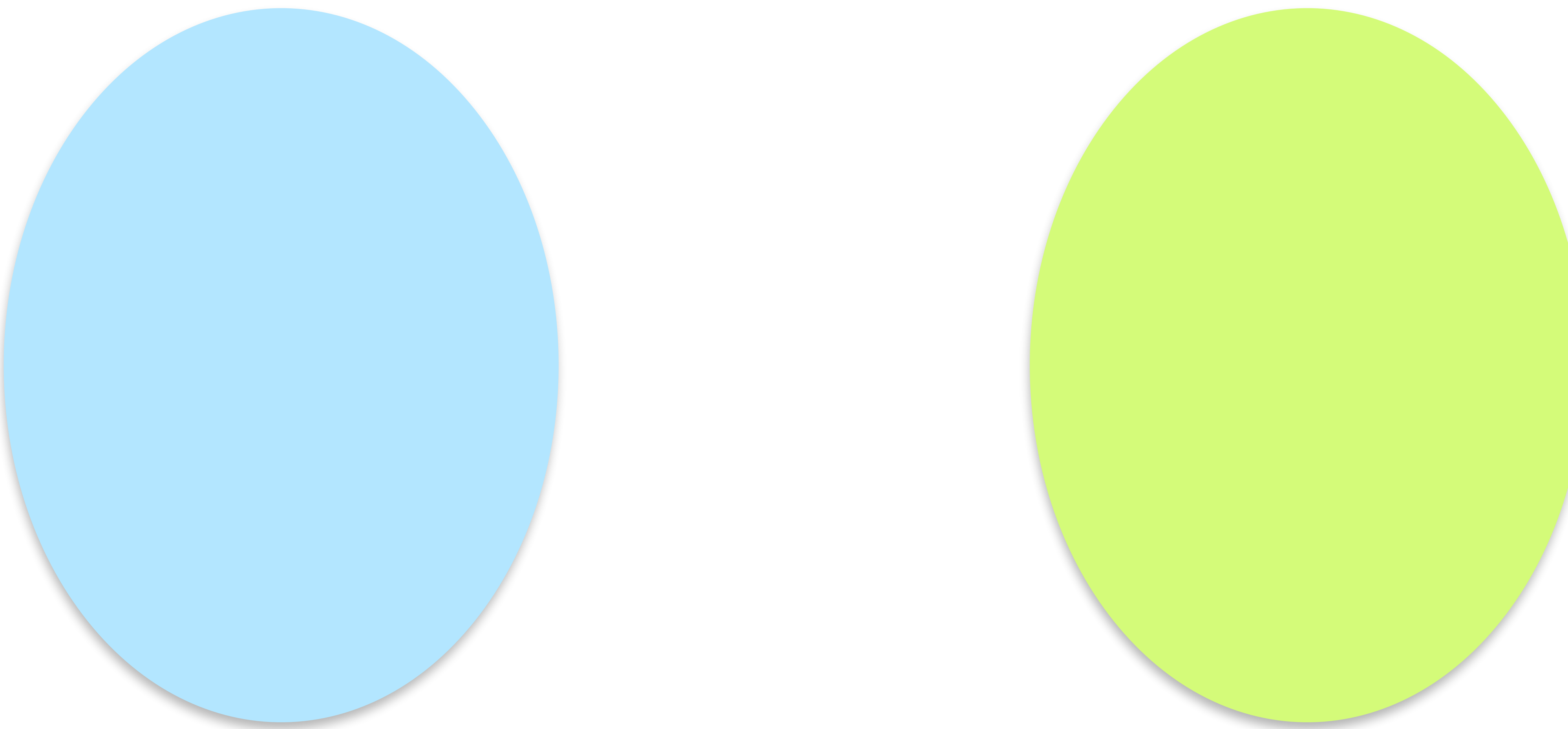
James Bornholt
Emina Torlak

University of Washington

Synthesis: write programs automatically



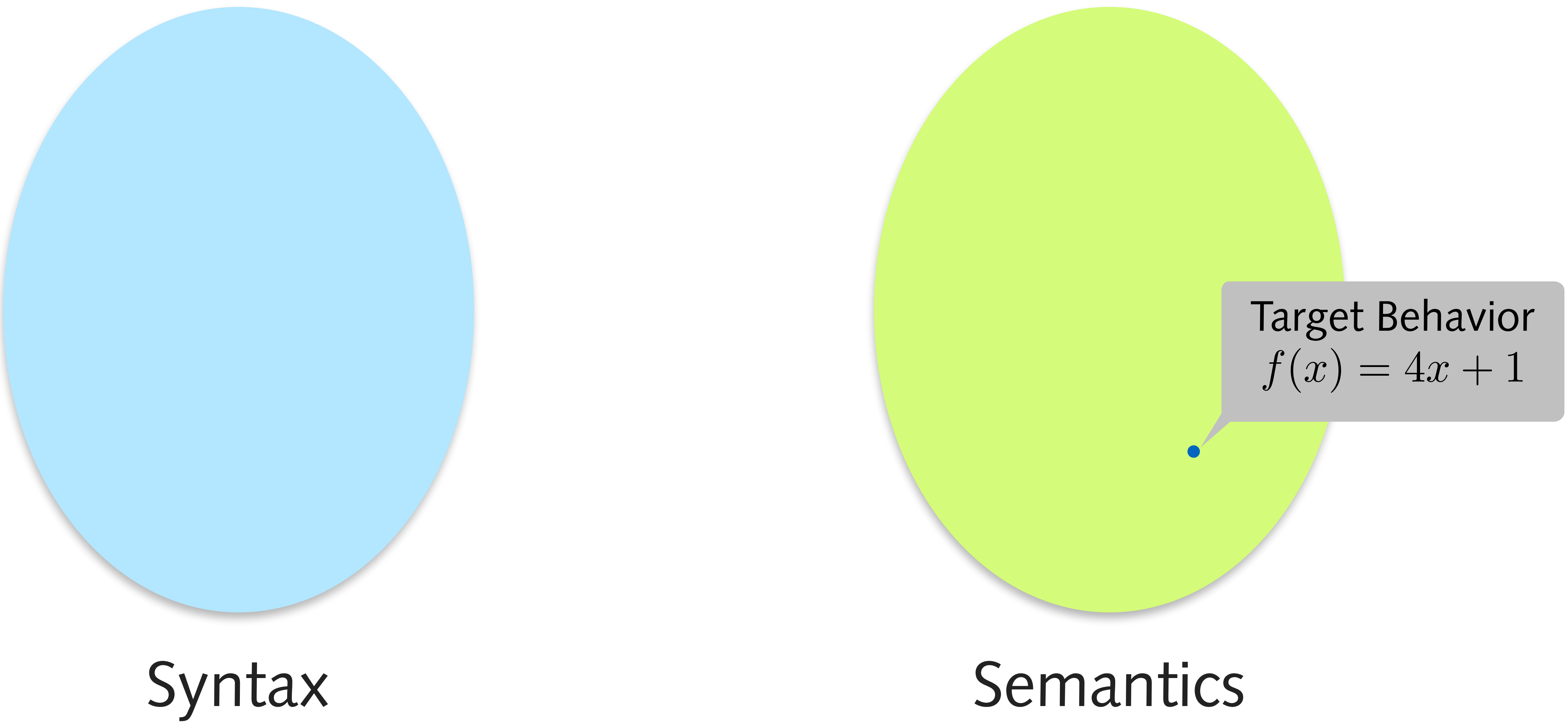
Synthesis: write programs automatically



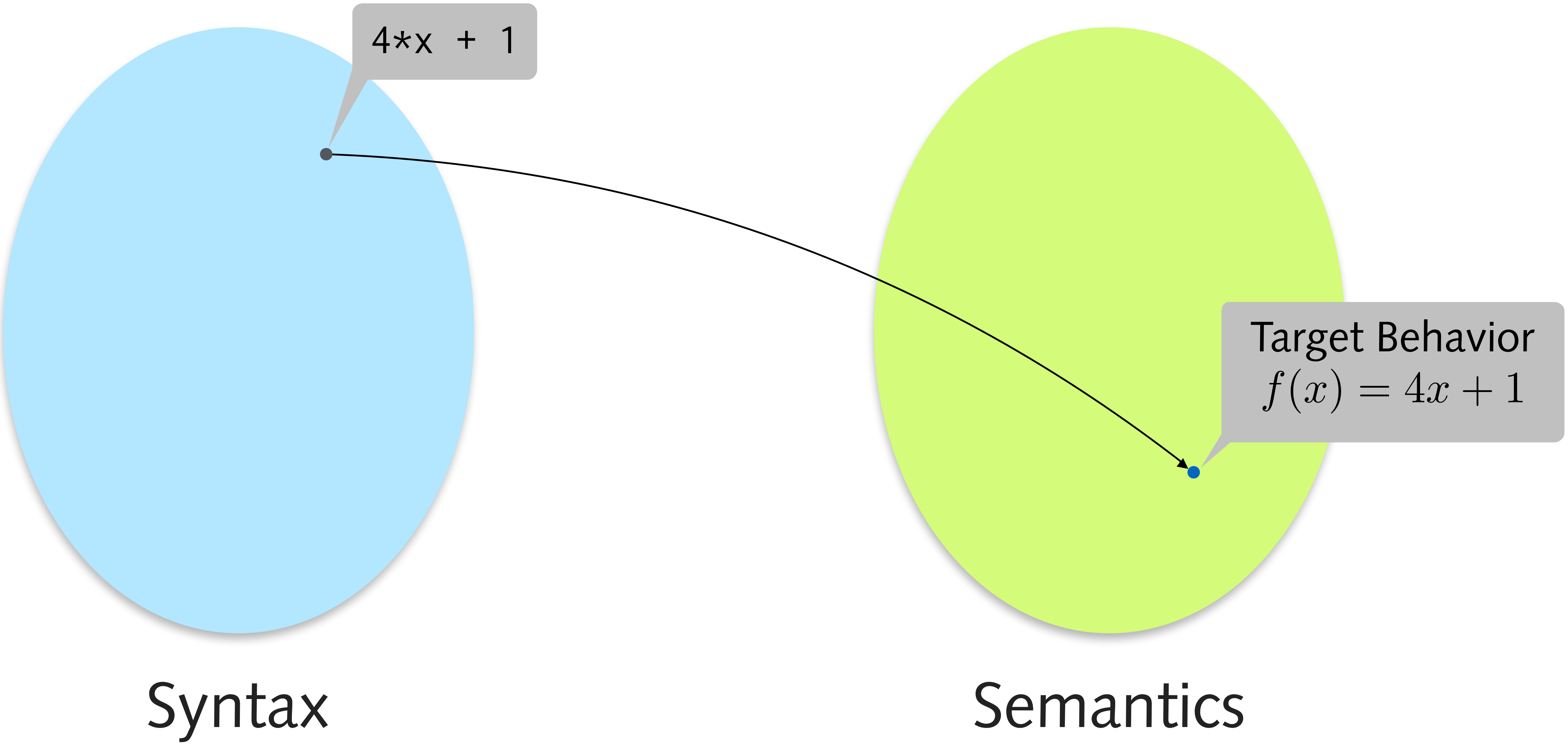
Syntax

Semantics

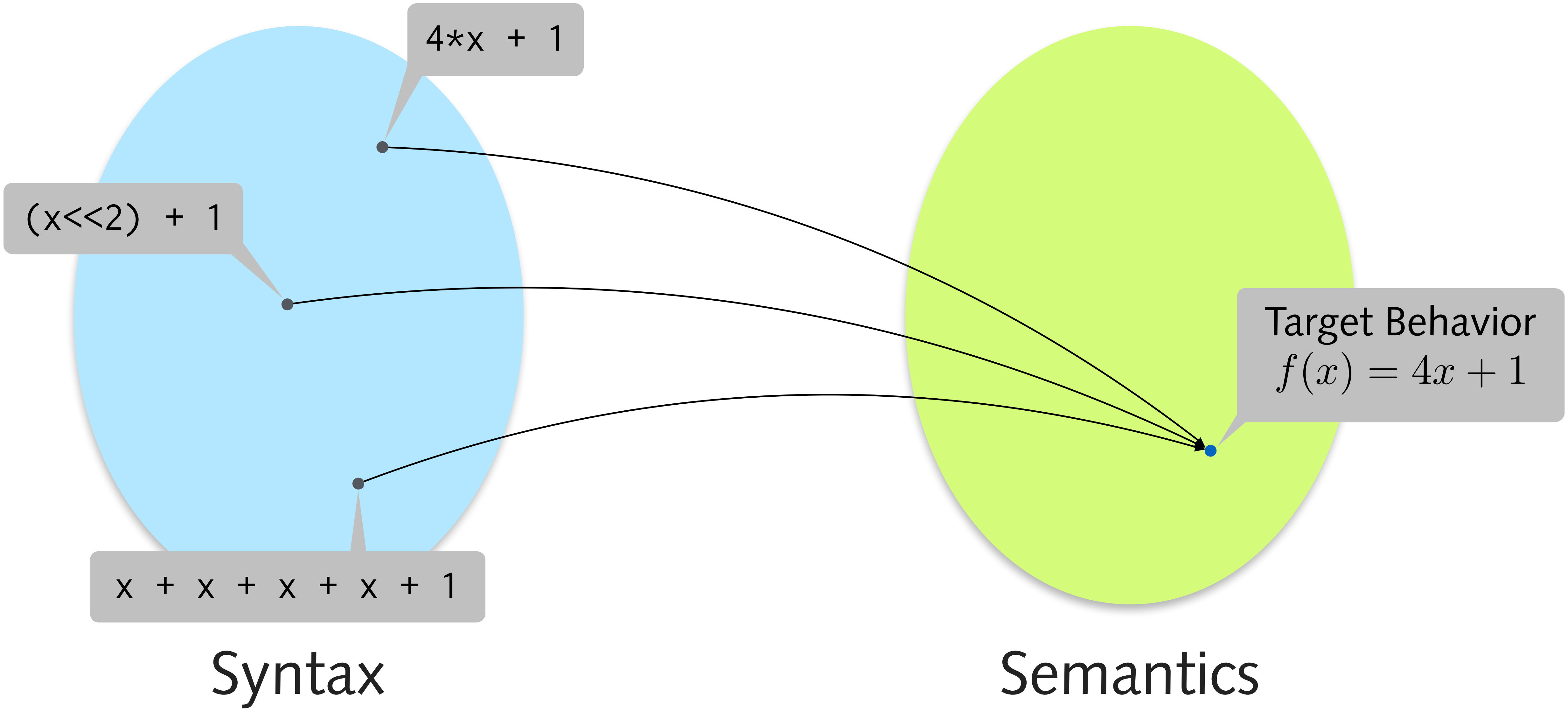
Synthesis: write programs automatically



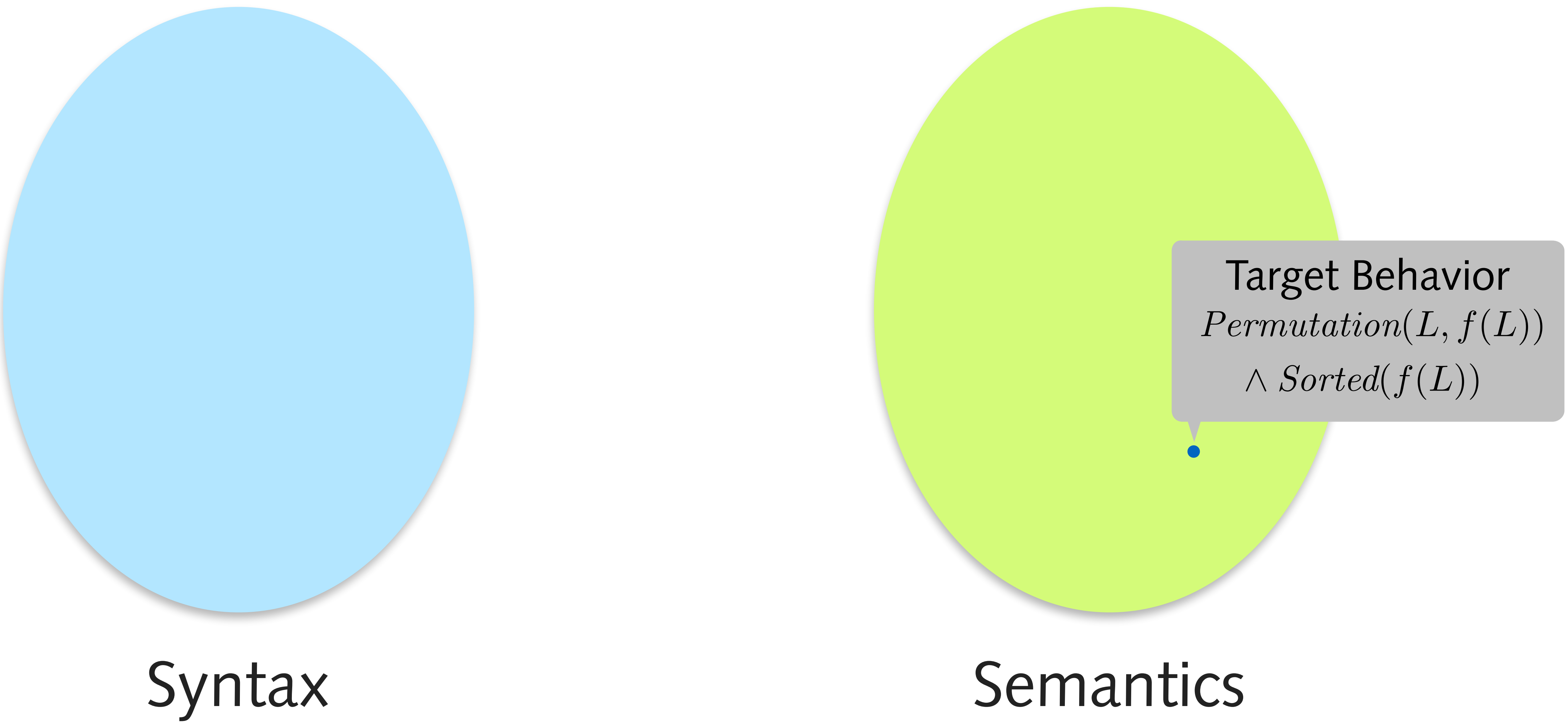
Synthesis: write programs automatically



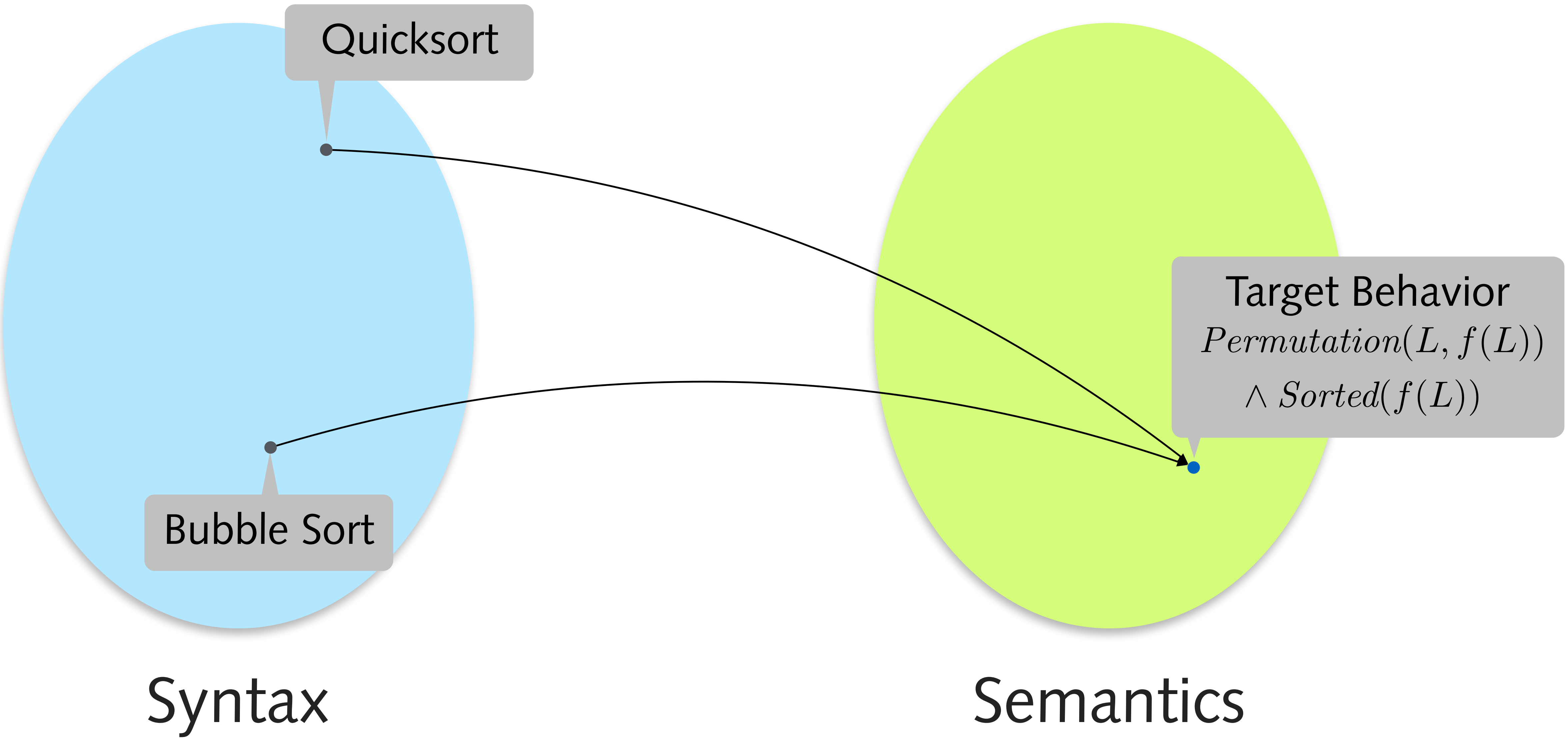
Synthesis: write programs automatically



Synthesis: write programs automatically



Synthesis: write programs automatically



The success of synthesis

End-user programming
by example [FlashFill, POPL'11]

Cache coherence protocols
[Transit, PLDI'13]

Parallel browser layout
engines [PPoPP'13]

Compilers for new spatial
architectures [Chlorophyll, PLDI'14]

The success of synthesis

End-user programming
by example [FlashFill, POPL'11]

Cach… …ls

e…

Synthesis
turns
high-level intent
into
low-level detail

Compilers for new spatial
architectures [Chlorophyll, PLDI'14]

The success of synthesis

Approximate Computing

quality bounds
into
approximate programs

End-user programming by example [FlashFill, POPL'11]

Synthesis
turns
high-level intent
into
low-level detail

Compilers for new spatial architectures [Chlorophyll, PLDI'14]

The success of synthesis

End-user programming
by example [FlashFill, POPL'11]

Synthesis
turns
high-level intent
into
low-level detail

Compilers for new spatial
architectures [Chlorophyll, PLDI'14]

Approximate Computing

quality bounds
into
approximate programs

Hardware Synthesis

programs
into
hardware designs

The success of synthesis

End-user programming by example [FlashFill, POPL'11]

Synthesis
turns
high-level intent
into
low-level detail

Compilers for new spatial architectures [Chlorophyll, PLDI'14]

Approximate Computing
quality bounds
into
approximate programs

Hardware Synthesis
programs
into
hardware designs

Black Box Systems
observed behaviors
into
specifications

Approximate Computing

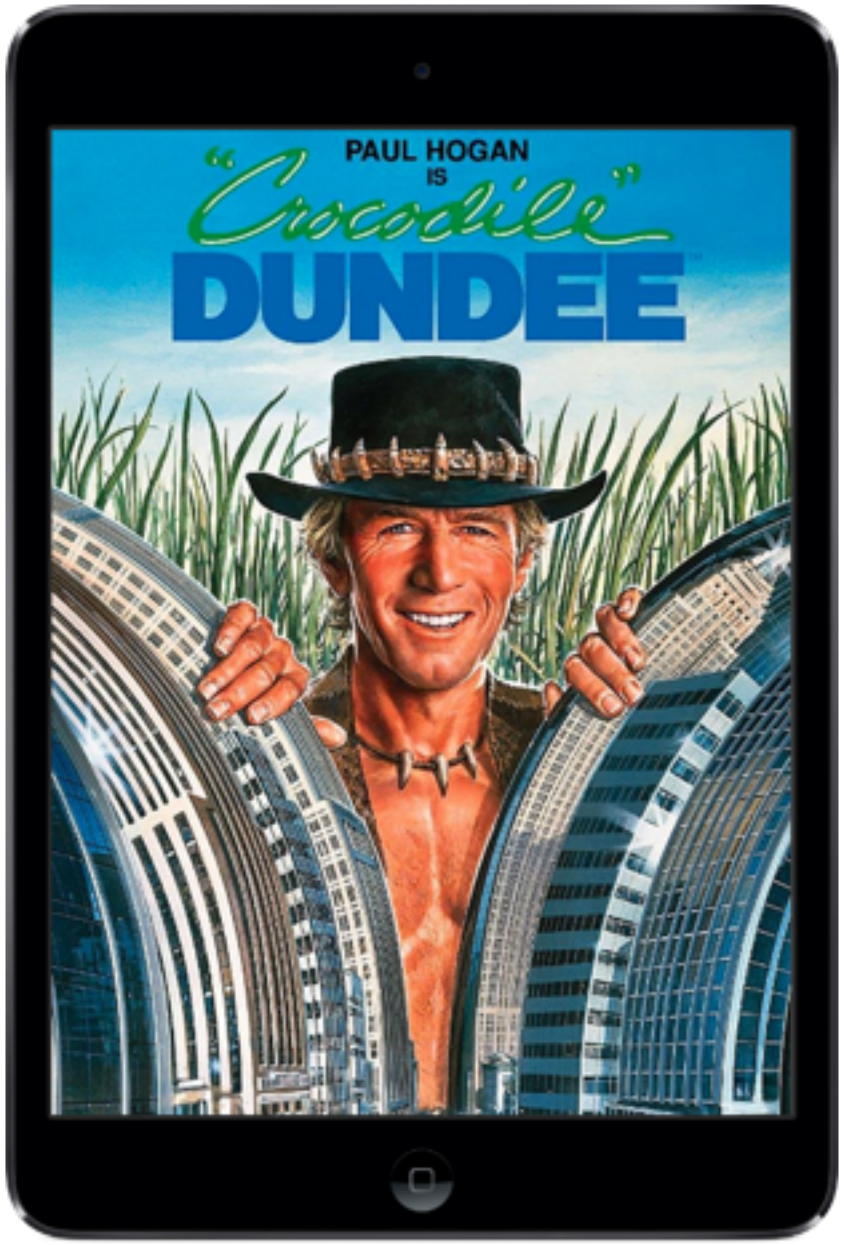
quality bounds
into
approximate programs

Not all applications require perfect accuracy

Approximate Computing

quality bounds
into
approximate programs

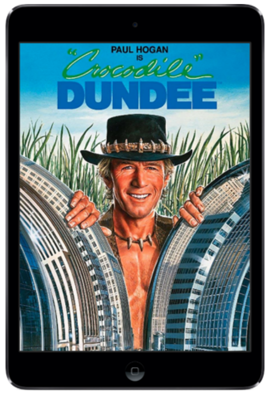
Not all applications require perfect accuracy



Approximate Computing

quality bounds
into
approximate programs

Not all applications require perfect accuracy

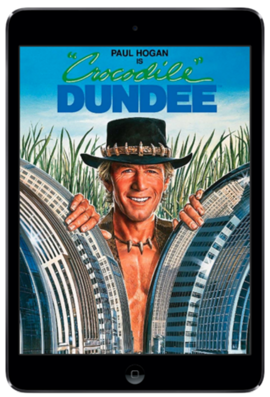


Video and image quality

Approximate Computing

quality bounds
into
approximate programs

Not all applications require perfect accuracy



Video and image quality

Sensors and simulation

Machine learning

Approximate Computing

quality bounds
into
approximate programs

Not all applications require perfect accuracy

Precise Implementation

Approximate Computing

quality bounds
into
approximate programs

Not all applications require perfect accuracy

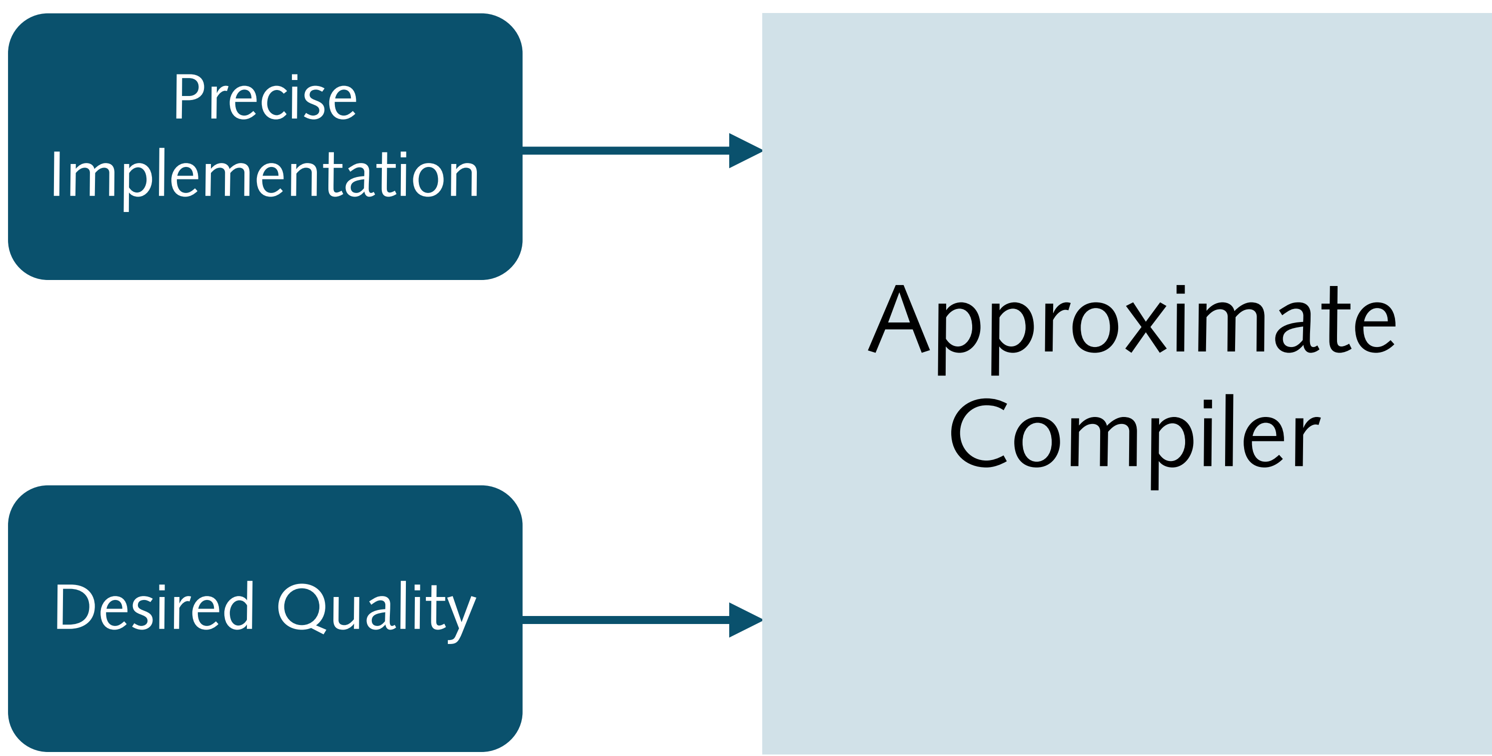
Precise Implementation

Desired Quality

Approximate Computing

quality bounds
into
approximate programs

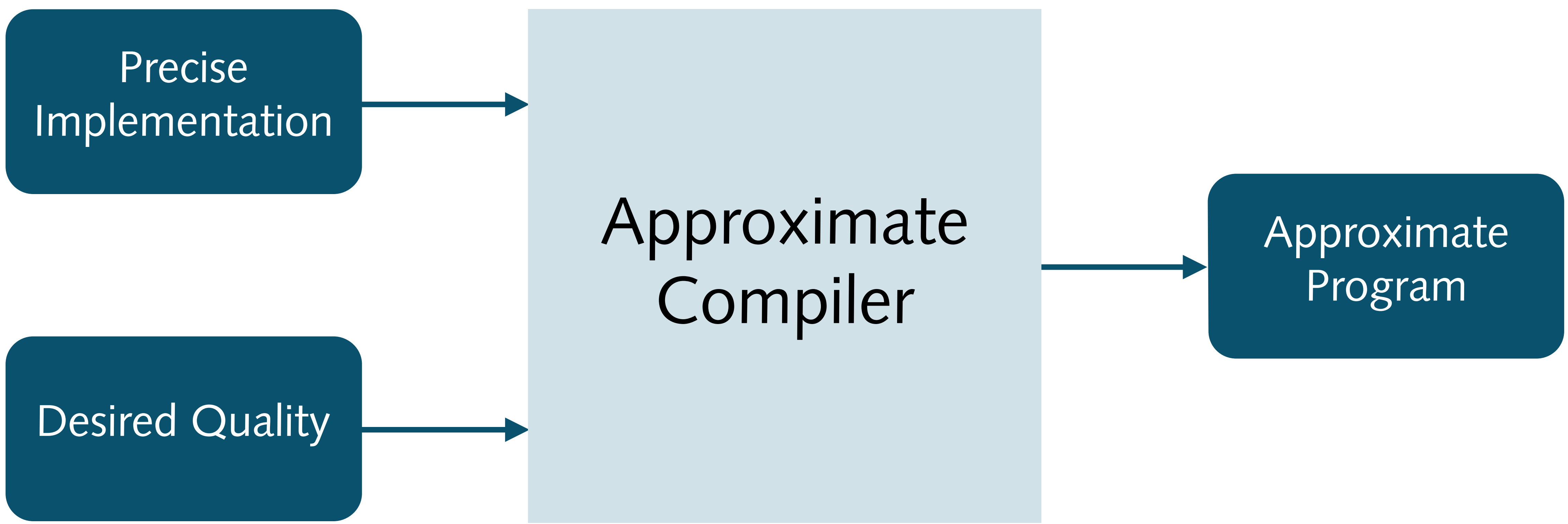
Not all applications require perfect accuracy



Approximate Computing

quality bounds
into
approximate programs

Not all applications require perfect accuracy



Approximate Computing

quality bounds
into
approximate programs

Not all applications require perfect accuracy

† Bornholt, Torlak, Ceze, Grossman. *Approximate Program Synthesis*. At WAX'15, collocated with PLDI'15.

Hardware Synthesis

programs
into
hardware designs

Synthesizing circuits from high-level languages

Hardware Synthesis

programs
into
hardware designs

Synthesizing circuits from high-level languages

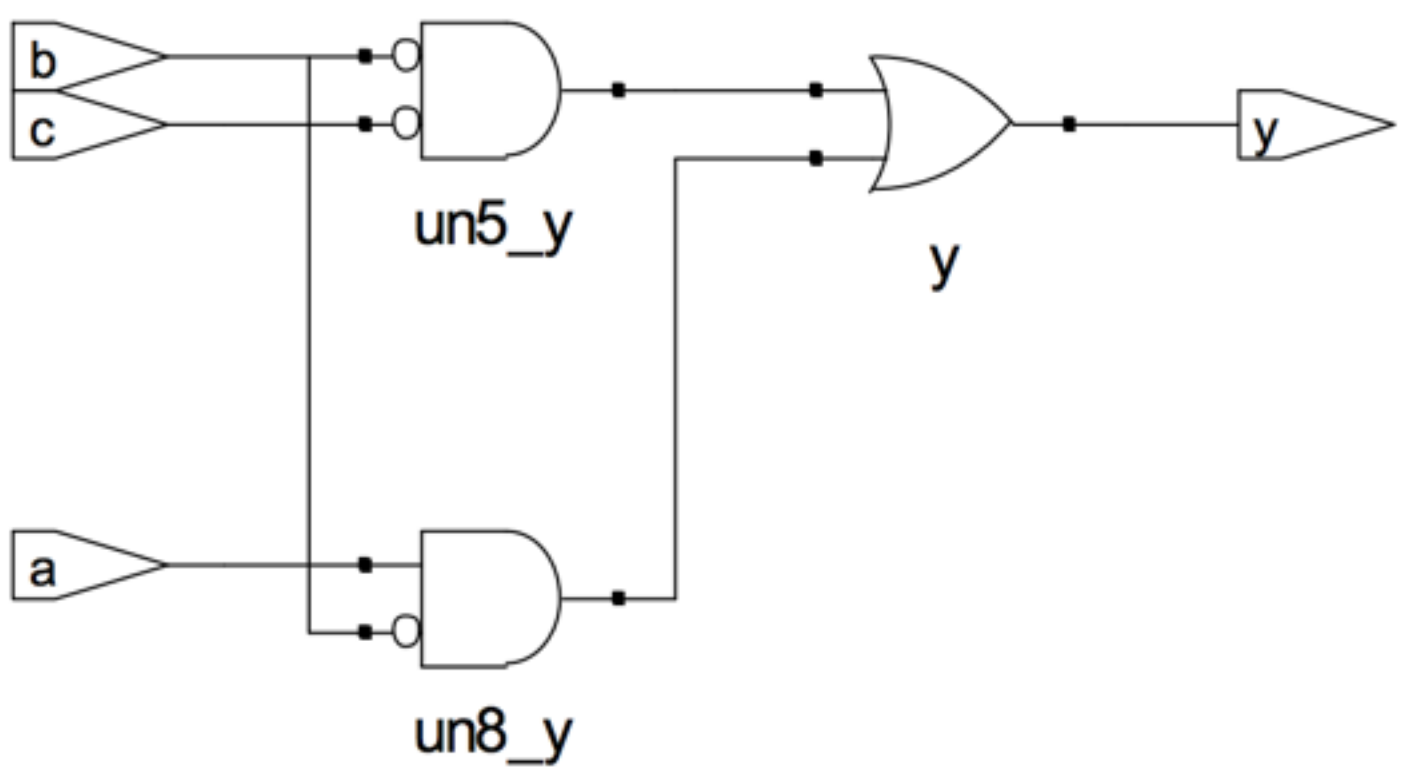
```
module example(input a, b, c, output y);
 assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule
```

Hardware Synthesis

programs
into
hardware designs

Synthesizing circuits from high-level languages

```
module example(input a, b, c, output y);
 assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule
```

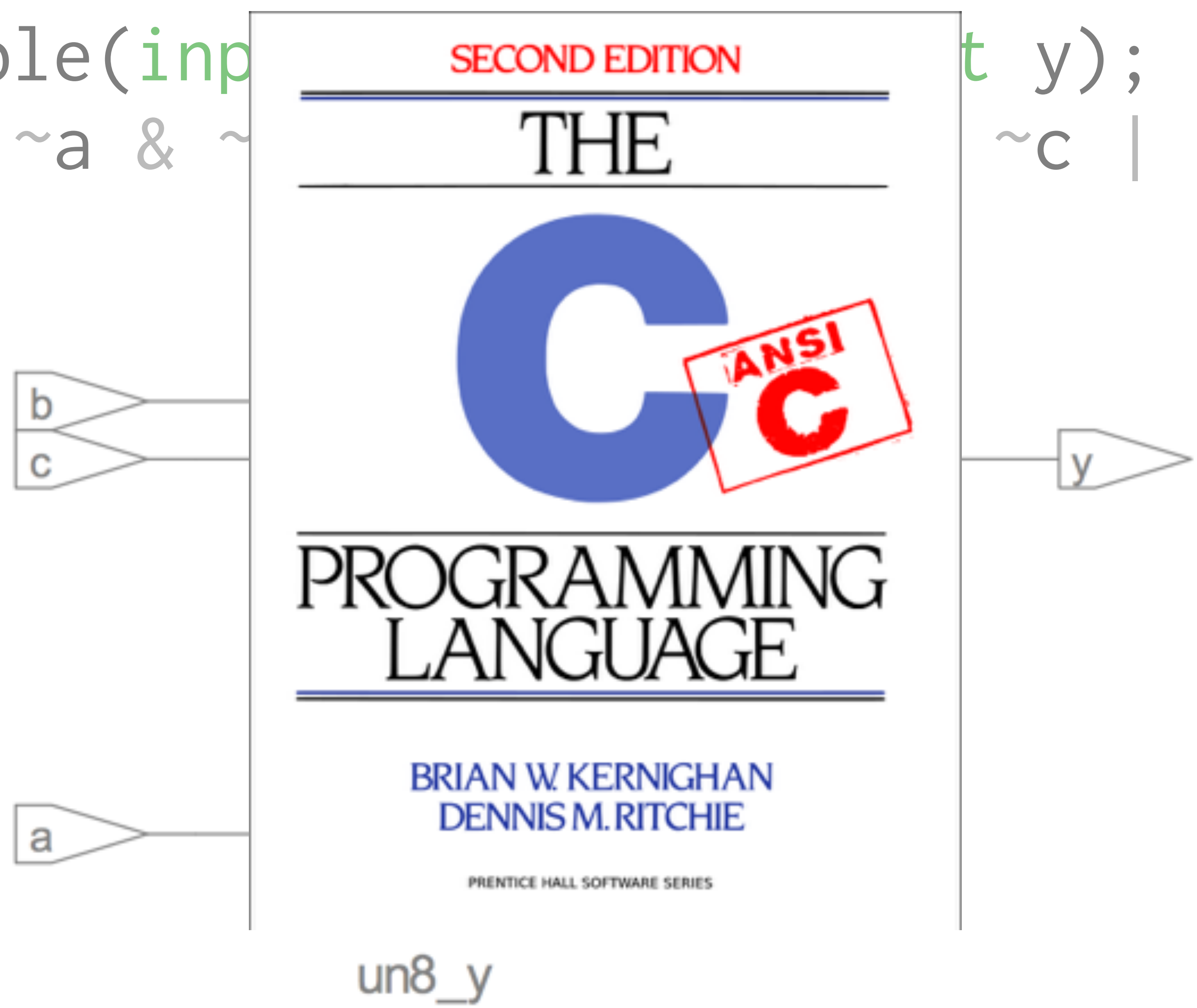


Hardware Synthesis

programs
into
hardware designs

Synthesizing circuits from high-level languages

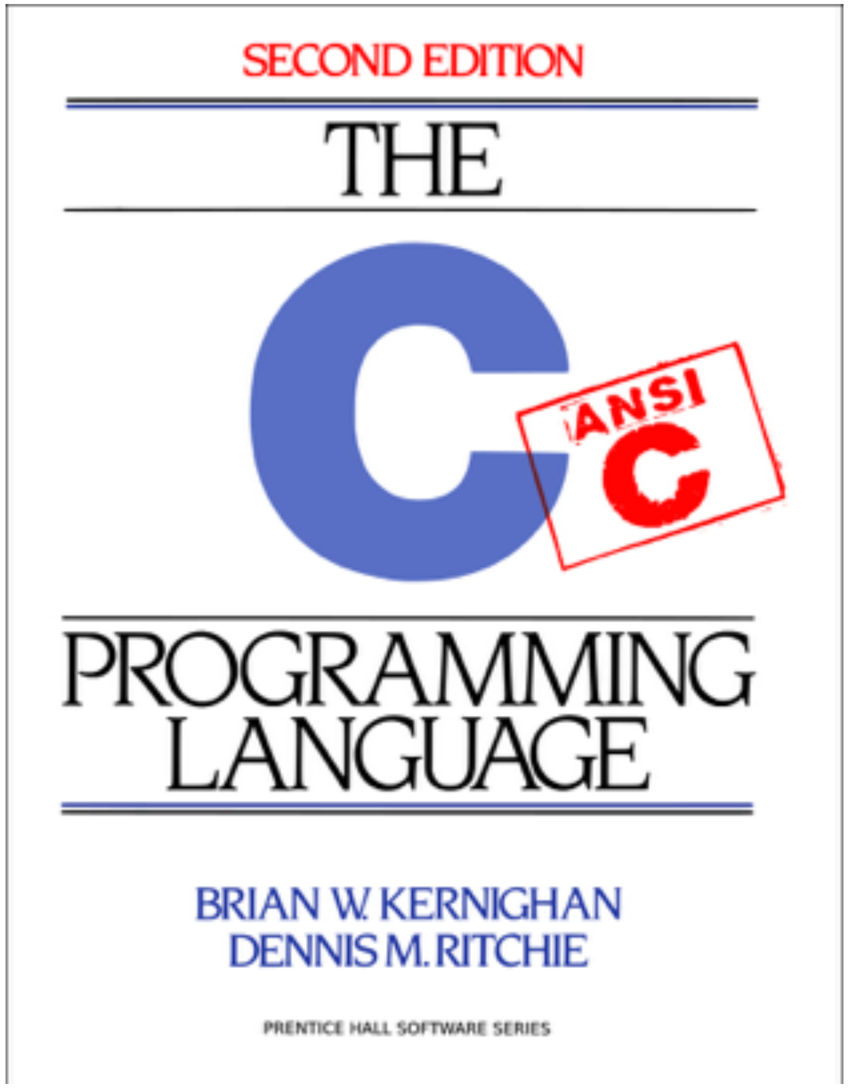
```
module example(inp          t y);
 assign y = ~a & ~          ~c | a & ~b & c;
endmodule
```



Hardware Synthesis

programs
into
hardware designs

Synthesizing circuits from high-level languages

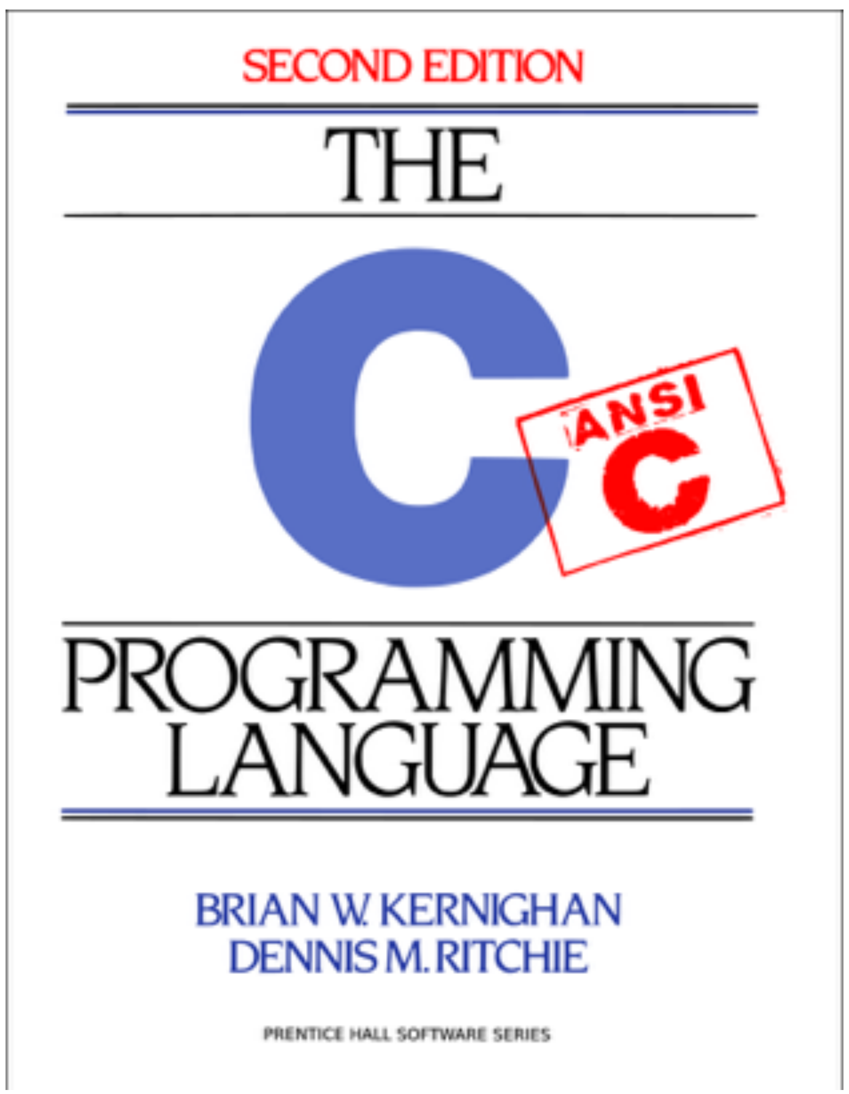


High-Level Synthesis (HLS)

Hardware Synthesis

programs
into
hardware designs

Synthesizing circuits from high-level languages



Place and route

Timing closure

High-Level Synthesis (HLS)

Crosstalk and feedback

Quantum!

Hardware Synthesis

programs
into
hardware designs

Synthesizing circuits from high-level languages

Mark Wyse
UW grad student and
HLS extraordinaire

Hardware Synthesis

programs
into
hardware designs

Synthesizing circuits from high-level languages

Mark Wyse
UW grad student and
HLS extraordinaire

```
float dist(float a[3], float b[3]) {
    float r = 0;
    r += (a[0] - b[0]) * (a[0] - b[0]);
    r += (a[1] - b[1]) * (a[1] - b[1]);
    r += (a[2] - b[2]) * (a[2] - b[2]);
    return sqrt(r);
}
```

Black Box Systems

observed behaviors
into
specifications

Defining system behavior by observation

Black Box Systems

observed behaviors
into
specifications

Defining system behavior by observation

Black Box Systems

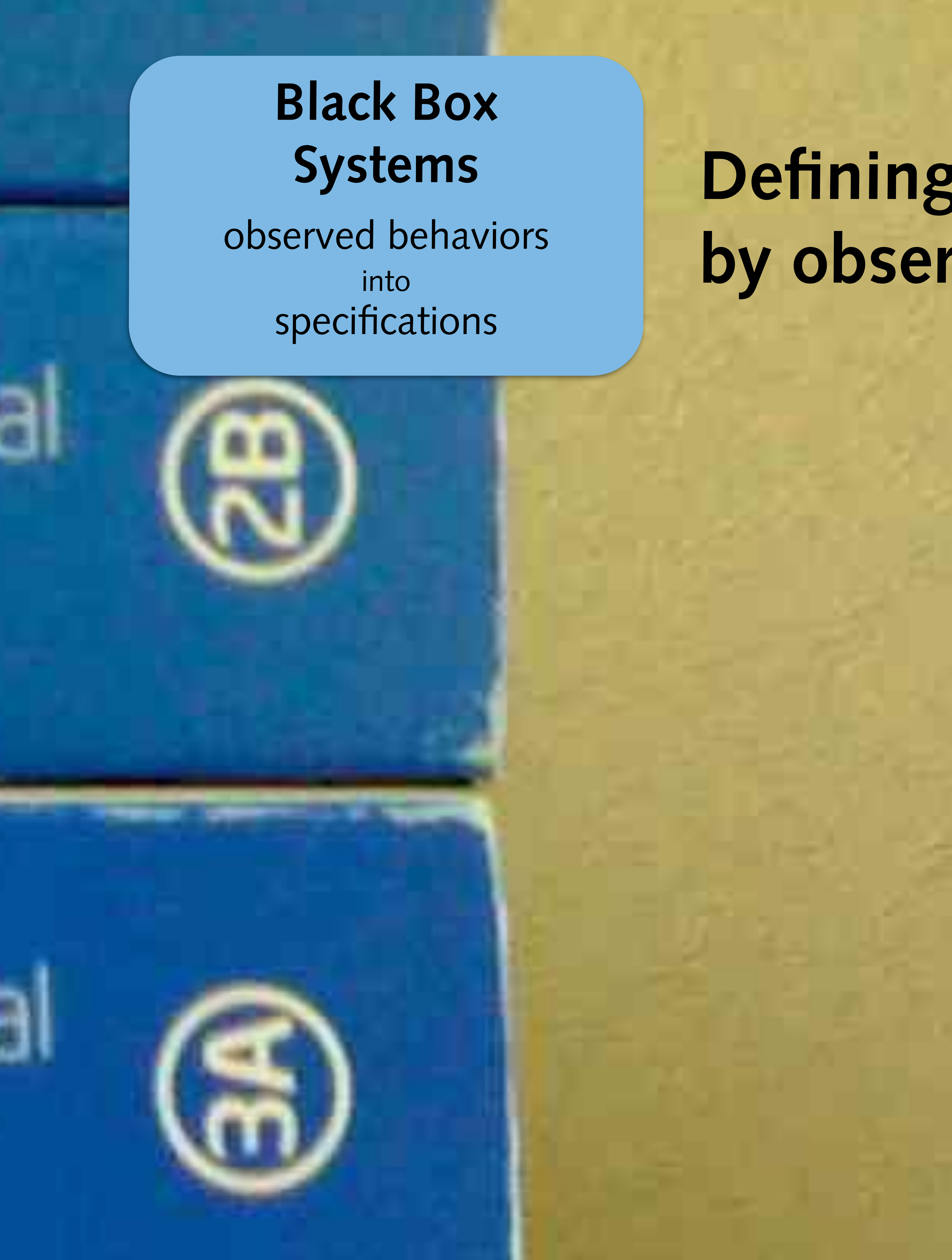
observed behaviors
into
specifications

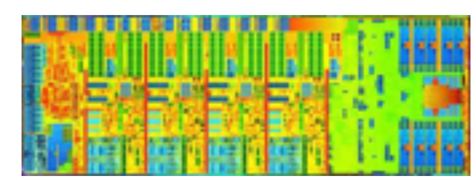
Defining system behavior by observation

Black Box Systems

observed behaviors
into
specifications

Defining system behavior by observation

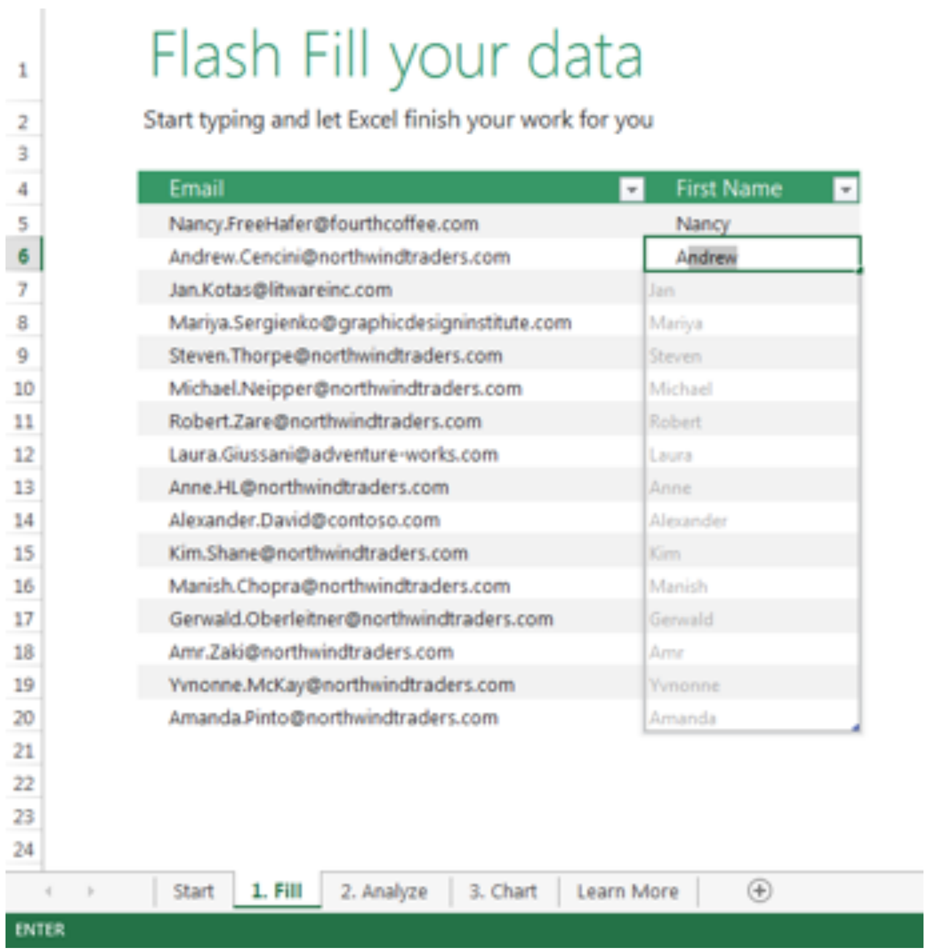




Black Box Systems

observed behaviors
into
specifications

Defining system behavior by observation

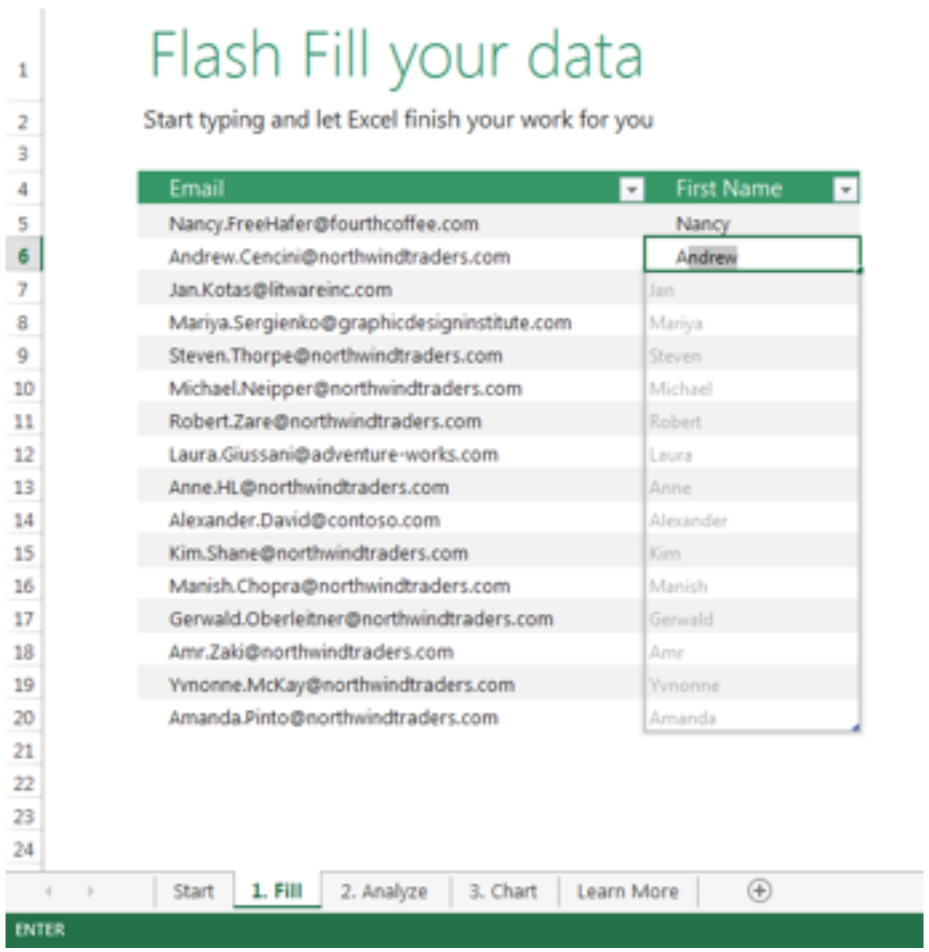


Programming
by example

Black Box Systems

observed behaviors
into
specifications

Defining system behavior by observation



Programming by example

Automated Synthesis of Symbolic Instruction Encodings from I/O Samples

Patrice Godefroid
Microsoft Research
pg@microsoft.com

Ankur Taly *
Stanford University
ataly@stanford.edu

Abstract

1. Introduction

Synthesizing x86 instruction specs

Godefroid and Taly. *Automated Synthesis of Symbolic Instruction Encodings from I/O Samples.* PLDI'12.

Approximate Computing

quality bounds
into
approximate programs

Hardware Synthesis

programs
into
hardware designs

Black Box Systems

observed behaviors
into
specifications

Machine Learning and Synthesis

Learning programs from examples

Black Box Systems

observed behaviors

into

specifications

$$\exists P. \bigwedge_{x_i \in X} \varphi(x_i, P(x_i))$$

Learning programs from examples

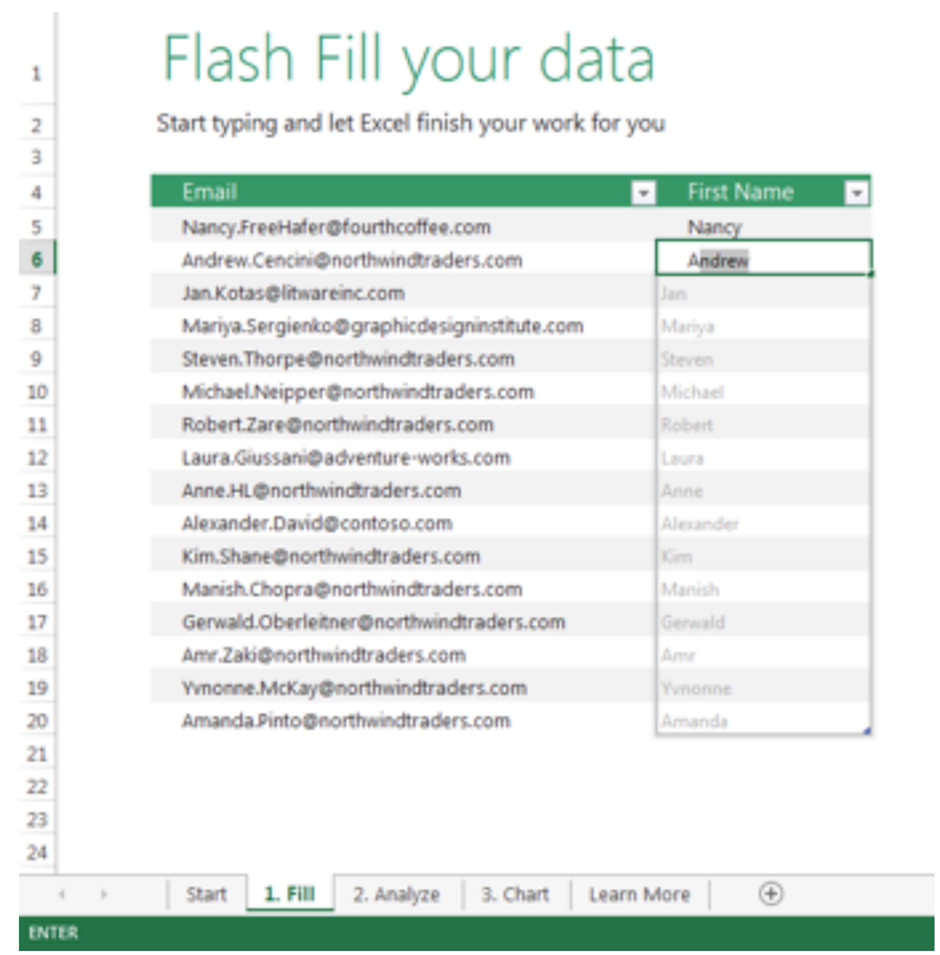
Black Box Systems

observed behaviors

into

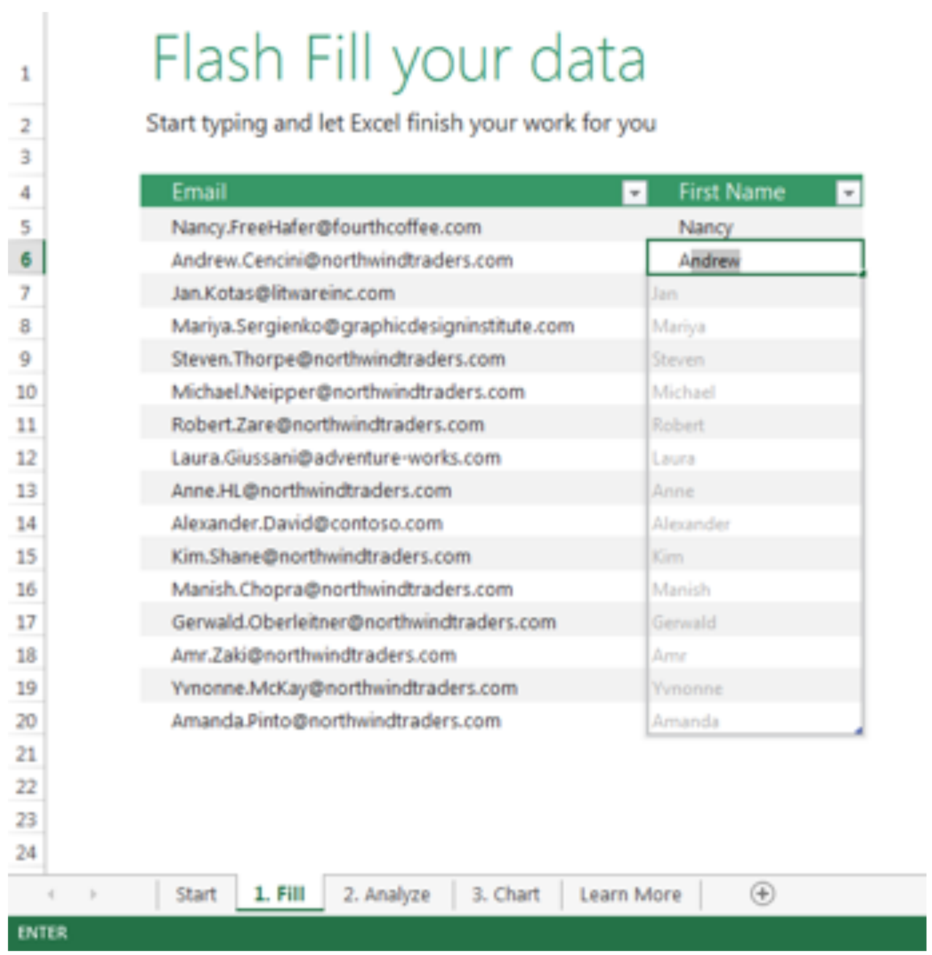
specifications

$$\exists P. \bigwedge_{x_i \in X} \varphi(x_i, P(x_i))$$

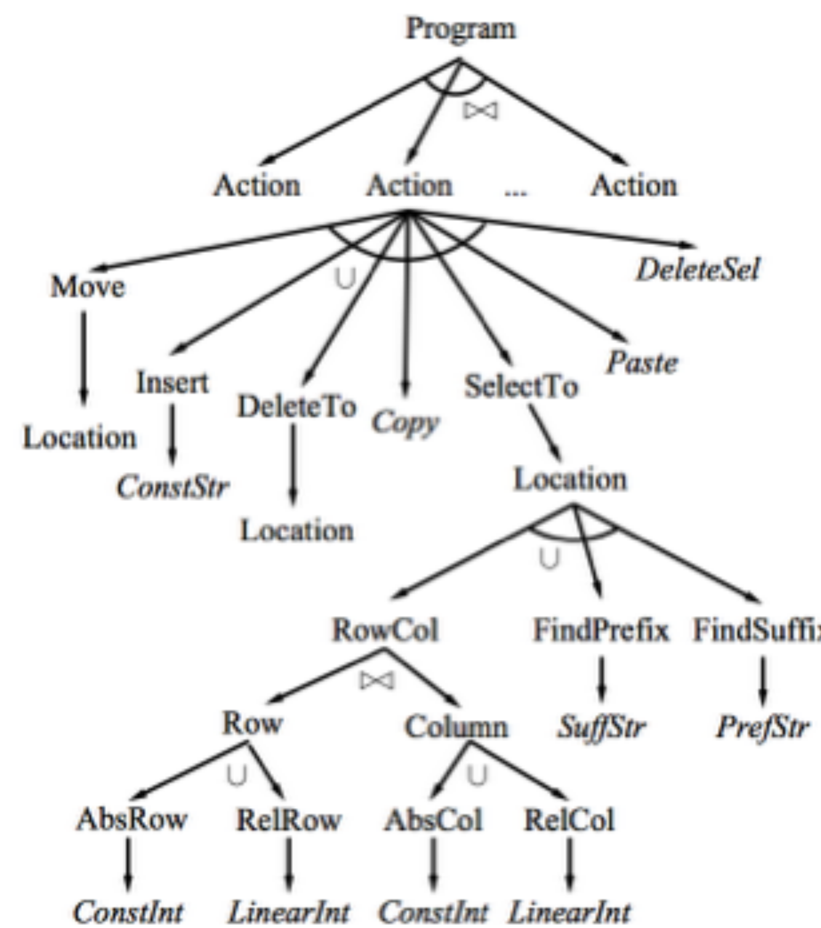


FlashFill

Learning programs from examples

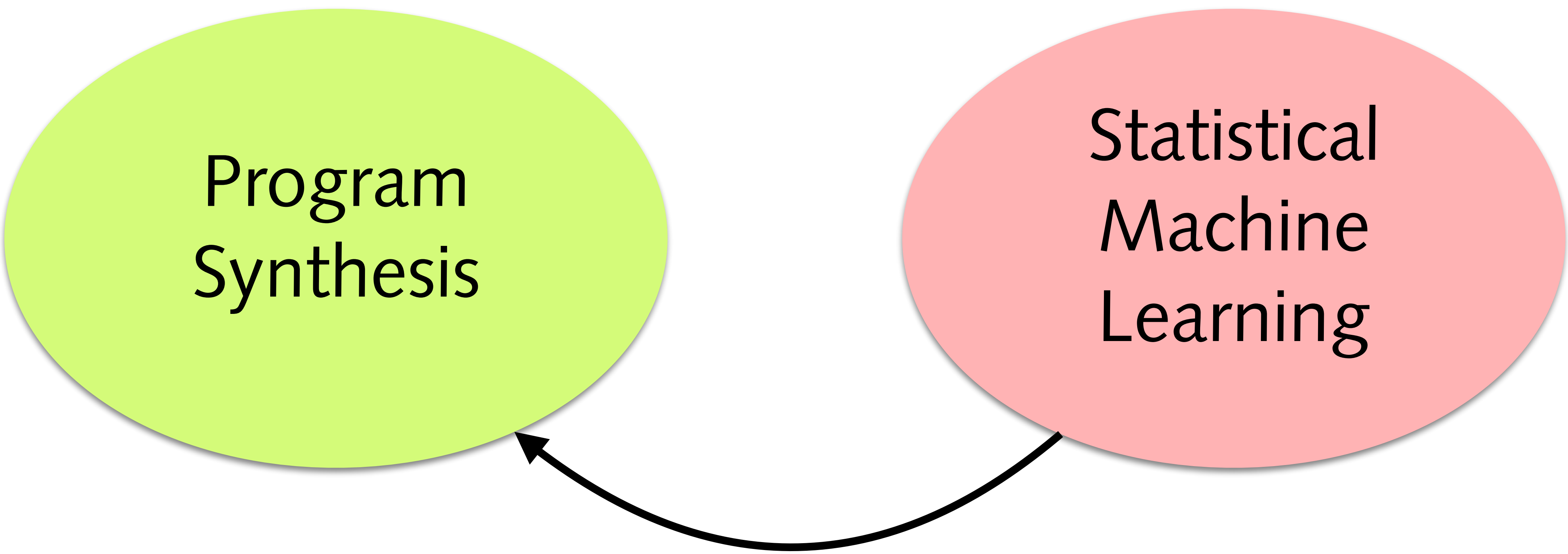


FlashFill

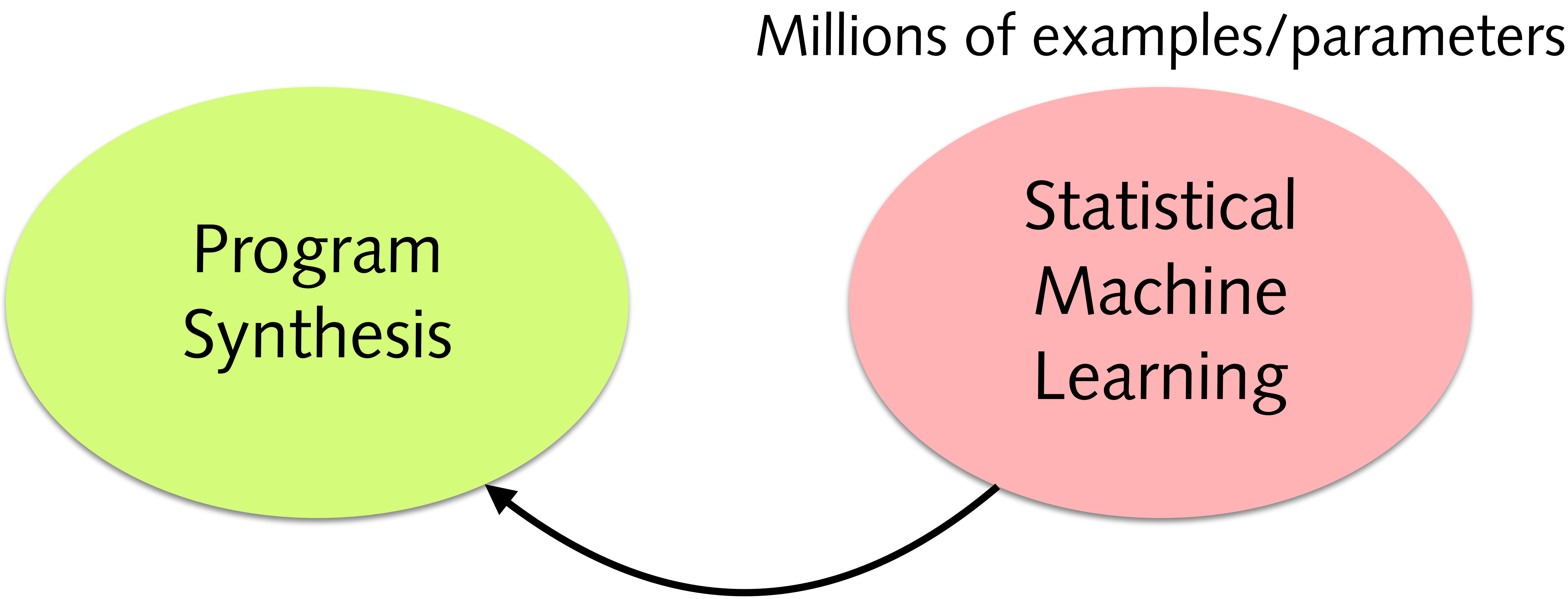


Version Space Algebra

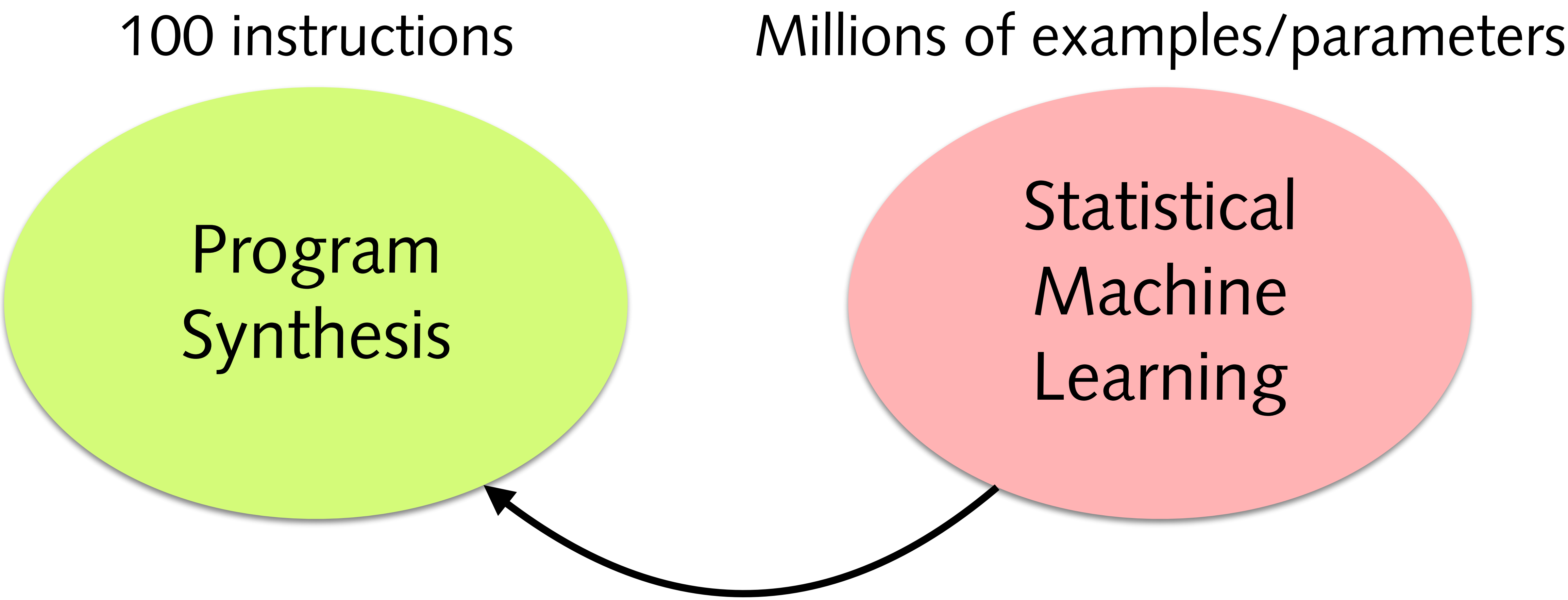
Machine learning and synthesis



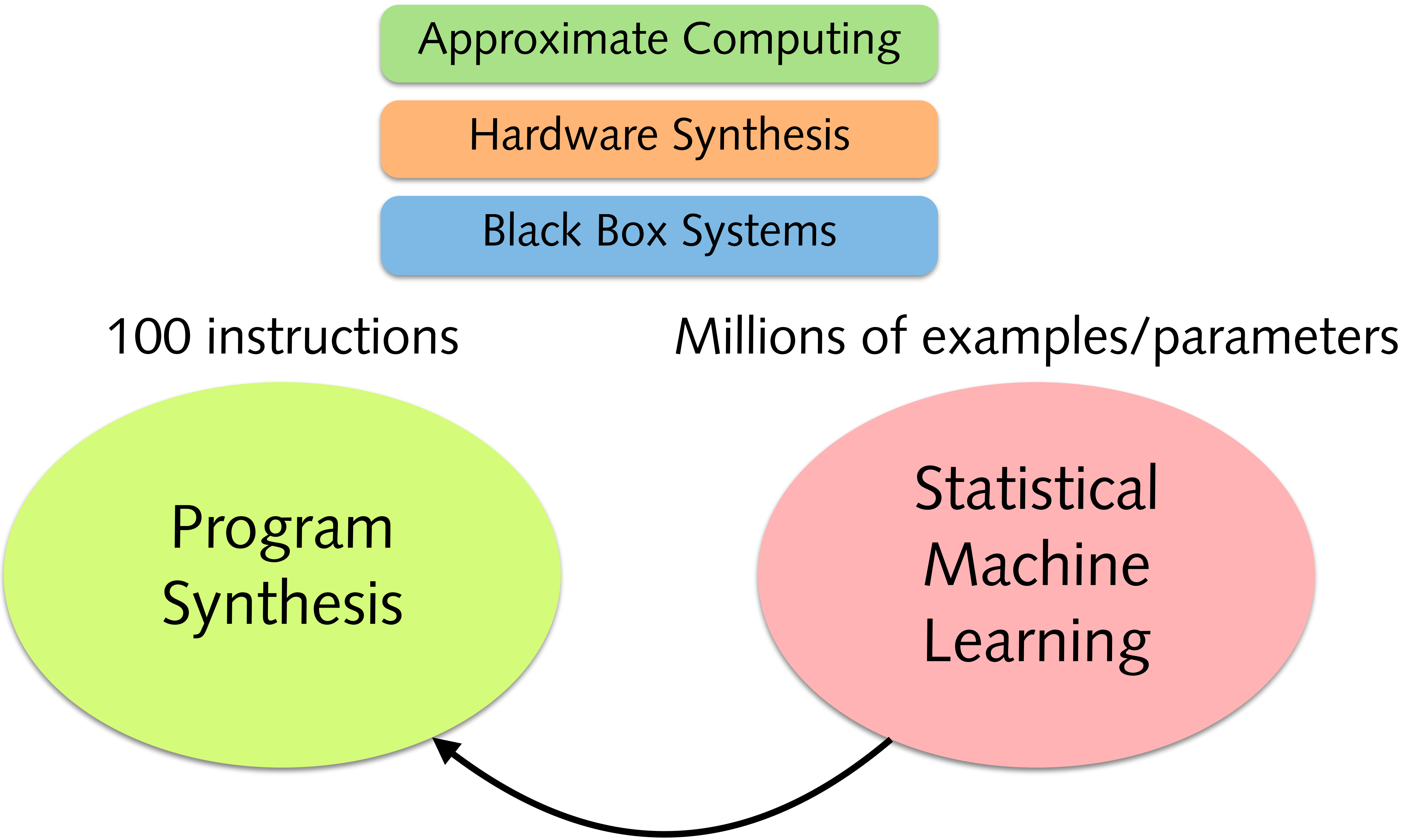
Machine learning and synthesis



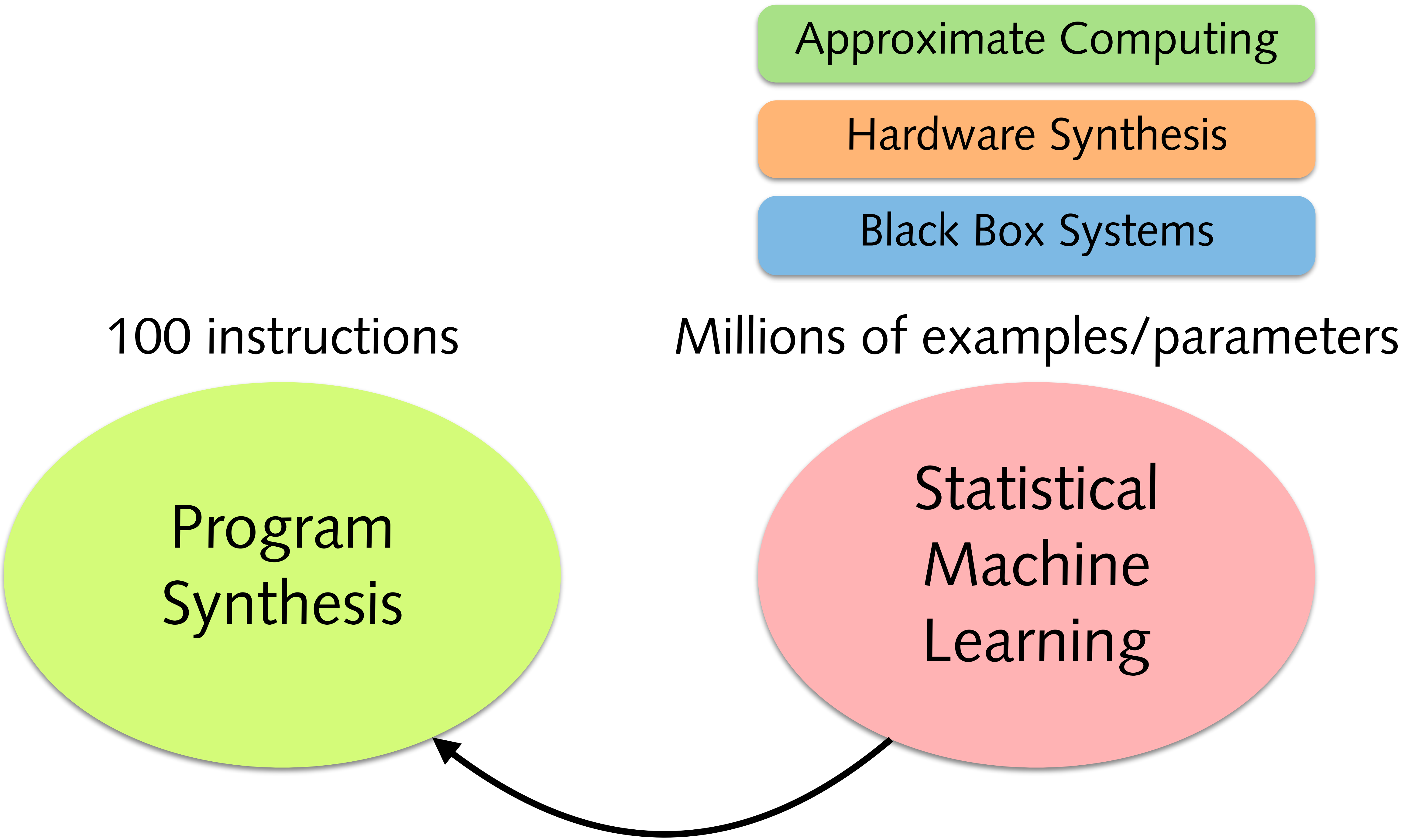
Machine learning and synthesis



Machine learning and synthesis

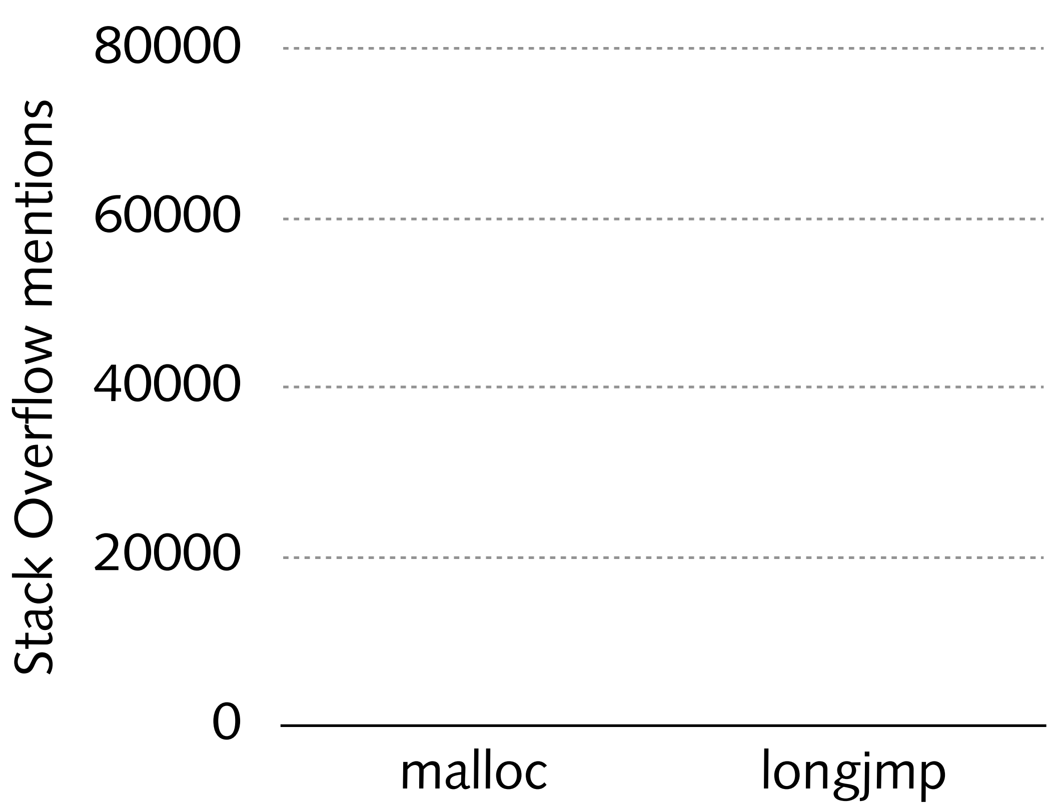


Machine learning and synthesis

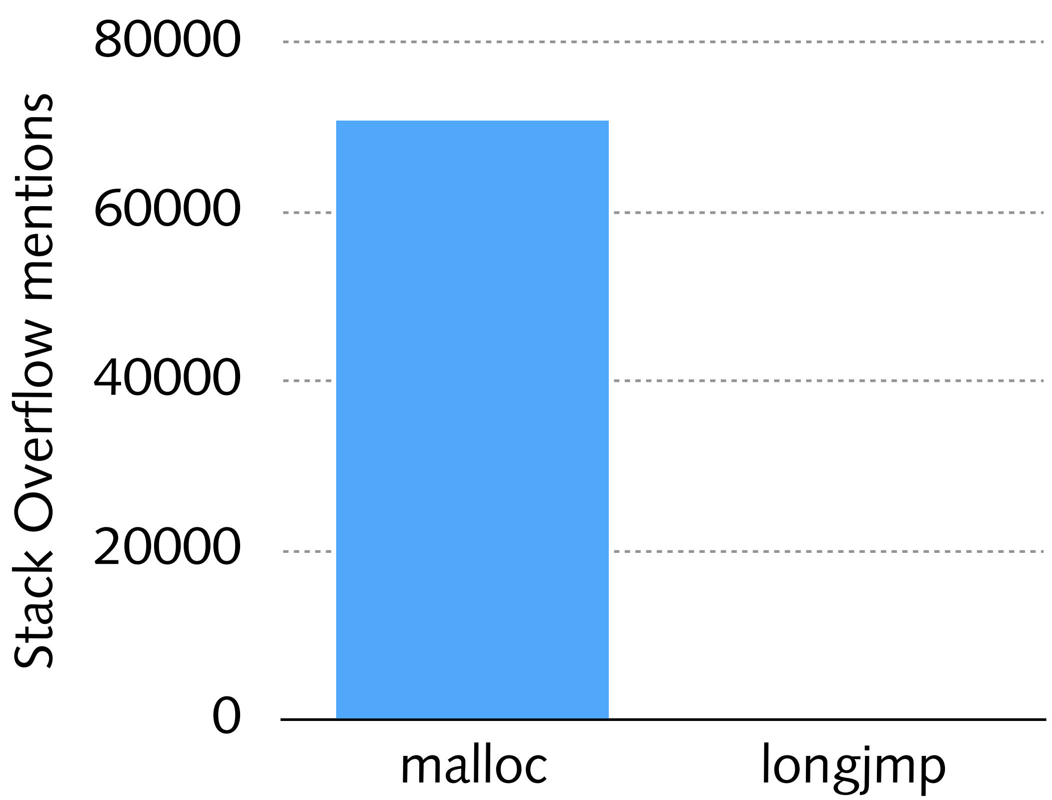


Programs are not uniformly distributed.

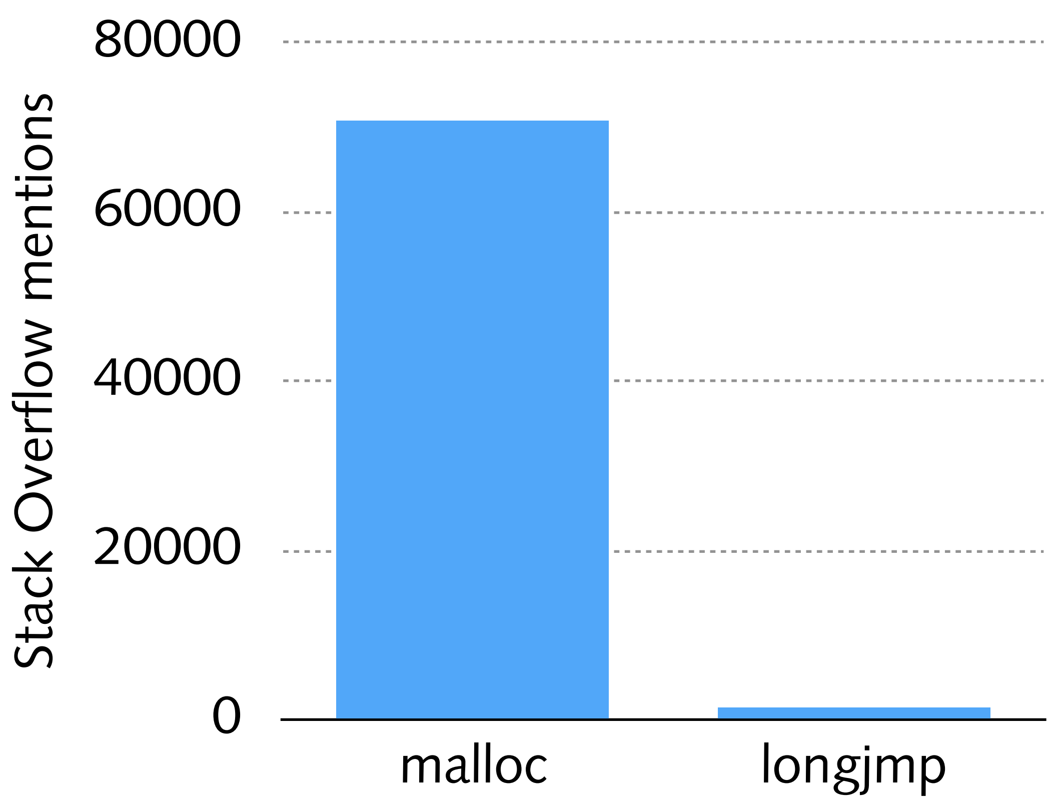
Programs are not uniformly distributed.



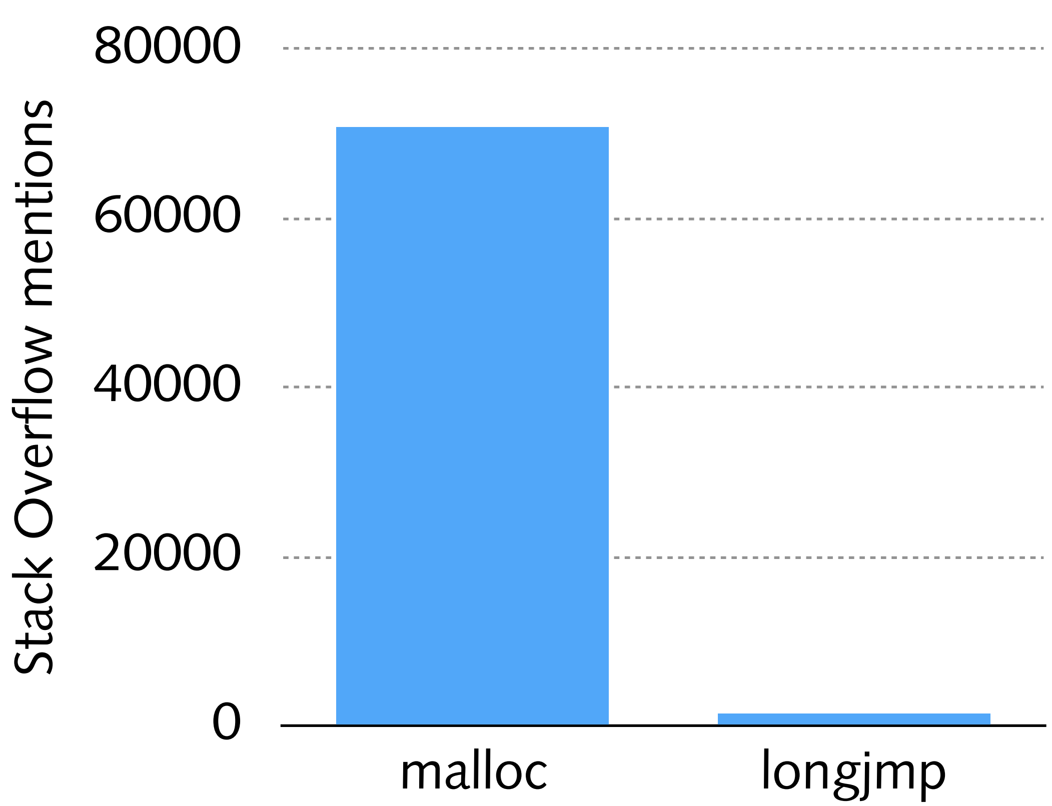
Programs are not uniformly distributed.

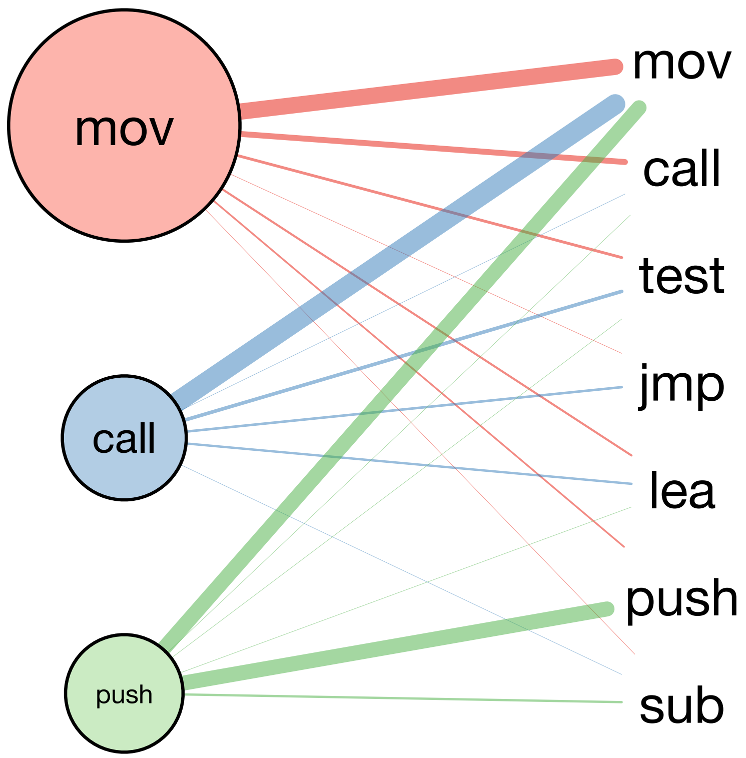


Programs are not uniformly distributed.



Programs are not uniformly distributed.





Component-Based Synthesis

Synthesis of Loop-Free Programs

Gulwani, Jha, Tiwari, and Venkatesan. *Synthesis of loop-free programs.* PLDI'11.

Synthesis of Loop-Free Programs

$$f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$

Gulwani, Jha, Tiwari, and Venkatesan. *Synthesis of loop-free programs.* PLDI'11.

Synthesis of Loop-Free Programs

$$f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$

and	xor
add	>> 1

Gulwani, Jha, Tiwari, and Venkatesan. *Synthesis of loop-free programs.* PLDI'11.

Synthesis of Loop-Free Programs

$$f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$

and	xor
add	>> 1

x

y

Gulwani, Jha, Tiwari, and Venkatesan. *Synthesis of loop-free programs.* PLDI'11.

Synthesis of Loop-Free Programs

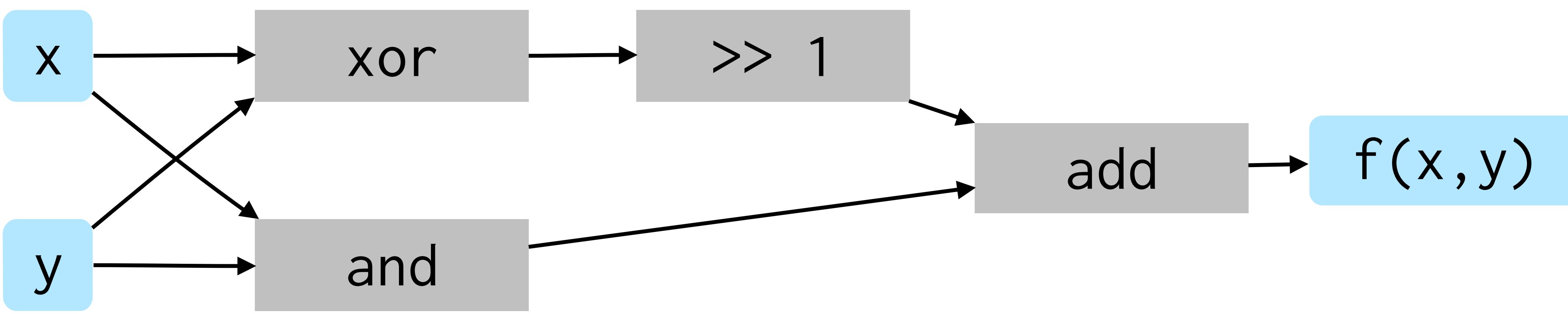
$$f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$

and	xor
add	>> 1

Gulwani, Jha, Tiwari, and Venkatesan. *Synthesis of loop-free programs.* PLDI'11.

Synthesis of Loop-Free Programs

$$f(x, y) = \left\lfloor \frac{x+y}{2} \right\rfloor$$



Gulwani, Jha, Tiwari, and Venkatesan. *Synthesis of loop-free programs.* PLDI'11.

Synthesis of Loop-Free Programs

$$f(x,y) = \left\lfloor \frac{x+y}{2} \right\rfloor$$

and	xor
add	>> 1

Gulwani, Jha, Tiwari, and Venkatesan. *Synthesis of loop-free programs.* PLDI'11.

Synthesis of Loop-Free Programs

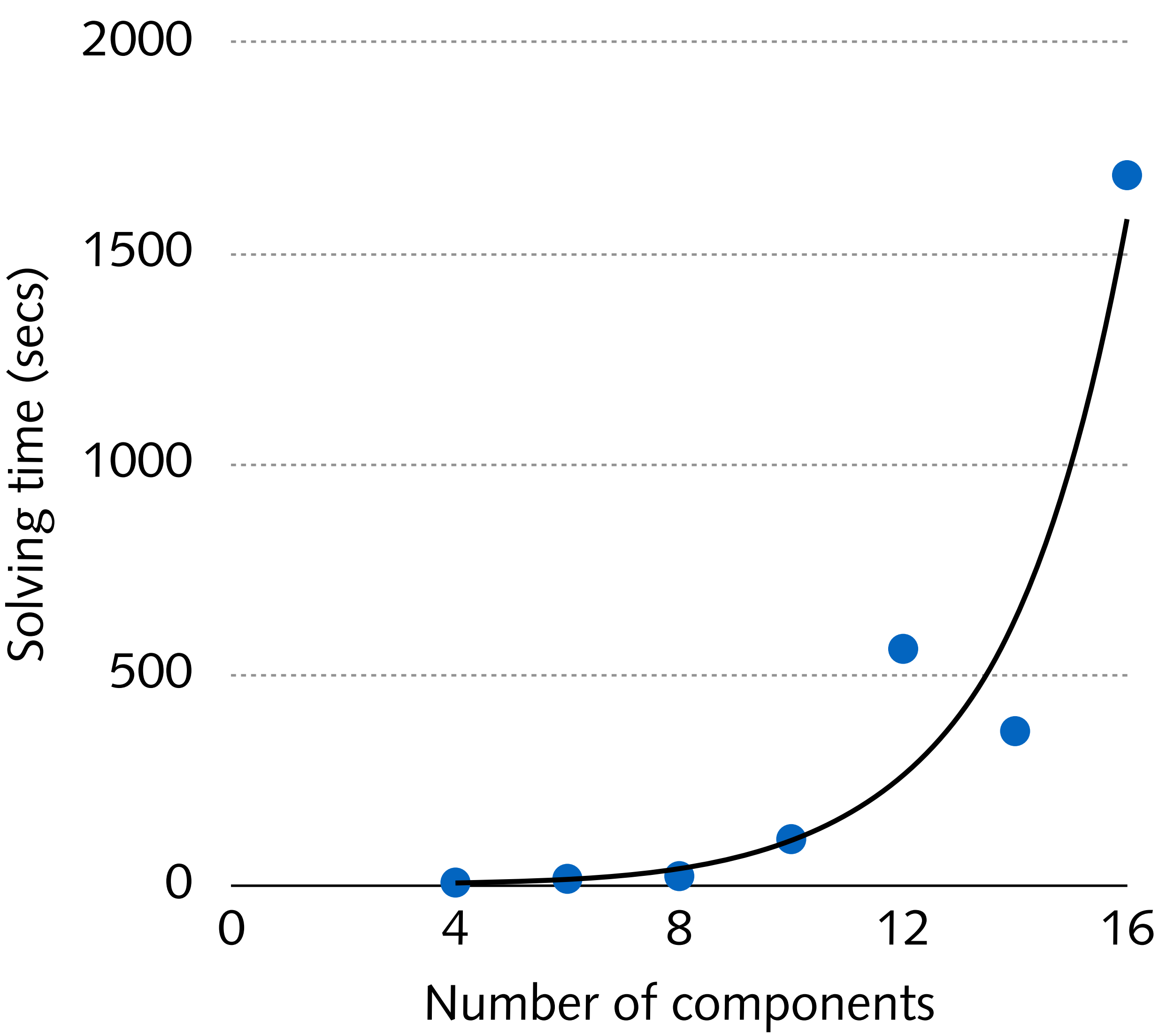
$$f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$

`and`	`xor`
`add`	`>> 1`
`sub`	`mul`
`div`	`udiv`
`rem`	`urem`
`or`	`nand`
`not`	`neg`
`<< 1`	`>>> 1`
`<< 2`	`eq`
`le`	`ge`
`lt`	`gt`

Gulwani, Jha, Tiwari, and Venkatesan. *Synthesis of loop-free programs.* PLDI'11.

Synthesis of Loop-Free Programs

$$f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$



and	xor
add	>> 1
sub	mul
div	udiv
rem	urem
or	nand
not	neg
<< 1	>>> 1
<< 2	eq
le	ge
lt	gt

Gulwani, Jha, Tiwari, and Venkatesan. *Synthesis of loop-free programs.* PLDI'11.

Higher-level components

$$f(L) = \min\{l_i \mid l_i \in L\}$$

and	xor
add	>> 1
sub	mul
div	udiv
rem	urem
or	nand
not	neg
<< 1	>>> 1
<< 2	eq
le	ge
lt	gt

Higher-level components

$$f(L) = \min\{l_i \mid l_i \in L\}$$

Higher-level components

$$f(L) = \min\{l_i \mid l_i \in L\}$$

`quicksort`

`0`

`[ ]`

Higher-level components

$$f(L) = \min\{l_i \mid l_i \in L\}$$

```
f(L) = quicksort(L)[0]
```

`quicksort`

`0`

`[ ]`

Higher-level components

$$f(L) = \min\{l_i \mid l_i \in L\}$$

```
f(L) = quicksort(L)[0]
```

quicksort

0

[ ]

Mine existing code bases for common idioms

Higher-level components

$$f(L) = \min\{l_i \mid l_i \in L\}$$

```
f(L) = quicksort(L)[0]
```

`quicksort`

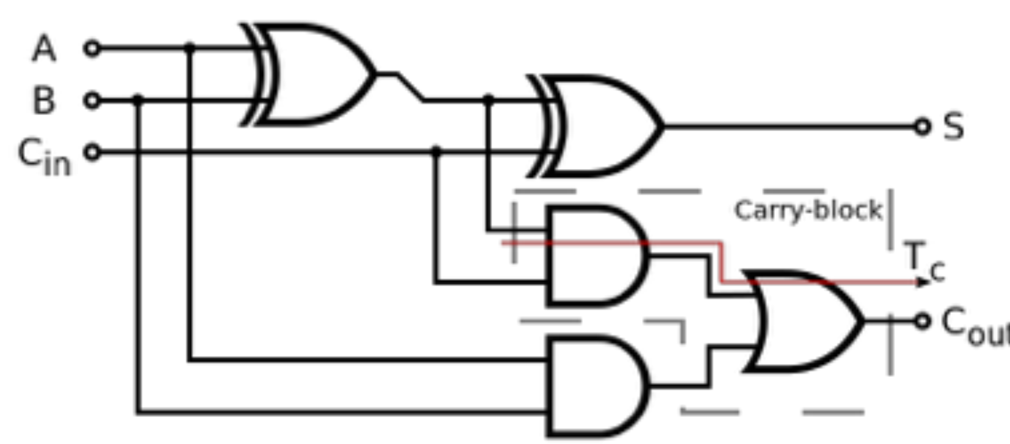
`0`

`[ ]`

Mine existing code bases for common idioms

Hardware Synthesis

programs
into
hardware designs



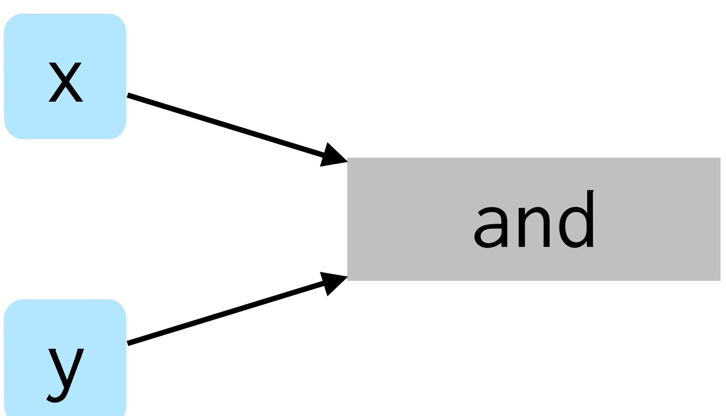
Producing candidate programs

$$f(x,y) = \left\lfloor \frac{x+y}{2} \right\rfloor$$

and	xor
add	>> 1

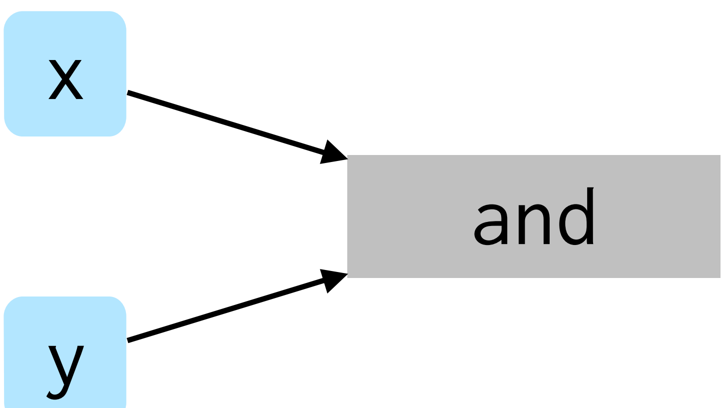
Producing candidate programs

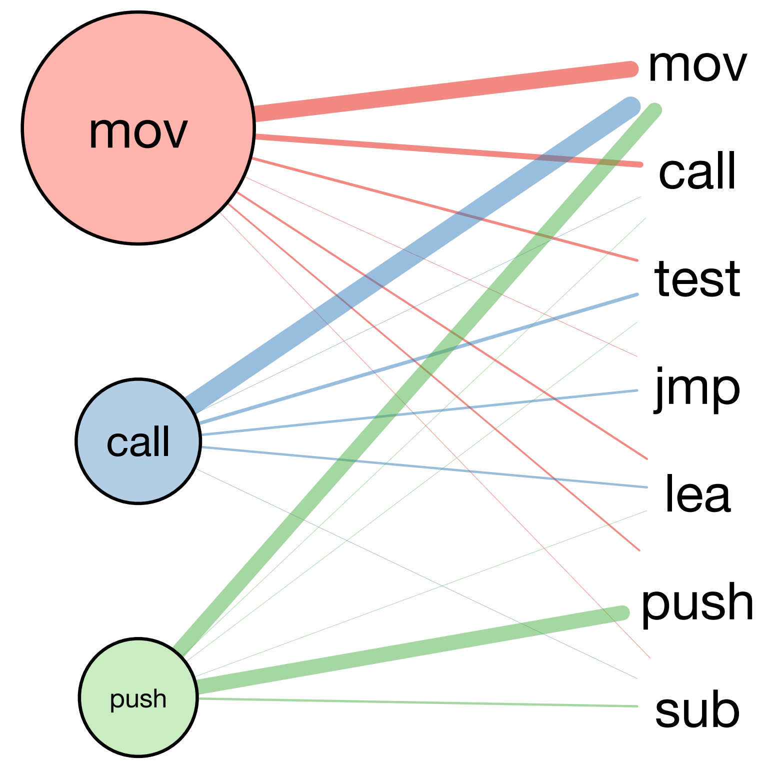
$$f(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$



Producing candidate programs

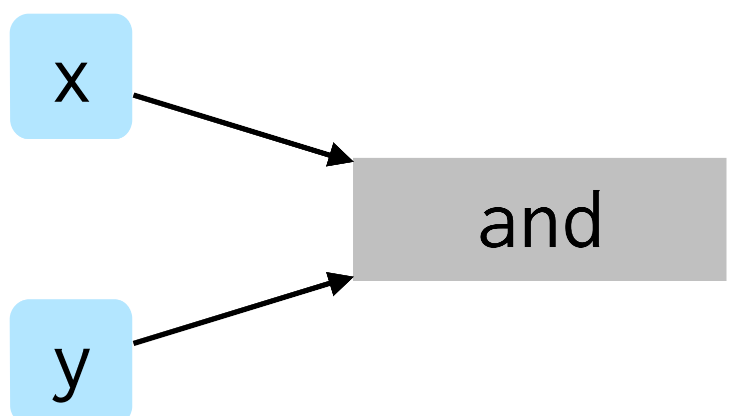
$$f(x,y) = \left\lfloor \frac{x+y}{2} \right\rfloor$$

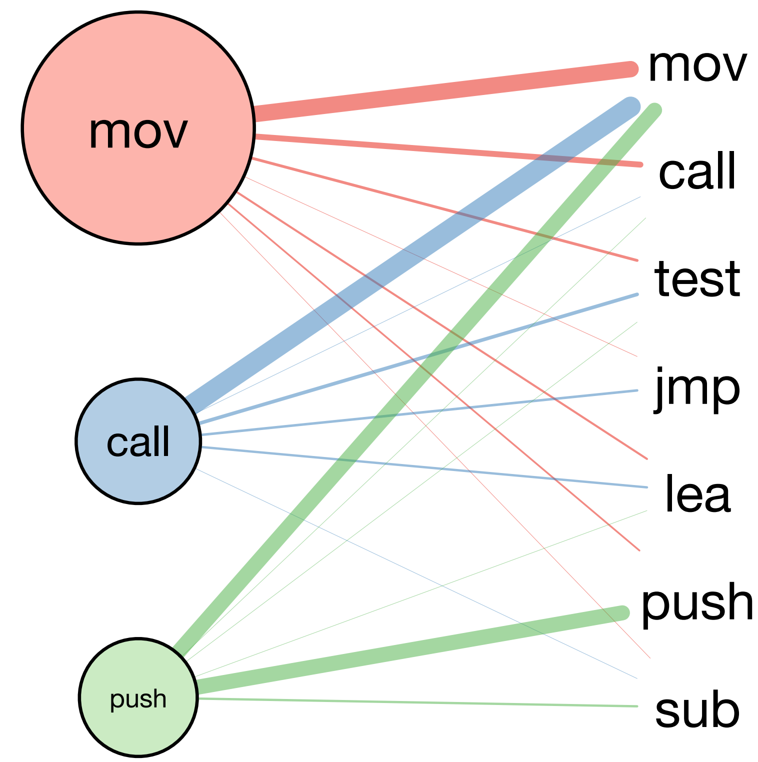




Producing candidate programs

$$f(x,y) = \left\lfloor \frac{x+y}{2} \right\rfloor$$

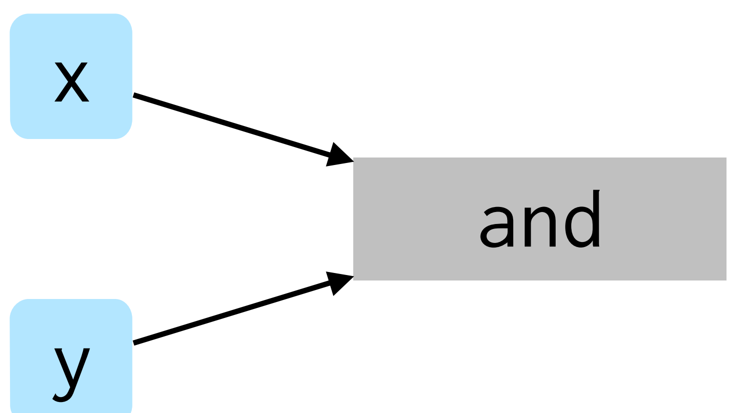


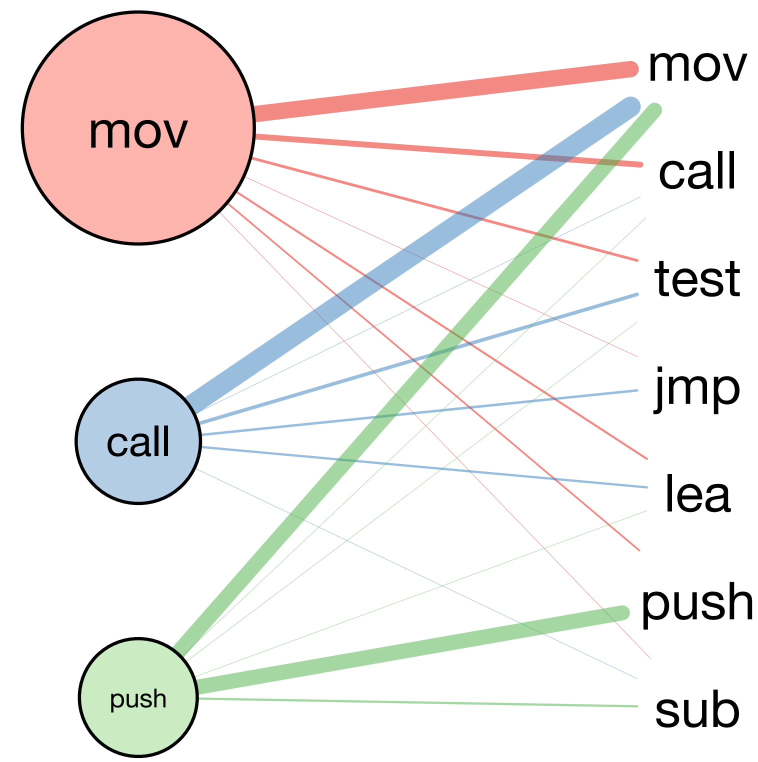


Learn search heuristics from existing code

Producing candidate programs

$$f(x, y) = \left\lfloor \frac{x+y}{2} \right\rfloor$$





Learn search heuristics from existing code

Programming by example

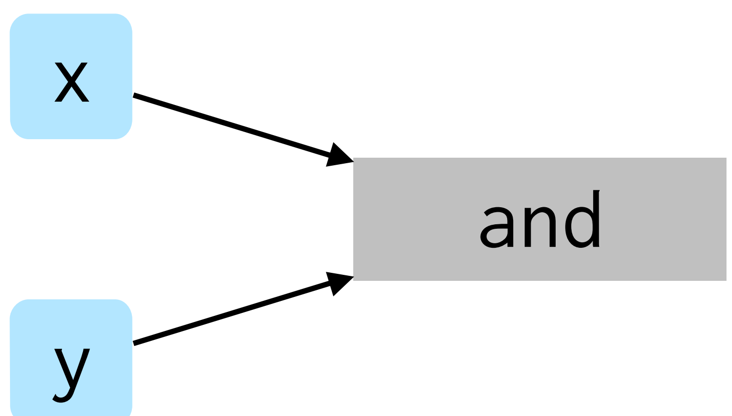
[FlashFill, POPL'11]

Autocomplete with synthesis

[CodeHint, ICSE'14]

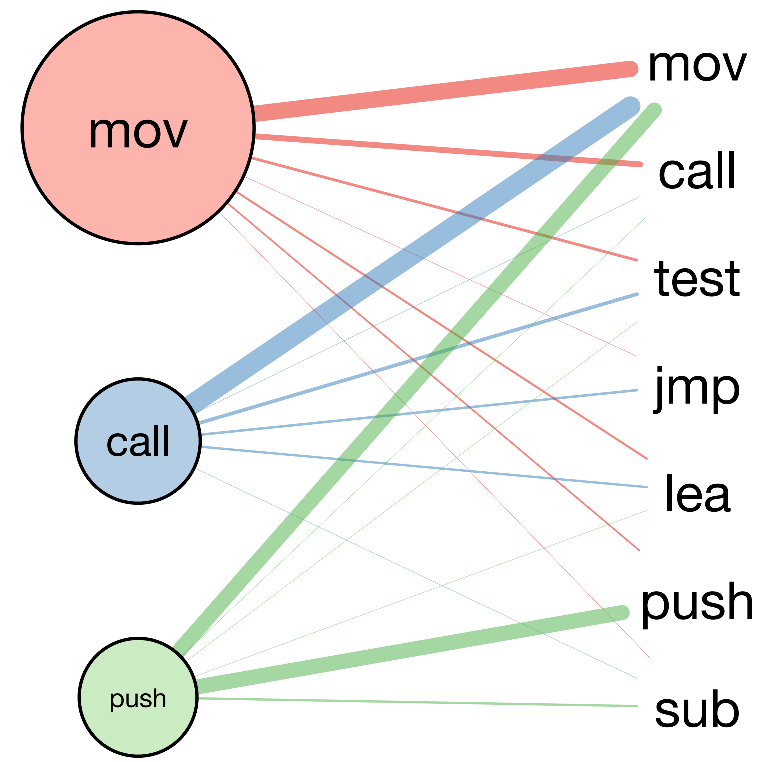
Producing candidate programs

$$f(x, y) = \left\lfloor \frac{x+y}{2} \right\rfloor$$



Mine existing code bases for common idioms

Learn search heuristics from existing code



Programming by example

[FlashFill, POPL'11]

Autocomplete with synthesis

[CodeHint, ICSE'14]

Producing candidate programs

Mine existing code bases for common idioms

Learn search heuristics from existing code

Black Box Systems

observed behaviors

into

specifications

Selecting among many candidate solutions

Sketch-Based Synthesis

Sketches make synthesis more tractable

Sketches make synthesis more tractable

```
bit[W] popCount(bit[W] x) {
    x = (x & 0x5555) + ((x >> 1) & 0x5555);
    x = (x & 0x3333) + ((x >> 2) & 0x3333);
    x = (x & 0x0077) + ((x >> 8) & 0x0077);
    x = (x & 0x000F) + ((x >> 4) & 0x000F);
    return x;
}
```

Sketches make synthesis more tractable

```
bit[W] popCount(bit[W] x) {
    x = (x & 0x5555) + ((x >> 1) & 0x5555);
    x = (x & 0x3333) + ((x >> 2) & 0x3333);
    x = (x & 0x0077) + ((x >> 8) & 0x0077);
    x = (x & 0x000F) + ((x >> 4) & 0x000F);
    return x;
}
```

```
bit[W] popSketched(bit[W] x) {
    loop (??) {
        x = (x & ??) + ((x >> ??) & ??);
    }
    return x;
}
```

Sketches make synthesis more tractable

```
bit[W] popSketched(bit[W] x) {
    loop (??) {
        x = (x & ??) + ((x >> ??) & ??);
    }
    return x;
}
```

Sketches make synthesis more tractable

```
bit[W] popSketched(bit[W] x) {
    loop (??) {
        x = (x & ??) + ((x >> ??) & ??);
    }
    return x;
}
```

```
int[] map(int[] xs) {
    int[] ys = {};
    for (int i=0; i<xs.length; i++) {
        ys.append(??(xs[i]));
    }
    return ys;
}
```

```
int[] foldl(int[] xs) {
    int acc = ??;
    for (int i=0; i<xs.length; i++) {
        acc = ??(acc, xs[i]);
    }
    return ys;
}
```

```
int[] filter(int[] xs) {
    int[] ys = {};
    for (int i=0; i<xs.length; i++) {
        if (??(xs[i])) ys.append(xs[i]);
    }
}
```

Sketches make synthesis more tractable

Identify common sketches from existing code

Sketches make synthesis more tractable

Mine existing code bases for common idioms

Identify common sketches from existing code

Sketches make synthesis more tractable

Mine existing code bases
for common idioms

Identify common sketches from
existing code

Approximate Computing

quality bounds
into
approximate programs

Sketches make synthesis more tractable

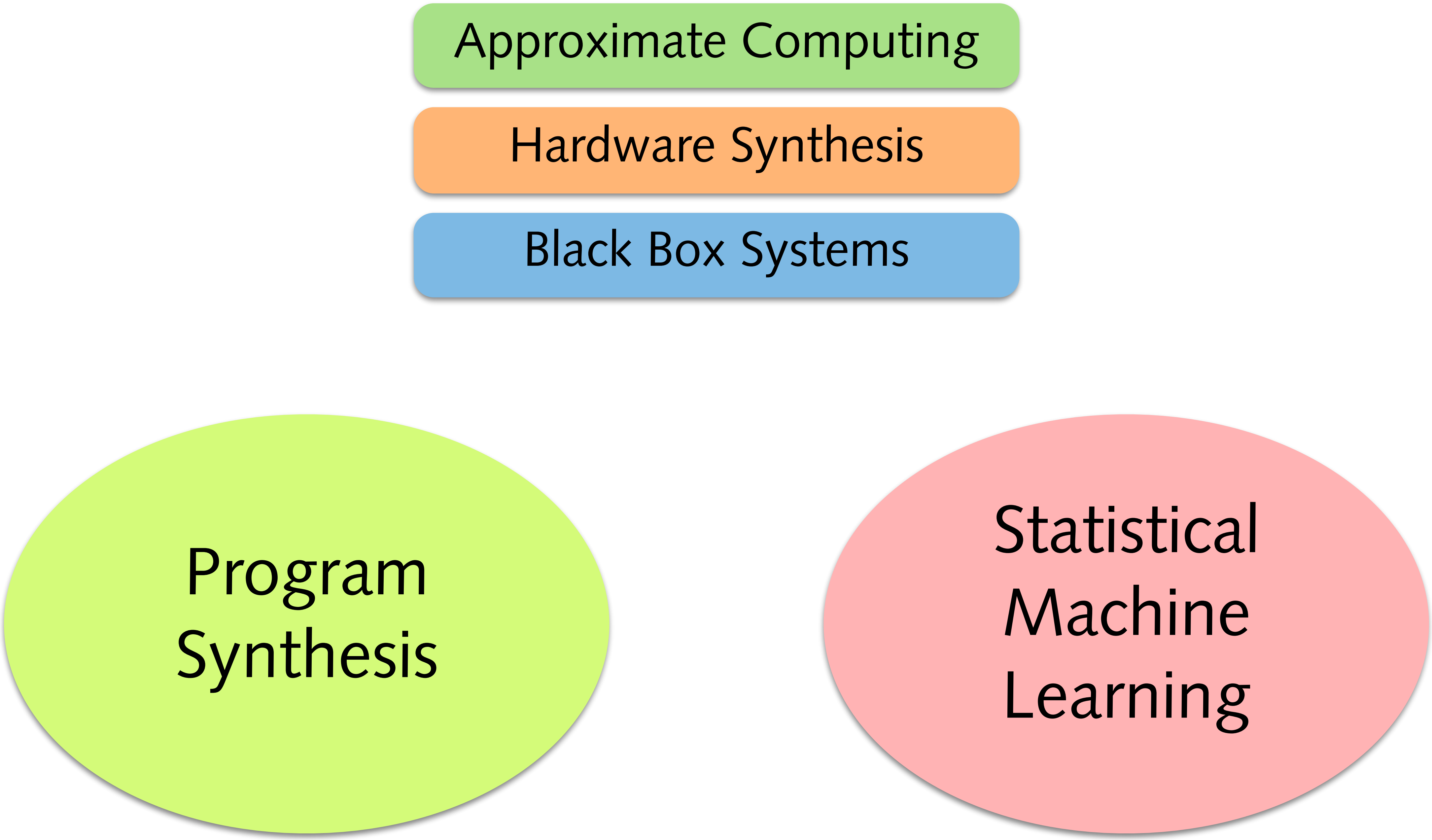
Mine existing code bases
for common idioms

Identify common sketches from
existing code

Approximate Computing

quality bounds
into
approximate programs

Precise Implementation → Approximate Program Synthesis


Approximate Computing
Hardware Synthesis
Black Box Systems
Program Synthesis
Statistical Machine Learning

Approximate Computing

Hardware Synthesis

Black Box Systems

Program Synthesis

Statistical Machine Learning

Approximate Computing

Hardware Synthesis

Black Box Systems

Program Synthesis

Statistical Machine Learning

Thanks!

PLSE W