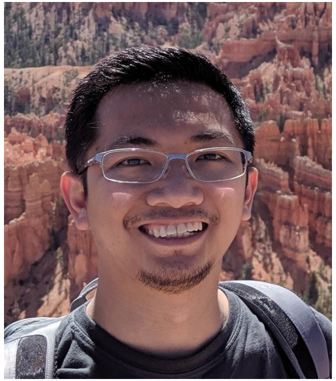
Understanding how novice programmers solve novel programming problems



Francis Castro
@_franciscastro_

Kathi Fisler
@KathiFisler

07 August 2019 University of Massachusetts Amherst

(Let's frame this a little bit)

What…

… do we know about how novices problem-solve in programming?

… is the learning context of the novices we're studying?

… do we challenge our novices to do?

(Let's frame this a little bit)

What…

… do we know about how novices problem-solve in programming?

… is the learning context of the novices we're studying?

… do we challenge our novices to do?

Soloway, Spohrer, Anderson (late 1980s)

Novices retrieve and use **plans** to write code.

Plans: organization of **tasks** or **code** that relate to the components of a problem

Rist (1990s)

Retrieval

Creation

```
sum = 0
for each num in input_list:
     sum = sum + num
return sum
```

(Let's frame this a little bit)

What...

... do we know about how novices problem-solve in programming?

... is the learning context of the novices we're studying?

... do we challenge our novices to do?

College students enrolled in **CS1**-level courses learning programming through the **design-recipe**.

Example: Write a function to sum a list of numbers

Describe the shape/ structure of the data

```
; A list-of-number is:
; - empty or
; - (cons number list-of-number)

(define even-nums (list 2 4 6))
```

Describe the expected behavior

```
; sum-nums : list-of-numbers -> number
; Produces the sum of all numbers in the list

(check-expect (sum-nums even-nums) 12)
```

Code skeleton based on the structure of the data

```
; (define (list-fxn list-input)
;   (cond [(empty? list-input) ... ]
;         [(cons? list-input) ... (first list-input)
;                                 (list-fxn (rest list-input)) ... ]))
```

Fill in the function details

```
(define (sum-nums nums-list)
  (cond [(empty? nums-list) 0 ]
        [(cons? nums-list) (+ (first nums-list)
                              (sum-nums (rest nums-list)))]))
```

(Let’s frame this a little bit)

What…

… do we know about how novices problem-solve in programming?

… is the learning context of the novices we’re studying?

… do we challenge our novices to do?

What if we gave students programming problems with some degree of “newness”?

Rainfall problem (a classic in CSEd research!)

Find the average of nonnegative numbers in a list of numbers up to a sentinel (-999), if the sentinel appears. If the average can’t be computed, return -1

```
(list 1 -3 2 3 -999 8 0) -> 2
```

- Have seen lists and most of the task-components (summing, counting, removing elements)
- May require integrating familiar tasks in new ways

Max-Temperatures problem

Given a list of sublists separated by a delimiter, where each sublist is a list of numbers, produce a list of the maximum values of each sublist.

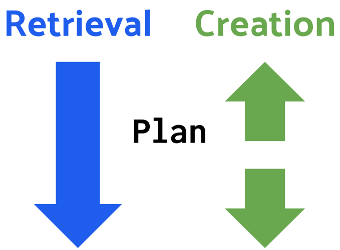
```
(list 40 42 “d” 50 “d” 56 52 50)
-> (list 42 50 56)
```

- Have seen lists and some task-components (max)
- Have not seen sublists embedded in a flat list
- May require plans just beyond what students have seen so far

(Let's frame this a little bit)

What…

… do we know about how novices problem-solve in programming?



… is the learning context of the novices we're studying?

College students enrolled in **CS1**-level courses learning programming through the **design-recipe**.

… do we challenge our novices to do?

Programming problems with some degree of "newness" (just beyond what students have seen so far)

RQ: How do CS1 students navigate through their knowledge of (1) plans and (2) programming tools to solve new programming problems?

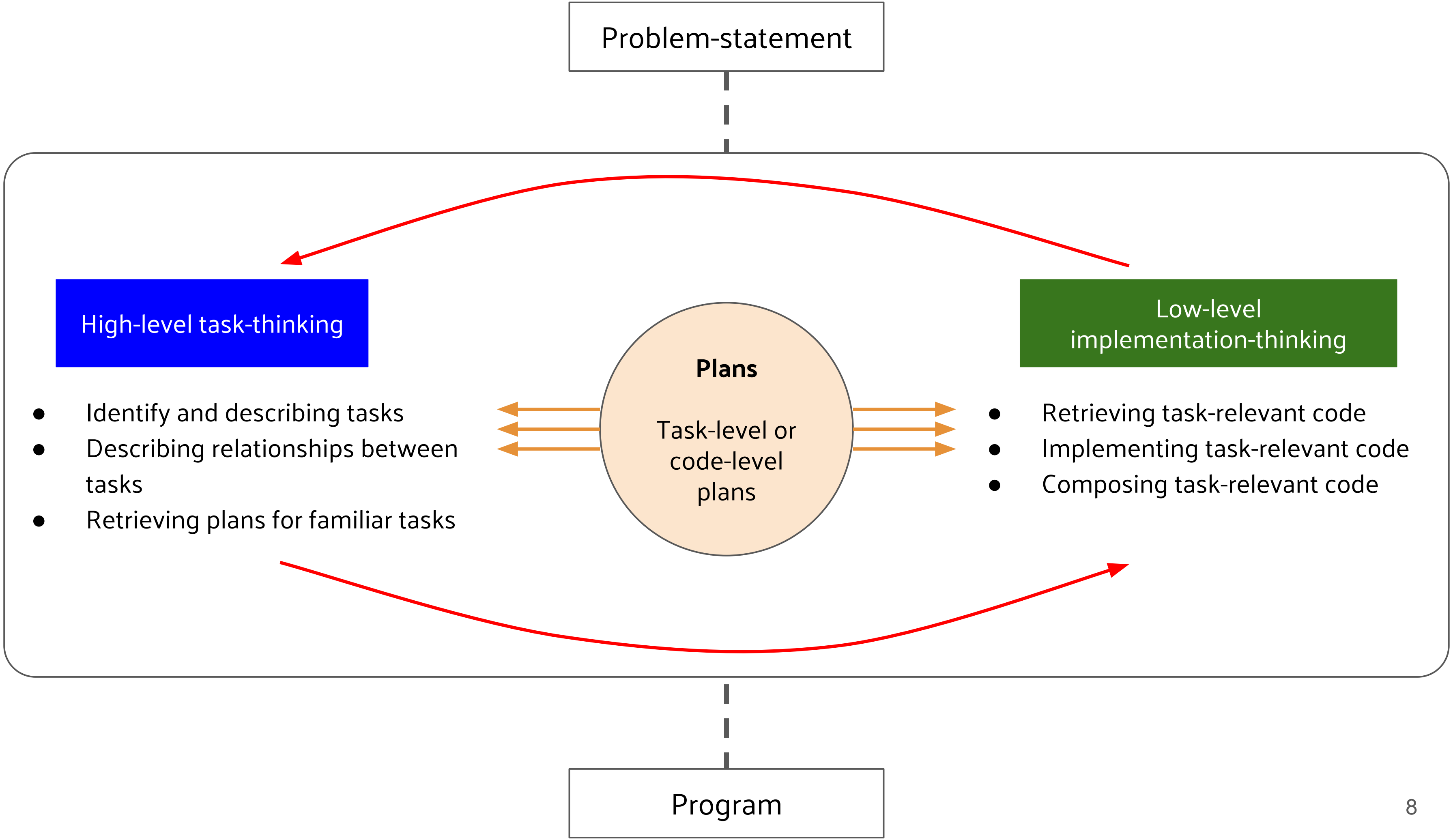
Goal: Develop ways-of-thinking (frameworks) about how students navigate plan and tool knowledge to solve programming problems

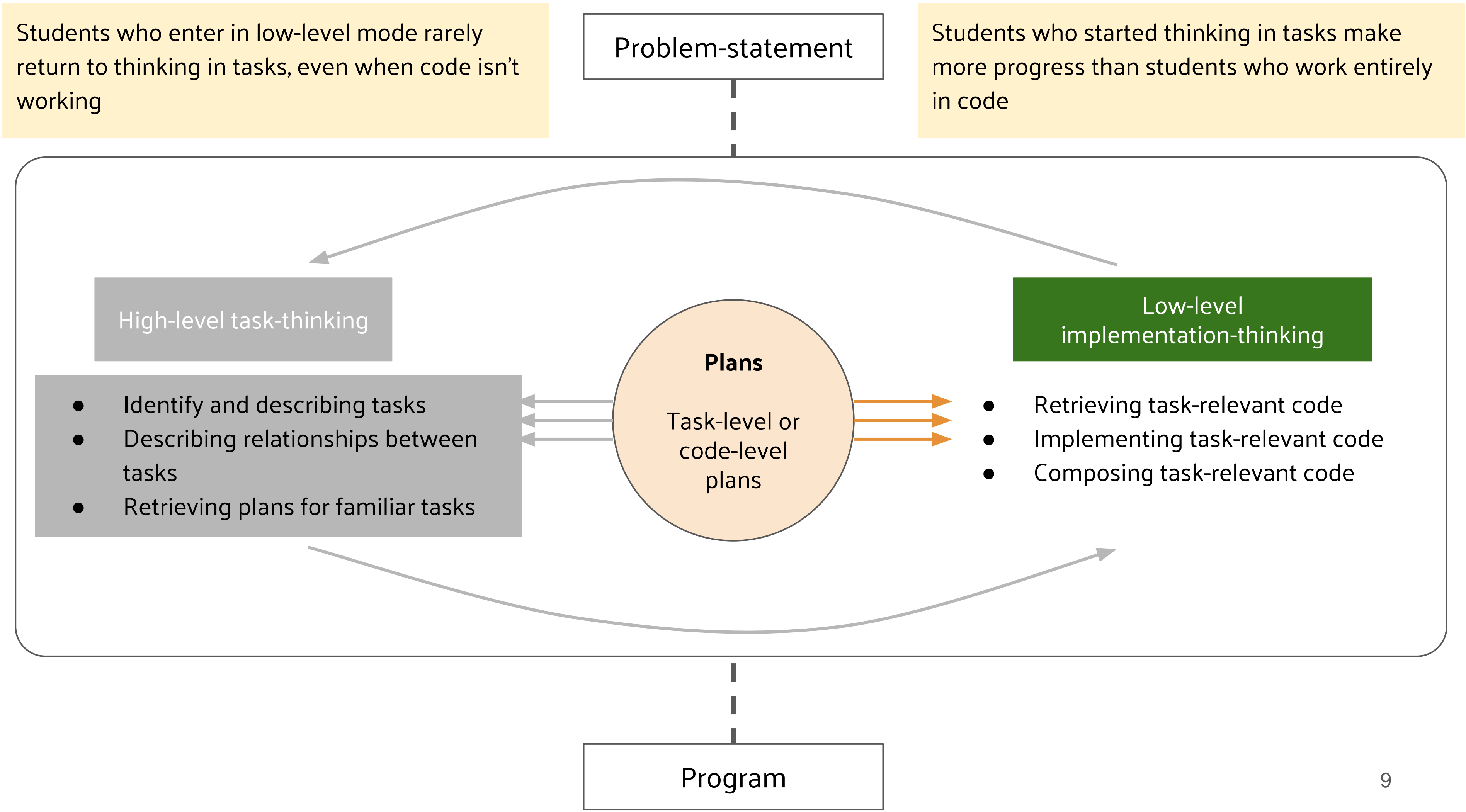
What does our data look like?

We go back to classic techniques used in cognitive science – **think-alouds**!

- Give students a programming problem
- Students think-aloud while solving the problem (audio-recorded)
- Post-hoc interviews (also recorded)
- Think-aloud and interviews are transcribed for analysis

= hundreds of hours of student verbalizations, explanations, decisions for analysis


Problem-statement
High-level task-thinking
Low-level implementation-thinking
Plans
Task-level or code-level plans
Identify and describing tasks
Describing relationships between tasks
Retrieving plans for familiar tasks
Retrieving task-relevant code
Implementing task-relevant code
Composing task-relevant code
Program


Students who enter in low-level mode rarely return to thinking in tasks, even when code isn't working
Problem-statement
Students who started thinking in tasks make more progress than students who work entirely in code
High-level task-thinking
Identify and describing tasks
Describing relationships between tasks
Retrieving plans for familiar tasks
Plans
Task-level or code-level plans
Low-level implementation-thinking
Retrieving task-relevant code
Implementing task-relevant code
Composing task-relevant code
Program

Students who enter in low-level mode rarely return to thinking in tasks, even when code isn't working

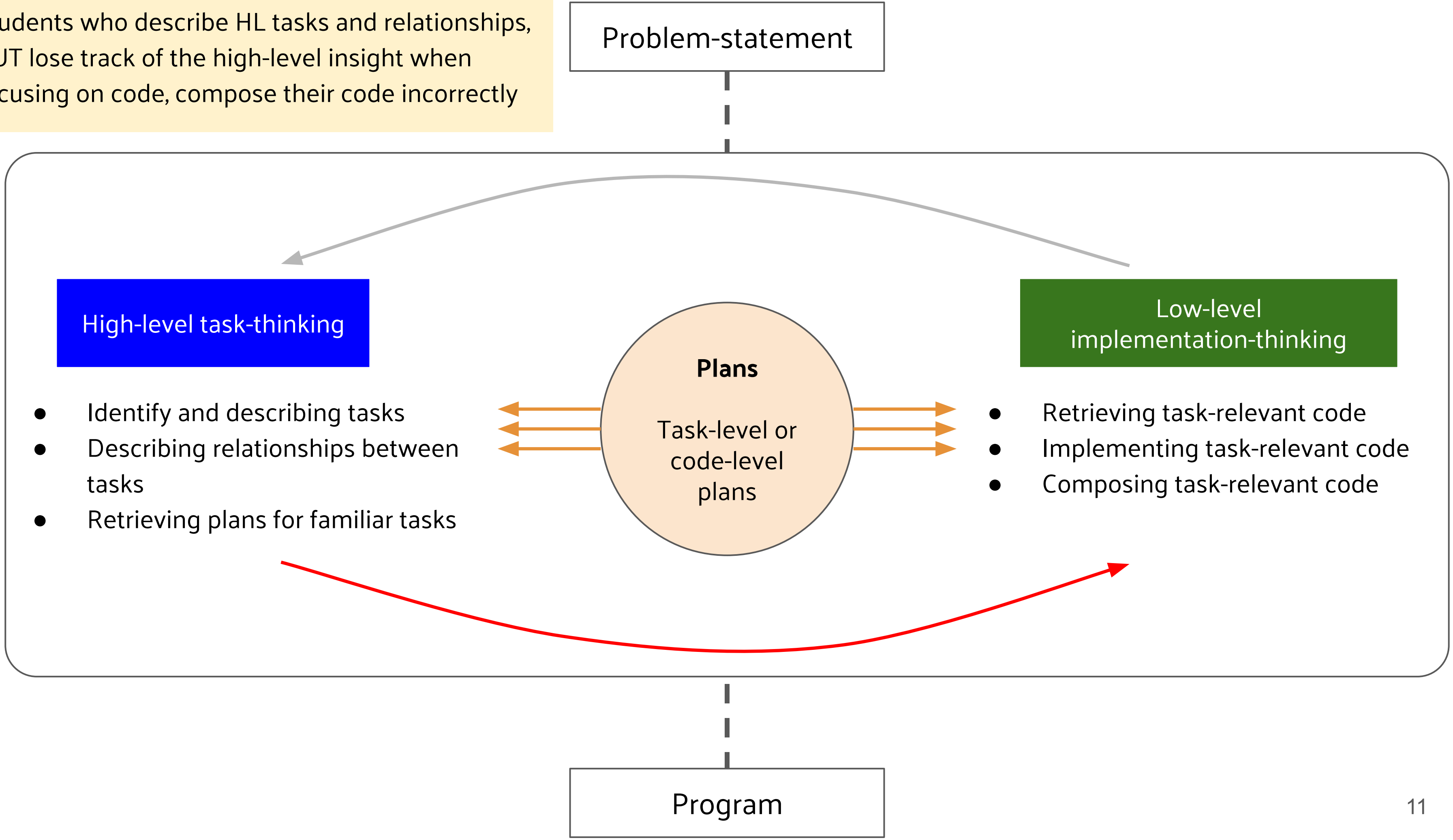
Students who started thinking in tasks make more progress than students who work entirely in code

Sounds like a for-loop/
Sounds like a recursive template…

Should take out the numbers before -999 first, then take out negatives. Then sum and count…

Find the average of nonnegative numbers in a list of numbers up to a sentinel (-999), if the sentinel appears. If the average can't be computed, return -1

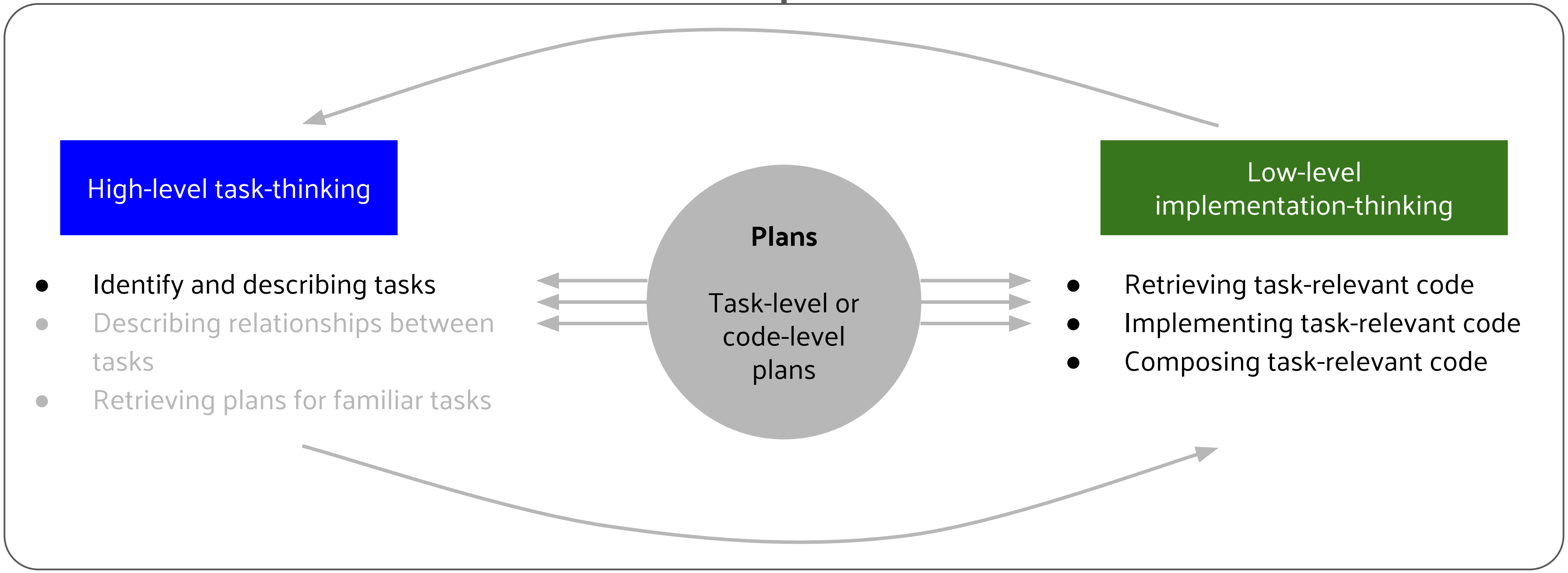
Students who describe HL tasks and relationships, BUT lose track of the high-level insight when focusing on code, compose their code incorrectly



Even when students can retrieve a plan, this does not mean they can necessarily see the subparts of the plan as things that could be separately implemented in code

Some can describe high-level plans, but lack concrete details to establish relationships between identified task-components

Problem-statement



Program

Even when students can retrieve a plan, this does not mean they can necessarily see the subparts of the plan as things that could be separately implemented in code

Just jammed the formula into the list-template

```
(define (average input)
  (cond [(empty? input) empty]
        [(cons? input) (/ (+ (first input) (average (rest input)))
                          (length input))]))
```

Correct version

```
(define(average input)
  (cond [(empty? input) -1]
        [(cons? input) (/ (sum input) (count input))]))

(define (sum input)
  (cond [(empty? input) 0]
        [(cons? input) (+ (first input) (sum (rest input)))]))

(define (count input)
  (cond [(empty? input) 0]
        [(cons? input) (+ 1 (count (rest input)))]))
```

Some can describe high-level plans, but lack concrete details to establish relationships between identified task-components

Max-Temperatures problem: *Given a list of sublists separated by a delimiter, where each sublist is a list of numbers, produce a list of the maximum values of each sublist.*

```
(list 40 42 “d” 50 “d” 56 52 50) -> (list 42 50 56)
```

“I think what would be the best if I split it up into lists and then worked through each list individually but I’m not sure quite how to do that.”

Doesn’t describe the “glue” that would make task-components work together
(i.e. how to store the “splitted” lists, e.g. list of lists)

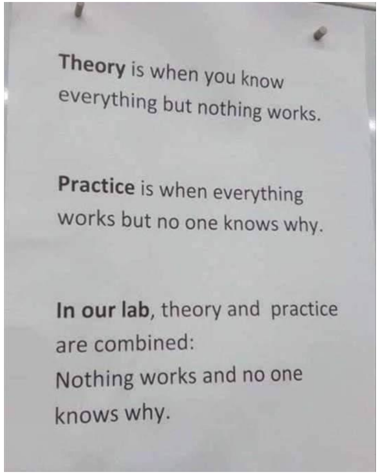
“you want to check [each element] and when you hit the [delimiter], you want to process the [numbers] before it, and then you want to [repeat the process] and continue doing that. […] I think I have the right idea […] but the problem is once I hit the [delimiter], I don’t know what to do.”

Doesn’t describe the “glue” that would make this work
(i.e. how to keep track of the sublist being processed and how to store the “processed” sublists)

If we can figure out patterns of where (in the HL-LL dynamic) students are struggling when solving problems, we can catch them at those points at potentially design interventions around those points

- Learning activities and assessments
- CS1-level IDEs (BlueJ, DrRacket, etc.)
- Modalities

(Future research topics!)



functions? We don't really know what makes programming so hard, and we don't yet have enough theory to explain why it works when we get it right.

- Mark Guzdial (BLOG@CACM: Learning Computer Science is Different than Learning Other STEM Disciplines, Jan. 5, 2018)