Université de Montréal

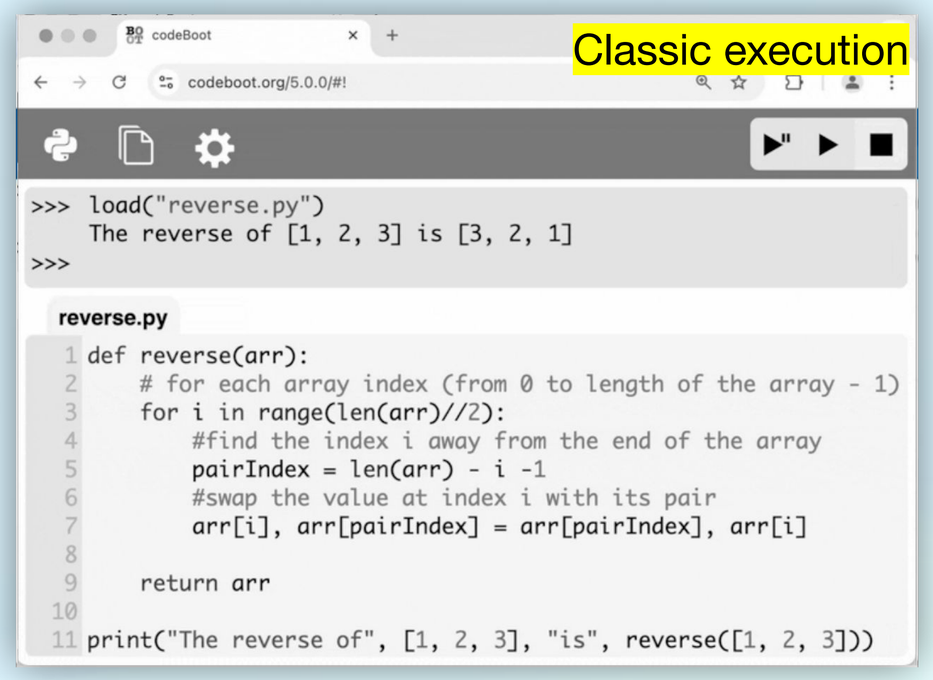
Does Back-stepping Help Programmers Debug?

Léonard Oest O'Leary, Jingyue Zhang, Olivier Melançon, Ian Arawjo and Marc Feeley

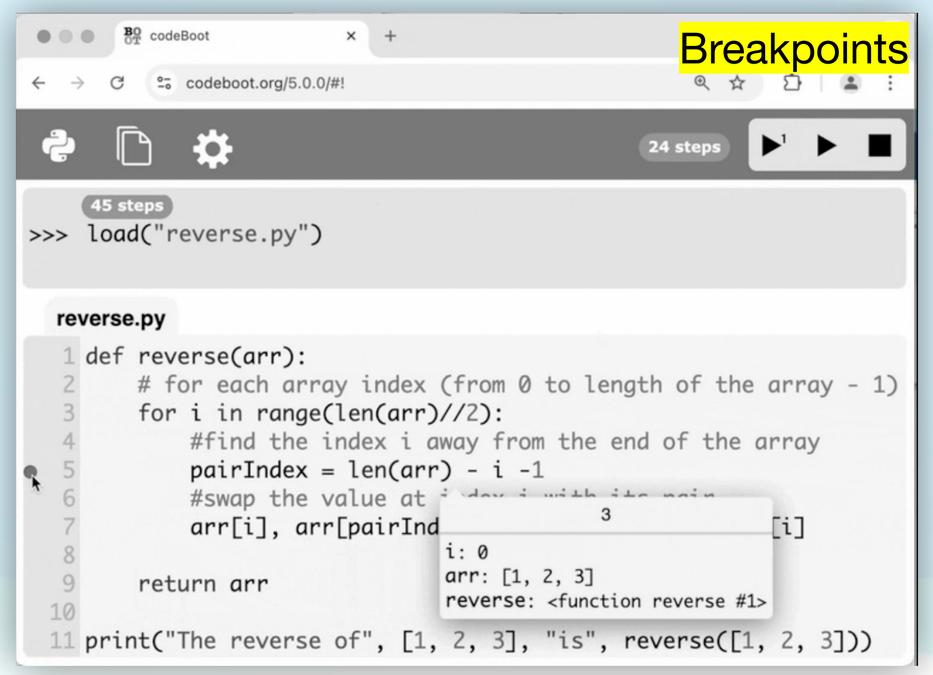
10th of March 2026

PLATEAU2026

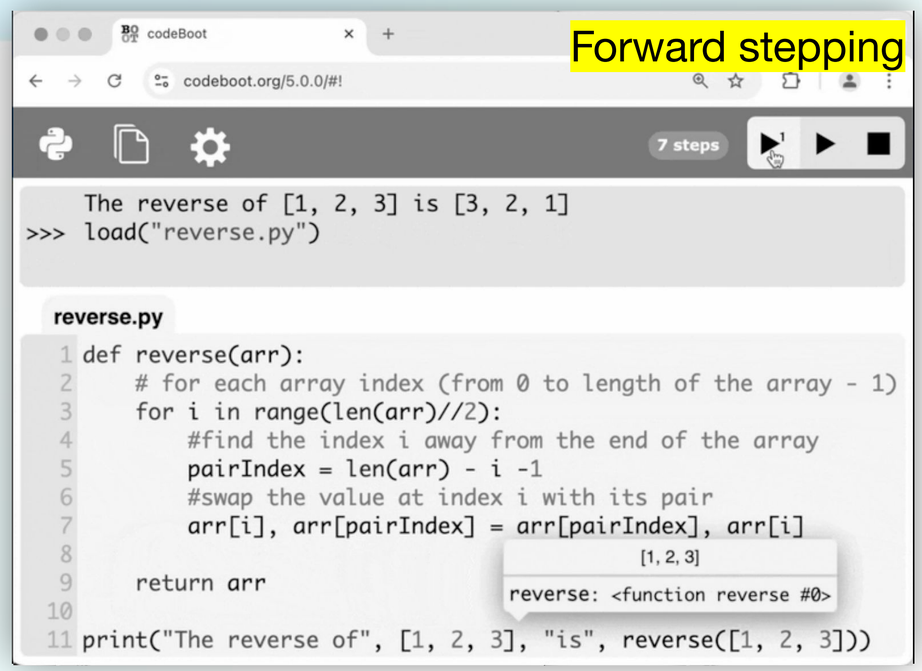
Classic execution



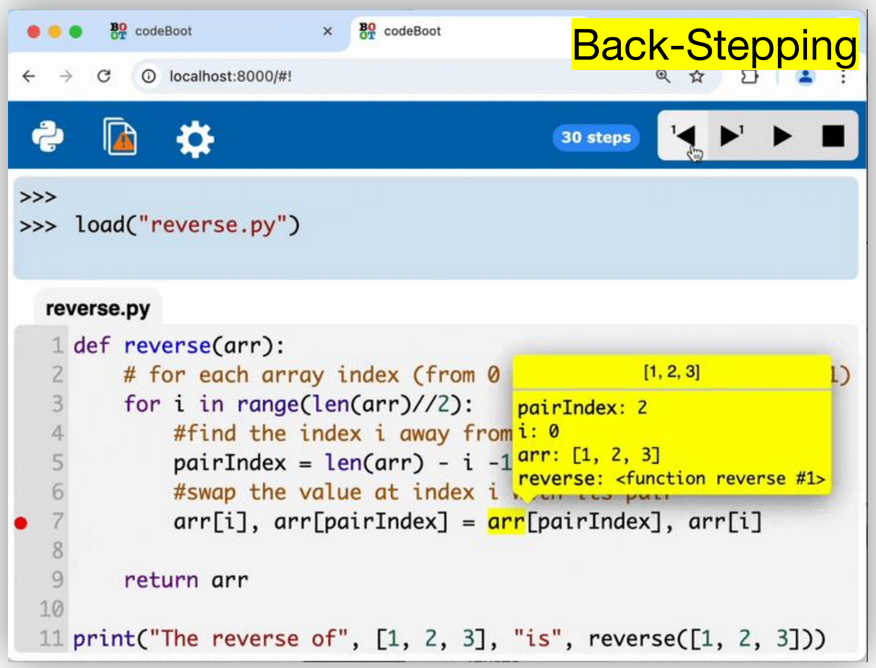
Breakpoints



Forward stepping

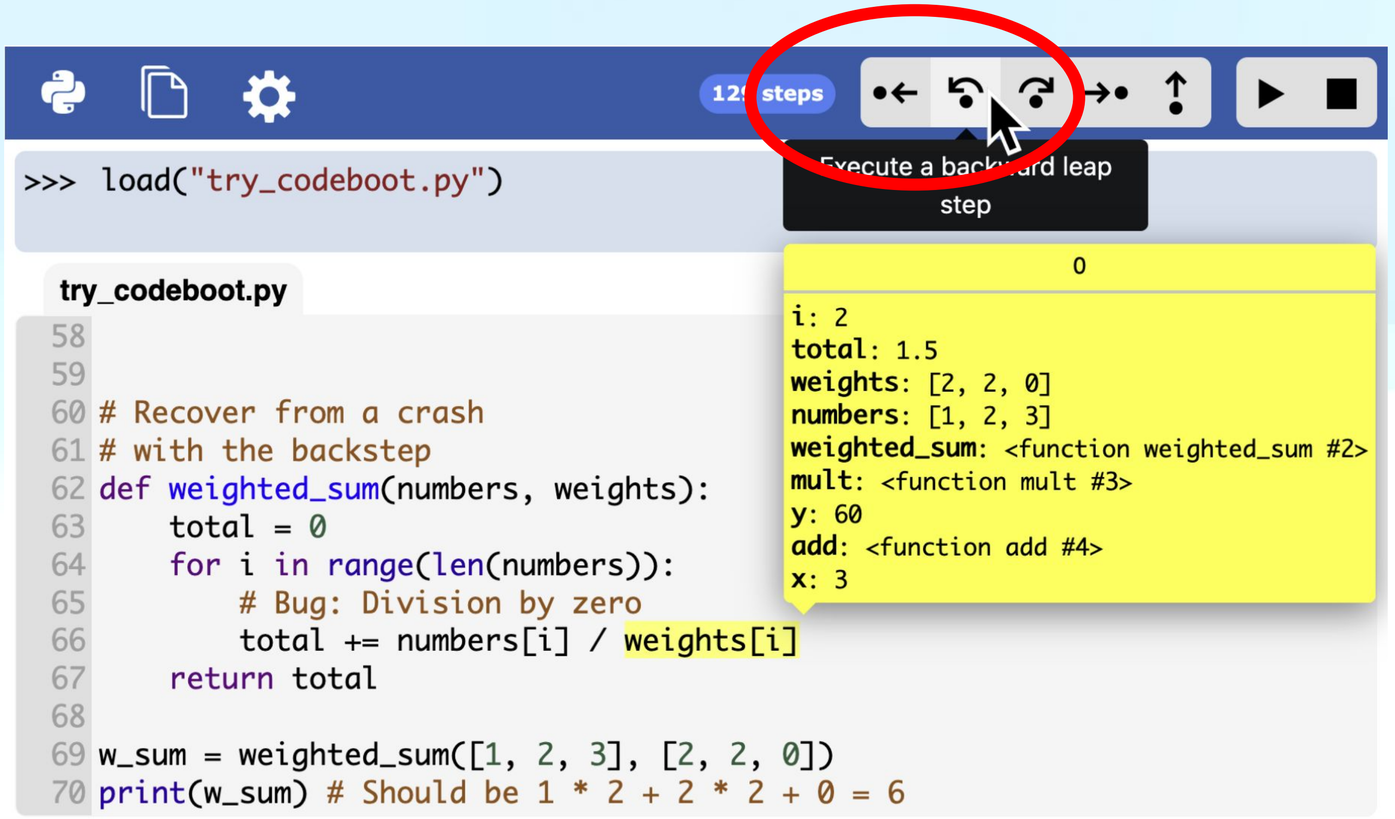


Back-Stepping

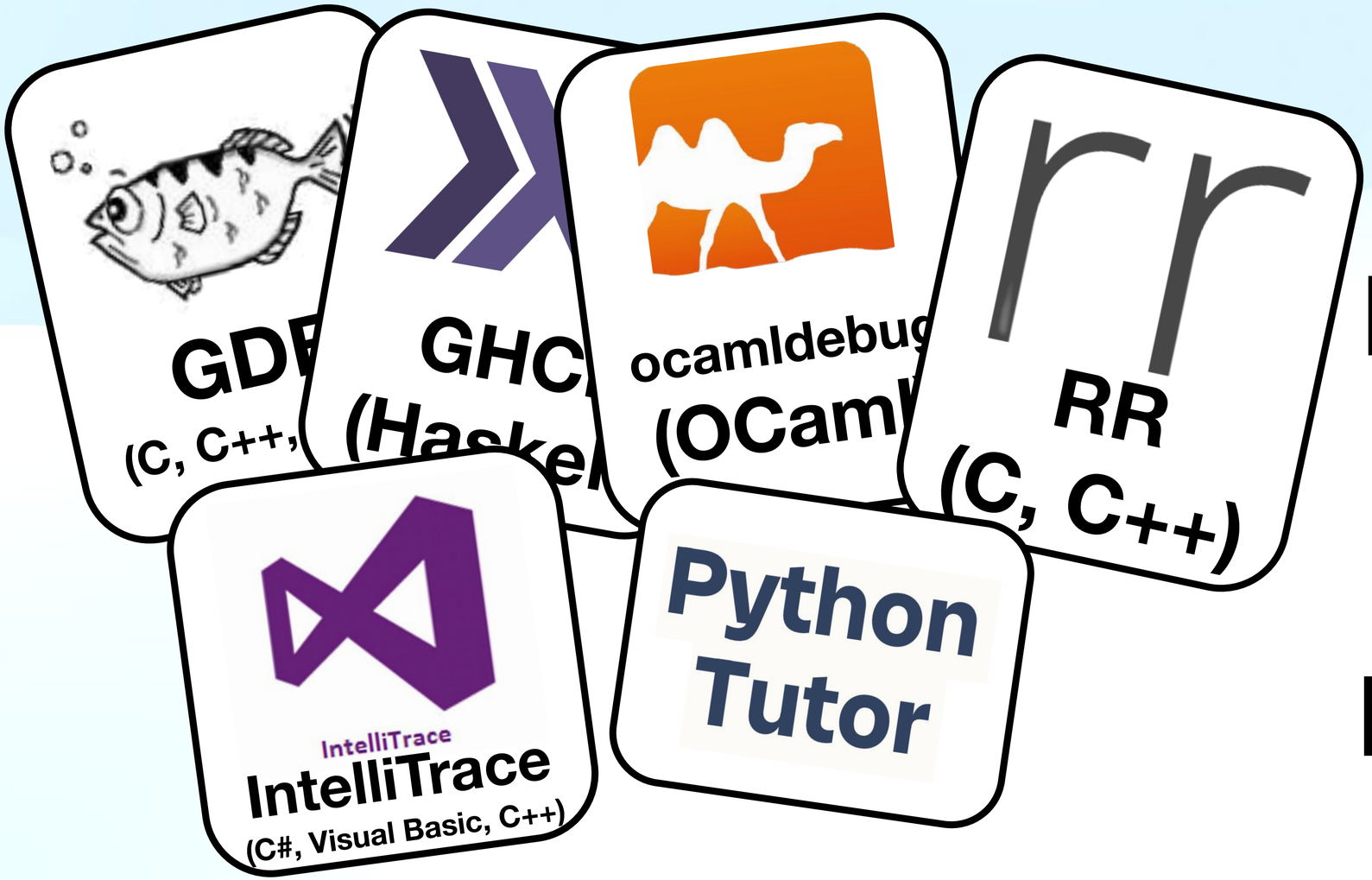


Digression on codeBoot

- Online code editor
- Beginner focused
- Executes Python
- Developed at UdeM (compiler lab)



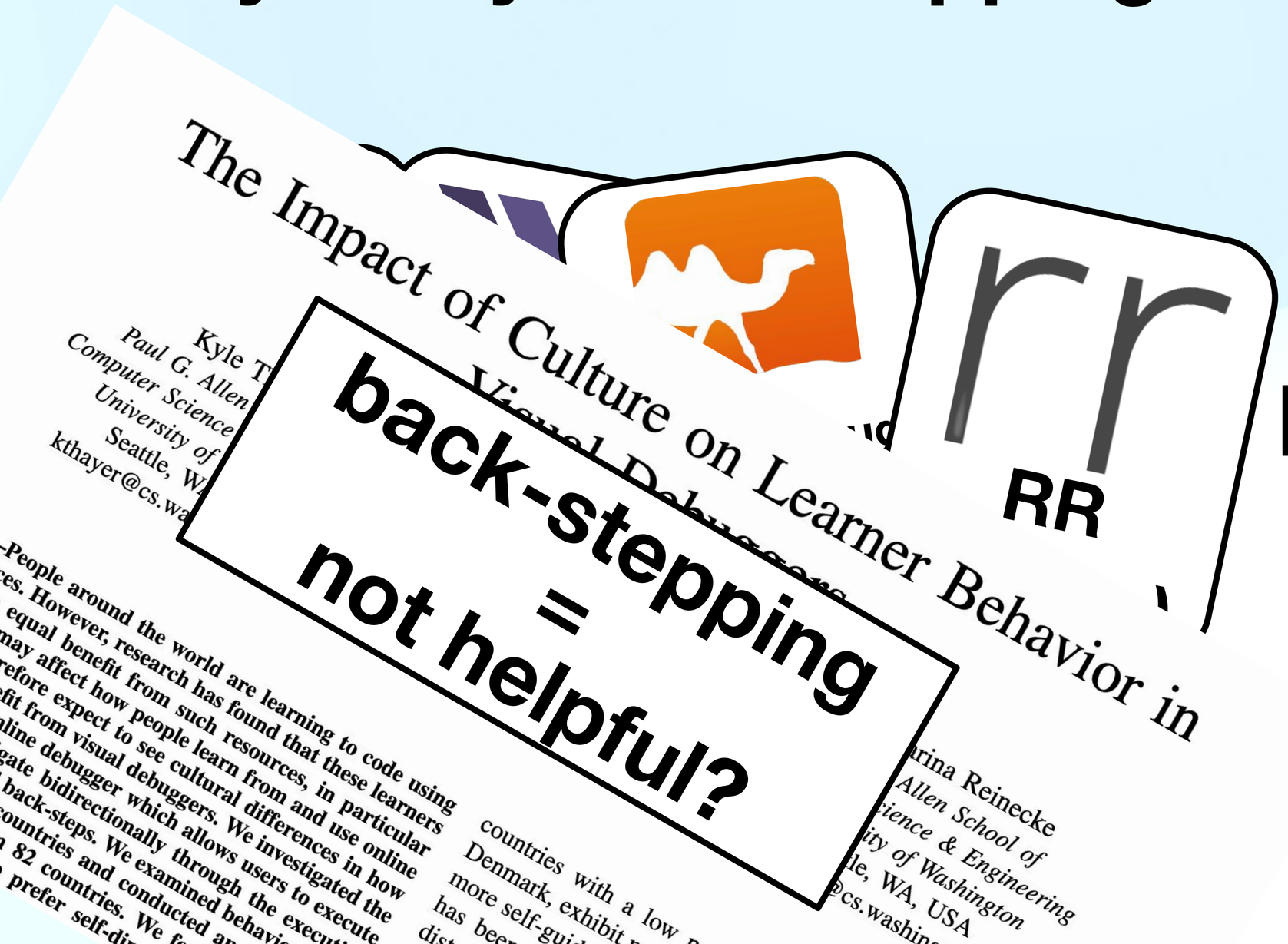
Why study back-stepping



But…

No **<u>controlled</u>** study evaluating back-stepping

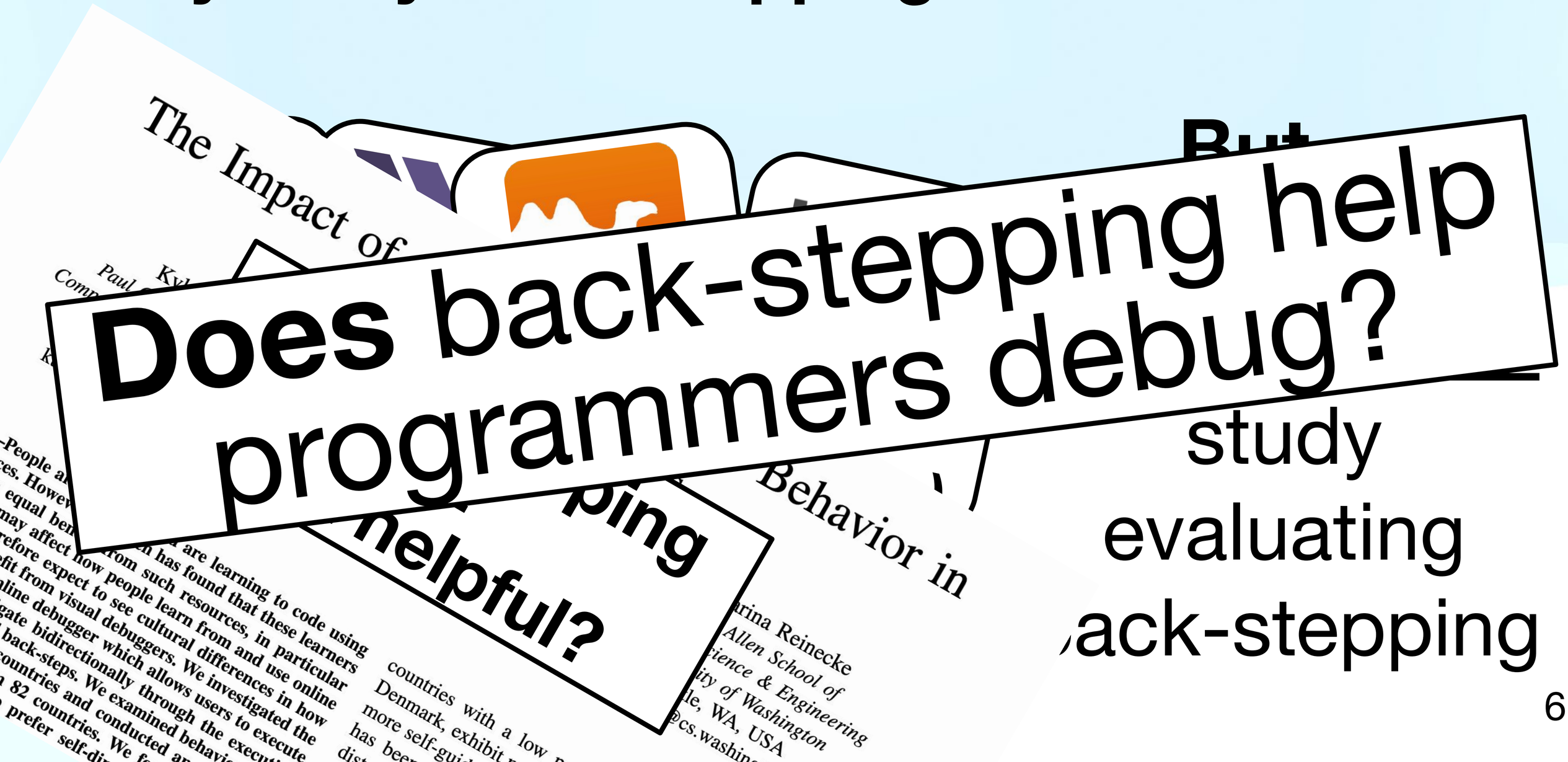
Why study back-stepping



But…

No **<u>controlled</u>** study evaluating back-stepping

Why study back-stepping

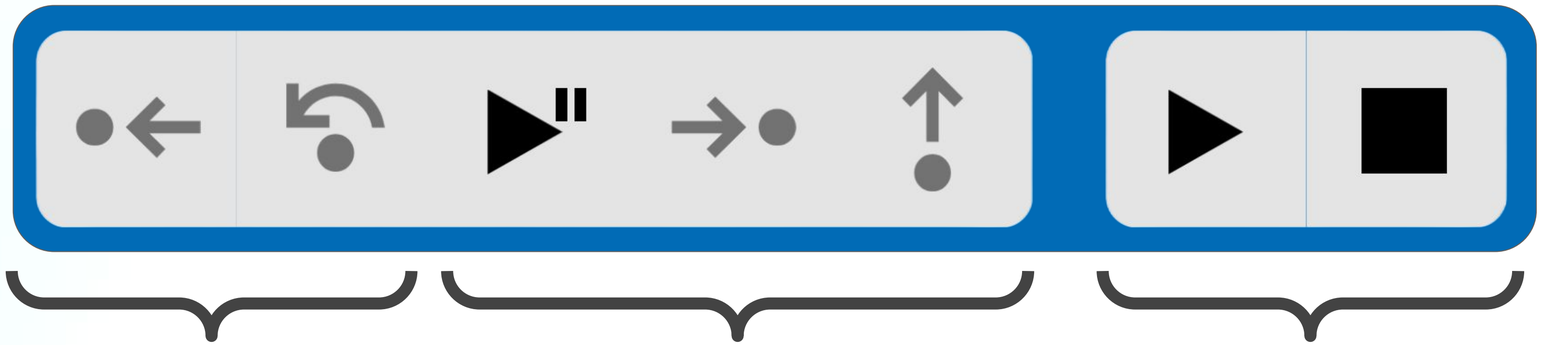


Research questions

- Does back-stepping **increase** debugging success? (Q1)
- Do programmers think that back-stepping **is useful**? (Q2)
- **What differences in strategies** can be observed when different stepping features are available? (Q3)

Study Design

- **Within-subject mixed methods** study with **three conditions**:
  - **No debugger** (no breakpoints)
  - **Step**, breakpoints
  - **Back-step**, **step** and breakpoints

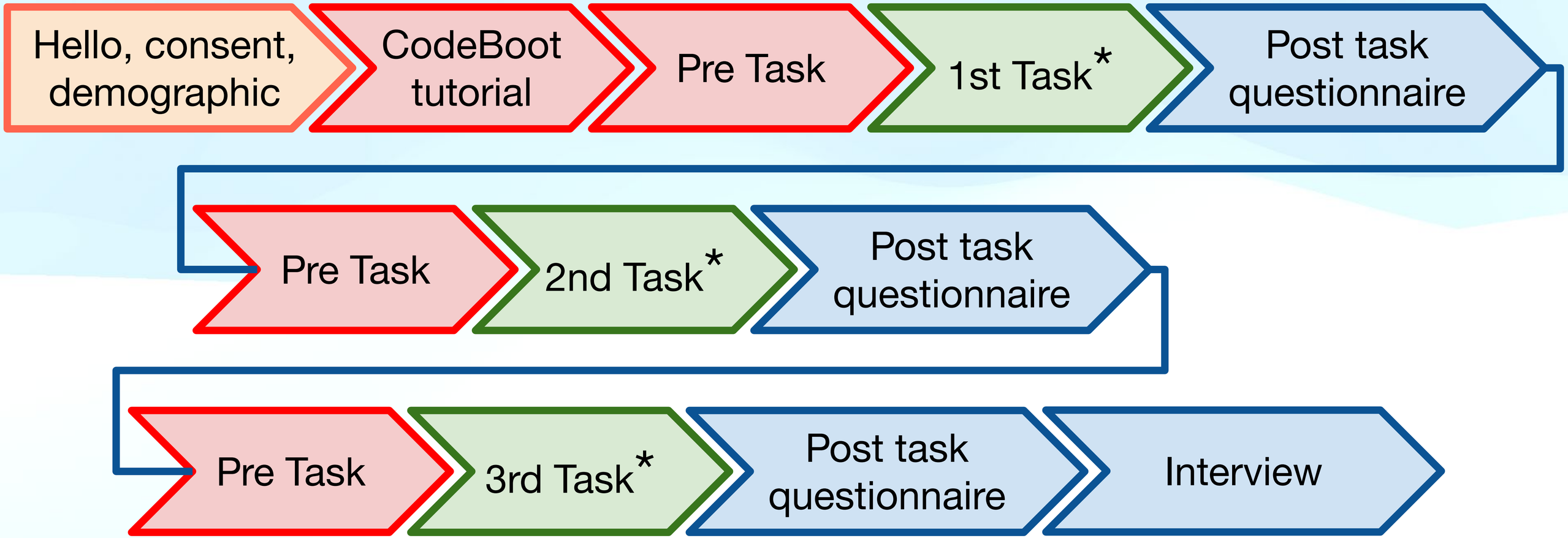


Back-step **Step** **No debugger**

Study Design

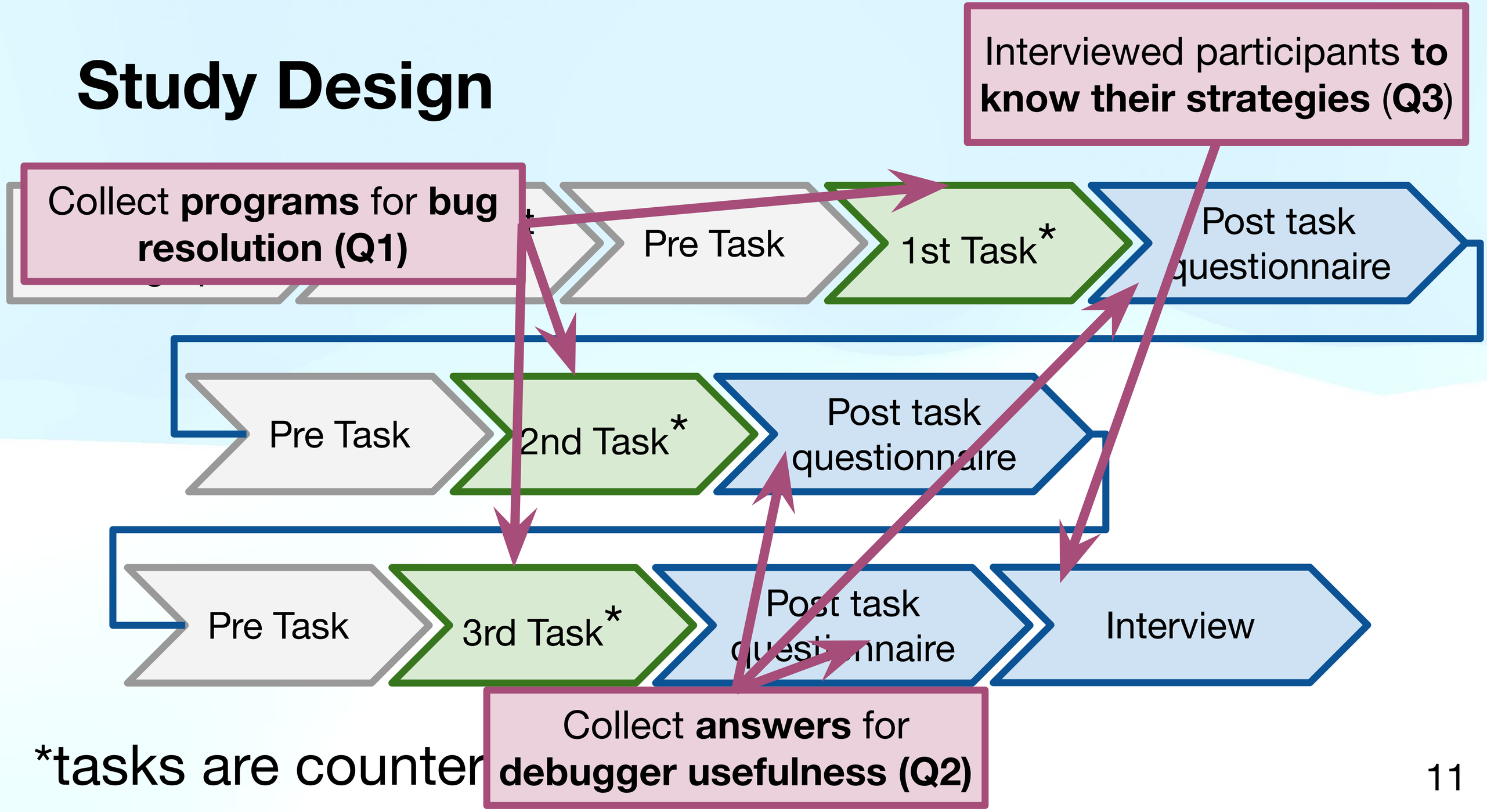
- Bias towards **back-stepping**:
  - Tasks designed for back-stepping
  - Tutorial and time to understand the feature
- Recruited **18** experienced participants:
  - ≥2 years python
  - 8 years of programming experience on average
  - 17/18 use visual debuggers once a week
  - 20$ compensation

Study Design



*tasks are counterbalanced

Study Design



*tasks are counter

Tasks design

Two evaluation metrics:
Test-rated and **Expert-rated**

```python
# [EN] Execute the code to see the first bug
# [FR] Execute le code pour voir le premier bogue

def quick_sort(objs, obj_key):
    """
    [EN] Sorts a list of objects `objs` in ascending order based on the key `obj_key`
         using the quick sort algorithm.
    [FR] Trie une liste d'objets `objs` par ordre croissant selon la clé `obj_key`
         en utilisant l'algorithme de tri rapide.
    """

    if len(objs) <= 2:
        return objs

    pivot_index = len(objs) / 2
    left, right = partition(objs, obj_key, pivot_index)
    pivot_value = objs[pivot_index]

    # [EN] Recursively sort left and right partitions and combine with the pivot.
    # [FR] Trie récursivement les partitions gauche et droite et les combine avec le pivot.
    return quick_sort(left, 'value') + [pivot_value] + quick_sort(right, 'value')

def partition(objs, obj_key, pivot_index):
    """
    [EN] Splits `objs` into two lists based on the pivot value.
         Elements <= pivot go left, others go right.
    [FR] Divise `objs` en deux listes selon une valeur pivot.
         Les éléments <= pivot vont à gauche, les autres à droite.
    """

    # [EN] Swap the pivot with the last element for easier partitioning.
    # [FR] Échange le pivot avec le dernier élément pour simplifier la partition.

        # [EN] Place elements <= pivot in `left`, others in `right`.
        # [FR] Place les éléments <= pivot dans `left`, les autres dans `right`.
        if elem[obj_key] <= pivot_value:
            left.append(elem)
        else:
            right.append(el

    return left, right
```

```python
def test(result, expected):
    if result == expected:
        print("PASS")
    else:
        print("FAIL (got", repr(result), "expected", repr(expected), ")")

print("Tests will be displayed after this line")
# Test 1
objs = [{'value': 2}, {'value': 1}, {'value': 1}]
expected = [{'value': 1}, {'value': 1}, {'value': 2}]
test(quick_sort(objs, 'value'), expected)

# Test 2
objs = [{'amount': 0}, {'amount': 0},
        {'amount': 1}, {'amount': 0},
        {'amount': 0}, {'amount': 0}]

expected = [{'amount': 0}, {'amount': 0},
            {'amount': 0}, {'amount': 0},
            {'amount': 0}, {'amount': 1}]
test(quick_sort(objs, 'amount'), expected)

# Test 3
objs = [{'dollars': 3}, {'dollars': 2}, {'dollars': 1}]
expected = [{'dollars': 1}, {'dollars': 2}, {'dollars': 3}]
test(quick_sort(objs, 'dollars'), expected)

# Test 4
objs = [{'things': 1}, {'things': 5}, {'things': 3}, {'things': 2}]
expected = [{'things': 1}, {'things':
test(quick_sort(objs, 'things'), expe
```

4 bugs to find in the first half

1 test case/bug

Each task has the same structure

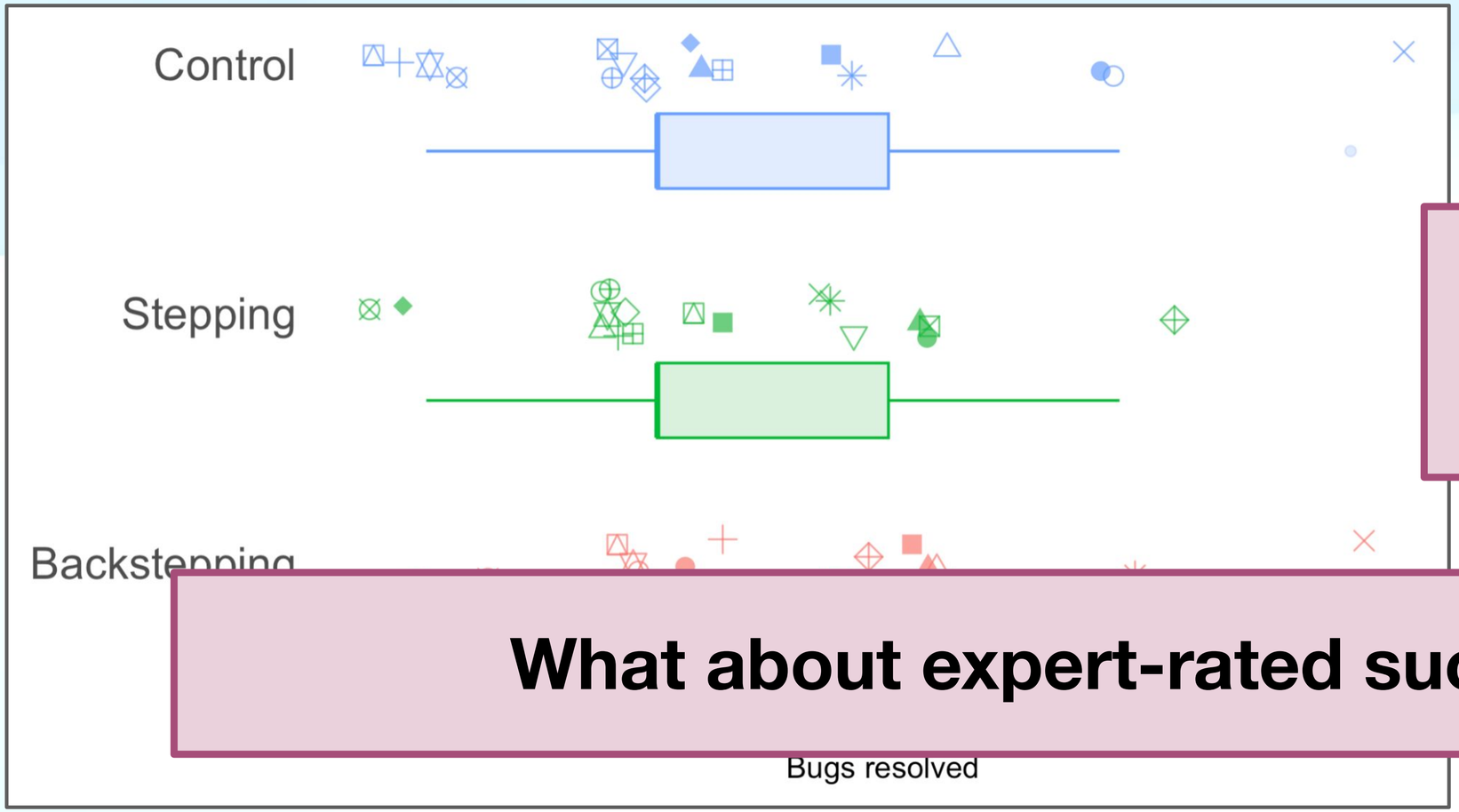
Quantitative findings

Qualitative methods

- Linear mixed effects model in R (LMM)
- Analysed:
  - Fixed & interaction effects of Condition, Task, and Order
  - Participants random effect
- Bronfolli corrections for p-values
- Ensured normality (via Q-Q plots) and homoscedasticity of residuals

Does backstepping **increase debugging success? (Q1)**

Test-rated success

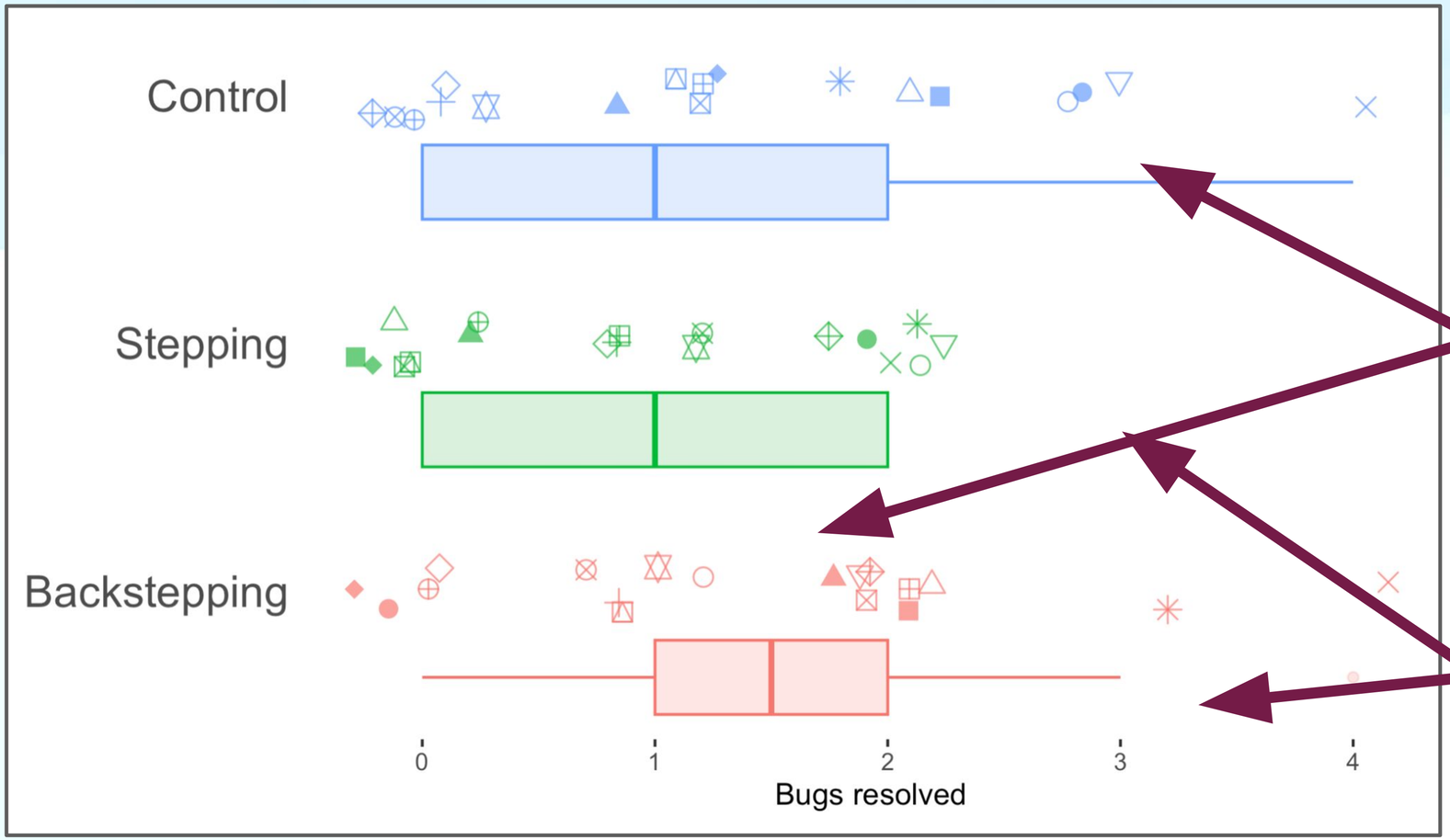


Results are not significant

What about expert-rated success?

Does backstepping **increase debugging success? (Q1)**

Expert-rated success



contrast	estimate	SE	df	t.ratio	p.value
B - C	0.111	0.19	12	0.586	0.8299
B - S	0.500	0.19	12	2.638	0.0526
C - S	0.389	0.19	12	2.052	0.1421

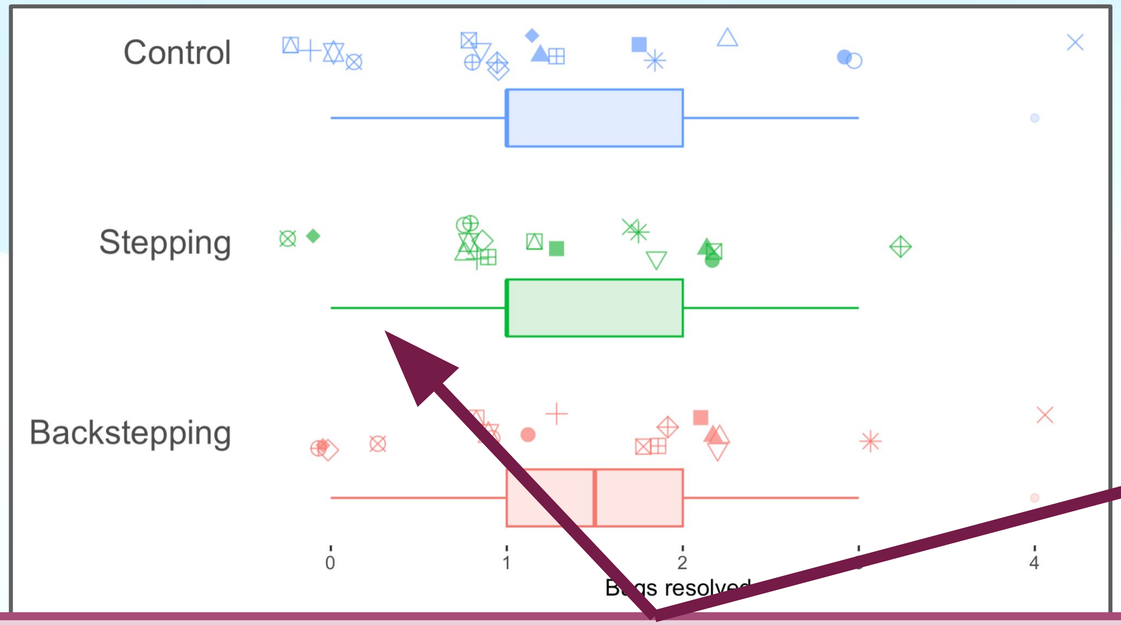
Control vs Back-stepping
Control vs Stepping
Not significant
Not used to codeBoot's UI

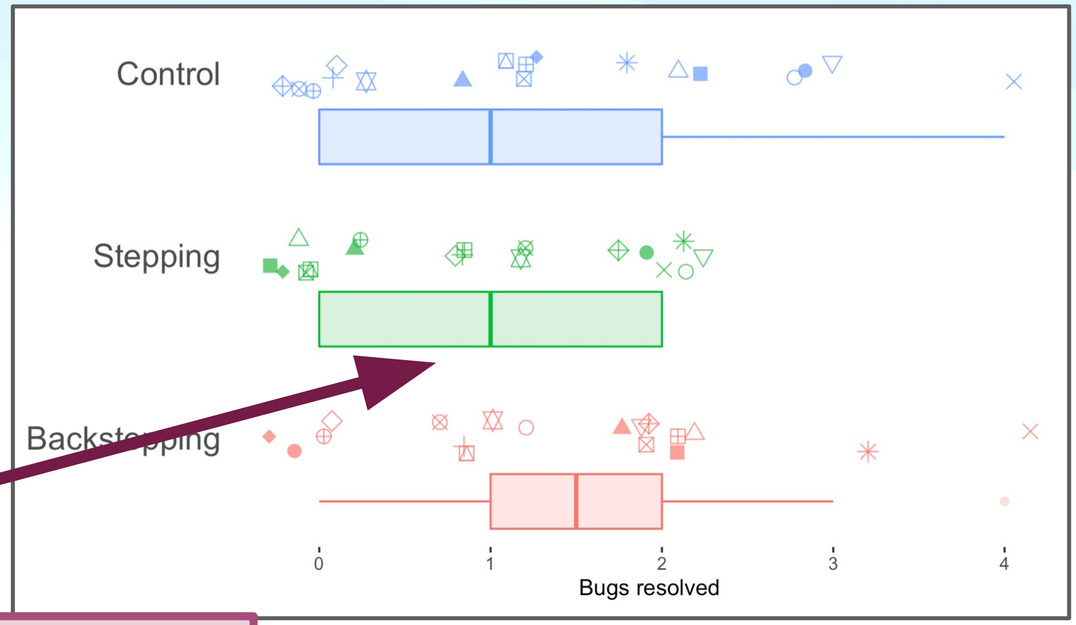
Stepping vs Back-stepping
Almost significant (p=0.0529)
Medium effect size (0.5 bugs/task)

WARNING: SPECULATIVE

Does back-stepping **increase debugging success? (Q1)**

Test-rated vs **Expert-rated** success





Analyzing the differences…

1. 62% (10/16) of corrections are localized one task
2. 50% (8/16) involve the 1st bug in this task

Digression

Test-rated vs **Expert-rated** success

WARNING: SPECULATIVE

Easy “wrong solve”:
replace by G[current_node - 1]
(but makes the test pass)

```python
node_counter = 0
def add_node(G):
    """
    [EN] Adds a new node to a graph `G`. Nodes are represented by numbers.
    [FR] Ajoute un nouveau noeud au graphe `G`. Les noeuds sont representés
         par des nombres.
    """
    global node_counter

    node = node_counter
    G[node] = []
    node_counter += 1

    return node_counter
```

Right solve:
change the returned value

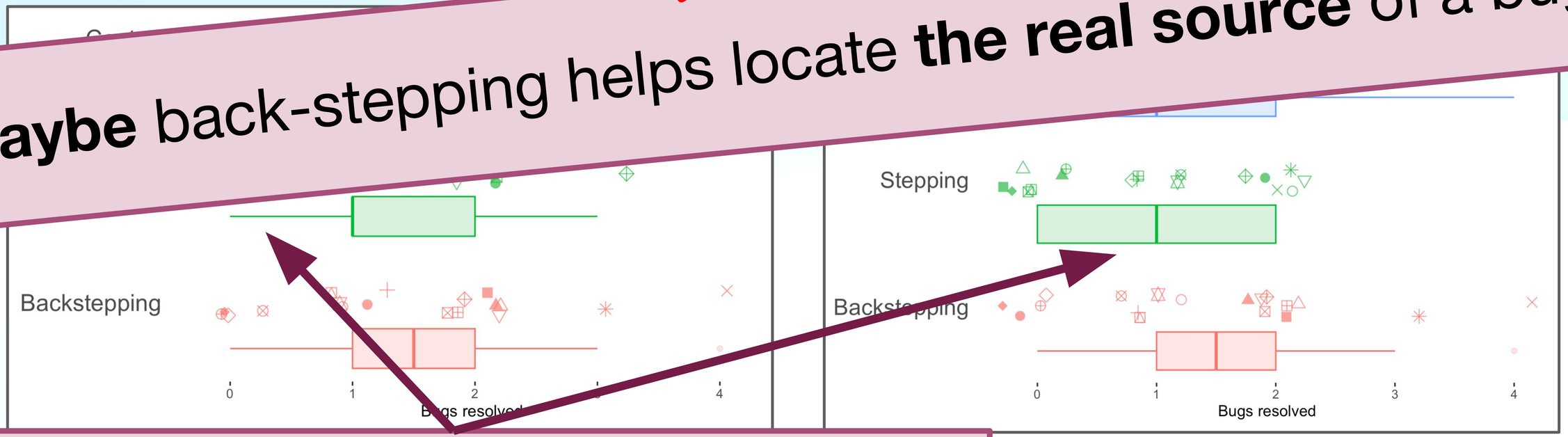
```python
def distance(G, node_a, node_b):
    """
    [EN] Calculate the distance between nodes `node_a` and `node_b` in
    [FR] Calcule la distance entre les noeuds `node_a` et `node_b` dans un

    dist = 0

    # [EN] Uses a breadth-first search (BFS) algorithm to explore the graph.
    # [FR] Utilise un algorithme de recherche en largeur (BFS) pour explorer le graphe.

    es:
    node]:
    n:
    urrent_node)
    ur)

    return -1 # not reachable / non atteignable
```

Digression

Test-rated vs **Expert-rated** success

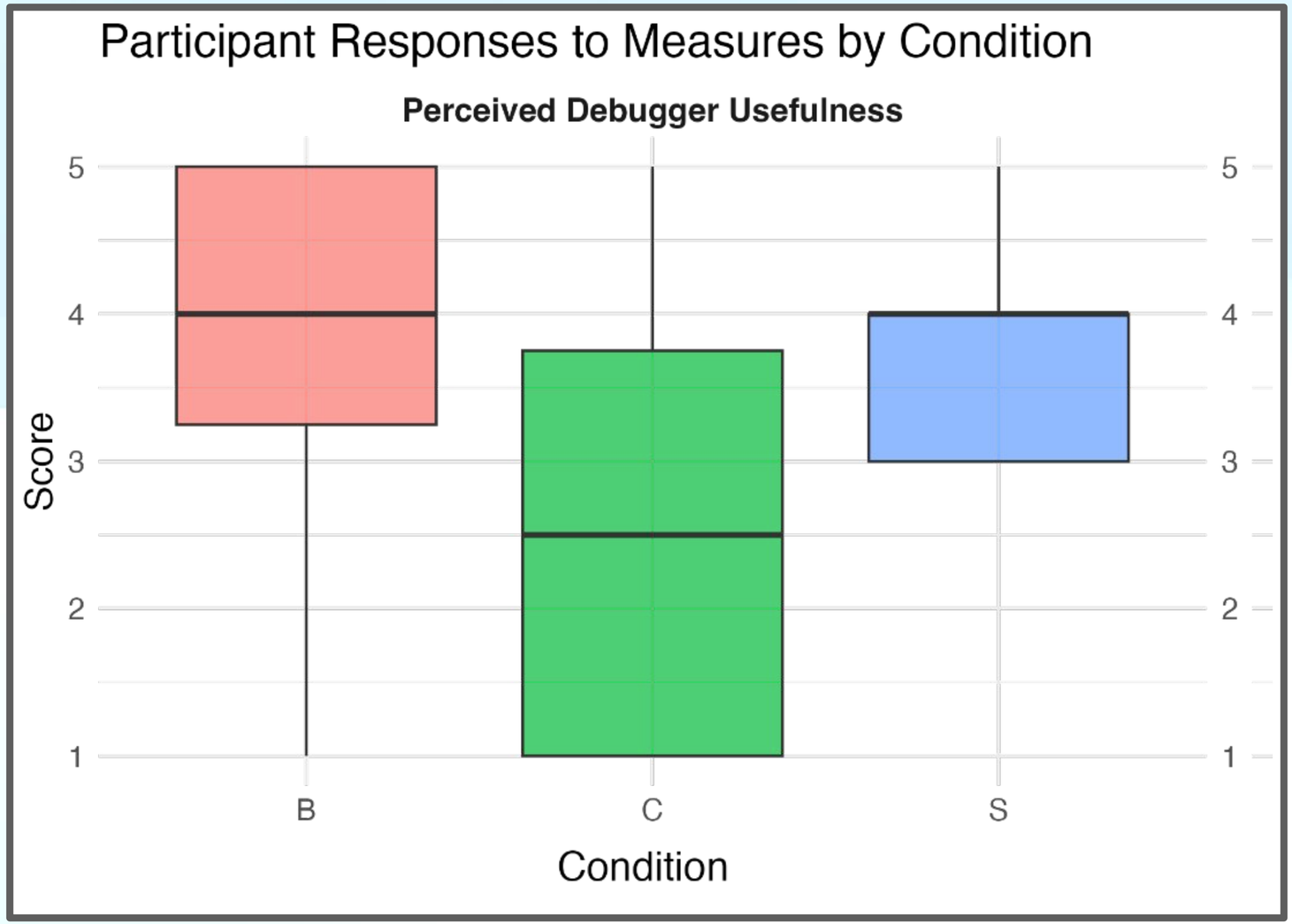
WARNING: SPECULATIVE

Maybe back-stepping helps locate **the real source** of a bug?



This difference might be caused by “wrong” solves!

Is backstepping perceived as **useful? (Q2)**



Back-stepping is
perceived as most helpful to identify bugs

contrast	estimate	SE	df	t.ratio	p.value
B - C	1.52	0.397	22.5	3.837	0.0026
B - S	0.19	0.390	22.1	0.487	1.0000
C - S	-1.34	0.390	22.1	-3.427	0.0072

Qualitative findings

Quantitative methods

- Iterative thematic analysis with affinity diagramming from post-study interviews to derive clusters (codes)
- Two coauthors (me and Jingyue) resolve discrepancies until reaching consensus

Removes the fear of “missing the moment”

1. Without back-stepping

“[...] the moment you pass it, you say to yourself "shit was it that step or the step before?"”
(P6)

“I have to be really careful with going forward cause I don’t wanna start all [...] over again”
(P10)

2. With back-stepping

“[...] you can really isolate the place where your change happens".”
(P6)

“[You] have a bit more freedom.”
(P8)

“I don’t have to load up the data again. I can just backstep.”
(P15)

“if it’s too easy to go back you’ll go faster and it’ll **take you even more time** to find bugs.”
(P1)

Maybe the fear itself is useful?

Strategy changed to failure-first navigation

1. Back-stepping **changes debugging** strategies

" I started **going straight to the error**, and **then doing step-back**"
(P2)

"I launched it, **it crashed**, **I went back** up a few steps to see what was happening."
(P8)

2. Without back-stepping, more **breakpoints** and **prints**

"[when] there was no stepping back, **I had to use breakpoints**"
(P8)

"You have to modify the code, [...] and afterwards you have to clean up"
(P9)

" just step, stepping, [...] with no constraints" and then reflected "it's more important to read the code first."
(P10)

Maybe the proactive "failure first" strategy hinders understanding

Lack of visual feedback makes backstepping hard to master

1. The extra **backward dimension** needs time to **master**

“it didn’t occur to me that I wanted to review something that was happening. I think I was too lost”
(P3)

“I lost time relaunching my code when I could have done four backward steps”
(P8)

2. Participants **asked** for visual feedback

“But just materializing [the stack] somewhere visually.”
(P4)

“What would be good, is to have a preview.”
(P6)

Suggesting that the difficulty of mastering back-stepping **comes from the misunderstanding of one’s position in the dynamic execution**

Conclusion

Conclusion

We performed a within-subject study with 3 conditions to study **back-stepping**.
We present the **first positive result** towards backstepping.

Quantitative results

1. Perceived as the **most useful** debugger feature
2. Almost significant for **expert-rated** success when comparing **stepping and back-stepping** (p=0.0526)

Qualitative results

1. Removes the fear of **“Missing the moment”**
2. Changes debugging strategies to **failure-first navigation and backtracking**
3. The lack of visual feedback makes back-stepping hard to master