

# Graphical Loop Invariant Programming in CS1

Simon Liénardy, Lev Malcev, Benoit Donnet

`simon.lienardy@uliege.be`



# Agenda

- Context
- Programming Methodology
- GLI
- CAFÉ
- Preliminary Evaluation
- Conclusion & Future Work

# Context: First Year Students in CS

- Introduction to programming course
  - University of Liège, Belgium
  - Open access to the University (and Higher Ed. in general)
    - No background required, esp. in Mathematics
- ~ 80 Students in Computer Science (Bloc 1)
- Programming skills required by next courses

# Programming Methodology

INIT

*{Invariant}*

**while** (B) {

*{Invariant  $\wedge$  B}*

LOOP BODY

*{Invariant}*

}

*{Invariant  $\wedge$   $\neg$  B}*

END

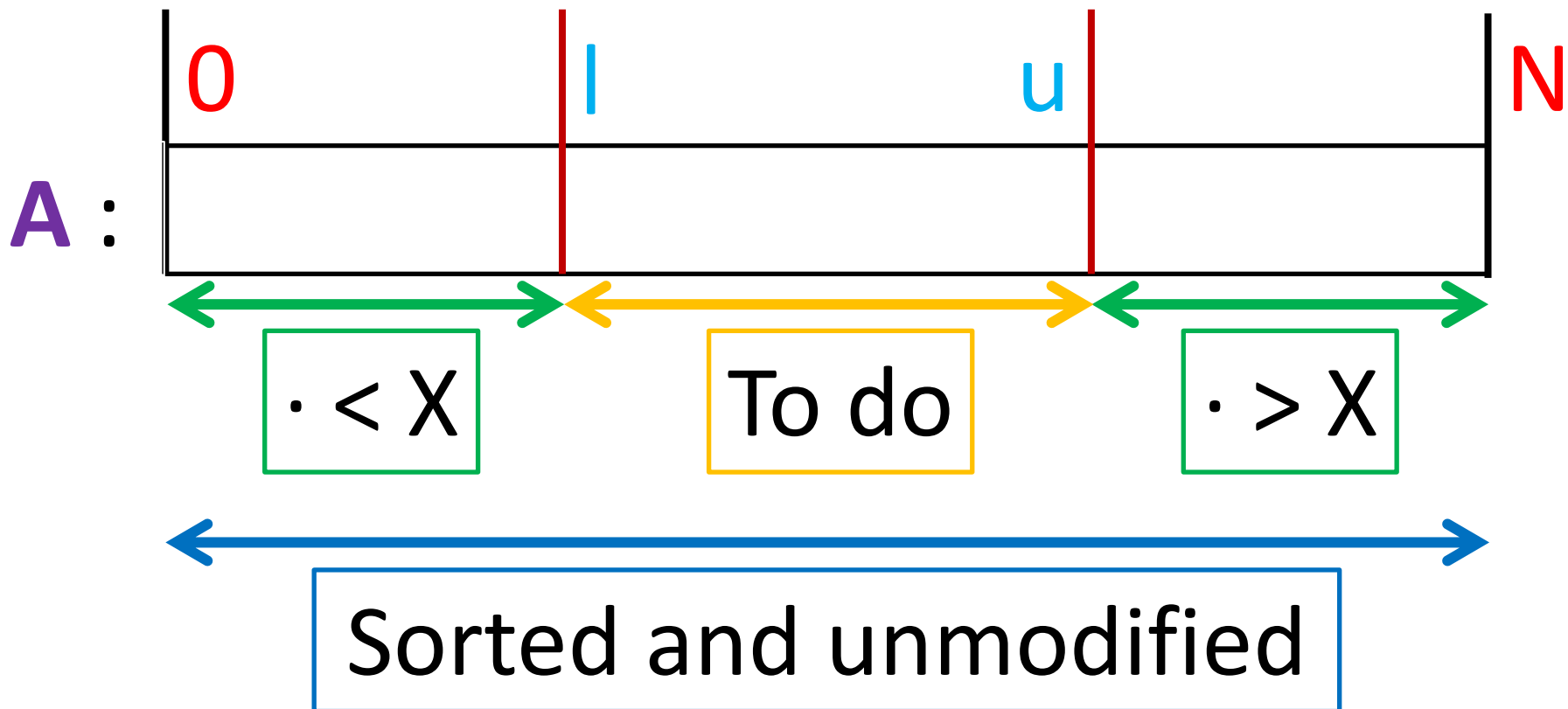
**Deriving** the code:

- Based on Dijkstra, *A Discipline of Programming* (1976)
  - Graphical representation
  - Represent what has already been computed
- $\Rightarrow$  Strategy to solve the problem  
(*Metacognition*)

# Graphical Loop Invariant: example

## Binary search in a sorted Array

Value searched :  $X$



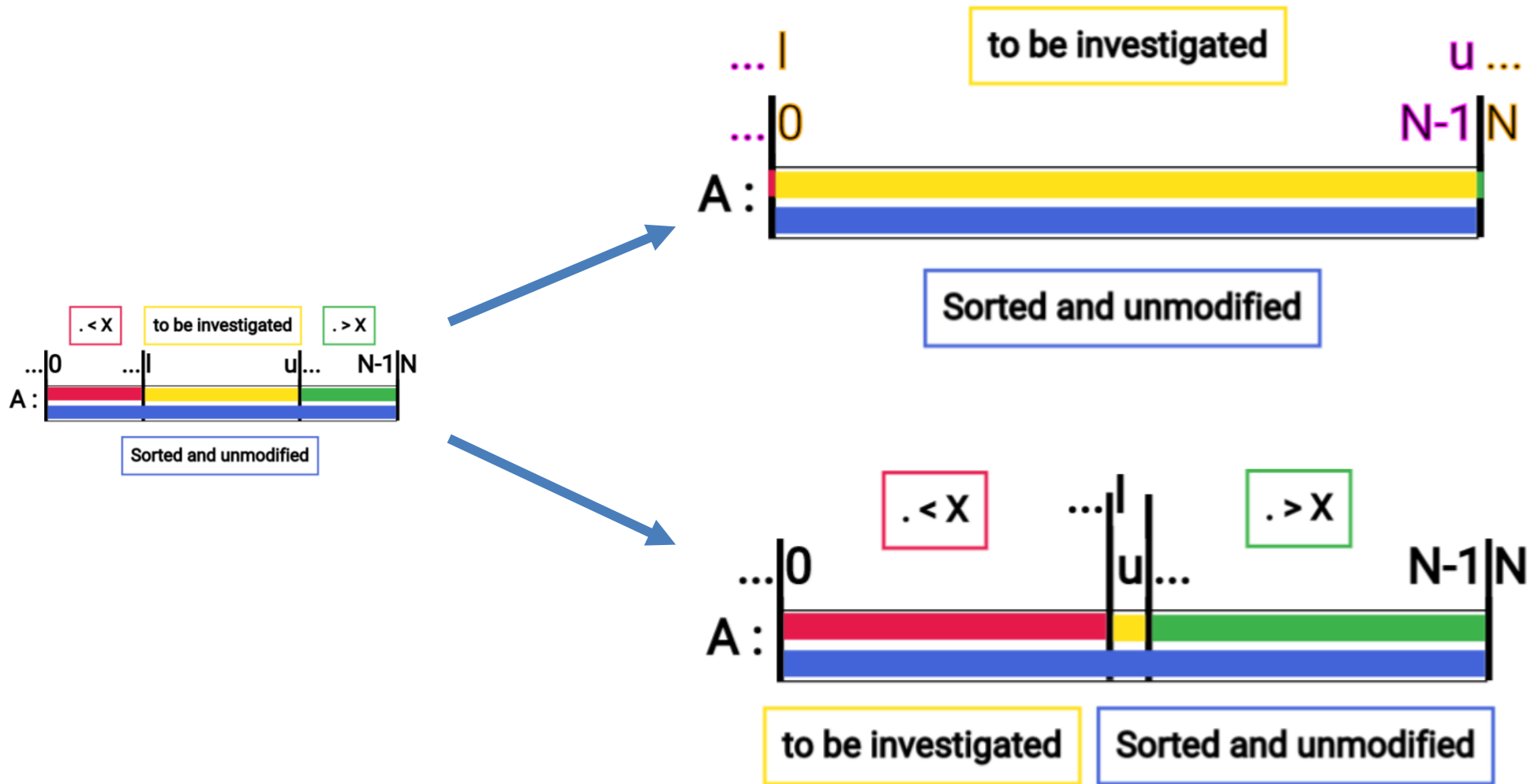
# Graphical Loop Invariant: Guidelines

1. Drawing must be relevant and **named**
2. The **boundaries** of the problem are provided
3. There must be one (or more) **dividing line(s)**
4. Each dividing line should be labeled (w/ **variables**)
5. Label(s) about **what has been achieved so far**
6. Label(s) about **what should be done**
7. All the named elements and variables are present in the code

# Graphical Loop Invariant: Rules

- Rules can be used to assess a Student's Invariant
- Mistakes can be sorted into 3 categories:
  - Syntax
    - E.g. missing elements
  - Semantic
    - E.g. labels that do not make sense or not relevant w/ the problem
  - Matching with the code
    - The Invariant should be used to write the code

# Deriving the code from the Invariant





# Introducing GLI



# GLI: Patterns available

Line :  $0 \quad 1 \quad 2 \quad 3 \quad \dots \quad i-1 \quad i \quad \dots \quad n-1 \quad n \rightarrow$

Text :  $a_0 + a_1x + a_2x^2 + \dots + a_ix^i + \dots + a_nx^n$

Array :  $\dots | 0 \quad \dots | i \quad \dots | N$

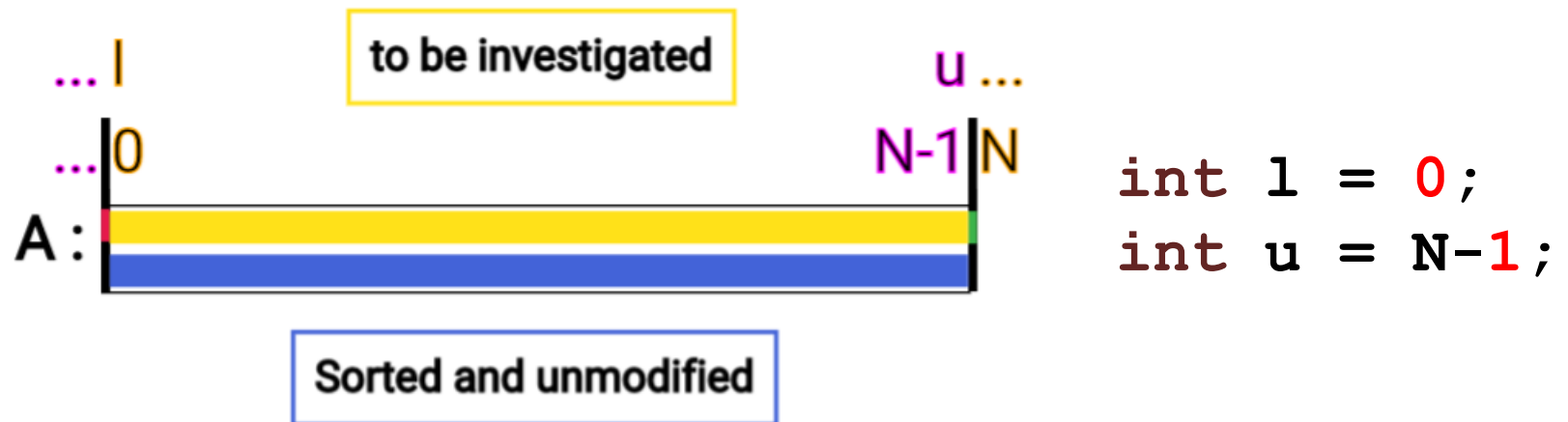
# GLI: Syntax checks

The screenshot displays the GLI (Graphical Loop Invariant) interface. At the top center is the logo, which consists of a colorful geometric shape resembling a stylized 'L' followed by the text "GLI Graphical Loop Invariant". Below the logo is a horizontal toolbar with five buttons: "Line", "Add", "Define a zone", "Validate", and "Delete". The "Validate" button is highlighted with a pink border. Below the toolbar is a diagram of a loop. The diagram shows a horizontal axis with tick marks labeled 0, 1, 2, 3, ..., i-1, i, ..., n-1, n. An orange bar is drawn under the axis from 0 to i. A red square icon with a white dot and three dots below it is positioned at the start of the orange bar. A mouse cursor is hovering over this icon, and a yellow tooltip box contains the text: "Something should be written here, councerning what has been done or is going to be." The "Validate" button in the toolbar above is highlighted, indicating that the user has just performed a validation check that failed.

# GLI: Deriving the code

As soon as the checks are passed:

- One can move the dividing bars to get the init state:



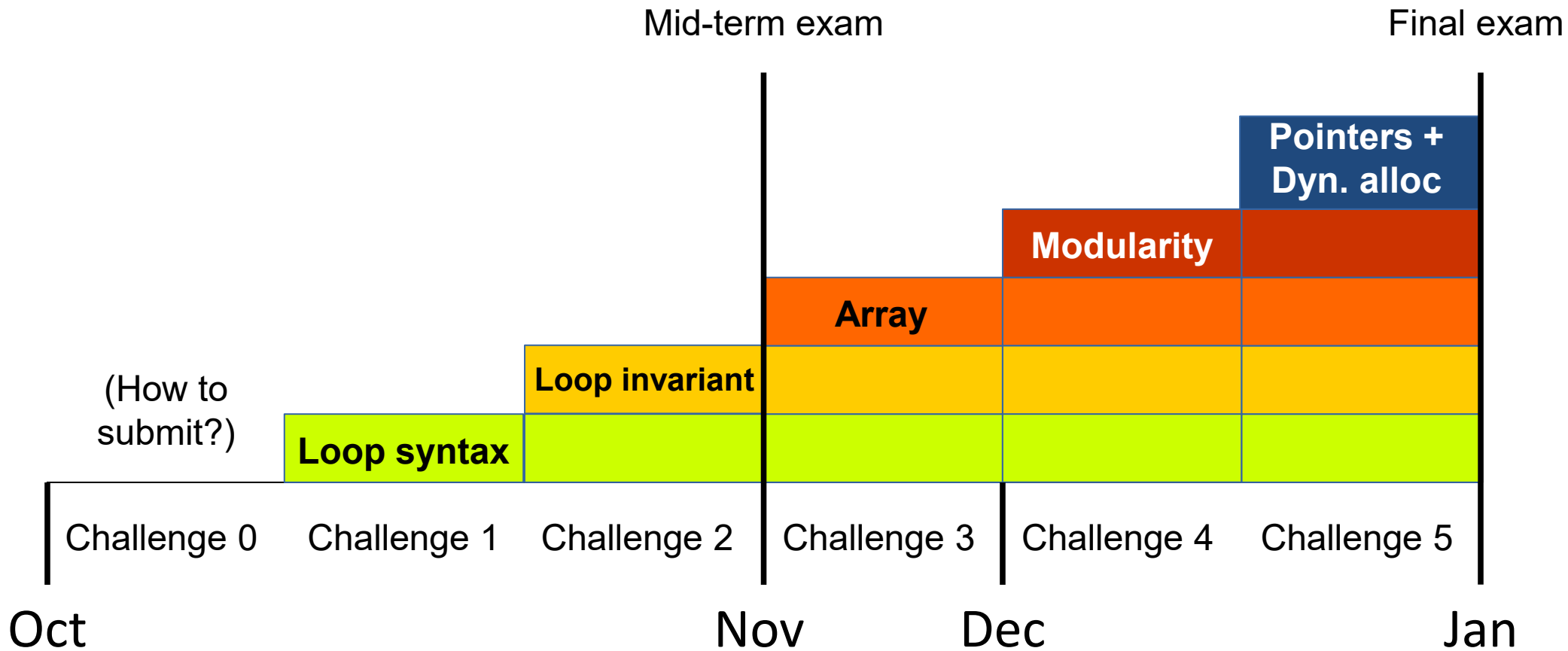
- Or move them to find a condition under which the loop must be stopped  
→ Infer the loop condition

# Introducing CAFÉ

- French acronym for  
Correction Automatique et Feedback des Étudiants
- Students submit on a web platform 5 Challenges of increasing difficulty during the semester
  - Plus Challenge 0 to learn how to submit
- Assessment for Learning oriented [Sambel et al., 2013, Wiliam, 2011]

S. Liénardy, L. Leduc D. Verpoorten and B. Donnet. 2020. Café: Automatic Correction and Feedback of Programming Challenges for a CS1 Course. In *Proc. ACE '20: Twenty-Second Australasian Computing Education Conference*

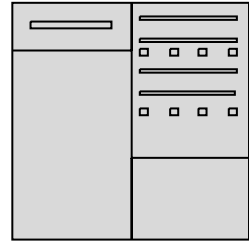
# Challenges?



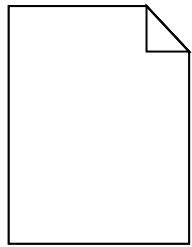
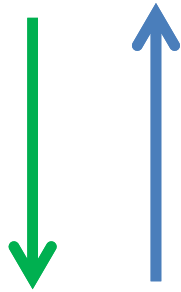
→ Cumulative difficulty

# Introducing CAFÉ

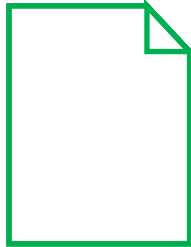
Day 1



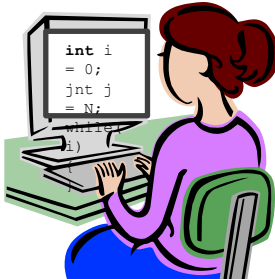
Blackboard



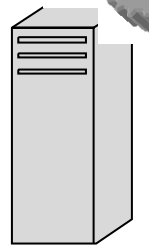
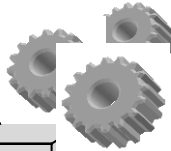
Instructions



Template



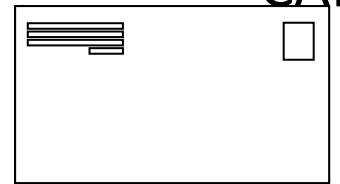
Challenge



CAFÉ

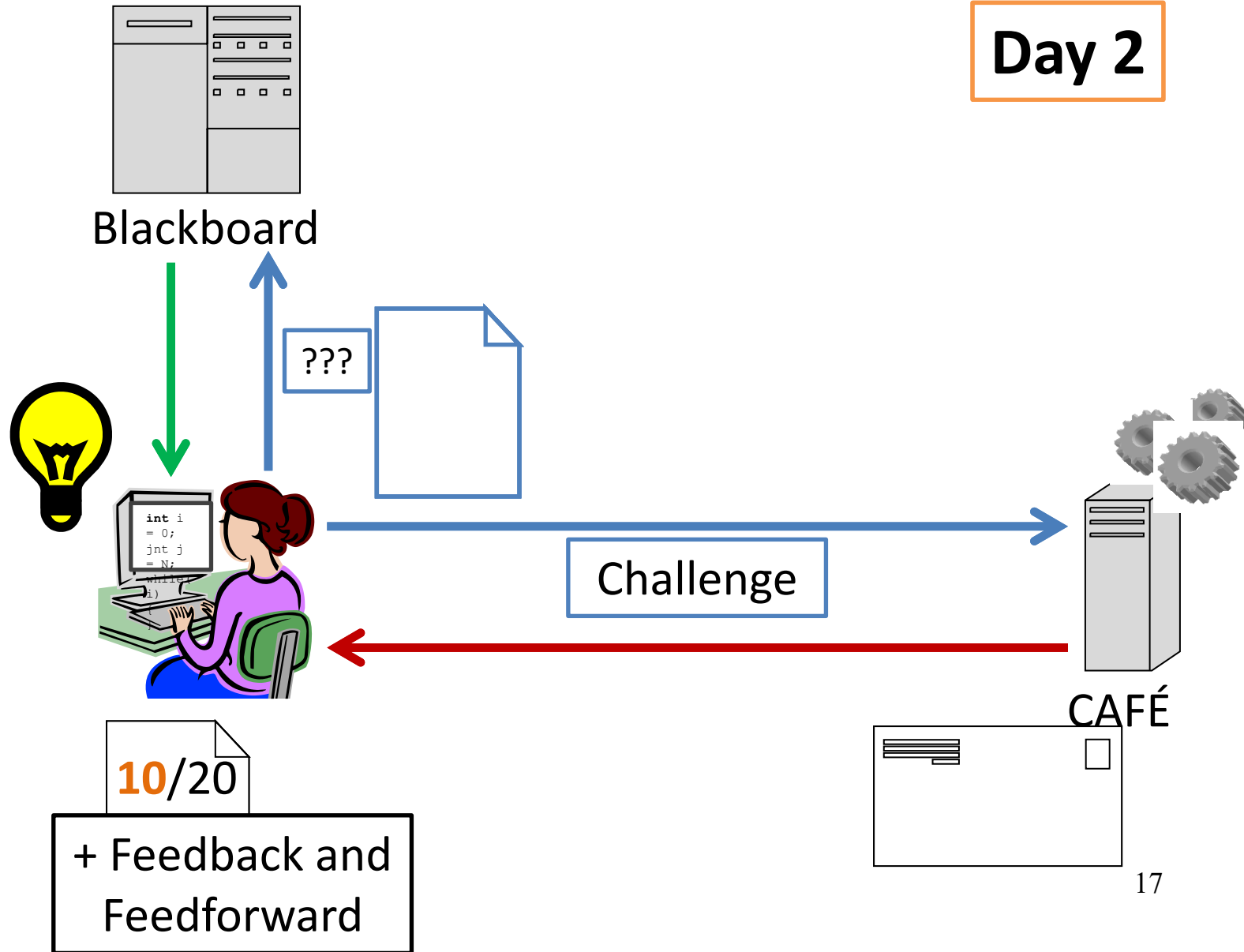


+ Feedback and Feedforward



# Introducing CAFÉ

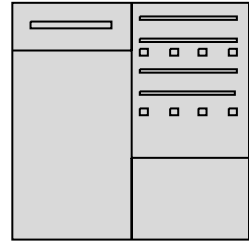
Day 2





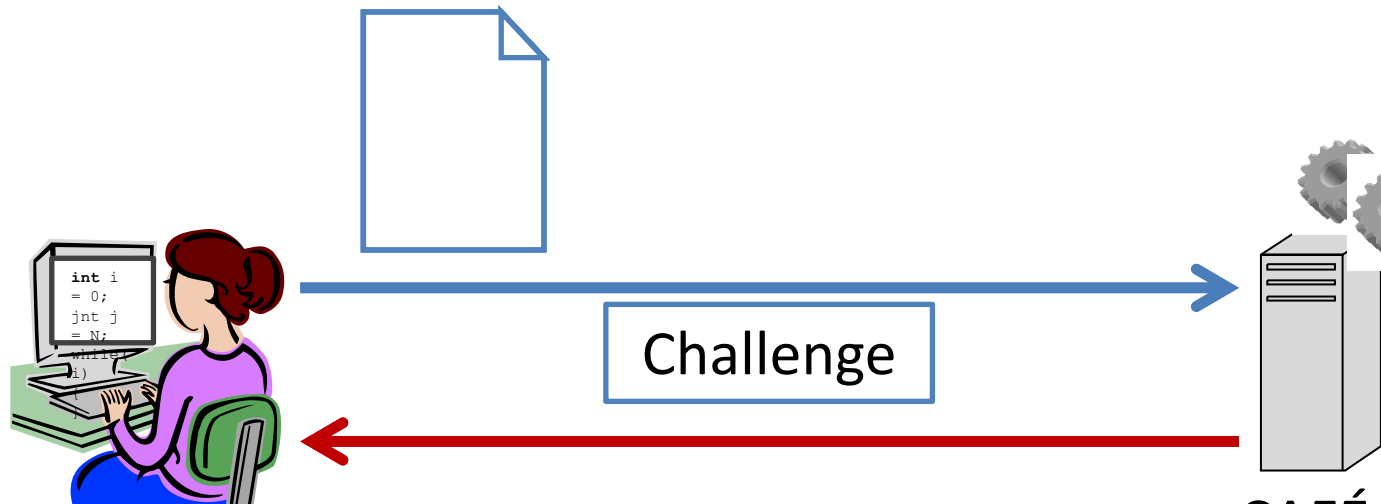
# Introducing CAFÉ

Day 3



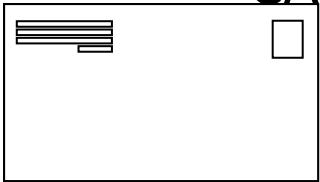
Blackboard

→ Closing the feedback loop [Boud, 2000]



19/20

+ Feedback and Feedforward

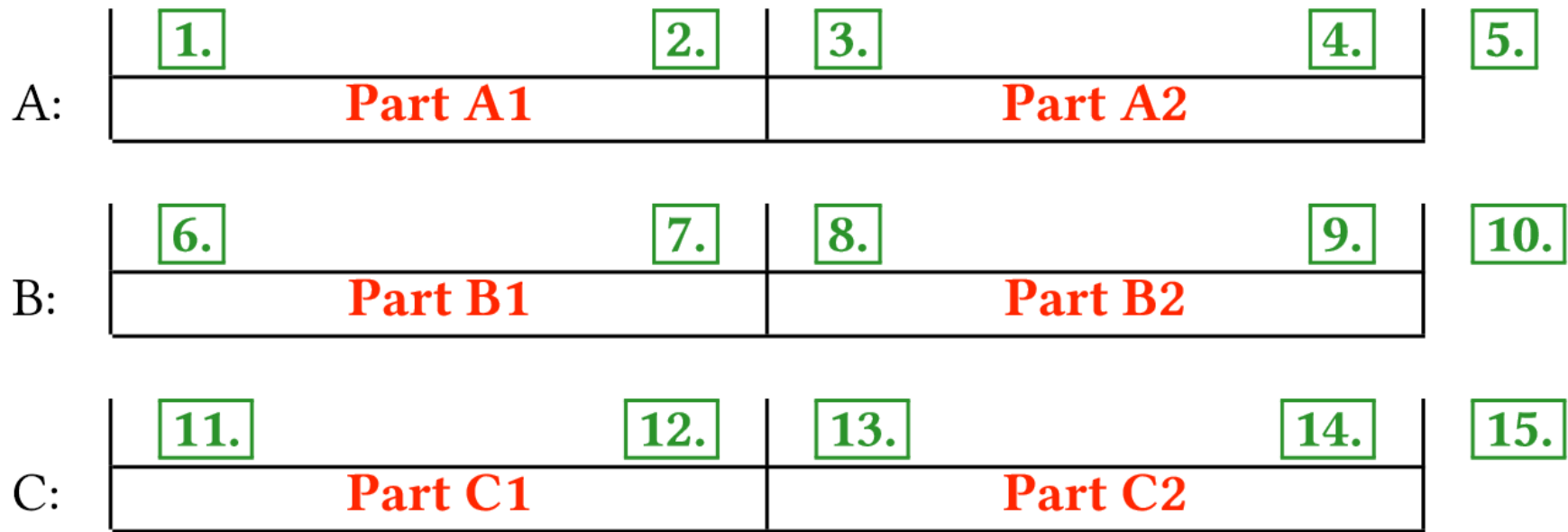


# CAFÉ: Submitting Invariant

- Graphical Loop Invariant is the **corner stone** of our methodology
- How to make students work with the Invariant during the Challenges?
- Blank Invariant
  - To be filled by the student
  - Bootstrap effect

# CAFÉ: Submitting Invariant

(Challenge : compute  $C = A \cap B$ )



And the values **16.** to the **17.** and to the **18.** are in the **19.**

- 1.** -> **15.** : Replace by variables, constants, numerical values
- 16.** : Replace by “different from”, “common to”, etc.
- 17.** -> **19.** : Replace by **Part A1**, **Part A2**, **Part B1**, ...

# CAFÉ: Invariant checking

- Variables present in the Inv are in the code too
  - And initialized according to the Invariant
- Array Indices in the Inv are used to index arrays
  - Out of Bound check
- Loop Variant correction
- Iterations count (if complexity constrains)
- Unit tests
- Feedback & Feedforward added after correction

# CAFÉ: Preliminary Evaluation

- Data about :
  - Students participation
  - Performance
  - Perception
- Over multiple years
- Work in progress

# Conclusion

- GLI:
  - Evaluation = work in progress
- CAFÉ:
  - Takes time but efficient and scalable
  - Language independent (modulo slight mods)
  - Lot of data to be analyzed
- Questions ?