Identifier Splitting for SBSE Software Maintenance

Kevin Graham

UNIVERSITY OF STIRLING
DAASE PROJECT

Supervisors: Prof. Edmund Burke
Dr. Jerry Swan

SBSE: Search-based Software Engineering

- **Formulating common software engineering problems in terms of a search-based optimisation problem**
  - Many competing and complex objectives
  - Large search spaces

- **Search based software engineering research has gained much interest in recent years** [10]
  - Improvements to software engineering processes
  - Quality of code produced

Software Maintenance

- **60/60 rule exists in software development**
  - Maintenance costs dominate development costs at around 60% [8]
  - Further, 60% of maintenance costs consumed by enhancement alone [8]

- **Steps therefore must be taken to reduce maintenance costs over the lifetime of a project**

Identifiers

- **Identifier names in source code are natural language tokens used to label program entities**
  - Including; variables/constants, methods/functions, classes and objects in OOP etc.

- **Notationally, they can be described as a character sequence** '$C\{c_0, \ldots, c_n\}$' [11]
  - *Where a character c_i can be a letter, digit or special character*

Identifiers

- **The natural language of identifier names captures**:
  - Domain-specific knowledge
  - Clues to programming intention

- **Much research exists exploring the role identifiers and the language used for the human comprehension of source code**
  - E.g. concise and consistent naming [4]

Identifiers: Composition

- **Programming languages and programming conventions place constraints on form and content of identifier names** [1]
  - Programming languages impose hard constraints
    - *You can be sure constraints are followed*
  - Programming conventions impose soft constraints
    - *Not applied universally*

- **Fully Autonomous techniques should be able to handle both conventionally and unconventionally constructed identifiers** [1]

Identifier Splitting Problem

- **Defined as the identification of the component words in identifiers, where:**
  - No explicit boundary
  - Present, but open to misinterpretation
- **Involves the solution to two sub-problems**
  - Mixed-case tokenisation [5,11]
  - Same-case tokenisation [5,11]
- **Often extended to include abbreviation expansion**
  - **E.g. HTTP = {hyper, text, transfer, protocol}**

Mixed-case

- **Involves the tokenisation of identifiers containing internal capitalisation boundaries** [1,5]
  - Lower to upper case (LCUC) e.g. getIdentifier
  - Upper to lower case (UCLC) e.g. XMLParser
- **LCUC transitions readily tokenised** [1, 5]
  - Well defined boundary between words i.e. easily identified
  - Conventional
- **UCLC has two possible boundaries** [1, 5]
  - GPSstate → {GP, Sstate} according to camel-casing
  - GPSstate → {GPS, state} according to alternative split

Mixed-case: UCLC

- **Has been addressed (somewhat) using dictionary-based scoring metrics**
  - Samurai Identifier Splitter [enslen]
  - Identifier Name Tokeniser Tool (INTT) [1]

- **But what about…**
  - Unconventional acronyms e.g. ‘connectDnDServer’
  - Acronyms/words with bounding digits e.g. ‘*POP3Server*’
    - *Pop3Server* doesn’t solve this problem
    - What about ‘*Cad3DModel*’?
  - Ambiguous boundaries e.g. ‘*GPRSend*’

Same-case

- **Sub-problem addresses the tokenisation of same-case multi word identifiers** [1, 5]
  - E.g. MAXVALUE or thenewestone

- **Existing techniques mainly rely on words being found in a dictionary**
  - Sometimes more than one dictionary e.g. abbreviations
  - But no dictionary can ever be complete
    - Invented words for some new concept
    - Neologisms e.g. ‘devoidify’, ‘detokenated’, ‘pathinate’, ‘precisify’ [1]

Motivation

- **Tools used to aid program comprehension**
  - Particularly during maintenance tasks
- **Many existing analysis tools don't leverage natural language**
  - Arguably, domain knowledge and programming intention is more informative to a programmer
- **Tools have emerged that exploit this natural language information** [12]
  - Natural Language Program Analysis (NLPA)

Motivation

- **To process natural language, accurate tokenisation is a necessary first step**
  - Words have individual meanings and probably have a different or no meaning while un-tokenised

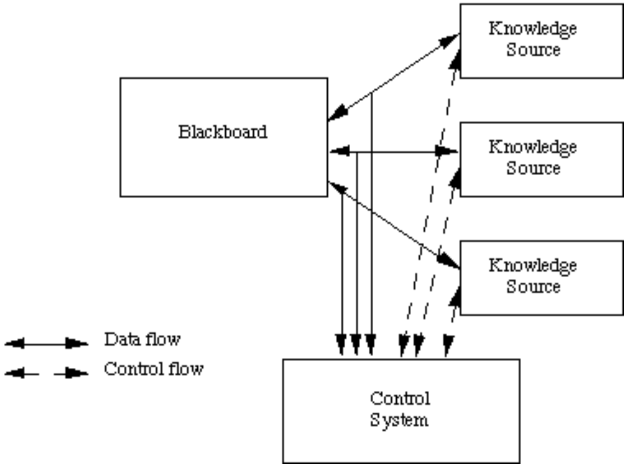
- **Incorrect tokenisation at this stage can propagate errors to later processes during program analysis** [1]

Tokenisation for Humans

- **For us the notion of splitting identifiers seems easy:**
  - E.g. tokenise: '*yankeedoodlewenttolondonridingonapony*'
- **We have the advantage of higher-level strategies on our side**
  - We have many a mental dictionary, an idea of probable word frequencies, n-gram frequencies, letters never seen together, context checking etc. etc.
  - We can apply all of these or only some of these to give evidence for a split point
  - We can even invent/analogise new methods from old ones
  - We can adapt to new information and change our conclusions and approach accordingly
- **For identifier splitting, we need our approaches to do the same**

Proposed Solution: Blackboard Architecture

- **A multi-agent problem solving architecture**
  - Opportunistic Problem Solving Model
- **First developed as a means to recognise vocal speech**
  - Hearsay II [6]
- **Involves 3 main components:**
  - Agents/Knowledge Sources
  - Blackboard/Global Workspace
  - Control Mechanism



Blackboard Architecture: Metaphor

The first week of the brainstorming session went slowly.

Courtesy of www.gograph.com

Blackboard Architecture: Characteristics

- Modularity
- Diversity of Problem Solving Techniques
- Flexible Representation of Information
- Common Interaction Language
- Event-driven Triggers
- Need for a Control Mechanism

Blackboard Architecture: Benefits to Tokenisation

- **Components can be applied at the most opportune time**
  - Some operations best performed before others
    - E.g. for ‘connectDnDServer’
- **Heterogeneous components and representations can easily collaborate in a blackboard system**
  - E.g. Neural net splitter [7] and graph-based tokenisation & abbreviation expansion [9, 3]
- **Supports concurrency of problem solving processes**
  - On multiple levels of speciality and granularity

End of the Line

References

- [1]
  - Butler, S., Wermelinger, M., Yu, Y., & Sharp, H. (2011). Improving the tokenisation of identifier names. In *ECOOP 2011–Object-Oriented Programming* (pp. 130-154). Springer Berlin Heidelberg.
- [2]
  - Corkill, D. (1991). Blackboard systems. *AI Expert*, *6*(September), 40–47. Retrieved from http://cs.uni.edu/~wallingf/teaching/162/readings/blackboard-systems.pdf
- [3]
  - Corazza, A., Di Martino, S., & Maggio, V. (2012). LINSEN: An Efficient Approach to Split Identifiers and Expand Abbreviations. *2012 28th Ieee International Conference on Software Maintenance*, 233–242. doi:10.1109/ICSM.2012.6405277
- [4]
  - Deissenboeck, F., & Pizka, M. (2006). Concise and consistent naming. *Software Quality Journal*, *14*(3), 261-282.
- [5]
  - Enslen, E., Hill, E., Pollock, L., & Vijay-Shanker, K. (2009). Mining source code to automatically split identifiers for software analysis. *2009 6th IEEE International Working Conference on Mining Software Repositories*, 71–80. doi:10.1109/MSR.2009.5069482

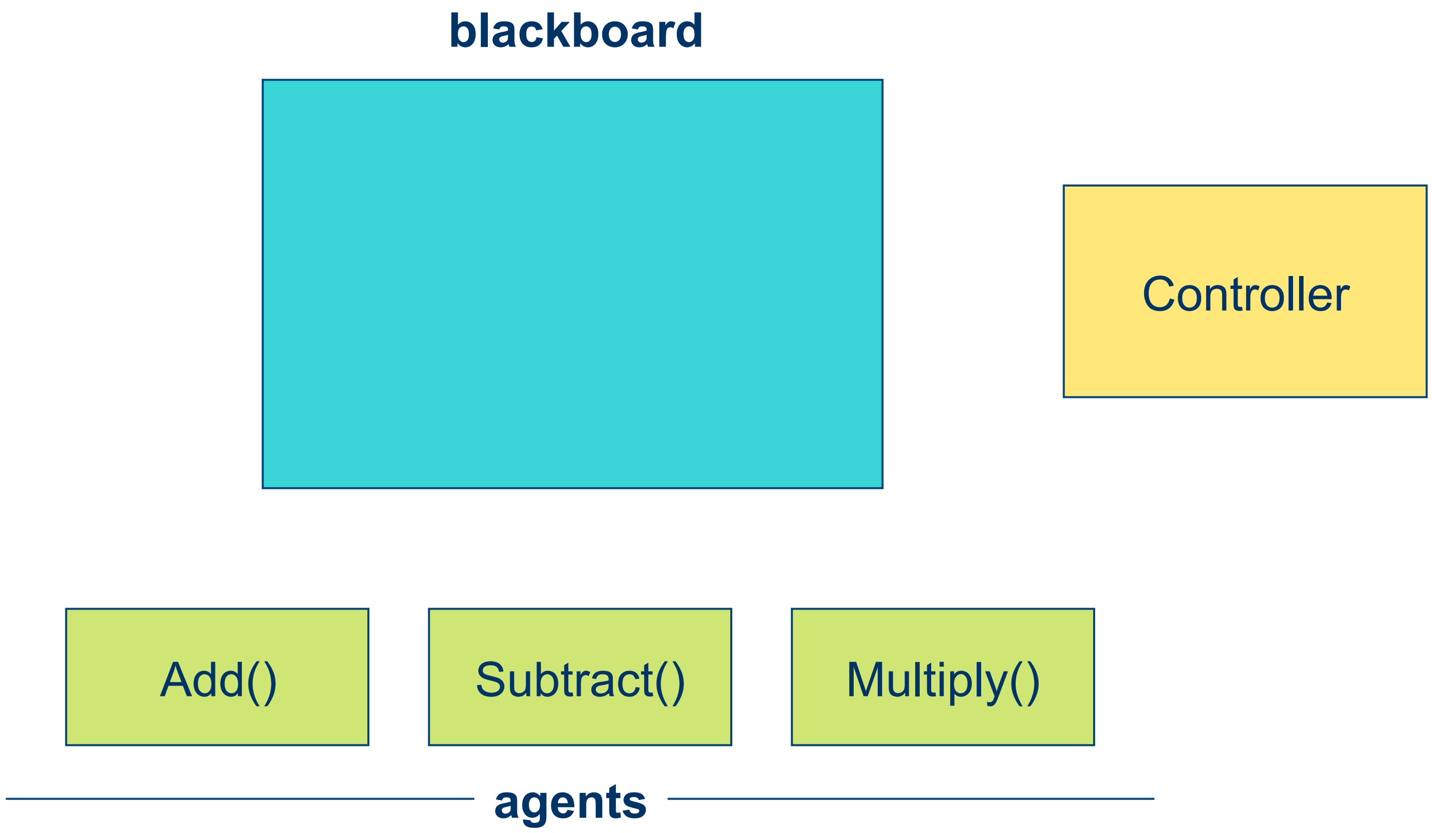
References 2

- [6]
  - Erman, L. D., Hayes-Roth, F., Lesser, V. R., & Reddy, D. R. (1980). The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys (CSUR)*, *12*(2), 213-253.
- [7]
  - Feild, H., Binkley, D., & Lawrie, D. (2006). An empirical comparison of techniques for extracting concept abbreviations from identifiers. In *Proceedings of the 10th Iasted International Conference on Software Engineering and Applications*.
- [8]
  - Glass, R. L. (2001). Frequently forgotten fundamental facts about software engineering. *IEEE software*, *18*(3), 112-112
- [9]
  - Guerrouj, L., Galinier, P., Gueheneuc, Y.-G., Antoniol, G., & Di Penta, M. (2012). TRIS: A Fast and Accurate Identifiers Splitting and Expansion Algorithm. *2012 19th Working Conference on Reverse Engineering*, 103–112. doi:10.1109/WCRE.2012.20
- [10]
  - Harman, M., Mansouri, S. A., & Zhang, Y. (2009). Search based software engineering: A comprehensive analysis and review of trends techniques and applications. *Department of Computer Science, King's College London, Tech. Rep. TR-09-03*.

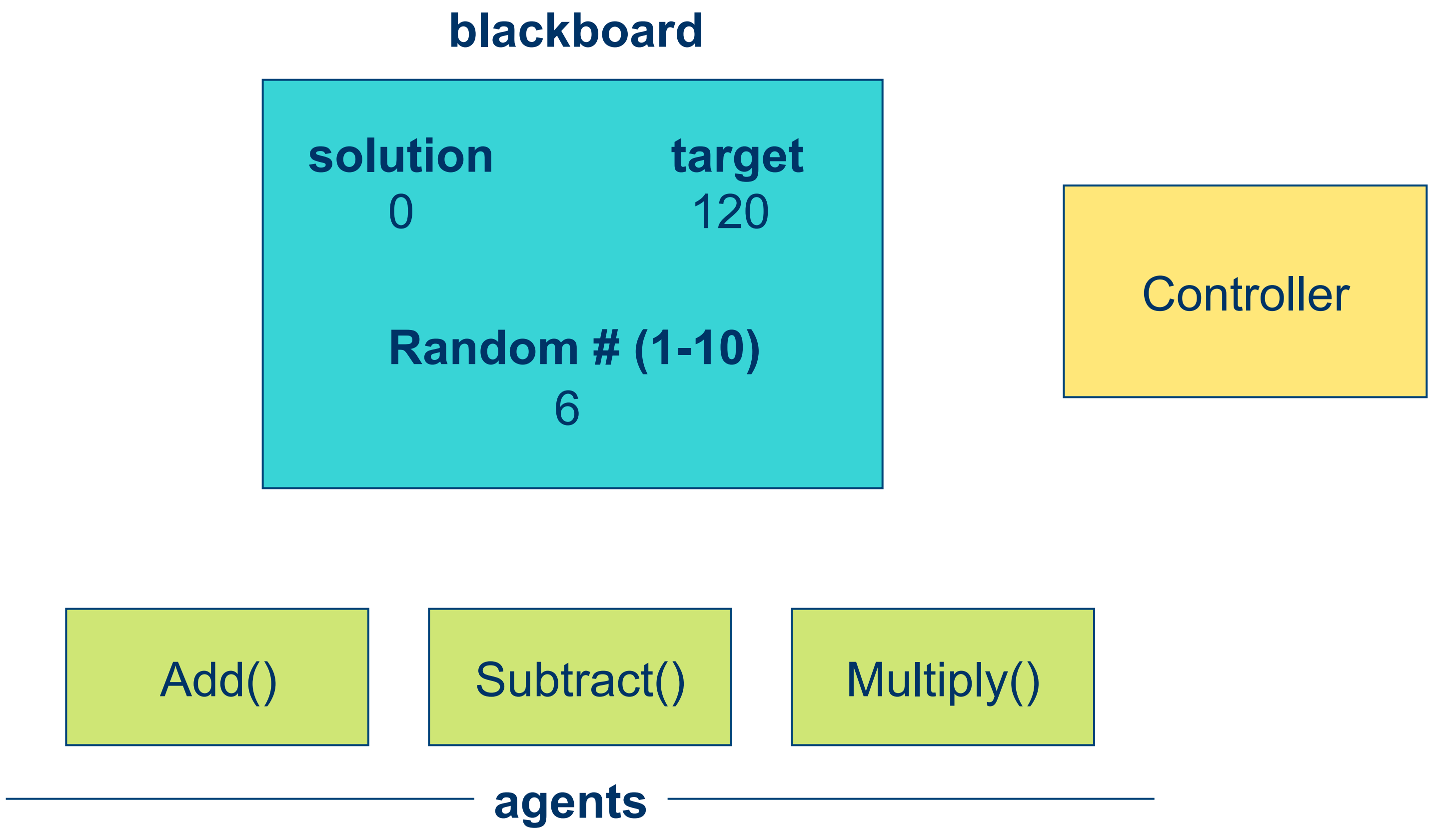
References 2 2/5

- [11]
  - Hill, E., Binkley, D., Lawrie, D., Pollock, L., & Vijay-Shanker, K. (2013). An empirical study of identifier splitting techniques. *Empirical Software Engineering*. doi:10.1007/s10664-013-9261-0
- [12]
  - Pollock, L., K., V.-S., Hill, E., Sridhara, G., & Shepherd, D. (2012). Natural Language-based Software Analyses and Tools for Software Maintenance. In *LNCS* (pp. 102–134). doi:10.1007/978-3-642-36054-1_4

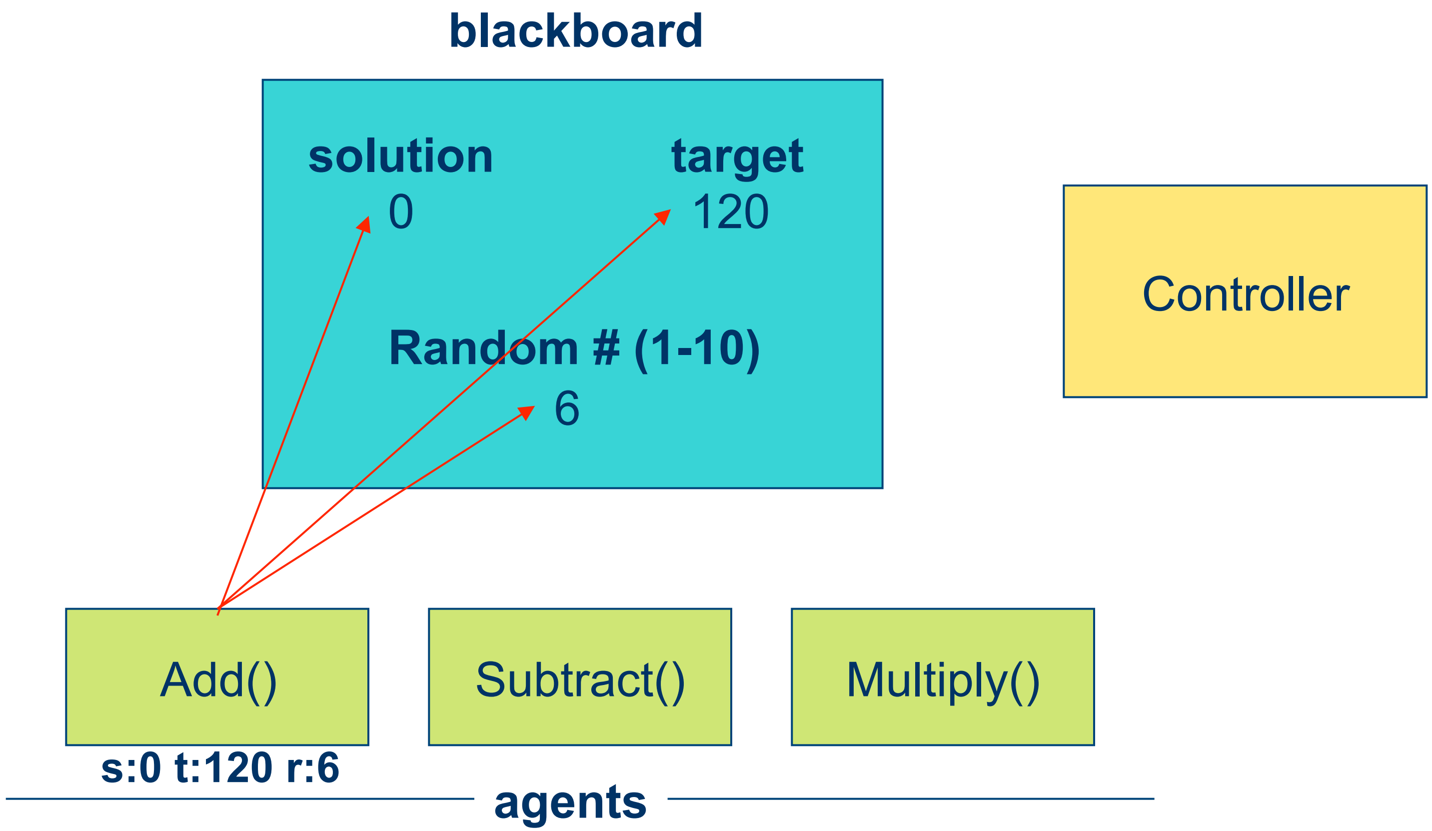
Example



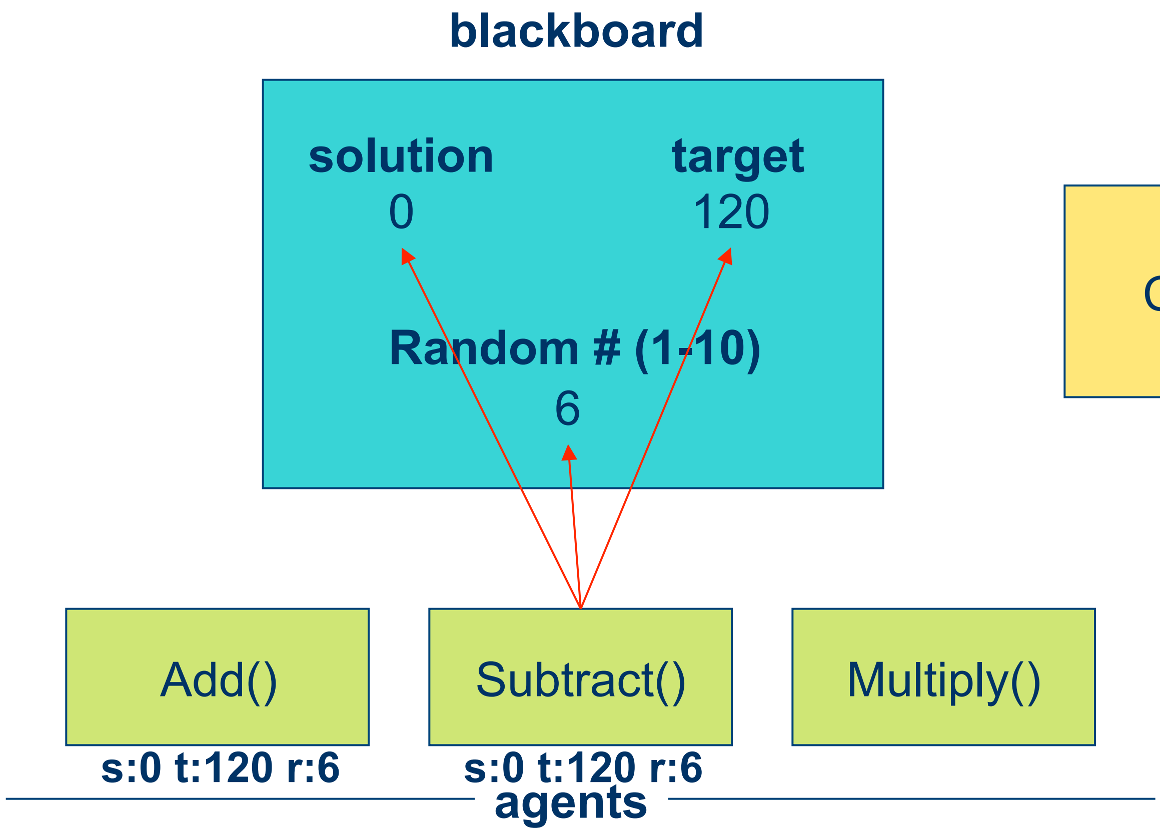
Example: Iteration 1



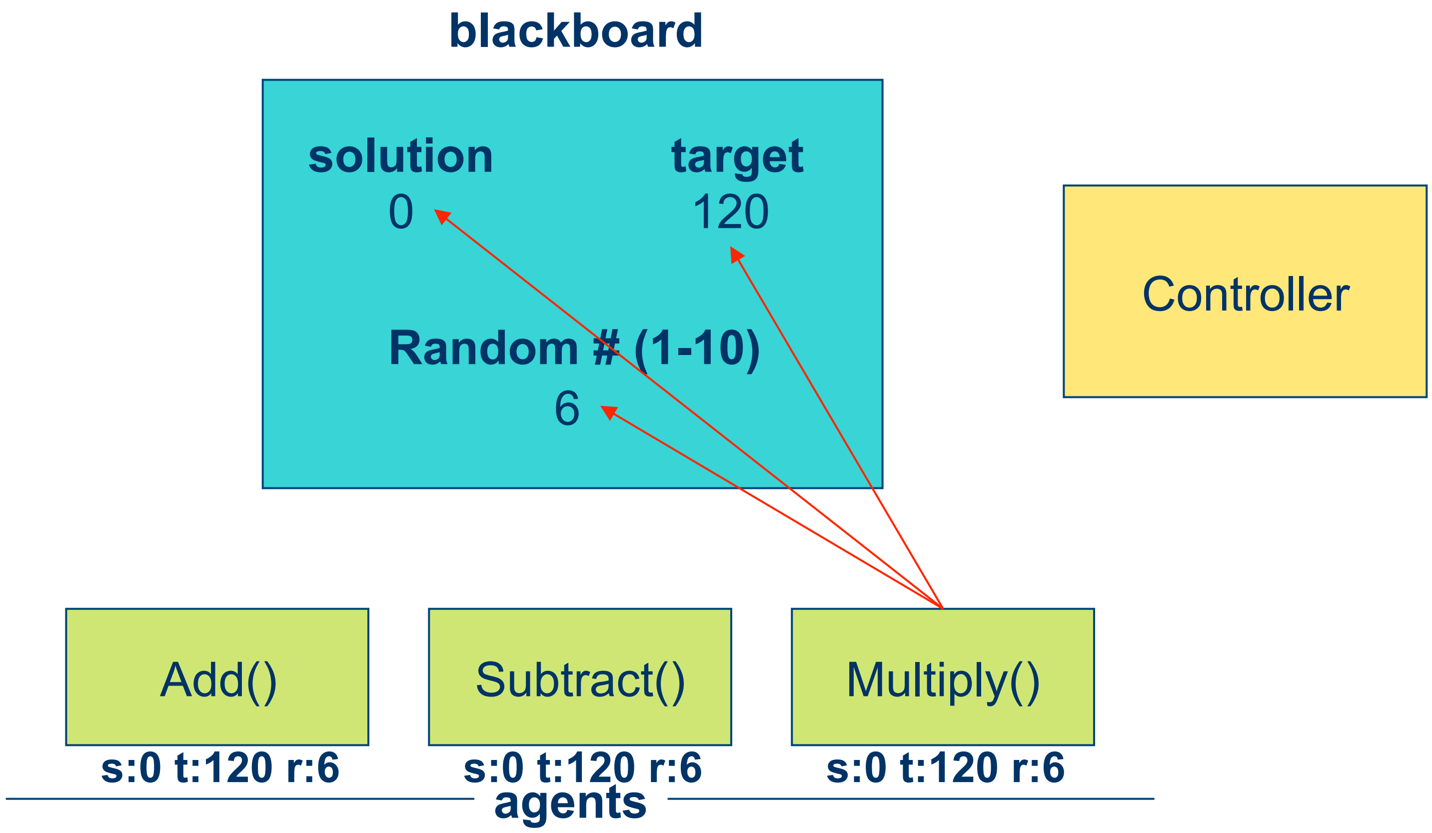
Example: Iteration 1



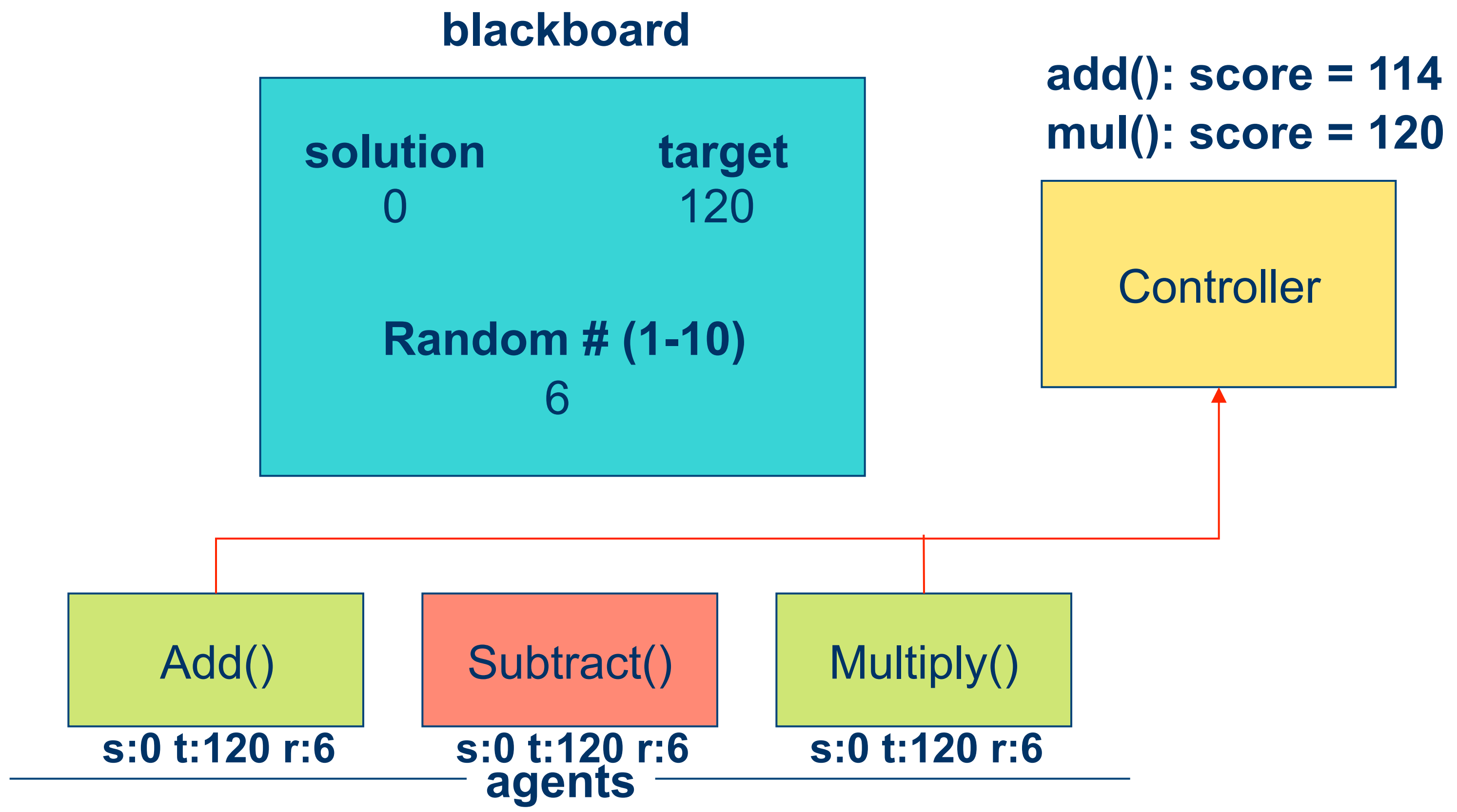
Example: Iteration 1



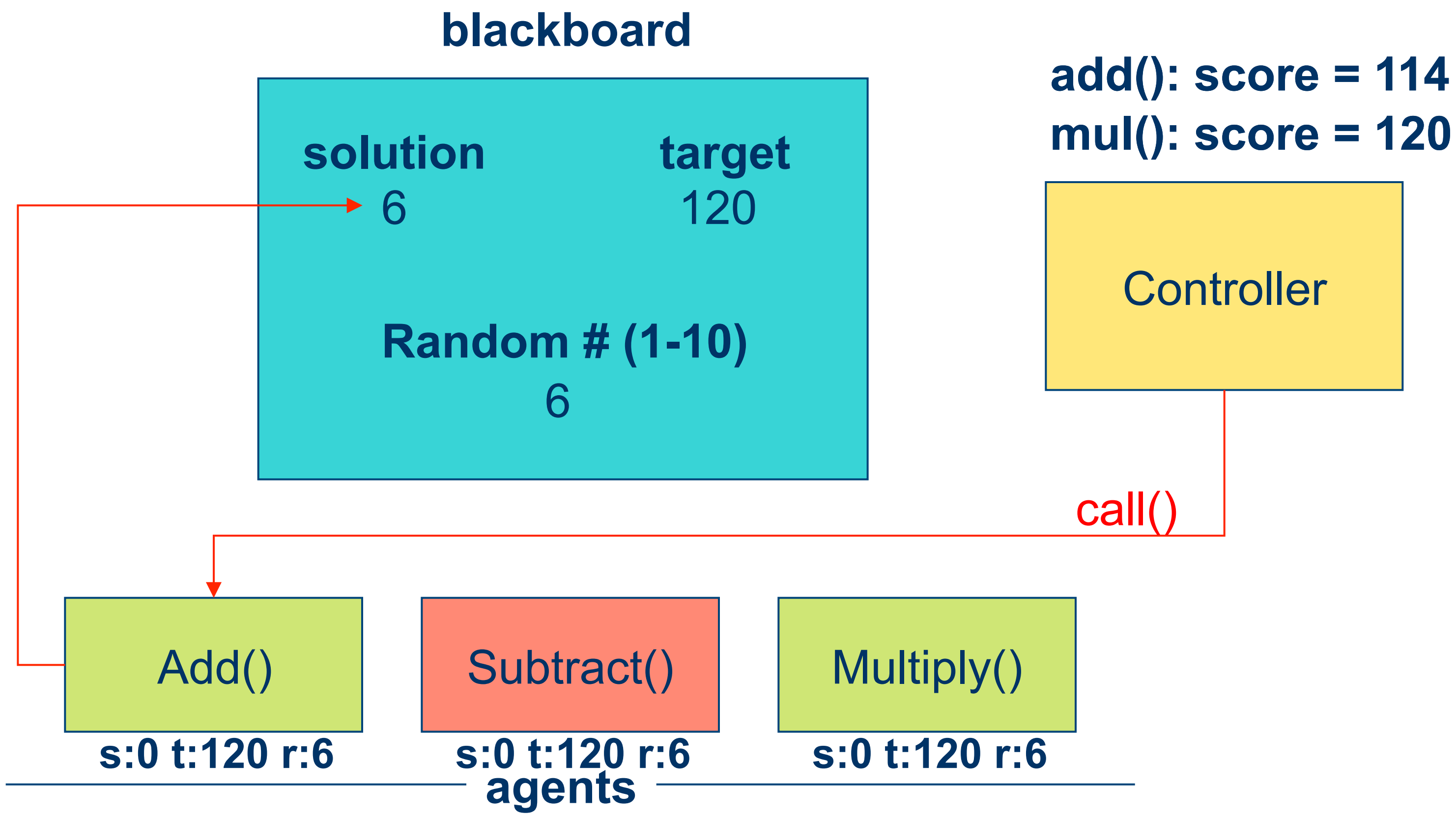
Example: Iteration 1



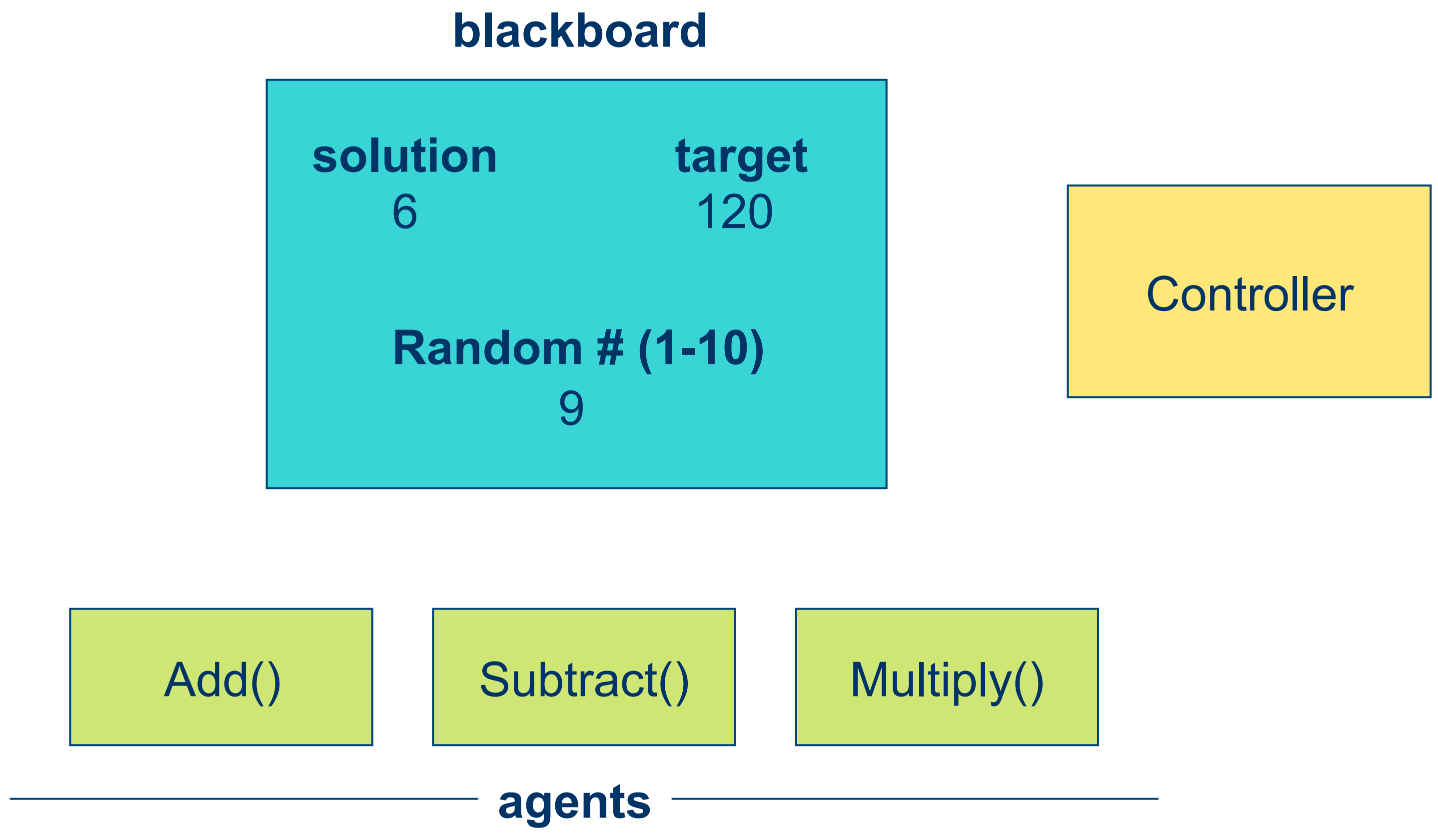
Example: Iteration 1



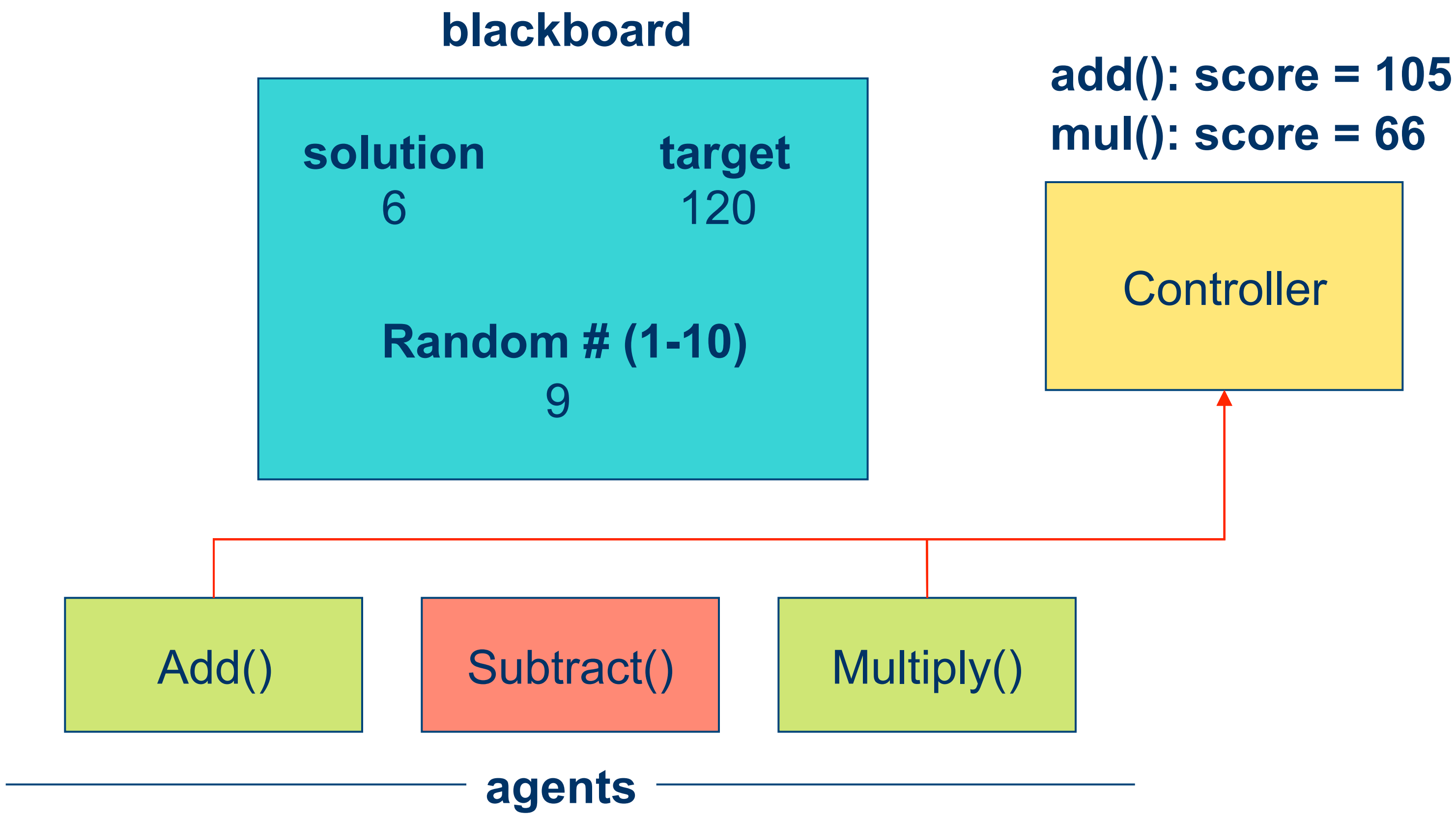
Example: Iteration 1



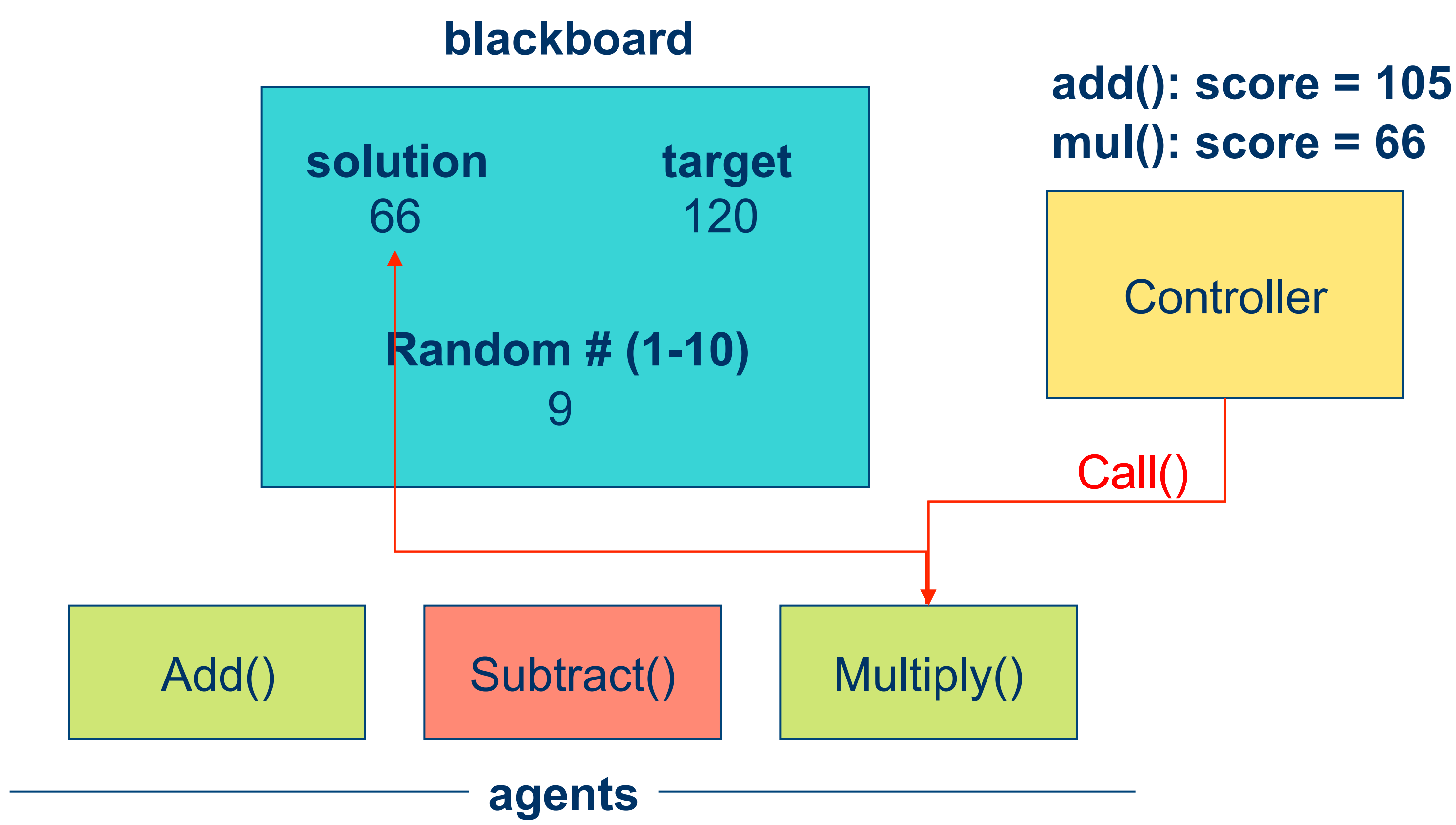
Example: Iteration 2



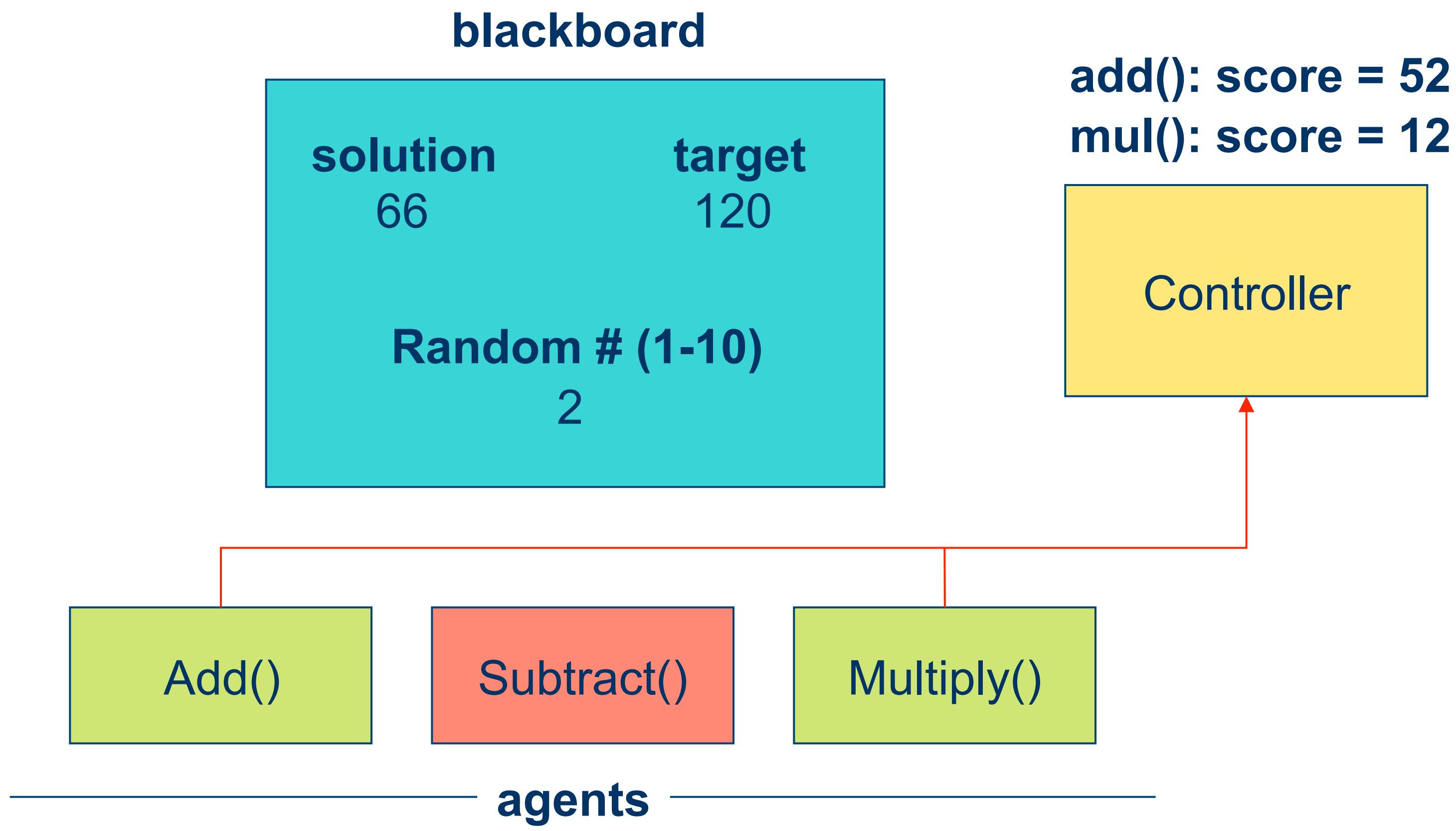
Example: Iteration 2



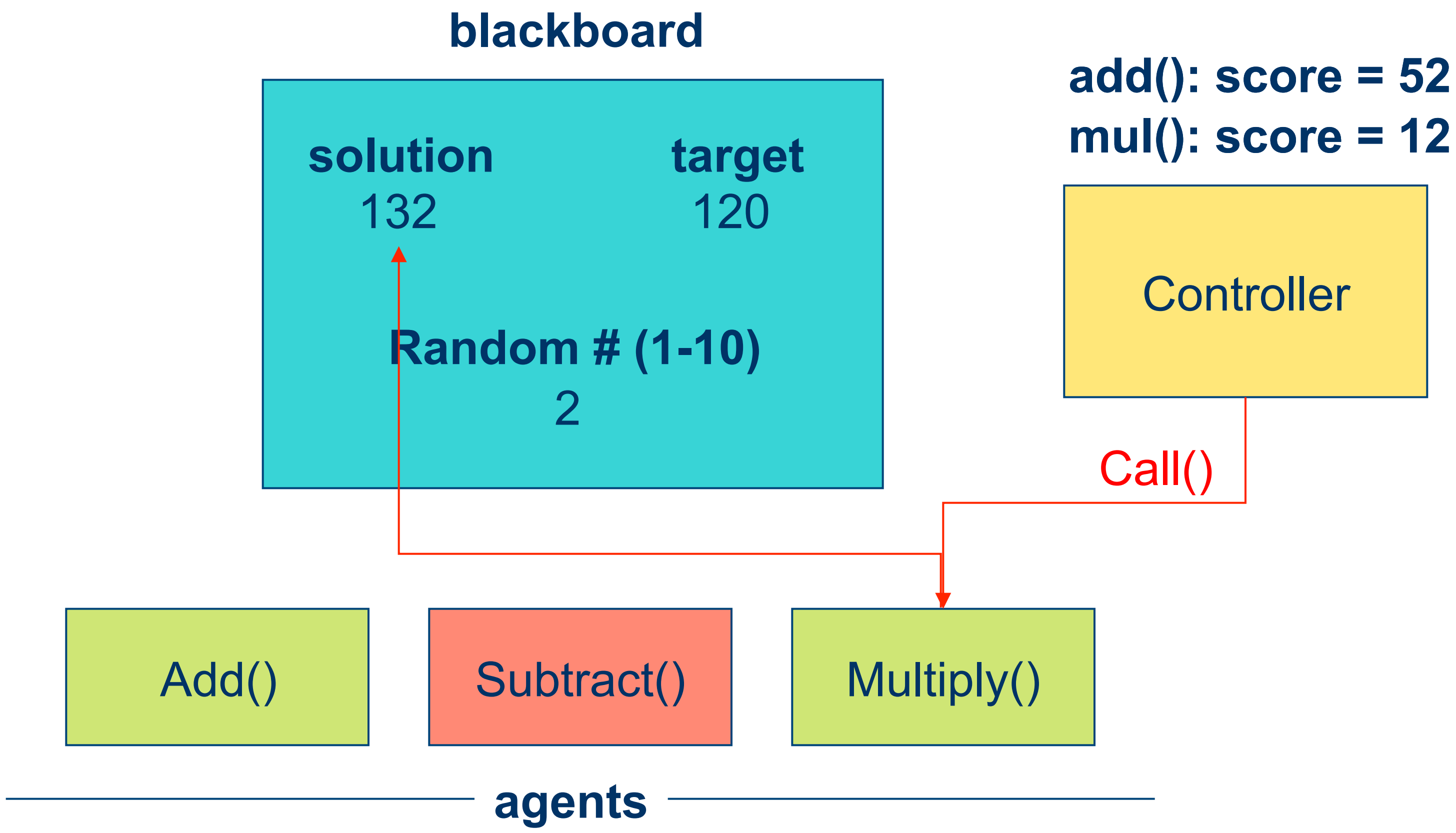
Example: Iteration 2



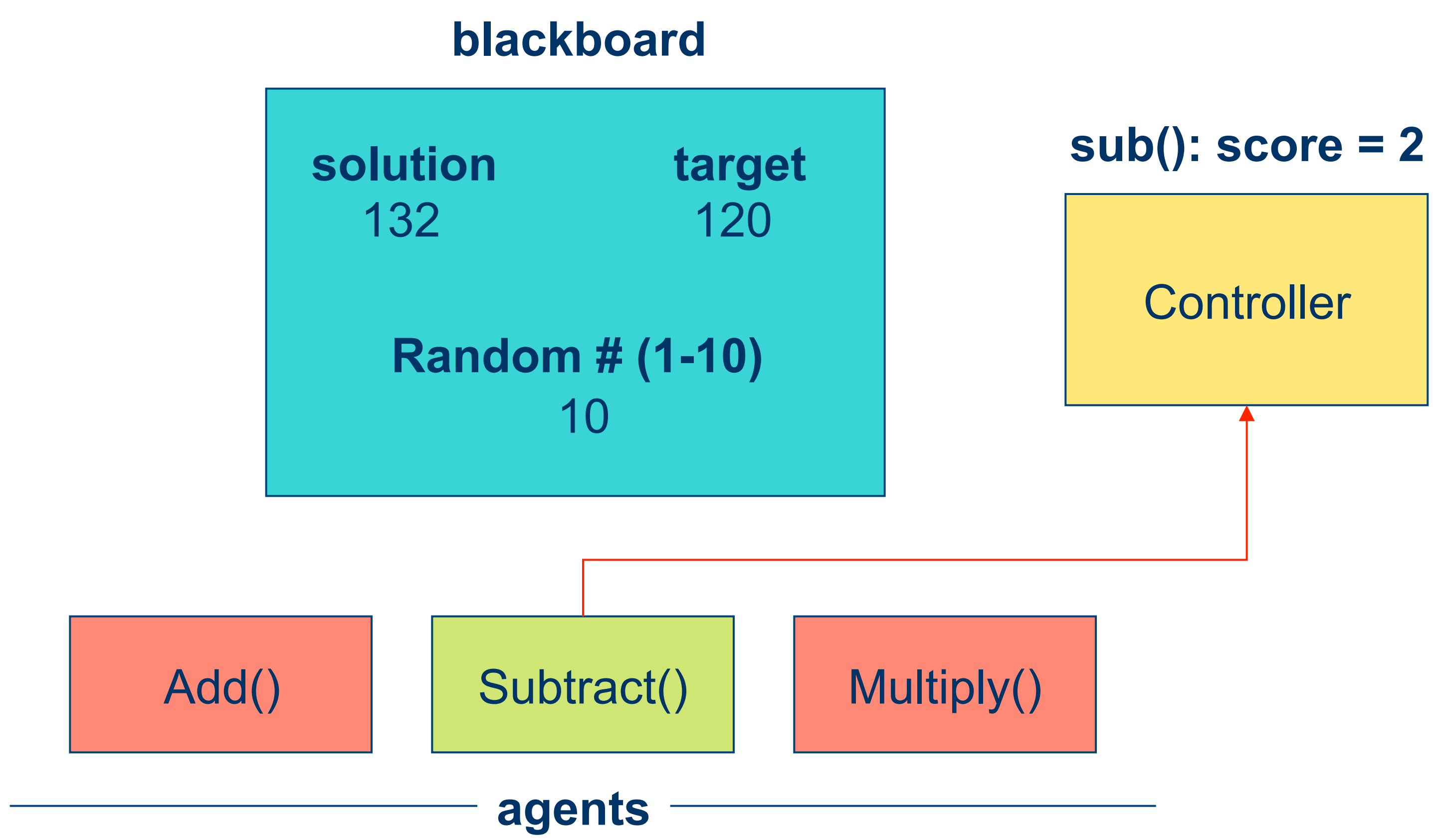
Example: Iteration 3



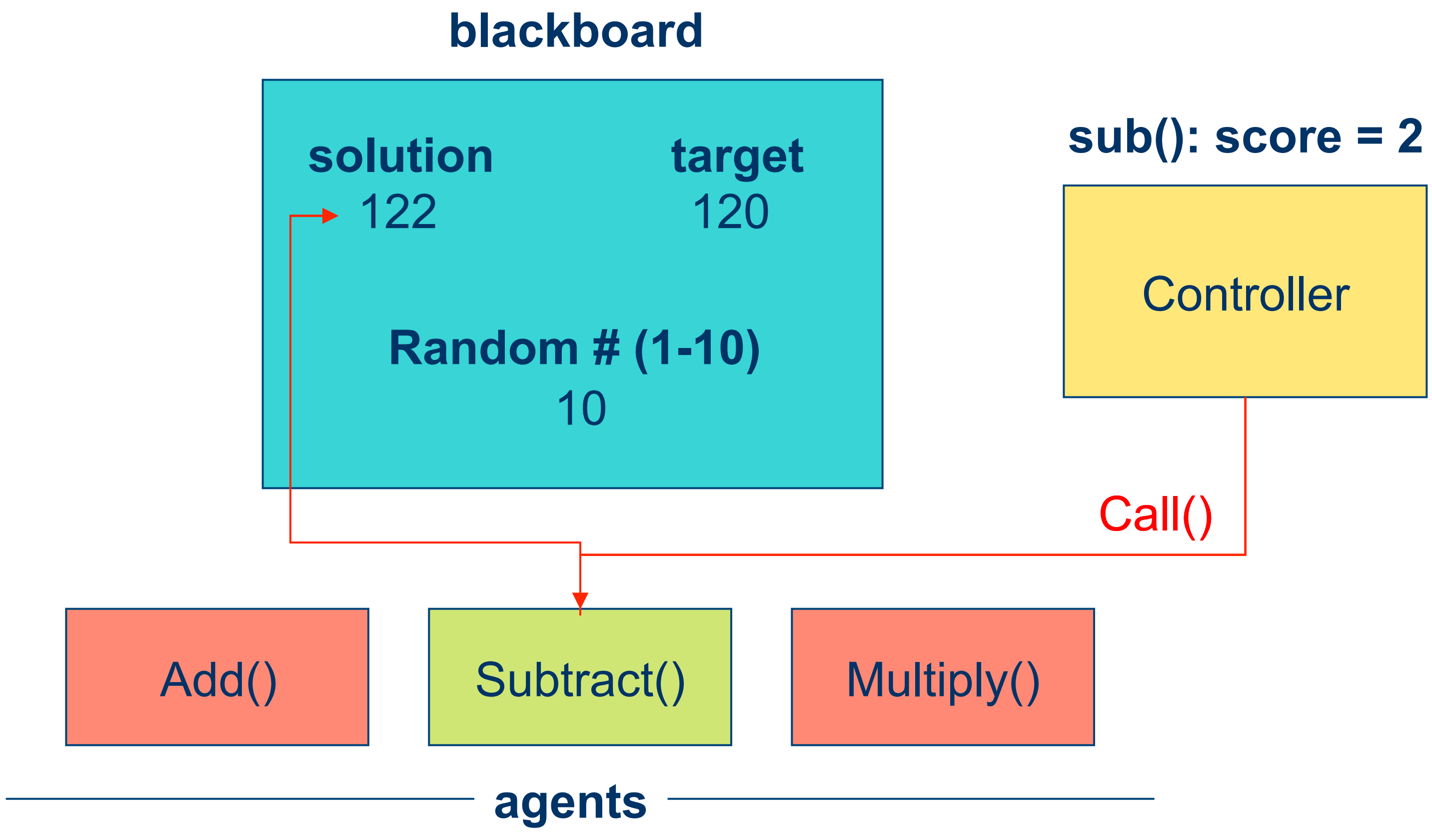
Example: Iteration 3



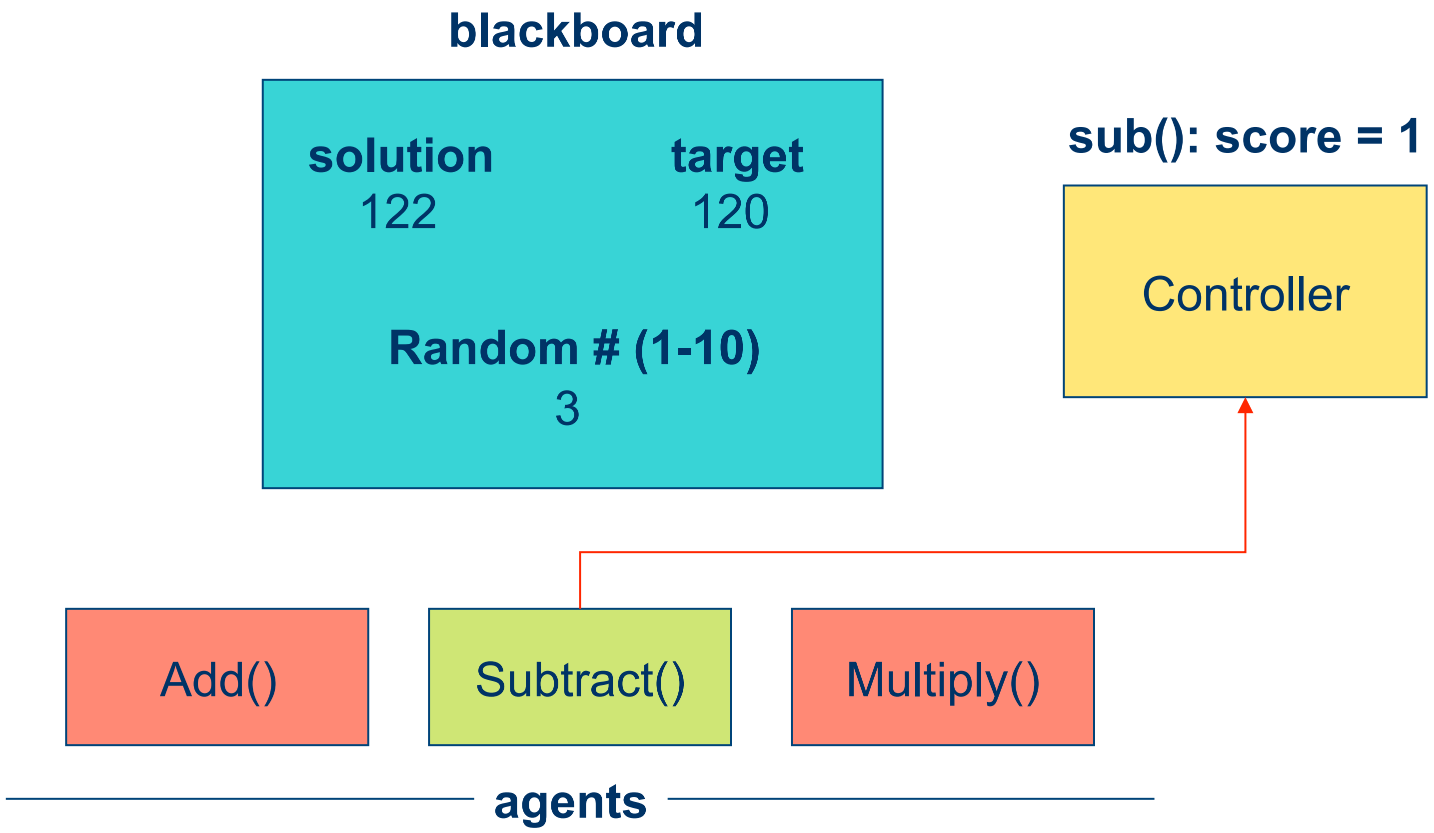
Example: Iteration 4



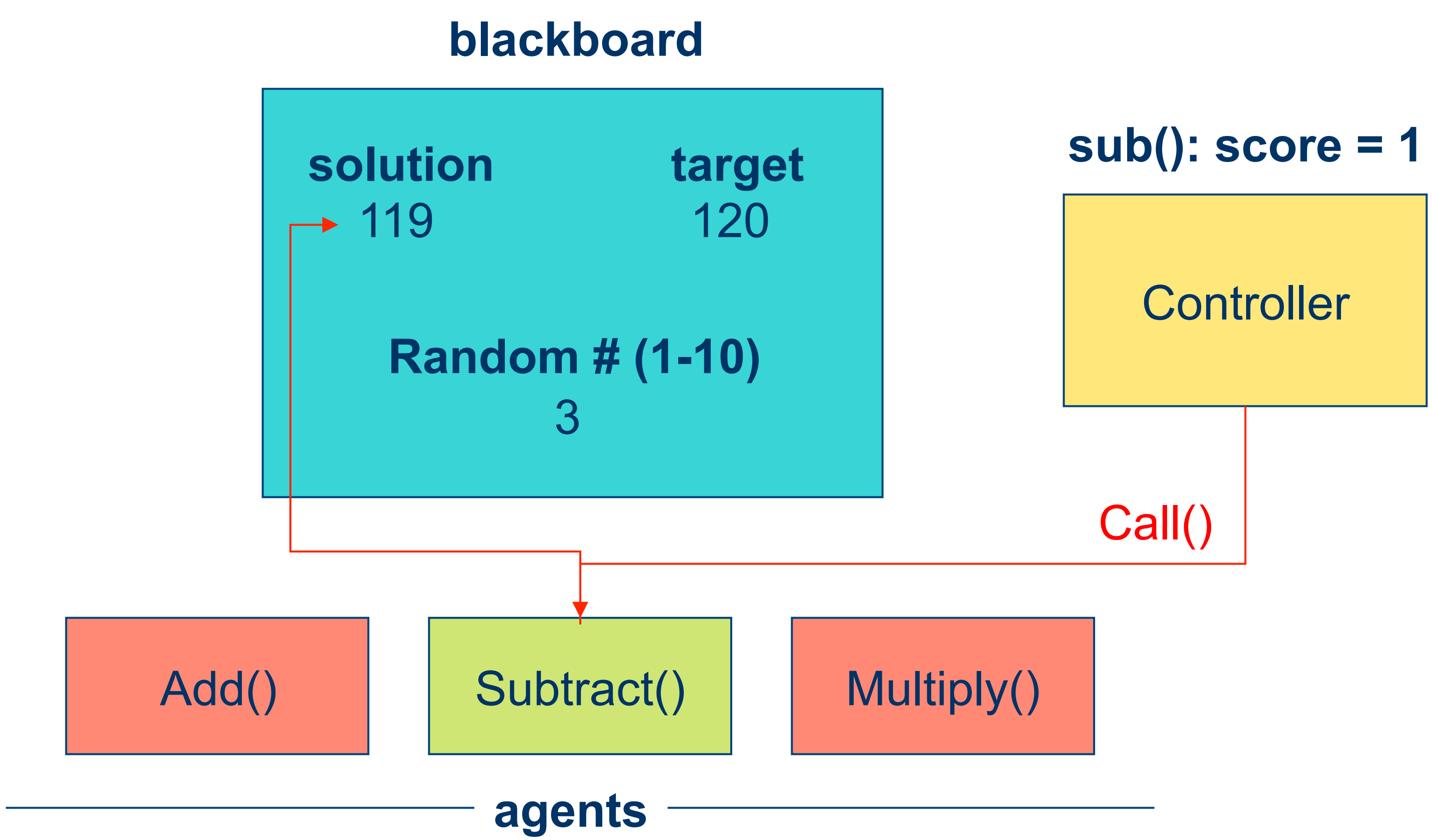
Example: Iteration 4



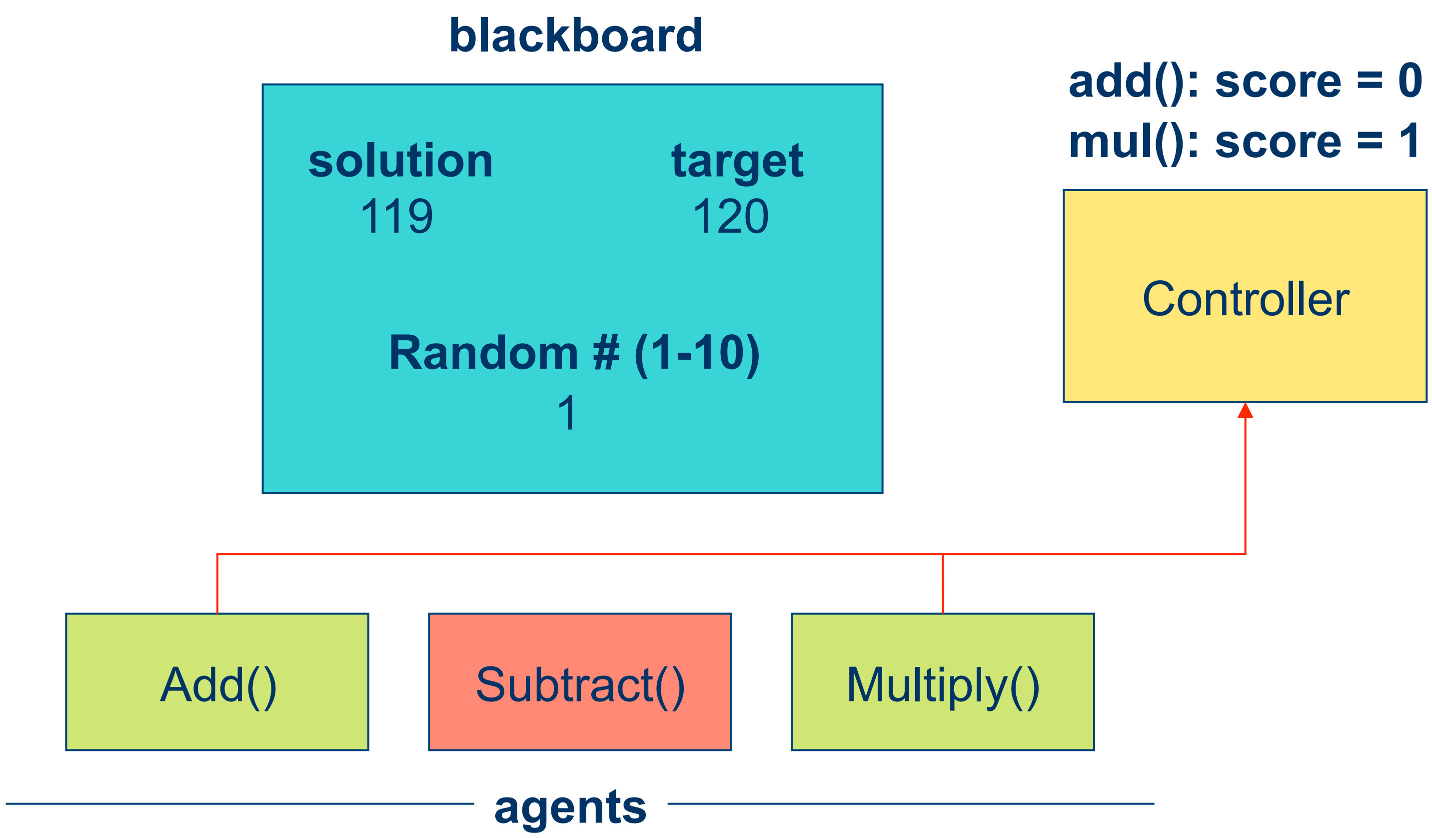
Example: Iteration 5



Example: Iteration 5



Example: Iteration 6



Example: Iteration 6