

# Providing Self-Aware Systems with Reflexivity



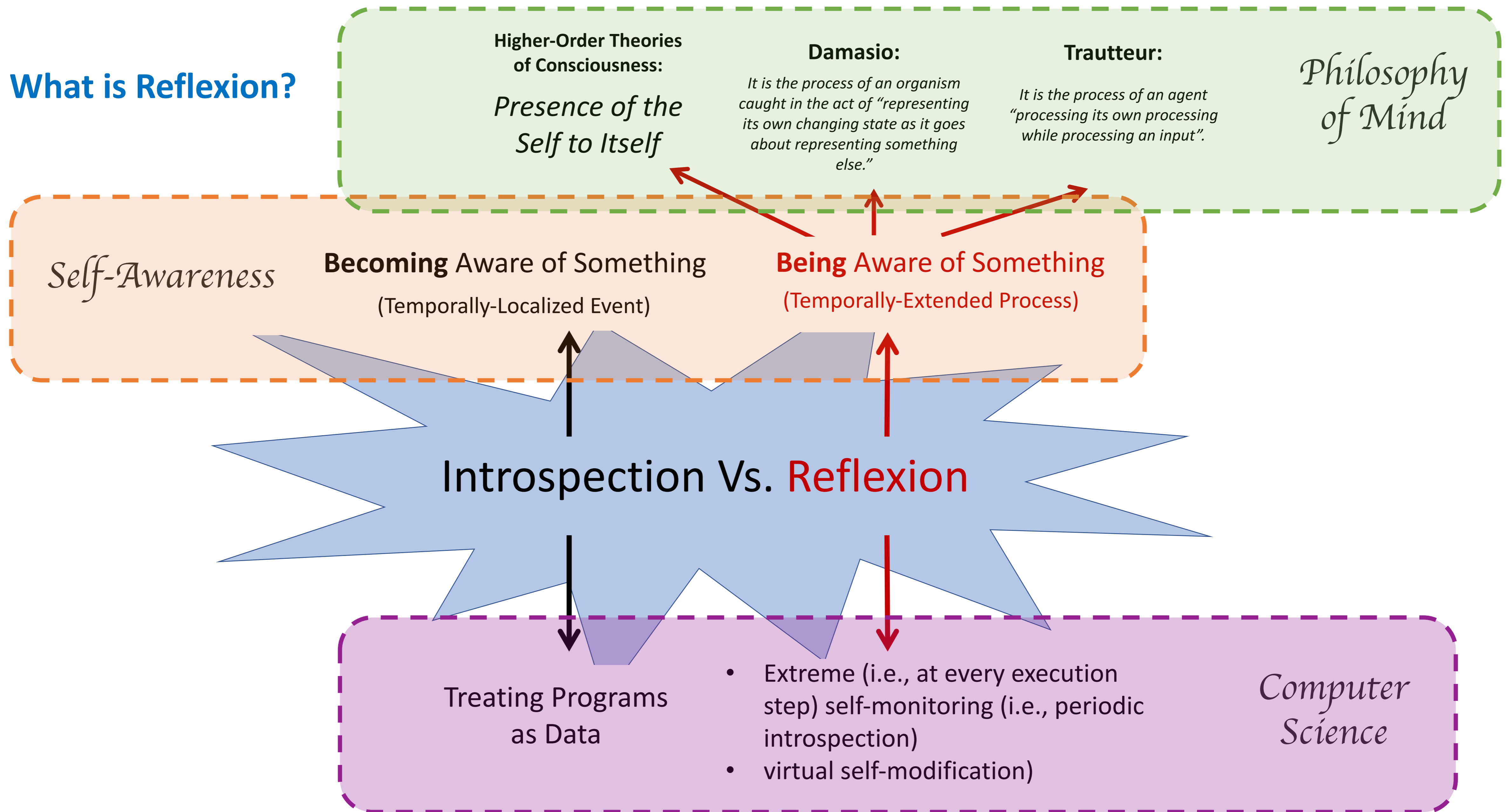
Alessandro Valitutti  
alessandro.valitutti@gmail.com

Giuseppe Trautteur  
trau@na.infn.it



The code of the proof-of-concept prototype is available at the URL: <http://valitutti.it/papers/reflexion/index.html>

## What is Reflexion?



## Computational Self-Awareness

	Introspection	Self-Monitoring	Meta-Programming	Reflexivity
Self-Representation	✓	✓	✓	✓
Self-Modification			✓	✓
Persistency		✓		✓

## Building up Computation Reflexion

The table shows different versions of the interpretation loop, with the addition of step components, and related computational processes.

Symbol	Step Components	Process Creation	Process
$(S_L)$ →	Lower Step	Standard Execution	Target Process
$(S_L, I_S)$ ↘	+ Single Introspection	Tracing	Execution Trace
$(S_L, I_S, S_{SU})$ ↗	+ Single Upper Step	Mirroring	Mirror Process
$(S_L, I_S, S_{DU})$ ↗	→ Double Upper Step	Augmentation	Augmented Process
$(S_L, I_D, S_{DU})$ ↗	→ Double Introspection	<b>Reflexion</b>	<b>Reflexive Process</b>

Two key characteristics of computational reflexivity:

- **Duality:** There are two concurrent processes (i.e., the *target process* and the *augmented process*).
- **Modularity:** It is a property of a particular type of interpreter, so it can be provided to any possible executable program.

- We employed and modified the code of a Lisp meta-circular interpreter (the **Lisp in Lisp**). It is one of the simplest ways to implement a general-purpose interpreter.
- At **every stage of the computation**, the interpreter accesses and executes the code both **locally** (current function call) and **globally** (access to the main function code).
- The code of the target program can be modified at each step, thus influencing the next execution and performing the **virtual self-modification**.

## Prototype