$$Debugging : \boldsymbol{T} \rightarrow \boldsymbol{Sets}_{\boldsymbol{DA+TA}}$$

Debugging and Category Theory

https://www.dumpanalysis.org/debugging-category-theory

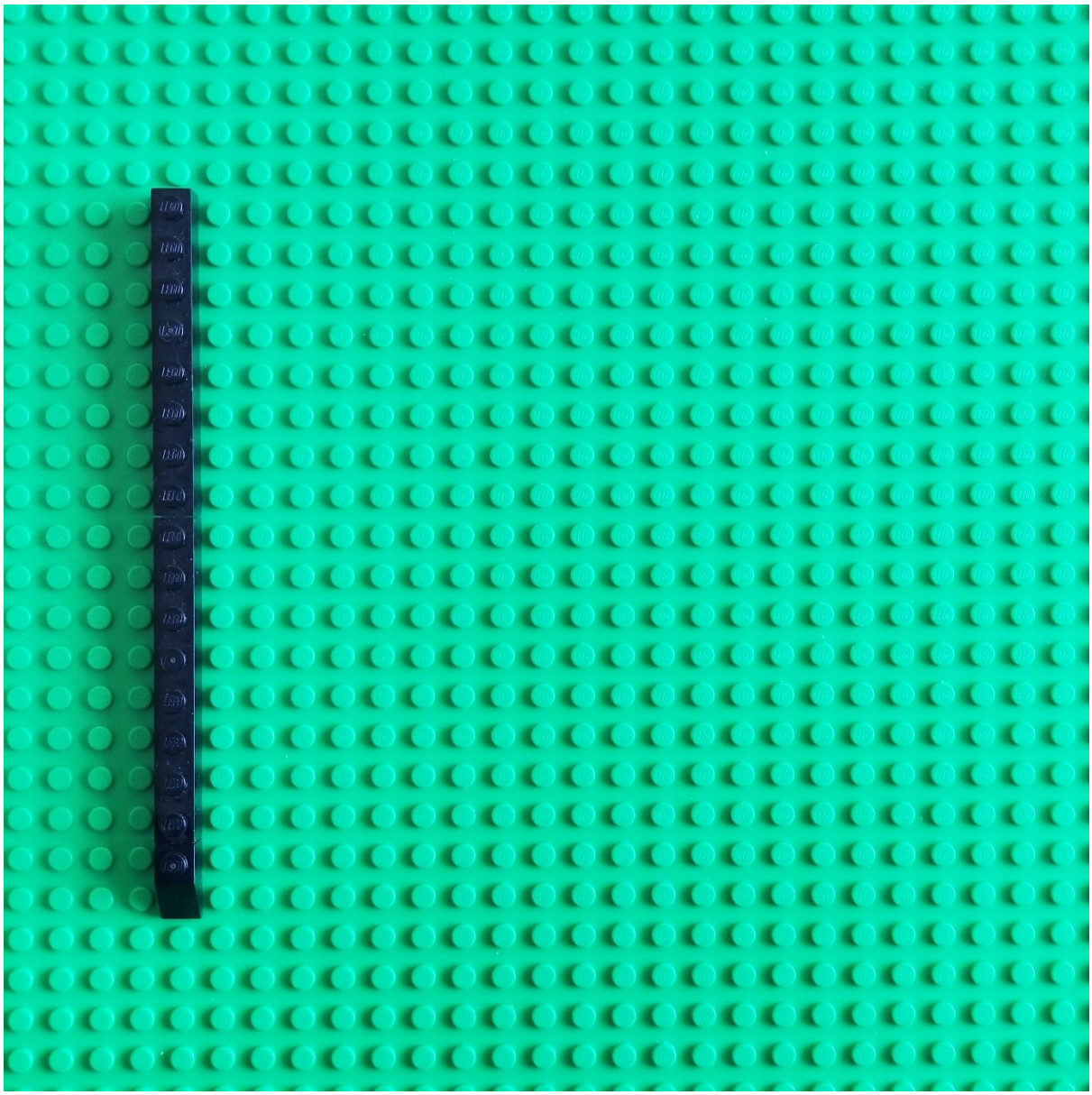
What is debugging? There are many definitions out there, including analogies with forensic science, victimology, and criminology. There are also definitions involving set theory. They focus on the content of debugging artifacts such as source code and its execution paths and values. We give a different definition based on debugging actions and using category theory. We also do not use mathematical notation in what follows.

What is the category theory? We do not give a precise mathematical definition based on axioms but provide a conceptual one as a worldview while omitting many details. A category is a collection of objects and associated arrows between them. Every pair of objects has a collection of arrows between them, which can be empty. So an arrow must have a source and a target object. Several sequential arrows can be composed into one arrow. We can even consider arrows as objects themselves, but this is another category with its new arrows between arrows as objects. If we consider categories as objects and arrows between these categories as objects, we have another category. So we can quickly build complex models out of that.

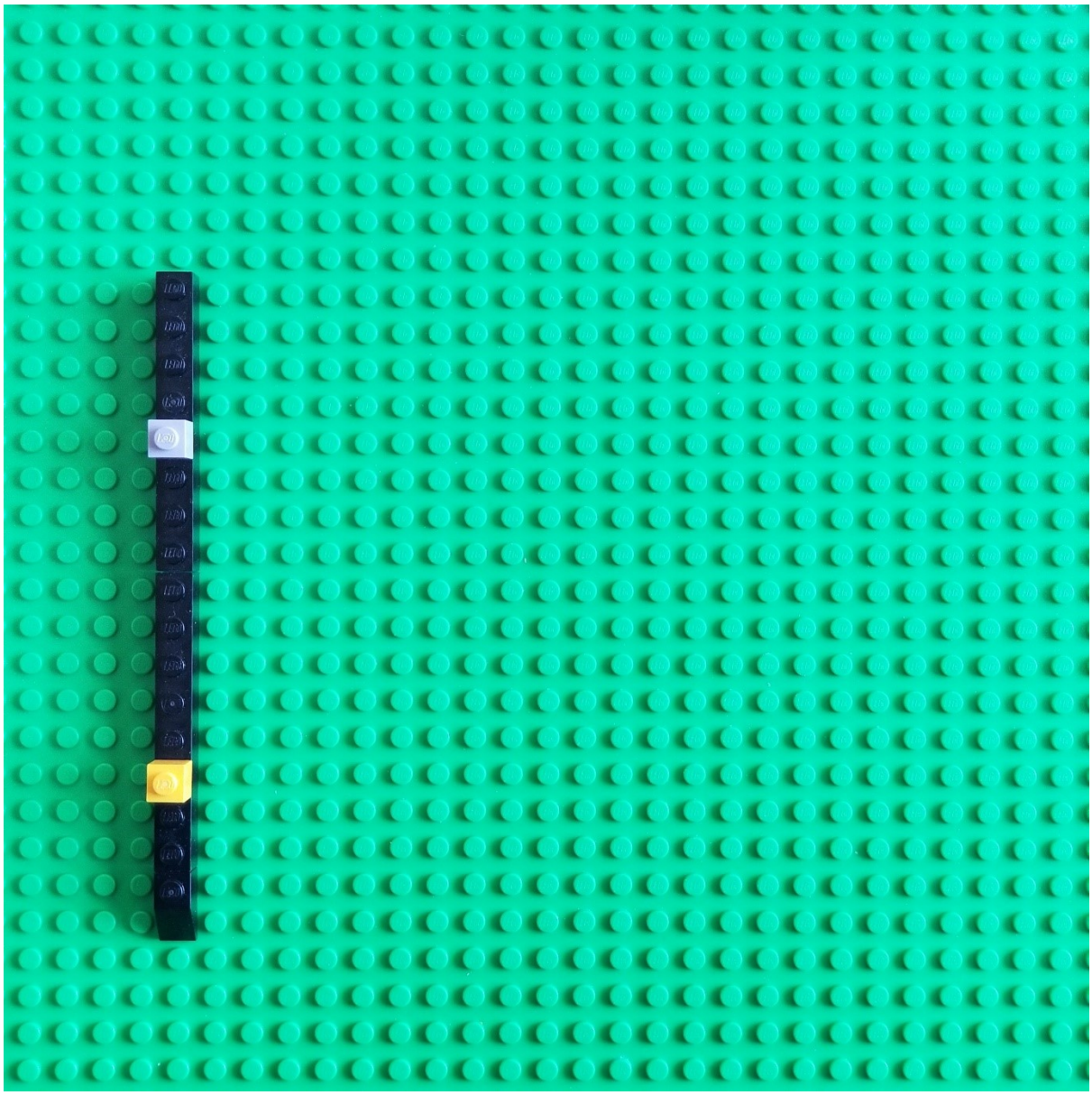
Can we build a conceptual model of debugging using objects and arrows? Yes, and it even has a particular name in category theory: a presheaf. So, debugging is a presheaf. To answer a question, what is a presheaf, we start constructing our debugging model focusing on objects and arrows. To avoid using mathematical language that may obscure debugging concepts, we use LEGO® bricks because we can feel the objects and arrows, and most importantly, arrows as objects[1]. This hands-on activity also reminds us that debugging is a construction process.

[1] Visual Category Theory (https://www.dumpanalysis.org/visual-category-theory)

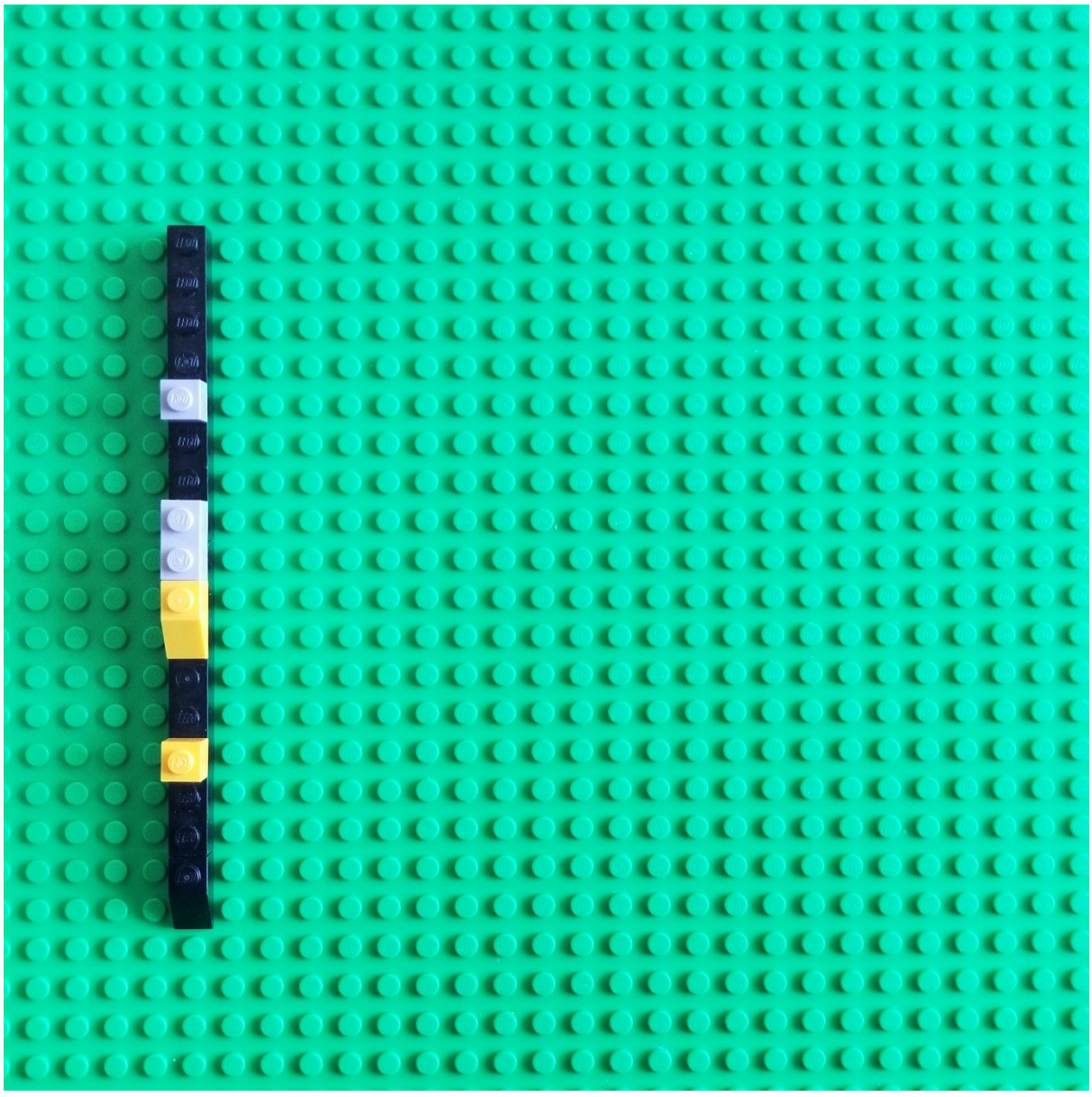
Debugging activity involves time. We, therefore, construct a time arrow that represents software execution:



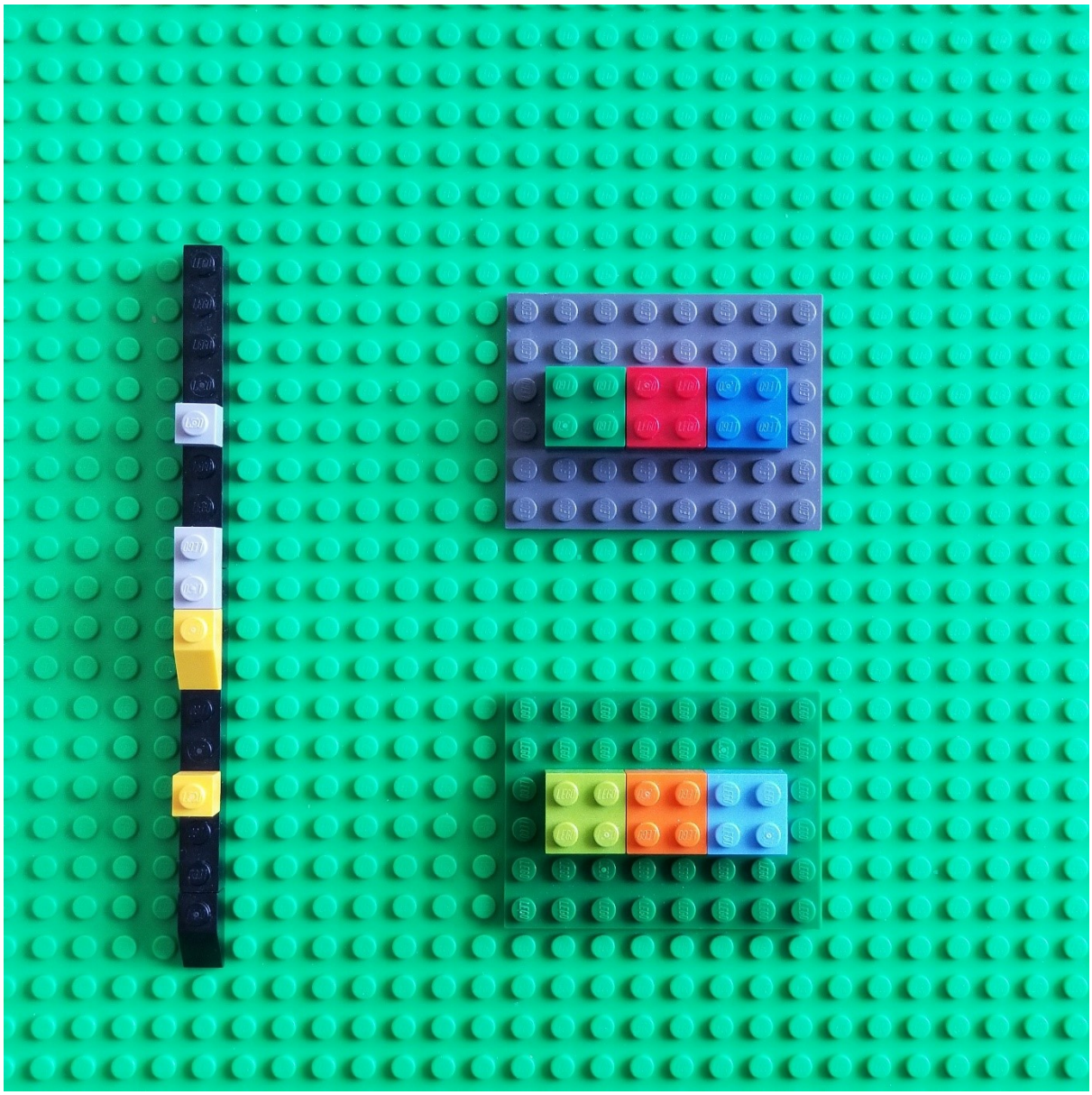
We pick two **Time** objects representing different execution times:



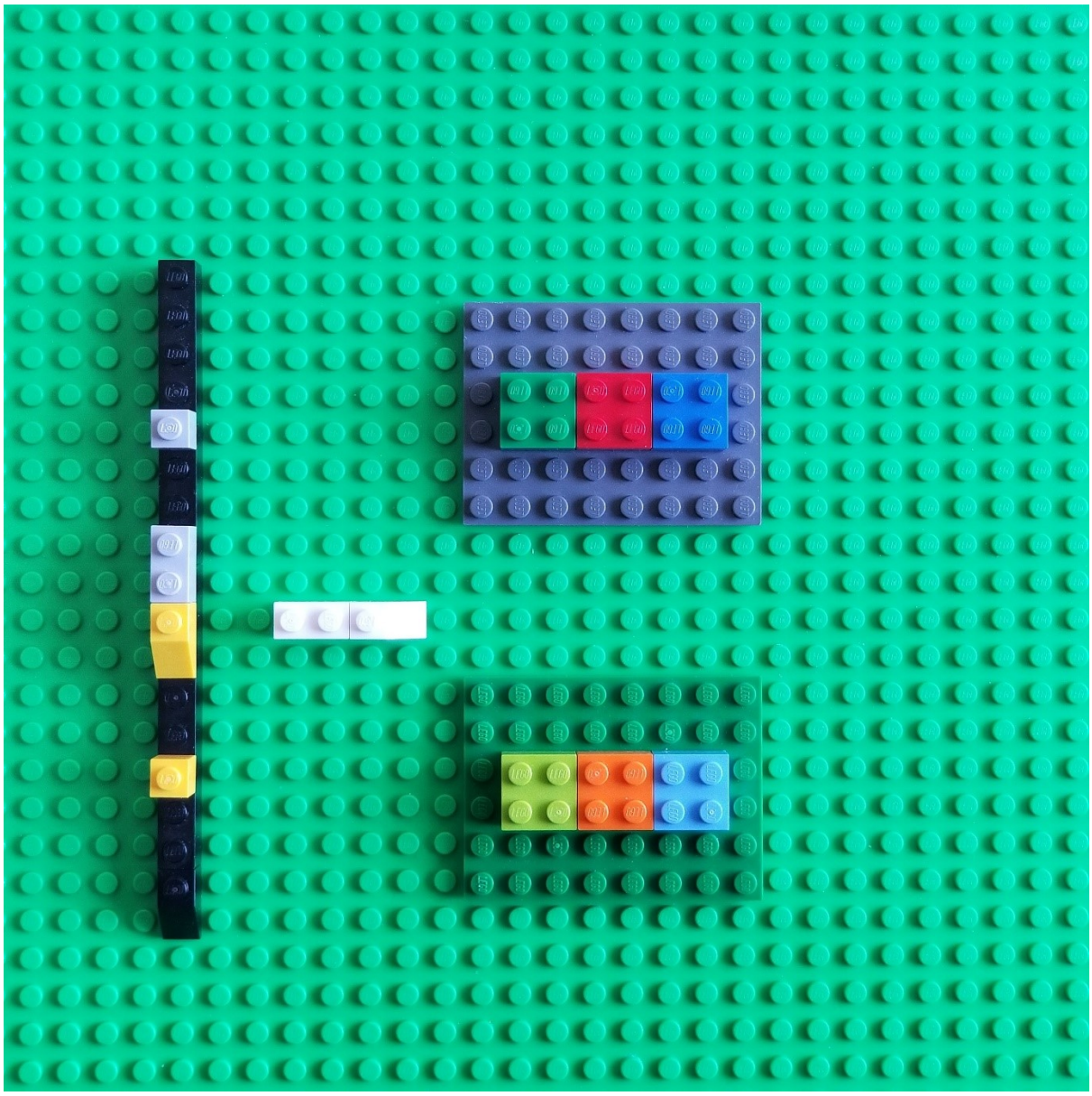
In our **Time** category, an arrow means the flow of time. It can also be some indexing scheme for time events or other objects (a different category) that represents some repeated activity. Please note that an arrow has specific object indicators assigned to it. Different object pairs have different arrows. It is not apparent when we use black and white mathematical notation and diagrams.



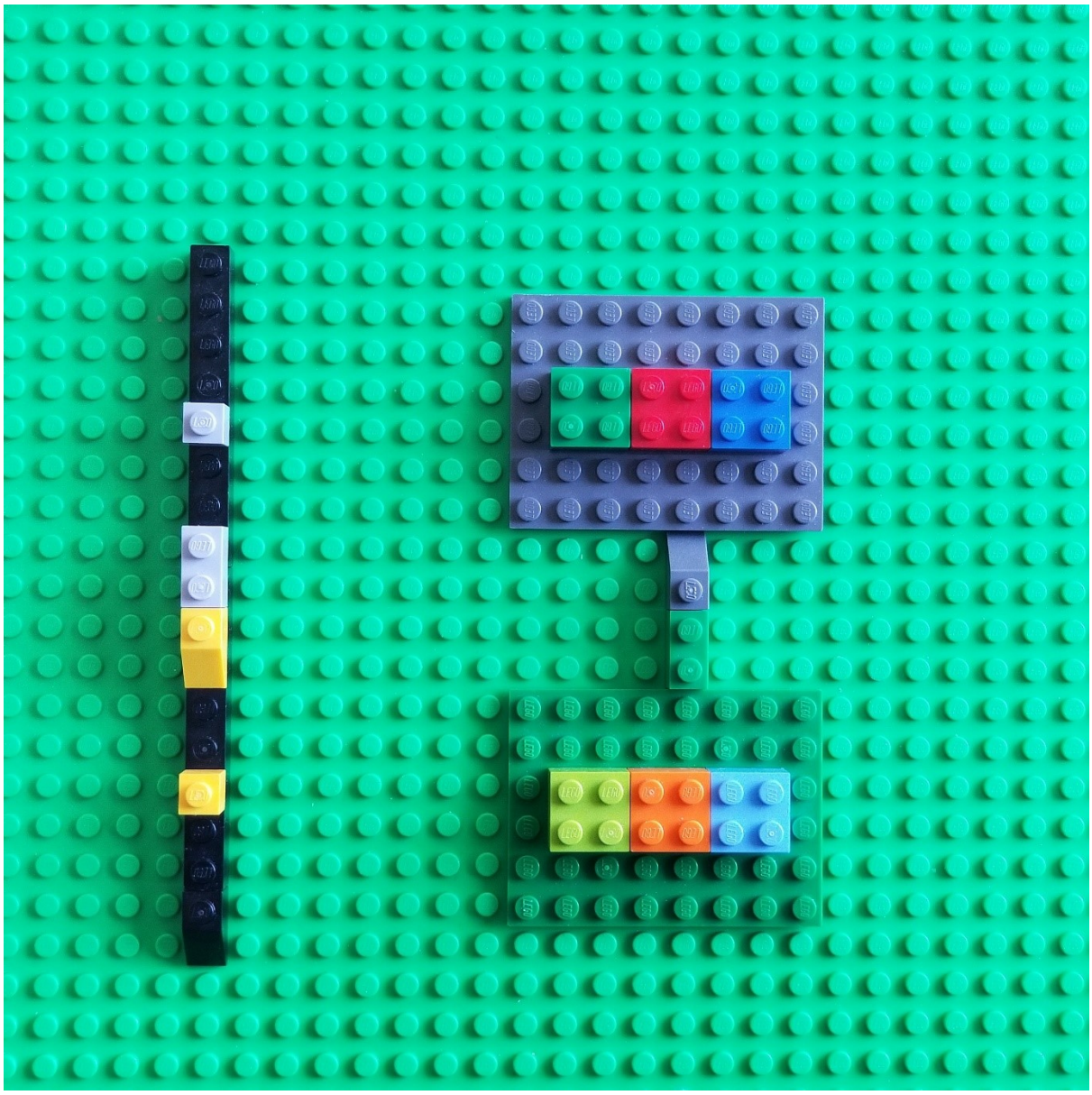
We can associate with **Time** objects some external objects, for example, memory snapshots, or some other software execution states, variables, execution artifacts, or even parts of the same artifact:



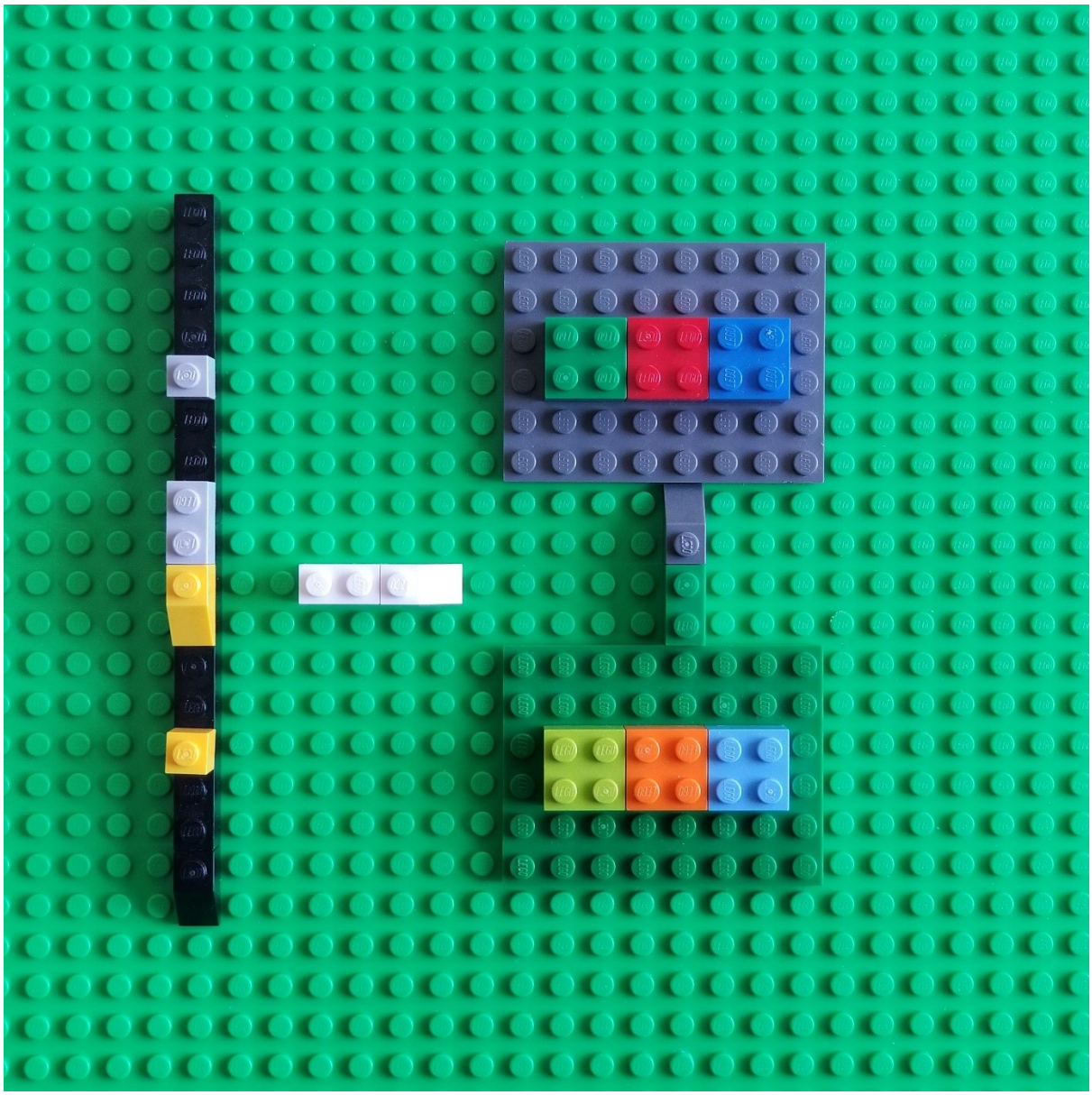
Therefore, we have a possible mapping from the **Time** category to a possible category of software execution artifacts that we name **DA+TA** (abbreviated [memory] dump artifacts + trace [log] artifacts). **DA+TA** objects are simply some sets useful for debugging. The mapping between different categories is usually called a functor in category theory. It maps objects from the source category to objects in the target category. It is itself an arrow in the category that includes source and target categories as objects:



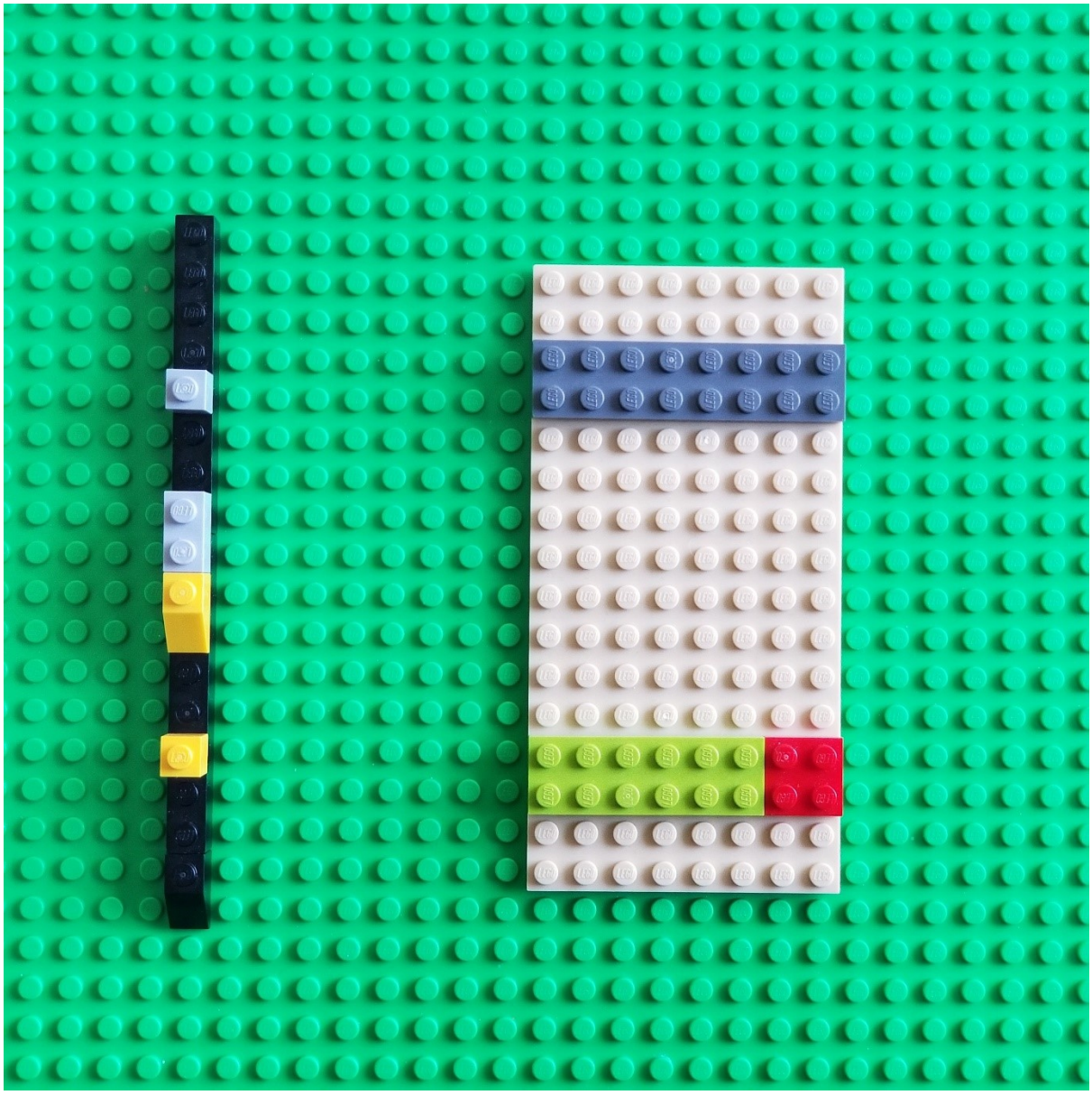
However, we forgot to designate arrows in the target **DA+TA** category. Of course, a different choice of arrows makes different categories. We choose arrows that represent debugging activities such as going back in time when trying to find the root cause, such as walking a stack trace. It is a reverse activity:



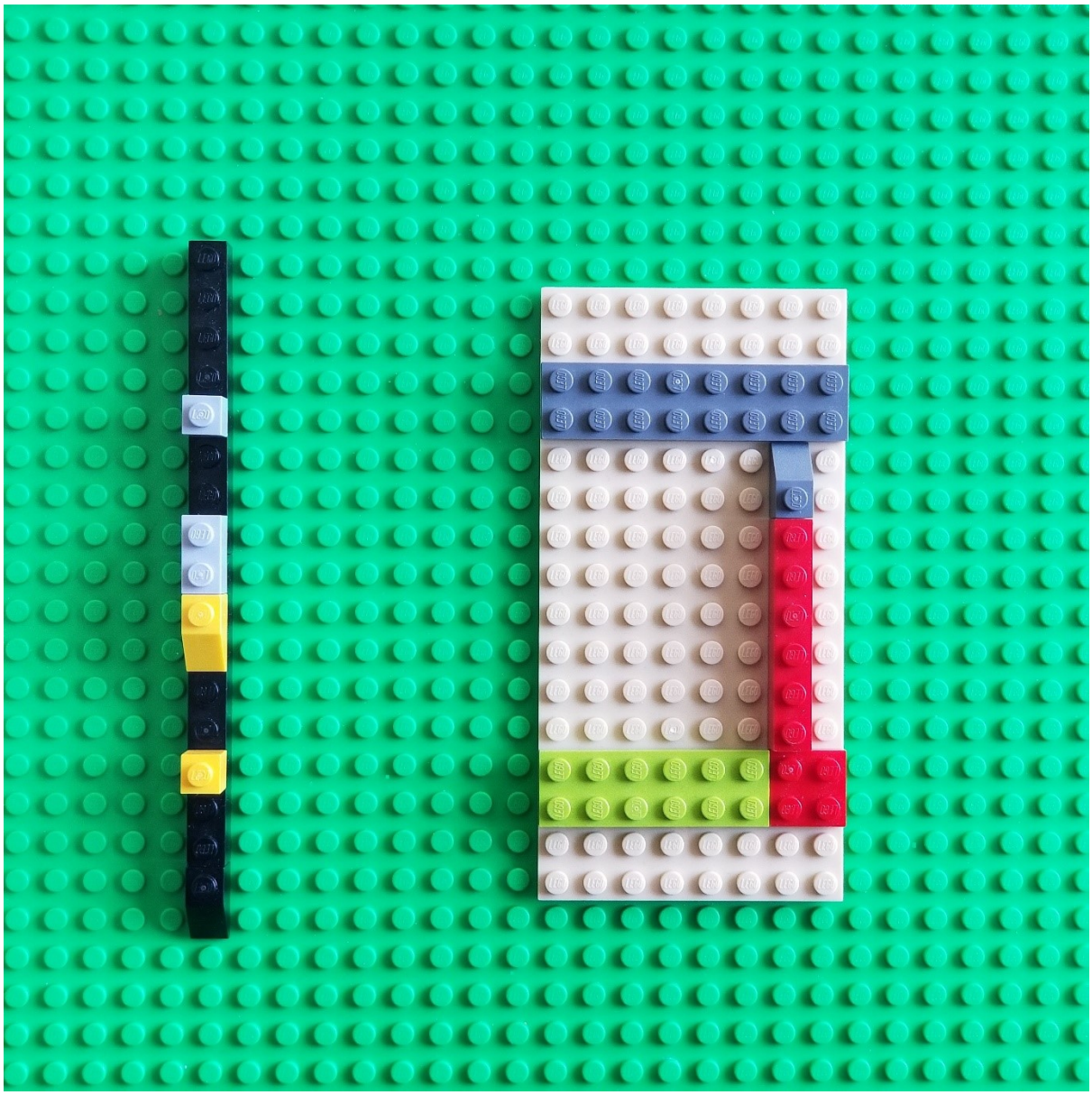
A functor that maps arrows to reversed arrows is called a contravariant functor in category theory:



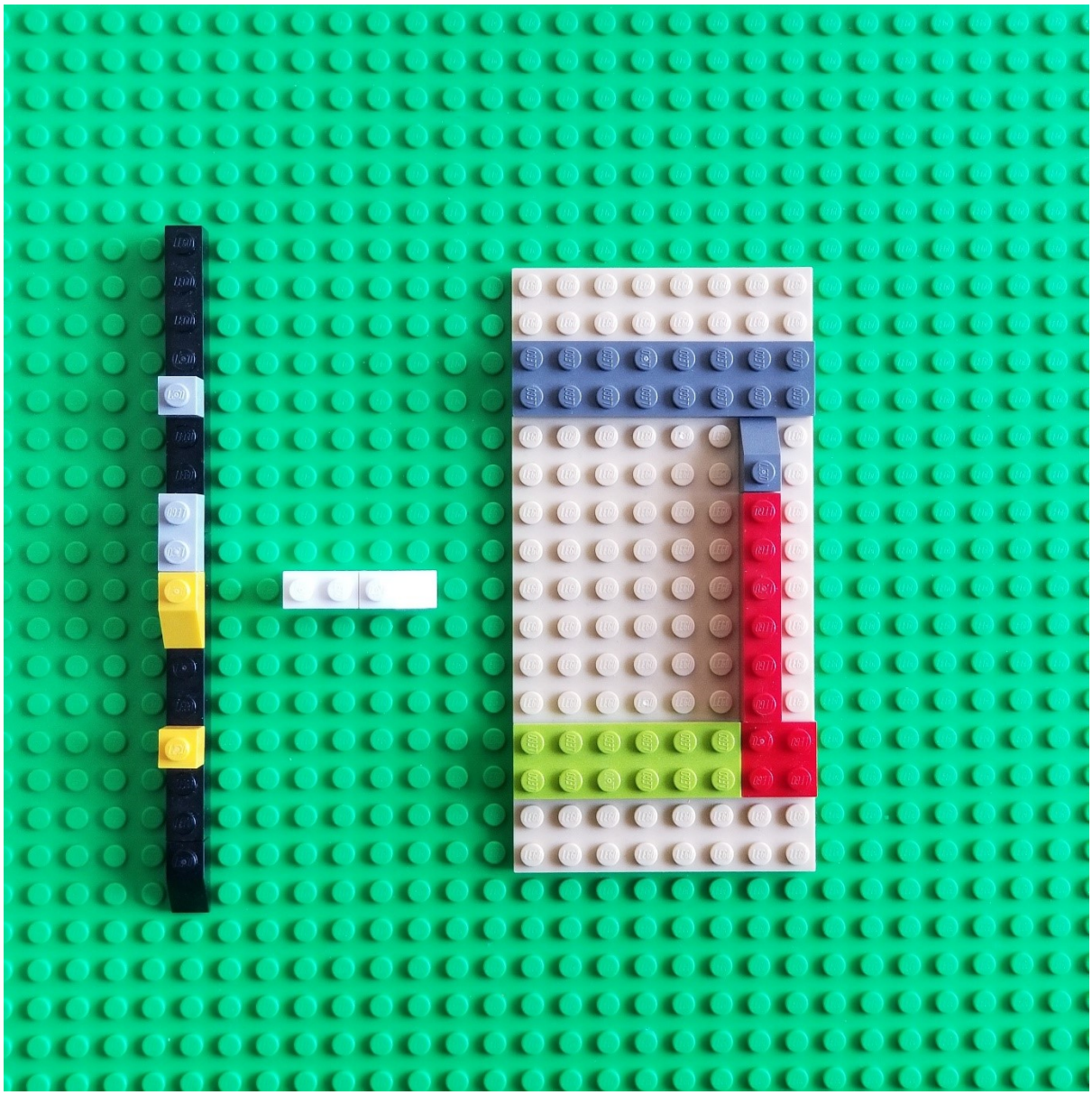
Such a contravariant functor from a category to the category of some sets is called a presheaf. Now we look at debugging using software traces and logs as another target category of sets. With our **Time** category objects, we associate different log messages:



When we use log and trace files for debugging we also go back in time trying to find the root cause message (or a set of messages) or some other clues:



Again, we have a presheaf, a contravariant functor that maps our **Time** category objects to sets of messages:



So, we see again, that debugging is a presheaf, a contravariant functor that maps software execution categories such as a category of time instants to sets of software execution artifacts.

Trace and log analysis pattern catalog includes another example of the source and target categories candidates for a debugging presheaf, **Trace Presheaf** analysis pattern[2] that maps trace messages to memory snapshots (sets of memory cells or some other state information).

Presheaves can be mapped to each other, for example, from a presheaf of logs to a presheaf of associated source code fragments or stack traces, and this is called a natural transformation in category theory. It also fits with natural debugging when we go back in logs and, at the same time, browse source code or some other associated information sets.

[2] Trace and Log Analysis: A Pattern Reference for Diagnostics and Anomaly Detection (https://www.dumpanalysis.org/trace-log-analysis-pattern-reference)