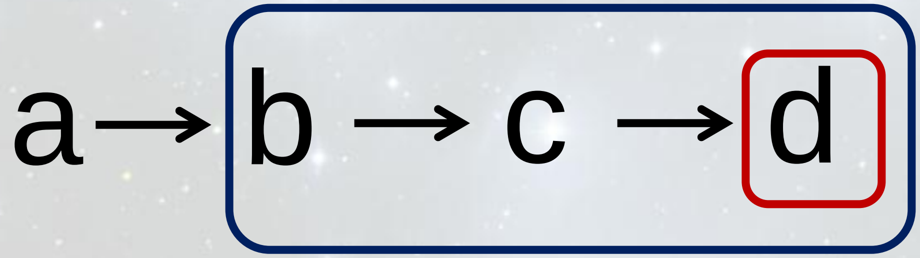
A Self-Replication Algorithm to Flexibly Match Join Point Traces

Paul Leger and Éric Tanter
Department of Computer Science
University of Chile

Stateful Aspects
In a Nutshell

An aspect can only match a **single join point**

A stateful aspect can match a **join point trace**
[Douence+2005]

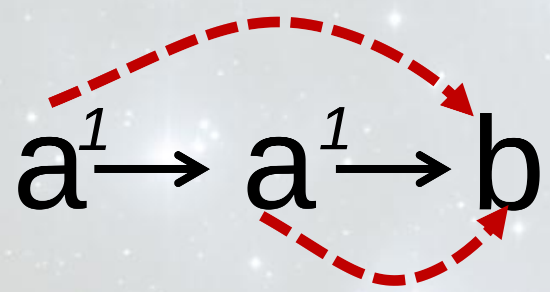


Stateful aspects are used in security flaws, application errors, and crosscutting concerns

Algorithms to Match Join Point Traces

Sequence $a^v \rightarrow b$

Join Point Trace



The matches of a sequence
depend on
the matching semantics of the algorithm

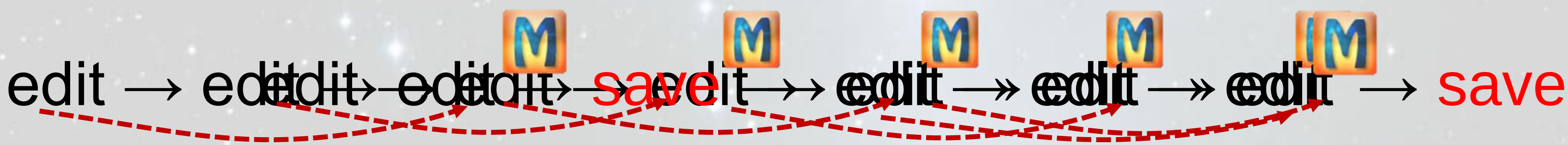
Fixed Semantics to Match Traces

Autosave feature: the document is automatically saved every three editions

```
tracematch() {
  sym edit after: call(Editor.edit());
  sym save after: call(Editor.save());
  edit edit edit {
    Editor.save();
} }
```

Tracematches support multiple matches

An artificial symbol is added to support single match



To adapt the matching semantics of an algorithm, developers code around it

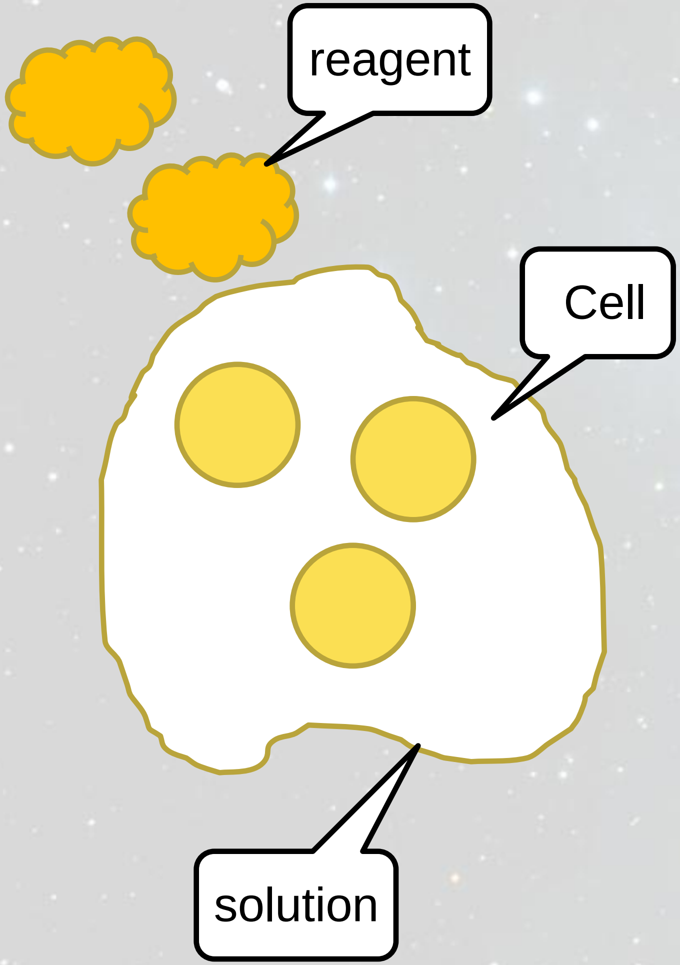
Matcher Cells

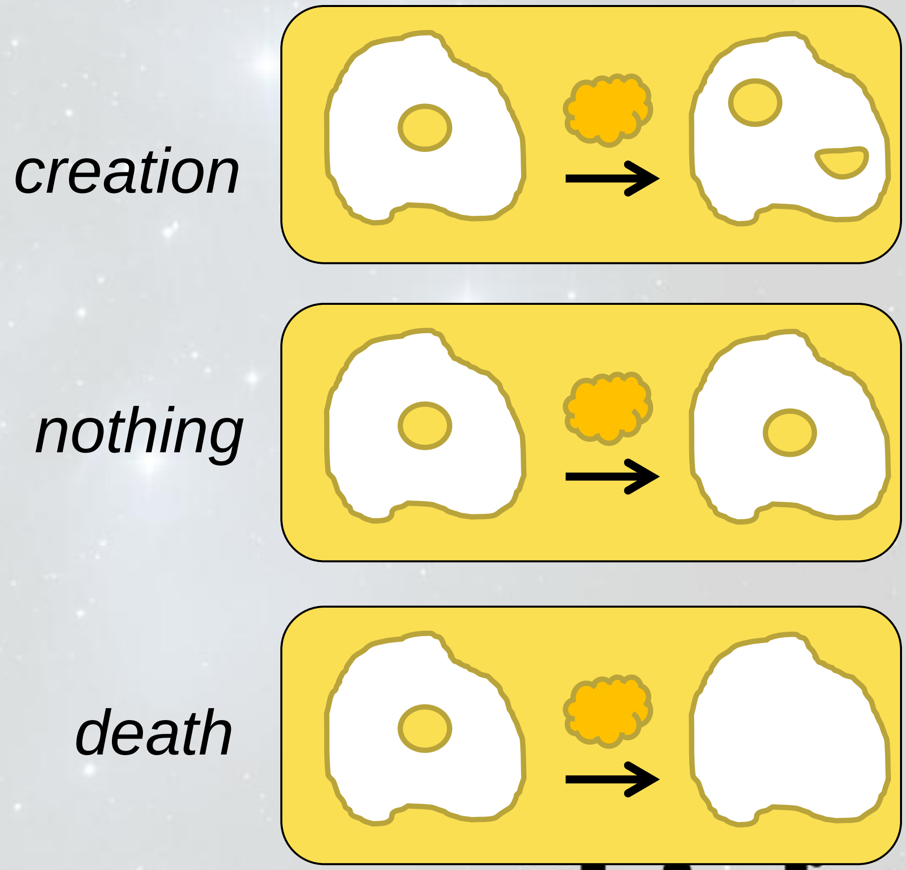
An algorithm to flexibly match join point traces, where developers can define their own semantics

Based on self-replication behavior

Self-Replicating Behavior In a Nutshell

Reactions of biological ***cells*** into a ***solution*** to a ***reagent*** trace

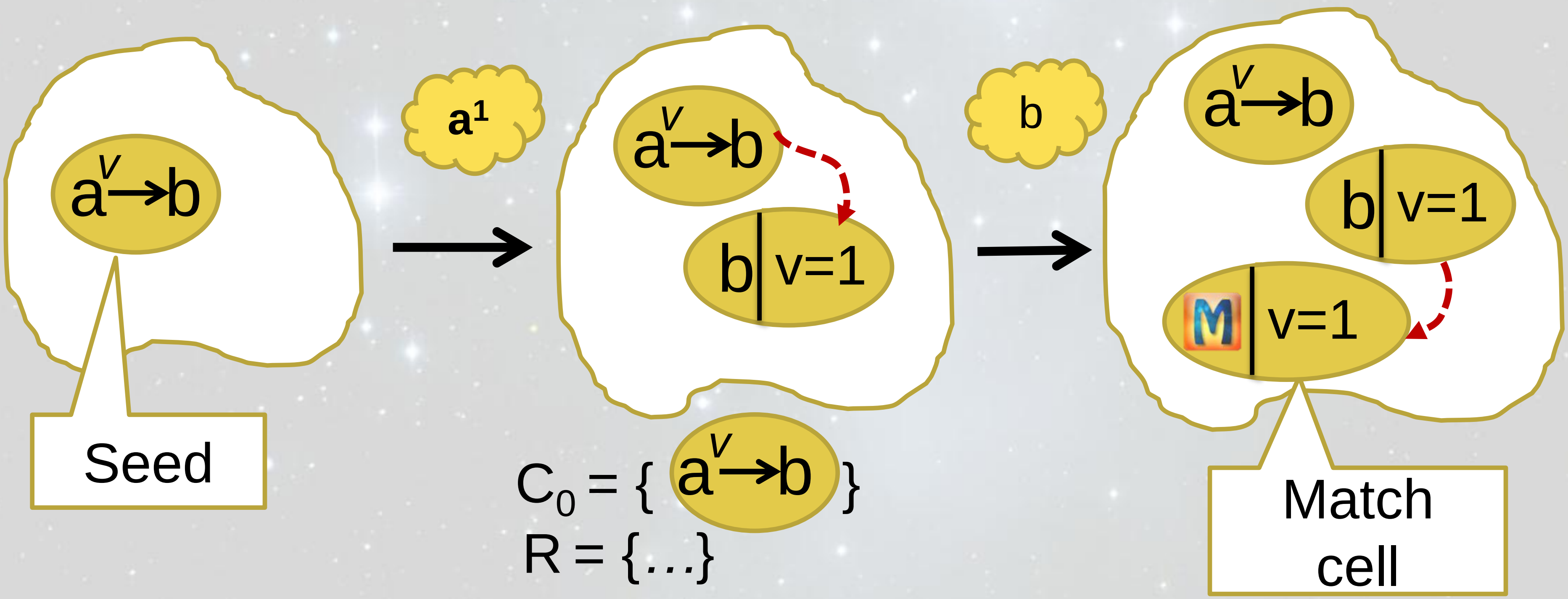




Pleiad

Matcher Cells

A *cell* contains a sequence and bound variables, and a *reagent* corresponds to a join point



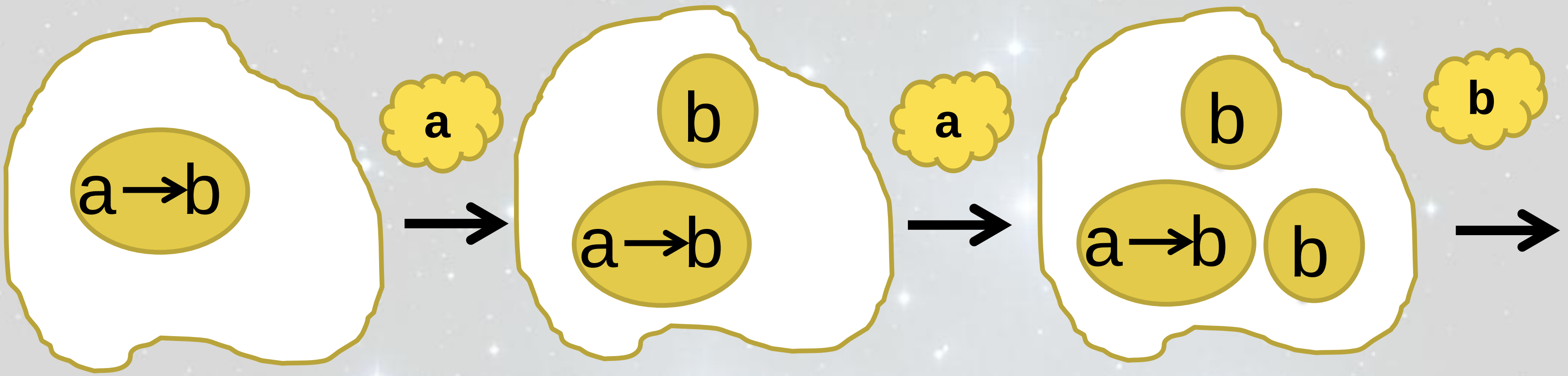
Pleiad

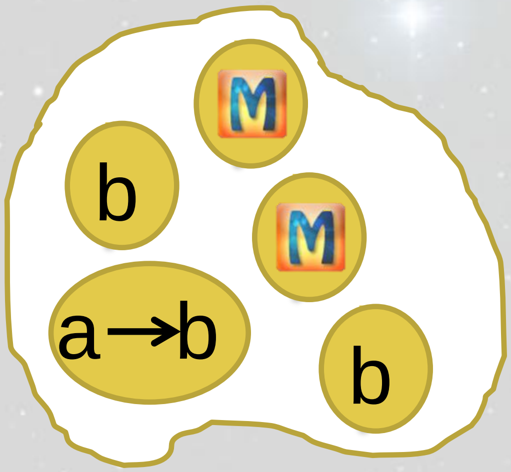
Pleiad
Programming Languages and Environments for Intelligent, Adaptable and Distributed systems

Examples of Matching Semantics with Matcher Cells

With simple reaction rules, Matcher Cells makes it possible to express a wide range of matching semantics

Multiple Matches

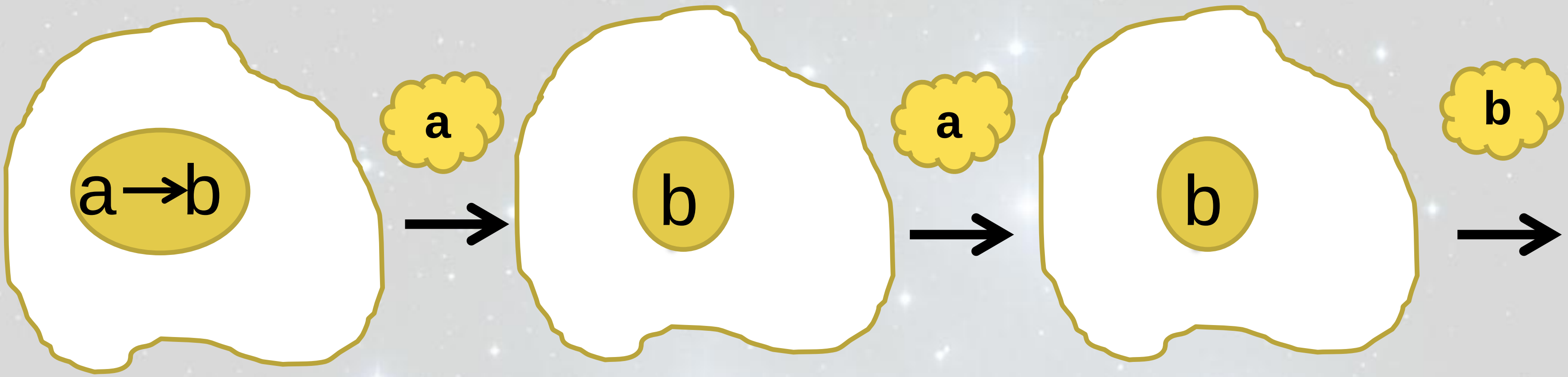




C_0 = { a→b }

R = {*apply reaction*}

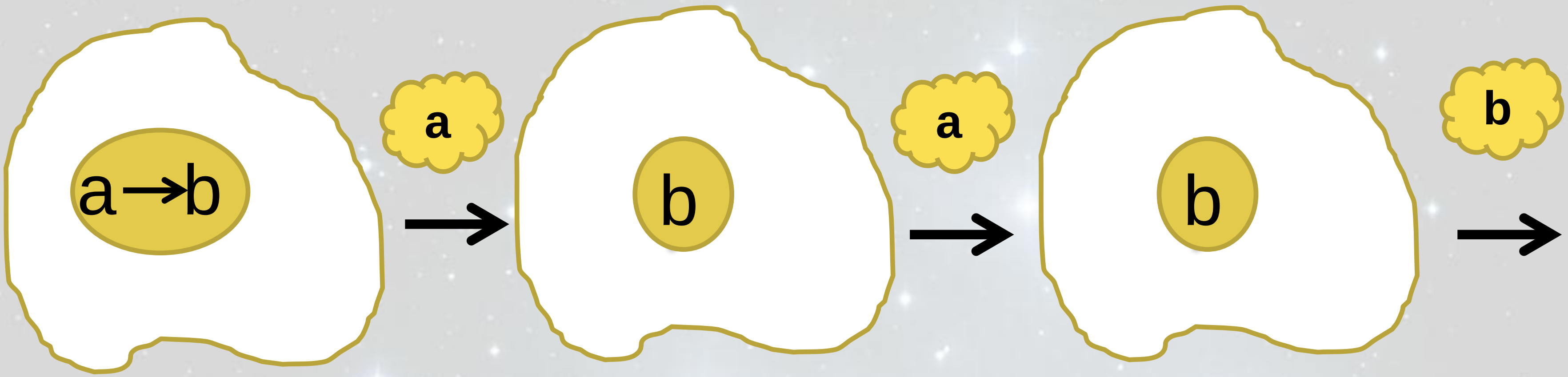
Single Match

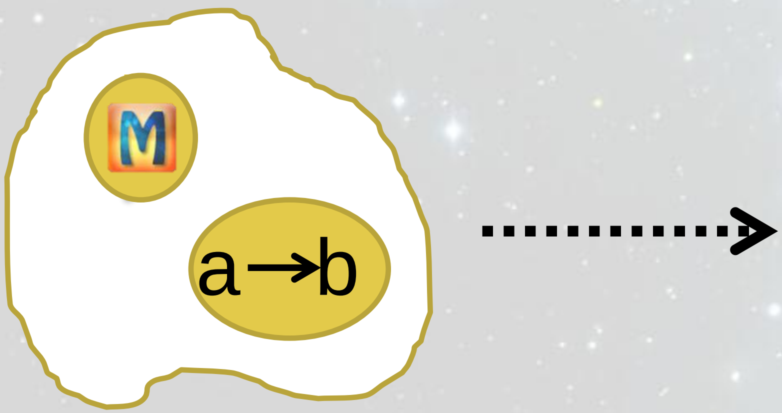


C_0 = {a→b}

R = {*apply reaction*, *kill creators*}

Single Match at a Time
(the *autosave feature* solution)



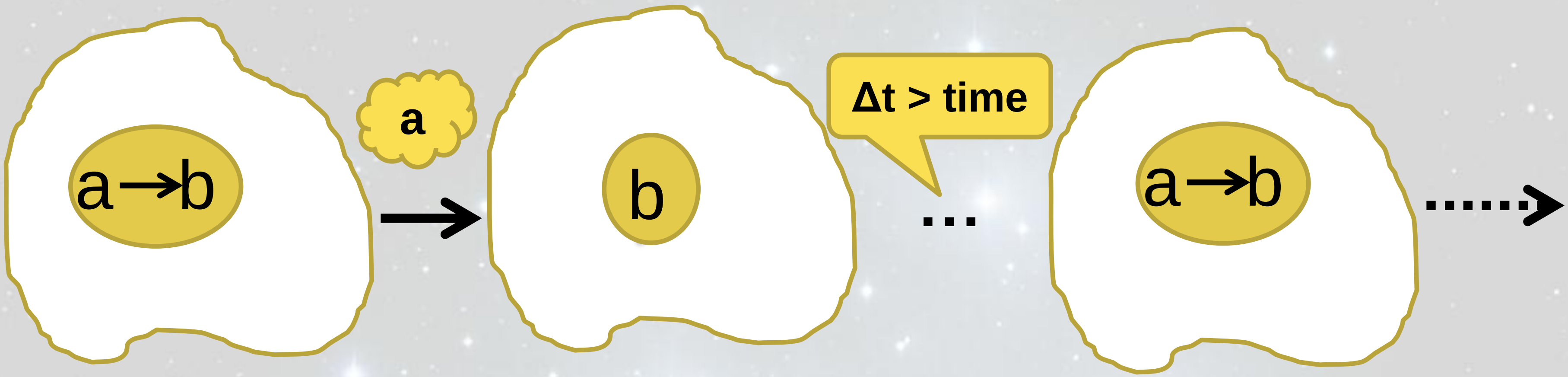


C_0 = {a→b}

R = {*apply reaction,*
kill creators,
add seed}

Pleiad

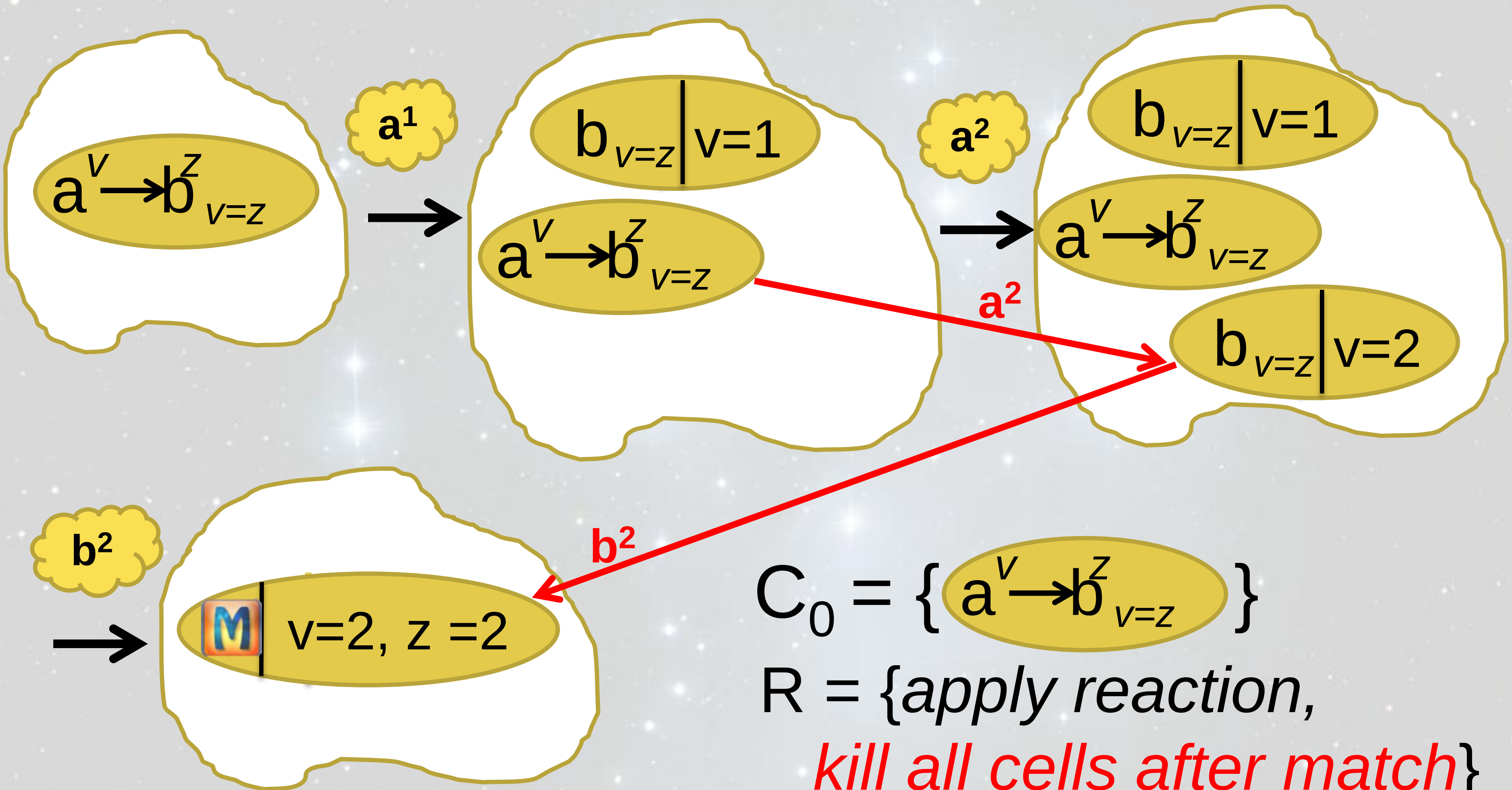
Life-time for a Match



$C_0 = \{a \rightarrow b\}$

R = {*apply reaction,*
kill creators,
trace life-time,
add seed}

Only the First Match



$C_0 = \{ a^v \rightarrow b^z_{v=z} \}$

R = {*apply reaction,*
kill all cells after match}

An implementation of Matcher Cells

Matching semantics is defined by
the composition of rules (small functions)

Reaction of a Cell

react: Cell **x** JP $\rightarrow$ Cell

- returns a *new cell* if *matches*
- returns the *same cell* if *does not match* the join point

Rules

rule: List<Cell> x JP → List<Cell>

```
var applyReaction = function (cells, jp) {
  return removeDuplicates(append(cells, map(cells, react, jp))); }
```

The elemental rule

```
var killCreators = function(rule) {
  return function (cells, jp) {
    var nextCells = rule (cells, jp);
    return difference(nextCells, getCreators(nextCells, cells)); } }
```

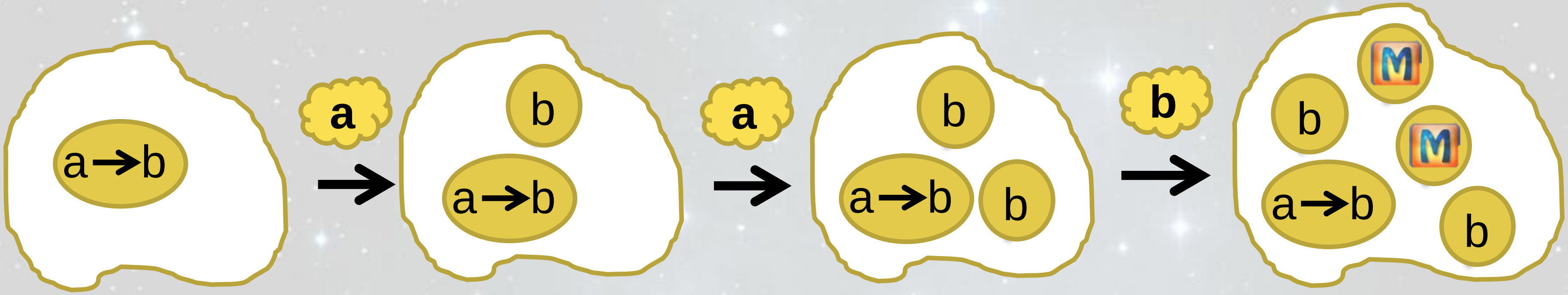
Rule designators allow rule composition

```
var addSeed = function(sequence) {
  return function (rule) {
    return function (cells, jp) {
      var nextCells = rule(cells, jp);
      return length(nextCells) == 0 || onlyMatchCells(nextCells)?
        append(nextCells,[createSeed(sequence)]): nextCells; }}}
```

Rule designators can be parametrized

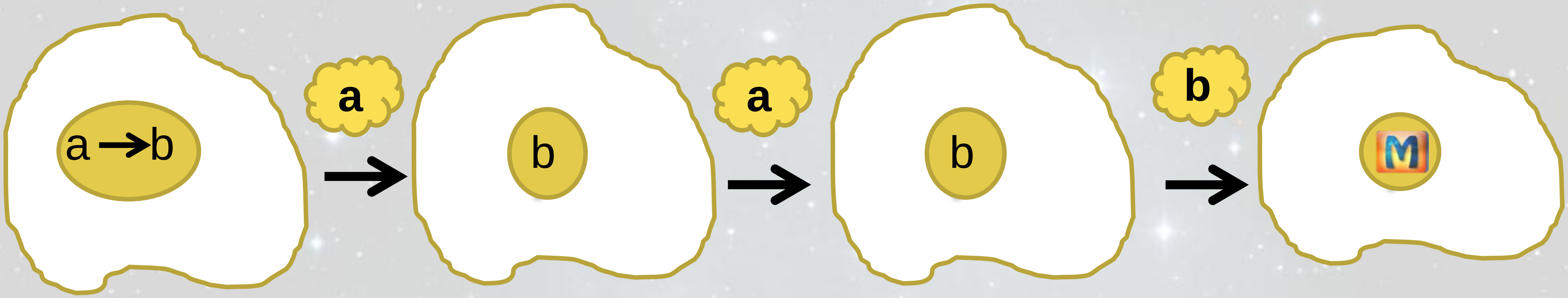
Pleiad

Multiple Matches



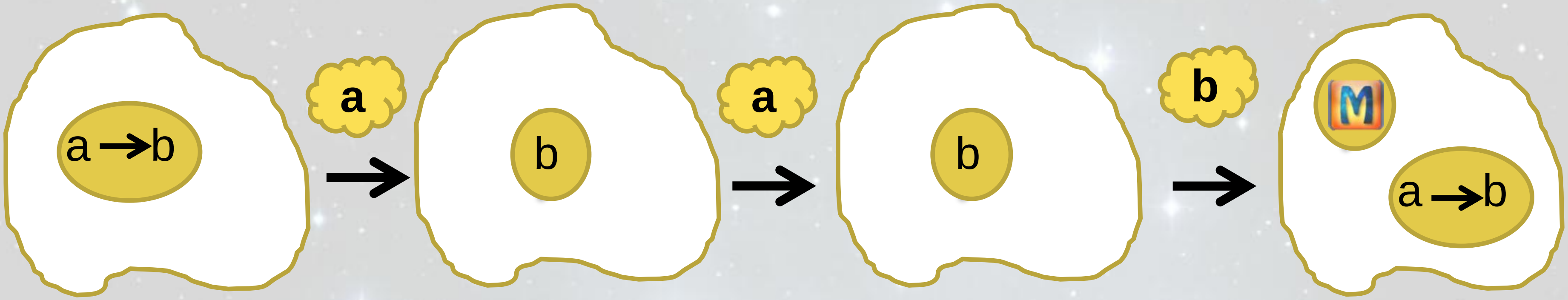

```
var multipleMatches = applyReaction;
```

Single Match



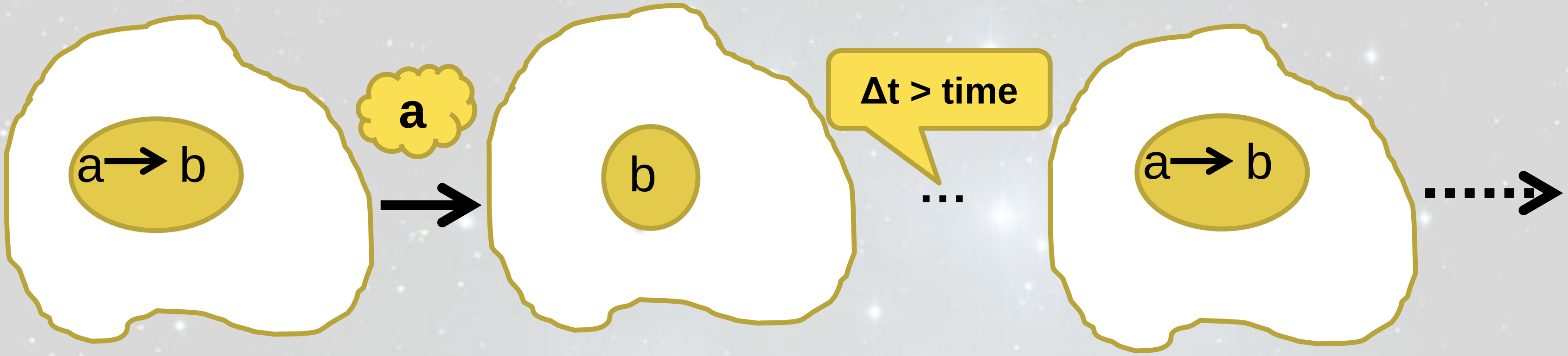

```
var singleMatch = killCreators(applyReaction);
```

Single Match at a Time



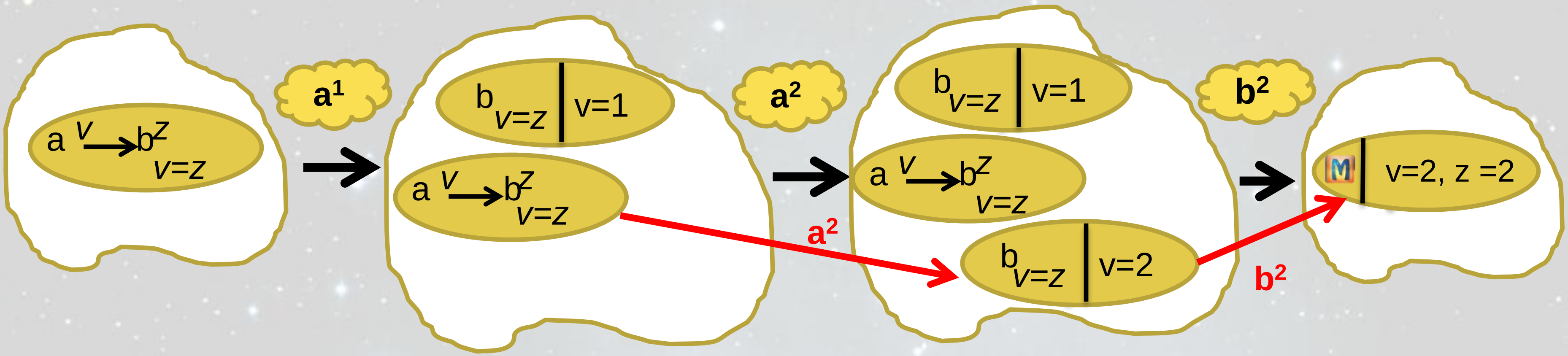

```
var singleMatchAtATime = addSeed(sequence)(killCreators(applyReaction));
```

Life-time for a Match




```
var lifeTimeForAMatch =
addSeed(sequence)(traceLifeTime(delta)(killCreators(applyReaction)));
```

Only the First Match




```
var onlyTheFirstMatch = killAllCellsAfterMatch(applyReaction);
```

Conclusions

The Matcher Cells algorithm

- allows developers to define their own matching semantics
- using the composition of reaction rules of self-replication algorithms

Application

We implement an expressive and open stateful aspect language using Matcher Cells (http://pleiad.cl/otm)

Try it on-line:
http://pleiad.cl/otm/matchercells

Pleiad

Adding Customized Information to Cells

Some rules require that *all cells* contain *customized information*

react: Cell **x** JP **x** [Seq **x** Env → Cell] → Cell

For example, the lifeTimeForAMatch rule requires a cell time

```
function (seq, env) {
   env = env.bind("time", getTime());
  return env;
};
```

Pleiad

Independence between Sequence Language and Matcher Cells

- The reaction of a cell strongly depends on the sequence language used

- When a cell matches a join point and/or binds a variable, the reaction of a cell has to return the next step in the matching

- Apart from the previous restriction, Matcher Cells does not impose another restriction to the sequence language

Pleiad