Debugging AST-Transformationen

Programming Languages: Concepts, Tools, and Environments

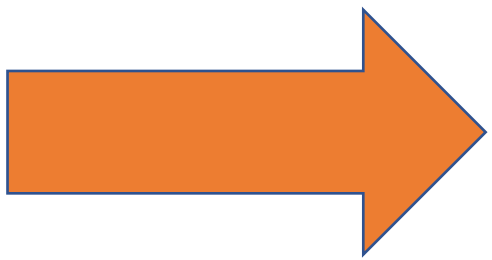
Tom Braun

Betreuer: Stefan Ramson

12.01.2021

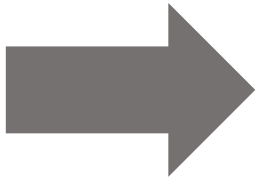
- Convert ECMAScript 2015+ code to backwards compatible code
- Polyfill missing features
- Source code transformations

5 + 8

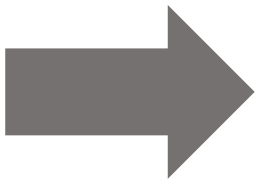


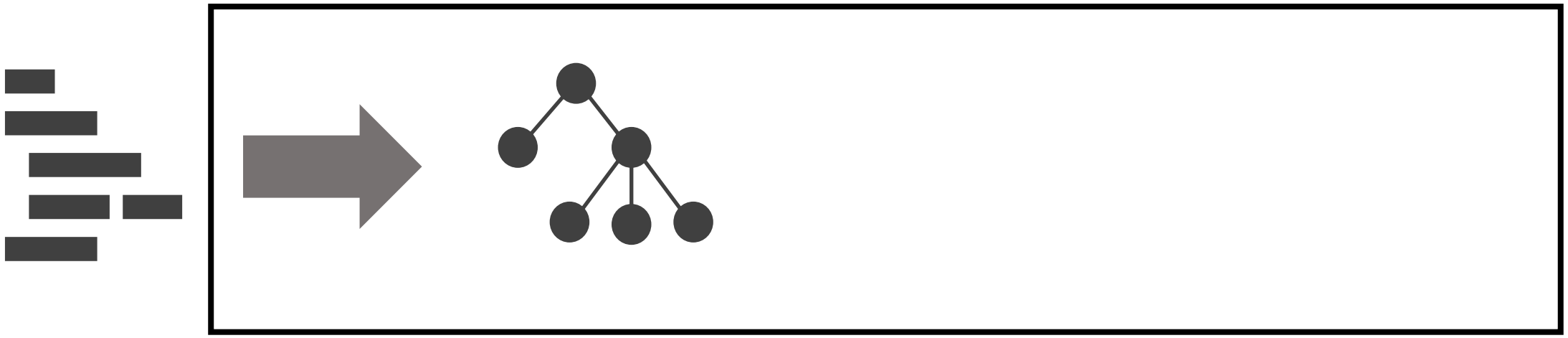
5 - 8

configure



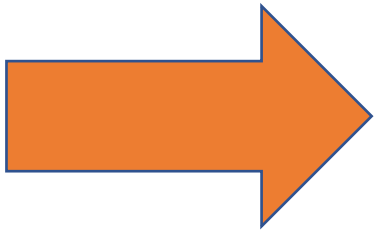
BABEL

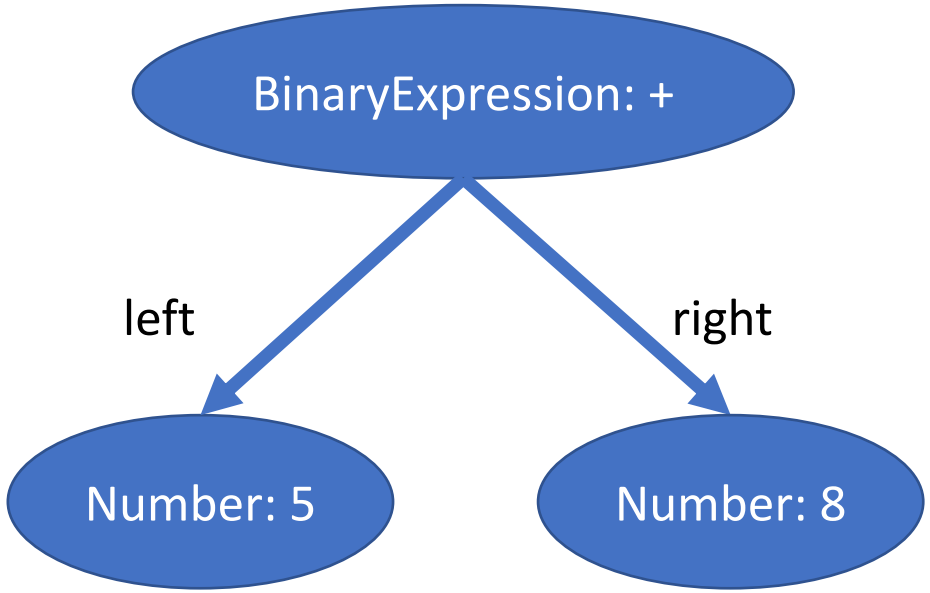


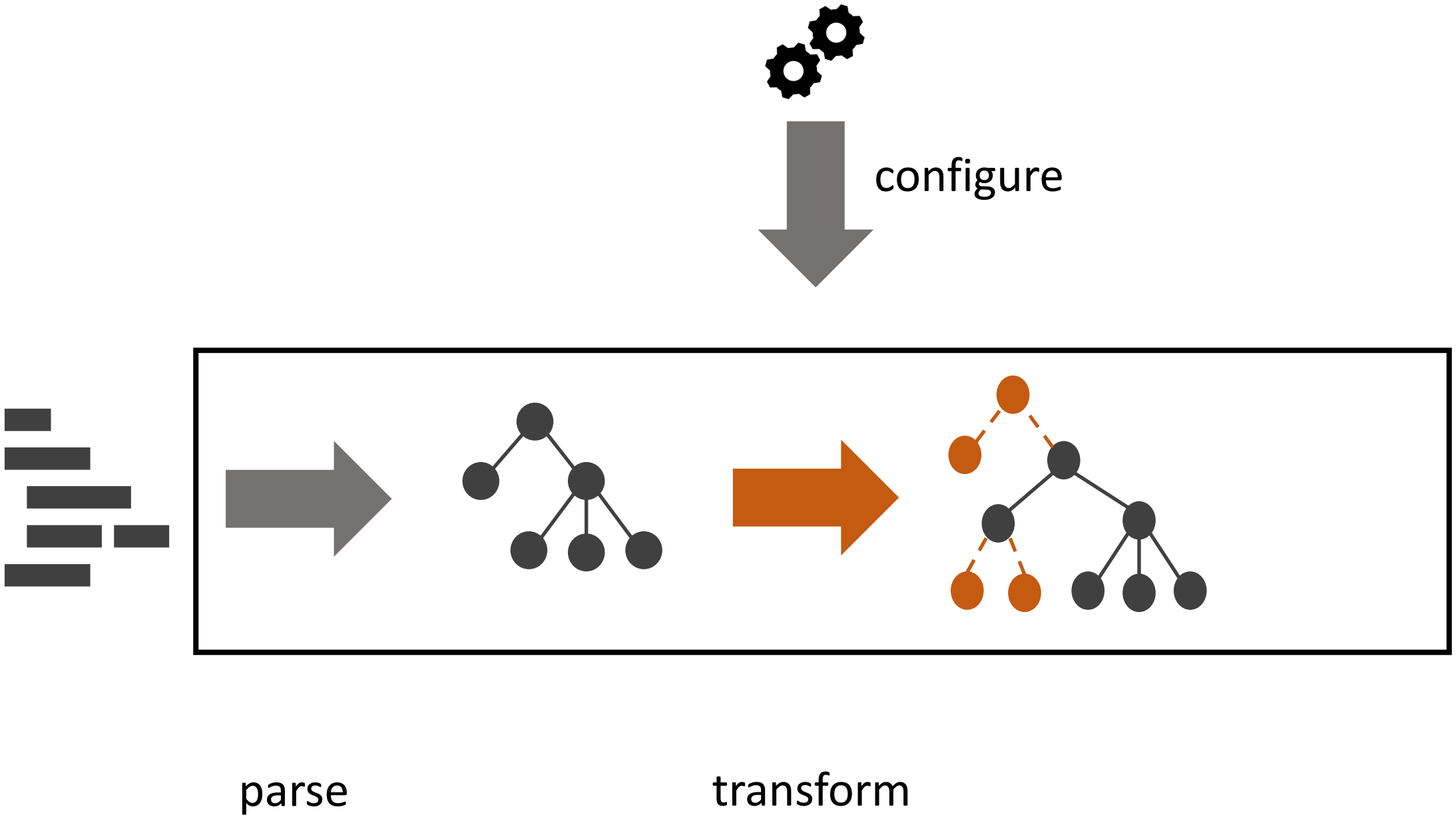


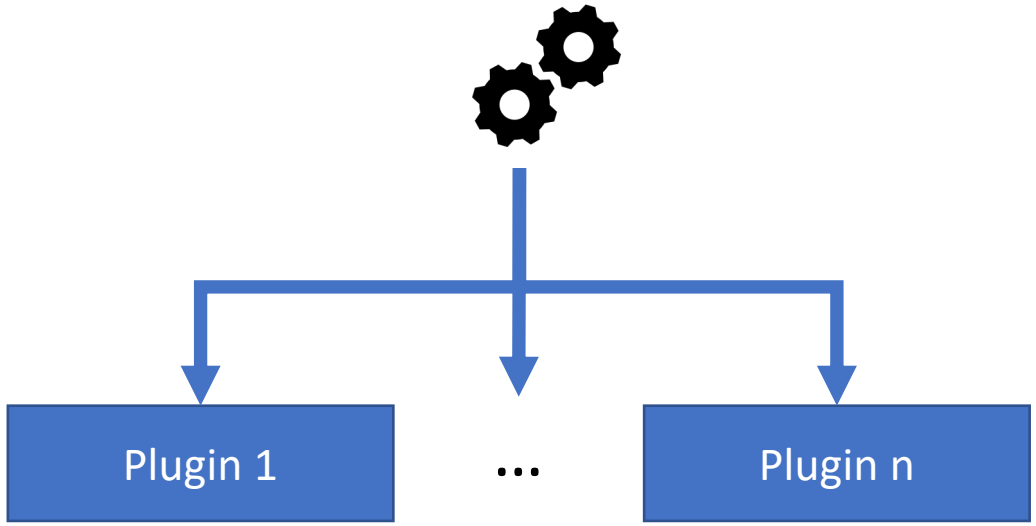
parse

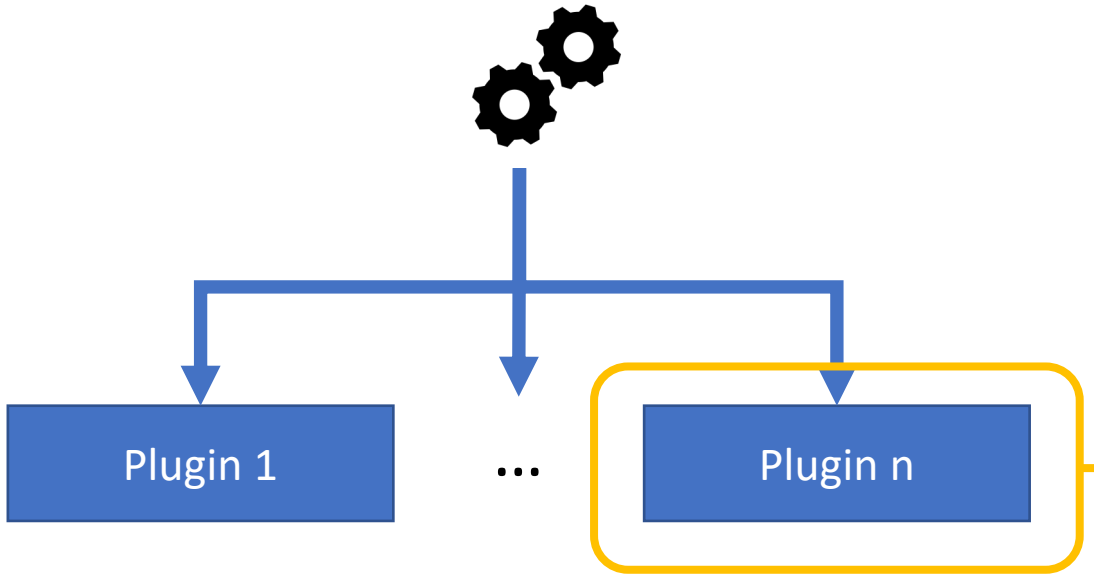
5 + 8



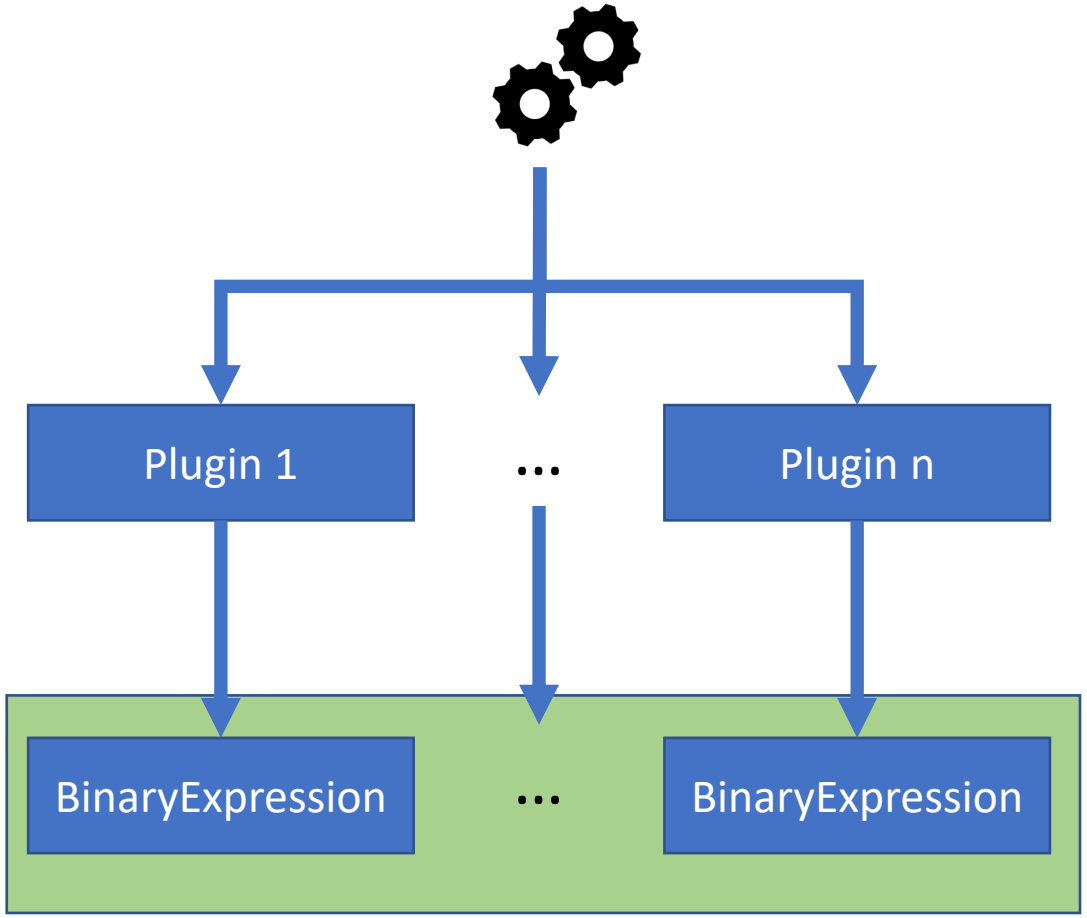


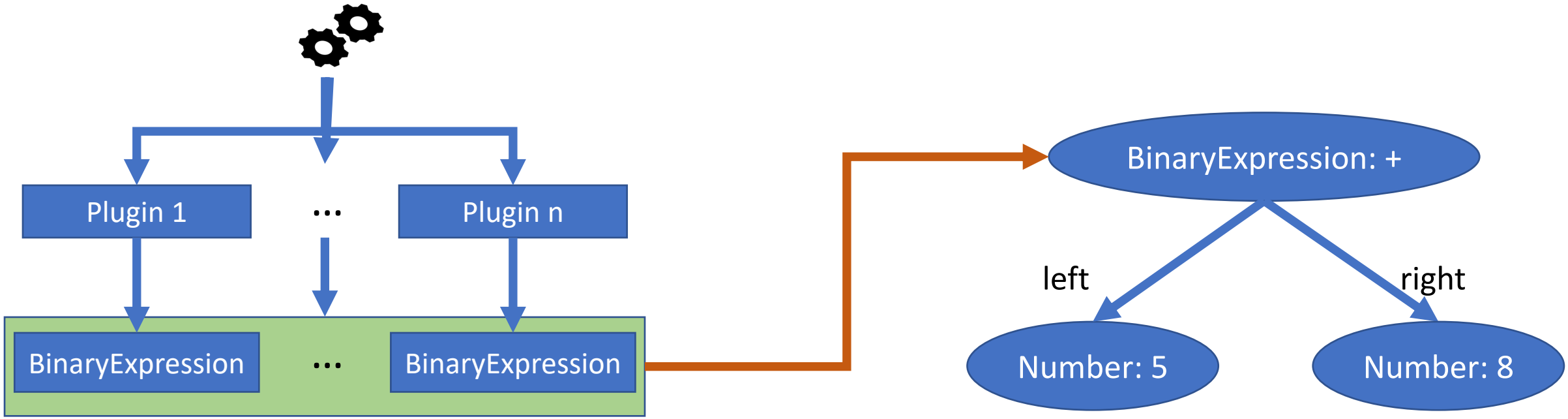

configure
parse
transform

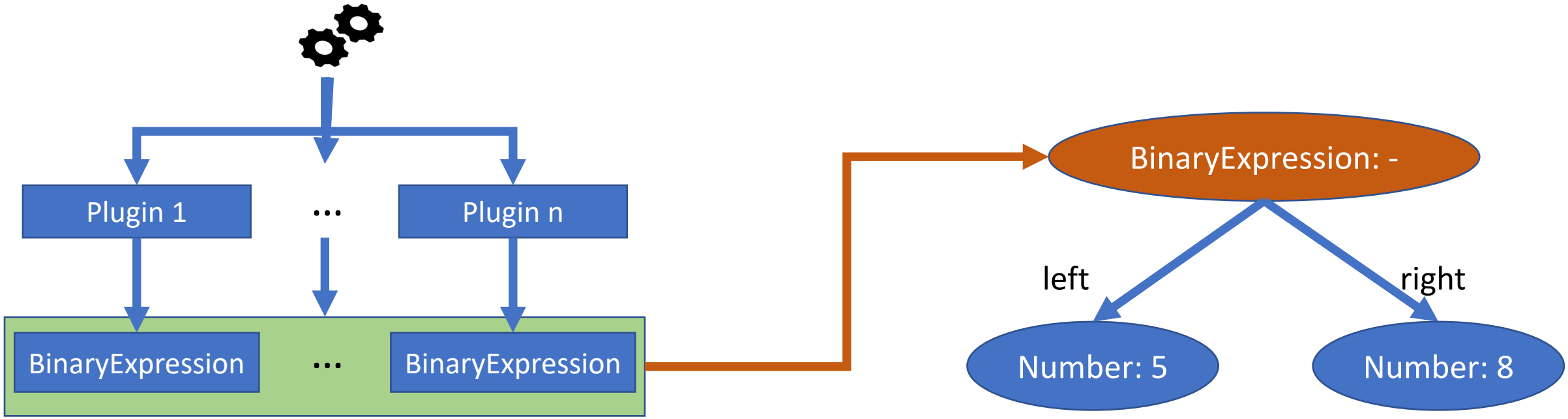

Plugin 1
...
Plugin n

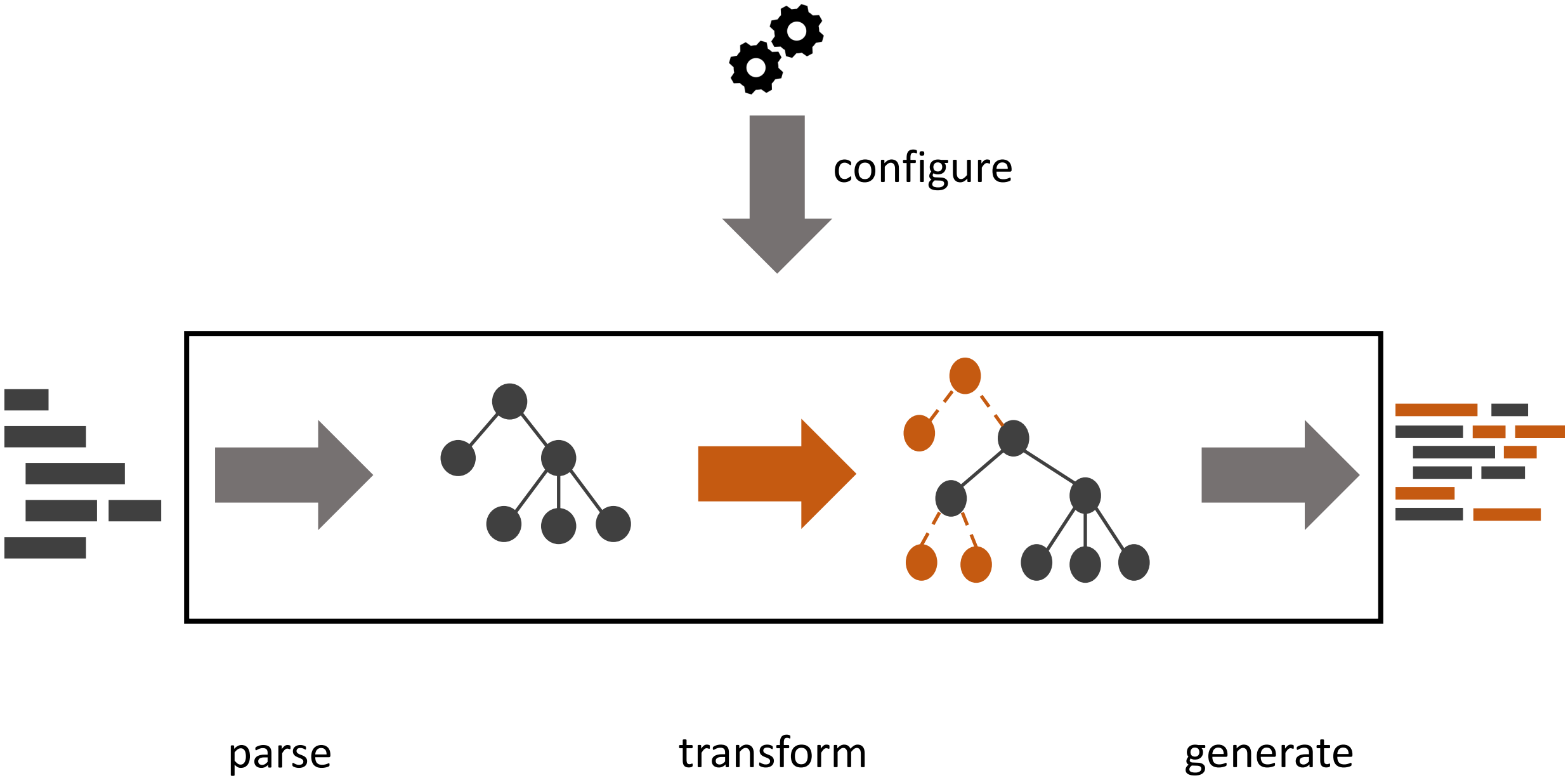


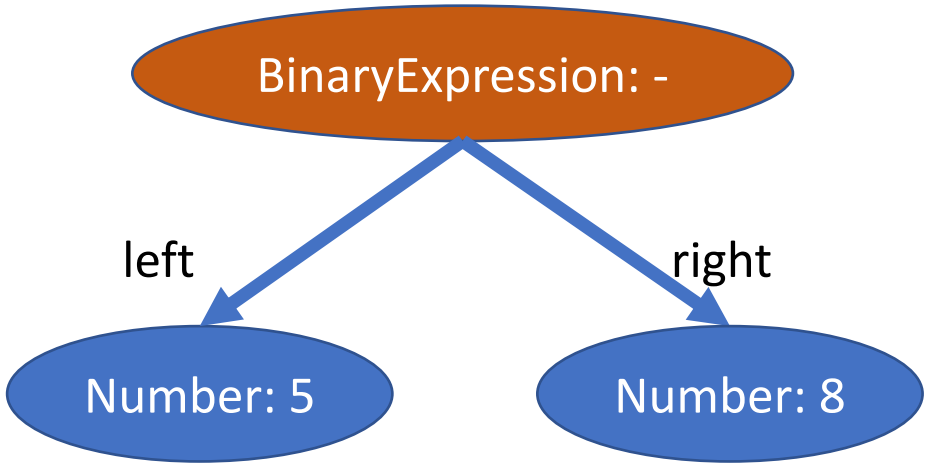

```
name: ‚plugin-name‘

visitor: {
   astNodeName(node) {
      // do something

      node.insertBefore(newNode)
   },
   // …
}
```


Plugin 1
...
Plugin n
BinaryExpression
...
BinaryExpression


Plugin 1
...
Plugin n
BinaryExpression
...
BinaryExpression
BinaryExpression: +
left
right
Number: 5
Number: 8


Plugin 1
...
Plugin n
BinaryExpression
...
BinaryExpression
BinaryExpression: -
left
right
Number: 5
Number: 8


configure
parse
transform
generate



5 - 8

Problems?

Problems

- Unexpected AST-combinations
- Insertion of unexpected AST-nodes
- Many assumptions

Consequences :

- Multiple level of error
- Can influence large amounts of code
- Sometimes only very specific code affected
- Sometimes certain plugin combinations are not possible

=> Debugging needed (Scope: Babel)

Native debugging

Printf debugging	Breakpoints + Stepping
+ Persistent information + Only selected values	+ Follows program flow + State is exlorable + State changeable
- Only selected values - No interactivity	- Step over important instructions - No abstractions, only JavaScript - Visualization cumbersome

Requirements

General:

- Save program flow and state
- Explore saved program flow and state
- Interactively change the state

Babel-spezific:

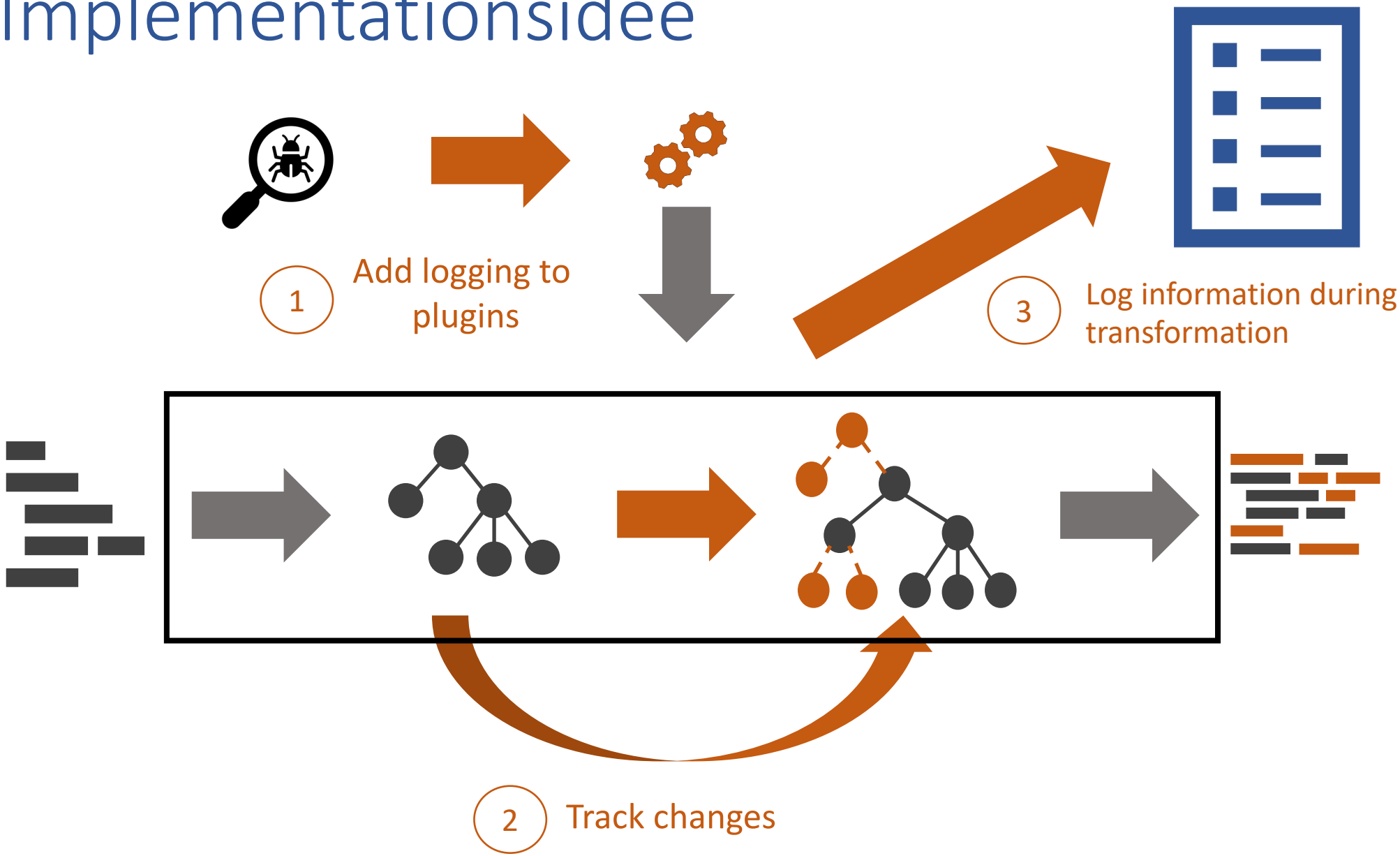
- Record only plugins
- Make AST-changes visible

Approach

Idee

- Record the complete execution of the applied plugins
- Of interest are:
  - What code was executed
  - Which state existed during the execution
  - AST-changes
- Visualize record and make it exploreable

Implementationsidee



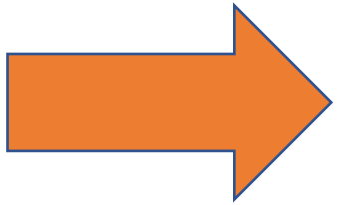
Implementation: Logging

- Transform Babel-plugins so that:
  - Relevant instructions are recorded
  - Positions of instructions are recorded
  - Relevant state is recorded

- Currently tracked: conditions; for loops; assignments; functions; function calls; return

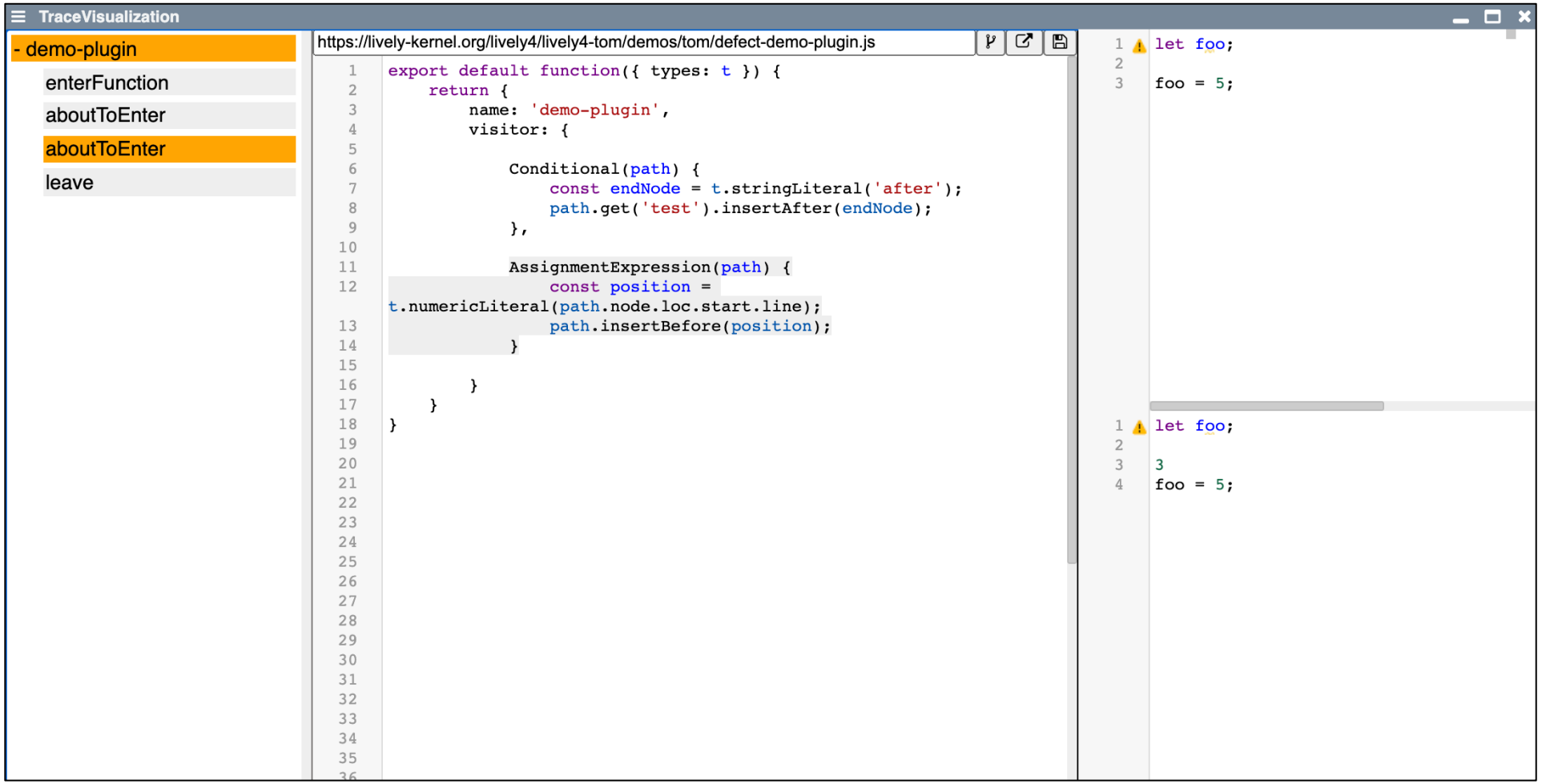
Example: conditions

```
if (node.modified) {

}
```



```
trace.beginCondition(1, „IfStatement“)

if (trace.conditionTest(2, node.modified)) { }

trace.endCondition(3)
```

Demo: Visualisierung



Advantages	Challenges
• Complete record in which you can navigate freely • AST-changes visible on high level of detail • Only plugins are recorded	• Very fast huge amounts of data • Performance

Requirements

General:

- Save program flow and state
- Explore saved program flow and state
- Interactively change the state

Babel-spezific:

- Record only plugins
- Make AST-changes visible

Possible next steps

Missing requirements

- Allow interactive state-changes

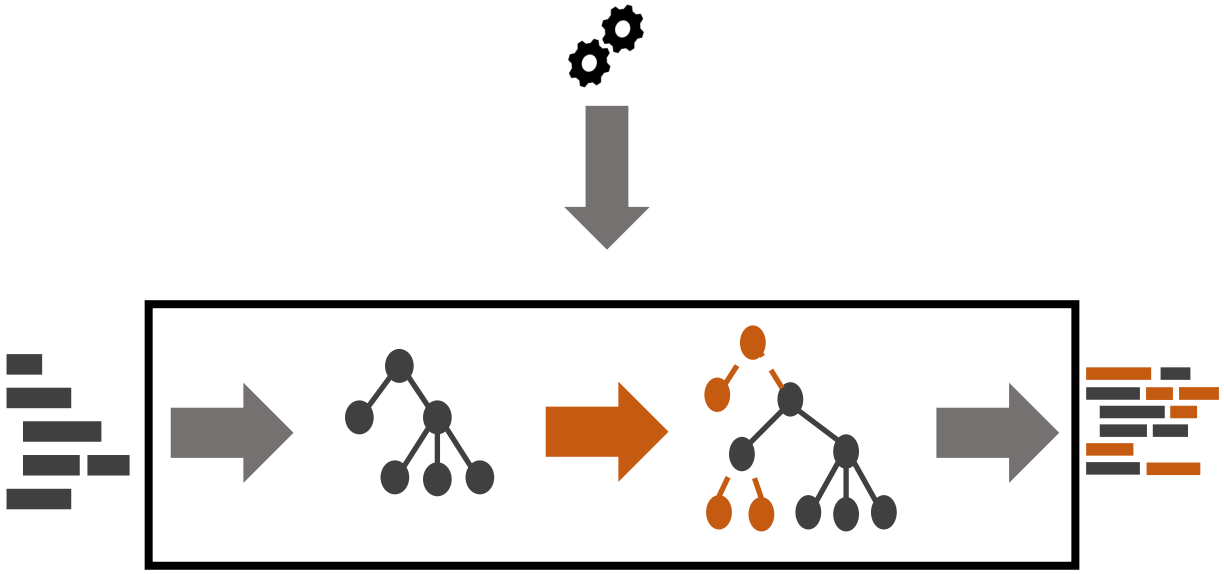
Usability

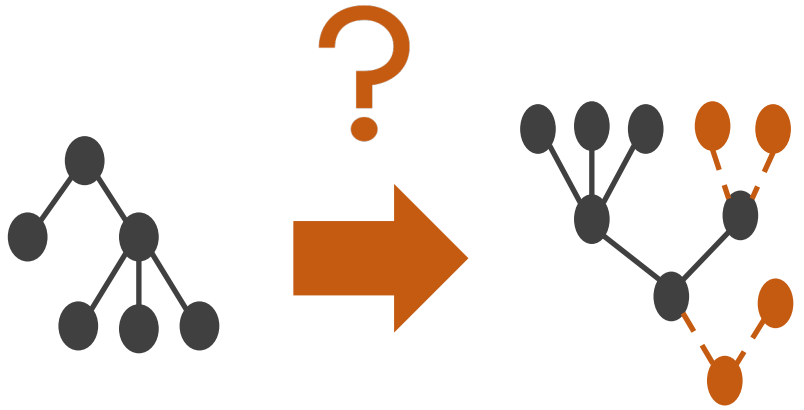
- Query traces

Performace and information quantity

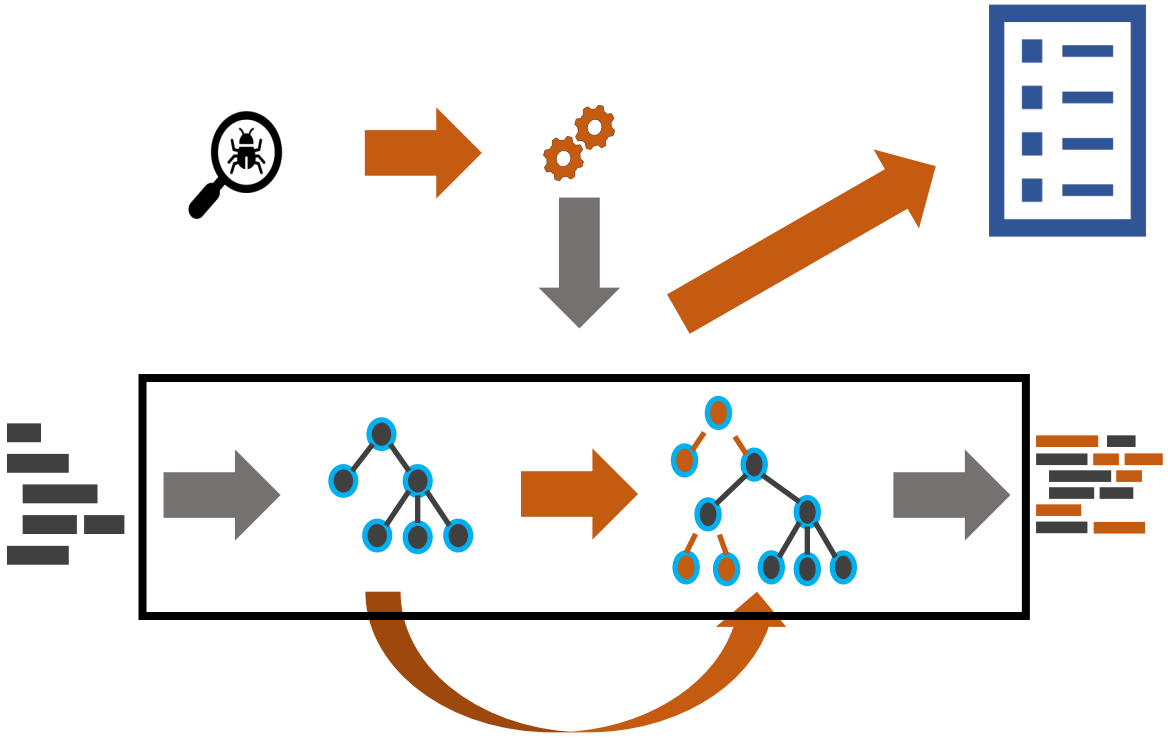
- Build traces incrementally
- Configurable trace accuracy

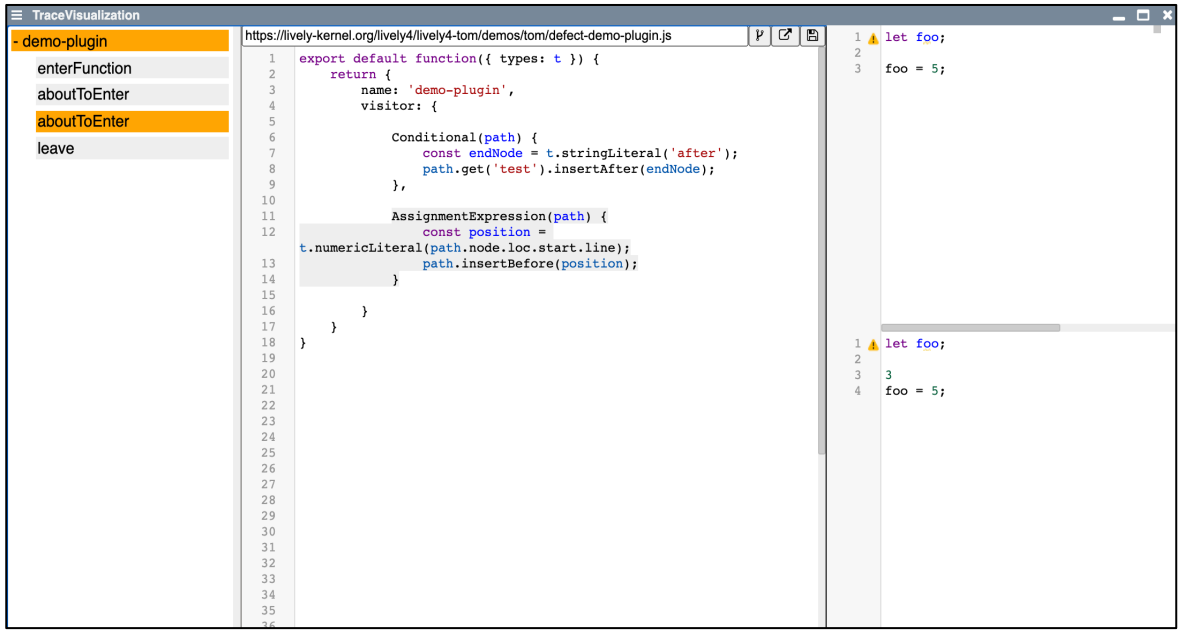
Summary





Summary




TraceVisualization
demo-plugin
enterFunction
aboutToEnter
aboutToEnter
leave
https://lively-kernel.org/lively4/lively4-tom/demos/tom/defect-demo-plugin.js
export default function({ types: t }) {
return {
name: 'demo-plugin',
visitor: {
Conditional(path) {
const endNode = t.stringLiteral('after');
path.get('test').insertAfter(endNode);
},
AssignmentExpression(path) {
const position =
t.numericLiteral(path.node.loc.start.line);
path.insertBefore(position);
}
}
}
}
let foo;
foo = 5;
let foo;
3
foo = 5;