# Provenance-Aware Dynamic Analysis of JavaScript

Laurent Christophe

Promoters:

- Prof. Dr. Coen De Roover
- Prof. Dr. Wolfgang De Meuter

President of the jury: Prof. Dr. Ann Nowé

Jury Members:

- Prof. Dr. Michael Pradel
- Prof. Dr. Tom Van Cutsem
- Prof. Dr. Elisa Gonzalez Boix
- Prof. Dr. Jens Nicolay

SOFTWARE LANGUAGES LAB

27th January 2026



VRIJE UNIVERSITEIT BRUSSEL

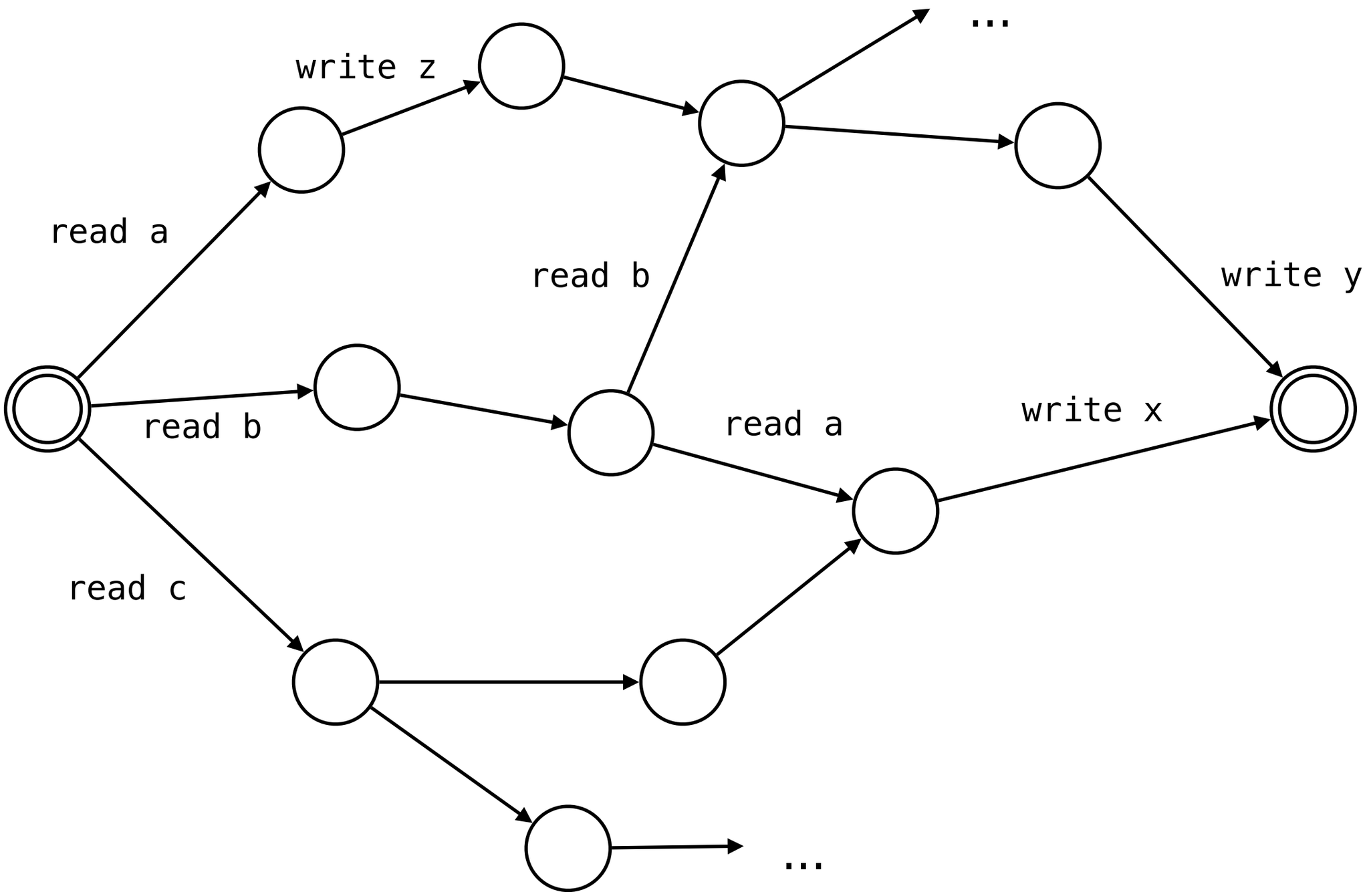
# Roadmap of the Presentation

- Background: Dynamic program analysis
    - Chapter 2: Dynamic program analysis and its challenges for managed languages
- Part #1: Dealing with non-essential syntactic complexity
    - Chapter 3: AranLang: a core variant of JavaScript
    - Chapter 4: Aran: instrumentation of JavaScript programs
- Part #2: Tracking the provenance of runtime values
    - Chapter 5: Transparent value promotion for tracking provenance
    - Chapter 6: Linvail: sound provenance tracking for JavaScript
- Part #3: Orchestrating analyses for distributed applications
    - Chapter 7: Orchestration of dynamic analysis for distributed applications
- Conclusion

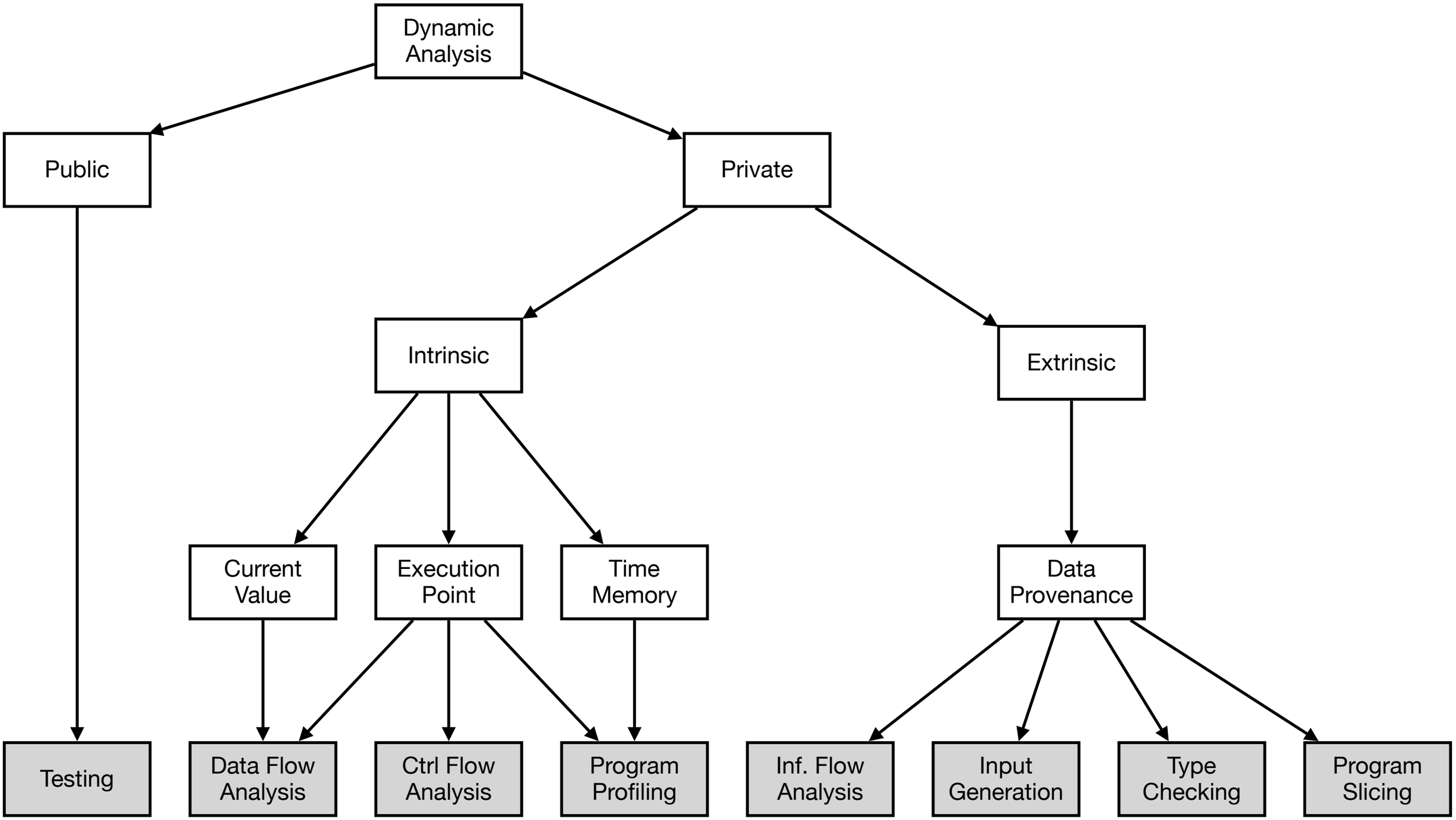
# Background: Dynamic Program Analysis

- Chapter 2: Dynamic program analysis and its challenges for managed languages

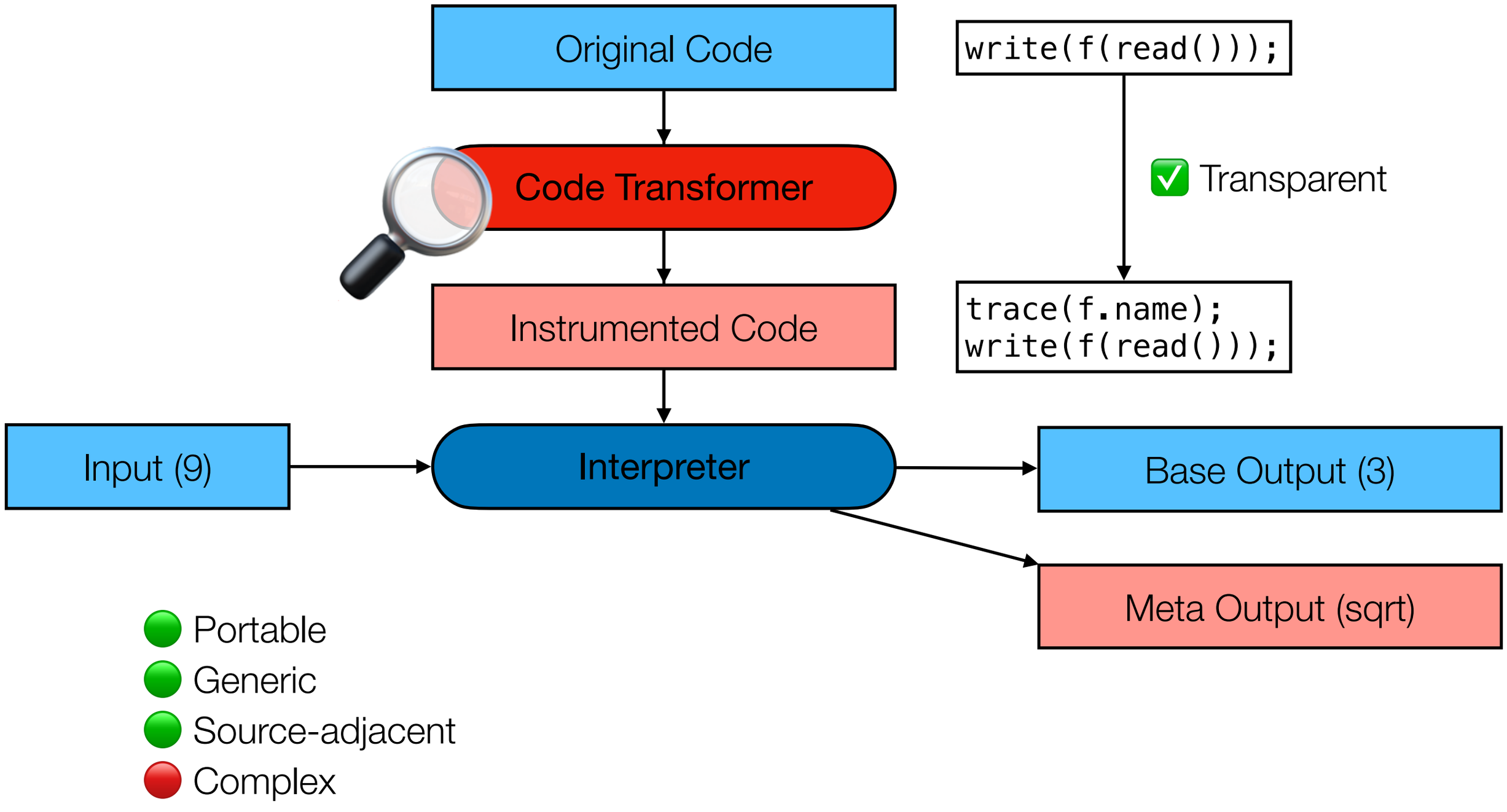
# Introduction to Dynamic Program Analysis



# Categorisation of Dynamic Program Analysis



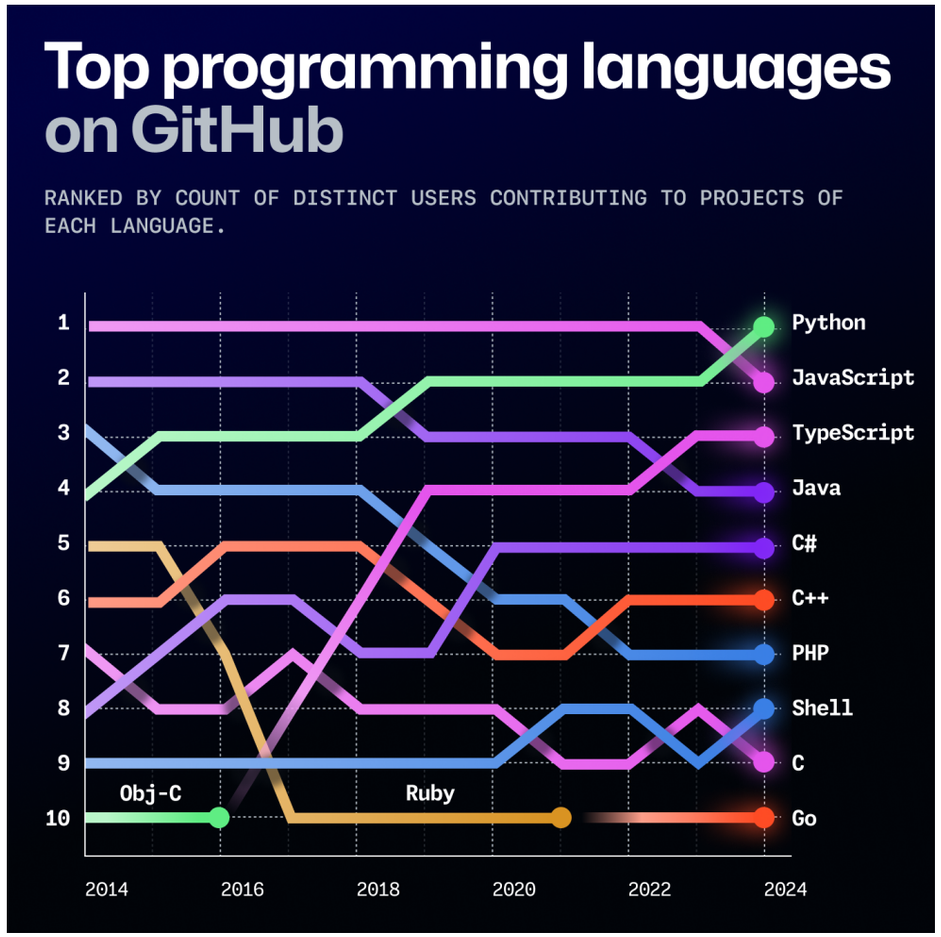
# Source Code Instrumentation

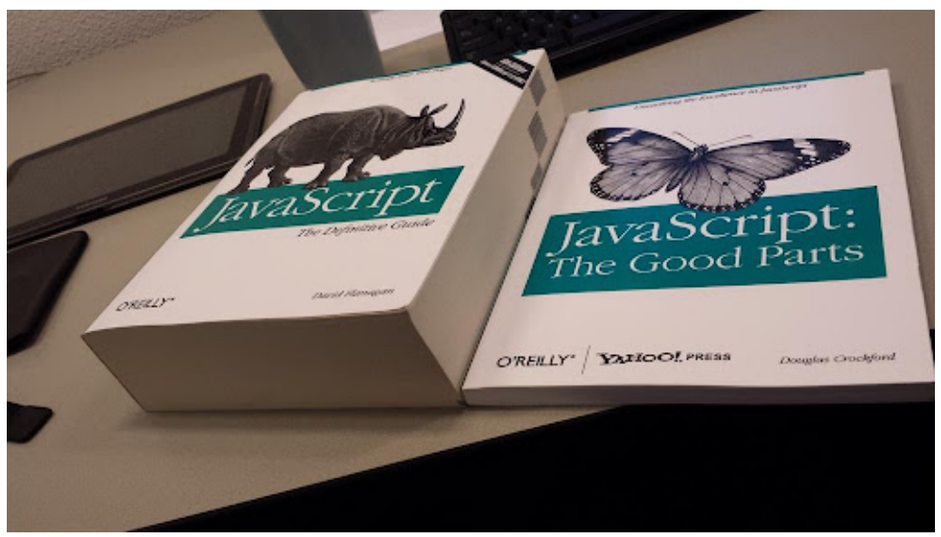


# Part #1:
# Dealing with Non-Essential Syntactic Complexity

- Chapter 3: AranLang: a core variant of JavaScript
- Chapter 4: Aran: instrumentation of JavaScript programs

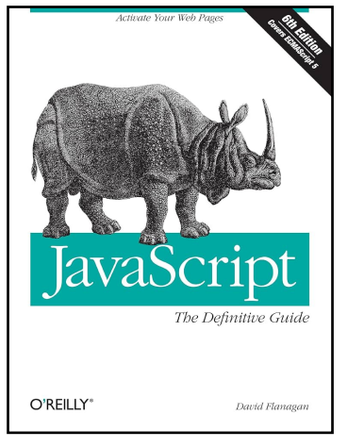
# Choosing JavaScript

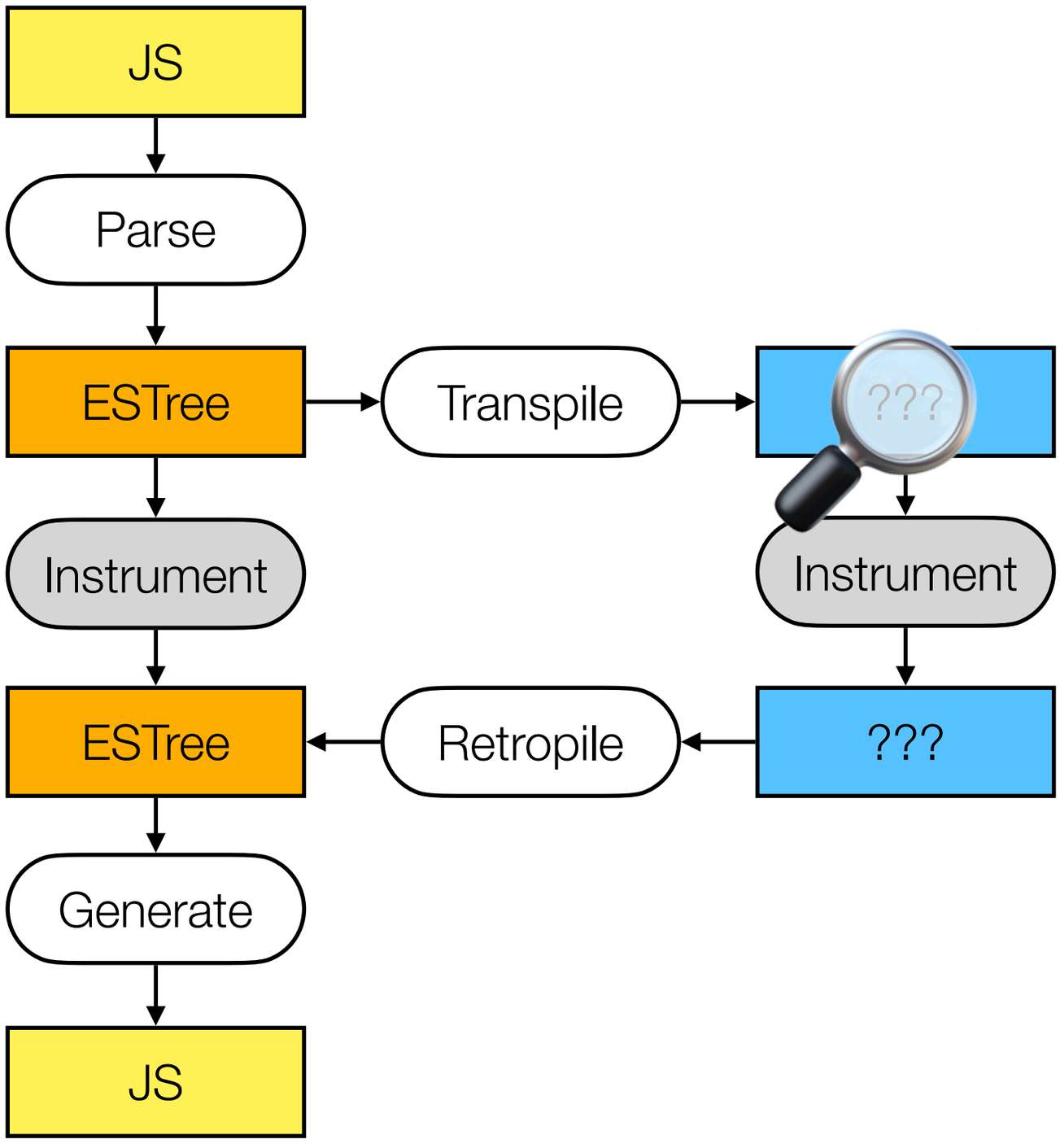


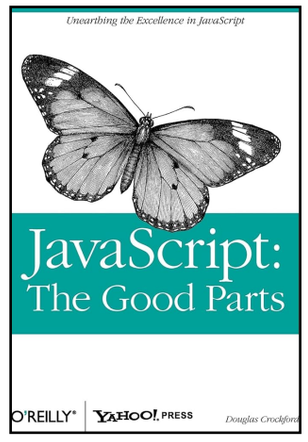




| Name | Publ. Date | New Language Features |
|---|---|---|
| ES1 | June 1997 | First edition |
| ES2 | August 1998 | Editorial changes |
| ES3 | December 1999 | - Regular expressions<br>- Exception handling |
| ES4 | Abandoned (July 2008) | Ambitious modifications;<br>some were completely dropped;<br>some were included in ES2015. |
| ES5 | December 2009 | - Strict mode<br>- Property accessors |
| ES5.1 | June 2011 | Editorial changes |
| ES2015 / harmony | June 2015 | - Classes<br>- Native modules<br>- Block scoping<br>- Proxy and Reflect<br>- Destructuring assignments<br>- Arrow functions<br>- Iteration and Promise protocols<br>- Generator functions |
| ES2016 | June 2016 | Few minor features |
| ES2017 | June 2017 | - Asynchronous functions<br>- Shared memory |
| ES2018 | June 2018 | - Asynchronous iteration<br>- Rest/Spread properties |
| ES2019 | June 2019 | - Optional binding in catch |
| ES2020 | June 2020 | - Dynamic import<br>- import.meta<br>- BigInt<br>- Optional chaining<br>- Nullish coalescing |
| ES2021 | June 2021 | - Weak reference<br>- Logical assignment |
| ES2022 | June 2022 | - Top level await<br>- Class fields<br>- Private fields and methods |
| ES2023 | June 2023 | Few minor features. |
| ES2024 | June 2024 | Few minor features. |
| ES2025 | June 2025 | - Non-JS import<br>- RegExp inline flags |

# Instrumenting JavaScript



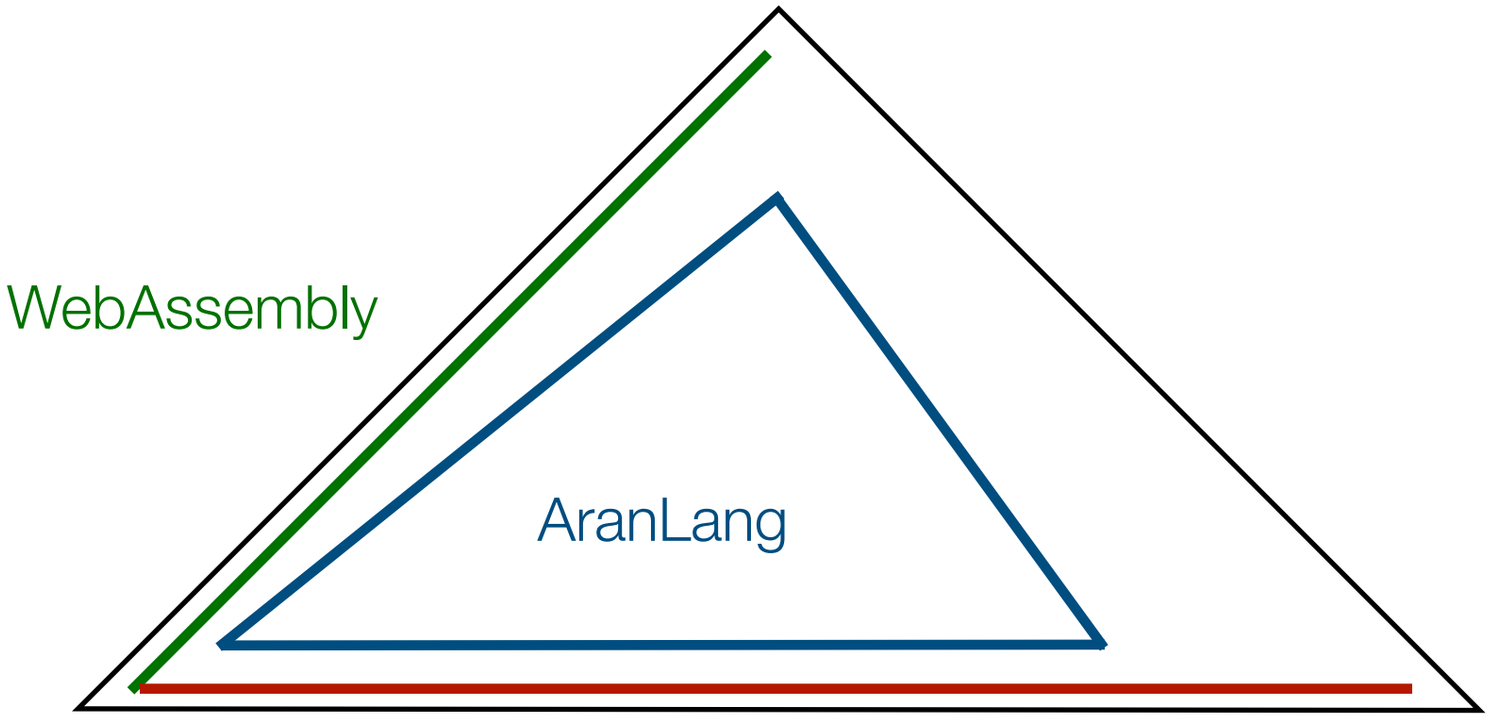




ESTree:

- 76 node types
- 🔴 Complex

# AranLang: Design Objectives

📌 Minimalism
🎯 Facilitate instrumentation
⬆️ Enhance simplicity




📌 Coverage
🎯 Able to represent source semantics
➡️ Preserve semantic transparency

📌 Proximity
🎯 Control transpilation expansion
➡️ Preserve source adjacency

AranLang: coverage > minimalism > proximity

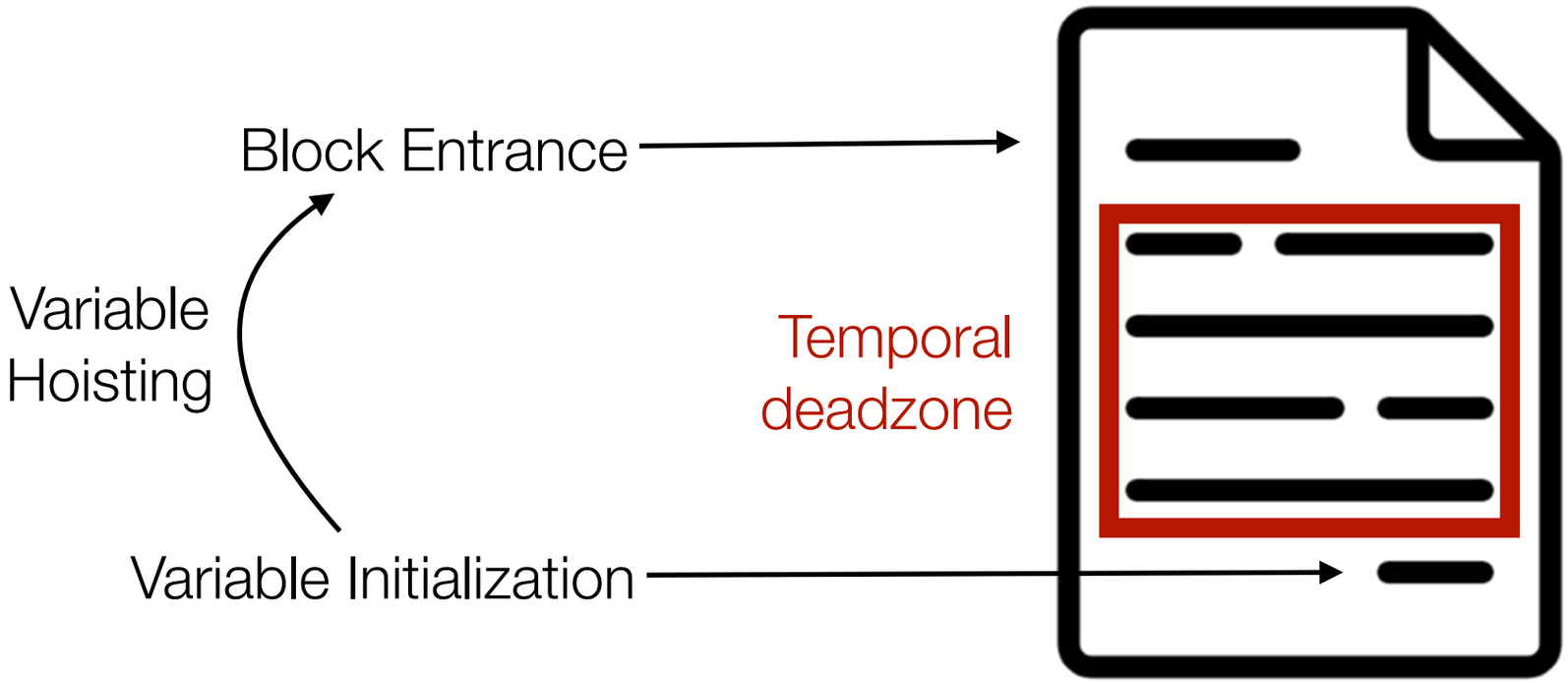
# AranLang: Syntax

34 simple AST node types
(55% less than ESTree)

| | |
|---|---|
| Primitive | JsonLiteral \| BigIntLiteral |
| InternalLabel | Identifier (. Identifier)*<br>! JavaScriptKeyword |
| InternalVariable | Identifier (. Identifier)*<br>! JavaScriptKeyword<br>! ImplicitParameter |
| ExternalVariable | Identifier<br>! JavaScriptKeyword |
| Intrinsic | % Identifier (. Identifier)* (@get\|@set)? % |
| Specifier | SingleQuoteStringLiteral |
| Source | SingleQuoteStringLiteral |
| ModuleHeader | ImportHeader \| ExportHeader \| AggregateHeader |
| ImportHeader | import (Specifier \| *) from Source ; |
| ExportHeader | export Specifier ; |
| AggregateHeader | export * from Source ; |
| AggregateHeader | export * as Specifier from Source ; |
| AggregateHeader | export Specifier as Specifier from Source ; |
| Program | ModuleProgram \| ... \| DeepLocalEvalProgram |
| ModuleProgram | "module" ; ModuleHeader* RoutineBlock |
| ScriptProgram | "script" ; ((var \| let) ExternalVariable ;)* RoutineBlock |
| GlobalEvalProgram | "eval-global" ; (var ExternalVariable ;)* RoutineBlock |
| RootLocalEvalProgram | "eval-local-root" ; (var ExternalVariable ;)* RoutineBlock |
| DeepLocalEvalProgram | "eval-local-deep" ; RoutineBlock |
| Declaration | let InternalVariable = Intrinsic ;<br>let Variable ; |
| SegmentBlock | (InternalLabel :)* { Declaration* Statement* } |
| RoutineBlock | { Declaration* Statement* return Expression ; } |
| GeneratorBlock | { Declaration*<br>([Effect (, Effect)* ] ;)?<br>Statement*<br>return Expression ; } |

| | |
|---|---|
| Statement | EffectStatement \| ... \| TryStatement |
| EffectStatement | Effect ; |
| BreakStatement | break InternalLabel ; |
| DebuggerStatement | debugger ; |
| BlockStatement | SegmentBlock |
| IfStatement | if ( Expression ) SegmentBlock else SegmentBlock |
| WhileStatement | while ( Expression ) SegmentBlock |
| TryStatement | try SegmentBlock catch SegmentBlock finally SegmentBlock |
| Effect | ExpressionEffect \| ... \| WriteEffect |
| ExpressionEffect | Expression |
| ConditionalEffect | Expression ? ( Effect* ) : ( Effect* ) |
| ExportEffect | export Specifier = Expression |
| WriteEffect | (InternalVariable \| ImplicitParameter) = Expression |
| Expression | PrimitiveExpression \| ... \| ConstructExpression |
| PrimitiveExpression | Primitive |
| IntrinsicExpression | Intrinsic |
| ReadExpression | InternalVariable \| ImplicitParameter |
| ImportExpression | import (Specifier \| *) from Source |
| ArrowExpression | async? arrow RoutineBlock |
| FunctionExpression | async? function RoutineBlock |
| MethodExpression | async? method RoutineBlock |
| GeneratorExpression | async? generator GeneratorBlock |
| AwaitExpression | await Expression |
| YieldExpression | yield *? Expression |
| SequenceExpression | ( (Effect ,)+ Expression ) |
| ConditionalExpression | Expression ? Expression : Expression |
| EvalExpression | eval Expression |
| ApplyExpression | Expression ( ) |
| ApplyExpression | Expression ( Expression (, Expression)* ) |
| ApplyExpression | Expression ( that Expression (, Expression)*) |
| ConstructExpression | new Expression ( ) |
| ConstructExpression | new Expression ( Expression (, Expression)* ) |

# JavaScript: Temporal Dead Zone




JS

```
{
  try { f(); } catch {}
  let x = 42;
  x; ✅
  function f () {
    return x; ⛔ / ✅
  }
  f();
}
```

# AranLang: Transpiling TDZ

JS

```
{
  try { f(); } catch {}
  let x = 42;
  x; ✅
  function f() {
    return x; ⛔ / ✅
  }
  f();
}
```

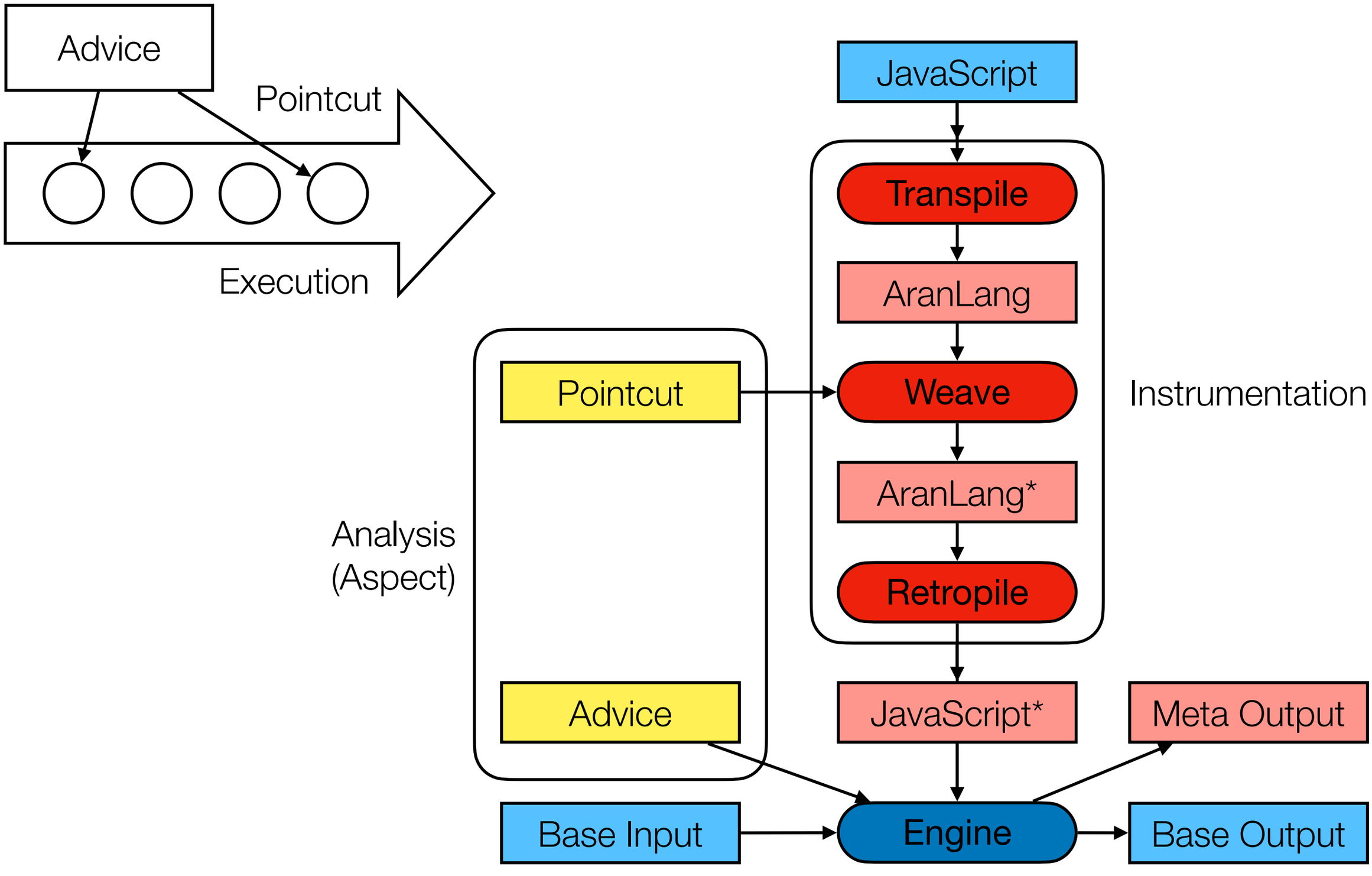
AL

```
{
  let x = %aran.deadzone_symbol%,
      f = %undefined%;
  f = function {
    return x === %aran.deadzone_symbol%
      ? %aran.throwException%(⛔
          new %ReferenceError%("x in TDZ"),
        )
      : x; ✅
  }
  try { f(); } catch {} finally {}
  x = 42;
  x; ✅
  f();
}
```

⬆️ Simplify instrumentation (because more explicit)

➡️ Still source-adjacent

# Aran: Aspect-Oriented API



# Aran: Tracing the Callstack

target.js

```
1 const fac = (n) =>
2   n == 1 ? 1 : n * fac(n - 1)
3 write(fac(read()))
```

pointcut.json

```
["apply@around"]
```

Aran

```
globalThis.__ADVICE__ = {
  "apply@around": (_s, f, t, xs, l) => {
    log(">> ", f.name, "(", xs, ") at ", l);
    const y = Reflect.apply(f, t, xs);
    log("<< ", y);
    return y;
  },
};
```

advice.js

```
>> read() at 3:10
<< 3
>> fac(3) at 3:6
   >> binary("-", 3, 1) at 2:23
   << 2
   >> fac(2) at 2:19
      >> binary("-", 2, 1) at 2:23
      << 1
      >> fac(1) at 2:19
      << 1
      >> binary("*", 2, 1) at 2:15
      << 2
   >> binary("*", 3, 2) at 2:15
   << 6
<< 6
>> write(6) at 3:0
<< undefined
```

3 → Engine → 6

# Aran: Join Point Model

`primitive@after`
`intrinsic@after`
`closure@after`
`import@after`
`read@after`

Producer (5)

`drop@before`
`test@before`
`eval@before`
`write@before`
`export@before`

Consumer (5)

`yield@before`
`yield@after`

`await@before`
`await@after`

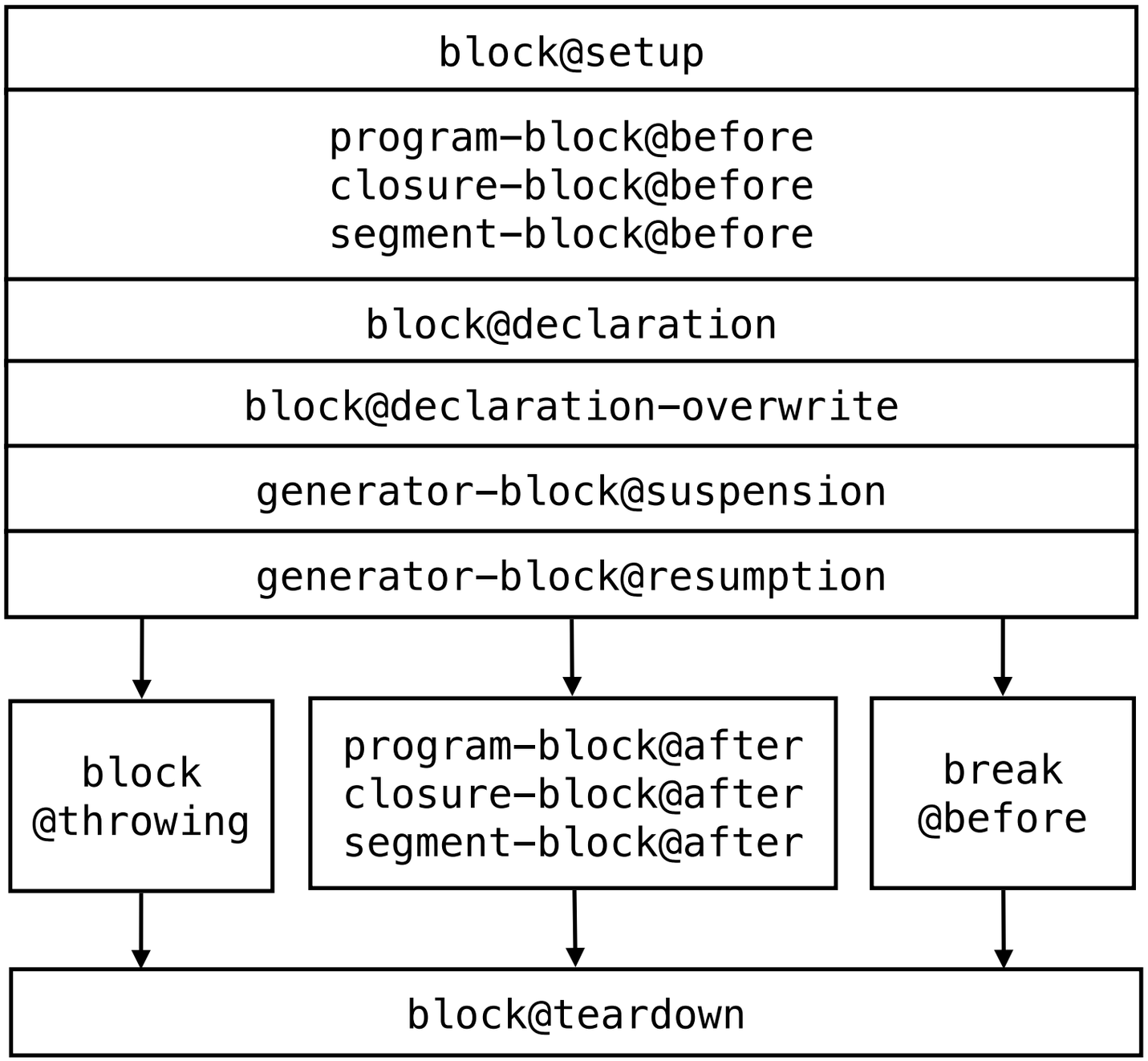
Hiatus (4)

`apply@around`

`construct@around`

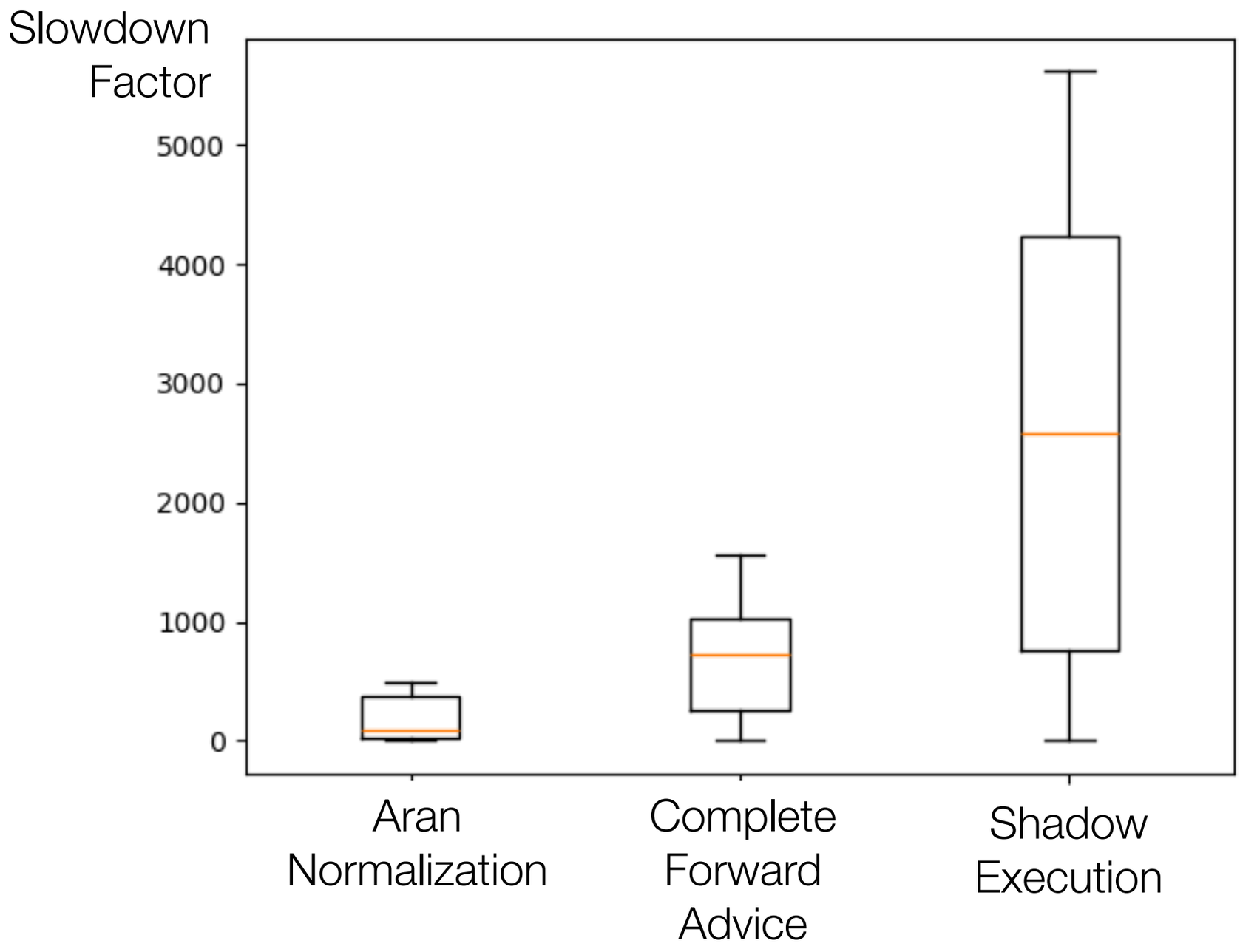
Combiner (2)

- 🟡 Intermediary simplicity
  (31 join points)
- ➡️ Still generic
  (support shadow execution)




Block-related (15)

# Aran: Performance Overhead

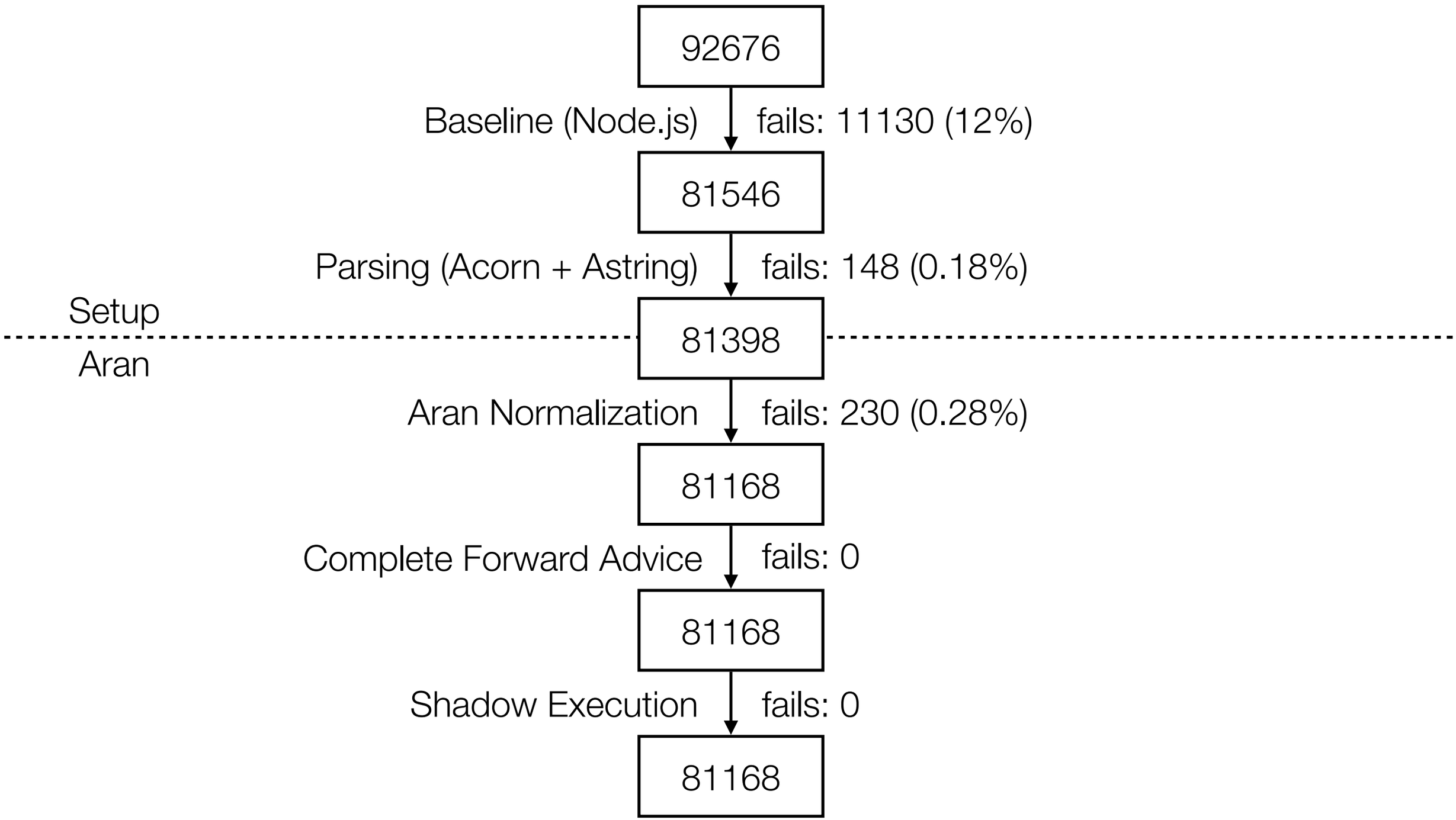



- Low performance transparency (10X - 1000sX slowdown)



https://github.com/chromium/octane

# Aran: Semantic Overhead (1/2)




- High semantic transparency (> 99.7%)



https://github.com/tc39/test262

# Aran: Semantic Overhead (2/2)

Corpus of 81398 Test262 cases (post: Node.js + Acorn + Astring)

| Discrepancy Tag | Discrepancy Count |
|---|---|
| `missing-iterable-return-in-pattern` | 84 (36%) |
| `function-string-representation` | 68 (30%) |
| `arguments-two-way-binding` | 32 (14%) |
| `early-declaration` | 20 (9%) |
| `async-iterator-async-value` | 14 (6%) |
| `wrong-realm-for-default-prototype` | 6 (3%) |
| `function-dynamic-property` | 2 (1%) |
| `duplicate-constant-global-function` | 2 (1%) |
| `duplicate-super-prototype-access` | 2 (1%) |
| **Total** | 230 |

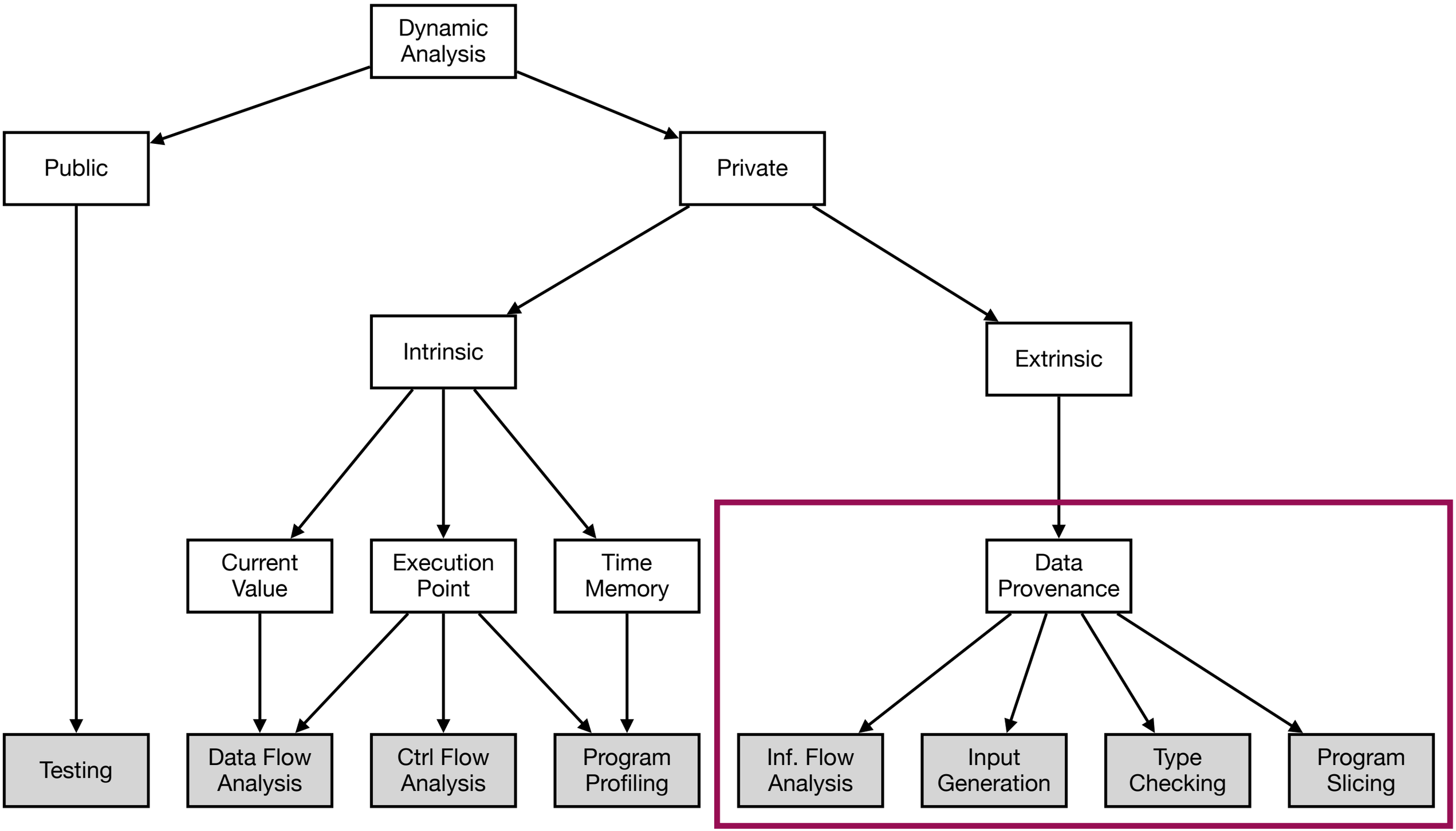


(< 0.3% discrepancy rate)

# Part #2:
# Tracking Value Provenance

- Chapter 5: Transparent value promotion for tracking provenance
- Chapter 6: Linvail: sound provenance tracking for JavaScript

# Provenance-Aware Program Analysis



# Motivation: Static Computations

```
const statics = [];
export const advice = {
  "primitive@after": (_state, primitive, _location) => {
    statics.push(primitive);
    return primitive;
  },
  "apply@around": (_state, callee, that, input, location) => {
    const output = Reflect.apply(callee, that, input);
    if (input.every((arg) => statics.some((item) => item ==== arg))) {
      console.log("AOT " + output + " at " + location);
      statics.push(output);
    }
    return output;
  },
};
```

PJS

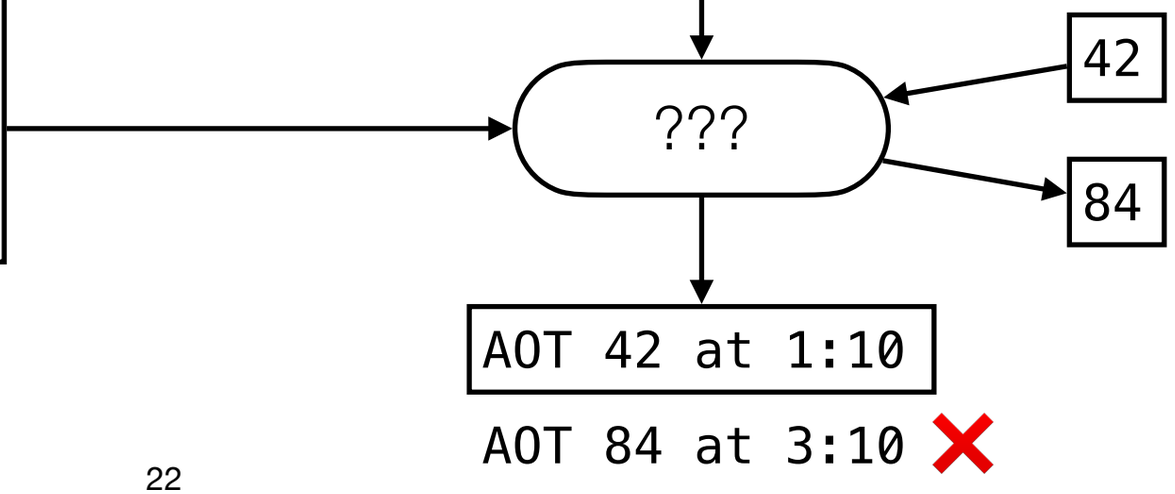
===❌

advice.pjs

```
1 const offset = 40 + 2;
2 const input = read();
3 const output = 2 * input;
4 write(output)
```

PJS ❌

target.pjs



# Requirement: Provenancial Equality

```
const num = 42;
// Basic provenance tracking //
assert(num ==== num);
assert(num !=== 42);
// Inter-procedural provenance tracking //
const identity = (arg) => arg;
assert(identity(num) ==== num);
assert(identity(num) !=== 42);
// Provenance tracking through objects //
const object = { key: num };
assert(object.key ==== num);
assert(object.key !=== 42);
// Provenance tracking through methods //
const array = [num];
assert(array.some((item) => item ==== num));
assert(array.every((item) => item !=== 42));
```

PJS

$$\forall \sigma \in Store,\ \forall v_1, v_2 \in Val\ :\quad \begin{matrix} Primitive & & Primitive \\ \left(v_1 \overset{ref}{\underset{\sigma}{=\!=}} v_2\right) & \begin{matrix}\Leftarrow \\ \Rightarrow \\ \not\Leftarrow\end{matrix} & \left(v_1 \overset{struct}{\underset{\sigma}{=\!=}} v_2\right) \\ Composite & & Composite \end{matrix}$$

# Challenge: Reverting Value Inlining

PJS

```
const x = 42;
const y = 42;
const z = { k: x };
```

*Store* : *Addr* → *Data*

| | |
|---|---|
| &0 | 42 |
| &1 | 42 |
| &2 | {k -> &0 } |

*Env* : *Var* → *Val*

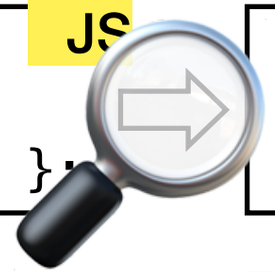
| | |
|---|---|
| x | &0 |
| y | &1 |
| z | &2 |

$Val := Addr$

Not portable

JS

```
const x = 42;
const y = 42;
const z = { k: x };
```



*Store* : *Addr* → *Data*

| | |
|---|---|
| &0 | {k -> 42 } |

*Env* : *Var* → *Val*

| | |
|---|---|
| x | 42 |
| y | 42 |
| z | &0 |

$Val := Prim \cup Addr$

JS

???

*Store* : *Addr* → *Data*

| | |
|---|---|
| &0 | {_ -> 42 } |
| &1 | {_ -> 42 } |
| &2 | {k -> &0 } |

*Env* : *Var* → *Val*

| | |
|---|---|
| x | &0 |
| y | &1 |
| z | &2 |

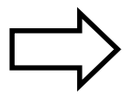
$Val := Prim \cup Addr$

Portable

# Partition of Run-time Values

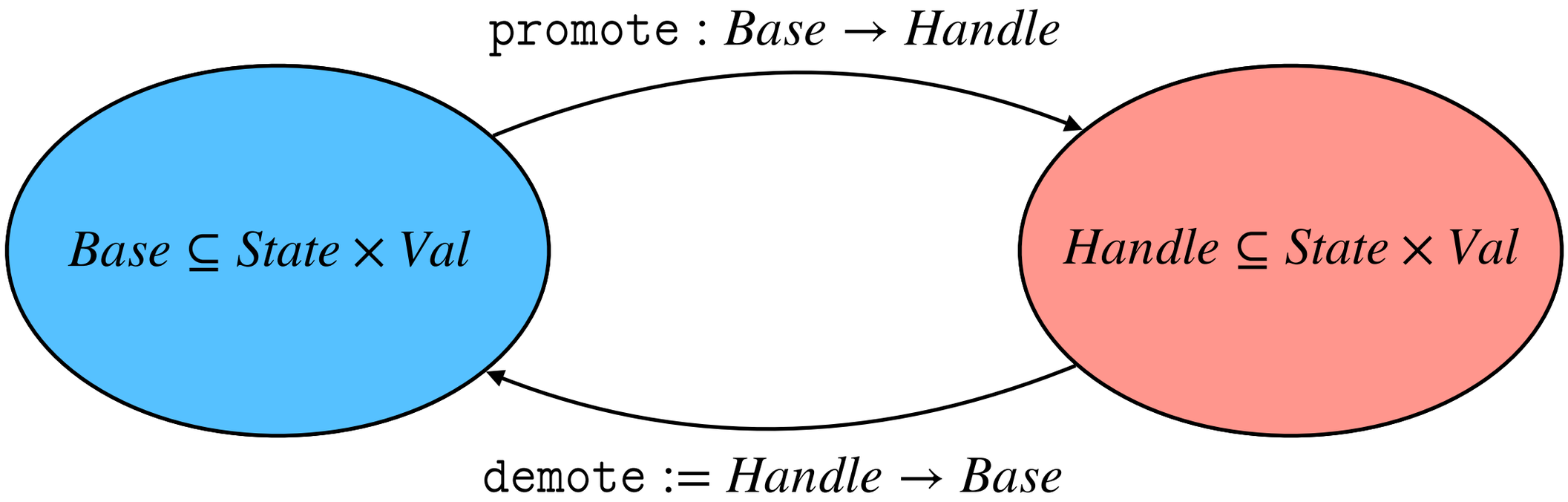
PJS

```
const num = 42;
assert(num ==== num);
assert(num !===  42);
log(num);
```

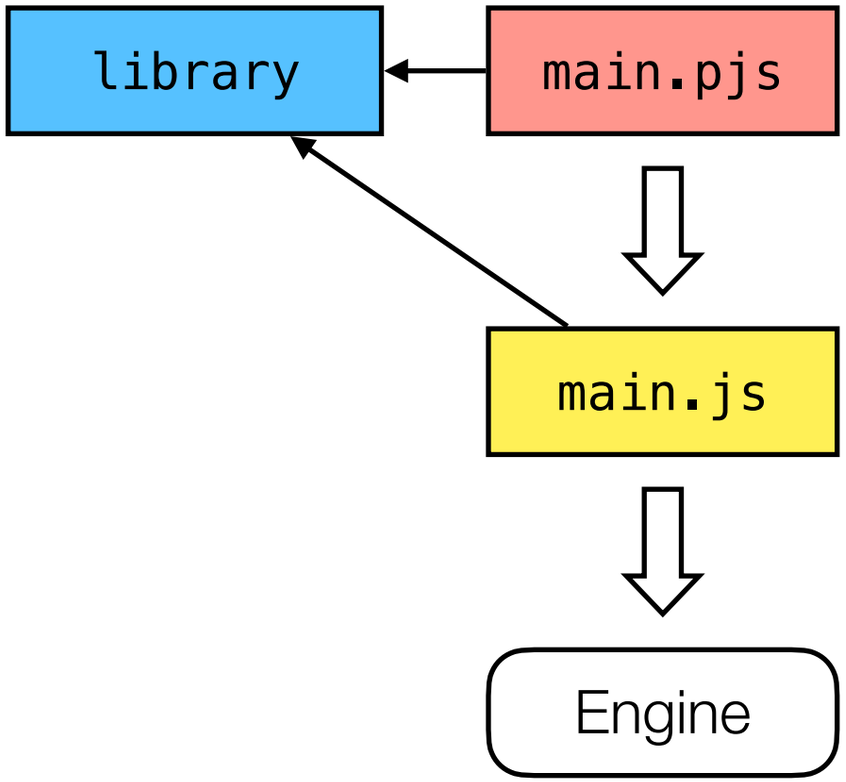


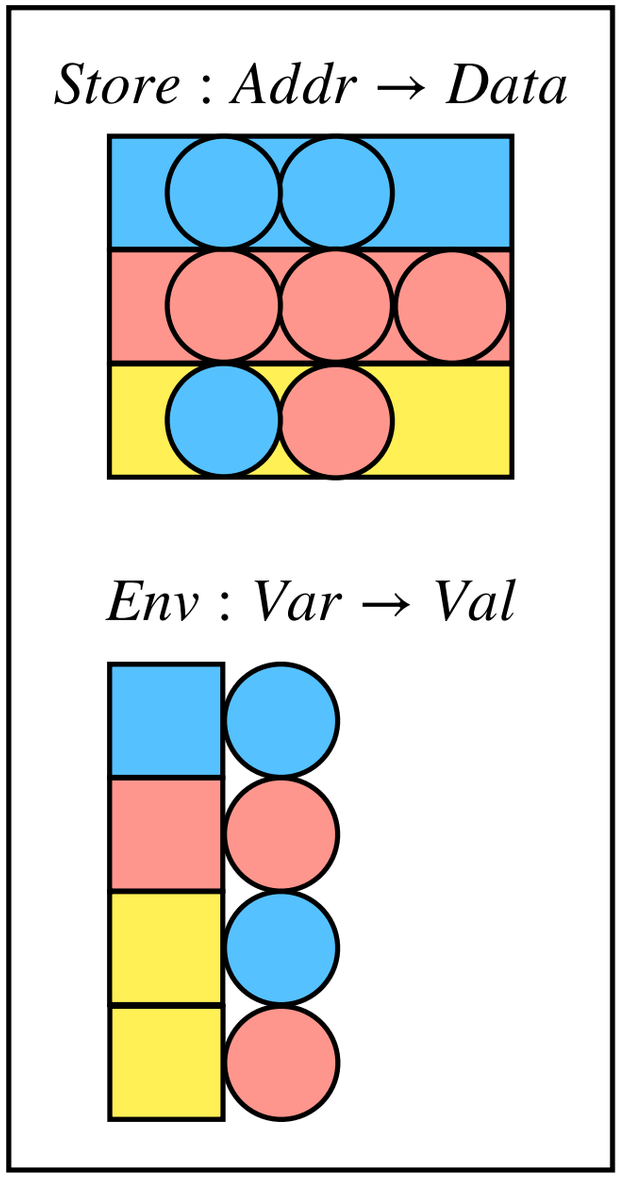
JS

```
const promote = (inner) => ({ inner });
const demote = ({ inner }) => inner;
const provEq = (h1, h2) => h1 === h2;
const num = promote(42);
assert(provEq(num, num));
assert(!provEq(num, promote(42));
log(demote(num)); // 42 ✅
```



# Partition of State Locations





$$Val := Prim \cup Addr$$

# Provenance Tracking in Objects

JS

```
export const promote = (inner) => ({ inner });
export const demote = ({ inner }) => inner;
export const provEq = (handle1, handle2) => handle1 === handle2;
export const virtualize = (target) => new Proxy(target, handler);
const handler = {
  get: (tgt, key, rec) => demote(Reflect.get(tgt, key, rec)),
  set: (tgt, key, val, rec) => Reflect.set(tgt, key, promote(val), rec),
};
```

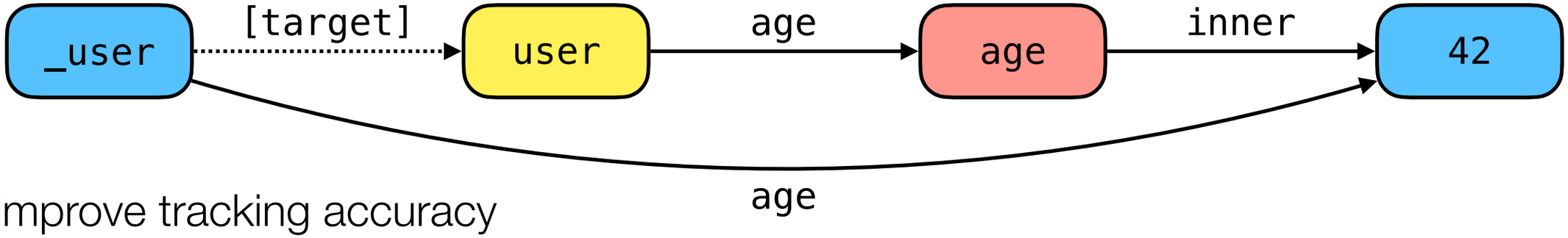
PJS

```
const age = 42;
const user = { age };

assert(user.age ==== age);
log(JSON.stringify(user));
// {"age":42}
```

JS

```
import { promote, virtualize, provEq }
  from "./prov.mjs";
const age = promote(42);
const user = { age };
const _user = virtualize(user);
assert(provEq(user.age, age));
log(JSON.stringify(_user));
```

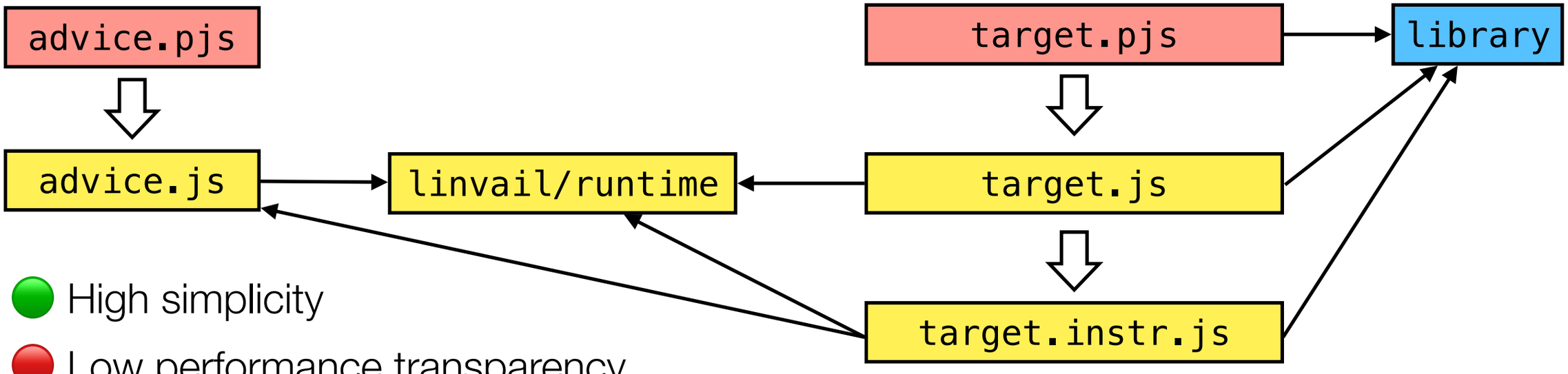



- ⬆️ Improve tracking accuracy
- ➡️ Preserve transparency

# Linvail: Advice Layering

PJS

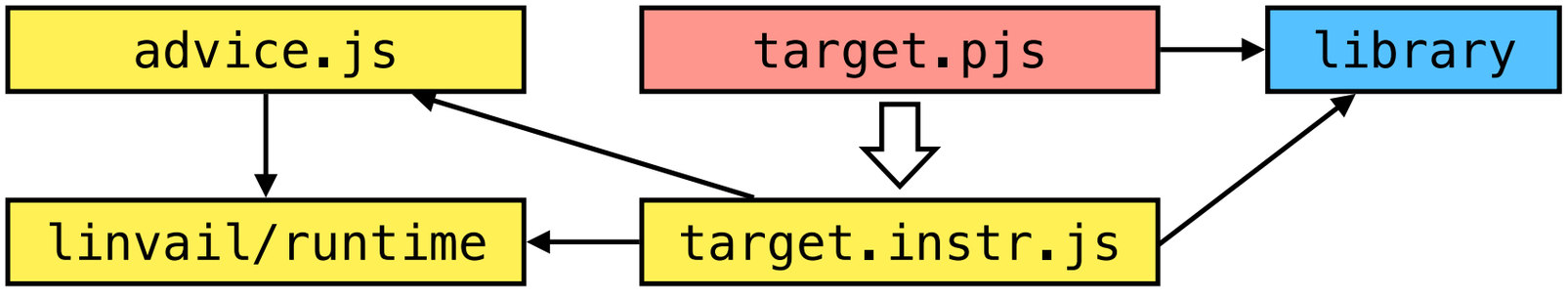
```
const statics = [];
export const advice = {
  "primitive@after": (_state, primitive, _location) => {
    statics.push(primitive);
    return primitive;
  },
  "apply@around": (_state, callee, that, input, location) => {
    const output = Reflect.apply(callee, that, input);
    if (input.every((arg) => statics((item) => item ==== arg))) {
      statics(output);
      console.log("AOT " + output + " at " + location);
    }
    return output;
  }
};
```



# Linvail: Advice Extension

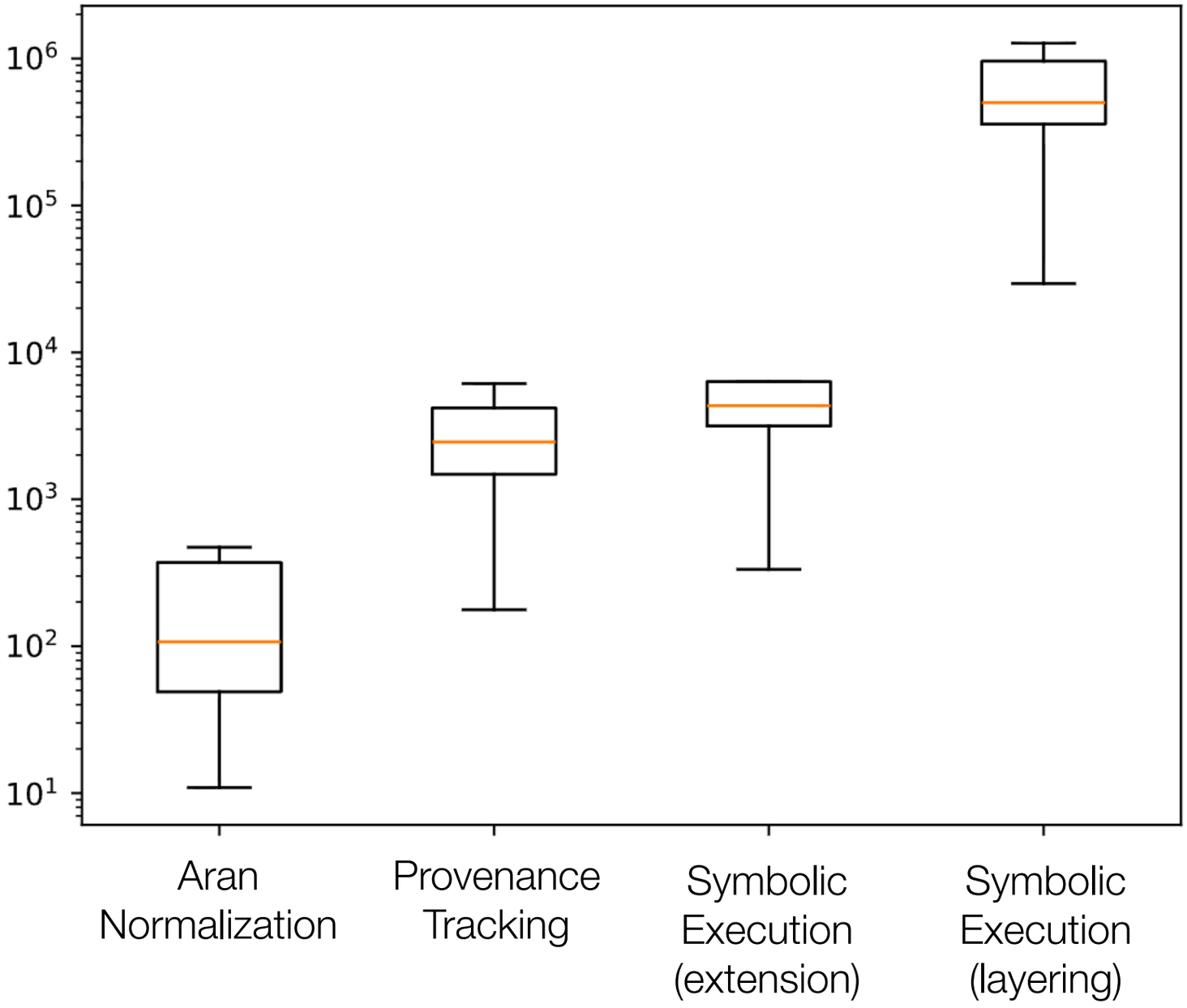
JS

```
import { frontier_advice, promote, demote, apply } from "linvail/runtime";
const statics = [];
export const advice = {
  ...frontier_advice,
  "primitive@after": (_state, primitive, _location) => {
    const handle = promote(primitive);
    statics.push(handle);
    return handle;
  },
  "apply@around": (_state, callee, that, input, location) => {
    const output = apply(callee, that, input);
    if (input.every((arg) => statics.some((item) => item === arg))) {
      statics.push(output);
      console.log("AOT " + demote(output) + " at " + location);
    }
    return output;
  }
};
```




- ⬇️ Lower simplicity
- ⬆️ Higher performance transparency

# Linvail: Performance Overhead




- Low performance transparency
  - 1,000sX (advice extension)
  - 100,000sX (advice layering)



https://github.com/chromium/octane

# Linvail: Semantic Overhead

Corpus of 81168 Test262 cases (post Aran)

| Discrepancy Tag | Discrepancy Count |
|---|---|
| `wrong-realm-default-array-prototype` | 18 |
| `elusive-dynamic-code-evaluation` | 12 |
| `no-cycle-detection-in-prototype-chain` | 8 |
| `v8-bug-proxy` | 2 |
| **Total** | 40 |

(< 0.1% discrepancy rate)

- 🟢 High semantic transparency (> 99.9%)



`https://github.com/tc39/test262`

# Part #3: Orchestrating Distributed Analysis

- Chapter 7: Orchestration of dynamic analysis for distributed applications

# Motivation: Distributed Constant Propagation

```
import { promote, demote, apply, frontier_advice } from "linvail/runtime";
const statics = [];
export const advice = {
  ...frontier_advice,
  "primitive@after": (_state, primitive, _location) => {
        const handle = promote(primitive);
    statics.push(handle);
    return handle;
  },
  "apply@around": (_state, callee, that, input, location) => {
    const output = apply(callee, that, input);
    if (input.every((arg) => statics.some((item) => item === arg))) {
      statics.push(output);
      console.log("Static " + demote(output) + " at " + location);
    }
    return output;
  }
}
```

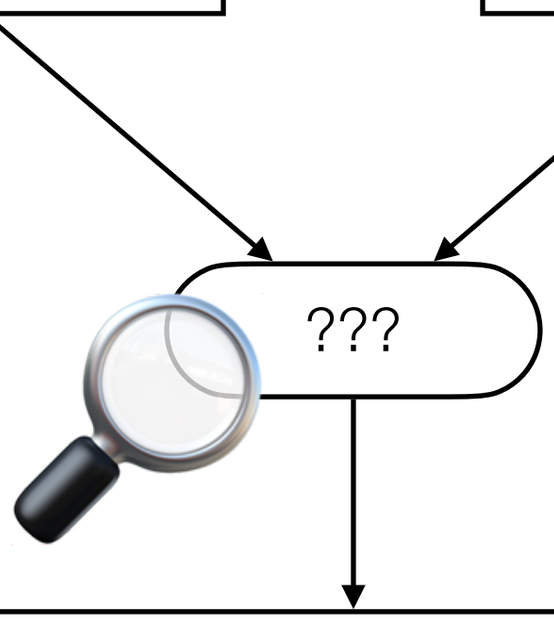
advice.js

server.js

```
rpc.id = async (x) => x;
```

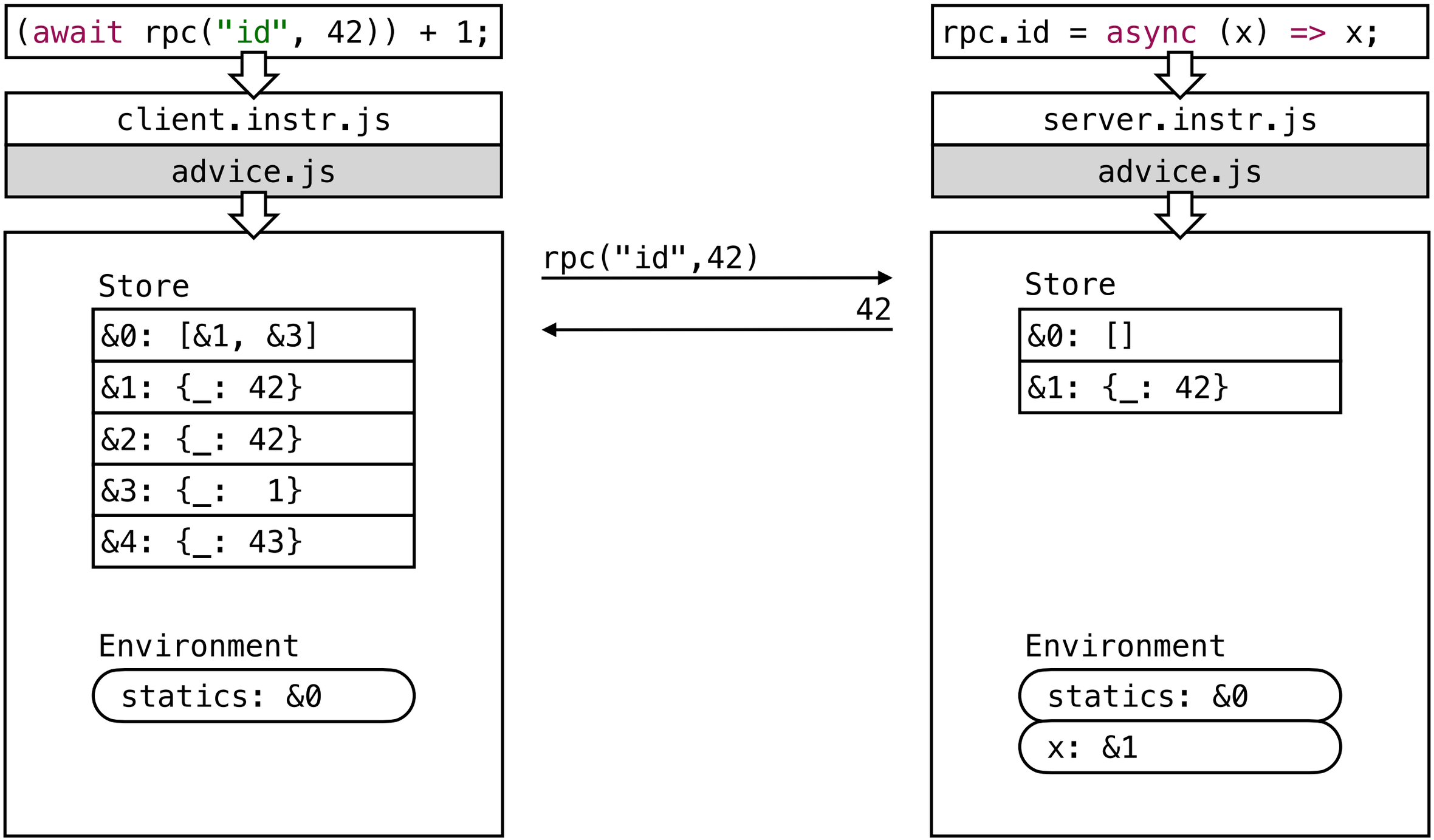
client.js

```
(await rpc("id", 42)) + 1;
```

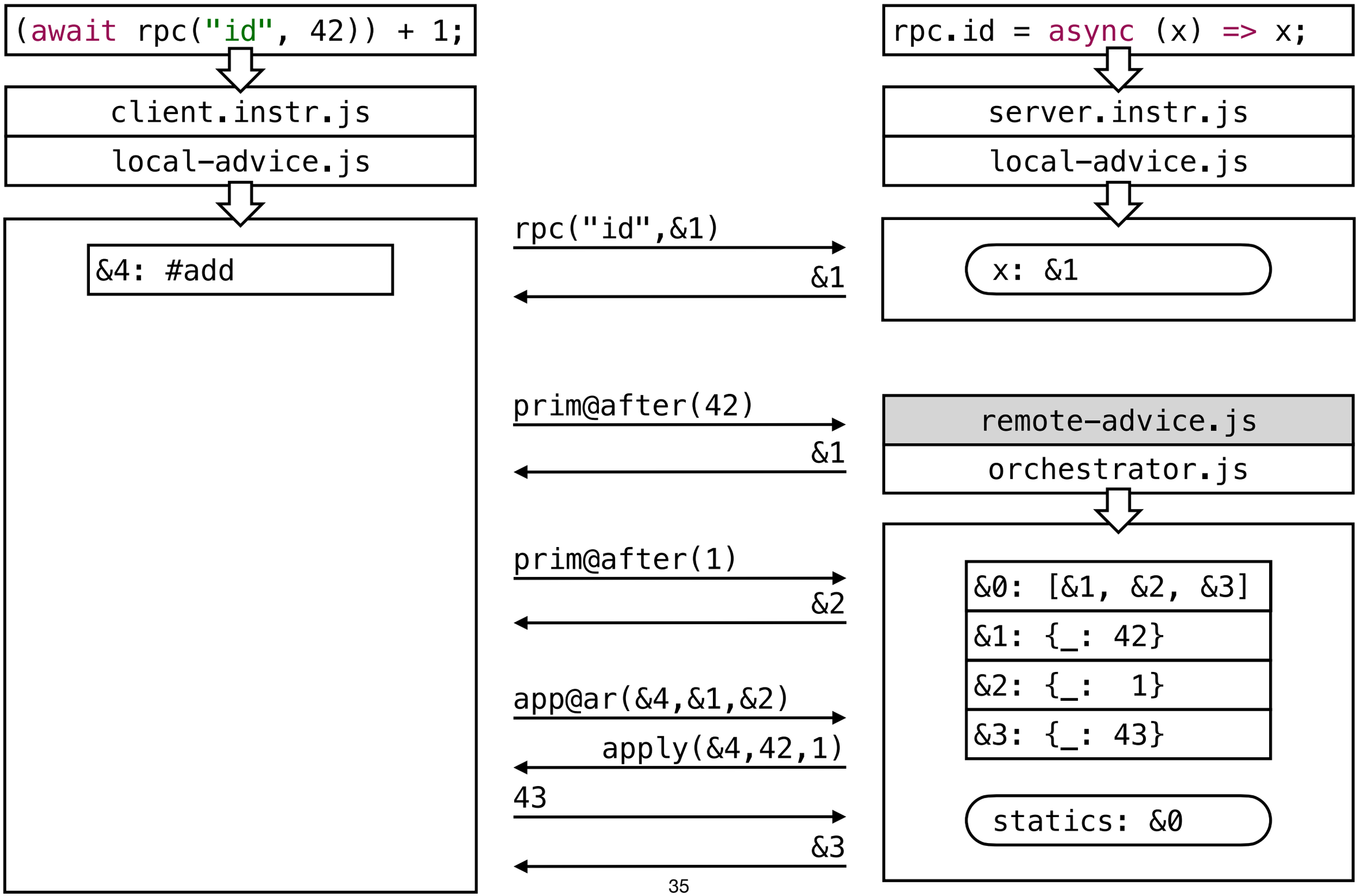



```
AOT 43 at client.mjs$.body.0.expression
```

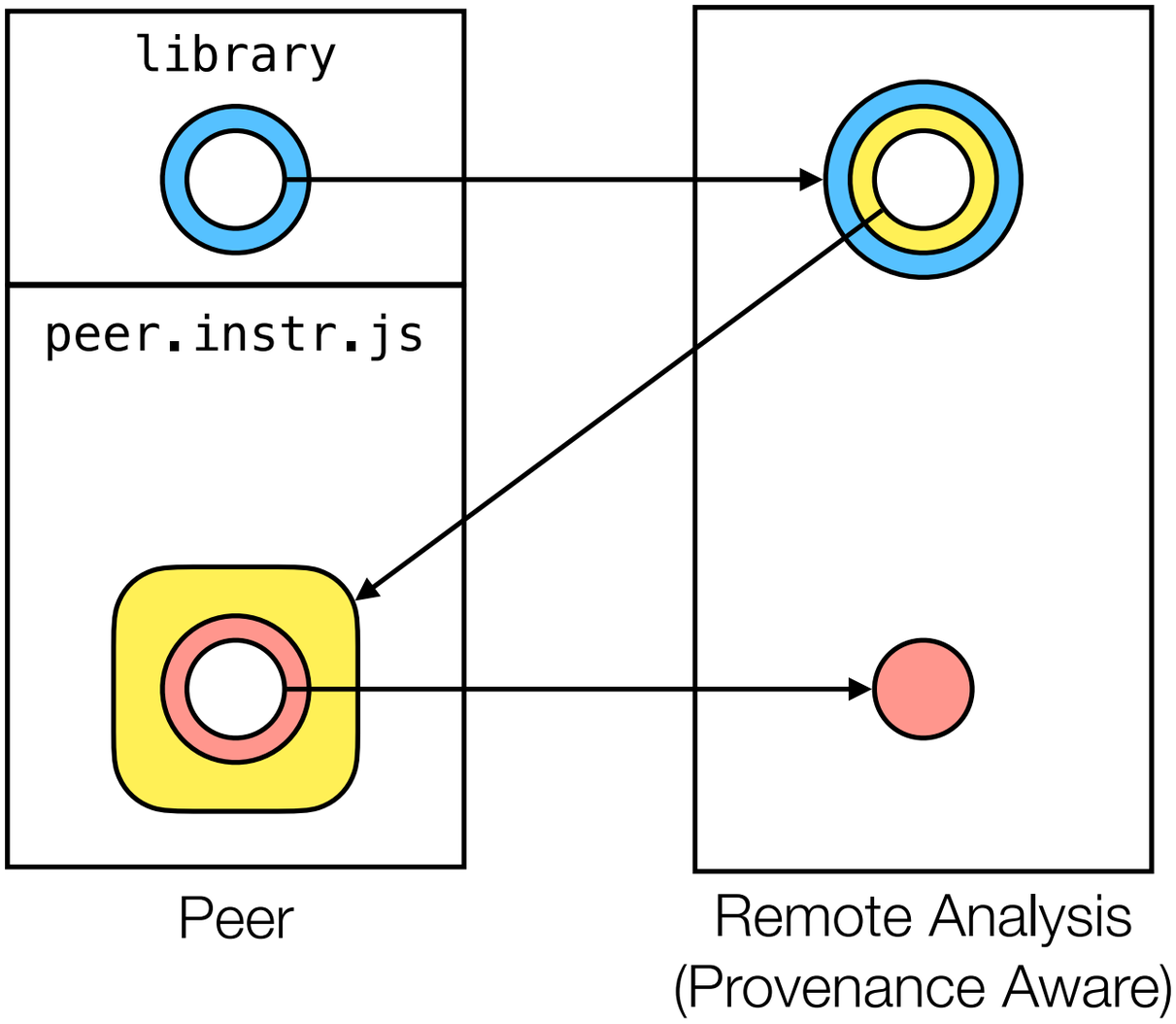
# Problem: Distributed Analysis State

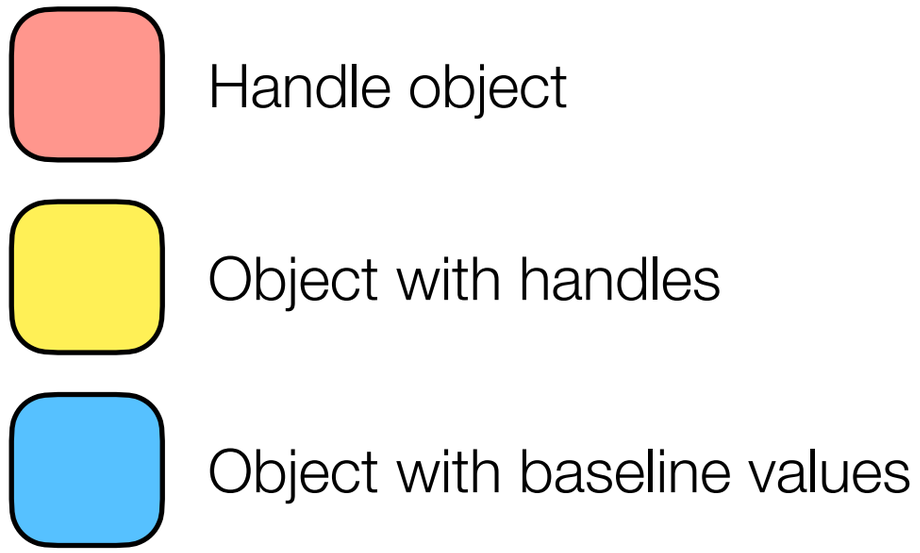


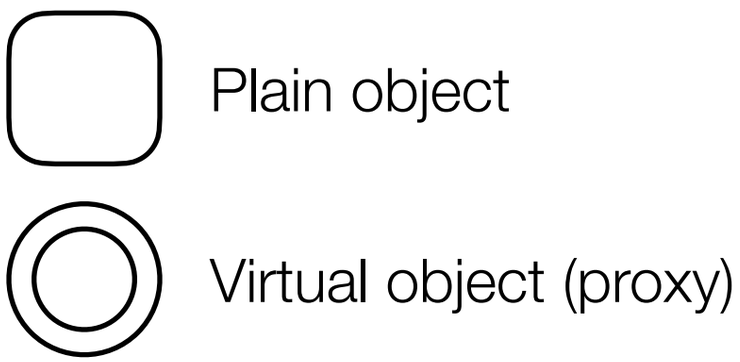
# Approach: Central Analysis

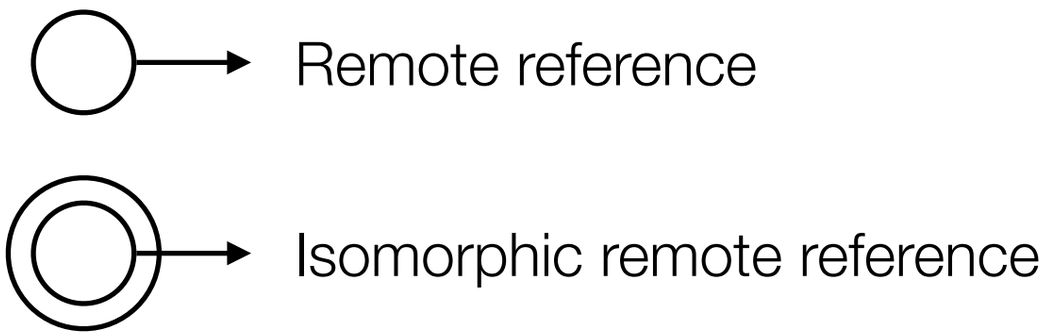


# Isomorphic Remote Reference



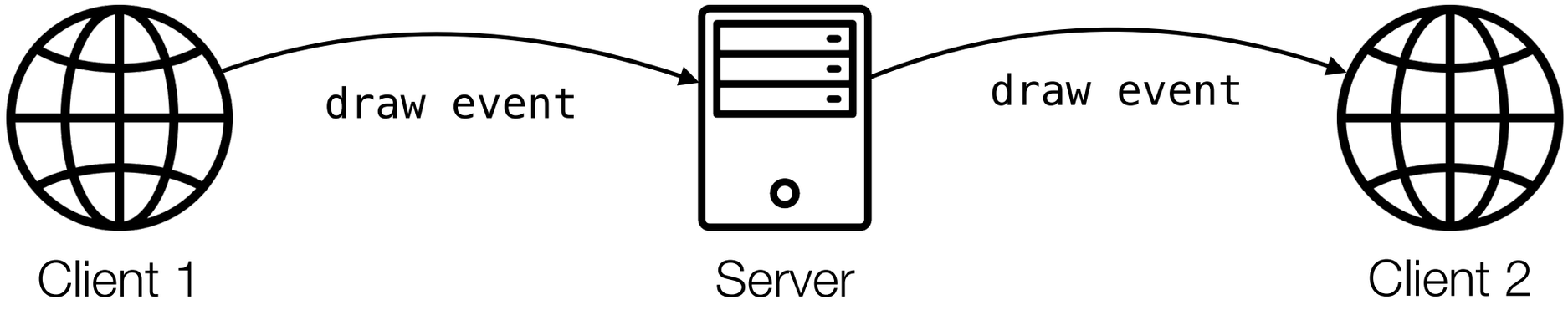



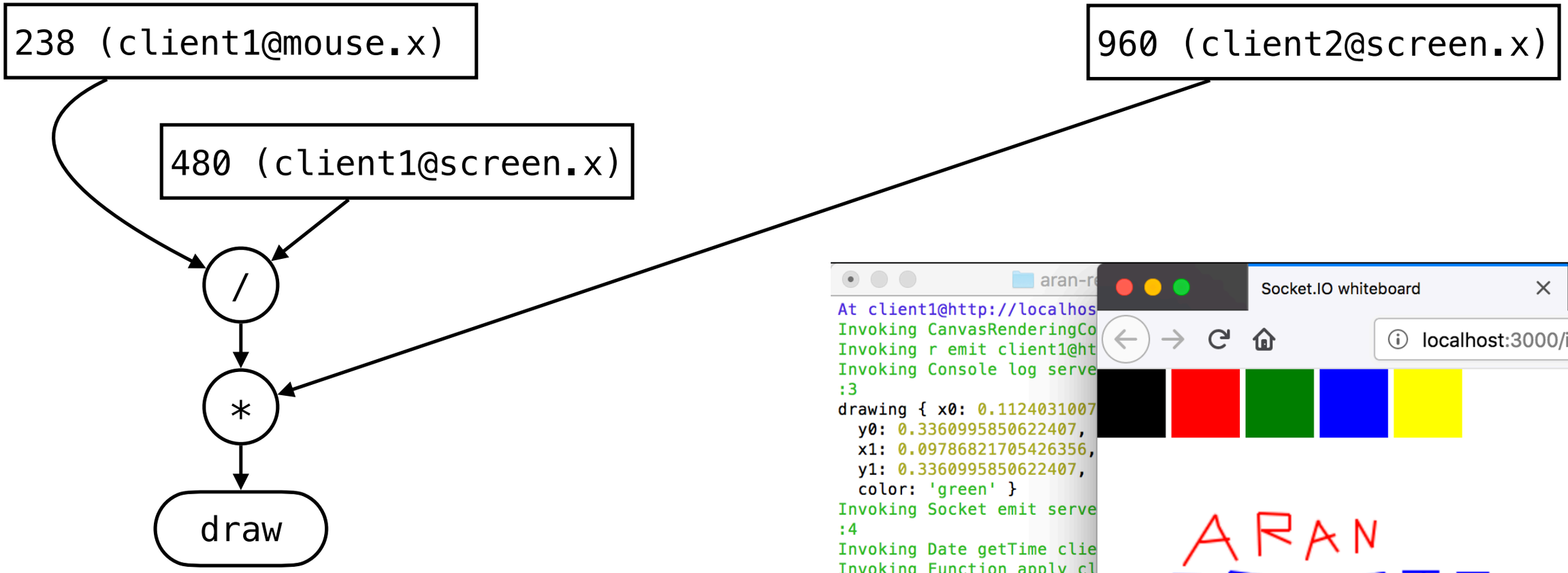



# Validation: Distributed Symbolic Execution

`https://github.com/socketio/socket.io/tree/master/examples/whiteboard`



# Conclusion

# Technical Overview

| Tool | Description | Files | LoC | Downloads |
|---|---|---|---|---|
| `otiluke` | Instrumentation infrastructure (Node.js + Browser via MITM) | 16 | 669 | 31,122 |
| `aran` | ECMAScript2025 instrumenter | 269 | 45,446 | 50,990 |
| `virtual-proxy` | Disentangle proxy target from invariant bookkeeping | 2 | 451 | 4,667 |
| `linvail` | Provenance tracking (stack + environment + store) | 78 | 12,898 | 23,290 |
| `posix-socket` | Synchronous socket API (C++ binding) | 4 | 997 | 100,974 |
| `antena` | Homogenous communication | 11 | 794 | 15,313 |
| `melf` | Synchronous yet responsive remote procedure call | 2 | 163 | 13,730 |
| `melf-share` | Isomorphic remote reference | 8 | 585 | 7,823 |
| `aran-remote` | Analysis orchestration | 11 | 426 | 5,600 |



https://github.com/lachrist

# Summary

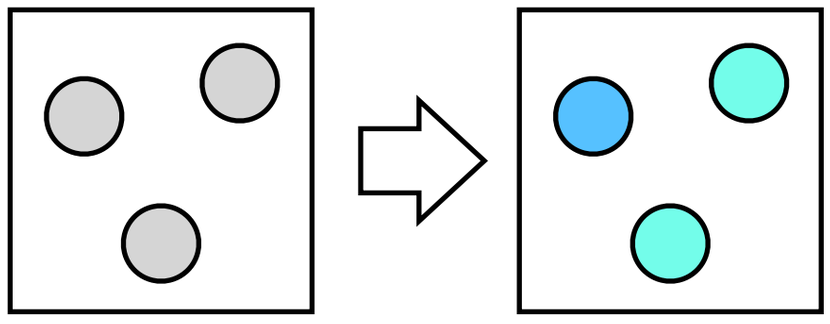
Background: source code instrumentation

- ✅ Transparent
- 🟢 Portable
- 🟢 Generic
- 🟢 Source-adjacent
- 🔴 Complex

Part #1: JavaScript transpilation with Aran

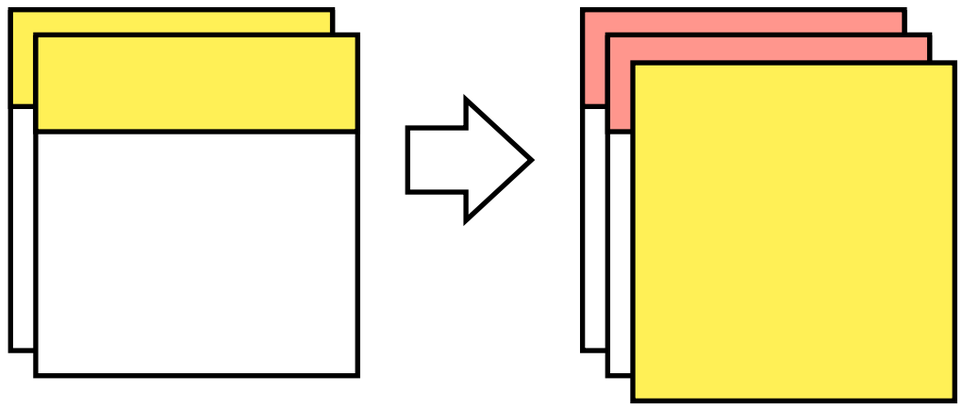

- ⬆️ Simplicity
- ↘️ Source-adjacency
- ⬇️ Perf. transparency
- ➡️ Sem. transparency

Part #2: Tracking provenance with Linvail



- ⬆️ Simplicity
- ⬇️ Performance Transparency

Part #3: Analysis orchestr. with AranRemote



- ⬆️ Simplicity
- ⬇️ ⬇️ Performance Transparency