Microsoft Research

Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available.

Scout: Using High-Level Design Constraints to Automatically Generate Design Variations

Amanda Swearngin, Andy Ko, James Fogarty

Information School
UNIVERSITY of WASHINGTON

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

DUB DESIGN USE BUILD

PLSE


Download free photos to fuel your next project

Version 1

Version 2

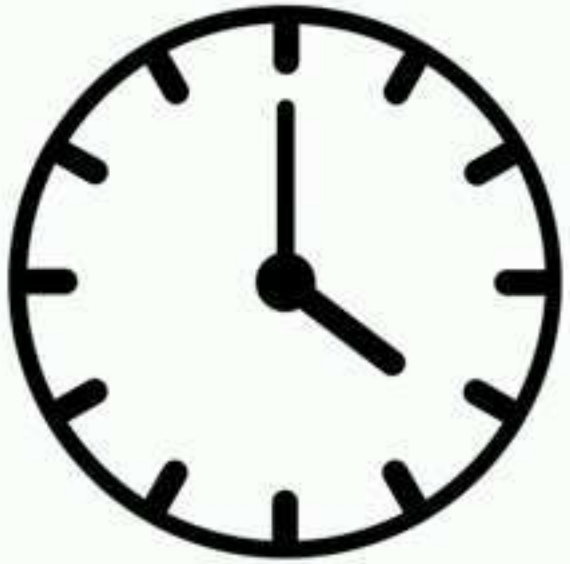
Version 3

Designing alternatives leads to better designs

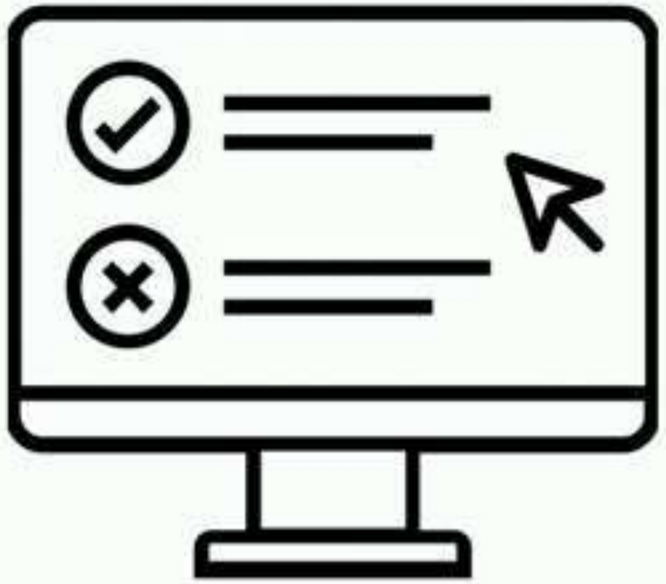
(Lee et. al, Dow et. al.)

Designer

???



Usability &
Visual Design



Designing alternatives is difficult.

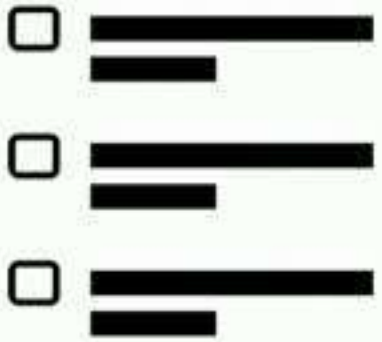
Scout: Using High-Level Design Constraints to Automatically Generate Design Variations

- Can we generate many **good** design variations automatically to help designers?

- Can we help designers follow design principles?

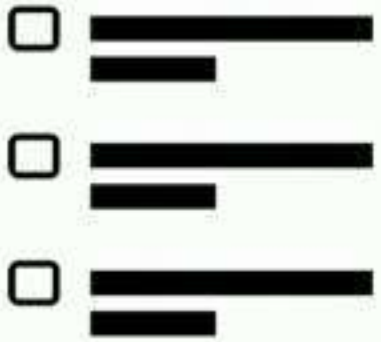
Scout: Inputs and Outputs

High-Level Design
Constraints



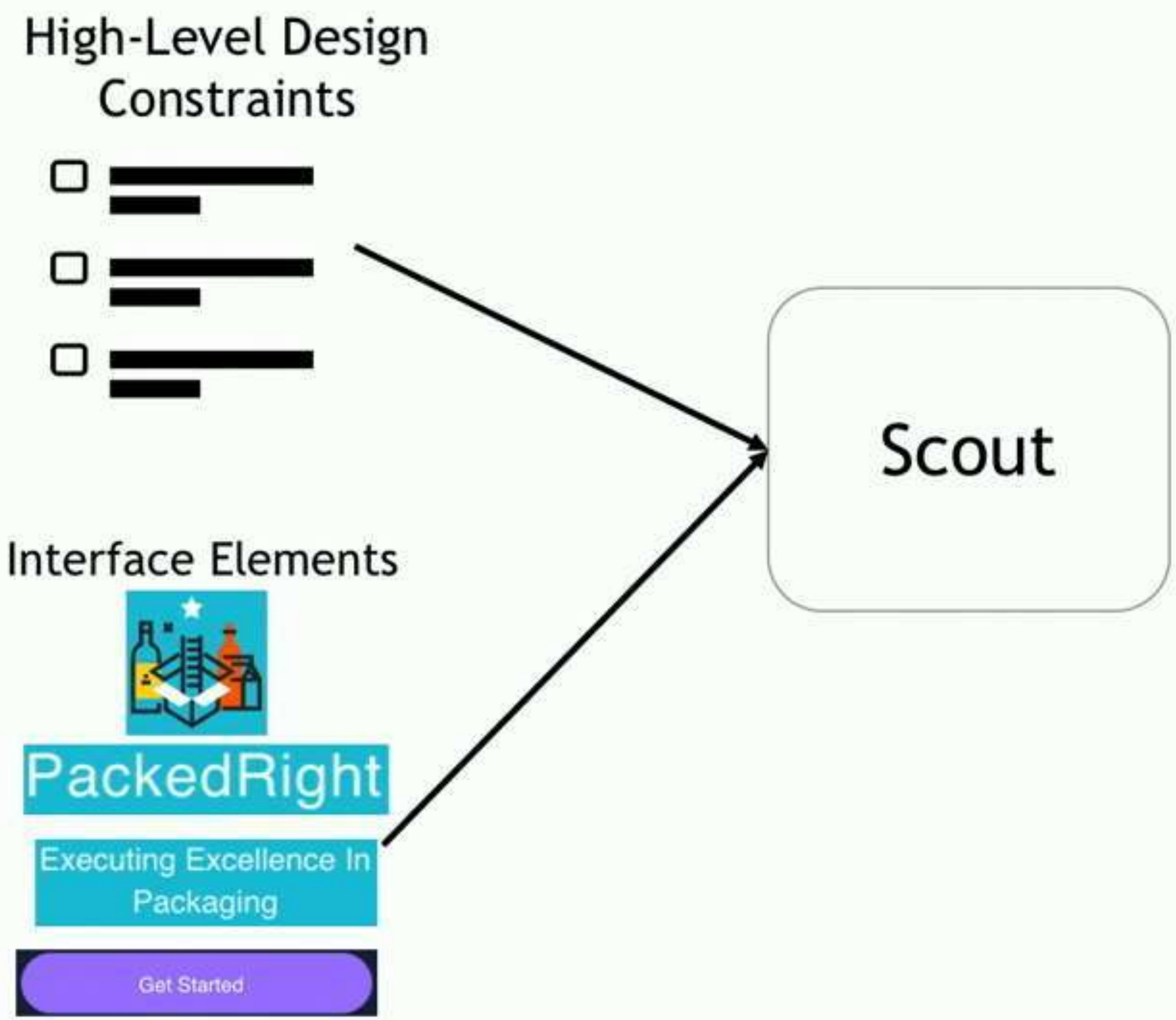
Scout: Inputs and Outputs

High-Level Design Constraints

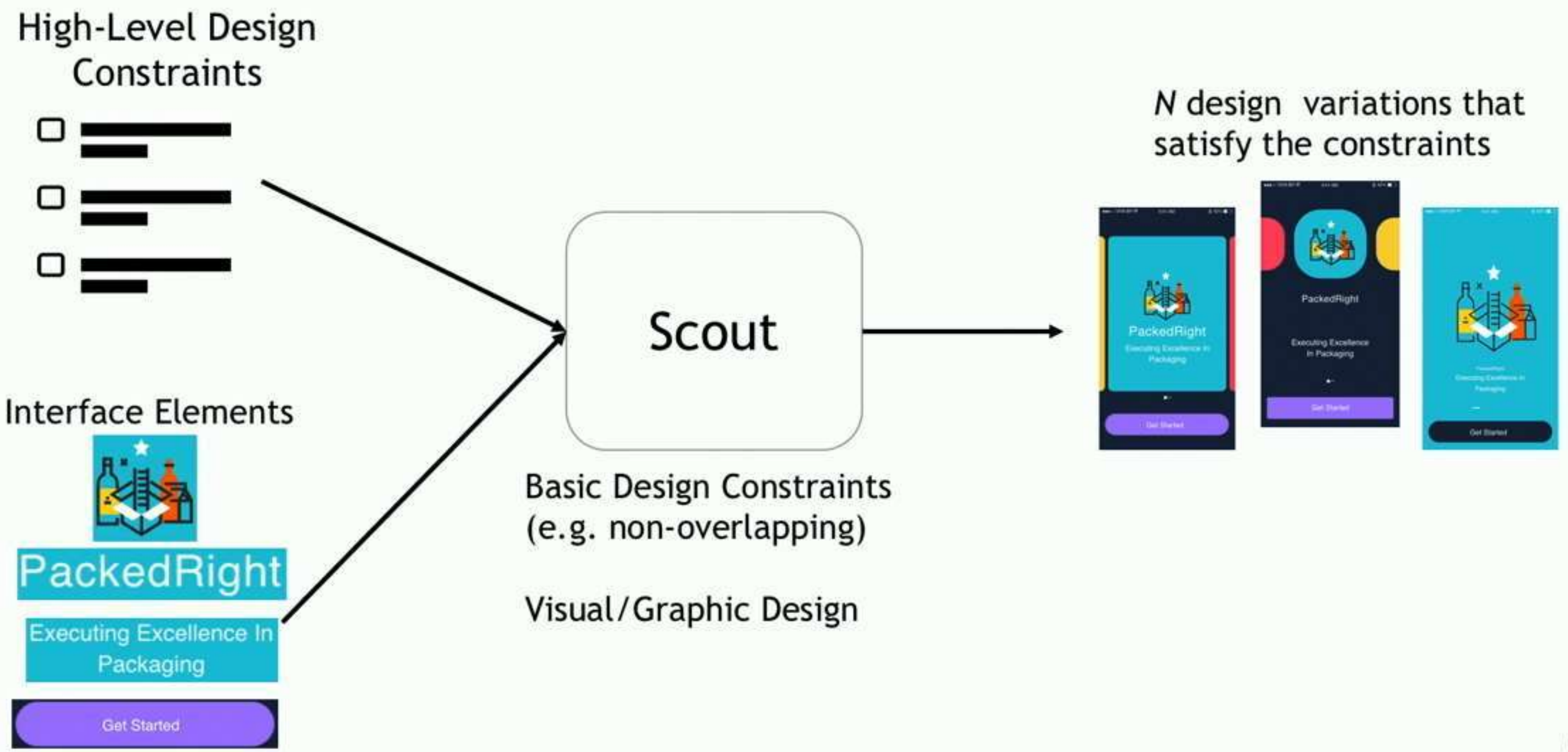


Title text should be prominent

Scout: Inputs and Outputs



Scout: Inputs and Outputs

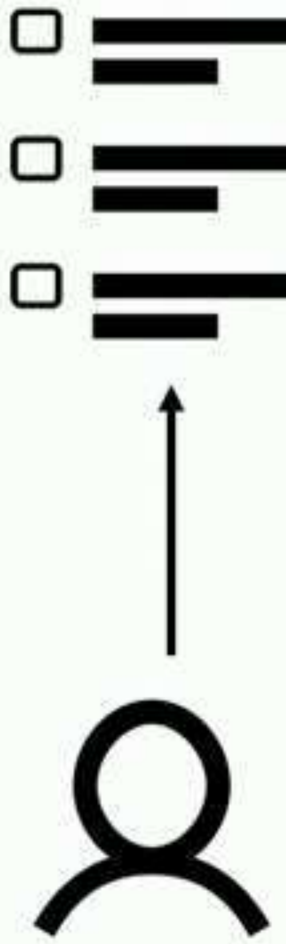


Scout System Overview

Designer

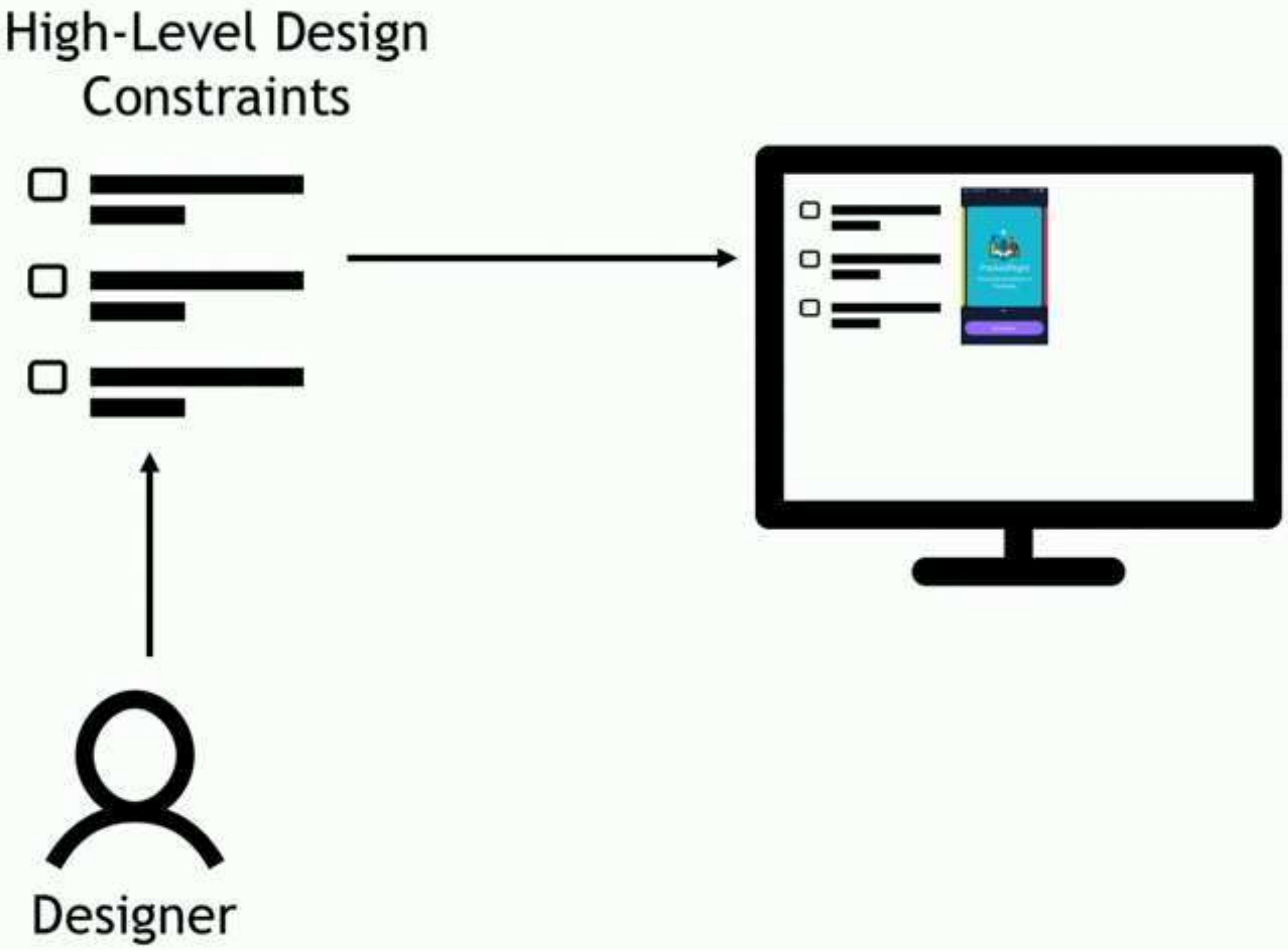
Scout System Overview

High-Level Design
Constraints

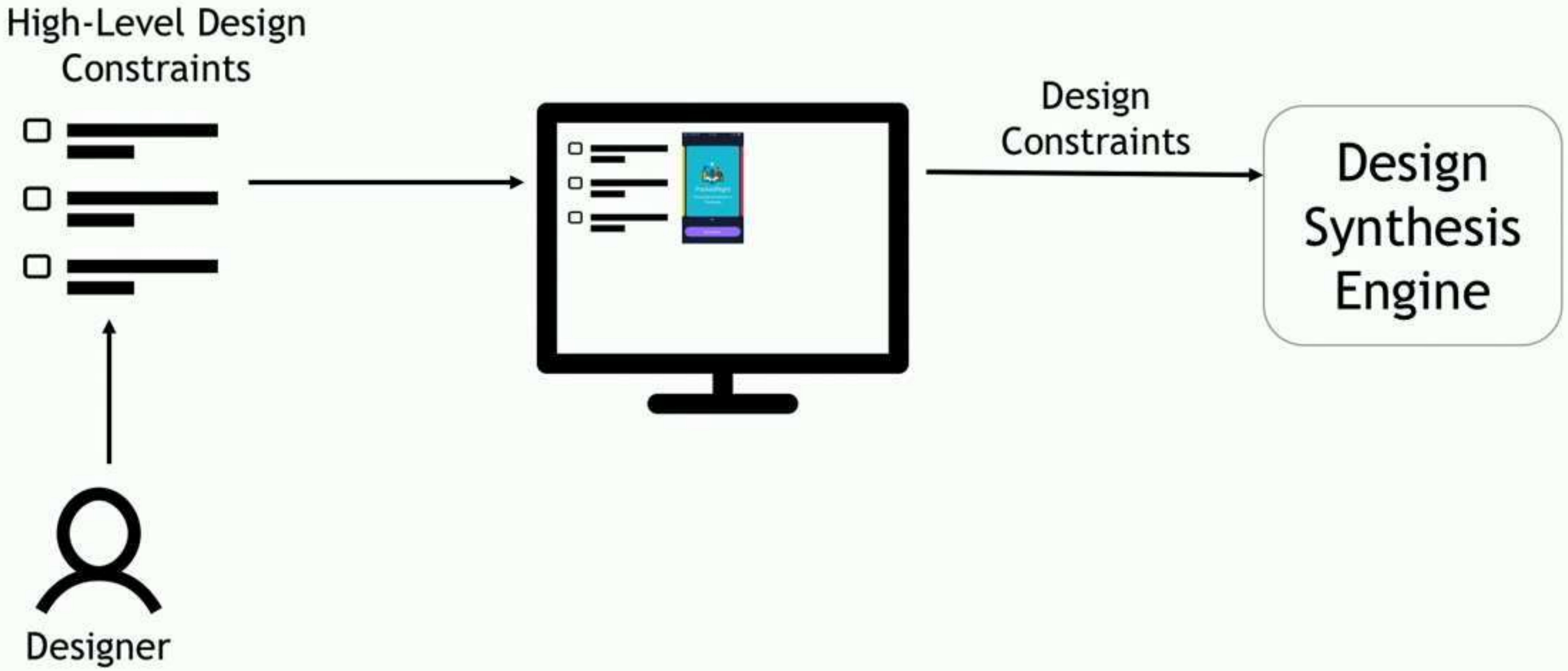


Designer

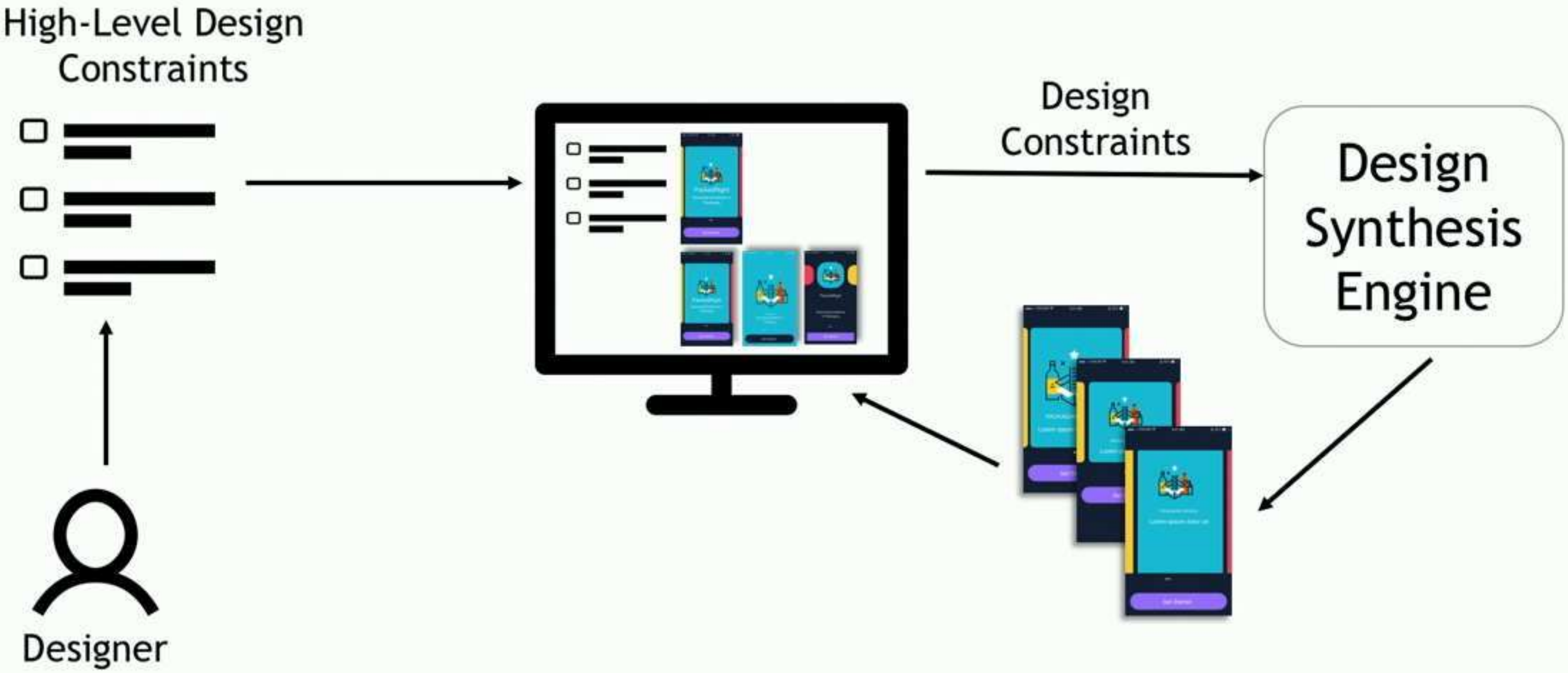
Scout System Overview



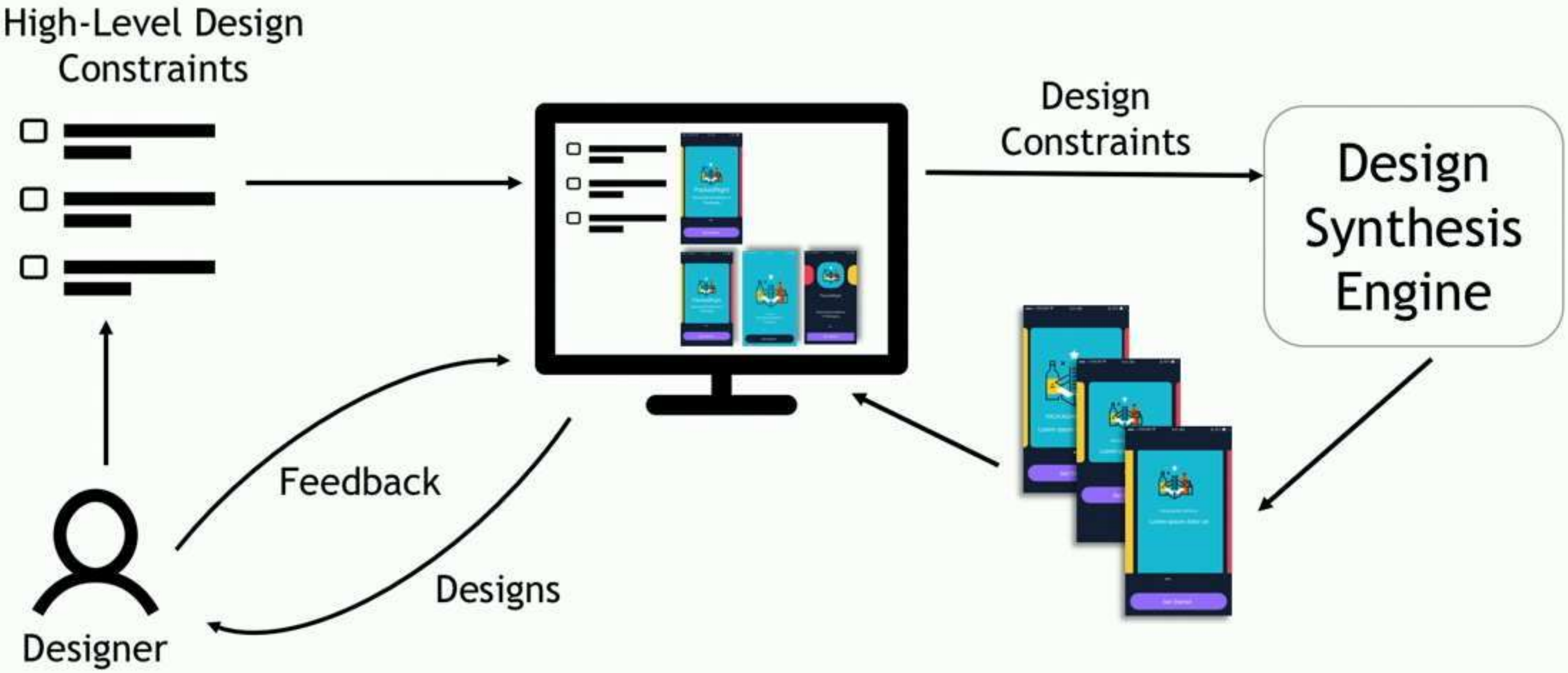
Scout System Overview



Scout System Overview



Scout System Overview



Overview

- Scout System Overview
- **High-Level Design Constraints**
- Design Synthesis Engine

Structure/Proximity Principle[1]

Keep related things together

1. Constantine and Lockwood, 1999

Structure/Proximity Principle[1]

Keep related things together

Title and tagline text are separate.

The user is less likely to know what the tagline is describing.

Bad

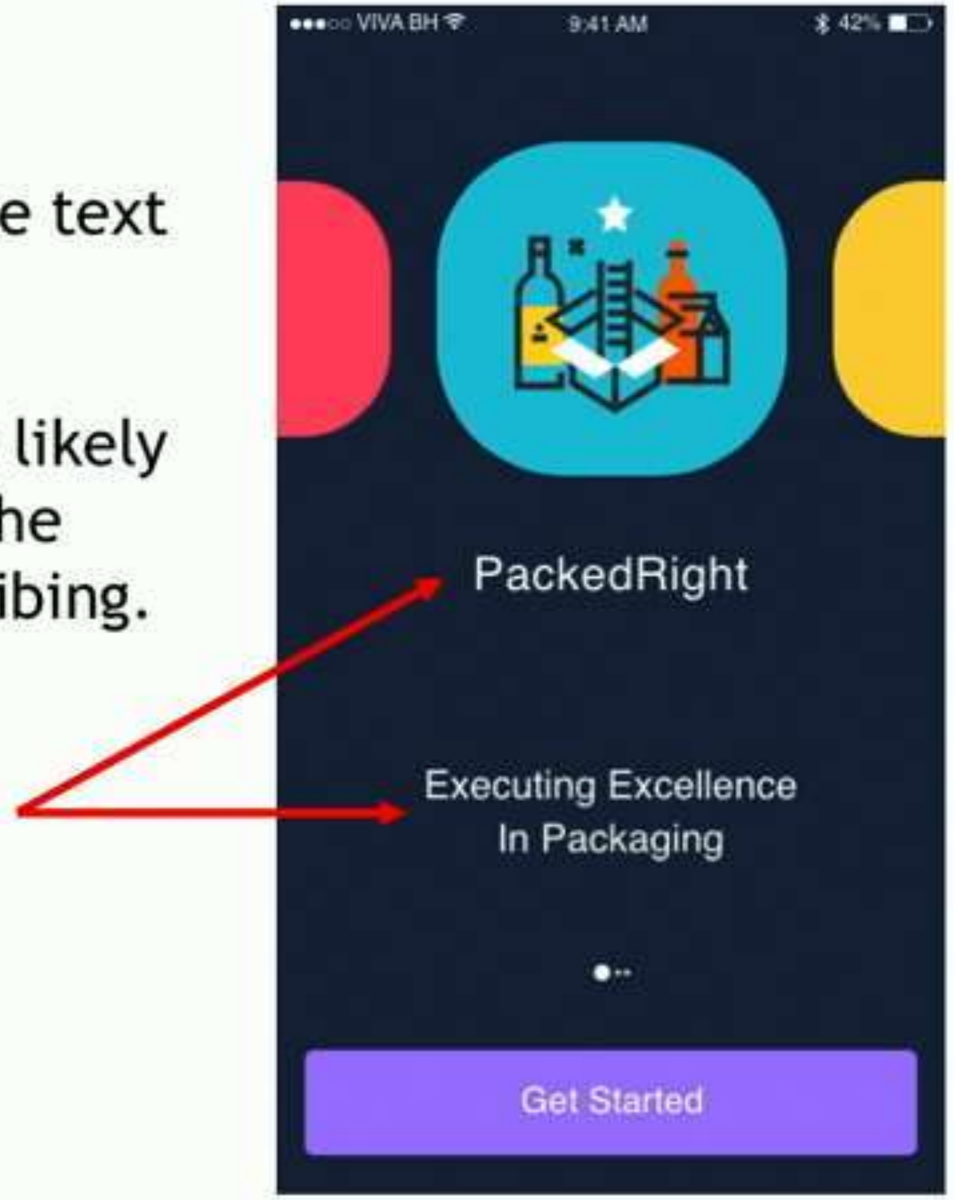
1. Constantine and Lockwood, 1999

Structure/Proximity Principle[1]

Keep related things together

Title and tagline text are separate.

The user is less likely to know what the tagline is describing.



Bad

1. Constantine and Lockwood, 1999

Structure/Proximity Principle[1]

Keep related things together

Title and tagline text are separate.

The user is less likely to know what the tagline is describing.

Bad

Good

Title text and tagline text appear together with the icon.

The user will see them as a cohesive unit.

1. Constantine and Lockwood, 1999

A High-Level Grouping Constraint

- Designer can use them to group a set of **related** elements

- Two aspects: **Order** and **Type**

A High-Level Grouping Constraint

- Designer can use them to group a set of **related** elements

- Two aspects: **Order** and **Type**

A High-Level Grouping Constraint

Group, Order unimportant

- Designer can use them to group a set of **related** elements

- Two aspects: **Order** and **Type**

A High-Level Grouping Constraint

- Designer can use them to group a set of **related** elements

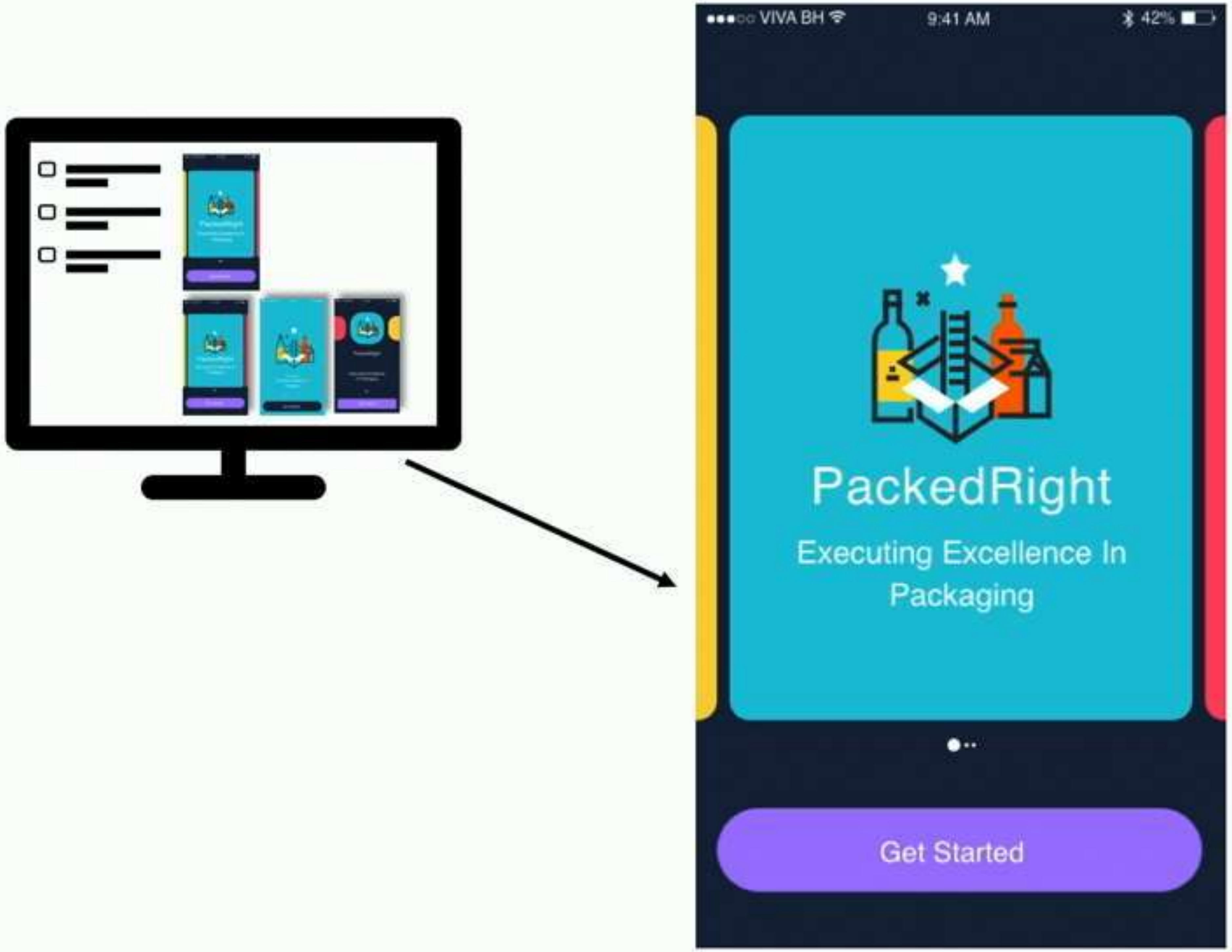
- Two aspects: **Order** and **Type**

Group, Order unimportant

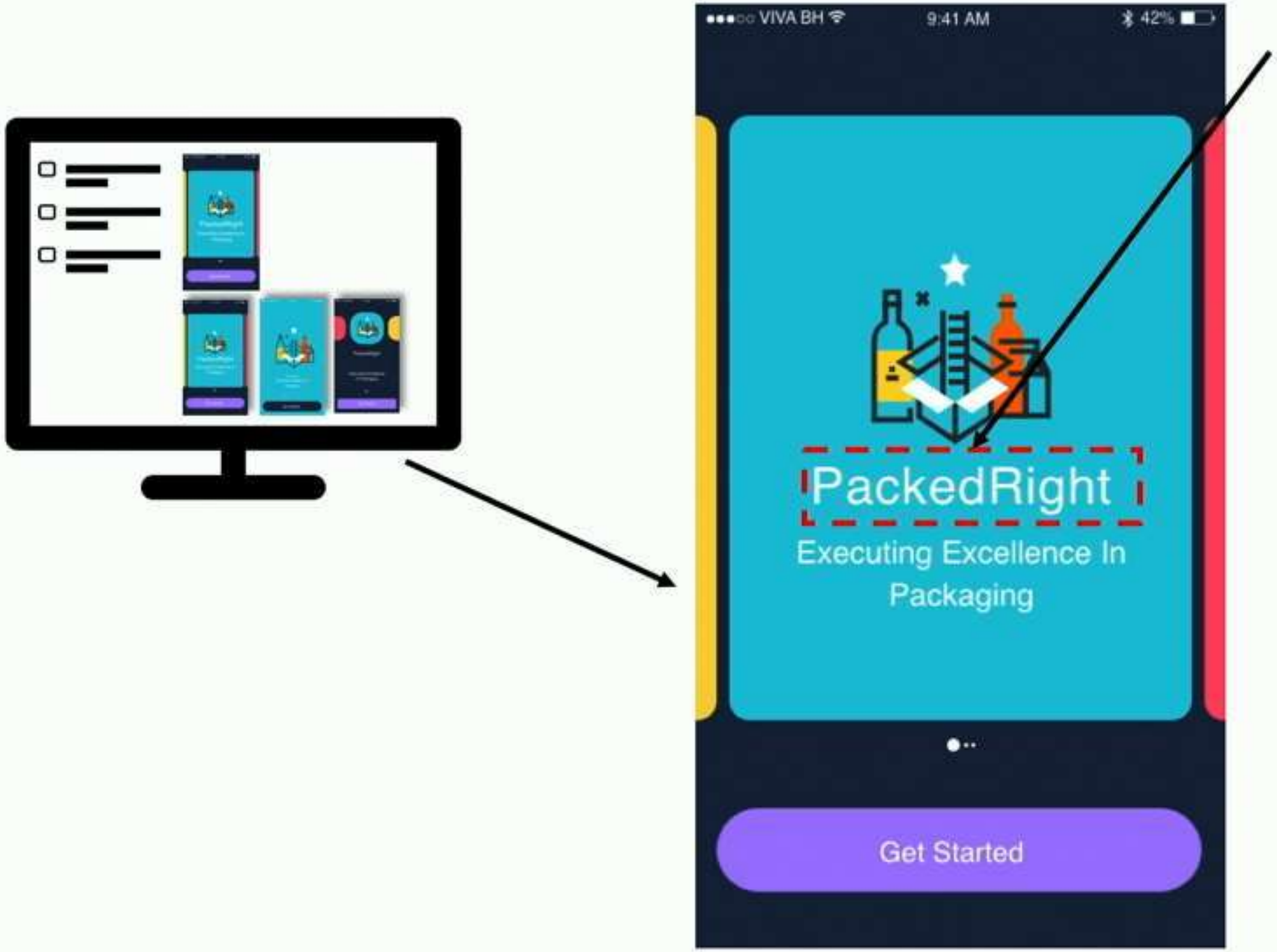
Variations

...

High-Level Feedback Constraints



High-Level Feedback Constraints



Element

Keep this element here.

High-Level Feedback Constraints

Element

Keep this element here.

Relational

Subtitle should appear underneath the tagline.

High-Level Feedback Constraints

Element

Keep this element here.

Relational

Subtitle should appear underneath the tagline.

Global

Use the 10px layout grid.

Overview

- Scout System Overview
- High-Level Design Constraints
- **Design Synthesis Engine**

Design Synthesis Goals

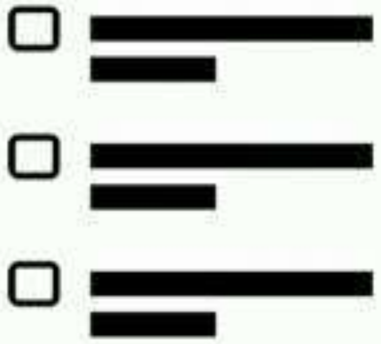
- Encode high-level **design constraints,** and **basic design** constraints (e.g. non-overlapping**,** and **feedback constraints** as a set of constraints to generate designs.

- Generate good designs that respect **usability** and **graphic design** principles (e.g. alignment, symmetry)

- Generate many designs quickly to make the system interactive.

Design Synthesis: Inputs and Outputs

High-Level Design Constraints



Interface Elements

Scout Synthesis Engine

Basic Design Constraints
(e.g. non-overlapping)

High-Level Constraints
- Semantic Groups, Labels, Prominence Levels, Feedback

Design Variables
- Alignment, proximity, arrangements

N design variations that satisfy the constraints

For each design, for all elements X, Y coordinates, height, width

Design Synthesis – Encoding the Constraints

- Basic design constraints
  - Non-overlapping
  - UI elements stay inside containers and design canvas

- High level constraints
  - Semantic Groups
  - Prominence Levels (e.g. increase or decrease visual salience)
  - Feedback constraints

Design Synthesis – Finding Solutions

Variables – modify different properties of design

- Alignment
- Margins
- Proximity
- Label position
- Arrangement
  (e.g. horizontal, rows)

Searching

- Randomly order variables
- Backtracking/Branch and bound to assign variables iteratively
- Check and discard invalid solutions
- Generate N designs

Design Synthesis – Getting Good Designs

- Visual Cost Variables
  - Whitespace
  - Balance & symmetry
  - Alignment
  - …

Design Synthesis – Getting Good Designs

- Visual Cost Variables
  - Whitespace
  - Balance & symmetry
  - Alignment
  - …

- Approach
  - Generate a bunch of designs
  - Rank them by cost
  - Return lowest cost first

Version 1,
Cost: 20

Version 2,
Cost: 50

Version 3,
Cost: 60

Challenges

- Generating good designs
  - Ranking function
  - Bias the search to choose good combinations of variables
- Diversity
  - Lots of spatially different designs
- Scalable and interactive
  - Can’t overwhelm the solver

Scout: Using High-Level Design Constraints to Automatically Generate Design Variations

Amanda Swearngin amaswea@cs.washington.edu
Andrew J. Ko ajko@uw.edu
James Fogarty jfogarty@cs.Washington.edu

DUB DESIGN USE BUILD
PLSE
PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING
Information School UNIVERSITY of WASHINGTON

Platform-Independent Migration of Stateful JavaScript IoT Applications

Workshop on Programming Languages and Software Engineering Research in the Pacific Northwest (PNWPLSE)

Julien Gascon-Samson, Kumseok Jung, Karthik Pattabiraman

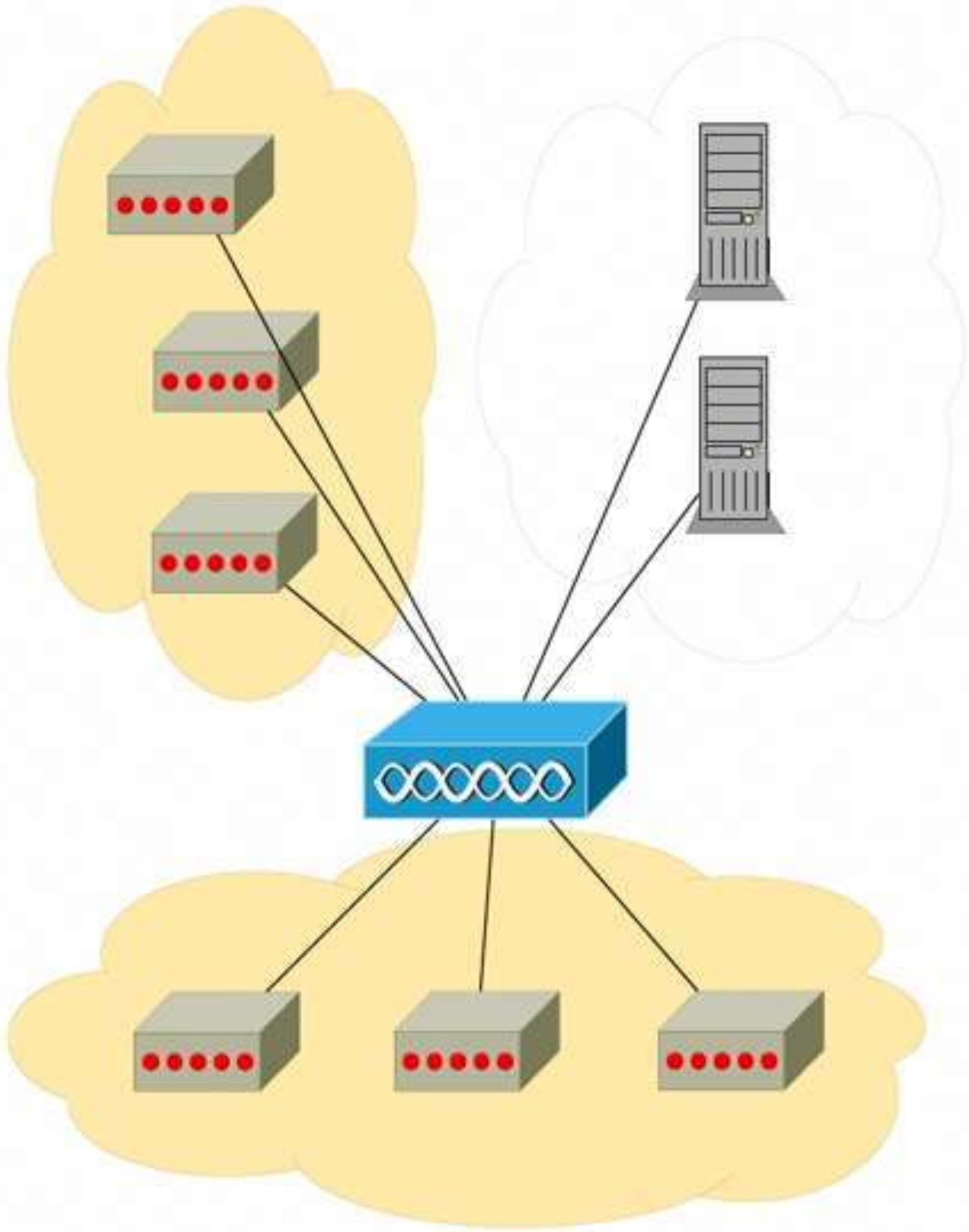
University of British Columbia
Department of Electrical and Computer Engineering
Vancouver, Canada

ece Electrical and Computer Engineering

UBC

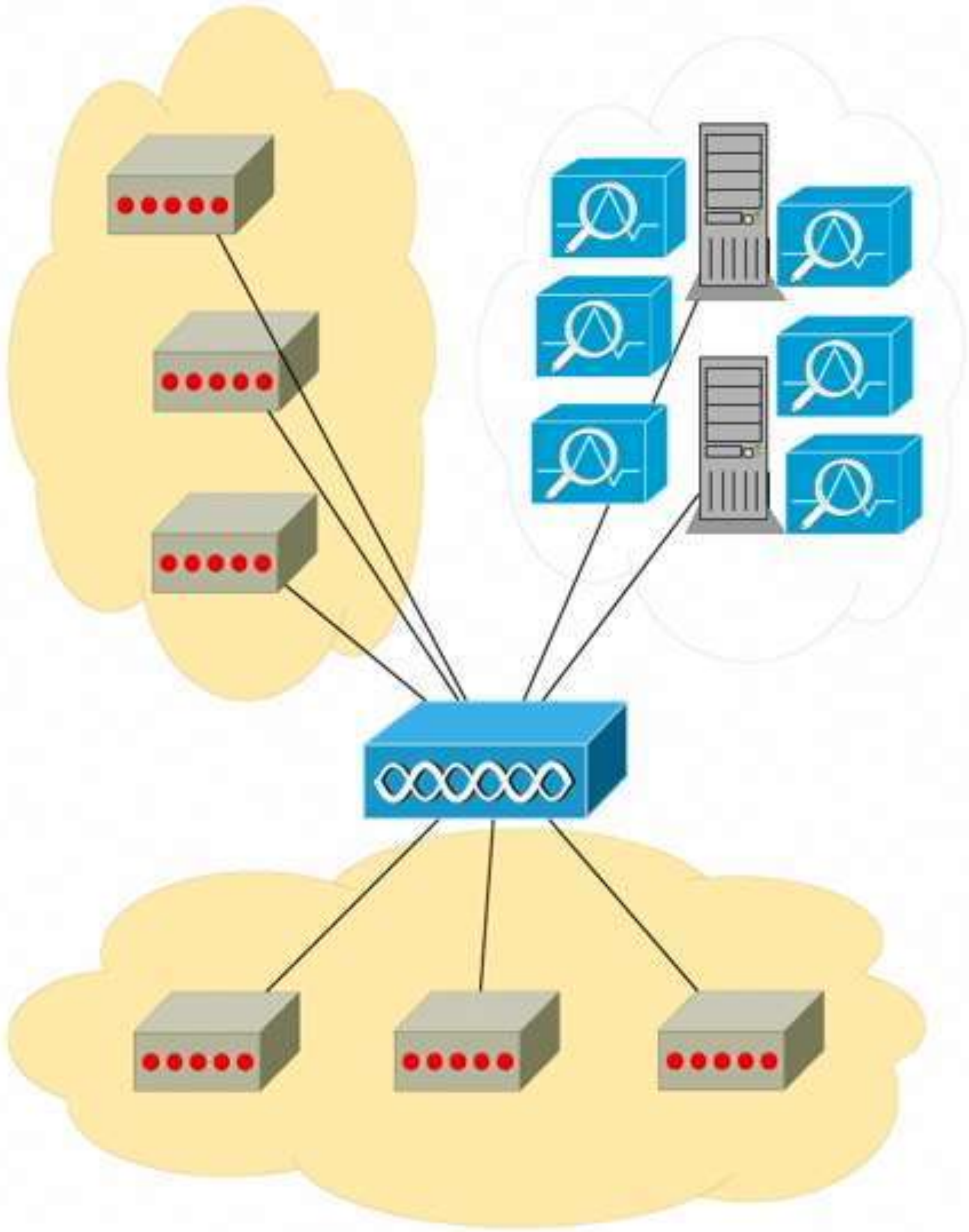
May 14th, 2018

Motivation



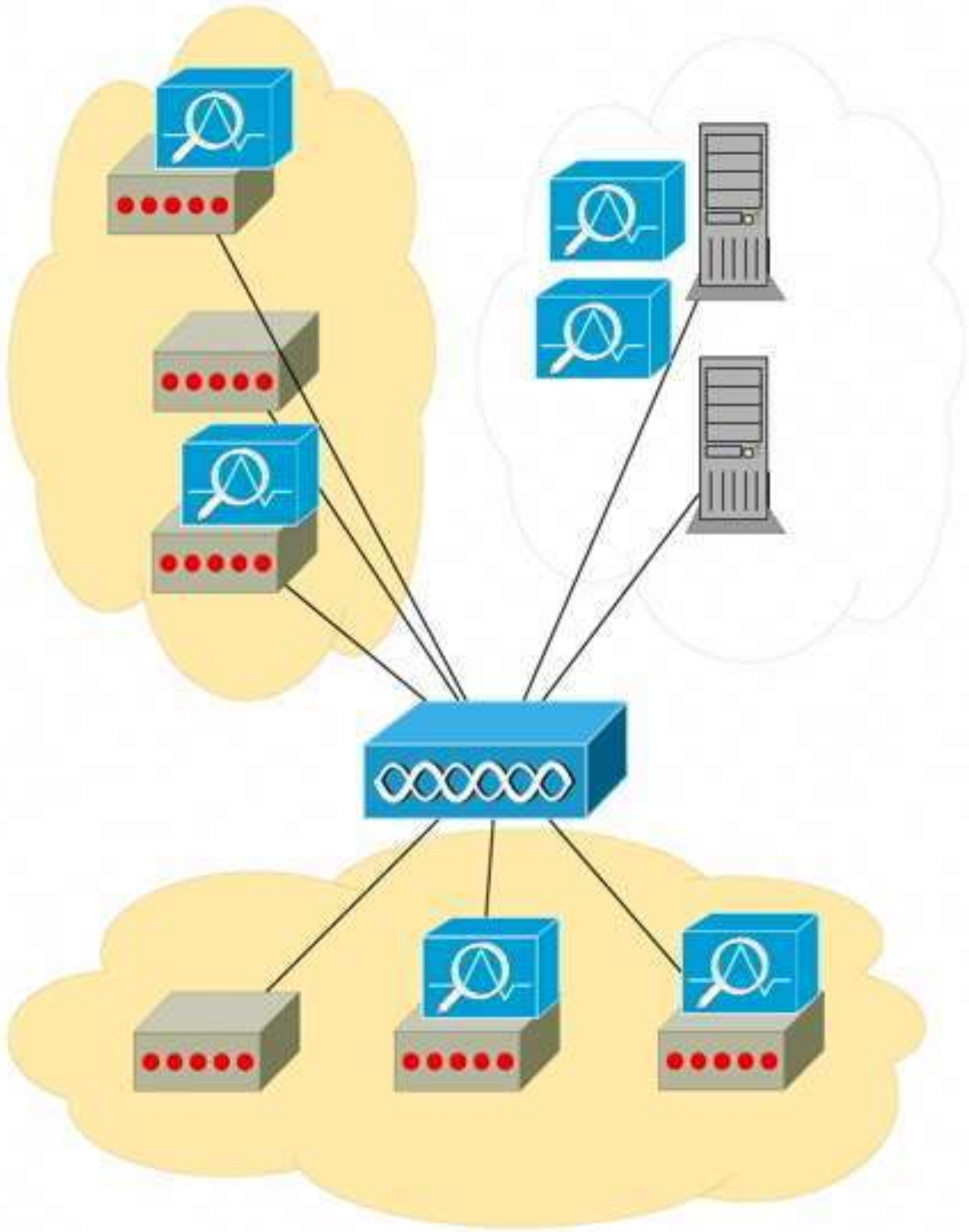
- World of IoT growing at a very fast pace!

Motivation



- World of IoT growing at a very fast pace!
- Traditionaly, processing was done in the cloud

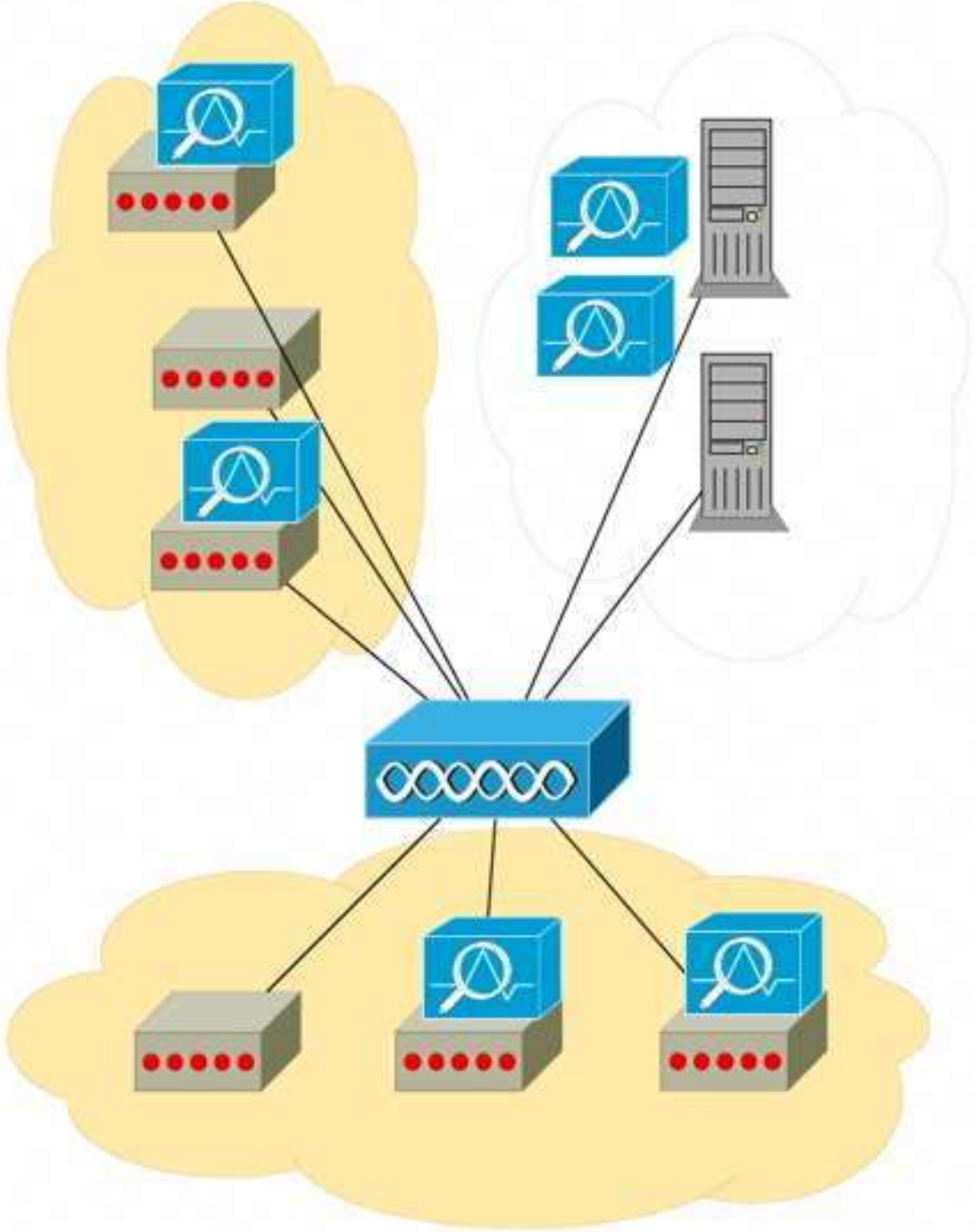
Motivation



- World of IoT growing at a very fast pace!
- Traditionaly, processing was done in the cloud
- Emerging trend: running applications on the IoT devices themselves (edge)
  - Performance, costs, reliability

Goals and Motivation

- **ThingsJS**: a framework for developing and deploying *high-level* applications on IoT devices (edge computing)

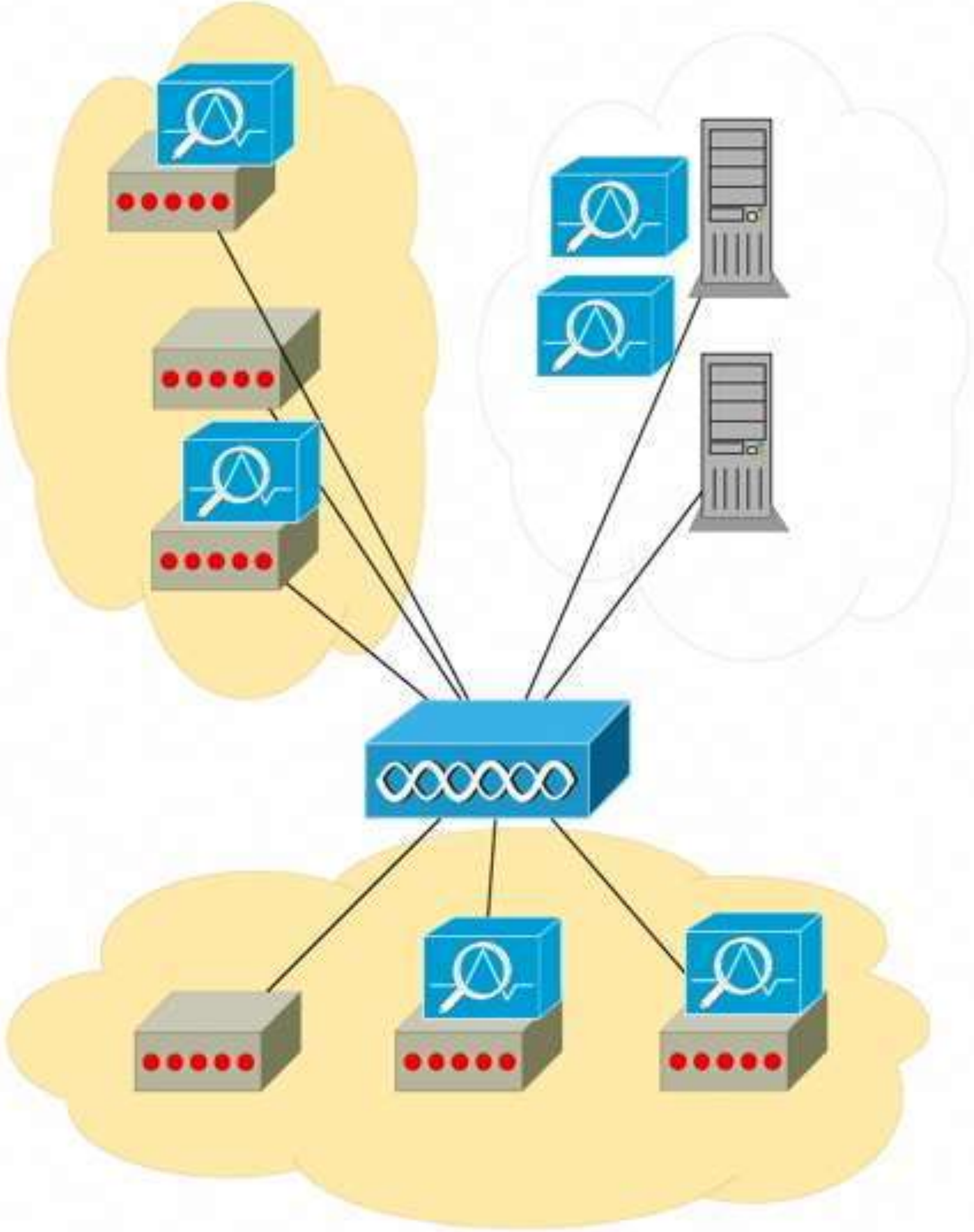


Goals and Motivation

- **ThingsJS**: a framework for developing and deploying *high-level* applications on IoT devices (edge computing)

⇓

JavaScript



Goals and Motivation

- **ThingsJS**: a framework for developing and deploying *high-level* applications on IoT devices (edge computing)

⇓

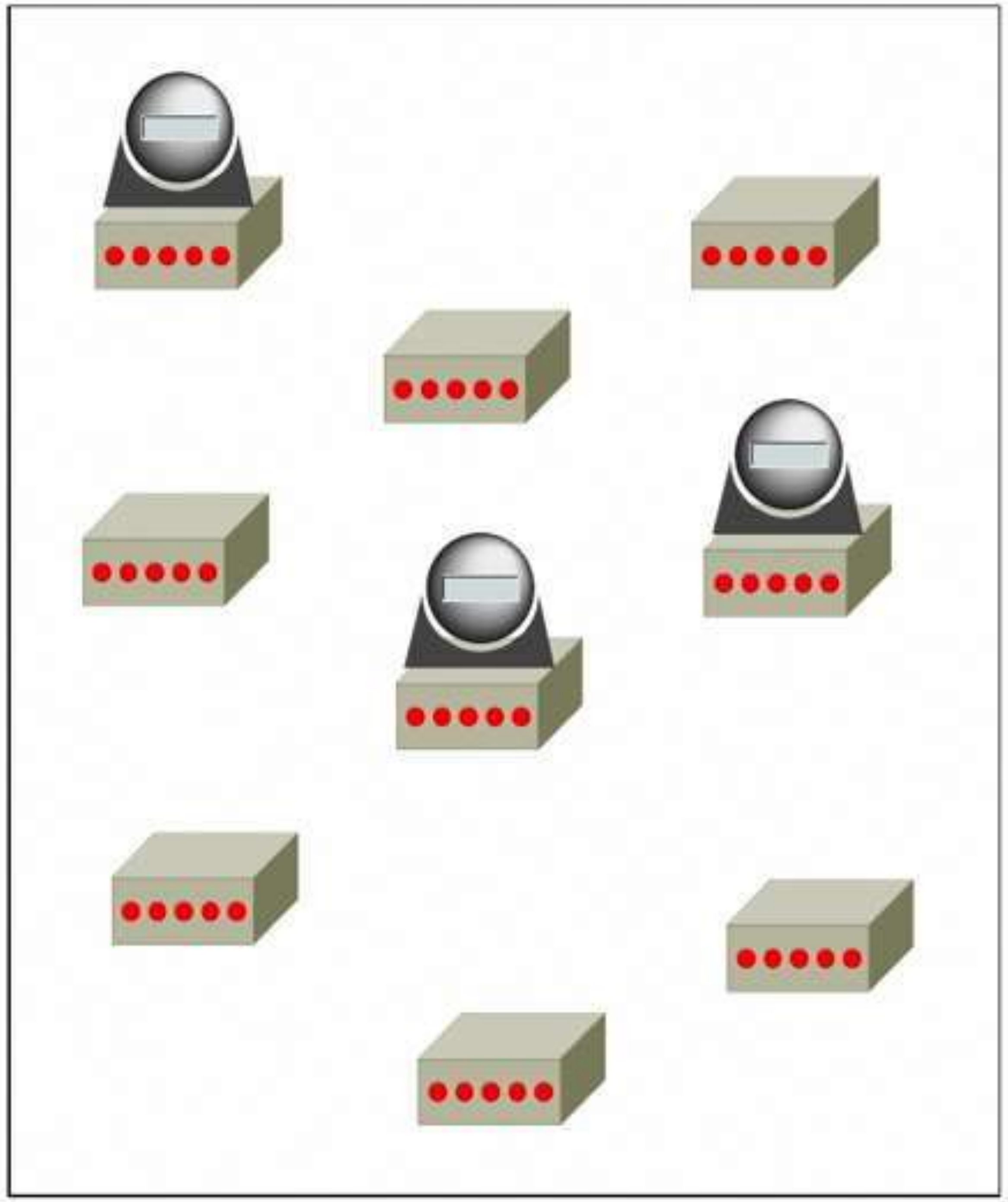
JavaScript

- Programmers are typically more productive in higher-level languages
- JavaScript: strong user base

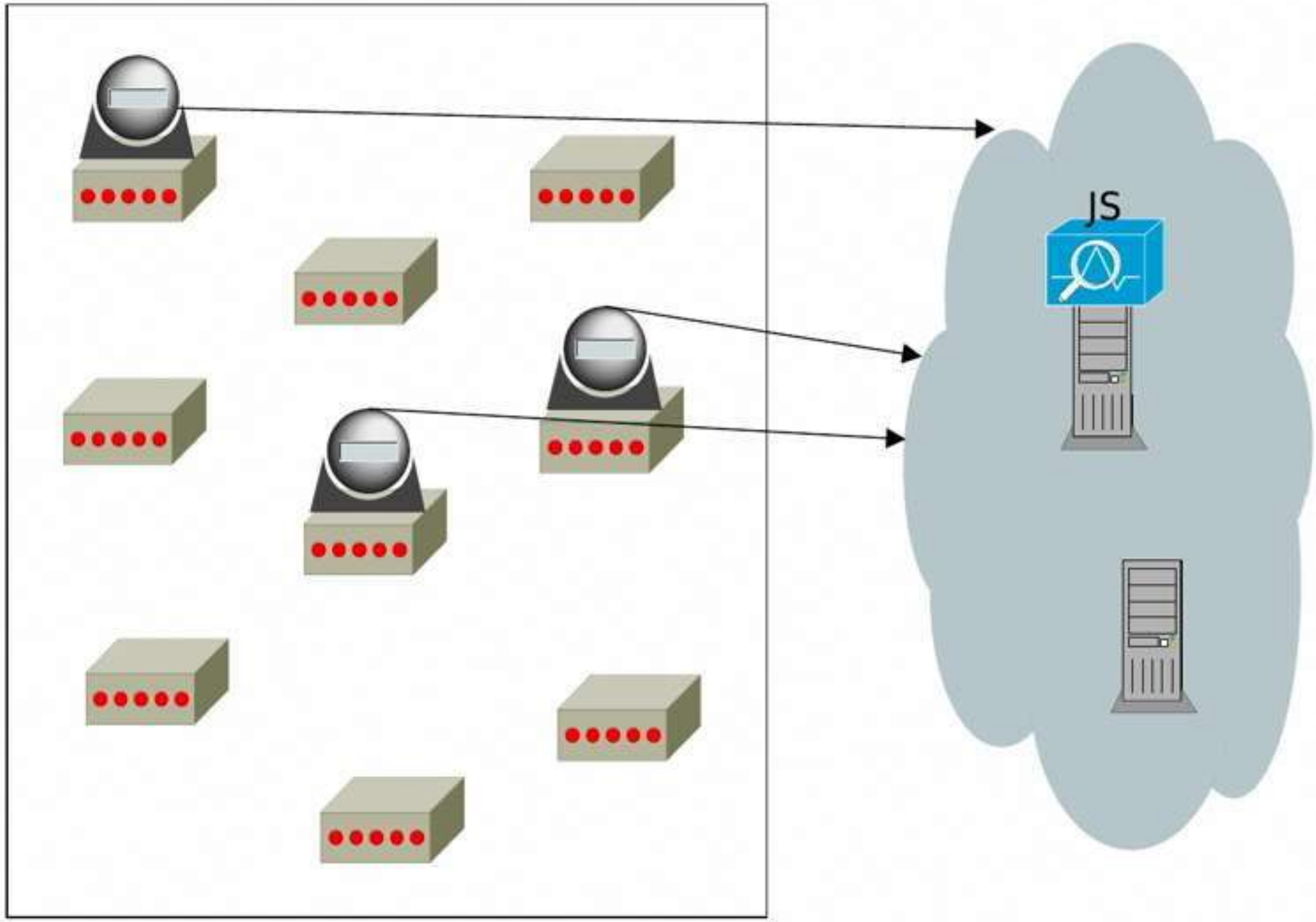
JavaScript VMs on IoT

- Samsung IoT.js
- Intel XDK
- DukServer
- Smart.js
- Node.js on IoT devices

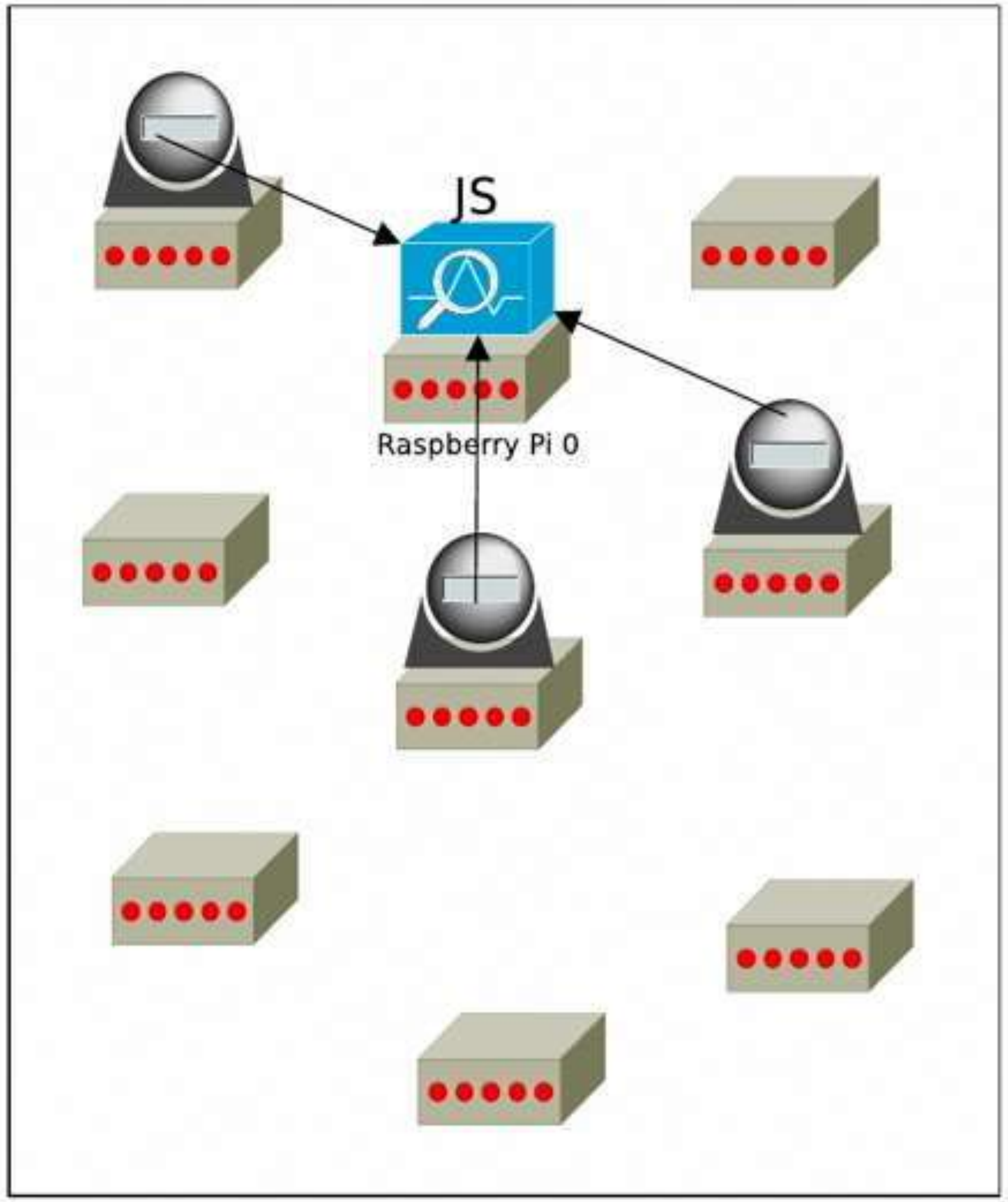
Scenario: Videosurveillance / Motion Detection



Scenario: Videosurveillance / Motion Detection



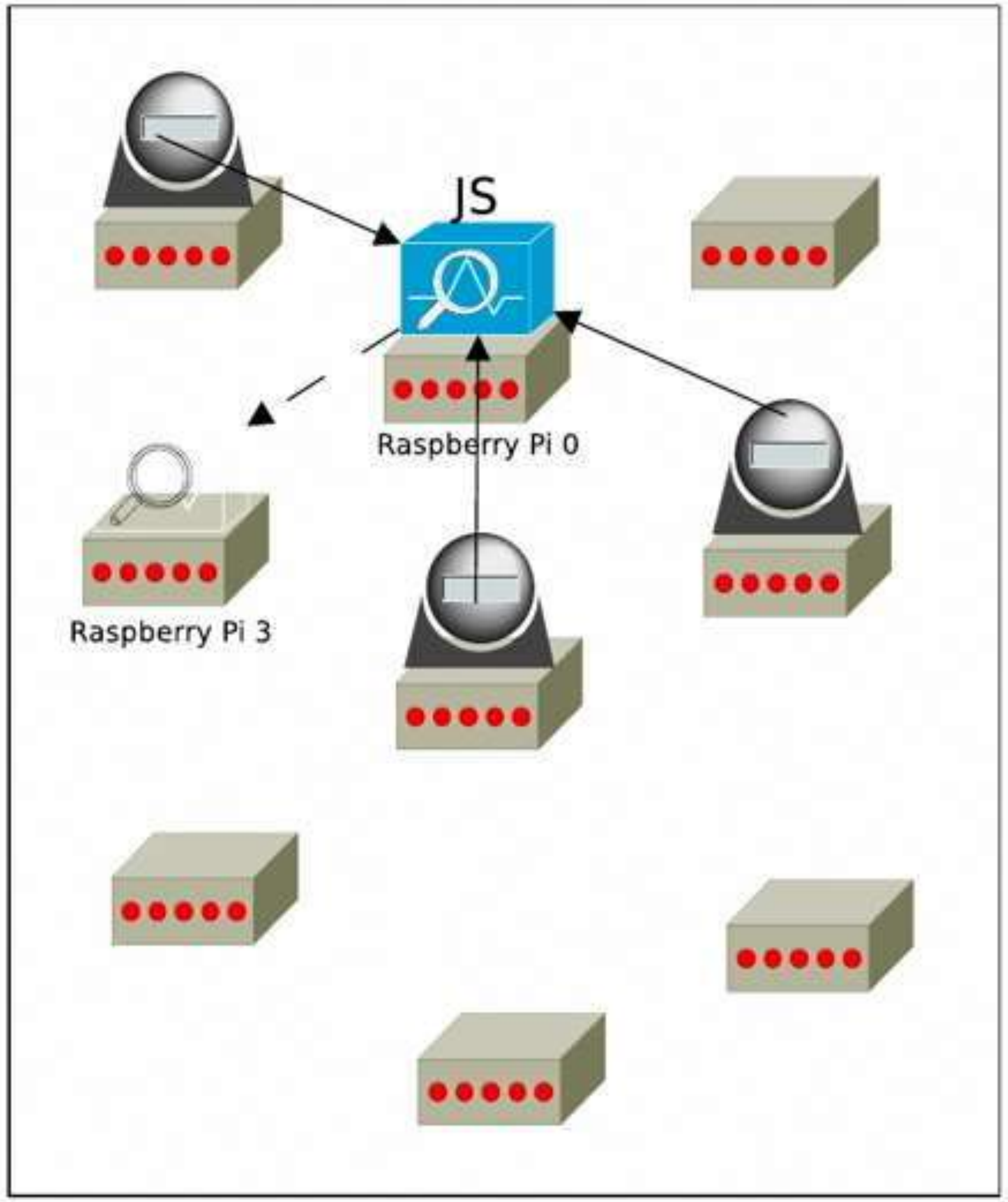
Scenario: Videosurveillance / Motion Detection



ThingsJS:

Executing High-Level Applications on IoT/Edge devices.

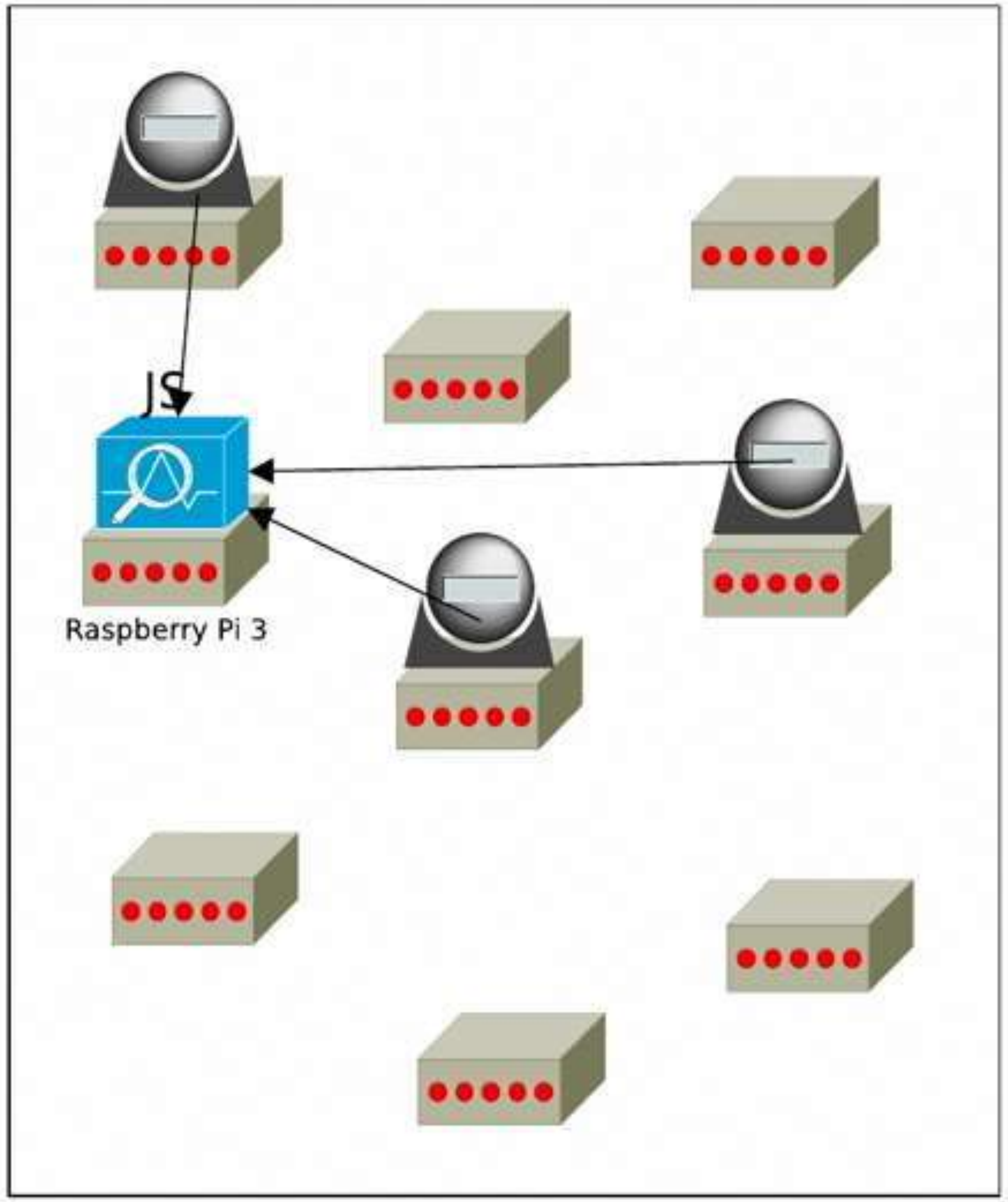
Scenario: Videosurveillance / Motion Detection



ThingsJS:

Executing High-Level Applications on IoT/Edge devices.

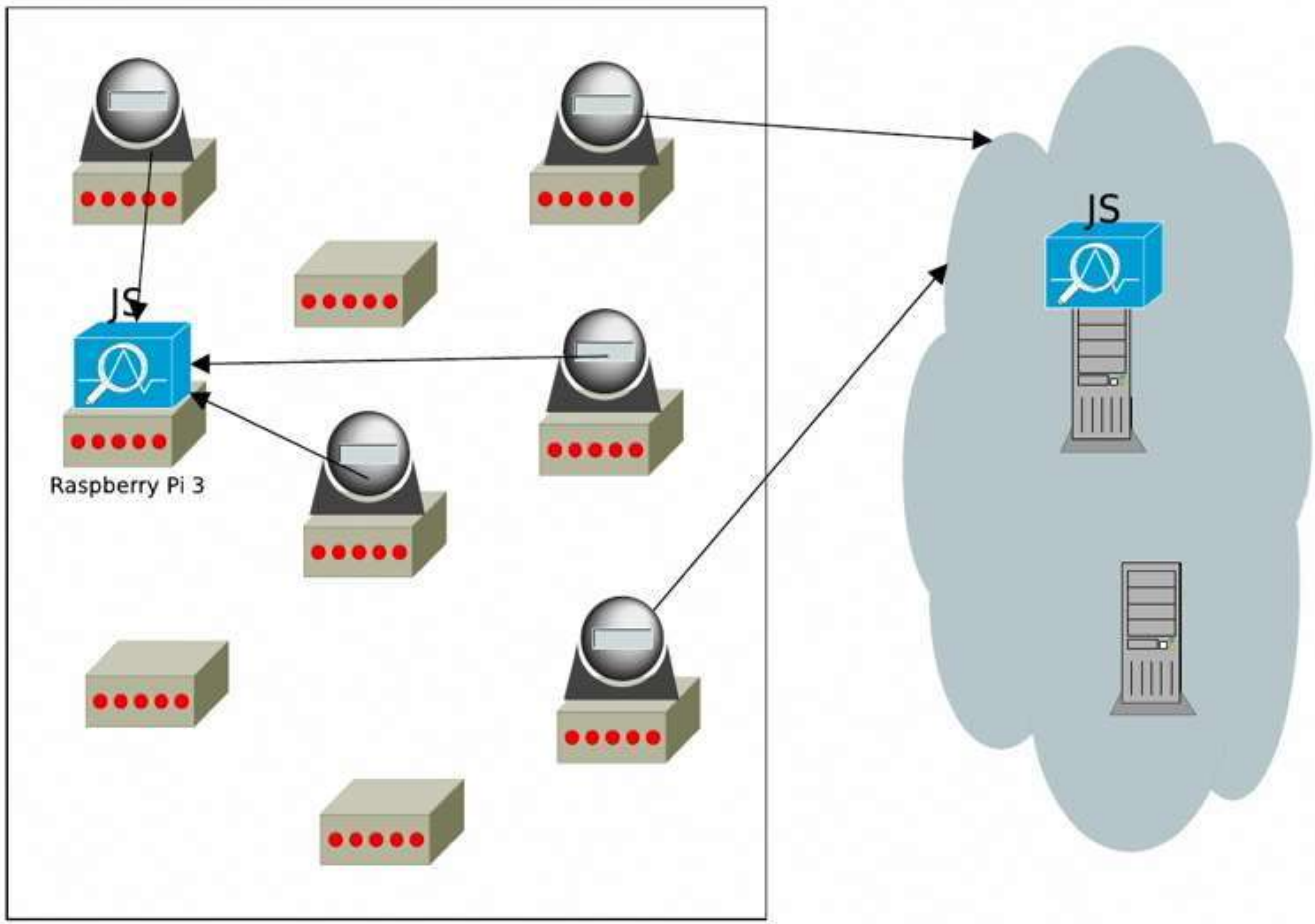
Scenario: Videosurveillance / Motion Detection



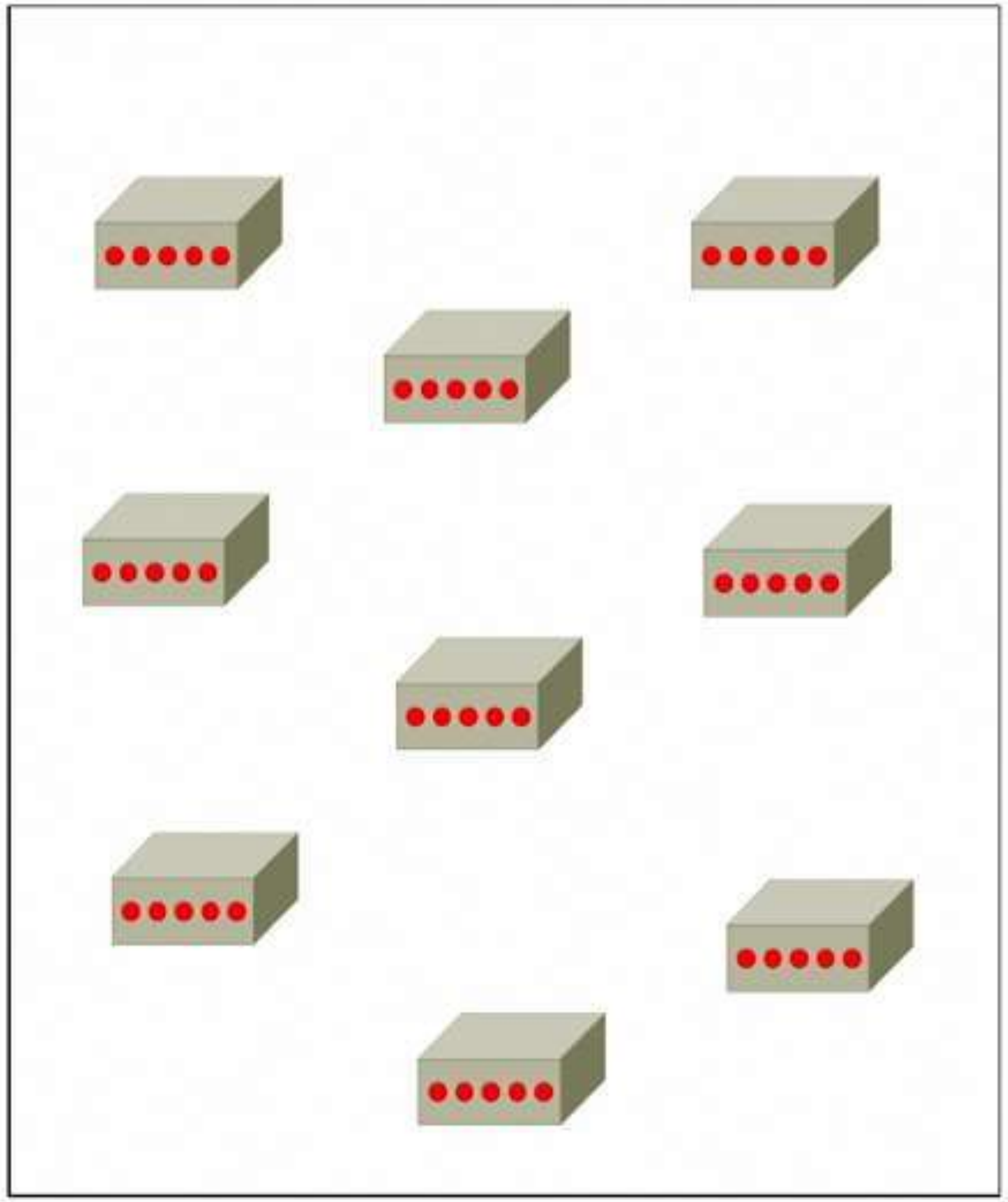
ThingsMigrate:

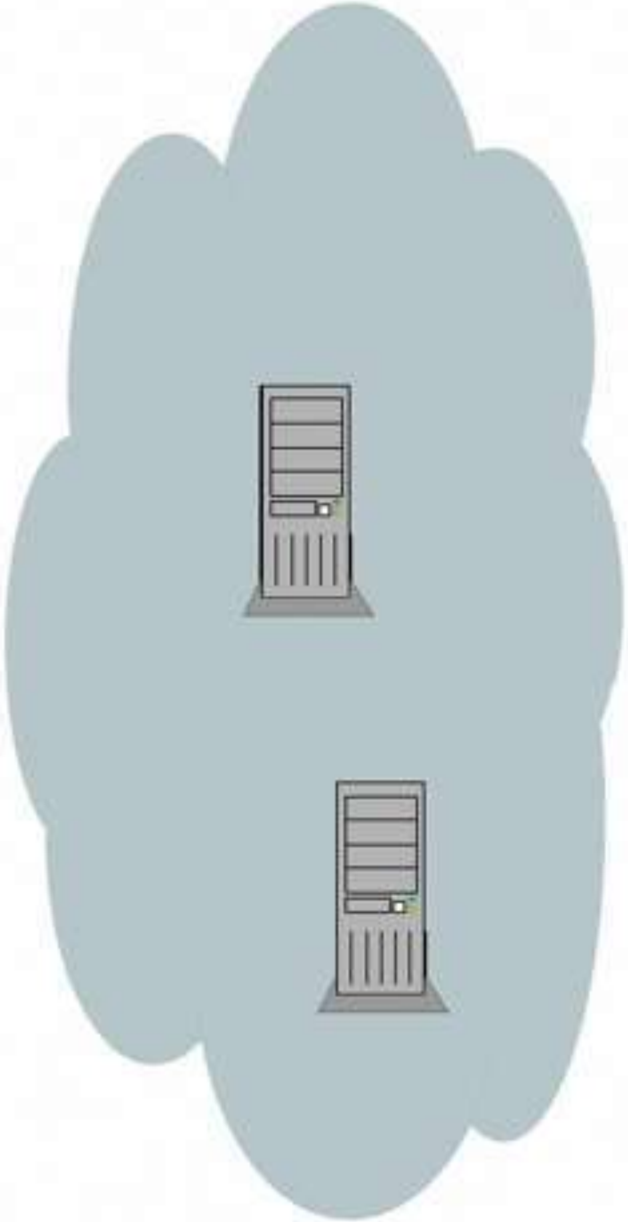
Transparently migrating JS applications between IoT/edge devices.

Scenario: Videosurveillance / Motion Detection

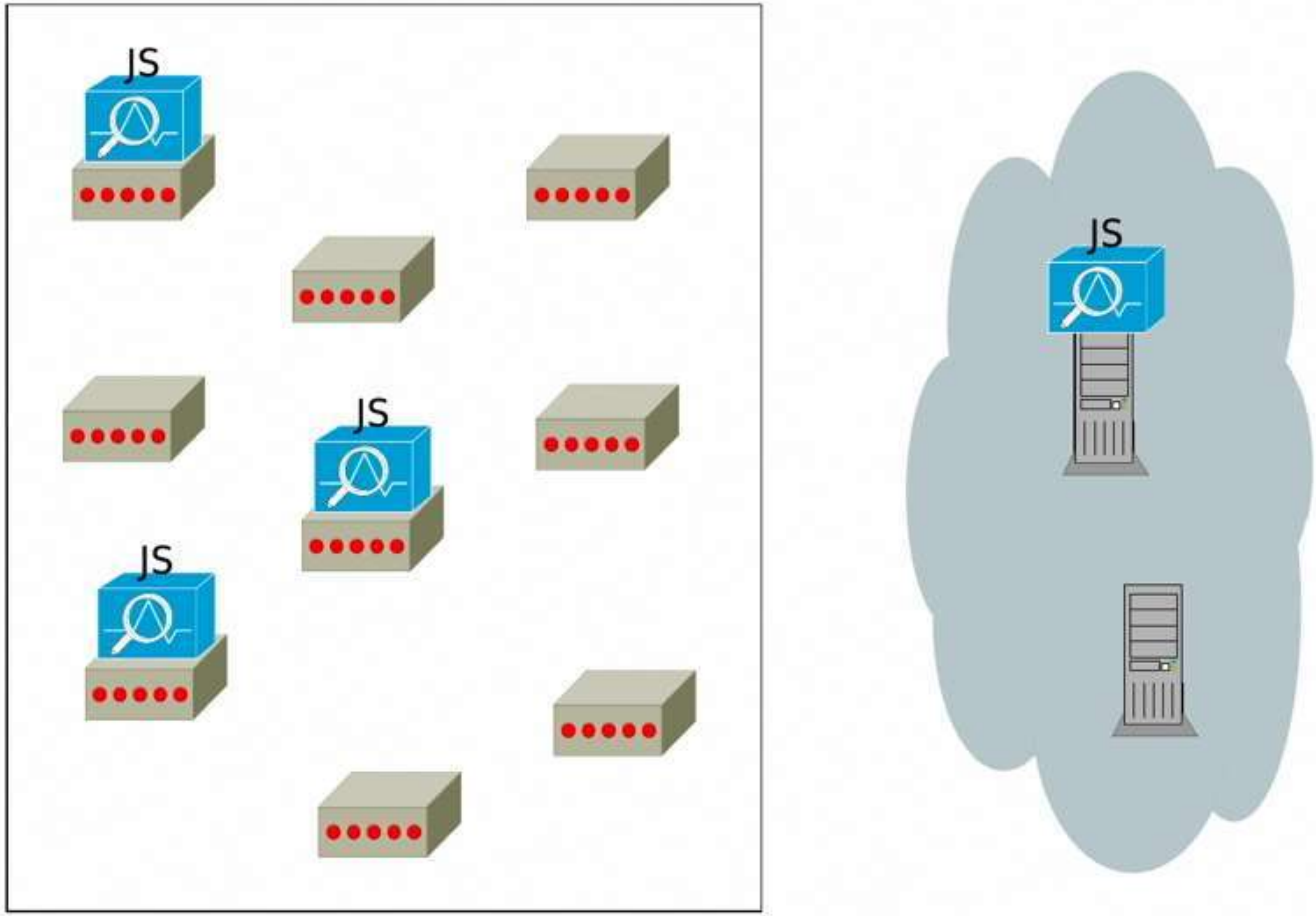


Migrating IoT Apps

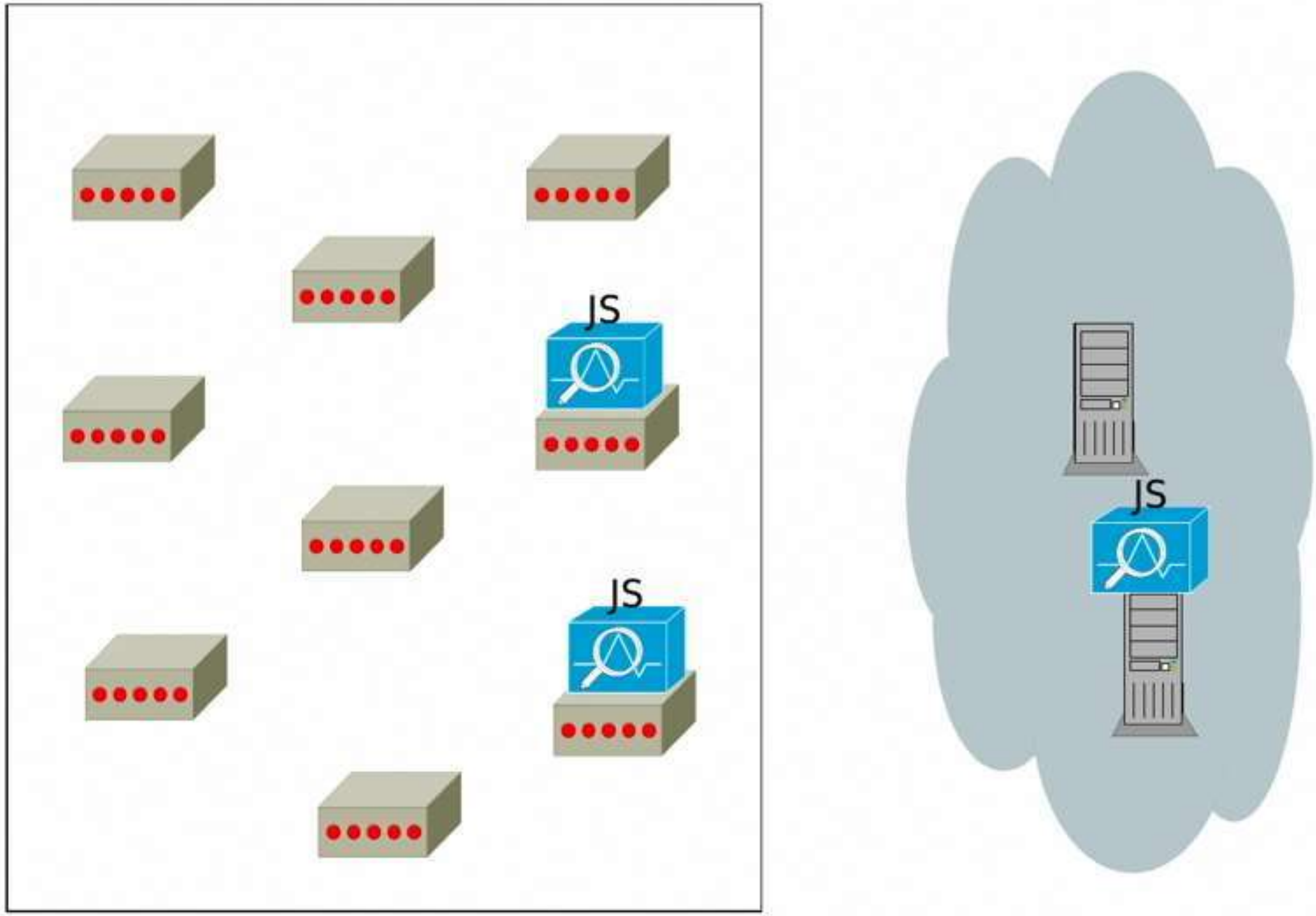




Migrating IoT Apps



Migrating IoT Apps



Challenges

Wide heterogeneity of devices, OS and JavaScript VMs!

Challenges

Wide heterogeneity of devices, OS and JavaScript VMs!

Challenge: capturing the state of the JavaScript app

1. Closures / data encapsulation in functions

```
function Counter() {
  var value = 0;

  return function() {
    value += 1;
    return value;
  }
};

var c = Counter();  // value in c is 0
console.log( c() ); // prints 1
console.log( c() ); // prints 2
```

Challenges

Wide heterogeneity of devices, OS and JavaScript VMs!

Challenge: capturing the state of the JavaScript app

1. Closures / data encapsulation in functions
2. Timers

```
function Counter() {
   var value = 0;

   return function() {
      value += 1;
      return value;
   }
};

var c = Counter();  // value in c is 0
console.log( c() ); // prints 1
console.log( c() ); // prints 2
setInterval(function() c(); ,1000);
```

Challenges

Wide heterogeneity of devices, OS and JavaScript VMs!

Challenge: capturing the state of the JavaScript app

1. Closures / data encapsulation in functions
2. Timers
3. Classes and prototypes

```
function Counter() {
   var value = 0;

   return function() {
      value += 1;
      return value;
   }
};

var c = Counter();  // value in c is 0
console.log( c() ); // prints 1
console.log( c() ); // prints 2
setInterval(function() { c(); },1000);}
```

Challenges

Wide heterogeneity of devices, OS and JavaScript VMs!

Challenge: capturing the state of the JavaScript app

1. Closures / data encapsulation in functions
2. Timers
3. Classes and prototypes
4. Asynchronous Model (Event-Based)

```
function Counter() {
   var value = 0;

   return function() {
      value += 1;
      return value;
   }
};

var c = Counter();  // value in c is 0
console.log( c() ); // prints 1
console.log( c() ); // prints 2
setInterval(function() { c(); },1000);}
```

Approach: Code Instrumentation & Reconstruction


```
function Counter() {
    var value = 0;

    return function() {
        value += 1;
        return value;
    }
};

var c = Counter();  // value in c is 0
console.log( c() ); // prints 1
console.log( c() ); // prints 2
setInterval(function() { c(); },1000);
```

Approach: Code Instrumentation & Reconstruction

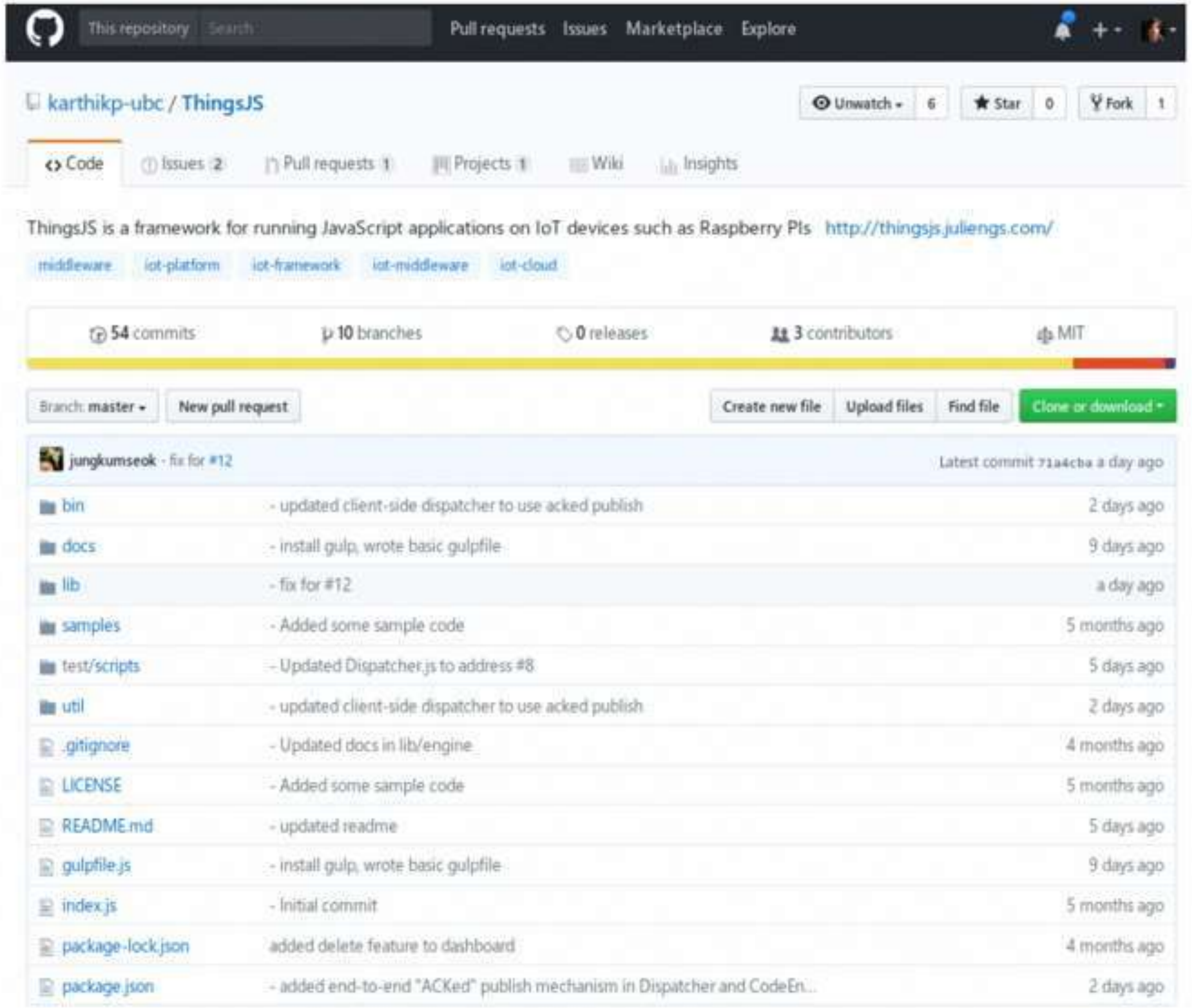

```
var global = new Scope("global");
function Counter() {
    counter. = new Scope(global, "Counter");
    var value = 0;
    counter.addVar("value", value);

    var anon1 = function() {
        anon1 = new Scope(createcounters, "anon1");
        value += 1;
        anon1.setVar("value", value);

        return value;
    }

    counter.addFunction("anon1", anon1);
    return anon1;
};
```

ThingsJS / ThingsMigrate: Open-Source on GitHub



Summary of Results

Test Bed

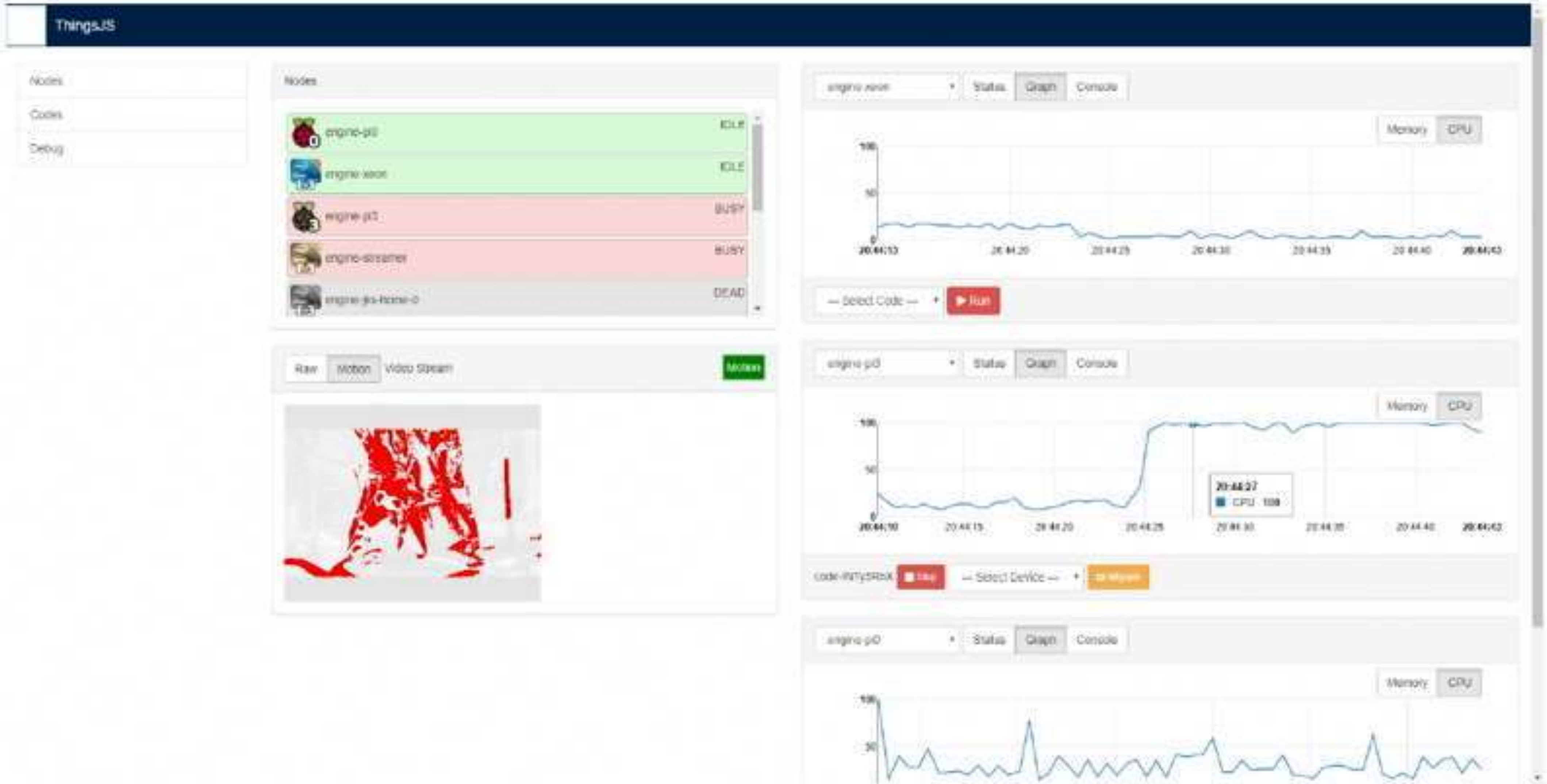
- Devices: Raspberry Pi 3, Raspberry Pi 0, “Cloud” server
- Several benchmarks

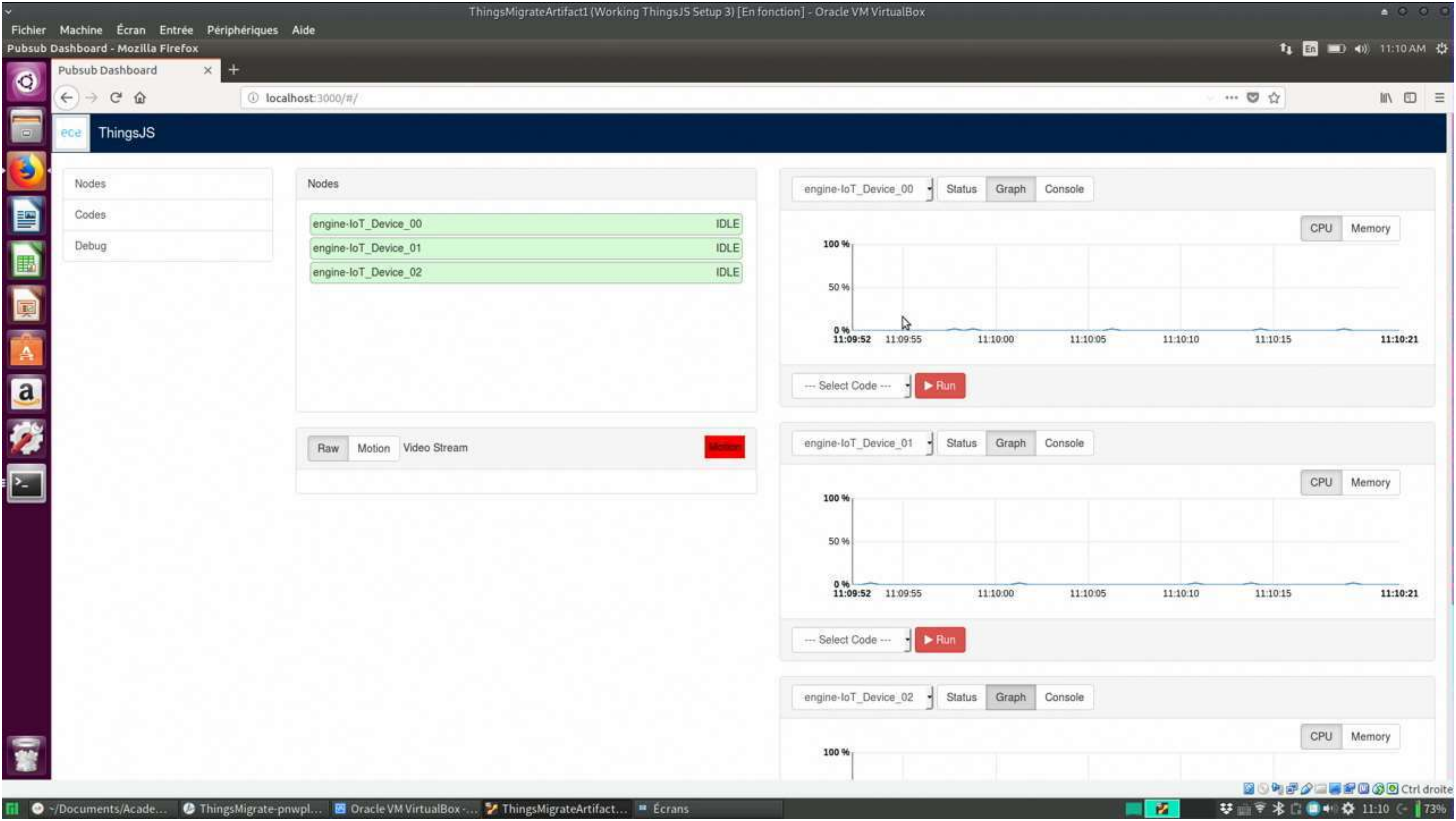
Overhead

- Execution time (CPU): ~30%
- Memory: significant overhead
- Support for multiple migrations without additional overhead
- *Implementation could be further optimized!*

ThingsMigrate: Paper accepted at ECOOP 2018

Demo


ThingsJS
Nodes
Codes
Debug
Status
Graph
Console
Memory
CPU
IDLE
BUSY
DEAD
Raw
Motion
Video Stream
Run


ThingsMigrateArtifact1 (Working ThingsJS Setup 3) [En fonction] - Oracle VM VirtualBox
Fichier Machine Écran Entrée Périphériques Aide
Pubsub Dashboard - Mozilla Firefox
11:10 AM
Pubsub Dashboard
localhost:3000/#/
ece ThingsJS
Nodes
Codes
Debug
Nodes
engine-IoT_Device_00 IDLE
engine-IoT_Device_01 IDLE
engine-IoT_Device_02 IDLE
Raw Motion Video Stream
engine-IoT_Device_00 Status Graph Console
CPU Memory
100 %
50 %
0 %
11:09:52 11:09:55 11:10:00 11:10:05 11:10:10 11:10:15 11:10:21
--- Select Code --- Run
engine-IoT_Device_01 Status Graph Console
CPU Memory
100 %
50 %
0 %
11:09:52 11:09:55 11:10:00 11:10:05 11:10:10 11:10:15 11:10:21
--- Select Code --- Run
engine-IoT_Device_02 Status Graph Console
CPU Memory
100 %
Ctrl droite
~/Documents/Acade... ThingsMigrate-pnwpl... Oracle VM VirtualBox -... ThingsMigrateArtifact... Écrans
11:10 73%

Team and Resources

Research Team

- Julien Gascon-Samson, PhD – NSERC Post-Doctoral Fellow
- Kumseok Jung – Master's Student
- Professor Karthik Pattabiraman – co-PI

Collaboration with Shivanshu Goyal and Armin Rezaiean-Asel (now at Microsoft)

Resources:

- ThingsJS: http://thingsjs.juliengs.com
- GitHub Repository: https://github.com/karthikp-ubc/ThingsJS

Work done in collaboration with Intel

PGo

Compiling Distributed Systems Specifications into Implementations

Matthew Do, Renato Costa, Brandon Zhang
Finn Hackett, Stewart Grant, Ivan Beschastnikh

Networks, Systems and Security

COMPUTER SCIENCE

UBC

PNW PLSE 2018

Distributed Systems are Hard

- Distributed systems are hard to **design** and **build**
- **Non-deterministic** sequence of events
- Components can **fail**

Google's data center, Council Bluffs, IA
https://www.google.com/about/datacenters/gallery

Distributed Systems are Everywhere

- Distributed systems are widely deployed [1]
- Failures can be very **costly**
  - DynamoDB's outage in 2015 caused downtime on Netflix, Reddit, etc [2]
  - S3's outage in 2017 caused loss of millions of dollars [3]

[1] Mark **Cavage**. 2013. *There's Just No Getting around It: You're Building a Distributed System*. Queue 11, 4, Pages 30 (April 2013)
[2] Fletcher **Babb**. *Amazon's AWS DynamoDB Experiences Outage, Affecting Netflix, Reddit, Medium, and More*. en-US. Sept. 2015
[3] Shannon **Vavra.** *Amazon outage cost S&P 500 companies $150M*. axios.com, Mar 3, 2017

Distributed Systems are Everywhere

- Distributed systems are widely deployed [1]

We need a better way to build reliable systems

- DynamoDB's outage in 2015 caused downtime on Netflix, Reddit, etc [2]
- S3's outage in 2017 caused loss of millions of dollars [3]

amazon web services

reddit

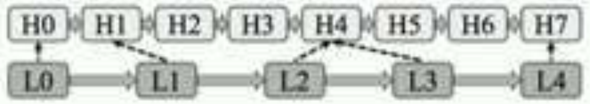
NETFLIX

[1] Mark **Cavage**. 2013. *There's Just No Getting around It: You're Building a Distributed System*. Queue 11, 4, Pages 30 (April 2013)
[2] Fletcher **Babb**. *Amazon's AWS DynamoDB Experiences Outage, Affecting Netflix, Reddit, Medium, and More*. en-US. Sept. 2015
[3] Shannon **Vavra.** *Amazon outage cost S&P 500 companies $150M*. axios.com, Mar 3, 2017

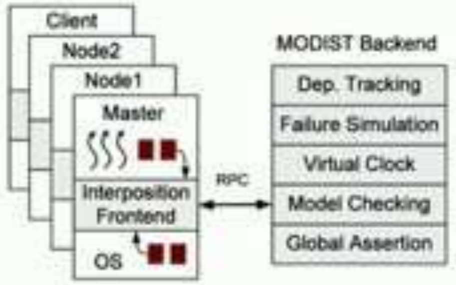
Related Work

- **`Verdi`** reduces proof burden by automatically handling failures [PLDI'15]

- **`IronFleet`** provides a framework to write specifications and implementations [SOSP'15]



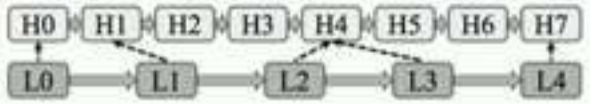
- **`MODIST`** checks the implementation rather than a specification [NSDI'09]



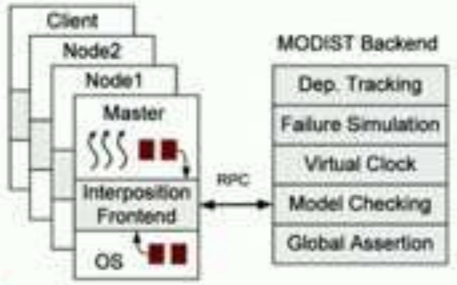
Related Work

- **`Verdi`** reduces proof burden by automatically handling failures [PLDI'15]

- **`IronFleet`** provides a framework to write specifications and implementations [SOSP'15]

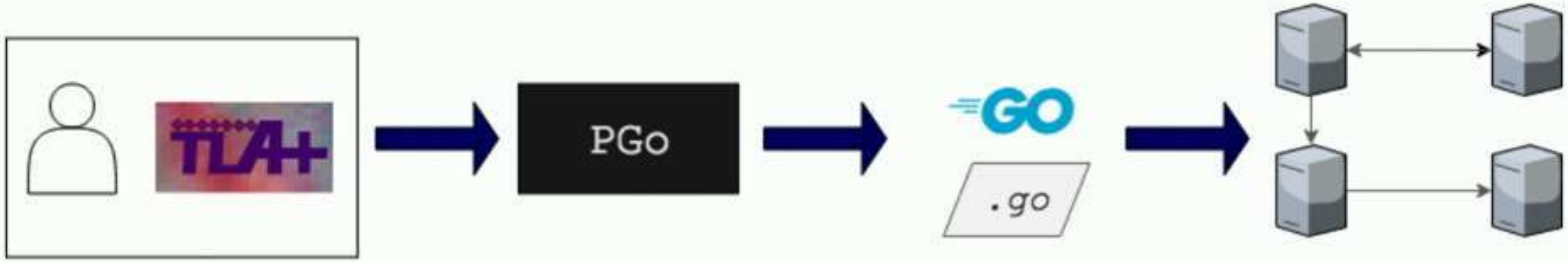


- **`MODIST`** checks the implementation rather than a specification [NSDI'09]



Hard to scale to large systems,
or require a lot of work from developers

PGo: Compiling Distributed Systems



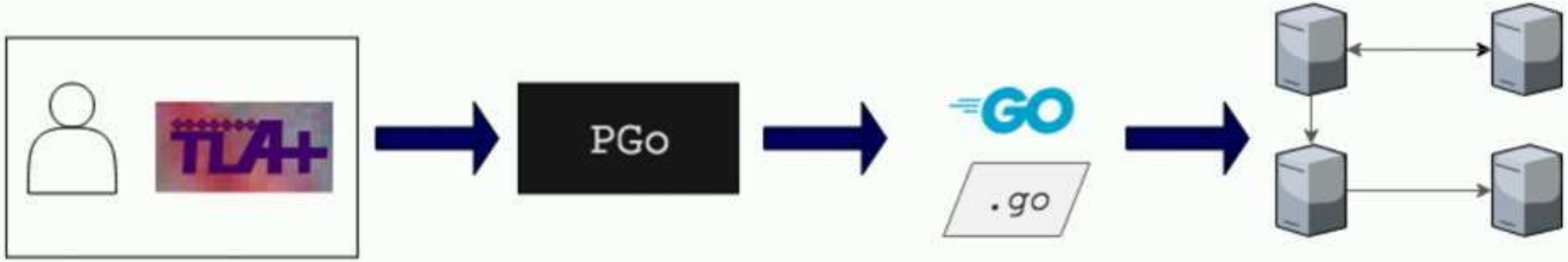
Developer writes **specification**

PGo compiles it to a *matching implementation*

Source is compiled

Verified Distributed System!

PGo: Compiling Distributed Systems



Developer writes **specification**

PGo compiles it to a *matching implementation*

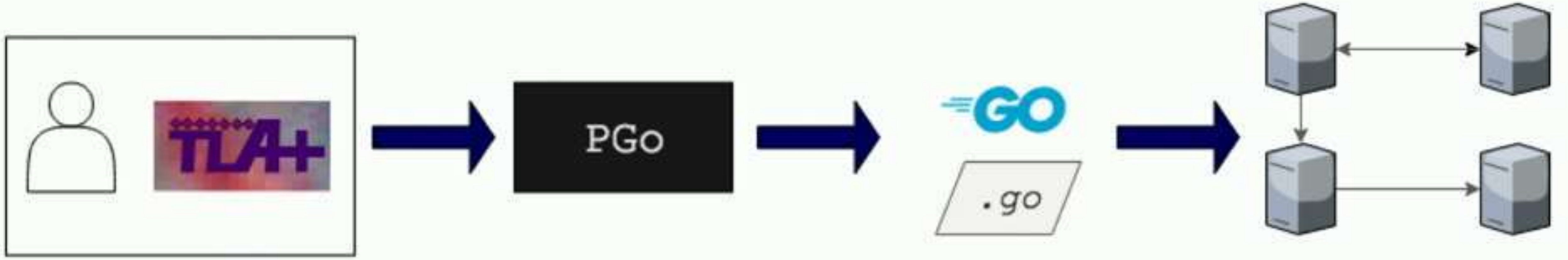
Source is compiled

Verified Distributed System!

✔ **TLC**

✔ **PGo**

PGo: Compiling Distributed Systems



Developer writes **specification** — PGo compiles it to a *matching implementation* — Source is compiled — **Verified Distributed System!**

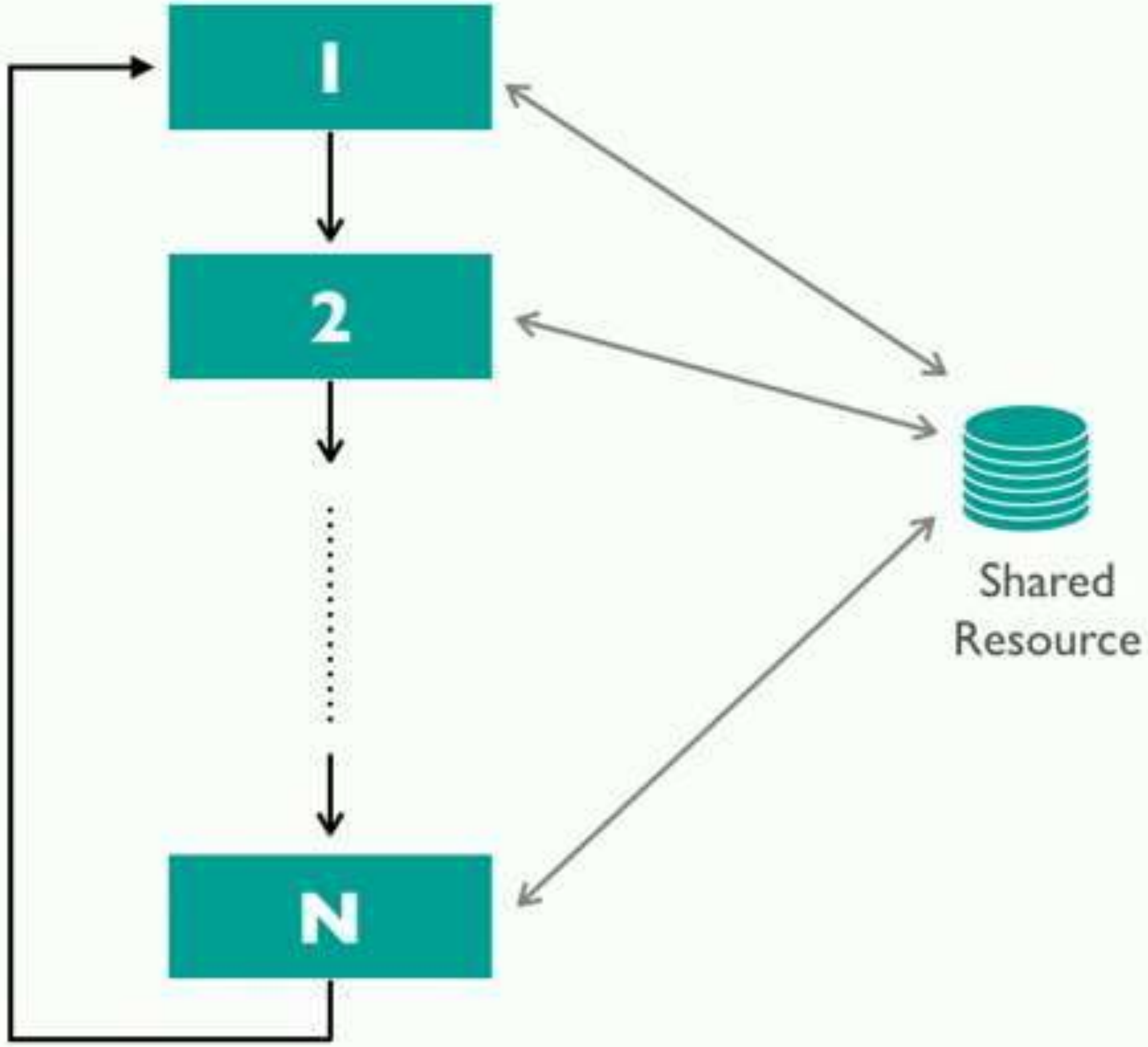
✔ **TLC** ✔ **PGo**

Transition from *design* (specification) to *implementation* is **automated**

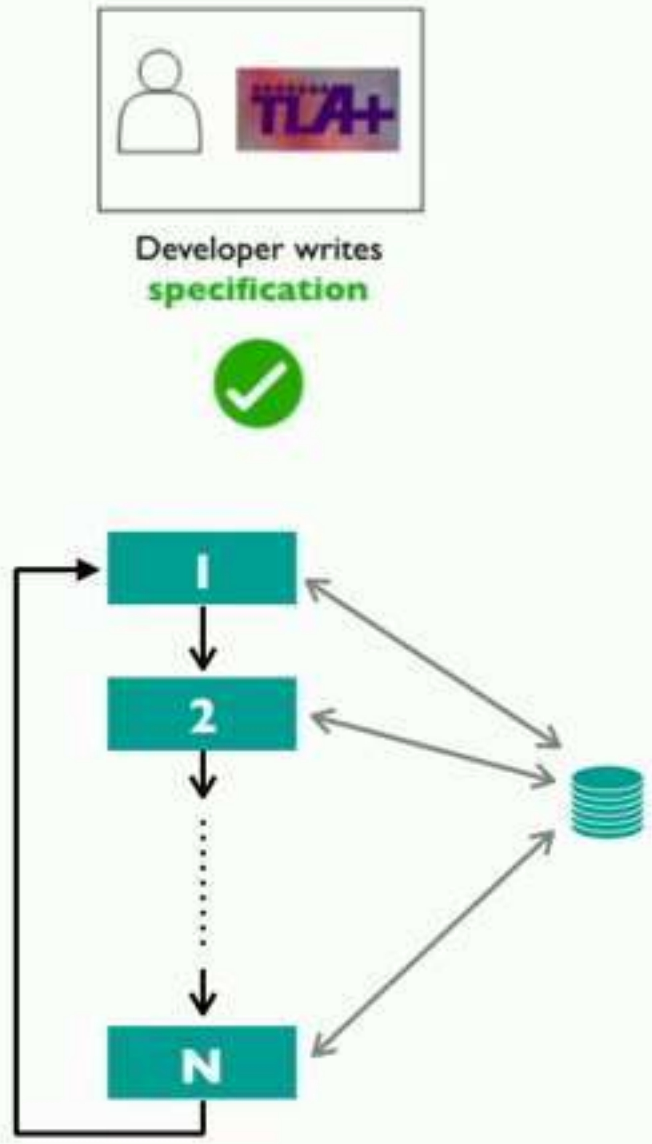
PGo Workflow: (1) Example System

Round-Robin Resource Sharing

Developer writes **specification**

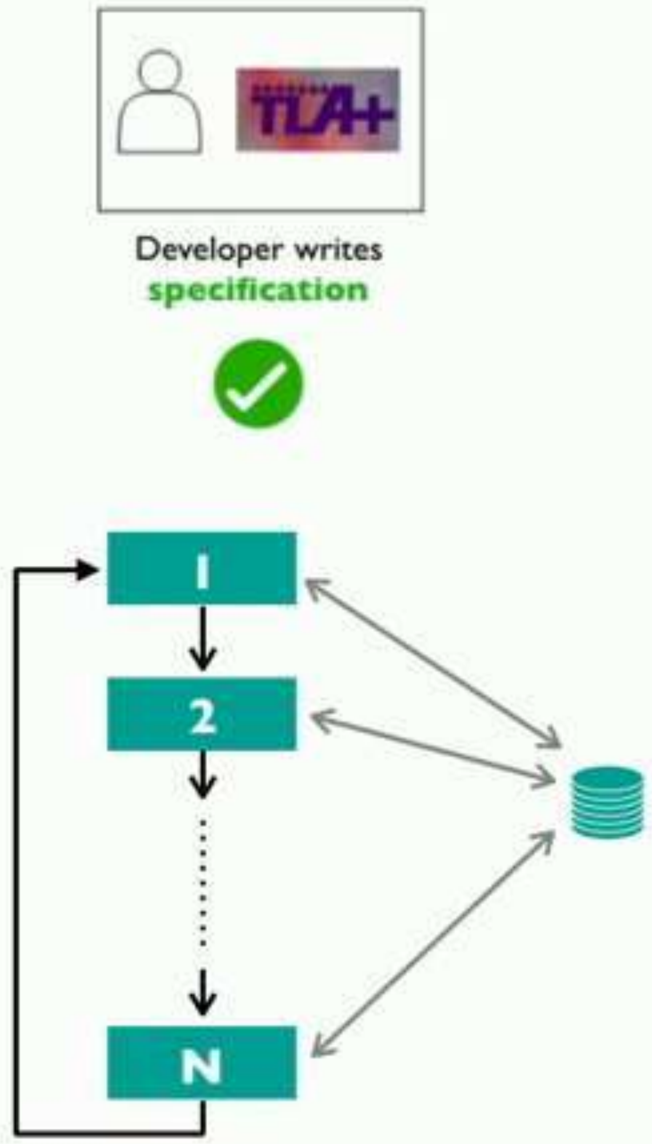


PGo Workflow: (1) PlusCal Spec



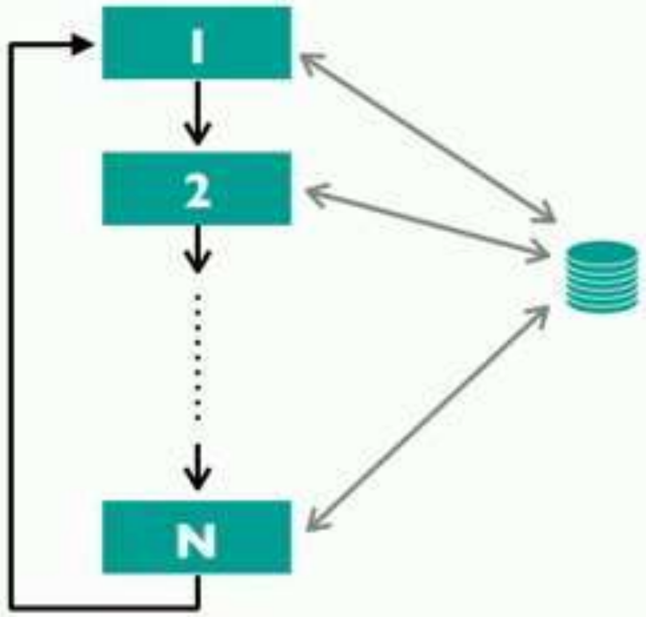
PGo Workflow: (1) PlusCal Spec

```
CONSTANTS procs, iters
```



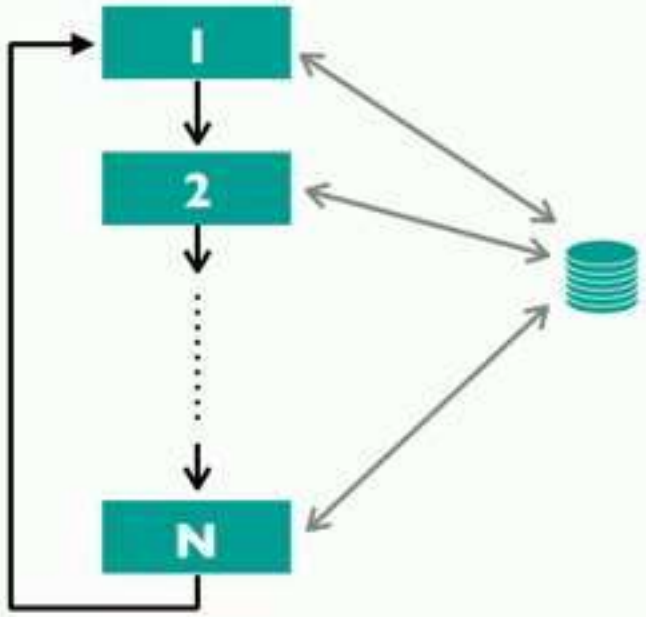
PGo Workflow: (1) PlusCal Spec

Developer writes **specification**



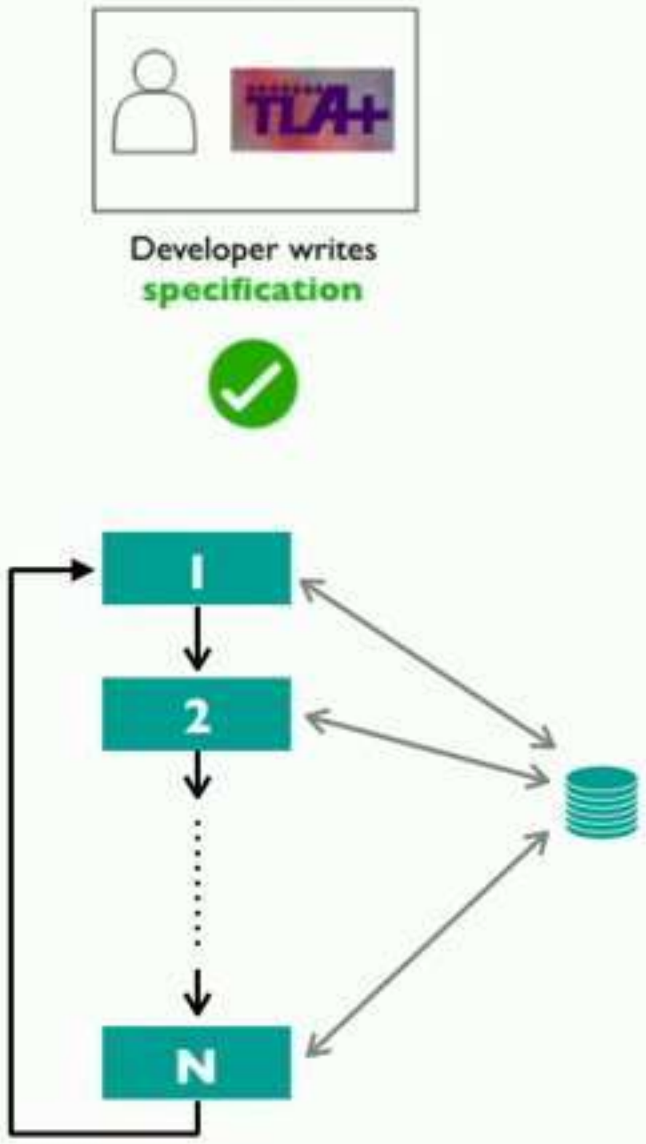

```
CONSTANTS procs, iters
(*
-- algorithm RoundRobin {
    variables counter = 0,
              token = 0;
```

PGo Workflow: (1) PlusCal Spec



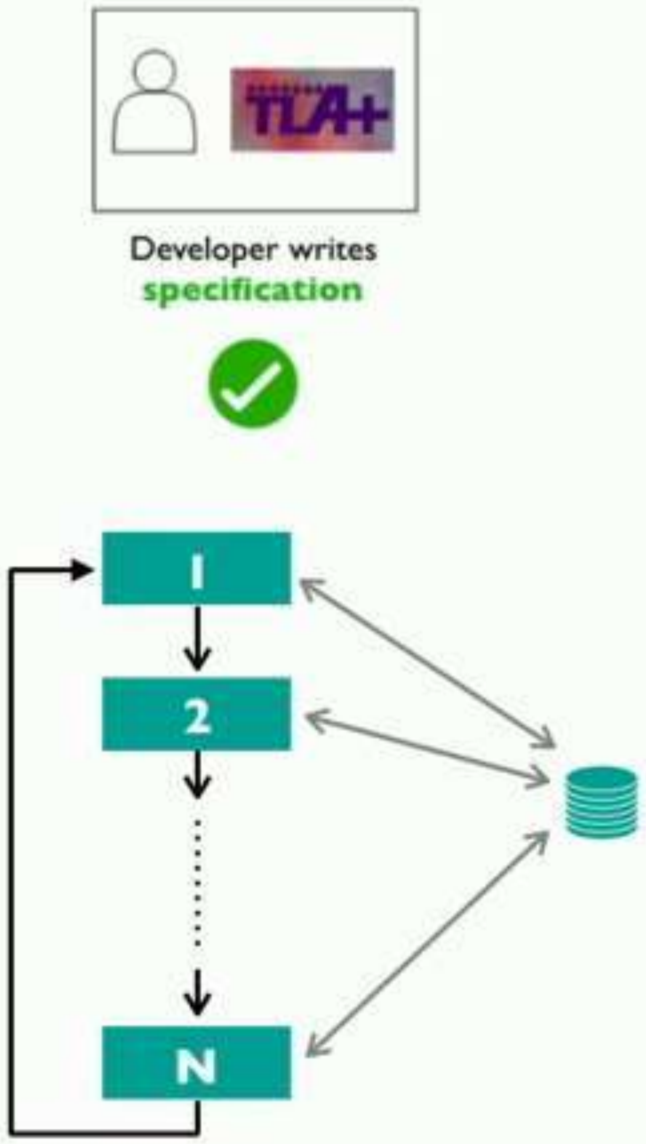

```
CONSTANTS procs, iters
(*
-- algorithm RoundRobin {
    variables counter = 0,
              token = 0;
fair process (P \in 0..procs-1)
variable i = 0;
```

PGo Workflow: (1) PlusCal Spec



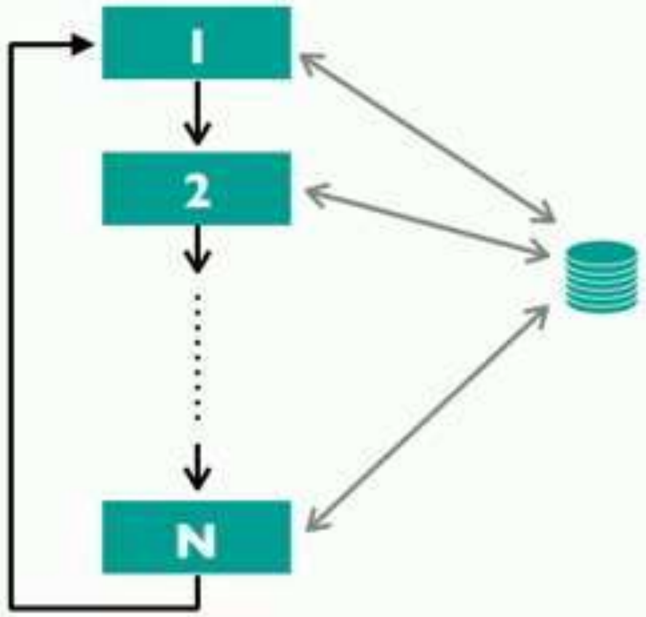

```
CONSTANTS procs, iters
(*
-- algorithm RoundRobin {
    variables counter = 0,
              token = 0;
fair process (P \in 0..procs-1)
variable i = 0;
{
    w: while ( i < iters) {
        inc: await token = self;
             counter := counter + 1;
             token := (self + 1) % procs;
             i := i + 1;
    }
}}
```

PGo Workflow: (1) PlusCal Spec



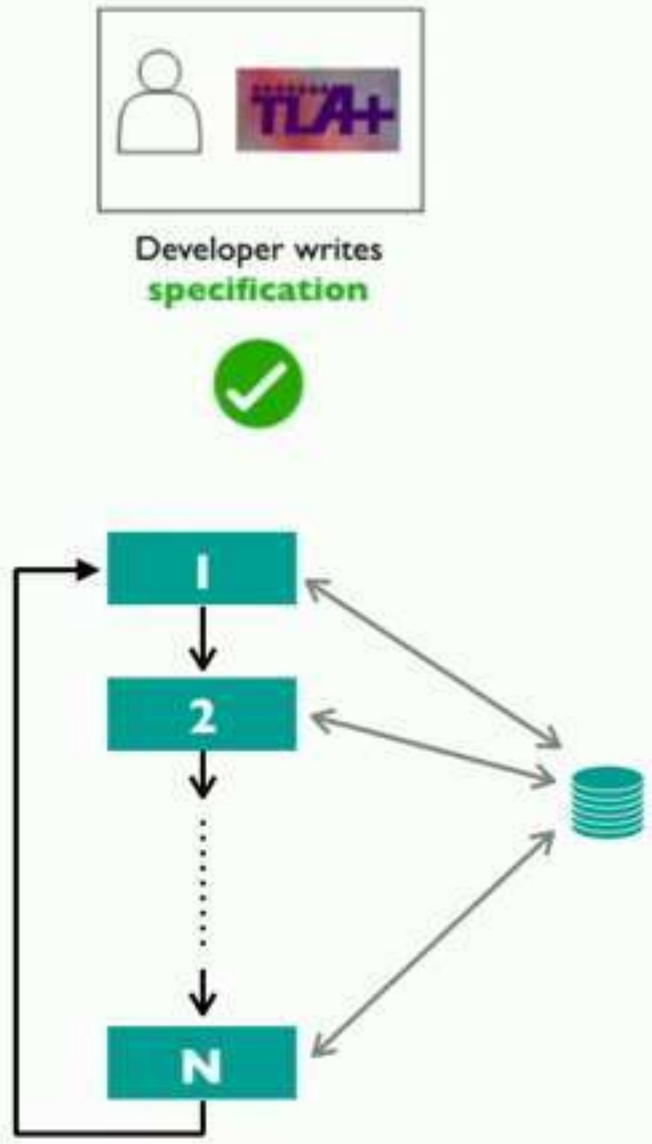

```
CONSTANTS procs, iters
(*
-- algorithm RoundRobin {
    variables counter = 0,
              token = 0;
fair process (P \in 0..procs-1)
variable i = 0;
{
    w: while ( i < iters) {
        inc: await token = self;
             counter := counter + 1;
             token := (self + 1) % procs;
             i := i + 1;
    }
}}
```

PGo Workflow: (1) PlusCal Spec



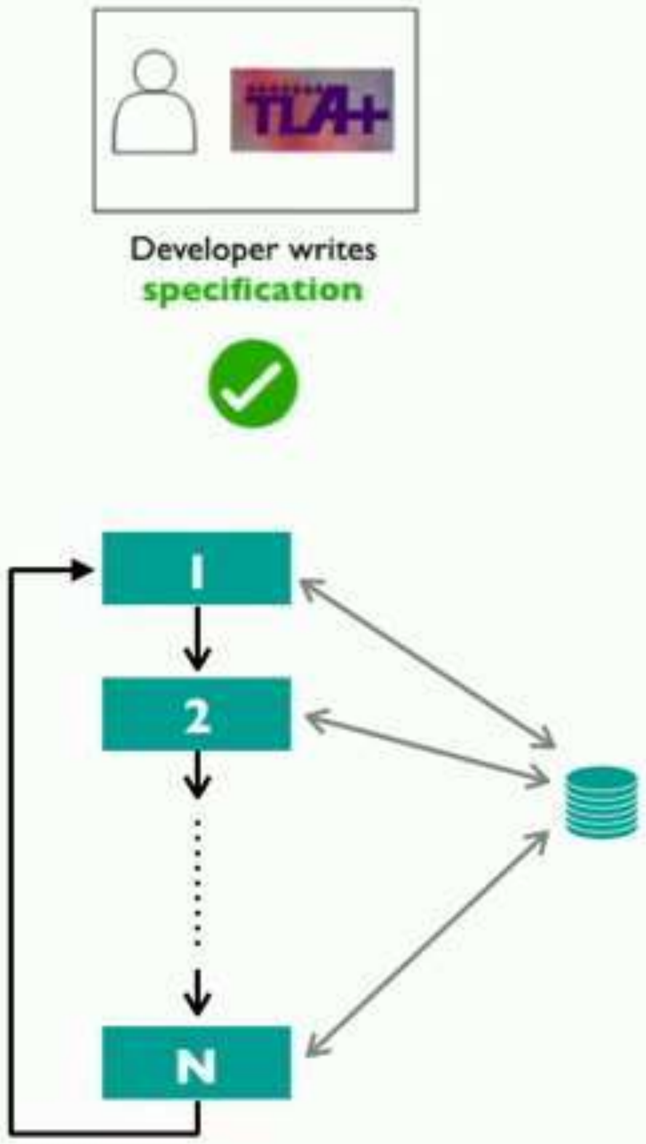

```
CONSTANTS procs, iters
(*
-- algorithm RoundRobin {
    variables counter = 0,
              token = 0;
fair process (P \in 0..procs-1)
variable i = 0;
{
    w: while ( i < iters) {
        inc: await token = self;
             counter := counter + 1;
             token := (self + 1) % procs;
             i := i + 1;
    }
}}
```

PGo Workflow: (1) PlusCal Spec



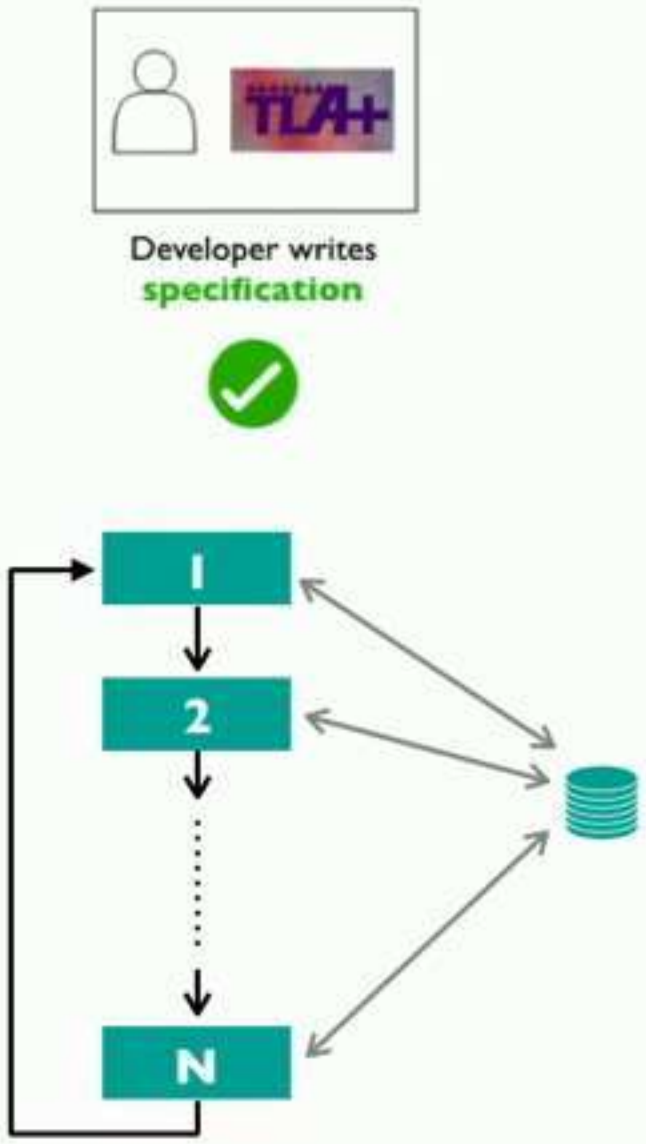

```
CONSTANTS procs, iters
(*
-- algorithm RoundRobin {
    variables counter = 0,
              token = 0;
fair process (P \in 0..procs-1)
variable i = 0;
{
    w: while ( i < iters) {
        inc: await token = self;
             counter := counter + 1;
             token := (self + 1) % procs;
             i := i + 1;
    }
}}
```

PGo Workflow: (1) PlusCal Spec




```
CONSTANTS procs, iters
(*
-- algorithm RoundRobin {
    variables counter = 0,
              token = 0;
fair process (P \in 0..procs-1)
variable i = 0;
{
    w: while ( i < iters) {
        inc: await token = self;
             counter := counter + 1;
             token := (self + 1) % procs;
             i := i + 1;
    }
}}
```

PGo Workflow: (1) Properties of our System



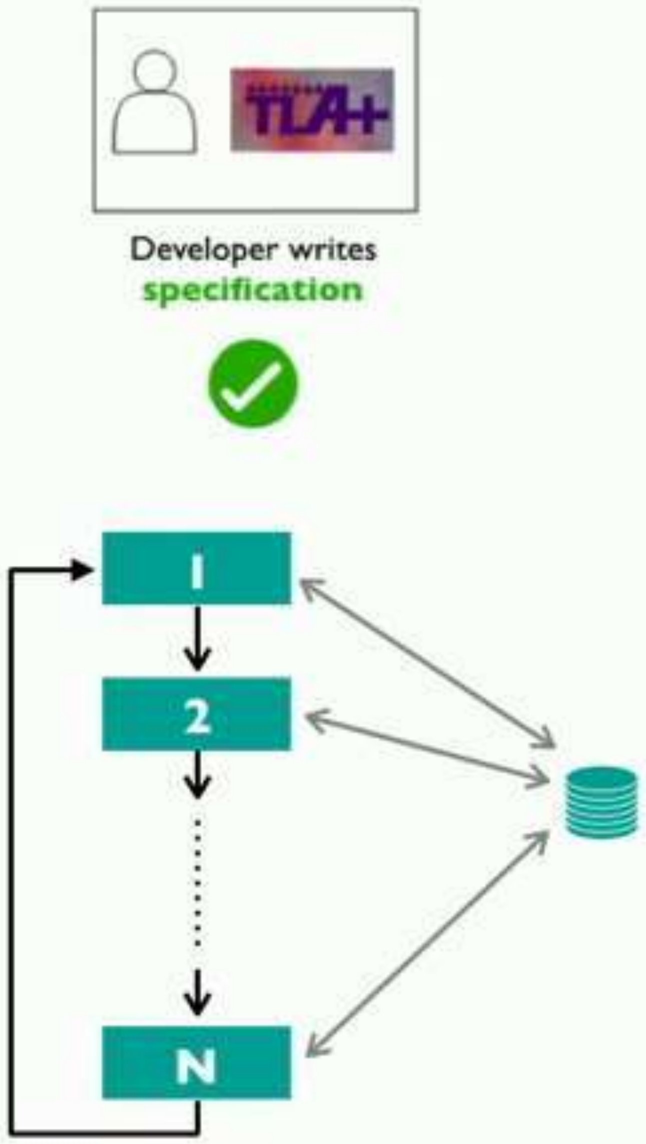
Invariants

Token is within bounds	`token \in 0..procs-1`

Properties

Counter Converges	`Termination => (counter = procs * iters)`
Processes Get the Token	`\A p \in \ProcSet : <>(token = p)`

PGo Workflow: (1) Verifying



Model Checked with TLC!

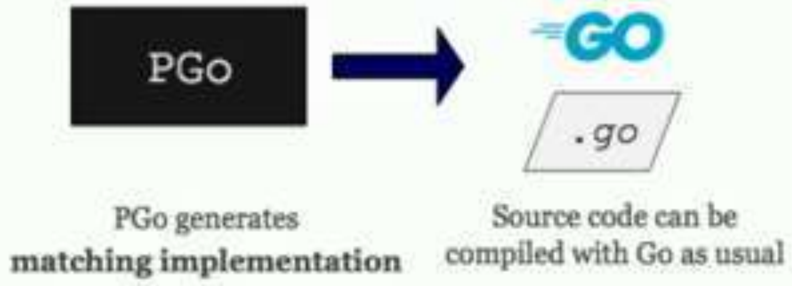
Model Checking Results

General

Start time:	Fri May 04 01:45:30 PDT 2018
End time:	Fri May 04 01:45:37 PDT 2018
TLC mode:	Breadth-first search
Last checkpoint time:	
Current status:	Not running
Errors detected:	No errors

PGo Workflow: (2) Compilation

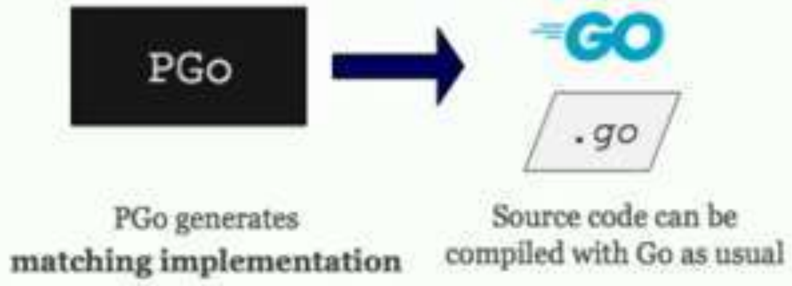
- `counter` is global: semantics need to be maintained
  - Runtime manages state across processes
- Labels are atomic
  - Processes coordinate access to atomic blocks
- High-level concepts such as `await`
  - Lock and check predicate




```
fair process (P \in 0..procs-1)
variable i = 0;
{
    w: while ( i < iters) {
        inc: await token = self;
             counter := counter + 1;
             token := (self + 1) % procs;
             i := i + 1;
    }
}}
```

PGo Workflow: (2) Compilation

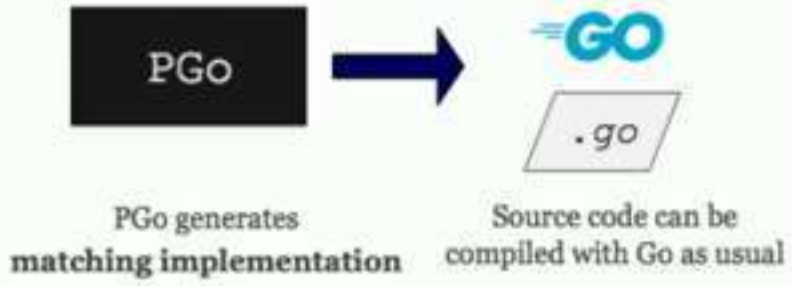
- `counter` is global: semantics need to be maintained
  - Runtime manages state across processes
- Labels are atomic
  - Processes coordinate access to atomic blocks
- High-level concepts such as `await`
  - Lock and check predicate




```
fair process (P \in 0..procs-1)
variable i = 0;
{
    w: while ( i < iters) {
        inc: await token = self;
             counter := counter + 1;
             token := (self + 1) % procs;
             i := i + 1;
    }
}}
```

PGo Workflow: (2) Compilation

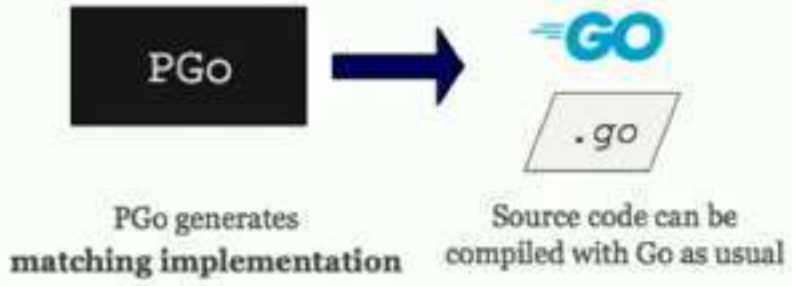
- `counter` is global: semantics need to be maintained
  - Runtime manages state across processes
- Labels are atomic
  - Processes coordinate access to atomic blocks
- High-level concepts such as `await`
  - Lock and check predicate




```
fair process (P \in 0..procs-1)
variable i = 0;
{
    w: while ( i < iters) {
        inc: await token = self;
             counter := counter + 1;
             token := (self + 1) % procs;
             i := i + 1;
    }
}}
```

PGo Workflow: (2) Compilation

- `counter` is global: semantics need to be maintained
  - Runtime manages state across processes
- Labels are atomic
  - Processes coordinate access to atomic blocks
- High-level concepts such as `await`
  - Lock and check predicate



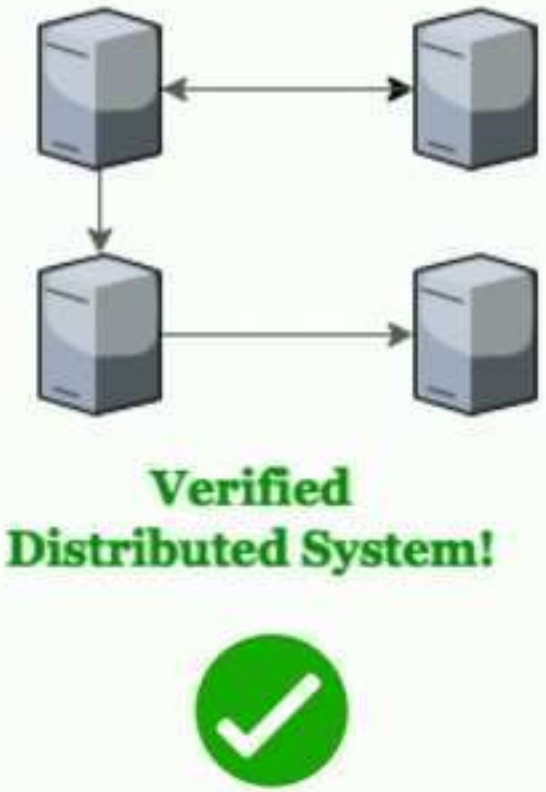

```
fair process (P \in 0..procs-1)
variable i = 0;
{
    w: while ( i < iters) {
        inc: await token = self;
             counter := counter + 1;
             token := (self + 1) % procs;
             i := i + 1;
    }
}}
```

PGo Workflow: (3) Using Compiled Code

- Generated Go code can run as any of the processes defined in PlusCal

```
$ ./counter
Usage: ./counter process(argument) ip:port


$ ./counter 'P(1)' 192.168.1.80:2222
```



Current Status

- PGo is currently able to compile **concurrent** and **distributed** systems

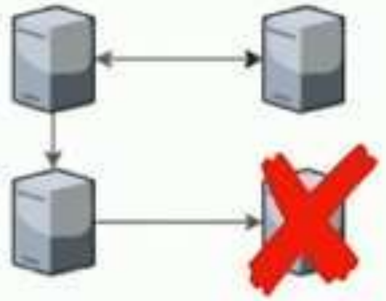
- Support for different strategies to deal with global state in a distributed system

- Compiles simple distributed applications

    - Example: ~30 lines of PlusCal generates ~80 lines of Go source code

Work in Progress

- Support a larger subset of PlusCal/TLA+
- Generating distributed systems that are fault tolerant
- Make it easy for developers to change generated code



MODULE *SyncQueue*

CONSTANT *Message*
VARIABLES *in*, *out*

$Internal(q) \triangleq$ INSTANCE *SyncQueueInternal*

$Fifo \triangleq \exists q : Internal(q)!FifoI$

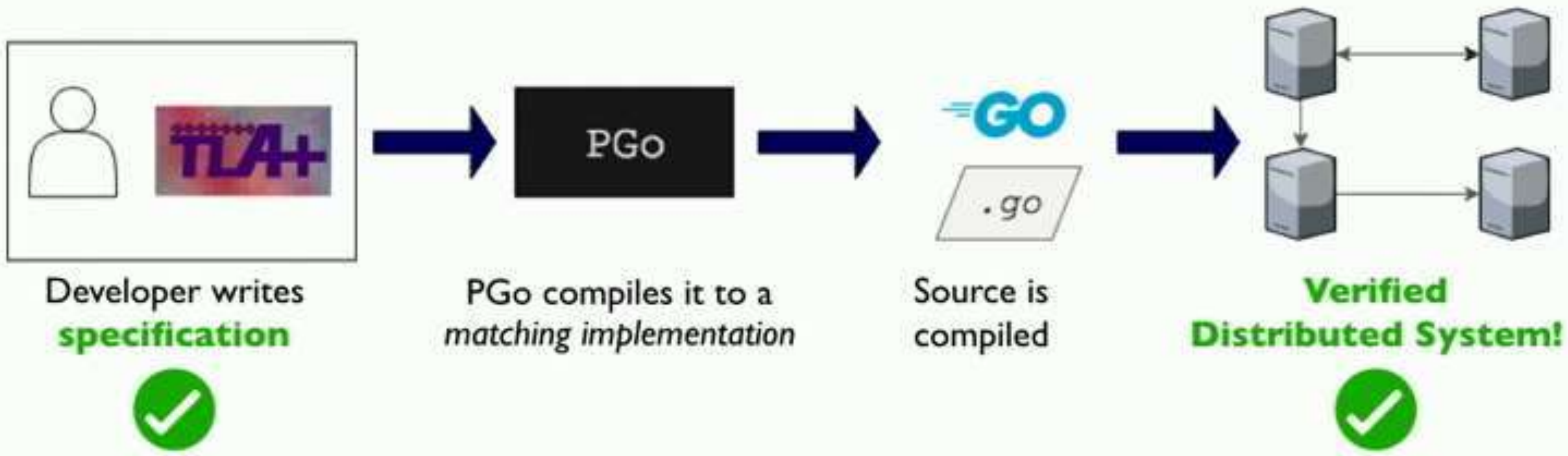
Limitations

- Specifications are very high level: not everything can be compiled efficiently

- May require developers to insert annotations when PGo cannot infer required information (e.g., types)

- Both the PGo compiler and the associated runtime need to be trusted in order to claim correctness

Conclusion — PGo: Compiling Verified Distributed Systems



Bridging the gap between design and implementation of a distributed system

Writing verified distributed systems easier to build

https://github.com/ubc-nss/pgo

Which bugs and tests should we use in experiments?

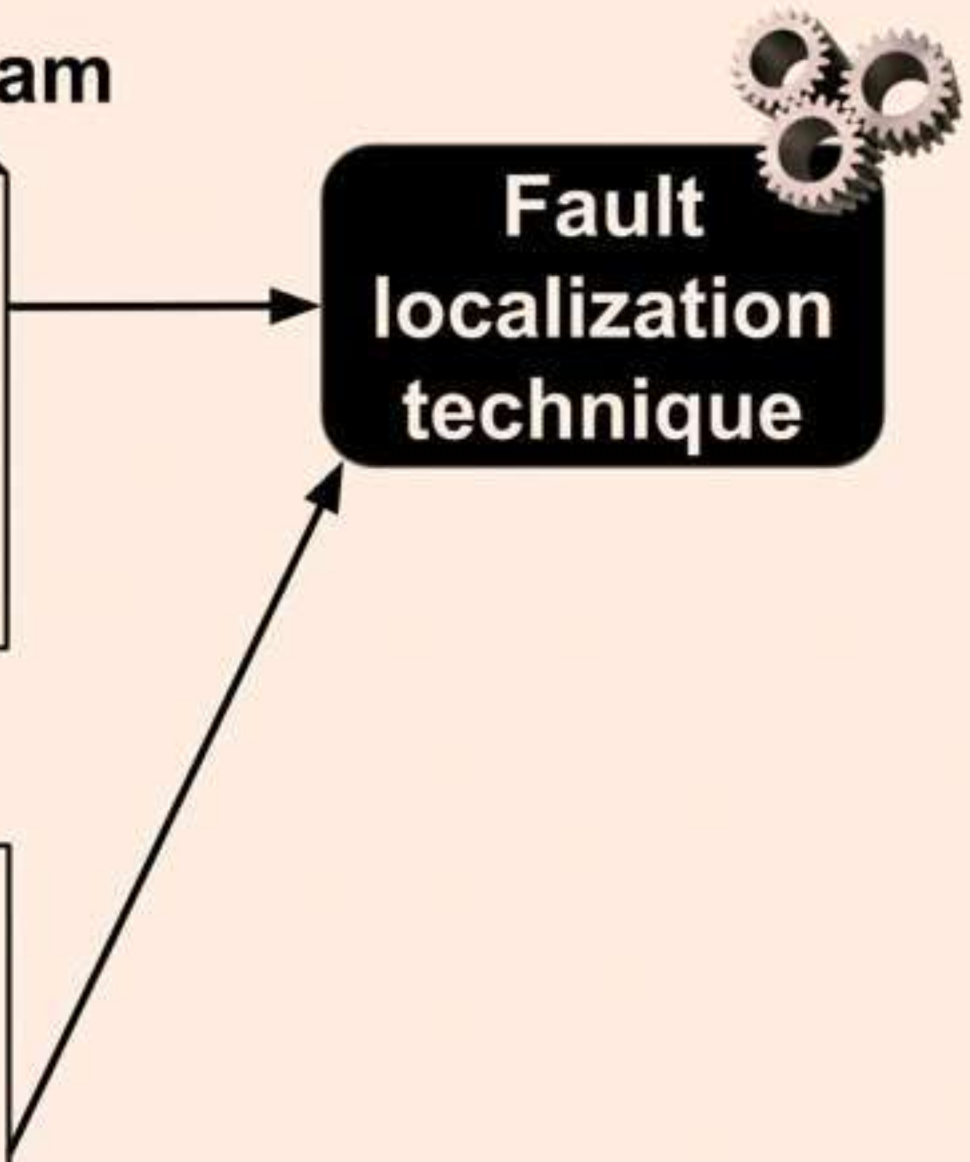
René Just and Michael Ernst
PNW PLSE meeting
May 14, 2018

Joint work with Spencer Pearson, José Campos, Gordon Fraser, Rui Abreu, Deric Pang, Benjamin Keller, Chris Parnin, Ian Drosos

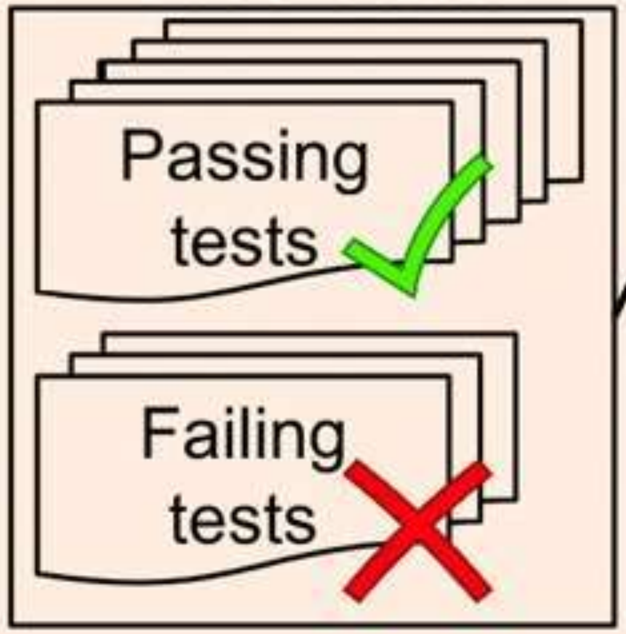
Fault localization: where is the defect?

Defective program

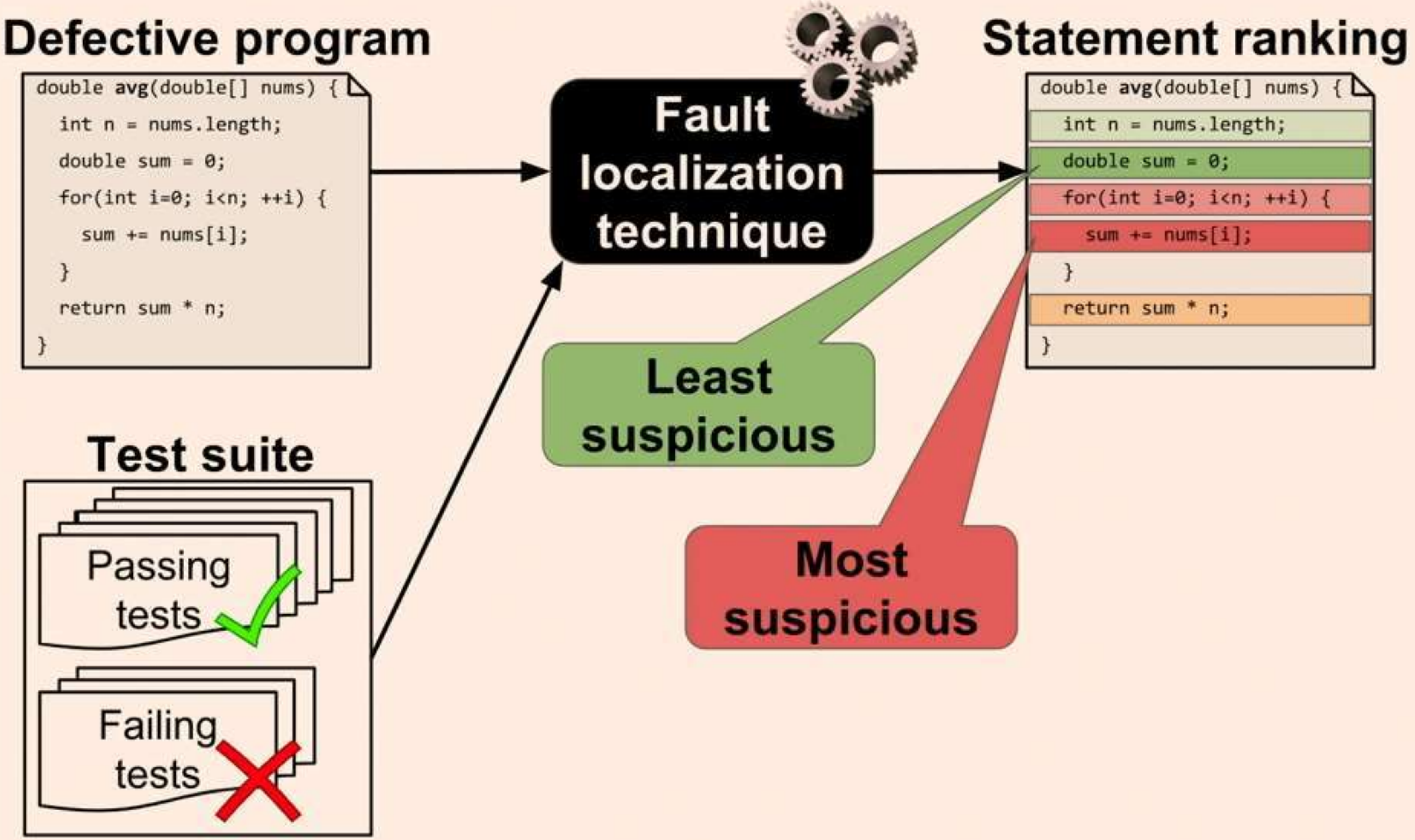
```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;
  for(int i=0; i<n; ++i) {
    sum += nums[i];
  }
  return sum * n;
}
```



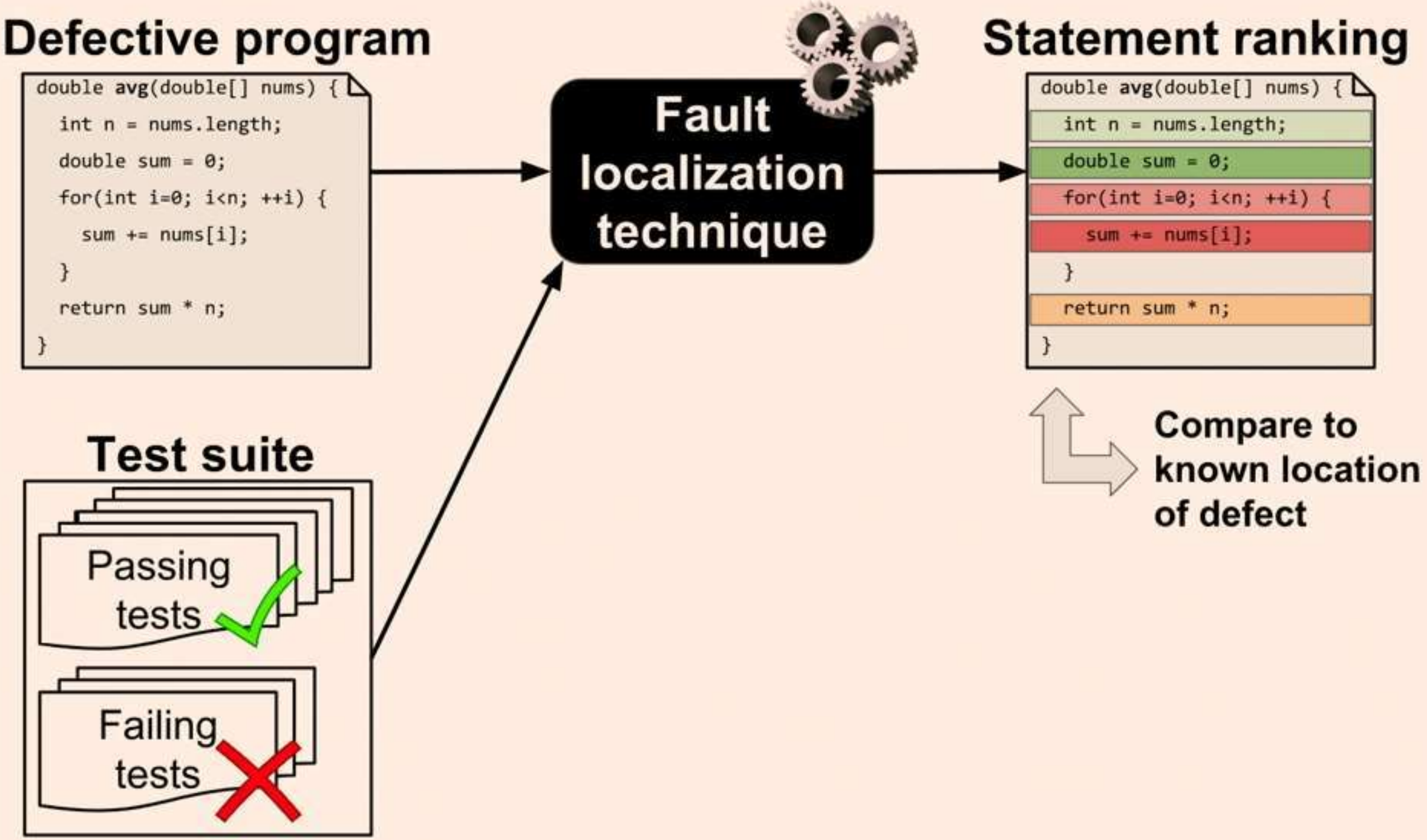
Test suite



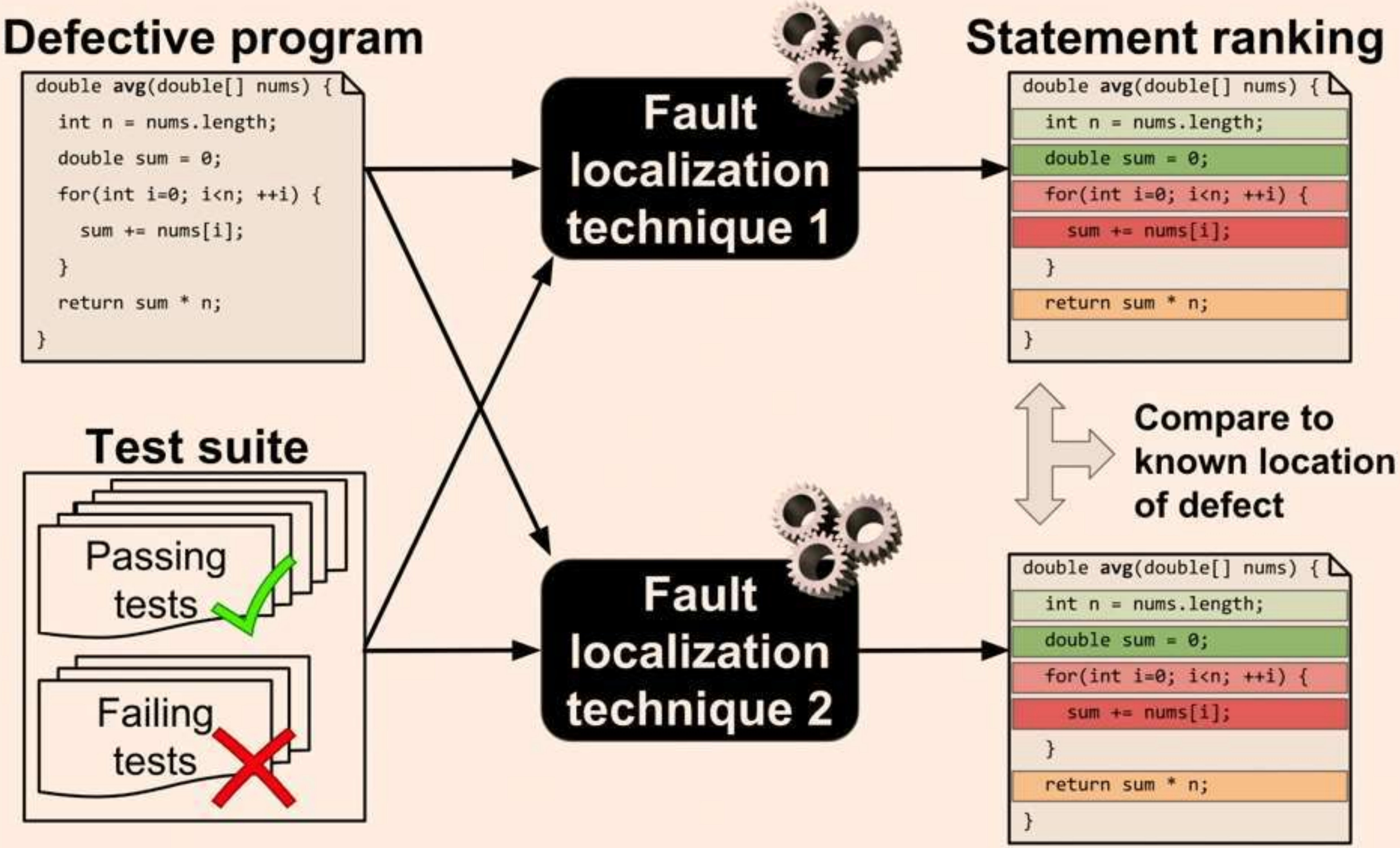
Fault localization: where is the defect?



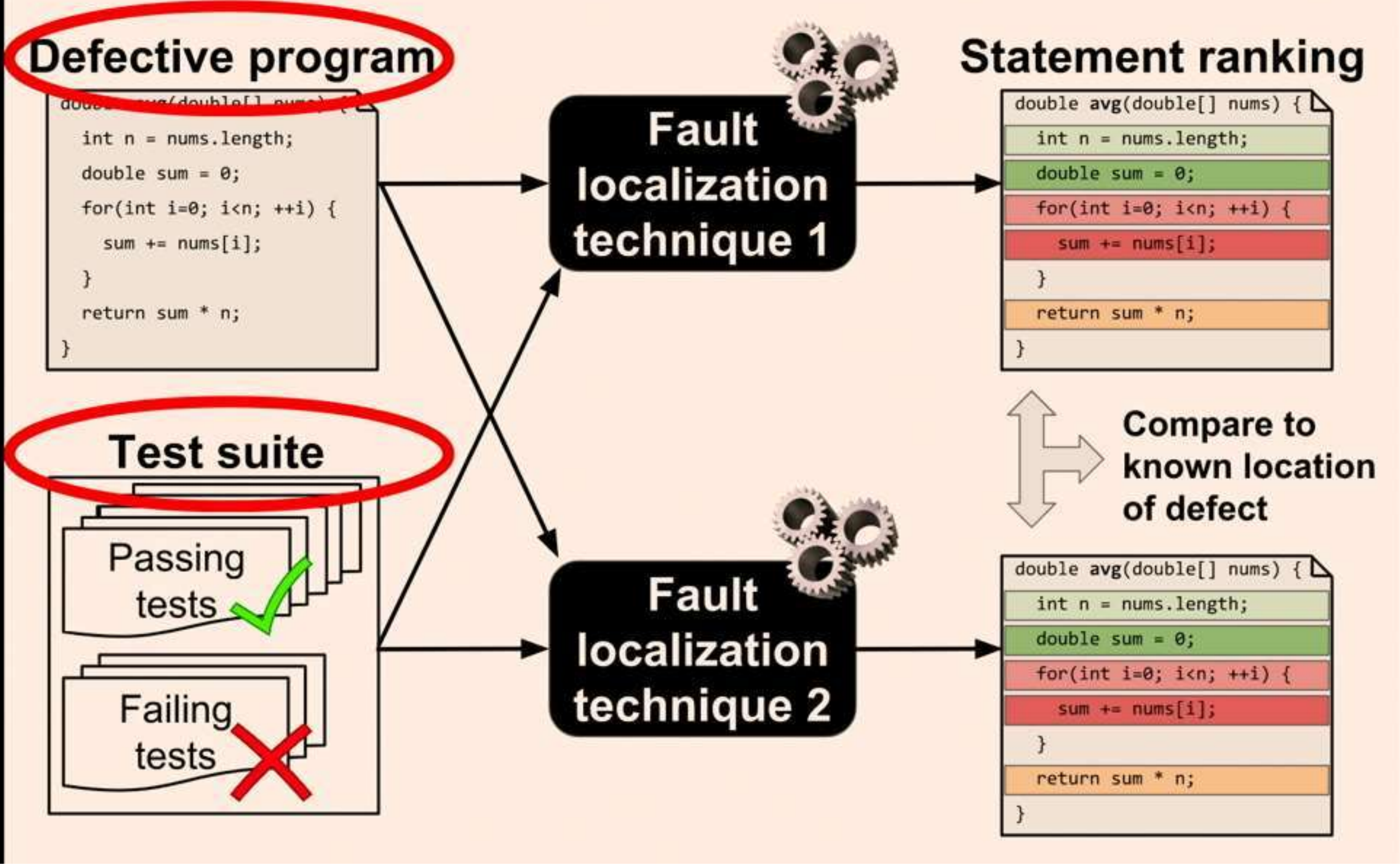
Evaluating fault localization

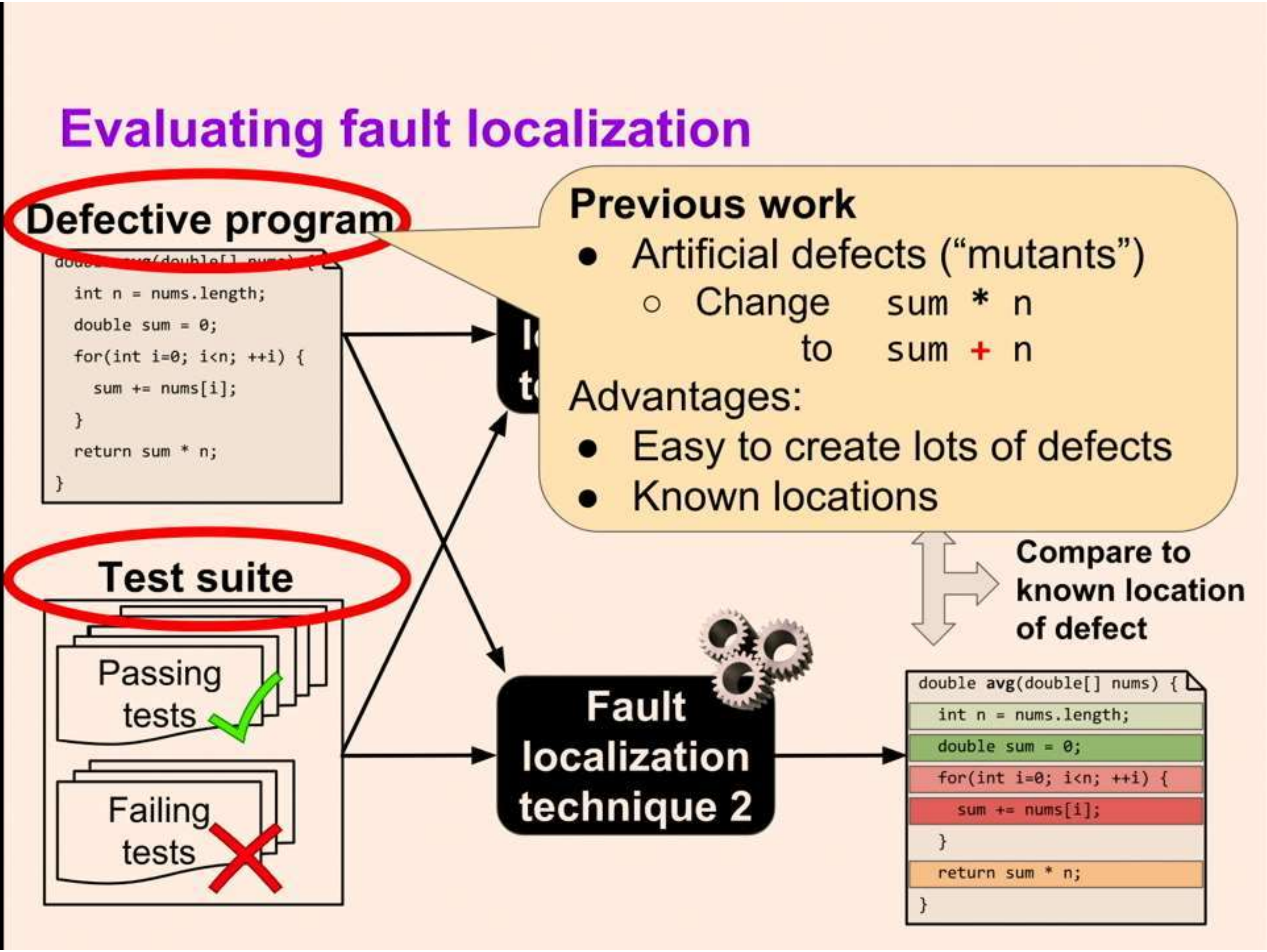


Evaluating fault localization



Evaluating fault localization




Evaluating fault localization
Defective program
double avg(double[] nums) {
int n = nums.length;
double sum = 0;
for(int i=0; i<n; ++i) {
sum += nums[i];
}
return sum * n;
}
Test suite
Passing tests
Failing tests
Previous work
Artificial defects ("mutants")
Change sum * n
to sum + n
Advantages:
Easy to create lots of defects
Known locations
Compare to known location of defect
Fault localization technique 2
double avg(double[] nums) {
int n = nums.length;
double sum = 0;
for(int i=0; i<n; ++i) {
sum += nums[i];
}
return sum * n;
}

Evaluating fault localization

Defective program

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;
  for(int i=0; i<n; ++i) {
    sum += nums[i];
  }
  return sum * n;
}
```

Test suite

Passing tests

Failing tests

Previous work

- Artificial defects ("mutants")
  - Change `sum * n` to `sum + n`

Advantages:

- Easy to create lots of defects
- Known locations

Our work

- 2995 artificial defects
  - > Σ previous studies
- 310 real defects
  - Each fixed by developers
  - 5× Σ previous studies
  - Several person-years
  - https://github.com/rjust/defects4j

[ICSE 2017]

Comparison of fault localization techniques

	Prior studies (winner > loser)
SBFL vs. SBFL	Ochiai > Tarantula
	Barinel > Ochiai
	Barinel > Tarantula
	Op2 > Ochiai
	Op2 > Tarantula
	DStar > Ochiai
	DStar > Tarantula
MBFL vs. SBFL	Metallaxis > Ochiai
	MUSE > Op2
	MUSE > Tarantula

Comparison of fault localization techniques

	Prior studies (winner > loser)	Ours (artificial faults) Replicated	Effect
SBFL vs. SBFL	Ochiai > Tarantula	**yes**	small
	Barinel > Ochiai	**no**	small
	Barinel > Tarantula	**yes**	*negligible*
	Op2 > Ochiai	**yes**	*negligible*
	Op2 > Tarantula	**yes**	small
	DStar > Ochiai	**yes**	*negligible*
	DStar > Tarantula	**yes**	small
MBFL vs. SBFL	Metallaxis > Ochiai	yes	*negligible*
	MUSE > Op2	**no**	*negligible*
	MUSE > Tarantula	**no**	*negligible*

Results agree with most prior studies **on artificial faults** but **only 3 effect sizes** are **not negligible.**

[ICSE 2017]

Comparison of fault localization techniques

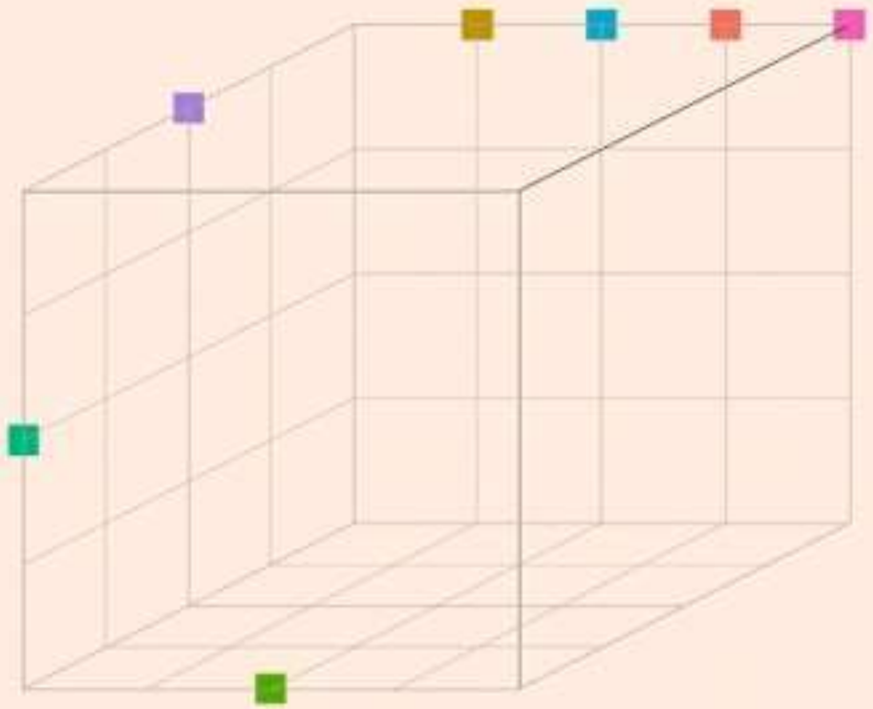
	Prior studies (winner > loser)	Ours (artificial faults) Replicated	Ours (artificial faults) Effect	Ours (real faults) Replicated	Ours (real faults) Effect
SBFL vs. SBFL	Ochiai > Tarantula	**yes**	small	*insignificant*	*negligible*
	Barinel > Ochiai	**no**	small	*insignificant*	*negligible*
	Barinel > Tarantula	**yes**	*negligible*	*insignificant*	*negligible*
	Op2 > Ochiai	**yes**	*negligible*	no	*negligible*
	Op2 > Tarantula	**yes**	small	*insignificant*	*negligible*
	DStar > Ochiai	**yes**	*negligible*	*insignificant*	*negligible*
	DStar > Tarantula	**yes**	small	*insignificant*	*negligible*
MBFL vs. SBFL	Metallaxis > Ochiai	yes	*negligible*	**no**	small
	MUSE > Op2	**no**	*negligible*	**no**	**large**
	MUSE > Tarantula	**no**	*negligible*	**no**	**large**

Results disagree with all prior studies **on real faults**.

What design decisions matter on real faults?

Defined and explored a design space for FL techniques

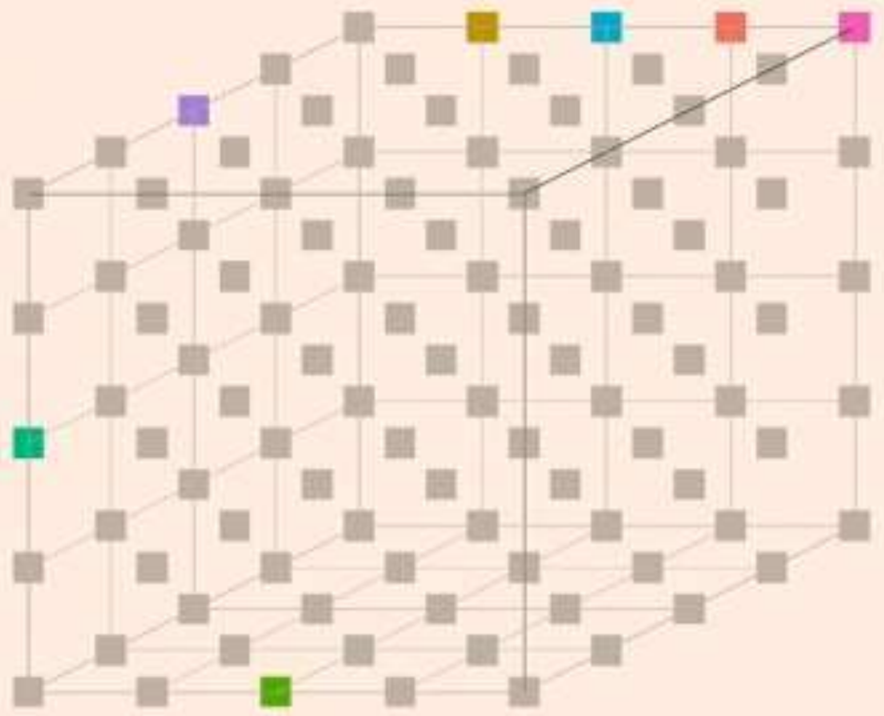
- 4 design factors
  (e.g., ranking formula)



What design decisions matter on real faults?

Defined and explored a design space for FL techniques

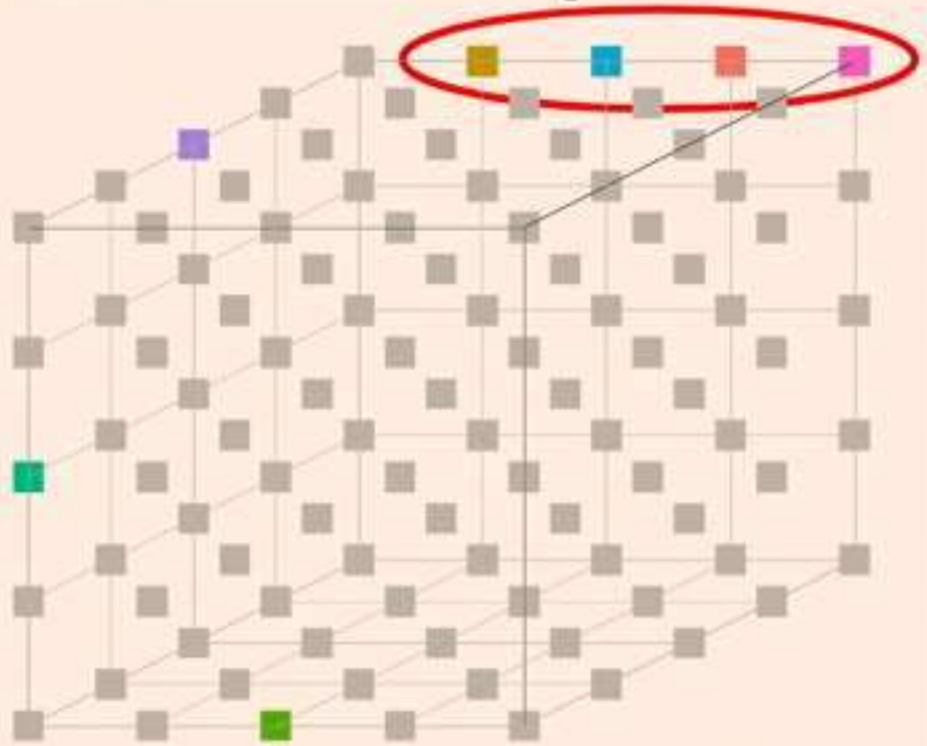
- 4 design factors
  (e.g., ranking formula)
- 156 FL techniques



What design decisions matter on real faults?

Defined and explored a design space for FL techniques

- 4 design factors (e.g., ranking formula)
- 156 FL techniques



Results

- Most design decisions **don't matter**
- Barinel, D*, Ochiai, and Tarantula are indistinguishable

Existing FL techniques perform **best.**
No breakthroughs in the explored **design space.**

New hybrid technique

In practice, only the top results matter

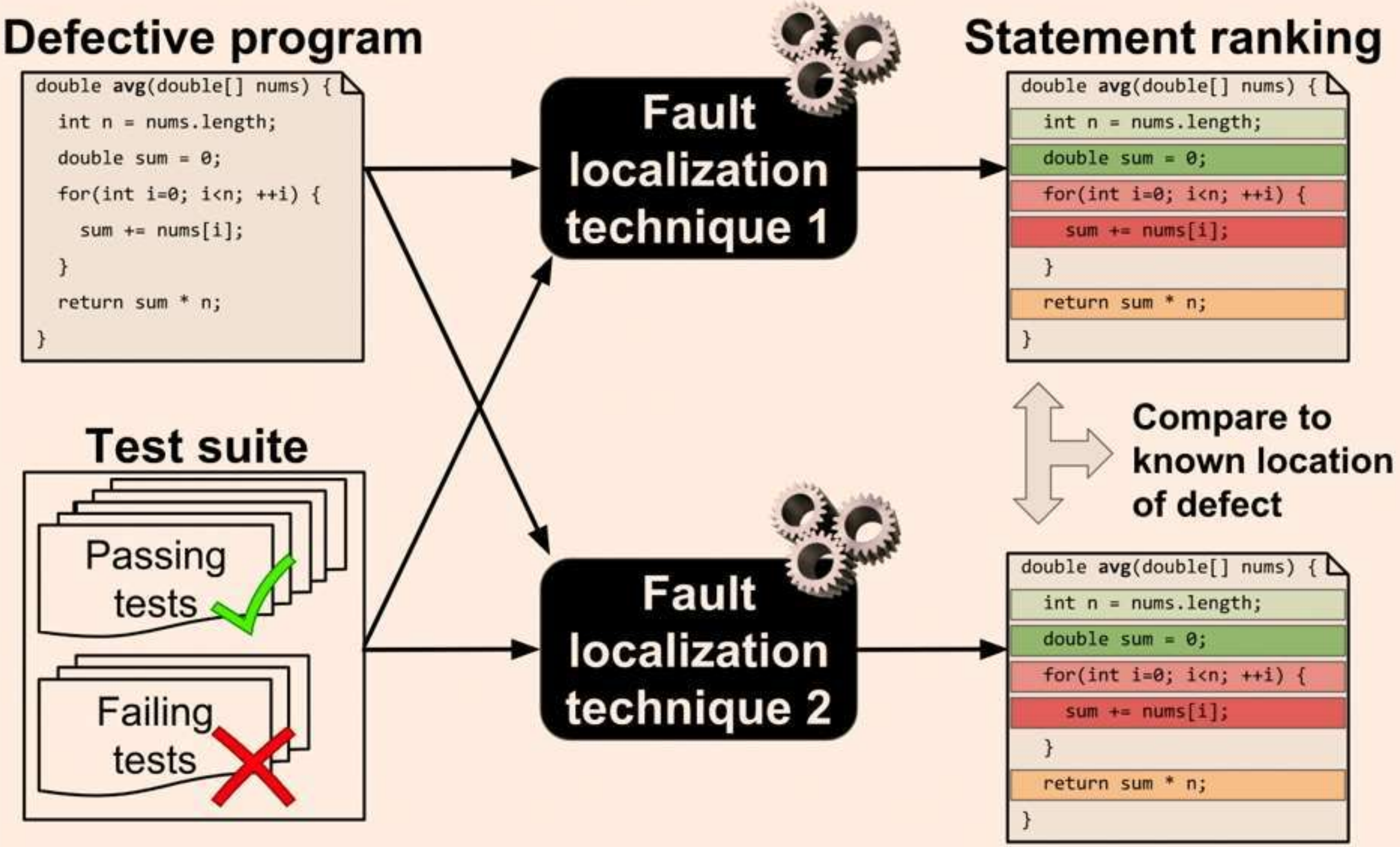
- Top-10 useful for practitioners[1].
- Top-200 useful for automated program repair[2].

Technique	Top-5	Top-10	Top-200
Hybrid	36%	45%	85%
DStar (*best SBFL*)	30%	39%	82%
Metallaxis (*best MBFL*)	29%	39%	77%

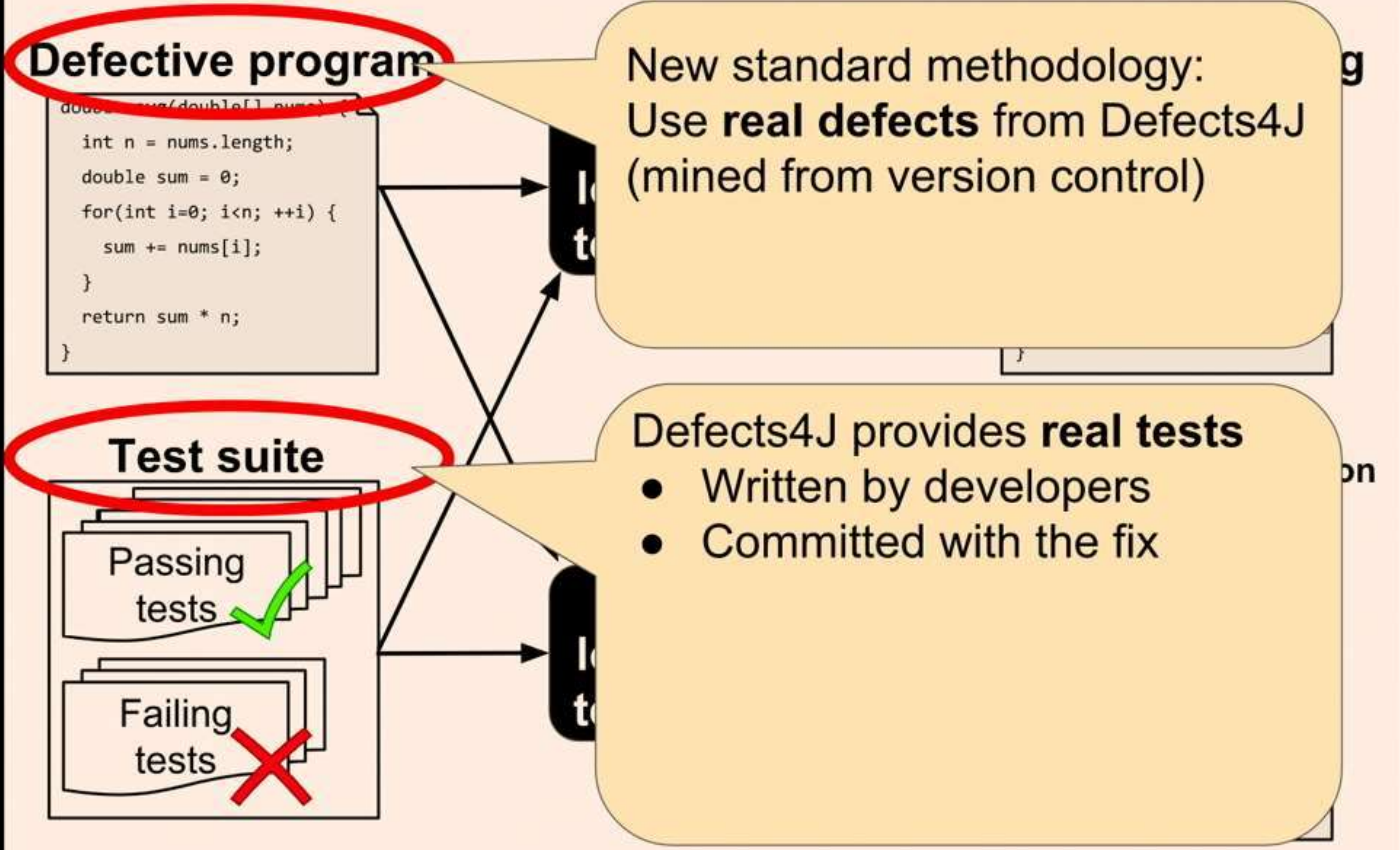
Hybrid technique **performs well** on the **metric that matters.**

[1]Kochhar et al., *Practitioners' Expectations on Automated Fault Localization*, ISSTA'16
[2]Long and Rinard, *An analysis of the search spaces for generate and validate patch generation systems*, ICSE'16

Evaluating fault localization



Evaluating fault localization

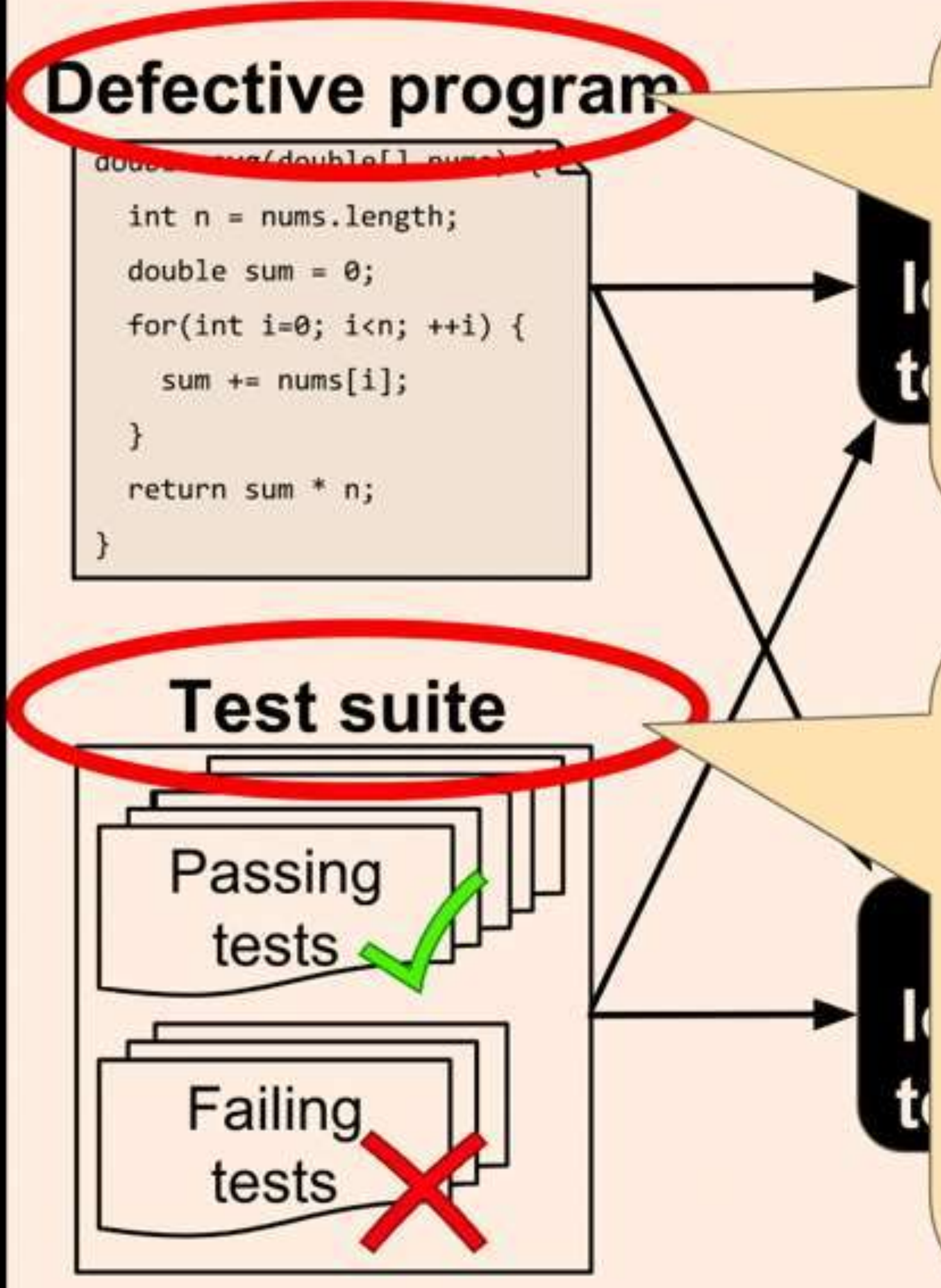


New standard methodology:
Use **real defects** from Defects4J
(mined from version control)

Defects4J provides **real tests**

- Written by developers
- Committed with the fix

Evaluating fault localization



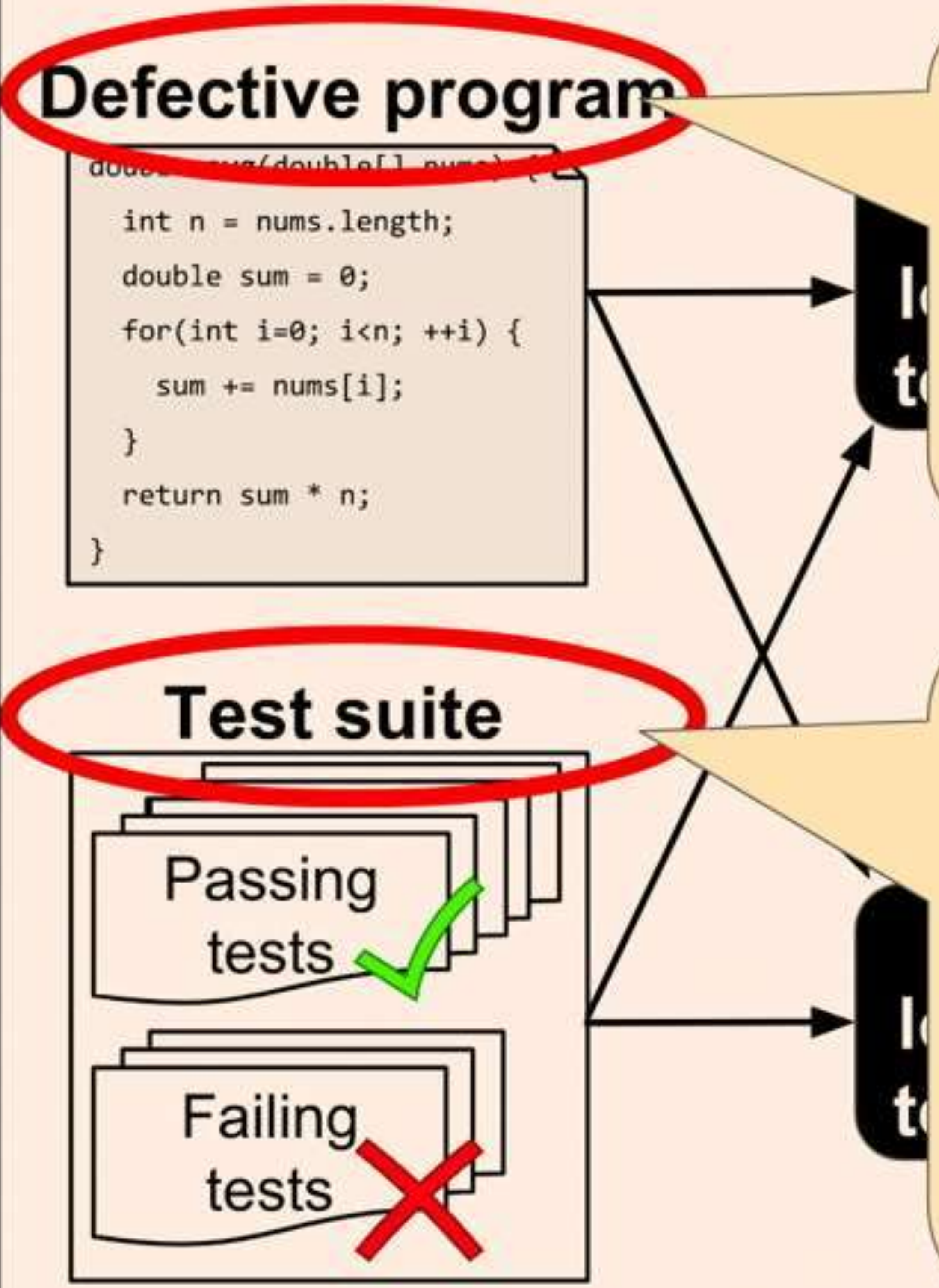
New standard methodology:
Use **real defects** from Defects4J
(mined from version control)

Defects4J provides **real tests**

- Written by developers
- Committed with the fix

Written before or after the fix?

Evaluating fault localization



New standard methodology:
Use **real defects** from Defects4J
(mined from version control)

Defects4J provides **real tests**

- Written by developers
- Committed with the fix

Written before or after the fix?

In practice, fault localization is run on tests from **bug reports**.

[ISSTA 2018]

User-provided tests in the bug report vs. developer-provided tests committed with the fix

Developer-provided tests have:

- More tests
- More lines of test code
- Less coverage (more focused)
- More assertions
- Stronger assertions

normal

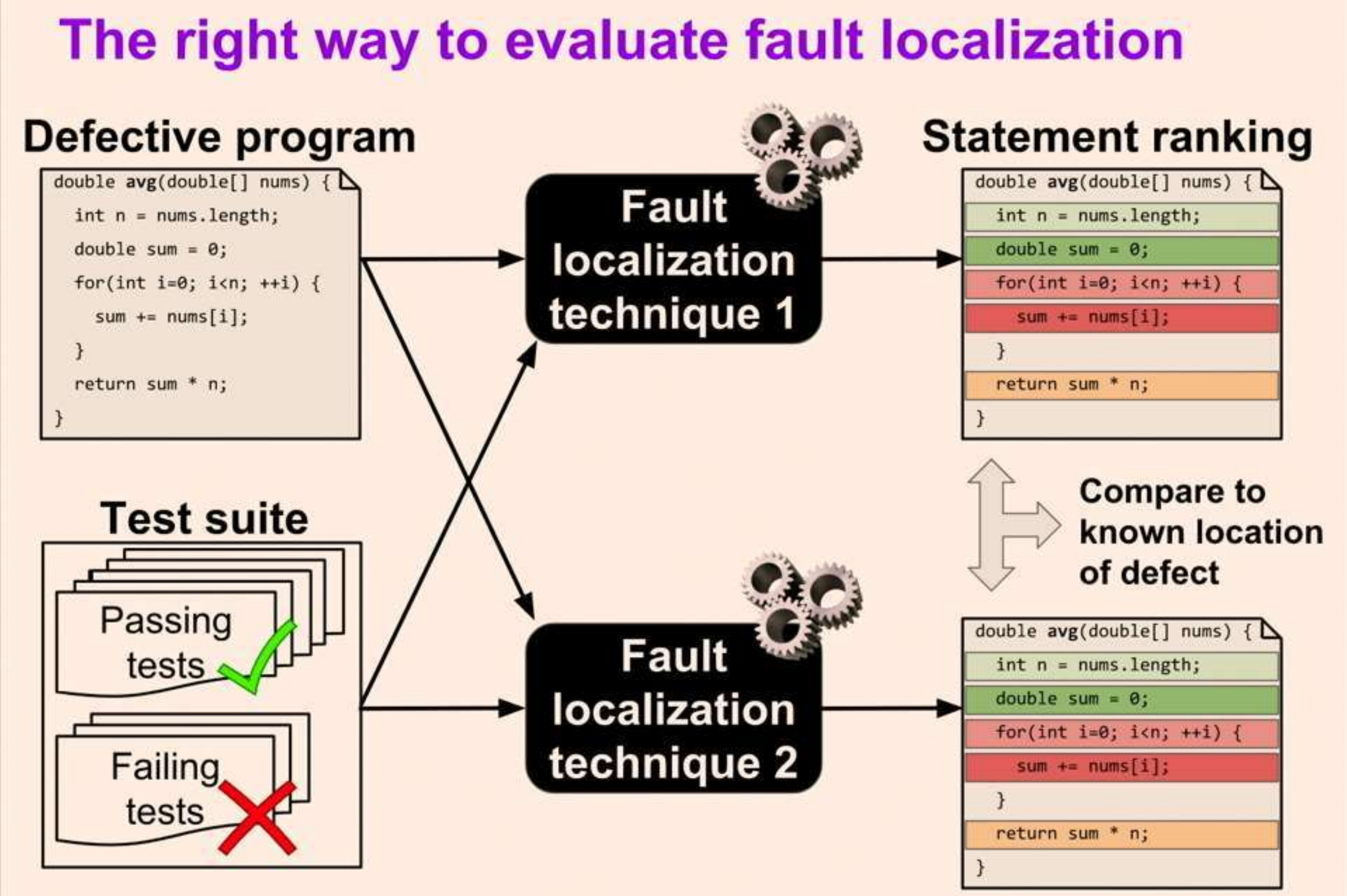
Effect on tools
(applied to user-provided vs. developer-provided tests)

Fault localization:

- Better EXAM score with developer-provided tests
- Better top-N score by 5-14%

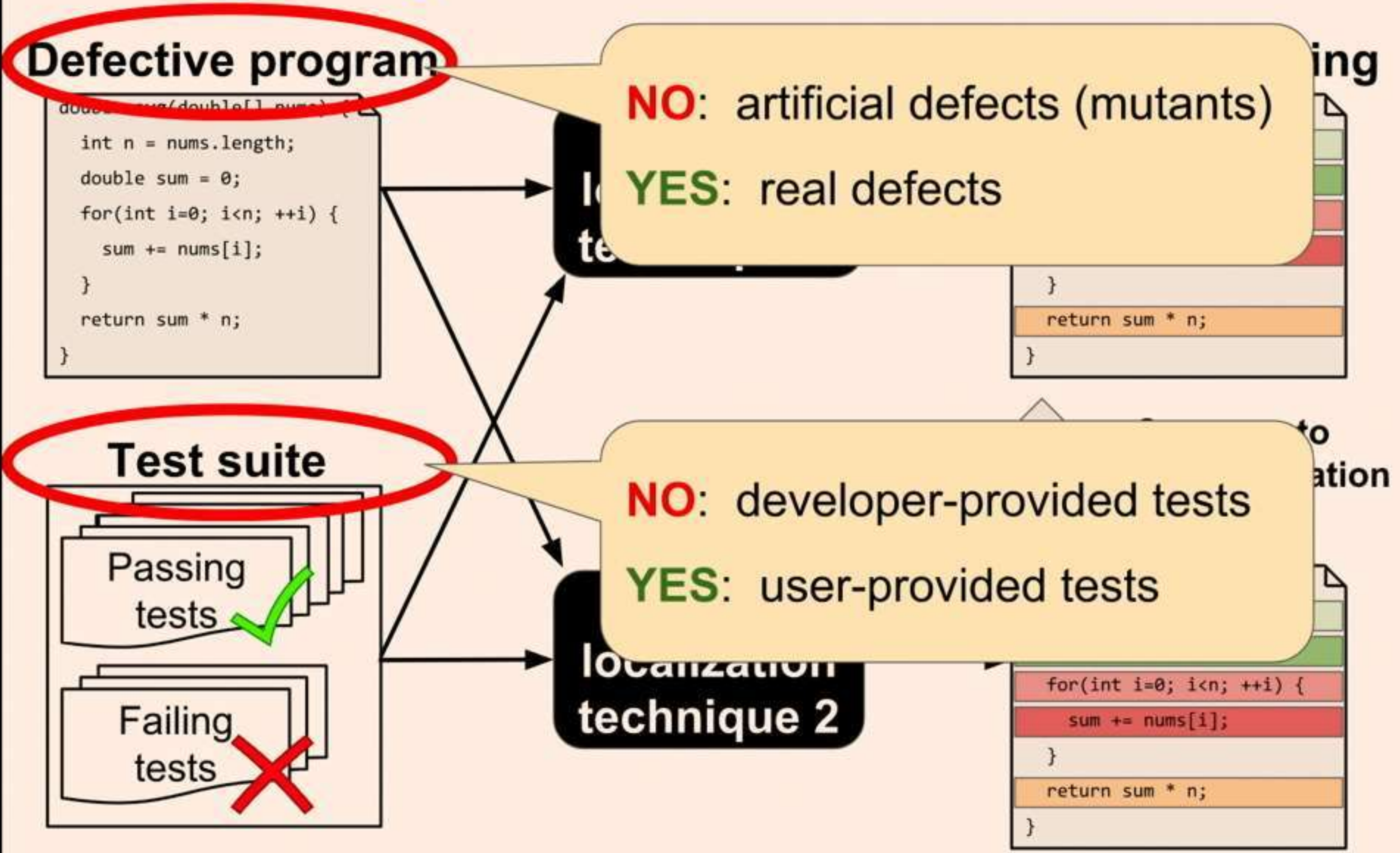
Automated program repair:

- Developer-provided tests: repair 5/100 defects
- User-provided tests: repair 1/100 defects
  (For that defect, user-submitted test = developer-provided!)
  - Fewer generated patches (irrelevant measure)
  - Fewer correct patches
  - Longer run time
  - Partly due to worse fault localization



The right way to evaluate fault localization
Defective program
double avg(double[] nums) {
int n = nums.length;
double sum = 0;
for(int i=0; i<n; ++i) {
sum += nums[i];
}
return sum * n;
}
Test suite
Passing tests
Failing tests
Fault localization technique 1
Fault localization technique 2
Statement ranking
double avg(double[] nums) {
int n = nums.length;
double sum = 0;
for(int i=0; i<n; ++i) {
sum += nums[i];
}
return sum * n;
}
Compare to known location of defect
double avg(double[] nums) {
int n = nums.length;
double sum = 0;
for(int i=0; i<n; ++i) {
sum += nums[i];
}
return sum * n;
}

The right way to evaluate fault localization



The right way to evaluate any research

Focus on results, not ideas

Evaluate using realistic artifacts

Evaluate in end-user context

The right way to evaluate any research

Focus on results, not ideas

Evaluate using realistic artifacts

Evaluate in end-user context

Is fault localization research especially bad?

It's no worse than other research, and better than much

It has found its conscience; other areas are still seeking

Cassius: Verifying Web Pages

Pavel Panchekha, Adam Geller, Michael D. Ernst, Shoaib Kamil, Zachary Tatlock

⟦ ⟧ λ

∀ ⊢

PL/SE

+

PL/SE

Websites

Server-side

Server-side

Client-side

Server-side

Client-side

Layout

Server-side

Client-side

This Talk

Layout


healthcare.gov
HealthCare.gov
Individuals & Families
Small Businesses
ESPAÑOL
LOG IN
Get Coverage
Keep or Update Your Plan
See Topics
Get Answers
Search
Preview 2018 plans & prices now!
Check out plans now. Enroll or renew from November 1 to December 15
PREVIEW 2018 PLANS & PRICES

Which bugs and tests sh | Which bugs and tests sh | PNW PLSE Cassius - Goo | PNW PLSE Cassius - Goo | Get 2018 health coverag

https://www.healthcare.gov

HealthCare.gov

Español   Log in

Get Coverage   Keep or Update Your Plan   See Topics   Get Answers   Search   SEARCH

2018 Open Enrollment is over. Still need health insurance?

You can enroll in or change plans if you have certain life changes, or qualify for Medicaid or CHIP

SEE IF I CAN ENROLL   SEE IF I CAN CHANGE

Looking for coverage for a small business? Learn more

NEED TO SUBMIT DOCUMENTS?

SEE HOW

FIND LOCAL HELP	GET CONTACTED	1095 & TAX INFO	INCOME/LIFE CHANGE?
SEARCH NOW	HELP FROM AGENT/BROKER	SEE NOW	SEE HOW TO REPORT

GET IMPORTANT NEWS & UPDATES

Sign up for email and text updates to get deadline reminders and other important information.

HEALTHCARE.GOV BLOG

May 10

Attention college grads: Know your health insurance options

normal

Web Pages as Programs

Web Pages as Programs

Specifications

No text overlap

Buttons on screen

High contrast

Heading hierarchy

No horizontal scroll

Web Pages as Programs

Specifications

No text overlap

Buttons on screen

High contrast

Heading hierarchy

No horizontal scroll

ADA Best Practices

PL/SE for web pages?

$[\![\ ?\ ]\!] =$ 

$\forall\ ?,\ P($  $)$

$?\ \vdash P($  $)$

1. Semantics of web pages
2. Logic for visual properties
3. Compositional reasoning

1. **Semantics of web pages**
2. Logic for visual properties
3. Compositional reasoning

Semantics of web pages

English-language

Informal

Ambiguous

Semantics of web pages

W3C® / (browsers)

English-language	Executable
Informal	1M+ lines of C++
Ambiguous	23 years of cruft

Semantics of web pages

\+

English-language	Executable
Informal	1M+ lines of C++
Ambiguous	23 years of cruft

Semantics of web pages

Conformance tests

Executable

1M+ lines of C++

23 years of cruft

Semantics of web pages

Conformance tests + Desired behavior

Semantics of web pages

+

Conformance tests

Desired behavior

Describes behavior of complex web pages

35 pages of text → 1000 lines of formalization

1. **Semantics of web pages**
2. Logic for visual properties
3. Compositional reasoning

1. Semantics of web pages

2. **Logic for visual properties**

3. Compositional reasoning

Logic of Visual Properties

$$\forall\, b : \text{Box}, \qquad b \in \$(\text{button}) \quad \Rightarrow \quad b \subset \text{root}$$

normal

Logic of Visual Properties

$$\forall\, b : \text{Box}, \quad b \in \$(\text{button}) \quad \Rightarrow \quad b \subset \text{root}$$

Quantify over boxes

Logic of Visual Properties

$$\forall\, b : \mathrm{Box}, \quad b \in \$(\mathrm{button}) \quad \Rightarrow \quad b \subset \mathrm{root}$$

Quantify over boxes

HTML Properties

Logic of Visual Properties

$$\forall\, b : \text{Box}, \qquad b \in \$(\text{button}) \quad \Rightarrow \quad b \subset \text{root}$$

Quantify over boxes

HTML Properties

Geometric Predicates

Logic of Visual Properties

$$\forall\, b : \text{Box}, \quad b \in \$(\text{button}) \quad \Rightarrow \quad b \subset \text{root}$$

Quantify over boxes

HTML Properties

Geometric Predicates

Expressed 14 accessibility guidelines

Compiles to decidable queries (in QFLRA)

1. Semantics of web pages
2. **Logic for visual properties**
3. Compositional reasoning

1. Semantics of web pages
2. Logic for visual properties
3. **Compositional reasoning**

Compositional Reasoning

Compositional Reasoning

Compositional Reasoning

Compositional Reasoning

Compositional Proofs

$$\forall\, b \quad ,\, b \in \$(\text{button}) \Rightarrow b \subset \text{root}$$

Compositional Proofs

$$\forall\, c : \text{Component},\quad c \subset \text{root} \Rightarrow$$

$$\forall\, b \in \text{c}, b \in \text{\$(button)} \Rightarrow b \subset \text{root}$$

Compositional Proofs

$$\forall\, c : \text{Component}, \quad c \subset \text{root} \Rightarrow$$

$$\forall\, b \in \text{c}, b \in \$(\text{button}) \Rightarrow b \subset \text{root}$$

Per-component reasoning

normal

Compositional Proofs

Component precondition

$$\forall\, c : \text{Component},\quad c \subset \text{root} \Rightarrow$$

$$\forall\, b \in \text{c}, b \in \$(\text{button}) \Rightarrow b \subset \text{root}$$

Per-component reasoning

Compositional Proofs

Component precondition

$$\forall\ c : \text{Component},\quad c \subset \text{root} \Rightarrow$$

$$\forall\ b \in \text{c}, b \in \$(\text{button}) \Rightarrow b \subset \text{root}$$

Per-component reasoning

Reuse across versions, pages, websites

Much faster: small problem size, parallelism

1. Semantics of web pages
2. Logic for visual properties
3. **Compositional reasoning**

1. Semantics of web pages
2. Logic for visual properties
3. Compositional reasoning

Cassius

Semantics of web pages

Logic for visual properties

Compositional reasoning

https://cassius.uwplse.org

1. Semantics of web pages
2. Logic for visual properties
3. Compositional reasoning

1. Semantics of web pages
2. Logic for visual properties
3. Compositional reasoning
4. **Client-server reasoning**

This Talk

php
Java
RAILS
JS
jQuery
React
HTML
CSS
Sass
B

Server-side | **Client-side** | **Layout**

normal

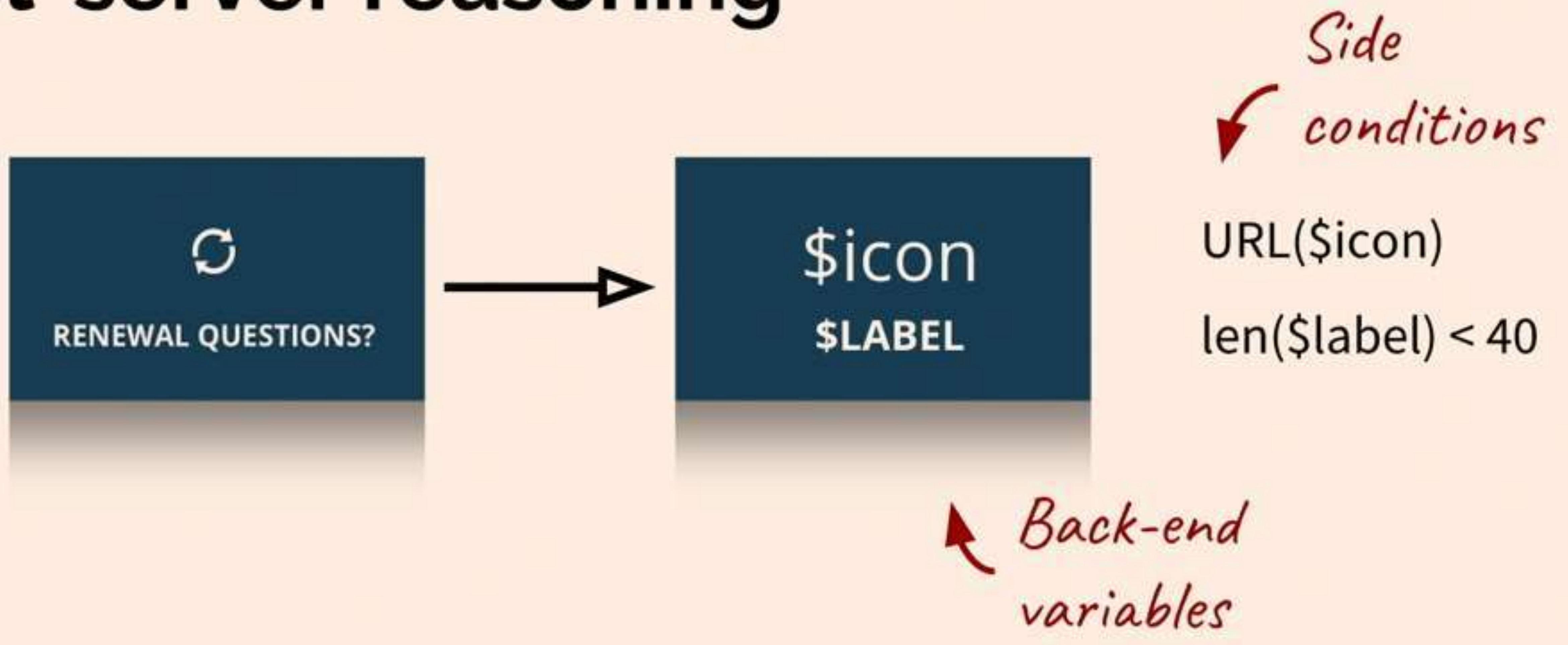
The Future

Server-side

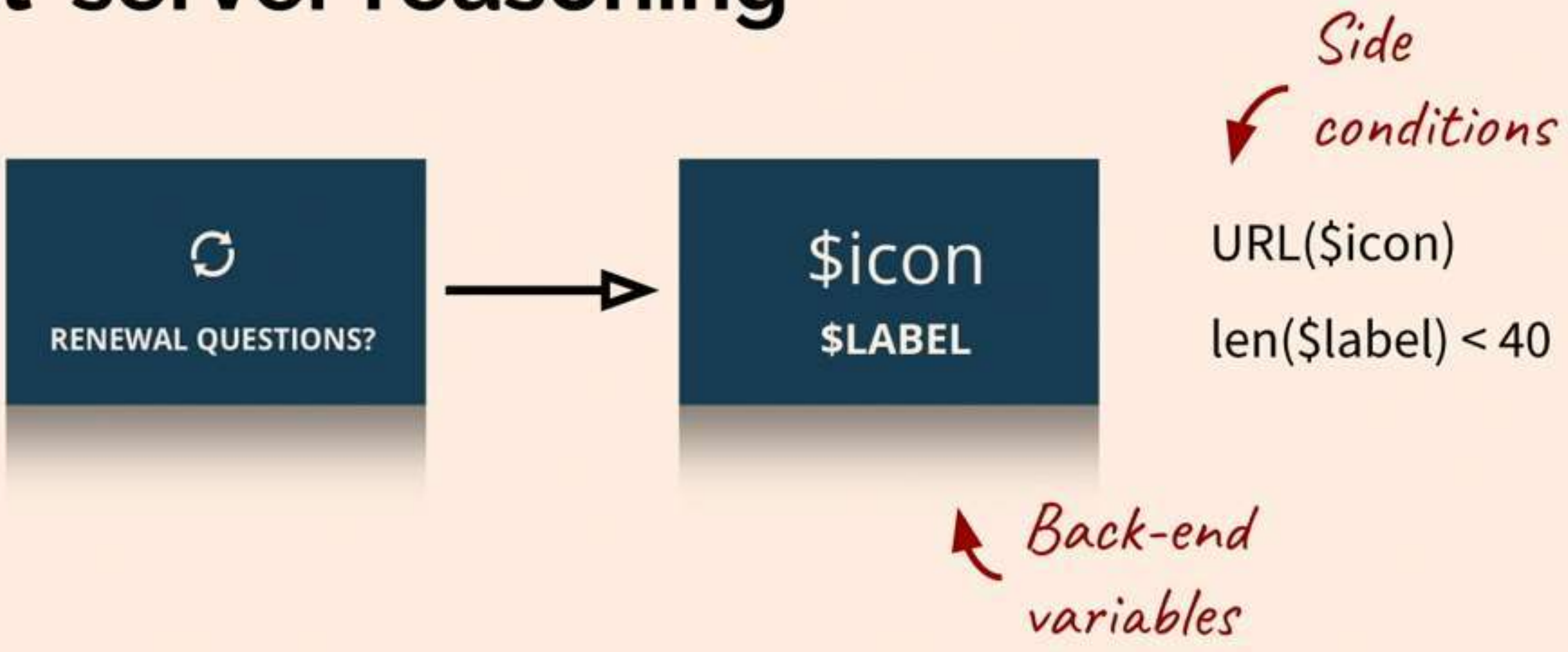
Client-side

Layout

Client-server reasoning



Client-server reasoning



Abstract page content into template

Prove properties of all pages a back-end can produce

Cassius

Semantics of web pages

Logic for visual properties

Compositional reasoning

https://cassius.uwplse.org