

Rely: Verifying Quantitative Reliability for Programs that Execute on Unreliable Hardware

Michael Carbin, Sasa Misailovic,
and Martin Rinard

MIT CSAIL

Image Scaling

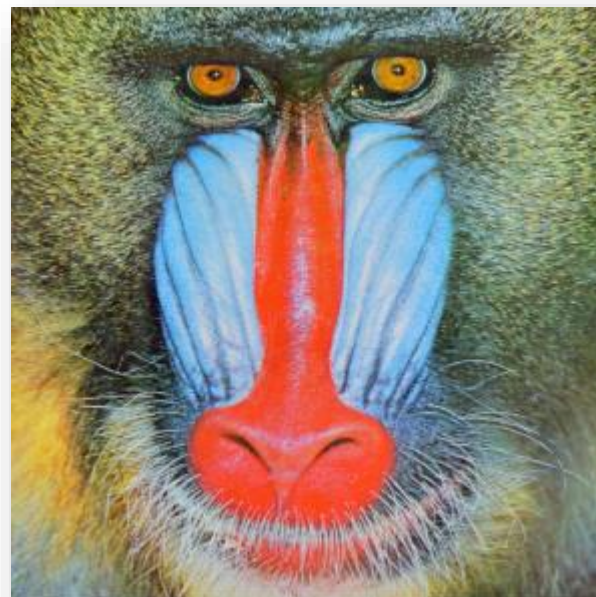
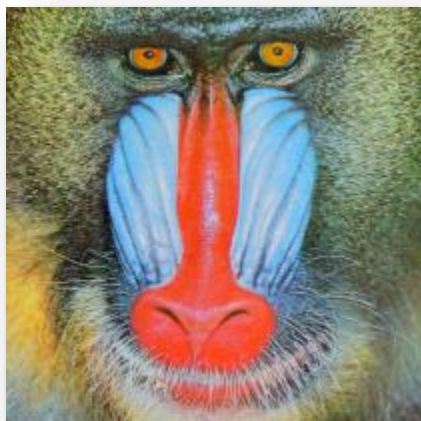
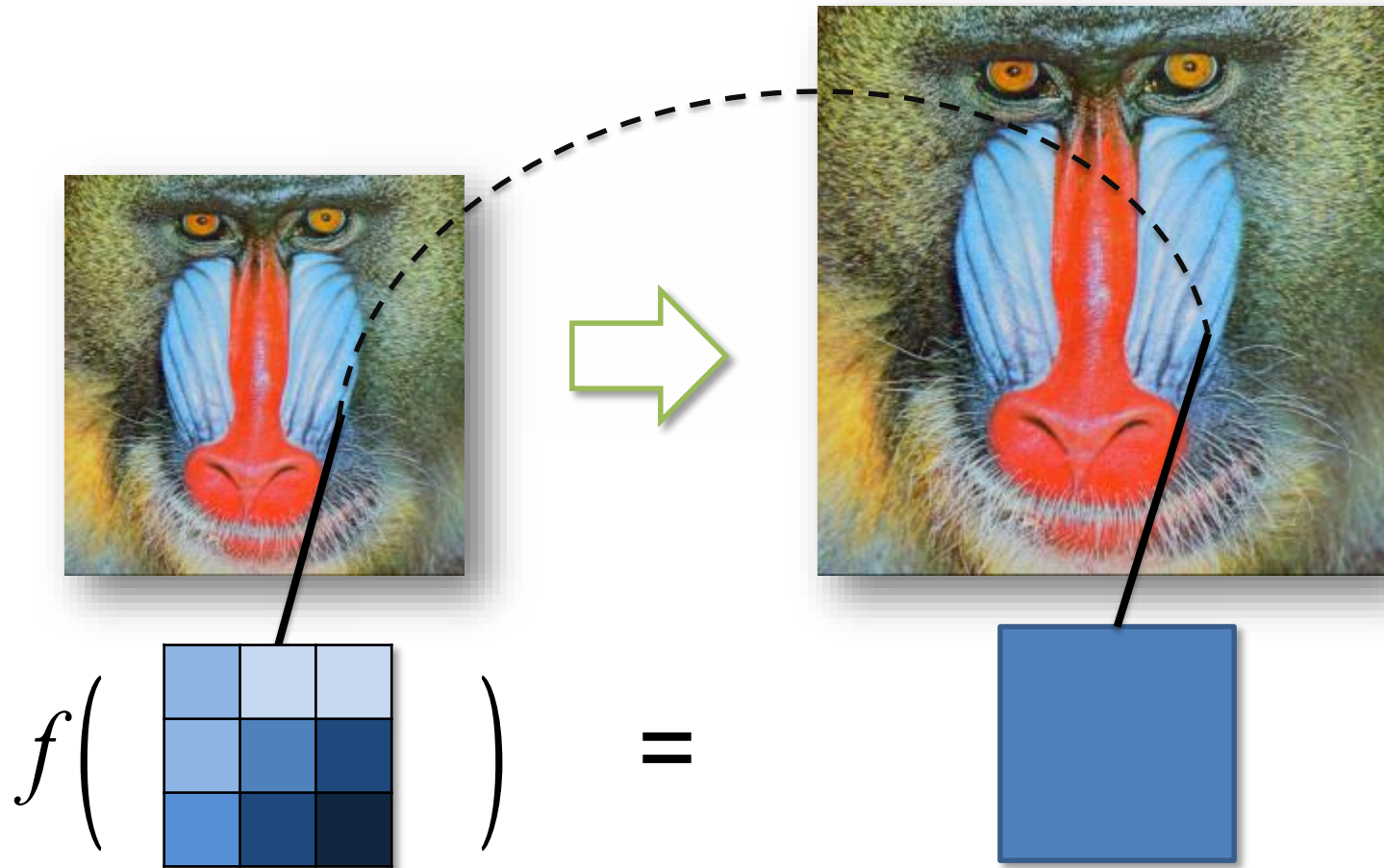


Image Scaling Kernel: Bilinear Interpolation



Bilinear Interpolation

```
int bilinear_interpolation(int i, int j,
                          int src[][[]], int dest[][[]])
{
    int i_src = map_y(i, src, dest),
        j_src = map_x(j, src, dest);

    int up    = i_src - 1, down    = i_src + 1,
        left  = j_src - 1, right  = j_src + 1;

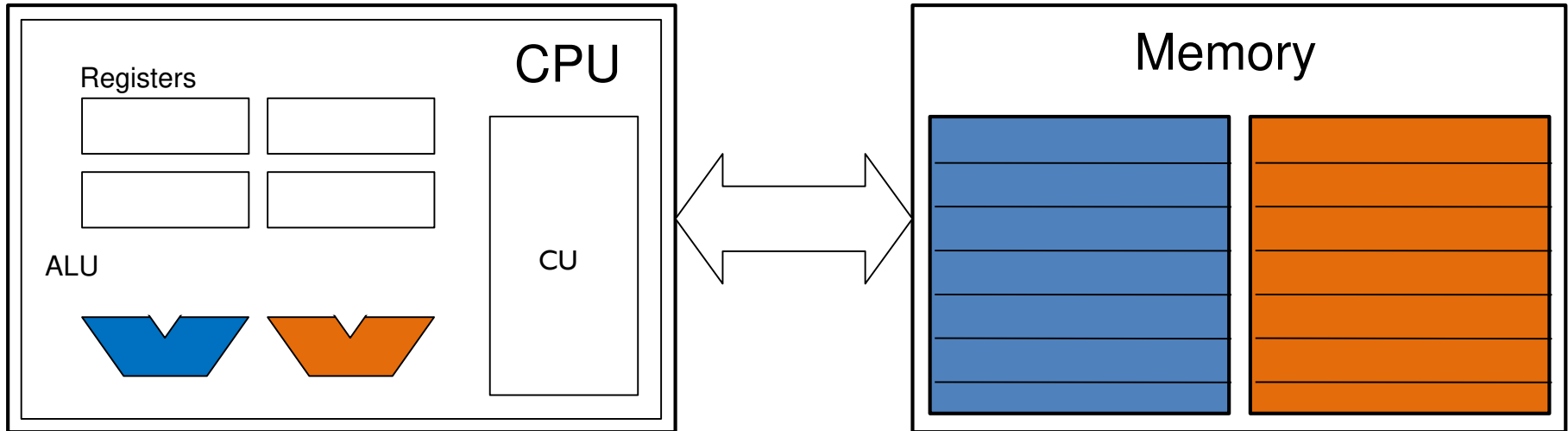
    int val = src[up][left]    + src[up][right] +
              src[down][right] + src[down][left];

    return 0.25 * val;
}
```

Bilinear Interpolation

```
int bilinear_interpolation(int i, int j,  
                           int src[][[]], int dest[][[]])  
{  
    int i_src = map_y(i, src, dest),  
        j_src = map_x(j, src, dest);  
  
    int up    = i_src - 1, down    = i_src + 1,  
        left  = j_src - 1, right  = j_src + 1;  
  
    int val = src[up][left]    + src[up][right] +  
              src[down][right] + src[down][left];  
  
    return 0.25 * val;  
}
```

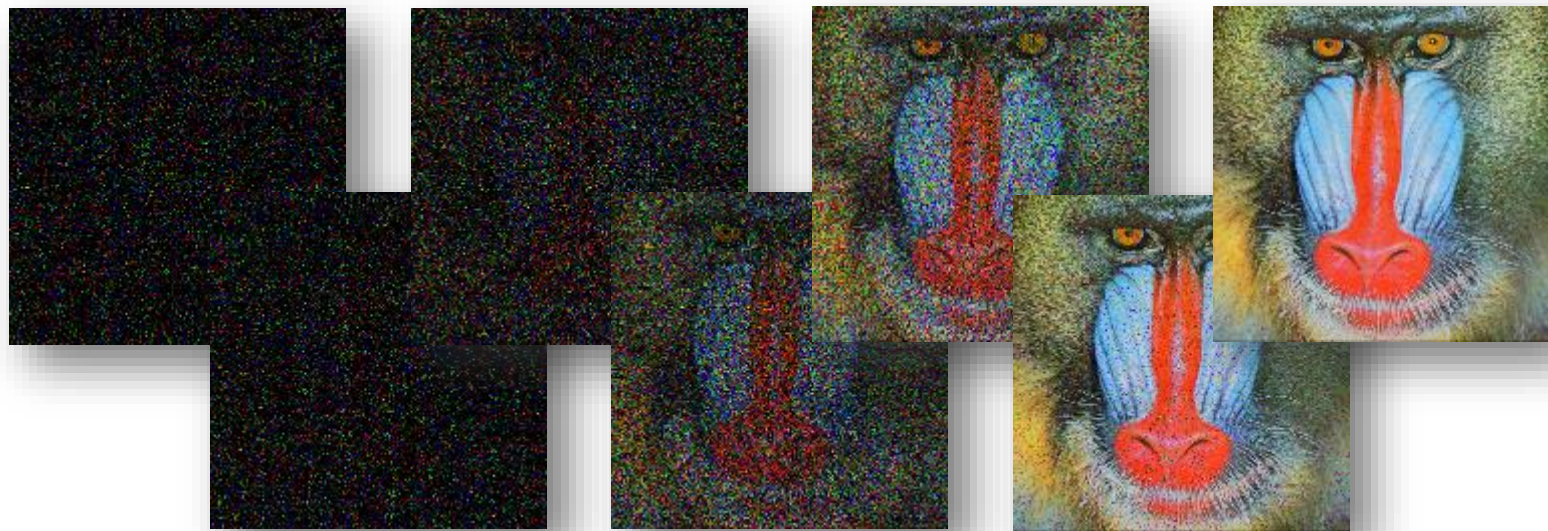
Unreliable Hardware



Unreliable Units (ALUs and Memories)

- May produce incorrect results
- Faster, smaller, and lower power

Image Scaling with Approximate Bilinear Interpolation



20%

40%

60%

80%

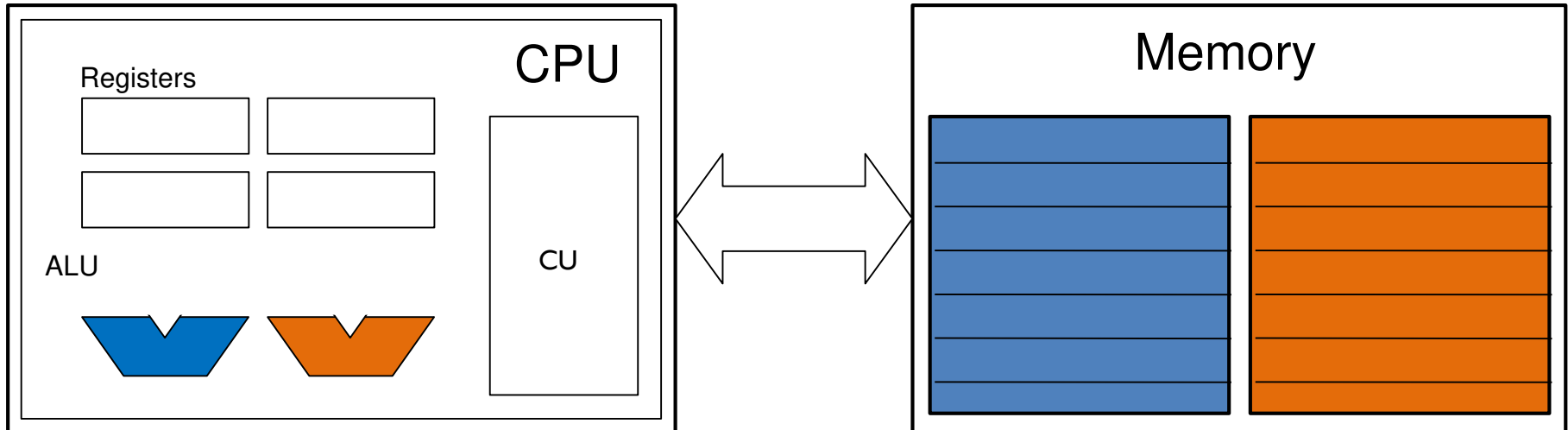
90%

99%

99.9%

Reliability

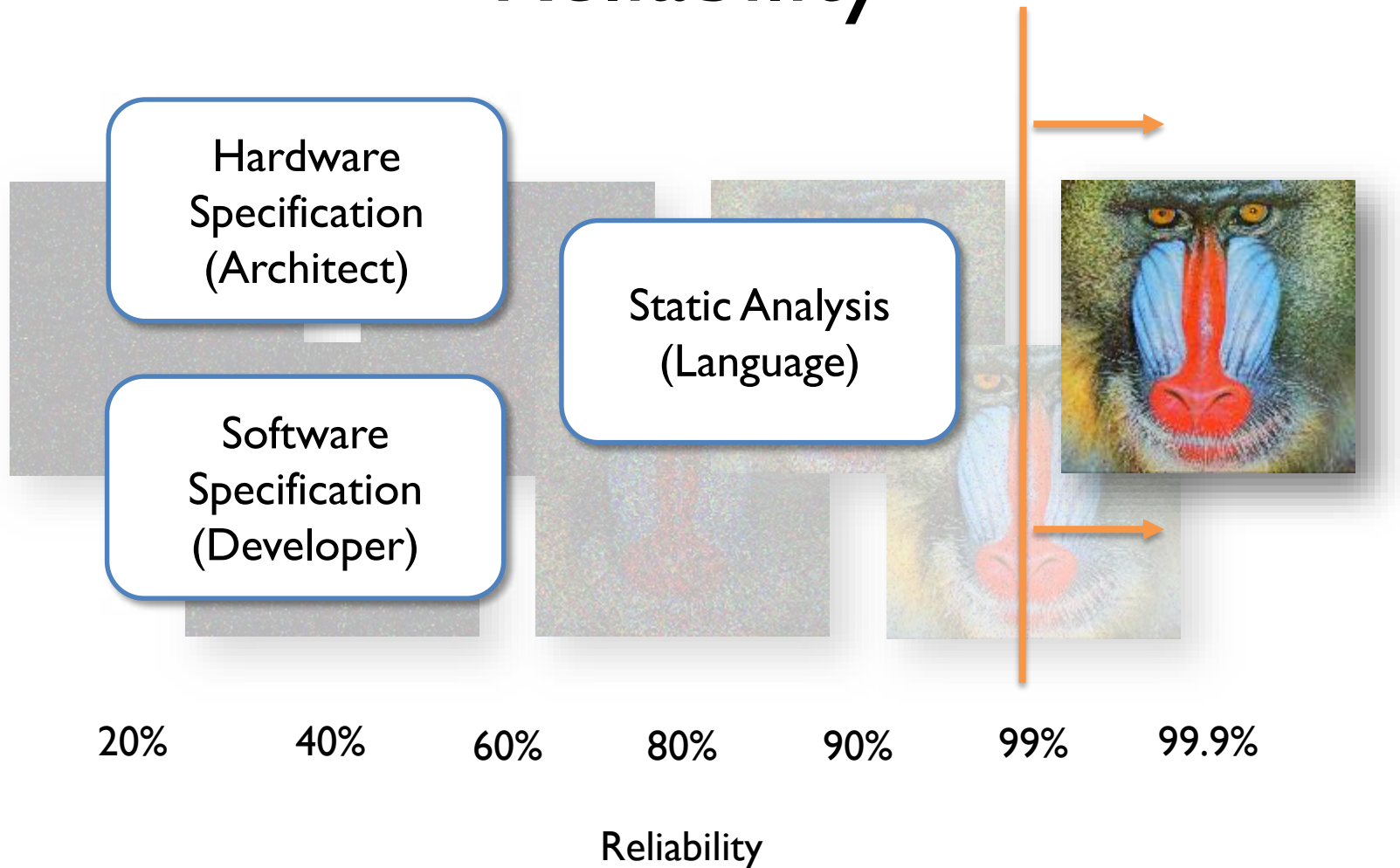
Unreliable Hardware



Necessitates

- Hardware Specification: probability operations execute correctly
- Software Specification: required reliability of computations
- Analysis: verify software satisfies its specification on hardware

Rely: a Language for Quantitative Reliability



Hardware Specification

```
hardware {  
  operator (+) = 1 - 10^-7;  
  operator (-) = 1 - 10^-7;  
  operator (*) = 1 - 10^-7;  
  operator (<) = 1 - 10^-7;  
  memory urel {rd = 1 - 10^-7, wr = 1};  
}
```

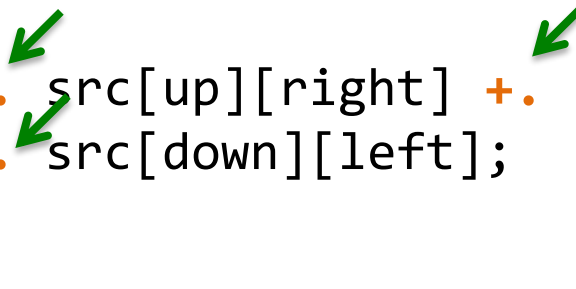
Approximate Bilinear Interpolation in Rely

```
int bilinear_interpolation(int i, int j,
                          int src[][[]], int dest[][[]])
{
    int i_src = map_y(i, src, dest),
        j_src = map_x(j, src, dest);

    int up    = i_src - 1, down    = i_src + 1,
        left  = j_src - 1, right   = j_src + 1;

    int val = src[up][left]    +. src[up][right] +.
              src[down][right] +. src[down][left];

    return 0.25 *. val;
}
```



Unreliable Operations: executed on unreliable ALUs

Approximate Bilinear Interpolation in Rely

```
int bilinear_interpolation(int i, int j,
                           int in urel src[][], int in urel dest[][])
{
    int i_src = map_y(i, src, dest),
        j_src = map_x(j, src, dest);

    int up    = i_src - 1, down  = i_src + 1,
        left  = j_src - 1, right = j_src + 1;

    int in urel val = src[up][left]    +. src[up][right] +.
                      src[down][right] +. src[down][left];

    return 0.25 *. val;
}
```

Unreliable Memories: stored in unreliable SRAM/DRAM

What is reliability?

Semantics of Reliability

- **Reliable Hardware**

- One Execution

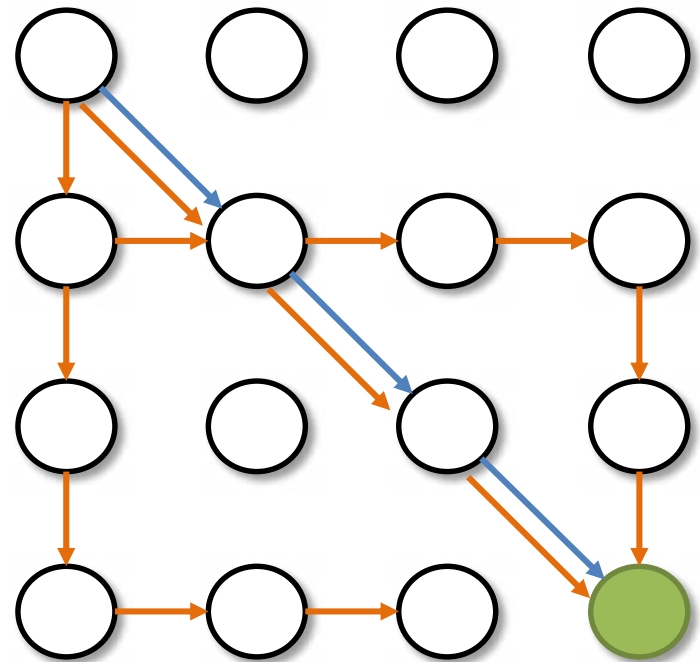
- **Unreliable Hardware**

- Multiple Executions

- **Reliability**

- Probability unreliable execution reaches **same state**

- Or, $R(\{x, y\}) =$ probability over distribution of states that x and y (only) have correct values.



Semantics of Reliability

- **Reliable Hardware**

- One Execution

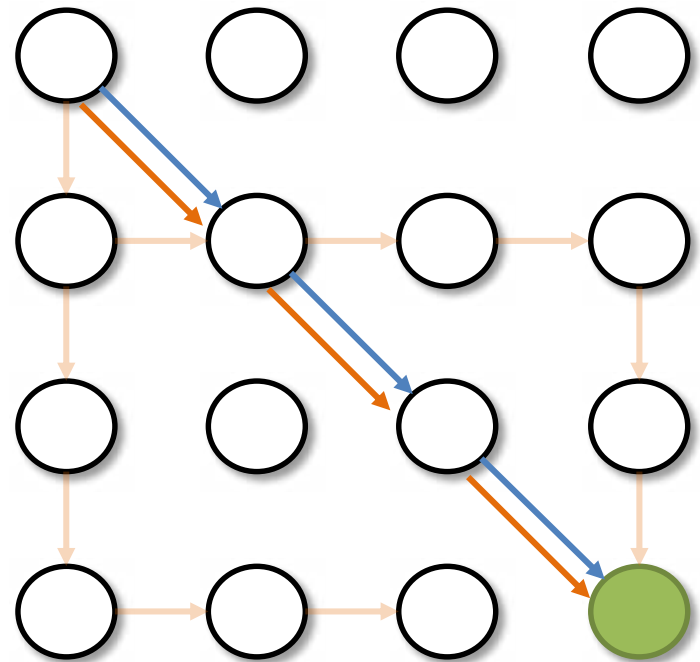
- **Unreliable Hardware**

- Multiple Executions

- **Reliability**

- Probability unreliable execution reaches **same state**

- Or, $R(\{x, y\}) =$ probability over distribution of states that x and y (only) have correct values.



Approximate Bilinear Interpolation Reliability Specification

```
int  
bilinear_interpolation(int i, int j,  
                       int in urel src[][], int in urel dest[][]);
```


Approximate Bilinear Interpolation Reliability Specification

```
int<.99>  
bilinear_interpolation(int i, int j,  
                       int in urel src[][], int in urel dest[][]);
```

- Reliability of output is a function of reliability of inputs

Approximate Bilinear Interpolation Reliability Specification

```
int<.99 * R(i, j, src, dest)>  
bilinear_interpolation(int i, int j,  
    int in urel src[][][], int in urel dest[][][]);
```

- Reliability of output is a function of reliability of inputs
- The term $R(i, j, src, dest)$ abstracts the **joint reliability** of the function's inputs on entry

Approximate Bilinear Interpolation Reliability Specification

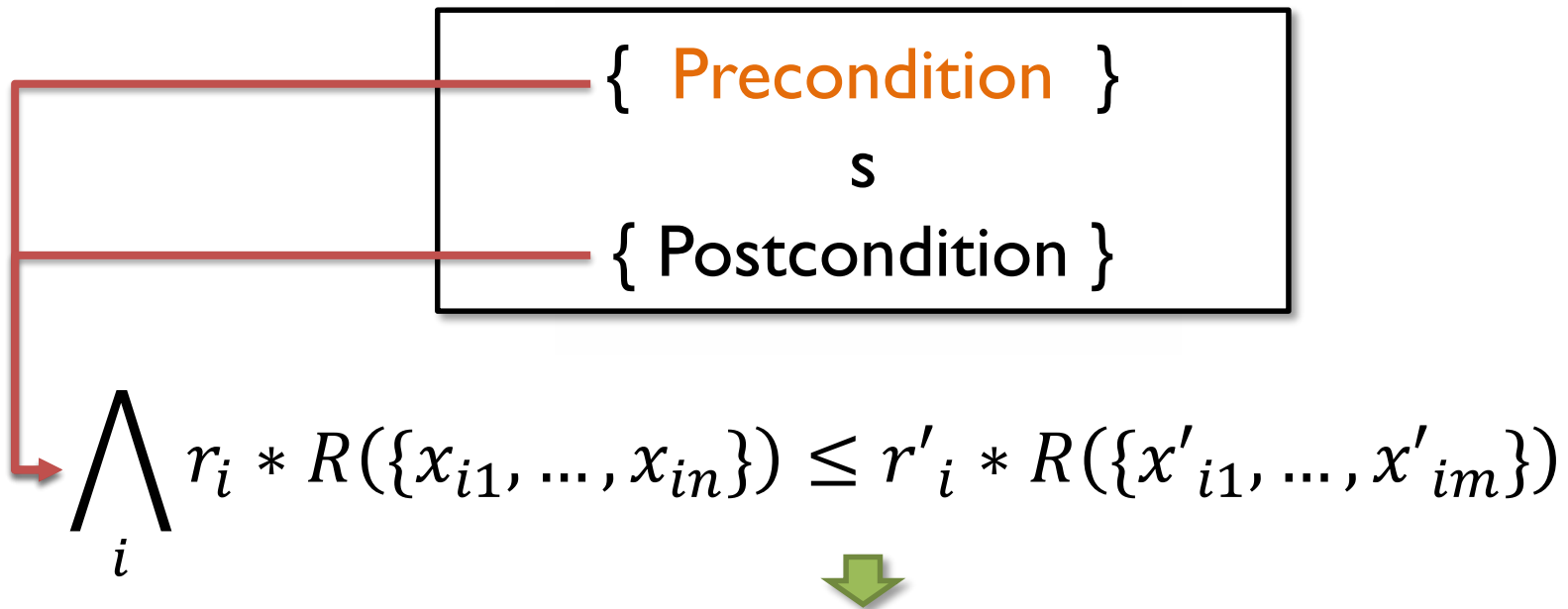
```
int<.99 * R(i, j, src, dest)>  
bilinear_interpolation(int i, int j,  
                       int in urel src[][][], int in urel dest[][][]);
```

- Reliability of output is a function of reliability of inputs
- The term **R(i, j, src, dest)** abstracts the **joint reliability** of the function's inputs on entry
- Coefficient **.99** bounds **reliability degradation**

How does Rely verify reliability?

Rely's Analysis Framework

- **Precondition** generator for statements



Specification

$$0.9 * R(\{x\}) \leq 0.99 * R(\{y\})$$

Computation

Assignment Rule

$$\{r_1 * R(\{x_1 \dots, x_n\}) \leq r_2 * rel(e) * wr(x') * R(\{x'_1, \dots, x'_m\} \cup fv(e))\}$$

$$x' = e$$

$$\{r_1 * R(\{x_1 \dots, x_n\}) \leq r_2 * R(\{x'_1, \dots, x', \dots, x'_m\})\}$$

Assignment Rule

$$\{r_1 * R(\{x_1 \dots, x_n\}) \leq r_2 * rel(e) * wr(x') * R(\{x'_1, \dots, x'_m\} \cup fv(e))\}$$

Unmodified

$$x' = e$$

$$\{r_1 * R(\{x_1 \dots, x_n\}) \leq r_2 * R(\{x'_1, \dots, x', \dots, x'_m\})\}$$

Assignment Rule

$$\{r_1 * R(\{x_1 \dots, x_n\}) \leq r_2 * rel(e) * wr(x') * R(\{x'_1, \dots, x'_m\} \cup fv(e))\}$$

$$x' = e$$

Standard
Substitution

$$\{r_1 * R(\{x_1 \dots, x_n\}) \leq r_2 * R(\{x'_1, \dots, x', \dots, x'_m\})\}$$

Assignment Rule

$$\{r_1 * R(\{x_1 \dots, x_n\}) \leq r_2 * \mathbf{rel}(e) * \mathbf{wr}(x') * R(\{x'_1, \dots, x'_m\} \cup fv(e))\}$$

$$x' = e$$

$$\{r_1 * R(\{x_1 \dots, x_n\}) \leq r_2 * R(\{x'_1, \dots, x', \dots, x'_m\})\}$$

- $rel(e) * wr(x')$ is the probability the expression and write execute correctly

Verifying the Reliability of Bilinear Interpolation

```
int<.99 * R(i,j,src,dest)>
bilinear_interpolation(int i,  int j,
                       int in urel src[][], int in urel dest[][])
{
    int i_src = map_y(i, src, dest),
        j_src = map_x(j, src, dest);

    int up    = i_src - 1, down  = i_src + 1,
        left  = j_src - 1, right = j_src + 1;

    int in urel val = src[up][left]    +. src[up][right] +.
                       src[down][right] +. src[down][left];

    return 0.25 *. val;
}
```

Verifying the Reliability of Bilinear Interpolation

1. Generate postcondition from return statement

```
return 0.25 *. val;
```



$$.99 * R(src, i, j, dest) \leq rd(val) * op(*.) * R(val)$$

2. Work backwards to produce verification condition

$$rd(val) * op(*.) * rd(src)^4 * op(+.)^3 * wr(val) * R(src, i, j, dest) \leq .99 * R(src, i, j, dest)$$

3. Use hardware specification to replace reliabilities

Reliability of return

Reliability of sum of neighbors

$$(1 - 10^{-7}) * (1 - 10^{-7}) * (1 - 10^{-7})^4 * (1 - 10^{-7})^3 * 1.0 * R(src, i, j, dest) \leq .99 * R(src, i, j, dest)$$

Verifying the Reliability of Bilinear Interpolation

1. Generate postcondition from return statement

`return 0.25 * . val;`



$$.99 * R(src, i, j, dest) \leq rd(val) * op(*.) * R(val)$$

2. Work backwards to produce verification condition

$$.99 * R(src, i, j, dest) \leq rd(val) * op(*.) * rd(src)^4 * op(+.)^3 * wr(val) * R(src, i, j, dest)$$

3. Use hardware specification to replace reliabilities

$$.99 * R(src, i, j, dest) \leq .999999 * R(src, i, j, dest)$$

4. Discharge Verification Condition

Verification Condition Checking Insight

$$\bigwedge_i r_i * \underline{R(\{x_{i1}, \dots, x_{in}\})} \leq r'_i * \underline{R(\{x'_{i1}, \dots, x'_{im}\})}$$

Computing full **joint distributions** is intractable
and input distribution dependent

$$\{x'_1, \dots, x'_m\} \subseteq \{x_1, \dots, x_n\} \rightarrow \\ R(\{x_1, \dots, x_n\}) \leq R(\{x'_1, \dots, x'_m\})$$

Conjunct Checking

- A conjunct is implied by a pair of constraints

$$r_1 \leq r_2 \quad \{x'_1, \dots, x'_m\} \subseteq \{x_1, \dots, x_n\}$$

$$r_1 * R(\{x_1, \dots, x_n\}) \leq r_2 * R(\{x'_1, \dots, x'_m\})$$

- Decidable, efficiently checkable, and **input distribution agnostic**

Verification Condition Checking for Approximate Bilinear Interpolation

Hardware
Specification



$.99 \leq .9999999$

Data
Dependences

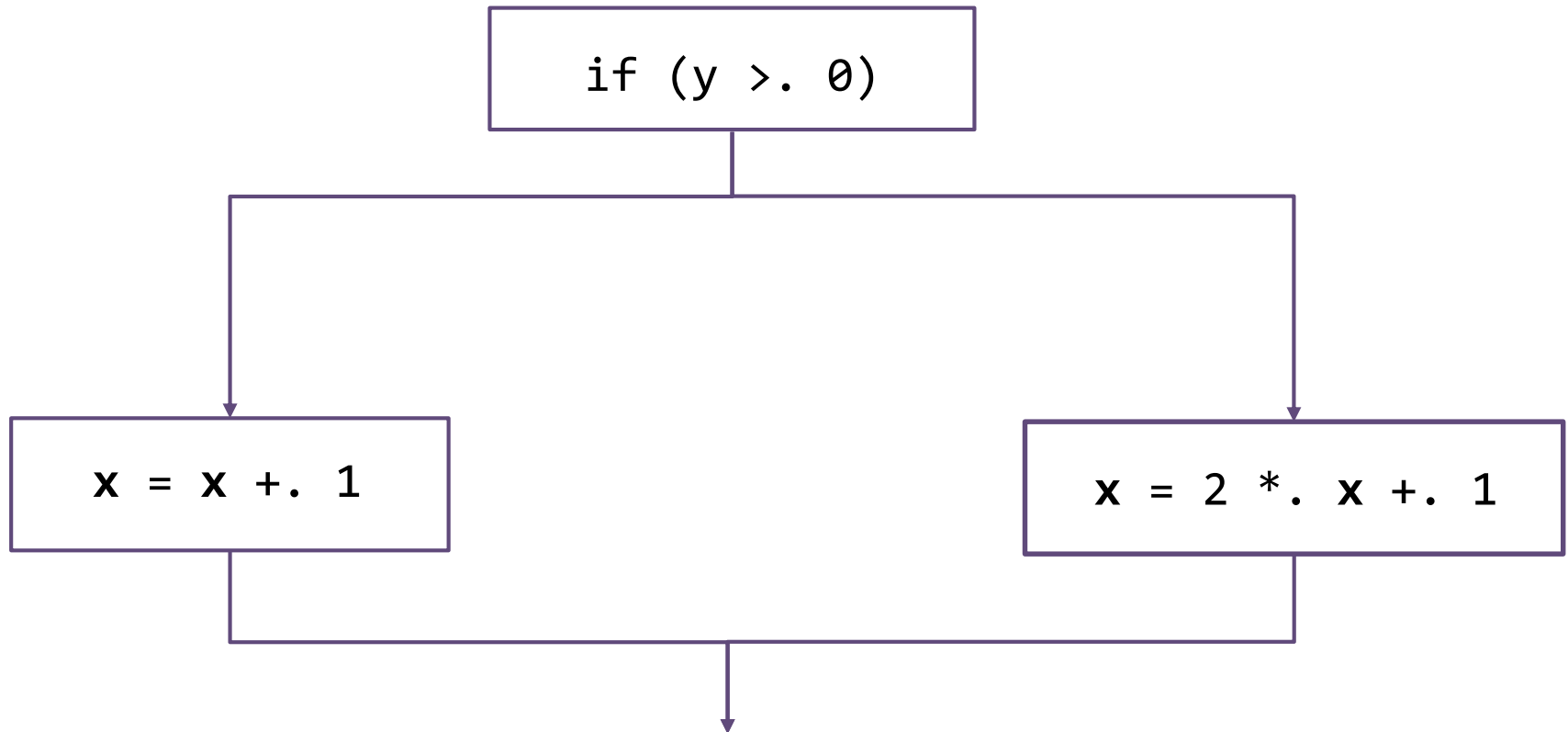


$\{\text{src}, i, j, \text{dest}\} \subseteq \{\text{src}, i, j, \text{dest}\}$

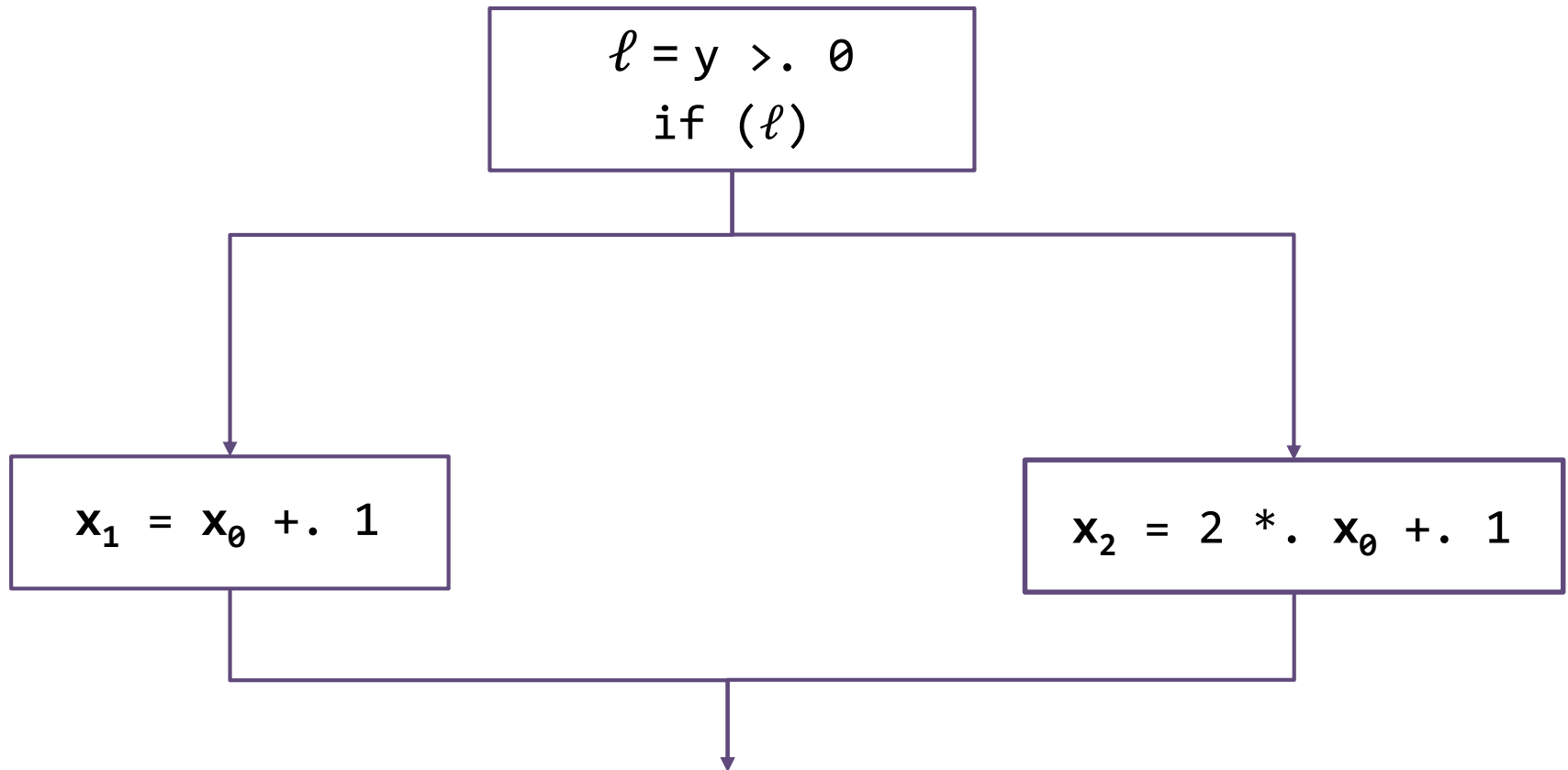
$.99 * R(\{\text{src}, i, j, \text{dest}\}) \leq .9999999 * R(\{\text{src}, i, j, \text{dest}\})$

**What about...programs?
(conditionals, loops, and functions)**

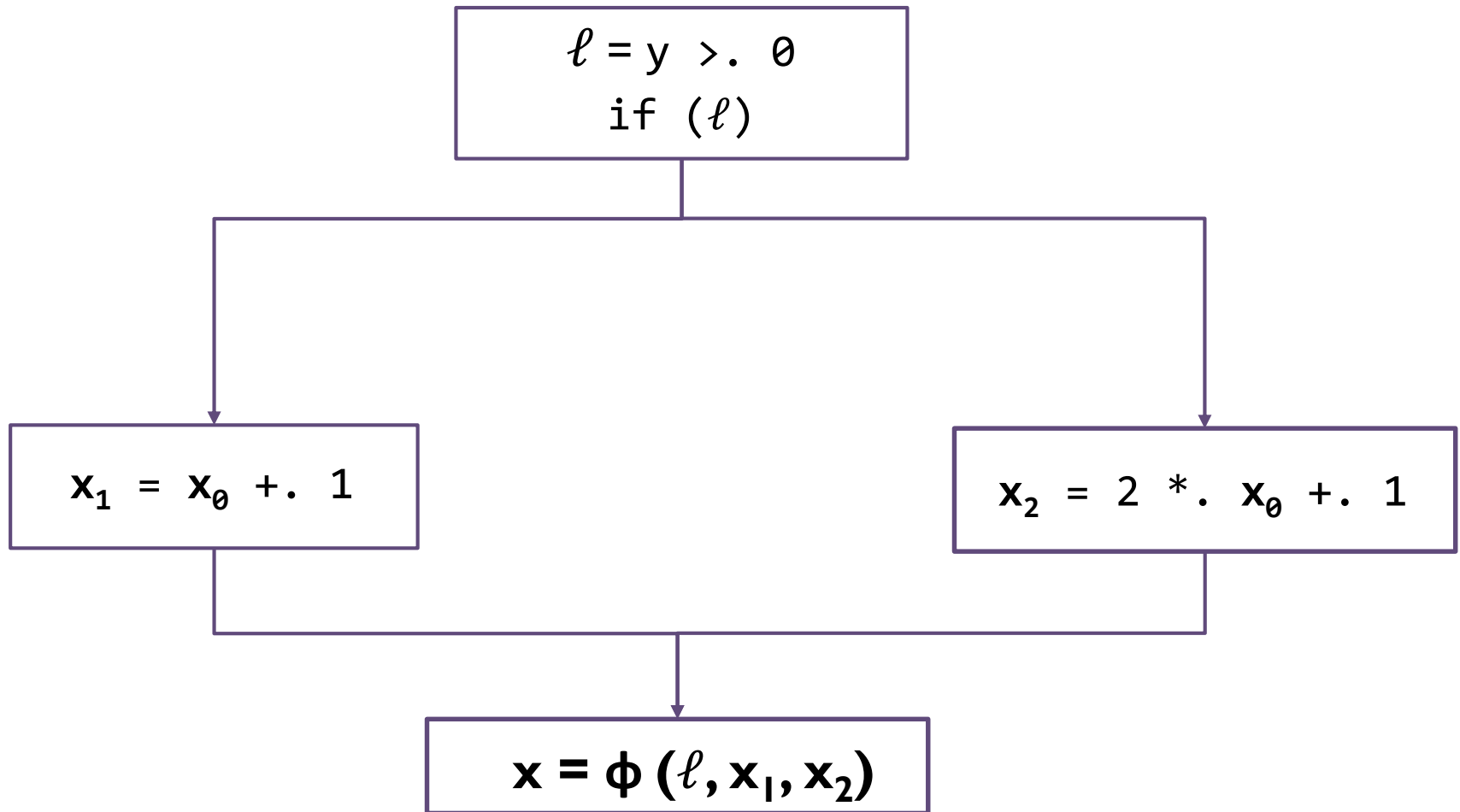
Conditionals



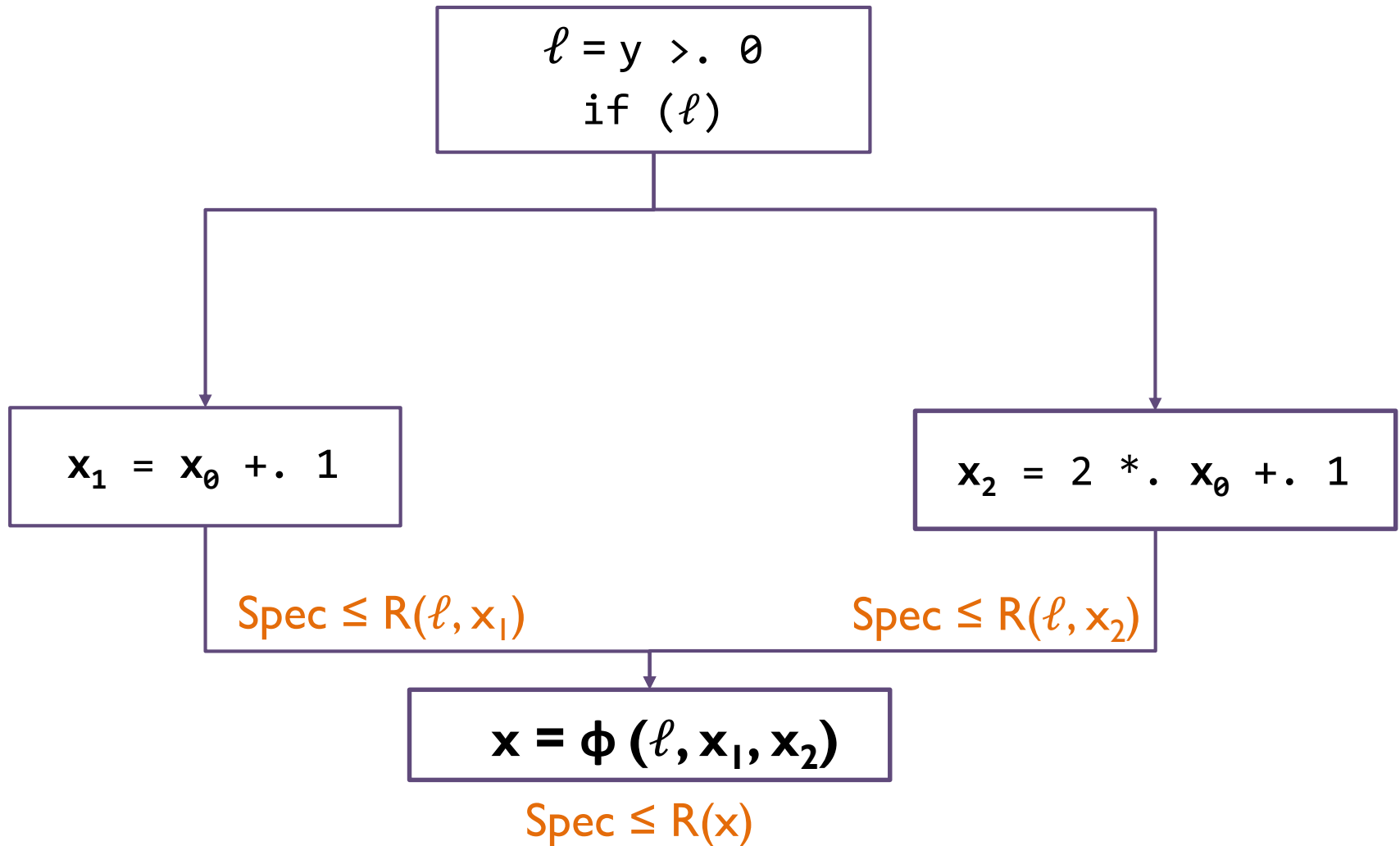
Conditionals



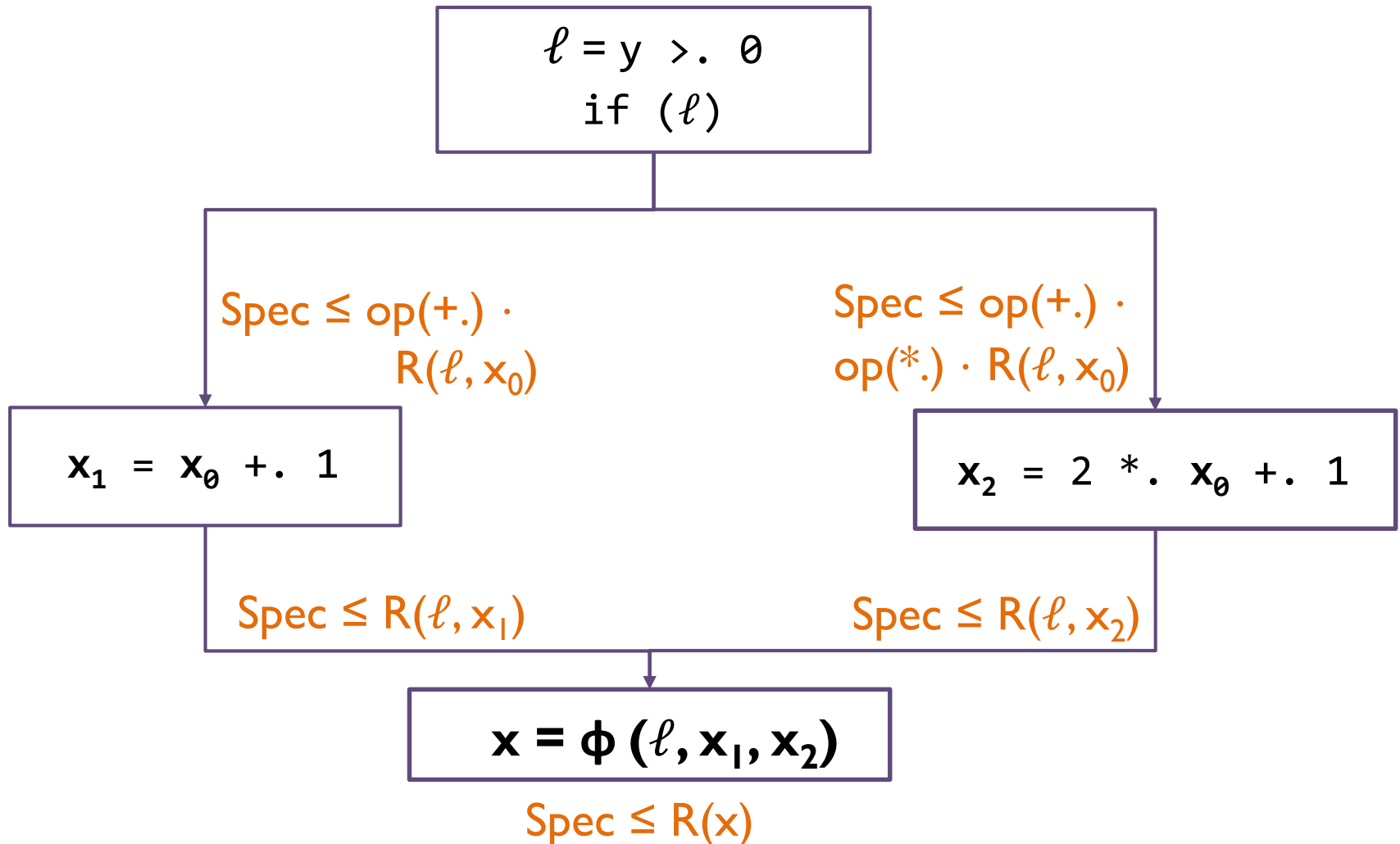
Conditionals



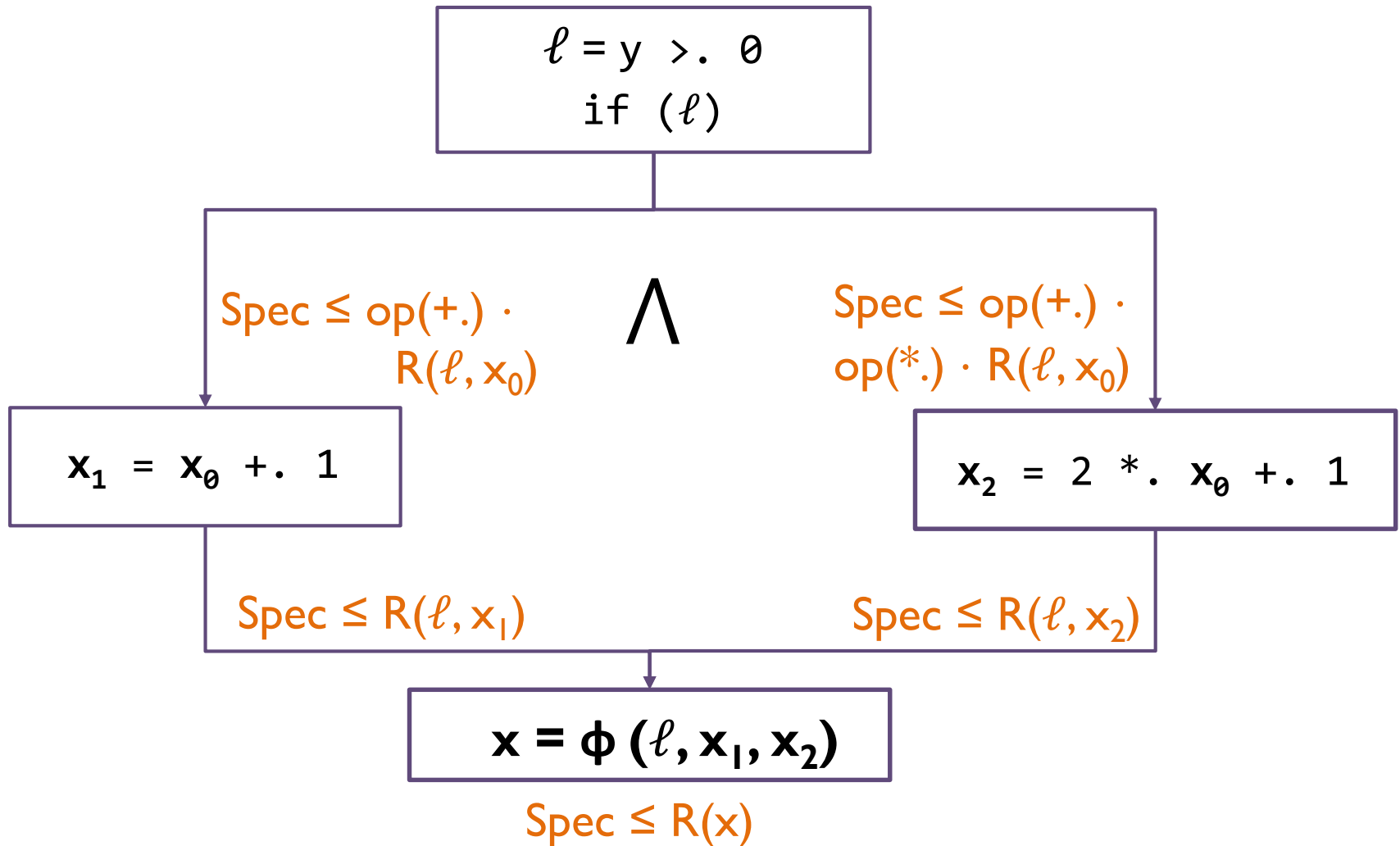
Conditionals



Conditionals



Conditionals



Conditionals

$$\text{Spec} \leq \text{op}(+.) \cdot \text{op}(>.) \cdot R(x_0, y) \quad \wedge \\ \text{Spec} \leq \text{op}(+.) \cdot \text{op}(*.) \cdot \text{op}(>.) \cdot R(x_0, y)$$

$$\ell = y > . 0 \\ \text{if } (\ell)$$

$$\text{Spec} \leq \text{op}(+.) \cdot R(\ell, x_0)$$

 \wedge

$$\text{Spec} \leq \text{op}(+.) \cdot \text{op}(*.) \cdot R(\ell, x_0)$$

$$x_1 = x_0 + . 1$$

$$x_2 = 2 * . x_0 + . 1$$

$$\text{Spec} \leq R(\ell, x_1)$$

$$\text{Spec} \leq R(\ell, x_2)$$

$$x = \phi(\ell, x_1, x_2)$$

$$\text{Spec} \leq R(x)$$

Simplification

$$\text{Spec} \leq \text{op}(+.) \cdot \text{op}(>.) \cdot R(x_0, y) \quad \wedge$$
$$\text{Spec} \leq \text{op}(+.) \cdot \text{op}(*.) \cdot \text{op}(>.) \cdot R(x_0, y)$$

$$\ell = y > . 0$$
$$\text{if } (\ell)$$

$$\text{Spec} \leq \text{op}(+.) \cdot R(\ell, x_0)$$



$$\text{Spec} \leq \text{op}(+.) \cdot \text{op}(*.) \cdot R(\ell, x_0)$$

$$x_1 = x_0 + . 1$$

$$x_2 = 2 * . x_0 + . 1$$

$$\text{Spec} \leq R(\ell, x_1)$$

$$\text{Spec} \leq R(\ell, x_2)$$

$$x = \phi(\ell, x_1, x_2)$$

$$\text{Spec} \leq R(x)$$

Simplification

~~$\text{Spec} \leq \text{op}(+.) \cdot \text{op>(>.)} \cdot R(x_0, y) \wedge$~~
 $\text{Spec} \leq \text{op}(+.) \cdot \text{op}(*.) \cdot \text{op>(>.)} \cdot R(x_0, y)$

$$\ell = y > . 0$$

if (ℓ)

~~$\text{Spec} \leq \text{op}(+.) \cdot$~~
 ~~$R(\ell, x_0)$~~

\wedge

$$\text{Spec} \leq \text{op}(+.) \cdot$$
$$\text{op}(*.) \cdot R(\ell, x_0)$$

$$x_1 = x_0 + . 1$$

$$x_2 = 2 * . x_0 + . 1$$

$$\text{Spec} \leq R(\ell, x_1)$$

$$\text{Spec} \leq R(\ell, x_2)$$


$$x = \phi(\ell, x_1, x_2)$$

$$\text{Spec} \leq R(x)$$

Loops

- Reliability of loop-carried, unreliably updated variables decreases monotonically

```
int sum = 0;
for (int i = 0; i < n; i = i + 1) {
    sum = sum +. a[i];
}
```

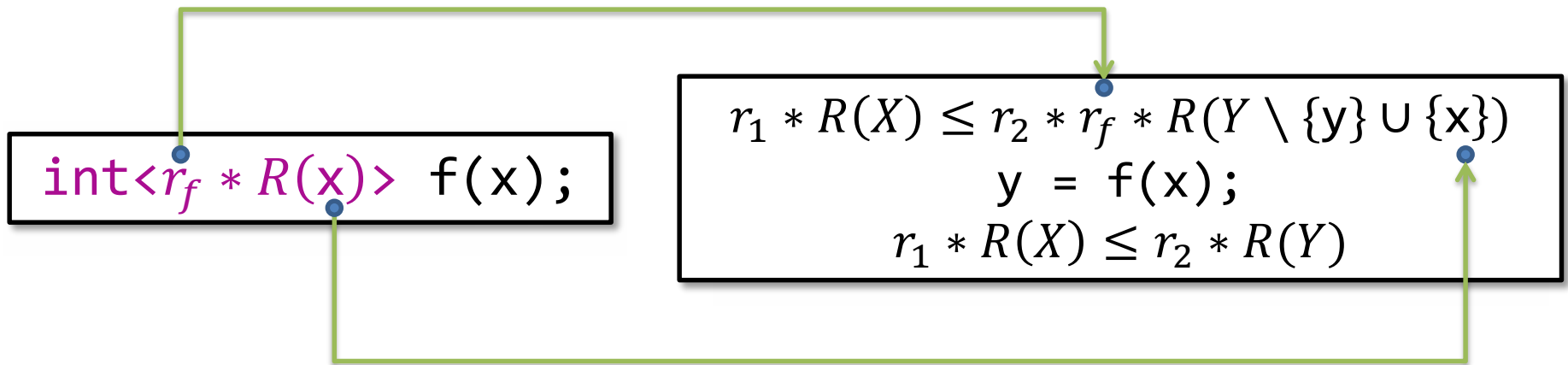


R(sum) depends on
n **unreliable adds**

- **Finitely Bounded Loops:** bounded decrease
- **Unbounded loops:** conservative result is 0

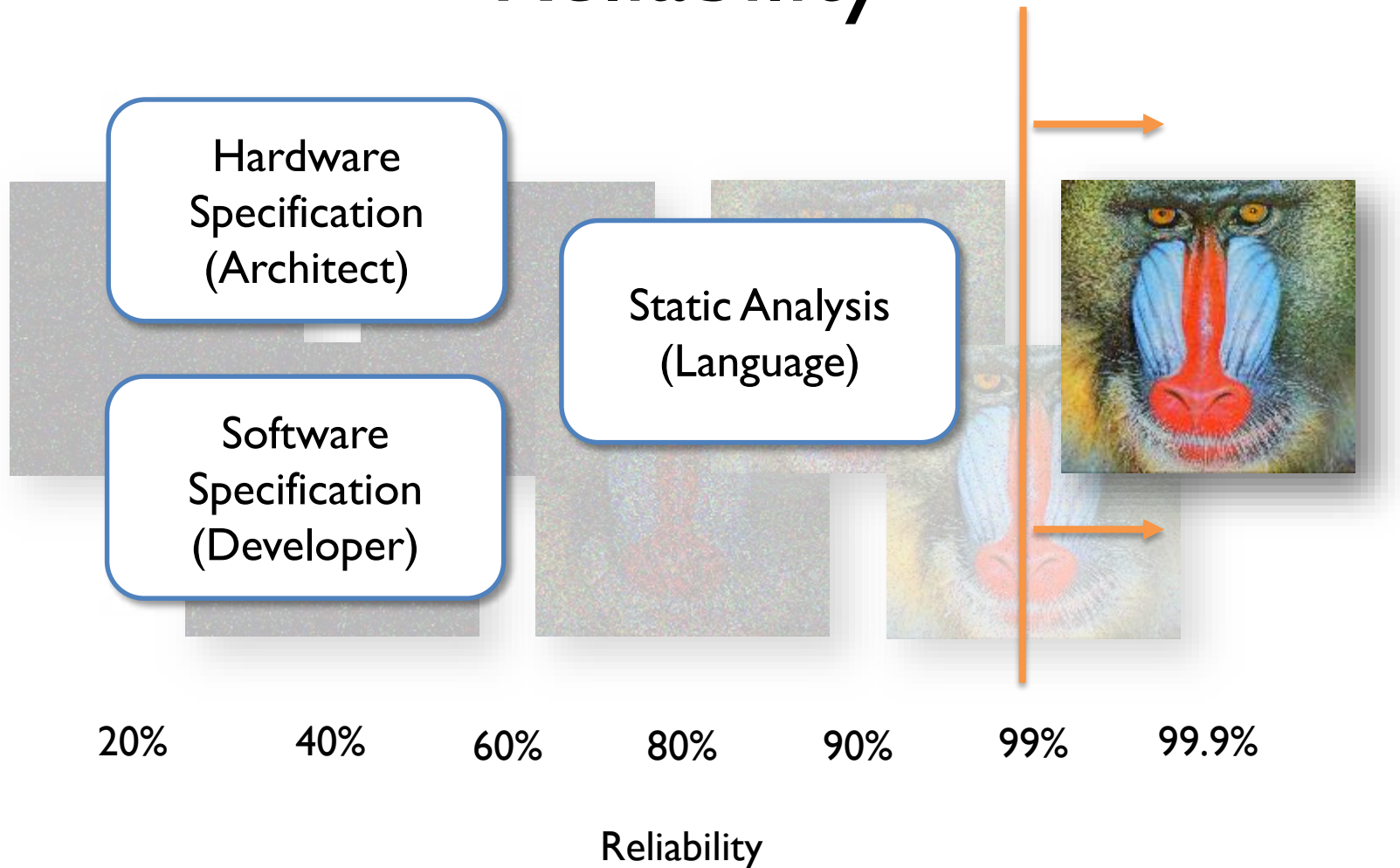
Functions

- Verification is **modular** (assume/guarantee)



- Recursion similar to loops: unreliably updated variables naturally have 0 reliability

Rely: a Language for Quantitative Reliability



Evaluation

- **Experiment #1:** verify specifications
 - How does the analysis behave?

Benchmarks

- **newton:** zero-finding using Newton's method
- **secant:** zero-finding using Secant Method
- **coord:** Cartesian to polar coordinate converter
- **search_ref:** motion estimation
- **mat_vec:** matrix-vector multiply
- **hadamard:** frequency-domain pixel-block difference metric

Experiment #1: Results

Benchmark	LOC	Time (ms)	Conjuncts	
			w/o	with
newton	21	8	82	1
secant	30	7	16356	2
coord	36	19	20	1
search_ref	37	348	36205	3
matvec	32	110	1061	4
hadamard	87	18	3	3

Observation: small number of conjuncts with simplification

Evaluation

- **Experiment #2:** application scenarios
 - How to use reliabilities?

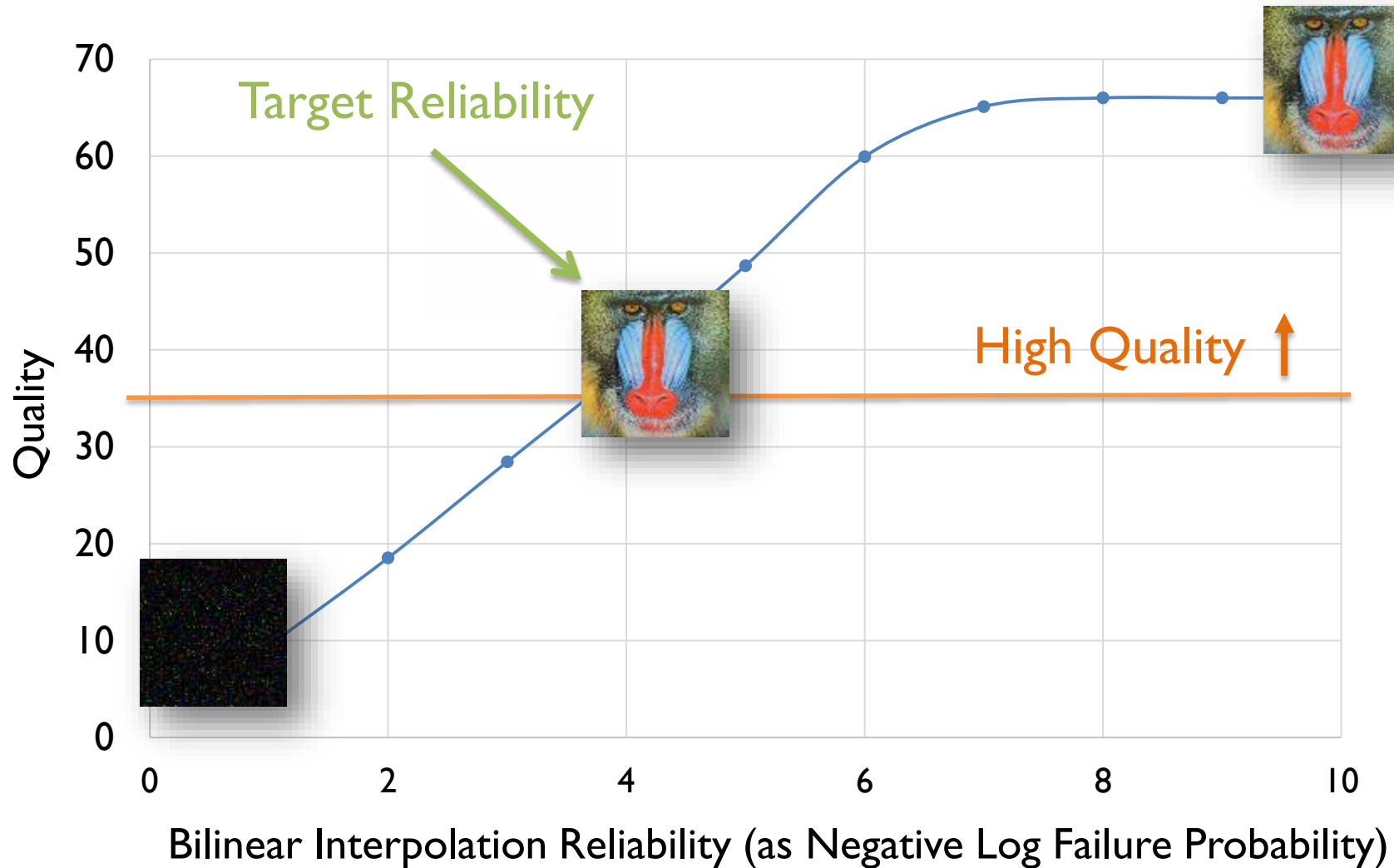
Checkable Computations

- A **simple checker** can validate whether the program produced a correct result
- Execution time optimization:

$T_{reliable}$ **vs.**

$$T_{unreliable} + T_{checker} + (1 - r) \cdot T_{reliable}$$

Approximate Computations



Other Concerns for Unreliable Hardware

Safety: does the program always produce a result?

- no failures or ill-defined behaviors

[Misailovic et al. ICSE '10; Carbin et al. ISSTA '10; Sidiroglou et al. FSE'11; Carbin et al., PLDI '12; Carbin et al., PEPM '13]

Accuracy: is result accurate enough?

- small expected error

[Rinard ICS'06; Misailovic et al., ICSE '10; Hoffmann et al. ASPLOS '11; Misailovic et al. SAS '11; Sidiroglou et al. FSE'11; Zhu et al. POPL '12; Misailovic et al. RACES '12]

Takeaway

- Separating approximate computation isn't enough
- Acceptability of results **depends on reliability**

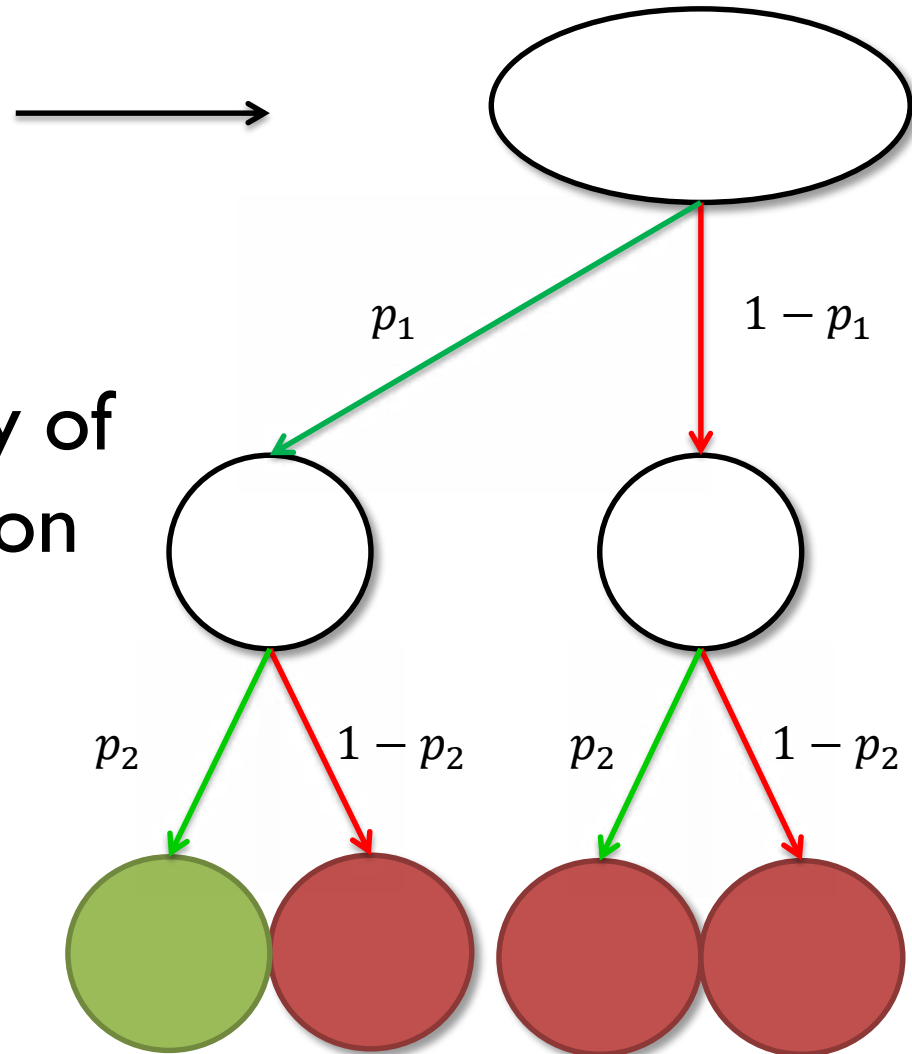
Rely

- Architect provides hardware specification
- Developer provides software specification
- Rely provides verified reliability guarantee

Backup Slides

Semantic Model

- Execution of e is a **stochastic process**
- Independent probability of failure for each operation
- Reliability is probability of fully reliable path



Semantic Formalization

- **Probabilistic** transition system

$$\langle s, \sigma \rangle \xrightarrow{p} \langle s', \sigma' \rangle$$

See paper for
inference rules!

- Set of possible executions on unreliable hardware gives **distributions** of states

$$\phi \in \Sigma \rightarrow \mathbb{R} \quad \langle s, \phi \rangle \Rightarrow \phi$$

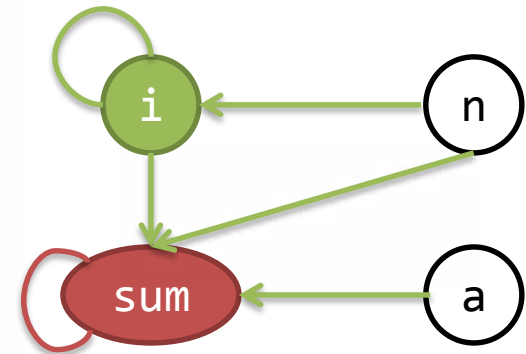
- **Predicates** defined over distributions

$$\llbracket P \rrbracket \in \wp(\Phi)$$

Identifying Reliably Update Variables

- **Reliably** updated vs. **unreliably** updated variables

```
int sum = 0;
for (int i = 0; i < n; i = i + 1)
{
    sum = sum + a[i];
}
```



- Dependence graph gives classification
- Reliably updated variables have same reliability