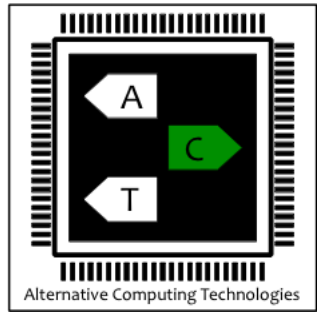
FLEXJAVA: Language Support for Safe and Modular Approximate Programming

Jongse Park, Hadi Esmaeilzadeh, Xin Zhang,
Mayur Naik, William Harris

Alternative Computing Technologies (**ACT**) Lab
Georgia Institute of Technology

ESEC/FSE 2015

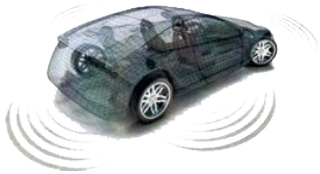


Energy is a primary constraint

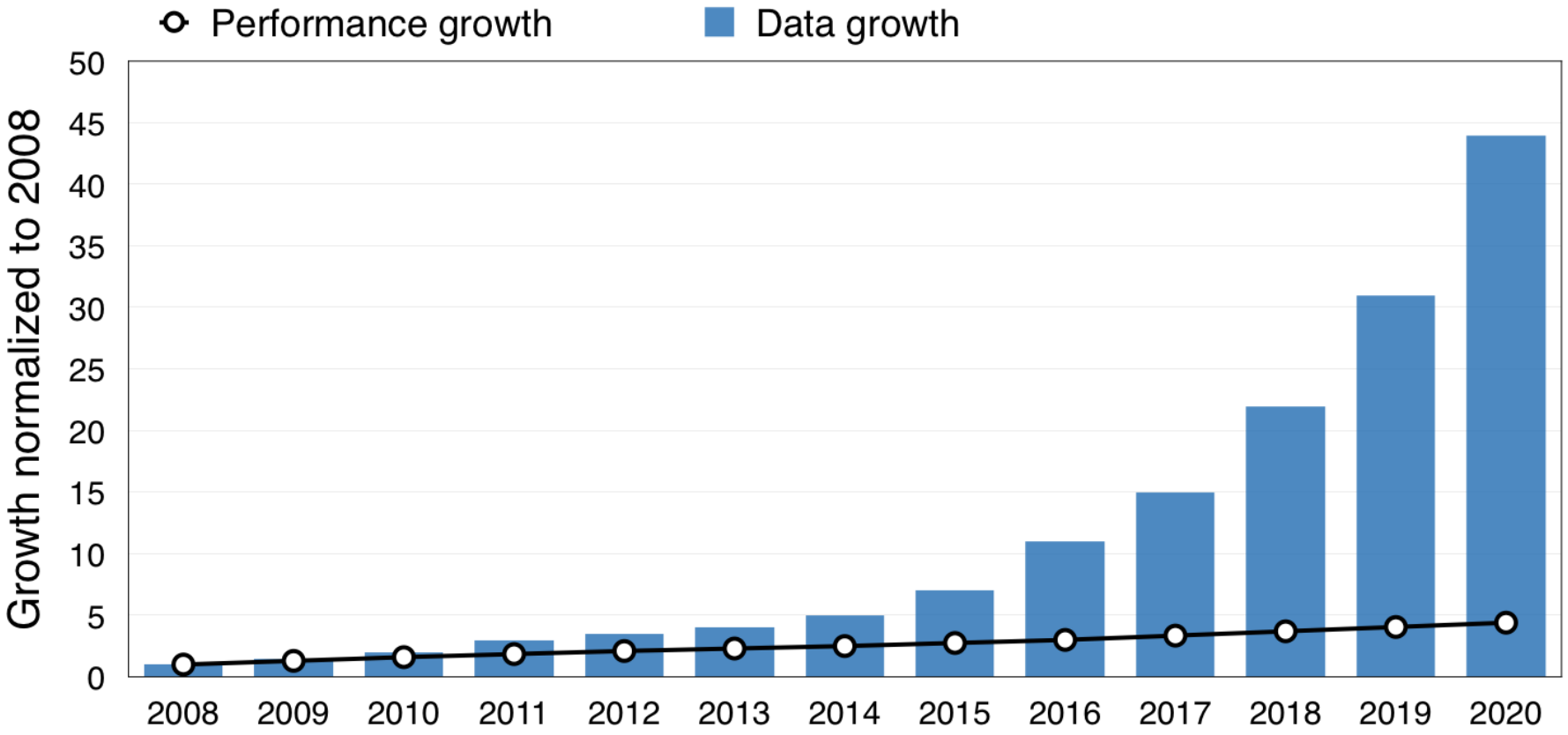
Data Center

Mobile

Internet of Things



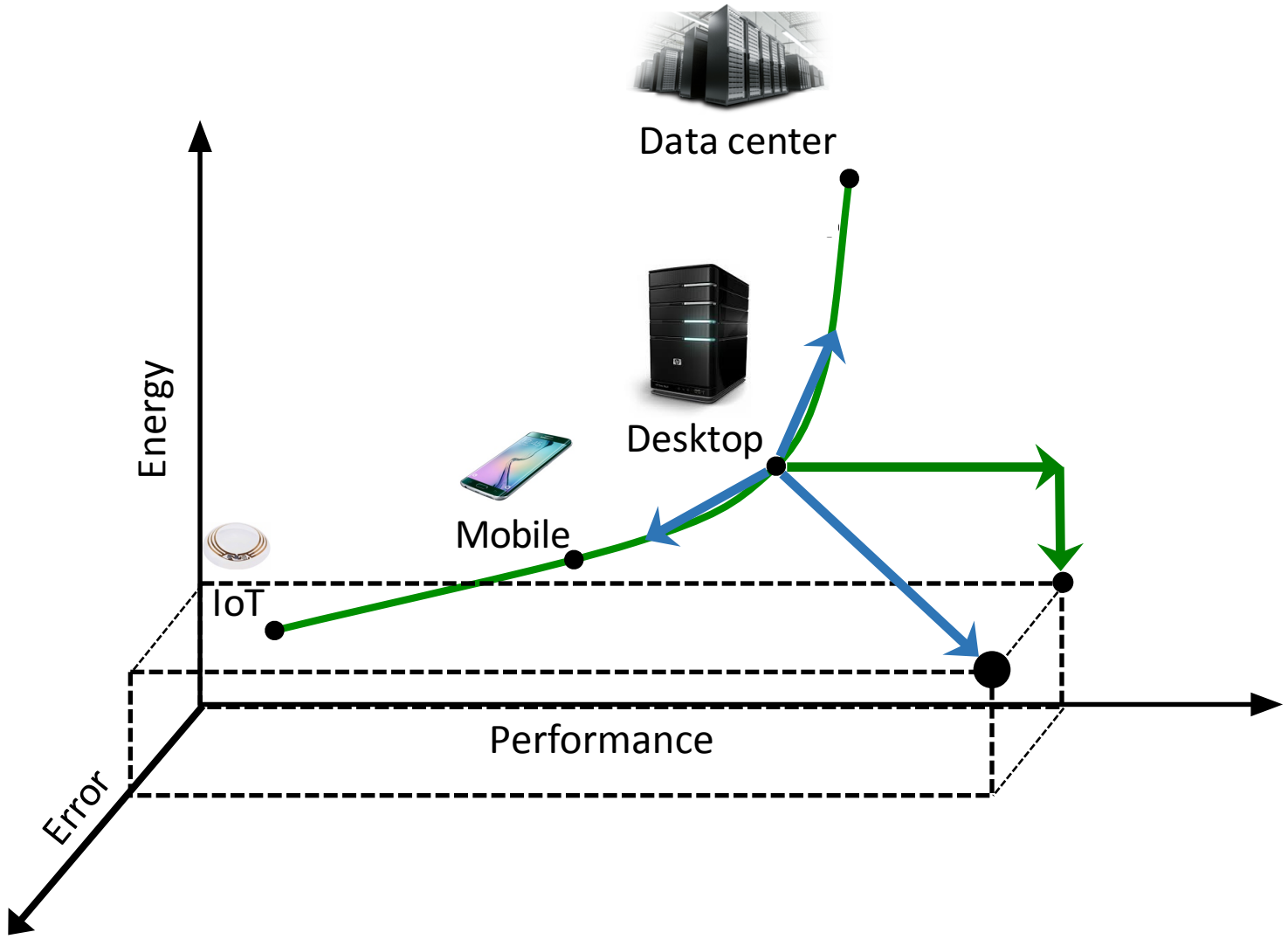
Data growth vs performance



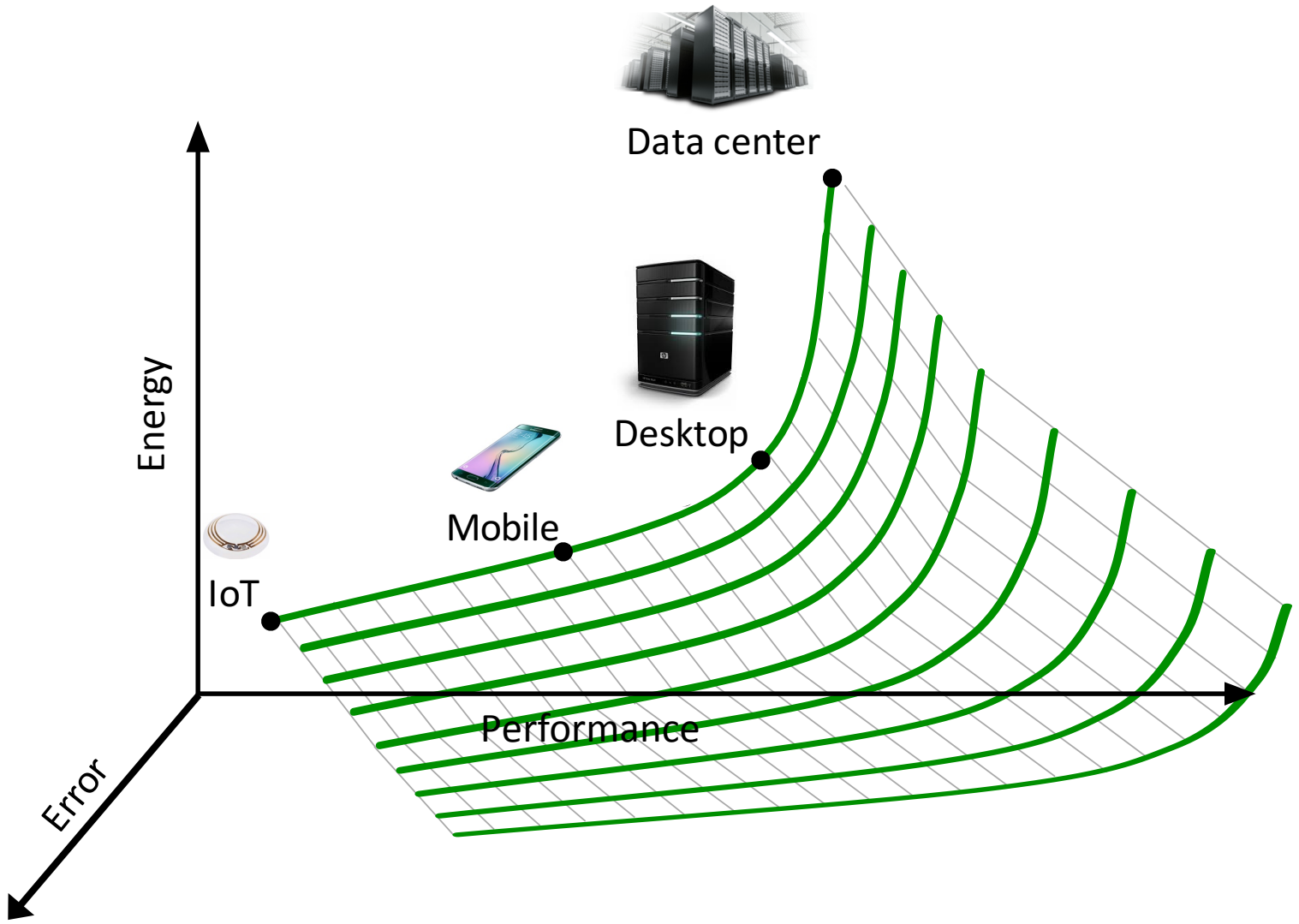
- Data growth trends: IDC's Digital Universe Study, December 2012
- Performance growth trends: Esmaeilzadeh et al, "Dark Silicon and the End of Multicore Scaling," ISCA 2011

Adding a third dimension

Embracing Error



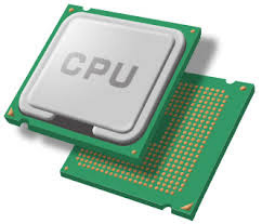
Navigating a three dimensional space



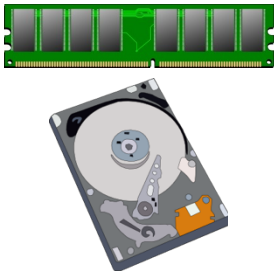
Approximate computing

Embracing error

Relax the abstraction of *"near perfect"* **accuracy** **in**



Processing



Storage

Communication

Allows **errors** to happen to improve

performance

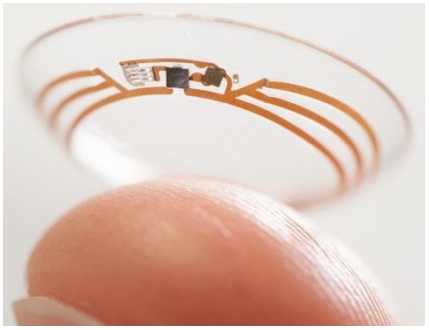
resource utilization **efficiency**

New landscape of computing

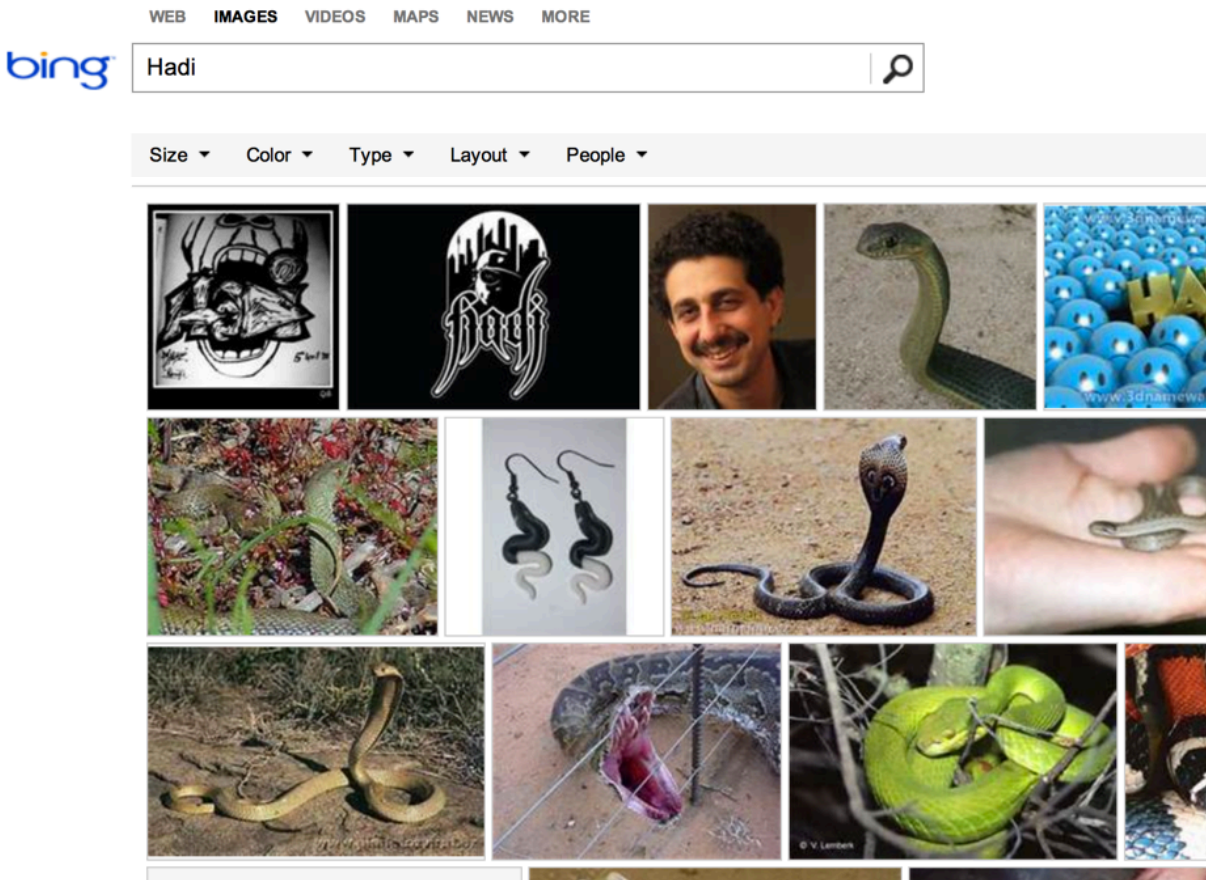
Personalized and targeted computing

Mobile Computing

IoT Computing



Cloud Computing



Classes of approximate applications

Programs with analog inputs

- Sensors, scene reconstruction

Programs with analog outputs

- Multimedia

Programs with multiple possible answers

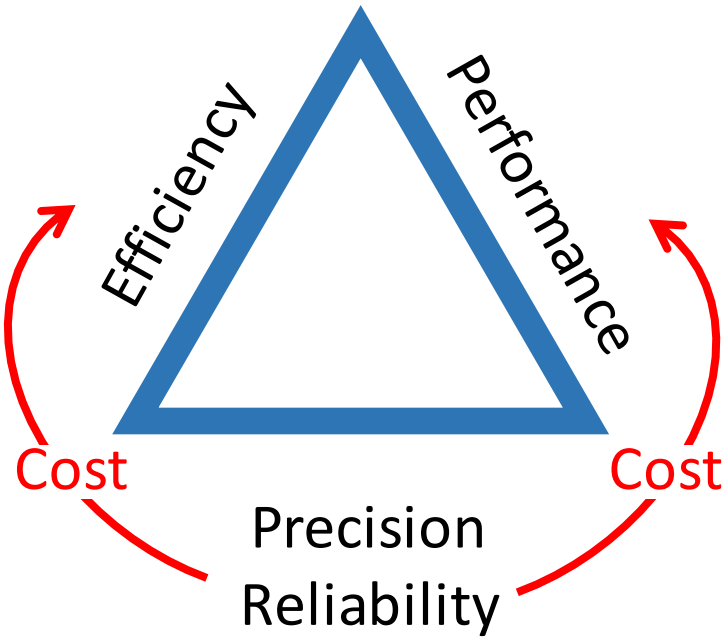
- Web search, machine learning

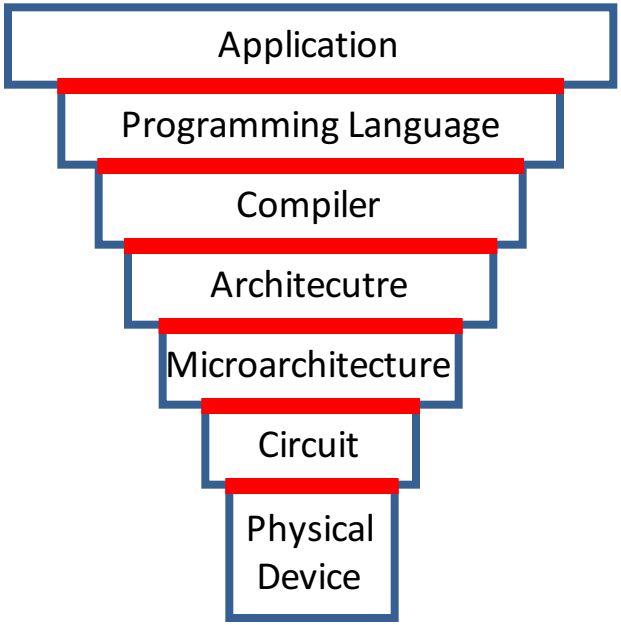
Convergent programs

- Gradient descent, big data analytics

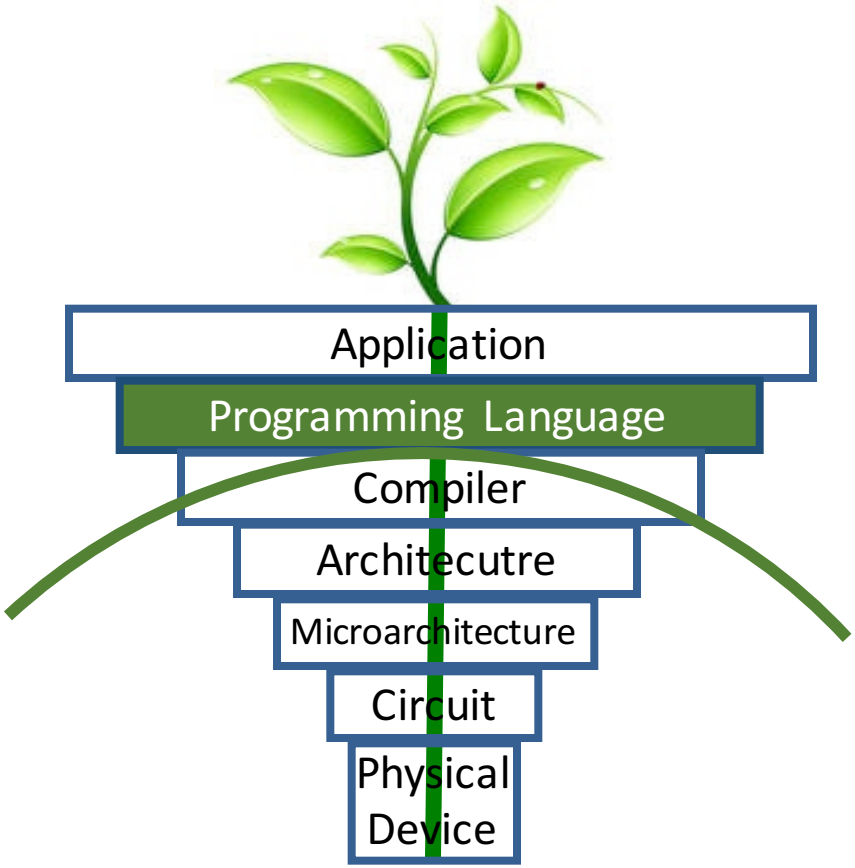
Avoiding overkill design

Approximate Computing





Cross-stack approach



WH² of Approximation

What
to approximate?

How much
to approximate?

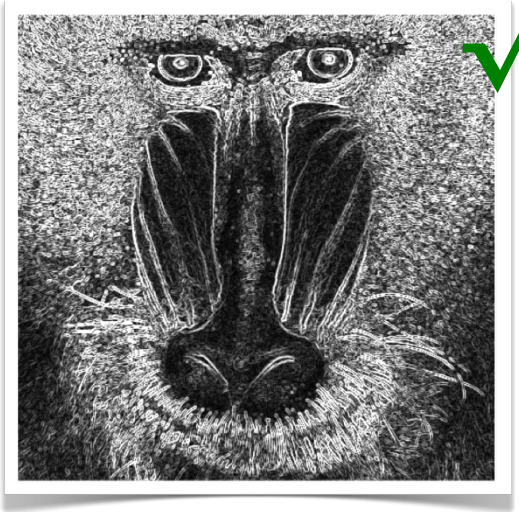
How
to approximate?

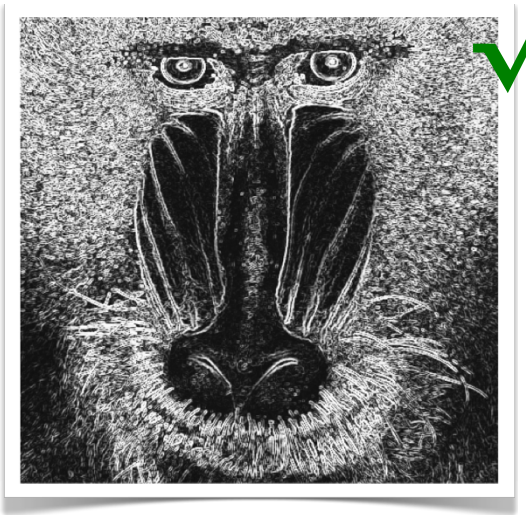
Language

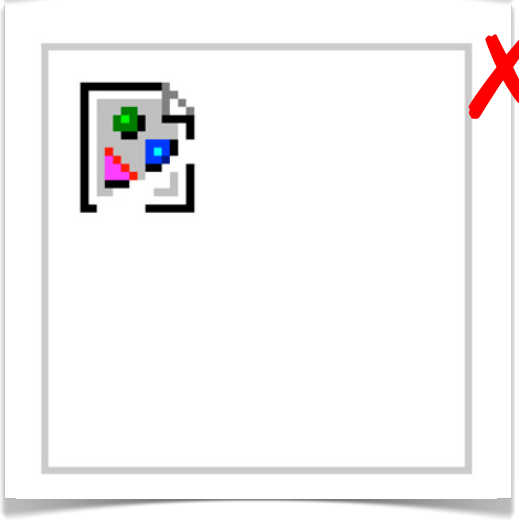
Compiler

Runtime

Hardware








You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.
Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de réinitialisation.
Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.
コンピュータを再起動する必要があります。パワーボタンを数秒間押し続けるか、リセットボタンを押してください。

Approximate program

Separation

Goal

Design an **automated** and **high-level** programming language
for approximate computing

Criteria

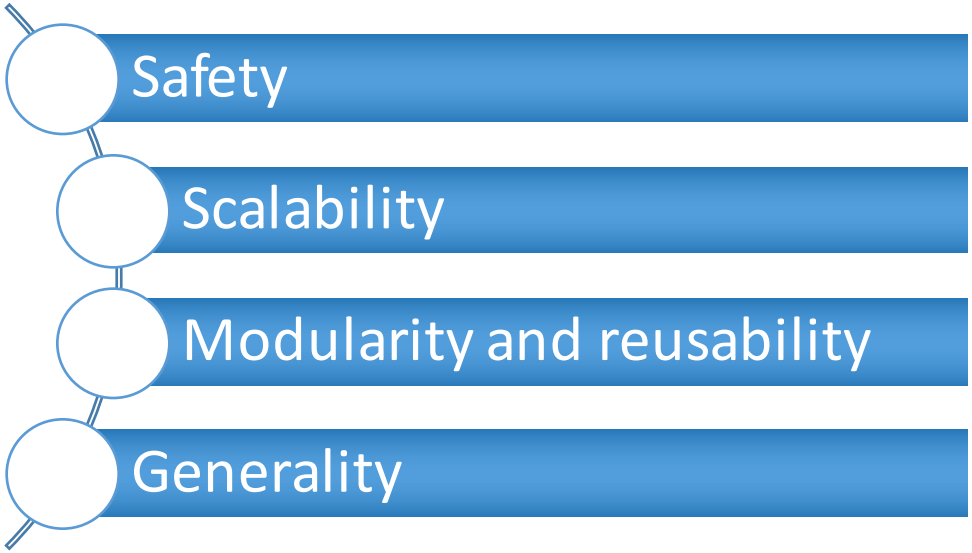
1) **Safety**
2) **Scalability**
3) **Modularity and reusability**
4) **Generality**

FLEXJAVA

Lightweight set of language extensions

Reducing annotating effort

Language-compiler co-design



FLEXJAVA Annotations

1. Approximation for individual data and operations
   loosen (variable)
   loosen_invasive (variable)
   tighten (variable)

2. Approximation for a code block
   begin_loose ("TECHNIQUE", ARGUMENTS)
   end_loose (ARGUMENTS)

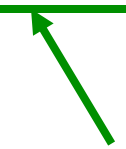
3. Expressing the quality requirement
   loosen (variable, QUALITY_REQUIREMENT);
   loosen_invasive (variable, QUALITY_REQUIREMENT);
   end_loose (ARGUMENTS, QUALITY_REQUIREMENT);

Safe and scalable approximation

```
float computeLuminance (float r,
                        float g,
                        float b) {
    float luminance = r * 0.3f + g * 0.6f + b * 0.1f;

    loosen (luminance);
    return luminance;
}
```

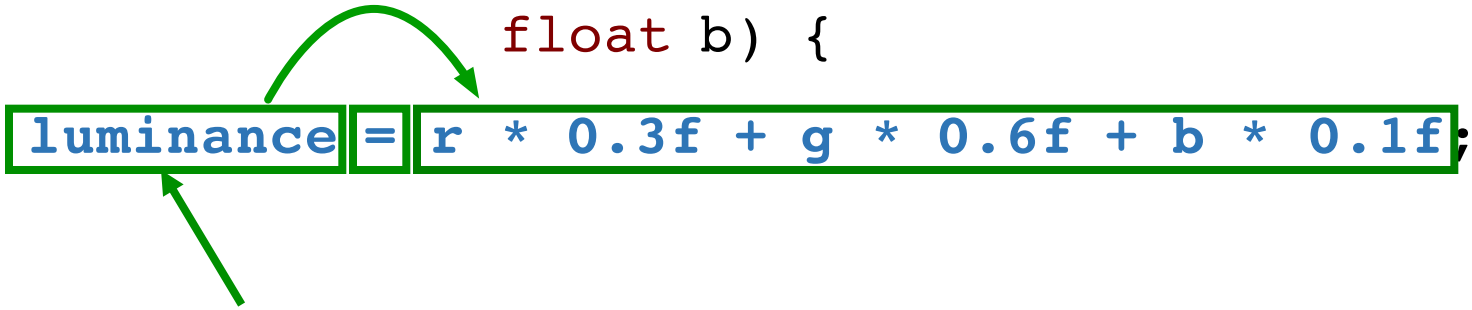
Safe and scalable approximation

```
float computeLuminance (float r,
                        float g,
                        float b) {
    float luminance = r * 0.3f + g * 0.6f + b * 0.1f;

    loosen (luminance);
    return luminance;
}
```

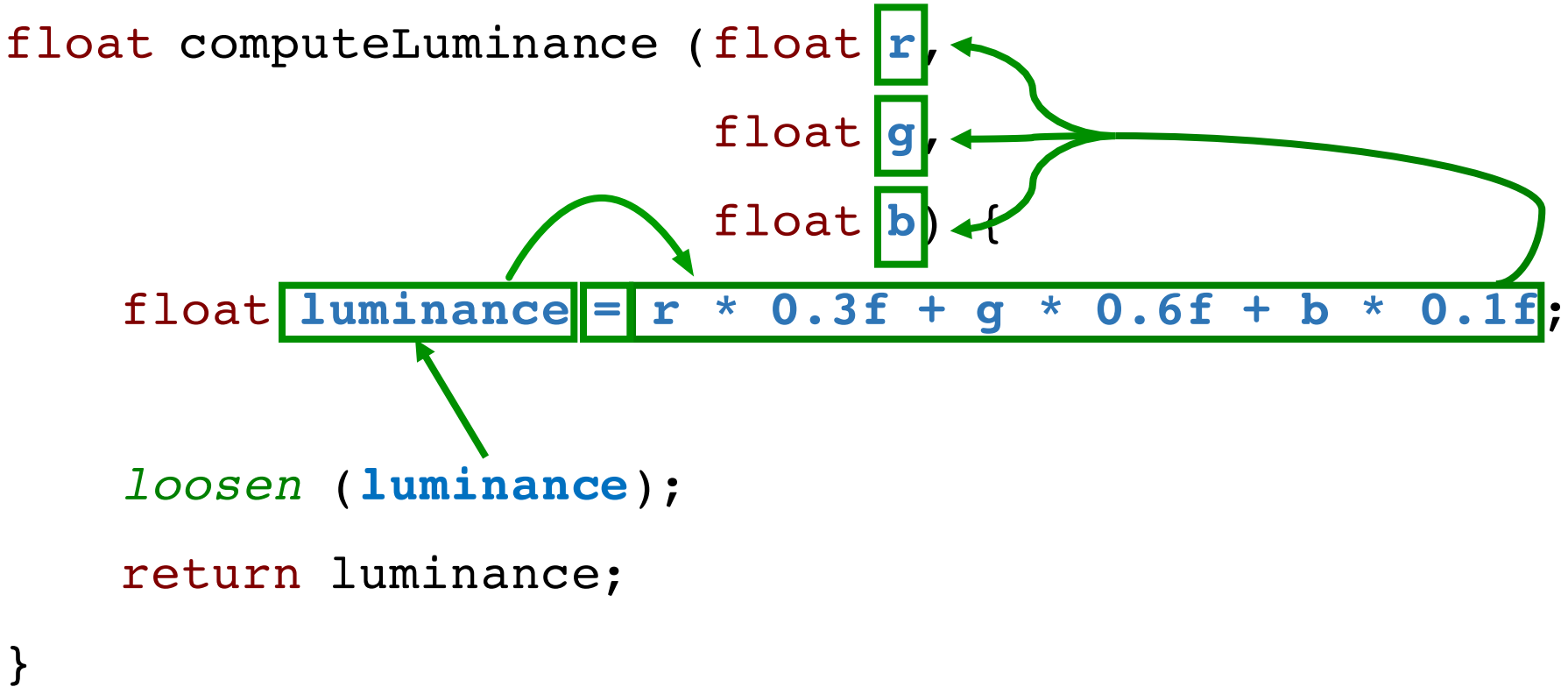


Safe and scalable approximation

```
float computeLuminance (float r,
                        float g,
                        float b) {
    float luminance = r * 0.3f + g * 0.6f + b * 0.1f;

    loosen (luminance);
    return luminance;
}
```



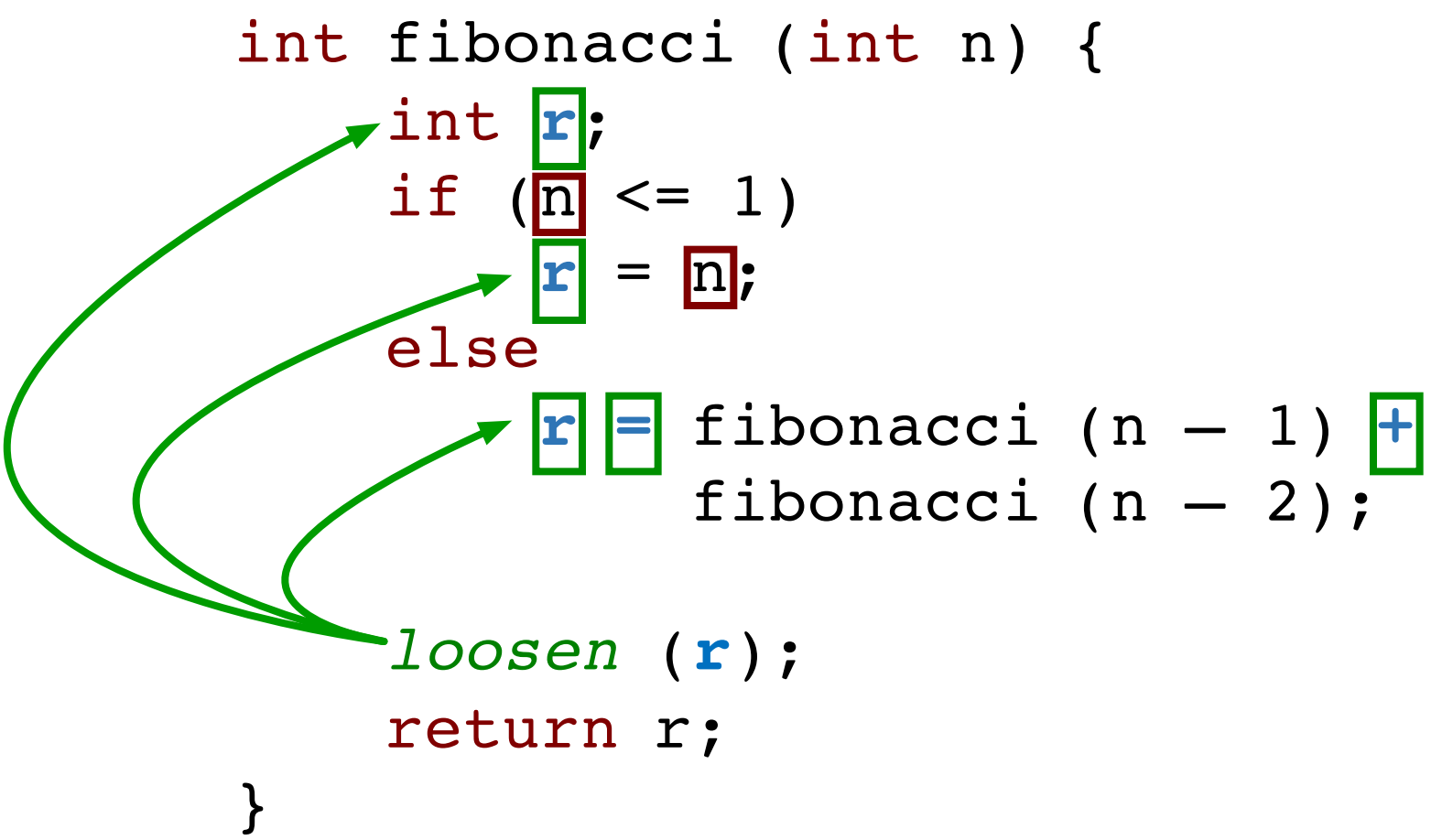
Safe and scalable approximation



Safe and scalable approximation

```
int fibonacci (int n) {
    int r;
    if (n <= 1)
        r = n;
    else
        r = fibonacci (n – 1) +
            fibonacci (n – 2);

    loosen (r);
    return r;
}
```

Safe and scalable approximation



Modularity

```
int p = 1;
for (int i = 0; i < a.length; i++) {
    p *= a[i];
}
for (int i = 0; i < b.length; i++) {
    p += b[i];
    loosen(p);
}
```

Reuse and library support

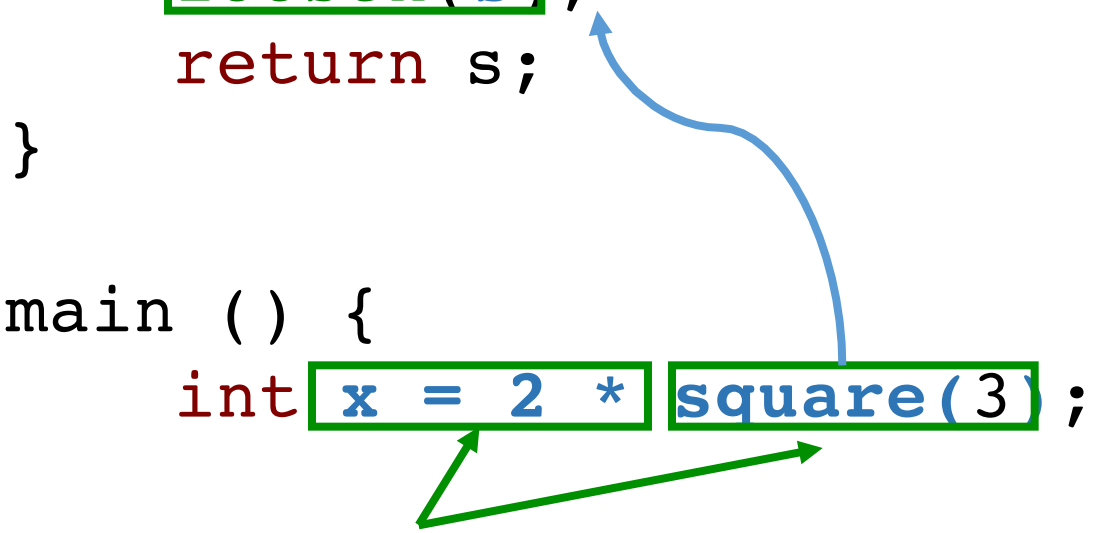
```
static int square(int a) {
    int s = a * a;
    loosen(s);
    return s;
}

main () {
    int x = 2 * square(3);


    System.out.println(x);
}
```

Reuse and library support

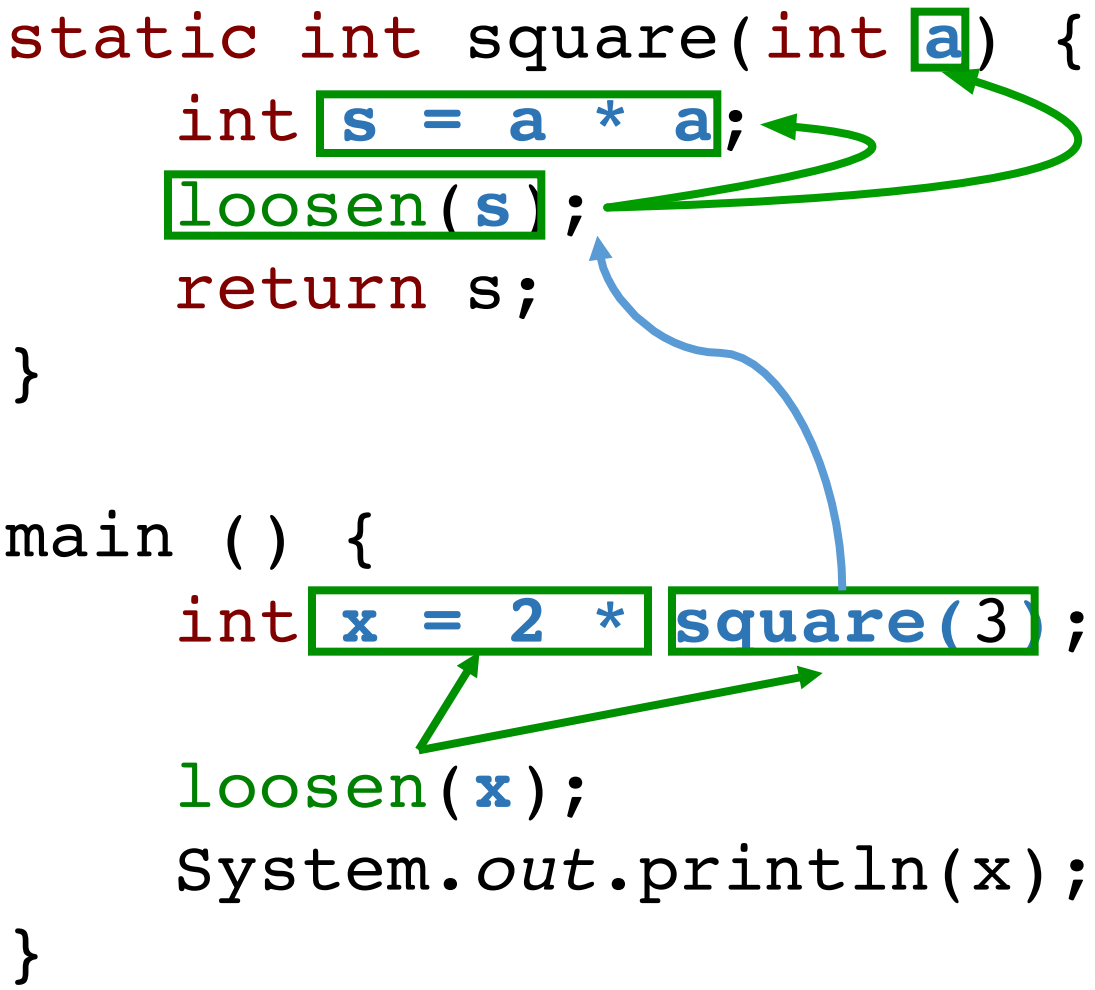
```
static int square(int a) {
    int s = a * a;
    loosen(s);
    return s;
}

main () {
    int x = 2 * square(3);

    loosen(x);
    System.out.println(x);
}
```

Reuse and library support

```
static int square(int a) {
    int s = a * a;
    loosen(s);
    return s;
}

main () {
    int x = 2 * square(3);

    loosen(x);
    System.out.println(x);
}
```



Reuse and library support

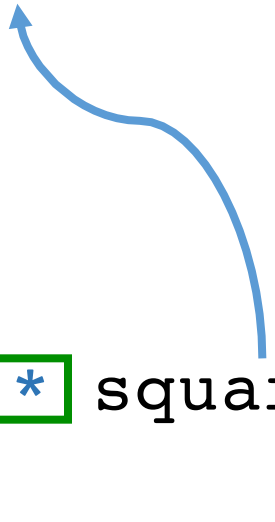


Reuse and library support

```
static int square(int a) {
    int s = a * a;

    return s;
}

main () {
    int x = 2 * square(3);

    loosen(x);
    System.out.println(x);
}
```

Reuse and library support

```
static int square(int a) {
    int s = a * a;

    return s;
}

main () {
    int x = 2 * square(3);

    loosen(x);
    System.out.println(x);
}
```

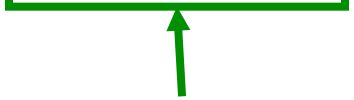


Reuse and library support

```
static int square(int a) {
    int s = a * a;

    return s;
}

main () {
    int x = 2 * square(3);

    loosen_invasive(x);
    System.out.println(x);
}
```

Reuse and library support

```
static int square(int a) {
    int s = a * a;

    return s;
}

main () {
    int x = 2 * square(3);

    loosen_invasive(x);
    System.out.println(x);
}
```

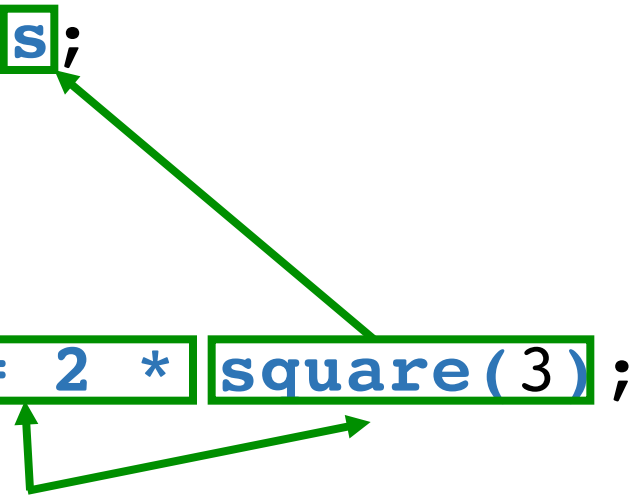


Reuse and library support

```
static int square(int a) {
    int s = a * a;

    return s;
}

main () {
    int x = 2 * square(3);

    loosen_invasive(x);
    System.out.println(x);
}
```

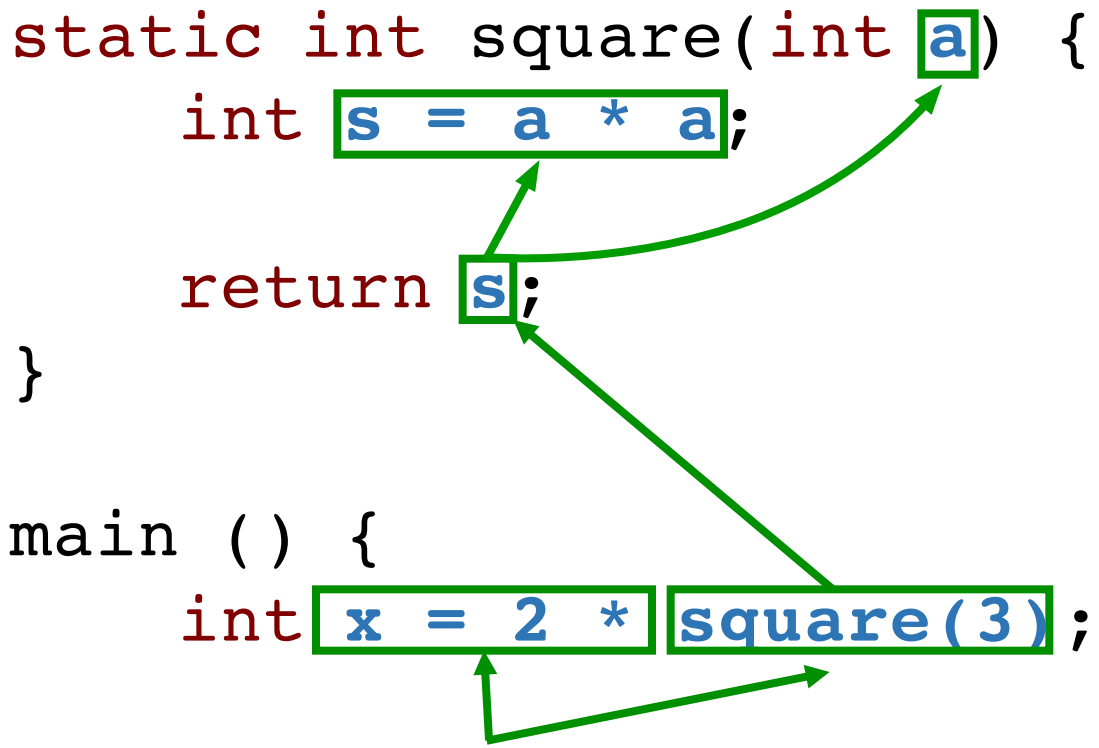
Reuse and library support

```
static int square(int a) {
    int s = a * a;

    return s;
}

main () {
    int x = 2 * square(3);

    loosen_invasive(x);
    System.out.println(x);
}
```

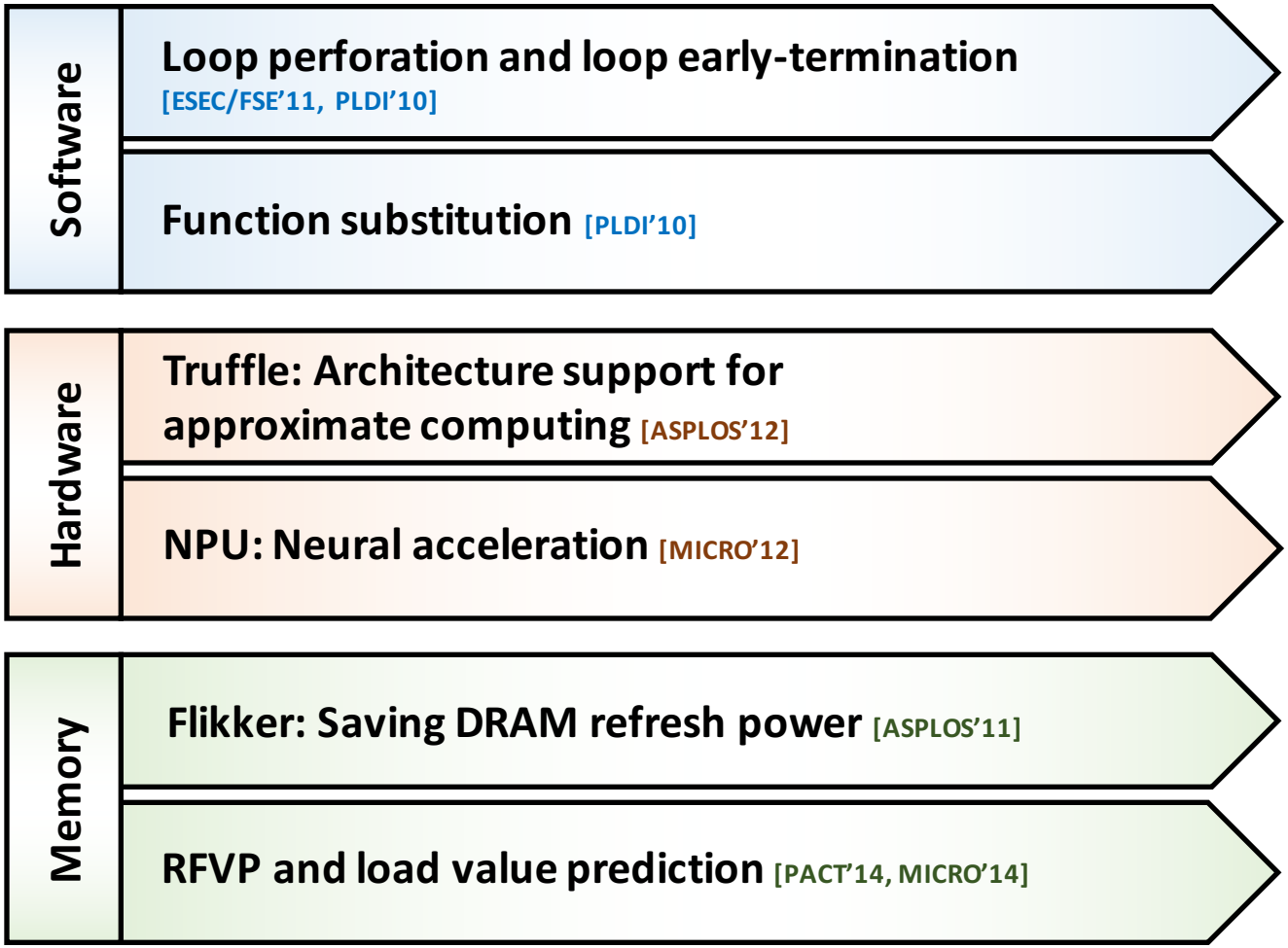


Reuse and library support

```
static int square(int a) {
    int s = a * a;

    return s;
}

main () {
    int x = 2 * square(3);

    loosen_invasive(x);
    System.out.println(x);
}
```



Generality


Software
Loop perforation and loop early-termination
[ESEC/FSE'11, PLDI'10]
Function substitution [PLDI'10]
Hardware
Truffle: Architecture support for approximate computing [ASPLOS'12]
NPU: Neural acceleration [MICRO'12]
Memory
Flikker: Saving DRAM refresh power [ASPLOS'11]
RFVP and load value prediction [PACT'14, MICRO'14]

Generality

```
begin_loose("PERFORATION", 0.10);
    for (int i = 0; i < n; i++) {
        …
    }
end_loose();
```

Generality

```
begin_loose("NPU");
    p = Math.sin(x) + Math.cos(y);
    q = 2 * Math.sin(x + y);
end_loose();
```

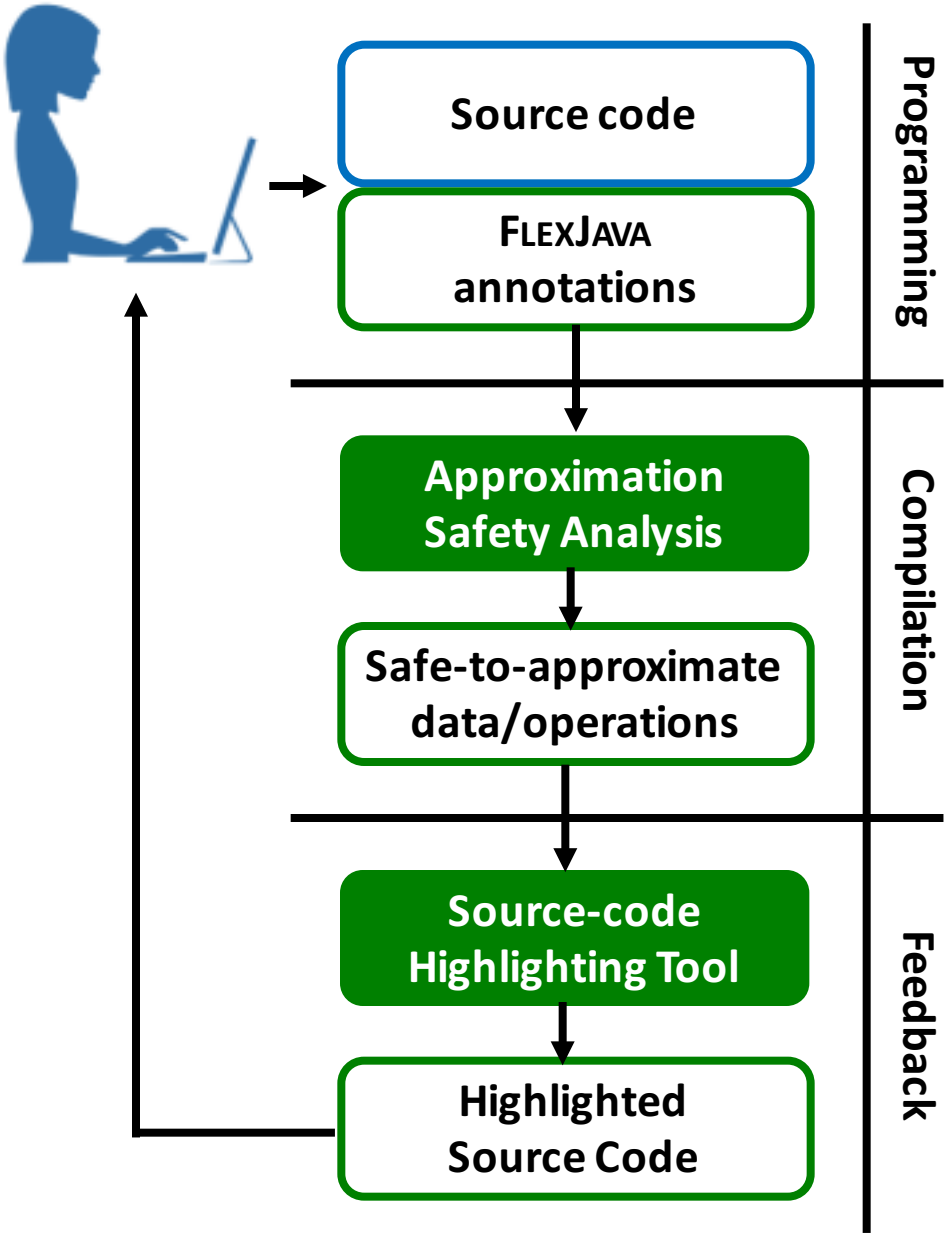
Approximation Safety Analysis

Find the maximal set of safe-to-approximate data/operations

1. For each annotation, loosen(**v**), find the set of **data** and **operations** that **affect v**
2. Merge all the sets
3. Exclude the data and operations that affect **control flow**
4. Exclude the data and operations that affect **address calculations**

The rest of the program data and operations are precise

Workflow

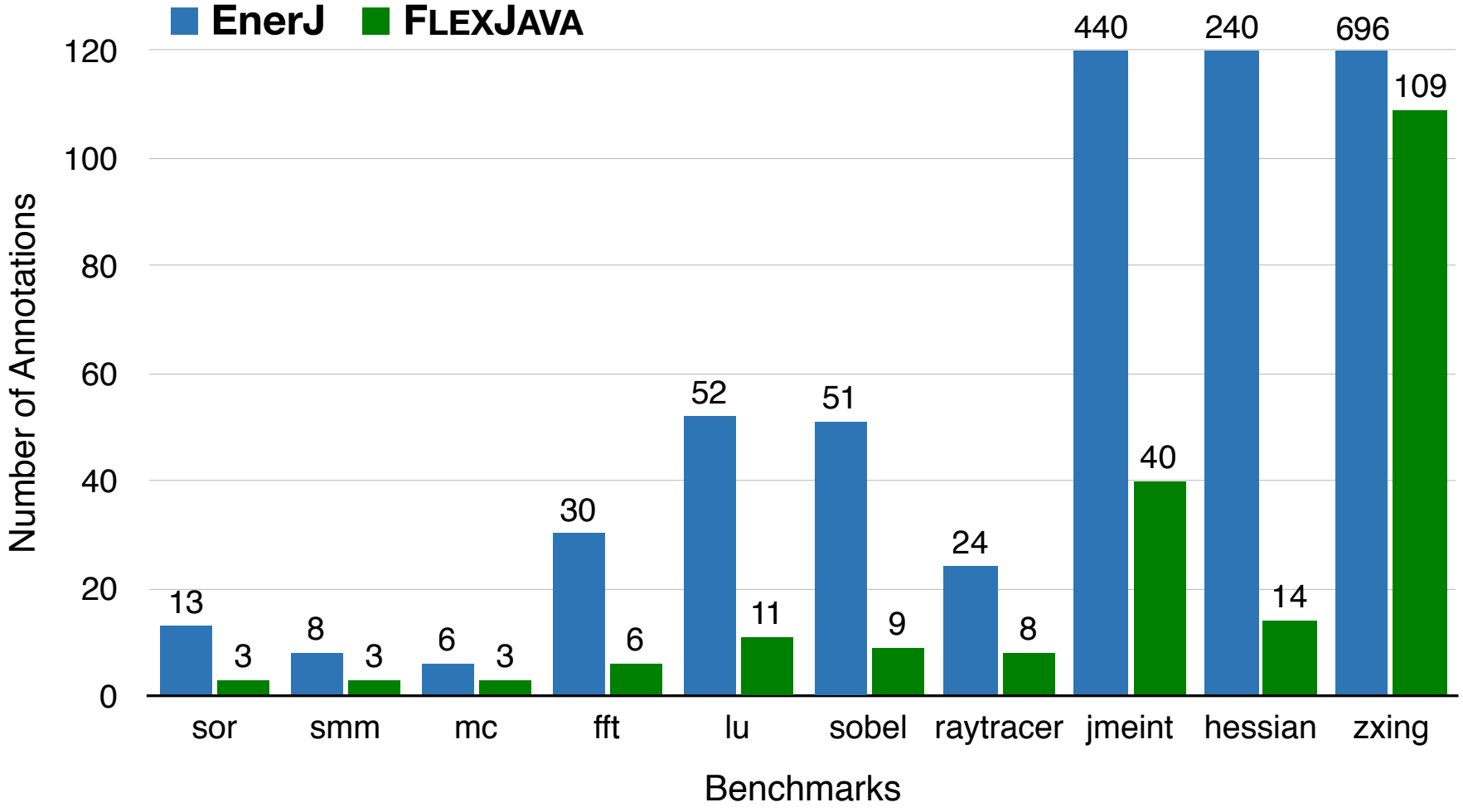

Source code
FLEXJAVA annotations
Approximation Safety Analysis
Safe-to-approximate data/operations
Source-code Highlighting Tool
Highlighted Source Code
Programming
Compilation
Feedback

Benchmark

sor **# lines: 36** SciMark2 benchmark	Successive Over-relaxation	**sobel** **# lines: 163** Image processing	Image edge detection
smm **# lines: 38** SciMark2 benchmark	Matrix-vector multiplication	**raytracer** **# lines: 174** Graphics	3D image renderer
mc **# lines: 59** SciMark2 benchmark	Mathematical approximation	**jmeint** **# lines: 5,962** 3D gaming	Triangle intersection kernel
fft **# lines: 168** SciMark2 benchmark	Signal processing	**hessian** **# lines: 10,174** Image processing	Interest point detection
lu **# lines: 283** SciMark2 benchmark	Matrix factorization	**zxing** **# lines: 26,171** Image processing	Bar code decoder

Lower is Better

Number of Annotations



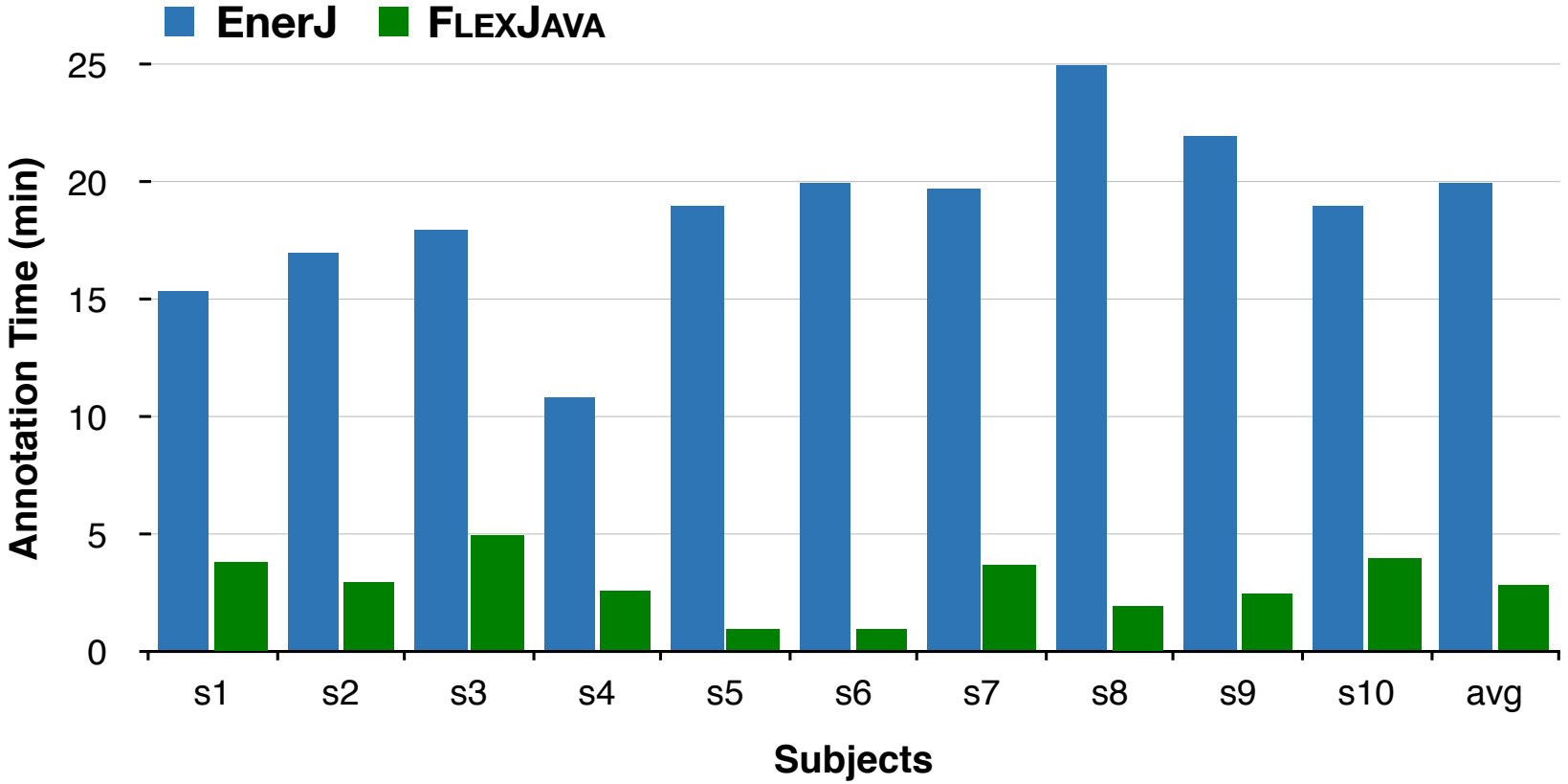
2× to **17×** reduction in the number of annotations

Lower is Better

User-study: Annotation Time

10 subjects (programmers)

Time for annotating **fft** using **EnerJ** and **FLEXJAVA**



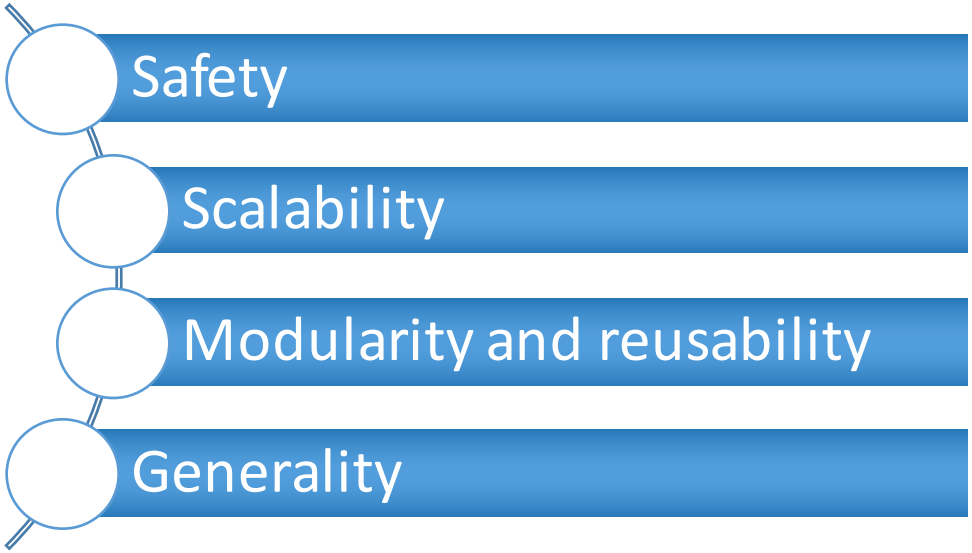
8× average reduction in annotation time

FLEXJAVA

Language-compiler **co-design** to

Automate approximate programming

Reduce programmer effort



Replication Package

http://act-lab.org/artifacts/flexjava/

1. Annotated benchmarks
2. Compiler with approximate safety analysis
3. Source code highlighting tool

Open source git repository:
https://bitbucket.org/act-lab/flexjava.code