EnerJ

Approximate Data Types for Safe and General Low-Power Computation

Adrian Sampson
Werner Dietl
Emily Fortuna
Danushen Gnanapragasam
Luis Ceze
Dan Grossman

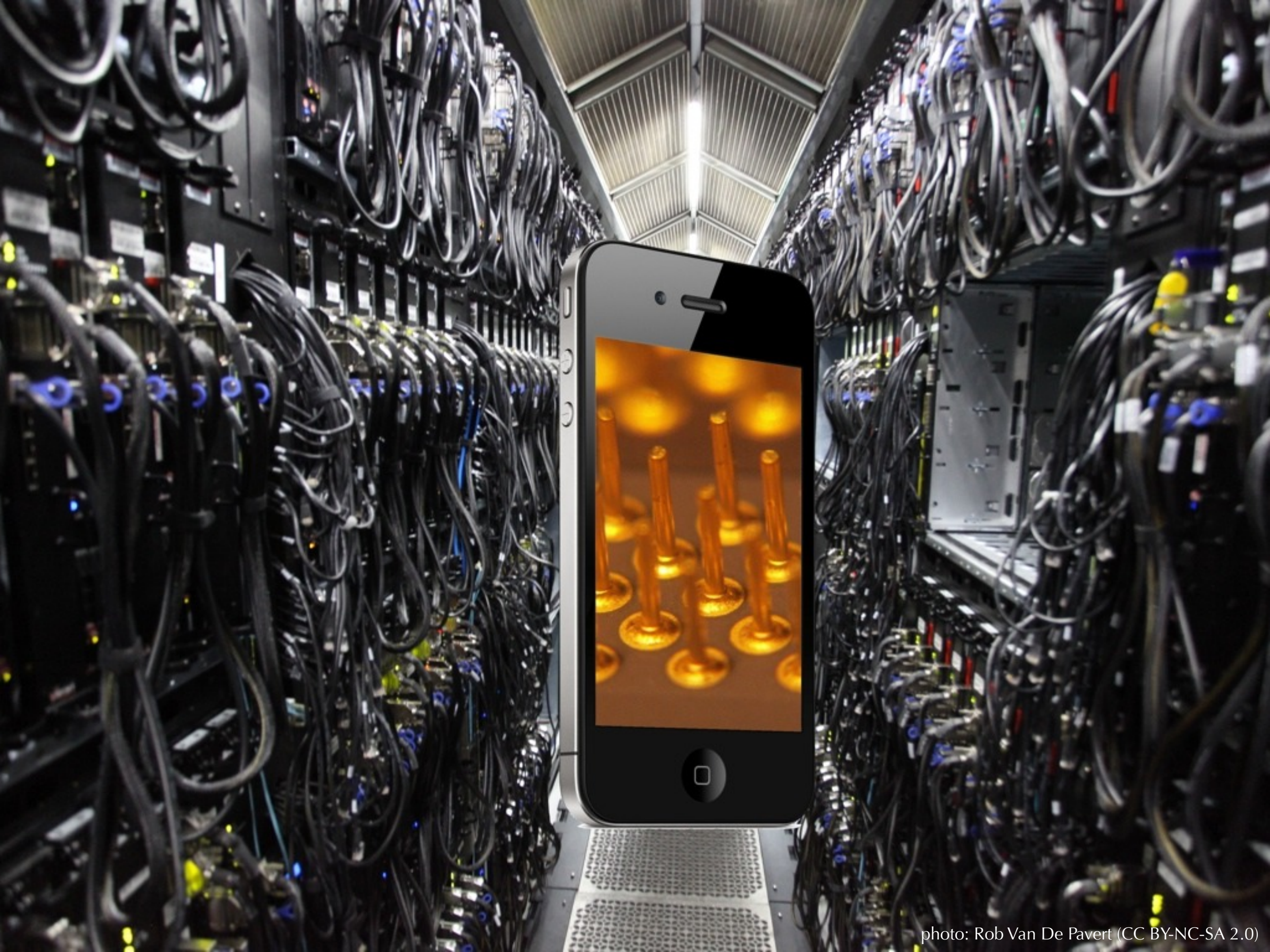
University of Washington

PLDI 2011

sampa

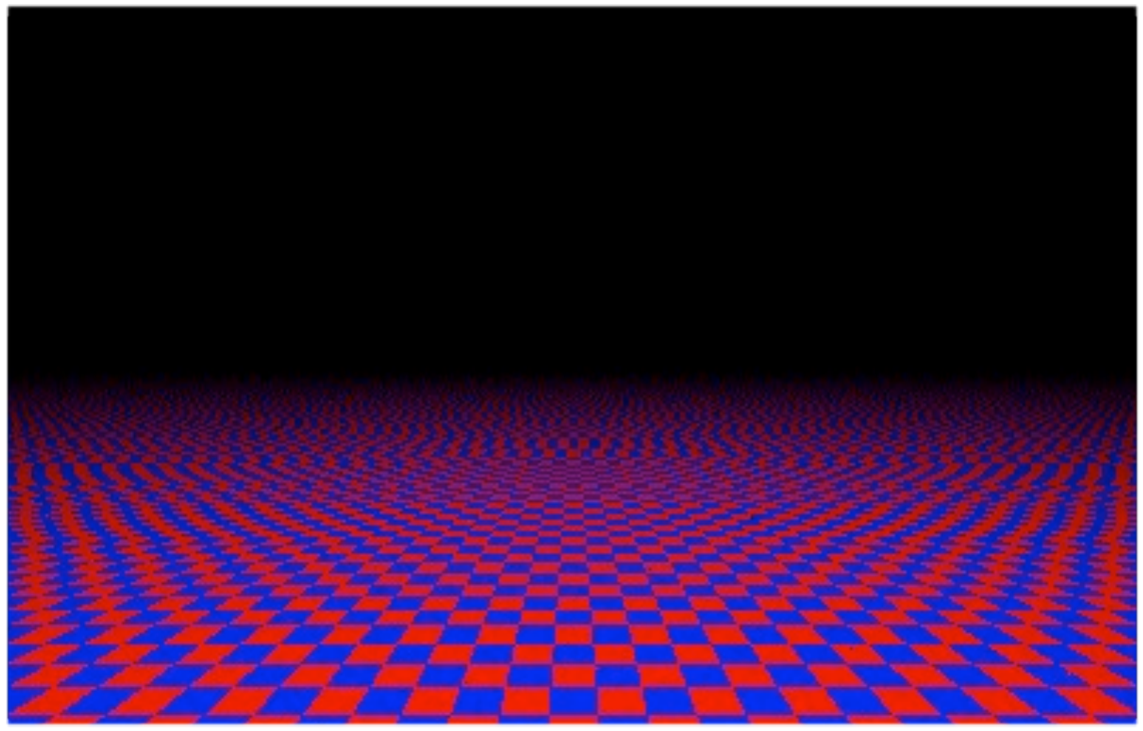
photo: Apple, Inc.

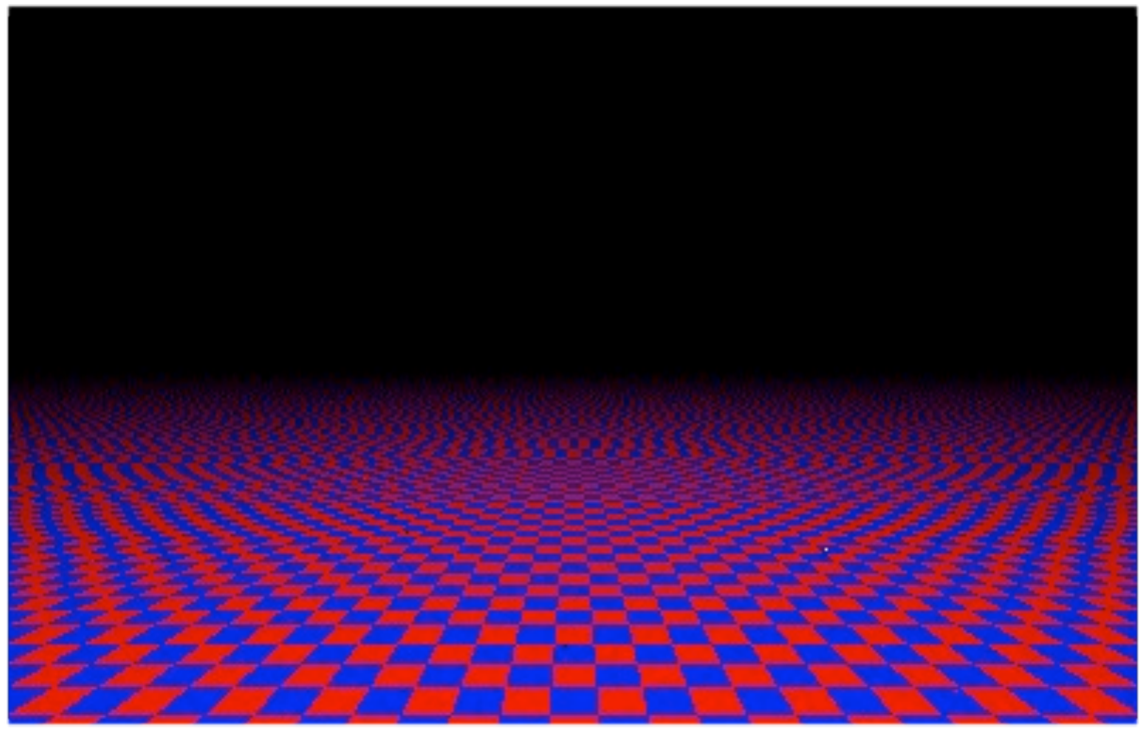
photo: Robert Scoble (CC BY 2.0)

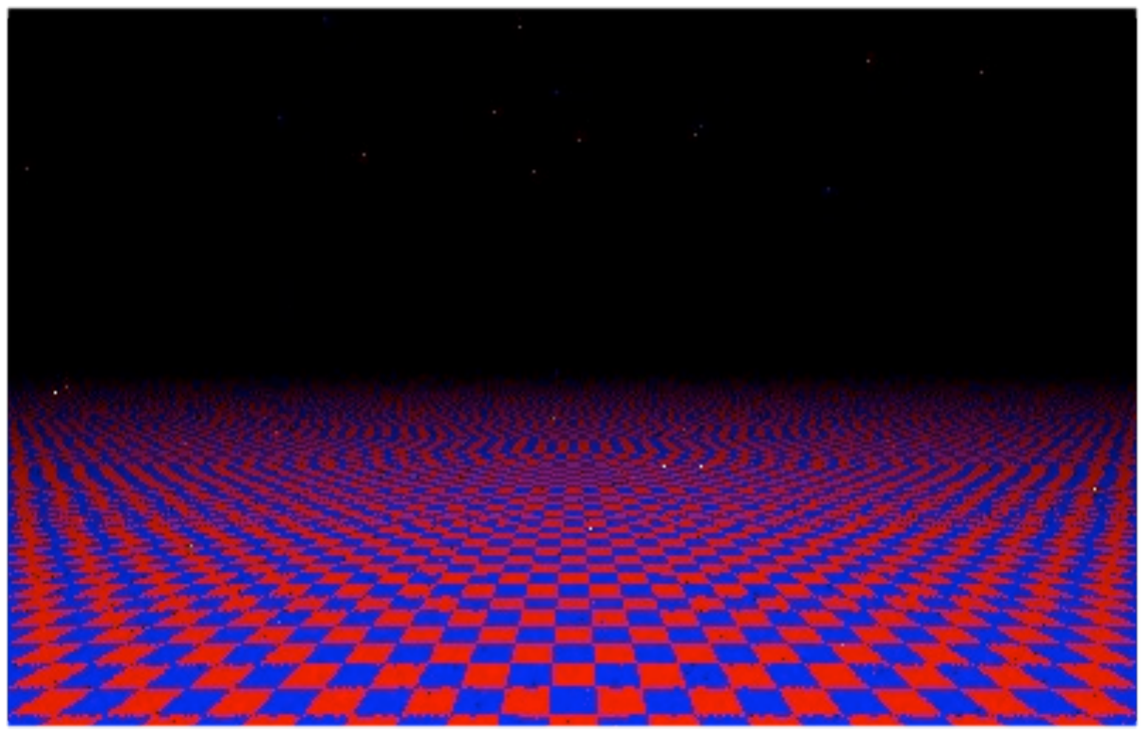


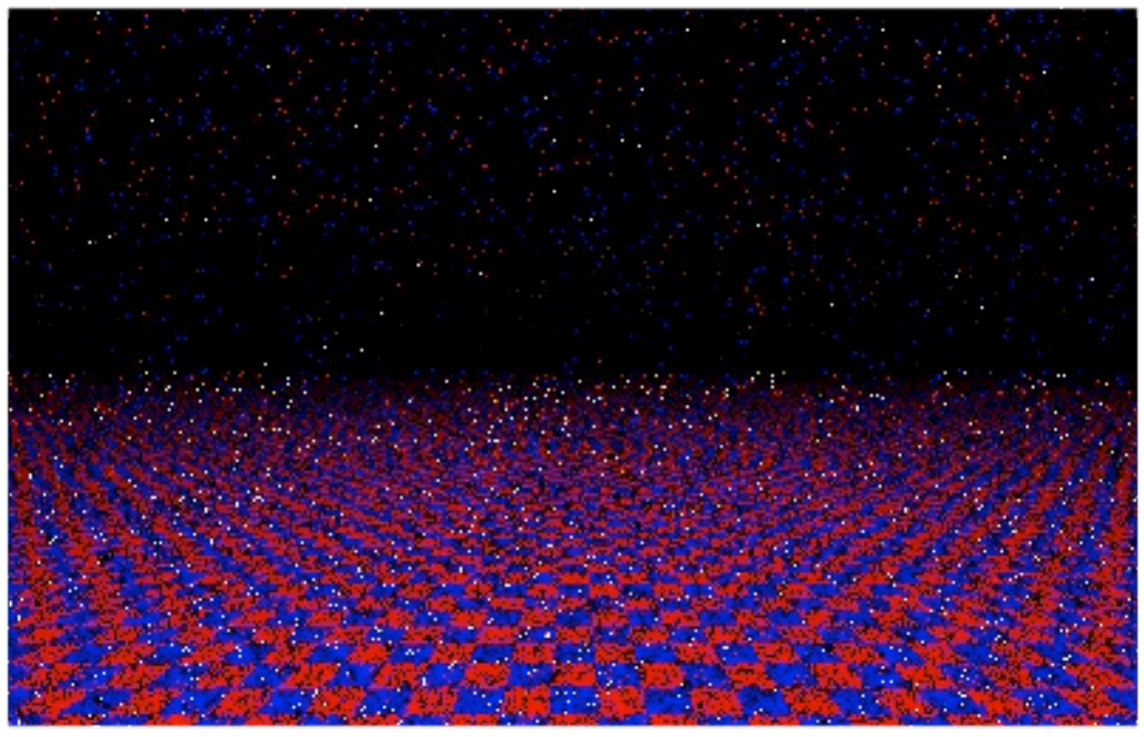
EnerJ:

Save **energy**
using programmer controls
over execution **correctness**.









Perfect correctness is not required

computer vision

machine learning

augmented reality

sensory data

games

scientific computing

information retrieval

physical simulation

[1] Anant Agarwal, Martin Rinard, Stelios Sidiroglou, Sasa Misailovic, and Henry Hoffmann. Using code perforation to improve performance, reduce energy consumption, and respond to failures. Technical report, MIT, 2009.

[2] B.E.S. Akgul, L.N. Chakrapani, P. Korkmaz, and K.V. Palem. Probabilistic CMOS technology: A survey and future directions. In IFIP Intl. Conference on VLSI, 2006.

[3] M. de Kruijf and K. Sankaralingam. Exploring the synergy of emerging workloads and silicon reliability trends. In SELSE, 2009.

[4] Larkhoon Leem, Hyungmin Cho, Jason Bau, Quinn A. Jacobson, and Subhasish Mitra. ERSA: Error resilient system architecture for probabilistic applications. In DATE, 2010.

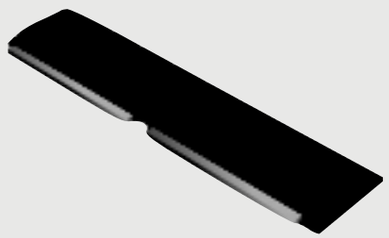
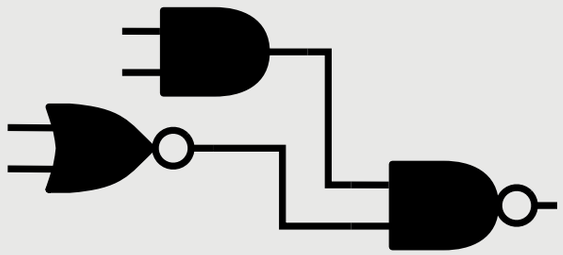
[5] Xuanhua Li and Donald Yeung. Exploiting soft computing for increased fault tolerance. In ASGI, 2006.

[6] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. Flicker: Saving refresh-power in mobile devices through critical data partitioning. Technical Report MSR-TR-2009-138, Microsoft Research, 2009.

[7] Sriram Narayanan, John Sartori, Rakesh Kumar, and Douglas L. Jones. Scalable stochastic processors. In DATE, 2010.

[8] Vicky Wong and Mark Horowitz. Soft error resilience of probabilistic inference applications. In SELSE, 2006.

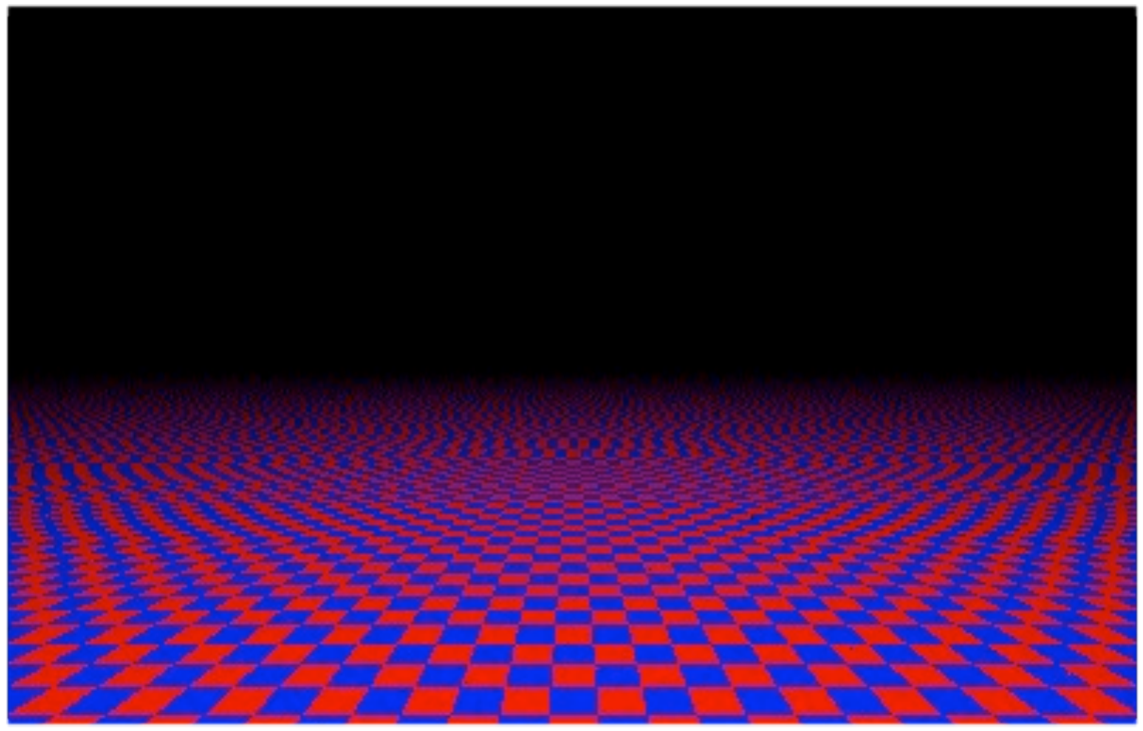
Kinds of imprecision

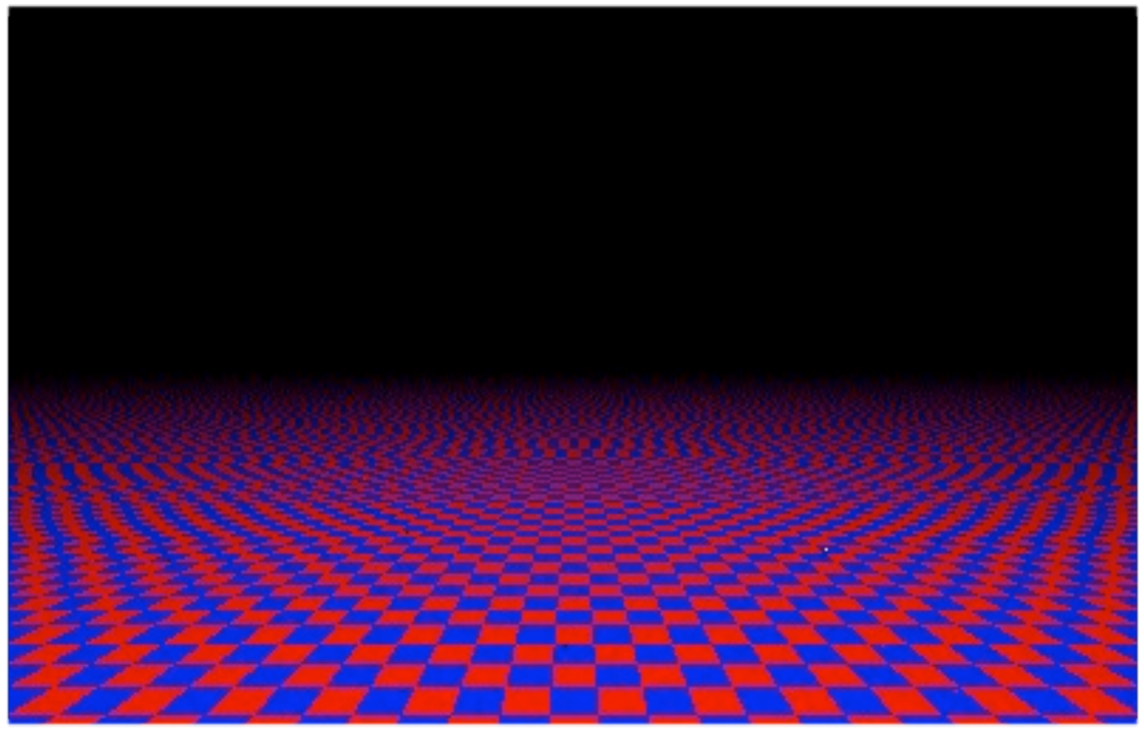
EnerJ		
Storage	Logic	Algorithms
		λ
Flikker ASPLOS 2011	Relax ISCA 2010	Green PLDI 2010

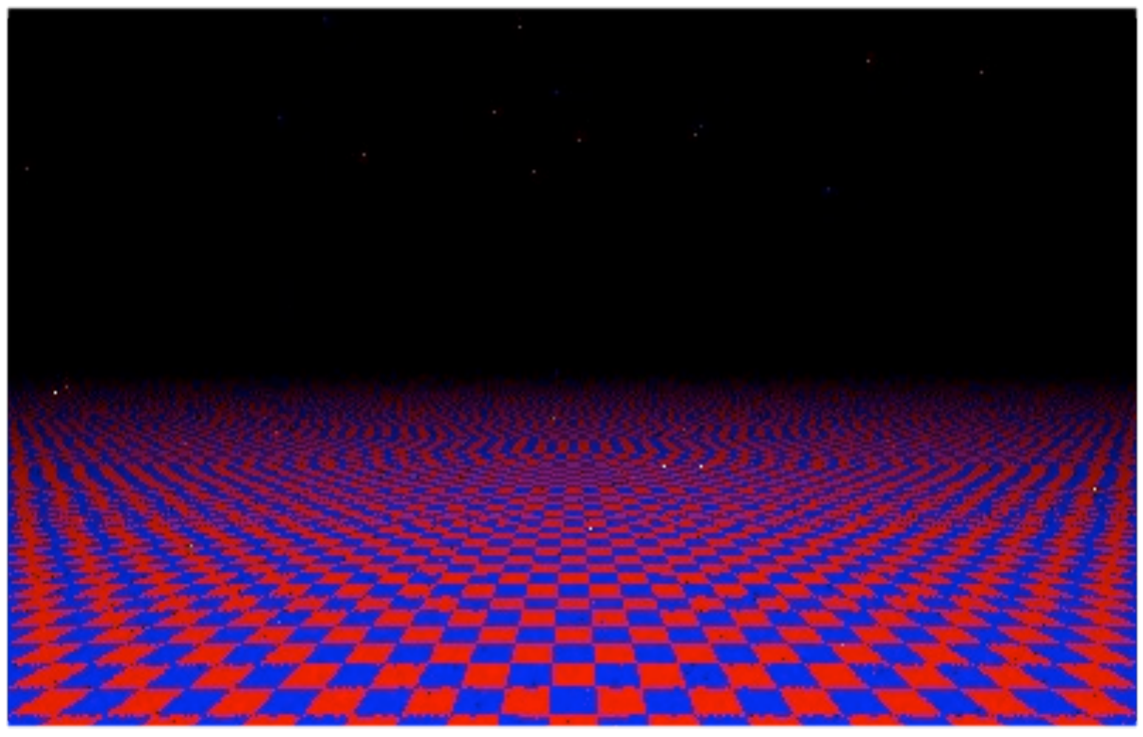
Generality

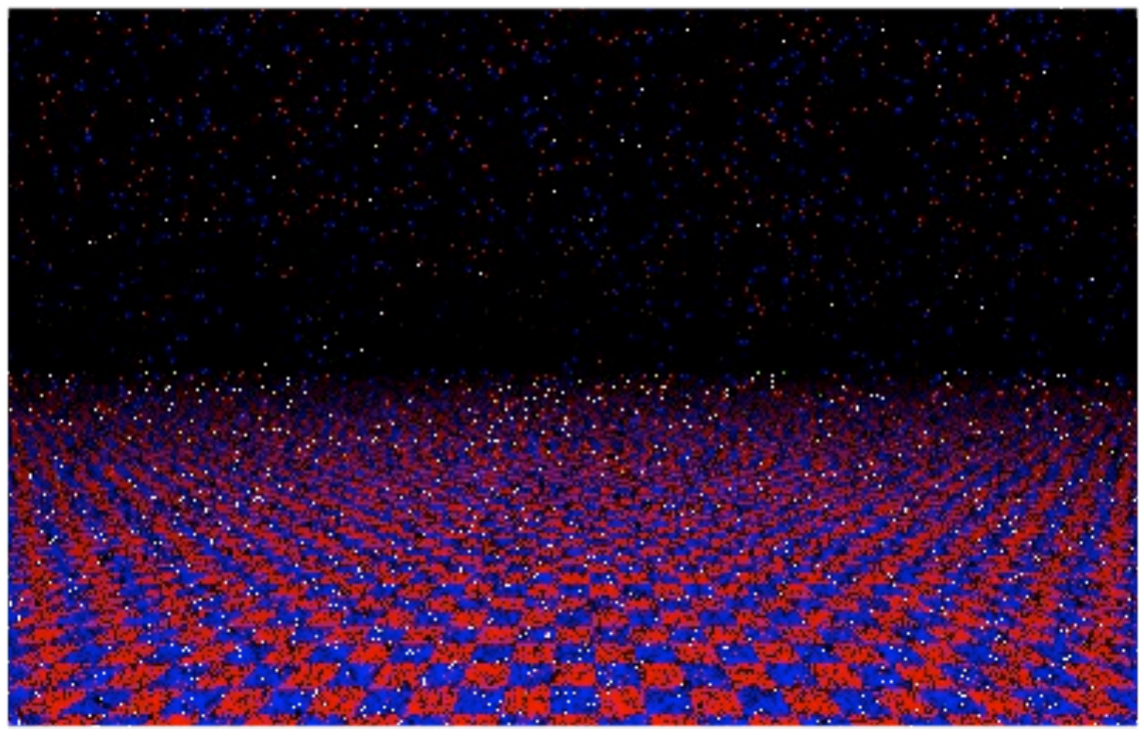
A range of approximation strategies supported with a single abstraction.

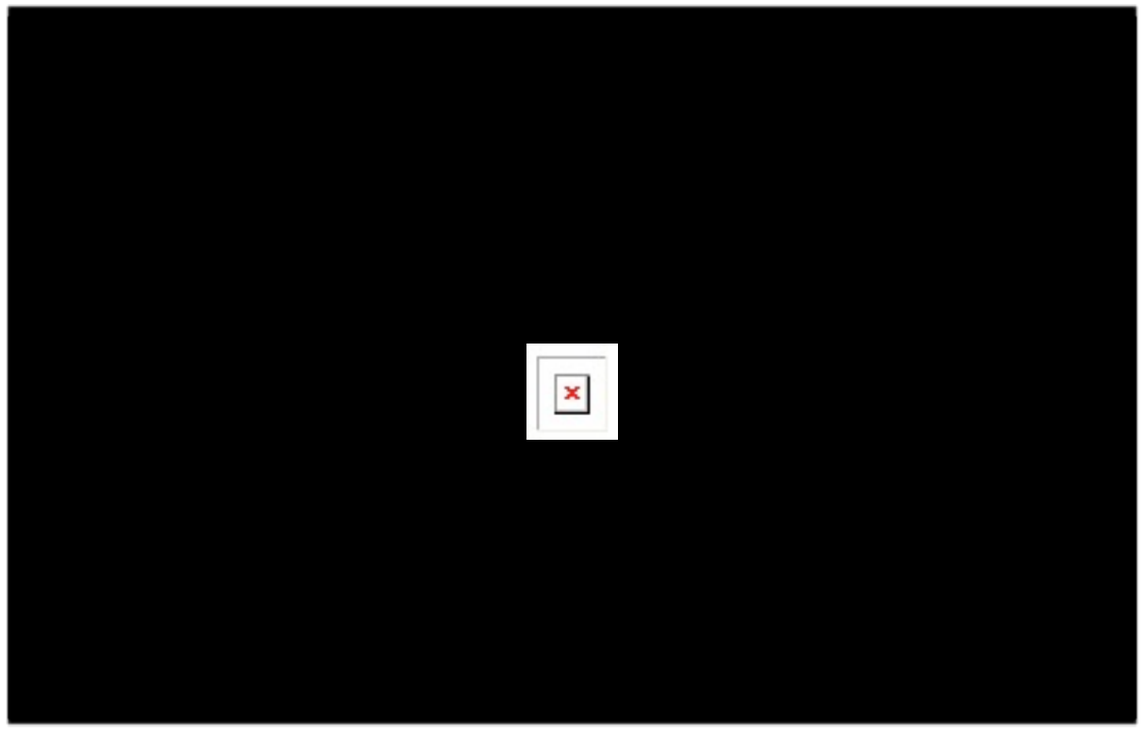
error-sensitive	**error-resilient**
references	pixel data
jump targets	audio samples
JPEG header	neuron weights
	video frames
Critical	**Non-Critical**

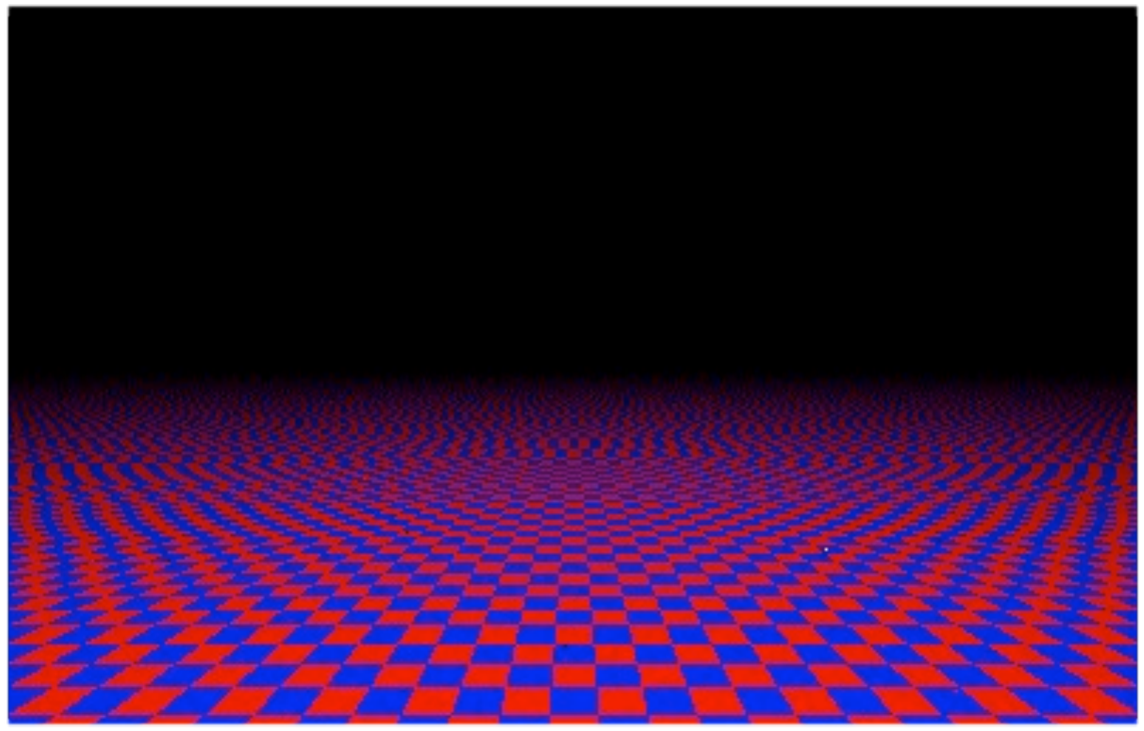


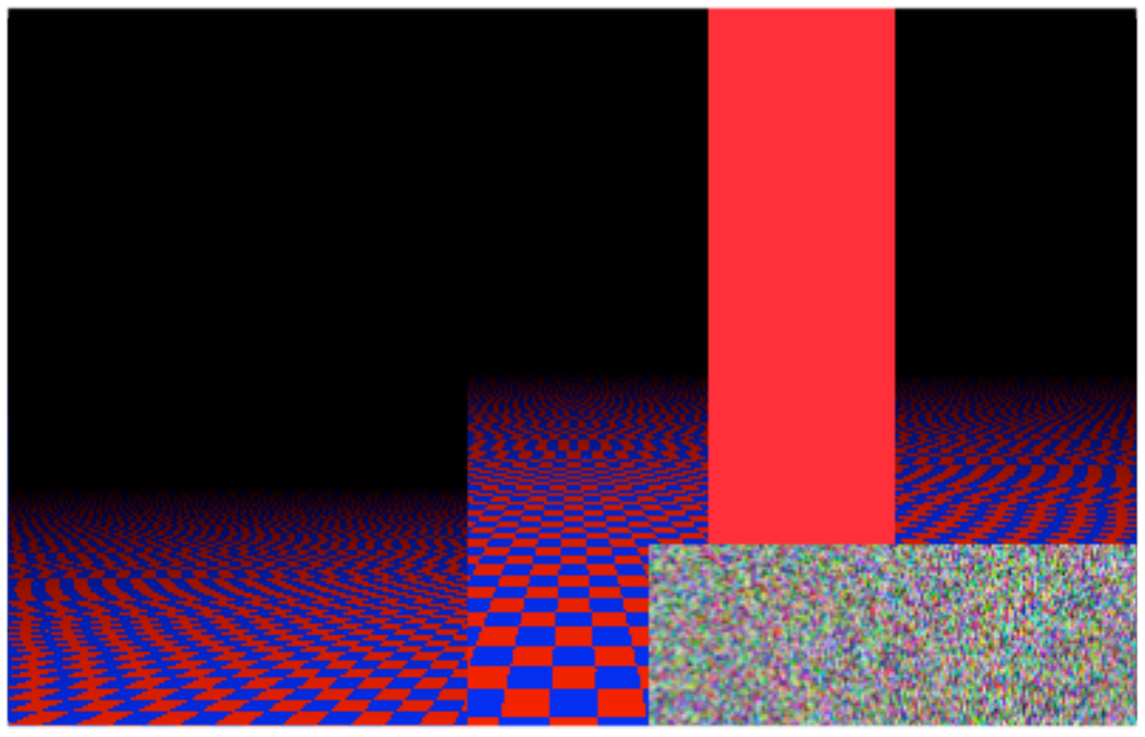


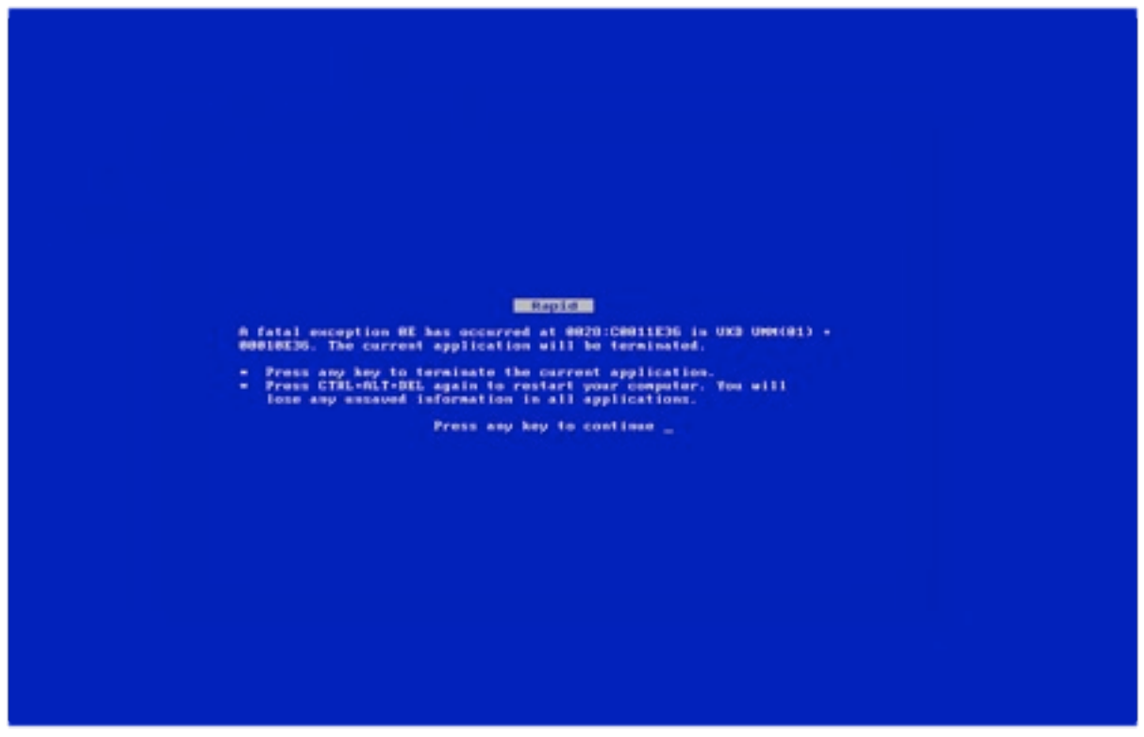


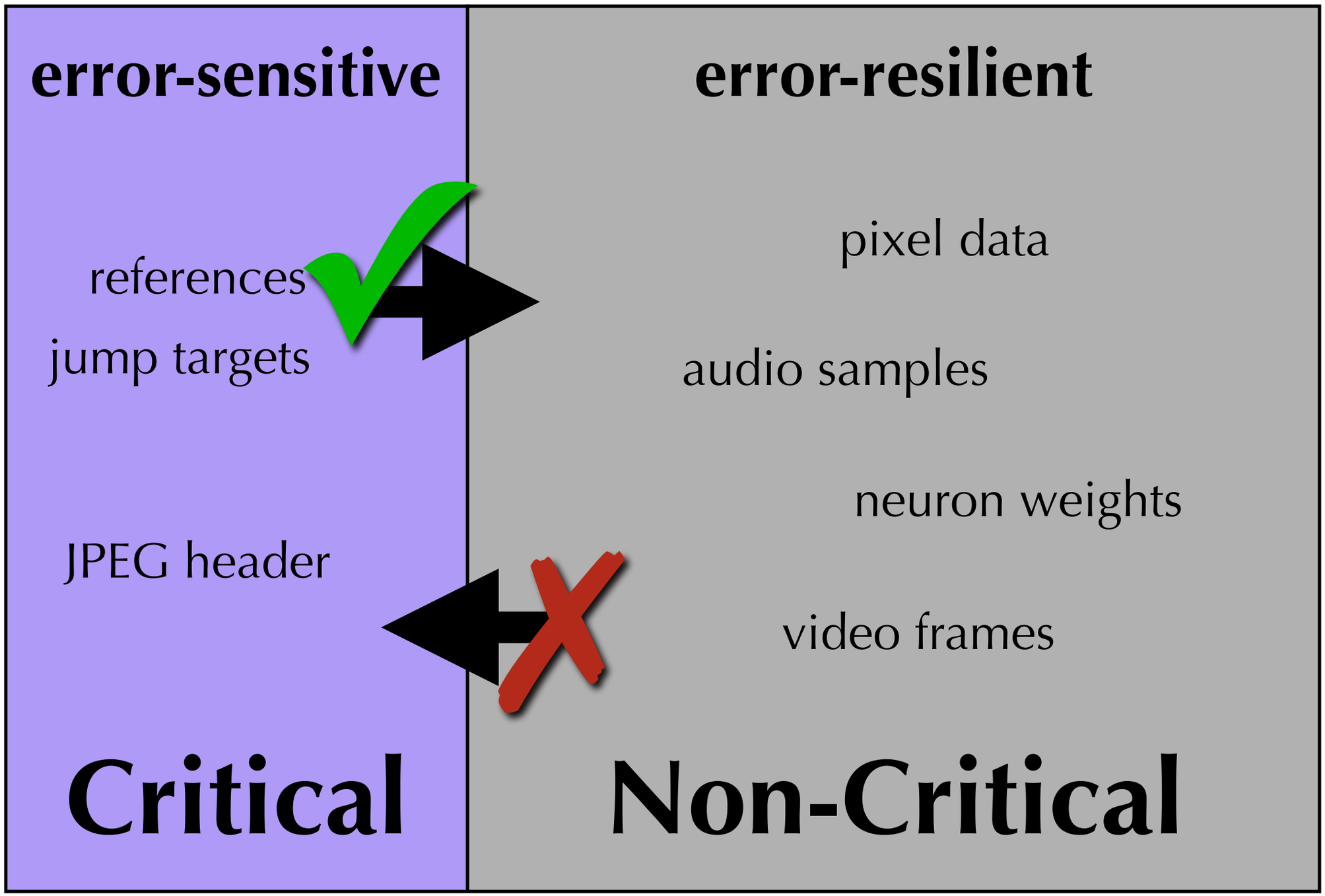









Kaplá
A fatal exception 0E has occurred at 0028:C0011E36 in VXD VMM(01) +
0001BE36. The current application will be terminated.
• Press any key to terminate the current application.
• Press CTRL+ALT+DEL again to restart your computer. You will
lose any unsaved information in all applications.
Press any key to continue _


error-sensitive
error-resilient
references
jump targets
pixel data
audio samples
neuron weights
JPEG header
video frames
Critical
Non-Critical

Generality

A range of approximation strategies supported with a single abstraction.

Safety

Separate *critical* and *non-critical* program components.

Java language extension
using type annotations

Proposed approximate hardware

Potential energy savings
in existing Java programs

Java language extension using type annotations

Proposed approximate hardware

Potential energy savings in existing Java programs

Java language extension using type annotations

- Type qualifiers: `@Approx` & `@Precise`
- Endorsement
- Operator overloading
- Prevention of implicit flows
- Objects: qualifier polymorphism

Type qualifiers

```
@Approx int a = ...;

@Precise int p = ...;

p = a;

a = p;
```

Endorsement: escape hatch

```
@Approx int a = expensiveCalc();

@Precise int p;

p = endorse( a );

quickChecksum(p);

output(p);
```

Logic approximation: overloading

```
@Approx int a = ...;
@Precise int p = ...;
```

```
p + p;
```

+ : @Precise int, @Precise int → @Precise int

```
p + a;
```

```
a + a;
```

+ : @Approx int, @Approx int → @Approx int

Control flow

```
@Approx int a = ...;

@Precise int p = ...;

if (a == 10) {
    p = 2;
}
```

Control flow

```
@Approx int a = ...;

@Precise int p = ...;

if (endorse(a == 10)) {
    p = 2;
}
```

Objects

```
class FloatSet {
    float[] nums = ...;
    float mean() {
        calculate mean
    }
}
new @Approx FloatSet()
new @Precise FloatSet()
```

Objects

```
class FloatSet {
    @Context float[] nums = ...;
    float mean() {
        calculate mean
    }
}
```

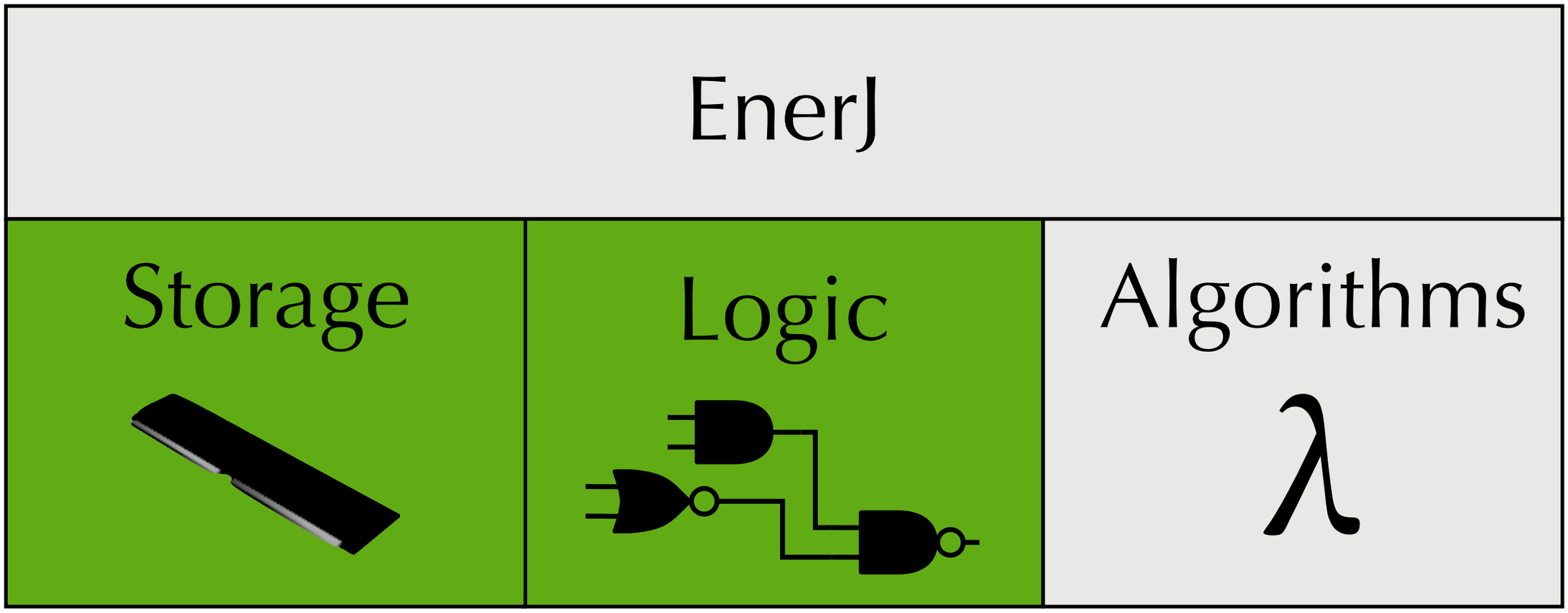
```
class FloatSet {
    @Context float[] nums = ...;
    float mean() {
        calculate mean
    }
    @Approx float mean_APPROX() {
        take mean of first ½
    }
}

@Approx FloatSet someSet = ...;
someSet.mean();
```

Java language extension
using type annotations

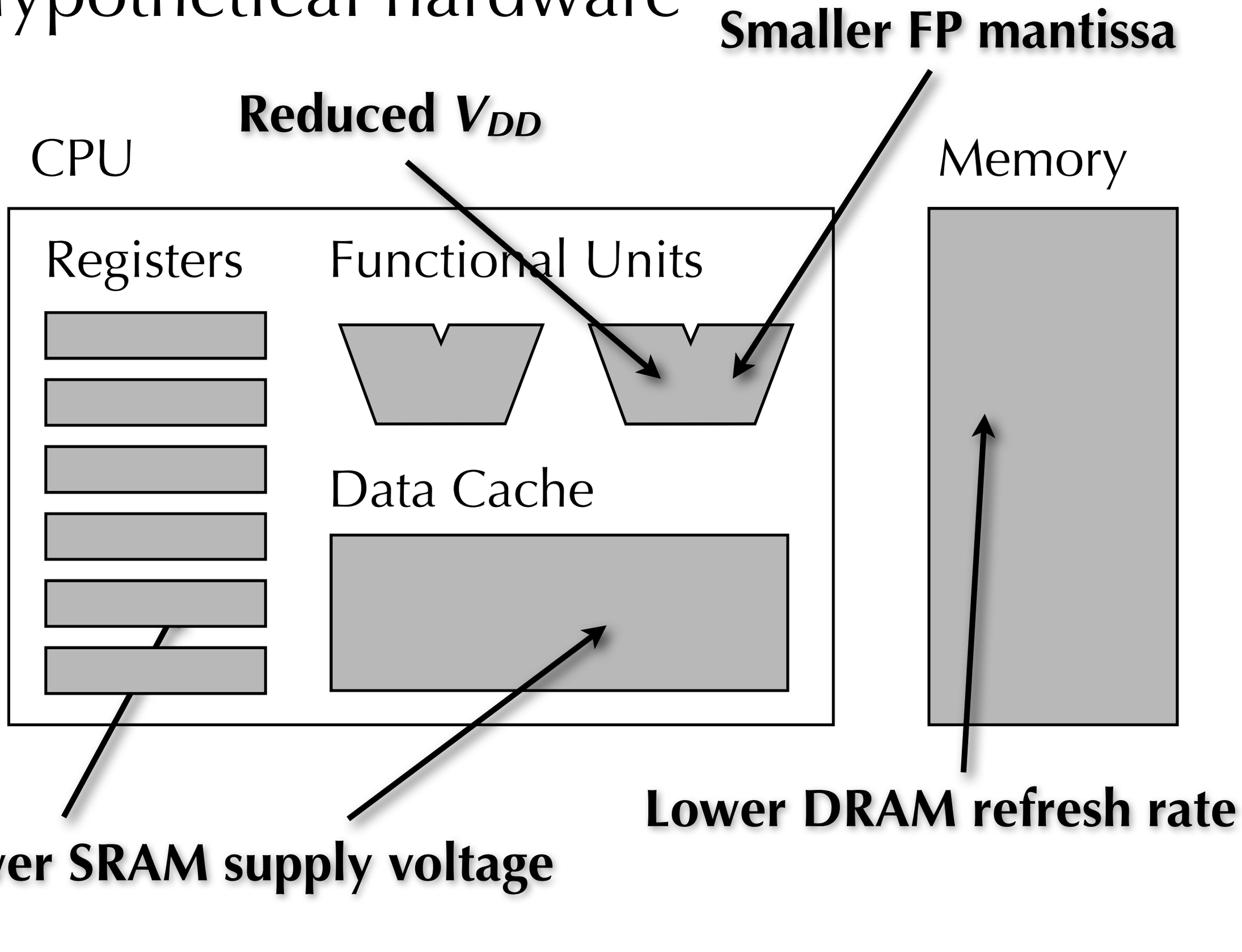
Proposed approximate hardware

Potential energy savings
in existing Java programs

Hypothetical hardware



Hypothetical hardware

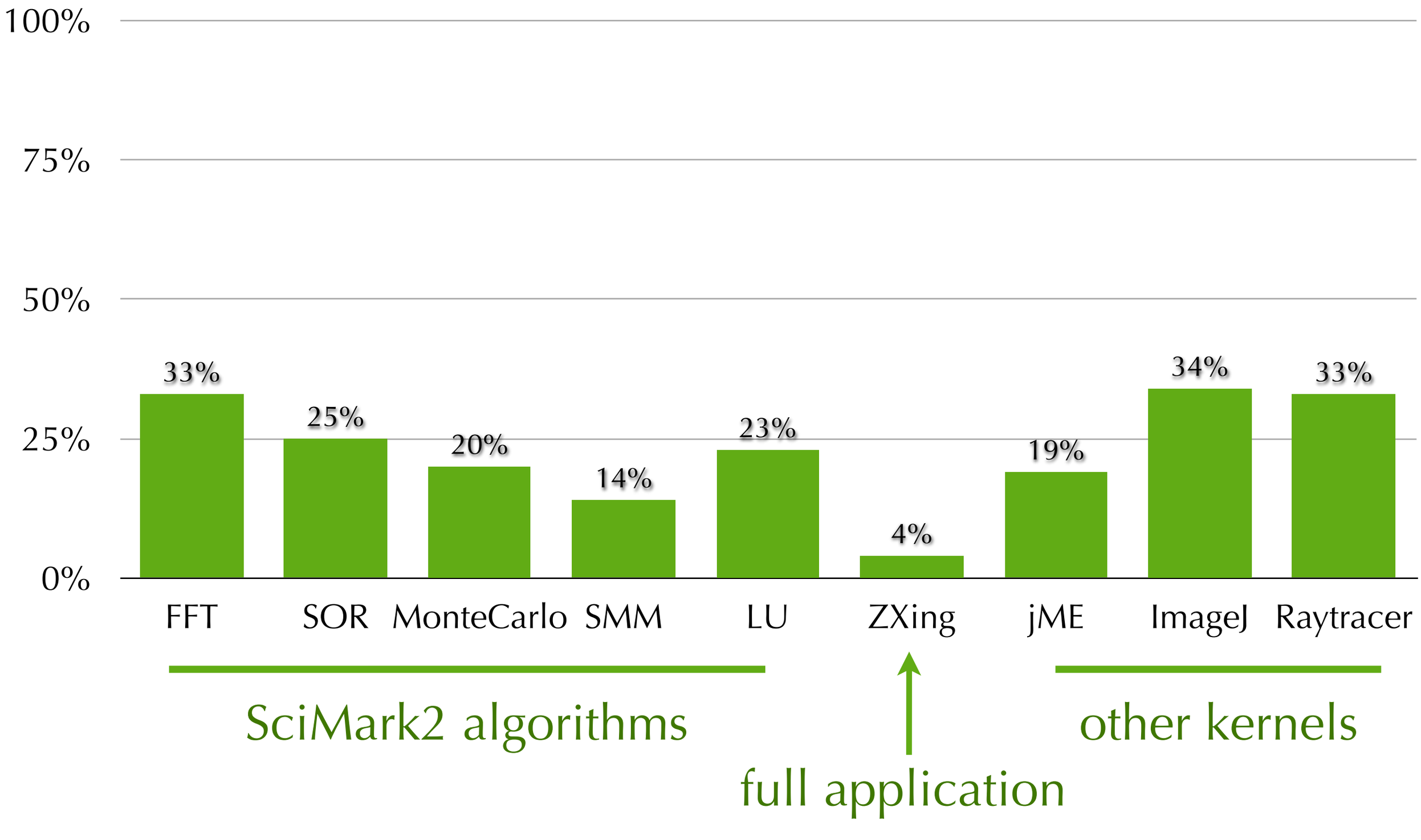


Java language extension
using type annotations

Proposed approximate hardware

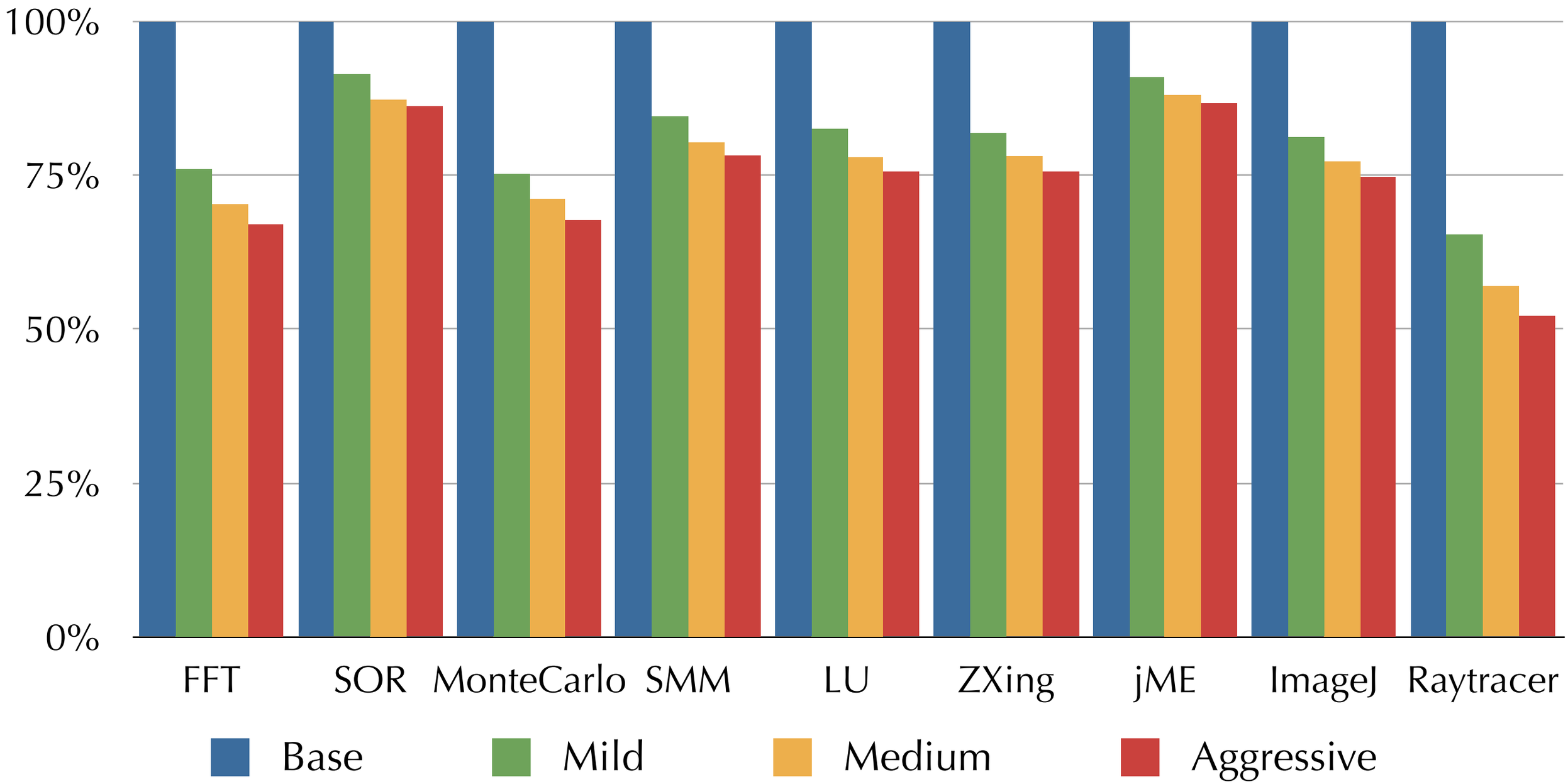
Potential energy savings
in existing Java programs

Annotated declarations



Annotations are sparse & straightforward to insert

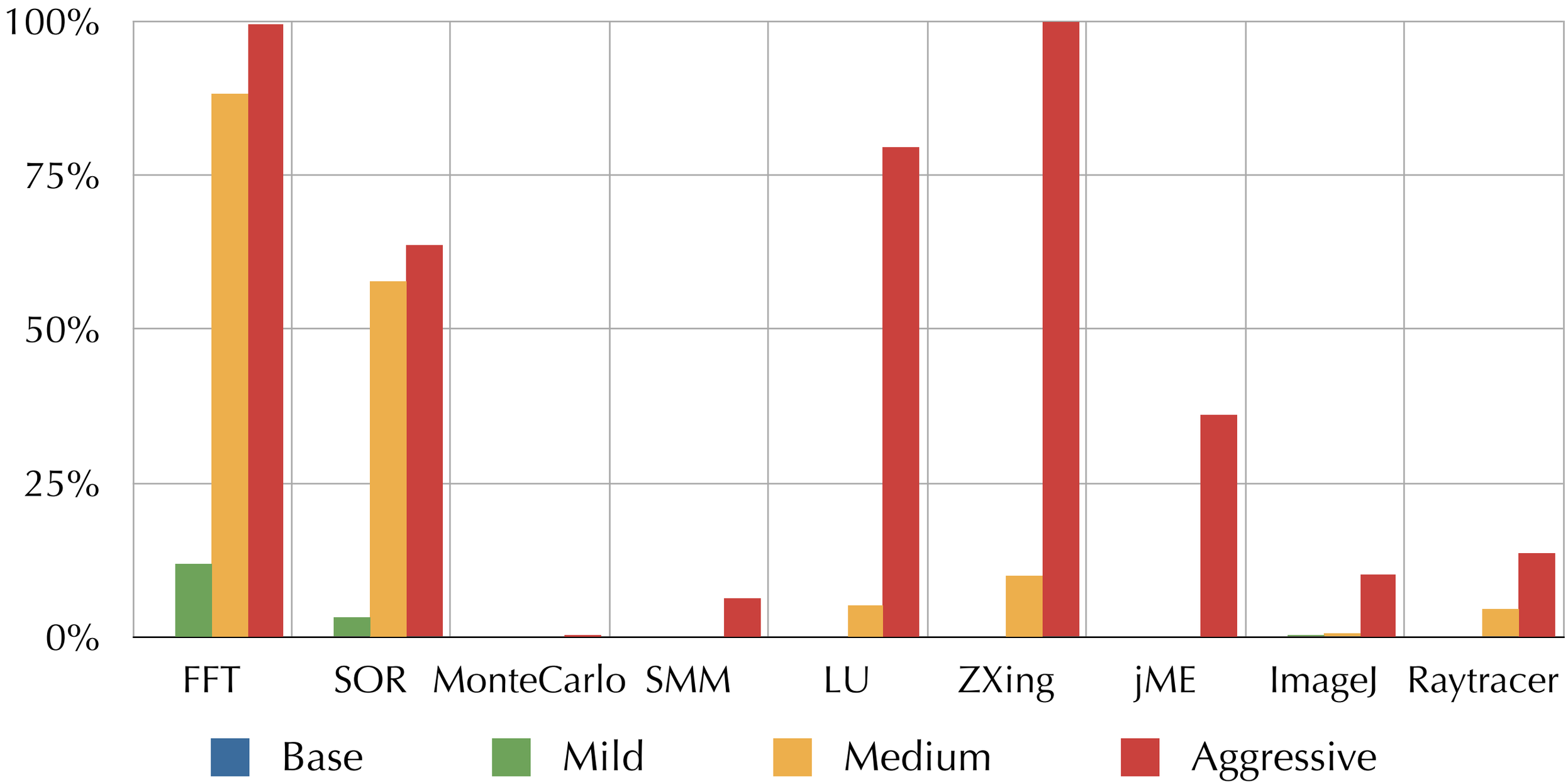
Total energy used



Saved 10%—50% of total execution energy

(in simulation)

Quality-of-service tradeoff: output error



"Mild" configuration is a good fit for all

Some applications can tolerate more approximation

Also in the paper:

Formal semantics

Noninterference claim

Object layout

Hardware model

Quality-of-service metrics

EnerJ:

Save **energy** using programmer controls over execution **correctness**.