Luis Ceze

areas: computer architecture, OS, programming languages

sampa ***Safe MultiProcessing Architectures at the University of Washington***

Safe and General Energy-Aware Programming with Disciplined Approximation

Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam

Luis Ceze, Dan Grossman

University of Washington

Motivation

Energy is already a first class concern in systems design

- Lower power bills
- Longer battery life (or smaller battery ;)

Better energy efficiency likely *necessary* for continued scaling (Dark Silicon)

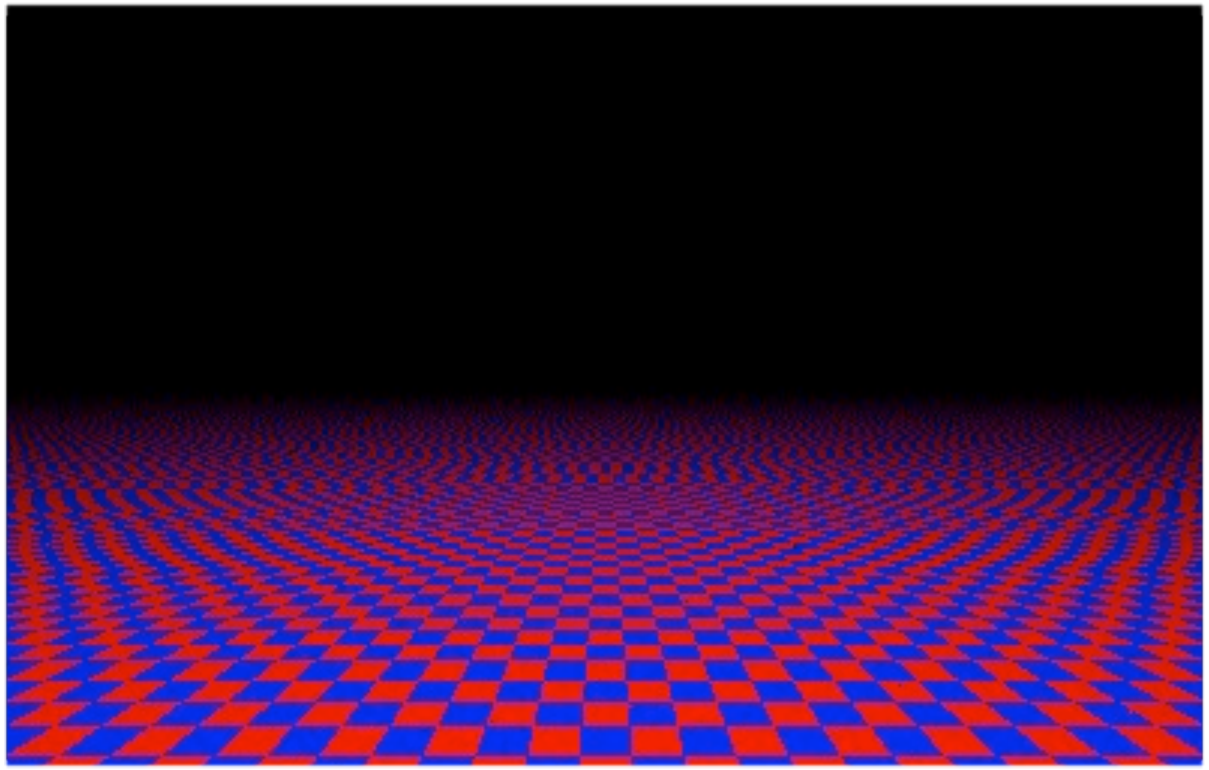
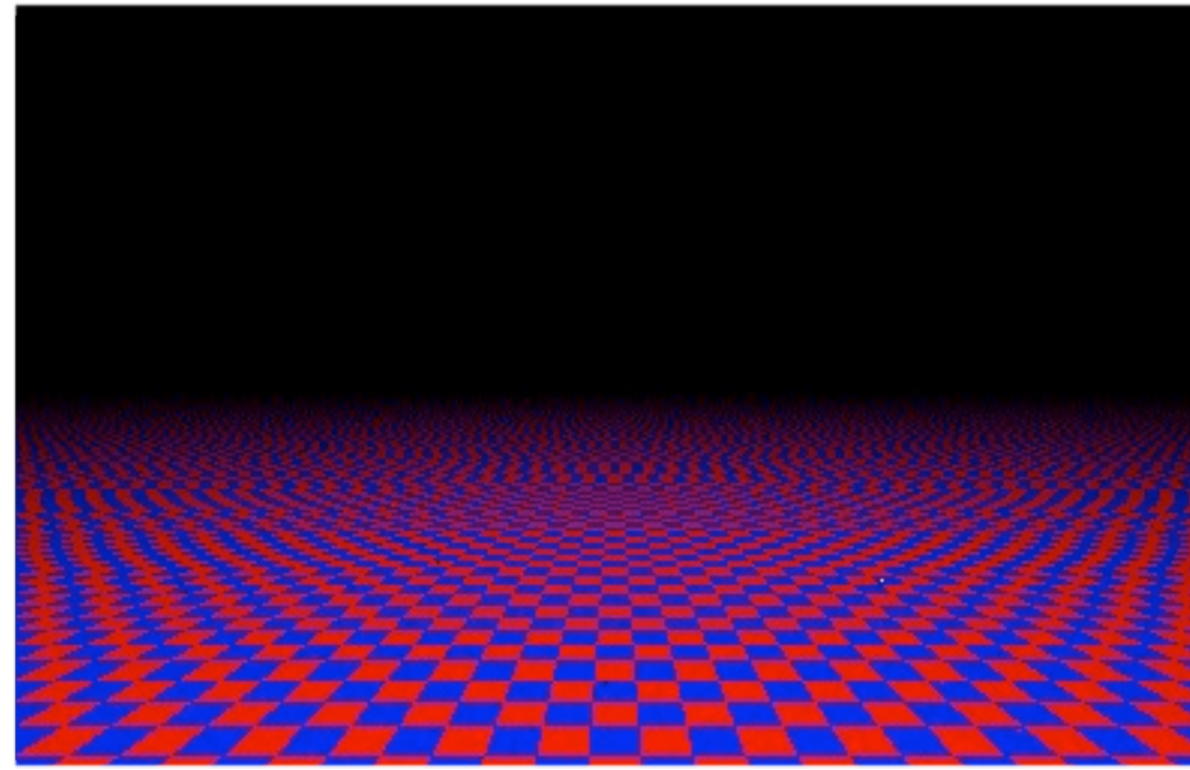
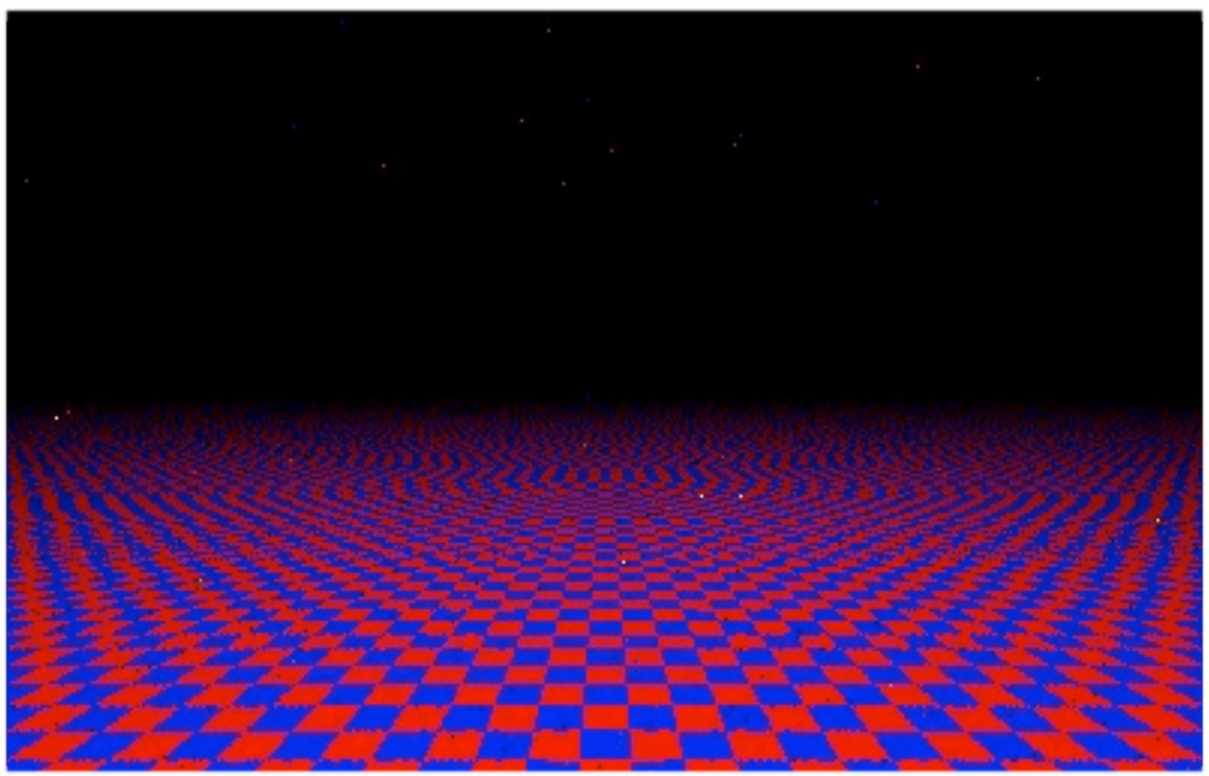
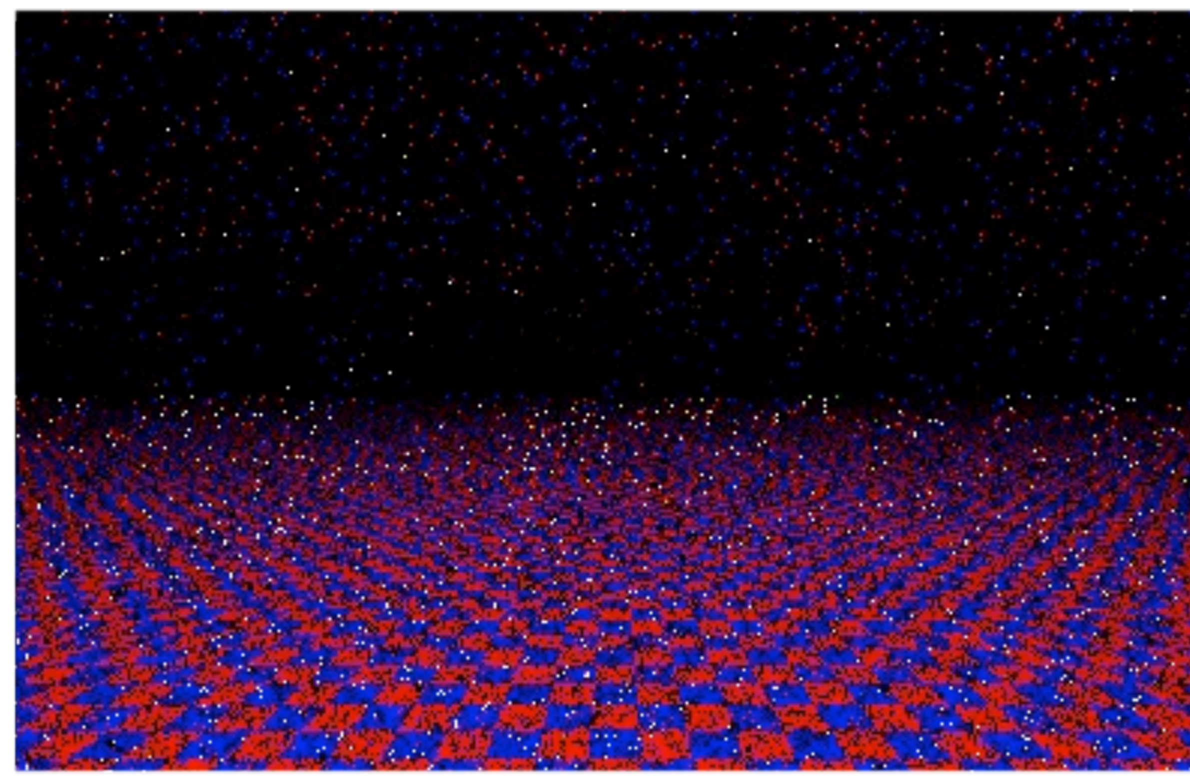
- Won't be able to power all transistors at once
- Need to either go specialized, or use *drowsy* transistors

How can programmers help make systems more energy efficient?

What are the language constructs and system support to enable that?

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.

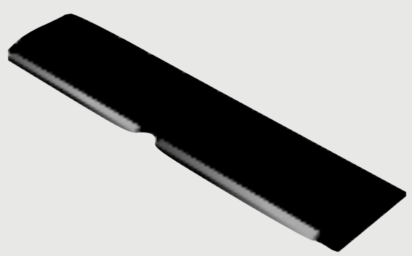
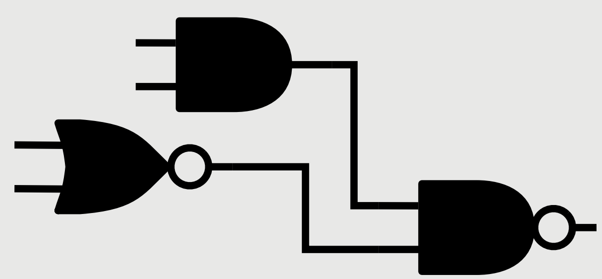
[5] Xuanhua Li and Donald Yeung. Exploiting soft computing for increased fault tolerance. In ASGI, 2006.

[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

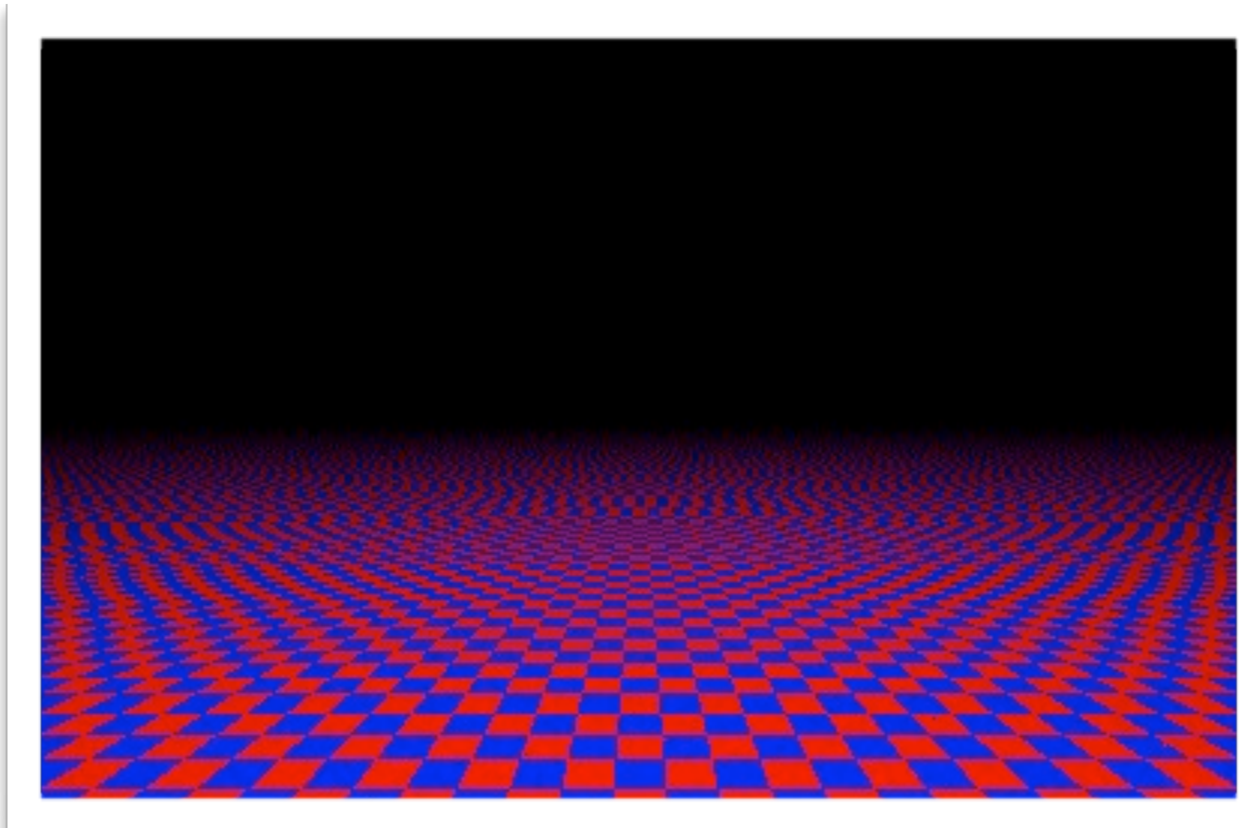
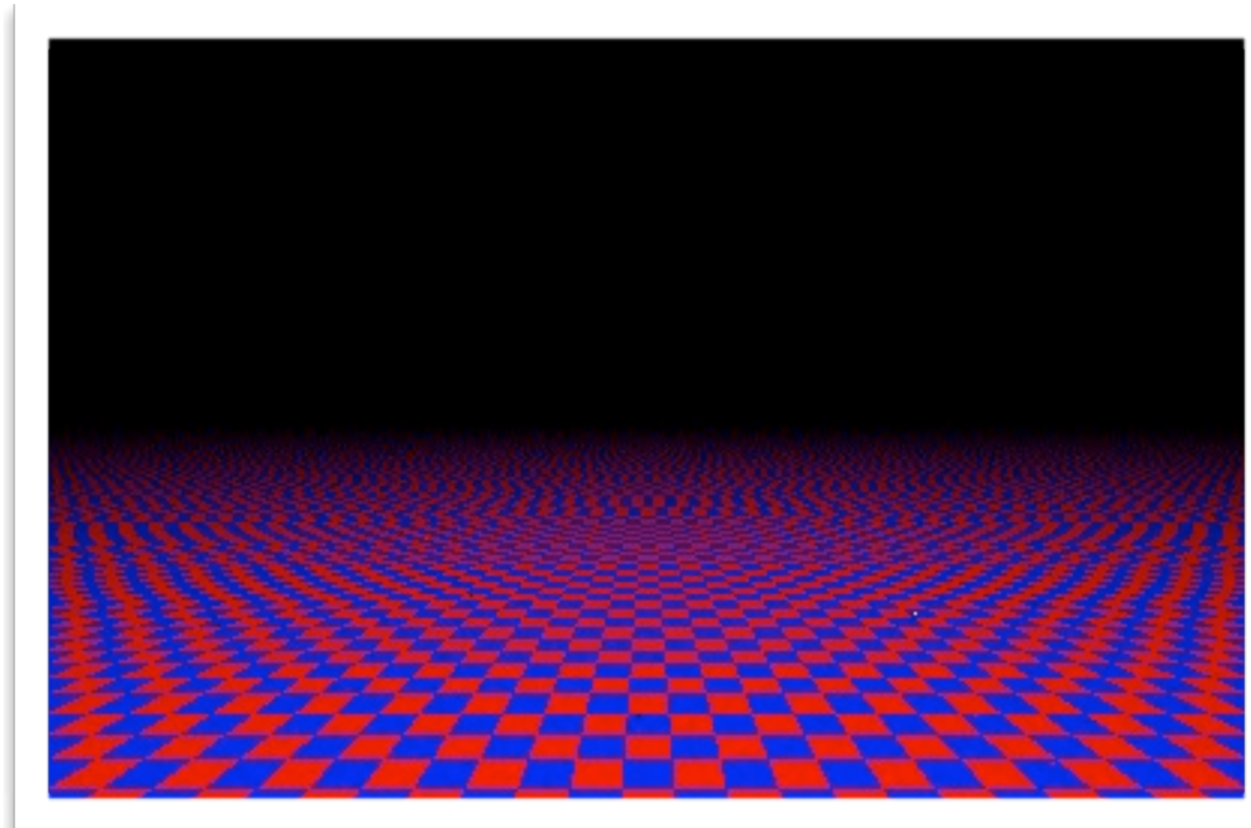
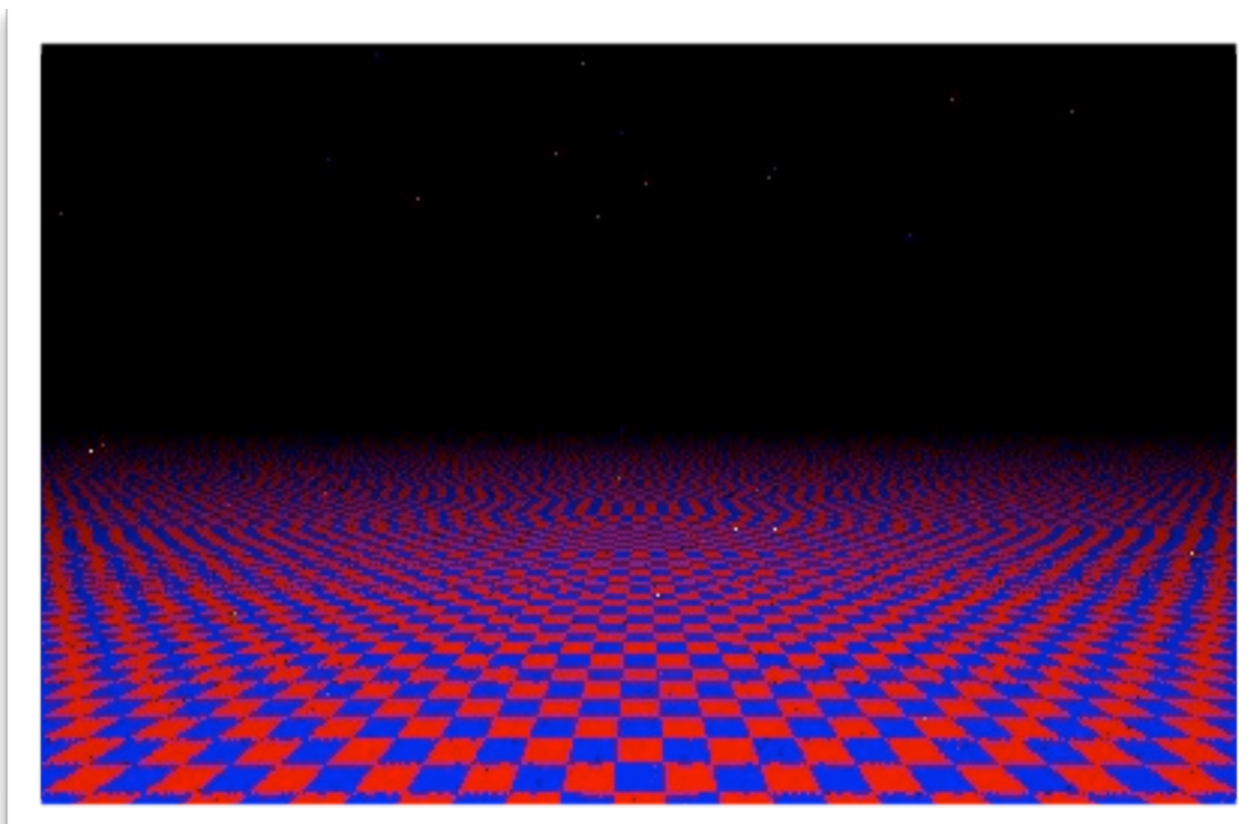
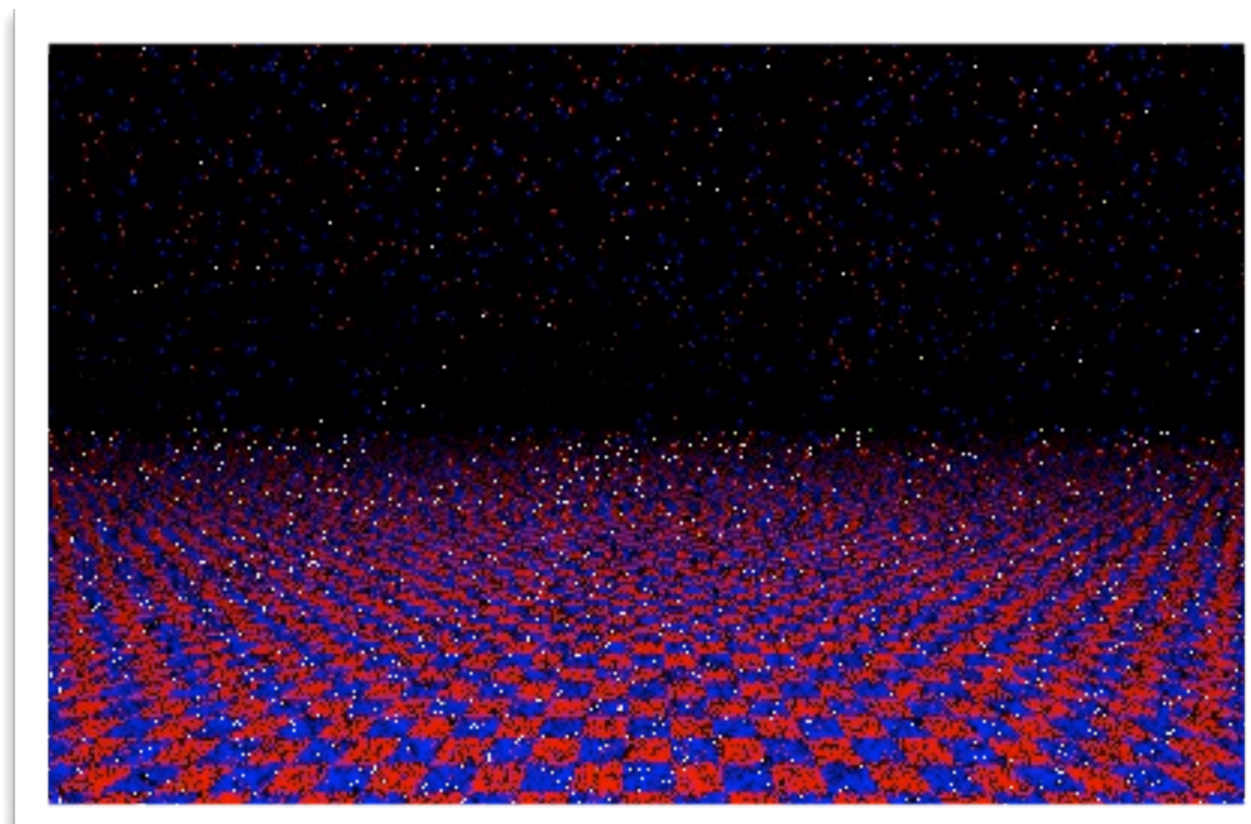
EnerJ		
Storage	Logic	Algorithms
		λ

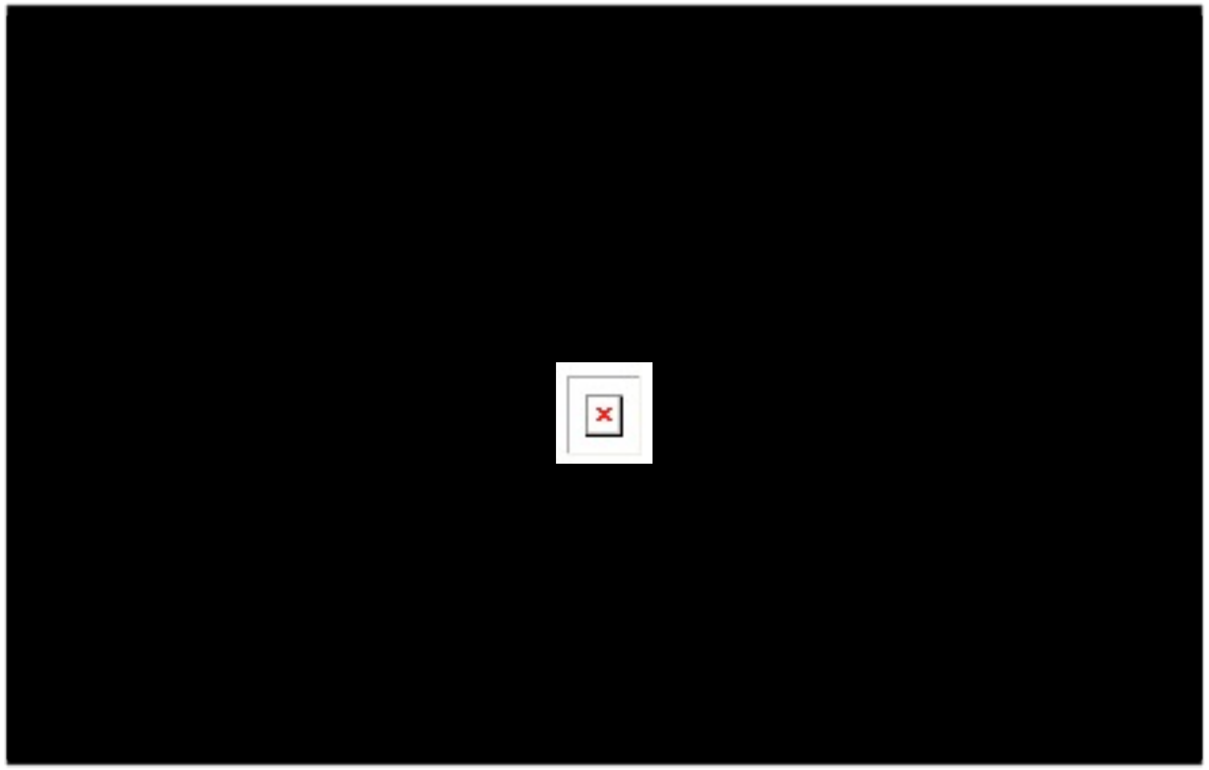
Flikker	Relax	Green
ASPLOS 2011	ISCA 2010	PLDI 2010

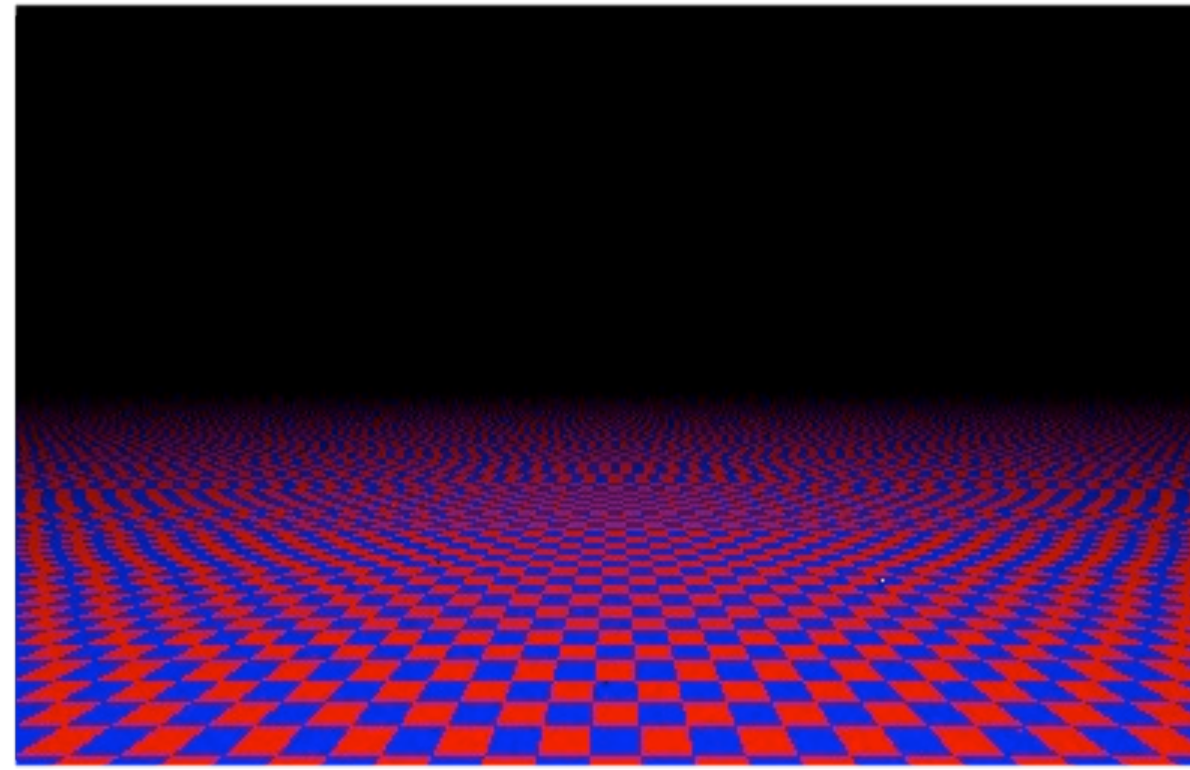
Generality

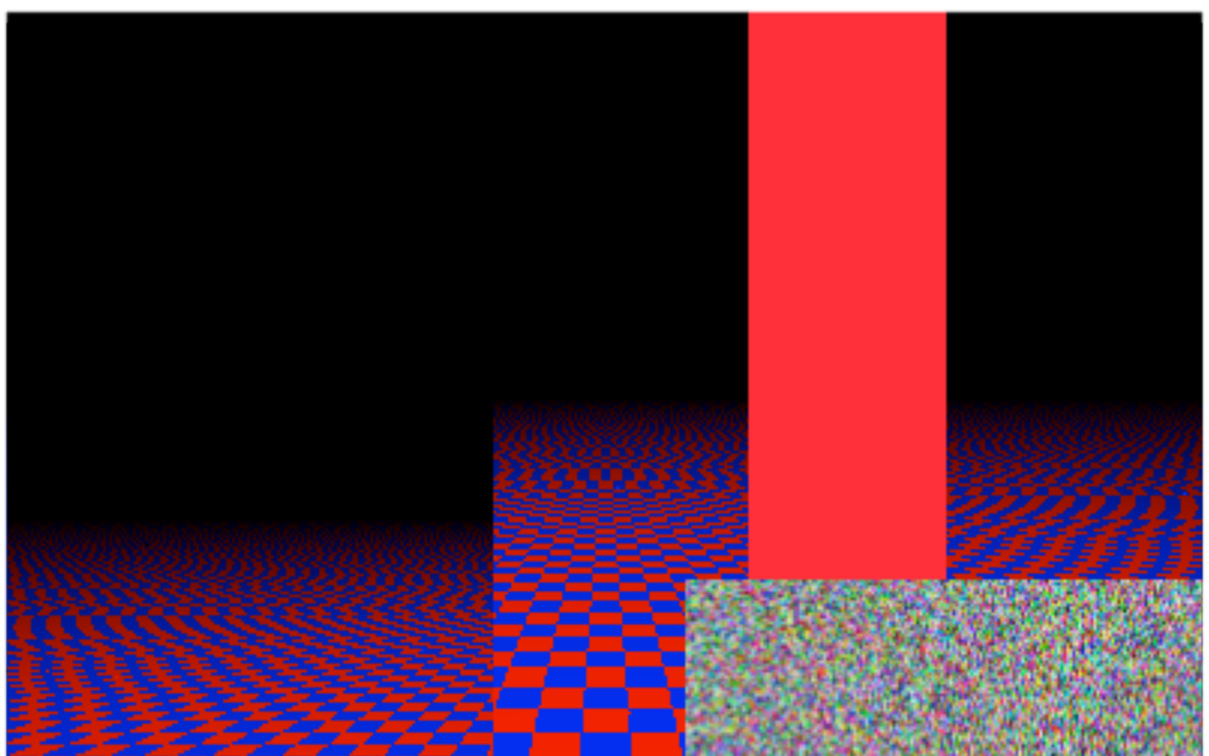
A range of approximation strategies supported with a single abstraction.

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

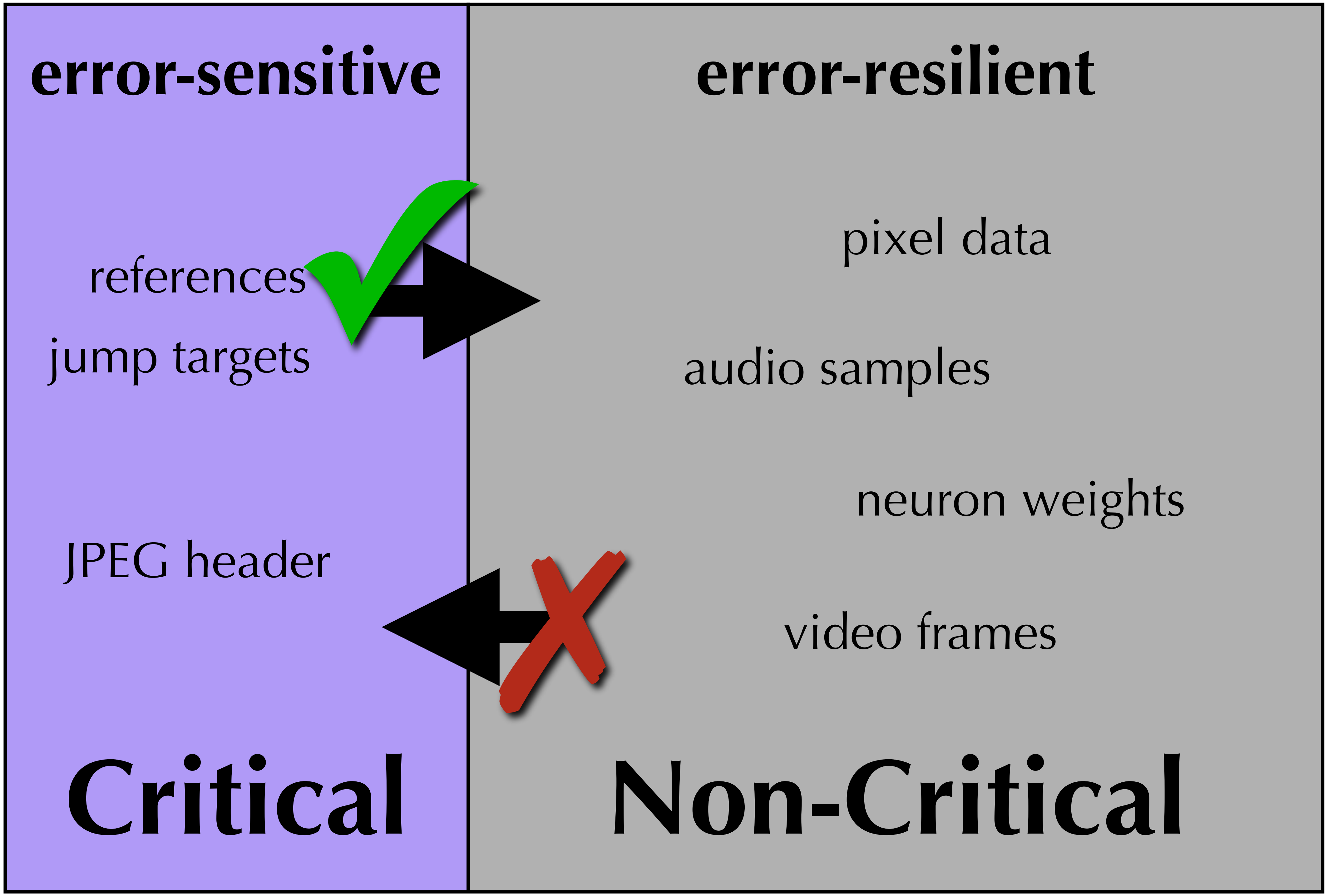












* Press any key to terminate the current application.
* Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.
Press any key to continue _


error-sensitive
error-resilient
references
jump targets
JPEG header
pixel data
audio samples
neuron weights
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 = a; endorse( )

quickChecksum(p);

output(p);
```

Logic approximation: overloading

@Approx int a = ...;

@Precise int p = ...;

p + p;

$+ :$ @Precise int, @Precise int $\rightarrow$ @Precise int

p + a;

a + a;

$+ :$ @Approx int, @Approx int $\rightarrow$ @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
    }
}
```

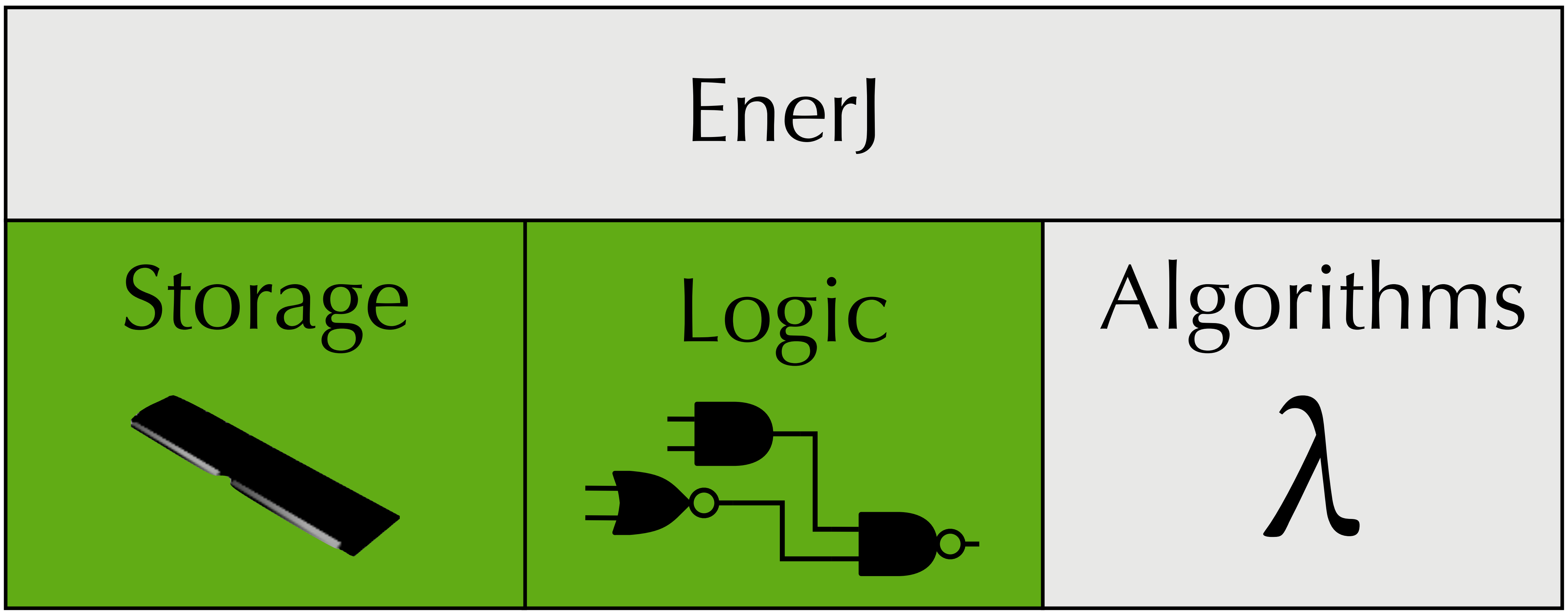
```
class FloatSet {
    @Context  float[] nums = ...;
    float mean() {
        calculate mean
    }

        @Approx float mean_APPROX() {
            take mean of first ½
        }
}

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

Java language extension
using type annotations

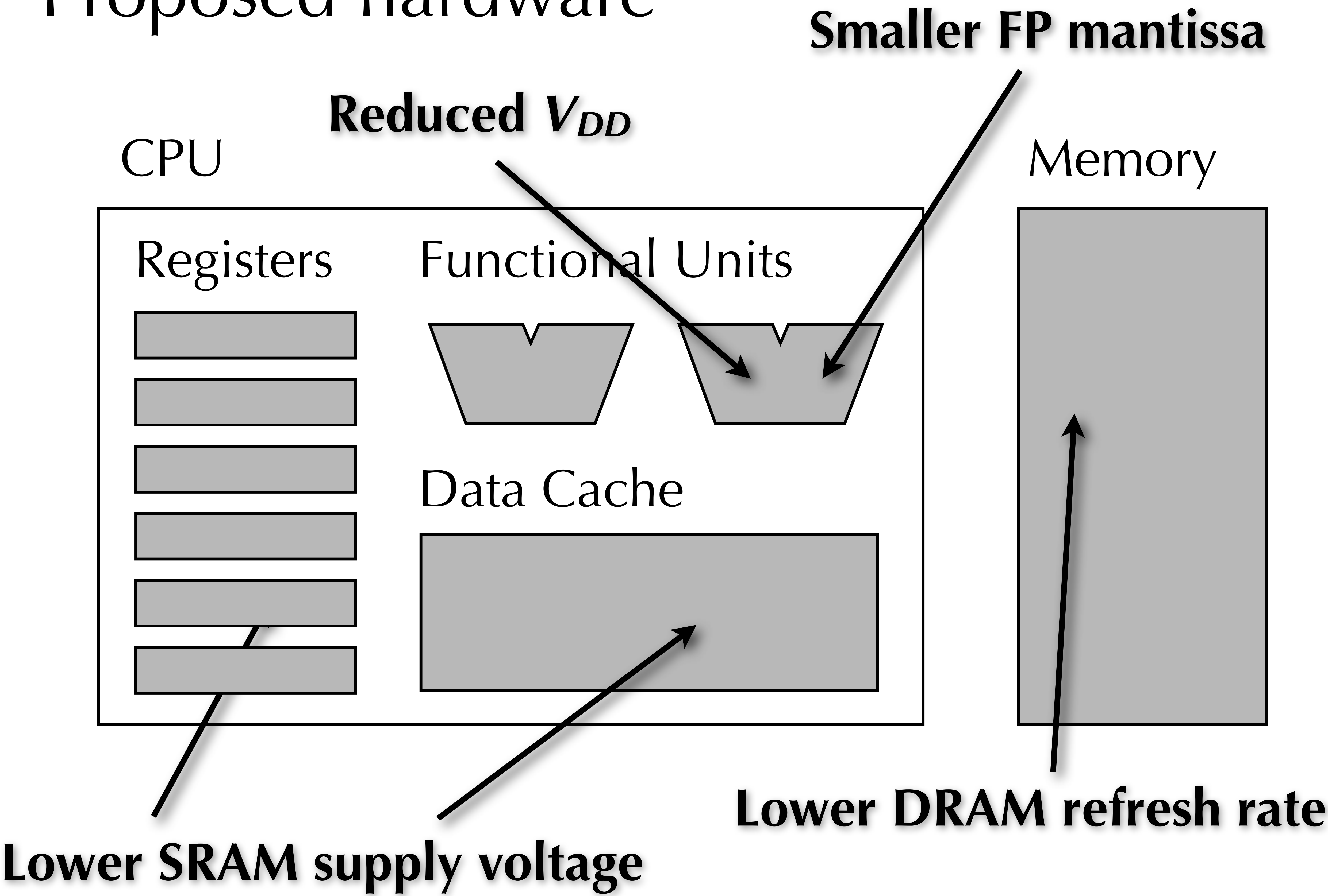
Proposed approximate hardware

Potential energy savings
in existing Java programs

Hypothetical hardware



Proposed hardware



Key aspects of the ISA extensions

Approximate flag in ops/mem instructions

Registers implicitly set as approx/precise based on the last writer

```
add.a $1, $2, $4
```

Memory regions declared as approximate

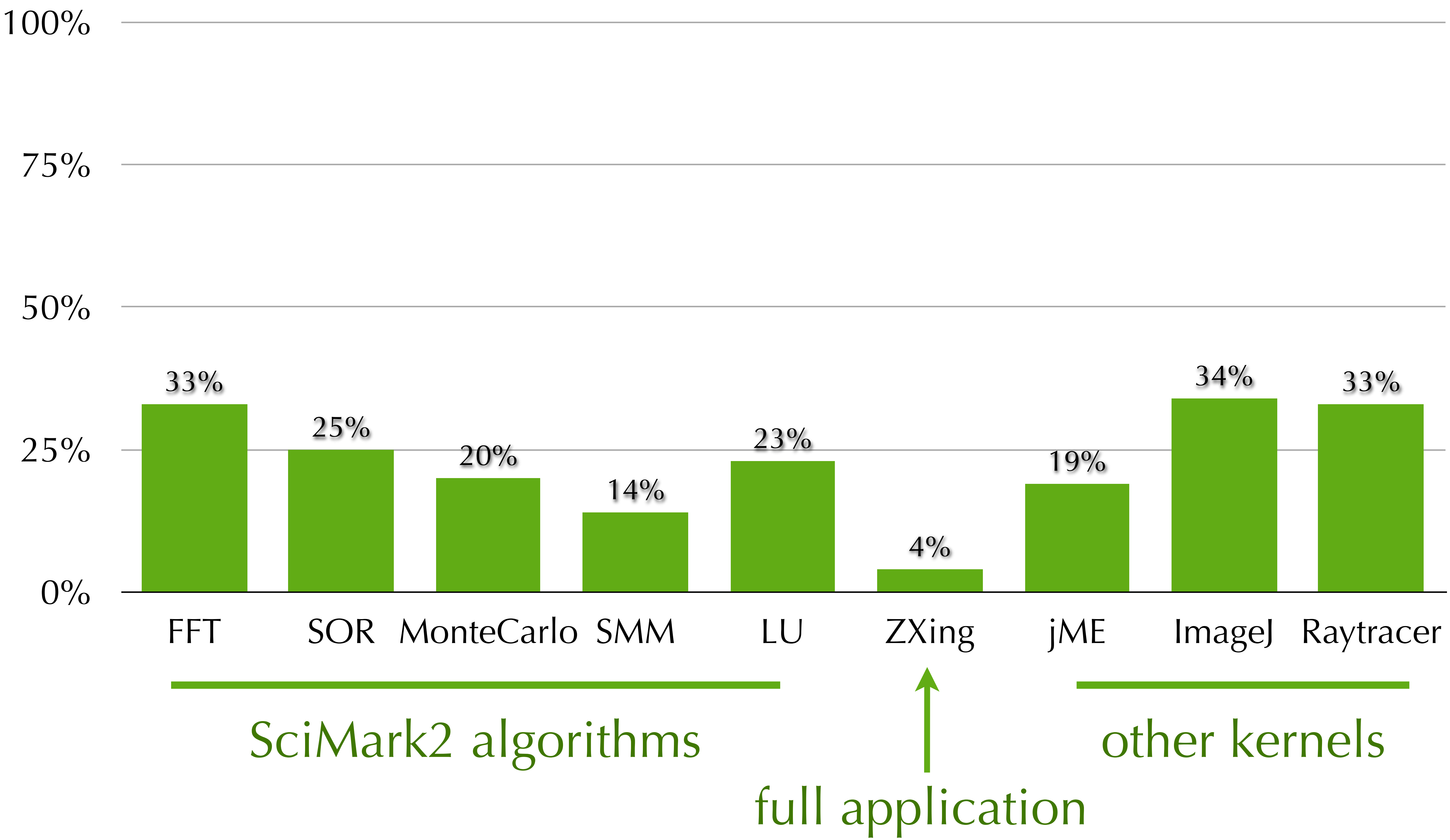
- Set at a block granularity (bitmap per page)
- Cache lines inherit property at fill time

Java language extension
using type annotations

Proposed approximate hardware

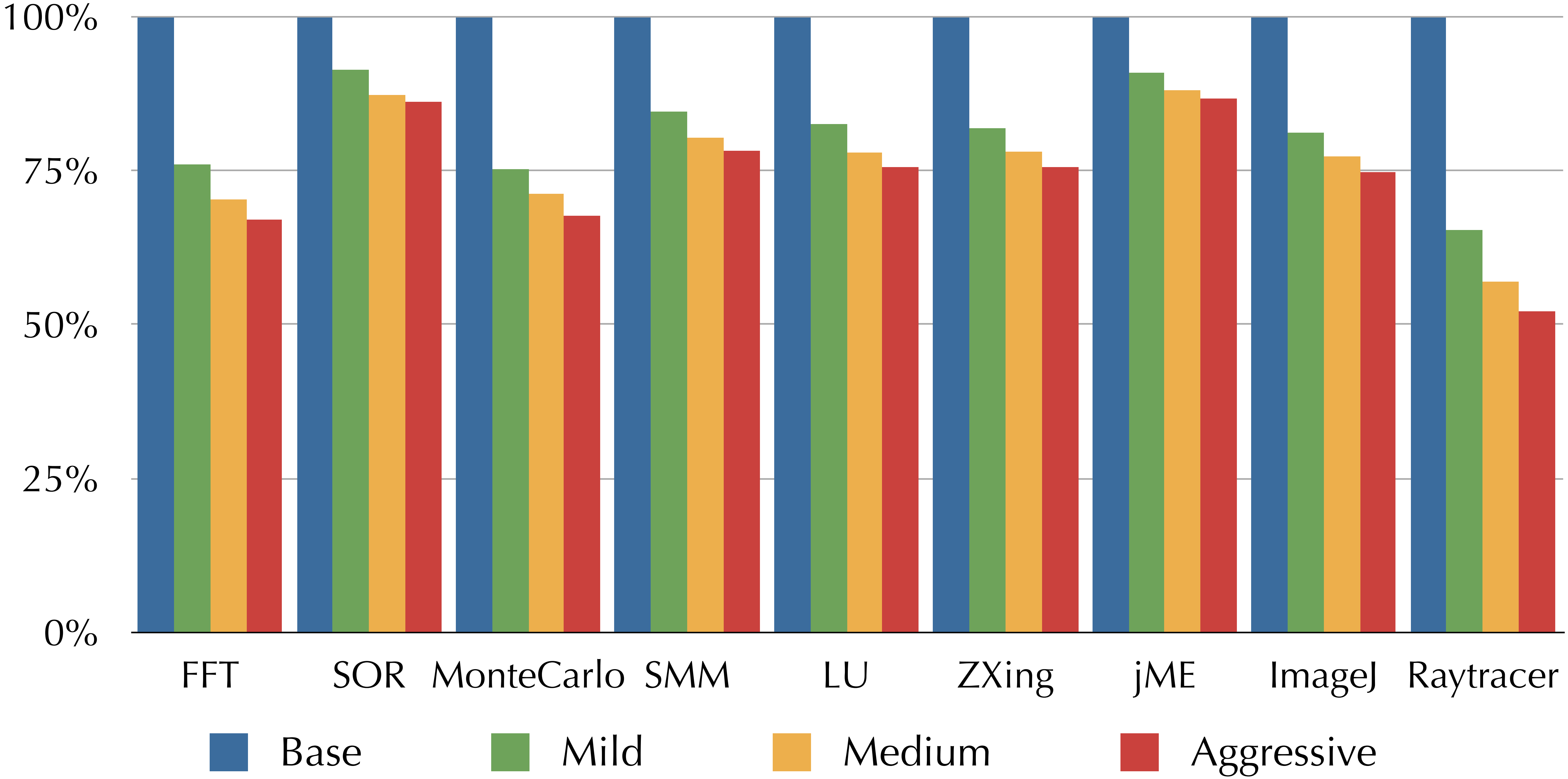
Potential energy savings
in existing Java programs

Annotated declarations



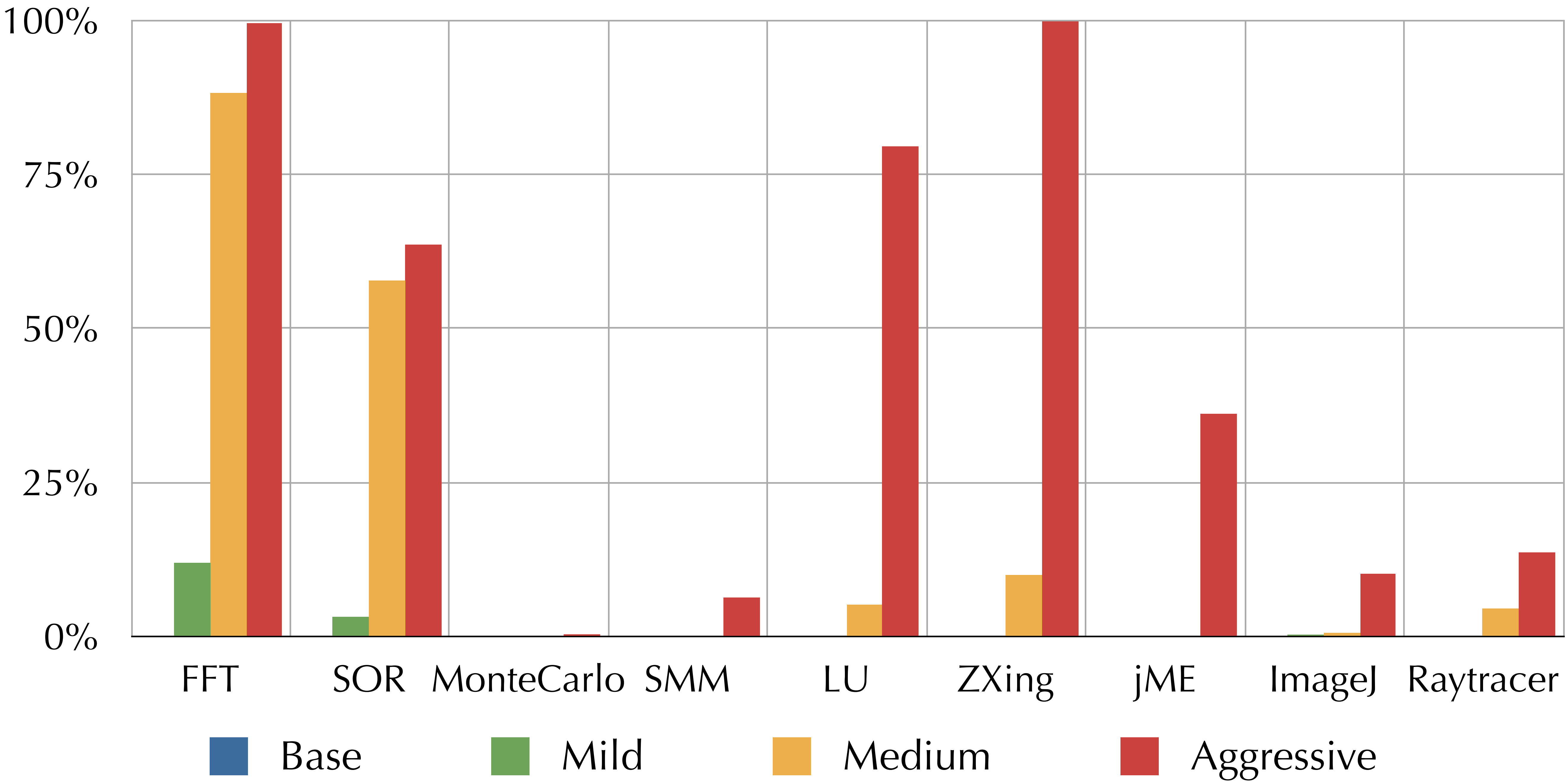
Annotations are sparse & straightforward to insert

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 PLDI paper:

Formal semantics

Noninterference claim

Object layout

Hardware model

Quality-of-service metrics

Safe and General Energy-Aware Programming with Disciplined Approximation

Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam

Luis Ceze, Dan Grossman

University of Washington

		Mild	Medium	Aggressive
DRAM	per-second bit flip probability	10^{-9}	10^{-5}	10^{-3}
DRAM	*memory power saved*	*17%*	*22%*	*24%*
SRAM	read upset probability	$10^{-16.7}$	$10^{-7.4}$	10^{-3}
SRAM	write failure probability	$10^{-5.59}$	$10^{-4.94}$	10^{-3}
SRAM	*supply power saved*	*70%*	*80%*	*90%**
FPU	single-precision mantissa bits	16	8	4
FPU	double-precision mantissa bits	32	16	8
FPU	*energy saved per operation*	*32%*	*78%*	*85%**
FPU/ALU	arithmetic timing error probability	10^{-6}	10^{-4}	10^{-2}
FPU/ALU	*energy saved per operation*	*12%**	*22%*	*30%*

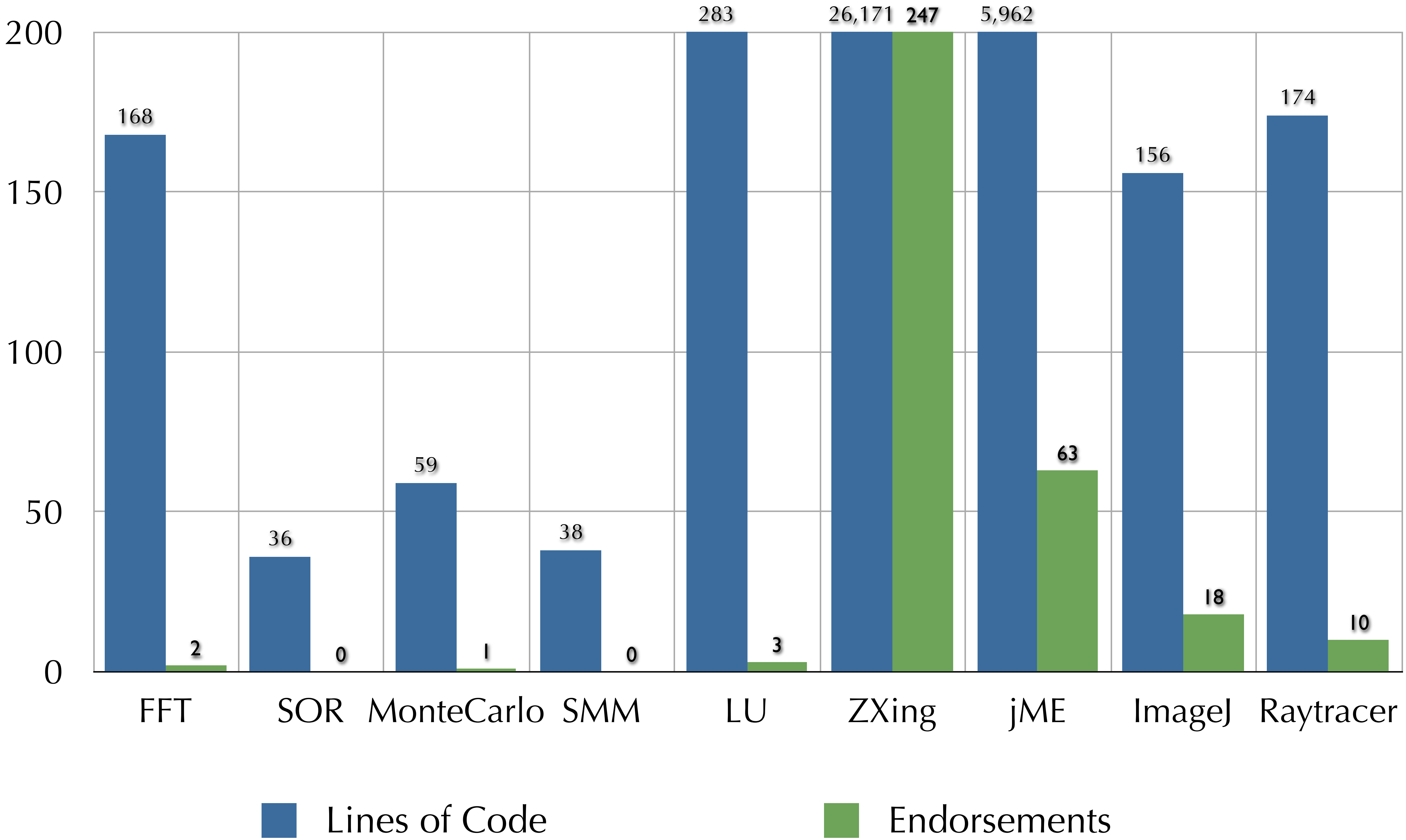
Energy model

integer operations	37 units	(22 fetch/decode)
floating point operations	40 units	(22 fetch/decode)

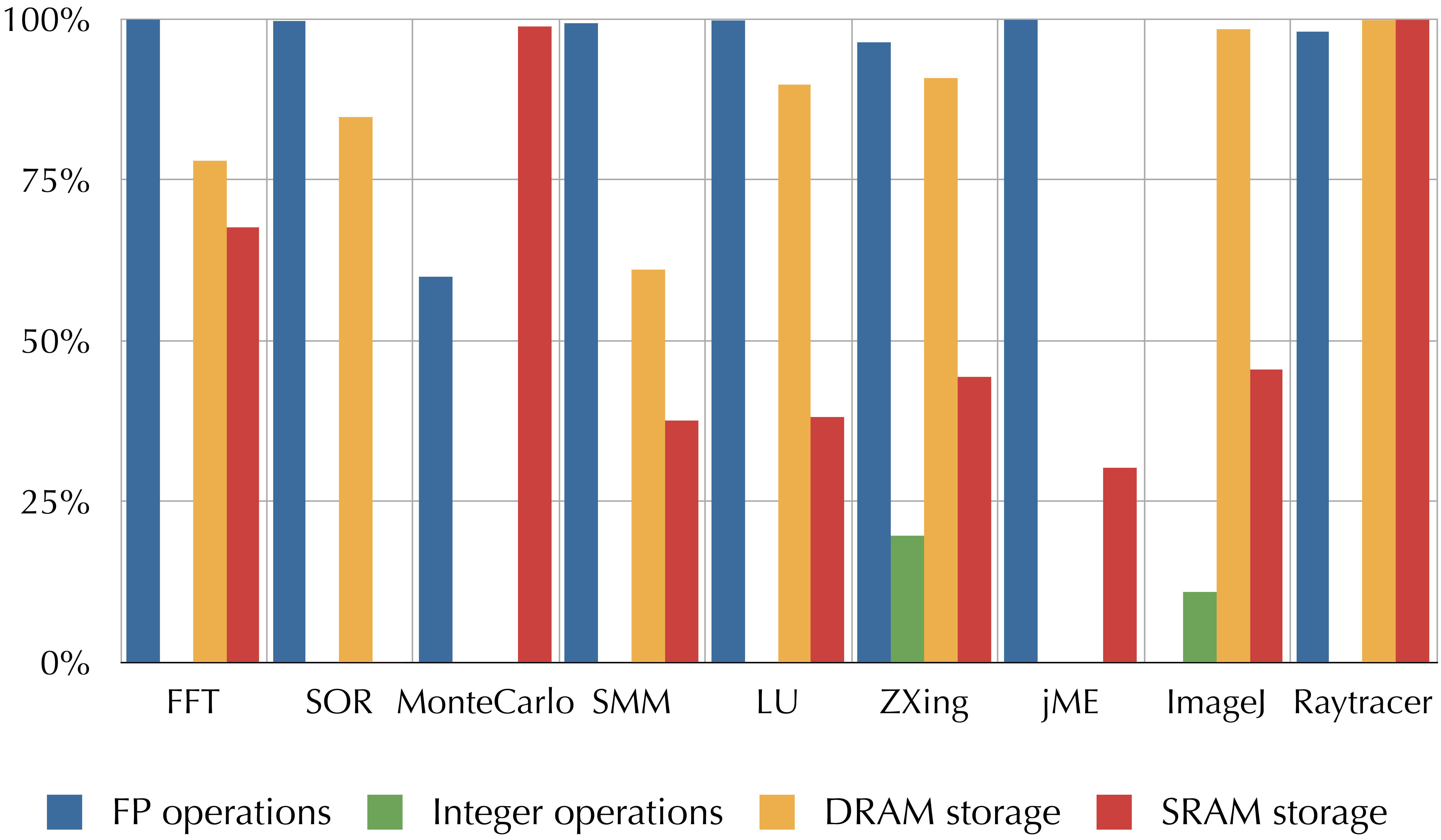
SRAM (cache & registers)	35% of chip power
instruction execution	65%

DRAM	45% of system power
processor	55%

Endorsements and lines of code



What can be approximated?



Green [Baek, Chilimbi; PLDI 2010]

Multiple method implementations

Online QoS monitoring

Relax [de Kruijf, Nomura, Sankaralingam; ISCA 2010]

Annotate "relaxed" *code blocks*

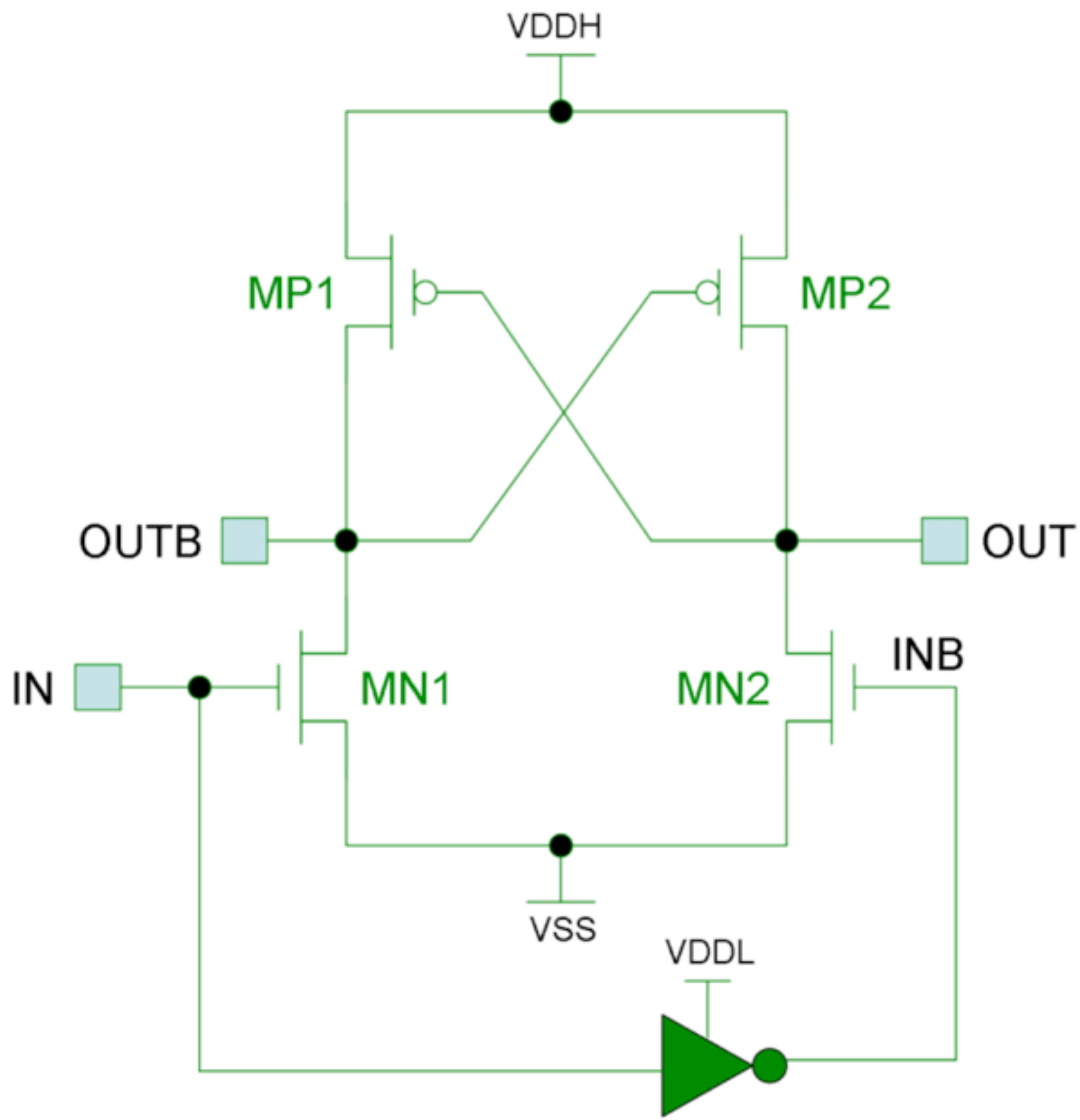
Permit, detect, and report hardware faults

Memory layout

Approximation at cache line granularity

Objects can only save energy when they occupy multiple cache lines

Large arrays of primitives provide greater opportunities for approximation


VDDH
MP1
MP2
OUTB
OUT
IN
MN1
MN2
INB
VSS
VDDL