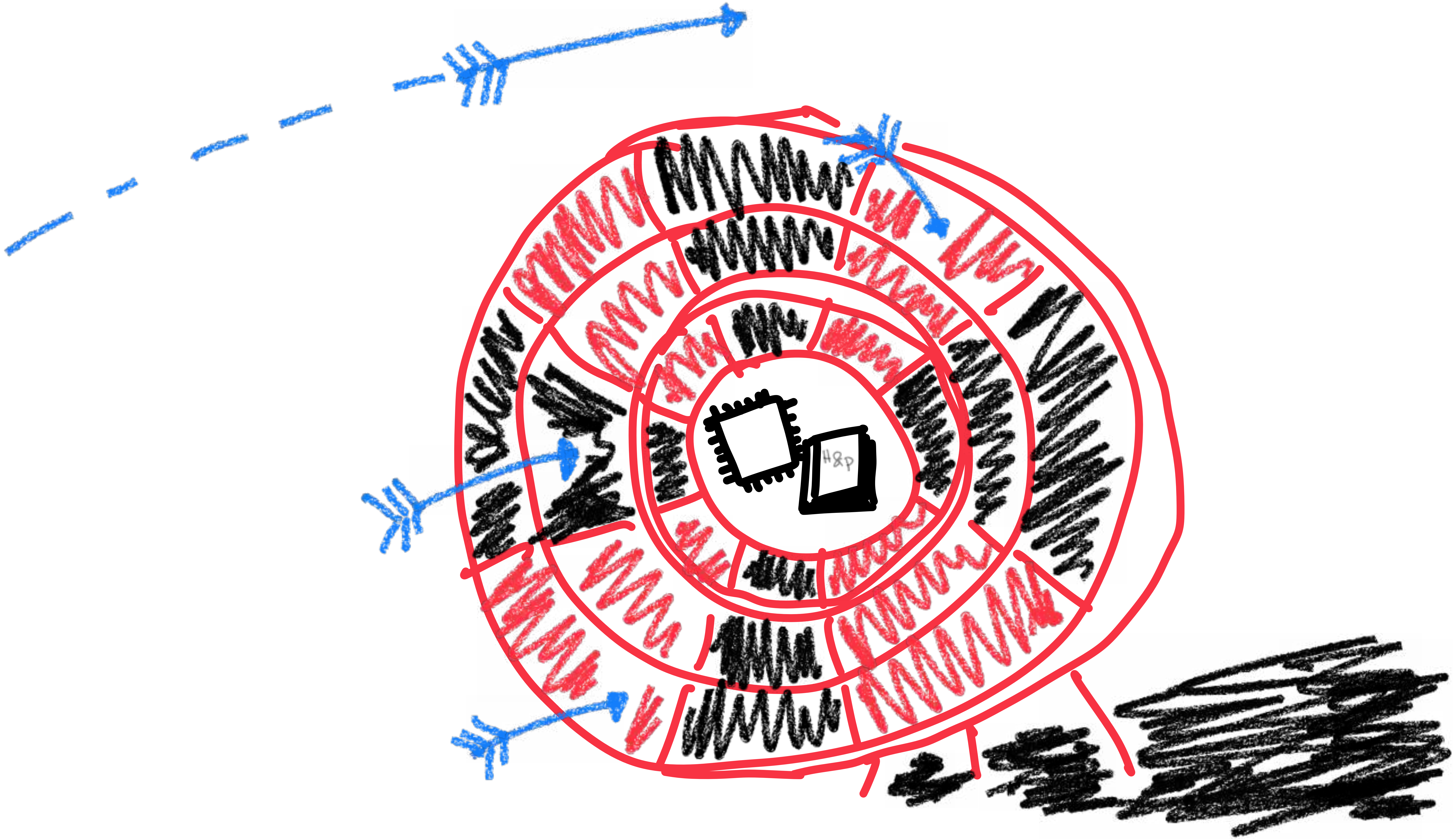


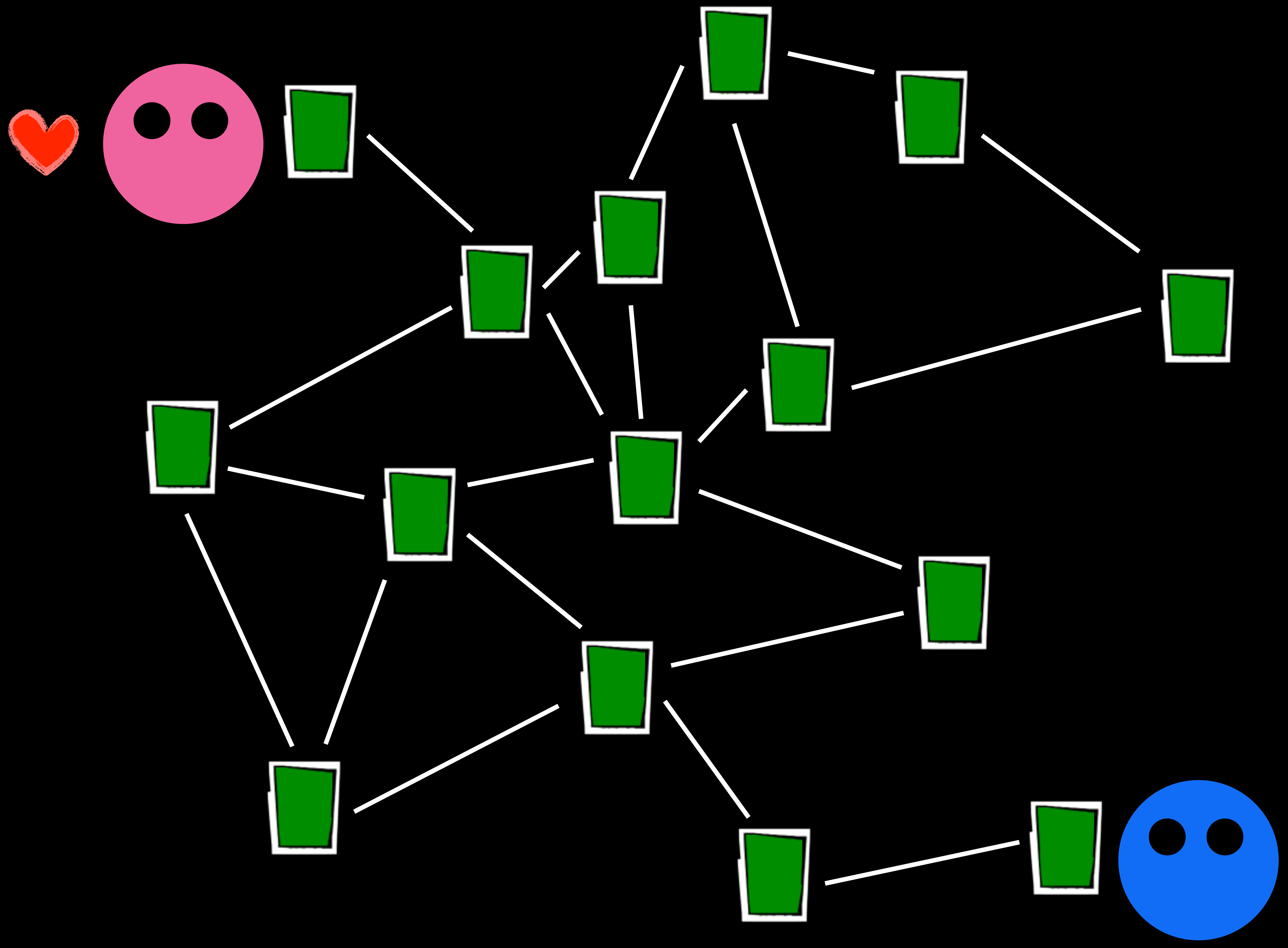
Approximating Computer Architecture

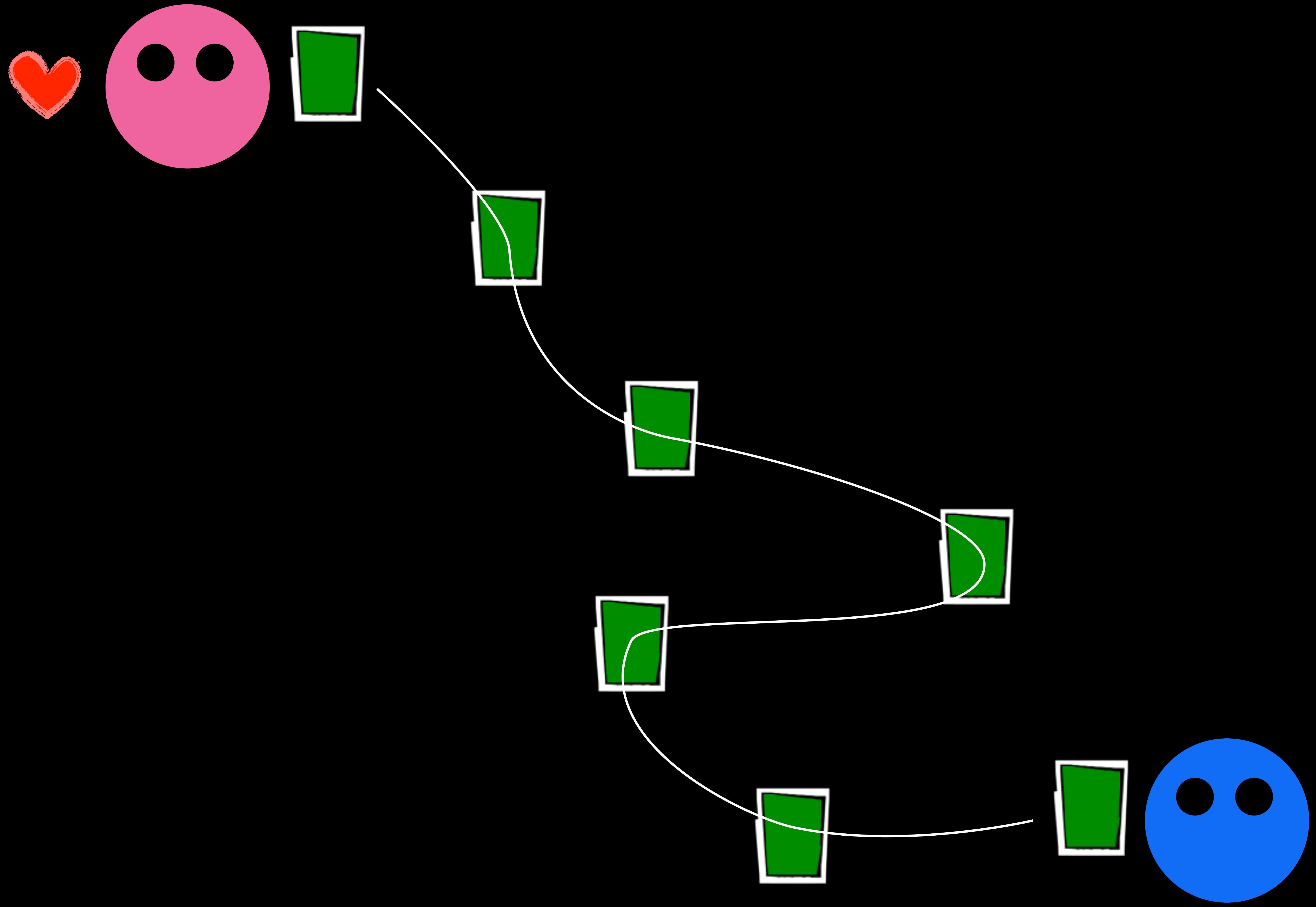
Adrian Sampson Cornell

SOME DUBIOUS ADVICE!

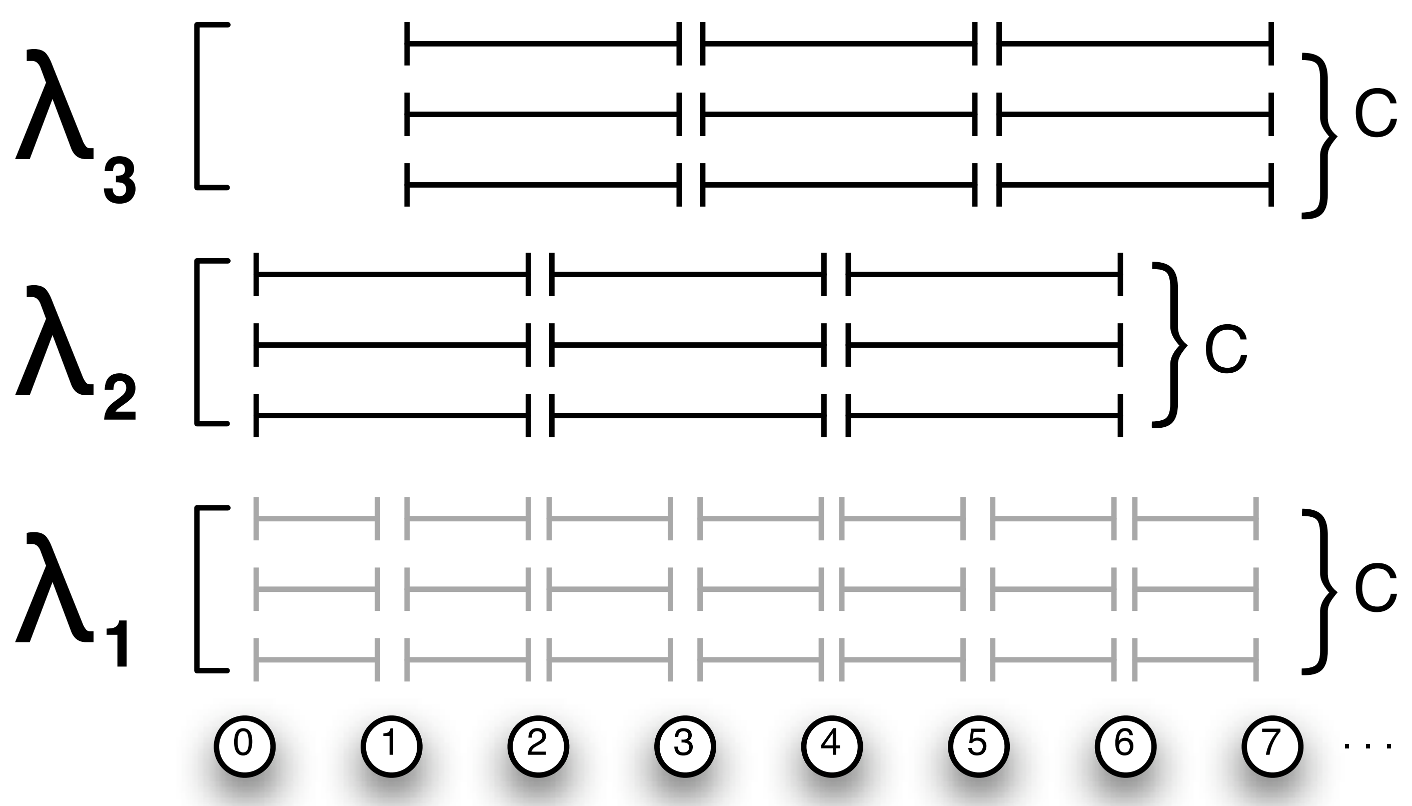
1. NEVER LISTEN TO ADVICE
2. ALWAYS LISTEN TO ADVICE
3. IT'S NOT NAÏVETÉ; IT'S <u>BEGINNER'S</u> <u>MIND</u>
4. THE BEST COMPUTER ARCHITECTURE RESEARCH ALWAYS "CHEATS"
5. DON'T DO FRAMEWORKS
6. START A BLOG
7. TALK TO YOUR FRIENDS


HARVEY MUDD COLLEGE
(YESTERDAY!)
HARVEY MUDD COLLEGE





virtual topology   traffic grooming



Dynamic Grooming Algorithm (DGA)

Input: A connection request $\mathbf{c}$ and a network state function U.

Output: Returns $true$ if $\mathbf{c}$ was satisfied and $false$ otherwise. If it was satisfied, U and $S(\mathbf{c})$ reflect the new network state.

$S(\mathbf{c}) \leftarrow \emptyset$
$s \leftarrow (L(\mathbf{c}), \min\{R(\mathbf{c}), L(\mathbf{c}) + r\})$
while $L(s) \neq R(s)$ **do**
    if $U(s) < C$ **then**
        `//` s `is available to be used by` $\mathbf{c}$
        $U(s)$++
        let $S(\mathbf{c})$ contain s
        $s \leftarrow (R(s), \min\{R(\mathbf{c}), R(s) + r\})$
    else
        `// a conflict occurs with` s
        $s \leftarrow (L(s), R(s) - 1)$
    end
end
return $R(s) == R(\mathbf{c})$ `// true iff connection is successfully satisfied`


RAN
LIBESKIND -
HADAS,
HARVEY MUDD

1. NEVER LISTEN TO ADVICE
2. ALWAYS LISTEN TO ADVICE
3. IT'S NOT NAÏVETÉ; IT'S <u>BEGINNER'S</u> <u>MIND</u>
4. THE BEST COMPUTER ARCHITECTURE RESEARCH ALWAYS "CHEATS"
5. DON'T DO FRAMEWORKS
6. START A BLOG
7. TALK TO YOUR FRIENDS


University of
Washington

Code-Communication Specifications

Writer Thread | Reader Thread

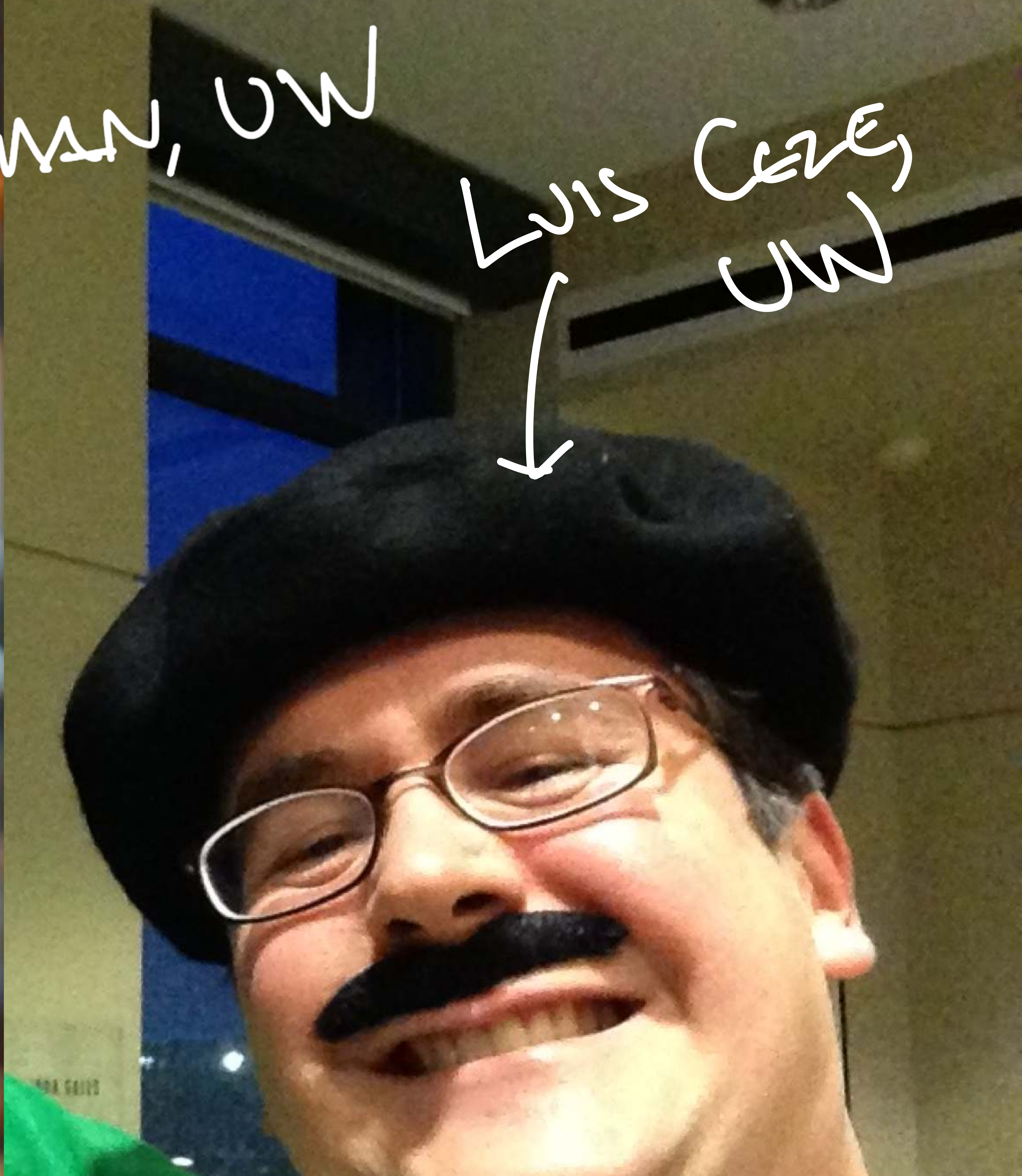
What code may communicate across threads?

`enqueue` → ✔ → `dequeue`

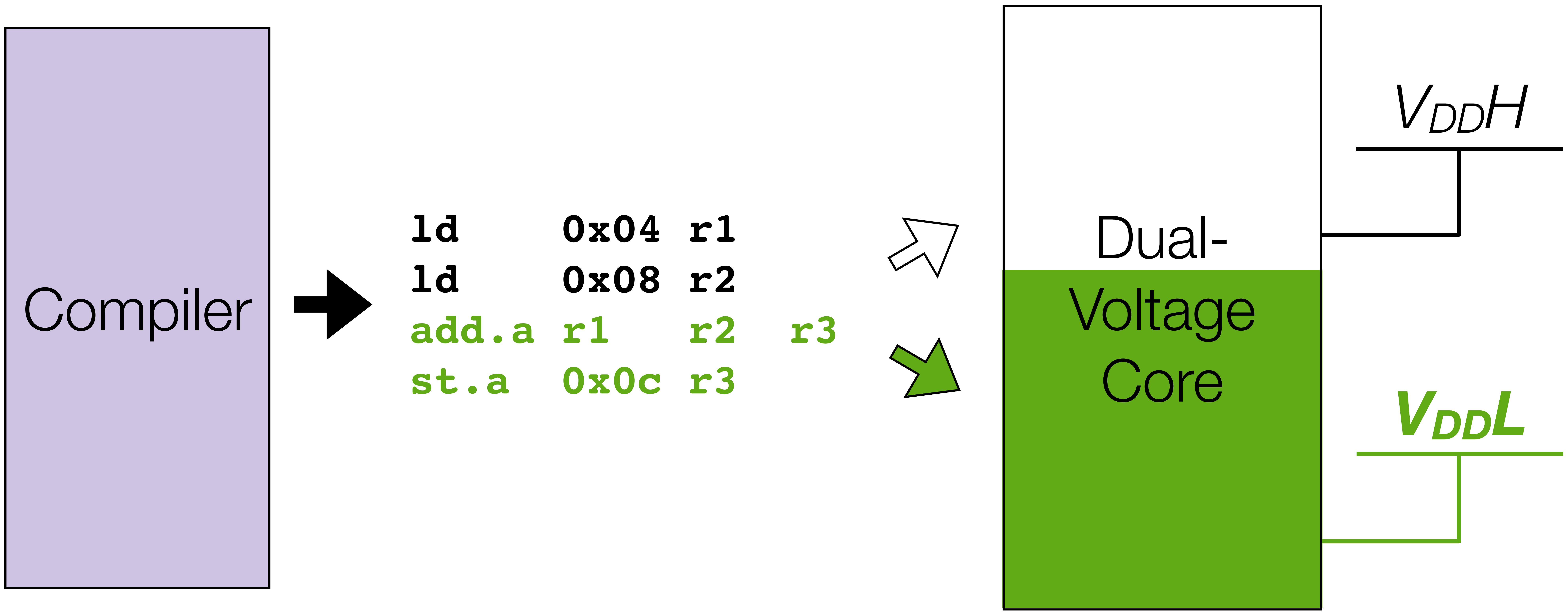
`enqueue` → ✘ → `render`

May writes in `enqueue` be read by other threads in `dequeue`?


COMPUTER ARCHITECTURE?
Is this [redacted]?


DAN GROSSMAN, UW

LUIS CEZE, UW

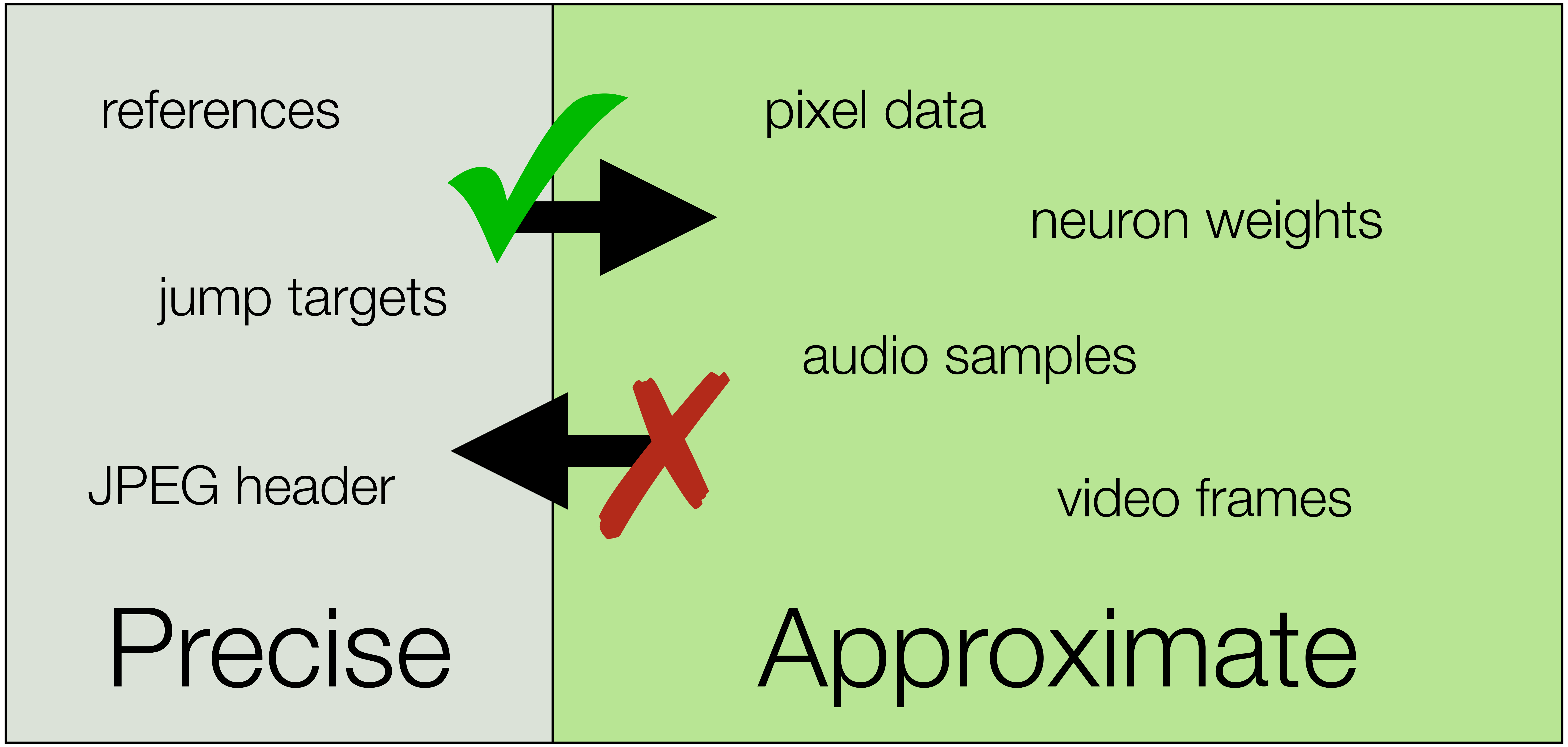
ML?
FUZZY SEARCH
PHYSICAL SIMULATION



Compiler
ld 0x04 r1
ld 0x08 r2
add.a r1 r2 r3
st.a 0x0c r3
Dual-Voltage Core
VDDH
VDDL

Safety by isolation

[PLDI 2011]



EnerJ type qualifiers

[PLDI 2011]

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

✗
```
p = a;
```

✓
```
a = p;
```

EnerJ type qualifiers

[PLDI 2011]

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

@Precise int p = ...;

✗ p = a;

✓ a = p;
```

Operator overloading for approximate computation

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

@Precise int p = ...;

p + p;

p + a;

a + a;
```

precise addition

approximate addition

1. NEVER LISTEN TO ADVICE
2. ALWAYS LISTEN TO ADVICE 初心
3. IT'S NOT NAÏVETÉ; IT'S BEGINNER'S MIND
4. THE BEST COMPUTER ARCHITECTURE RESEARCH ALWAYS "CHEATS"
5. DON'T DO FRAMEWORKS
6. START A BLOG
7. TALK TO YOUR FRIENDS

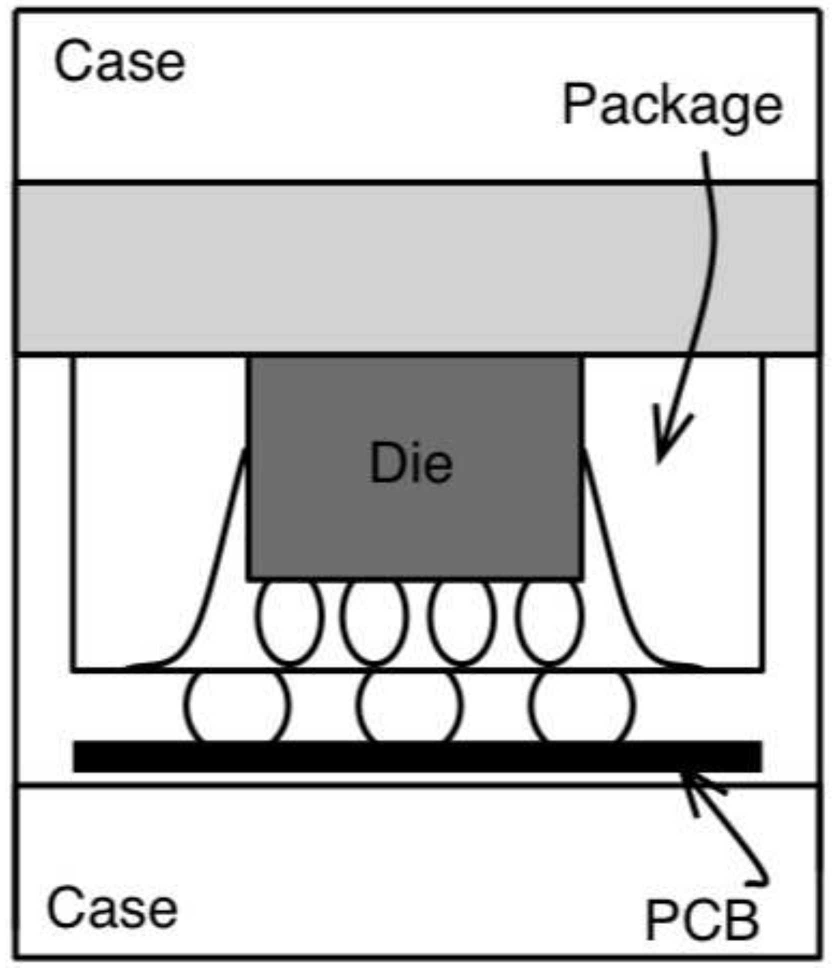
In the Proceedings of the

Marios Papaefthymi

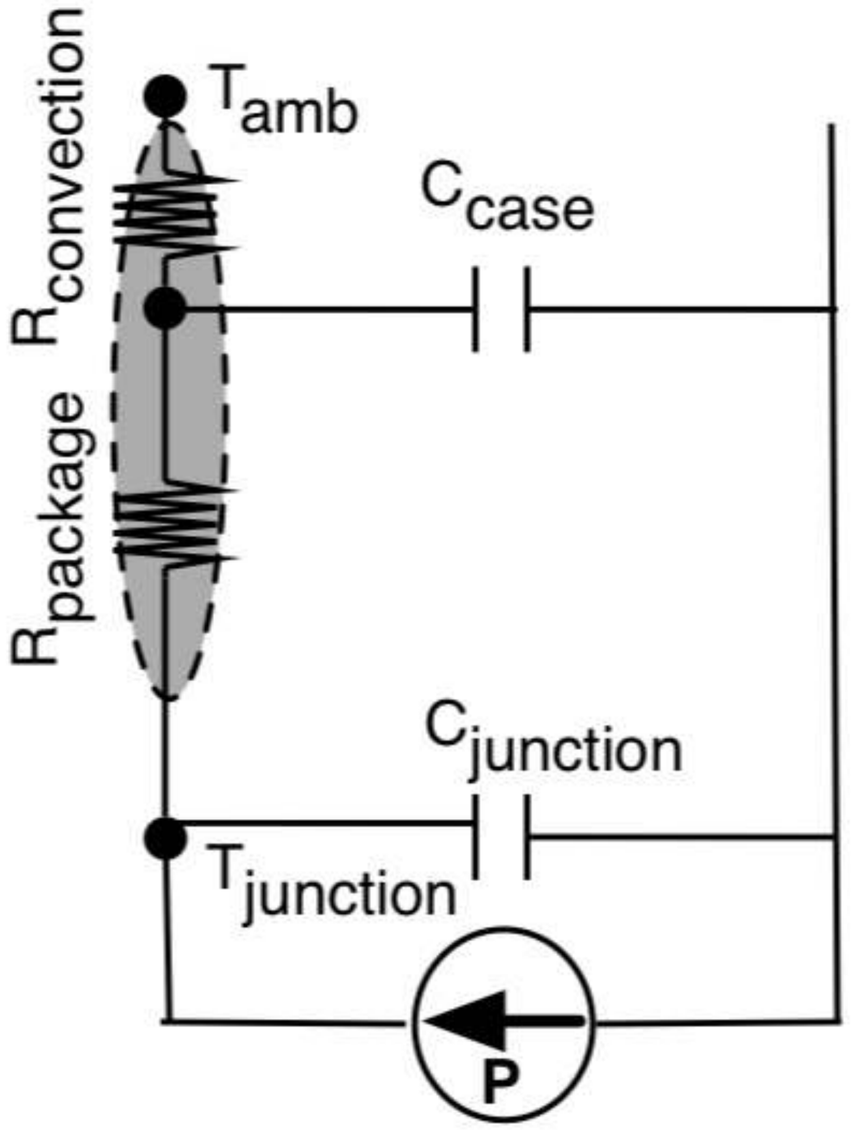
*Department o

†Department of El

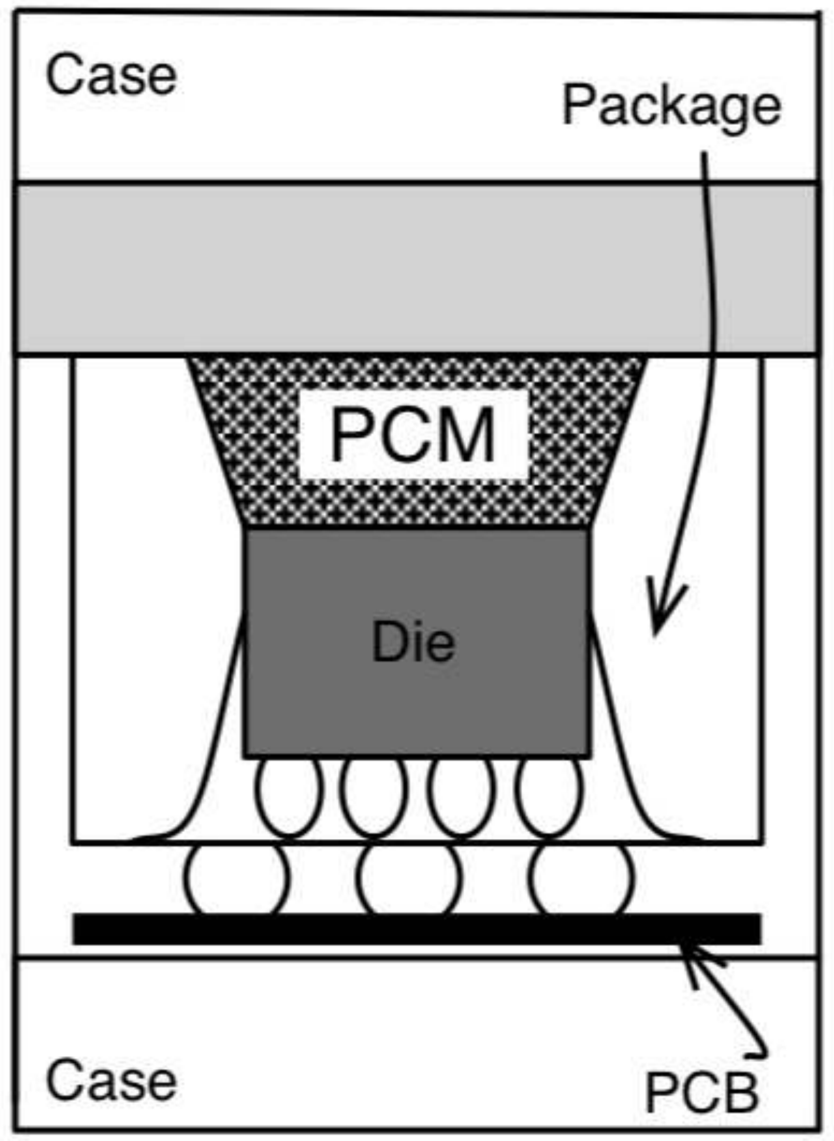
‡Depart



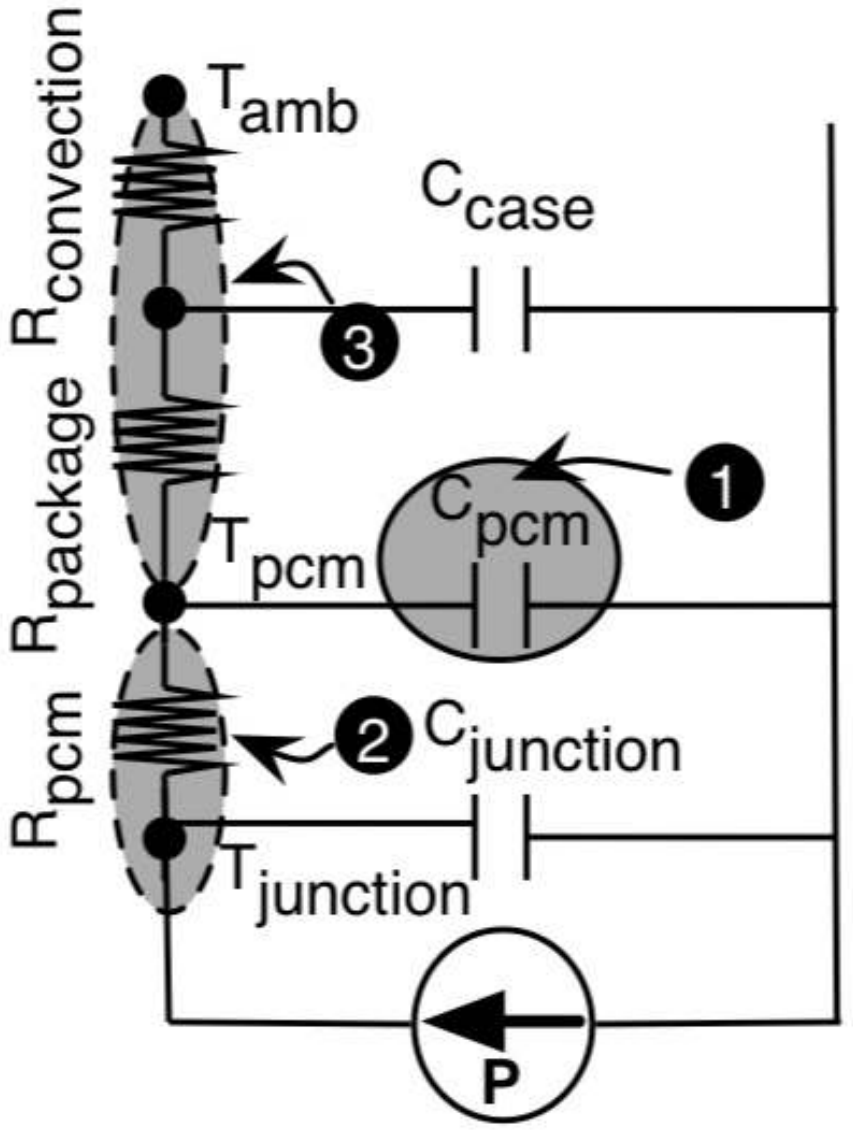
(a)



(b)



(c)



(d)

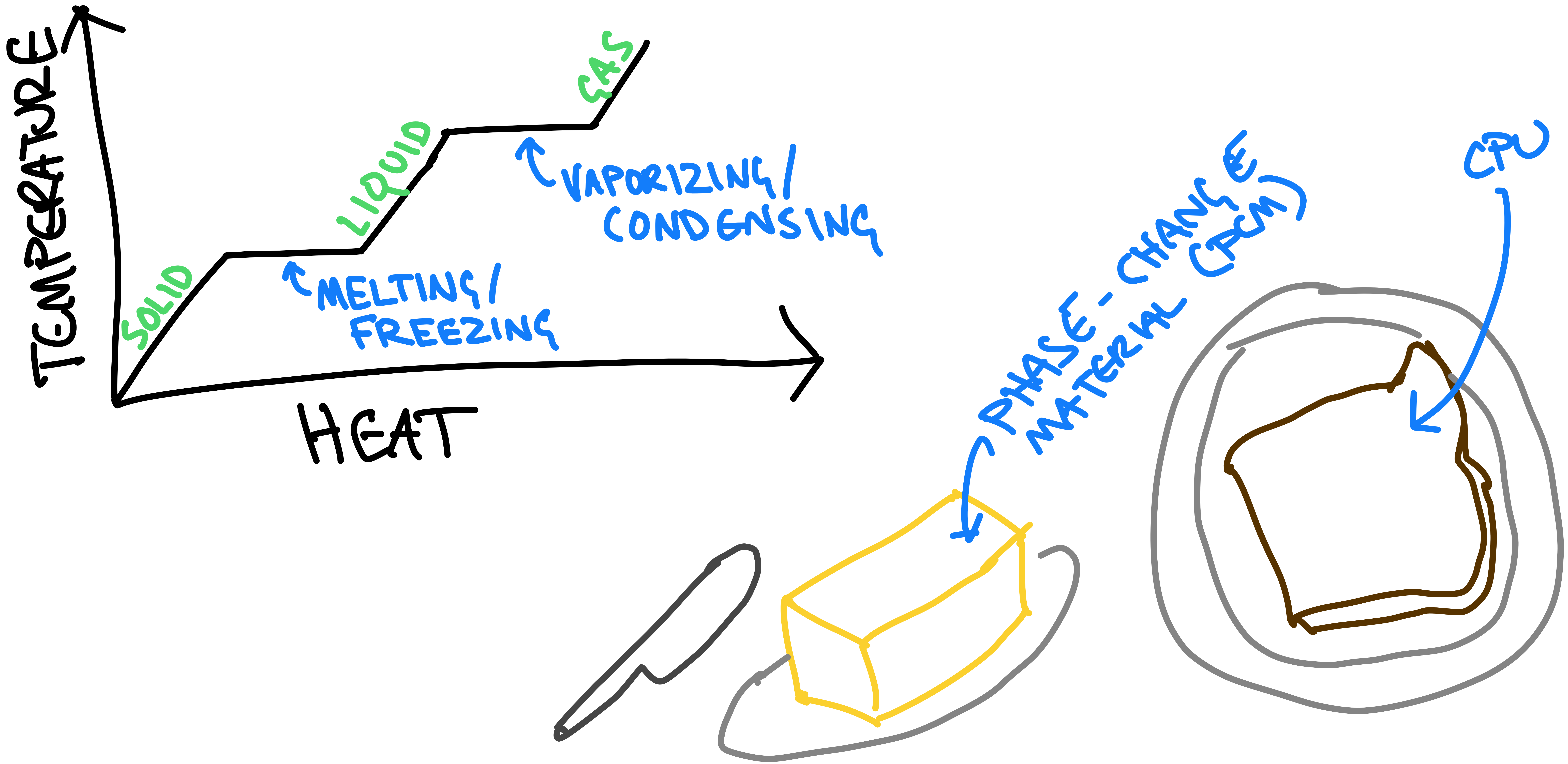
Abstract

Although transistor density continues to increase, voltage scaling has stalled and thus power density is increasing each technology generation. Particularly in mobile devices, which have limited cooling options, these trends lead to a utilization wall *in which sustained chip performance is limited primarily by power rather than area. However, many mobile applications do not demand sustained performance; rather they comprise short bursts of computation in response to sporadic user activity.*

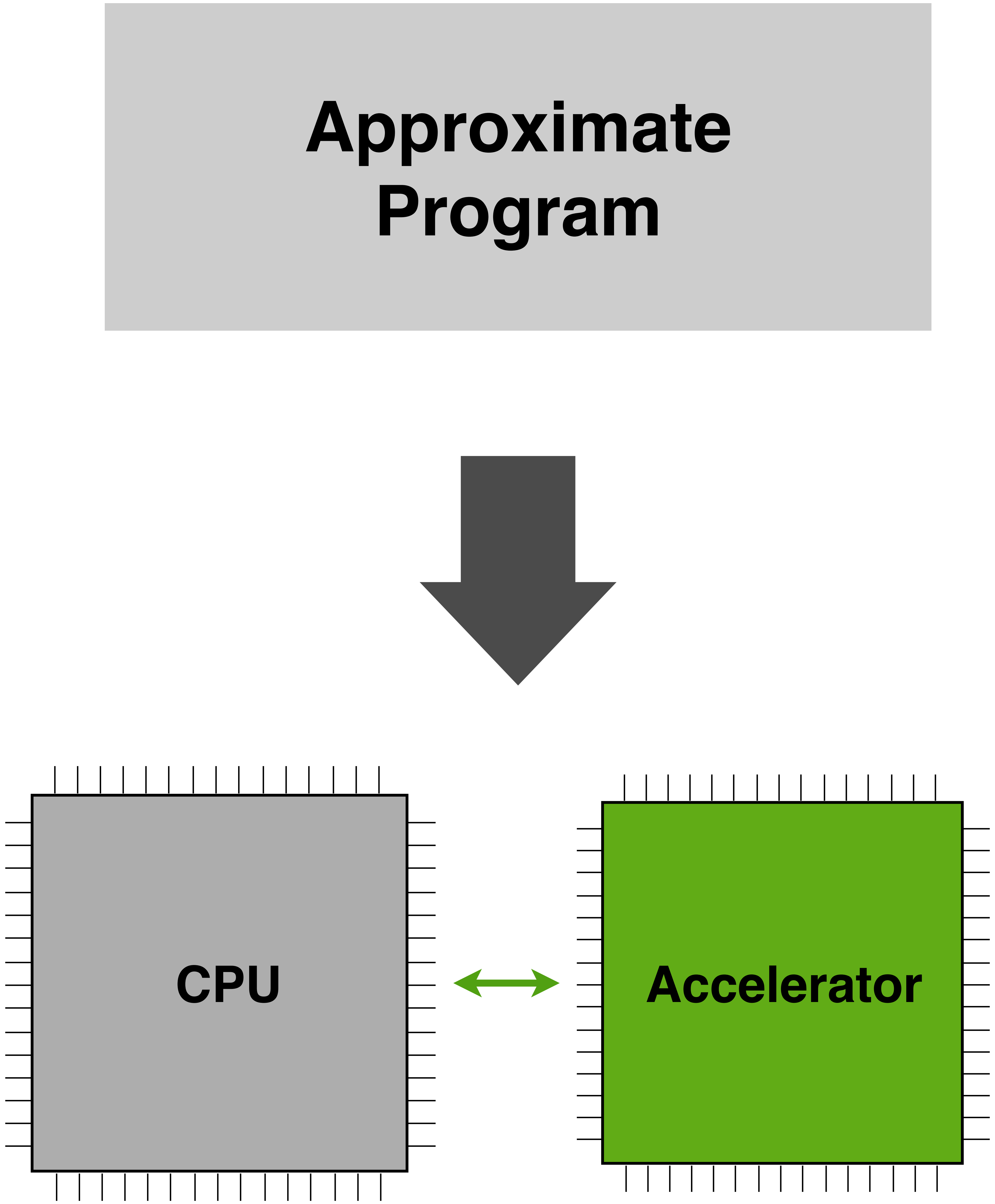
To improve responsiveness for such applications, this

tablets, and smart phones, which, despite relatively large chip area, have thermal budgets that are constrained by the poor heat dissipation of passive convection.

Processors today (and their heat sinks and energy delivery systems) are designed primarily for *sustained performance*. Although the focus on sustained performance is the right design choice for some applications (for example, batch-mode high-performance computing), many workloads are interactive in nature and thus demand *responsiveness*—how long does the user have to wait after initiating a command? In such settings, responsiveness may be more important than sustained


TEMPERATURE
HEAT
SOLID
LIQUID
GAS
MELTING/
FREEZING
VAPORIZING/
CONDENSING
PHASE-CHANGE
MATERIAL (PCM)
CPU

1. NEVER LISTEN TO ADVICE
2. ALWAYS LISTEN TO ADVICE
3. IT'S NOT NAÏVETÉ; IT'S <u>BEGINNER'S</u> <u>MIND</u>
4. THE BEST COMPUTER ARCHITECTURE RESEARCH ALWAYS "CHEATS"
5. DON'T DO FRAMEWORKS
6. START A BLOG
7. TALK TO YOUR FRIENDS

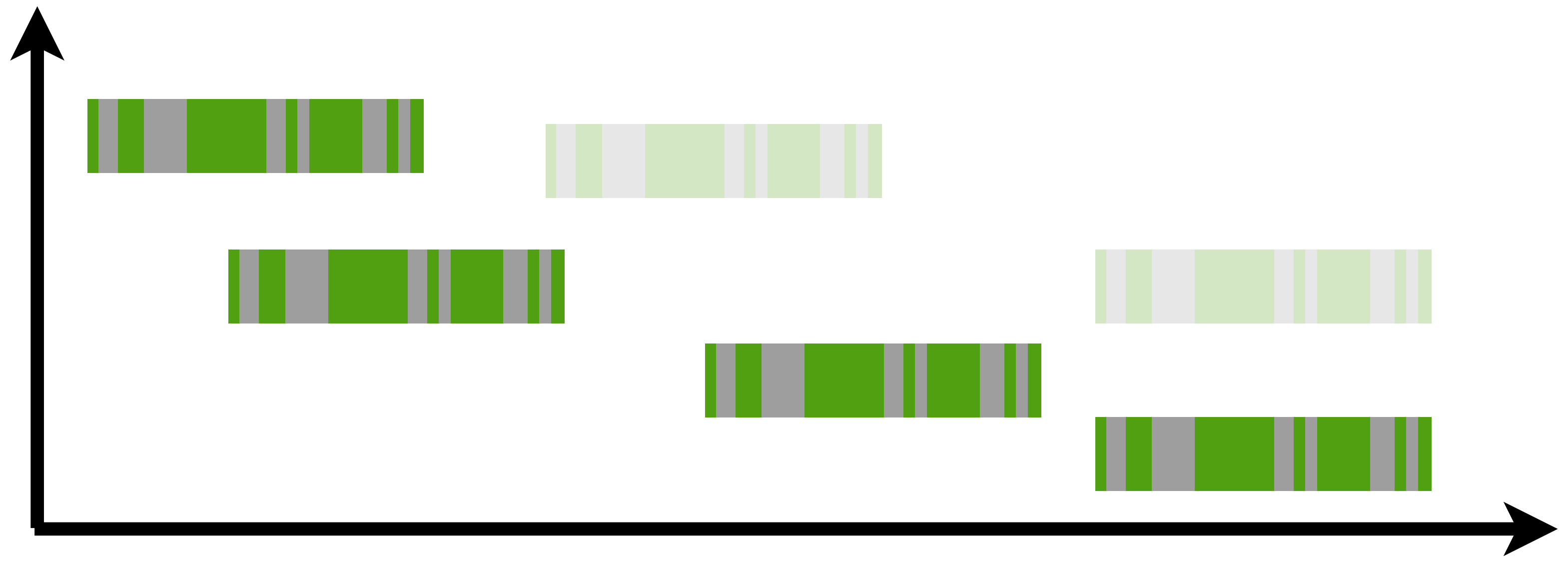

Approximate
Program
CPU
Accelerator

Approximate Program

Code with Annotations + $d(y, y')$ quality metric

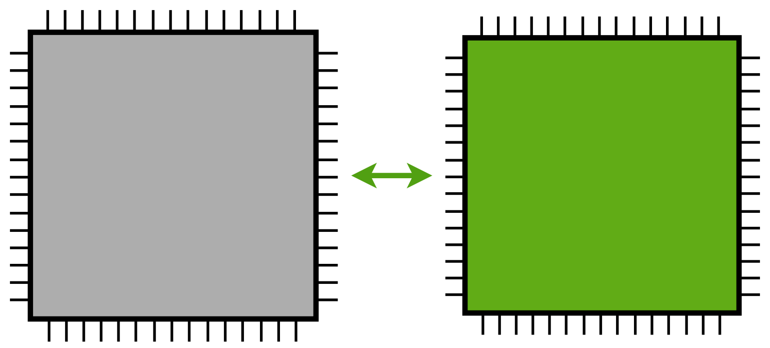
qualifiers for C & C++

interactive feedback



analysis & optimization

auto-tuning



FPGA accelerator

EnerJ type qualifiers for the Clang compiler

```
for (int k = k1; k < k2; k++) {
  APPROX float distance =
    dist(points->p[k],
         points->p[0]);
  ...
}
```

Optimization feedback loop

analysis library finds coarse-grain, safe-to-approximate regions

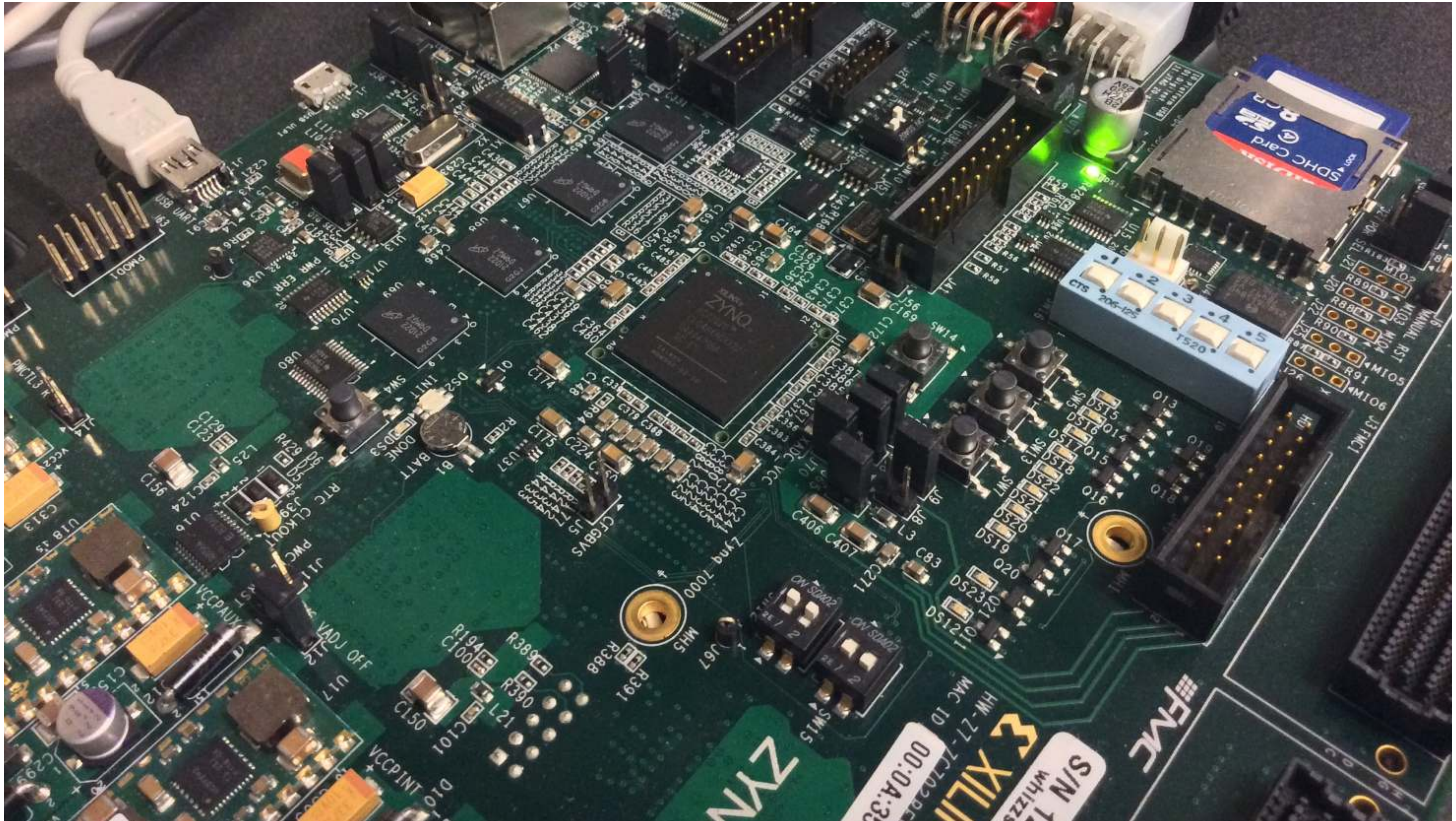
client optimizations use analysis to relax approximable code

```
for (int k = k1; k < k2; k++) {
  APPROX float distance =
    dist(points->p[k],
         points->p[0]);
  ...
}
```

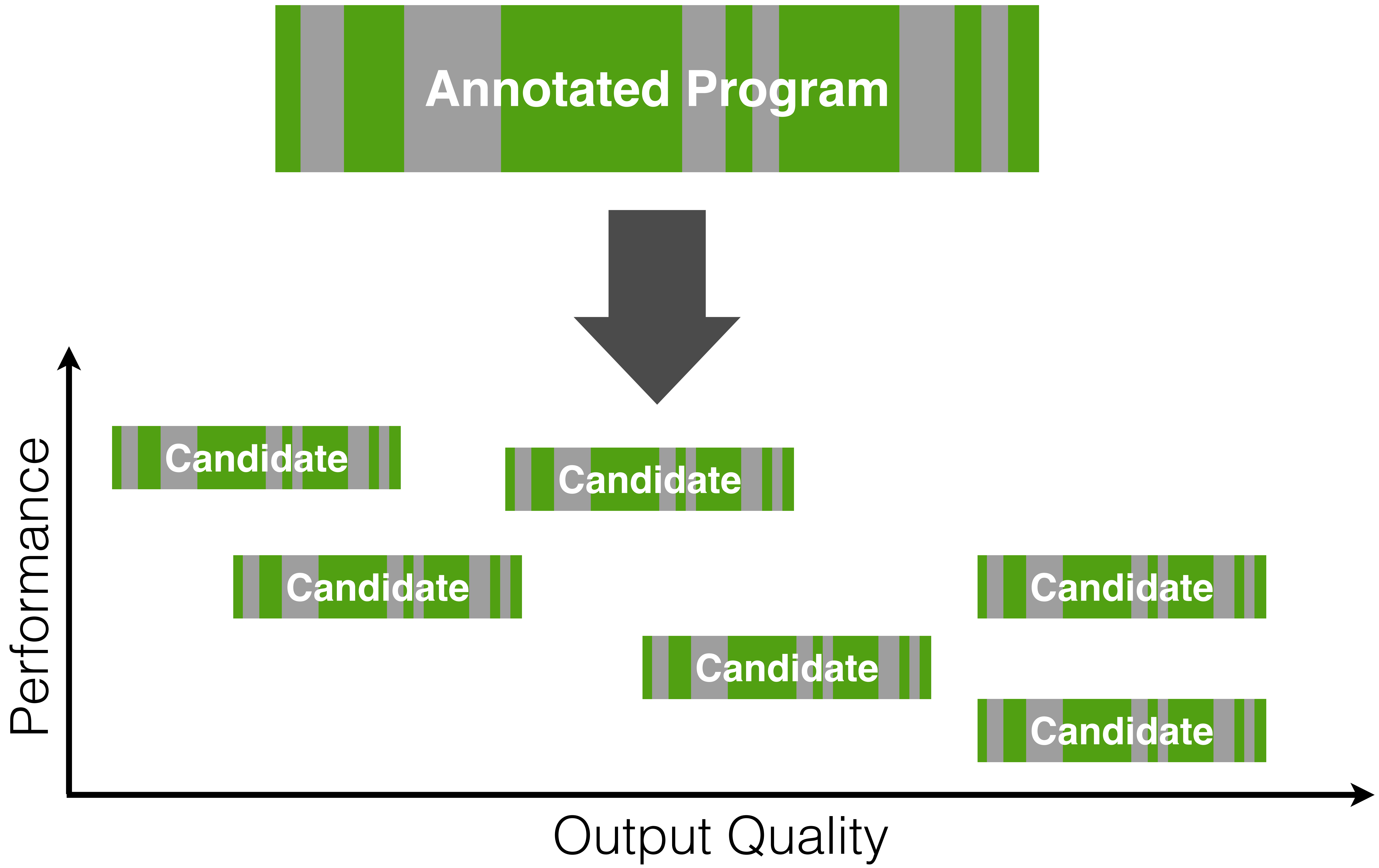
```
$ accept log
loop at streamcluster.cpp:651
can offload to NPU
$
```

reports tell developers where annotations are preventing optimization

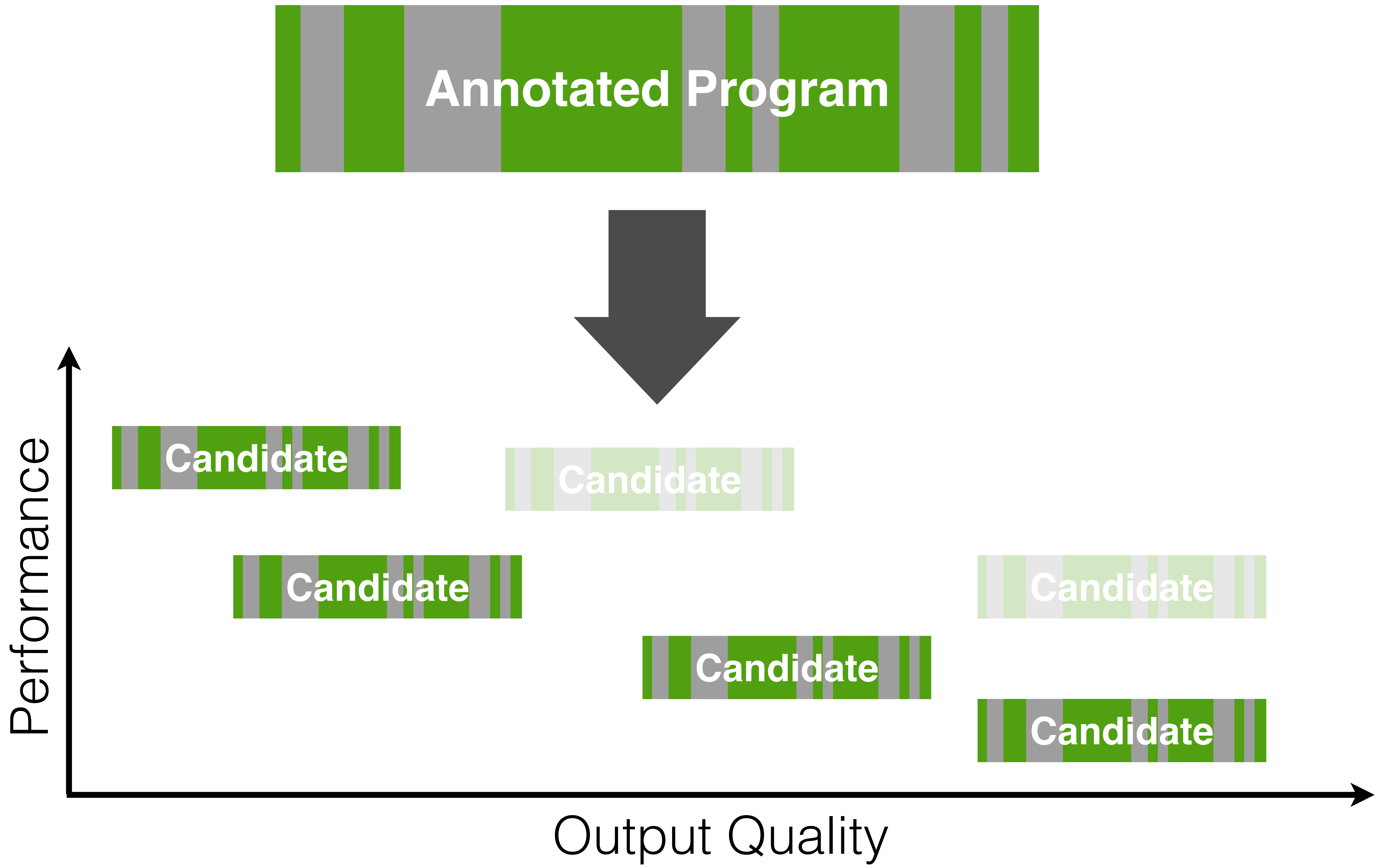
Neural acceleration on a commercial FPGA



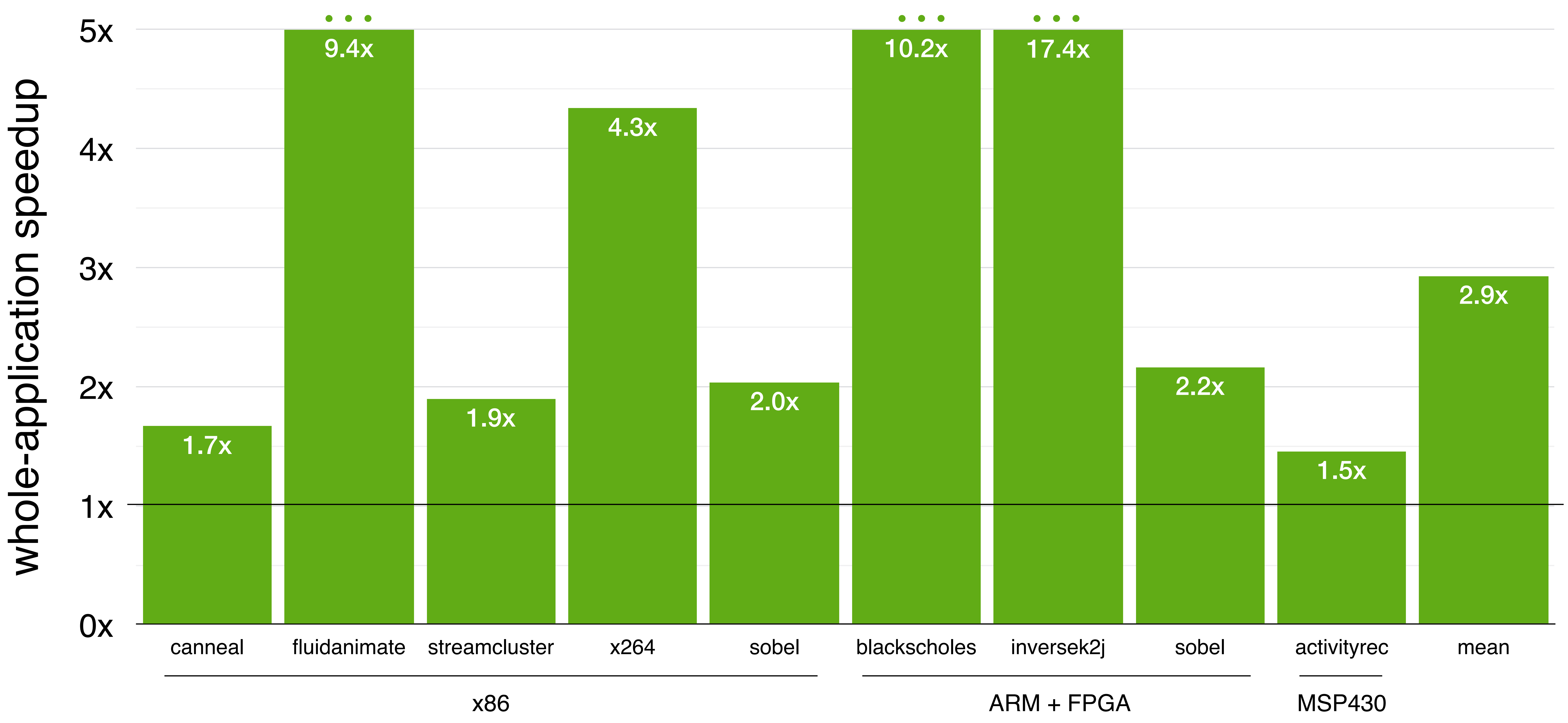
Auto-tuning for optimal trade-offs



Auto-tuning for optimal trade-offs



ACCEPT speedup over precise execution



Benchmarks are 2.9× faster on average with quality loss under 10%

dada.cs.washington.edu/research/tr/2015/01/UW-CSE-15-01-01.pdf

https://dada.cs.washington.edu/research/tr/2015/01/UW-CSE-15-01-01.pdf

ASPLOS
PLDI
ASPLOS
OOPSLA
CGO
ICS

ACCEPT: A Programmer-Guided Compiler Framework for Practical Approximate Computing

Adrian Sampson André Baixo Benjamin Ransford Thierry Moreau
Joshua Yip Luis Ceze Mark Oskin
University of Washington

Abstract

Approximate computing trades off accuracy for better performance and energy efficiency. It offers promising optimization

sion [Rubio-González et al. 2013]; using special low-power hardware structures that produce wrong results probabilistically [Esmaeilzadeh et al. 2012b; Liu et al. 2011]; and training hardware neural networks to mimic the behavior of costly

1. NEVER LISTEN TO ADVICE
2. ALWAYS LISTEN TO ADVICE
3. IT'S NOT NAÏVETÉ; IT'S BEGINNER'S MIND
4. THE BEST COMPUTER ARCHITECTURE RESEARCH ALWAYS "CHEATS"
5. DON'T DO FRAMEWORKS
6. START A BLOG
7. TALK TO YOUR FRIENDS

Adrian Sampson
Department of Computer Science
Cornell University

Closed Problems in Approximate Computing

OCTOBER 14, 2017

> These are notes for a talk I will give at the NOPE workshop at MICRO 2017, where the title is *Approximate Computing Is Dead; Long Live Approximate Computing*.

Approximate computing has reached an adolescent phase as a research area. We have picked bushels of low-hanging fruit. While there are many approximation papers left to write, it's a good time to enumerate the closed problems: research problems that are probably no longer worth pursuing.

Closed Problems in Approximate Hardware

No more approximate functional units. Especially for people who love VLSI work, a natural first step in approximate computing is to design approximate adders, multipliers, and other basic functional units. Cut a carry chain here, drop a block of intermediate results there, or use an automated search to find "unnecessary" gates—there are lots of ways to design an FU that's mostly right most of the time. Despite dozens of papers in this

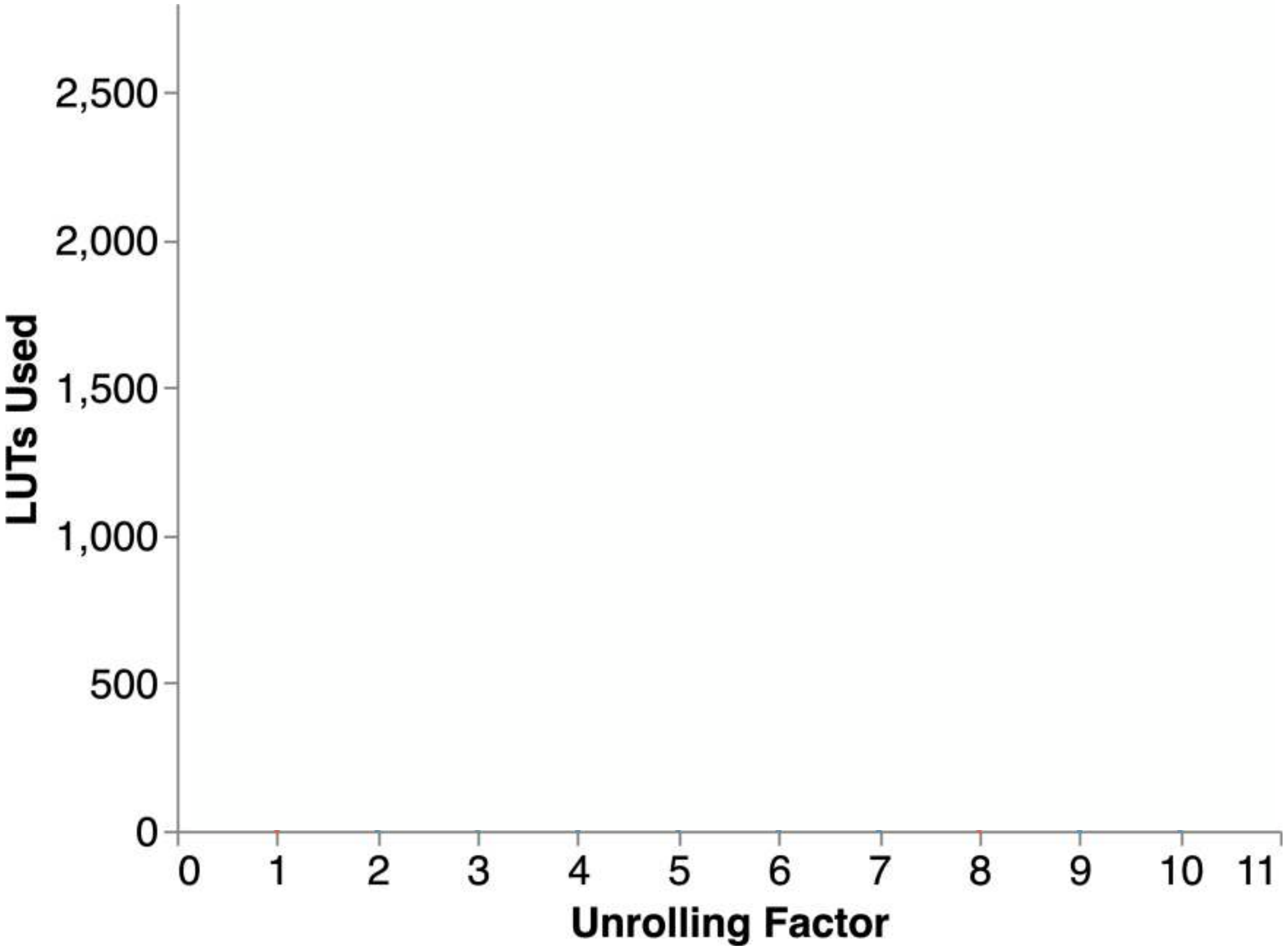
- PRACTICE WRITING
- CLARIFY YOUR OWN THOUGHTS
- SAVE TIME EXPLAINING STUFF
- "MARKET" YOUR WORK
- GET FAMOUS

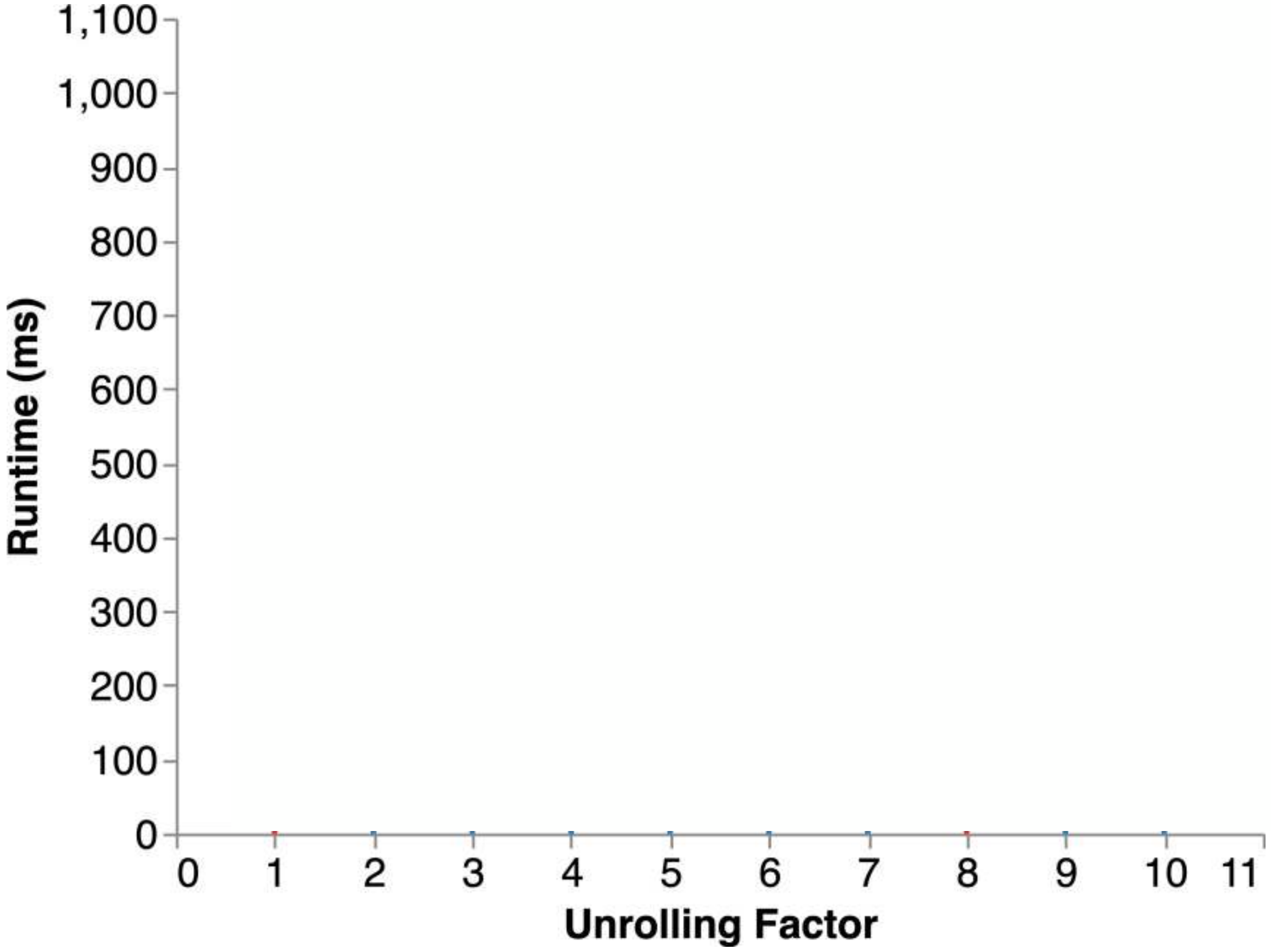
1. NEVER LISTEN TO ADVICE
2. ALWAYS LISTEN TO ADVICE
3. IT'S NOT NAÏVETÉ; IT'S <u>BEGINNER'S</u> <u>MIND</u>
4. THE BEST COMPUTER ARCHITECTURE RESEARCH ALWAYS "CHEATS"
5. DON'T DO FRAMEWORKS
6. START A BLOG
7. TALK TO YOUR FRIENDS


UCLA
Zhiru Zhang
Cornell

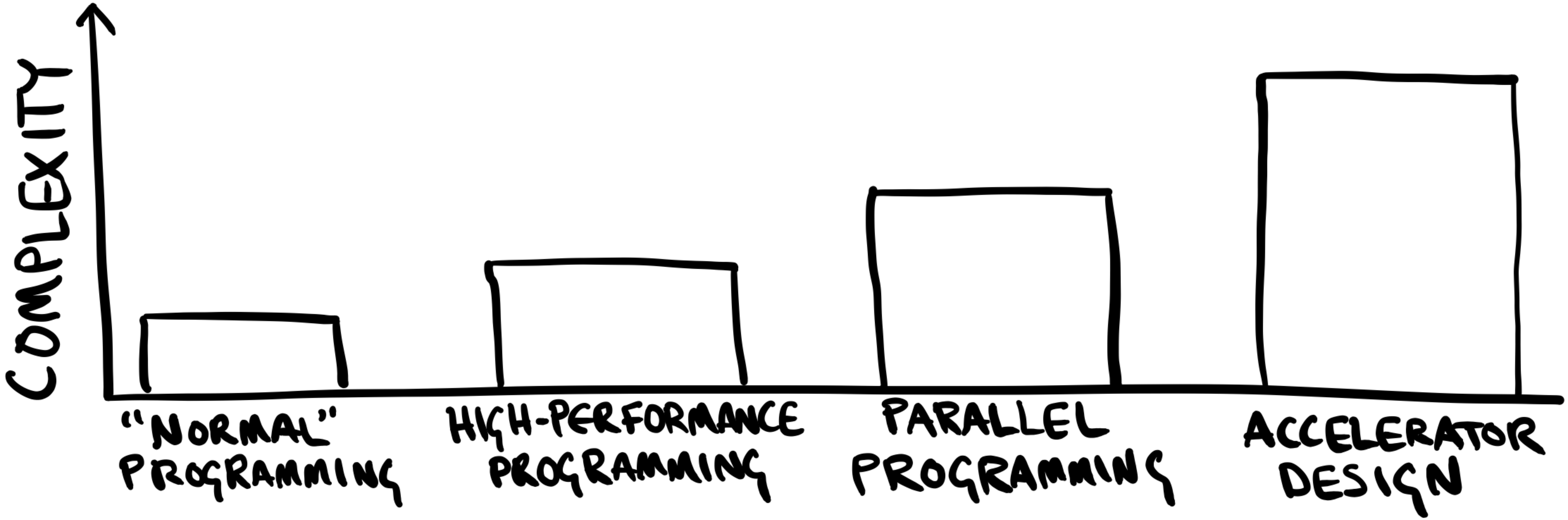
```
#pragma HLS ARRAY_PARTITION variable=m1 factor=3
#pragma HLS ARRAY_PARTITION variable=m2 factor=3
#pragma HLS ARRAY_PARTITION variable=prod factor=3
int m1[512][512];
int m2[512][512];
int prod[512][512];
for (int i = 0; i < 512; i++) {
  for (int j = 0; j < 512; j++) {
    int sum = 0;
    for (int k = 0; k < 512; k++) {
      #pragma HLS UNROLL factor=3
      sum += m1[i][k] * m2[k][j];
    }
    prod[i][j] = sum;
  }
}
```

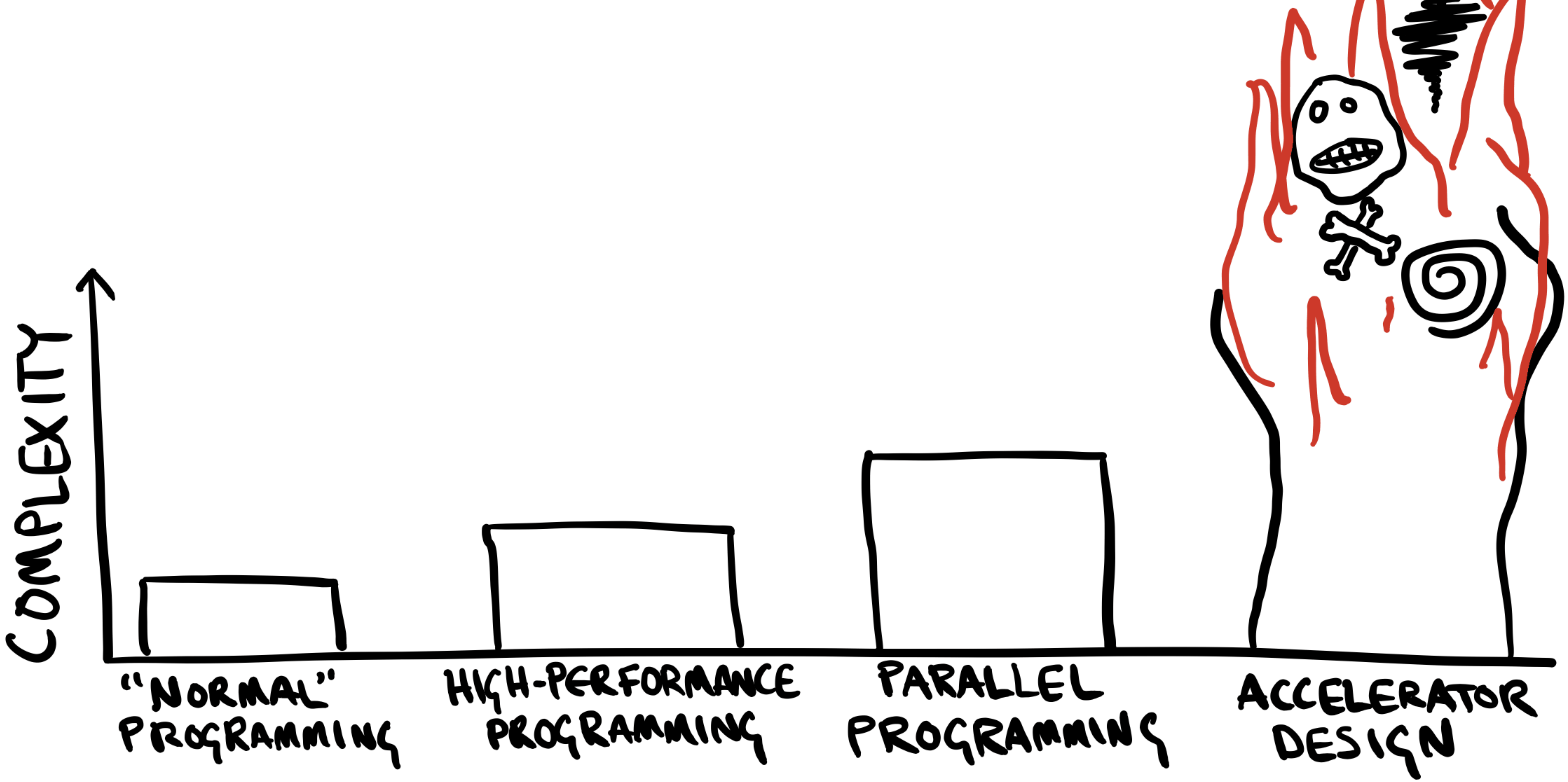
Area–Performance Trade-Offs in Unrolling

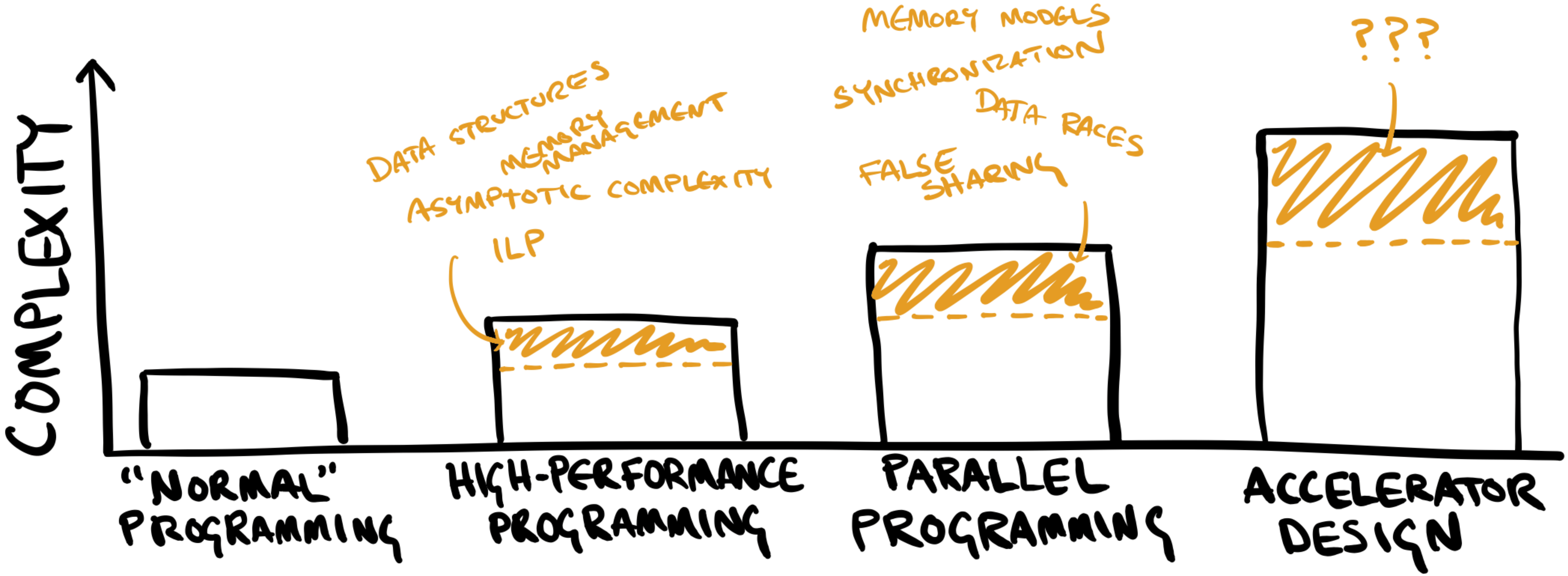


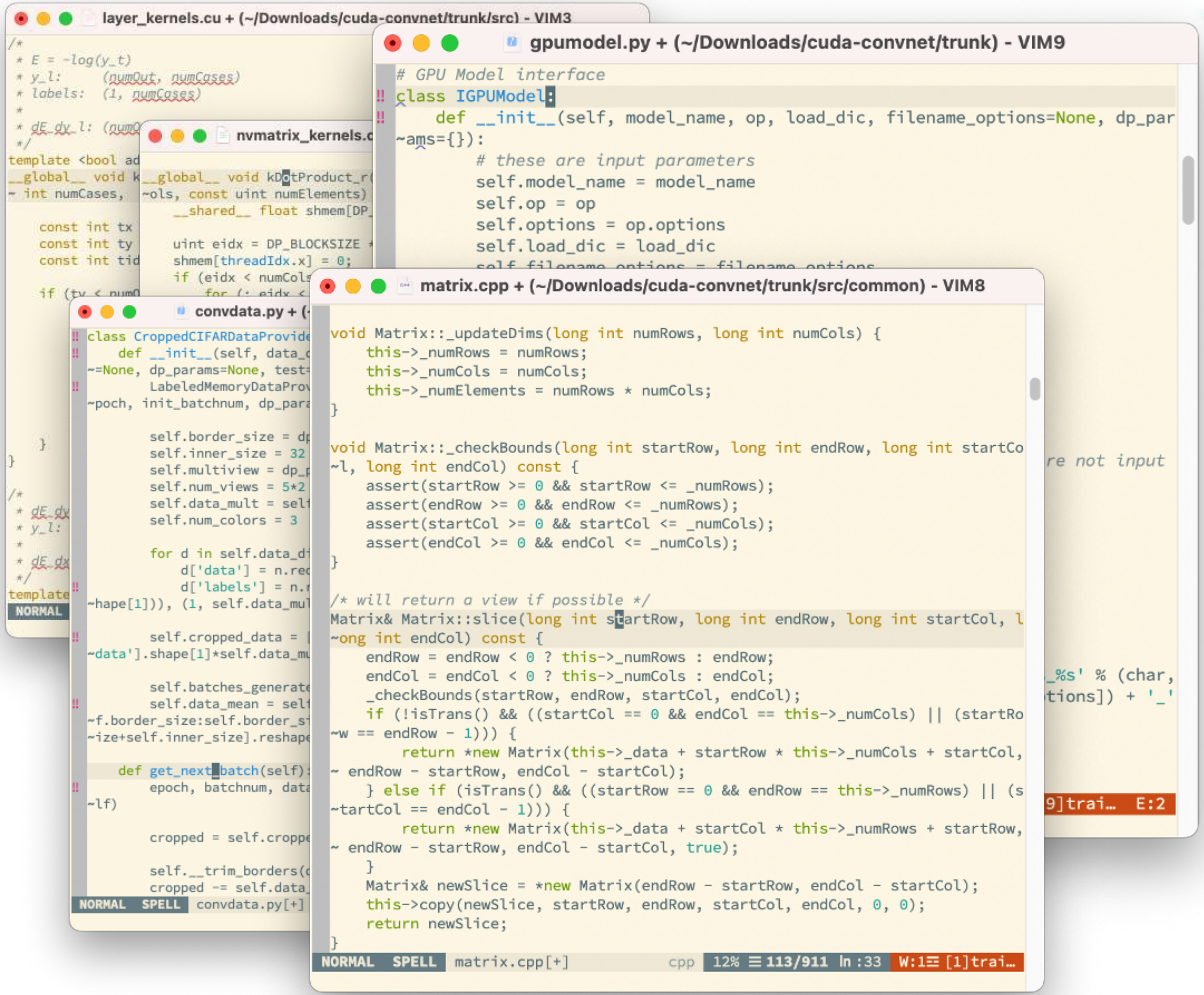


1. NEVER LISTEN TO ADVICE
2. ALWAYS LISTEN TO ADVICE
3. IT'S NOT NAÏVETÉ; IT'S <u>BEGINNER'S</u> <u>MIND</u>
4. THE BEST COMPUTER ARCHITECTURE RESEARCH ALWAYS "CHEATS"
5. DON'T DO FRAMEWORKS
6. START A BLOG
7. TALK TO YOUR FRIENDS


COMPLEXITY
"NORMAL" PROGRAMMING
HIGH-PERFORMANCE PROGRAMMING
PARALLEL PROGRAMMING
ACCELERATOR DESIGN


COMPLEXITY
"NORMAL" PROGRAMMING
HIGH-PERFORMANCE PROGRAMMING
PARALLEL PROGRAMMING
ACCELERATOR DESIGN


COMPLEXITY
DATA STRUCTURES
MEMORY MANAGEMENT
ASYMPTOTIC COMPLEXITY
ILP
MEMORY MODELS
SYNCHRONIZATION
DATA RACES
FALSE SHARING
???
"NORMAL" PROGRAMMING
HIGH-PERFORMANCE PROGRAMMING
PARALLEL PROGRAMMING
ACCELERATOR DESIGN




```python
class AlexNet(nn.Module):

    def __init__(self, num_classes: int = 1000) -> None:
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

AlexNet (2012)

CUDA: 12k lines
Python: 2k lines
C++: 1k lines

AlexNet (2024)

PyTorch: 35 lines

assembly : :: CUDA :

assembly : :: HDLs : **Accelerator Design Languages**

Spatial

[Koeplinger et al., PLDI 2018]

HeteroCL

[Lai et al., FPGA 2019]

assembly : JS OCaml :: HDLs :

Aetherling

[Durst et al., PLDI 2020]

Dahlia

[Nigam et al., PLDI 2020]

Academic ADLs

debuggers [Berlstein et al., ASPLOS 2023]
profilers
module systems

assembly : :: HDLs :

rethinking the system stack
for reconfigurable hardware (FPGAs)
[Vega et al., PLDI 2021]

type systems for
modular reasoning
[Nigam et al., PLDI 2023]

1. NEVER LISTEN TO ADVICE
2. ALWAYS LISTEN TO ADVICE
3. IT'S NOT NAÏVETÉ; IT'S <u>BEGINNER'S</u> <u>MIND</u>
4. THE BEST COMPUTER ARCHITECTURE RESEARCH ALWAYS "CHEATS"
5. DON'T DO FRAMEWORKS
6. START A BLOG
7. TALK TO YOUR FRIENDS