Catapult the Masses

James Larus
EPFL

Reconfigurable Computing for the Masses, Really?

A Workshop at and after FPL'15, **4th September 2015**

My Perspective

- Masses == software developers
- Reconfigurable computing == FPGA

- Can SDEs program FPGAs without learning HW design or getting an EE degree?

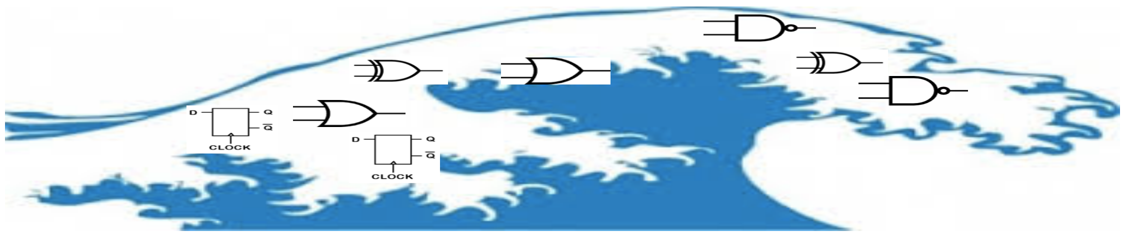
- Can high-level <u>programming</u> languages be compiled down to FPGAs?
  - Not hardware description languages

- Can reconfigurable computing be made as easy as GPU programming?

Semantic Gap

?

MIND THE GAP



Alternative View of Microsoft Catapult System


A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services

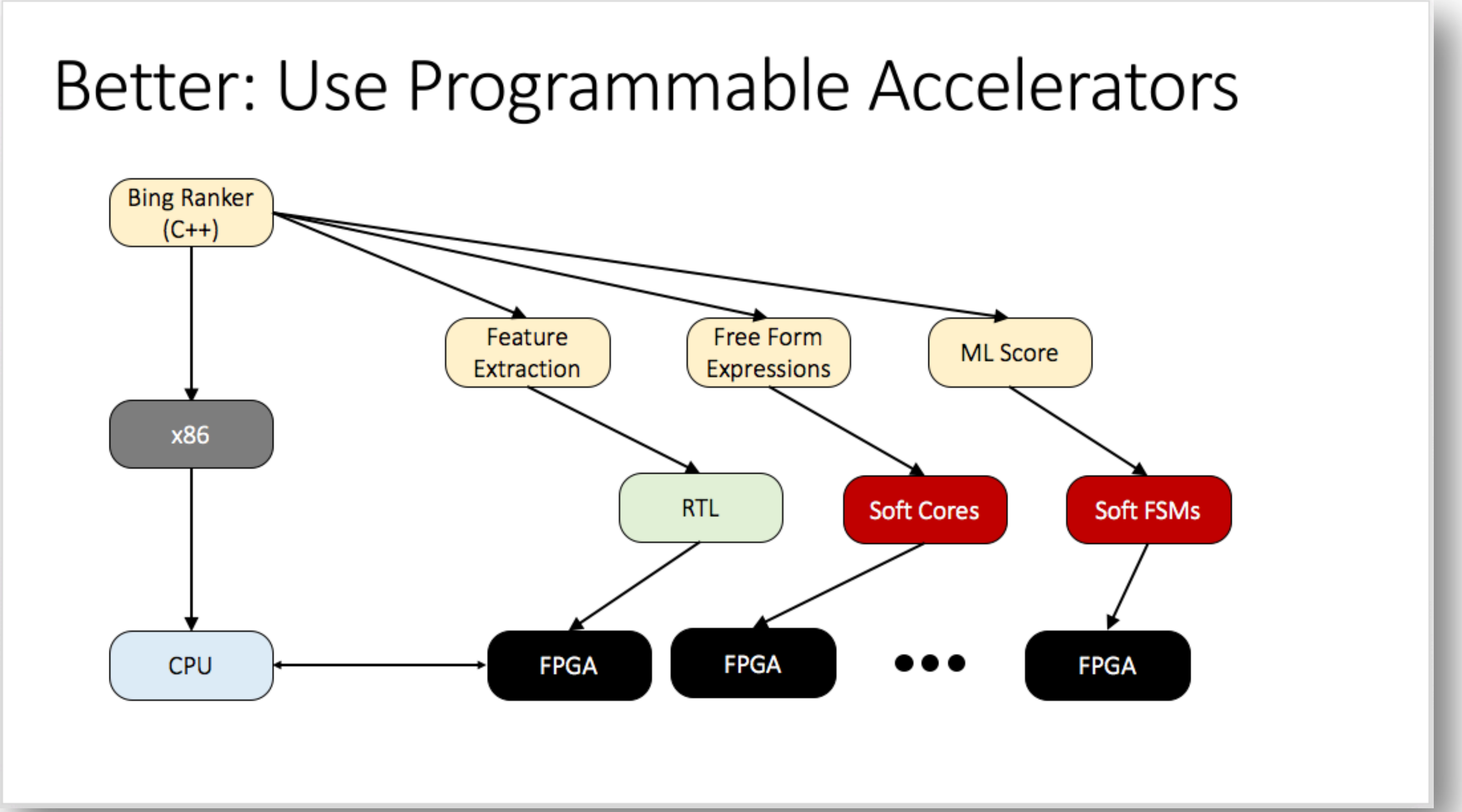
Microsoft

Abstract

1. Introduction

- Another way of telling the same story
- Design principles from this story suggest an alternative approach

Microsoft Catapult



Eric Chung, **Programming a Reconfigurable Fabric for Large Scale Datacenter Services**

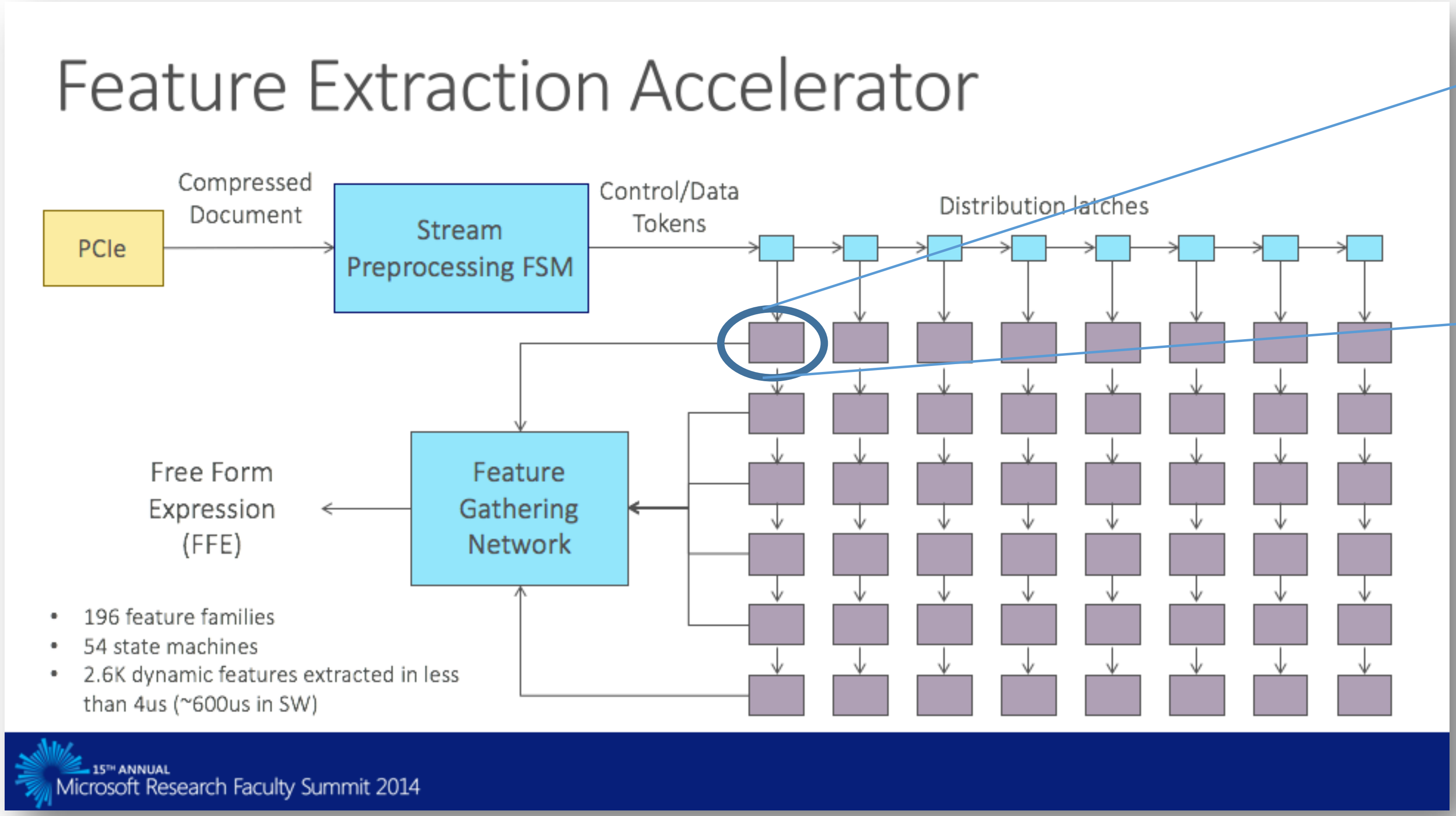
Accelerators == Non von Neumann Computers (NonvoN)

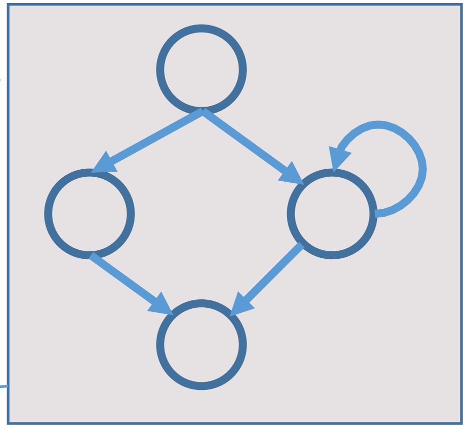
- Massively parallel
- Not general purpose
  - Not Turing complete (non-Turing)
- Instructions != data

Catapult

- Simple to program directly from Bing language model
- Quickly reprogrammable as search model evolved
- “Easy” to implement
- High throughput at low clock speed

Catapult Feature Extractor

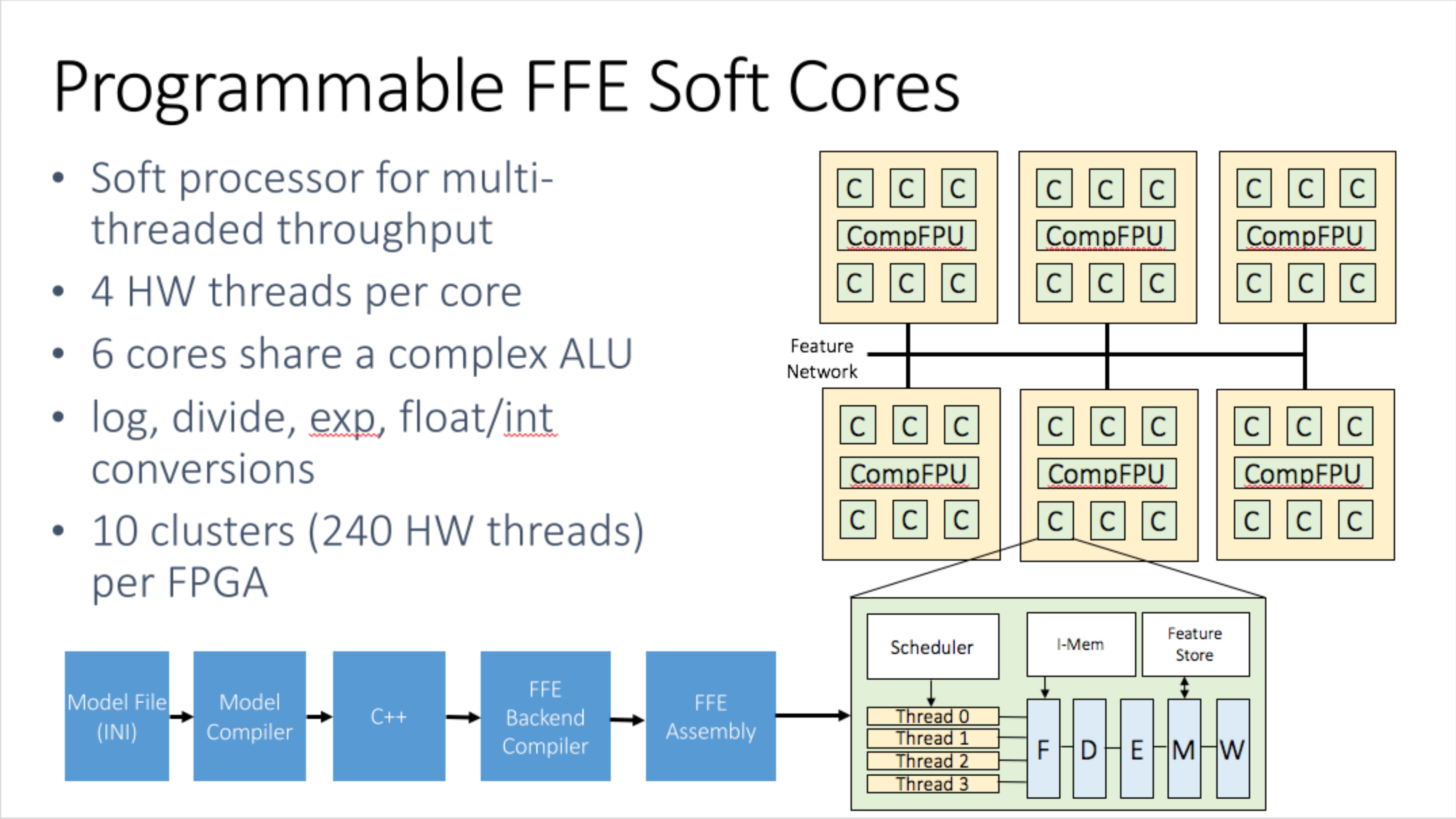




- 196 feature families
- 54 state machines
- 2.6K dynamic features extracted in less than 4us (~600us in SW)

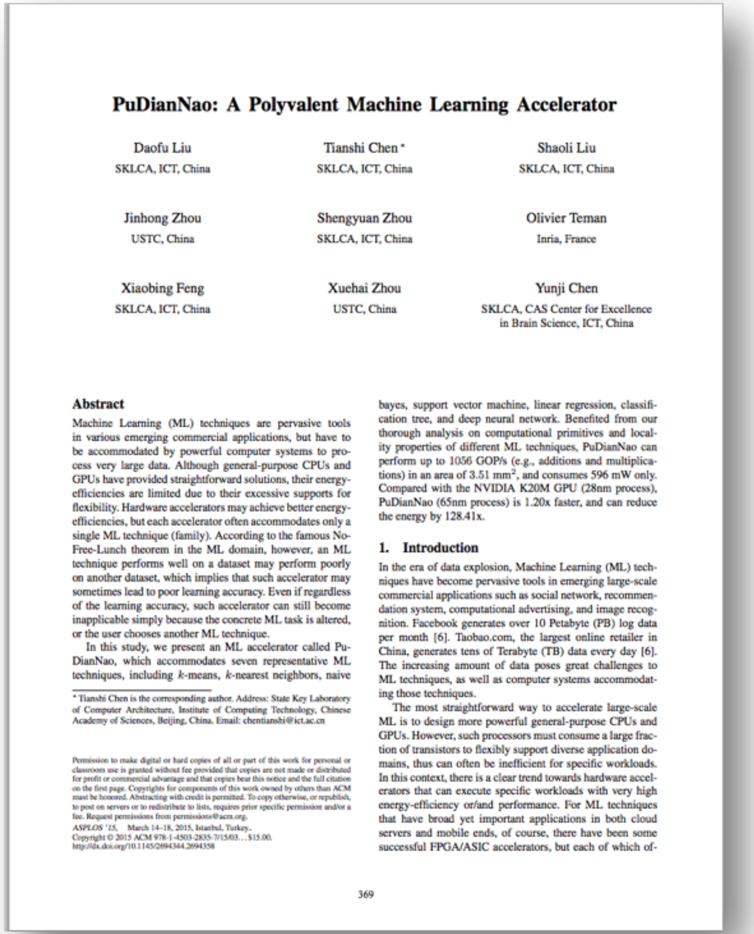
Adrian Caulfield, **A Reconfigurable Fabric for Large Scale Datacenter Services**

Catapult Free-Form Expressions



Eric Chung, **Programming a Reconfigurable Fabric for Large Scale Datacenter Services**

Catapult Scoring Model



PuDianNao: A Polyvalent Machine Learning Accelerator

Daofu Liu
SKLCA, ICT, China

Tianshi Chen*
SKLCA, ICT, China

Shaoli Liu
SKLCA, ICT, China

Jinhong Zhou
USTC, China

Shengyuan Zhou
SKLCA, ICT, China

Olivier Teman
Inria, France

Xiaobing Feng
SKLCA, ICT, China

Xuehai Zhou
USTC, China

Yunji Chen
SKLCA, CAS Center for Excellence in Brain Science, ICT, China

Abstract

Machine Learning (ML) techniques are pervasive tools in various emerging commercial applications, but have to be accommodated by powerful computer systems to process very large data. Although general-purpose CPUs and GPUs have provided straightforward solutions, their energy-efficiencies are limited due to their excessive supports for flexibility. Hardware accelerators may achieve better energy-efficiencies, but each accelerator often accommodates only a single ML technique (family). According to the famous No-Free-Lunch theorem in the ML domain, however, an ML technique performs well on a dataset may perform poorly on another dataset, which implies that such accelerator may sometimes lead to poor learning accuracy. Even if regardless of the learning accuracy, such accelerator can still become inapplicable simply because the concrete ML task is altered, or the user chooses another ML technique.

In this study, we present an ML accelerator called PuDianNao, which accommodates seven representative ML techniques, including *k*-means, *k*-nearest neighbors, naive bayes, support vector machine, linear regression, classification tree, and deep neural network. Benefited from our thorough analysis on computational primitives and locality properties of different ML techniques, PuDianNao can perform up to 1056 GOP/s (e.g., additions and multiplications) in an area of 3.51 mm^2, and consumes 596 mW only. Compared with the NVIDIA K20M GPU (28nm process), PuDianNao (65nm process) is 1.20x faster, and can reduce the energy by 128.41x.

1. Introduction

In the era of data explosion, Machine Learning (ML) techniques have become pervasive tools in emerging large-scale commercial applications such as social network, recommendation system, computational advertising, and image recognition. Facebook generates over 10 Petabyte (PB) log data per month [6]. Taobao.com, the largest online retailer in China, generates tens of Terabyte (TB) data every day [6]. The increasing amount of data poses great challenges to ML techniques, as well as computer systems accommodating those techniques.

The most straightforward way to accelerate large-scale ML is to design more powerful general-purpose CPUs and GPUs. However, such processors must consume a large fraction of transistors to flexibly support diverse application domains, thus can often be inefficient for specific workloads. In this context, there is a clear trend towards hardware accelerators that can execute specific workloads with very high energy-efficiency or/and performance. For ML techniques that have broad yet important applications in both cloud servers and mobile ends, of course, there have been some successful FPGA/ASIC accelerators, but each of which of-

* Tianshi Chen is the corresponding author. Address: State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. Email: chentianshi@ict.ac.cn

ASPLOS '15, March 14–18, 2015, Istanbul, Turkey..
Copyright © 2015 ACM 978-1-4503-2835-7/15/03…$15.00.
http://dx.doi.org/10.1145/2694344.2694358

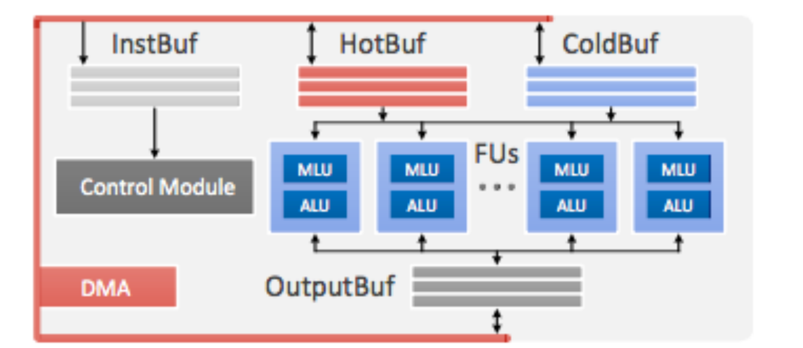


Figure 11: Accelerator architecture of PuDianNao.

Seven representative ML techniques

- k-means
- k-nearest neighbors
- naive bayes
- support vector machine
- linear regression
- classification tree
- deep neural network

Liu, et al., PuDianNao: A Polyvalent Machine Learning Accelerator, ASPLOS 2015

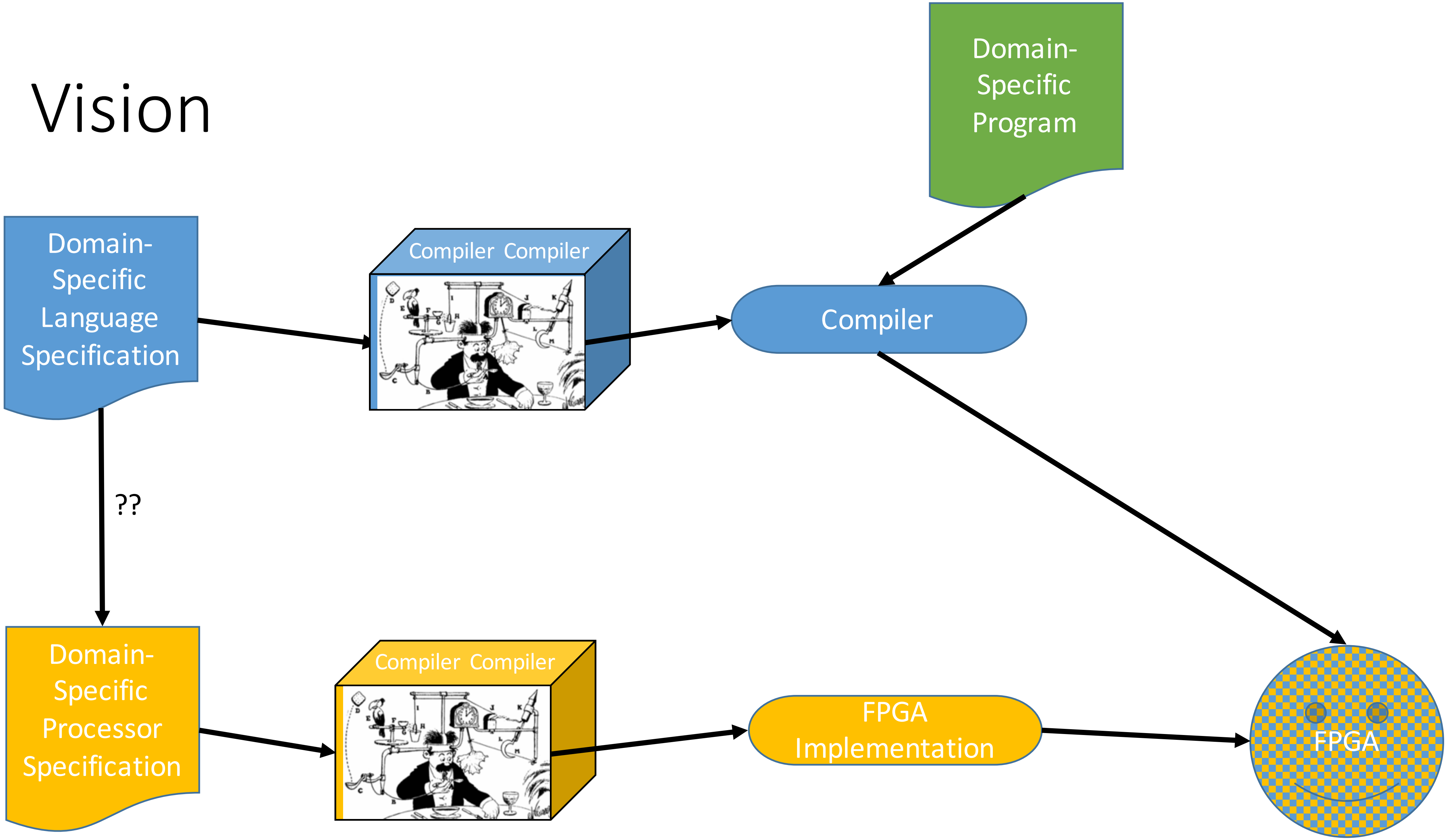
Why Was NonvoN Architecture a Good Idea?

- Small compiler-HW semantic gap
  - Some compilers (SM) could have been perl scripts
  - Others (FFE) were sophisticated (~llvm) compilers
- HW was easy to get right and to extend
  - Simple, regular, modular
  - Can track software evolution
- Computations were fine-grain parallel, HW effective at exploiting
- Easy to compose computations in a pipeline
- “Soft” programmability for alternative language models
  - < 200 ms

Limitations

- Lack of generality
  - Will not work as well when
    - No parallel implementation
    - Complex HW (eg GPU)
    - Too sophisticated compilation / programming model (eg GPU)
- Interpretation overhead
  - Probably could do better with ‘pure’ HW implementation
  - But, Bing language models change every 3 months
- Still requires HW designer to implement processors
  - One-time expense, primitives change rarely
  - More importantly, another topic for research

Vision


Domain-Specific Program
Domain-Specific Language Specification
Compiler Compiler
Compiler
??
Domain-Specific Processor Specification
Compiler Compiler
FPGA Implementation
FPGA

Open Problems

- High-level description of domain-specific languages (DSL)
  - Currently, DSL (mostly) described by imperative implementation
- Declarative techniques for implementing HL DSLs
  - Current, DSL implemented by writing compiler and optimizer (using framework)
- High-level description of domain-specific processors (DSProc)
  - Processor description is an old idea. Time to revive?
  - Possible to derive DSProc from DSL?
- Techniques for implementation HL DSProcs
  - Processor compiler?
- Methodology for analyzing domain, designing DSL, co-designing DSProc

LMS: Program Generation and Embedded Compilers in Scala

- Used to build DSL like Delite, Spiral, LegoBase
  - DSLs are concise and expressive
  - Constructing a DSL is still complex and requires compiler expertise
- Type-directed meta/macro programming

```
var n: Double = 0.0
var i: Int = 0
val end = data.length
while (i < end) {
  val x = data(i)
  val c = x > 0
  if (c) n += x }
println(n)
```

Putting on Compiler Hat

- High-level description of domain-specific languages ✔
- Declarative techniques for implementing HL DSLs
- High-level description of domain-specific processors
- Techniques for implementation HL DSProcs

Programmer's
~~Compiler Writer's~~ Nightmare