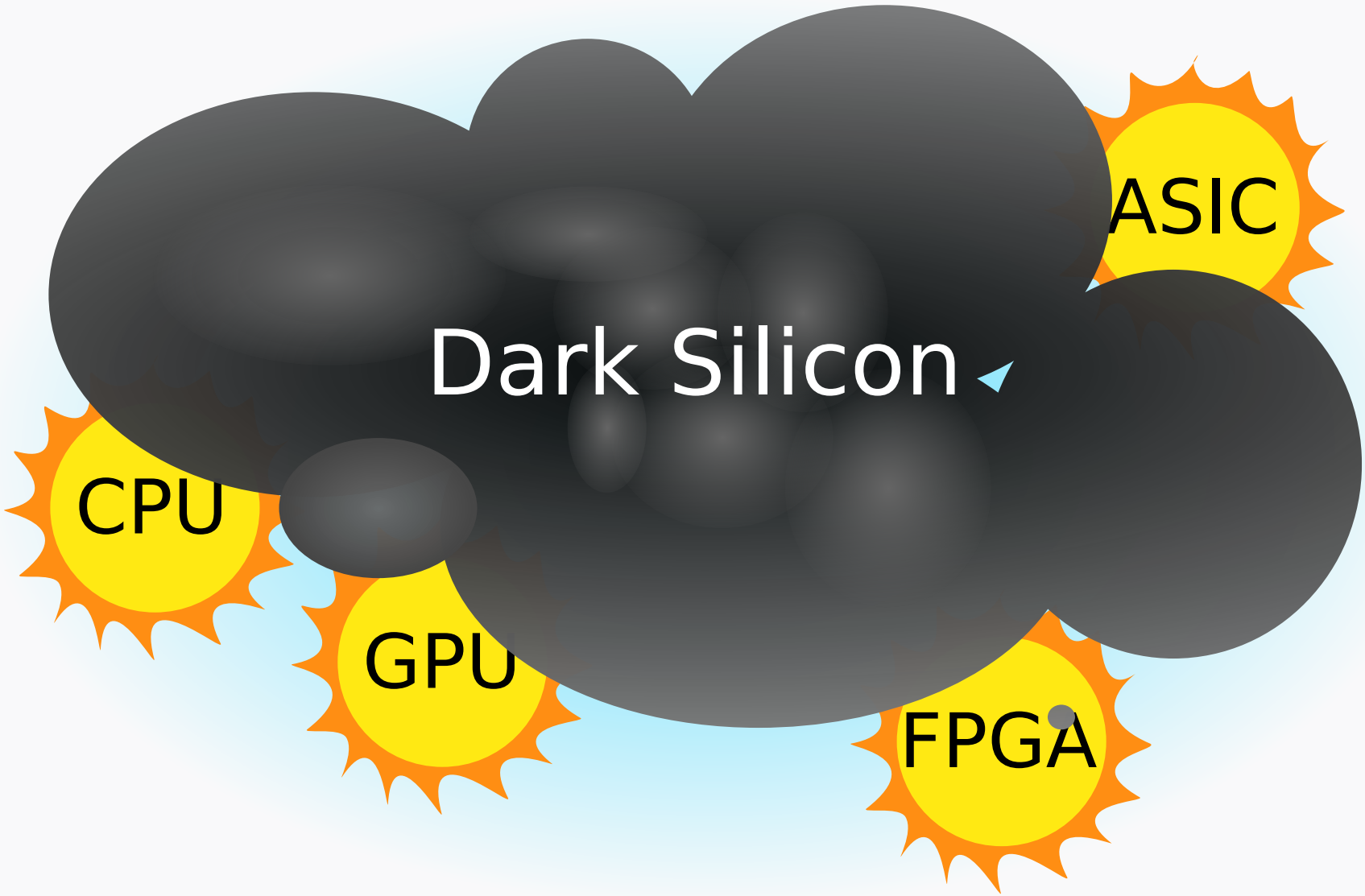
Designing an adaptive VM that combines vectorized and JIT execution on heterogeneous hardware

Tim Gubner

ICDE PhD Symposium, 2018

Modern hardware vs. data processing systems



State of the art

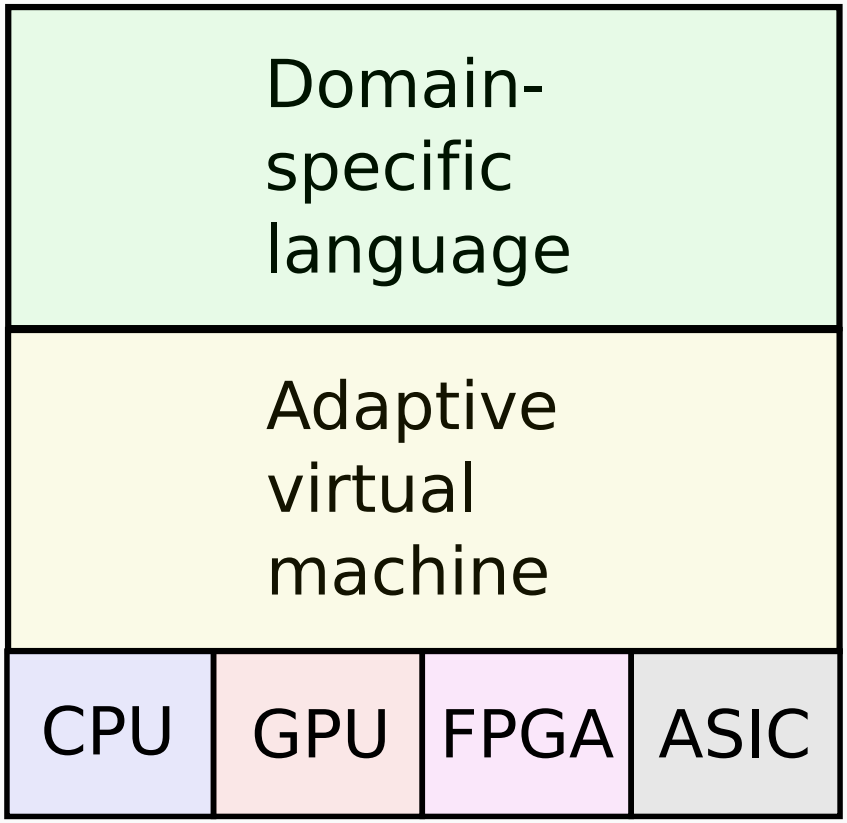
System	CPU	GPU	FGPA	ASIC
MonetDB	✓			
doppioDB	✓		✓	
Ocelot	✓	✓		
HyPer	✓			
MapD	✓	✓		
CoGaDB	✓	✓		
TensorFlow	✓	✓	?	✓

Goal

One system to rule them all

One system to bring them, and in Dark Silicon bind them

Idea


Domain-
specific
language
Adaptive
virtual
machine
CPU
GPU
FPGA
ASIC

Virtual Machine

Compile or not to compile

- Compilation is time consuming ($\geq$ 20 ms [1])
- Also noticeable in HyPer [2]

- Compilers make assumptions.
  Resulting code either:
  - Static and concise
  - Dynamic and bulky (code explosion)

[1] Using LLVM C++ API and optimization passes
[2] Kohn et al. "Adaptive Execution of Compiled Queries", ICDE 2018

Compile or not to compile

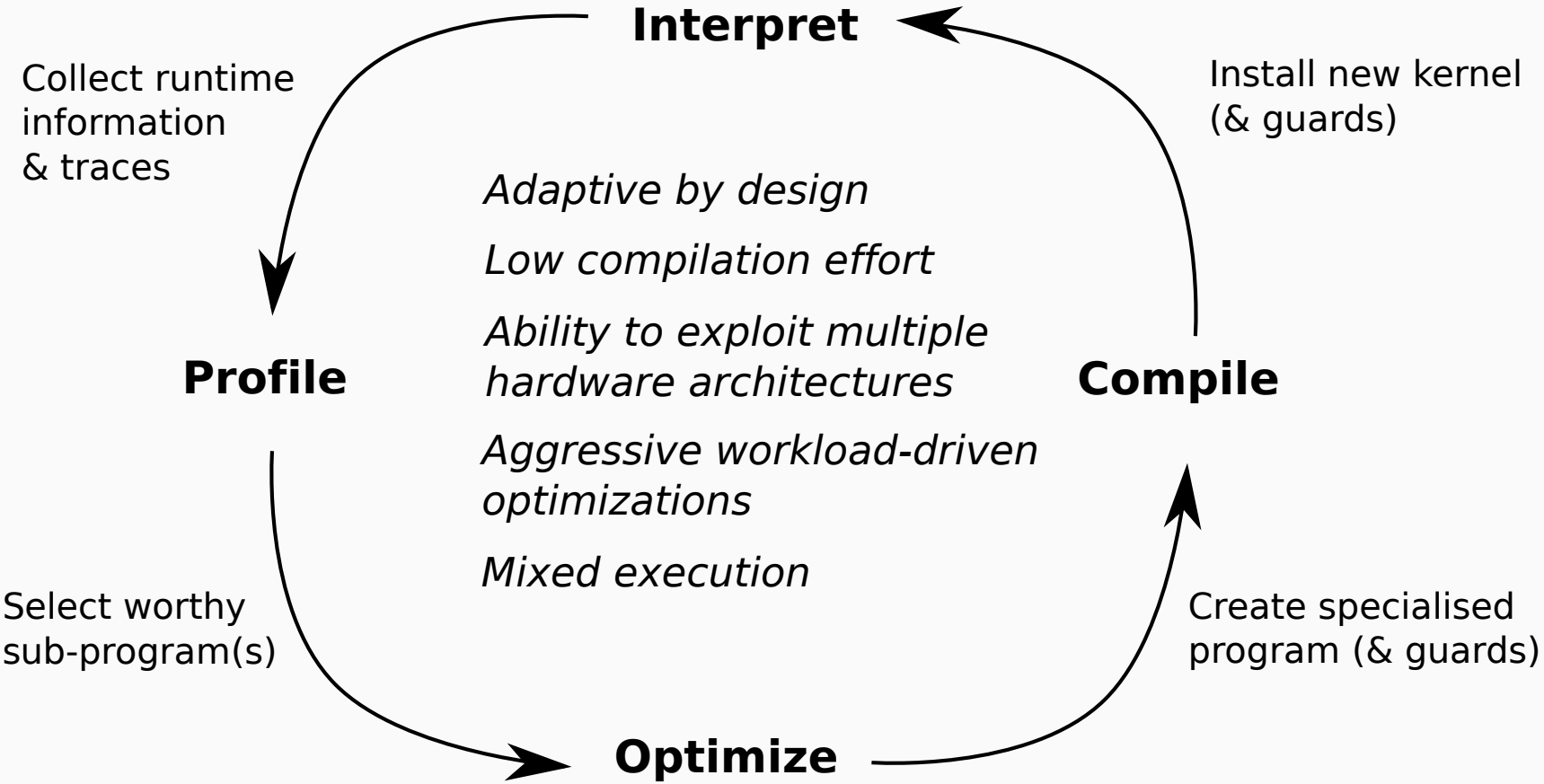
- Compilation is time consuming ($\geq$ 20 ms [1])
- Also noticeable in HyPer [2]

- Compilers make assumptions.
  Resulting code either:
  - Static and concise
  - Dynamic and bulky (code explosion)

Why would we ALWAYS want to compile EVERYTHING?

[1] Using LLVM C++ API and optimization passes
[2] Kohn et al. "Adaptive Execution of Compiled Queries", ICDE 2018

(Real) JIT-compilation



Domain-Specific Language

The seek for the right level of abstraction

Low enough

- Micro-adaptivity [3]
- Efficient interpretation
- JIT / incremental compilation

[3] Răducanu et al. "Micro adaptivity in Vectorwise", SIGMOD 2013

The seek for the right level of abstraction

CWI
Database Architectures

Low enough

- Micro-adaptivity [3]
- Efficient interpretation
- JIT / incremental compilation

High enough

- Effcient execution on multiple devices
- Macro-adaptivity: e.g. reorder operations

[3] Răducanu et al. "Micro adaptivity in Vectorwise", SIGMOD 2013

The seek for the right level of abstraction

CWI
Database Architectures

Low enough

- Micro-adaptivity [3]
- Efficient interpretation
- JIT / incremental compilation

High enough

- Effcient execution on multiple devices
- Macro-adaptivity: e.g. reorder operations

Goal

Relation algebra $\rightarrow$ **?** $\rightarrow$ Assembly, OpenCL ...

[3] Răducanu et al. "Micro adaptivity in Vectorwise", SIGMOD 2013

Why (another) DSL?

Relational algebra

Too high-level

(Scalar) Monad/Monoid comprehension

Weld [4], MRQL [5]

High-level but per-tuple transformations lose information

[4] S. Palkar et al. "Weld: A Common Runtime for High Performance Data Analytics", CIDR 2017

[5] Fegaras, L. "An Algebra for Distributed Big Data Analytics", 2016

Why (another) DSL?

CWI
Database Architectures

C alikes

OpenCL, CUDA, Intel SPMD (ispc) ...

Too low-level

MonetDB assembly language

Heavily data-parallel, too low-level

Our vision

Data-parallelism as first-class citizen

- Data-parallel skeletons/patterns
  Specialized operations on chunks of data
  For example: `map`, `filter`, `scatter`, `gather` ...
- Lambda functions
- Immutable variables for intermediates (Static single assignment form)
- Mutable variables for remaining state
- Partially typed ($a \in$ `DECIMAL(6,2)` instead of $a \in$ `int64_t`)

Skeletons

Op.	`map`	`filter`	`scatter`	`gather`	`ht_ins`	`merge`
π	✓					
σ	✓	✓				
$\bowtie_{\text{Hash}}$	✓	✓	✓	✓	✓	
G_{Hash}	✓	✓	✓	✓	✓	
$\cup_{\text{Hash}}$	✓	✓	✓	✓	✓	
$\bowtie_{\text{Merge}}$			✓	✓		✓
Sort			✓	(✓)		✓

Skeletons themselves do not need to be implemented data-parallel (e.g. `ht_ins`)...

Example

CWI Database Architectures

```
mut i
mut k
i := 0
k := 0
loop
  let input = read i some_data in
    let a = map (\x -> 2*x) input in
      let t = filter (\x -> x>0) a in
        let b = condense t
          write x i a
          write y k b
          i := i + len(a)
          k := k + len(b)

  if i >= 4096 then
    break
```

Plan

Plan

Base framework

DSL, vectorized interpreter

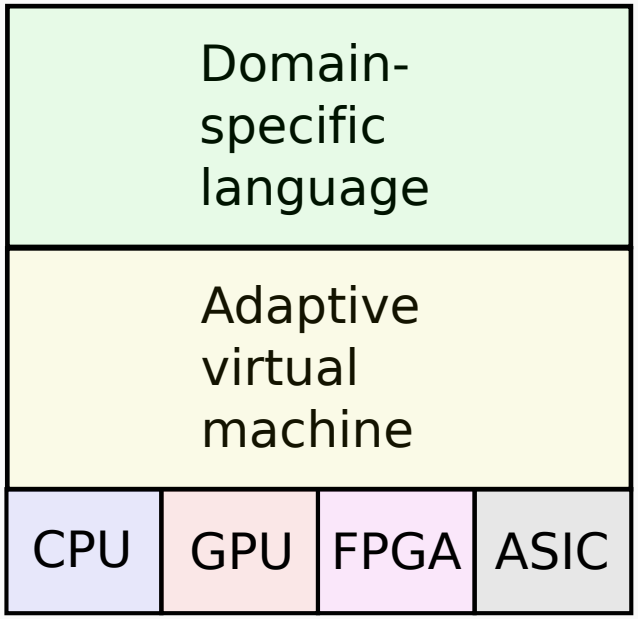
Dynamic VM

Workload-specific optimizations

Multiple target architectures

GPUs, potentially FPGAs

Takeaways



DSL

- Abstract enough for:
  - Efficient portability
  - Adaptive optimizations
  - Efficient interpretation
- State of art does not fit!
- *Data parallelism as first-class citizen*

VM

- *Interpret first, maybe compile later*
- Cost-models are hard to get right!
- Adaptive by design
- Aggressive workload-driven optimizations
- Mixed execution