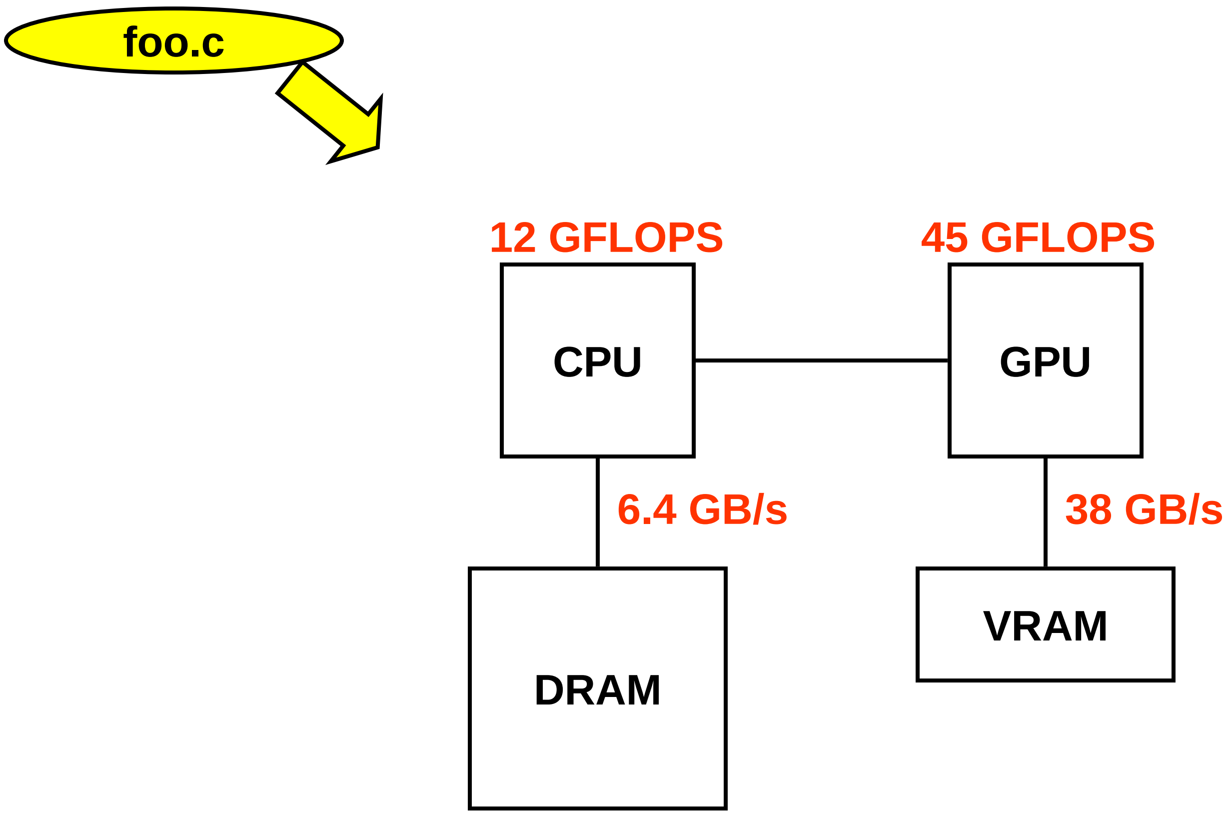
A Streaming Virtual Machine for GPUs

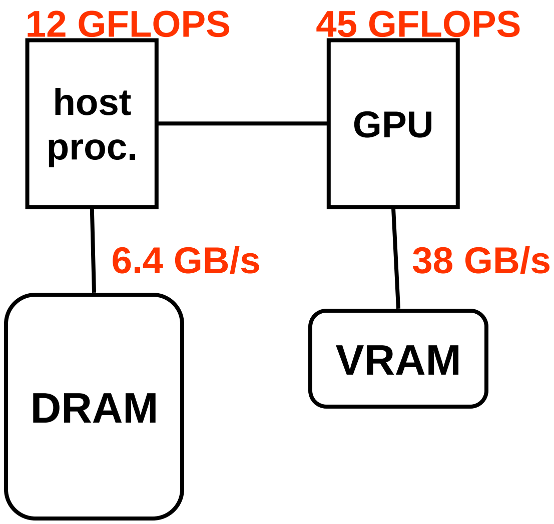
Kenneth Mackenzie **(Reservoir Labs, Inc)**
Dan Campbell (Georgia Tech Research Institute)
Peter Szilagyi (Reservoir Labs, Inc)

Goal: Compile to PCs w/GPUs



Barriers to General-Purpose Use

- **Hardware:**
  - Severe GPU programming restrictions! y=f(x) applied in parallel over an array, y.
  - CPU<->GPU bottleneck: 4GB/s
- **Compiler:**
  - No existing streaming compiler
- **Abstraction:**
  - GPU drivers built for graphics
  - Driver and hardware details are proprietary



Subgoal: Build and Evaluate an Abstraction atop GPUs

- **Hardware:**
  - **Severe GPU programming restrictions! y=f(x) applied in parallel over an array.**
  - **CPU<->GPU pipe: 4GB/s**

GPU vendors working on more general functionality

- **Compiler:**
  - **No existing streaming compiler**

Reservoir and others working under DARPA Polymorphous Computing Architectures (PCA) program

- **Abstraction:**
  - **GPU drivers built for graphics**
  - **Driver and hardware details are proprietary**

This project: implement PCA's Streaming Virtual Machine (SVM) abstraction atop GPUs and evaluate it.

Status; Related Work

- **Status: in-progress**
  - **Runs simple programs end-to-end**
    - **Must spoon-feed programs through the not-quite-GPU-aware streaming compiler.**
  - **Experimenting with feedback**

- **Related Work:**
  - **BrookGPU, Ian Buck, et al (Stanford), SIGGRAPH, 2004.**
  - **PUG, Mark Harris (nVidia), GPU Gems 2, 2005.**
  - **Sh, Michael McCool, et al (Waterloo), Graphics Hardware 2002.**
  - **All are programmer interfaces, not compiler targets.**

Outline

- Background on GPUs (2 slides)
- Streaming Virtual Machine
- Prototype SVM Toolchain
- Results
- Future Work

GPUs

- **GPUs implement the last few stages of a standard 3D graphics rendering pipeline.**

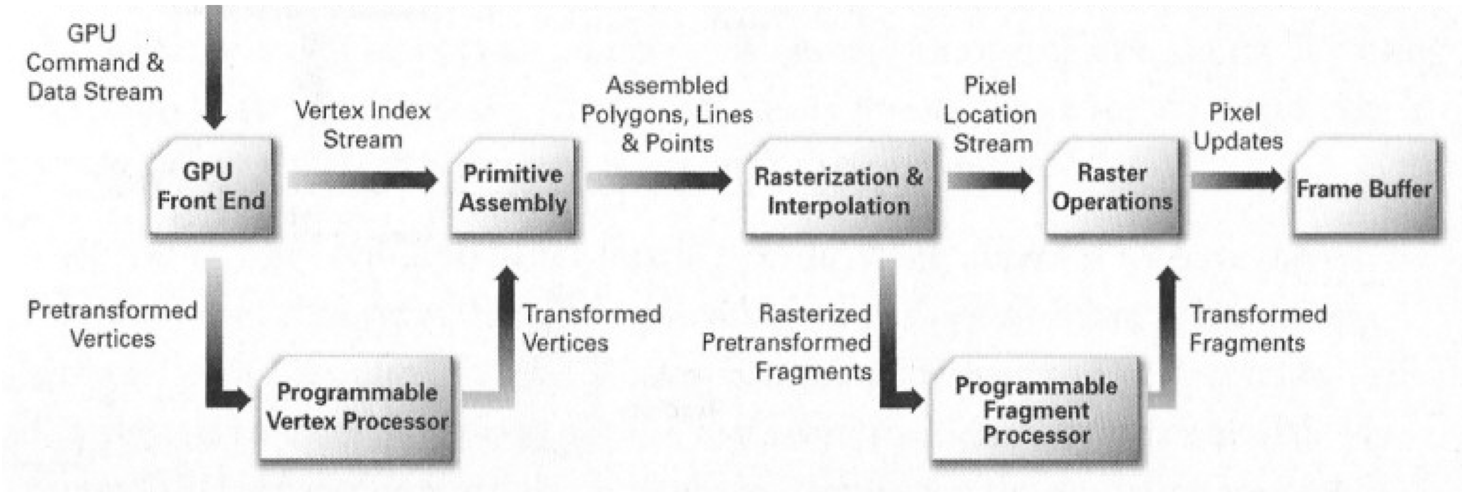


Illustration: from Cg Toolkit User's Manual, nVidia corp.

- **Recent GPUs employ embedded multiprocessors (e.g. 24-way SIMD) for programmability in several the stages.**
- **Trend is toward more generality and wider multiprocessing.**

GPUs for non-Graphics Programs

- **Use the "fragment processor" embedded multiprocessor only.**
  - **Ignore for now potentially useful but mind-bending hardware goodies.**
- **Place data arrays in textures.**
- **Compute y=f(x1, x2, ...) where y, xs are textures and f() is a function of any entries in the xs onto each entry in y.**

- **Many and serious restrictions:**
  - **No-scatter constraint: gather from xs but no scatter to y**
  - **No local storage; no loop-carried dependencies.**
  - **Ops are 32-bit, not-quite-IEEE floating-point; no integer.**
  - **Branches permitted but penalized by SIMD architecture**
  - **Byzantine limits/costs on the complexity of f()**
  - **Substantial startup overhead; N1/2 in 1000s**

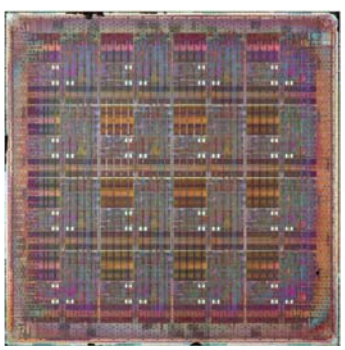
Streaming Virtual Machine

DARPA Polymorphous Computing Architectures (PCA)

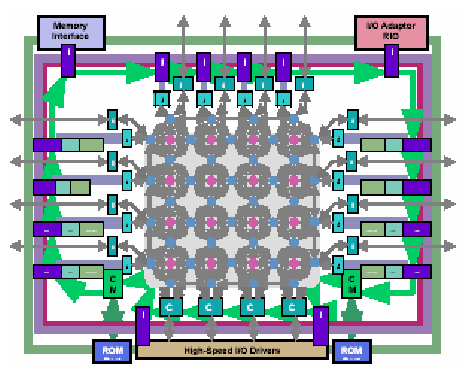
Tiled Multiprocessors

- **Chip multiprocessors built of replicated tiles**
- **Architectural novelty: mechanisms for combining tiles into larger units**
- **“Polymorphous”: configure the hardware to match the application, e.g. “threaded” vs. “streaming”**

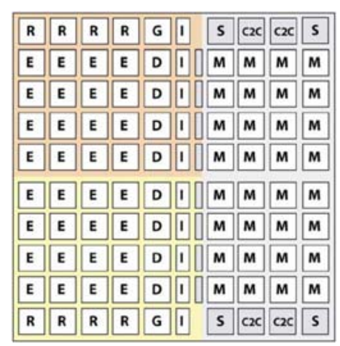
MIT RAW



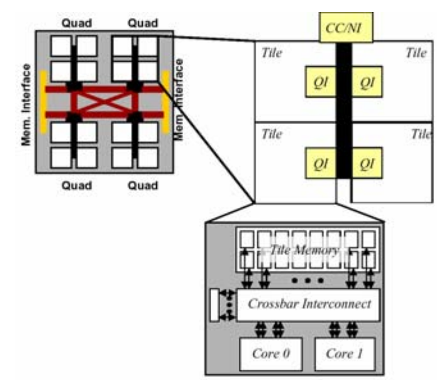
ISI/Raytheon Monarch



UT Austin TRIPS

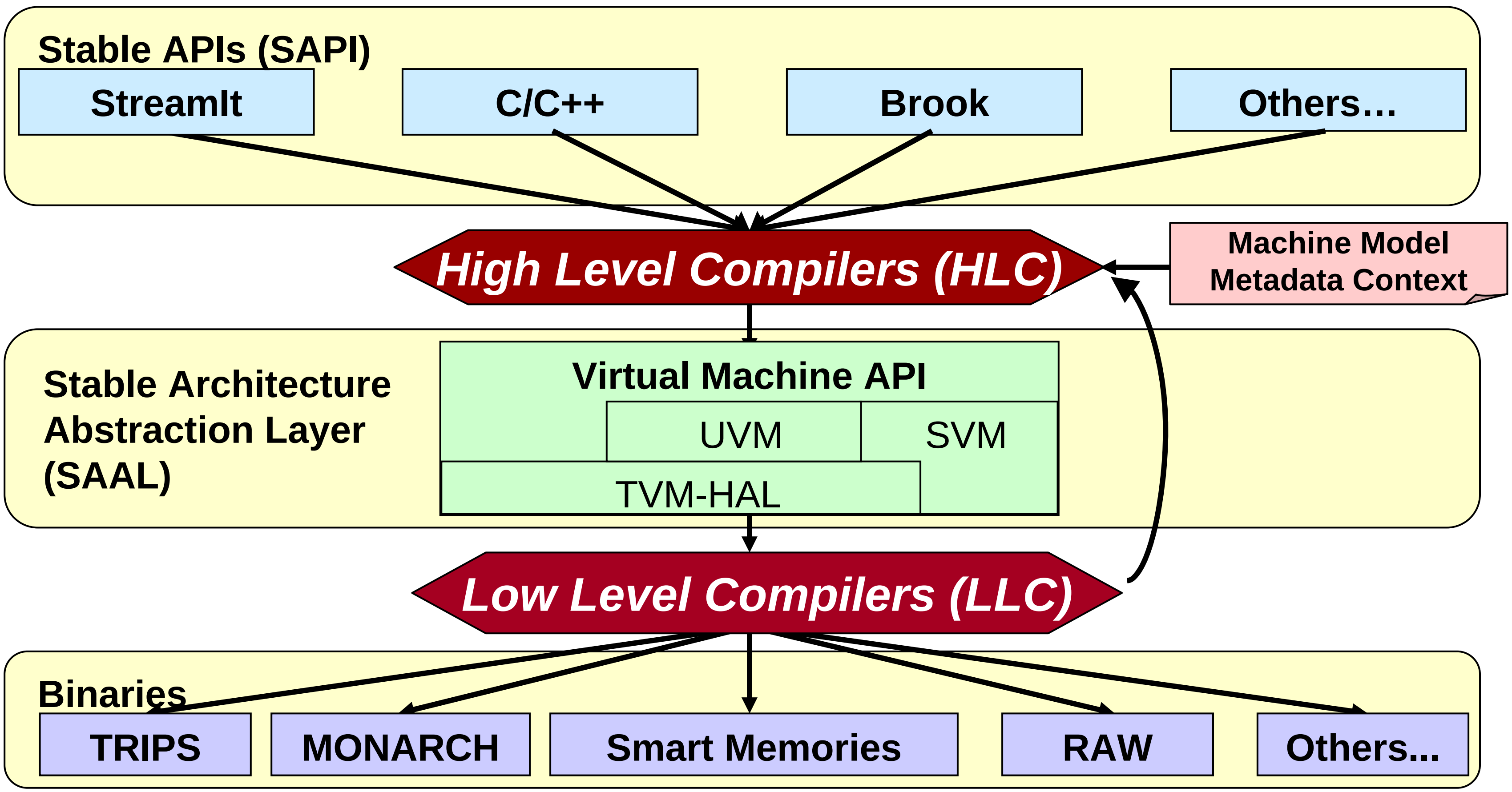


Stanford Smart Memories

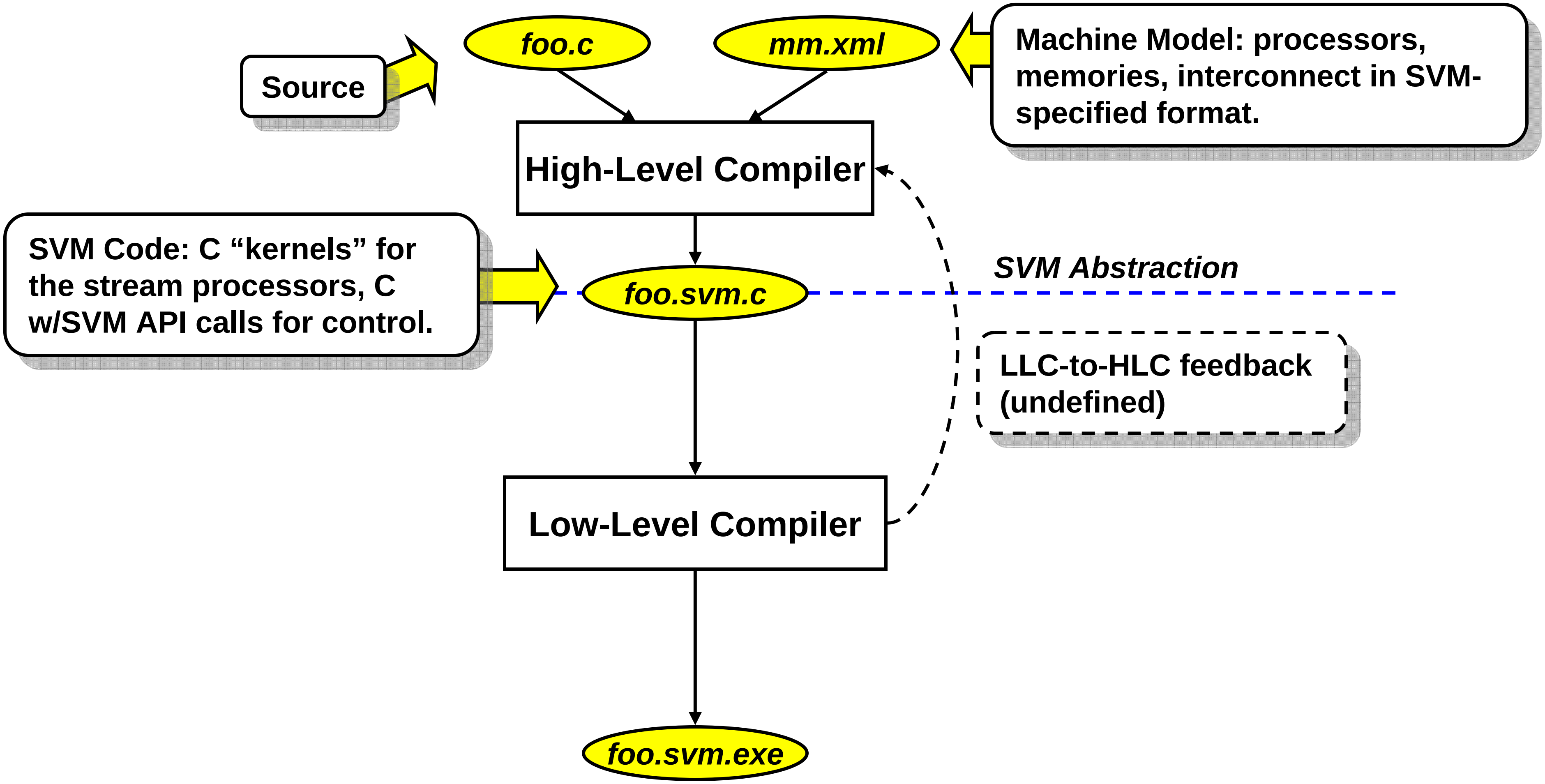


PCA Toolchain

- Two-level compilation factors the compilation problem.
- SVM is one abstraction and path through the toolchain.



SVM Slice of the PCA Toolchain



SVM Details

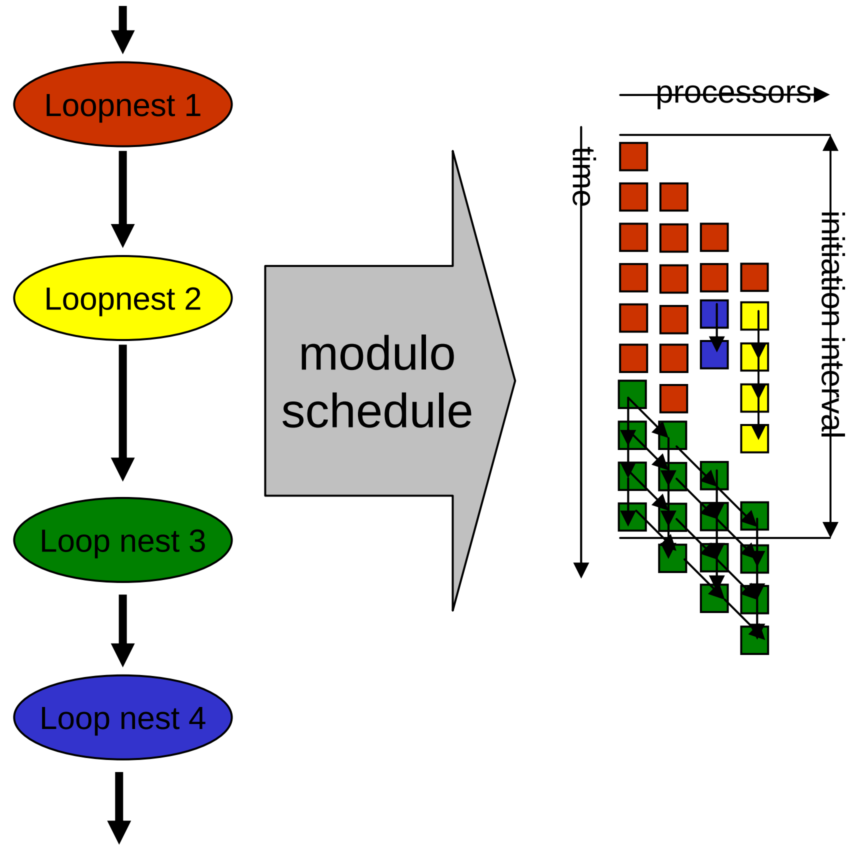
- **Machine Model: abstract architecture description in terms of processors, memory units, dma unit and interconnect in some topology.**

- **High Level Compiler: parallelizes, maps and schedules computation, storage and communication onto the machine model resources.**

- **Low Level Compiler: a hardware-specific uniprocessor compiler.**

SVM Detail: R-Stream High-Level Compiler

- **Map and schedule computation, storage and communication**
- **Reservoir's R-Stream**
  - **Oriented to static computation, e.g. radar front-end.**
  - **Converts loop bodies to kernels sized to fit local memory constraints.**
  - **modulo-schedules kernels on stream processors in a macro-pipeline.**



SVM Detail: R-Stream High-Level Compiler

Input is "Gumdrop": an annotated C

```
#pragma res parallel
doloop (int i = 0; i < N; ++i) {
 z[[i]] = a * x[[i]] + y[[i]];
}
```

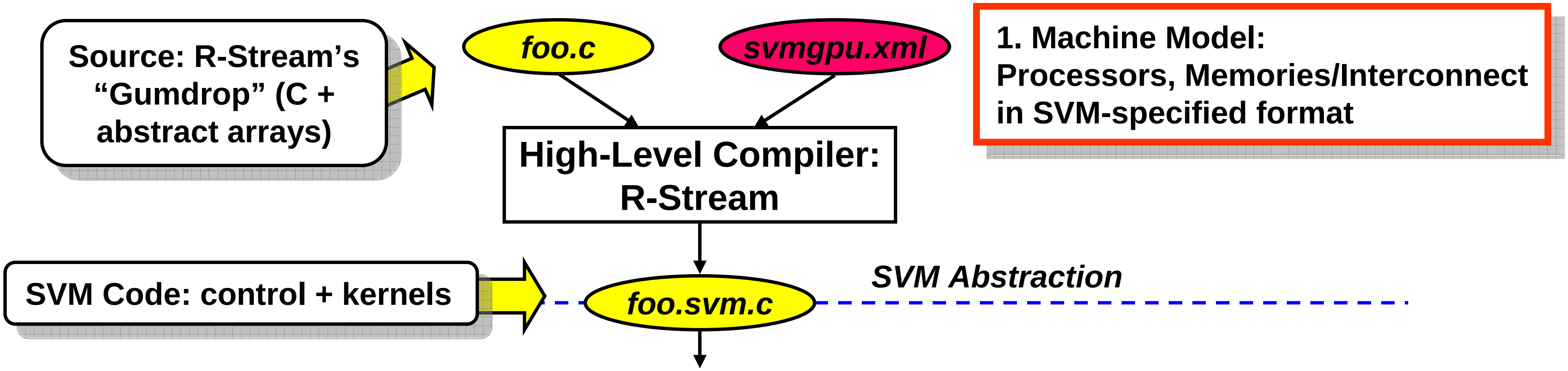
Output is SVM: C for kernels (shown)
plus C w/API calls to invoke kernels (not shown)

```
static void main_kernel_work_0(struct kernel_data_tag_0 *d) {
  int i;
  int const hlc_hi_i = d->i_max;
  for (i = d->i_min; i < hlc_hi_i; i++) {
    float _t, _t_1, _t_2;
    SVM_BLOCK_READ(d->x_block, i - d->x_block_offset_0, &_t_2);
    SVM_BLOCK_READ(d->y_block, i - d->y_block_offset_0, &_t_1);
    _t = d->a * _t_2 + _t_1;
    SVM_BLOCK_WRITE(d->z_block, i - d->z_block_offset_0, &_t);
  }
}
// ...
```

Prototype SVM-GPU Toolchain

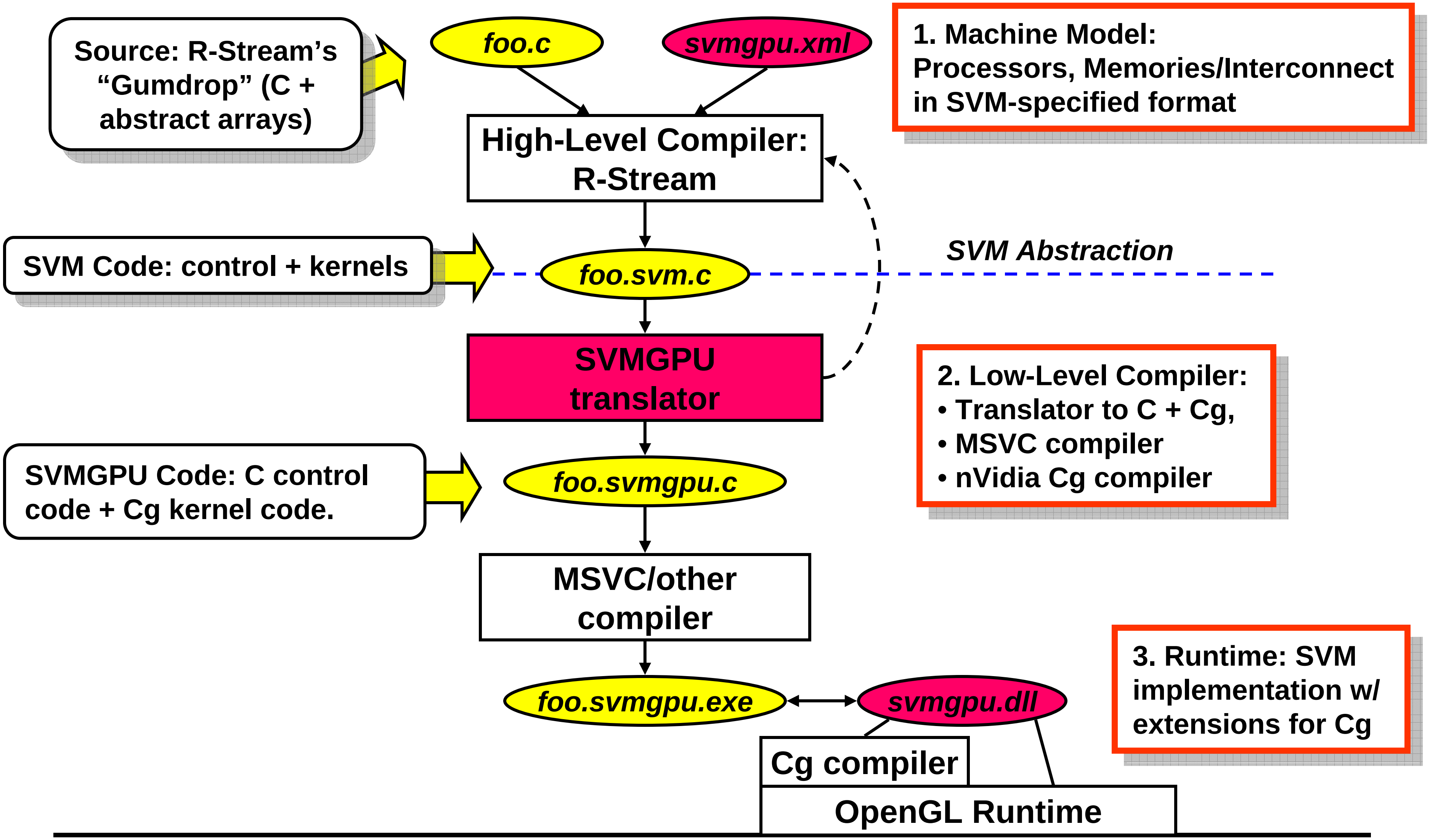
1. Machine Model
2. Low-Level Compiler
3. Runtime

Toolchain (HLC)



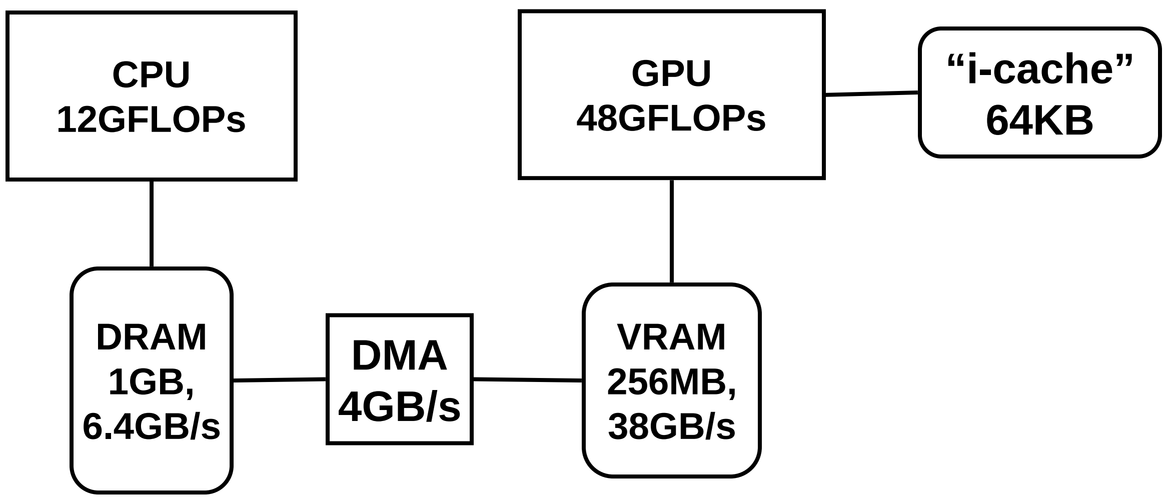
reservoir labs® Georgia Institute of Technology DARPA PCA

Toolchain (all)



1. Machine Model

- Model the GPU as one fast processor (the fragment shader).
- Model the VRAM as local memory.
- Model a GPU “i-cache” to indicate limited program store
- Model DMA between DRAM and VRAM although hidden by driver.



- Handles multiple GPUs (duplicate VRAM and DMA to match)
- Handles multiple CPUs

Machine Model Approximations

- **No model of extra hardware features, e.g. interpolation, z-sort**
  - **Use of these features is likely limited to libraries**
- **No model of SIMD details: startup cost, branch cost**
  - **Fixable**
- **No model of the no-scatter constraint**
  - **Conceivable in SVM's machine model schema but R-Stream does not currently understand it.**
- **No model of detailed resource constraints**
  - **Number of registers (shader programs cannot spill registers)**
  - **Cost of instruction combinations**
  - **Cost of register usage vs. # of threads**
  - **Note: much of this detail is <u>impossible</u> to model precisely!**

2. Translator

- **What it is:**
  - **SVM (C) to SVMGPU (C + Cg) translator**
  - **Combines with vendor C and Cg compilers to form an SVM “Low-Level Compiler”**

- **Compact experimental prototype**
  - **1400 lines of SML**

Translator Operation

- Translates kernel bodies to Cg fragment shader programs
  - Outermost loop in a kernel removed (becomes hardware rasterization)
  - Input arrays become Cg textures
  - Input loop-invarient values become Cg uniform parameters
  - Output arrays become Cg out parameters

- Translates the outermost loop in kernels to hardware rasterization
  - Fragment program invocation over a block of data
  - Block extents given by loop bounds

- Checks correctness conditions at compile- and/or at runtime
  - check no-scatter constraint
  - A kernel that fails this check is run on the CPU instead of the GPU

3. Runtime

- **Implements SVM functionality**
- **Includes support for SMP/clusters of CPUs and multiple GPUs**
- **Built atop OpenGL, Cg, nVidia/ATI drivers, and Windows.**
- **Compact experimental prototype**
  - **2300 lines of C**

Runtime Operation

- Manages textures as storage for SVM blocks
- Executes Cg code for translated SVM kernels
  - Falls back to running the kernel on the CPU if Cg compilation fails
- Implements DMA kernels using OpenGL calls

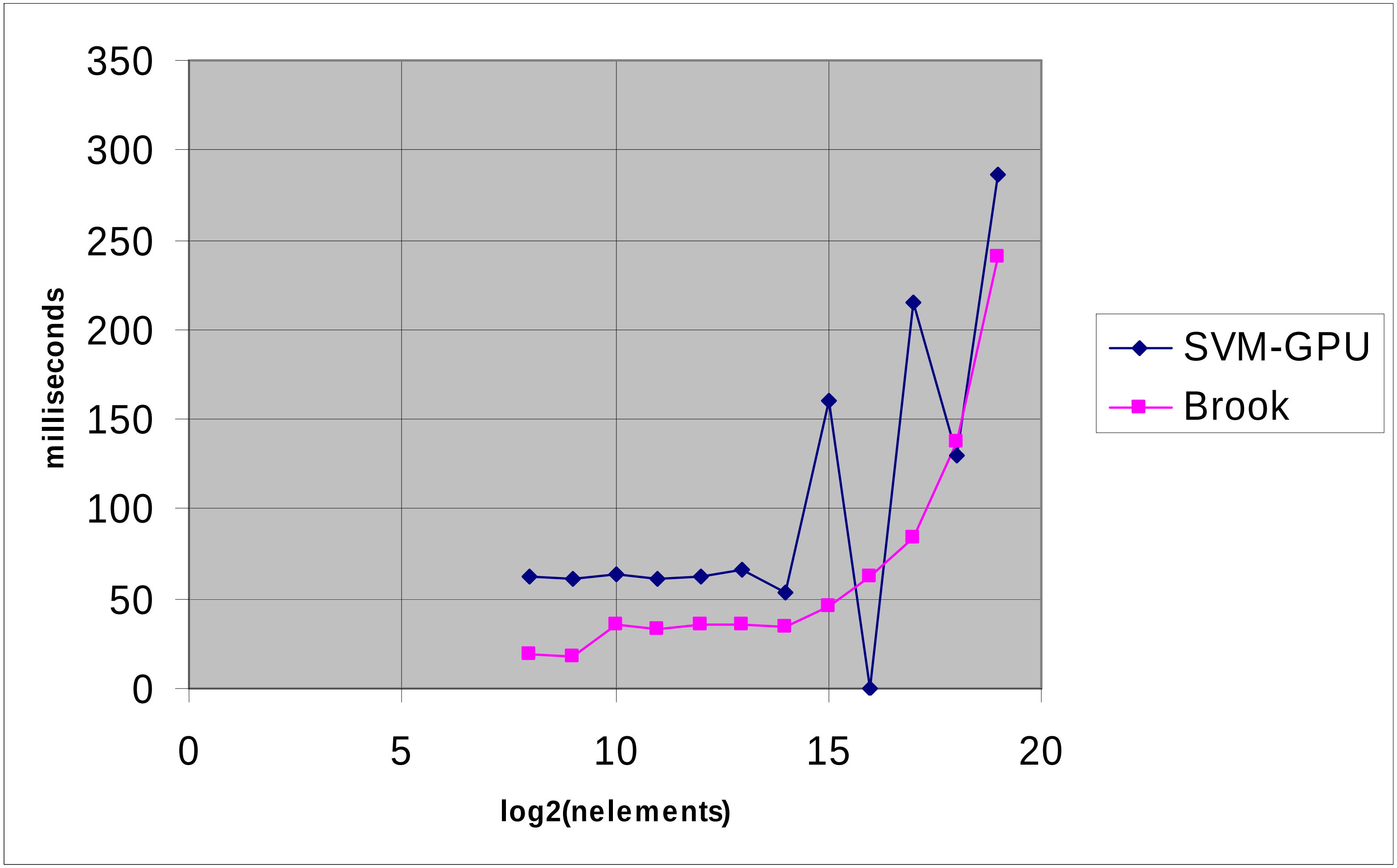
Results

Results

- **Quantitative:**
  - **Successfully executes simple programs.**
  - **Still tuning to reduce overhead to the level of BrookGPU.**

- **Qualitative:**
  - **GPUs**
    - **The no-scatter constraint is the most serious.**
    - **The no-local-storage constraint is the next worst.**
  - **R-Stream**
    - **Needs to recognize the basic GPU constraints to be automatic.**
    - **We can work around this in source code for experiments.**
  - **SVM**
    - **C is tough to translate; the HLC's analyses are lost.**
    - **Feedback is necessary.**

Result: SAXPY Execution Time



GPU Kernel Constraints

- **Fragment programs write outputs exactly once, in-order.**
- **Fragment programs have no local storage.**

- **R-Stream currently doesn't recognize the constraints and will, e.g., fuse together GPU-friendly loops into one GPU-unfriendly loop.**

```
#pragma res parallel
{
  for (i = 1; i < N; i++) {
    y[i] = x[i - 1] + x[i];
  }
  for (i = 1; i < N; i++) {
    z[i] = y[i - i] + y[i]
  }
}
```

- **Workaround: mark loops separately.**

Feedback

- Feed-forward via the machine model is preferable
- Feedback is inevitable
  - Some constraints are impractical to model or to solve
  - Some constraints are unknown/proprietary
  - Conservative interpretation of constraints is sub-optimal
- Feedback makes the compilation process a search
- What kind of feedback is available when:
  - From the translator (arbitrary but imprecise)
  - From Cg (pass/fail, little else without vendor assist)
  - From trial execution of code (performance)

Summary and Future Work

- **A Streaming Virtual Machine for GPUs**
  - **Machine model**
  - **Low-level compiler built via a translator to C + Cg**
  - **Runtime atop ATI/nVidia targets**

- **Work in progress:**
  - **Characterize feedback requirements and propose mechanisms**

- **Future work:**
  - **Supporting library code; optimization across libraries.**
  - **Exporting special hardware features via SVM.**