“Streaming” as a pattern

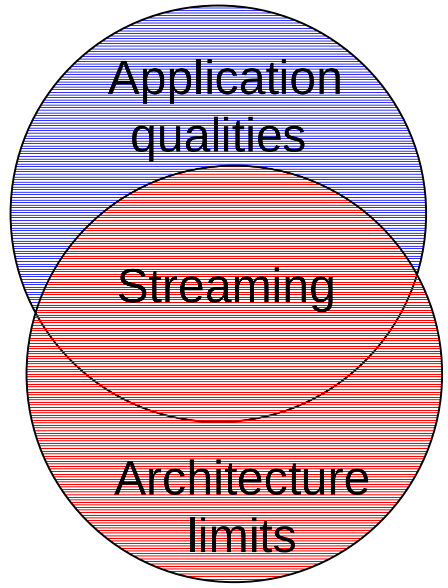
Peter Mattson, Richard Lethin
Reservoir Labs

“Streaming” as a pattern

- Streaming is a *pattern in efficient implementations of computation- and data-intensive applications*
- Pattern has three key characteristics:
  - Processing is largely parallel
  - Data access patterns are apparent
  - Control is high-level, steady, simple

Causes of the pattern

	Applications process, simulate, or render physical systems	**Architectural limits**
Processing is largely parallel	Independence of spatially and/or temporally distributed data	VLIW, SIMD, multiprocessor demand for parallelism
Data access patterns are apparent	Continuous, N-dimensional spatial/temporal data	Small local memories
Control is high-level, steady, simple	Few tasks per application, few unpredictable events	Handling unpredictability in hardware is expensive



Streaming is not only 1D

- Multidimensional data, data rearrangement
- Multidimensional architecture topologies
- Time is favored dimension, yielding 1D tendency
  - Unmapped, coarse-grain streams of whole-array "elements" moving between "actors" with whole-loop-nest invocations
  - Serialization for transmission, especially address stream to memory to get data stream to processor

Using streaming

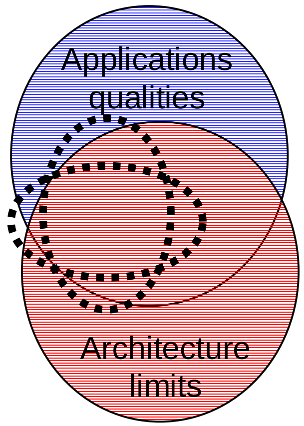
- Streaming applications expose maximum opportunities, information to compiler
- Streaming architectures expose maximum resources, control to the compiler

- **Streaming languages should enforce the pattern**
  - **Force programmer thought to reach pattern**
  - **Guarantee pattern to compiler**
- **Streaming compilers should exploit the pattern**
  - **Expand scope of application optimization**
  - **Expand scope of resource choreography**

Streaming languages

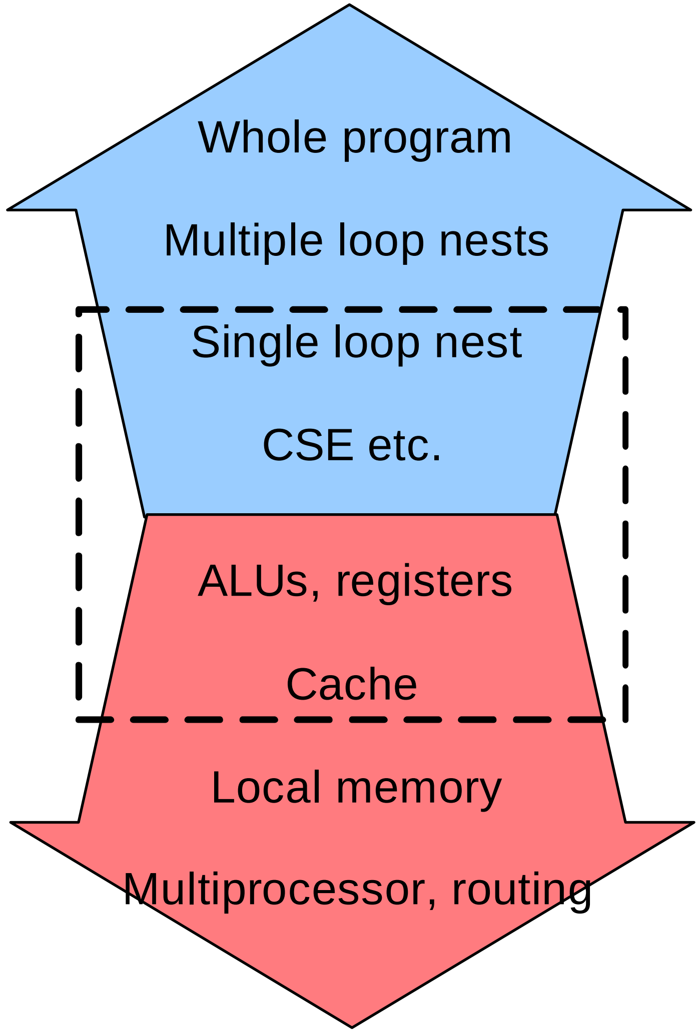
- Enforce similar, but not identical patterns

StreamIt	Brook/StreamC
Think structured synchronous data flow	Think pointer-less C, with embedded dataflow graphs instead of loop nests
Single stream graph	Multiple stream graphs, surrounded by C-subset
Streams are infinite length	Streams are finite length
Static rates	Dynamic rates
“Filters” can have state, may require sequential processing	“Kernels” must be state-less, allow parallel processing
Designed by compiler people, clean but more constrained	Designed by application and architecture people, rough but more expressive



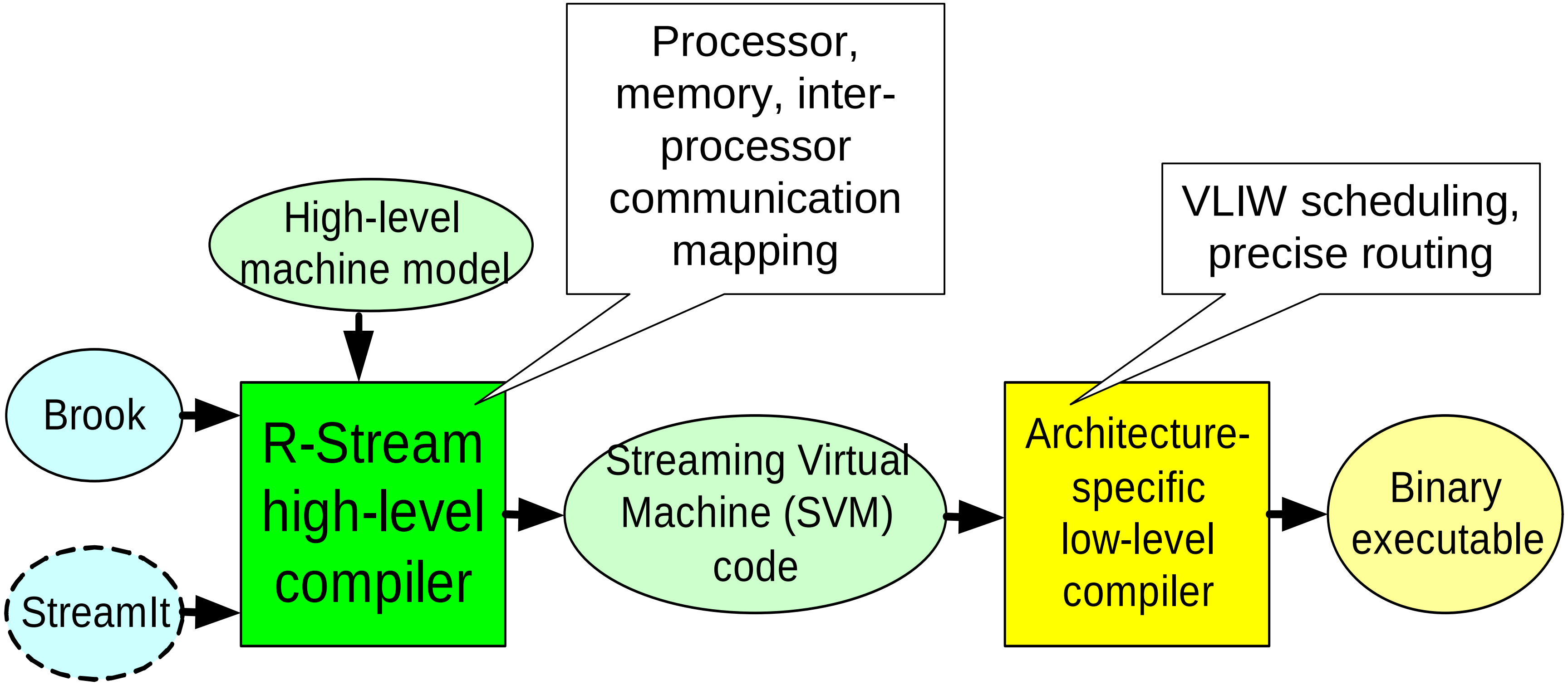
Streaming compilers

- Expand scope of optimization
  - Application is transparent to compiler
    - Access patterns, aliasing, etc. **completely** known
    - Top-down, not just bottom-up optimization
    - Exploit task parallelism
- Expand scope of choreography
  - Compiler lays out all computation, data, and communication
    - Closely model architecture
    - VLIW scheduling writ large
    - Difficult! Many variables, phase ordering…



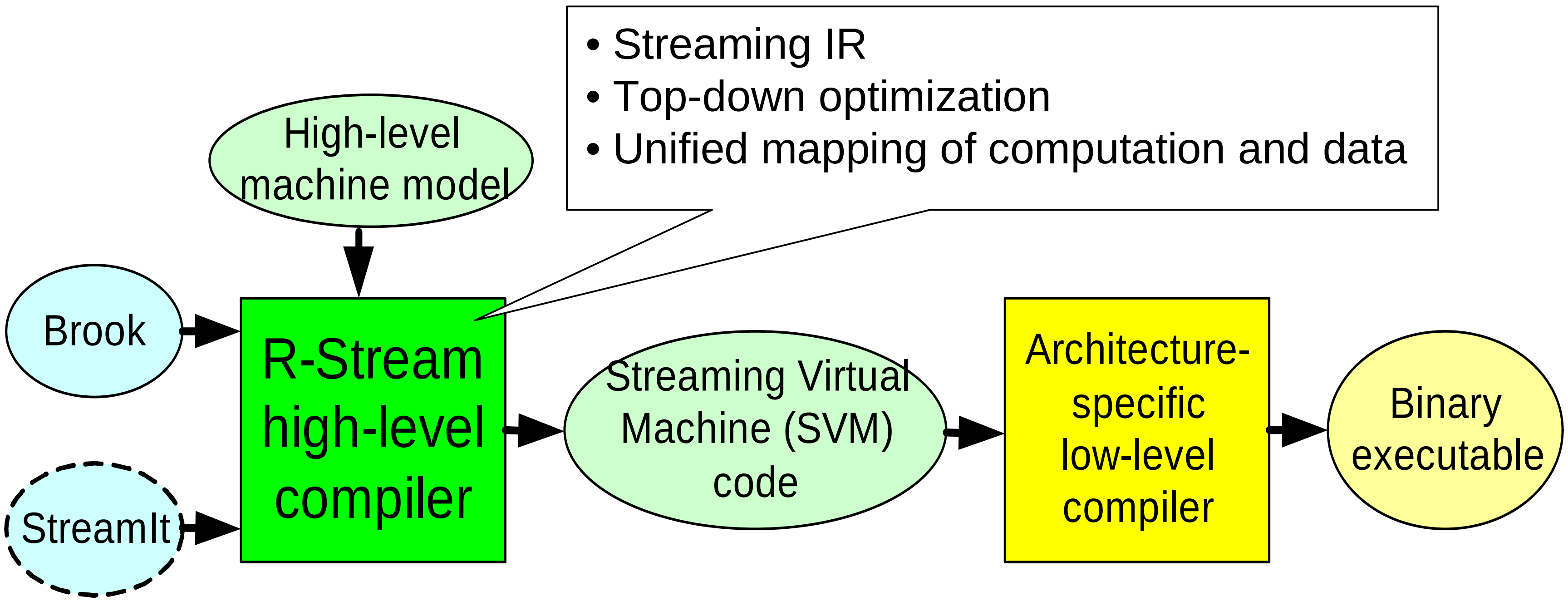
R-Stream

- Reservoir Labs is developing the R-Stream high-level compiler
  - Goal is *portable, consistently high-performance* compilation



R-Stream

- Reservoir Labs is developing the R-Stream high-level compiler
  - Goal is *portable, consistently high-performance* compilation



Streaming Pattern

- Characterized by parallelism, apparent data access, high-level etc. control
- Driven by application class, architectural limits
- Enforced by languages, exploited by compilers
- Used by R-Stream; goal of portable, consistently high-performance compilation