Universität Karlsruhe (TH)

Forschungsuniversität · gegründet 1825

Streaming Extensions for Object-Oriented Languages

Frank Otto, Victor Pankratius, Walter F. Tichy

University of Karlsruhe, Germany

Workshop on Streaming Systems

November 8, 2008, Lake Como, Italy

Fakultät für **Informatik**

Lehrstuhl für Programmiersysteme

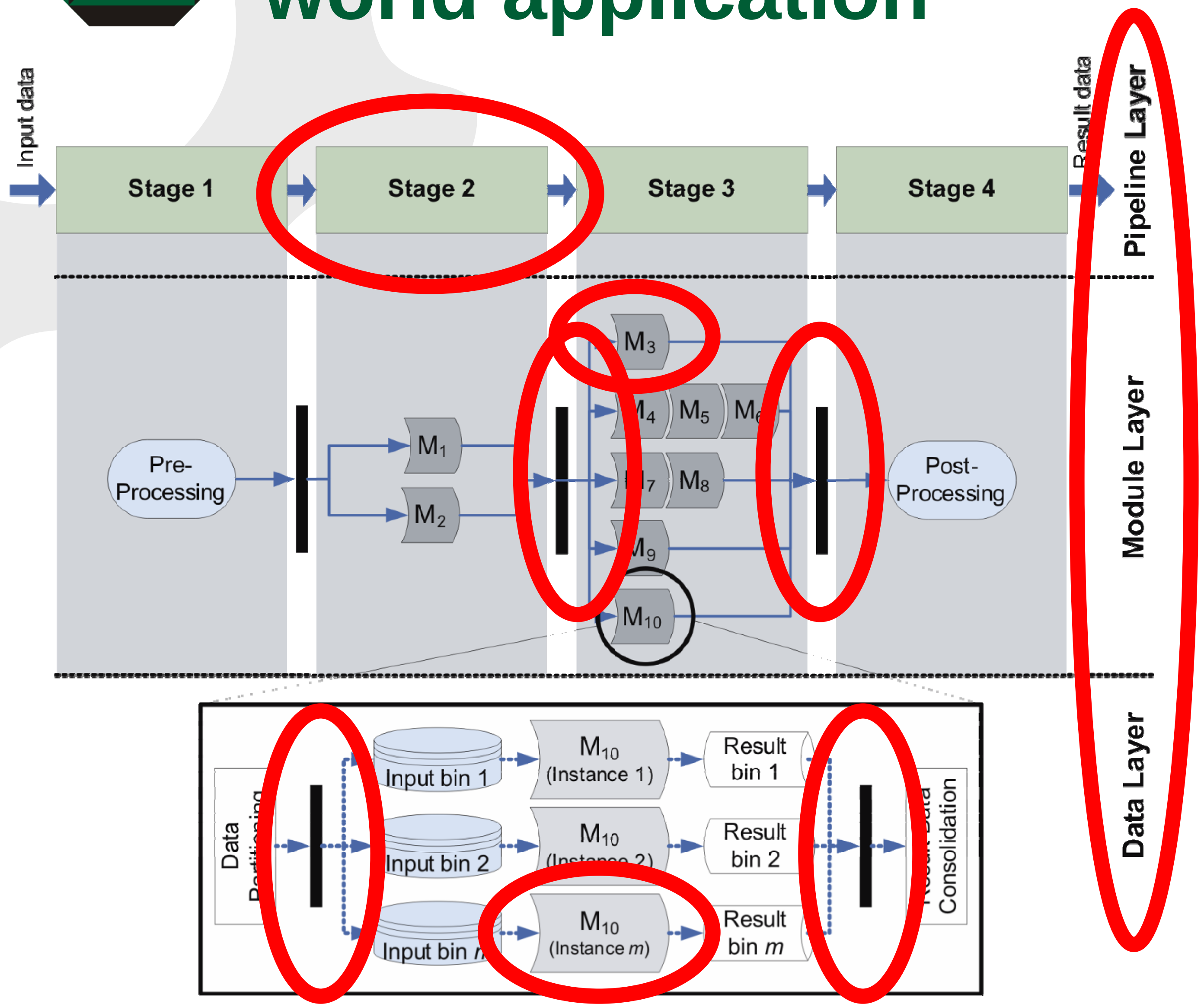
Streaming and OO

- Stream languages, e.g. StreamIt
  - Express different types of parallelism in a simple way
    - „Pipe-and-filter“ style
    - Exploit task/data/pipeline parallelism
  - Domain-specific: signal processing and graphics

- Object-oriented languages, e.g. Java, C#
  - Powerful, universal
  - Explicit multithreading is difficult and error-prone

➔ Combine the best of both

A parallelized object-oriented real-world application



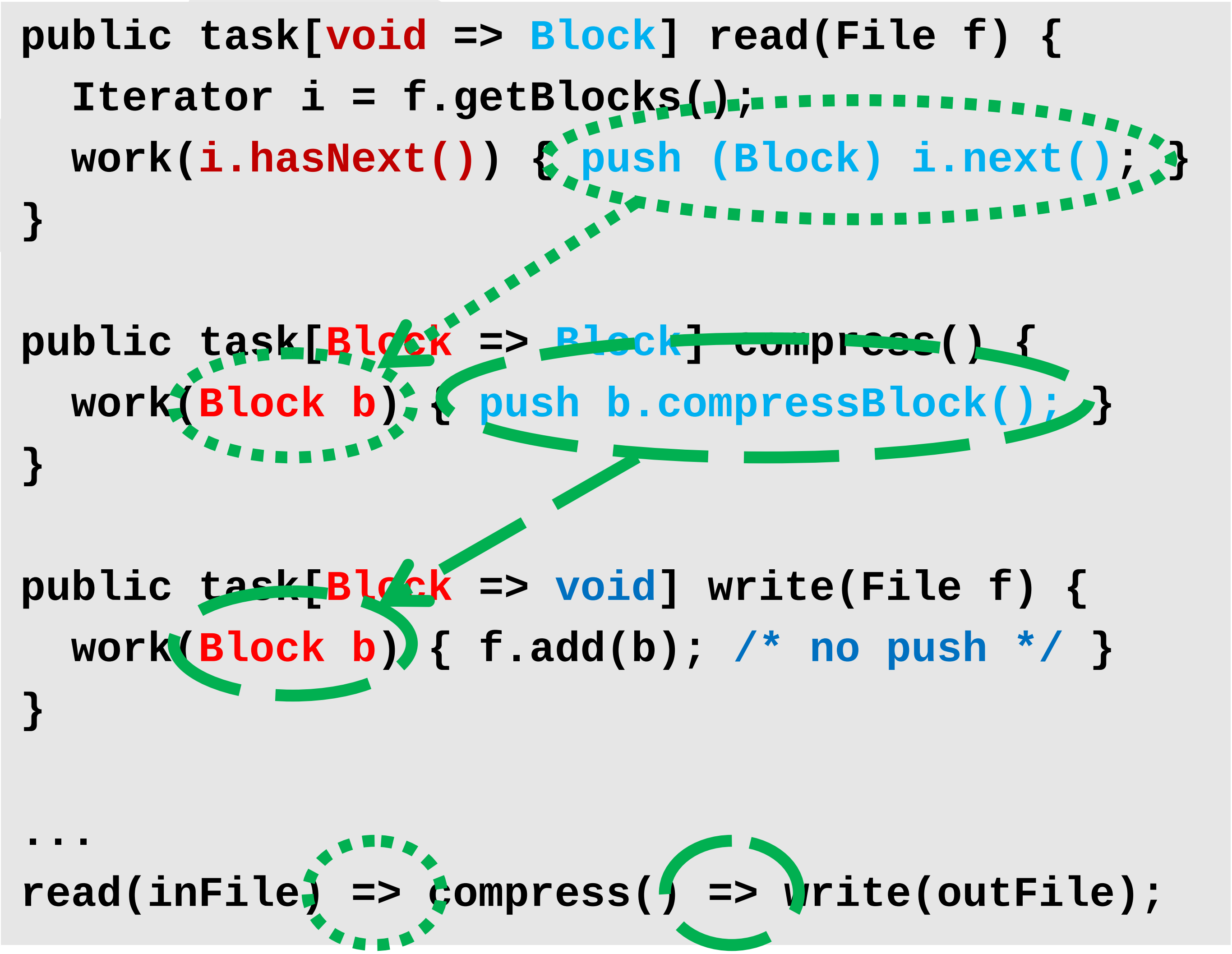
Different layers (abstraction levels)

Components for processing elements are similar to filters (input/output)

Split-join data flow

Stream constructs would help a lot (not only) in this case

Language Extensions for Java (1)



```
public task[void => Block] read(File f) {
  Iterator i = f.getBlocks();
  work(i.hasNext()) { push (Block) i.next(); }
}

public task[Block => Block] compress() {
  work(Block b) { push b.compressBlock(); }
}

public task[Block => void] write(File f) {
  work(Block b) { f.add(b); /* no push */ }
}

...
read(inFile) => compress() => write(outFile);
```

Tasks are declared like methods

Work statement for processing the stream (using any legal OO code)

Each task call is mapped to a thread

„=>“ operator takes care of the rest

Language Extensions for Java (2)

- Tasks are specialized methods
  - Dedicated input and output
  - May call other tasks as well (nested parallelism)

- Use tasks like methods
  - Can be public, private, static, abstract, final, ...
  - Throw and handle exceptions
  - Define them in interfaces or abstract classes
  - Inherit or override them

- Don‘t care about synchronization, indeterminism, etc.

Hopes and Promises

- Write parallel general-purpose applications in a „what-you-see-is-what-you-get“-style...
  - ... without sacrificing the power of OO!
- Performance
  - Exploit OO parallelism on all fronts (stream languages already do it)
- Abstraction
  - Hide confusing details wherever it is possible
- Less bugs & easier debugging
  - Intuitive language constructs & implicit parallelism: less error-prone
  - Compiler/debugger: more knowledge about semantics
- Code savings

Discussion

- Related work: StreamIt, Streamware, Merge, …
- Open questions:
  - Handling the number of threads, scheduling
  - Performance optimization at compiler and runtime level
  - Integration of (new) locking protocols or synchronization mechanisms
- **Discussion: potential, problems, limitations?**