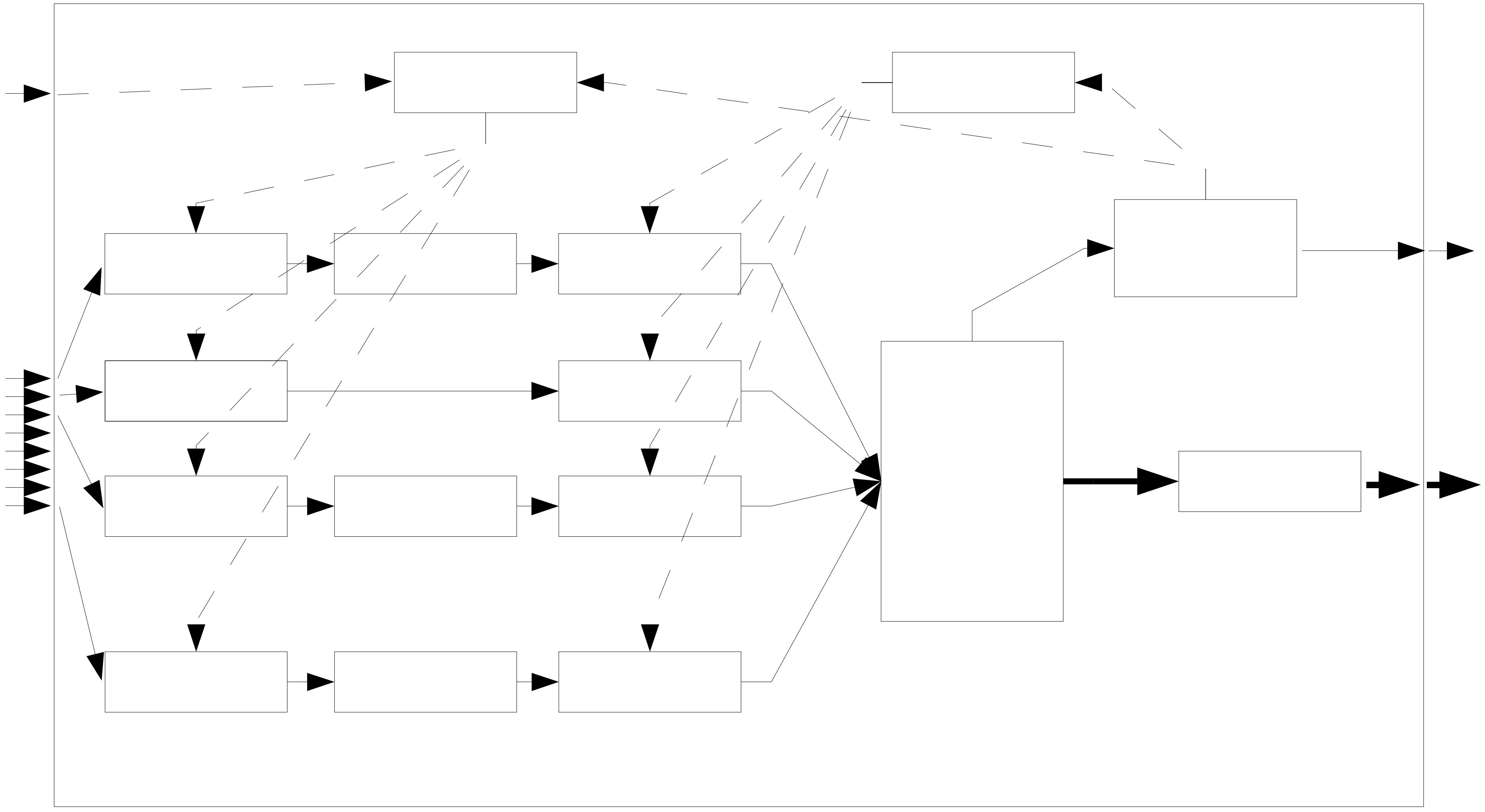
ArchiDeS
(*Architecture, Deployment, Scheduling*)
A Programming Framework
for Multicore Chips

Mats Brorsson, Karl-Filip Faxen
and Konstantin Popov
SICS/KMC

Software Architectures for Massive Data-Stream Processing



ArchiDeS

- An *application development framework* for dynamically reconfigurable data-flow systems
- Simple, expressive, efficient
- Major example of target application classes: RBS (radio base station) software model
- *Separates* architecture specification from practicalities of running it on multicore hardware
  - RBS model(s) are to be *understood* by experts
  - we design *different* schedulers for MC hardware
- We believe it can be used for “real” applications!

Rationale for a New Model

- Simple (yet expressive)
  - *simple*: few intuitive concepts, separation of aspects
  - supplement common industrial practices, like OOP
    - use it only where and when necessary
- “tailored” for multi-core execution
  - focus on exploiting available cores
  - supporting different, large-scale multi-core chips
    - shared memory *and* message-passing
  - support for application-specific scheduling
    - that can be optimized for latency, throughput, QoS, ..

Concerns and Their *Separation*

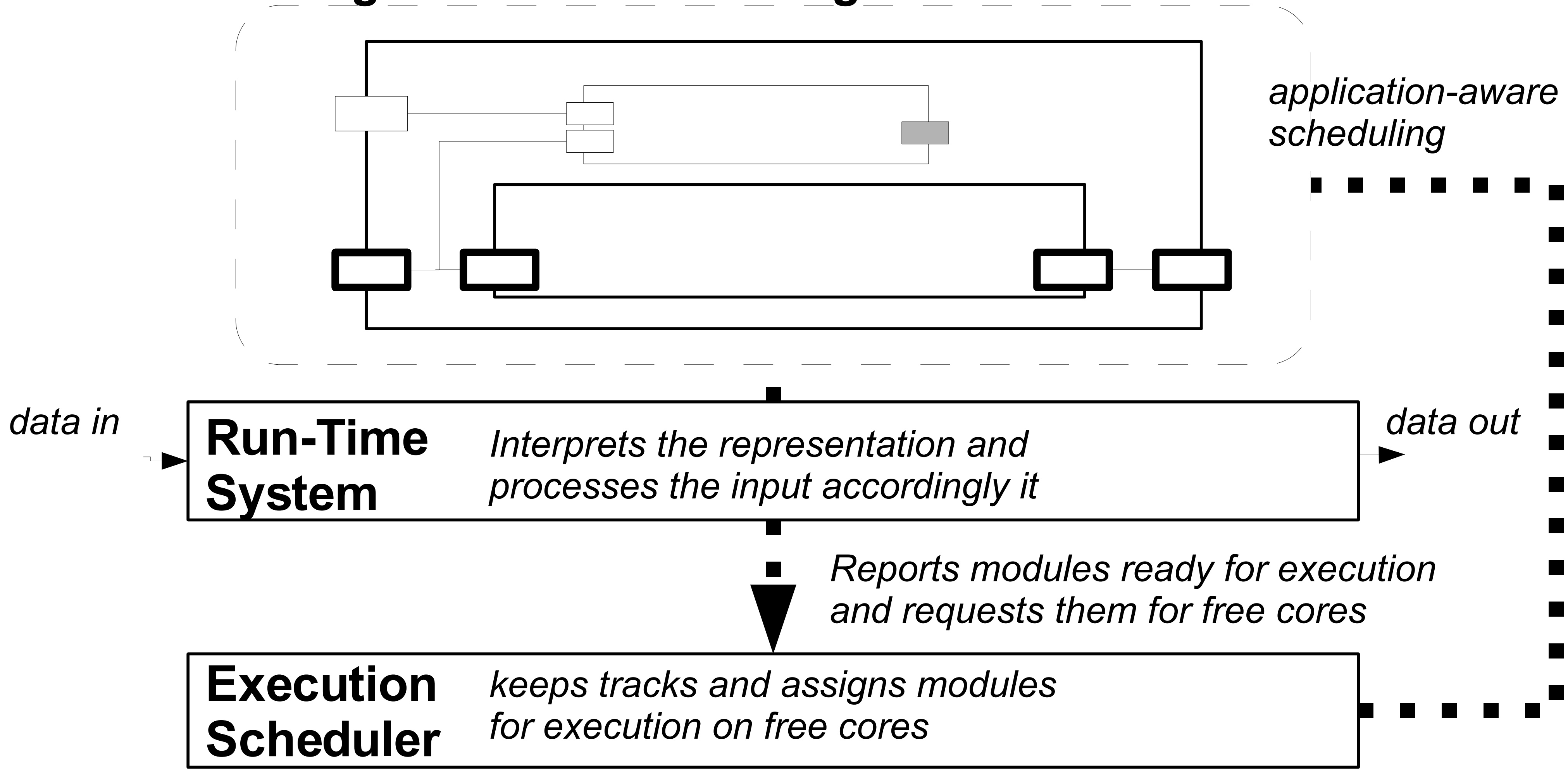
- Specifying application architecture
  - applications' *functional* behaviour
- Deployment
  - architecture “(re)adaptation” for particular hardware
- Scheduling on multi-core architectures
  - optimizing it for particular multicore hardware and execution requirements (throughput, energy, ..)
- Application execution framework
  - Run-Time System (RTS)
  - *independent* of the particular hardware platform

Types of Parallelism

- Data parallelism
  - e.g. multiple clients processed by identical chains up to “multiplexing” modules
- Pipeline parallelism
  - multi-stage data processing
  - different stages working on different units of input data simultaneously
    - not necessarily with the same time to traverse the pipeline for different system users (user-level data streams) etc.
- Module internal parallelism
  - is *orthogonal*: unsupported but not disallowed either

Systems, RTS and Execution Schedulers

first-class representation of subsystems, configurations and bindings

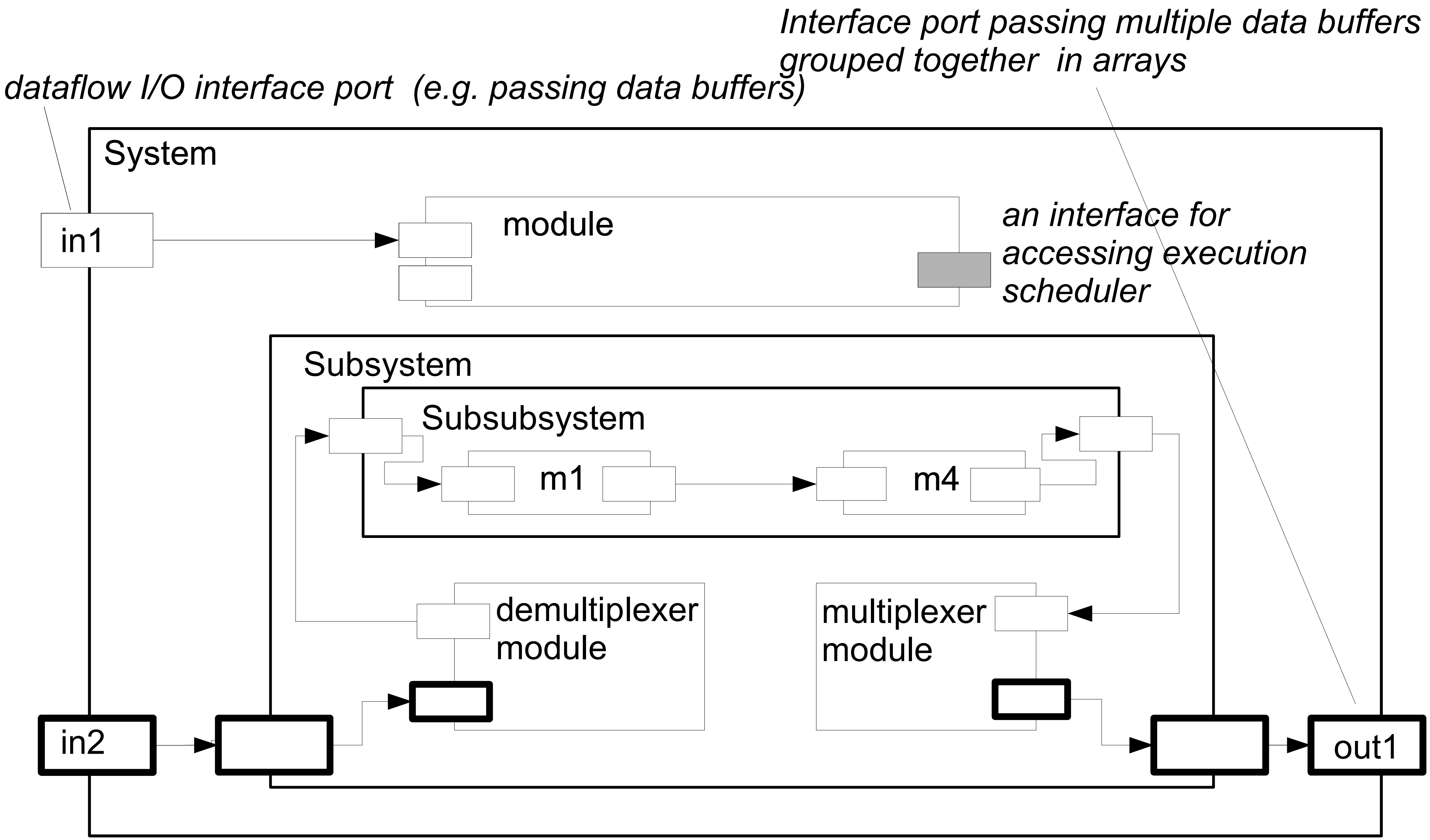


Run-Time System *Interprets the representation and processes the input accordingly it*

Reports modules ready for execution and requests them for free cores

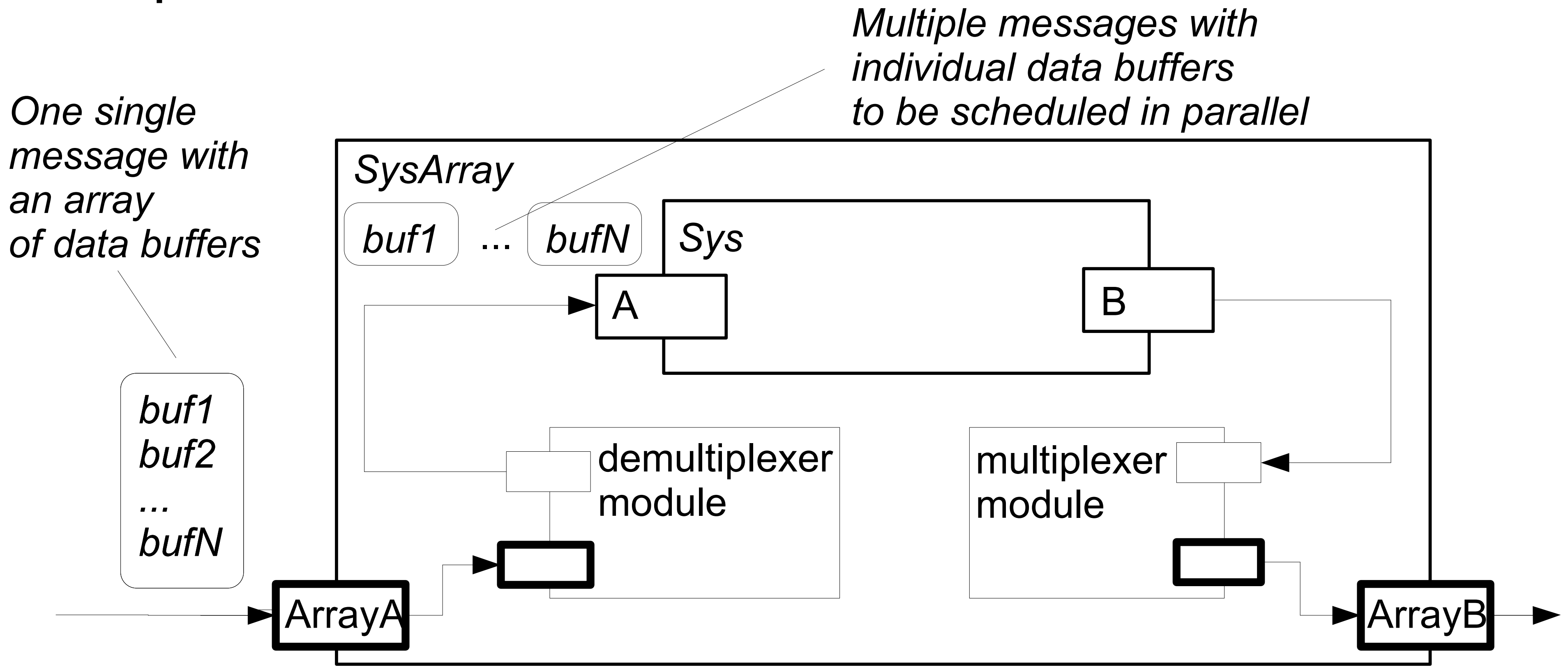
Execution Scheduler *keeps tracks and assigns modules for execution on free cores*

Data-Flow Software Architectures

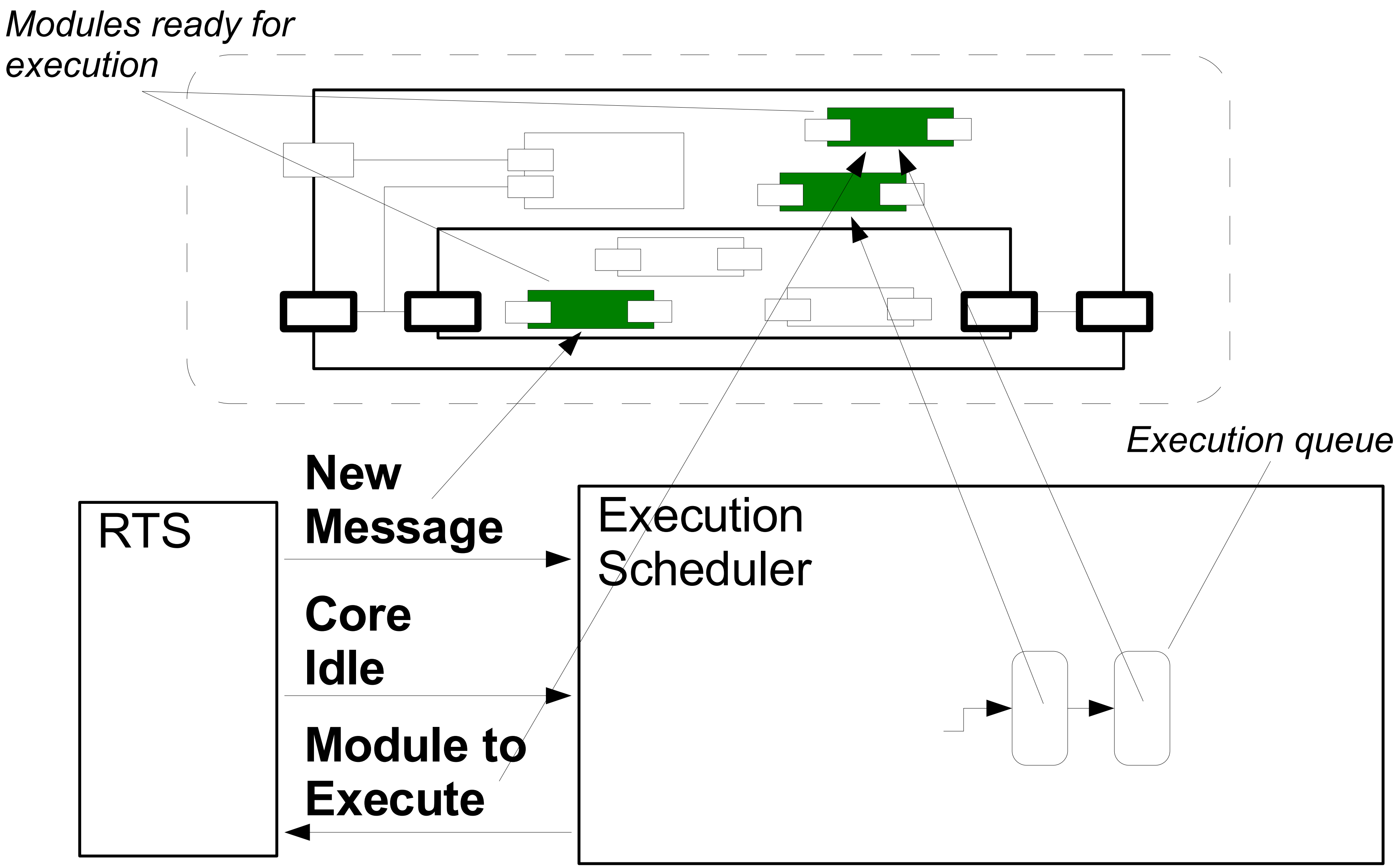

Interface port passing multiple data buffers grouped together in arrays
dataflow I/O interface port (e.g. passing data buffers)
System
in1
module
an interface for accessing execution scheduler
Subsystem
Subsubsystem
m1
m4
demultiplexer module
multiplexer module
in2
out1

Data-Parallel Execution

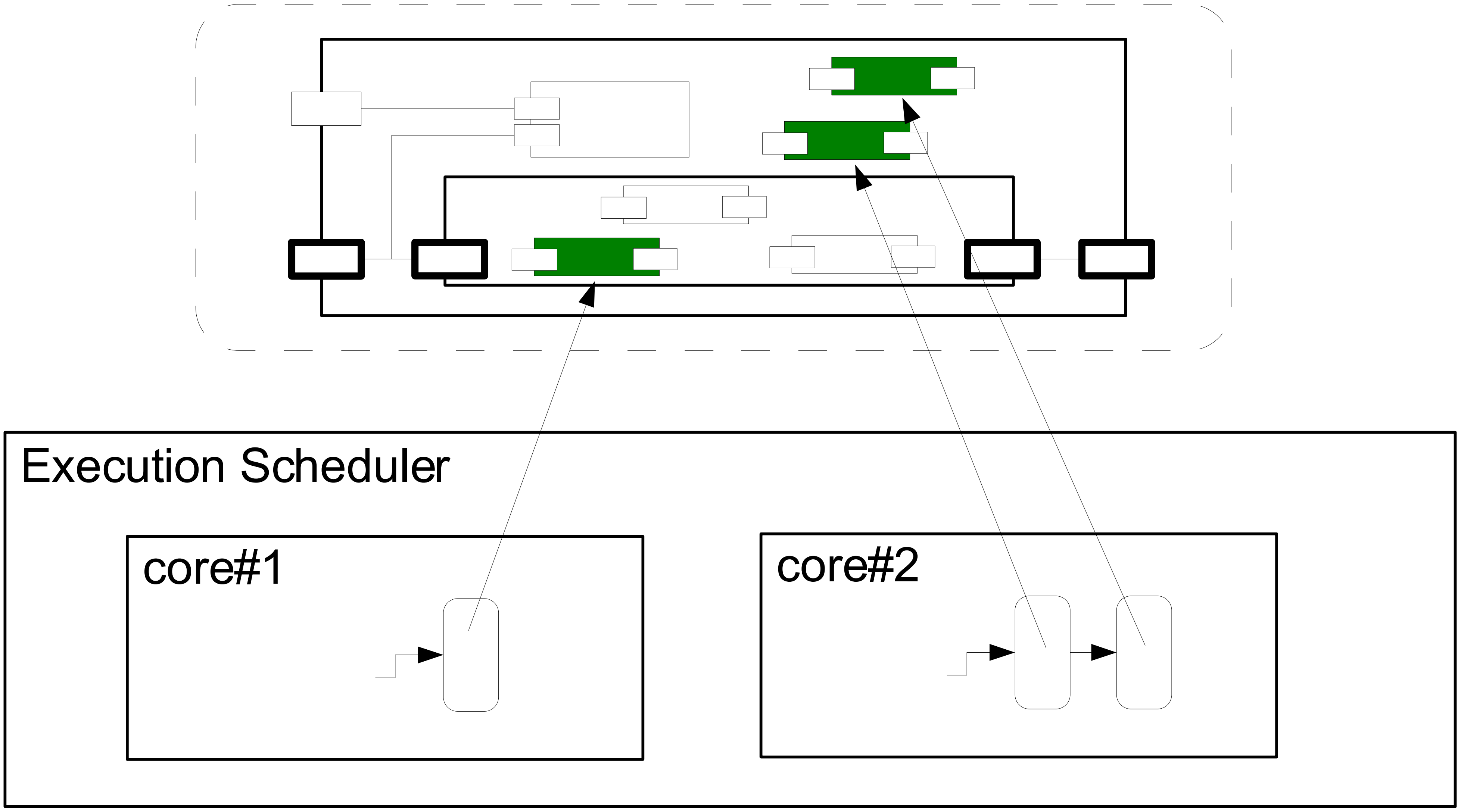
- *SysArray* provides for data-parallel execution
  - provided *Sys* is stateless, thus can be scheduled for parallel execution


Multiple messages with individual data buffers to be scheduled in parallel
One single message with an array of data buffers
SysArray
buf1
...
bufN
Sys
A
B
buf1
buf2
...
bufN
demultiplexer module
multiplexer module
ArrayA
ArrayB

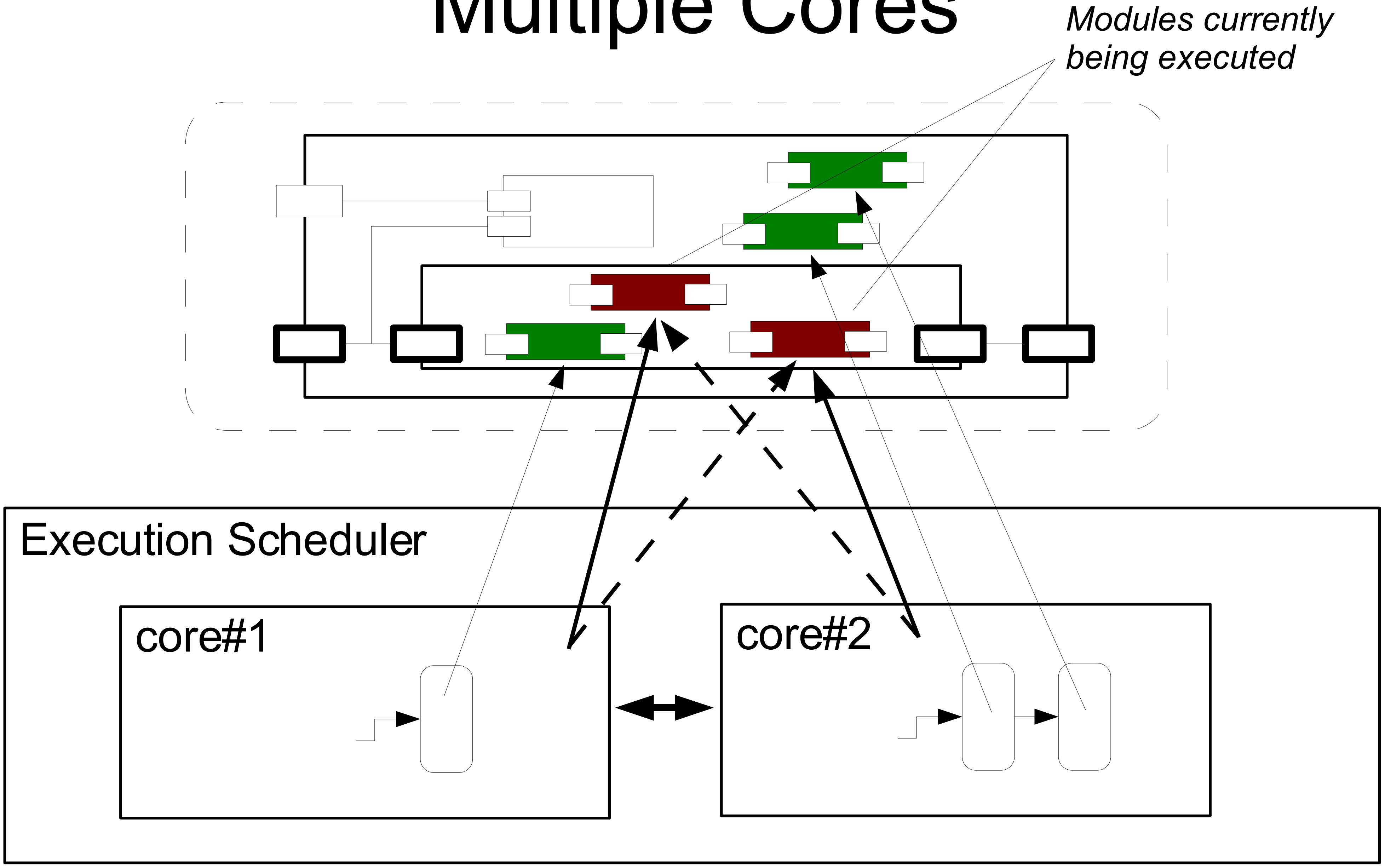
Execution Scheduler Interface



Scheduler Modules on Individual Processor Cores



Application Execution on Multiple Cores



Related Work

- Component-based programming
  - The Fractal component model
- Message-passing languages and systems
  - Erlang, ..
- The Actor programming model
- Real-Time Object-Oriented Modeling (ROOM)
- Work-stealing load-balancing
- Scala

Conclusions

- A novel message-passing programming framework for data-flow software systems
- “lean”, focusing on separation of architecture specification, deployment, and execution scheduling
- First-class architecture representation is the key for application-specific scheduling
- Future work: deployment and execution control abstractions, and scheduling policies