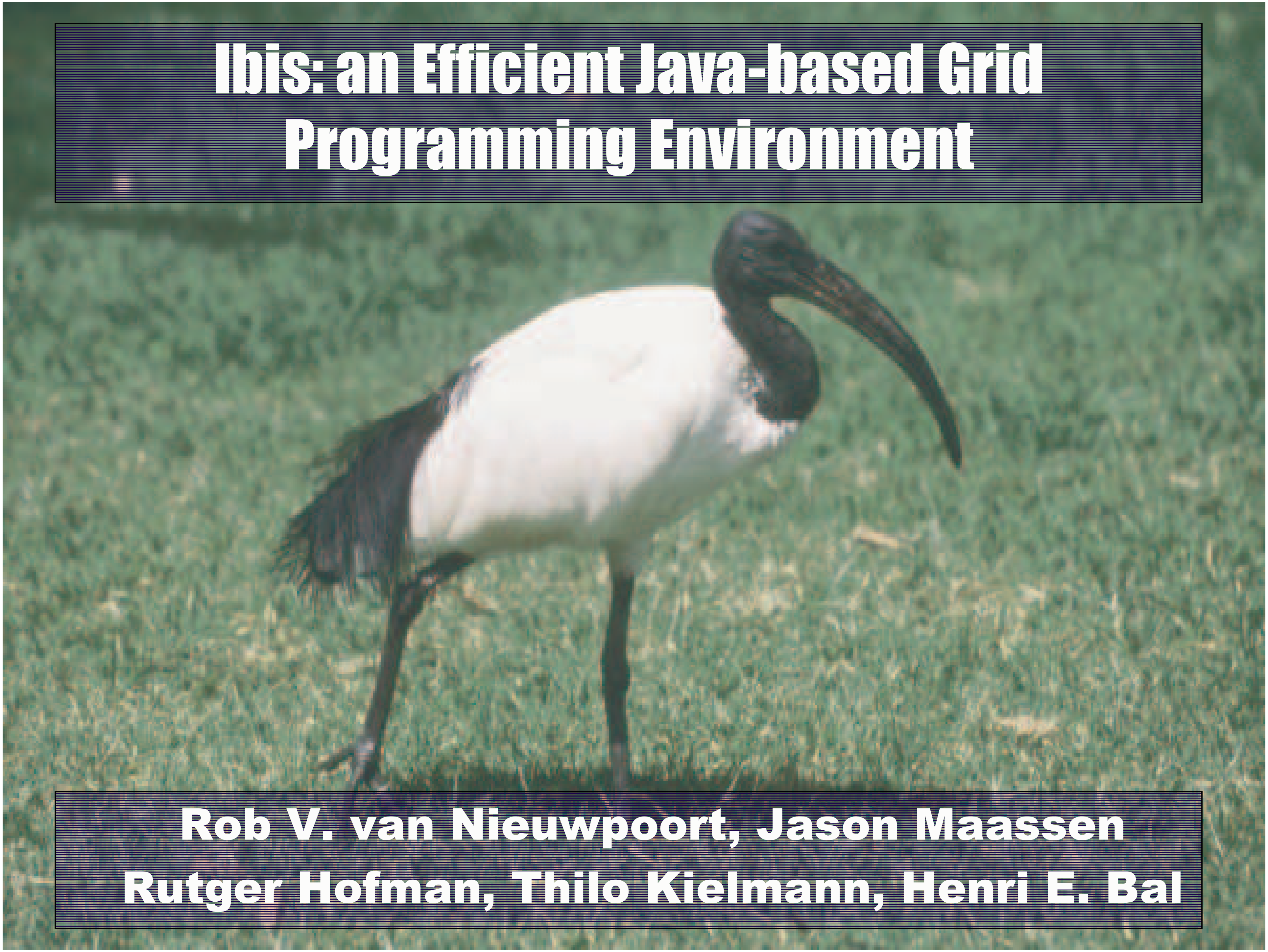
Ibis: an Efficient Java-based Grid Programming Environment



Rob V. van Nieuwpoort, Jason Maassen
Rutger Hofman, Thilo Kielmann, Henri E. Bal

Introduction

- Goal: distributed supercomputing
- Grid programming environments
  - Portable, support for heterogeneity
  - Flexible (multiple protocols, malleable)
  - Convenient programming model(s)
  - Efficient
- Problem: No single programming environment combines all features

C/Fortran and MPI on the grid

- Portable only in the traditional sense
- Limited to a SPMD-style programming model
  - No RPC (implicit receive)
  - No dynamic data structures
  - Does not fit in object oriented programming model
- But efficient !

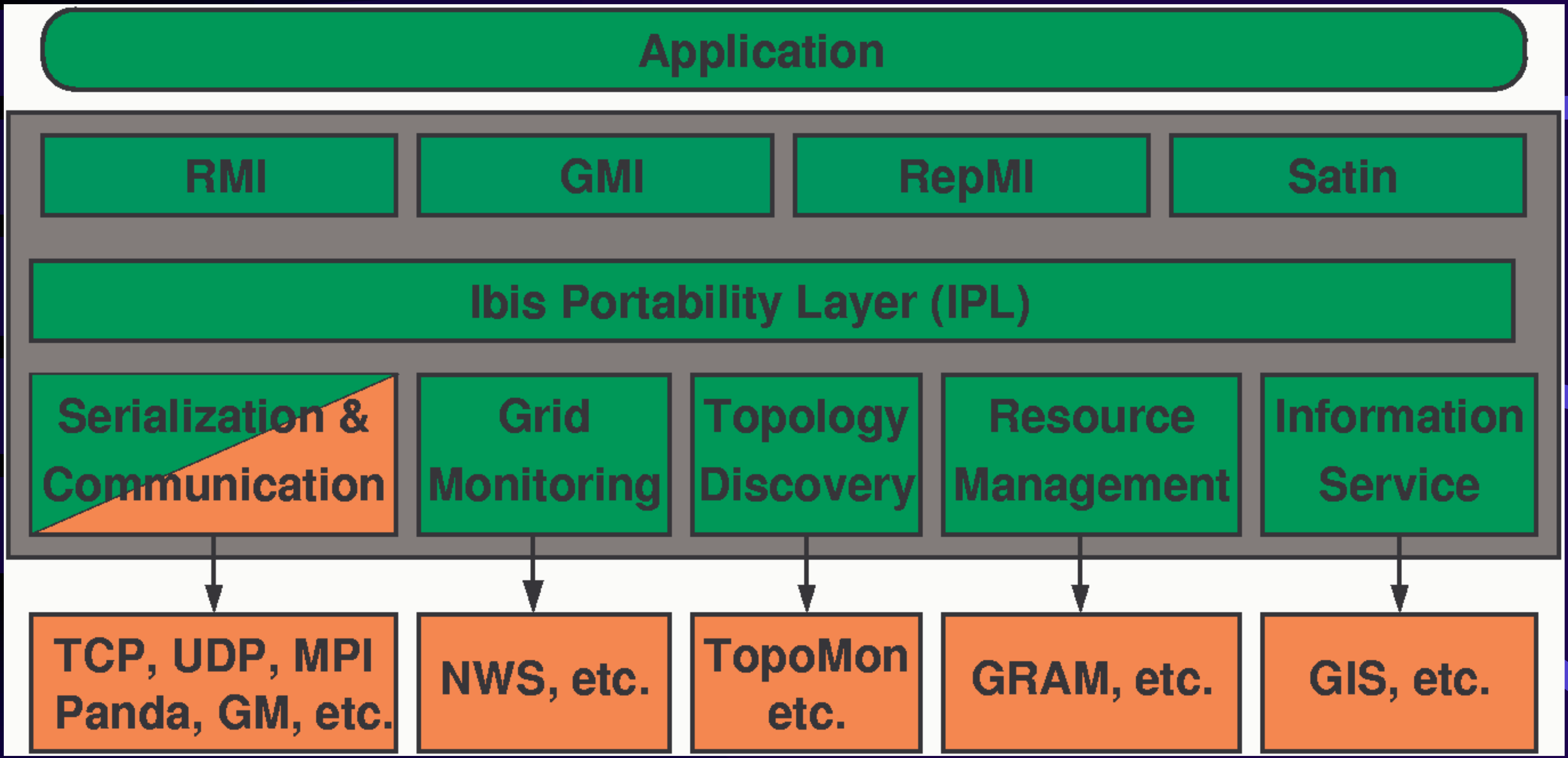
Java on the grid

- Virtual machine approach / set of libraries
  - "Write once, run everywhere" portability
- Communication
  - RMI is flexible: serialization, polymorphism
  - Only synchronous and point-to-point
  - Sub-optimal performance
- Many Java-based grid projects, but they lack either flexibility or portability or performance

The Ibis approach

- Java-based middleware and programming models, combining in one system:
  - Reasonable efficient "run everywhere" solution
  - Very efficient solutions for special cases
    - Native Myrinet communication
    - Manta native compiler
- One common interface to the hardware (IPL)
- Negotiate properties at runtime and dynamically load the best components

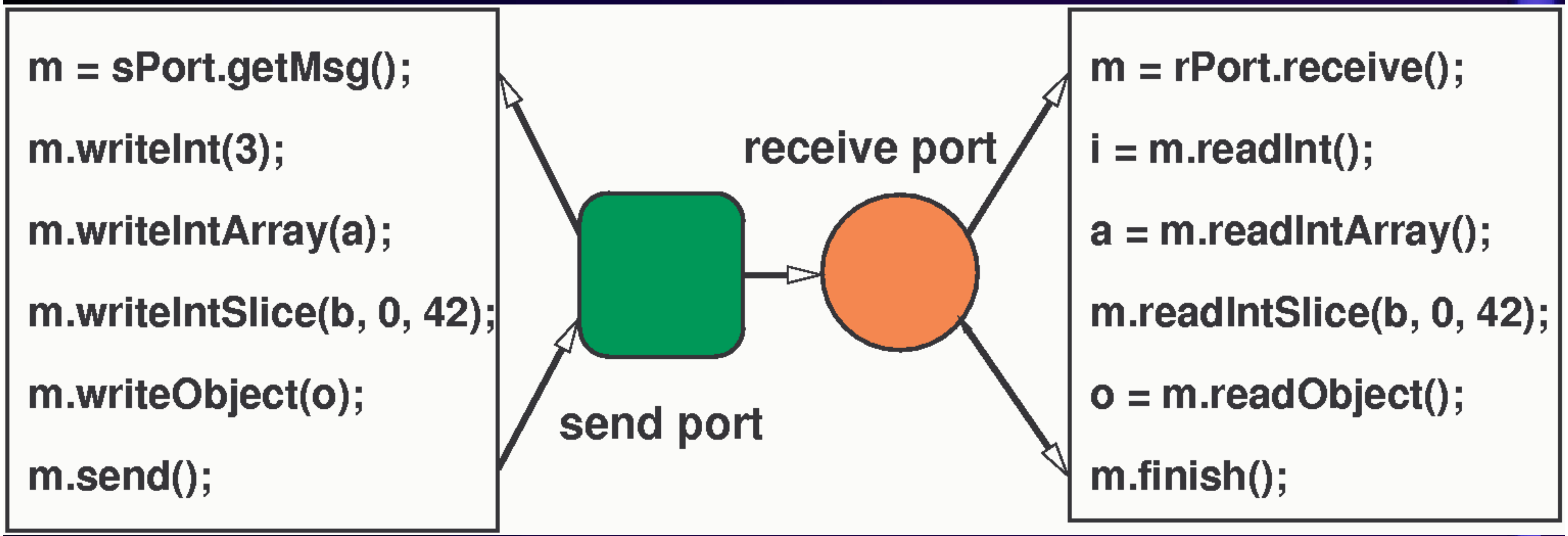
Ibis structure



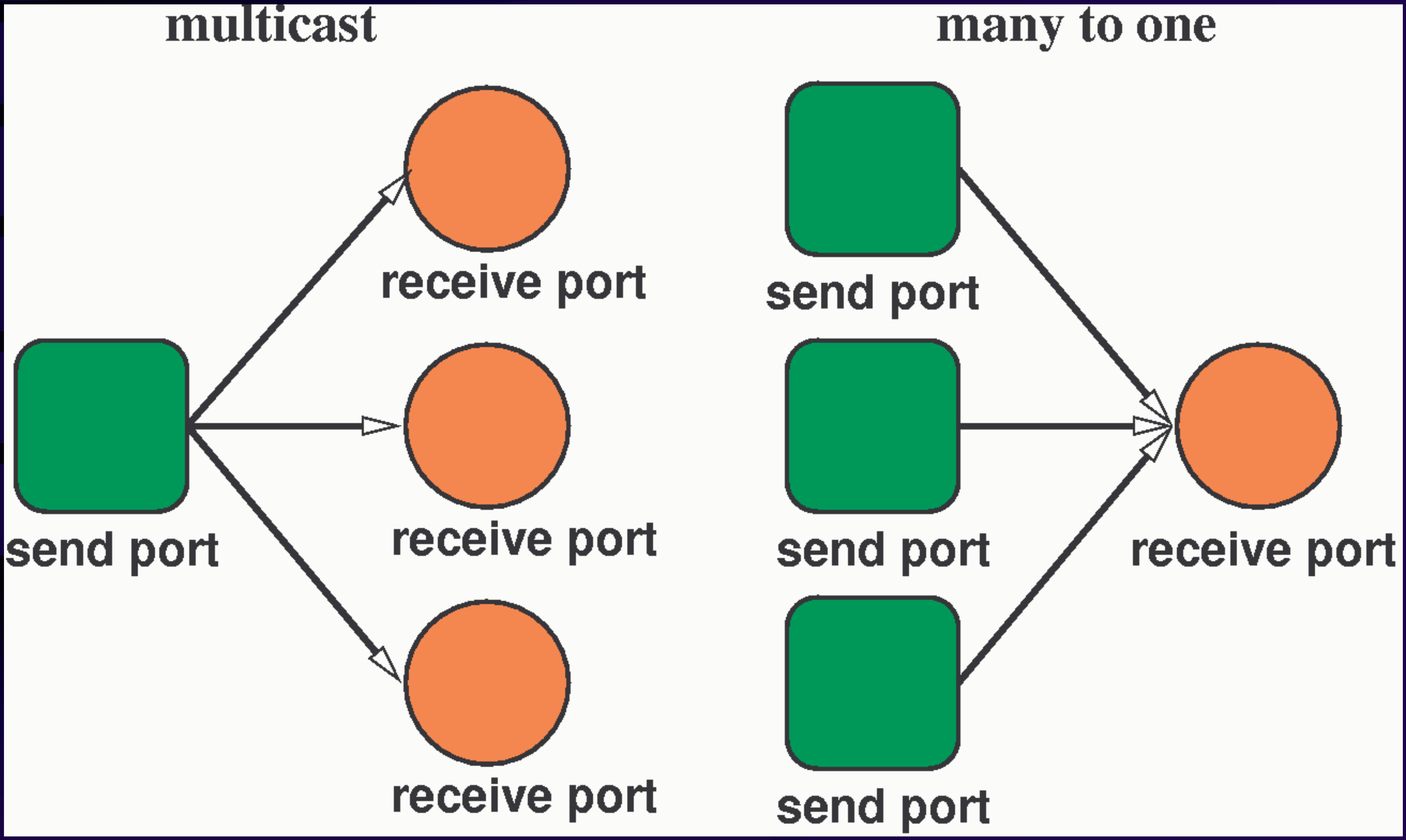
Challenges

- □ow to make the system flexible enough
  - Run seamlessly on different hardware / protocols
  - When efficient hardware primitives are available, make it possible to use them
- Make the □□□□ Java solution efficient enough
  - Serialization and communication
- Study which additional optimizations can be done to improve performance in special cases
  - JIT is black box

IPL: send ports and receive ports

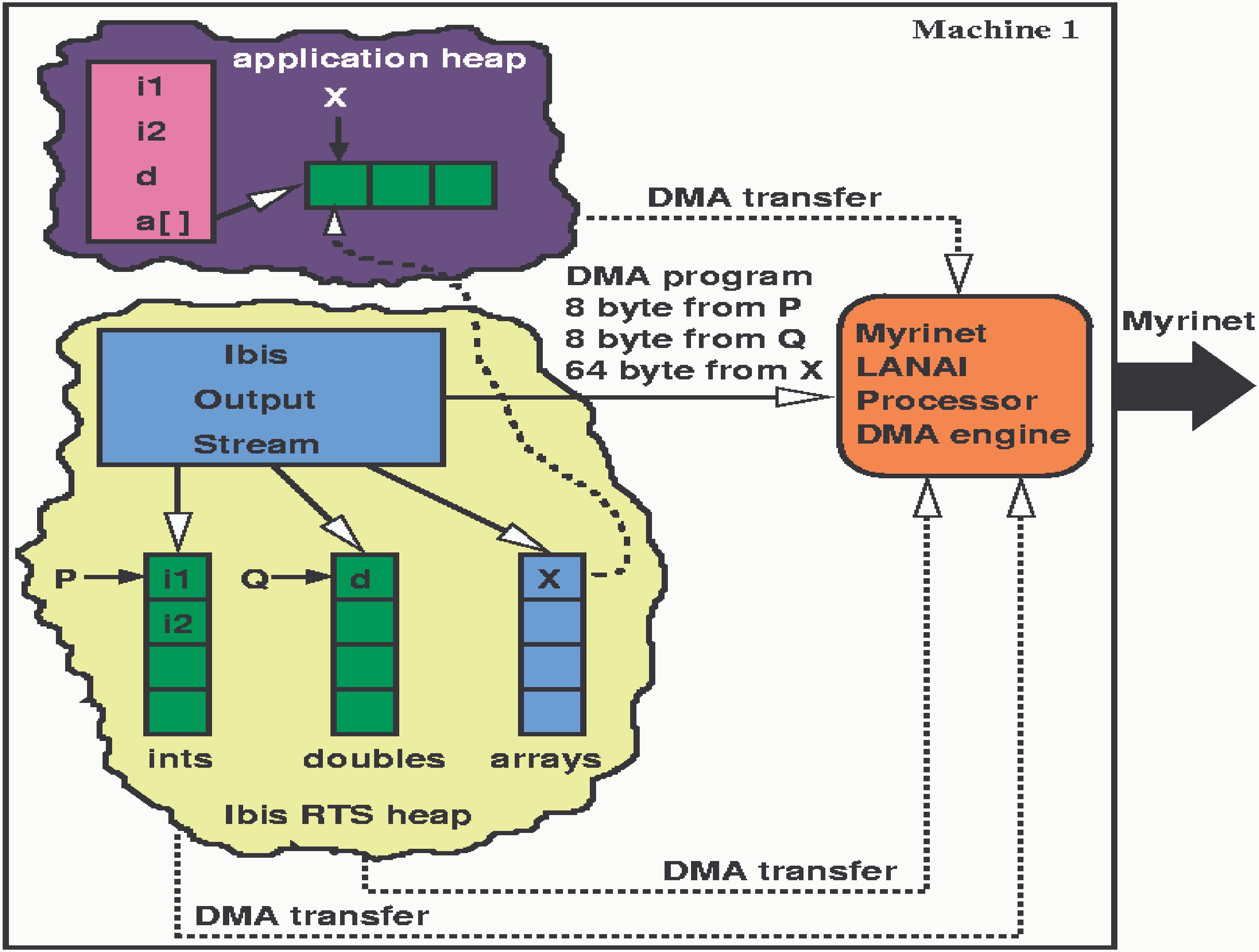


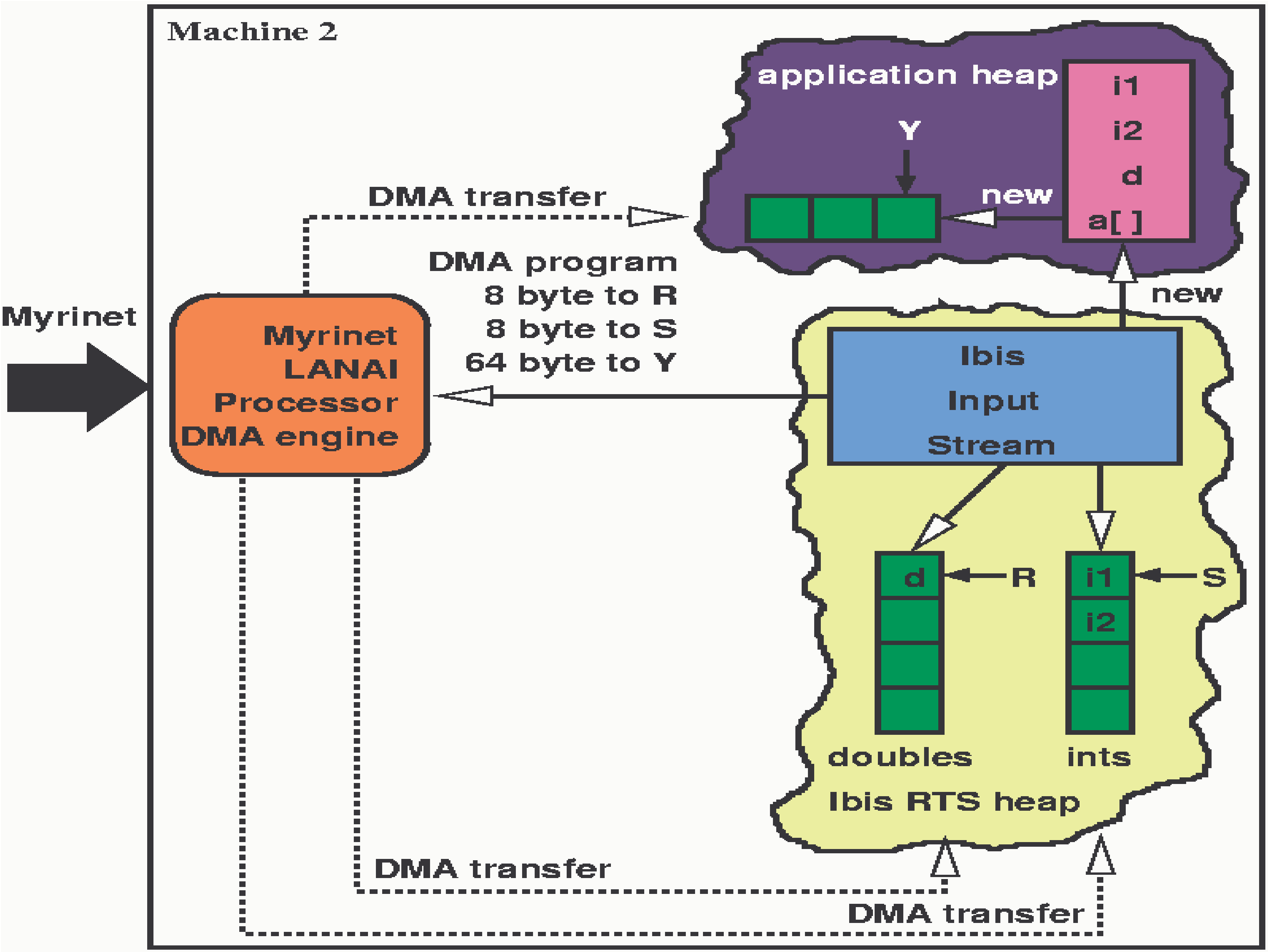
Other communication patterns



Serialization and communication

- □void runtime type inspection
  – Compiler generated serialization
  – □ses Java bytecode for portability
- Reduce data copying and conversion with typed buffers for native implementations
  – □ero copy implementation for primitive arrays
  – One copy for objects
  – Depending on JIT


Machine 1
application heap
i1
i2
d
a[ ]
X
DMA transfer
DMA program
8 byte from P
8 byte from Q
64 byte from X
Myrinet
LANAI
Processor
DMA engine
Myrinet
Ibis
Output
Stream
P
i1
i2
Q
d
X
ints
doubles
arrays
Ibis RTS heap
DMA transfer
DMA transfer



Machine 2
application heap
Y
new
i1
i2
d
a[ ]
DMA transfer
DMA program
8 byte to R
8 byte to S
64 byte to Y
new
Myrinet
Myrinet
LANAI
Processor
DMA engine
Ibis
Input
Stream
d
R
i1
S
i2
doubles
ints
Ibis RTS heap
DMA transfer
DMA transfer

Communication performance

network		low level		RMI performance		
		Ibis	MPI/C	Java	KaRMI	Ibis
Fast	latency	126	161	218.3	127.9	131.3
Ethernet	throughput int [ ]	9.6	11.1	9.5	9.6	9.6
	throughput tree	5.8	5.4	2.2	2.3	4.3
	Latency	33	22	n.a.	32.2	42.2
Myrinet	throughput int [ ]	122	143	n.a.	45.6	76.0
	throughput tree	23.6	16.6	n.a.	2.5	22.9

Conclusions

- Ibis: high performance computing for grids
  - Efficient object-based communication combined with “write one, run everywhere”
    - Reasonable efficient “run everywhere” solution
    - Very efficient solutions for special cases
- Interface that allows dynamic selection of exactly the right functionality
- Streaming, compiler generated serialization
- □ero copy for native implementations

Future work

- Write a □DP implementation for even better "run everywhere performance"
- More research on "special cases"
- Investigate adaptivity, malleability and fault tolerance for the Ibis programming models
- Integrate with existing grid software

www.cs.vu.nl/ibis