Ibis: A Java-based grid programming environment

Henri Bal

Vrije Universiteit Amsterdam

vrije Universiteit

Euro-Par 2003, Klagenfurt, 29 August

Life among animals

Orca 1985-1997 Object-based parallel computing

Manta 1997-2002 High-performance parallel Java

Albatross 1996-2002 Wide-area parallel computing

+

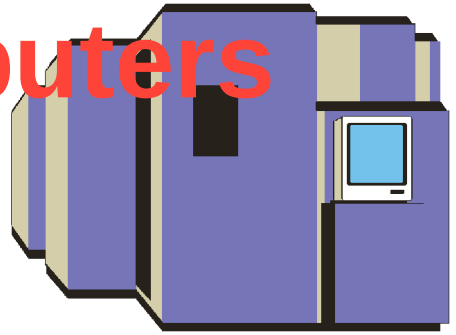
Ibis 2002-… Distributed supercomputing on grids in Java

Distributed supercomputing

- Parallel processing on geographically distributed computing systems (grids)
- Examples:
  - SETI@home (), RSA-155, Entropia, Cactus

- Currently limited to trivially parallel applications
- Our goals:
  - Generalize this to more HPC applications
  - Provide high-level programming support

Grids versus supercomputers

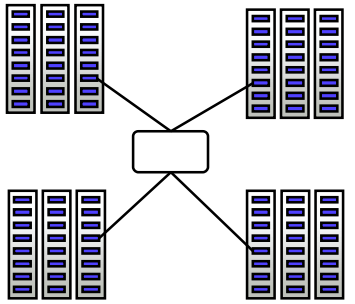


- Performance/scalability
  - Speedups on geographically distributed systems?
- Heterogeneity
  - Different types of processors, operating systems, etc.
  - Different networks (Ethernet, Myrinet, WANs)
- General grid issues
  - Resource management, co-allocation, firewalls, security, monitoring, authorization, accounting, ….

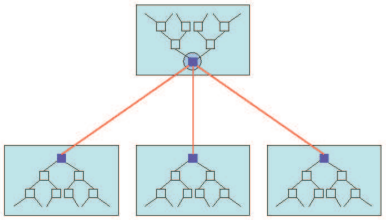
Our approach

- Performance/scalability
  - Exploit hierarchical structure of grids (Albatross project)
- Heterogeneity
  - Use Java + JVM (Java Virtual Machine) technology
- General grid issues
  - Import knowledge from elsewhere (GGF, GridLab)

Speedups on a grid?



- Grids usually are hierarchical
  - Collections of clusters, supercomputers
  - Fast local links, slow wide-area links
- Can optimize algorithms to exploit this hierarchy
  - Message combining + latency hiding on wide-area links
  - Collective operations for wide-area systems
  - Load balancing



- Successful for many applications
  - Did many experiments on a homogeneous wide-area test bed (DAS) [HPCA 1999, IEEE TPDS 2002]

The Ibis system

- High-level & efficient programming support for distributed supercomputing on heterogeneous grids
- Use Java-centric approach + JVM technology
  - Inherently more portable than native compilation
    "*Write once, run everywhere*"
  - Requires entire system to be written in Java
- Use special-case (native) optimizations on demand

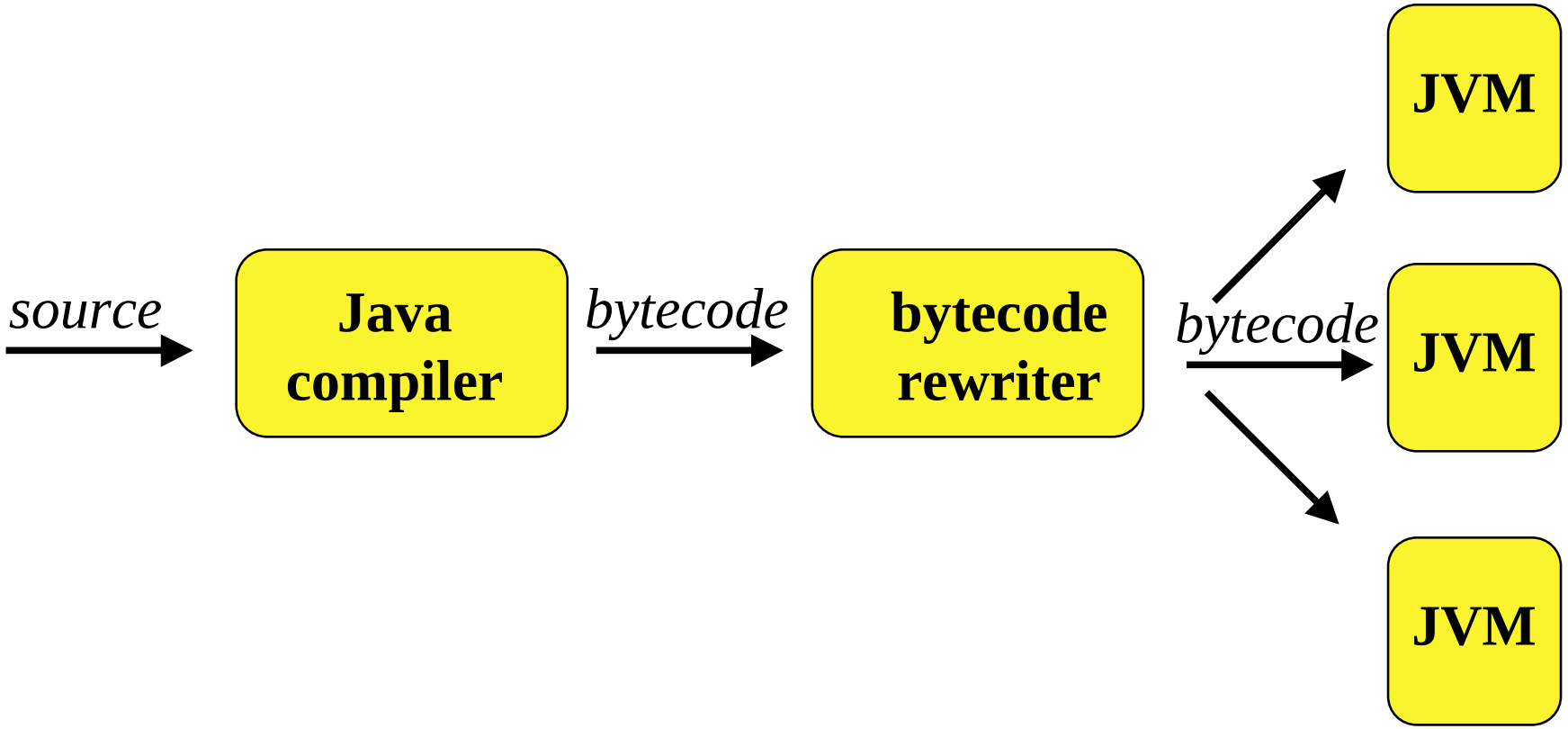
Outline

- Programming support
- Highly portable & efficient implementation
- Experiences on DAS-2 and EC GridLab testbeds

Ibis programming support

- Ibis provides
  - Remote Method Invocation (RMI)
  - Replicated objects (RepMI) - as in Orca
  - Group/collective communication (GMI) - as in MPI
  - Divide & conquer (Satin) - as in Cilk
- All integrated in a clean, object-oriented way into Java, using special “marker” interfaces
  - Invoking native library (e.g. MPI) would give up Java’s “run everywhere” portability

Compiling Ibis programs



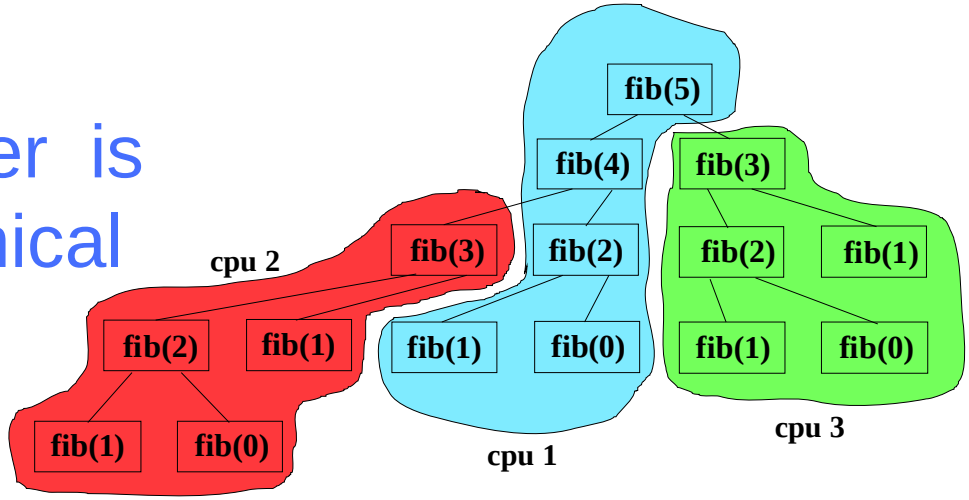
GMI (group communication)

- Generalizes Remote Method Invocation
  - Modify how invocations & results are handled
  - Invoke multiple objects, combine/gather results, etc.
  - Expresses many forms of group communication

```
interface Example extends GroupInterface {
        public int get() throws ...;   ….
}
Example e =  ;                          // get group stub
m = findMethod(”int get()", e); // method to configure
m.useGroupInvocation();  // get() will be multicast
m.useCombineResult(…);   //results are combined
int result = e.get();    // example invocation
```

Divide-and-conquer parallelism

- Divide-and-conquer is inherently hierarchical



- Satin
  - Cilk-like primitives (spawn/sync)
- New load balancing algorithm
  - Cluster-aware random work stealing [PPoPP'01]

Example

```
interface FibInter {
      public int fib(long n);
}

class Fib implements FibInter {
   int fib (int n) {
      if (n < 2) return n;
      return fib(n-1) + fib(n-2);
   }
}
```

Single-threaded Java

Example

```
interface FibInter
    extends ibis.satin.Spawnable {
        public int fib(long n);
}

class Fib
  extends ibis.satin.SatinObject
  implements FibInter {
    public int fib (int n) {
        if (n < 2) return n;
        int x = fib (n - 1);
        int y = fib (n - 2);
        sync();
        return x + y;
    }
}
```

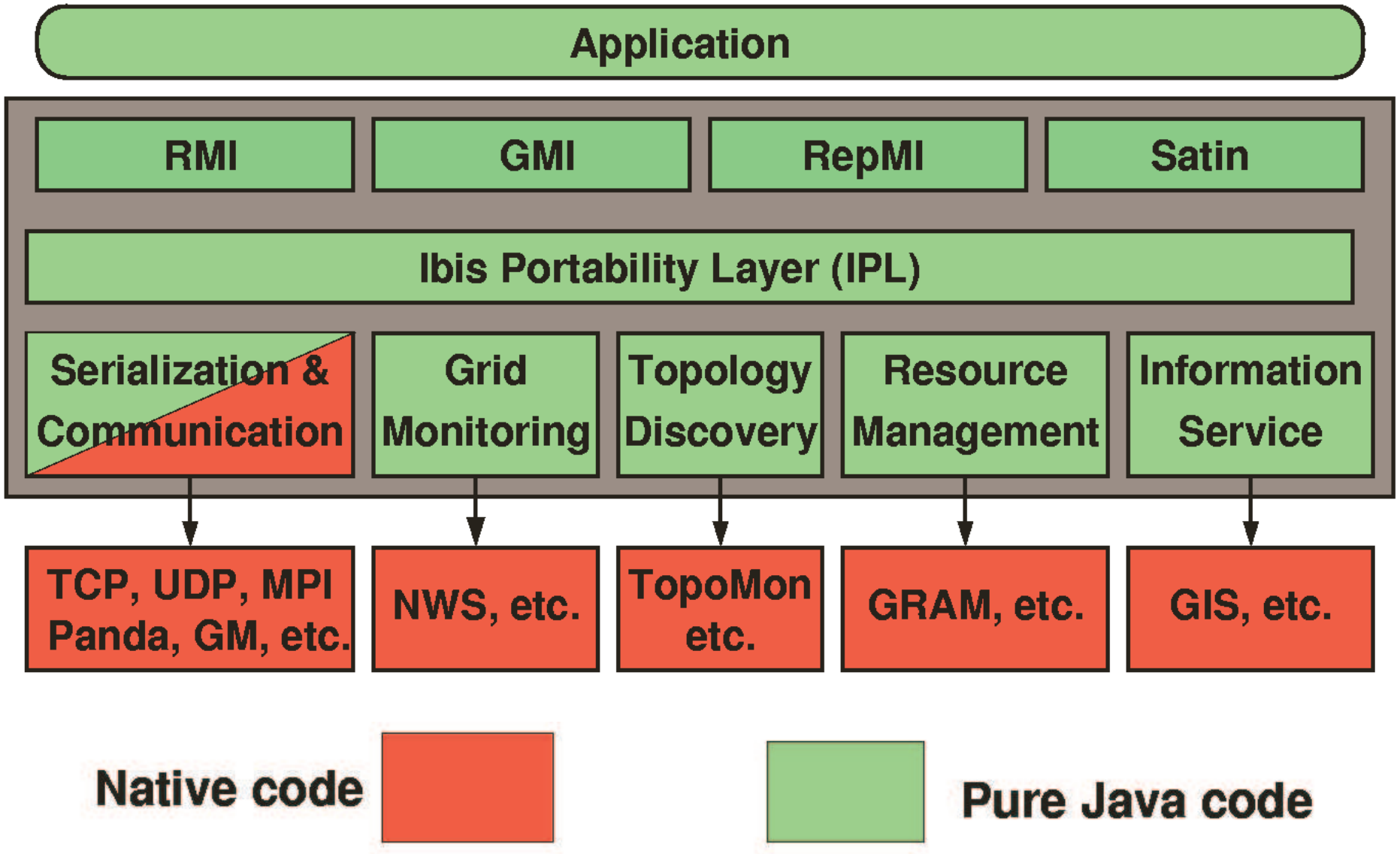
Java + divide&conquer

GridLab testbed

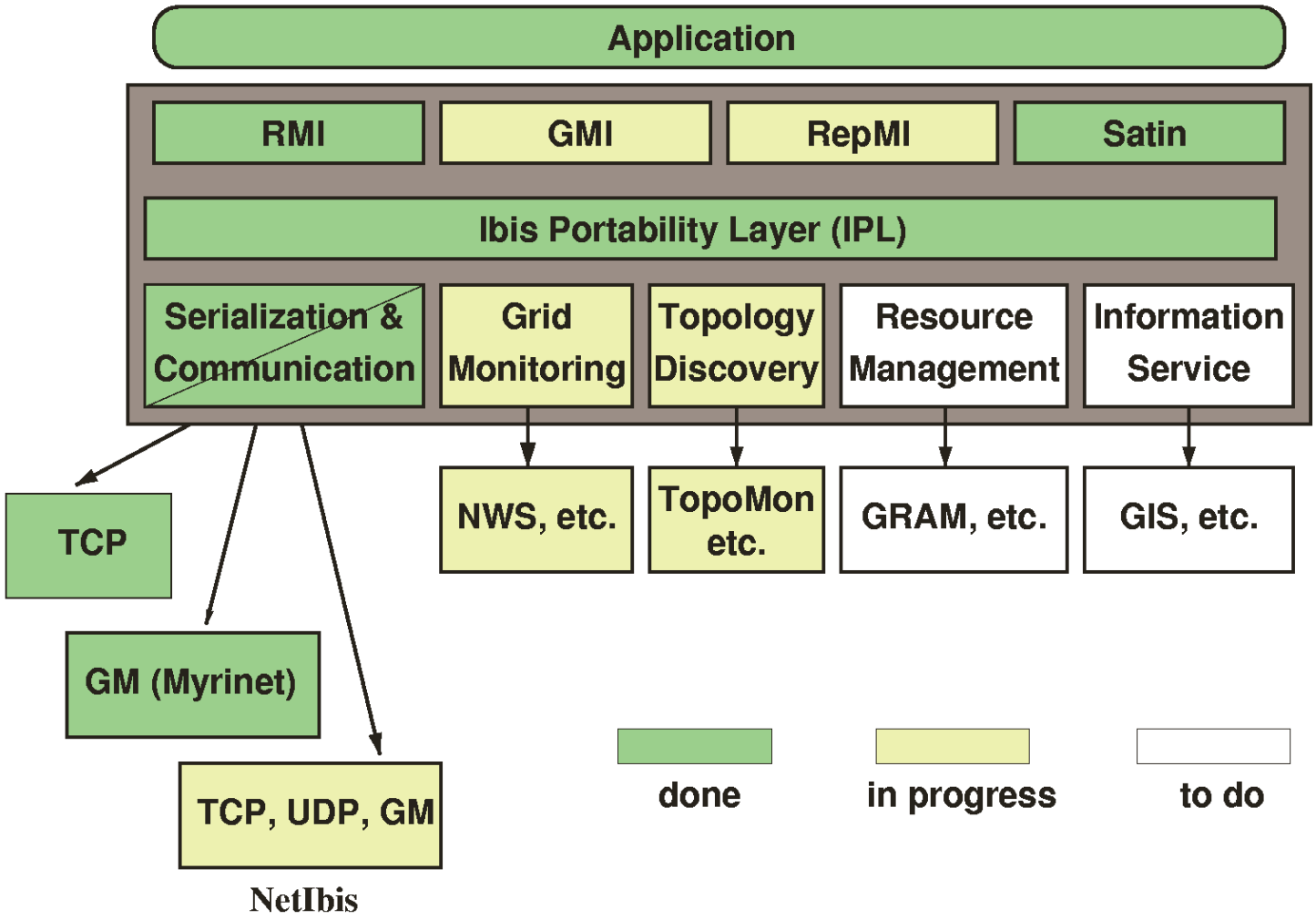
Ibis implementation

- Want to exploit Java's "run everywhere" property, **but**
  - That requires 100% pure Java implementation, **no** single line of native code
  - Hard to use native communication (e.g. Myrinet) or native compiler/runtime system
- Ibis approach:
  - Reasonably efficient pure Java solution (for any JVM)
  - Optimized solutions with native code for special cases

Ibis design


Application
RMI
GMI
RepMI
Satin
Ibis Portability Layer (IPL)
Serialization & Communication
Grid Monitoring
Topology Discovery
Resource Management
Information Service
TCP, UDP, MPI Panda, GM, etc.
NWS, etc.
TopoMon etc.
GRAM, etc.
GIS, etc.
Native code
Pure Java code

Current status



Challenges

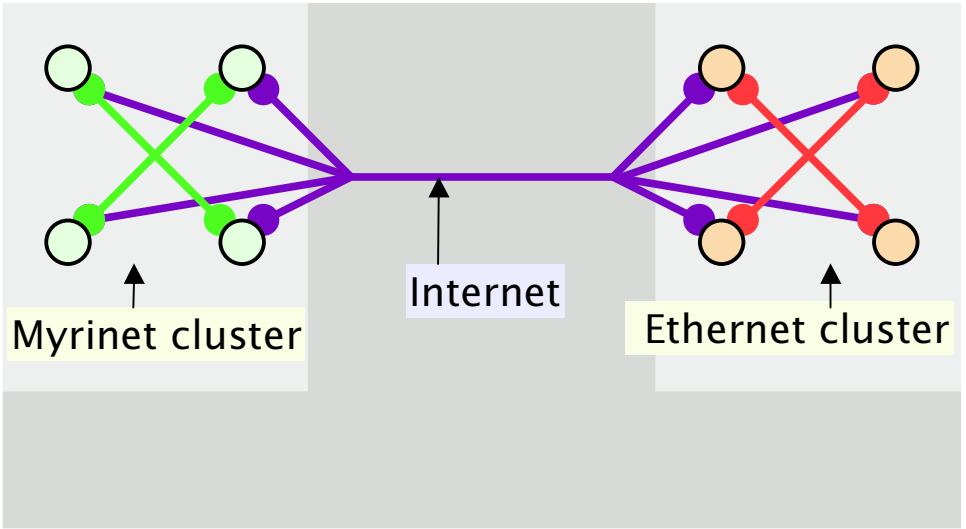
- How to make the system flexible enough
  - Run seamlessly on different hardware / protocols
- Make the pure-Java solution efficient enough
  - Need fast local communication even for grid applications
- Special-case optimizations

Flexibility

- IPL just defines an *interface* between high-level programming systems and underlying platforms
- Higher levels can ask IPL to load different *implementations* at runtime, using class loading
  - Eg FIFO ordering, reliable communication
- Support different communication substrates
  - NetIbis layer

NetIbis

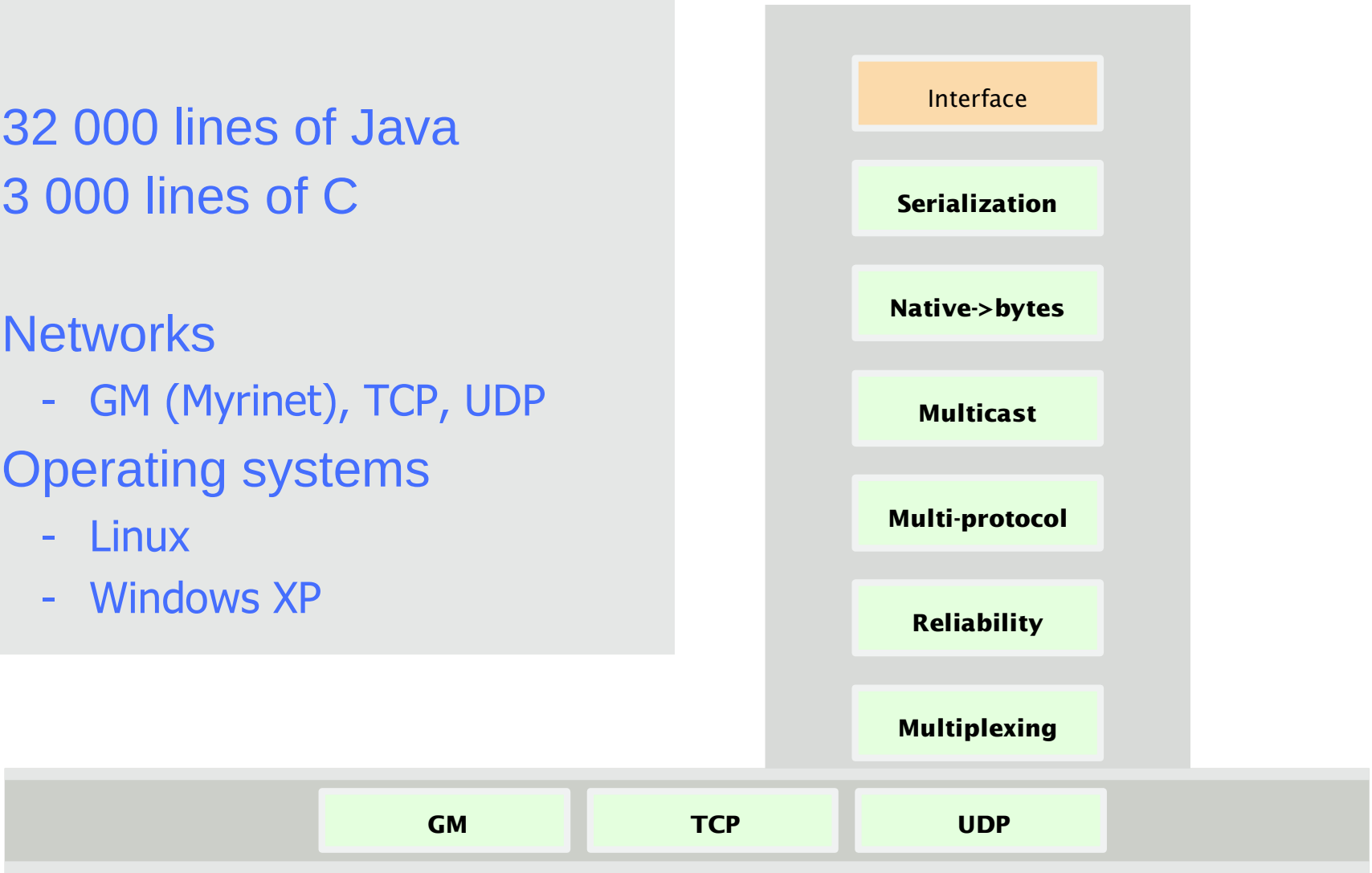
- Communication system implementing IPL
- Modular
  - Runtime configurable protocol stacks
  - Code inheritance
- Open world model
- Portable (Java)
- Efficient
  - Native drivers for special cases (e.g., Myrinet)



NetIbis Status

- 32 000 lines of Java
- 3 000 lines of C

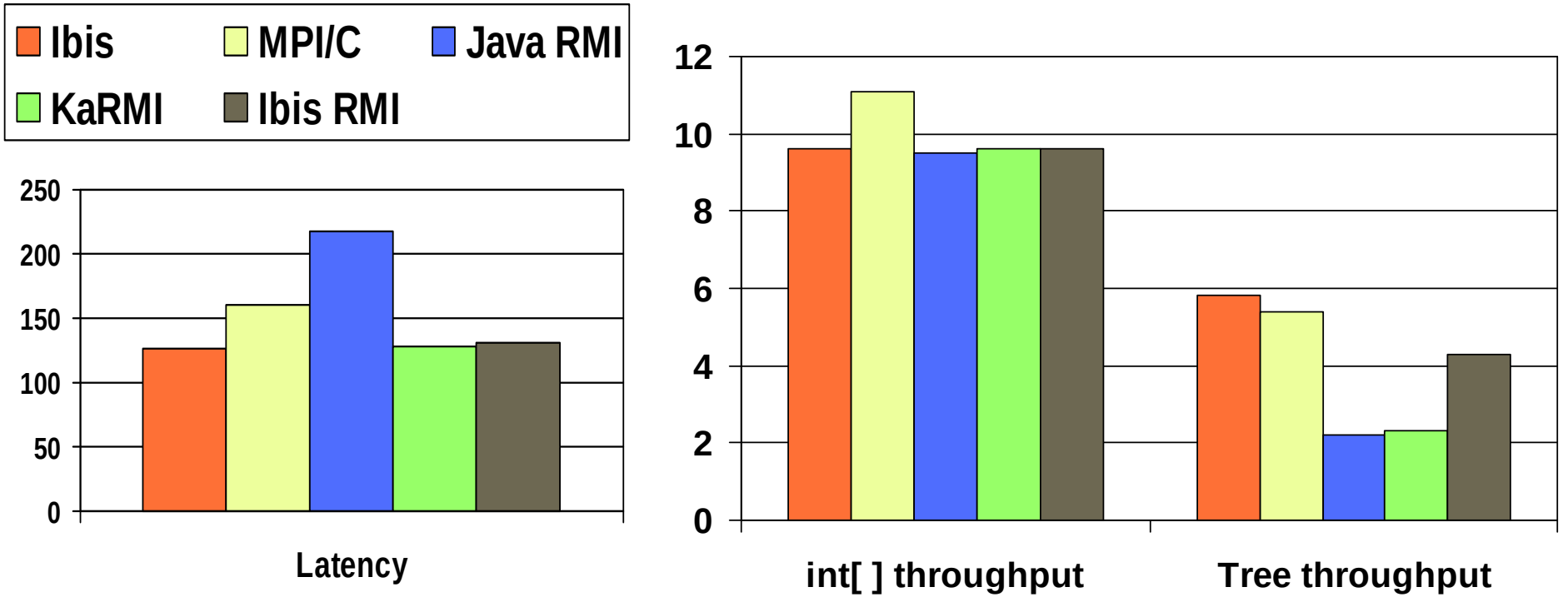
- Networks
  - GM (Myrinet), TCP, UDP
- Operating systems
  - Linux
  - Windows XP



Fast communication in pure Java

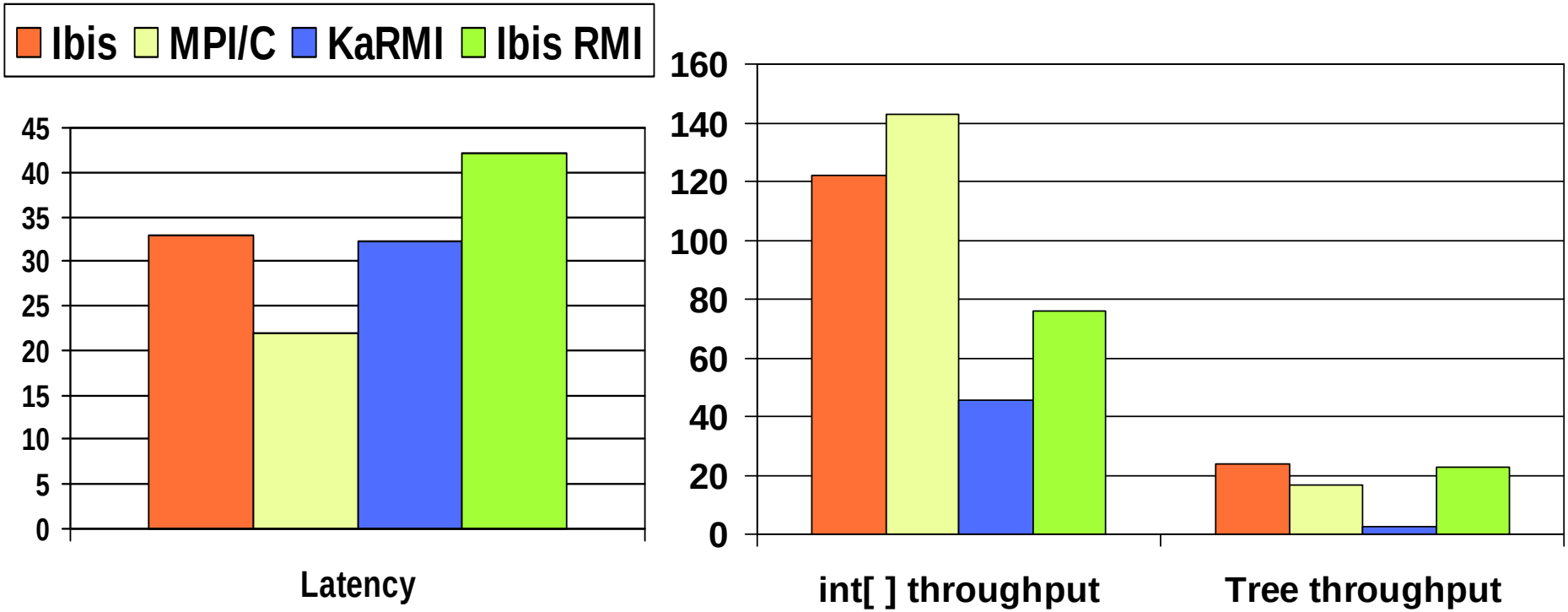
- Manta system [ACM TOPLAS Nov. 2001]
  - RMI at RPC speed, but using native compiler & RTS
- Ibis does similar optimizations, but in pure Java
  - Compiler-generated serialization at bytecode level
    5-9x faster than using runtime type inspection
  - Reduce copying overhead
    Zero-copy native implementation for primitive arrays
    Pure-Java requires type-conversion (=copy) to bytes

Communication performance on Fast Ethernet

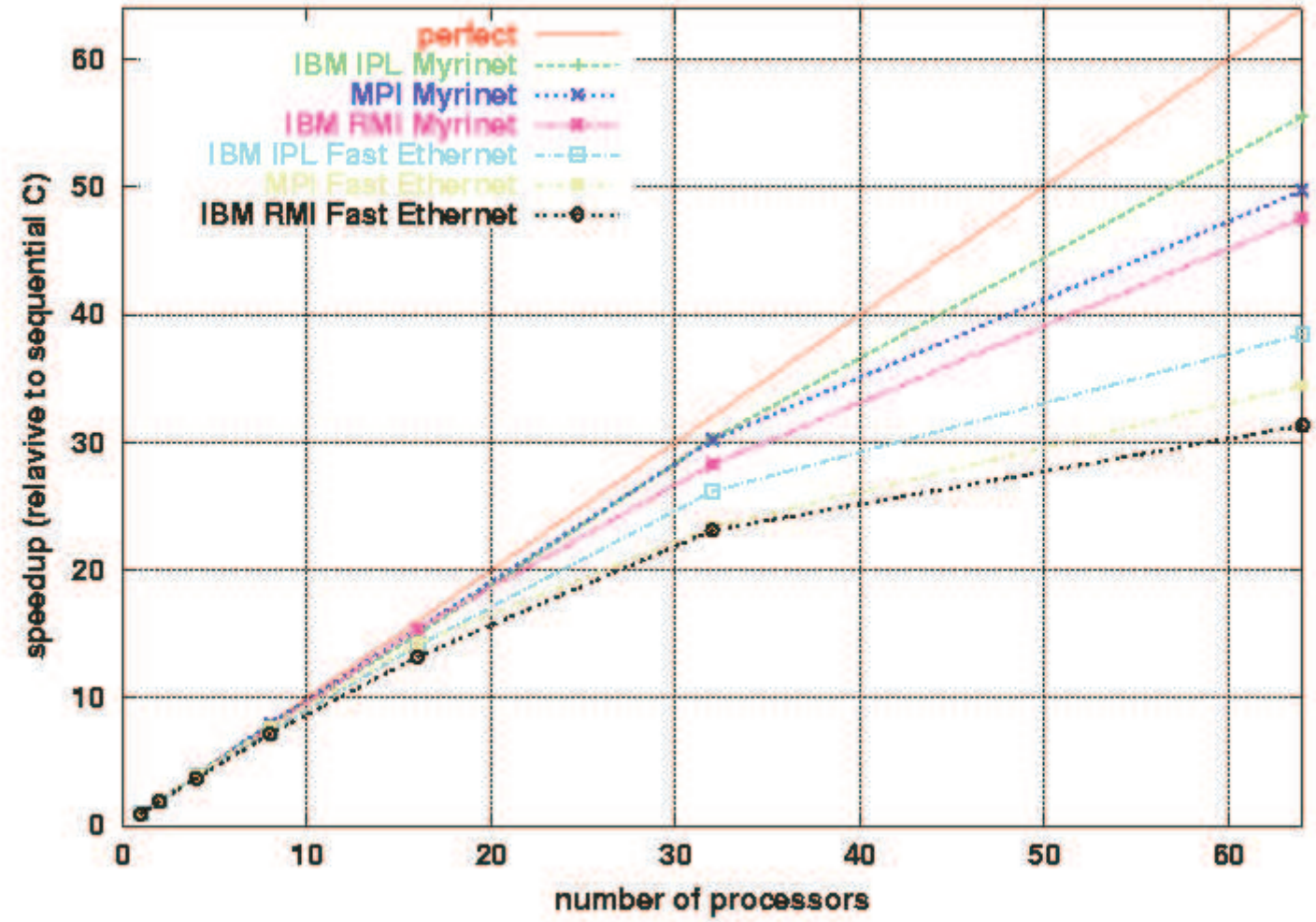


Latency (µs) & throughput (MB/s), measured on
1 GHz Pentium-IIIs (KaRMI = Karlsruhe RMI)

Communication performance on Myrinet



Java/Ibis vs. C/MPI on Pentium-3 cluster (using SOR)



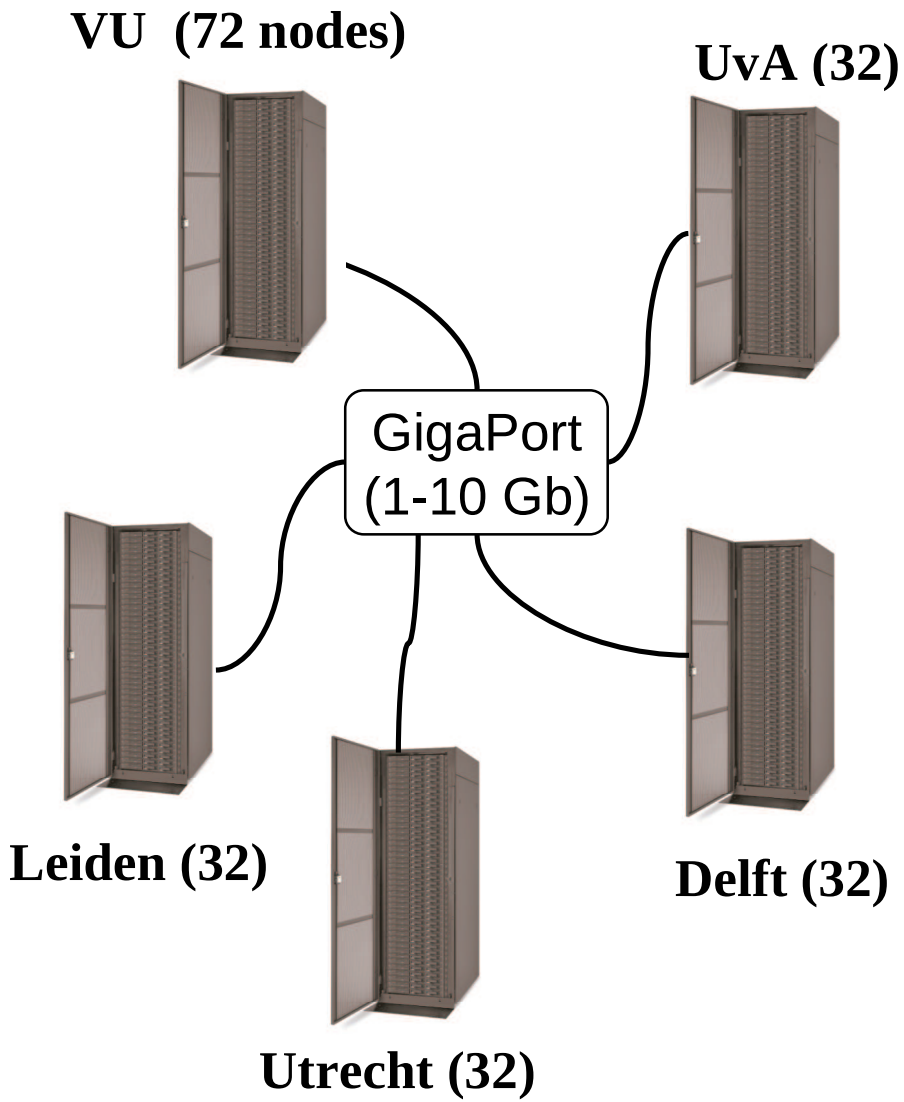
Grid experiences with Ibis

- Using Satin divide-and-conquer system
  - Implemented with Ibis in pure Java, using TCP/IP
- Application measurements on
  - DAS-2 (homogeneous)
  - Testbed from EC GridLab project (heterogeneous)

Distributed ASCI Supercomputer (DAS) 2

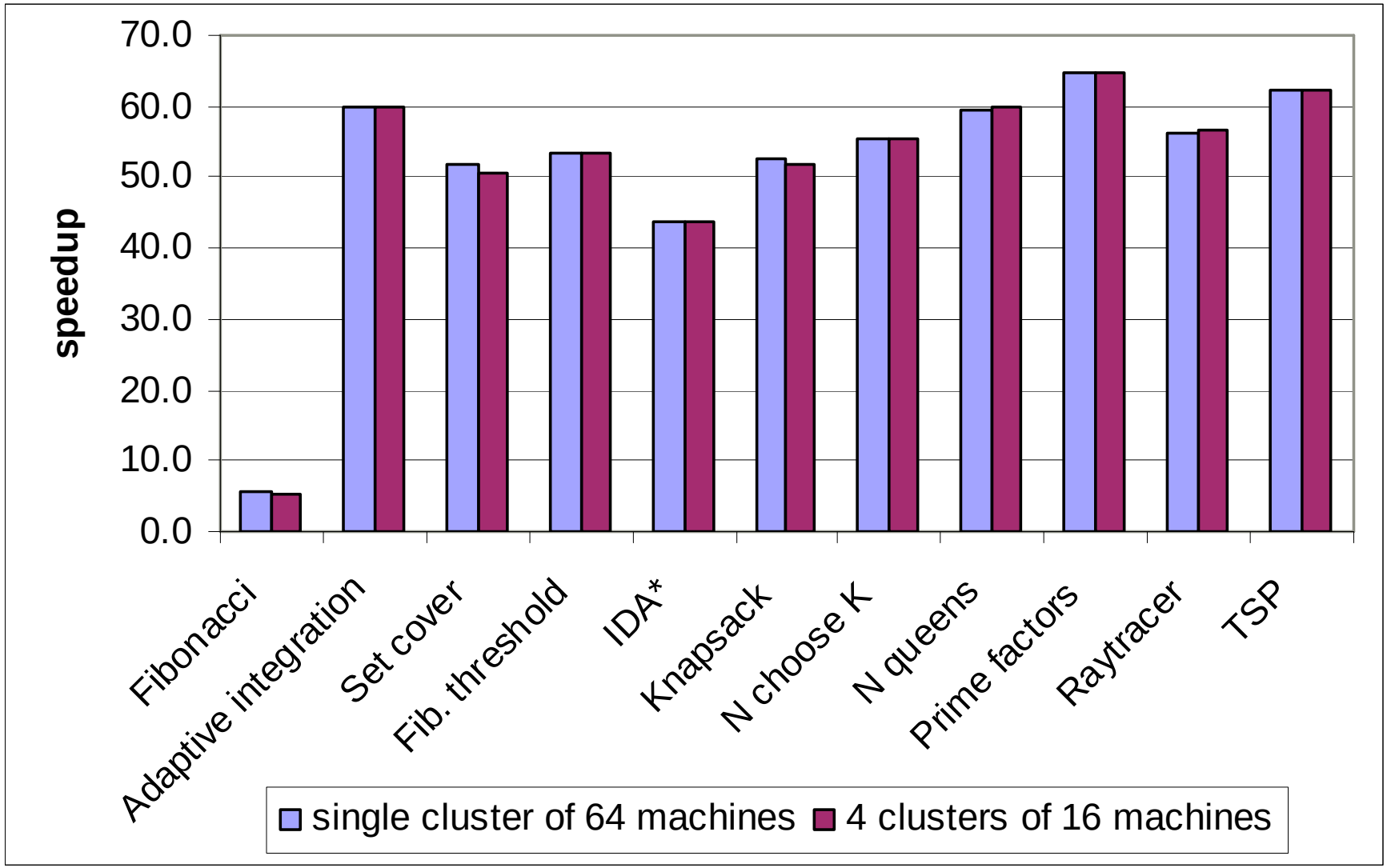
Node configuration

Dual 1 GHz Pentium-III
>= 1 GB memory
Myrinet
Linux



Advanced School for Computing and Imaging

Satin on wide-area DAS-2



Satin on GridLab

- Heterogeneous European grid testbed
- Implemented Satin/Ibis on GridLab, using TCP
- Experiments with Raytracer application

- Source: van Nieuwpoort et al., AGRIDM'03 (Workshop on Adaptive Grid Middleware, New Orleans, Sept. 2003)

GridLab

- Latencies:
  - 9-200 ms (daytime), 9-66 ms (night)
- Bandwidths:
  - 9-4000 KB/s

Configuration

Type	OS	CPU	Location	CPUs
Cluster	Linux	Pentium-3	Amsterdam	8×1
SMP	Solaris	Sparc	Amsterdam	1×2
Cluster	Linux	Xeon	Brno	4×2
SMP	Linux	Pentium-3	Cardiff	1×2
Origin 3000	Irix	MIPS	ZIB Berlin	1×16
SMP	Unix	Alpha	Lecce	1×4

Experiences

- No support for co-allocation yet (done manually)
- Firewall problems everywhere
  - Currently: use a range of site-specific open ports
  - Future: use multiplexing over ssh connections
- Java indeed runs everywhere
  modulo bugs in (old) JVMs
  - IBM 1.3.1 JIT: bug in versioning mechanism
  - Origin JDK: bug in thread synchronization
- Need clever load balancing mechanism ➔ CRS

Cluster-aware Random Stealing

- Use Cilk's Random Stealing (RS) inside cluster
- When idle
  - Send *asynchronous* wide-area steal message
  - Do random steals locally, and execute stolen jobs
  - Only 1 wide-area steal attempt in progress at a time
- Prefetching adapts
  - More idle nodes ➔ more prefetching

- Source: van Nieuwpoort et al., ACM PPoPP'01

Performance on GridLab

- Problem: how to define *efficiency* on a grid?
- Our approach:
  - Benchmark each CPU with Raytracer on small input
  - Normalize CPU speeds (relative to a DAS-2 node)
  - Our case: 40 CPUs, equivalent to 24.7 DAS-2 nodes
  - Define:

    T_perfect = sequential time / 24.7

    efficiency = T_perfect / actual runtime
  - Also compare against single 25-node DAS-2 cluster

Results for Raytracer

	Time (sec)	Efficiency (%)
Night RS	878	62.6
CRS	677	81.3
Day RS	2084	26.4
CRS	693	79.3
1 cluster	580	96.1
sequential	13,564	100

RS = Random Stealing, CRS = Cluster-aware RS

Some statistics

- Variations in execution times:
  - RS @ day: 0.5 - 1 hour
  - CRS @ day: less than 20 secs variation
- Internet communication (total):
  - RS: 11,000 (night) - 150,000 (day) messages
    137 (night) - 154 (day) MB
  - CRS: 10,000 - 11,000 messages
    82 - 86 MB

Summary

- Ibis: a programming environment for grids
  - RMI, group communication, divide&conquer
- Portable
  - Using Java's "write once, run everywhere" property
- Efficient
  - Reasonably efficient "run everywhere" solution
  - Optimized solutions for special cases
- Experience with prototype system on GridLab

Current/future work on Ibis

- Fault tolerance
  - Automatic checkpointing/restart mechanism (based on Brakes)
  - Fault-tolerant Satin
- Malleability (nodes join/leave dynamically)
- Porting other high-level systems on top of Ibis
  - INRIA's ProActive system (object mobility)

Future project?

Acknowledgements

Rob van Nieuwpoort

Jason Maassen

Thilo Kielmann

Rutger Hofman

Ceriel Jacobs

Gosia Wrzesinska

Olivier Aumage

Kees Verstoep

Maik Niehuis

web site: www.cs.vu.nl/ibis