

# Grid Programming Models: Requirements and Approaches

*Thilo Kielmann*

Vrije Universiteit, Amsterdam

[kielmann@cs.vu.nl](mailto:kielmann@cs.vu.nl)



# Programming Models

## Computer scientists:

- Dedicate their lives to them
- Get Ph.D.'s for them
- Love them

## Application programmers:

- Want to get their work done
- Choose the smallest evil

# Types of Application Programmers

Patrick Geoffrey (Myricom) said:

- Parallel programmers are from Mars
- ... and Grid programmers are from Melmac!

# Newsflash from Melmac:

MPI sucks!



## Programming Models (2)

### Single computer (a.k.a. sequential)

- Object-oriented or components

- High programmer productivity through high abstraction level

### Parallel computer (a.k.a. cluster)

- Message passing

- High performance through good match with machine architecture

## Programming Models (3)

### Grids (a.k.a. Melmac)

– ???

- Fault-tolerance
- Security
- Platform independence
- ...

## Grid Platforms (1)

Writing Grid applications means programming against the interface(s) of the respective middleware:

- Globus 2.x (C-based services, proprietary protocols and interfaces), the “de-facto standard”
- Globus 3.1 (Grid services, OGSA/OGSI, Java-based) outdated before it was widely deployed
- Globus 4.0.x Web services, WSRF-based, Java services

## Grid Platforms (2)

- **Unicore, started as what today would be called “portal”**
  - no real “API”, tries to hide the Grid from the applications
  - Started with proprietary protocols and formats, now Web services
- **Avaki, commercial version of the Legion project**
  - Now “Sybase Avaki Enterprise Information Integration System”
  - Data access interfaces (data bases via Web services)



## Grid Platforms (3)

### – Condor-G

- Condor's high throuput computing, job submission via Globus
- No explicit API and interfaces  
(Condor hides remoteness of execution)

### – OMII-UK: Open Middleware Infrastructure Institute UK

- Web services for remote compute/data access
- Aims to provide the SAGA API to its clients

## Grid Platforms (4)

### – gLite

- DataGrid / EGEE projects provided set of services
- Proprietary API's / interfaces

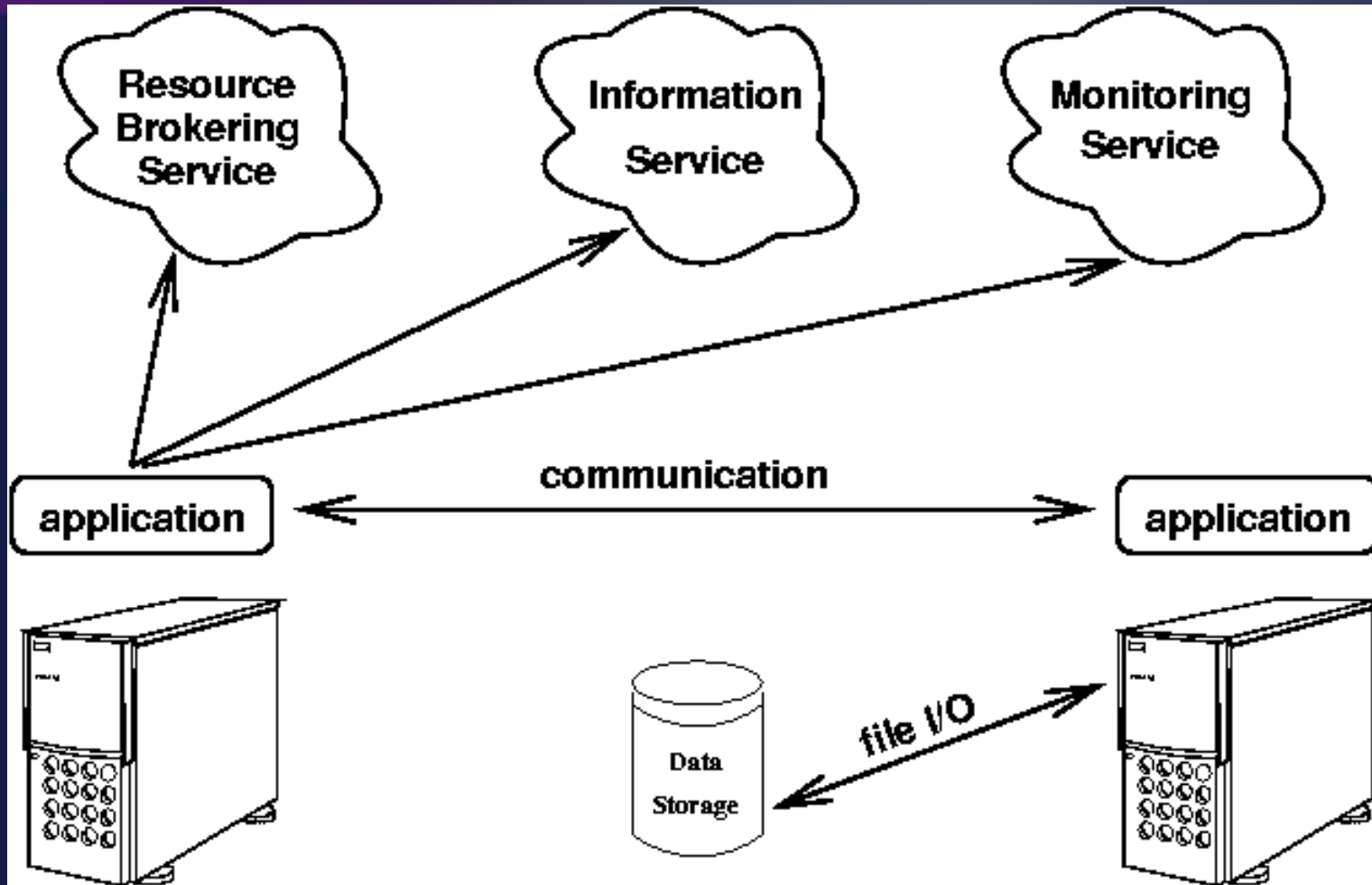
### – NAREGI (Japanese NATIONAL REsearch Grid Infrastructure)

- (OGSA) services to build virtual, integrated supercomputer
- Provides GridMPI and GridRPC interfaces

### – ssh

- Minimalistic approach (e.g. used in PlanetLab)

# A Grid Application Execution Scenario



# Applications' View: Functional Properties

What applications need to do:

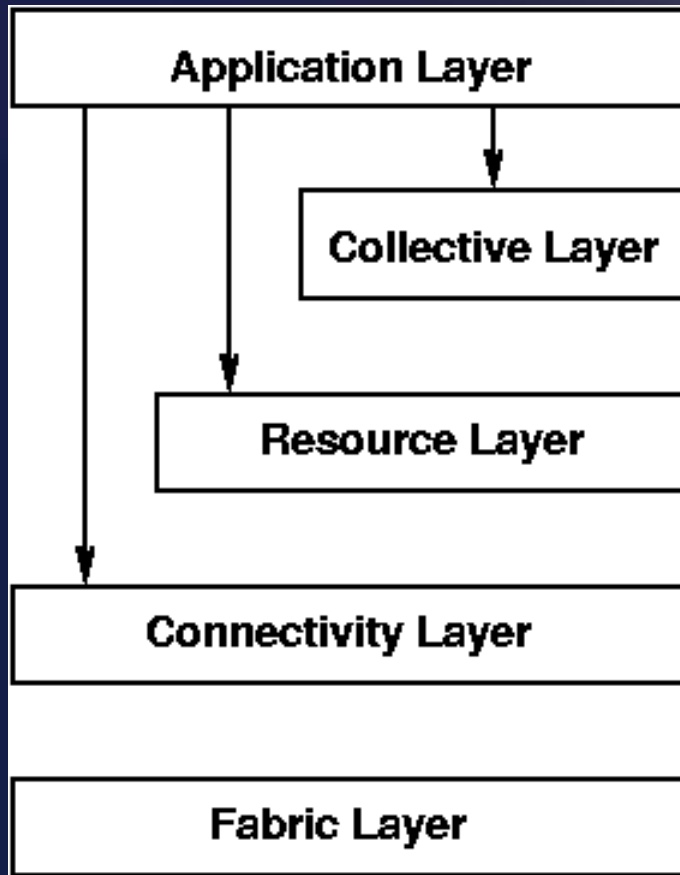
- Access to compute resources, job spawning and scheduling
- Access to file and data resources
- Communication between parallel and distributed processes
- Application monitoring and steering

# Applications' View: Non-functional Properties

What else needs to be taken care of:

- Performance
- Fault tolerance
- Security and trust
- Platform independence

# Middleware's View: (from: Foster et al., "Anatomy of the Grid")



OGSA: execution, data, res.mgmt., security, info., self mgmt., MPI...

Monitoring of + information about resources (resource access control)

Network conn., authentication

“The hardware”

## Features: Application vs. Middleware

<i>Application View</i>	<i>Feature</i>	<i>Middleware View</i>
Application	Monitoring/Info	Resources
Non-Functional	Resource Access	Functional
Non-Functional	Security	Functional
Non-Functional	Connectivity	Functional
Functional	Data	Functional
Functional	Compute Nodes	Functional

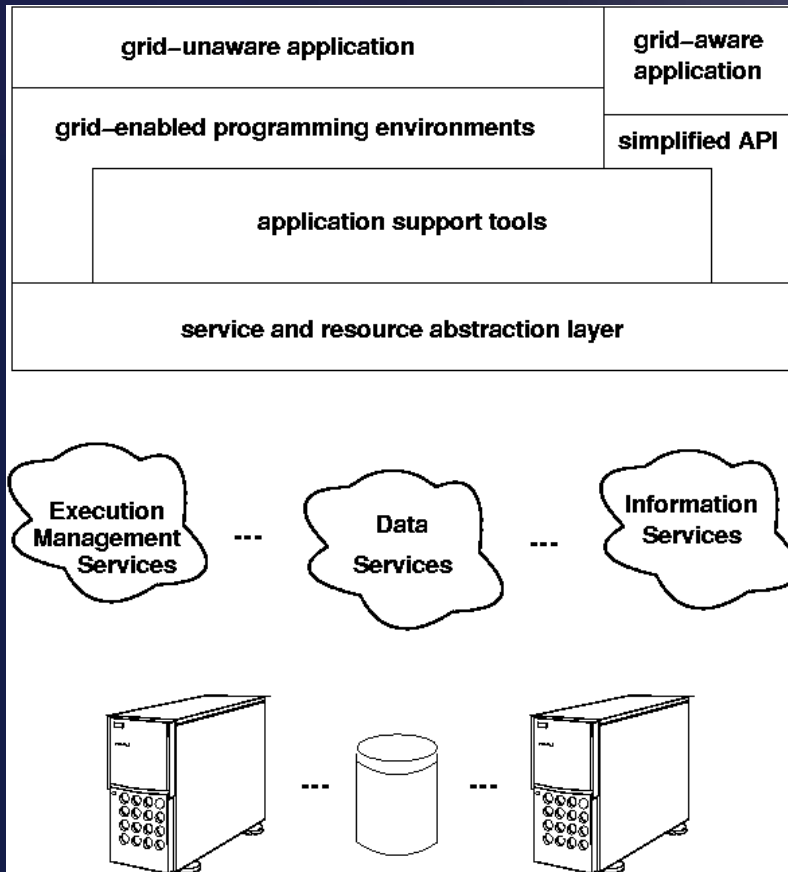
## Levels of Virtualization

Collective layer	Service APIs	Individual resources
Resource layer	Resource API (GRAM?)	resource/local scheduler
Connectivity layer	IP	Network links
Cluster OS	Management API	Compute nodes
JVM	Java Language	OS(?)
Virtual OS	System calls	OS
OS	System calls	Hardware

Each virtualization brings a trade-off between abstraction and control.



# Translating to API's



Application + runtime env.

Middleware

Resources

# Grid Application Runtime Stack

“just want to run fast”

“want to handle remote data/machines”

MPICH-G  
Workflow  
Satin/Ibis  
NetSolve  
...



SAGA

Added value for applications

Grid Application Toolkit (GAT)

## Your API depends on what you want to do

Legacy apps

Parallel apps

Grid-aware codes

Support tools

Services/resource management

Sand boxing (VM's?)

Grid-enabled environment

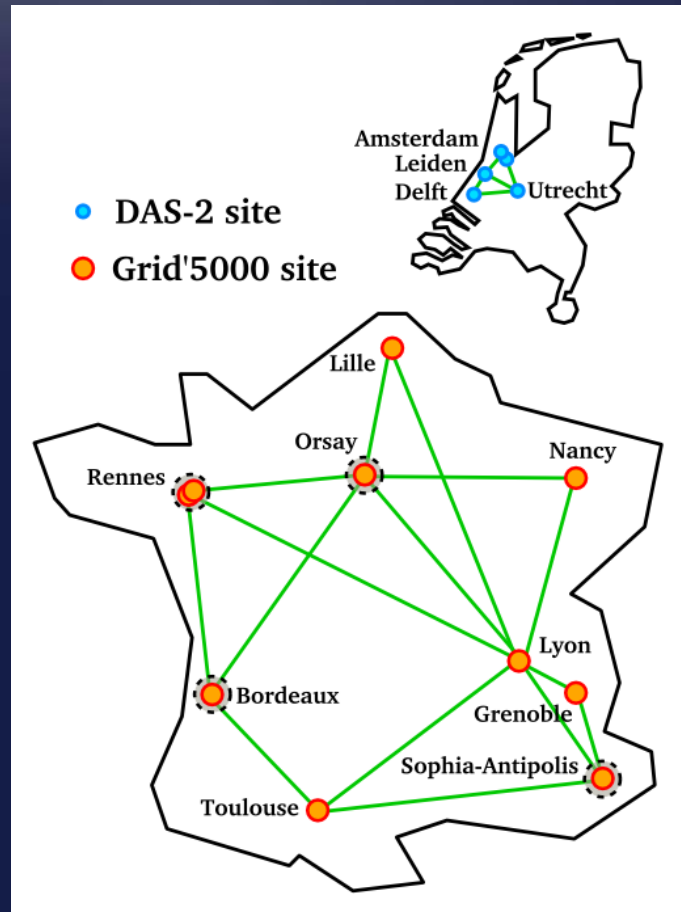
Simplified API (SAGA)

resource/service abstraction (GAT)


Service API's ("bells and WSDL's")

# A Case Study in Grid Programming

- Grids @ Work, Sophia-Antipolis, France, October 2005
- VU Amsterdam team participating in the N-Queens contest
- Aim: running on a 1000 distributed nodes



## The N-Queens Contest

- Challenge: solve the most board solutions within 1 hour
- Testbed:
  - Grid5000, DAS-2, some smaller clusters
  - Globus, NorduGrid, LCG, ???
  - 
  - In fact, there was not too much precise information available in advance...

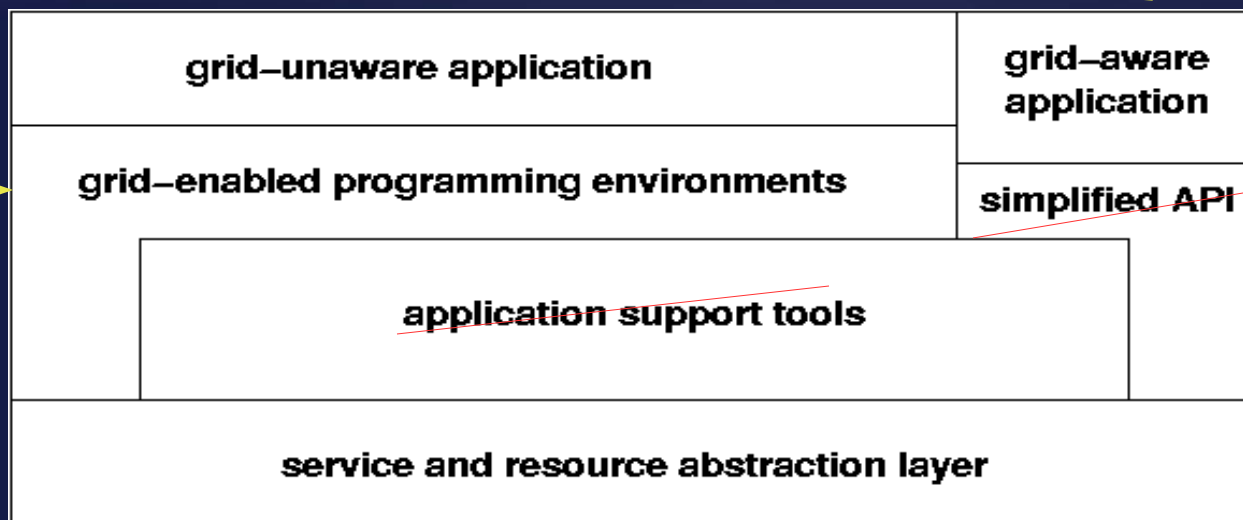
## Computing in an Unknown Grid?

- Heterogeneous machines (architectures, compilers, etc.)
  - Use Java: “write once, run anywhere”  
Use Ibis!
- Heterogeneous machines (fast / slow, small / big clusters)
  - Use automatic load balancing (divide-and-conquer)  
Use Satin!
- Heterogeneous middleware (job submission interfaces, etc.)
  - Use the Grid Application Toolkit (GAT)!

# Assembling the Pieces

N-Queens

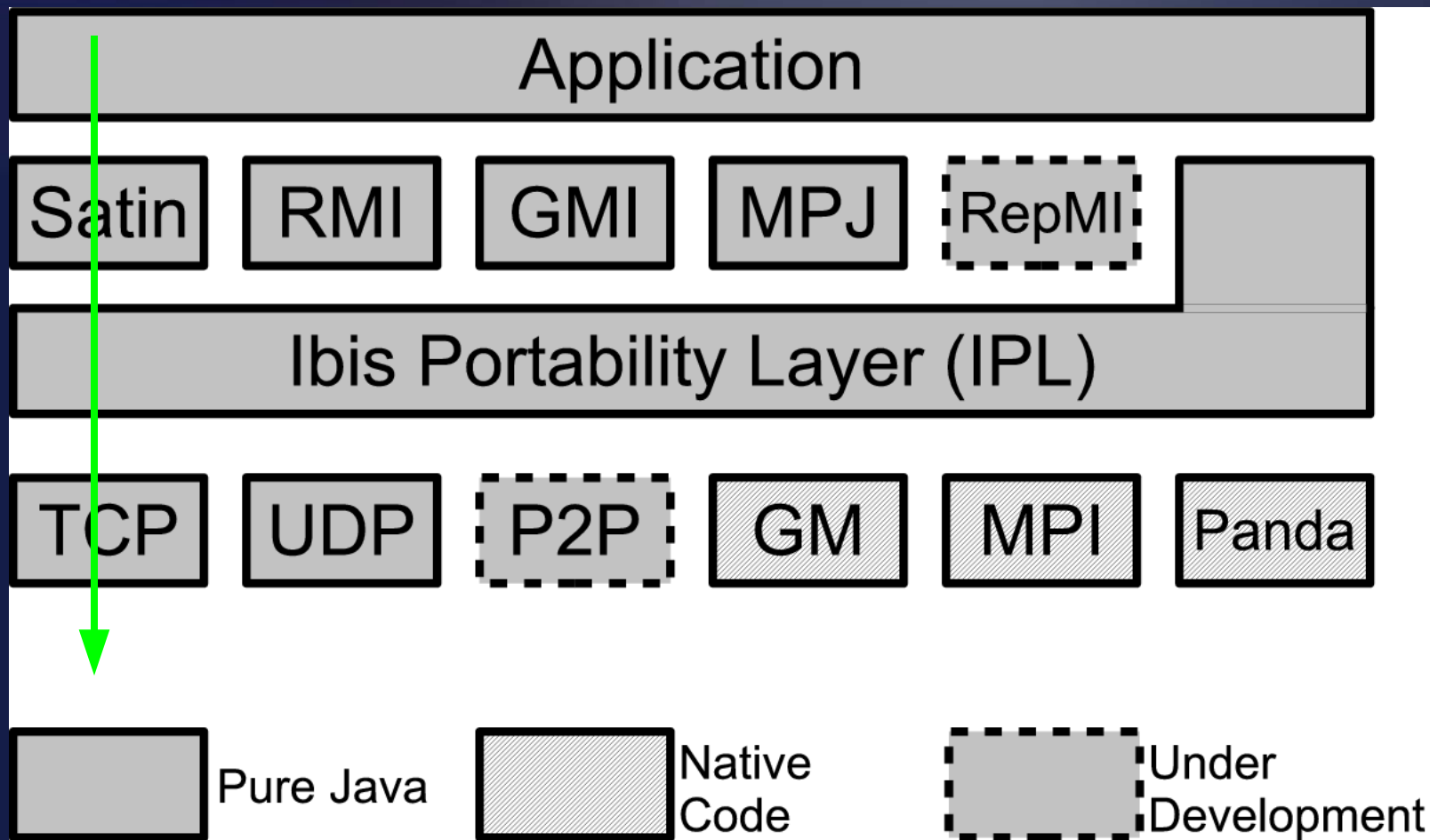
Deployment application



Satin/Ibis

Java GAT on top of ProActive and ssh

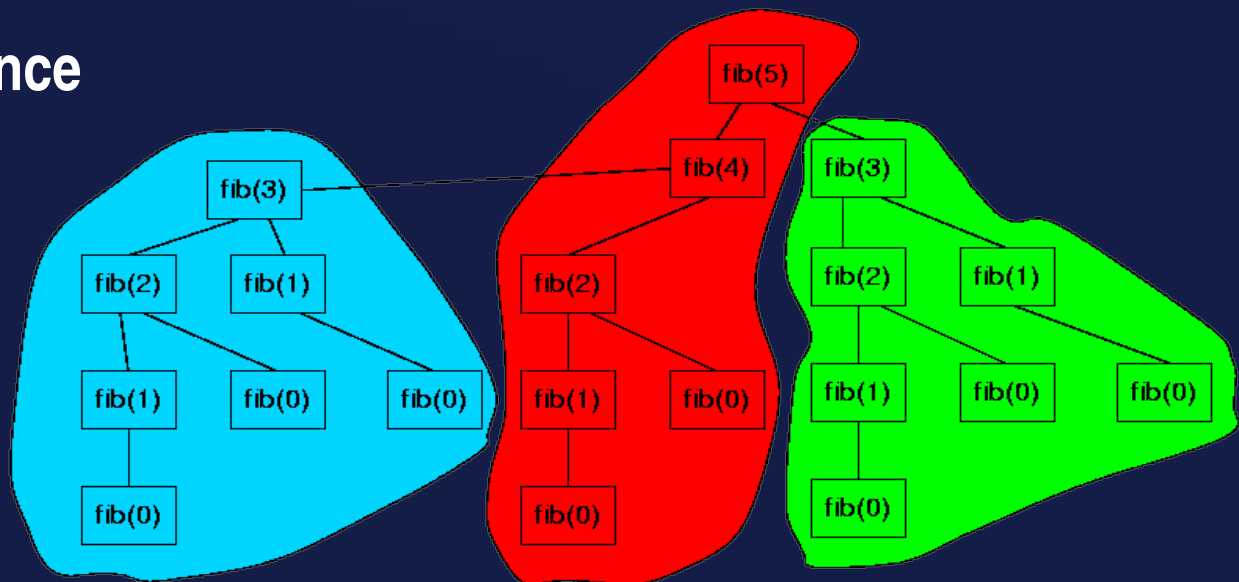
# The Ibis Grid Programming System





## Satin: Divide-and-conquer

- Effective paradigm for Grid applications (hierarchical)
- Satin: Grid-aware load balancing (work stealing)
- Also support for
  - Fault tolerance
  - Malleability
  - Migration

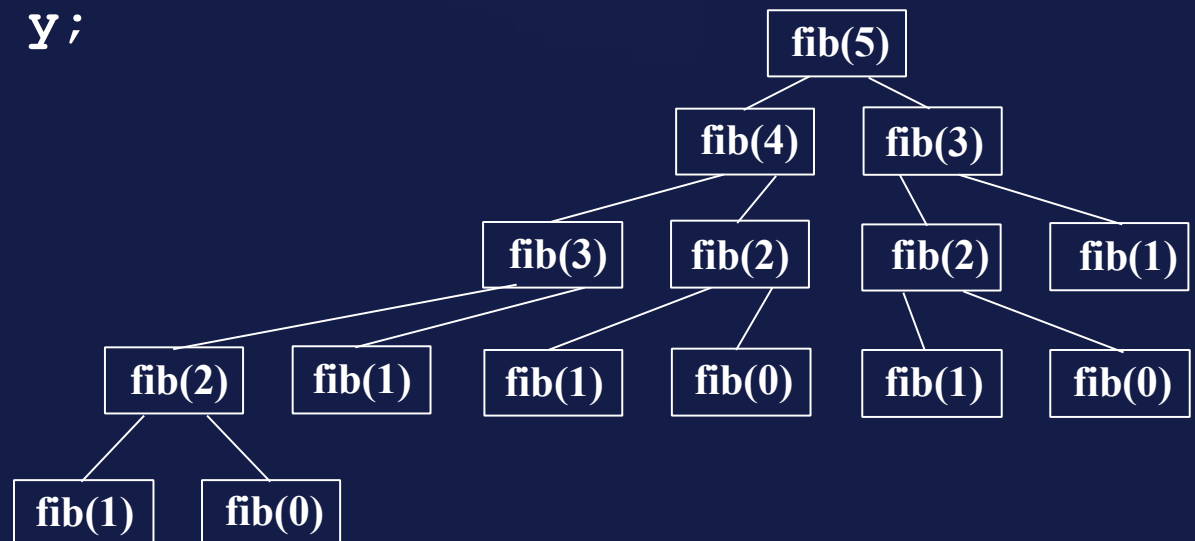


# Satin Example: Fibonacci

```

class Fib {
  int fib (int n) {
    if (n < 2) return n;
    int x = fib(n-1);
    int y = fib(n-2);
    return x + y;
  }
}
    
```

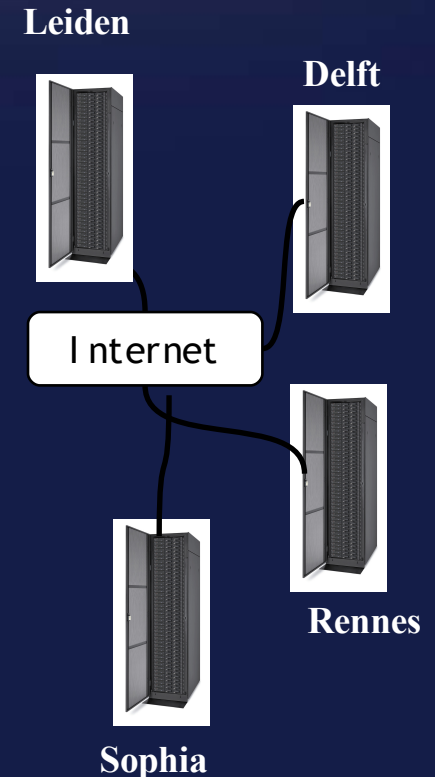
**Single-threaded  
Java**



# Satin Example: Fibonacci

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

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



(use byte code rewriting to generate parallel code)

# Satin: Fault-Tolerance, Malleability, Migration

- Referential transparency (jobs can be recomputed)
- Approach: fix the job tree, once
  - either fault is detected
  - or node wants to leave (gracefully)
- Adding nodes: let them start stealing work
- Migration: adding and removing nodes simultaneously

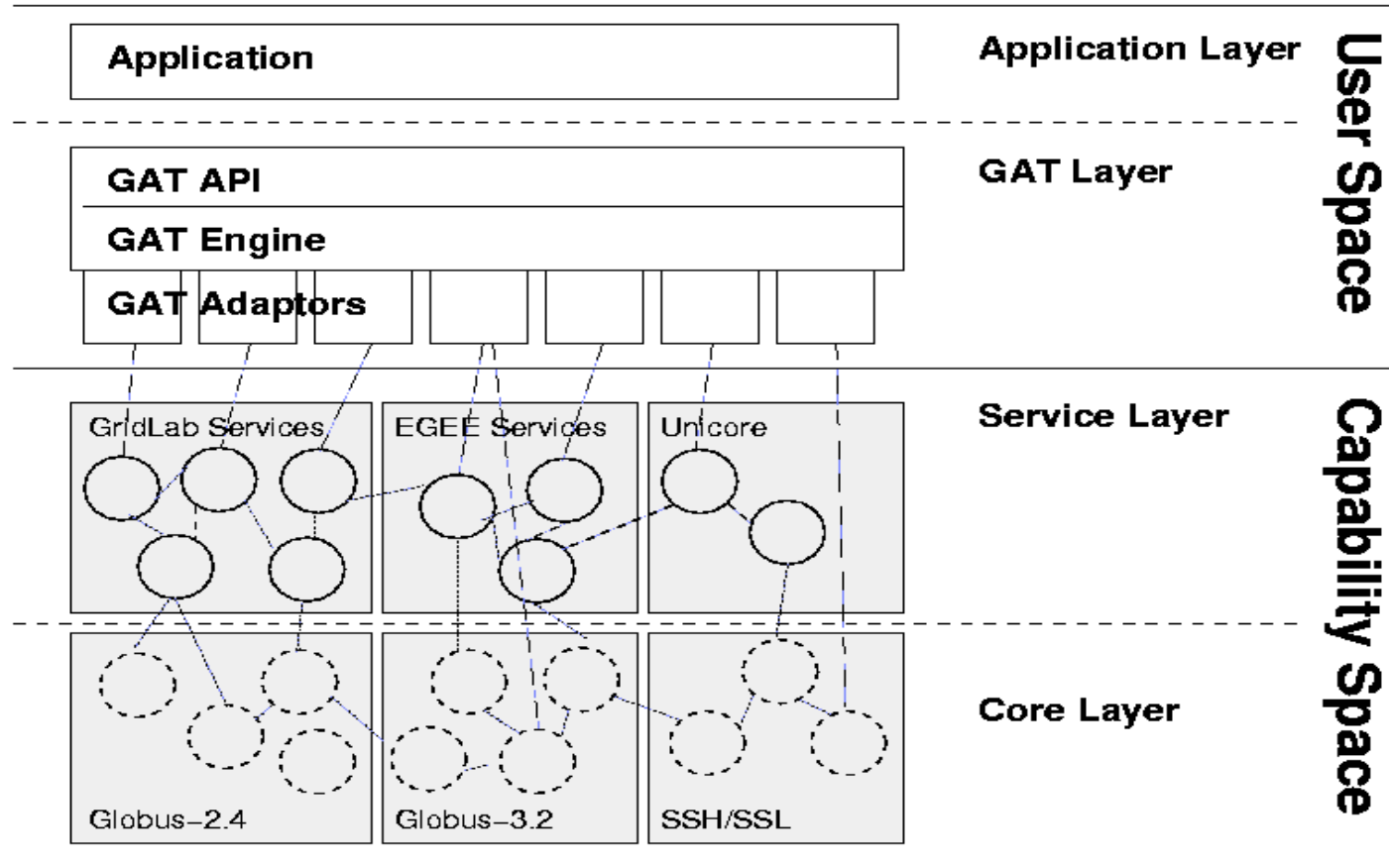
## Summary: Ibis

- **Java: “write once, run anywhere”**
  - machine virtualization
- **Ibis: efficient communication**
  - network virtualization
- **Satin: load balancing, fault-tolerance, migration**
  - resource virtualization

**But how do we deploy our Ibis / Satin application?**

**A (non-) functional problem to be solved**

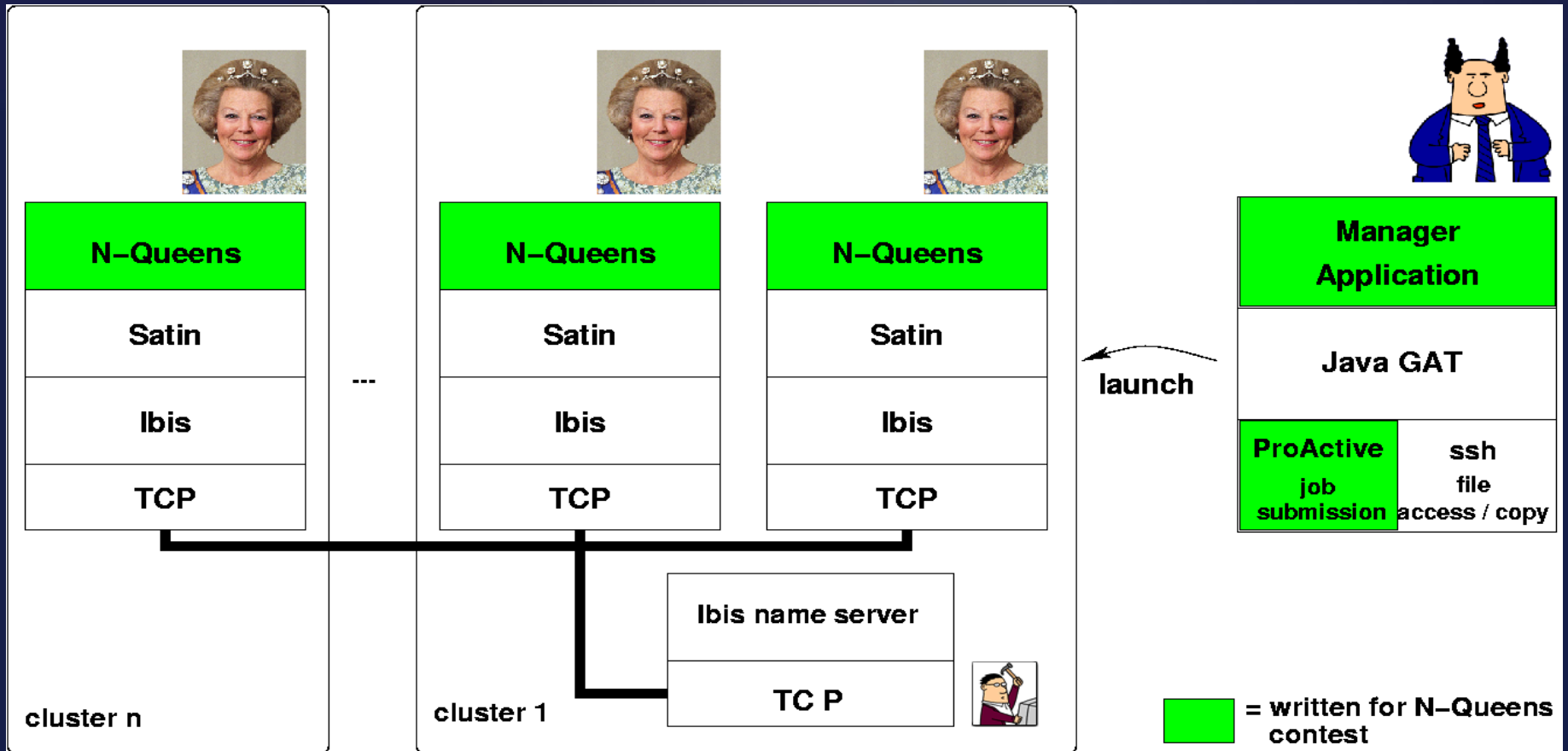
# The Grid Application Toolkit (GAT)



## The Grid Application Toolkit (GAT)

- Simple and uniform API to various Grid middleware:
  - Globus 2,3,4, ssh, Unicore, ...
- Job submission, remote file access, job monitoring and steering
- Implementations:
  - C, with wrappers for C++ and Python
  - Java

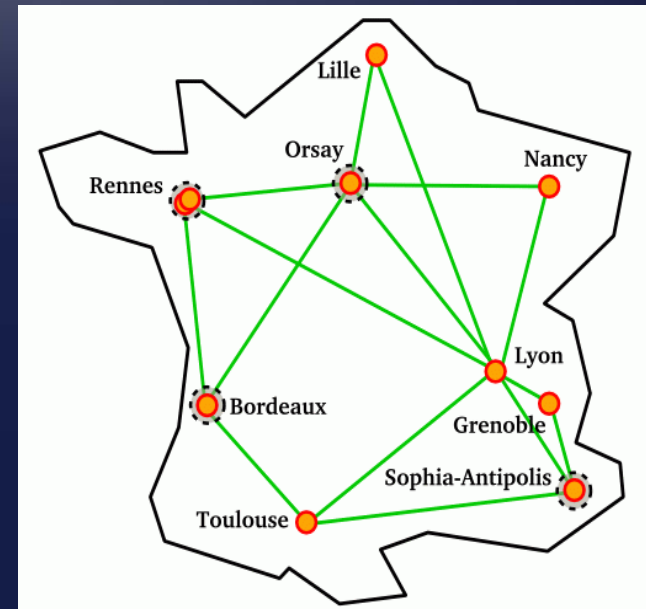
# The Beatrix N-Queens Architecture





## Results achieved on the Grid5000 Testbed

Site	CPUs
Orsay	426
Bordeaux	92
Rennes, Opteron cluster	120
Rennes, Xeon cluster	128
Sophia Antipolis	196
<b>Total:</b>	<b>960</b>



- Solved  $n=22$  in 25 minutes
- 4.7 million jobs, 800,000 load balancing messages

## Summary / Thoughts about Grid API's

- Grid applications have many problems to address
- Different problems require different API's
- It's all about virtualization (on all levels)
- Can we find the “MPI equivalent” for the grid? Should we?
- Grids are considered successful as soon as they become invisible/ubiquitous.
- Are we done once everything is nicely “virtualized away”?
- Should everything just be a Web service? (maybe not)

## Acknowledgements

- **Ibis:**  
Jason Maassen, Rob van Nieuwpoort, Cerial Jacobs, Rutger Hofman, Gosia Wrzesinska, Niels Drost, Olivier Aumage, Alexandre Denis, Fabrice Huet, Henri Bal, the Dutch VL-e project
- **GAT:**  
Andre Merzky, Rob van Nieuwpoort, the EU GridLab project
- **N-Queens:**  
Ana-Maria Oprescu, Andrei Agapi, the EU CoreGRID NoE
- **SAGA:**  
Andre Merzky, Shantenu Jha, Pascal Kleijer, Hartmut Kaiser, Stephan Hirmer, the OGF SAGA-RG, the EU XtremOS project