JavaGAT and Ibis Demo

SP 3.1

high-performance distributed computing

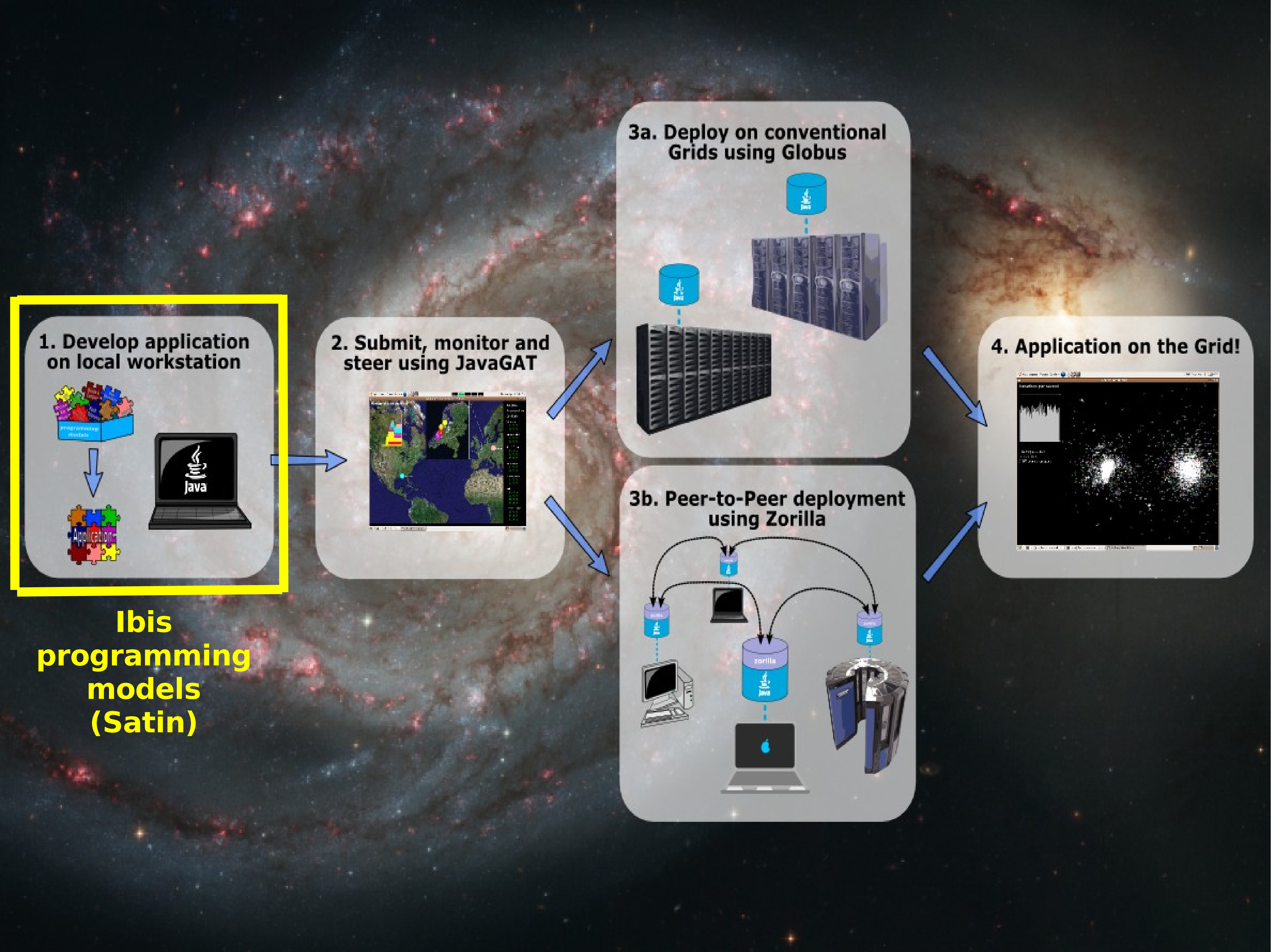
Mid-term progress report page 24

Henri Bal
Niels Drost
Ceriel Jacobs
Jason Maassen
Rob van Nieuwpoort

vrije Universiteit

www.cs.vu.nl/ibis


1. Develop application on local workstation
2. Submit, monitor and steer using JavaGAT
3a. Deploy on conventional Grids using Globus
3b. Peer-to-Peer deployment using Zorilla
4. Application on the Grid!
Java
Application
zorilla


1. Develop application on local workstation
Java
2. Submit, monitor and steer using JavaGAT
3a. Deploy on conventional Grids using Globus
3b. Peer-to-Peer deployment using Zorilla
zorilla
4. Application on the Grid!
Ibis programming models (Satin)

Satin

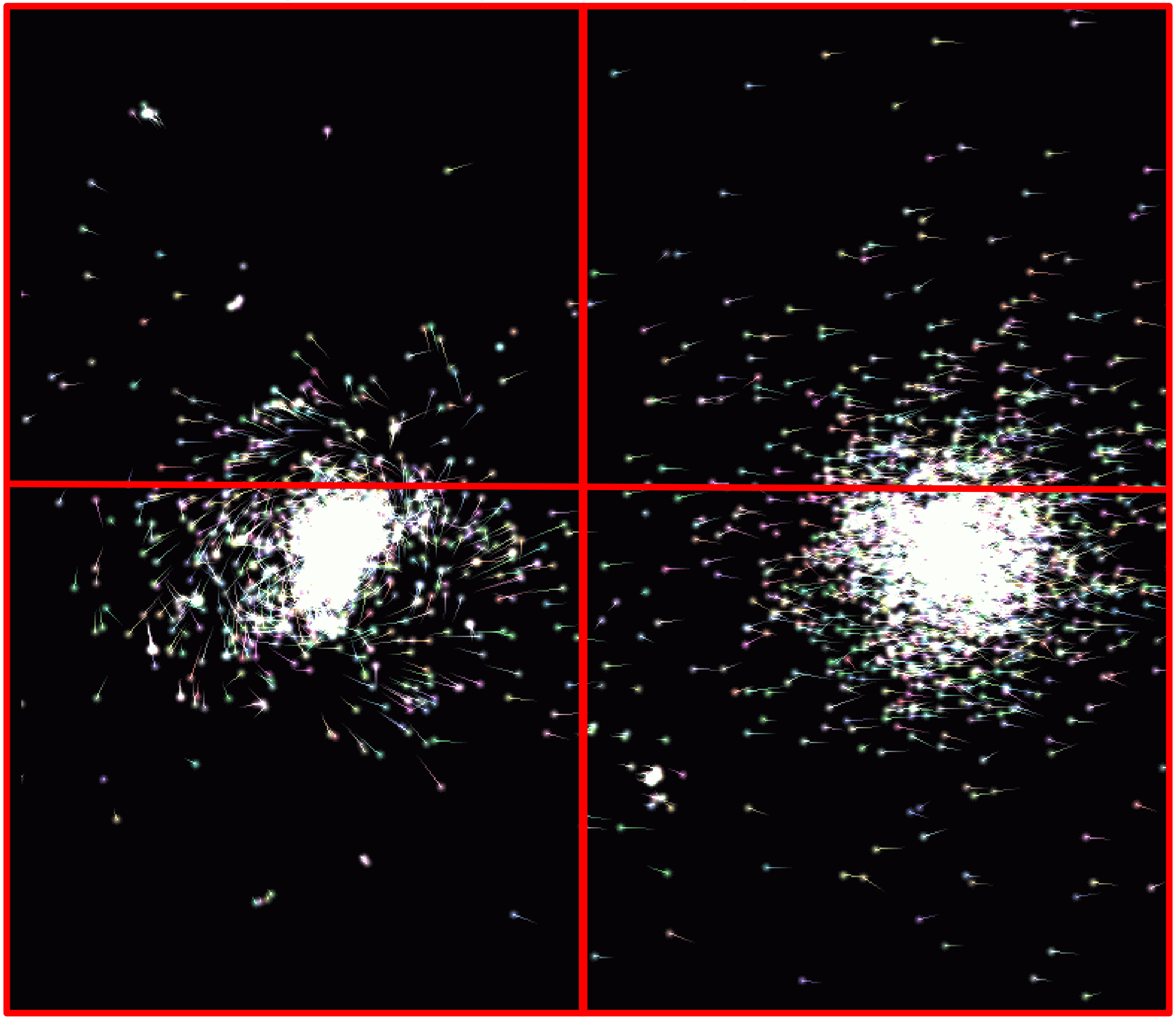
- One of the high-level programming models for the Ibis communication library
- Provide a powerful programming model
  - master/worker, divide-and-conquer, shared objects
- Extremely easy to use
- Satin allows applications to transparently deal with grid issues
  - load balancing, malleability, migration, fault-tolerance, adaptivity, firewalls, heterogeneity


1. Develop application on local workstation
Application
2. Submit, monitor and steer using JavaGAT
3a. Deploy on conventional Grids using Globus
3b. Peer-to-Peer deployment using Zorilla
zorilla
4. Application on the Grid!
Barnes-Hut Nbody simulation

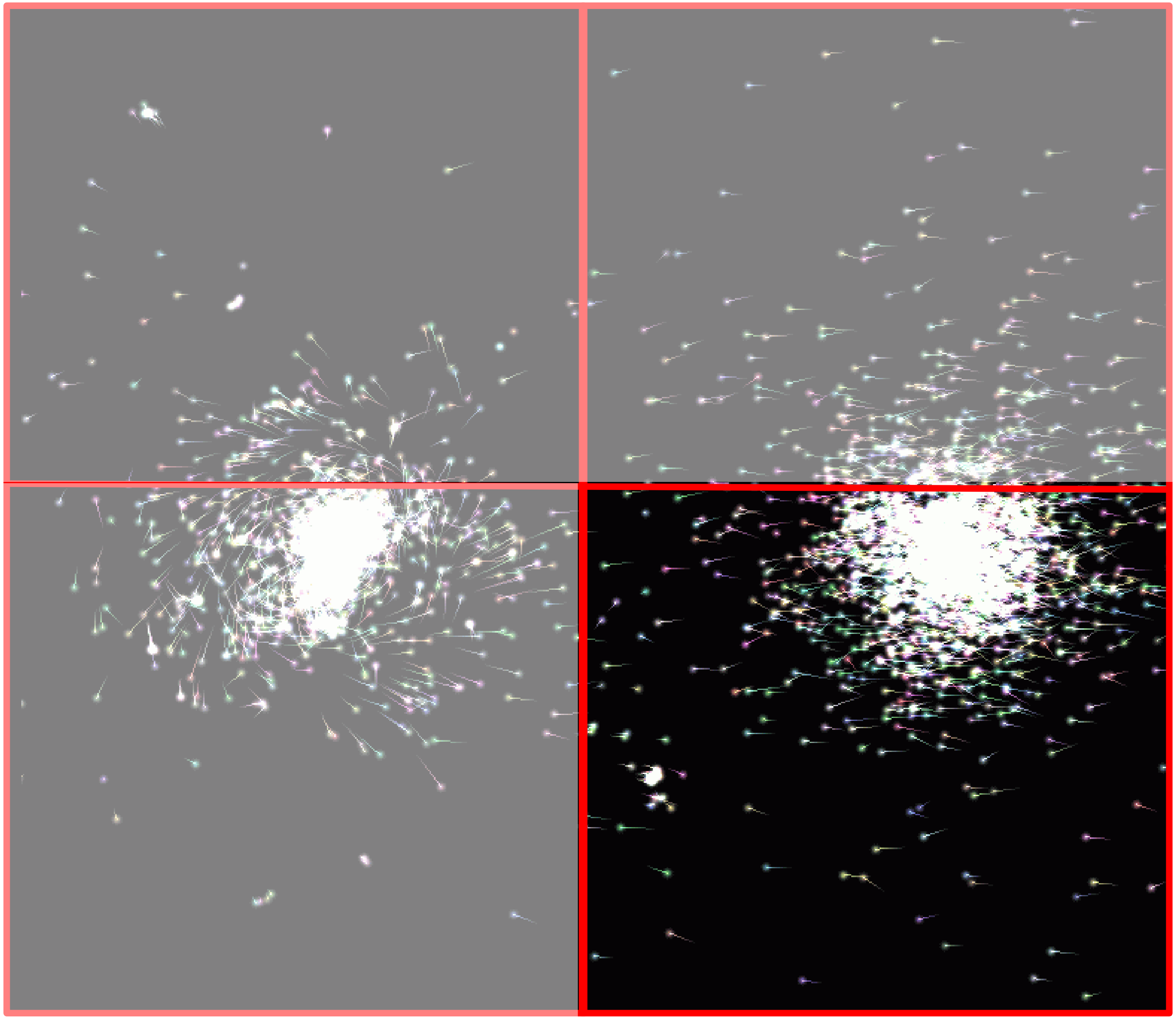
Barnes-Hut N-body simulation

Bodies (stars) are organized in a tree
complexity is not N^2 but $N \log N$

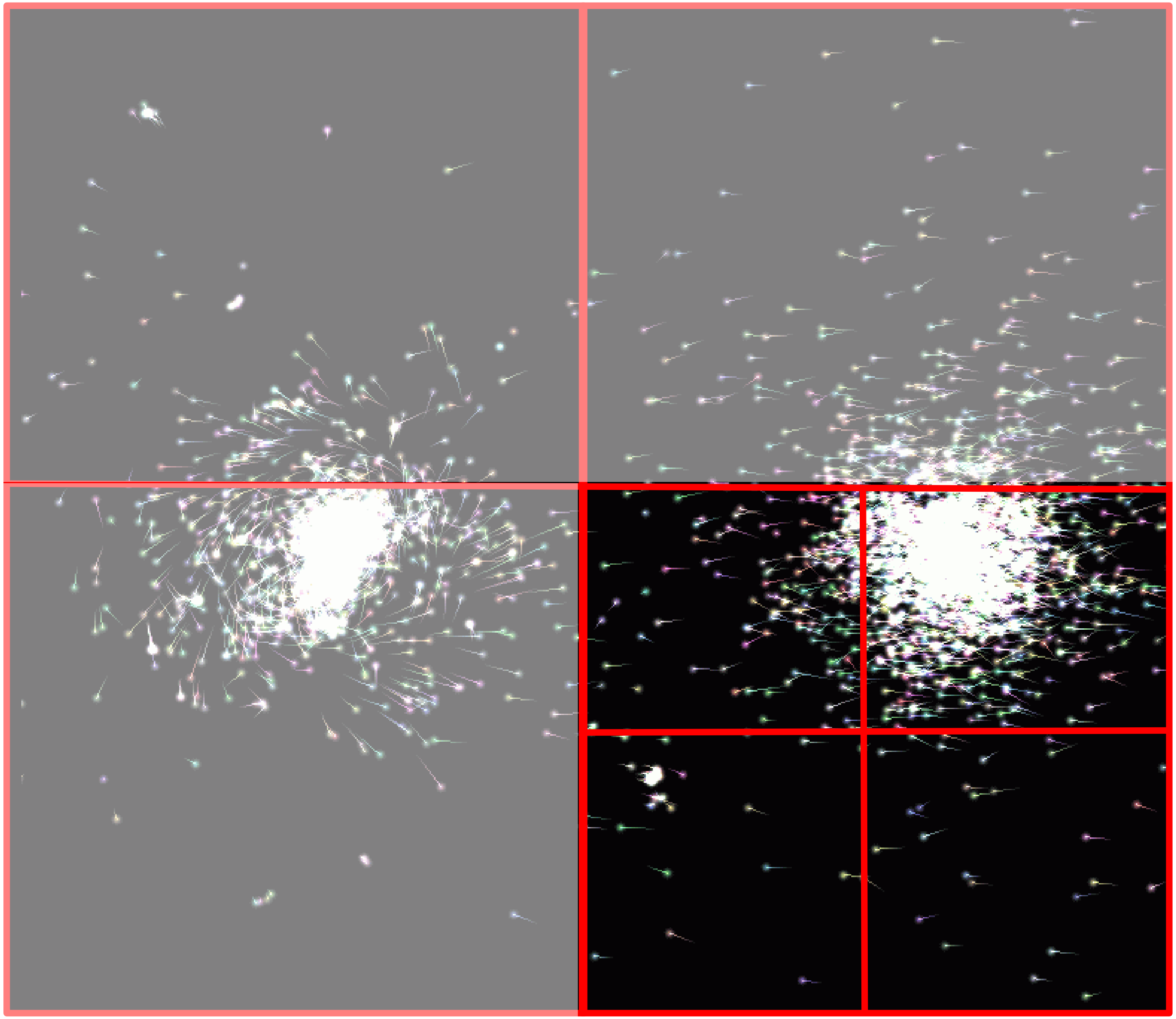
Barnes-Hut N-body simulation



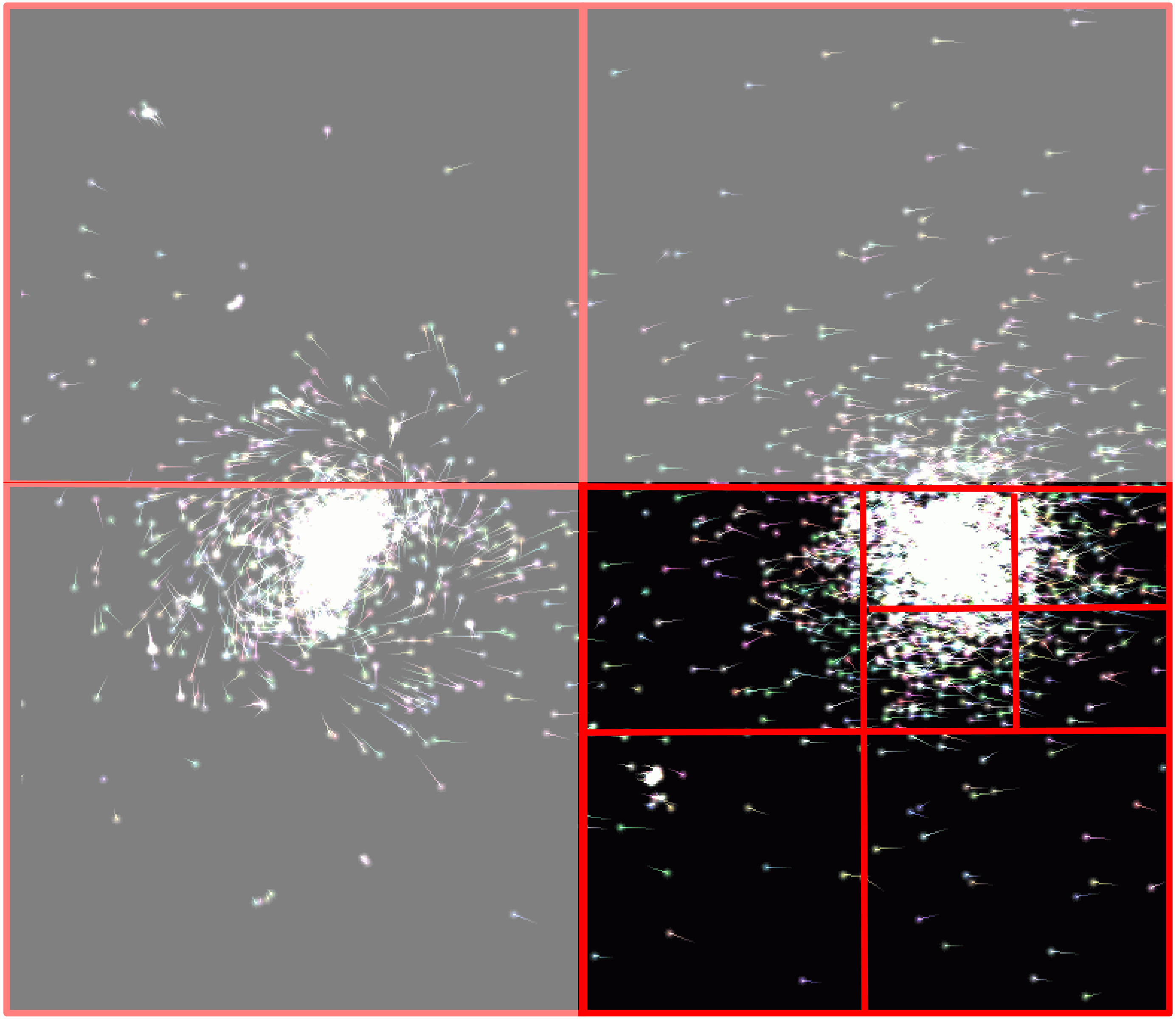
Barnes-Hut N-body simulation



Barnes-Hut N-body simulation



Barnes-Hut N-body simulation



Satin Applications

- Vl-e
  - Grammar induction, SP 2.2
    - Pieter Adriaans, Ceriel Jacobs
  - N-body simulations (DEMO)
- Grammar-based text analysis
- VLSI routing
- Satisfiability solver
- Gene sequencing
- Game-tree search
- Raytracing
- Numerical functions
- ...


1. Develop application on local workstation
Java
2. Submit, monitor and steer using JavaGAT
Java Grid Application Toolkit (JavaGAT)
3a. Deploy on conventional Grids using Globus
3b. Peer-to-Peer deployment using Zorilla
zorilla
4. Application on the Grid!

The Java Grid Application Toolkit

- API for developing and running portable grid applications independently of the underlying grid infrastructure and available services
- Simple API
- GAT Adaptors (“plugins”) connect GAT to grid services (Globus, Unicore, SSH, ProActive, ...)

Java GAT users

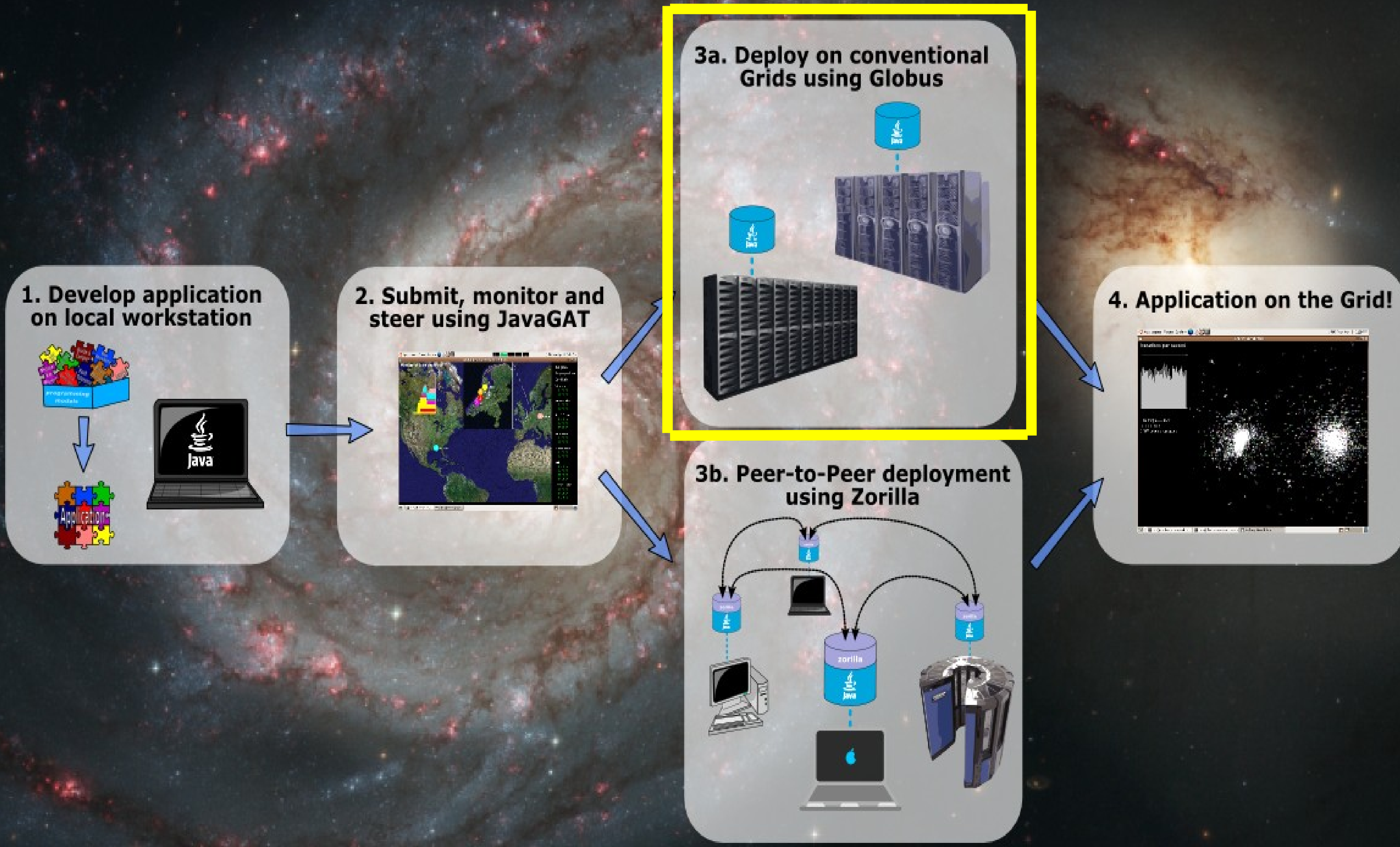
- The Virtual Labs for E-science project (Vl-e)
  - AMOLF, Institute for Atomic and Molecular Physics, SP 1.6
  - Vrije Universiteit Amsterdam, SP 3.1
  - VU Medical Center Amsterdam, SP 1.3
  - Vbrowser, SP 2.5 / SP 1.3
- The Multimedian project
- Max Planck Institute for Astrophysics in Garching
- D-Grid, Astrogrid
- Louisiane State University
- University of Texas
- The workflow system Triana (University of Cardiff)
- Georgia State University
- Zuse Institute Berlin, Germany
- Download is anonymous, so we don't know

vl·e

ibis

vrije Universiteit

rapid prototyping environment: DAS-3
(also works on proof of concept environment)



Demo …

Distributed supercomputing

- Parallel processing on geographically distributed computing systems (grids)
- Our goals:
  - Don’t use individual supercomputers / clusters, but combine multiple systems
  - Provide high-level programming support

Optimizing for the grid

- Grids usually are hierarchical
  - Collections of clusters, supercomputers
  - Fast local links, slow wide-area links
- Optimize algorithms to exploit hierarchy
  - Message combining + latency hiding on wide-area links
  - Collective operations for wide-area systems
  - Successful for many applications

Satin master-worker

- Master-worker parallelism
  - Divide work into parts
  - Spawn job for each part
  - Solve parts in parallel
  - Combine results

Satin divide-and-conquer

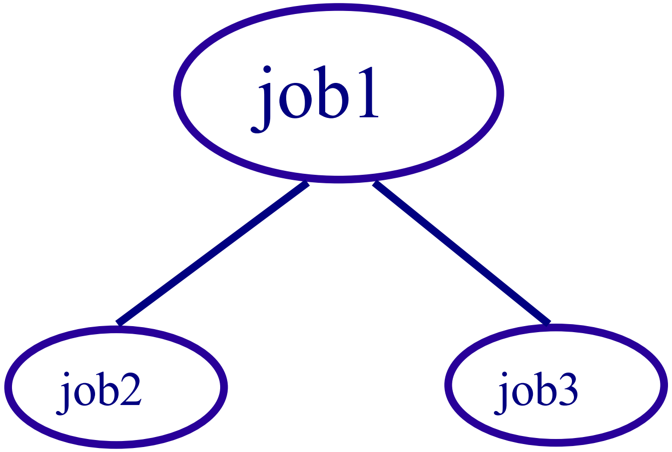
- Parallel Divide-and-conquer
  - Divide work into parts
  - Spawn job for each part
  - Solve parts in parallel
  - Combine results
  - But now recursively!
  - sub-problems split the work up further and spawn their own sub-jobs

Satin

- Divide-and-conquer

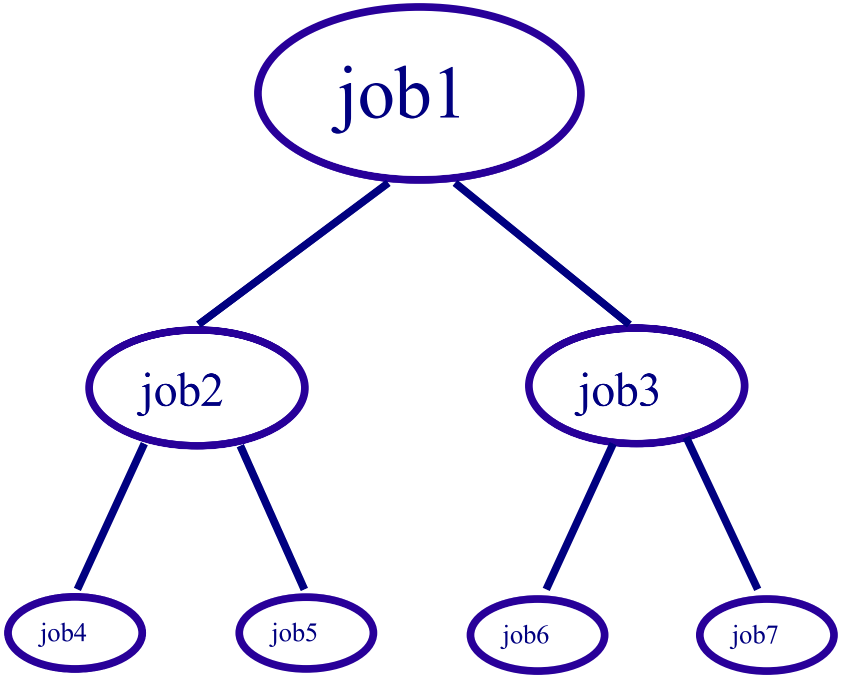
Satin

- Divide-and-conquer



Satin

- Divide-and-conquer



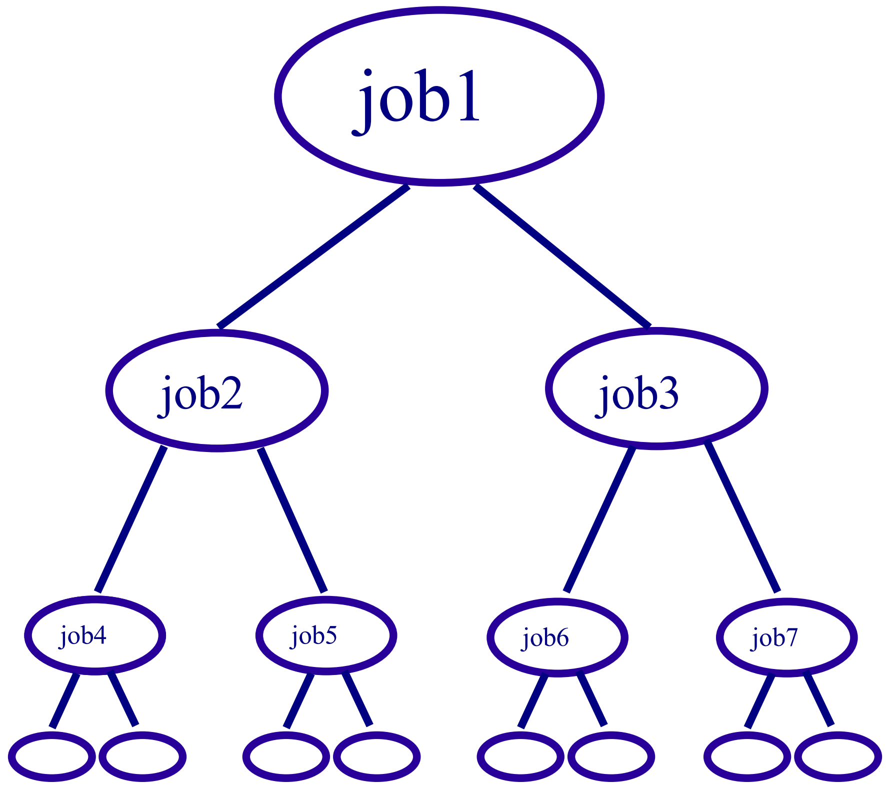
vl·e

ibis

vrije Universiteit

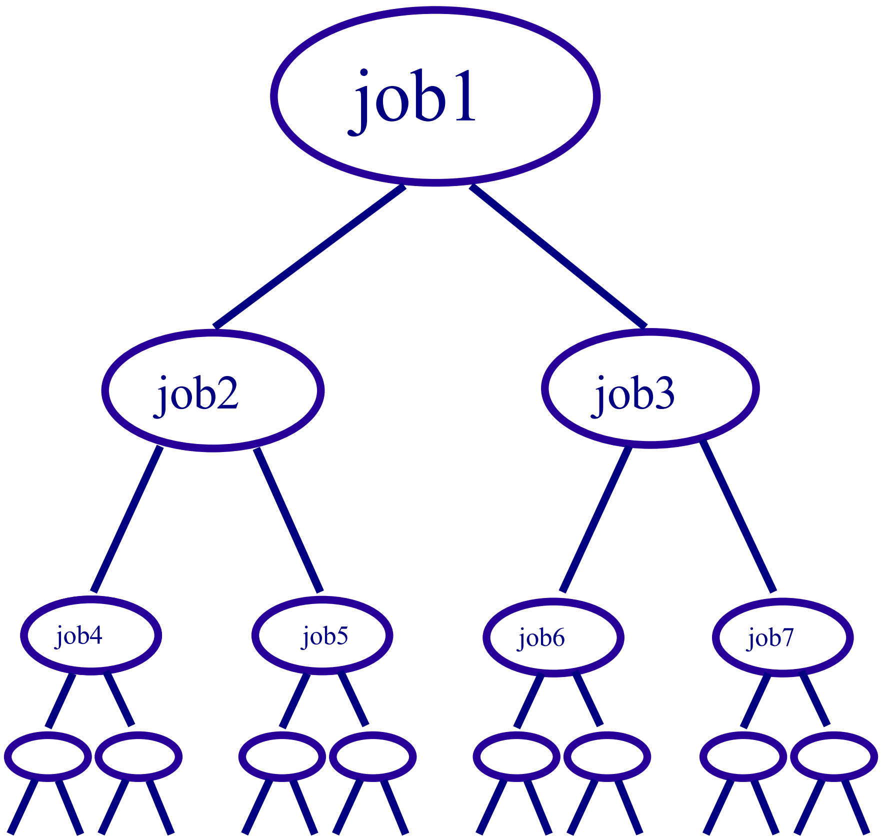
Satin

- Divide-and-conquer



Satin

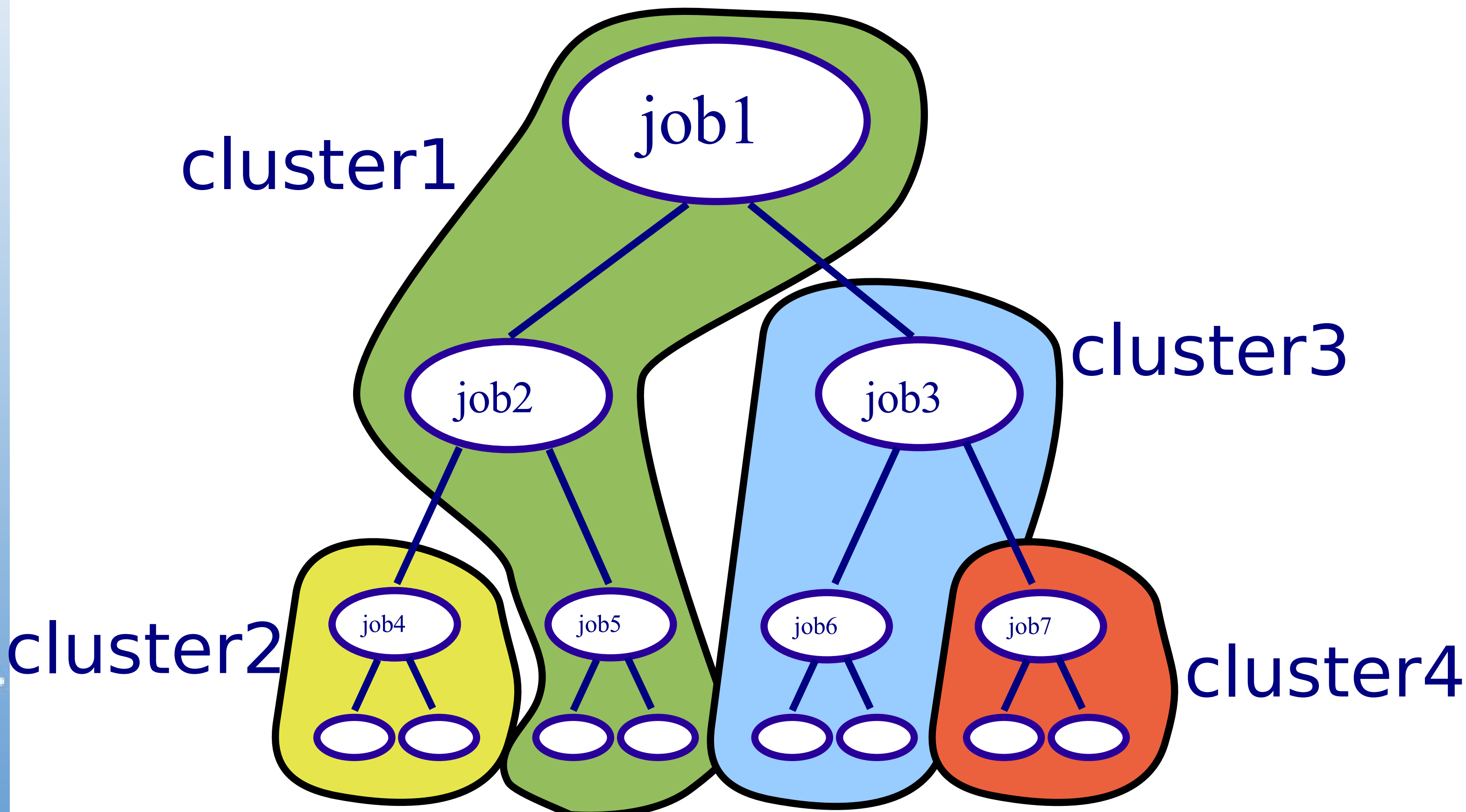
- Divide-and-conquer



... millions of jobs ...

Satin

- Fits hierarchical structure of Grids



The Grid

- Satin explicitely targets the grid
- Different architectures
- Firewalls
- Slow networks (latency)
- Distributed memory
- Machines come and go
- Machines crash
- Machines have different speeds

Barnes-Hut Nbody code

```
// Marker interface that defines updateBodies as a global method.
interface BodiesInterface extends satin.GlobalMethods {
  void updateBodies(BodyUpdates b, int iter);
}

// A shared object containing the tree of bodies.
class Bodies extends satin.SharedObject implements BodiesInterface {
  BodyTreeNode root;

  void updateBodies(BodyUpdates b, int iter) { // Global method.
    root.applyUpdates(b, iter); // Update bodies in our tree.
  }

  BodyTreeNode getRoot() { // Local method.
    return root;
  }
}

// Mark the computeForces method as a spawn operation.
interface BHSpawns extends satin.Spawnable {
  BodyUpdates computeForces(Subtree s, int iter, Bodies bodies);
}

class BarnesHut extends satin.SatinObject implements BHSpawns {
  boolean guard_computeForces(Subtree s, int iter, Bodies bodies) {
    return bodies.iter + 1 == iter;
  }

  // Spawnable method. The "bodies" parameter is a shared object.
  BodyUpdates computeForces(Subtree s, int iter, Bodies bodies) {
    if(s.hasNoChildren) {
      computeSequentially(s, iter, bodies.getRoot());
    } else { // Divide the work and spawn tasks (recursion step).
      for(int i=0; i<s.nrChildren; i++) {
        res[i] = computeForces(s.child[i], iter, bodies); // Spawn.
      }
      sync(); // Wait for the spawn operation to finish.

      return mergeSubresults(res); // Merge results and return.
    }
  }

  public static void main(String[] args) {
    BarnesHut bh = new BarnesHut();
    Bodies bodies = new Bodies(); // Create shared object.
    for (int iter = 0; iter < N; iter++) {
      results = bh.computeForces(root, iter, bodies); // Spawn.
      sync(); // Wait for the spawn operation to finish.
      bodies.update(results, iter); // Shared method invocation.
    }
  }
}
```

Barnes-Hut Nbody code

```
// Marker interface that defines updateBodies as a global method.
interface BodiesInterface extends satin.GlobalMethods {
  void updateBodies(BodyUpdates b, int iter);
}

// A shared object containing the tree of bodies.
class Bodies extends satin.SharedObject implements BodiesInterface {
  BodyTreeNode root;

  void updateBodies(BodyUpdates b, int iter) { // Global method.
    root.applyUpdates(b, iter); // Update bodies in our tree.
  }

  BodyTreeNode getRoot() { // Local method.
```

```
// Marker interface that defines updateBodies as a global method.
interface BodiesInterface extends satin.GlobalMethods {
    void updateBodies(BodyUpdates b, int iter);
}

// A shared object containing the tree of bodies.
class Bodies extends satin.SharedObject implements BodiesInterface {
    BodyTreeNode root;

    void updateBodies(BodyUpdates b, int iter) {
      root.applyUpdates(b, iter);
    }

    BodyTreeNode getRoot() {
        return root;
    }
}
```

Barnes-Hut Nbody code

```
// Marker interface that defines updateBodies as a global method.
interface BodiesInterface extends satin.GlobalMethods {
  void updateBodies(BodyUpdates b, int iter);
}

// A shared object containing the tree of bodies.
class Bodies extends satin.SharedObject implements BodiesInterface {
  BodyTreeNode root;

  void updateBodies(BodyUpdates b, int iter) { // Global method.
    root.applyUpdates(b, iter); // Update bodies in our tree.
  }

  BodyTreeNode getRoot() { // Local method.
    return root;
  }
}

// Mark the computeForces method as a spawn operation.
interface BHSpawns extends satin.Spawnable {
  BodyUpdates computeForces(Subtree s, int iter, Bodies bodies);
}

class BarnesHut extends satin.SatinObject implements BHSpawns {
  boolean guard_computeForces(Subtree s, int iter, Bodies bodies) {
```

```
// Mark the computeForces method as a spawn operation.
interface BHSpawns extends satin.Spawnable {
  BodyUpdates computeForces(Subtree s, int iter, Bodies bodies);
}
```

```
      return mergeSubresults(res); // Merge results and return.
    }
  }

  public static void main(String[] args) {
    BarnesHut bh = new BarnesHut();
    Bodies bodies = new Bodies(); // Create shared object.
    for (int iter = 0; iter < N; iter++) {
      results = bh.computeForces(root, iter, bodies); // Spawn.
      sync(); // Wait for the spawn operation to finish.
      bodies.update(results, iter); // Shared method invocation.
    }
  }
}
```

Barnes-Hut Nbody code

```
// Marker interface that defines updateBodies as a global method.
interface BodiesInterface extends satin.GlobalMethods {
  void updateBodies(BodyUpdates b, int iter);
}

// A shared object containing the tree of bodies.
class Bodies extends satin.SharedObject implements BodiesInterface {
  BodyTreeNode root;
```

```
public static void main(String[] args) {
    BarnesHut bh = new BarnesHut();
    Bodies bodies = new Bodies();
    for (int iter = 0; iter < N; iter++) {
        results = bh.computeForces(root, iter, bodies);
        sync();
        bodies.updateBodies(results, iter);
    }
}
```

```
      }
      sync(); // Wait for the spawn operation to finish.

      return mergeSubresults(res); // Merge results and return.
    }
  }
```

```
  public static void main(String[] args) {
    BarnesHut bh = new BarnesHut();
    Bodies bodies = new Bodies(); // Create shared object.
    for (int iter = 0; iter < N; iter++) {
      results = bh.computeForces(root, iter, bodies); // Spawn.
      sync(); // Wait for the spawn operation to finish.
      bodies.update(results, iter); // Shared method invocation.
    }
  }
}
```

Barnes-Hut Nbody code

```
// Marker interface that defines updateBodies as a global method.
interface BodiesInterface extends satin.GlobalMethods {
  void updateBodies(BodyUpdates b, int iter);
}

// A shared object containing the tree of bodies.
class Bodies extends satin.SharedObject implements BodiesInterface {
  BodyTreeNode root;
```

```
BodyUpdates computeForces(Subtree s, int iter, Bodies bodies) {
    if(s.hasNoChildren) {
        computeSequentially(s, iter, bodies.getRoot());
    } else {
        for(int i=0; i<s.nrChildren; i++) {
            res[i] = computeForces(s.child[i], iter, bodies);
        }
        sync();

        return mergeSubresults(res);
    }
}
```

```
  }

  public static void main(String[] args) {
    BarnesHut bh = new BarnesHut();
    Bodies bodies = new Bodies(); // Create shared object.
    for (int iter = 0; iter < N; iter++) {
      results = bh.computeForces(root, iter, bodies); // Spawn.
      sync(); // Wait for the spawn operation to finish.
      bodies.update(results, iter); // Shared method invocation.
    }
  }
}
```

Barnes-Hut Nbody code

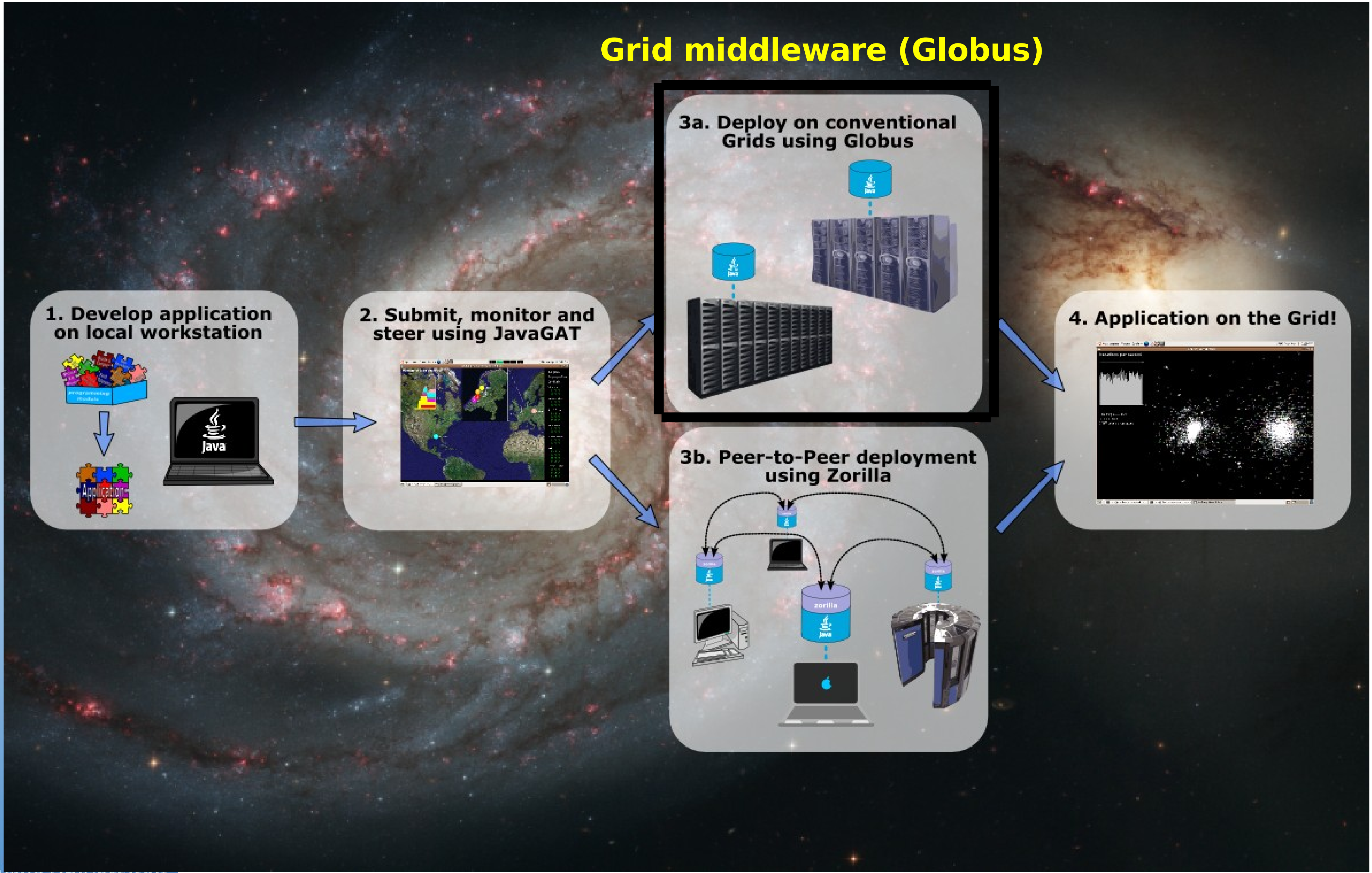
```
// Marker interface that defines updateBodies as a global method.
interface BodiesInterface extends satin.GlobalMethods {
  void updateBodies(BodyUpdates b, int iter);
}

// A shared object containing the tree of bodies.
class Bodies extends satin.SharedObject implements BodiesInterface {
  BodyTreeNode root;

  void updateBodies(BodyUpdates b, int iter) { // Global method.
    root.applyUpdates(b, iter); // Update bodies in our tree
```

```
boolean guard_computeForces(Subtree s, int iter, Bodies bodies) {
  return iter == bodies.iter + 1;
}
```

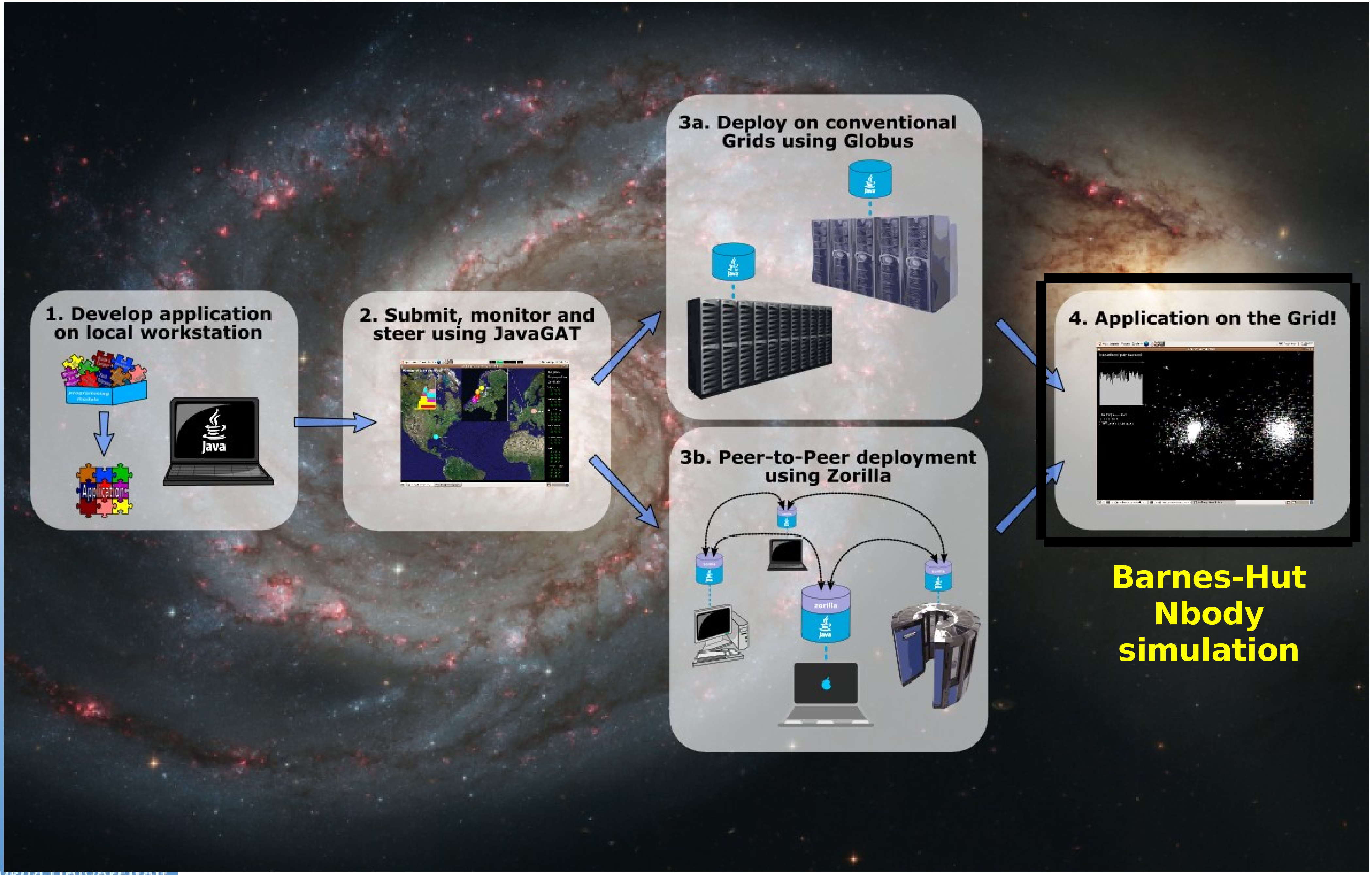
```
}

class BarnesHut extends satin.SatinObject implements BHSpawns {
  boolean guard_computeForces(Subtree s, int iter, Bodies bodies) {
    return bodies.iter + 1 == iter;
  }

  // Spawnable method. The "bodies" parameter is a shared object.
  BodyUpdates computeForces(Subtree s, int iter, Bodies bodies) {
    if(s.hasNoChildren) {
      computeSequentially(s, iter, bodies.getRoot());
    } else { // Divide the work and spawn tasks (recursion step).
      for(int i=0; i<s.nrChildren; i++) {
        res[i] = computeForces(s.child[i], iter, bodies); // Spawn.
      }
      sync(); // Wait for the spawn operation to finish.

      return mergeSubresults(res); // Merge results and return.
    }
  }

  public static void main(String[] args) {
    BarnesHut bh = new BarnesHut();
    Bodies bodies = new Bodies(); // Create shared object.
    for (int iter = 0; iter < N; iter++) {
      results = bh.computeForces(root, iter, bodies); // Spawn.
      sync(); // Wait for the spawn operation to finish.
      bodies.update(results, iter); // Shared method invocation.
    }
  }
}
```

vrije Universiteit

Application Example



Application Example



Barnes-Hut Nbody simulation

Sequential Fibonacci

```
public long fib(int n) {
        if (n < 2) return n;

        long x = fib(n - 1);
        long y = fib(n – 2);

        return x + y;
}
```

Parallel Fibonacci

```
interface FibInterface extends ibis.satin.Spawnable {
    public long fib(int n);
}
public long fib(int n) {
        if (n < 2) return n;

        long x = fib(n - 1);
        long y = fib(n – 2);
        sync();
        return x + y;
}
```

Parallel Fibonacci

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

public long fib(int n) {
        if (n < 2) return n;

        long x = fib(n - 1);
        long y = fib(n – 2);
        sync();
        return x + y;
}
```

Mark methods as

Spawnable.

They can run in parallel.

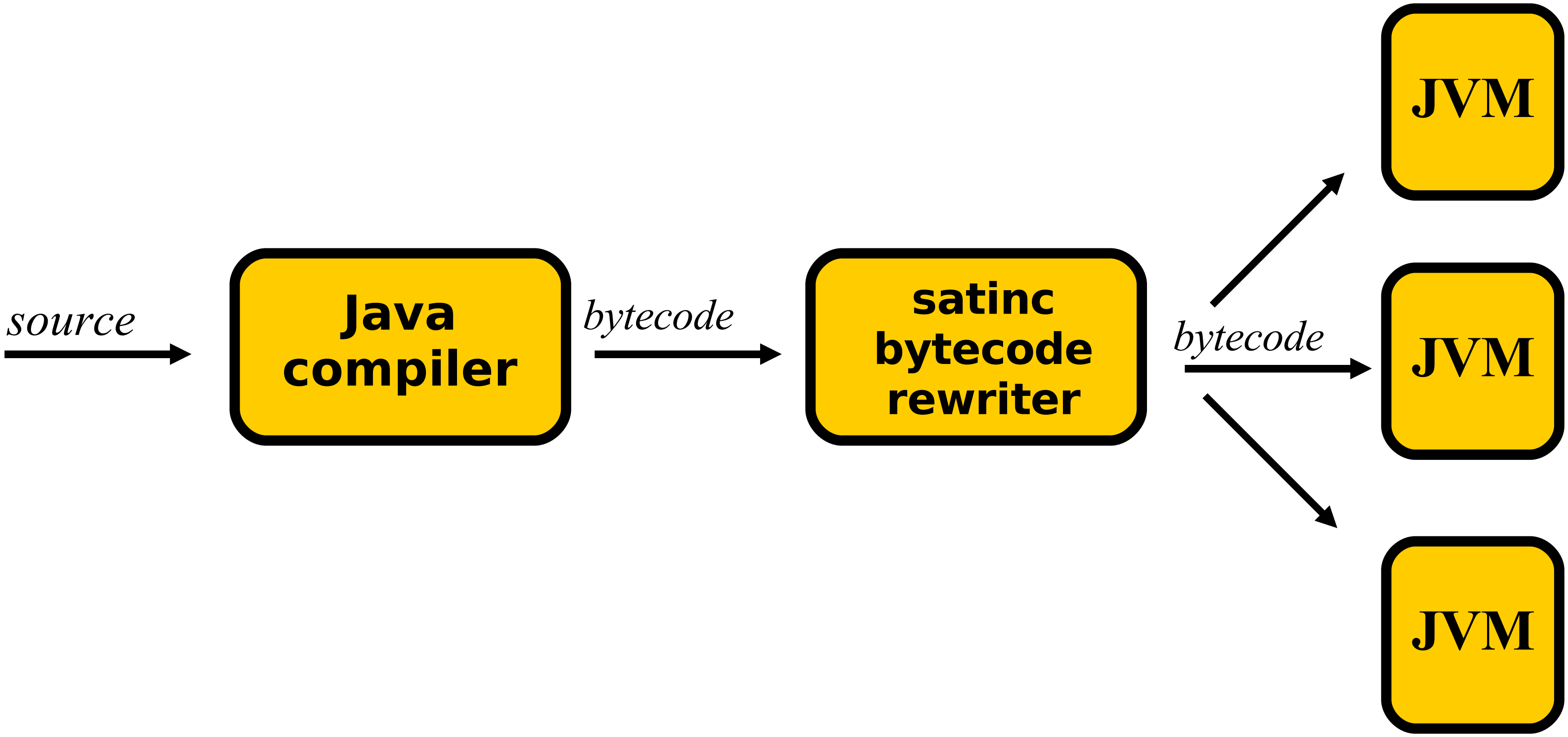
Parallel Fibonacci

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

public long fib(int n) {
        if (n < 2) return n;

        long x = fib(n - 1);
        long y = fib(n – 2);
        sync();
        return x + y;
}
```

Mark methods as
Spawnable.
They can run in parallel.

Wait until spawned methods are done.

Compiling Satin Programs



Java GAT API features

- Security (deal with passwords, credentials, etc)
- Grid I/O
  - File operations, remote file access, file replication
  - Inter-process communication
- Resource Management
  - Resource brokering
  - Forking grid applications, job management
- Application Information Management
  - Global repository for application specific information
  - Query this information repository
- Monitoring
  - Grid monitoring
  - Application monitoring and steering

vl·e

ibis

vrije Universiteit

Java GAT File example

```
public class RemoteCopy {
    public static void main(String[] args) {
        GATContext context = new GATContext();

        URI src = new URI(args[0]);
        URI dest = new URI(args[1]);
        File file = GAT.createFile(context, src);
        file.copy(dest);
        GAT.end();
    }
}
```