Outlook on Existing and Future Grid Programming Paradigms

EchoGrid Workshop Athens 2008

Bernhard Schott bschott@platform.com
Guillaume Mecheneau gmecheneau@platform.com
Platform EU-Research Team, Paris

Date: 9th June 2008

Platform™

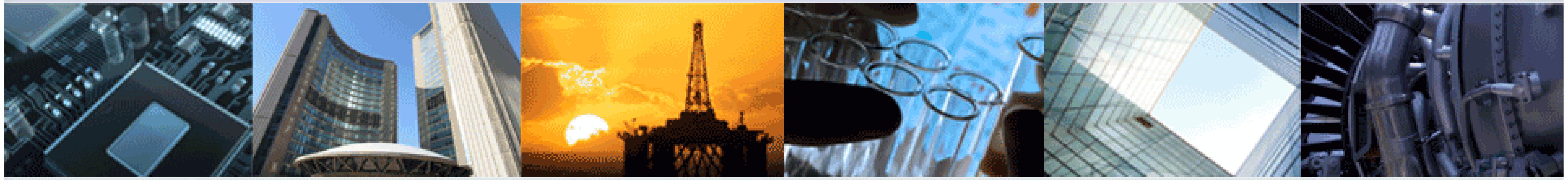
Grid Programming Paradigms

Agenda

- Platform Computing
- (Grid) Programming Paradigms
- State of the Art and possible extensions
  3 Examples:
  1. Grid embracing the Application: Platform LSF & EGO
  2. SOA application integration: Platform Symphony
  3. On-demand resource acquisition and formation of application specific Grids: QosCosGrid
- Summary: Trends in Grid Programming

Platform™

Introduction



Platform

Platform Computing

Platform is a pioneer and the global leader in High Performance Computing infrastructure software, delivering integrated software solutions that enable organizations to improve time-to-results and reduce computing costs.

Over 2200 Customers Worldwide

- Electronics, Financial Services, Manufacturing, Life Sciences, Oil & Gas, Government, Universities & Research, Telco …

Recognized Leader in HPC, Cluster and Grid Computing

- 16 years global experience
- Worldwide offices, resellers and partners
- 24x7 follow the sun support and services

Growing & Profitable since inception in 1992

- Self-funded
- No debt; money in bank

Founded by Songnian Zhou, Bing Wu, Jingwen Wang

Offices and VARs

Office Locations

North America

- Toronto (HQ)
- San Jose
- Washington
- Detroit
- Los Angeles
- Boston
- New York

International

- China
- Japan
- Korea
- UK
- Germany
- France

VARs

U.S.	South Africa
Italy	Sweden
Israel	Turkey
Germany	India
Spain	Malaysia
Korea	Thailand
Taiwan	Australia
Singapore	Austria
Japan	France
U.K.	China
The Netherlands	U.A.E.
Poland	Portugal
Brunei	

Platform

Big Companies Trust Us

Financial Services

- BNP Paribas
- Citigroup
- Deutsche Bank
- Fortis
- HSBC
- JP Morgan Chase
- Lehman
- Mizuho Financial
- MUFG
- Prudential
- Société Générale
- Sal Oppenheim

Electronics

- AMD
- ARM
- ATI
- Broadcom
- Cadence
- Cisco
- HP
- IBM
- Motorola
- NVIDIA
- Qualcomm
- Samsung
- ST Micro
- Synopsys
- TI
- Toshiba

Industrial Manufacturing

- Airbus
- BMW
- Boeing
- Bombardier
- British Aerospace
- Daimler & Chrysler
- GM
- Lockheed Martin
- Pratt & Whitney
- Toyota
- Volkswagen
- Xi'an Aircraft Design

Life Sciences

- AstraZeneca
- Bristol Myers-Squibb
- Celera
- Dupont
- GSK
- Johnson & Johnson
- Merck
- Novartis
- Novo-Nordisk
- Pfizer
- Wellcome Trust Sanger Institute
- Wyeth

Government & Research

- ASCI
- CERN
- CINECA
- DoD, US
- DoE, US
- ENEA
- ETH
- Fleet Numeric
- GSI
- INFN
- MaxPlanck
- SSC, China
- TACC
- TU Dresden
- Univ Tokyo

Other Business

- Bell Canada
- Cablevision
- Deutsche Telekom
- Ebay
- Starwood Hotels
- Telecom Italia
- Telefonica
- Sprint
- GE
- IRI
- Cadbury Schweppes

Platform™

Partner Ecosystem

Strategic Partners

intel® redhat. DELL™ sas® hp® invent Microsoft® IBM®

Premier Partners

GEMSTONE® Sun microsystems NICE The MathWorks Mentor Graphics® macrovision®

cadence® ANSYS ARROW® AMD vmware CSC

Novell® the BioTeam CISCO sgi®

WIPRO Applying Thought CALYPSO BULL MSC SOFTWARE SIMULATING REALITY SCS AVNET® SIMULIA

SYNOPSYS® Schlumberger Vinetech Utility Computing Service Provider

ORACLE® TANGOSOL DATA GRID SOLUTIONS NEC Satyam What Business Demands. GIGASPACES SC science + computing LSTC Livermore Software Technology Corp.

Select Partners

Apache DESIGN SOLUTIONS ATHENA DESIGN WinterLogic D'GIPRO T··Systems··· ATopTech SCHRÖDINGER. 安託集團 ATOZ GROUP

CoWare® The ESL Design Leader electric cloud carus GNS Systems HPC SYSTEMS SCS LIVING TECHNOLOGY ITOCHU

PTC Data Management Solutions KIMOTION TECHNOLOGIES COMLINK LINUX NETWORX QUIC® SiSoft ASPEED Software Formerly Powerllel

immixTechnology Selim NANGATE Engineous software TOTALVIEW TECHNOLOGIES a•s solido DESIGN AUTOMATION

CRAY Voltaire Troline Integrated Intelligence denali EVERGRID ALWAYS ON. ALWAYS OPTIMIZED. accelrys® FUJITSU it peers optimization services

INCAT BCS SONNET® gridwise tech

CVIS RCH SOLUTIONS redIT ponte the design-for-yield company KEL WOLFRAMRESEARCH MAKERS OF MATHEMATICA® T2 TECHNOLOGIES I|C|E systems ENGAGE Technology GRID Computing Solutions

Platform™

(Grid) Programming Paradigms

Platform

(Grid) Programming Paradigms

- (Grid) Programming Paradigms
  - New Programming Paradigms will include non-Grid applications or usage scenarios, e.g. apps running with our without Grid depending on problem sizes.
- Expected Future Programming Paradigm
  - Application express their needs and behavior towards infrastructure that re-arranges itself accordingly
  - Infrastructure describes resource availability and properties (e.g. HPC backbone topology) to the application that (re-)compiles or (re-) configures itself accordingly
  - Platform position paper contributed to the Oxford Challengers Workshop on Standards Roadmap to 2020: http://www.w3c.rl.ac.uk/pastevents/ChallengersWorkshop/Schott.pdf (Bernhard Schott, Chris Smith, Werner Dubitzky)

Platform

(Grid) Programming Paradigms

We assume a world were applications are capable of adjusting themselves to the available environment; at the same time intelligent environments adjust themselves to applications requirements.

Whether the resource (Grid, Storage) managers provide the necessary dynamic configuration services or the complementary launcher components are provided by the applications, this scenario will result in automatically provisioned environments that would go along with computing resources emulating different machine architectures, types.

<…>

It's this notion of application portability that needs the descriptive capabilities on the application side; portable applications that can work in the grid environment and a grid fabric that can adapt to application needs.

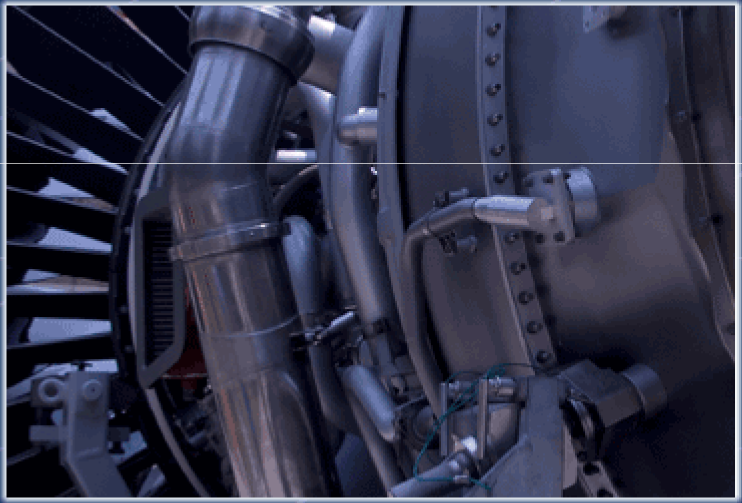
Full document: http://www.w3c.rl.ac.uk/pastevents/ChallengersWorkshop/Schott.pdf

Platform

(Grid) Programming Paradigms

- 3 Examples on State of the Art application to Grid integration
  1. Grid embracing the Application: Platform LSF
     - Platform LSF is the industry workhorse for some >2000 customers around the world. In most cases applications are not “written” for the Grid use, do not use Grid API. The Grid is in charge to adapt itself to the application by comfortable, sophisticated and even dynamic “embracement”.
  2. SOA application integration: Platform Symphony
     - Platform Symphony implements “real time” execution of some application classes (SOA) in the HPC Grid (=cluster, extended cluster). Keep It Sweet and Simple: Very easy to use APIs allow the application to request resources (=service instances) instantaneously..
  3. On-demand resource acquisition and formation of application specific Grids: QosCosGrid
     - QosCosGrid (= Quasi Opportunistic Supercomputing for Complex Systems on the Grid) Complex Systems applications use the QosCosGrid-Toolbox to parallelize their workload in order to use distributed resources.
     - Towards the QosCosGrid-Broker they express requirements and behavior.
     - The Broker acquires resources on-demand and form an application specific Grid

Platform™



Example 1: Grid embracing the Application: Platform LSF & EGO

Platform

Embracing the application: Platform LSF

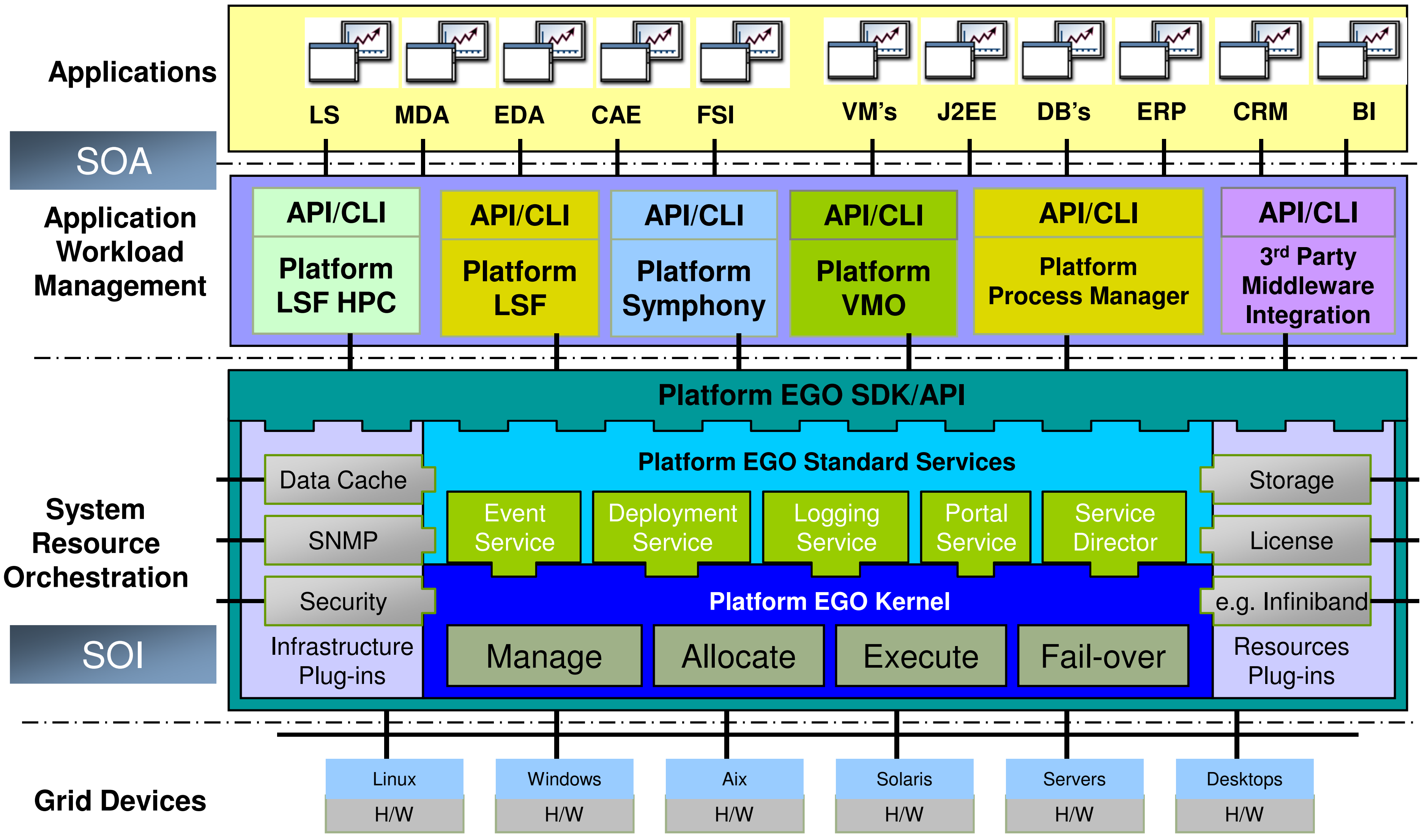
Whatever the Grid does for you already, saves you lines of code and more …

- Overview on the common layered architecture and LSF
  - Foundation: unified resource layer – Enterprise Grid Orchestrator
  - Describing Resource Allocation: Policies on supply
- Embracing the Application:
  - Interfaces: GUI, CLI, APIs, BES/JSDL/HPC-Profile++, DRMAA, Workflow (GUI, CLI, XML), ESUB
  - Arbitrary (dynamic) Resource Semantics
- Describing Application behavior: Application Profiles
- Describing Usage: Policies on consumption
  - LSF Modular Scheduler

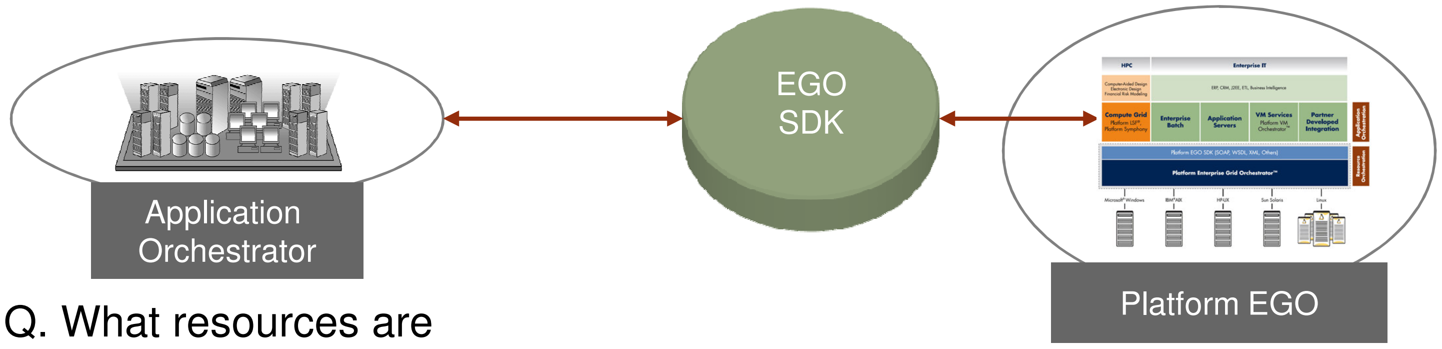
Platform

Platform Enterprise Grid Orchestrator

Open & Decoupled Architecture



Platform EGO – Simple Use Case



Q. What resources are available for my application based on current usage?

A. Resources available (Host1, Host2, etc) for Consumer

Q. Submit Allocation Request ?

A. Allocate Hosts to Consumer

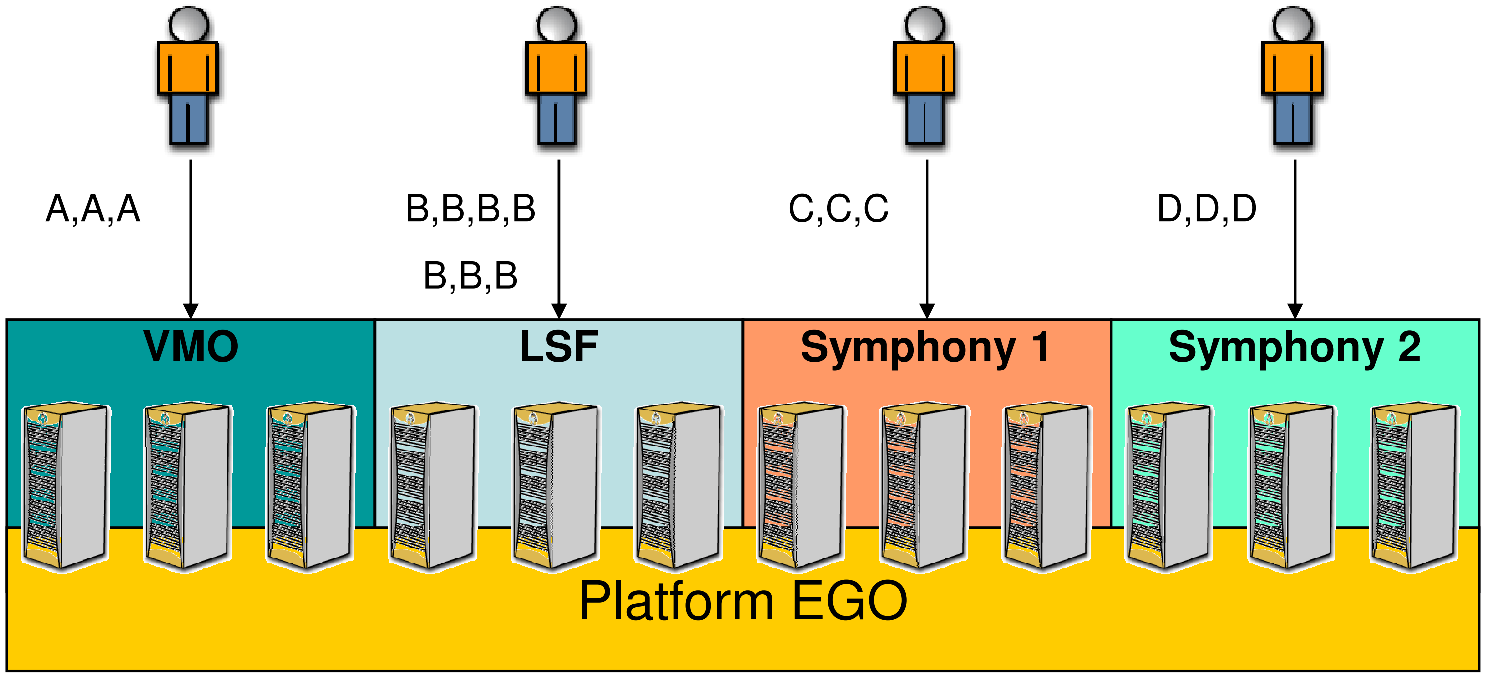
Q. Run my Services on EGO

A. Start Up required Application infrastructure (Activities and Services)

Platform

Dynamic, flexible, scalable

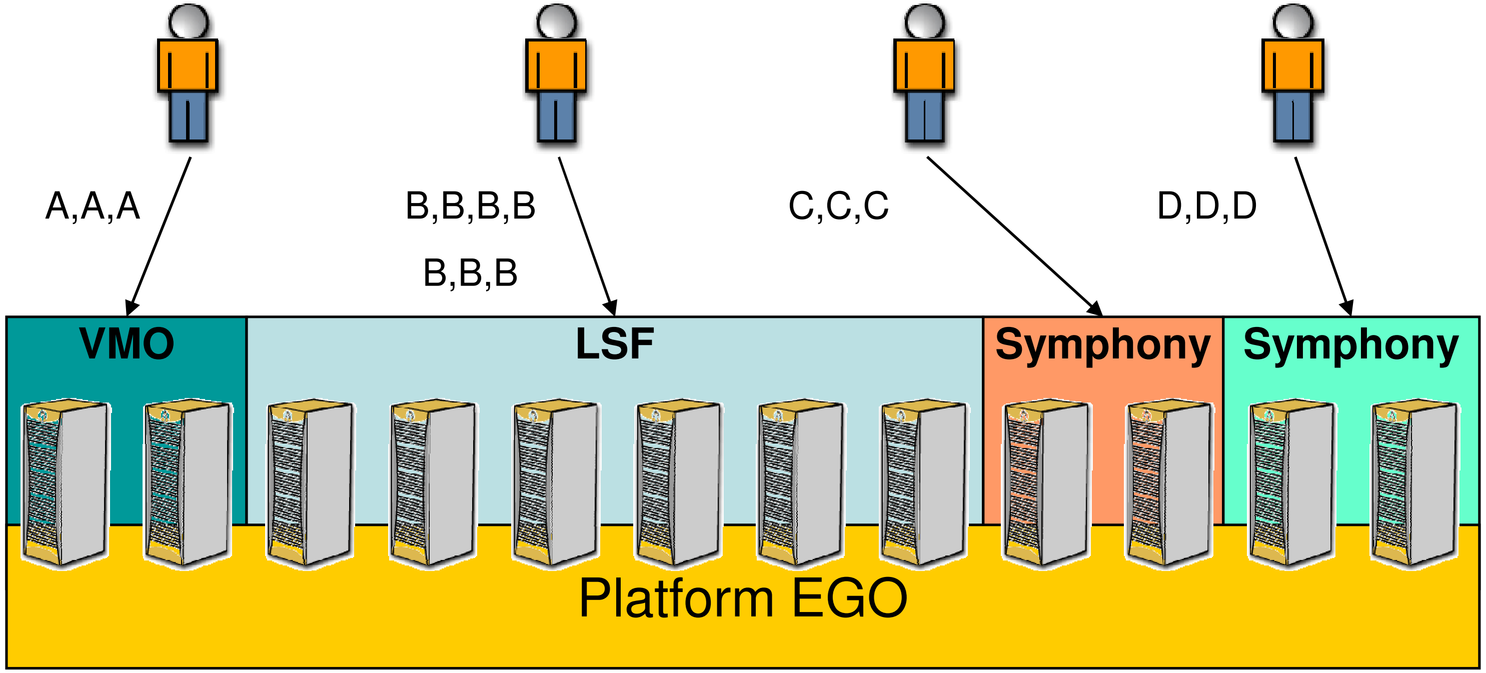
- By decoupling DRM and Workload Management, multiple Workload-Managers can dynamically flex based on workload and relative priorities – borrowing resources from other clusters and consumers
- Applications can request additional resources - even during runtime



Platform

Dynamic, flexible, scalable

- By decoupling DRM and Workload Management, multiple Workload-Managers can dynamically flex based on workload and relative priorities – borrowing resources from other clusters and consumers
- Applications can request additional resources - even during runtime



Platform™

Embracing the application: Platform LSF

- Application Interfaces:
  - GUI, CLI, APIs,
  - BES/JSDL/HPC-Profile++,
  - DRMAA,
  - Workflow (GUI, CLI, XML), (example: SAS)
  - ESUB: post submit / pre-queuing admin controlled modification of parameters
  - ELIM: dynamic resource metric update
  - Scheduler-Plug-In interface

Platform

Embracing the application: Platform LSF

- Arbitrary (dynamic) Resource Semantics
  - Add custom resource descriptions, dynamic metrics by ELIMs
  - Boolean, numeric, string, static & dynamic values, consumable
  - Example: resource [tape_drive==tape-ID]
    bsub –R "rusage tape_drive==XYZ1711" will send the job to the host with a tape drive that has loaded tape with ID=XYZ1711. If not (yet) fulfilled, job will be pending with pending reason "waiting for resource tape_drive==XYZ1711"
    Extension: ELIM to "measure" tape requests, forwarding to tape-library controller
  - Example: Application metrics. DB indicating to load balancing infrastructure remaining #connection capacity via ELIM interface.
  - Example: Application license metric. License manager interfaces via ELIM to load balancing infrastructure, indicating possible number of concurrent use..

Platform

Embracing the application: Platform LSF

- Describing Application behavior and specifics: Application Profiles
  - Independent from “per host” or “per queue” definitions
  - pre-execution, post-execution, jobstarter (job environment), queue
  - resource requirements: mem, cpu-type, scratch space, I/O, licenses, access to networks, databases, file systems,… etc.
  - limits, runtime (soft, hard), mem (per job / per process), limits on processes, threads, …
  - check-pointing method, signaling (signal sequence)
  - chunk (size)
- Controlling application runtime behavior
  - Exit codes (lists of -“success” exit codes, rerun exit codes, rerun but not on same host exit codes, max #reruns / #requeues)
  - Trigger on run too long / too short
  - Trigger on lack of activity & mem growth. (Not mem limit!)
  - Triggers optional actions like (default: email to admin) bstop –u $username all

Platform

Embracing the application: Platform LSF

- Describing Usage: Policies on consumption
  - Fairshare methods: host partition, cross-queues, inter-queues, hierarchical,
  - Limits matrix: per queue, per user, per host (queue based, host based), per system
  - Policies: priorities, preemption, dynamic priority, parallel-greedy (job slot, mem), backfill, infinite backfill, absolute priority, checkpointing, migration, advance reservation …
  - SLA / goal based scheduling: deadline, speed, throughput, resource-SLA, combinations, … (more next slides)
  - Scheduling target host/CPU/core (per host definition)
  - LSF Modular Scheduler: custom policy plug-ins – use your own policy without need to rewrite the whole scheduler.
- System reliability:
  - self-healing, recovery from incidents,
  - policy driven proactive problem containment, “black hole” isolation
  - **no** job loss during operation or in error condition, reconfiguration or failover

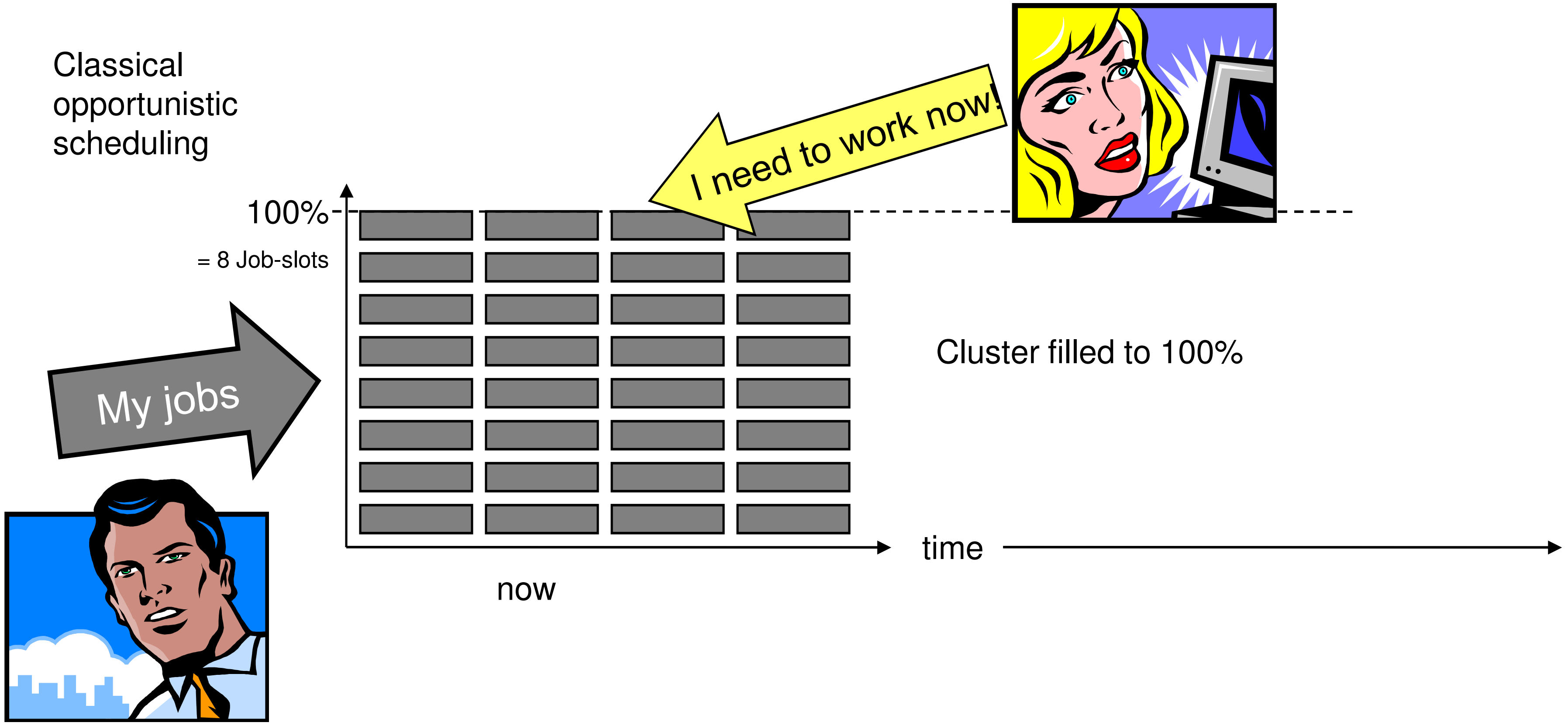
Platform

LSF: SLA scheduling

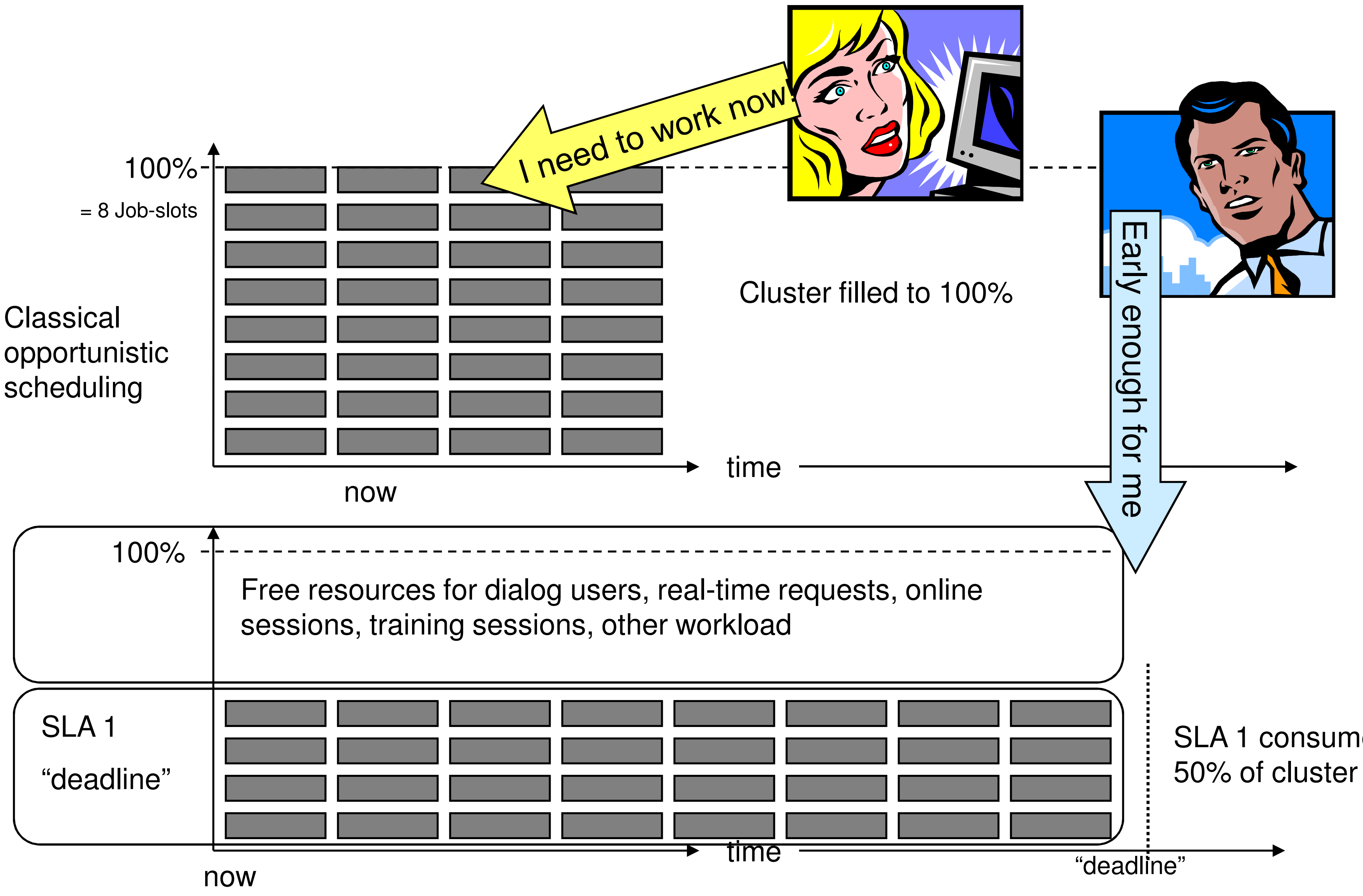
- LSF implements and executes SLAs for workload submitted to a “service class” (e.g. deadline, throughput, speed, …)
- SLA is proactively managed by allocating more resources or starting more jobs to achieve scheduling goal
- Handing over estimated run times for the jobs improve scheduler precision – otherwise, scheduler will learn about the runtime
- To resources, this translates into “least impact scheduling”
  - A given set of resources (= a (set of) cluster(s)) can serve more “happy” scientists at the same time
  - = progress in more projects at the same time
  - = faster turn-over, shorter time to results for more projects a the same time.

LSF-SLAs : SLA "Deadline"

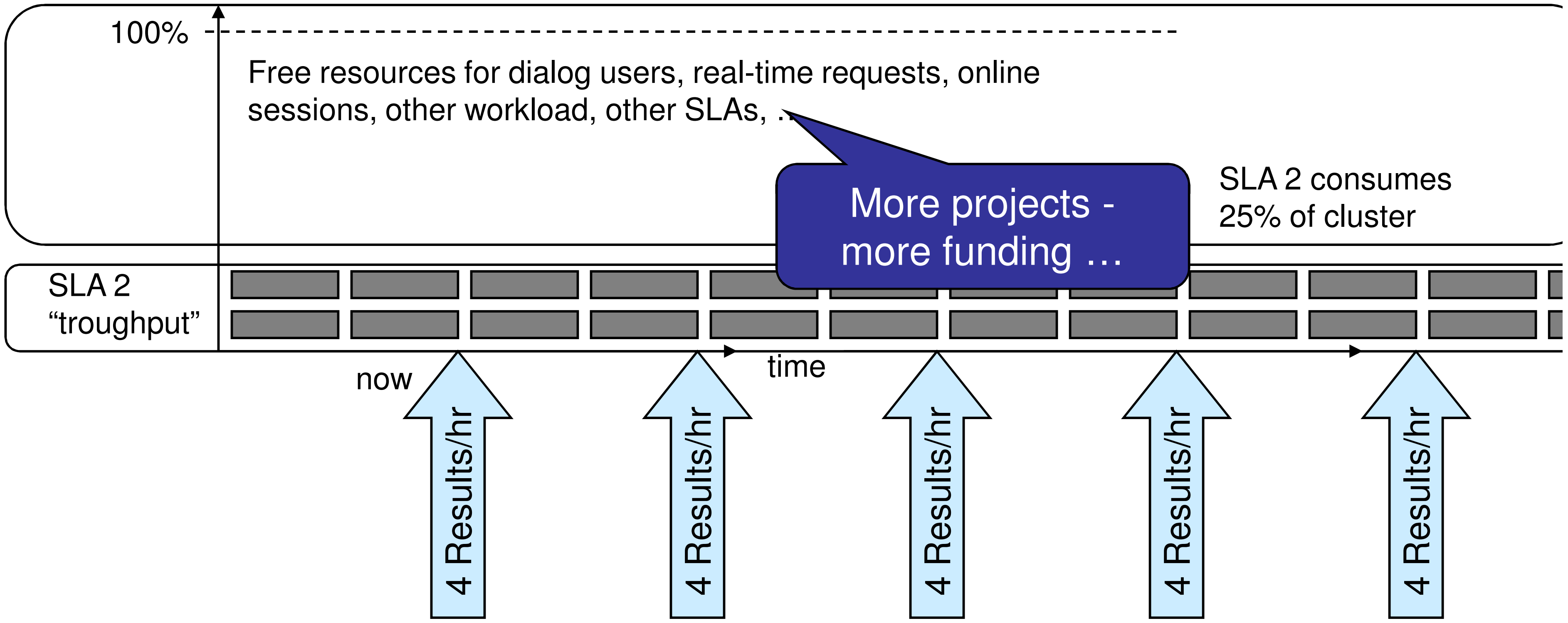
Classical
opportunistic
scheduling

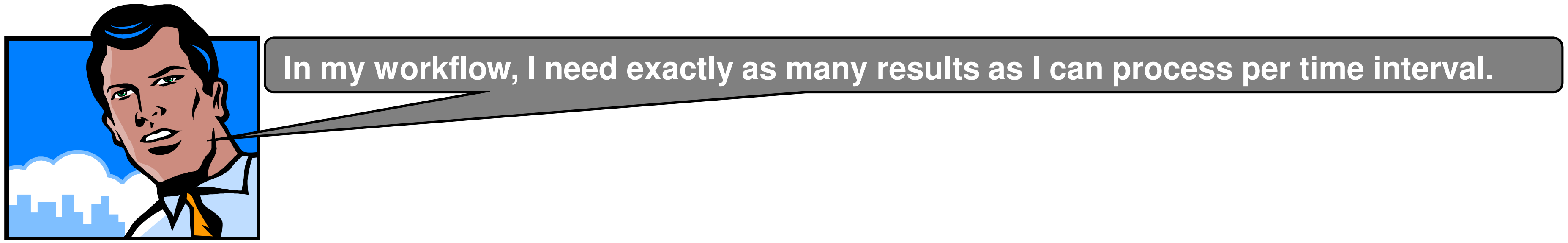


LSF-SLAs : SLA “Deadline”


I need to work now!
100%
= 8 Job-slots
Classical opportunistic scheduling
Cluster filled to 100%
Early enough for me
time
now
100%
Free resources for dialog users, real-time requests, online sessions, training sessions, other workload
SLA 1
“deadline”
SLA 1 consumes 50% of cluster
time
now
“deadline”

LSF-SLAs : SLA “Throughput”


100%
Free resources for dialog users, real-time requests, online sessions, other workload, other SLAs, …
More projects - more funding …
SLA 2 consumes 25% of cluster
SLA 2 “troughput”
now
time
4 Results/hr
4 Results/hr
4 Results/hr
4 Results/hr
4 Results/hr


In my workflow, I need exactly as many results as I can process per time interval.

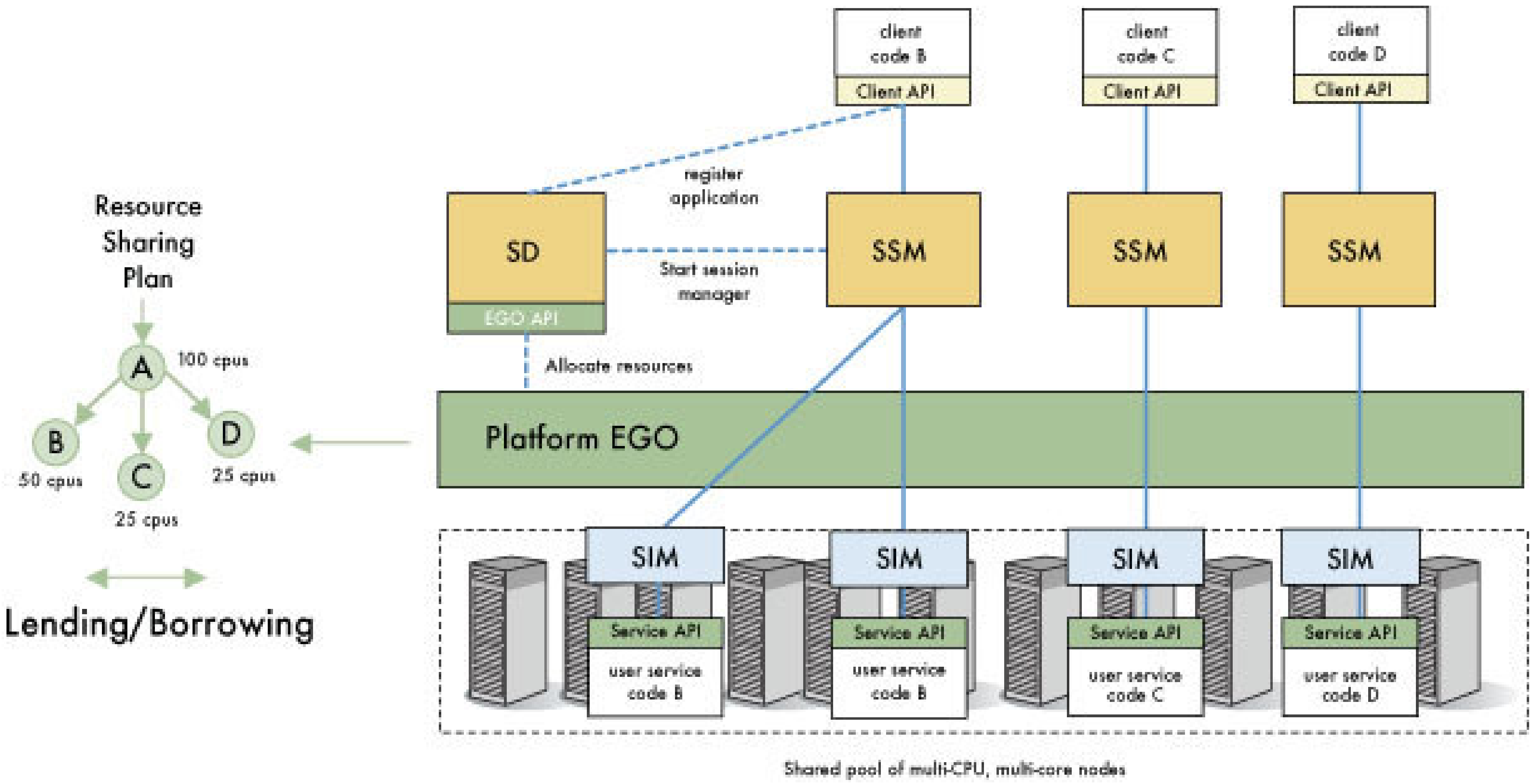
(Grid) Programming Paradigm
Embracing the application: Platform LSF

- How are those scheduling features related to a Grid Programming Paradigm? (question from EchoGrid Athens'08)
- By embracing the application for its complete lifecycle in the Grid, Platform LSF simplifies Grid integration for applications
  - Application profiles implement a simple information exchange between application and infrastructure.
  - Distinct application, consumption and resource policies allow for dynamic adaptation and balancing between application and infrastructure requirements
  - Self-management functions for automated reaction on not-intended application or infrastructure behaviour care for incidents
- These rich infrastructure features take away the need to implement respective functions in the application code
- Build your application to your application needs
- For using the Grid, just supply an application profile giving instructions to the infrastructure: “This is how I want to be executed”

Platform™

Example 2: HPC-SOA application integration Platform Symphony

Platform Symphony – HPC-SOA



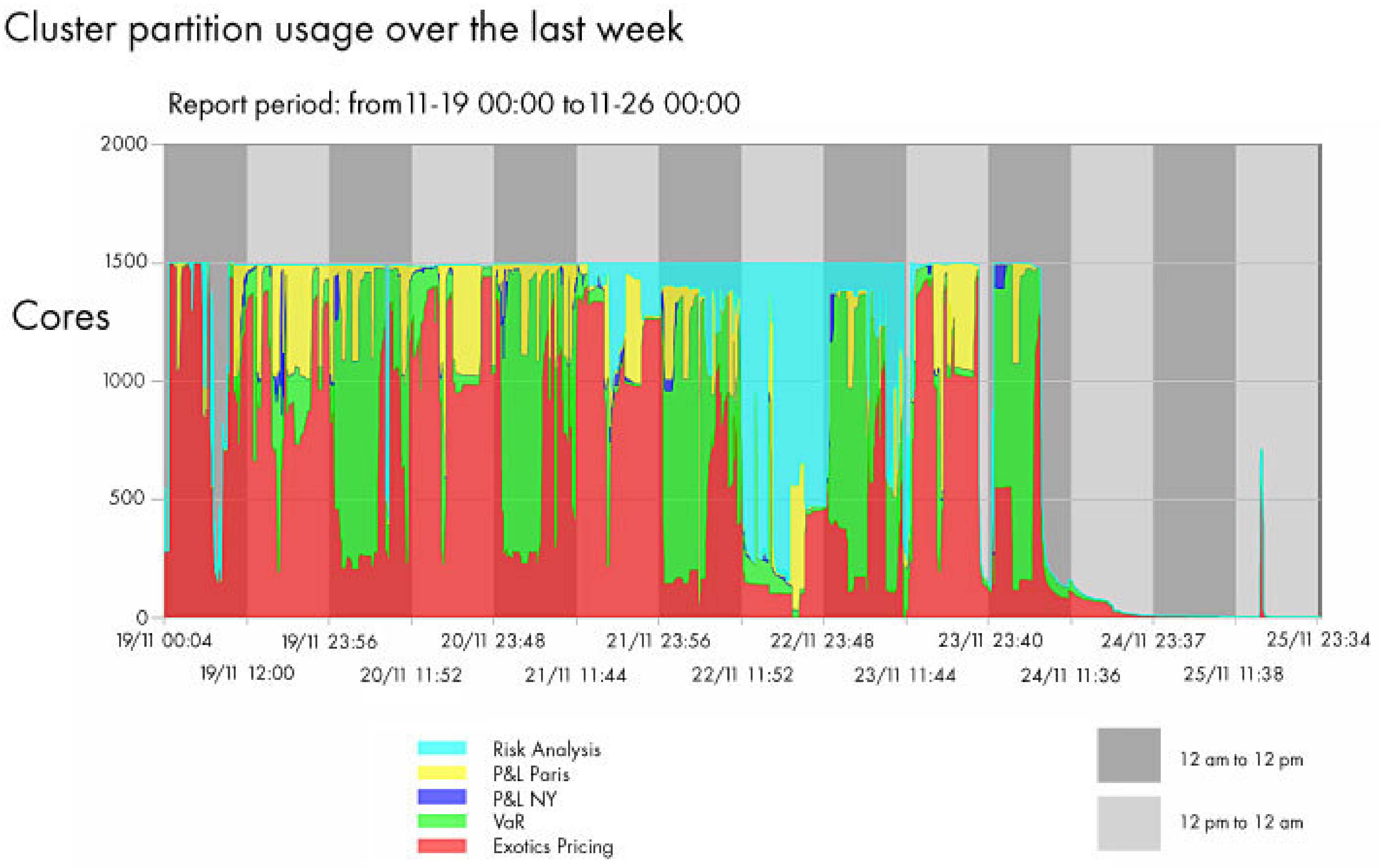
Platform

Performance, at any Scale

	Symphony at IBM DCCoD
Scalability 1,000 concurrent clients, 100 applications	**20,000+ CPU's** **simulated on 1,000 physical CPUs in one cluster**
CPU Utilization 1-100 clients, 1 sec task, 1KB message, 2,000 CPU	**98%**
Task Throughput 1KB Message	**2,700 messages/sec**
Single Task Round Trip	**2.4 ms**
Single Session Round Trip 100KB common data, 10 second 1KB Task	**11.8 ms**

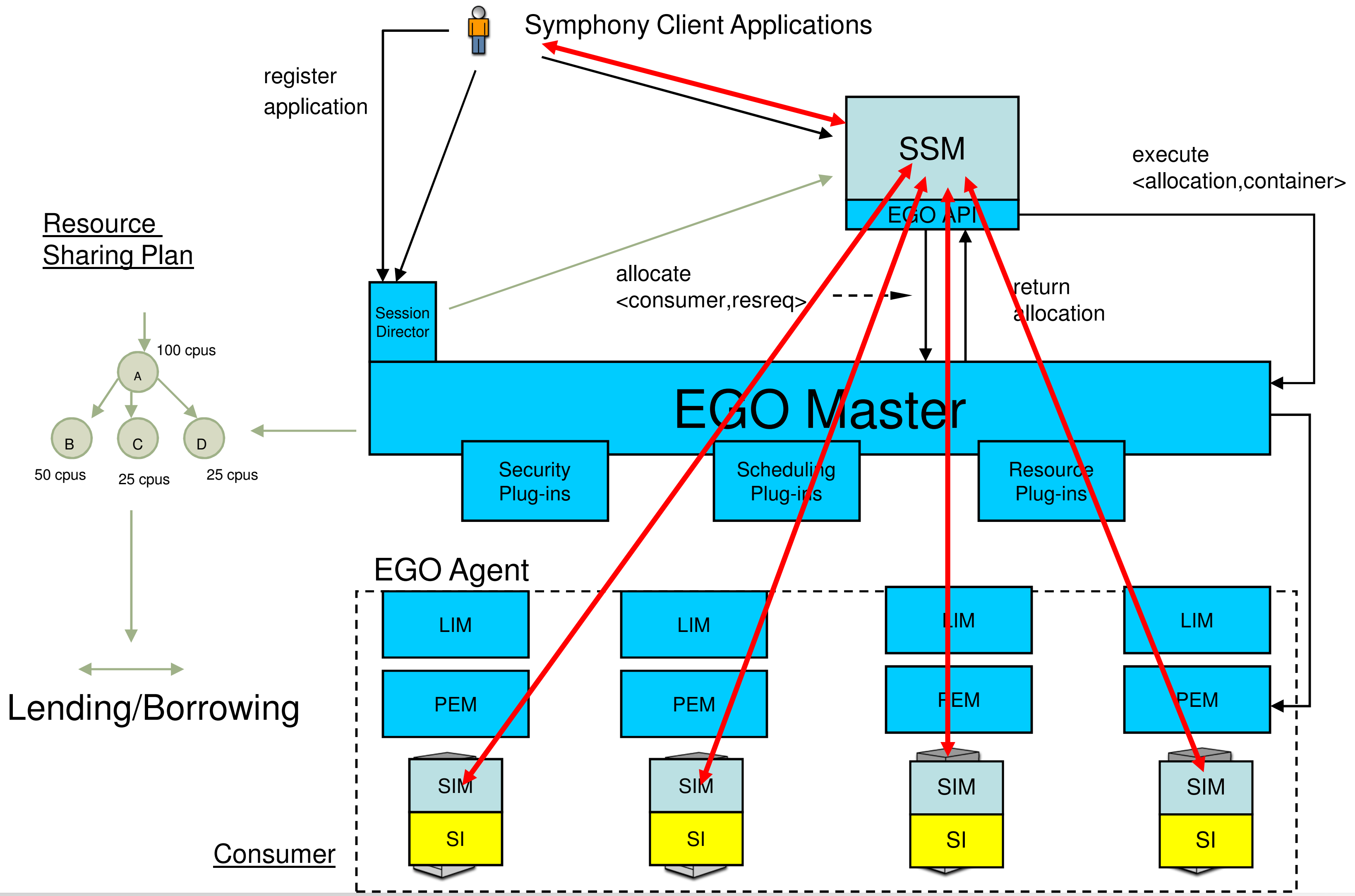
Platform

Enterprise wide Grid … Sharing with SLA



Platform

Application execution

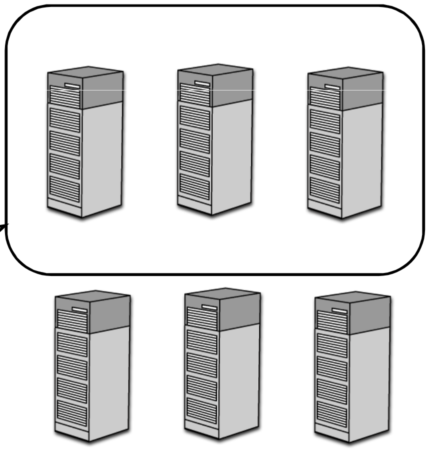


SOA application integration: Platform Symphony

- Symphony: parameter sweep
  1. decompose the problem
  2. execute parameter sweep
  3. merge results / aggregation of results
- Keep It Sweet and Simple application integration:

```
static double GridMonteCarloPi(int simulations) {
    // connect to the grid and create a session
    Connection connection = SoamFactory.Connect("computePi");
    Session session = connection.CreateSession(...);





}
```

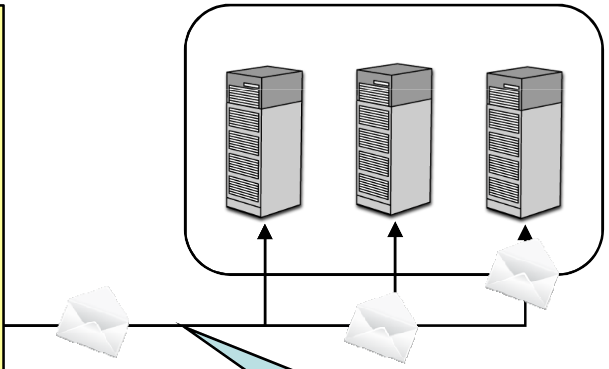
Ask the grid to allocate resources for the "computePi" application, and make them available in a session



SOA application integration: Platform Symphony

- Symphony: parameter sweep
  1. decompose the problem
  2. execute parameter sweep
  3. merge results / aggregation of results
- Keep It Sweet and Simple application integration:

```
static double GridMonteCarloPi(int simulations) {
     // connect to the grid and create a session
     Connection connection = SoamFactory.Connect("computePi");
     Session session = connection.CreateSession(…);

     // send the tasks on the grid
     int numTasksToSend = 10;
     double numberOfSimulationsPerTask = simulations/numTasksToSend;
     for (int taskCount = 0; taskCount < numTasksToSend; taskCount++)  {
               session.SendTaskInput(numberOfSimulationsPerTask);
     }









}
```



Send parameters of sweep as input messages to compute nodes

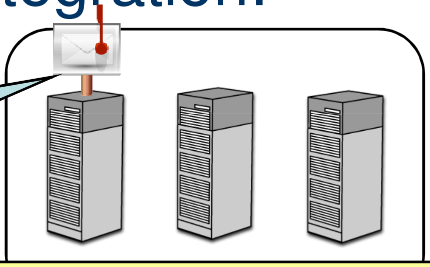
Platform

SOA application integration: Platform Symphony

- Symphony: parameter sweep
  1. decompose the problem
  2. execute parameter sweep
  3. merge results / aggregation of results
- Keep It Sweet and Simple application integration:

```
static double GridMonteCarloPi(int simulations) {
    // connect to the grid and create a
    Connection connection = SoamFa
    Session session = connection.Cre

    // send the tasks on the grid
    int numTasksToSend = 10;
    double numberOfSimulationsPerT
    for (int taskCount = 0; taskCount 
            session.SendTaskInput(numberOfSimulationsPerTask);
    }




                                        ;
}
```

Upon message reception each compute node executes the service code and returns a result



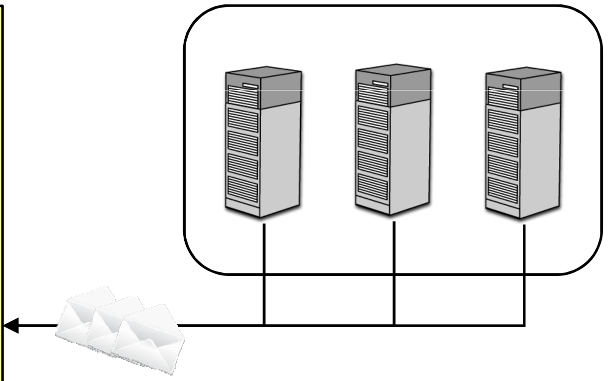
```
public override void OnInvoke(TaskContext taskContext)
    {
        double hits = 0; double x, y;
        double simulations = (double) taskContext.GetTaskInput();
        // seed
        Random Rnd = new Random(1234);
        // random throw for each simulation
        for (int i = 1; i < simulations; i++)
        {
           x = Rnd.NextDouble();y = Rnd.NextDouble();
           hits += ((x * x + y * y) <= 1) ? 1 : 0;
        }
        // return the number of hits
        taskContext.SetTaskOutput(hits);
    }
```

Platform

SOA application integration: Platform Symphony

- Symphony: parameter sweep
  1. decompose the problem
  2. execute parameter sweep
  3. merge results / aggregation of results
- Keep It Sweet and Simple application integration:

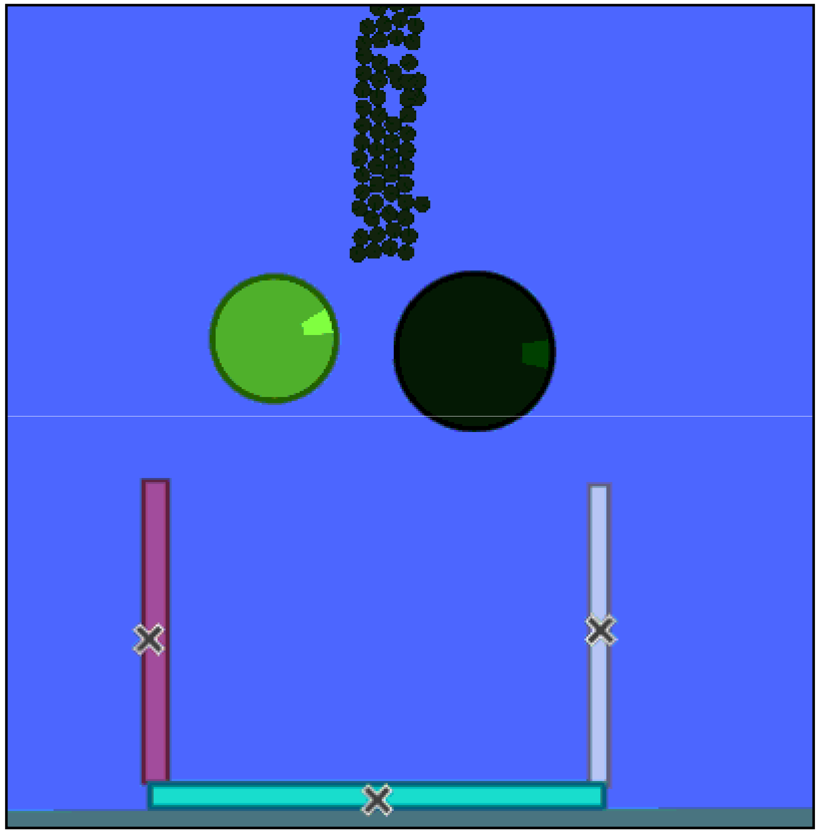
```
static double GridMonteCarloPi(int simulations) {
    // connect to the grid and create a session
    Connection connection = SoamFactory.Connect("computePi");
    Session session = connection.CreateSession(…);

    // send the tasks on the grid
    int numTasksToSend = 10;
    double numberOfSimulationsPerTask = simulations/numTasksToSend;
    for (int taskCount = 0; taskCount < numTasksToSend; taskCount++)  {
            session.SendTaskInput(numberOfSimulationsPerTask);
    }

    // get results back and aggregate them
    EnumItems enumItems = session.FetchTaskOutput((ulong)numTasksToSend);
    foreach (TaskOutputHandle output in enumItems) {
        hits += (double) output.GetTaskOutput();
    }
    return 4 * hits / simulations;
}
```



Client program receives the messages and aggregates the results

Platform

Smaller tasks for a fluid approach



On the grid, bigger is not necessarily better

Platform

SOA application integration: Platform Symphony

- Handle reliability in large scale computation
  - rerunning a task is a drop in the ocean
  - rerunning tasks costs nothing compared to the length of the entire job
- Single task failure:
  - automated isolation of black holes - application(-session) specific
  - other application may reuse this host and run perfectly
- Potential failure reasons:
  - One (or some) corrupted data records
  - in case of parameter sweep, only that one task will fail, that tries to compute the corrupted record

Platform

SOA application integration: Flexibility

- XML Application profile (metadata) defines
  - Local action on failure
    - (e.g. "this machine's DB driver is not working properly, restart my task elsewhere")
  - Global action on failure
    - (e.g. "this job's data is corrupted, fail entire job ")
- Client application defines
  - action upon exception or result
    - (e.g. "this input didn't work, let me try another one")
- Overall, more flexibility for failure control
  - Yet simple to define
  - and not always necessary to recompile entire application to extend failure control

Platform

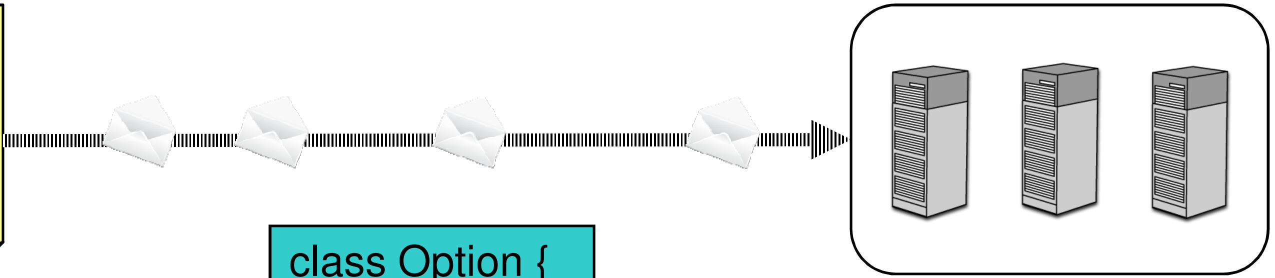
SOA application integration: Application steering

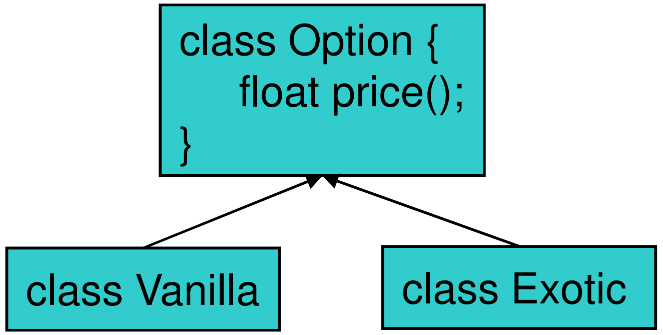
- Task is expendable: may be used to steer the application
  - Probe tasks can be used to measure application-specific properties,
    - Send a few montecarlo tasks, get their results, and use the elapsed time to predict overall computation time and reduce/increase precision of simulation
  - Or to measure grid properties
    - In effect, failure control and black listing is a sanity check on resources
- Grid as an open book
  - APIs to get resource availabily allow the application to fine-tune its own problem decomposition
    - No use to launch 100 tasks on 10 cpus, if 20 will do the job…

Platform

SOA application integration: gridify objects

```
static float GridPortfolioPricing() {
…
    // send the tasks on the grid
    for (each option in Portfolio)  {
            session.SendTaskInput(option);
    }
}
```





Client program prices a diversified portfolio by sending options on the grid

They may be Vanilla (very quick pricing , ~1s) or Exotic (long pricing, ~1h)

Problem: Exotic options finish pricing long after vanillas

More generally, how can a grid cater for complex object models ?

SOA application integration: gridify objects

```
static float GridPortfolioPricing() {
…
     // send the tasks on the grid
     for (each option in Portfolio)  {
               session.SendTaskInput(option);
     }
}
```

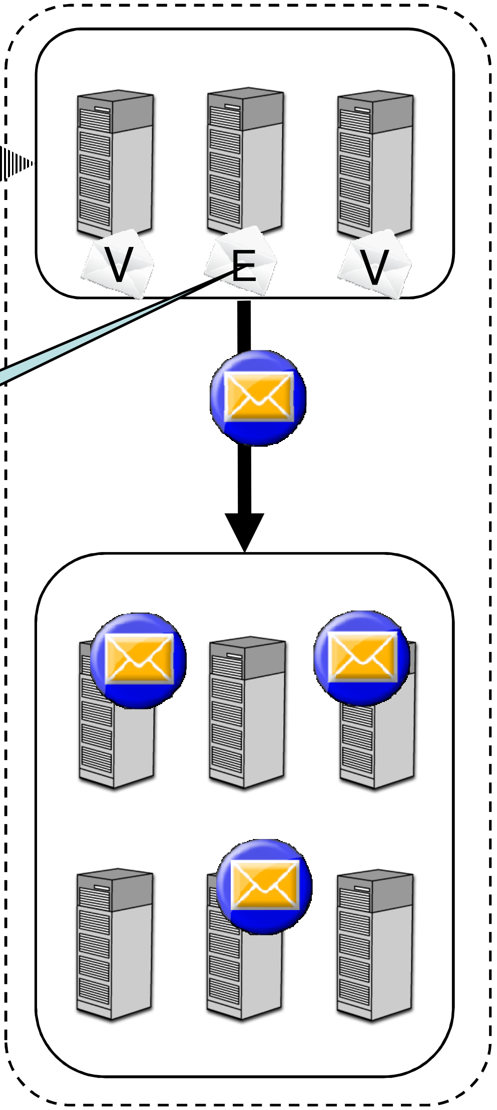
```
class Option {
     float price();
}
```

class Vanilla

class Exotic

```
float price() {
…
     // send montecarlo tasks on the grid
     for (each trajectory)  {
               session.SendTaskInput(trajectory);
     }
}
```

Exotic option reuses the grid by launching a montecarlo job



Solution: Exotic objects reuse the grid to speed up computation!

Grid Oriented Object, a.k.a. GOO, is fluid…

Platform

SOA application integration: fluid containers

- If tasks are expendable, preemption becomes possible
- Preemption means reallocation of an application’s resource to another application
  - Moving resources according to priorities becomes less costly (in terms of lost cpu time)
- Consequence is: to take full advantage of application fluidity, the underlying container of application must also be quick and flexible

Platform

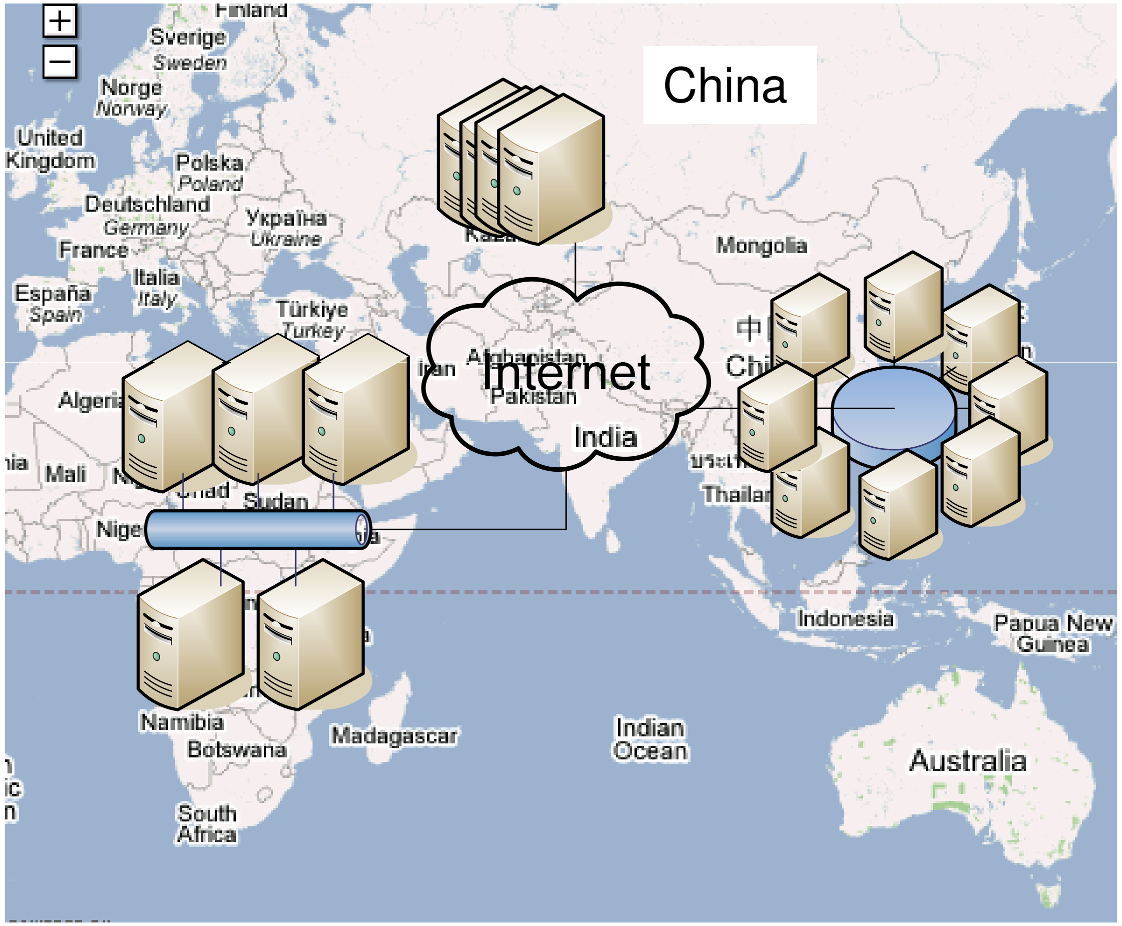
(Grid) Programming Paradigm
SOA application integration with Platform Symphony

- How do those Platform Symphony features and applied methods relate to a Grid Programming Paradigm? (question from EchoGrid Athens'08)
- By delivering high performance instantiation services to SOA applications, and provide for its messaging and complete lifecycle in the Grid, Platform Symphony simplifies Grid integration for SOA applications
  - Easy to use API provide for short time to results. Low number of additional code lines (see example)
  - Detailed policies and resource SLAs allow for dynamic adaptation and balancing between application and infrastructure requirements. (lending & borrowing between application domains based on controlled policies)
- Application steering looks for the optimal balance between resource allocation size and problem decomposition – automatically achieving best possible performance AND best resource utilization even for applications with dramatically varying runtimes per problem.
  - In the systematic approach towards improved usability of Grids, we investigate whether application steering – up to now done application specific – will become a general infrastructure feature
- These rich infrastructure features take away the need to implement respective functions in the application code

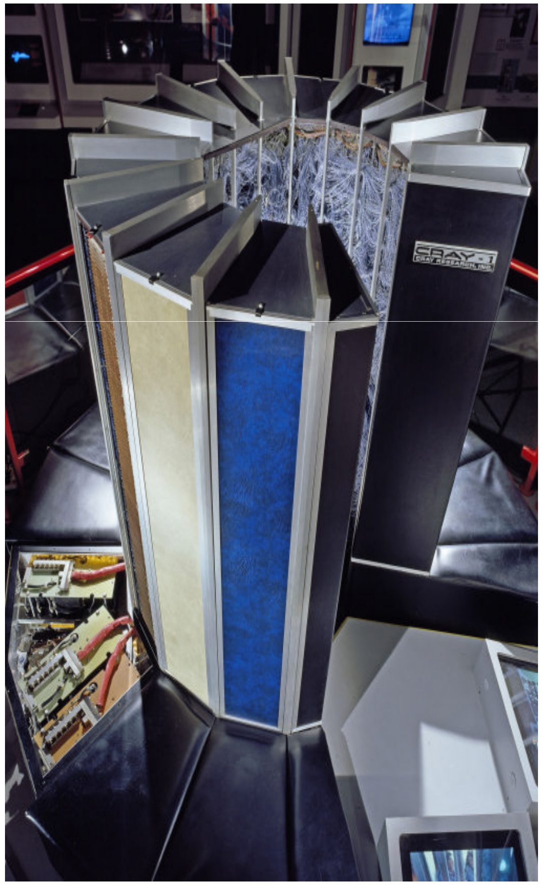
Example 3: QosCosGrid

This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

Vision:
Grid ≈ virtual supercomputer

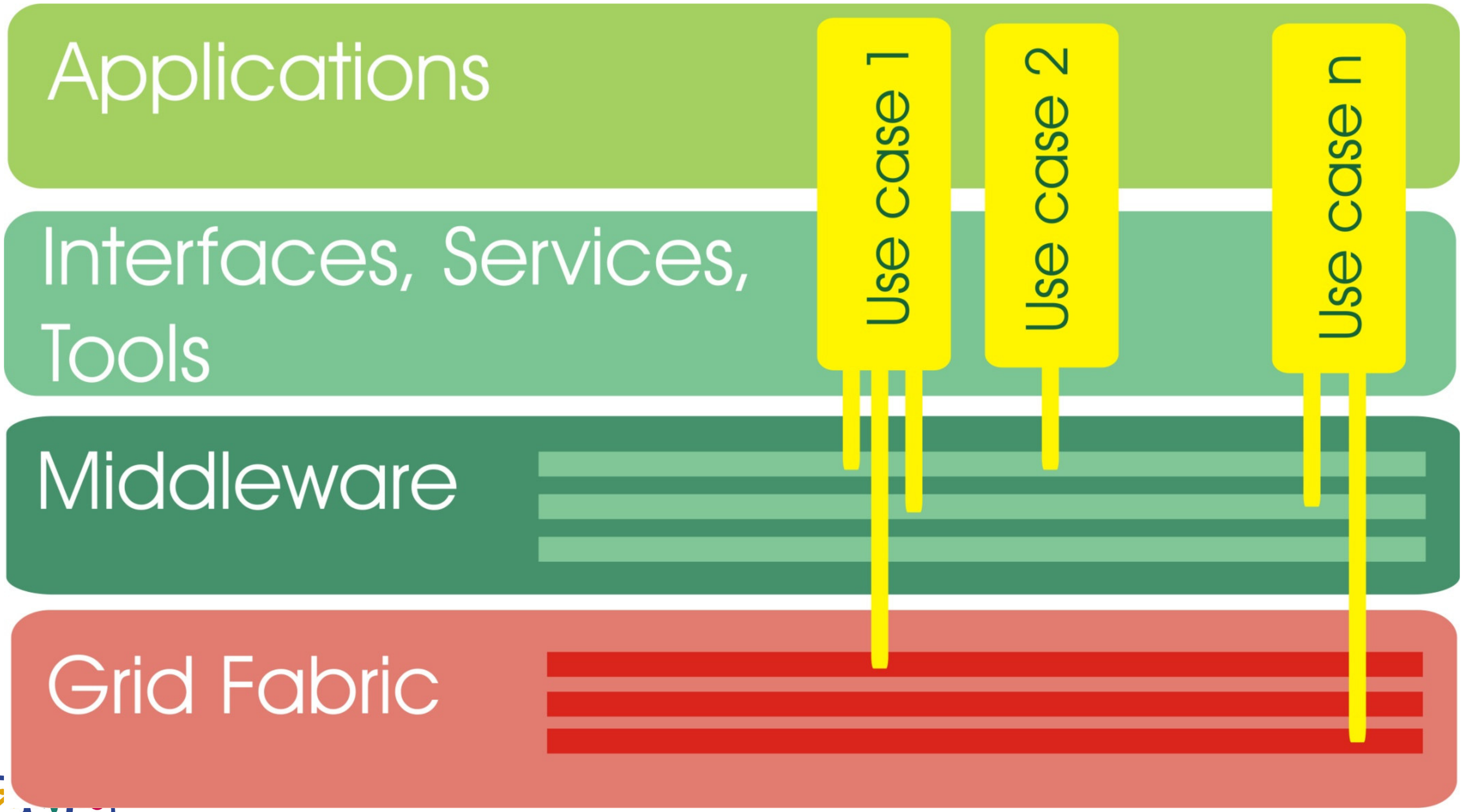


≈



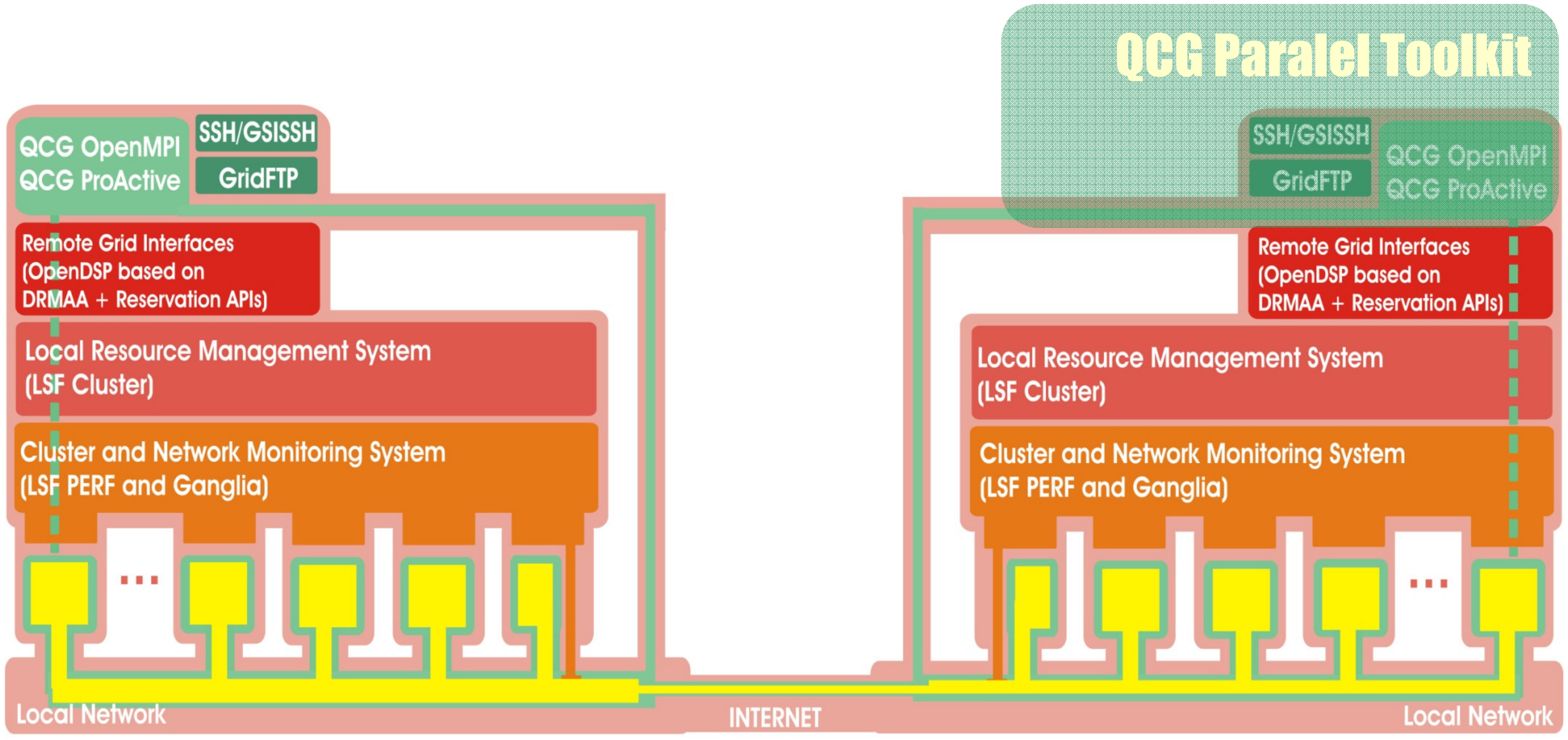
This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

Architecture Overview



This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

Current Implementation



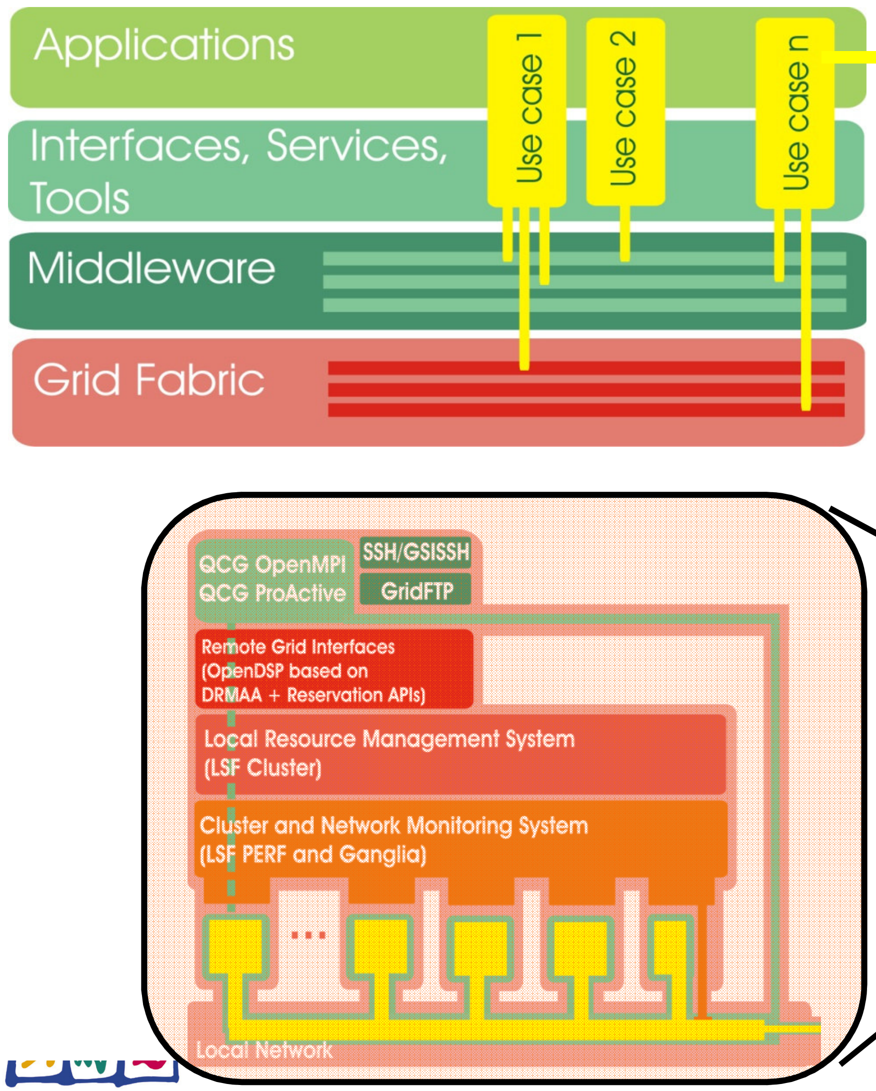
QCG Paralel Toolkit
QCG OpenMPI
QCG ProActive
SSH/GSISSH
GridFTP
Remote Grid Interfaces (OpenDSP based on DRMAA + Reservation APIs)
Local Resource Management System (LSF Cluster)
Cluster and Network Monitoring System (LSF PERF and Ganglia)
...
Local Network
INTERNET
SSH/GSISSH
GridFTP
QCG OpenMPI
QCG ProActive
Remote Grid Interfaces (OpenDSP based on DRMAA + Reservation APIs)
Local Resource Management System (LSF Cluster)
Cluster and Network Monitoring System (LSF PERF and Ganglia)
...
Local Network

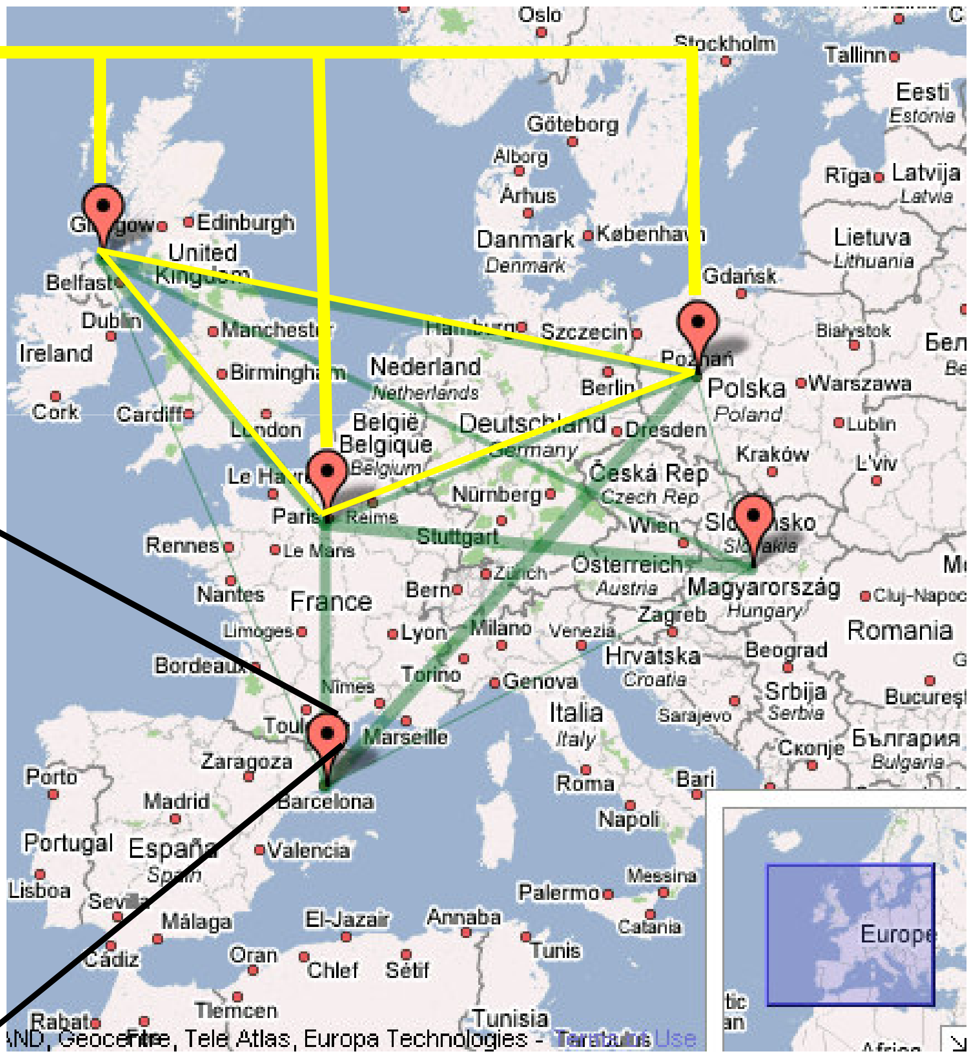
Legend

Use cases	Parallel application communication channels	Data and file channels	Resource reservation and remote job management	Network and topology monitoring

This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

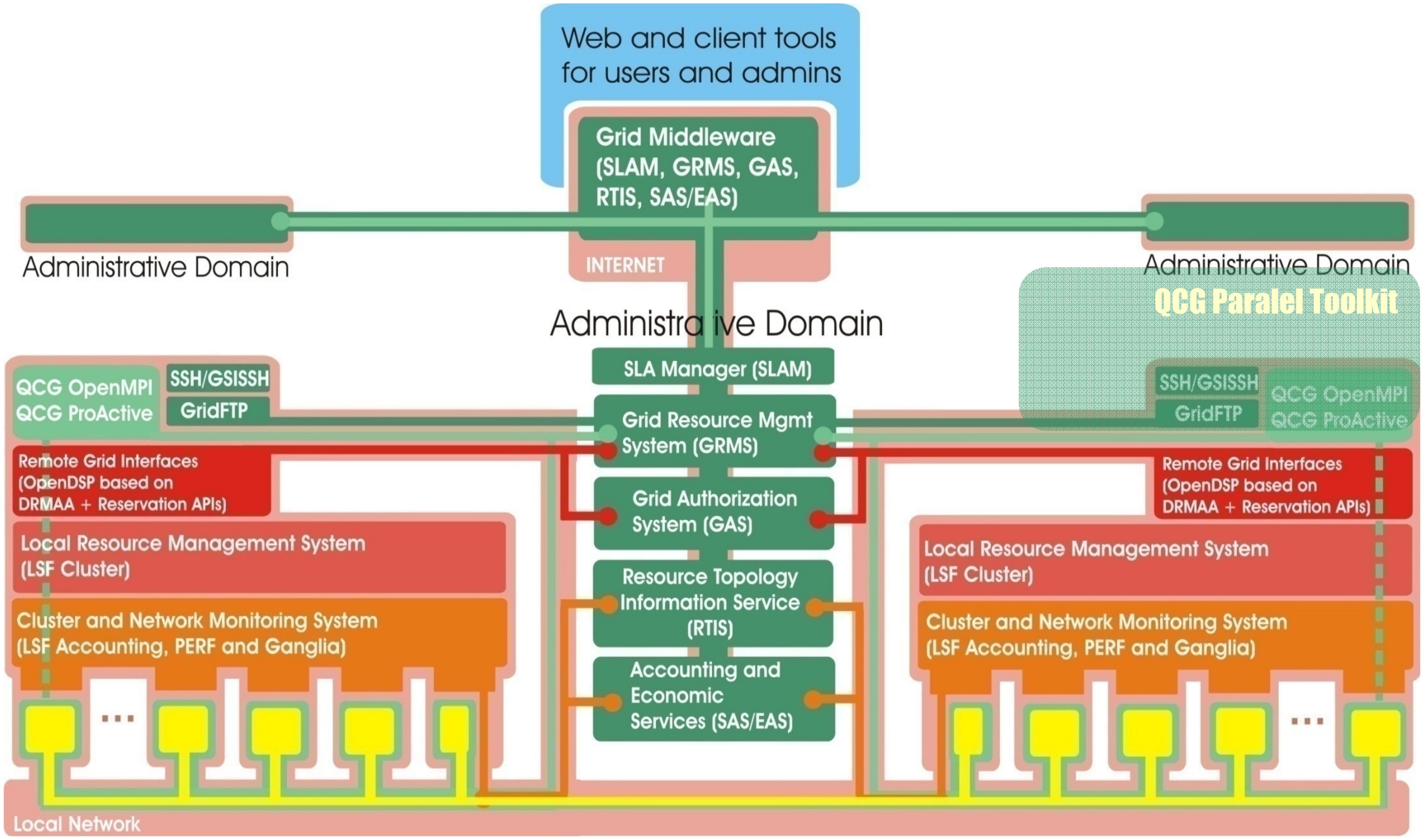
Current Implementation


Applications
Interfaces, Services, Tools
Middleware
Grid Fabric
Use case 1
Use case 2
Use case n
QCG OpenMPI
QCG ProActive
SSH/GSISSH
GridFTP
Remote Grid Interfaces (OpenDSP based on DRMAA + Reservation APIs)
Local Resource Management System (LSF Cluster)
Cluster and Network Monitoring System (LSF PERF and Ganglia)
...
Local Network



This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

Target Implementation



Web and client tools for users and admins
Grid Middleware (SLAM, GRMS, GAS, RTIS, SAS/EAS)
INTERNET
Administrative Domain
Administrative Domain
QCG Paralel Toolkit
Administrative Domain
SLA Manager (SLAM)
Grid Resource Mgmt System (GRMS)
Grid Authorization System (GAS)
Resource Topology Information Service (RTIS)
Accounting and Economic Services (SAS/EAS)
QCG OpenMPI
QCG ProActive
SSH/GSISSH
GridFTP
Remote Grid Interfaces (OpenDSP based on DRMAA + Reservation APIs)
Local Resource Management System (LSF Cluster)
Cluster and Network Monitoring System (LSF Accounting, PERF and Ganglia)
SSH/GSISSH
GridFTP
QCG OpenMPI
QCG ProActive
Remote Grid Interfaces (OpenDSP based on DRMAA + Reservation APIs)
Local Resource Management System (LSF Cluster)
Cluster and Network Monitoring System (LSF Accounting, PERF and Ganglia)
Local Network

QosCosGrid workflow

- On-demand resource acquisition and formation of application specific Grids: QosCosGrid
  - QosCosGrid (= Quasi Opportunistic Supercomputing for Complex Systems on the Grid) Complex Systems applications use the QosCosGrid-Toolbox to parallelize their workload in order to use distributed resources.
  - Towards the QosCosGrid-Broker they express requirements and behavior by xml “job profile”
  - The Broker acquires resources on-demand and form an application specific Grid, creates RTG = resource topology graph
  - RTG is used to map the application to the resources, placing MPI communicators at the right place

This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

QCG Job profile

- The QCG Job Profile document is inspired by an existing XML-based job description language supported by one of the main components of the QCG middleware called GRMS.
- End users can describe topology and resource requirements, in particular:
  - required aggregations and hierarchies of resources (computing nodes, clusters, sub-clusters, storage elements etc.),
  - required resource properties (operating system, memory, number of CPUs, speed of the CPU on a resource),
  - required network and connection properties (bandwidth, latency and capacity),
  - required applications and licenses available at destination computing resources.

This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

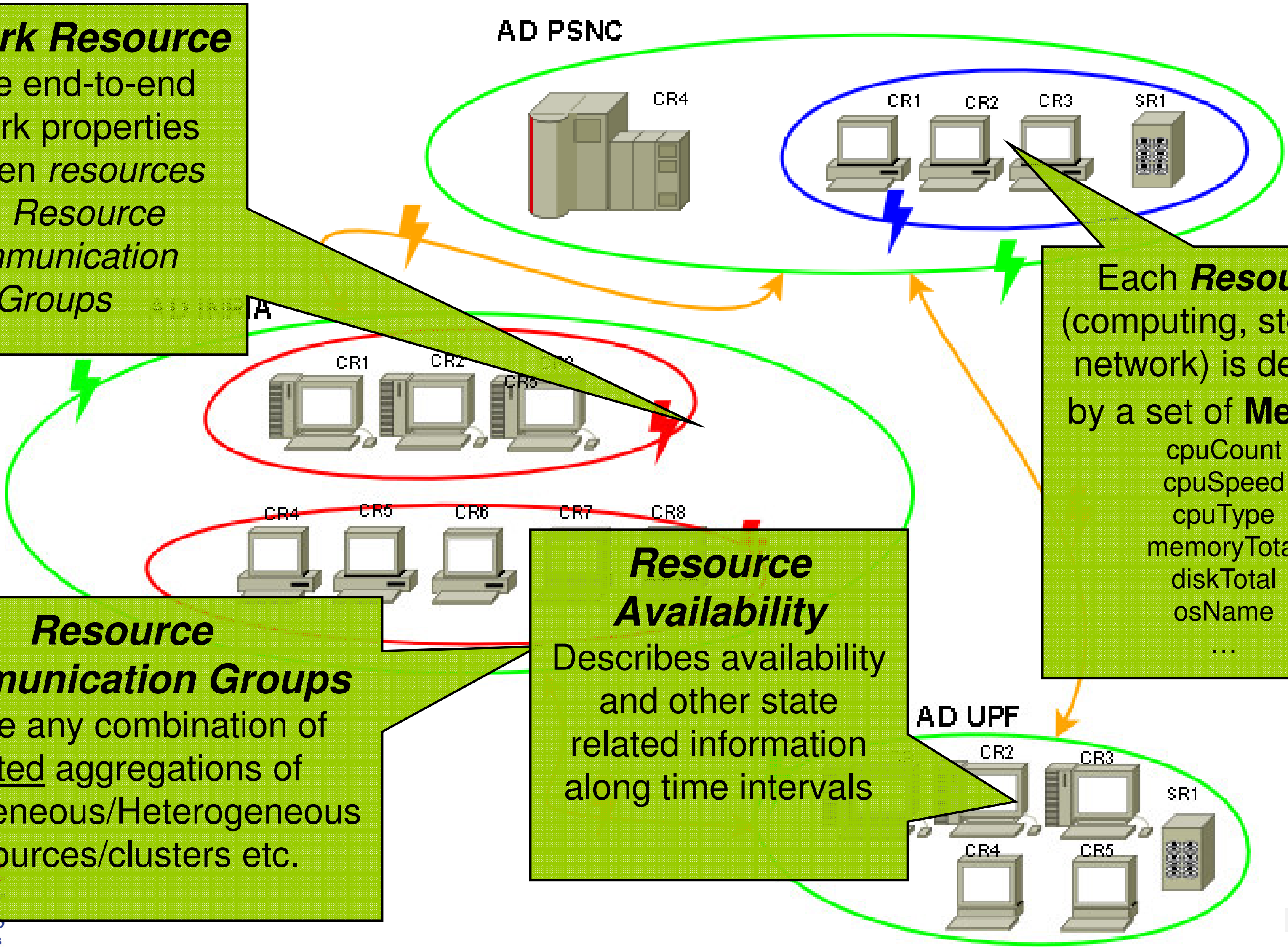
QCG Resource Description Model

- RTG (Resource Topology Graph)
- A common XML resource description language
- Provide description of:
  - Resources, tasks, processes
  - Topology
  - Communication properties
- Serves as a “bridge” between the various system components
- Used to describe, publish, evaluate, reserve and monitor heterogeneous resources across the QosCos Grid

- Supplementary Java implementation:
  - Functional behavior and logic
  - XML to Java objects marshaling/un-marshaling
  - Specialized types of RTG objects, according to the middleware requirements (i.e. Resource advertisement, Meta-scheduling, SLA’s, Monitoring, and User requirements.)

This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

QCG RTG – Resource Description View



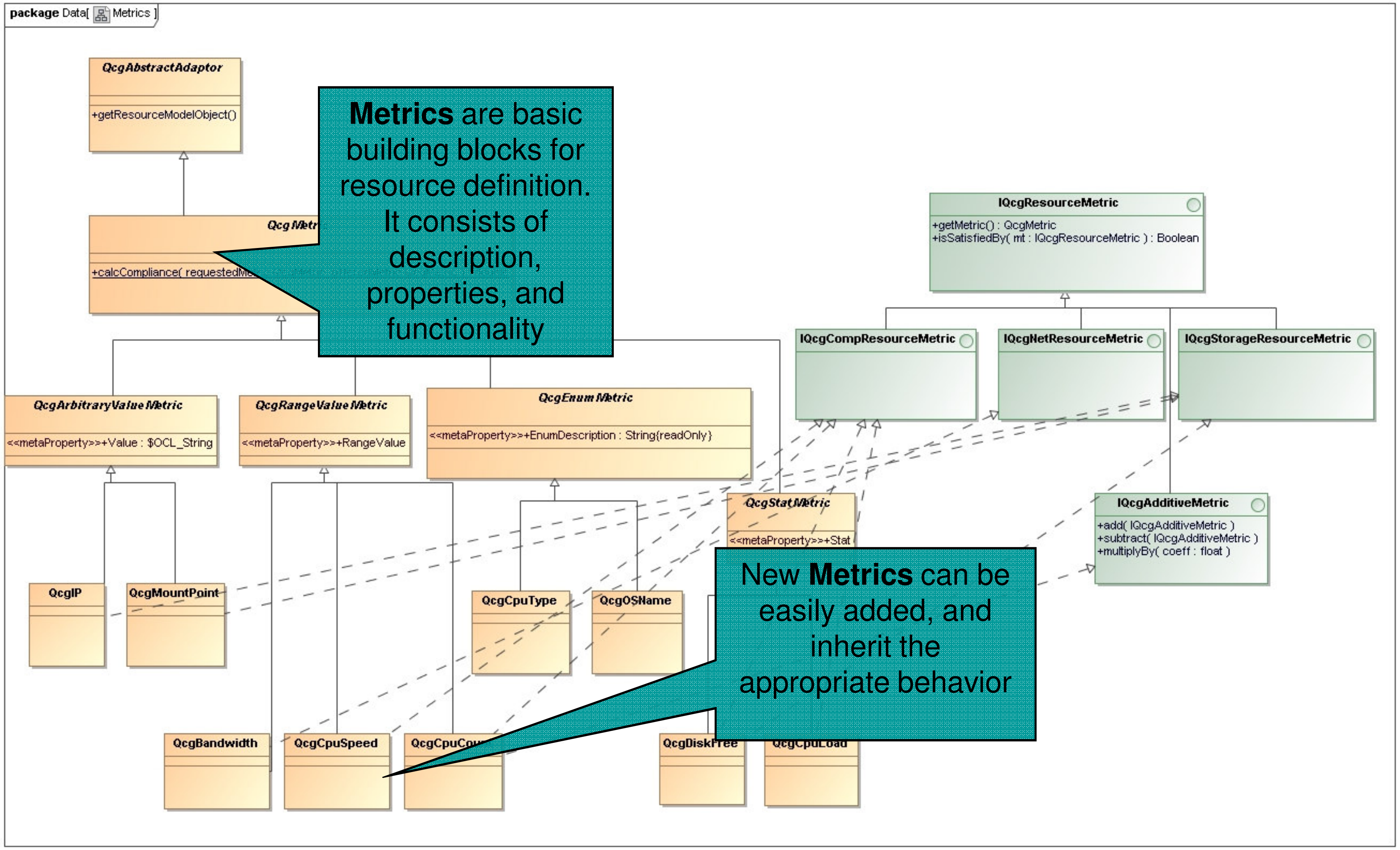
Network Resource
Define end-to-end network properties between *resources* and *Resource Communication Groups*

Each ***Resource*** (computing, storage, network) is defined by a set of **Metrics**:
cpuCount
cpuSpeed
cpuType
memoryTotal
diskTotal
osName
…

Resource Communication Groups
Define any combination of nested aggregations of homogeneous/Heterogeneous resources/clusters etc.

Resource Availability
Describes availability and other state related information along time intervals

QCG RTG - Metrics



Metrics are basic building blocks for resource definition. It consists of description, properties, and functionality

New **Metrics** can be easily added, and inherit the appropriate behavior

Resource Co-Allocation Protocol

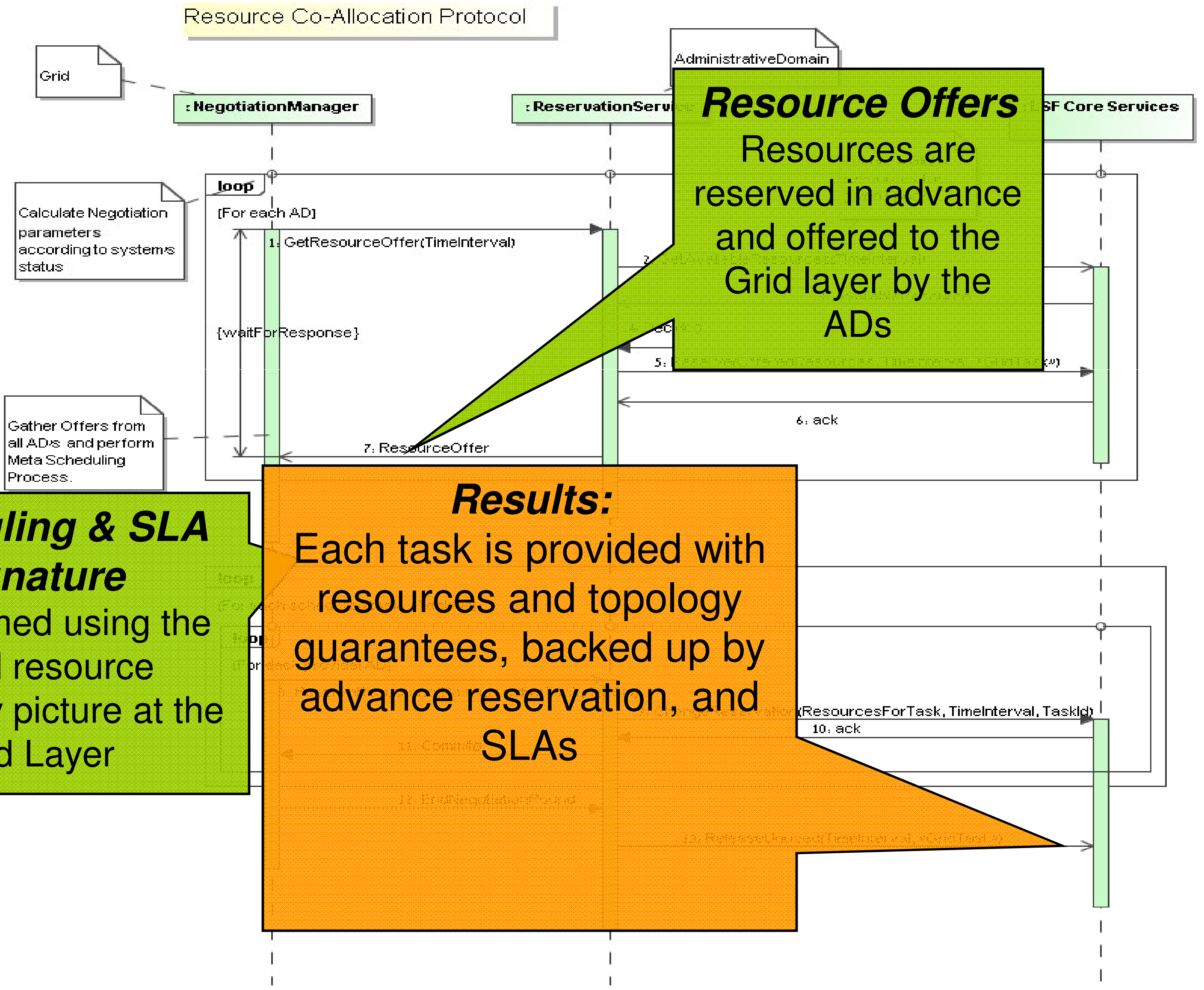
- A simple, robust 2-phase commit alike protocol for planning based co-allocation of resource across multiple ADs

Highlights:

- – Simple negotiation strategy, prevent “bargaining” which would prolong the number of negotiation rounds.
- – Global guarantied resource pricing methodology, allows each AD to estimate theirs potential profit.
- – Each AD decides what to contribute, and provides a guaranteed “Resource Offer” accordingly.
- – Support co-allocation scenarios, requiring a coordinated booking from different ADs.

This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

Resource Co-Allocation Protocol


Resource Co-Allocation Protocol
Grid
:NegotiationManager
AdministrativeDomain
Core Services
loop
[For each AD]
Calculate Negotiation parameters according to systems status
1: GetResourceOffer(TimeInterval)
{waitForResponse}
6: ack
7: ResourceOffer
Gather Offers from all ADs and perform Meta Scheduling Process.
(ResourcesForTask, TimeInterval, TaskId)
10: ack

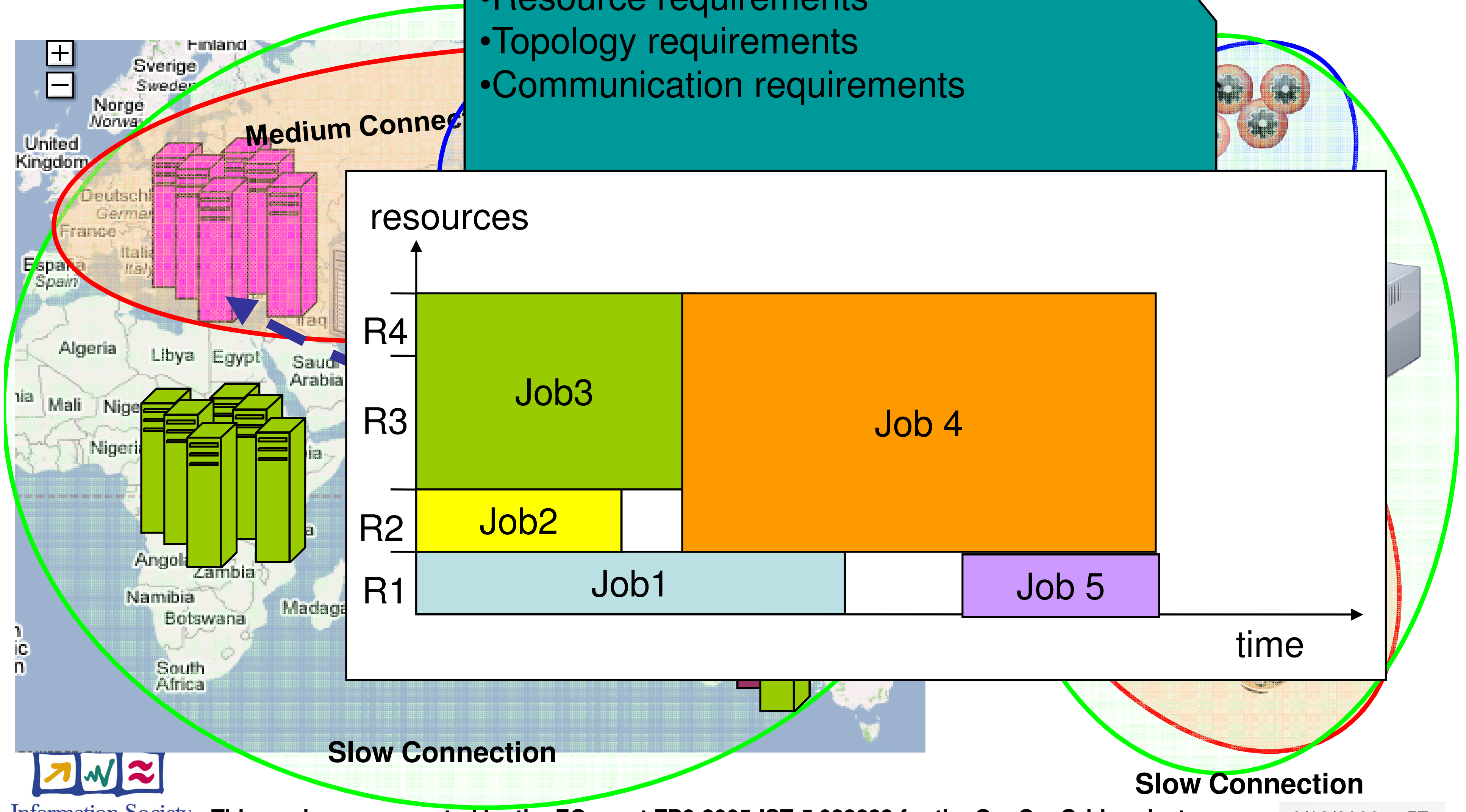
Resource Offers
Resources are reserved in advance and offered to the Grid layer by the ADs

Scheduling & SLA Signature
Is performed using the global resource availability picture at the Grid Layer

Results:
Each task is provided with resources and topology guarantees, backed up by advance reservation, and SLAs

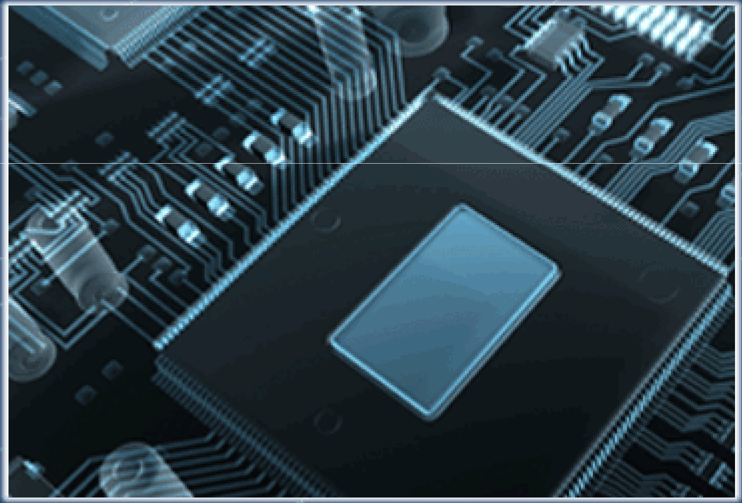
Meta Scheduler - The Objective

To provide a resource allocation for the user's tasks, constrained by:
- •Resource requirements
- •Topology requirements
- •Communication requirements



This work was supported by the EC grant FP6-2005-IST-5 033883 for the QosCosGrid project

Platform™



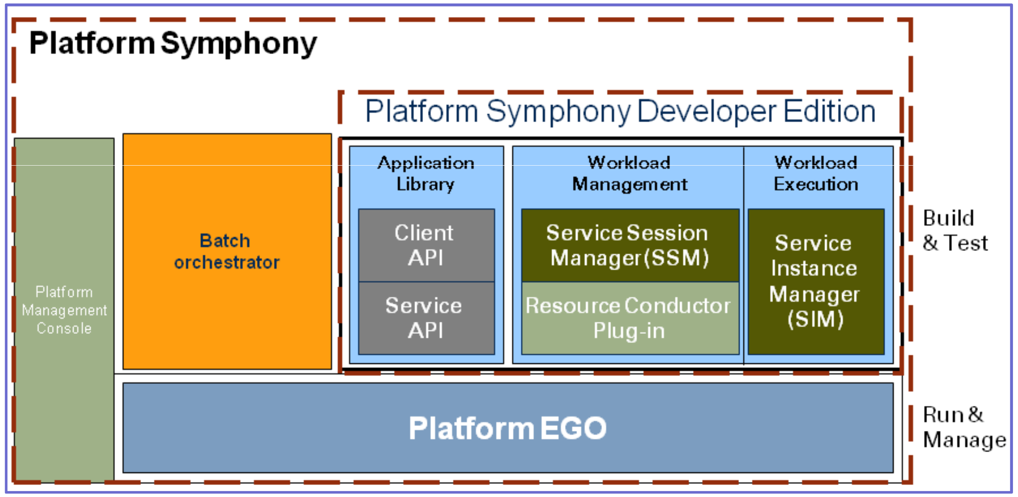
Summary

Platform

Trends in Grid Programming

Conclusion:

- Less code due to infrastructure services
- Auto-adaption of apps & infrastructure to each other
  - Advantage: overcoming the versioning problem, legacy code problem application tells infrastructure which version to provide.
  - Advanced Grid services: changing app configuration, optimizing problem decomposition
- Maintain application autonomy: run on single host must be possible
  - Example:
    Symphony Developer Edition: single host Grid
- Multi-Components applications / multi-instance (MPI, HPC-SOA) / SOA:
  - production stability depends on total availability (error propagation, see next page!)
  - Auto-response Grid infrastructure to improve total availability



Platform

Reliability → Usability → Value

Component and System availability			Comments
p-component	# component	p-total	Probability per component and total system
0,9	2	0,81000	
0,99	2	0,98010	98% good = 175 hours bad every year
0,999	2	0,99800	
0,9999	2	0,99980	
0,99999	2	0,99998	production stability: four 9's minimum
0,9	4	0,65610	
0,99	4	0,96060	96% good = 350 hours bad every year
0,999	4	0,99601	
0,9999	4	0,99960	99,96% good = 210 minutes bad every year
0,99999	4	0,99996	production stability: four 9's minimum
0,9	12	0,28243	
0,99	12	0,88638	
0,999	12	0,98807	
0,9999	12	0,99880	
0,99999	12	0,99988	will not reach production stability

🟥 probability worse than 0,9	=>	more than	876	hours disfunct per annum
🟨 probability better than 0,9	=>	less than	876	hours disfunct per annum
🟩 probability better than 0,9999	=>	less than	53	minutes disfunct per annum

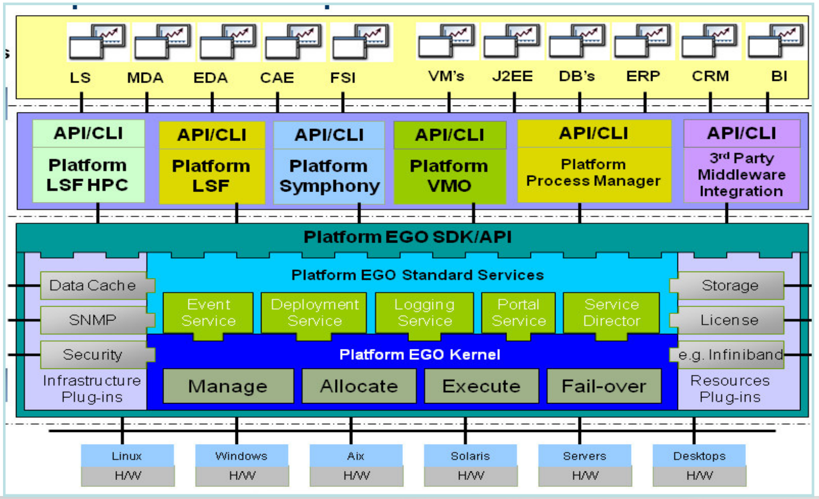
Caution:
be aware –
even 52 minutes per year could mean 52 breakdowns !!
(1 minute duration each)

Reliability → Usability → Value

- Performance Platform LSF (by 2006/07):
  - 10millions jobs per day throughput with >95% job-slot utilization based on EDA job mix, max 5min for failover. (EDA job mix: 5min, 15min, 30min job-runtime, 120Hz job submission and dispatch rate)
- Performance and Scalability translates into Reliability
- Reliability can be measured as "MTBF" -
  Mean Transactions (=Jobs) Between Failure
- Reliability is achieved by proactive incident management
  - self-healing, recovery from incidents, policy driven proactive problem containment (both for resources and applications), **no** job loss during operation or in error condition, reconfiguration or failover

Trends in Grid Programming

- Grid moves toward ease of use, plug&play app integration.
- Applications and infrastructure will describe themselves and adapt to each others requirements resp. offerings
- Infrastructure will cover up on disruptions, incidents, both on resource as application side
- First steps already done! Move on towards flexibility && reliability. Need for standards.
- Looking forward to discuss and share with you!



Bernhard Schott
Dipl. Phys.
EU-Research Program Manager
Platform Computing GmbH
Frankfurt Office

Direct	+49 (0) 69 348 123 35
Mobile	+49 (0) 171 6915 405
Email:	bschott@platform.com
Skype:	bernhard_schott
Web:	http://www.platform.com/

Platform™

Powering High Performance

Thank you

www.platform.com