



GlobalGridForum

Leading the pervasive adoption of grid computing
for research and industry

GGF Activity in Programming Models and Environments

Craig Lee and Thilo Kielmann, GGF Steering Group

GridCoord Workshop at Grids@Work

October 13, 2005



Motivations

- Much work being done on grid programming in GGF and the grid community at large
- Important to make a distinction between SOAs and APIs
- Service-Oriented Architectures (SOAs)
 - Address interoperability in loosely coupled distributed environments
 - Open Grid Services Architecture, for example
- Application Programmer Interfaces (APIs)
 - Address portability
 - Provide abstractions to the programmer to hide underlying machinery
- Both typically involve an underlying model or abstractions
 - Typical issue: can API be supported by a given SOA?
 - (But unfortunately we will not address this issue here)



What We Will Address

- Review much current work in GGF with regards to APIs, programming models, and related topics
 - Including OGSA, GFS, DAIS, DRMAA, JSDL, GridCPR, GridRPC, SAGA
 - (Acronyms will be expanded later)
 - More information available at:
http://www.gridforum.org/ggf_areasgrps_overview.htm
- While this is very useful and interesting, we will also examine future directions
 - Possible programming tools and models to provide new capabilities and improve performance



Open Grid Service Architecture

- OGSA is a web-services based architecture providing basic services to manage distributed applications and systems
 - Execution management services
 - Data services
 - Security services
 - Resource Management Services
 - Self-management services
 - Information services
- OGSA is a “brand” that covers many current GGF activities
 - OGSA Basic Execution Services
 - OGSA Resource Selection Services
 - OGSA-DAI (Data Access)
 - OGSA Data
 - OGSA Data Replication Services
 - OGSA Authorization
 - OGSA P2P Security
 - OGSA Naming
- But...Web Services are not Distributed Objects
 - No inheritance, no public/private members, no state
- *Many programming models and properties remain to be/need to be built on this platform*

Grid File System WG

- Location-, Infrastructure-, Organization-independent name space
- Independent of physical data access, transport, authentication mechanisms

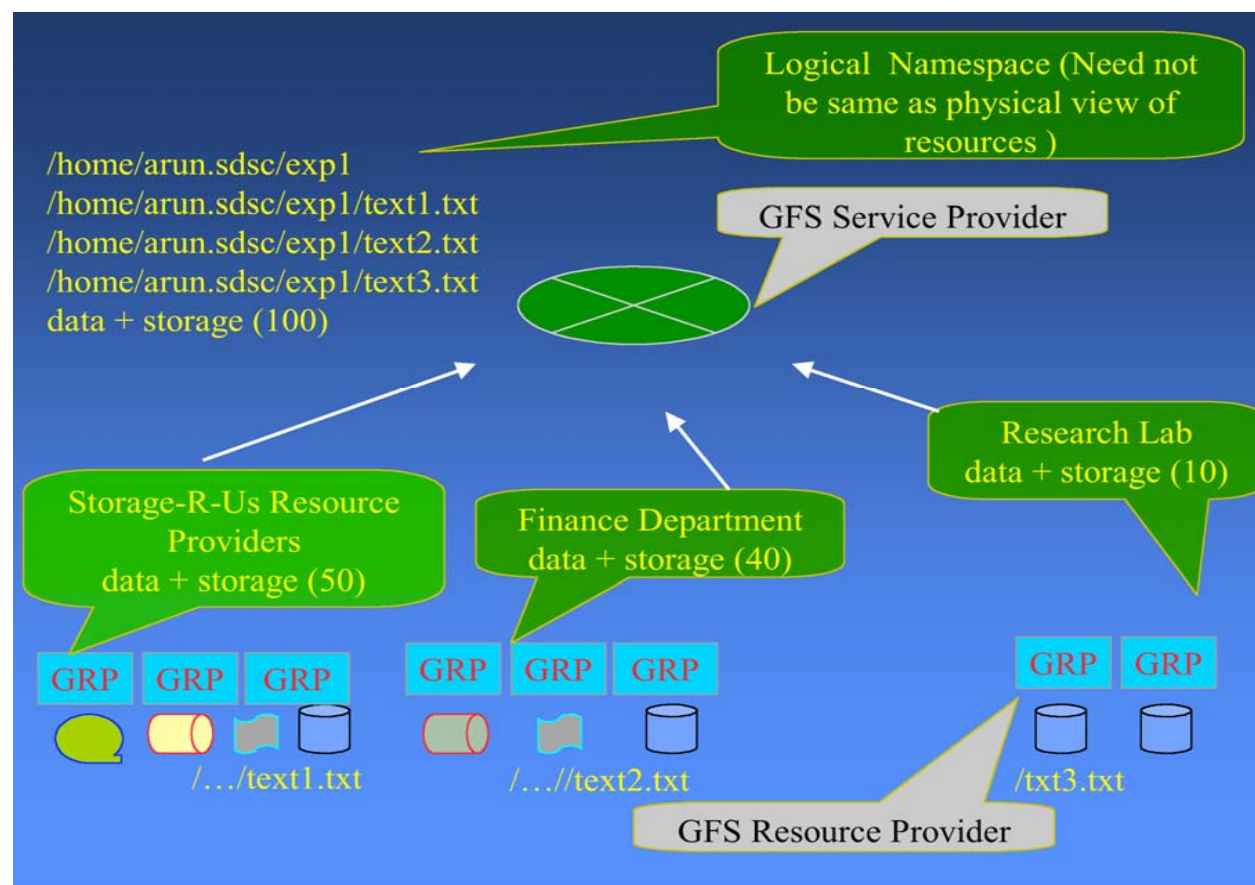
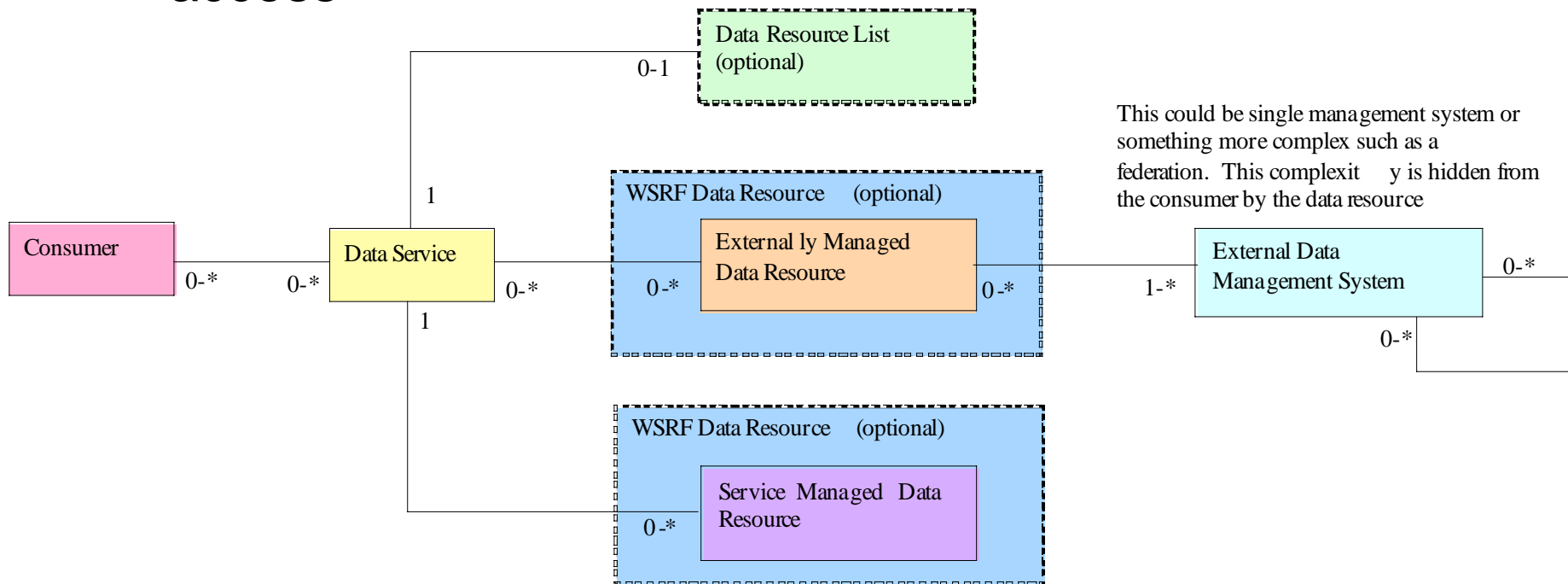


Figure 1: Architecture of Grid File System

DAIS WG (Database Access and Integration Services)



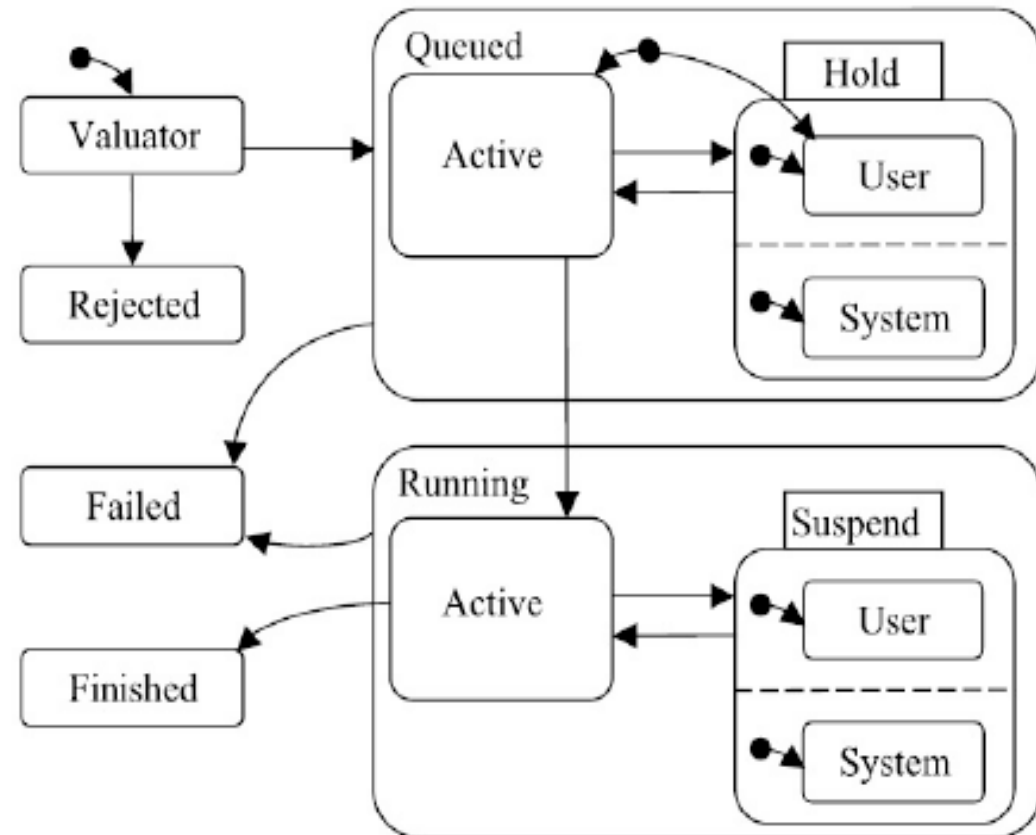
- Focuses on defining base data services
 - Externally managed data resource, e.g., relational DB
 - Service managed data resource, e.g., a data product preserved by DAIS implementation for subsequent access



DRMAA (Distributed Resource Management Application API)

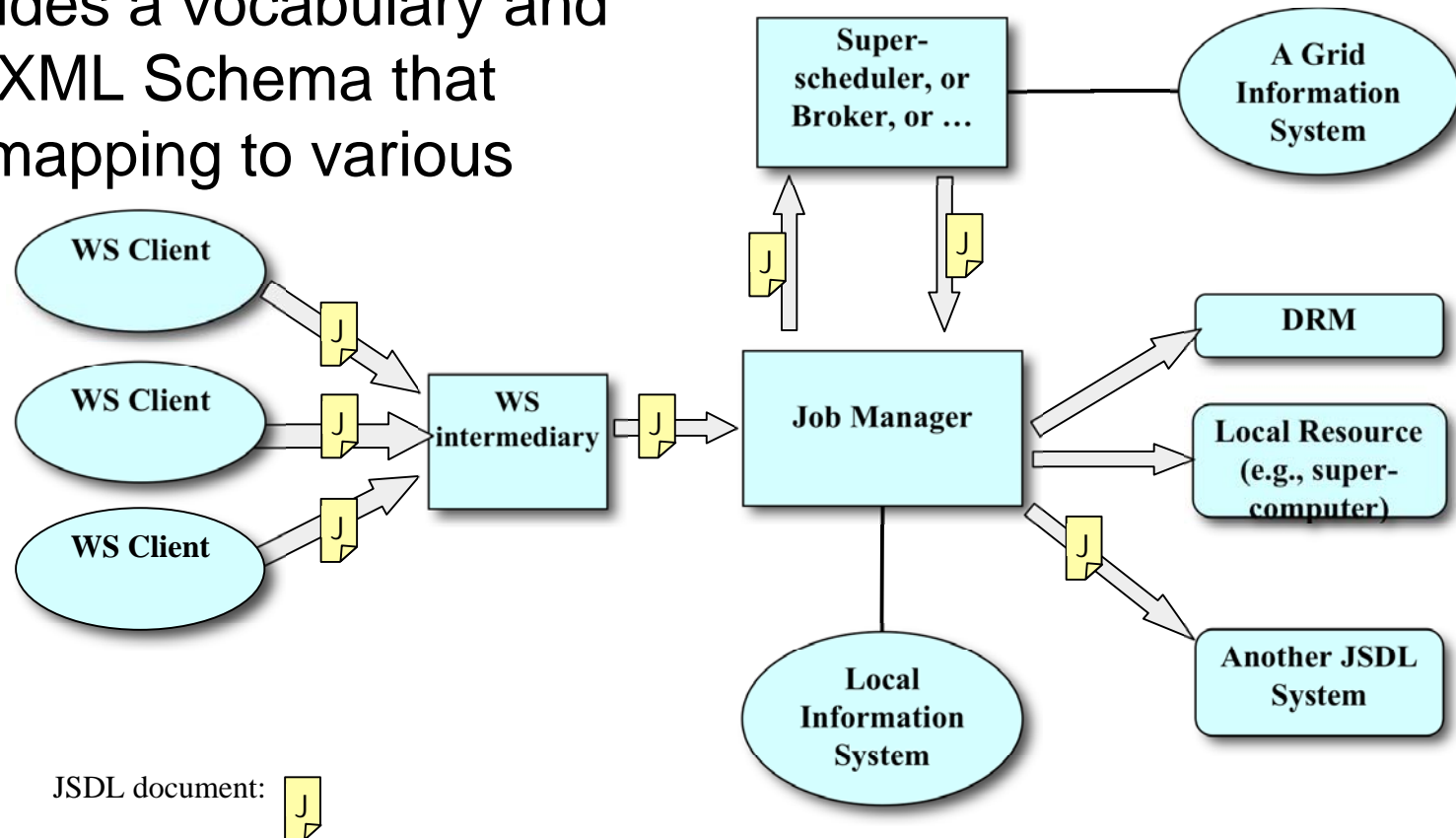


- Job Submission API
- Job Template describes all job parameters when job submitted
 - Valuator validates parameters before actual submission
- Supports single jobs, bulk jobs
- Job progresses according to *State Transition Diagram*
 - User/System Hold/Suspend available
- Typically job control operations on *job_id*



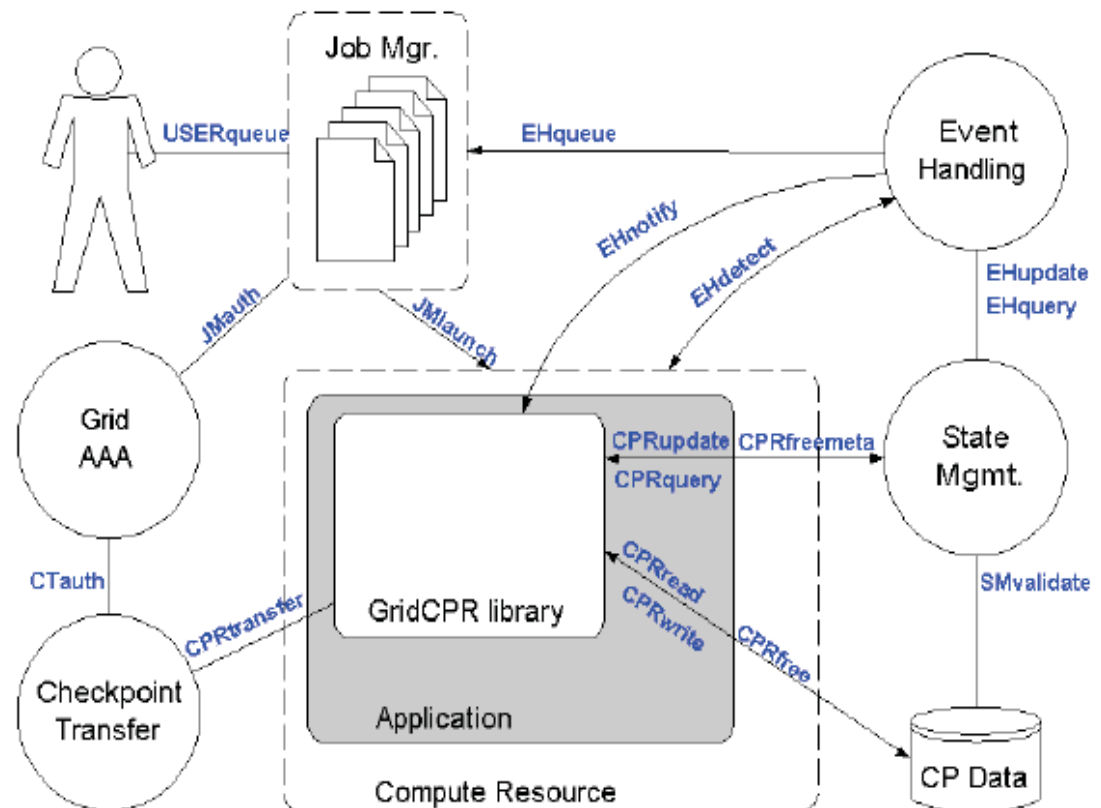
Job Submission Description Language

- Grid jobs may be submitted to a variety of job management systems
- JSDL provides a vocabulary and normative XML Schema that facilitates mapping to various systems



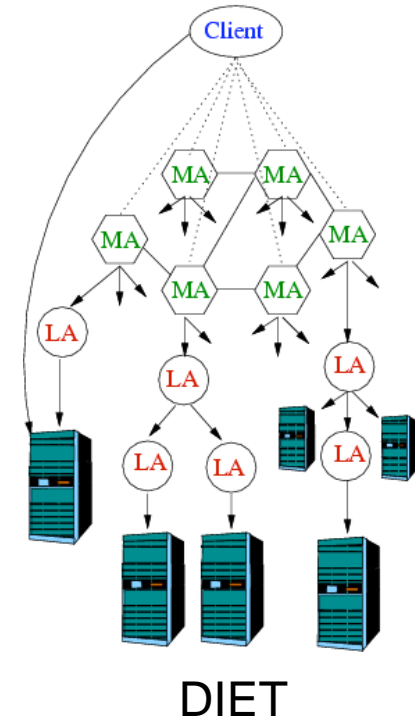
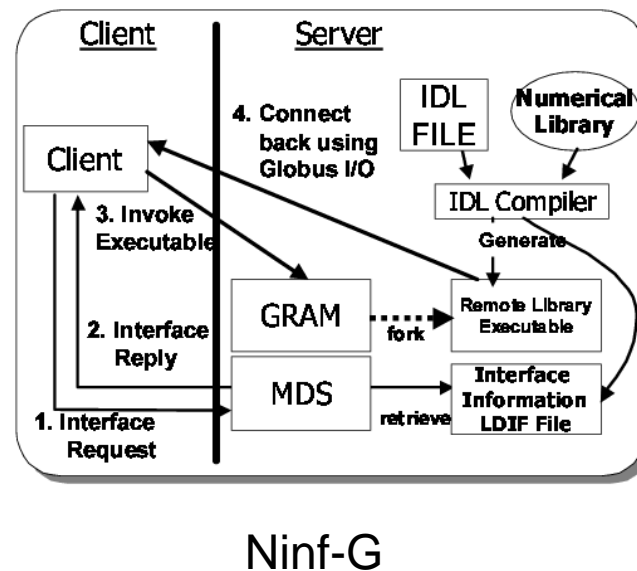
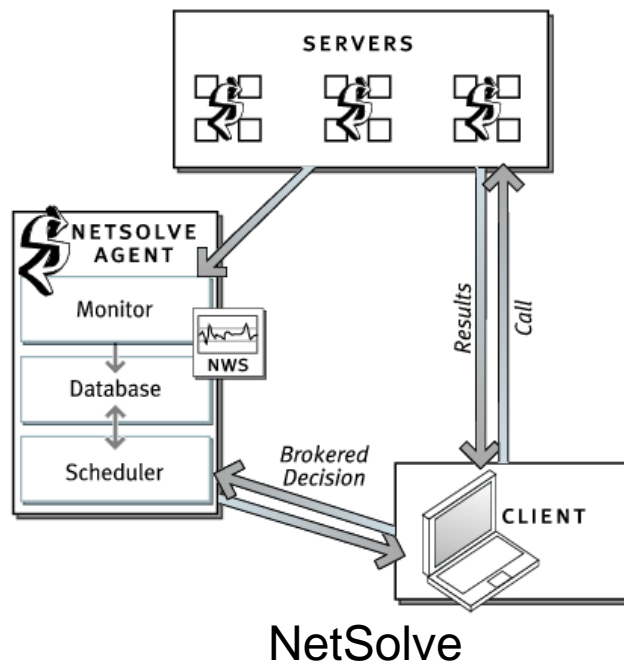
GridCPR (CheckPoint and Recovery)

- A checkpoint and recovery system that provides
 - Services for State Management, Event Handling, Checkpoint Transfer
 - GridCPR Resources for managing job snapshots and related information
 - GridCPR library provides API for reading/writing application state, failure/event notification, checkpoint transport, etc.



GridRPC (Remote Procedure Call)

- Simple, portable API for Remote Procedure Call
 - Significant users for network-enabled server models: NetSolve, Ninf-G, DIET
- *Function Handles* used to manage call instances
 - Blocking and non-blocking function calls
 - Asynchronous control/wait functions
- *Data Handles* being explored to manage RPC argument movement
 - Data Handles can be *bound* to location, read, written





SAGA (Simple API for Grid Applications)

- Core Subsystem
 - Session – manage interactions with various grid services
 - Context – security related info – can be attached to session handle
 - Task Model – used to manage asynchronous ops
 - Attributes
- Data Subsystem
 - NameSpaces – Logical Files
 - Files – Adverts – proposed
- Resource Subsystem
 - Jobs – GridCPR – proposed
- Communication Subsystem
 - Streams – Messages - proposed – GridRPC – proposed
- *SAGA may be considered as an API complimentary to OGSA*



Future Directions

- Much work is being done on basic tools for grid application programming
- BUT... A number of fundamental issues are outstanding
- Grid environments are inherently loosely coupled
 - *How can we enable applications to be as tightly coupled as possible?*
- Grid environments are inherently fault-prone
 - *How can we make them more fault-tolerant?*
- In large distributed environments, it will not be feasible to have perfect information about the entire system at all times
 - *How can we manage applications that will have imperfect knowledge of their environment and limited control over it?*

Exploiting Computational Density

Computational density varies over space and time

Can computational resources be dynamically allocated based on computational density and reduce comm and sync requirements?

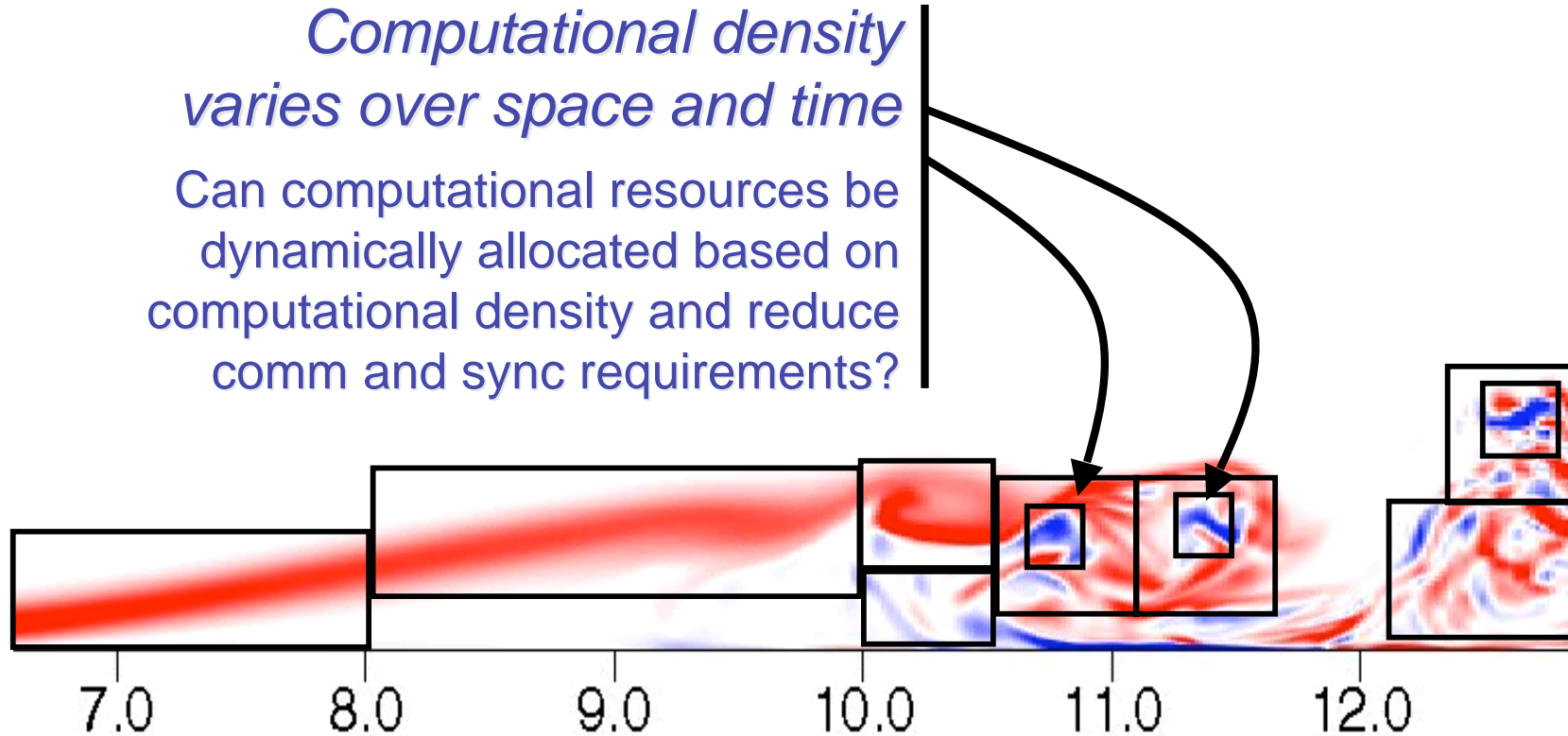
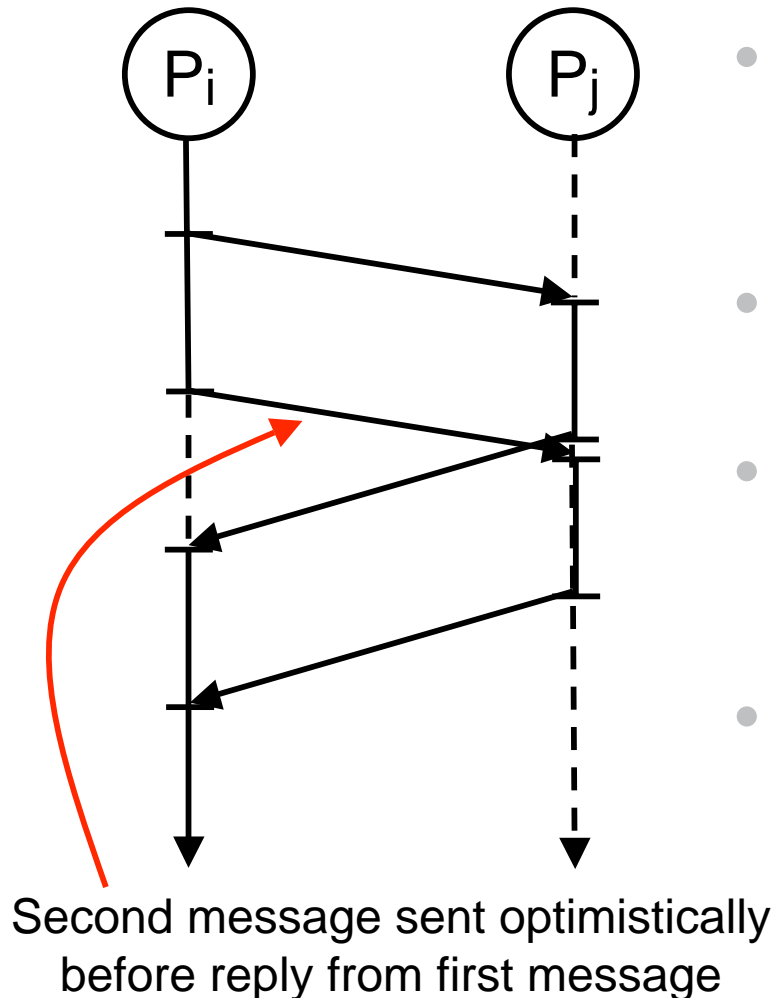


Image: Low pressure turbine, Fasel & Seidel, U. Arizona

- Can we re-design or re-conceive computational problems to more closely match the grid infrastructures that will be available?
- Can we design grids to provide better support to computational problems?

Loosening Synchronization with Optimistic Techniques

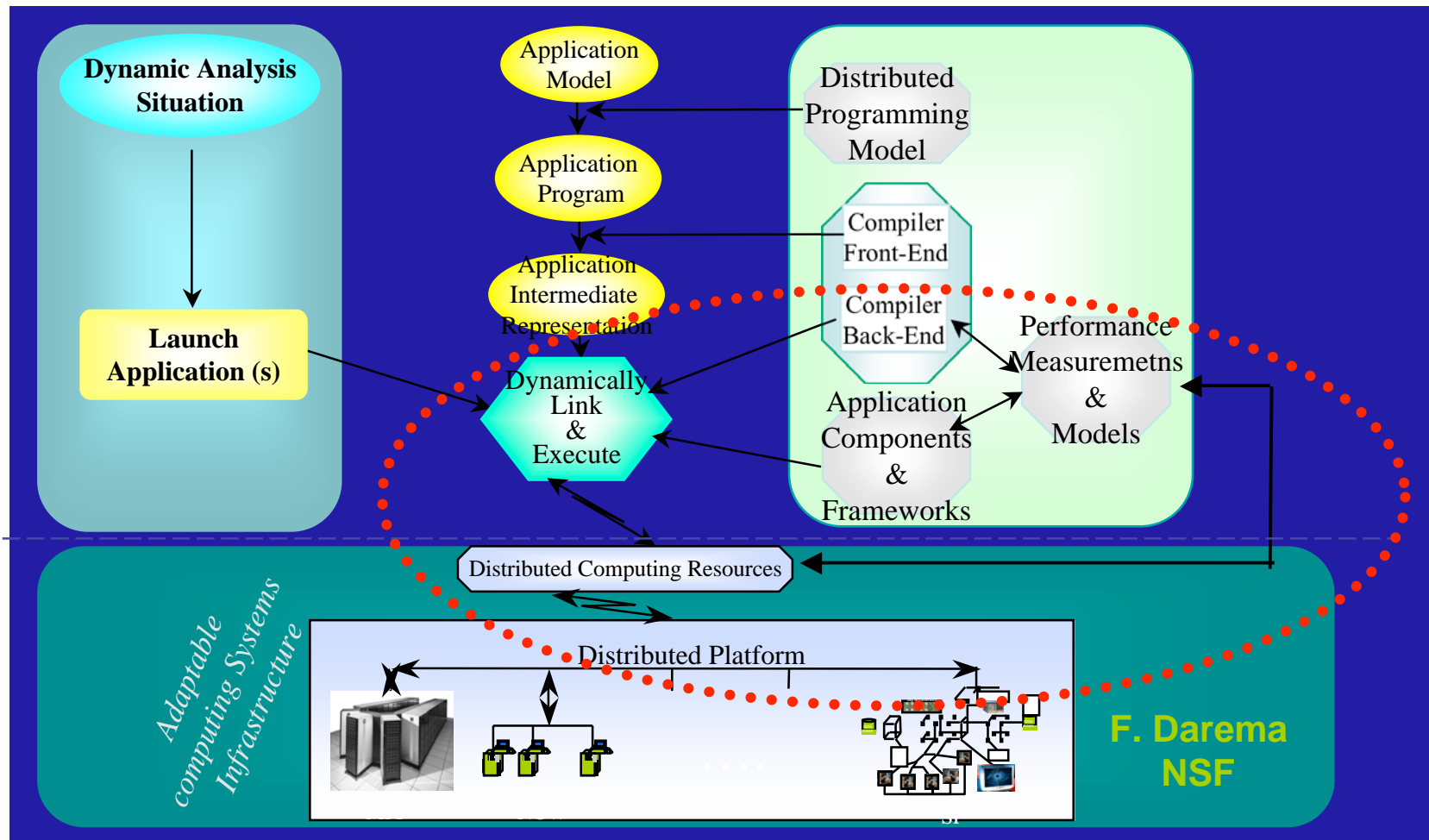


- Optimistic (speculative) execution allows latency to be hidden
 - At the cost of potential rollbacks
- Application must control probability and cost of rollbacks
- Incremental state saving typically used to manage “backing out the state”
- Similar to *Transaction Processing*
 - Transaction committed when correctness established
 - Transaction abandoned when not

Runtime Compiling System (RCS) and Dynamic Application Composition



- “Smart Run-time” integrates feedback and control
- Known services cached based on stable availability, usage patterns



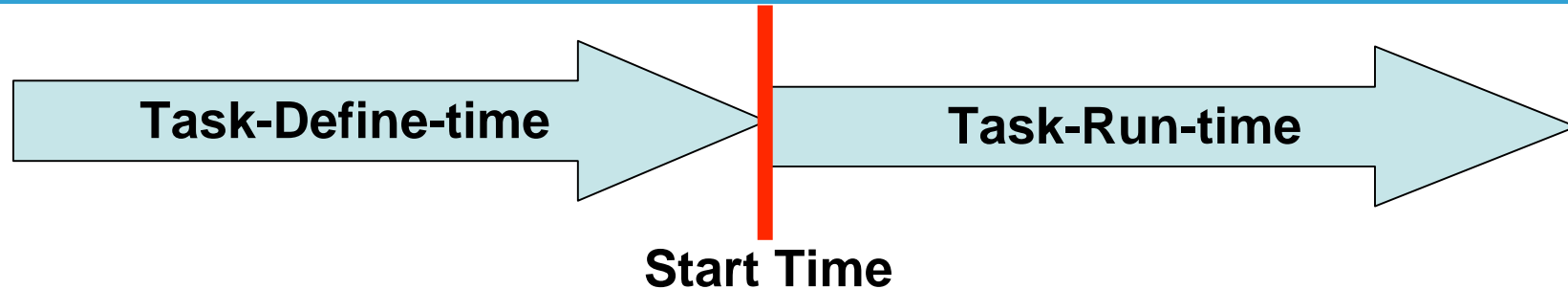


Dynamic Event-Driven Workflows

- System events may require an existing workflow be altered
 - Current amount of workflow completed must be determined
 - Current tasks on the “leading edge” of the workflow must be terminated or allowed to complete
 - Status and disposition of data referenced by tasks must be determined
 - “Classical” storage management issues reoccur
 - Dangling references to no data or stale data
 - Unaccessible data referenced by no one
- Such event-driven task mgmt is similar to fault tolerance
 - Similar mechanisms could be used to detect and respond to faults (failed servers, networks, etc.)
- *Workflow engines may be centralized or decentralized*

Use of *A Priori* Information in Grids

Knowable independently of experience



- Expects the world to have certain properties or be in a known state
- Everything that can be statically defined *a priori* takes complexity out of the application and improves performance
- Increasing use of *a posteriori* information learned from experience
- “Smart” run-time
- Limited control and imperfect knowledge of the environment



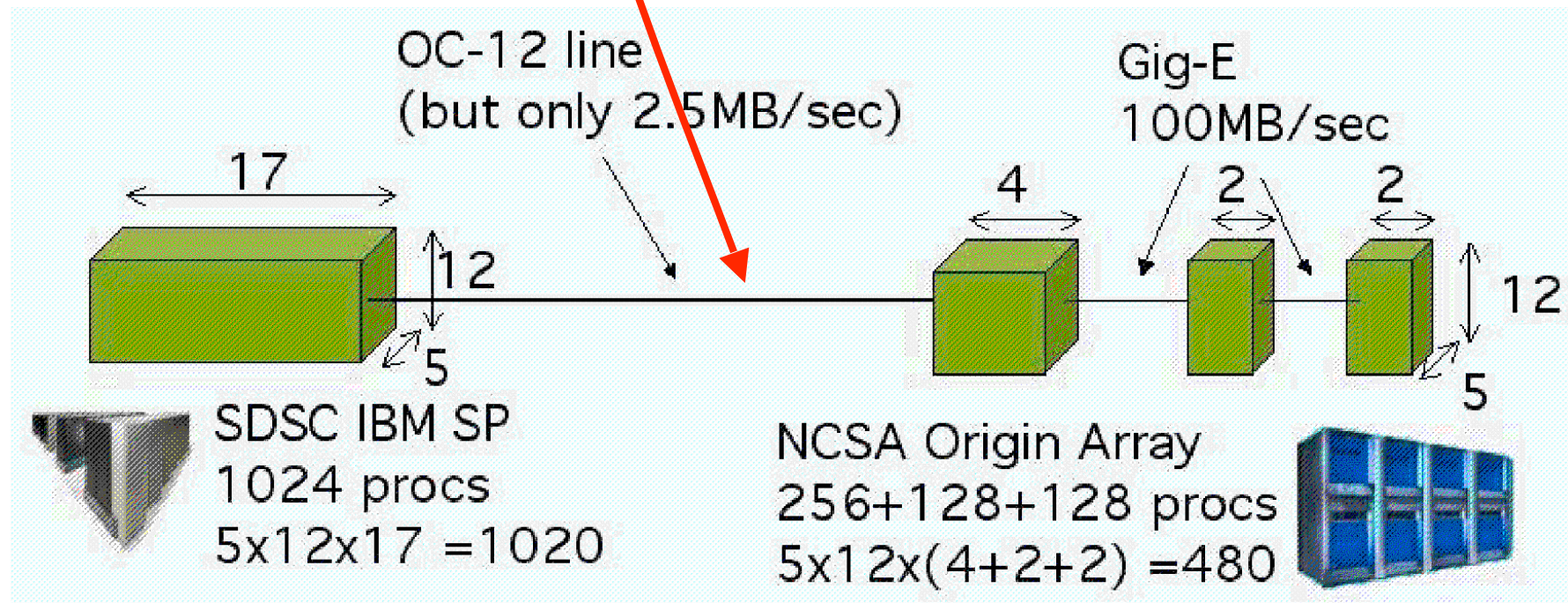
Summary

- Much work is being done in GGF on the “basic building blocks” for grid programming, BUT...
- Fundamental nature of grid environments will push us to
 - Reconceive/restructure application models to tolerate bandwidth, latencies, and unreliability
 - Reconsider other programming models
 - Optimistic/Speculative/Transactional
 - Enable codes to be more loosely coupled
 - Performance penalty for “wrong” executions
 - Declarative programming techniques
 - Program the “What”, not the “How”
 - Performance penalty for deciding everything dynamically
 - Agents and Artificial Intelligence
 - Intelligent agents that can inference on known and discoverable facts in their environment to plan (and replan) tasks to reach a goal state
 - AI is hard! Much effort over many years
- Much work remains to be done!

Many Traditional Techniques Can Be Used to Tolerate Latency

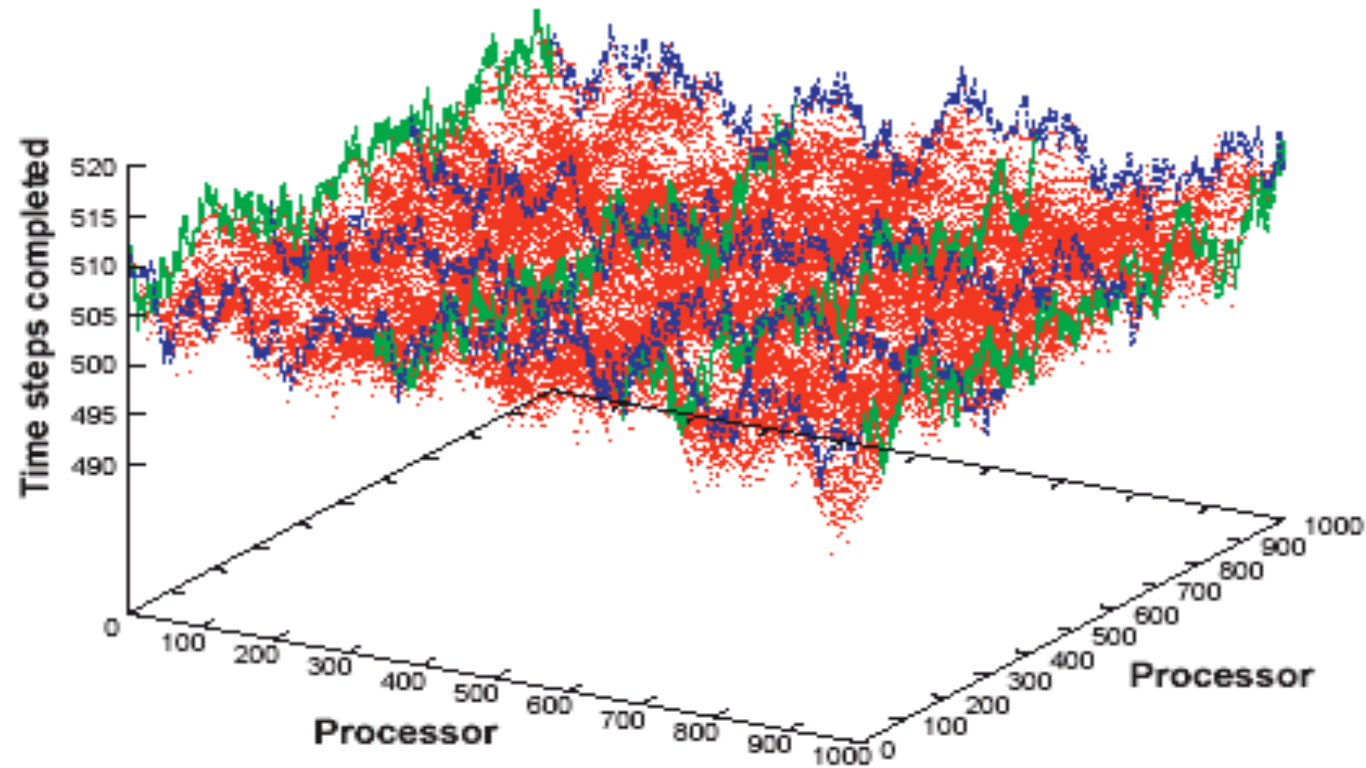


- Compression
- Pre-fetching
- Grid-aware Comm Scheds
- Ghost Zones
- TCP Tuning



G. Allen, et al. SC01

Loosening Synchronization



Out of sync. This simulated temporal surface was created by allowing 1000 x 1000 processing elements to proceed in equal time steps under local synchronization.

Scott Kirkpatrick, *Science*, vol. 299, p.668-669, 31 Jan 03 "COMPUTER SCIENCE: Rough Times Ahead"



How to Program Decentralized Workflows?

- “Process programming” in a distributed environment
 - Agent Coordination Languages
 - Example: Little-JIL
 - A coordination tree with four non-leaf operations
 - sequential, parallel, try, choice
- Other possibilities?
 - Stream-based languages?
 - Dataflow languages?
- Decentralized Workflows similar to Active Messages
 - “Programming the message, not the node”
- *There is motivation to revisit these programming paradigms in grid environments*

The Impact of Programming with Web/Grid Services



- Document transfer-based services provide great flexibility
 - Very late binding of service name to service provider
 - Dynamic WSDL interpretation avoids having to “compile-in” stubs or proxy classes
- Compilers can statically enforce strong type hierarchies
- Service documents used for essentially the same function
 - Dynamically
 - Incrementally
- How can “semantic analysis” tools, e.g. compilers, be used when much of the “code” and the environment itself is decided at run-time?