Diseño de Sistemas Distribuidos

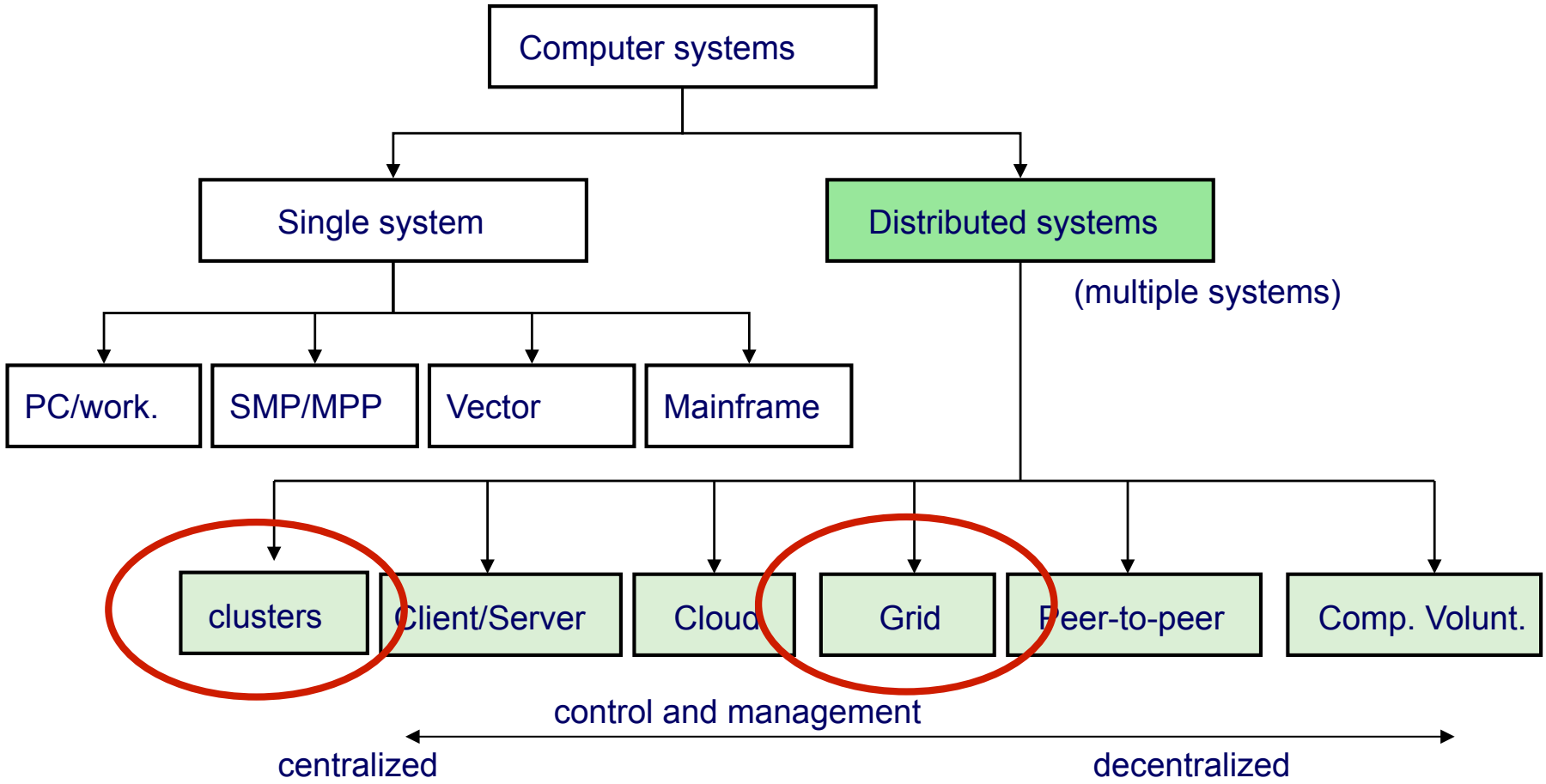
Máster en Ciencia y Tecnología Informática

Curso 2014-2015

Cluster y grid computing

Félix García Carballeira
Grupo de Arquitectura de Computadores
felix.garcia@uc3m.es

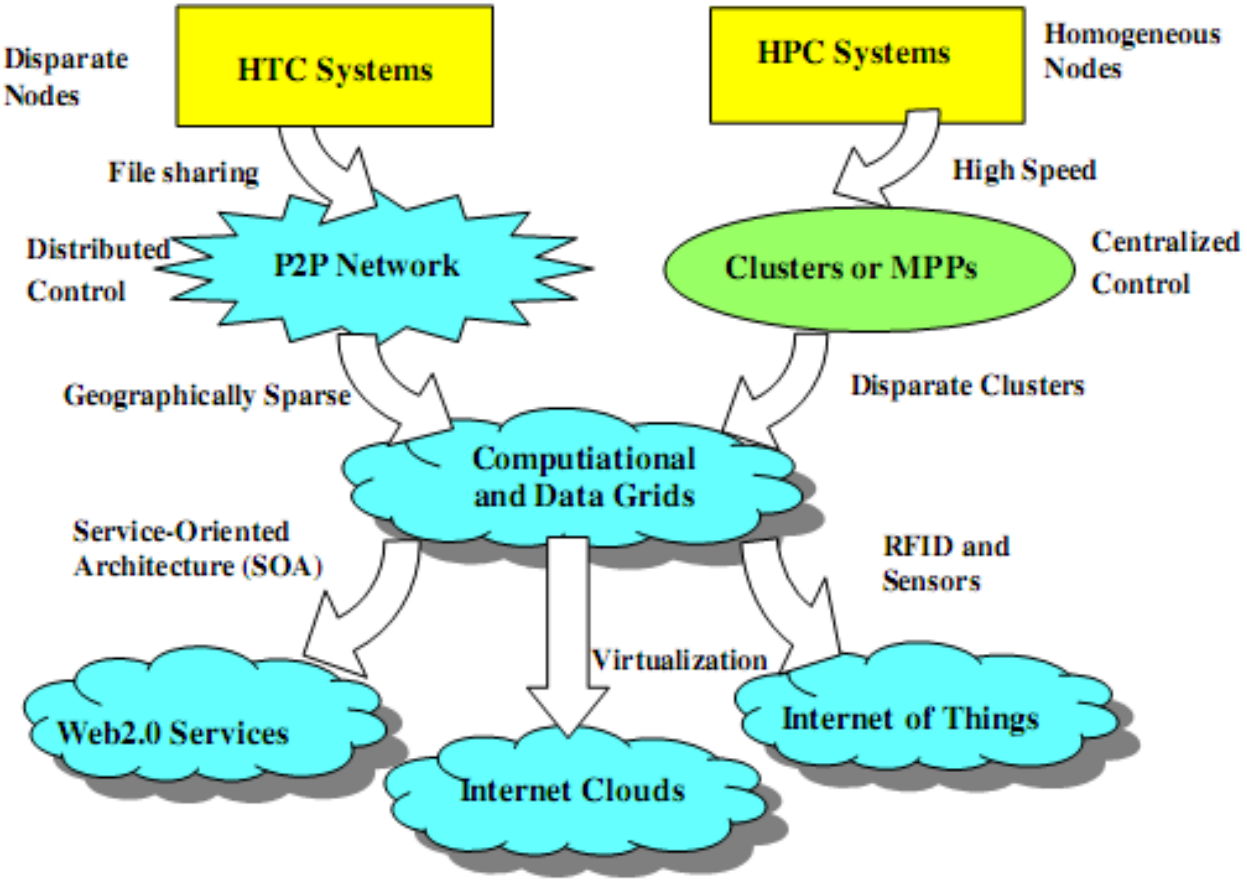
Computing systems



Características de los sistemas distribuidos de gran escala

- Gran distribución geográfica
- Gran capacidad
  - Gran número de usuarios/clientes
  - De almacenamiento
  - Muchas conexiones de red
  - De procesamiento
  - Alto nivel de paralelismo
  - Capacidad para tolerancia a fallos

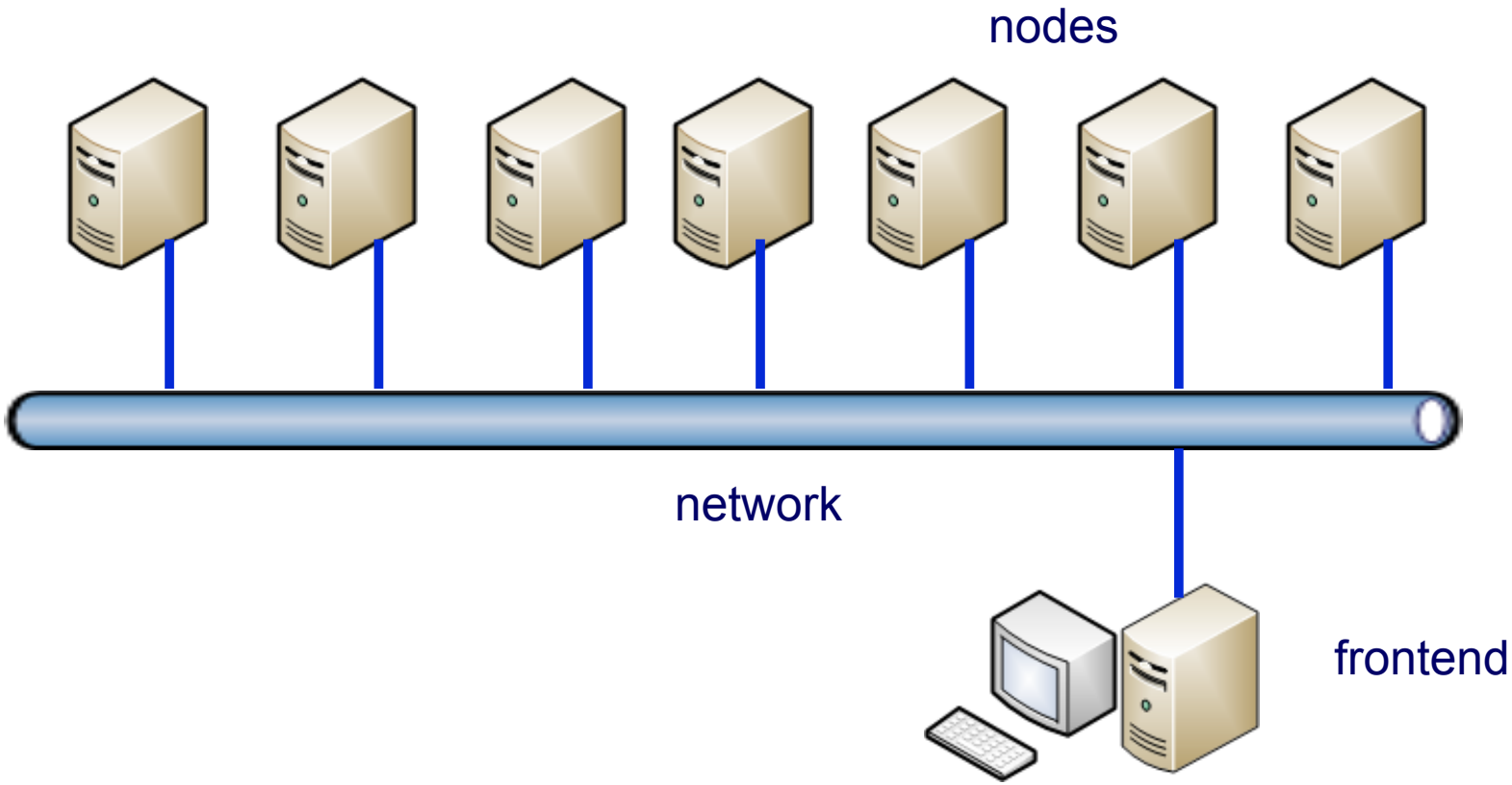
Evolución de HTC y HPC



Source: K. Hwang, G. Fox, and J. Dongarra,
Distributed and Cloud Computing,
Morgan Kaufmann, 2012.

Cluster

- Arquitectura distribuida formada por un conjunto de computadores independientes interconectados que funciona como un único sistema (*single system image*)



Típico cluster

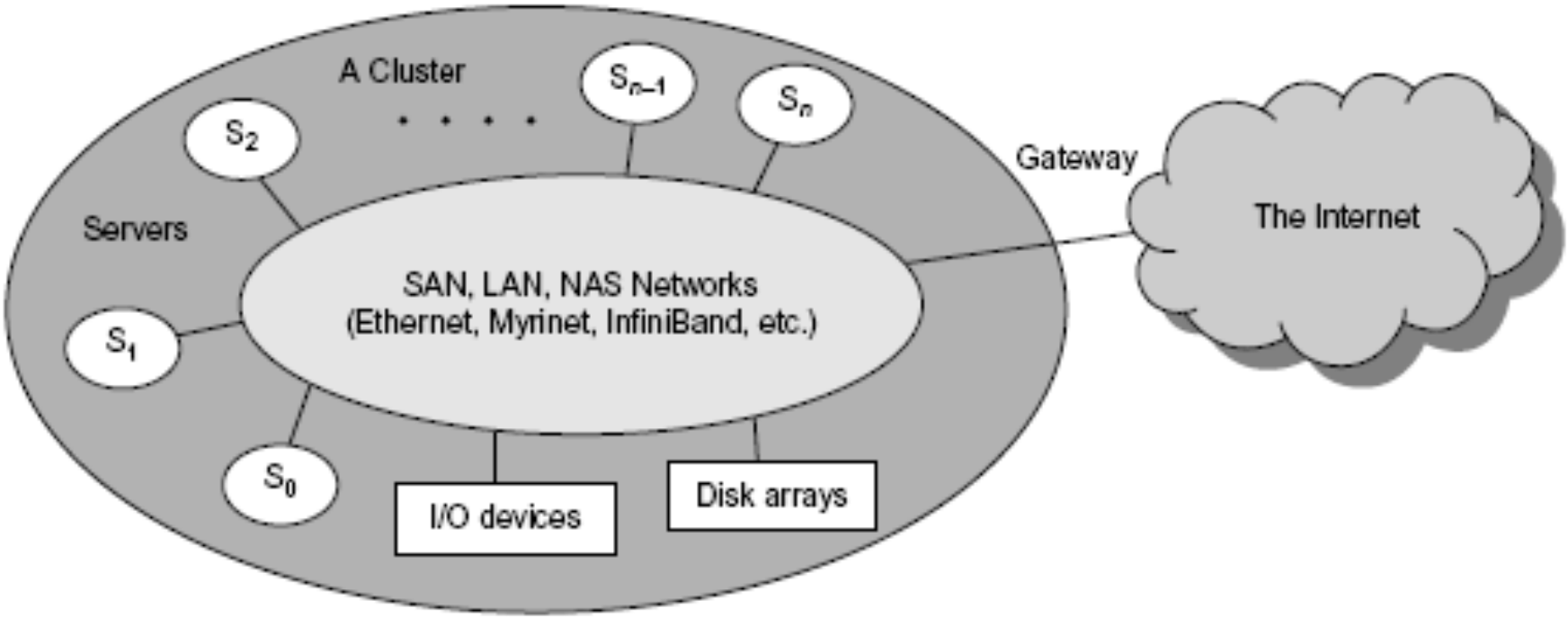
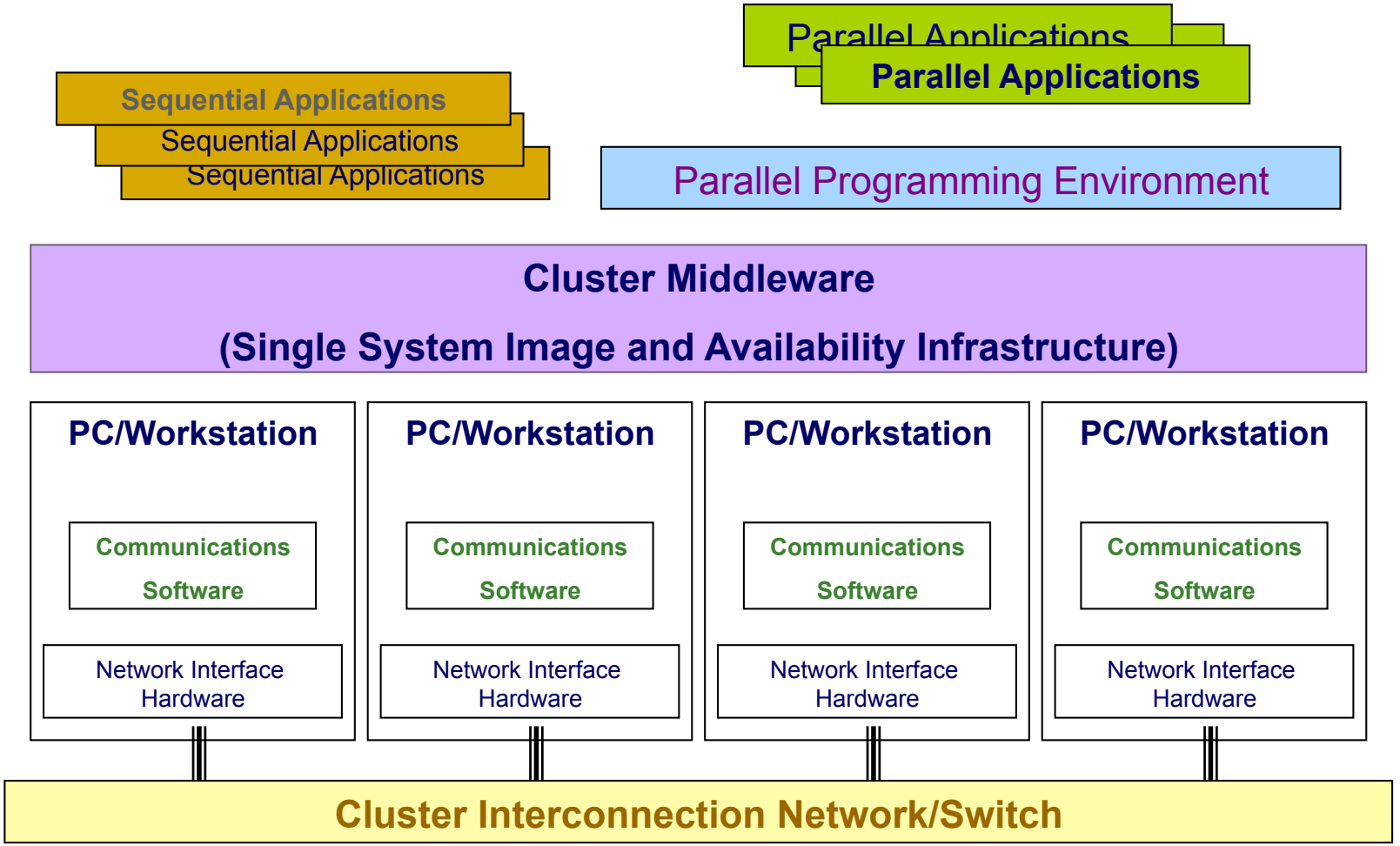


FIGURE 1.15

A cluster of servers interconnected by a high-bandwidth SAN or LAN with shared I/O devices and disk arrays; the cluster acts as a single computer attached to the Internet.

Source: K. Hwang, G. Fox, and J. Dongarra, *Distributed and Cloud Computing,* Morgan Kaufmann, 2012.

Arquitectura de un cluster

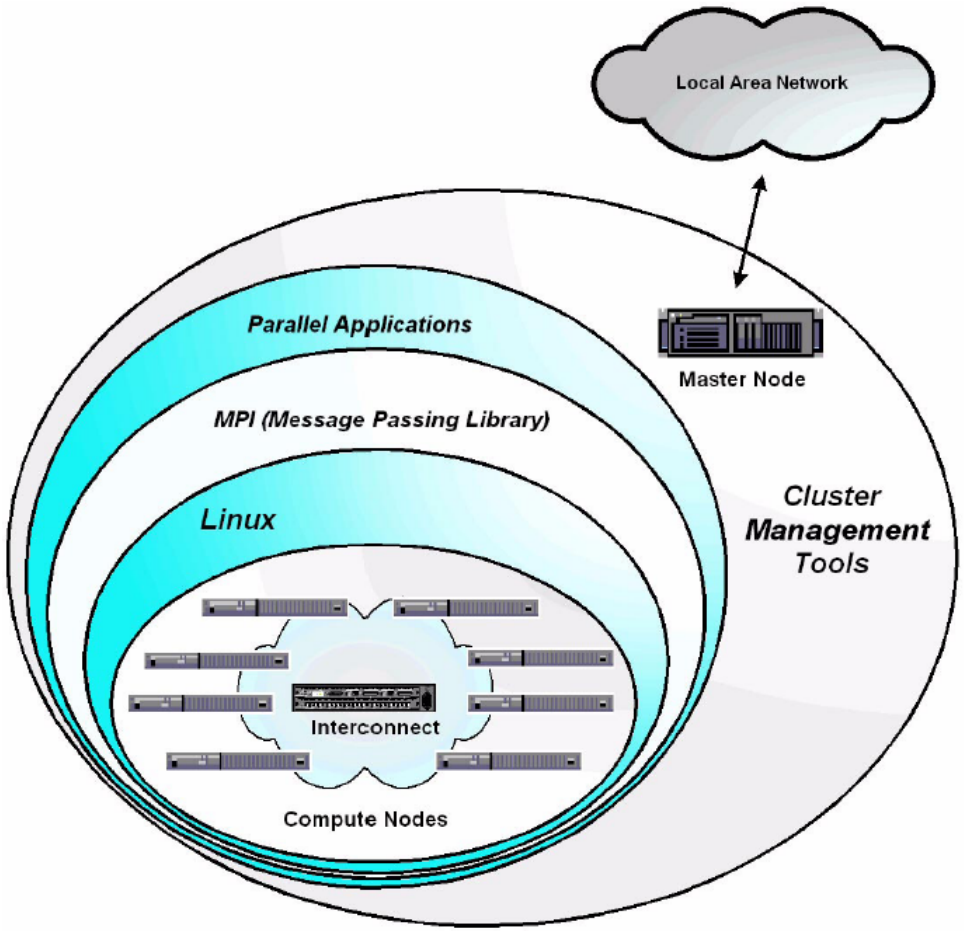


Buyya

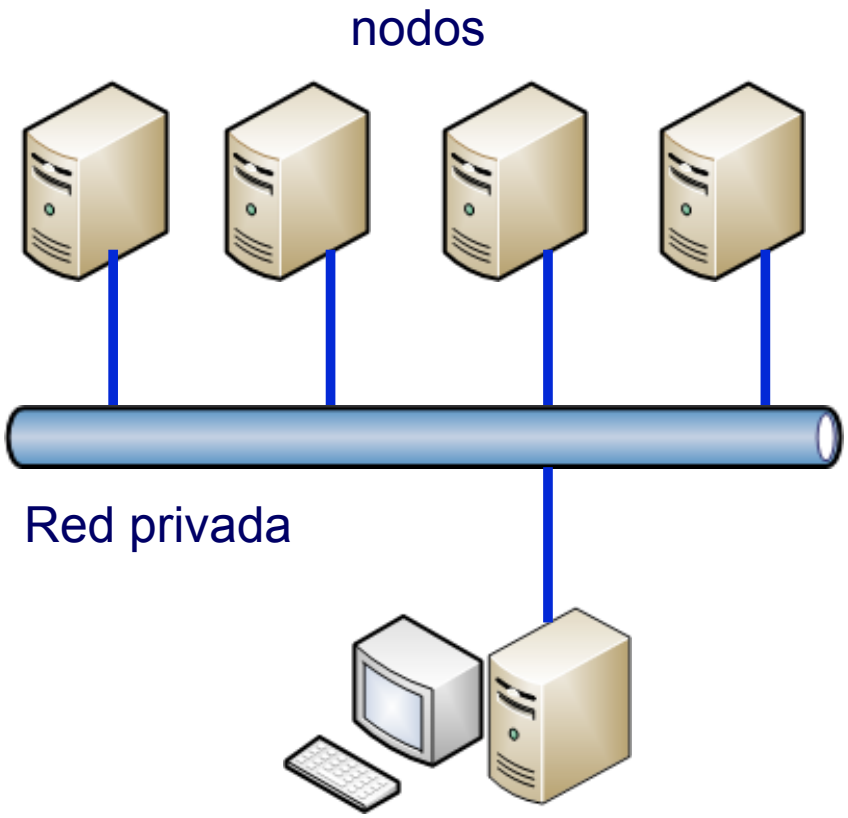
Cluster Beowulf

- Primer cluster construido con computadores personales 1994:
  - 16 DX4 procesadores conectados mediante *channel bonded* Ethernet. NASA Ames Centre
- Nodos con mínimos recursos (sin monitor, ratón, tarjeta de vídeo)
- Un nodo servidor (*frontend*)
- Linux
- Sistema de ficheros compartido
- Herramientas de programación paralela (PVM, MPI)
- Sistemas de gestión de trabajos
  - Batch, interactivos , secuenciales, paralelos

Logical vision of a Beowulf cluster



Arquitectura de cluster Beowulf



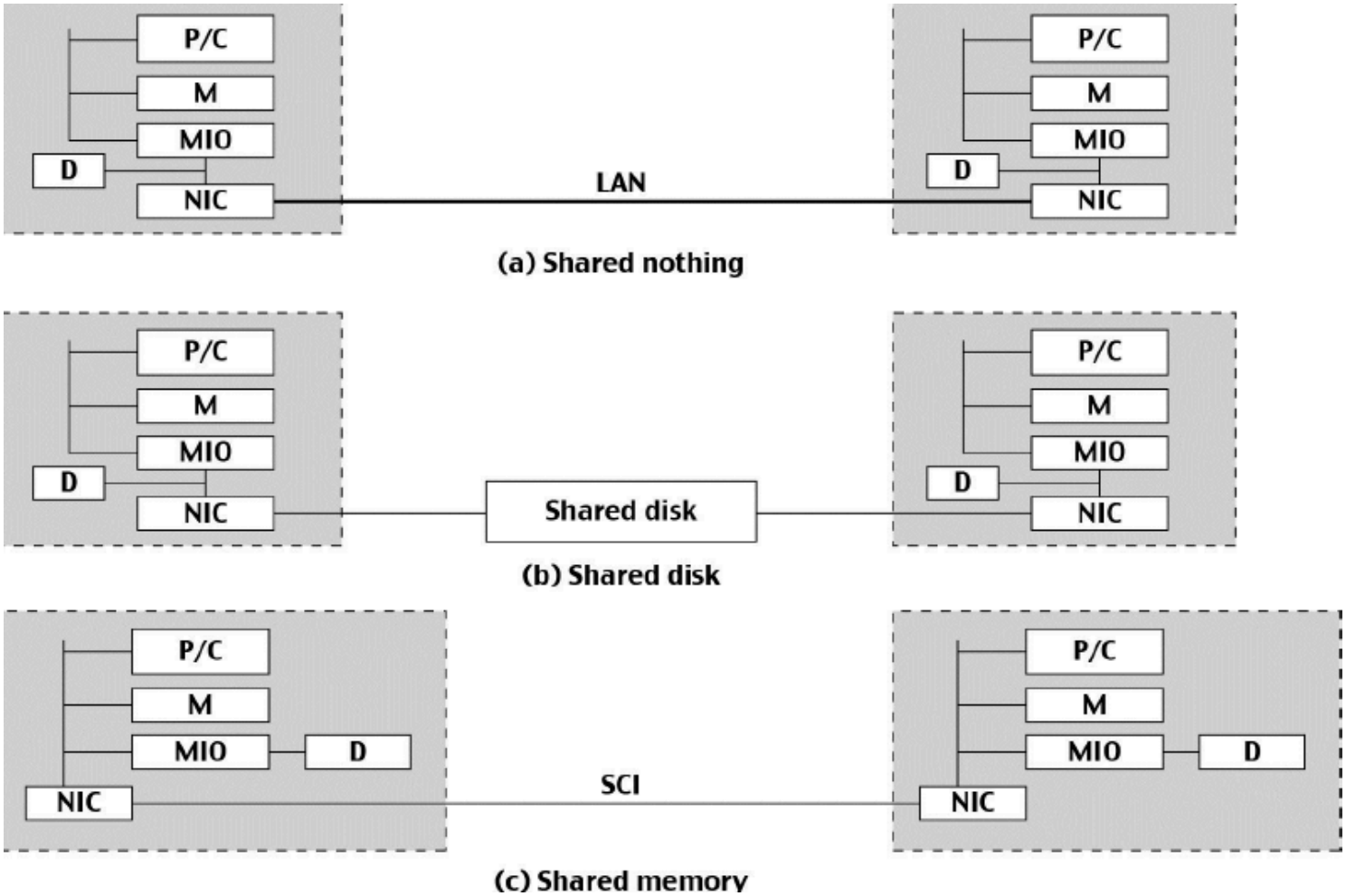
Beneficios que ofrece un cluster

- Disponibilidad: por la redundancia del
  - Hardware
  - Sistema operativo y aplicaciones
- Escalabilidad: mediante la incorporación de nuevos servidores o más clusters a la red
- Alto rendimiento
- Alta productividad

Clasificación de los clusters

- Clusters de altas prestaciones (HPC)
  - Aplicaciones paralelas
- Clusters de alto rendimiento (HTC)
  - Gran número de tareas independientes
- Clusters de alta disponibilidad (HA)
  - Aplicaciones críticas
- Clusters para equilibrio de carga
  - Servidores web, de correo, servidores en general
- Clusters híbridos
  - HPC + HA
- Según la dedicación
  - Dedicados, no dedicados
- Según la configuración:
  - Homogéneos y heterogéneos

Compartición de recursos en un cluster



Source: K. Hwang, G. Fox, and J. Dongarra, *Distributed and Cloud Computing,* Morgan Kaufmann, 2012.

Problemas de diseño de un cluster

- Escalabilidad: física y de aplicaciones
- Disponibilidad: gestión de fallos
- *Single system image*: middleware y extensiones del SO
- Equilibrio de carga (CPU, red, memoria y discos)
- Seguridad y crifado
- Entorno distribuido (amigable y fácil de usar)
- Facilidad de programación (API)
- Facilidad de gestión (trabajos y recursos)

Sistemas de gestión de recursos

- **LSF**
  - http://www.platform.com/
- **SGE**
  - http://www.sun.com/grid/
- **NQE**
  - http://www.cray.com/
- **LL**
  - http://www.ibm.com/systems/clusters/software/loadleveler/
- **PBS**
  - http://www.pbsgridworks.com/

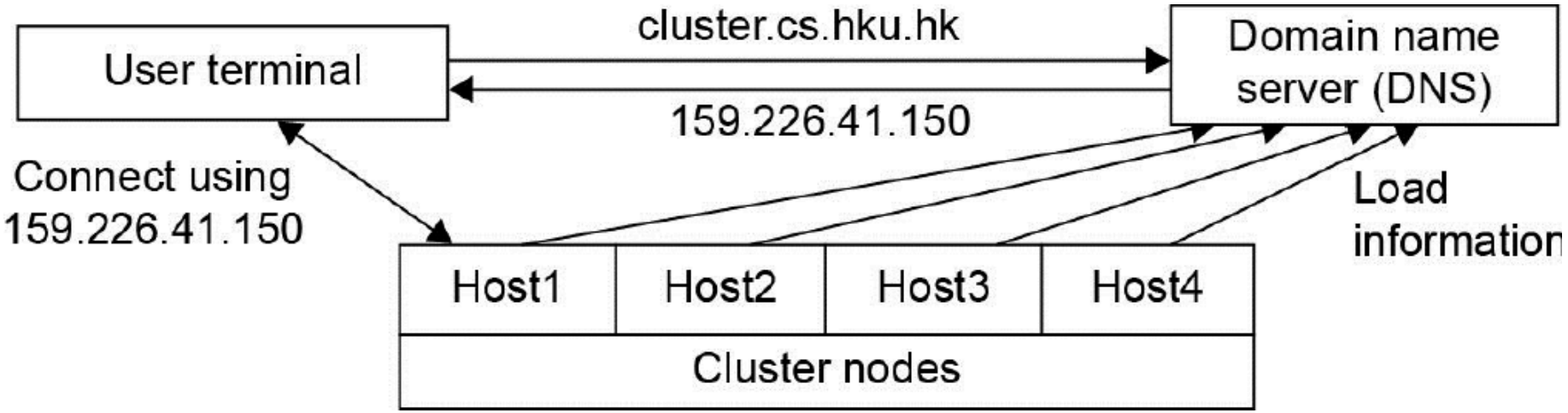
Cluster middlware

- Reside entre el SO y las aplicaciones. Ofrece:
  - *Single system image* (SSI)
  - *System Availability* (SA)
- SSI se encarga de que todos los recursos del cluster aparezacan globalmente como una única máquina
- Checkpointing y migración de procesos

Servicios deseados en un sistema SSI

- Punto de acceso único
- Jerarquía de ficheros única
- Gestión unificada de trabajos
- Interfaz de usuario única
- Rede virtual única sobre varias redes físicas
- Mismo espacio de memoria

Ejemplo de punto de acceso único



Source: K. Hwang, G. Fox, and J. Dongarra, *Distributed and Cloud Computing,* Morgan Kaufmann, 2012.

Tipos de trabajos a ejecutar en un cluster

- Trabajos secuenciales: ejecutan en un único nodo
- Trabajos paralelos: ejecutan en múltiples nodos
- Trabajos interactivos: ejecución asociada a un terminal, ejecución inmediata
- Trabajos *batch*:
  - No necesitan respuesta inmediata
  - Se envían a una cola para su planificación

Características de la carga de trabajo de un cluster

- La mitad de los trabajos paralelos se envían durante el horario de trabajo normal
- Casi el 80% de los trabajos ejecutan menos de 3 minutos
- Los trabajos paralelos que tardan más de 90 minutos contabilizan el 50% del tiempo total
- Entre el 60% y el 50% de los nodos están disponibles para ejecutar trabajos paralelos en cualquier momento
- En un nodo, el 53% de los periodos de inactividad duran en media menos de 3 minutos, pero el 95% del tiempo de inactividad es gastado en periodos de más de 10 minutos

Tipos de planificación

- Planificación dedicada:
  - Solo un trabajo es ejecutado en el cluster a la vez y solo un proceso del trabajo es asignado a cada nodo
  - El trabajo ejecuta hasta su terminación
- Planificación compartida en el espacio
  - Diferentes trabajos se ejecutan en particiones de nodos disjuntas de forma simultánea
  - Un proceso por nodo
  - El sistema de E/S puede ser compartido

Tipos de planificación

- Tiempo compartido
  - Varios procesos se asignan al mismo nodo
  - Tipos de planificación:
    - Planificación independiente: utiliza el sistema operativo de cada nodo para planificar los procesos como una workstation tradicional
    - Planificación Gang: planifica todos los procesos de un trabajo paralelo juntos. Cuando un proceso está activo, todos están activos
    - Competición con trabajos locales: los trabajos enviados al cluster cuando se planifican en un nodo compiten con los trabajos locales (más prioritarios)

MOSIX

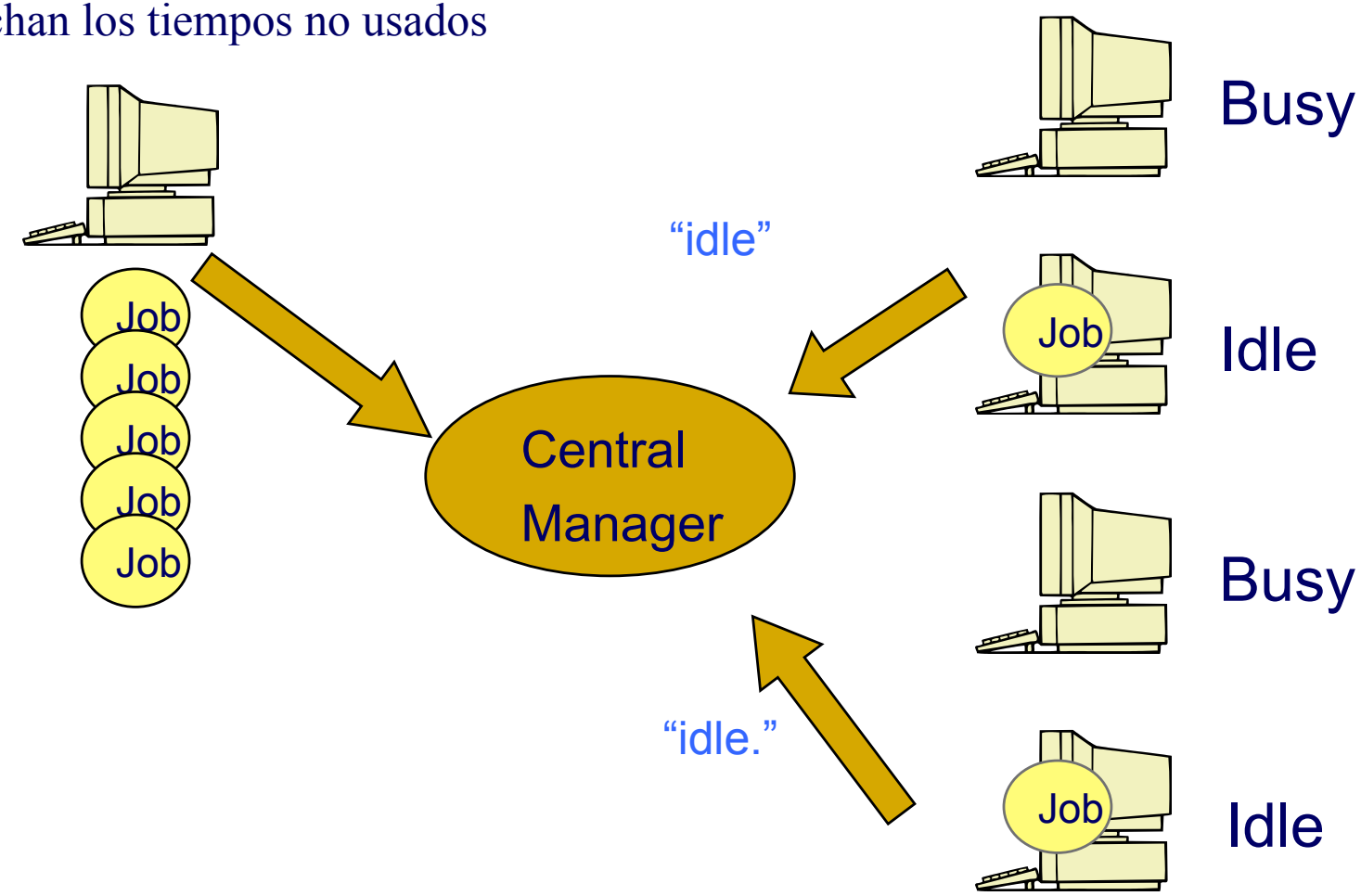
- Extensión de Linux para construir clusters
- http://www.mosix.org
- Ofrece:
  - Single system image
  - Migración de procesos transparente a las aplicaciones
  - Recogida y envío de información de los nodos (velocidad de la CPU, carga, utilización, memoria libre,…)
  - Equilibrio de carga
  - Sistemas de ficheros compartido y paralelo (MFS, GFS)

Estaciones de trabajo inactivas

- En entornos típicos con estaciones de trabajo se desperdicia cerca del 80% de ciclos totales de CPU.
- Uso de estaciones de trabajo inactivas:
  - Ejecutar procesos de forma totalmente transparente en máquinas remotas que se encuentran *inactivas*.
  - Los usuarios de las estaciones de trabajo inactivas no deberían observar una degradación del rendimiento como consecuencia de la ejecución de procesos remotos.

Clusters de workstations (COW)

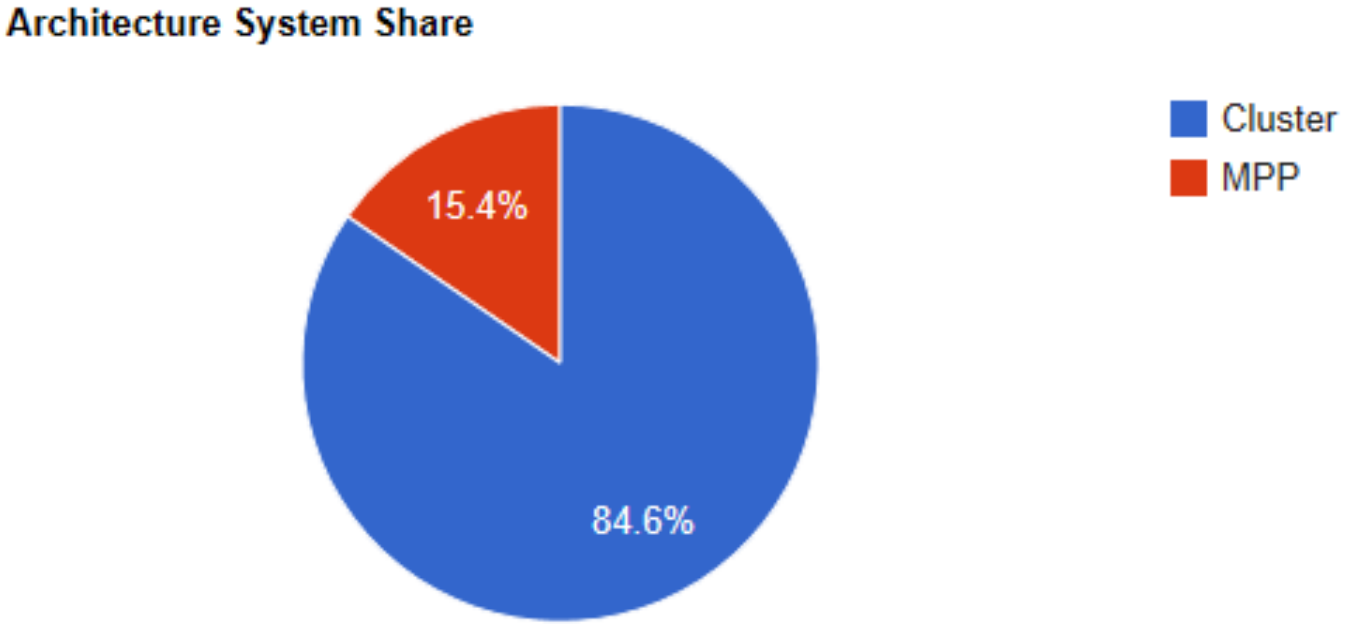
- PCs completos conectados por una red
- Se aprovechan los tiempos no usados



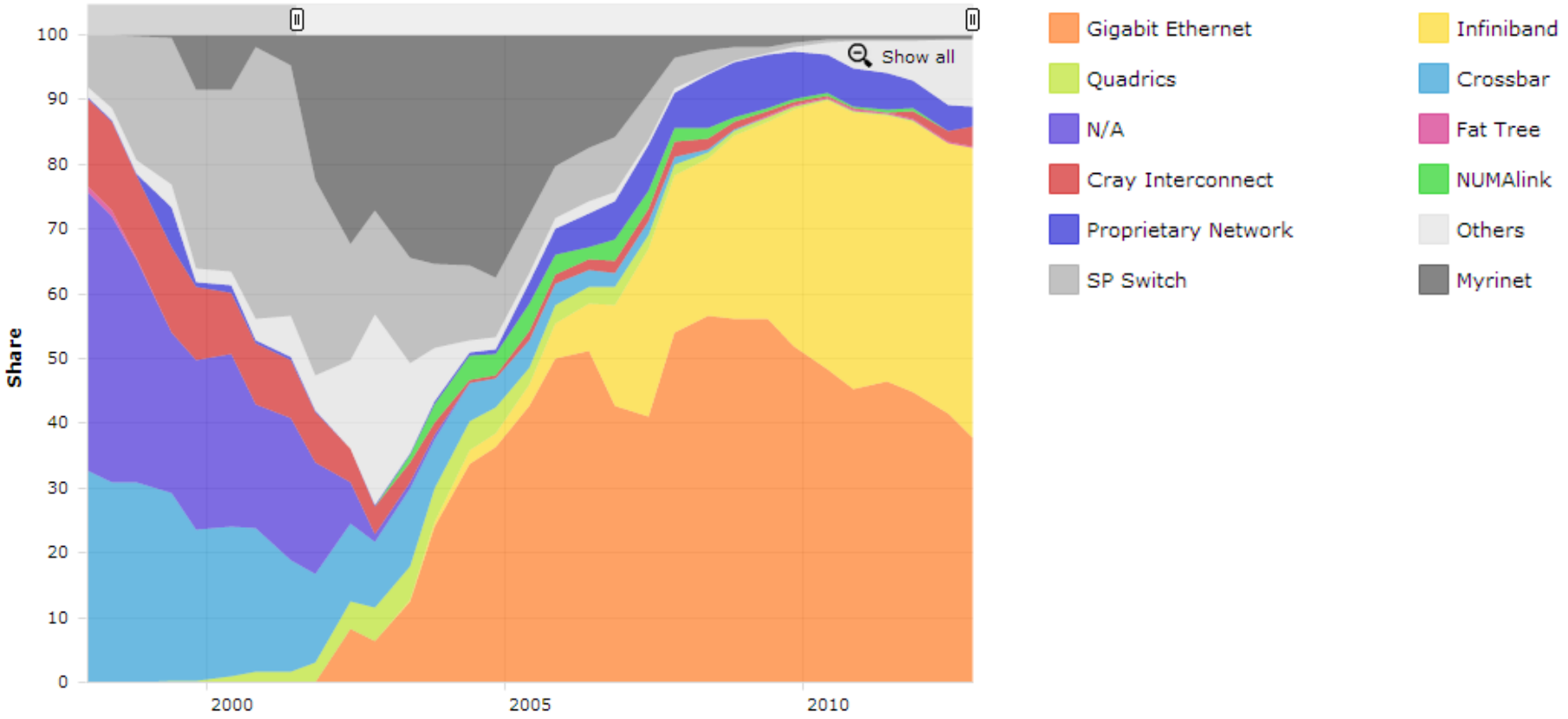
Condor

- http://www.cs.wsic.edu/confor
- Sistema que convierte un conjunto de workstations y clusters dedicados en un sistema distribuido de alto productividad
- Los trabajos se ejecutan en un nodo solo cuando el usuario no está usándolo interactivamente
- Tipos de aplicaciones a ejecutar:
  - Secuenciales
  - Paralelas (MPI)
  - DAG

Clusters en el top500



Evolución de los sistemas de interconexión



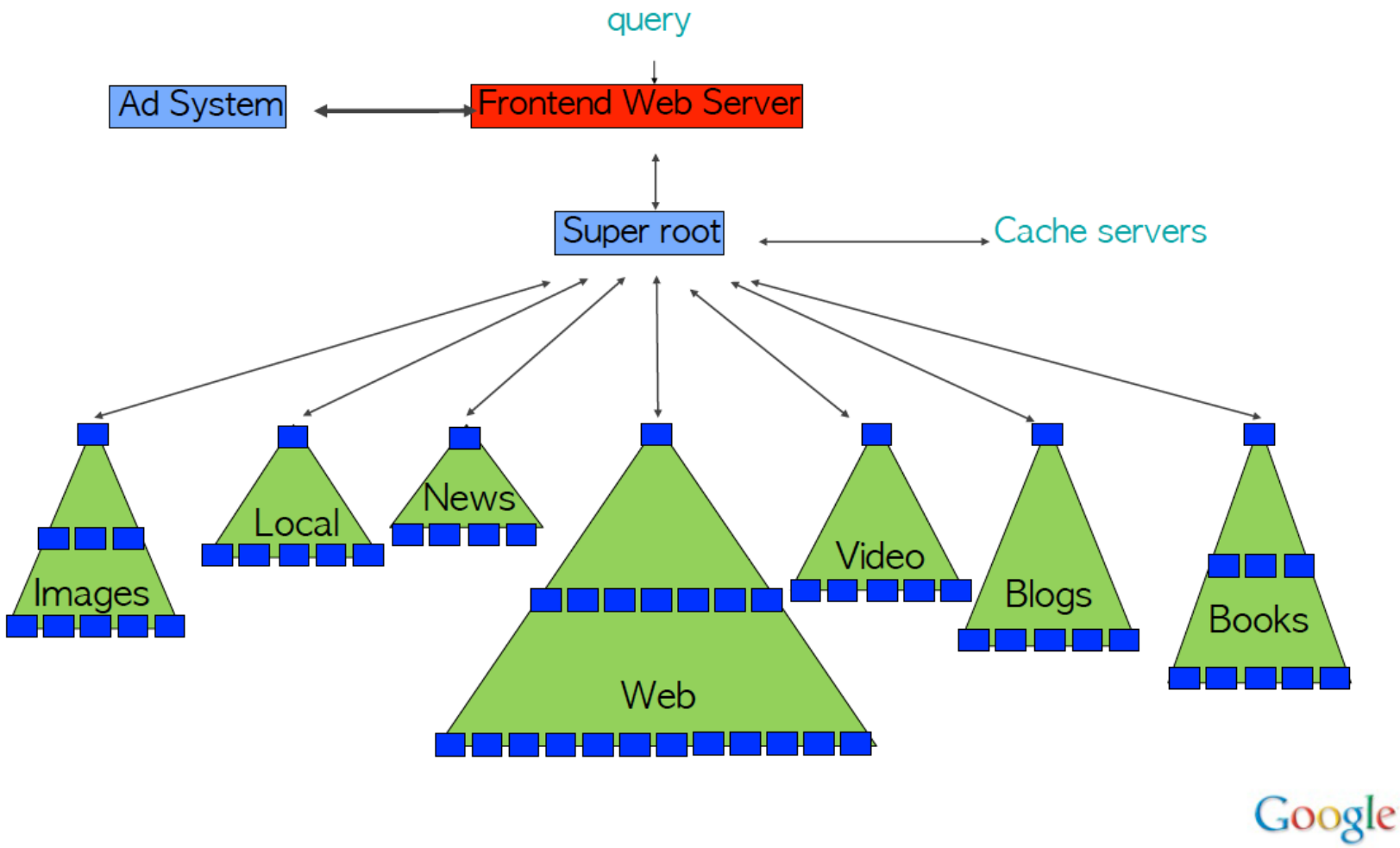
Ejemplo: Google server

- Clusters geográficamente distribuidos
  - Cada uno con miles de máquinas
- El DNS traduce las peticiones al cluster más cercano
- Los servicios ofrecen paralelismo y tolerancia a fallos
- El objetivo de la arquitectura es atender miles de peticiones con tiempos de servicio muy pequeños

Software techniques to tolerate latency variability in large-scale services

- From:
  - The Tail at Scale
    Jeffrey Dean and Luiz anDdré Barroso
    Doi:10.1145/2408776.2408794
    Communications of theACM| february 2013 | vol. 56 | no. 2

Target


query
Ad System
Frontend Web Server
Super root
Cache servers
Images
Local
News
Web
Video
Blogs
Books
Google

Objective

- To provide low latency for interactive services as the size and complexity of the system scales up
- To reduce temporary high-latency episodes
- To provide services with predictably responsive using less-predictable parts
  - Just as fault tolerant computing create a reliable service using less-reliable parts

Why variability exists?

- Shared resources. Different applications using the same machine (CPU cores, caches,…)
- Daemons that execute in background and when are scheduled produce multi-milliseconds stops
- Global resources sharing. Applications running on different machines that use global resources
  - Shared file systems, network switches, …
- Maintenance activities:
  - Data reconstruction in distributed file systems
  - Periodic log compactions
  - Periodic garbage collection in garbage-collected languages

Why variability exists? (cont.)

- Queuing. Multiple layers of queuing in intermediate servers and network switches amplify latency variability
- Power limits. Modern CPUs are designed to temporarily run above their average power envelope to mitigate thermal effects
- Garbage collection in solid-sate storage devices, that need to periodically garbage collect a large number of data blocks increasing read latency by a factor of 100
- Energy management. Power-saving modes in many types save energy but add latency when moving from inactive to active modes.

Latency variability is amplified by scale

- Common technique to reduce latency in large scale online services:
  - Parallelize sub-operations across many different machines, with each sub-operation co-located with its portion of a large dataset.
    - Large fanout services
  - These sub-operations must all complete within a strict deadline for the service to feel responsive

Problem of parallelization

- Consider a system where each server typically responds in 10 ms the 99 % of the time and 1 second the 1 %
- What happens when a request in handled only in one such server?
  - One user request in 100 will be slow (one second)
- What happen when the request is parallelize in 100 sub-operations that run in 100 such servers in parallel?

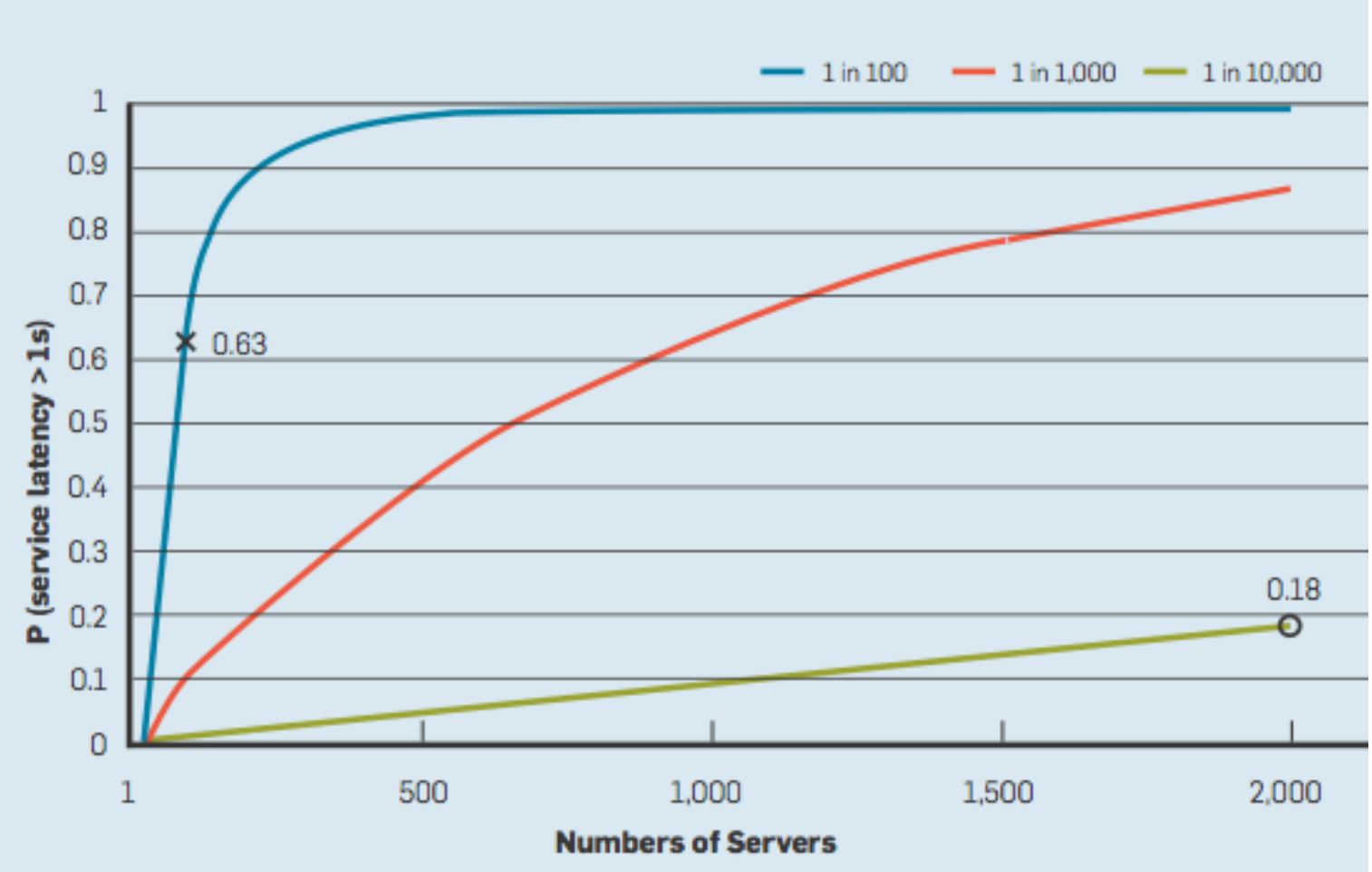
Problem of parallelization

- Consider a system where each server typically responds in 10 ms the 99 % of the time and 1 second the 1 %
- What happen when a request in handled only in one such server?
  - One user request in 100 will be slow (one second)
- What happen when the request is parallelize in 100 sub-operations that run in 100 such servers in parallel?
  - A request finalize in 1 ms when all servers complete the sub-operation in 1 ms
    - Probability = $0.99^{100} = 0.37$ (parallel system)
  - Therefore: 63% of user requests will take more than one second

For large scale

- Consider a better serer with only one in 10000 requests experiencing more than one second latencies
- If the request is parallelize in 2000 such servers:
  - 20 % of user requests will take more than one second

Probability on one-second service-level response time



Techniques to reduce variability latency

- Techniques to reduce latency variability
- Techniques to tolerate latency variability

Basic latency reduction techniques

- Differentiated service classes
  - Prioritized request queues in servers
    - Objective: keep low-level queues short so higher-level policies take effect more quickly
    - This allows the servers to issue incoming high-priority interactive requests before request for latency-intensive batch operations are served
- Break large requests into sequence of small requests to allow interleaving of the execution of other short-running requests
  - Google's web search engine uses it to prevent a small number of very computationally expensive queries to add latency to a large number of concurrent cheaper queries

Basic latency reduction techniques (cont.)

- Manage expensive background activities
  - Background tasks can create CPU, disk or network load
  - e.g: log compaction in distributed storage systems, garbage collection
  - Break down heavyweight operations into smaller operations
  - Defer expensive activity until load is lower
- Synchronize background activities in all servers
  - Enforces a brief burst of activity on each machine simultaneously
  - Without synchronization, a few machines are always doing some background activity, increasing the latency in parallelized requests

Techniques to tolerate latency variability

- Within request short-term adaptations
  - Take action within single high-level request
  - Goals:
    - Reduce overall latency
    - Don’t increase resource use too much
- Cross request adaptation
  - Examine recent behavior
  - Improve latency of future requests
  - Time scale: 10 seconds to minutes

Within request short-term adaptations

- Hedged requests: to issue the same request to multiple replicas and use the results from whichever replica respond first
  - The client sends the requests with a brief delay
  - The client cancels remaining outstanding requests once the first result is received
- Google benchmark that reads the value of 1000 keys stored in a BigTable distributed across 100 different servers:

	Avg	Std Dev	95%ile	99%ile
No replication	33ms	1524ms	24 ms	52 ms
Delay of 10 ms	14ms	4 ms	20 ms	23 ms
Delay of 50 ms	16ms	12ms	57ms	63ms

- Low increase in request load: 10 ms delay: < 5% extra requests; 50 ms delay: 1 <%

Within request short-term adaptations (cont.)

- Cross-server cancellation
  - Client sends a request to first replica
  - Wait a brief dealty (1 or 2 ms) and send to second replica
  - Servers cancel request on other replica when starting to execute

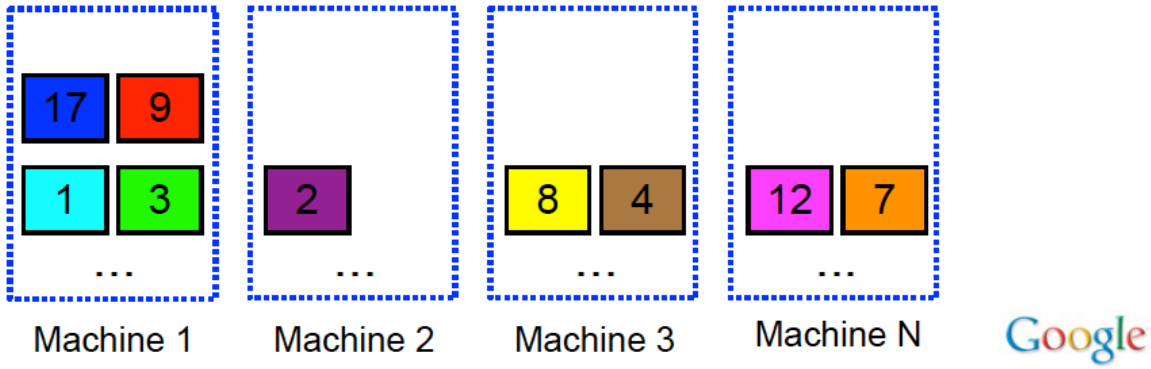
Cluster state	Policy	50%ile	90%ile	99%ile	99.9%ile
Mostly idle	No backups	19 ms	38 ms	67 ms	98 ms
	Backup after 2 ms	16 ms	28 ms	38 ms	51 ms
+Terasort	No backups	24 ms	56 ms	108 ms	159 ms
	Backup after 2 ms	19 ms	35 ms	67 ms	108 ms

Probe remote queues first vs submit work to two queues simultaneously

- Probe remote queues first is less effective than submitting work to two queues simultaneously:
  - Load levels can change between prove and request time
  - Requests service times can be difficult to estimate due to underlying system and hardware variability
  - Clients can create temporary hot spots by all clients picking the same (least loaded) server at the same time

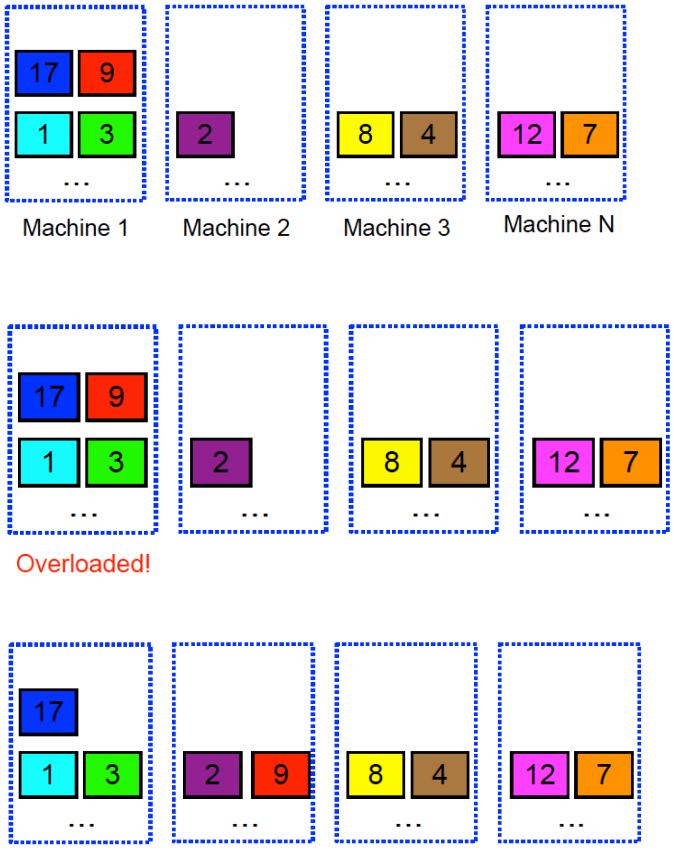
Cross request adaptation

- Micro-partitions:
  - Generate many more partitions than there are machines in the service
  - Dynamic assignment and load balancing of theses partitions to particular machines
  - E.g: BigTable, query serving systems, GFS,…



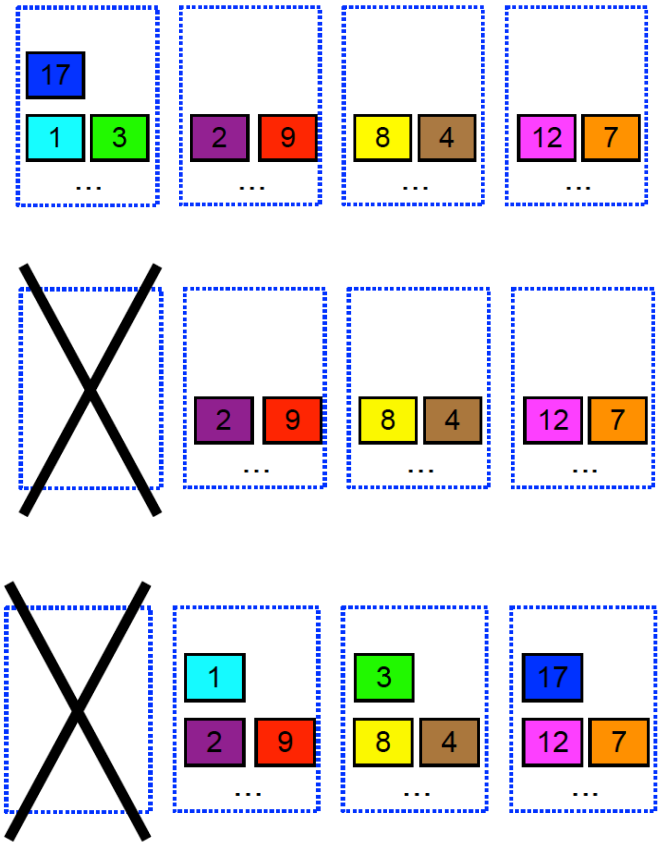
Micro partitions

- Load balancing:



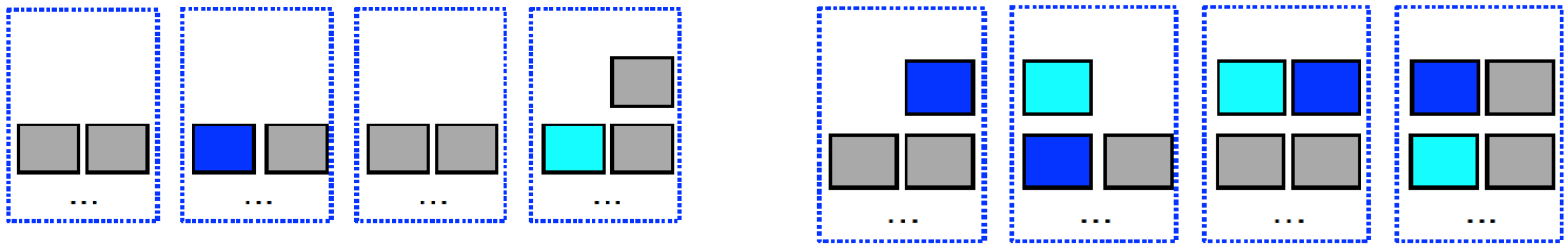
Micro partitions

- Speeds failure recovery



Cross request adaptation (cont.)

- Selective replication: detect heavily used items and make more replicas (statically or dynamically)
  - Static: more replicas of important docs
  - Dynamic: more replicas of particular documents as the query load increases



Cross request adaptation (cont.)

- Latency-induced probation: exclude a particularly slow machine, putting it on probation
  - The source of this problem can be temporary
- Removing capacity and resources under load improve, however, latency
- Continue sending shadow requests to server
  - Allow to measuring latency
  - Return to service when latency back down

Grid Computing

- Motivación, objetivos y definición
- Características y modelos
- Componentes
- Ejecución de trabajos
- Acceso a datos
- Ejemplo de planificación de recursos
- Introducción a Globus

Typical Scenario at ESA

- Scientific data received from satellites are important for scientists
- VILSPA maintains an archive of all scientific data obtained from completed and ongoing missions
  - The XMM-Newton Observatory
  - The Infrared Space Observatory
  - The International Ultraviolet Explorer
- This information is made available and shared to the scientific community for research purposes.

Example of data processing

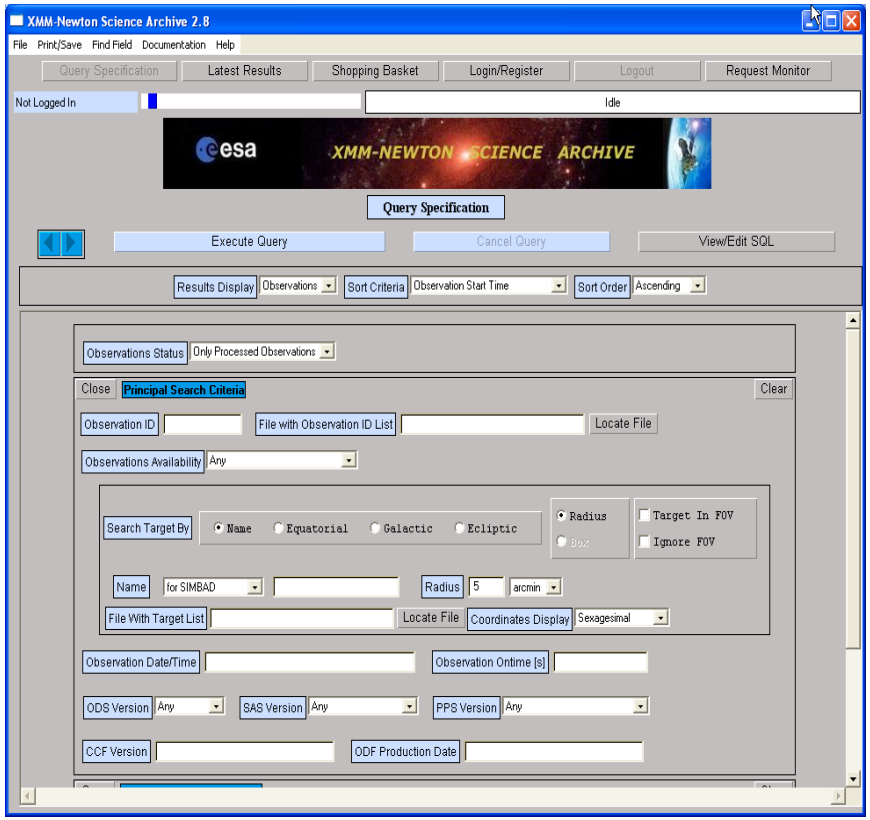
Search data

results

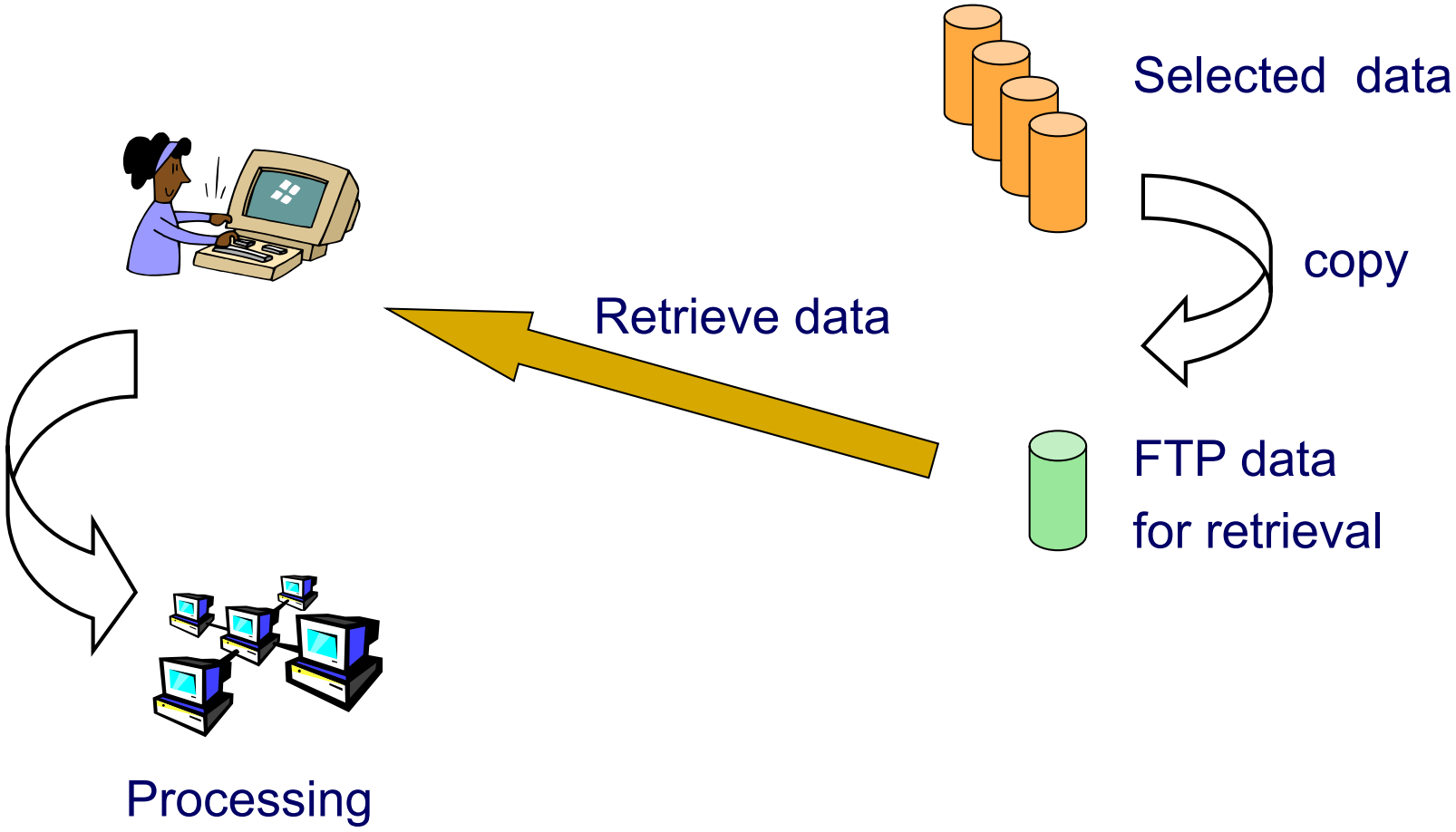
Select data

Retrieval request

XMM-Newton Science Archive (XSA) application



Example of data processing



Simulaciones

- Se quiere simular el funcionamiento de una función F(x,y,z) para 20 valores de X, 10 de y y 30 de Z
  - 20 *10*30 = 6000 combinaciones
- F tarda una media de 2 horas de ejecución en una estación de trabajo convencional con un consumo de memoria y de E/S moderado

Grid computing

- Tecnología cuyo objetivo es la **compartición de recursos en Internet** de forma uniforme, transparente, segura, eficiente y fiable
- Análoga a las **redes de suministro eléctrico**:
  - Ofrecen un único punto de acceso a un conjunto de recursos distribuidos
  - Geográficamente **en diferentes dominios de administración**

Some definitions

- A grid system is a collection of distributed resources connected by a network, located at different administrative domains, accessible to users and applications in order to reduce overhead and increase the performance.
- A grid application is an application that operates in a grid environment.
- A grid middleware is the software that facilitates writing grid applications and manages the grid infrastructure.

Other Definitions

- The Globus Project defines Grid as:
  - Infrastructure that enables the integrated, collaborative use of high-end computers, networks, databases, and scientific instruments owned and managed by multiple organizations

    What is the Grid? A Three Point Checklist (2002)
- The Gridbus project defines Grid as:
  - Type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost and users' quality-of-service requirements.

Main grid characteristics

- Large scale
- Geographical distribution
- Heterogeneity
- Resource sharing
- Multiple administration domains
- Resource coordination
- Transparent access
- Dependable access
- Consistent access

Main grid uses

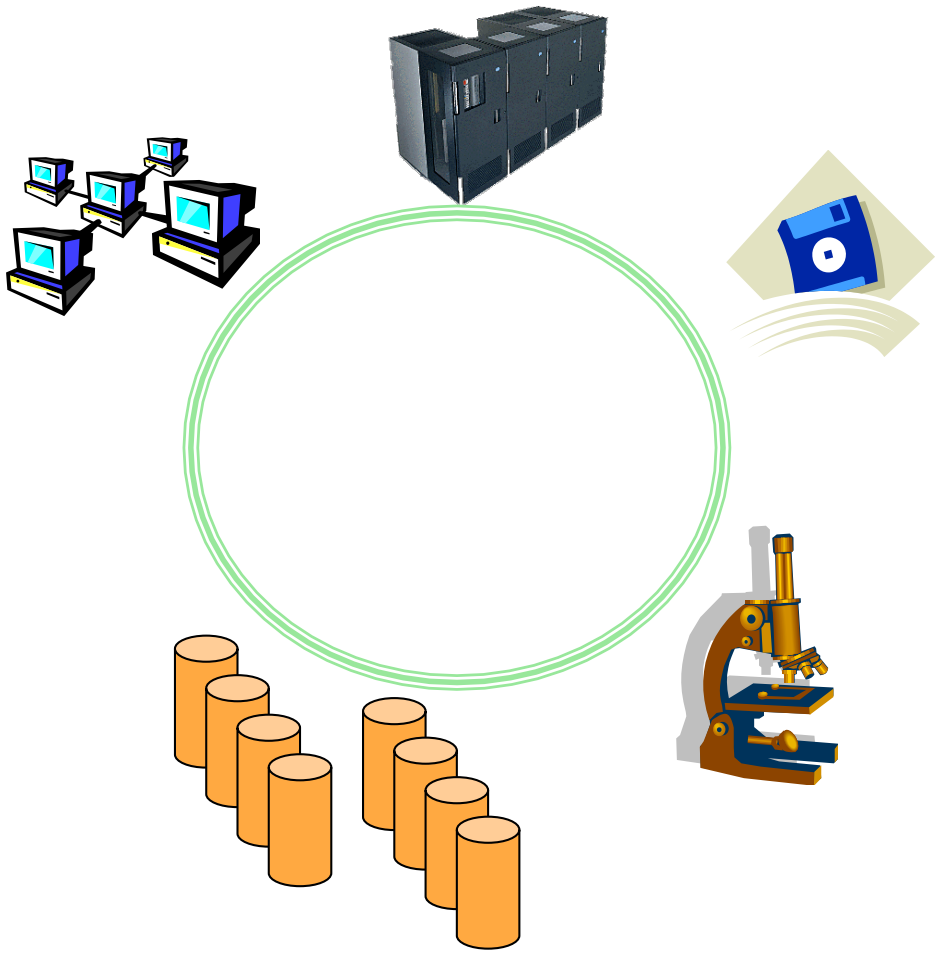
- Distributed supercomputing
- High throughput computing
- On-demand computing
- Data-intensive computing
- Collaborative computing

Result of several trends

- Grid computing is the result of several trends:
  - New standards for object-to-object communications making it easier to build multivendor and multiapplication networks
  - High performance microprocessors
  - High speed networking

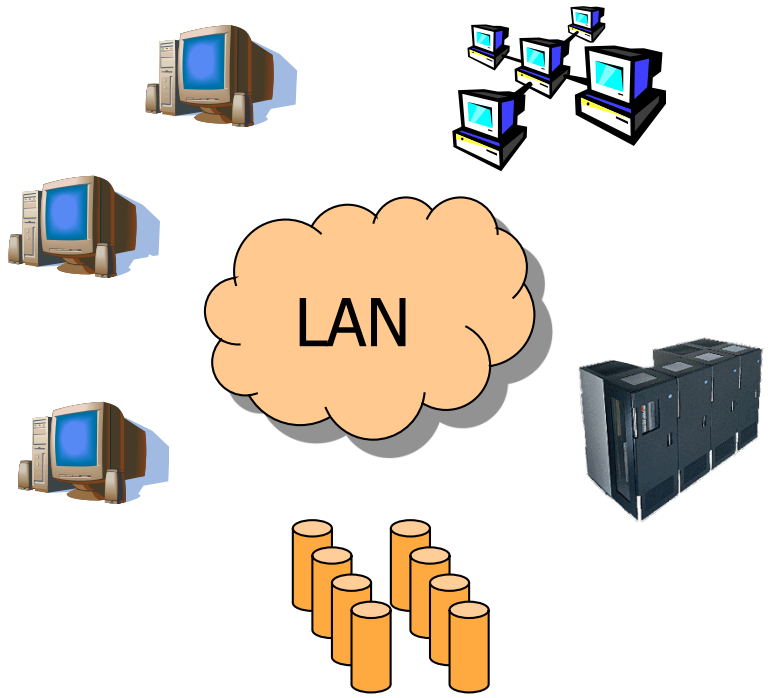
Resources

- Processors
- Storage
- Networks
- Software
- Special Devices



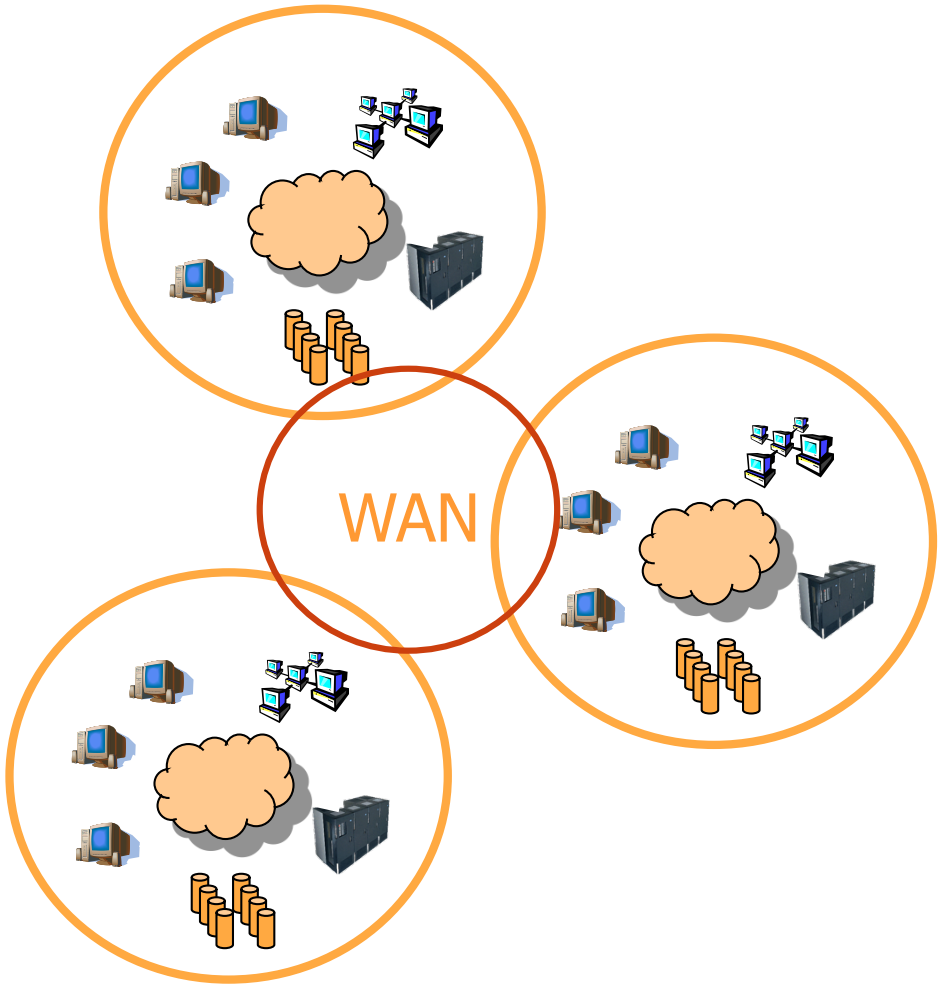
Types of grids

- Intragrids
  - Same organization
  - Heterogeneous resources
  - LAN connection
  - Less security problems
  - Less reliability problems



Types of grids

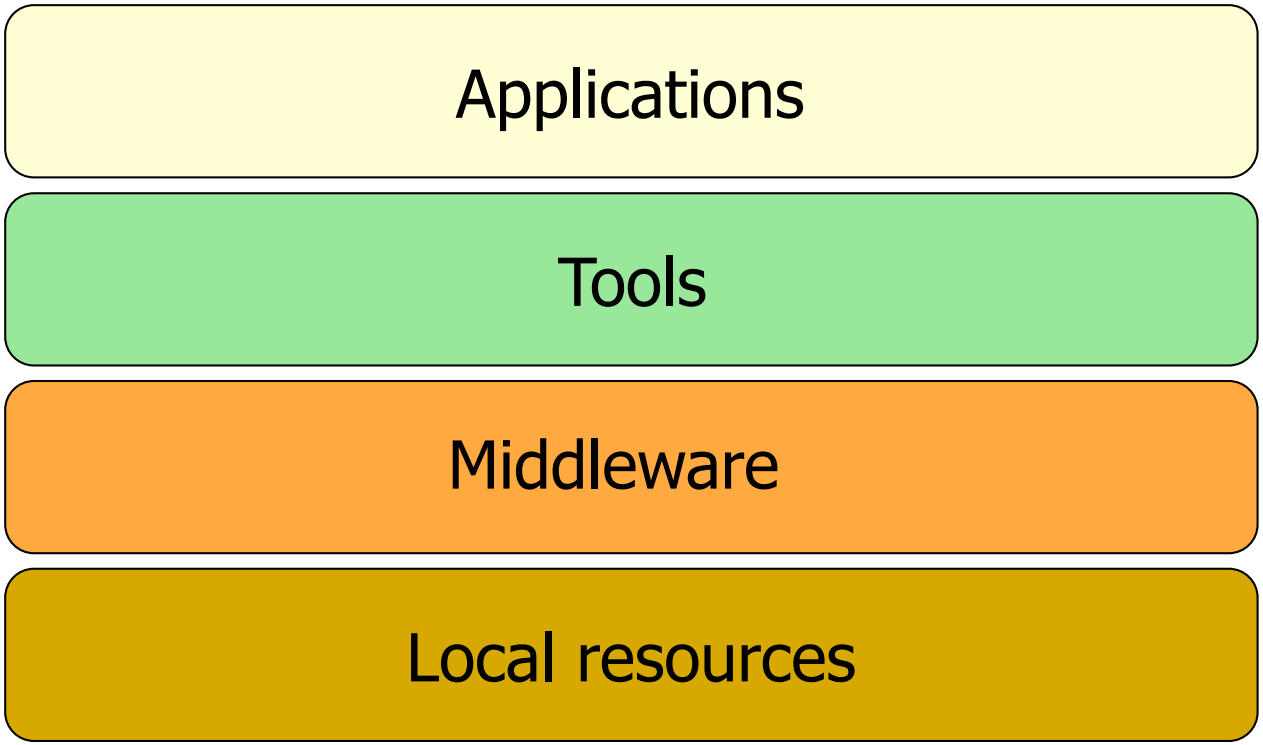
- Intergrid
  - Multiple organizations
  - Multiple administrative domains
  - Connection using a WAN
  - Security is an important thing
  - Reliability problems



Grid Architectures

- Virtual organizations (VO)
  - Different physical organizations that share resources and collaborate in order to achieve a common goal
  - A VO defines the resources available and the rules for access
- Economic grid
  - Based on economic principles
  - Resources providers (owners) compete to provide the best service to resource consumers (users) who select appropriate resources based on their specific requirements

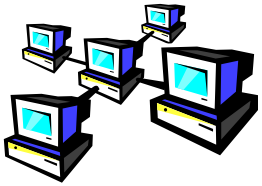
Grid components

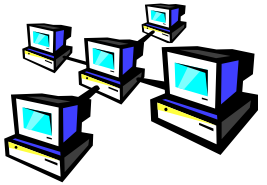


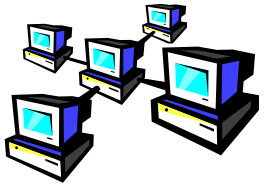
User point of view

applications

results

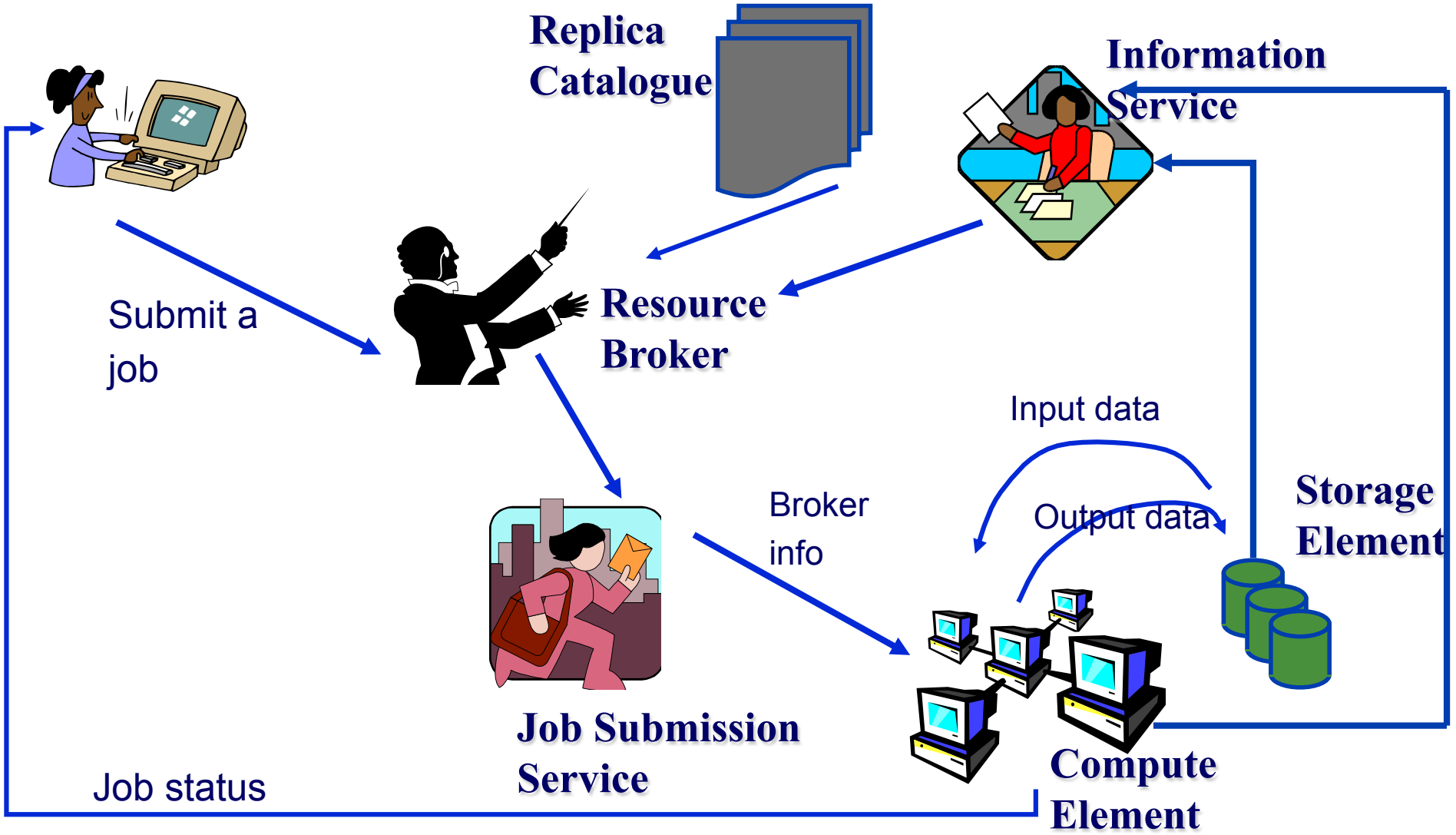






- Brokers
- Schedulers
- Job management
- Data management
- Security model

Jobs execution



JSDL

- JSDL stands for Job Submission Description Language
  - A language for describing the requirements of computational jobs for submission to Grids and other systems.
- A JSDL document describes the job requirements
  - What to do, not how to do it

JSDL Document

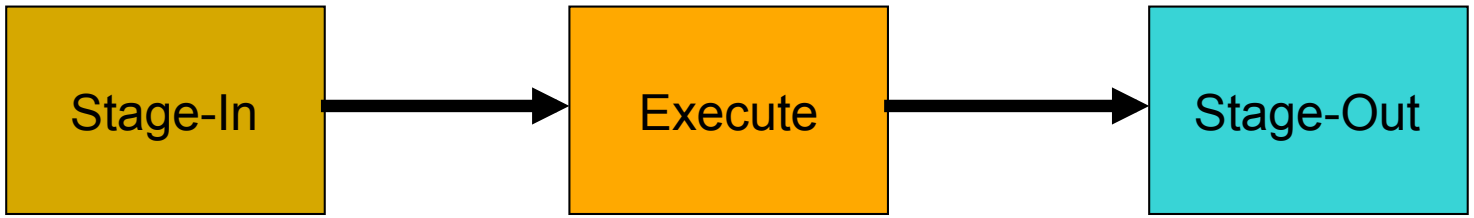
- A JSDL document is an XML document
- It may contain
  - Generic (job) identification information
  - Application description
  - Resource requirements (main focus is computational jobs)
  - Description of required data files

```
<jsdl-hpcp:HPCProfileApplication>
    <jsdl-hpcp:Executable>/bin/echo</jsdl-hpcp:Executable>
    <jsdl-hpcp:Argument>hello, world!</jsdl-hpcp:Argument>
    <jsdl-hpcp:Output>${GLOBUS_USER_HOME}/stdout</jsdl-hpcp:Output>
    <jsdl-hpcp:Error>${GLOBUS_USER_HOME}/stderr</jsdl-hpcp:Error>
</jsdl-hpcp:HPCProfileApplication>
```

- Out of scope, for JSDL version 1.0
  - Scheduling
  - Workflow
  - Security
  - …

Data Staging Requirement

- Solution
  - Assume simple model: Stage-in – Execute – Stage-Out
  - Files required for execution
    - Files are staged-in before the job can start executing
  - Files to preserve
    - Files are staged-out after the job finishes execution



Applications not appropriate for grid

- Parallel applications with many interprocess communication
- Transaction
- Applications with many interdependencies between jobs
- Applications with non standard network protocols

What need applications?

- Applications need programs that process input data and to produce output data in a reliable and efficient way
  - Executables transfer
  - Input data transfer
  - Output data gathering
  - *Checkpoints*
- Applications need transparent access to its input and output data is required
- Two key aspects of this area are:
  - Intra-cluster Storage (local)
  - Distributed Storage (wide-area)

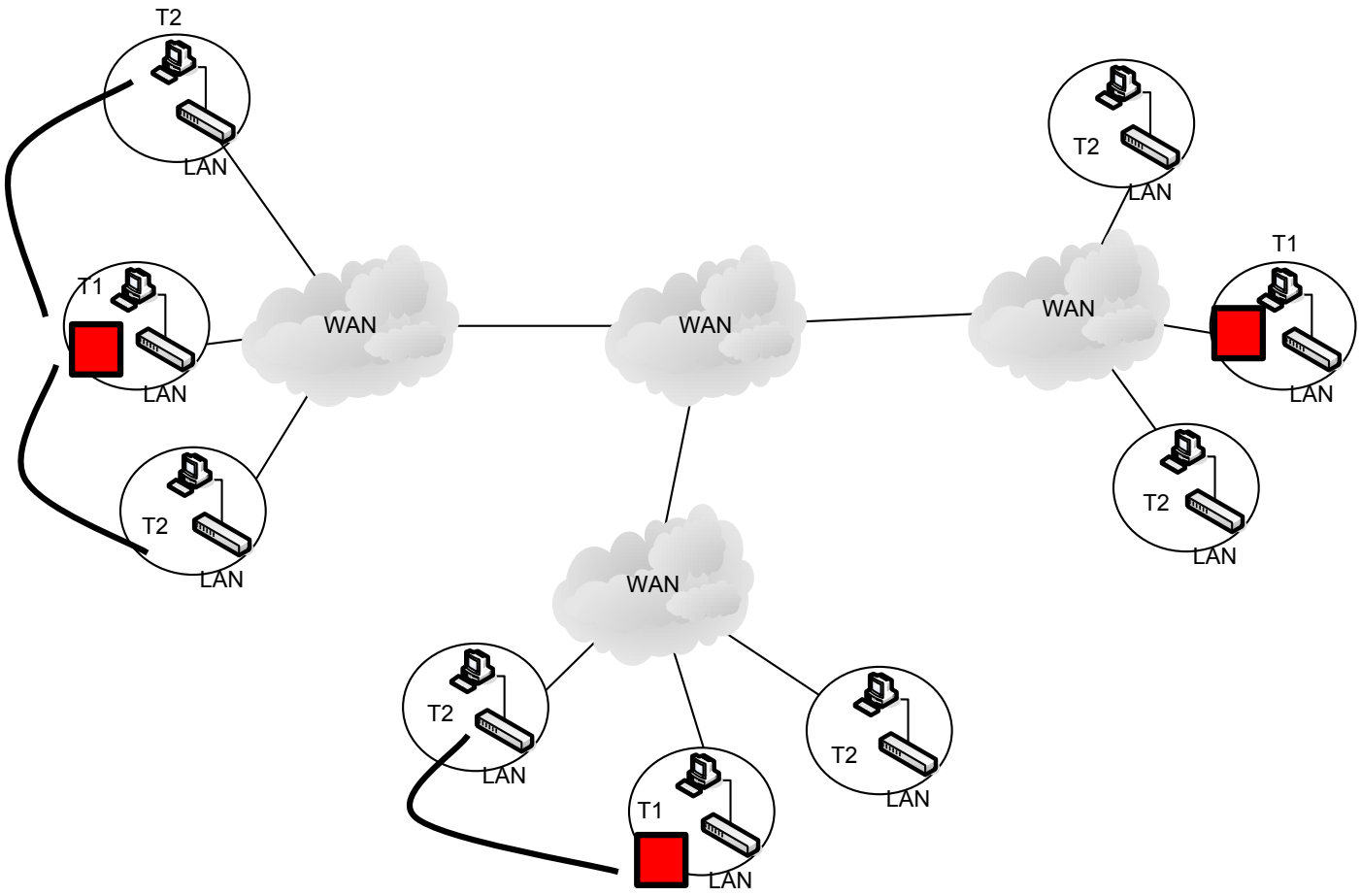
Data considerations

- Where can be executed the jobs?
- How to distribute the job?
- How to send input data?
- Where executables are stored?
- Where is the resource more appropriate to run the job?
- How to obtain and gather output data?
- Data sizes?
- Data localization.
- Interprocess communication.
- *Checkpoint*
- Data availability

Replication

- One server with famous datasets can become in a bottleneck.
- Large Bandwidth could be consumed to transfer huge datasets from server to client.
- Access latency could be significant in grids
- If one server fail the dataset stored in the server can not be accessed.

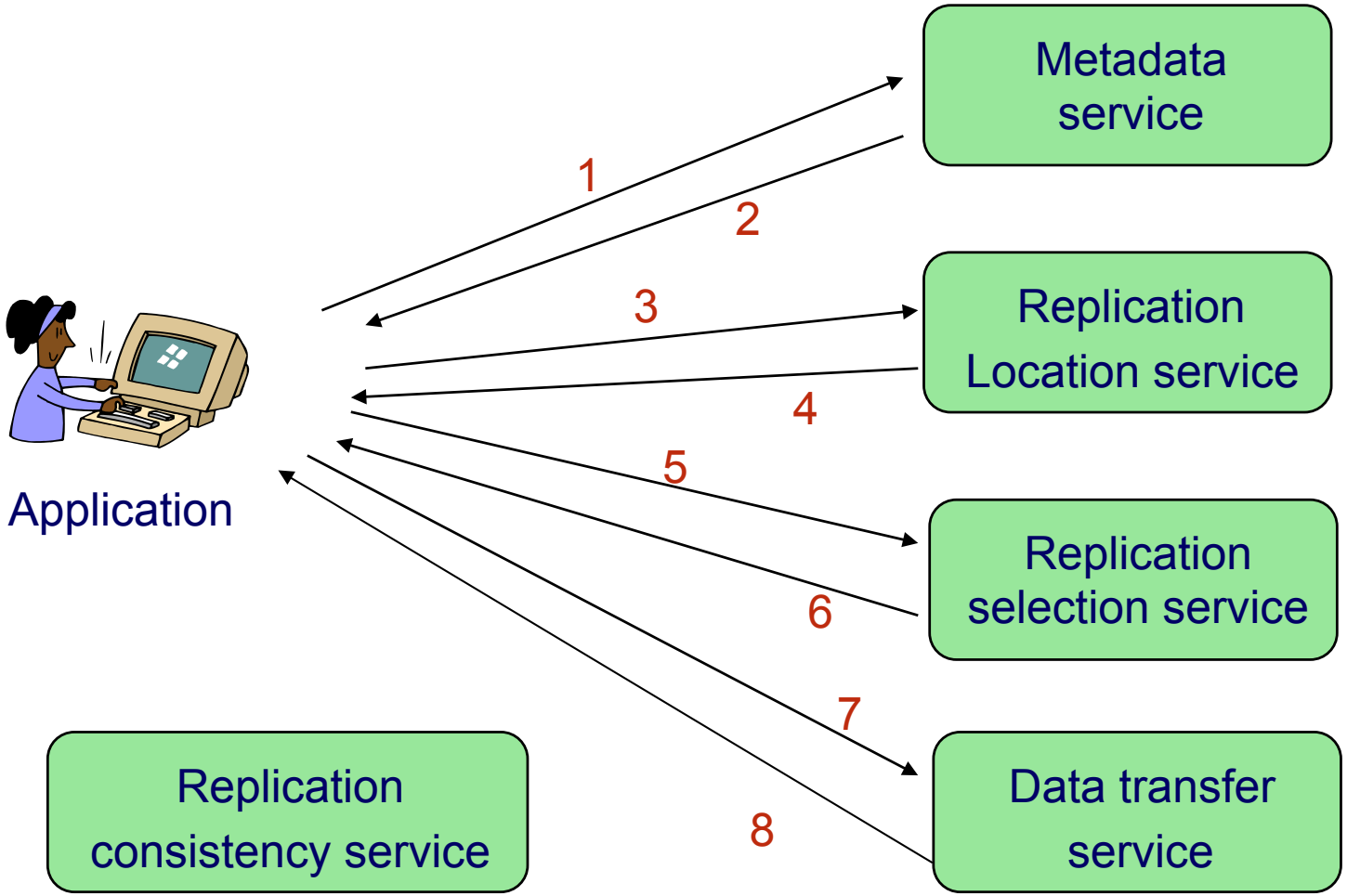
Load balancing



Concepts

- Replication is the distribution of multiple copies of data across the grid.
- Reduce data access overhead providing faster access to data
- Increases data availability
  - Avoid single points of failure
  - Resilient to component failures
- Helps in load-balancing in the Grid
- Improved Scalability
  - Reduce bottlenecks

Replication system components



Filenames

- ***Logical Filename*** *(LFN)* is the name that refers to the full set of replicas for a file.
  - Logical identifier
  - Based on the LFN, all replicas can be looked up in a replica catalogue.

- ***Physical file name*** *(PFN)* is the location of a copy of the file on a storage system .
  - Host + physical location
  - May be a *physical filename* of a file stored on disk

Types of replication

- ***Static Replication***:
  - Replicas have to be manually created and managed.
  - Not feasible for large datasets, and large number of nodes.
  - Cannot adapt to changes.

- ***Dynamic Replication***:
  - All tasks (replica creation, deletion and management) are done automatically.

Replica selection

- Given a logical file name, discover the physical locations of the file.

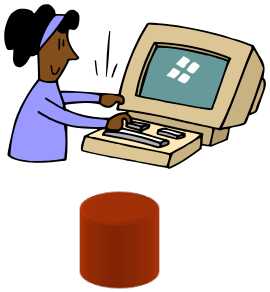
- Replica optimization:
  - Select the "*best*" replica from this set of physical replica locations, based on its performance and data access features.

Consistency

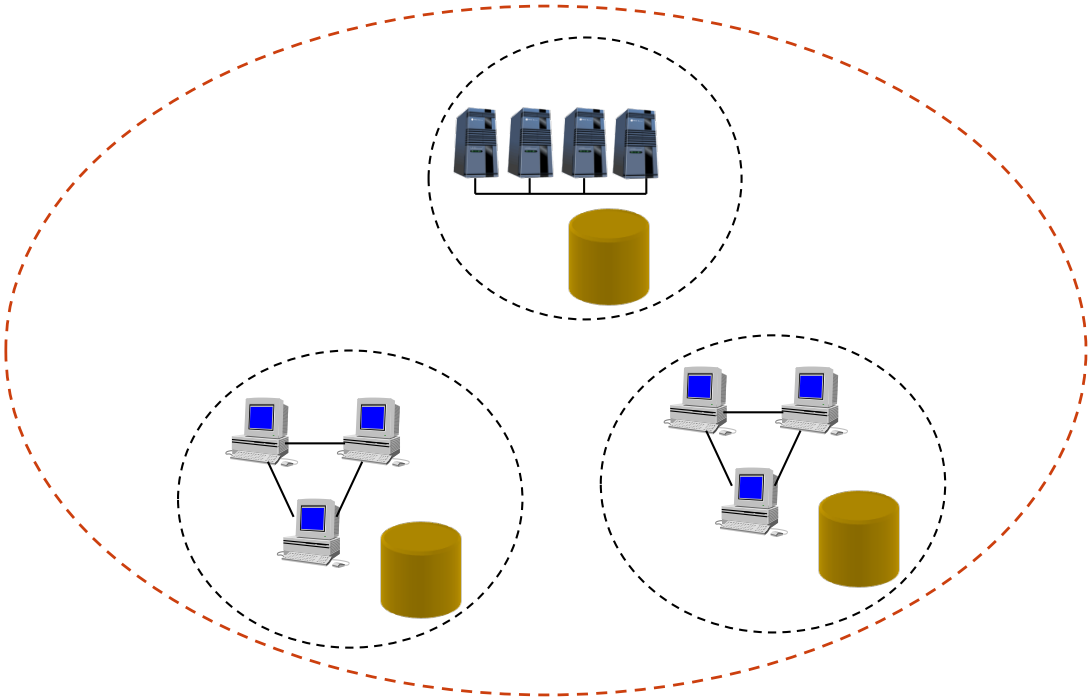
- Most of the data is read-only. If updates occur, general update propagation mechanisms are needed.

- Protocols:
  - Strong consistency protocols
  - Weak consistency (lazy replication)

Modelo de ejecución I

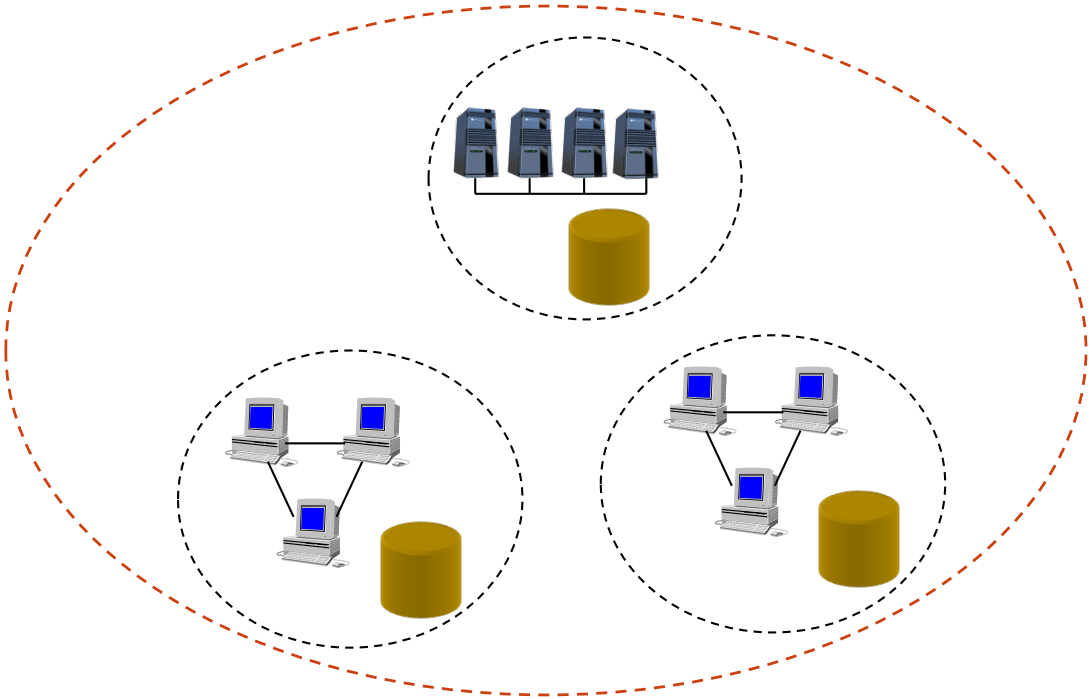


- Programa
- Datos

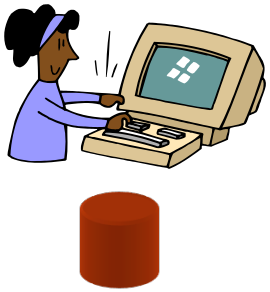


Modelo de ejecución I

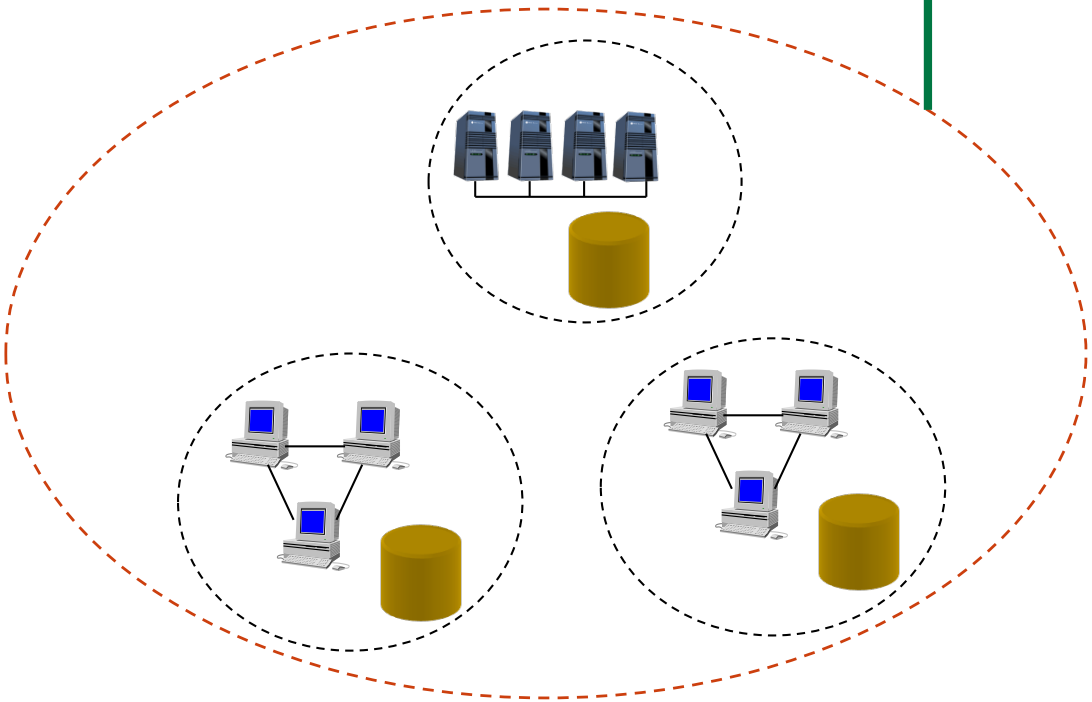
- Programa
- Datos



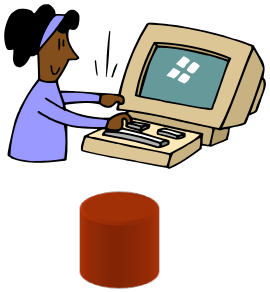
Modelo de ejecución I



- Programa
- Datos



Modelo de ejecución I

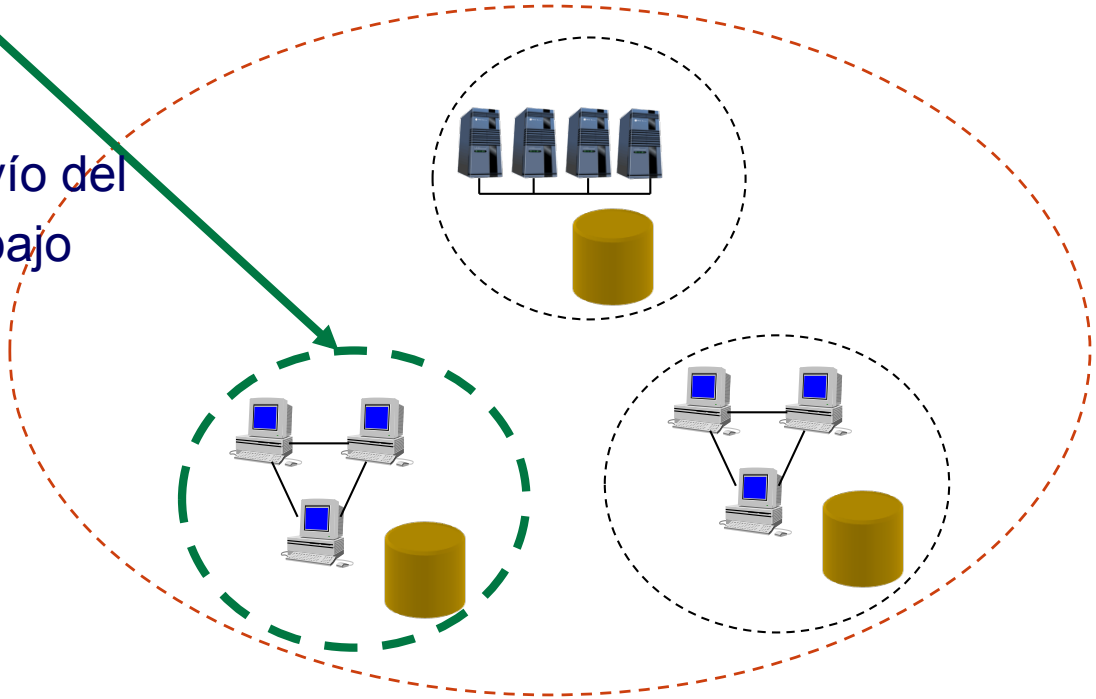


- Programa
- Datos

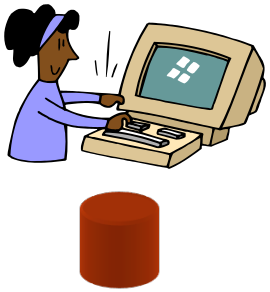
Gestor de recursos

Servicio de información

Envío del trabajo



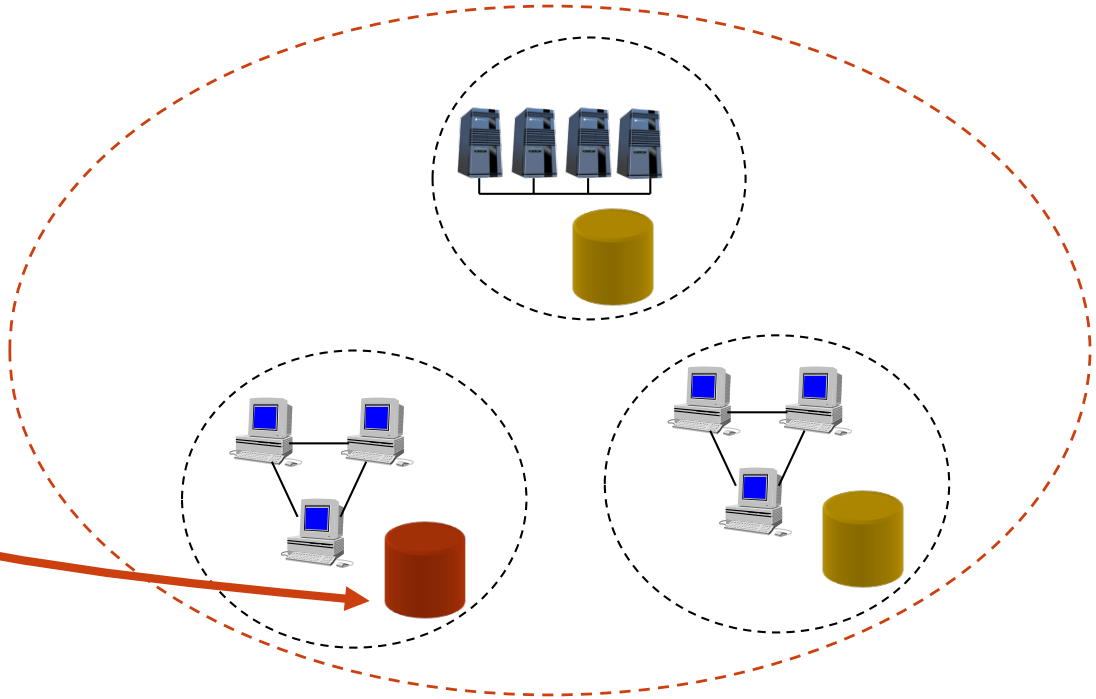
Modelo de ejecución I



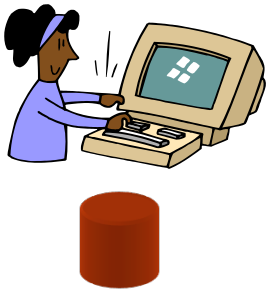
Gestor de recursos

Servicio de información

- Programa
- Datos



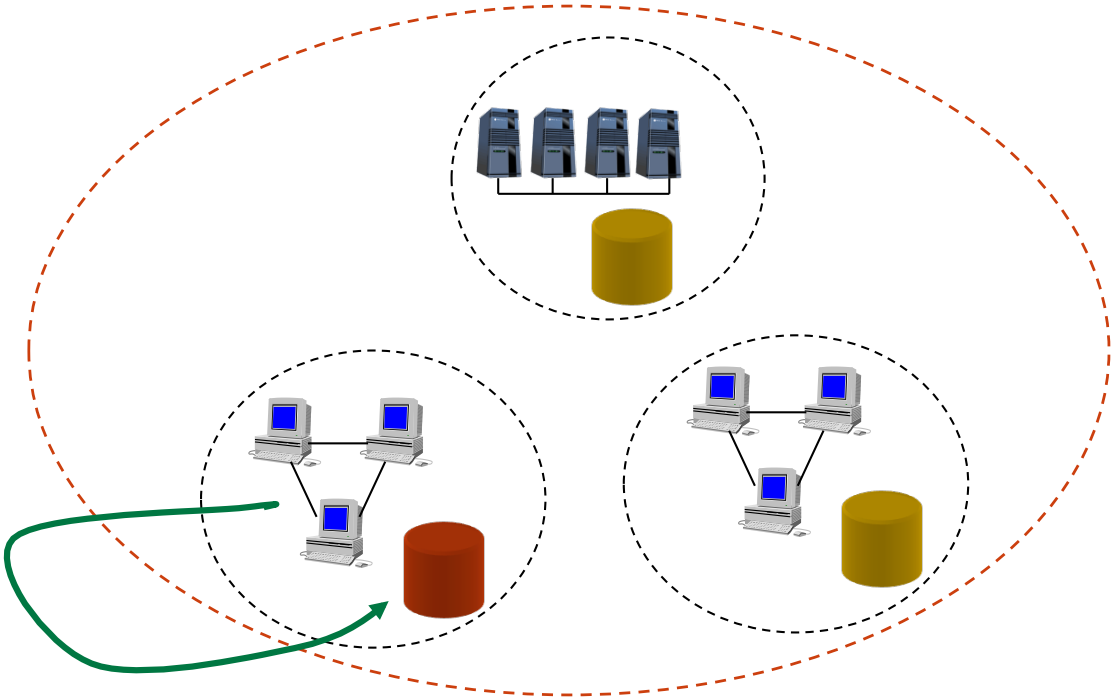
Modelo de ejecución I



- Programa
- Datos

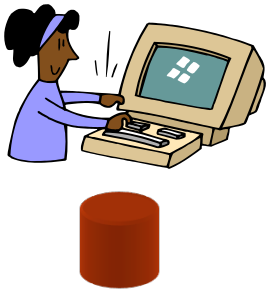
Gestor de recursos

Servicio de información



Se ejecuta El trabajo

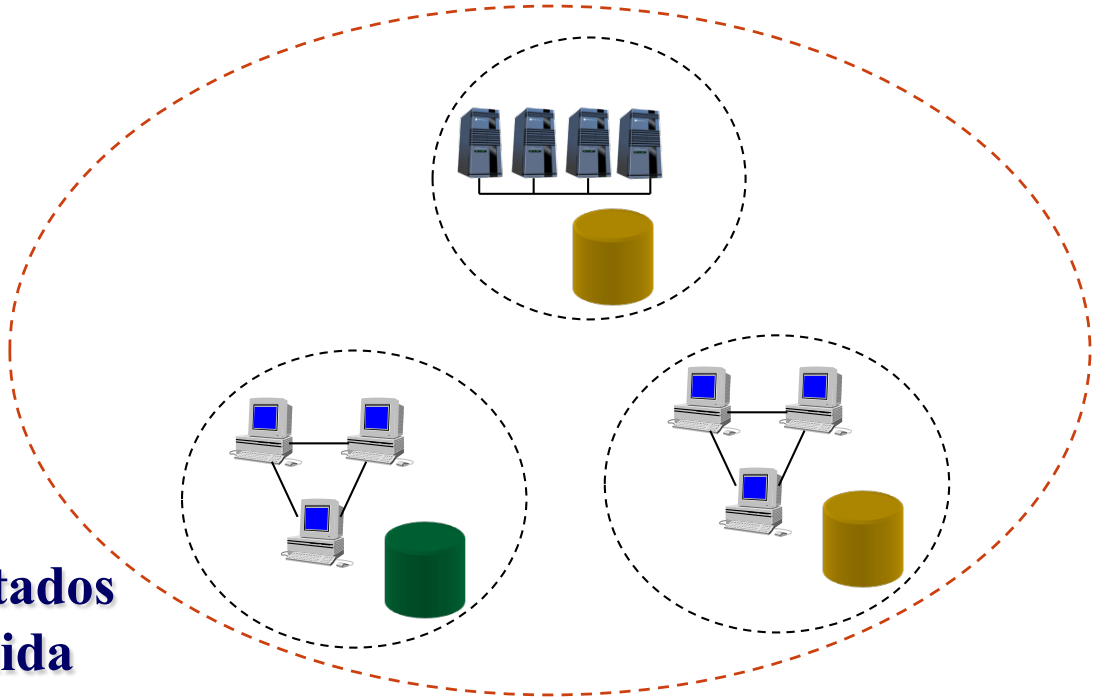
Modelo de ejecución I



- Programa
- Datos

Gestor de recursos

Servicio de información

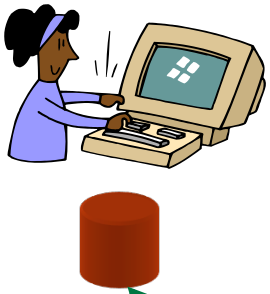


Resultados De salida

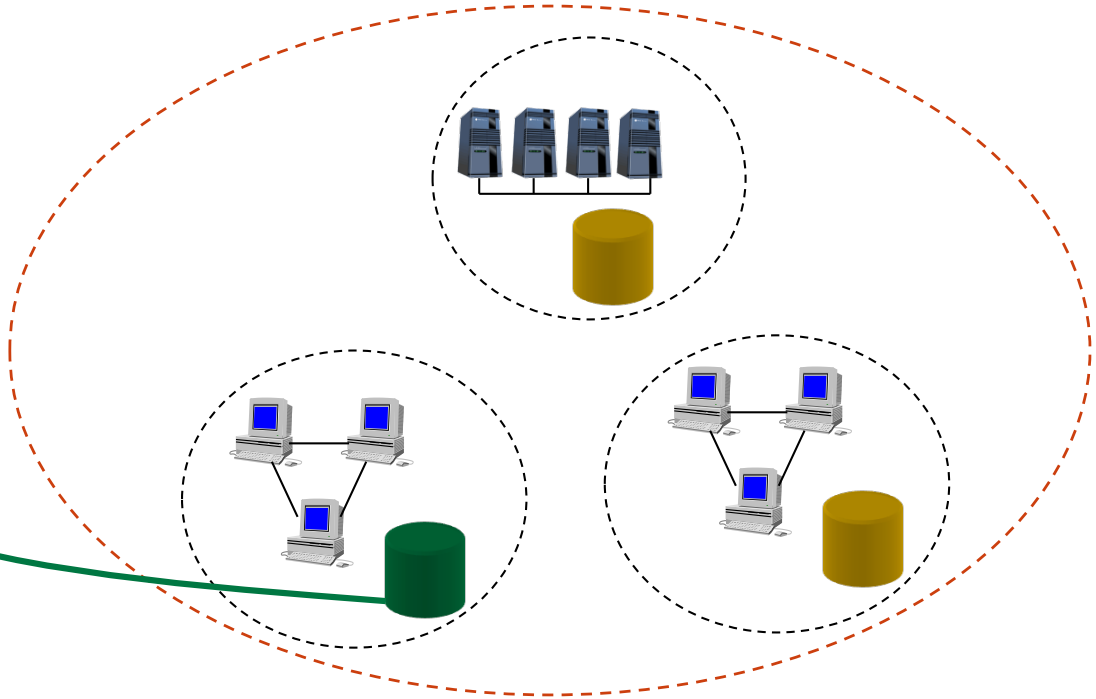
Modelo de ejecución I

Servicio de información

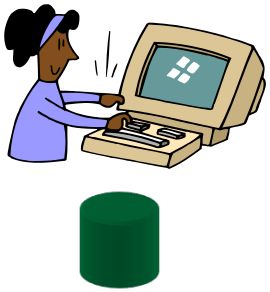
Gestor de recursos



Resultados De salida

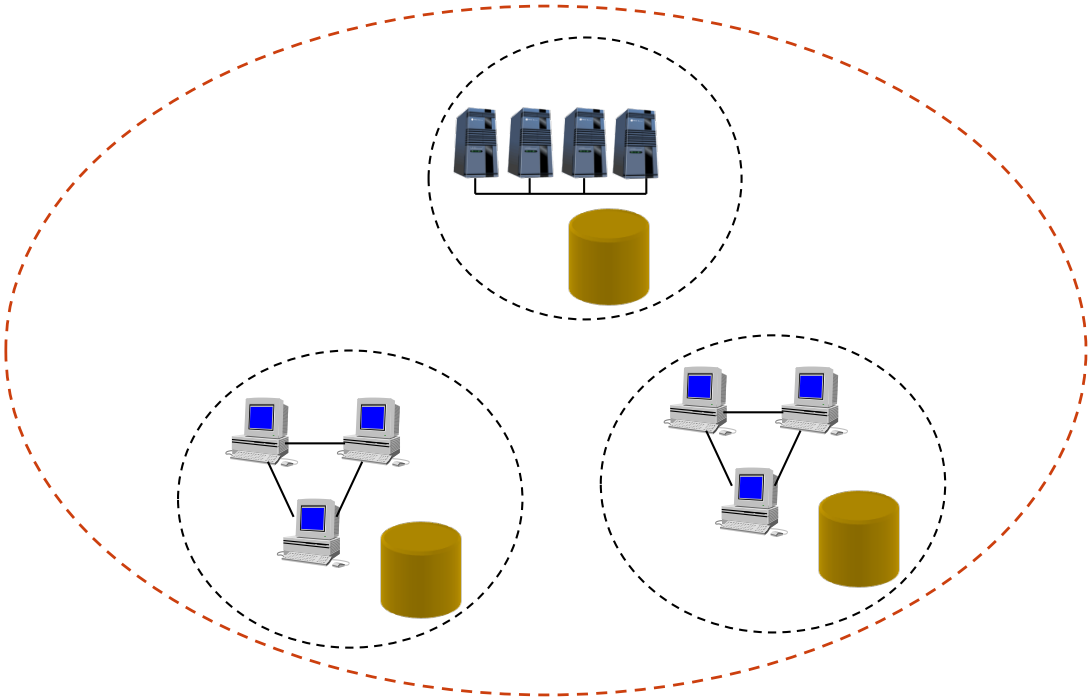


Modelo de ejecución I

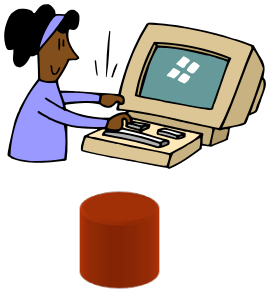


Gestor de recursos

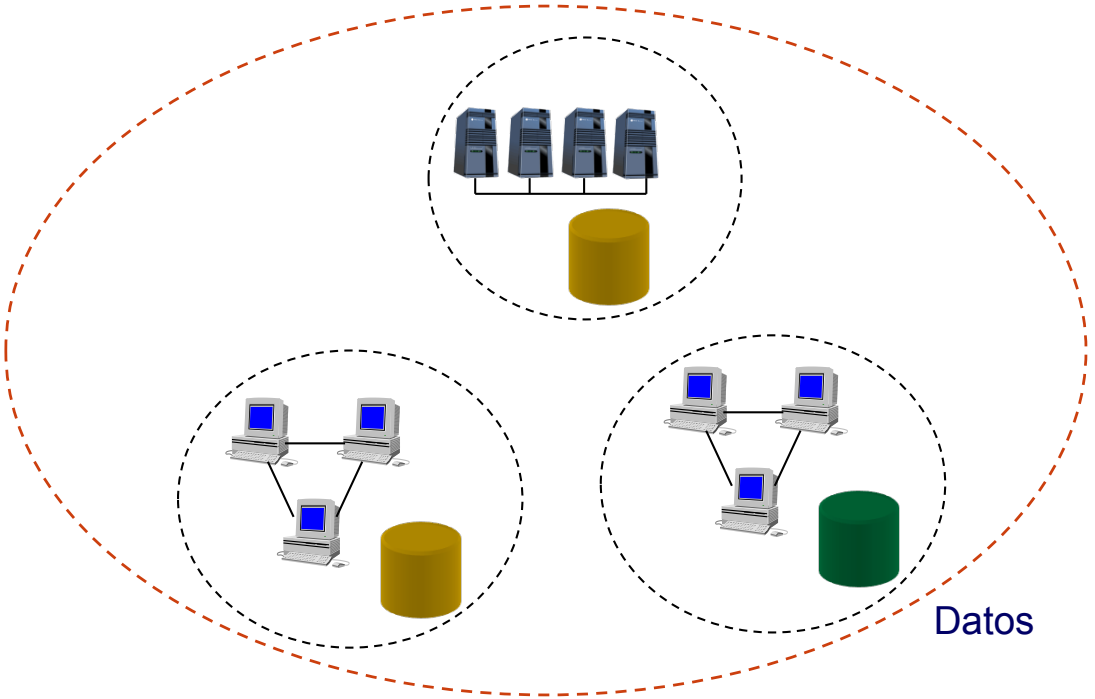
Servicio de información



Modelo de ejecución II

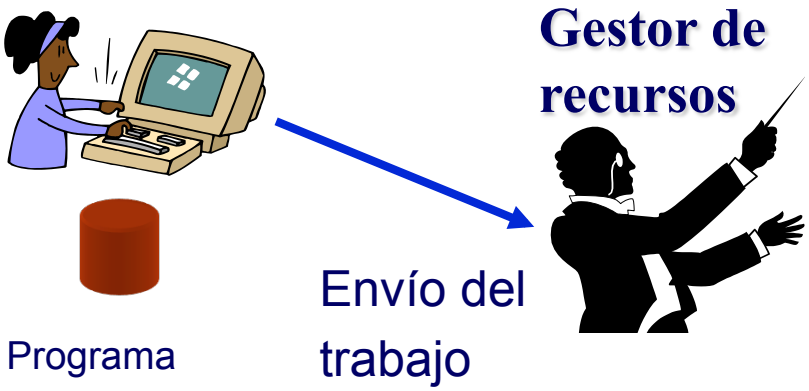


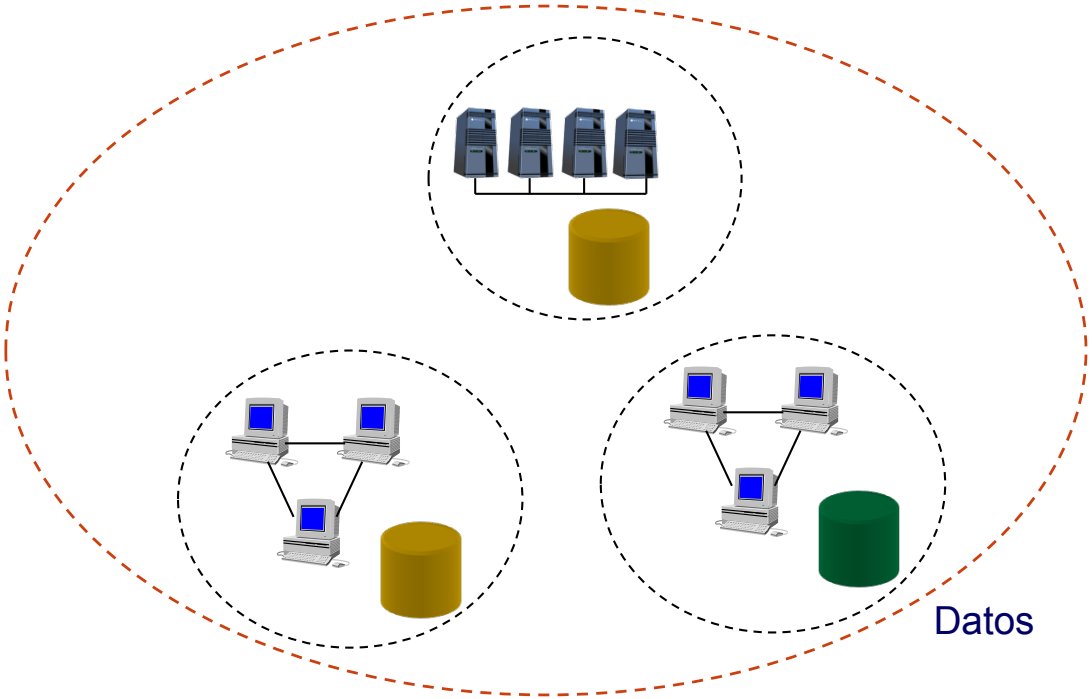
Programa



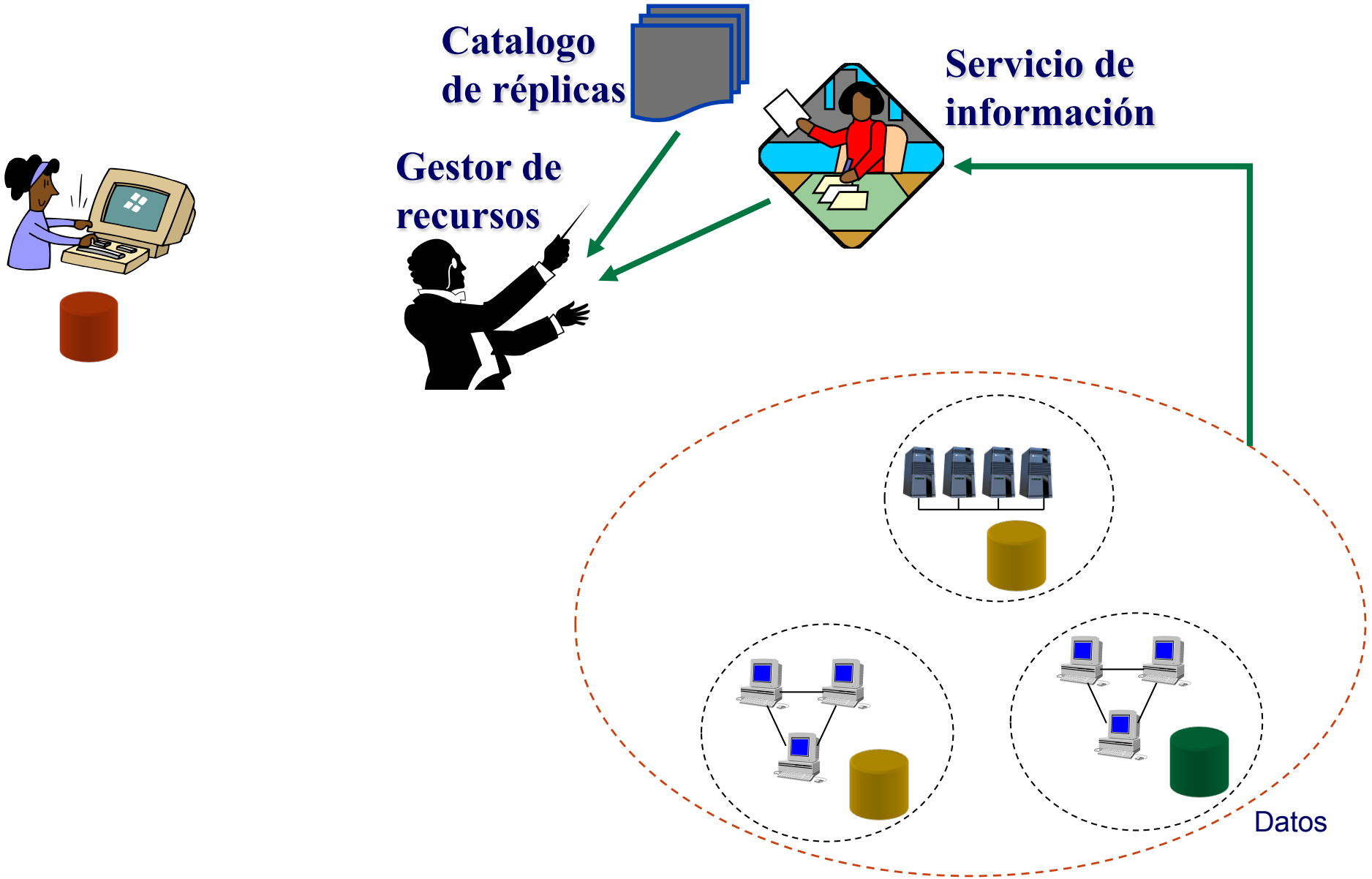
Datos

Modelo de ejecución II

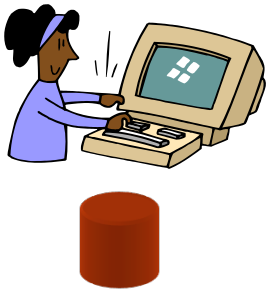




Modelo de ejecución II



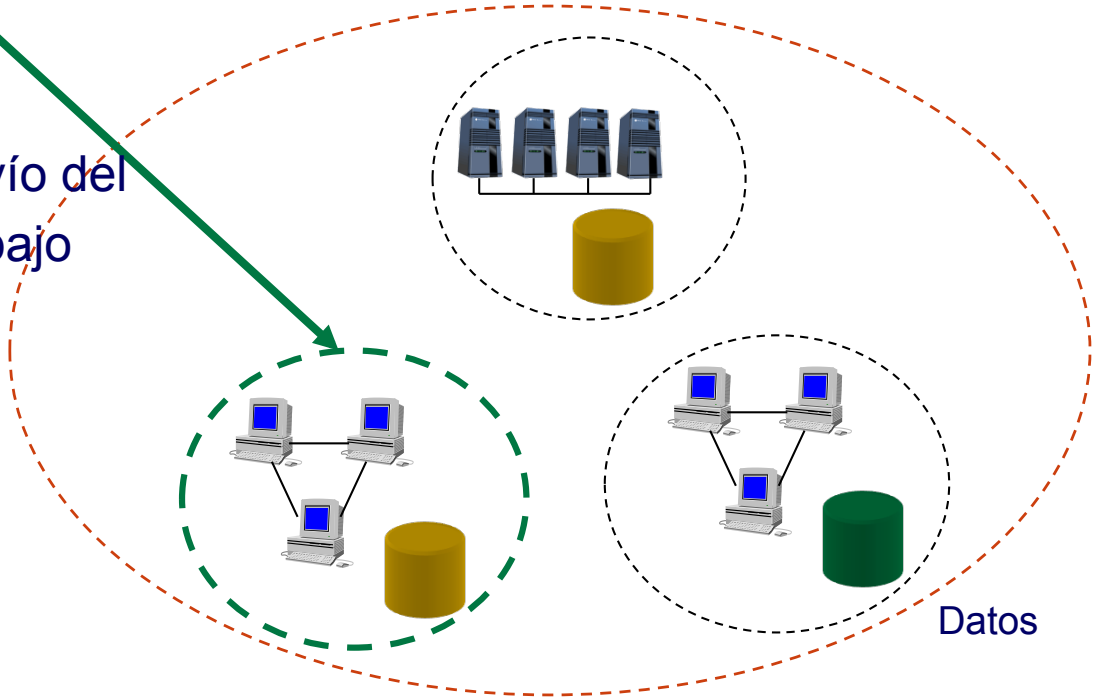
Modelo de ejecución II



Servicio de información

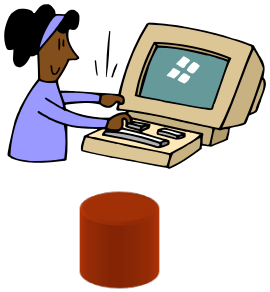
Gestor de recursos

Envío del trabajo



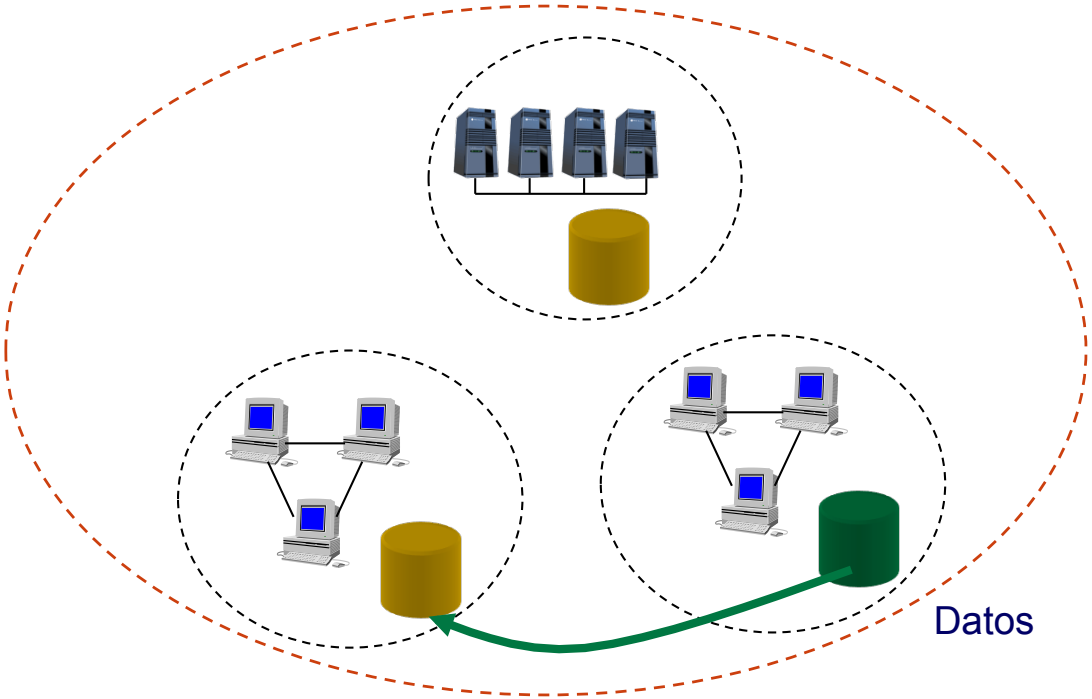
Datos

Modelo de ejecución II

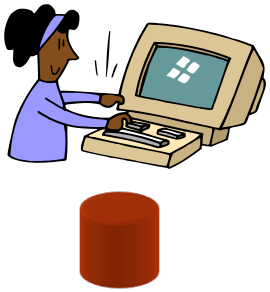


Gestor de recursos

Servicio de información

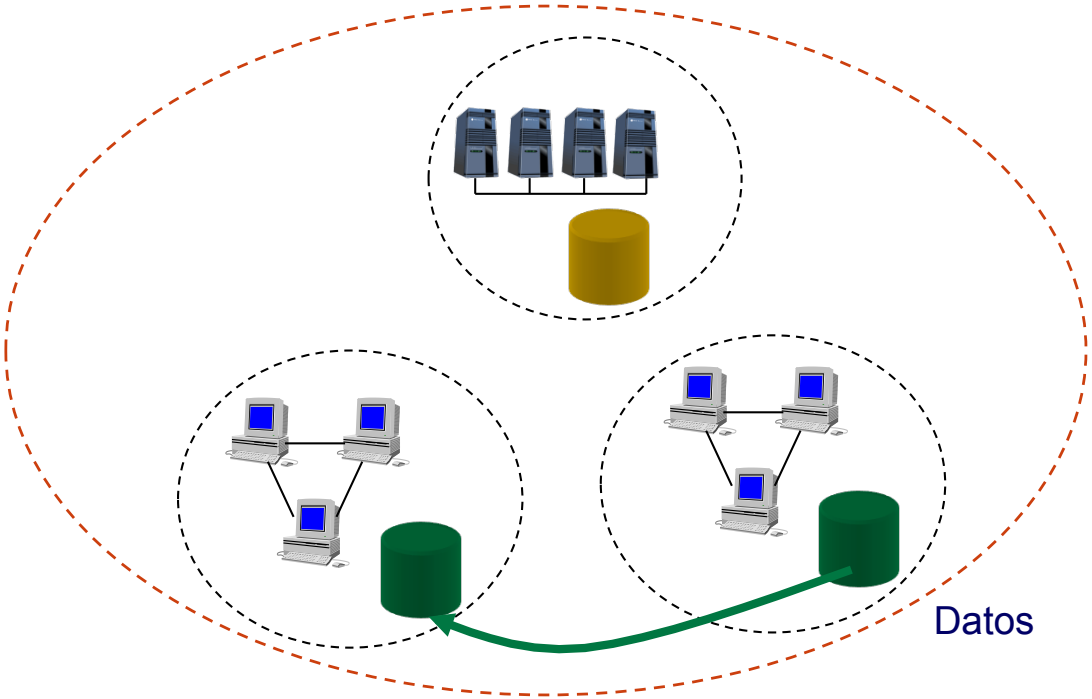


Modelo de ejecución II

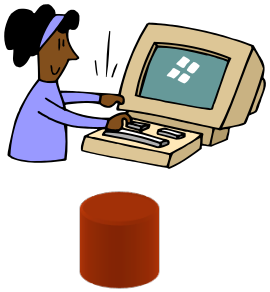


Gestor de recursos

Servicio de información

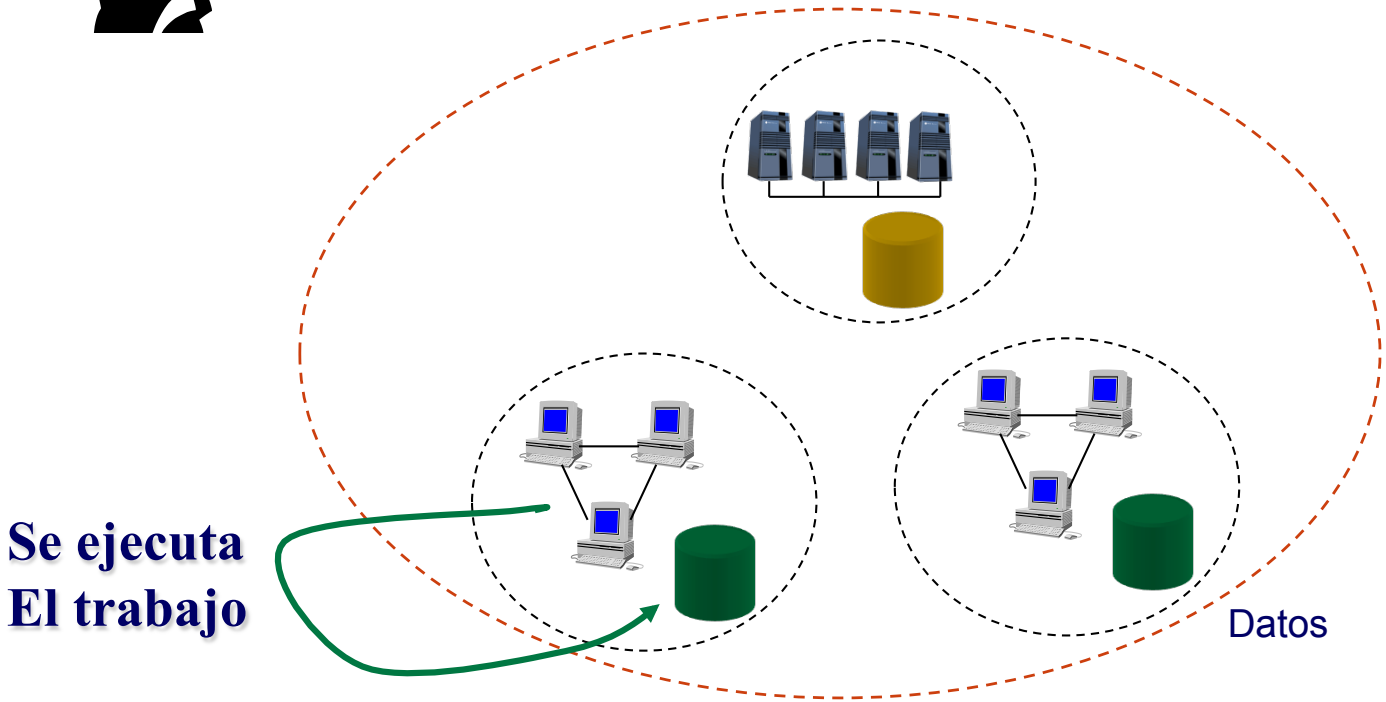


Modelo de ejecución II

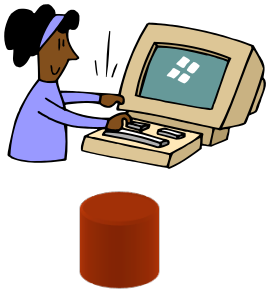


Gestor de recursos

Servicio de información

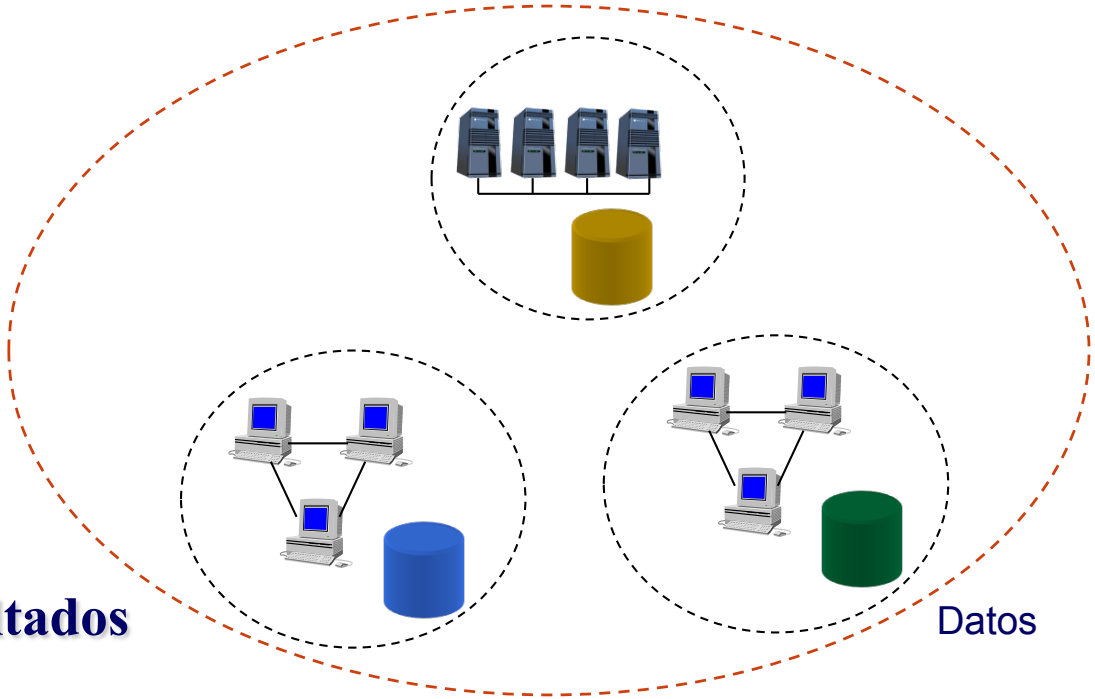


Modelo de ejecución II



Gestor de recursos

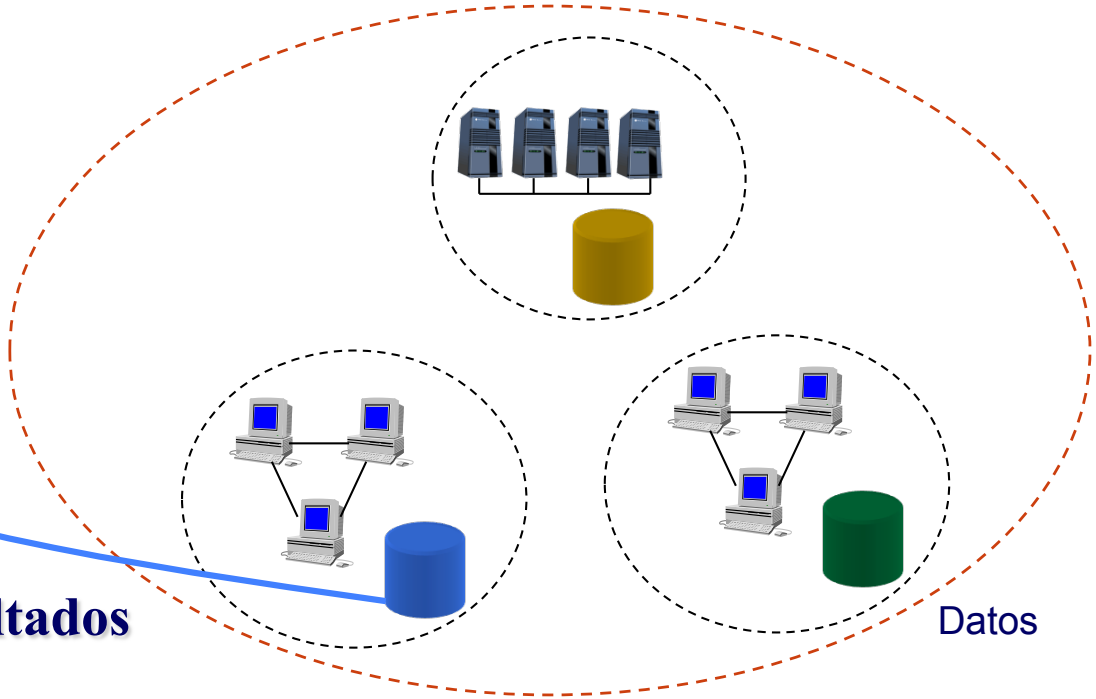
Servicio de información



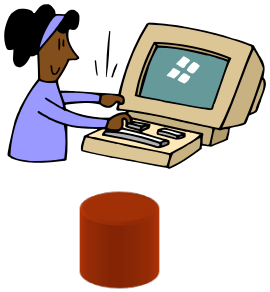
Modelo de ejecución II

Gestor de recursos

Servicio de información



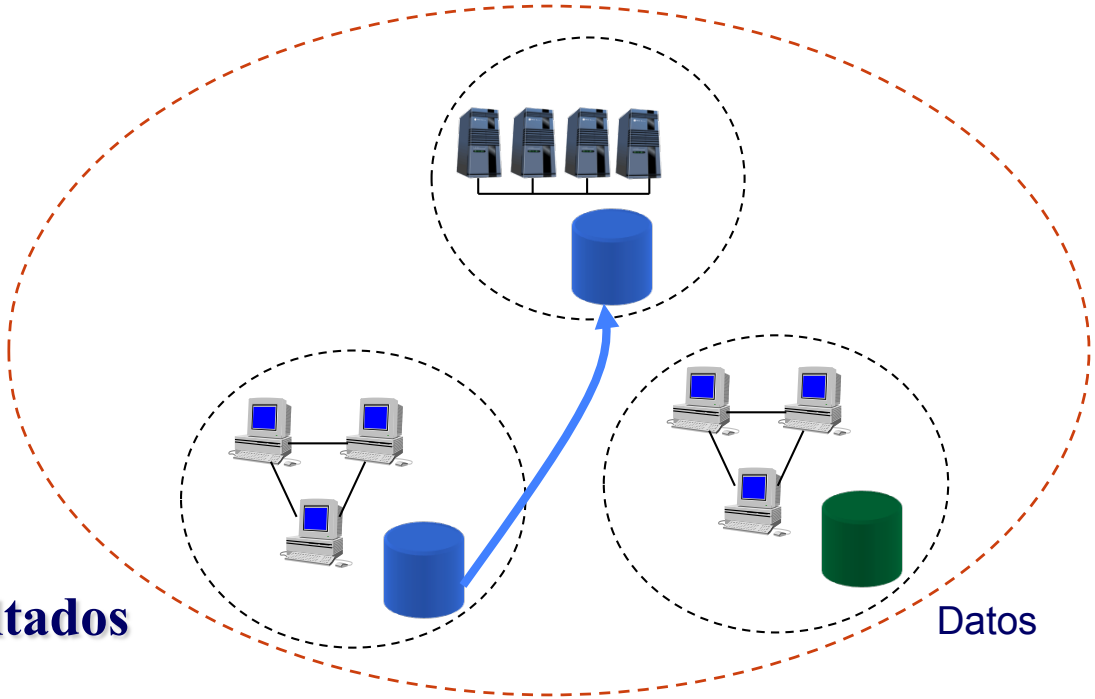
Modelo de ejecución II



Programa

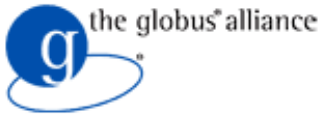
Gestor de recursos

Servicio de información



Globus toolkit

- Introduction to Globus Toolkit



- Components
  - Common runtime components
  - Security
  - Execution management
  - Data management
  - Information services

Globus Toolkit

- Software toolkit, developed by The Globus Alliance http://www.globus.org
- It can use to program grid-based applications.
- Includes *high-level services* that we can use to build Grid applications. It includes:
  - A job submission infrastructure
  - A security infrastructure
  - Data management services

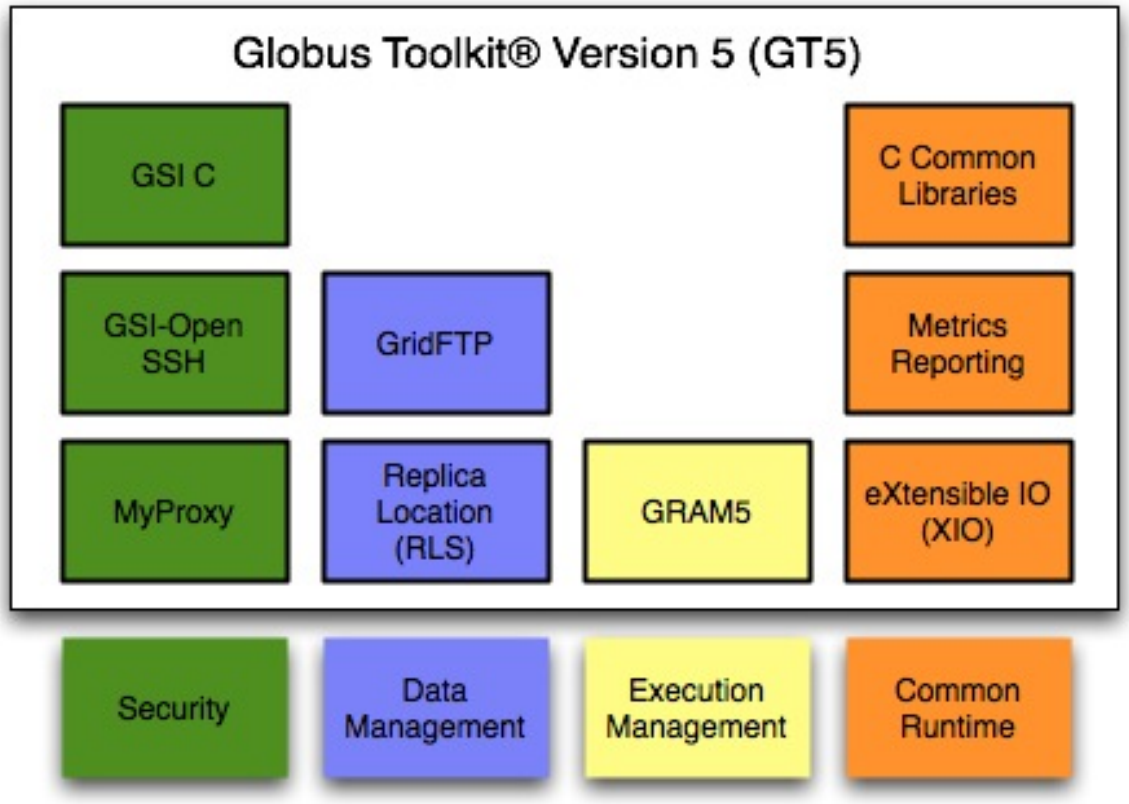
What is not the Globus toolkit?

- A user tool
- An application
- A scheduler

Stables versions

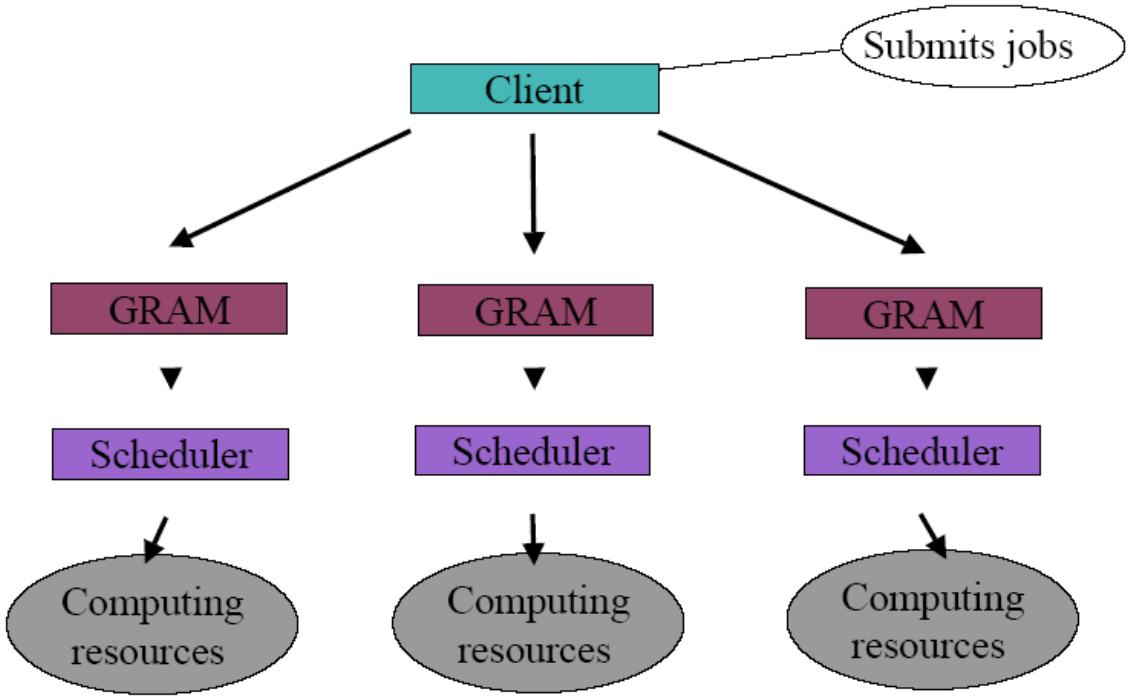
- Version v1.0.0 (10/29/1998)
- Version 1.1.1 (12/1999)
- Release 1.1.4 (9/2000)
- Version 2.0 (2001)
- Version 2.4 (2002)
- Version 3 (jun/2003)
- Version 4 (may/2005)
- Versión 5 (2010)
- Versión 5.2.4 (2013)
- Versión 6.0 (actual)

Globus Toolkit Components



Execution management

- Grid Resource Allocation and Management (GRAM)
- GRAM allows to *submit*, *monitor* and *cancel* job.
- GRAM is not a scheduler.
- Two implementations
  - WS GRAM
  - Pre-WS GRAM



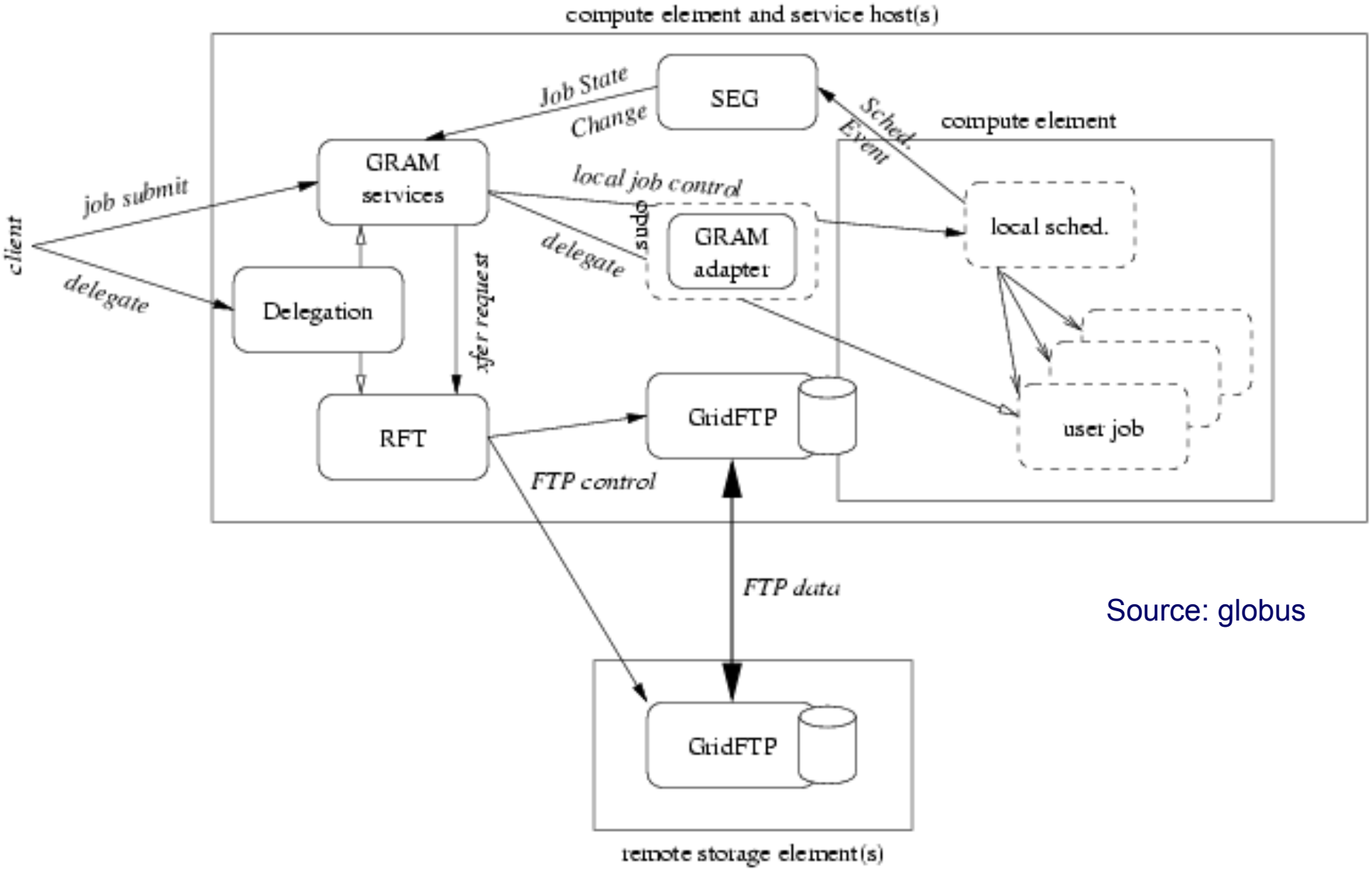
Job submission model

- Create and manage one job on a resource
- Submit and wait
  - Secure and reliable submission
- Not with an interactive TTY
  - File based stdin/out/err
  - Supported by all batch schedulers
- Monitoring
  - Simple query for current state
  - Asynchronous notifications to client

Components used by WS GRAM

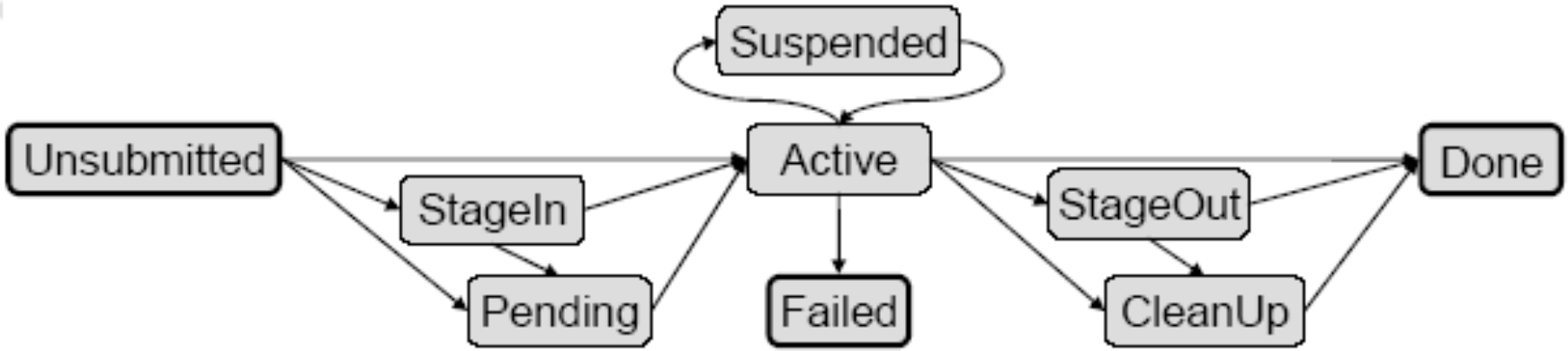
- Globus Components:
  - Reliable File Transfer (RFT)
  - GridFTP
  - Delegation service
    - Used by clients to delegate credentials into the correct hosting environment
- Internal components:
  - Scheduler Event Generator
    - Provides the job monitoring
- External Components:
  - Optional local job scheduler

GRAM Overview

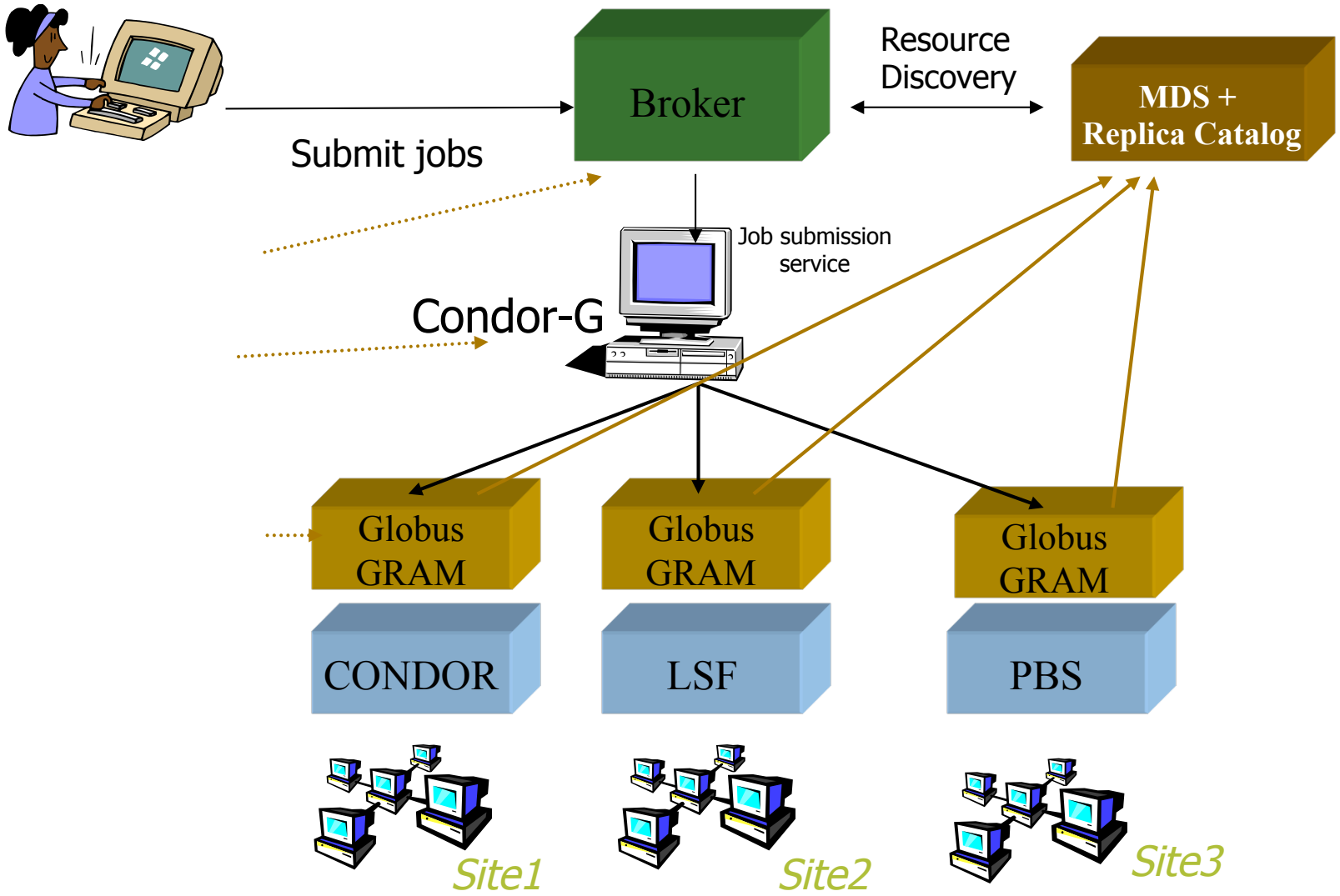


Source: globus

State transition diagram for GRAM jobs



MetaSchedulers



Data Management

- Data transfer
  - GridFTP
  - Reliable File Transfer (RFT) service

- Data Replication
  - Replica Location Service (RLS)

GridFTP

- GridFTP is a *high-performance*, *secure*, *reliable* data transfer protocol optimized for high-bandwidth wide-area networks
- Based on FTP protocol
  - GridFTP: Protocol Extensions to FTP for the Grid
  - Global Grid Forum Recommendation
  - http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf
- Globus provides
  - Server implementation
  - Client tools (command line programs)
  - Development libraries

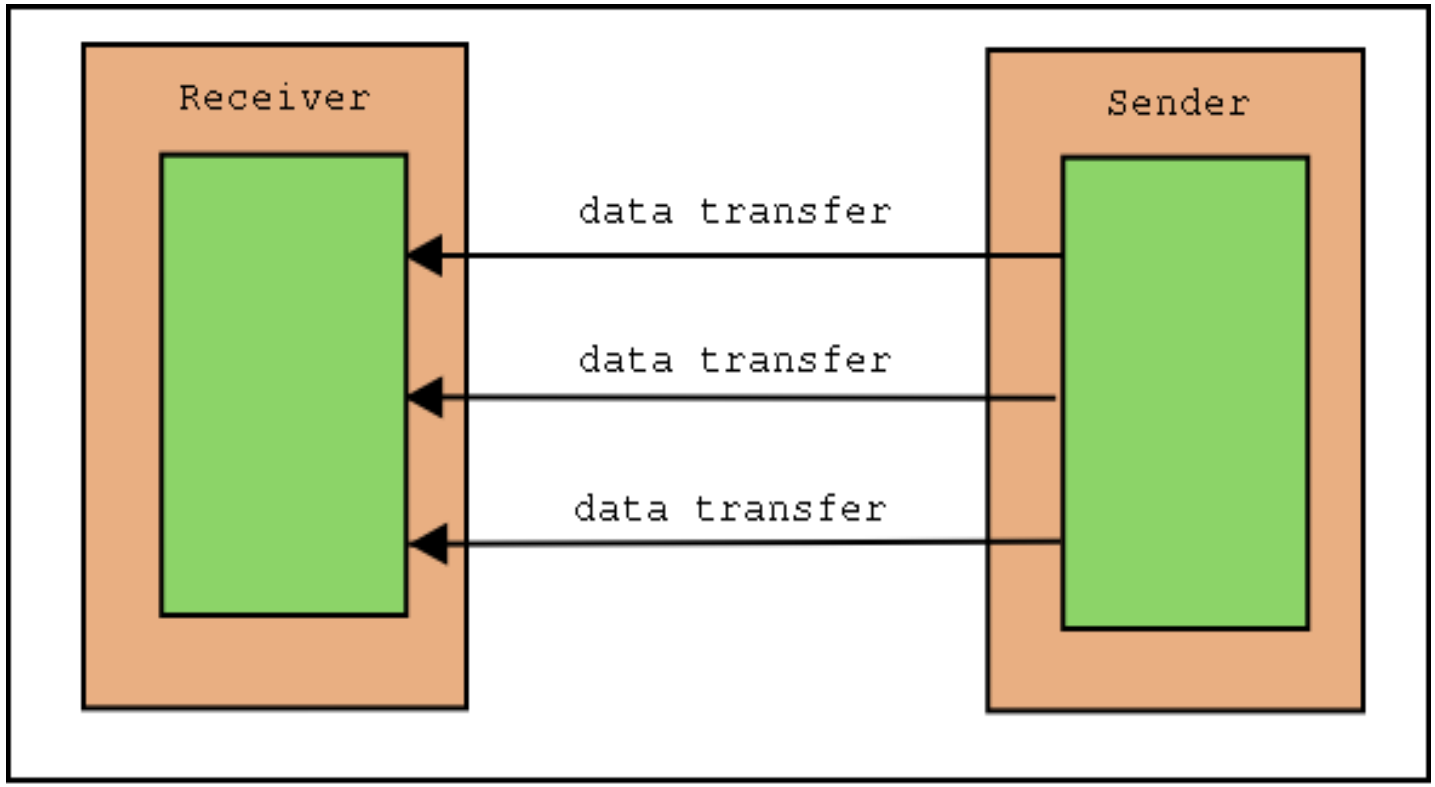
Features

- GSI Security
- Third-party transfers
- Partial file access
- Reliability/restart
- Large file support
- Data channel reuse
- Parallel transfers
- TCP Buffer size control
- Based on Standards
- Client side tool called *globus-url-copy*
- Development libraries and APIs

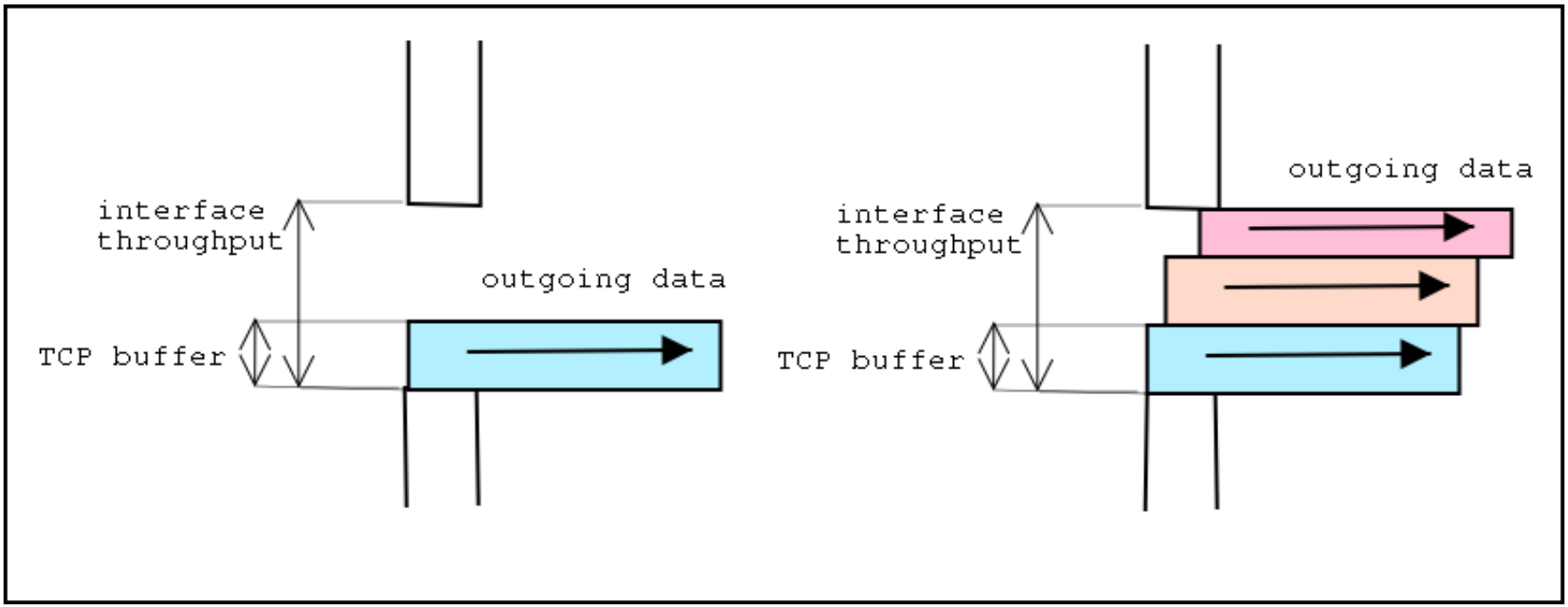
globus-url-copy

- Command line tool that can do multi-protocol data movement
- Mainly used for GridFTP, however, it supports many protocols
  - gsiftp:// (GridFTP)
  - ftp://
  - http://
  - https://
  - file://
- You must have a certificate to use it

Parallel transfers

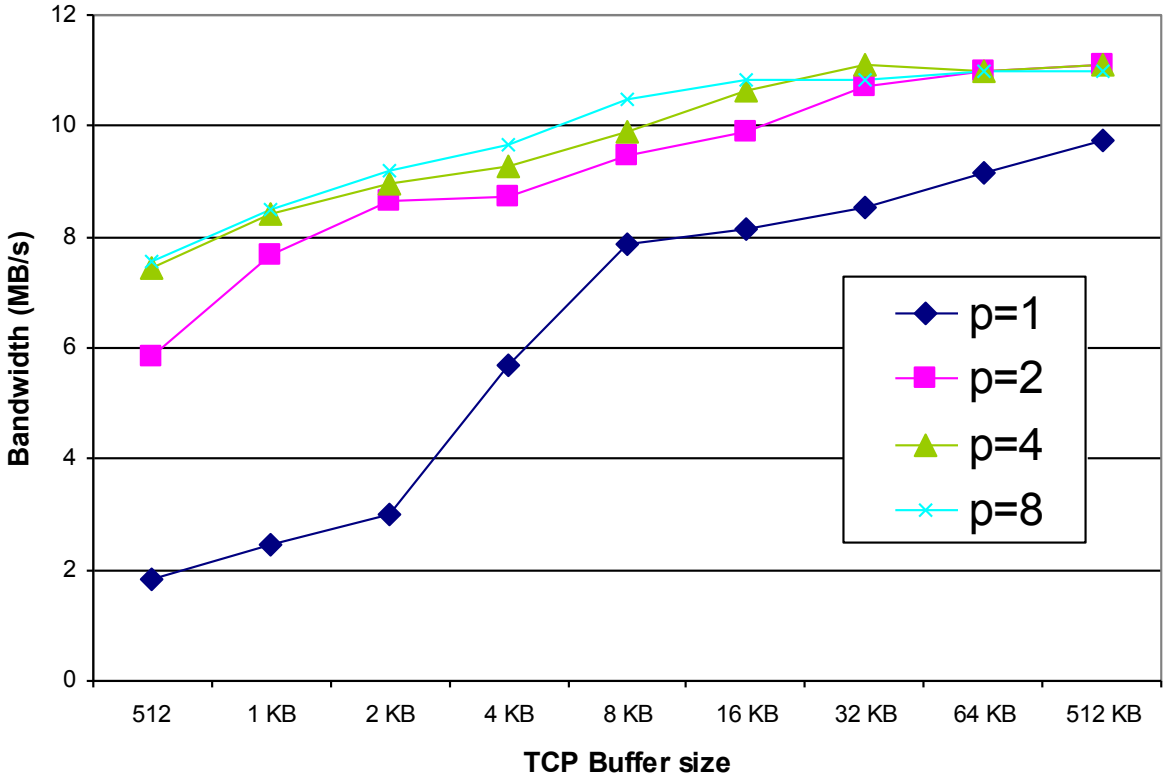


One stream versus multiples streams

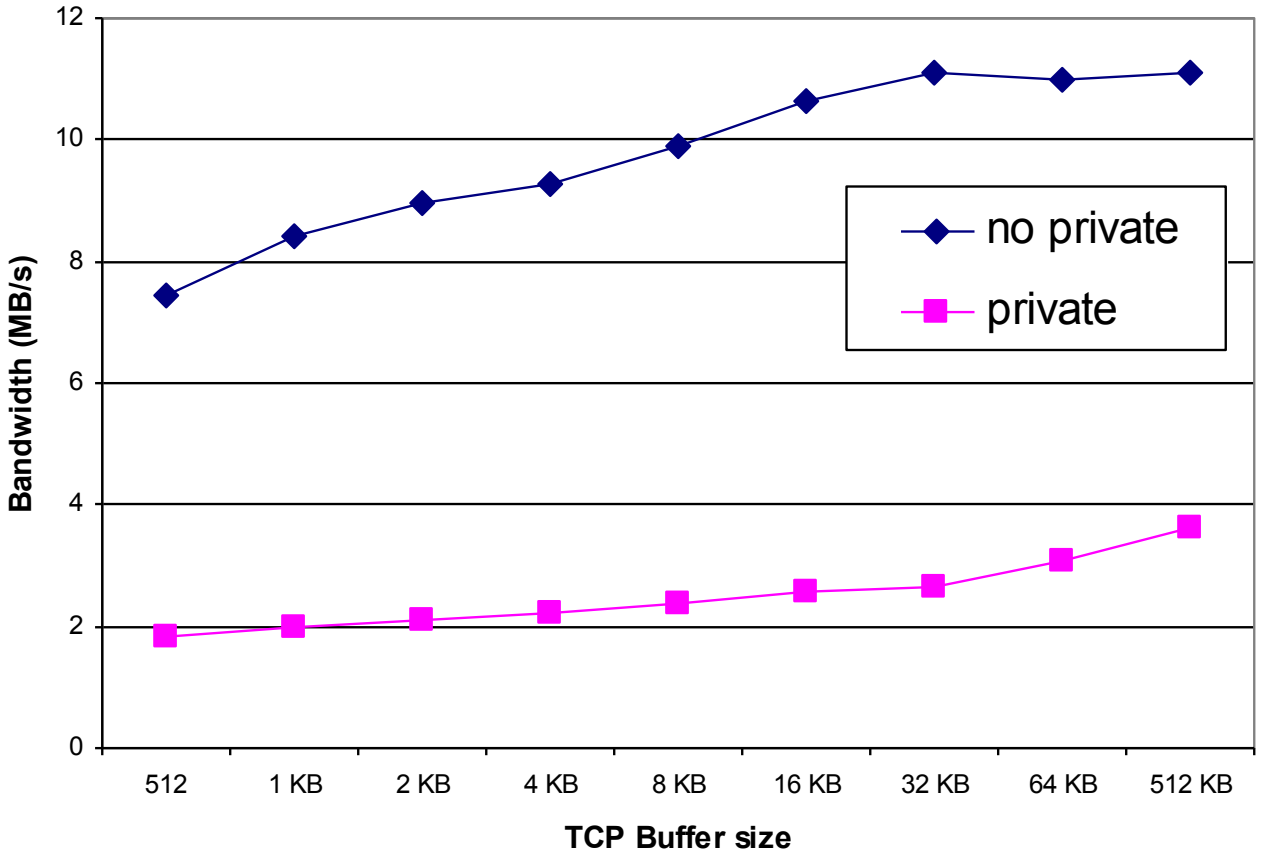


Choosing a value for the parallelism option

- There is not a good formula
- Using 4 streams is a very good rule of thumb



Performance: Private channel

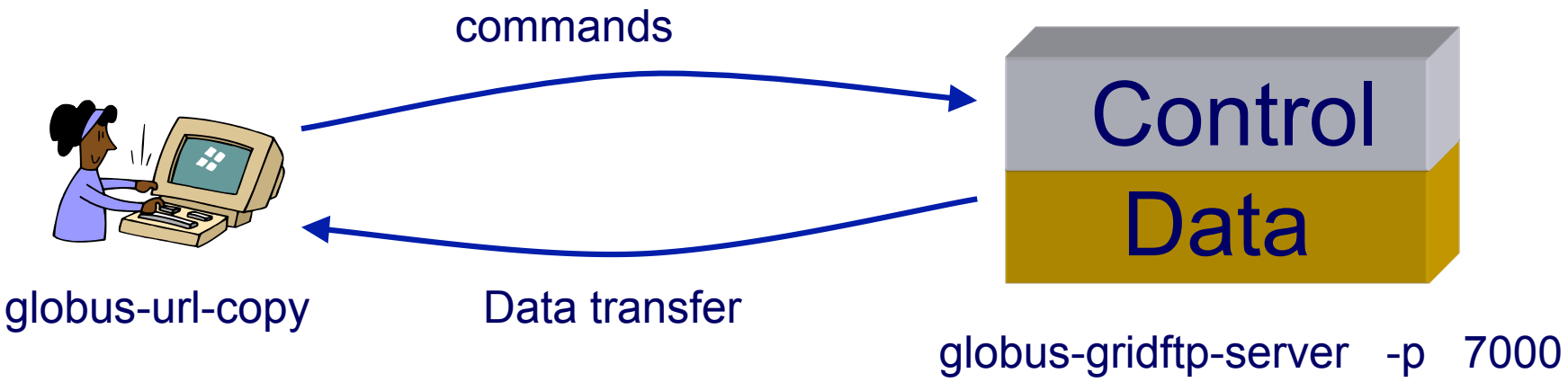


Parallelism=4

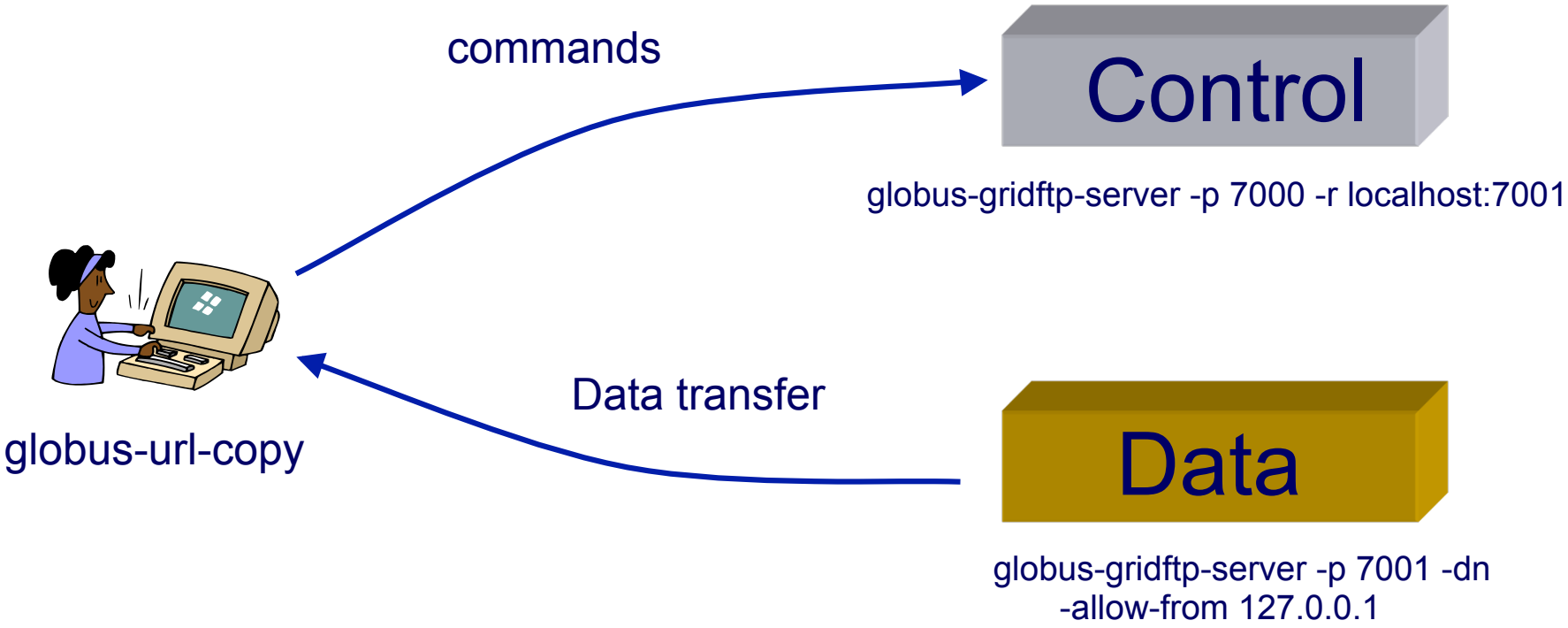
Server Architecture

- GridFTP use two separate socket connections:
  - A control channel for commands and responses
  - A data channel for data transfer
- Control and data channel can be in separate process
- A single control channel can have multiple data channels
  - Used in striped operation

Same Server for control and data

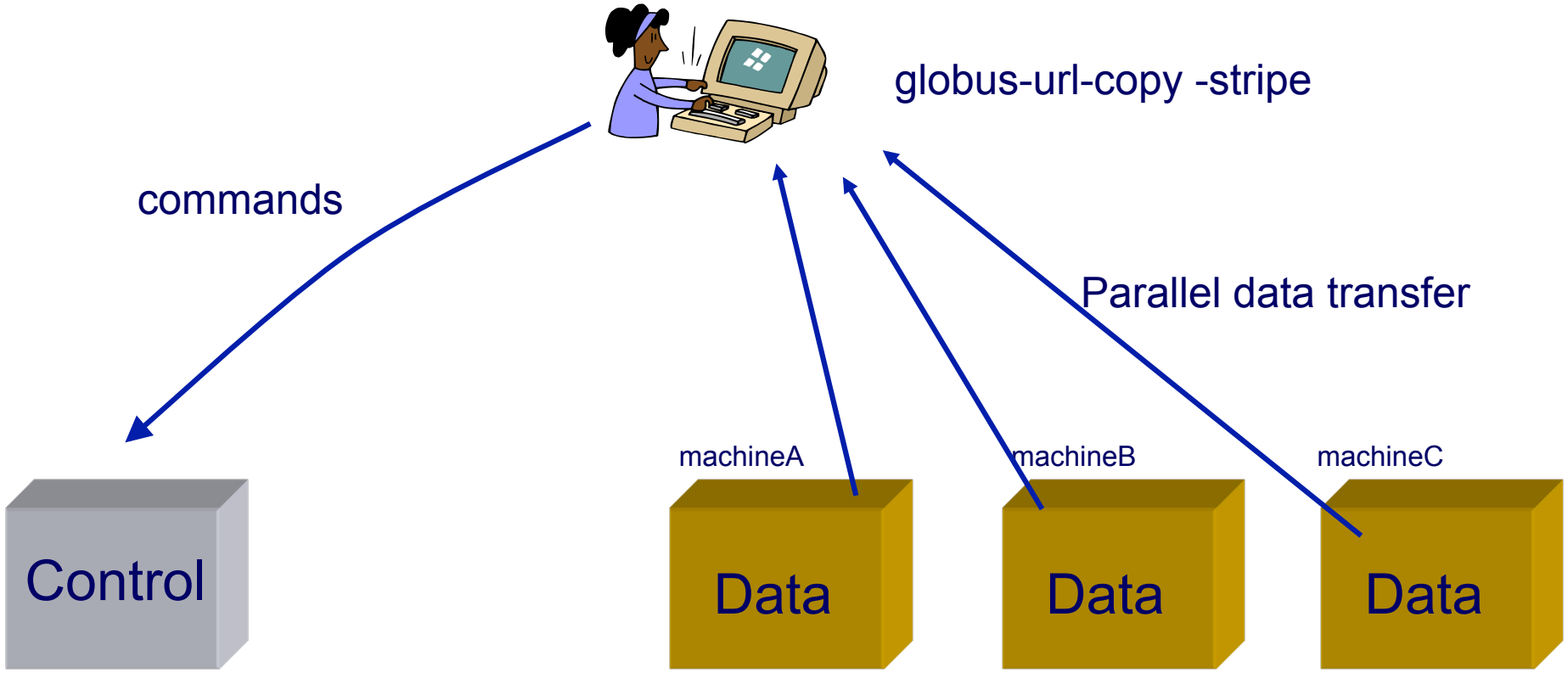


Different servers for control and data



- High level of security
  - The frontend can be run as any user
  - The backend is run as root, but configured to only allow connections from the frontend

Striped servers



globus-gridftp-server -p 7000
-r machineA:5000,machineB:6000,machineC4000

machineA> globus-gridftp-server -p 5000 –dn

machineB> globus-gridftp-server -p 6000 -dn

machineC> globus-gridftp-server -p 4000 -dn

Third party transfers

(1)

Machine A

FileA

(2)

(3)

Machine B

FileB

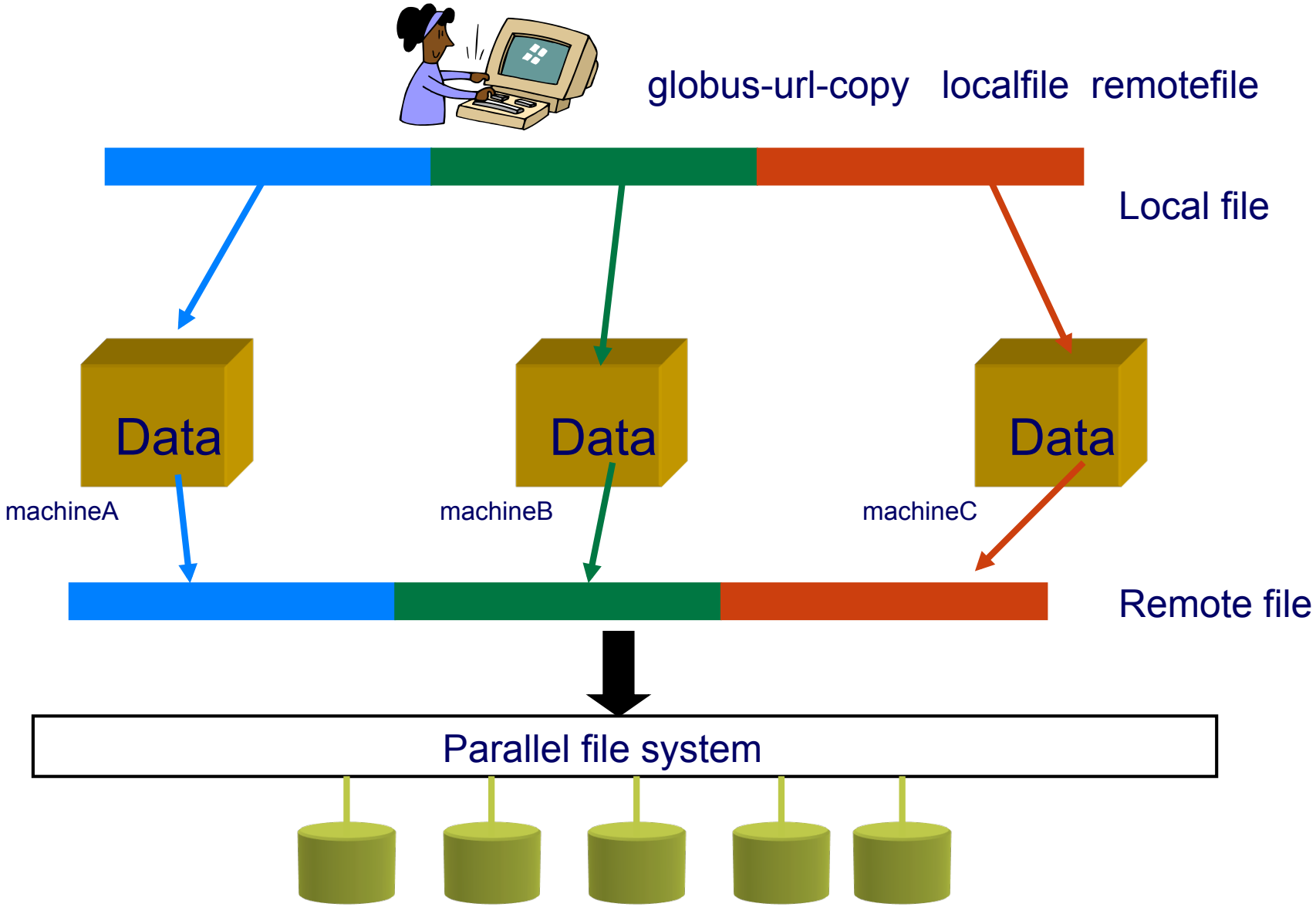
globus-url-copy
gsiftp://machineA/tmp/FileA
gsiftp://machineB/tmp/FileB

(1) Client send data requests to machine A
(2) Machine A init the transfer and client it disconnected
(3) Machine B receives the file

Striped servers

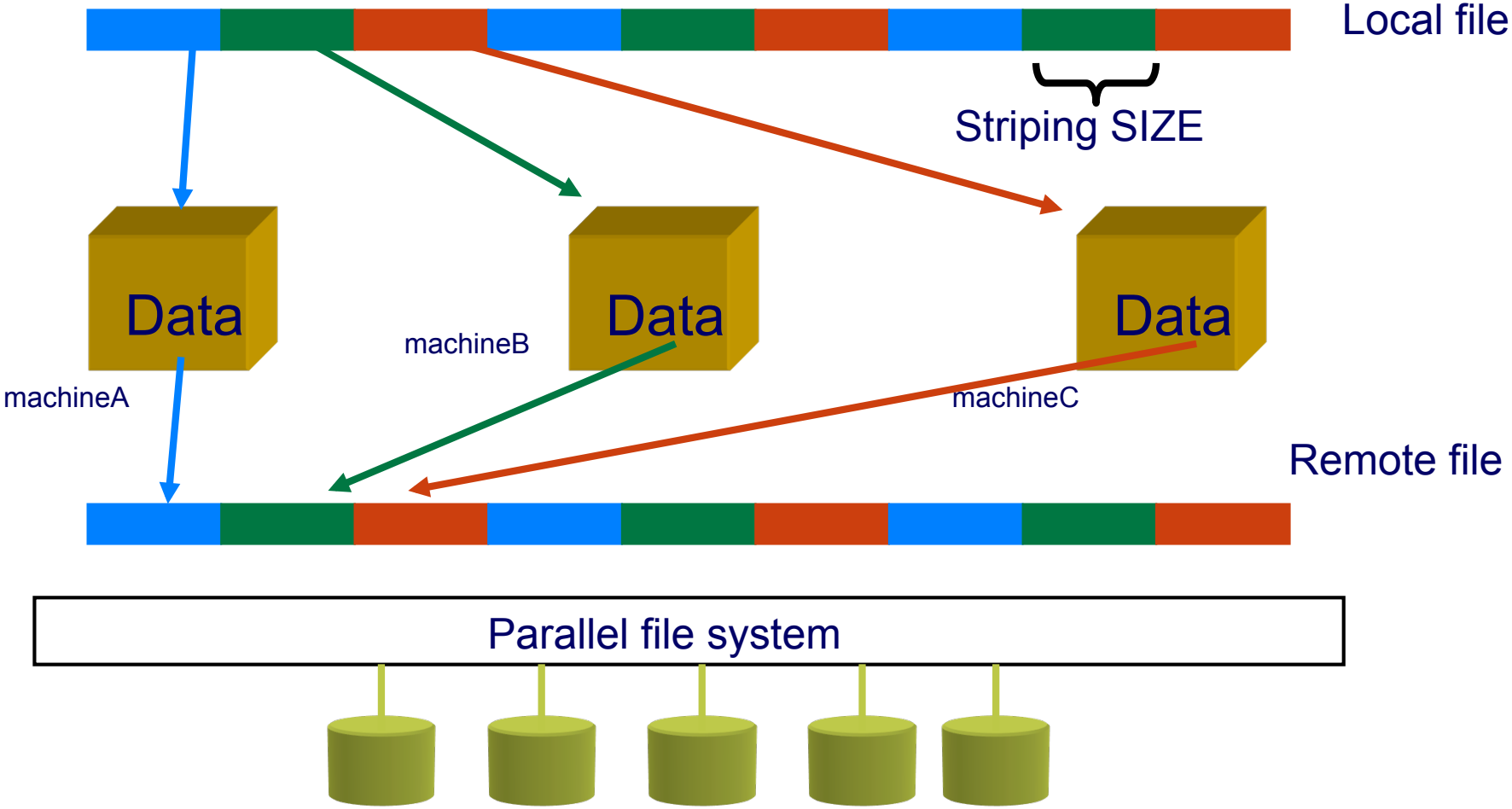
- Multiple nodes work together and act as a single GridFTP server
- All data nodes must see the same file system.
- Partitioned layout
  - **Partitioned**
    - File is divided into large sections and each stripe is given one of them
  - **Blocked**
    - File is divided into small portions that are dispatched to the stripes in round-robin fashion

Partitioned layout



Blocked layout

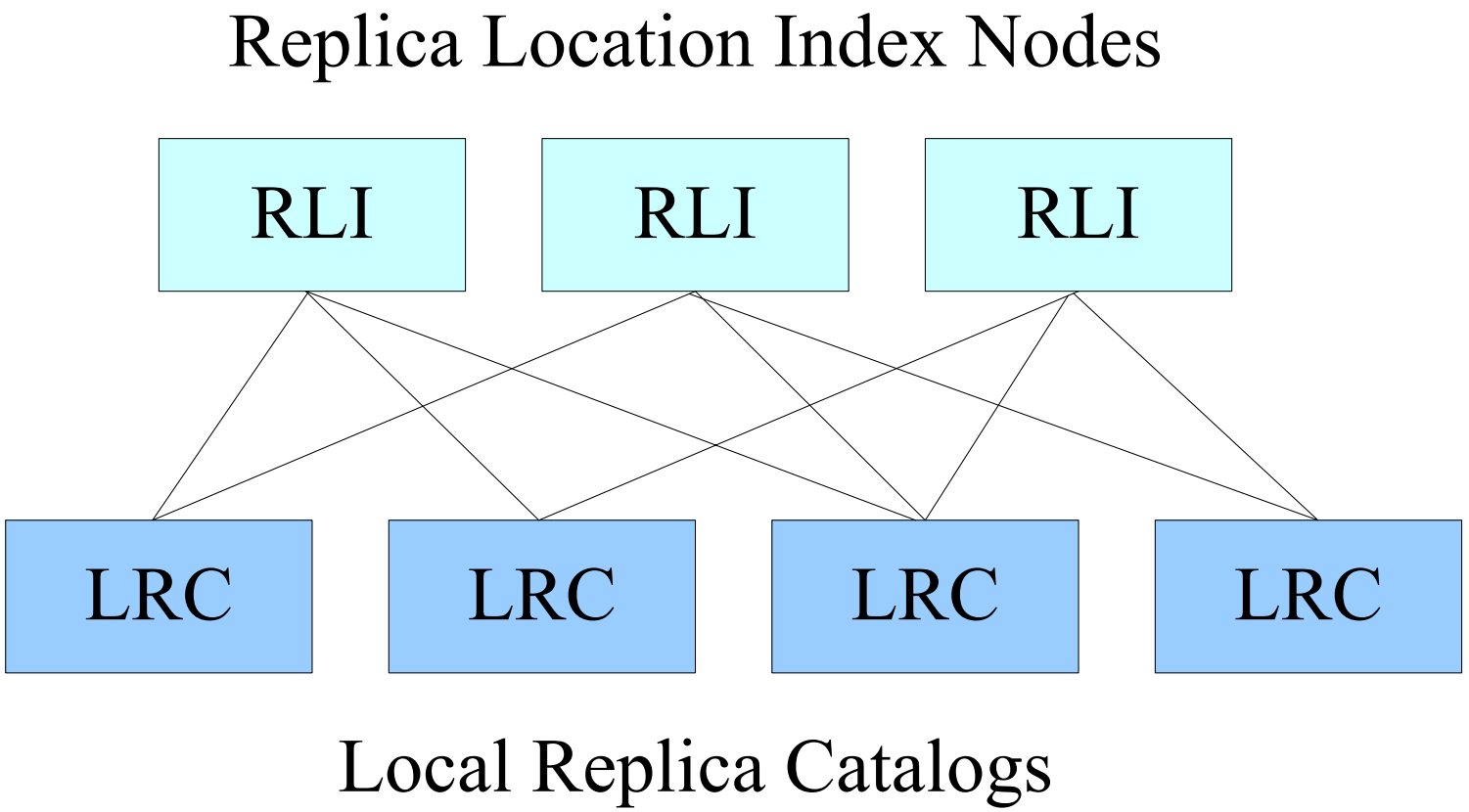
globus-url-copy -sbs SIZE localfile remotefile



Replica location service in Globus

- **The Replica Location Service (RLS)** is a distributed registry service that records the locations of data copies and allows discovery of replicas.
- Maintains mappings between logical identifiers and physical names.
- RLS was designed and implemented in a collaboration between the European DataGrid project and the Globus Alliance

RLS components



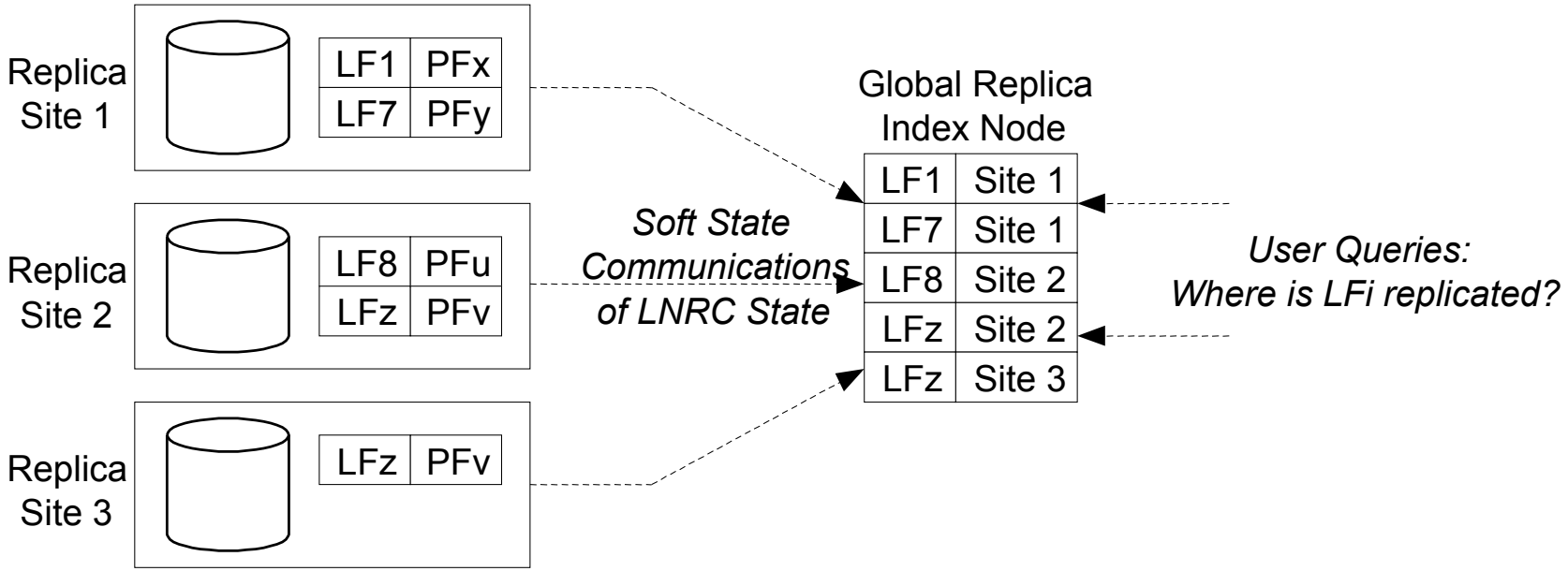
RLS components

- ***Local Replica Catalogs*** (LRCs) stores mappings between logical names for data items and the physical locations of replicas of those items.
- ***Replica Location Index*** (RLI) nodes aggregate information about one or more LRCs.
- LRCs use soft state update mechanisms to inform RLIs about their state: relaxed consistency of index.
- Optional compression of state updates reduces communication, CPU and storage overheads
- Membership service registers participating LRCs and RLIs and deals with changes in membership

RLS behavior

- Files are registered at LRC. LRC updates information to RLI.
- The replication service begins by locating the desired files, first by querying a Replica Location Index (RLI), which identifies the site or sites where the file may be present.
- Then by querying one or more Local Replica Catalogs (LRCs) to identify the exact physical location of the file.
- Finally, transfer de data using gridFTP or RFT.

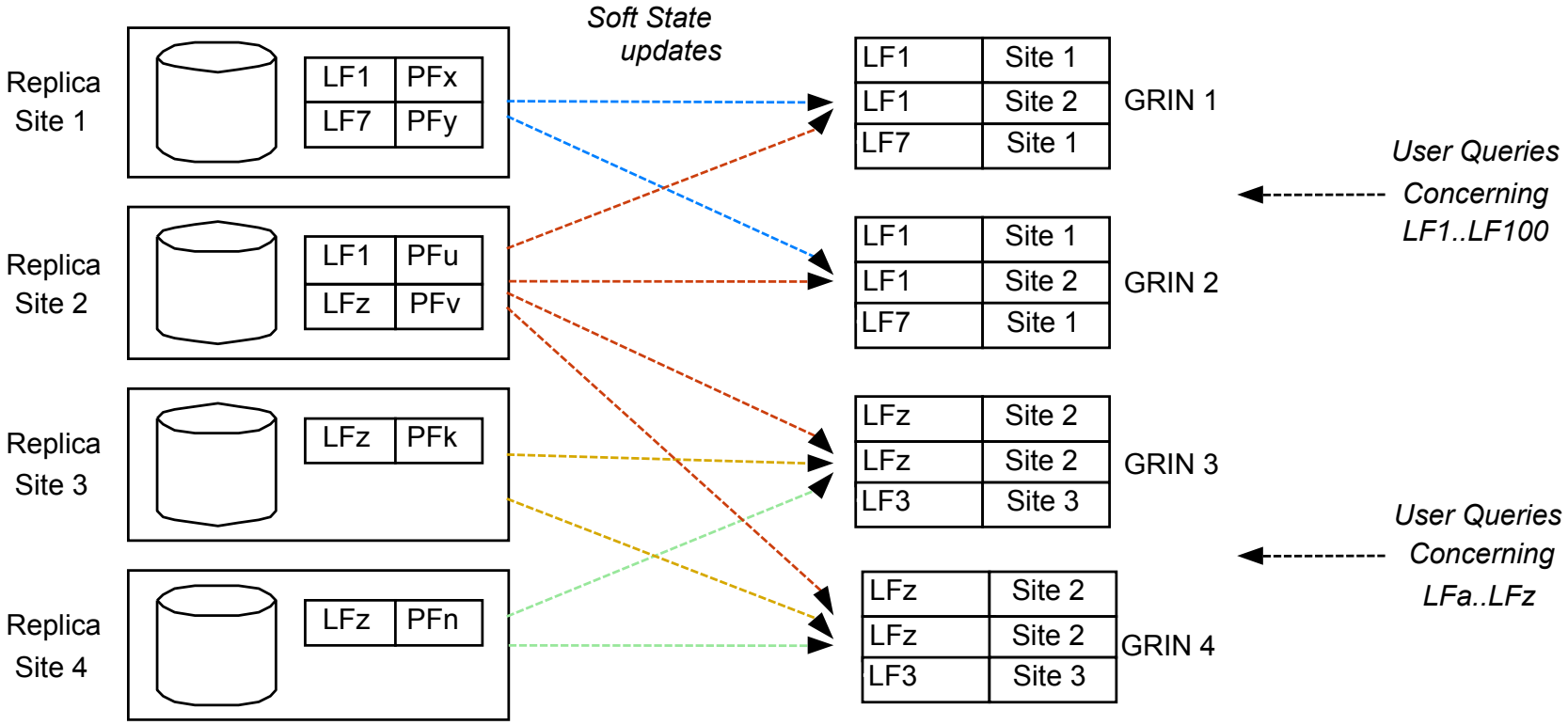
Example 1



Source: Globus Alliance

- All updates sent to a centralized GRIN
- Not scalable: All queries serviced by a single index
- Not reliable: Single point of failure

Example 2



Source: Globus Alliance

- RLS with LFN partitioning
- Replicated RLI
- More scalable, reliable

The RLS and Replica Consistency Services

- RLS is a simple tool that provides registration and discovery of files.
- RLS does not perform consistency tasks
- RLS relies on higher-level consistency services to perform these functions.
- Reasons for this design choice:
  - Provides a fast service avoiding additional overheads
  - Users may have different definitions of what constitutes a “valid” replica