

# Execution Management: Key Concepts

DRAFT

---

## Execution Management: Key Concepts

### Overview

The Globus Toolkit provides both a suite of web services and a "pre-web services" Unix server suite to submit, monitor, and cancel jobs on Grid computing resources. Both systems are known under the moniker "GRAM", while "GRAM4" refers only to the web service implementation. Jobs are computational tasks that may perform input/output operations while running which affect the state of the computational resource and its associated file systems. In practice, such jobs may require coordinated staging of data into the resource prior to job execution and out of the resource following execution. Some users, particularly interactive ones, benefit from accessing output data files as the job is running. Monitoring consists of querying and subscribing for status information such as job state changes.

Grid computing resources are typically operated under the control of a scheduler which implements allocation and prioritization policies while optimizing the execution of all submitted jobs for efficiency and performance. GRAM is not a resource scheduler, but rather a protocol engine for communicating with a range of different local resource schedulers using a standard message format.

For more detailed information about the concepts behind the software implementation, see [Chapter 2, GT 4.2.0 GRAM4 Approach](#).

---

# Table of Contents

- 1. Conceptual details ..... 1
  - 1. Targeted job types ..... 1
  - 2. Component architecture ..... 1
  - 3. Security ..... 2
  - 4. Job Management ..... 2
  - 5. Data Management ..... 2
  - 6. Task coordination ..... 3
- 2. GRAM4 Approach ..... 4
  - 1. Introduction ..... 4
  - 2. Component architecture approach ..... 4
  - 3. GRAM4 Protocol ..... 7
  - 4. Security model ..... 15
  - 5. Audit ..... 16
  - 6. Job Description Language ..... 16
  - 7. Persistence Data ..... 16
  - 8. Job Lifetime ..... 17
  - 9. GRAM4 software architecture ..... 18

DRAFT

## List of Figures

2.1. GRAM4 Components .....	5
2.2. Minimal Protocol Sequence .....	11
2.3. GRAM4 Components .....	12
2.4. Non-staging Delegation Sequence .....	13
2.5. Staging Sequence .....	14
2.6. Staging Sequence with Hold .....	15
2.7. GRAM4 Software .....	18
2.8. GRAM4 Lifecycle .....	19

DRAFT

# Chapter 1. Conceptual details

A number of concepts underly the purpose and motivation for GRAM. These concepts are divided into broad categories below.

## 1. Targeted job types

GRAM is meant to address a range of jobs where arbitrary programs, reliable operation, stateful monitoring, credential management, and file staging are important. GRAM is not meant to serve as a "remote procedure call" (RPC) interface for applications not requiring many of these features, and furthermore its interface model and implementation may be too costly for such uses. The GRAM4 service will become cheaper over time as the underlying web service technologies improve, but as with the older pre-web service GRAM, its protocols will always involve multiple round-trips to support these advanced features that are not required for simple RPC applications.

The underlying WSRF core environment used for GRAM4 should also be considered as a direct application-hosting solution for lightweight, shared applications. In other words, if an application requires only modest input and output data transport in a stateless manner (all that matters is the result data or fault signal) and will be invoked many times, it may be a good candidate for exposure as an application-specific Web service.

## 2. Component architecture

Rather than consisting of a monolithic solution, GRAM is based on a component architecture at both the protocol and software implementation levels. This component approach serves as an ideal which shapes the implementation as well as the abstract design and features.

GRAM service suite	<p>Job management with GRAM makes use of multiple types of service:</p> <ul style="list-style-type: none"> <li>• Job management services represent, monitor, and control the overall job life cycle. These services are the job-management specific software provided by the GRAM solution.</li> <li>• File transfer services support staging of files into and out of compute resources. GRAM makes use of these existing services rather than providing redundant solutions; GRAM4 has further refactored some file transfer mechanisms present in pre-web service GRAM.</li> <li>• Credential management services are used to control the delegation of rights among distributed elements of the GRAM architecture based on users' application requirements. Again, GRAM makes use of more general infrastructure rather than providing a redundant solution, and GRAM4 has continued this refactoring to better separate credential management at the protocol level.</li> </ul>
Service model	<p>For GRAM4, the Globus Toolkit software development environment, and particularly WSRF core, is used to implement distributed communications and service state. For pre-web service GRAM, the "gatekeeper" daemon and GSI library are used for communications and service dispatch.</p>
Scheduler adapters	<p>A scripted plug-in architecture is provided by GRAM to enable extension with adapters to control a variety of local schedulers.</p>

### 3. Security

Secure operation	GRAM4 utilizes WSRF functionality to provide for authentication of job management requests as well as to protect job requests from malicious interference, while pre-web service GRAM uses GSI and secure sockets directly. The use of GRAM does not reduce the ability for system administrators to control access to their computing resources. The use of GRAM also does not reduce the safety of jobs users run on a given computing resource.
Local system protection domains	To protect users from each other, jobs are executed in appropriate local security contexts, e.g. under specific Unix user IDs based on details of the job request and authorization policies. Additionally, GRAM mechanisms used to interact with the local resource are design to minimize the privileges required and to minimize the risks of service malfunction or compromise.
Credential delegation and management	A client may delegate some of its rights to GRAM services in order to facilitate the above functions, e.g. rights for GRAM to access data on a remote storage element as part of the job execution. Additionally, the client may delegate rights for use by the job process itself. With pre-web service GRAM, these two uses of rights are inseparable, while GRAM4 provides separate control for each purpose (while still allowing rights to be shared if that is desired).
Audit	To assist with normal accounting functions as well as to further mitigate risks from abuse or malfunction, GRAM uses a range of audit and logging techniques to record a history of job submissions and critical system operations.

### 4. Job Management

Reliable job submission	GRAM4 provides an "at most once" job submission semantics. A client is able to check for and possibly resubmit jobs, in order to account for transient communication errors without risk of running more than one copy of the job. Similarly, pre-web service GRAM provides a two-phase submission mechanism to submit and then commit a job to be run.
Job cancellation	While many jobs are allowed to run to their natural completion, GRAM provides a mechanism for clients to cancel (abort) their jobs at any point in the job life cycle.

### 5. Data Management

Reliable data staging	GRAM4 provides for reliable, high-performance transfers of files between the compute resource and external (gridftp) data storage elements before and after the job execution. Pre-web service GRAM can also stage with gridftp systems but with less flexible reliable-transfer logic driving its requests.
Output monitoring	GRAM supports a mechanism for incrementally transferring output file contents from the computation resource while the job is running. GRAM4 uses a new mechanism to allow arbitrary numbers of files to be transferred in this fashion, while pre-web service GRAM only supports incremental transfer of the job's standard output and error streams.

## 6. Task coordination

Parallel jobs	Some jobs are parallel, meaning that they consist of more than one simultaneous tasks. These tasks are often hosted on parallel computing hardware to provide increased computational throughput.
Task rendezvous	<p>Some parallel programming environments, such as MPI, provide mechanisms for all tasks in a parallel computation to find out about each other: to know the number of peer tasks as well as possibly to exchange some information between tasks. Native parallel programming models often support this within the local job start mechanism.</p> <p>GRAM provides a mechanism for task rendezvous which job applications may use if they do not have another more appropriate solution. This mechanism may be used directly by application code or by intervening middleware libraries, e.g. libraries that are designed to present a simplified programming model to applications run via GRAM. GRAM does not require tasks to be coordinated, but addresses this requirement in order to facilitate a wider range of applications. The protocols and APIs used to support task rendezvous are different for GRAM4 and GRAM2.</p>

---

# Chapter 2. GT 4.2.0 GRAM4 Approach

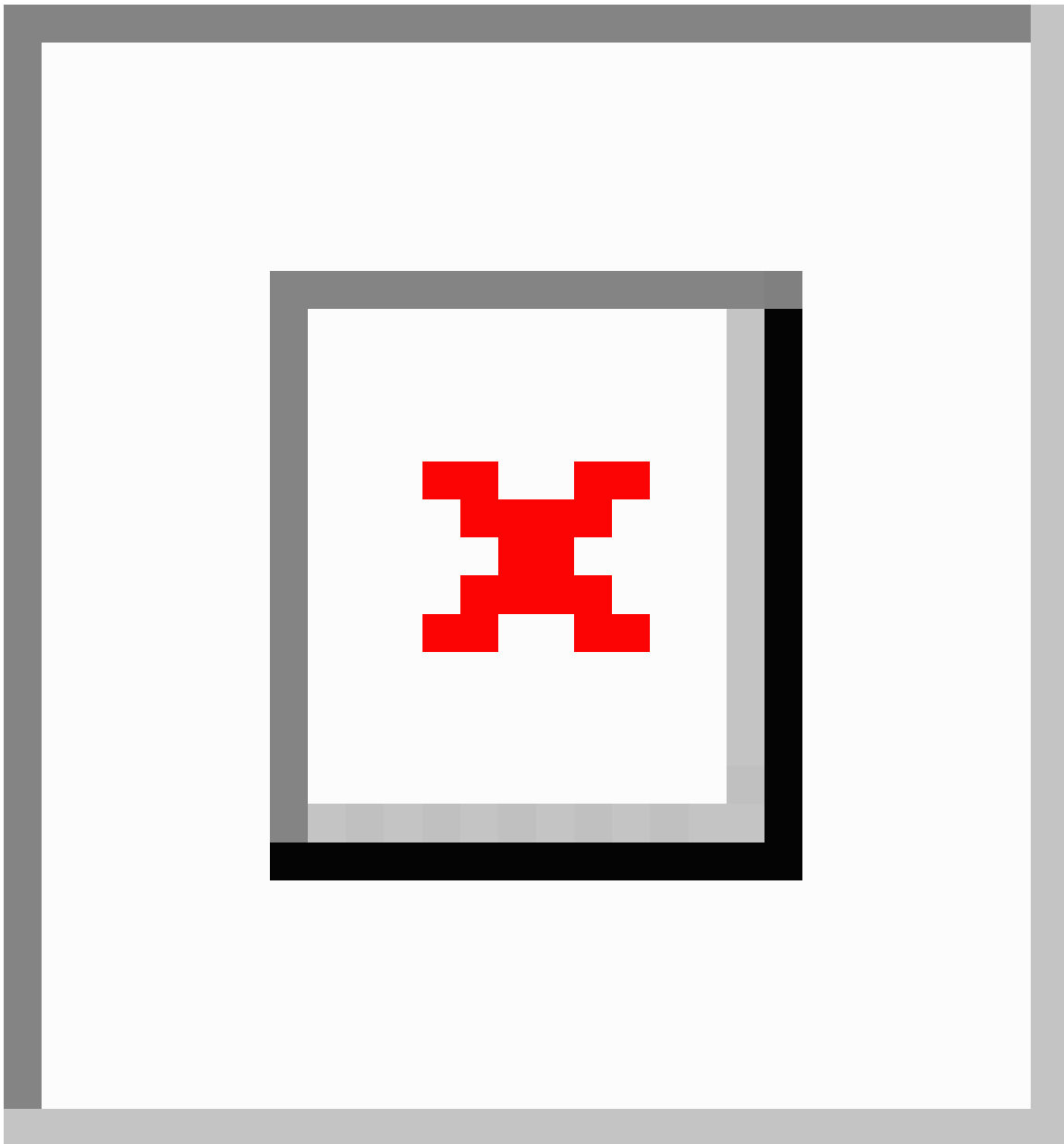
## 1. Introduction

The GRAM4 software implements a solution to the job-management problem described in [Key Concepts](#), providing Web services interfaces consistent with the WSRF model. This solution is specific to operating systems following the Unix programming and security model.

## 2. Component architecture approach

GRAM4 combines job-management services and local system adapters with other service components of GT 4.2.0 in order to support job execution with coordinated file staging.

DRAFT

**Figure 2.1. GRAM4 Components**

## 2.1. GRAM4

The heart of the GRAM4 service architecture is a set of Web services designed to be hosted in the Globus Toolkit's WSRF core hosting environment. Note, these services, described below, make use of platform-specific callouts to other software components described in the next section.

ManagedJob	Each submitted job is exposed as a distinct resource qualifying the generic ManagedJob service. The service provides an interface to monitor the status of the job or to terminate the job (by terminating the ManagedJob resource). The behavior of the service, i.e. the local scheduler adapter implementation, is selected by the specialized <i>type</i> of the resource.
------------	--

ManagedJobFactory	Each compute element, as accessed through a local scheduler, is exposed as a distinct resource qualifying the generic ManagedJobFactory service. The service provides an interface to create ManagedJob resources of the appropriate type in order to perform a job in that local scheduler.
-------------------	--

## 2.2. Globus Toolkit Components used by GRAM4

### 2.2.1. ReliableFileTransfer

The ReliableFileTransfer (RFT) service of GT 4.2.0 is invoked by the GRAM4 services to effect file staging before and after job computations.

The integration with RFT provides a much more robust file staging manager than the ad-hoc solution present in previous versions of the GRAM job manager logic. RFT has better support for retry, restart, and fine-grained control of these capabilities. GRAM4 exposes the full flexibility of the RFT request language in the job staging clauses of the job submission language.

### 2.2.2. GridFTP

GridFTP servers are required to access remote storage elements as well as file systems accessible to the local compute elements that may host the job. The ReliableFileTransfer Web service acts as a so-called third-party client to the GridFTP servers in order to manage transfers of data between remote storage elements and the compute element file systems. It is not necessary that GridFTP be deployed on the same host/node as the GRAM4 services, but staging will only be possible to the subset of file systems that are shared by the GridFTP server that is registered with the GRAM4 service. (REF TO DEPLOY/CONFIG HERE) If no such server or shared file systems are registered, staging is disallowed to that GRAM4 compute element.

GridFTP is also used to monitor the contents of files written by the job during job execution. The standard GridFTP protocol is used by a slightly unusual client to efficiently and reliably check the status of files and incrementally fetch new content as the file grows. This method supports "streaming" of file content from any file accessible by GridFTP, rather than only the standard output and error files named in the job request--the limitation of previous GRAM solutions. This approach also simplifies failover and restart of streaming to multiple clients.

The integration with GridFTP replaces the legacy GASS (Globus Access to Secondary Storage) data transfer protocol. This changeover is advantageous both for greater performance and reliability of data staging as well as to remove redundant software from the GRAM codebase.

### 2.2.3. Delegation

The Delegation service of GT 4.2.0 is used by clients to delegate credentials into the correct hosting environment for use by GRAM4 or RFT services.

The integration of the Delegation service replaces the implicit, binding-level delegation of previous GRAM solutions with explicit service operations. This change not only reduces the requirements on client tooling for interoperability purposes, but also allows a new separation of the life cycle of jobs and delegated credentials. Credentials can now be shared between multiple short-lived jobs or eliminated entirely on an application-by-application basis to make desired performance and security tradeoffs. Meanwhile, for unique situations GRAM4 retains the ability to refresh credentials for long-lived jobs and gains an ability to designate different delegated credentials for different parts of the job's life cycle.

## 2.3. External Components used by GRAM4

### 2.3.1. Local job scheduler

An optional local job scheduler is required in order to manage the resources of the compute element. GRAM4 has the ability to spawn simple time-sharing jobs using standard Unix `fork()` methods, but most large-scale compute elements will be under the control of a scheduler such as PBS, LSF, Loadleveler, etc.

### 2.3.2. sudo

The de facto standard Unix `sudo` utility is used by GRAM4 to gain access to target user accounts without requiring GRAM4 to have general super-user privilege on the system. The `sudo` command is used to execute GRAM4 adapter tools in the user account context; these adapters perform the local system-specific operations needed to initialize and run user jobs.

The `sudo` utility not only provides a simple way for GRAM4 to run programs as other users without "root" privilege, but it provides fine-grained controls for the system administrator to restrict which user accounts are valid GRAM4 targets as well as secure auditing of all operations requested by GRAM4. This mechanism replaces the root-privileged Gatekeeper component of the GRAM2 service in order to avoid running an entire WSRF hosting environment as root. This change provides enhanced security, at the expense of slightly more complicated deployment effort, and is motivated by the relative increase in the size of the GRAM4 and WSRF codebase as compared to the traditional Gatekeeper.

## 2.4. Internal Components used by GRAM4

### 2.4.1. Scheduler Event Generator

The [Scheduler Event Generator](#)<sup>1</sup> component program provides the job monitoring capability for the GRAM4 service. Plugin modules provide an interface between the Scheduler Event Generator and local schedulers.

### 2.4.2. Fork Starter

The [Fork Starter](#)<sup>2</sup> program starts and monitors job processes for GRAM4 services which do not a local scheduler. The starter executes the user application and then waits for it to terminate. It records the start time, termination time, and exit status of each process it starts. This information is used by a Scheduler Event Generator plugin for triggering job state changes.

## 3. GRAM4 Protocol

### 3.1. Major Protocol Steps

The components in the GRAM4 solution are organized to support a range of optional features that together address different usage scenarios. These scenarios are explored in depth in terms of protocol exchanges in the Protocol Variations section. However, at a high level we can consider the main client activities around a GRAM4 job to be a partially ordered sequence.

---

<sup>1</sup> [../gram4/developer/internal-components.html#seg](#)

<sup>2</sup> [../gram4/developer/internal-components.html#forkstarter](#)

### 3.1.1. Creation of a Job

The main component of the GRAM4 service model is the ManagedJob resource created by a ManagedJobFactory::createManagedJob invocation. A meaningful GRAM4 client MUST create a job that will then go through a life cycle where it eventually completes execution and the resource is eventually destroyed (the core black-and-white nodes in the high-level picture).

### 3.1.2. Optional Staging Credentials

Optionally, the client MAY request staging activities to occur before or after the job. If these are requested in the create call, suitable delegated credential EPRs MUST be passed in as part of the creation input, meaning that delegation operations MUST be performed sometime before createManagedJob when staging is enabled (the light-blue delegation nodes in the high-level picture). Two credential fields must be initialized: the staging and transfer credentials, which may refer to distinct credentials or may both refer to the same credential. The staging credential gives GRAM4 the right to interact with the RFT service, while the transfer credential gives RFT the right to interact with GridFTP servers.

### 3.1.3. Optional Job Credential

Optionally, the client MAY request that a credential be stored into the user account for use by the job process. When this is requested in the create call, a suitable delegated credential EPR is passed as part of the creation input. As for staging, the credential MUST have been delegated before the job is created (the green nodes in the picture).

### 3.1.4. Optional Credential Refresh

Optionally, credentials delegated for use with staging, transfer, or job processes may be refreshed using the Delegation service interface. This operation may be performed on any valid Delegation EPR (the blue/green striped node in the picture).

### 3.1.5. Optional Hold of Cleanup for Streaming Output

If the client wishes to directly access output files written by the job (as opposed to waiting for the stage-out step to transfer files from the job host), the client should request that the file cleanup process be held until released. This gives the client an opportunity to fetch all remaining/buffered data after the job completes but before the output files are deleted. (See the pink nodes in the high-level picture).

The cleanup hold and release are not necessary if the client will not be accessing files that are scheduled for cleanup in the job request, either because the client is not accessing any files or because the files it is accessing will remain on the job host after ManagedJob termination.

### 3.1.6. ManagedJob Destruction

Under nearly all circumstances, ManagedJob resources will be eventually destroyed after job cleanup has completed. Clients may hasten this step via an explicit terminate request or by manipulation of the scheduled termination time. Most system administrators will set a default and maximum ManagedJob linger time after which automatic purging of completed ManagedJob resources will occur.

## 3.2. Protocol Overview

As depicted above, the GRAM4 protocol is centered around the creation of a stateful ManagedJob resource using the ManagedJobFactory createManagedJob() operation. A simple batch job may involve nothing more than this initial client creation step, with all other job life cycle steps occurring automatically in the server. A number of optional protocol elements are available for more complex scenarios.

### 3.2.1. DelegationFactory::requestSecurityToken

This (optional) step allows a client to delegate credentials that will be required for correct operation of GRAM4, RFT, or the user's job process. Such credentials are only used when referenced in the subsequent job request and under the condition that GRAM4 or RFT is configured to make use of the DelegationFactory, respectively.

### 3.2.2. Delegation::refresh

This (optional) step allows a client to update the credentials already established for use with the previous requestSecurityToken step.

### 3.2.3. ManagedJobFactory::getResourceProperty and getMultipleResourceProperties

These (optional) steps allow a client to retrieve information about the scheduler and the jobs associated with a particular factory resource before or after job creation. The delegationFactoryEndpoint and stagingDelegationFactoryEndpoint resource properties are two examples of information that may need to be obtained before job creation.

### 3.2.4. ManagedJobFactory::createManagedJob

This required step establishes the stateful ManagedJob resource which implements the job processing described in the input request.

### 3.2.5. ManagedJob::release

This (optional) step allows the ManagedJob to continue through a state in its life cycle where it was previously held or scheduled to be held according to details of the original job request.

### 3.2.6. ManagedJob::setTerminationTime

This (optional) step allows the client to reschedule automatic termination to be different than was originally set during creation of the ManagedJob resource.

### 3.2.7. ManagedJob::terminate

This (optional) step allows the client to explicitly terminate a ManagedJob resource. Options to the terminate call determine whether the ManagedJob resource associated with the job and delegated credentials will be destroyed after the termination, or not. If the job has already completed (i.e. is in the Done or Failed state), no clean up steps need to be done. If the job has not completed, appropriate steps will be taken to purge the job process from the scheduler and perform clean up operations before setting the job state to UserTerminateDone or UserTerminateFailed if errors during clean up occurred. If the ManagedJob resource is not destroyed after clean up steps it will be destroyed when its lifetime expired.

### 3.2.8. ManagedJob::subscribe

This (optional) step allows a client to subscribe for notifications of status (and particularly life cycle status) of the ManagedJob resource. For responsiveness, it is possible to establish an initial subscription in the createManagedJob() operation without an additional round-trip communication to the newly created job.

### 3.2.9. ManagedJob::getResourceProperty and getMultipleResourceProperties

These (optional) steps allow a client to query the status (and particularly life cycle status) of the ManagedJob resource.

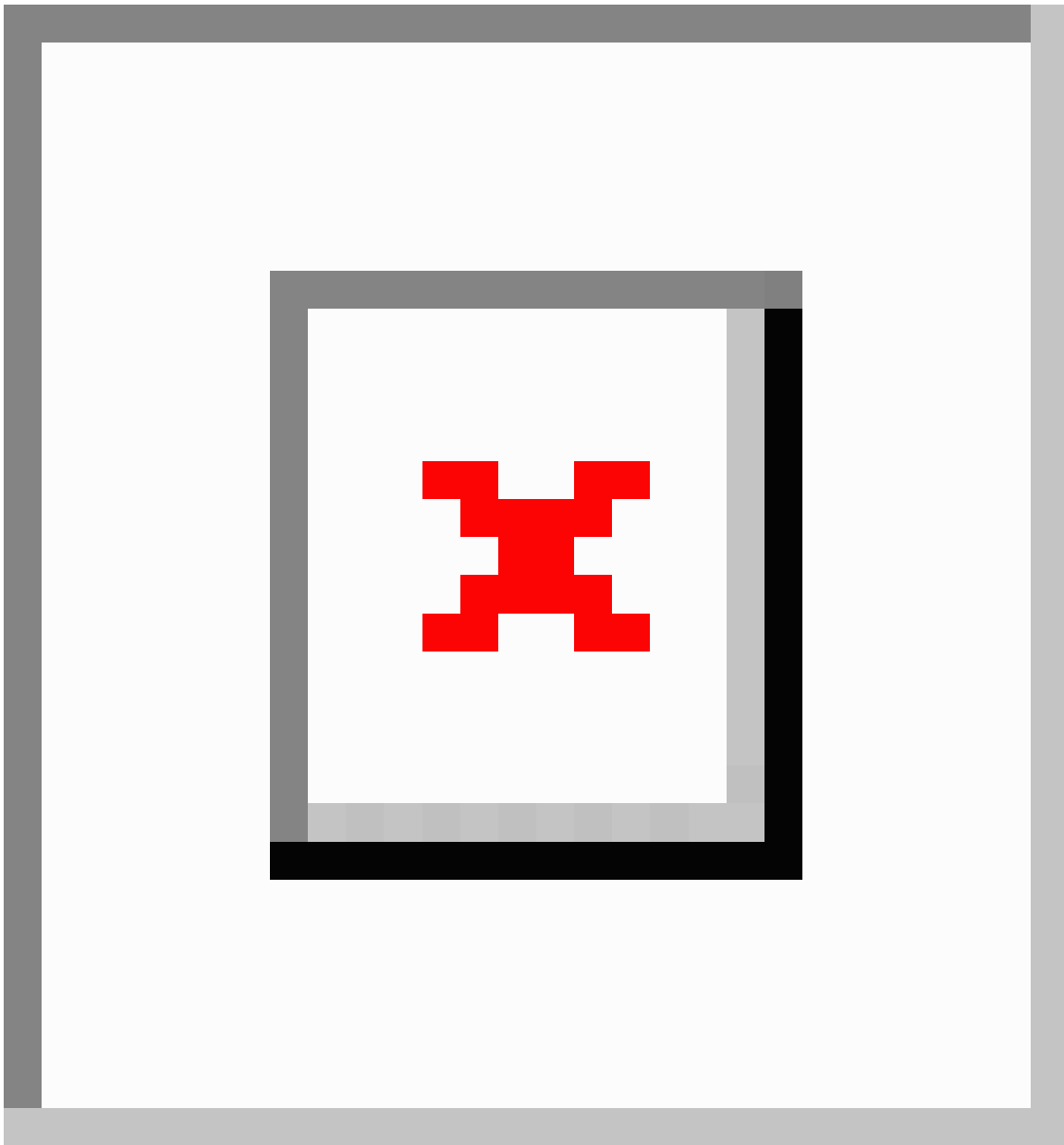
## 3.3. Protocol Variations

From a protocol perspective, the longest latency GRAM4 submission scenario involves credential delegation, staging before and after the job, and an explicit hold handshake on file cleanup after the job. The credential refresh feature of GRAM4 can be repeated any number of times, so by longest sequence we mean the longest fixed sequence with at most one credential delegation. Explicit termination is not necessary with GRAM4 so we will not consider that case further.

To understand the following figures which illustrate the protocol sequence: the arrows show communication, signalling, or causal links between tiers in the architecture and the vertical span indicates elapsed time (with the start time at the top of the diagram). Due to unpredictable implementation delays, client and job-observed times are not necessarily ordered with respect to the GRAM4 observed times and the GRAM4 generated state notification messages. The diagrams show one possible ordering but applications (and our measurement methods) must tolerate other orderings as well.

### 3.3.1. Minimal Protocol Sequence

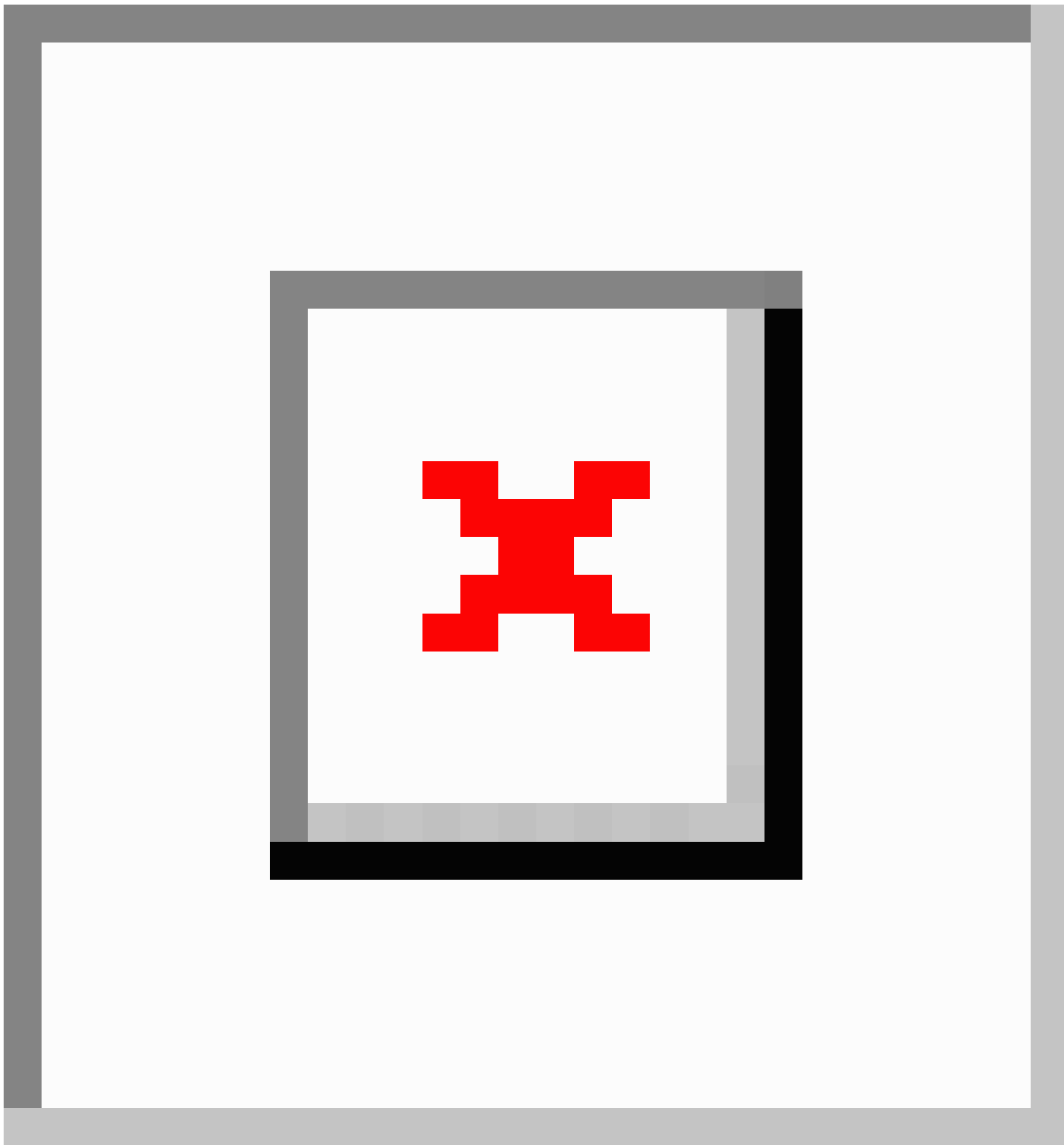
The simplest GRAM4 scenario involves a job that requires neither delegated credentials nor staging and that makes use of the automatic termination of resources to avoid an explicit termination request. In this case, we can measure the latency and throughput for job submission and notification alone.

**Figure 2.2. Minimal Protocol Sequence**

Note: Any difference between this case and the same measurement points in the full scenario must be due to the additional overhead of the delegation and staging services on the front-end node?

### 3.3.2. Non-staging Delegation Sequence

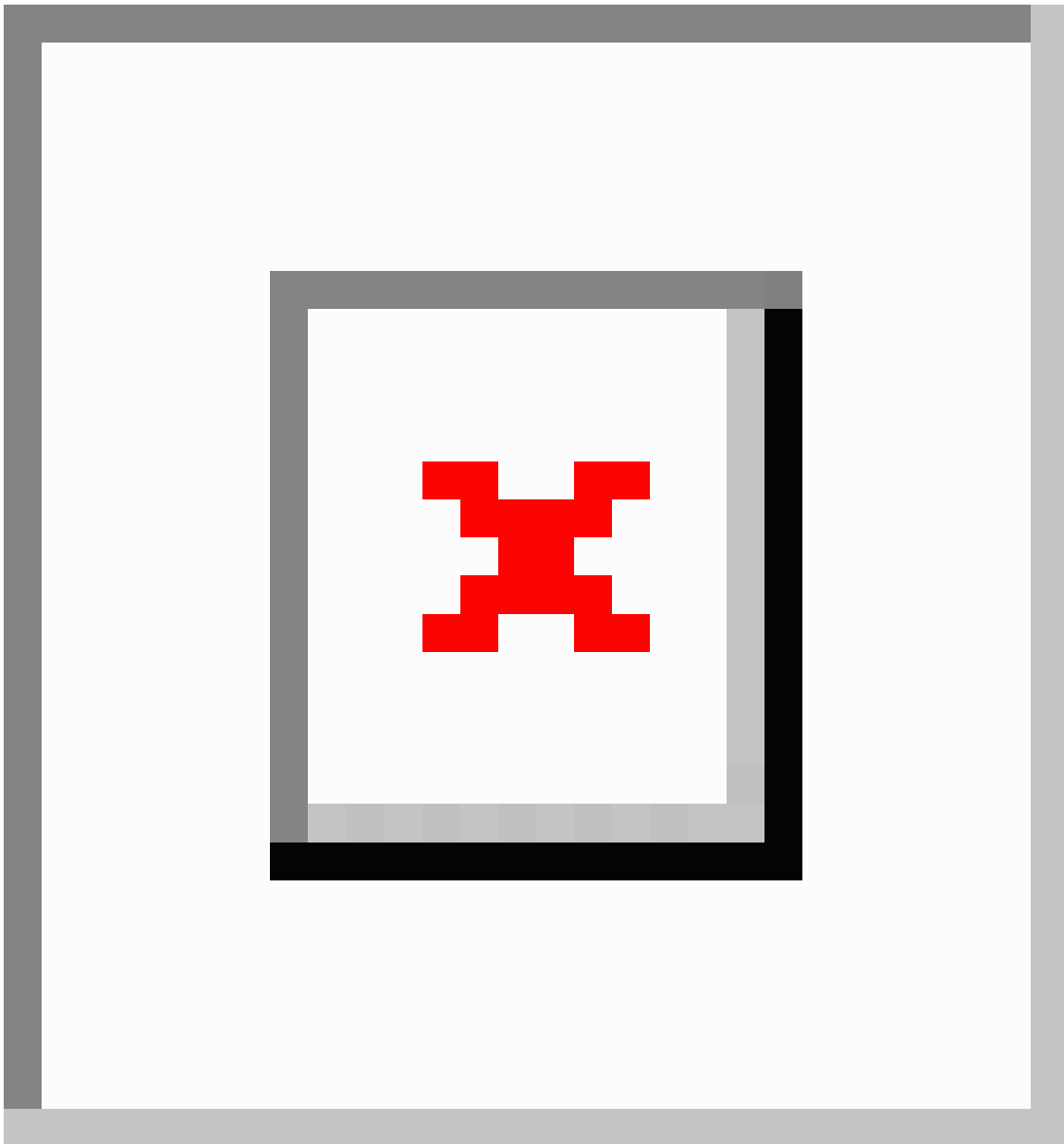
A slightly longer form of job than the minimal sequence is to include credential delegation for use by the job itself, without any staging directives. This sequence is comparable in functionality to previous GRAM releases where delegation was mandatory but staging could be omitted as per the client's request.

**Figure 2.3. GRAM4 Components**

### 3.3.3. Non-staging Delegation Sequence with Hold

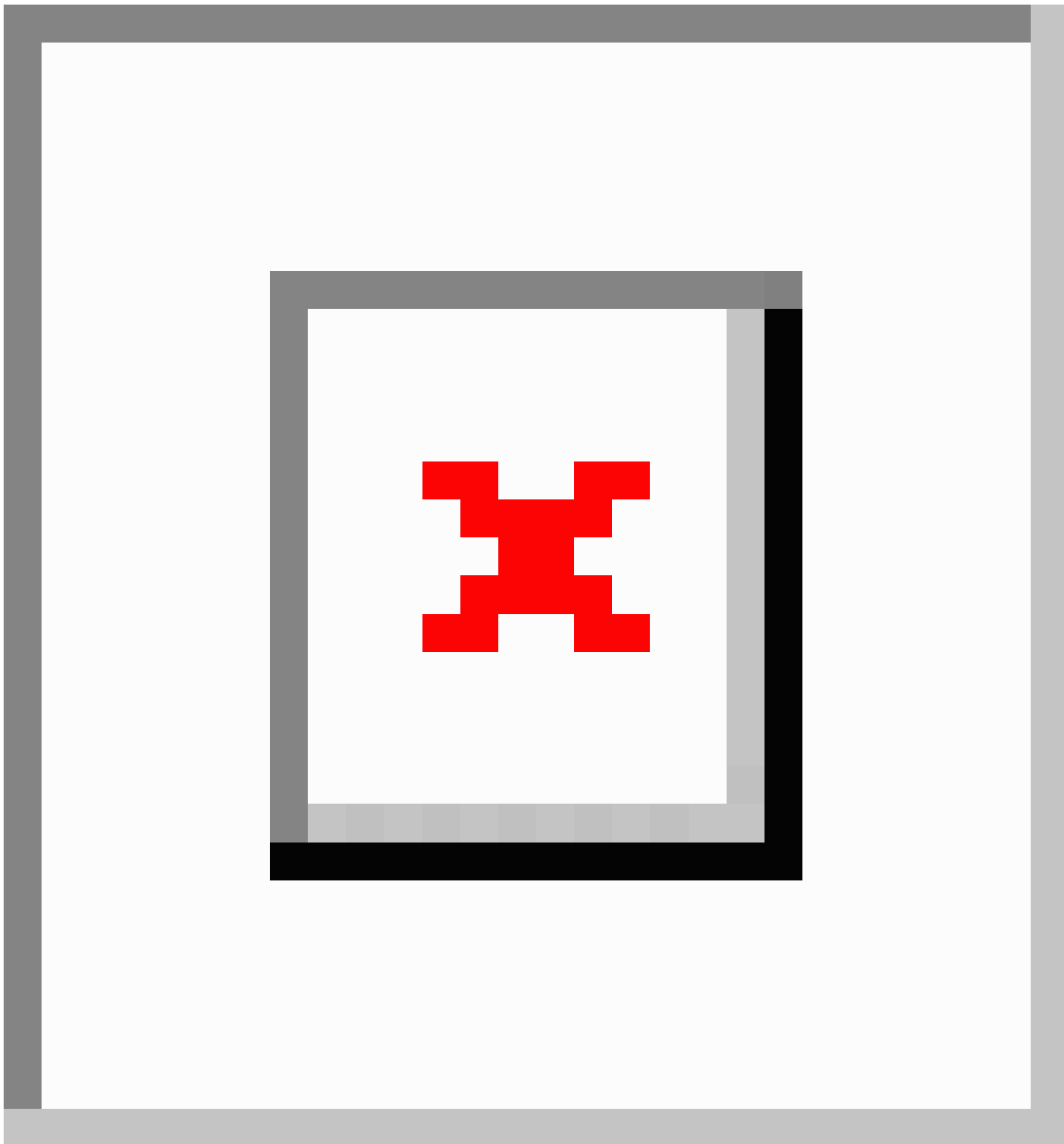
An optional protocol sequence allows the cleanup state to be held in order to allow a client to safely access output files via the GridFTP server after the job has finished writing them and before the cleanup step deletes them. This variant adds the cleanup hold handshake to the previous scenario.

**Figure 2.4. Non-staging Delegation Sequence**



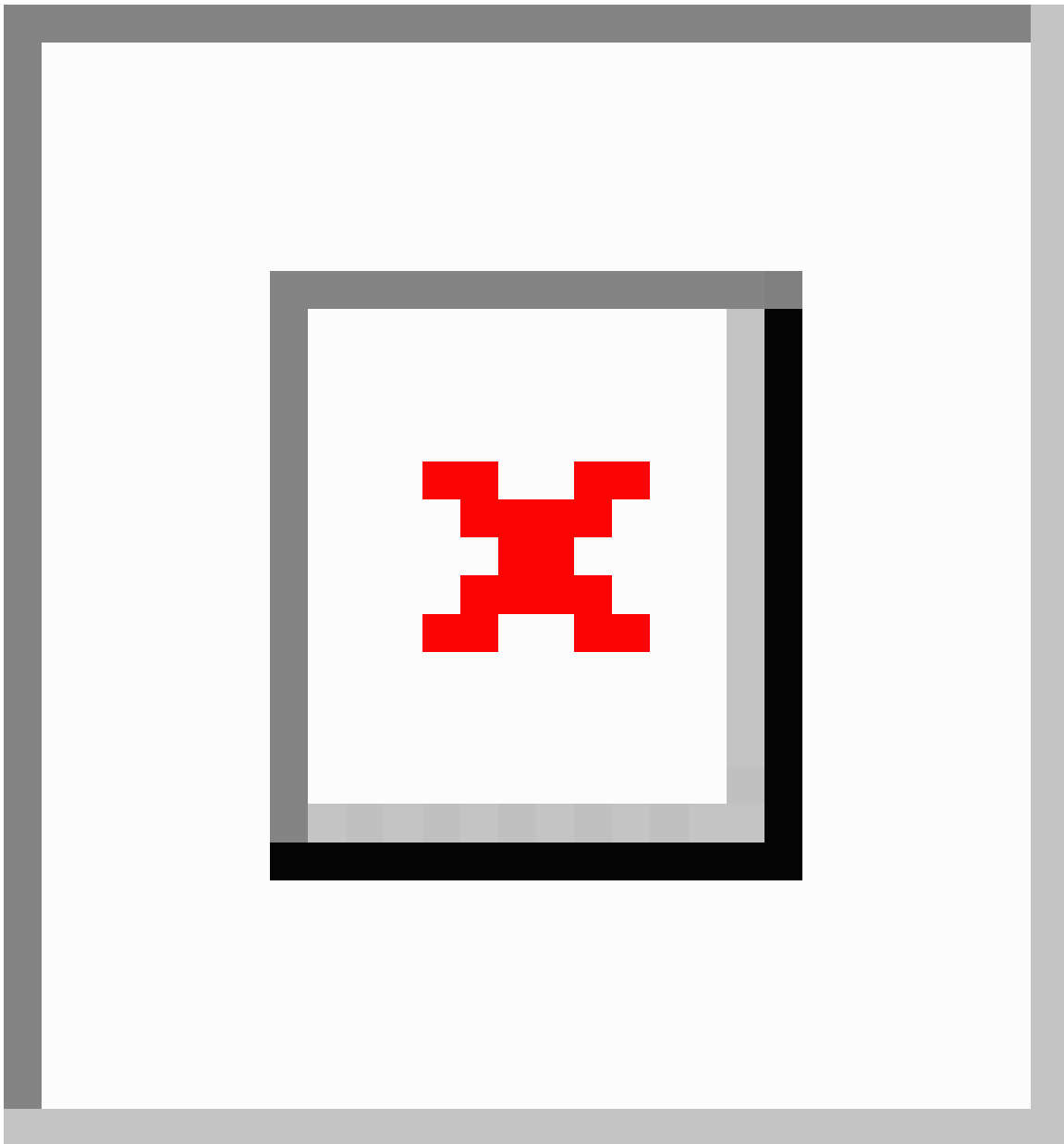
### 3.3.4. Staging Sequence

This staging sequence uses almost all of the protocol elements.

**Figure 2.5. Staging Sequence**

### 3.3.5. Staging Sequence with Hold

This staging sequence adds the cleanup hold handshake to the staging example to represent a job that has staged files as well as "streamed" output.

**Figure 2.6. Staging Sequence with Hold**

## 4. Security model

### 4.1. Secure operations

GRAM4 utilizes secure Web service invocation, as provided by the WSRF core of the Globus Toolkit, for all job-management and file-management messages. This security provides for authentication of clients, tamper-resistant messaging, and optional privacy of message content.

## 4.2. Local system protection domains

User jobs are executed within Unix user accounts. GRAM4 authentication mechanisms allow the administrator to control to which local accounts a Grid-based client may submit jobs. GRAM4 uses the Unix `sudo` command to access user accounts after determining that the client has the right to access the account. Additionally, the administrator may use access and allocation policy controls in the local scheduler to further restrict the access of specific clients and Unix users to local computing resources.

## 4.3. Credential delegation and management

A client may optionally delegate some of its rights to GRAM4 and related services in order to facilitate file staging. Additionally, the client may delegate rights for use by the job process itself. If no delegation is performed, staging is disallowed and the job will have no ability to request privileged Grid operations.

# 5. Audit

GRAM4 provides three types of logging or auditing support:

## 5.1. WSRF core message logging

Detailed logging of the underlying client messages may be logged if such logging is enabled in the container configuration. [See Java WS Core debugging doc.](#)

## 5.2. GRAM4 custom logging

GRAM4 generates domain-specific logging information about job requests and exceptional conditions. [See GRAM4 debugging doc](#)

## 5.3. Local scheduler logging

For systems using a local batch scheduler, all of the accounting and logging facilities of that scheduler remain available for the administrator to track jobs whether submitted through GRAM4 or directly to the scheduler by local users.

## 5.4. Local system logging

The use of `sudo` for all operations against target user accounts allows the administrator to log the low-level system operations requested by GRAM4 using `sudo`'s native auditing support.

# 6. Job Description Language

Jobs submitted to GRAM4 must be described in GRAM4 job description language. Elements of a job description are used to generate the job description which is required to submit jobs to local resource managers (LRM) supported by GRAM4 like Condor, PBS, LSF or to start a job on the Head node directly using the fork starter.

# 7. Persistence Data

GRAM4 stores information about job resources on disk to ensure that no information about running jobs is lost in case of a container shutdown.

## 8. Job Lifetime

Jobs submitted to GRAM4 have a lifetime. If the lifetime of a ManagedJob resource expires the job will be terminated and the job's persistence data removed. Removing expired jobs keeps the GRAM4 service and the GT4 container in a clean and operational state. A client has full control over the lifetime of its job resources:

If a client does not specify a lifetime the ManagedJob resource will not be removed until it is fully processed, regardless how long the job takes. After the ManagedJob resource has been fully processed it will still exist for an amount of time configured on the server-side before it expires. Fully processed means the job state has reached a final state, one of the following: Done, Failed, UserTerminatedDone, UserTerminatedFailed.

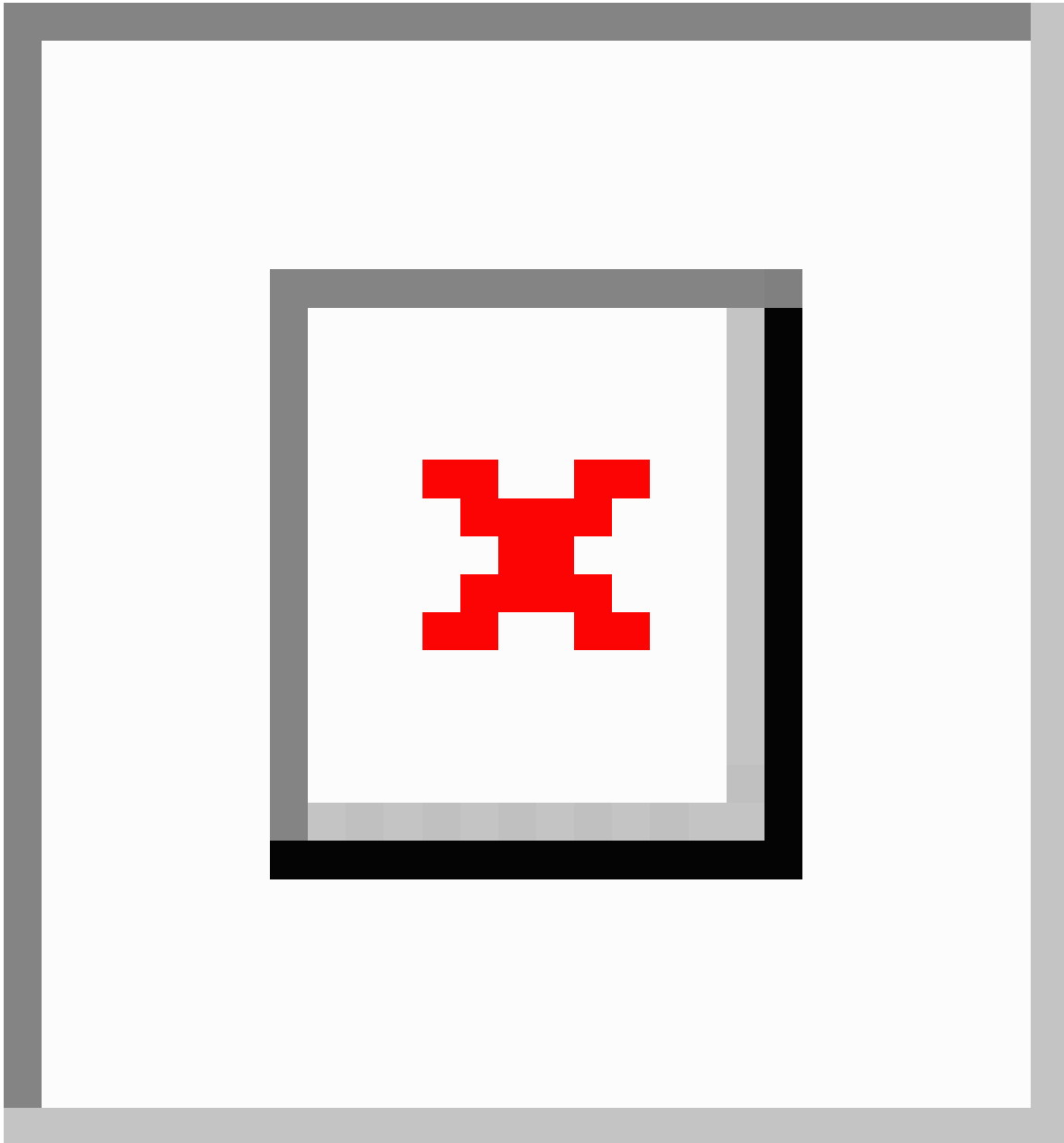
If the client explicitly sets a lifetime the ManagedJob resource and the specified lifetime expires, then the job will be terminated and persistence data removed. However, a client can extend the lifetime at any time before expiration. An upper limit for setting/extending the max lifetime can be configured by the admin, so the max value could vary between GRAM4 service deployments.

DRAFT

## 9. GRAM4 software architecture

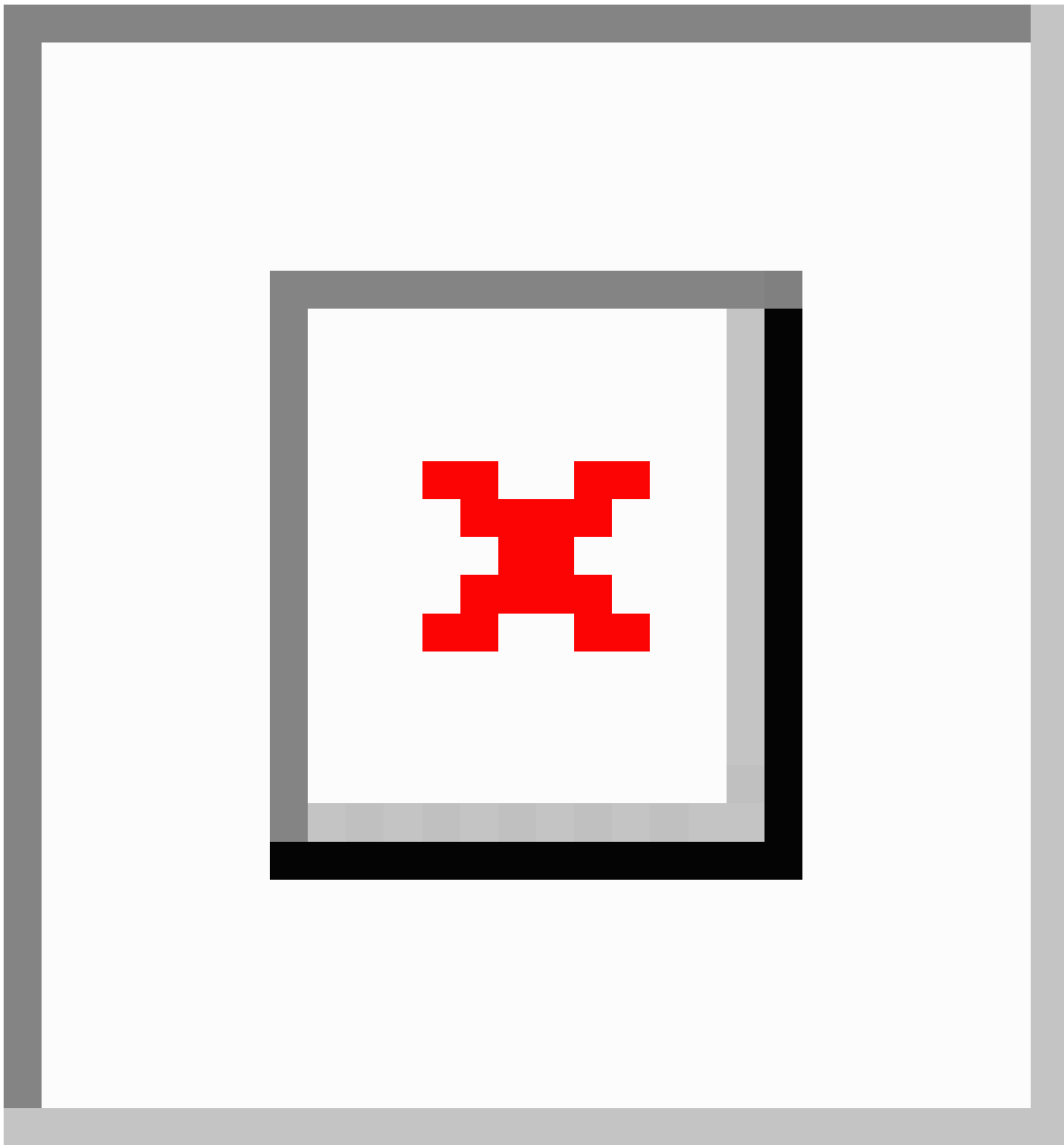
### 9.1. Overview

Figure 2.7. GRAM4 Software



### 9.2. ManagedJob Resource Life Cycle Logic

The ManagedJob resource has a complex life cycle. The generic behavior is depicted in the following flowchart as a partially ordered sequence of processes and decision points. The status of the ManagedJob resource, including its "job state" is set as a side-effect of this control flow. The processes in the flowchart do not all directly correspond to client-visible job states.

**Figure 2.8. GRAM4 Lifecycle**

## 9.3. Software for local system interaction

### 9.3.1. Scheduler adapters

Support to control each local scheduler is provided in the form of adapter scripts in the Perl programming language, following the proprietary GRAM adapter plugin API. These adapters implement the system-specific submission, job exit detection, job cancellation, and (optionally) job exit status determination processes.

### 9.3.2. globus-gridmap-and-execute

The `gridmap_authorize_and_exec` tool is a default, but optional, program that is invoked in the target user account as a wrapper around GRAM4 operations in order to make a final safety check for whether GRAM4 should be allowed to operate in that account. This tool provides reasonable privilege limits to guard against service compromise without requiring additional system administrator efforts to manage user policies.

DRAFT