Advantages and challenges of application execution management

Toni Cortes

Barcelona Supercomputing Center

What is AEM

- **Application Execution Management tasks**
  - Build the environment to run jobs
    - Select resources to be used and create reservations
    - Set up the infrastructure to allow interactivity
  - Control the execution of jobs
    - Start/stop/resume/cancel
  - Monitor the execution of jobs
    - System events
    - User/application events
  - Monitor the status of resources
  - Guarantee the tolerance to failures

Overview

- Why XtreemOS is the OS to use
- What can I do with XtreemOS and how
- AEM internals

Overview

- **Why XtreemOS is the OS to use**
  - What does the AEM in XtreemOS offers that other systems do not
  - Performance comparison
- **What can I do with XtreemOS and how**
- **AEM internals**

Grid awareness

- **Users may be unaware of Grid issues**
  - Grid used like any interactive system
    - If you know Linux you know Grid
    - Application can be interactive
  - "Grid parameters" used
    - Default ones (system, vendor, …)
    - Learned ones
- **Grid-aware users may use all potential**
  - Define "Grid parameters"
- **Current systems are only for Grid-aware users**
- **Current systems only allow batch jobs**

Grid like Linux

- **XtreemOS tries to reuse Unix/Linux concepts**
  - Not invent new ones
- **Parent hierarchy**
  - XtreemOS implements parent hierarchy
    - Including `jobWait` (mimicking process wait)
- **Processes are to jobs as threads to processes**
- **Job control is managed via signals to jobs**
  - Including new Grid signals
- **Current systems reinvent new mechanisms**
  - Make users life more complex

Execution environment

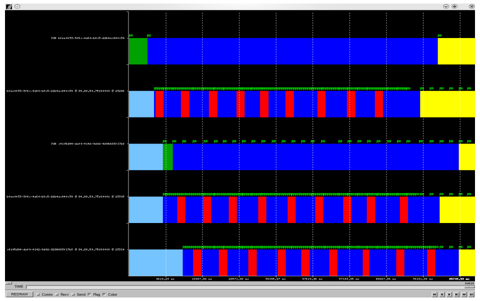
- **Jobs may not need to run in exclusive access**
  - Not all jobs require exclusive access
    - Especially interactive ones
- **XtreemOS allows the user to decide whether**
  - To use exclusive node
  - To use shared nodes
    - Nodes will run more than one job at a time
- **Current systems do not allow the user to decide the environment**

Reservations

- **XtreemOS allows parallel applications**
  - Several resources allocated to the same application
    - Resources can be coordinated if needed
  - All managed via reservations
    - May be implicit if the user does not care about them
- **One reservation may be used by several jobs**
  - Simplify the work of workflow managers
- **Current systems, at least not all of them, offer reservations**

Monitoring

- **Extensible job monitoring**
  - The system monitors its own events
  - Any component can add information
    - Including the application itself
  - The user can decide what is monitored and what is not
- **Monitoring is done at thread level**



- Current systems **have very limited monitoring**
- Current systems **only monitor at job level**

Dependencies

- **XtreemOS allows users to define dependencies among jobs**
  - Dependency trees are tagged
    - User can have one for each need (workflow, monitoring, …)
- **The meaning of dependencies is user-decided**
- **Implemented examples**
  - Monitor a dependency tree
  - Kill a dependency tree
  - …
- **Current systems, at most, have predefined ones**

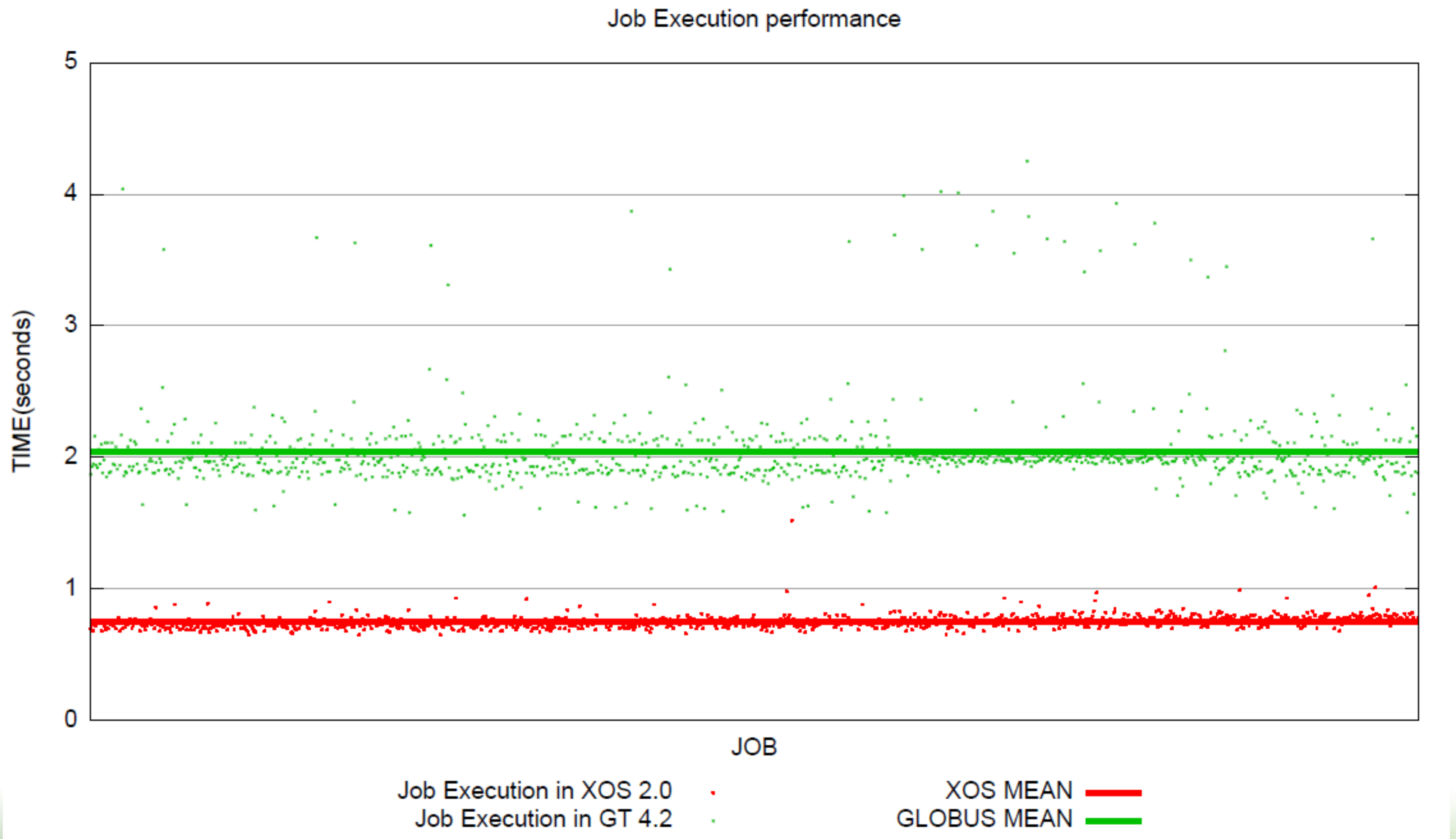
Jobs and files

- **XtreemOS is aware that jobs use files**
  - When selecting the resources, file location will be taken into account
    - Nodes close to the files will be requested
    - The user needs to specify the files used
  - If cannot find resources close to files
    - Replicas will be requested to XtreemFS
- **Current systems are not file closeness aware**

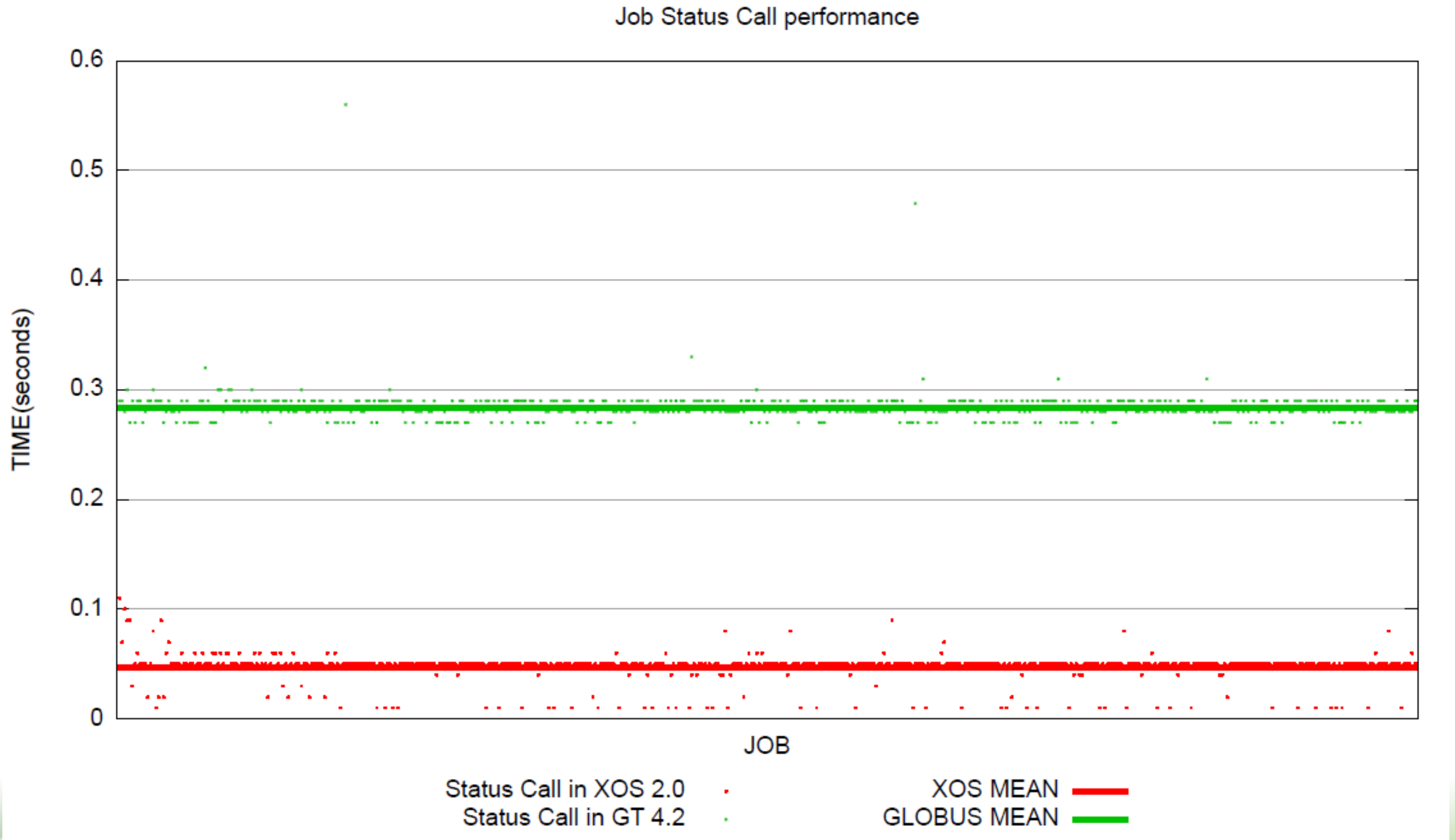
Testbed

- **Comparison: AEM 2.0 – Globus 4.2.1 (1 Node)**
  - Job Execution(/bin/true)
  - Cost of checking job status
- **Environment**
  - 4 nodes (1 Core and 3 resources)

Job execution



Job status


Job Status Call performance
0.6
0.5
0.4
0.3
0.2
0.1
0
TIME(seconds)
JOB
Status Call in XOS 2.0
Status Call in GT 4.2
XOS MEAN
GLOBUS MEAN

Overview

- Why XtreemOS is the OS to use
- What can I do with XtreemOS and how
  - Description of the procedures involved to do all you need to do to manage jobs
- AEM internals

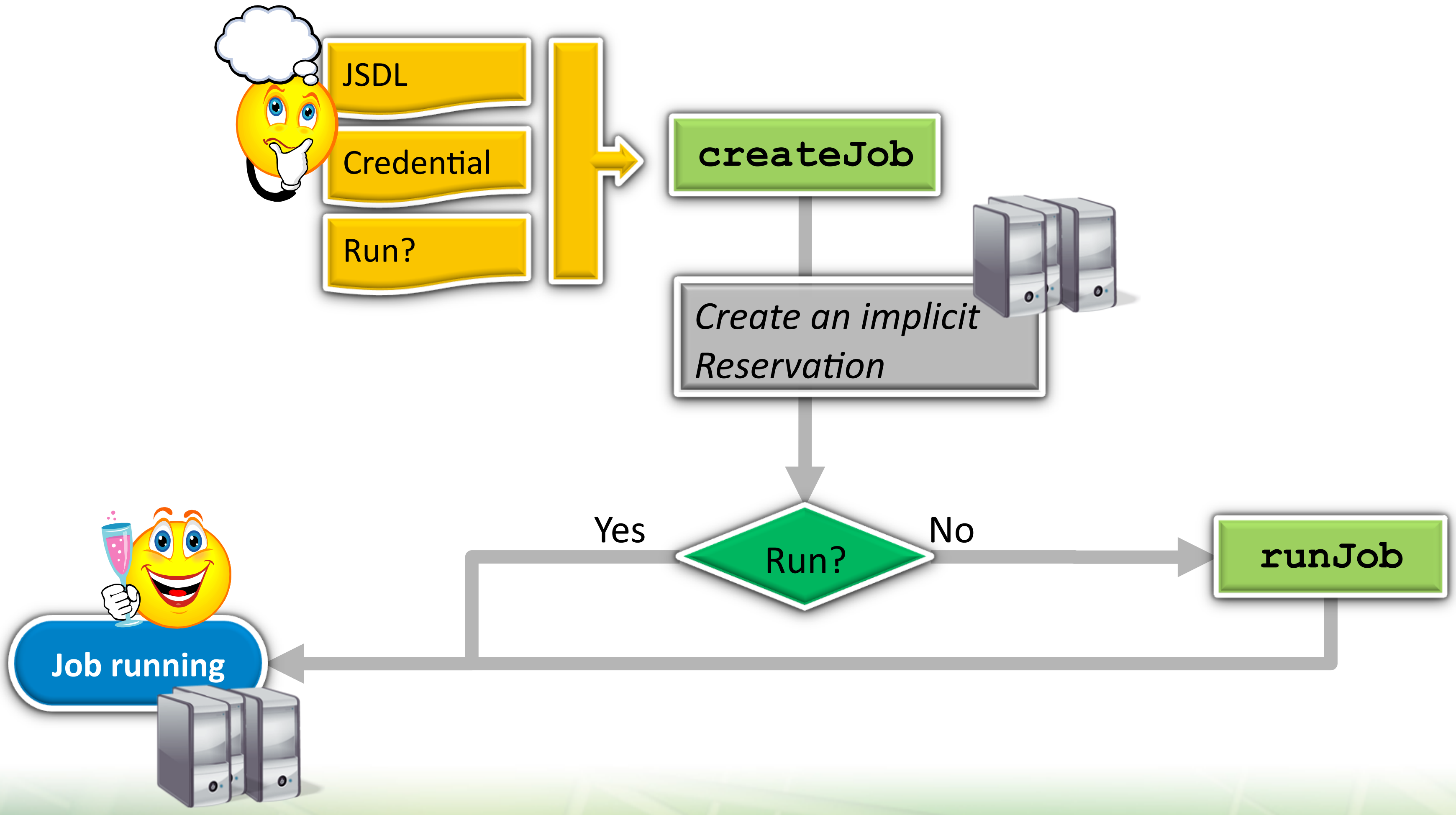
Job description

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition
   xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
   xmlns:jsdl-posix”http://schemas.ggf.org/jsdl/2005/11/jsdl-
   posix">
<jsdl:JobDescription>
     <jsdl:JobIdentification>
         <jsdl:JobName>ls</jsdl:JobName>
     </jsdl:JobIdentification>
     <jsdl:Application>
         <jsdl-posix:POSIXApplication>
             <jsdl-posix:Executable>sleep</jsdl-posix:Executable>
             <jsdl-posix:Argument>300</jsdl-posix:Argument>
         </jsdl-posix:POSIXApplication>
     </jsdl:Application>
```

Job description

```
  <jsdl:Resources>
         <jsdl:TotalResourceCount>
             <jsdl:Exact>2</jsdl:Exact>
         </jsdl:TotalResourceCount>
     </jsdl:Resources>
</jsdl:JobDescription>
</jsdl:JobDefinition>
```

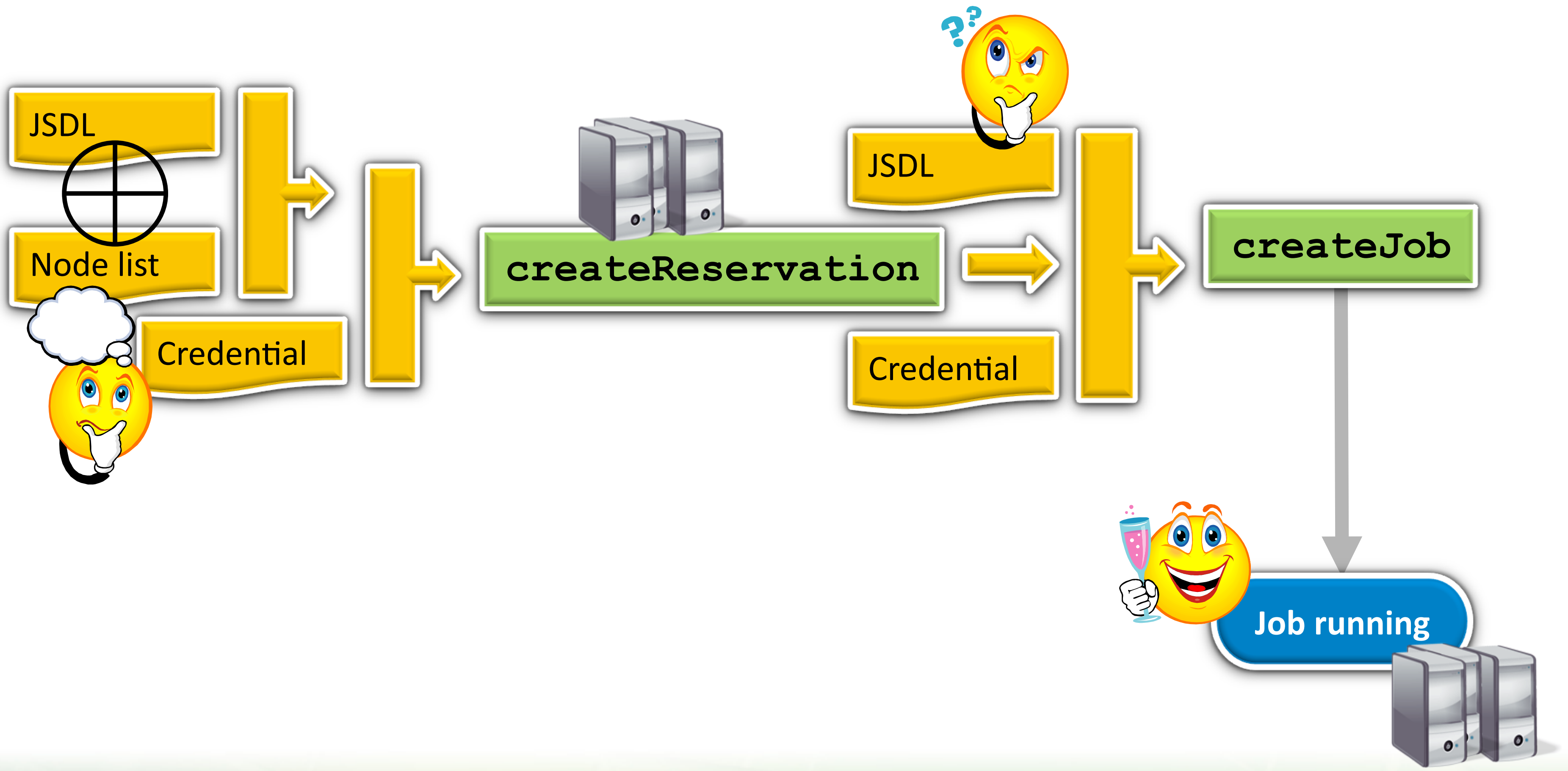
Executing a job



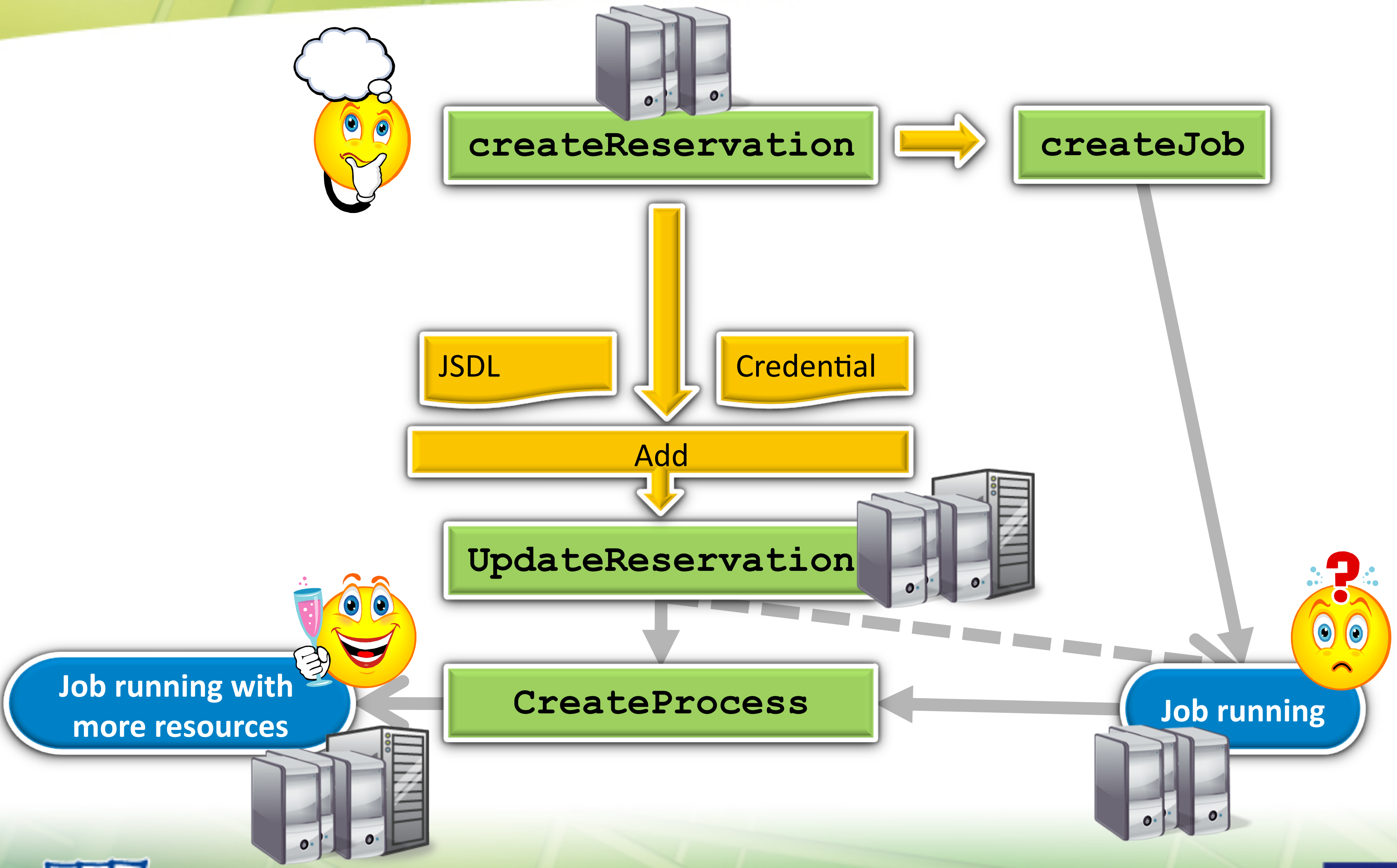
Executing a job Linux-like

- **Ways to execute a job from the shell**
  - Option 1
    - $ `executable.jsdl [params] -in f -out ff`
      - If this file is empty, the system will fill it
      - This is the most Unix-like version
      - Credential will be taken automatically
      - Parameters and redirections can also be inside JSDL
  - Option 2
    - $ `xsub -f executable.jsdl`
  - Option 3
    - $ `xsub.sh executable [params] -in f -out ff`

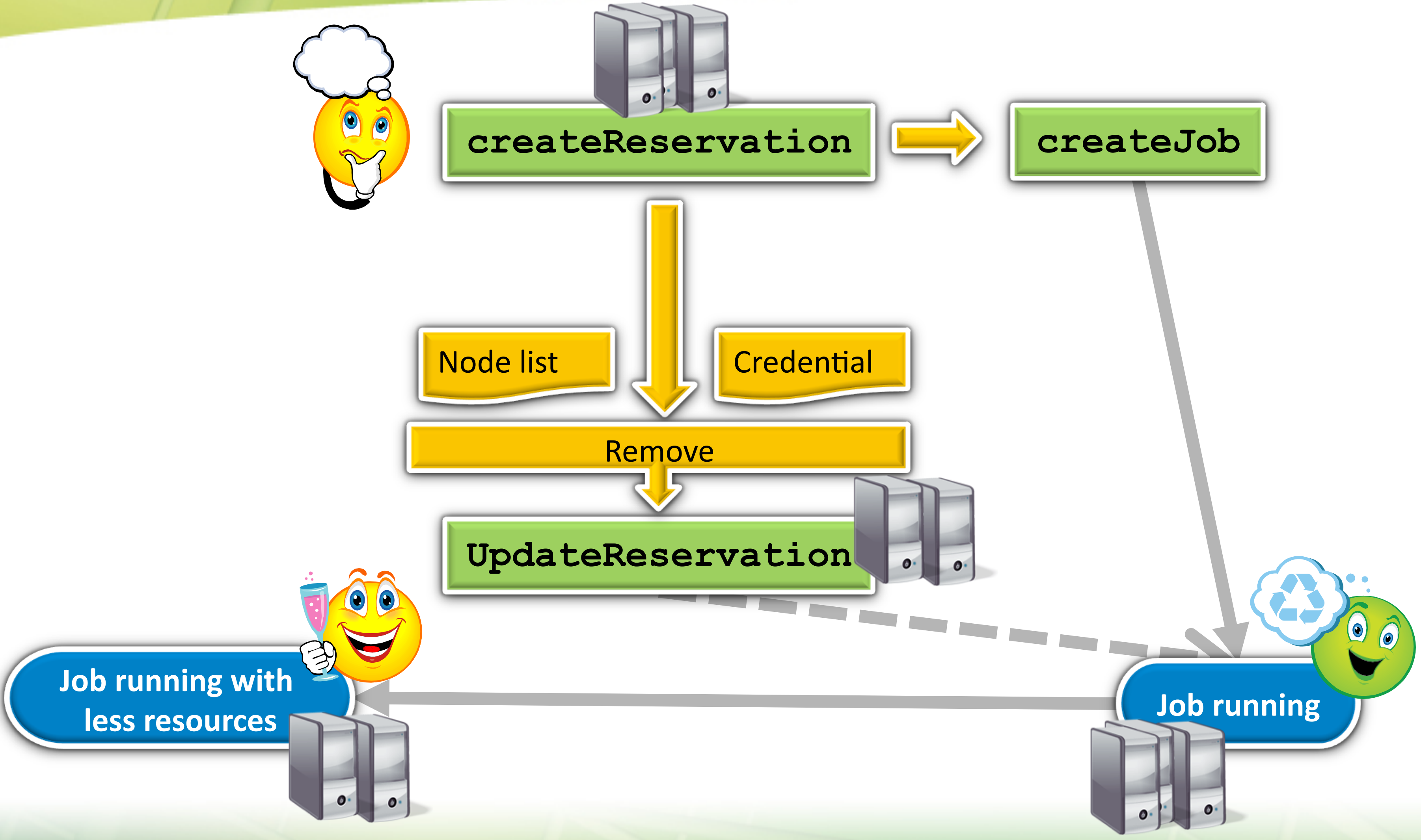
Explicit reservations



Dynamic reservations



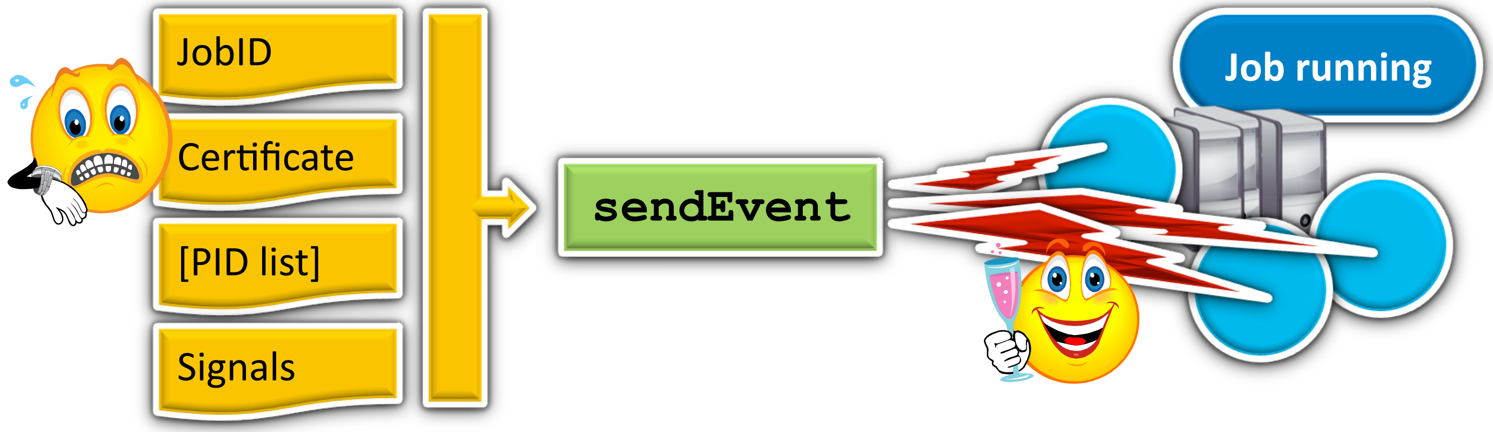
Dynamic reservations



Multi-job reservation

- **A job can only use 1 reservation**
- **A reservation may hold many jobs**
  - Easy to implement workflow tools
  - Easy to implement applications with several jobs
  - User/programmer responsibility to coordinate them

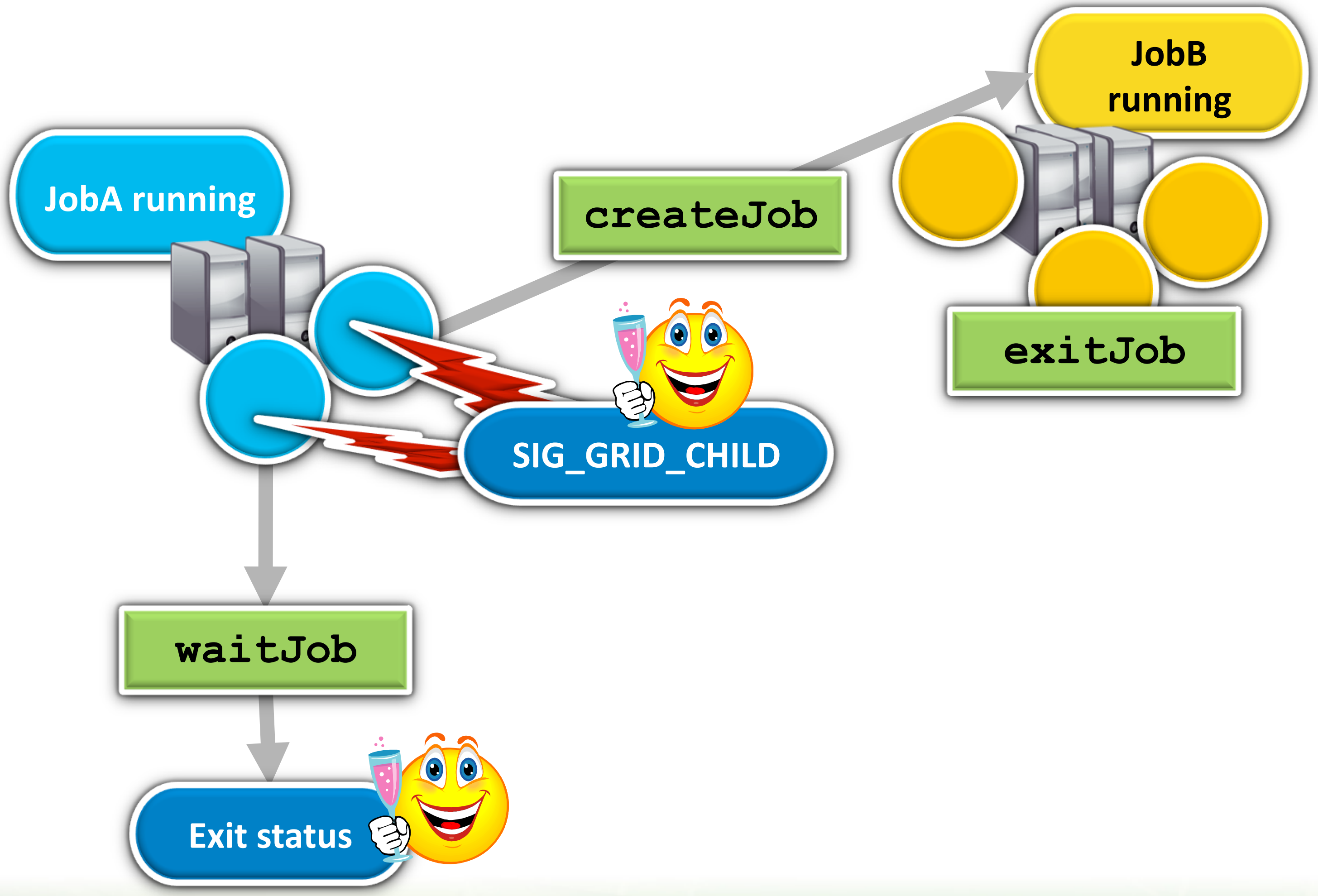
Signals to jobs



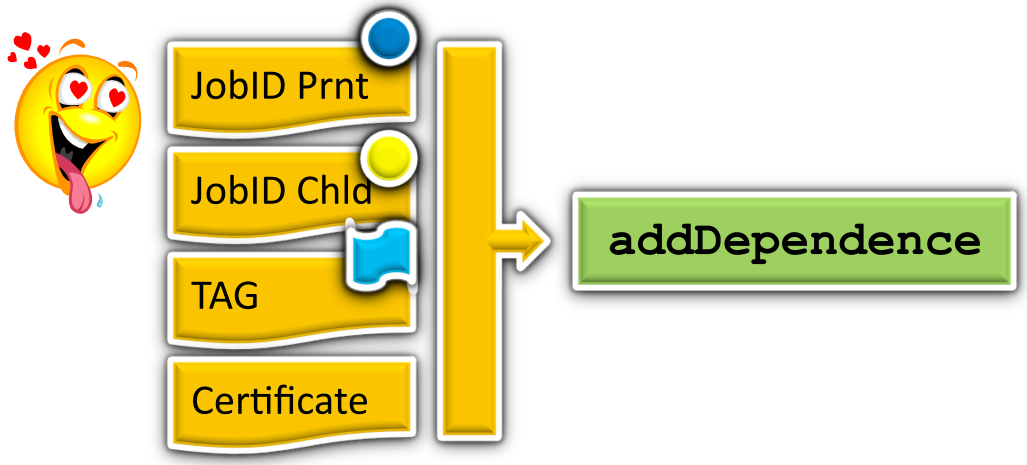
Job control

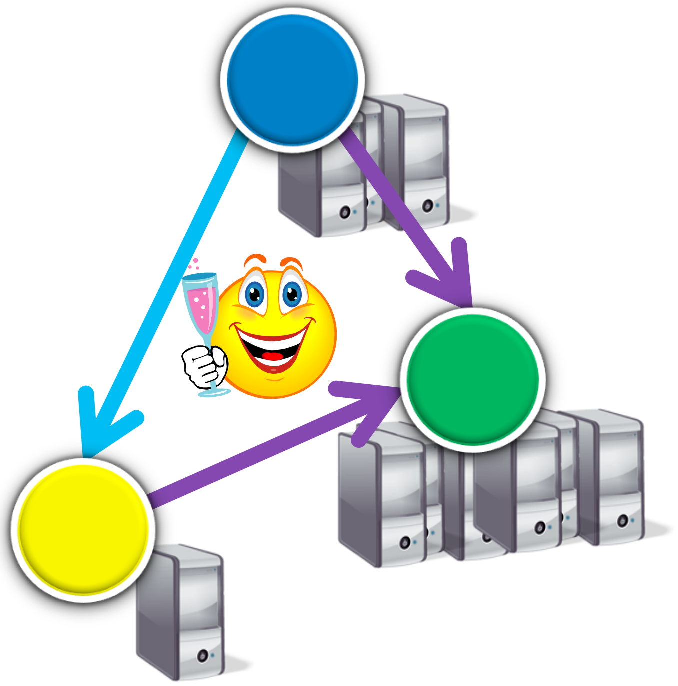
- **"Duplicated" call**
  - `jobControl (jobId, ctrOp, userCtx)`
  - Mapped to a signal event
- **Current control operations**
  - STOP ➔ `SIG_STOP`
  - CONTINUE ➔ `SIG_CONT`
  - KILL ➔ `SIG_KILL` or `SIG_TERM`
  - Or any Linux process control event
- **Sending signals Linux-like**
  - `Kill signal jobID`

Special Grid signals

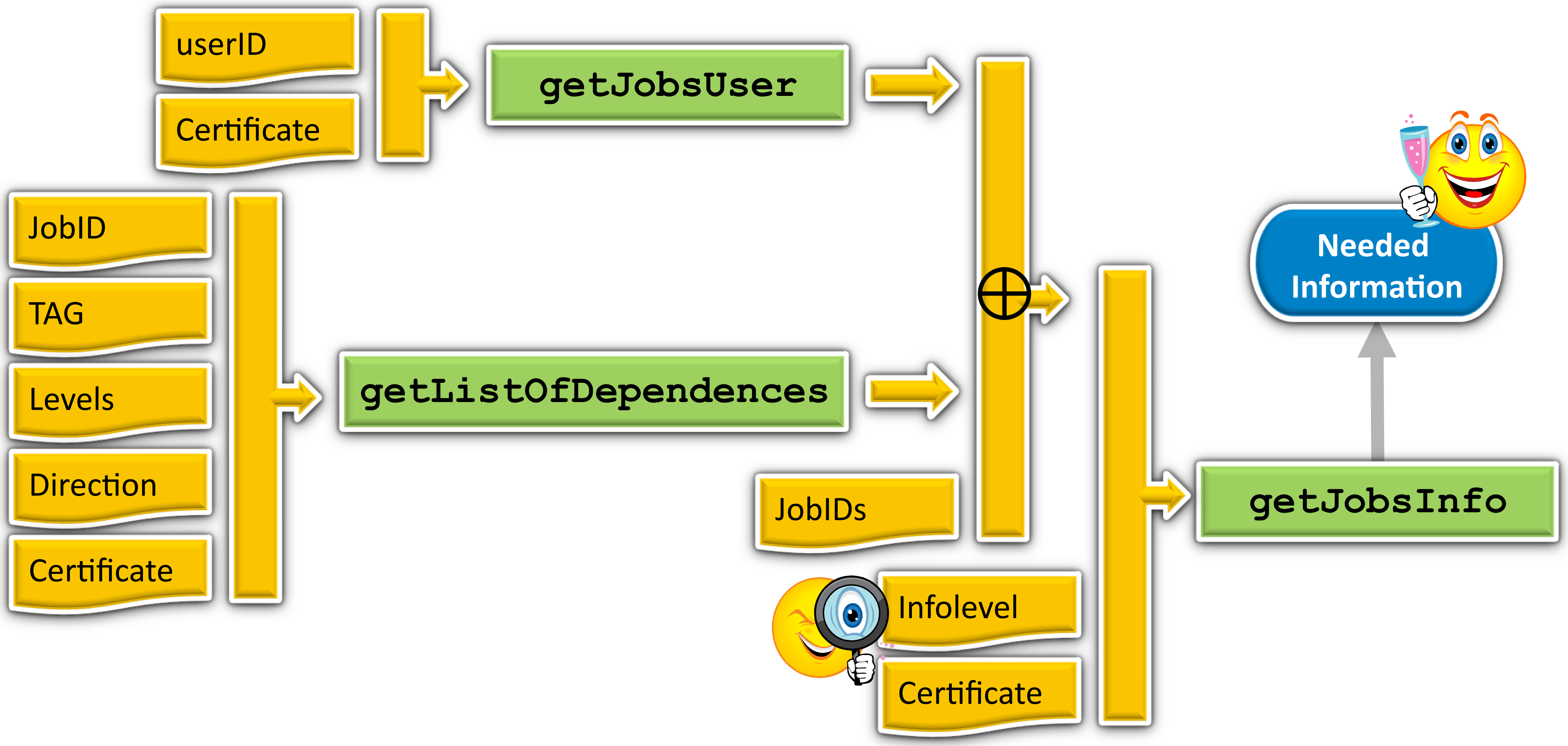


Dependences





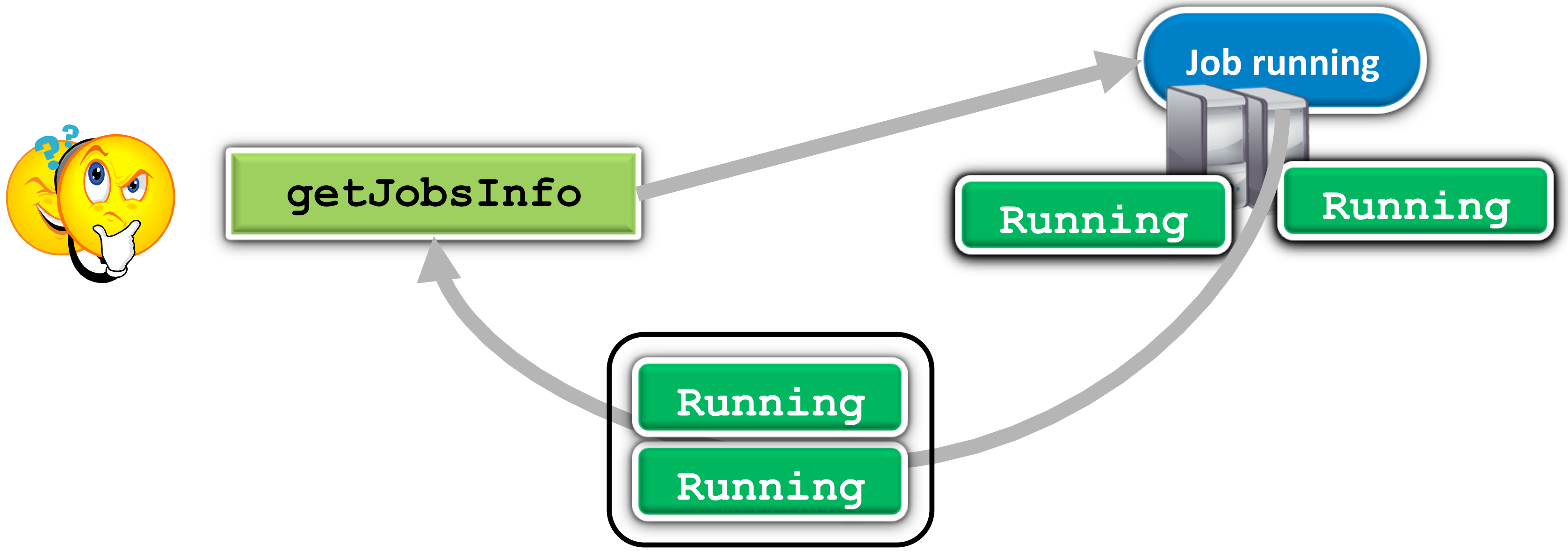
Getting job information



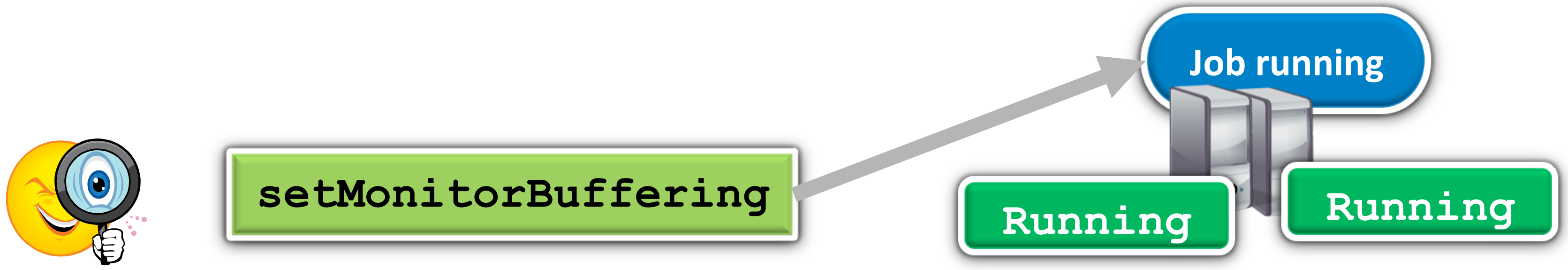
Job info. Linux-like

- **Job information will appear on /proc**
  - Another way to get information instead of using special calls
  - `/proc/XtreemOS/jobID/…`
- **Will be integrated in the `ps`**

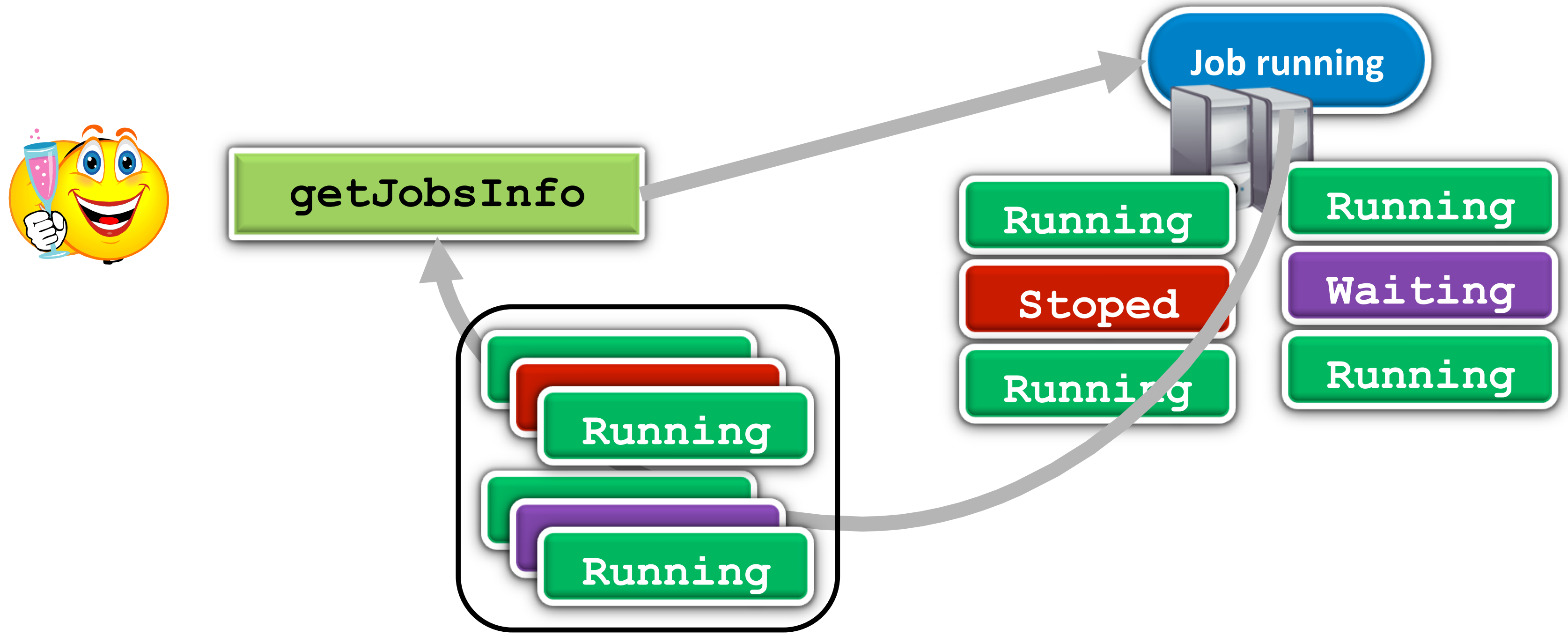
Monitoring buffers



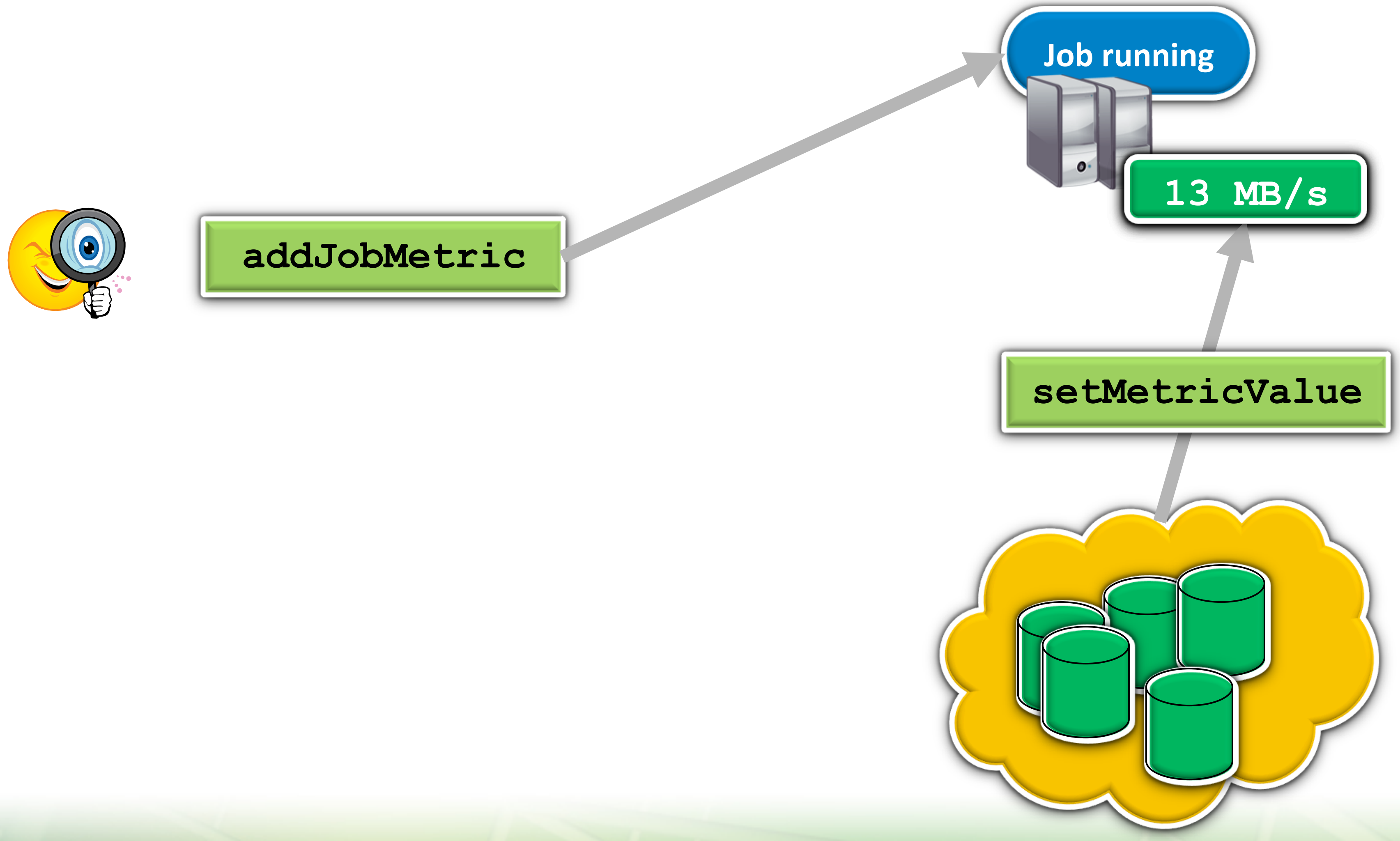
Monitoring buffers



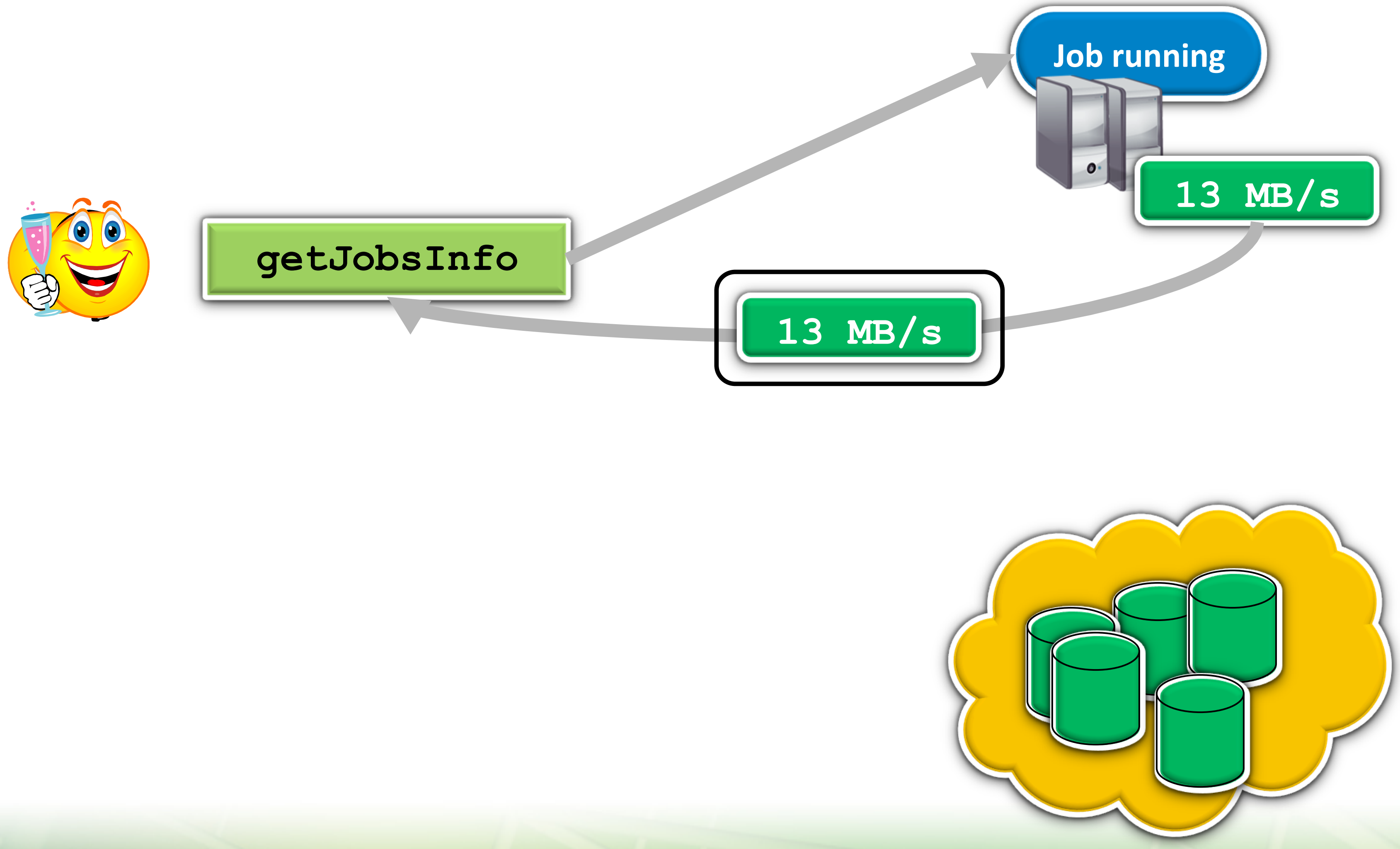
Monitoring buffers



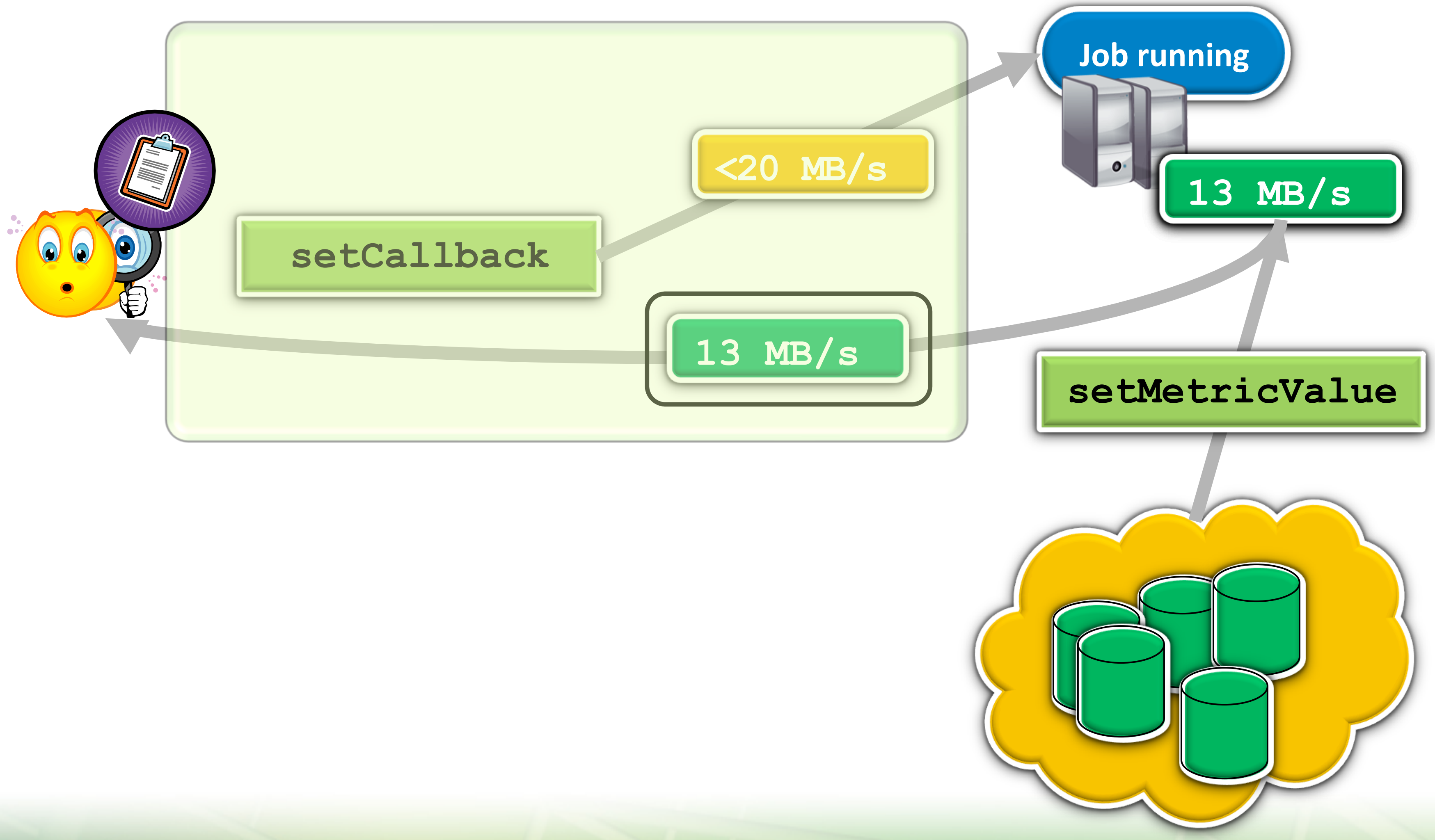
Monitoring user metrics



Monitoring user metrics



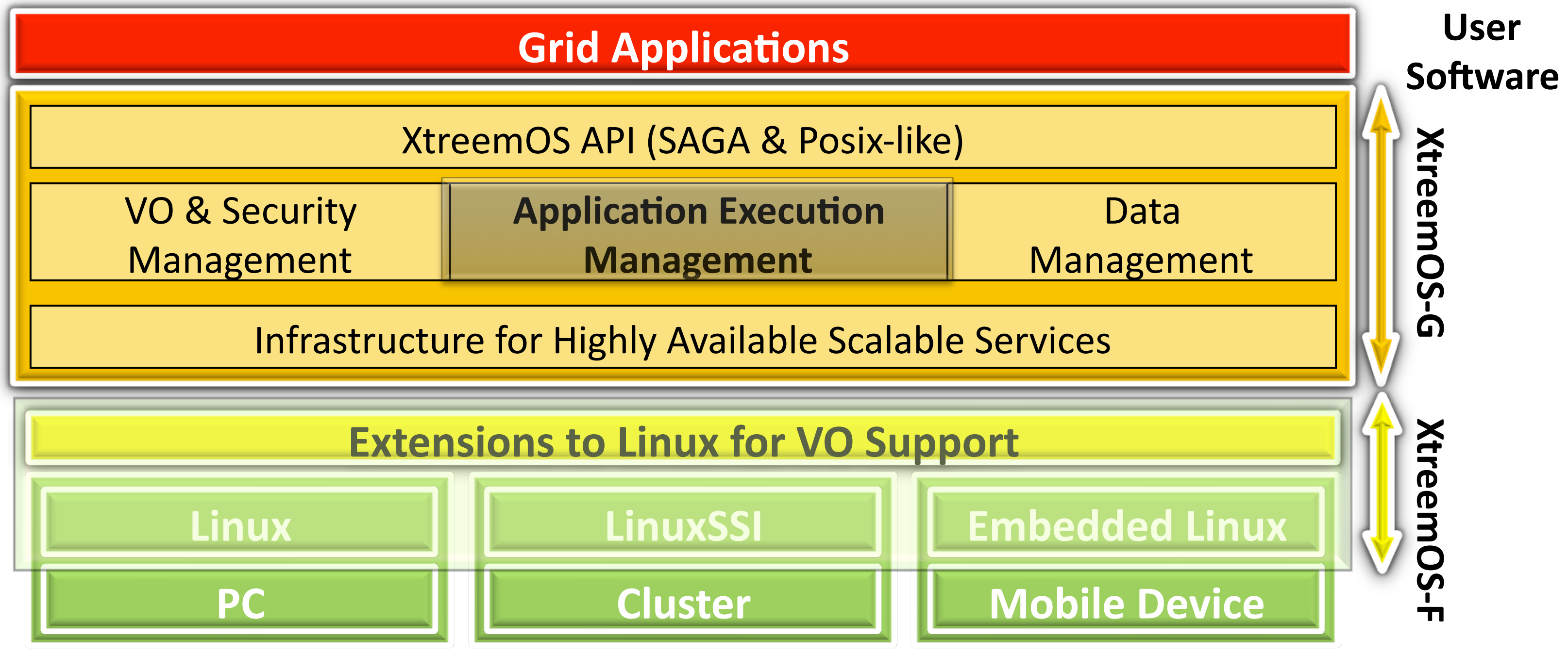
Monitoring callback



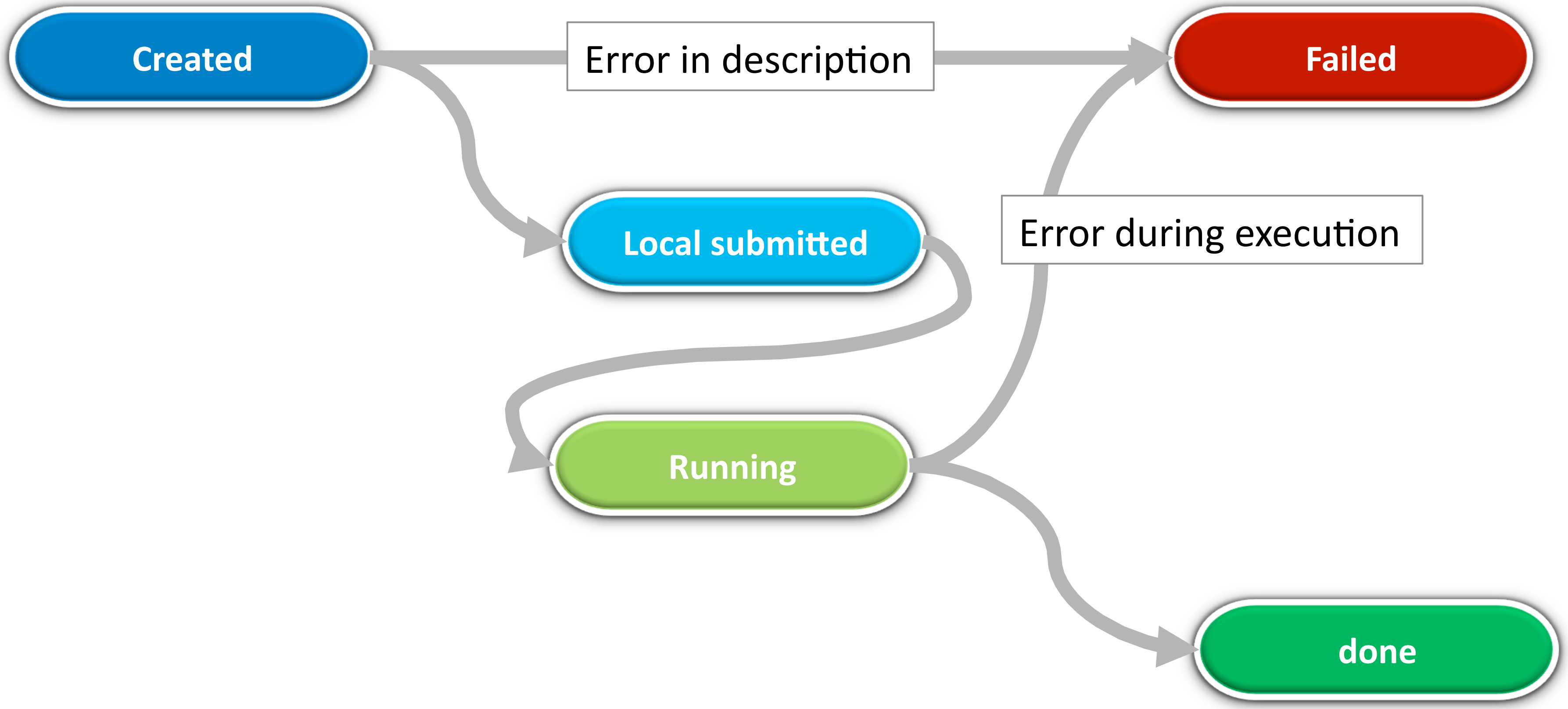
Overview

- Why XtreemOS is the OS to use
- What can I do with XtreemOS and how
- AEM internals
  - How is all this functionality implemented

AEM in XtreemOS

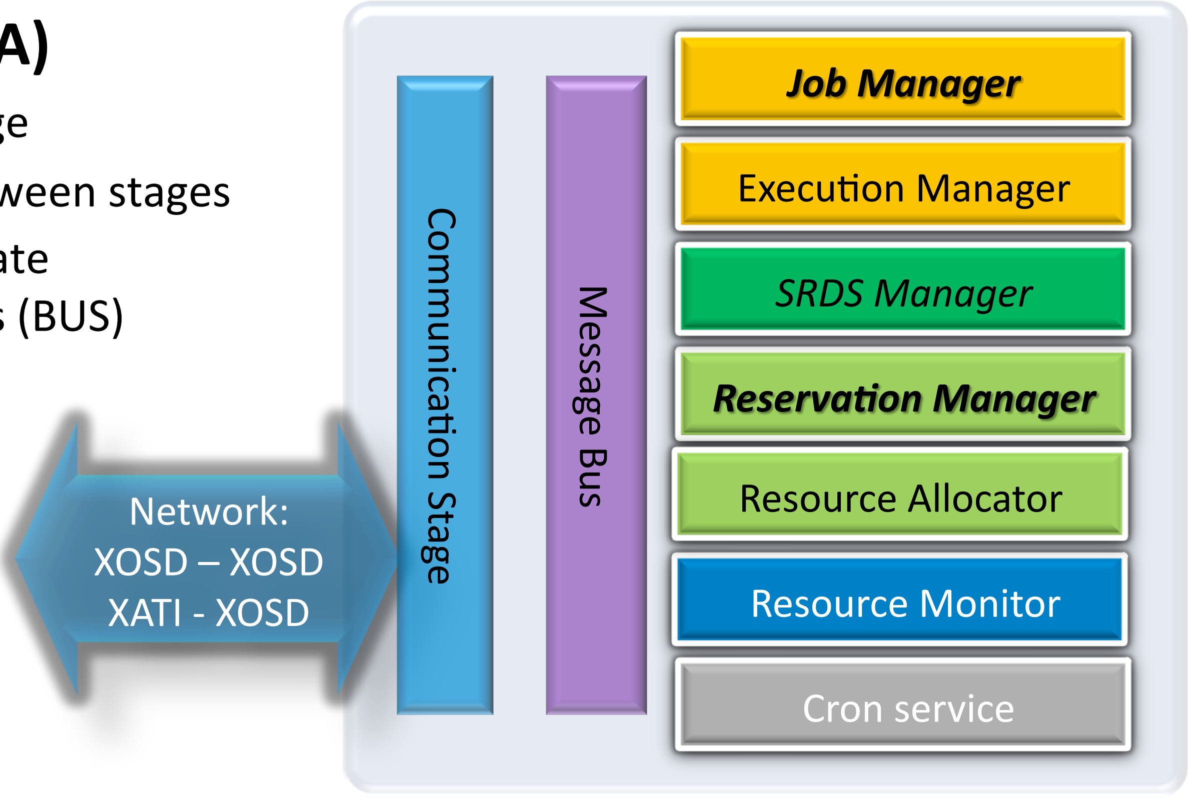

Grid Applications
User Software
XtreemOS API (SAGA & Posix-like)
VO & Security Management
Application Execution Management
Data Management
Infrastructure for Highly Available Scalable Services
XtreemOS-G
Extensions to Linux for VO Support
Linux
LinuxSSI
Embedded Linux
PC
Cluster
Mobile Device
XtreemOS-F

Job states

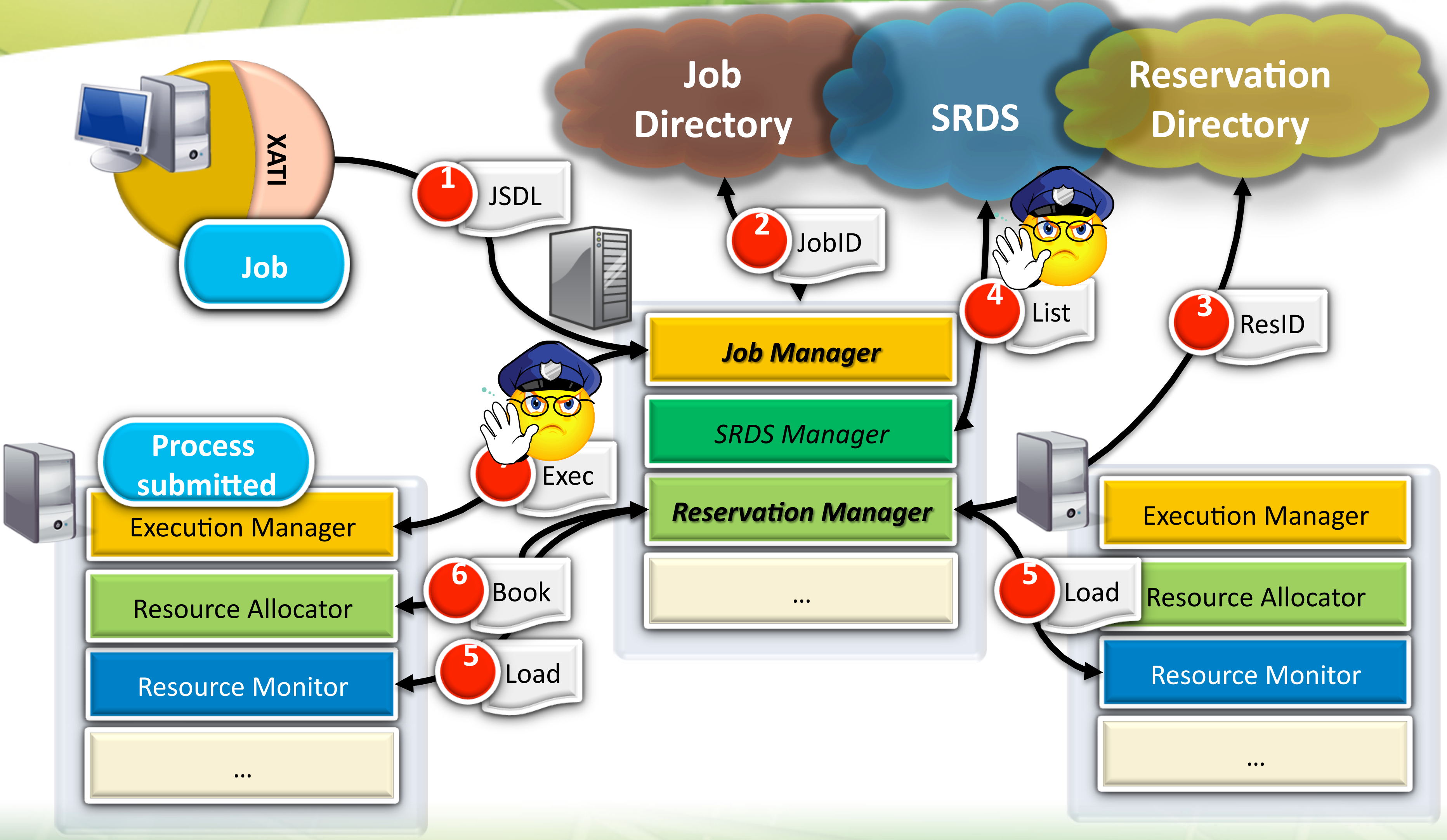

Created
Error in description
Failed
Local submitted
Error during execution
Running
done

XOSD architecture

- **Staged event driven architecture (SEDA)**
  - One thread per stage
  - No shared data between stages
  - Threads communicate via message queues (BUS)



Example: job submission



SRDS

- **Infrastructure**
  - Exploits multiple P2P overlays
    - Each resource and core node joins the overlays
- **How resources are discovered**
  - Three-pass filtering
    - Static checks (few attributes) performed by RSS overlay
    - Is node available? - XACML filters exploited on leaf nodes
    - Extensive and dynamic info is indexed within a DHT
- **DHTs also index**
  - heterogeneous/partially available data (e.g. JDS, ADS)

Reservations and time

- **Nodes may have different times**
  - Different time zones
  - Skews when setting the time
- **Solutions**
  - Times are converted to GMT+0
  - `Ntpd` required to synchronize times
    - Some skew will always exist
  - Threshold to queue sent jobs/procs
    - If a job/proc arrives a bit early, it is queued and started later
    - If a job/proc arrives too early an error is returned

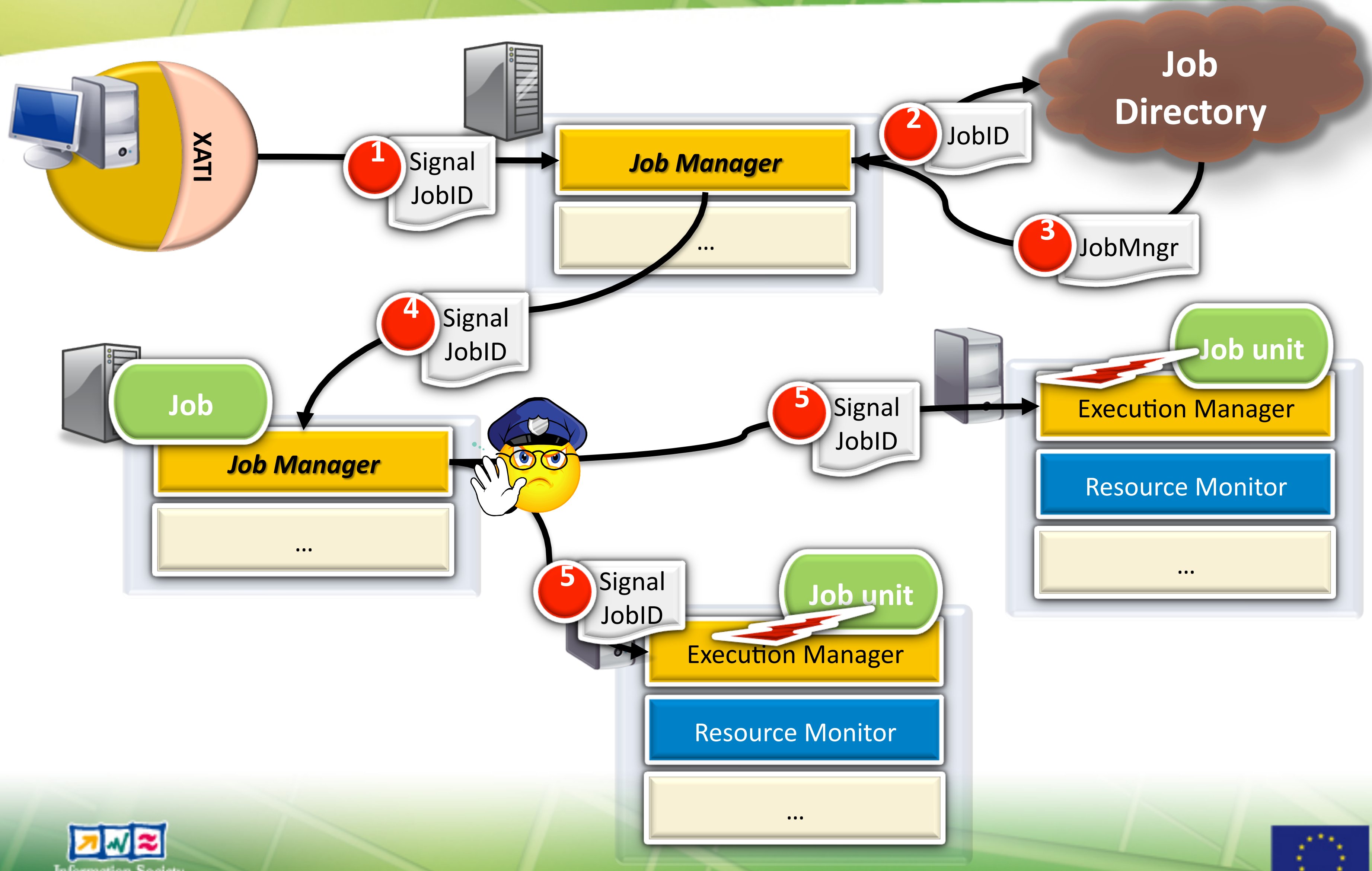
Scheduling algorithms

- **Algorithms (system configuration file)**
  - Random
  - Round robin
    - global on a per reservation basis
      - Several jobs may share a reservation
  - Load obtained during the negotiation phase
  - File closeness
- **Scheduling hints (user defined)**
  - Shared/exclusive
  - 1 process per node
    - Do not repeat node till necessary

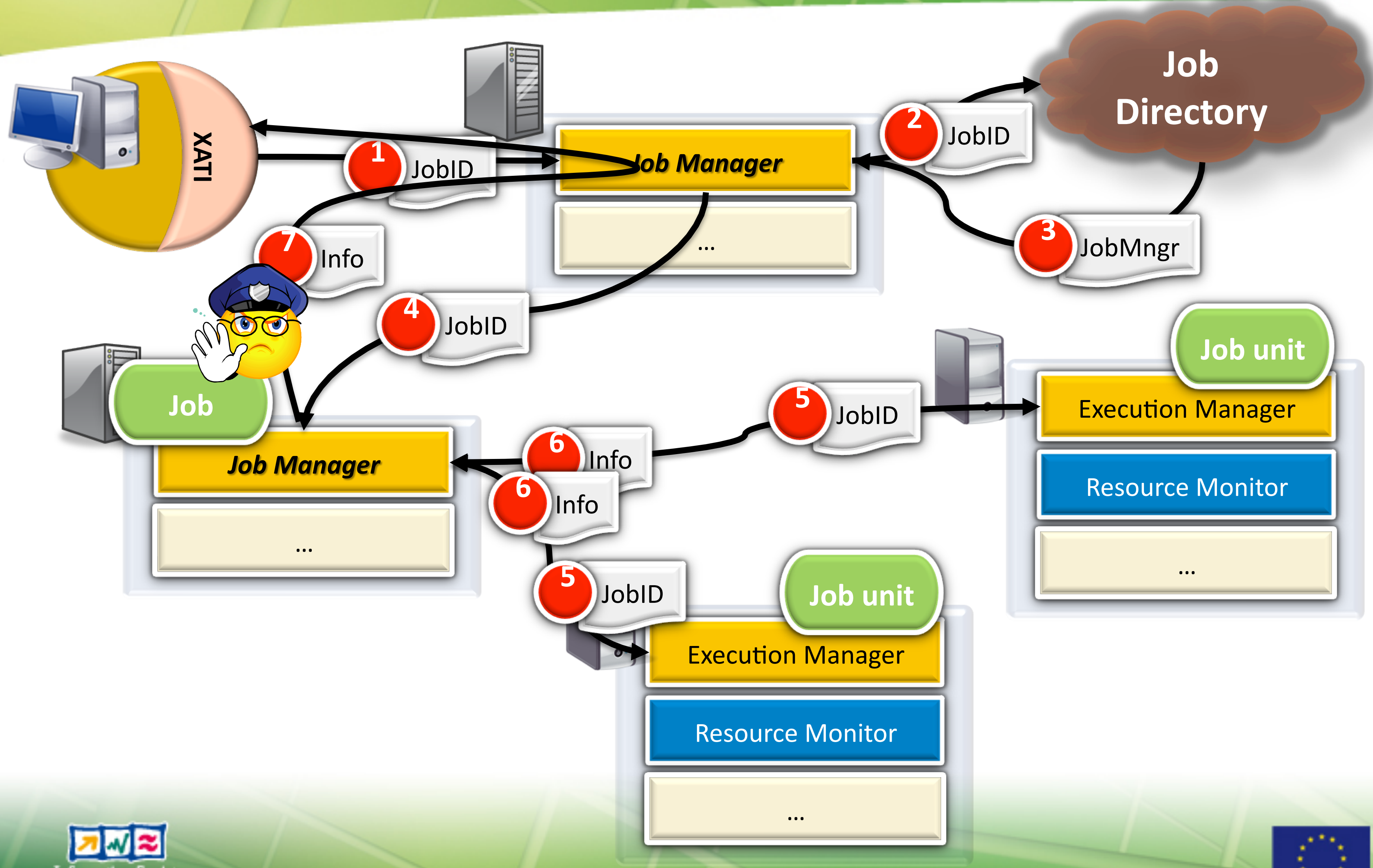
Interaction with FS

- **Cooperate to reduce**
  - File transfer and remote access
- **Background**
  - File system maps nodes in a 2D space (Vivaldi)
  - Exports the coordinates to SRDS
- **Two step cooperation**
  - Scheduler will request nodes “close to files” to SRDS
    - X,Y coordinates and a radium
  - Scheduler will inform of files to be used
    - File system will try to create replicas in advance (if possible)

Example: Job signal



Example: job information



Buffering

- **All monitoring events can be buffered**
  - Reduces monitoring traffic and overhead
  - Buffers have a finite size
    - Configurable per event: small, medium, large
  - If too many events, old events are lost
  - When events are read, the buffer is emptied
- **To reduce overwriting unread information**
  - Call back when the buffer is half full
    - Will be available as soon as call backs are available

Low-level monitoring

- **LTTng**
  - Linux Trace Toolkit new generation
  - Monitors kernel events
  - Also implements buffering
- **Best option for a detailed kernel information**
  - No kernel modifications needed in XtreemOS packages
- **Example**
  - Monitor thread/process status changes
    - Without LTTng it means kernel changes

Control of new processes

- **Control non XtreemOS events**
  - Forks done by a process do not go through XtreemOS
  - But… have to be known
- **Linux informs of these events via connectors**
  - Execution manager learns about them
  - Execution manager informs job manager
    - If necessary

Achieving scalability

- **Services have a job/resource view**
  - Exceptions: Job Directory
    - Implemented using DHT → scalable
  - Some times a few hops are needed
    - The performance price is reasonable
- **No global scheduler**
  - Schedule a job in a “good enough” way
  - Not make the best potential system schedule
    - It would be impossible → do not try

Fault tolerance

- **Fault tolerance**
  - Services keep no vital state
  - Exceptions:
    - Job and reservation Managers
    - Job Directory
- **Job and reservation managers**
  - Built on top of virtual nodes
  - Master/slave replication
- **Job directory**
  - Uses DHT replication mechanisms

Interactive jobs

- **When creating a job a helper is created**
  - To execute commands in the job context
    - Sets all the context needed for interactivity
  - Secured via SSH
  - The real application execution is requested to this helper

Future: integrate it with XOSD