Unprivileged subdivision of job resources

Within the ALICE Computing Grid

Maxim Storetvedt, *on behalf of the ALICE Collaboration* | CHEP 2024 | Kraków, PL | 23/10/2024

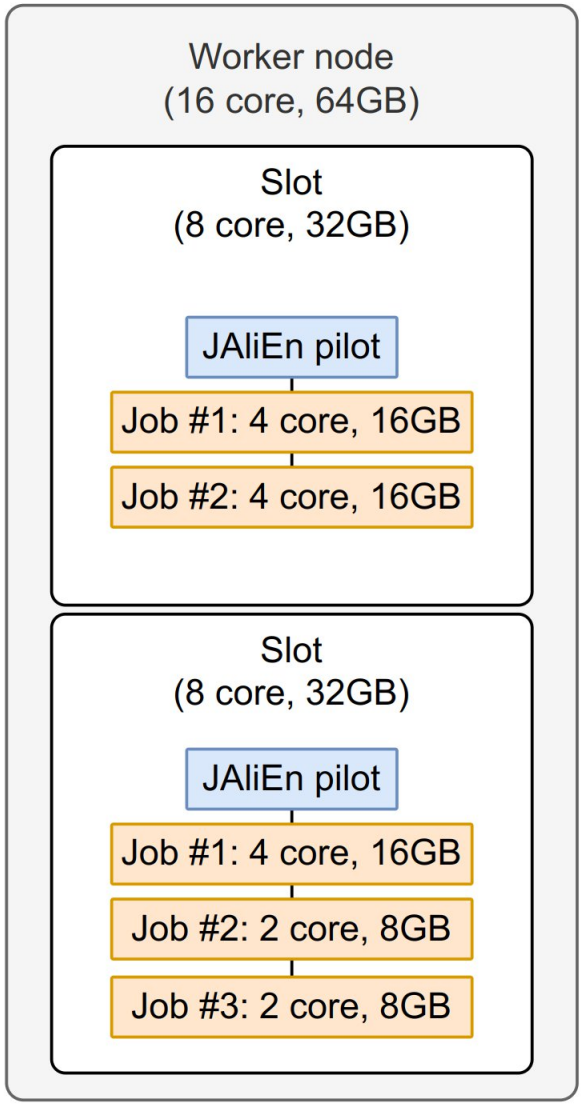
Running Jobs in the ALICE Grid

- 12k+ nodes, consisting of over 50 sites across 27 countries and regions
  - Just over 70k jobs[1] last week, using 210k+ cores
- Managed by the **J**ava **ALI**CE **En**vironment (**JAliEn**) Grid middleware
  - JAliEn job pilot fills a "**slot**" on each host (i.e. worker node), assigned by site resource manager
    - Various limitations apply : **CPU(s)** / **Memory** / **Storage**
- Each slot may have a **mix** of both single- and multi-core payloads **running in parallel**
  - Number of cores may vary, with 1, 2 and 8 core jobs being most common
  - Filled and managed by the job pilot to best utilise the slot resources
- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)

[1]Executable batch-type task

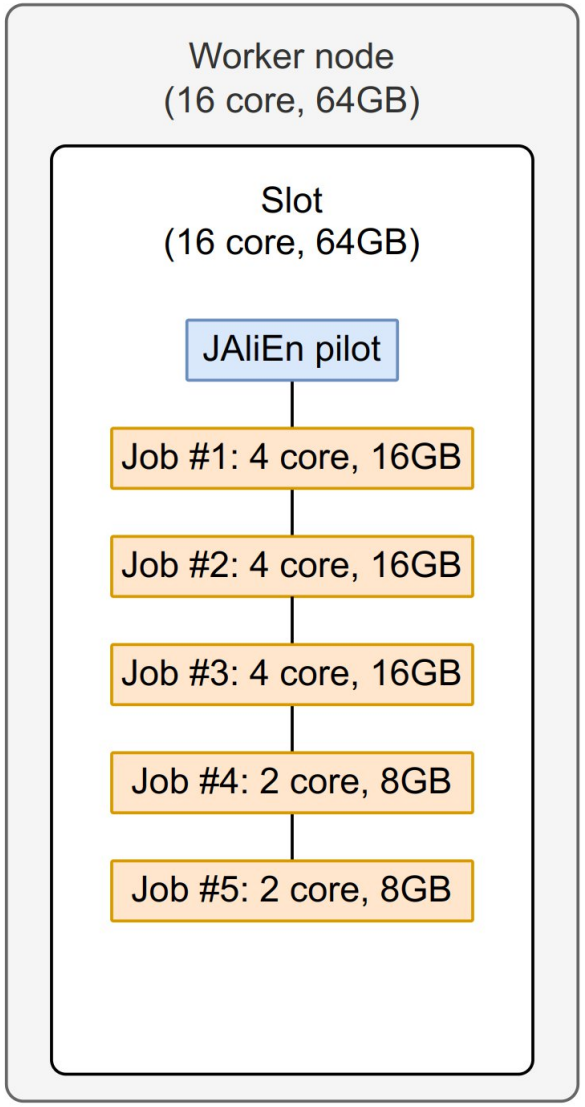
Inside a JAliEn job slot

- Each slot, **8+ cores** is managed by a job single pilot
  - Same pilot may host multiple jobs
  - The same worker node may also have multiple JAliEn slots



Inside a JAliEn job slot

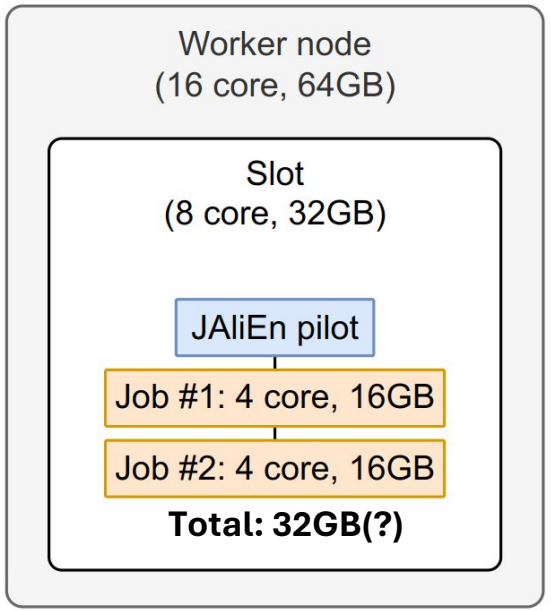
- Each slot, **8+ cores** is managed by a job single pilot
  - Same pilot may host multiple jobs
  - The same worker node may also have multiple JAliEn slots
- Increasing number of sites also offering **whole node** configurations
  - Slot encompassing all resources of a worker node
  - JAliEn pilot takes up full management responsibility
    - Allows for **better resource handling**



Managing misbehaving jobs

- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!

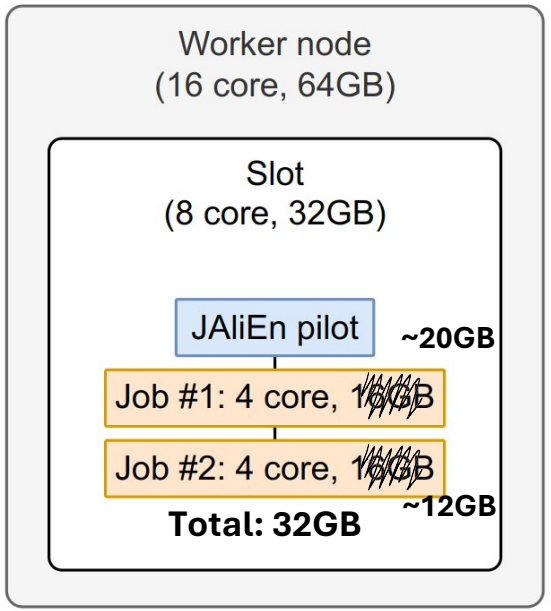
Scenario #1:
job overusing memory,
but total still *within slot*



Managing misbehaving jobs

- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!

Scenario #1:
job overusing memory,
but total still *within slot*

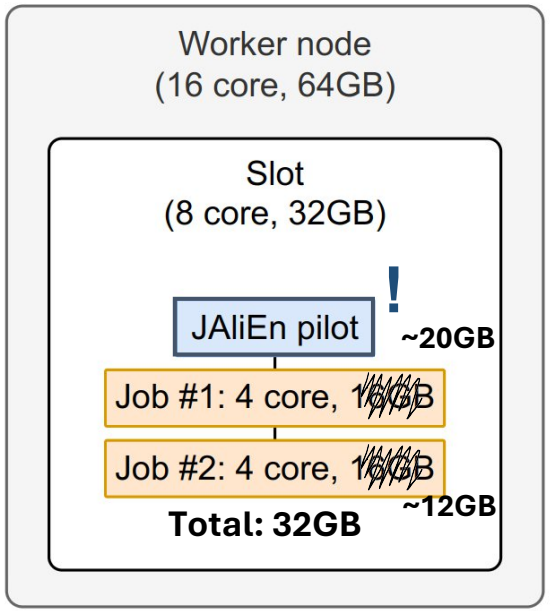


Managing misbehaving jobs

- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!

Scenario #1:
job overusing memory,
but total still *within slot*

- Detected by JAliEn pilot

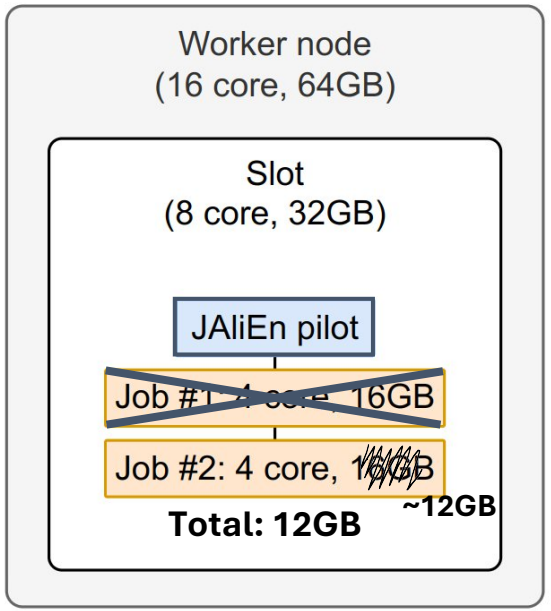


Managing misbehaving jobs

- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!

Scenario #1:
job overusing memory,
but total still *within slot*

- Detected by JAliEn pilot
- Misbehaving ***job*** killed

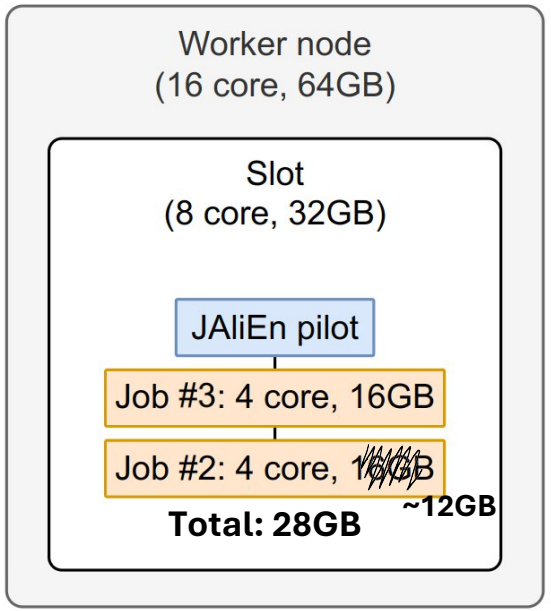


Managing misbehaving jobs

- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!

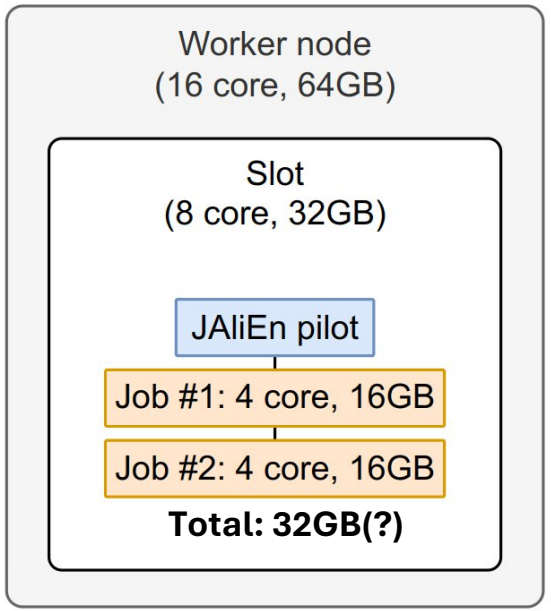
Scenario #1:
job overusing memory,
but total still *within slot*

- Detected by JAliEn pilot
- Misbehaving ***job*** killed
- Assigned new job(s)



Managing misbehaving jobs

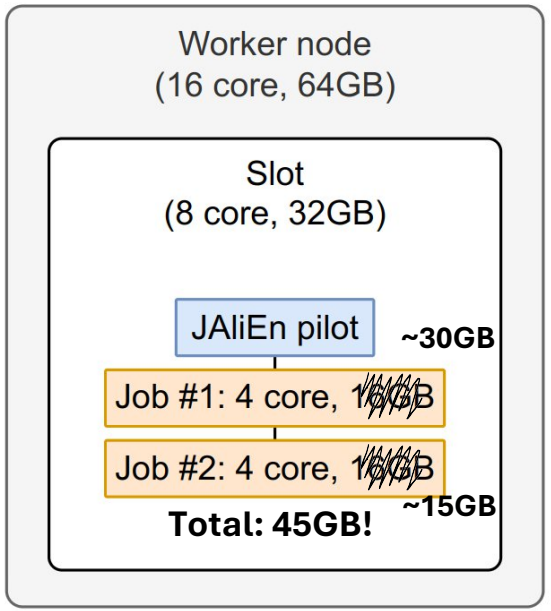
- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!



Scenario #2:
job overusing memory,
but total *beyond slot*

Managing misbehaving jobs

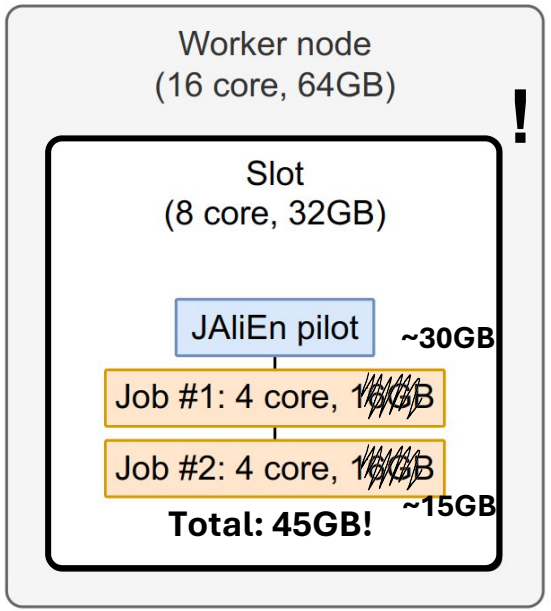
- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!



Scenario #2:
job overusing memory,
but total *beyond slot*

Managing misbehaving jobs

- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!

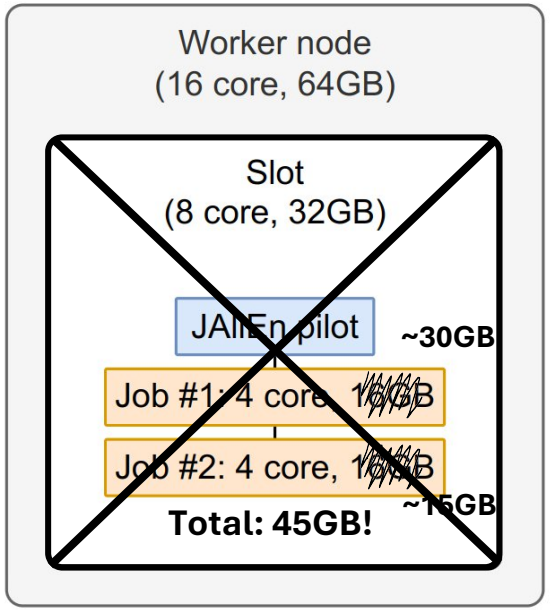


Scenario #2:
job overusing memory,
but total *beyond slot*

- Detected by resource manager

Managing misbehaving jobs

- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!

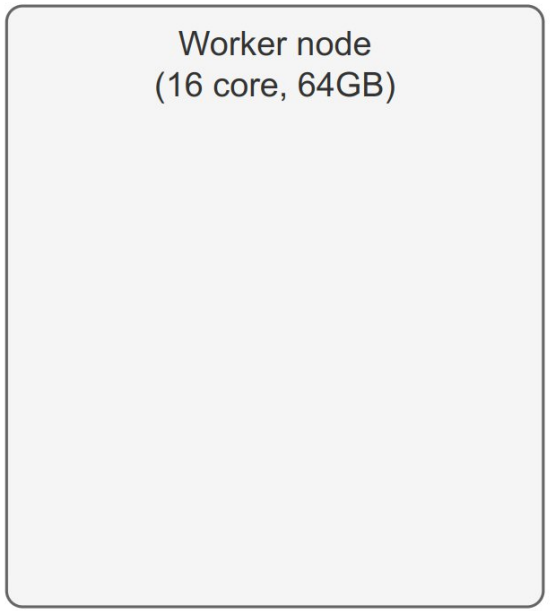


Scenario #2:
job overusing memory,
but total *beyond slot*

- Detected by resource manager
- ***Slot*** killed by resource manager

Managing misbehaving jobs

- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!

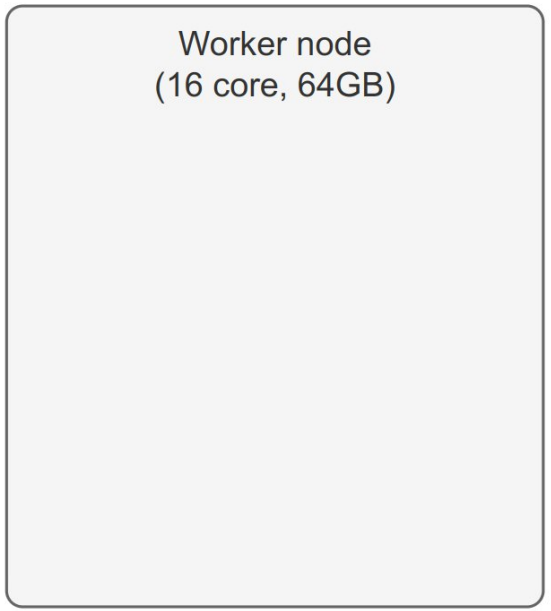


Scenario #2:
job overusing memory,
but total *beyond slot*

- Detected by resource manager
- ***Slot*** killed by resource manager
- ***All jobs*** are lost!

Managing misbehaving jobs

- Jobs overusing slot resources are automatically **killed**
  - Either by JAliEn, or by other safeguards (local resource manager, e.g. HTCondor / Slurm)
- But the consequences are **very different** depending on **who/what** called the kill!



Scenario #2:
job overusing memory,
but total *beyond slot*

- Detected by resource manager
- ***Slot*** killed by resource manager
- ***All jobs*** are lost!

Often consequence of arbitrary upper limits set by users

The need for better resource management

- While **single-core** was the norm:
  - 1 slot = 1 job
  - Minimal impact if job killed by agent or something else
- This has **changed** with modern **multi-core** workflows:
  - Misbehaving jobs, *detected and killed by the JAliEn pilot*, remain **low impact**
    - Only offending job terminated
    - Slot refilled
    - No interruption to other jobs
  - But misbehaving jobs, *detected and killed by a resource manager*, have **very high impact**
    - Slot is terminated
    - Offending job within is **killed**, but so are **all other jobs**
    - Can be disastrous on **whole-node** slots

Towards better management

- A number of tools available within Linux, such as *"taskset"*
  - Now used by the pilot to better constrain CPU resources through ***CPU pinning*** [1]
- But majority of Linux utilities for resource management require **root**
  - This changes with the recent introduction of **Control Groups (Cgroups) v2** in Linux
  - Allows for delegating per-process resource controls to unprivileged users…*

Controller	Can be controlled by user	Options
cpu	Requires delegation	CPUAccounting, CPUWeight, CPUQuota, AllowedCPUs, AllowedMemoryNodes
io	Requires delegation	IOWeight, IOReadBandwidthMax, IOWriteBandwidthMax, IODeviceLatencyTargetSec
memory	Yes*	MemoryLow, MemoryHigh, MemoryMax, MemorySwapMax
pids	Yes*	TasksMax

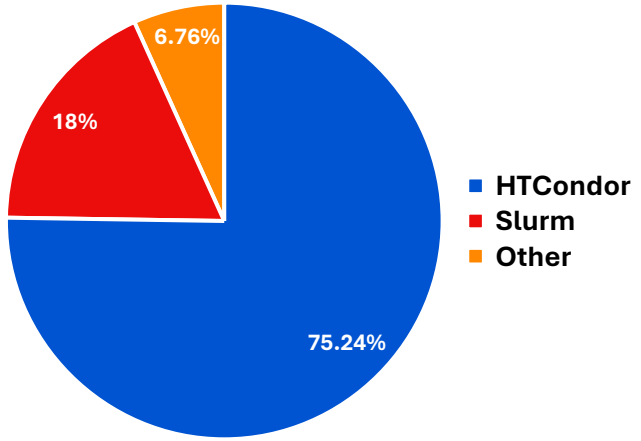
Limitations of unprivileged cgroups v2

- Fully unprivileged cgroups generally run inside the **user.slice**
  - Permissions/ownership set up automatically during user login
  - But this does **not apply** to users in a **batch slot**
- **Unavailable** to non-interactive users
  - Including user in slot running the JAlien pilot
- Unless, the user is given ownership of
  - The **cgroup** — i.e. its top-level directory in /sys/fs/cgroup
  - The **cgroup.procs** file — to allow moving processes in/out of it
  - The **cgroup.subtree_control** file — to allow delegation of controllers to subgroups

…but, is this not done by CE/LRMS when slot (and its cgroup) is created?

Status of unprivileged cgroups v2 in common CEs

- **SLURM**
  - New cgroup/directory created on each new job for slot
  - Cgroup ownership set to that of the executing user — **great!**
    - But **only** cgroup ownership. All files inside still owned by root
- **HTCondor**
  - New cgroup/directory created on each new job for slot
    - But **all** files/directories owned by root



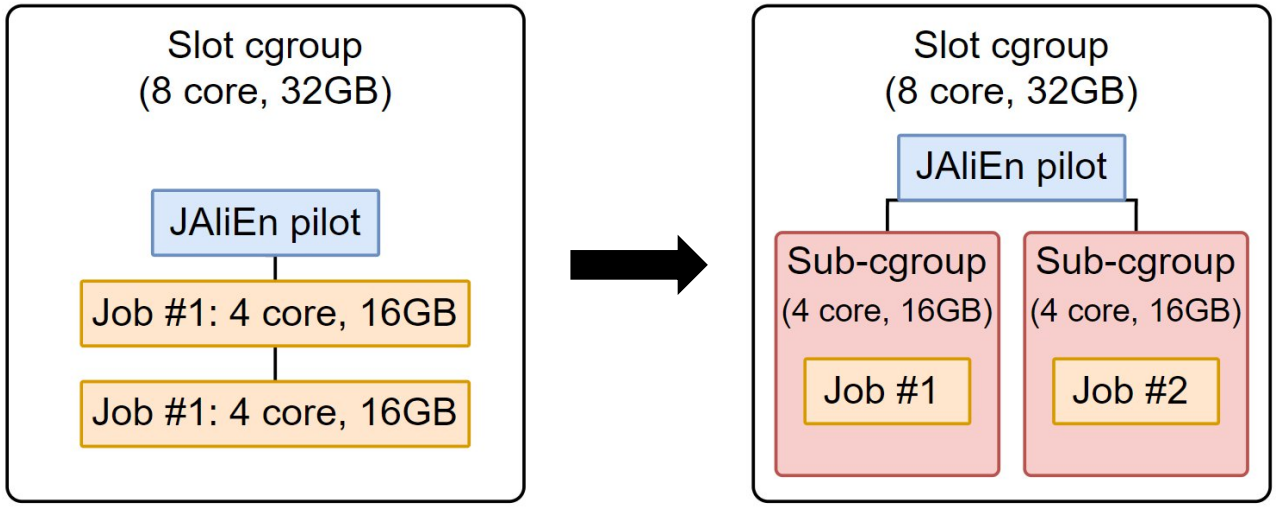
Distribution of LRMs in the ALICE Grid, Oct. 2024

Workaround: a custom cgroups v2 plugin

- Proof of concept Cgroups v2 plugins created for SLURM & HTCondor
  - Sets the appropriate permissions, and checks for subgroups in cleanup
- Tested with custom jobs that
  - Checked for given privileges
  - Attempted creating subgroups, move processes and apply limits on them
  - Attempt breaking process limits
- Fully working, even with an **unprivileged user**
- Change **upstreamed** and included in **HTCondor 23.1** [2]
- Workarounds still required for Slurm

New Cgroups v2 features within the job pilot

- Unprivileged cgroups open up new possibilities for the JAliEn job pilot
- Can be used to **box-in** and **subdivide** a slot
  - Cgroup controllers available for **CPU**, **IO**, **Memory**
  - Each job may then run in a smaller subpartition of given resources
    - Precise control over job resource use, without overreaching



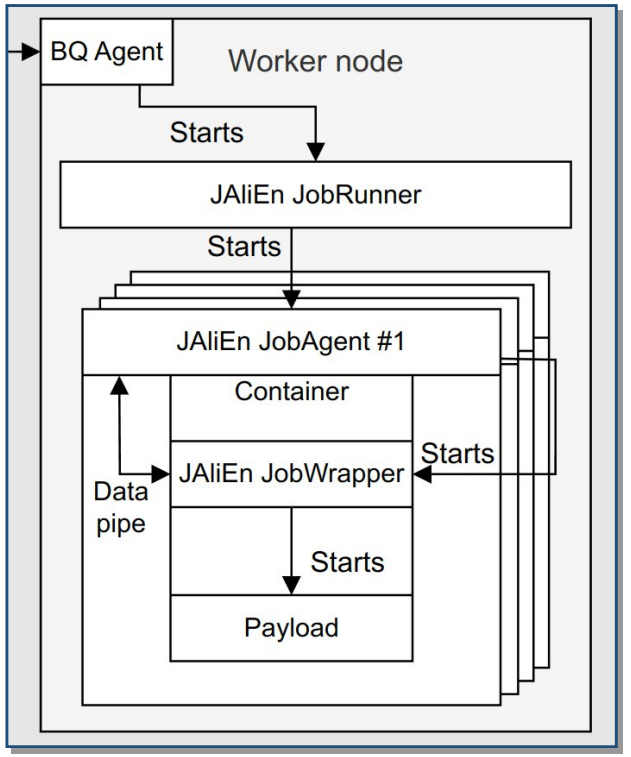
New Cgroups v2 features within the job pilot

- Unprivileged cgroups open up new possibilities for the JAliEn job pilot
- Can be used to **box-in** and **subdivide** a slot
  - Cgroup controllers available for **CPU**, **IO**, **Memory**
  - Each job may then run in a smaller subpartition of given resources
    - Precise control over job resource use, without overreaching

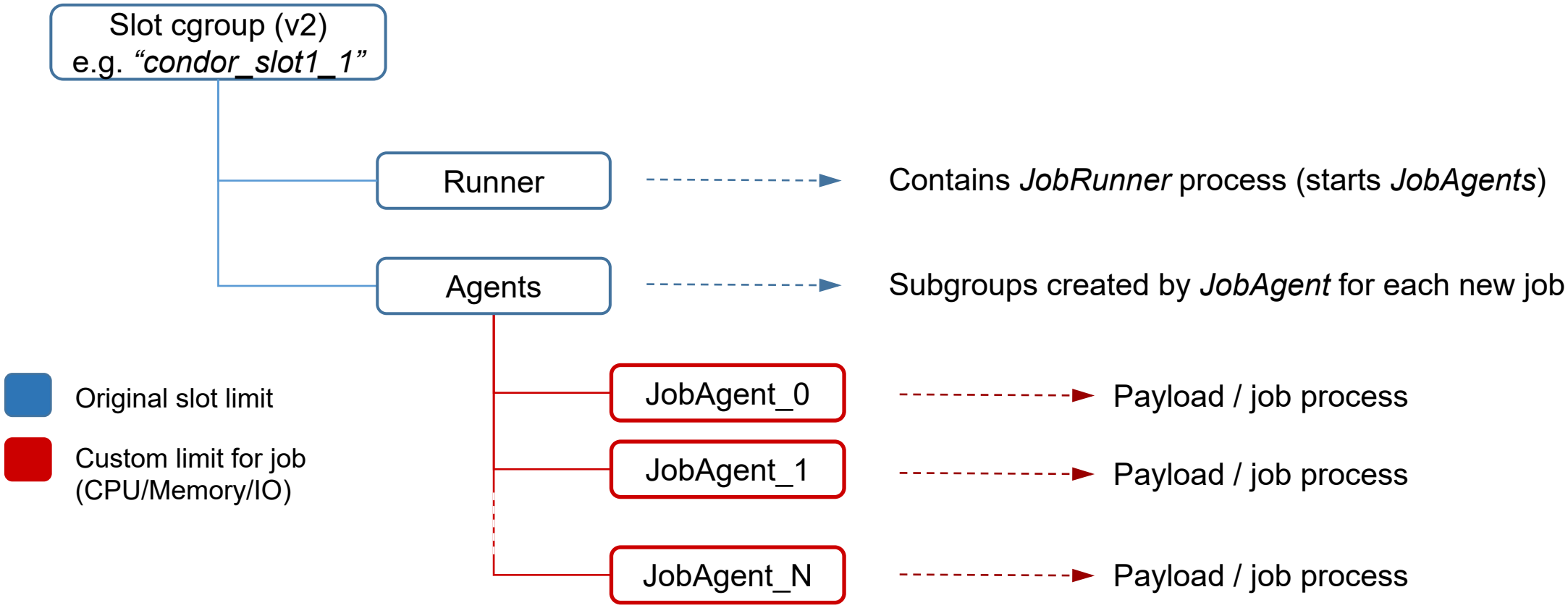
- As long as all required controllers are properly **delegated** down the cgroup tree!
  - HTCondor will create a new cgroup with correct ownership
  - But setup and management of new subgroups must instead be done by **the job pilot**

New Cgroups v2 features within the job pilot (2)

- Each JAliEn pilot consists of three components:
  - JAliEn **JobRunner**: Resource/multicore handler
  - JAliEn **JobAgent**: Job matcher/monitoring handler
  - JAliEn **JobWrapper**: Payload executor
- The JobWrapper runs on a separate JVM
  - Handles payload that can be several cores per job slot
- Pilot logic adjusted to accommodate for cgroups v2 limits:
  - Top level slot cgroup creation and delegation of controllers
    - Via **JobRunner**
  - Job cgroup creation and limits
    - Via **JobAgent**



New Cgroup tree for JAliEn slots



Subdividing a slot cgroup

- Controllers may only be delegated down **empty** cgroups
  - JobRunner and initial bootstrap procs must be moved somewhere first
  - New cgroup created for this purpose: "***runner***"
  - New cgroup is also created for where to place each job in the same step: **"agents"**
    - While still empty, delegate the controllers from previous step down to this group
- Once controllers in place, each agent will impose a **resource limit** on **each new subgroup**
    - **Partitioned** depending on **free resources remaining** in slot
    - Ideally, equivalent to what is requested by job
    - But **never more than total** of free resources in a slot

Summary

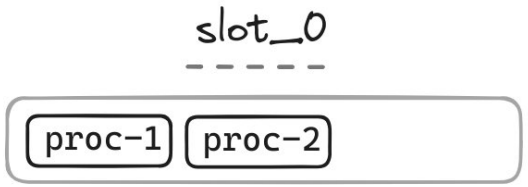
- The shift to **multicore** has given job pilots **more responsibility** in resource management
  - An increasingly challenging process, as misbehaving jobs **risk being killed** by slot
  - Exacerbated by several payloads in same slot, and arbitrary requirements set by users
- **Cgroups v2** provide means for better resource control, **unprivileged**
  - But generally unavailable to non-interactive batch users
- Steps taken to **enable** use of unprivileged cgroups v2 in ALICE Grid
  - Changes in upstream **HTCondor** for appropriate group ownership in slot
  - Group creation and **delegation of controllers** by JAliEn pilot
- Combined, allows the resources of a slot to be **subdivided** into smaller “partitions”
  - Each with a **custom resource limit** appropriate for its job
  - Prevented from going above slot, and may easily be managed by JAliEn

Backup

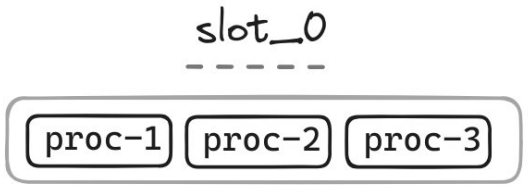
Subdividing a slot cgroup

Subdividing a slot cgroup

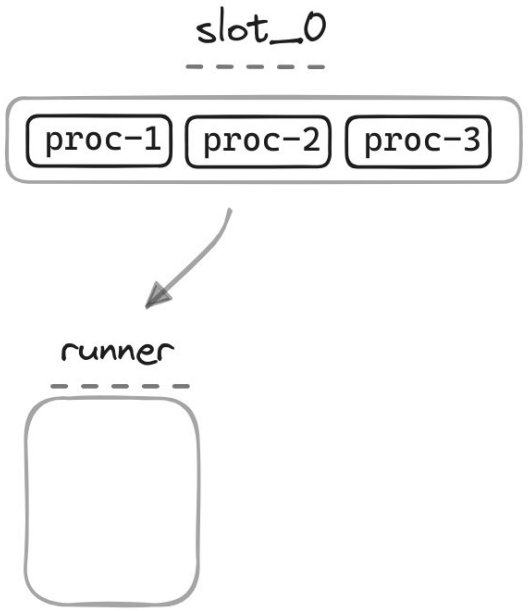
Subdividing a slot cgroup



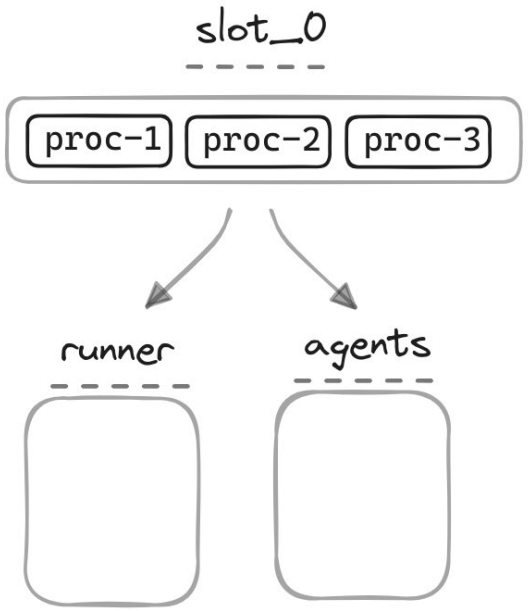
Subdividing a slot cgroup



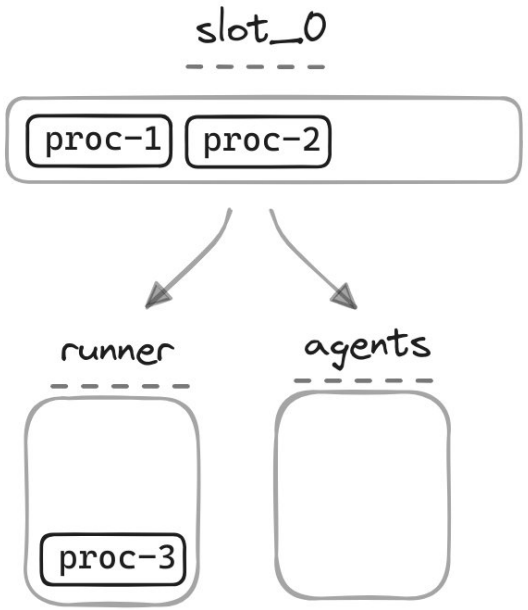
Subdividing a slot cgroup



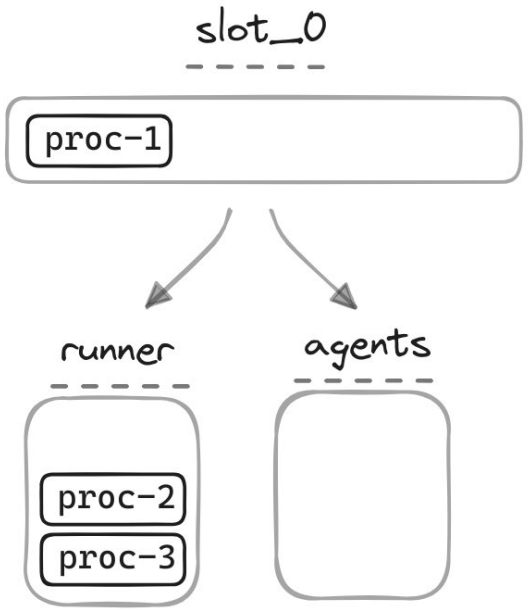
Subdividing a slot cgroup



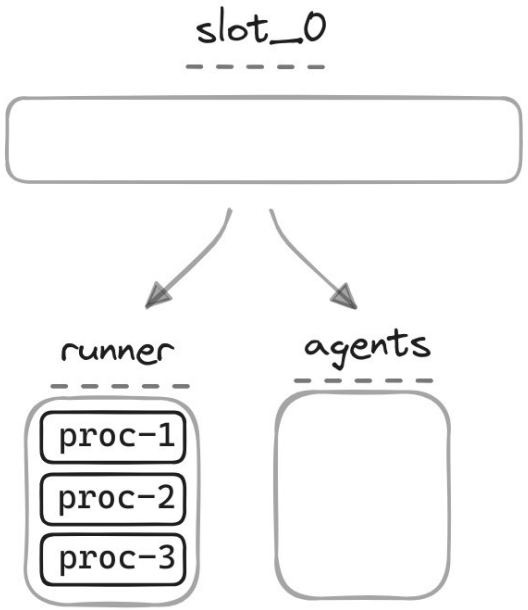
Subdividing a slot cgroup



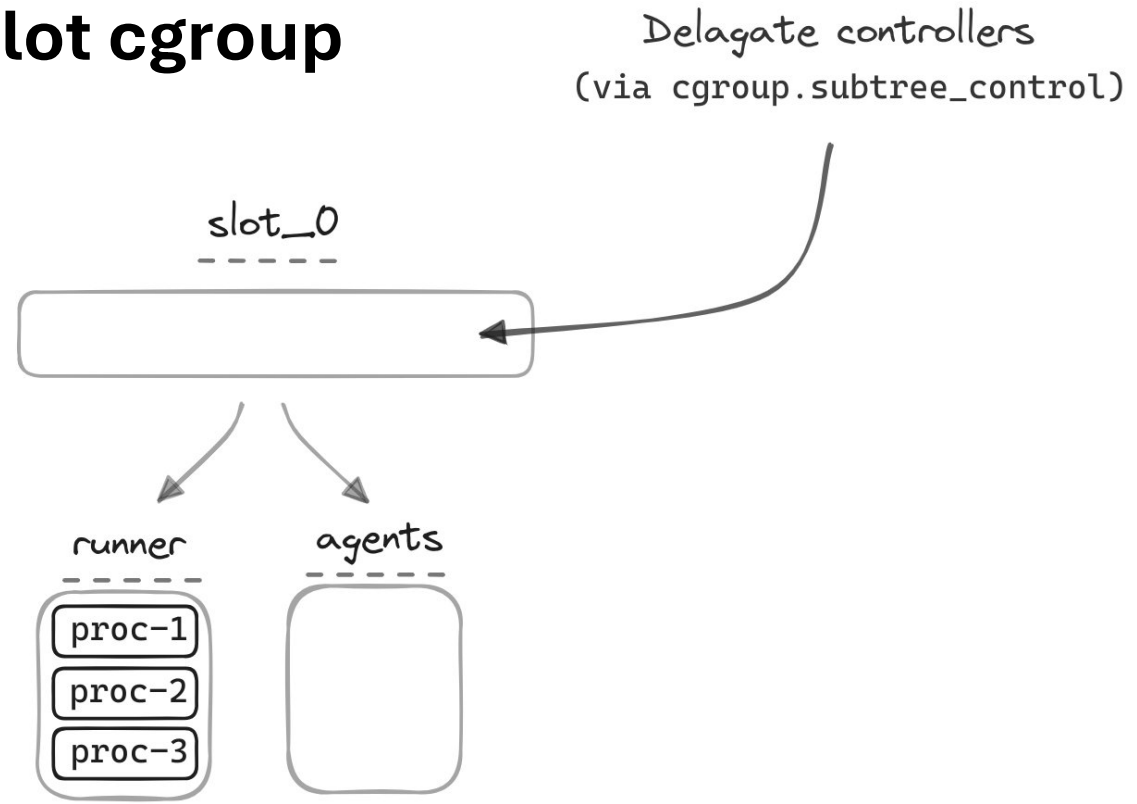
Subdividing a slot cgroup



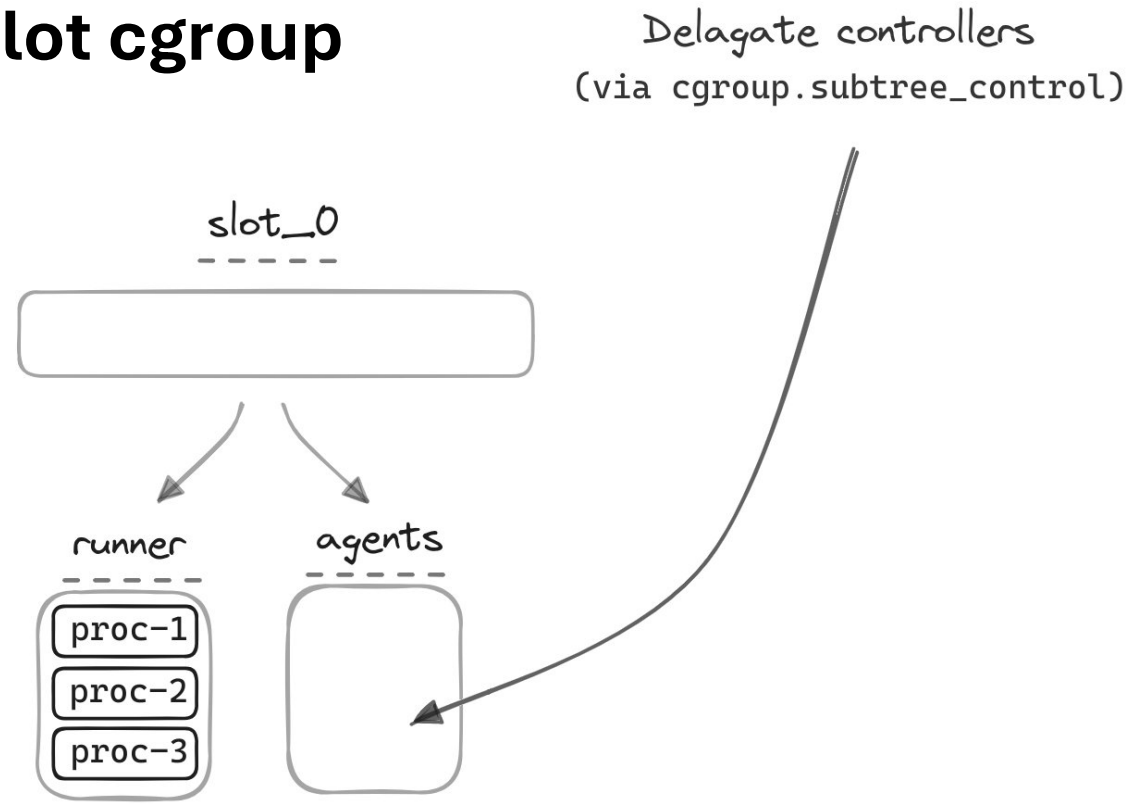
Subdividing a slot cgroup



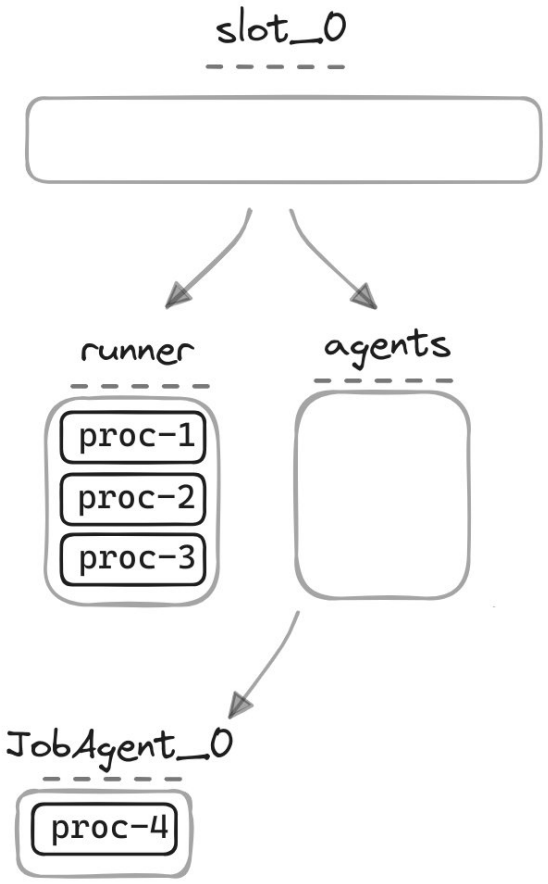
Subdividing a slot cgroup



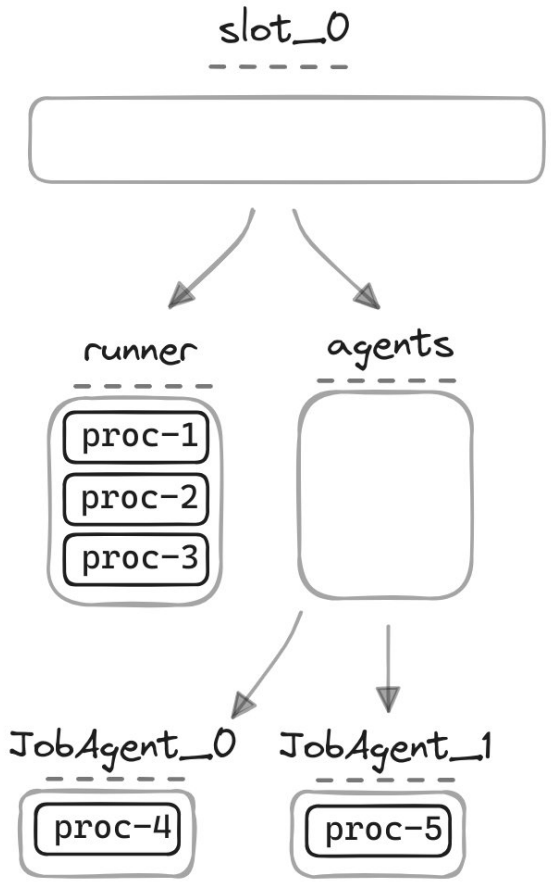
Subdividing a slot cgroup



Subdividing a slot cgroup



Subdividing a slot cgroup



Subdividing a slot cgroup