System tools for payload control

Marta Bertran Ferrer
marta.bertran.ferrer@cern.ch

Alice T1/T2 Workshop
16-18 April 2024

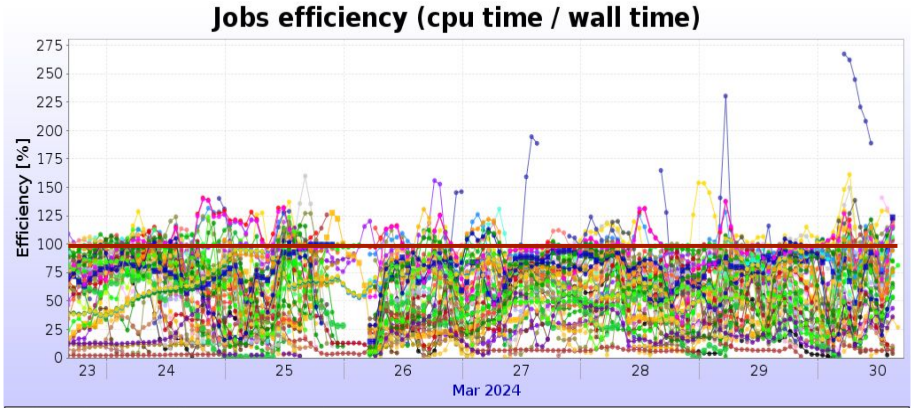
Current situation on Grid worker nodes

- Resource overconsumption is a recurrent issue in the Grid landscape
  - **Memory** and **CPU** are our main concerns
  - Leads to job interference and termination
- **Grid heterogeneity** on available resources and constraining policies
  - Custom configurations and tools per site
  - Some jobs can only complete with big resource allocations / extendable limits
- Take into account site policies and perform custom actions to **maximise efficient use of resources**
  - Imposing usage limits once the job starts (static and uniform)
  - Letting jobs freely expand and act when reaching site-specific policies (dynamic)

ALICE

CPU control strategies

- Many sites do not constraint CPU allocation
  - Leads to unpredictable execution time, even for the same CPU type and payload
  - As a consequence, the turnaround time varies considerably and may lead to job termination
- Special concern when running in **whole-node scheduling**
  - The resources must be allocated as requested by the job



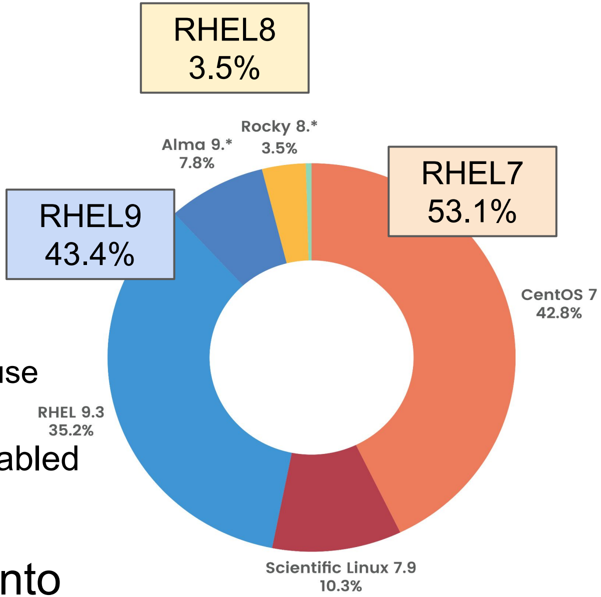
CPU pinning

- CPU pinning using `taskset` is **already in production** running in a set of sites
  - Available tool in all linux flavors, CentOS7 included
- Core selection based on **NUMA architecture** of the executing node to promote data locality
- In sites with fixed-size slots, JA pinning coordination by communicating with Central Services
  - As a best-effort attempt to avoid co-pinned cores
- Presently CPU pinning is using a **static approach**
  - CPU usage is always kept under the user requested number of CPU cores
  - Ensuring fairness with other co-executing payloads
  - In whole-node or multi-core scenarios, **unassigned cores are shared** between running workflows. If slot is not full, job efficiencies can be higher than 100%
- **No negative impact** on CPU efficiency of well-behaved jobs has been observed

ALICE

Using *cgroups v2* for configuring limits on CPU

- ***cgroups v2*** to control workflow **resource allowance**
  - With different controllers to manage different resources (CPU, IO, memory, PIDs…)
- Not supported by all operating systems, need to satisfy multiple requirements (configuration and versions)
  - Grid current default OS (CentOS 7) does not support *cgroups v2*
    - Expected growth this year when 7 reaches EoL, pushing sites to use RHEL 9 for a complete feature set in cgroups v2
  - Few sites running with OSs that support it (+ has to be explicitly enabled by site admins)
- Lets unprivileged users **divide the granted resources** into new cgroups
  - Allocation of **pre-defined portion** of the original resources to workflows

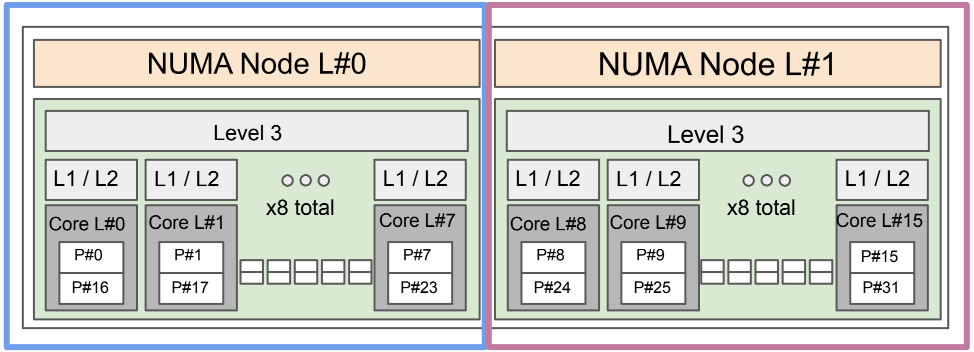


Distribution of OS versions among Grid hosts

Cgroups v2 integration in JAliEn

- Most popular batch systems (Slurm and HTCondor) can already enable **rootless fragmentation** into smaller sub-slots
  - HTCondor built-in feature since version 23
  - Plugin available for Slurm implemented for this use-case
  - Each of the running jobs will have its own constraints
- Batch queue sets a general *cgroup(v2)* to the granted slot
  - **Root *cgroup***, big box with general limits
  - Its limits and custom configurations are set by site admins
- Depending on batch allocation will use different CPU limiting options:
  - **Slot of limited amount of cores:** Usage of `cpu.max` for allocating a maximum bandwidth limit proportional to the job's requested cores
  - **Whole-node scenarios:** Usage of `cpuset.cpus` for exploiting reduced memory access latency with explicit core selection
    - The whole machine is occupied by our processes - full visibility of the pinning status
    - Profit from the already-implemented core-selection given the NUMA architecture

CPU constraining in whole node scenarios



Flexibility of allocations
Exploiting **memory locality**
Promote **fairness** among co-executors

In whole node scenarios we can:

- Assign jobs to **all** the cores of **one NUMA domain** using cgroups v2 (***cpuset.cpus***) or `taskset`, depending on the machine's availability
- If running with *cgroups v2*, imit their CPU bandwidth tuning ***cpu.max***

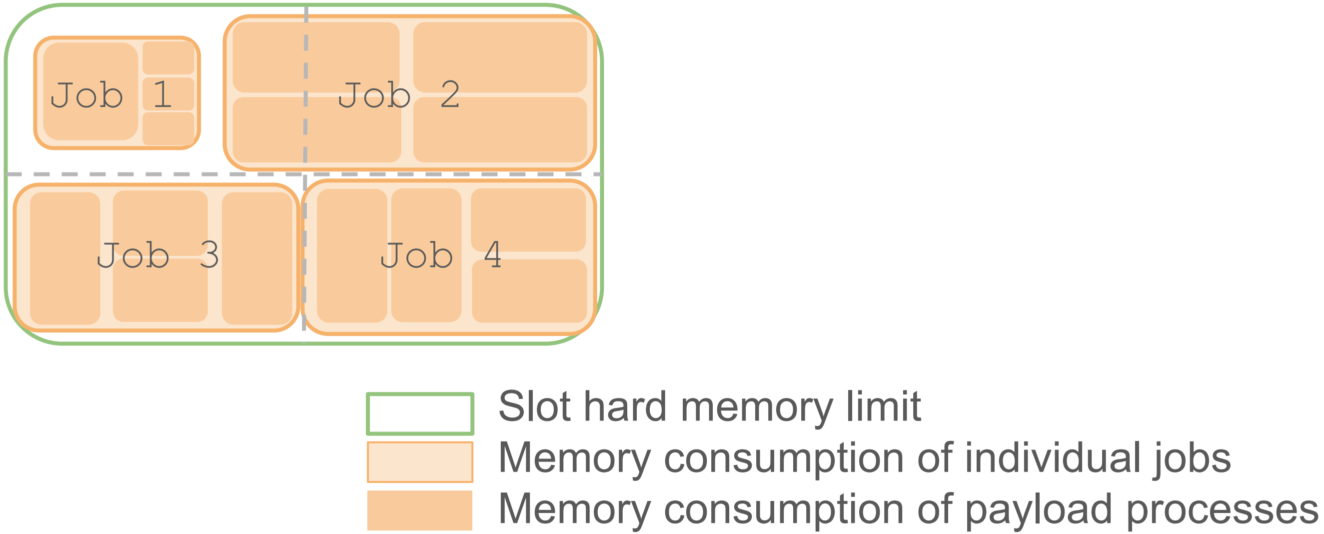
Memory - Site configurations landscape

- Heterogeneous memory capacity and memory management on the Grid
  - Sites provide **minimum** 2GB RAM per core, more if available
  - Machine memory usage **fluctuates**, dynamically increasing/reducing the available RAM
- Allocation **limits on memory** resources bound by:
  - Physical machine specs
  - Constraining policies applied to global machine, batch system slots or by kernel OOM

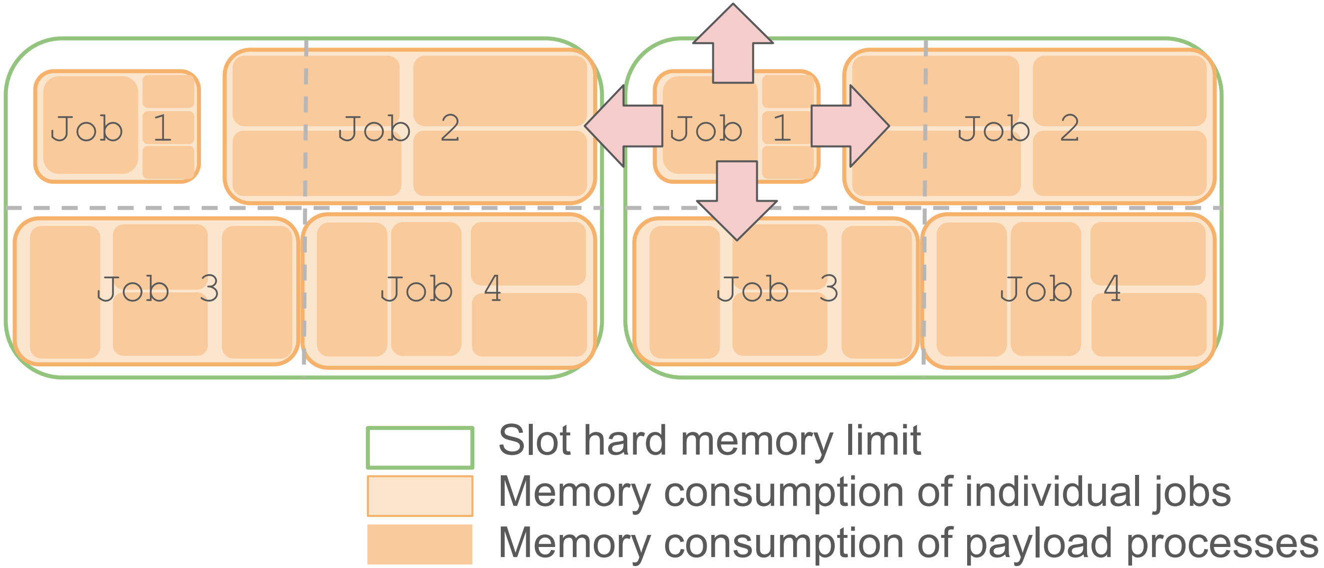
First approach: Using *cgroups v2* for limiting memory

- `memory.low`: if cgroup and all its descendants are below this threshold, the cgroup's memory **won't be reclaimed** (best-effort guarantee)
    - Initial idea: Guaranteeing that our jobs are granted (at least) **2GB/core**
- Experimenting a **test scenario** by setting hard memory limit in parent *cgroup* and *memory.low* in the descendants, running a set of processes each
    - Main observation: Instead of the global cgroup memory, the **individual processes memory usage** is the factor that **triggers the payload termination**
        - *memory.low* limit is **not taken into account** when used together with *memory.max* on parent cgroup
        - Once the *memory.max* limit is reached, the **process that consumes the most** is killed
          As the relative size of it is ignored (number of cores), the wrong process might be picked up

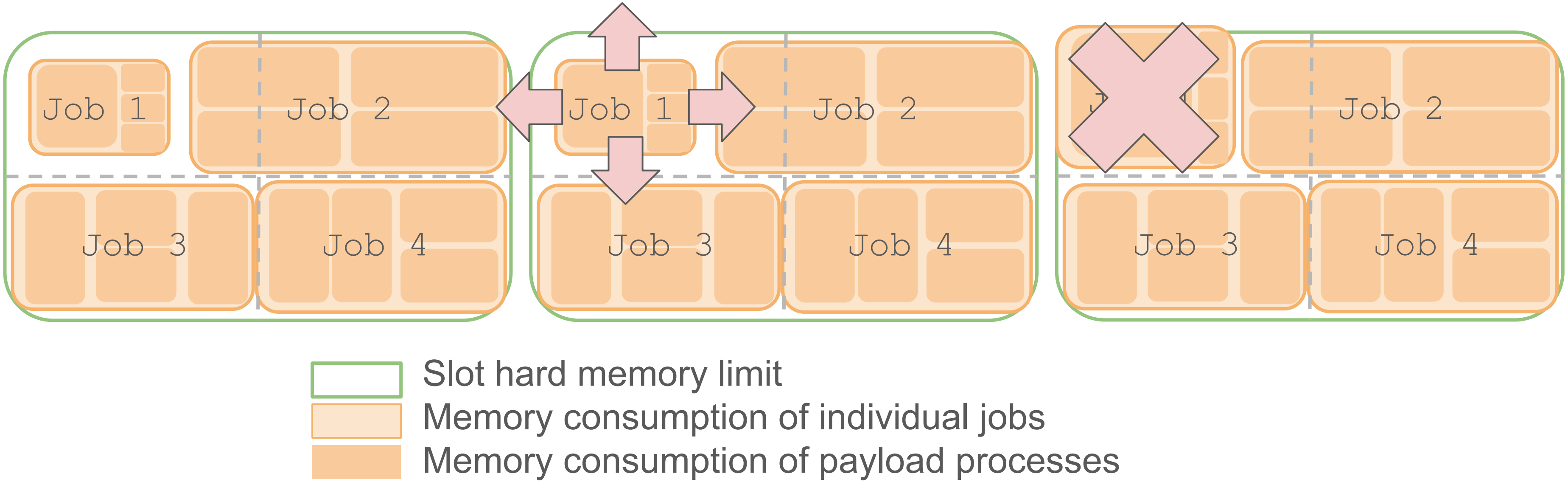
First approach: Using *cgroups v2* for limiting memory



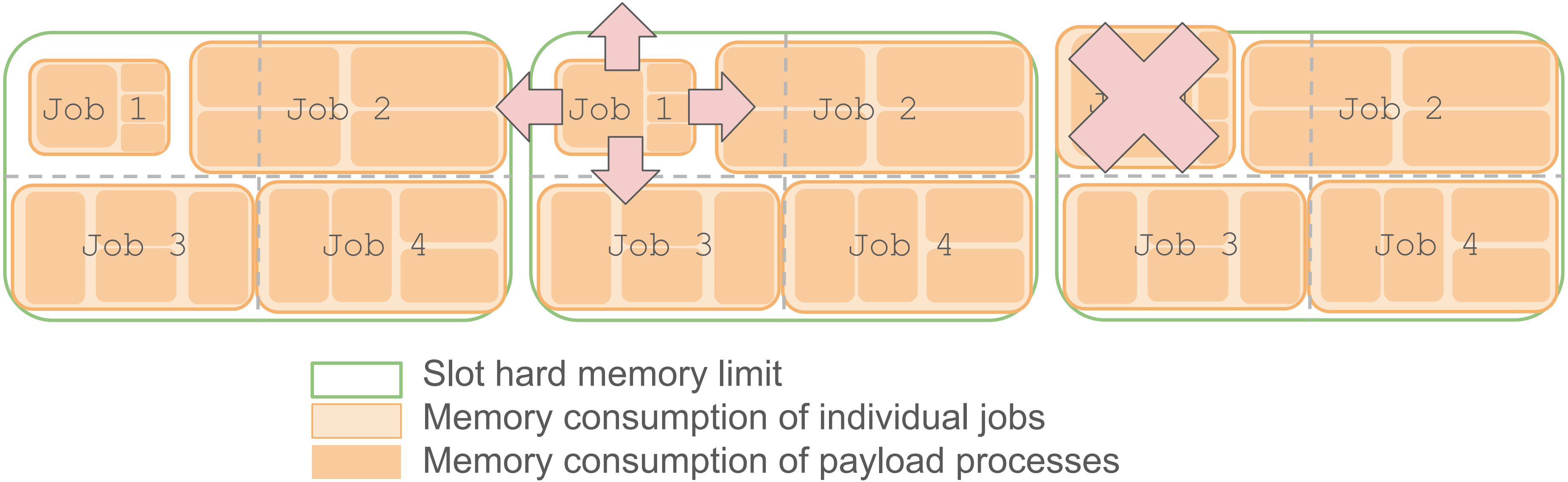
First approach: Using *cgroups v2* for limiting memory



First approach: Using *cgroups v2* for limiting memory



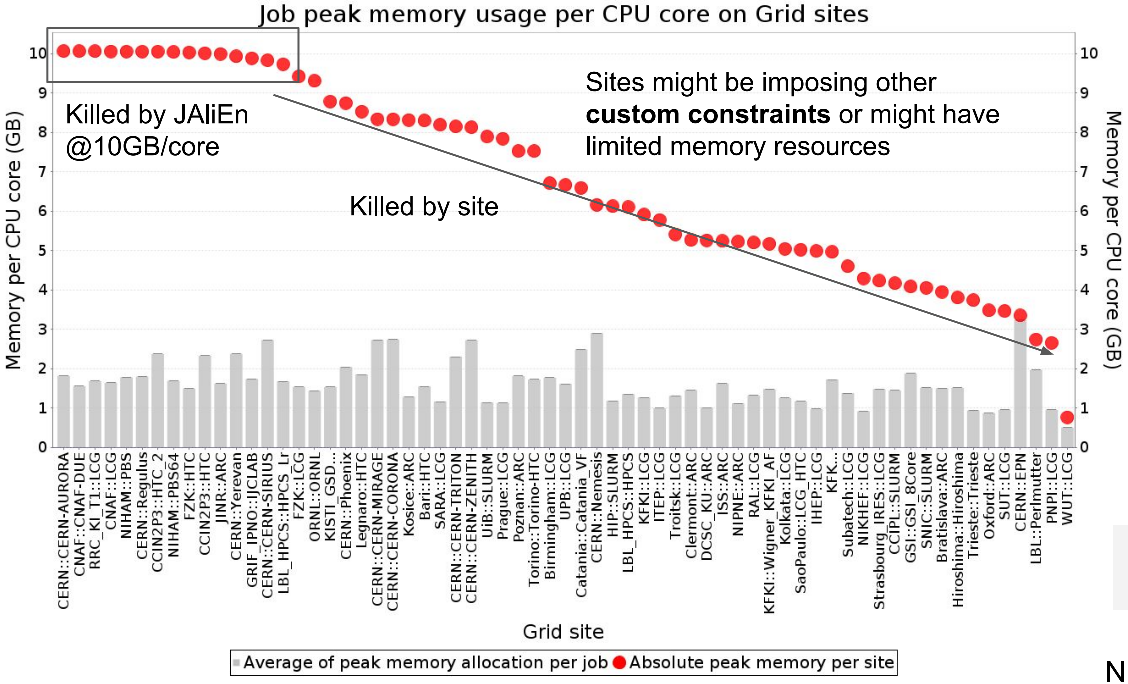
First approach: Using *cgroups v2* for limiting memory

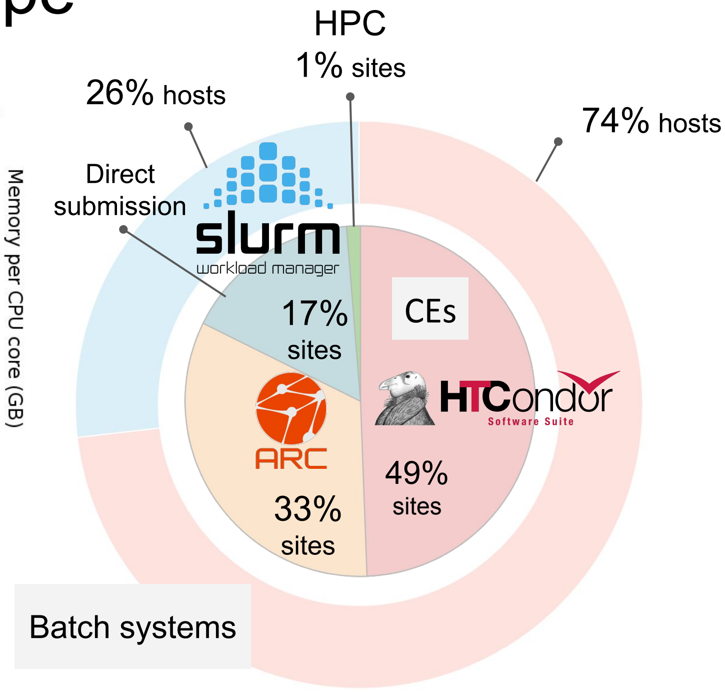


- For the moment, we **can not use** *cgroups v2* as we envisioned for constraining jobs in memory

ALICE

Memory - Site configurations landscape





Number of Computing Elements (inner ring) and Batch Systems (outer ring) used in the Grid

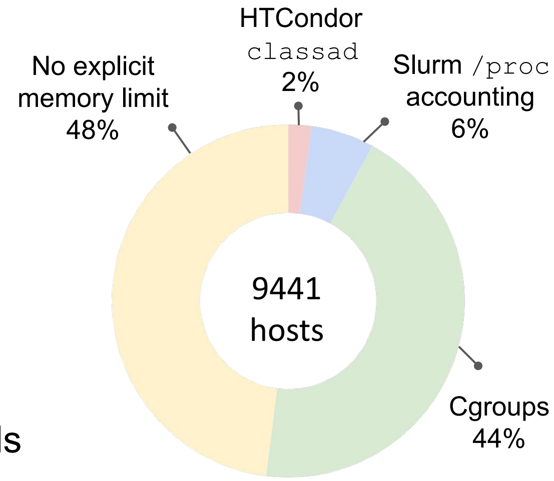
Memory - Site configurations landscape

- A consequence of slot memory limitation is the **killing of the whole slot** when memory consumption is above a threshold
  - Due to JAliEn sub-partitioning, over-consuming payloads might trigger the killing of well-behaved co-executors
  - Becomes a bigger problem in a **whole-node scenario**
- Need a continuous supervision of job memory utilization to **anticipate worker-node payload termination decision**
  - Identifying top-offending payloads and proceed to targeted preemption given job priorities
  - Guaranteeing that slot co-executing workflows can continue

ALICE

Memory - Site configurations landscape

- Sites with sharp kill thresholds - imposing memory **hard limits**
  - Very **predictable** behaviour
  - **Perfect candidates** for controlled job preemption
  - Need to understand and anticipate all ways of defining them (configs on machine or different batch system parameters)
- Hosts with no memory limit or **soft limit**
  - **Fluctuating** allocation, depending on machine utilization levels
  - Machine status needs to be monitored



Memory limiting on Grid hosts

Memory - Configurations to promote job success

ALICE

- **Avoid** setting **too strict** memory thresholds
  - **2GB/core** might be **not enough** to run many of our jobs
  - Apart from the 2GB/core minimal requirement, some extra memory is needed to run the JAliEn agents
- In case physical machines do **not** have **enough memory** resources, advertising less CPU cores is better
  - Increasing the memory/CPU core ratio
  - In whole-node scenarios, JAliEn takes care of advertising less cores in such cases
- Machines with swap memory should **not disable swappiness**
  - Letting jobs use swap to extend their memory usage allowance

Memory - Dealing with jobs with high memory demands

- In **whole-node** scenarios JAliEn manages **all the resources** of the execution machine
  - Allowing custom partitioning and allocation to the running workflows
- Jobs can request a custom amount of memory, to be considered in the matching process
  - Without the caveat of needing to ask for extra CPU cores (allocated memory proportional to the core demand)
- Accurate memory demands will potentially lead to **less job and system crashes** due to OOM
- The other side of this coin is idling CPU cores
  - The process should weigh efficiency vs importance of payload
  - … and should not dispense from code optimization efforts to use less memory

Whole-node scheduling

With whole-node scheduling we have a **better control** of the resources:

- **Custom resource partitioning**, mainly CPU and memory
- **Improved data locality** with a NUMA-aware job allocation
- **CPU oversubscription** to promote efficient resource utilization
    - Running jobs with complementary resource usage patterns in idle CPU cycles
- When run with RHEL 9 OS, **rootless partitioning of *cgroups v2*** slot
    - Apart from NUMA aware scheduling, constraining of CPU bandwidth to prevent job interference with co-executors
    - Custom allocation settings in memory