Google

Static Lock Declaration and Predictive Queue System for Ganeti

morg@google.com

GanetiCon 2016

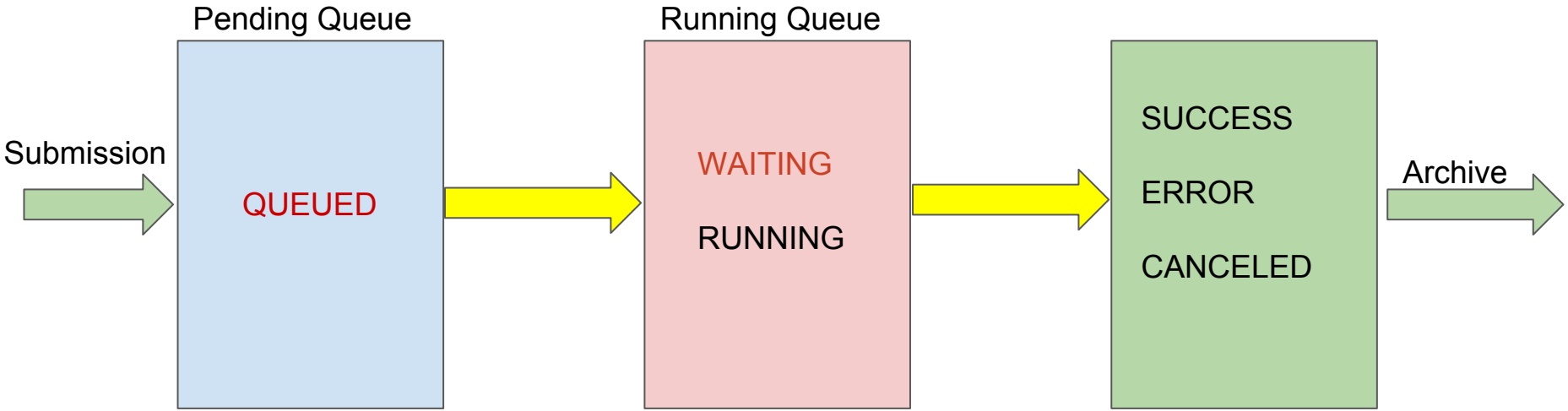
Current state of Ganeti infrastructure (2.16):

- A Ganeti **cluster** is divided in node groups, nodes, and instances.
- A **node group** contains some nodes, a **node** contains some instances.
- **Instances** have a primary node and, usually, at least a secondary node.
- Each cluster has a **master node** and various master candidates.
- Ganeti commands create **jobs** that are scheduled and executed by the master node.

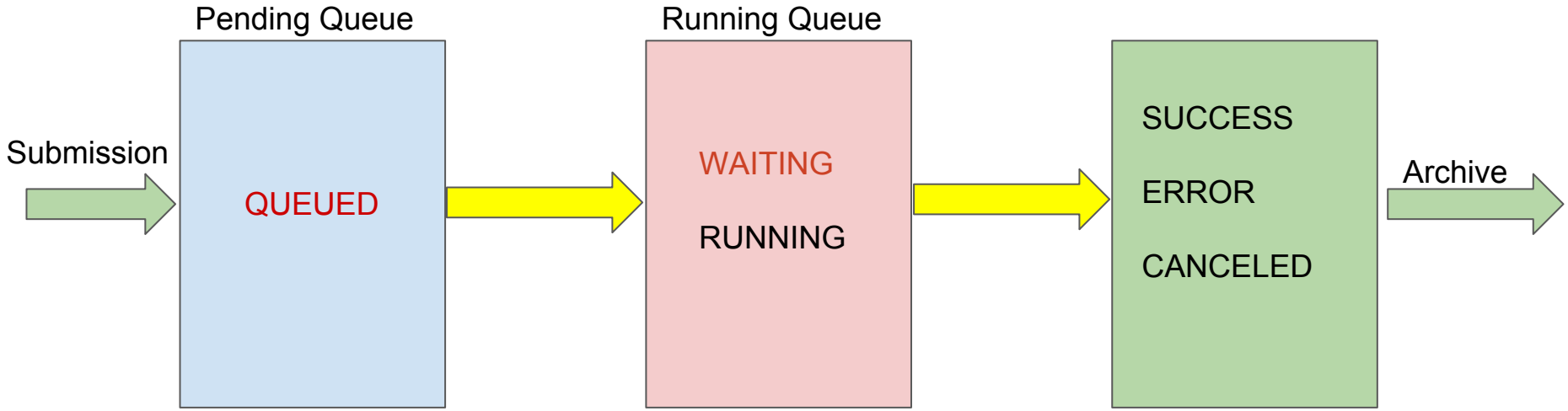
Overview of a Ganeti job:

- A job is a command that is scheduled on a Ganeti cluster to be executed.
- Jobs can be comprised of multiple opcodes (usually only one).
- An **opcode** is the smallest logical unit of operation, it usually maps 1:1 with a task to be executed.
- Jobs keep track of various data like submission time, start time, execution time and various logs for failures or debugging reasons.
- Jobs are put in a **queue** and subsequently scheduled for execution based on various job scheduler policies.

The Job State Machine

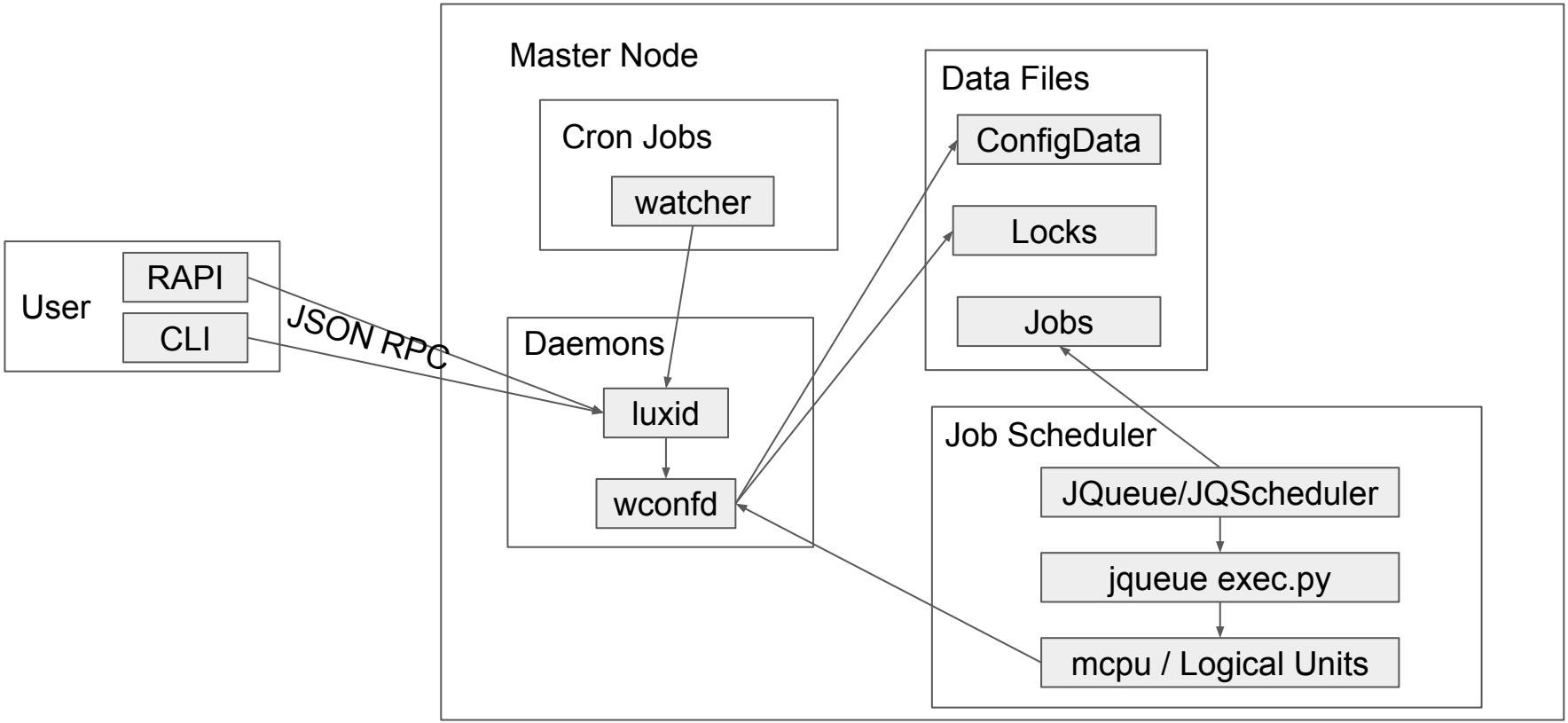


The Job State Machine

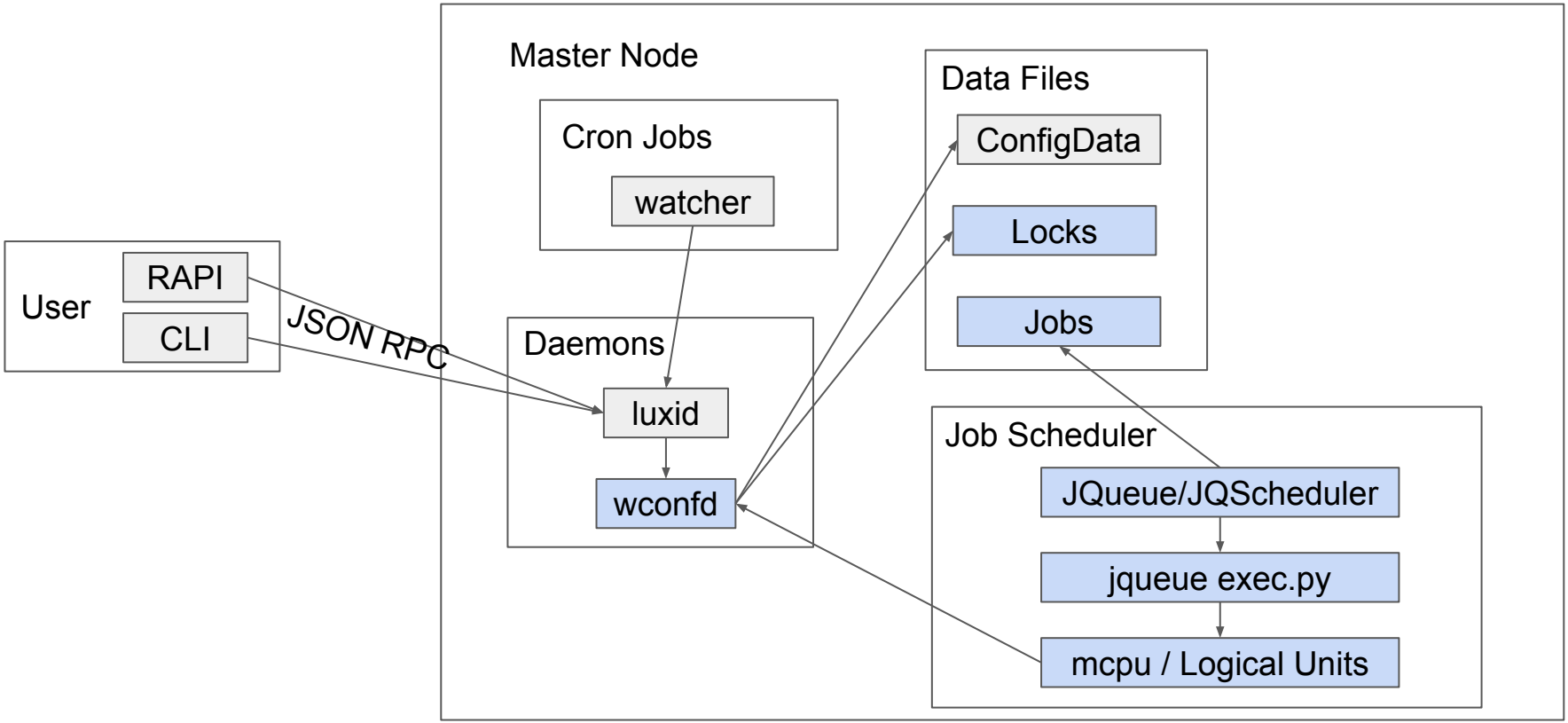


For all intents and purposes, this talk will only care about jobs in the pending and running queues.

Job Architectural Overview



Job Architectural Overview



Locks in Ganeti:

- Ganeti uses a **lock** system around cluster resources to avoid race conditions between jobs.
- Try to automatically acquire all the locks it needs as a job transitions from the pending to the running queue.
- Failure to acquire any lock will block the job and set it to WAITING state.
- It is **not possible** to acquire more locks after a job transitions to RUNNING, however it is possible to release those already acquired.

Locks in Ganeti (2):

- wconfd is the daemon in charge of handing out locks.
- The lock state can be found in /var/lib/ganeti/locks.data
- Each lock is listed under its job ID with its opcodes, type and resource name:

[[[[13,"/var/run/ganeti/livelocks/job_13_1473891518",1455],
[["cluster/BGL","shared"],
["instance/instancename.ganeti.org", "shared"], …
["nodegroup/a4991b92-808f-4184-8534-65039149aa44", "shared"],
["node/032b4a55-c079-44c2-8e3c-62bb65050a53","shared"], ...]]], []]

Ganeti Lock Levels:

Once Upon A Time…

The Big Ganeti Lock (BGL)

Ganeti Lock Levels:

- Few jobs still make use of the BGL.
- Locks are spread among different resource levels if the BGL is not necessary.
- In decreasing order of importance/acquisition:
  - Instance
  - Node Group
  - Node
  - Node Resource
  - Network

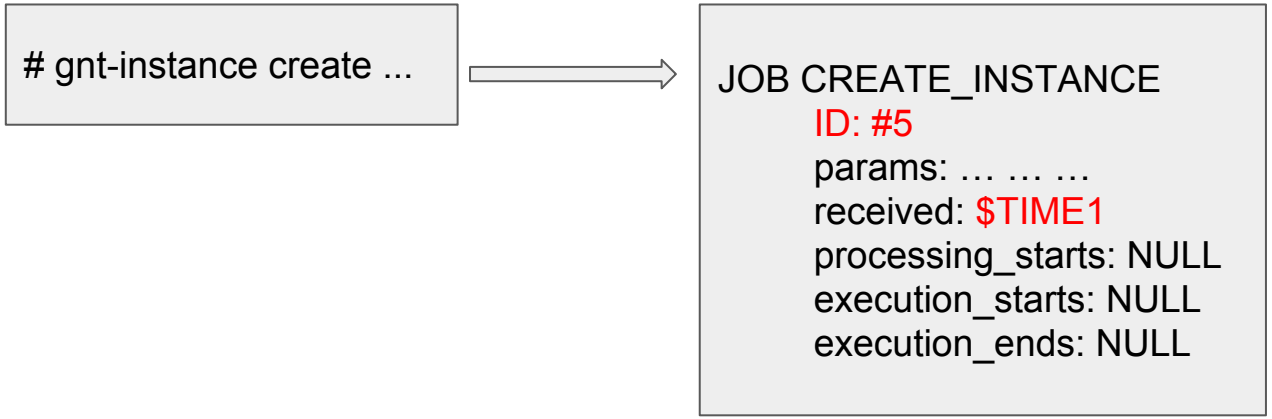
Lock Types:

- There are two major different lock types:
  - **Exclusive lock**, no other lock can be held on the same resource.
  - **Shared lock**, the same resource can be locked by multiple jobs at the same time in shared mode. It will block if an exclusive lock is already taken.
- Some opcodes can specify the "**all set**" of locks for a resource to acquire all locks for that level.
- Some locks can also be **opportunistic**: try to acquire as many free locks as possible for its resource level without blocking.

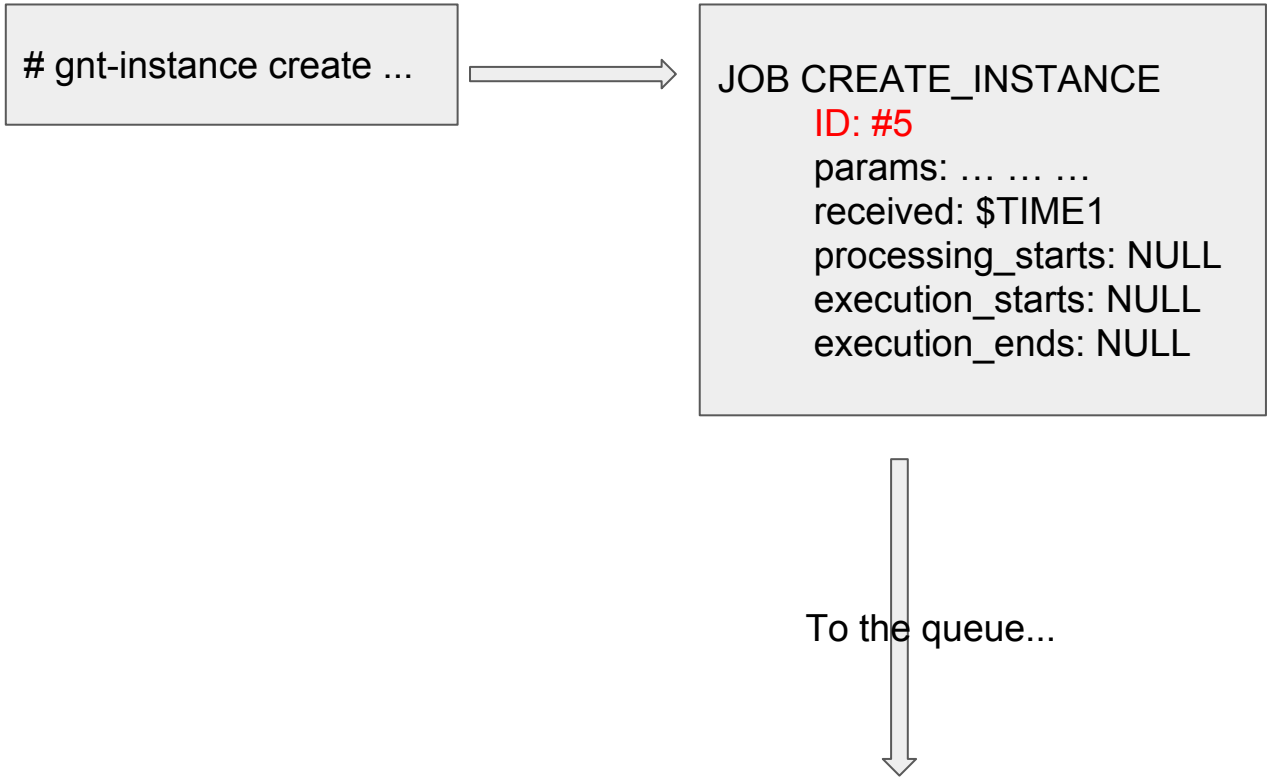
The Life of a Job

```
# gnt-instance create ...
```

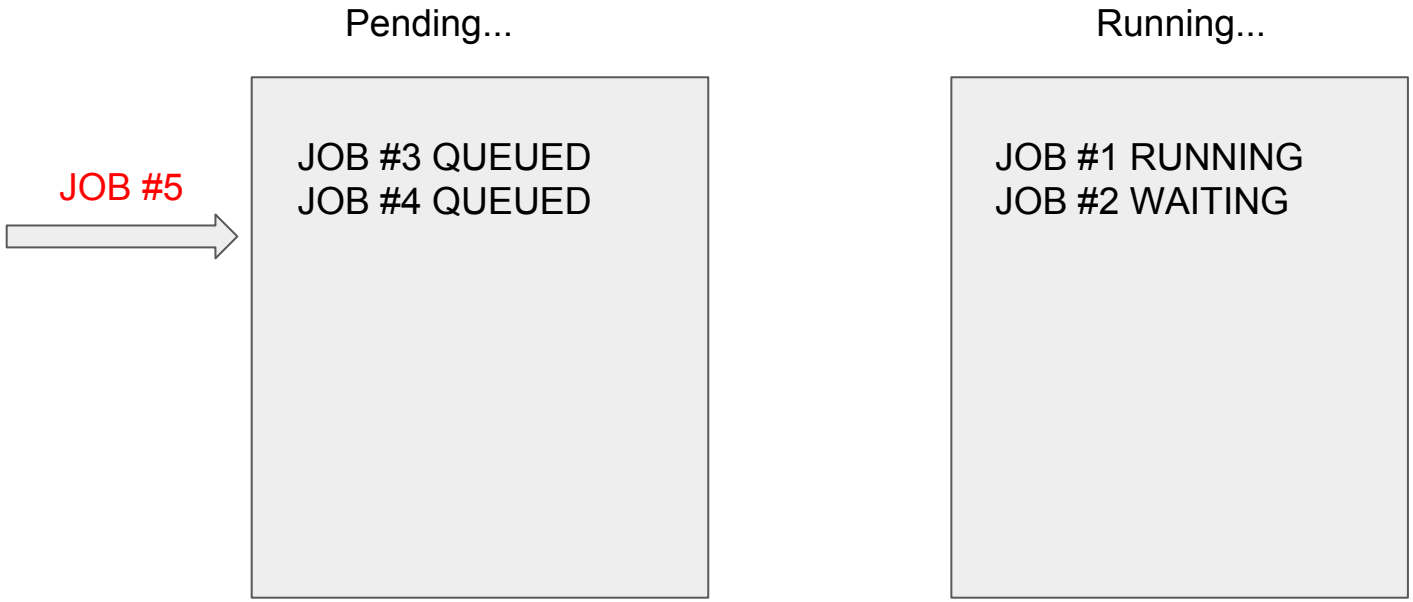
The Life of a Job



The Life of a Job



The Life of a Job



The Life of a Job

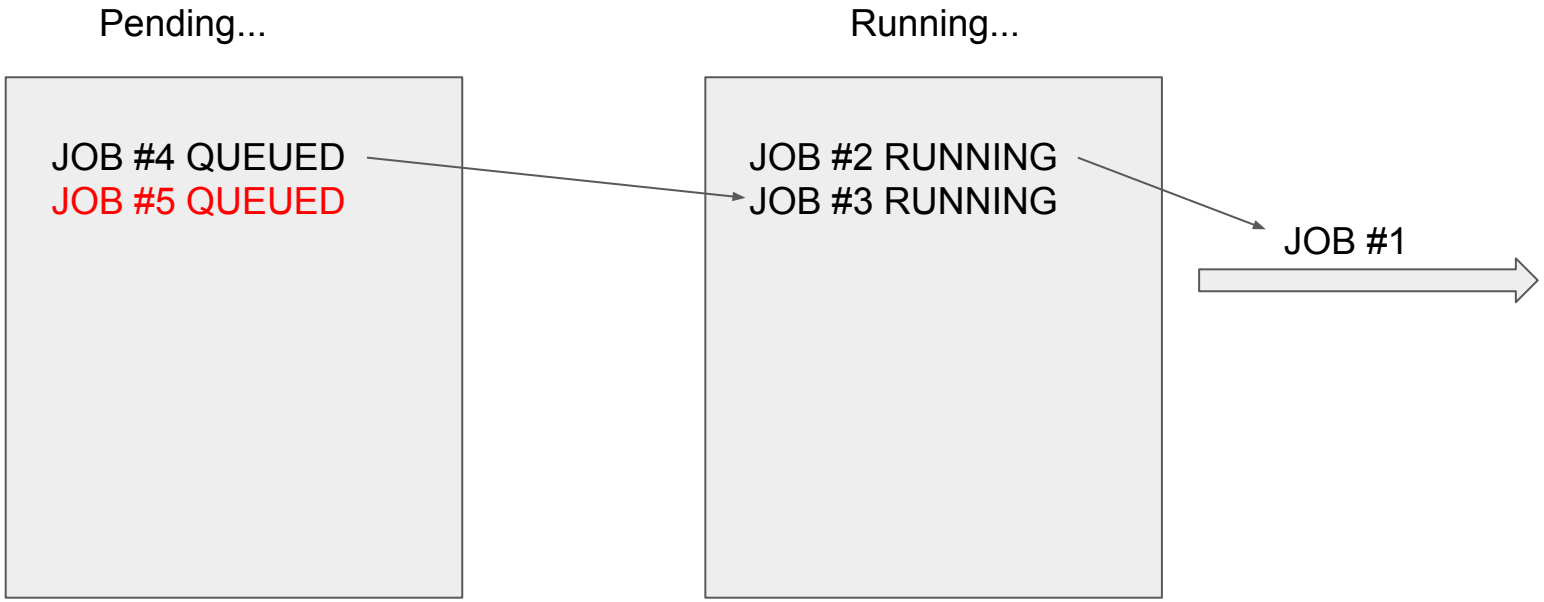
Pending...

JOB #3 QUEUED
JOB #4 QUEUED
JOB #5 QUEUED

Running...

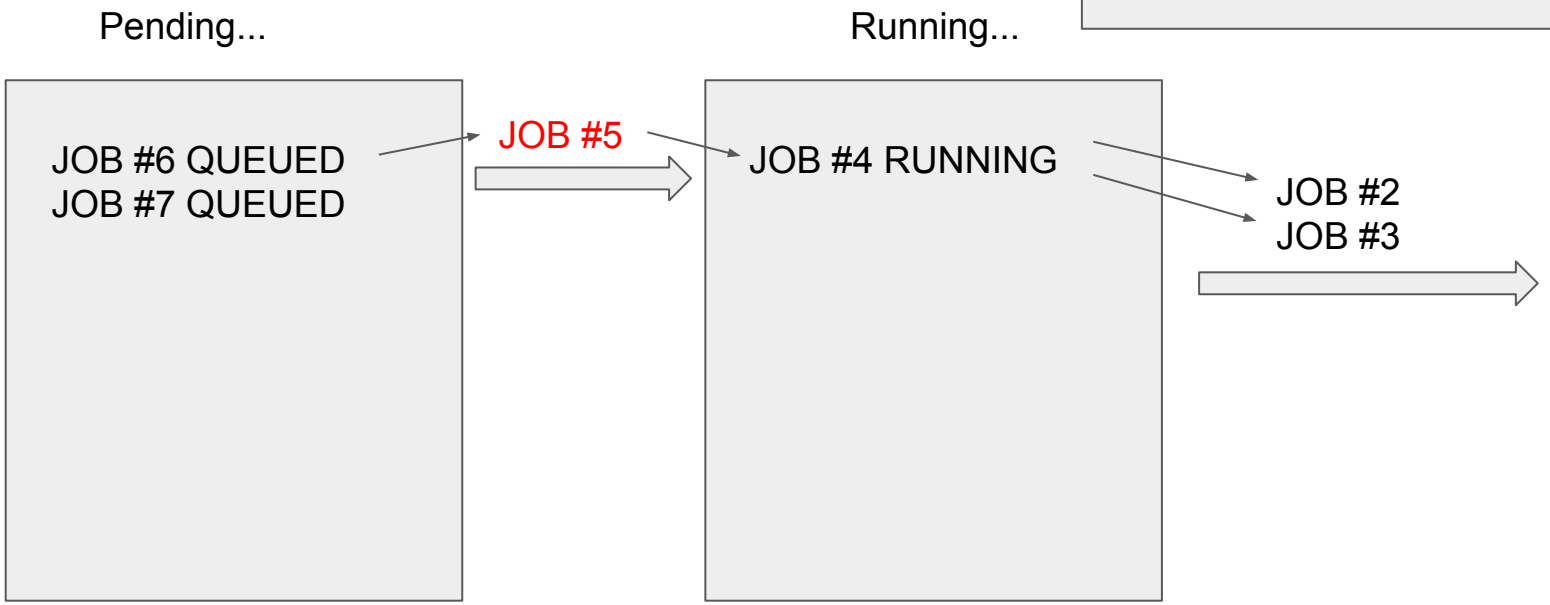
JOB #1 RUNNING
JOB #2 WAITING

The Life of a Job

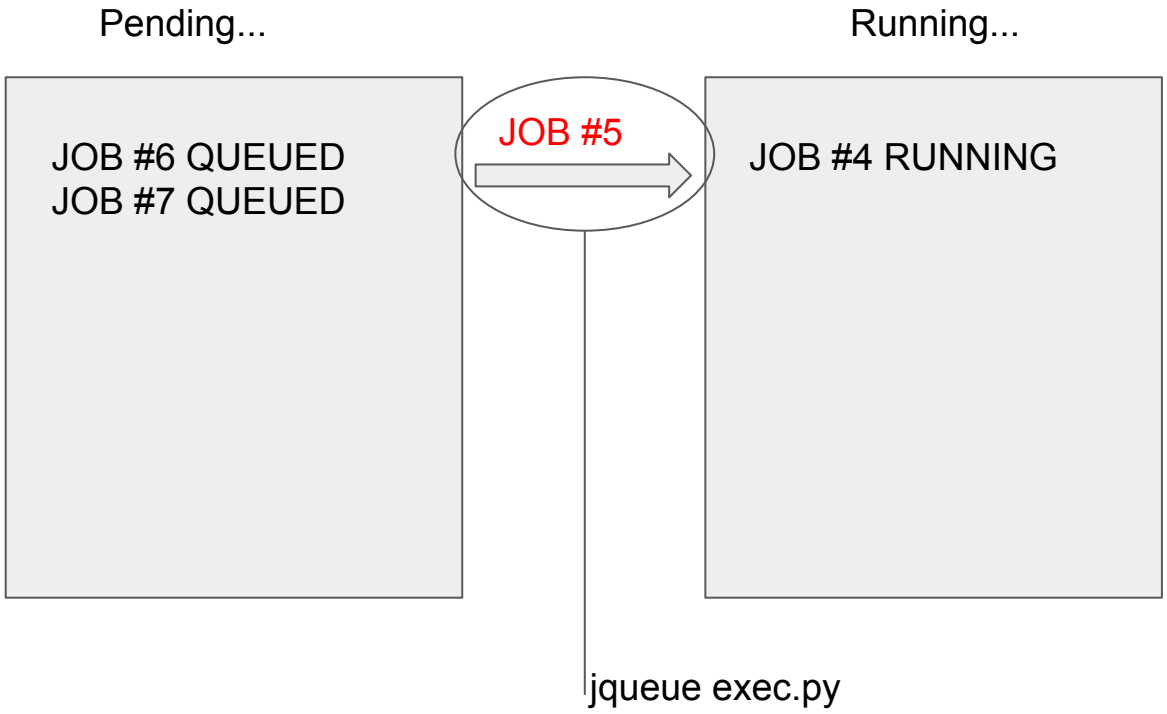


The Life of a Job

JOB CREATE_INSTANCE
ID: #5
params: … … …
received: $TIME1
processing_starts: $TIME2
execution_starts: NULL
execution_ends: NULL

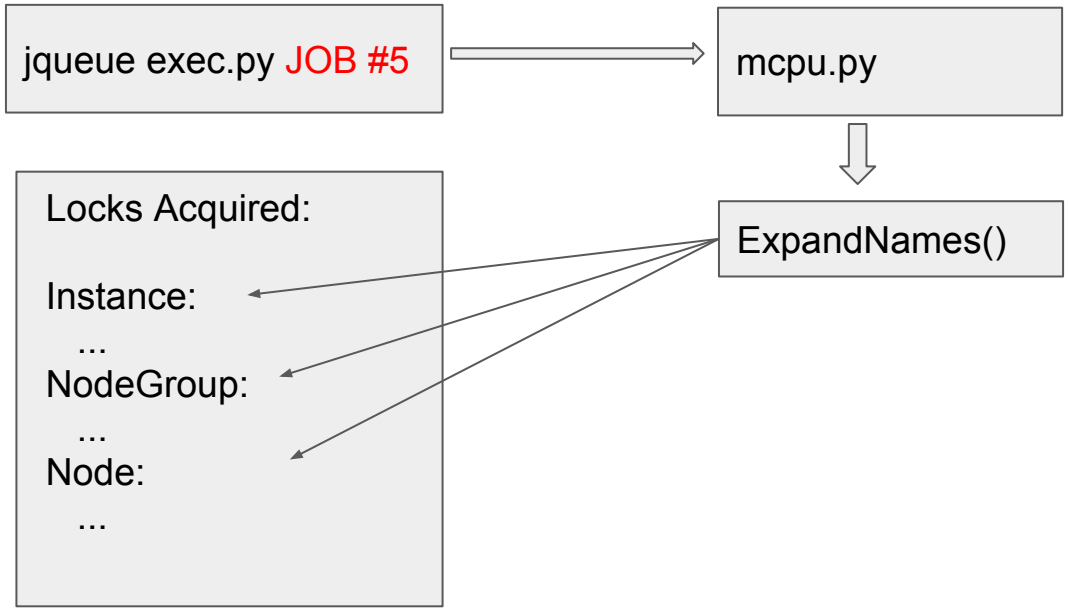


The Life of a Job

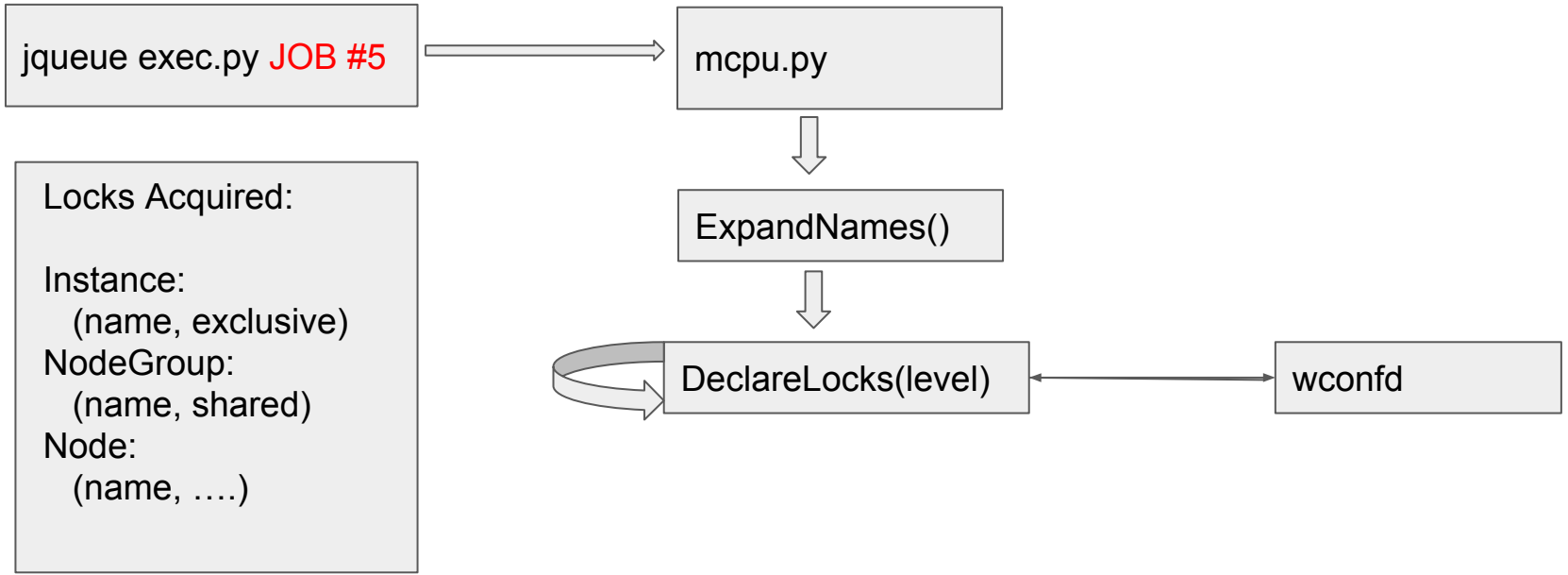


The Life of a Job

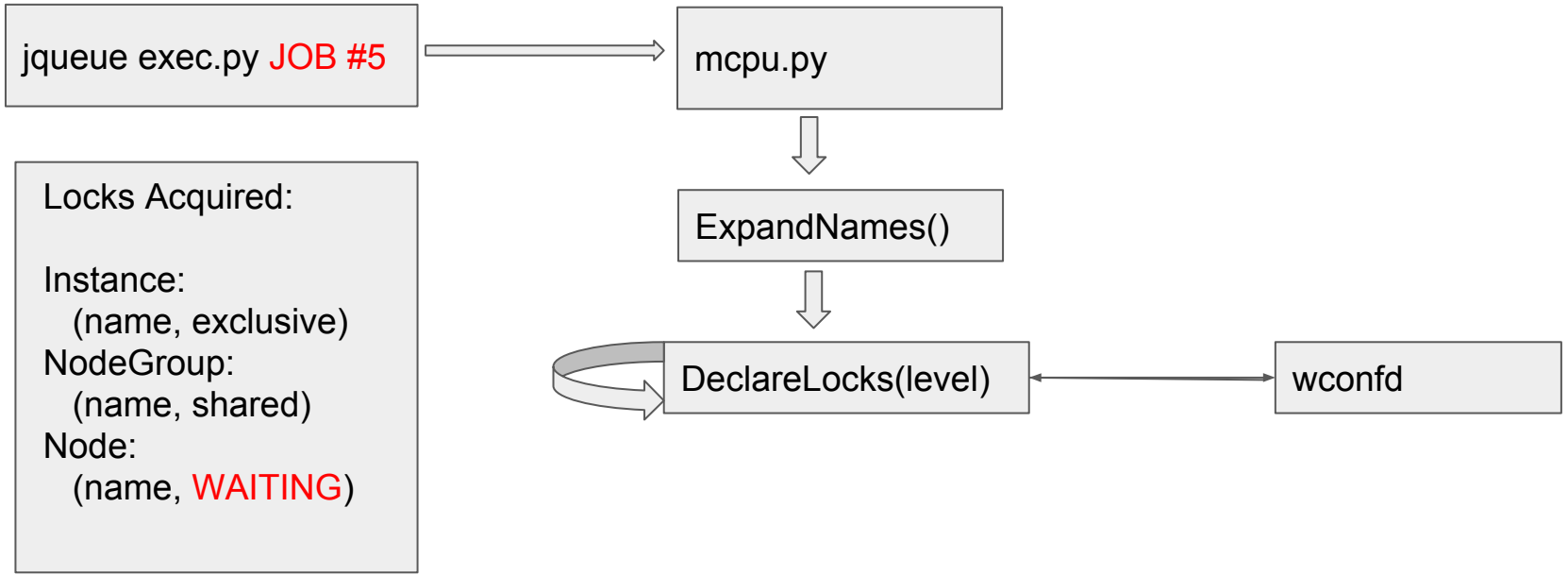
The Life of a Job



The Life of a Job



The Life of a Job



The Life of a Job

Pending...

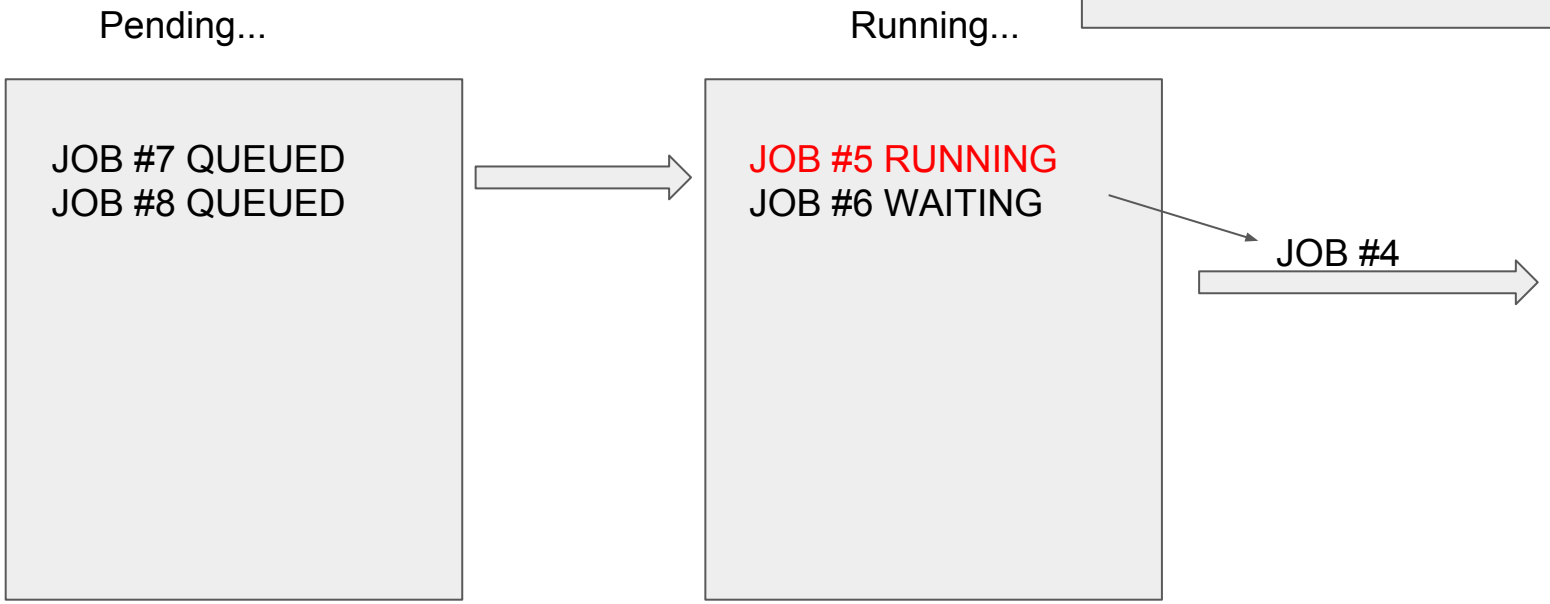
JOB #6 QUEUED
JOB #7 QUEUED
JOB #8 QUEUED

Running...

JOB #4 RUNNING
JOB #5 WAITING

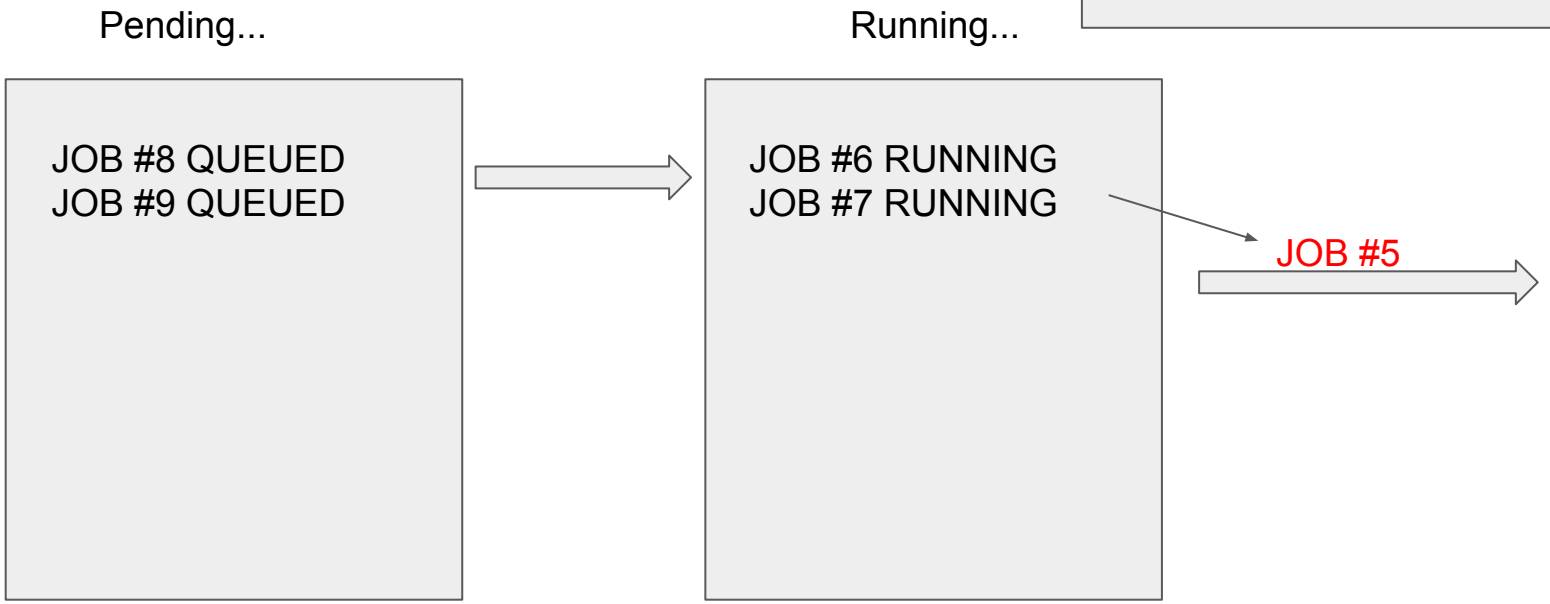
The Life of a Job

JOB CREATE_INSTANCE
ID: #5
params: … … …
received: $TIME1
processing_starts: $TIME2
execution_starts: $TIME3
execution_ends: NULL



The Life of a Job

JOB CREATE_INSTANCE
ID: #5
params: … … …
received: $TIME1
processing_starts: $TIME2
execution_starts: $TIME3
execution_ends: $TIME4



Current Pitfalls of the Queue System

- By design, it is not possible to transition back from RUNNING to WAITING.
- Once a job is RUNNING, it is not possible to cancel it anymore.
- The running queue can **fill up** with a lot of WAITING jobs that are **doing nothing** waiting for their locks and waste precious queue space.
- On average, jobs are scheduled and executed on a **first-come/first-serve** basis.
- Features like job filtering, rate limiting, and priority buckets don't easily automate and possibly **increase operational toil**.
- It is **not possible to know** which **locks** are required by a job **before** it transitions to the **running queue**.

The Predictive Queue Scheduler

The (new) Predictive Queue Scheduler

- Allow the job scheduler to **choose** which **jobs** to move from the pending **to the running queue** to solve the problem of a **first come/first serve** implementation.
- Analyze the jobs/opcodes in the queue to **estimate** their **lock acquisition** and **compare** them with the currently running jobs.
- Prefer scheduling multiple **smaller jobs** instead of a **large job** that is going to be **blocked** in a WAITING state for a long time, to increase the job throughput of a cluster.

Heuristic Algorithm

- **Read the state** of the currently running jobs.
- **Analyze** which locks are going to be acquired and are going to **conflict** with the already running jobs.
- Estimate which jobs are the **least likely to block** in WAITING and schedule those ahead of others.
- **Do not allow starvation** in the occurrence of recurrent smaller jobs being scheduled in front of a single long-lasting job with a worse locking value. There must be an aging factor.

Static Locks - Problem

- It is not possible to know exactly all locks required by all types of opcodes before executing them.
- Locks are **acquired iteratively** by the jqueue executor, **at runtime**.
- They **depend** heavily on the type of opcode and its **parameters**.
- Some locks also **depend** on the current **state of the cluster** and might change while a job is already running (opportunistic locking, iallocator).

Static Locks - Solution

- Analyze the **parameters** of each opcode to attempt to build an accurate **set** of needed locks.
- Fall back to a less accurate and more **generic definition** for those locks that are impossible to know beforehand.
- Always operate on a **worst-case** scenario, it's better to overshoot rather than undershoot the prediction.
- Assign a numerical score called "**Static Predictive Value**" based on the job's estimated locks.

Static Predictive Value

- Each job requires different locks at different resource levels (instance, nodegroup, etc).
- Divide the locks for each resource level in one of the following categories:
  - **None**: No locks are required for this level.
  - **Shared**(***resource***): Acquire a lock for the *resource* in shared mode.
  - **UnknownShared**: Acquire an unknown number of shared locks.
  - **AllShared**: Acquire a shared lock for all resources on that level.
  - **Exclusive**(***resource***): Acquire a lock for the *resource* in exclusive mode.
  - **UnknownExclusive**: Acquire an unknown number of exclusive locks.
  - **AllExclusive**: Acquire an exclusive lock for all resources on that level.

Static Predictive Value (2)

- For each resource level, compare the lock of the current job with those of the already running jobs and assign the following heuristic values to each comparison:
  - **0**: There is no contention, and no contention is added to the state of the cluster.
  - **0.3**: There should be no contention, however the likelihood for future contention of resources in the cluster is slightly increased.
  - **0.5**: There should be no contention, however the likelihood for future contention of resource in the cluster is greatly increased.
  - **1.5**: There is a chance that the job will get stuck but there is no certain way of knowing it.
  - **3**: The job will certainly get stuck in WAITING.

Lock Comparison Operation

	N	S_j	US	AS	E_j	UE	AE
N	0	0	0	0	0	0	0
S_i	0.3	0	0	0	if (i ∩ j) then 3 else 0.3	1.5	3
US	0.3	0.3	0.3	0.3	1.5	1.5	3
AS	0.3	0.3	0.3	0.3	3	3	3
E_i	0.5	if (i ∩ j) then 3 else 0.5	1.5	3	if (i ∩ j) then 3 else 0.5	1.5	3
UE	0.5	1.5	1.5	3	1.5	1.5	3
AE	0.5	3	3	3	3	3	3

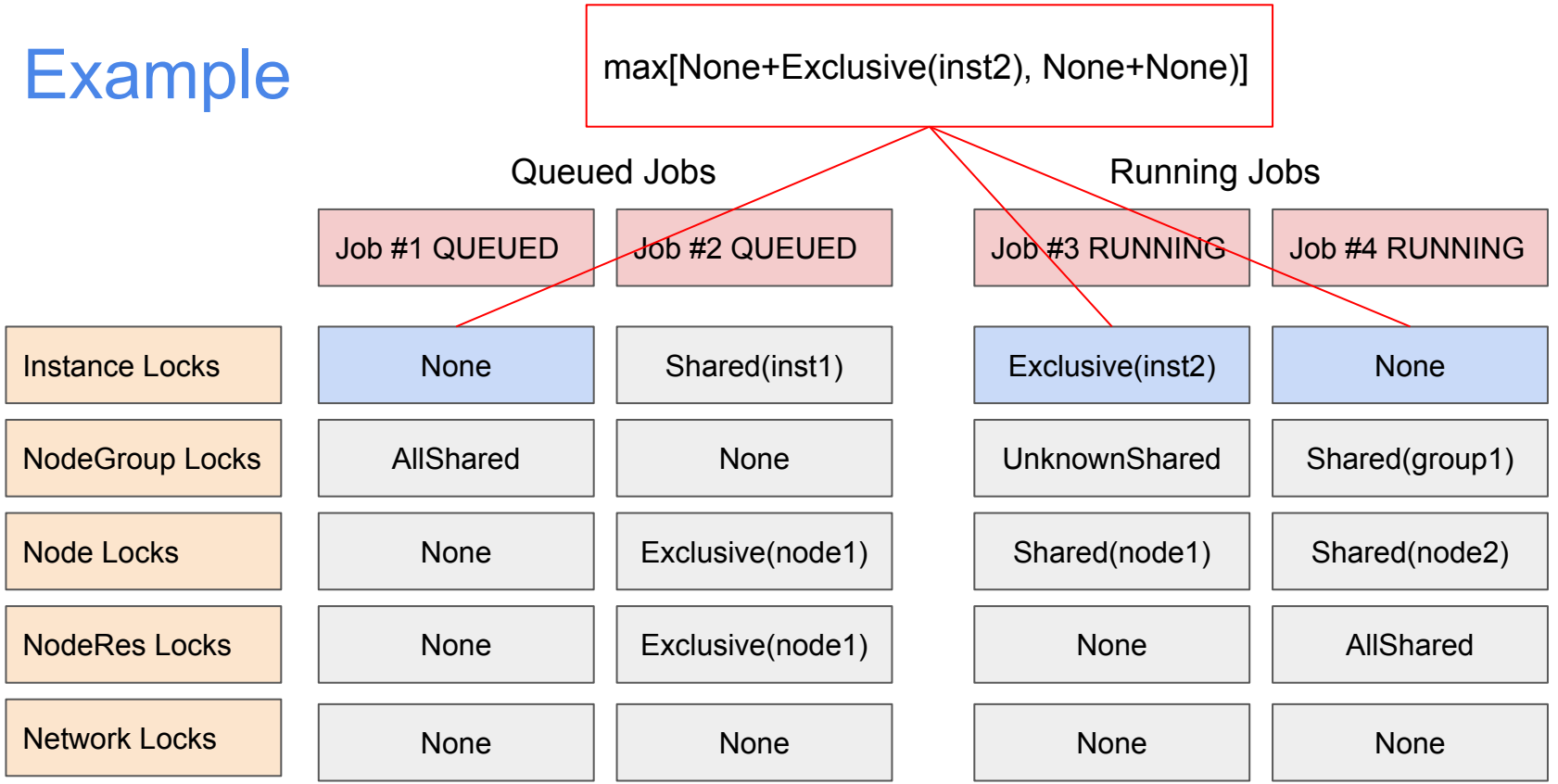
Static Predictive Value (3)

- **Compare** the given job **with each** of the running jobs **for each** of the 5 resource levels (instance, nodegroup, node, noderes, and network).
- Save the **max** calculated **value for each comparison** at each resource level.
- **Sum** all values together.
- The value is between 0 and 15, where 0 means no locks and 15 means definitely blocked on each level.
- If the job requires the BGL, the SPV is automatically set as 15.

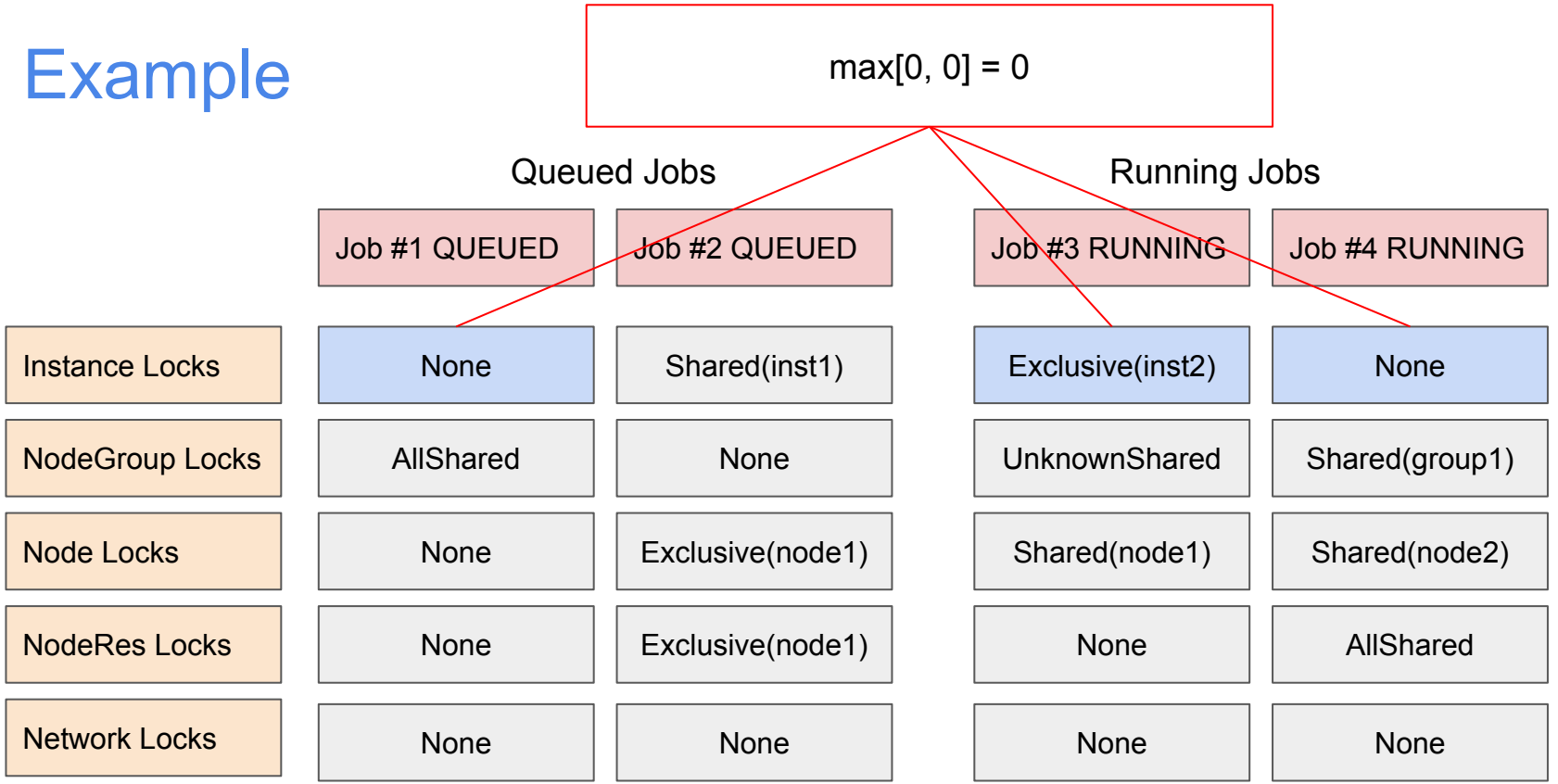
Example

	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	None	Shared(inst1)	Exclusive(inst2)	None
NodeGroup Locks	AllShared	None	UnknownShared	Shared(group1)
Node Locks	None	Exclusive(node1)	Shared(node1)	Shared(node2)
NodeRes Locks	None	Exclusive(node1)	None	AllShared
Network Locks	None	None	None	None

Example


max[None+Exclusive(inst2), None+None)]
Queued Jobs
Running Jobs
Job #1 QUEUED
Job #2 QUEUED
Job #3 RUNNING
Job #4 RUNNING
Instance Locks
None
Shared(inst1)
Exclusive(inst2)
None
NodeGroup Locks
AllShared
None
UnknownShared
Shared(group1)
Node Locks
None
Exclusive(node1)
Shared(node1)
Shared(node2)
NodeRes Locks
None
Exclusive(node1)
None
AllShared
Network Locks
None
None
None
None

Example

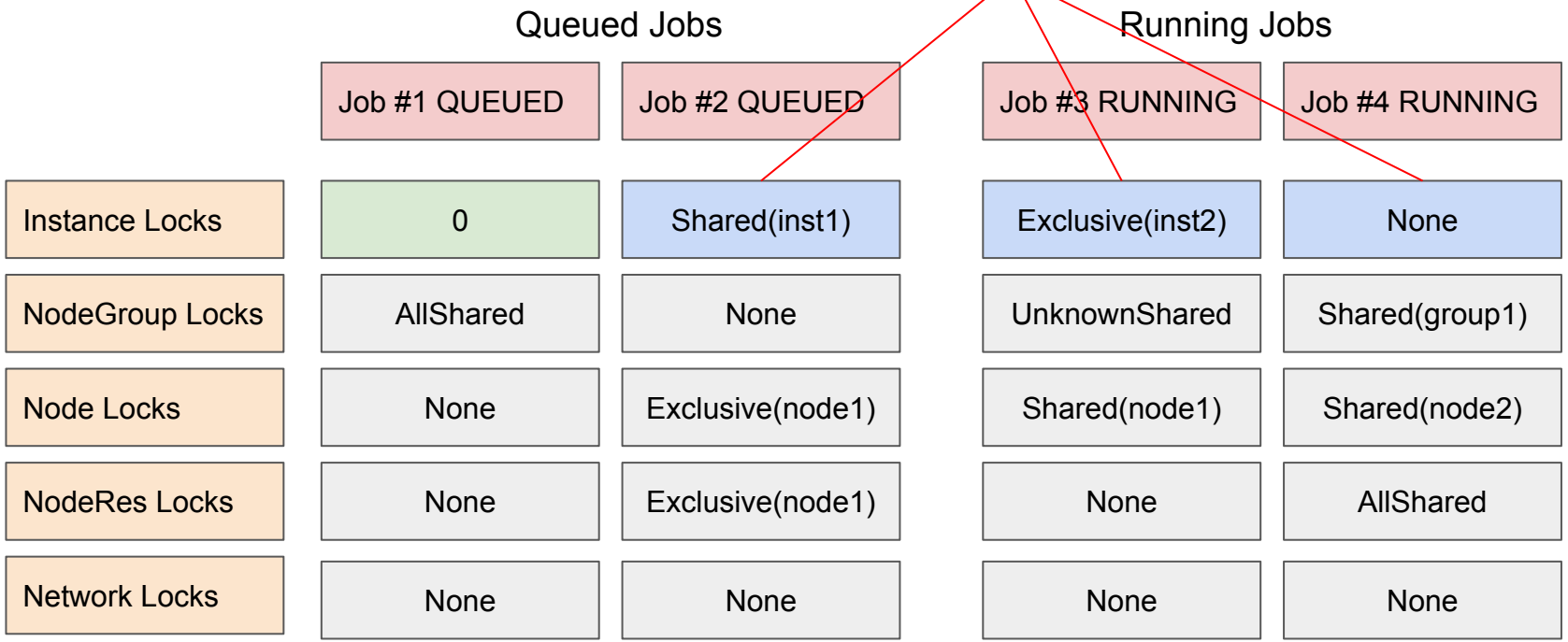

max[0, 0] = 0
Queued Jobs
Running Jobs
Job #1 QUEUED
Job #2 QUEUED
Job #3 RUNNING
Job #4 RUNNING
Instance Locks
None
Shared(inst1)
Exclusive(inst2)
None
NodeGroup Locks
AllShared
None
UnknownShared
Shared(group1)
Node Locks
None
Exclusive(node1)
Shared(node1)
Shared(node2)
NodeRes Locks
None
Exclusive(node1)
None
AllShared
Network Locks
None
None
None
None

Example

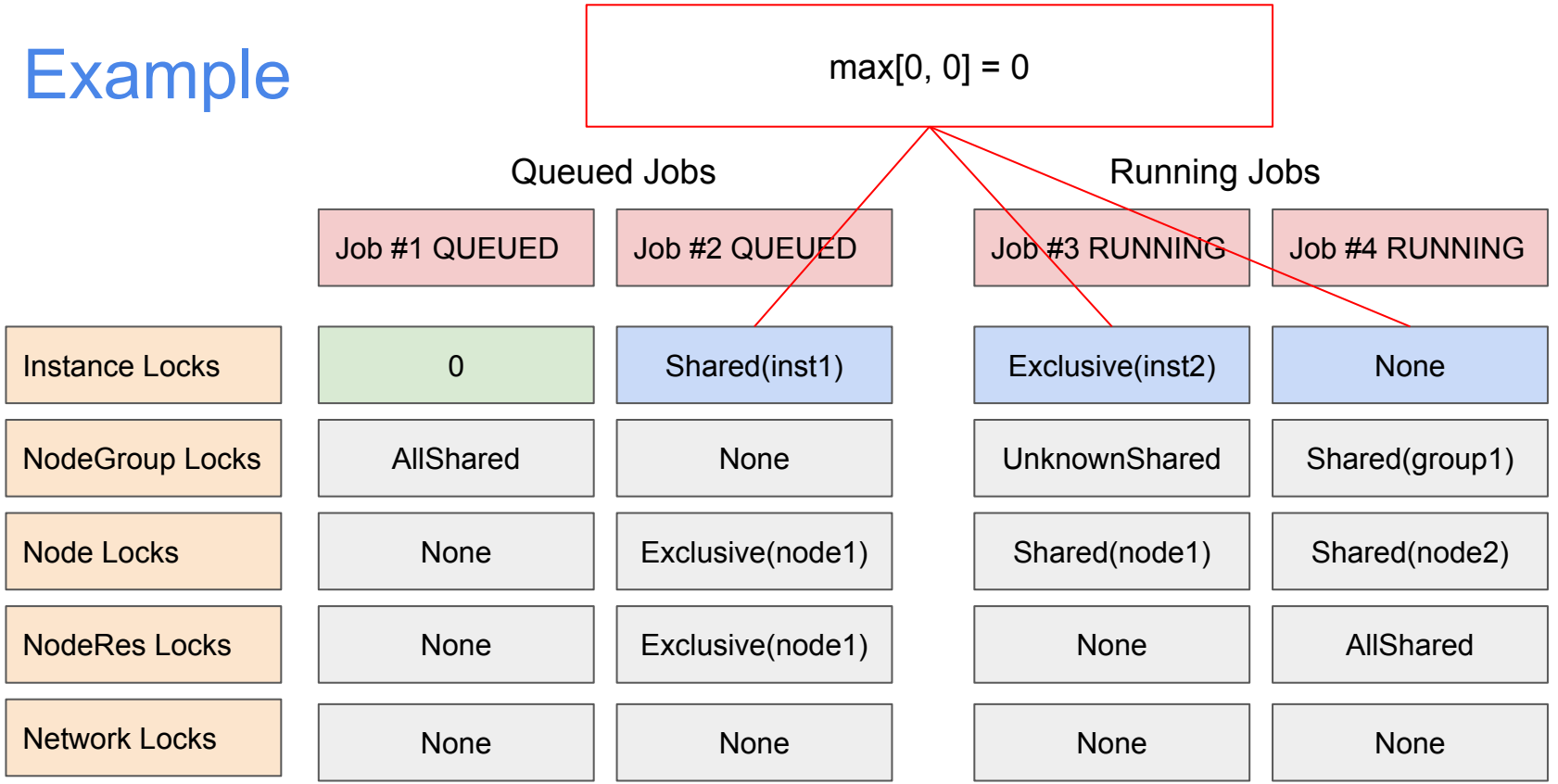
	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	0	Shared(inst1)	Exclusive(inst2)	None
NodeGroup Locks	AllShared	None	UnknownShared	Shared(group1)
Node Locks	None	Exclusive(node1)	Shared(node1)	Shared(node2)
NodeRes Locks	None	Exclusive(node1)	None	AllShared
Network Locks	None	None	None	None

Example

max[Shared(inst1)+Exclusive(inst2), Shared(inst1)+None)]



Example

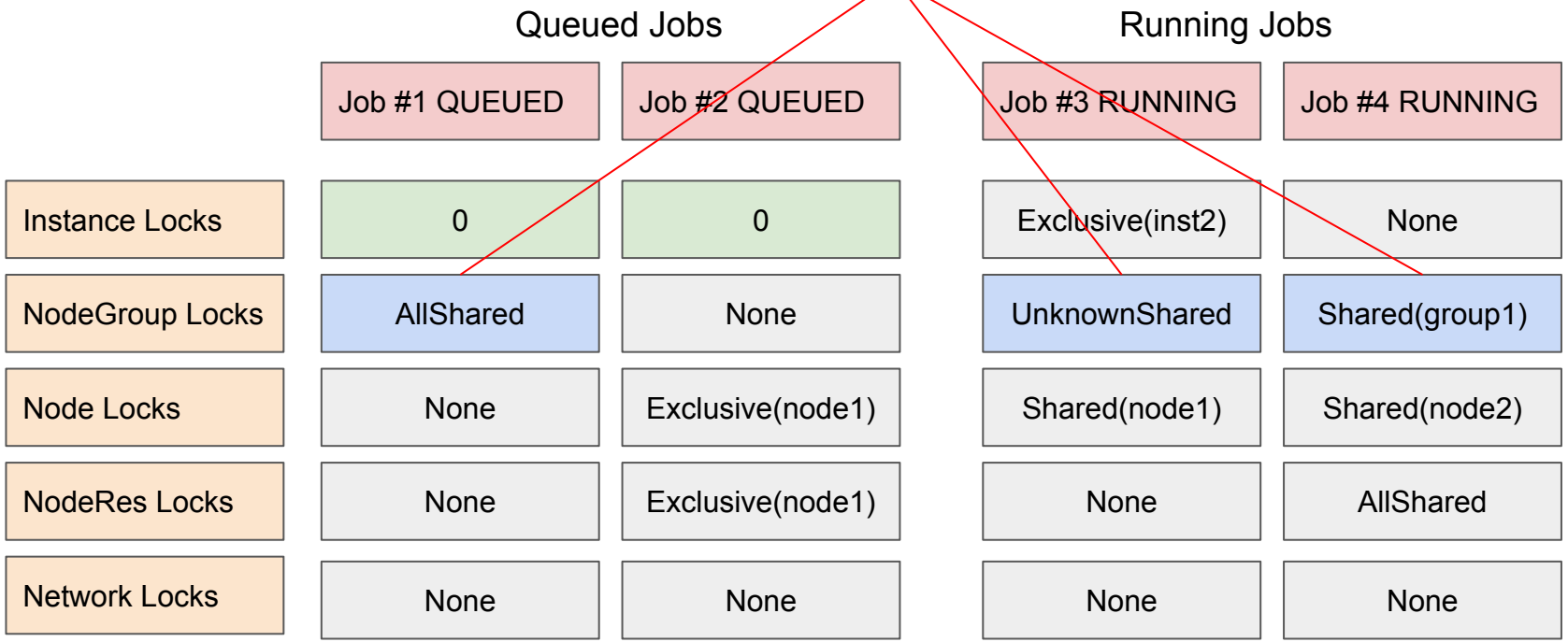

max[0, 0] = 0
Queued Jobs
Running Jobs
Job #1 QUEUED
Job #2 QUEUED
Job #3 RUNNING
Job #4 RUNNING
Instance Locks
0
Shared(inst1)
Exclusive(inst2)
None
NodeGroup Locks
AllShared
None
UnknownShared
Shared(group1)
Node Locks
None
Exclusive(node1)
Shared(node1)
Shared(node2)
NodeRes Locks
None
Exclusive(node1)
None
AllShared
Network Locks
None
None
None
None

Example

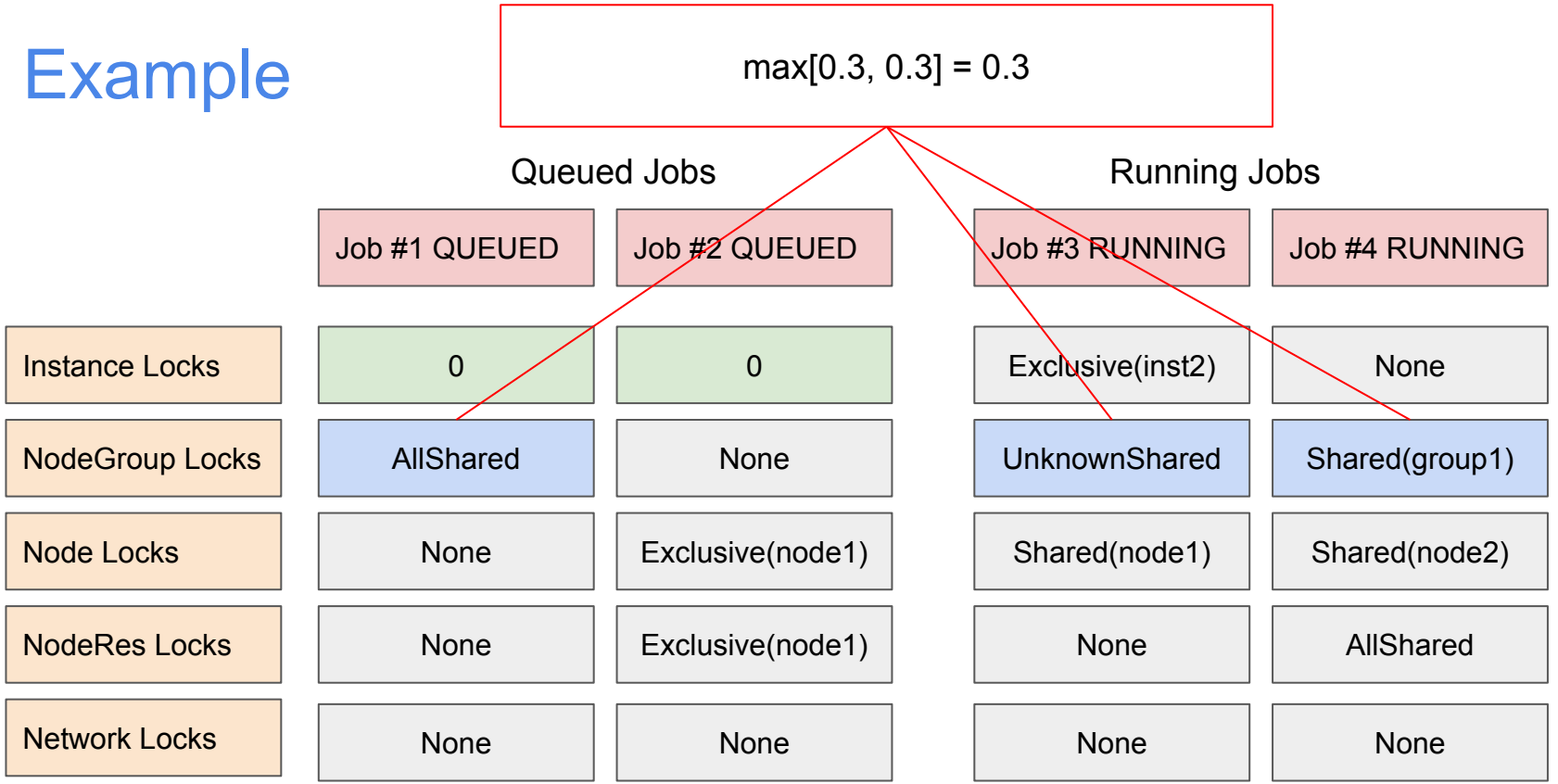
	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	0	0	Exclusive(inst2)	None
NodeGroup Locks	AllShared	None	UnknownShared	Shared(group1)
Node Locks	None	Exclusive(node1)	Shared(node1)	Shared(node2)
NodeRes Locks	None	Exclusive(node1)	None	AllShared
Network Locks	None	None	None	None

Example

max[AllShared + UnknownShared,
AllShared+Shared(group1)]



Example


max[0.3, 0.3] = 0.3
Queued Jobs
Running Jobs
Job #1 QUEUED
Job #2 QUEUED
Job #3 RUNNING
Job #4 RUNNING
Instance Locks
0
0
Exclusive(inst2)
None
NodeGroup Locks
AllShared
None
UnknownShared
Shared(group1)
Node Locks
None
Exclusive(node1)
Shared(node1)
Shared(node2)
NodeRes Locks
None
Exclusive(node1)
None
AllShared
Network Locks
None
None
None
None

Example

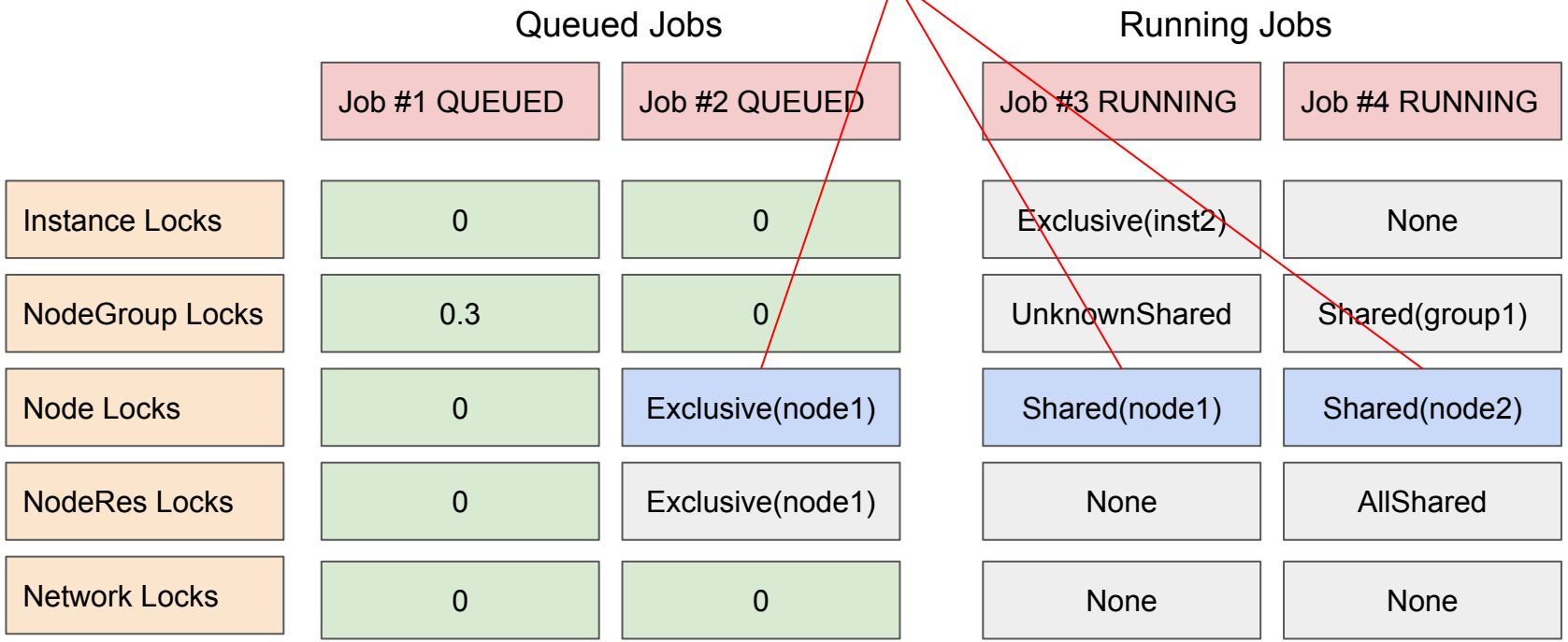
	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	0	0	Exclusive(inst2)	None
NodeGroup Locks	0.3	None	UnknownShared	Shared(group1)
Node Locks	None	Exclusive(node1)	Shared(node1)	Shared(node2)
NodeRes Locks	None	Exclusive(node1)	None	AllShared
Network Locks	None	None	None	None

Example

	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	0	0	Exclusive(inst2)	None
NodeGroup Locks	0.3	0	UnknownShared	Shared(group1)
Node Locks	0	Exclusive(node1)	Shared(node1)	Shared(node2)
NodeRes Locks	0	Exclusive(node1)	None	AllShared
Network Locks	0	0	None	None

Example

max[Exclusive(node1) + Shared(node1),
Exclusive(node1) + Shared(node2)]



Example

max[3, 0.5] = 3

	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	0	0	Exclusive(inst2)	None
NodeGroup Locks	0.3	0	UnknownShared	Shared(group1)
Node Locks	0	Exclusive(node1)	Shared(node1)	Shared(node2)
NodeRes Locks	0	Exclusive(node1)	None	AllShared
Network Locks	0	0	None	None

Example

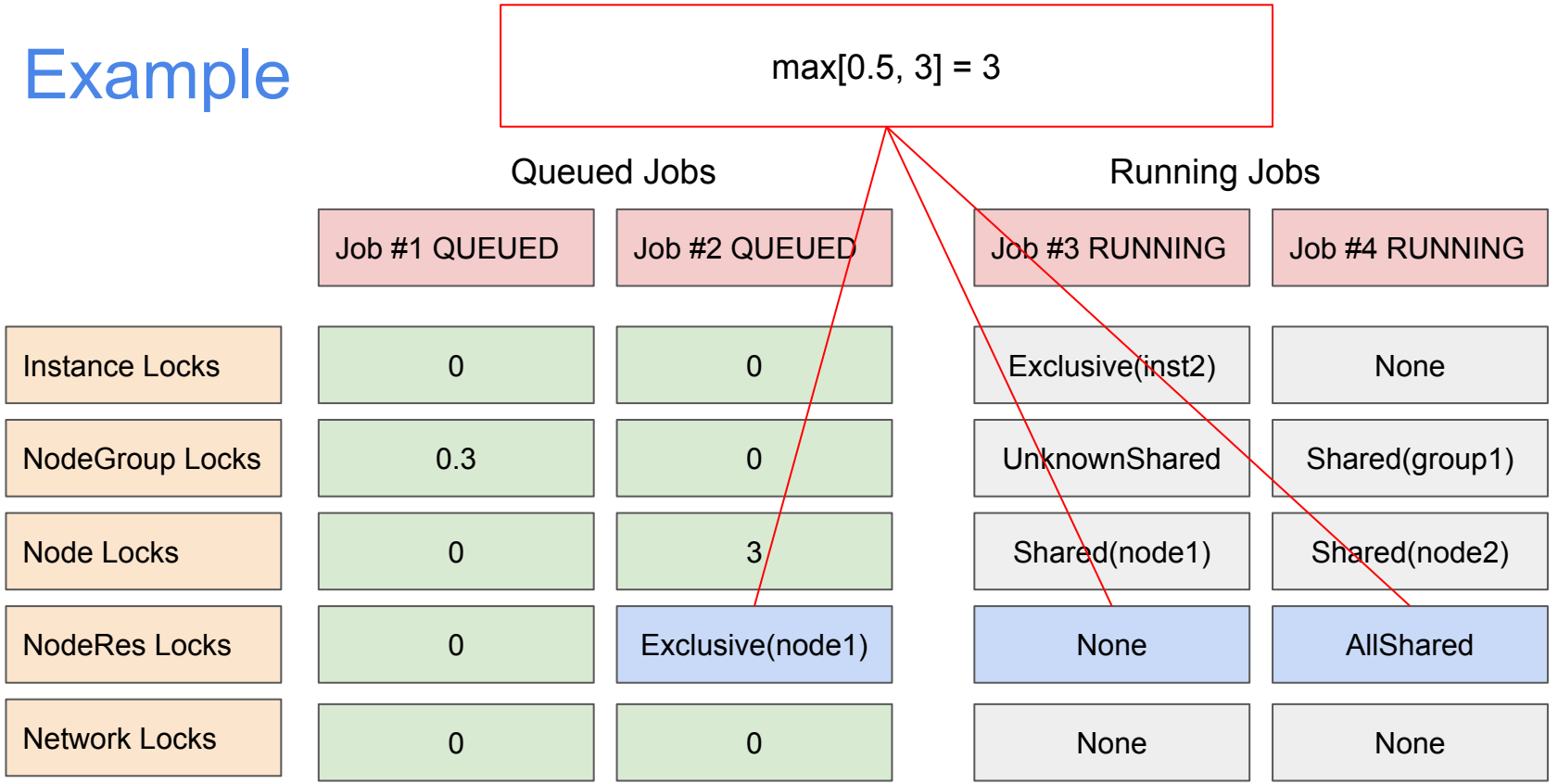
	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	0	0	Exclusive(inst2)	None
NodeGroup Locks	0.3	0	UnknownShared	Shared(group1)
Node Locks	0	3	Shared(node1)	Shared(node2)
NodeRes Locks	0	Exclusive(node1)	None	AllShared
Network Locks	0	0	None	None

Example

max[Exclusive(node1) + None,
Exclusive(node1) + AllShared]

	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	0	0	Exclusive(inst2)	None
NodeGroup Locks	0.3	0	UnknownShared	Shared(group1)
Node Locks	0	3	Shared(node1)	Shared(node2)
NodeRes Locks	0	Exclusive(node1)	None	AllShared
Network Locks	0	0	None	None

Example


max[0.5, 3] = 3
Queued Jobs
Running Jobs
Job #1 QUEUED
Job #2 QUEUED
Job #3 RUNNING
Job #4 RUNNING
Instance Locks
0
0
Exclusive(inst2)
None
NodeGroup Locks
0.3
0
UnknownShared
Shared(group1)
Node Locks
0
3
Shared(node1)
Shared(node2)
NodeRes Locks
0
Exclusive(node1)
None
AllShared
Network Locks
0
0
None
None

Example

	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	0	0	Exclusive(inst2)	None
NodeGroup Locks	0.3	0	UnknownShared	Shared(group1)
Node Locks	0	3	Shared(node1)	Shared(node2)
NodeRes Locks	0	3	None	AllShared
Network Locks	0	0	None	None

Example

	Queued Jobs		Running Jobs	
	Job #1 QUEUED	Job #2 QUEUED	Job #3 RUNNING	Job #4 RUNNING
Instance Locks	0	0	Exclusive(inst2)	None
NodeGroup Locks	0.3	0	UnknownShared	Shared(group1)
Node Locks	0	3	Shared(node1)	Shared(node2)
NodeRes Locks	0	3	None	AllShared
Network Locks	0	0	None	None
SPV:	0.3	6		

Starvation Prevention

- The age of a job is used as an **anti-starvation** coefficient to adjust the predictive score and ensure all jobs get a chance to run.
- The age of a job (seconds since "received" in the queue) is quantized in ***ticks*** of 30 seconds each.
- The more ticks a job has, the more likely it will be scheduled over other jobs with less time spent in the queue.
- With the introduction of an **aging factor** K we make sure that after K ticks a job will be at the top of the queue. It defaults to 30 in the current implementation.

Heuristic Formula

Static Predictive Value: Heuristic value assigned to the job **j** based on its estimated lock and the runtime locks on the cluster.

$$APV(j) = max[0, SPV(j) \cdot (1 - \frac{Age_j}{K})]$$

Actual Predictive Value: Final weight assigned to the job **j**. The lower the value, the more likely it is to be scheduled next.

Aging Coefficient: Dynamic value calculated on the age of the job **j** to avoid starvation in the queue.

Heuristic Formula

1.0 ≤ SPV(j) ≤ 16.0

$$APV(j) = max[0,\ SPV(j) \cdot (1 - \frac{Age_j}{K})]$$

0.0 ≤ APV(j) ≤ 16.0

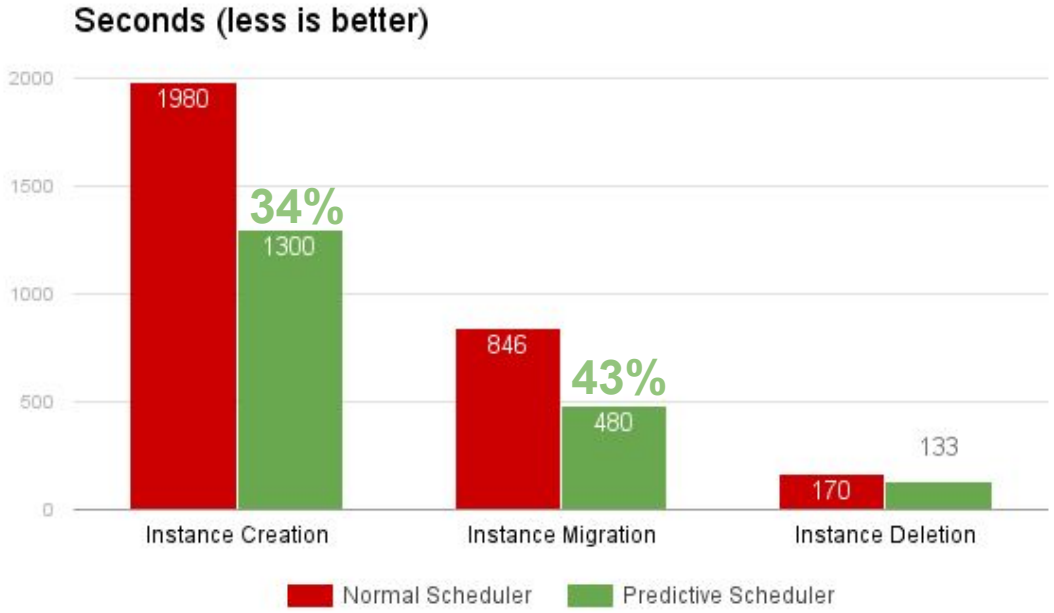
less than or equal to 1.0
(can go in the negatives)

Data Results

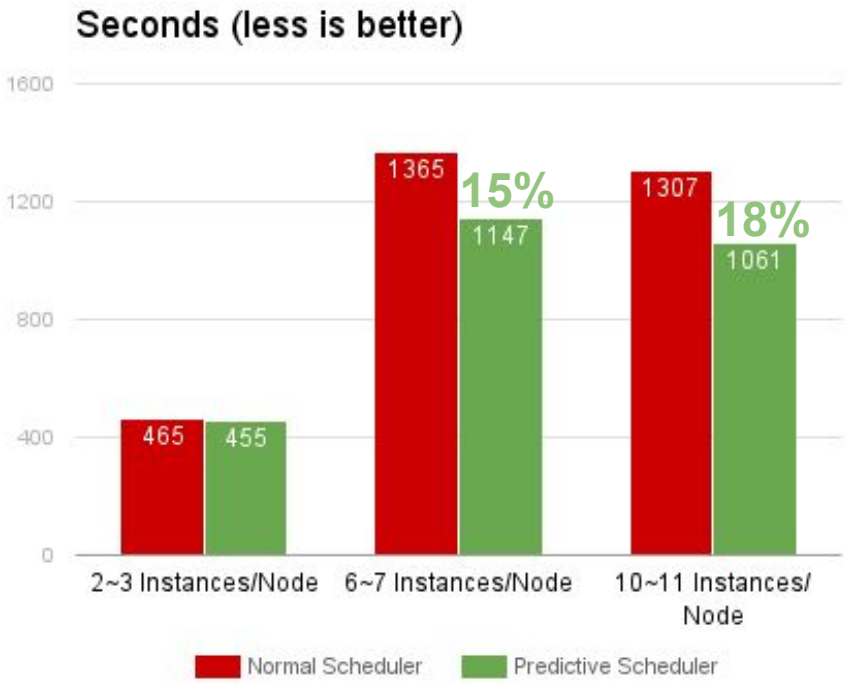
Test Cluster Environment

- 1 Cluster, 1 NodeGroup, 5 Nodes
- Between 2 and 12 instances per node on average.
- Instances have 2~4GB of RAM, 10~50GB of DRBD disks.
- Between 2 and 4 max parallel running jobs.

No Contention Jobs



High Contention Jobs



Old Scheduler

209 **running** INSTANCE_MIGRATE(instance1.ganeti.org)
210 waiting INSTANCE_MIGRATE(instance2.ganeti.org)
211 waiting INSTANCE_MIGRATE(instance3.ganeti.org)
212 waiting INSTANCE_MIGRATE(instance4.ganeti.org)
213 queued INSTANCE_MIGRATE(instance5.ganeti.org)
214 queued INSTANCE_MIGRATE(instance6.ganeti.org)
215 queued INSTANCE_MIGRATE(instance7.ganeti.org)
216 queued INSTANCE_MIGRATE(instance8.ganeti.org)
217 queued INSTANCE_MIGRATE(instance9.ganeti.org)
218 queued INSTANCE_MIGRATE(instance10.ganeti.org)
219 queued INSTANCE_MIGRATE(instance11.ganeti.org)
220 queued INSTANCE_MIGRATE(instance12.ganeti.org)
221 queued INSTANCE_MIGRATE(instance13.ganeti.org)
222 queued INSTANCE_MIGRATE(instance14.ganeti.org)
223 queued INSTANCE_MIGRATE(instance15.ganeti.org)
224 queued INSTANCE_MIGRATE(instance16.ganeti.org)
225 queued INSTANCE_MIGRATE(instance17.ganeti.org)
226 queued INSTANCE_MIGRATE(instance18.ganeti.org)
227 queued INSTANCE_MIGRATE(instance19.ganeti.org)
228 queued INSTANCE_MIGRATE(instance20.ganeti.org)
229 queued INSTANCE_MIGRATE(instance21.ganeti.org)

New Scheduler

209 **running** INSTANCE_MIGRATE(instance1.ganeti.org)
210 waiting INSTANCE_MIGRATE(instance2.ganeti.org)
211 queued INSTANCE_MIGRATE(instance3.ganeti.org)
212 queued INSTANCE_MIGRATE(instance4.ganeti.org)
213 queued INSTANCE_MIGRATE(instance5.ganeti.org)
214 queued INSTANCE_MIGRATE(instance6.ganeti.org)
215 queued INSTANCE_MIGRATE(instance7.ganeti.org)
216 **running** INSTANCE_MIGRATE(instance8.ganeti.org)
217 queued INSTANCE_MIGRATE(instance9.ganeti.org)
218 queued INSTANCE_MIGRATE(instance10.ganeti.org)
219 queued INSTANCE_MIGRATE(instance11.ganeti.org)
220 **running** INSTANCE_MIGRATE(instance12.ganeti.org)
221 queued INSTANCE_MIGRATE(instance13.ganeti.org)
222 queued INSTANCE_MIGRATE(instance14.ganeti.org)
223 queued INSTANCE_MIGRATE(instance15.ganeti.org)
224 queued INSTANCE_MIGRATE(instance16.ganeti.org)
225 queued INSTANCE_MIGRATE(instance17.ganeti.org)
226 queued INSTANCE_MIGRATE(instance18.ganeti.org)
227 queued INSTANCE_MIGRATE(instance19.ganeti.org)
228 queued INSTANCE_MIGRATE(instance20.ganeti.org)
229 queued INSTANCE_MIGRATE(instance21.ganeti.org)

Questions?