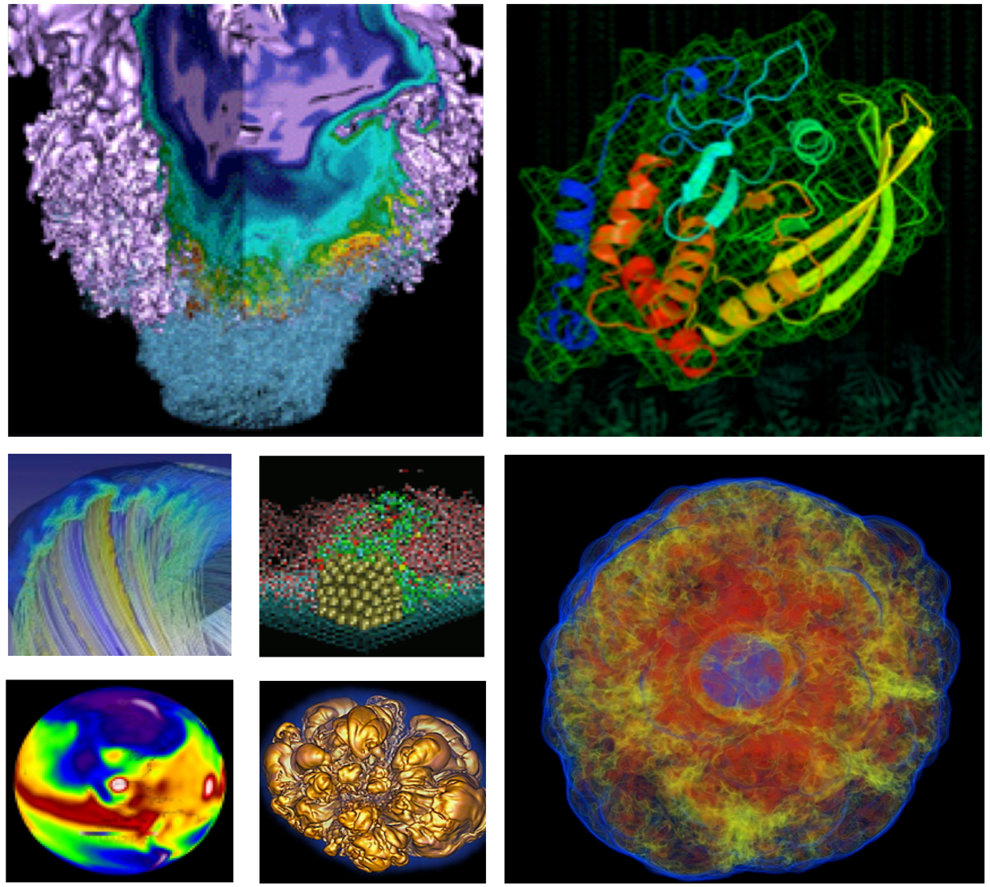
A declarative programming style job submission filter.



Douglas Jacobsen
Computational Systems Group Lead
NERSC

Slurm User Group 2018

NERSC Vital Statistics

860 projects   7750 users   700+ applications

Edison

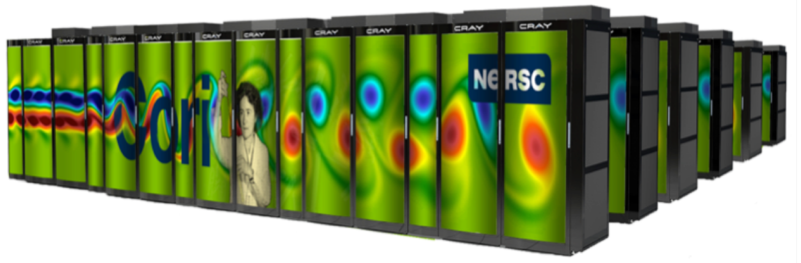
NERSC-7

Cray XC30
5,603 ivybridge nodes   134,472 cores

- 24 cores per node, 134,472 cores total
- 64 GB per node, 2.6 GB/core, 350 TB total
- Primarily used for large capability jobs
- Small – midrange as well
- ~ 7PB of local Lustre scratch

Cori

NERSC-8



Cray XC40
12,070 compute nodes
9,685 KNL   658,580 cores
2,385 Haswell   76,320 cores

- 1.16 PiB DRAM, 151 GiB MCDRAM (KNL Flat mode)
- DataWarp aka Burst Buffer (1.6 PiB)
- realtime jobs for experimental facililities
- massive quantities of serial jobs
- regular HPC workload
- shifter for Linux containers
- ~30 PB of Lustre scratch, also shared with edison

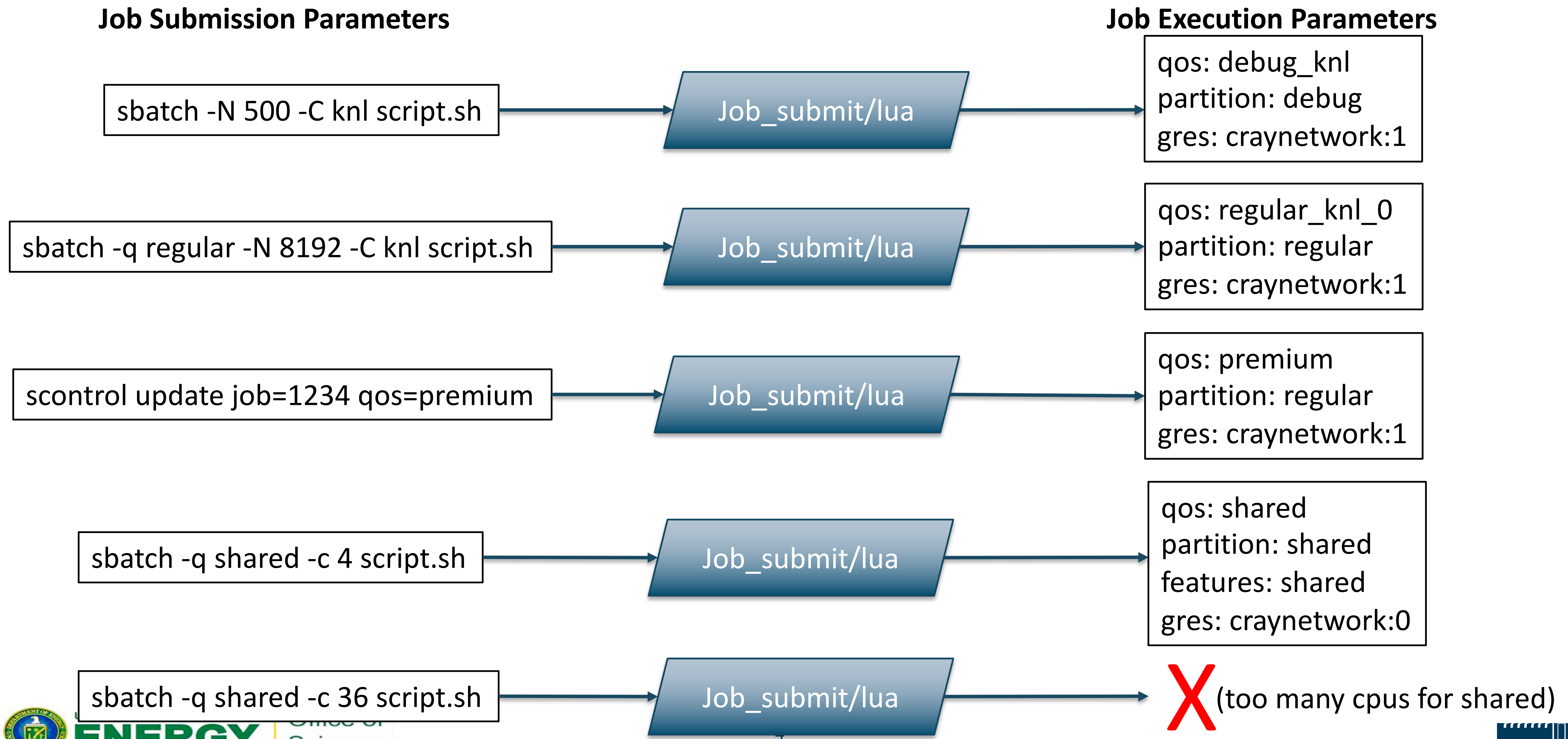
Motivation

The NERSC job_submit.lua was getting difficult to maintain and tended to have some buggy edge cases.

In redesigning the system we wanted a generic abstraction of the process, that met these goals:

- **Aim 1:** Provide simple user interface to batch system
  - Focus user effort on resource requests, not site policies
  - User scripts should rarely change in response to a policy change
  - Enable "backwards compatibility" for policy syntax
- **Aim 2:** Allow site policies to change in a minimally disruptive way
  - Implementation of policy changes should be error free
  - Job submission logic and changes to policy should be traceable
  - Job submission logic should be testable and provably correct before deployment
- **Aim 3:** Separate job submission logic, policies, and user authorization

Motivation: User Interface


Job Submission Parameters

Job Execution Parameters

```
sbatch -N 500 -C knl script.sh
```
Job_submit/lua
```
qos: debug_knl
partition: debug
gres: craynetwork:1
```

```
sbatch -q regular -N 8192 -C knl script.sh
```
Job_submit/lua
```
qos: regular_knl_0
partition: regular
gres: craynetwork:1
```

```
scontrol update job=1234 qos=premium
```
Job_submit/lua
```
qos: premium
partition: regular
gres: craynetwork:1
```

```
sbatch -q shared -c 4 script.sh
```
Job_submit/lua
```
qos: shared
partition: shared
features: shared
gres: craynetwork:0
```

```
sbatch -q shared -c 36 script.sh
```
Job_submit/lua

X (too many cpus for shared)

/etc/slurm/job_submit.lua

```
slurm.log_info("loading job_submit.lua")

package.path = package.path .. ';/usr/lib/nersc-slurm-plugins/?.lua'
local jsl = require "lib_job_submit"

jsl.setupFromYaml("/etc/slurm/policy.yaml")



return slurm.SUCCESS
```

lib_job_submit.lua (The Code)

Library code for implementing generic abstraction of site job policies.

- **Enforces same logic for all job submissions and job modifications**
- **Code design rewards small simple functions with limited scope.**
- **Internally akes extensive use of lua metatables to map and remap functions for match parse and match policies with jobs without using complex procedural code**
- **Full unit test suite, which can run independent of slurm**
  - Some limited capabilities to test policies ahead of deployment

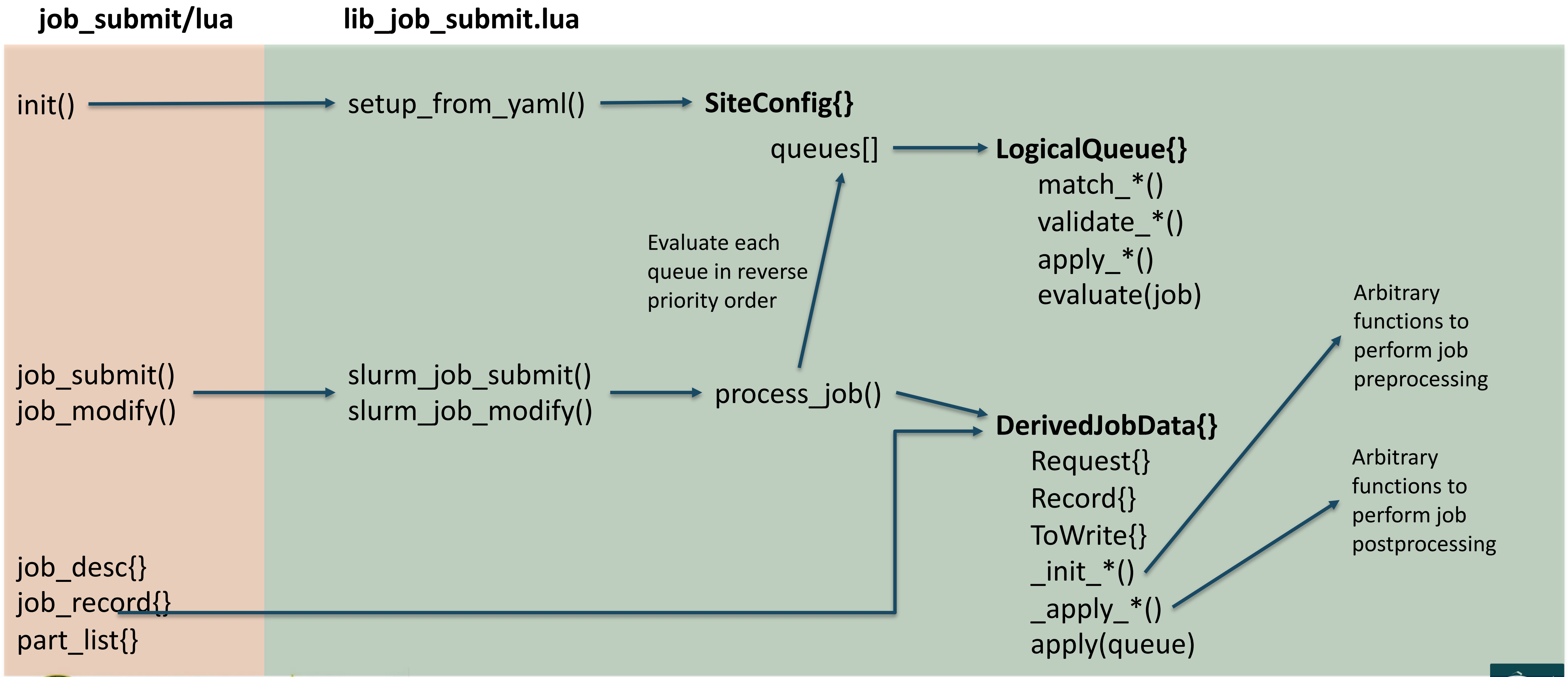
/etc/slurm/policy.yaml (The Data)

Single YAML file that describes:

- **Job submission policies (Logical Queues, system defaults, etc)**
- **QOS names and limits (Not covered today)**
- **License Descriptions and mapping to external resources (Not covered today)**
- **Spank plugin actions (Not covered today)**
  - Actions to take in Job Submission logic
  - Record user requests in AdminComment

YAML document is easily managed, simple to track in git for better auditing of policy change.

General Data Model and Flow



Logical Queues

- **Each LogicalQueue defined in policy.yaml**
  - Each Logical Queue instantiated as a LogicalQueue class instance
  - Methods are functions for evaluating arbitrary expressions
  - LogicalQueue can be inherited from to implement new, site-specific functionality
- **Upon slurm_job_submit() or slurm_job_modify():**
  - Job data is gathered into a DerivedJob instance, initial job processing occurs
  - LogicalQueues are evaluated in reverse priority order (highest to lowest)
  - Evaluation stops upon first full match
  - Final job processing occurs (static, regardless of matched LogicalQueue)
  - LogicalQueue Execution parameters are applied to the job

Logical Queues - Matching Criteria

- **Criteria (key, value) that, if evaluated to True may match a given queue**
- **Can be multiple sets of criteria, subject to an OR (any single criteria can match)**
- **Each criterium may have several matching expressions, ANDed**

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Request* fields are matched against the job_desc data structure, set in either job_submit() or job_modify()

Record* fields are matched against the existing job_record data structure, only set in job_modify()

Logical Queues - Matching Criteria

- **Criteria (key, value) that, if evaluated to True may match a given queue**
- **Can be multiple sets of criteria, subject to an OR (any single criteria can match)**
- **Each criterium may have several matching expressions, ANDed**

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

RequestQos is set to None here to ensure that the job_desc "qos" field is empty. This prevents an scontrol update trying to change a QOS from seeing lower priority LogicalQueues

Logical Queues - Matching Criteria

- Criteria (key, value) that, if evaluated to True may match a given queue
- Can be multiple sets of criteria, subject to an OR (any single criteria can match)
- Each criterium may have several matching expressions, ANDed

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Logical Queues - Matching Criteria

- **Criteria (key, value) that, if evaluated to True may match a given queue**
- **Can be multiple sets of criteria, subject to an OR (any single criteria can match)**
- **Each criterium may have several matching expressions, ANDed**

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Which matches?

```
sbatch --qos regular -N 5 script.sh
```

Logical Queues - Matching Criteria

- **Criteria (key, value) that, if evaluated to True may match a given queue**
- **Can be multiple sets of criteria, subject to an OR (any single criteria can match)**
- **Each criterium may have several matching expressions, ANDed**

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Which matches?

```
sbatch --qos regular -N 5 script.sh
```

Logical Queues - Matching Criteria

- Criteria (key, value) that, if evaluated to True may match a given queue
- Can be multiple sets of criteria, subject to an OR (any single criteria can match)
- Each criterium may have several matching expressions, ANDed

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Logical Queues - Matching Criteria

- **Criteria (key, value) that, if evaluated to True may match a given queue**
- **Can be multiple sets of criteria, subject to an OR (any single criteria can match)**
- **Each criterium may have several matching expressions, ANDed**

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Which matches?

```
sbatch --qos regular -N 5000 script.sh
```

Logical Queues - Matching Criteria

- **Criteria (key, value) that, if evaluated to True may match a given queue**
- **Can be multiple sets of criteria, subject to an OR (any single criteria can match)**
- **Each criterium may have several matching expressions, ANDed**

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Which matches?

```
sbatch --qos regular -N 5000 script.sh
```

Logical Queues - Matching Criteria

- **Criteria (key, value) that, if evaluated to True may match a given queue**
- **Can be multiple sets of criteria, subject to an OR (any single criteria can match)**
- **Each criterium may have several matching expressions, ANDed**

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Logical Queues - Matching Criteria

- **Criteria (key, value) that, if evaluated to True may match a given queue**
- **Can be multiple sets of criteria, subject to an OR (any single criteria can match)**
- **Each criterium may have several matching expressions, ANDed**

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Which matches?

```
$sbatch --qos regular -N 50 script.sh
Submitted batch job 1234
$scontrol update job=1234 NumNodes=80
```

Logical Queues - Matching Criteria

- **Criteria (key, value) that, if evaluated to True may match a given queue**
- **Can be multiple sets of criteria, subject to an OR (any single criteria can match)**
- **Each criterium may have several matching expressions, ANDed**

```
regular_large:
  MatchingCritera:
  - RequestQos: regular
    MinNodes: 1400
  - RecordQos: regular_0
    RecordPartition: regularx
    RequestQos: None
    RequestPartition: None
    MinNodes: 1400
  …
  EvaluationPriority: 100
```

```
regular:
  MatchingCriteria:
  - RequestQos: regular
  - RecordQos: regular_1
    RequestQos: None
  …
  EvaluationPriority: 10
```

Which matches?

```
$sbatch --qos regular -N 50 script.sh
Submitted batch job 1234
$scontrol update job=1234 NumNodes=80
```

Logical Queue: Matching Function in lib_job_submit

```
function LogicalQueue:match_RecordPartition(value, job)
    value = value or {}
    return self._findString(job.Record.partition, value)
end

function LogicalQueue:match_MinNodes(value, job)
    local min_nodes,max_nodes = job:getNodes()
    if not min_nodes or min_nodes < tonumber(value) then
        return false
    end
    return true
end
```

Logical Queues - Requirements

- **Similar syntax and functions as Matching expressions**
- **Used to reject a job if specified requirements are not true**
  - Allows a policy to match a job, but still reject it (useful to provide good error messages)

policy.yaml (in a LogicalQueue):

```
Requirements:
    RequireArchSpecified: true
```

lib_job_submit.lua:

```
function LogicalQueue:validate_RequireArchSpecified(value, job)
    if value and not job.NodeClass then
        local msg = string.format("No hardware architecture specified (-C)!")
        error(SlurmError:new(slurm.ERROR, msg, msg))
    end
    return true
end
```

Logical Queues – Execution Parameters

- **Similar syntax and functions as Matching expressions**
- **Used to rewrite job parameters**
  - Ideally each Execution* function should make only one change.
  - Good to log changes here to expose changes in the slurmctld log

policy.yaml (in a LogicalQueue):

```
Apply:
    ExecutionQos: regular_0
    ExecutionPartition: regular
```

lib_job_submit.lua:

```
function LogicalQueue:apply_ExecutionQos(value, job)
    -- this will get set elsewhere if append account is enabled
    if self.Apply.ExecutionQosAppendAccount then return end

    slurm.log_info("apply_ExecutionQos (LogicalQueue:%s): setting qos to %s",
self.Label, value)
    job.Request.qos = value
end
```

Examples: Default Logical Queue with Legacy Support

- **Goal:** Send jobs to the "debug" logical queue by default, assuring previous user interface is supported. Final qos and partition should be "debug"
- **User Interface:** `sbatch script.sh; sbatch -q debug …; sbatch -p debug …;`

```
queues:
  debug:
    MatchingCriteria:
    # in the case of a no option job
    # submission, need all blank to prevent
    # accidental matching during an update
    - RequestQos: None
      RequestPartition: None
      RecordQos: None
      RecordPartition: None

    # expected job submission (-q debug)
    - RequestQos: debug
      RequestPartition: None
```

```
# old interface (-p debug)
# allow qos if user is verbose
- RequestPartition: debug
  RequestQos: [None, debug]

# match for job update
- RecordQos: debug
  RequestQos: None
  RequestPartition: None
Apply:
  ExecutionQos: debug
  ExecutionPartition: debug
EvaluationPriority: 1
```

Examples: Reservations

- **Goal:** Allow jobs run in advanced reservations to *never* be limited to normal job policies. Rather, delegate limits to the reservation limits.
- **User Interface:** `sbatch --reservation=myres --exclusive …`
- **Implementation:** resv partition and resv qos only matched and used when a reservation is specified.

```
reservation:
    MatchingCriteria:
    - RequestAdvancedReservation: true
      Exclusive: true
    - RecordAdvancedReservation: true
      Exclusive: true
    Requirements:
      RequireArchSpecified: true
    Apply:
      ExecutionQos: resv
      ExecutionPartition: resv
    EvaluationPriority: 2501
```

```
reservation_shared:
    MatchingCriteria:
    - RequestAdvancedReservation: true
      Exclusive: false
    - RecordAdvancedReservation: true
      Exclusive: false
    Requirements:
      RequireMaxCpuPerNodeFraction: 0.5
    Apply:
      ExecutionQos: resv_shared
      ExecutionPartition: resv_shared
      ExecutionArch: haswell
      ExecutionGres: craynetwork:0
    EvaluationPriority: 2500
```

Examples: Special Users with Extraordinary Needs

- **Goal:** Allow slurm account ResGroup1 to get static priority boost of 100000 for a subset of their work (the rest at normal priority), not to exceed 300 nodes simultaneously.
- **User Interface:** `sbatch -q special -A ResGroup1 …`

"special" Logical Queue

- provides a generic method for implementing account-focused QOS policies
- Maintains user interface simplicity and keeps policy management at qos level (not at account level)

policy.yaml changes

```
queues:
  special:
    MatchingCriteria:
    - RequestQos: special
      RequestPartition: None
    - RecordQosRegexMatch: special_%S+
      RequestQos: None
      RequestPartition: None
    Apply:
      ExecutionQos: special
      ExecutionQosAppendAccount: true
      ExecutionPartition: regular
    EvaluationPriority: 999
...
qos:
  special_ResGroup1:
    GrpTRES: nodes=300
    priority: 100000
    MaxSubmitJobsPerUser: 10
```

Conclusions

- **lib_job_submit.lua provides a generic library code for manipulating jobs and reading Slurm state when enforcing policy**
- **/etc/slurm/policy.yaml describes all the system policies**
  – other than those in slurm.conf
  – Each system gets its own policy.yaml
- **Enables focus on desired user interface**
  – Flexibly support new and old interfaces
  – Separate job submission parameters from execution parameters – simplifies policy change management.
- **Managing policy.yaml is *much* easier than managing procedural code for making these decisions**

Cori has 22 logical queues (policy sets)
50 Matching criteria
Two distinct accounting hierarchies (funding sources)
44 Execution QOS in operation

Future Work

- **Abstracting NERSC-specific logic to allow fully generic implementation**
  - Site-specific lua code can be implemented in classes inheriting SiteConfig, DerivedJob, or LogicalQueue
- **Public release of this and all other NERSC spank plugins**
  - http://github.com/NERSC/nersc-slurm-ext
  - Still awaiting open source approval from Lab, DOE
- **Support introspection of Slurm account hierarchies**
  - Allow policies to make decisions based on accounting hierarchy metadata instead of account name (e.g., RecordAccountAncestor: fundingA)
  - Need modifications to job_submit/lua and lib_job_submit.lua
- **Extend policy.yaml for Slurm Federation and pseudo Federation**
  - Enable job submission and verification for slurm federated clusters supporting non-homogeneous policies

NERSC

Thank You.