

Flux: Practical Job Scheduling

Dong H. Ahn, Ned Bass, Al Chu, Jim Garlick, Mark Grondona,
Stephen Herbein, Tapasya Patki, Tom Scogland, Becky Springmeyer

August 15, 2018



What is Flux?

- New Resource and Job Management Software (RJMS) developed here at LLNL
- A way to manage remote resources and execute tasks on them

What is Flux?

- New Resource and Job Management Software (RJMS) developed here at LLNL
- A way to manage **remote resources** and execute tasks on them



What is Flux?

- New Resource and Job Management Software (RJMS) developed here at LLNL
- A way to manage **remote resources** and execute tasks on them



What is Flux?

- New Resource and Job Management Software (RJMS) developed here at LLNL
- A way to manage **remote resources** and execute tasks on them



flickr: dannychamoro



What is Flux?

- New Resource and Job Management Software (RJMS) developed here at LLNL
- A way to manage remote resources and **execute tasks** on them



What is Flux?

- New Resource and Job Management Software (RJMS) developed here at LLNL
- A way to manage remote resources and **execute tasks** on them



What is Flux?

- New Resource and Job Management Software (RJMS) developed here at LLNL
- A way to manage remote resources and execute tasks on them



What about ...?



IBM
Spectrum
LSF



kubernetes



Altair



Adaptive
COMPUTING



HTCondor
High Throughput Computing

slurm
workload manager

What about ...?



IBM
Spectrum
LSF



kubernetes



Altair



Adaptive
COMPUTING



HTC Condor
High Throughput Computing

slurm
workload manager

Closed-source

What about ...?



IBM
Spectrum
LSF



kubernetes



Altair



Adaptive
COMPUTING



HTCCondor
High Throughput Computing



slurm
workload manager

Not designed for HPC

What about ...?



IBM
Spectrum
LSF



kubernetes



Altair



Adaptive
COMPUTING



HTCCondor
High Throughput Computing

The Slurm logo, a grid of white squares of varying sizes.

slurm
workload manager

Limited Scalability, Usability, and Portability

Why Flux?

Why Flux?

- Extensibility
 - Open source
 - Modular design with support for user plugins

Why Flux?

- Extensibility
 - Open source
 - Modular design with support for user plugins
- Scalability
 - Designed from the ground up for exascale and beyond
 - Already tested at 1000s of nodes & millions of jobs

Why Flux?

- **Extensibility**
 - Open source
 - Modular design with support for user plugins
- **Scalability**
 - Designed from the ground up for exascale and beyond
 - Already tested at 1000s of nodes & millions of jobs
- **Usability**
 - C, Lua, and Python bindings that expose 100% of Flux's functionality
 - Can be used as a single-user tool or a system scheduler

Why Flux?

- **Extensibility**
 - Open source
 - Modular design with support for user plugins
- **Scalability**
 - Designed from the ground up for exascale and beyond
 - Already tested at 1000s of nodes & millions of jobs
- **Usability**
 - C, Lua, and Python bindings that expose 100% of Flux's functionality
 - Can be used as a single-user tool or a system scheduler
- **Portability**
 - Optimized for HPC and runs in Cloud and Grid settings too
 - Runs on any set of Linux machines: only requires a list of IP addresses or PMI

Why Flux?

- **Extensibility**
 - Open source
 - Modular design with support for user plugins
- **Scalability**
 - Designed from the ground up for exascale and beyond
 - Already tested at 1000s of nodes & millions of jobs
- **Usability**
 - C, Lua, and Python bindings that expose 100% of Flux's functionality
 - Can be used as a single-user tool or a system scheduler
- **Portability**
 - Optimized for HPC and runs in Cloud and Grid settings too
 - Runs on any set of Linux machines: only requires a list of IP addresses or PMI

Flux is designed to make hard scheduling problems easy

Portability: Running Flux

Portability: Running Flux

- Already installed on LC systems (including Sierra)
 - spack install flux-sched for everywhere else

Portability: Running Flux

- Already installed on LC systems (including Sierra)
 - spack install flux-sched for everywhere else
- Flux can run anywhere that MPI can run, (via PMI - Process Management Interface)
 - Inside a resource allocation from: itself (hierarchical Flux), Slurm, Moab, PBS, LSF, etc
 - flux start **OR** srun flux start

Portability: Running Flux

- Already installed on LC systems (including Sierra)
 - `spack install flux-sched` for everywhere else
- Flux can run anywhere that MPI can run, (via PMI - Process Management Interface)
 - Inside a resource allocation from: itself (hierarchical Flux), Slurm, Moab, PBS, LSF, etc
 - `flux start` **OR** `srun flux start`
- Flux can run anywhere that supports TCP and you have the IP addresses
 - `flux broker -Sboot.method=config -Sboot.config_file=boot.conf`
 - `boot.conf`:

```
session-id = "mycluster"
tbon-endpoints = [
    "tcp://192.168.1.1:8020",
    "tcp://192.168.1.2:8020",
    "tcp://192.168.1.3:8020"]
```

Why Flux?

- **Extensibility**
 - Open source
 - Modular design with support for user plugins
- **Scalability**
 - Designed from the ground up for exascale and beyond
 - Already tested at 1000s of nodes & millions of jobs
- **Usability**
 - C, Lua, and Python bindings that expose 100% of Flux's functionality
 - Can be used as a single-user tool or a system scheduler
- **Portability**
 - Optimized for HPC and runs in Cloud and Grid settings too
 - Runs on any set of Linux machines: only requires a list of IP addresses or PMI

Usability: Submitting a Batch Job

Usability: Submitting a Batch Job

- Slurm

- `sbatch -N2 -n4 -t 2:00 sleep 120`

Usability: Submitting a Batch Job

- Slurm
 - `sbatch -N2 -n4 -t 2:00 sleep 120`
- Flux CLI
 - `flux submit -N2 -n4 -t 2m sleep 120`

Usability: Submitting a Batch Job

- Slurm

- `sbatch -N2 -n4 -t 2:00 sleep 120`

- Flux CLI

- `flux submit -N2 -n4 -t 2m sleep 120`

- Flux API:

```
import json, flux
```

```
jobreq = {  
    'nnodes'    : 2,  
    'ntasks'    : 4,  
    'walltime'  : 120,  
    'cmdline'   : ["sleep", "120"]}
```

```
f = flux.Flux ()  
resp = f.rpc_send ("job.submit", json.dumps(jobreq))
```

Usability: Running an Interactive Job

Usability: Running an Interactive Job

- Slurm

- `srun -N2 -n4 -t 2:00 sleep 120`

Usability: Running an Interactive Job

- Slurm

- `srun -N2 -n4 -t 2:00 sleep 120`

- Flux CLI

- `flux wreckrun -N2 -n4 -t 2m sleep 120`

Usability: Running an Interactive Job

- Slurm

- `srun -N2 -n4 -t 2:00 sleep 120`

- Flux CLI

- `flux wreckrun -N2 -n4 -t 2m sleep 120`

- Flux API:

```
import sys
from flux import kz
```

```
resp = f.rpc_send ("job.submit", json.dumps(jobreq))
kvs_dir = resp['kvs_dir']
```

```
for task_id in range(jobreq['ntasks']):
    kz.attach (f, "{}.{}.stdout".format(kvs_dir, task_id), sys.stdout)
```

```
f.reactor_run (f.get_reactor (), 0)
```

Usability: Tracking Job Status

Usability: Tracking Job Status

- CLI: slow, non-programmatic, inconvenient to parse
 - `watch queue -j JOBID`
 - `watch flux wreck ls JOBID`

Usability: Tracking Job Status

- CLI: slow, non-programmatic, inconvenient to parse
 - `watch squeue -j JOBID`
 - `watch flux wreck ls JOBID`
- Tracking via the filesystem
 - `date > $JOBID.start; srun myApp; date > $JOBID.stop`

Usability: Tracking Job Status

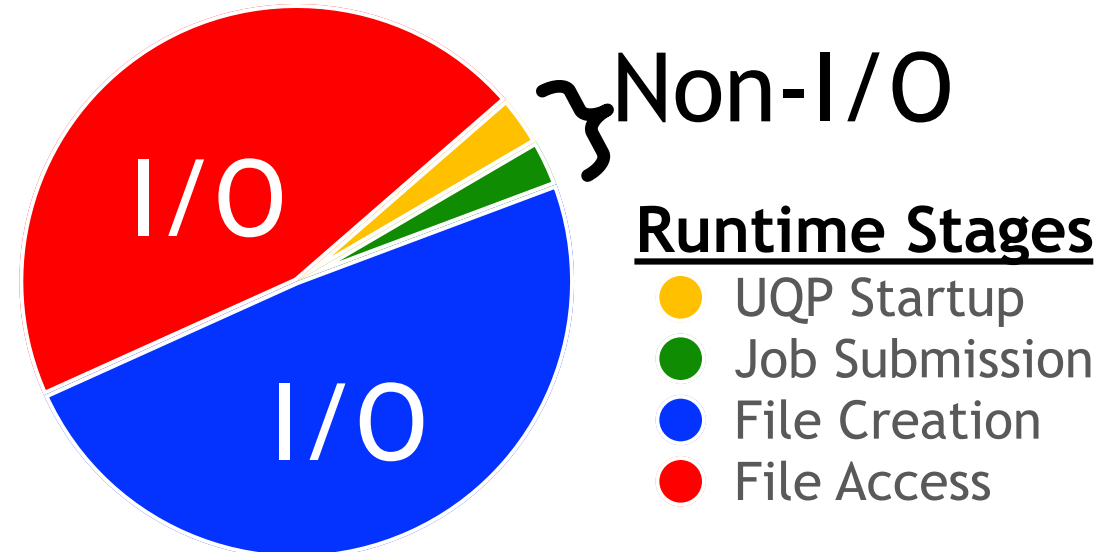
- CLI: slow, non-programmatic, inconvenient to parse
 - `watch squeue -j JOBID`
 - `watch flux wreck ls JOBID`
- Tracking via the filesystem
 - `date > $JOBID.start; srun myApp; date > $JOBID.stop`

```
→ quota -vf ~/quota.conf
Disk quotas for herbein1:
Filesystem      used  quota  limit  files
/p/lscratchrza 760.3G n/a     n/a     8.6M
```

Usability: Tracking Job Status

- CLI: slow, non-programmatic, inconvenient to parse
 - watch squeue -j JOBID
 - watch flux wreck ls JOBID
- Tracking via the filesystem
 - date > \$JOBID.start; srun myApp; date > \$JOBID.stop

```
→ quota -vf ~/quota.conf
Disk quotas for herbein1:
Filesystem      used      quota  limit  files
/p/lscratchrza 760.3G  n/a    n/a    8.6M
```



Usability: Tracking Job Status

- CLI: slow, non-programmatic, inconvenient to parse

- watch squeue -j JOBID
 - watch flux wreck ls JOBID

- Tracking via the filesystem

- date > \$JOBID.start; srun myApp; date > \$JOBID.stop

- Push notification via Flux's Job Status and Control (JSC):

```
def jsc_cb (jcbstr, arg, errnum):  
    jcb = json.loads (jcbstr)  
    jobid = jcb['jobid']  
    state = jsc.job_num2state (jcb[jsc.JSC_STATE_PAIR][jsc.JSC_STATE_PAIR_NSTATE])  
    print "flux.jsc: job", jobid, "changed its state to ", state
```

```
jsc.notify_status (f, jsc_cb, None)
```

Why Flux?

- **Extensibility**
 - Open source
 - Modular design with support for user plugins
- **Scalability**
 - Designed from the ground up for exascale and beyond
 - Already tested at 1000s of nodes & millions of jobs
- **Usability**
 - C, Lua, and Python bindings that expose 100% of Flux's functionality
 - Can be used as a single-user tool or a system scheduler
- **Portability**
 - Optimized for HPC and runs in Cloud and Grid settings too
 - Runs on any set of Linux machines: only requires a list of IP addresses or PMI

Scalability: Running Many Jobs

Scalability: Running Many Jobs

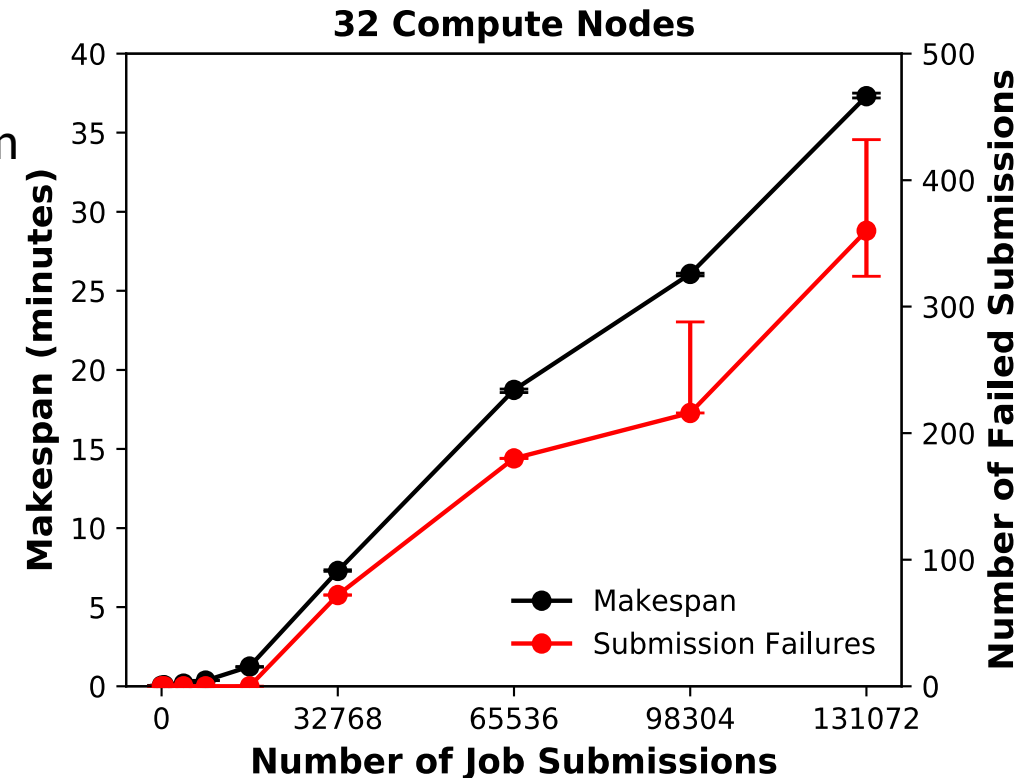
■ Slurm

- `find ./ -exec sbatch -N1 tar -cf {}.tgz {} \;`
 - Slow: requires acquiring a lock in Slurm, can timeout causing failures
 - Inefficient: uses 1 node for each task
- `find ./ -exec srun -n1 tar -cf {}.tgz {} \;`
 - Slow: spawns a process for every submission
 - Inefficient: is not a true scheduler - can overlap tasks on cores

Scalability: Running Many Jobs

■ Slurm

- `find ./ -exec sbatch -N1 tar -cf {}.tgz {} \;`
 - Slow: requires acquiring a lock in Slurm, can timeout causing failures
 - Inefficient: uses 1 node for each task
- `find ./ -exec srun -n1 tar -cf {}.tgz {} \;`
 - Slow: spawns a process for every submission
 - Inefficient: is not a true scheduler - can overlap tasks on



Scalability: Running Many Jobs

- Slurm

- `find ./ -exec sbatch -N1 tar -cf {}.tgz {} \;`
 - Slow: requires acquiring a lock in Slurm, can timeout causing failures
 - Inefficient: uses 1 node for each task
- `find ./ -exec srun -n1 tar -cf {}.tgz {} \;`
 - Slow: spawns a process for every submission
 - Inefficient: is not a true scheduler - can overlap tasks on cores

Scalability: Running Many Jobs

- Slurm

- `find ./ -exec sbatch -N1 tar -cf {}.tgz {} \;`
 - Slow: requires acquiring a lock in Slurm, can timeout causing failures
 - Inefficient: uses 1 node for each task
- `find ./ -exec srun -n1 tar -cf {}.tgz {} \;`
 - Slow: spawns a process for every submission
 - Inefficient: is not a true scheduler - can overlap tasks on cores

- Flux API:

```
for f in $(ls .):  
    payload['command'] = ["tar", "-cf", "{}.tgz".format(f), f]  
    resp = f.rpc_send("job.submit", payload)
```

Scalability: Running Many Jobs

■ Slurm

- `find ./ -exec sbatch -N1 tar -cf {}.tgz {} \;`
 - Slow: requires acquiring a lock in Slurm, can timeout causing failure
 - Inefficient: uses 1 node for each task
- `find ./ -exec srun -n1 tar -cf {}.tgz {} \;`
 - Slow: spawns a process for every submission
 - Inefficient: is not a true scheduler - can overlap tasks on cores

■ Flux API:

```
for f in os.listdir('.'):
    payload['command'] = ["tar", "-cf", "{}.tgz".format(f), f]
    resp = f.rpc_send("job.submit", payload)
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	MODELITTON	(REASON)
1306868	pbatch	F_114.20	golo	PD	0:00	1	(Priority)	
1306858	pbatch	F_118.18	golo	PD	0:00	1	(Priority)	
1306910	pbatch	F_103.24	golo	PD	0:00	1	(Priority)	
1306872	pbatch	F_113.19	golo	PD	0:00	1	(Priority)	
1306888	pbatch	F_113.18	golo	PD	0:00	1	(Priority)	
1306912	pbatch	F_123.24	golo	PD	0:00	1	(Priority)	
1306913	pbatch	F_111.24	golo	PD	0:00	1	(Priority)	
1306914	pbatch	F_112.24	golo	PD	0:00	1	(Priority)	
1306915	pbatch	F_166.31	golo	PD	0:00	1	(Priority)	
1306916	pbatch	F_107.25	golo	PD	0:00	1	(Priority)	
1306917	pbatch	F_141.27	golo	PD	0:00	1	(Priority)	
1306918	pbatch	F_129.26	golo	PD	0:00	1	(Priority)	
1306919	pbatch	F_122.23	golo	PD	0:00	1	(Priority)	
1306920	pbatch	F_117.25	golo	PD	0:00	1	(Priority)	
1307080	pbatch	F_129.27	golo	PD	0:00	1	(Priority)	
1307081	pbatch	F_141.26	golo	PD	0:00	1	(Priority)	
1307082	pbatch	F_130.28	golo	PD	0:00	1	(Priority)	
1307083	pbatch	F_164.29	golo	PD	0:00	1	(Priority)	
1307084	pbatch	F_135.26	golo	PD	0:00	1	(Priority)	
1307085	pbatch	F_169.27	golo	PD	0:00	1	(Priority)	
1307086	pbatch	F_122.23	golo	PD	0:00	1	(Priority)	
1307087	pbatch	F_106.23	golo	PD	0:00	1	(Priority)	
1307088	pbatch	F_170.28	golo	PD	0:00	1	(Priority)	
1307089	pbatch	F_169.27	golo	PD	0:00	1	(Priority)	
1307091	pbatch	F_135.26	golo	PD	0:00	1	(Priority)	
1307092	pbatch	F_113.19	golo	PD	0:00	1	(Priority)	
1307093	pbatch	F_170.28	golo	PD	0:00	1	(Priority)	
1307094	pbatch	F_107.25	golo	PD	0:00	1	(Priority)	
1307095	pbatch	F_122.23	golo	PD	0:00	1	(Priority)	
1307096	pbatch	F_141.27	golo	PD	0:00	1	(Priority)	
1307097	pbatch	F_163.26	golo	PD	0:00	1	(Priority)	
1307098	pbatch	F_135.27	golo	PD	0:00	1	(Priority)	
1307099	pbatch	F_106.24	golo	PD	0:00	1	(Priority)	
1307100	pbatch	F_129.26	golo	PD	0:00	1	(Priority)	
1307101	pbatch	F_112.25	golo	PD	0:00	1	(Priority)	
1307102	pbatch	F_106.24	golo	PD	0:00	1	(Priority)	
1307103	pbatch	F_135.26	golo	PD	0:00	1	(Priority)	
1307104	pbatch	F_117.25	golo	PD	0:00	1	(Priority)	
1307105	pbatch	F_170.28	golo	PD	0:00	1	(Priority)	
1307106	pbatch	F_135.27	golo	PD	0:00	1	(Priority)	
1307107	pbatch	F_164.28	golo	PD	0:00	1	(Priority)	
1307108	pbatch	F_106.23	golo	PD	0:00	1	(Priority)	
1307109	pbatch	F_117.25	golo	PD	0:00	1	(Priority)	
1307110	pbatch	F_122.23	golo	PD	0:00	1	(Priority)	
1307111	pbatch	F_123.24	golo	PD	0:00	1	(Priority)	
1307112	pbatch	F_135.26	golo	PD	0:00	1	(Priority)	
1307113	pbatch	F_121.22	golo	PD	0:00	1	(Priority)	
1307114	pbatch	F_111.24	golo	PD	0:00	1	(Priority)	
1307115	pbatch	F_112.24	golo	PD	0:00	1	(Priority)	
1307116	pbatch	F_107.25	golo	PD	0:00	1	(Priority)	
1307117	pbatch	F_122.22	golo	PD	0:00	1	(Priority)	
1307118	pbatch	F_135.26	golo	PD	0:00	1	(Priority)	
1307119	pbatch	F_135.26	golo	PD	0:00	1	(Priority)	
1307120	pbatch	F_121.22	golo	PD	0:00	1	(Priority)	
1307121	pbatch	F_106.24	golo	PD	0:00	1	(Priority)	
1307122	pbatch	F_117.25	golo	PD	0:00	1	(Priority)	
1307123	pbatch	F_117.25	golo	PD	0:00	1	(Priority)	
1307124	pbatch	F_107.25	golo	PD	0:00	1	(Priority)	

Scalability: Running Many Jobs

■ Slurm

- `find ./ -exec sbatch -N1 tar -cf {}.tgz {} \;`
 - Slow: requires acquiring a lock in Slurm, can timeout causing failure
 - Inefficient: uses 1 node for each task
- `find ./ -exec srun -n1 tar -cf {}.tgz {} \;`
 - Slow: spawns a process for every submission
 - Inefficient: is not a true scheduler - can overlap tasks on cores

■ Flux API:

```
for f in os.listdir('.'):
    payload['command'] = ["tar", "-cf", "{}.tgz".format(f), f]
    resp = f.rpc_send("job.submit", payload)
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NOELIST(REASON)
1306868	pbatch	F_114.20	golo	PD	0:00	1	(Priority)
1306858	pbatch	F_118.18	golo	PD	0:00	1	(Priority)
1306910	pbatch	F_103.24	golo	PD	0:00	1	(Priority)
1306872	pbatch	F_113.19	golo	PD	0:00	1	(Priority)
1306888	pbatch	F_113.18	golo	PD	0:00	1	(Priority)
1306912	pbatch	F_123.24	golo	PD	0:00	1	(Priority)
1306913	pbatch	F_111.24	golo	PD	0:00	1	(Priority)
1306914	pbatch	F_112.24	golo	PD	0:00	1	(Priority)
1306915	pbatch	F_166.31	golo	PD	0:00	1	(Priority)
1306916	pbatch	F_107.25	golo	PD	0:00	1	(Priority)
1306917	pbatch	F_141.27	golo	PD	0:00	1	(Priority)
1306918	pbatch	F_129.26	golo	PD	0:00	1	(Priority)
1306919	pbatch	F_122.23	golo	PD	0:00	1	(Priority)
1306920	pbatch	F_117.25	golo	PD	0:00	1	(Priority)
1307080	pbatch	F_129.27	golo	PD	0:00	1	(Priority)
1307081	pbatch	F_141.26	golo	PD	0:00	1	(Priority)
1307082	pbatch	F_130.28	golo	PD	0:00	1	(Priority)
1307083	pbatch	F_164.29	golo	PD	0:00	1	(Priority)
1307084	pbatch	F_135.26	golo	PD	0:00	1	(Priority)
1307085	pbatch	F_169.27	golo	PD	0:00	1	(Priority)
1307086	pbatch	F_122.23	golo	PD	0:00	1	(Priority)
1307087	pbatch	F_106.23	golo	PD	0:00	1	(Priority)
1307088	pbatch	F_170.28	golo	PD	0:00	1	(Priority)
1307089	pbatch	F_169.27	golo	PD	0:00	1	(Priority)
1307091	pbatch	F_135.26	golo	PD	0:00	1	(Priority)
1307092	pbatch	F_113.19	golo	PD	0:00	1	(Priority)
1307093	pbatch	F_170.28	golo	PD	0:00	1	(Priority)
1307094	pbatch	F_107.25	golo	PD	0:00	1	(Priority)
1307095	pbatch	F_122.23	golo	PD	0:00	1	(Priority)
1307096	pbatch	F_141.27	golo	PD	0:00	1	(Priority)
1307097	pbatch	F_163.26	golo	PD	0:00	1	(Priority)
1307098	pbatch	F_135.27	golo	PD	0:00	1	(Priority)
1307099	pbatch	F_106.24	golo	PD	0:00	1	(Priority)
1307100	pbatch	F_129.26	golo	PD	0:00	1	(Priority)
1307101	pbatch	F_112.25	golo	PD	0:00	1	(Priority)
1307102	pbatch	F_106.24	golo	PD	0:00	1	(Priority)
1307103	pbatch	F_135.26	golo	PD	0:00	1	(Priority)

Subject: Good Neighbor Policy

You currently have 271 jobs in the batch system on lamoab.

The good neighbor policy is that users keep their maximum submitted job count at a maximum of 200 or less. Please try to restrict yourself to this limit in the future. Thank you.

1307121	pbatch	F_106.24	golo	PD	0:00	1	(Priority)
1307122	pbatch	F_117.25	golo	PD	0:00	1	(Priority)
1307123	pbatch	F_117.25	golo	PD	0:00	1	(Priority)
1307124	pbatch	F_107.25	golo	PD	0:00	1	(Priority)

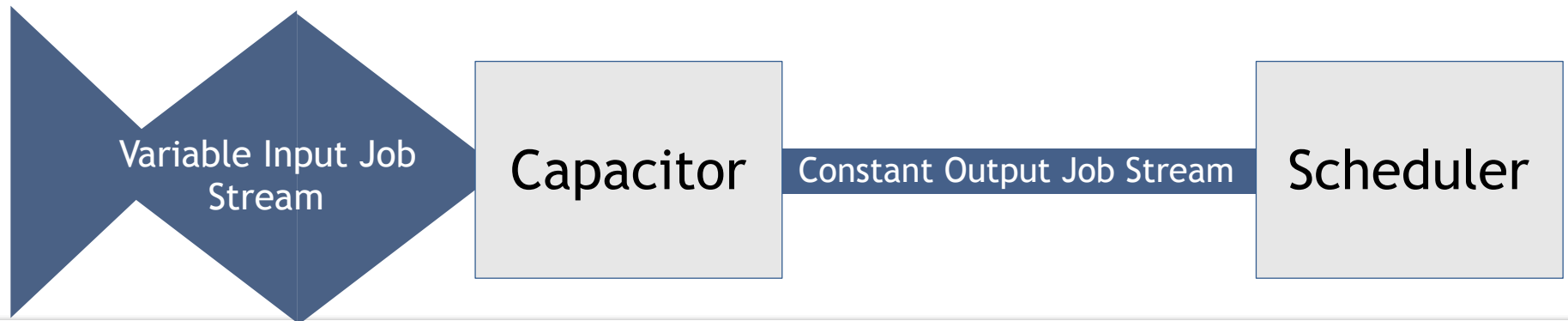
Scalability: Running Many Jobs

■ Slurm

- `find ./ -exec sbatch -N1 tar -cf {}.tgz {} \;`
 - Slow: requires acquiring a lock in Slurm, can timeout causing failures
 - Inefficient: uses 1 node for each task
- `find ./ -exec srun -n1 tar -cf {}.tgz {} \;`
 - Slow: spawns a process for every submission
 - Inefficient: is not a true scheduler - can overlap tasks on cores

■ Flux API:

```
for f in os.listdir('.'):
    payload['command'] = ["tar", "-cf", "{}.tgz".format(f), f]
    resp = f.rpc_send("job.submit", payload)
```



Scalability: Running Many Jobs

■ Slurm

- `find ./ -exec sbatch -N1 tar -cf {}.tgz {} \;`
 - Slow: requires acquiring a lock in Slurm, can timeout causing failures
 - Inefficient: uses 1 node for each task
- `find ./ -exec srun -n1 tar -cf {}.tgz {} \;`
 - Slow: spawns a process for every submission
 - Inefficient: is not a true scheduler - can overlap tasks on cores

■ Flux API:

```
for f in os.listdir('.'):
    payload['command'] = ["tar", "-cf", "{}.tgz".format(f), f]
    resp = f.rpc_send("job.submit", payload)
```

■ Flux Capacitor

- `find ./ -printf -n1 tar -cf %p.tgz %p | flux-capacitor`
- `flux-capacitor --command_file my_command_file`
 - `-n1 tar -cf dirA.tgz ./dirA`
 - `-n1 tar -cf dirB.tgz ./dirB`
 - `-n1 tar -cf dirC.tgz ./dirC`

Scalability: Running Many Heterogeneous Jobs

Scalability: Running Many Heterogeneous Jobs

- Slurm
 - No support for heterogeneous job steps in versions before 17.11
 - Limited support in versions after 17.11

Scalability: Running Many Heterogeneous Jobs

■ Slurm

- No support for heterogeneous job steps in v
- Limited support in versions after 17.11

Limitations

The backfill scheduler has limitations in how it tracks usage of CPUs and memory in the future. This typically requires the backfill scheduler be able to allocate each component of a heterogeneous job on a different node in order to begin its resource allocation, even if multiple components of the job do actually get allocated resources on the same node.

In a federation of clusters, a heterogeneous job will execute entirely on the cluster from which the job is submitted. The heterogeneous job will not be eligible to migrate between clusters or to have different components of the job execute on different clusters in the federation.

Job arrays of heterogeneous jobs are not supported.

The `srun` command's `--no-allocate` option is not supported for heterogeneous jobs.

Only one job step per heterogeneous job component can be launched by a single `srun` command (e.g. "`srun --pack-group=0 alpha : --pack-group=0 beta`" is not supported).

The `sattach` command can only be used to attach to a single component of a heterogeneous job at a time.

Heterogeneous jobs are only scheduled by the backfill scheduler plugin. The more frequently executed scheduling logic only starts jobs on a first-in first-out (FIFO) basis and lacks logic for concurrently scheduling all components of a heterogeneous job.

Heterogeneous jobs are not supported with Slurm's `select/serial` plugin.

Heterogeneous jobs are not supported on Cray ALPS systems.

Heterogeneous jobs are not supported on IBM PE systems.

Slurm's PERL APIs currently do not support heterogeneous jobs.

The `srun --multi-prog` option can not be used to span more than one heterogeneous job component.

The `srun --open-mode` option is by default set to "append".

https://slurm.schedmd.com/heterogeneous_jobs.html#limitations



Scalability: Running Many Heterogeneous Jobs

- Slurm

- No support for heterogeneous job steps in versions before 17.11
- Limited support in versions after 17.11

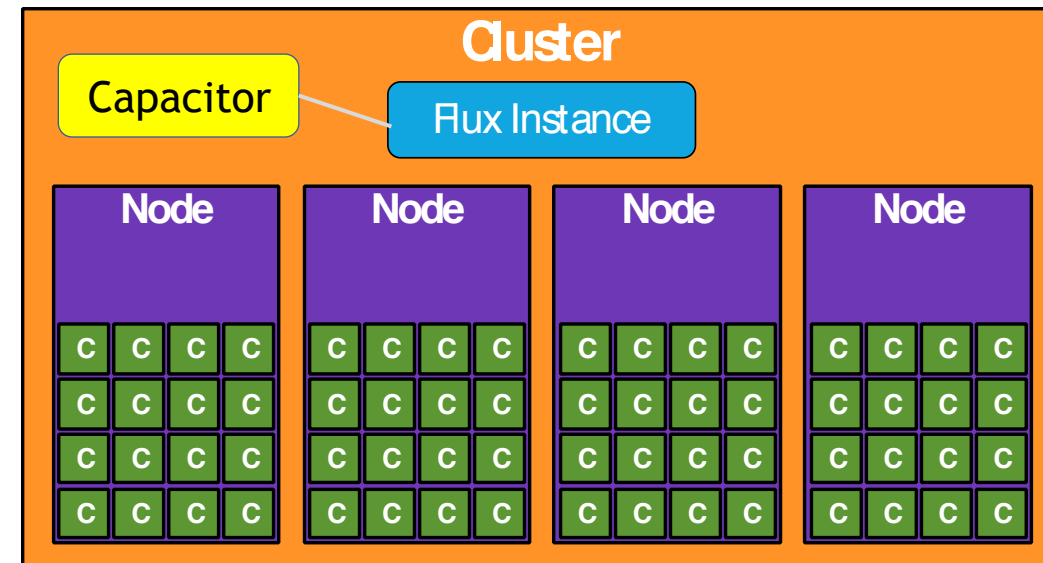
- Flux Capacitor

- `flux-capacitor --command_file my_command_file`
 - `-n1 tar -cf dirA.tgz ./dirA`
 - `-n32 make -j 32`
 - `-N4 my_mpi_app`
 - ...

Scalability: Running Millions of Jobs

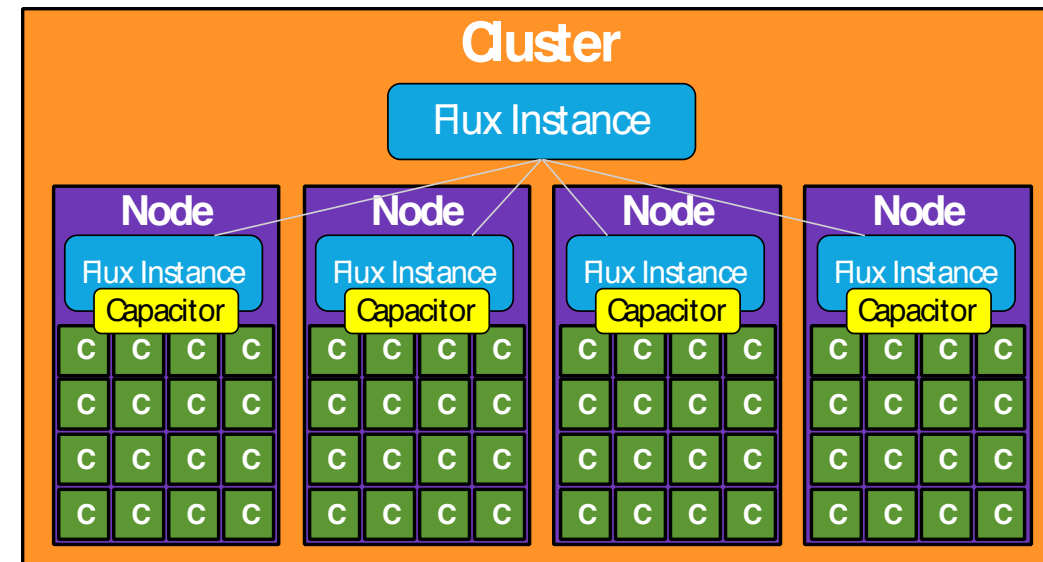
Scalability: Running Millions of Jobs

- Flux Capacitor (Depth-1)
 - `flux-capacitor --command_file my_command_file`



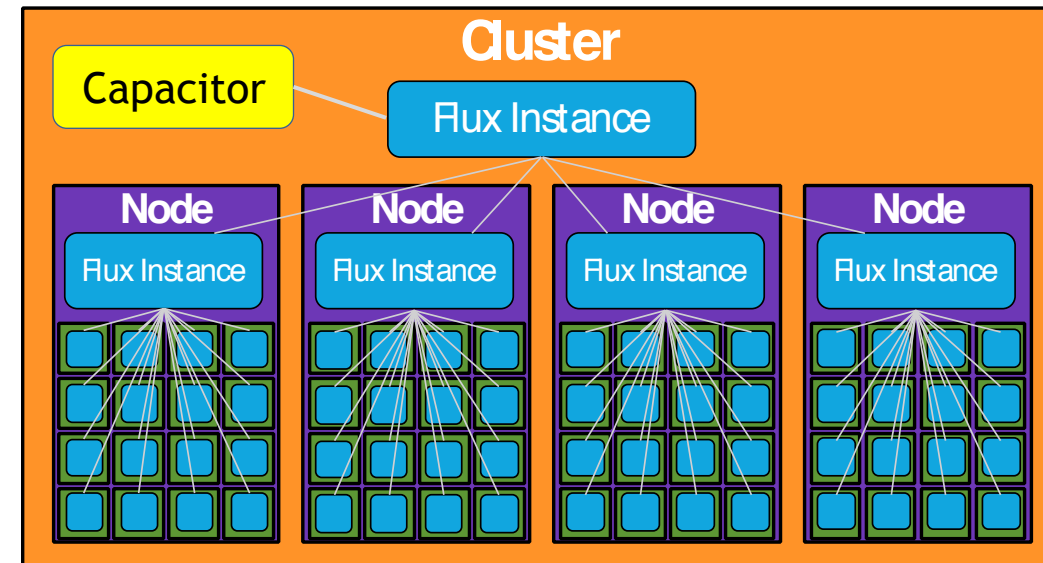
Scalability: Running Millions of Jobs

- Flux Capacitor (Depth-1)
 - `flux-capacitor --command_file my_command_file`
- Hierarchical Flux Capacitor (Depth-2)
 - ```
for x in ./*.commands; do
 flux submit -N1 flux start \
 flux-capacitor --command_file $x
done
```



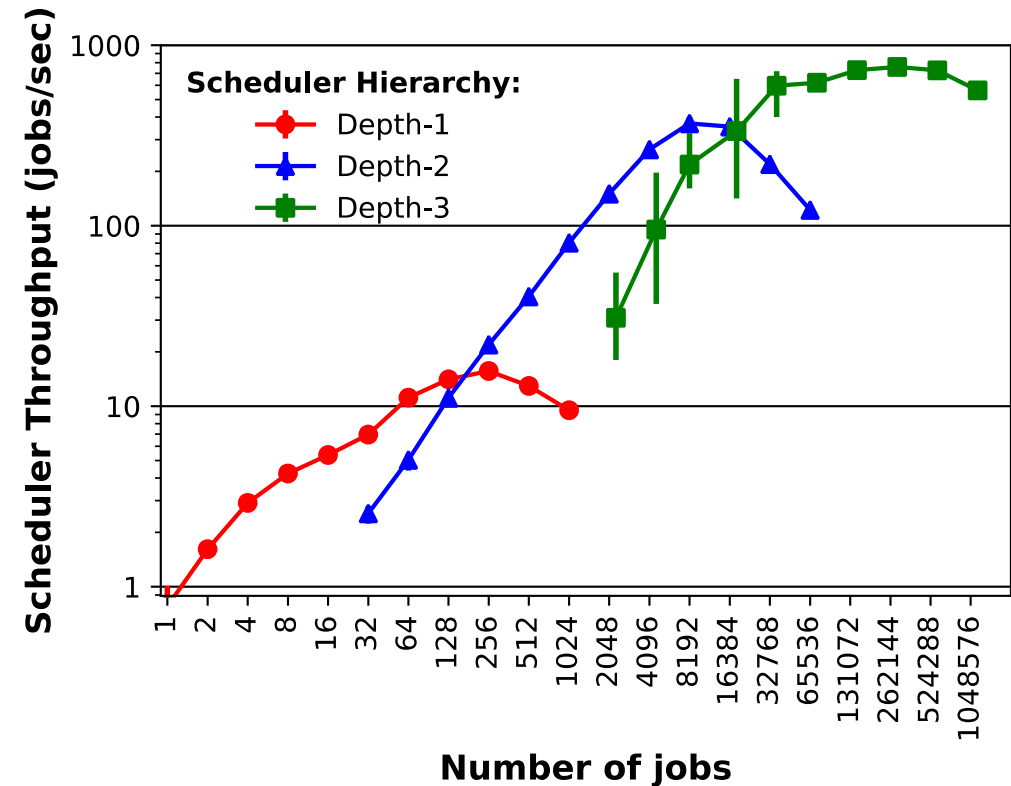
# Scalability: Running Millions of Jobs

- Flux Capacitor (Depth-1)
  - `flux-capacitor --command_file my_command_file`
- Hierarchical Flux Capacitor (Depth-2)
  - ```
for x in ./*.commands; do  
    flux submit -N1 flux start \  
        flux-capacitor --command_file $x  
done
```
- Flux Hierarchy (Depth-3+)
 - `flux-hierarchy --config=config.json`
– `--command_file my_command_file`



Scalability: Running Millions of Jobs

- Flux Capacitor (Depth-1)
 - `flux-capacitor --command_file my_command_file`
- Hierarchical Flux Capacitor (Depth-2)
 - ```
for x in ./*.commands; do
 flux submit -N1 flux start \
 flux-capacitor --command_file $x
done
```
- Flux Hierarchy (Depth-3+)
  - `flux-hierarchy --config=config.json`
  - `--command_file my_command_file`





# Why Flux?

- **Extensibility**
  - Open source
  - Modular design with support for user plugins
- **Scalability**
  - Designed from the ground up for exascale and beyond
  - Already tested at 1000s of nodes & millions of jobs
- **Usability**
  - C, Lua, and Python bindings that expose 100% of Flux's functionality
  - Can be used as a single-user tool or a system scheduler
- **Portability**
  - Optimized for HPC and runs in Cloud and Grid settings too
  - Runs on any set of Linux machines: only requires a list of IP addresses or PMI

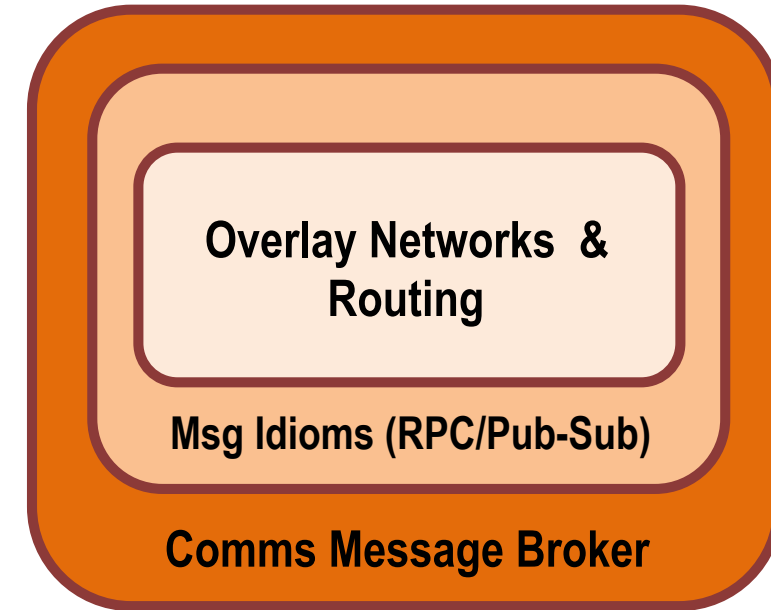
# Extensibility: Modular Design

---

# Extensibility: Modular Design

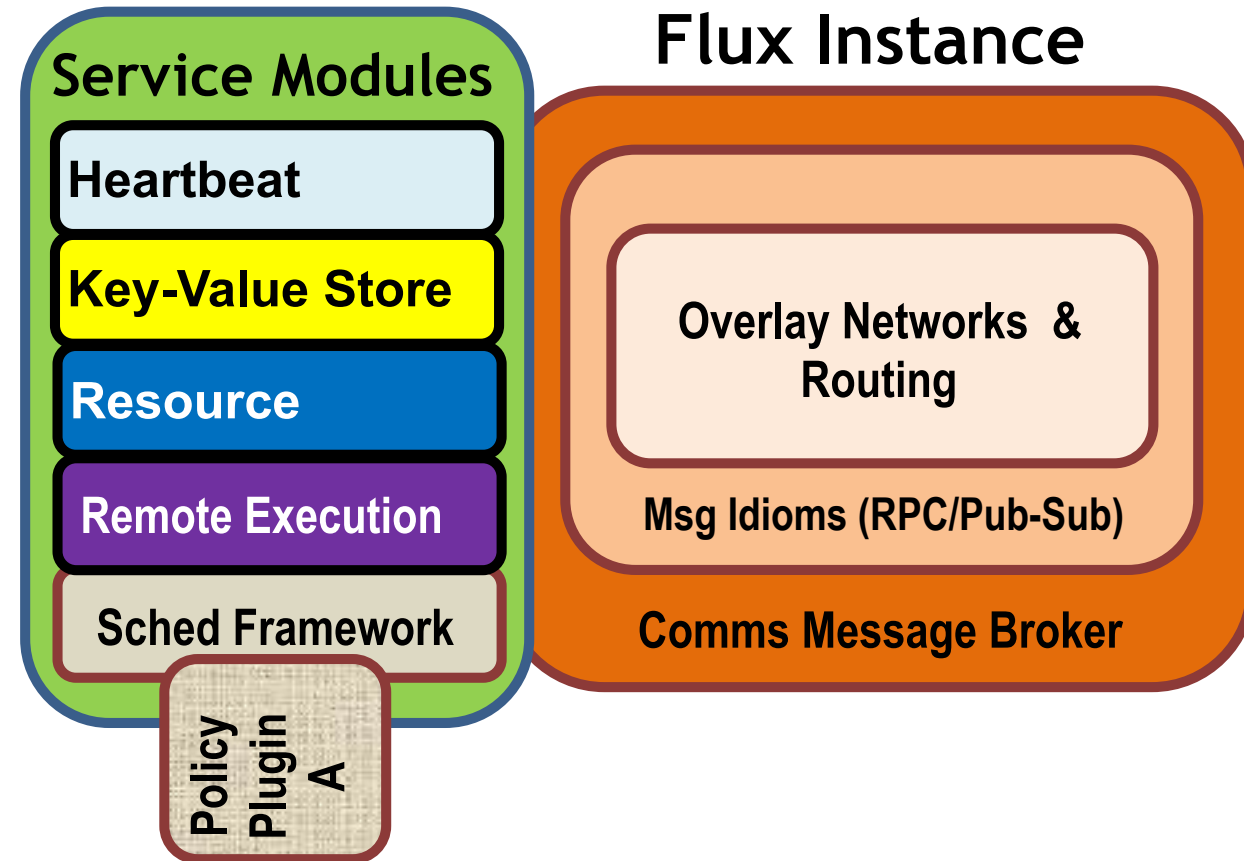
- At the core of Flux is an overlay network
  - Built on top of ZeroMQ
  - Supports RPCs, Pub/Sub, Push/Pull, etc

## Flux Instance



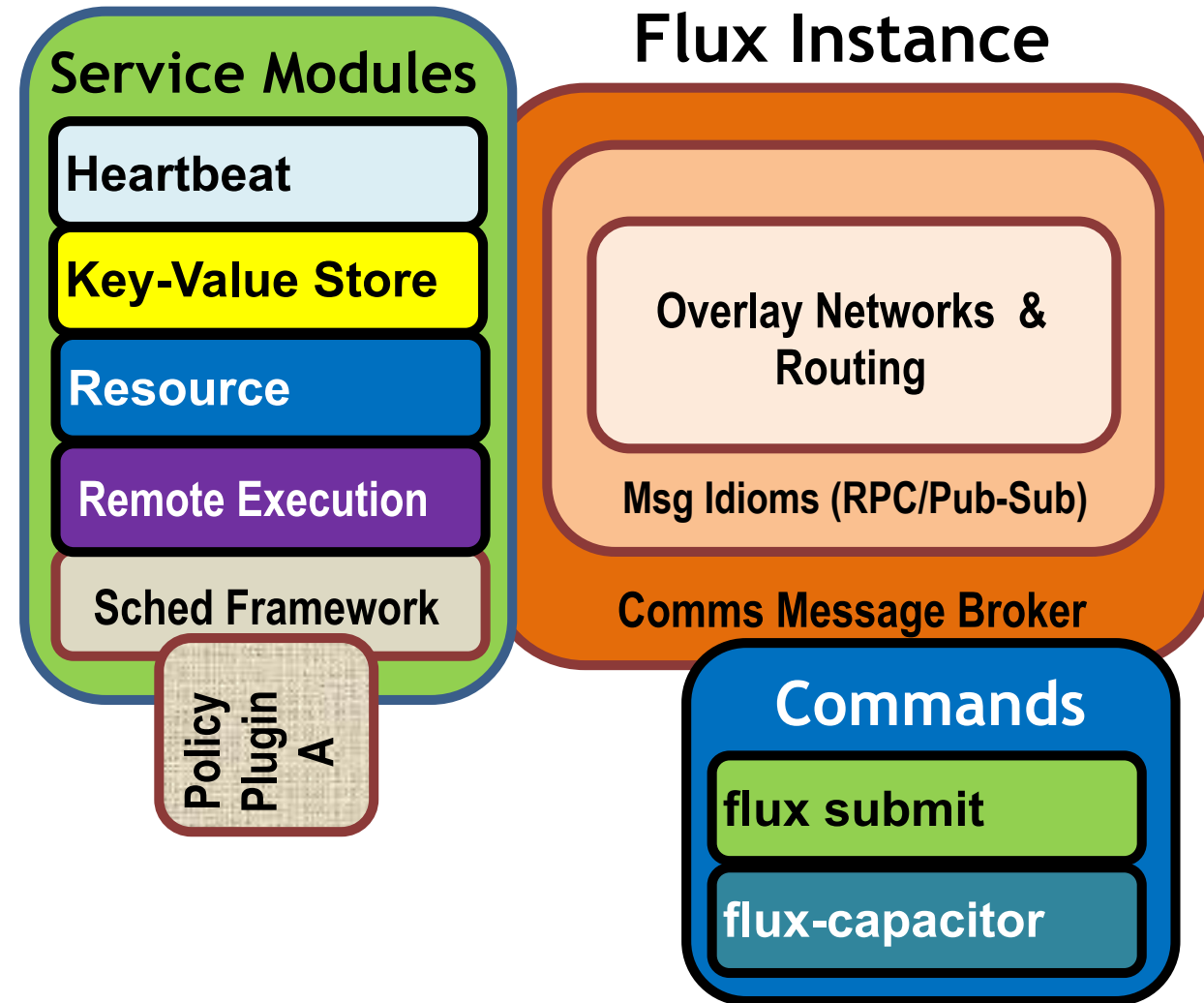
# Extensibility: Modular Design

- At the core of Flux is an overlay network
  - Built on top of ZeroMQ
  - Supports RPCs, Pub/Sub, Push/Pull, etc
- Modules provide extended functionality (i.e., services)
  - User-built modules are loadable too
  - Some modules also support plugins



# Extensibility: Modular Design

- At the core of Flux is an overlay network
  - Built on top of ZeroMQ
  - Supports RPCs, Pub/Sub, Push/Pull, etc
- Modules provide extended functionality (i.e., services)
  - User-built modules are loadable too
  - Some modules also support plugins
- External tools and commands can access services
  - User authentication and roles supported



# Extensibility: Creating Your Own Module

---

# Extensibility: Creating Your Own Module

---

- Register a new service “pymod.new\_job” that ingests jobs and responds with a Job ID

# Extensibility: Creating Your Own Module

- Register a new service “pymod.new\_job” that ingests jobs and responds with a Job ID

```
import itertools, json, flux

def handle_new_job(f, typemask, message, arg):
 job_queue, job_ids = arg
 job_queue.append(message.payload)
 response = {'jobid' : job_ids.next()}
 f.respond(message, 0, json.dumps(response))

def mod_main(f, *argv):
 f.msg_watcher_create(flux.FLUX_MSGTYPE_REQUEST,
 handle_new_job, "pymod.new_job",
 args=([], itertools.count(0))).start()

 f.reactor_run(f.get_reactor(), 0)
```



# Extensibility: Creating Your Own Module

- Register a new service “pymod.new\_job” that ingests jobs and responds with a Job ID

```
import itertools, json, flux

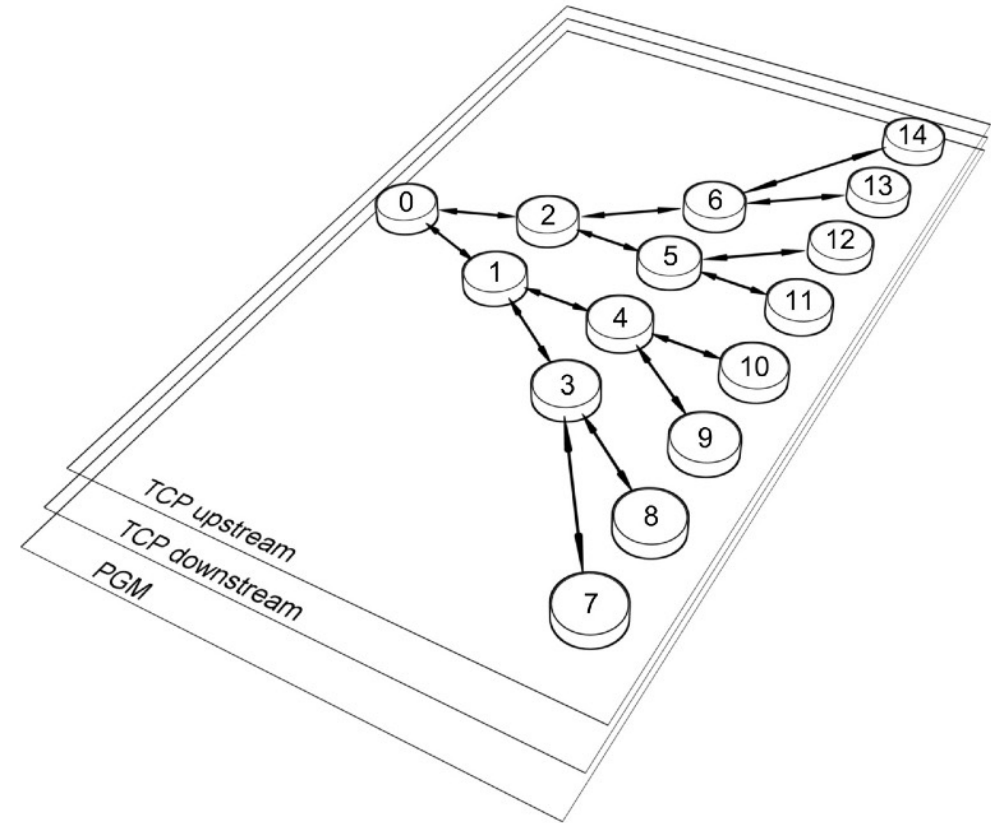
def handle_new_job(f, typemask, message, arg):
 job_queue, job_ids = arg
 job_queue.append(message.payload)
 response = {'jobid' : job_ids.next()}
 f.respond(message, 0, json.dumps(response))

def mod_main(f, *argv):
 f.msg_watcher_create(flux.FLUX_MSGTYPE_REQUEST,
 handle_new_job, "pymod.new_job",
 args=([], itertools.count(0))).start()

 f.reactor_run(f.get_reactor(), 0)
```

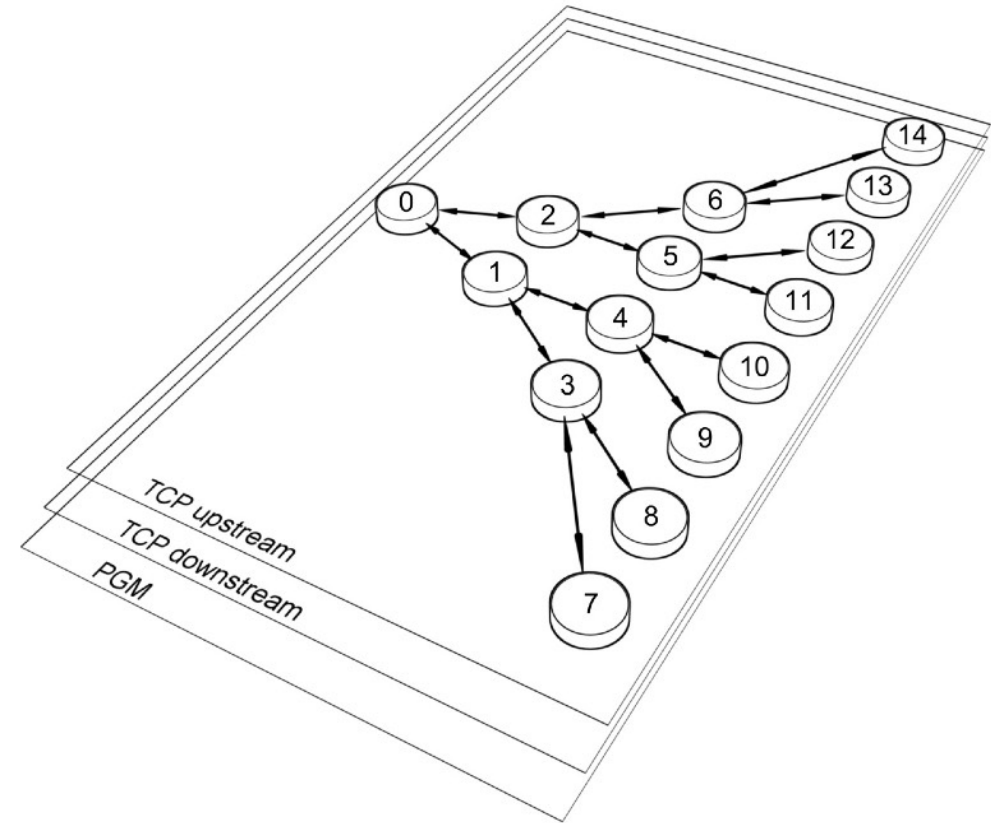
- Load using flux module `load pymod --module=path/to/file.py`

# Extensibility: Flux's Communication Overlay



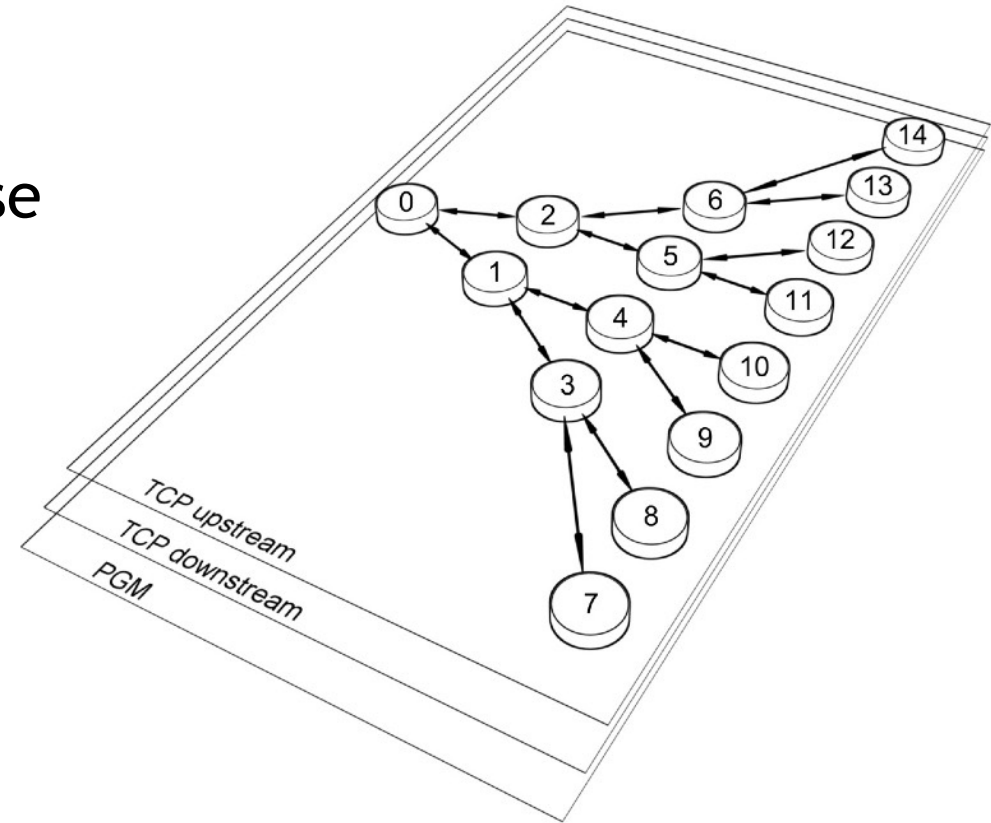
# Extensibility: Flux's Communication Overlay

- Connect to a running flux instance
  - `f = flux.Flux()`



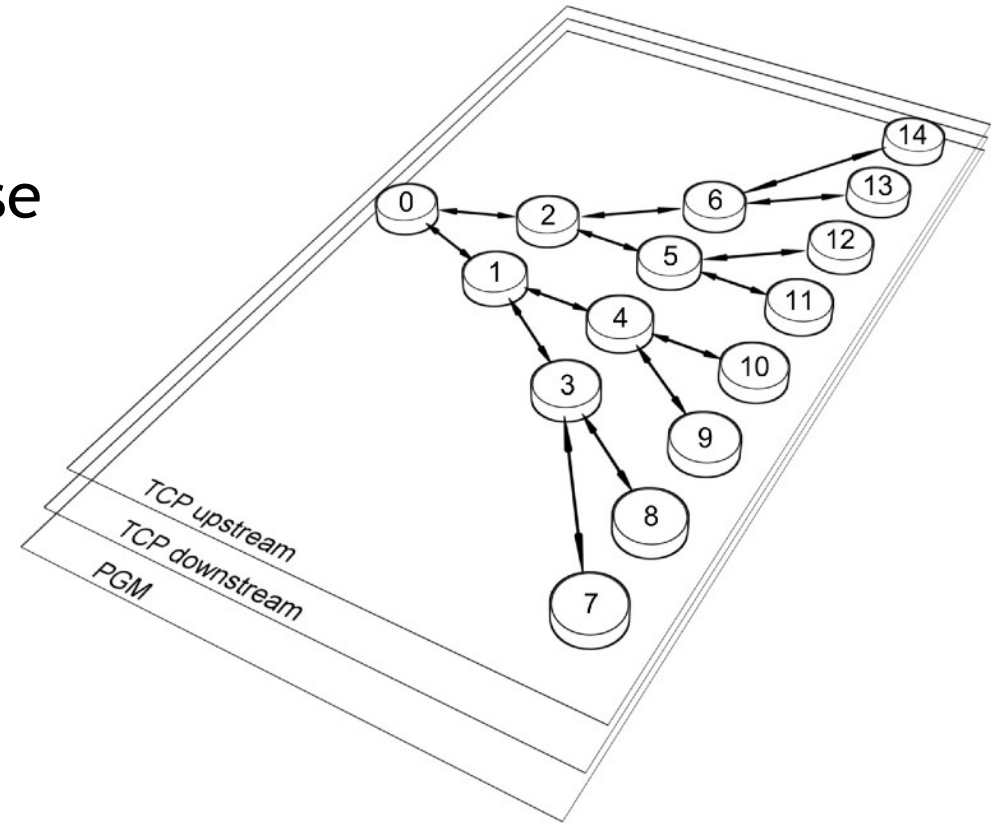
# Extensibility: Flux's Communication Overlay

- Connect to a running flux instance
  - `f = flux.Flux()`
- Send an RPC to a service and receive a response
  - `resp = f.rpc_send("pymod.new_job", payload)`  
`jobid = json.loads(resp)['jobid']`



# Extensibility: Flux's Communication Overlay

- Connect to a running flux instance
  - `f = flux.Flux()`
- Send an RPC to a service and receive a response
  - `resp = f.rpc_send("pymod.new_job", payload)`  
`jobid = json.loads(resp)['jobid']`
- Subscribe to and publish an event
  - `f.event_subscribe("node_down")`  
`f.msg_watcher_create(node_down_cb, raw.FLUX_MSGTYPE_EVENT, "node_down").start()`
  - `f.event_send("node_down")`



# Extensibility: Scheduler Plugins

---

# Extensibility: Scheduler Plugins

---

- Common, built-in scheduler plugins:
  - First-come First-Served (FCFS)
  - Backfilling
    - Conservative
    - EASY
    - Hybrid

# Extensibility: Scheduler Plugins

- Common, built-in scheduler plugins:
  - First-come First-Served (FCFS)
  - Backfilling
    - Conservative
    - EASY
    - Hybrid
- Various, advanced scheduler plugins:
  - I/O-aware
  - CPU performance variability aware
  - Network-aware



# Extensibility: Scheduler Plugins

- Common, built-in scheduler plugins:
  - First-come First-Served (FCFS)
  - Backfilling
    - Conservative
    - EASY
    - Hybrid
- Various, advanced scheduler plugins:
  - I/O-aware
  - CPU performance variability aware
  - Network-aware
- Create your own!

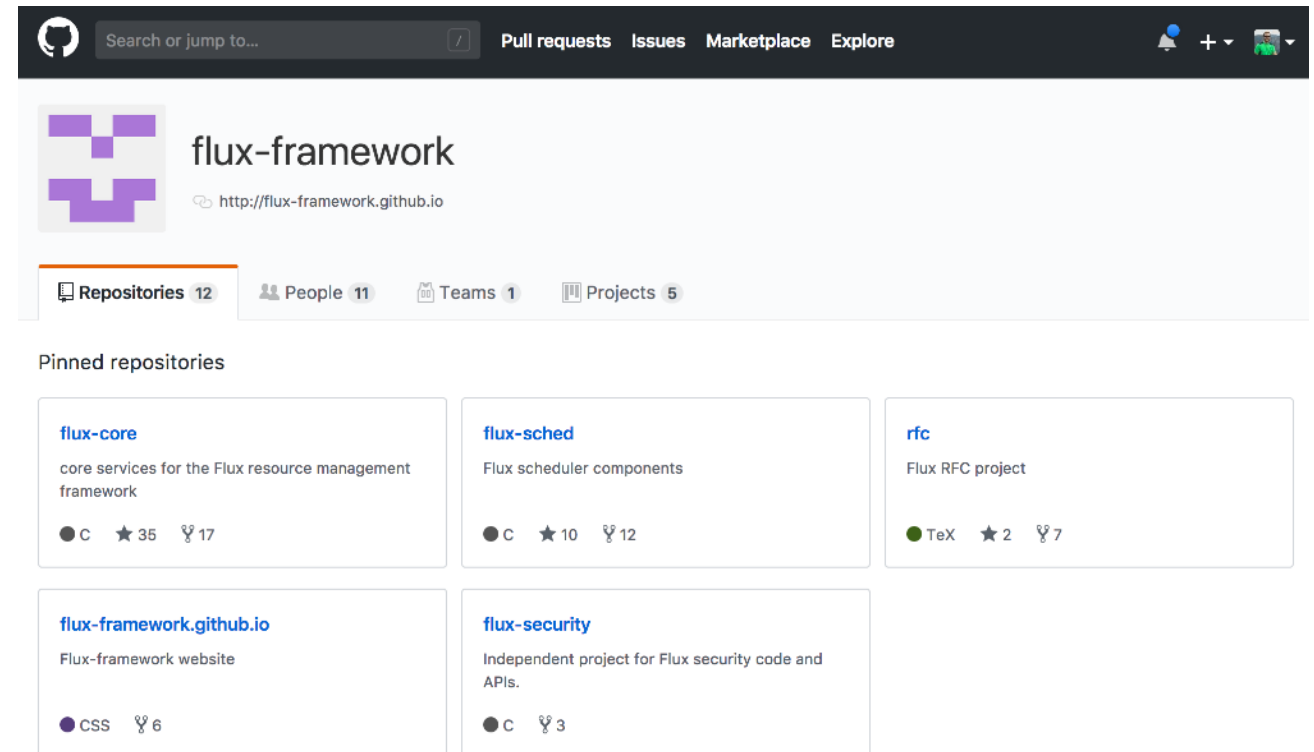
# Extensibility: Scheduler Plugins

- Common, built-in scheduler plugins:
  - First-come First-Served (FCFS)
  - Backfilling
    - Conservative
    - EASY
    - Hybrid
- Various, advanced scheduler plugins:
  - I/O-aware
  - CPU performance variability aware
  - Network-aware
- Create your own!
- Loading the plugins
  - `flux module load sched.io-aware`
  - `FLUX_SCHED_OPTS="plugin=sched.fcfs" flux start`

# Extensibility: Open Source

- Flux-Framework code is available on GitHub
- Most project discussions happen in GitHub issues
- PRs and collaboration welcome!

# GitHub



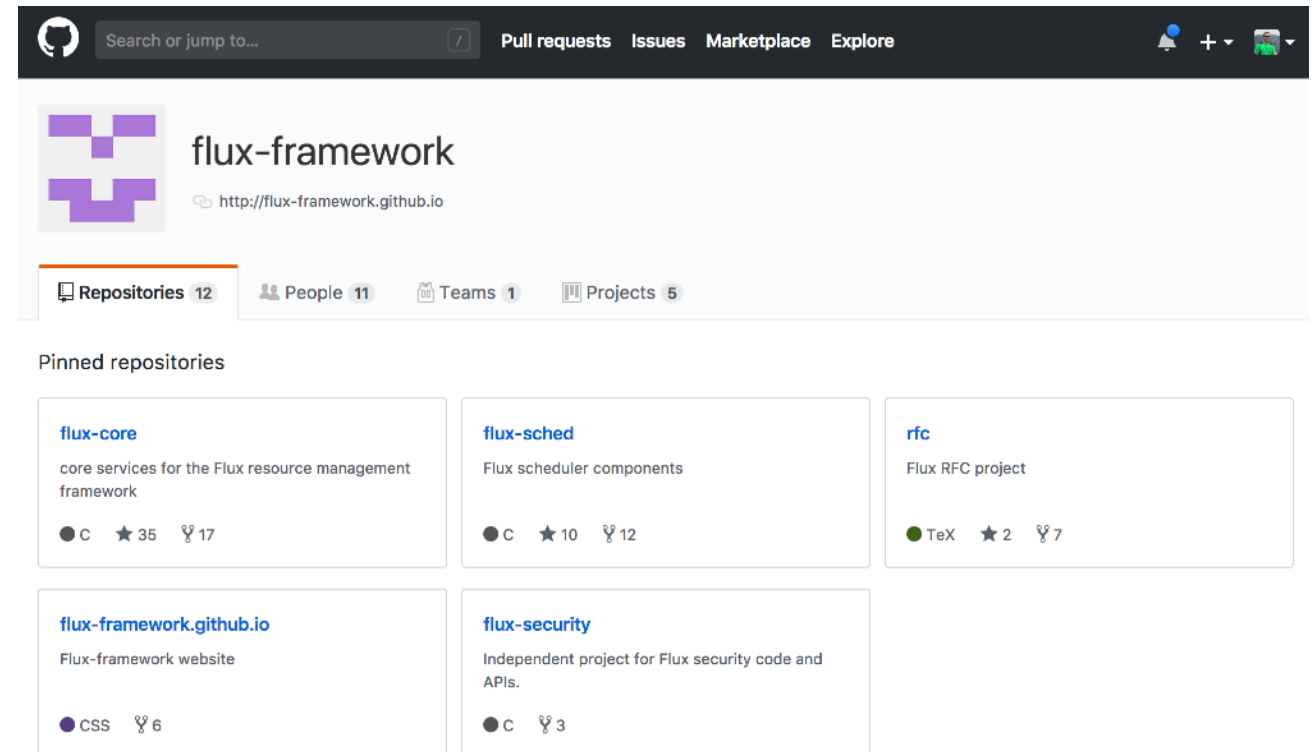
The screenshot shows the GitHub profile page for the 'flux-framework' organization. At the top, there is a search bar and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The profile header includes the organization's name 'flux-framework' and its website 'http://flux-framework.github.io'. Below the header, there are tabs for 'Repositories 12', 'People 11', 'Teams 1', and 'Projects 5'. The 'Pinned repositories' section displays five repositories:

- flux-core**: core services for the Flux resource management framework. 35 stars, 17 forks.
- flux-sched**: Flux scheduler components. 10 stars, 12 forks.
- rfc**: Flux RFC project. 2 stars, 7 forks.
- flux-framework.github.io**: Flux-framework website. 6 forks.
- flux-security**: Independent project for Flux security code and APIs. 3 forks.

# Extensibility: Open Source

- Flux-Framework code is available on GitHub
- Most project discussions happen in GitHub issues
- PRs and collaboration welcome!

# GitHub



The screenshot shows the GitHub profile page for the 'flux-framework' organization. At the top, there is a search bar and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The profile header includes the organization's name 'flux-framework' and its website 'http://flux-framework.github.io'. Below this, there are tabs for 'Repositories 12', 'People 11', 'Teams 1', and 'Projects 5'. The 'Pinned repositories' section displays five repositories:

- flux-core**: core services for the Flux resource management framework. 35 stars, 17 forks.
- flux-sched**: Flux scheduler components. 10 stars, 12 forks.
- flux-framework.github.io**: Flux-framework website. 6 forks.
- flux-security**: Independent project for Flux security code and APIs. 3 forks.
- rfc**: Flux RFC project. 2 stars, 7 forks.

# Thank You!



#### Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.