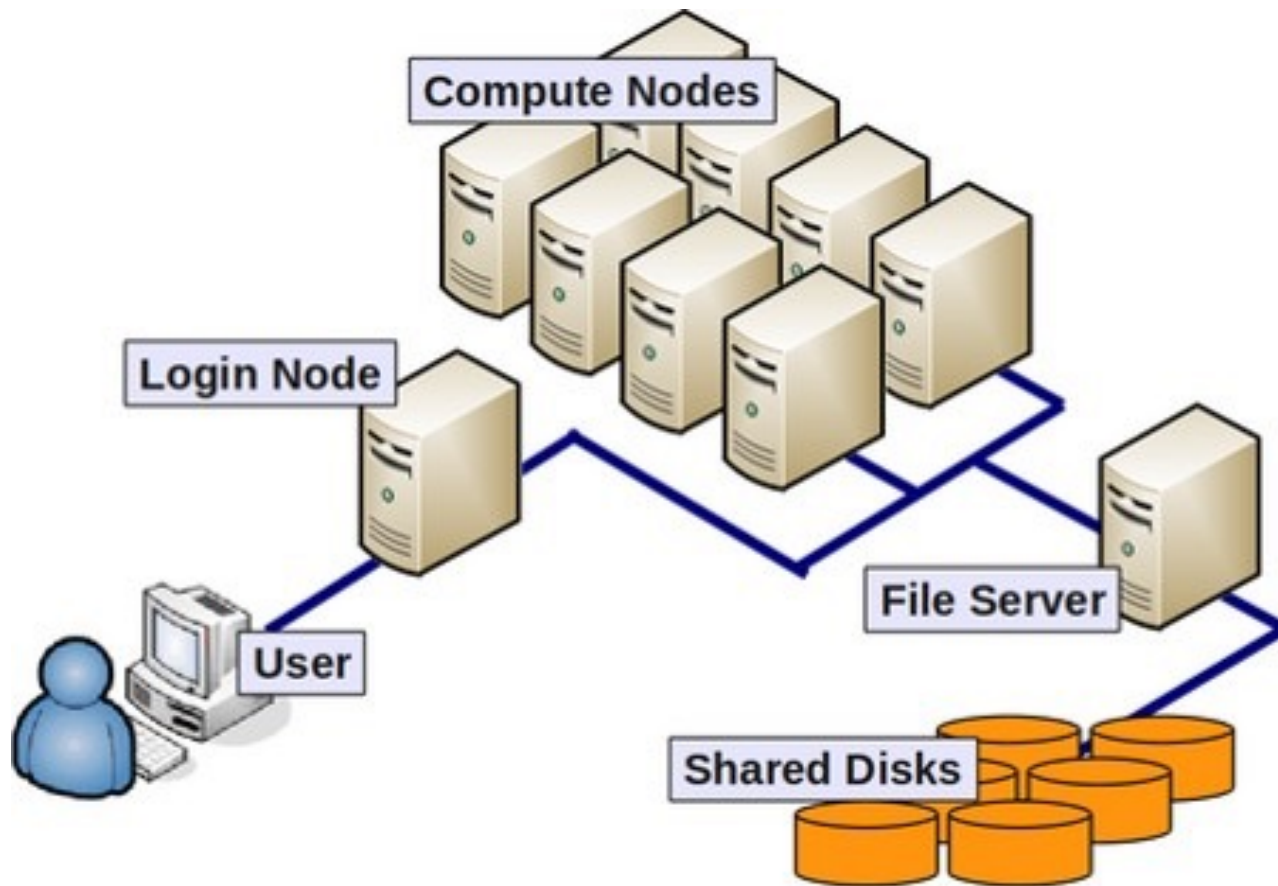


Dynamically optimize your jobs' RAM and run time requests with SmartSlurm

Lingsheng Dong and Kathleen Chappell
Research Computing Consultant at HMS RC



Generic Cluster Architecture



What is Slurm scheduler

- Allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work.
- Provides a framework for starting, executing, and monitoring work (normally parallel jobs) on the set of allocated nodes.
- Arbitrates contention for resources by managing a queue of pending work.

-- from: <https://slurm.schedmd.com/overview.html>

Slurm scheduler

- Sbatch command to submit jobs:

```
sbatch --mem 2G -t 2:0:0 cmd
```

- Job dependency:

```
sbatch --mem 2G -t 2:0:0 \  
-d afterok:123:321 cmd
```

HMS RC O2 Cluster

- O2 runs wildly diverse software and each software has unique resource requirement
- Same software run on different data set can have different specs
- The software x reqs x dataset combination space is huge. Impossible to efficiently assign static resources
- Number of jobs also very large, not possible to manually manage

Issues to resolve

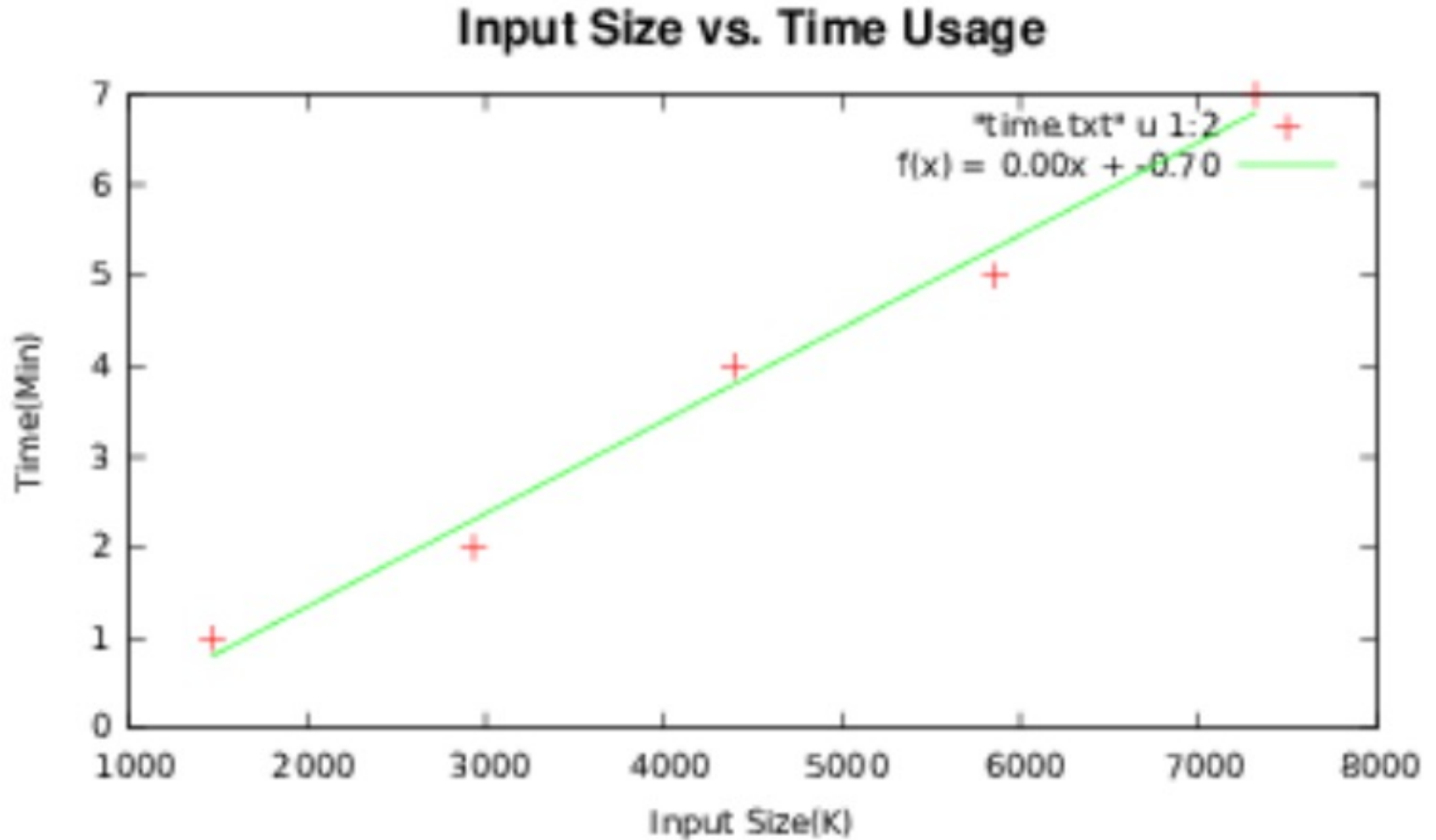
Jobs reserve excessive resources

- Jobs succeed
- Wasted resources unavailable to other users

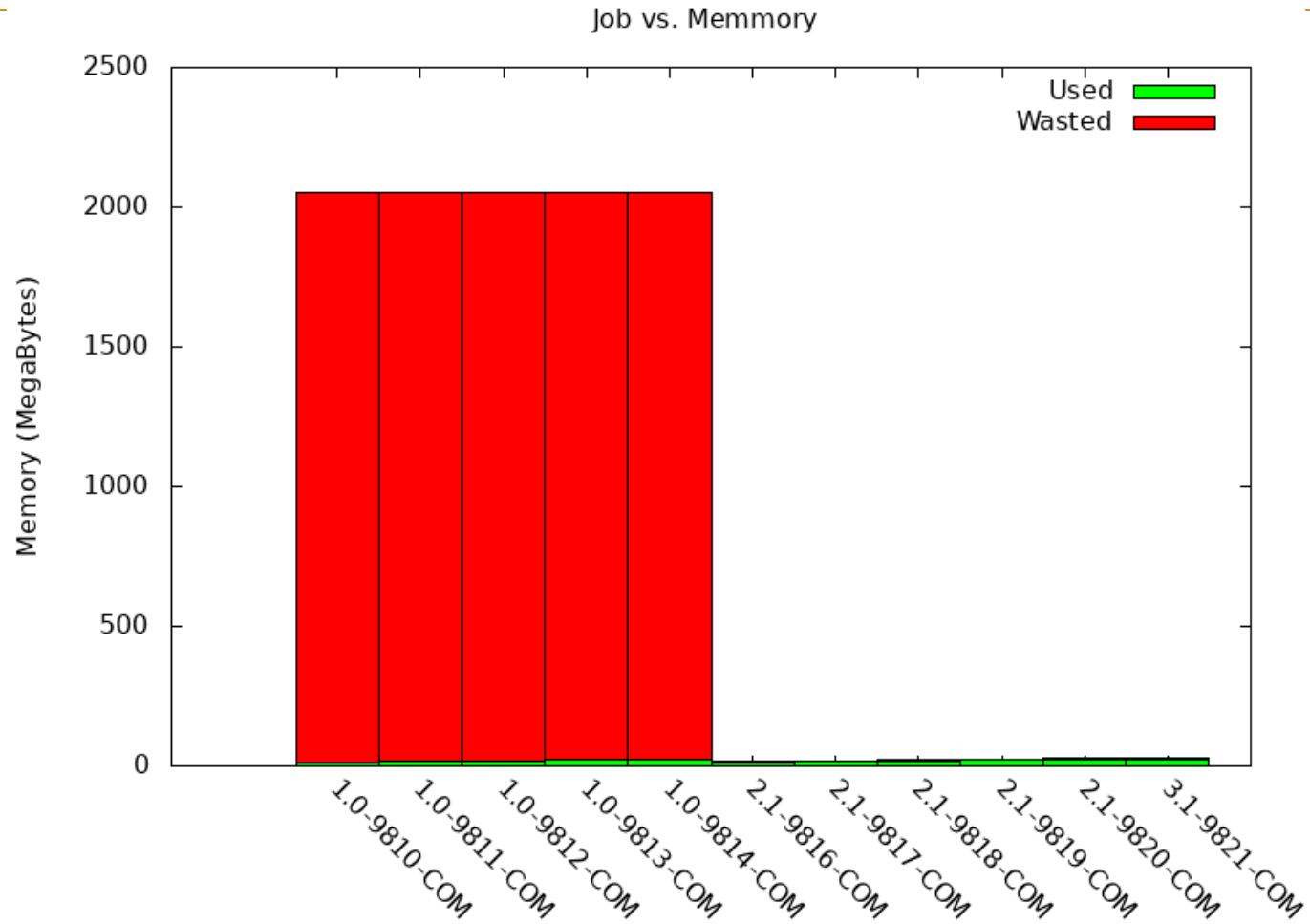
Jobs reserve insufficient resources

- Jobs fail
- Jobs must be manually rerun with more resources
- Wastes user time. Delays results

Ideal solution



Ideal solution



Smart Slurm

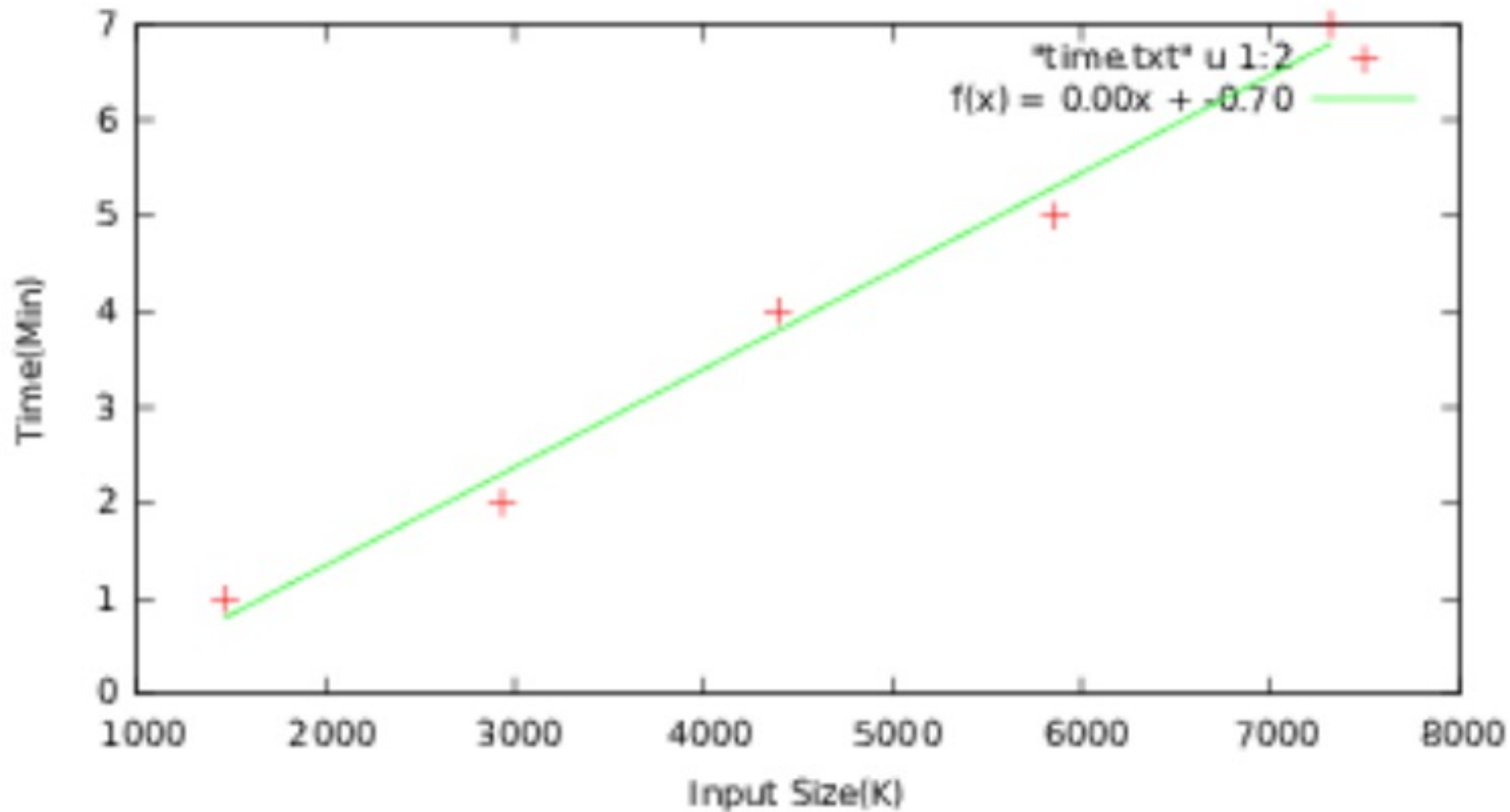
- `ssbatch`, which is a `sbatch` wrapper:
 - Check job was finished successfully or not
 - Estimate memory RAM and time based on several factors (i.e., program type, input size, previous job records)
 - Submit job
 - Monitor and log memory and time usage by the job
 - Re-run OOT/OOM job and send detailed email notifications
- `runAsPipeline`: It parses special bash script to find user defined commands and call `ssbatch` to submit jobs to Slurm. It take care of job dependency.

ssbatch

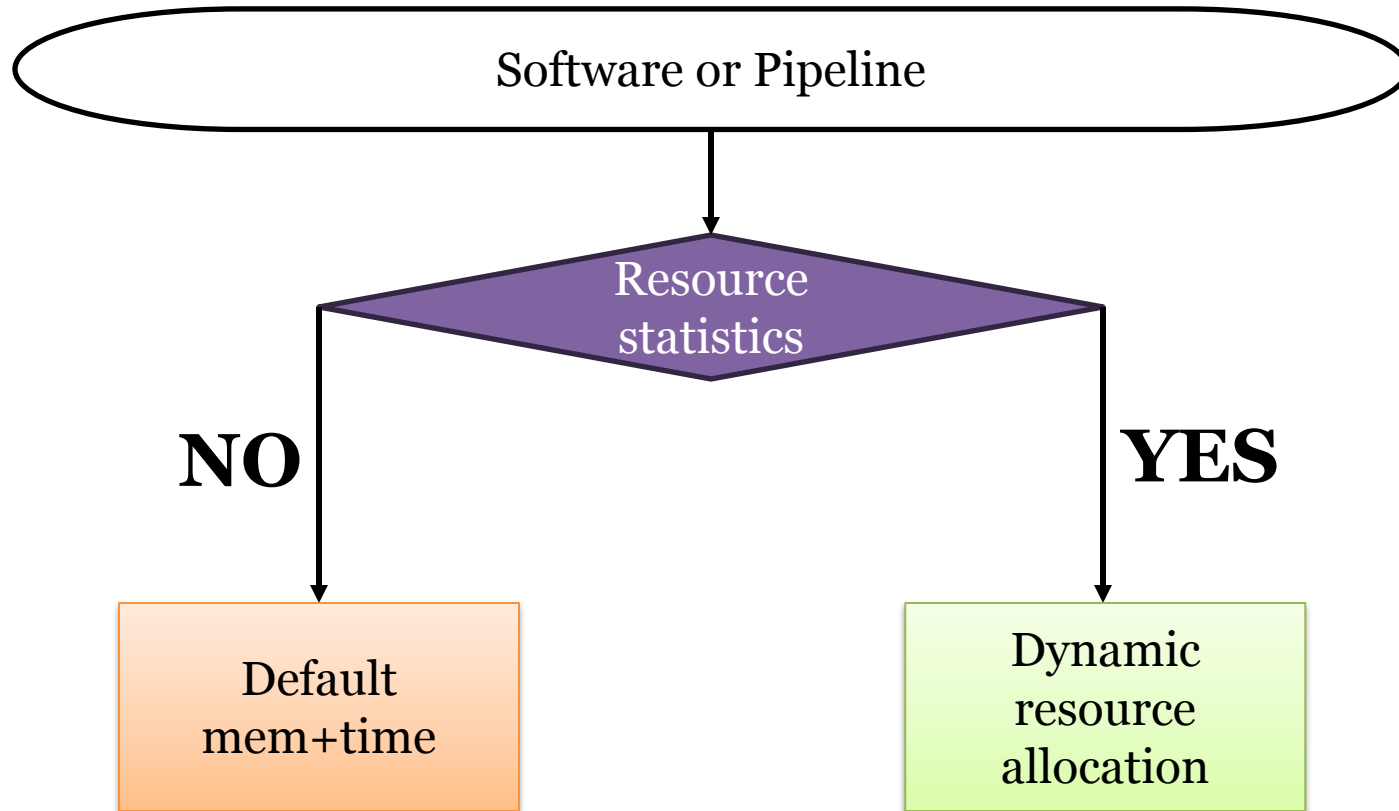
```
$ ssbatch \
-P program \
-R reference \
-I input.txt \
-F uniqueJobFlag \
--mem 2G -t 2:0:0 -p short cmd
```



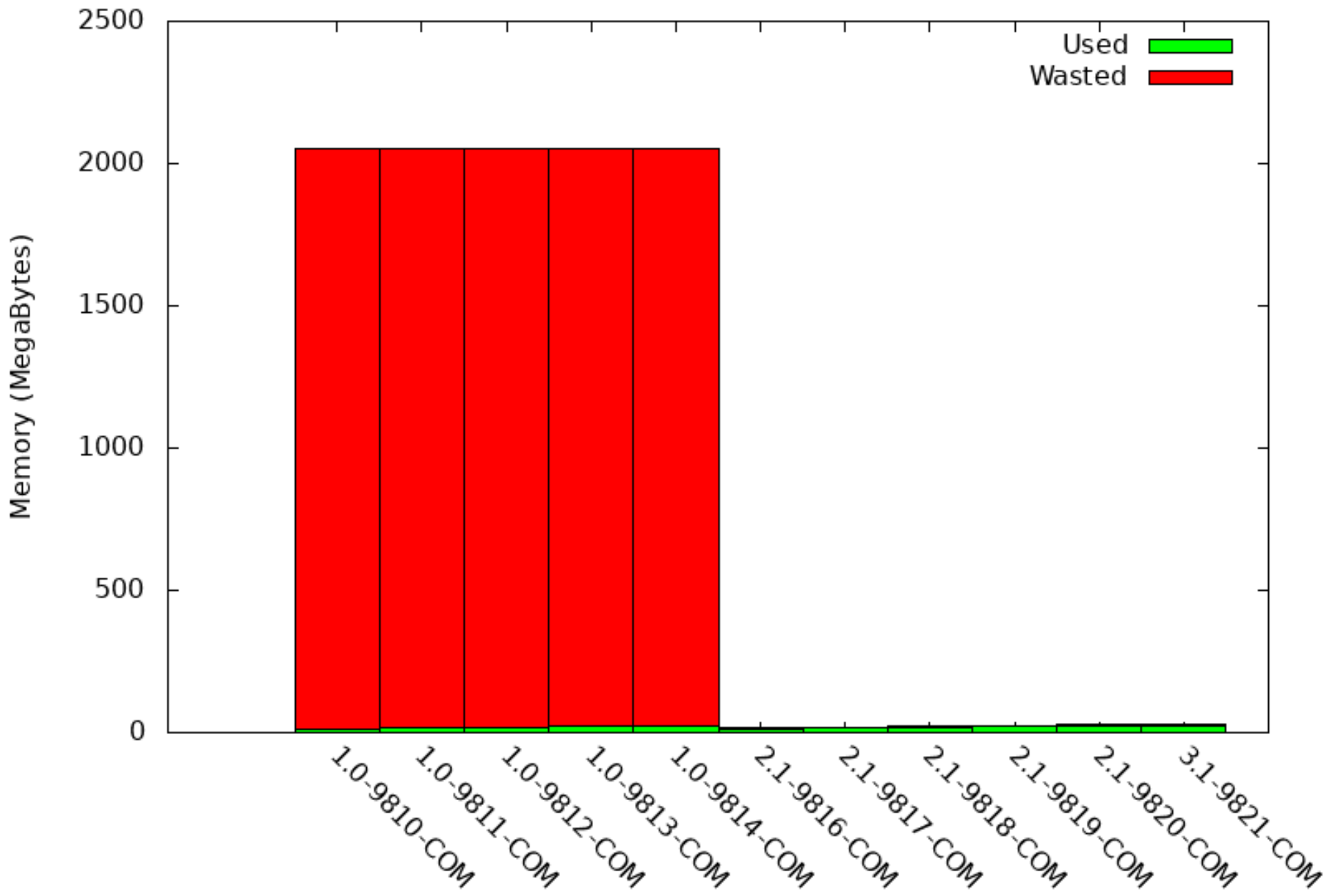
Input Size vs. Time Usage



ssbatch logic



Job vs. Memmory



Inside each job

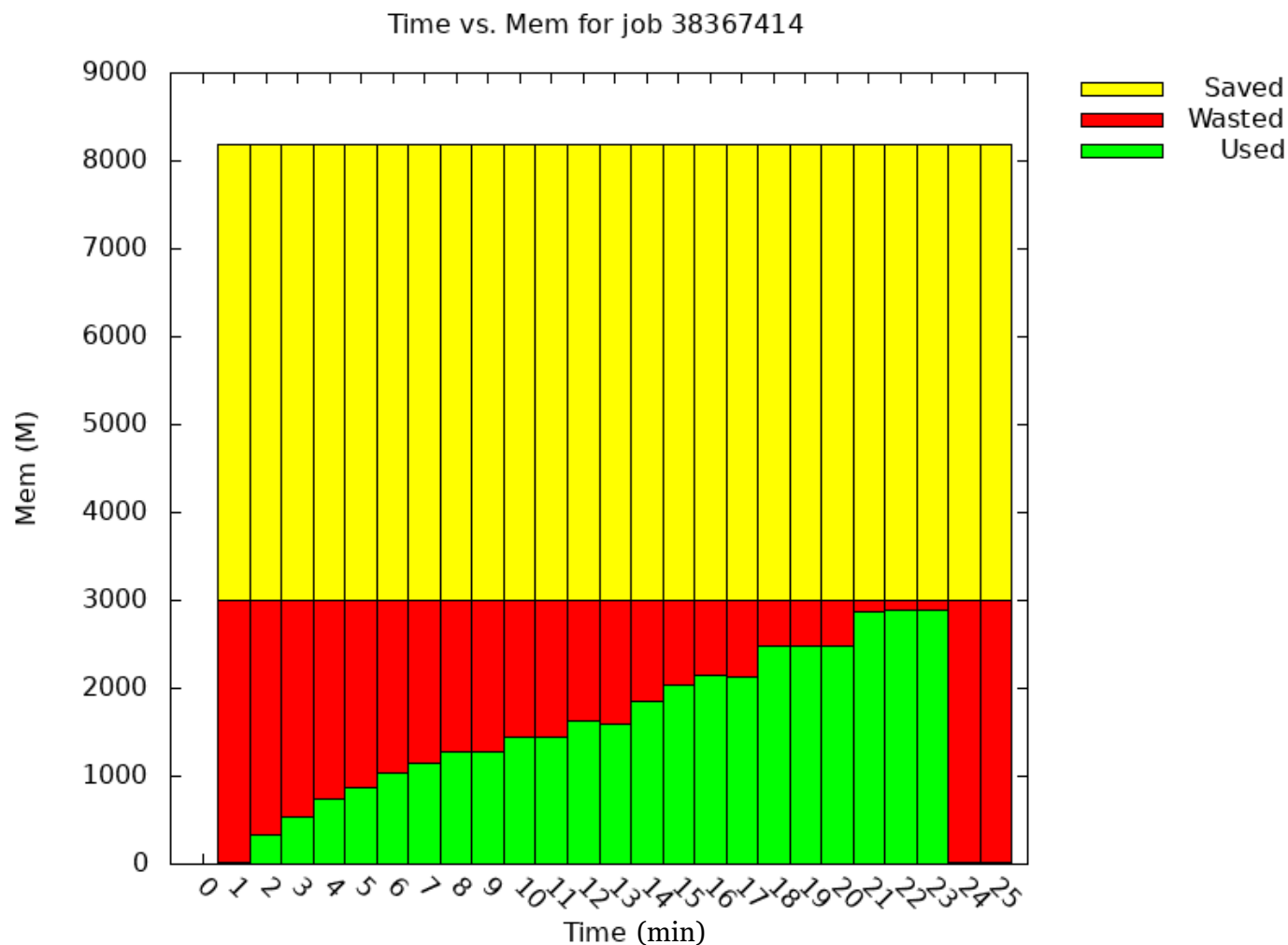
- Monitor real-time mem/CPU usage
- When job finishes, send email notification
- If job succeeds, keep a log of max mem and time
- Otherwise
 1. OOM ==> re-submit with more memory
 2. OOT ==> re-submit with more time
 3. Fail: save error message to output file
- Send user notification email
- Manage pending/downstream jobs

Scripts

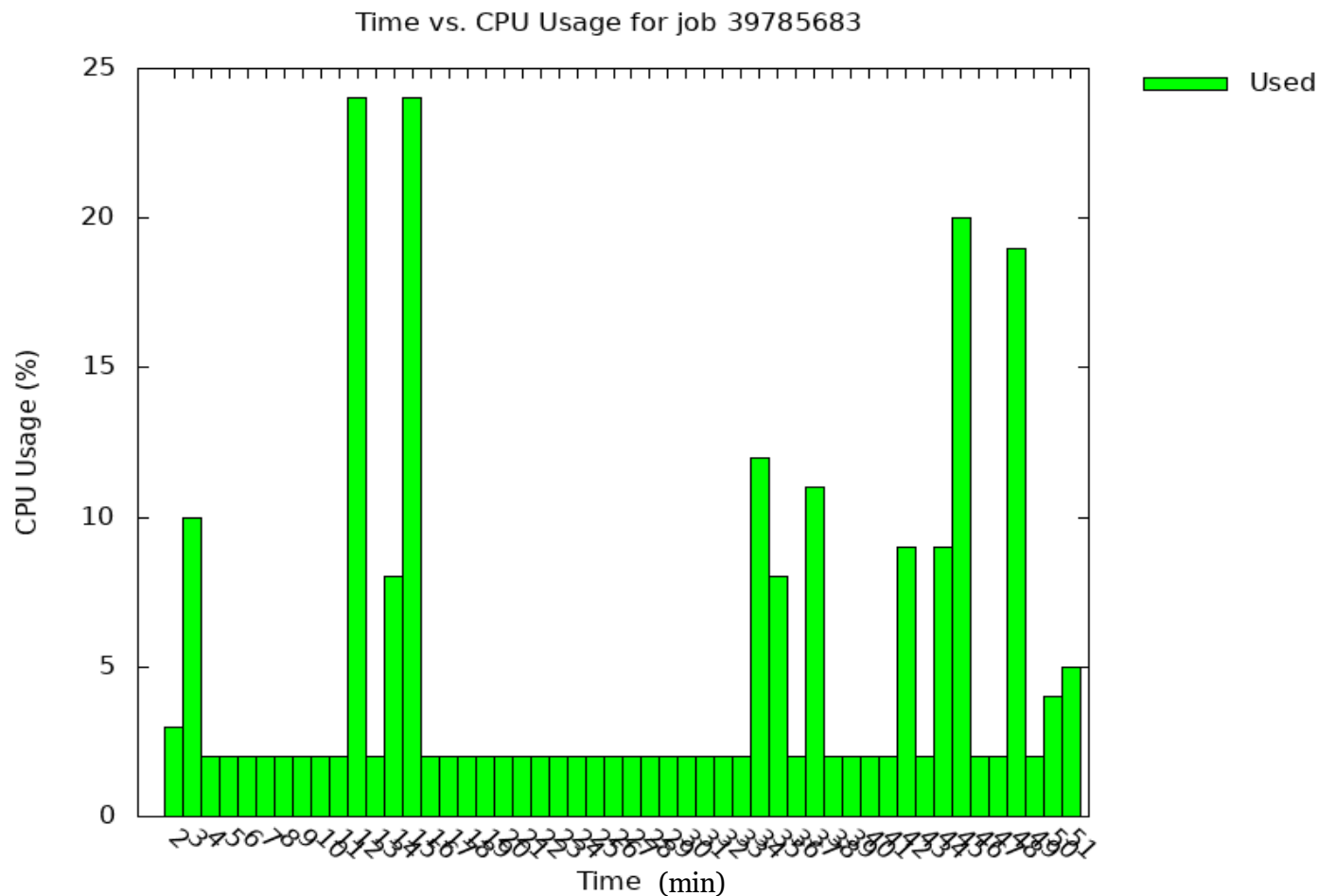
```
jid=$(checkIfItIsDone.sh && \
      estimateMemTime.sh && \
      sbatch -t $mT --mem $mMem \
        -o jobx.out -e jobx.err \
        --wrap "monitorResource.sh & \
        someCmd && touch jobx.success; \
        cleanUp.sh;")
```

What the red scripts do?

Memory usage report



CPU usage report



Smart Slurm

- `ssbatch`, which is an `sbatch` wrapper:
 - Check job was finished successfully or not
 - Estimate memory RAM and time based on several factors (i.e., program type, input size, previous job records)
 - Submit job
 - Monitor and log memory and time usage by the job
 - Re-run OOT/OOM job and send detailed email notifications
- `runAsPipeline`: It parses special bash script to find user defined commands and call `ssbatch` to submit jobs to Slurm. It take care of job dependency.

A simple pipeline in bash

```
$ cat bashScript.sh
```

```
for i in {1..5}; do
```

```
    input=numbers$i.txt
```

```
    grep 123 $input > number.$i.txt
```

```
Done
```

```
cat number.*.txt > all.txt
```

```
$ sbatch -mem 25G -t 20-0:0 bashScript.sh
```

Converting to Slurm pipeline

```
for i in {1..5}; do
    input=numbers$i.txt

    id=$(sbatch -p short -t 5 --wrap "grep 123 \
        $input > number$i.txt")
done

sbatch -p short -t 5 -d afterok:$id --wrap "cat \
    number.*.txt > all.txt"
```

Converting to Slurm pipeline

```
deps=afterok
for i in {1..5}; do
    input=numbers$i.txt

    id=$(sbatch -p short -t 5 --wrap "grep 123 \
        $input > number$i.txt")
    deps=$deps:$id
done

sbatch -p short -t 5 -d $deps --wrap "cat \
    number.*.txt > all.txt"
```

Can we automate it?

```
for i in {1..5}; do  
    input=numbers$i.txt
```

```
#@ Please submit as job with 1G memory,  
#@ with 5 minute and 1 CPU without pending.  
grep 123 $input > number.$i.txt
```

Done

```
#@ Please submit as job with 2G memory,  
#@ with 5 minute and 1 CPU and wait for the other 5 jobs  
cat number.*.txt > all.txt
```

Smart Slurm decorator syntax

```
#@1,0,find,none,input,sbatch -t 5 --mem 1G  
find.sh 123 $input > number.$i.txt
```

#@ → This row is a decorator for runAsPipeline to parse
1 → Step ID

0 → The step IDs this step depends on
find → Step name.

None → Reference

Input. → input for the step.

sbatch → default sbatch command to use

Can we automate it?

```
$ cat specialBashScript.sh
```

```
for i in {1..5}; do  
    input=numbers$i.txt
```

```
#@1,0,find,,input,sbatch -t 5 --mem 1G  
grep 123 $input > number.$i.txt
```

Done

```
#@2,1,merge  
cat number.*.txt > all.txt
```


runAsPipeline

```
$ runAsPipeline \
specialBashScript.sh \
"sbatch --mem 2G -t 2:0:0" \
useTmp/noTmp [run]
```

Other tools

\$ **checkRun**

Available log folders or log file:

1 folder: log

2 file: .smartSlurm.log

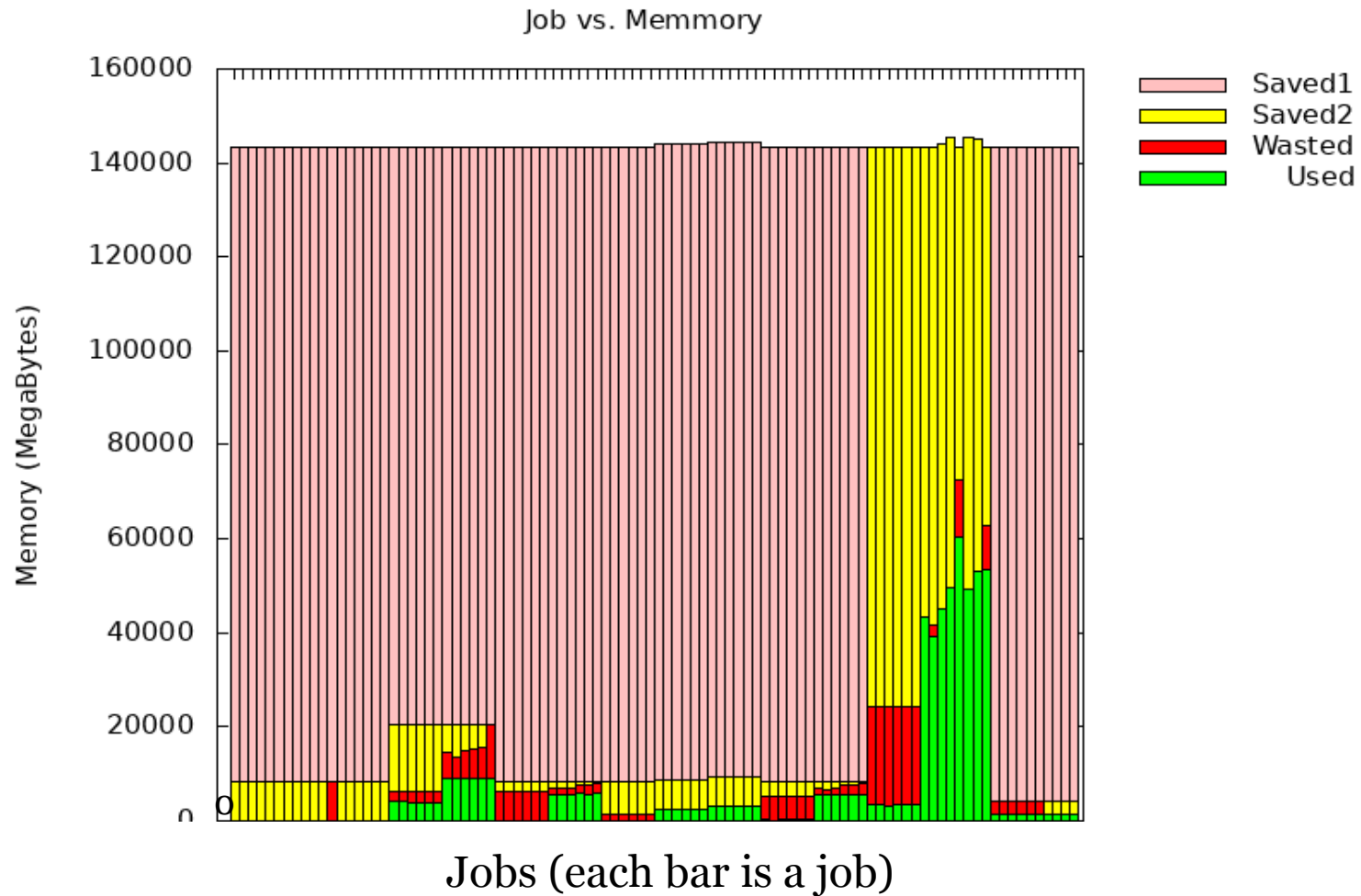
3 folder: log.2024-05-30.10-06-18

4 file: .smartSlurm.log.2024-05-30.10-06-15

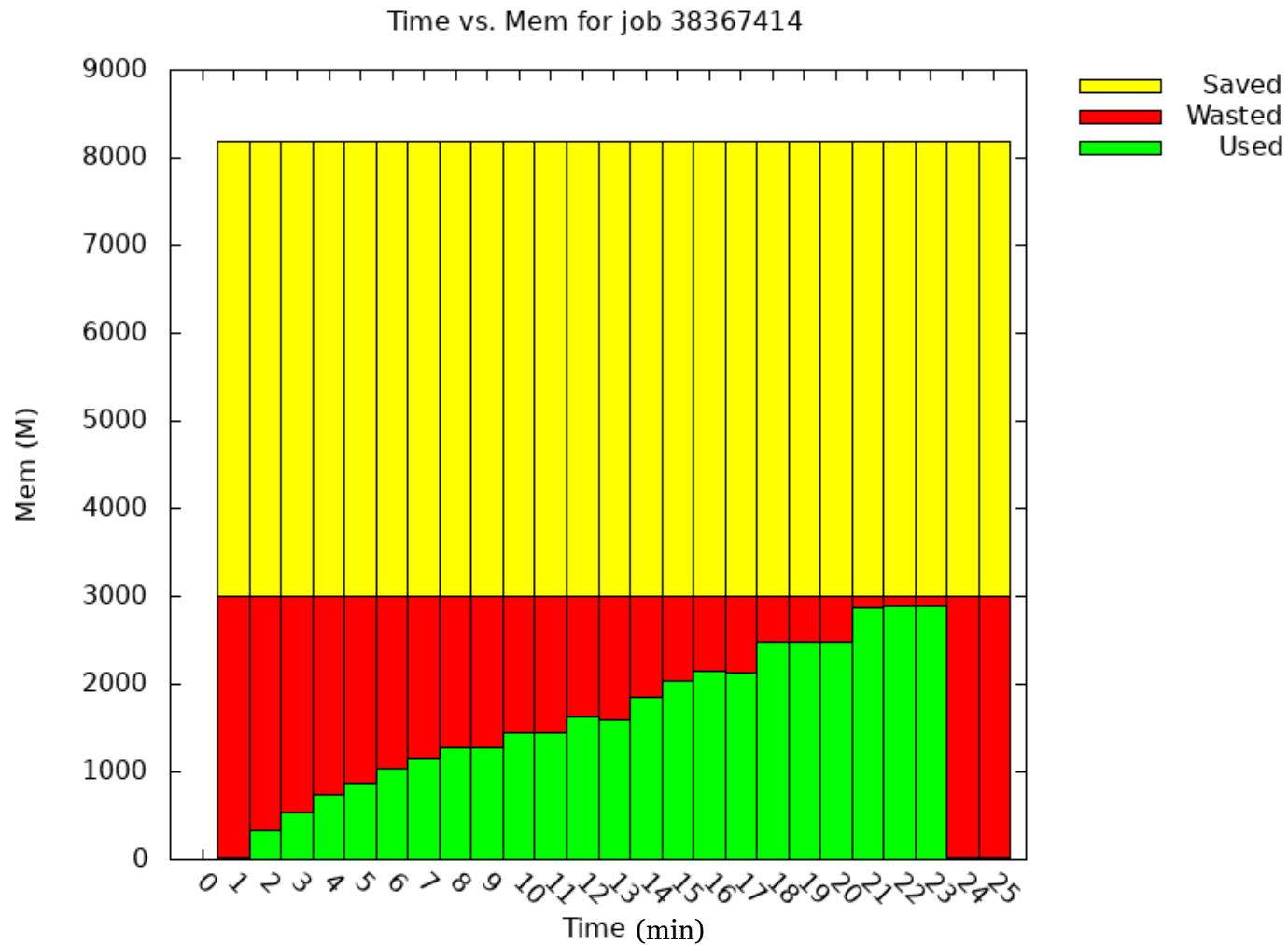
Please select the log or folder you want to
check or type q to quit:

\$ **cancelAllJobs**

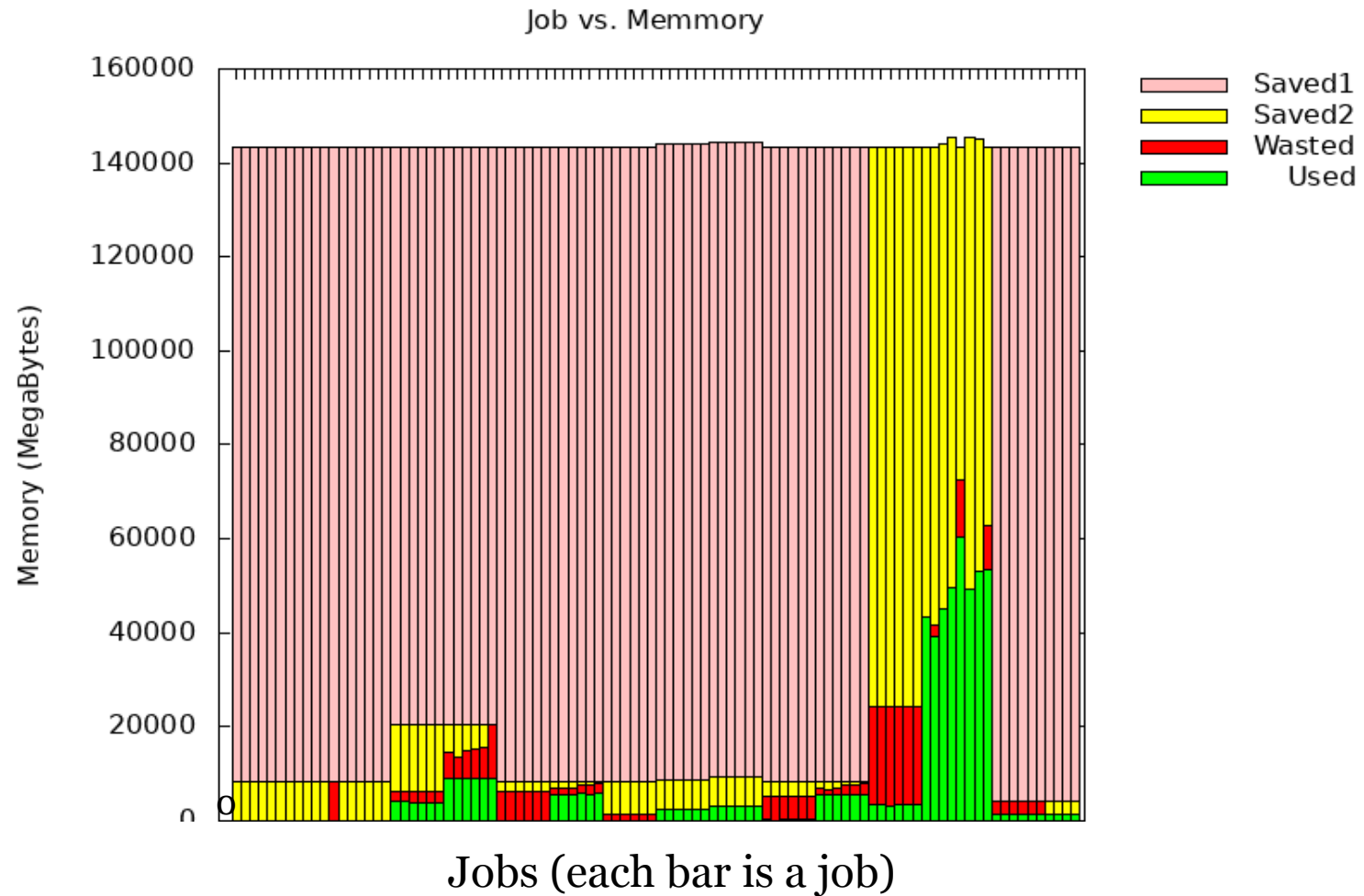
Case study(Nascent Transcriptomics Core @ HMS)



Future 1: Breakpoint support



Future 2: Auto split workflow



How do I access Smart Slurm?

- <https://github.com/ld32/SmartSlurm>
- Git clone <https://github.com/ld32/smartSlurm>
- `export PATH=$PWD/SmartSlurm/bin:$PATH`

Solution 1: Cromwell using 'CWL' file

```
workflow myWorkflowName {
```

```
File my_ref  
File my_input  
String name
```

```
call task_A {
```

```
input: ref= my_ref, in= my_input, id= name
```

```
}
```

```
call task_B {
```

```
input: ref= my_ref, in= task_A.out
```

```
}
```

```
}
```

```
task task_A {
```

```
...
```

```
}
```

```
task task_B {
```

```
...
```

```
}
```

```
task task_A {
```

```
File ref  
File in  
String id
```

```
command {
```

```
do_stuff -R ${ref} -I ${in} -O ${id}.ext
```

```
}
```

```
runtime {
```

```
docker: "my_project/do_stuff:1.2.0"
```

```
}
```

```
output {
```

```
File out= "${id}.ext"
```

```
}
```

```
}
```

<https://bluewaters.ncsa.illinois.edu/liferay-content/document-library/content/Ruchi%20Munshi.%20Blue%20Waters%20webinar.%20Cromwell%20and%20WDL.pdf>

Solution 2: BCBio using 'yaml/CWL' file

```
upload:
  dir: ../final
details:
  - files: [../input/NA12878-NGv3-LAB1360-A_1.fastq.gz, ..
    description: NA12878
    metadata:
      sex: female
    analysis: variant2
    genome_build: hg38
    algorithm:
      aligner: bwa
      variantcaller: gatk-haplotype
      validate: giab-NA12878/truth_small_variants.vcf.gz
      validate_regions: giab-NA12878/truth_regions.bed
      variant_regions: capture_regions/Exome-NGv3
```

`bcbio_nextgen.py config/NA12878-exome-methodcmp.yaml -n 8`

<https://bcbio-nextgen.readthedocs.io/en/latest/contents/intro.html#workflow>

Solution 3: NextFlow script

```
#!/usr/bin/env nextflow

params.in = "$baseDir/data/sample.fa"

/*
 * split a fasta file in multiple files
 */
process splitSequences {

    input:
    path 'input.fa' from params.in

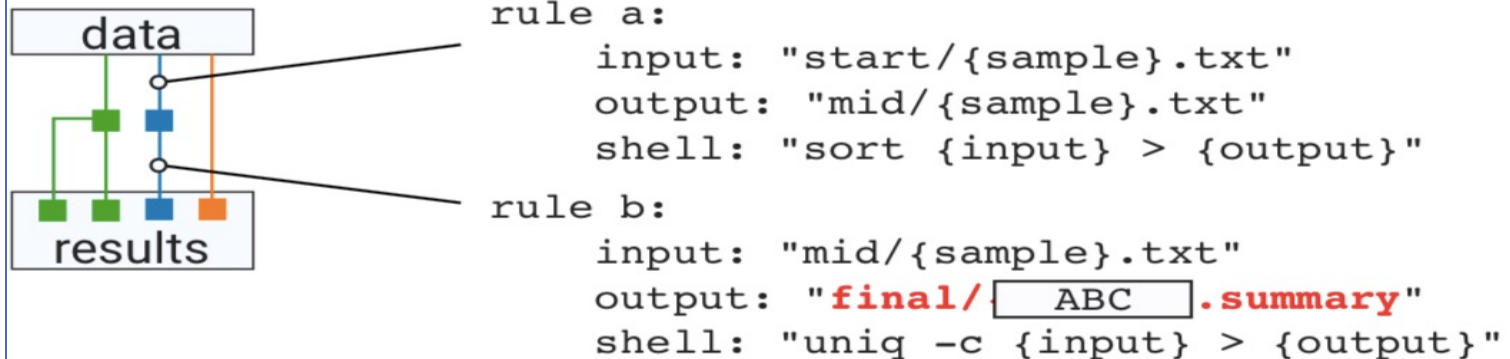
    output:
    path 'seq_*' into records

    """
    awk '/^>/{f="seq_"++d} {print > f}' < input.fa
    """
}
```

<https://www.nextflow.io/example1.html>

Solution 4: Snakemake using makefile

Dependencies are implicit and 'backwards'



```
$ snakemake final/ABC.summary
```

https://hpc.nih.gov/training/handouts/180221_snakemake_class_web.pdf

For more direction

- **Email:** rchelp@hms.harvard.edu
- **O2 wiki:**
<https://harvardmed.atlassian.net/wiki/spaces/O2/overview>
- **Website:** <http://rc.hms.harvard.edu>
- **RC Office Hours:** Wed 1-3p Gordon Hall 500
 - <https://rc.hms.harvard.edu/office-hours/> for Zoom web conferencing during remote work