Self-Checkpointing Jobs with HTCondor

Christina Koch

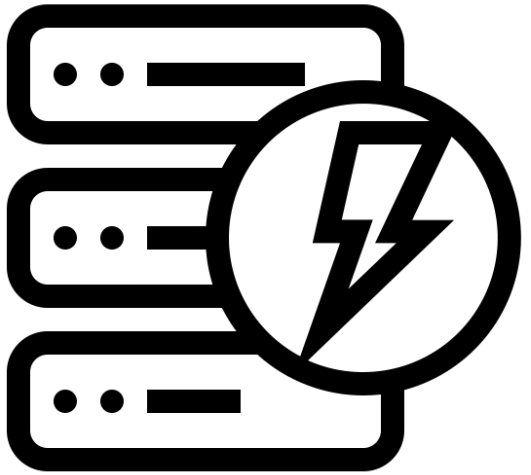
Center for High Throughput Computing

What is Checkpointing?

A program is able to save progress periodically to a file and resume from that saved file to continue running, losing minimal progress.

Why Self-Checkpoint?

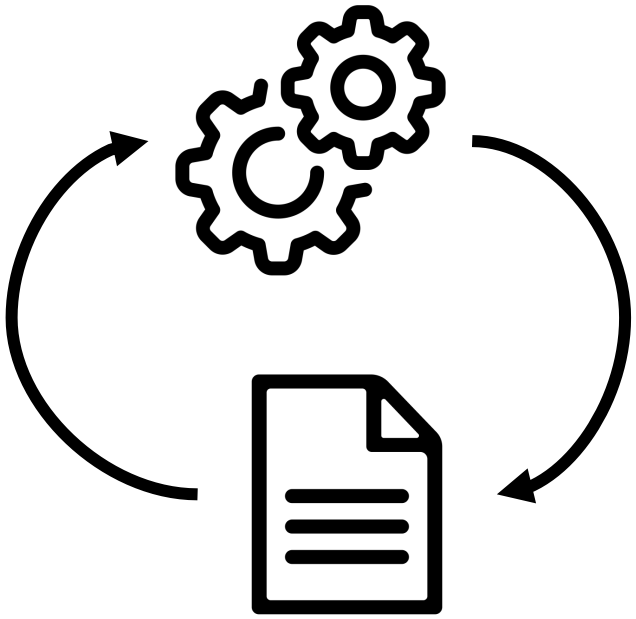
- **Interruptions happen**:
  - Hardware or networking failures
  - Cluster/node policy (jobs can only run for 8 hours before getting killed)
  - Using opportunistic or backfill resources with no runtime guarantee
- Self-checkpointing allows you to make progress through interruptions, **especially for longer-running jobs**.



Lightning by Bernar Novalyi from the Noun Project

Requirements: Your Code

- **Ability to checkpoint and restart:**
  - *Checkpoint*: Periodically write state to a file on disk.
  - *Restart*: Code can both find the checkpoint file and can resume from it.
  - *Exit*: Code exits with a non-zero exit code after writing a certain number of checkpoints, exits normally after writing final output.
  - (May need a wrapper script to do some of this.)
- **Ability to checkpoint sufficiently* frequently**



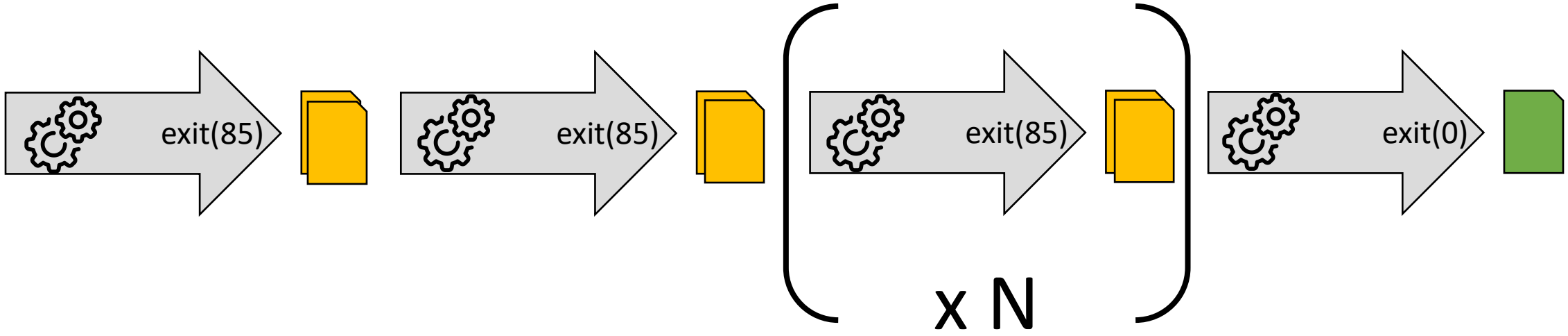
File by Tanvir Islam from the Noun Project
Gears by Gregor Cresnar from the Noun Project

* Varies by code and available resources

Context: HTCondor

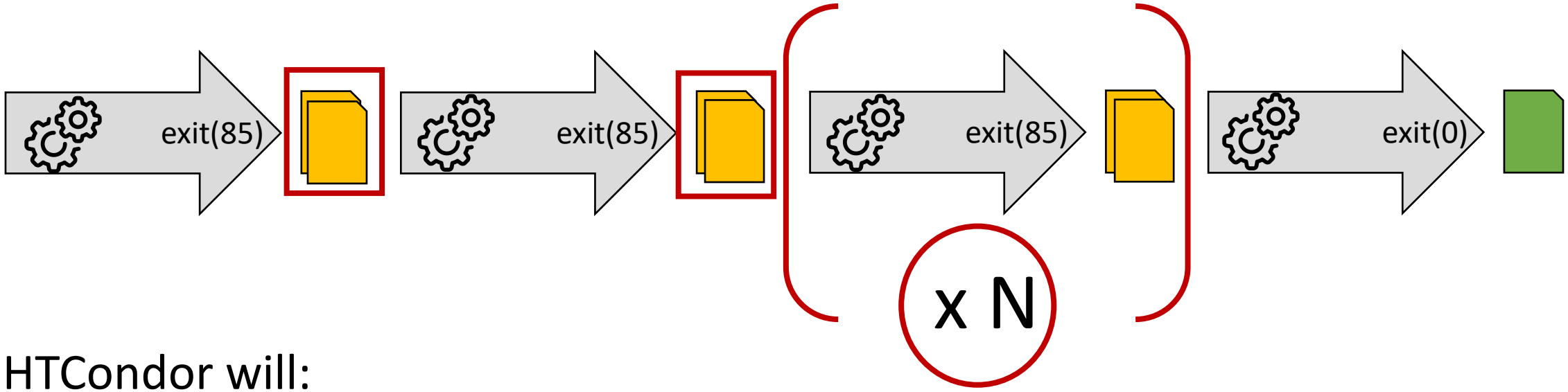
- The self-checkpointing executable is going to be run many times (to completion) by HTCondor.
- This talk assumes that HTCondor is managing job files (using file transfer, not a shared file system)
- This talk applies to jobs using the vanilla (or Docker) universe.
- The features discussed are first available in HTCondor 8.8.4/8.9.3; with improvements as of 9.0.6.

Executable Exits After Checkpoint



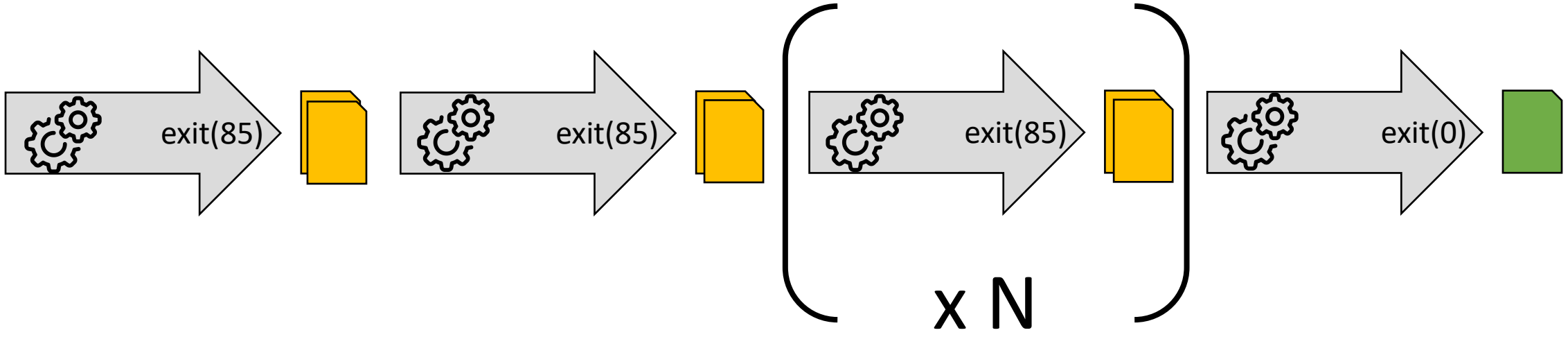
- Each executable run:
  - Produces checkpoint file(s)
  - Exits with a specific code when checkpointing, and a final exit code when done.
- Note that the executable, on its own, won't run a complete execution. It needs an external process to make it repeat.

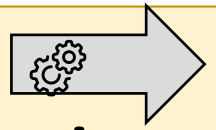
Save Checkpoint File/Resume with HTCondor



- HTCondor will:
  - Restart the executable until the overall calculation is done (exit 0).
  - Copy the checkpoint file(s) to a persistent location, to facilitate restarts if the job is interrupted.

Save Checkpoint File/Resume with HTCondor



executable = 

checkpoint_exit_code = 85

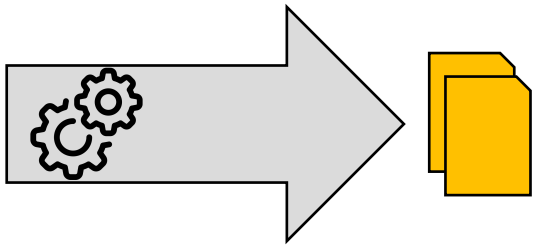
transfer_checkpoint_files = 

Sample Code: Fibonacci Sequence Generator

Stopping condition: number of iterations

Input Logic: needs to set

- Two sequence values
- Iteration counter

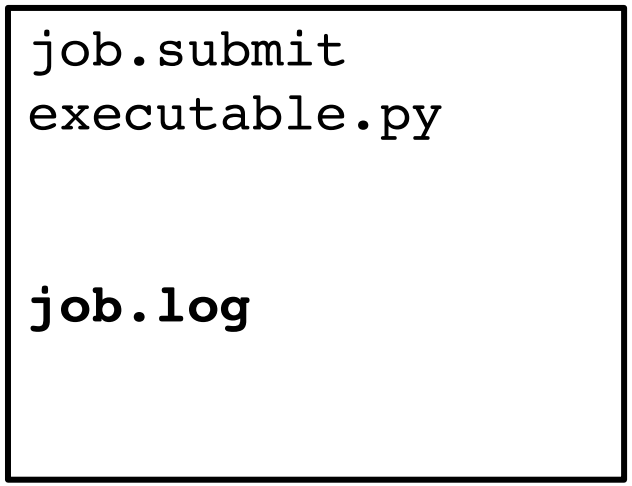


Output logic: needs to save

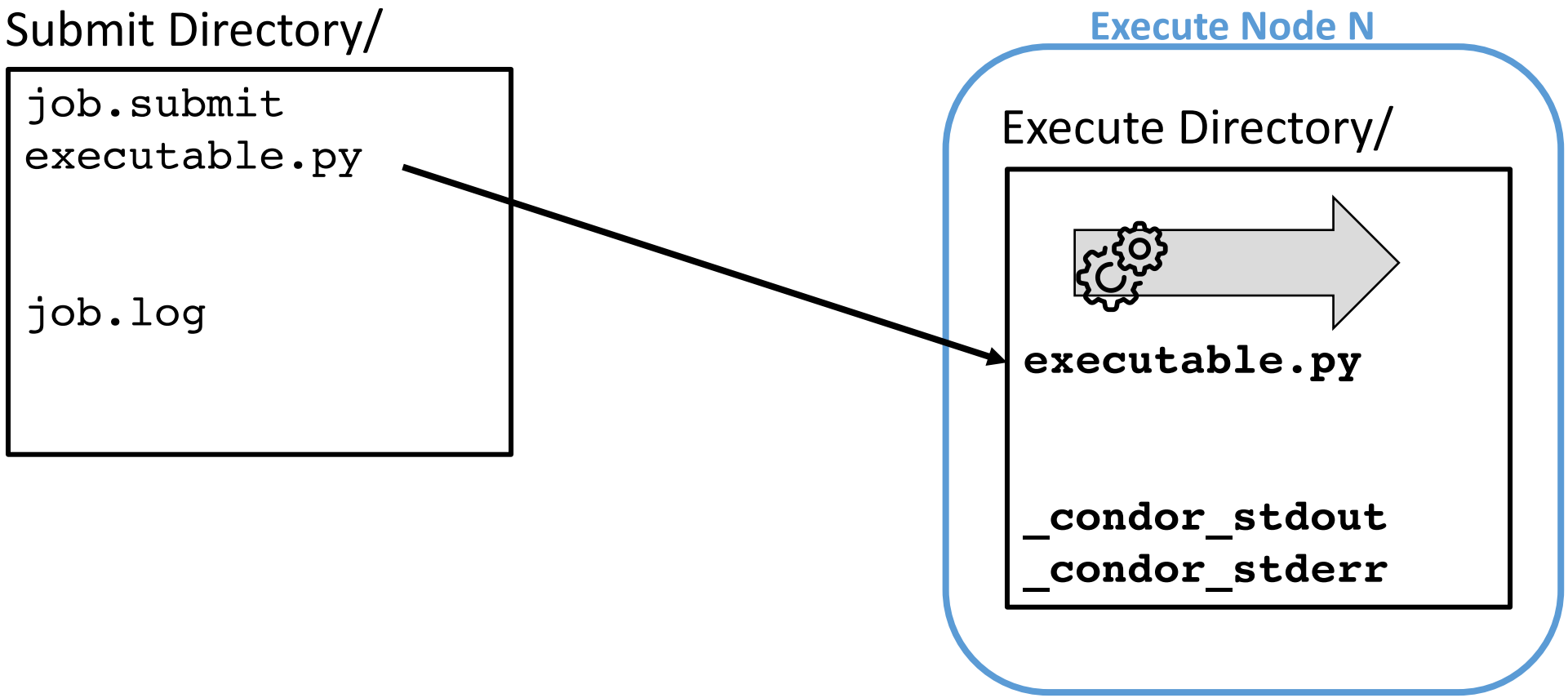
- Most recent two sequence values
- Updated iteration counter

Job Submitted

Submit Directory/



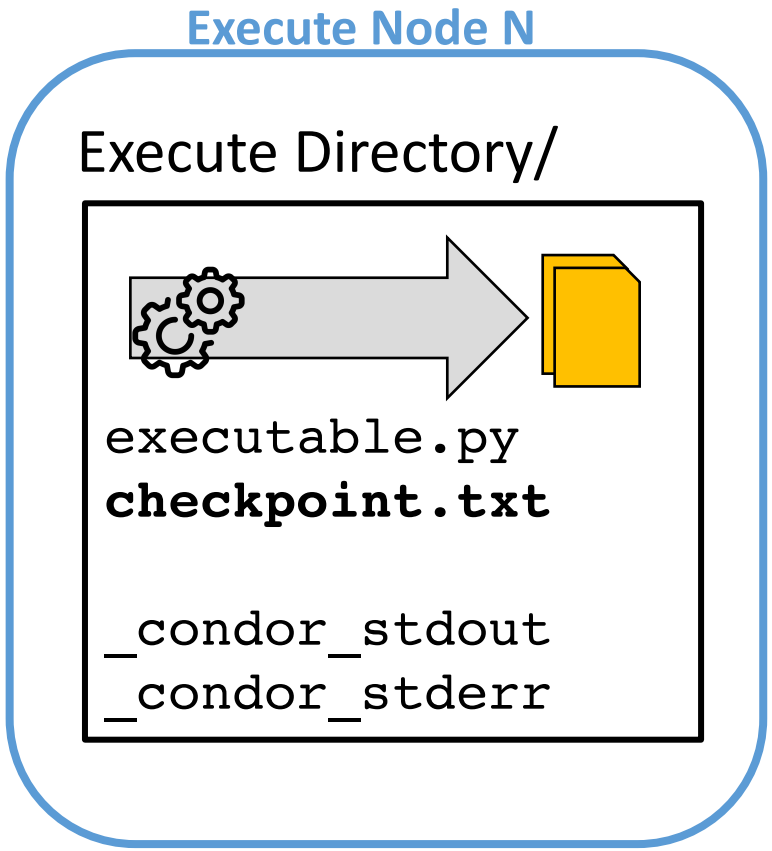
Job Starts, Executable Starts



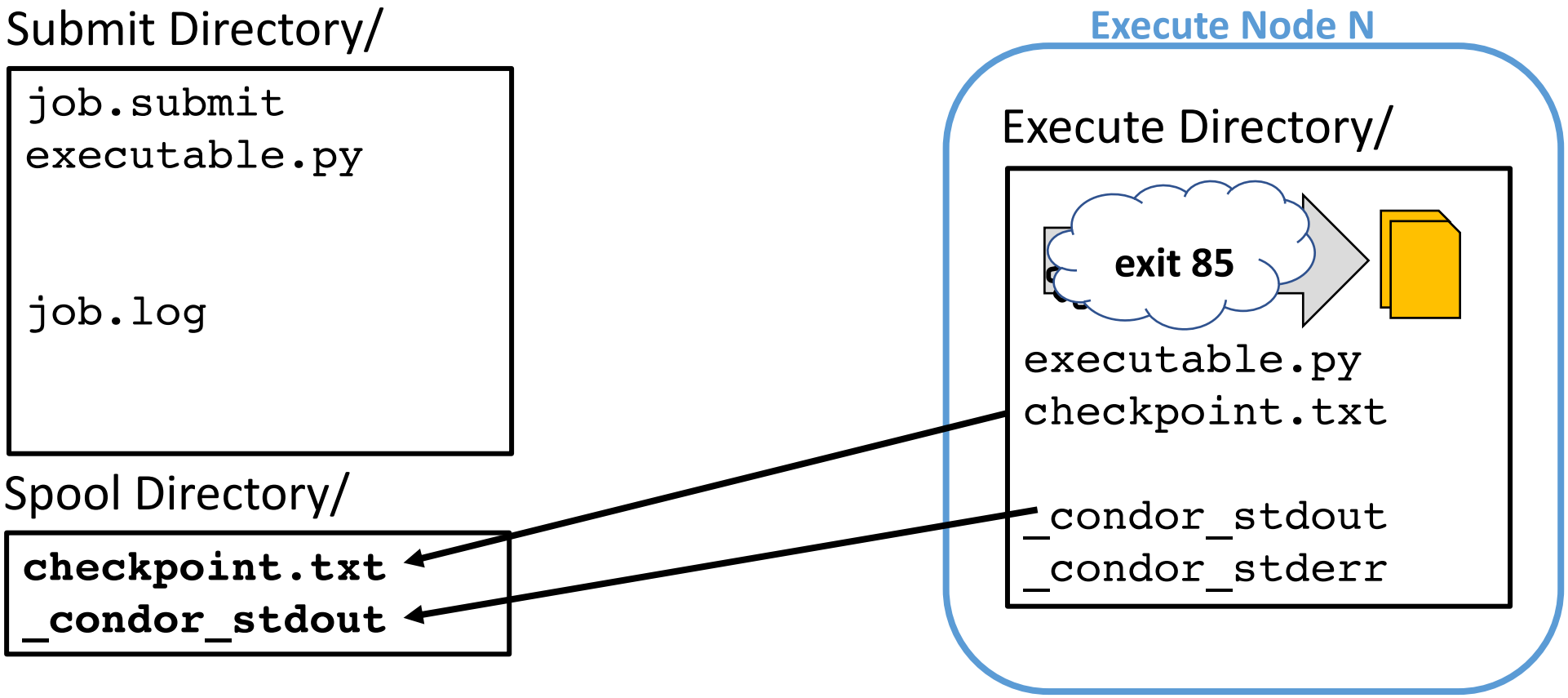
Executable Checkpoints

Submit Directory/

```
job.submit
executable.py


job.log
```



Executable Exits, Checkpoint Spooled



Executable Started Again

Submit Directory/

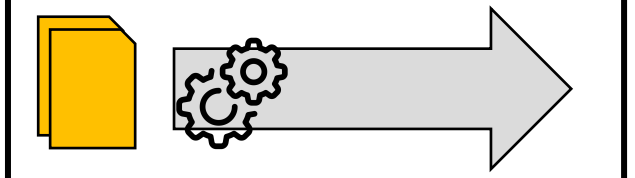
```
job.submit
executable.py


job.log
```

Spool Directory/

```
checkpoint.txt
_condor_stdout
```

Execute Node N

Execute Directory/



```
executable.py
checkpoint.txt

_condor_stdout
_condor_stderr
```

Checkpoint Cycle Continues

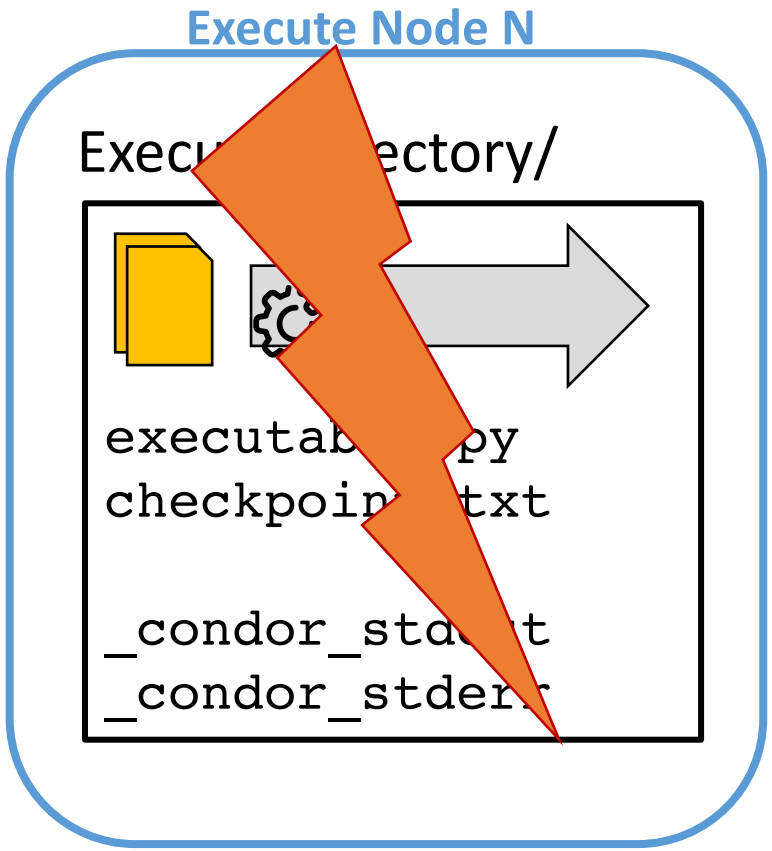
Executable Interrupted

Submit Directory/

```
job.submit
executable.py


job.log
```

Spool Directory/

```
checkpoint.txt
_condor_stdout
```



Job Idle

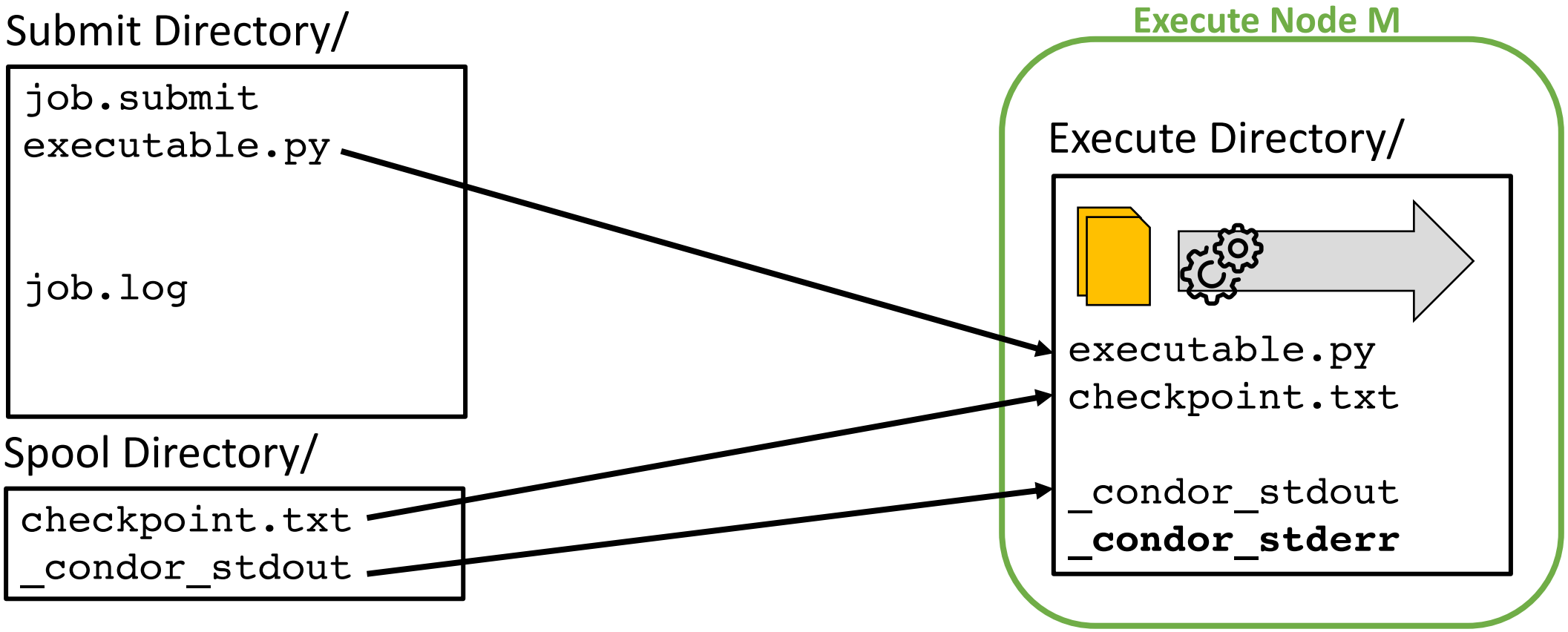
Submit Directory/

```
job.submit
executable.py


job.log
```

Spool Directory/

```
checkpoint.txt
_condor_stdout
```

Job Restarts, Executable Restarts



Checkpoint Cycle Continues

Final Execution: Executable Creates Output

Submit Directory/

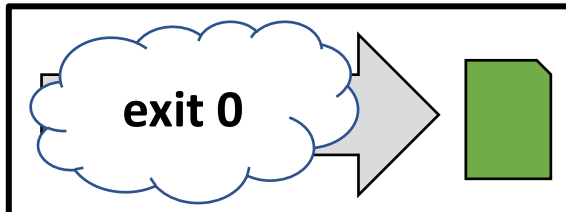
```
job.submit
executable.py


job.log
```

Spool Directory/

```
checkpoint.txt
_condor_stdout
```

Execute Node M

Execute Directory/




```
executable.py
checkpoint.txt
results.txt
_condor_stdout
_condor_stderr
```

Output Returned

Submit Directory/

```
job.submit
executable.py
checkpoint.txt
results.txt
job.log
job.out
job.err
```

Think About Output Files

- Same mechanisms for transferring output at the end of the job (triggered by executable's exit 0)
  - New output files are transferred back to the submission directory
  - To transfer specific output files or directories, use:

```
transfer_output_files = file1, outputdir/
```

- ANY output file you want to save between executable iterations (like a log file), should be included in the list of `transfer_checkpoint_files`
- This includes stdout and stderr (if using < 9.0.6.)!

Testing and Troubleshooting

- Simulate a job interruption:
  - `condor_vacate_job` *`JobID`*
- Examine your checkpoint files in the SPOOL directory:
  - Use `condor_evicted_files` *`JobID`*
  - To find the SPOOL directory: `condor_config_val SPOOL`
- Look at the HTCondor job log for file transfer information.

Sample Code: Fibonacci Generator

Best Practices

- Scaling Up
  - How many jobs will be checkpointing?
  - How big are the checkpoint files?
  - How much data is that total?

Avoid:

- Filling up the SPOOL directory.
- Transferring large checkpoint files.

- Checkpoint Frequency
  - How long does it take to produce a checkpoint and resume?
  - How likely is your job to be interrupted?

Avoid:

- Spending more time checkpointing than running.
- Jobs that will never reach a checkpoint.

Alternative Checkpointing Method

- If code can't exit after each checkpoint, but only run + checkpoint continuously, transfer of checkpoint files can be triggered by eviction.
- Search for "when_to_transfer_output" on the condor_submit manual page; read about ON_EXIT_OR EVICT
- This method of backing up checkpoint files is less resilient, as it won't work for other job interruption reasons (hardware issues, killed processes, held jobs)

Resources

- HTCondor Manual
  - Manual > Users' Manual > Self Checkpointing Applications
  - https://htcondor.readthedocs.io/en/latest/users-manual/self-checkpointing-applications.html
- Materials from the OSG Virtual School 2021
  - OSG Virtual School > Materials > Overview or Checkpointing Exercises
  - https://opensciencegrid.org/virtual-school-2021/materials/#self-checkpointing-for-long-running-jobs

Questions

Acknowledgements

This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATh Project.