Enabling Fastai Multi-GPU/DDP Training in Jupyter

- `fastai`: to make building high-performing AI application easy
  - Open source, free lessons videos + Jupyter notebooks
- **Distributed Data Parallel (DDP)**:
  - *a multiprocess-based parallelism to speed up training with multiple GPUs/nodes.*

"`fastai` + DDP" cmdline app is trivial [1], but ...

> "*Distributed training **doesn't** work in a notebook...*"[2]

Ddip tries to bridge this gap.

[1] See Reproducing DAWNBench winning results in a few lines of code
[2] *FastAI's tutorial on How to launch a distributed training*

`Ddip` - Distributed data "interactive" parallel

An iPython extension to control PyTorch DDP from within Jupyter, uses `ipyparallel` underneath.

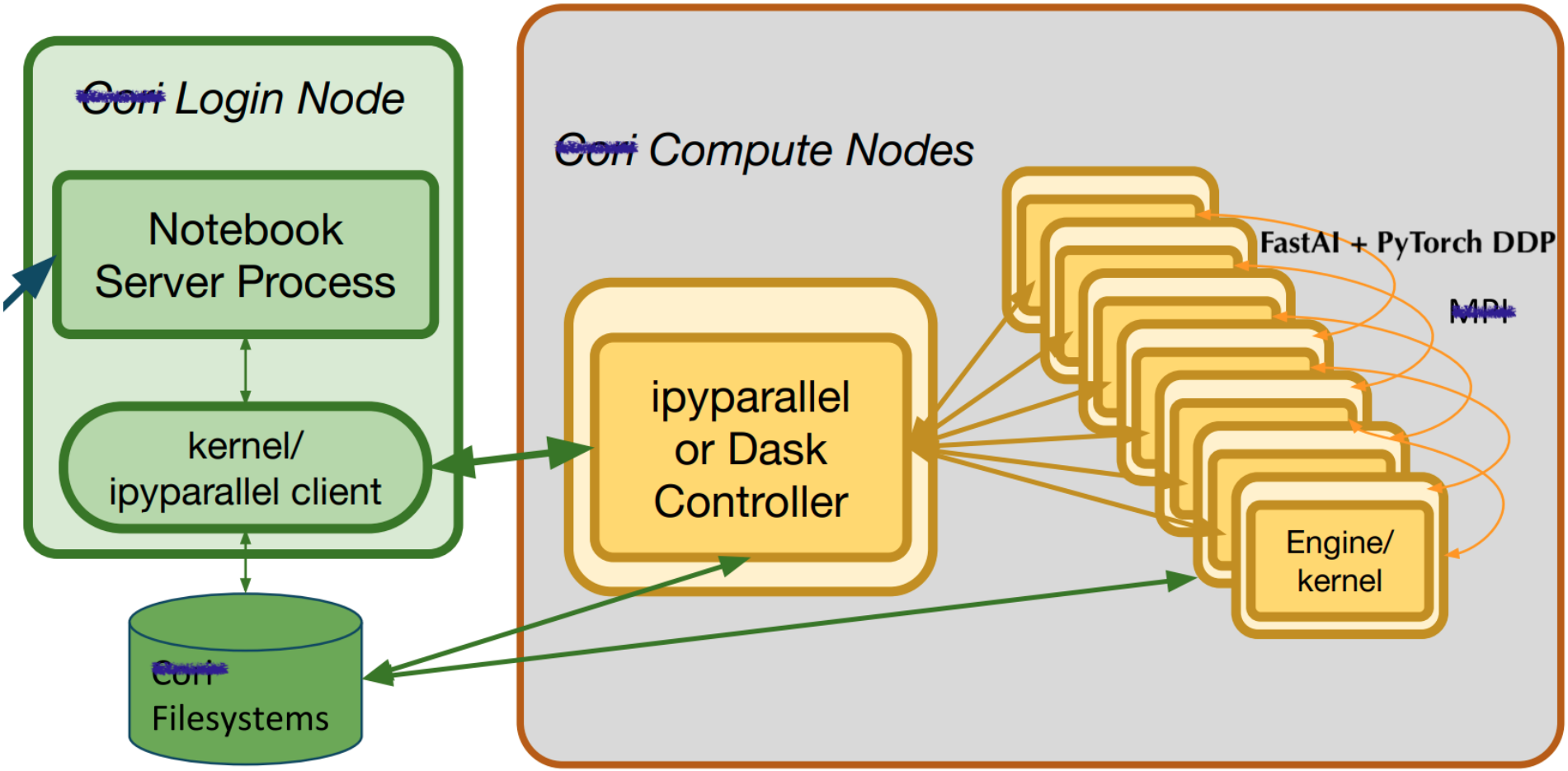

Diagram borrowed from a presentation by NERSC/LBNL at https://bluewaters.ncsa.illinois.edu/liferay-content/document-library/content/Blue%20Waters%20Webinar.%20Jupyter%20as%20the%20Interface%20to%20High-Performance%20Computing.%20Thomas%20&%20Cholia.pdf

`Ddip` - Distributed data "interactive" parallel

[Overview and Usage]

[Speedup in Training]

Limitations

- Works a single host only. Luckily `ipyparallel` does support cluster of nodes.
- Not all models gain: one model is flat, one has accuracy problem.
- ***a wgan model manages to achieve linear slow down!!*** 😭

`Ddip` - Distributed data "interactive" parallel

Lessons learned:

1. Python's dynamic nature empowers: patch classes dynamically, toss objects/functions across multiple processes.
2. Hooks/callback architecture brings openness and flexibility: Jupyter, `fastai`.
3. Multiprocess, multi-GPUs and Jupyter: both **Complexity and Opportunities**:
    - data movement, race conditions, proc & mem mgmt.
4. Design choices: semantics (`%`,`%%` vs library calls), deadlock solution, implicit convenience.

Looking Forward

- `fastai v2` is out already.
- `nbdev` as productivity boost.
- Support cluster of nodes.
- Feedbacks and contribution via github are welcome!

Github Repo: https://github.com/philtrade/Ddip/