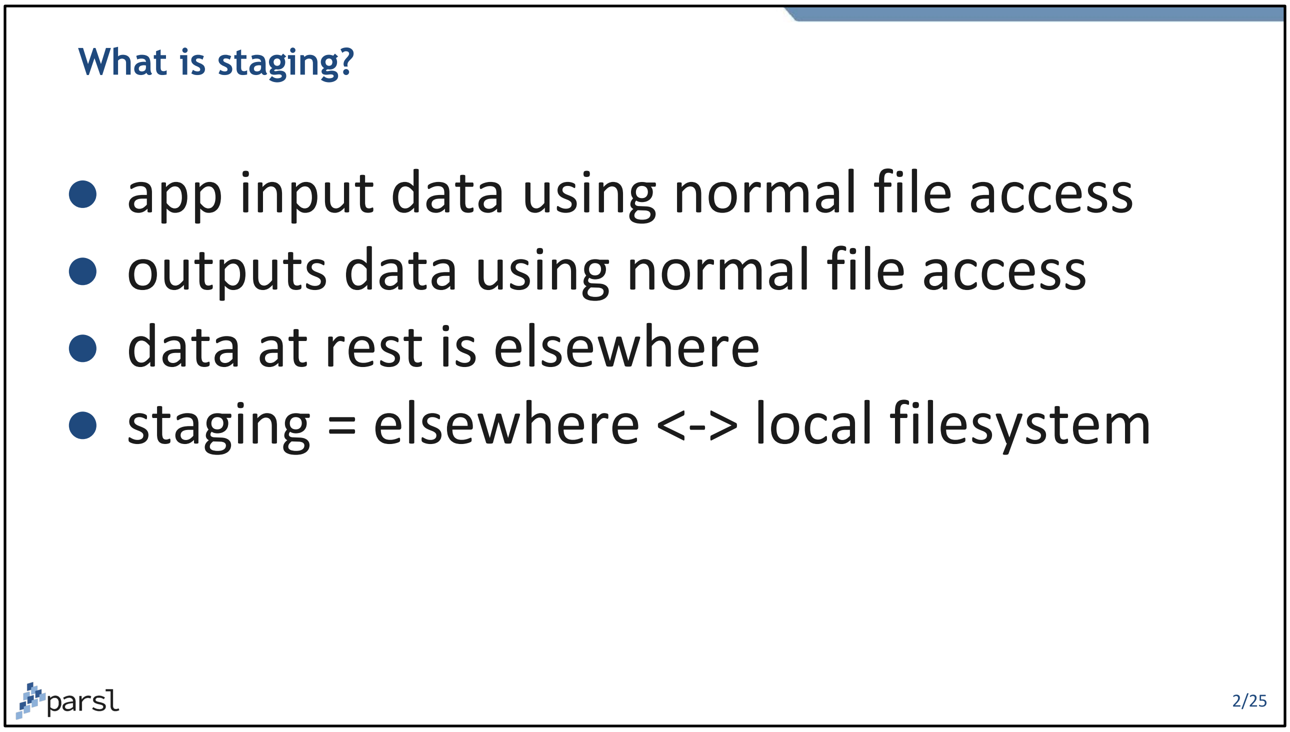
Data Staging in Parsl

Ben Clifford (benc@hawaga.org.uk)

http://parsl-project.org



What is staging?

- app input data using normal file access
- outputs data using normal file access
- data at rest is elsewhere
- staging = elsewhere <-> local filesystem

parsl

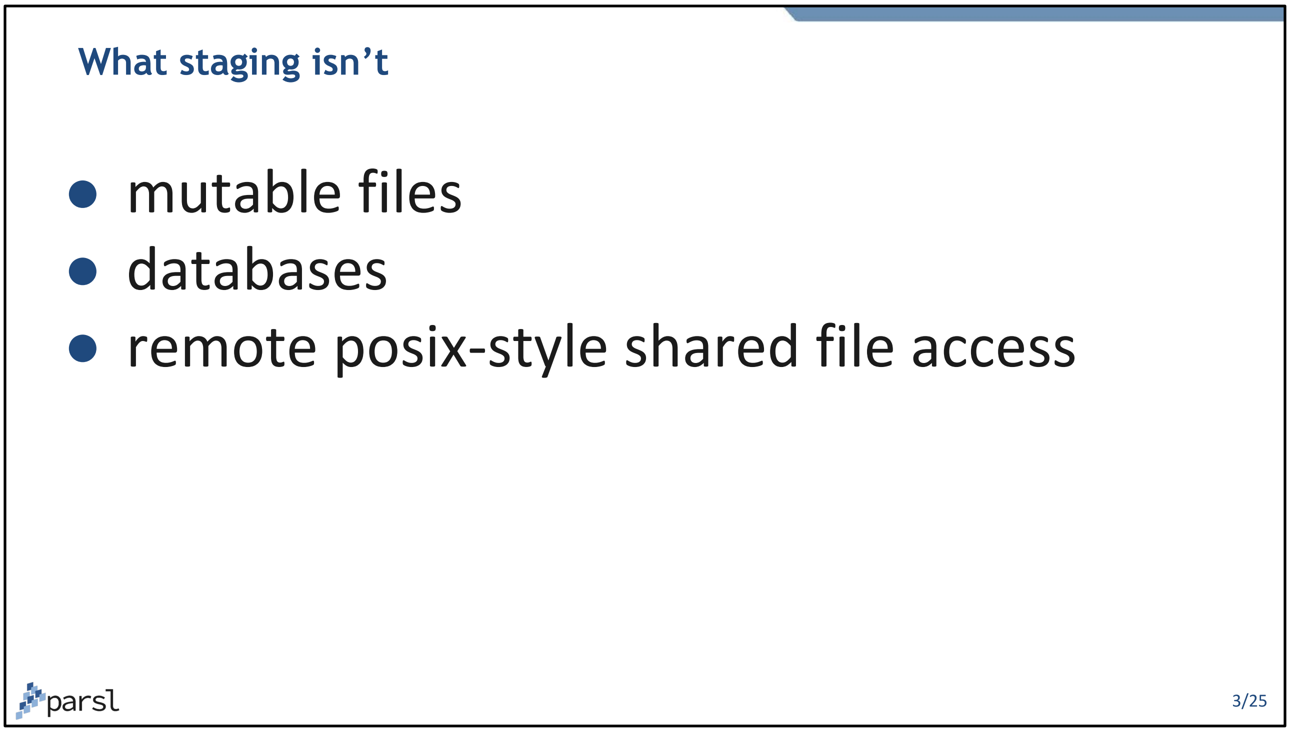
2/25

* We want to specify the environment an app runs in. Usually pass in parameters and get return values out via usual python function invocation.

* but maybe we want some files to be around when app runs, and maybe some of the output will also be files rather than the python return value - especially but not only bash apps

* At the workflow level, want to describe the files that will be present when the app runs - but not *how* those files get there. facilitate location independence.

* When I invoke an app, I want parsl to get those files from their global resting place and put them into your app environment (stage-in) and get them from your app environment and put them in their resting place (stage-out)



What staging isn't

- mutable files
- databases
- remote posix-style shared file access

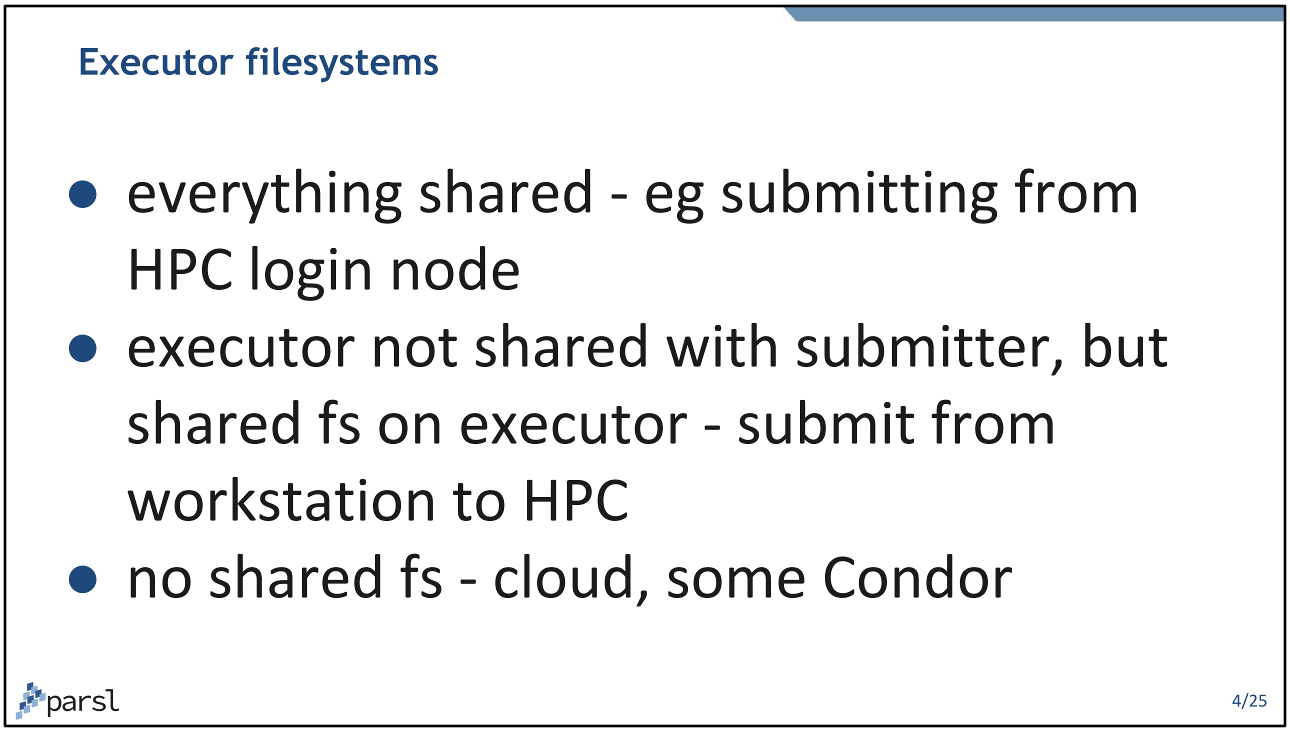
parsl

3/25

* mutable state - eg "databases", whole directory trees where people are updating different bits all over. There are different protocols for that, that have different semantics: shared file systems, sql over a network connection.

* deliberately avoiding "posix" style of file operations on remote files - like a shared filesystem would.

If you want to use a shared file system under this, that is fine - this is a subset of the operations provided by such a thing. if you want to do your own db-style access - that's also fine, in as much as parsl won't stop you (but you have to deal with things like location abstraction yourself). parsl data staging isn't going to help you.



Executor filesystems

- everything shared - eg submitting from HPC login node
- executor not shared with submitter, but shared fs on executor - submit from workstation to HPC
- no shared fs - cloud, some Condor

parsl

4/25

Three broad models of executor file systems:

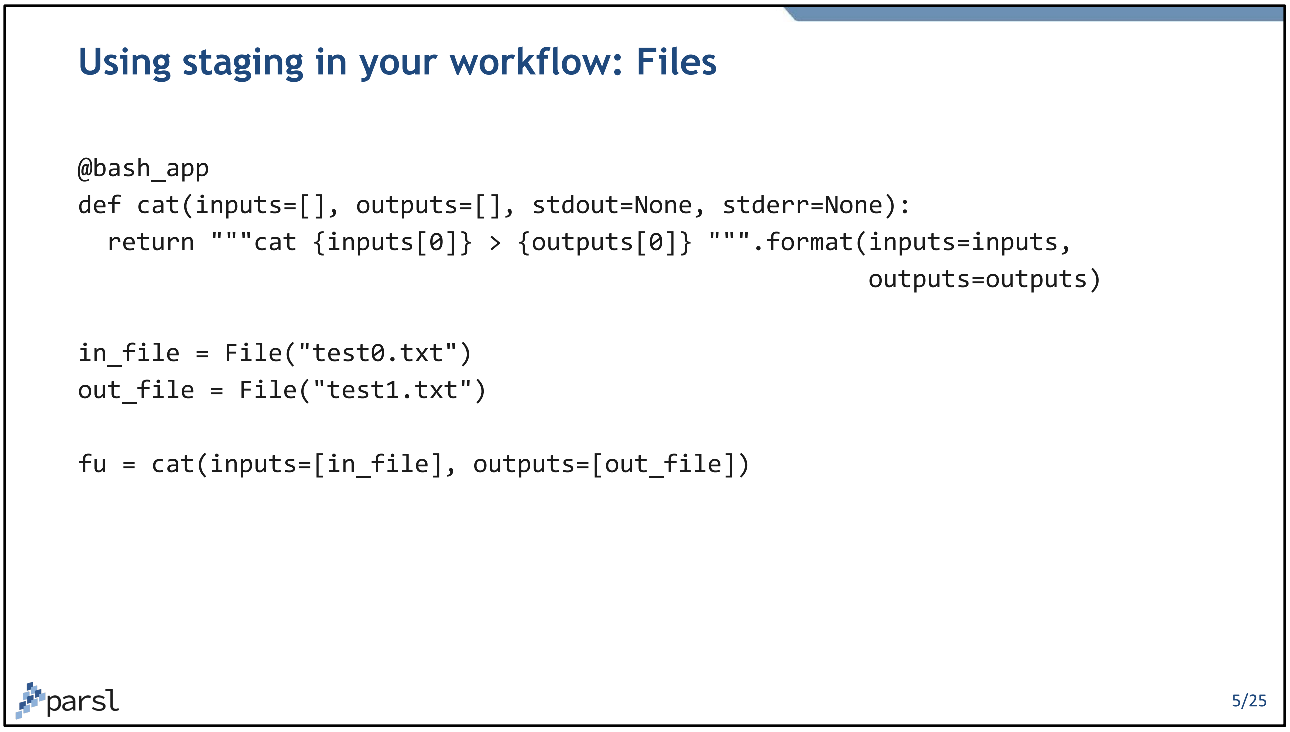
* everything relevant is shared - anywhere I run code (parsl submit side, eg login node), any worker node, i always have access to the relevant filesystems via posix. eg Submitting from a login node of a classic HPC supercomputer. this might be a traditional cluster shared FS (i.e. stuff that looks like NFS) or it might be stuff that is like that but not quite a cluster fs - eg cern vm fs

* parsl submit side lives in on place, submitting to one or more remote executors, and on that executor, there is a filesystem shared between all places that my job will execute - eg submit from my workstation onto an HPC supercomputer

* individual worker nodes of the executor do not have a shared file system:

* submit from my workstation to cloud. eg a number of AWS nodes, which don't have a shared filesystem.

* some condor environments.



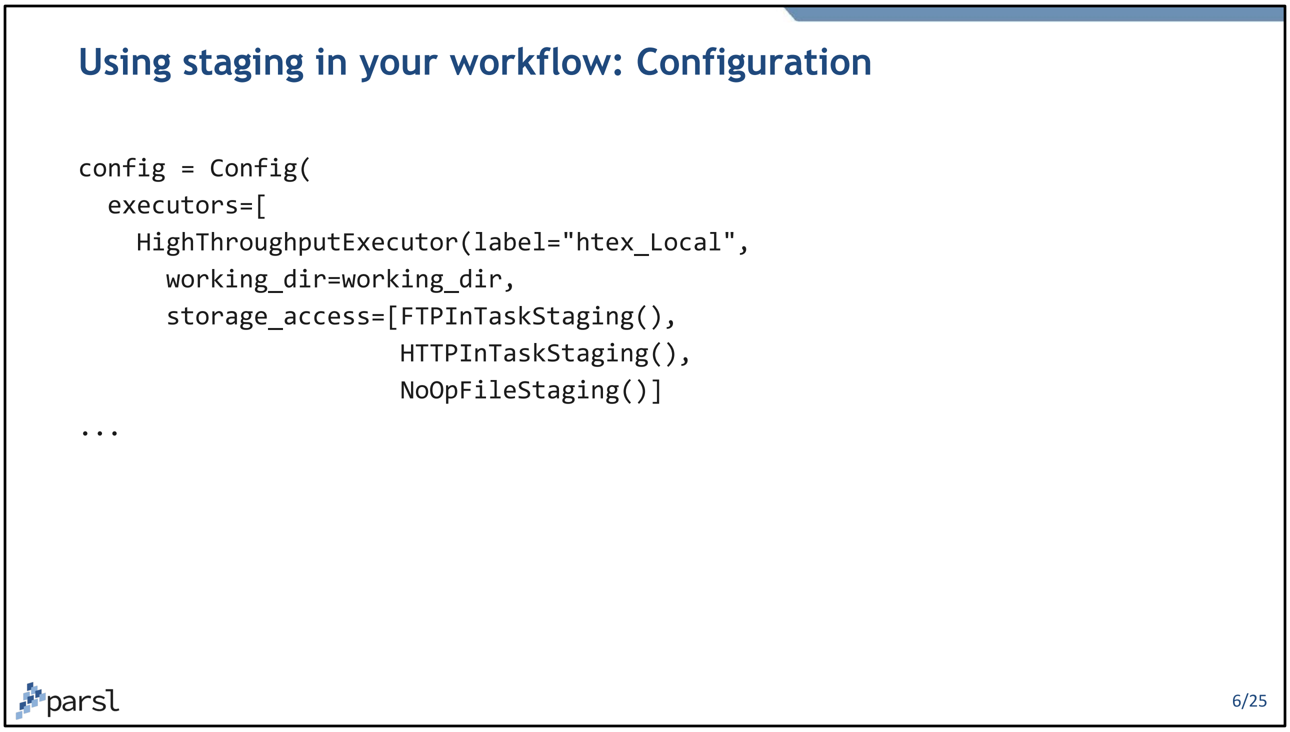
Using staging in your workflow: Files

```
@bash_app
def cat(inputs=[], outputs=[], stdout=None, stderr=None):
  return """cat {inputs[0]} > {outputs[0]} """.format(inputs=inputs,
                                                        outputs=outputs)


in_file = File("test0.txt")
out_file = File("test1.txt")

fu = cat(inputs=[in_file], outputs=[out_file])
```

parsl 5/25

This is a consolidation and extension of the code that was already in there in 0.8.0 so if you've worked with parsl staging already, some of this will be familiar.

 * what this looks like in the parsl app API:
  * File class - describes a file that usually lives outside of Parsl
   * identify by a URL or partial URL - specifically a path fragment without a scheme refers to files on the local filesystem (file:)
   * code example - an app, a file and an invocation
    * use in inputs=[], outputs=[]
    * get filepath on remote side: File.filepath - or str() - or implicit str()



Using staging in your workflow: Configuration

```
config = Config(
  executors=[
    HighThroughputExecutor(label="htex_Local",
      working_dir=working_dir,
      storage_access=[FTPInTaskStaging(),
                      HTTPInTaskStaging(),
                      NoOpFileStaging()]
...
```

* configuration in Config()

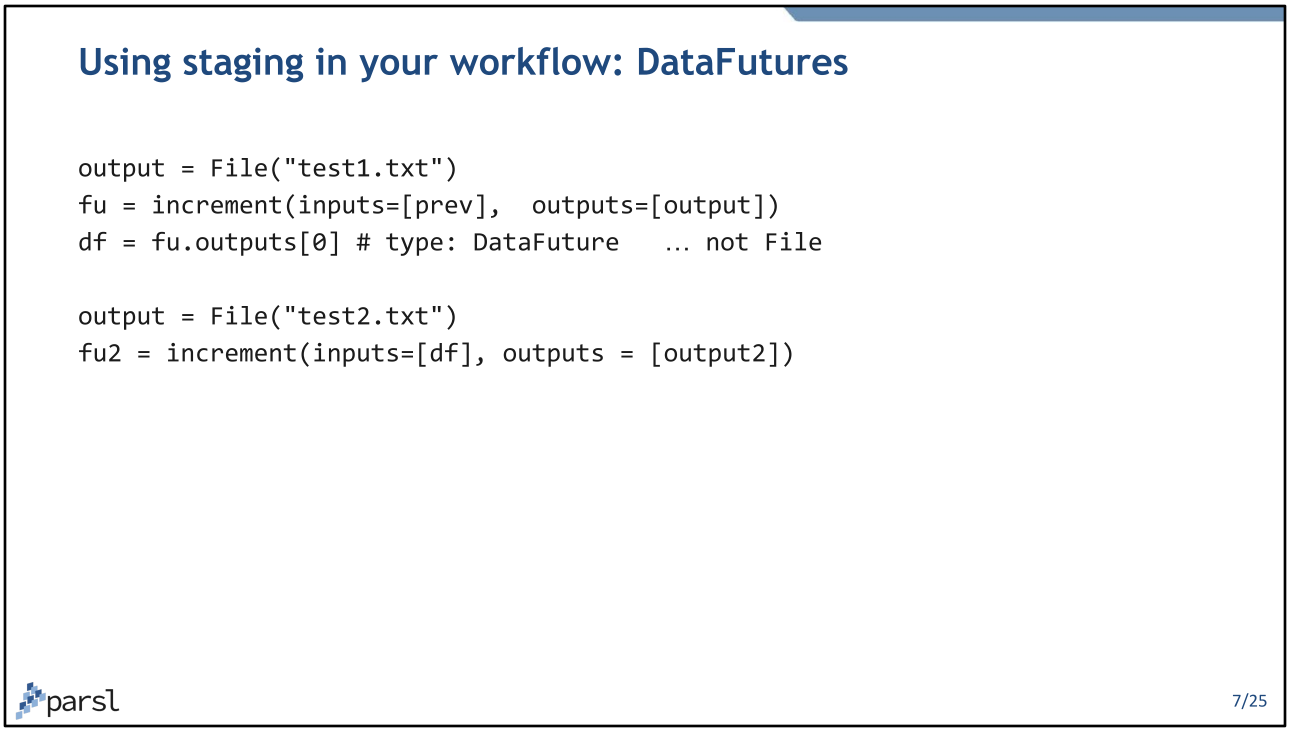
    * for each executor, give a list of StagingProviders

* a staging provider is:

    * part description of how the world is - "you can find X kind of file here; this globus endpoint is the same as my cluster shared filesystem"

    * part description of how the world can be changed - "you can make this kind of file appear by doing Y"

* staging providers are configured *per executor* - because in the same way that batch submission systems might be different when executing in different places, the way that a particular file ends up staged in/out might also be different. general abstraction of location independence where environment specific stuff should be pushed into the Config as much as possible



Using staging in your workflow: DataFutures

```
output = File("test1.txt")
fu = increment(inputs=[prev],  outputs=[output])
df = fu.outputs[0] # type: DataFuture   … not File

output = File("test2.txt")
fu2 = increment(inputs=[df], outputs = [output2])
```

parsl

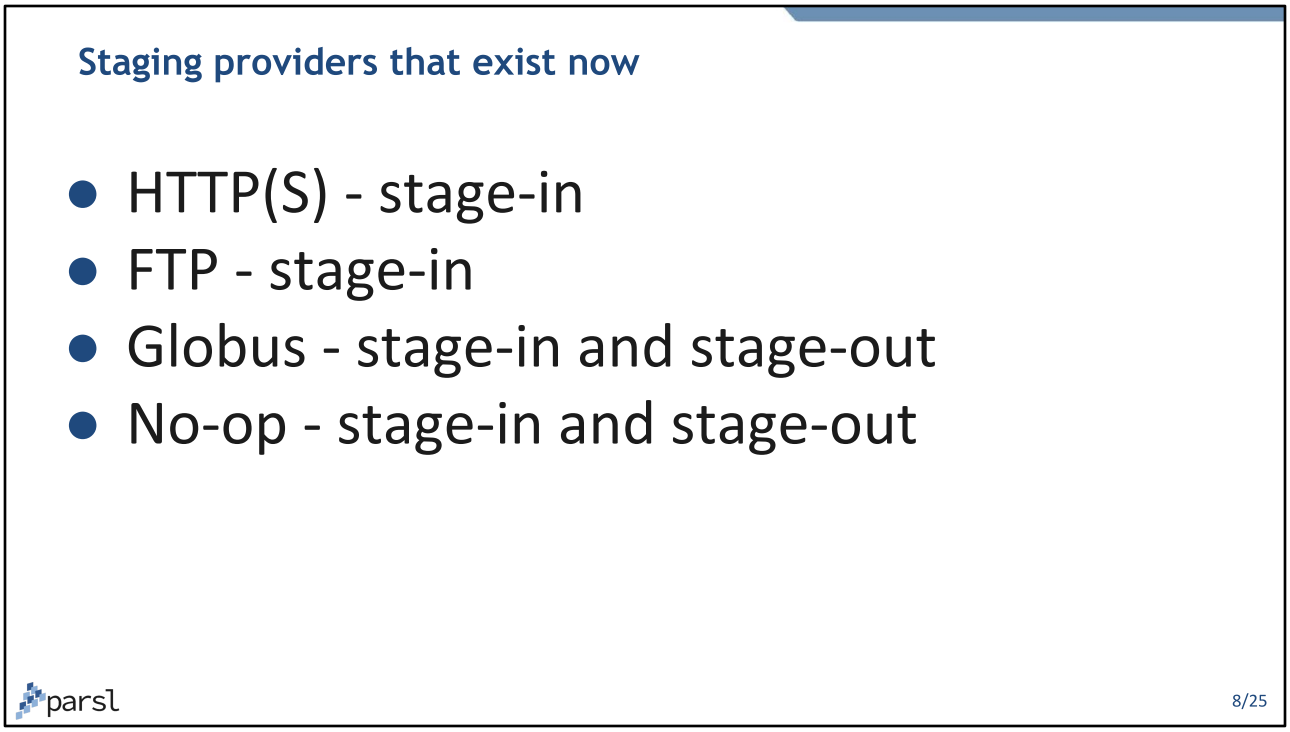
7/25

* `DataFuture`s for ordering things

    * Already: Parsl orders thing using standard python Futures: Futures that contain a result of a previous app; or Futures that complete so we know something is done, without containing a value - Application futures.

    * DataFuture: A future that completes when staging is complete. On the workflow dev side, only really see this on the stageout side: when the named output file has been staged out.

* code exmaple: we use a DataFuture to represent the file passing between the apps



Staging providers that exist now

- HTTP(S) - stage-in
- FTP - stage-in
- Globus - stage-in and stage-out
- No-op - stage-in and stage-out

parsl

8/25

 * staging providers: that parsl has in master now

  * it is the job of staging providers to make that environment come to be, in their own unique way.

  * parsl core is deliberately not prescriptive of what providers should do other than "make the world look like this"

  * what is in master now:

   * stagein: http, ftp [two modes]

   * stagein and out: globus

   * shared filesystem via file: and no-op file provider

A Diagram of Staging

parsl

9/25

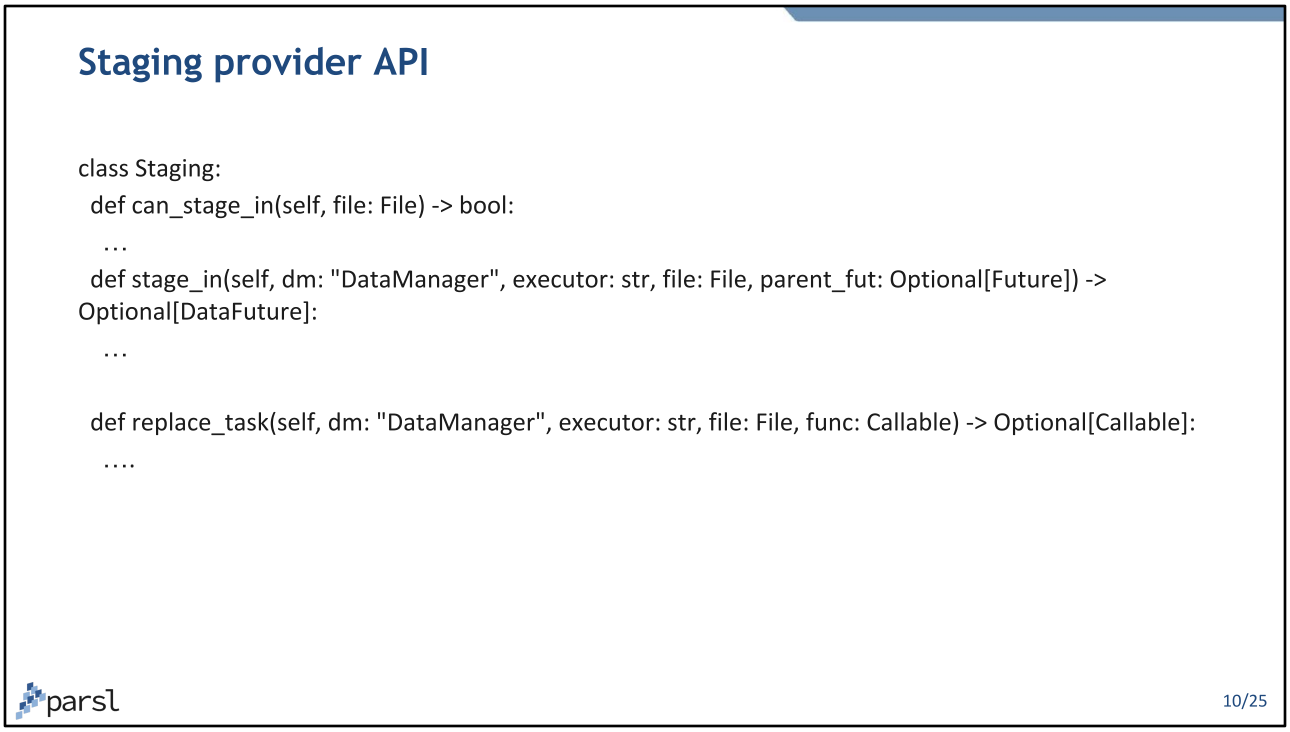
* concrete description of how Globus staging works at the moment

- involving descriptions of parameters (briefly) on a diagram that also shows site-local shared filesystem.

- brief description of what Globus is

- note that it needs a globus server at both ends, which can be awkward if you're submitting from your workstation

but eg "globus connect personal" (I have this running on my laptop fairly straightforwardly)



Staging provider API

```
class Staging:
  def can_stage_in(self, file: File) -> bool:
    …
  def stage_in(self, dm: "DataManager", executor: str, file: File, parent_fut: Optional[Future]) ->
Optional[DataFuture]:
    …

  def replace_task(self, dm: "DataManager", executor: str, file: File, func: Callable) -> Optional[Callable]:
    ….
```

parsl

10/25

* The other side of what is going on is the staging provider API:

* previous code talked about how to tell parsl about what files need staging

* the other side of things is telling parsl *how* to do that staging

* mentioned thinks like globus, http previously

* the core of parsl doesn't care *how* a file gets staged in. There are staging providers (mentioned earlier in Config()) and you can plug your own in.

* Show API slide of two calls:

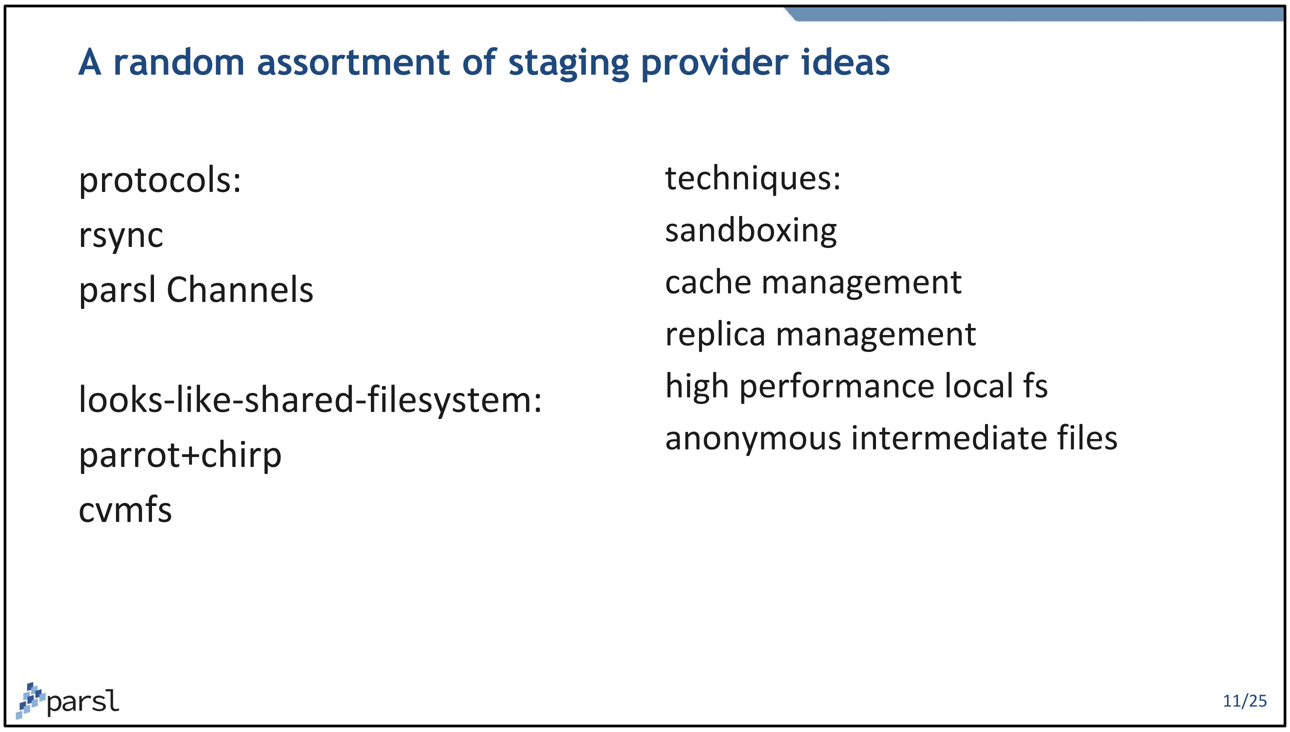
* it's more complicated than this, but basically a staging provider gives:

* can_stage_in: decide if the provider can handle a particular file

* Especially note can_stage is not specifically based on the scheme: it can look at the whole URL and make decisions (for example, maybe based on a hostname, or a path fragment). Although usually probably only looking at the URI scheme is what we would do.

* stage_in: prepare for staging, and add arbitrary staging tasks into the task graph

* replace_task: put a wrapper around code to run as part of the task itself



A random assortment of staging provider ideas

protocols:
rsync
parsl Channels

looks-like-shared-filesystem:
parrot+chirp
cvmfs

techniques:
sandboxing
cache management
replica management
high performance local fs
anonymous intermediate files

parsl

11/25

* random bucket of prototypes and ideas for staging providers -

* go through these pretty fast and deliberately "here are some features we definitely want, but also some that are half baked"

so although there is a lot of text in these notes, it shouldn't take much time - leave them in the speaker notes?

* whatever isn't in master above

* rsync

* in-args staging

* cache management

* sandboxing

* shared filesystems are not only conventional NFS-but-better filesystems, but also things that look close enough (eg cern vm fs or parrot)

"assume cvmfs is available everywhere so skip /cvmfs staging for file:///cvmfs/*"

parrot: "i know url scheme foo://bar/ is mapped into my workers fs namespace by parrot, so I don't need to do any staging - in the same way that I don't for shared-fs -- but instead i need to rewrite the path to where I know parrot will expose it"

* staging from shared fs to different part of fs (eg local tmp) which has different access characteristics (different sharedness - my local scratch space -> cluster shared fs; different performance characteristics - eg burst buffer) - rewrite file URL to a different path rather than leaving as is, and make copy at the same time.

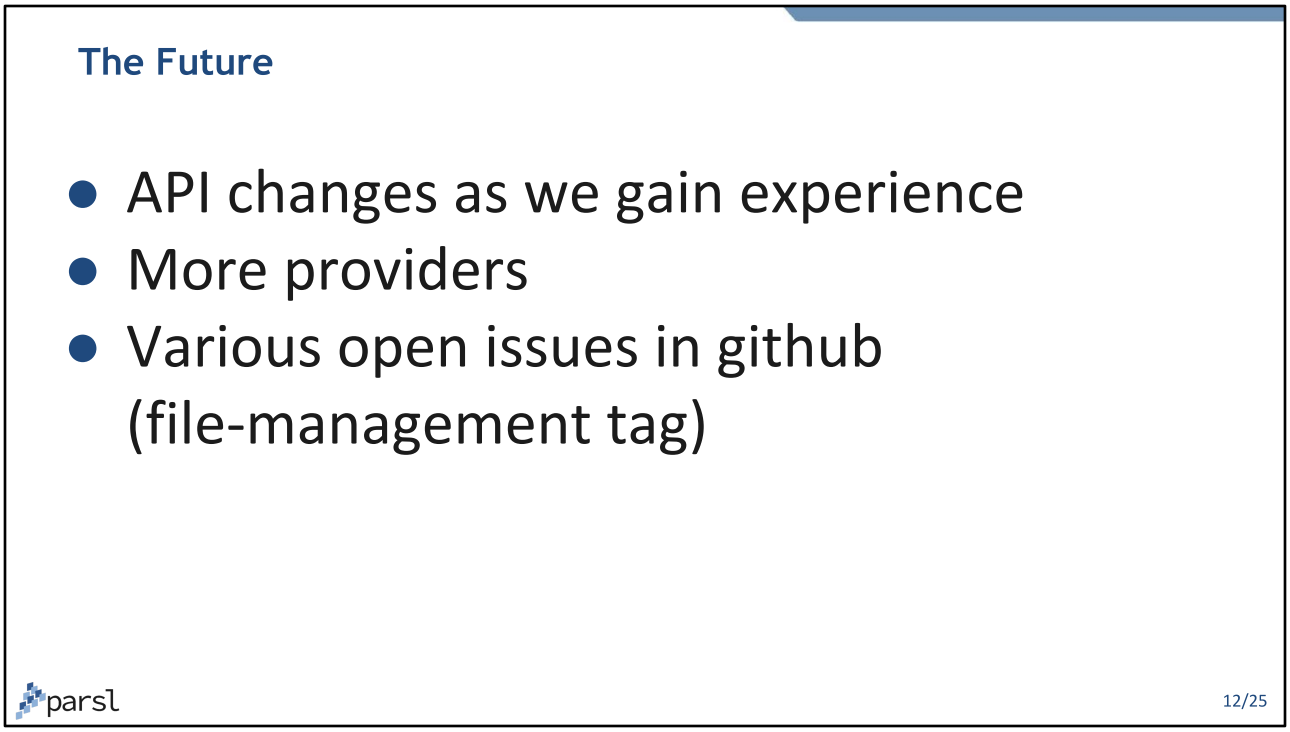
* multi-hop staging: no globus access to worker node - so stage somewhere else with globus and then a subsequent copy

* Channel based staging: channels can move files around - and do for the purpose of getting scripts into place. so can file: provider use Channels for get file:'s in place too?

* globus-based access for file: URLs - same staging code but understand that a file:// URL also corresponds to a globus:// URL if you have a globus connect server installed

* "intermediate anonymous-location" files moved around by globus provider: i want an intermediate file - it doesn't need to be staged out to anywhere persistent outside of the workflow run, so leave it where it is created (by task A) until such time as a future task (task B) wants it, and do globus staging directly from the original location. In this case, the globus://location/ URL scheme is probably not the right URL scheme. Drive task placement based on intermediate file location rather than staging?

* Replicate management: URL to a logical filename, interface to replica management to find where the file - distrinct from "internmediate anonymous-location" model which doesn't persist data beyond a particular workflow.



The Future

- API changes as we gain experience
- More providers
- Various open issues in github (file-management tag)

parsl

* The future

* I expect the API to change some as it is used more

* for example poor support for transferring in a lot of files through the same mechanism

* I expect more interesting providers to appear - teaser for "tomorrow I'll be doing a hands on write-a-provider live-coding presentation"

* outstanding issues and work in github (file-management label)