Overview

- What's the problem?
  - Software licenses are a limited resource.
  - How do we run jobs so that they are guaranteed to get all the licenses they need?
- A solution
  - Model the license economy using shadow licenses
- The implementation
  - The License Shadowing Daemon
- What can go wrong?

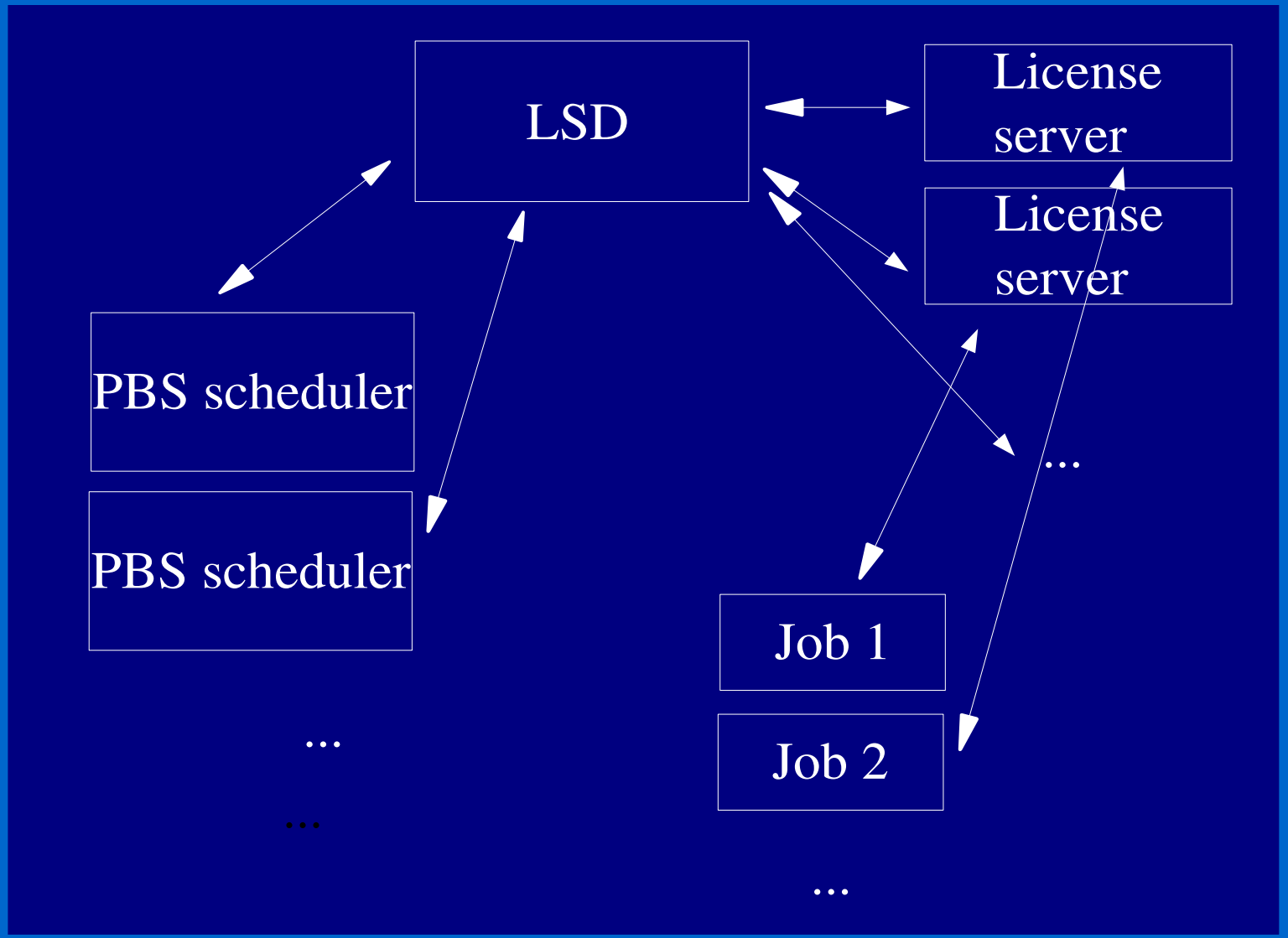
What's the problem?

- License basics: seats, servers, etc.
- The easy but flawed approach: don't start a job if the license server(s) say there aren't enough seats.
- The big problem: jobs don't check out licenses as soon as they run.
- e.g. 2 seats for software `foo`, 2 already running jobs are going to use foo in 10s time. Nothing to stop scheduling a 3rd foo job which will fail.

A partial solution

- Use the software usage assertions in the job to model the license consumption across the whole economy
- ```
  #!/bin/sh
  #PBS -lsoftware=foo:bar
  #PBS -lncpus=12
  foo foo.in > foo.out
  bar bar.in > bar.out
  ```
- But what if they don't? We kill them.
  ```

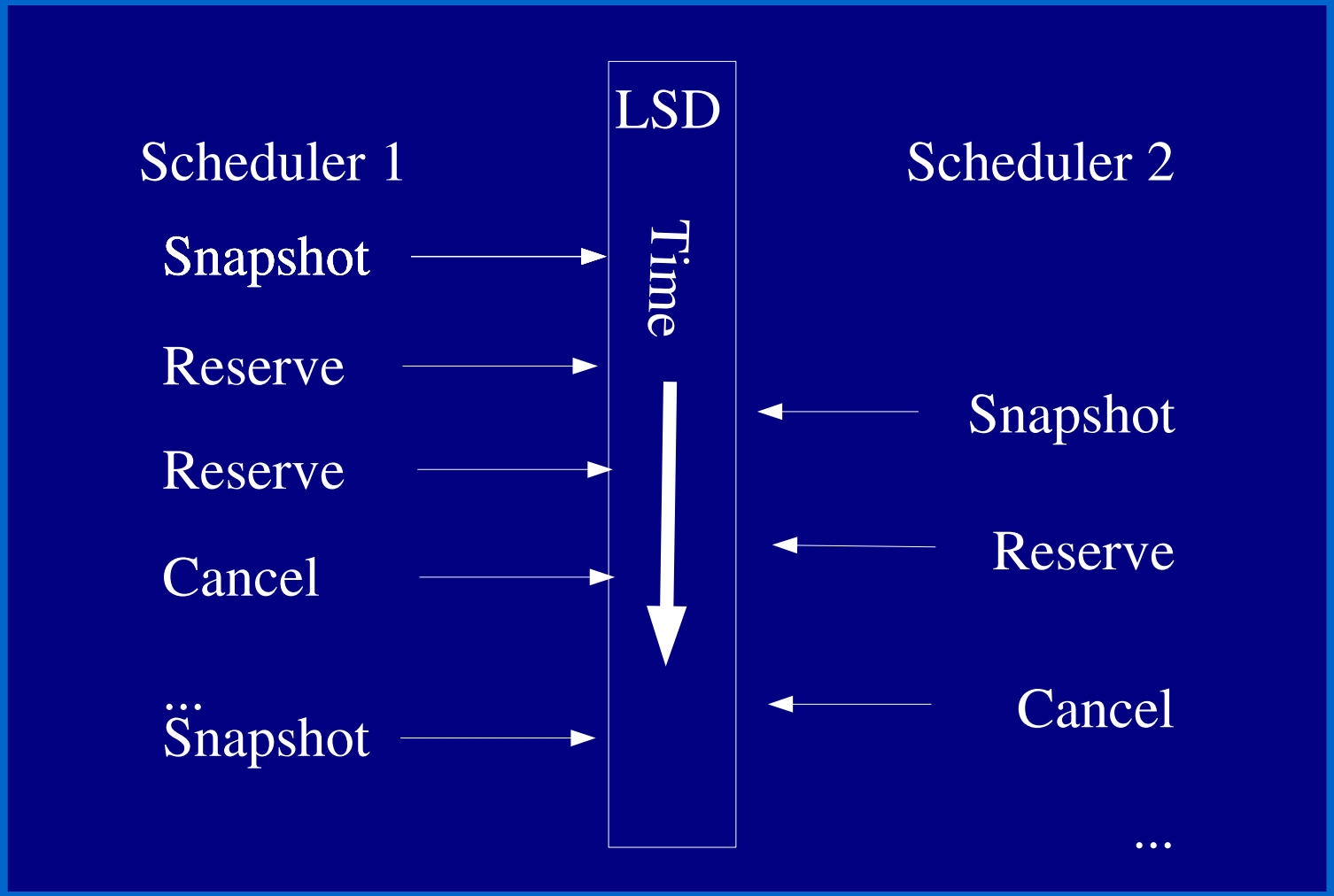
The knee bone's connected to the....



Model behaviour

- LSD maintains an internal model of the license economy that shadows the actual license economy.
- The model is built from info sent from the PBS schedulers
  - Snapshots: all running and suspended jobs
  - Reservations: Can I run job X?
  - Cancellations: OK, you said I can run X, but I've changed my mind.

Who says what when



The really ugly parts

- Licenses are consumed in idiosyncratic ways. We have to model these idiosyncrasies.
- How many seats do we have anyway?
- External consumption of seats which effectively depletes the pool.
- We have to detect rogue jobs and kill them.
- Have to parse the human-oriented output from license servers.

The implementation

- Event-driven python program built on top of asyncore.
- No blocking reads: if LSD blocked, both PBS schedulers would block.
- Vendors' license consumption idiosyncrasies are handled by plugins.
- A plugin is just a python module containing some agreed definitions.

Plug and play

- A plugin must have
  `def createInstances()`
  and return a dictionary of BasePlugin instances, keyed by software name.
  e.g. ***`{"foo":FooPlugin(), "bar":BarPlugin()}`***
- Extend BasePlugin to encapsulate all vendor weirdness and parse license query output.
- And drop your ***`*.py`*** files in the ***`plugins`*** dir

What can go wrong?

- If a job checks out a license without asking for it, there will be a delay before it gets killed. During that period, licenses are overcommitted.
- Possible ambiguity in identifying rogue jobs. When in doubt, don't kill.
- Users outside of PBS can consume seats. Allow some headroom for this and dynamically adjust pool size to cope.
- All schedulers need to be running in order to model the whole economy.

Now what?

- Its ready to test.
- Probably just fire it up on the live system, but with the schedulers ignoring what it says for the time being
- A test environment could be created by building a basic license mechanism for test jobs to use.
- Will be able to monitor it via ***`http://hostname:nnnn`***