Parameters:

- not just theoretically infinite space, but practically infinite.
- no easily definable common set, but serious thought might produce something useable.
- One alternative: straightforward map. Big & ugly, but not an inner loop operation.
- Another alternative: accept that, like callbacks, parameters are inherently solver-specific and simply act as relay agent for a parameter object.
- Neither is particularly satisfying, but a long history of attempts to identify common parameters is not encouraging.
- And it's the differences between solvers that allow an individual solver to excel on a particular class of problems. Somehow defeats the purpose to suppress that.

Callbacks: Except for very generic functionality, inherently solver-specific because requires interrogating solver object supplied as parameter. So one possible architecture is to provide one generic — abort — to be applied at each available callback point, plus a solver-specific object that user loads & returns. Osi simply passes this object between solver shim & user.

Alternate model: identify a union of solver capabilities. Osi hands out to user an object with function pointer slots for all identified callback points. This object is presented to solver shim, which installs all it can support. Still, each function must know what to do with parameters when called, so solver-specific. Additional complexity with no real gain.

Communication within Osi object:

- modules will need the same ability as user to request objects supporting particular interfaces
- pull model when module "takes over" functionality.
- solver loads in ↪ takes over model ↪ basis maintenance funtions. Request pointers to existing model ↪ basis (factorisation) objects ↪ pulls required information, forces unload of preexisting modules.

Modules:

- we have this notion of relatively large-grain functional blocks oriented around functional clusters
- large grain ⇒ relatively few pieces; practical to manage through a map.
- define semantics of interface for a module as a pure virtual abstract class.
- now have ability to add/remove/replace modules, and interrogate capababilities
- we have (or can provide) an upgrade path by versioning the modules.
- load a module ⇒ return pointer to object.