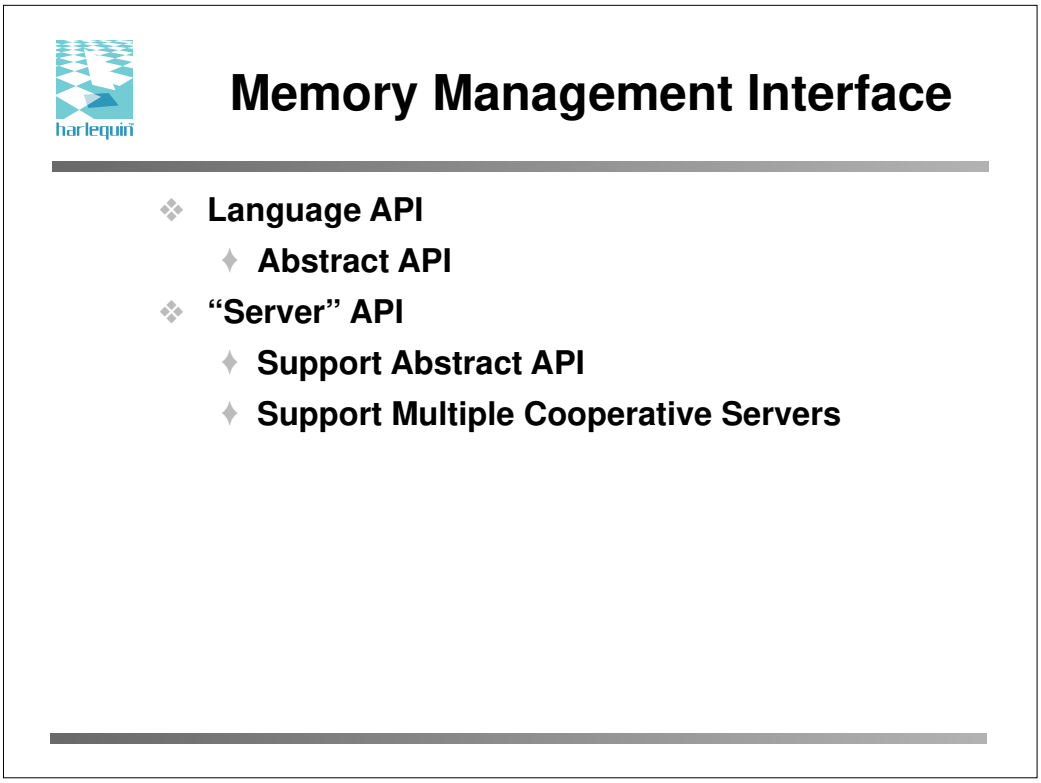
A Proposal for a Standard Memory Management Interface

The Harlequin Memory Management Group

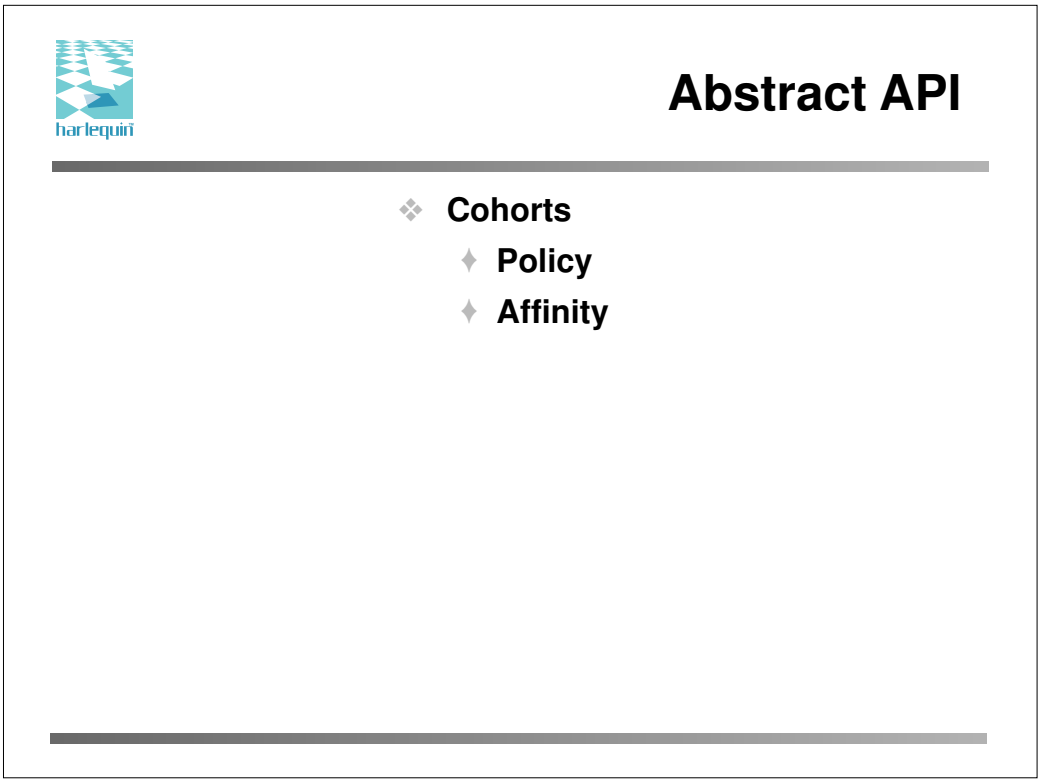
- The Harlequin Memory Management Group is chartered with designing, developing, and maintaining the memory management systems of all of Harlequin's products, which include OEM PostScript emulators and various programming language development and runtime systems including ML, Lisp, and Dylan.
- The group can be reached at mm@harlequin.com

Memory Management Interface



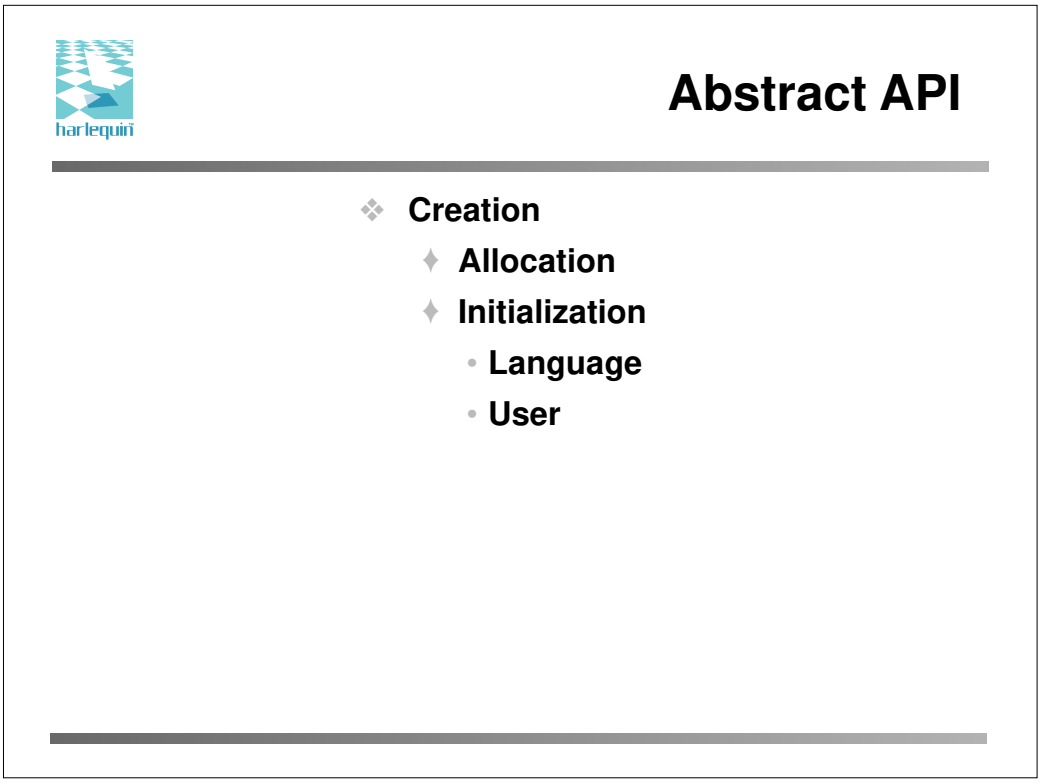
- There is no well-defined memory-management library API which would allow programmers to easily choose the best memory management implementation for their application.
- Some languages allow replacement of their memory management functions, but usually only the program API is specified, hence replacement of the entire program interface is required.
- Few languages support multiple memory management policies within a single program. Those that do use proprietary memory management policies.
- We believe that the design of an abstract program API is a prerequisite to the design of a “server” API and eventually an API that would permit multiple cooperating memory “servers”. If the interface is simple yet powerful enough to encompass most memory management systems, it stands a good chance of being widely adopted.

Abstract API



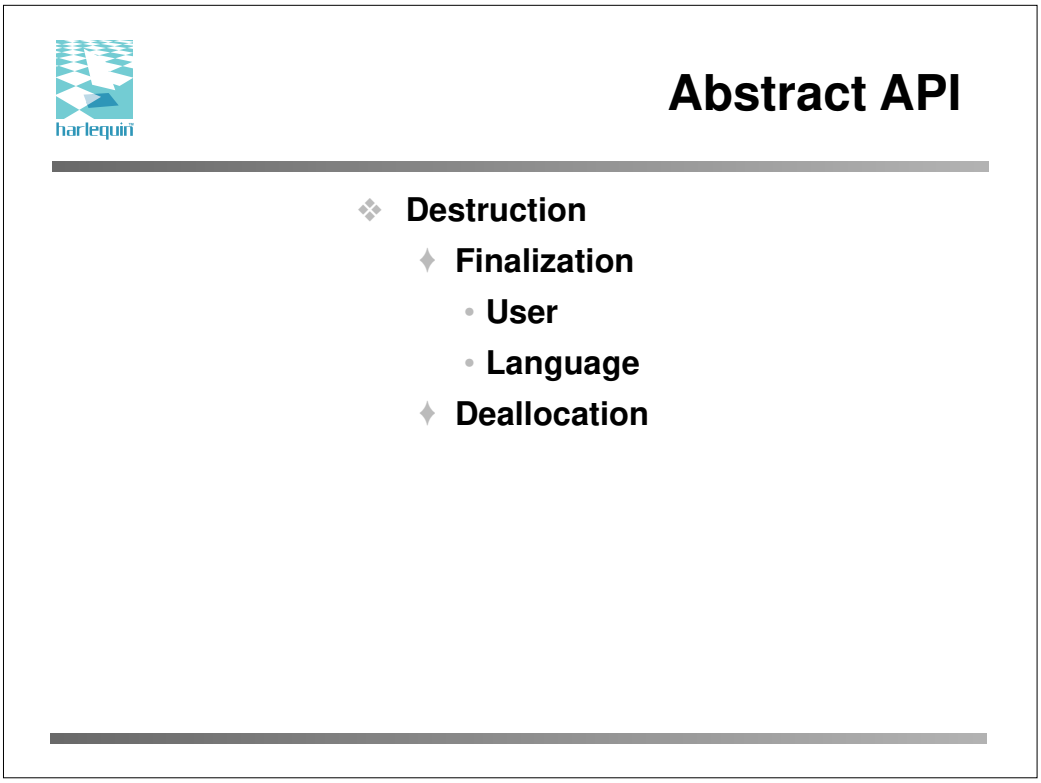
- There are two important attributes of memory management implementations: Policy and Affinity.
- Memory managers may allow a choice of management policy such as automatic, manual, or stack, to match performance and reliability requirements.
- Memory managers may allow objects to be grouped and managed according to any affinity, such as lifetime, type, or interconnection, for potential performance gains.
- We propose encapsulating these two attributes in a single concept, the cohort. At object allocation time, specifying a cohort would cause the object to be managed according to the cohort's policy and affinity attributes.

Abstract API



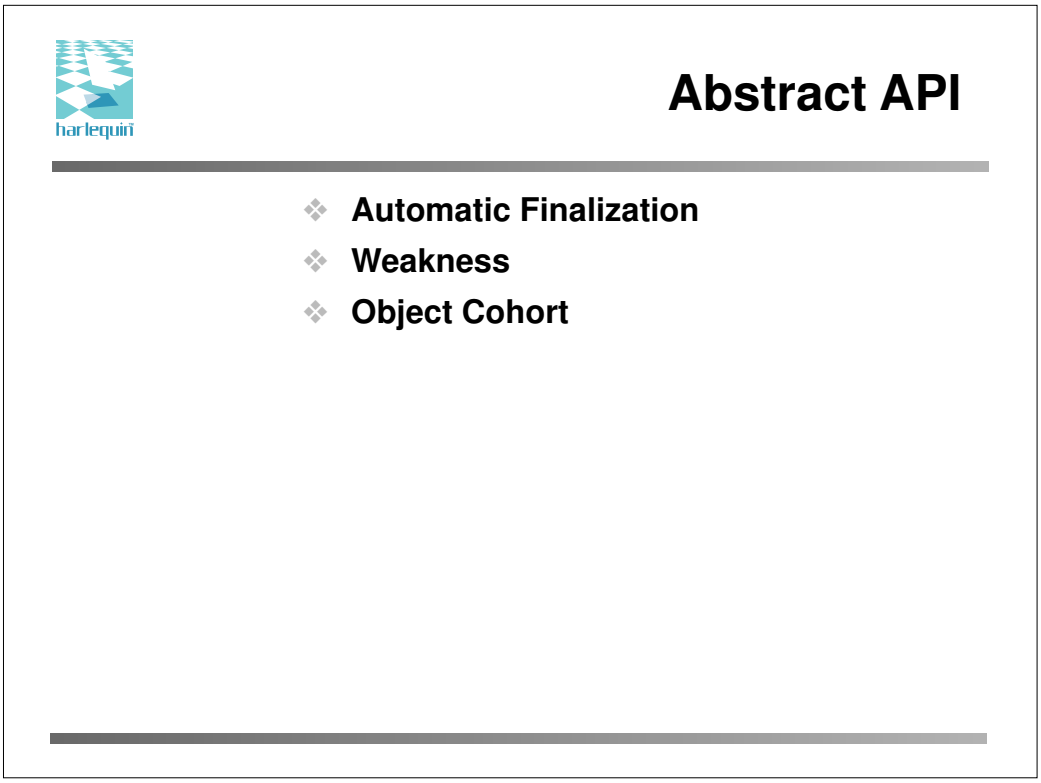
- Object creation consists of allocating the memory resources for the object and performing any language- and program-specific initializations.
- The memory manager server API needs to support allocation in a particular cohort. Some memory management implementations will also need to be made aware of when an object has been made “safe” by the language-specific initializations. We do not believe the memory manager need be aware of program-specific initializations.

Abstract API



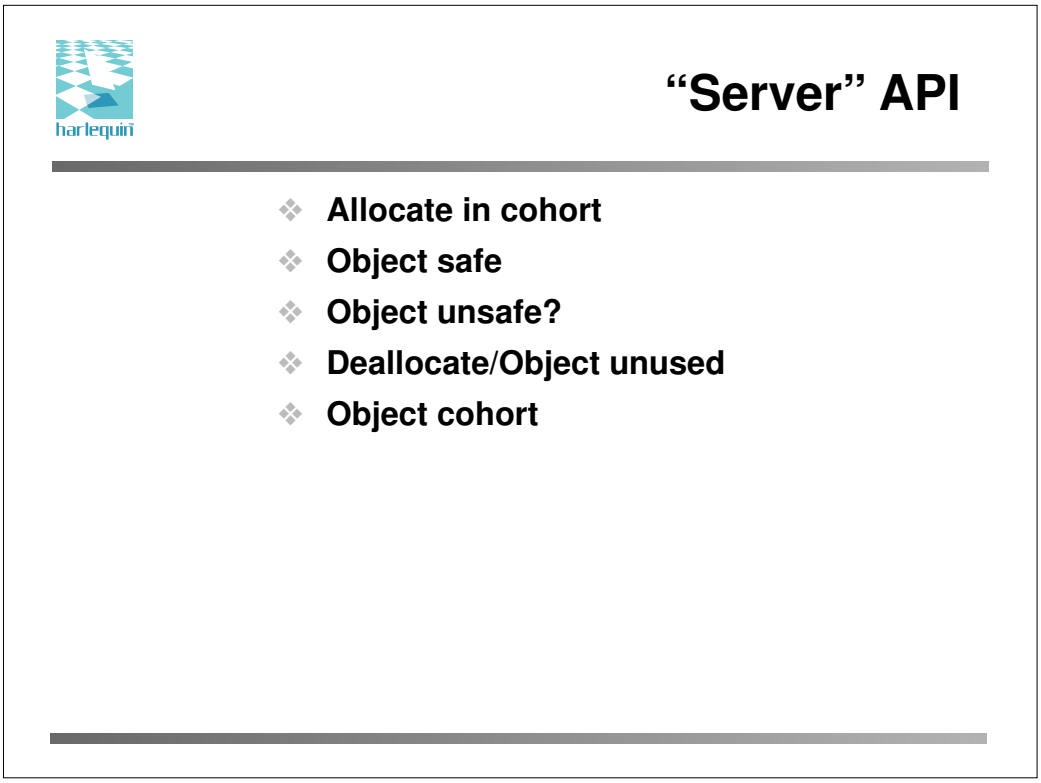
- Object destruction consists of performing the program- and language-specific finalizations and deallocating the memory resources of the object.
- Any of the steps of destruction can cause program errors. The latter two can cause “out-of-language” errors. In an automatic memory manager, the latter two steps can be delayed until the object is known to be dead, to prevent such errors. However, the first step cannot be so delayed, as it may have dependencies the memory manager is unaware of.
- The implementation of destruction is a tradeoff between (perceived) efficiency and safety.
- The memory manager server API needs to support deallocation of an object. It should not be up to the programmer to remember the cohort the object was allocated in — the server should be able to determine the cohort from the object, if necessary. This presents a particular challenge for the multiple-server case.

Abstract API



- We believe it is important to distinguish “automatic finalization” from finalization. C++ and Java take opposite stances on whether finalization should always be automatic or not in an automatic memory manager. The memory manager server API will need to support both policies.
- Weak references have even less agreement as to their semantics. We don’t propose to standardize them at this time.
- We do propose that the abstract interface support discovering the cohort of an existing object so that additional, related objects can be allocated in the same cohort. The memory manager server API needs such functionality itself, so that it can direct a destroy operation to the correct cohort.

“Server” API



- By examining ou abstract memory management interface, we can derive what a standard “server” interface might look like.
- Does deallocate take a cohort, use object cohort?

Inter-”Server” API

Inter-”Server” API

- Can an standard interface be developed that would allow two (or more) different memory management servers to be used by the same program (e.g., possibly from competing vendors)?