NetKernel : Distributed Cache or Distributed Representation State?

Peter Rodgers
2009-11-15

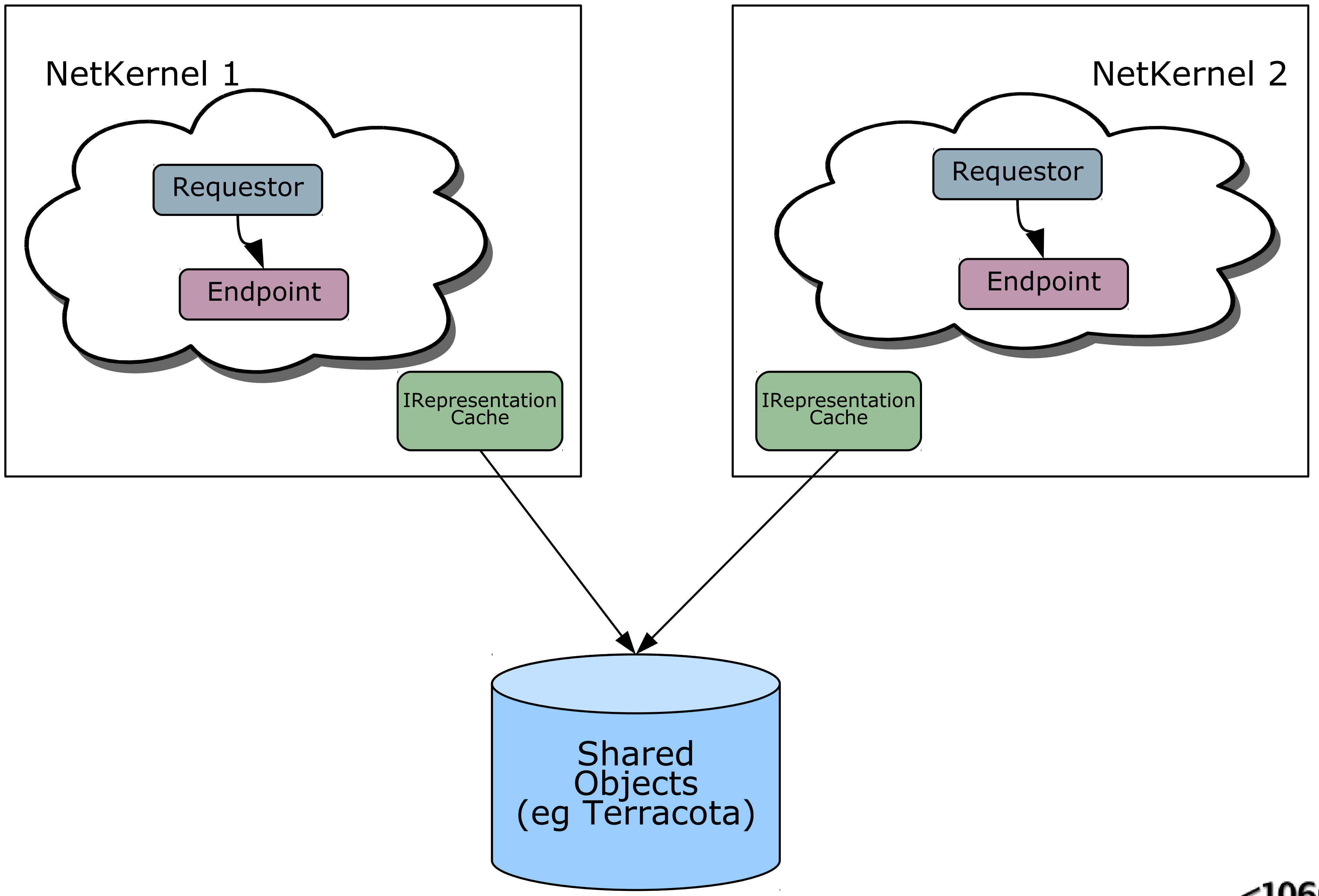
Can I distribute the NetKernel Cache?

- The representation I have generated has a lifetime that is potentially very long and could be shared across different nodes in my architecture.
  - Do I use an L1/L2/.../LN tiered NK cache?
  - Do I distribute the cache?
- Answer: The above are possible, we'll show how, but there is a better “ROC way” of looking at this.

Howto Roll your own NetKernel Cache

- Has a simple interface IRepresentationCache
- Can supply your own implementation to the Kernel (see embedding tutorial)
- Has to be very efficient since it is interrogated for nearly every request.
- To minimize footprint need to understand the spacial context (superstack scope etc) – the NK cache has some very fancy internal tricks for determining scope overlap (it caches one thing for many request contexts)
- Could create a configurable cache in which a known set of resources are placed in a distributed object container (like Terracota). You'd then have transparent distributed cache. But this is potentially clumsy and is a net global system cost for something that is probably of localized architectural benefit (see next slide)

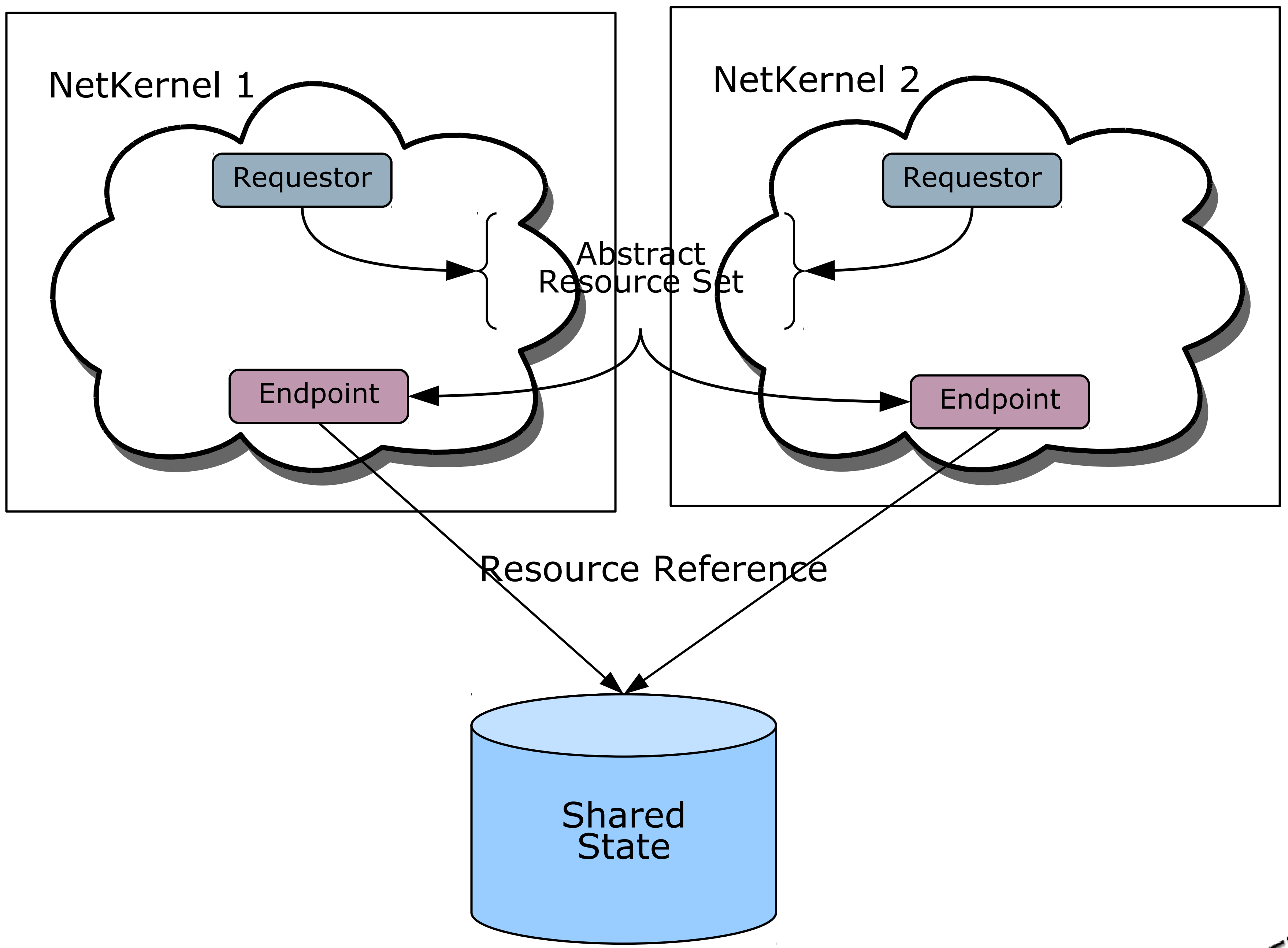
Shared Distributed Cache



Share Resource State

- Isn't what you really want “Shared Resource State”.
- That is, as long as all nodes see the same state consistently for the same resource identifier, that state can be either remotely accessed or locally cached.
- Next slide shows the ROC diagram view. Essentially we need to think of our abstract set as distributed...

Distributed Resource Set



How to do this?

- The endpoint responsible for access to the abstract set needs to do a couple of things
  - It should be able to connect to a common persistence mechanism (file, db, distributed objects, etc etc)
  - It should implement a fallback processing pattern in which it first attempts to SOURCE from the external state, if not present it generates state locally and then externally persists (ie for the cluster - for consistency it might be easiest to do this with RDBMS since it has the necessary locking)
  - The local representation is locally cacheable in the NK cache too but it should have a user-assigned expiry function (see next slide)

Programmable Cache Control

```
• INKFResponse.setExpiry(
        INKFResponse.EXPIRY_FUNCTION,
        INKFExpiryFunction
  )

  INKFExpiryFunction.isExpired(long now)
```

Optimal Balance
Distributed State / Local Caching

- User expiry function can be very lightweight – is the only external state monitor required to maintain local cached performance with cluster consistency.
- Can also attach golden threads for system/application-wide cache control.
- Solution gives distributed state, long term persistence (even after node restarts) and minimal external latency. Plus it allows local NK cache to self-balance.