CORBA

features for large-scale application development

Dr Duncan Grisby
dgrisby@apasphere.com

`www.grisby.org/presentations/`

Outline

1. Introduction
2. What is CORBA?
3. Object model
4. Object references
5. Simple object activation
6. Database access
7. Stateless adapters

About me

- BA and PhD at the University of Cambridge Computer Laboratory.
- Worked at AT&T Laboratories Cambridge until its closure in 2002.
- Founder of Apasphere Ltd. Apasphere
  - Consultancy, omniORB commercial support.
- Co-founder of Tideway Systems Ltd. TIDEWAY
  - Tools to understand distributed applications.
- Lead developer of omniORB.

What is CORBA?

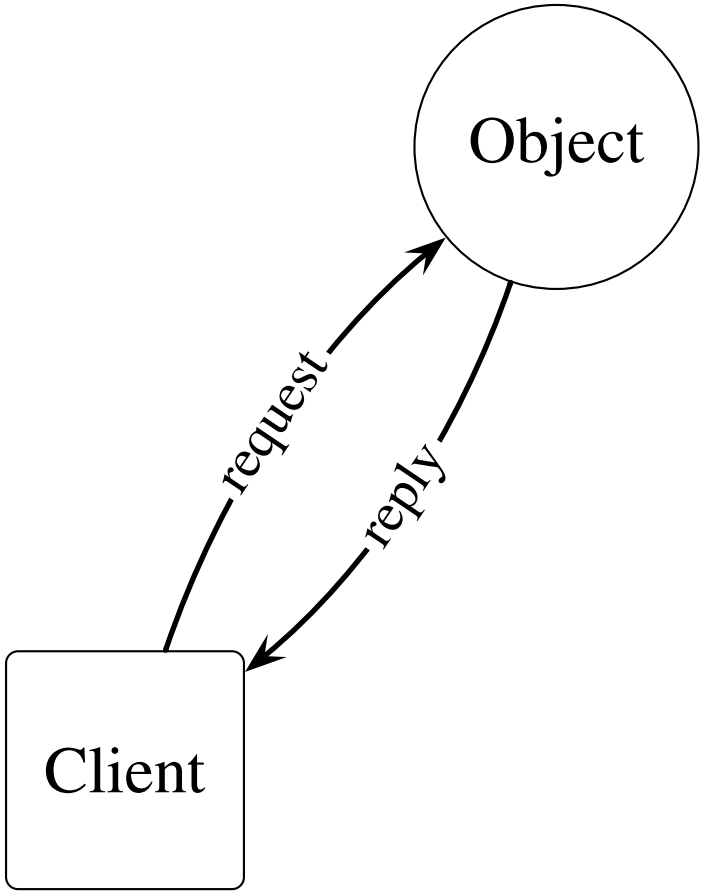
Common Object Request Broker Architecture.

- i.e. a common architecture for object request brokers.
- A framework for building *object oriented* distributed systems.
- Cross-platform.
- Language neutral.
- An extensive open standard, defined by the Object Management Group.
  - `www.omg.org`

Resources

- ‘Advanced CORBA Programming with C++’
  by Michi Henning and Steve Vinoski.
- CORBA 2.6 specification.
  `www.omg.org/cgi-bin/doc?formal/01-12-01`
- Python language mapping specification.
  `www.omg.org/cgi-bin/doc?formal/02-11-05`
- omniORB manual.
  `omniorb.sourceforge.net/docs/`
- `www.grisby.org/presentations/`

Object model



- A classical object model
  - the client sends request messages to the object; the object sends replies back.
- The client does not care where the object is
  - because the ORB deals with it.
- The client knows what messages it can send, because the object has an *interface*
  - specified in CORBA IDL.
- What is an object?…

Object model

- Often, a CORBA object is simply a programming language object which is remotely accessible.
- But really, an object is a *virtual* entity.
- In general, an object's existence may be independent of:
  - Clients holding references
  - References elsewhere
  - Operation invocations
  - Implementation objects (servants)
  - Server processes

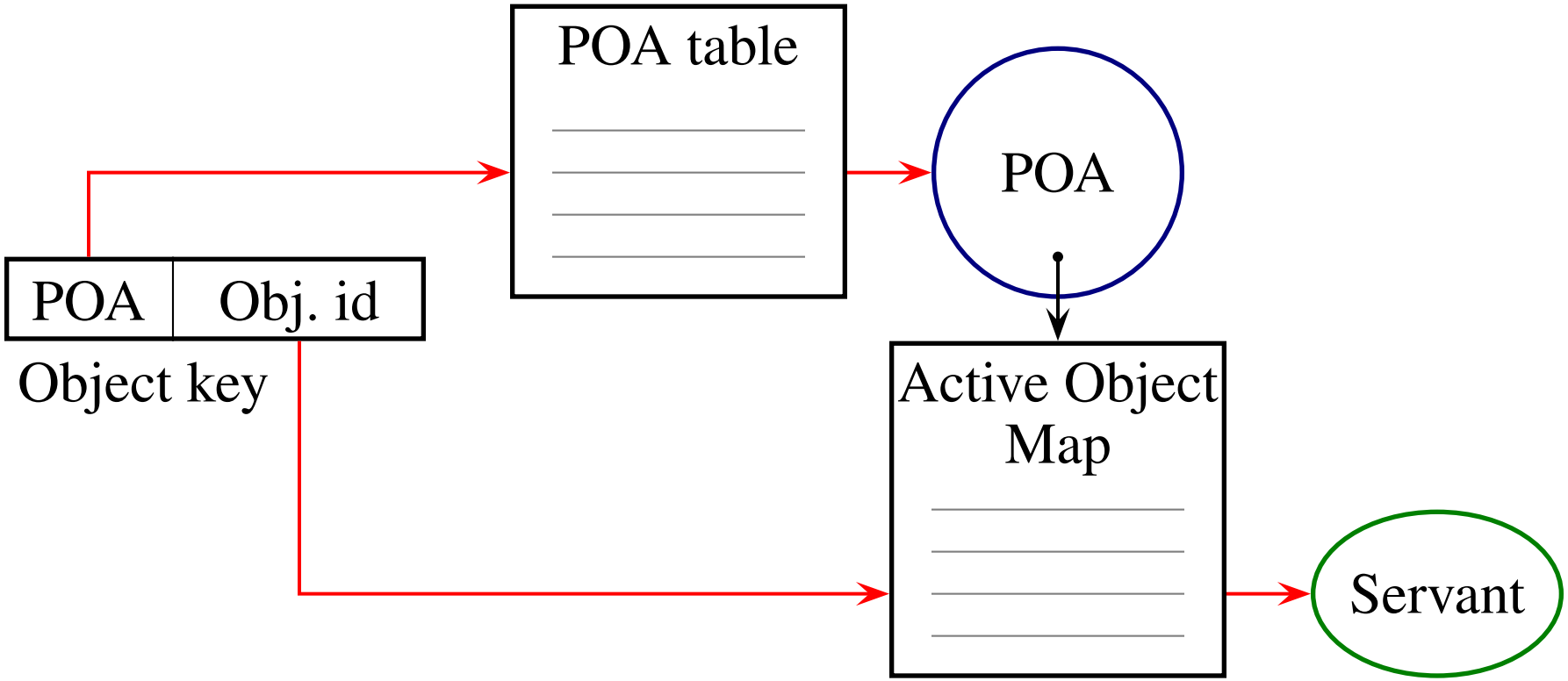
Object references

- Clients access objects through *object references*.
- An object reference contains sufficient information to locate the object.
- The object may not exist
  - at the moment
  - ever.
- Refers to a single object.
- An object may have many references to it.
- Analogous to a pointer in C++.

Inside object references

- Look inside a stringified object reference with omniORB's catior or similar tools.
- IOR contains:
  - Type id.
  - IIOP version.
  - Host and port for the server.
  - **Object key**.
  - Various other things.
- Object keys are opaque to clients.
- *Object adapters* map object keys to servants.
- The Portable Object Adapter (POA) is the standard object adapter.

Simple dispatch

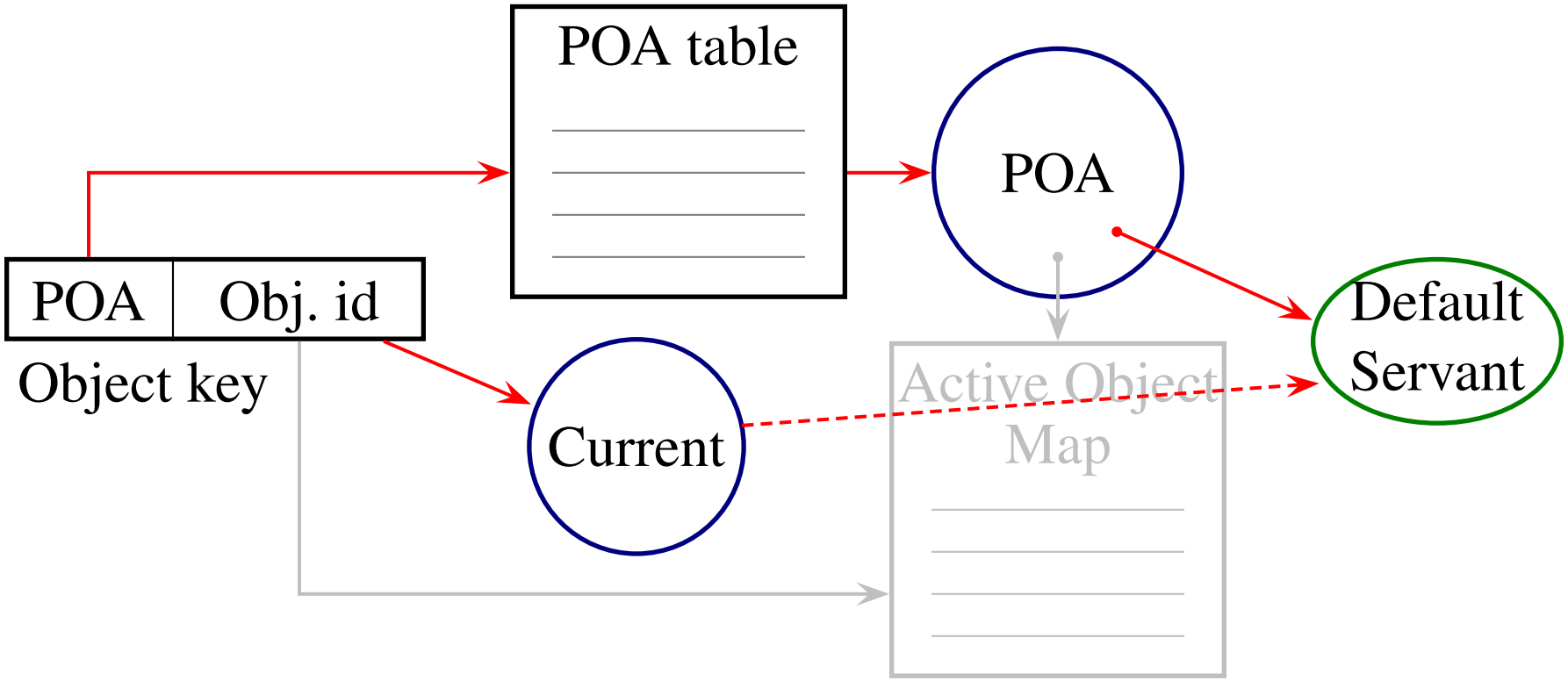


- Activated object found in the POA's Active Object Map (AOM).

Simple dispatch

- Simple dispatch with individual activated objects is good for many things.
  - Easy to set up.
  - Easy to understand.
  - Efficient.
- But imagine you have a relational database with millions of rows.
  - and you want each row to be a CORBA object.
- A ‘default servant’ can help…

Default servant

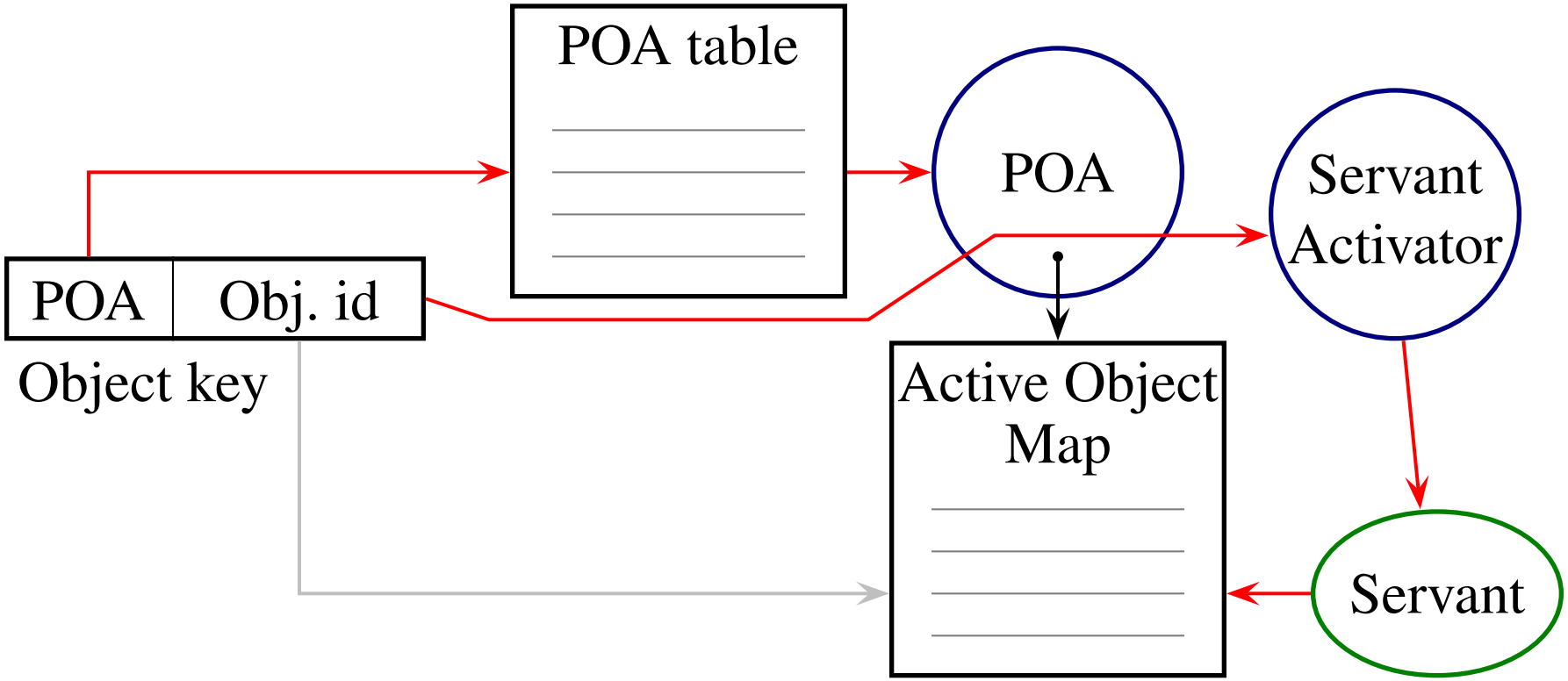


- No AOM, or id not in AOM.
- Current object knows what id was requested.

Default servant

- The default servant is used for objects not found in the AOM.
- Ask the Current object for the current object id, (or object reference, or servant).
  - Current is always available, but most useful here.
- Common pattern is to use database primary keys as object ids, then do a database lookup on each CORBA call.
- What if you want to cache database records?
  - Use a Servant Activator…

Servant Activator



- id not in AOM.
- Servant Activator *incarnates* servant.

Servant Activator

- Upon a request for an id not in the AOM, the Servant Activator *incarnates* a servant.
- Once incarnated, it is in the AOM.
  - Future calls go directly to the servant.
- Later on, a timeout or other event can deactivate the object.
  - The Servant Activator *etherealizes* it.
- POA hides lots of nasty concurrency issues.
- Alternative is a Servant Locator.
  - preinvoke / postinvoke called for every call.
  - Locator maintains its own set of servants.

Stateless adapting proxies

- Sometimes want to ‘adapt’ one interface to another.
- Notice that object id can have arbitrary contents.
- Store object reference to proxied object in object id of proxy.
- Use default servant to retrieve object reference and make proxied call.
- No state in the proxies.
- Can use for non-adapting proxies too, e.g. firewall traversal.

Other POA features

- Single / main thread policies.
- Transient policy to guarantee id uniqueness.
- Multiple activations for a single servant.
- State management: active, holding, discarding.
- Adapter Activators.

Servant Activator example

```
import CORBA, PortableServer__POA
from PortableServer import USER_ID, RETAIN
from PortableServer import USE_SERVANT_MANAGER, PERSISTENT

servantList = []

class ServantActivator_i (PortableServer__POA.
                          ServantActivator):
    def incarnate(self, oid, poa):
        servant = DatabaseServant(oid)
        servantList.append(servant)
        return servant

    def etherealize(self, oid, poa, servant,
                    cleanup, remaining):
        servant.flushState()
```

Servant Activator example

```
class Scavenger (threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.setDaemon(1)

    def run(self):
        while 1:
            time.sleep(10)
            for s in servantList[:]:
                if s.inactive():
                    poa.deactivate_object(s.oid())
                    servantList.remove(s)
```

Servant Activator example

```
class DatabaseServant (Database__POA.RowObject):
    def __init__(self, oid):
        self._oid = oid
        self._use = 1
        # ... read state from database

    def inactive(self):
        if self._use:
            self._use = 0
            return 0
        else:
            return 1

    def flushState(self):
        # ... write state to database

    def readRecord(self):
        self._use = 1
        # ...
```

Servant Activator example

```
def main(argv):
    orb = CORBA.ORB_init(argv)
    rp  = orb.resolve_initial_references("RootPOA")
    poaManager = rp._get_the_POAManager()
    poaManager.activate()

    ps = [rp.create_id_assignment_policy(USER_ID),
          rp.create_servant_retention_policy(RETAIN),
          rp.create_request_processing_policy(USE_SERVANT_
          rp.create_lifespan_policy(PERSISTENT)]

    poa = rp.create_POA("DatabasePOA", poaManager, ps)
    sa  = ServantActivator_i()
    poa.set_servant_manager(sa._this())

    scavenger = Scavenger()
    scavenger.start()

    orb.run()
```