[simula . research laboratory]

Work group meeting no. 3
- Tuning using policies

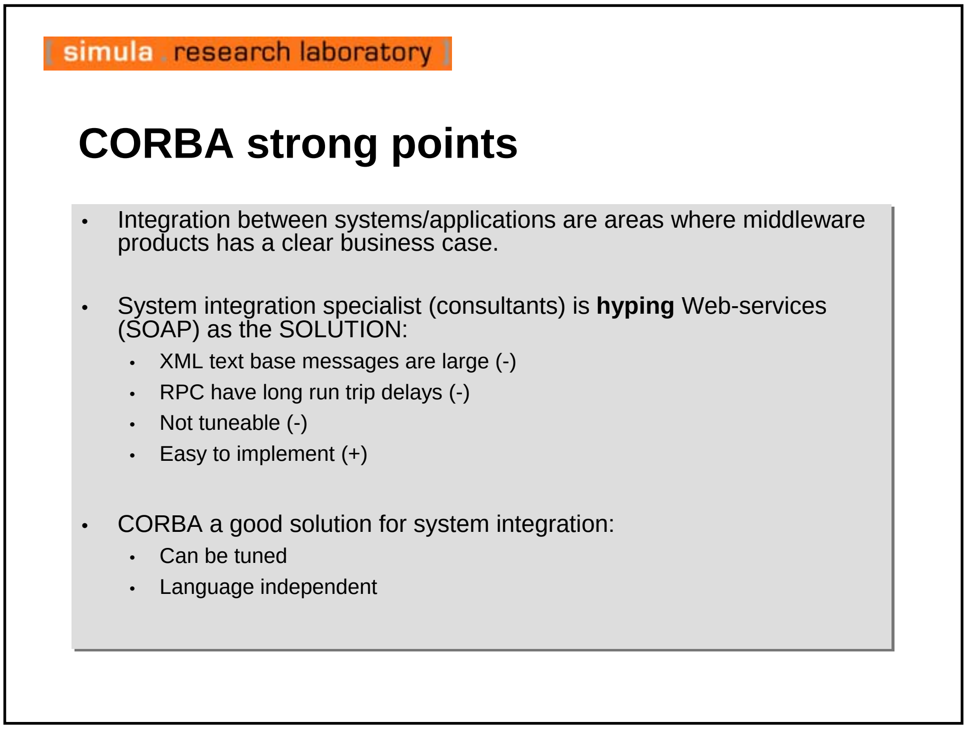
INF5040 (Distributed systems)

Name: Sten L. Amundsen
Date: 16 September 2004

e-mail: stena@simula.no

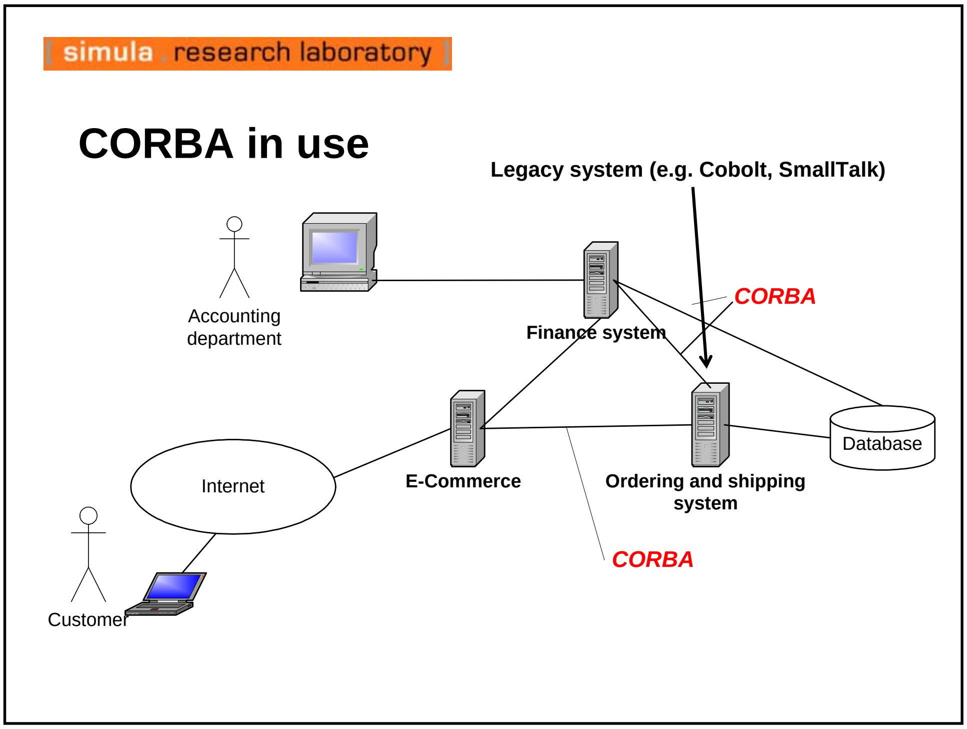
Agenda

- Tuning using POA policies and POA architecture
- Example: Concurrency model
- Example: Data objects



CORBA strong points

- Integration between systems/applications are areas where middleware products has a clear business case.
- System integration specialist (consultants) is **hyping** Web-services (SOAP) as the SOLUTION:
  - XML text base messages are large (-)
  - RPC have long run trip delays (-)
  - Not tuneable (-)
  - Easy to implement (+)
- CORBA a good solution for system integration:
  - Can be tuned
  - Language independent



CORBA in use

Tuning possibilities

- One can tune both:
  - ORB
  - POA

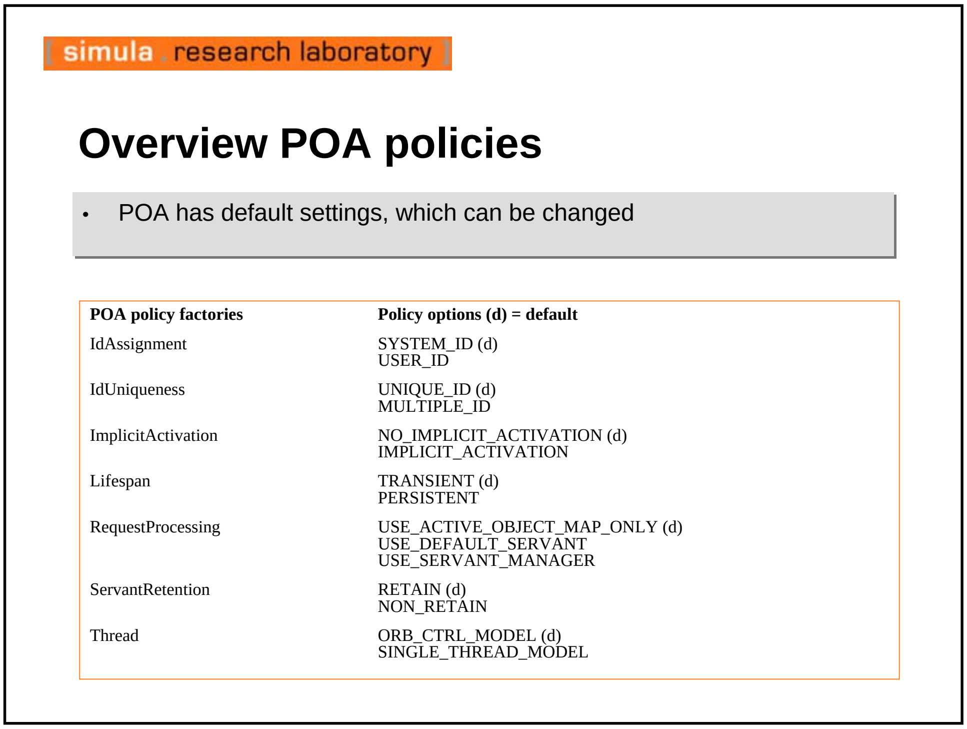
Example of tuning:

- **Concurrency model**
- **Data objects**
- Object lifetime
- Connection time-out
- End-point/protocol
- Trace level for protocols
- Retry mechanisms
- Message size
- Interceptors
- Automatic server shutdown
- etc**.**

Overview rootPOA policies

- The root POA has the following policy settings, which cannot be changed.

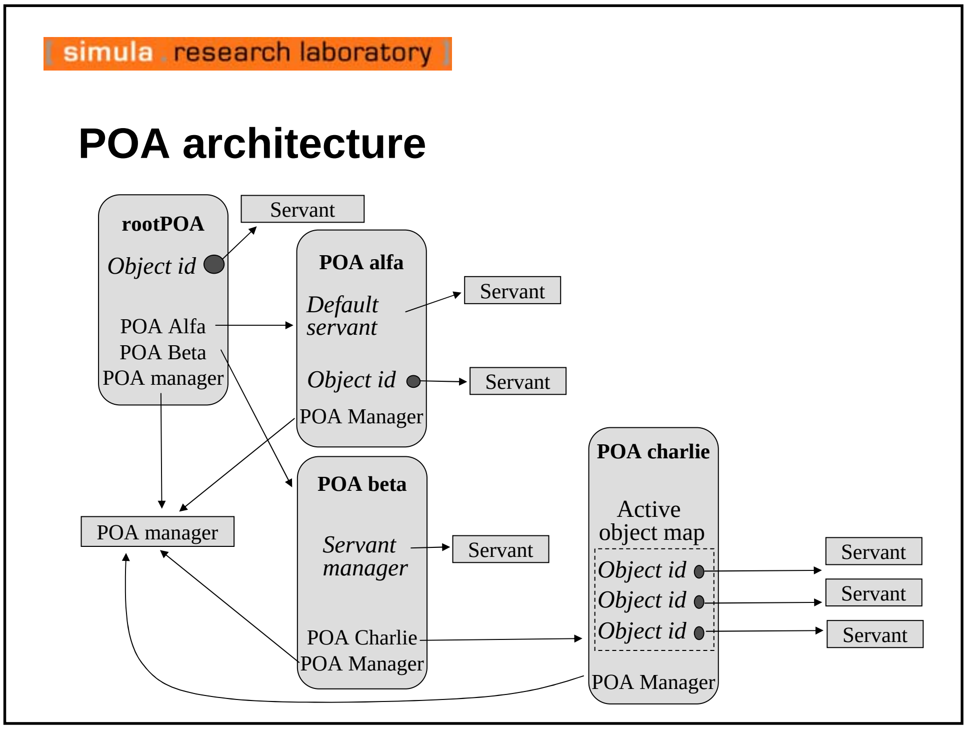
Policy	**Default setting**	**Comment**
IdAssignment	SYSTEM_ID	POA sets object id.
IdUniqueness	UNIQUE_ID	Each object id is uniquely mapped to servant.
ImplicitActivation	IMPLICIT_ACTIVATION	POA allocated object id when servant registered with POA
Lifespan	TRANSIENT	Non-persistent objects
RequestProcessing	USE_ACTIVE_OBJECT_MAP_ONLY	POA routes requests.
ServantRetention	RETAIN	Objects reference are kept in POA after processing request.
Thread	ORB_CTRL_MODEL	ORB handles threading (multi-thread)



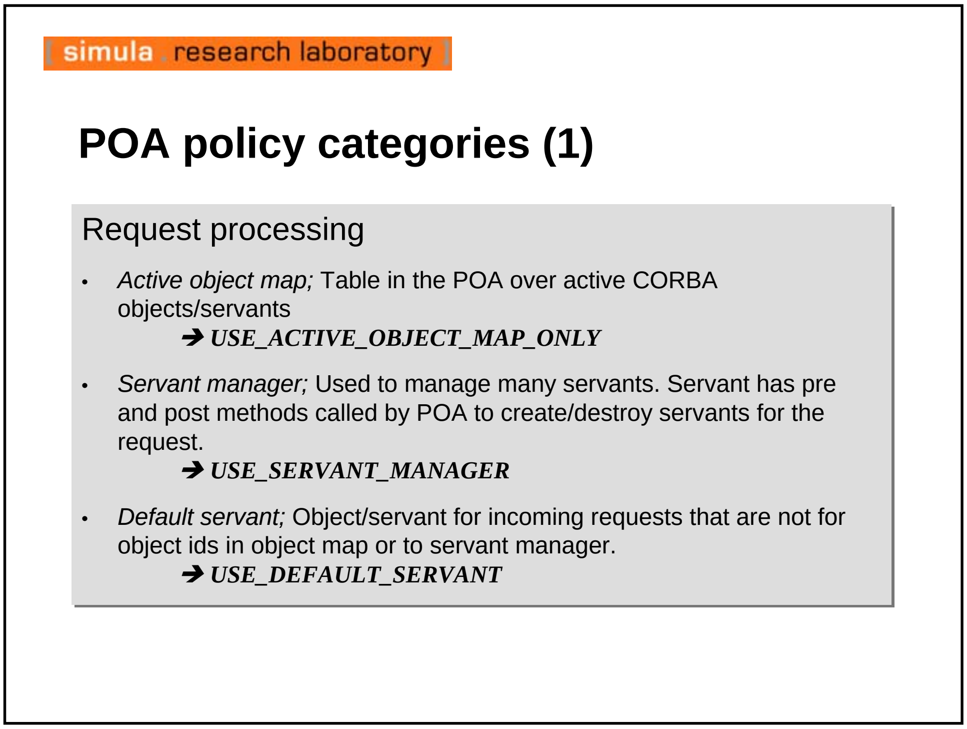
Overview POA policies

- POA has default settings, which can be changed

POA policy factories	Policy options (d) = default
IdAssignment	SYSTEM_ID (d) USER_ID
IdUniqueness	UNIQUE_ID (d) MULTIPLE_ID
ImplicitActivation	NO_IMPLICIT_ACTIVATION (d) IMPLICIT_ACTIVATION
Lifespan	TRANSIENT (d) PERSISTENT
RequestProcessing	USE_ACTIVE_OBJECT_MAP_ONLY (d) USE_DEFAULT_SERVANT USE_SERVANT_MANAGER
ServantRetention	RETAIN (d) NON_RETAIN
Thread	ORB_CTRL_MODEL (d) SINGLE_THREAD_MODEL



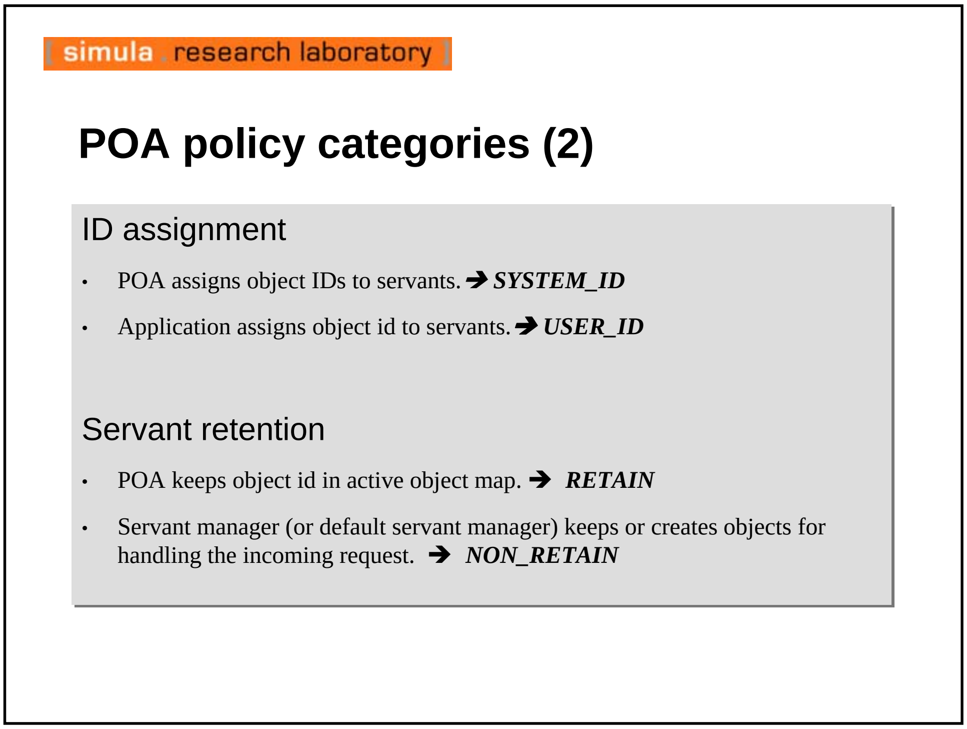
POA architecture



POA policy categories (1)

Request processing

- *Active object map;* Table in the POA over active CORBA objects/servants
  ➔ ***USE_ACTIVE_OBJECT_MAP_ONLY***
- *Servant manager;* Used to manage many servants. Servant has pre and post methods called by POA to create/destroy servants for the request.
  ➔ ***USE_SERVANT_MANAGER***
- *Default servant;* Object/servant for incoming requests that are not for object ids in object map or to servant manager.
  ➔ ***USE_DEFAULT_SERVANT***



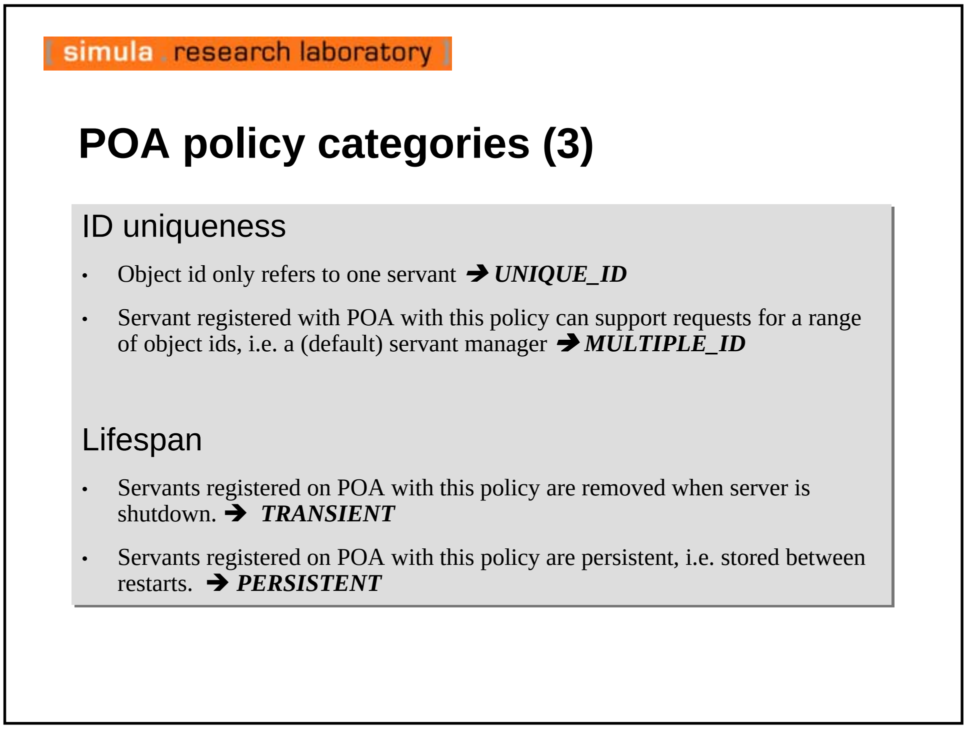
POA policy categories (2)

ID assignment

- POA assigns object IDs to servants.➔ ***SYSTEM_ID***
- Application assigns object id to servants.➔ ***USER_ID***

Servant retention

- POA keeps object id in active object map. ➔ ***RETAIN***
- Servant manager (or default servant manager) keeps or creates objects for handling the incoming request. ➔ ***NON_RETAIN***



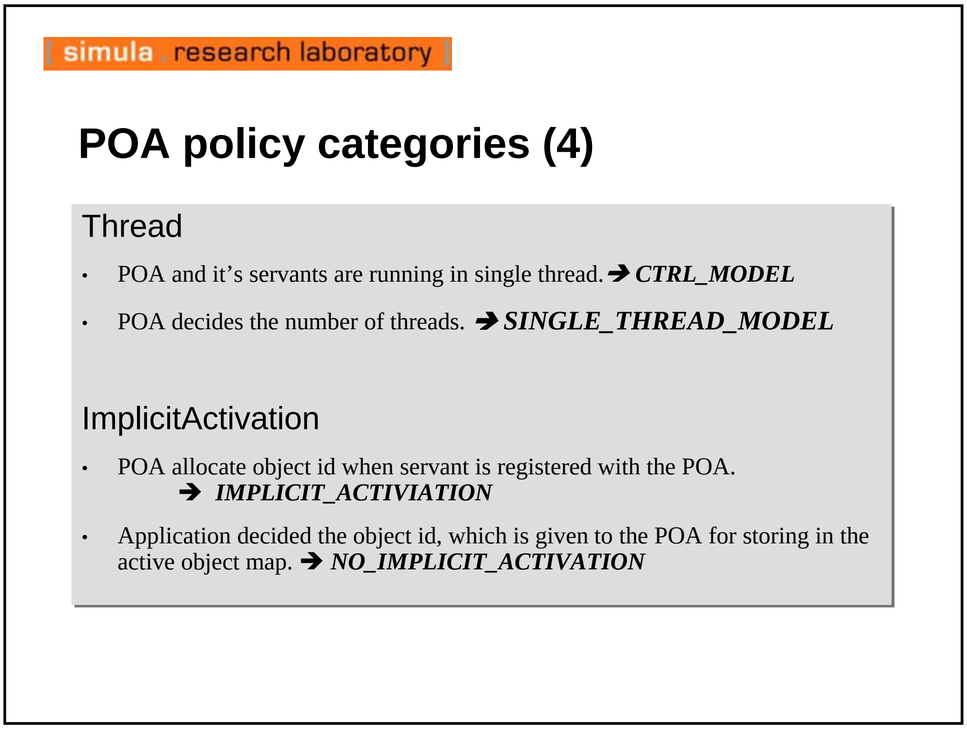
POA policy categories (3)

ID uniqueness

- Object id only refers to one servant ➔ ***UNIQUE_ID***
- Servant registered with POA with this policy can support requests for a range of object ids, i.e. a (default) servant manager ➔ ***MULTIPLE_ID***

Lifespan

- Servants registered on POA with this policy are removed when server is shutdown. ➔ ***TRANSIENT***
- Servants registered on POA with this policy are persistent, i.e. stored between restarts. ➔ ***PERSISTENT***



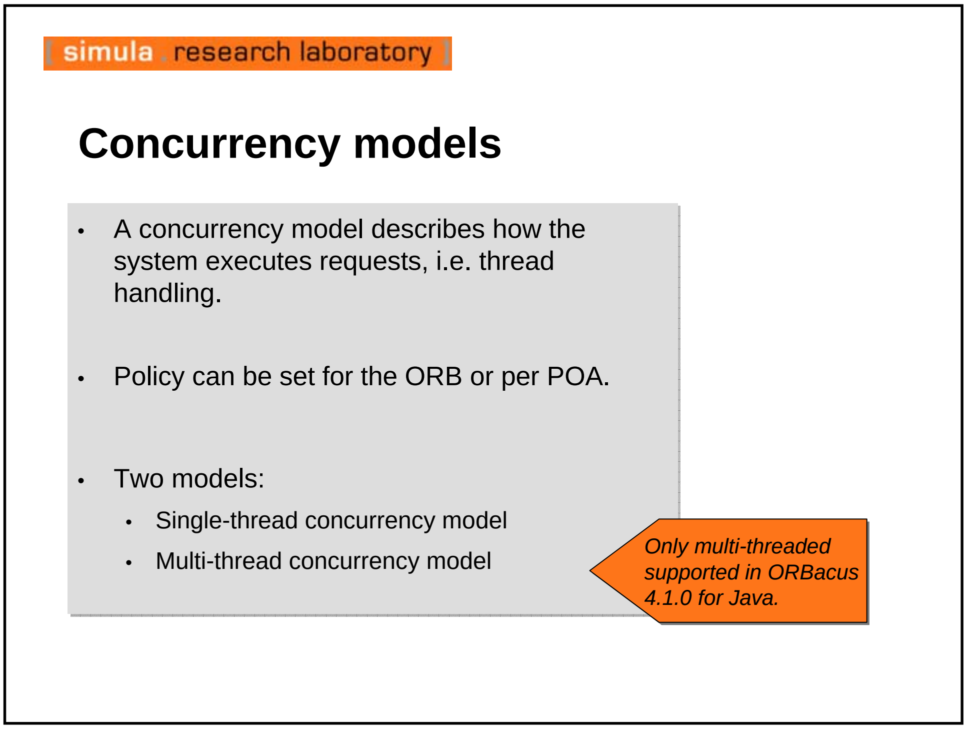
POA policy categories (4)

Thread

- POA and it’s servants are running in single thread.➔ ***CTRL_MODEL***
- POA decides the number of threads. ➔ ***SINGLE_THREAD_MODEL***

ImplicitActivation

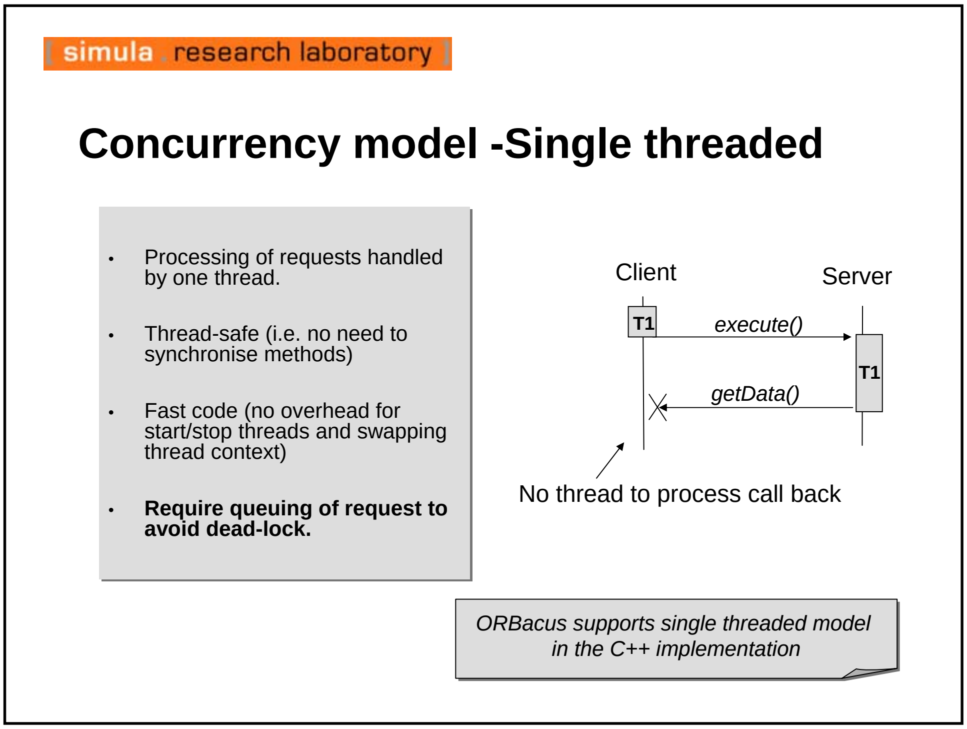
- POA allocate object id when servant is registered with the POA. ➔ ***IMPLICIT_ACTIVIATION***
- Application decided the object id, which is given to the POA for storing in the active object map. ➔ ***NO_IMPLICIT_ACTIVATION***



Concurrency models

- A concurrency model describes how the system executes requests, i.e. thread handling.
- Policy can be set for the ORB or per POA.
- Two models:
  - Single-thread concurrency model
  - Multi-thread concurrency model

Only multi-threaded supported in ORBacus 4.1.0 for Java.



Concurrency model -Single threaded

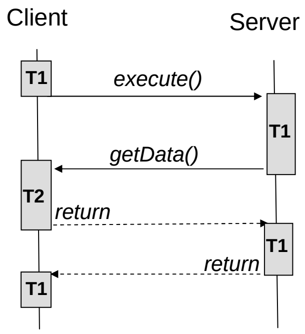
- Processing of requests handled by one thread.
- Thread-safe (i.e. no need to synchronise methods)
- Fast code (no overhead for start/stop threads and swapping thread context)
- **Require queuing of request to avoid dead-lock.**

No thread to process call back

ORBacus supports single threaded model in the C++ implementation

Concurrency model -Multi threaded

- One or more dedicated threads per request.
- Easy to develop nested processes including call backs
- Dead-lock safe
- **Prudent to synchronisation problems of states and data.**
- **High number of threads results in slow code.**



Concurrency model -setting policy

- Multi-threaded default concurrency model.
- In ORBacus can tune further by setting:
  - Thread-per-client on the server side
  - Thread-per-request
  - Thread pool size
- Policy can be set for the POA using a policy object.
- For ORB properties uploaded from a config file or hard-coded.

Concurrency model -setting policy - ORB

- Example with hard-coded

```
java.util.Properties properties = System.getProperties();
properties.put(”ooc.orb.oa.conc_model”, ”thread_pool”);
properties.put(”ooc.orb.oa.thread_pool”, ”5”);

//start ORB in the JVM that has the new properties
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, properties )
```

Concurrency model -setting policy - POA

- Policy is an object.
- Set policy in POA when starting the POA or after:
  - ORB_CTRL_MODEL (d)
  - SINGLE_THREAD_MODEL
- Root-POA can not change concurrency model.

Operation on the POA class.

```
ThreadPolicy.create_thread_policy (ThreadPolicyValue value);
```

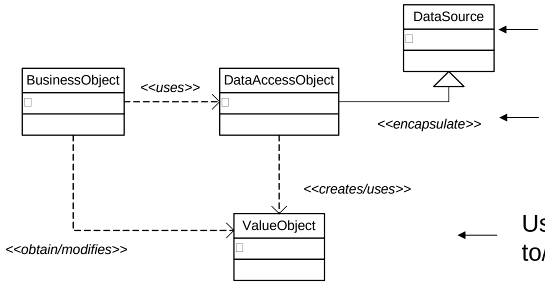
Data objects

- Data objects represents data from database/flat file.
- Large systems can't have large volume of data objects.

- POA policies can assist in automatic instantiating and removing data objects.
- POA policies of interest:
  - IdAssignments
  - ServerRetention
  - RequestProcessing

Data object – design pattern

- General solution is the design pattern Data Access Object (DAO).



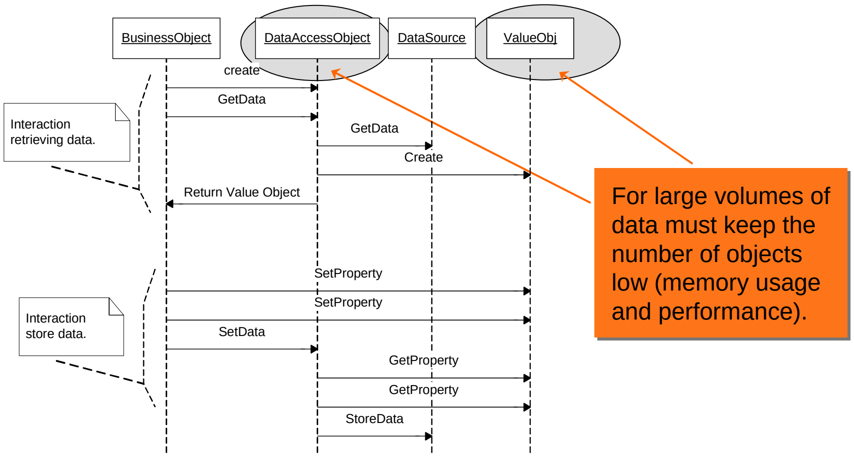
Represents database or flat file

Abstracts the data access implementation from the business logic.

Used to transfer the data to/from client.

simula . research laboratory

Data object –design pattern -sequence



For large volumes of data must keep the number of objects low (memory usage and performance).

simula . research laboratory

Data object – CORBA

- POA policies can be combined.
- Right combination give automatic generation and removing of objects in the DAO pattern.
- Clue is:
  - define an POA architecture
  - Set POA policies differently for business logic compared to GUI and data objects.

Exercise

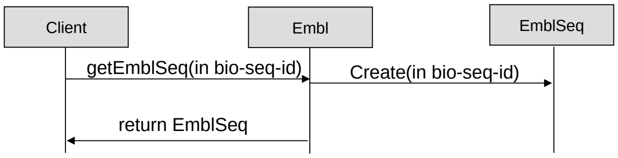
- Case from UiO. See paper: http://genomebiology.com/content/pdf/gb-2000-1-5-research0010.pdf
- Used CORBA together with a data model suitable for large databases for DNA and RNA sequences.

Write two slides with:

1. The POA architecture
2. Specified POA policies

Exercise - hints

- Client submit to CORBA server the id to the DNA (or RNA) sequence, which then invokes a factory object (or a manager) representing the database.
- Consider only the interaction between factory/manager (Embl class) and the data access object (EmblSeq class).
- Combine the two patters Factory and DAO.


Client
Embl
EmblSeq
getEmblSeq(in bio-seq-id)
Create(in bio-seq-id)
return EmblSeq