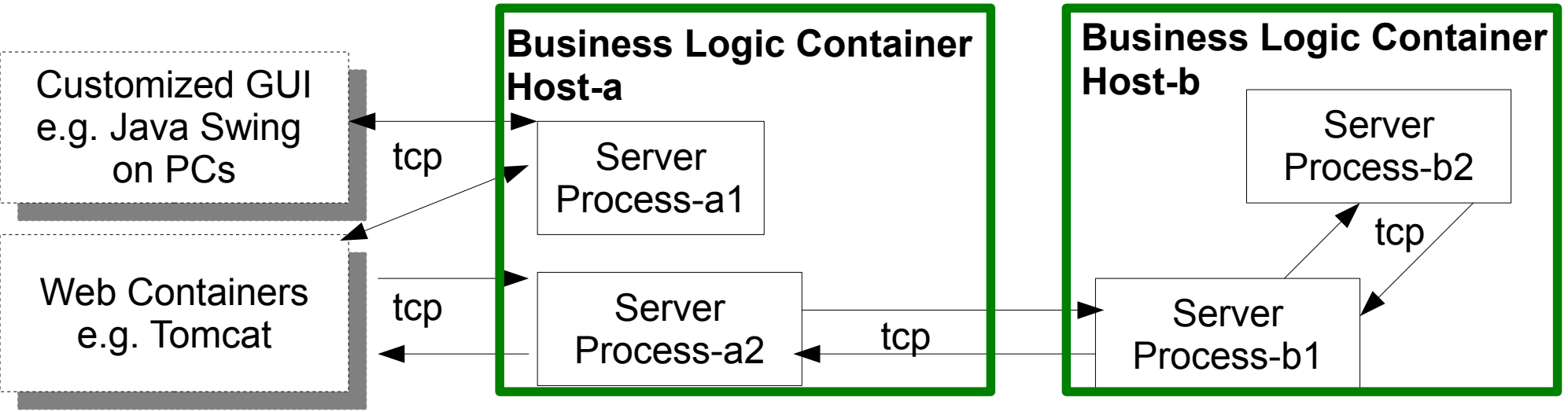
# Business Logics Container

- Architect, design & implement an Object Oriented Java multi-threaded container to process business transactions
- Dynamically instantiate  and execute Application Servant classes with business logics (similar to J2EE EJB Container)
- Each Application Servant class implements the standard Java abstract interfaces as defined by the container
- All message exchanges between the container processes are XML over TCP based

# Container Architecture Request/Response Flow

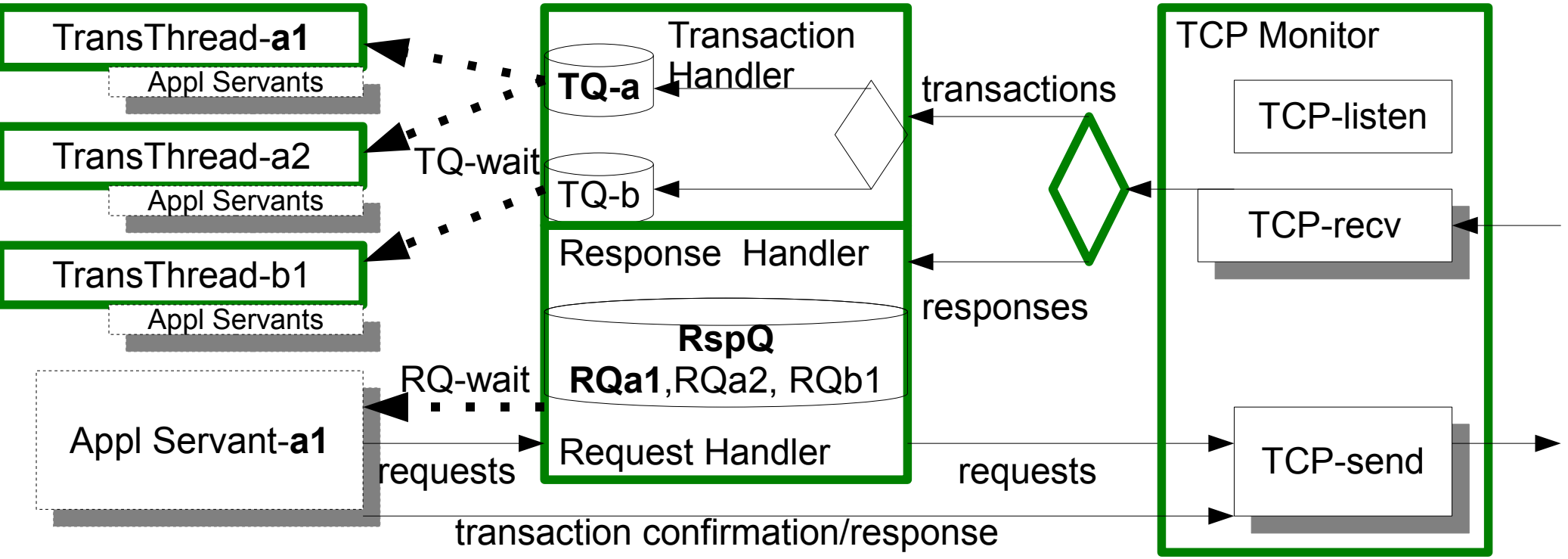



- **All message switchings among Server Processes are XML over TCP**
- **Request/response from/to Web container or other GUI clients is over TCP**
- **Physical machines boundary of the distributed Business Logic Containers is transparent to Server Processes by using TCP message switching**

Server Process-a1 has **self contained business logics** to finish web/GUI request/response

Server Process-a2, b1 & b2 collaborate as one unit of **distributed business logics** to finish a transaction request/response cycle
Process-a2 sends sub-request to Process-b1 which sends sub-request to process-b2
Process-b2 sends response to Process-b1, then Process-a2 to complete the transaction

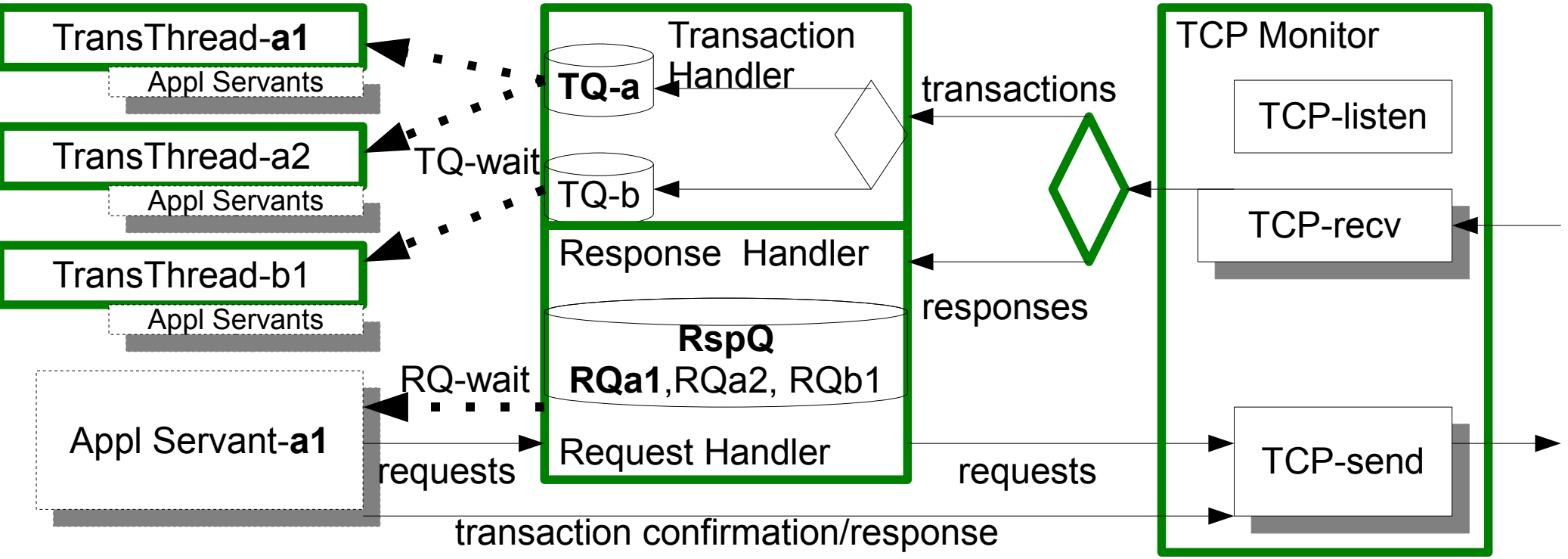
# Multi-threaded Transaction Server Components




Each Server Process uses the Business Logic Container components to:

- Instantiates multiple Transaction Threads (e.g. a1, a2 & b1)
- Instantiates a singleton Transaction/Response Handler to classify input TCP messages into either Transactions (TQs) or Responses (RspQ/RQa1)
- Instantiates a singleton TCP monitor which creates TCP listen thread as well as a shared pool of TCP-send/TCP-recv thread pairs, one for each TCP session

# Multi-threaded Transaction Server Components



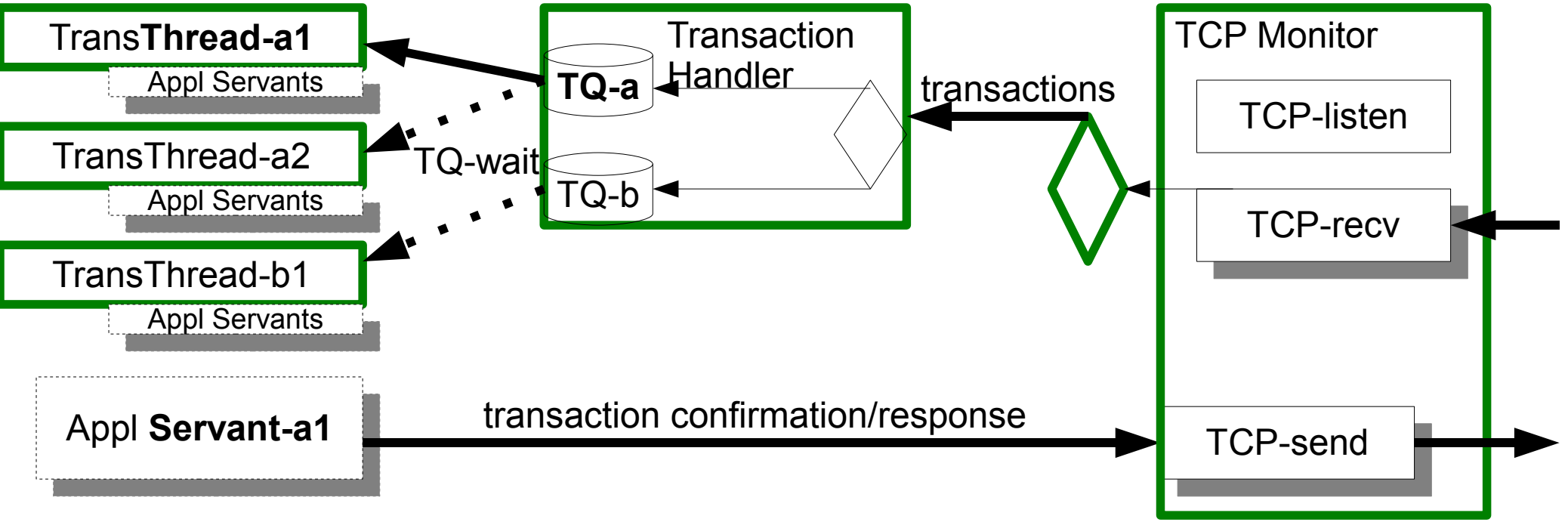

Each Server Process uses the Business Logic Container components to:

•(continue)

•Each Transaction Thread may dynamically instantiate Application Servant Classes per the incoming transaction type from TQs

•An Application Servant class may initiate further distributed requests/responses to chain linked other Server Processes to complete a business transaction

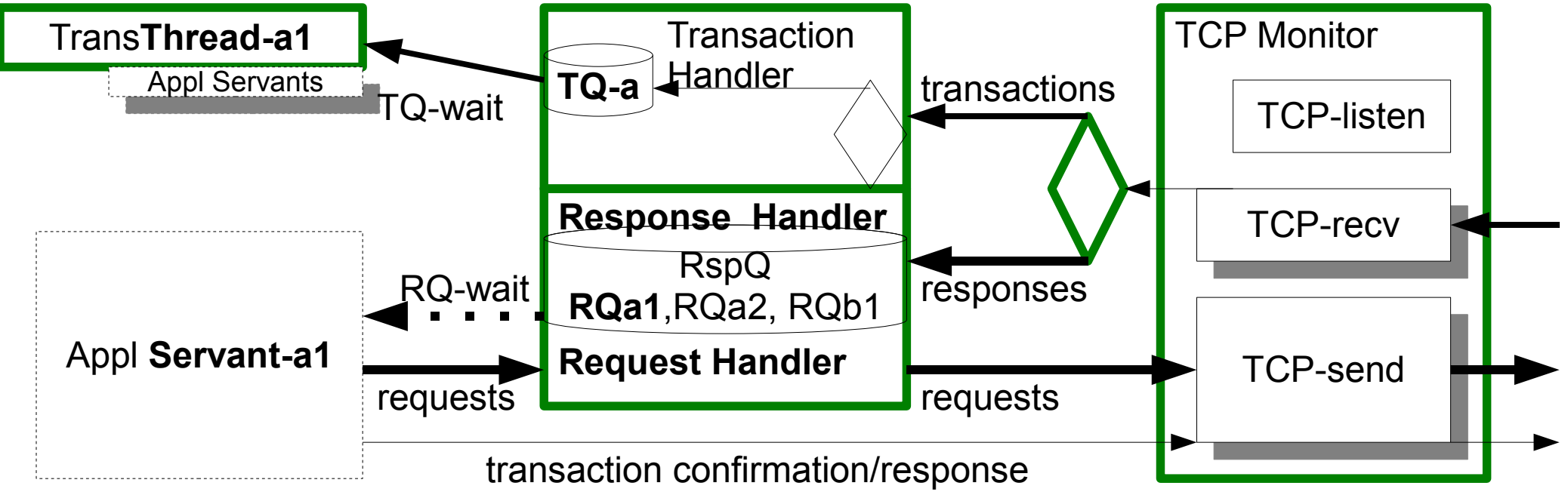
# Flow of Self Contained Business Logic




Assuming Application **Servant-a1** contains complete business logic

- TCP-recv receives a type-a transaction.
- Transaction Handler stores it into the Transaction Queue **TQ-a** (publisher)
- Both Trans**Thread-a1** & a2 (consumers) are waiting for transaction in **TQ-a**.
- Trans**Thread-a1** instantiates Appl **Serverant-a1** to match the incoming transaction
- Appl **Servant-a1** finishes the transaction processing by sending transaction confirmation/response out and returns control to the Trans**Thread-a1**
- Trans**Thread-a1** now ready to do TQ-wait on Queue **TQ-a** again for next transaction

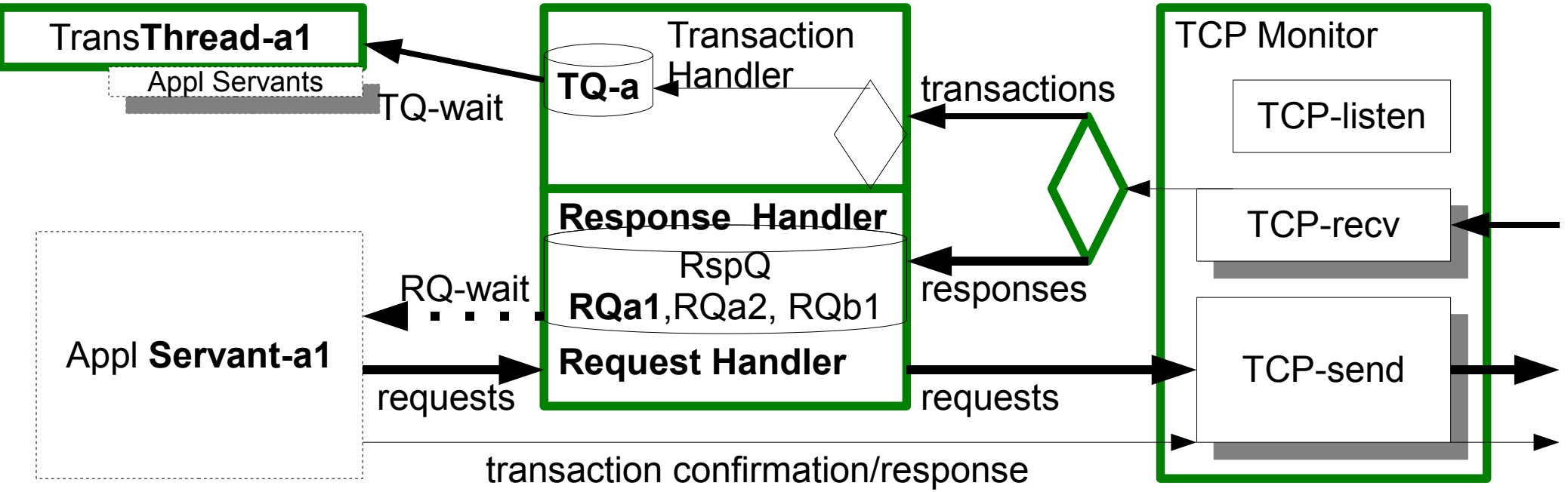
# Flow of Distributed Business Logics




Assuming the Appl **Servant-a1** would need to send request(s) to other server process(es) to finish the transaction

- In addition to the "Flow of Self Contained Business Logic", the Appl **Servant-a1** needs to use the Request/Response Handler for further distributed request/response
- Appl **Servant-a1** sends requests thru **Request Handler** & TCP-send to other process
- **Request Handler** instantiates the **RQa1** for **Servant-a1** to do RQ-wait

# Flow of Distributed Business Logics




(Continue)

- TCP-recv receives responses for **Response Handler** to store to **RQa1**
- Appl **Servant-a1** wakes up on **RQa1** and process responses
- Appl **Servant-a1** finishes the transaction processing by sending transaction confirmation/response out and returns control to the Trans**Thread-a1**
- Trans**Thread-a1** now ready to do TQ-wait on Queue **TQ-a** again for next transaction

# Business Logic Container Tools

- Recording major and/or debugging **event logs** by configuring Apache Log4j
- Recording different levels of **TCP data traces** by configuring Apache Log4j
- **JDBC** utility classes for Oracle database connections pooling and sessions fail/recovery
- Sophisticated **XML** encoding/parsing using Apache XmlBean