The Case for Erlang as a Testing Language

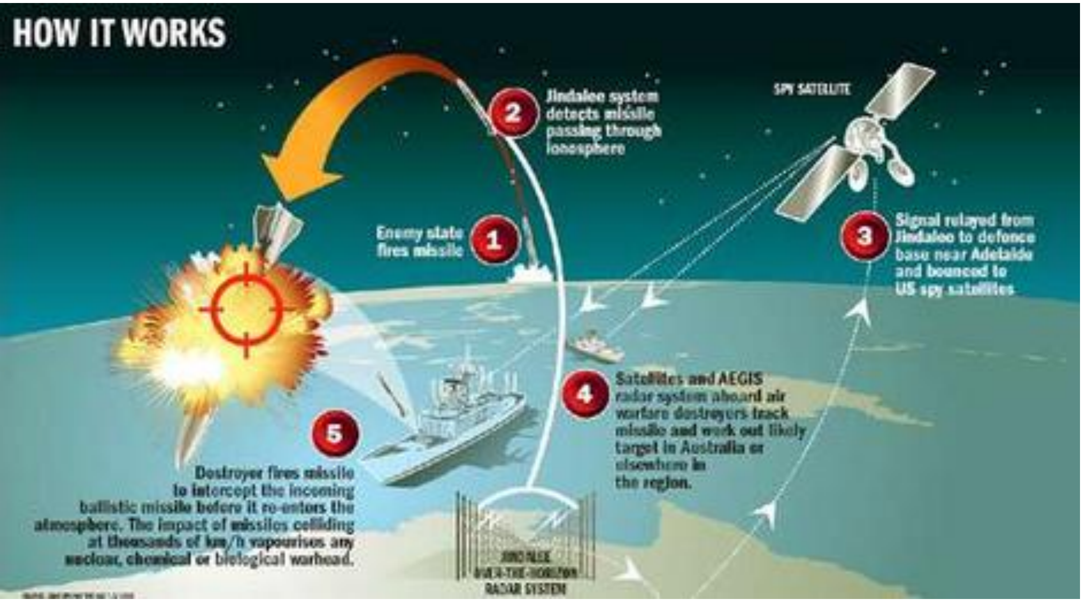
Automation and Property Testing

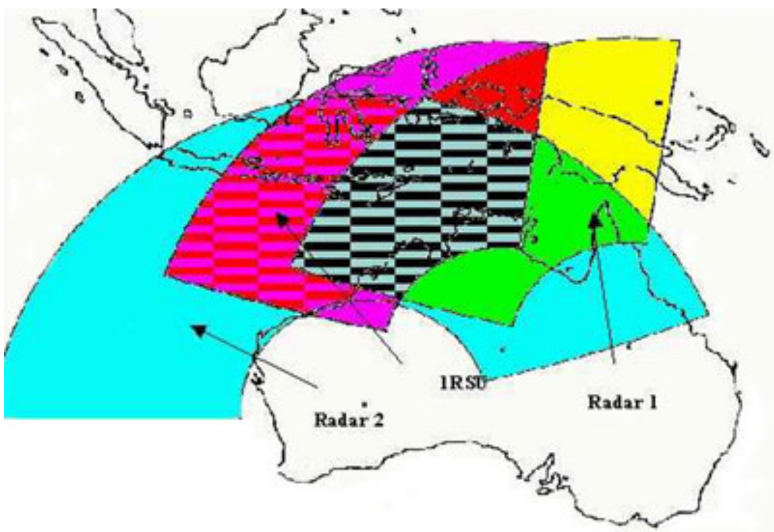
Graham Crowe

Overview

- A broad summary of my experiences:
  - Hardware Development Engineer
  - AXE (Digital Telephone Exchanges)
  - Using Python as testing language
  - Discovering Erlang
- Examples of applying Erlang for testing:
  - Stateless Property Testing
  - Stateful Property Testing

Jindalee Operational Radar Network





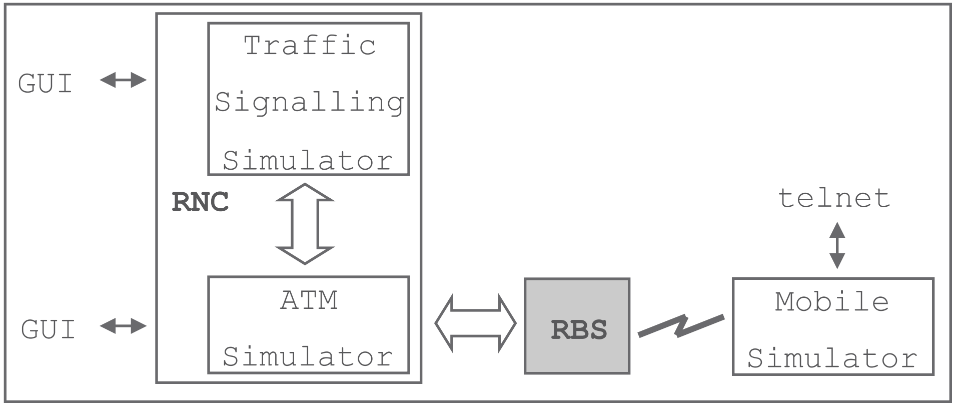
CAND98/0153-25
JORN PROJECT RECEIVER SITE ANTENNA ARRAY, LAVERTON W.A.
PIC BY CPL DAVE BROOS, DEFENCE PUBLIC AFFAIRS.

The AXE 10 Years (1994-2002)

- Structured and consistent
  - blocks, subsystems
- Proprietary language
  - PLEX (PASCAL flavored)
  - ASA (assembler language)
- Execution Model
  - signaling between blocks
  - no shared data between blocks
  - global job buffers for signaling
- Debugging
  - "Test System;"
- Patchable (ASA)
- Forlopp

The Python Years (2002-2005)

- Manual testing laborious and error prone
- Developed a test environment for *"black box"* testing
  - reduced manual effort but …
  - complete automation not supported
- Concurrency essential for automation
  - multiple interfaces to the *"black box"* that require coordination
  - Multiple clients to the *"black box"*
- Added concurrency to the test environment



The Erlang Years (2005-present)

- Easy to grasp its notion of concurrency
  - my PLEX background helped
- Functional programming seemed odd to begin with
  - now feels natural
- High abstraction
  - very easy to design code that does rather complex things
- Introduced to QuickCheck

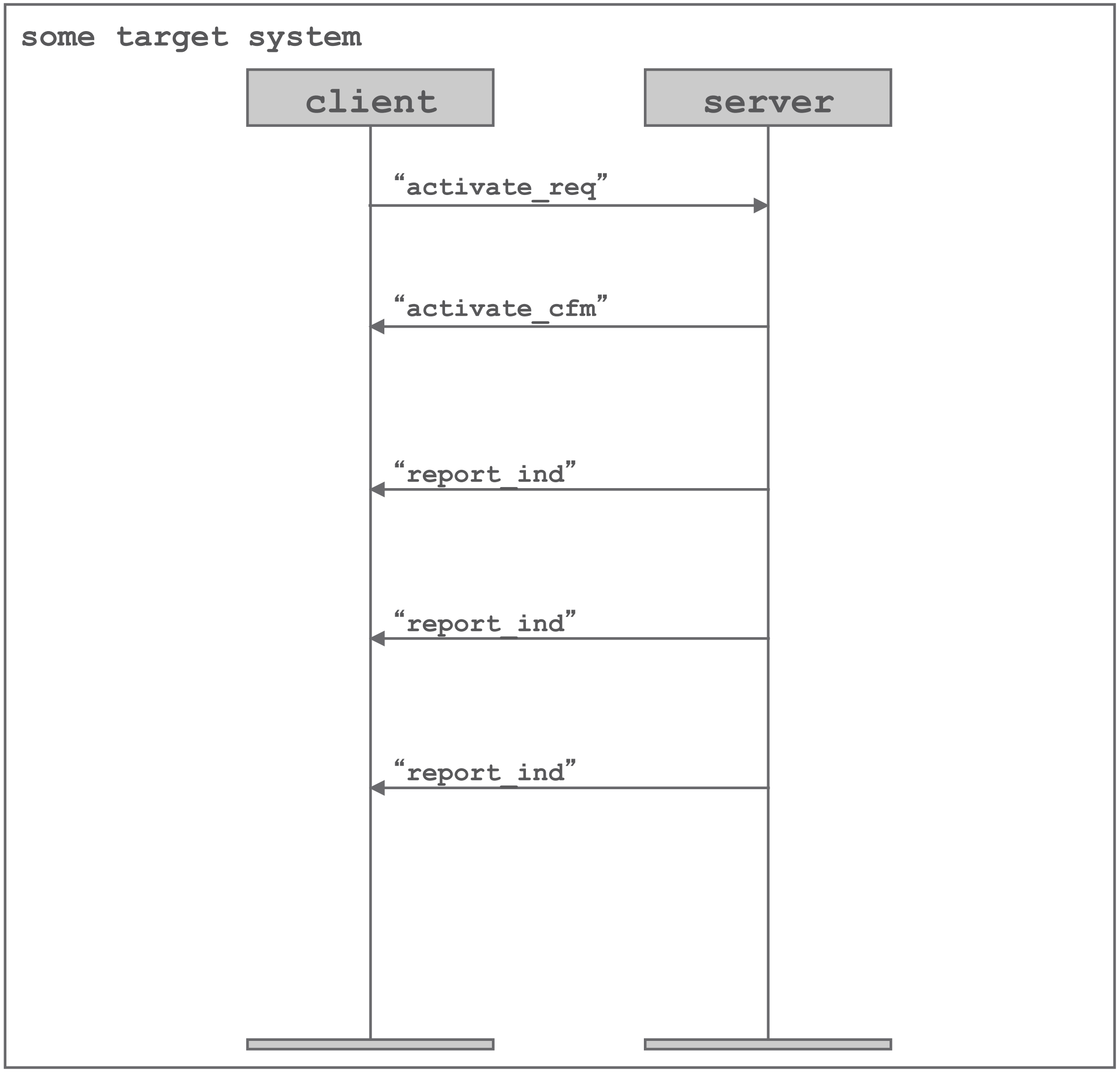
The case for Erlang as a testing language

- Erlang has a rich tool set for testing systems written in Erlang, ranging from:
  - unit test -> integration test -> system test
- What about testing systems that are not written in Erlang?
- In general unit testing is performed in the native language
  - Exception: Quickcheck has support for unit testing C code compiled with GCC
- Test environments for integration test and system test are often written in other languages
- Why not use Erlang?
- Why use Erlang?

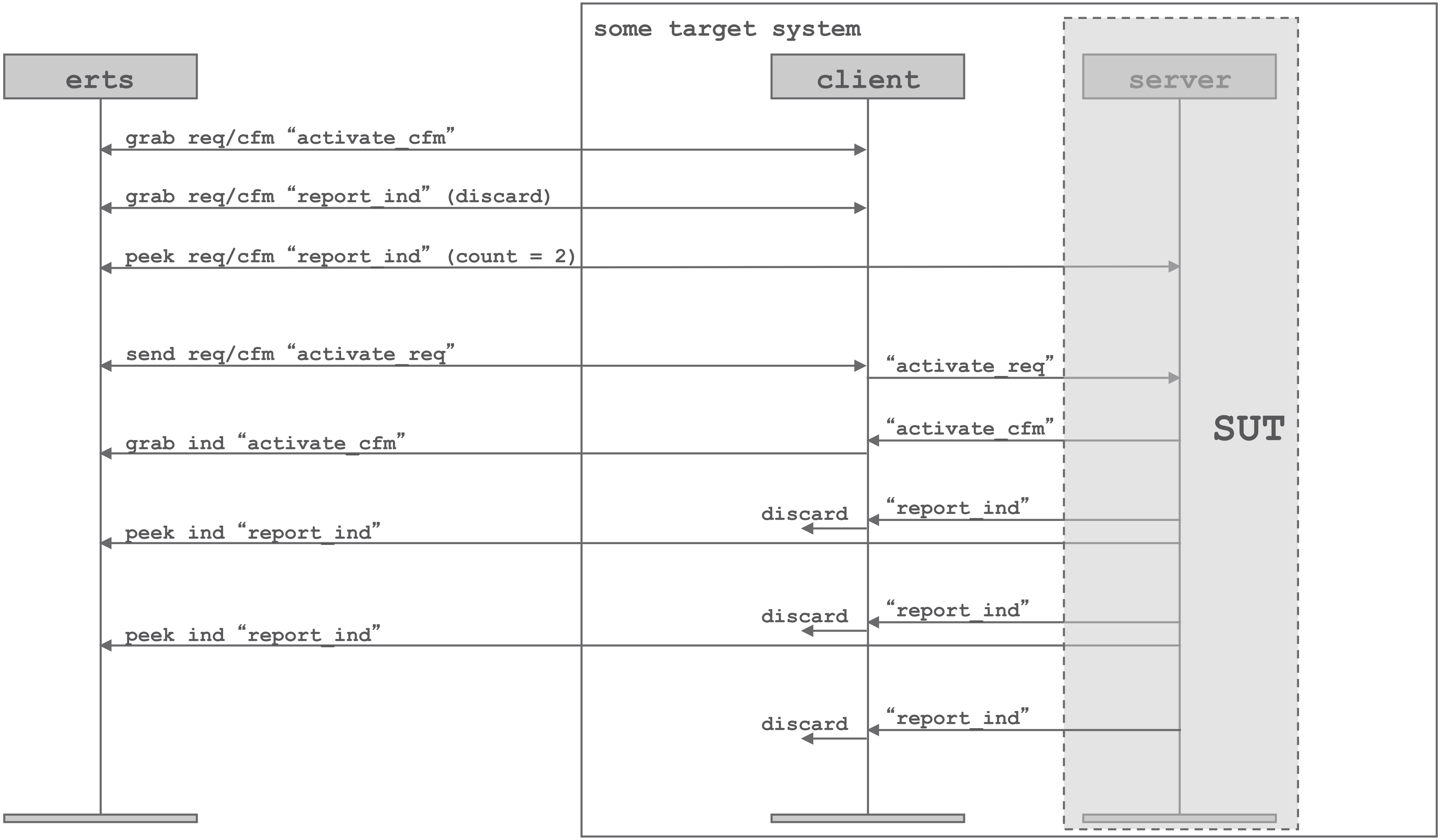
Some Concepts for Grey Box Testing

- We need some enablers in our SUT for communication with the ERTS:
  - SEND
    - Send an asynchronous signal to a component
  - PEEK
    - Forward a copy of a particular sent or received signal to the requester
  - GRAB
    - Redirect a particular sent or received signal (to the requester or discard)

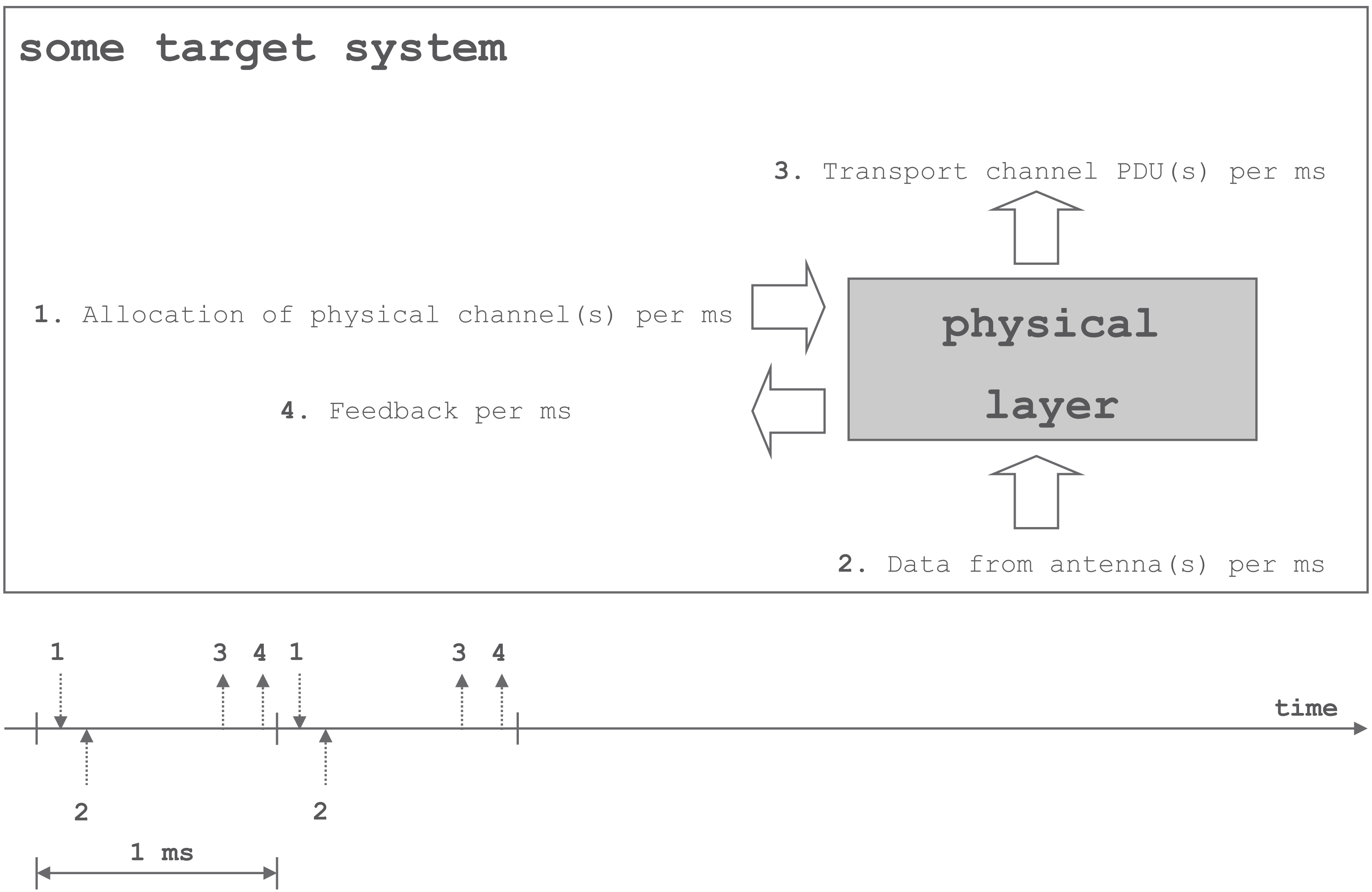
Some Simple Client Server Use Case



Applying PEEK, GRAB & SEND

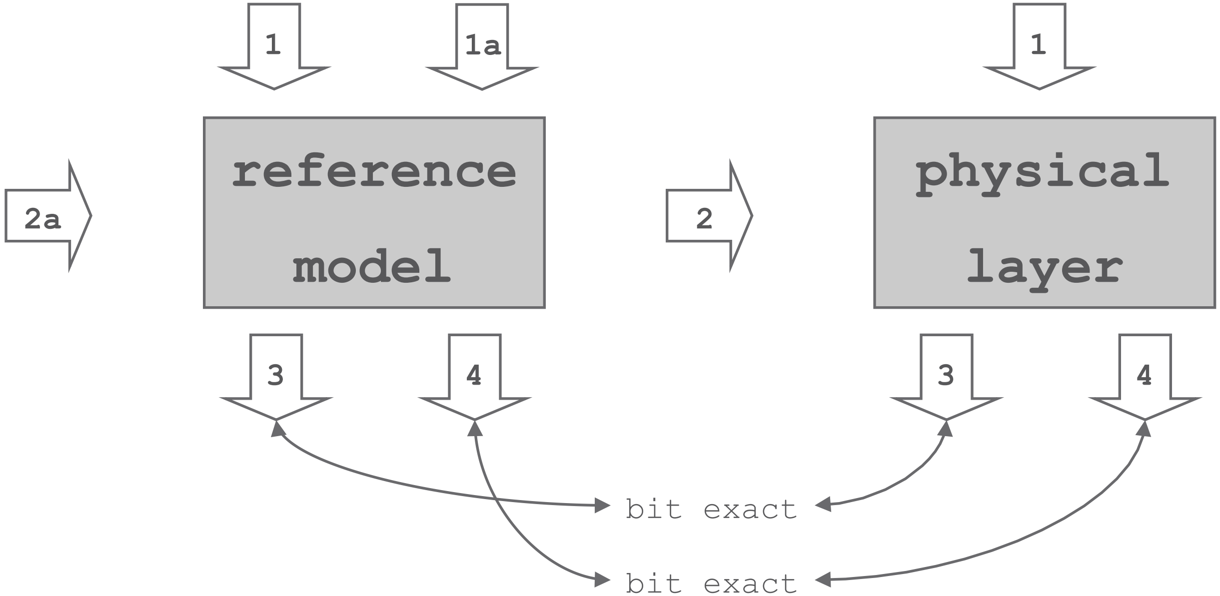


Stateless Test: Hard Real Time Component



Bit Exact Verification

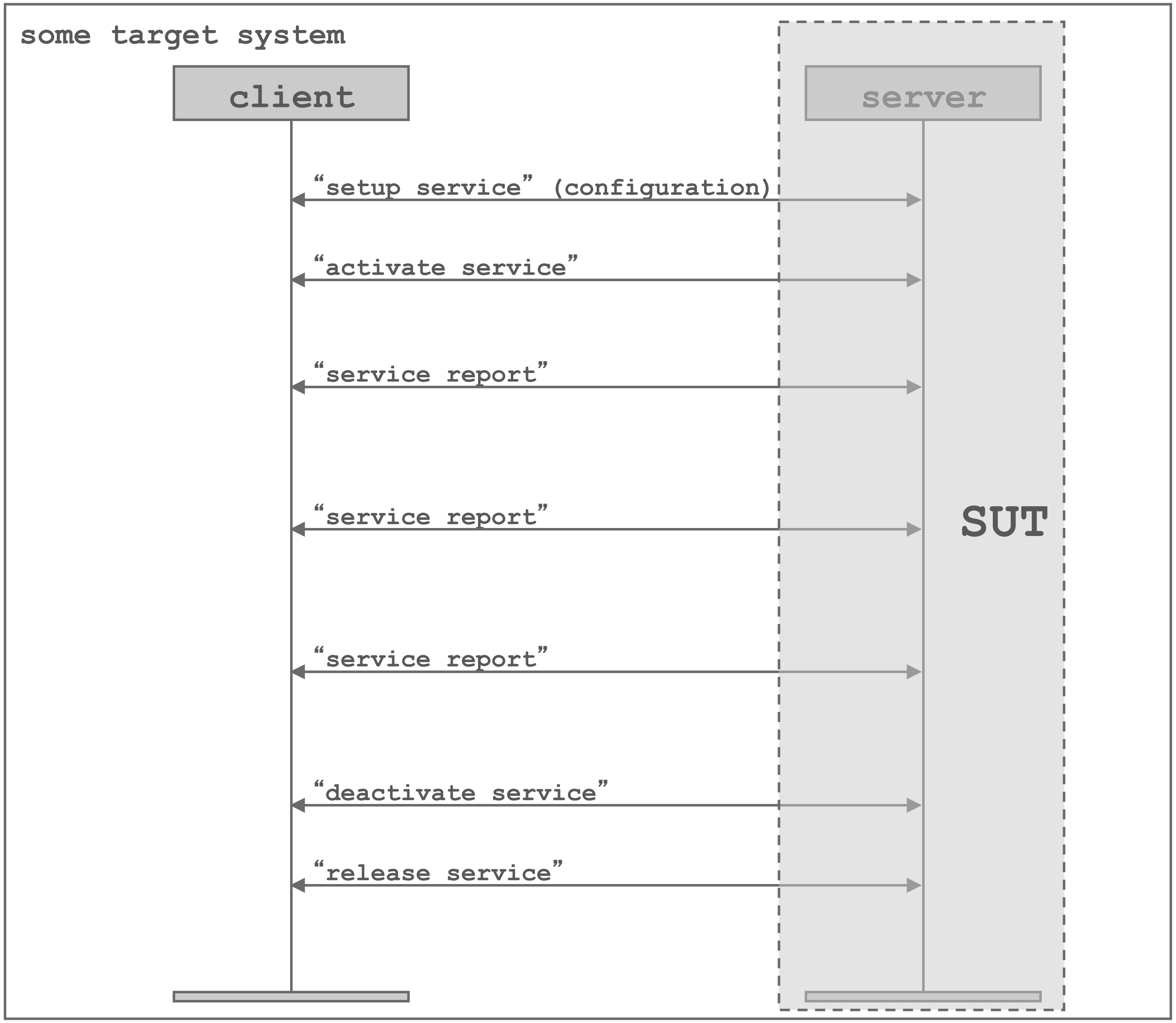
1. Allocation of physical channel(s) per ms
1a. Physical channel(s) content
2. Data from antenna(s) per ms
2a. Modeled radio conditions
3. Transport channel PDU(s) per ms
4. Feedback per ms

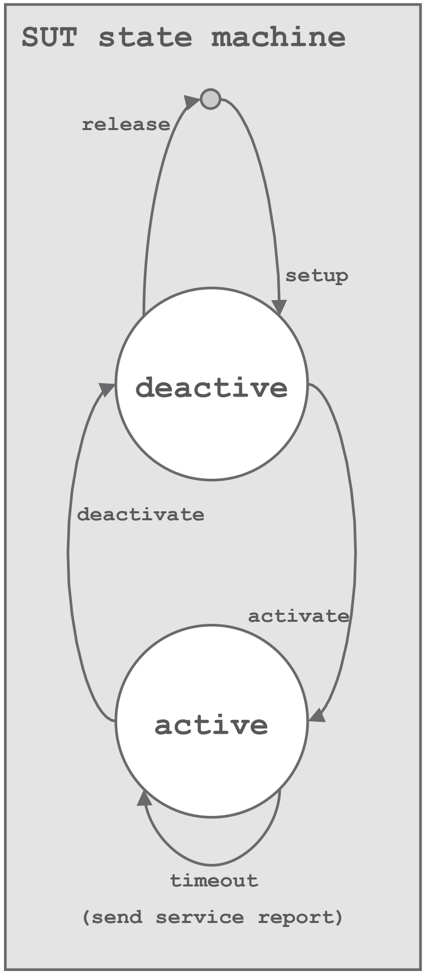


Stateless Test: Test Data Generation

- Each test case using the bit exact methodology can be described by an Abstract Data Type:
  - Allocation of physical channels (and content) per ms
  - Modeled radio conditions
- The ADT is used to generate the test vectors
  - antenna data (input)
  - feedback & transport channel data (expected output)
- The ADT is used to control the allocation of physical resources on the SUT
- We can then use stateless QuickCheck properties to explore the parameter space of this Hard Real Time component

Stateful Test: Another Simple Client Server





Stateful Test: Sequence of Events

- We need to test that scalable servers (as depicted) can provide multiple instances of their services in all cases
- We need to generate sequences of events (within the specification) for testing the server
- We need to test that properties of the server hold after each event

```
An example of a sequence of events in this case
Event1: “setup      instance 1”
Event2: “setup      instance 2”
Event3: “activate   instance 1”
Event4: “release    instance 2”
Event5: “setup      instance 3”
Event6: “activate   instance 3”
Event7: “deactivate instance 1”
Event8: “activate   instance 1”
…
EventN
```

```
Examples of Properties in this case

Event: “activate instance N”
Test that instance N sends reports periodically

Event: “deactivate instance N”
Test that instance N does not send any reports
```

Summary

- Given some enablers (such as the PEEK, SEND and GRAB concepts) within a large complex product it is possible to deploy property based testing
  - Stateless and Stateful
- Such enablers are often necessary in any case to assist integration (grey box testing) of large complex products
- When using Erlang as a testing language together with QuickCheck the potential benefits are enormous

ERICSSON