z/TPF Detailed Summary
z/TPF Automated Test Framework

Jennifer Chiarieri
z/TPF Development

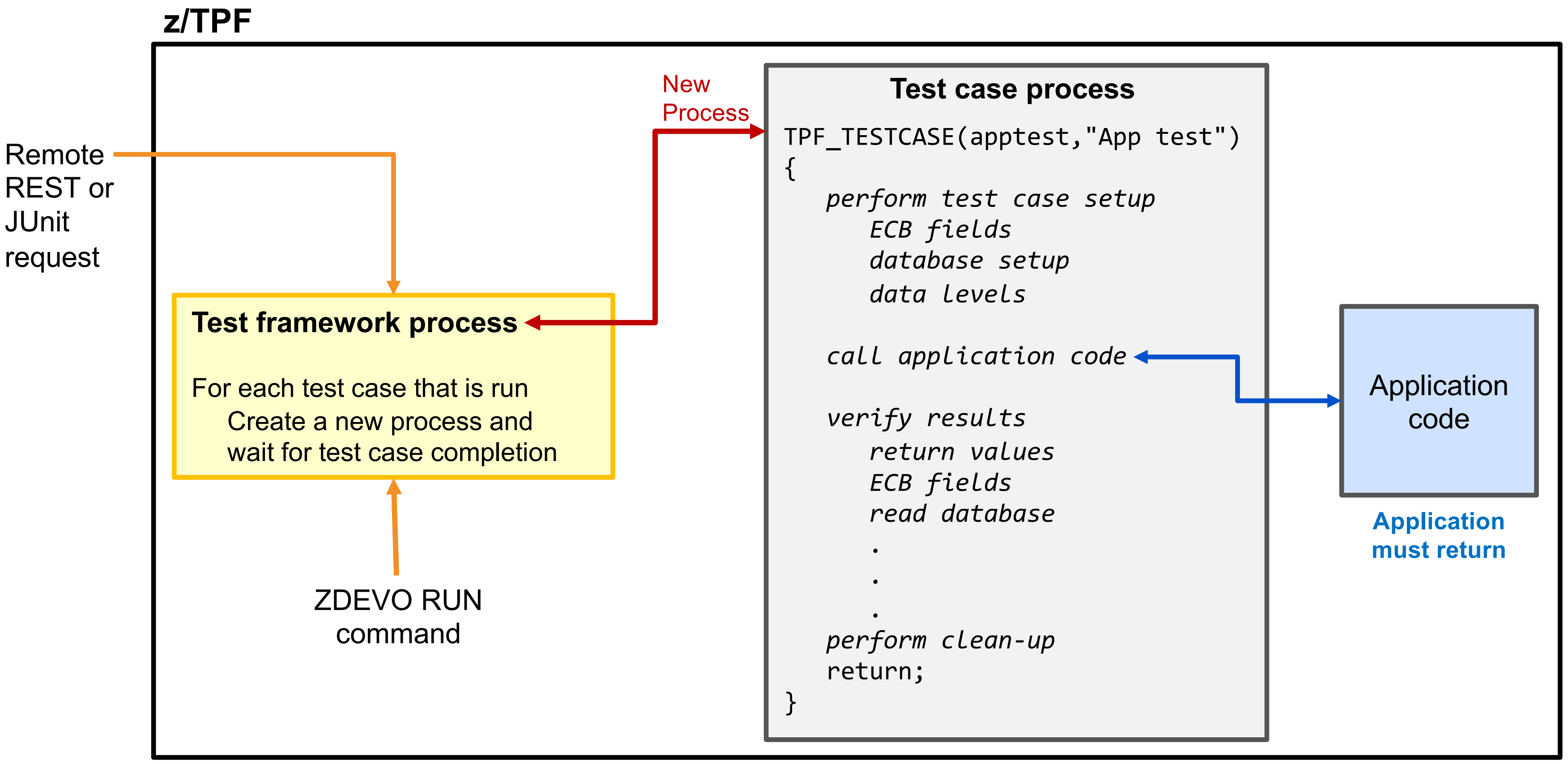
z/TPF automated test framework agenda:

z/TPF automated test framework agenda:

Overview

- Provides a testing framework tailored for z/TPF testing
- Provides a method to create self-validating programmatic test cases similar to other testing frameworks, like Google Test
- Simplifies development of new unit-level and function-level test cases
- Provides an interface to convert existing test drivers into automated test cases
- Provides multiple levels of diagnostics (debug, info, error)
- Uses a set of z/TPF APIs to address many of the challenge areas that are unique to z/TPF
- Organizes tests by namespace, for example, `airco.res.overbook`
- Allows multiple-ECB testing with parameter passing capabilities
- Provides multiple interfaces to query and run test cases defined in the z/TPF automated test framework (ZDEVO commands, REST interface, and JUnit interface)
- Detects new test cases automatically when they are loaded to the z/TPF system

Architecture


z/TPF
Remote REST or JUnit request
Test framework process
For each test case that is run
Create a new process and wait for test case completion
ZDEVO RUN command
New Process
Test case process
Application code
Application must return

```
TPF_TESTCASE(apptest,"App test")
{
   perform test case setup
      ECB fields
      database setup
      data levels

   call application code

   verify results
      return values
      ECB fields
      read database
      .
      .
      .
   perform clean-up
   return;
}
```

z/TPF automated test framework agenda:

Overview

Getting started

Test case handle

Test case properties

Multiple ECB testing

Overrides and intercepts

Running test cases

References

Getting started in 5 easy steps

- Step 1: Enable test automation on z/TPF
- Step 2: Include the **`idevops`** environment
- Step 3: Create a test suite
- Step 4: Create test cases
- Step 5: Build and load the test program to z/TPF

Step 1: Enable test automation on z/TPF

- Test cases can potentially consume resources, hold locks, or other dangerous system activity and should not be run on a production system
- To avoid running automated test cases on production or shared test systems, the automated test framework is **off** by default
- To enable test automation on z/TPF, use the **`ZSTRC ALTER`** command with the **`TESTAUTO`** option:

```
ZSTRC ALTER TESTAUTO
```

Step 2: Include the **`idevops`** environment

- Include the **`idevops`** environment variable in all makefiles that are used in the test application
- The **`idevops`** environment variable sets up the necessary environment for automated testing

qovb.mak

```
TPF_DRIVER := temp
APP := QOVB

maketpf_env := drvs
maketpf_env += idevops
maketpf_env += base_rt

C_SRC := qovb.c

include maketpf.rules
```

Step 3: Create a test suite

- Add `TPF_TESTSUITE` to one source file of a shared object
- Source file must include the `tpf/c_devops.h` header file

qovb.c

```
#include <tpf/c_devops.h>
.
.
.
TPF_TESTSUITE("airco.res.overbook","TOV*")
```

Step 4: Create test cases

- Add TPF_TESTCASE macros for each test case in the test suite
- Include a test case name and description
- Add the test case code

qovb.c

```
#include <tpf/c_devops.h>
.
.
.
TPF_TESTSUITE("airco.res.overbook","TOV*")

TPF_TESTCASE(overbook_firstClass, "Overbooking in first class") {
     // Test case code
}

TPF_TESTCASE(overbook_economy, "Overbooking in economy")  {
     // Test case code
}
.
.  Additional test cases
.
```

Step 4: Create test cases (continued)

- Use basic macros to improve test case usability:
  - `TPF_TC_INFO` – includes an informational output message for a test case; status not changed
  - `TPF_TC_IGNORE` – notifies automated test framework of a skipped test case; status changed to ignored
  - `TPF_TC_ERROR` – marks a test case as failed and generates an error message; status changed to error
  - `TPF_TC_DEBUG` – includes a debug output message for a test case; status not changed
- These basic macros use familiar `printf` style formatting

Basic test case example

qovb.c

```
TPF_TESTCASE(overbook_firstClass,"Test overbooking in first class") {
     struct overbook_input overbook_parms;
     ⋮
     setup environment for call
     ⋮
     TPF_TC_INFO("Calling overbook routine");
     ⋮
     TPF_TC_TIMEOUT(15);

     TPF_TC_DEBUG("Changed the timeout value to 15 seconds");

     int rc = process_overbook(&overbook_parms);

     if (rc == RETURN_ERROR) {
          TPF_TC_ERROR("process_overbook failure-%d", overbook_parms.errCode);
     } else {
          TPF_TC_INFO("process_overbook completed successfully");
          validate results
     }
     ⋮
     restore environment
     return;
}
```

Creating a test case flow message

Changing the default timeout value

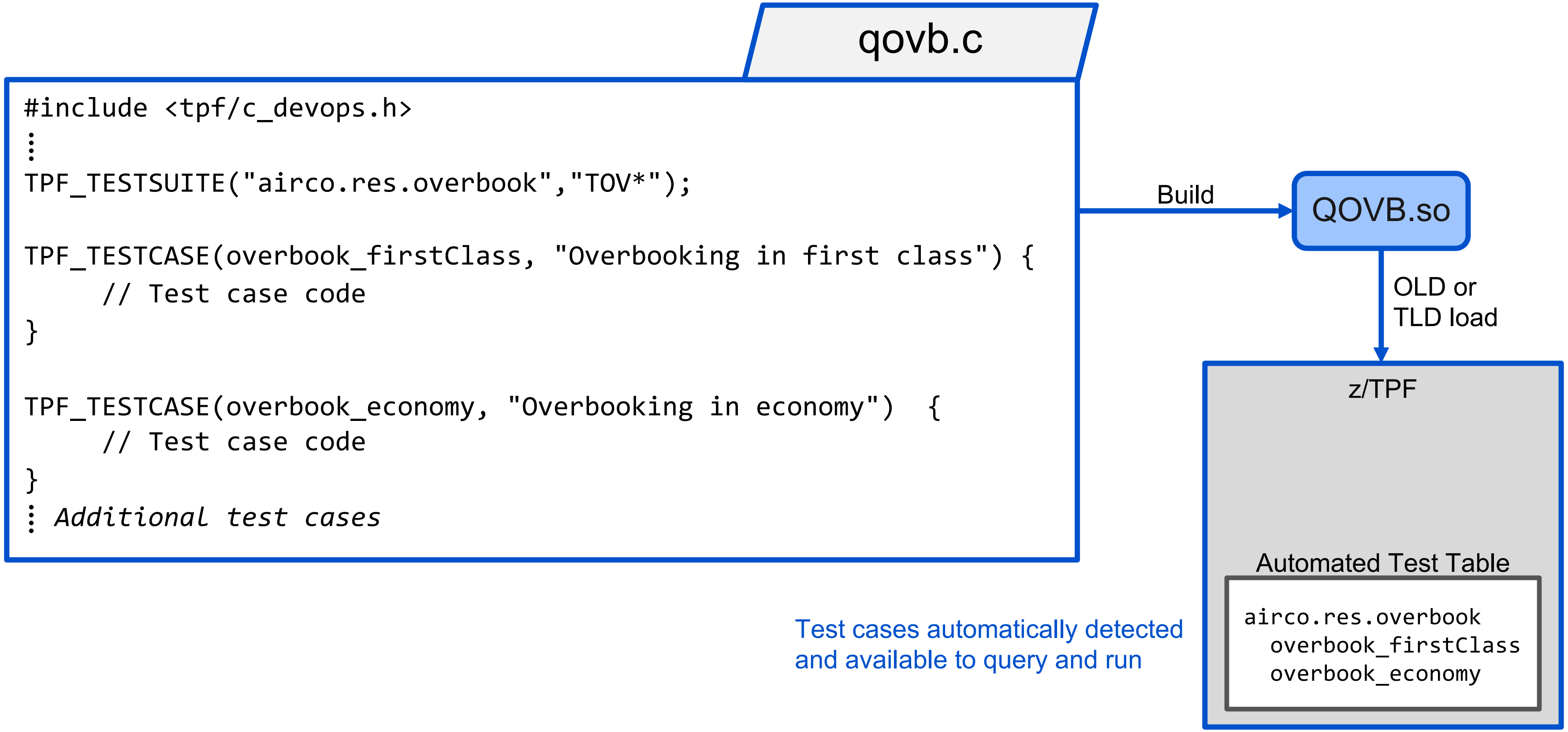
Creating a debug message

Calling an application function

Creating an error message

Creating another test case flow message

Step 5: Build and load the test program to z/TPF



qovb.c

```
#include <tpf/c_devops.h>
⋮
TPF_TESTSUITE("airco.res.overbook","TOV*");

TPF_TESTCASE(overbook_firstClass, "Overbooking in first class") {
     // Test case code
}

TPF_TESTCASE(overbook_economy, "Overbooking in economy")  {
     // Test case code
}
⋮ Additional test cases
```

```
airco.res.overbook
  overbook_firstClass
  overbook_economy
```

Test cases automatically detected and available to query and run

z/TPF automated test framework agenda:

Overview

Getting started

Test case handle

Test case properties

Multiple ECB testing

Overrides and intercepts

Running test cases

References

Understanding the test case handle

- The test case handle stores some state information to manage the test case
  - Represented as an unsigned integer
  - Initially set when you call TPF_TESTCASE
  - Must be set to call the test case APIs
- The test case handle is propagated between ECBs using a name-value pair
  - Propagation is automatic for most normal ECB creation: `cremc`, `swisc`, or `tpf_fork`
  - Propagation is not automatic for `cretc` or `activtate_on_receipt`, so `TPF_TC_GET_HANDLE` and `TPF_TC_SET_HANDLE` are required

Using the test case handle – an example

test.c

```
TPF_TESTCASE(hand1,"test case handle") {
// TEST CASE STARTS HERE
  unsigned int handle = TPF_TC_GET_HANDLE();
  TPF_TC_NEW_ECB();
  cretc(CRETC_SECONDS, QZZ1, 2, &handle);
  return;
}
```

qzz1.c

```
extern "C" void QZZ1() {
// TEST CASE CONTINUES HERE
  unsigned int handle = *(unsigned int *)&ecbptr()->ebw000;
  TPF_TC_SET_HANDLE(handle);
  TPF_TC_INFO("entered ecb 2");
  TPF_TC_ECB_DONE();   // mark ECB 2 as finished
  return;
}
```

Passing the handle to an ECB that is created with `cretc`

z/TPF automated test framework agenda:

Understanding test case properties

- Properties are “variables” that you can use to store information about the test case
- Properties are saved in shared memory
- Properties can be used for:
  - Test case specific processing in common routines
  - Serialization of multi-ECB testing
- Properties are scoped for a test case handle – you can only access properties passed in or set as part of the current test case
- Use `TPF_TC_SET_PROPERTY` to set a property
- Use `TPF_TC_GET_PROPERTY` to get a property previously set with `TPF_TC_SET_PROPERTY`

Using properties – an example

Test case start

```
TPF_TESTSUITE("airco.res.overbook","TOV*");

TPF_TESTCASE(overbook_multiECB,"overbooking multiECB") {
   int option = 1;
   TPF_TC_SET_PROPERTY("myOption", (void*)option, sizeof(option));

   swisc_create(…)
   ⋮
   return;
}
```

New ECB

```
void main() {
  int length = 0;
  int option = *(int *) TPF_TC_GET_PROPERTY("myOption",length);

  switch (option) {
  ⋮
  }
}
```

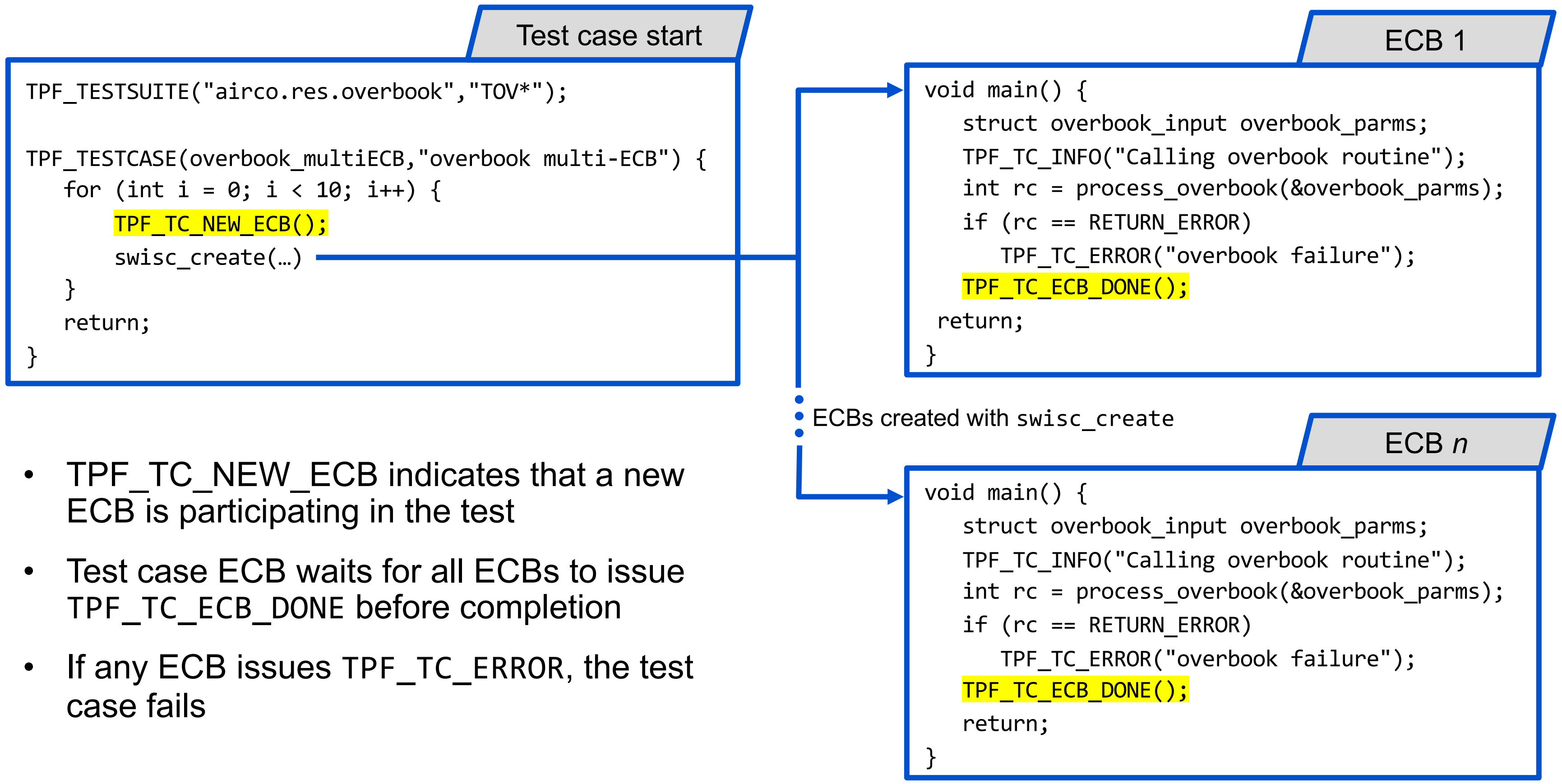
Passing properties to an ECB that is created as part of the test case

z/TPF automated test framework agenda:

Multiple ECB testing

- TPF applications / APIs often need multiple ECBs to perform a unit of test
- The z/TPF automated test framework provides a built-in mechanism to follow multiple ECBs without having to create custom tracking code
- Use `TPF_TC_NEW_ECB` to notify the framework that another ECB is participating in the test
- Use `TPF_TC_ECB_DONE` to notify the framework that the created ECB has completed

Multiple ECB testing – example



```
TPF_TESTSUITE("airco.res.overbook","TOV*");

TPF_TESTCASE(overbook_multiECB,"overbook multi-ECB") {
   for (int i = 0; i < 10; i++) {
      TPF_TC_NEW_ECB();
      swisc_create(…)
   }
   return;
}
```

```
void main() {
   struct overbook_input overbook_parms;
   TPF_TC_INFO("Calling overbook routine");
   int rc = process_overbook(&overbook_parms);
   if (rc == RETURN_ERROR)
      TPF_TC_ERROR("overbook failure");
   TPF_TC_ECB_DONE();
 return;
}
```

```
void main() {
   struct overbook_input overbook_parms;
   TPF_TC_INFO("Calling overbook routine");
   int rc = process_overbook(&overbook_parms);
   if (rc == RETURN_ERROR)
      TPF_TC_ERROR("overbook failure");
   TPF_TC_ECB_DONE();
   return;
}
```

- TPF_TC_NEW_ECB indicates that a new ECB is participating in the test
- Test case ECB waits for all ECBs to issue `TPF_TC_ECB_DONE` before completion
- If any ECB issues `TPF_TC_ERROR`, the test case fails

z/TPF automated test framework agenda:

Overview

Getting started

Test case handle

Test case properties

Multiple ECB testing

Overrides and intercepts

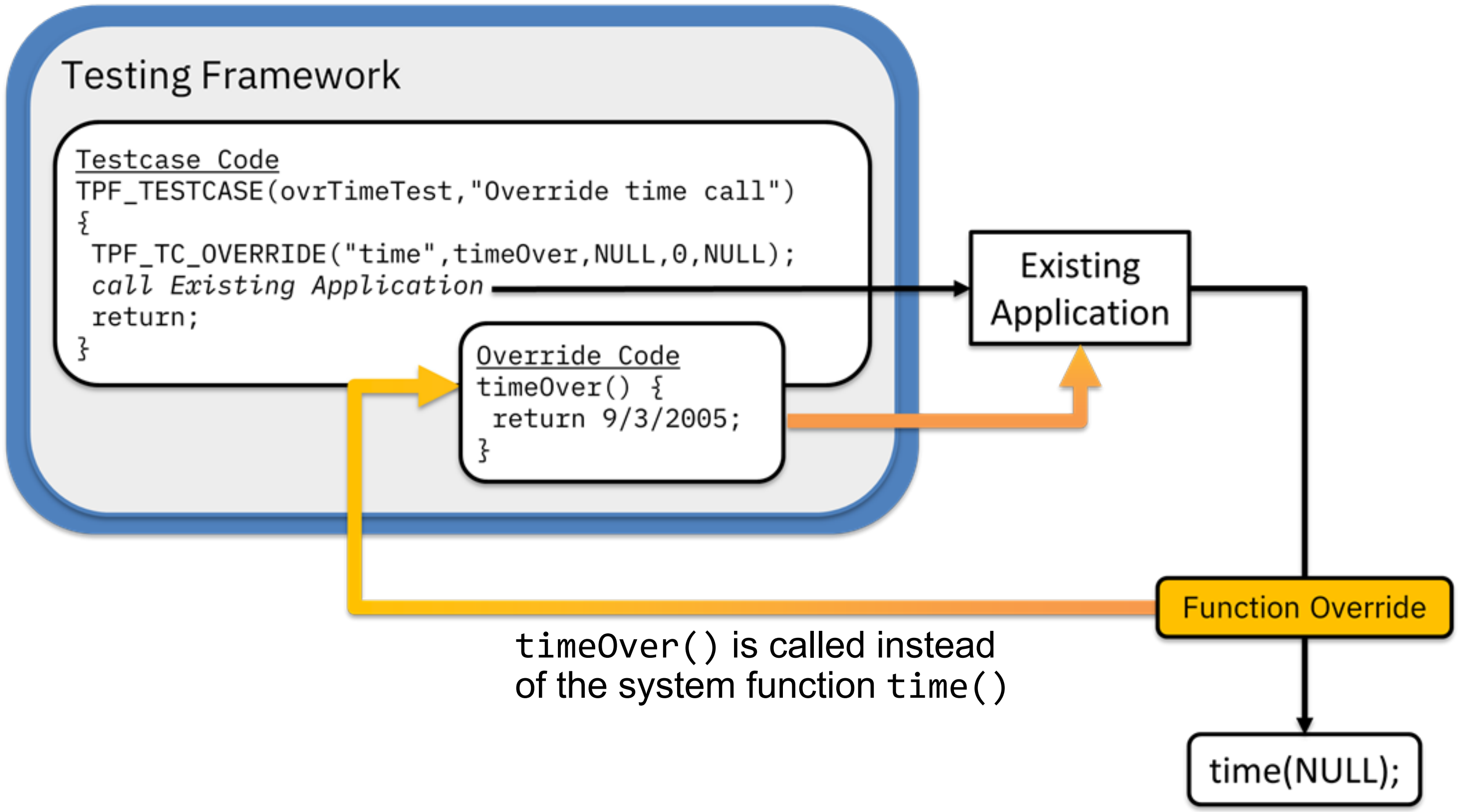
Running test cases

References

Overrides and intercepts

- Test more complex application code and programming models
- `TPF_TC_OVERRIDE` and `TPF_TC_INTERCEPT` macros provide support to override and intercept user functions, system functions, and 4-character program names
- Pass data to override and intercept functions in the test case logic
- Limit the scope in which a function is overridden or intercepted
- `TPF_TC_COMPLETE` macro provides more control over when a test case ends

TPF_TC_OVERRIDE – example

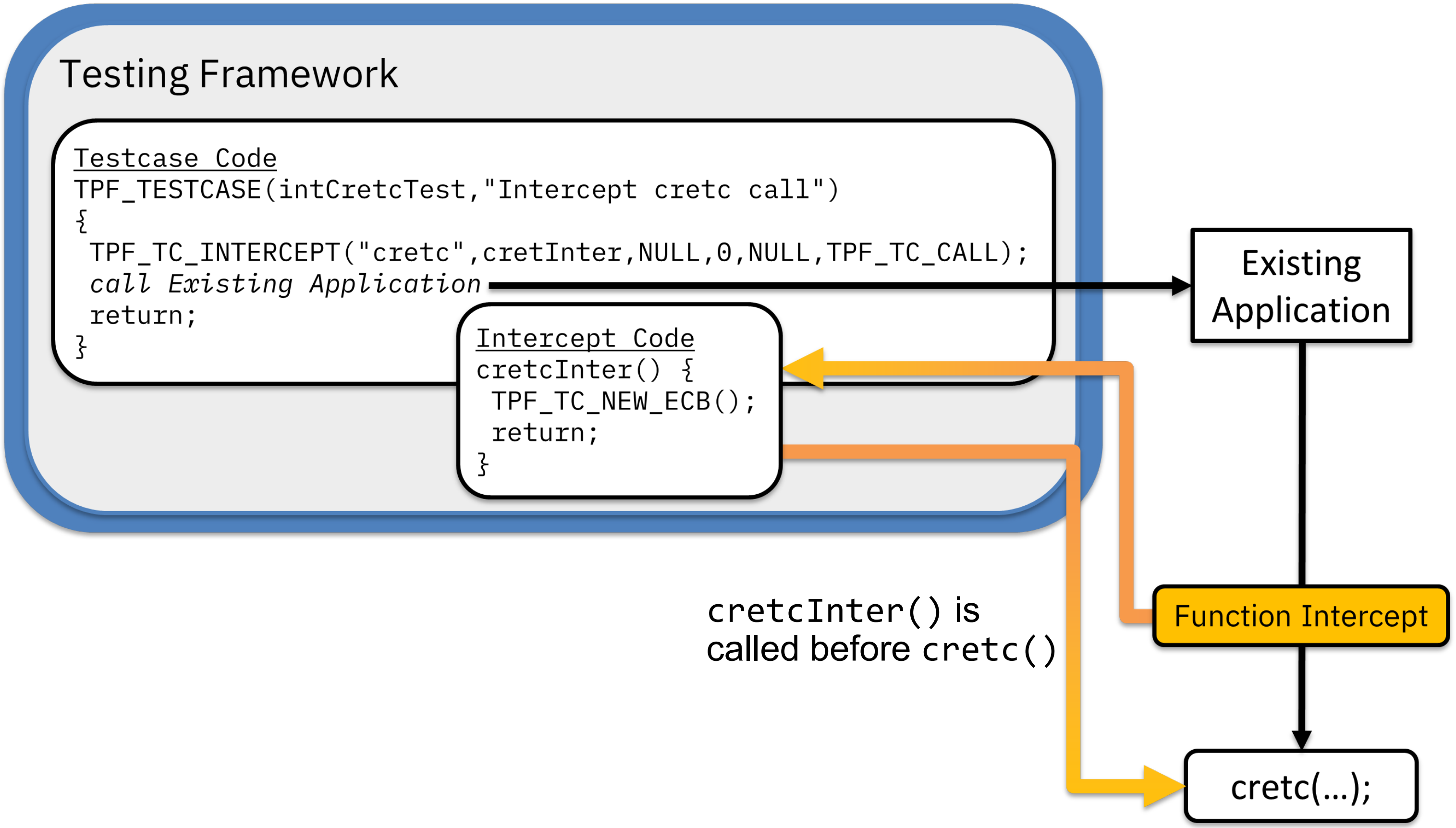


```
TPF_TESTCASE(ovrTimeTest,"Override time call")
{
 TPF_TC_OVERRIDE("time",timeOver,NULL,0,NULL);
 call Existing Application
 return;
}
```

```
timeOver() {
 return 9/3/2005;
}
```

`timeOver()` is called instead of the system function `time()`

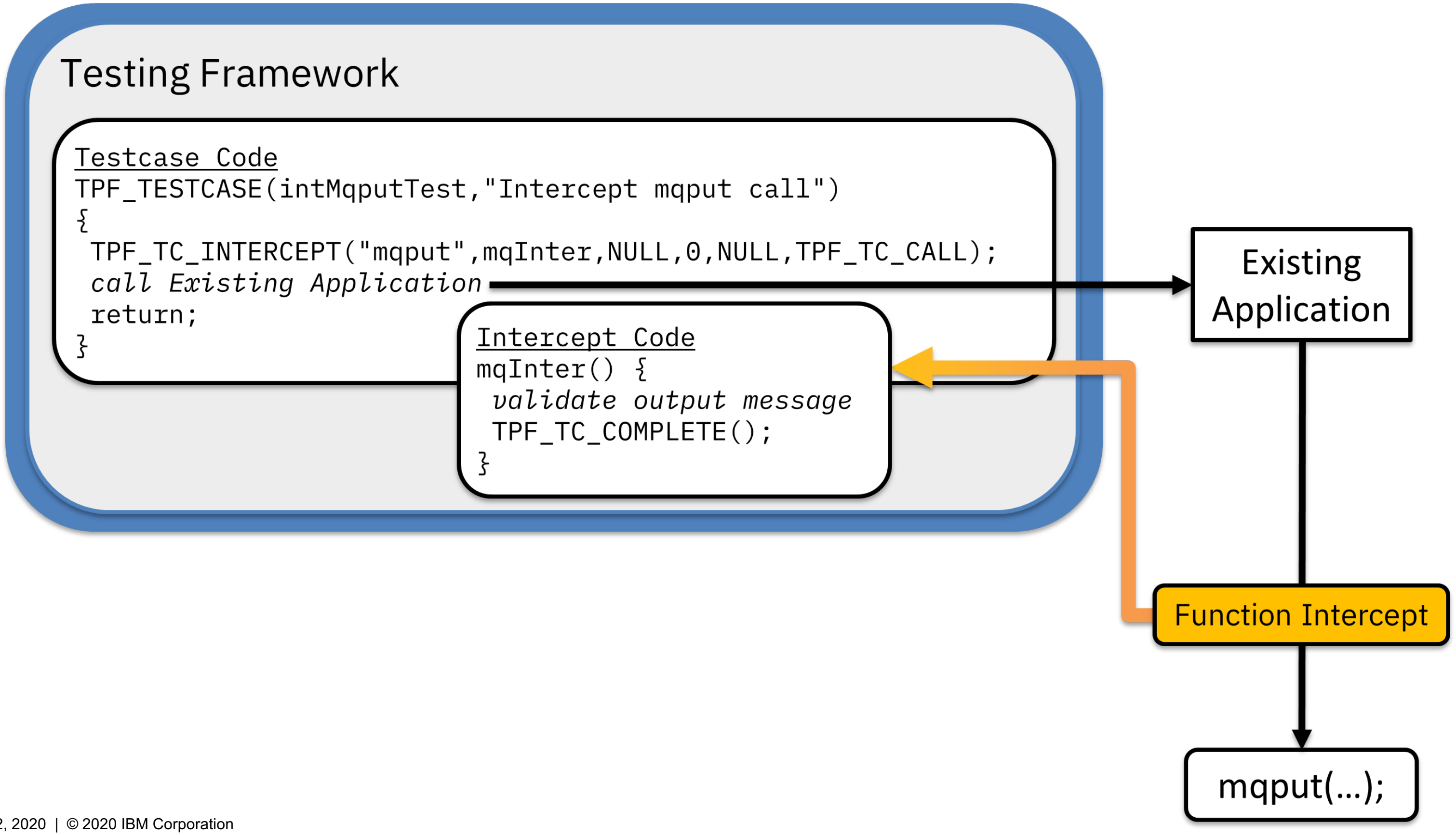
TPF_TC_INTERCEPT – example



```
TPF_TESTCASE(intCretcTest,"Intercept cretc call")
{
 TPF_TC_INTERCEPT("cretc",cretInter,NULL,0,NULL,TPF_TC_CALL);
 call Existing Application
 return;
}
```

```
cretcInter() {
 TPF_TC_NEW_ECB();
 return;
}
```

Using TPF_TC_COMPLETE



```
Testcase Code
TPF_TESTCASE(intMqputTest,"Intercept mqput call")
{
 TPF_TC_INTERCEPT("mqput",mqInter,NULL,0,NULL,TPF_TC_CALL);
 call Existing Application
 return;
}
```

```
Intercept Code
mqInter() {
 validate output message
 TPF_TC_COMPLETE();
}
```

z/TPF automated test framework agenda:

Overview

Getting started

Test case handle

Test case properties

Multiple ECB testing

Overrides and intercepts

Running test cases

References

Running test cases

- ZDEVO commands
- REST interface
- JUnit interface

ZDEVO RUN: Run test cases

```
                              .-,------------.
                              V              |  .- -Loud----.
>>-ZDEVO Run-- --progspec-- ----+-caseName-+-+--+-----------+--+--------+--+-----------+-><
                                '-*--------'    +- -Whisper-+  '- -File-'  +- -Verbose-+
                                                '- -Quiet---'              '- -Debug---'
```

progspec	A program name, namespace filter, or comma-delimited program list; wildcards are accepted.
caseName	Run the matched test cases; wildcards are accepted.
Loud **Whisper** **Quiet**	Display status messages for all test cases, and information for passing and failing test cases. Display status messages for all test cases and information for failing and skipped test cases. Display status messages and information for failing test cases while suppressing status messages and information for passing and ignored test cases.
File	Direct output to the file system instead of the z/TPF console. Output is written to the `ZDEVO-progspec-testspec.out` file in the `/tmp` directory.
Verbose **Debug**	Display extra framework information. Display extra debugging information.

ZDEVO INFO: Query test cases

```
                               .-,------------.
                               V              |
>>-ZDEVO Info-- --progspec-- ----+-caseName-+-+--+---------+---><
                                 '-*--------'    '- -Quiet-'
```

progspec	A program name, namespace filter, or comma-delimited program list; wildcards are accepted.
caseName	Display information about the matched test cases; wildcards are accepted.
Quiet	Display the number of test cases in each of the selected namespaces that have test cases that match the specified criteria.

ZDEVO RUN and INFO: Example commands

`ZDEVO RUN ibm.comms.* *`	Runs all test cases in all test suites that begin with the namespace "`ibm.comms.`"
`ZDEVO RUN airco* overbook*`	Runs all test cases that start with the name "`overbook`", and that are part of test suites that begin with the "`airco`" namespace
`ZDEVO RUN airco.res.overbook *economy`	Runs all test cases that end with "`economy`", and that are part of the "`airco.res.overbook`" namespace
`ZDEVO RUN QBCD FIN*A?C`	Runs all test cases that start with "`fin`", end with "`a`" followed by some character then "`c`", and that are part of the QBCD shared object (for example "`finfa1c`" but not "`fin2abdc`")
`ZDEVO INFO QXZY *`	Displays information about all test cases that are part of the QXZY shared object
`ZDEVO INFO CXYZ,CABC,BDFE *`	Displays information about all test cases that are part of the CXYZ, CABC, and BDFE shared objects

ZDEVO RUN: Example output

zdevo run airco.res.overbook *firstClass

```
DEVO0004I 08.44.23 PROCESSING FOR THE SELECTED TEST CASES IS STARTED.+
DEVO0005I 08.44.23 RESULTS FOR TEST 1 - overbook_firstClass
-- Test overbooking in first class --
TEST CASE STARTED
  Calling overbook routine
  process_overbook completed successfully
TEST CASE COMPLETED IN 6ms - PASSED - overbook_firstClass
END OF DISPLAY+
DEVO0018I 08.44.23 1 TEST WERE COMPLETED.
                   1 PASSED, 0 FAILED, 0 SKIPPED+
```

zdevo run qovb * quiet

```
DEVO0004I 08.38.35 PROCESSING FOR THE SELECTED TEST CASES IS STARTED.+
CSMP0097I 08.38.35 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0018I 08.38.35 2 TESTS WERE COMPLETED.
                   2 PASSED, 0 FAILED, 0 SKIPPED+
```

ZDEVO INFO: Example output

`zdevo info airco.res.overbook *`

```
DEVO0010I 08.34.24 TEST CASE INFORMATION DISPLAY
PGM   NAME                         DESCRIPTION
----- ------------------------- ----------------------------------------------
******************************************************************************
QOVB  airco.res.overbook
******************************************************************************
QOVB  overbook_firstClass          overbooking in first class
QOVB  overbook_economy             overbooking in economy
******************************************************************************
2 TEST CASES TOTAL
END OF DISPLAY+
```

`zdevo info qovb * quiet`

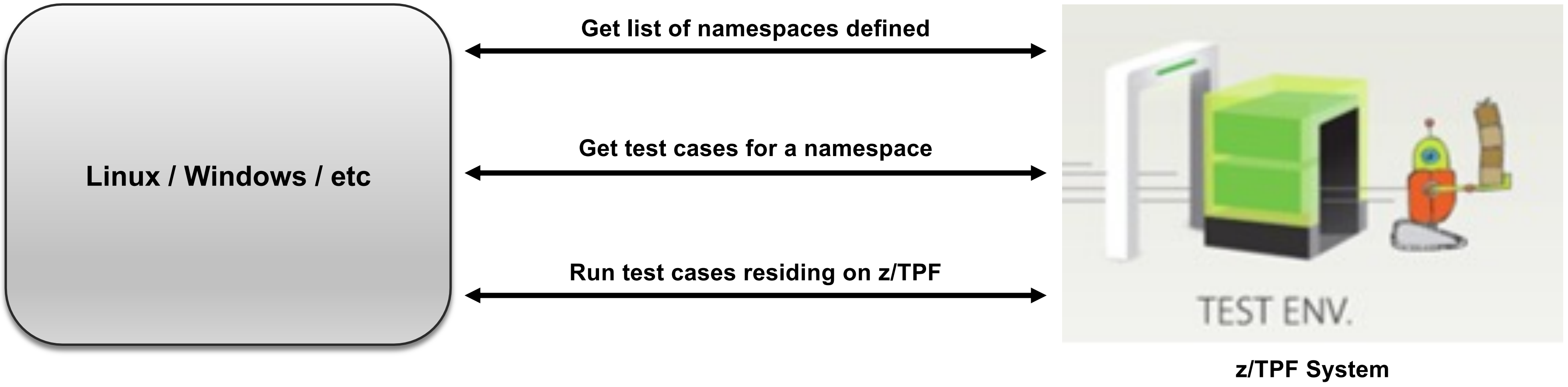
```
DEVO0010I 08.10.31 TEST CASE INFORMATION DISPLAY
PGM   #TEST NAMESPACE
----- ----- ------------------------------------------------------------------
QOVB      2 airco.res.overbook
******************************************************************************
2 TEST CASES TOTAL
END OF DISPLAY+
```

ZDEVO STATUS: Get status for running test cases

```
ZDEVO STATUS
DEVO0021I 11.17.37 TEST CASE RUN STATUS
HANDLE 0x26e53000
  TC RRMD:rmdir19
  - (rmdir when directory in use)
  expected completion in 16 seconds
  29 total, 18 passed, 0 failed, 0 skipped
HANDLE 0x26e54000
  TC RACE:access23_posix28 _
  - (access returns ENOTDIR when path prefix is not a dir)
  expected completion in 10 seconds
  34 total, 19 passed, 0 failed, 3 skipped
END OF DISPLAY+


ZDEVO STATUS
DEVO0021I 07.59.58 TEST CASE RUN STATUS
DEVO0017I 07.59.58 NO TEST CASES ARE RUNNING
END OF DISPLAY+
```

Rest interface: invoking test cases remotely


Linux / Windows / etc
Get list of namespaces defined
Get test cases for a namespace
Run test cases residing on z/TPF
TEST ENV.
z/TPF System

REST interface: enabling remote invocation

- Add `tpftest.tpf.swagger.json` to `/etc/tpf_httpserver/url_program_map.conf`
- Deploy `tpftest.tpf.swagger.json`

```
ZMDES DEPLOY FILE-tpftest.tpf.swagger.json
```

- Do not enable on production systems
- Manage access appropriately for test systems with sensitive data

REST interface: external properties

- You can pass external properties to a test case when using the REST interface. These are provided as EBCDIC string values when accessing through `TPF_TC_GET_PROPERTY`. For example, the IP address and port of a test server may be specified to a test case in this manner.
- The current set of properties are also returned as part of the execution result to retrieve "output" values from a test case.
- REST services have been created to query and run tests from a remote platform

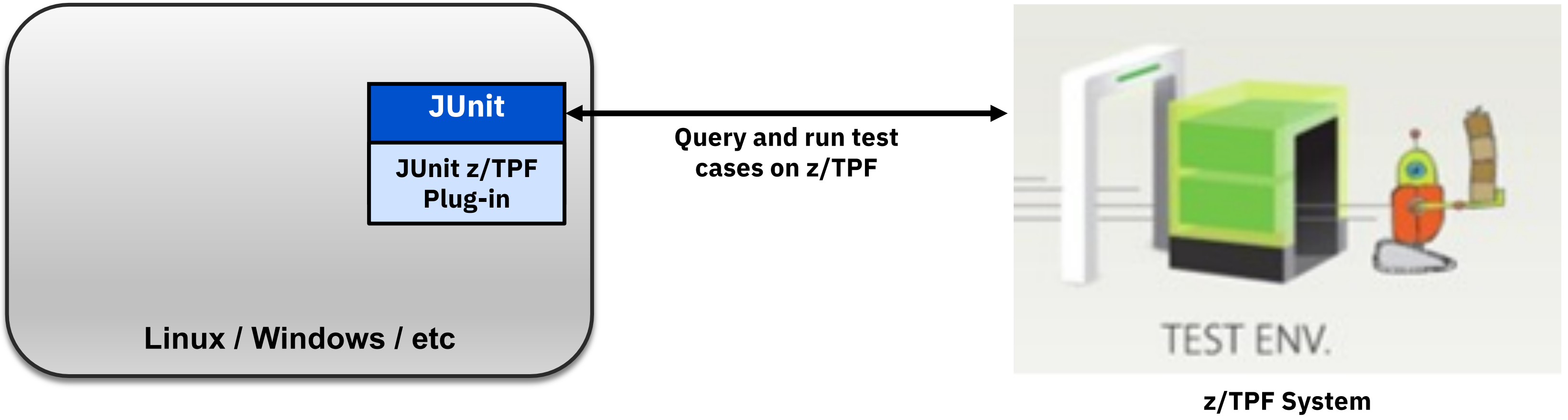
REST interface: running and querying test cases

`/ns`	List available namespaces
`/prog`	List available programs in a namespace
`/query`	List available test case in a program
`/run`	Execute a test case or check the status of a currently running test case

JUnit interface

- Provides the ability to run z/TPF automated tests from a Java application on a remote platform
- Allows integration between z/TPF and non-z/TPF test flows, for example, setup a test server then run a test case that connects to that server
- Allows invocation of test cases through TPF Toolkit, Maven, and other common Java tools for integrated build testing
- Allows use of Java MongoDB client for the database setup stage (if enabled) outside of your test case logic

JUnit interface: plug-in for z/TPF

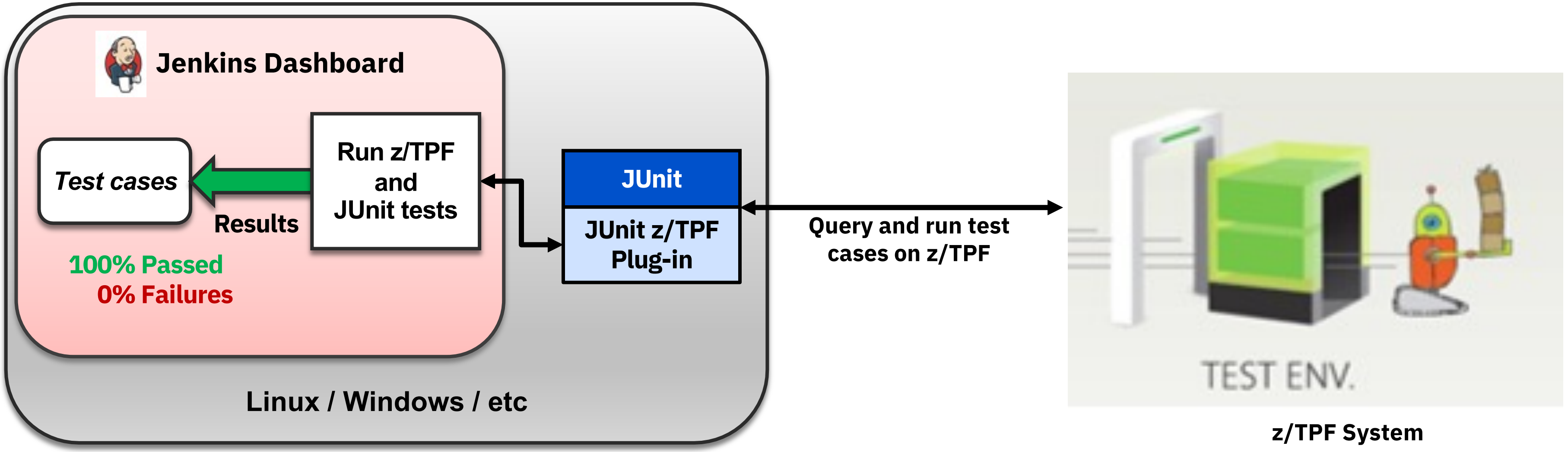


JUnit interface: running test cases

Sample implementation of how to query and run test cases over the REST interface, providing integration with JUnit framework:

`base/samples/junit`	root directory of java maven project
`com/ibm/tpf/test/TpfFrameworkTestRunner.java`	JUnit parameterized test that runs all the defined test cases over REST

Integrating into automated testing platforms



z/TPF automated test framework agenda:

References

- z/TPF automated test framework in IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/SSB23S_latest/gtpa2/tpfautotestfrwk.html
- TPFUG challenge: http://ibm.biz/tpfchallenge
- z/TPF automated test framework APARs:

APAR number	Quarter delivered	Description
PJ45217	1Q 2018	Infrastructure APAR
PJ43782	3Q 2018	Initial support - invocations from ZDEVO
PJ45488	4Q 2018	Remote invocation support - includes delivery and support of the z/TPF JUnit plug-in
PJ45801	3Q 2019	Overrides and intercepts

Thank You!

Questions or Comments?

Trademarks