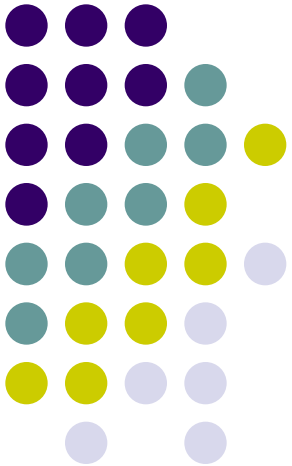
Chapter 10: Remote Procedure Call

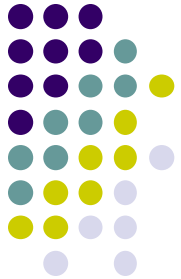
EE2405

嵌入式系統與實驗

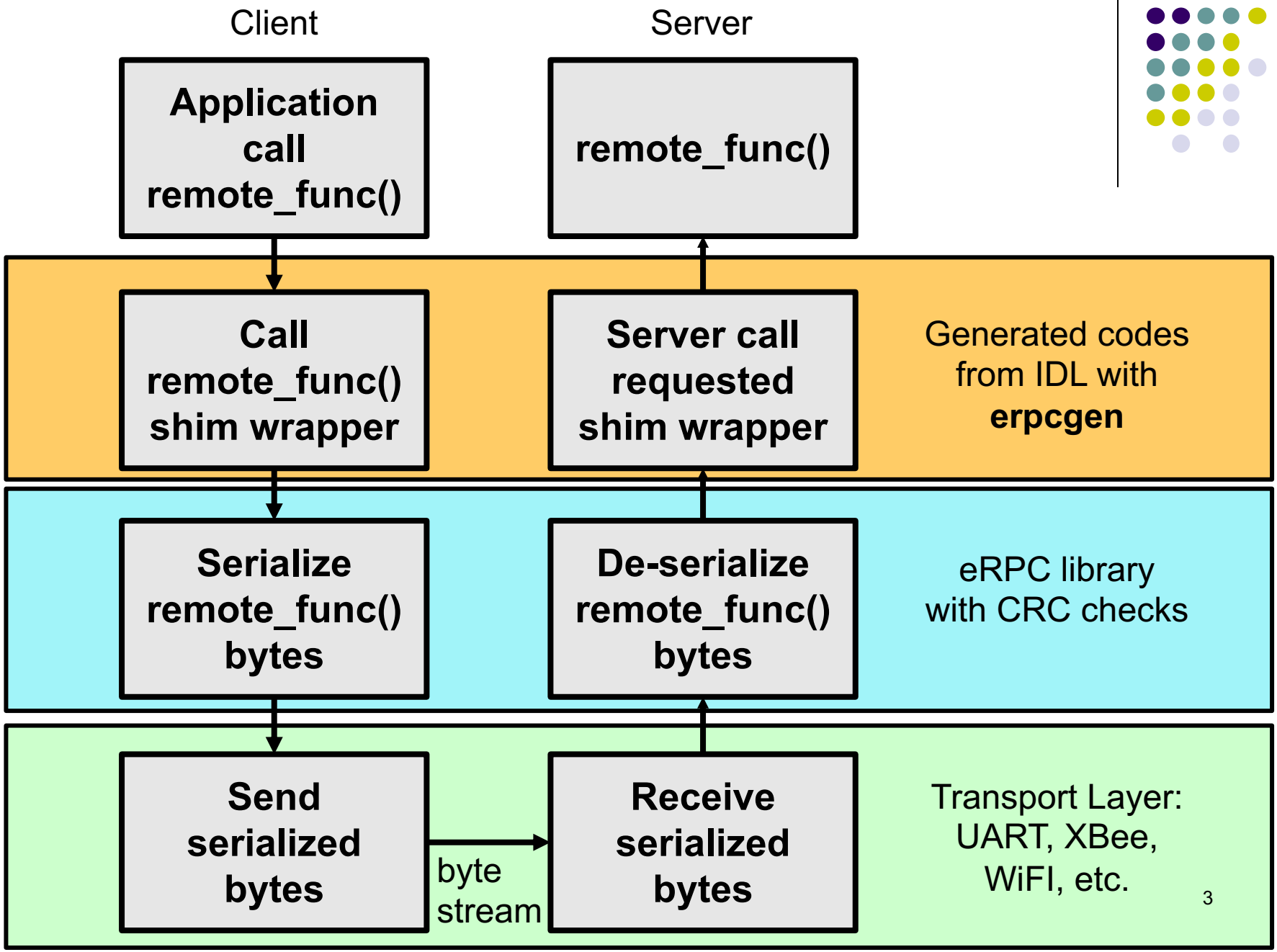
Embedded System Lab



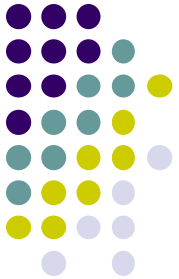
Remote Procedure Call



- Remote Procedure Call (RPC)
  - A pre-defined data format
  - The formatted string can trigger function calls at remote side
- The RPC library is an useful interface to define and send commands over a data channel
  - For example, serial port, Zigbee, WiFi, ethernet, etc..

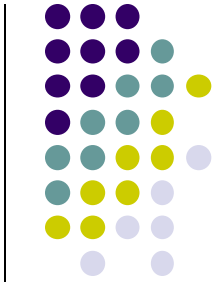

Client
Server
Application call remote_func()
remote_func()
Call remote_func() shim wrapper
Server call requested shim wrapper
Generated codes from IDL with erpcgen
Serialize remote_func() bytes
De-serialize remote_func() bytes
eRPC library with CRC checks
Send serialized bytes
byte stream
Receive serialized bytes
Transport Layer: UART, XBee, WiFI, etc.

eRPC



- Embedded Remote Procedure Call
  - A library created by NXP
- Cross-platform
- Lightweight
- Abstracted transport interface
- Multithreading

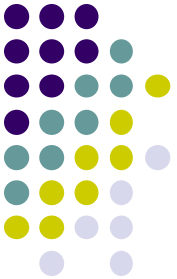
LED Example



- Generating and importing eRPC shim code.
- Import eRPC common files and call functions

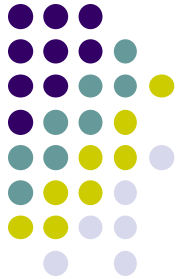
- Please also read the Getting Started Guide for a more generic example: https://github.com/EmbeddedRPC/erpc/wiki/Getting-Started

IDL (Interface Definition Language)



- LED Example: **led-service.erpc**

```
program blink_led;

interface LEDBlinkService
{
    led_on(in uint8 led) -> void
    led_off(in uint8 led) -> void
}
```

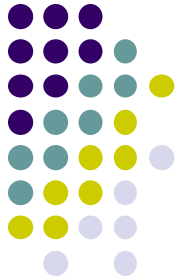


$ erpcgen led-service.erpc

blink_led.h blink_led_client.cpp blink_led_server.cpp

blink_led_server.h led-service.erpc

Files for server shim code (eprc wrapper).
They will be copied to mbed program folder.



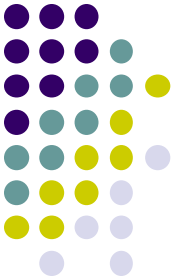
$ erpcgen –g py led-service.erpc

blink_led/

__init__.py client.py common.py interface.py server.py

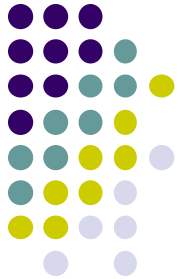
Python codes for client and server shim code (eprc wrapper).

mbed Server Code (1): RPC Functions



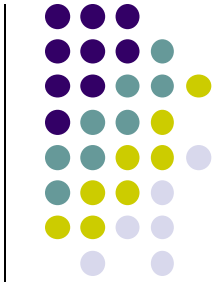
```
mbed::DigitalOut led1(LED1, 1);
mbed::DigitalOut led2(LED2, 1);
mbed::DigitalOut led3(LED3, 1);
mbed::DigitalOut* leds[] = { &led1, &led2, &led3 };

/****** erpc declarations *******/

void led_on(uint8_t led) {
    if(0 < led && led <= 3) {
        *leds[led - 1] = 0;
        printf("LED %d is On.\n", led);
    }
}

void led_off(uint8_t led) {
    if(0 < led && led <= 3) {
        *leds[led - 1] = 1;
        printf("LED %d is Off.\n", led);
    }
}
```

mbed Server Code (2): Server



```
/** erpc infrastructure */
ep::UARTTransport uart_transport(D1, D0, 9600);
ep::DynamicMessageBufferFactory dynamic_mbf;
erpc::BasicCodecFactory basic_cf;
erpc::Crc16 crc16;
erpc::SimpleServer rpc_server;

/** LED service */
LEDBlinkService_service led_service;

int main(void) {

    // Initialize the rpc server
    uart_transport.setCrc16(&crc16);
    rpc_server.setTransport(&uart_transport);
    rpc_server.setCodecFactory(&basic_cf);
    rpc_server.setMessageBufferFactory(&dynamic_mbf);

    // Add the led service to the server
    rpc_server.addService(&led_service);

    // Run the server. This should never exit
    rpc_server.run();
}
```

Python Client Code



```
import erpc
from blink_led import *

# Initialize all erpc infrastructure
xport = erpc.transport.SerialTransport(sys.argv[1], 9600)
client_mgr = erpc.client.ClientManager(xport,
                                       erpc.basic_codec.BasicCodec)
client = client.LEDBlinkServiceClient(client_mgr)

# Blink LEDs on the connected erpc server
turning_on = True
while True:
    for i in range(1, 4):
        if(turning_on):
            print("Call led_on ", i)
            client.led_on(i)
        else:
            print("Call led_off ", i)
            client.led_off(i)
        sleep(0.5)
    turning_on = not turning_on
```

Remote function calls