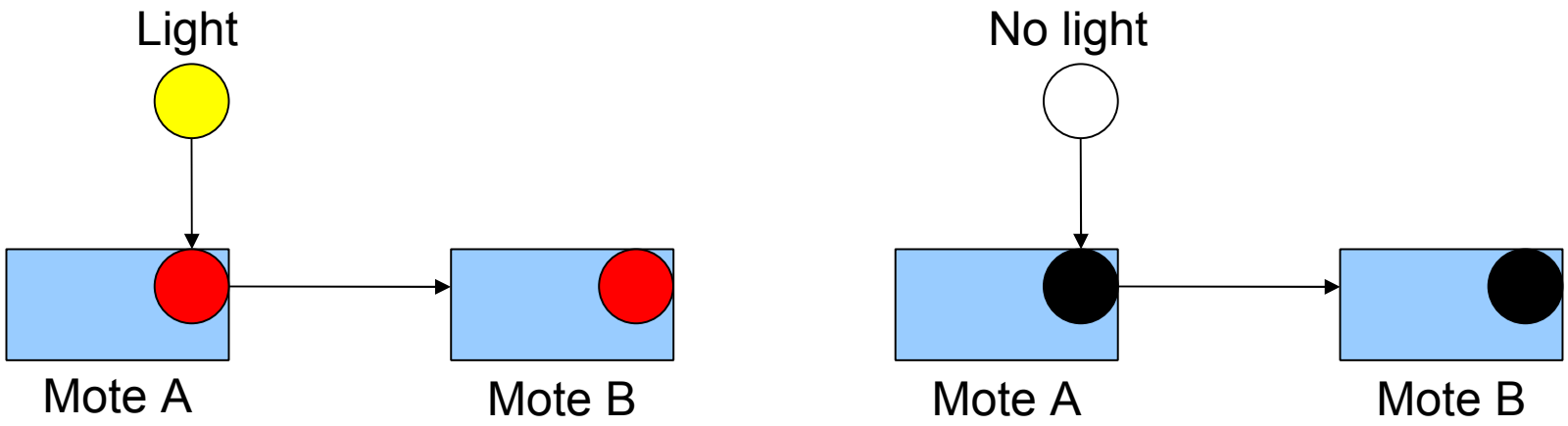
Lab 3

- Goal 1: combine sensing, communicating and processing tasks in a single application
- Goal 2: see the application running on real motes

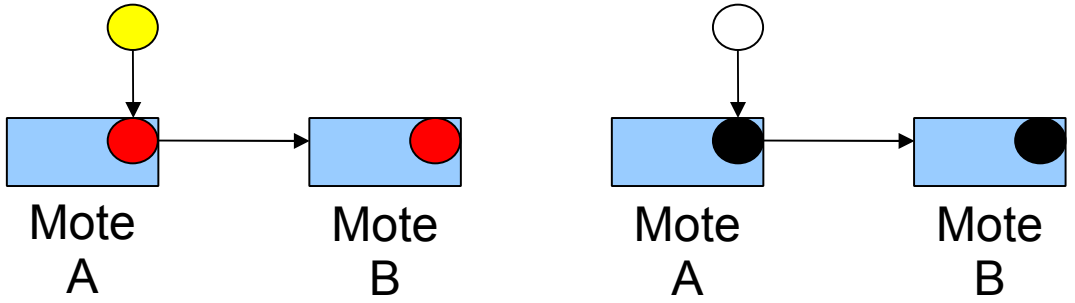
Description of the problem

- Problem:
  - Application `SendLight` described by the following figure



Description of the solution

- Solution:
  - Mote A raises a timer every second
  - When the timer is fired, A samples the light
  - When the light is sampled
    - A shows the light level
    - A sends the light reading to B
  - When receiving the light reading, B shows the light level



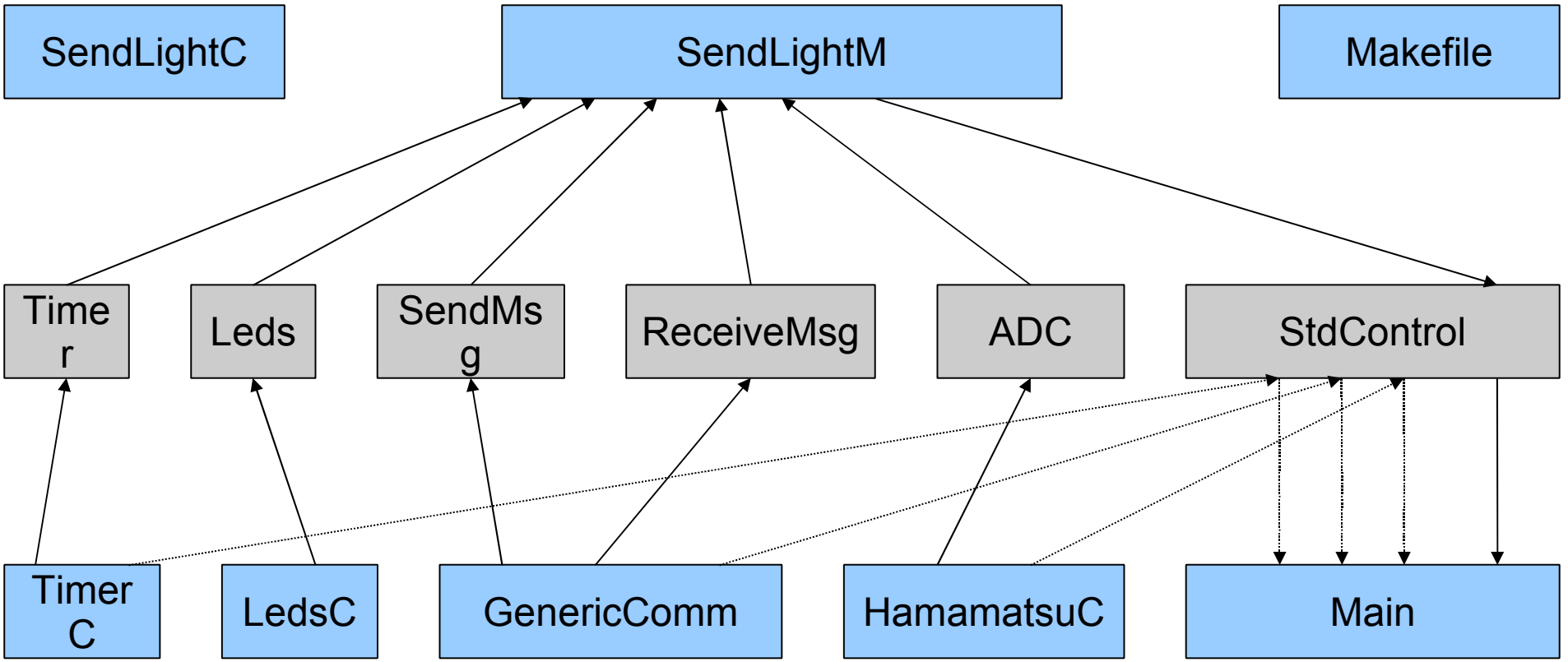
Remarks (1/2)

- The components available on the motes are not the same as the components you use when compiling for the PC
- For the PC: `DemoSensorC.ADC`
- For the motes: `HamamatsuC.PAR` (`PAR` is an alias for `ADC`)
- Use `make telosb` to compile for a mote

Remarks (2/2)

- When using the PC as a simulator, you can use debug messages
- Command `dbg`, syntax similar to `printf` in C
  - `dbg(DBG_USR1, “the value of the light reading for mote %d is %d\n”, TOS_LOCAL_ADDRESS, data);`
- This message is displayed by the simulator if you use `export DBG=usr1`
- You can also use `usr2` or `usr3`

Architecture of `SendLight`



Makefile

```
COMPONENT = SendLightC
include /opt/tinyos-1.x/apps/Makerules
```

SendLight.h

```
#ifndef SENDLIGHT_H
#define SENDLIGHT_H

#define SECOND_DURATION 1024

enum {
    AM_LIGHT_MSG = 100,
};

typedef struct Light_Msg {
    uint16_t value;
} Light_Msg;

#endif
```

SendLightC.nc (1/2)

```
includes SendLight;
configuration SendLightC {
}
implementation {
    components Main, SendLightM, LedsC,
    TimerC, GenericComm as Comm,
    HamamatsuC;

    Main.StdControl -> SendLightM;
    Main.StdControl -> TimerC;
    Main.StdControl -> Comm.Control;
    Main.StdControl -> HamamatsuC;

// ...
```

SendLightC.nc (2/2)

```
    SendLightM.Leds -> LedsC;
    SendLightM.ReceiveLightMsg ->
        Comm.ReceiveMsg[AM_LIGHT_MSG];
    SendLightM.SendMsg ->
        Comm.SendMsg[AM_LIGHT_MSG];
    SendLightM.LightTimer ->
        TimerC.Timer[unique("Timer")];
    SendLightM.Light -> HamamatsuC.PAR;
    // PAR stands for Partial Ambient Radiation
}
```

SendLightM.nc (1/6)

```
includes SendLight;
module SendLightM {
    provides interface StdControl;
    uses {
        interface Leds;
        interface SendMsg as
SendLightMsg;
        interface ReceiveMsg as
            ReceiveLightMsg;
        interface Timer as LightTimer;
        interface ADC as Light;
    }
}
// ...
```

SendLightM.nc (2/6)

```
implementation {
#define YOUR_PC_NUMBER 1
    TOS_Msg message;
    uint16_t lightReading;
    task void SendLightTask() {
        Light_Msg * payload;
        payload = (Light_Msg
*)message.data;
        payload->value = lightReading;
        call SendLightMsg.send(
            YOUR_PC_NUMBER,
            sizeof(Light_Msg), &message);
    }
    //...
}
```

SendLightM.nc (3/6)

```
task void ShowLightTask() {
    uint16_t data;
    data = lightReading >> 7;
    call Leds.redOff();
    call Leds.greenOff();
    call Leds.yellowOff();
    if (data>=1)
        call Leds.redOn();
    if (data>=2)
        call Leds.greenOn();
    if (data>=4)
        call Leds.yellowOn();
}
```

SendLightM.nc (4/6)

```
command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
}
command result_t StdControl.start() {
    if (TOS_LOCAL_ADDRESS==0) {
        call LightTimer.start(
            TIMER_REPEAT,
            SECOND_DURATION/10);
    }
    return SUCCESS;
}
```

SendLightM.nc (5/6)

```
command result_t StdControl.stop() {
    if (TOS_LOCAL_ADDRESS==0) {
        call LightTimer.stop();
    }
    return SUCCESS;
}
event TOS_MsgPtr ReceiveLightMsg.receive(
    TOS_MsgPtr receivedMessage) {
    Light_Msg * payload;
    payload =
        (Light_Msg *)receivedMessage->data;
    lightReading = payload->value;
    post ShowLightTask();
    return receivedMessage;
}
```

SendLightM.nc (6/6)

```
event result_t SendLightMsg.sendDone(
    TOS_MsgPtr sent, result_t result) {
    return SUCCESS;
}
event result_t LightTimer.fired() {
    call Light.getData();
    return SUCCESS;
}
async event result_t Light.dataReady(
    uint16_t data) {
    dbg(DBG_USR1, "Light=%d\n", data);
    lightReading = data;
    post ShowLightTask();
    post SendLightTask();
    return SUCCESS; }
```