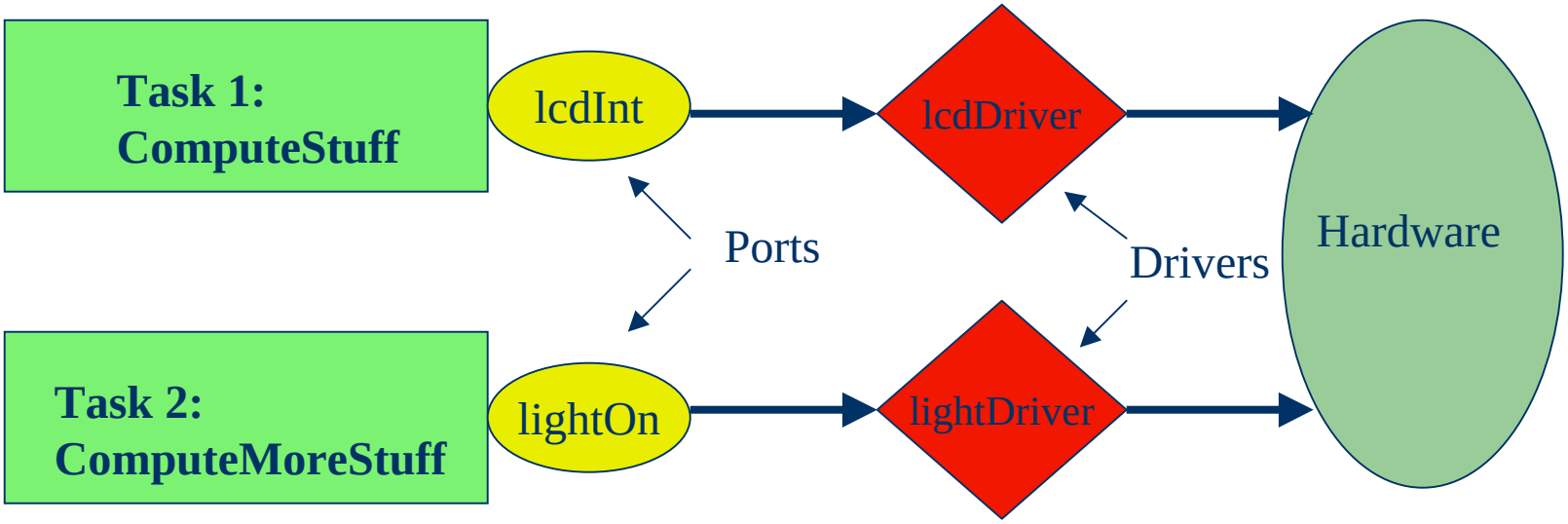
Two Task E-Machine

Darren Liccardo
Mark McKelvin

EE290-O
2/28/2002

Overview

- Functional diagram of our 2-task E-machine



Ports and Drivers

```
//PORTS
int lcdInt = 0;
int lightOn = 0;

//DRIVERS
void lcdDriver ( ) {  lcd_int ( lcdInt );  }

void lightDriver ( ) {
        if ( lightOn != 0 ) ds_active ( &SENSOR_3 );
        else ds_passive ( &SENSOR_3 );
}
```

Tasks

Task 1:

```
void computeStuff ( ) {
        int i;
        for( i=0; i<10000; i+=1 );
        lcdInt += 1;
        exit ( 0 );
}
```

Task 2:

```
void computeMoreStuff ( ) {
        int i;
        for( i=0; i<8000; i+=1 );
        if ( lightOn ) lightOn = 0;
        else lightOn = 1;
        exit ( 0 );
}
```

E-Machine Implementation

```
 void call ( void ( *ptToFunc ) ( ) ) {
                ptToFunc();
 }
void future ( int timeTrigger , void ( *ptToFunc ) ( ) ) {
               msleep ( timeTrigger );
               execi ( ptToFunc , 0 , NULL , 5 , DEFAULT_STACK_SIZE );
               exit ( 0 );
}
 void schedule ( void ( *ptToFunc ) ( ) , int priority) {
                execi ( ptToFunc , 0 , NULL , priority , DEFAULT_STACK_SIZE );
 }
```

E-Machine Program

```
void b1 ( ) {
    call ( &lcdDriver );
    call ( &lightDriver );
    schedule ( &computeStuff , 1 );
    schedule ( &computeMoreStuff , 2 );
    future ( 500 , &b2 );
}
```

```
void b2 ( ) {
    call ( &lightDriver );
    schedule ( &computeMoreStuff , 2 );
    future ( 500 , &b1 );
}
```

Conclusion

- No Kernel Hacking
  - Maintains abstraction
  - Simple Implementation
- Lacks true scheduling mechanism
  - Use prioritized threads instead
    - Effectively a RMA Scheduler
    - Avoids uncontrolled preemption