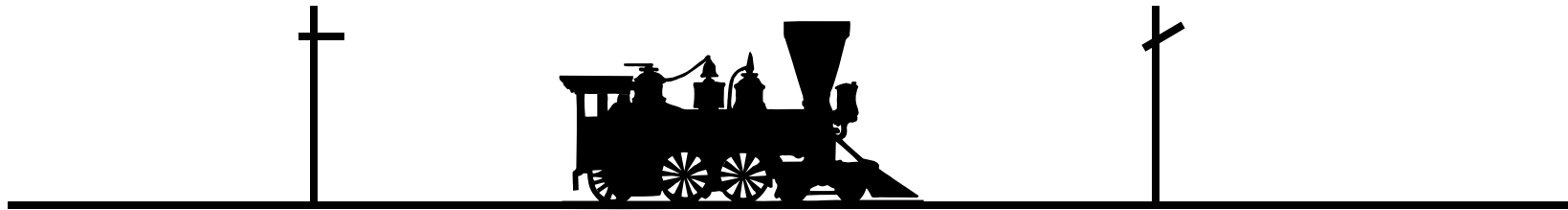

EECE 276

Embedded Systems

Simple task synchronization
Semaphores and shared data

Semaphores Control Tasks



Semaphore operations

- Semaphore: a special variable that controls task execution. When waiting for a semaphore, tasks are blocked. The semaphore has an integer value.
- (Atomic) Operations (logical code):

```
void init(Semaphore& s, int v) { s = v; }
```

```
void P(Semaphore& s) {  
    while (s == 0) BlockTask();  
    s--;  
}
```

```
void V(Semaphore& s) {  
    s++; ReleaseOtherTask();  
}
```

Note: In the RTOS the operations are called OSSemPend() and OSSemPost() – see lab book. Other names are also common, e.g. wait(), signal(), etc.

Semaphore example

- Critical section (max 1 task, anytime)

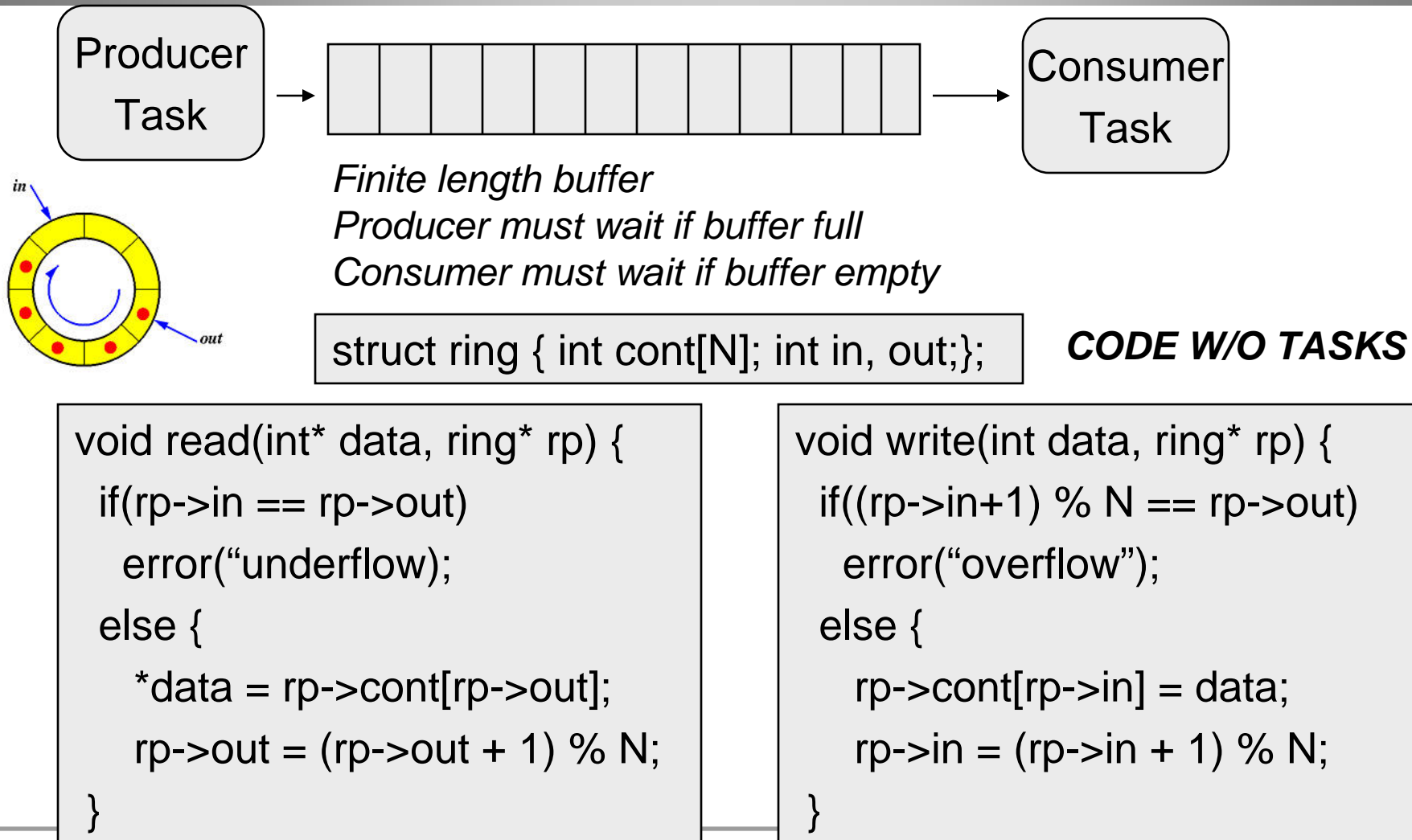
```
Semaphore s;  
...  
init(s,1); // Initialize to 1  
...
```

```
// Task 1  
...  
P(s);  
// Code for critical section  
V(s);  
...
```

```
// Task 2  
...  
P(s);  
// Code for critical section  
V(s);  
...
```

The example shows a “*binary*” semaphore (values: 0,1)

Circular buffer



Circular buffer with semaphores

```
struct ring { int cont[N]; int in, out; Semaphore elements, spaces; };
```

```
void init(ring* rp) { init(rp->elements, 0); init(rp->spaces, N); }
```

```
void read(int* data, ring* rp) {  
    P(rp->elements);  
    *data = rp->cont[rp->out];  
    rp->out = (rp->out + 1) % N;  
    V(rp->spaces);  
}
```

```
void write(int data, ring* rp) {  
    P(rp->spaces);  
    rp->cont[rp->in] = data;  
    rp->in = (rp->in + 1) % N;  
    V(rp->elements);  
}
```

The example shows a *counting* semaphore.

Note: The semaphore is used by a process to signal the other one that it can proceed.

Problems with semaphores

- Forgetting to take the semaphore
 - » Shared data bug
- Forgetting to release the semaphore
 - » Waiting task blocked forever
- Taking the wrong semaphore
 - » Unexpected behavior
- Holding the semaphore for too long
 - » Tasks may miss real-time deadlines

Problems with semaphores

- Deadlock (“deadly embrace”)

```
int a, b;  
Semaphore sA, sB;
```

```
// Task 1  
...  
P(sA); P(sB);  
a = b;  
V(sB);V(sA);  
...
```

```
// Task 2  
...  
P(sB); P(sA);  
b = a;  
V(sA);V(sB);  
...
```

Scenario:

T1: P(sA) ->T2:P(sB)->T2:P(sA) [Blocks] -> T1:P(sB)[Blocks]
