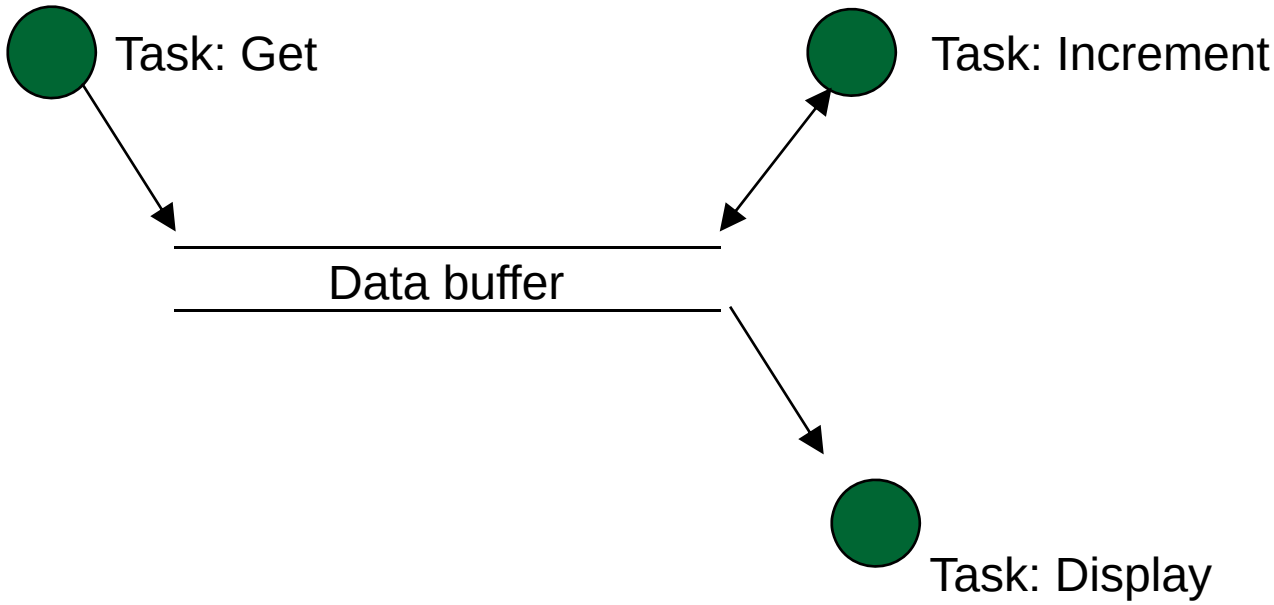
Today

- Discuss final project scheduling
- TCB / Simple kernel in-class exercise… a very very tiny OS
  - Simple kernel 1
    - Finish entering code, compile, run
  - Simple kernel 2
    - Finish entering code, compile, run
    - Extend functionality : add a task, make behavior depend on input, etc
- Course evals

Simple kernel

From James Peckol, Embedded Systems

- Example that kernel runs: 3 asynchronous tasks sharing a common data buffer



Simple kernel, step 1
Init code

```
// From James Peckol, Embedded Systems
// A simple OS kernel - step 1
#include <stdio.h>

// Prototypes for the tasks
void get (void* aNumber); // input task
void increment (void* aNumber); // computation task
void display (void* aNumber); // display task
```

Simple kernel, step 1
main()

```
void main(void) {
  int i=0;                  // queue index
  int data;                 // declare a shared var, data
  int* aPtr = &data;        // point to it

  void(*queue[3])(void*); // declare queue as an array of pointers to
                          // fns taking an arg of type void*
  queue[0]=get;             // enter the tasks into the queue
  queue[1]=increment;
  queue[2]=display;

  while(1) {
    queue[i]((void*)aPtr); // dispatch each task in turn
    i=(i+1)%3;
  }
  return;
}
```

Simple kernel, step 1
tasks

```
void get(void* aNumber) {      // perform input operation
  printf("Enter a number, 0..9 ");
  *(int*) aNumber = getchar();
  getchar(); // discard CR
  *(int*) aNumber -= '0';      // convert to decimal from ASCII
  return;
}

void increment(void* aNumber) {// perform computation
  int* aPtr = (int*) aNumber;
  (*aPtr)++;
  return;
}

void display (void* aNumber) { // perform output operation
  printf("The result is: %d\n",*(int*)aNumber);
  return;
}
```

How can we make Ex 1 more general?

- Add Task Control Blocks (TCBs)

Simple kernel, step 2
Init code

```
// From James Peckol, Embedded Systems
// A simple OS kernel - step 2
#include <stdio.h>

// Prototypes for the tasks
void get (void* aNumber);              // input task
void increment (void* aNumber);        // computation task
void display (void* aNumber);          // display task

// Declare a TCB structure
typedef struct {
  void* taskDataPtr;
  void (*taskPtr)(void*);
}
TCB;
```

Simple kernel, step 2
main(), part 1

```
void main(void) {
  int i=0;                // queue index
  int data;               // declare a shared var, data
  int* aPtr = &data;      // point to it
  TCB* queue[3]; // declare queue as an array of ptrs to TCBs

  // Declare some TCBs
  TCB inTask;
  TCB compTask;
  TCB outTask;
  TCB* aTCBPtr;

  // Initialize the TCBs
  inTask.taskDataPtr=(void*)&data;
  inTask.taskPtr=get;

  compTask.taskDataPtr=(void*)&data;
  compTask.taskPtr=increment;

  outTask.taskDataPtr=(void*)&data;
  outTask.taskPtr=display;
```

Simple kernel, step 2
main(), part 2

```
  // initialize the task queue
  queue[0]= &inTask;
  queue[1]= &compTask;
  queue[2]= &outTask;

  // schedule and dispatch the tasks
  while(1) {
    aTCBPtr=queue[i];
    aTCBPtr->taskPtr( (aTCBPtr->taskDataPtr) );
    i=(i+1)%3;
  }
  return;
}
```

Simple kernel, step 2
tasks [unchanged from step 1]

```
void get(void* aNumber) {      // perform input operation
  printf("Enter a number, 0..9 ");
  *(int*) aNumber = getchar();
  getchar();                           // discard CR
  *(int*) aNumber -= '0';              // convert to decimal from ASCII
  return;
}

void increment(void* aNumber) {        // perform computation
  int* aPtr = (int*) aNumber;
  (*aPtr)++;
  return;
}

void display (void* aNumber) {         // perform output operation
  printf("The result is: %d\n",*(int*)aNumber);
  return;
}
```

What's missing?

- Preemption, via interrupts
- This would prevent any task from hogging the system
- You could actually do this on your blimps…