Section Solution

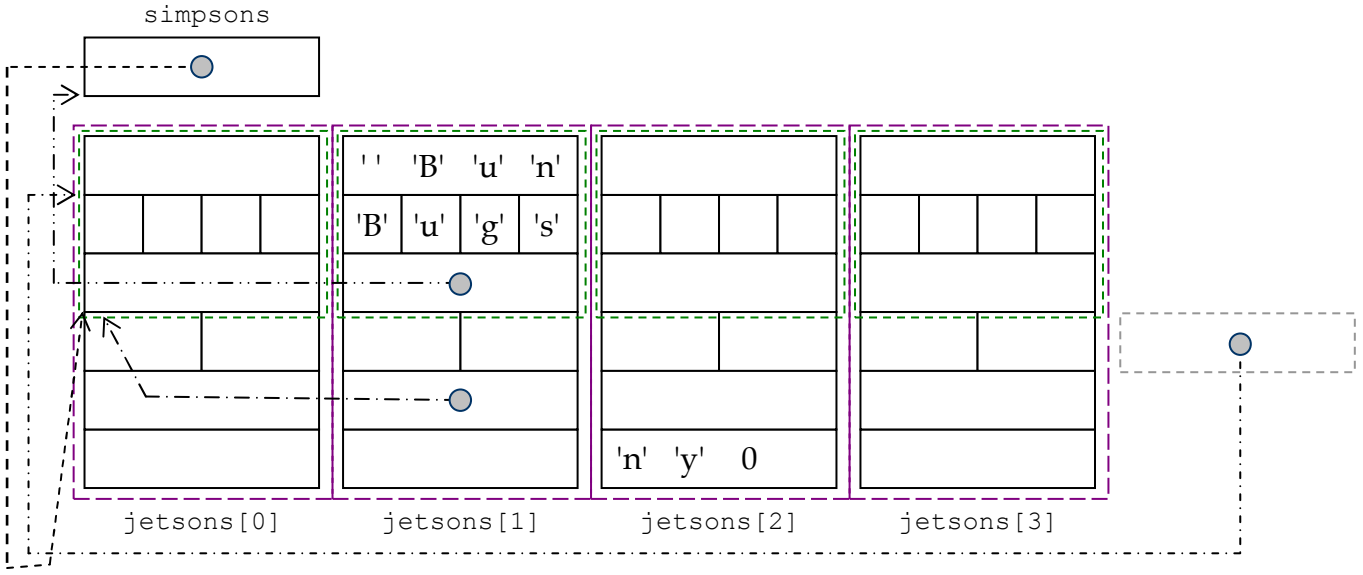
Problem 1: Meet The Flintstones

```
typedef struct rubble {
   int betty;
   char barney[4];
   struct rubble *bammbamm;
} rubble;

typedef struct {
   short *wilma[2];
   short fred[2];
   rubble dino;
} flintstone;


rubble *simpsons;
flintstone jetsons[4];

simpsons = &jetsons[0].dino;
jetsons[1].wilma[3] = (short *) &simpsons;
strcpy(simpsons[2].barney, "Bugs Bunny");
((flintstone *)(jetsons->fred))->dino.bammbamm = simpsons;
*(char **)jetson[4].fred = simpsons->barney + 4;
```



Problem 2: Scheme

```
/**
 * Traverses a properly structured list, and returns the ordered
 * concatenation of all strings, including those in nested sublists.
 *
 * When applied to the two lists drawn above, the following strings
 * would be returned:
 *
 *    ConcatAll(gameThree) would return "YankeesDiamondbacks"
 *    ConcatAll(nestedNumbers) would return "onethreesix"
 */

typedef enum {
   Integer, String, List, Nil
} nodeType;

static char *ConcatStrings(const char *first, const char *second)
{
   char *result = malloc(strlen(first) + strlen(second) + 1);
   strcpy(result, first);
   strcat(result, second);
   return result;
}

char *ConcatAll(nodeType *list)
{
   switch (*list) {
      case Integer:
      case Nil:      return strdup("");
      case String:   return strdup((char *)(list + 1));
   }

   nodeType **lists = (nodeType **)(list + 1);
   char *front = ConcatAll(lists[0]);
   char *back = ConcatAll(lists[1]);
   char *result = ConcatStrings(front, back);
   free(front);
   free(back);
   return result;
}
```