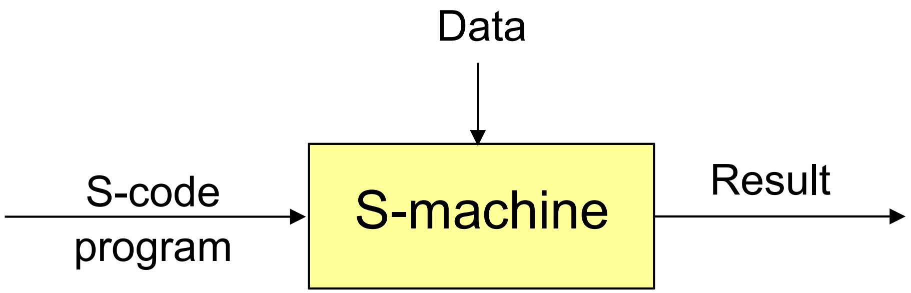
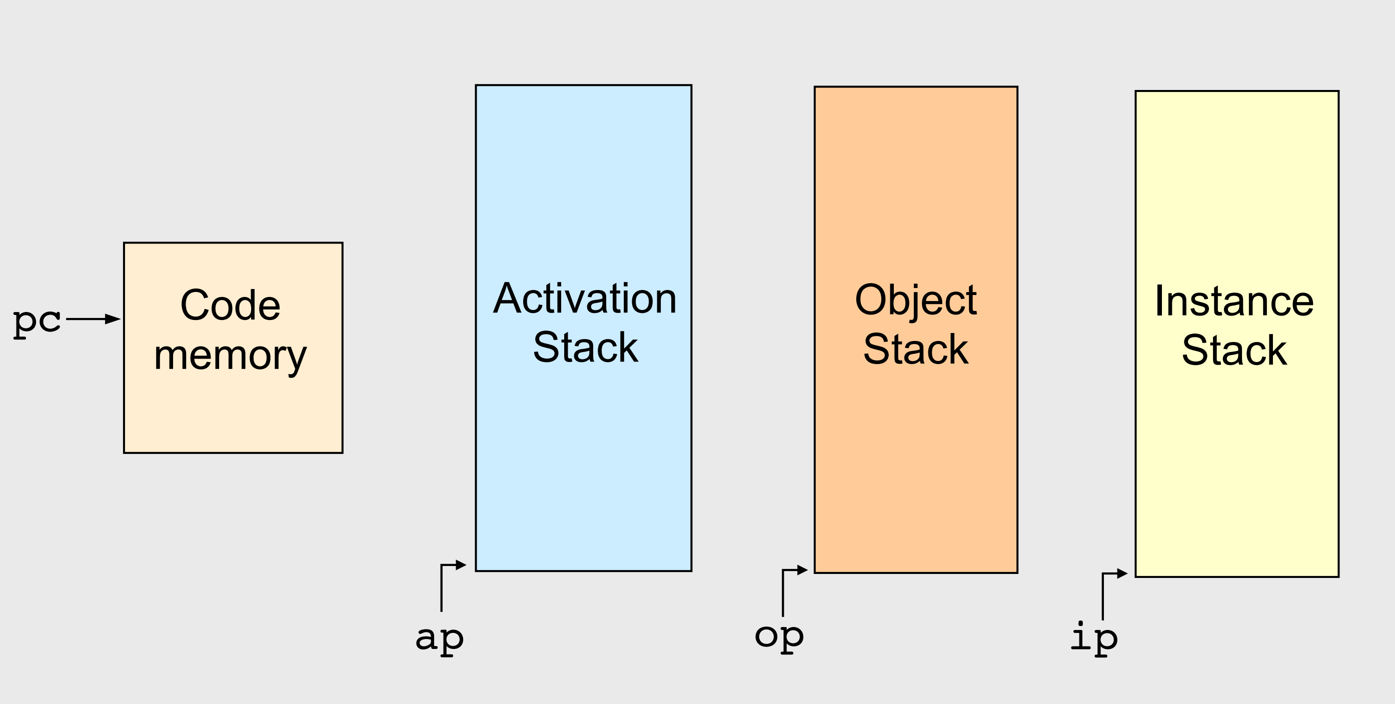
Abstract Machine



- Structure of loaded S-code program:

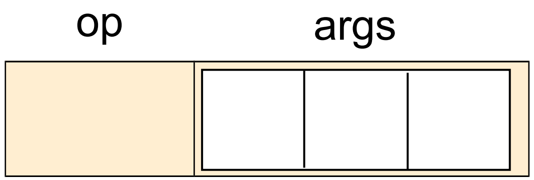
```
SCODE size

PUSH num-formals num-locals -1
  GOTO ^prog
POP
HALT

⟨prog⟩

⟨f1⟩

⟨f2⟩

...
```

Architecture of S-machine



Code Memory

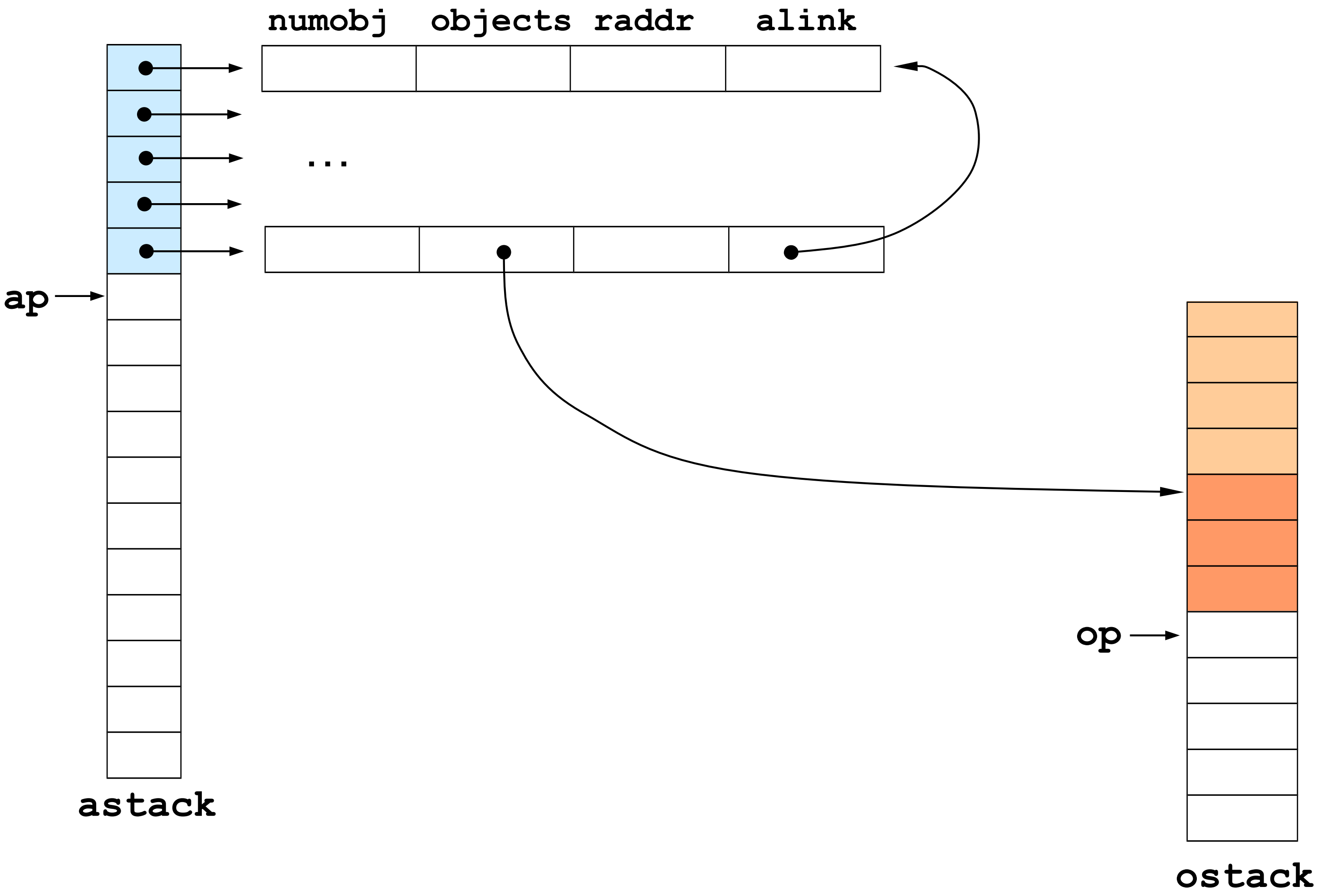
- Scode:



```
typedef struct
{
    Operator op;
    Lexval args[MAXARGS];
} Scode;

Scode *prog;
```

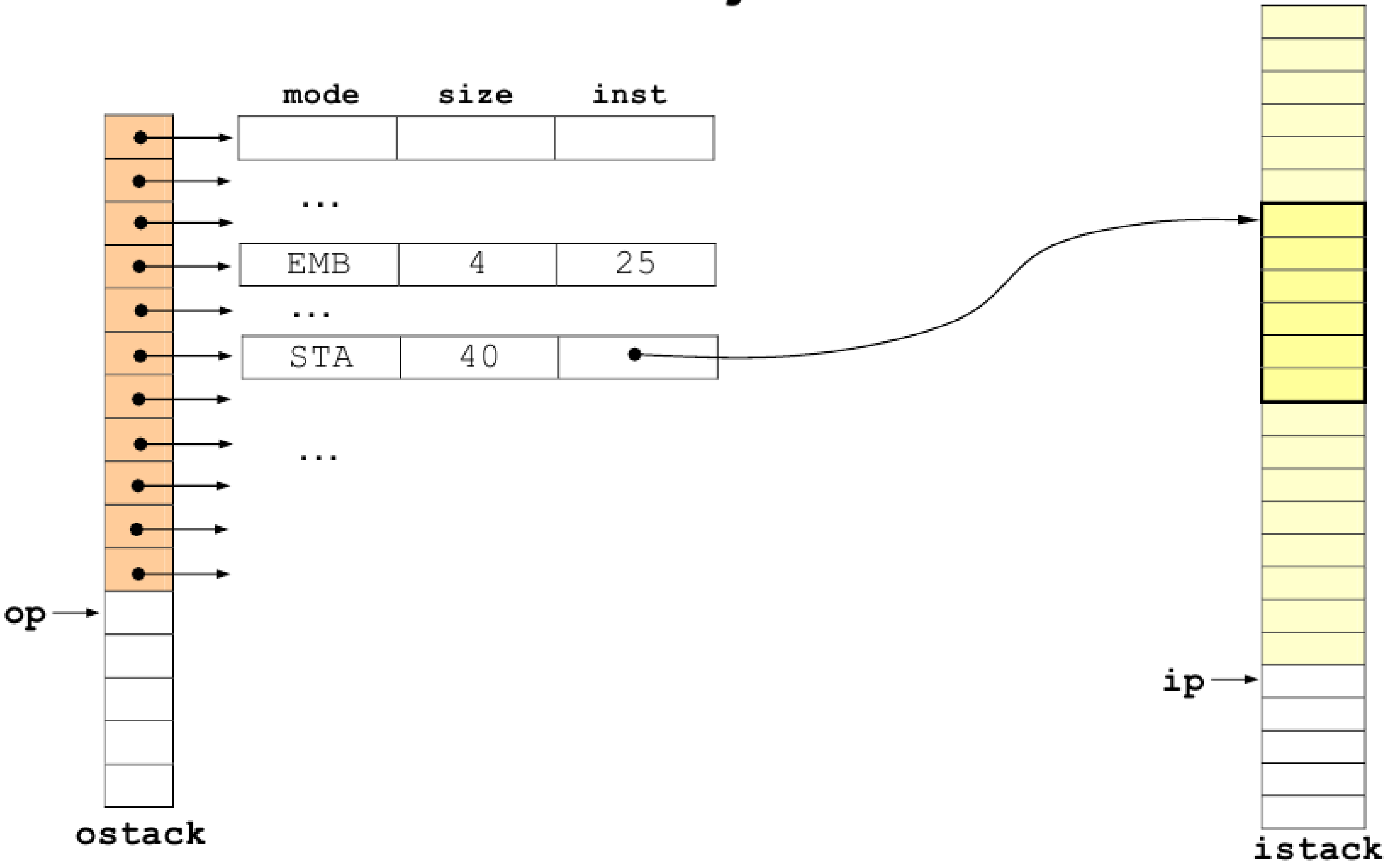
- Allocated at machine initialization → `SCODE` *size*

Activation Stack



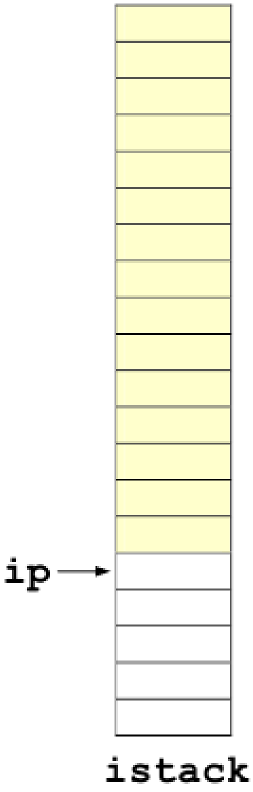
```
typedef struct adescr {int numobj; Odescr *objects; int raddr; struct adescr *alink;} Adescr;
Adescr **astack;
```

Object Stack



```
typedef enum {EMB, STA} Mode;
typedef struct {Mode mode; int size; Lexval inst;} Odescr;
Odescr **ostack;
```

Instance Stack



```
char *istack;
```

main()

```
main(int argc, char *argv[])
{
    Scode *stat;

    start_machine();
    while((stat = &prog[pc++])->op != S_HALT)
        exec(stat);
    end_machine();
}
```

Initialization of Abstract Machine

```
extern Scode *prog;
extern int pc;
Adescr **astack;
Odescr **ostack;
char *istack;
int asize, osize, isize;

int ap, op, ip;

long size_allocated = 0,
     size_deallocated = 0;

void start_machine()
{
    load_scode();
    pc = ap = op = ip = 0;
    astack = (Adescr**)newmem(sizeof(Adescr*)*ASTACK_UNIT);
    asize = ASTACK_UNIT;
    ostack = (Odescr**)newmem(sizeof(Odescr*)*OSTACK_UNIT);
    osize = OSTACK_UNIT;
    istack = (char*)newmem(ISTACK_UNIT);
    isize = ISTACK_UNIT;
}
```

Termination of Abstract Machine

```
void end_machine()
{
    freemem((char*)prog, sizeof(Scode)*code_size);
    freemem((char*)astack, sizeof(Adescr*)*asize);
    freemem((char*)ostack, sizeof(Odescr*)*osize);
    freemem(istack, isize);
    printf("Program executed without errors\n");
    printf("Allocation: %ld bytes\n", size_allocated);
    printf("Deallocation: %ld bytes\n", size_deallocated);
    printf("Residue: %ld bytes\n", size_allocated - size_deallocated);
}
```

Allocation and Deallocation of Memory

```
void *newmem(int size)
{
    void *p;

    if((p = malloc(size)) == NULL)
        machine_error("Failure in memory allocation");
    size_allocated += size;
    return p;
}
```

```
void freemem(char *p, int size)
{
    free(p);
    size_deallocated += size;
}
```

Allocation and Deallocation of Activation Record

```
Adescr *push_astack()
{
    Adescr **old_astack;
    int i;

    if(ap == asize)
    {
      old_astack = astack;
      astack = (Adescr**) newmem(sizeof(Adescr*)*(asize + ASTACK_UNIT));
      for(i = 0; i < asize; i++)
        astack[i] = old_astack[i];
      freemem((char*)old_astack, sizeof(Adescr*)*asize);
      asize += ASTACK_UNIT;
    }
    return (astack[ap++] = (Adescr*)newmem(sizeof(Adescr)));
}
```

```
void pop_astack()
{
    if(ap == 0) machine_error("pop_adescr()");
    freemem((char*)astack[--ap], sizeof(Adescr));
}
```

Instruction Execution

return address

```
Scode *prog;
int pc;

void exec(Scode *stat)
{
  switch(stat->op)
  {
  case S_PUSH: exec_push(stat->args[0].ival, stat->args[1].ival, stat->args[2].ival, pc+1); break;
  case S_GOTO: exec_goto(stat->args[0].ival); break;
  case S_POP: exec_pop(); break;
  case S_NEW: exec_new(stat->args[0].ival); break;
  case S_NEWS: exec_news(stat->args[0].ival); break;
  case S_LDC: exec_ldc(stat->args[0].cval); break;
  case S_LDI: exec_ldi(stat->args[0].ival); break;
  case S_LDS: exec_lds(stat->args[0].sval); break;
  case S_LDR: exec_ldr(stat->args[0].rval); break;

  ...

  case S_RETURN: exec_return(); break;
  default: machine_error("Unknown operator"); break;
  }
}
```

Jumps

```
void exec_goto(int addr)
{
    pc = addr;
}

void exec_jmp(int offset)
{
    pc += offset-1;
}

void exec_jmf(int offset)
{
    if(!pop_bool())
      pc += offset-1;
}

void exec_return()
{
    pc = top_astack()->raddr;
}
```

Miscellaneous

```
void exec_iplus()
{
    int n, m;

    n = pop_int();
    m = pop_int();
    push_int(m+n);
}
```

```
void exec_igt()
{
    int n, m;

    n = pop_int();
    m = pop_int();
    push_bool(m>n);
}
```

```
void exec_new(int size)
{
    Odescr *po;

    po = push_ostack();
    po->mode = EMB;
    po->size = size;
}
```