

CS5600: Computer Systems

Spring 2020

Goals

- Study some of the fundamental problems and solutions of OS
- Project based course

Course Setup

- Lectures, notes and readings provide an exposition of the problem(s) and their solutions
- Course staff treats students as professional developers

Course Evaluation

- 4 projects individual projects
- You are professional developers
 - Clear and Clean and documented source code
 - Documentation for your solution
 - Design and reasons for your choice
 - Analysis (space, time)
 - Concurrency (starvation, liveness)

Hard Work

- There is a lot of work!
- We will expose some of the inner workings
 - what most classes up to know relied upon as “black box”
- Low lever (architecture, assembly)
- Interesting and hard problems
 - Concurrency, Garbage Collection etc.

Assumptions

- Medium level familiarity with C
 - structs, pointers, malloc, free etc.
 - See last Fall's class web site (<https://course.ccs.neu.edu/cs5600f17>)
- Basic understanding of shell and OS
 - process, pids, pipes, files, directories etc.
- Basic understanding of linking loading and execution
 - object files, libs, linking, execution stack etc.

Logistics

- Class Web Site:
 - <https://course.ccs.neu.edu/cs5600f20/>
- Piazza
 - <https://piazza.com/class/ke84eupbcvi3rk>

Introduce Yourself

- Name
- Why are you taking this course (need the credit is fine)?
- Experience with C or low level languages (scale 1-10)?
- Mac/PC

Compile and Run

loop.c

```
int main(void) {  
    int sum = 0;  
    int i;  
  
    for (i = 0; i < 10; i++){  
        sum = sum + i;  
    }  
    return sum;  
}
```

1.Compile
`gcc -O0 loop.c`

2.Run
`./a.out`

Compile

loop.c

```
int main(void) {  
    int sum = 0;  
    int i;  
  
    for (i = 0; i < 10; i++){  
        sum = sum + i;  
    }  
    return sum;  
}
```

1. Compile
gcc -O0 loop.c

loop.c

a.out

Compile

loop.c

```
int main(void) {  
    int sum = 0;  
    int i;  
  
    for (i = 0; i < 10; i++){  
        sum = sum + i;  
    }  
    return sum;  
}
```

1. Compile
gcc -O0 loop.c

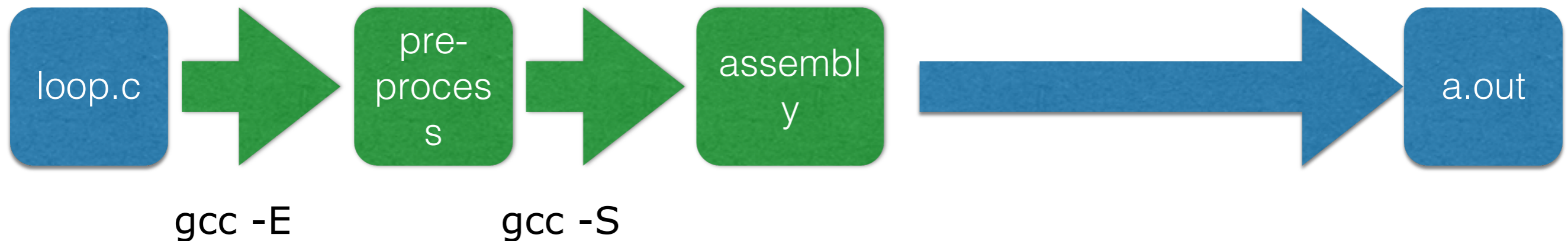


Compile

loop.c

```
int main(void) {  
    int sum = 0;  
    int i;  
  
    for (i = 0; i < 10; i++){  
        sum = sum + i;  
    }  
    return sum;  
}
```

1. Compile
gcc -O0 loop.c

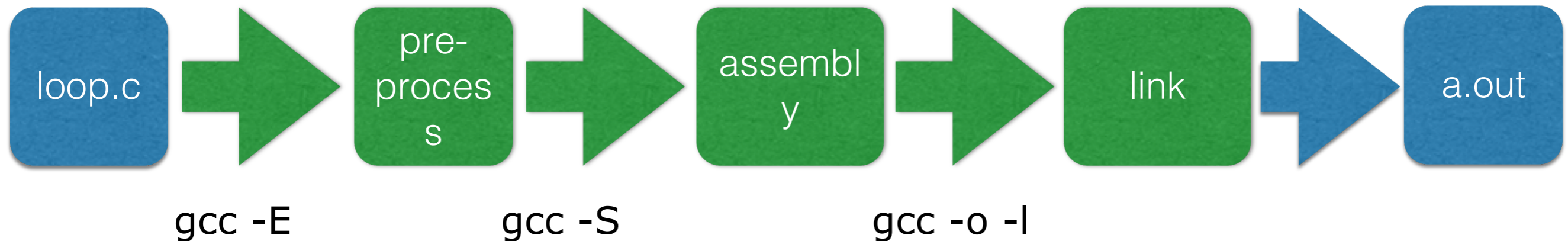


Compile

loop.c

```
int main(void) {  
    int sum = 0;  
    int i;  
  
    for (i = 0; i < 10; i++){  
        sum = sum + i;  
    }  
    return sum;  
}
```

1. Compile
gcc -O0 loop.c

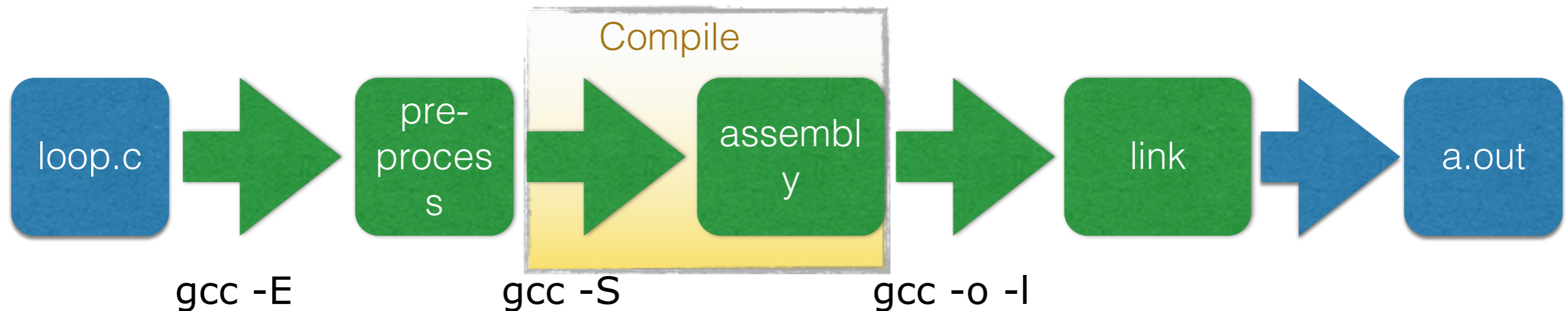


Compile

loop.c

```
int main(void) {  
    int sum = 0;  
    int i;  
  
    for (i = 0; i < 10; i++){  
        sum = sum + i;  
    }  
    return sum;  
}
```

1. Compile
gcc -O0 loop.c

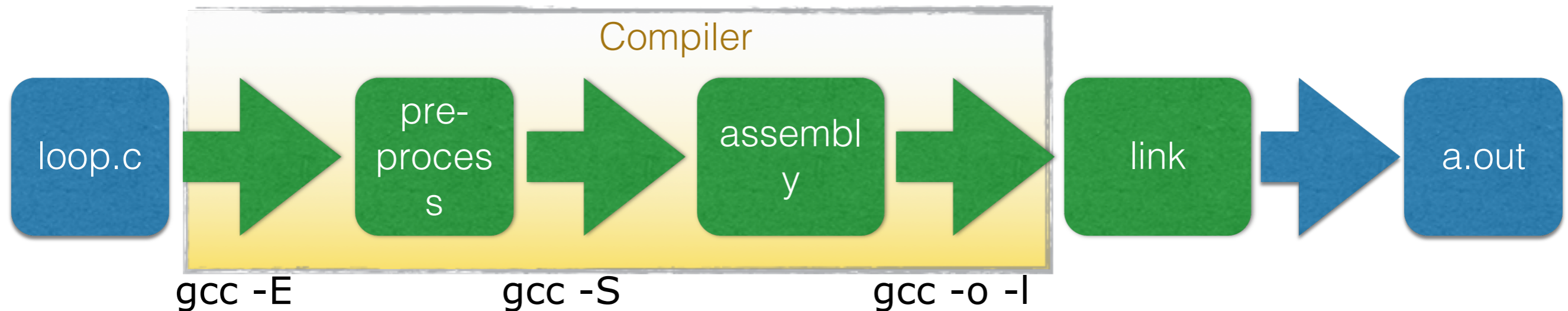


Compile

loop.c

```
int main(void) {  
    int sum = 0;  
    int i;  
  
    for (i = 0; i < 10; i++){  
        sum = sum + i;  
    }  
    return sum;  
}
```

1. Compile
gcc -O0 loop.c



gcc -O0 -E loop.c

loop.c

```
int main(void) {
    int sum = 0;
    int i;

    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}
```

```
# 1 "loop.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "loop.c"
int main(void) {
    int sum = 0;
    int i;

    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}
```

gcc -O0 -S loop.c

loop.c

```
int main(void) {
    int sum = 0;
    int i;

    for (i = 0; i < 10; i++){
        sum = sum + i;
    }
    return sum;
}
```

With some debugging instructions removed

```
.file "loop.c"
.text
.globl     main
.type main, @function

main:
.LFB0:
    pushl %ebp
    movl  %esp, %ebp
    subl  $16, %esp
    movl  $0, -4(%ebp)
    movl  $0, -8(%ebp)
    jmp   .L2

.L3:
    movl  -8(%ebp), %eax
    addl  %eax, -4(%ebp)
    addl  $1, -8(%ebp)

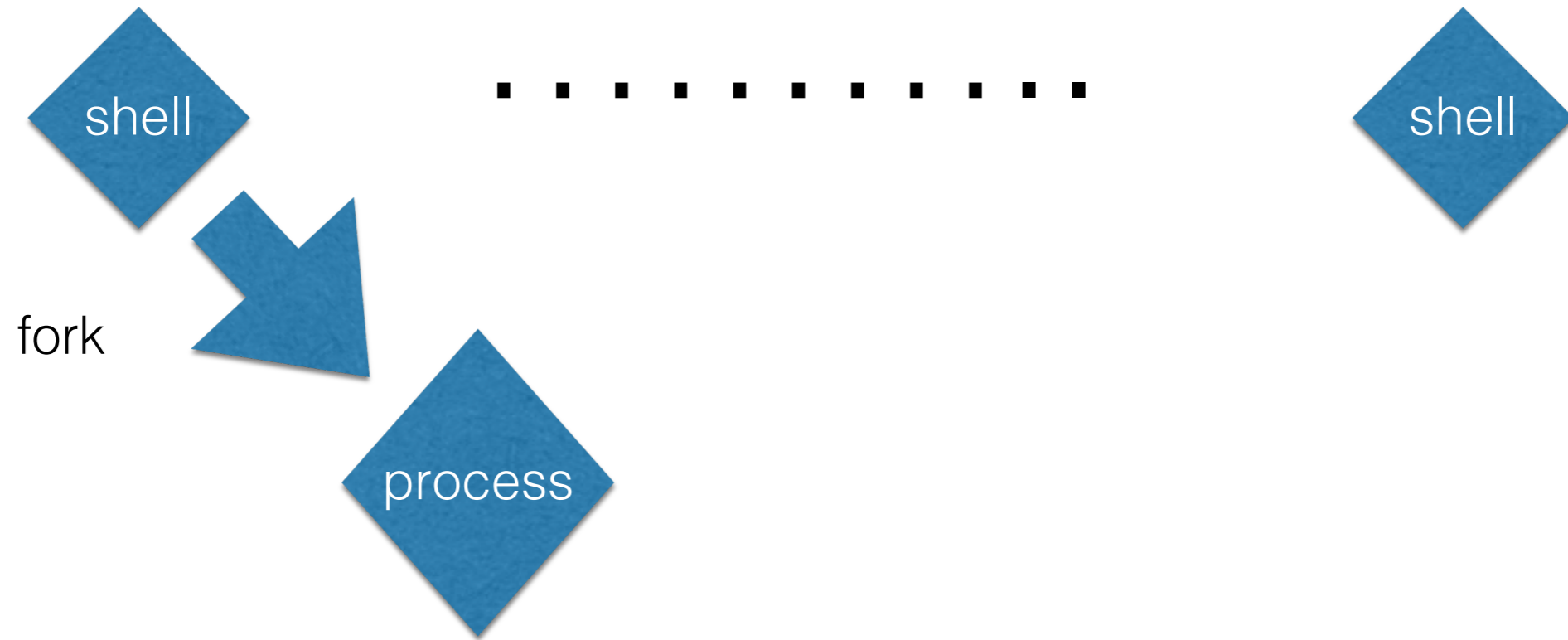
.L2:
    cmpl  $9, -8(%ebp)
    jle   .L3
    movl  -4(%ebp), %eax
    leave
    ret

.LFE0:
.size main, .-main
.ident     "GCC: (Debian 4.9.2-10) 4.9.2"
.section   .note.GNU-stack,"",@progbits
```

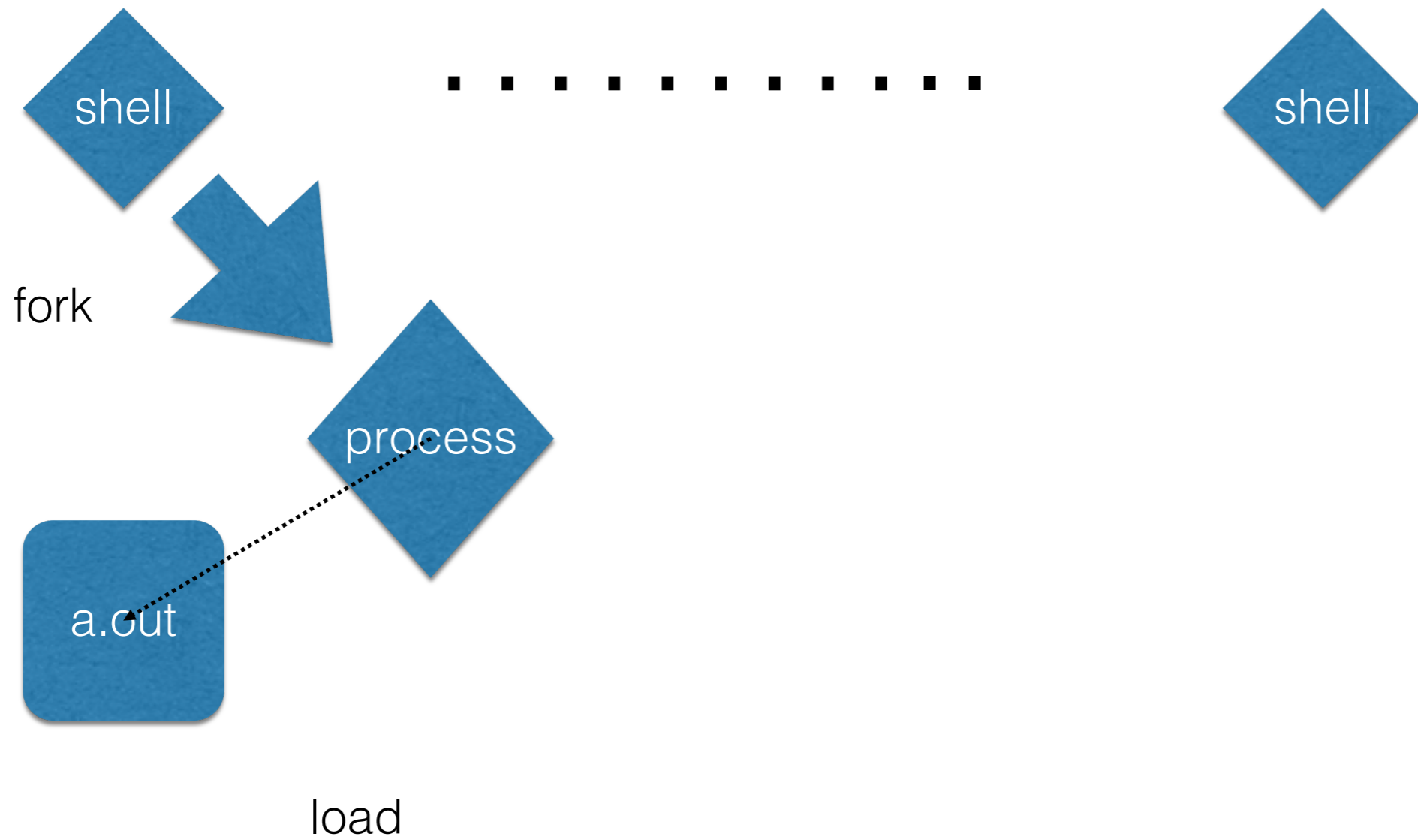
Run



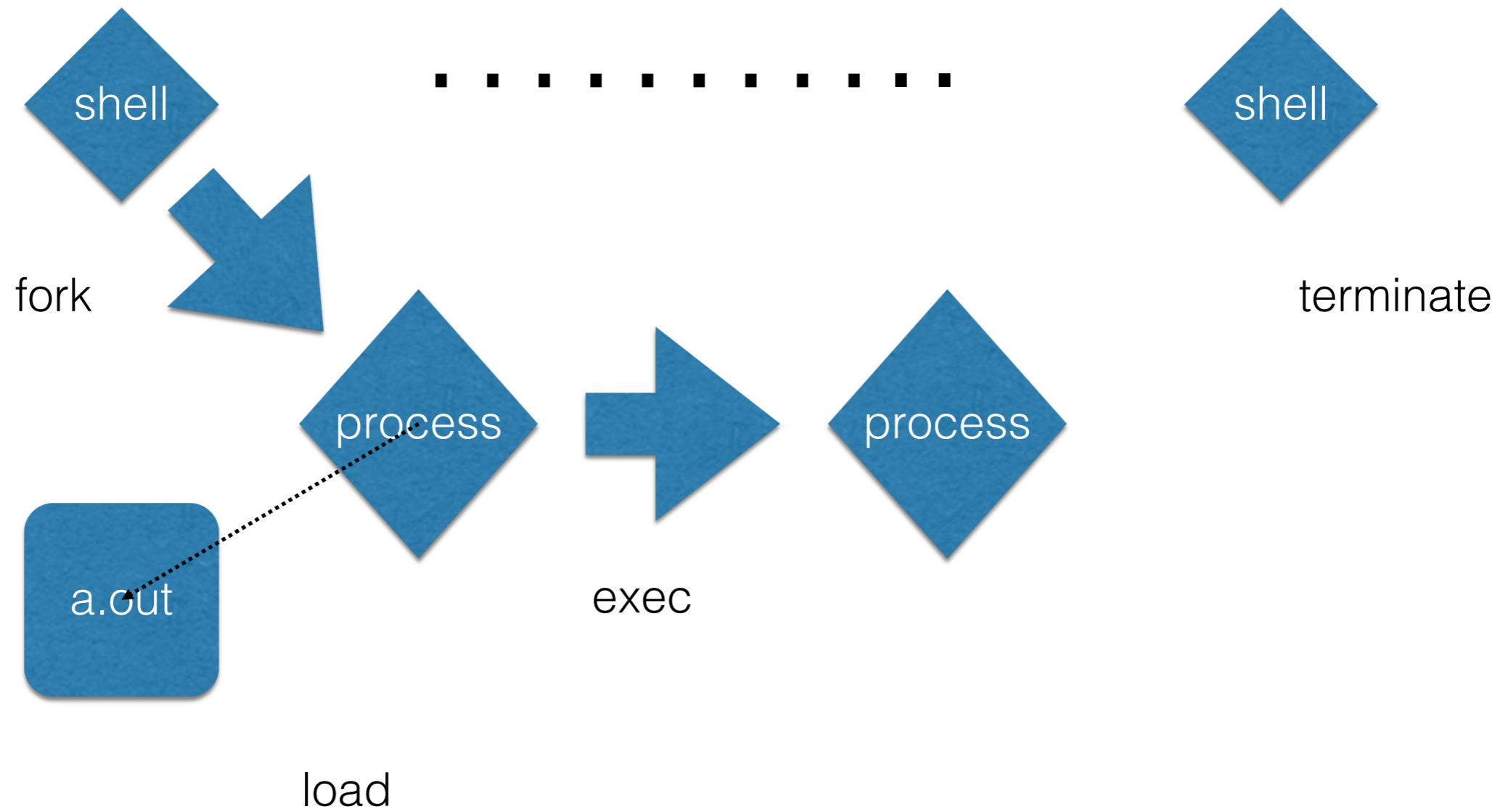
Run



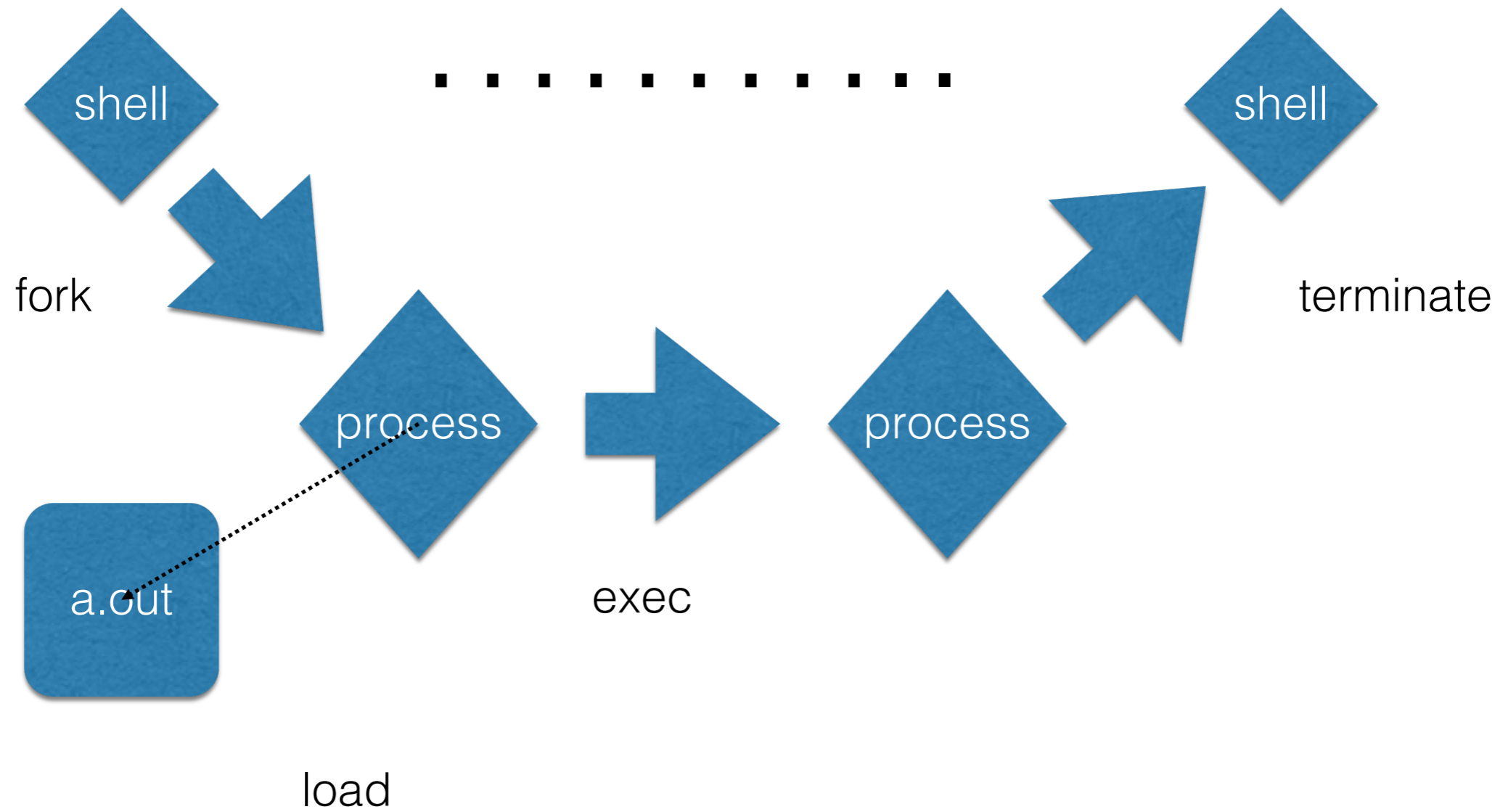
Run



Run



Run



Run

loop.c

```
int main(void) {  
    int sum = 0;  
    int i;  
  
    for (i = 0; i < 10; i++){  
        sum = sum + i;  
    }  
    return sum;  
}
```

1.Run
./a.out

What type of a file is a.out?

Run

loop.c

```
int main(void) {  
    int sum = 0;  
    int i;  
  
    for (i = 0; i < 10; i++){  
        sum = sum + i;  
    }  
    return sum;  
}
```

1.Run
./a.out

What type of a file is a.out?

```
therapon@vdebian:~/Lectures/01$ file a.out
```

```
a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=f725959a7360fc04d6cc7190ccc4a0428946c217, not stripped
```

ELF

- **E**xecutable and **L**inkable **F**ormat
- A way to organize the information in the file
- One header and data
- Data is made up of
 - Program header table, describes 0 or more segments
 - Section header table, describe 0 or more sections
 - Data referred to by entries in the program and section header tables

ELF

```
therapon@vdebian:~/Lectures/01$ readelf -h a.out
```

```
ELF Header:
```

```
  Magic:  7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00
```

```
  Class:           ELF32
```

```
  Data:           2's complement, little endian
```

```
  Version:        1 (current)
```

```
  OS/ABI:         UNIX - System V
```

```
  ABI Version:    0
```

```
  Type:           EXEC (Executable file)
```

```
  Machine:        Intel 80386
```

```
  Version:        0x1
```

```
  Entry point address: 0x80482d0
```

```
  Start of program headers: 52 (bytes into file)
```

```
  Start of section headers: 3616 (bytes into file)
```

```
  Flags:          0x0
```

```
  Size of this header: 52 (bytes)
```

```
  Size of program headers: 32 (bytes)
```

```
  Number of program headers: 8
```

```
  Size of section headers: 40 (bytes)
```

```
  Number of section headers: 30
```

```
  Section header string table index: 27
```

The “black boxes”

- How do we go from file to running program?
- How do we manage a running program?
 - stack, access to files, devices, memory, etc.
- How do we provide isolation and communication between programs?
 - inter-process communication, privileged access to devices
- How do we get the OS to run and who manages the OS while running?

It's all 0s and 1s

- We will see
 - how a machine boots up
 - how processes are created and managed
 - how code is loaded, executed and managed (memory management)
 - access to hardware features (shared, privileged etc)
 - filesystem