Übungen Betriebssysteme (BS)

U6 - Sicherheit

Alwin Berger

TU Dortmund - AG Systemsoftware

Agenda

- Besprechung von “A5 - Dateioperationen”
- UNIX, C und Sicherheit
  - Gefährliche Funktionen in C
  - *Buffer Overflows*
  - Schutzmaßnahmen
- Alte Klausuraufgabe zu I/O-Scheduling

Foliensatz A5-Besprechung

“Gefährliche” Funktionen in C

- `scanf()`
- `gets()` (verhält sich ähnlich wie `scanf("%s", buf)`)
- `strcpy()`
- `strcat()`
- `sprintf()`

“Gefährliche” Funktionen in C

- `scanf()`
- `gets()` (verhält sich ähnlich wie `scanf("%s", buf)`)
- `strcpy()`
- `strcat()`
- `sprintf()`

Was haben diese Funktionen gemeinsam?

- Alle schreiben ab einer bestimmten Adresse in den Speicher

“Gefährliche” Funktionen in C

- `scanf()`
- `gets()` (verhält sich ähnlich wie `scanf("%s", buf)`)
- `strcpy()`
- `strcat()`
- `sprintf()`

Was haben diese Funktionen gemeinsam?

- Alle schreiben ab einer bestimmten Adresse in den Speicher
- Zur Laufzeit kann nicht entschieden werden, ob die Abburchbedingung irgendwann erfüllt wird.

“Gefährliche” Funktionen in C - `gets()`

```c
int ask_passwd(void) {
    char buf[8];
    printf("Passwort:");
    gets(buf);      // <--
    if (check_passwd(buf)) {
        return 1;
    } else {
        printf("Falsches Passwort!\n");
        return 0;
    }
}
```

```
studi@bsvm:~$ gcc -o login login.c
/tmp/ccoMGotc.o: In function 'ask_passwd': login.c:(.text+0x19): warning:
    the 'gets' function is dangerous and should not be used.
```

“Gefähliche” Funktionen in C - `scanf()`

```c
int ask_passwd(void) {
    char buf[8];
    printf("Passwort:");
    scanf(buf);     // <--
    if (check_passwd(buf)) {
        return 1;
    } else {
        printf("Falsches Passwort!\n");
        return 0;
    }
}
```

“Gefähliche” Funktionen in C - `scanf()`

```c
int ask_passwd(void) {
    char buf[8];
    printf("Passwort:");
    scanf(buf);     // <--
    if (check_passwd(buf)) {
        return 1;
    } else {
        printf("Falsches Passwort!\n");
        return 0;
    }
}
```

- Übersetzt **ohne** eine Warnung!

Pufferüberlauf

```
void start_shell(void) {
    gid_t gid = getegid();
    uid_t uid = geteuid();
    if (setresgid(gid, gid, gid))
        perror("setresgid");
    if (setresuid(uid, uid, uid))
        perror("setresuid");
    printf("Starte Shell als Nutzer %d (uid=%d,gid=%d)\n", uid, uid, gid);
    execlp("/bin/bash", "/bin/bash", NULL);
}

int main(void) {
    if (ask_passwd())
        start_shell();
    return 0;
}
```

Pufferüberlauf

```
void start_shell(void) {
    gid_t gid = getegid();
    uid_t uid = geteuid();
    if (setresgid(gid, gid, gid))
        perror("setresgid");
    if (setresuid(uid, uid, uid))
        perror("setresuid");
    printf("Starte Shell als Nutzer %d (uid=%d,gid=%d)\n", uid, uid, gid);
    execlp("/bin/bash", "/bin/bash", NULL);
}

int main(void) {
    if (ask_passwd())       // <--
        start_shell();
    return 0;
}
```

Pufferüberlauf

```
void start_shell(void) {
    gid_t gid = getegid();
    uid_t uid = geteuid();
    if (setresgid(gid, gid, gid))
        perror("setresgid");
    if (setresuid(uid, uid, uid))
        perror("setresuid");
    printf("Starte Shell als Nutzer %d (uid=%d,gid=%d)\n", uid, uid, gid);
    execlp("/bin/bash", "/bin/bash", NULL);
}

int main(void) {
    if (ask_passwd())
        start_shell();      // <--
    return 0;
}
```

Pufferüberlauf

```
void start_shell(void) {
    gid_t gid = getegid();      // <--
    uid_t uid = geteuid();      // <--
    if (setresgid(gid, gid, gid))
        perror("setresgid");
    if (setresuid(uid, uid, uid))
        perror("setresuid");
    printf("Starte Shell als Nutzer %d (uid=%d,gid=%d)\n", uid, uid, gid);
    execlp("/bin/bash", "/bin/bash", NULL);
}

int main(void) {
    if (ask_passwd())
        start_shell();
    return 0;
}
```

- Abfragen der effektiven Nutzer- und Gruppen-ID. Entspricht Nutzer und Gruppe der Datei.

Pufferüberlauf

```
void start_shell(void) {
    gid_t gid = getegid();
    uid_t uid = geteuid();
    if (setresgid(gid, gid, gid))       // <--
        perror("setresgid");
    if (setresuid(uid, uid, uid))       // <--
        perror("setresuid");
    printf("Starte Shell als Nutzer %d (uid=%d,gid=%d)\n", uid, uid, gid);
    execlp("/bin/bash", "/bin/bash", NULL);
}

int main(void) {
    if (ask_passwd())
        start_shell();
    return 0;
}
```

- Setzt die Nutzer und Gruppen-ID auf die Eigentümer der Datei

Pufferüberlauf

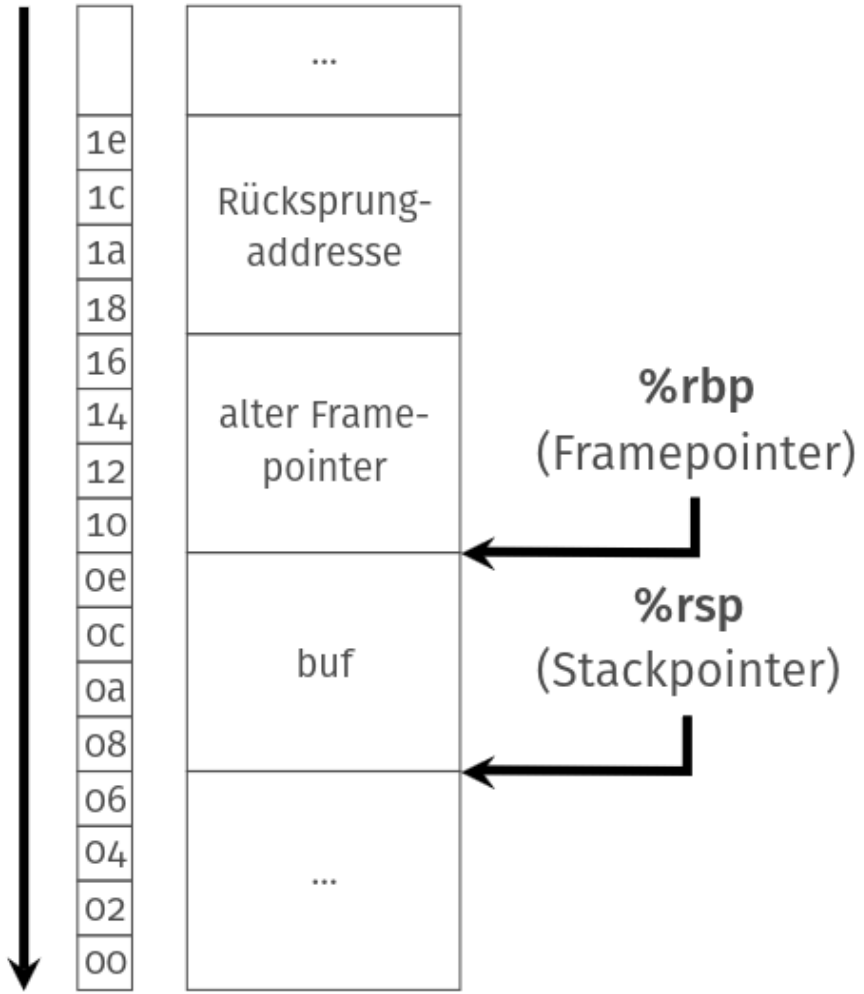
```
void start_shell(void) {
    gid_t gid = getegid();
    uid_t uid = geteuid();
    if (setresgid(gid, gid, gid))
        perror("setresgid");
    if (setresuid(uid, uid, uid))
        perror("setresuid");
    printf("Starte Shell als Nutzer %d (uid=%d,gid=%d)\n", uid, uid, gid);
    execlp("/bin/bash", "/bin/bash", NULL);     // <--
}

int main(void) {
    if (ask_passwd())
        start_shell();
    return 0;
}
```

```
studi@bsvm:~$ ./login
Passwort:12345678abcdefgh12345678
Falsches Passwort
Segmentation Fault
```

Was ist passiert?

```c
int ask_passwd(void) {
    char buf[8];
    printf("Passwort:");
    scanf("%s", buf);       // <--
    // ...
```

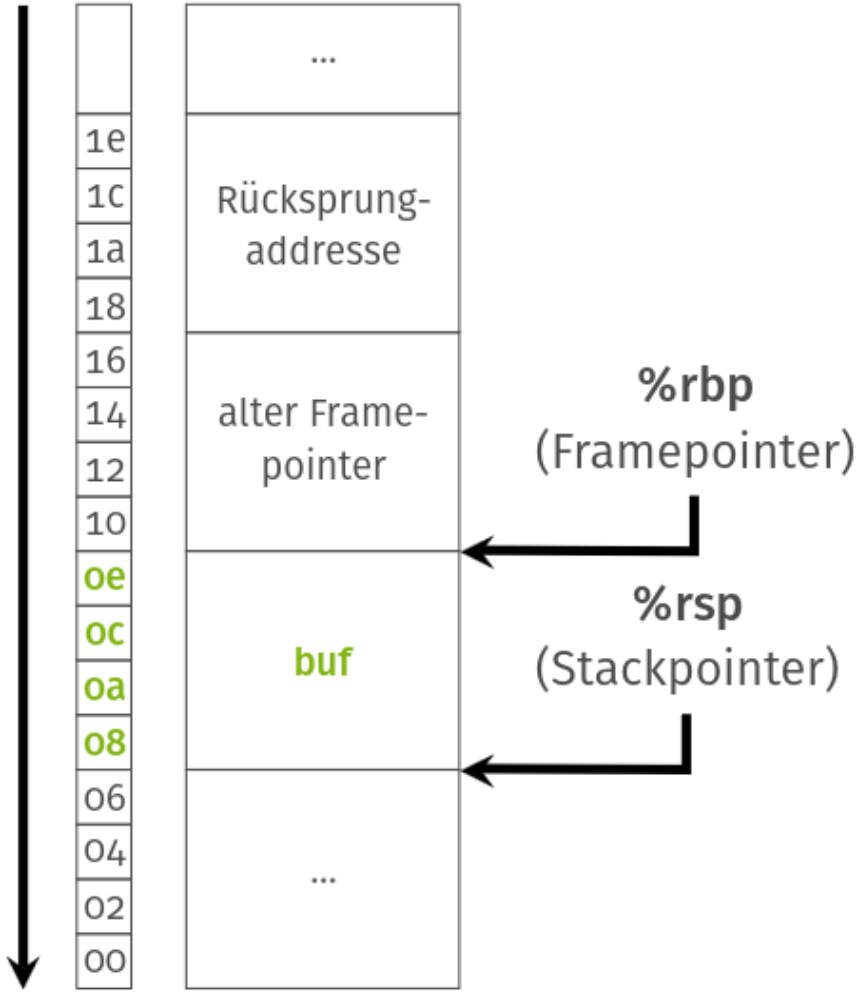
- Eingabe: “12345678abcdef12345678”



Was ist passiert?

```c
int ask_passwd(void) {
    char buf[8];
    printf("Passwort:");
    scanf("%s", buf);       // <--
    // ...
```

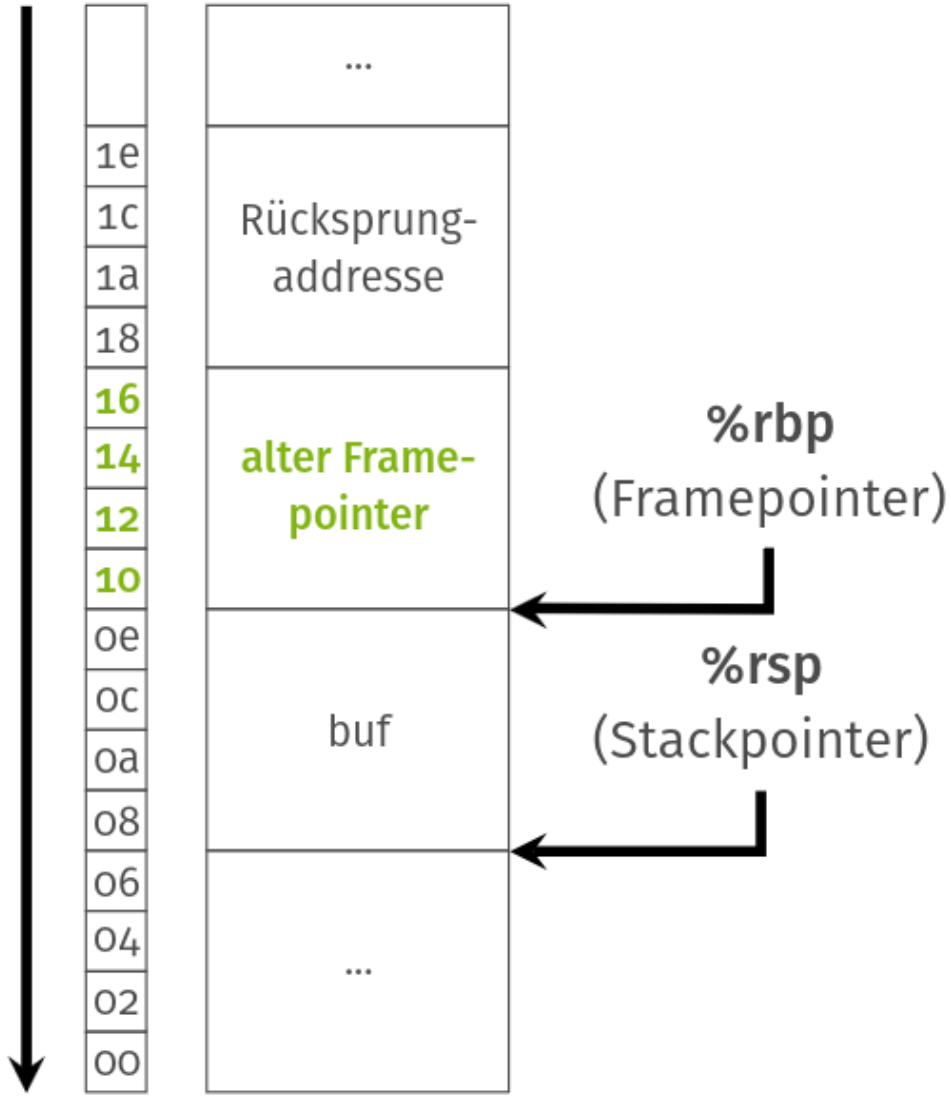
- Eingabe: “12345678abcdef12345678”



Was ist passiert?

```c
int ask_passwd(void) {
    char buf[8];
    printf("Passwort:");
    scanf("%s", buf);       // <--
    // ...
```

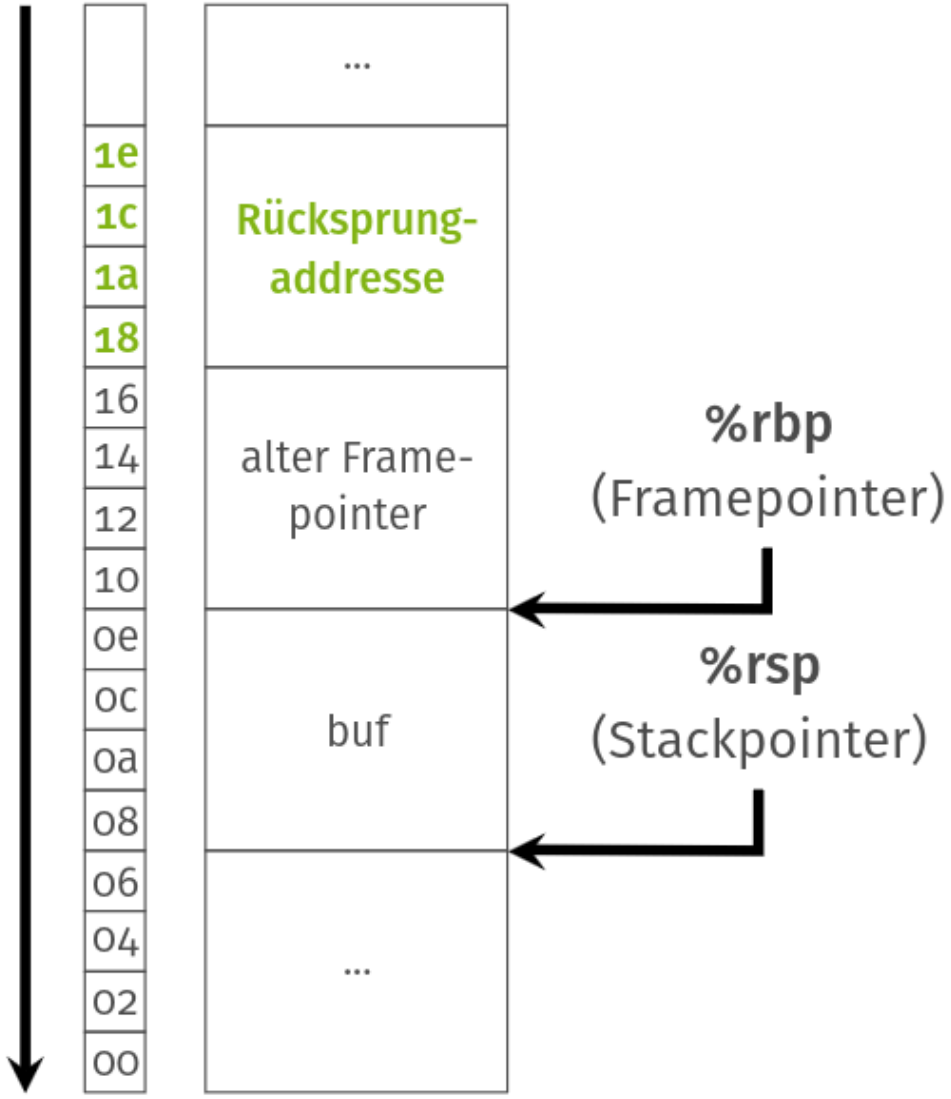
- Eingabe: “12345678**abcdef**12345678”
- Die Eingabe hat den Framepointer



Was ist passiert?

```c
int ask_passwd(void) {
    char buf[8];
    printf("Passwort:");
    scanf("%s", buf);       // <--
    // ...
```

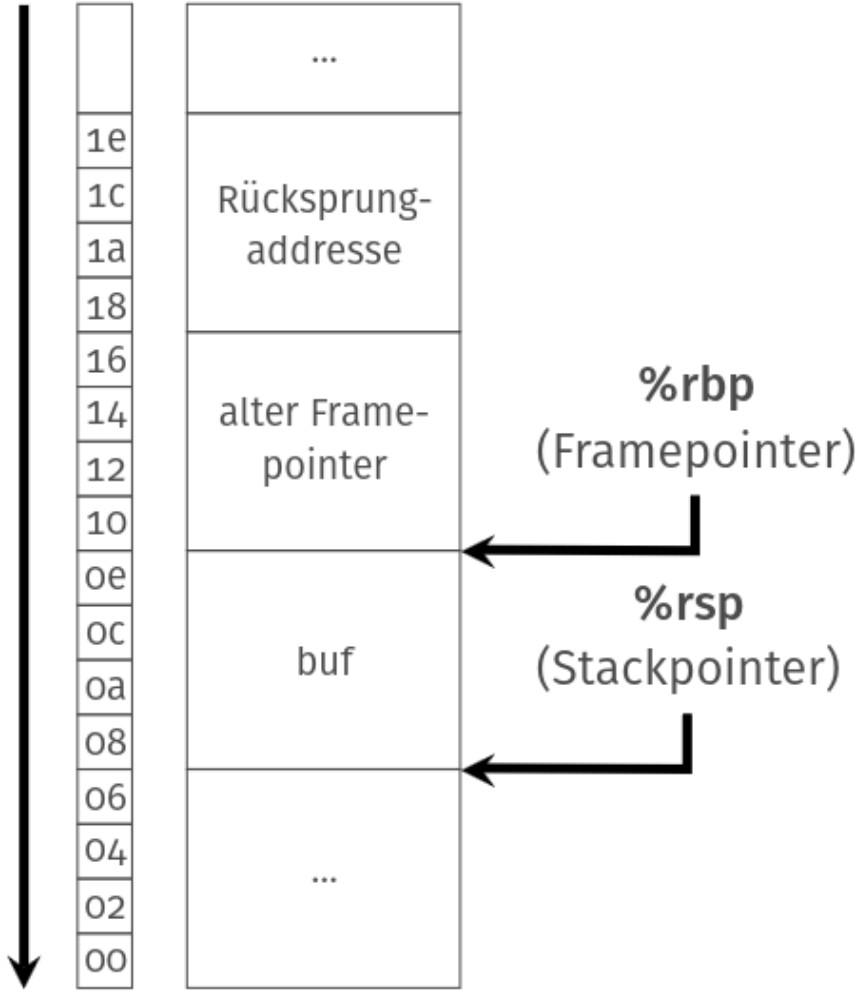
- Eingabe: “12345678**abcdef**12345678”
- Die Eingabe hat den Framepointer und die Rücksprungadresse überschrieben



Was ist passiert?

```c
int ask_passwd(void) {
    char buf[8];
    printf("Passwort:");
    scanf("%s", buf);       // <--
    // ...
```

- Eingabe: “12345678**abcdef**12345678”
- Die Eingabe hat den Framepointer und die Rücksprungadresse überschrieben
- Sprung zu einer nicht-ausführbaren Adresse
- `Segmentation Fault`
- Das ist noch der harmloseste Fall!



Pufferüberlauf hinter den Kulissen

- Prozess kann auf die Weise zum Sprung zu **beliebigen** Adressen gezwungen werden!

Pufferüberlauf hinter den Kulissen

- Prozess kann auf die Weise zum Sprung zu **beliebigen** Adressen gezwungen werden!
- Das kann auch ausgenutzt werden

```
studi@bsvm:~$ nm login | grep -F start_shell
00000000004008a1 T start_shell
```

Pufferüberlauf hinter den Kulissen

- Prozess kann auf die Weise zum Sprung zu **beliebigen** Adressen gezwungen werden!
- Das kann auch ausgenutzt werden

```
studi@bsvm:~$ nm login | grep -F start_shell
00000000004008a1 T start_shell
```

- Da Adressen aus nicht-darstellbaren Zeichen bestehen → Hilfsprogramm/-skript nötig
  - Umleiten der Standardeingabe
  - Ausgabe der Bytesequenz mit der Adresse am Ende (Byteorder!)
  - “Weiterleitung” der Standardeingabe

```
studi@bsvm:~$ ( printf "1234567812345678\xa1\x08\x40\x00\x00\x00\x00\x00\n" ; cat /dev/stdin ) | ./login
```

Codeinjektion

- Ein solcher Einsprungspunkt (`start_shell()`) ist nett, aber nicht unbedingt nötig
- Sogar Programme mit “ungefährlichen” Funktionen können dazu gebracht werden, Beliebiges zu tun!
- Anstatt nur einer Rücksprungadresse wird gleich Maschinencode injeziert → Rücksprungadresse zeigt auf den Stack selbst, wo der Code liegt

Weitere Techniken zur Codeinjektion

- **Return to libc:** Rücksprungadresse zeigt auf `libc`-Funktion, z.B. `system()`
  - `system(3)` führt **beliebige** Shell-Kommandos aus `system("ls -l");`
  - Stack wird wie bei einem normalen Aufruf von `system` präpariert
- **Heap-Überlauf:** Überschreiben von Variablen auf dem Heap möglich
  - Dadurch indirekt oft ebenfalls beliebiger Code ausführbar
- Empfohlene Artikel (natürlich nicht prüfungsrelevant):
  - Klassiker: “Smashing the Stack for Fun and Profit”
  - Modernere Techniken: “x86-64 buffer overflow exploits and borrowed code chunks exploitation technique”

Ist das praxisrelevant?

- Unix Morris Worm (sendmail) - Buffer Overflow / gets (siehe VI.)
- Blaster Work (aka. Lovsan, 2003)
- SQL Slammer (2003)
  - Stack Buffer Overflow in Microsoft SQL Server (über UDP-Port 1434)
  - Rapide Ausbreitung (bis zu 75.000 Hosts in den ersten 10 Minuten)
  - Starke Beeinträchtigung des Internet-Datenverkehrs
- Sasser (2004)
  - "Stack Buffer Overflow in LSASS" (Local Security Authority Subsystem Service) (TCP-Port 445), unterschiedl. Varianten
  - Author: 17j. deutscher Informatikstudent
- Conficker (2008 bis heute)
  - "RPC Buffer Overflow, Shellcode"

- Ähnlich starke Verbreitung wie SQL Slammer (3-15 Mio. PCs), u.a. Teilausfall diverser Streitkräfte (Frz./Brit. Marine, Bundeswehr)

Schutzmaßnahmen (1)

- Hardware
  - NX-Bit / XD-Bit (SPARC, IA32 seit 2005, IA64-Prozessoren)
  - Stack-/Heap- und Daten-Speicherseiten Not eXecutable
- Betriebssystem
  - **stack/address space randomization** (ASLR)
  - Benötigt relozierbare Programme (`gcc -pie`)
- Compiler
  - z.B. **GCC Stack Smashing Protector**
  - Standardmäßig im Einsatz bei z.B. OpenBSD, FreeBSD, Linux
  - Teils Standard, ansonsten via Parameter `-fstack-protector`
  - Funktionsweise: Bekannte Zahlen, sogenannte *Canaries*, vor und hinter Puffer schreiben und überwachen

Schutzmaßnahmen (1)

- Hardware
  - NX-Bit / XD-Bit (SPARC, IA32 seit 2005, IA64-Prozessoren)
  - Stack-/Heap- und Daten-Speicherseiten Not eXecutable
- Betriebssystem
  - **stack/address space randomization** (ASLR)
  - Benötigt relozierbare Programme (`gcc -pie`)
- Compiler
  - z.B. **GCC Stack Smashing Protector**
  - Standardmäßig im Einsatz bei z.B. OpenBSD, FreeBSD, Linux
  - Teils Standard, ansonsten via Parameter `-fstack-protector`
  - Funktionsweise: Bekannte Zahlen, sogenannte *Canaries*, vor und hinter Puffer schreiben und überwachen

- **“Canaries” sind eine Anspielung auf die Kanarienvögel in Kohleminen, die bei giftigen Gasen als erste umkippten.**

Schutzmaßnahmen (2)

- Programmierung
  - `strcpy(),strcat()` → `strncpy(),strncat()`
  - `sprintf()` → `snprintf()`
  - `gets()` → `fgets()`
  - scanf() → Feldbreite beschränken: `scanf("%10s", buf)`

Klausuraufgabe: I/O-Scheduling (1)

Gegeben sei ein Plattenspeicher mit 16 Spuren. Der jeweilige I/O-Scheduler bekommt immer wieder Leseaufträge für eine bestimmte Spur. Die Leseaufträge in L_1 sind dem I/O-Scheduler bereits bekannt. **Nach zwei** bearbeiteten Aufträgen erhält er die Aufträge in L_2. **Nach weiteren vier** (d.h. nach insgesamt sechs) bearbeiteten Aufträgen erhält er die Aufträge in L_3. Zu Beginn befindet sich der Schreib-/Lesekopf über Spur 0.

Eingabe

$$L_1 = \{5, 15, 2, 9\} \quad L_2 = \{4, 10, 1\} \quad L_3 = \{8, 6, 14\}$$

Klausuraufgabe: I/O-Scheduling (1)

Gegeben sei ein Plattenspeicher mit 16 Spuren. Der jeweilige I/O-Scheduler bekommt immer wieder Leseaufträge für eine bestimmte Spur. Die Leseaufträge in L_1 sind dem I/O-Scheduler bereits bekannt. **Nach zwei** bearbeiteten Aufträgen erhält er die Aufträge in L_2. **Nach weiteren vier** (d.h. nach insgesamt sechs) bearbeiteten Aufträgen erhält er die Aufträge in L_3. Zu Beginn befindet sich der Schreib-/Lesekopf über Spur 0.

Eingabe

$$L_1 = \underbrace{\{5, 15, 2, 9\}}_{\text{Sofort bekannt}} \quad L_2 = \underbrace{\{4, 10, 1\}}_{\text{Nach 2 Ops bekannt}} \quad L_3 = \underbrace{\{8, 6, 14\}}_{\text{Nach 6 Ops bekannt}}$$

Klausuraufgabe: I/O-Scheduling (1)

Eingabe

$$L_1 = \underbrace{\{5, 15, 2, 9\}}_{\text{Sofort bekannt}} \quad L_2 = \underbrace{\{4, 10, 1\}}_{\text{Nach 2 Ops bekannt}} \quad L_3 = \underbrace{\{8, 6, 14\}}_{\text{Nach 6 Ops bekannt}}$$

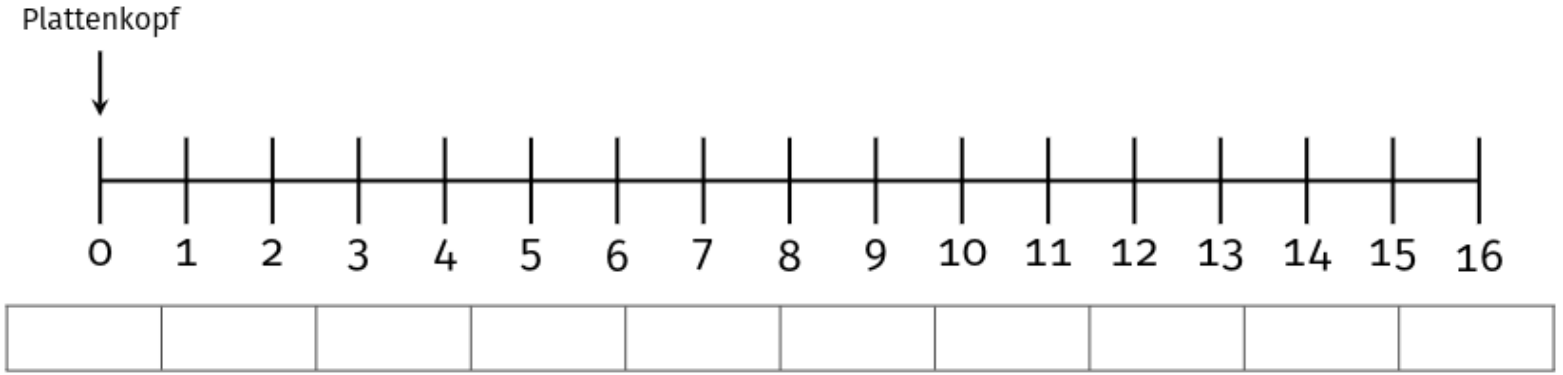
Bitte tragen Sie hier die Reihenfolge der gelesenen Spuren für einen I/O-Scheduler, der nach der ***Shortest Seek Time First*** (**SSTF**)-Strategie arbeitet, ein:

Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 0$

$L = \{5, 15, 2, 9\}$

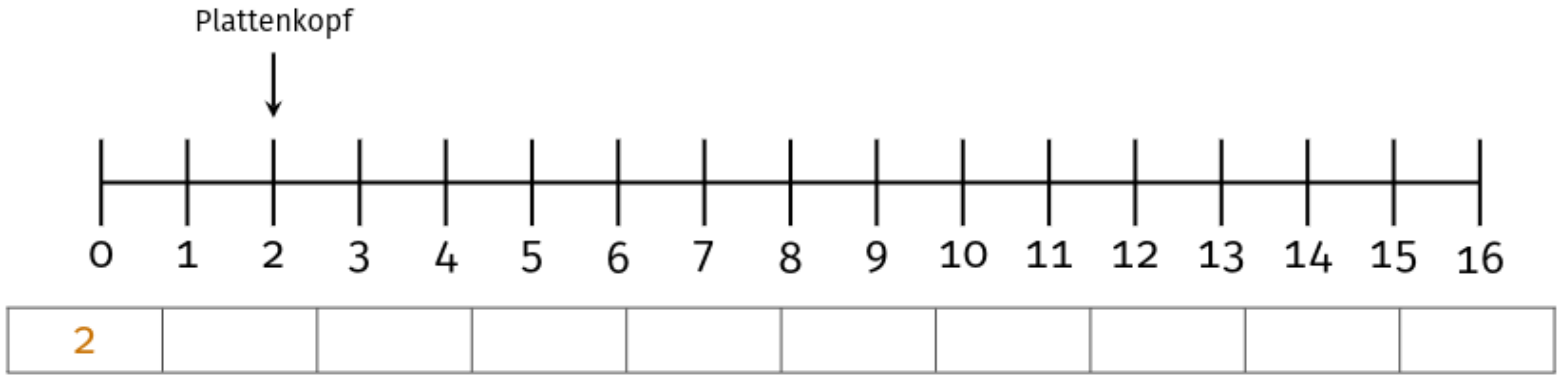


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 1$

$L = \{5, 15, 9\}$

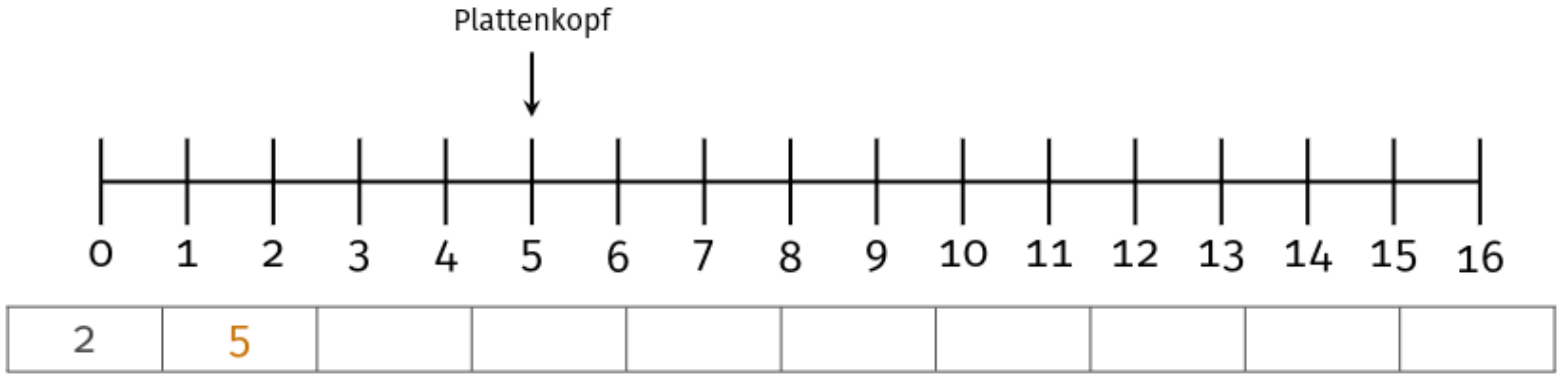


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 2$

$L = \{15, 9\}$

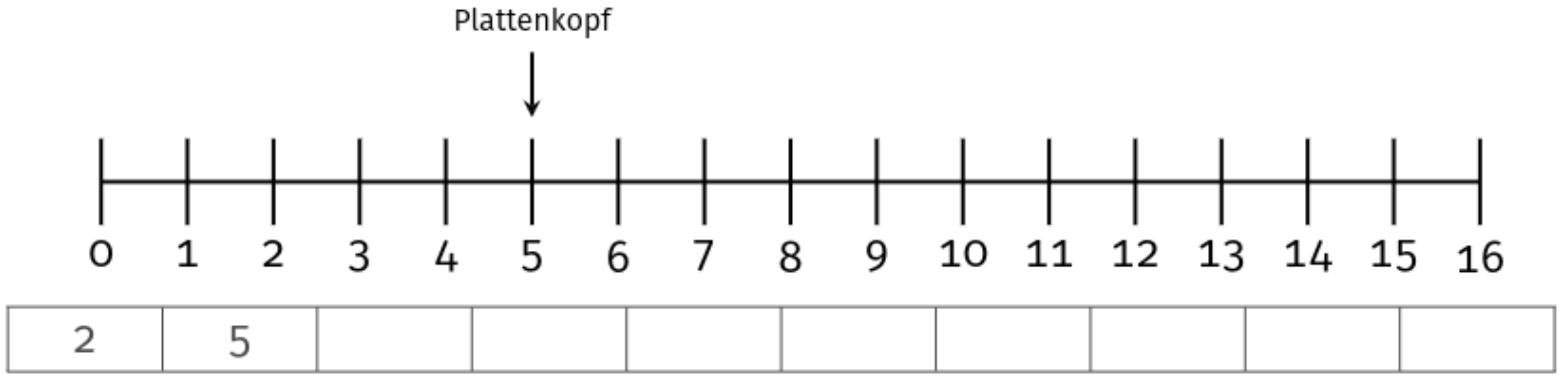


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 2$

$L = \{15, 9, 4, 10, 1\}$

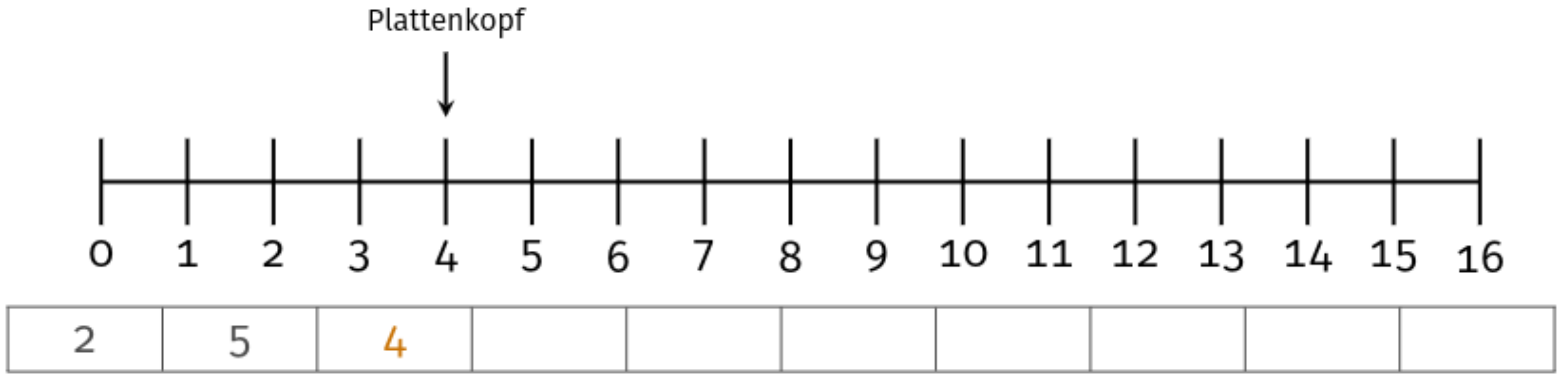


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 3$

$L = \{15, 9, 10, 1\}$

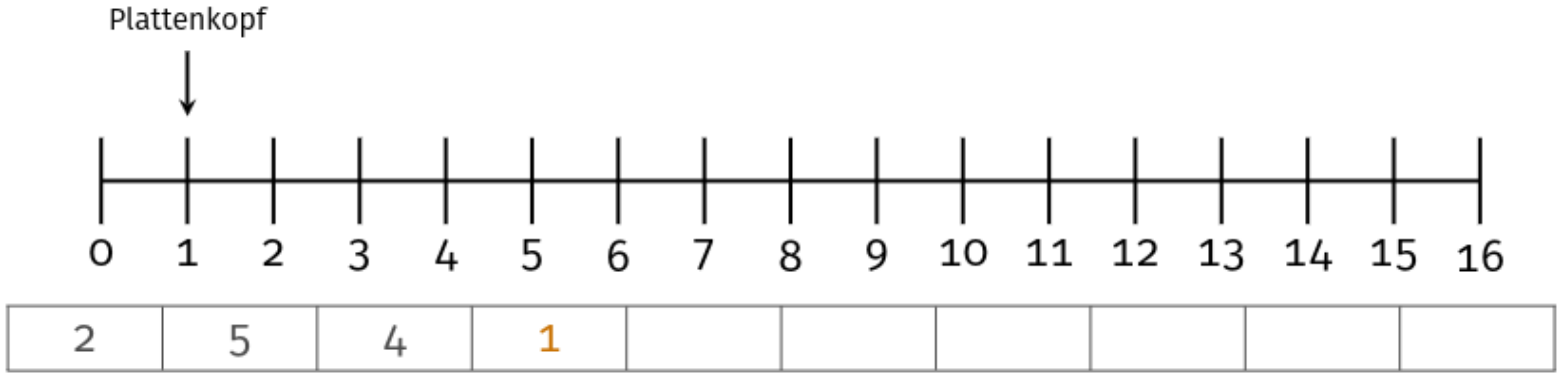


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 4$

$L = \{15, 9, 10\}$

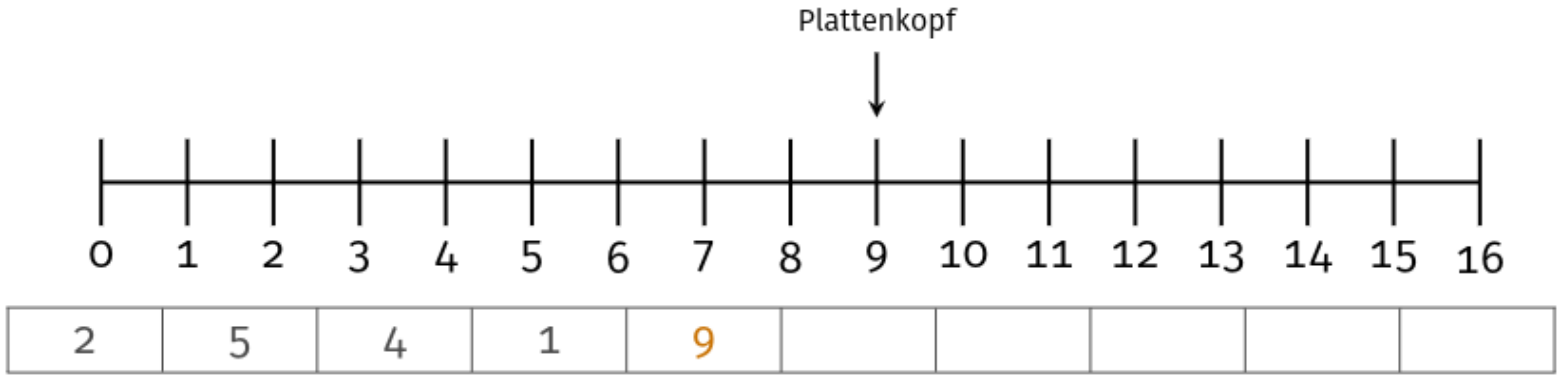


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 5$

$L = \{15, 10\}$

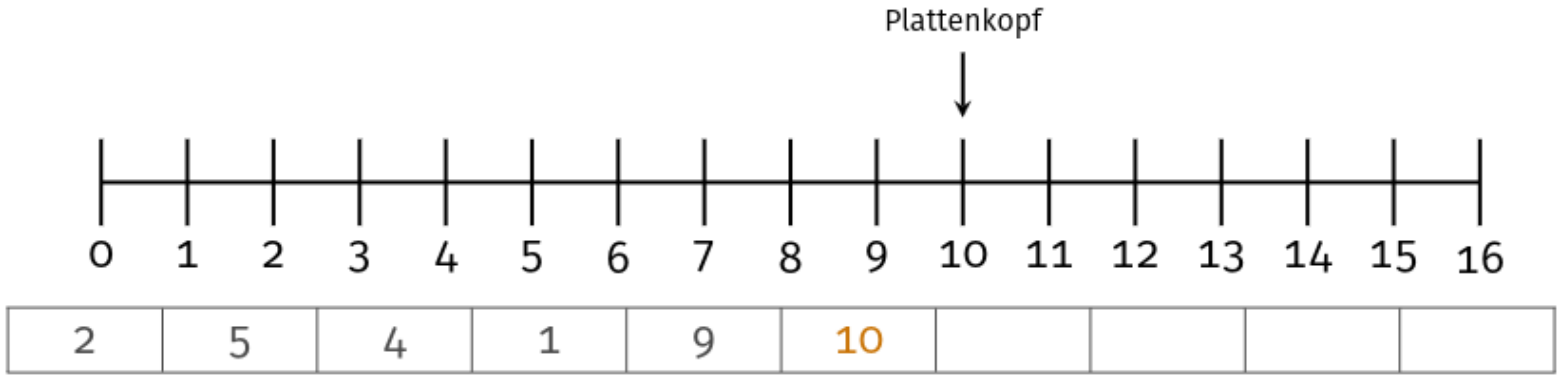


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 6$

$L = \{15\}$

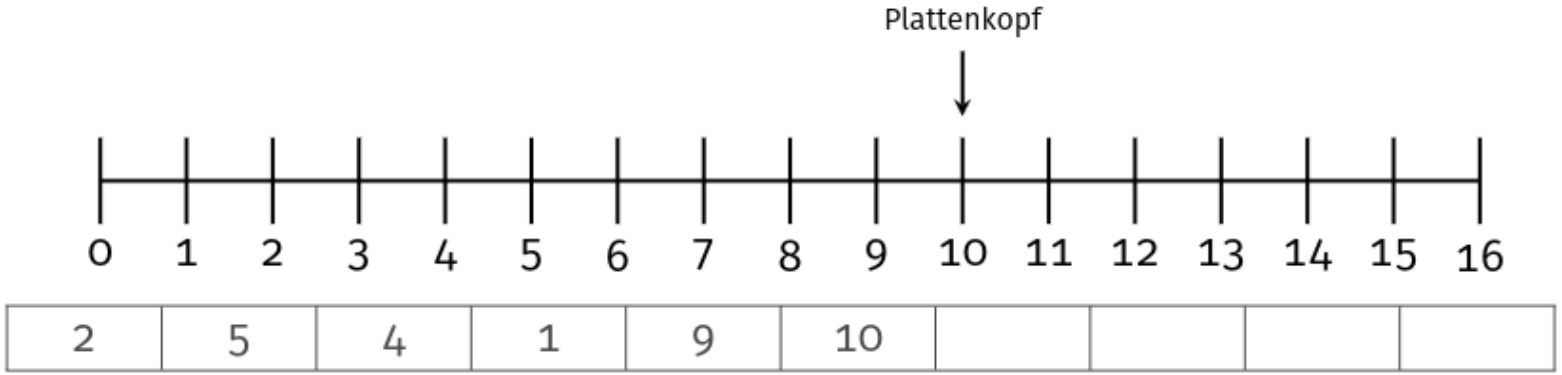


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 6$

$L = \{15, 8, 6, 14\}$

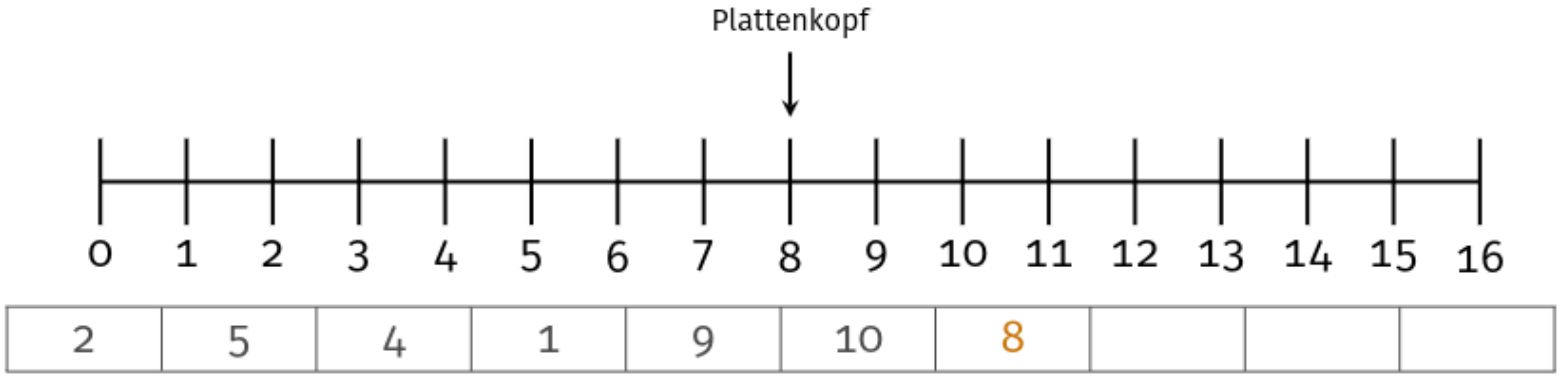


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 7$

$L = \{15, 6, 14\}$

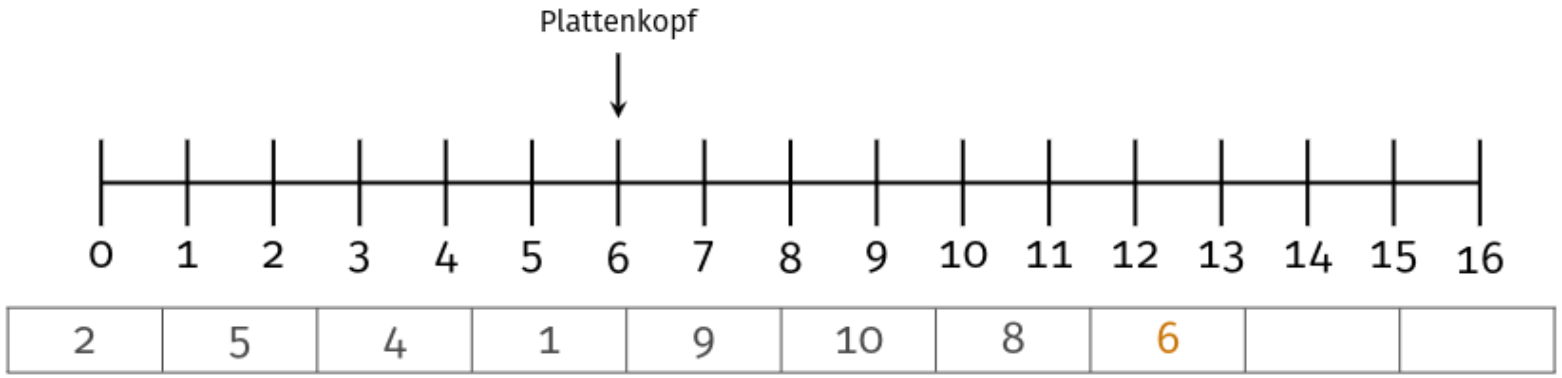


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 8$

$L = \{15, 14\}$

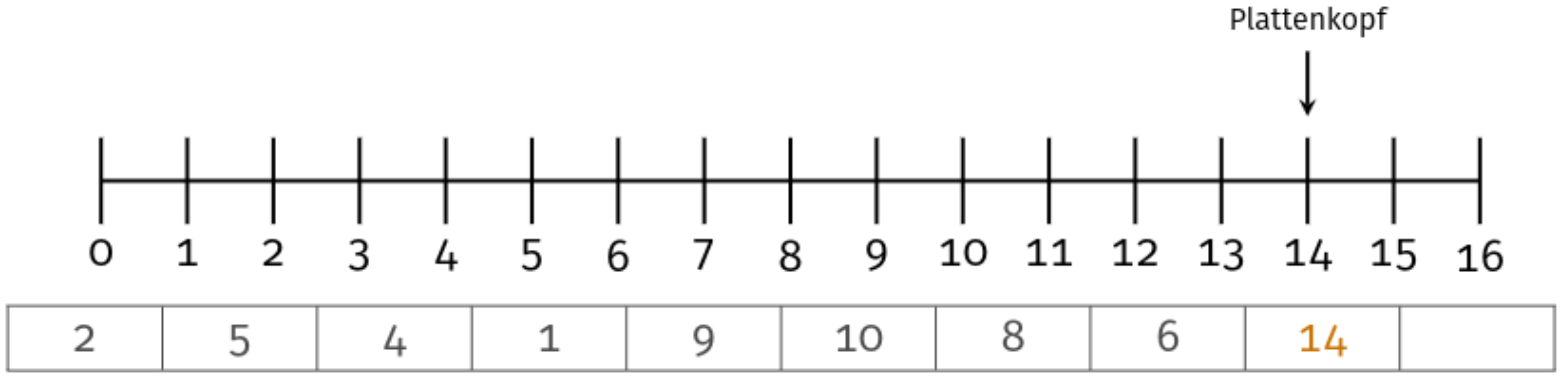


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 9$

$L = \{15\}$

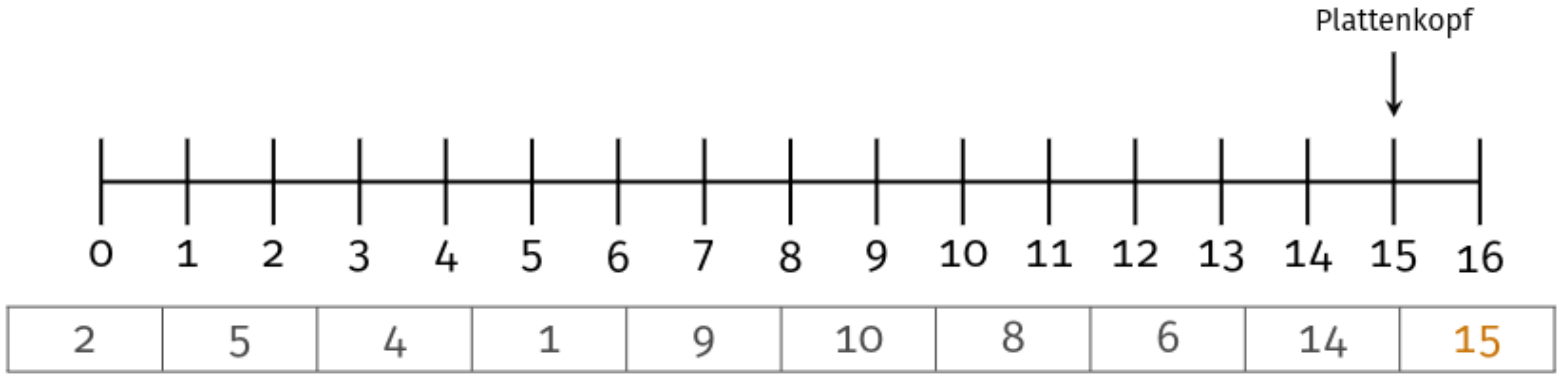


Klausuraufgabe: I/O-Scheduling (1)

I/O-Anfragen

$T = 10$

$L = \{\}$



Klausuraufgabe: I/O-Scheduling (2)

Eingabe

$$L_1 = \underbrace{\{5, 15, 2, 9\}}_{\text{Sofort bekannt}} \quad L_2 = \underbrace{\{4, 10, 1\}}_{\text{Nach 2 Ops bekannt}} \quad L_3 = \underbrace{\{8, 6, 14\}}_{\text{Nach 6 Ops bekannt}}$$

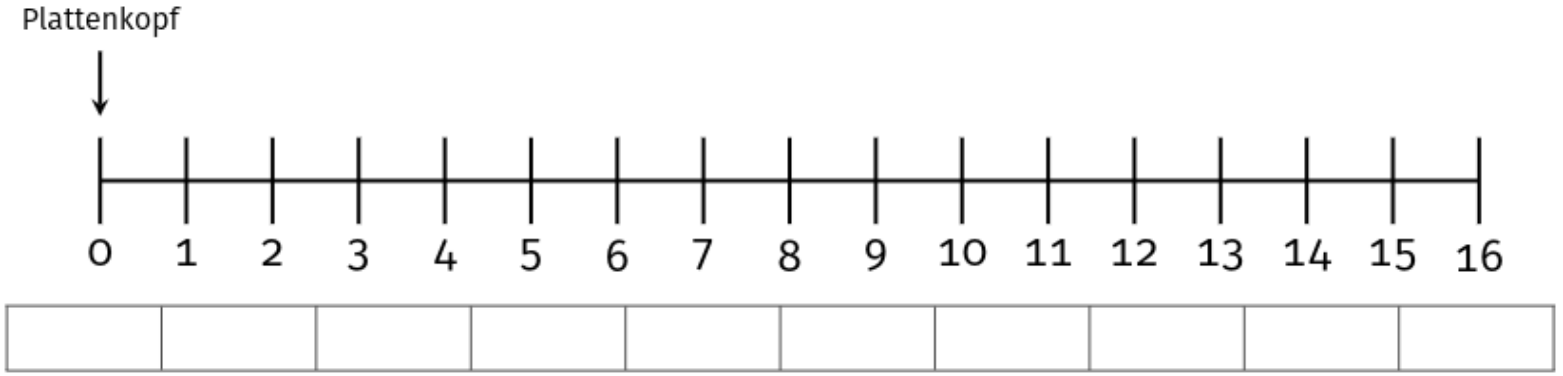
Bitte tragen Sie hier die Reihenfolge der gelesenen Spuren für einen I/O-Scheduler, der nach der **Fahrstuhl** (***Elevator***-)-Strategie arbeitet, ein:

Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 0$

$L = \{5, 15, 2, 9\}$

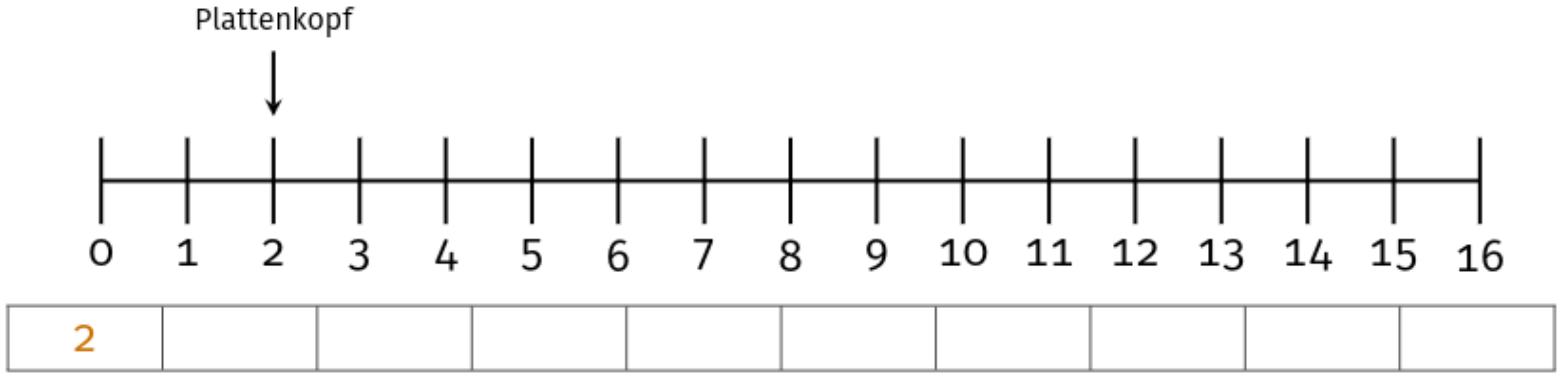


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 1$

$L = \{5, 15, 9\}$

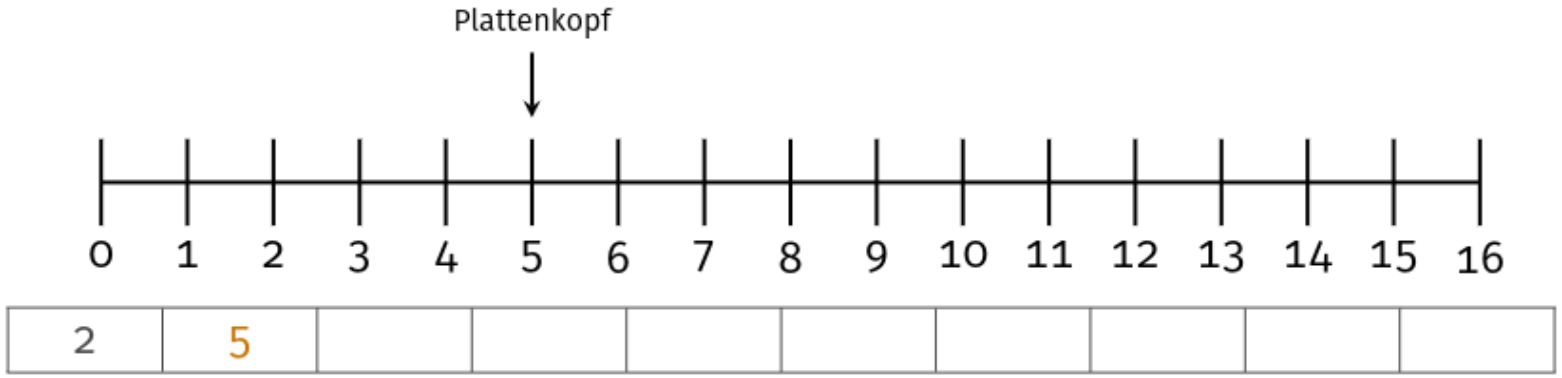


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 2$

$L = \{15, 9\}$

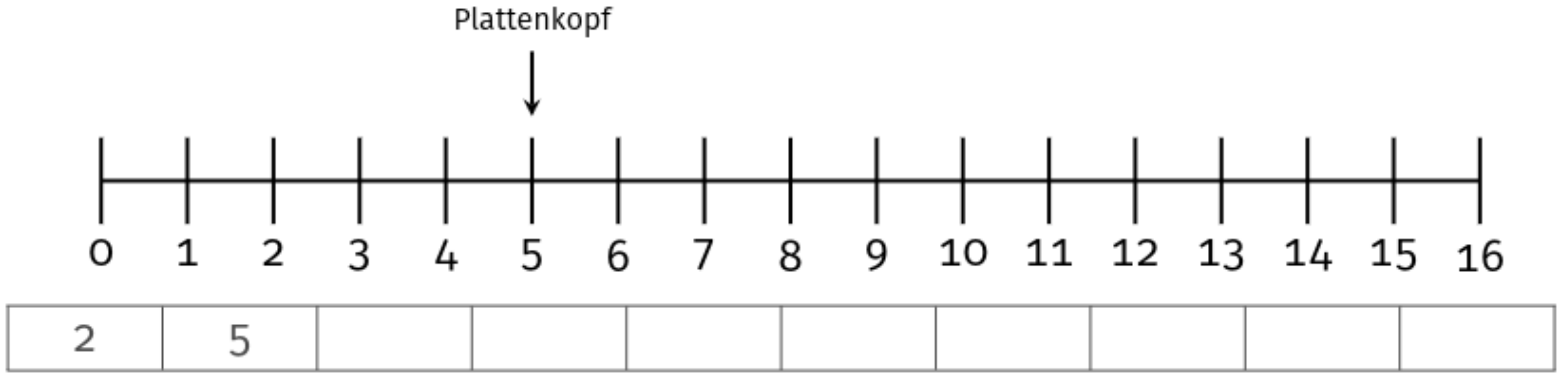


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 2$

$L = \{15, 9, 4, 10, 1\}$

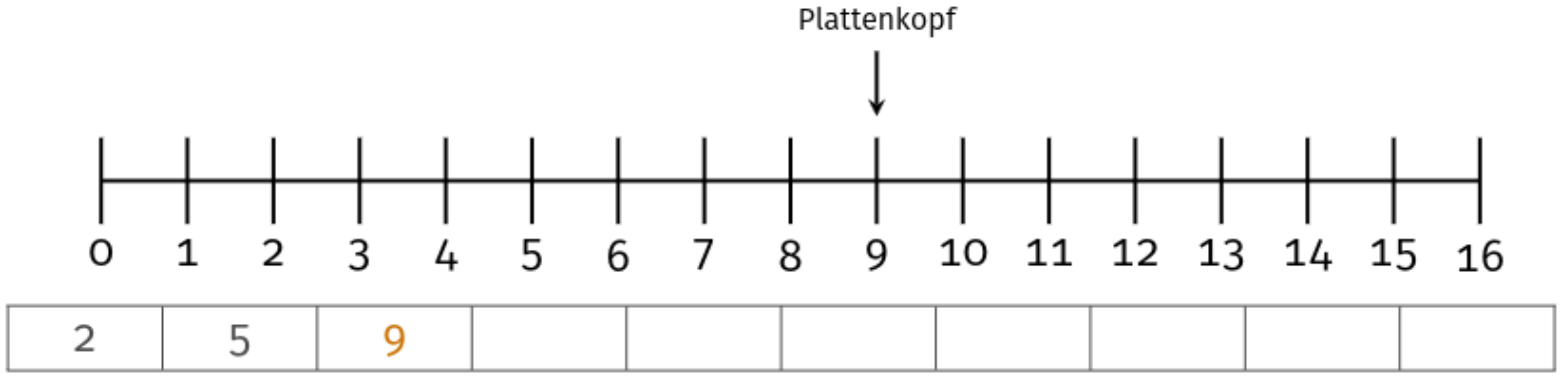


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 3$

$L = \{15, 4, 10, 1\}$

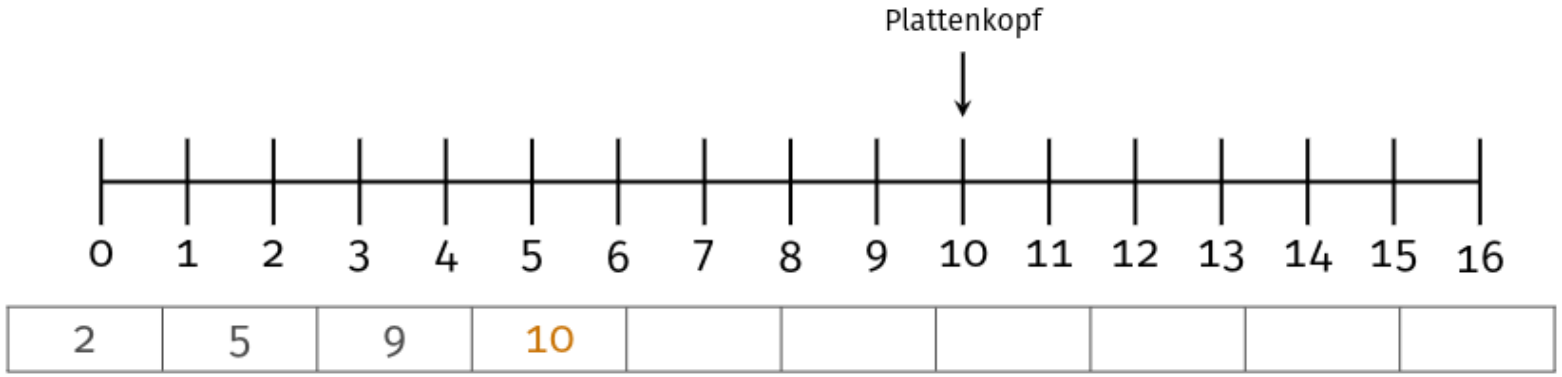


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 4$

$L = \{15, 4, 1\}$

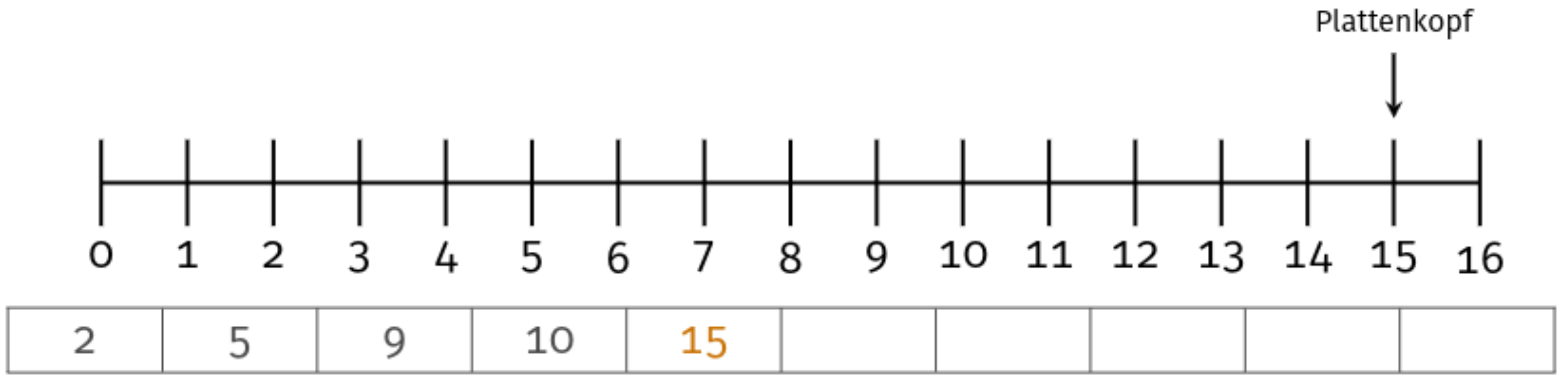


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 5$

$L = \{4, 1\}$

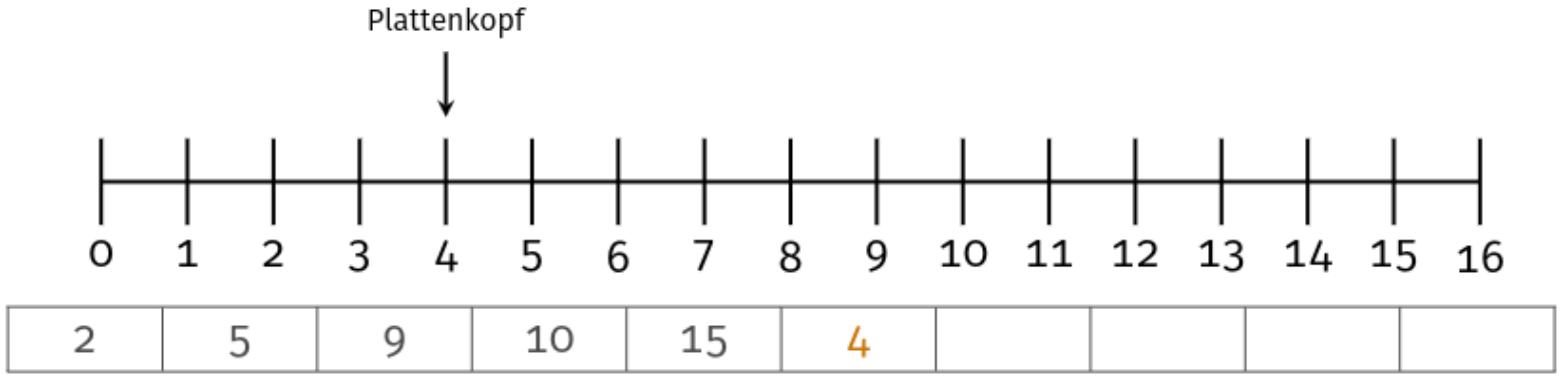


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 6$

$L = \{1\}$

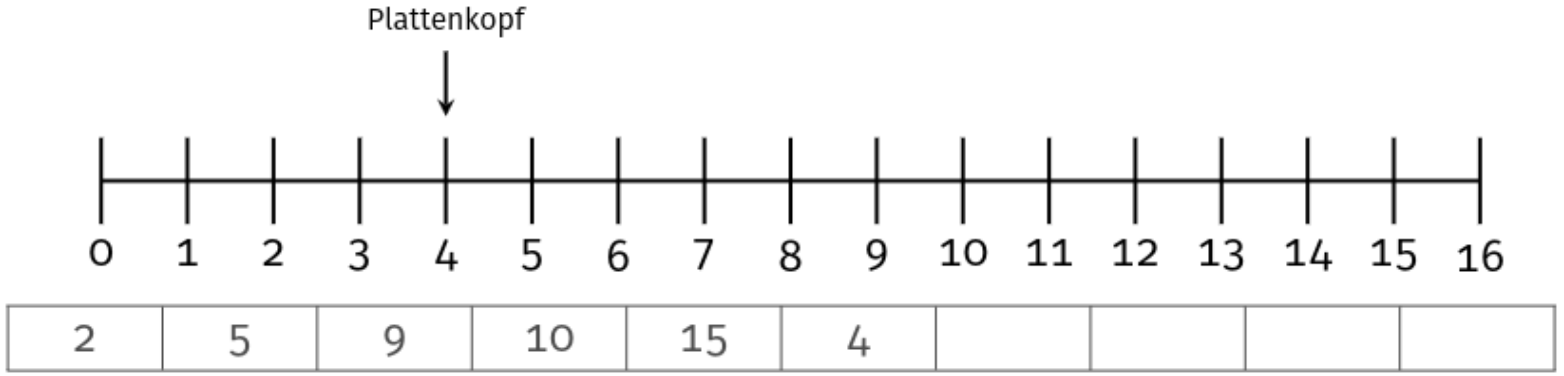


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 6$

$L = \{1, 8, 6, 14\}$

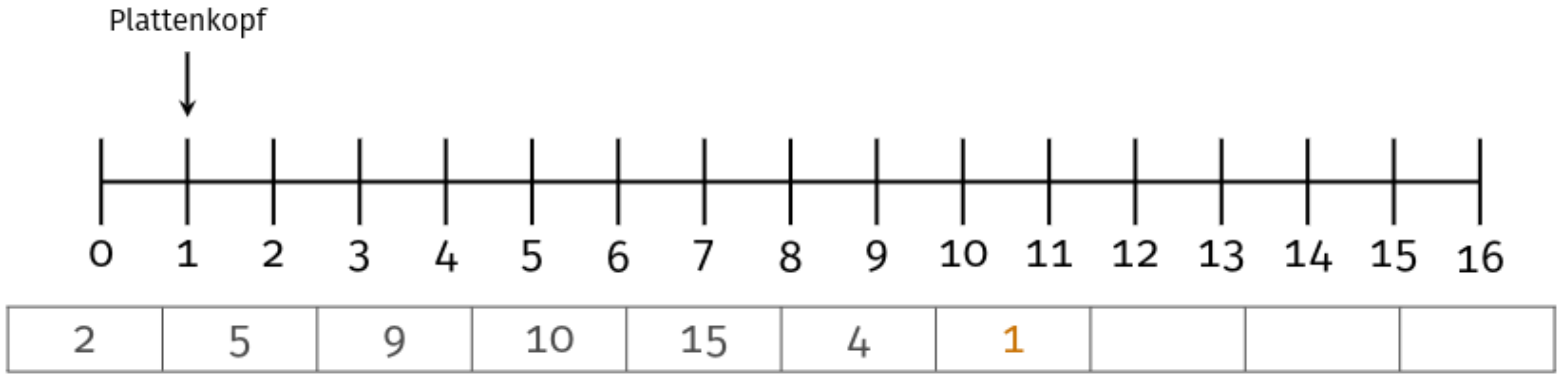


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 7$

$L = \{8, 6, 14\}$

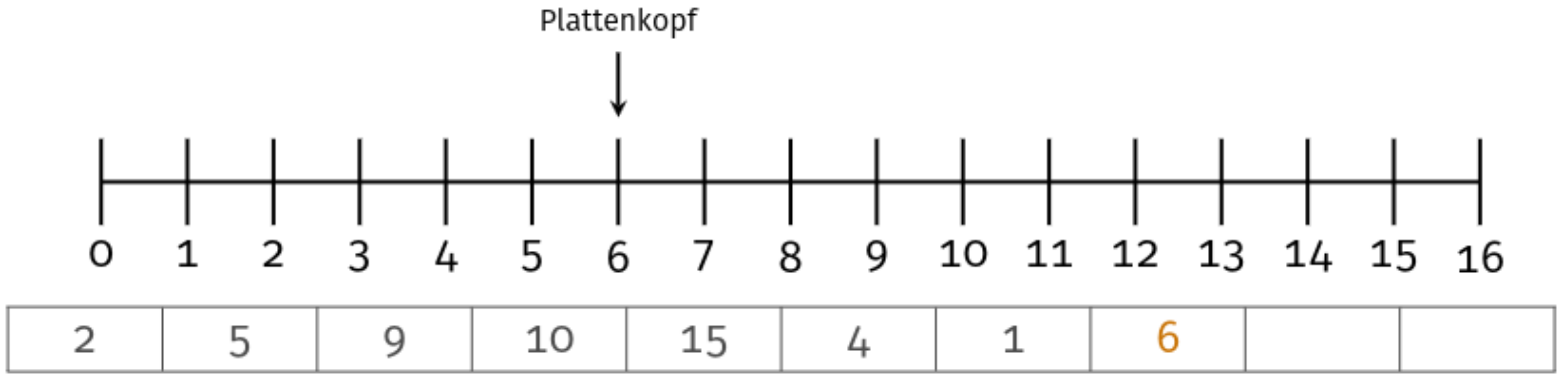


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 8$

$L = \{8, 14\}$

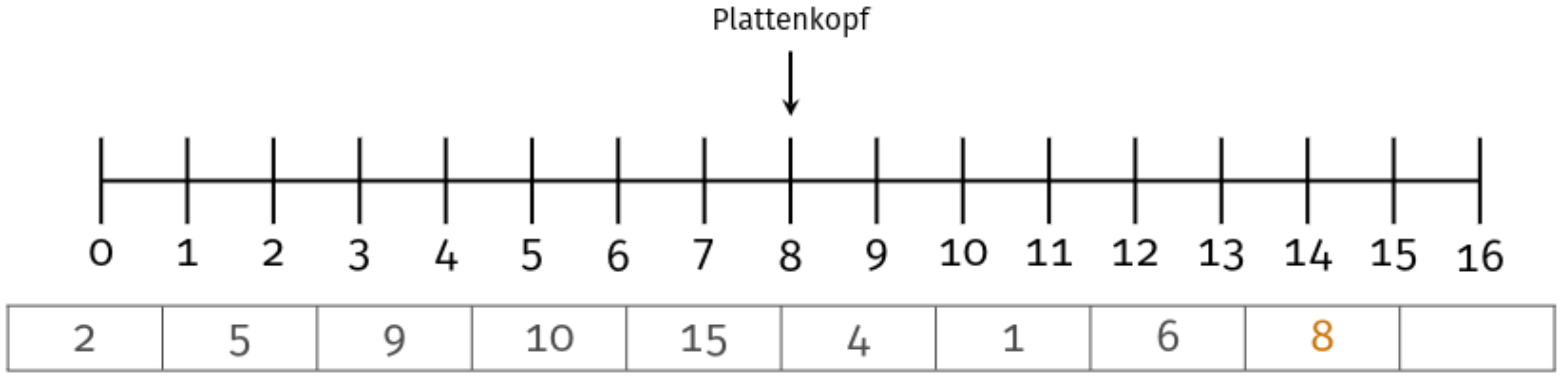


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 9$

$L = \{14\}$

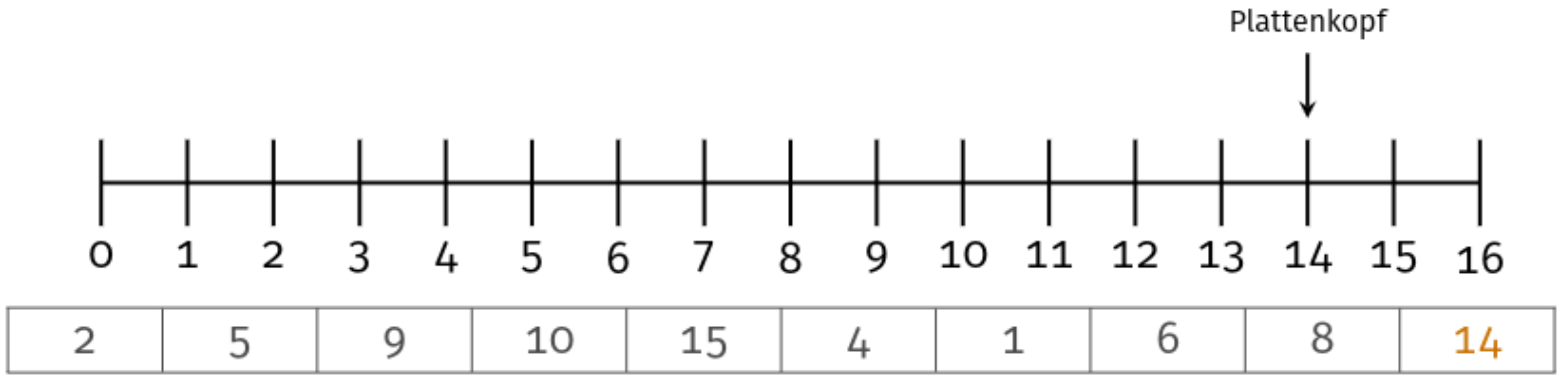


Klausuraufgabe: I/O-Scheduling (2)

I/O-Anfragen

$T = 10$

$L = \{\}$



Was haben wir heute gelernt?

Zusammenfassung

- Sicherheitsrisiken in C
- Gegenmaßnahmen

Informationen

Bei Fragen und Problemen

- Helpdesk finden montags 14-16 Uhr sowie mittwochs 10-12 Uhr
- Via Matrix #bs-helpdesk:fachschaften.org
- Via E-Mail bs-problems@ls12.cs.tu-dortmund.de