ELEC 377 – Operating Systems

Week 11 – Class 1

Last Class

- Security
  - ◊ Network Security
    - ◊ System Threats
    - ◊ Passwords and Authentication

Today

- Security
  - ◊ Authentication/Accuracy
  - ◊ System Threats

Network Security

- pharming
  - ◊ reverse proxy for a online bank/Paypal
  - ◊ compromise a DNS server/Or DHCP server
    - new attack, DNS poisoning
  - ◊ point bank/PayPal at your reverse proxy
  - ◊ pass transactions through to the bank
    - but record information for later use.
    - security images???
  - ◊ compromise router
    - backbone routers
    - consumer grade routers
  - DLINK advertising...

Authentication

- Passwords
  - ◊ main login
  - ◊ access to resources (databases, Unix groups)
- Vulnerable
  - ◊ guessing - most user chosen passwords are easy to remember, short, easy to guess
    -WPA interface
  - ◊ shoulder surfing (ATM hack)
  - ◊ packet sniffing (conferences)
  - ◊ masquerade
  - ◊ account sharing
- System generated?
  - ◊ too hard to remember?

Passwords

- Must store to verify?
  - ◊ If passwords are stored on OS must be secure
  - ◊ encrypted passwords (websites??/London)
  - ◊ one way encryption
    - how to check?
    - safe???
  - ◊ brute force attack (Dictionary Attack)
  - ◊ public file?
    /etc/secure

Passwords

- One Time Passwords
  - ◊ challenge response
    - – hardware key
  - ◊ one time pad
    - – list of random numbers
    - – early on-line banking

- Biometrics

◊ Fingerprints, retina, iris

◊ replay attacks?

◊ major disadvantage

Passwords

- Biometrics
  - ◊ Fingerprints, retina, iris
  - ◊ accuracy
    - false positives (identifies me as you)
    - false negatives (denies you)
  - ◊ anonymity (my yahoo account is anoymous)
  - ◊ multiple accounts
    high security/low security
    - limited number of biometric keys

Passwords

- Biometrics
  - ◊ false sense of security
    - thermal sensors
    - repudiation
  - ◊ replay attacks?
  - ◊ fake fingers
    - silicone fingers

  Tsutomu Matsumoto of Yokohama National University

  - Gelatin fingers (same electrical characteristics as flesh)
  - can be made from finger prints left on any object

Other Biometric based Auth

- ◊ Oder Based Authentication
- ◊ Blood Vessel Patterns in Fingers

About Accuracy

- accuracy - what does it mean?
- 300 Million People in the USA
- Assume 1000 terrorists (1 per 300,000 = .00033%)
- Assume 40 percent positive detection (finds 40%) (400 terrorists)
- Assume 0.01% misidentification (30,000 people)

So What is the chance that someone identified as a terrorist is a terrorist?

About Accuracy

- accuracy - what does it mean?
- 300 Million People in the USA
- Assume 1000 terrorists (1 per 300,000 = .00033%)
- Assume 40 percent positive detection (finds 40%) (400 terrorists)
- Assume 0.01% misidentification (30,000 people)

So What is the chance that someone identified as a terrorist is a terrorist?

400/30,000 = 1.32 %

About Accuracy

- 300 Million People in the USA
- Assume 1000 terrorists (1 per 300,000 = .00033%)
- Assume 70% positive detection (700 terrorists)
- Assume 0.01% misidentification (30,000 people)

So What is the chance that someone identified as a terrorist is a terrorist?

About Accuracy

- 300 Million People in the USA
- Assume 1000 terrorists (1 per 300,000 = .00033%)
- Assume 70% positive detection (700 terrorists)
- Assume 0.01% misidentification (30,000 people)

So What is the chance that somone identified as a terrorist is a terrorist?

700/30,000 = 2.3%

Program Threats

- Trojan Horse
  - ◊ game program that sends the contents your mail box to another server
  - ◊ utility that wipes out your accounting program (DOS)

- Masquerade
  - ◊ special type of trojan horse
  - ◊ pretends to be a valid service
  - ◊ login masquerade
  - ◊ web site masquerade (spelling error/email)

Program Threats

- Trap Door/Back Door
  - ◊ Intentional hole left by programmer
  - ◊ Hard coded account numbers or Ids
  - ◊ War Games (Matthew Broderick)

Buffer Overflow (Globals)

- Variants

◊ function pointers in the heap within range of a global buffer (simple overwrite)

```
char buffer[1024];
struct proc_dir{
   int (*read_proc)(char *page, char**start...)
} theProcDir;
```

◊ theProcDir is after buffer in memory, overwrite read_proc variable, next time called, calls our code

Buffer Overflow (Globals)

- Variants

◊ vtable pointers (C++)

```
class A {
  virtual int foo(){....};
  int bar(){.....};
 }
```

```
class B: public A {
virtual int foo(){....};
   int bar(){.....};
}
```

- call to bar is known at compile time (called directly)
- foo is based on type of instance in variable
- called through a global table of functions

Buffer Overflow in the Heap

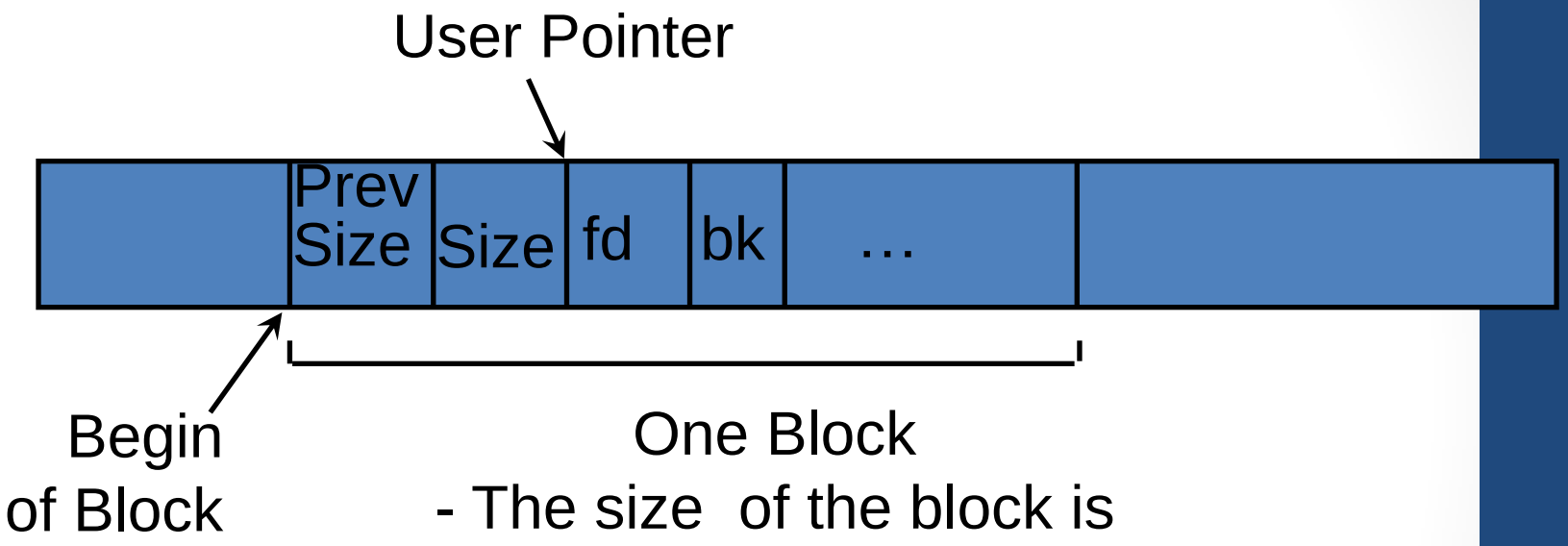
- What if the buffer is in the heap (after pointers)?
  - unused memory is kept in bins based on size of block
  - each bin is represented by a double linked list

```
#define INTERNAL_SIZE_T size_t

struct malloc_chunk {
    INTERNAL_SIZE_T prev_size;
    INTERNAL_SIZE_T size;
    struct malloc_chunk * fd;
    struct malloc_chunk * bk;
};
```

This section based on “Smashing the Heap for Fun and Profit”, Michel "MaXX" Kaempf, http://doc.bughunter.net/buffer-overflow/heap-corruption.html

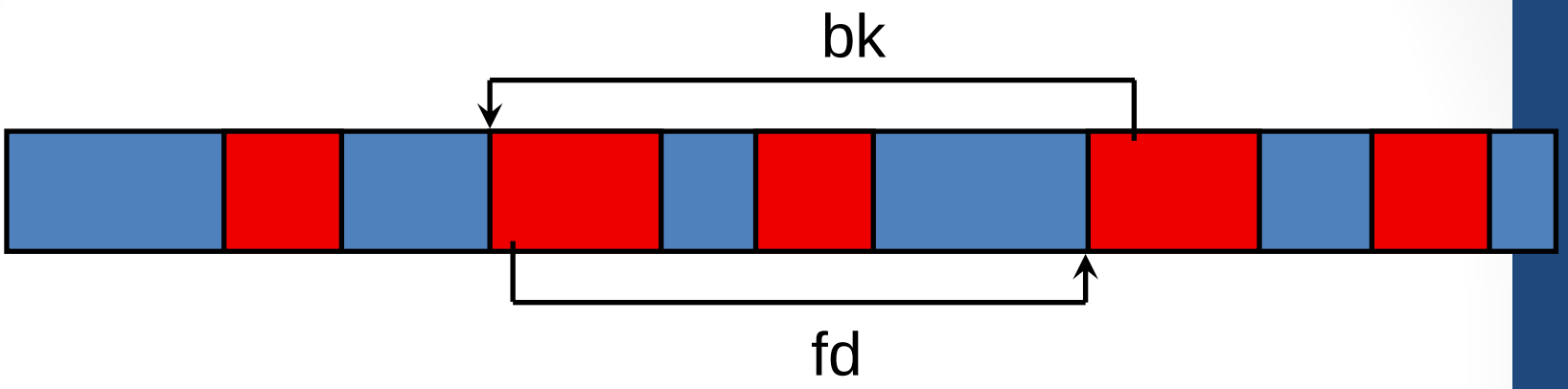
Heap Data Structure



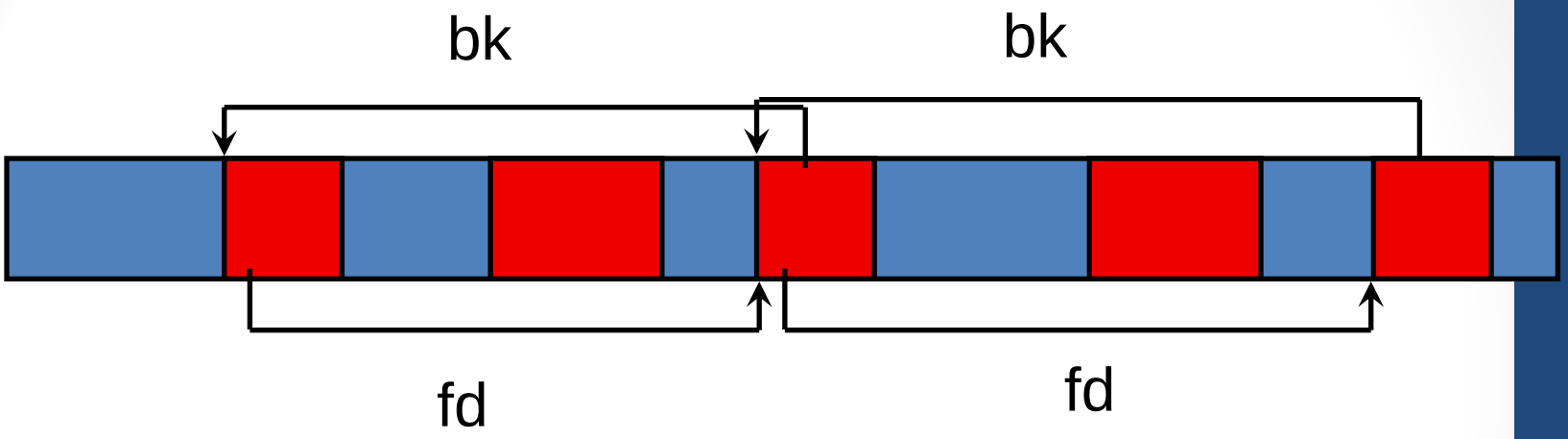
fd (forward) and bk (backward) are only used when the block is unallocated

Prev Size and Size are always used

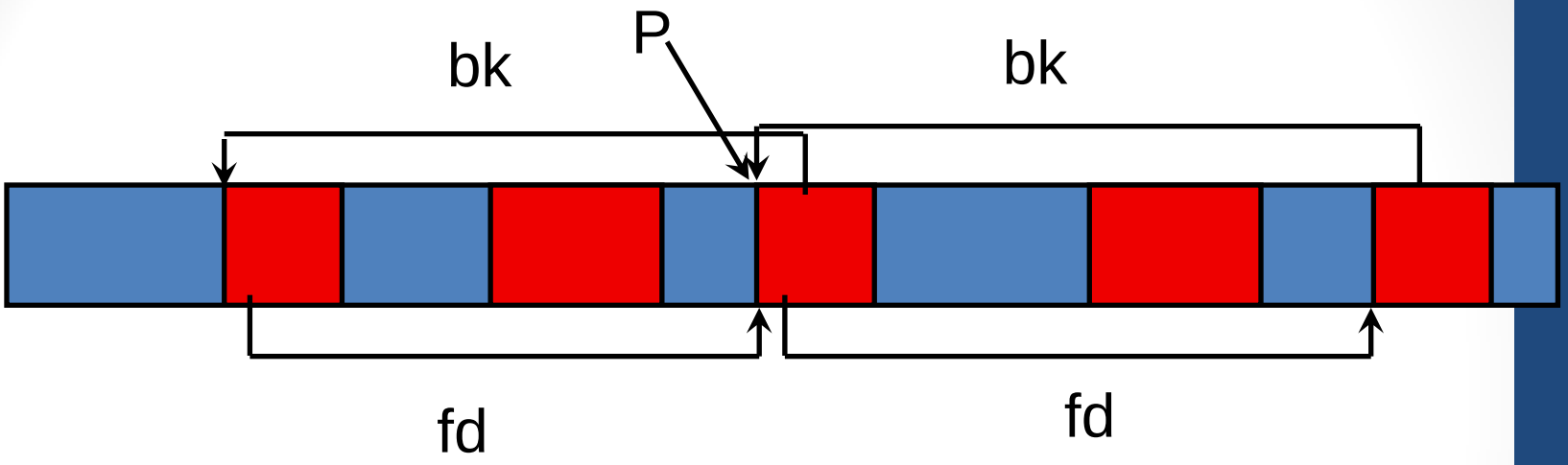
Linking Blocks



Linking Blocks

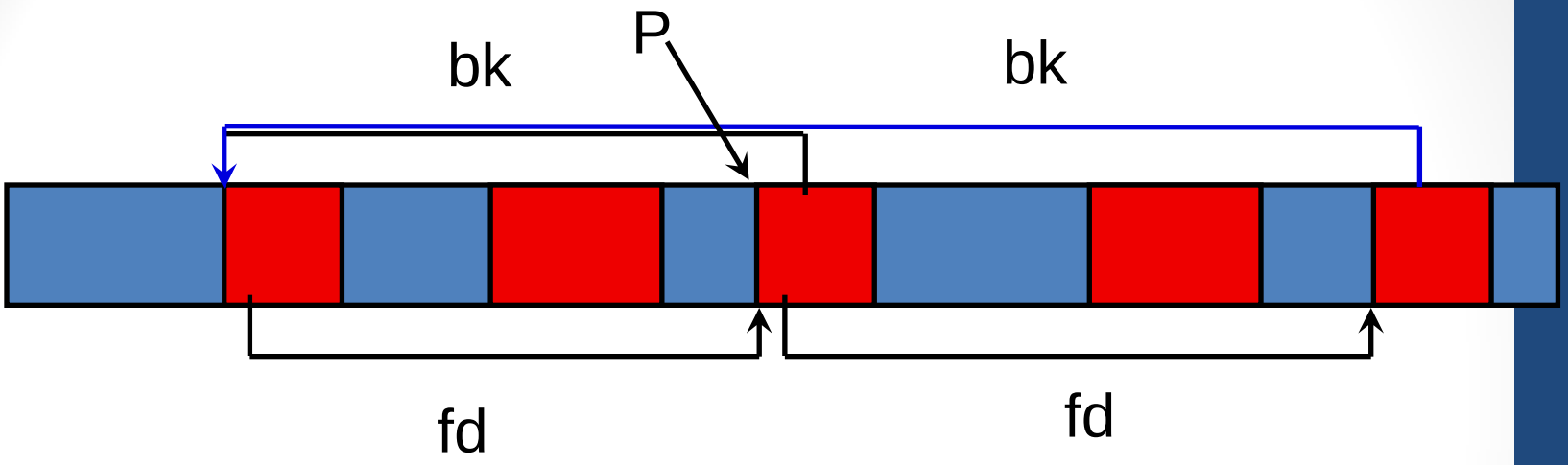


Unlinking



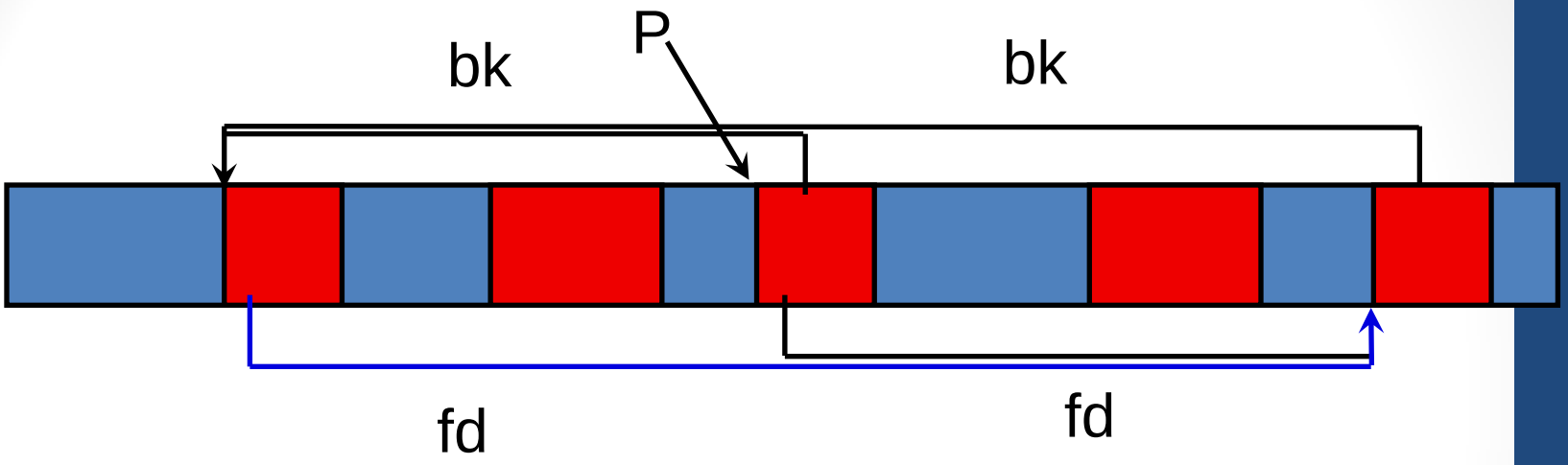

```
#define unlink( P, BK, FD ) { \
  BK = P->bk;                 \
  FD = P->fd;                 \
  FD->bk = BK;                \
  BK->fd = FD;                \
}
```

Unlinking



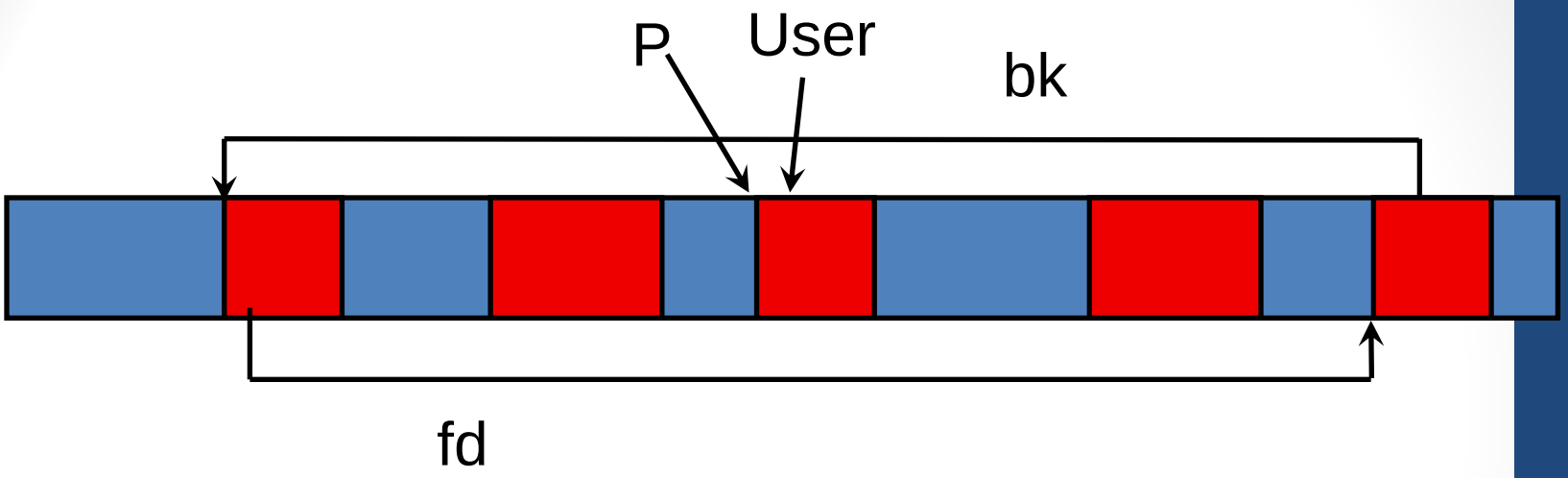

```
#define unlink( P, BK, FD ) { \
  BK = P->bk;                 \
  FD = P->fd;                 \
  FD->bk = BK;                 \
  BK->fd = FD;                 \
}
```

Unlinking



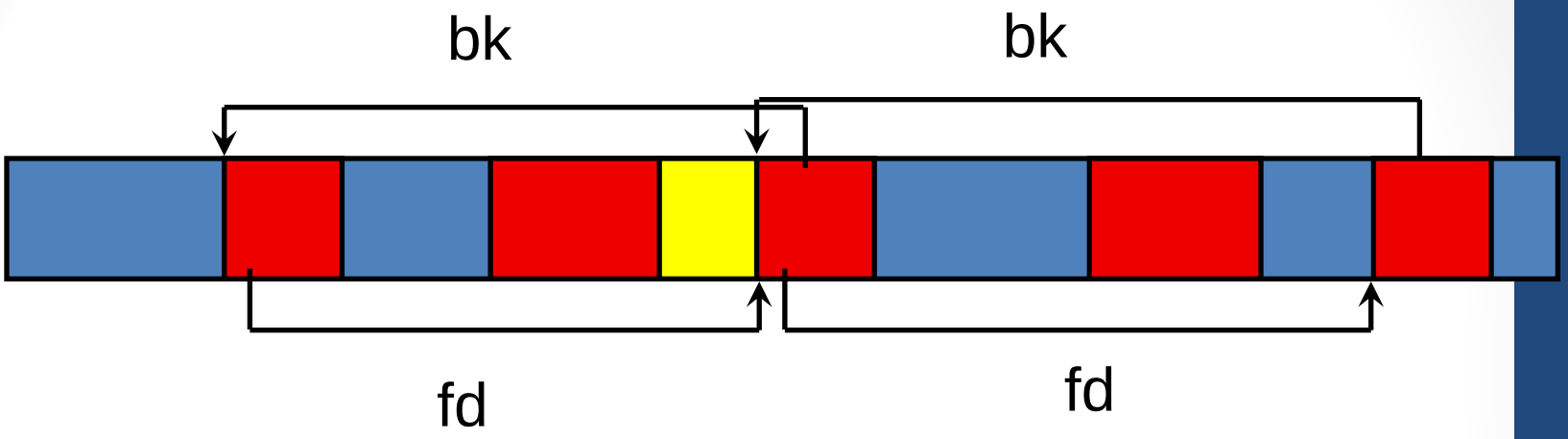

```
#define unlink( P, BK, FD ) { \
  BK = P->bk;                 \
  FD = P->fd;                 \
  FD->bk = BK;                \
  BK->fd = FD;                \
}
```

Unlinking

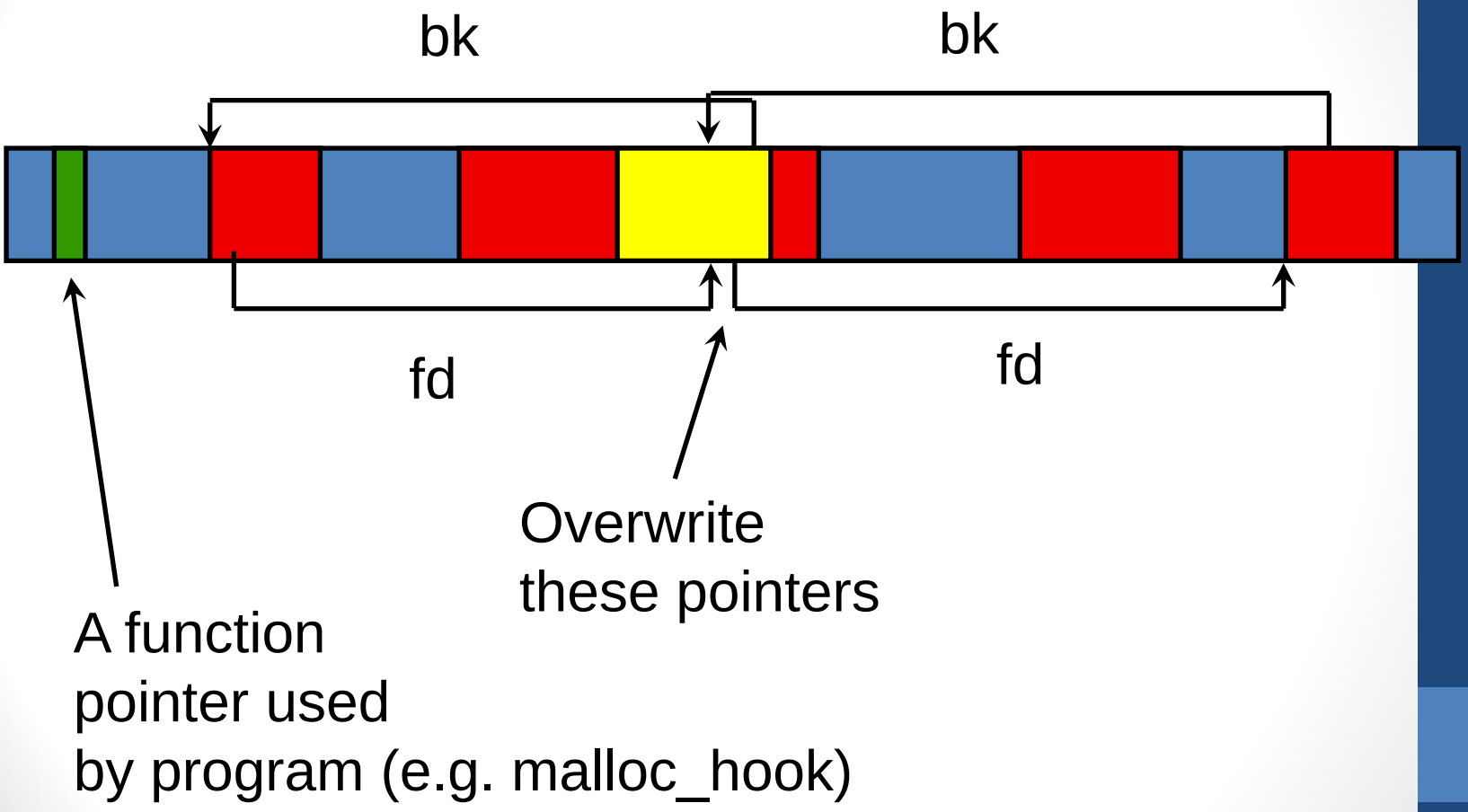



```
#define unlink( P, BK, FD ) { \
  BK = P->bk;                 \
  FD = P->fd;                 \
  FD->bk = BK;                \
  BK->fd = FD;                \
}
```

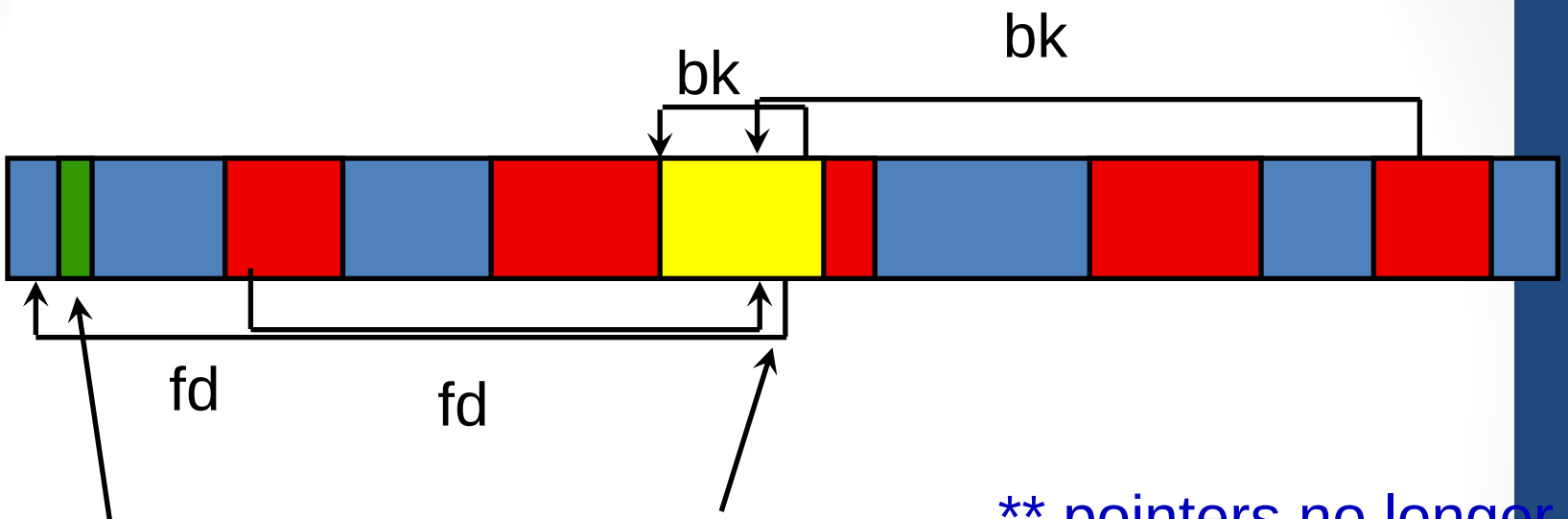
The Vulnerable Buffer



The Vulnerable Buffer



The New Pointers



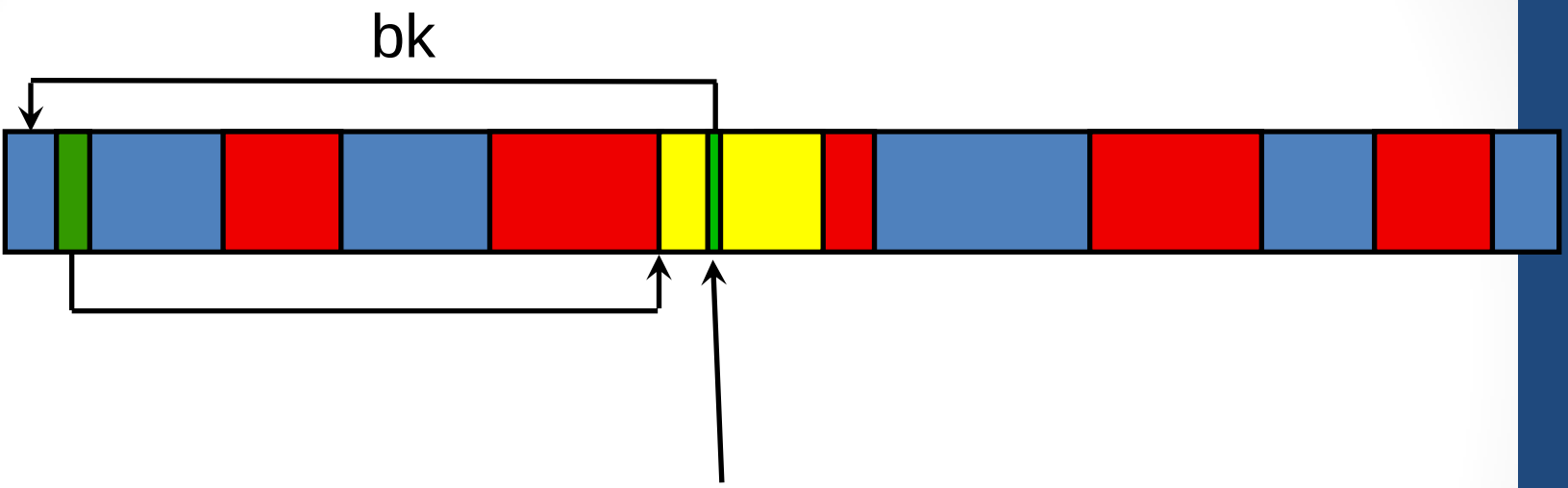
Overwrite these pointers

A function pointer used by program (e.g. _malloc_hook) -- located fd+8

** pointers no longer point to free blocks

** wait for a malloc call.....

After Unlinking...



- 4 bytes at offset 8 get overwritten
- shell code has to jmp around..

**Next time the function pointer is used...
Our code gets executed!!

Buffer Overflow

- Other Examples
  - PDF Javascript Bug
  - Outlook Date Bug

- Whats the point
  - not here to teach you how to break in.
  - illustrate how easy it is to take advantage of errors
  - implications of certain classes of errors in code.

Program Threats

- Race Conditions
  - ◊ suid programs (programs that run with administrator priveledges)
  - ◊ make a security check before doing an action
  - ◊ do the action

In the moment between check and do, attacker switches the action. Often involves files in /tmp directory (writable by anyone)

protection: don’t execute something the user can change!!

Program Threats

- Checking parameters
  - ◊ shell scripts on unix. File contains:
    %!/bin/sh
    …shell commands…
  - ◊ execute with -i flag (means interactive shell)
  - ◊ if setuid shell script, now interactive shell in other users name
  - ◊ Most Unixes now do not support setuid shell scripts

Program Threats

- Checking parameters
  - ◊ web parameters
  - ◊ execute a system command using parameters taken from a web form
    e.g. “mail -f confirmation $remote_address”
    - where remote_address comes from web form
    - remote_address contains
      “joe@foo.com ; rm -rf /*”
  - ◊ cannot rely on javascript to verify form data
    - anyone can write a program to send data to a web server!!

Program Threats

- Checking parameters
  - ◊ SQL Injection
  - ◊ Take user input and insert into a query

SELECT from Table1 where Parm=’<user input here>’

user input = fred’;update employee set salary=70000
where emp=’barney

Program Threats

- Checking parameters
◊ SQL Injection
◊ Take user input and insert into a query

SELECT from Table1 where Parm='<user input here>'

user input = fred';update employee set salary=70000
where emp='barney

Program Threats

- Checking parameters
◊ SQL Injection
◊ Take user input and insert into a query

SELECT from Table1 where Parm='<user input here>'

user input = fred';update employee set salary=70000
where emp='barney

◊ must scan input for key (escape) characters before issuing database commands

System Threats

- Virus
  - ◊ covers a lot of ground
  - ◊ trojan horse as vector
  - ◊ infects boot sector/other programs
  - ◊ macro viruses
  - ◊ mail viruses
  - ◊ often combined with other attacks
    - date overflow bug
  - ◊ more sophisticated
    - contains own mail servers
    - camouflage

System Threats

- Worms
  - ◊ Automated program that breaks into another system and creates a copy on the new system
  - ◊ soon running on many vulnerable systems
  - ◊ can take a delayed action (Code Red)

- Distinction between worm and virus is the vector. Virus needs a human action, worm contains code to attack the next machine.
  - ◊ fuzzy distinction, two techniques have been merging for some time