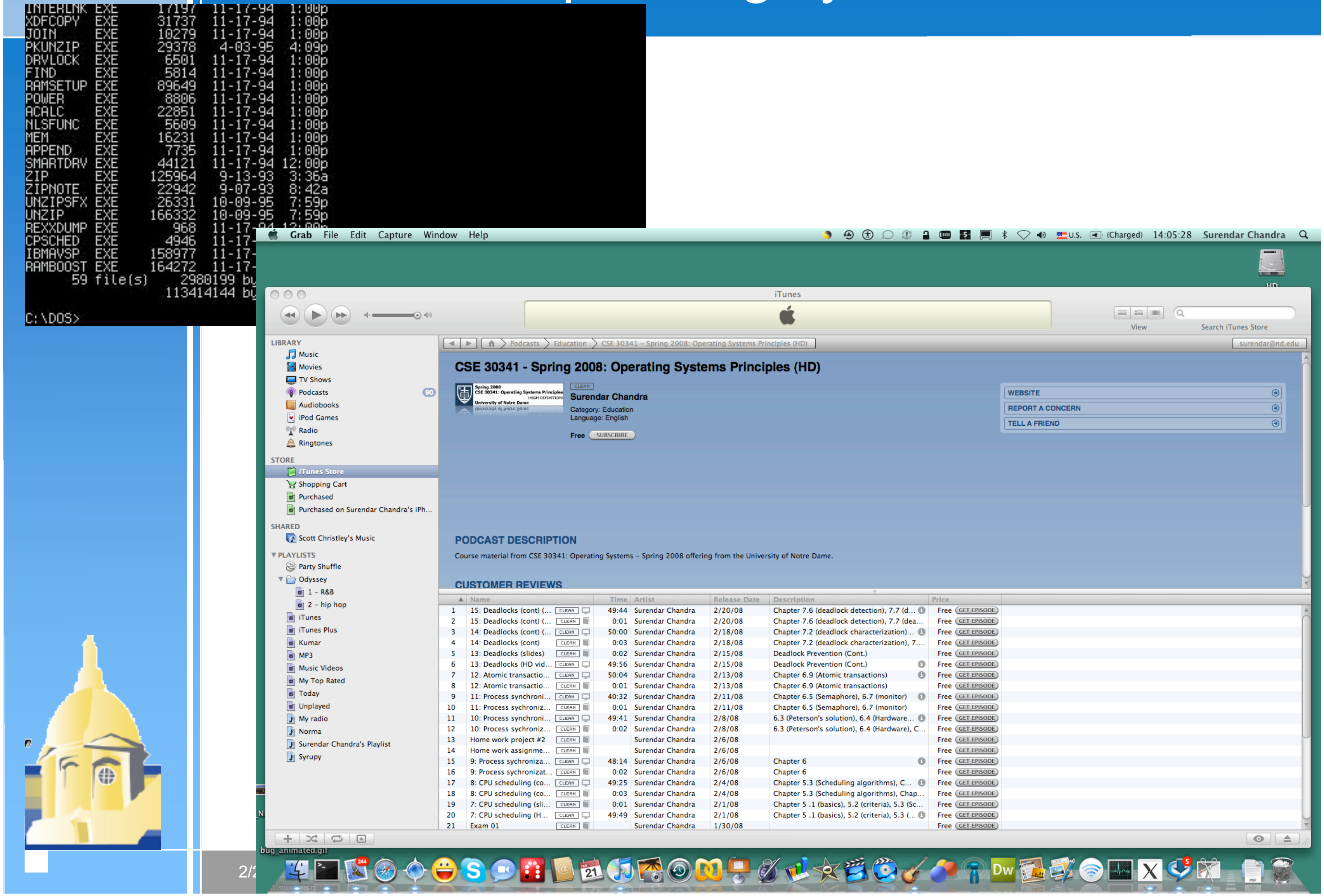
Survey feedback

- “I definately agree that we need more code in class. The idea of an operating system for a lot of newbie cse's is what they witness for mac, windows or linux. Very finite and tangible. With the openness of your lectures we get an image of something much more theoretical, so there is an added challenge of connecting the concrete to abstract. I feel that by presenting more (and more 'real', not just a loop segment would help put in context) examples of code we could understand OS's much more clearly and wholly.”

What is an operating system?



```
INTERLNK EXE     17197  11-17-94   1:00p
XDFCOPY  EXE     31737  11-17-94   1:00p
JOIN     EXE     10279  11-17-94   1:00p
PKUNZIP  EXE     29378   4-03-95   4:09p
DRVLOCK  EXE      6501  11-17-94   1:00p
FIND     EXE      5814  11-17-94   1:00p
RAMSETUP EXE     89649  11-17-94   1:00p
POWER    EXE      8806  11-17-94   1:00p
ACALC    EXE     22851  11-17-94   1:00p
NLSFUNC  EXE      5609  11-17-94   1:00p
MEM      EXE     16231  11-17-94   1:00p
APPEND   EXE      7735  11-17-94   1:00p
SMARTDRV EXE     44121  11-17-94  12:00p
ZIP      EXE    125964   9-13-93   3:36a
ZIPNOTE  EXE     22942   9-07-93   8:42a
UNZIPSFX EXE     26331  10-09-95   7:59p
UNZIP    EXE    166332  10-09-95   7:59p
REXXDUMP EXE       968  11-17-94  12:00p
CPSCHED  EXE      4946  11-17-
IBMAVSP  EXE    158977  11-17-
RAMBOOST EXE    164272  11-17-
      59 file(s)      2980199 by
                    113414144 by

C:\DOS>
```

CSE 30341 - Spring 2008: Operating Systems Principles (HD)

Surendar Chandra

Category: Education
Language: English

Free

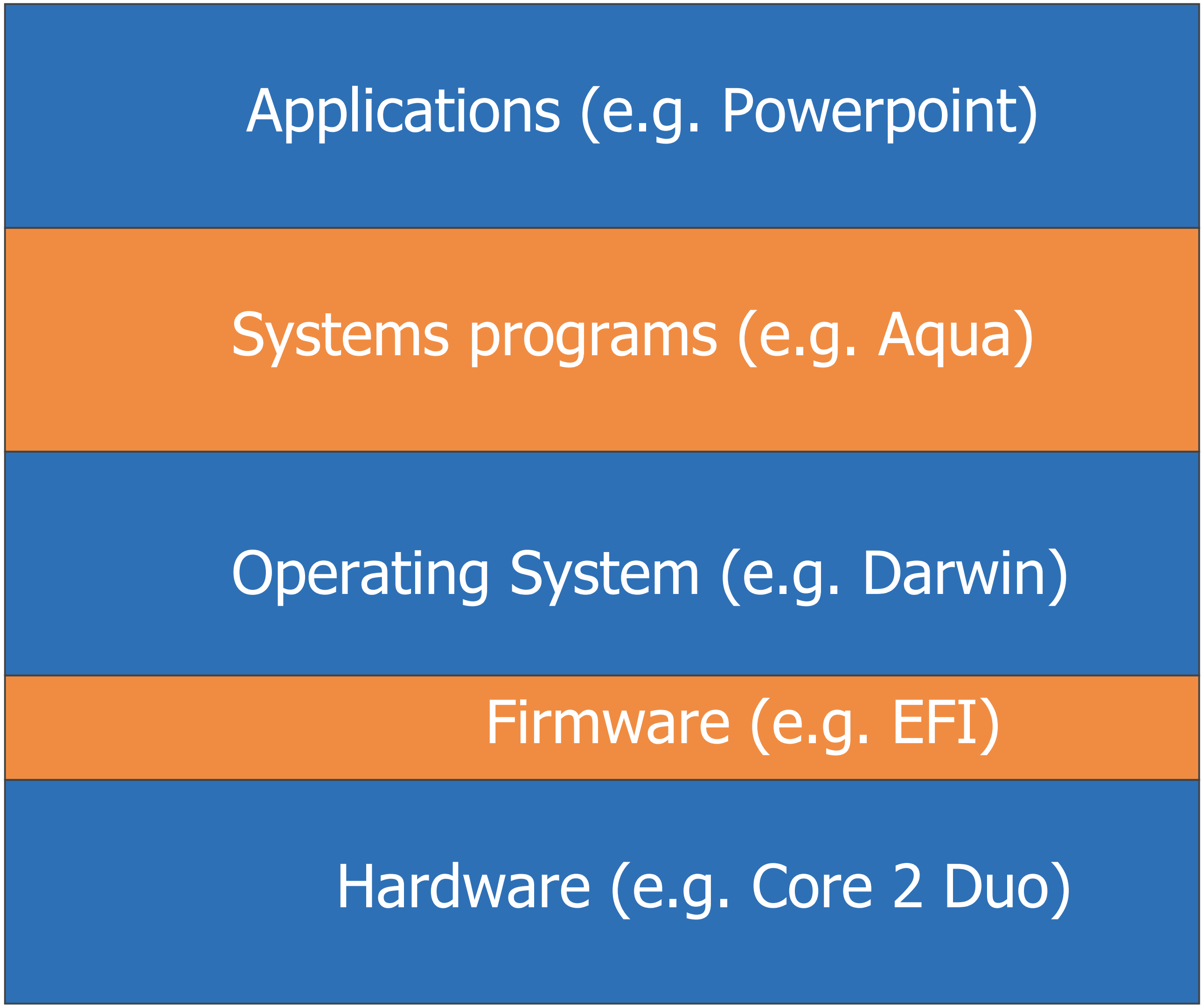
PODCAST DESCRIPTION

Course material from CSE 30341: Operating Systems – Spring 2008 offering from the University of Notre Dame.

CUSTOMER REVIEWS

▲	Name	Time	Artist	Release Date	Description	Price
1	15: Deadlocks (cont) (... CLEAN	49:44	Surendar Chandra	2/20/08	Chapter 7.6 (deadlock detection), 7.7 (d...	Free GET EPISODE
2	15: Deadlocks (cont) (... CLEAN	0:01	Surendar Chandra	2/20/08	Chapter 7.6 (deadlock detection), 7.7 (dea...	Free GET EPISODE
3	14: Deadlocks (cont) (... CLEAN	50:00	Surendar Chandra	2/18/08	Chapter 7.2 (deadlock characterization)...	Free GET EPISODE
4	14: Deadlocks (cont) CLEAN	0:03	Surendar Chandra	2/18/08	Chapter 7.2 (deadlock characterization), 7....	Free GET EPISODE
5	13: Deadlocks (slides) CLEAN	0:02	Surendar Chandra	2/15/08	Deadlock Prevention (Cont.)	Free GET EPISODE
6	13: Deadlocks (HD vid... CLEAN	49:56	Surendar Chandra	2/15/08	Deadlock Prevention (Cont.)	Free GET EPISODE
7	12: Atomic transactio... CLEAN	50:04	Surendar Chandra	2/13/08	Chapter 6.9 (Atomic transactions)	Free GET EPISODE
8	12: Atomic transactio... CLEAN	0:01	Surendar Chandra	2/13/08	Chapter 6.9 (Atomic transactions)	Free GET EPISODE
9	11: Process synchroni... CLEAN	40:32	Surendar Chandra	2/11/08	Chapter 6.5 (Semaphore), 6.7 (monitor)	Free GET EPISODE
10	11: Process sychroniz... CLEAN	0:01	Surendar Chandra	2/11/08	Chapter 6.5 (Semaphore), 6.7 (monitor)	Free GET EPISODE
11	10: Process synchroni... CLEAN	49:41	Surendar Chandra	2/8/08	6.3 (Peterson's solution), 6.4 (Hardware...	Free GET EPISODE
12	10: Process sychroniz... CLEAN	0:02	Surendar Chandra	2/8/08	6.3 (Peterson's solution), 6.4 (Hardware), C...	Free GET EPISODE
13	Home work project #2 CLEAN		Surendar Chandra	2/6/08		Free GET EPISODE
14	Home work assignme... CLEAN		Surendar Chandra	2/6/08		Free GET EPISODE
15	9: Process sychroniza... CLEAN	48:14	Surendar Chandra	2/6/08	Chapter 6	Free GET EPISODE
16	9: Process sychronizat... CLEAN	0:02	Surendar Chandra	2/6/08	Chapter 6	Free GET EPISODE
17	8: CPU scheduling (co... CLEAN	49:25	Surendar Chandra	2/4/08	Chapter 5.3 (Scheduling algorithms), C...	Free GET EPISODE
18	8: CPU scheduling (co... CLEAN	0:03	Surendar Chandra	2/4/08	Chapter 5.3 (Scheduling algorithms), Chap...	Free GET EPISODE
19	7: CPU scheduling (sli... CLEAN	0:01	Surendar Chandra	2/1/08	Chapter 5 .1 (basics), 5.2 (criteria), 5.3 (Sc...	Free GET EPISODE
20	7: CPU scheduling (H... CLEAN	49:49	Surendar Chandra	2/1/08	Chapter 5 .1 (basics), 5.2 (criteria), 5.3 (...	Free GET EPISODE
21	Exam 01 CLEAN		Surendar Chandra	1/30/08		Free GET EPISODE

What is an operating systems?



Applications (e.g. Powerpoint)

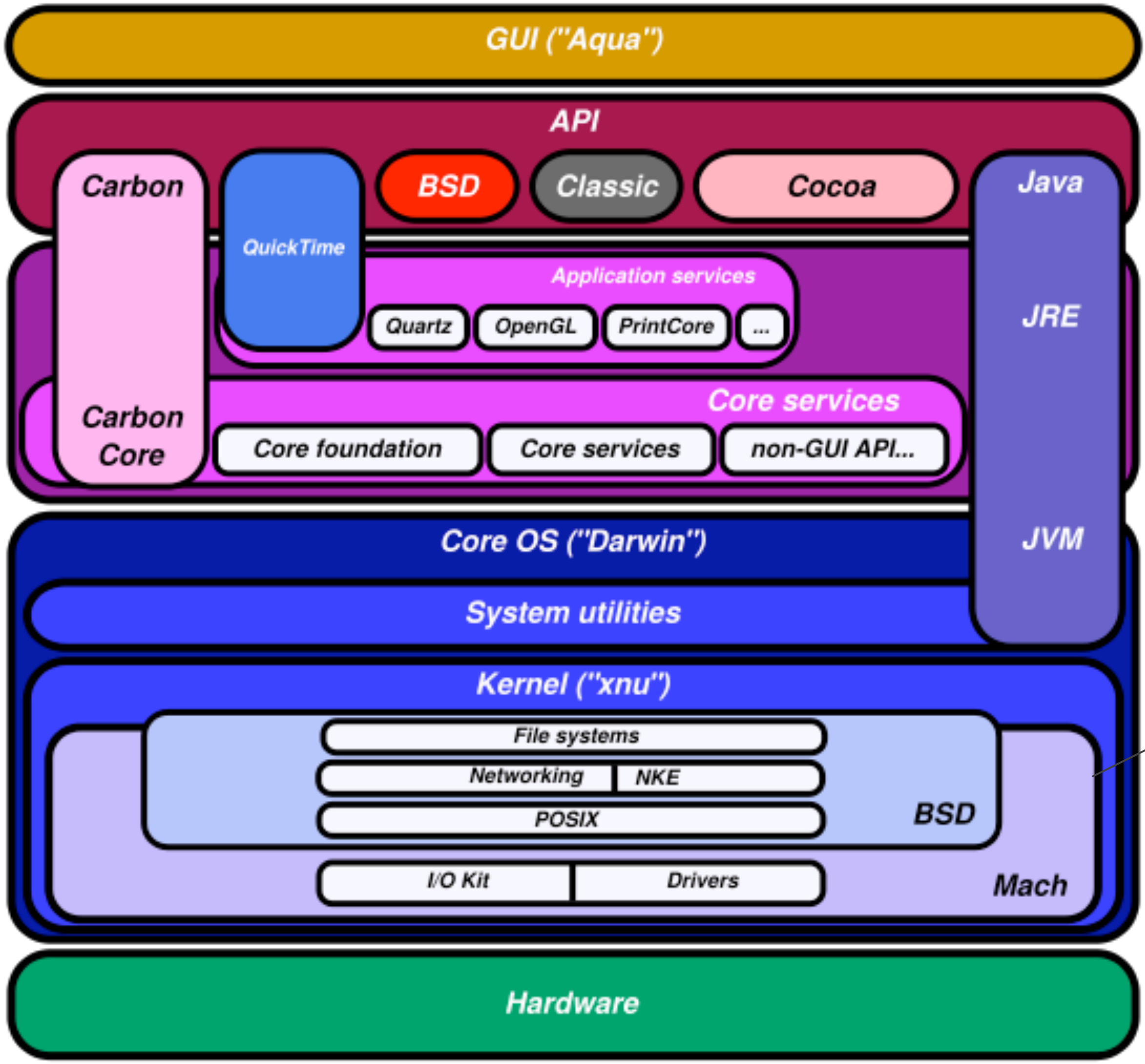
Systems programs (e.g. Aqua)

Operating System (e.g. Darwin)

Firmware (e.g. EFI)

Hardware (e.g. Core 2 Duo)

Mac OSX (from Wikipedia)



Our focus

Recap

- Module 1:
  - Process is an abstraction for a program in execution
  - Threads is a way to assign multiple processors to a program
  - PCB internally represents a process to the OS
  - System calls are a way for applications (user level) to communicate and get services from OS (kernel level)
  - Scheduling algorithm (OS service) decides which thread to run and for how long

Module 2

- Challenge: Multiple threads running inside a process can cause race conditions for some applications
  - Threads that did not share data were fine
  - Threads that shared data can give unpredictable results
    - One solution is to force each thread to behave in a predictable fashion - significant slowdown because you cannot use unexpected slack (extra CPU)
    - Another solution is to force some partial order which is flexible enough and yet gives you good performance
      - Notion of conflict serializability

Module 2: Critical sections

- We defined notions of critical sections - to be used by the programmer.
  - need hardware support (TestandSet, Swap)
  - it is sufficiently important that you get OS support
    - With OS support, you can implement block vs spinlock

- The whole module is concerned with the problem faced by application program that exploit a feature provided by the OS (threads) and the solutions. A bit of the solution involves the Operating System. Some of the other services that could be provided by the OS (deadlock prevention/detection) is too hard that no OS actually implements it

Kernel mutex lock

```
 __mutex_fastpath_lock(&lock->count, __mutex_lock_slowpath);

#define __mutex_fastpath_lock(v, fail_fn)
do {
        unsigned long dummy;

        typecheck(atomic_t *, v);
         typecheck_fn(void (*)(atomic_t *), fail_fn);
        __asm__ __volatile__(
                LOCK_PREFIX "   decl (%%rdi)     \n"
                        "   jns 1f              \n"
                        "   call "#fail_fn"     \n"
                        "1:"

                :"=D" (dummy)
                : "D" (v)
                : "rax", "rsi", "rdx", "rcx",
                  "r8", "r9", "r10", "r11", "memory");
} while (0)
```

Review

- Critical section problem: Primarily a problem for threaded application that share some data. There is a need to ensure that only one thread gets to be inside the critical section. There is a need to be fair
- Semaphore, monitors are good programming abstraction
- Typical problem with threaded, shared data
  - Bounded buffer
  - Reader-writer: More than one reader inside CS
  - Dining philosophers: solutions dead-lock prone

Atomic transactions

- Notions of Databases
  - Transaction, commit, abort, rollback/roll forward, logs
  - Serializability, conflict serializability

- Deadlocks:
  - A phenomenon faced by applications that used multiple mutually exclusive resources
  - OS can prevent deadlocks during allocation or detect deadlocks after they have happened
    - Applications can use these techniques while requesting locks
  - Most current OS's do not do any of them

Survey concern

- “Programming examples would be good for help in identifying critical sections for multithreaded processes, and how to handle them”
- Identifying critical sections requires a deep understanding of your program. The finer grain you have, the more performance you achieve
  - Steps: Identify all shared variables
    - Lock all accesses to them
  - Deeply understand the program and only lock accesses that are likely to cause a problem
    - Miss a variable that should’ve been protected - tragedy

Code example

```
Func()
{
    for (int I=start_x; I < end_x; I++)
        for (int j=start_y; j < end_y; j++)
            matrix[I][j]=10;
}
```

thread_create func() with start_x, end_x, start_y, end_y equalling (0,10,0,10), (11,20,11,20)

might not require locks