SSD 6

icarnegie

curriculum powered by Carnegie Mellon

Lecture 6 (1/2)

Performance Measurement and Improvement

武漢大學

国际软件学院

Lecture 3 Performance Measurement and Improvement

The Contents in SSD6 cover:

4.1 Measurement and Profiling

Where to start? Where to stop? How-to?

4.2 Hot Spots

Find the weak-knees!

4.3 Practical Hints

Join a tour…

Lecture 3 Performance Measurement and Improvement

Key Points:

Trade-offs

- Making program run fast!
- Making program run correct!
- Which is more important?
- Computers double in speed every 18 months

General Hints:

- Understand what makes the program slow.
  - A few places account for a lot of time.
- Design and implement a solution
  - The most speedup for the least effort
- Iterate steps 1 and 2
  - until performance is satisfactory or,
  - until the remaining optimizations are not worth the effort

Lecture 3 Measurement and Profiling

Where to measure:

- Where programs spend most of their time.
  - Don't optimize a under-dev. program.
- CPU time
  - For instruction execution
  - System time vs. User time
- Wall time
  - For all elapsed time
- Difference between 2&3

Lecture 3 Measurement and Profiling

Time scale:

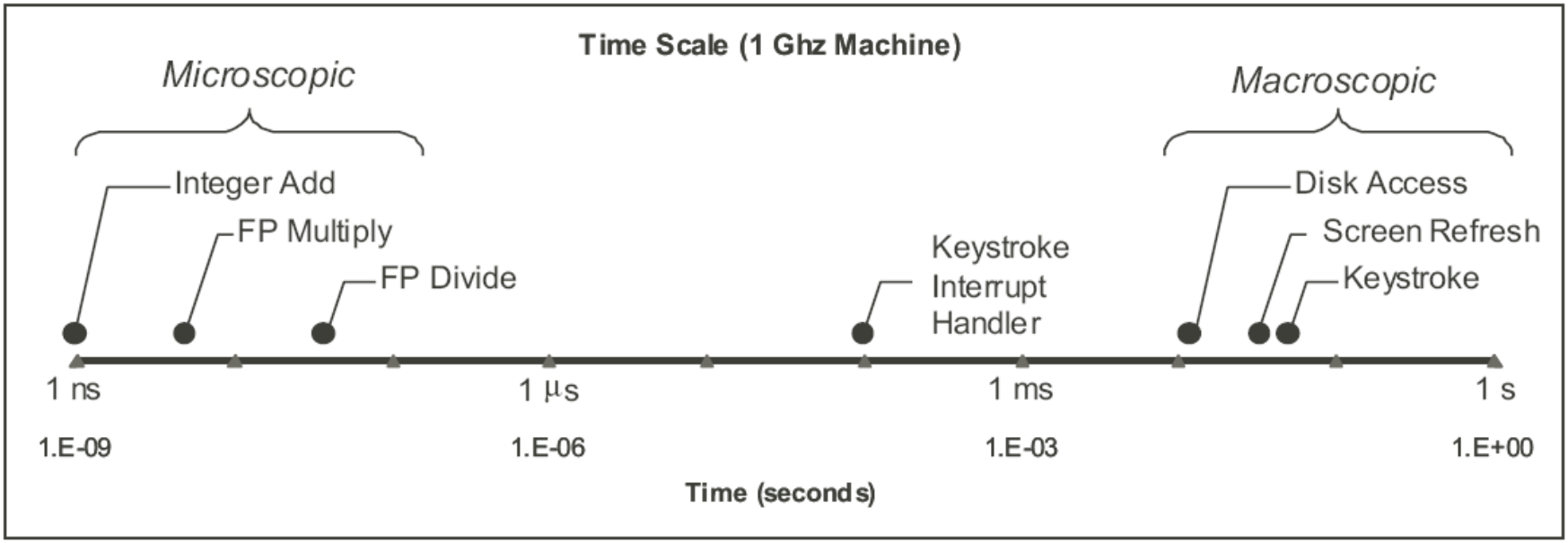


Figure 9.1: **Time Scale of Computer System Events.** The processor hardware works at a microscopic a time scale in which events having durations on the order of a few nanoseconds (ns). The OS must deal on a macroscopic time scale with events having durations on the order of a few milliseconds (ms).

Lecture 3 Measurement and Profiling

Time scale:

- **Computers operate on two fundamentally different time scales**
- **At a microscopic level**
  - To execute instructions at a rate of one or more per clock cycle, where each clock cycle requires only around one nanosecond (abbreviated "*ns*")
- **On a macroscopic scale**
  - processor must respond to external events that occur on time scales measured in milliseconds (abbreviated "*ms*")

Lecture 3 Measurement and Profiling

For example:

- During video playback, the graphics display for most computers must be refreshed every 33 *ms*
- A world-record typist can only type keystrokes at a rate of around one every 50 *milliseconds*
- Disks typically require around 10 *ms* to initiate a disk transfer
- The processor continually switches between these many tasks on a macroscopic time scale, devoting around 5 to 20 *milliseconds* to each task at a time
- People cannot discern time durations shorter than around 100 *ms*. Within that time the processor can execute millions of instructions

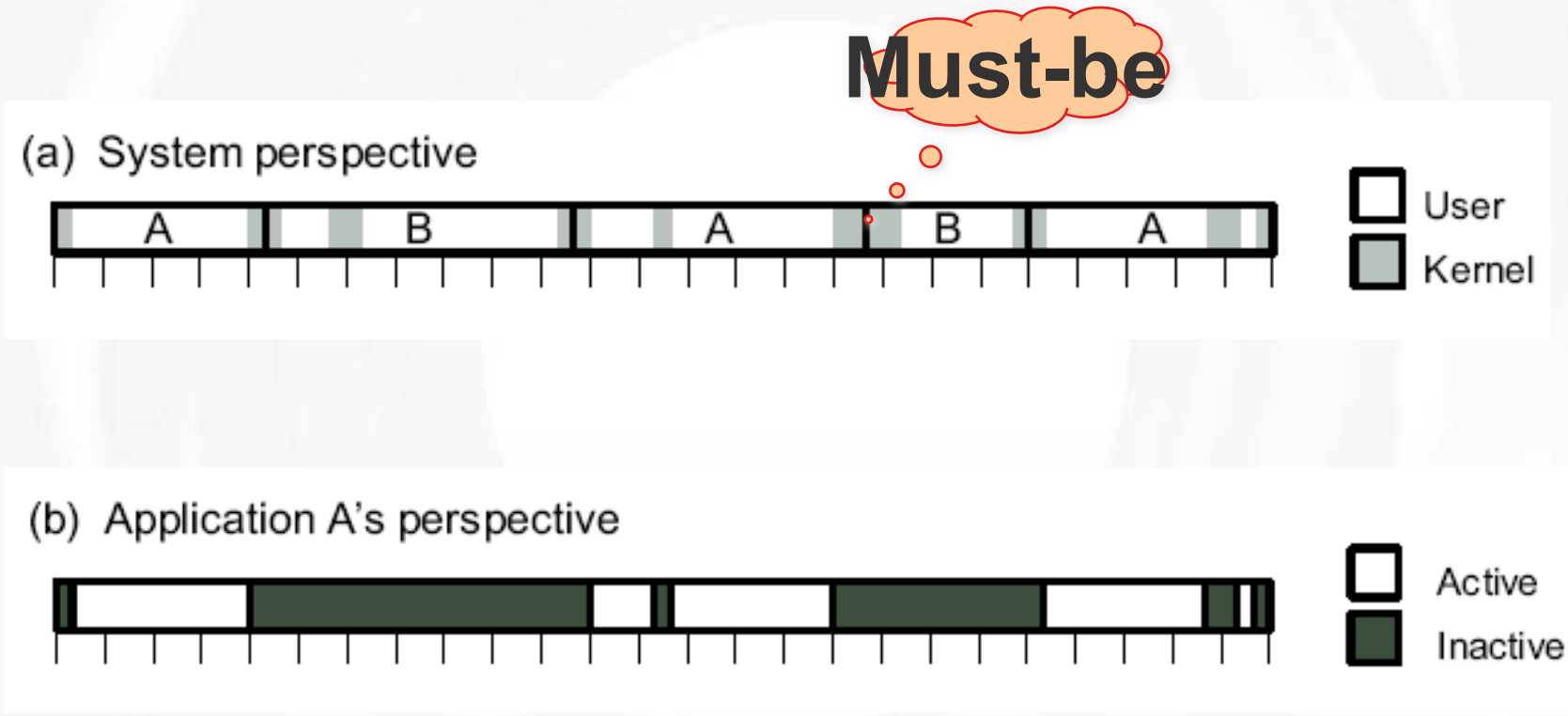
Lecture 3 Measurement and Profiling

Process Scheduling and Timer Interrupts

- Computers have an external timer that periodically generates an interrupt signal to the processor
- The spacing between these interrupt signals is called the ***interval time***
- When a timer interrupt occurs, the OS scheduler can choose to either resume the currently executing process or to switch to a different process.
- This interval must be set short enough to ensure that the processor will switch between tasks
- Typical timer intervals range between 1 and 10 milliseconds

Lecture 3 Measurement and Profiling

The Flow of Time on a Computer System:



Lecture 3 Measurement and Profiling

Time from an Application Program's Perspective

- Active *vs.* Inactive

```
A0    Time        0    (0.00 ms),   Duration  3726508    (6.776448 ms)
I0    Time  3726508    (6.78 ms),   Duration   275025    (0.500118 ms)
A1    Time  4001533    (7.28 ms),   Duration        0    (0.000000 ms)
I1    Time  4001533    (7.28 ms),   Duration     7598    (0.013817 ms)
A2    Time  4009131    (7.29 ms),   Duration  5189247    (9.436358 ms)
I2    Time  9198378   (16.73 ms),   Duration   251609    (0.457537 ms)
A3    Time  9449987   (17.18 ms),   Duration  2250102    (4.091686 ms)
I3    Time 11700089   (21.28 ms),   Duration    14116    (0.025669 ms)
A4    Time 11714205   (21.30 ms),   Duration  2955974    (5.375275 ms)
I4    Time 14670179   (26.68 ms),   Duration   248500    (0.451883 ms)
A5    Time 14918679   (27.13 ms),   Duration  5223342    (9.498358 ms)
I5    Time 20142021   (36.63 ms),   Duration   247113    (0.449361 ms)
A6    Time 20389134   (37.08 ms),   Duration  5224777    (9.500967 ms)
I6    Time 25613911   (46.58 ms),   Duration   254340    (0.462503 ms)
A7    Time 25868251   (47.04 ms),   Duration  3678102    (6.688425 ms)
I7    Time 29546353   (53.73 ms),   Duration     8139    (0.014800 ms)
A8    Time 29554492   (53.74 ms),   Duration  1531187    (2.784379 ms)
I8    Time 31085679   (56.53 ms),   Duration   248360    (0.451629 ms)
A9    Time 31334039   (56.98 ms),   Duration  5223581    (9.498792 ms)
I9    Time 36557620   (66.48 ms),   Duration   247395    (0.449874 ms)
```

Lecture 3 Measurement and Profiling

Time from an Application Program's Perspective

- Active ***vs.*** Inactive

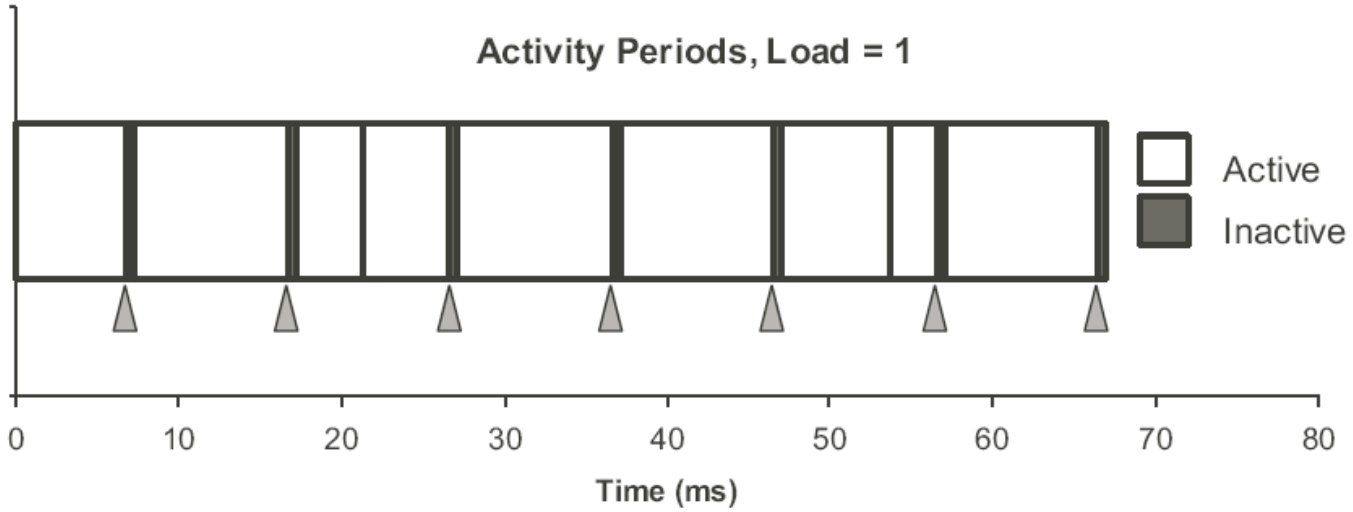


Figure 9.4: **Graphical Representation of Trace in Figure 9.3.** Timer interrupts are indicated with gray triangles.

Lecture 3 Measurement and Profiling

Measuring Time by Interval Counting [间隔计数]

- The operating system also uses the timer to record the cumulative time used by each process.
- This information provides a somewhat imprecise measure of program execution time
- A ***time segment*** refers to the period during which just one process executes.

Lecture 3 Measurement and Profiling

Measuring Time by Interval Counting [间隔计数]

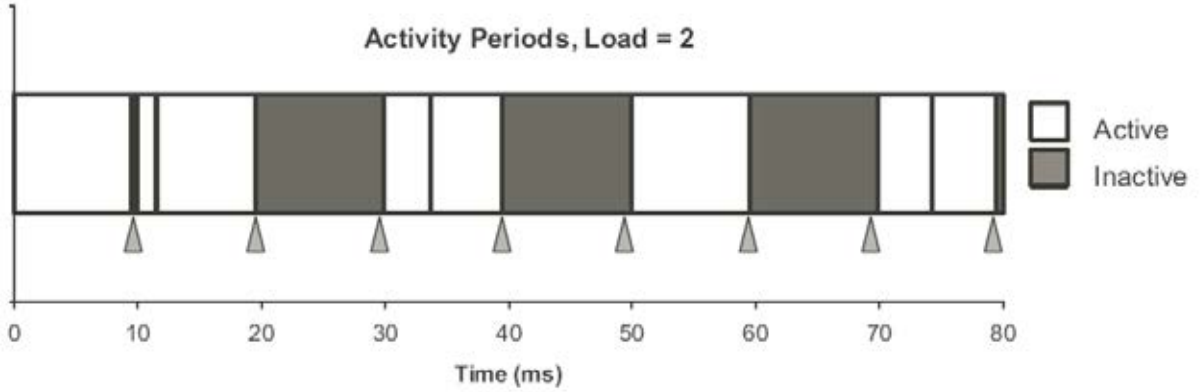


Figure 9.6: **Graphical Representation of Activity Periods for Trace in Figure 9.5.** Timer interrupts are indicated by gray triangles

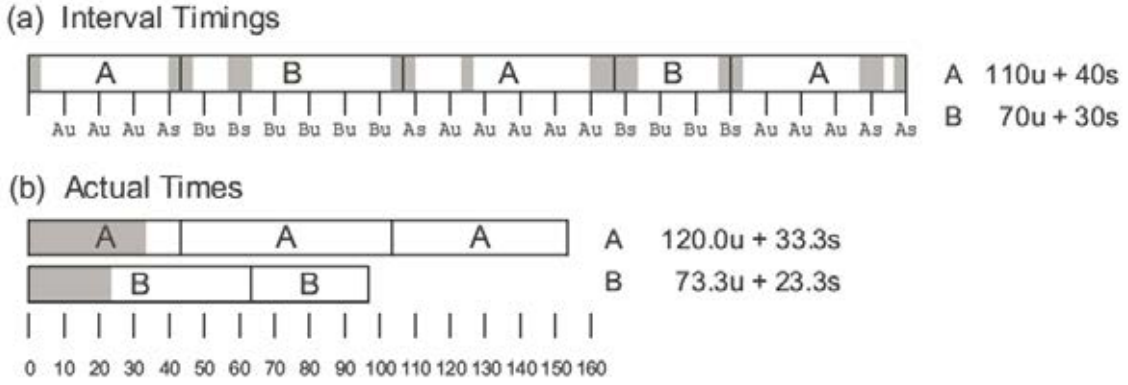


Figure 9.7: **Process Timing by Interval Counting.** With a timer interval of 10 ms, every 10 ms segment is assigned to a process as part of either its user (u) or system (s) time. This accounting provides only an approximate measure of program execution time.

Lecture 3 Measurement and Profiling

Measuring Time by Interval Counting [间隔计数]

unix> time prog -n 17

2.230u 0.260s 0:06.52 38.1% 0+0k 0+0io 80pf+0w

- The first three numbers shown in this line are times.
- The first two show the seconds of user and system time.
- With a timer interval of 10 ms, all timings are multiples of hundredths of seconds. The third number is the total elapsed time.
- Observe that the system and user time sum to 2.49 seconds, less than half of the elapsed time of 6.52 seconds.
- The percentage indicates what fraction the combined user and system times were of the elapsed time, e.g. (2.23+0.26)/6.52=0.381
- The remaining statistics summarize the paging and I/O behavior.

Lecture 3 Measurement and Profiling

Measuring Time by Interval Counting [间隔计数]

Do U remember the RECURSION example?

Gzhu$ time ./recursion 10000

18%

```
real    0m0.456s
user    0m0.031s
sys     0m0.052s
```

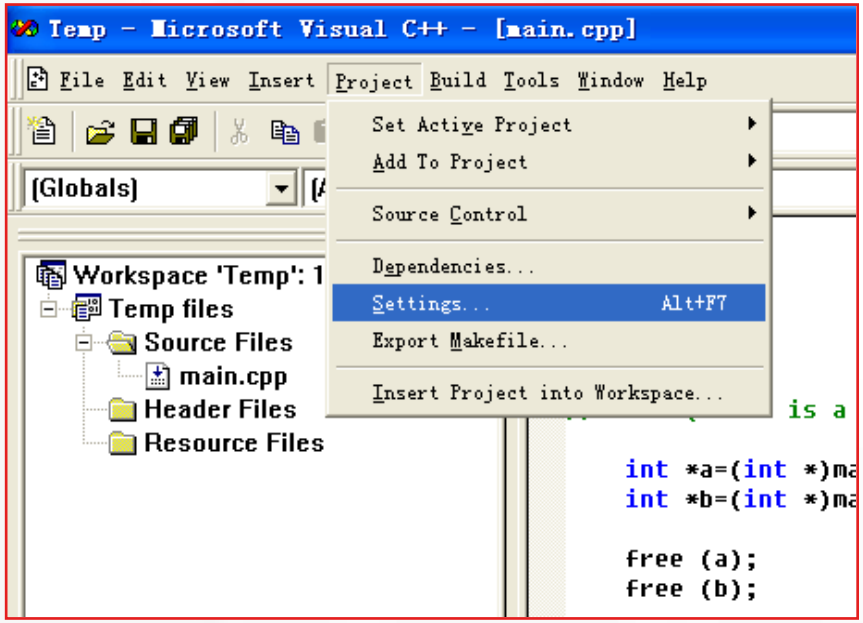
Lecture 3 Measurement and Profiling

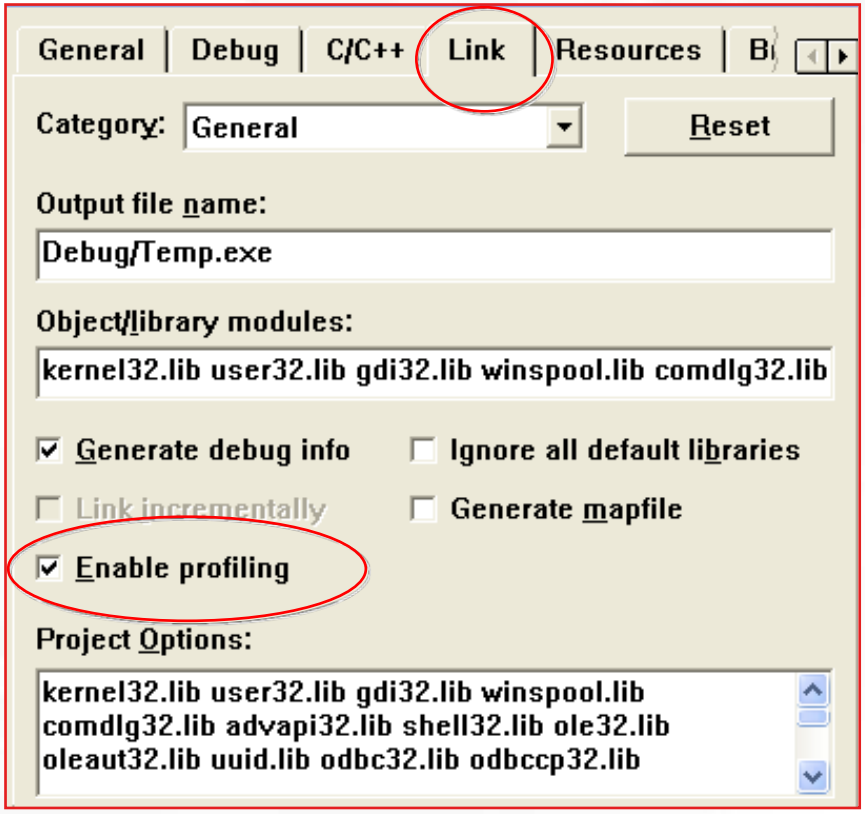
Timing Mechanisms:

- Using timer directly
  - See precise.zip (downloadable).
- Using *statistical sampling*
  - Timer periodically interrupts the program and records the program counter or increments a counter representing a range of program counters
- Not necessary to implement from scratch.
  - Profiling

Lecture 3 Measurement and Profiling

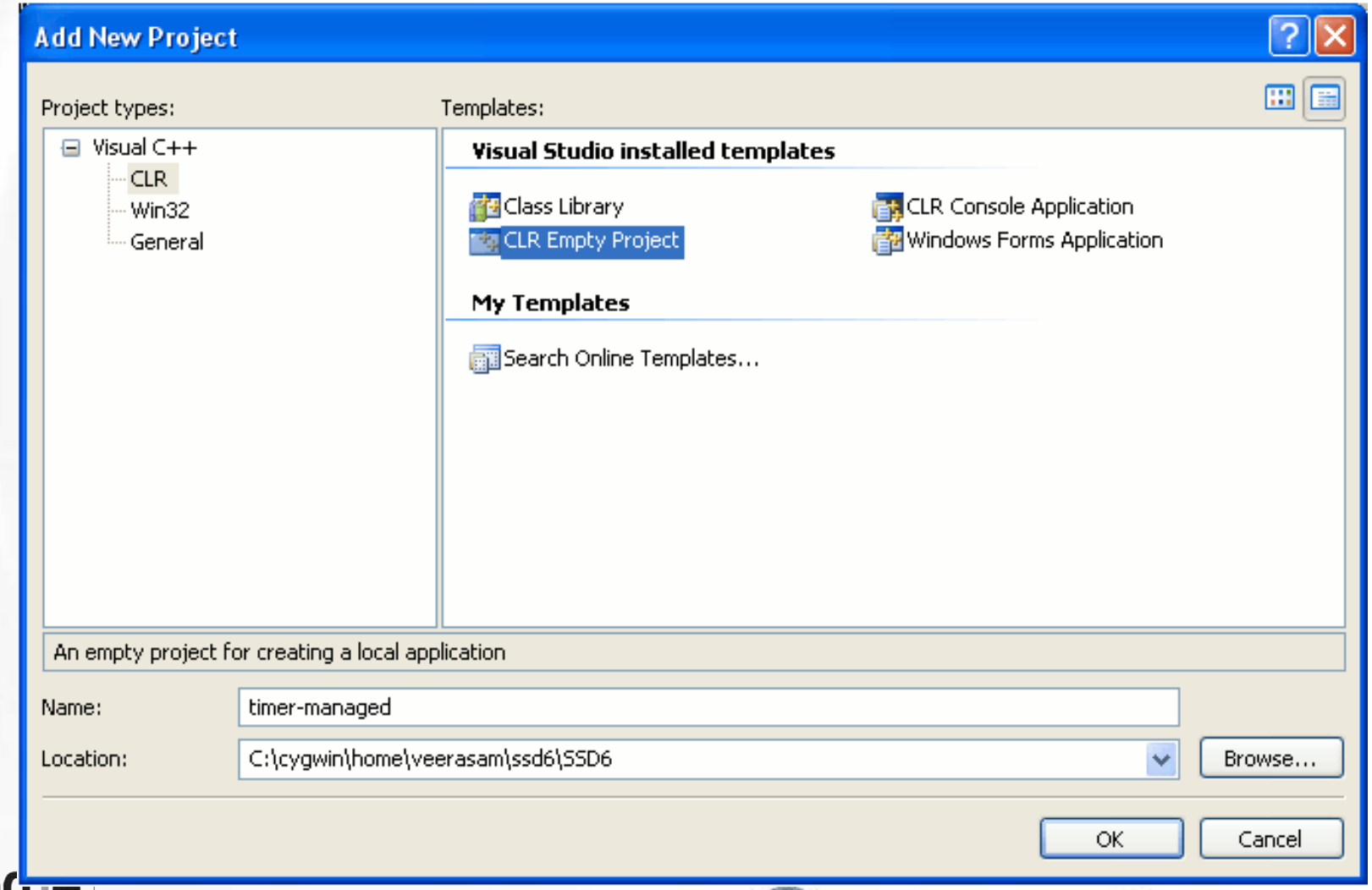
Setting profiling:





Lecture 3 Measurement and Profiling

In Visual Studio .Net C++:



Lecture 3 Measurement and Profiling

In Visual Studio .Net C++:

See:
profiler-mod

1. open a MS-DOS command prompt window, and run the **EnableProfiler.bat**
2. The batch file will register the profiler in your computer's **registry**.
3. Then, run your **managed application** (*.exe) from the ***same*** MS-DOS window to profile your program.
4. The profiling data will be written to a file named ***output.log***.

Lecture 3 Measurement and Profiling

In Visual Studio .Net C++:

- **Thread ID:** The thread under which the function executed
- **Function:** Name of the function
- **Times Called:** The number of times the function was called
- **Exclusive Time:** Amount of time (in seconds) spent in the function excluding time spent in its callees, Suspended Time and Profiler Time
- **Callee Exclusive Time:** Amount of time (in seconds) spent in the function and its callees (children) excluding Suspended Time and Profiler Time
- **Inclusive Time:** Amount of time (in seconds) spent in the function including Callee Time, Suspended Time and Profiler Time
- **Callee Time:** Amount of time (in seconds) spent in the callees (including Suspended Time and Profiler Time spent under the callees)
- **Suspended Time:** Amount of suspended time (in seconds)
- **Profiler Time:** Amount of time spent by profiler (in seconds)

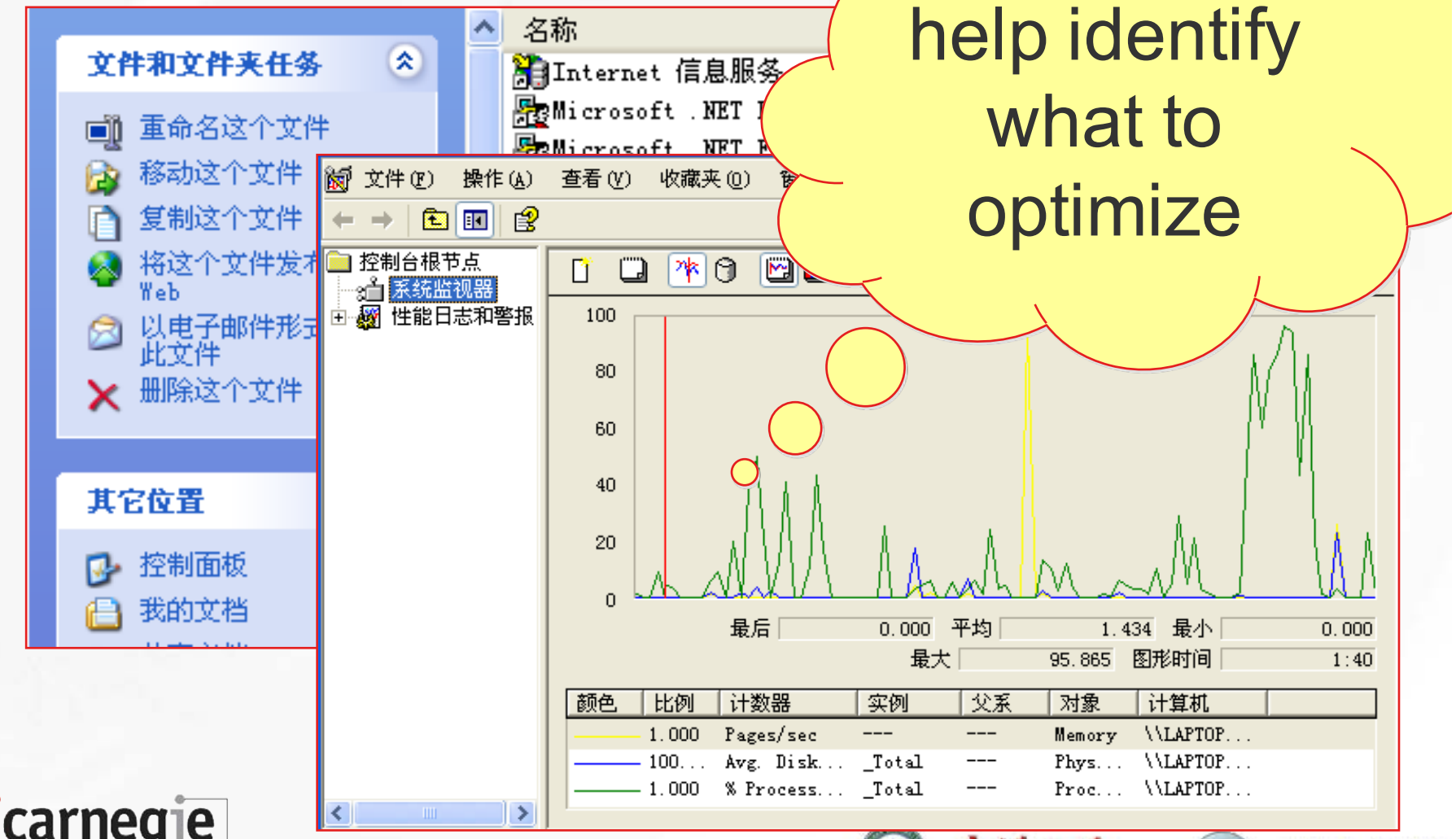
Lecture 3 Measurement and Profiling

Profiling [summary]:

- Program performance can be measured
  - timing the execution
  - statistical sampling (profiling).
- Timers to measure the run time
- Profiling to help identify what to optimize
- System resource monitors:
  - CPU time
  - memory usage
  - I/O behavior

Lecture 3 Measurement and Profiling

System resource monitors:



help identify what to optimize

Lecture 3 Hot Spots

What's a Hot Spot?

- The 80/20 Rule
  - 80% of the CPU time is spent in 20% of the program
  - These places where the computer spends most of its time are also called *hot spots*, *inner loops, and kernels*.

Lecture 3

Hot Spots

Amdahl's Law

- All programs have hot spots
- A little effort can have large payoffs.
- If only part of a program can be sped up with parallel processing, then the sequential part of the program would determine and limit the possible speed up.
- Knowing how to optimize is an important skill. Knowing when to stop optimizing is also important

Lecture 3 Hot Spots





An example for Hot Spots

Next Time…

Lecture 3 Next Time……



- Read:
  - Unit 4 (Go through it)
- Do:
  - Multiple-Choice Quizzes in Unit 4

In the Lab……

- Finish Excise 3 (Debugging Malloc Lab)
  - Get debugging_malloc.zip from ftp@iss
  - Define your own Malloc() & Free()
  - Define your own Structure for Payload

SSD 6

icarnegie

curriculum powered by Carnegie Mellon

Lecture 6 (2/2)

Performance Measurement and Improvement

武漢大學

国际软件学院

Lecture 6 Performance Measurement and Improvement

The Contents in SSD6 cover:

4.1 Measurement and Profiling

Where to start? Where to stop? How-to?

4.2 Hot Spots

Find the weak-knees!

4.3 Practical Hints

Take a tour…

Lecture 6 Hot Spots

Amdahl's Law

- All programs have hot spots
- A little effort can have large payoffs.
- If only part of a program can be sped up with parallel processing, then the sequential part of the program would determine and limit the possible speed up.
- Knowing how to optimize is an important skill. Knowing when to stop optimizing is also important

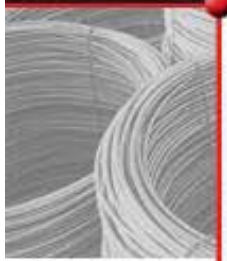
Lecture 6 Hot Spots

Amdahl's Law

$$T_{New} = (1-\alpha)T_{Old} + (\alpha T_{Old})/k$$

$$S = \frac{T_{Old}}{T_{New}} = \frac{1}{(1-\alpha)+\alpha/k}$$

Lecture 6 Hot Spots





Amdahl's Law

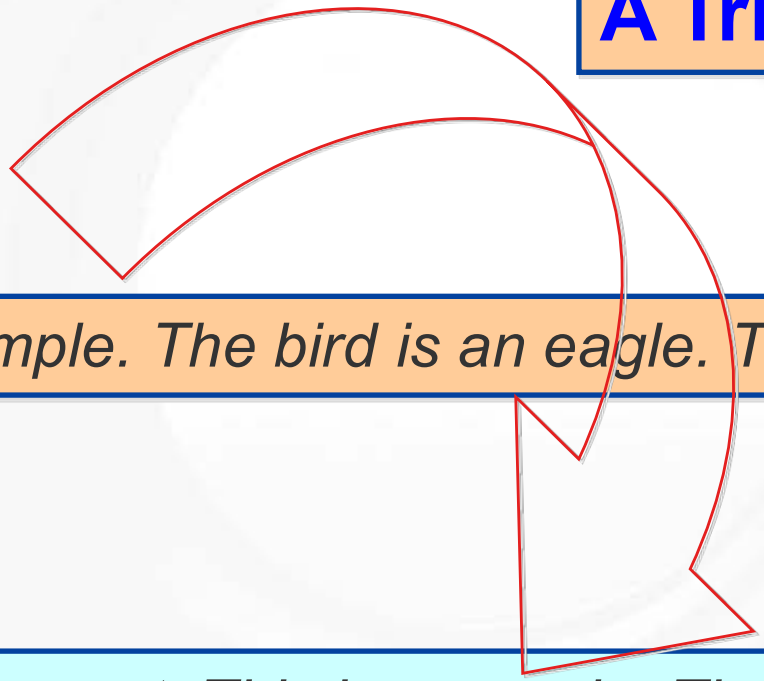
$$S = \frac{T_{Old}}{T_{New}} = \frac{1}{(1-\alpha)+\alpha/k}$$

$$S_{\infty} = \frac{1}{(1-\alpha)};\ k \to \infty$$

Lecture 6 Hot Spots

An example for Hot Spots

A Trigram Game



This is an example. The bird is an eagle. The boy is an expert.

The bird is an expert. This is an eagle. The boy is an example.

Lecture 6 Hot Spots

A Trigram Game



See:

Words.cpp
Words3.cpp

Lecture 6 Hot Spots

A Trigram Game

	B	C	D	E	F	G	H	I	J	K
1	Function	Times Called	Exclusive Time	Callee Exclusive Time	Inclusive Time	Callee Time	Suspended Time	Profiler Time	Func%	Func+child%
2	static void <Module>::main( )	1	0.004192	951.23658	4732.46844	4732.46385	0	0.000398	0.00%	100.00%
3	static void <Module>::generate(void, ir	1	0.384371	949.754105	4725.739688	4725.355035	0	0.000282	0.06%	99.84%
4	static void <Module>::countpairs(int32	9998	421.458386	631.672108	3152.368877	2730.597402	0	0.313089	66.25%	66.41%
5	static void <Module>::pick_nth(*, int8, *	9998	213.207295	315.888776	1570.155352	1356.638362	0	0.309695	33.51%	33.21%
6	static void <Module>::readwords( )	1	0.095483	1.476856	6.722488	6.626706	0	0.000298	0.02%	0.16%
7	static void <Module>::getword(int32)	8254	0.760527	1.127512	5.373768	4.356743	0	0.256499	0.12%	0.12%
8	static void <Module>::string_to_heap(*	8253	0.161802	0.25386	1.252938	0.838691	0	0.252445	0.03%	0.03%
9	void System.AppDomain::SetupDomain	1	0.024919	0.090077	0.098413	0.073145	0	0.000349	0.00%	0.01%
10	void System.AppDomain::SetupEusion9	1	0.056852	0.064624	0.072301	0.013876	0	0.001574	0.01%	0.01%

Lecture 6 Hot Spots

A Trigram Game

```
char *string_to_heap(char *s)
{
  for (int i = 0; i < wordcount; i++) {
      if (streql(s, words[i])) {
        return words[i];
  }
}
  char *h = (char *) malloc(strlen(s) + 1);
  if (!h) fatal("no more memory for words");
  strcpy(h, s);
  return h;
}
```

1

```
char *string_to_heap(char *s) {
  int slen = strlen(s)+1;
  char *h = (char *) malloc(slen);
  if (!h) fatal("no more memory for words");
  strcpy_s(h, slen, s);
  return h;
}
```

Lecture 6 Hot Spots

```
#define streql(s1, s2) (strcmp(s1, s2) == 0)
```

A Trigram Game

```
int countpairs(char *word1, char *word2) {
   int count = 0; int i;
   for (i = 0; i < wordcount; i++) {
      // find word1:
      if (streql(words[i], word1)) {
         // if found, check if it's followed by word2:
         if (streql(words[(i + 1) % wordcount], word2)) {
         // yes, we found (word1, word2) count++;
         }
      }
   }
   return count;
}
```

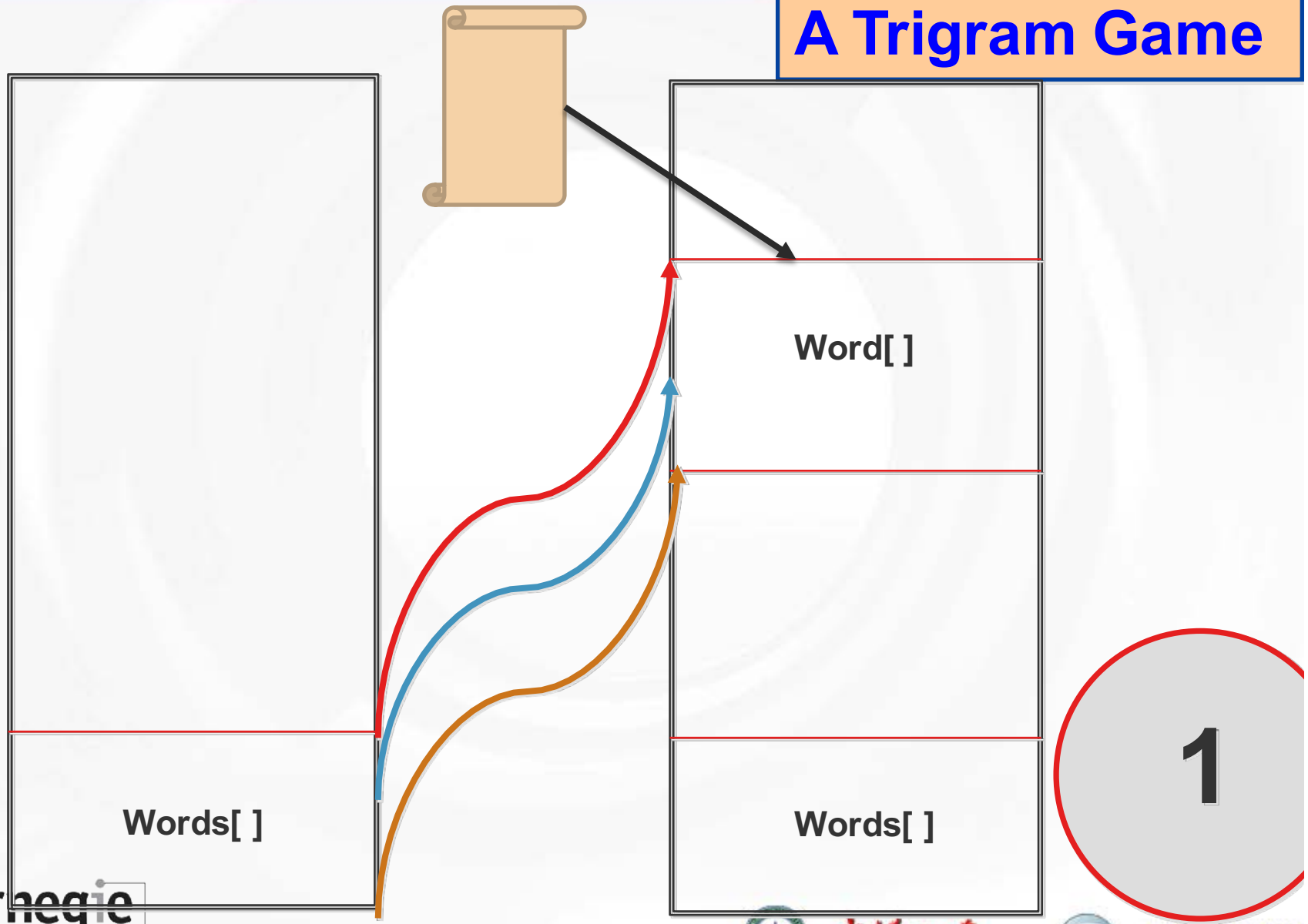
2

```
int countpairs(char *word1, char *word2) {
   int count = 0; int i;
   for (i = 0; i < wordcount; i++) {
      // find word1:
      if (words[i] == word1) {
         // if found, check if it's followed by wo
         if (streql(words[(i + 1) % wordcount], wo
         // yes, we found (word1, word2) count++;
```

Lecture 6

Hot Spots

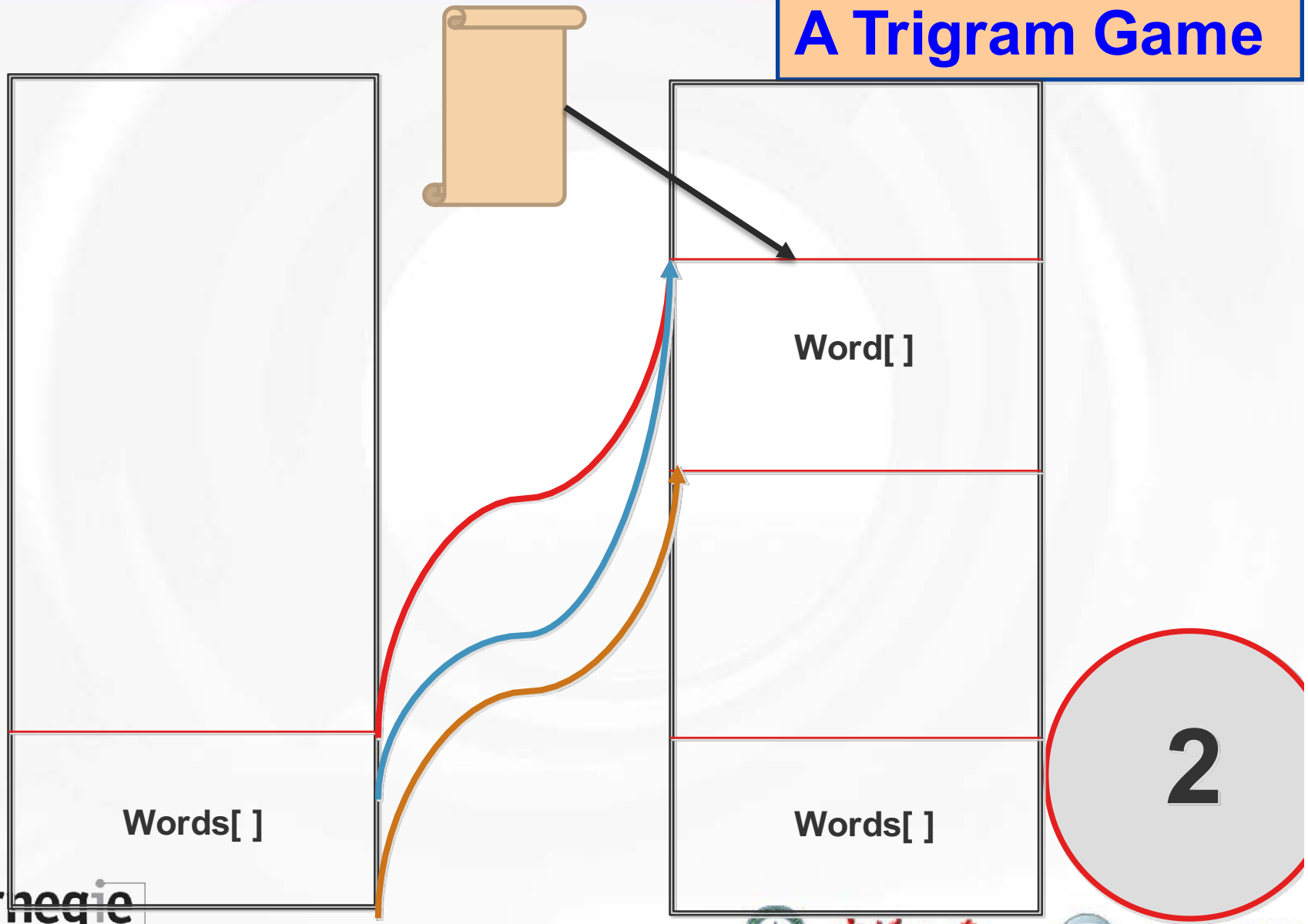
A Trigram Game



Lecture 6

Hot Spots

A Trigram Game



2

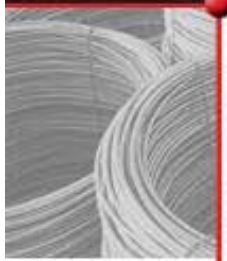
Lecture 6 Hot Spots

	B	C	D	E	F	G	H	I	J	K
1	Function	Times Called	Exclusive Time	Callee Exclusive	nclusive Time	Callee Time	Suspended Time	Profiler Time	Func%	Func +child%
2	static void <Module>::main( )	1	0.004192	951.23658	4732.46844	4732.46385	0	0.000398	0.00%	100.00%
3	static void <Module>::generate(void, ir	1	0.384371		4725.739688	4725.355035	0	0.000282	0.06%	99.84%
4	static void <Module>::countpairs(int32	9998	421.458386	631.672108	3152.368877	2730.597402	0	0.313089	66.25%	66.41%
5	static void <Module>::pick_nth(*, int8, *	9998	213.207295	315.888776	1570.155352	1356.638362	0	0.309695	33.51%	33.21%
6	static void <Module>::readwords( )	1	0.095483	1.476856	6.722488	6.626706	0	0.000298	0.02%	0.16%
7	static void <Module>::getword(int32)	8254	0.760527	1.127512	5.373768	4.356743	0	0.256499	0.12%	0.12%
8	static void <Module>::string_to_heap(*	8253	0.161802	0.25386	1.252938	0.838691	0	0.252445	0.03%	0.03%
9	void System.AppDomain::SetupDomain	1	0.024919	0.090077	0.098413	0.073145	0	0.000349	0.00%	0.01%
10	void System.AppDomain::SetupFusion	1	0.056852	0.064624	0.072301	0.013876	0	0.001574	0.01%	0.01%

1	Function	Times Called	Exclusive Time	Callee Exclusive	Inclusive Time	Callee Time	Suspended Time	Profiler Time	Func%	Func +child%
2	static void <Module>::main( )	1	0.003923	89.564498	411.379271	411.374964	0	0.000385	0.01%	100.00%
3	static void <Module>::readwords( )	1	0.093546		407.500631	407.406796	0	0.000289	0.15%	97.47%
4	static void <Module>::string_to_heap(	8253	60.626506	86.131939	402.416761	341.549836	0	0.240419	96.51%	96.17%
5	static void <Module>::generate(void,	1	0.296708	2.261382	3.872616	3.575621	0	0.000287	0.47%	2.52%
6	static void <Module>::getword(int32)	8254	0.700869	1.072232	4.990035	4.049166	0	0.24	1.12%	1.20%
7	static void <Module>::countpairs(int3	9998	0.751609	0.751609	1.068417	0	0	0.316809	1.20%	0.84%
8	static void <Module>::pick_nth(*, int8	9998	0.317123	0.317123	0.623271	0	0	0.306148	0.50%	0.35%
9	void System.AppDomain::SetupDomai	1	0.004213	0.030956	0.039072	0.034516	0	0.000343	0.01%	0.03%
10	void System.AppDomain::SetupFusior	1	0.018392	0.026332	0.033811	0.01384	0	0.001579	0.03%	0.03%

After optimized!

Lecture 6 Practical Hints





- *Some Expensive Operations in C*
  - *printf()*
  - *malloc()*
  - *new()*

Lecture 6

Practical Hints

- *Malloc() in Printf()*
  - *Printf*() causes *malloc* to be called in the usual implementation, adding an unexpected cost
  - String manipulation is expensive, whether it is formatting text as in *printf*, reading ASCII text and converting to numbers, or performing string comparisons
  - Modern computers are designed to access memory a word at a time. Accessing a character is no faster than accessing a word, and may be slower
  - String operations usually operate one character at a time, *i.e* strlen(), has to examine every character, one at a time

Lecture 6

Practical Hints

- On *Malloc():*
  - use static or local variables to avoid allocating memory on the heap;
  - keep a list of objects that need to be allocated often ➔ explicit free-list
  - Then, allocation is just a matter of removing an object from the list, and freeing simply inserts the object on the list

Lecture 6

Practical Hints

```
void main()
{
    [redacted]
    cout << "Allocated: ";
    for (int i = 0; i < 10; i++) {
        a[i] = Myobject::alloc();
        cout << a[i] << " ";
    }
    cout << endl;
    // free the even-numbered objects:
    cout << "Freed: ";
    for (i = 0; i < 5; i++) {
        cout << a[i * 2] << " ";
        a[i * 2]->free();
    }
    cout << endl;
    // reallocate the even-numbered objects:
    cout << "Allocated: ";
    for (i = 0; i < 5; i[redacted]
        a[i * 2] = Myobject::alloc();
        cout << a[i * 2] << " ";
    }
    cout << endl;
}
```

Lecture 6

Practical Hints



```
#include <iostream>
using namespace std;

class Myobject {
public:
    static Myobject *alloc();
    void free();

    Myobject *next; // used for allocation
    static Myobject *freelist;

    long x; // your member variables go here,
    long y; // etc.
};

// declare and initialize empty free list:
Myobject *Myobject::freelist = NULL;
```

Lecture 6

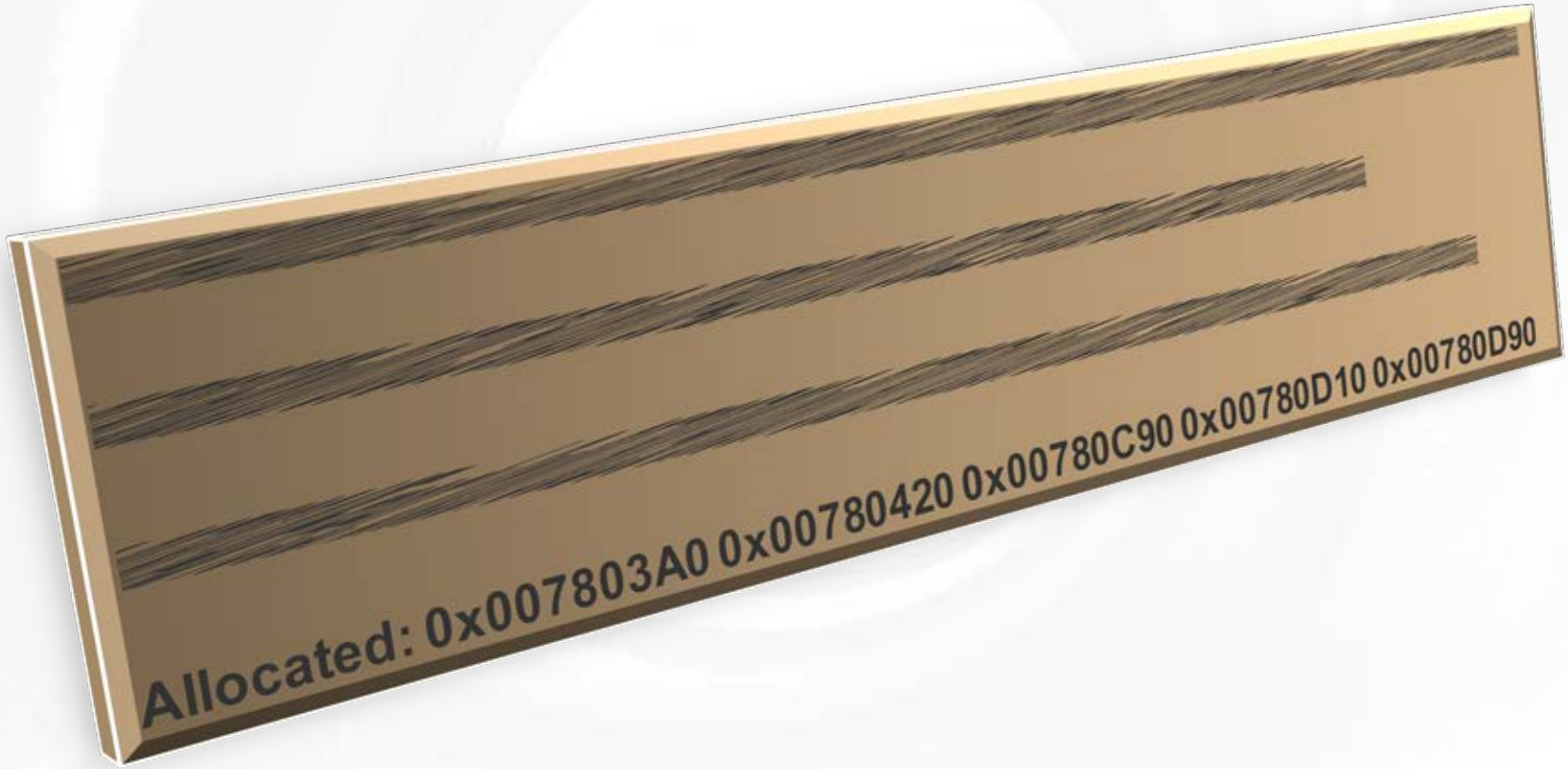
Practical Hints

```
Myobject *Myobject::alloc()
{
        Myobject *p;
        // fast allocation: pop object from freelist
        if (freelist
                p = freelist;
                = -
        } else {
                = Myobject;
        }
        return p;
}

void Myobject::free()
{
        this- freelist;
        freelist = this;
}
```

Lecture 6 Practical Hints

- Output:



Lecture 6 Practical Hints {more}

■ Fast Allocation and Free

Use a large block of memory from which smaller chunks are allocated to compute some result. After the result is obtained, the entire block is freed. This is fast because:

- Memory is allocated simply by incrementing the "free" pointer by the number of bytes you need to allocate.
- There is no need to free each allocated object; you free all objects at once by freeing the entire pool at once.

An Allocator!

About Excise 4 ……

- Excise 4 (Understanding Program Performance)
  - Get substitution.c and data.zip from ftp@iss
  - Goto support.microsoft.com, search for an article with *Article ID Q224382*
  - Optimize code to get better performance
  - Measure performance by *profiling*