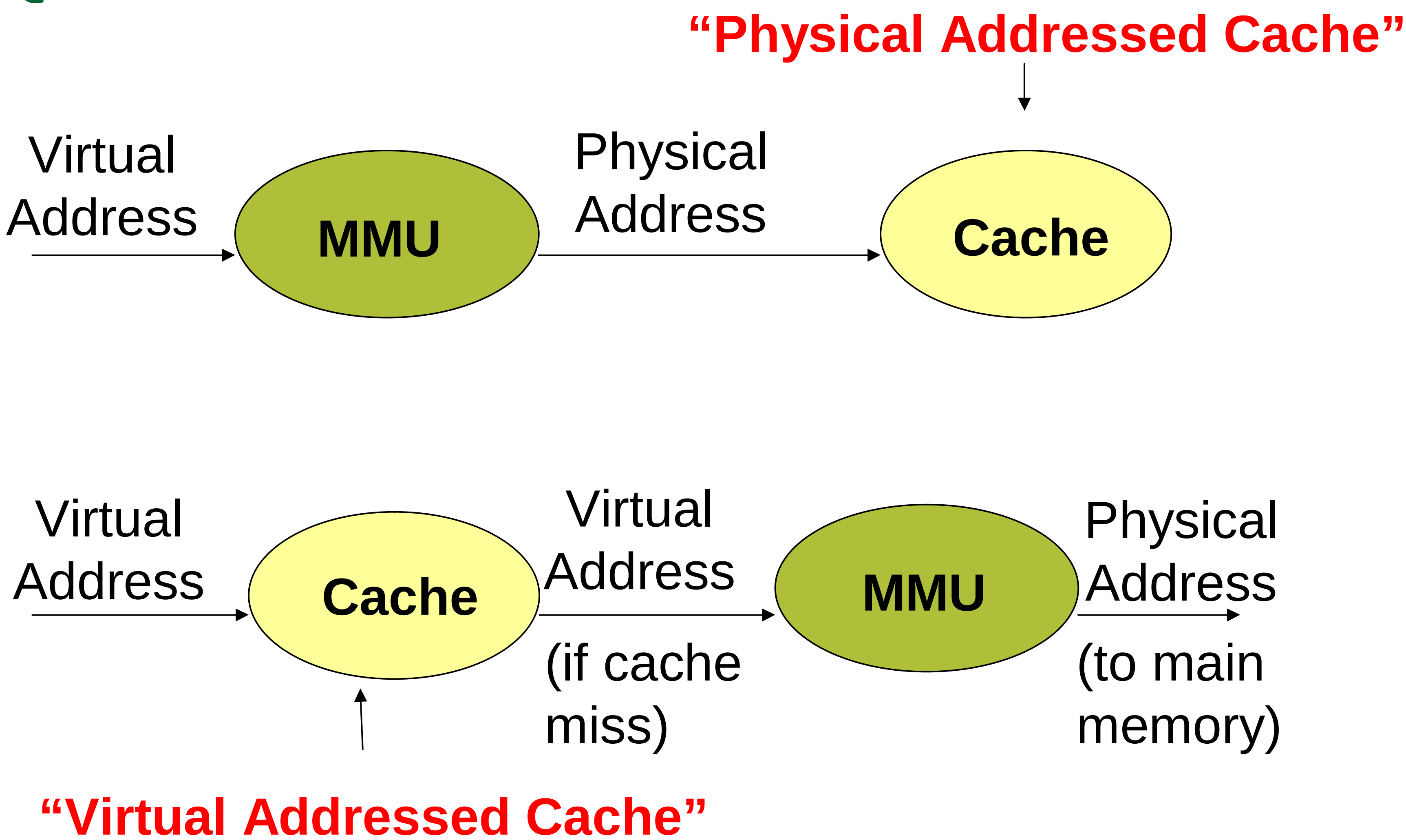
High Performance Computing
Lecture 32

Matthew Jacob
Indian Institute of Science

Reality Check

- Question 1: Are real caches built to work on virtual addresses or physical addresses?
- Question 2: Do modern processors use pipelining of the kind that we studied?

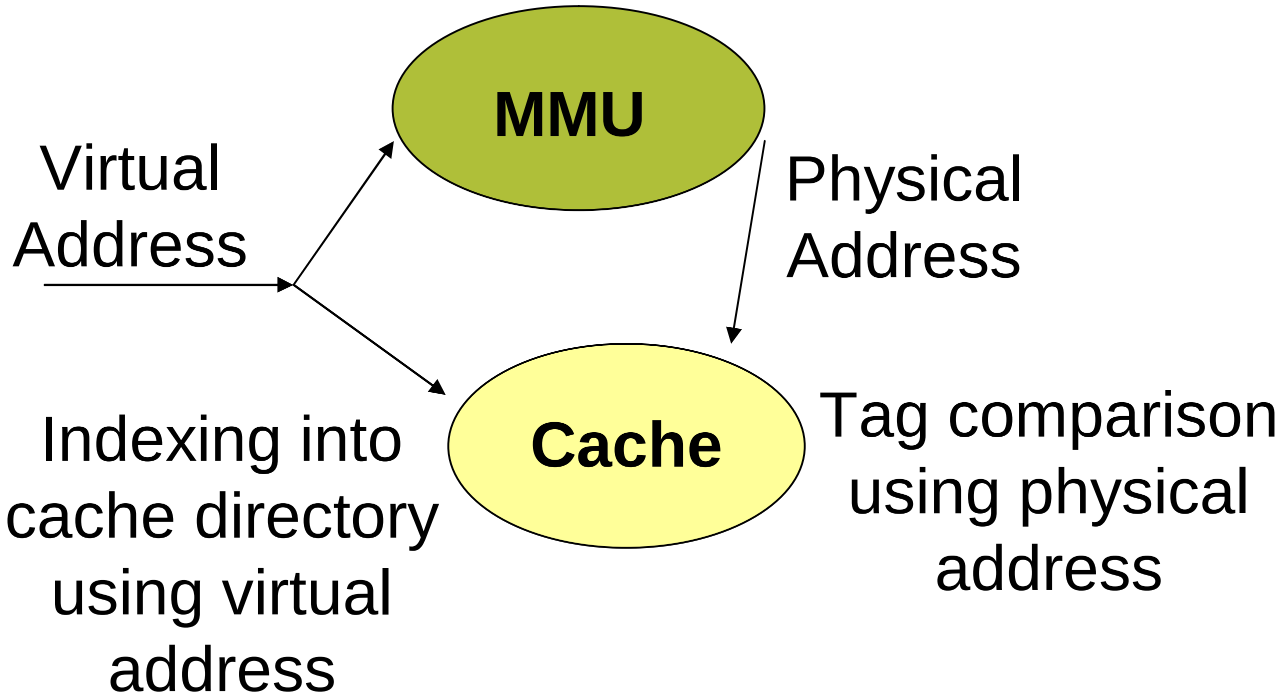
Q1: Caches and Address Translation



Which is less preferable?

- Physical addressed cache
  - Hit time higher (cache access after translation)
- Virtual addressed cache
  - Data/instruction of different processes with same virtual address in cache at the same time …
    - Flush cache on context switch, or
    - Include Process id as part of each cache directory entry
  - Synonyms
    - Virtual addresses that translate to same physical address
    - More than one copy of a block in cache …

Another possibility: Overlapped operation



Virtual indexed physical tagged cache

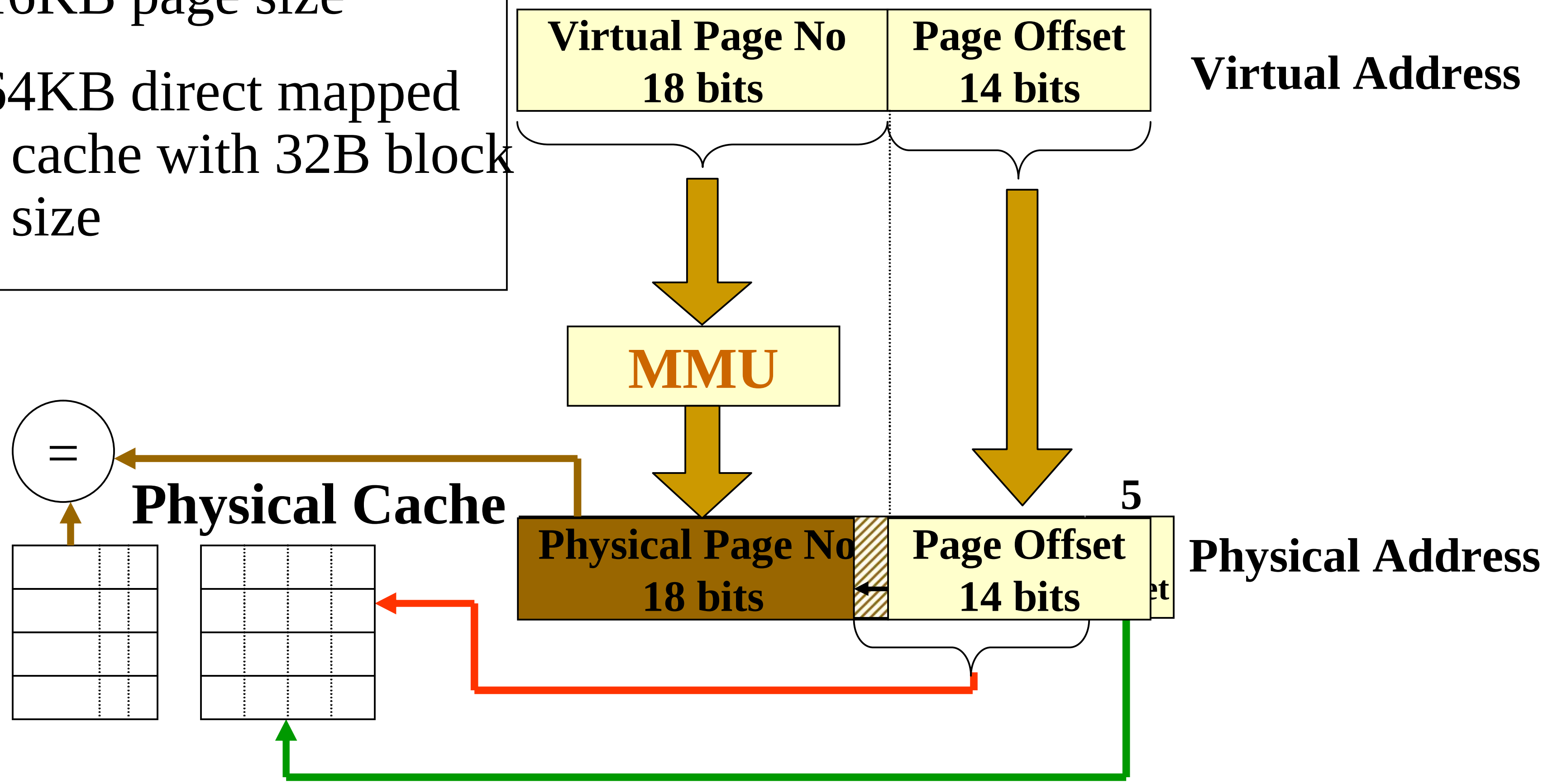
Addresses and Caches

- `Physical Addressed Cache'
  Physical Indexed Physical Tagged
- `Virtual Addressed Cache'
  Virtual Indexed Virtual Tagged
- Overlapped cache indexing and translation
  Virtual Indexed Physical Tagged

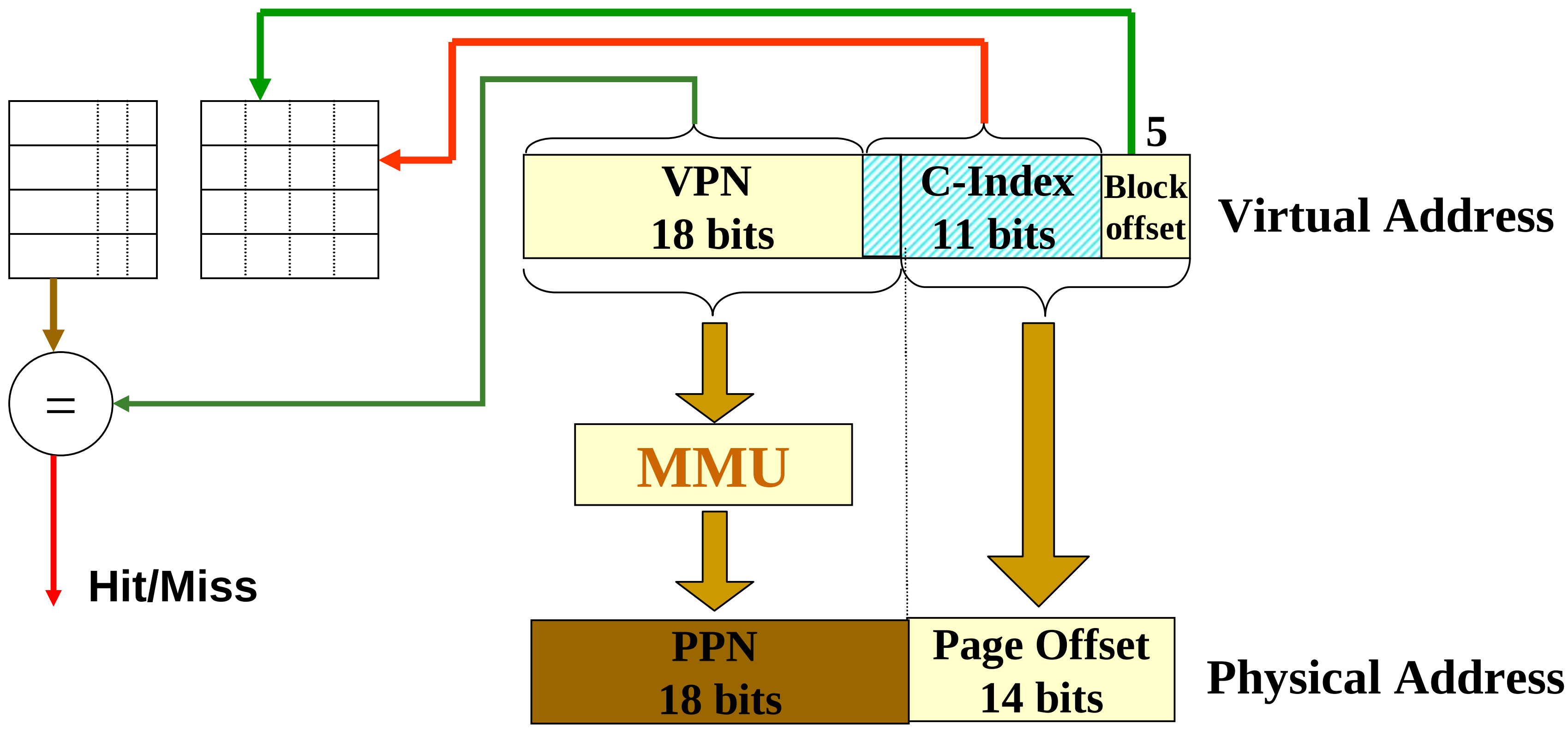
Physical Indexed Physical Tagged Cache

16KB page size

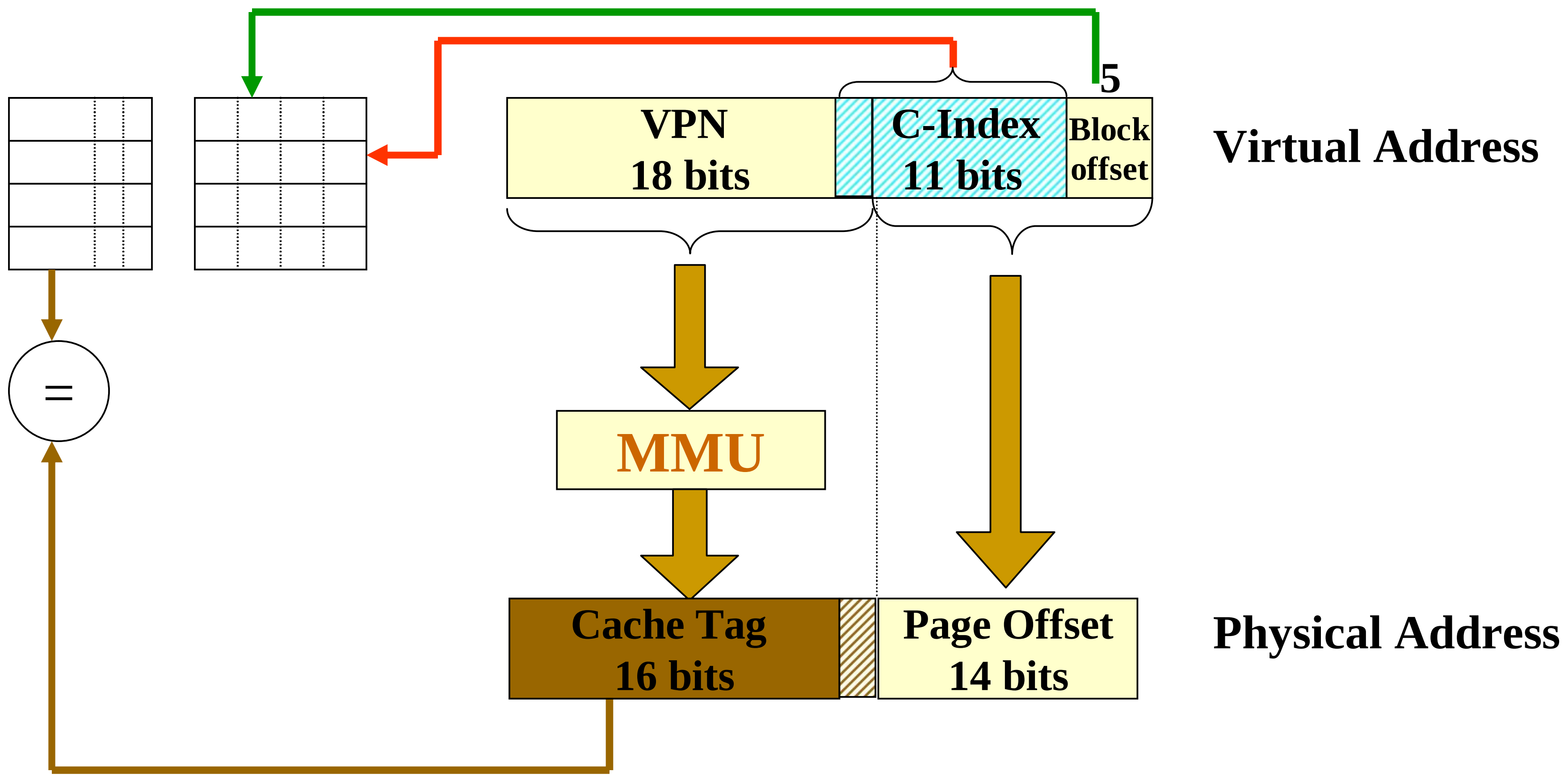
64KB direct mapped cache with 32B block size



Virtual Index Virtual Tagged Cache



Virtual Index Physical Tagged Cache



Reality Check

- Question 1: Are real caches built to work on virtual addresses or physical addresses?
- Question 2: Do modern processors use pipelining of the kind that we studied?

Q2: High Performance Pipelined Processors

- Pipelining
  - Overlaps execution of consecutive instructions
  - Performance of processor improves
- Current processors use more aggressive techniques for more performance
- Some exploit Instruction Level Parallelism - often, many consecutive instructions are independent of each other and can be executed in parallel (at the same time)

Instruction Level Parallelism Processors

- Challenge: identifying which instructions are independent
- Approach 1: build processor hardware to analyze and keep track of dependences
  - Superscalar processors

Instruction Level Parallelism Processors

- Challenge: identifying which instructions are independent
- Approach 1: build processor hardware to analyze and keep track of dependences
- Approach 2: compiler does analysis and packs suitable instructions together for parallel execution by processor
  - VLIW (very long instruction word) processors

Agenda

1. Program execution: Compilation, Object files, Function call and return, Address space, Data & its representation (4)
2. Computer organization: Memory, Registers, Instruction set architecture, Instruction processing (6)
3. Virtual memory: Address translation, Paging (4)
4. Operating system: Processes, System calls, Process management (6)
5. Pipelined processors: Structural, data and control hazards, impact on programming (4)
6. Cache memory: Organization, impact on programming (5)
7. Program profiling (2)
8. File systems: Disk management, Name management, Protection (4)
9. Parallel programming: Inter-process communication, Synchronization, Mutual exclusion, Parallel architecture, Programming with message passing using MPI (5)

TIMING AND PROFILING

- Profiling: Identifying the important parts of your program
  - Concentrate your optimization efforts on those parts
- Timing: Determining program execution time

Timing

Timing: measuring the time spent in specific parts of your program

- Examples of `parts’: Functions, loops, …
- Recall: Different kinds of time that can be measured (real/wallclock/elapsed vs virtual/CPU)

1. Decide
   - which time you are interested in measuring
   - at what granularity
2. Find out what mechanisms are available and their granularity of measurement

time command

Usage: % time a.out

Reports Real/Elapsed/Wallclock time, CPU time in user mode, CPU time in system mode

Example: % time ls

    0.00user 0.002sys 0:0.003elapsed

Example: % time man csh

    0.268user 0.032sys 0:15.486elapsed

gettimeofday( )

Reports real time that has elapsed since 00:00 GMT 1 January 1970 (The Epoch)

```
#include <sys/time.h>

int gettimeofday(struct timeval *tv, struct timezone *tz);

struct timeval {
    long tv_sec;        /* seconds */
    long tv_usec;       /* microseconds */
};
```

Usage: Insert calls to gettimeofday in your C program

Using gettimeofday( )

Your C program

```
struct timeval before, after;

gettimeofday(&before);
          /
          / region of program you want to time
          /
gettimeofday(&after);

printf (“%d\n”, after.tv_sec – before.tv_sec);
```

High resolution, real timers

- Most modern processors provide a hardware cycle counting mechanism
  1. A special purpose register that is incremented every clock cycle
  2. An instruction to read the value in that register
- Example: Intel® time stamp counter and `rdtsc` instruction

Profiling

- Profiler: A tool that helps you identify the `important' parts of your program to concentrate your optimization efforts
- Profile: a breakup (of execution time) across the different parts of the program
- Can be done by adding statements to your program (instrumentation) -- so that during execution, data is gathered, outputted and possibly processed later
- Automation: where a profiling tool adds those instructions into your program for you

Profiling Mechanisms

- Levels of Granularity typically supported
  - Function level
  - Statement level
  - Basic block level: A basic block is a sequence of contiguous instructions in a program with a single entry point (the first instruction in the basic block) and a single exit point (the last instruction in the basic block)
- Two examples of profile data
  - execution time
  - execution counts
- We will look at examples of profiling mechanisms at the function and basic block level