

# About the Slides for Introduction to Computer Systems

15-213: Introduction to Computer Systems  
0<sup>th</sup> Lecture, Sep. 1, 2015

Markus Püschel  
ETH Zurich

(with small contributions by Dave O'Hallaron)

# On the Design

- All slides are in Powerpoint 2007 (mix of PC and Mac versions)
- Probably could be edited using Powerpoint 2003 plus
  - File format plugin
  - Calibri font
  - I would still recommend to use 2007 for editing
- Design is suitable for printing out slides
  - Only light colors, in particular for boxes
- Some slides have covered areas (that disappear later) suitable for quizzing in class
- The design follows the Small Guide to Giving Presentations
- Next slides: Color/format conventions

*Style for title slides*

# System-Level I/O

15-213/18-243: Introduction to Computer Systems  
14<sup>th</sup> Lecture, Oct. 12, 2010

## Instructors:

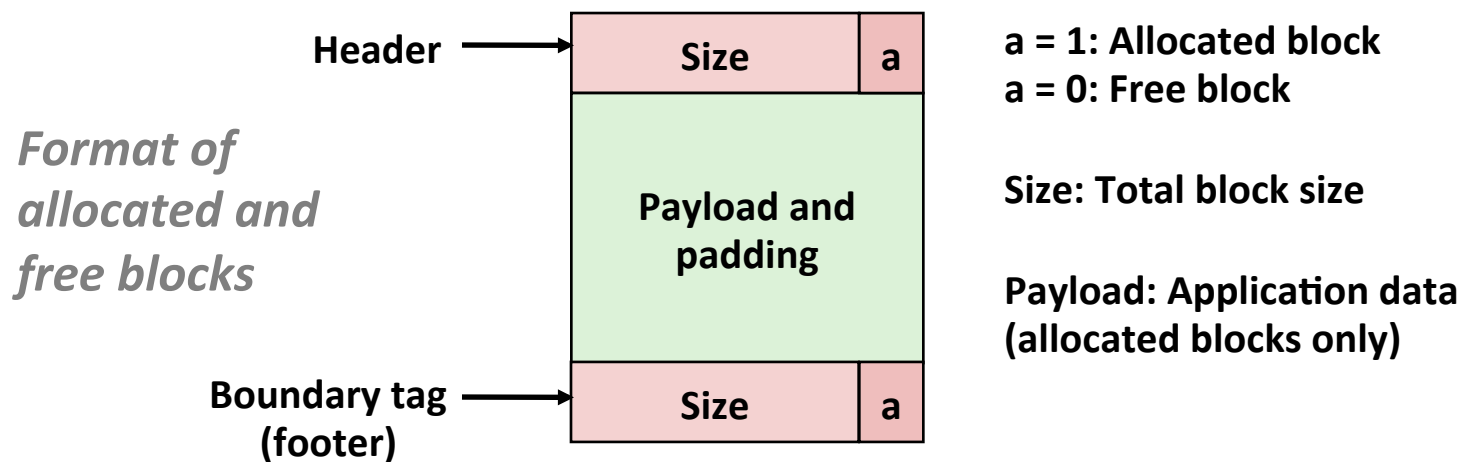
Randy Bryant and Dave O'Hallaron

# Today

- Unix I/O
- RIO (robust I/O) package
- **Metadata, sharing, and redirection**
- Standard I/O
- Conclusions and examples

# Style for Figure Labels

- **Capitalize only the first word in each figure label**
  - E.g., “Payload and padding”, not “Payload and Padding”, or “payload and padding”
  - This is the same style convention that we used in CS:APP2e.



# Style for Code

```
/*
 * hello.c - Pthreads "hello, world" program
 */
#include "csapp.h"

void *thread(void *vargp);

int main() {
    pthread_t tid;

    Pthread_create(&tid, NULL, thread, NULL);
    Pthread_join(tid, NULL);
    exit(0);
}

/* thread routine */
void *thread(void *vargp) {
    printf("Hello, world!\n");
    return NULL;
}
```

# Style for Code and Alternative Code

## C Code

```
int fact_do(int x)
{
    int result = 1;
    do {
        result *= x;
        x = x-1;
    } while (x > 1);

    return result;
}
```

## Goto Version

```
int fact_goto(int x)
{
    int result = 1;
loop:
    result *= x;
    x = x-1;
    if (x > 1)
        goto loop;
    return result;
}
```

# Style for Assembly Code: Version I

```
int absdiff(int x, int y)
{
    int result;
    if (x > y) {
        result = x-y;
    } else {
        result = y-x;
    }
    return result;
}
```

```
absdiff:
    pushl    %ebp
    movl     %esp, %ebp
    movl     8(%ebp), %edx
    movl     12(%ebp), %eax
    cmpl     %eax, %edx
    jle      .L7
    subl     %eax, %edx
    movl     %edx, %eax
.L8:
    leave
    ret
.L7:
    subl     %edx, %eax
    jmp      .L8
```

# Style for Assembly Code: Version II

```
struct rec {  
    int i;  
    int a[3];  
    int *p;  
};
```

```
void  
set_p(struct rec *r)  
{  
    r->p =  
        &r->a[r->i];  
}
```

```
# %edx = r  
movl (%edx),%ecx      # r->i  
leal 0(,%ecx,4),%eax  # 4*(r->i)  
leal 4(%edx,%eax),%eax # r+4+4*(r->i)  
movl %eax,16(%edx)    # Update r->p
```

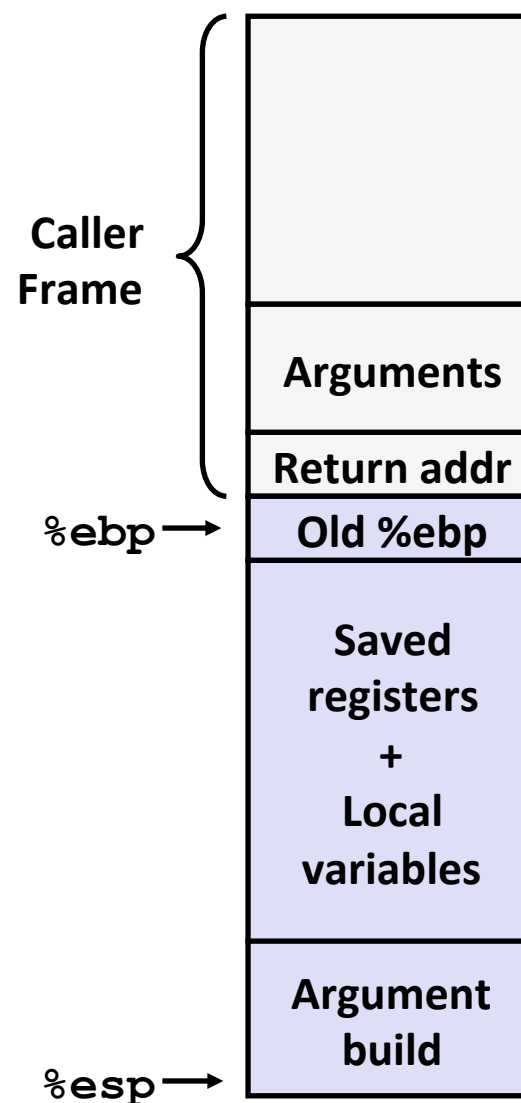
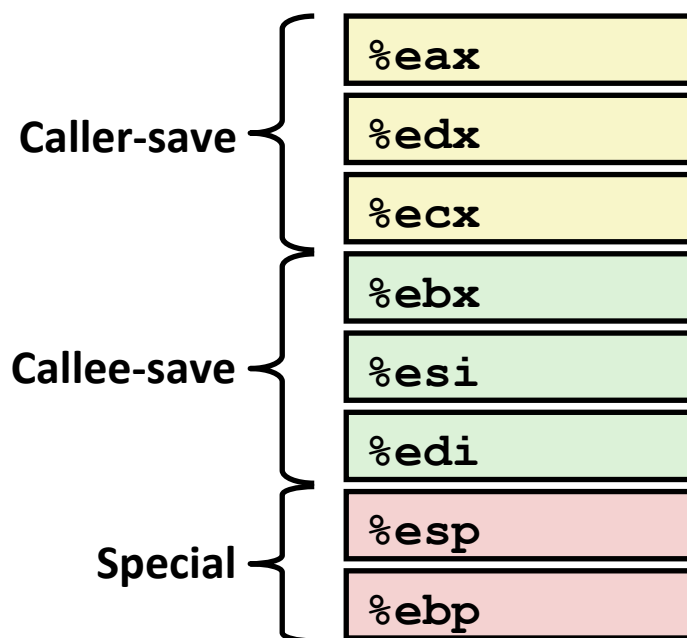
# Linux Command Prompt

```
linux> ./badcnt  
BOOM! cnt=198841183
```

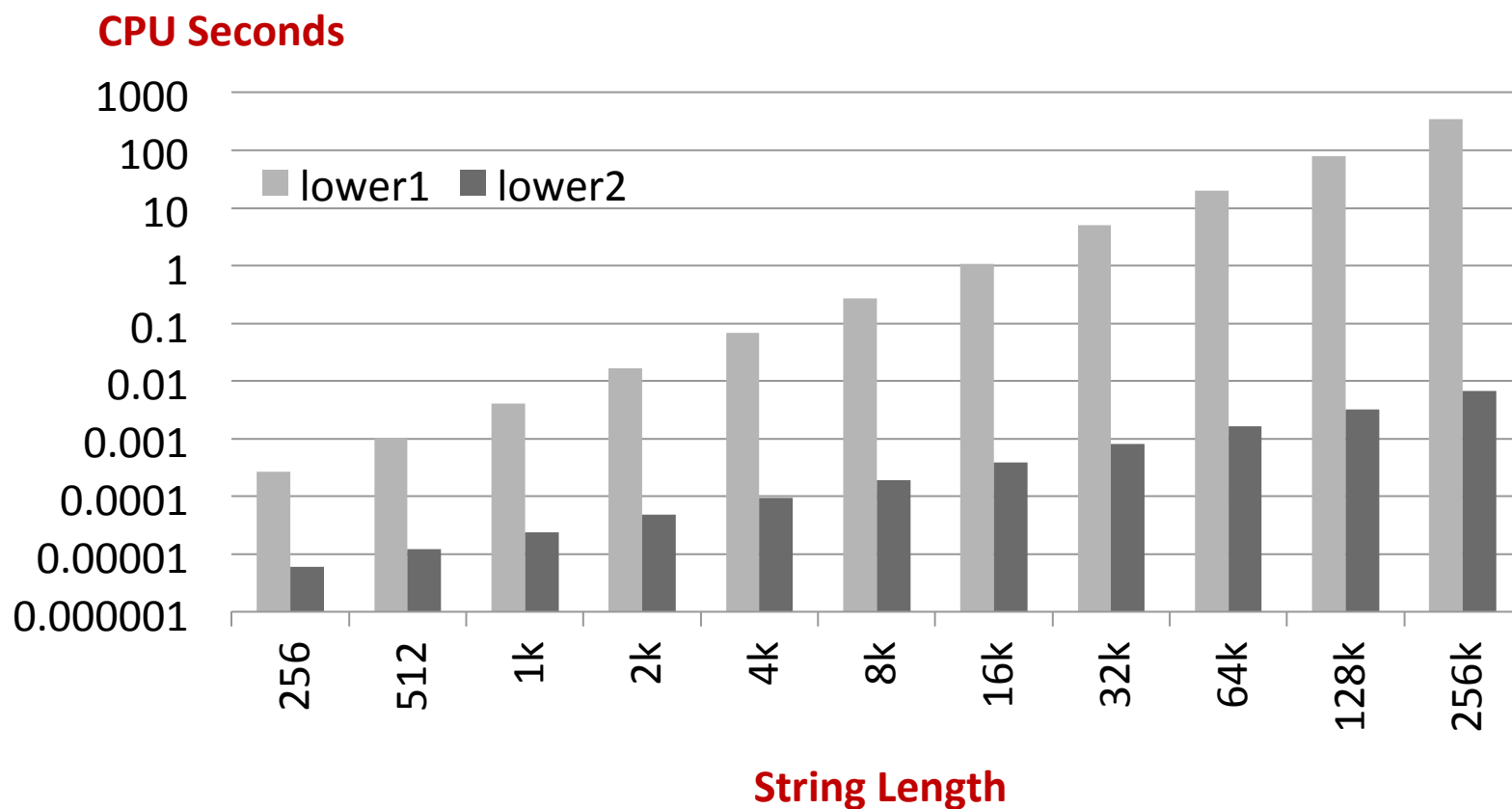
```
linux> ./badcnt  
BOOM! cnt=198261801
```

```
linux> ./badcnt  
BOOM! cnt=198269672
```

# Stack and Registers



# Bar Plot



# Tables

## Cycles per element (or per mult)

Machine	Nocona	Core 2
rfact	15.5	6.0
fact	10.0	3.0

Method	Int (add/mult)		Float (add/mult)	
combine4	2.2	10.0	5.0	7.0
unroll2	1.5	10.0	5.0	7.0
unroll2-ra	1.56	5.0	2.75	3.62
bound	1.0	1.0	2.0	2.0

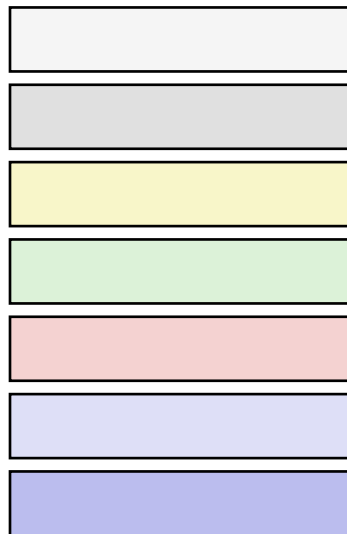
## ■ Some instructions take > 1 cycle, but can be pipelined

<i>Instruction</i>	<i>Latency</i>	<i>Cycles/Issue</i>
Load / Store	5	1
Integer Multiply	10	1
<b>Integer/Long Divide</b>	<b>36/106</b>	<b>36/106</b>
Single/Double FP Multiply	7	2
Single/Double FP Add	5	2
<b>Single/Double FP Divide</b>	<b>32/46</b>	<b>32/46</b>

# Color Palette

## ■ Boxes/areas:

- Assembly, memory, ...
- Linux, memory, ...
- Code, ...
- Code, registers, ...
- Registers, ...
- Memory, ...
- Memory, ...



## ■ Occasionally I use darker versions of the colors above

## ■ Text:

- **Emphasizing** something in the text
- **Comments** inside yellow code boxes