Course 2: Hardware

Kristaps Dzonsons

05 December, 2011

Course site: *http://kristaps.bsd.lv/minicourse_12_2011*

Environment

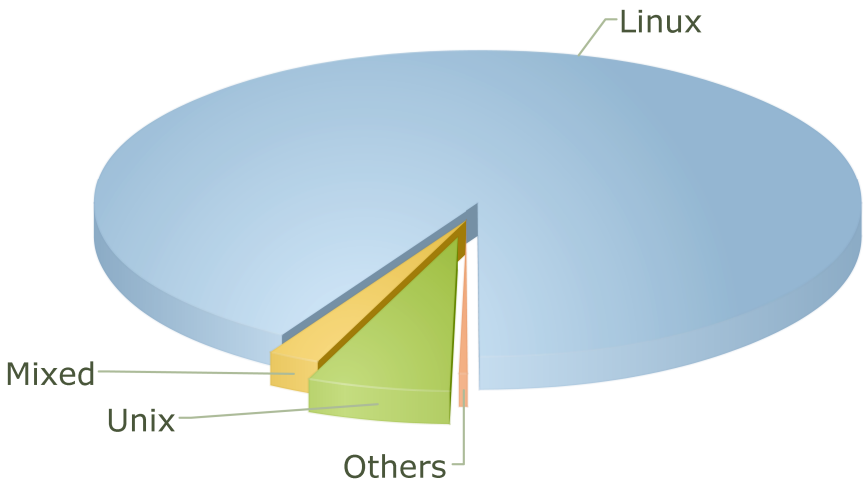
Most high-performance computing environments consist of:

- **UNIX** an operating system popular in non-desktop environments
- **C** a low-level, minimal programming language
- **hardware** purpose-built, high-performance hardware

In this lecture, we focus on *hardware*.

Review: Why UNIX?

Top500 (99.8%):



Sweden (100%):

Lindgren	Linux (Cray)
Ekman	Linux (Dell)
Povel	Linux (Proto)
Zorn	Linux (NVIDIA)
Hebb	Linux (IBM)

Review: Why C?

1. supported by default on most (all?) UNIX systems
2. extremely good support for performance ("up", distributed computing; "down", mixing assembly)
3. many open and closed-source domain-specific supporting libraries for
4. bindings for most other modern programming languages
5. familiar syntax (most languages are based upon C)

Review: Case Study

```
for (i = 0; i < sz; i++) {
    for (j = 2; j < p[i] - 1; j++)
        if (0 == p[i] % j)
            break;
    if (j == p[i] - 1)
        printf("%d\n", p[i]);
}
```

Review: Case Study

```
% cc -W -Wall -o example0 example0.c
% objdump -S example0
        for (i = start; i < sz; i++) {
  400d1f:       8b 45 8c                mov    0xffffffffffffff8c(%rbp),%eax
  400d22:       89 45 ac                mov    %eax,0xffffffffffffffac(%rbp)
  400d25:       eb 7e                   jmp    400da5 <f+0xd5>
                for (j = 2; j < p[i] - 1; j++)
  400d27:       c7 45 a8 02 00 00 00    movl   $0x2,0xffffffffffffffa8(%rbp)
  400d2e:       eb 21                   jmp    400d51 <f+0x81>
                        if (0 == p[i] % j)
  400d30:       8b 45 ac                mov    0xffffffffffffffac(%rbp),%eax
  400d33:       48 98                   cltq
  400d35:       48 c1 e0 02             shl    $0x2,%rax
  400d39:       48 03 45 90             add    0xffffffffffffff90(%rbp),%rax
  400d3d:       8b 10                   mov    (%rax),%edx
  400d3f:       89 d0                   mov    %edx,%eax
  400d41:       c1 fa 1f                sar    $0x1f,%edx
  400d44:       f7 7d a8                idivl  0xffffffffffffffa8(%rbp)
  400d47:       89 d0                   mov    %edx,%eax
  400d49:       85 c0                   test   %eax,%eax
  400d4b:       74 1b                   je     400d68 <f+0x98>
  400d4d:       83 45 a8 01             addl   $0x1,0xffffffffffffffa8(%rbp)
  400d51:       8b 45 ac                mov    0xffffffffffffffac(%rbp),%eax
  400d54:       48 98                   cltq
  400d56:       48 c1 e0 02             shl    $0x2,%rax
  400d5a:       48 03 45 90             add    0xffffffffffffff90(%rbp),%rax
  400d5e:       8b 00                   mov    (%rax),%eax
  400d60:       83 e8 01                sub    $0x1,%eax
  400d63:       3b 45 a8                cmp    0xffffffffffffffa8(%rbp),%eax
  400d66:       7f c8                   jg     400d30 <f+0x60>
                                break;
```

Hardware: Introduction

Why do we want to understand hardware? Computation, in practise, involves:

execution
: the actual execution of instructions

memory
: where instructions and data are stored

Both of these occur on Real Machines.

Case Study: Source

```c
#include <stdio.h>
#include <stdlib.h>

int
main(void)
{
        int             **matrix;
        int               i, j;
        long long int     res;

        matrix = malloc(11000 * sizeof(int *));
        for (i = 0; i < 11000; i++)
                matrix[i] = malloc(11000 * sizeof(int));

        res = 0;
        for (i = 0; i < 11000; i++)
                for (j = 0; j < 11000; j++)
                        res += matrix[i][j];

        printf("%lld\n", res);
        return(EXIT_SUCCESS);
}
```

Case Study: Source

```
#include <stdio.h>
#include <stdlib.h>

int
main(void)
{
        int             **matrix;
        int               i, j;
        long long int     res;

        matrix = malloc(11000 * sizeof(int *));
        for (i = 0; i < 11000; i++)
                matrix[i] = malloc(11000 * sizeof(int));

        res = 0;
        for (i = 0; i < 11000; i++)
                for (j = 0; j < 11000; j++)
                        res += matrix[j][i];

        printf("%lld\n", res);
        return(EXIT_SUCCESS);
}
```

Case Study: Difference

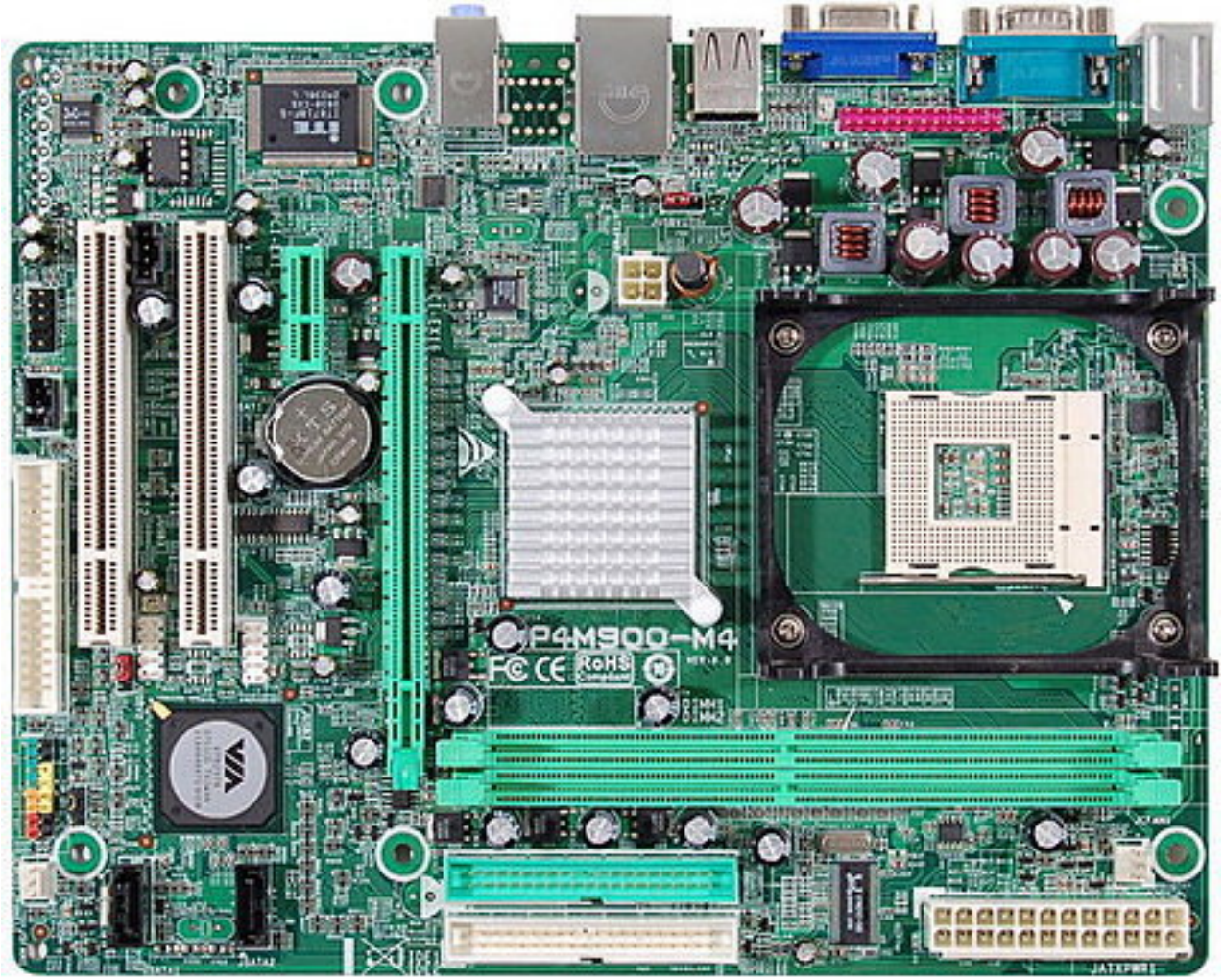
```
% mkdir kristaps ; cd kristaps
% ftp http://kristaps.bsd.lv/minicourse_12_2011/example1.c .
% ftp http://kristaps.bsd.lv/minicourse_12_2011/example2.c .
% diff -u example1.c example2.c
--- example1.c  Sun Dec  4 17:37:37 2011
+++ example2.c  Sun Dec  4 17:37:29 2011
@@ -15,7 +15,7 @@
        res = 0;
        for (i = 0; i < 11000; i++)
                for (j = 0; j < 11000; j++)
-                       res *= matrix[i][j];
+                       res *= matrix[j][i];

        return(EXIT_SUCCESS);
 }
```

Case Study: Benchmark

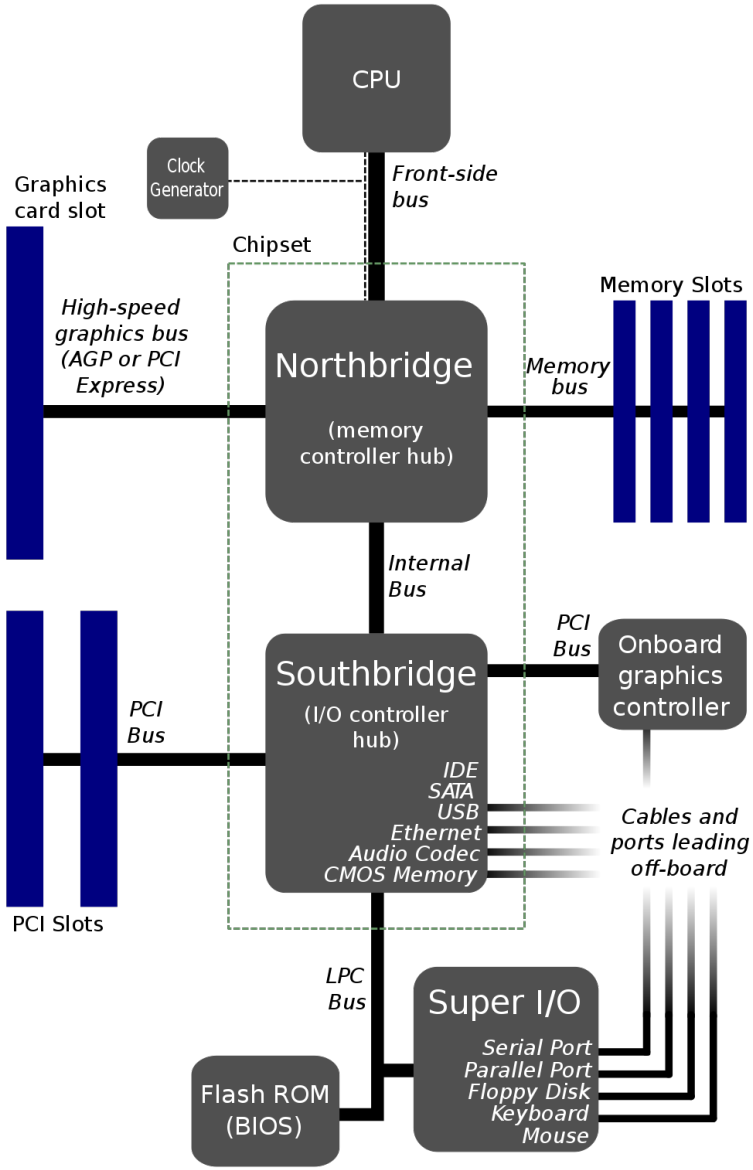
```
% mkdir kristaps ; cd kristaps
% ftp http://kristaps.bsd.lv/minicourse_12_2011/example1.c .
% ftp http://kristaps.bsd.lv/minicourse_12_2011/example2.c .
% cc -W -Wall -O2 -o example1 example1.c
% cc -W -Wall -O2 -o example2 example2.c
% time ./example1
    0m0.77s real
    0m0.28s user
    0m0.30s system
% time ./example2
    0m4.33s real
    0m3.66s user
    0m0.40s system
```

Hardware: Internals

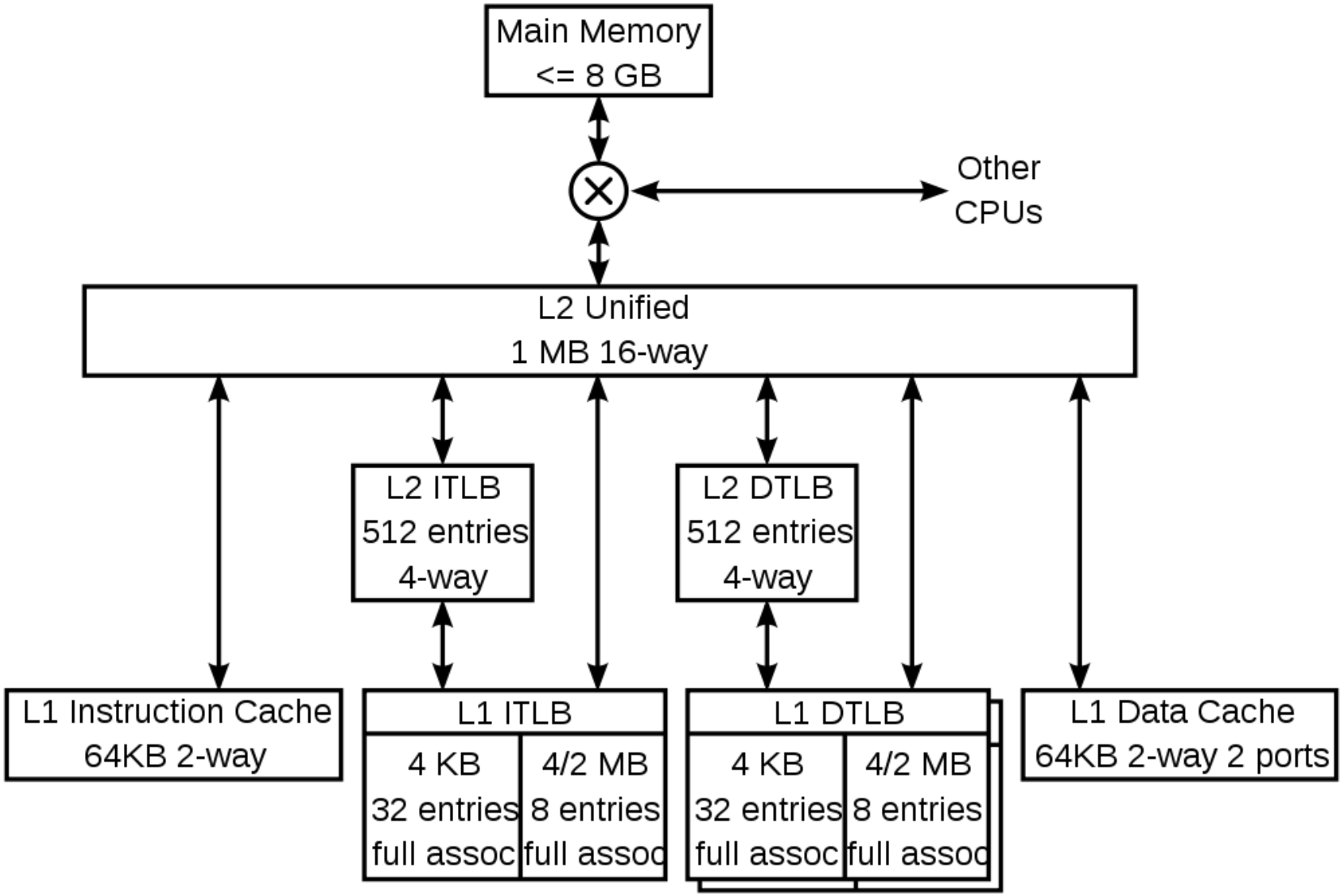
Hardware: Simple Motherboard



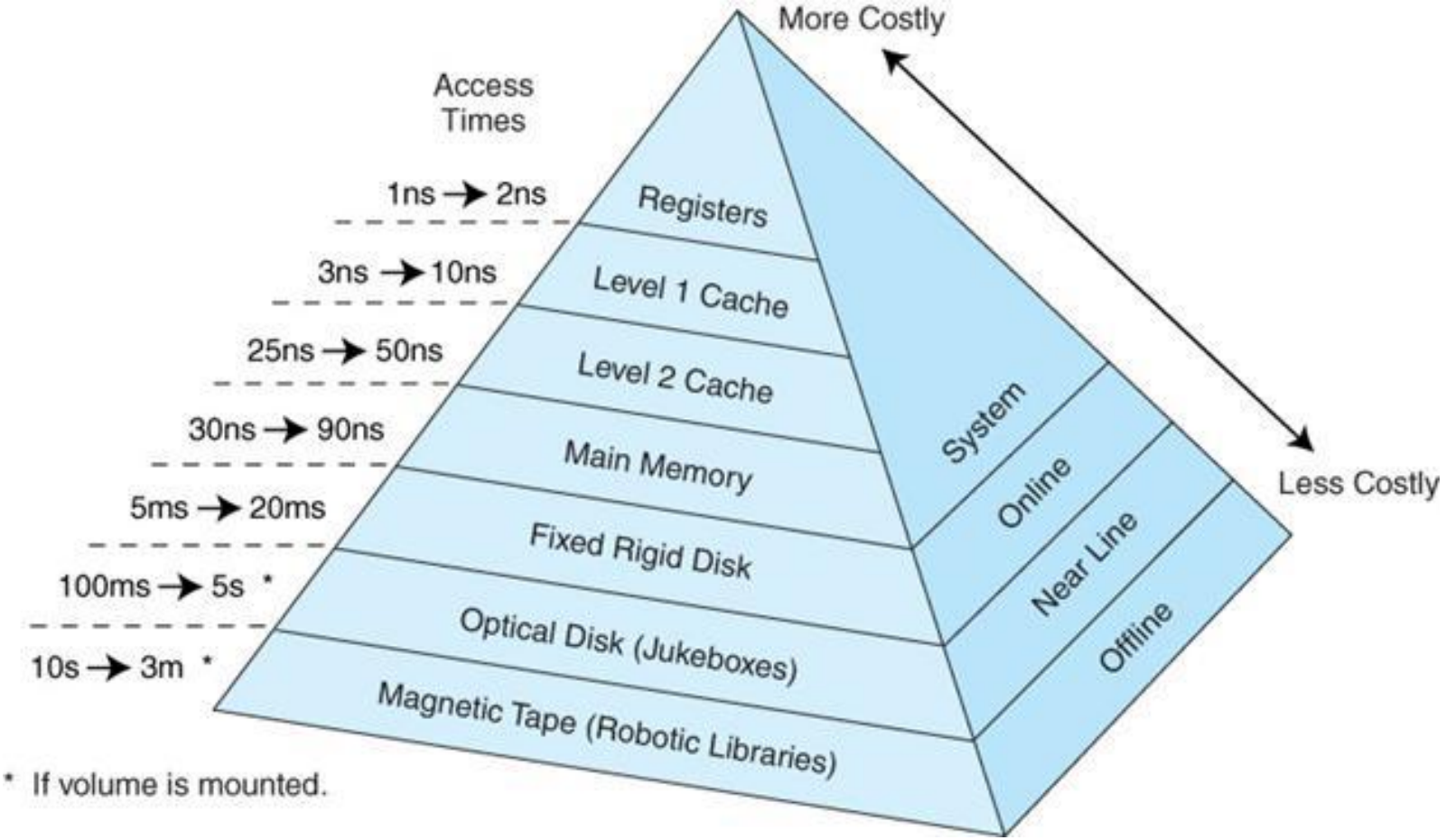
Hardware: Simple Motherboard



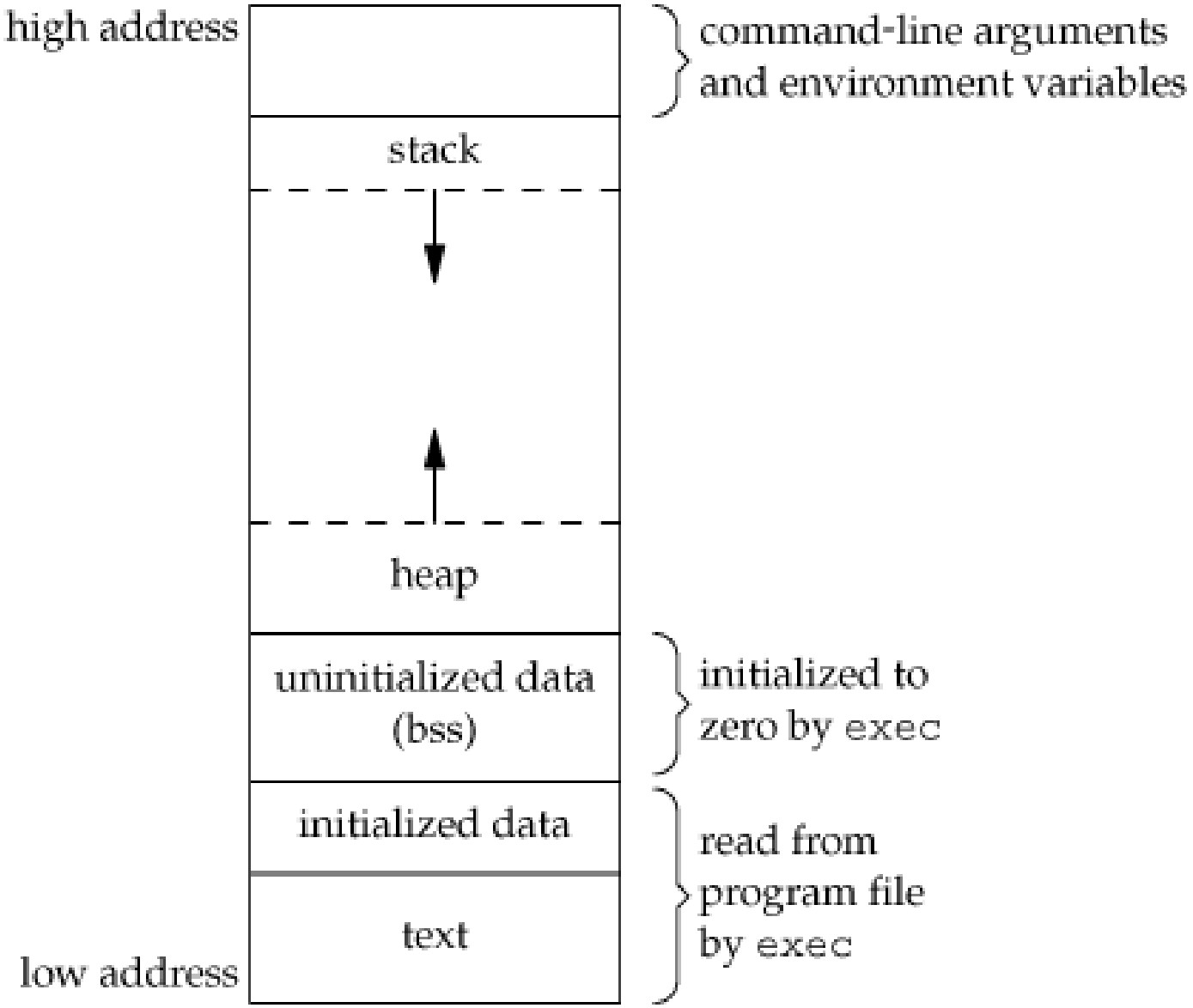
Hardware: Simple CPU



Memory Hierarchy



Process Memory



Case Study: Revisited

```
% mkdir kristaps ; cd kristaps
% ftp http://kristaps.bsd.lv/minicourse_12_2011/example1.c .
% ftp http://kristaps.bsd.lv/minicourse_12_2011/example2.c .
% diff -u example1.c example2.c
--- example1.c  Sun Dec  4 17:37:37 2011
+++ example2.c  Sun Dec  4 17:37:29 2011
@@ -15,7 +15,7 @@
        res = 0;
        for (i = 0; i < 11000; i++)
                for (j = 0; j < 11000; j++)
-                       res *= matrix[i][j];
+                       res *= matrix[j][i];

        return(EXIT_SUCCESS);
 }
```

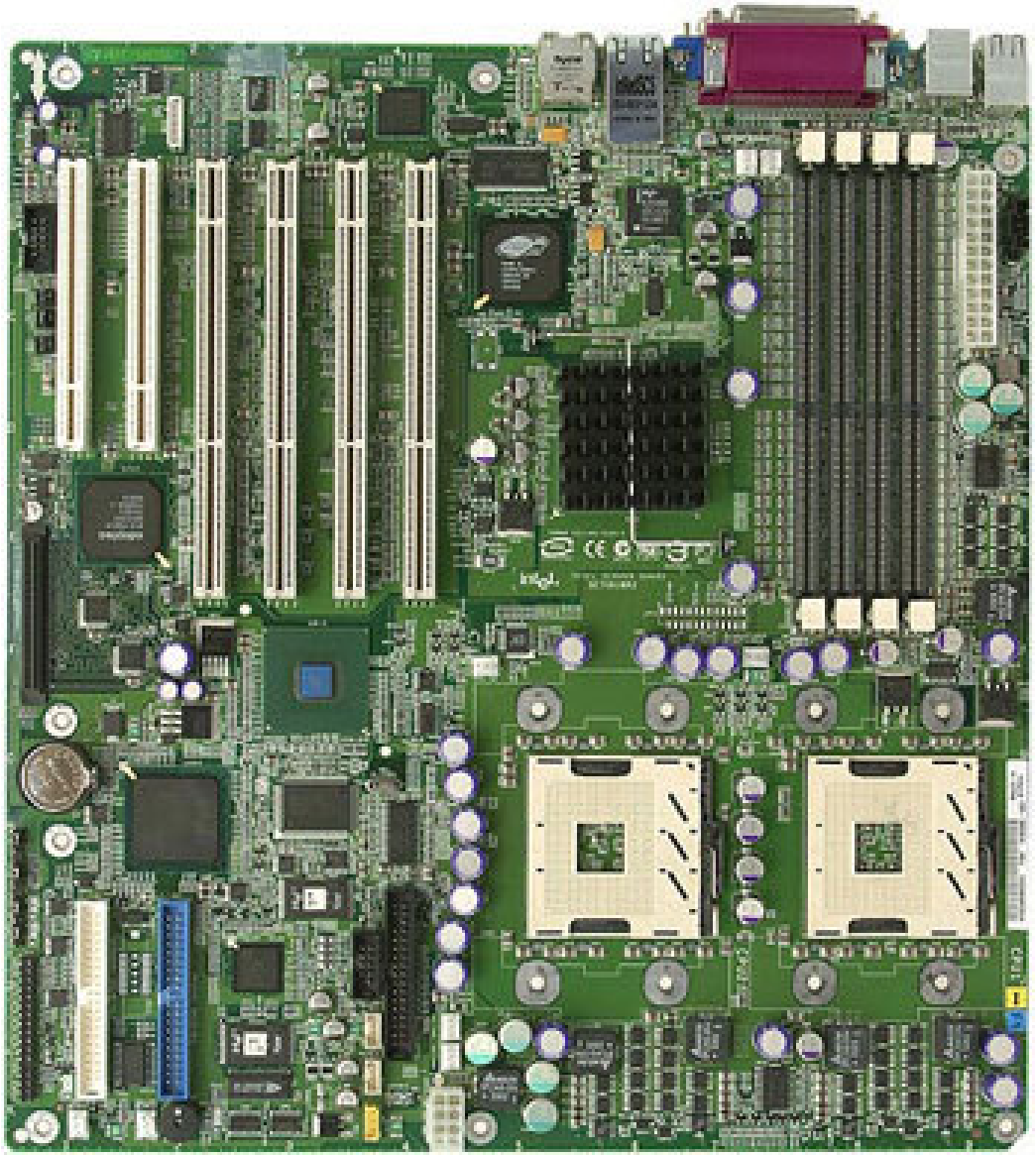
Case Study: Post-op

```
% mkdir kristaps ; cd kristaps
% ftp http://kristaps.bsd.lv/minicourse_12_2011/example1.c .
% cc -W -Wall -O2 -o example1 example1.c
% valgrind --tool=cachegrind ./example1
I   refs:      1,818,598,714
I1  misses:            1,423
LLi misses:            1,354
I1  miss rate:          0.00%
LLi miss rate:          0.00%
D   refs:        969,299,744  (968,863,668 rd   + 436,076 wr)
D1  misses:        7,584,406  (  7,571,336 rd   +  13,070 wr)
LLd misses:        7,584,016  (  7,570,986 rd   +  13,030 wr)
D1  miss rate:           0.7% (        0.7%     +     2.9%  )
LLd miss rate:           0.7% (        0.7%     +     2.9%  )
LL refs:           7,585,829  (  7,572,759 rd   +  13,070 wr)
LL misses:         7,585,370  (  7,572,340 rd   +  13,030 wr)
LL miss rate:            0.2% (        0.2%     +     2.9%  )
```

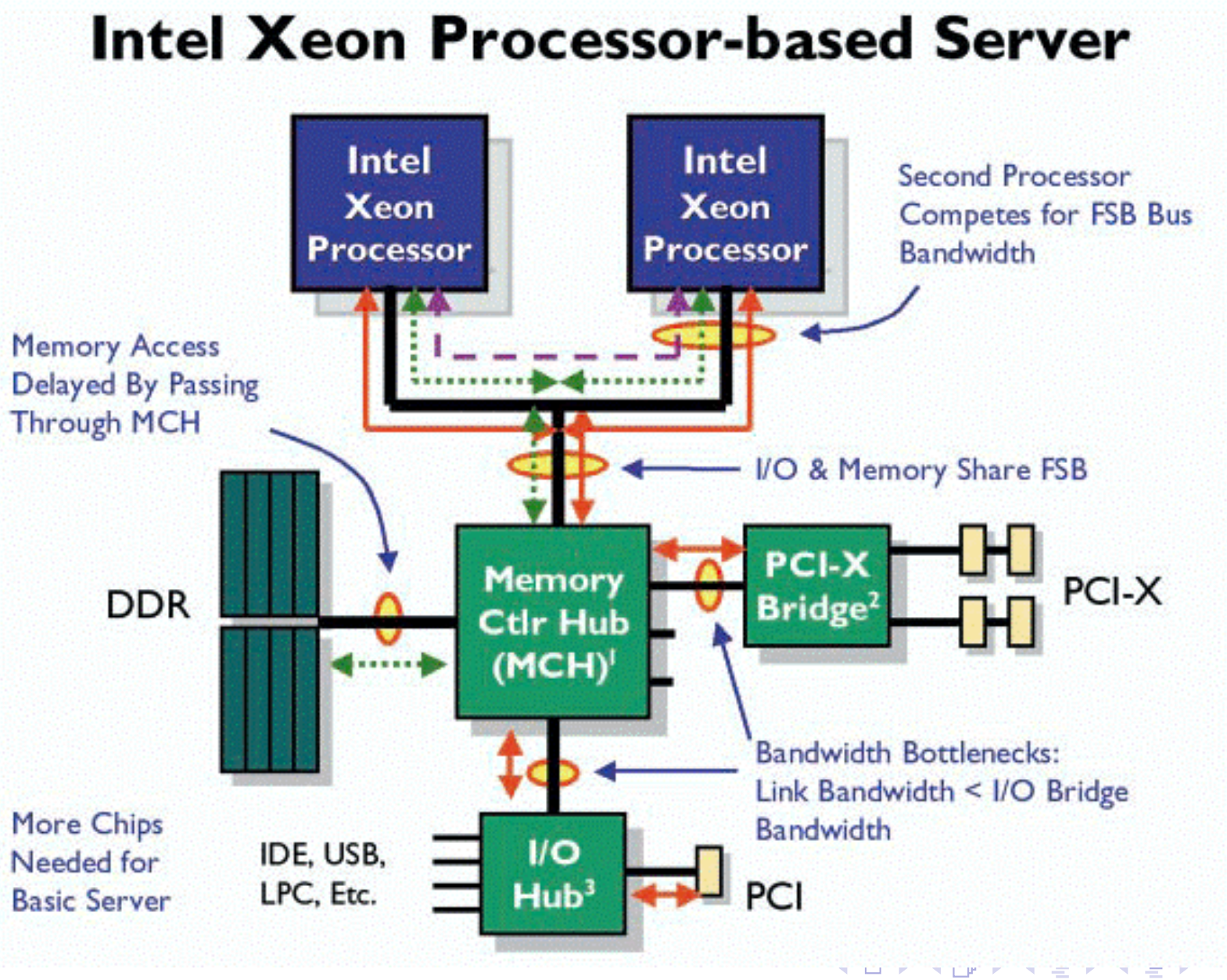
Case Study: Post-op

```
% mkdir kristaps ; cd kristaps
% ftp http://kristaps.bsd.lv/minicourse_12_2011/example2.c .
% cc -W -Wall -O2 -o example2 example2.c
% valgrind --tool=cachegrind ./example2
I   refs:      1,818,598,714
I1  misses:            1,423
LLi misses:            1,354
I1  miss rate:          0.00%
LLi miss rate:          0.00%
D   refs:        969,299,744  (968,863,668 rd   + 436,076 wr)
D1  misses:      136,140,031  (136,126,961 rd   +  13,070 wr)
LLd misses:      121,744,698  (121,731,668 rd   +  13,030 wr)
D1  miss rate:          14.0% (       14.0%     +     2.9%  )
LLd miss rate:          12.5% (       12.5%     +     2.9%  )
LL refs:         136,141,454  (136,128,384 rd   +  13,070 wr)
LL misses:       121,746,052  (121,733,022 rd   +  13,030 wr)
LL miss rate:            4.3% (        4.3%     +     2.9%  )
```

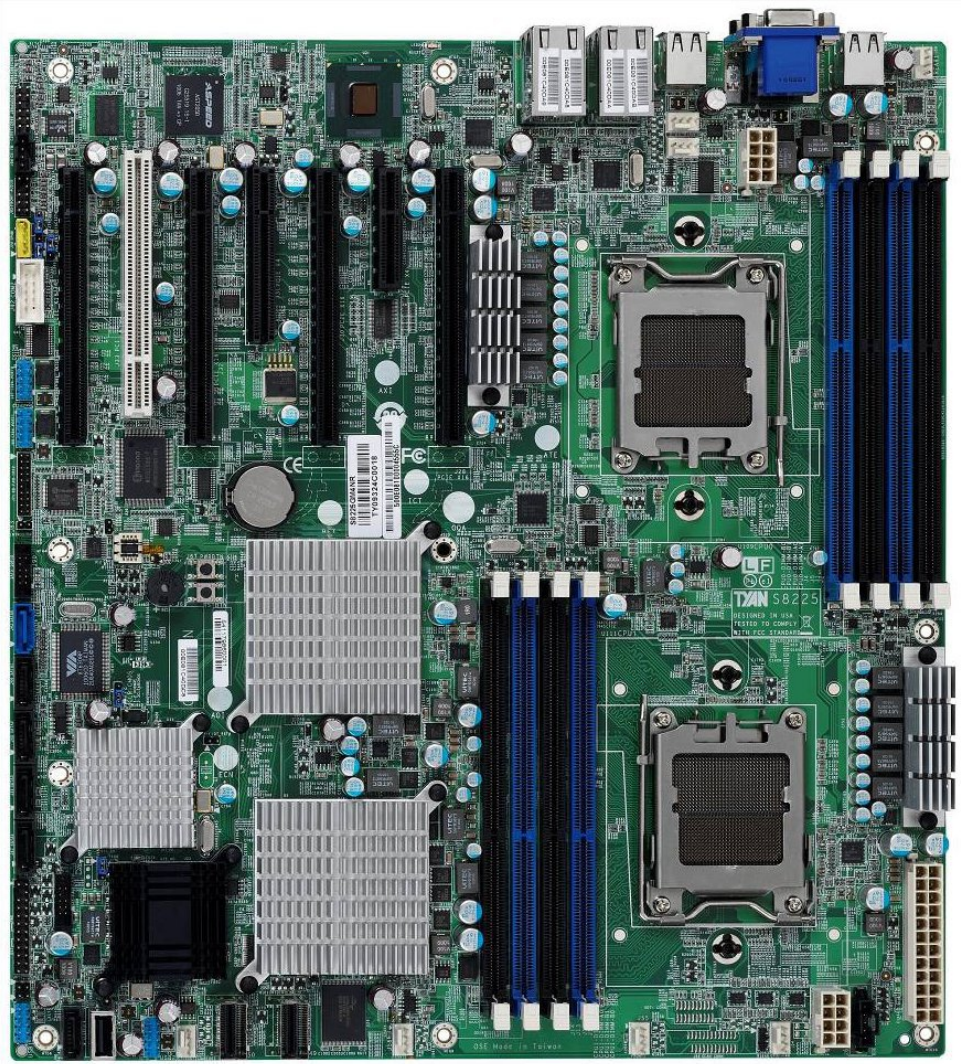
Hardware: Multiple CPUs (One FSB)



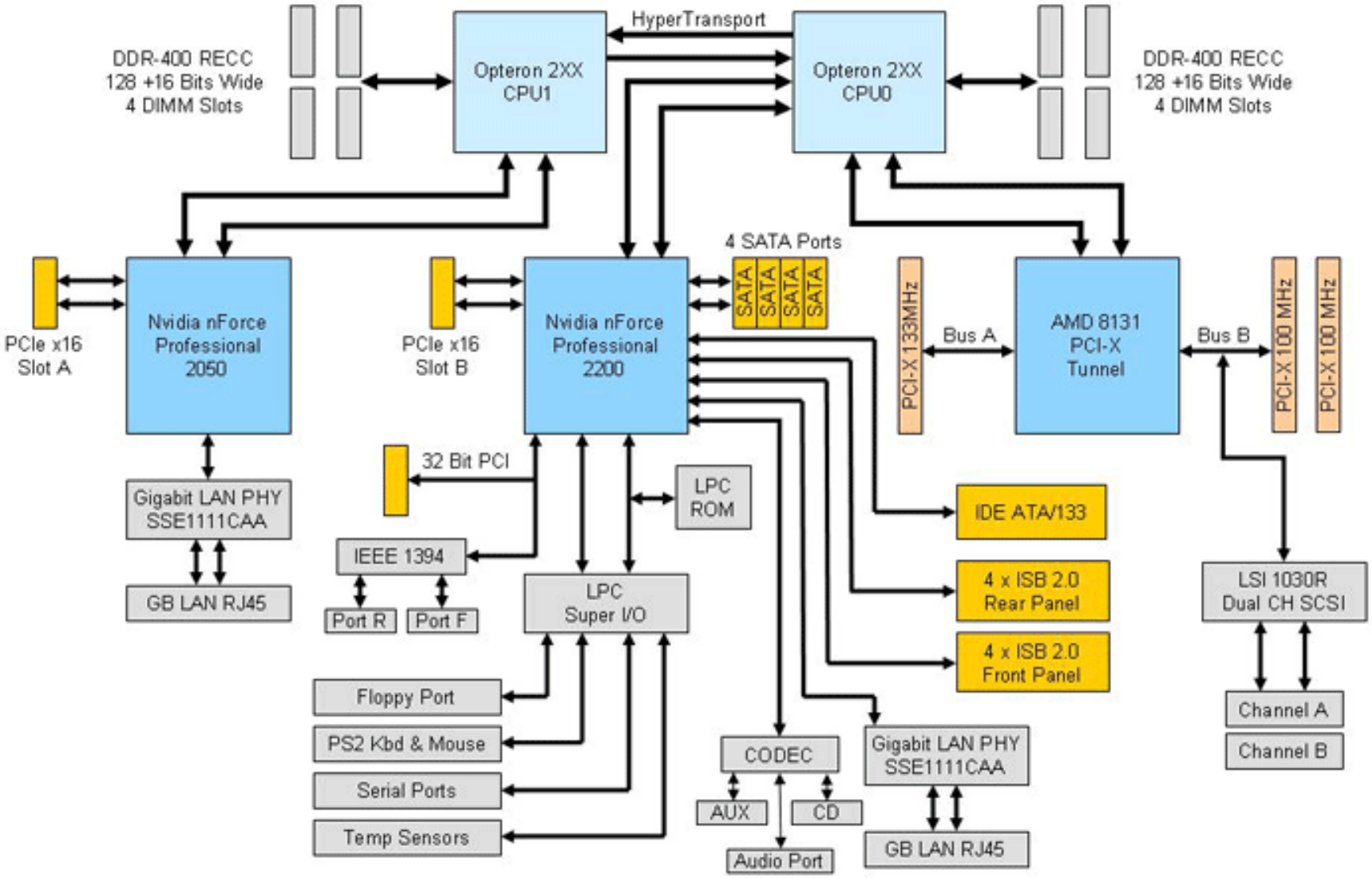
Hardware: Multiple CPUs (One FSB)



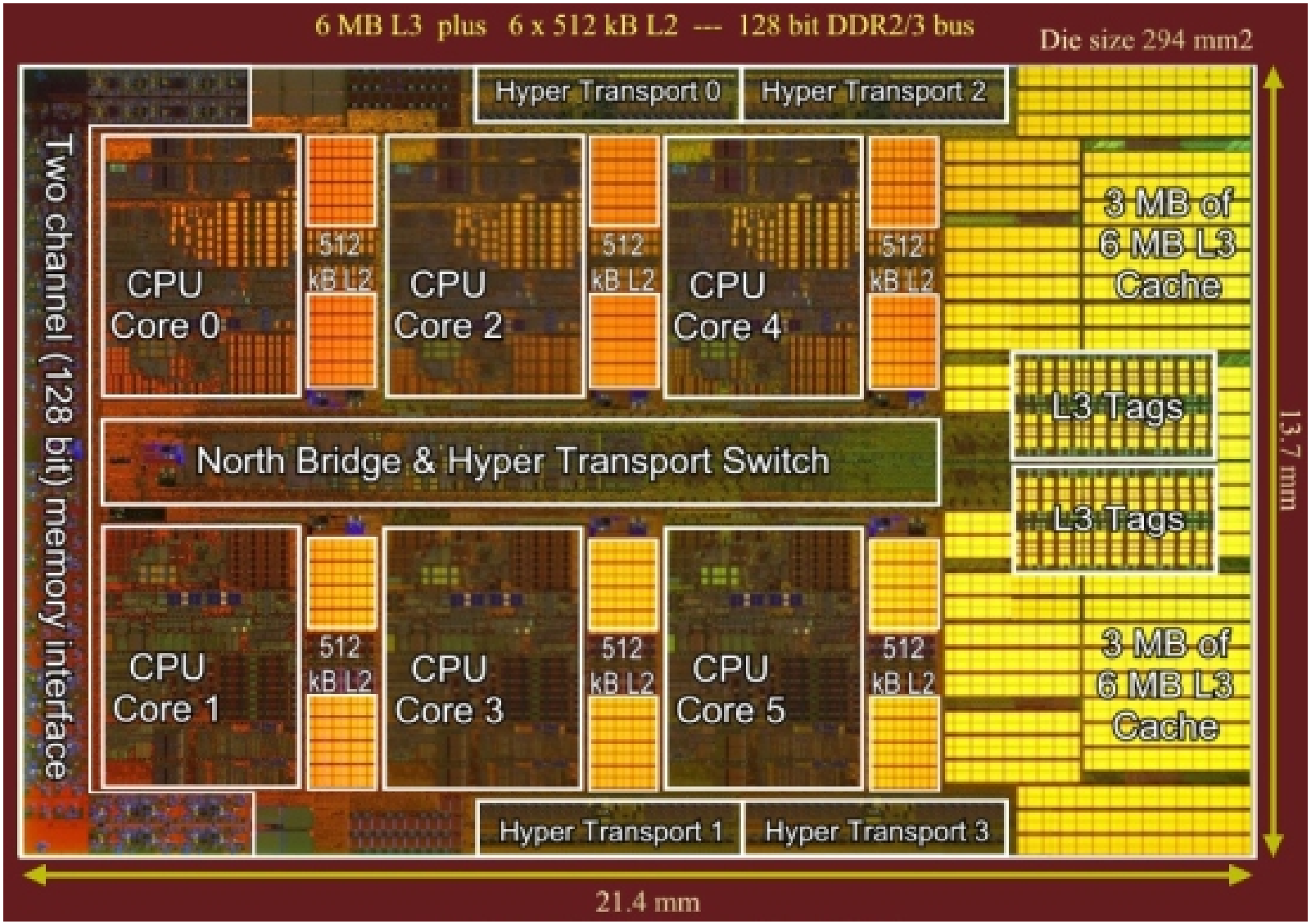
Hardware: Multiple CPUs (Multiple FSB)



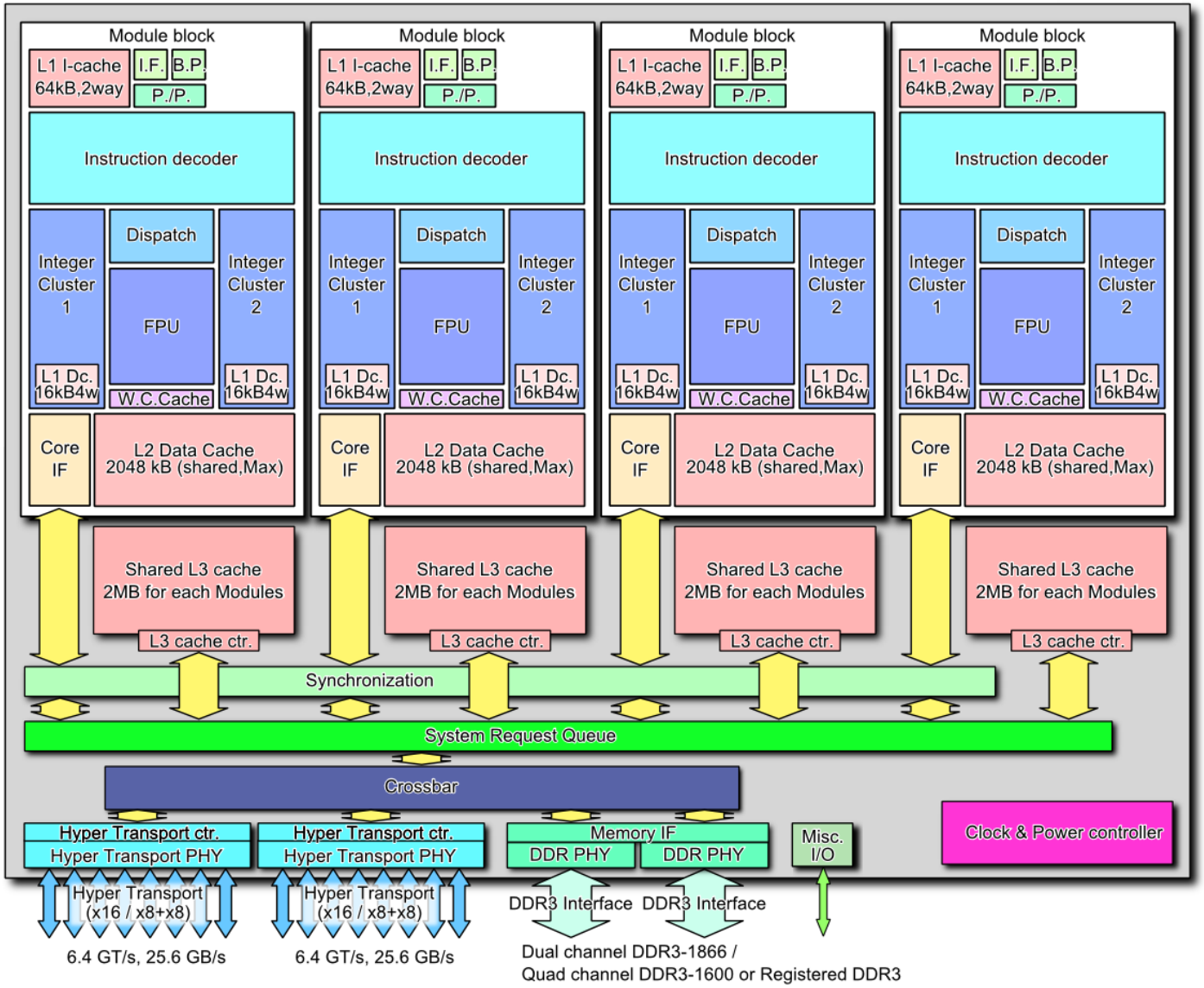
Hardware: Multiple CPUs (Multiple FSB)



Hardware: Multiple Cores



Hardware: Multiple Cores



Conclusion

In course1, we used *fork()* to split execution into two parts, each running on a core/chip.

How do we share memory between executing parts?