Analysis of Programming Languages

Introduction

Lembit Jürimägi

Why even have programming languages?

- Advantages
  - productivity
  - platform independence (to a certain extent)
- Disadvantages
  - not an optimal use of resources
  - programmers forget what is actually happening in hardware

Natural Language

- Means of communication
- Vocabulary
  - Words (nouns, verbs, adjectives, etc), names, punctuation
- Grammar
  - Flexible

Algorithmic Language

- Means of … programming, describing
- Vocabulary
  - Reserved words, constants, names (variables)
- Grammar
  - Very strict

Machine Language (Machine Code)

- Sequence of instructions
- Vocabulary (binary)
  - Instructions, constants, addresses, offsets
- Grammar?
  - Illegal instructions

Assembly Language

- Readable form of machine language
- Vocabulary
  - Reserved words, constants, labels
- Grammar
  - Very strict and simple

Natural Language vs Computer Language

- Number of words
- Flexibility of adding new words
- Complexity of grammar rules
- Flexibility of grammar
- Anything else?

Horse Trade

- You buy a horse for 5000€
- You sell the horse for 6000€
- You buy another horse for 7000€
- You sell that horse for 8000€

- How much money do you have?
- Why?

Context

- Derived from
  - previous statements
  - shared knowledge
  - personal knowledge

Horse Trade (C)

```
int cash;
cash -= 5000; // bought horse
cash += 6000; // sold horse
cash -= 7000; // bought another
cash += 8000; // sold it
printf("%d\n", cash);
```

Chomsky's Mathematical Model of Grammar

- G = {S, N, T, R}, where
- R – production rules
- T – terminal symbols
- N – nonterminal symbols
- S – starting symbol

- Grammar is used to generate all possible sentences in a language

Chomsky's Mathematical Model of Grammar

Example:

S: S

N: { A, N, V }

T: { boy, dog, happy, is, loud }

R: S -> N V A

N -> boy

N -> dog

V -> is

A -> happy

A -> loud

Chomsky's Hierarchy of Grammars

- Regular
  - Example: regular expressions
  - Implementation: finite state machine
- Context-Free
  - Example: most programming languages
  - Implementation: stack machine (pushdown automaton)
- Context-Sensitive
  - Example: ?
  - Implementation: memory machine
- Unrestricted
  - Example: natural languages
  - Implementation: ?

Critique of Chomsky

- Mathematical model isn't really useful for natural languages
- However, most computer languages have context-free grammar, even the ones that were created before Chomsky's theory
- For natural language tasks like speech recognition, other models are used
- Example: n-gram
  - N words in sequence that have high probability to belong together
  - Audio-to-text "dog is wood"
  - 3-gram determines that "dog is good" is way more probable than "dog is wood" and replaces it

State of computer "languagescape"

- Humans are conservative and don't like change
- There is an impressive library of already existing software
  - In binary form
  - In some programming language
- This requires programming languages and even machine code to be backwards compatible
  - C is 48 years old
  - x86 assembly language is compatible with Intel 8080 from 46 years ago
- However technology has changed significantly and this backwards compatibility is hurting IT

Parallel programming with OpenMP

```
x = 0;
sum = 0.0;
step = 1.0/(double) num_steps;




for (i=0; i<num_steps; i++){
    x = (i+0.5)*step;
    sum = sum + 4.0/(1.0+x*x);


}

pi = step * sum;
```

```
x = 0;
sum = 0.0;
step = 1.0/(double) num_steps;
#pragma omp parallel
private(i,x,aux) shared(sum)
{
#pragma omp for schedule(static)
  for (i=0; i < num_steps; i++){
      x=(i+0.5)*step;
      aux=4.0/(1.0+x*x);
#pragma omp critical
      sum = sum + aux;
  }
}
pi = step * sum;
```

The von Neumann Paradigm (1940s)

$$\lim_{i \to \infty} \left(\frac{TALU\ (i)}{TCOMM\ (i)} \right) \to \infty$$

Optimal Solution: Finite Automata

The Nobel Laureate Richard Feynman Observations

$$\lim_{i \to \infty} \left(\frac{TALU \quad (i)}{TCOMM \quad (i)} \right) \to 0 (t \to \infty)$$

Where is the technology now?

A. Closer to 1940s?

B. Closer to t->∞?

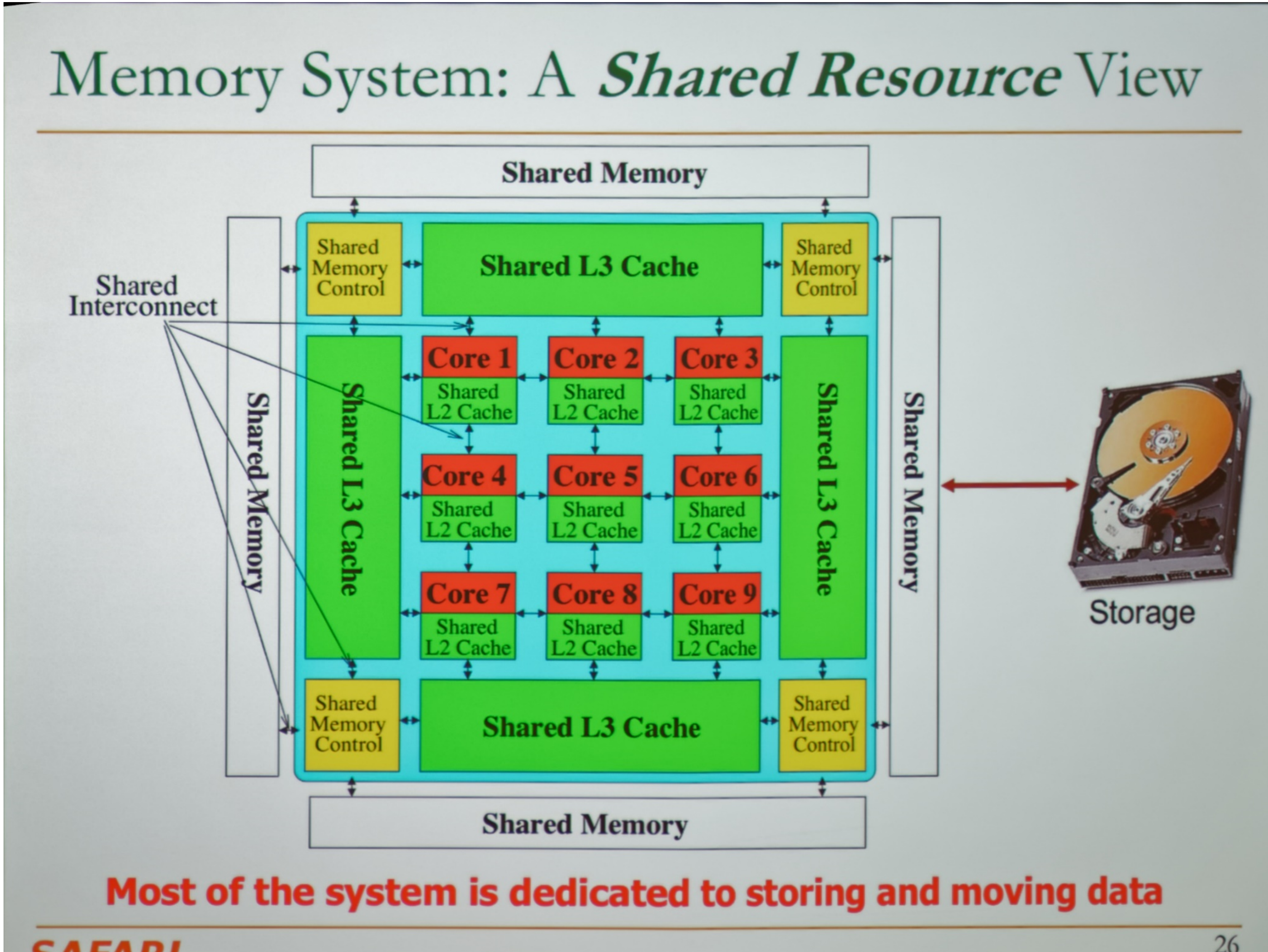
State of the Art in Technology Today

~~The Power Challenge~~

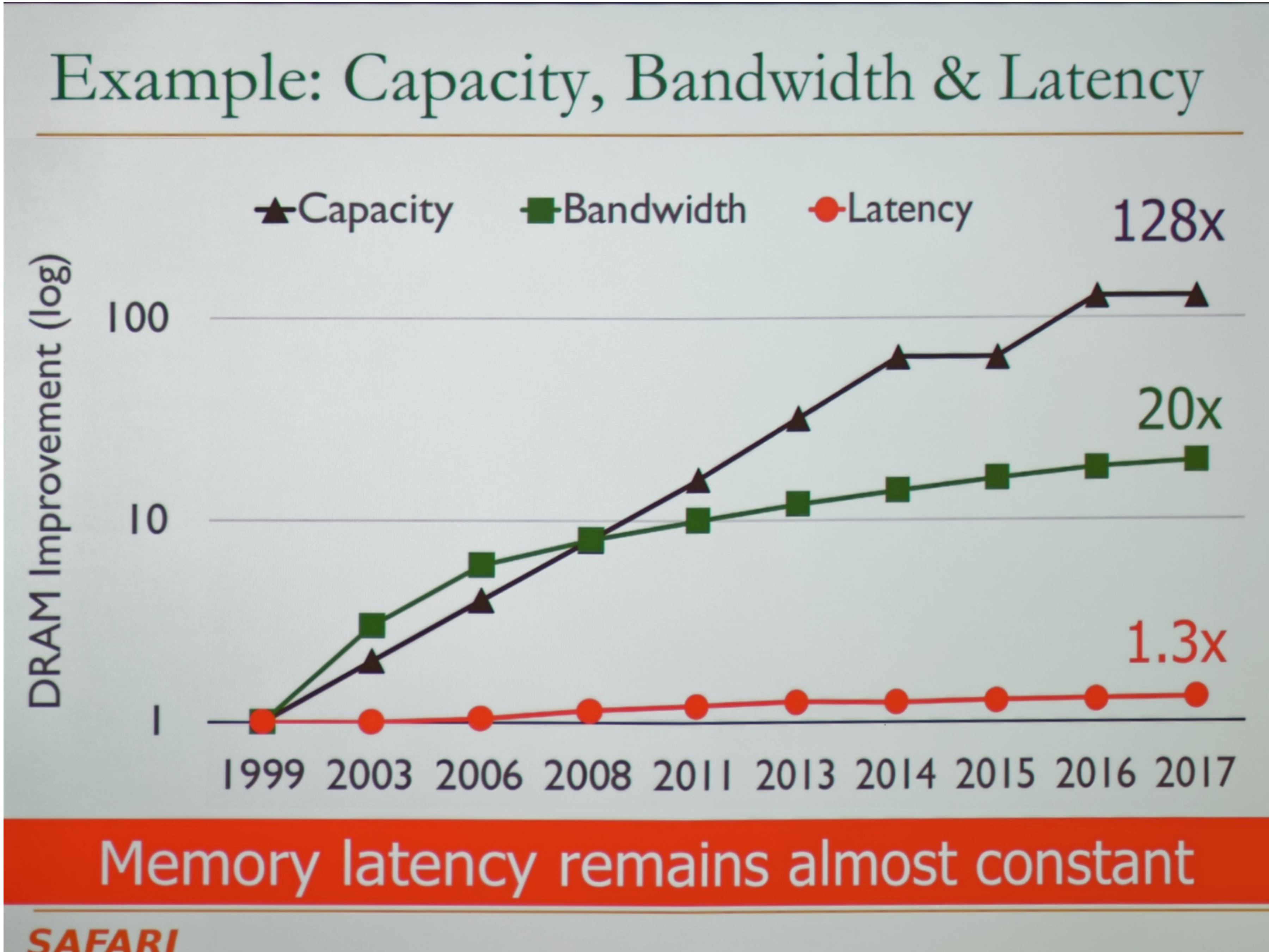
The Data Movement Challenge

	2015	2020
Double precision FLOP	100pj	10pj
Moving data on-chip: 1mm	6pj	
Moving data on-chip: 20mm	120pj	
Moving data to off-chip memory	5000pj	2000pj

- Moving data off-chip will use **200x more energy than computing**!
- Moving data in 1940s was using **1/60x** ...



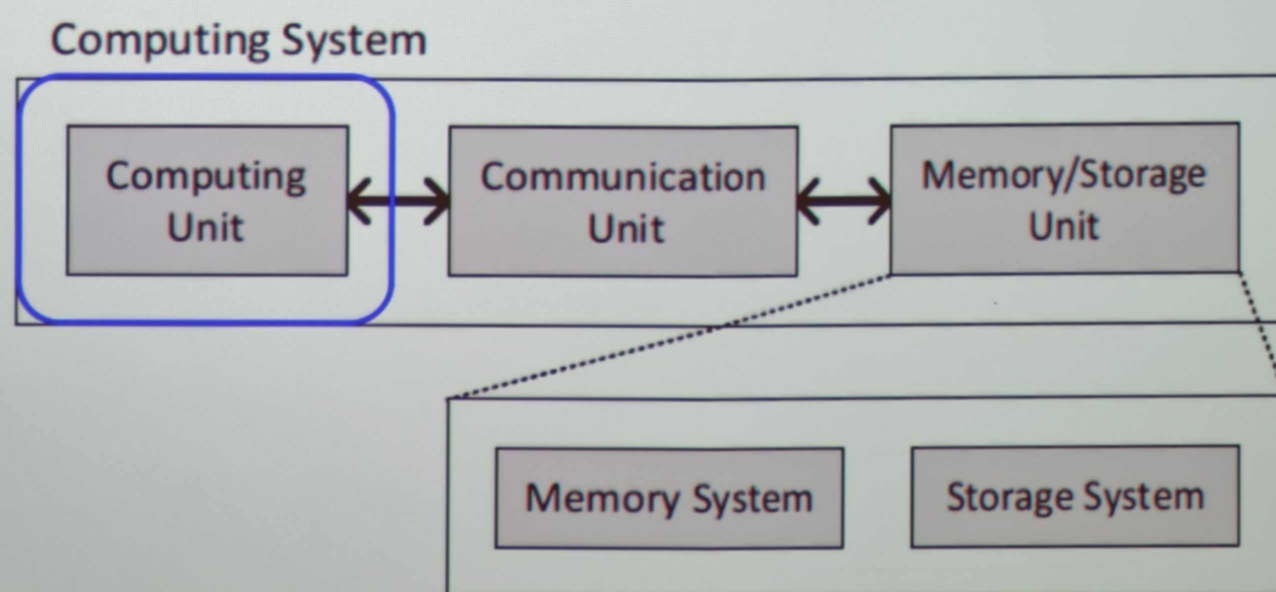
O. Mutlu, Processing Data Where It Makes Sense: Enabling In-Memory Computation, MECO 2018

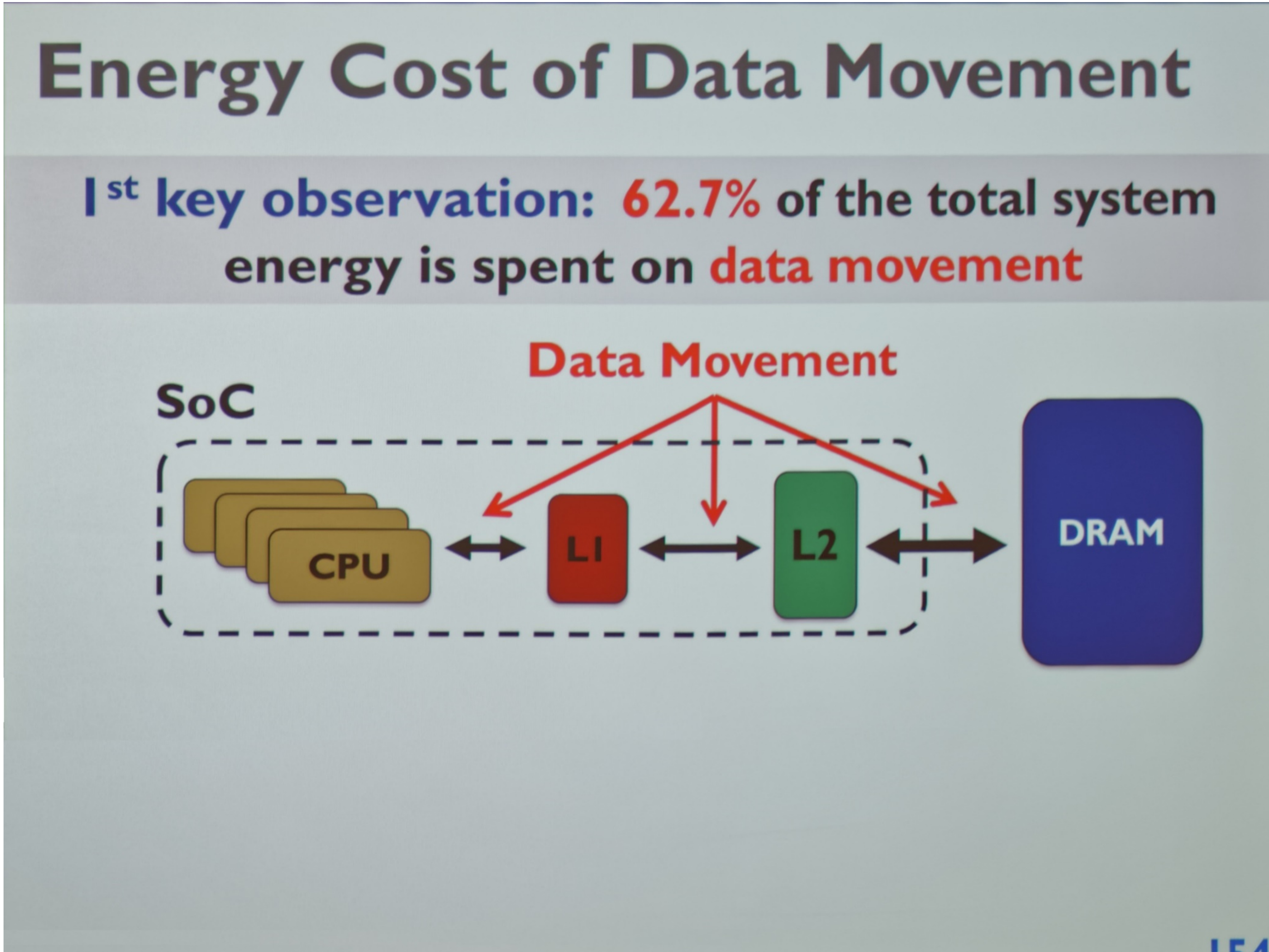


O. Mutlu, Processing Data Where It Makes Sense: Enabling In-Memory Computation, MECO 2018

Today's Computing Systems

- Are overwhelmingly processor centric
- All data processed in the processor → at great system cost
- Processor is heavily optimized and is considered the master
- Data storage units are dumb and are largely unoptimized (except for some that are on the processor die)





O. Mutlu, Processing Data Where It Makes Sense: Enabling In-Memory Computation, MECO 2018

Why take this course?

- To get an overview of how compiling works
- To get some insight into why some language constructs have been made the way they are
- To get some knowledge on how the computer hardware will interprete your code (and perhaps be able to take it into account)

- To be able to construct a parser (or interpreter) on your own if necessary