Fresh Breeze
A Radical Approach to Massively Parallel Architecture and Programming

Jack Dennis

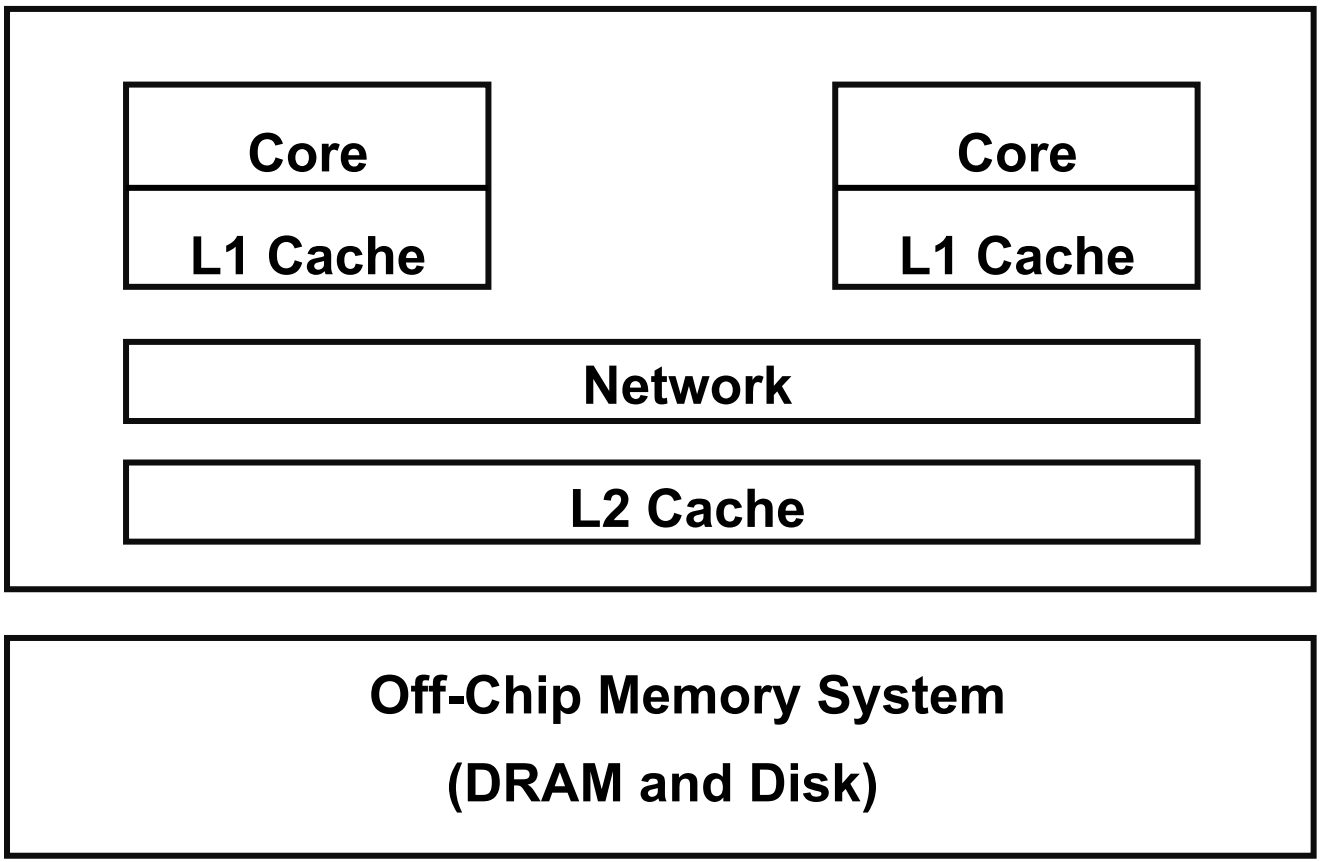
MIT-CSAIL

Computer Science and Artificial Intelligence Laboratory

The Multi Core Challenge

- Many processing cores provides for high potential performance.
- Goal: Achieve high core utilization
- Goal: With highest Energy Efficiency.
- Goal: Support Modular Construction of Software for Parallel Computation.
- Goal: Unify Memory with the File System.

Typical Processor Chip



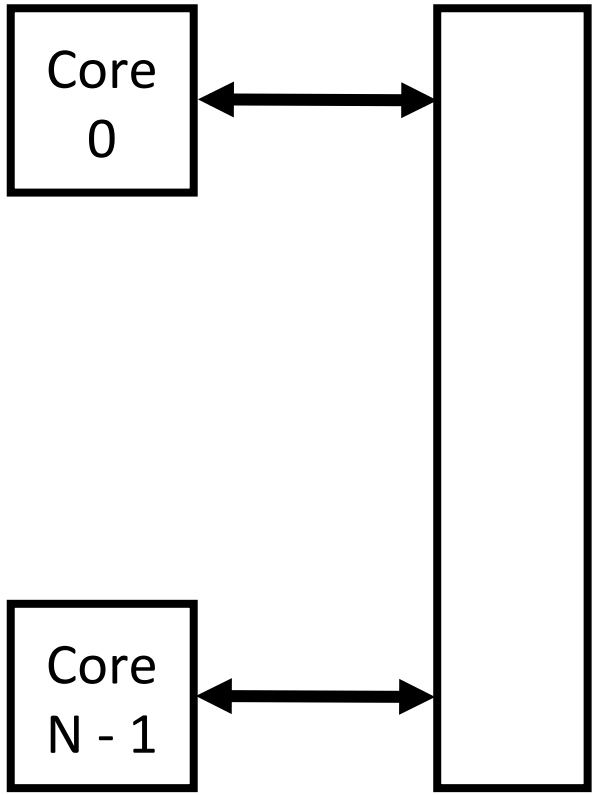
The Popular Approach

MPI: Message Passing Interface

Issues:

- Overhead
- No satisfactory notion of Program Module
- Difficult sharing of data objects

Message Passing System



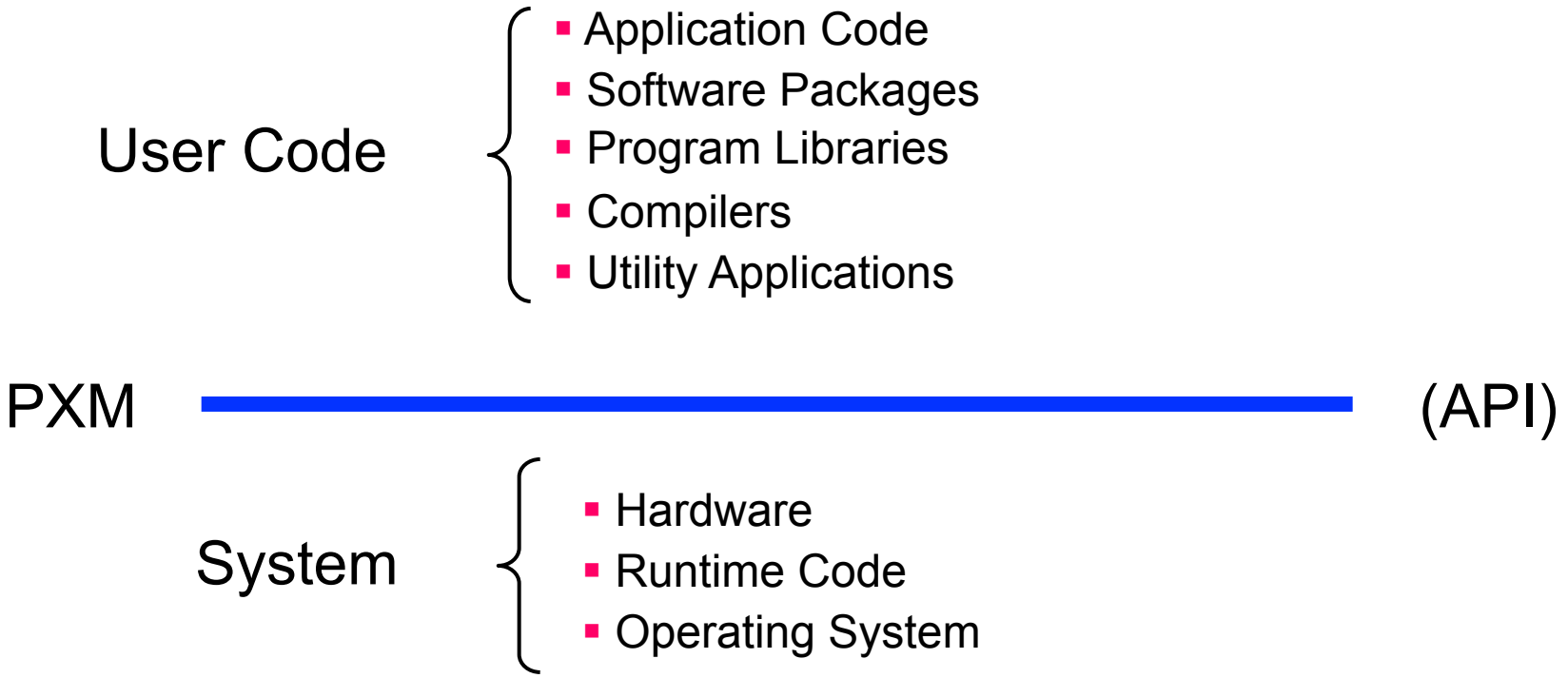
Core
Interconnection
Network

Basic Commands:
Send m to p
Receive m from p

The Fresh Breeze Project

- Co-design of Programming Model and System Architecture.
- Goal: Support Fine-Grain Dynamic Resource Management.
- Goal: Support Modular Construction of Software for Parallel Computation.

What is a Program Execution Model?



Features a User Program Depends On

Features expressed within a Programming language

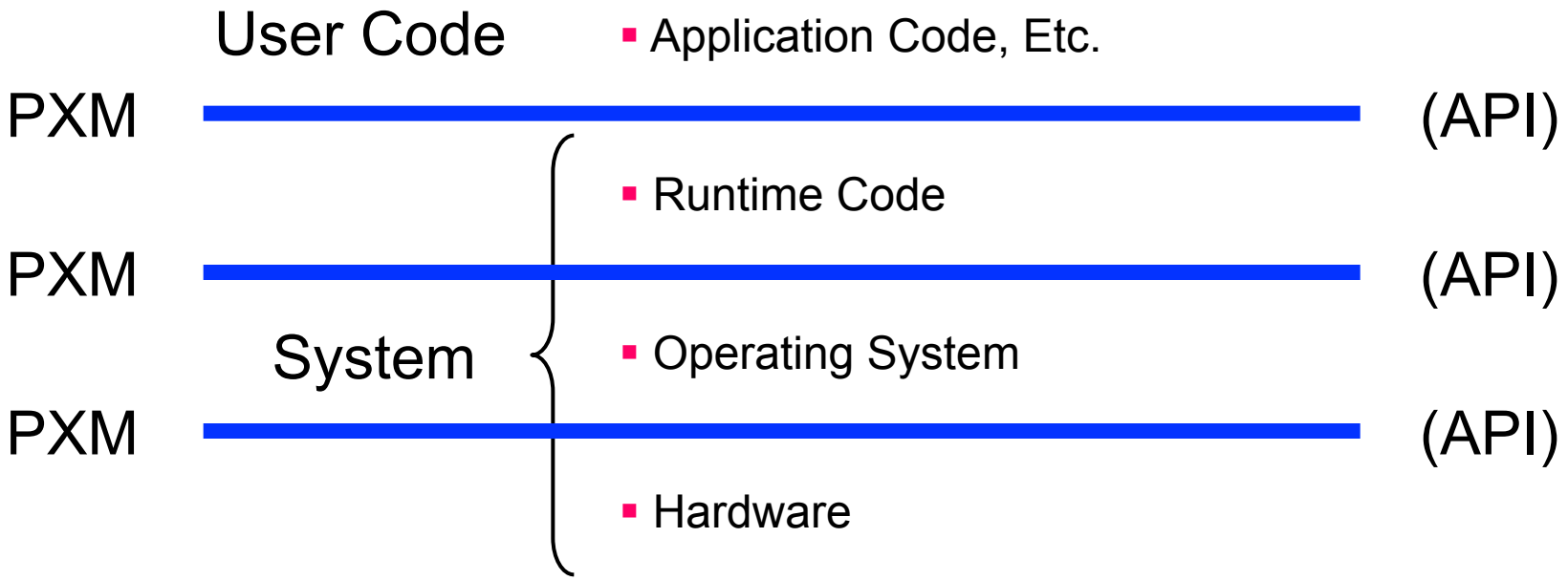
- Procedures; call/return
- Access to parameters and variables
- Use of data structures (static and dynamic)

But that's not all !!

Features expressed Outside a (typical) programming language

- File creation, naming and access
- Object directories
- Communication: networks and peripherals
- Concurrency: coordination; scheduling

Today's Conventional Software Stack



Each system layer compensates for inadequacies of the layers below, leading to an inefficient whole.

Fresh Breeze Characteristics

- Use of fixed size **memory chunks** to represent all data objects, simplifying dynamic memory management. **Write once** data eliminates cache consistency problems.
- Use of **codelets** executed according to **data flow** principles yields a fine-grain **tasking model**.
- Hardware **task scheduler** and **load balancer** provide highly effective dynamic management of processing load.

Project Components

- The funJava Programming Language for functional programming to support parallel execution.
- The Fresh Breeze architecture for parallel computing with fine-grain execution of many codelets.
- The Kiva system simulator capable of cycle accurate simulation of systems with large numbers of components.
- The Fresh Breeze compiler for generating codelets for highly parallel computation from funJava programs.

funJava

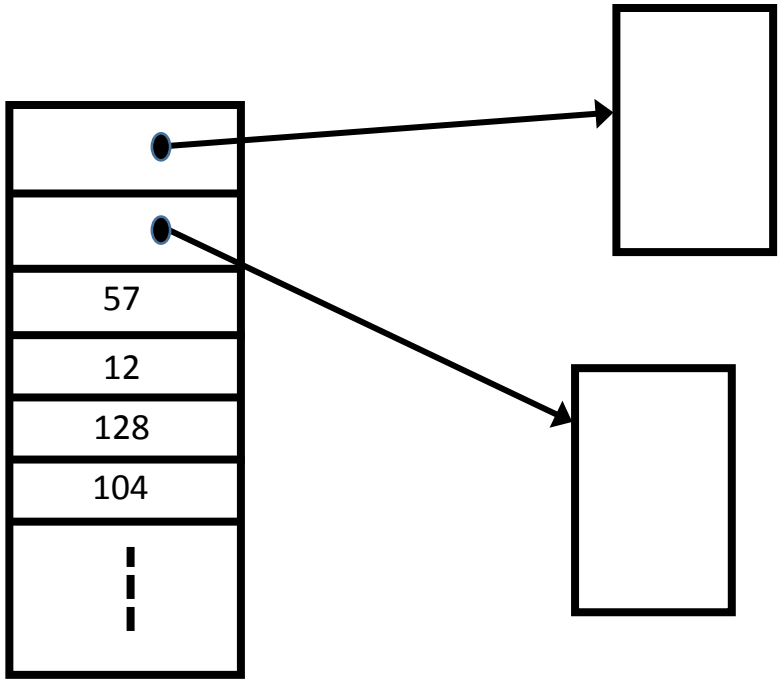
A Functional Programming Language

- A language in which all forms of parallelism are readily expressed: Expression Parallel, Data Parallel, Producer-Consumer and Transaction Processing.
- A high level programming language in which data streams are first class data objects
- Retains the type secure features of the Java language.

Flexibility of resource management requires choice of a unit of exchange for memory and for processing

- **Unit of Memory – Fixed Size Memory Chunk**
- **Unit of Processing – Execution of a Codelet**

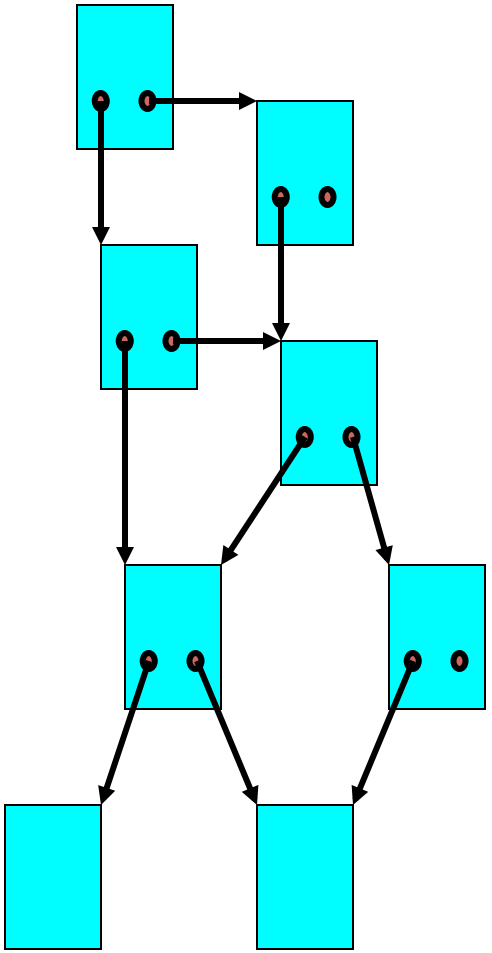
What is a Memory Chunk ?



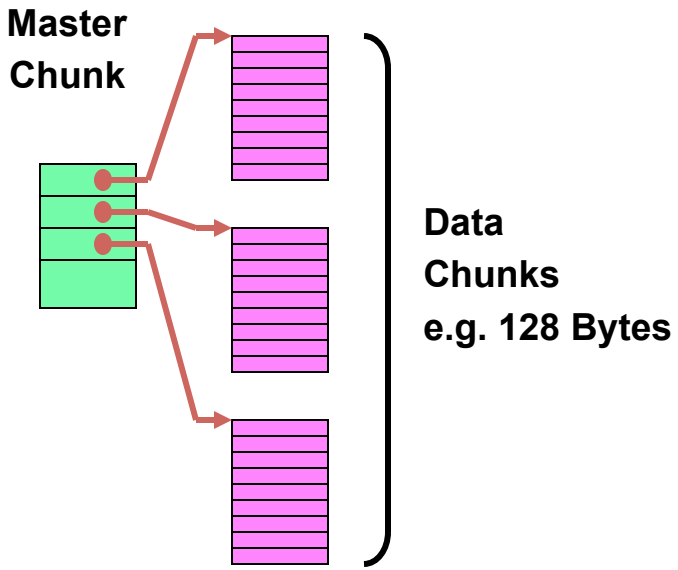
A chunk holds sixteen data items that may be data values or pointers to other memory chunks

Data Structures as Trees of Chunks

Cycle-Free Heap



Arrays as Trees of Chunks

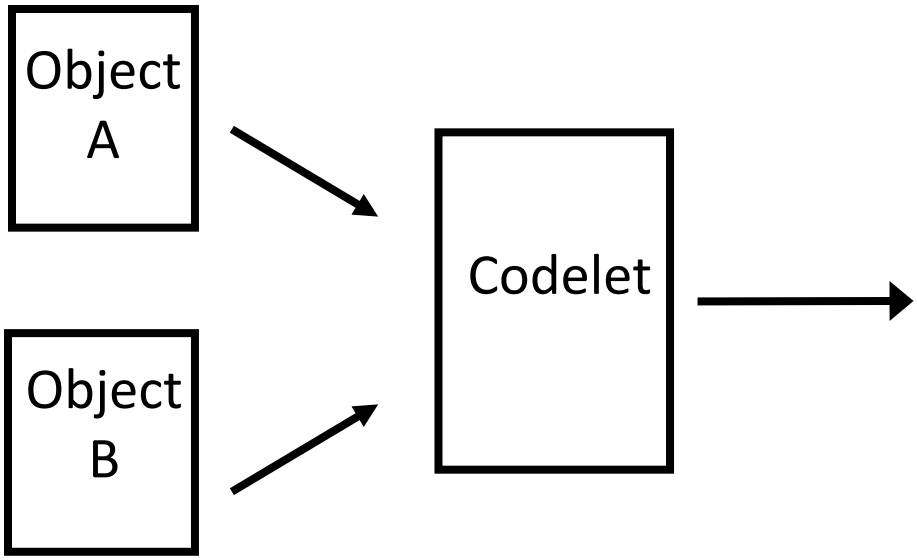


- **Fan-out as large as 16**
- **Arrays: Three levels yields 4096 elements (longs or doubles)**
- **Write-Once then Read Only**

Benefits of the Memory Model

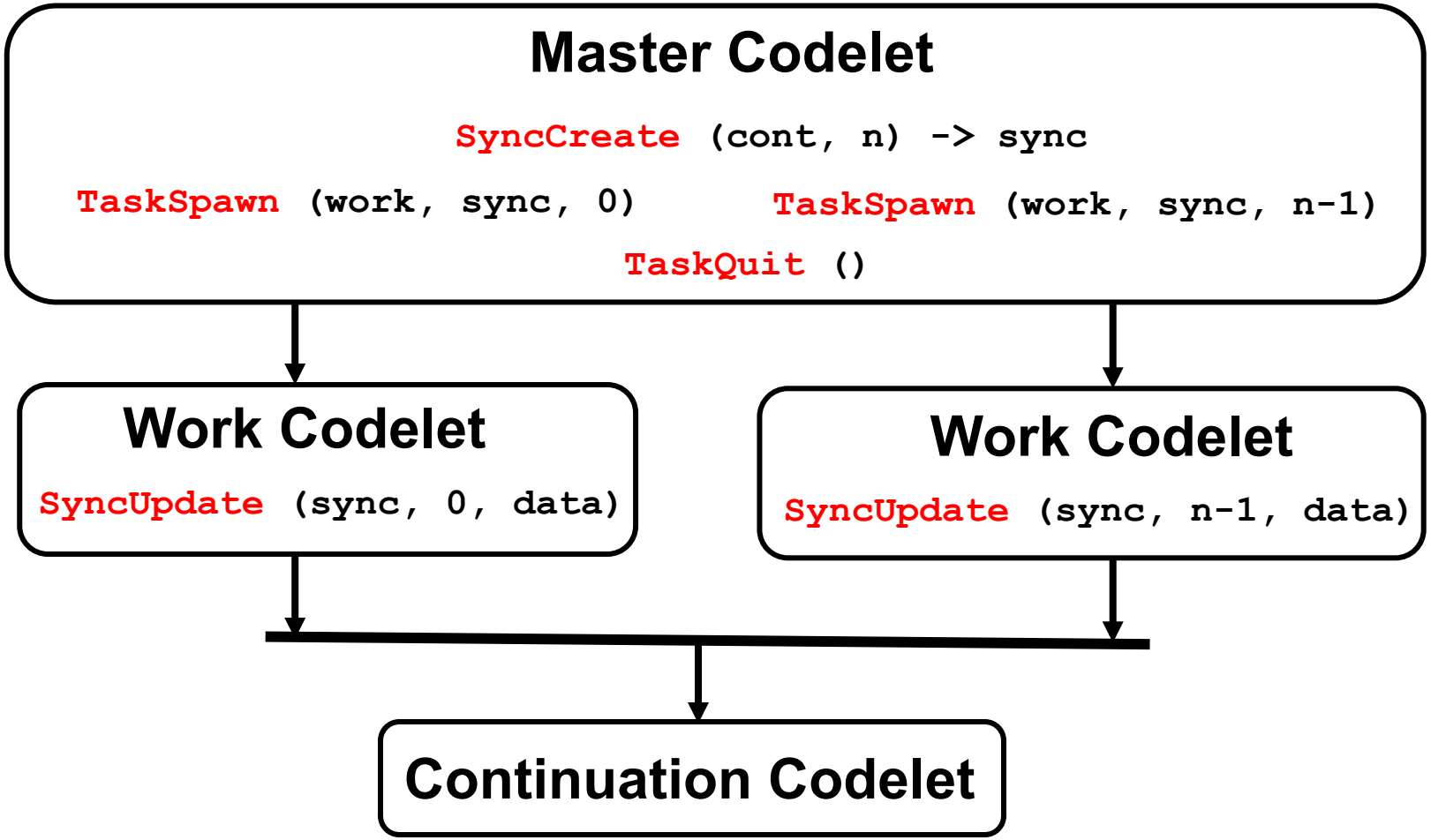
- **Uniform representation scheme for all data objects**
- **Ease of selecting components of a data object.**
- **Simplified memory management.**
- **Write-once policy eliminates coherence issues**

What is a Codelet ?

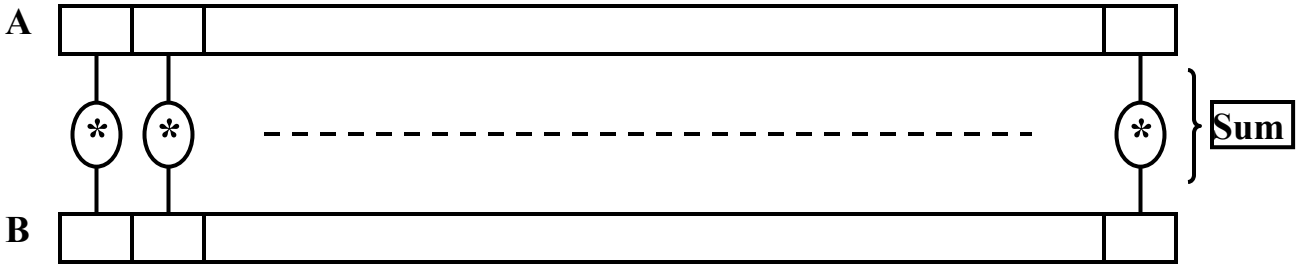


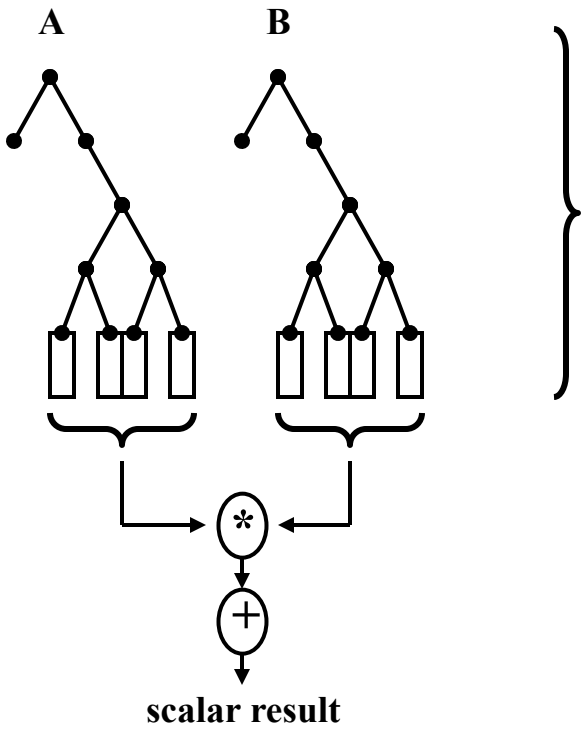
- A block of Instructions scheduled for execution when needed data objects are available.
- Results made available to successor codelets.
- Data objects are trees of chunks.

Work and Continuation Codelets (Data Parallel Computation)



Example: The Dot Product

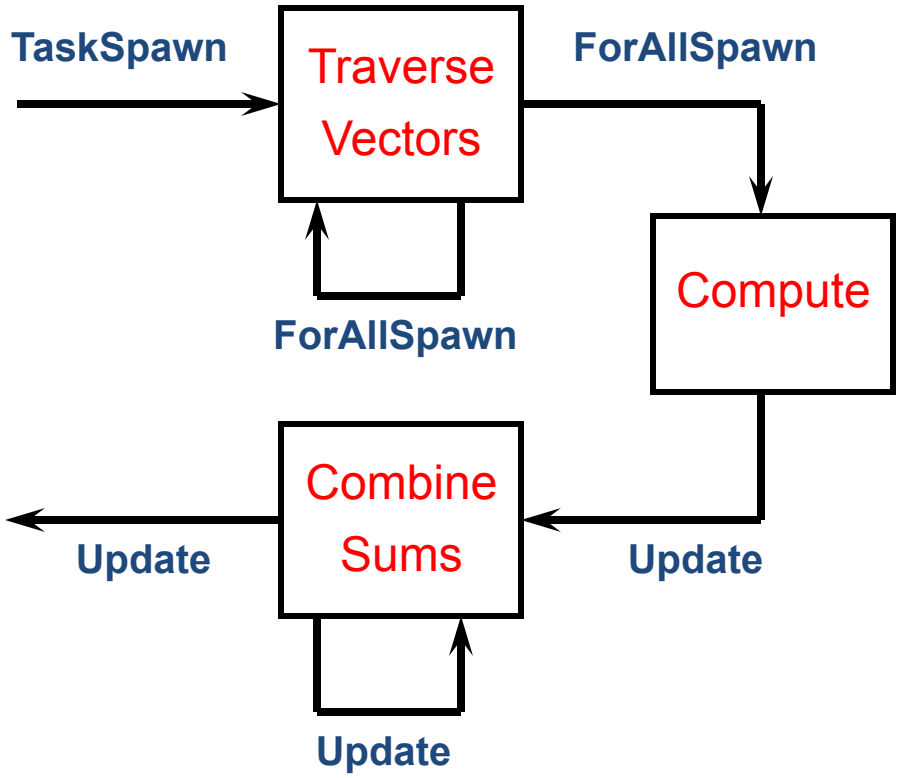




5 levels:
Vector length =
16^5 = 1,048,576

Each of 65536 Leaf Tasks:
Dot Product of two
16-element vectors:
16 multiplies; 15 adds

Codelets for the Dot Product



Fresh Breeze Multicore Chip

S - Scheduler

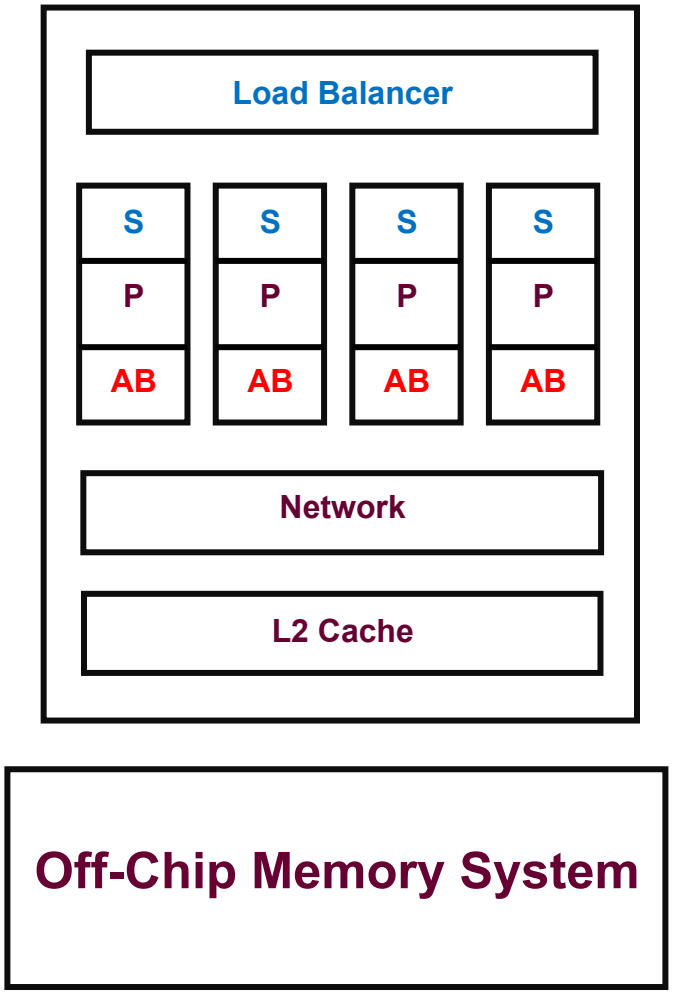
P - Processor Core

AB - AutoBuffer

Innovations:

AutoBuffer - AB

Load Balancer



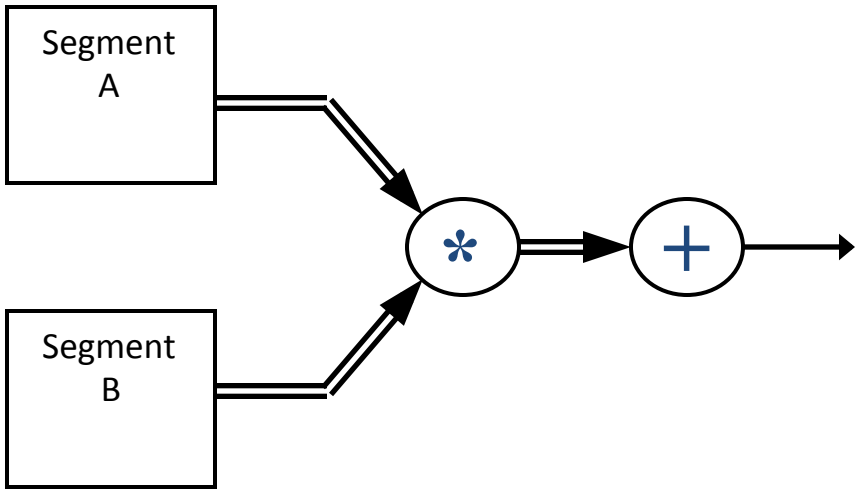
Linear Algebra: Three Algorithms

- Dot Product
- Matrix Multiply
- Fast Fourier Transform

Let's consider the special characteristics of each.

Dot Product

Leaf Task: Dot Product of 16-element segments A and B



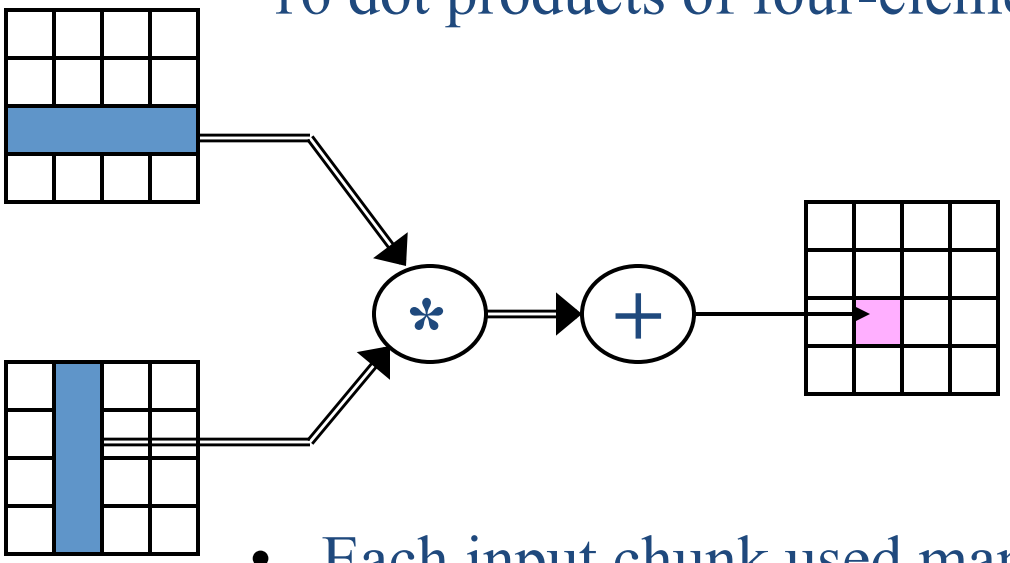
16	Multiplies
15	Adds
31	Operations

- No data reuse
- No intermediate data
- Large volume of input data

Matrix Multiply

Leaf Task: Product of two 4-by-4 matrices

16 dot products of four-element vectors



64 Multiplies

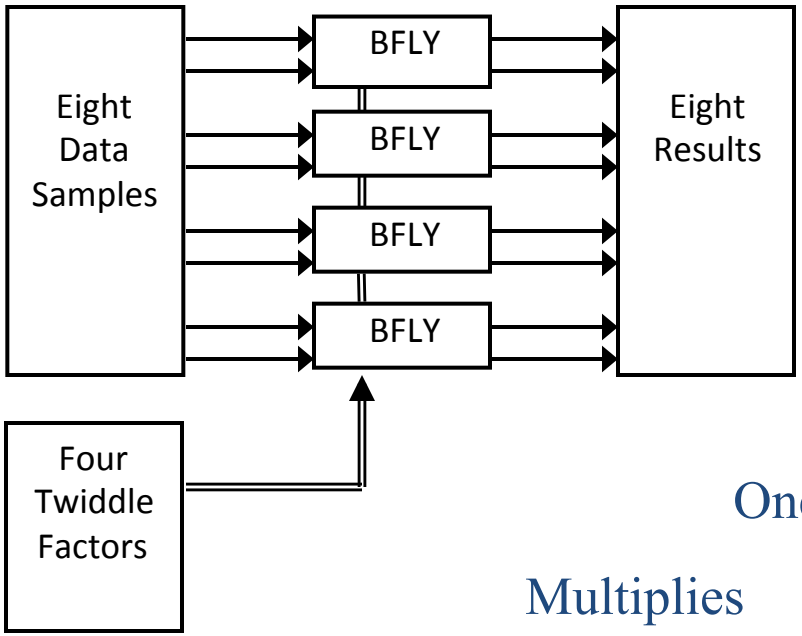
48 Adds

112 Operations

- Each input chunk used many times
- Result chunks written to memory
- No intermediate data
- Relatively small input data

Fast Fourier Transform

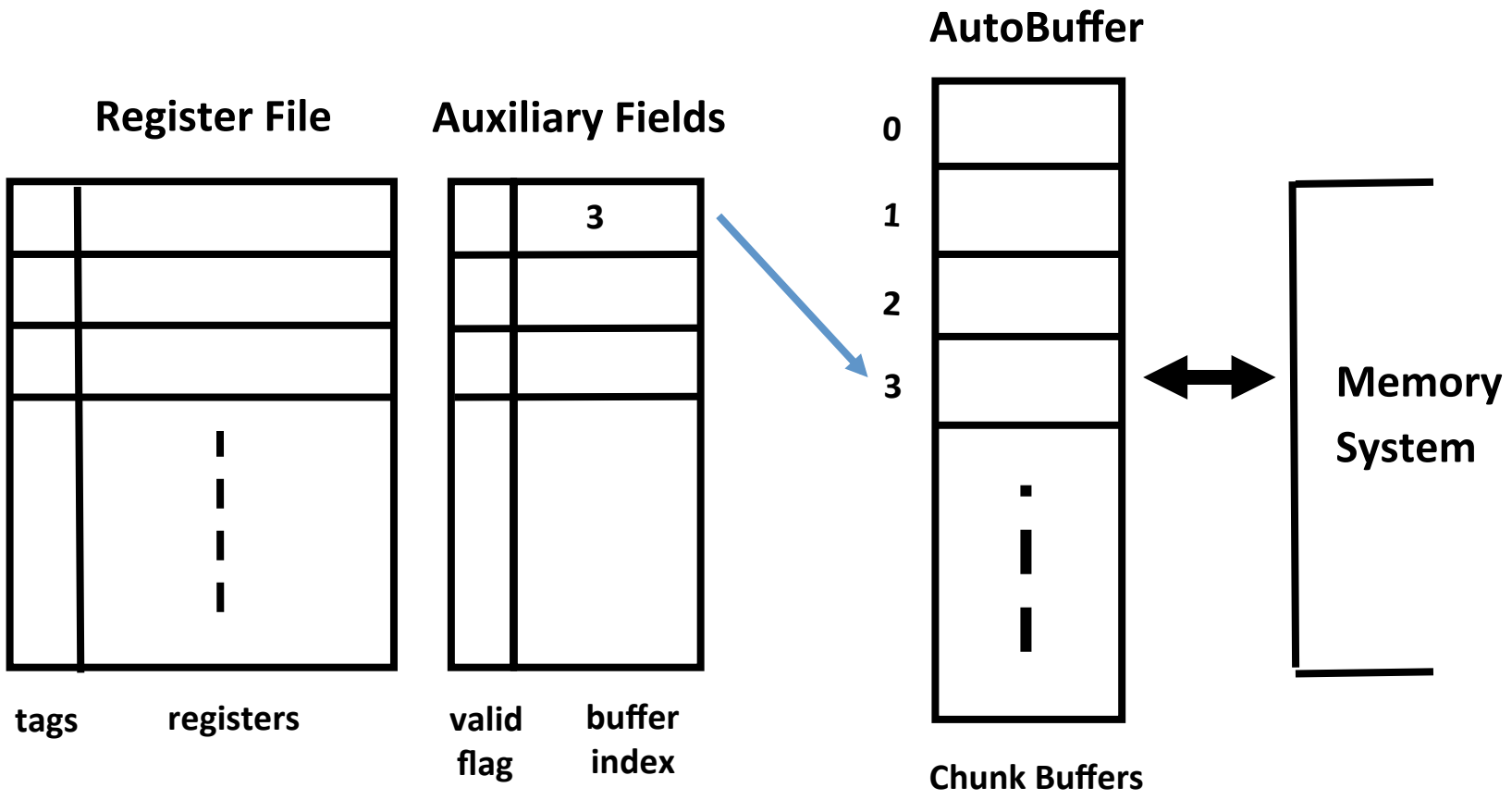
Leaf Task: Group of Four Butterfly Computations



- Log_2 (n) stages
- Intermediate data
- Chunks written and read

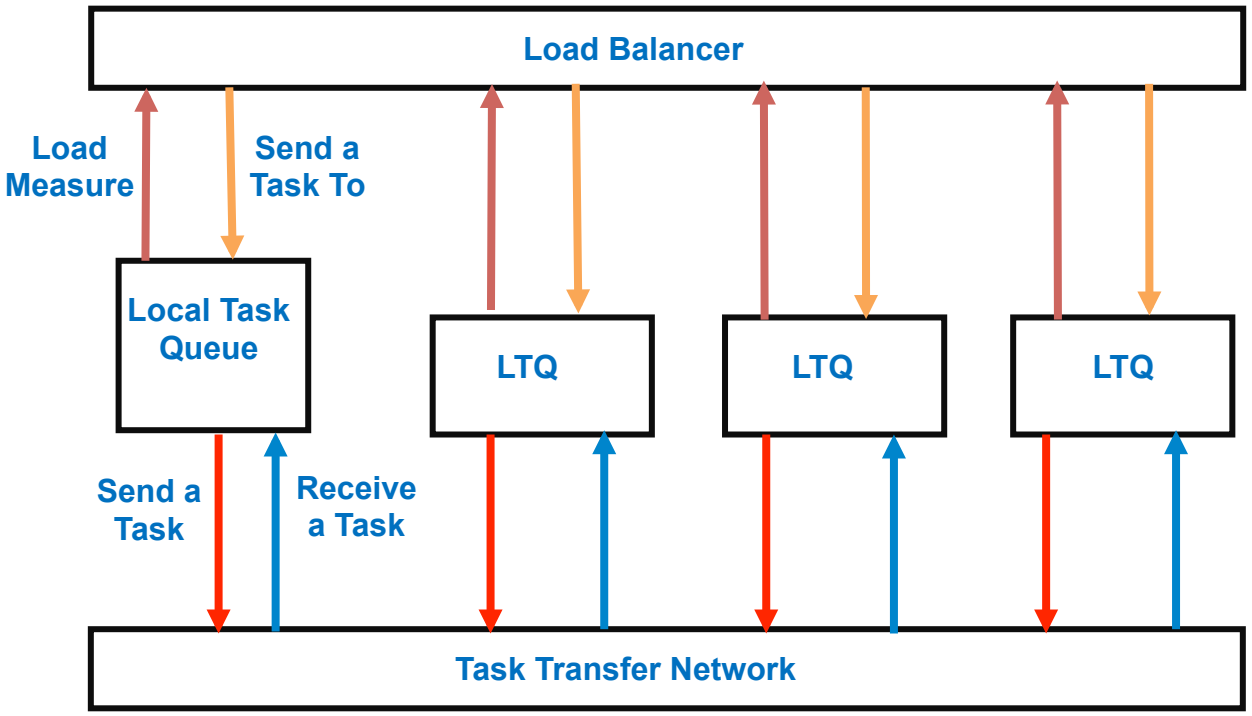
	One Butterfly	Four Butterflies
Multiplies	4	16
Adds	6	24
Operations	10	40

Principle of the Auto Buffer



Codelets access chunks using chunk handles held in processor registers. Once a chunk is assigned a buffer, its index is held by the register containing the handle, providing direct access to the chunk.

Dynamic Load Balancing



The load Balancer monitors the number of tasks queued at each processor and instructs local schedulers to send tasks from processors with high load to processors with low load.

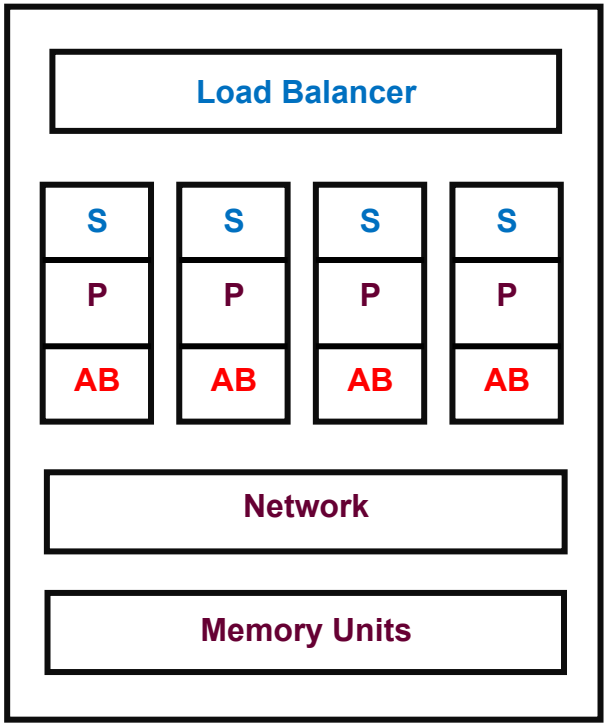
The Task Record

- Codelet – index of codelet within the codelet library.
- Arguments – The handle of an argument chunk

Simulated Fresh Breeze System

System Parameters

- **Number of cores**
- **Execution Slots**
- **Size of AutoBuffer**
- **Latency of Read**



Speed Up Data – Dot Product

Depth											
5	0	2	4	7.9	15.4	30.4	59.4	114	204.5		
4	1	2	3.9	7.8	15.2	29.4	54.8	96.1	151		
3	1	2	3.8	7.3	12.8	19.9	26.3	30.3	27.9	26.5	26.4
2	1	1.8	2.7	3.3	3.1	3.1	3.1	2.7	2.9	2.9	2.9
1	1	1	0.9	0.9	0.8	0.8	0.7	0.7	0.7	0.6	0.6
	1	2	4	8	16	32	64	128	256	512	1024

Processing Cores

Running Two Jobs Together

System Configuration: 64 Processing Cores

Job DP: 4096-element Dot Product

Job MM: 16 x 16 Matrix Multiply

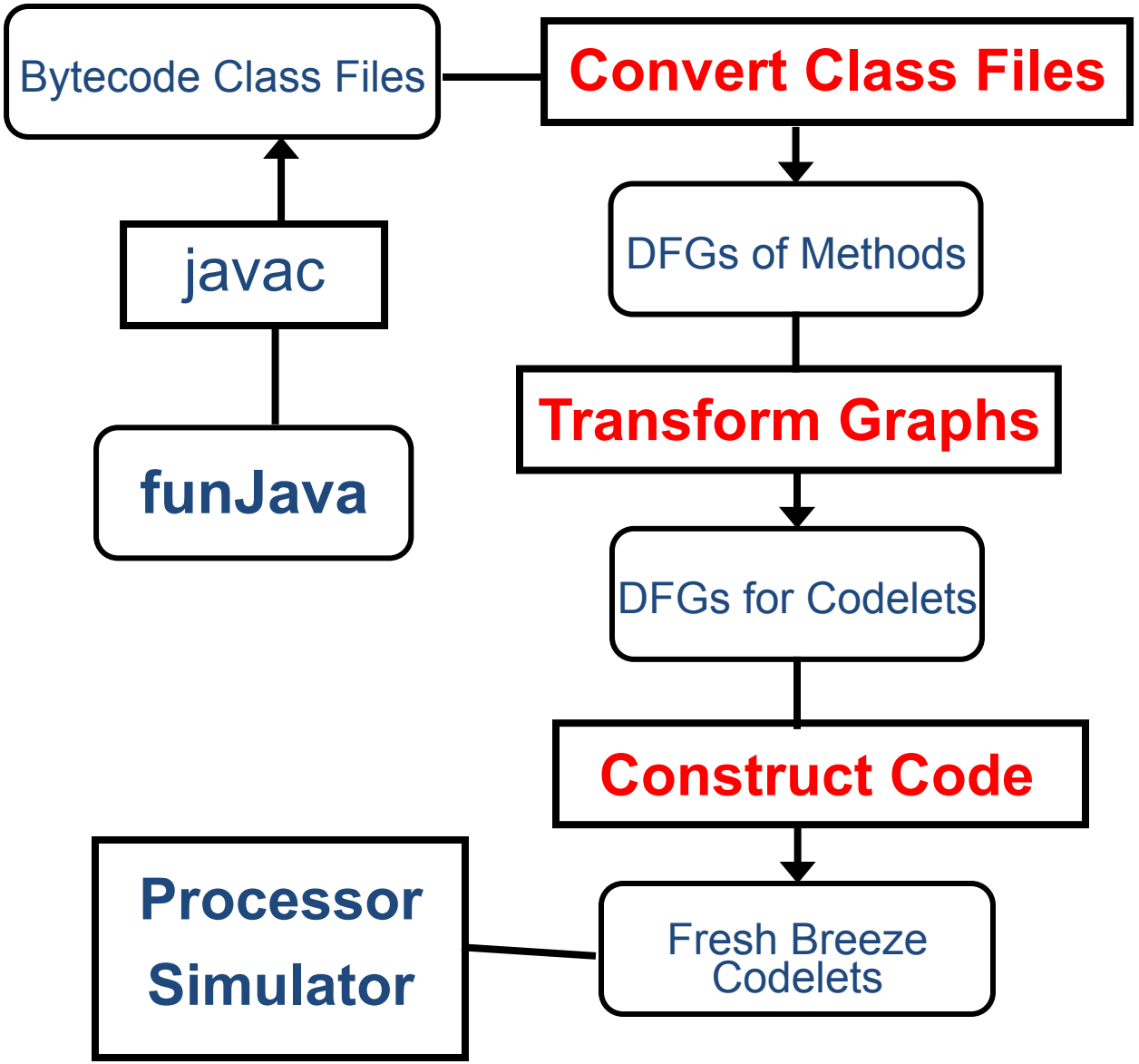
Job	Cycles
DP	10,979
MM	10,409
DP + MM	14,291

Ratio: Together / Separate : 0.67

Sources of Energy Savings

- The AutoBuffer does not use a cache tag memory
- Absence of TLB
- No software cycles for task scheduling
- No software cycles to handle page misses
- No file system software

Fresh Breeze Compiler



Structured Parallelism

Program modules are determinate unless nondeterminate behavior is desired and explicitly introduced by the programmer.

A program execution model must permit parallel execution of two modules whenever there is no data dependence between them, that is, neither module requires any result produced by the other.

Information Hiding Principle

The user of a module must not need to know anything about the internal mechanism of the module to make effective use of it.

Invariant Behavior Principle

The functional behavior of a module must be independent of the site or context from which it is invoked.

Data Generality Principle

The interface to a module must be capable of passing any data object an application may require.

Secure Arguments Principle

The interface to a module must not allow side-effects on arguments supplied to the interface.

Recursive Construction Principle

A program constructed from modules must be useable as a component in building larger programs or modules.

System Resource Management Principle

Resource management for program modules must be performed by the computer system and not by individual program modules.

The list processing language Lisp

Data Objects: Lists – binary trees

Module: Function declaration

Garbage Collection: Yes

Secure Arguments:
Pure Lisp: Yes
Complete Lisp: No

Unified Memory and File System: No

Parallel Execution: Pure Lisp: Yes, with functional behavior.

The IBM AS/400 System

Designed to serve the corporate data processing market.

Data Objects: Files and segments of memory identified by *handles*.

Module: Procedure Declaration

Secure Arguments: Not Known

Unified Memory and File System: Yes

Garbage Collection: Yes