UNIVERSITY of DELAWARE

Architecture and Programming Model for High Performance Interactive Computation

—Based on “Air Force Project—DDDAS”

UD Collaborates with MIT

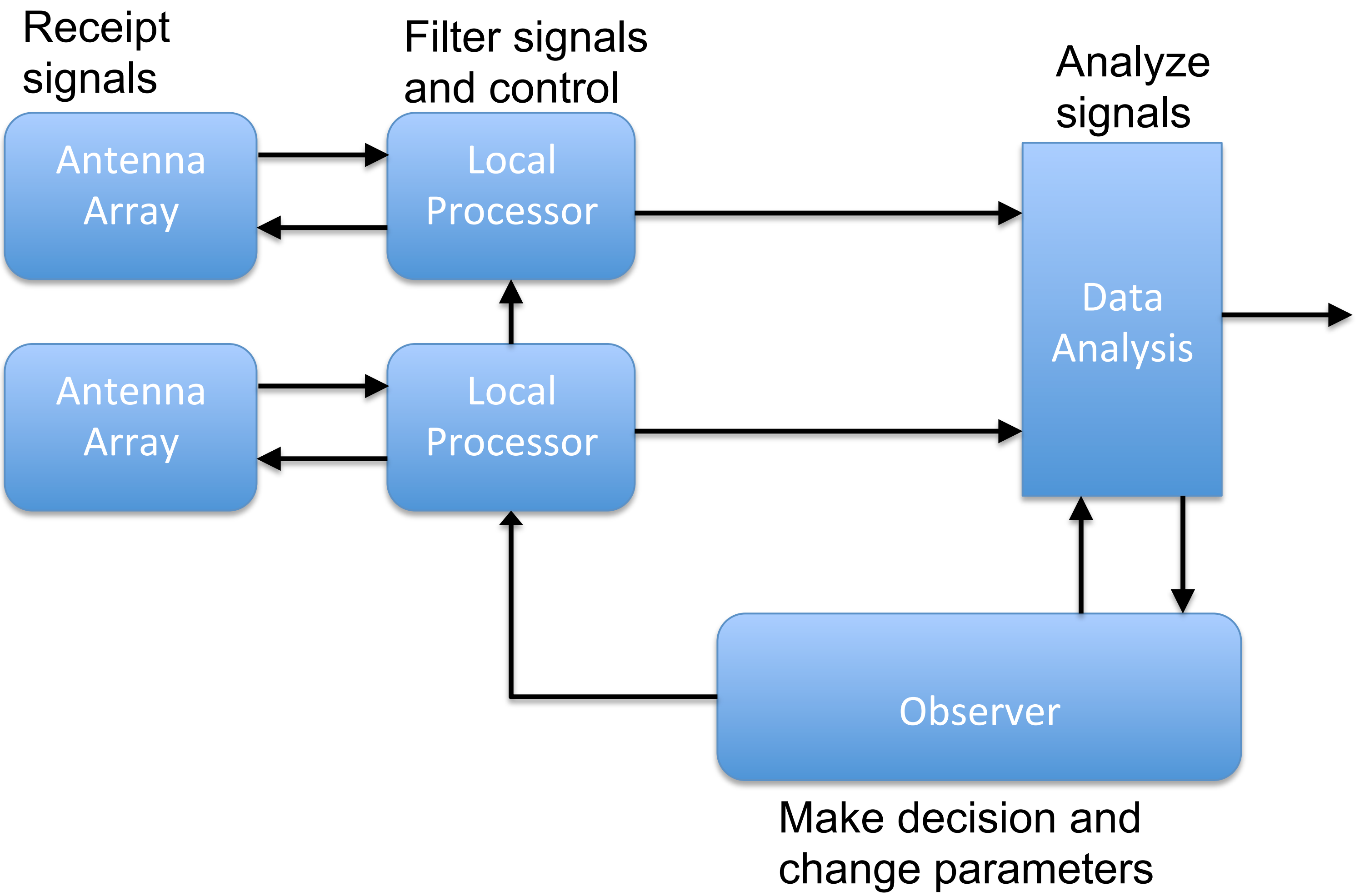
Jack B. Dennis, Arvind, Guang R. Gao , Xiaoming Li and Lian-Ping Wang

Haitao Wei
CAPSL at UDEL

Outline

- Introduction to DDDAS/Interaction Computation
  - An Example and Problems
- Fresh Breeze Execution Model and Architecture
  - Execution Model
  - Memory Model
  - Task Model
  - Architecture
- Compiler Framework for Fresh Breeze
- Streaming and Transactions
  - Stream Type and Operations
  - Concurrency Operations of Transaction Style

An Example of DDDAS/Interaction Computation —Radio Astronomy



Dynamic Data Driven Application System (DDDAS)—Challenges

- real time interaction with parts of the physical environment.
- management of processing and memory resources according to dynamic needs generated by local events
- input and output devices process streams of data items
- make decisions about the work using transaction processing

Our Solutions: Programming Model and Architecture Support

- Fresh Breeze Execution Model and Architecture
  - based on codelet execution model
  - support fine-grained execution and memory management
- Streaming
  - support streaming data expression and operations
- Transaction
  - support concurrency operations of transaction style

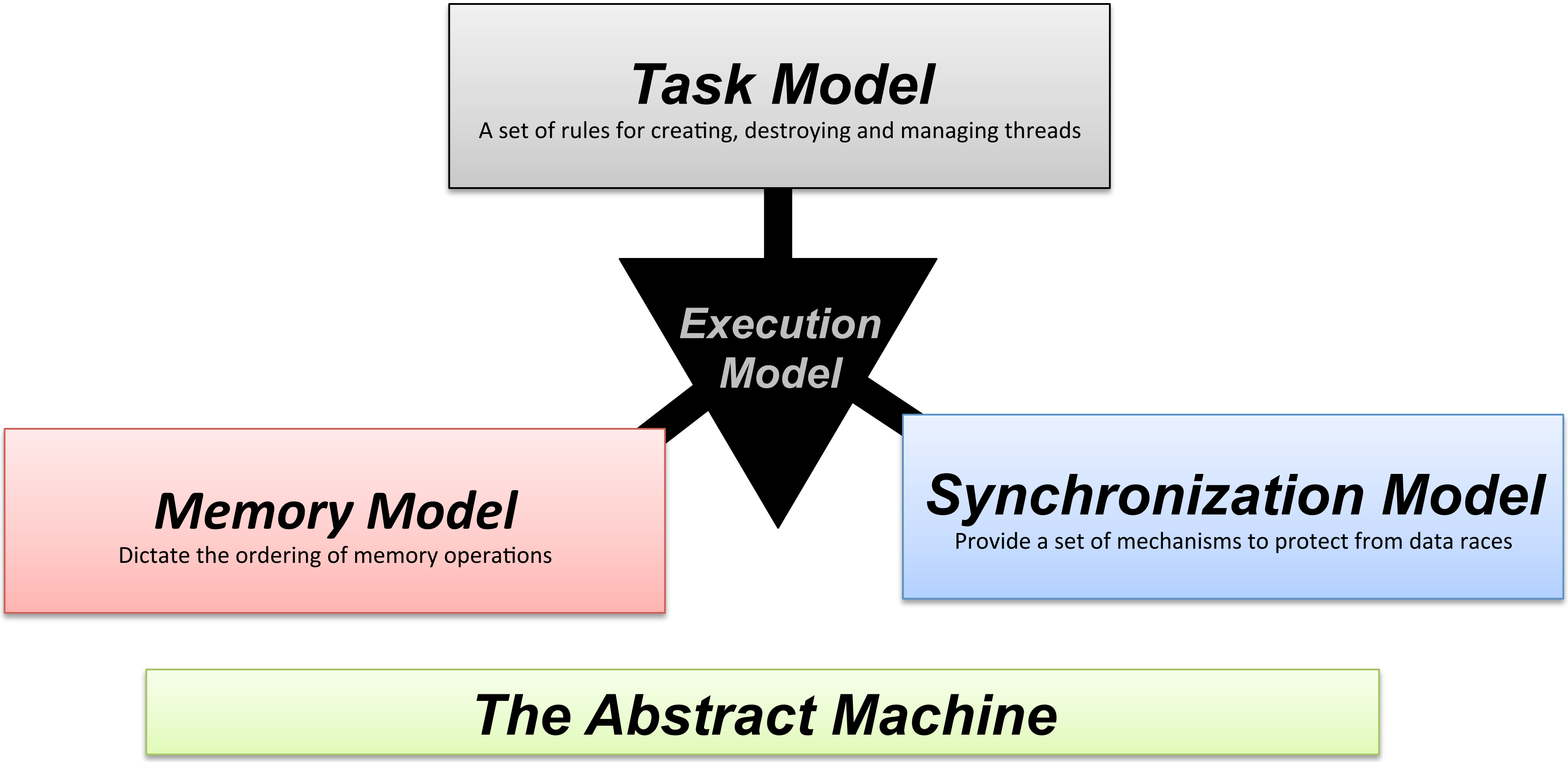
Outline

- Introduction to DDDAS/Interaction Computation
  - An Example and Problems
- Fresh Breeze Execution Model and Architecture
  - Execution Model
  - Memory Model
  - Task Model
  - Architecture
- Compiler Framework for Fresh Breeze
- Streaming and Transactions
  - Stream Type and Operations
  - Concurrency Operations of Transaction Style

Case Studies of Fine-Gran Execution Models

- Dataflow Model (1970s -)
- EARTH Model (1993 -2006)
- HTVM Model (2000 -2010)
- Fresh Breeze Model (2000 -)
- Codelet Model (2010-)

Fresh Breeze Execution Model



Task Model

A set of rules for creating, destroying and managing threads

Memory Model

Dictate the ordering of memory operations

Synchronization Model

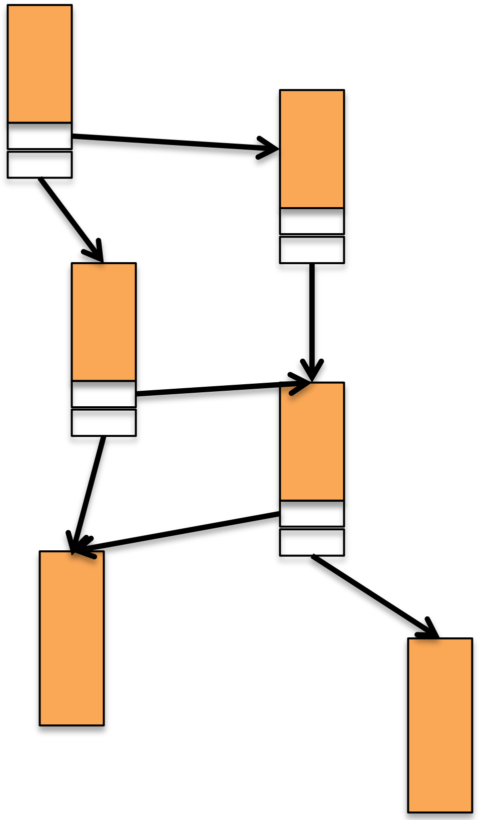
Provide a set of mechanisms to protect from data races

The Abstract Machine

Fresh Breeze Memory Model -- *Main Features and Vision*

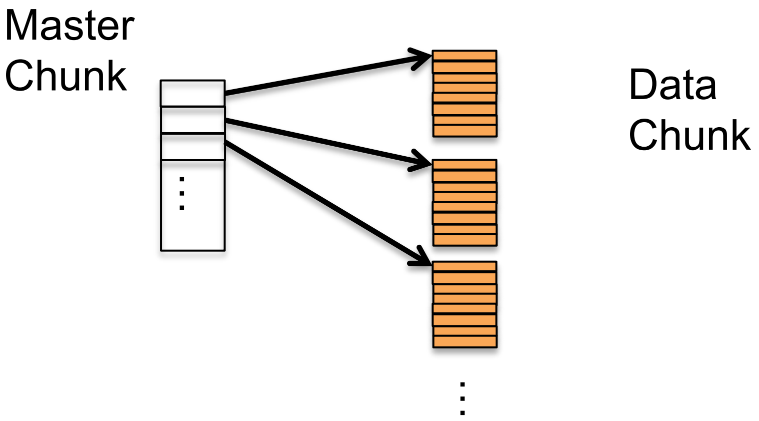
- Global shared name space with "one-level store"
- A *single-update* storage model to eliminate the cache-coherence problem
- Concept of "***sealed***" memory chunks/sections with single assigned property
- Trees of fixed-sized chuncks
- Fine-Grain memory management support
- memory allocation and data transfer is performed entirely by architecture/hardware mechanisms

Fresh Breeze Memory Model



DAG heap

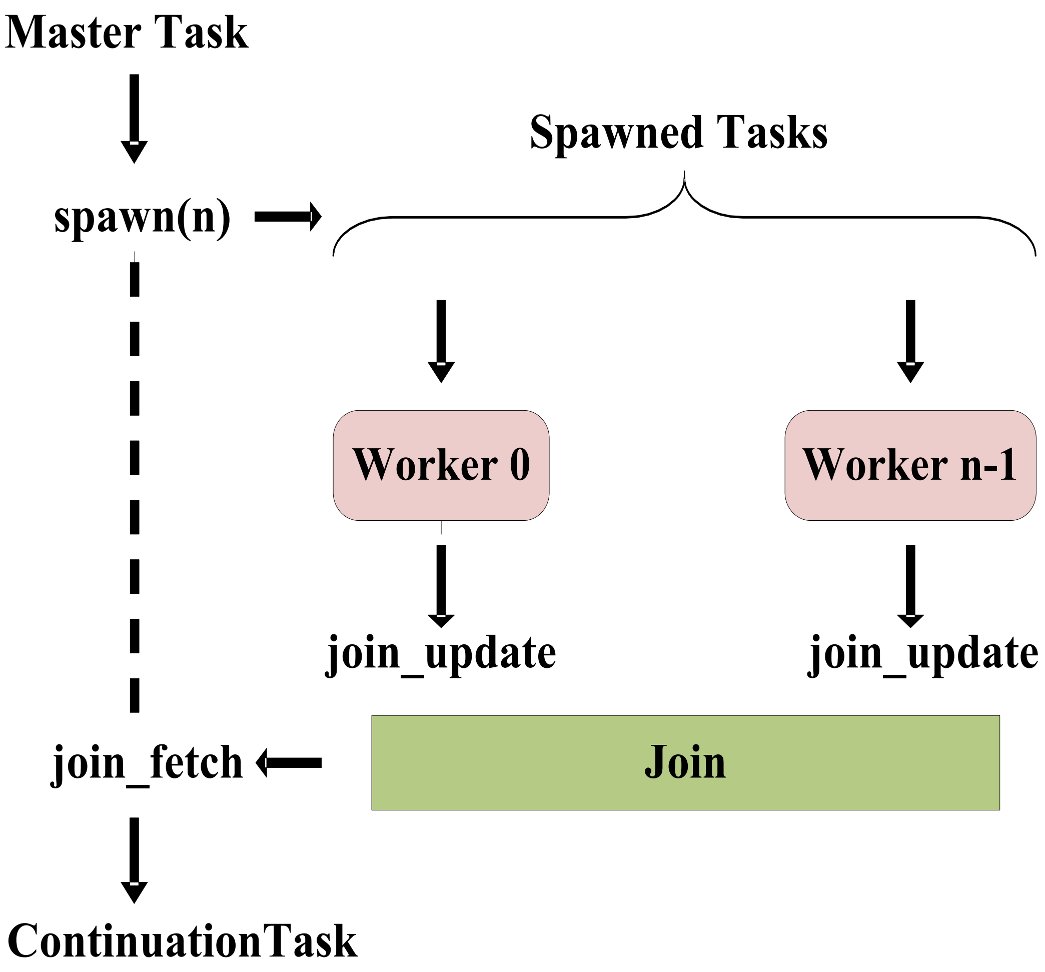
🟧 data item □ chunk handle



Arrays as Trees of Chunks

- Write Once then Read only
- Fix chunk size: 128 Bytes: 16 doubles, 32 integers,…
- Chunk handle: 64 bits unique identifier
- Arrays: Three levels yields 4096 elements(longs)

Task/Concurrency Model

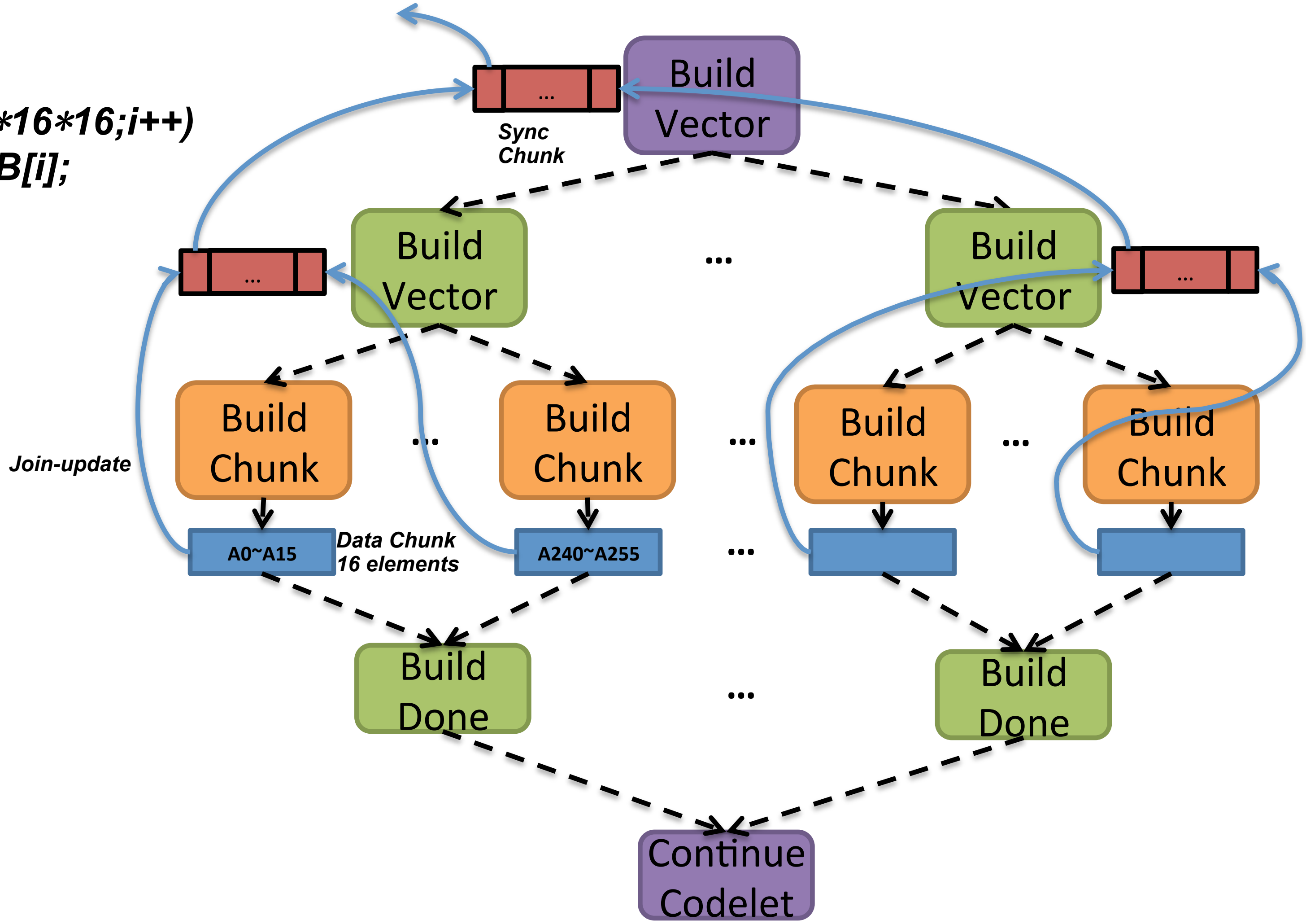


- *Asynchronous tasking*
- *Continuation Task receives children's results*
- *Non-blocking continuation*
- *Light-Weight Tasks*

Example—Dot Product

```
sum=0;
for(i=0;i<16*16*16;i++)
sum+=A[i]*B[i];
```

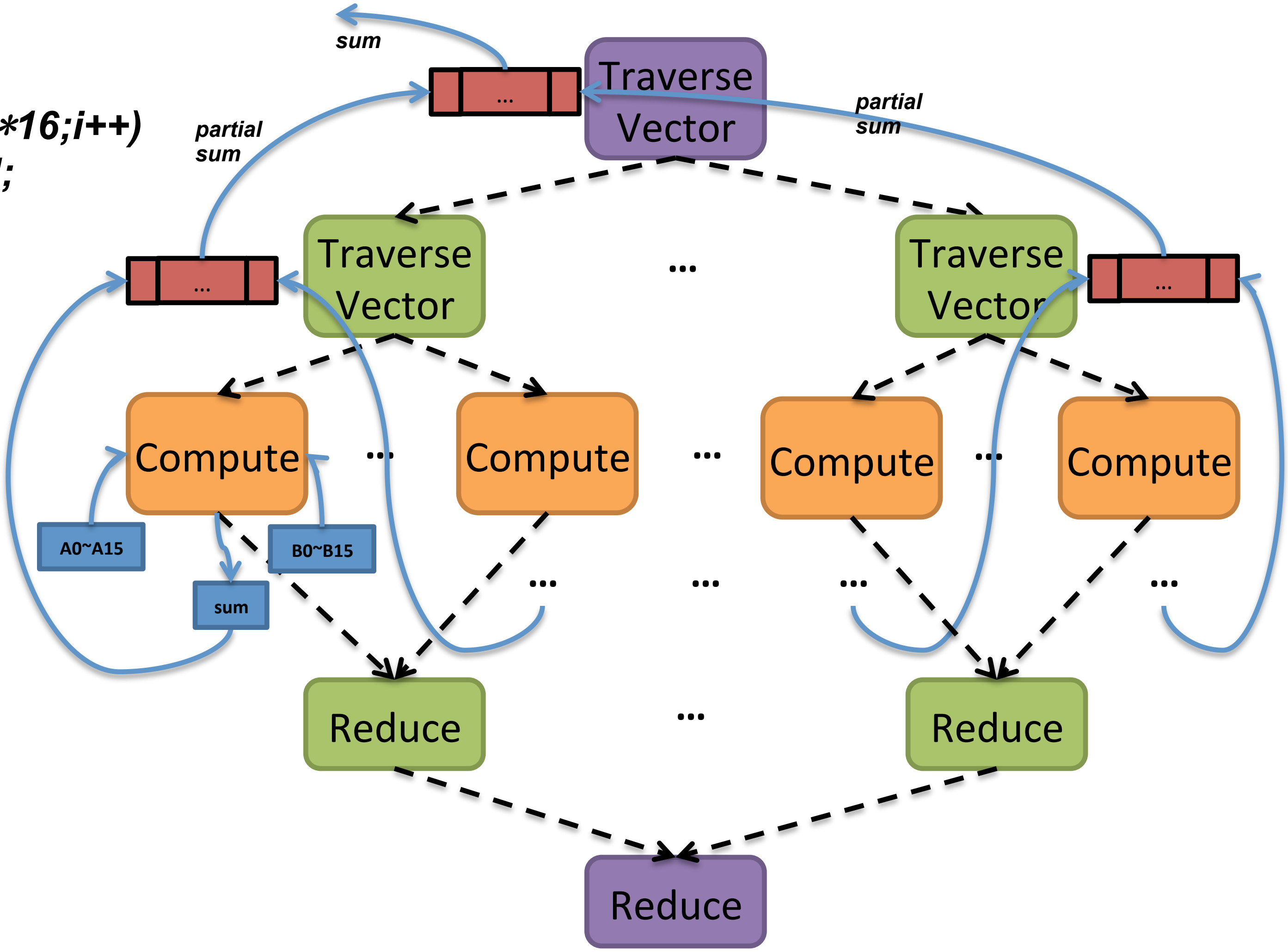
Step1: Build Vector



Example—Dot Product

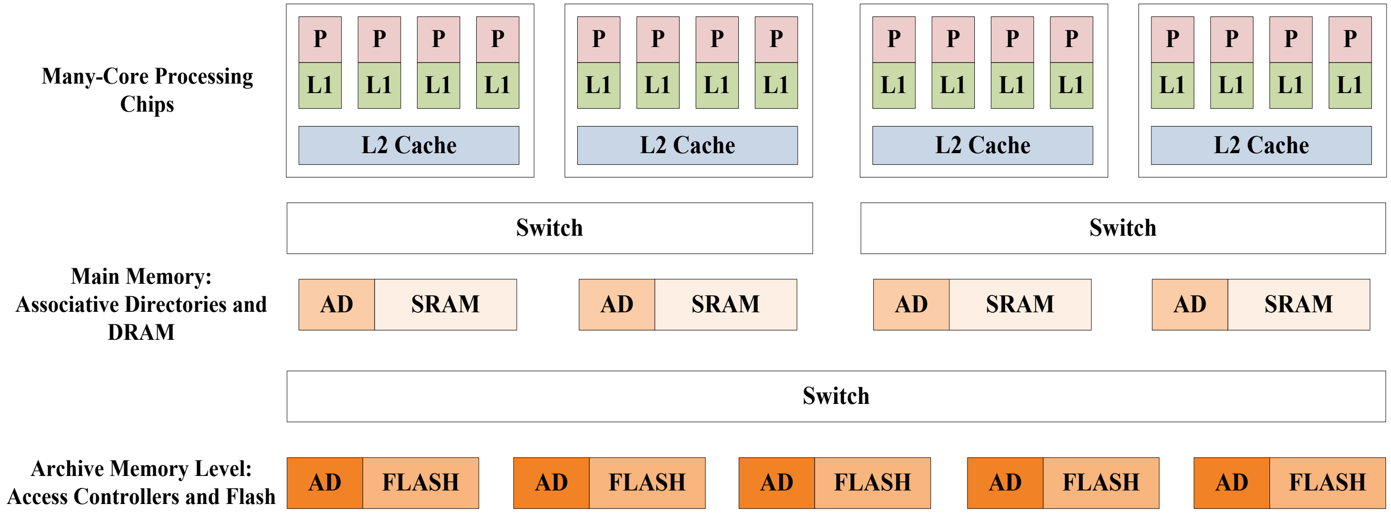
sum=0;
*for(i=0;i<16*16*16;i++)*
*sum+=A[i]*B[i];*

Step2: Compute



Fresh Breeze Architecture

-- a Massively Parallel Computing System

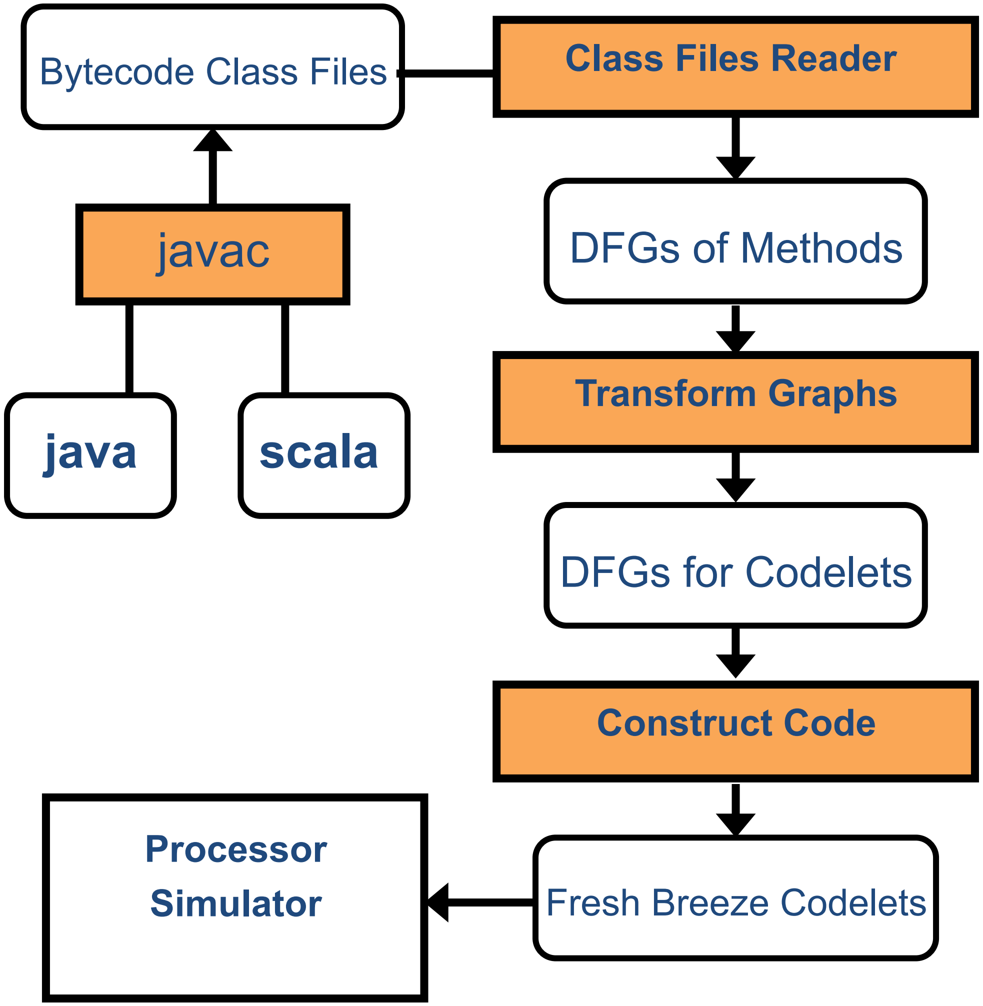

Many-Core Processing Chips
P
L1
L2 Cache
Switch
Main Memory: Associative Directories and DRAM
AD
SRAM
Switch
Archive Memory Level: Access Controllers and Flash
AD
FLASH

- Many-core architecture with shared memory
- Argument Fetching Dataflow Processor Design
- Instruction Scheduler can be Sequential (single thread) or Parallel (multithread)
- The cache memories are organized around chunks
- Memory system maps chunk handle to physical location

Outline

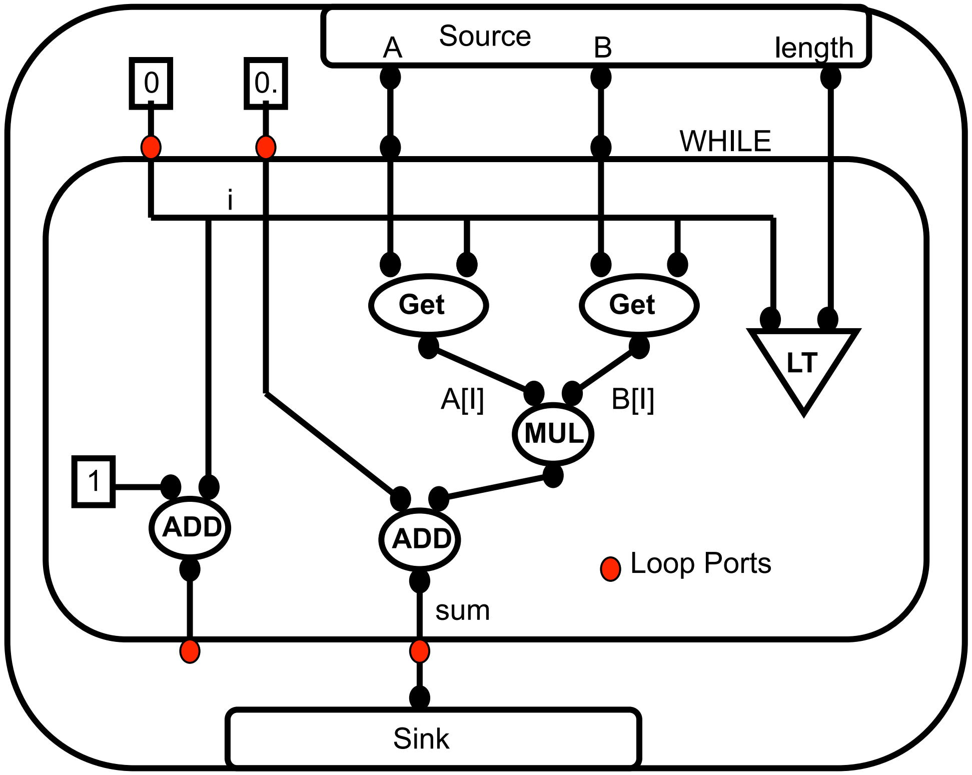
- Introduction to DDDAS/Interaction Computation
  - An Example and Problems
- Fresh Breeze Execution Model and Architecture
  - Execution Model
  - Memory Model
  - Task Model
  - Architecture
- Compiler Framework for Fresh Breeze
- Streaming and Transactions
  - Stream Type and Operations
  - Concurrency Operations of Transaction Style

Fresh Breeze Compiler Framework



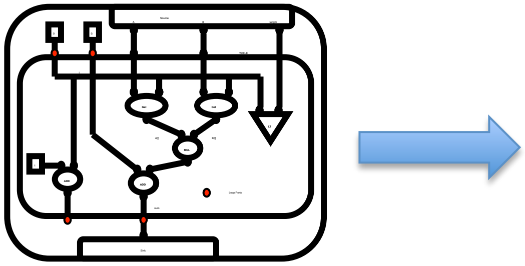
- **Javac** compiles the source code into java byte code
- **Class File Reader** translates bytecode into linear internal representation and constructs data flow graph
- **Transform** identifies the data parallelism, transform it into for all parallel structure **Construct Code** converts each DFG representing a codelet into FreshBreeze ISA

Data Flow Graph for Dot Product

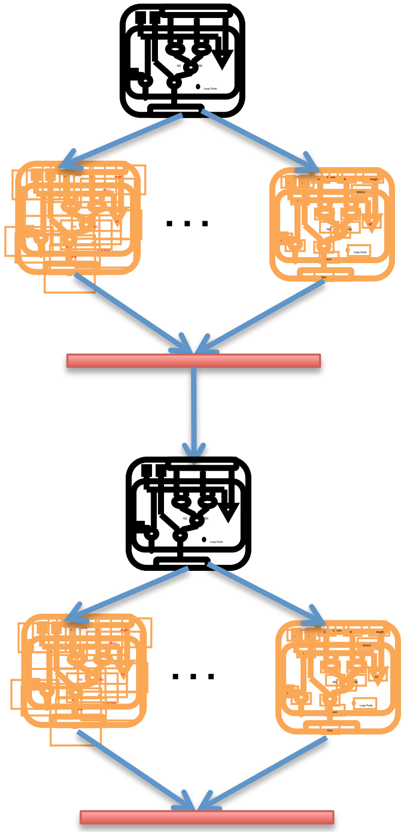


- Intermediate representation in the compiler
- Hierarchical graph structure
- Each structure has source and sink node
- Using ports to connect different components

Transform Component



DFG for a loop in one codelet

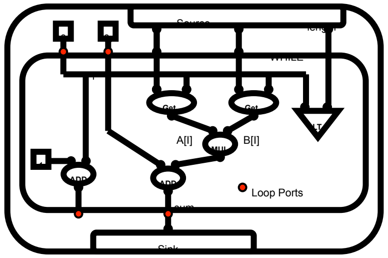


DFG for parallelized multiple codelets

- Analyze the loop to extract the data parallelism
- Create codelets to construct the chunk tree for the data representation
- Create codelets to traverse the tree and compute using fork-join parallel pattern

Code Generation

DFG of one codelet



```
0]:  ISet LV: 0; -> D: 8
1]:  ISet LV: 1; -> D: 9
2]:  LSet LV: 0; -> D: 10
3]:  IMove S0: 8; -> D: 12
4]:  LMove S0: 10; -> D: 14
5]:  IfILeq S0: 12; S1: 3; Lab: 12
6]:  LoadFull H: 4; Off: 12; -> D: 16
7]:  LoadFull H: 6; Off: 12; -> D: 18
8]:  LMul S0: 16; S1: 18; -> D: 16
9]:  LAdd S0: 14; S1: 16; -> D: 14
10]: IAdd S0: 12; S1: 9; -> D: 12
11]: Jump Lab: 5
12]: SyncUpdate Sync: 0; Off: 2; Data: 14
13]: TaskQuit
```

Instruction of FreshBreeze codelet

- Build Attribute Tree： notate constant node, literal operands ect.
- Perform Variable Assignment: similar to register allocation
- Build Codelet: convert each dataflow node into instructions

Outline

- Introduction to DDDAS/Interaction Computation
  - An Example and Problems
- Fresh Breeze Execution Model and Architecture
  - Execution Model
  - Memory Model
  - Task Model
  - Architecture
- Compiler Framework for Fresh Breeze
- Streaming and Transactions
  - Stream Type and Operations
  - Concurrency Operations of Transaction Style

Stream Type and Operations

- Stream: A sequence of values of type, maybe infinite

- Define a stream
  - Stream <DataItem> inStream = **new Stream <DataItem>**();
    DateItem can be any data type

- Concatenate two streams
  - Stream <DataItem> strm1 =
    strm0 **+** new Stream <DataItem>{i0, i1, ... }

- Get first element in stream
  - strm.**first**();

Stream Type and Operations (cont'd)

- Remove the first element in stream
  - Stream <DataItem> strm1 = strm0.**rest** ()
  - Stream <DataItem> strm = strm.**first** () + strm.**rest** ()

- Append an data item to stream
  - strm.**append**(item) ;

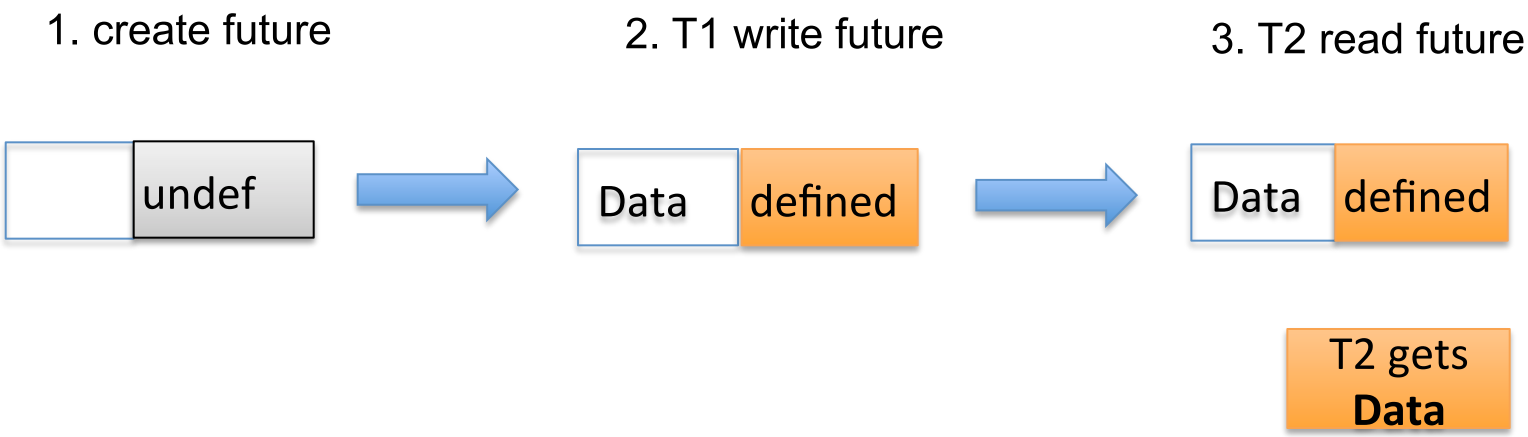
- It is the end of data stream
  - if (strm.**moreData** ()) { statement }

Stream Implementation in FreshBreeze

- Stream representation
  - a linear chain of chunks, each chunk holds data items and a reference to the next chunk
- Stream operations
  - FIFO queue operations on chain of chunks
  - read from the head of the chain of chunks, write to the tail of the chain of chunks
- Synchronization between Producer and Consumer
  - Special Object: ***Future***

Future

- A future is a memory cell with a state waiting to receive a data value：status: undefined, defined, waiting
- Future Read and Future Write are Atomic



1. create future

2. T1 write future

3. T2 read future

undef

Data defined

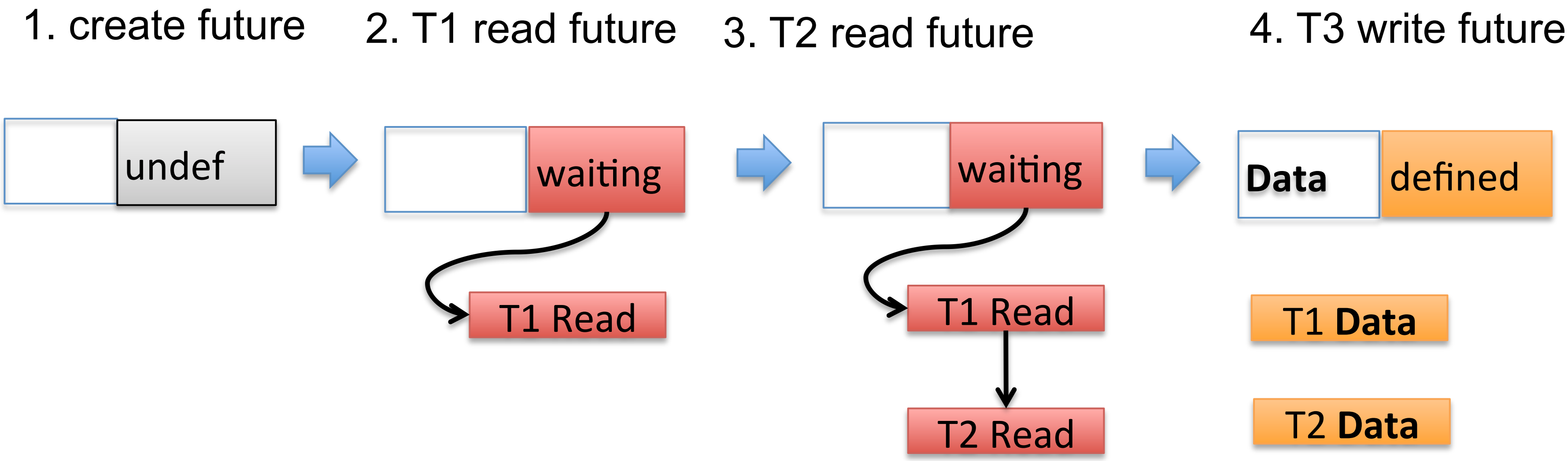
Data defined

T2 gets **Data**

Read After Write

Future (Cont'd)

- A future is a memory cell with a state waiting to receive a data value：status: undefined, defined, waiting
- Future Read and Future Write are Atomic



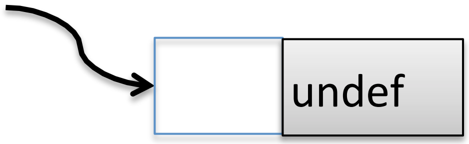
1. create future
2. T1 read future
3. T2 read future
4. T3 write future

Write After Read

Stream Operation Based on Future

- Fresh Breeze Instruction Set Support 4 stream operations
  - New, Append, First and Rest

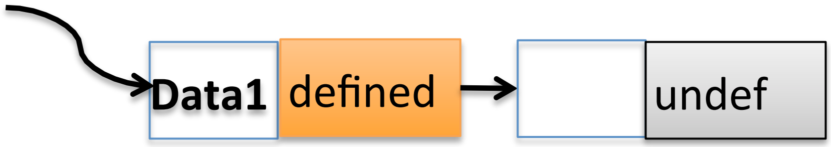
1. new stream



Stream Operation Based on Future

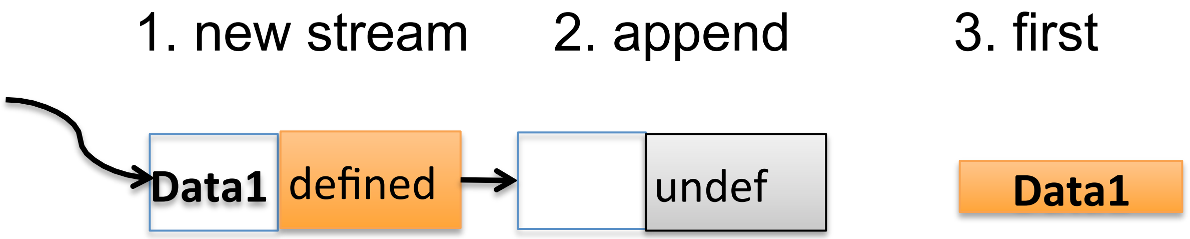
- Fresh Breeze Instruction Set Support 4 stream operations
  - New, Append, First and Rest

1. new stream 2. append



Stream Operation Based on Future

- Fresh Breeze Instruction Set Support 4 stream operations
  - New, Append, First and Rest



1. new stream
2. append
3. first

Data1 defined

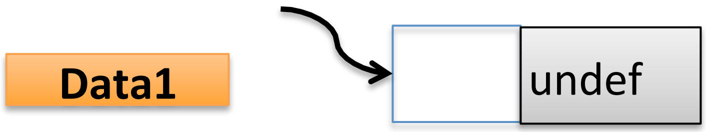
undef

Data1

Stream Operation Based on Future

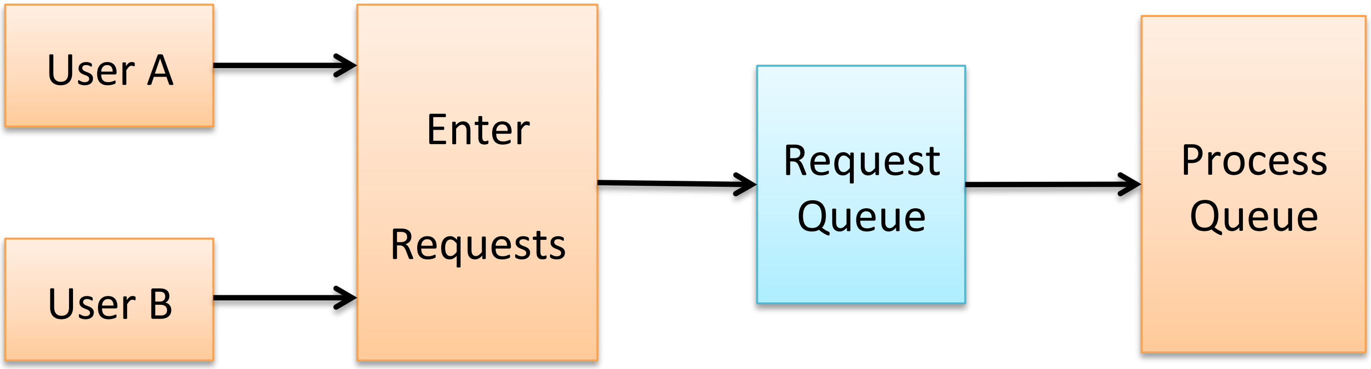
- Fresh Breeze Instruction Set Support 4 stream operations
  - New, Append, First and Rest

1. new stream 2. append 3. first 4. rest



Concurrent Transactions

- Scenario: A Simple Shared Hash Table
  - Shared by two concurrent users. Either user may search the value corresponding to a key, and either user may add or delete entries
  - Using concurrent shared queue

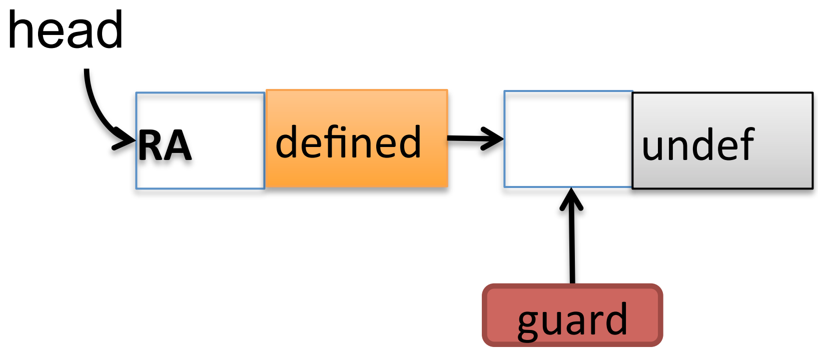


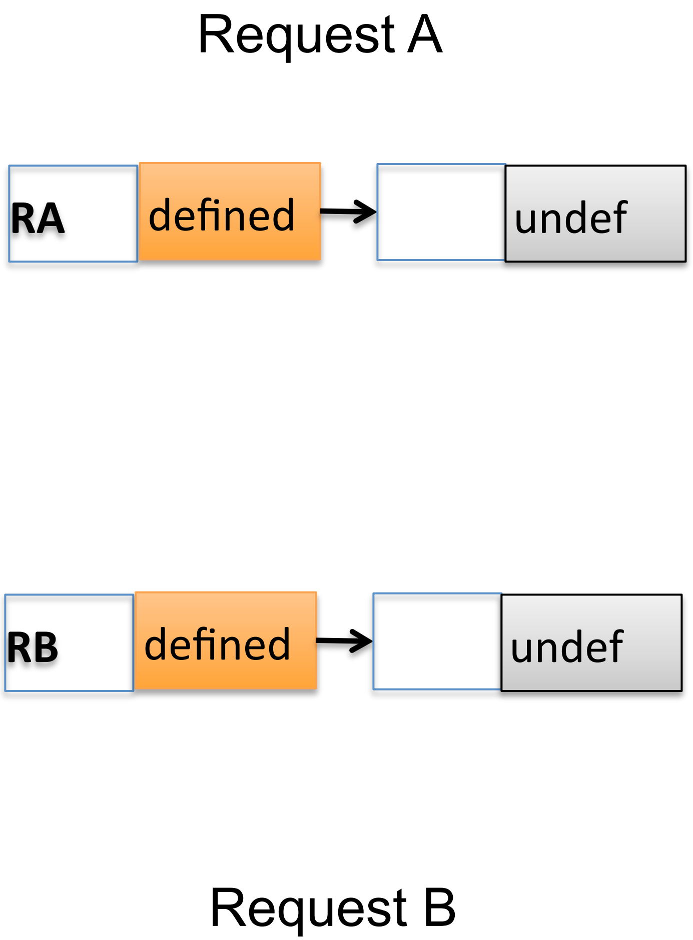
Support Transaction Using Guard In FreshBreeze

- **Guard object**
  - special data object which can only be accessed by **GuardSwap** instruction
- **GuardSwap**
  - atomic instruction
  - put the new data object into guard, and return the old data object in guard
- **For the Concurrent Request Example**
  - using a guard to "lock" the tail of the queue
  - each request needs to get the guard before be added to the tail of the queue

Concurrent Requests

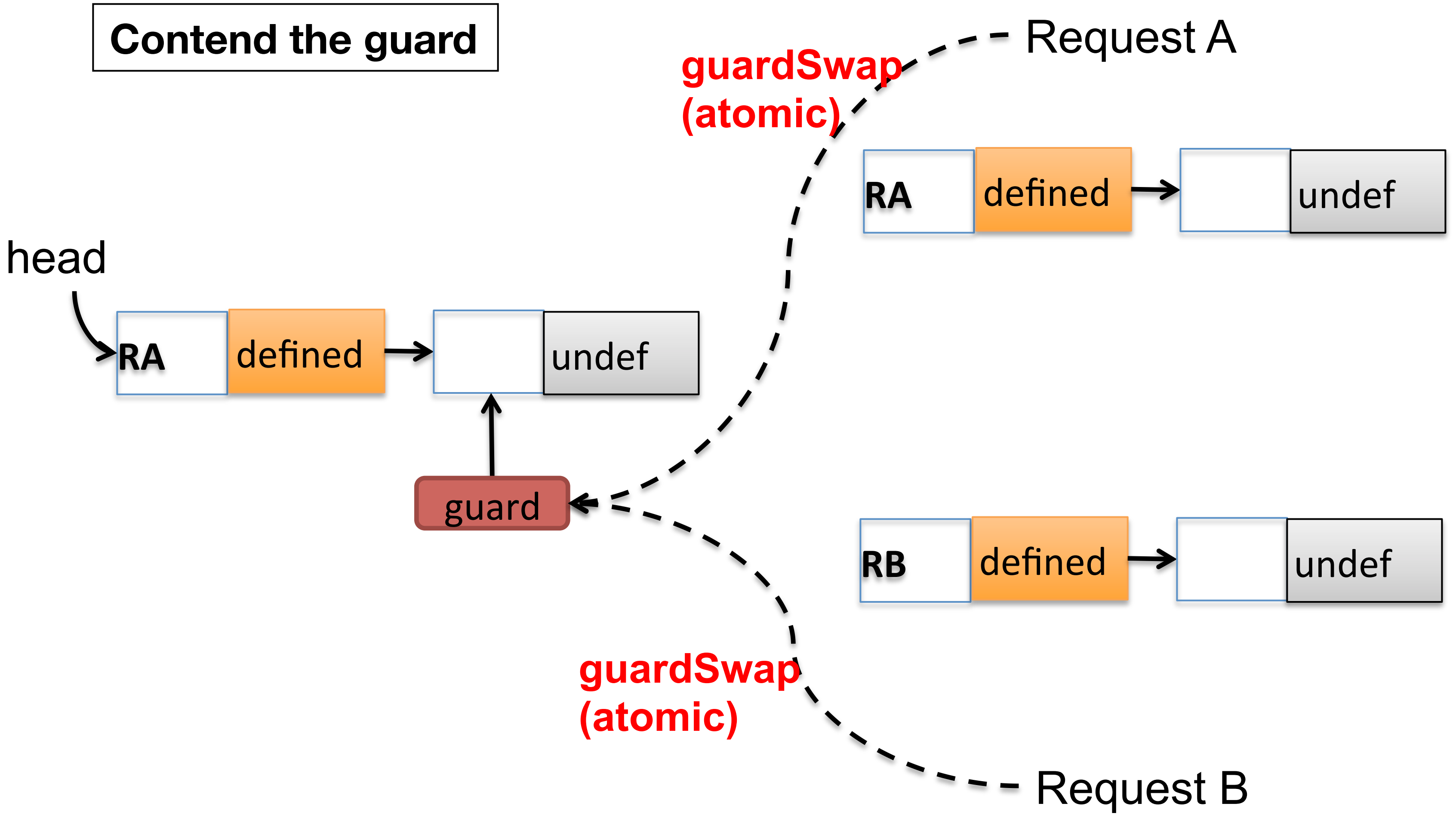
Two requests arrive





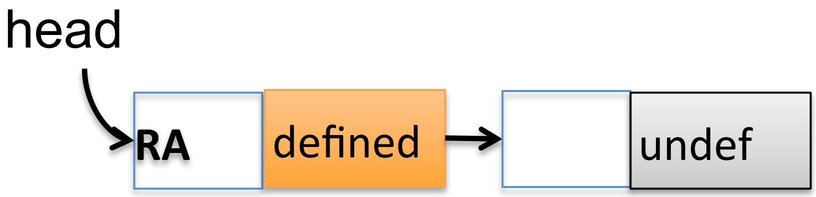
Concurrent Requests

Contend the guard

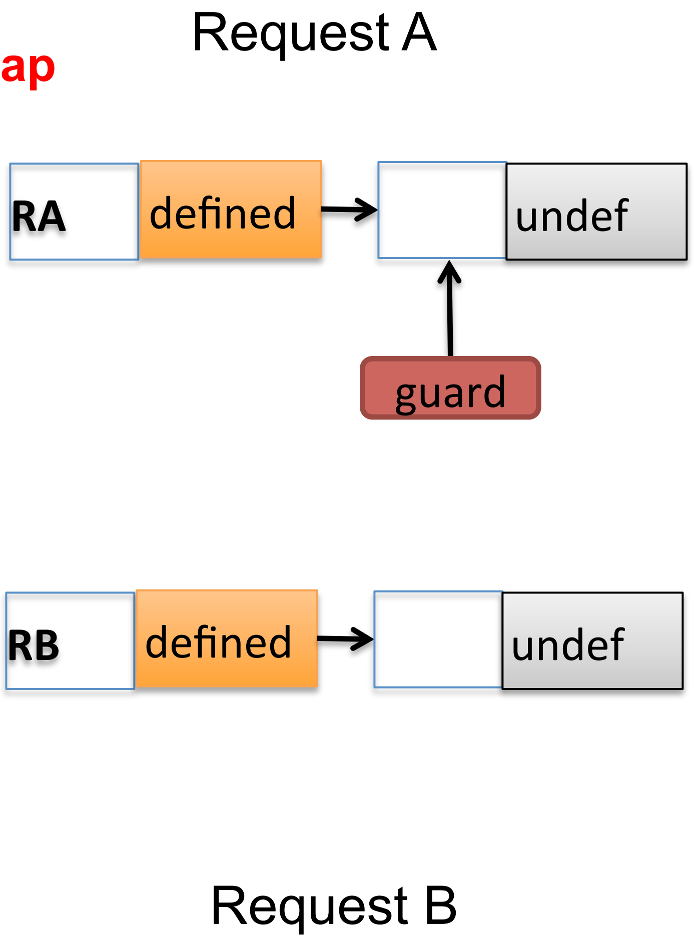


Concurrent Requests

Request A gets the guard and old tail

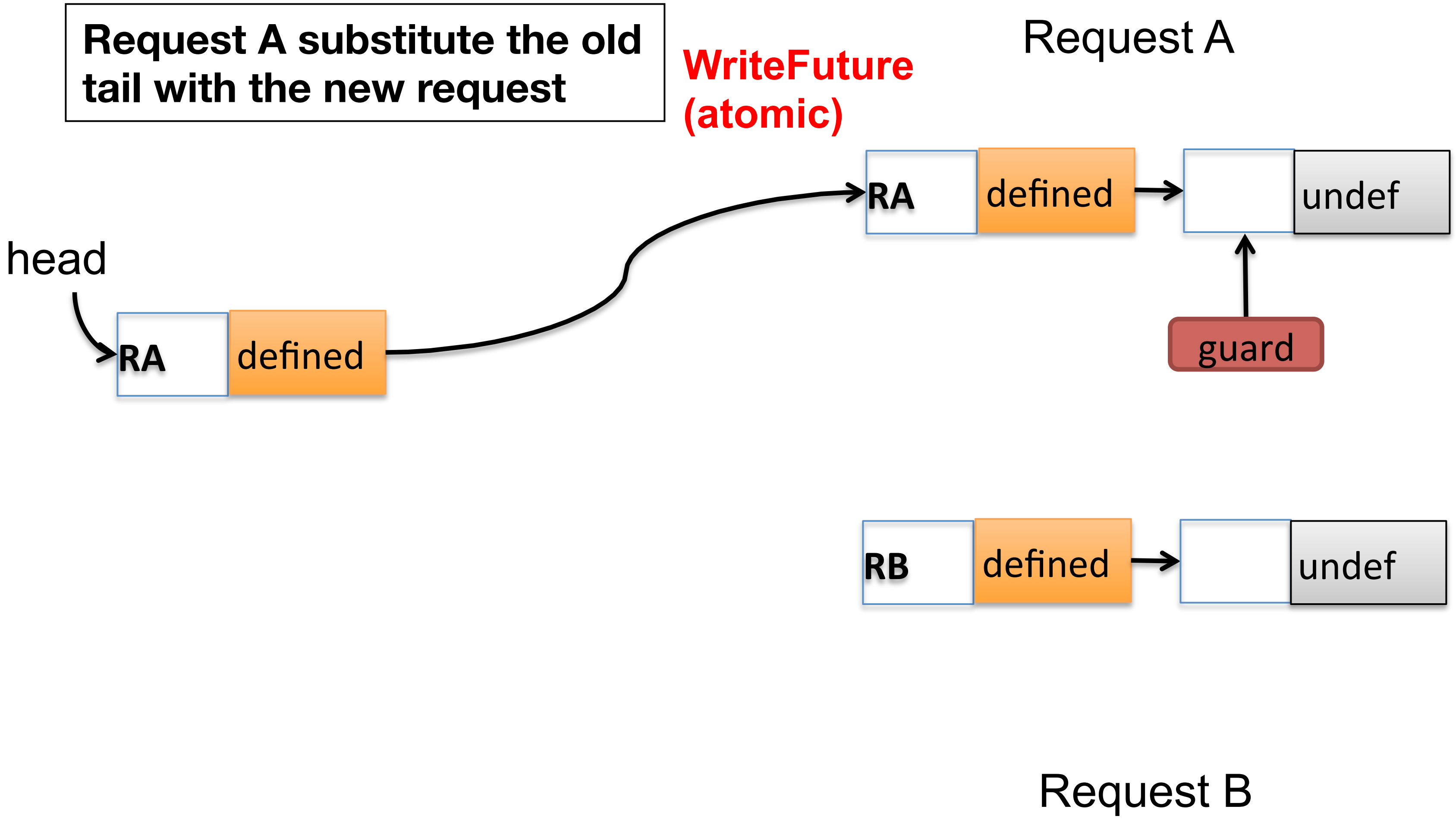


guardSwap (atomic)



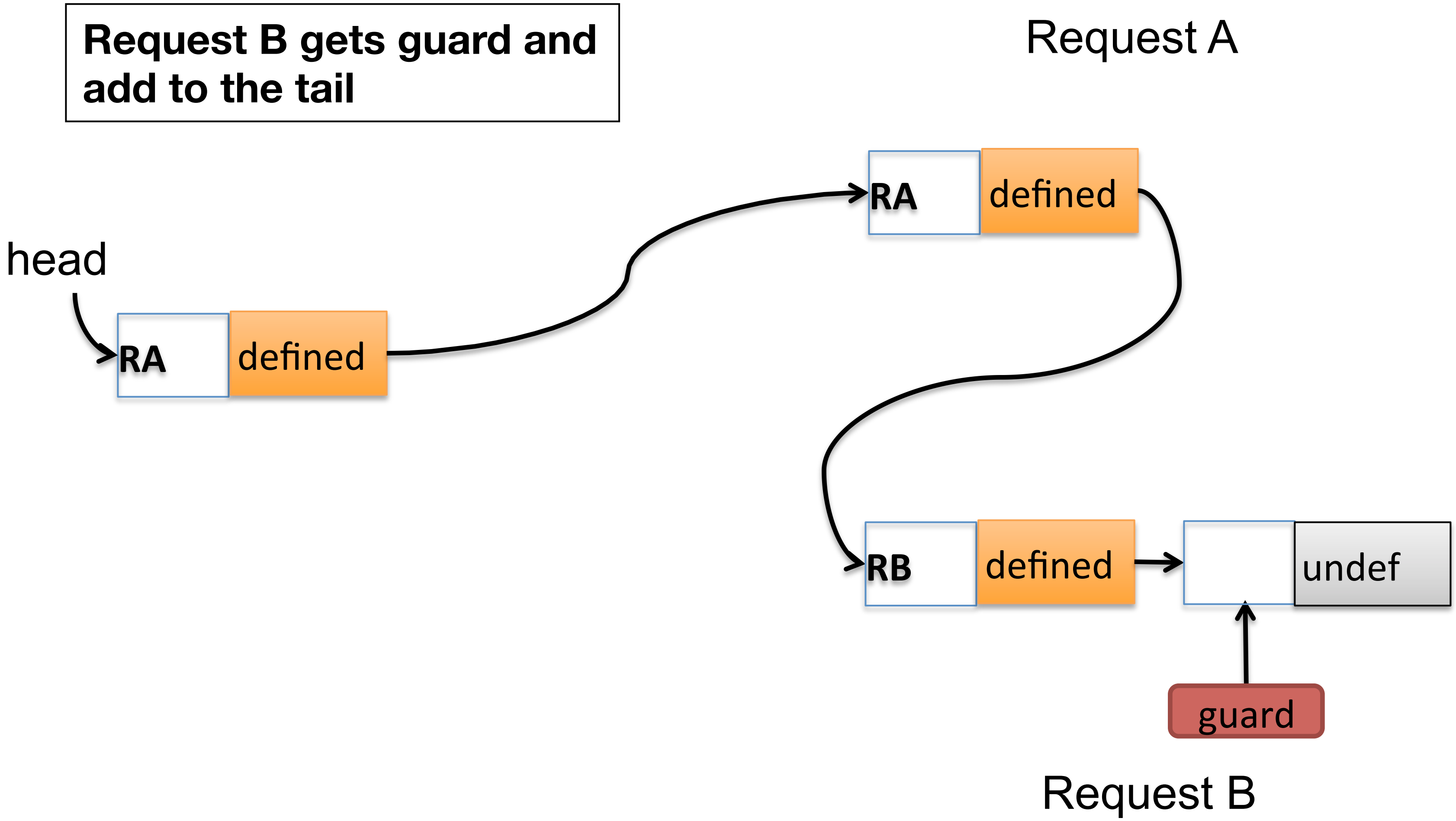
Concurrent Requests

Request A substitute the old tail with the new request



Concurrent Requests

Request B gets guard and add to the tail



Project Status and Future Work

- Project Status
  - SystemOne, the simulator of FreshBreeze with one core.
  - Compiler framework which can handle perfect loop transformation
- Future Work
  - SystemTwo is under developing, simulator with multi-core
  - Compiler framework is under developing which tries to handle nested loops and complicated loops
  - Stream and Transaction
  - ISA improvements, for now only support integer
  - New benchmarks

…

Acknowledgement

MIT: Prof. Jack Dennis, Prof. Arvind

UDEL: Prof. GuangR. Gao, Prof. Xiaoming Li and Prof. Lian-Ping Wang

Students who worked and is working on the project : Xiaoxuan Meng, Tom St. John, Yao Wu, Chao Yang

And all CAPSL members who helped…