# Distributed Vector Architecture:

# Beyond a Single Vector-IRAM

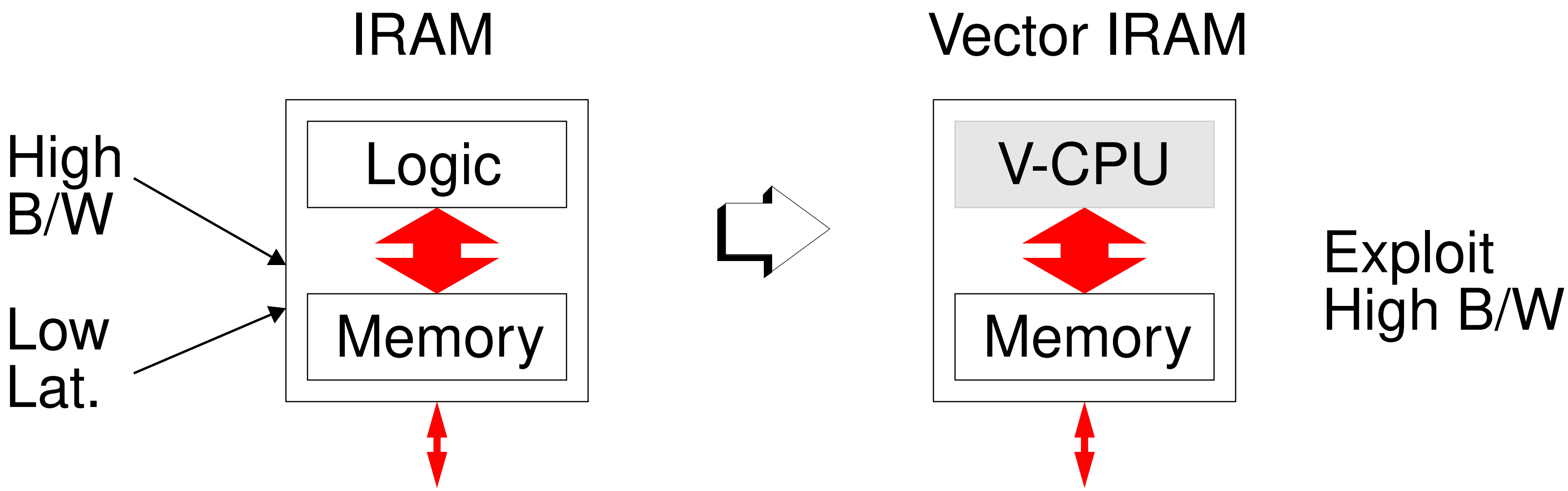
Stefanos Kaxiras[†], Rabin Sugumar[‡],

James Schwartzmeier[‡]

[†]Department of Computer Sciences
University of Wisconsin-Madison

[‡]CRAY RESEARCH A Silicon Graphics Company

# Vector Processing on IRAM

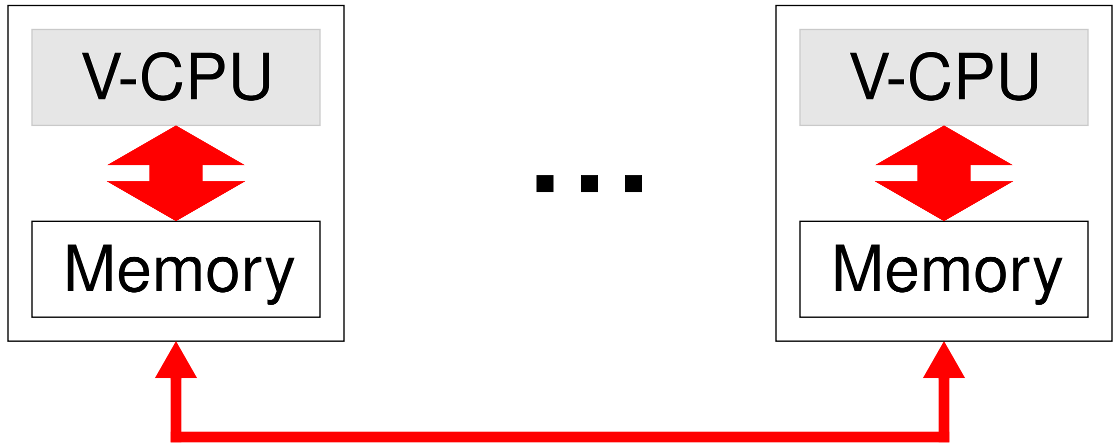



**Focus: Large scientific vector applications**

**Problem: Don't fit on a single IRAM → Thrash**


Department of Computer Sciences
University of Wisconsin-Madison

# DIstributed Vector Architecture (DIVA)




**Use multiple IRAMs**

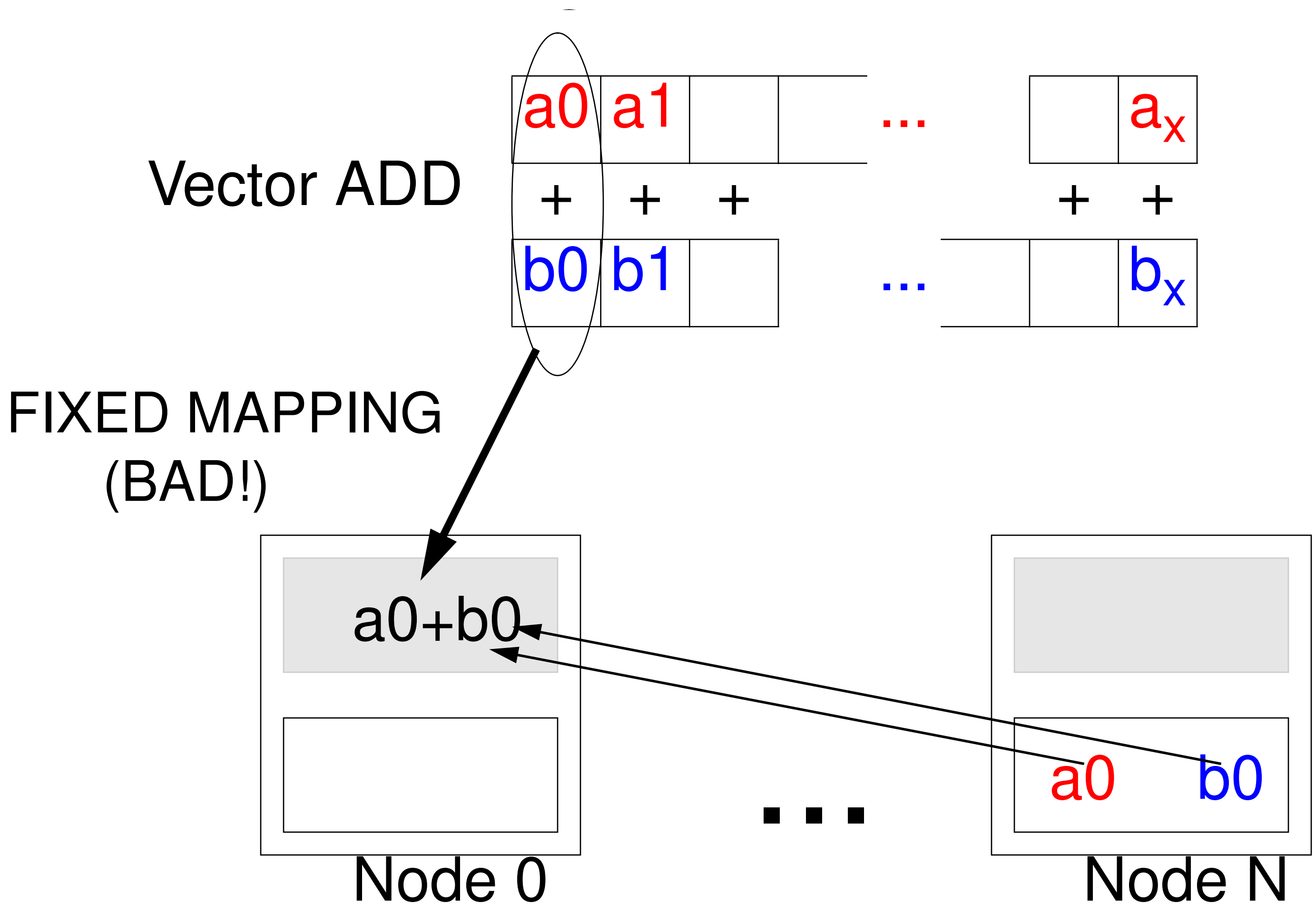
- Statically distribute dataset
- Parallelize execution of vector instructions

**Bottleneck: External traffic**



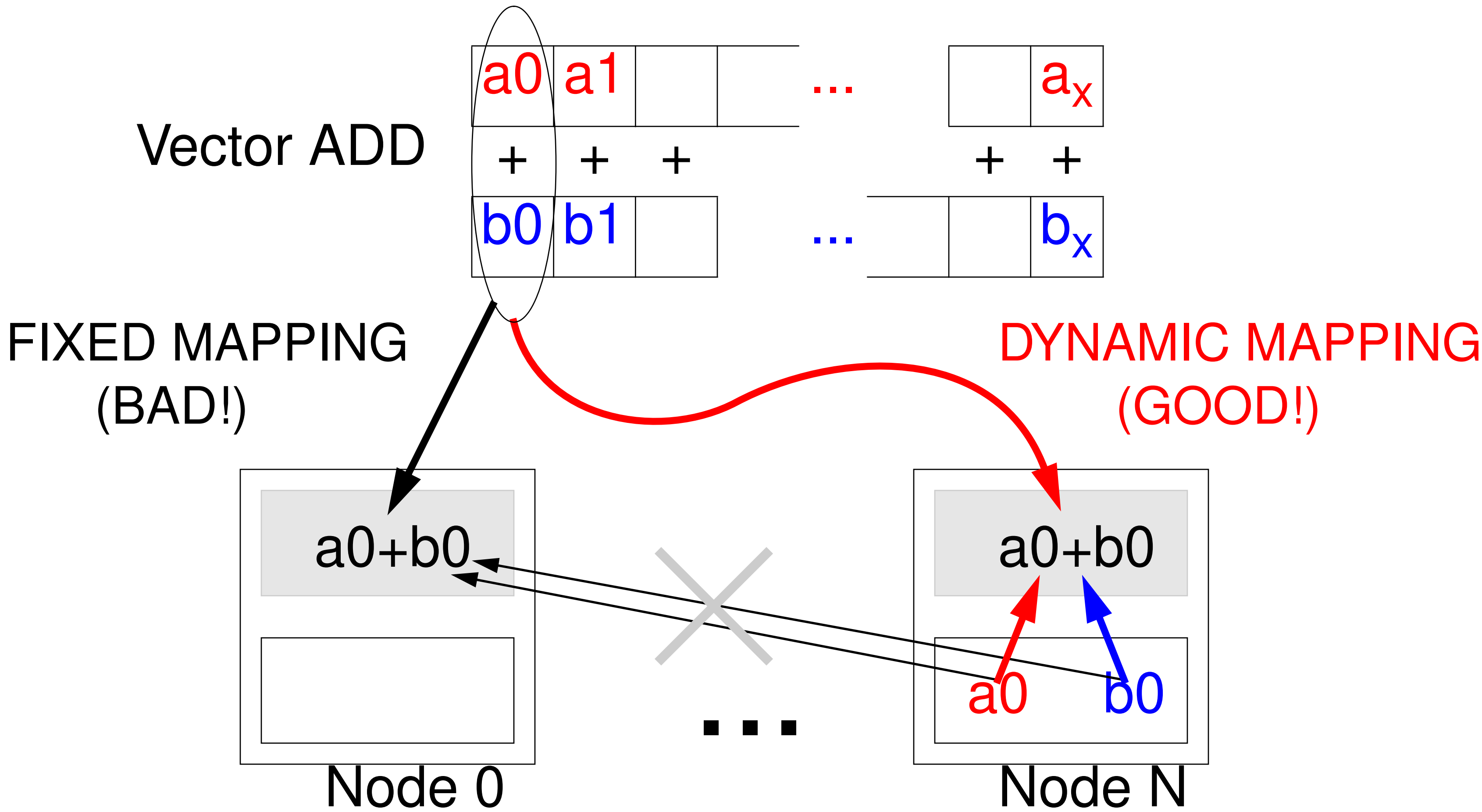

Department of Computer Sciences
University of Wisconsin-Madison

# Computation Mapping



# Key Contribution

## Dynamic computation mapping



# Outline

~~Overview~~

**DIVA**

- Execution Model
- Dynamic Computation Mapping
- SETMV & Mapping Vector Registers

**Evaluation**

**Summary**

# Characteristics of DIVA System

## Hardware

- Commodity V-IRAM (software DIVA)
- Modified Commodity V-IRAM (hardware DIVA)

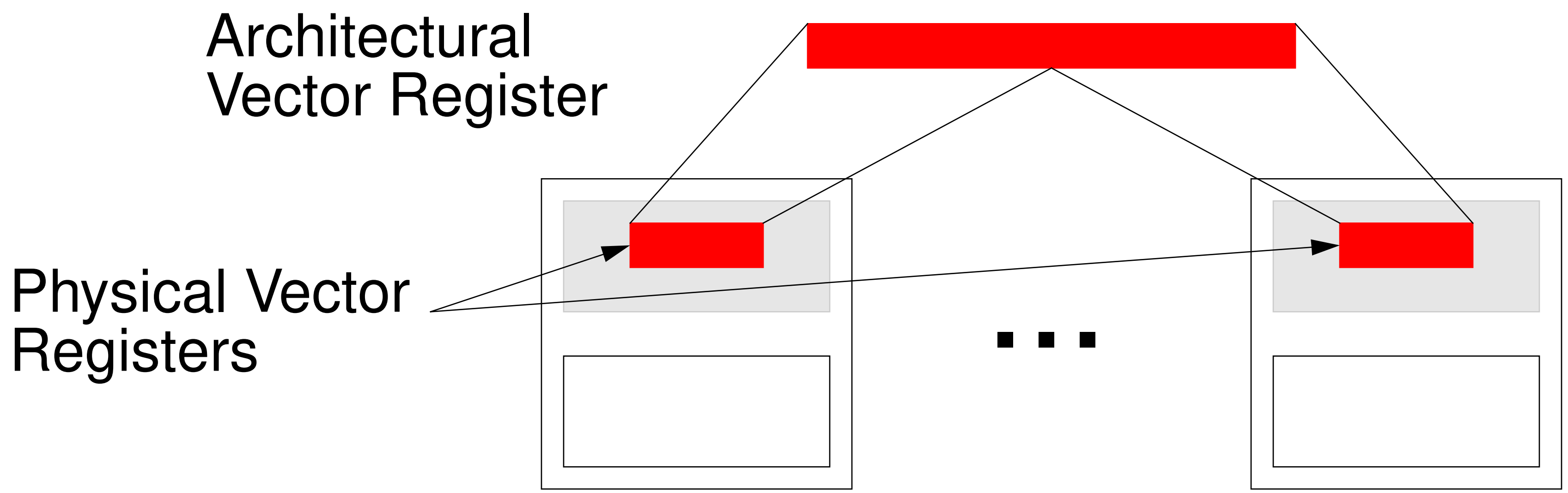
## Programming model

- Same as single node
- Transparent parallelization

## Execution model

- All nodes execute the same application
- Scalar inst. can execute redundantly (MMM/DataScalar)
- Vector inst. in parallel

# Execution Model



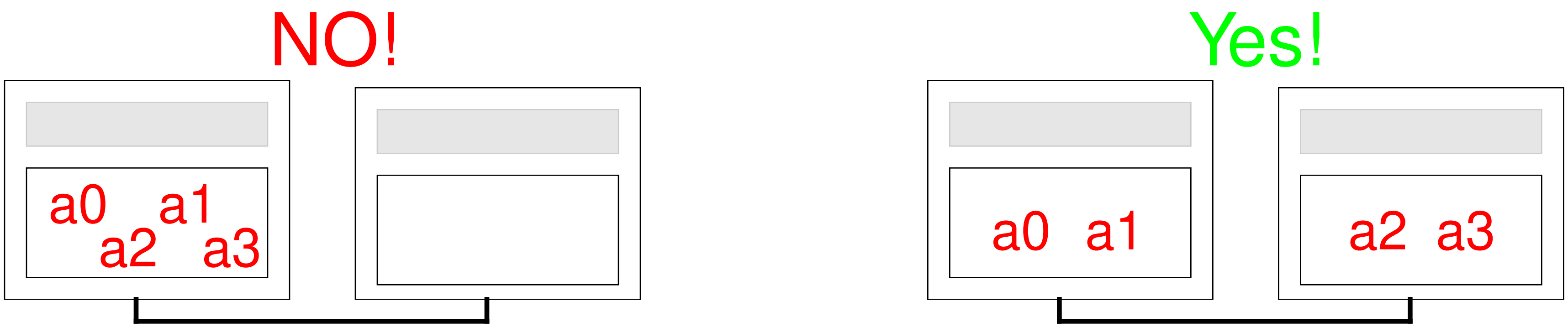

**Transparent parallelization of vector instructions:**

- Vector instructions refer to Architectural Vector Registers
- Nodes compute on Physical Vector registers


Department of Computer Sciences
University of Wisconsin-Madison

# Dataset Distribution

**Parallelize accesses: spread memory vectors across nodes**



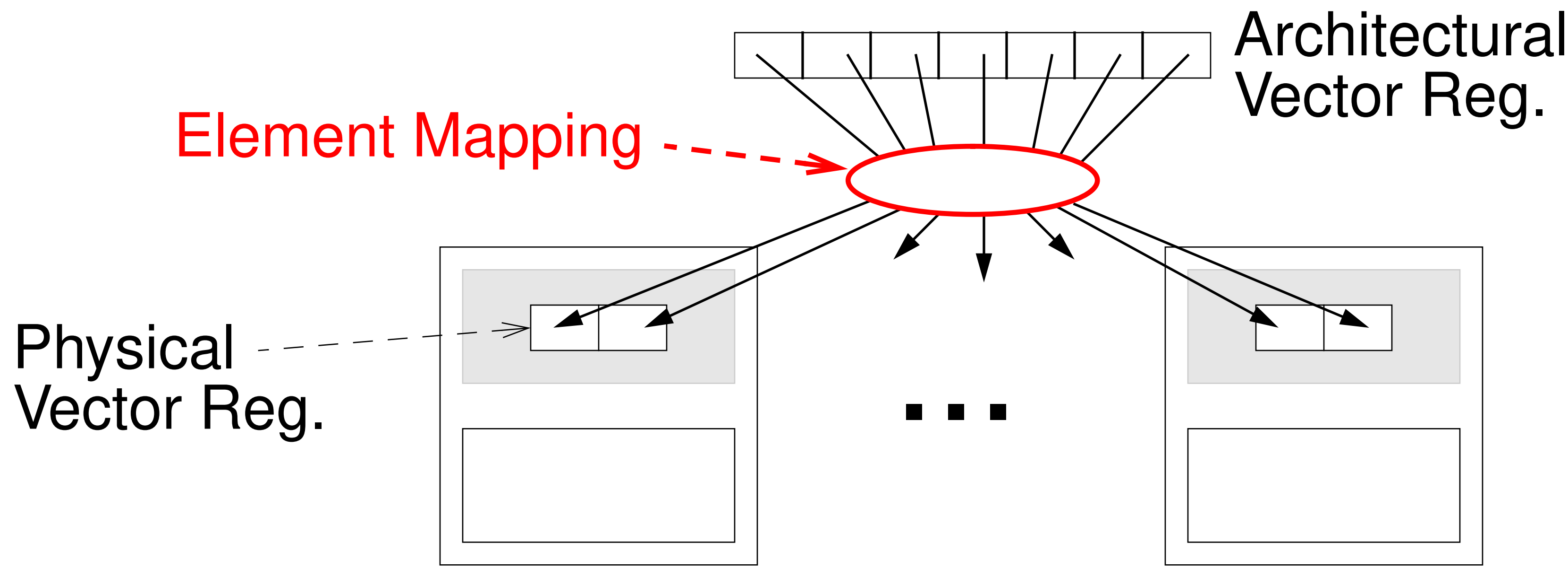

**How? Static partitioning of the dataset among nodes**

- Current method:

  Block-interleaving of memory across nodes
- Potential:

  Sophisticated data placement

# Computation Mapping




**Element mapping = Computation mapping**

- Renaming architectural to physical elements

**Dynamically create element mappings**

# Computation Mapping (cont.)

**WHEN?**

At beginning of every computation slice:

Create element mapping
Load Registers
Compute
Store
Registers are dead

Single mapping for all registers of slice

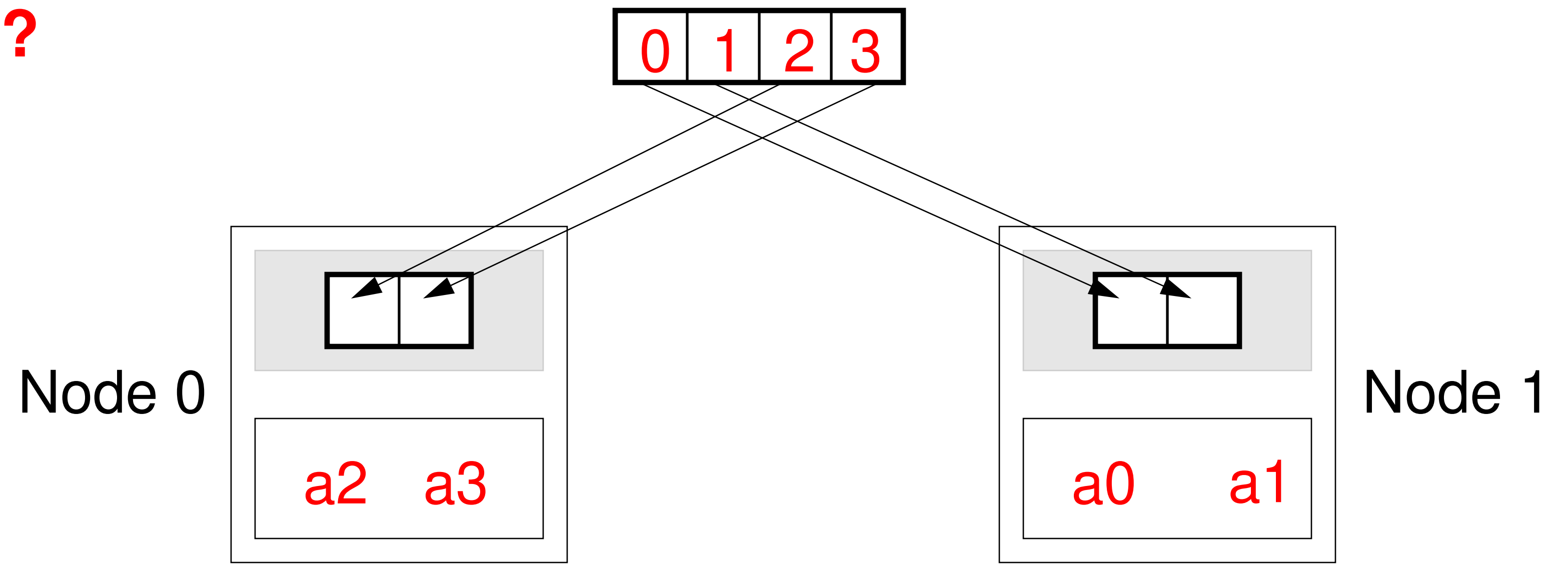
Element mapping cost amortized over slice




Department of Computer Sciences
University of Wisconsin-Madison

# Computation Mapping (cont)

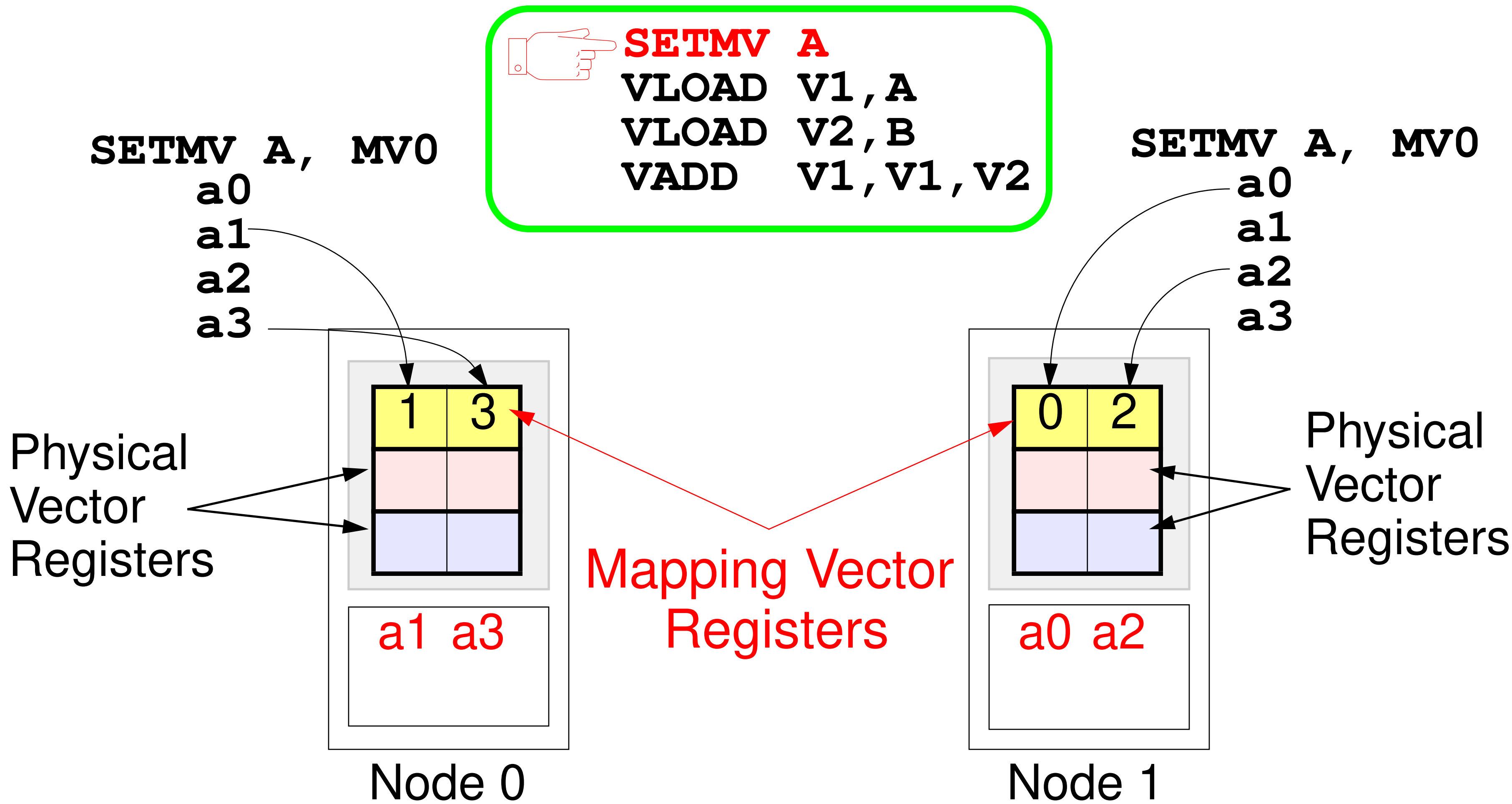
**HOW?**




**Idea: copy the layout of a memory vector**

- Pick one accessed in the computation slice
- Pick first by default (Compiler may pick “better”)
- SETMV instruction & Mapping Vector Registers

# SETMV & Mapping Vector Registers





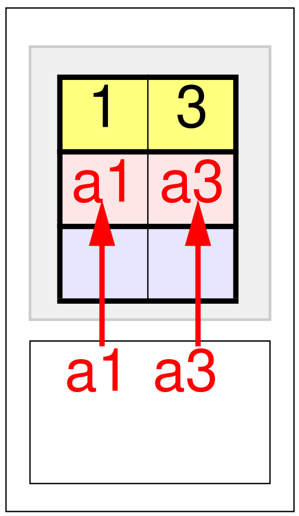
Department of Computer Sciences
University of Wisconsin-Madison

# Mapping Vectors

```
SETMV  A
VLOAD  V1,A
VLOAD  V2,B
VADD   V1,V1,V2
```

**VLOAD V1,A**

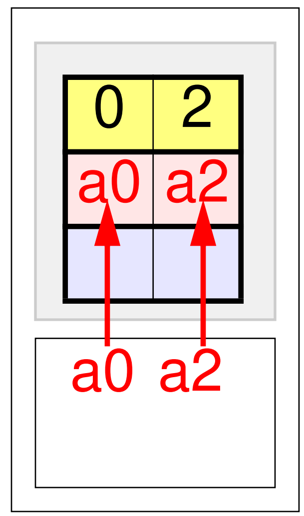
**a1**
**a3**




Node 0

**VLOAD V1,A**

**a0**
**a2**

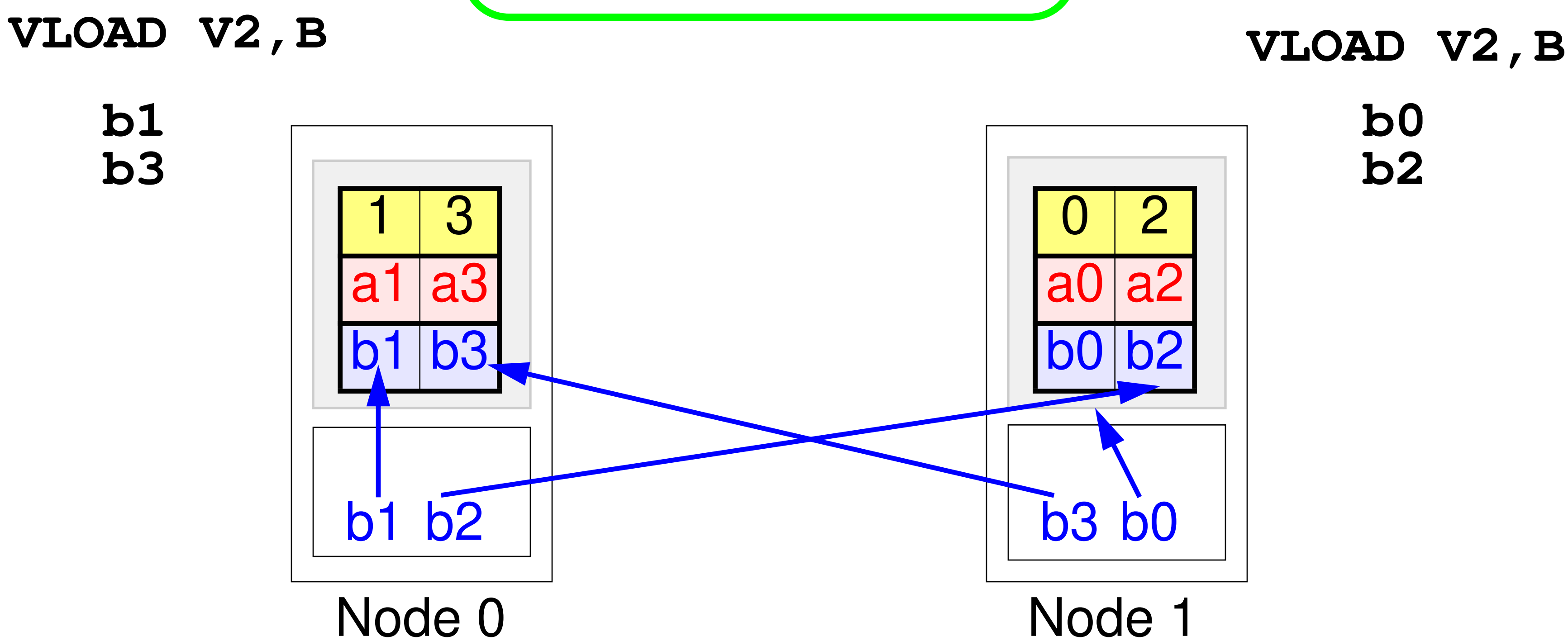



Node 1


Department of Computer Sciences
University of Wisconsin-Madison

# Mapping Vectors

```
    SETMV  A
    VLOAD  V1,A
☞   VLOAD  V2,B
    VADD   V1,V1,V2
```



# Evaluation

## External Traffic as a metric

- Bottleneck
- Low traffic enables DIVA’s parallelism advantage
- Independent of timing parameters

## Methodology:

- Trace driven simulation
- 6 NAS Benchmarks
- CRAY C-90 Traces
- DIVA vs. Caching

# DIVA vs. Caching

4-Node DIVA

1-Node All-cache

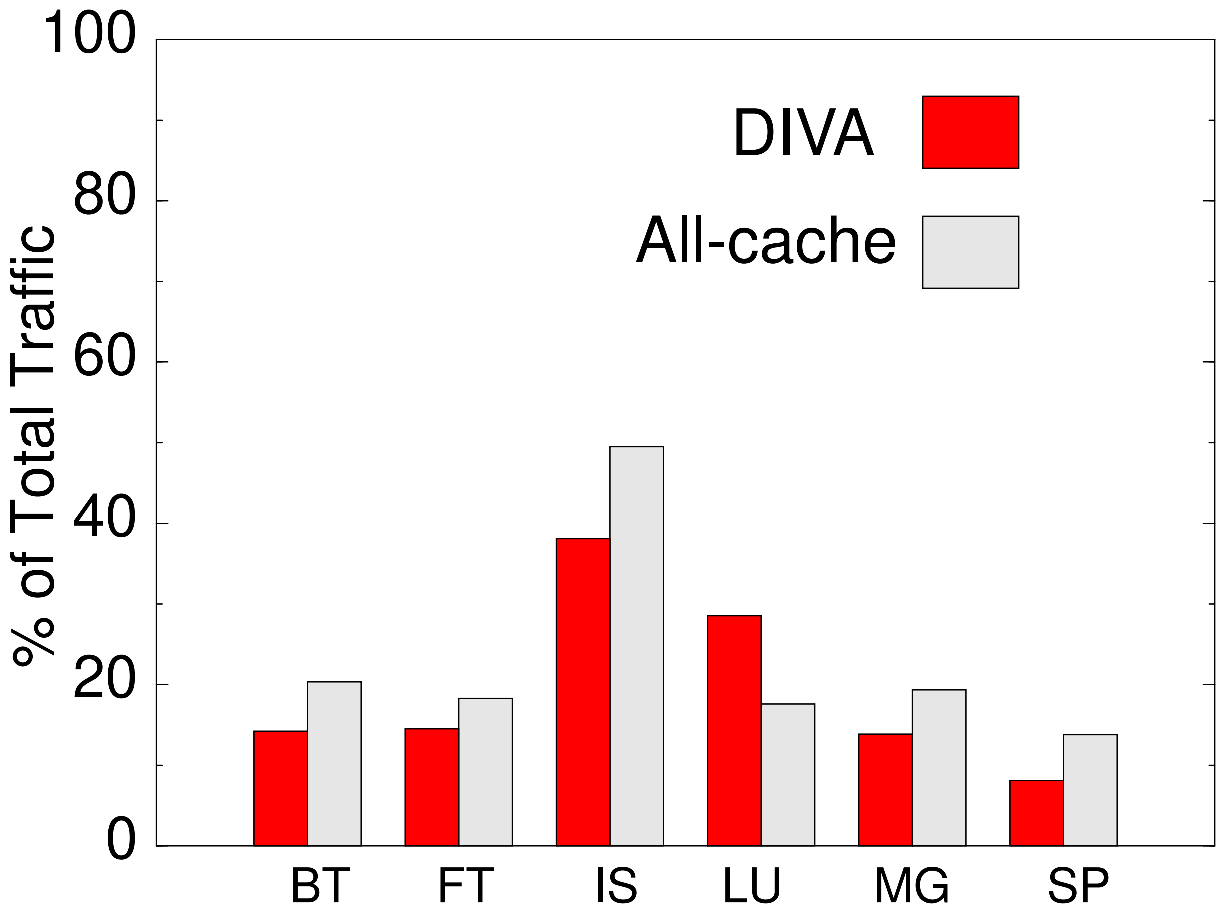

**4-node DIVA**

- on-chip mem. = 1/4 app. size
- small caches 1/16 of on-chip mem. (not critical for DIVA)

**1-node All-cache**

- on-chip cache = 1/4 app. size
- optimized for traffic: 2-way, sectored, write validate

# External Traffic: DIVA vs. Caching



# Summary

**DIVA:**

- Distribute dataset
- Distribute & dynamically map computation
- \+ Potential for lower traffic than caches
- \+ Parallelism

**Future work**

- Performance evaluation
- Data Optimization/Compiler involvement
- DIVA implementations




Department of Computer Sciences
University of Wisconsin-Madison