CS 4240: Compilers

Lecture 11: Tree Pattern Matching

Instructor: Vivek Sarkar
(vsarkar@gatech.edu)

February 18, 2019

ANNOUNCEMENTS & REMINDERS

- Homework 2 to be released today
  - Due by 11:59pm on Monday, March 4th
  - 5% of course grade

- Wednesday lecture on Register Allocation to be given by PhD student Chris Porter (GT B.S. '12, M.S. '14)
  - Regular lecture with worksheet

- MIDTERM EXAM: Wednesday, March 13, 4:30pm - 5:45pm

- FINAL EXAM: Wednesday, May 1, 2:40 pm - 5:30 pm

Instruction Selection: The Big Picture

Need pattern matching techniques

- Must produce good code (some metric for good)
- Must run quickly

Linear IR with three-address code suggests using some sort of string matching

- Process takes strings as input, matcher as output
- Each string maps to a target-machine instruction sequence
- Use text matching or **peephole matching (last lecture)**

Tree-oriented IR suggests **pattern matching on trees (today's lecture)**

- Process takes tree-patterns as input, matcher as output
- Each pattern maps to a target-machine instruction sequence
- Use dynamic programming or bottom-up rewrite systems
- It is possible to extract trees from Linear IR as well

In practice, both work well; matchers are quite different

Worksheet – 10 Solution

(From Lecture given on 02/13/2019)

- **Problem** : Use a 3-instruction window peephole matcher to generate optimized code for the following input IR fragment, by generating MIPS instructions in the subset shown below. You can perform whatever optimizations you think are reasonable as part of peephole matching, but state your assumptions.

- **Input IR:**

j = i+1

k = j+1

A[j] = k

l = A[k]

A[j] = l

Example MIPS instructions (subset)

1) Load Word instruction (read word at address ($t2+8) from memory and store in register $t1. **lw $t1,8($t2)**

2) Store Word instruction (write word from register $t1into memory address ($t2+4).
sw $t1,4($t2)

3) Add instruction (add $t1 and $t2, and store the result in $t3). **add $t3, $t1, $t2**

4) Load Immediate instruction (load constant, 99, into register $t4. **li $t4, 99**

5) Add immediate instruction (add $t1 and 1 and store the result in $t2) **addi $t2, $t1, 1**

Questions from students

- “I felt confused. Not sure what was the task for the worksheet.”
  - Sorry, we’ll try and make worksheets clearer in the future. Trade-off between make worksheet problems simple vs. interesting!
- “How do we know that it’s safe to remove/consolidate a **def** if we don’t look outside the fixed-size window?”
  - You don’t know in general. Often, a reaching definitions analysis is performed before peephole matching.

Peephole Matching (Recap)

Basic idea

- Compiler can discover local improvements locally
  - Look at a small set of adjacent operations
  - Move a "peephole" over code & search for improvement
- Classic example was store followed by load
  - Expressed using textbook's IR

Original code	Improved code
storeAI r_1 $\Rightarrow$ r_0,8	storeAI r_1 $\Rightarrow$ r_0,8
loadAI r_0,8 $\Rightarrow$ r_{15}	i2i r_1 $\Rightarrow$ r_{15}

The optimal MIPS code possible from the given IR code.

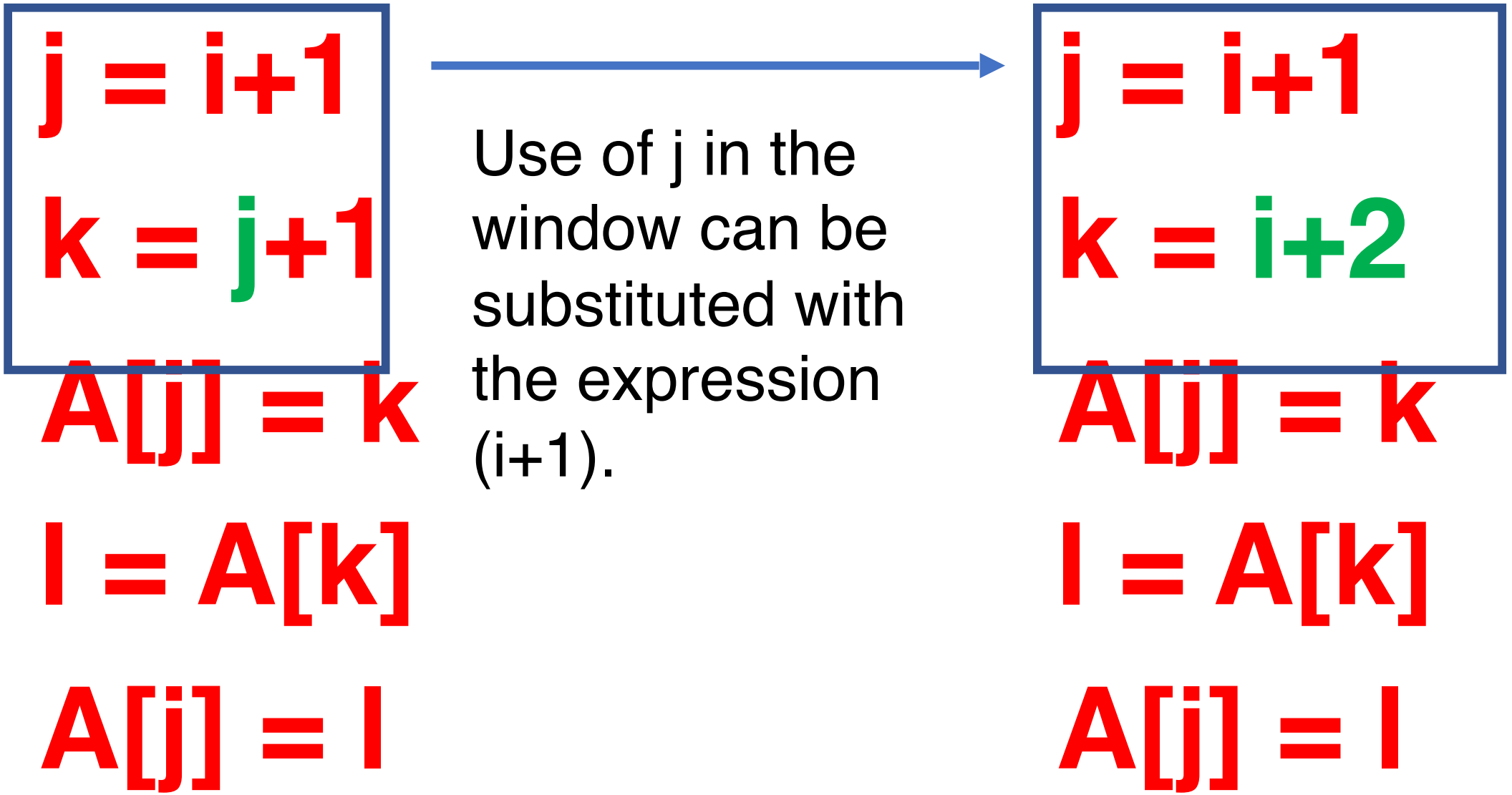
Input IR:
j = i+1
k = j+1
A[j] = k
l = A[k]
A[j] = l

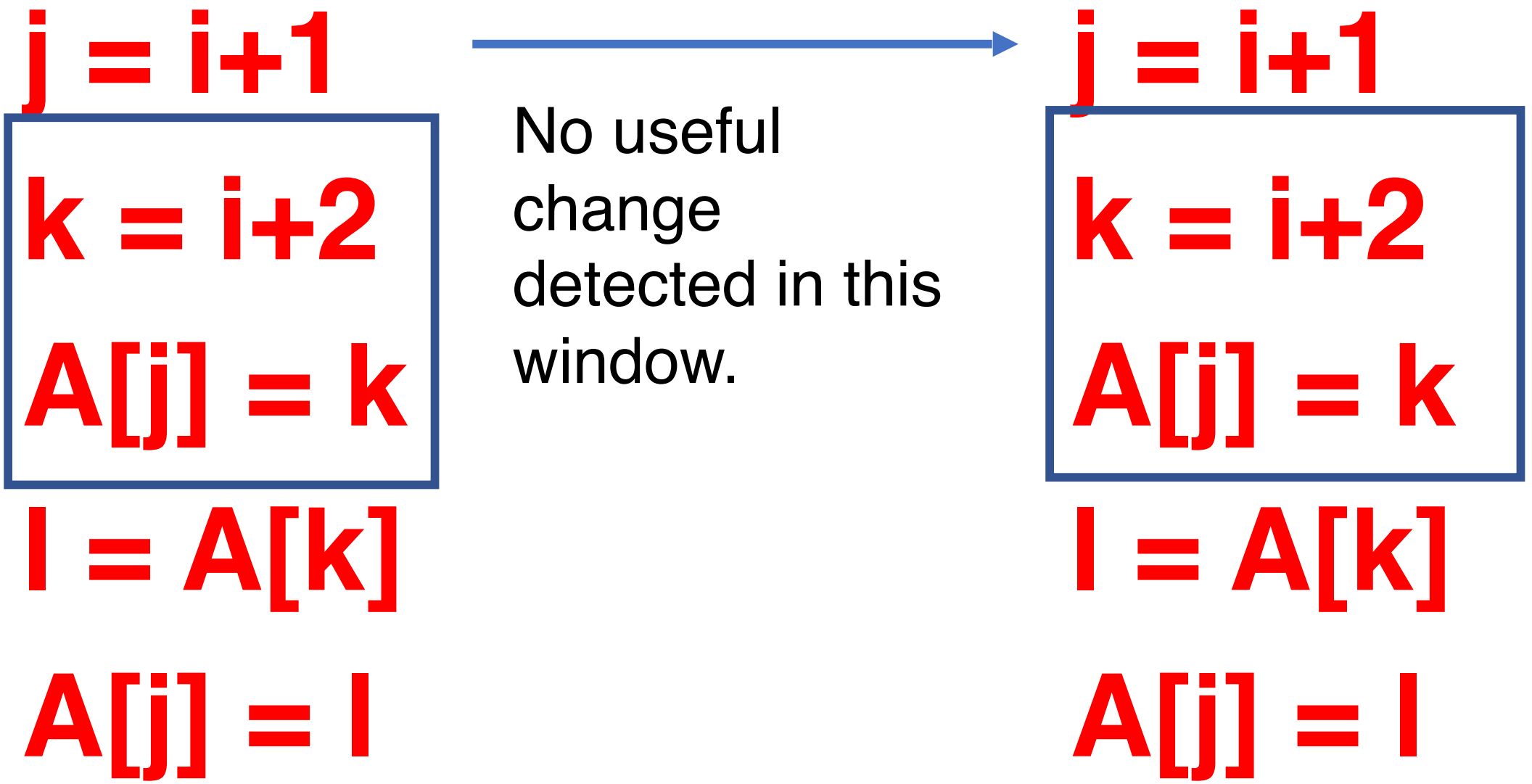
- Assumptions:
  $a0 = contains base address of array A
  $t0 = contains variable i

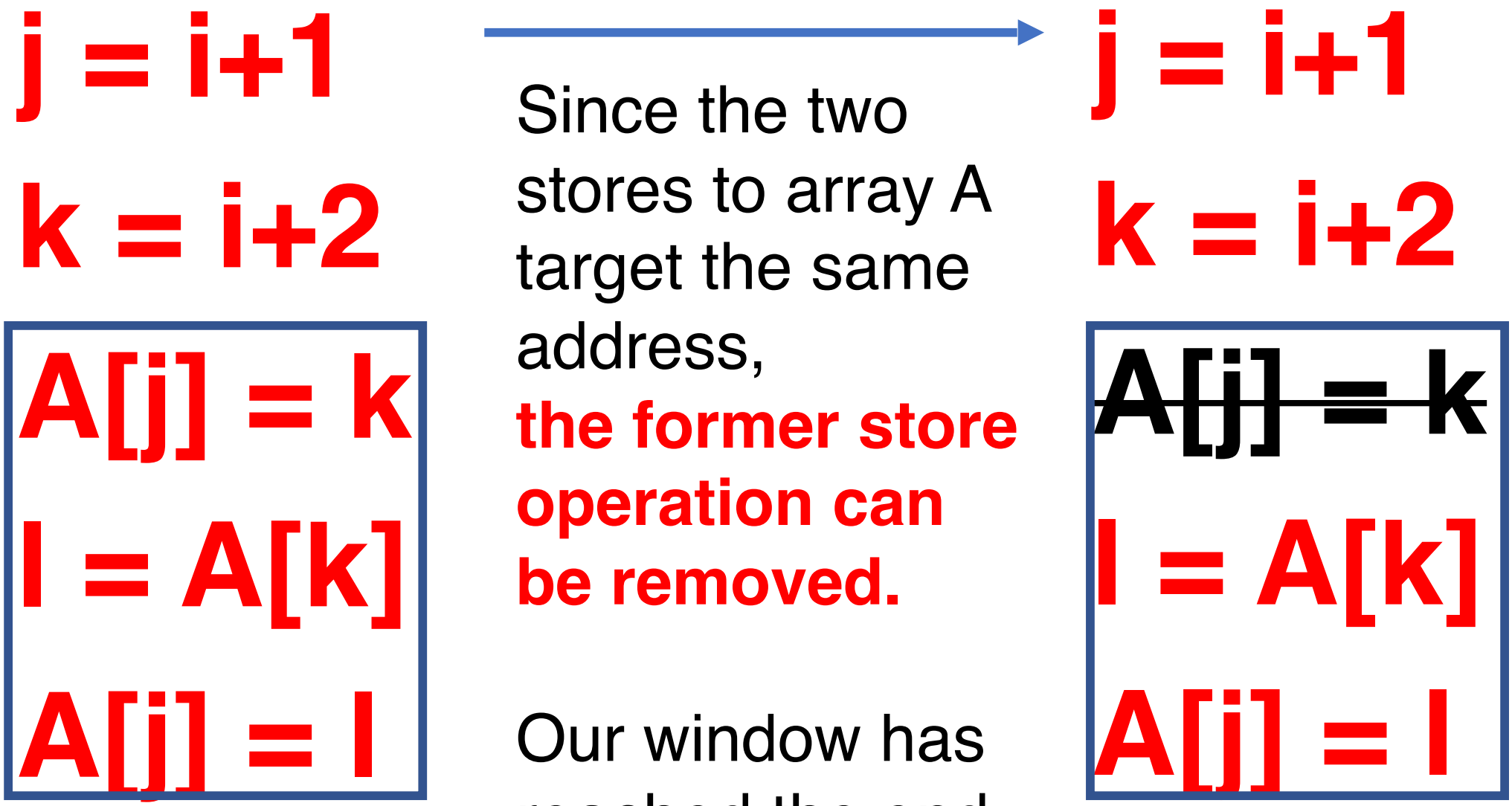
- ADDI $t1, $t0, $a0 // $t1 = address of A[i]
  LW $t2, 2($t1) // $t2 = A[i+2]
  SW $t2, 1($t1) // A[i+1] = $t2

Peephole Matching

- Using a 3-instruction window.
- Restrictions to the size of the observable region.


j = i+1
k = j+1
A[j] = k
l = A[k]
A[j] = l
Use of j in the window can be substituted with the expression (i+1).
j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l


j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l
No useful change detected in this window.
j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l


j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l
Since the two stores to array A target the same address,
the former store operation can be removed.
Our window has reached the end of the code.
End of peephole
j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l

j = i+1

k = i+2

(REMOVED)

l = A[k]

A[j] = l

TRANSLATION TO MIPS ASSEMBLY (Example)

Assumptions:
$1 = variable i
$2 = variable j
$3 = variable k
$4 = variable l
$5 = address

1. Addi $2, $1, 1

2. Sll $2, $2, 2

3. Add $t0, $2, $5

4. lw $4, 0($t0)

5. Addi, $3, $1, 2

6. Sll $t0, $3, 2

7. Add $t0, $t0, $5

8. sw $4, 0($t0)

Minimizing Cost in Instruction Selection

- Goal:
  - Find instructions with low overall cost

- Difficulty
  - How to find these patterns?
  - Machine idioms may subsume IR operations that are not adjacent

- Idea: use tree matching to go beyond peephole matching
  - Convert computation into a tree
  - Match parts of the tree

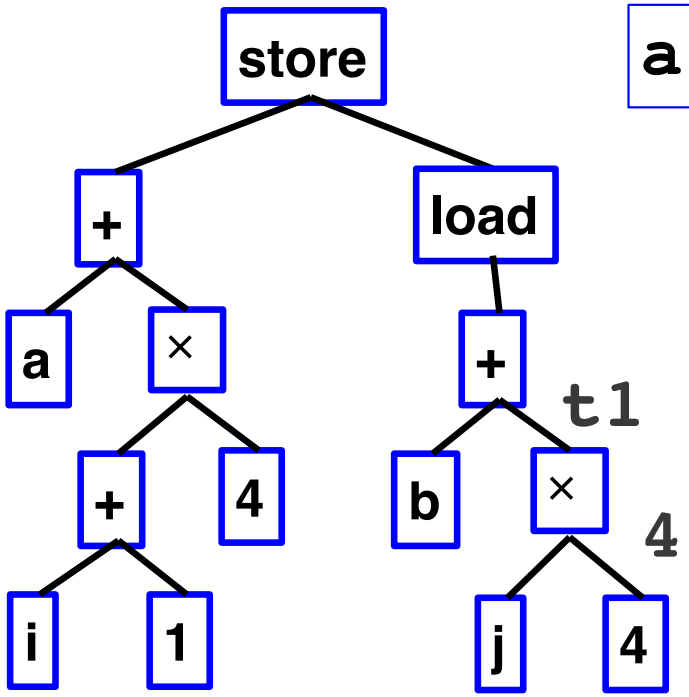
IR

```
t1 = j*4
t2 = b+t1
t3 = *t2
t4 = i+1
t5 = t4*4
t6 = a+t5
*t6 = t4
```

```
movem rb, ra
```

Dependence Tree Representation

- Edge from child to parent represents flow of values in constants / variables / temporary registers
- A tree is built for each subsequence of IR instructions in a basic block *with a single root*
  - Typically corresponds to a single source code expression




```
a[i+1] = b[j]
```

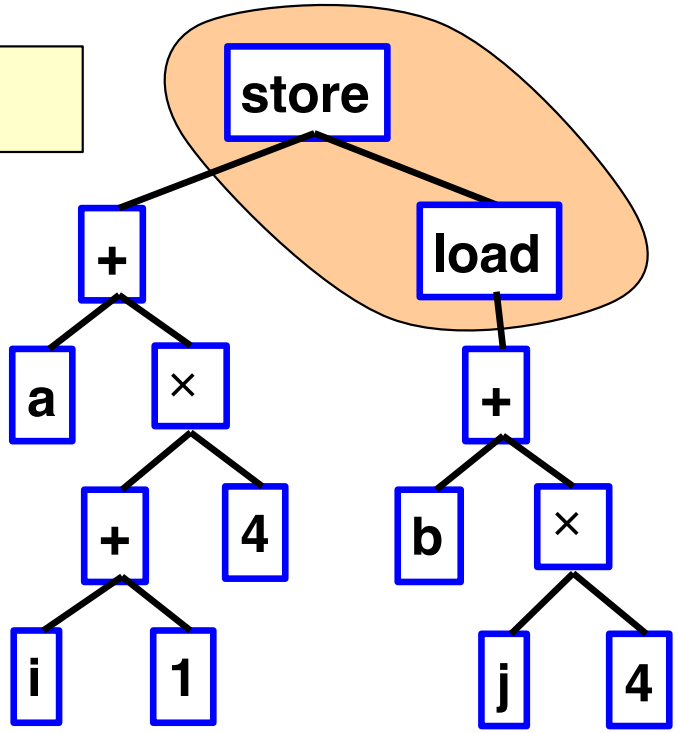
IR

```
t1 = j*4
t2 = b+t1
t3 = *t2
t4 = i+1
t5 = t4*4
t6 = a+t5
*t6 = t3
```

Tiles

- Idea: a *tile* is contiguous piece of the tree that correponds to a machine instruction

```
movem rb, ra
```

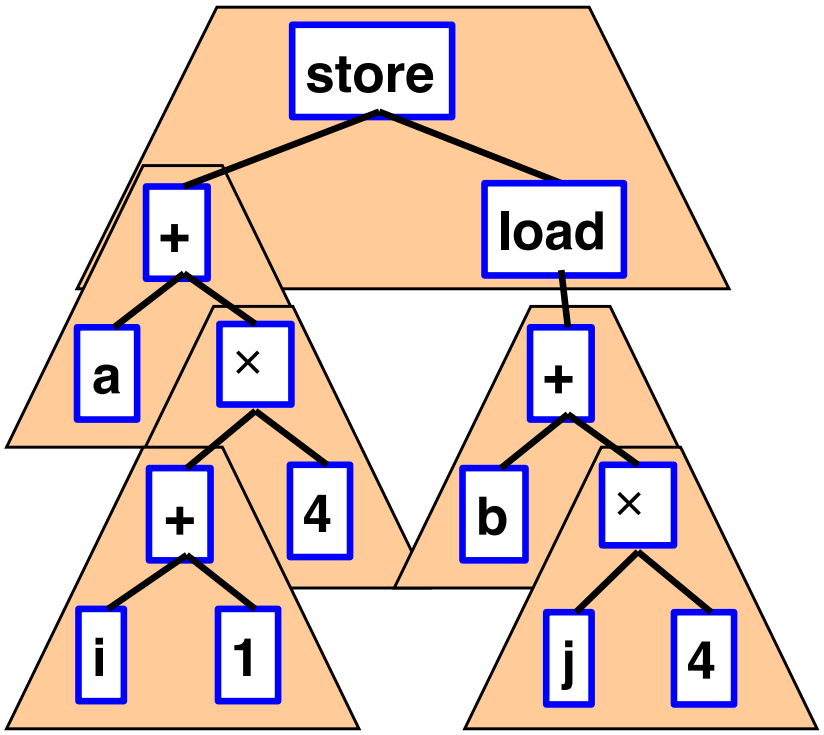


IR

```
t1 = j*4
t2 = b+t1
t3 = *t2
t4 = i+1
t5 = t4*4
t6 = a+t5
*t6 = t3
```

Tiling

- *Tiling*: cover the tree with tiles



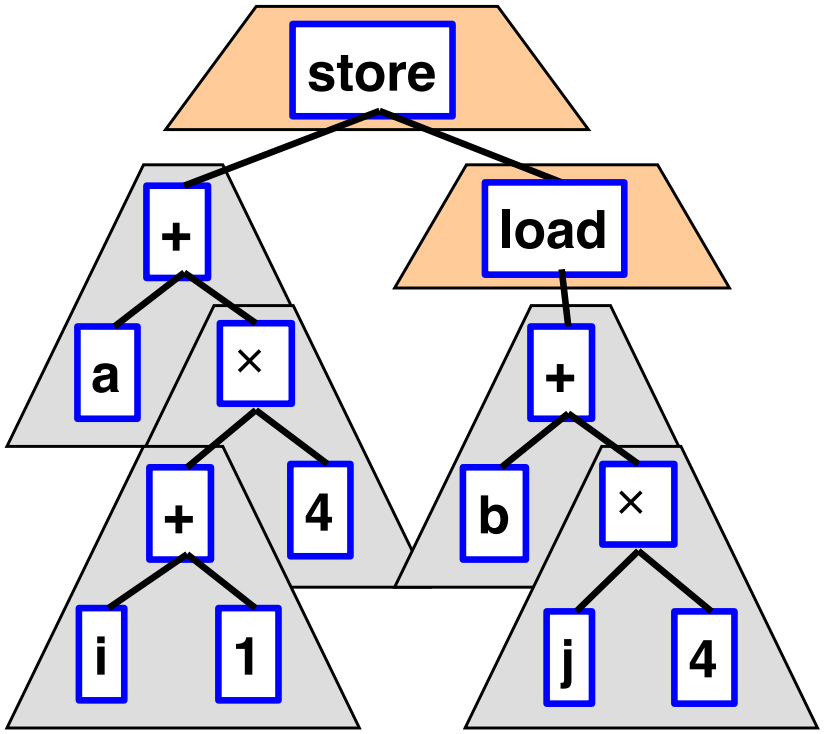
Output machine code

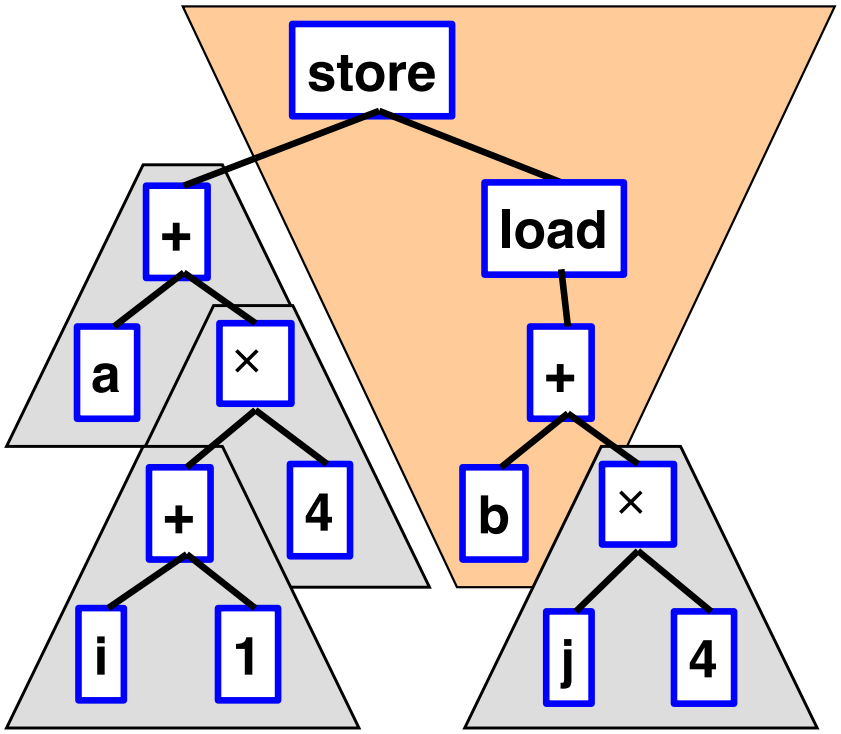
```
muli 4, rj
add rj, rb
addi 1, ri
muli 4, ri
add ri, ra
movem rb, ra
```

Multiple Tilings possible for the same Dependence Tree

```
load rb, r1
store r1, ra
```

```
movex rj, rb, ra
```



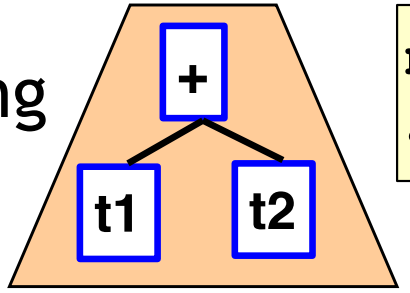


Tiling

- What's hard about this?
  - Define system of tiles in the compiler
  - Finding a tiling that implements the tree
    (Covers all nodes in the tree)
  - Finding a "good" tiling

- Different approaches
  - Greedy pattern matching
  - Optimal dynamic programming

To guarantee every tree can be tiled, provide a tile for each individual IR node




```
mov    t1, t3
add    t2, t3
```

Algorithms

- Goal: find a tiling with the lowest cost tiles
- Greedy top-down algorithm
  - Start at top of the tree
  - Find largest tile that matches top node
  - Tile remaining subtrees recursively
  - Example:

```
Tile(n) {
 if ((op(n) == PLUS) &&
      (left(n).isConst()))
  {
    Code c = Tile(right(n));
    c.append(ADDI left(n) right(n))
  }
}
```

Including Cost

- Algorithm:
  - For each node, find minimum total cost tiling for that node and the subtrees below
- Key:
  - Once we have a minimum cost for subtrees, we can find minimum cost tiling for a node by trying out all possible tiles that match the node
- Use dynamic programming!

Dynamic Programming

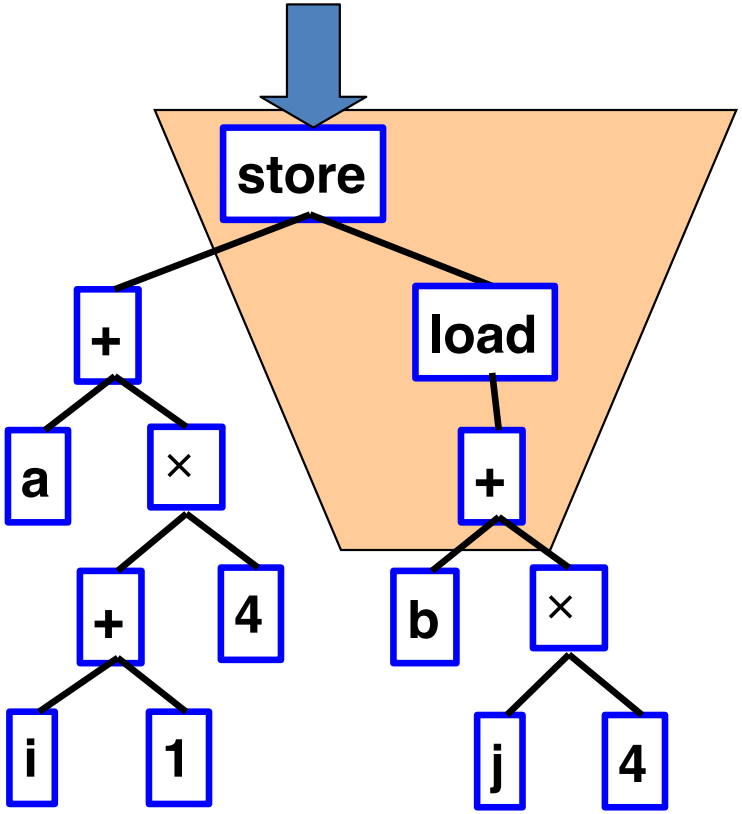
- Idea
  - For problems with ***optimal substructure***
  - Compute optimal solutions to sub-problems
  - Combine into an optimal overall solution

- How does this help?
  - Use ***memoization***: *Save previously computed solutions to sub-problems*
  - Sub-problems recur many times
  - Can work top-down or bottom-up

Recursive Algorithm

- Memoization
  - For each subtree, record best tiling in a table
- At each node
  - First check table for optimal tiling for this node
  - If none, try all possible tiles, remember lowest cost
  - Record lowest cost tile in table
  - Greedy, top-down algorithm
- We can emit code from table

Pseudocode




```
Tile(n) {
 if (best(n)) return best(n)
 // -- Check all tiles
 if ((op(n) == STORE) &&
      (op(right(n)) == LOAD) &&
      (op(child(right(n)) == PLUS)) {
    Code c = Tile(left(n))
    c.add(Tile(left(child(right(n))))
    c.add(Tile(right(child(right(n)))))
    c.append(MOVEX . . .)
    if (cost(c) < cost(best(n))
      best(n) = c
  }
 // . . . and all other tiles . . .
 return best(n)
}
```

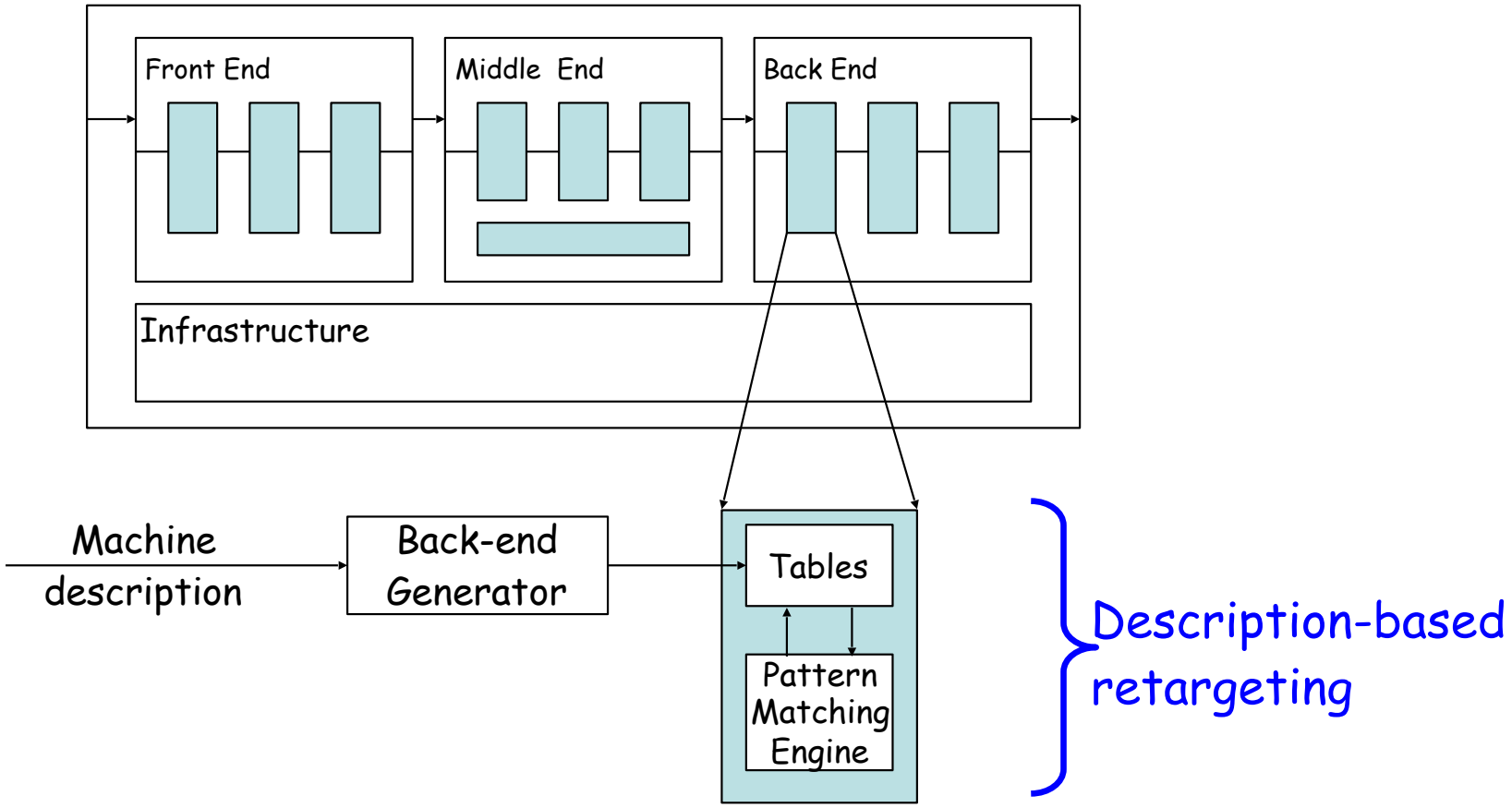
Towards a more general Algorithm

- Problem:
  - Hard-codes the tiles in the code generator

- Alternative:
  - Define tiles in a separate specification
  - Use a generic tree pattern matching algorithm to compute tiling
  - Tools: *back-end generators*

The Goal (Recap)

Want to automate generation of instruction selectors



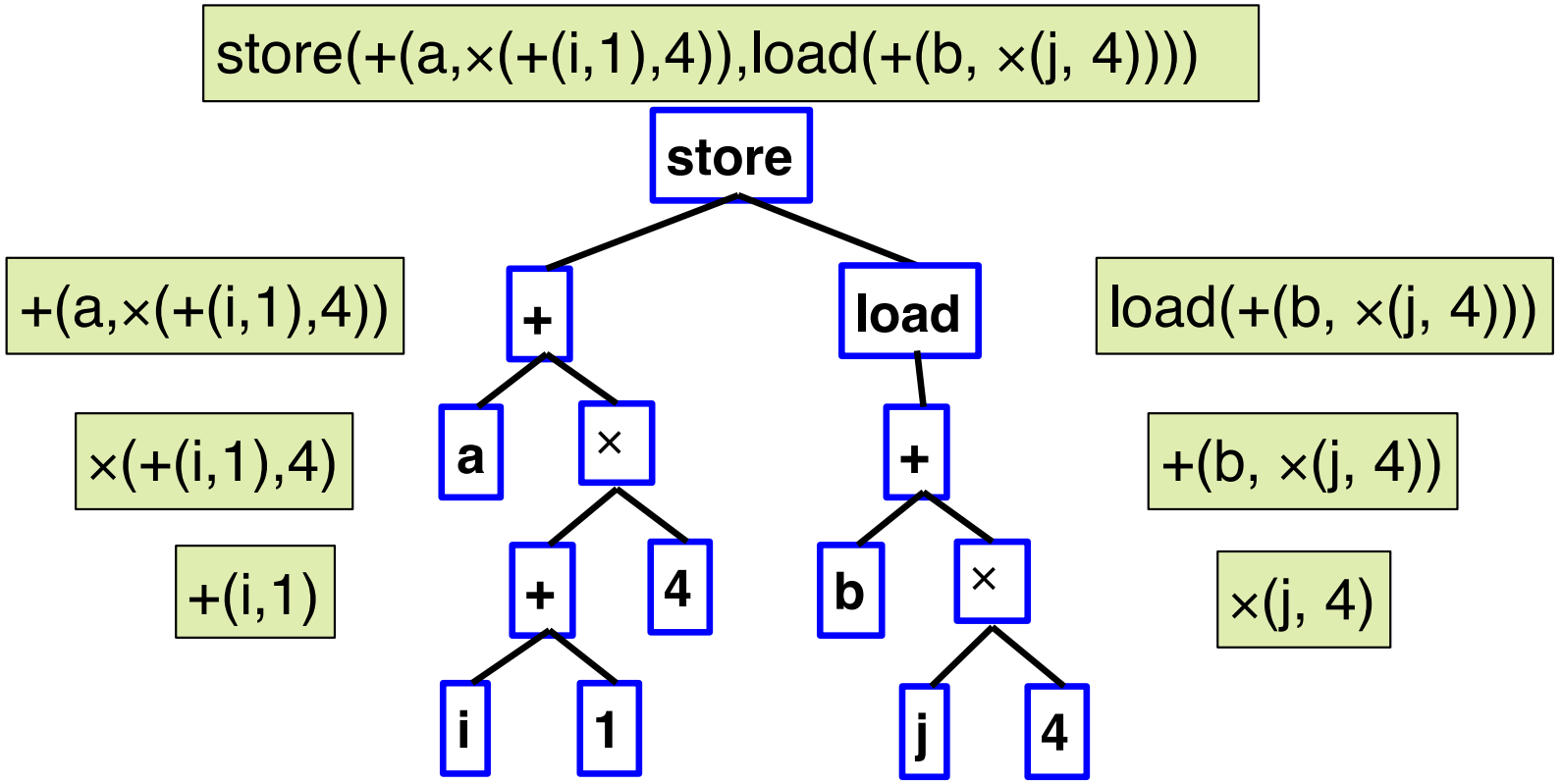
Machine description should also help with scheduling & allocation

Back-end Generators

- Tree description language
  - Represent IR tree as text
- Specification
  - IR tree patterns
  - Code generation actions
- Generator
  - Takes the specification
  - Produces a code generator
- Provides linear time optimal code generation
  - BURS (bottom-up rewrite system)
  - burg, Twig, BEG

Tree Notation

- Use prefix notation to linearize tree



Rewrite Rules

- Rule
  - Pattern to match and replacement
  - Cost
  - Code generation template
  - May include actions – e.g., generate register name

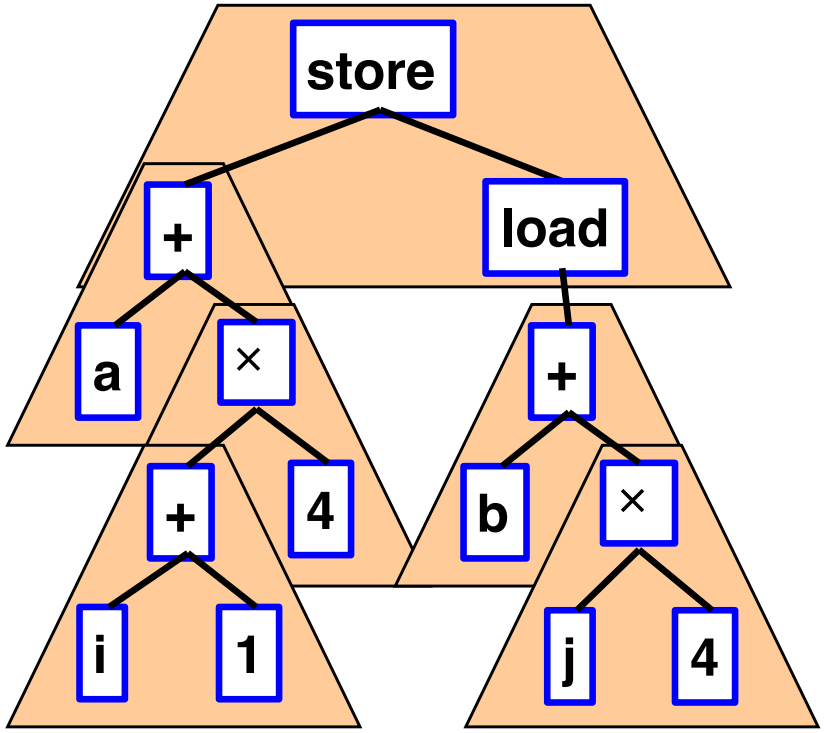
Pattern, replacement	*Cost*	*Template*
+(reg_1,reg_2) → reg_2	**1**	**add r1, r2**
store(reg_1, load(reg_2)) → *done*	**5**	**movem r2, r1**

Rewrite Rules

Example rules:

#	*Pattern, replacement*	*Cost*	*Template*
1	$+(reg_1,reg_2) \rightarrow reg_2$	1	`add r1, r2`
2	$\times(reg_1,reg_2) \rightarrow reg_2$	10	`mul r1, r2`
3	$+(num,reg_1) \rightarrow reg_2$	1	`addi num, r1`
4	$\times(num,reg_1) \rightarrow reg_2$	10	`muli num, r1`
5	$store(reg_1, load(reg_2)) \rightarrow$ *done*	5	`movem r2, r1`

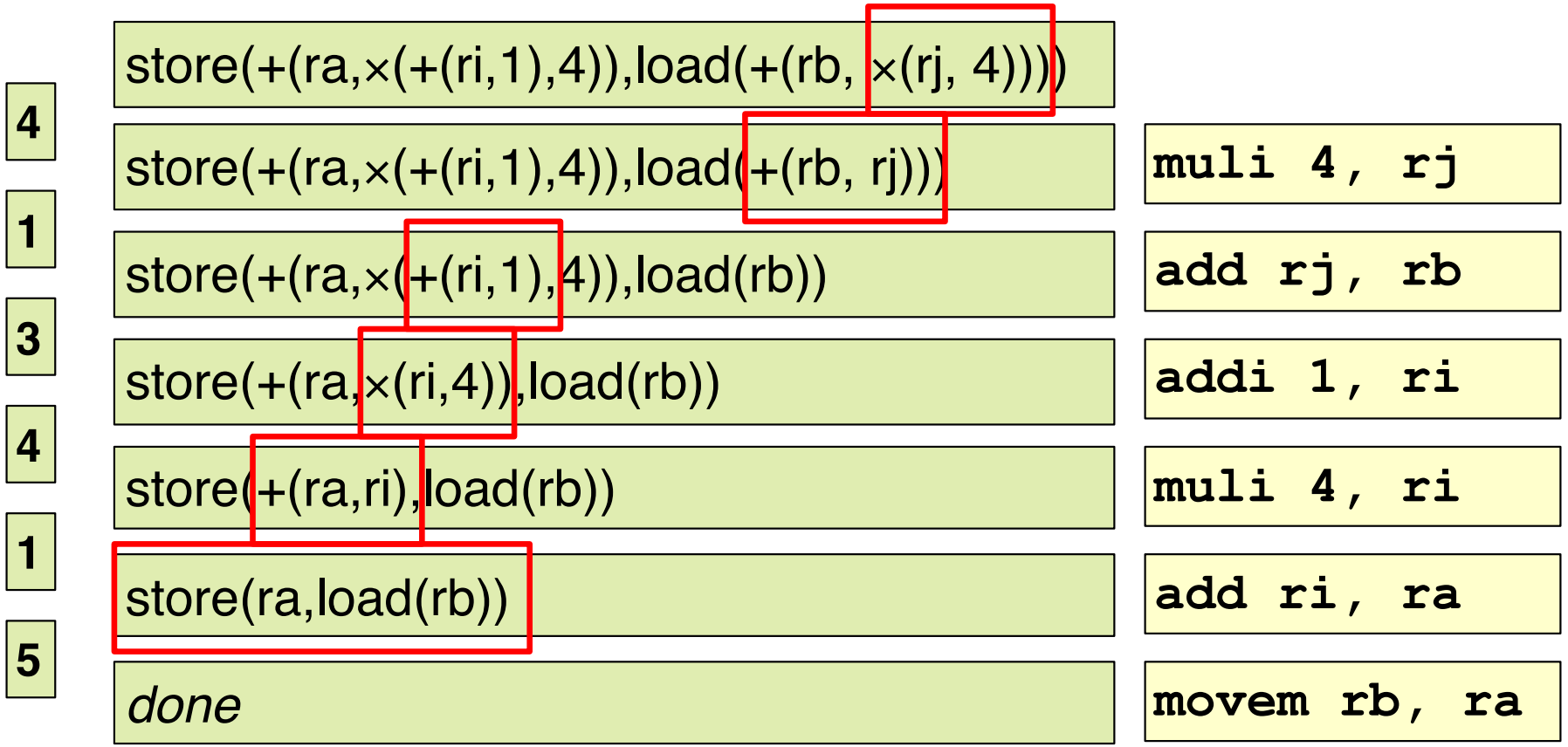
Example



Assembly

```
muli 4, rj
add rj, rb
addi 1, ri
muli 4, ri
add ri, ra
movem rb, ra
```

Rewriting Process



Did you receive this email from our CoC Dean, Zvi Galil?

Begin forwarded message:
From: "Galil, Zvi" <galil@cc.gatech.edu>
Subject: [Faculty] more good news
Date: February 13, 2019 at 9:05:36 AM EST
To: faculty <faculty@cc.gatech.edu>, "staff@cc.gatech.edu" <staff@cc.gatech.edu>, "newsandevents@cc.gatech.edu" <newsandevents@cc.gatech.edu>, "ms-coc-announce@cc.gatech.edu" <ms-coc-announce@cc.gatech.edu>, "phd-coc-announce@cc.gatech.edu" <phd-coc-announce@cc.gatech.edu>, "omscs-official@cc.gatech.edu" <omscs-official@cc.gatech.edu>

Hello everyone,

We are in one of our busiest times of year, managing all of the classes, lectures, conferences and other activities of the spring semester. I encourage all of you, however, to make the time to take part in the search for the next dean of the College of Computing. Four finalists have been announced, including our own Charles Isbell and Ellen Zegura. In order to learn how to see the candidates' public presentations – and how to offer feedback to the search committee – you may check the dean's search website.

Our faculty and students continue to win grants and to be recognized for their cutting-edge work:

- **Jimeng Sun** (CSE) was ranked among the top 100 leaders in drug discovery and advanced healthcare by Deep Knowledge Analytics, a top technology think tank.
- **David Joyner** received the USG Board of Regents' award for Online Teaching for his work in OMSCS and the online section of CS1301.
- **Vivek Sarkar** (CS), **Tom Conte** (CS), **David Bader** (CSE) and **Rich Vuduc** (CSE) have been awarded $4.5 million from the DARPA Electronics Resurgence Initiative. The team will build new programming systems for developing data-intensive algorithms with automatic software and hardware co-optimization. The project is titled "Dynamic Data-Aware Reconfiguration, Integration and Generation," or DDARING. Read more here.
- The Institute's story on the fifth anniversary of OMSCS has been the top item on the home page for more than two weeks. If you haven't read it yet (or watched the video), you can check it out here.

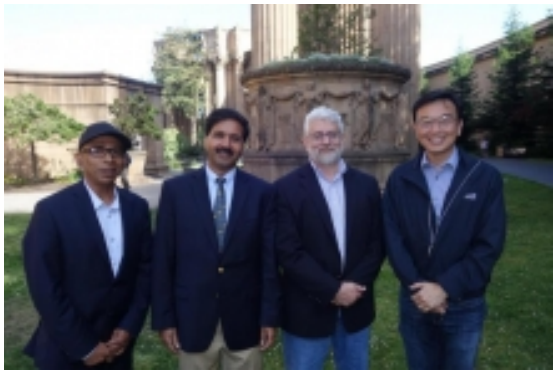
The OMSCS program continues to go from strength to strength, with 8,664 students enrolled. According to researchers at Harvard, we now have the largest MS in CS program in the country, and perhaps in the world. I will give my 62nd talk on the subject at Princeton University this week - Mark Braunstein (IC) and four OMSCS students are coming to help, or at least to watch.

Best,

zvi

College of Computing Professors Receive DARPA Contract Award to Improve Software and Hardware Co-optimization

Mon, 01/28/2019



Georgia Tech researchers have been awarded $4.5 million to build new programming systems for developing data-intensive algorithms with automatic software and hardware co-optimization. The project is named *Dynamic Data-Aware Reconfiguration, INtegration and Generation (DDARING)*.

School of Computer Science Professors and Co-Directors of the Center for Research into Novel Computing Hierarchies (CRNCH) **Vivek Sarkar** and **Tom Conte** and School of Computational Science and Engineering's Chair **David Bader** and Associate Professor **Richard Vuduc** are co-investigators on this project from Tech. The award will also be used to support research collaborators at University of Illinois, University of Michigan, and University of Southern California.

"This is a unique opportunity to address post-Moore computing challenges through new research on software and hardware co-optimization by adapting the hardware to executing applications and the data sets being analyzed," said Sarkar. "Advancing data analysis algorithms and pushing the boundaries of hardware are areas of strength for Georgia Tech, and we're excited to work with our partners at Illinois, Michigan, and USC on this important challenge for future computing platforms."

DDARING: Dynamic Data-Aware Reconfiguration, INtegration and Generation

PI & POC: Vivek Sarkar[1] (vsarkar@gatech.edu)

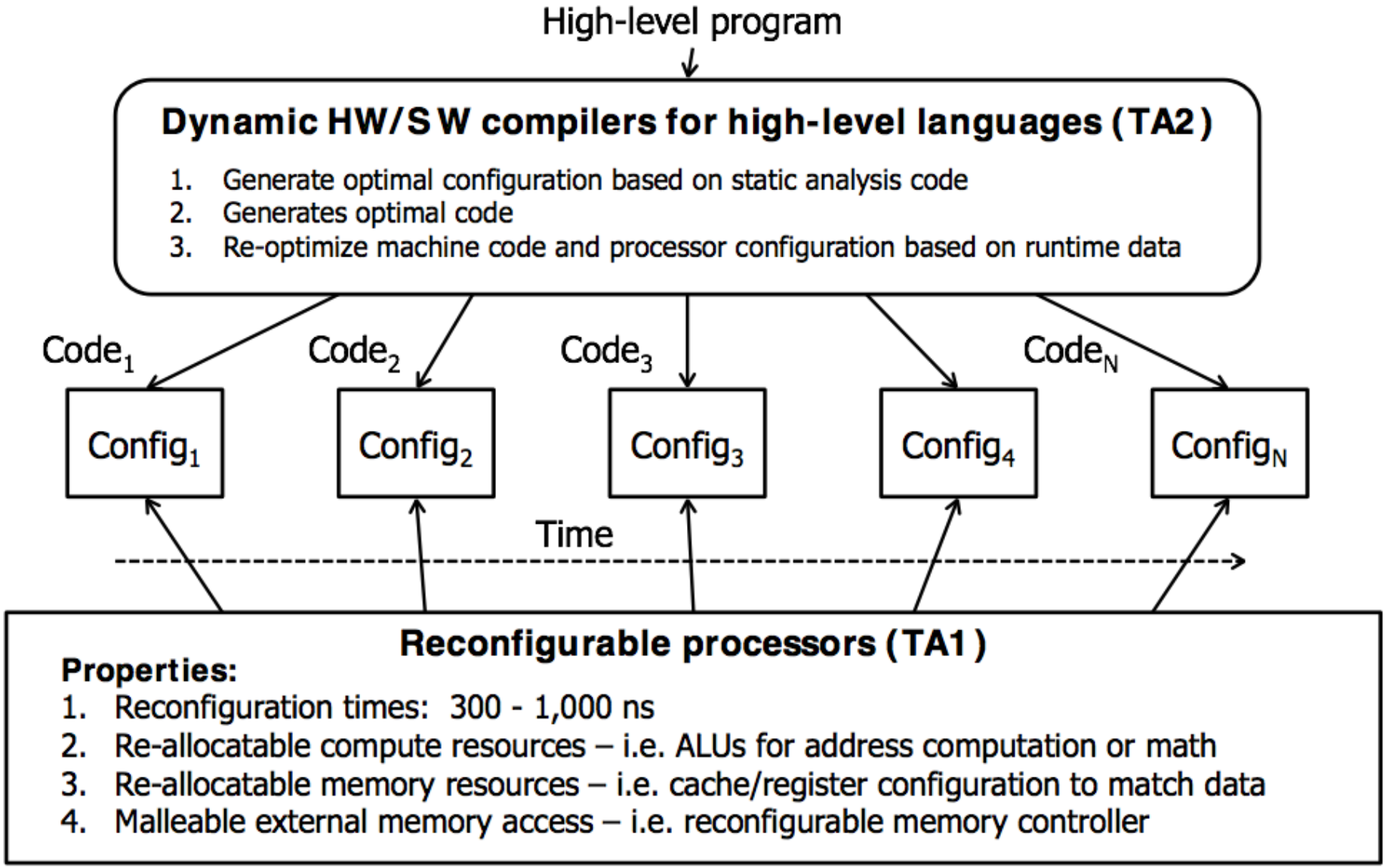
Co-PIs: David Bader[1], Deming Chen[2], Tom Conte[1], Wen-mei Hwu[2], Scott Mahlke[3], Viktor Prasanna[4], Richard Vuduc[1]

1 Georgia Institute of Technology
3 University of Michigan
2 Illinois — University of Illinois at Urbana-Champaign
4 USC University of Southern California

DDARING Team Members

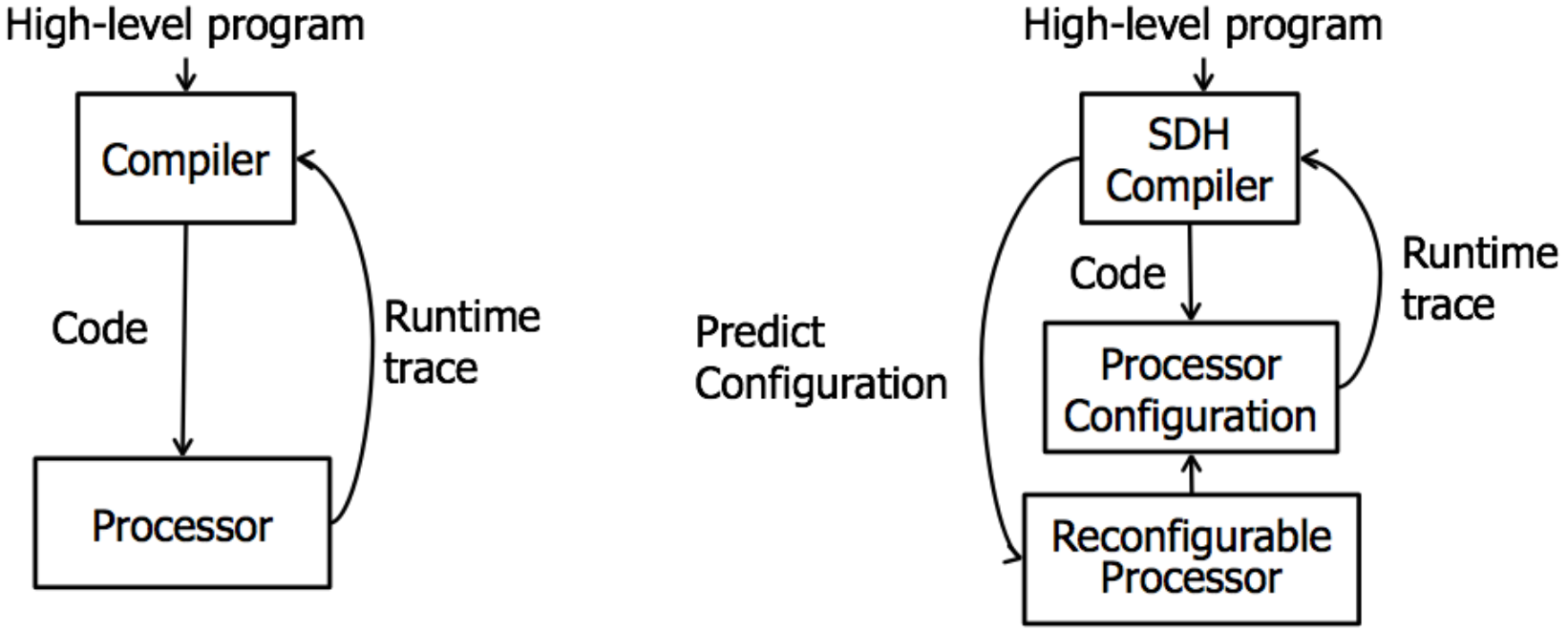
- **Georgia Tech:** Vivek Sarkar (Principal Investigator), David Bader, Tom Conte, Rich Vuduc (Faculty), Jun Shirako (Senior Research Scientist), Prithayan Barua, Prasanth Chatarasi, Mikhail Isaev, Anirudh Jain, Joseph Lennon, Ankush Mandal, Tong Zhou (PhD students)
- **UIUC:** Deming Chen, Wen-Mei Hwu (Faculty), Abdul Majed Dakkak, Ashutosh Dhar, Sitao Huang, Vandana V Kulkarni (PhD students), Mang Yu (MS student)
- **U.Michigan:** Scott Mahlke (Faculty), Jonathan Bailey, Armand Behroozi, Pedram Zamirai (PhD students)
- **USC:** Viktor Prasanna (Faculty), Rajgopal Kannan (Research Faculty), Ajitesh Srivastava (Postdoc), Hongkuan Zhou (PhD student)

Software-defined hardware



Source: https://www.darpa.mil/attachments/SDH_Proposers_Day.pdf

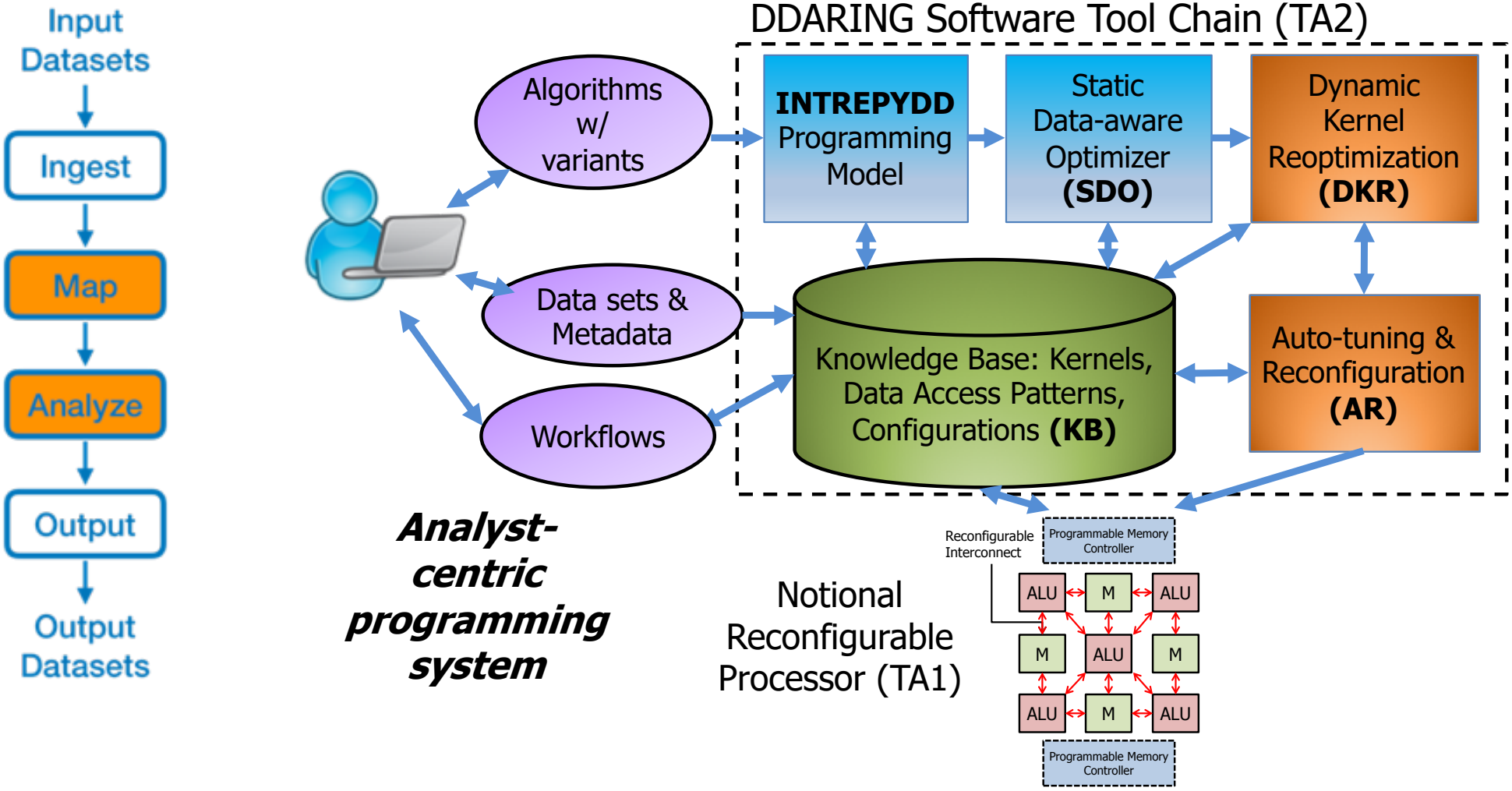
TA2: Compilers to build hardware and software



- Compilers generate optimal code via static analysis + tracing methods
  - Assume static processor configuration, compile code, run, trace, recompile
- SDH compilers don't assume a static processor configuration
  - Generates optimal configuration/code given program + data
  - Problem: Resources and architecture optimization space is large
- Solution:
  1. Configure initial processor configuration, compile code, run and trace, then
  2. Predict best configuration via reinforcement learning/stochastic optimization

Source: https://www.darpa.mil/attachments/SDH_Proposers_Day.pdf

Overview of DDARING project



Overview of INTREPYDD tool chain