NTNU – Trondheim
Norwegian University of
Science and Technology

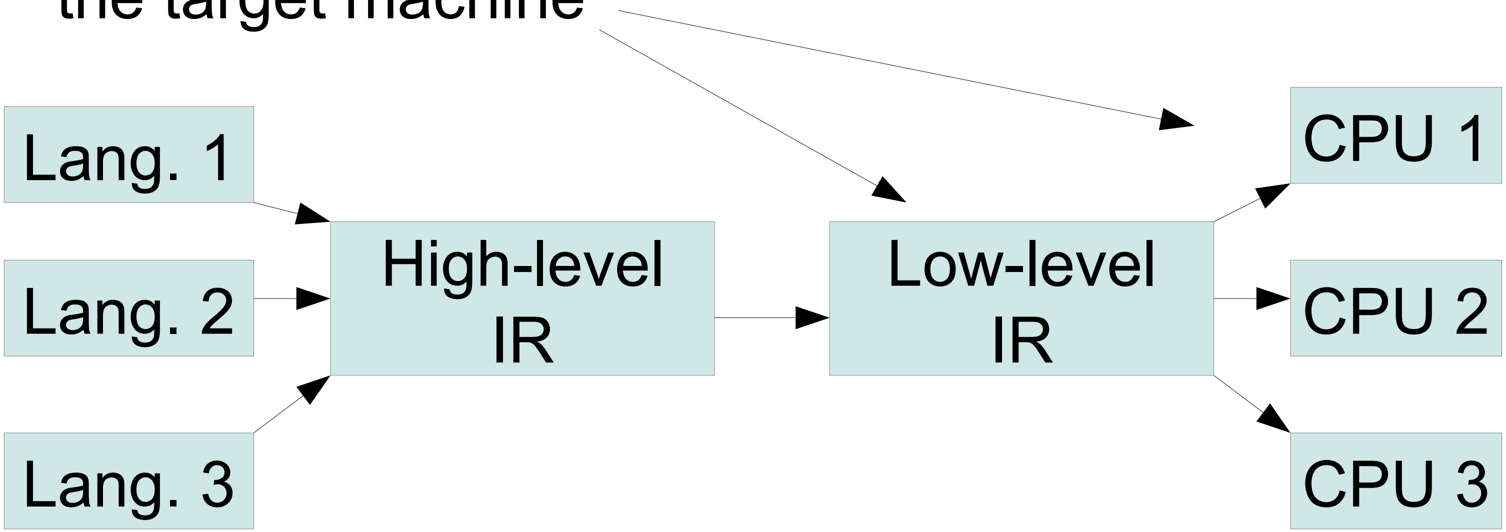
Instruction selection

Where we are

- We have a fairly low-level view of the program, but
  - It features a memory model of infinite temporary variables
  - It isn't specific in terms of operations provided by the architecture
- These will be our last two topics
  - Selecting machine-specific operations
  - Mapping variables to memory locations

Low-IR vs. machinery

- The instructions of low-level IR are not the same as the target machine



Straightforward solution

- Map every low-level IR to a fixed sequence of assembly instructions

x = y + z →

```
move y,r1
move z,r2
add r1,r2
move r2, x
```

- Disadvantages:
  - Lots of redundant operations
  - More memory traffic than necessary

There may be several alternatives

- Translate a[i+1] = b[j]

  using these operations

add r2,r1	←	r1 = r1 + r2
mul c, r1	←	r1 = r1 * c
load r2, r1	←	r1 = *r2
store r2, r1	←	*r1 = r2
movem r2, r1	←	*r1 = *r2
movex r3, r2, r1	←	*r1 = *(r2+r3)

The general steps

Let's say that everything is 8-byte elements, and

- Register r_a holds &a
- Register r_b holds &b
- Register r_i holds i
- Register r_j holds j

a[i+1] = b[j] needs to

- Find address of b[j]
- Load b[j]
- Find address of a[i+1]
- Store into a[i+1]

One translation

- Address of b[j]
  - mulc 8,r_j
  - add r_j, r_b
- Load b[j]
  - load r_b, r1
- Address of a[i+1]
  - add 1, r_i
  - mulc 8, r_i
  - add r_i, r_a
- Store into a[i+1]
  - store r1, r_a

TAC

```
t1 = j*8
t2 = b+t1
t3 = *t2
t4 = i+1
t5 = t4*8
t6 = a+t5
*t6 = t3
```

Another translation

- Address of b[j]

  mulc 8,r_j

  add r_j, r_b

- Address of a[i+1]

  add 1, r_i

  mulc 8, r_i

  add r_i, r_a

- Store into a[i+1]

  movem r_b, r_a

TAC

```
t1 = j*8
t2 = b+t1
t3 = *t2
t4 = i+1
t5 = t4*8
t6 = a+t5
*t6 = t3
```

NTNU – Trondheim
Norwegian University of Science and Technology

One more translation

- Address of b[j]
  - mulc 8,r_j
- Address of a[i+1]
  - add 1, r_i
  - mulc 8, r_i
  - add r_i, r_a
- Store into a[i+1]
  - movex r_j, r_b, r_a

TAC

```
t1 = j*8
t2 = b+t1
t3 = *t2
t4 = i+1
t5 = t4*8
t6 = a+t5
*t6 = t3
```

NTNU – Trondheim
Norwegian University of Science and Technology

Why care?

- Not all instructions are created equal
- Some complete in a clock cycle
- Others decompose into a sequence of steps, and take many

- If we have a choice of translations, we'd like the one with the smallest sum of costs

Partial instructions aren't necessarily adjacent

- Address of b[j]
  - mulc 8,r_j
- Address of a[i+1]
  - add 1, r_i
  - mulc 8, r_i
  - add r_i, r_a
- Store into a[i+1]
  - movex r_j, r_b, r_a

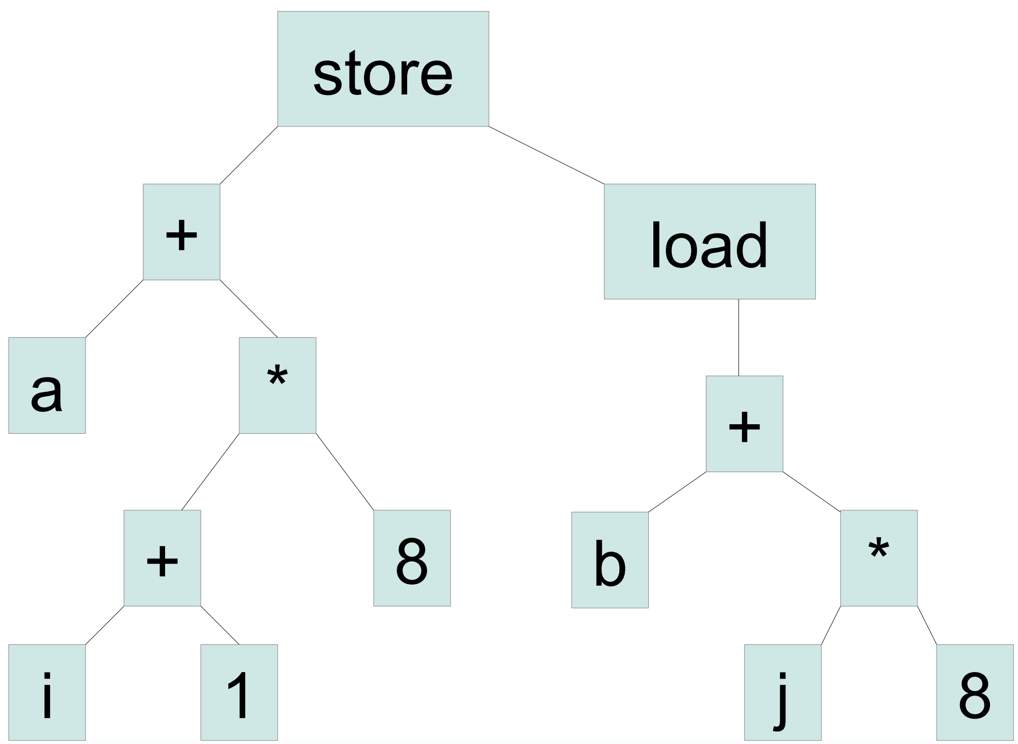
TAC

```
t1 = j*8
t2 = b+t1
t3 = *t2
t4 = i+1
t5 = t4*8
t6 = a+t5
*t6 = t3
```

NTNU – Trondheim
Norwegian University of
Science and Technology

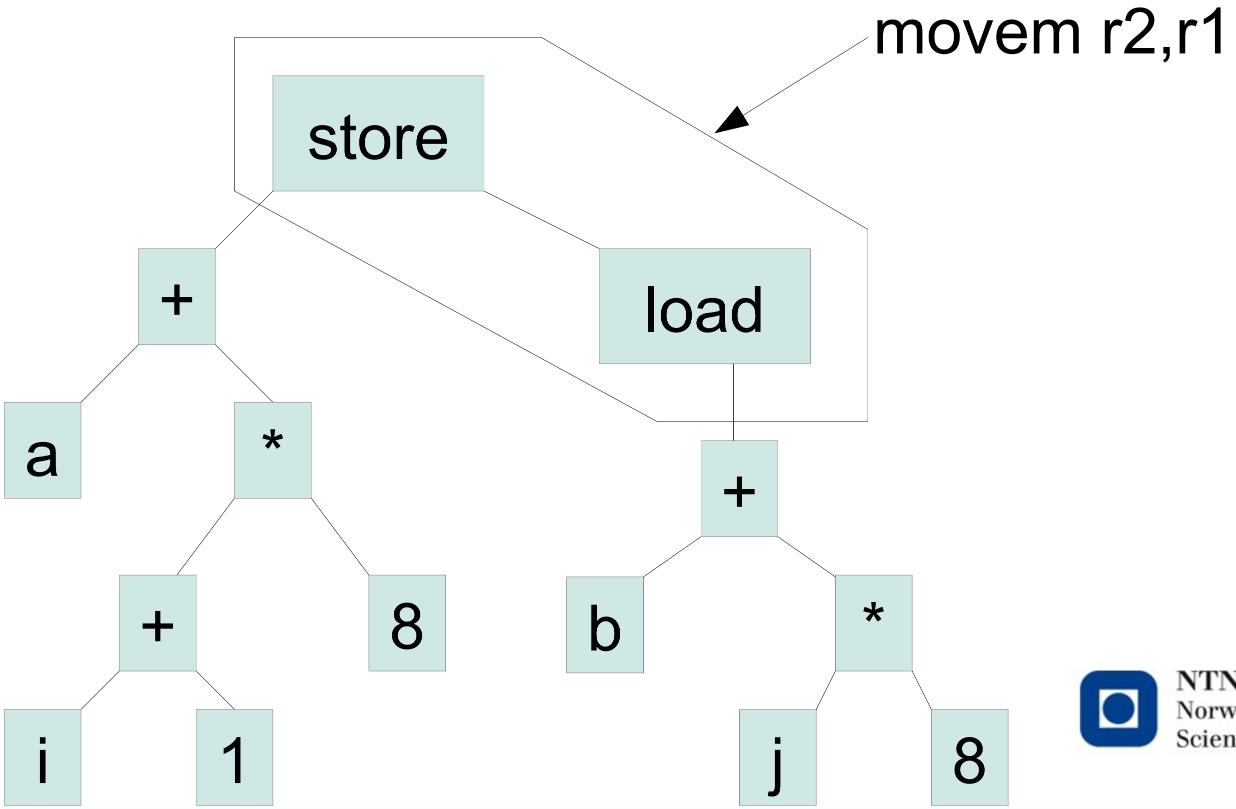
Tree representation

- The 4 overall steps can be written as a tree



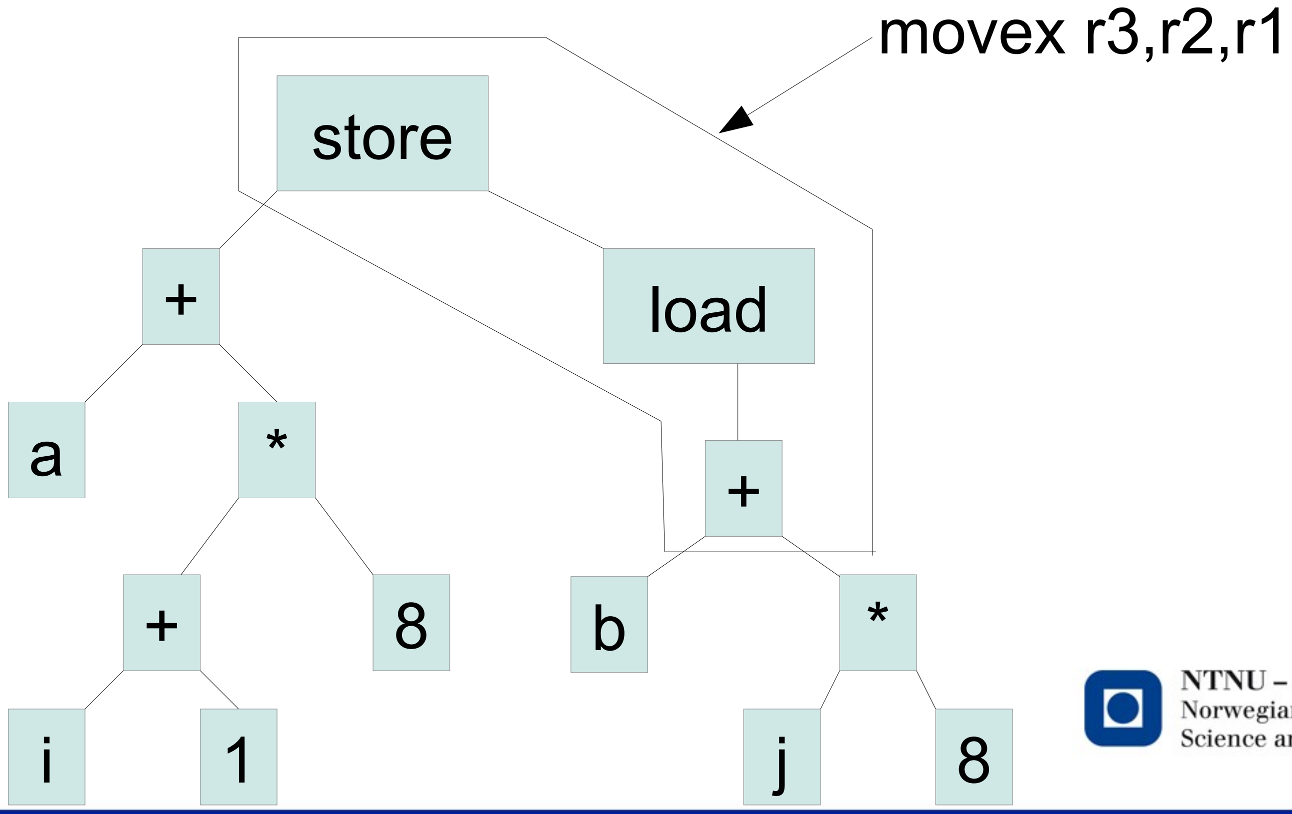
Instructions can be *tiles*

(Subtrees of a particular pattern)



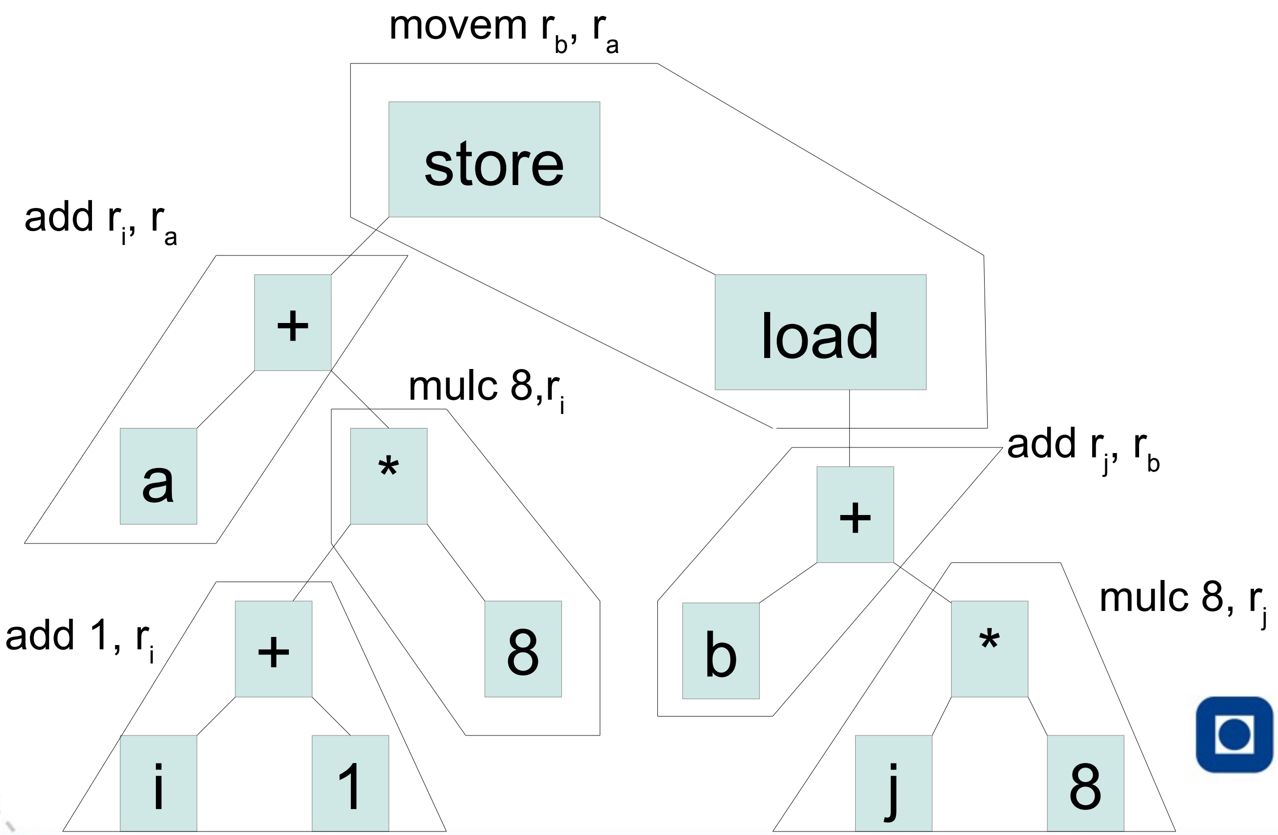
Instructions can be *tiles*

(Subtrees of a particular pattern)



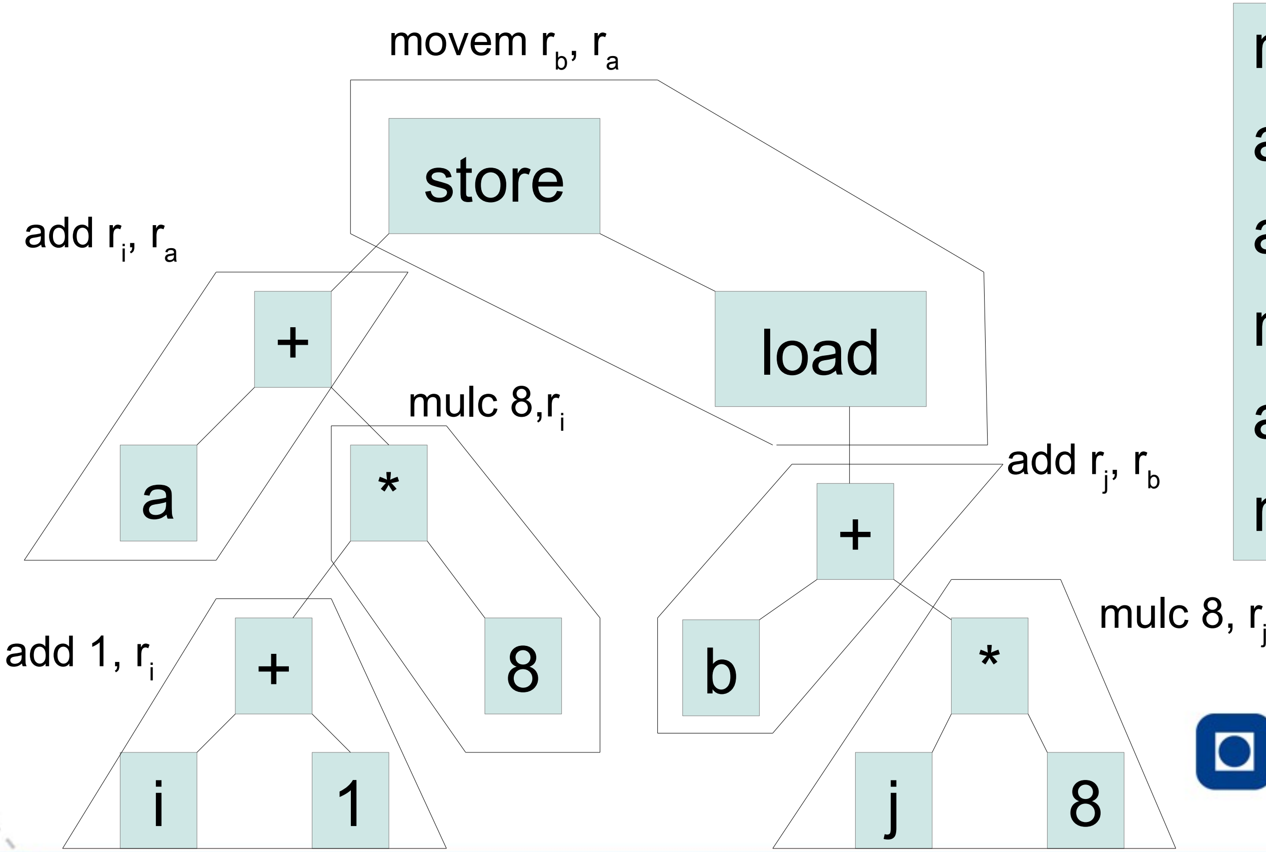
Tiling

An instruction selection covers the tree with disjoint tiles



Tiling

An instruction selection covers the tree with disjoint tiles



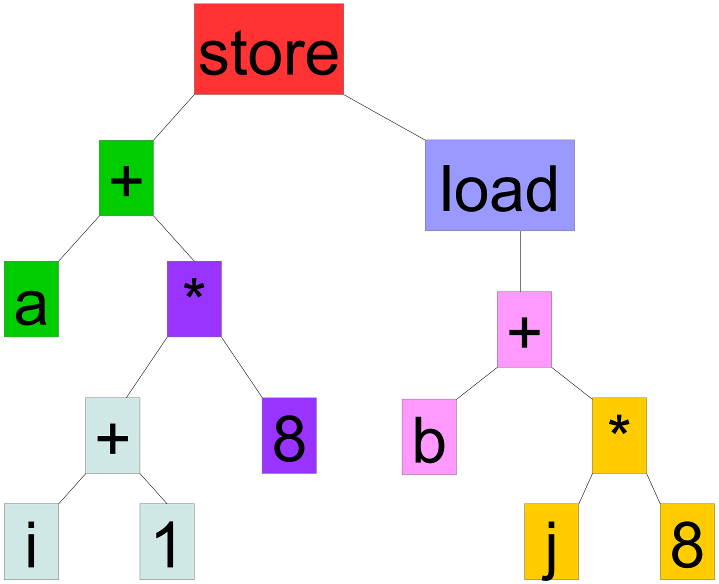
mulc 8, r_j
add r_j, r_b
add 1, r_i
mulc 8,r_i
add r_i, r_a
movem r_b,r_a

NTNU – Trondheim
Norwegian University of Science and Technology

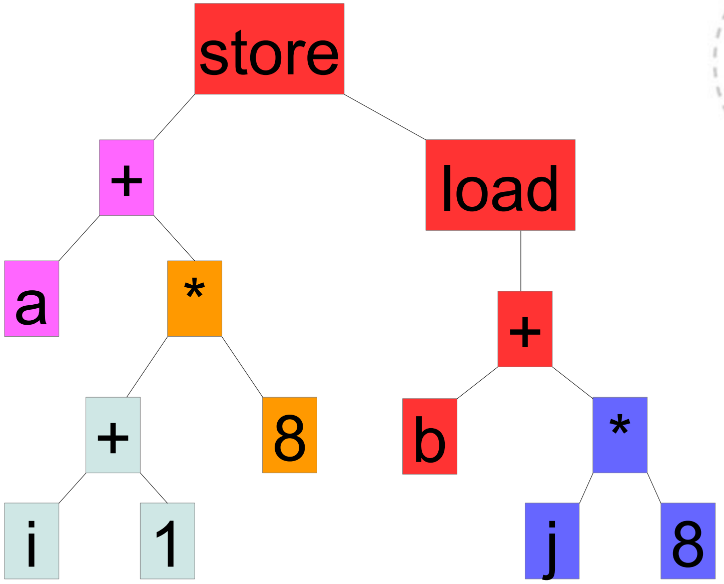
Tilings for comparison

Alternate tilings give different costs

Using store r_b,r_a



Using movex r_j,r_b,r_a



NTNU – Trondheim
Norwegian University of Science and Technology

Better than trees

- If we let common sub-expressions be represented by the same node, the trees become *directed acyclic graphs* (DAGs)
- Separate labels and annotations
  - Label nodes with variales, constants or operators
  - Annotate nodes with variables that hold their value
  - Construct DAG from low-level IR

Basic procedure

- For each instruction in a basic block

  if it's "x = y *op* z"

    find or create a node annotated y

    find or create a node annotated z

    find or create a node labeled *op* with operands y and z

    remove annotation x from everywhere

    add annotation x to the *op* node

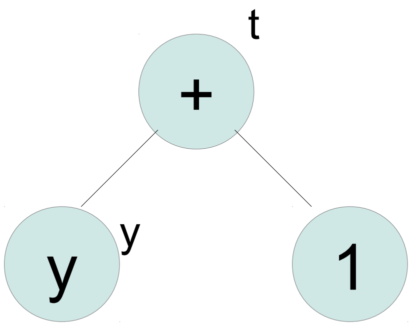
  if it's "x = y"

    find or create a node annotated y

    add annotation x to it

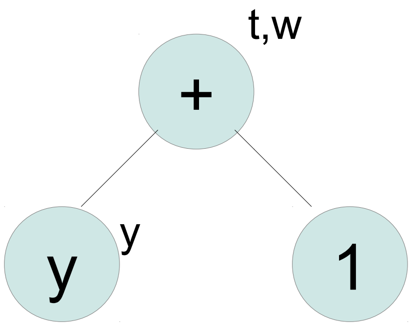
Like so: step 1

t = y + 1
w = y + 1
y = z *t
t = t + 1
z = t * y
w = z



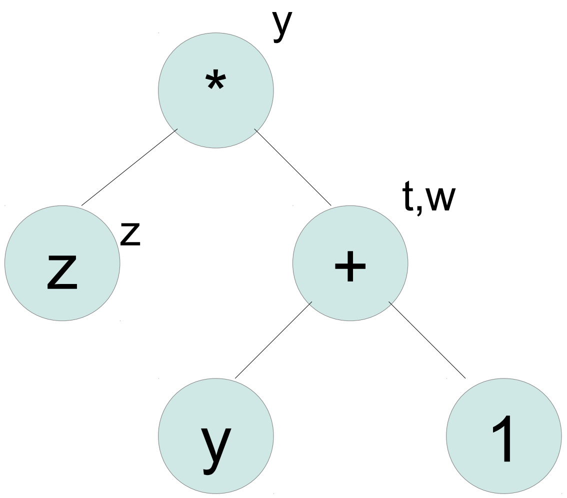
Like so: step 2

```
t = y + 1
w = y + 1
y = z *t
t = t + 1
z = t * y
w = z
```



Like so: step 3

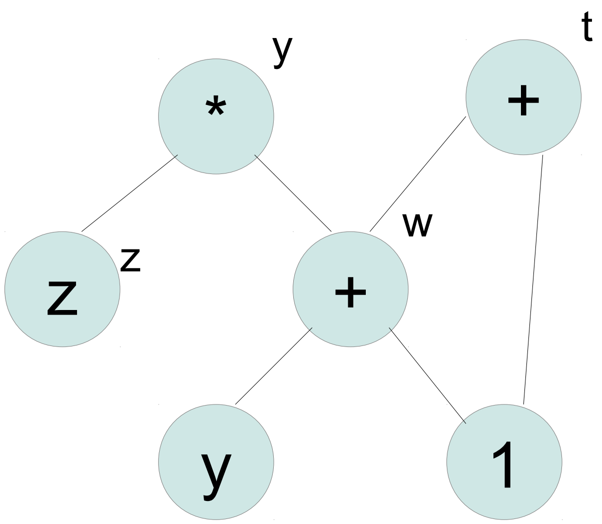
t = y + 1
w = y + 1
y = z *t
t = t + 1
z = t * y
w = z



NTNU – Trondheim
Norwegian University of Science and Technology

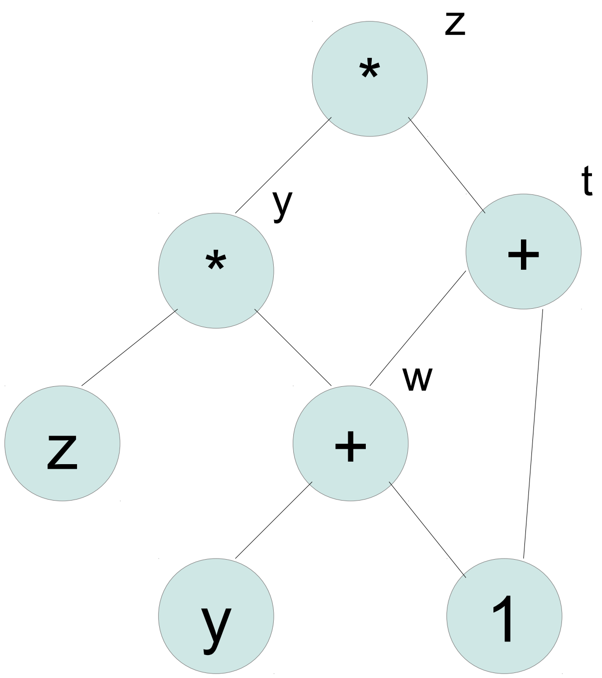
Like so: step 4

t = y + 1
w = y + 1
y = z *t
t = t + 1
z = t * y
w = z



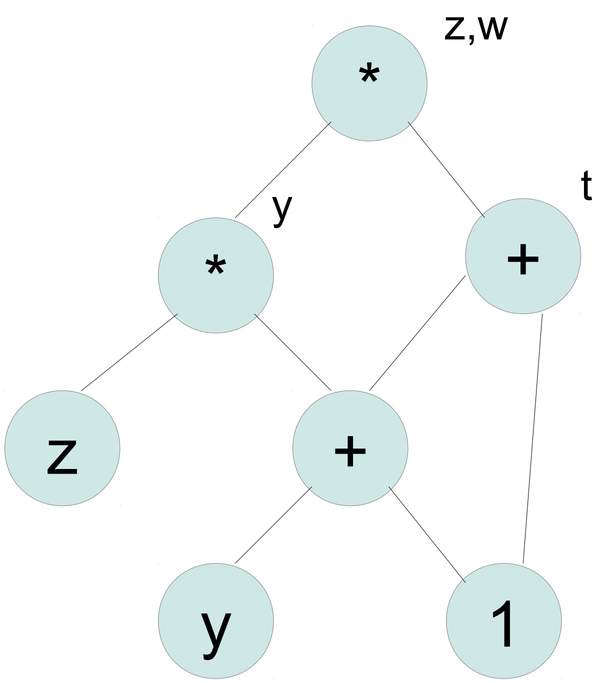
Like so: step 5

t = y + 1
w = y + 1
y = z *t
t = t + 1
z = t * y
w = z



Like so: step 6

t = y + 1
w = y + 1
y = z *t
t = t + 1
z = t * y
w = z



NTNU – Trondheim
Norwegian University of
Science and Technology