Advances in Loop Analysis Frameworks and Optimizations

Adam Nemet & Michael Zolotukhin
Apple

Loop Unrolling

```
for (x = 0; x < 6; x++) {
  foo(x);
}
```

Loop Unrolling

```
for (x = 0; x < 6; x += 2) {
  foo(x);
  foo(x + 1);
}
```

Loop Unrolling

```
{
  foo(x);
  foo(x + 1);
  foo(x + 2);
  foo(x + 3);
  foo(x + 4);
  foo(x + 5);
}
```

Unrolling: Pros and Cons

+ Removes loop overhead

+ Enables other optimizations

– Increases code size

– Increases compile time

– Might regress performance

New Heuristics

- Aim for bigger loops
- Analyze the loop body and predict potential optimization candidates for later passes

Example

```
const int b[50] = {1, 0, 0, …, 0, 0};

int foo(int *a) {
  int r = 0;

  for (int i = 0; i < 50; i++) {
    r += a[i] * b[i];
  }

  return r;
}
```

Example

```
const int b[50] = {1, 0, 0, …, 0, 0};

int foo(int *a) {
  int r = 0;
  r += a[0] * b[0];
  r += a[1] * b[1];
  …
  r += a[48] * b[48];
  r += a[49] * b[49];
  return r;
}
```

Example

```
const int b[50] = {1, 0, 0, …, 0, 0};

int foo(int *a) {
  int r = 0;
  r += a[0] * 1;
  r += a[1] * 0;
  …
  r += a[48] * 0;
  r += a[49] * 0;
  return r;
}
```

Example

```
const int b[50] = {1, 0, 0, …, 0, 0};

int foo(int *a) {
  return a[0];
}
```

Analyzing Loop

- Simulate the loop execution instruction by instruction, iteration by iteration
- Try to predict possible simplifications of every instruction
- Compute accurate costs of the original loop and its unrolled version

How It Works

Iteration 0

```
%r = 0
loop:
  %y = b[i]
  %x = a[i]
  %t = %x * %y
  %r = %r + %t
  %i = %i + 1
  %cmp = %i < 50
  br %cmp, loop, exit
exit:
  ret %r
```

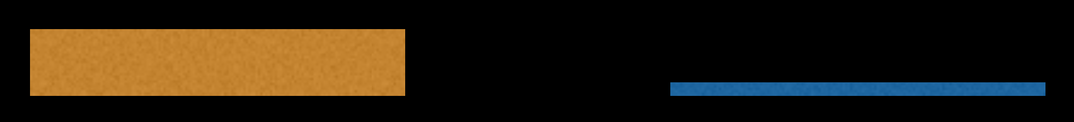
Original loop cost

Unrolled loop cost

How It Works

Iteration 0

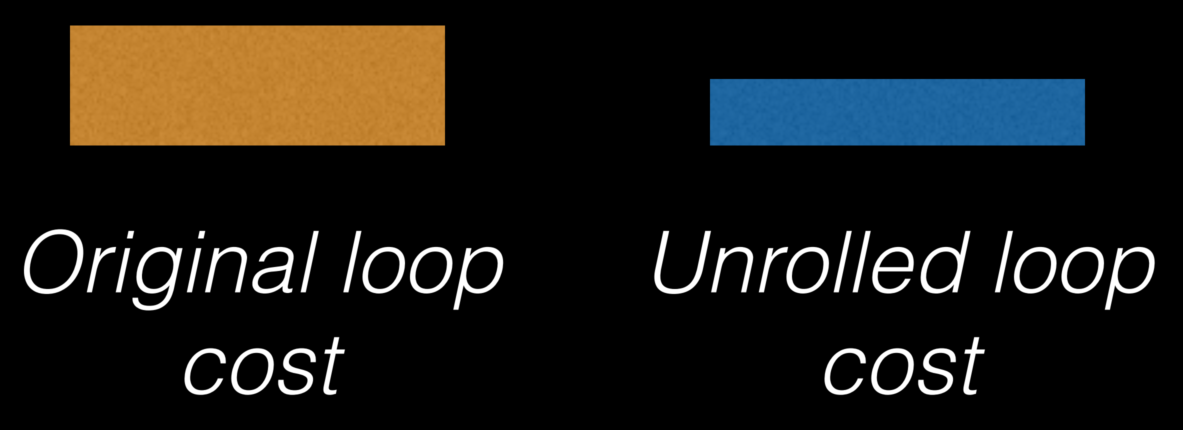
```
%r = 0
loop:
  %y = b[i] = 1
  %x = a[i]
  %t = %x * %y
  %r = %r + %t
  %i = %i + 1
  %cmp = %i < 50
  br %cmp, loop, exit
exit:
  ret %r
```



How It Works

Iteration 0

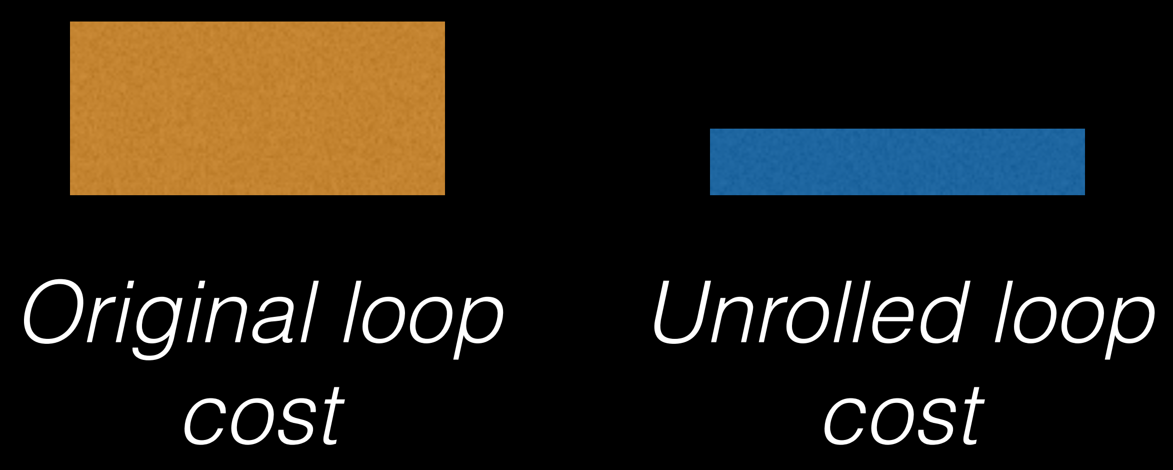
```
%r = 0
loop:
  %y = b[i] = 1
  %x = a[i]
  %t = %x * %y
  %r = %r + %t
  %i = %i + 1
  %cmp = %i < 50
  br %cmp, loop, exit
exit:
  ret %r
```



How It Works

Iteration 0

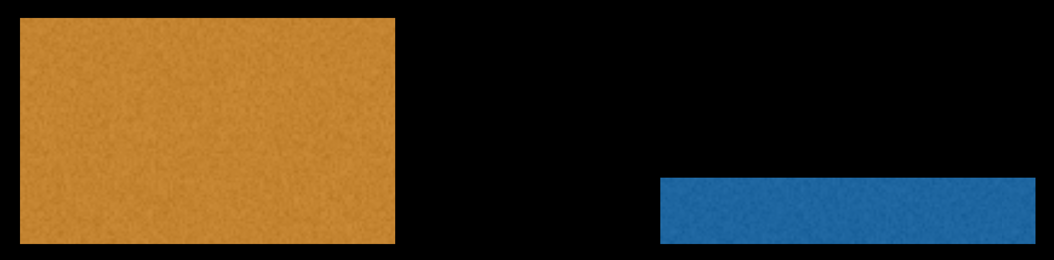
```
%r = 0
loop:
  %y = b[i] = 1
  %x = a[i]
  %t = %x * %y = %x
  %r = %r + %t
  %i = %i + 1
  %cmp = %i < 50
  br %cmp, loop, exit
exit:
  ret %r
```



How It Works

Iteration 0

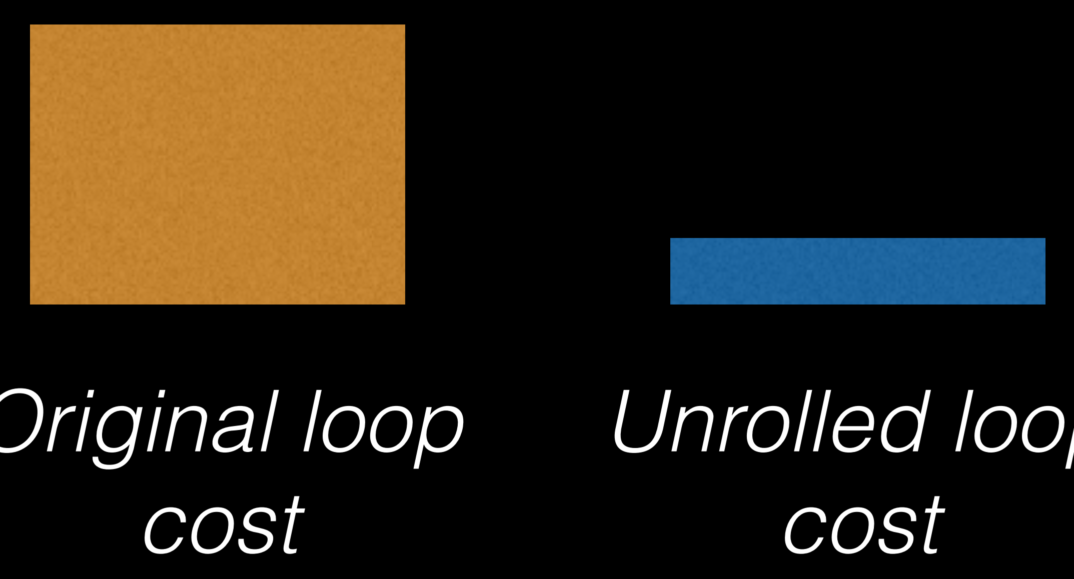
%r = 0
loop:
  %y = ~~b[i]~~ = 1
  %x = a[i]
  %t = ~~%x * %y~~ = %x
  %r = ~~%r + %t~~ = %t
  %i = %i + 1
  %cmp = %i < 50
  br %cmp, loop, exit
exit:
  ret %r



How It Works

Iteration 0

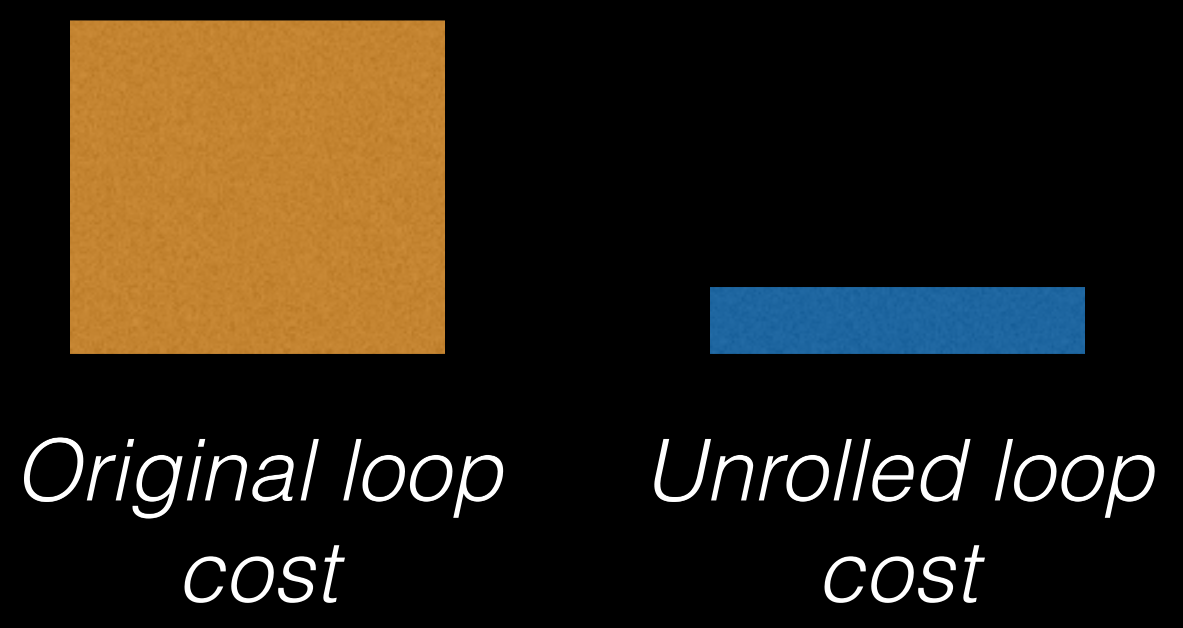
%r = 0
loop:
  %y = ~~b[i]~~ = 1
  %x = a[i]
  %t = ~~%x * %y~~ = %x
  %r = ~~%r + %t~~ = %t
→ %i = ~~%i + 1~~ = 1
  %cmp = %i < 50
  br %cmp, loop, exit
exit:
  ret %r



How It Works

Iteration 0

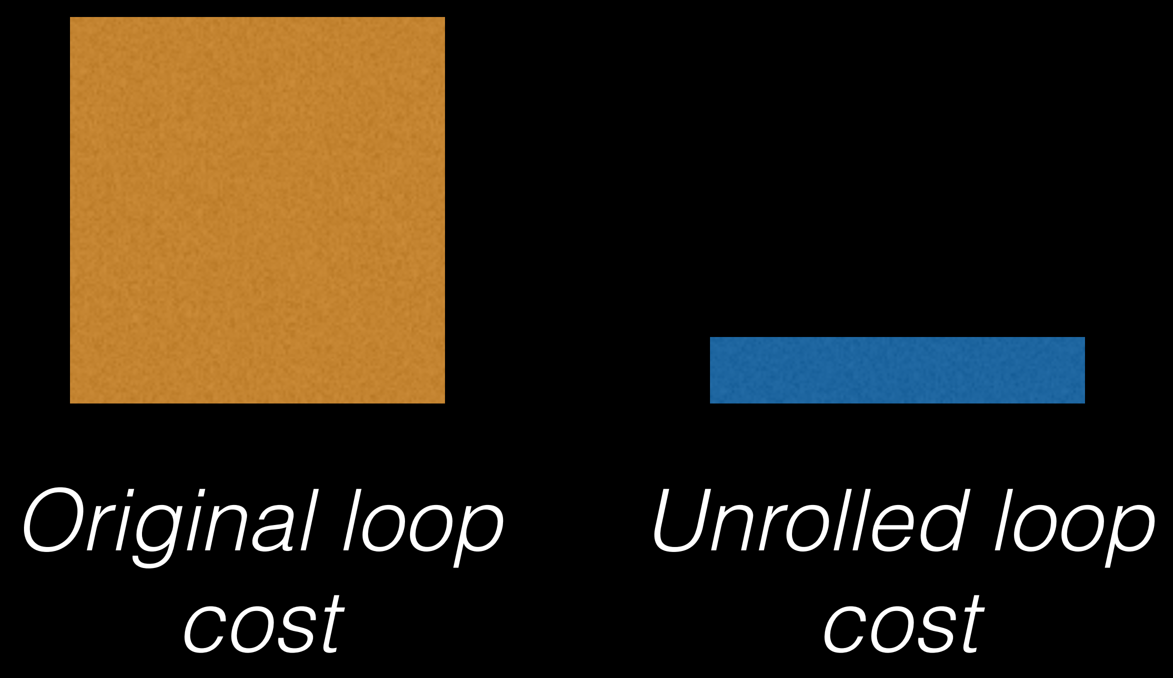
```
%r = 0
loop:
  %y = b[i] = 1
  %x = a[i]
  %t = %x * %y = %x
  %r = %r + %t = %t
  %i = %i + 1 = 1
  %cmp = %i < 50 = true
  br %cmp, loop, exit
exit:
  ret %r
```



How It Works

Iteration 0

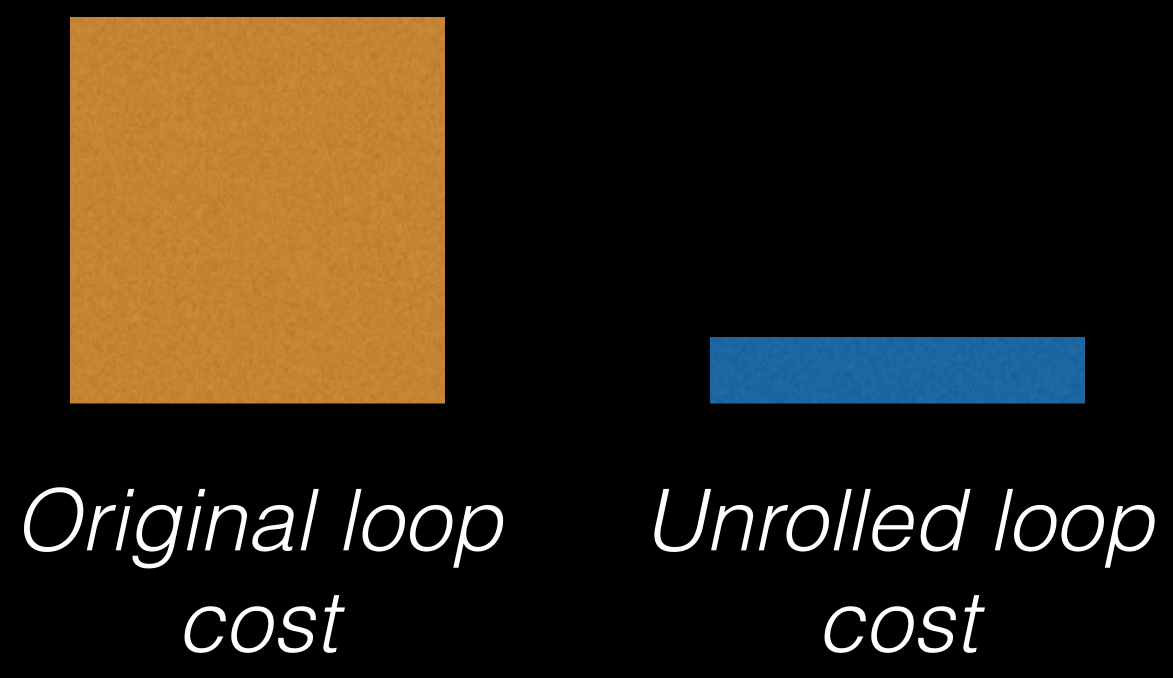
%r = 0
loop:
  %y = ~~b[i]~~ = 1
  %x = a[i]
  %t = ~~%x * %y~~ = %x
  %r = ~~%r + %t~~ = %t
  %i = ~~%i + 1~~ = 1
  %cmp = ~~%i < 50~~ = true
→ br %cmp, loop, exit
exit:
  ret %r



How It Works

Iteration 1

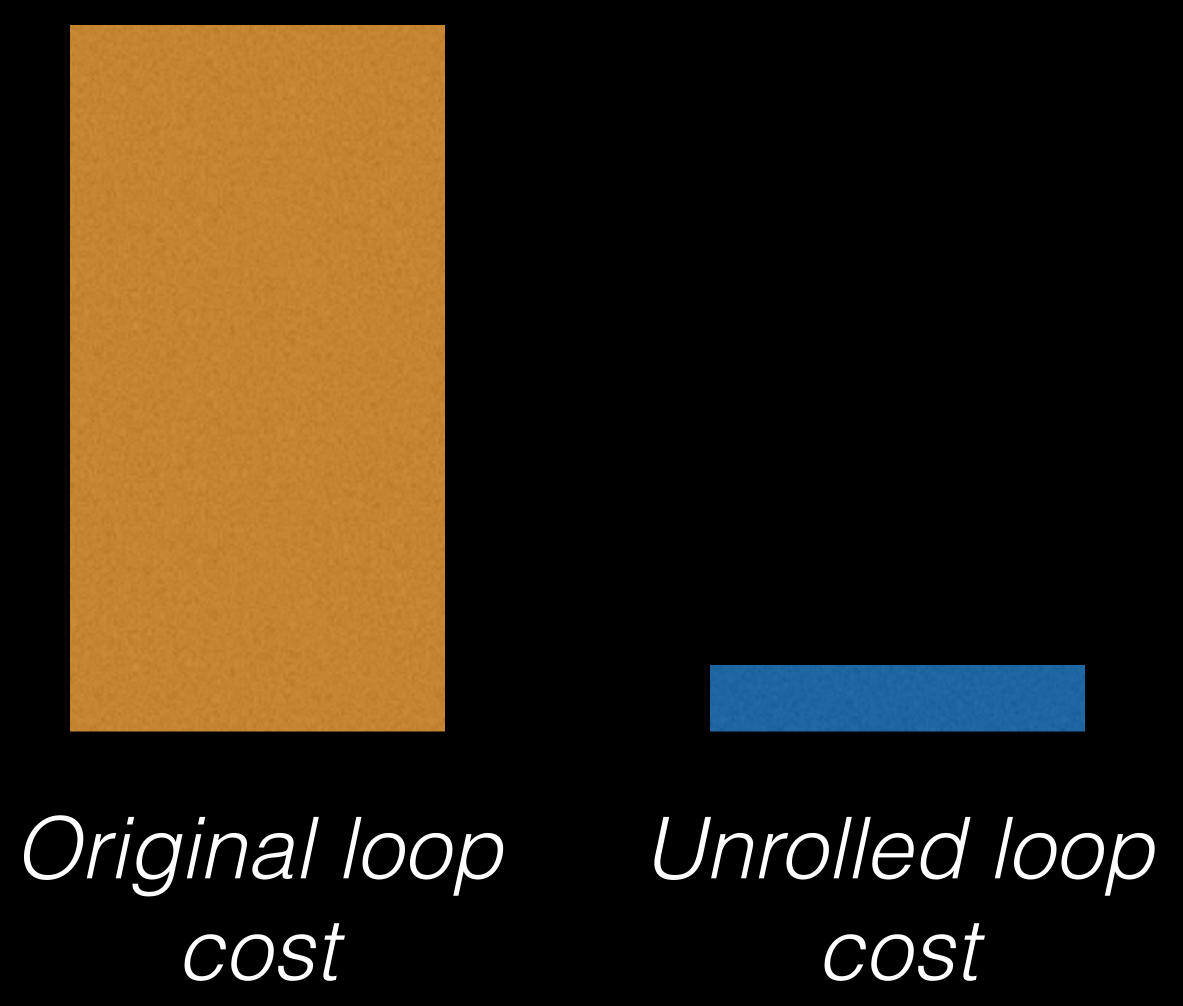
```
%r = 0
loop:
  %y = b[i]
  %x = a[i]
  %t = %x * %y
  %r = %r + %t
  %i = %i + 1
  %cmp = %i < 50
  br %cmp, loop, exit
exit:
  ret %r
```



How It Works

Iteration 1

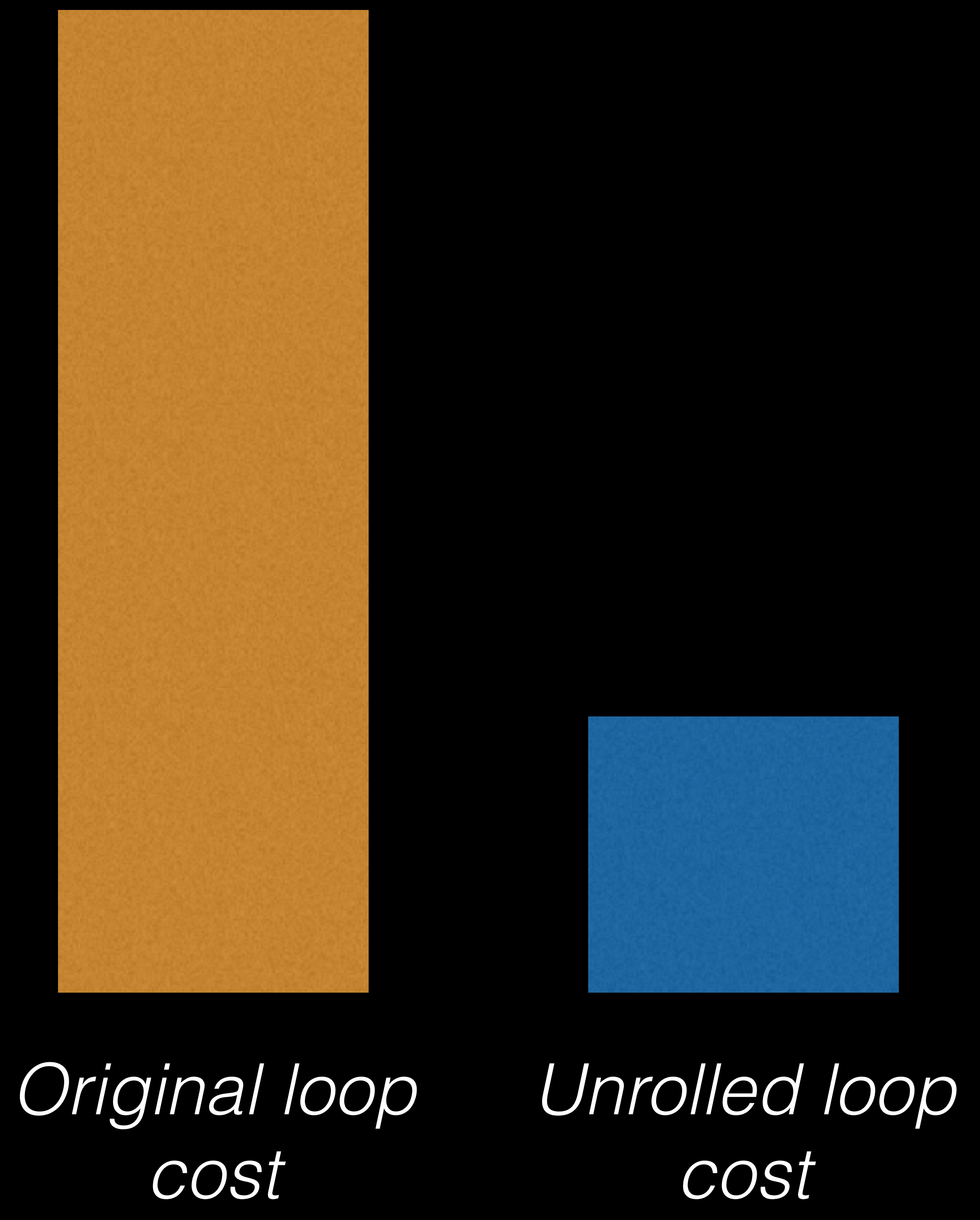
```
%r = 0
loop:
  %y = b[i] = 0
  %x = a[i]
  %t = %x * %y = 0
  %r = %r + %t = %r
  %i = %i + 1  = 2
  %cmp = %i < 50 = true
  br %cmp, loop, exit
exit:
  ret %r
```



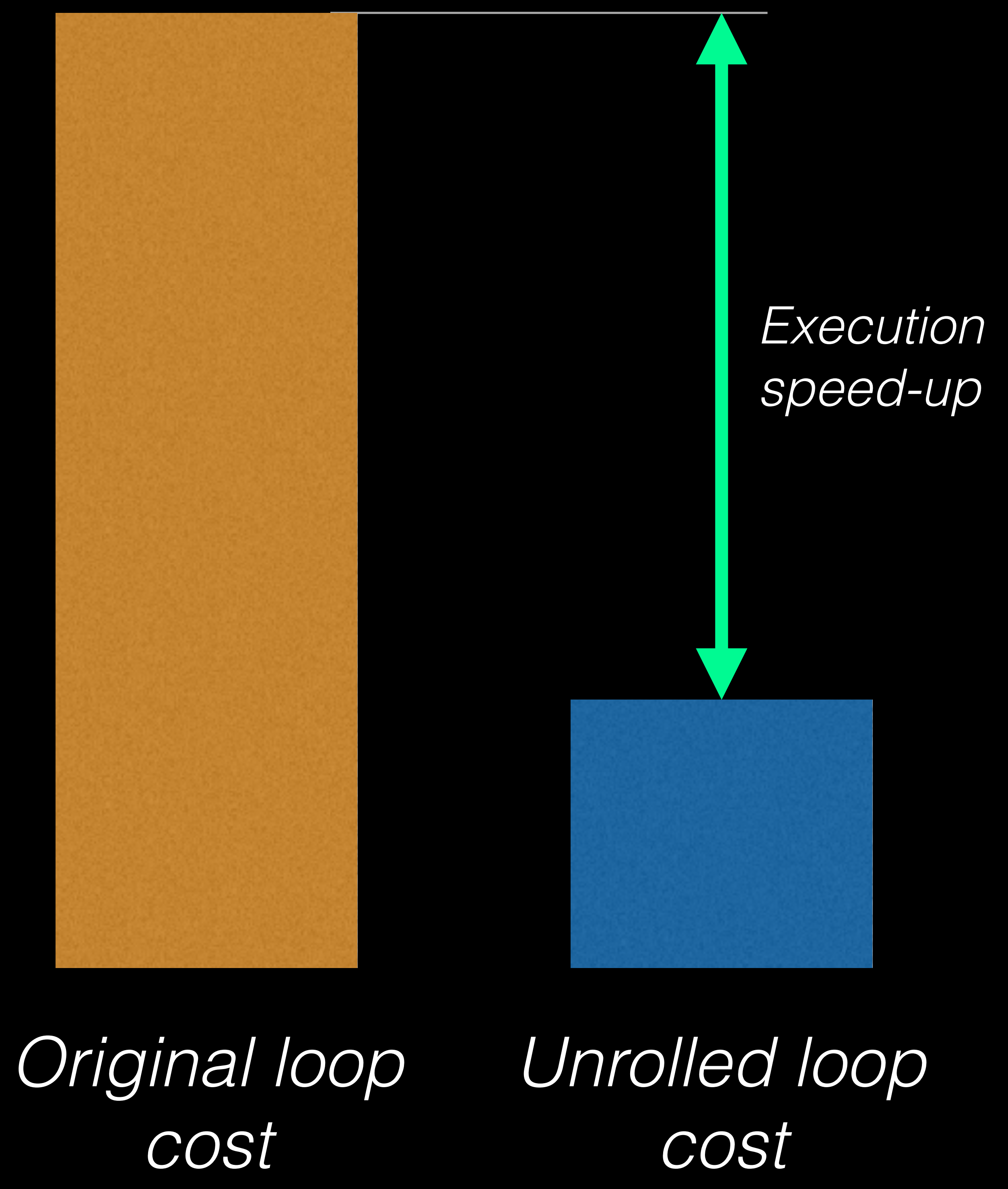
How It Works

Iteration 49

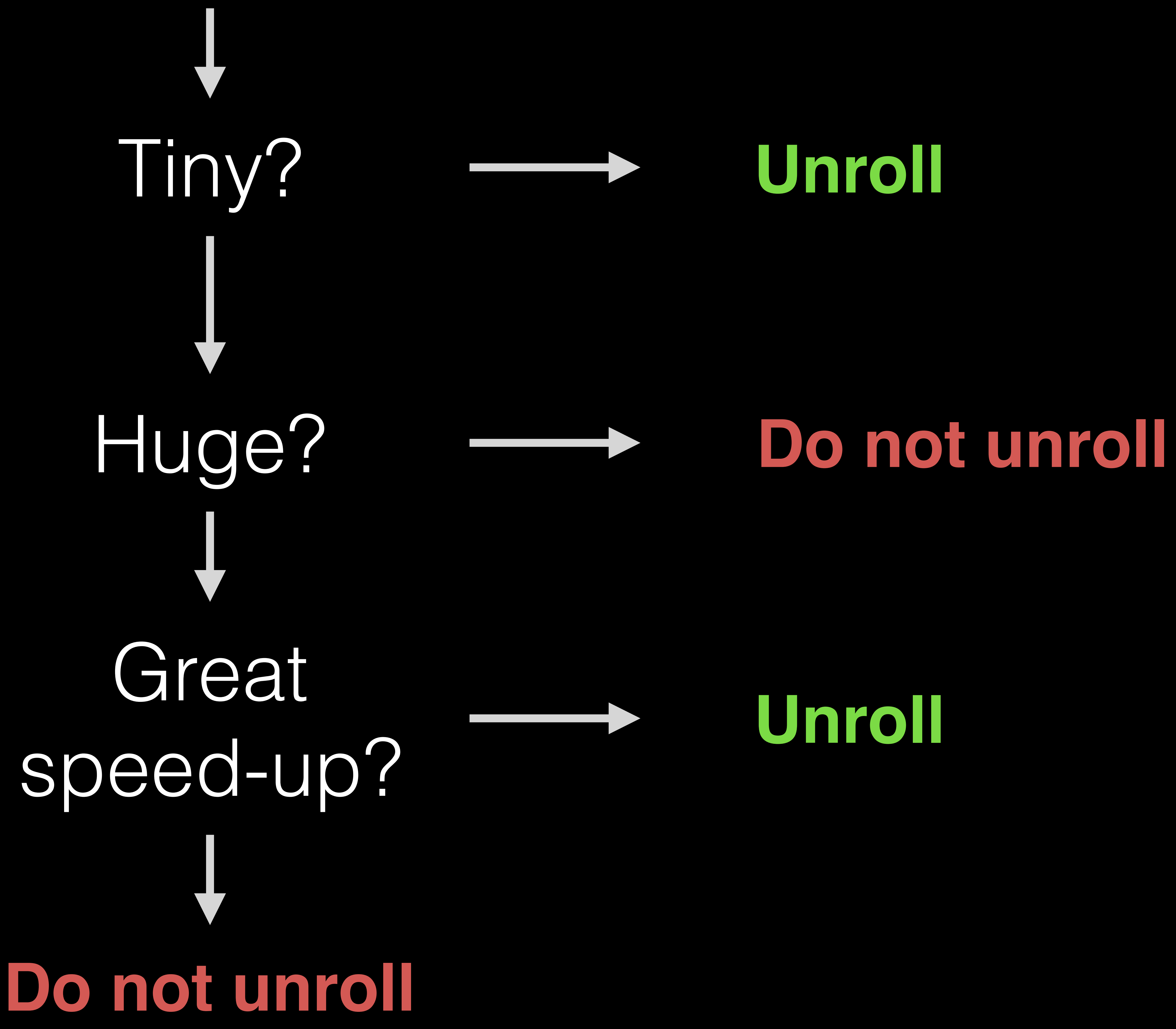
```
%r = 0
loop:
  %y = b[i]
  %x = a[i]
  %t = %x * %y
  %r = %r + %t
  %i = %i + 1
  %cmp = %i < 50
  br %cmp, loop, exit
exit:
  ret %r
```



How It Works



How It Works


Tiny?
Unroll
Huge?
Do not unroll
Great speed-up?
Unroll
Do not unroll

Unrolling: Results

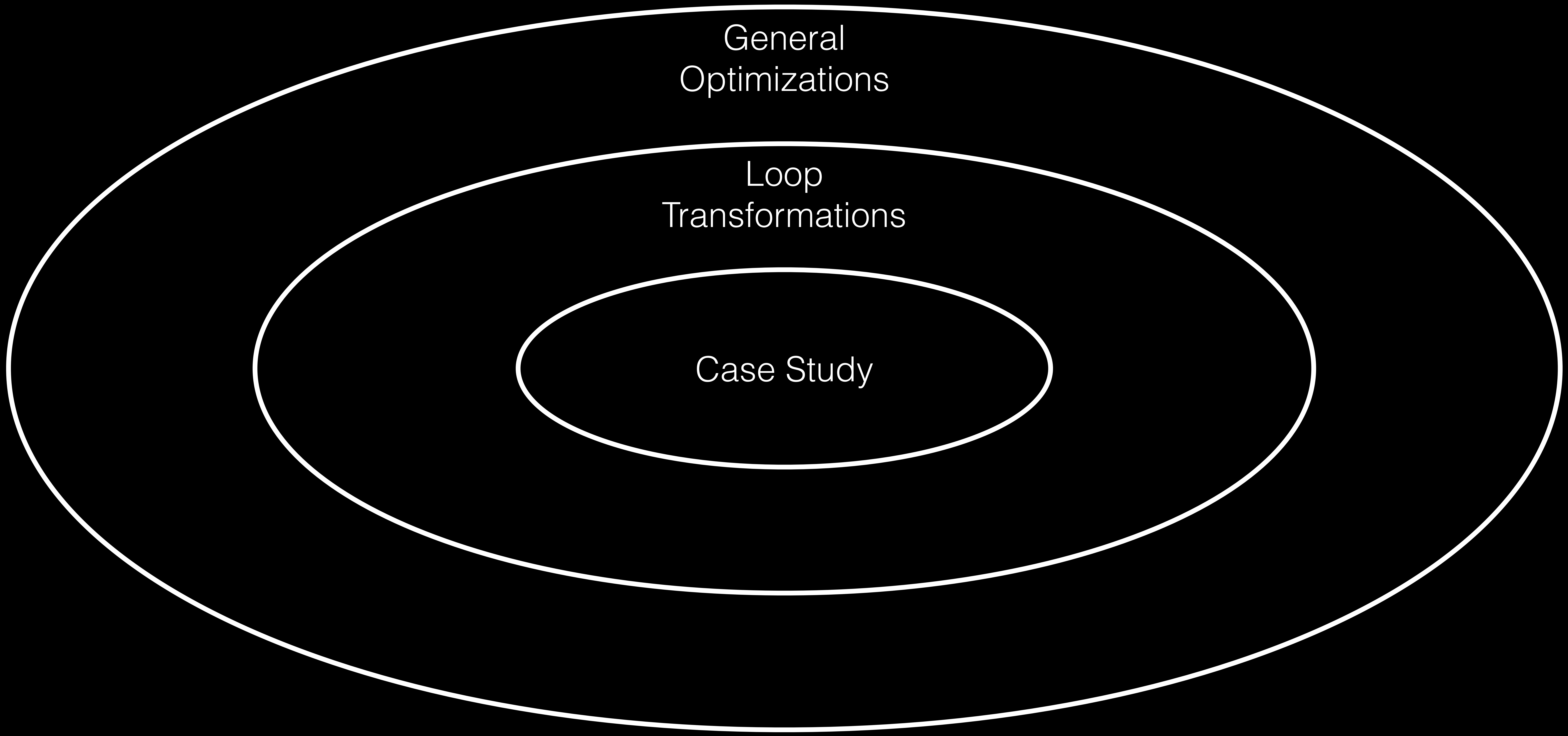
- Up to 70% performance gains on kernels
- Few performance gains across various testsuites
- No performance regressions
- Some compile time regressions

Unrolling: Future Work

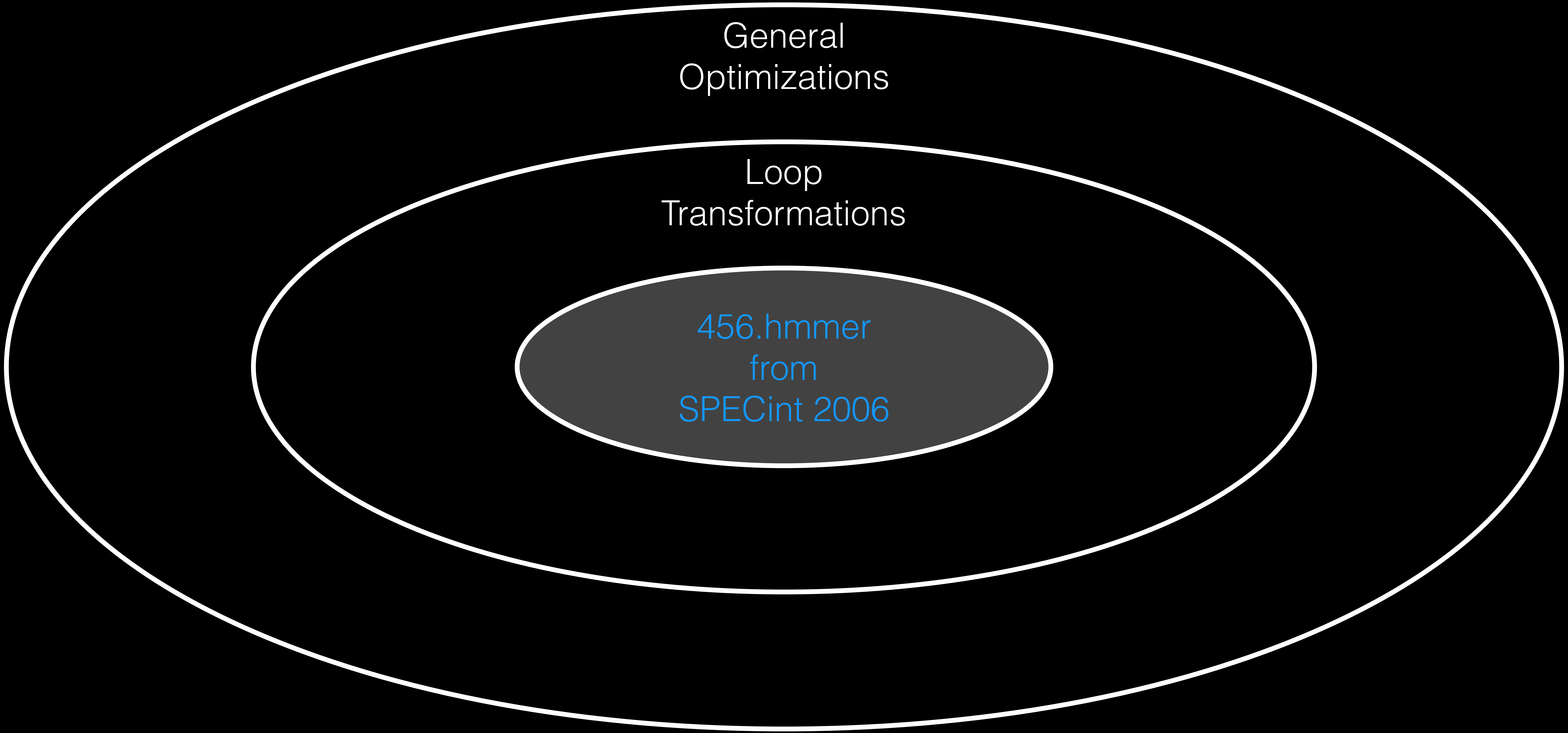
- Enable new heuristics by default after investigating compile time regressions
- Model other optimizations
- Find trip count

Next Up

Approach



Approach


General
Optimizations
Loop
Transformations
456.hmmer
from
SPECint 2006

Case Study

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;

  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

Case Study

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;

  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

What does this loop do?

Case Study

```
mc[k] = mpp[k-1] + tpmm[k-1];
if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
mc[k] += ms[k];
if (mc[k] < -INFTY) mc[k] = -INFTY;
```

Case Study

```
mc[k] = mpp[k-1] + tpmm[k-1];
if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
mc[k] += ms[k];
if (mc[k] < -INFTY) mc[k] = -INFTY;
```

Case Study

```
dc[k] = dc[k-1] + tpdd[k-1];
if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
if (dc[k] < -INFTY) dc[k] = -INFTY;
```

Case Study

```
dc[k] = dc[k-1] + tpdd[k-1];
if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
if (dc[k] < -INFTY) dc[k] = -INFTY;
```

Case Study

```
if (k < M) {
  ic[k] = mpp[k] + tpmi[k];
  if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
  ic[k] += is[k];
  if (ic[k] < -INFTY) ic[k] = -INFTY;
}
```

Case Study

```
if (k < M) {
  ic[k] = mpp[k] + tpmi[k];
  if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
  ic[k] += is[k];
  if (ic[k] < -INFTY) ic[k] = -INFTY;
}
```

Case Study

How can we optimize this loop?

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;

  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
sk}
}
```

Can We Vectorize It?

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;

  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
sk}
}
```

No!

Can We Vectorize It?

```
dc[k] = dc[k-1] + tpdd[k-1];
if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
if (dc[k] < -INFTY) dc[k] = -INFTY;
```

Can We Vectorize It?

```
dc[k] = dc[k-1] + tpdd[k-1];
```

Can We Vectorize It?

```
t     = dc[k-1] + tpdd[k-1];

dc[k] = t;
```

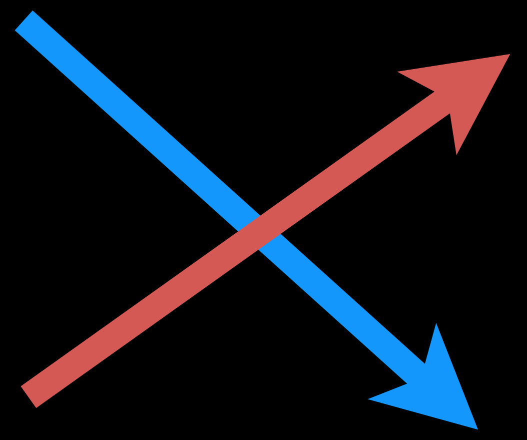
Can We Vectorize It?

```
t      = dc[k-1] + tpdd[k-1];

dc[k] = t;
```

Can We Vectorize It?

```
t      = dc[k-1] + tpdd[k-1];
```



```
dc[k] = t;
```

Can We Vectorize It?

Iteration K:

```
t      = dc[k-1] + tpdd[k-1];



dc[k] = t;
```

Iteration K+1:

```
t2     = dc[k] + tpdd[k];



dc[k+1] = t2;
```

Can We Vectorize It?

Iteration K:

Iteration K+1:

```
t      = dc[k-1] + tpdd[k-1];
    t2      = dc[k] + tpdd[k];


dc[k] = t;
    dc[k+1] = t2;
```

Can We Vectorize It?

```
dc[k] = dc[k-1] + tpdd[k-1];
if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
if (dc[k] < -INFTY) dc[k] = -INFTY;
```

Case Study

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;

  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

Non-vectorizable

Case Study

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;
```

Vectorizable

```
  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;
```

Non-vectorizable

```
  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

Case Study

```
if (k < M) {
  ic[k] = mpp[k] + tpmi[k];
  if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
  ic[k] += is[k];
  if (ic[k] < -INFTY) ic[k] = -INFTY;
}
```

Case Study

```
if (k < M) {
  ic[k] = mpp[k] + tpmi[k];
  if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
  ic[k] += is[k];
  if (ic[k] < -INFTY) ic[k] = -INFTY;
}
```

Case Study

Vectorizable

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;
```

Non-vectorizable

```
  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
sk}
}
```

Case Study

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;
}
```

Vectorizable

```
for (k = 1; k <= M; k++) {
  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
sk}
}
```

Non-vectorizable

Plan

- Distribute loop
- Let LoopVectorizer vectorize top loop

-> Partial Loop Vectorization

Loop Distribution

Pros and Cons

+ Partial loop vectorization

+ Improve memory access pattern:

  - Cache associativity
  - Number of HW prefetcher streams

+ Reduce spilling

- Loop overhead

- Instructions duplicated across new loops

- Instruction-level parallelism

Legality

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;
}
for (k = 1; k <= M; k++) {
  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

Loop Dependence Analysis	Run-time Alias Checks

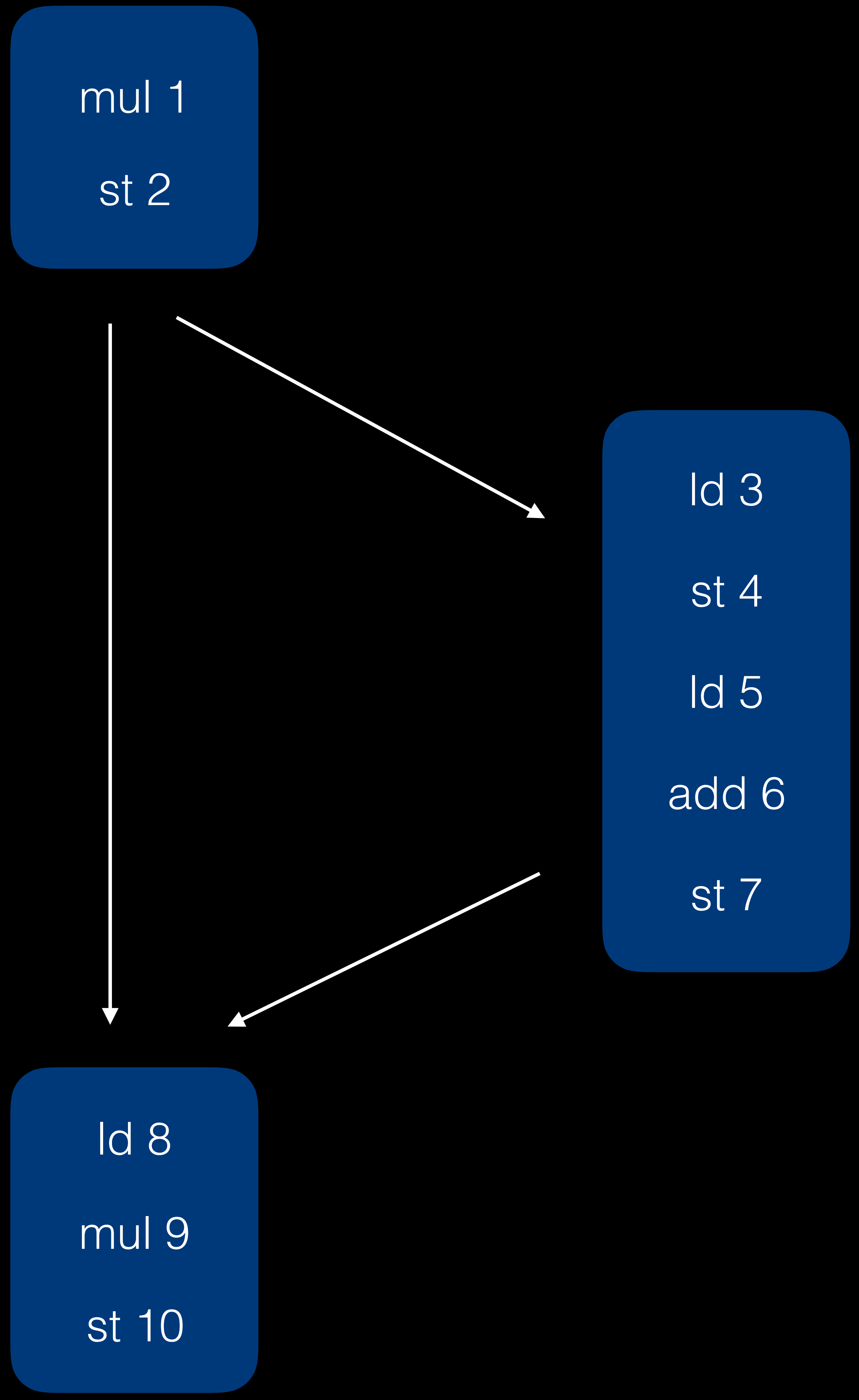
Loop Access Analysis

- Born from the Loop Vectorizer
- Generalized as new analysis pass
- Computed on-demand and cached
- New Loop Versioning utility

Algorithm

- Light-weight
  - Uses only LoopAccessAnalysis
  - No Program Dependence Graph
  - No Control Dependence
- Inner loops only
- Different from textbook algorithm
  - No reordering of memory operations

Algorithm



Algorithm

mul 1

st 2

ld 3

st 4

ld 5

add 6

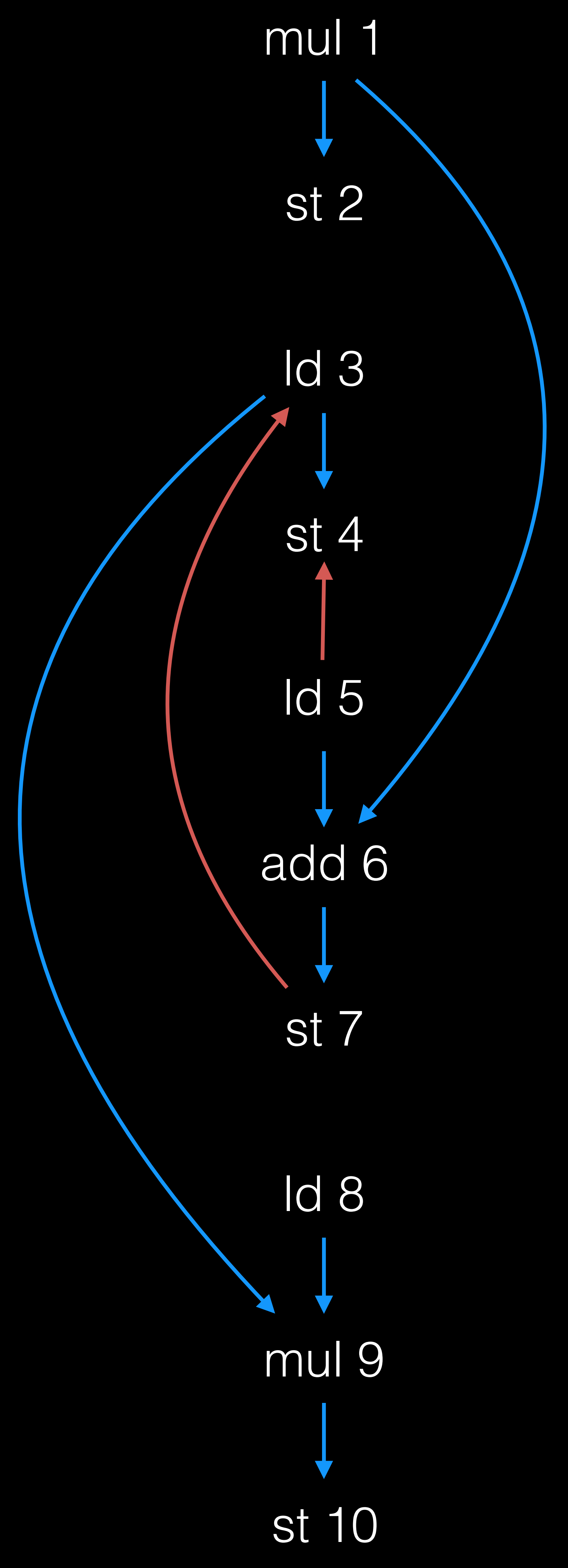
st 7

ld 8

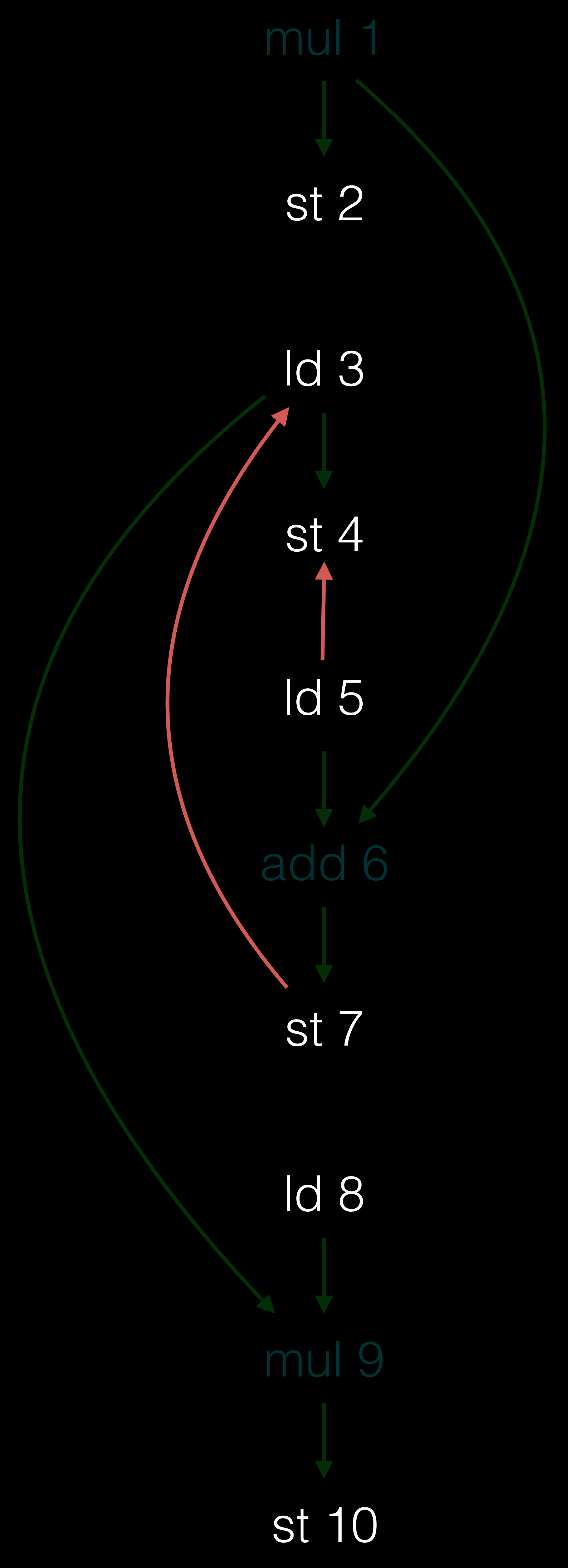
mul 9

st 10

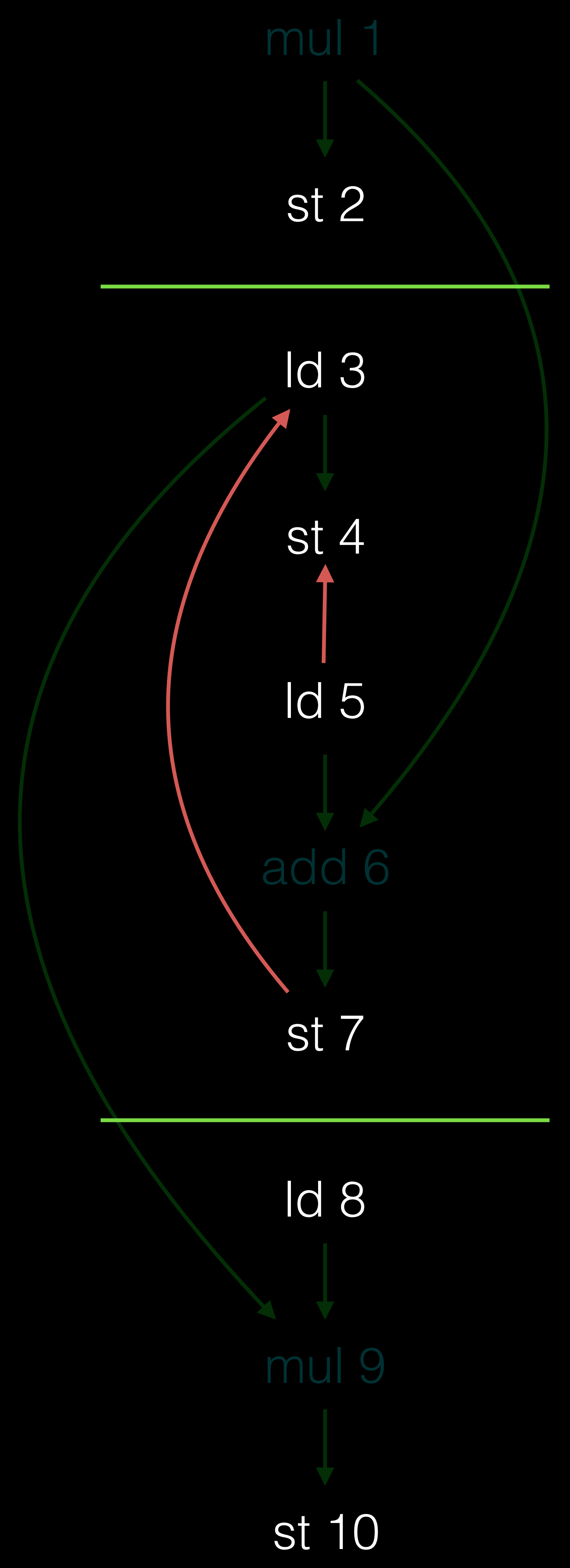
Algorithm



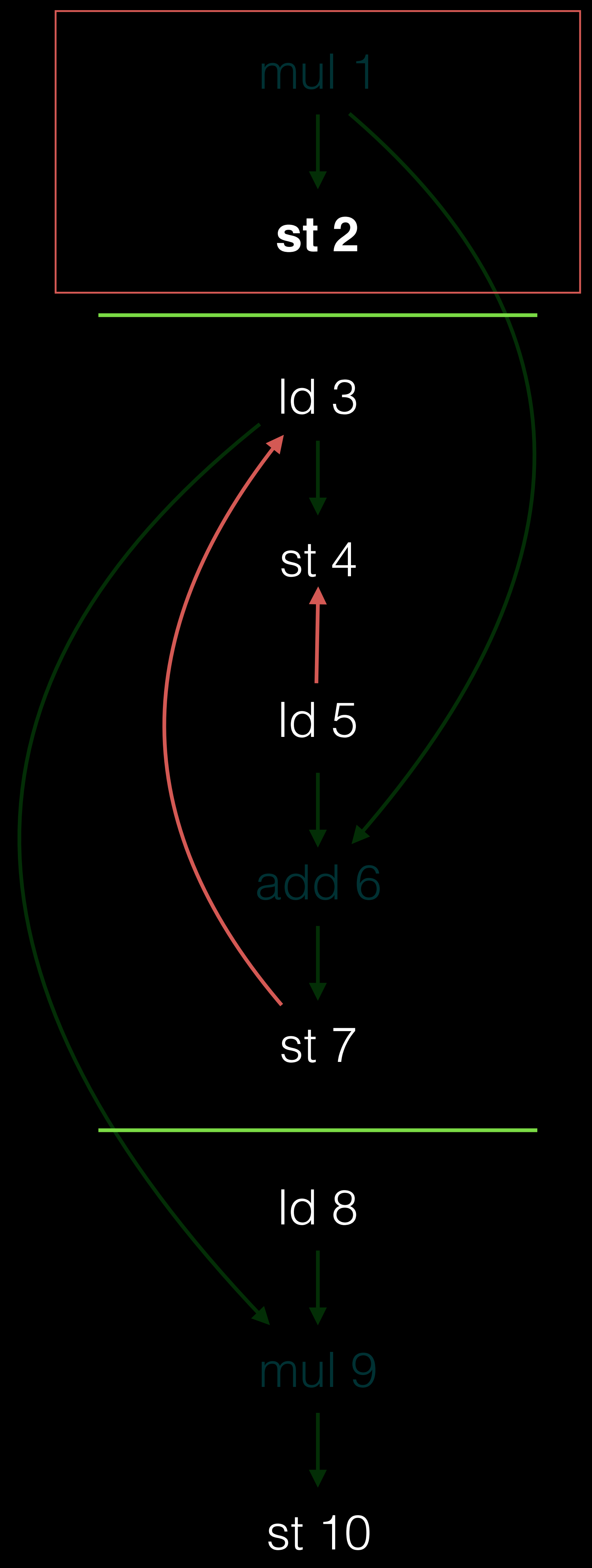
Algorithm



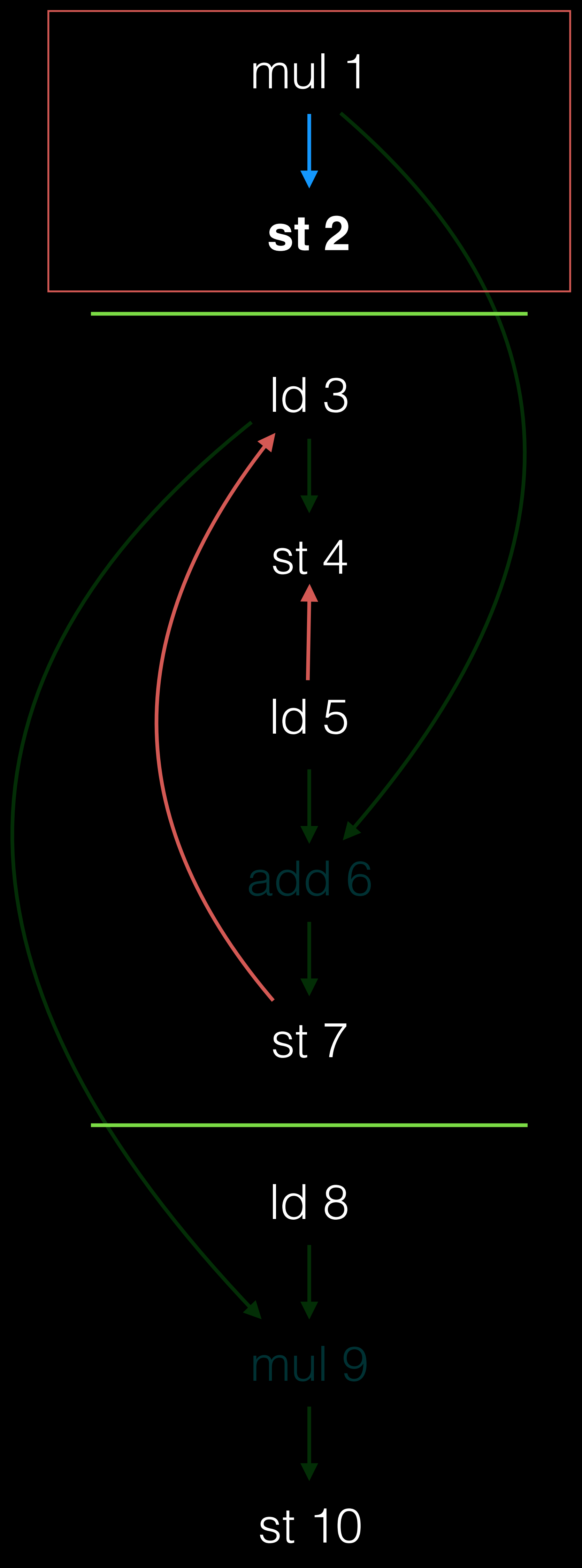
Algorithm



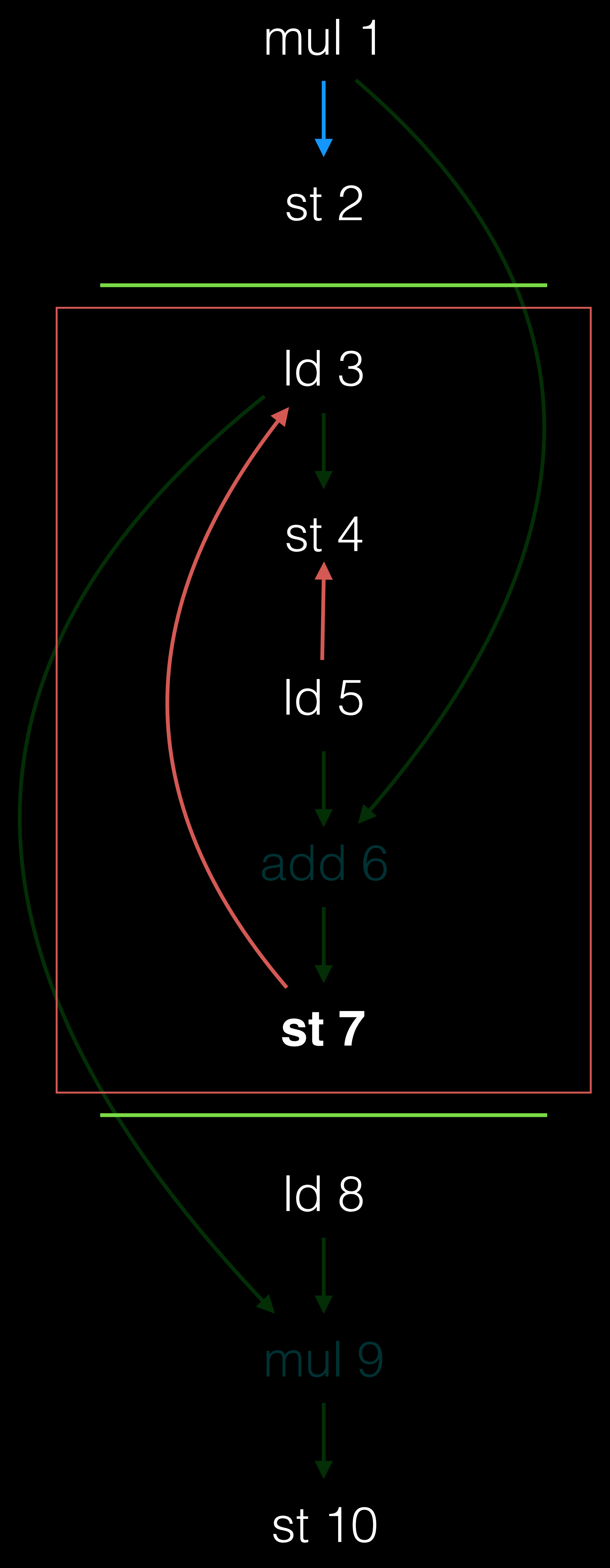
Algorithm



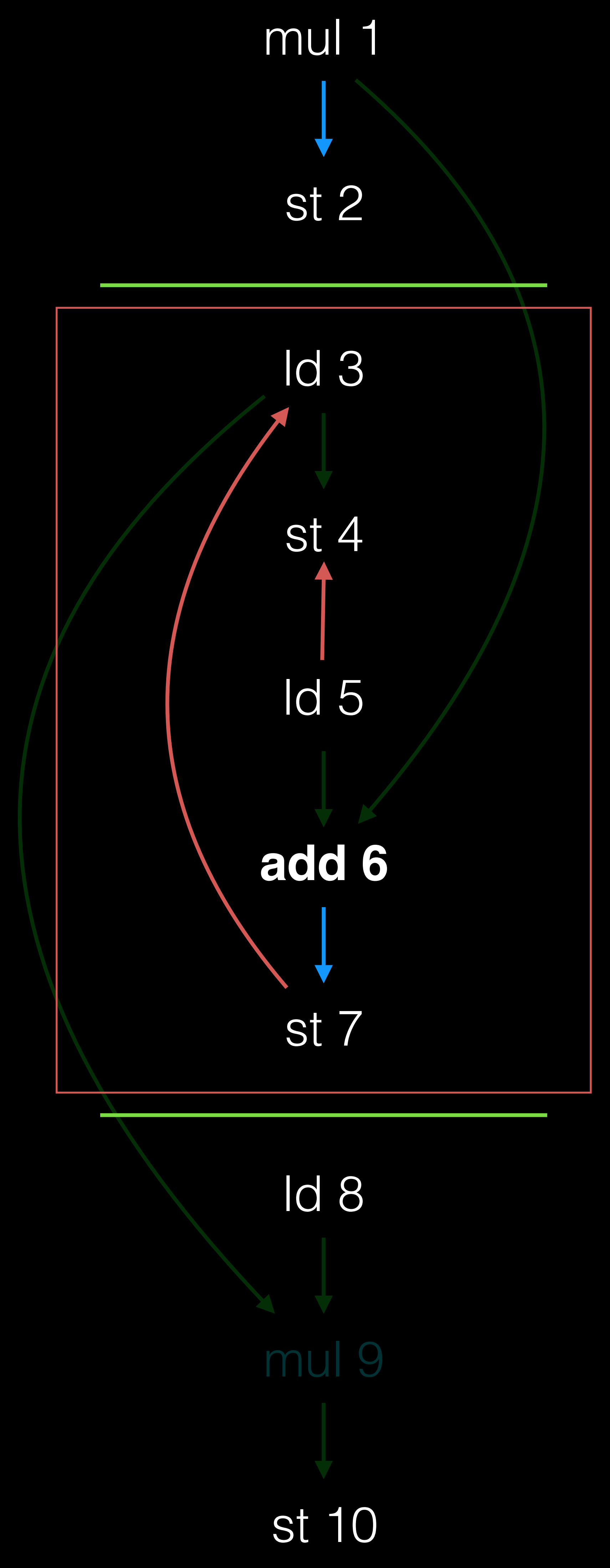
Algorithm



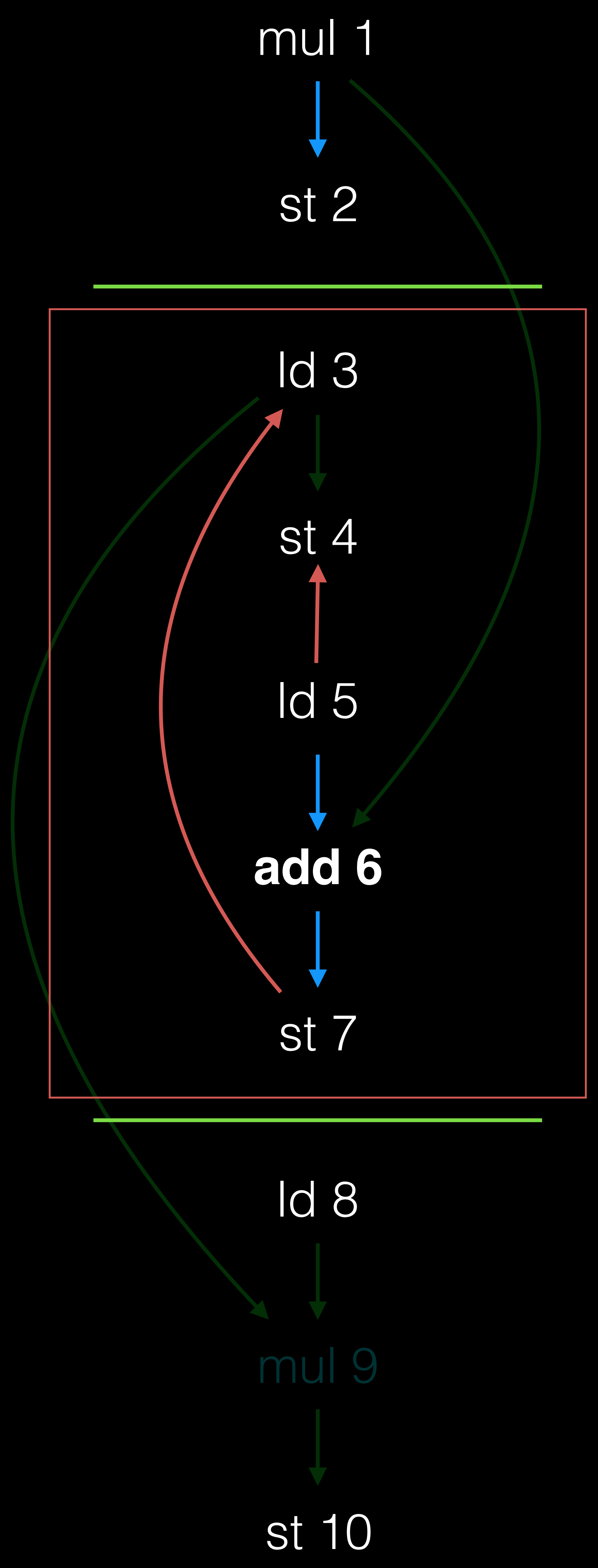
Algorithm



Algorithm

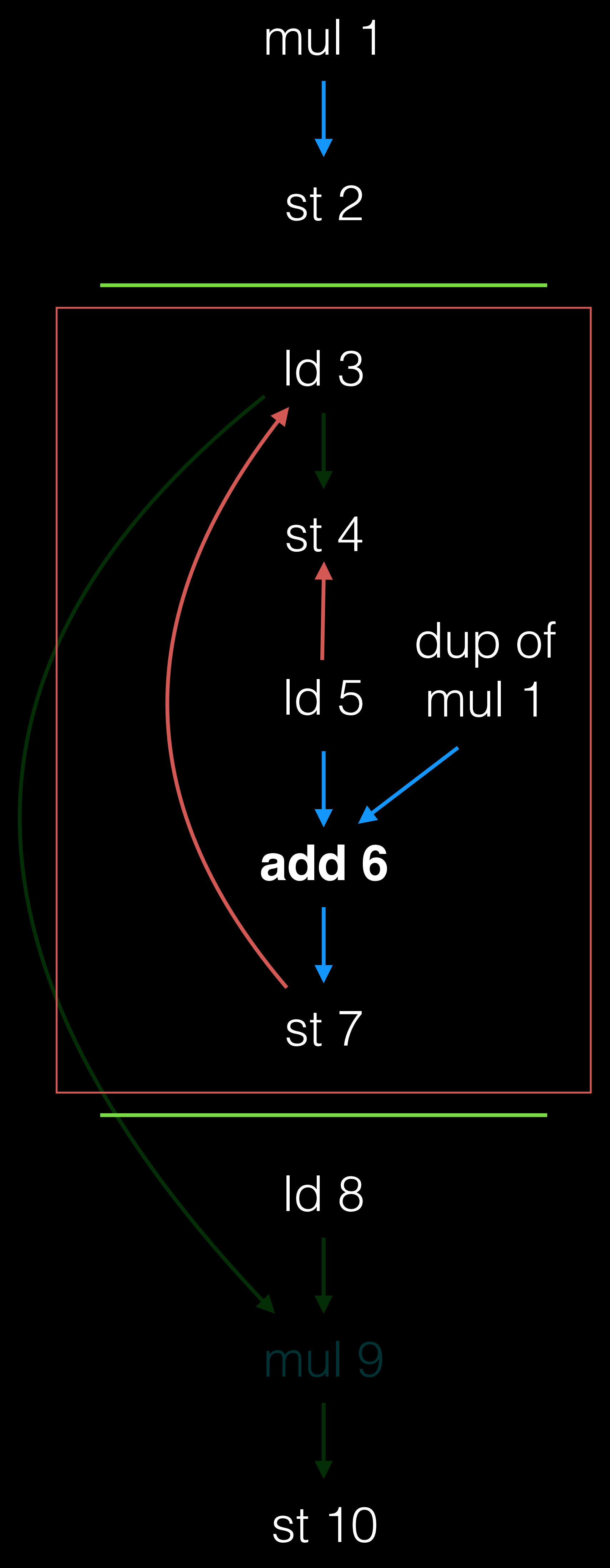


Algorithm

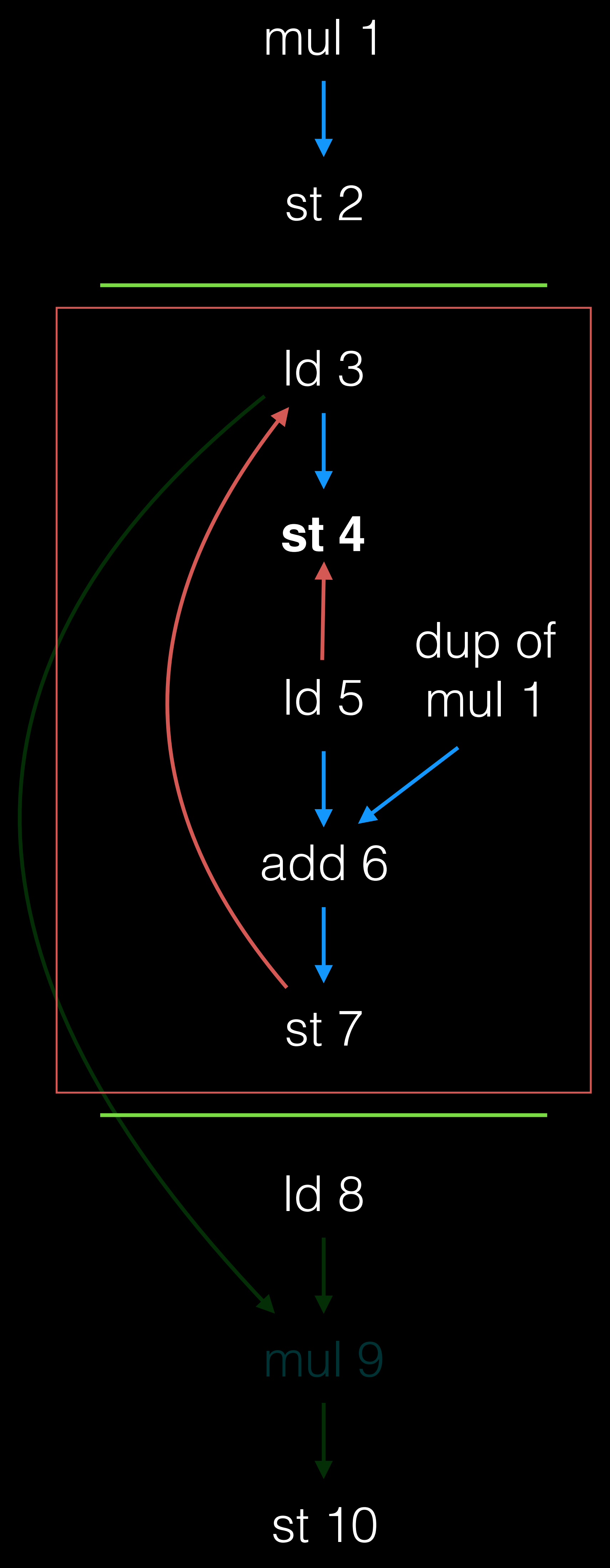


Algorithm

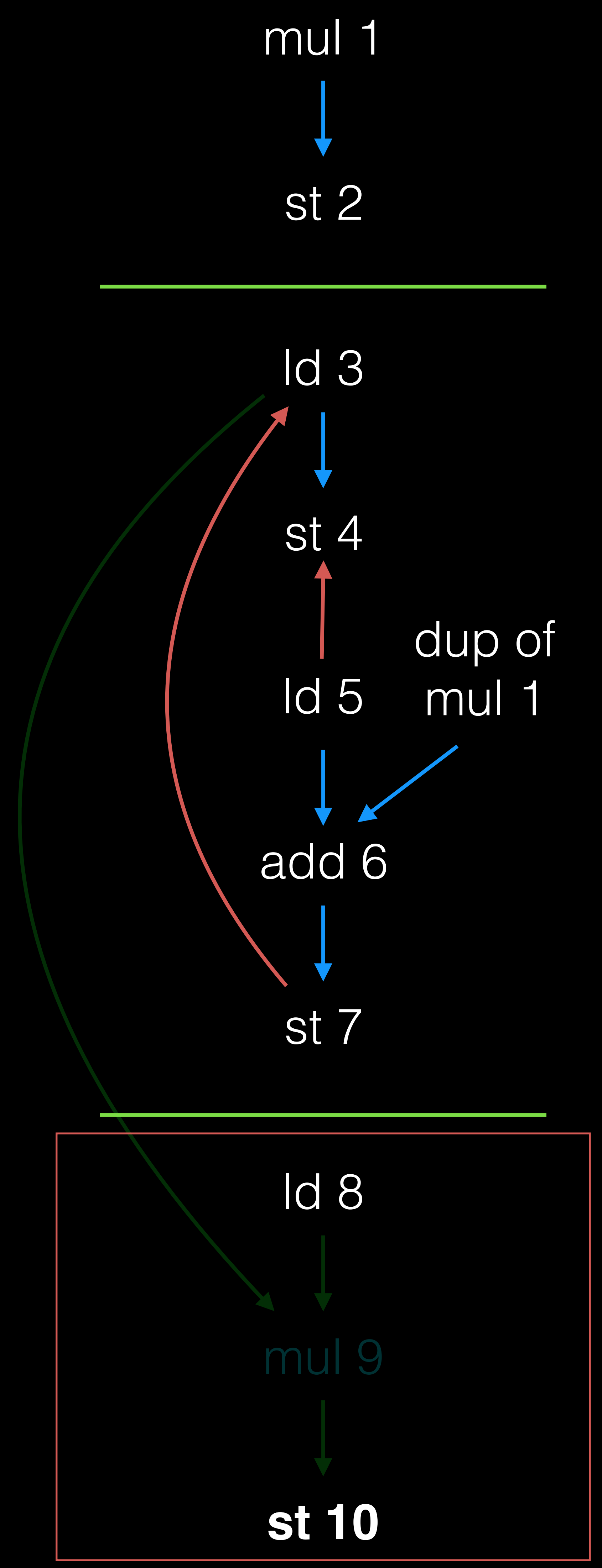
Algorithm



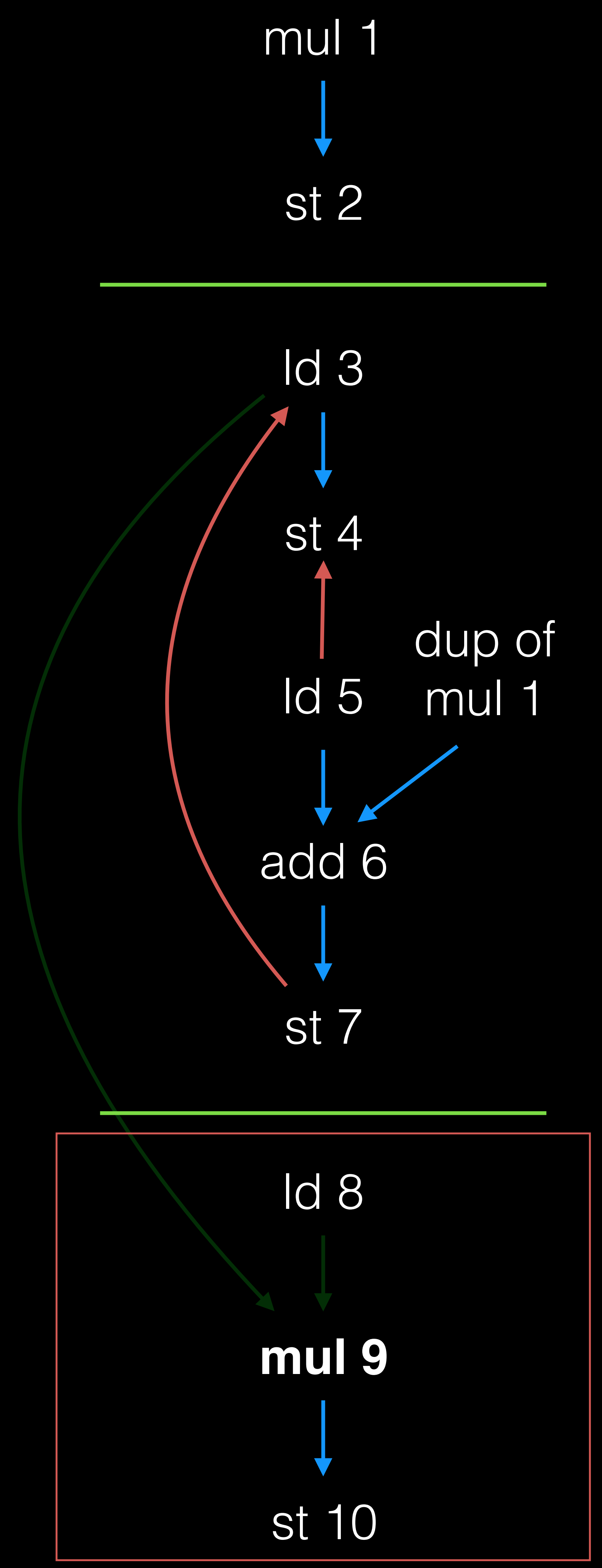
Algorithm



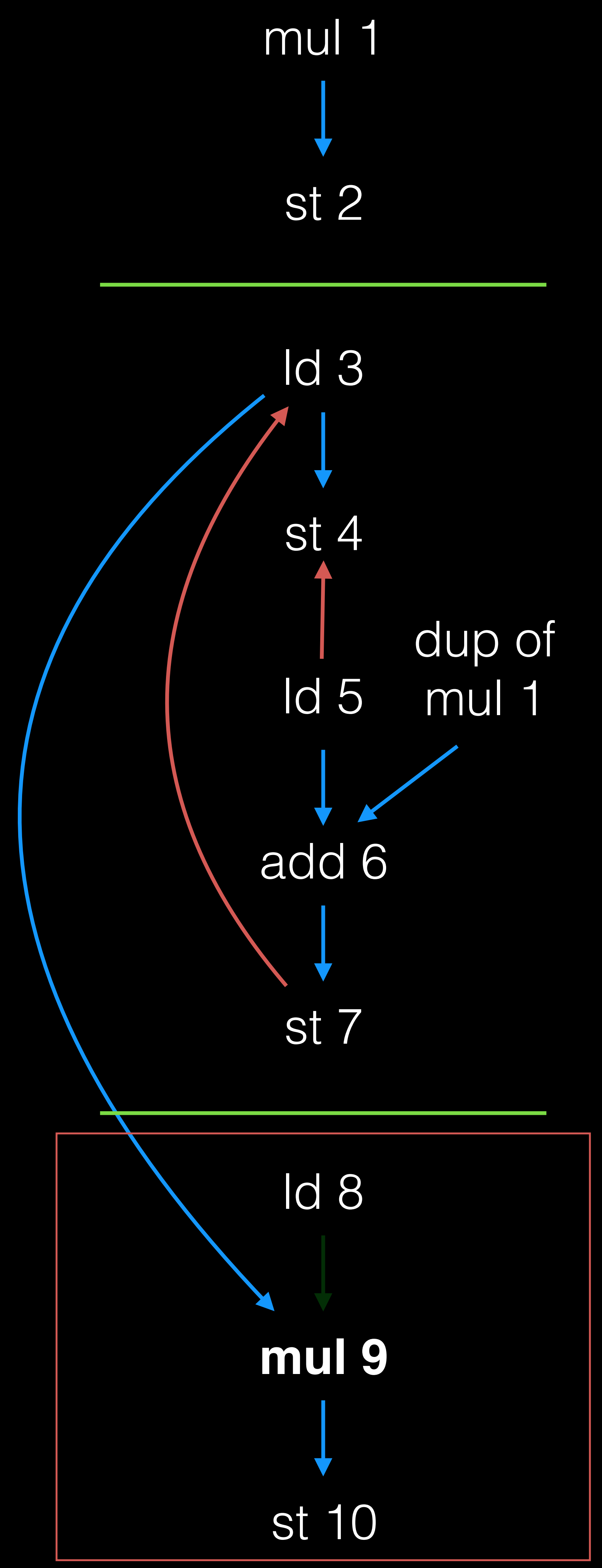
Algorithm



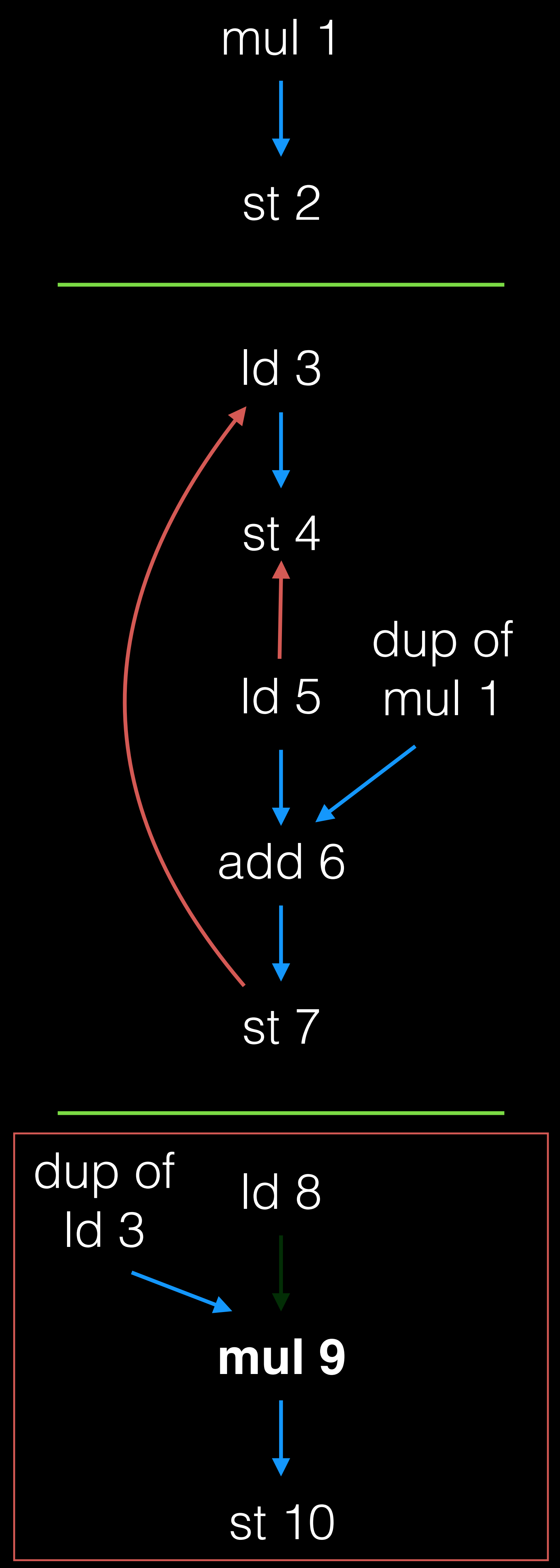
Algorithm



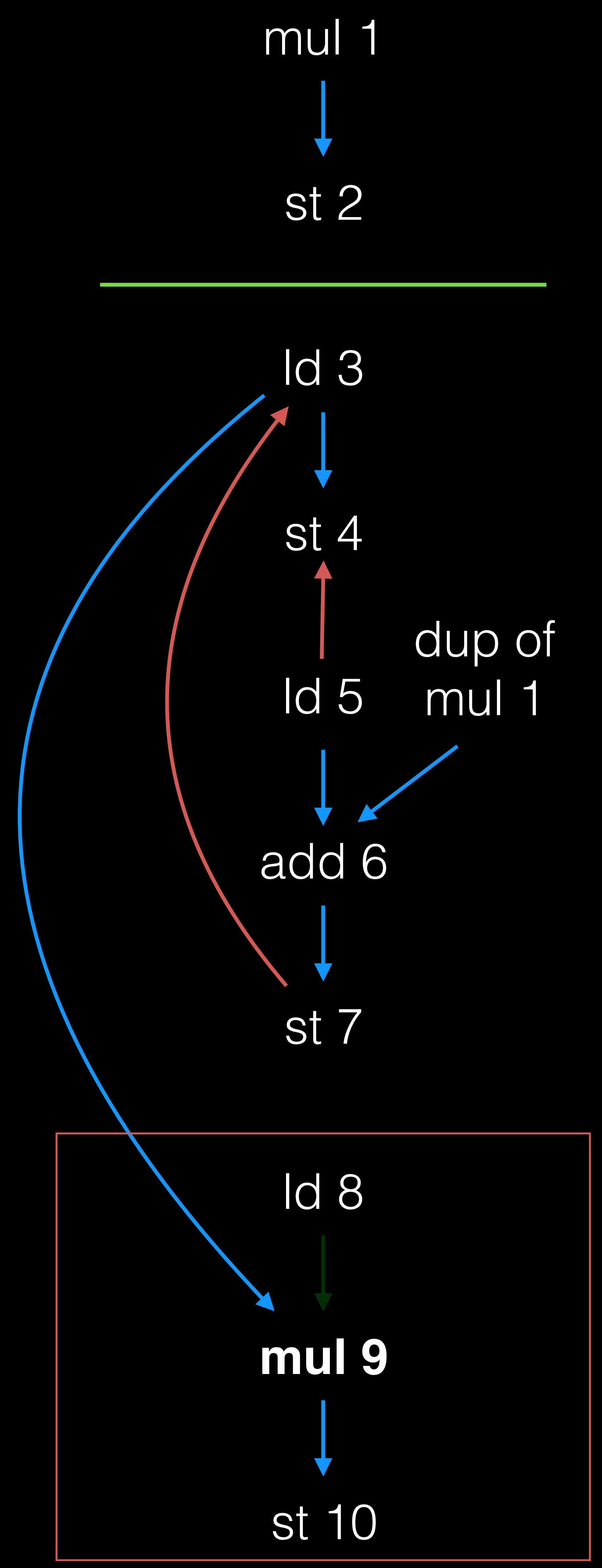
Algorithm



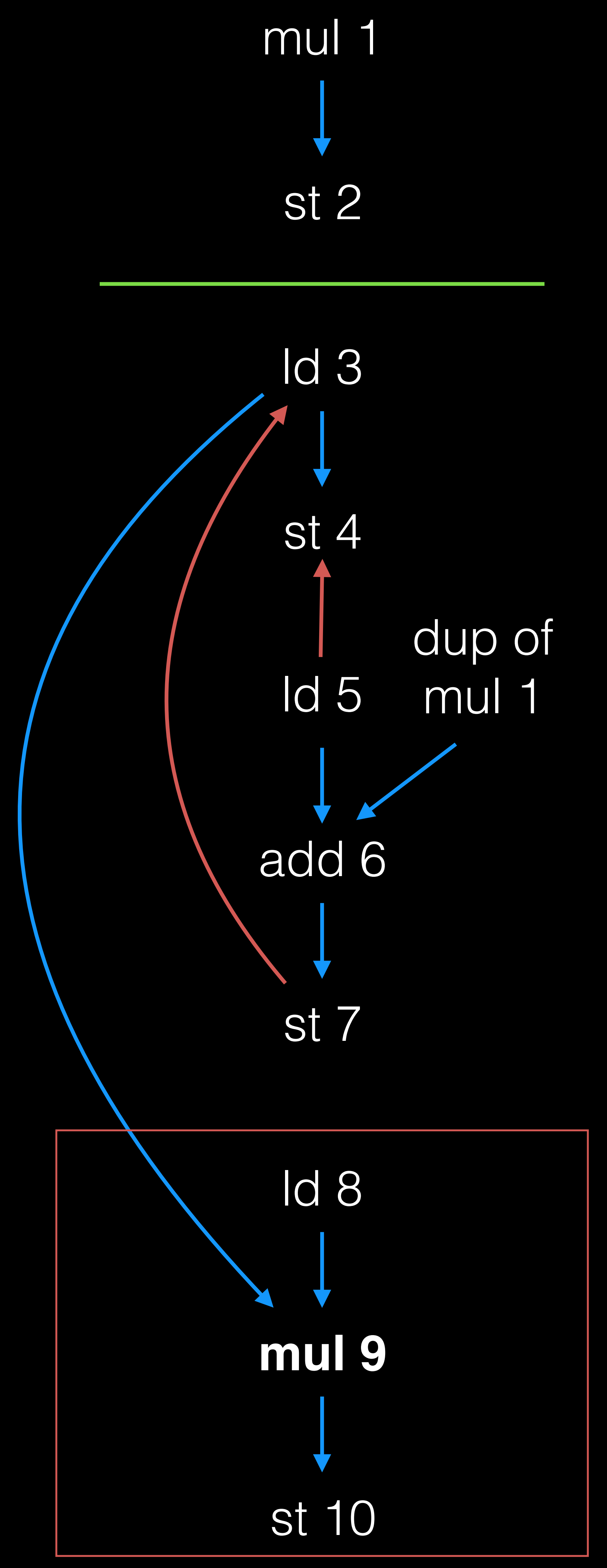
Algorithm


mul 1
st 2
ld 3
st 4
ld 5
dup of
mul 1
add 6
st 7
dup of
ld 3
ld 8
mul 9
st 10

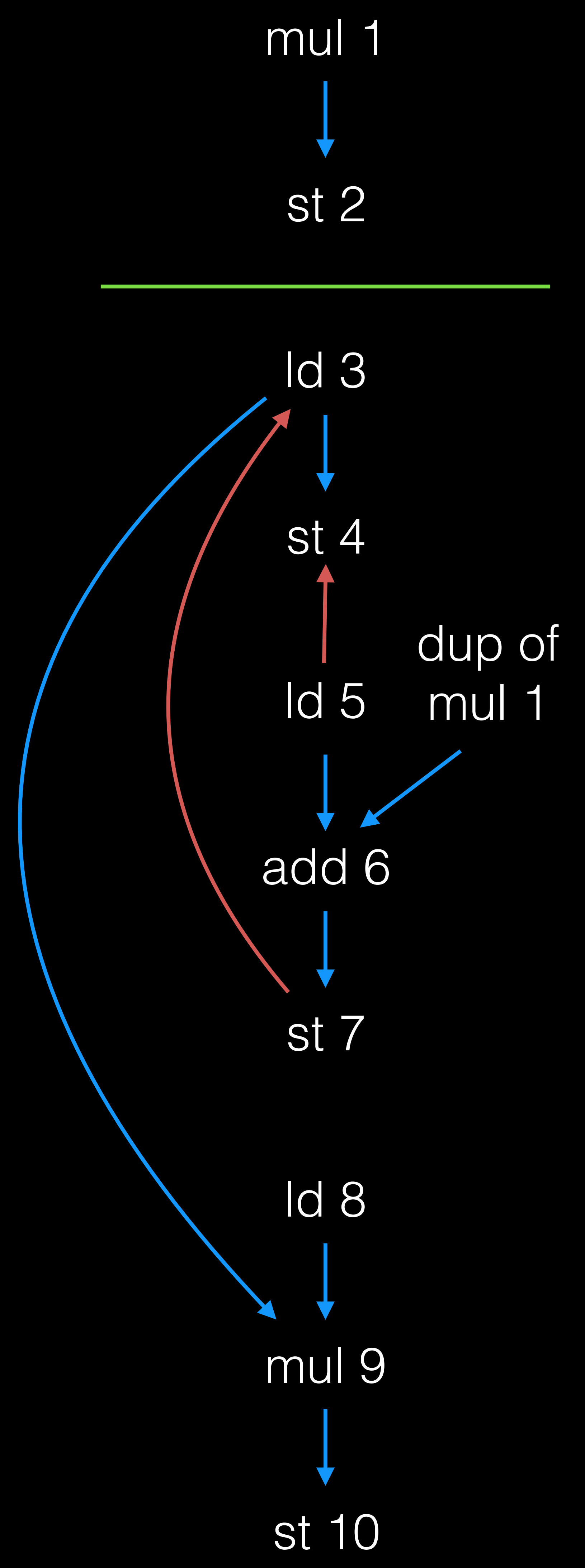
Algorithm



Algorithm



Algorithm



Recap

- Distributed loop
  - Versioned with run-time alias checks
- Top loop vectorized

Case Study

Vectorized

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k-1] + tpmm[k-1];
  if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = dp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;
}
```

```
for (k = 1; k <= M; k++) {
  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
sk}
}
```

Case Study

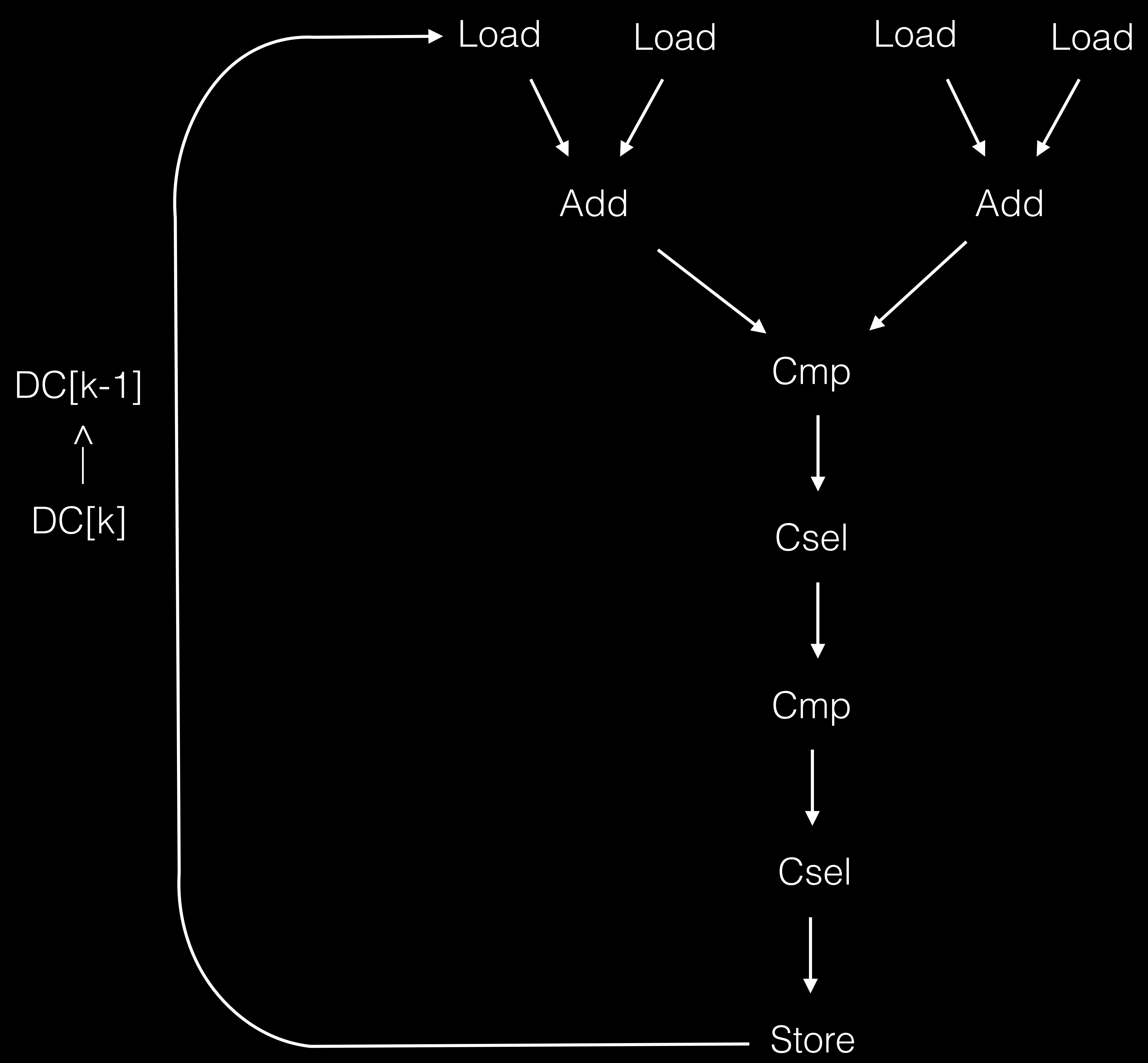
```
for (k = 1; k <= M; k++) {
  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

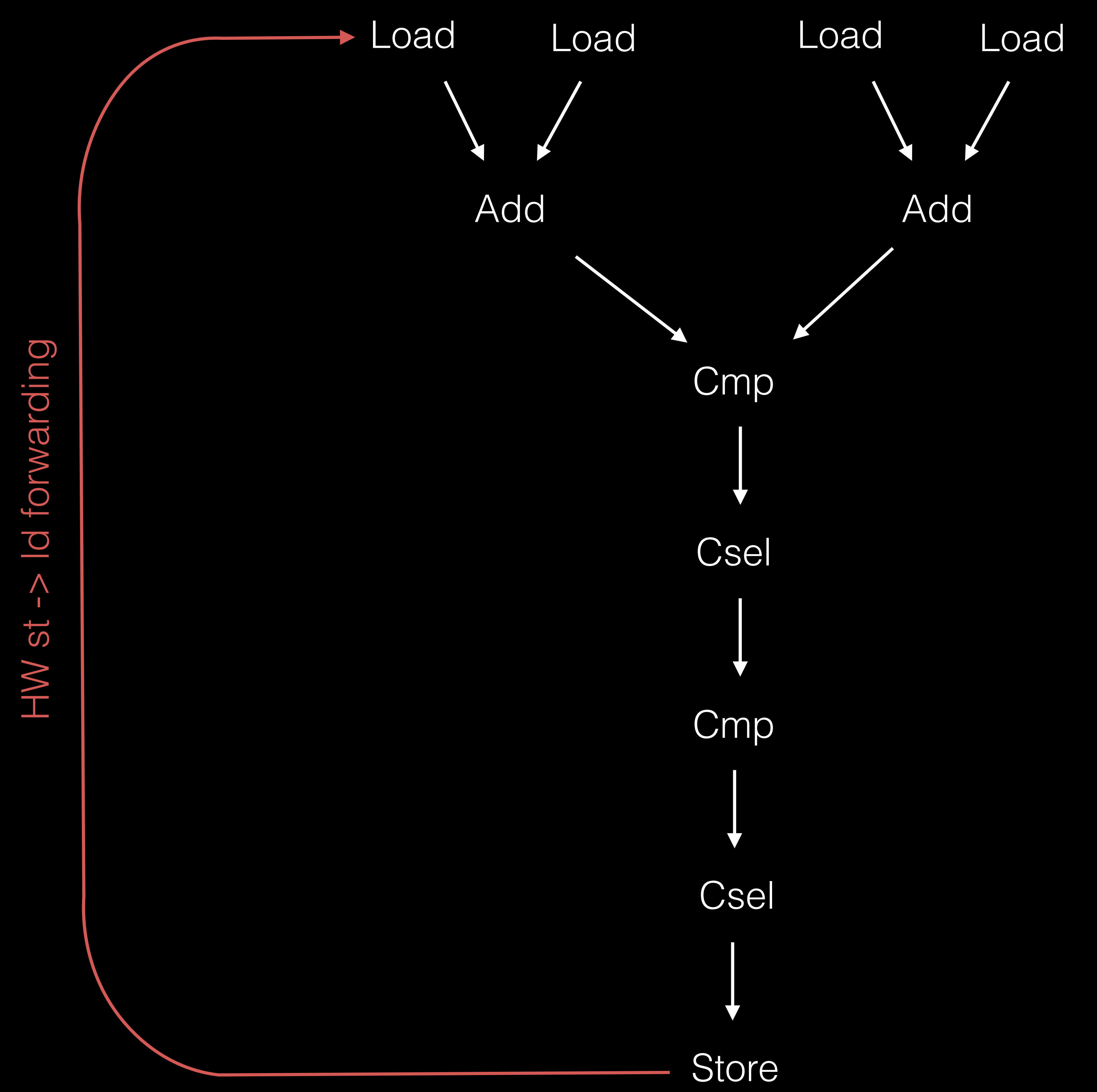
Case Study

```
dc[k] = dc[k-1] + tpdd[k-1];
if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
if (dc[k] < -INFTY) dc[k] = -INFTY;
```

Case Study

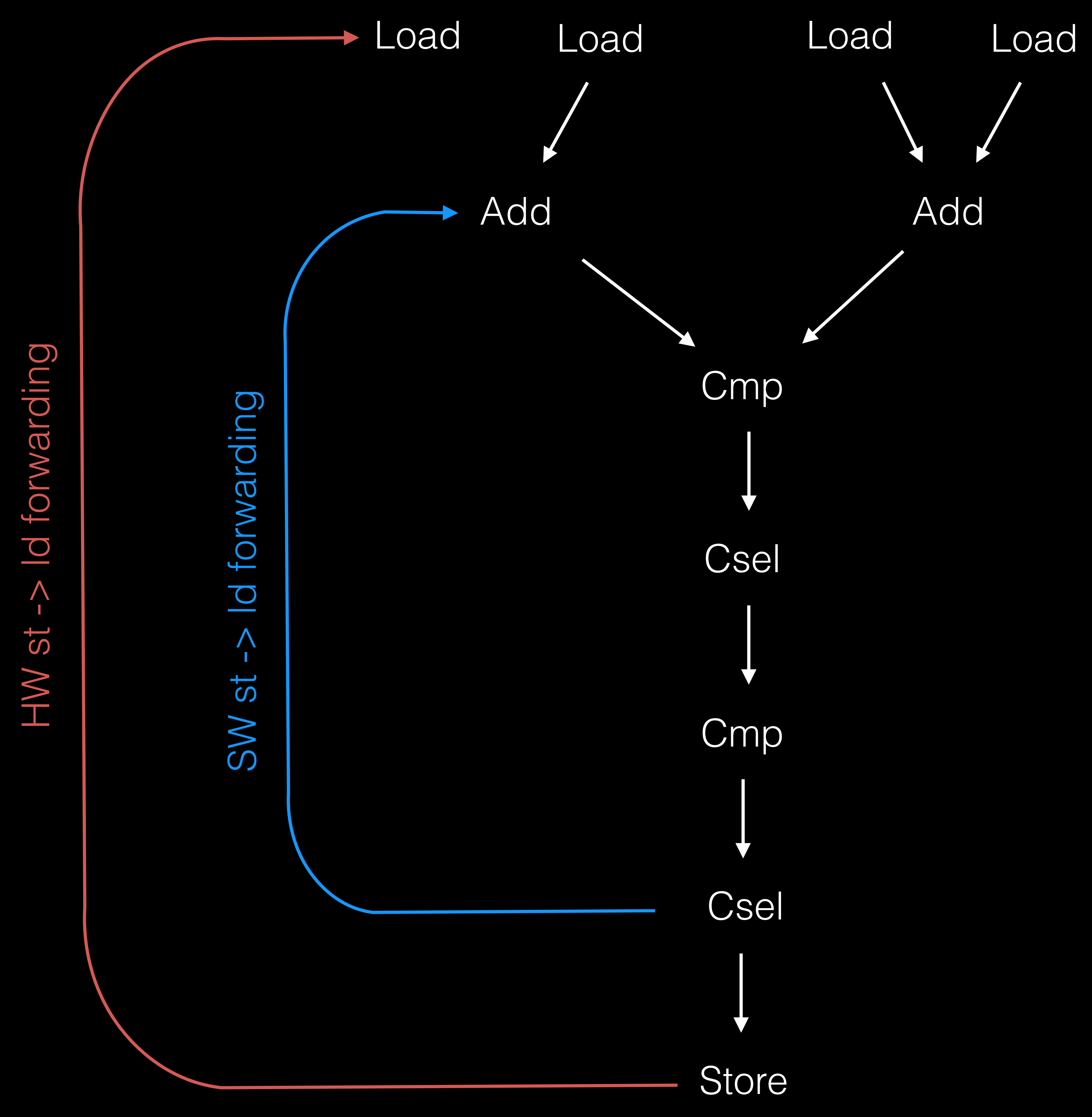


Case Study



Load
Load
Load
Load
Add
Add
Cmp
Csel
Cmp
Csel
Store
HW st -> ld forwarding

Case Study



Load
Load
Load
Load
Add
Add
Cmp
Csel
Cmp
Csel
Store
HW st -> ld forwarding
SW st -> ld forwarding

Case Study

```
dc[k] = dc[k-1] + tpdd[k-1];
if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
if (dc[k] < -INFTY) dc[k] = -INFTY;
```

Loop Load Elimination

Algorithm

1. Find loop-carried dependences with iteration distance of one

2. Between store -> load?

3. No (may-)intervening store

4. Propagate value stored to uses of load

Algorithm

```
for (k = 1; k <= M; k++) {
  dc[k] = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

Algorithm

```
for (k = 1; k <= M; k++) {
  dc[k] = T = dc[k-1] + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = T = sc;
  if (dc[k] < -INFTY) dc[k] = T = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

Algorithm

```
for (k = 1; k <= M; k++) {
  dc[k] = T =     T     + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = T = sc;
  if (dc[k] < -INFTY) dc[k] = T = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

Algorithm

```
T = dc[0];
for (k = 1; k <= M; k++) {
  dc[k] = T =    T    + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = T = sc;
  if (dc[k] < -INFTY) dc[k] = T = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
  }
}
```

Algorithm

```
T = dc[0];
for (k = 1; k <= M; k++) {
  dc[k] = T =    T    + tpdd[k-1];
  if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = T = sc;
  if (dc[k] < -INFTY) dc[k] = T = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
sk}
}
```

Loop Load Elimination

- Simple and cheap using Loop Access Analysis
  - With Loop Versioning can optimize more loops
- GVN Load-PRE can be simplified to not worry about loop cases

Recap

- Distributed loop into two loops
  - Versioned with run-time alias checks
- Vectorized top loop
- Store-to-load forwarding in bottom loop
  - Versioned with run-time alias checks

Results

- 20-30% gain on 456.hmmer on ARM64 and x86
- Loop Access Analysis pass
- Loop Versioning utility
- Loop Distribution pass
- Loop Load Elimination pass

Future Work

- Commit Loop Load Elimination
- Tune Loop Distribution and turn it on by default
- Loop Distribution with Program Dependence Graph

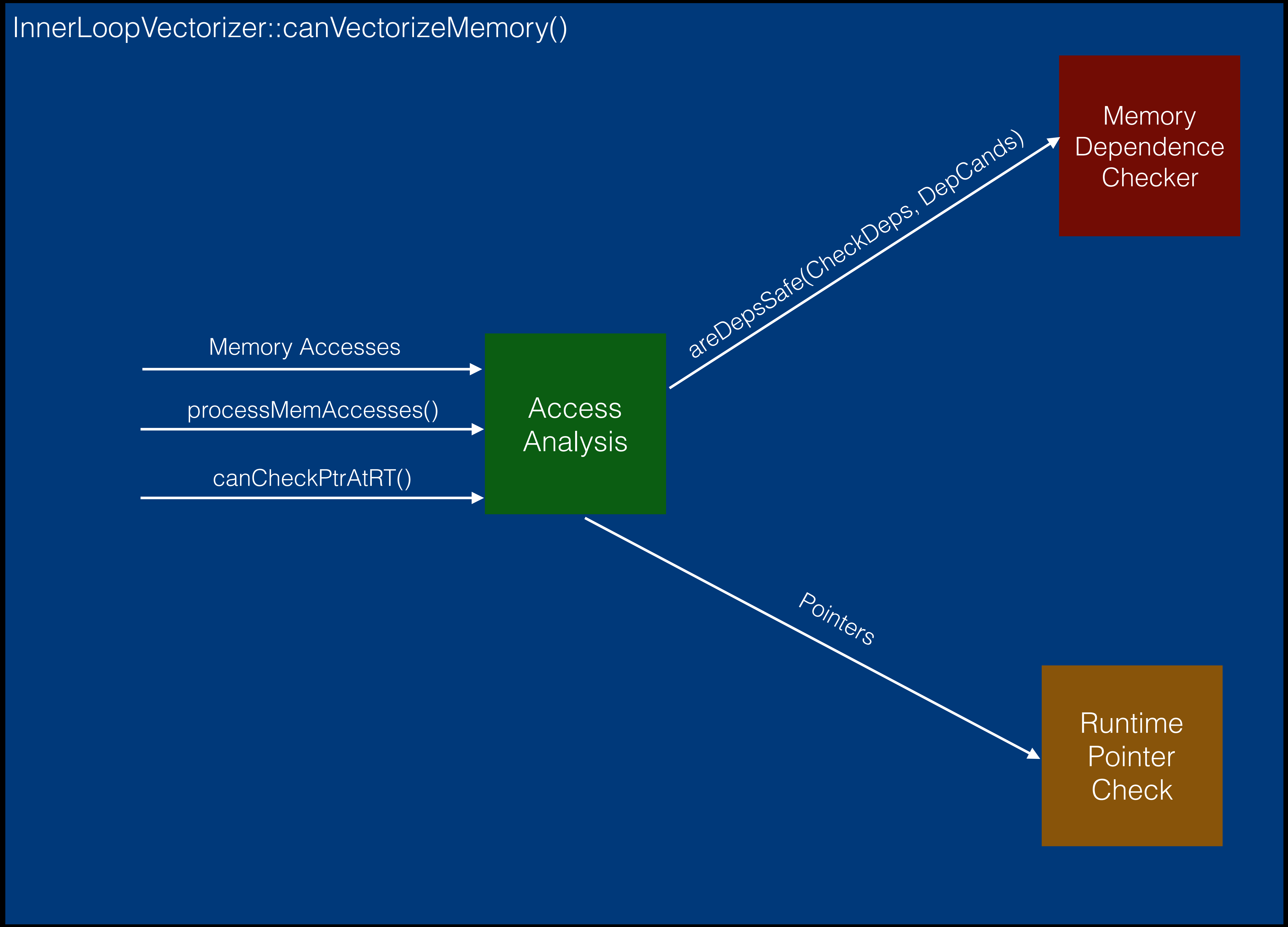
Acknowledgements

- Chandler Carruth
- Hal Finkel
- Arnold Schwaighofer
- Daniel Berlin

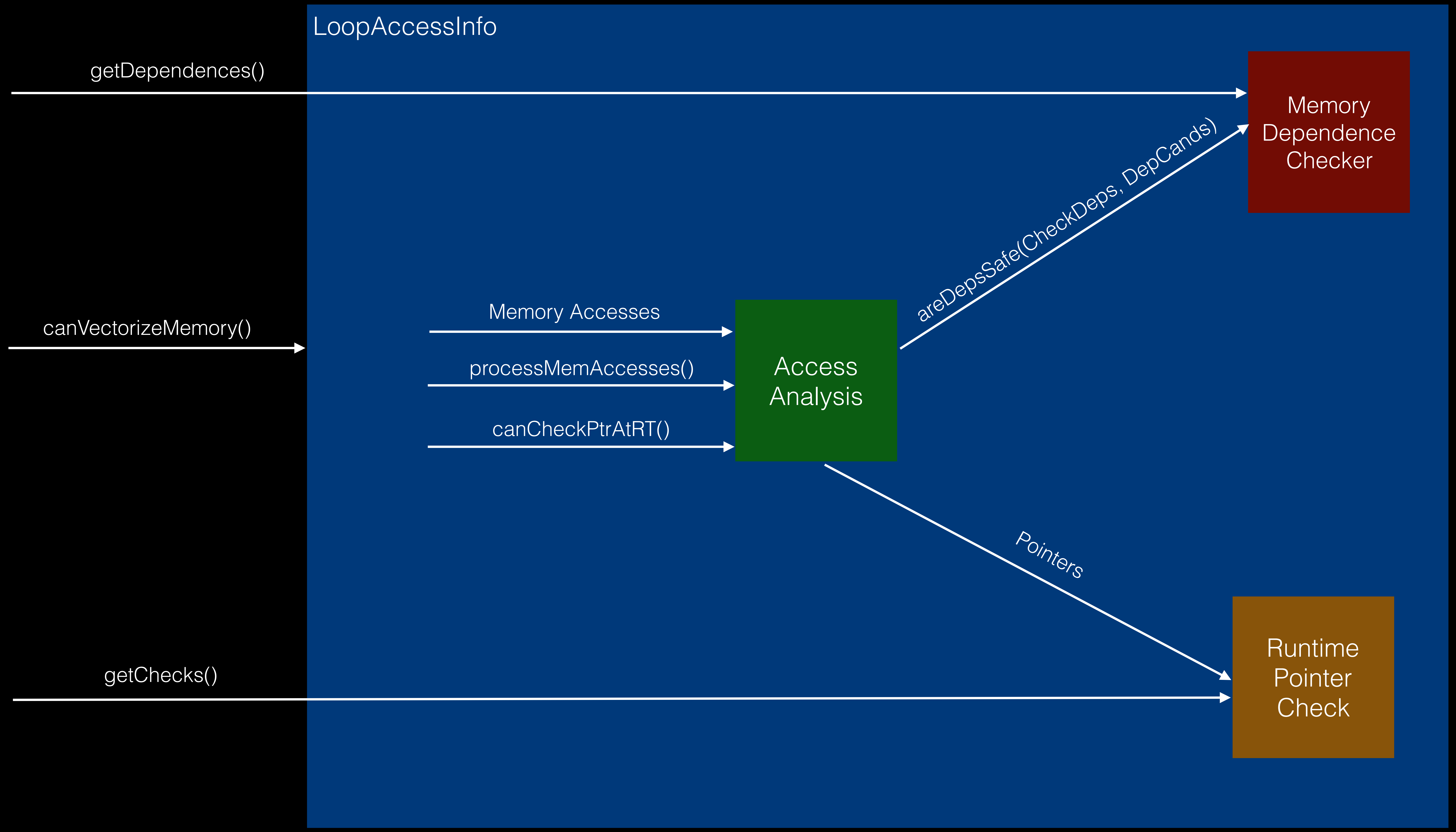
Q&A

Back-up

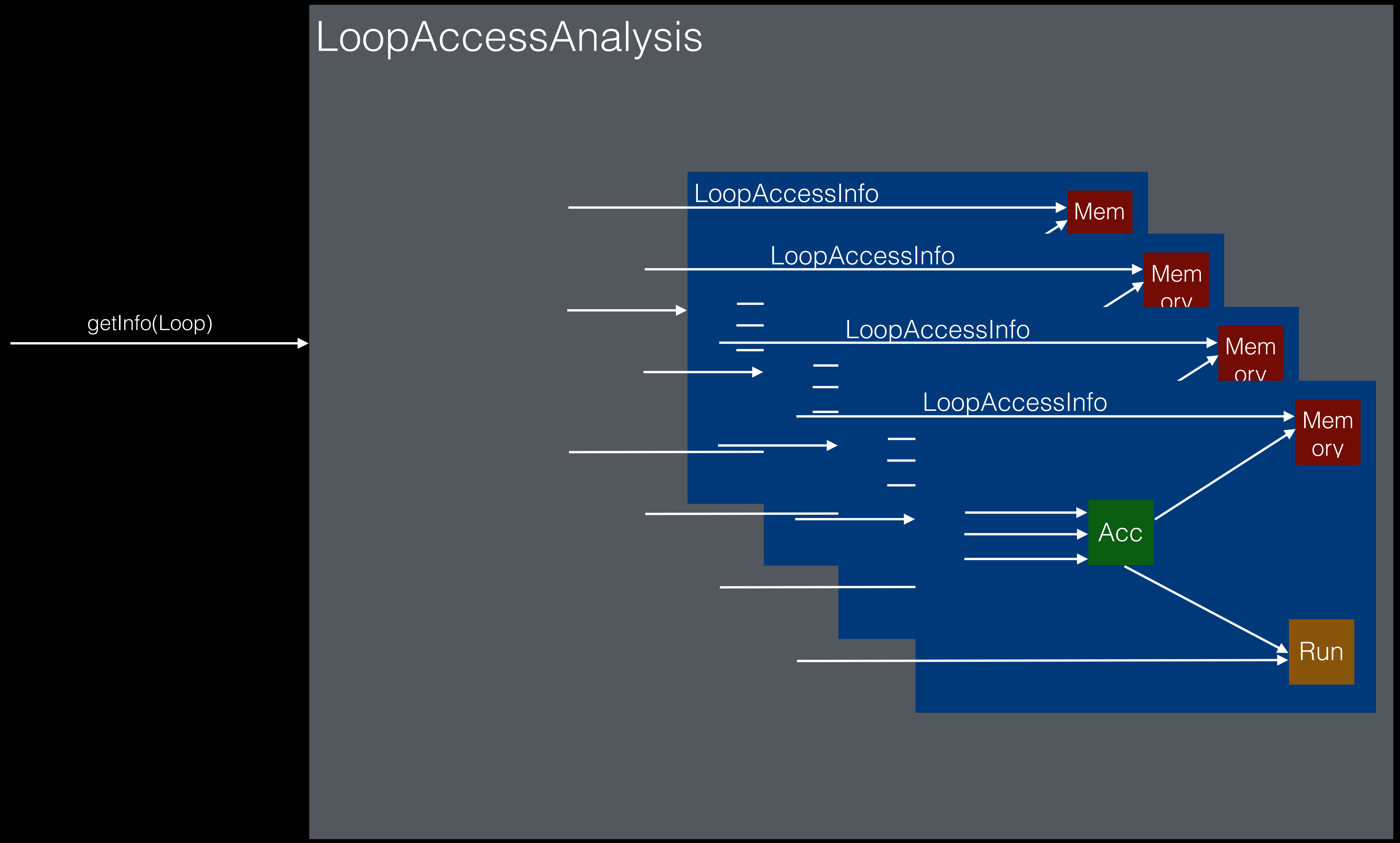
Loop Vectorizer



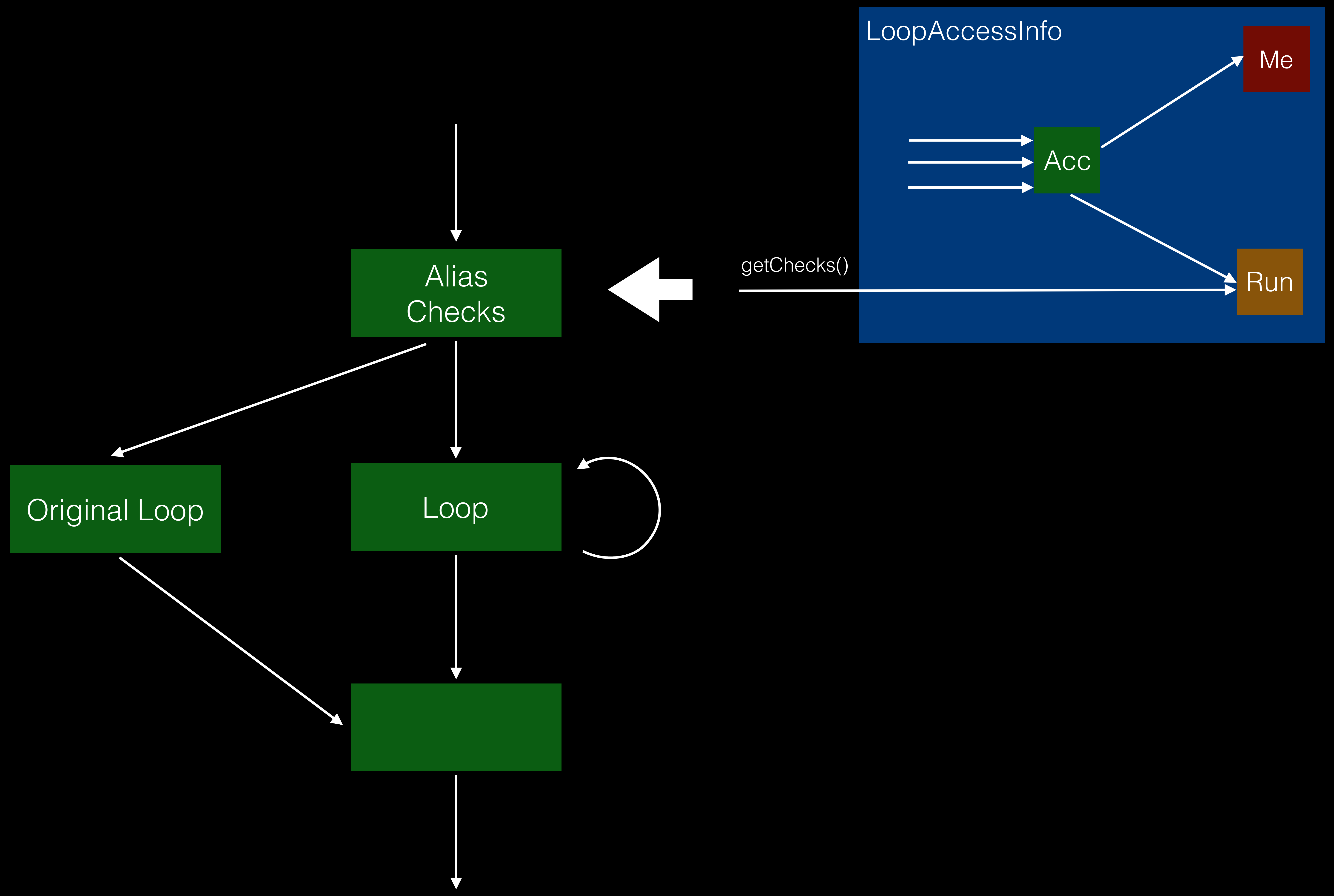
Loop Access Analysis



Loop Access Analysis



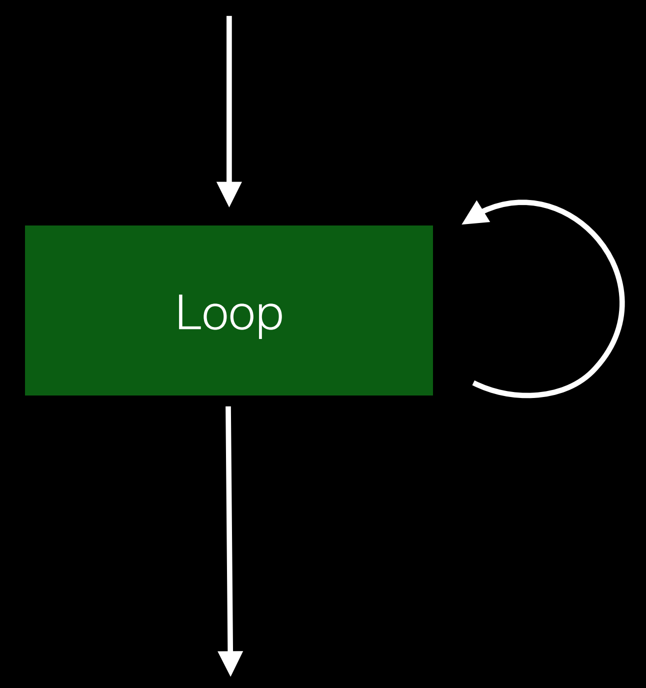
Loop Versioning



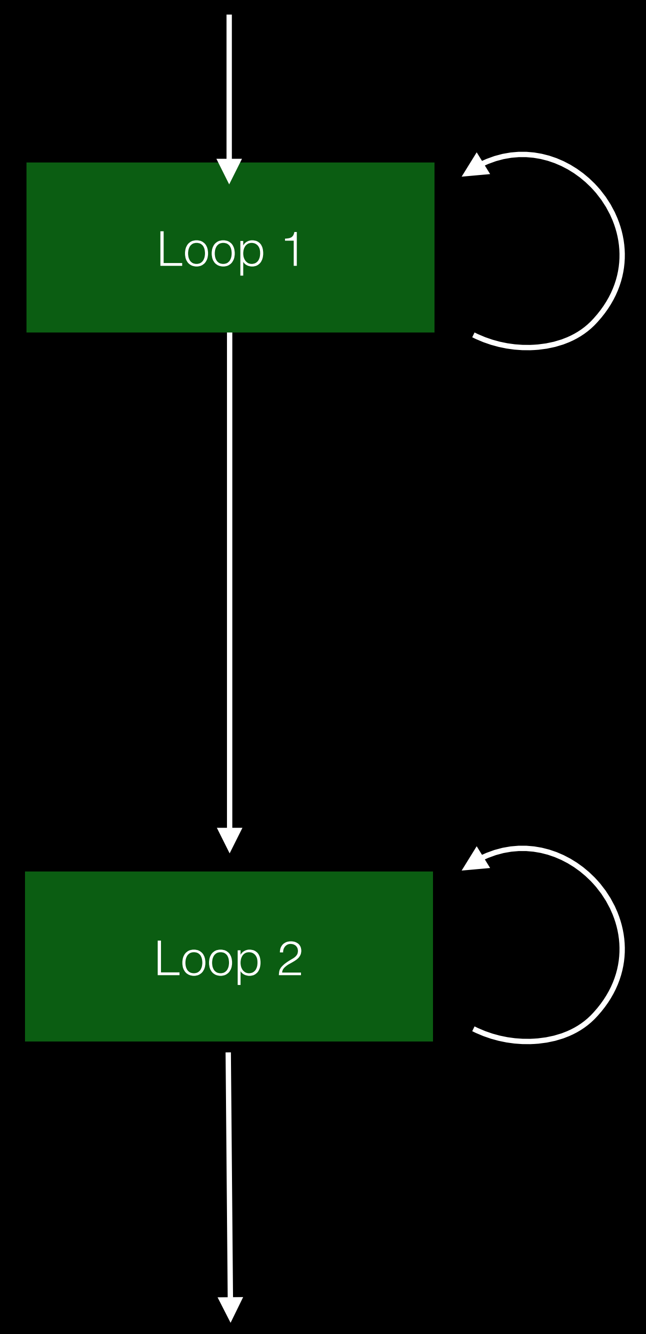
Loop Versioning

- Users:
  - Loop Distribution
  - Loop Load Elimination
  - WIP LICM-based Loop Versioning
- Future work:
  - Run-time trip count check
  - Merge versions into a slow path and a fast path

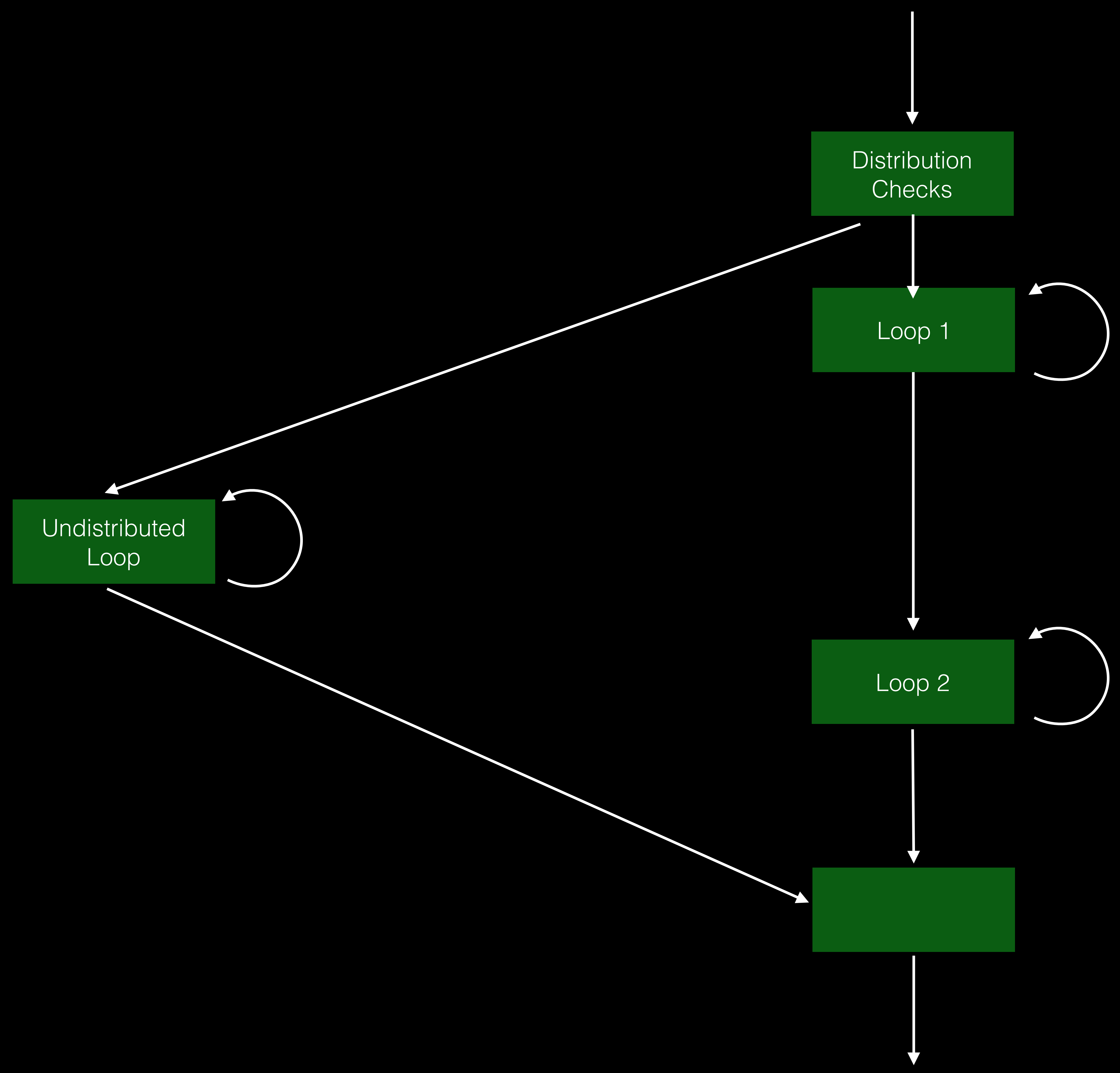
Loop Versioning



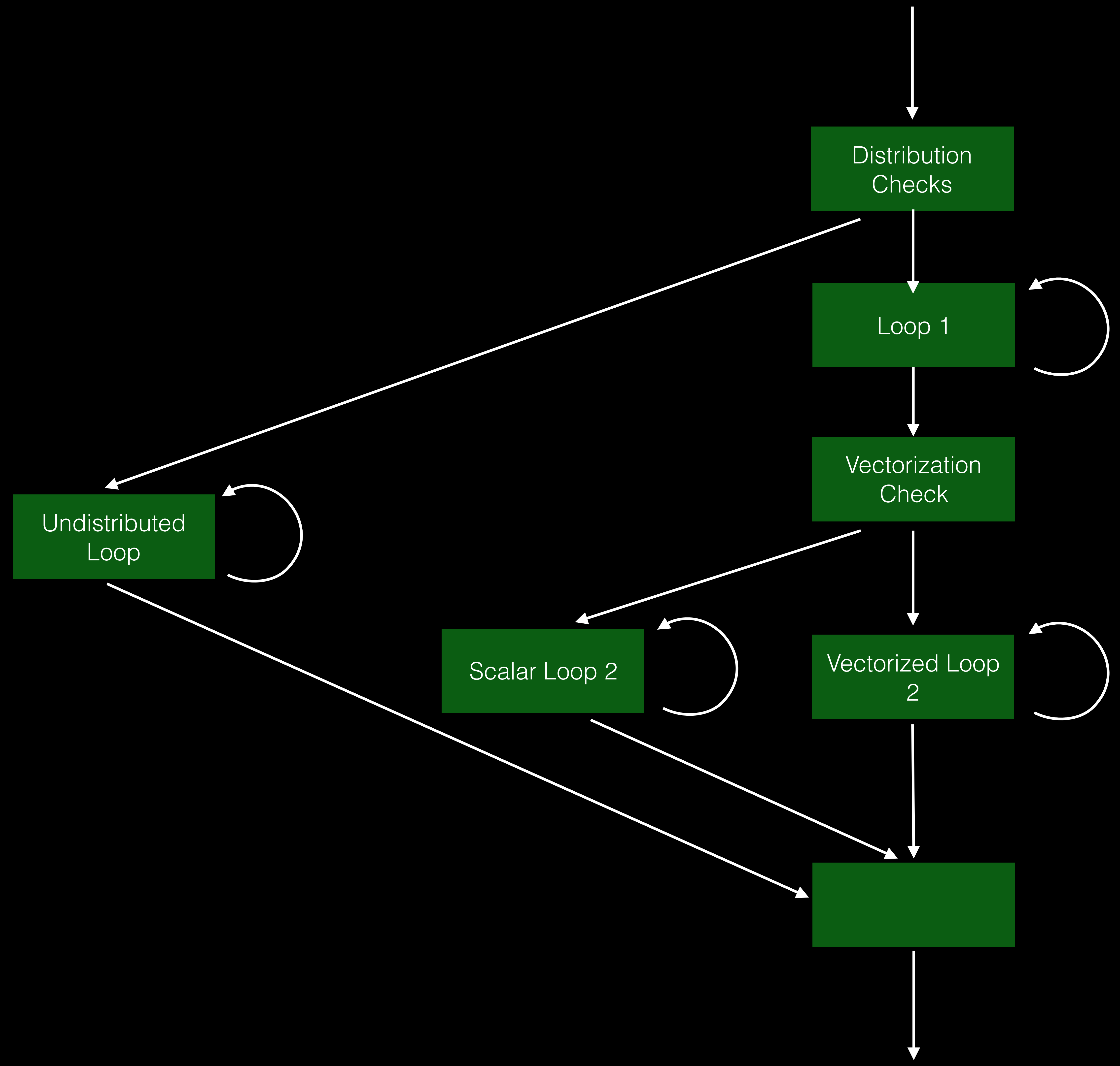
Loop Versioning



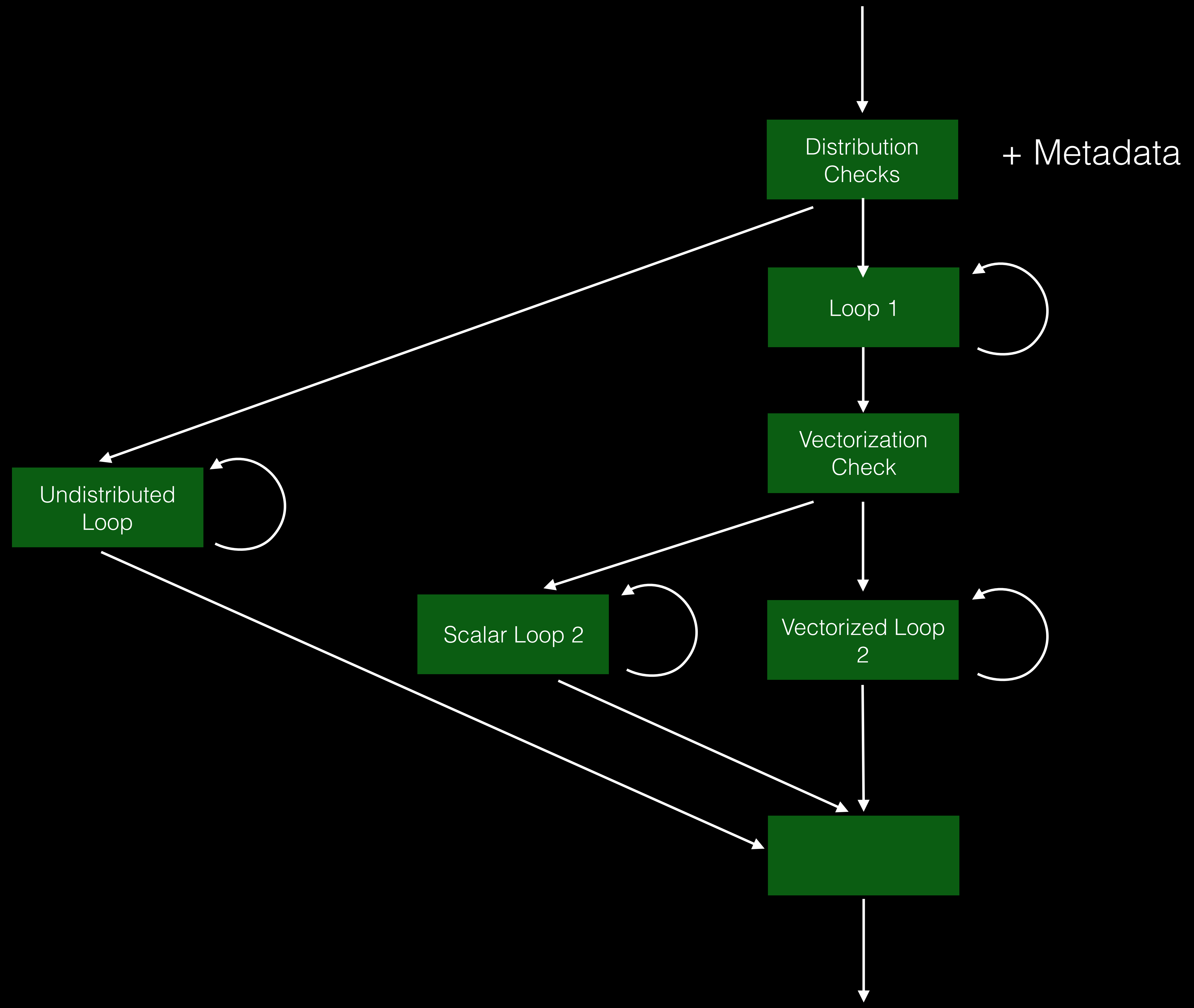
Loop Versioning



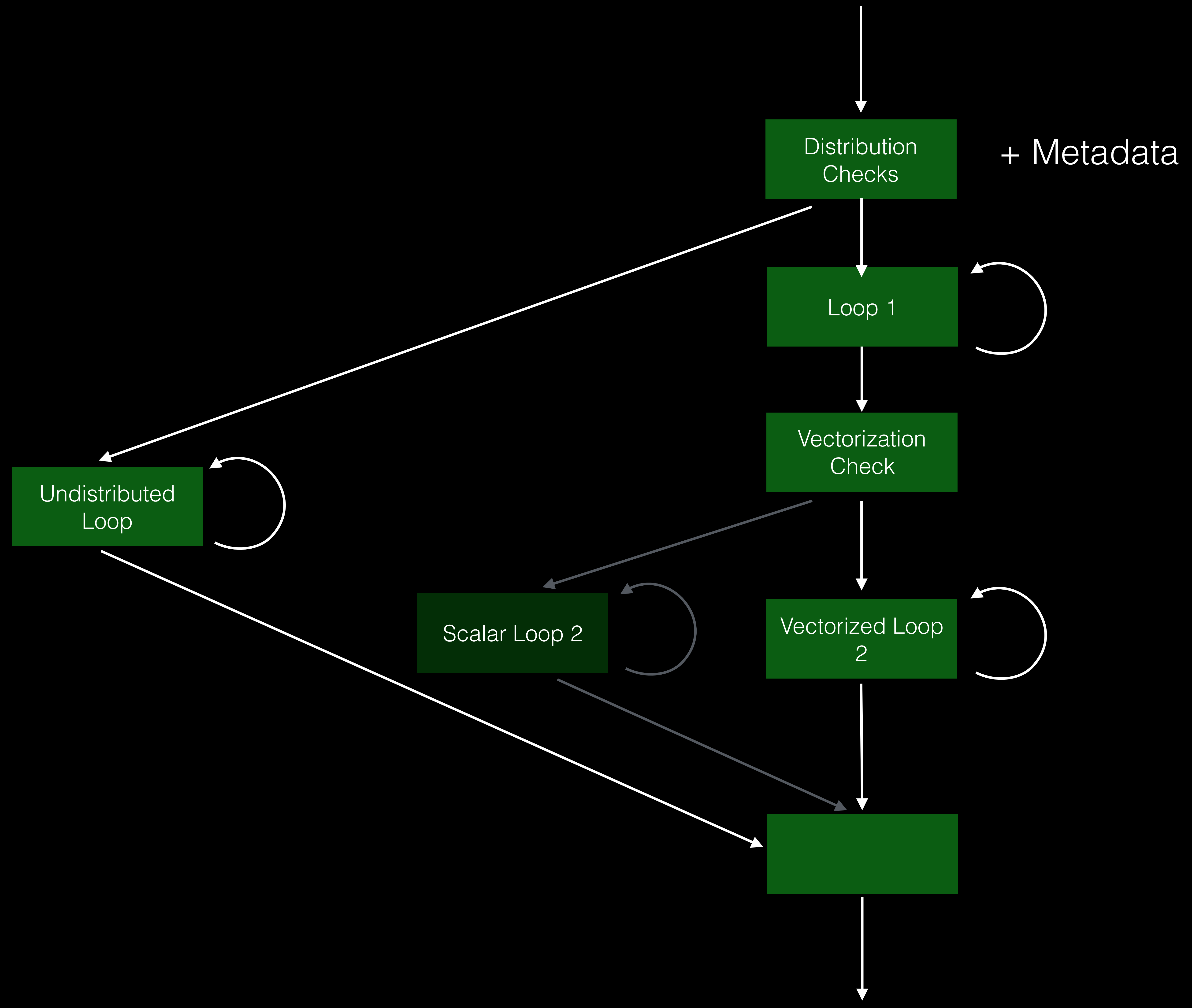
Loop Versioning



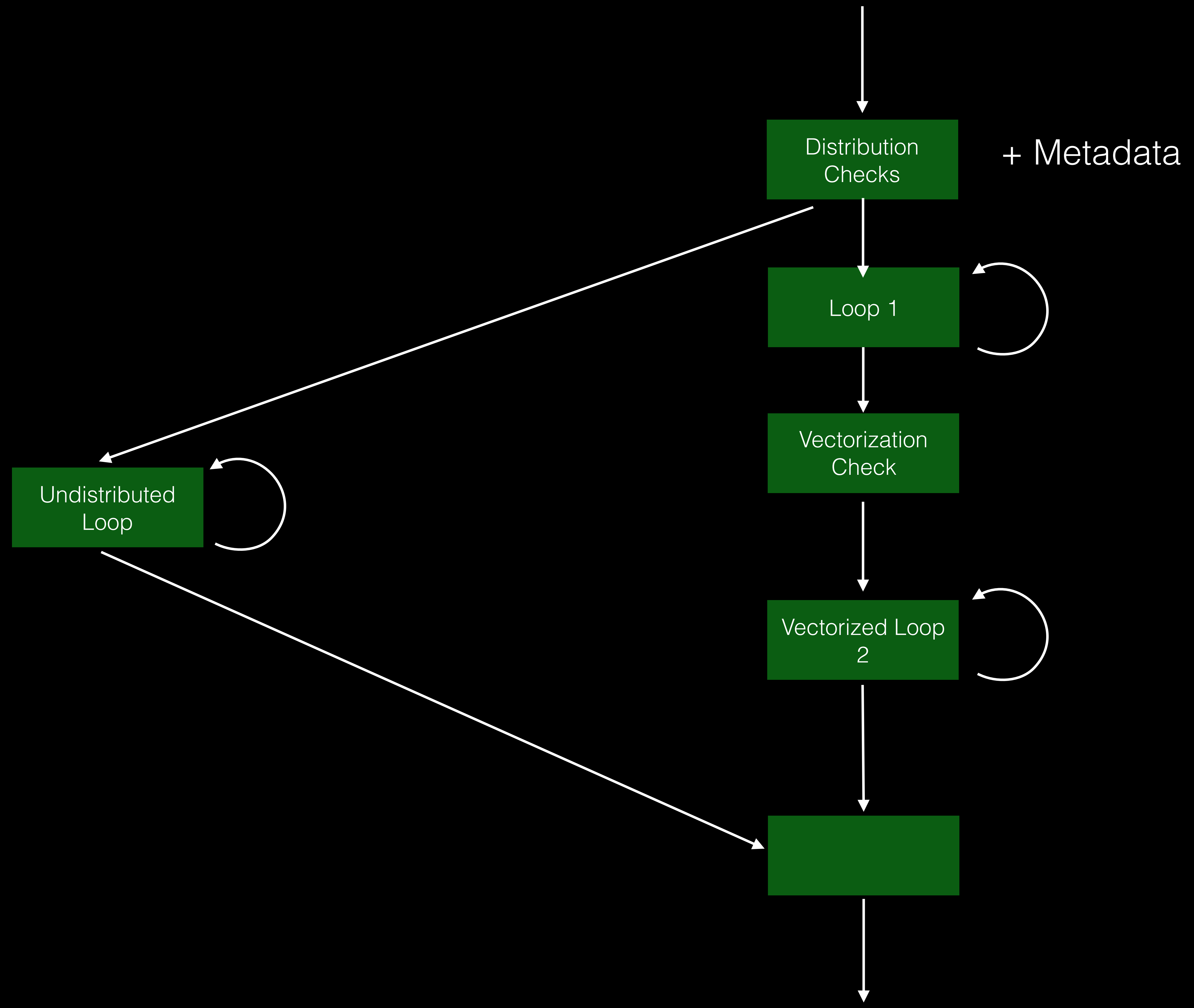
Loop Versioning



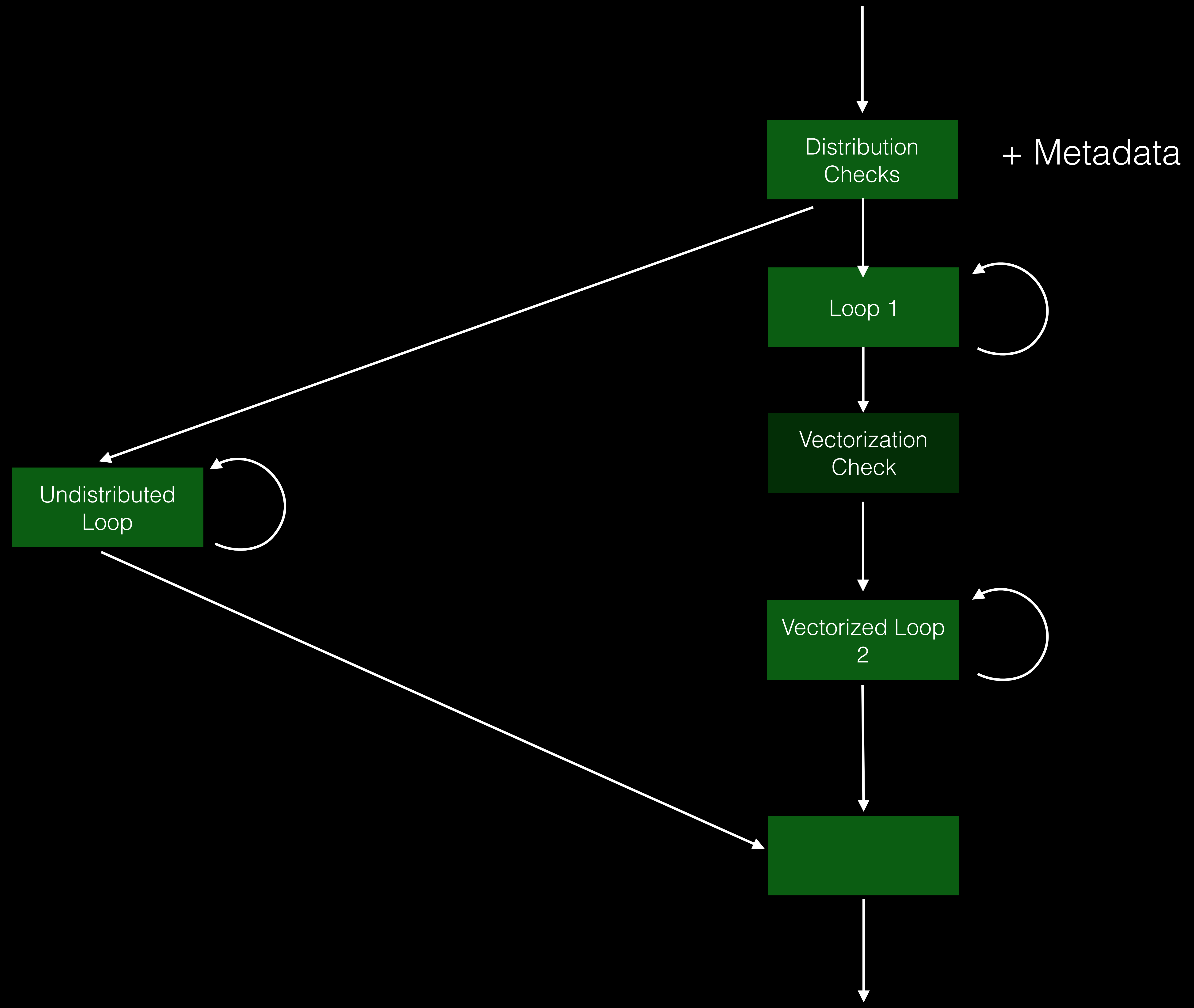
Loop Versioning



Loop Versioning



Loop Versioning



Loop Versioning