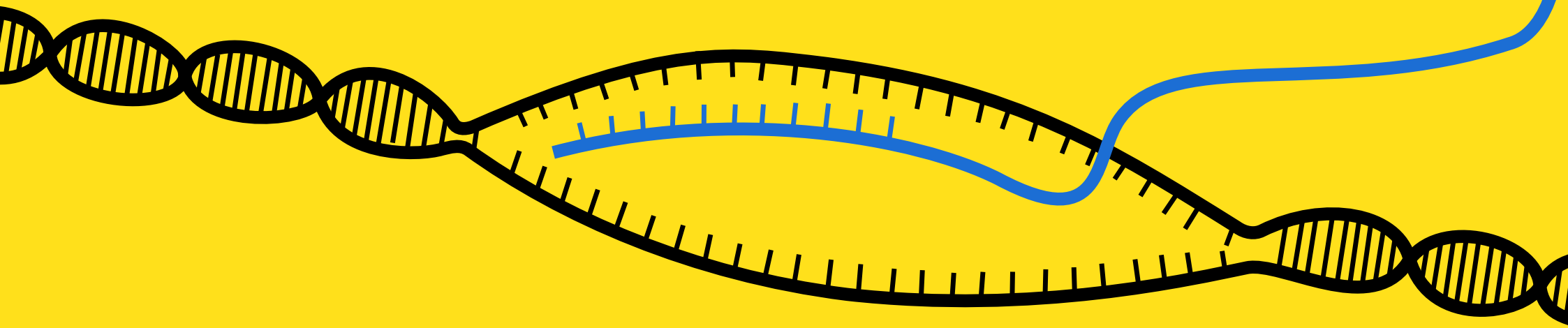
Rodrigo Rocha University of Edinburgh http://rcor.me

FUNCTION MERGING

by

SEQUENCE ALIGNMENT

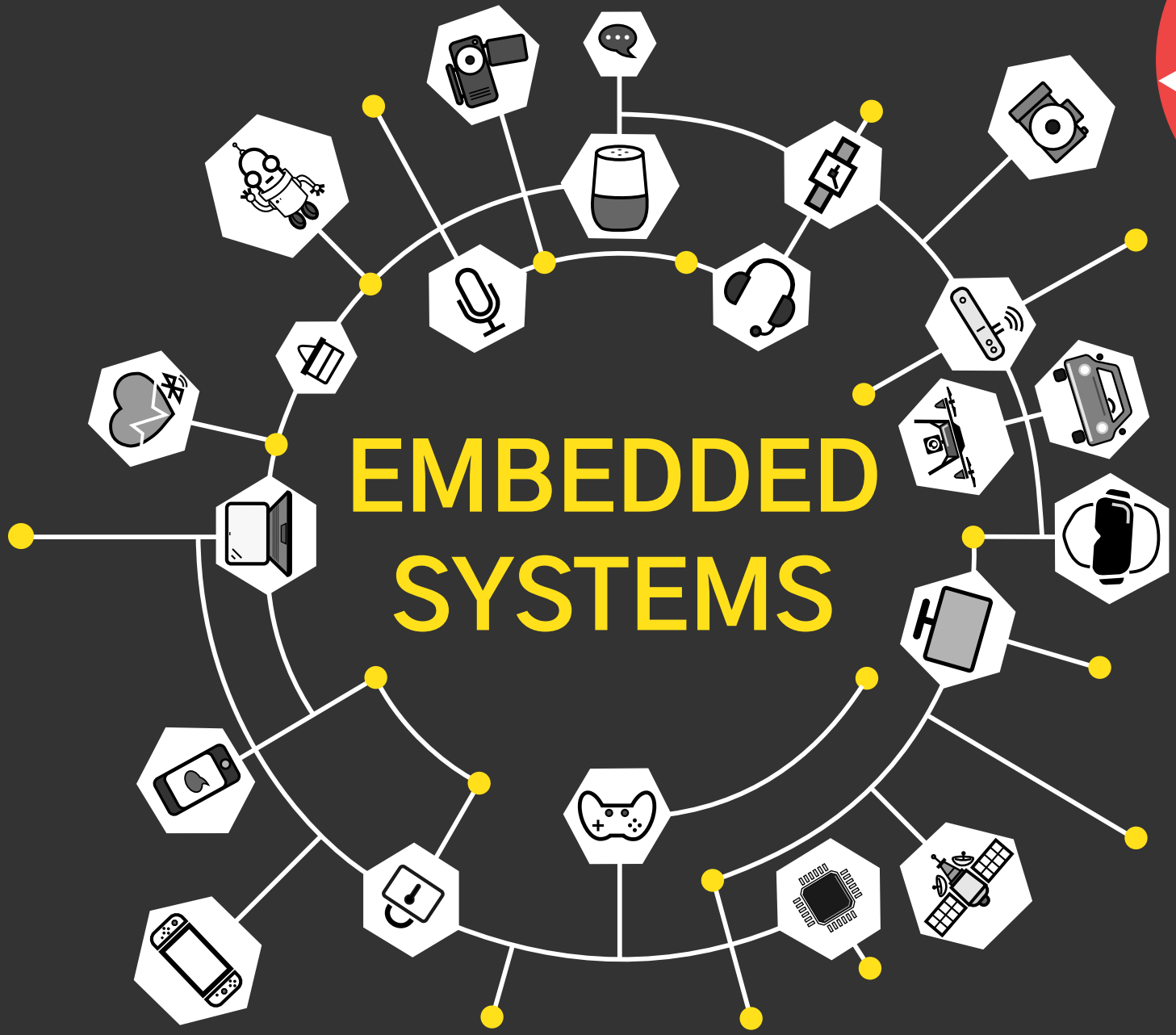


DOES SIZE MATTER?

DOES SIZE MATTER?

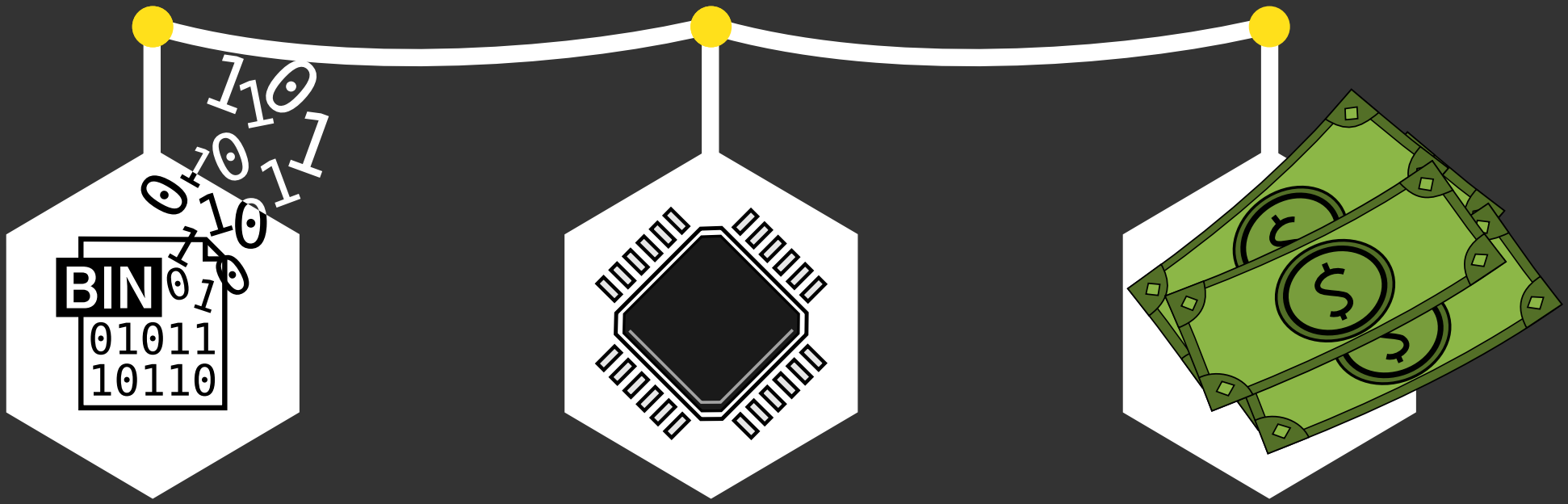
DOES SIZE MATTER?

YES



SAVINGS IN CODE

Smaller Memories



COST SAVINGS

HOW TO REDUCE SIZE?

HOW TO REDUCE SIZE?

MERGE

Similar or Identical

FUNCTIONS

MERGING FUNCTIONS

```c
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```c
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

MERGING FUNCTIONS

Identical Code

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

MERGING FUNCTIONS

Identical Code

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
merged(int a, int b
```

```
int var = a*b +
int result = foo(var);
printf("result: %d\n", result);
```

MERGING FUNCTIONS

Selecting Different Operands

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
merged(int a, int b
```

```
int var = a*b +
int result = foo(var);
printf("result: %d\n", result);
```

MERGING FUNCTIONS

Selecting Different Operands

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
merged(int a, int b, int func_id) {

int var = a*b +
int result = foo(var);
printf("result: %d\n", result);
```

MERGING FUNCTIONS

Selecting Different Operands

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
merged(int a, int b, int func_id) {
int term = (func_id==1)?
int var = a*b +
int result = foo(var);
printf("result: %d\n", result);
```

MERGING FUNCTIONS

Selecting Different Operands

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
merged(int a, int b, int func_id) {
int term = (func_id==1)? 10 :
int var = a*b +
int result = foo(var);
printf("result: %d\n", result);
```

MERGING FUNCTIONS

Selecting Different Operands

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
merged(int a, int b, int func_id) {
int term = (func_id==1)? 10 : b ;
int var = a*b +
int result = foo(var);
printf("result: %d\n", result);
```

MERGING FUNCTIONS

Selecting Different Operands

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
merged(int a, int b, int func_id) {
int term = (func_id==1)? 10 : b ;
int var = a*b + term;
int result = foo(var);
printf("result: %d\n", result);
```

MERGING FUNCTIONS

Branching to Distinct CFG

```c
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```c
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```c
merged(int a, int b, int func_id) {
int term = (func_id==1)? 10 : b ;
int var = a*b + term;
int result = foo(var);
printf("result: %d\n", result);
```

MERGING FUNCTIONS

Branching to Distinct CFG

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
merged(int a, int b, int func_id) {
int term = (func_id==1)? 10 : b ;
int var = a*b + term;
int result = foo(var);
printf("result: %d\n", result);

  if (result==0)
    printf("result is zero\n");
```

MERGING FUNCTIONS

Branching to Distinct CFG

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
merged(int a, int b, int func_id) {
int term = (func_id==1)? 10 : b ;
int var = a*b + term;
int result = foo(var);
printf("result: %d\n", result);
if (func_id==2)
  if (result==0)
    printf("result is zero\n");
```

MERGING FUNCTIONS

Combining the Return Value

```c
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```c
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
     printf("result is zero\n");
}
```

```c
merged(int a, int b, int func_id) {
int term = (func_id==1)? 10 : b ;
int var = a*b + term;
int result = foo(var);
printf("result: %d\n", result);
if (func_id==2)
  if (result==0)
     printf("result is zero\n");
```

MERGING FUNCTIONS

Combining the Return Value

```c
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```c
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```c
merged(int a, int b, int func_id) {
int term = (func_id==1)? 10 : b ;
int var = a*b + term;
int result = foo(var);
printf("result: %d\n", result);
if (func_id==2)
  if (result==0)
    printf("result is zero\n");
```

MERGING FUNCTIONS

Combining the Return Value

```c
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```c
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```c
int merged(int a, int b, int func_id) {
  int term = (func_id==1)? 10 : b ;
  int var = a*b + term;
  int result = foo(var);
  printf("result: %d\n", result);
  if (func_id==2)
    if (result==0)
      printf("result is zero\n");
```

MERGING FUNCTIONS

Combining the Return Value

```c
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```c
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```c
int merged(int a, int b, int func_id) {
  int term = (func_id==1)? 10 : b ;
  int var = a*b + term;
  int result = foo(var);
  printf("result: %d\n", result);
  if (func_id==2)
    if (result==0)
      printf("result is zero\n");
  return result;
}
```

MERGING FUNCTIONS

Reduces Code Size!

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
int merged(int a, int b, int func_id) {
  int term = (func_id==1)? 10 : b ;
  int var = a*b + term;
  int result = foo(var);
  printf("result: %d\n", result);
  if (func_id==2)
    if (result==0)
      printf("result is zero\n");
  return result;
}
```

MERGING FUNCTIONS

However...

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

```
int merged(int a, int b, int func_id) {
  int term = (func_id==1)? 10 : b ;
  int var = a*b + term;
  int result = foo(var);
  if (result==0)
    printf("result is zero\n");
  return result;
}
```

State of the Art FAILS

WHAT WE HAVE

Production compilers:

LLVM, GCC, MSVC

WHAT WE HAVE

Production compilers:

LLVM, GCC, MSVC

Only **Identical** Functions

WHAT WE HAVE

The state of the art

WHAT WE HAVE

The state of the art

significant improvement but...

WHAT WE HAVE

The state of the art
significant improvement but...

MUST HAVE IDENTICAL

WHAT WE HAVE

The state of the art
significant improvement but...

MUST HAVE IDENTICAL

- CFGs

WHAT WE HAVE

The state of the art
significant improvement but...

MUST HAVE IDENTICAL

- CFGs
- Return types

WHAT WE HAVE

The state of the art

significant improvement but...

MUST HAVE IDENTICAL

- CFGs
- Return types
- Lists of parameters

WHAT WE HAVE

The state of the art

significant improvement but...

MUST HAVE IDENTICAL

- CFGs
- Return types
- Lists of parameters
- Number of instructions

WHAT WE WANT

WHAT WE WANT

Merge **ANY** two functions

WHAT WE WANT

Merge **ANY** two functions

and

CHOOSE when to do it

HOW WE DO IT

```
int f1(int a, int b) {
  int var = a*b + 10;
  int result = foo(var);
  printf("result: %d\n", result);
  return result;
}
```

```
void f2(int a, int b) {
  int var = a*b + b;
  int result = foo(var);
  printf("result: %d\n", result);
  if (result==0)
    printf("result is zero\n");
}
```

HOW WE DO IT

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

IR Level

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
```

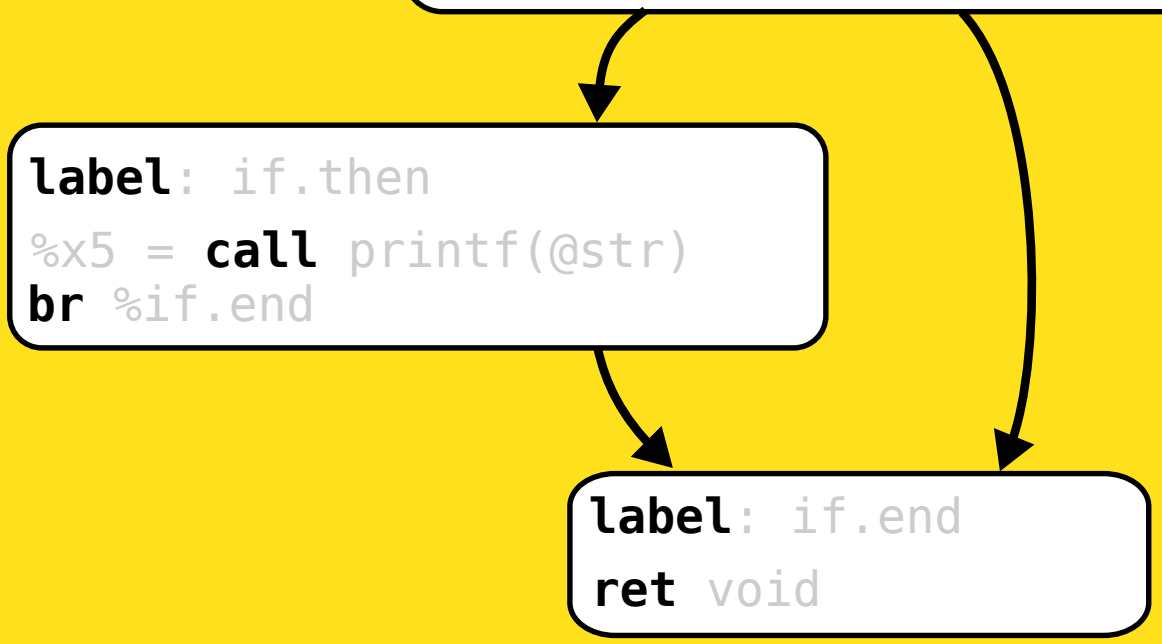
```
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
ret void
```

HOW WE DO IT

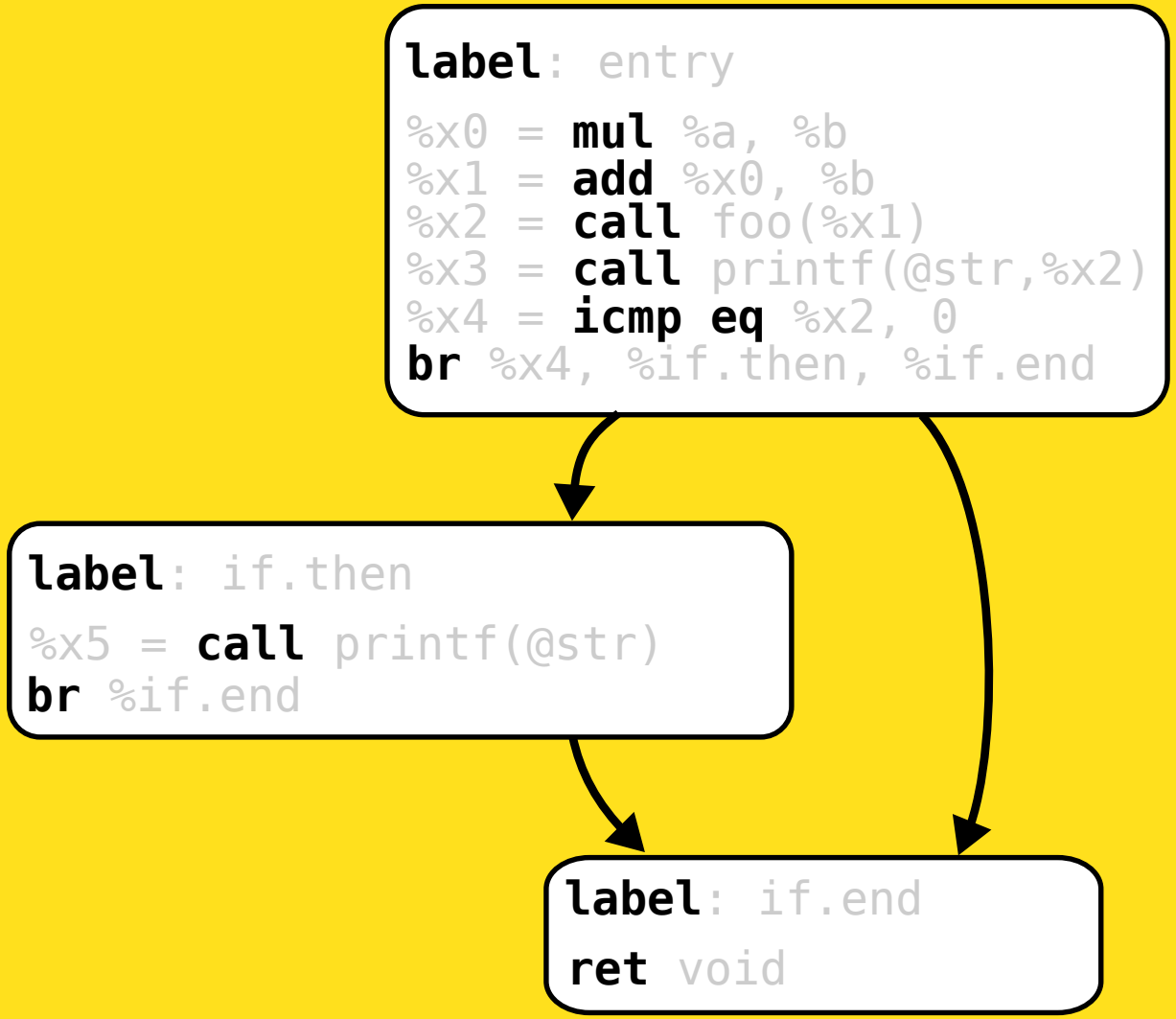
```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
```



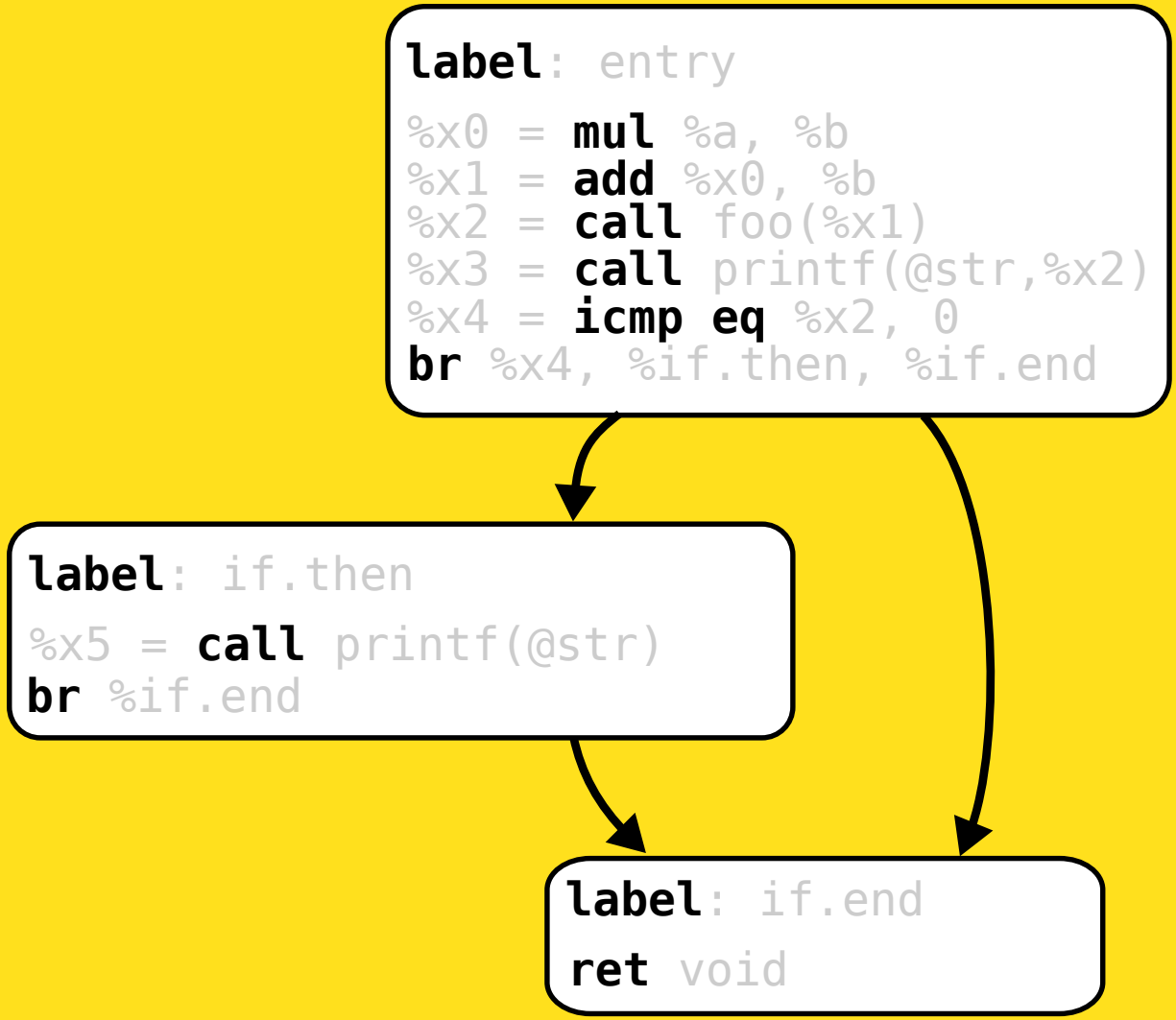
LINEARIZATION

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```



LINEARIZATION

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```



LINEARIZATION

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
ret void
```

LINEARIZATION

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
ret void
```

LINEARIZATION

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
ret void
```

LINEARIZATION

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

Match

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

Match

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

Match

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

Match

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

Match

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

Match

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

Match

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

Match

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
```

Mismatch

```
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)

ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)

ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

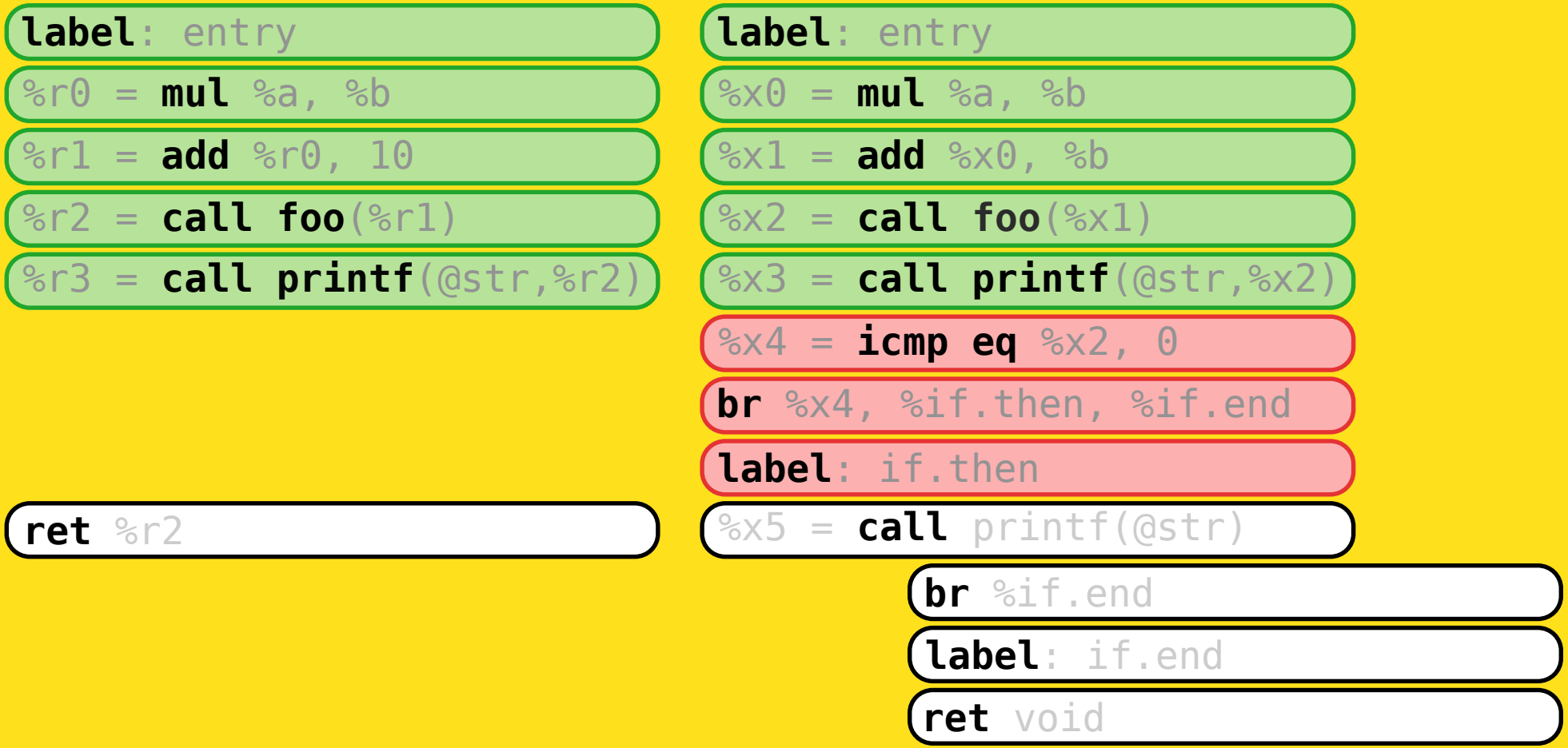
```
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE



IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
```

```
br %if.end
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE

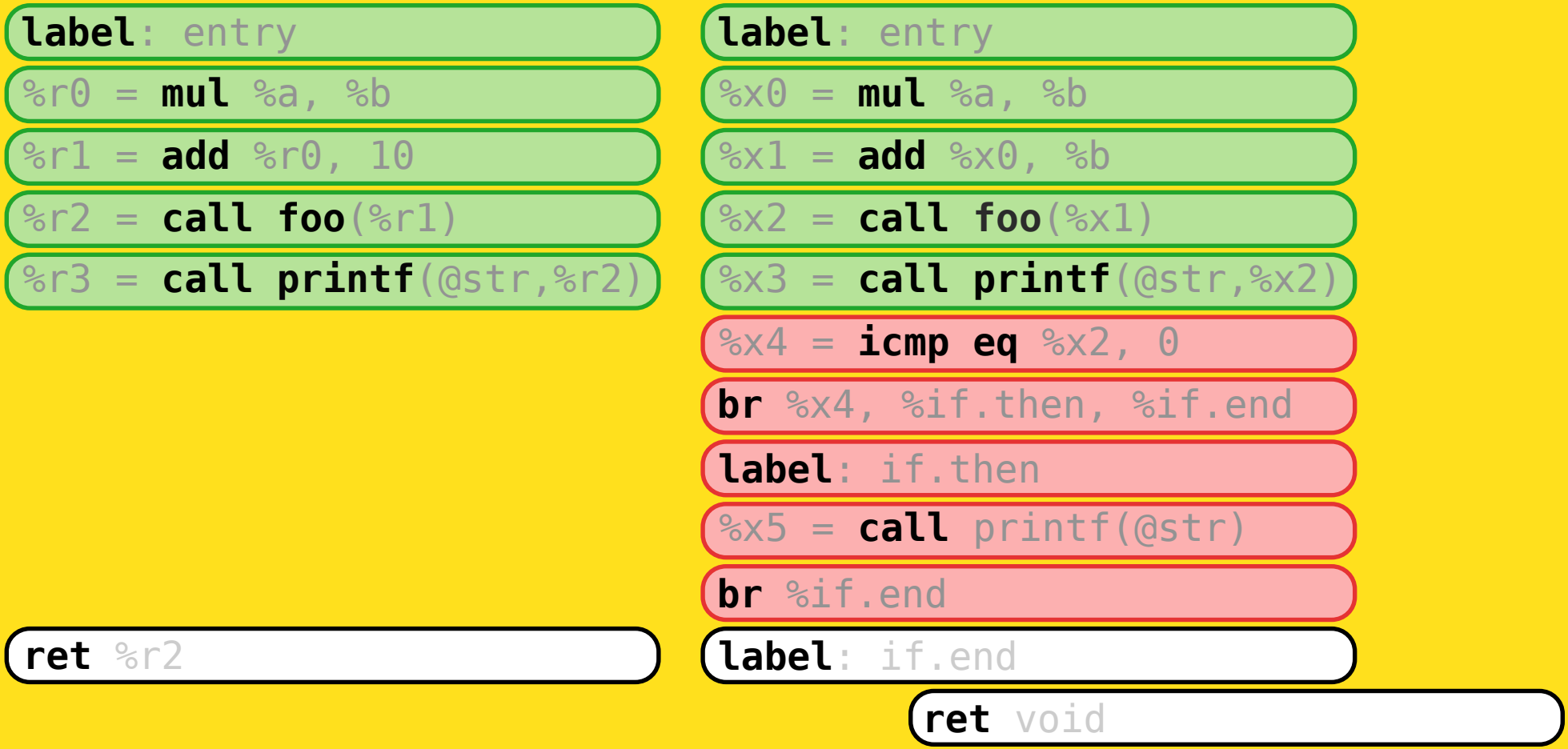
```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
ret void
```

IDENTIFY WHAT TO MERGE



IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

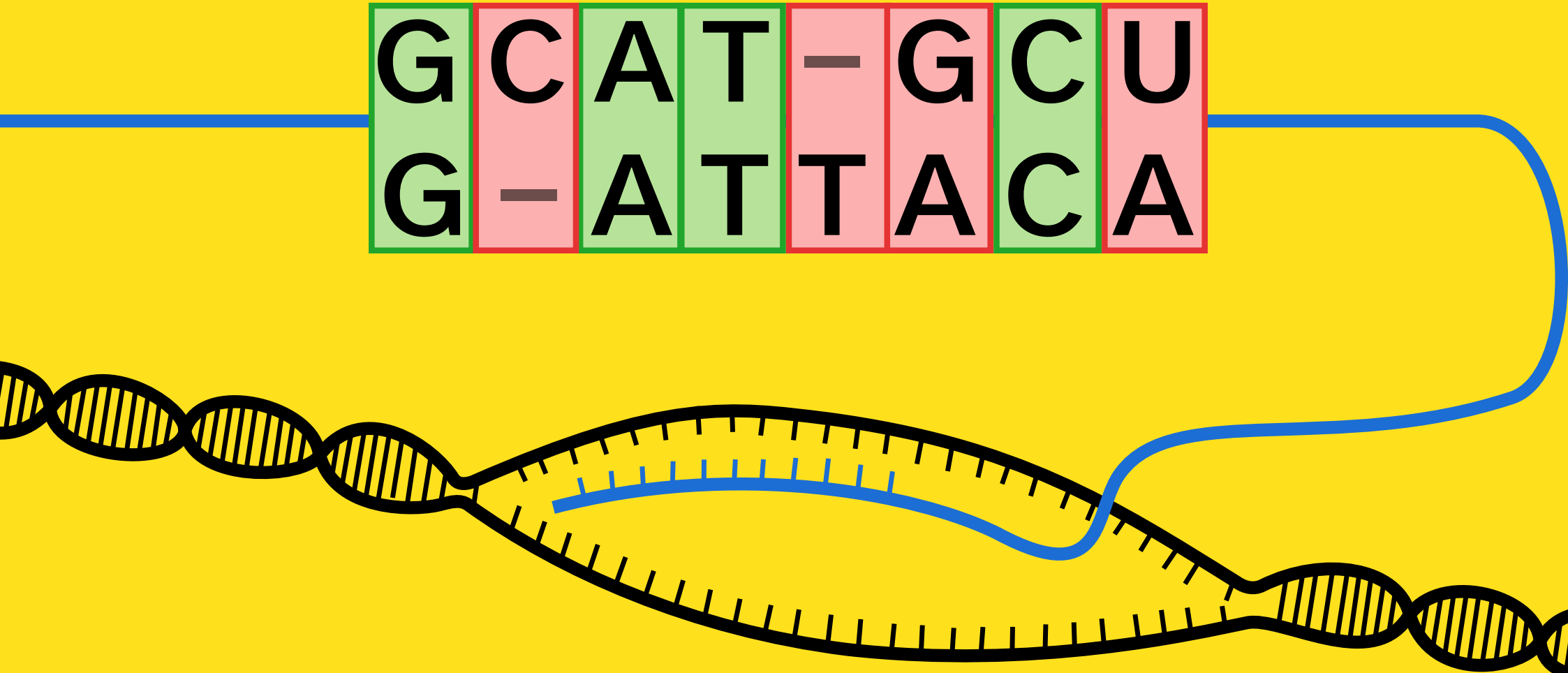
IDENTIFY WHAT TO MERGE

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

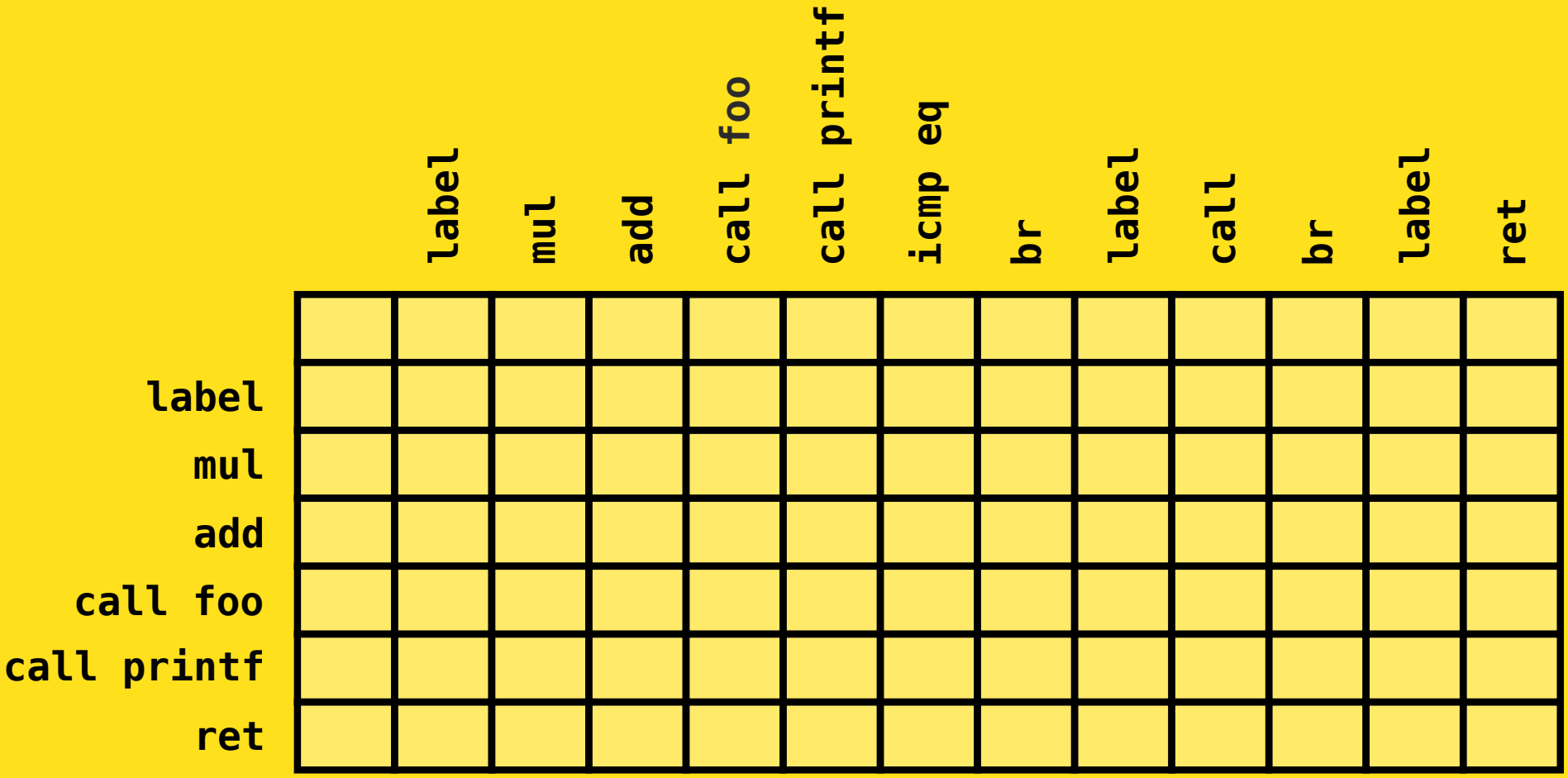
```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

SEQUENCE ALIGNMENT

Bioinformatics

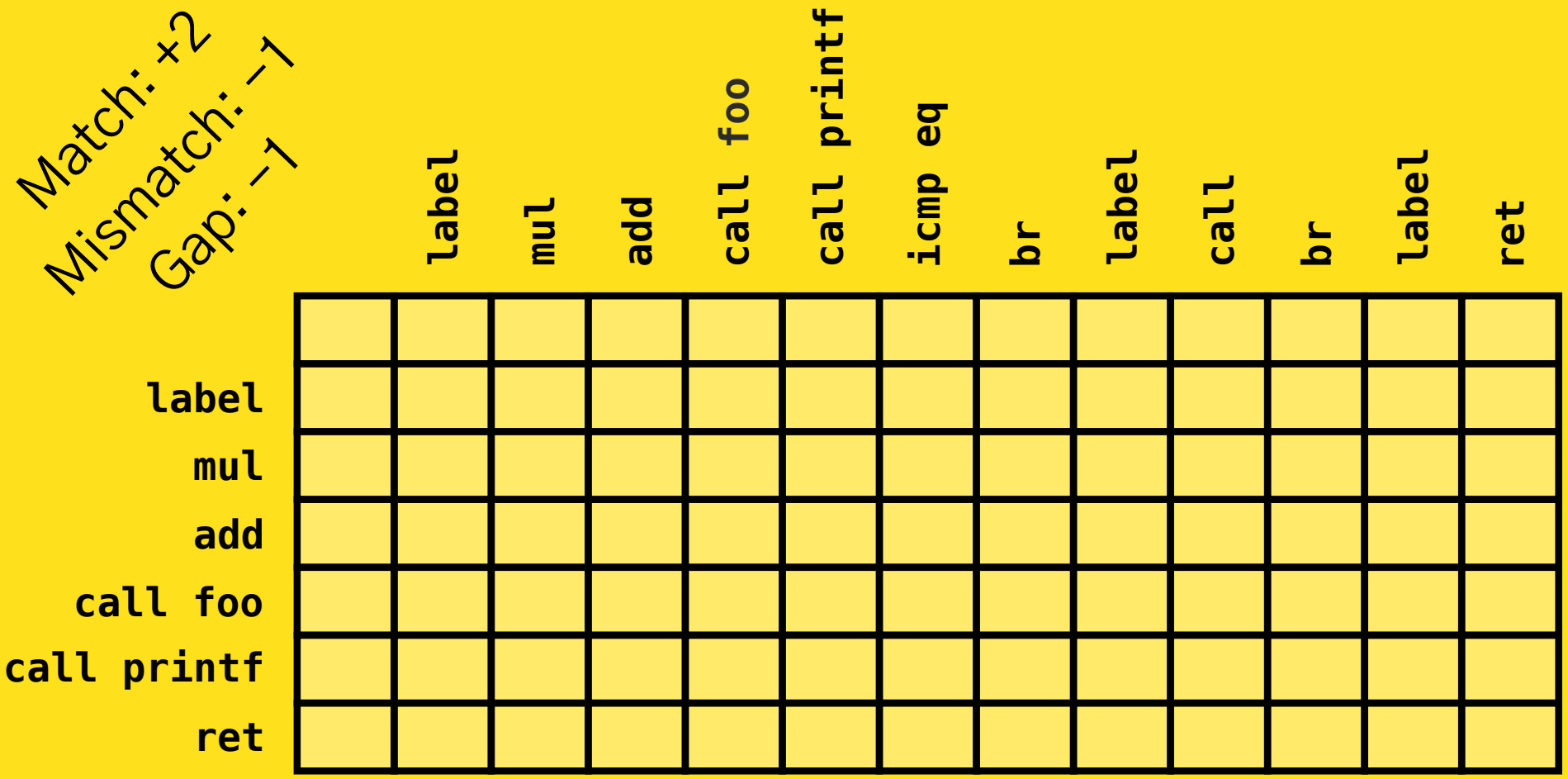


SEQUENCE ALIGNMENT



SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1



SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1												
mul	−2												
add	−3												
call foo	−4												
call printf	−5												
ret	−6												

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1												
mul	−2												
add	−3												
call foo	−4												
call printf	−5												
ret	−6												

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	**−1**												
mul	−2												
add	−3												
call foo	−4												
call printf	−5												
ret	−6												

→ −1 −1 = −2

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	**−1**	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1												
mul	−2												
add	−3												
call foo	−4												
call printf	−5												
ret	−6												

→ −1 −1 = −2

↓ −1 −1 = −2

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1												
mul	−2												
add	−3												
call foo	−4												
call printf	−5												
ret	−6												

→ −1 −1 = −2

↓ −1 −1 = −2

↘ 0 + 2 = 2

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1	**2**											
mul	−2												
add	−3												
call foo	−4												
call printf	−5												
ret	−6												

→ −1 −1 = −2

↓ −1 −1 = −2

↘ **0 + 2 = 2**

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	**−1**	**−2**	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1	**2**											
mul	−2												
add	−3												
call foo	−4												
call printf	−5												
ret	−6												

→ 2 −1 = 1

↓ −2 −1 = −3

↘ −1 −1 = −2

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1	2	**1**										
mul	−2												
add	−3												
call foo	−4												
call printf	−5												
ret	−6												

→ **2 −1 = 1**

↓ −2 −1 = −3

↘ −1 −1 = −2

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1	2	1	0	−1	−2	−3	−4	−5	−6	−7	−8	−9
mul	−2	1	4	3	2	1	0	−1	−2	−3	−4	−5	−6
add	−3	0	3	6	5	4	3	2	1	0	−1	−2	−3
call foo	−4	−1	2	5	8	7	6	5	4	3	2	1	0
call printf	−5	−2	1	4	7	10	9	8	7	6	5	4	3
ret	−6	−3	0	3	6	9	9	8	7	6	5	4	6

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

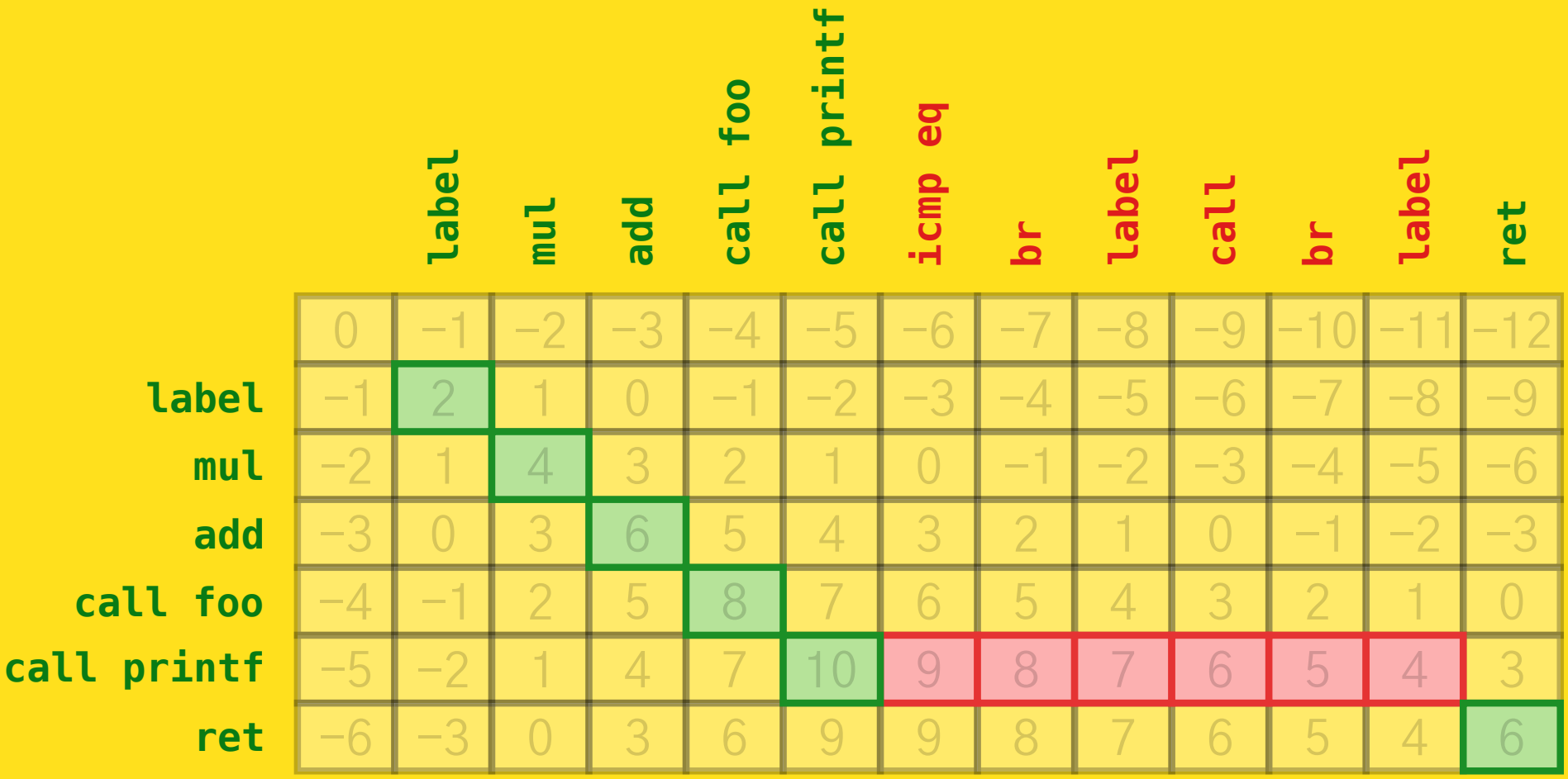
		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1	2	1	0	−1	−2	−3	−4	−5	−6	−7	−8	−9
mul	−2	1	4	3	2	1	0	−1	−2	−3	−4	−5	−6
add	−3	0	3	6	5	4	3	2	1	0	−1	−2	−3
call foo	−4	−1	2	5	8	7	6	5	4	3	2	1	0
call printf	−5	−2	1	4	7	10	9	8	7	6	5	4	3
ret	−6	−3	0	3	6	9	9	8	7	6	5	4	6

SEQUENCE ALIGNMENT

Match: +2
Mismatch: −1
Gap: −1

		label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
label	−1	2	1	0	−1	−2	−3	−4	−5	−6	−7	−8	−9
mul	−2	1	4	3	2	1	0	−1	−2	−3	−4	−5	−6
add	−3	0	3	6	5	4	3	2	1	0	−1	−2	−3
call foo	−4	−1	2	5	8	7	6	5	4	3	2	1	0
call printf	−5	−2	1	4	7	10	9	8	7	6	5	4	3
ret	−6	−3	0	3	6	9	9	8	7	6	5	4	6

SEQUENCE ALIGNMENT



CODE GENERATION

```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
label: entry
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

CODE GENERATION

Renaming Table

entry	b1

```
label: b1
```

```
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x0 = mul %a, %b
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1

```
label: b1
```

```
%r0 = mul %a, %b
```

```
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x0 = mul %a, %b
```

```
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1

```
label: b1
%m0 = mul
```

```
%r0 = mul %a, %b
```

```
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x0 = mul %a, %b
```

```
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1

```
label: b1
%m0 = mul %a, %b
```

```
%r0 = mul %a, %b
```

```
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x0 = mul %a, %b
```

```
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0

```
label: b1
%m0 = mul %a, %b
```

```
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0

```
label: b1
%m0 = mul %a, %b
%m1 = add
```

```
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0

```
label: b1
%m0 = mul %a, %b
%m1 = add
```

```
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0

```
label: b1
%m0 = mul %a, %b
%m1 = add %m0,
```

```
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0

```
label: b1
%m0 = mul %a, %b
%m1 = add %m0,
```

```
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0,
```

```
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
%x1 = add %x0, %b
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
```

```
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x2 = call foo(%x1)
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
```

```
%r2 = call foo(%r1)
```

```
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x2 = call foo(%x1)
```

```
%x3 = call printf(@str,%x2)
```

```
%x4 = icmp eq %x2, 0
```

```
br %x4, %if.then, %if.end
```

```
label: if.then
```

```
%x5 = call printf(@str)
```

```
br %if.end
```

```
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
```

```
%r2 = call foo(%r1)
```

```
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x2 = call foo(%x1)
```

```
%x3 = call printf(@str,%x2)
```

```
%x4 = icmp eq %x2, 0
```

```
br %x4, %if.then, %if.end
```

```
label: if.then
```

```
%x5 = call printf(@str)
```

```
br %if.end
```

```
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
```

```
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
```

```
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
```

```
%r3 = call printf(@str,%r2)
```

```
ret %r2
```

```
%x3 = call printf(@str,%x2)
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
```

```
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

```
ret %r2
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
```

```
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

```
ret %r2
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
```

```
%x4 = icmp eq %x2, 0
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

```
ret %r2
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq
```

```
%x4 = icmp eq %x2, 0
```

```
br %x4, %if.then, %if.end
```

```
label: if.then
```

```
%x5 = call printf(@str)
```

```
br %if.end
```

```
label: if.end
```

```
ret %r2
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
```

```
%x4 = icmp eq %x2, 0
```

```
br %x4, %if.then, %if.end
```

```
label: if.then
```

```
%x5 = call printf(@str)
```

```
br %if.end
```

```
label: if.end
```

```
ret %r2
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
```

```
br %x4, %if.then, %if.end
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

```
ret %r2
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
```

```
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret %r2
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

```
ret %r2
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
```

```
%x5 = call printf(@str)
br %if.end
label: if.end
```

```
ret %r2
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
```

```
%x5 = call printf(@str)
br %if.end
label: if.end
ret void
```

```
ret %r2
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
```

```
%x5 = call printf(@str)
```

```
br %if.end
```

```
label: if.end
```

```
ret %r2
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
```

```
br %if.end
```

```
label: if.end
```

```
ret %r2
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
```

```
br %if.end
```

```
label: if.end
```

```
ret %r2
```

```
ret void
```

CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3

```
label: b1
%m0 = mul %a, %b
%t0 = select %0, 10, %b
%m1 = add %m0, %t0
%m2 = call foo(%m1)
%m3 = call printf(@str,%m2)
br %0, %b2, %b3
```

```
label: b2
```

```
label: b3
%x4 = icmp eq %m2, 0
br %x4, %if.then, %if.end
```

```
label: if.then
%x5 = call printf(@str)
br %if.end
```

```
label: if.end
```

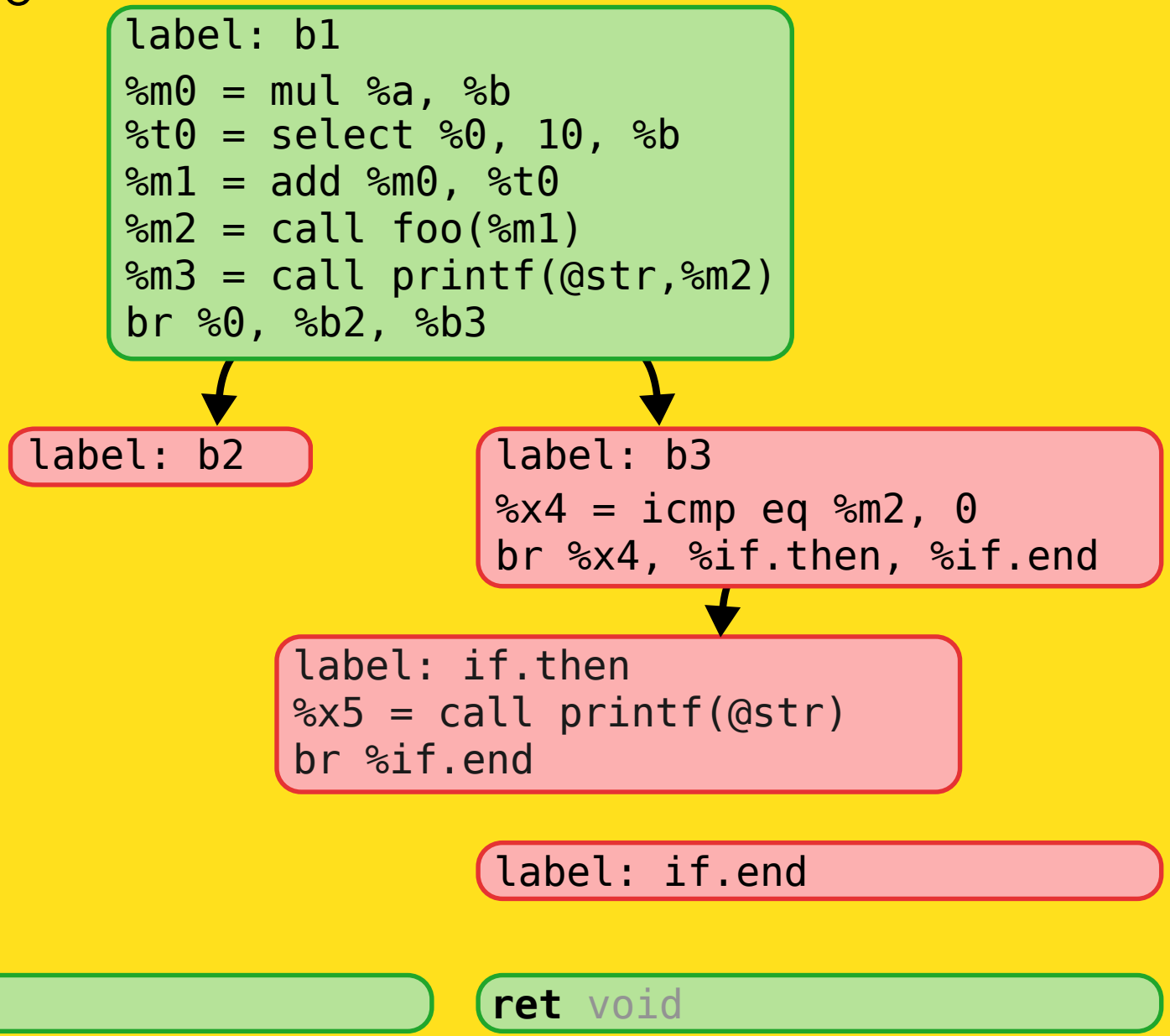
```
ret %r2
```

```
ret void
```

CODE GENERATION

Renaming Table

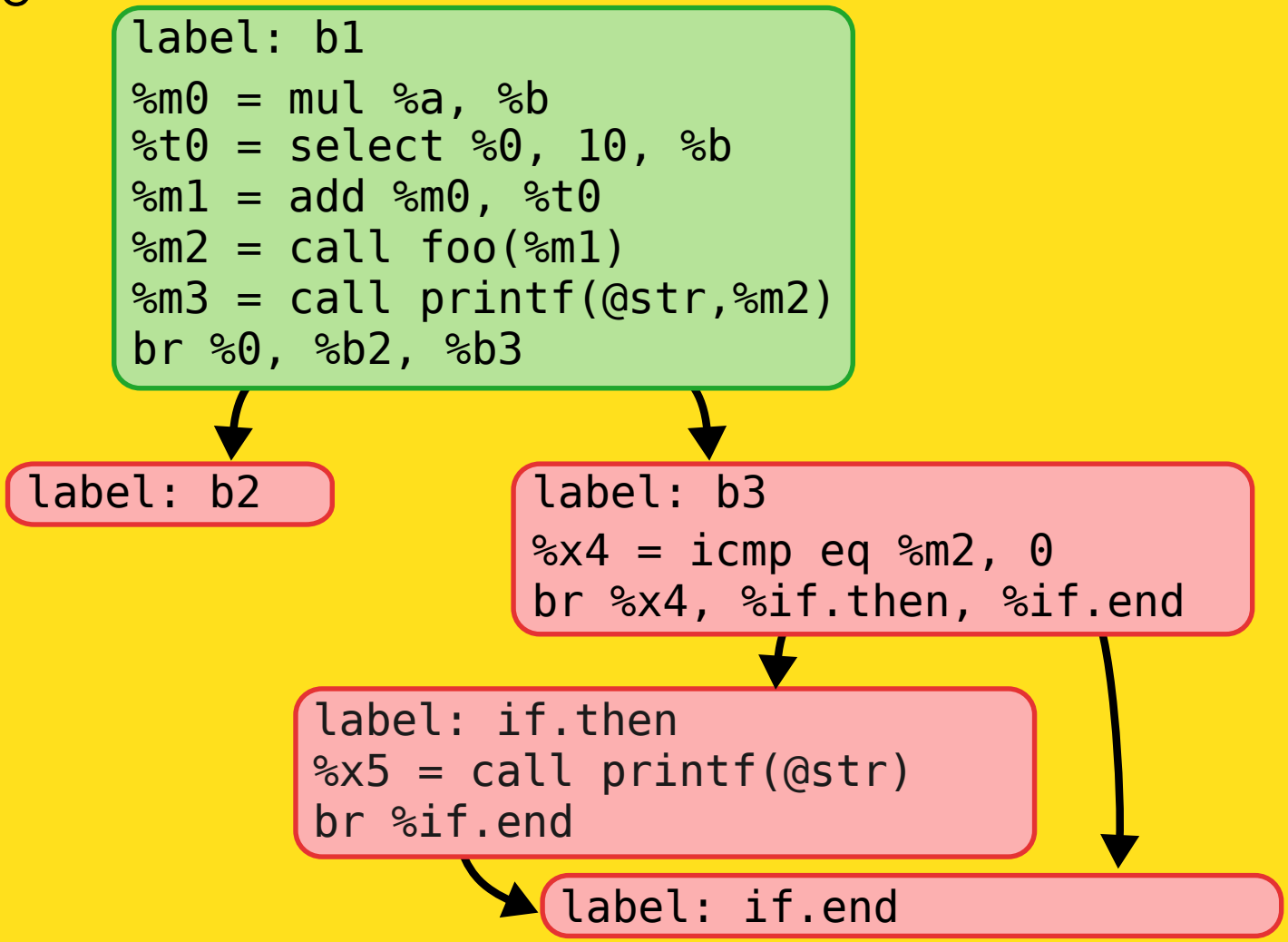
entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3



CODE GENERATION

Renaming Table

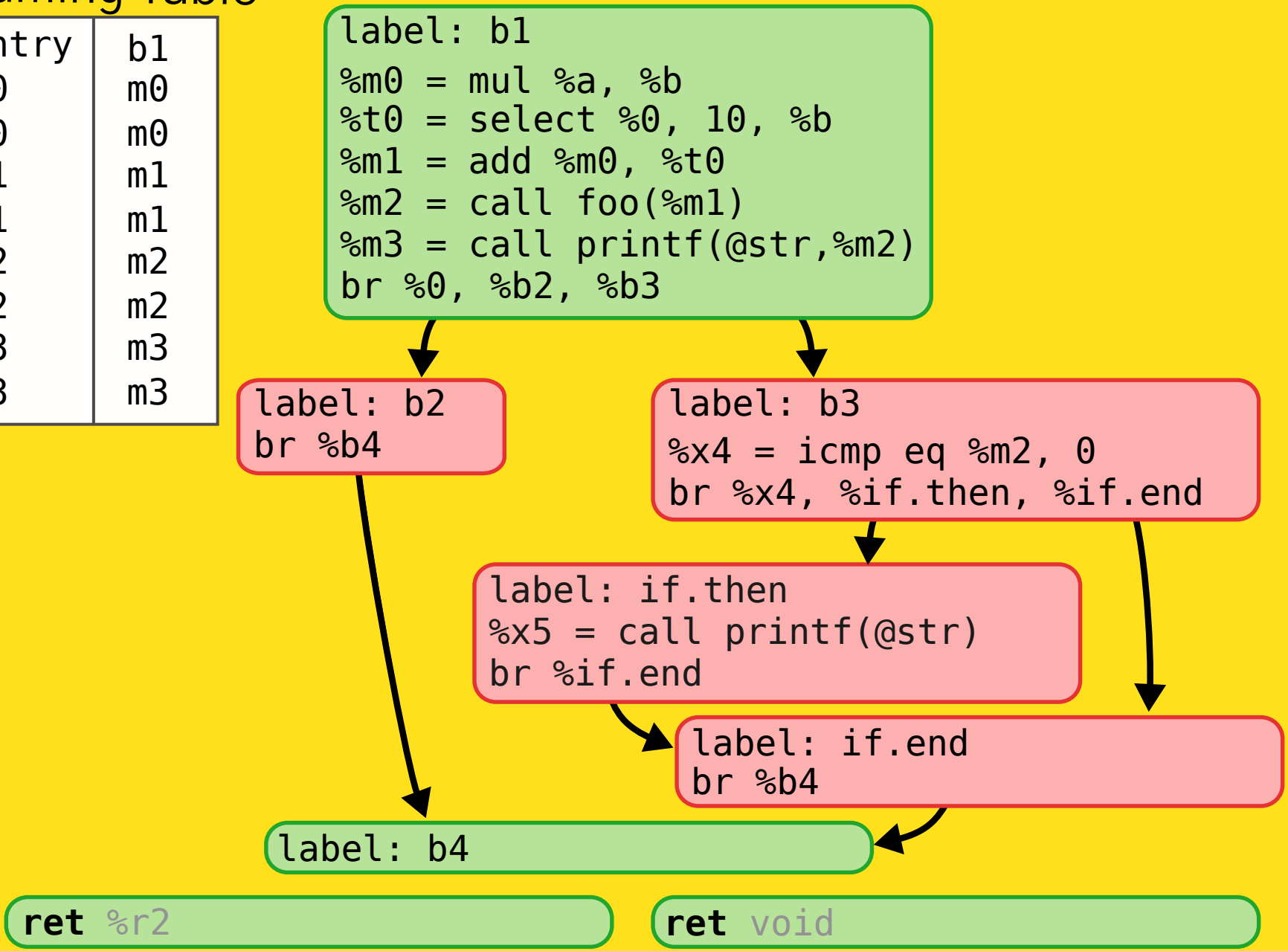
entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3



CODE GENERATION

Renaming Table

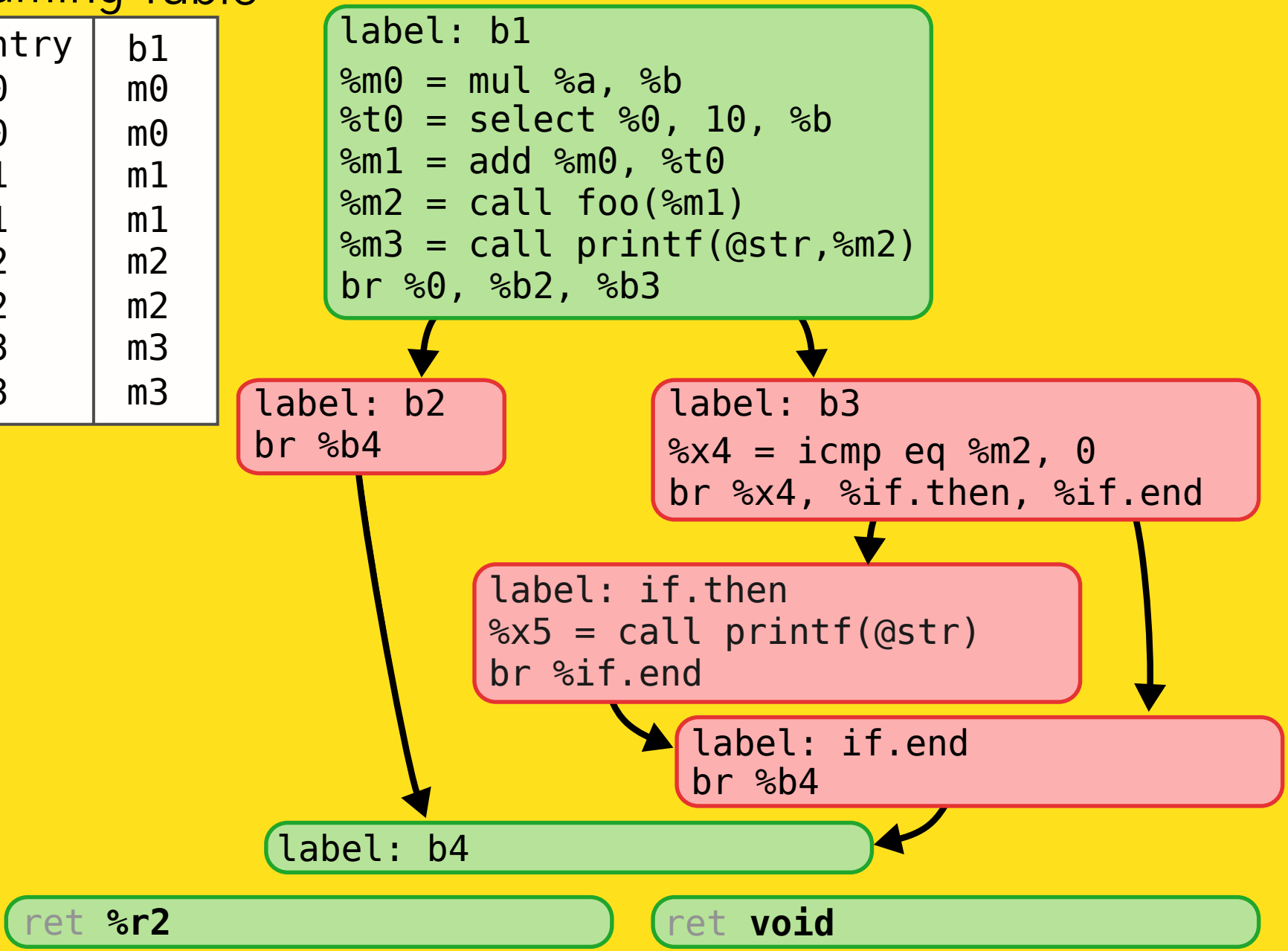
entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3



CODE GENERATION

Renaming Table

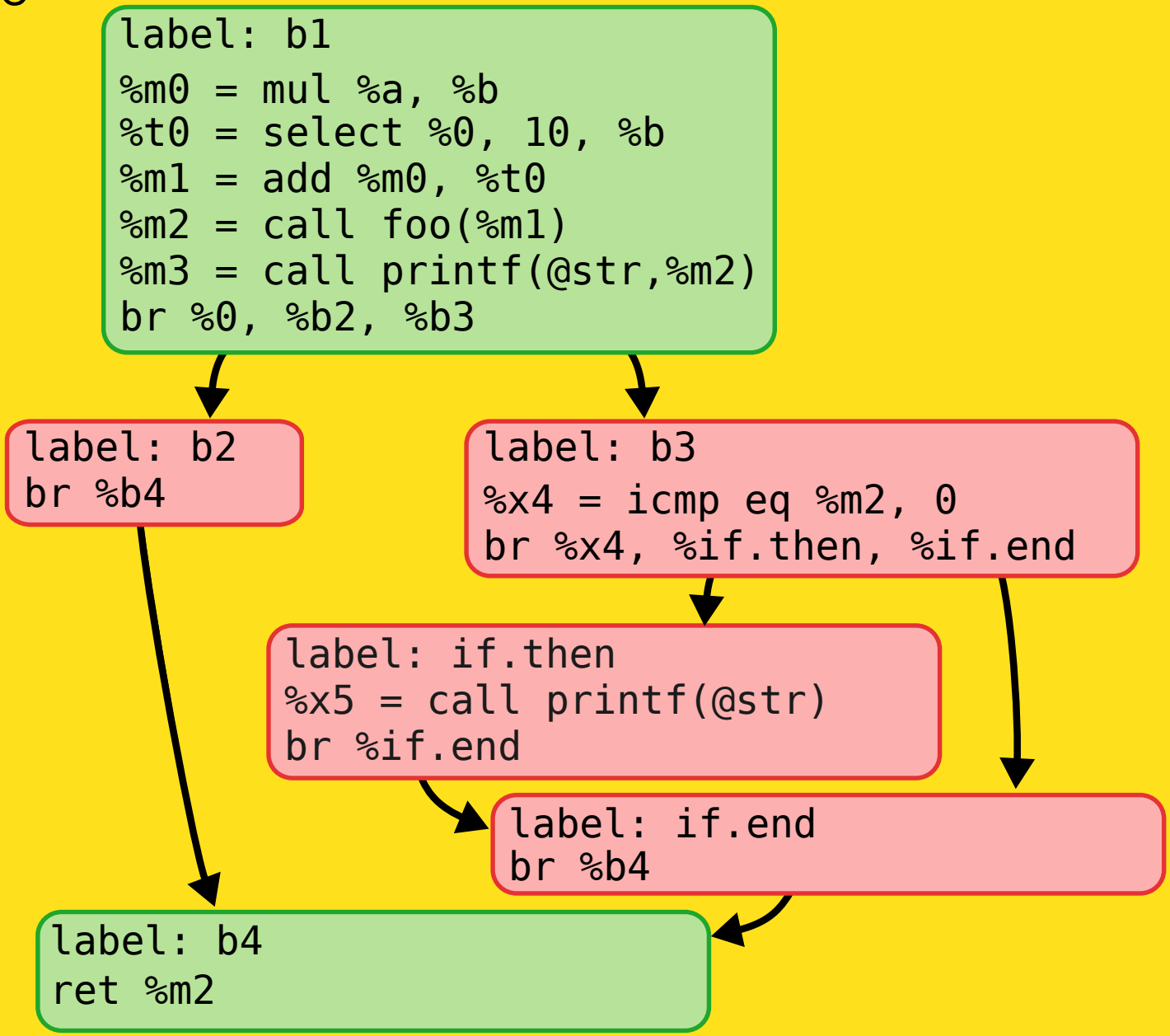
entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3



CODE GENERATION

Renaming Table

entry	b1
r0	m0
x0	m0
r1	m1
x1	m1
r2	m2
x2	m2
r3	m3
x3	m3



MERGING PARAMETERS

Function 1 `i1` `i32` `i32*` `float`

Function 2 `double` `float` `float` `i32` `i32*`

MERGING PARAMETERS

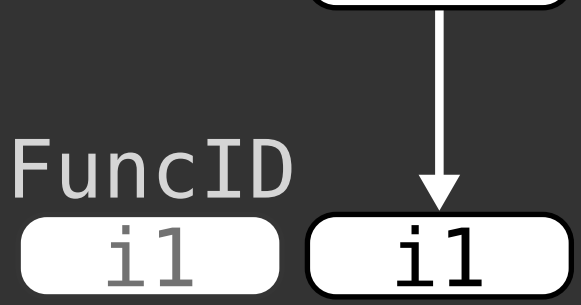
Function 1 `i1` `i32` `i32*` `float`

`FuncID`
`i1`

Function 2 `double` `float` `float` `i32` `i32*`

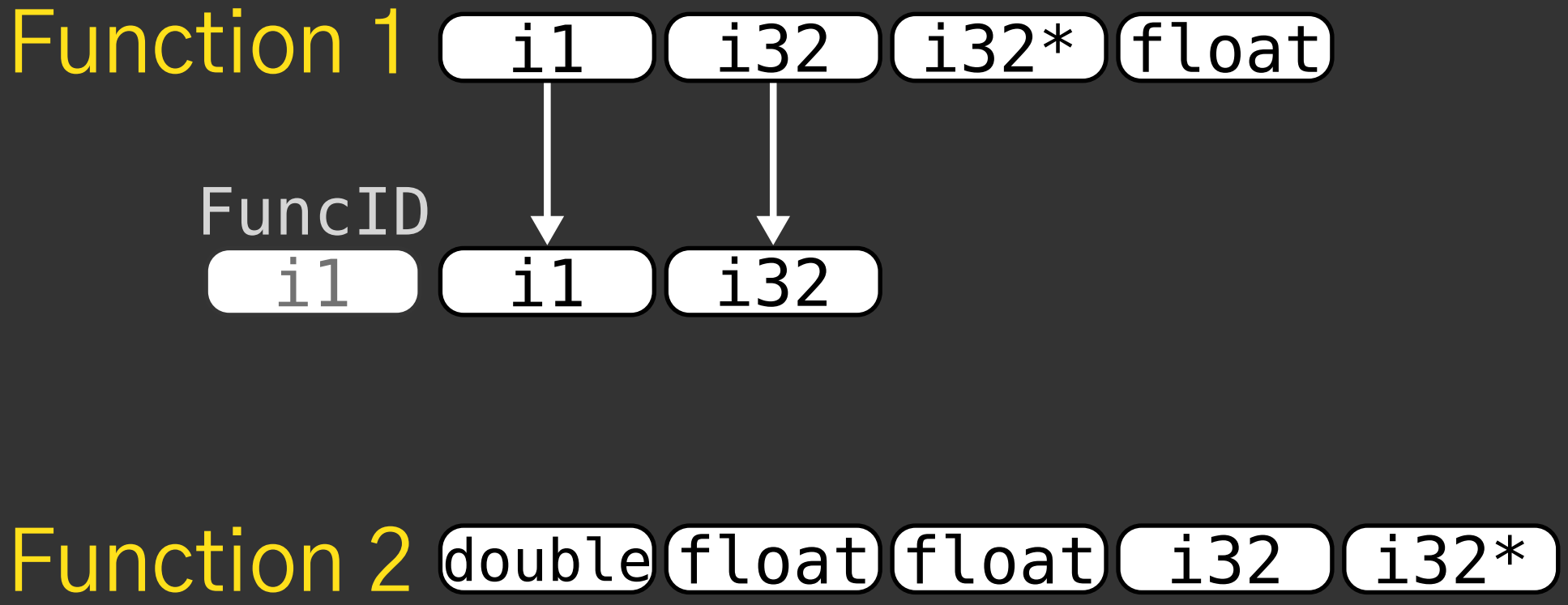
MERGING PARAMETERS

Function 1 `i1` `i32` `i32*` `float`

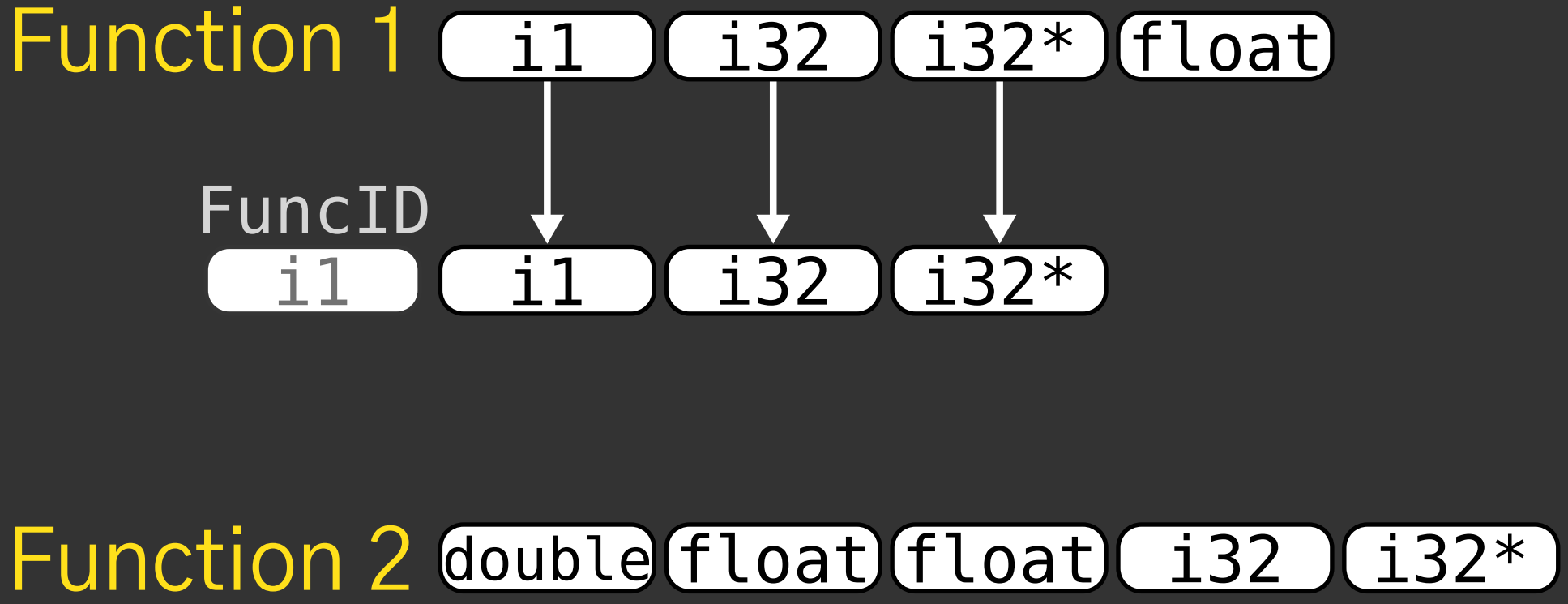


Function 2 `double` `float` `float` `i32` `i32*`

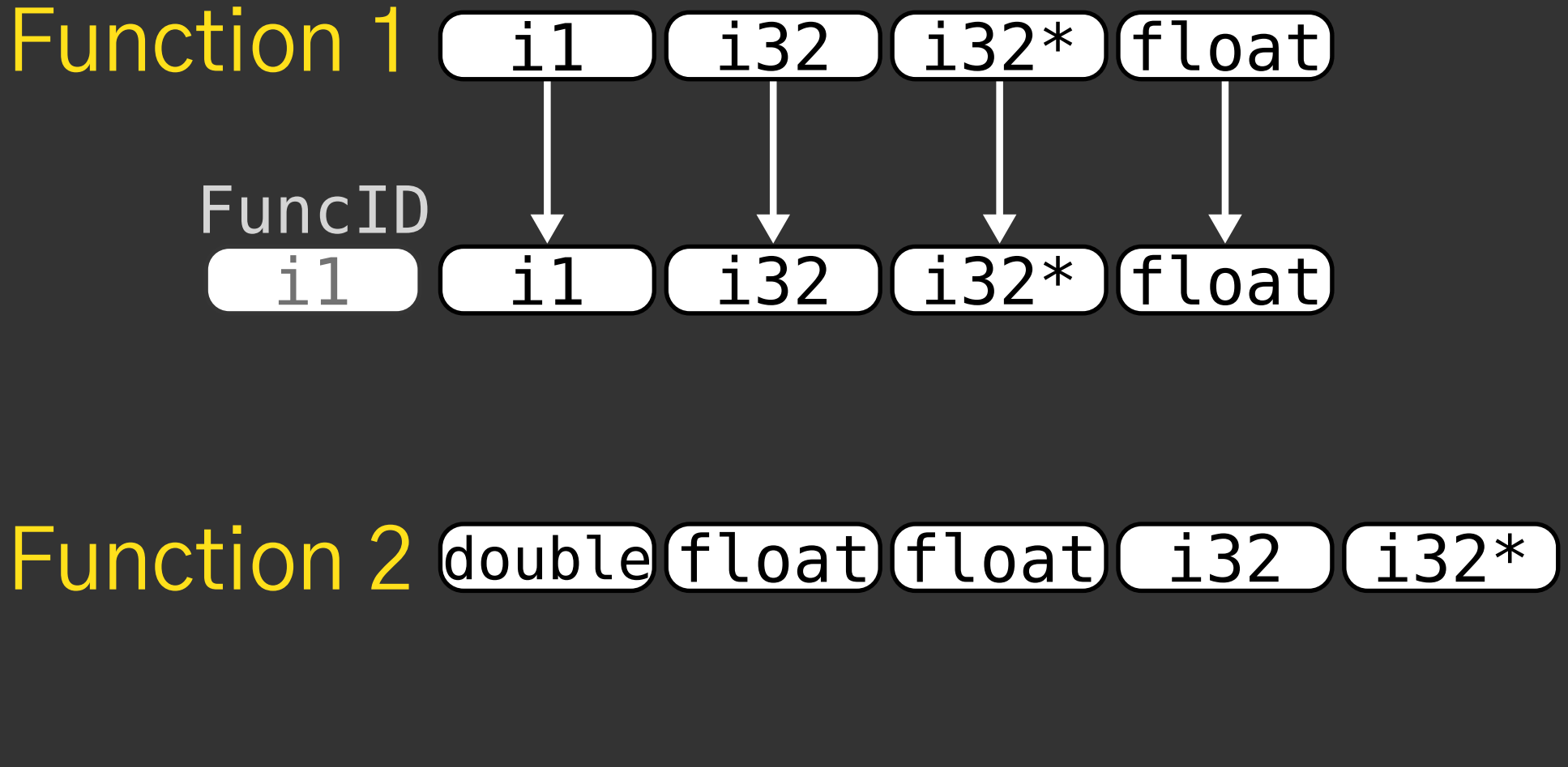
MERGING PARAMETERS



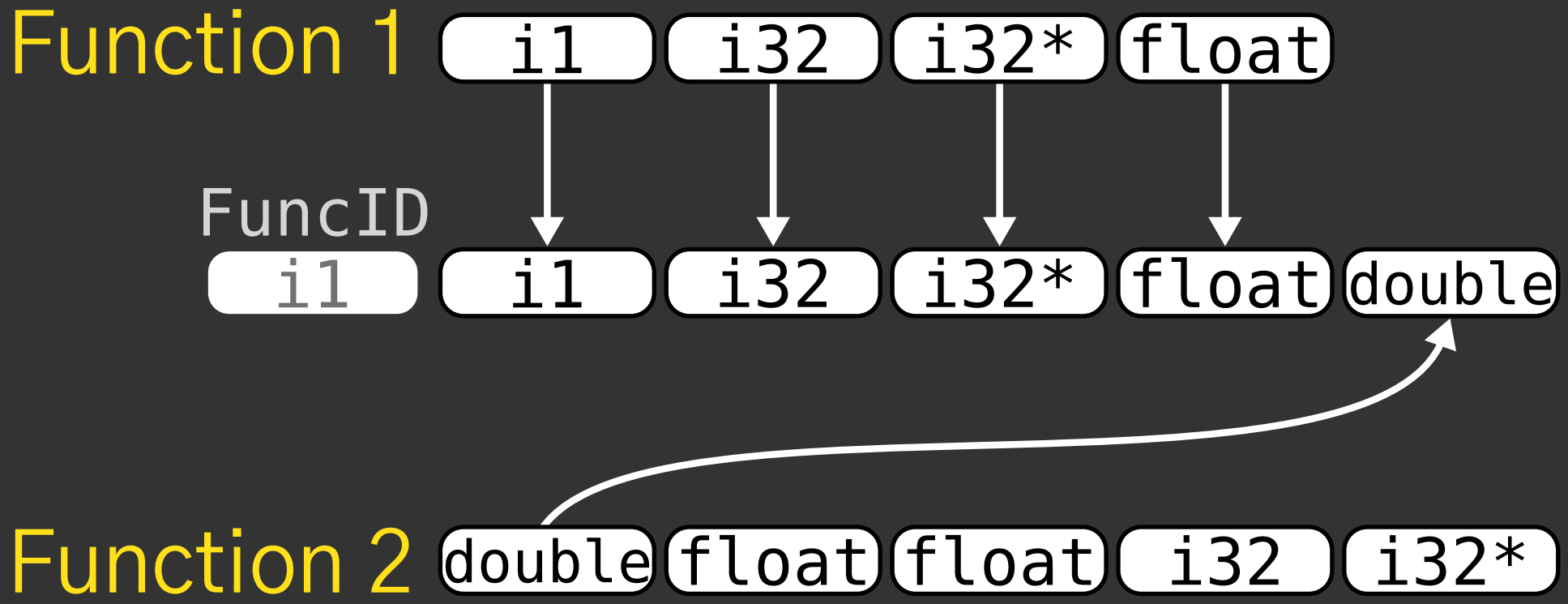
MERGING PARAMETERS



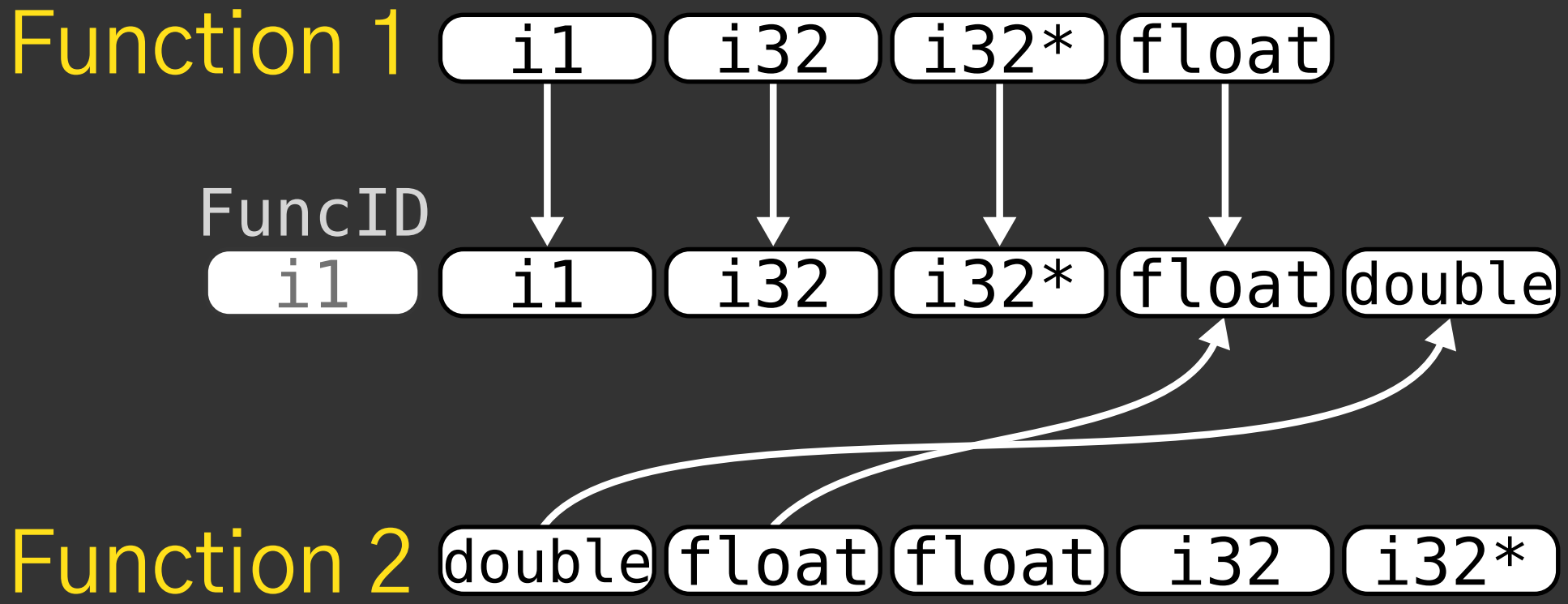
MERGING PARAMETERS



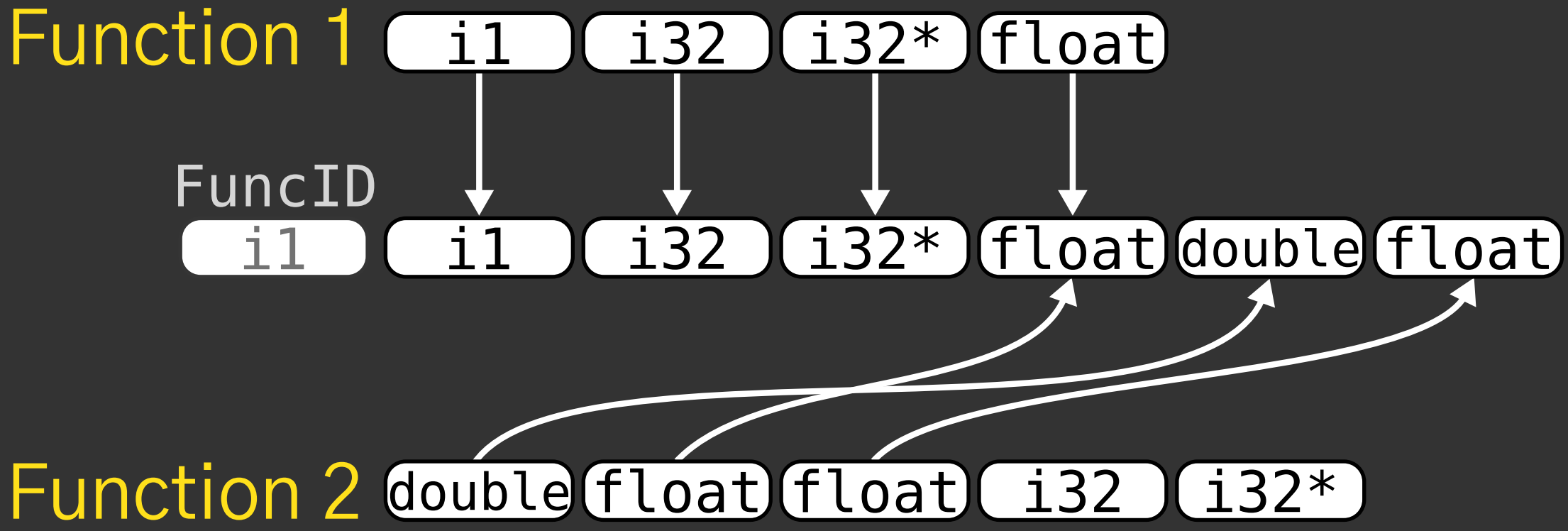
MERGING PARAMETERS



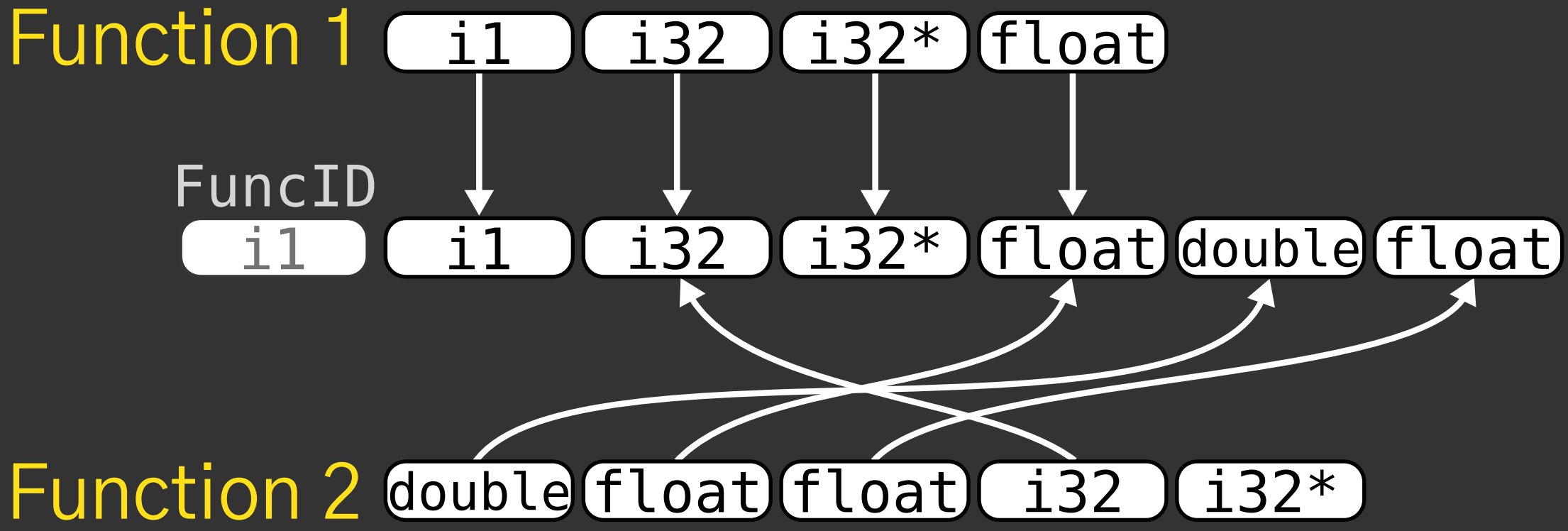
MERGING PARAMETERS



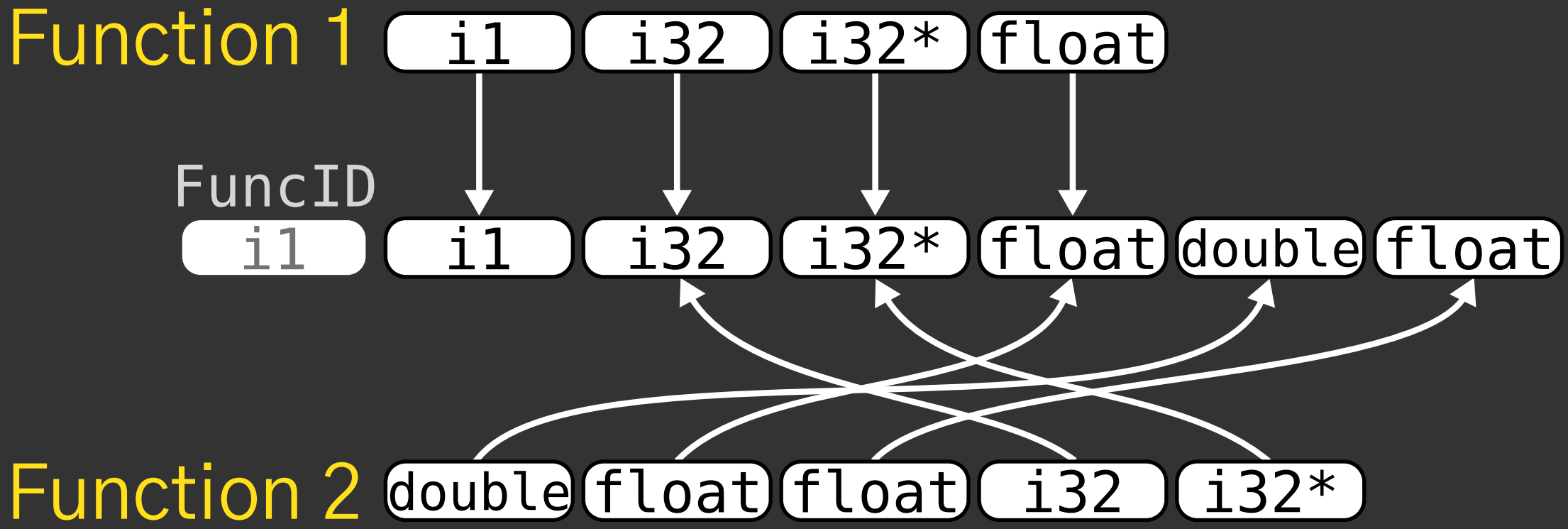
MERGING PARAMETERS



MERGING PARAMETERS



MERGING PARAMETERS

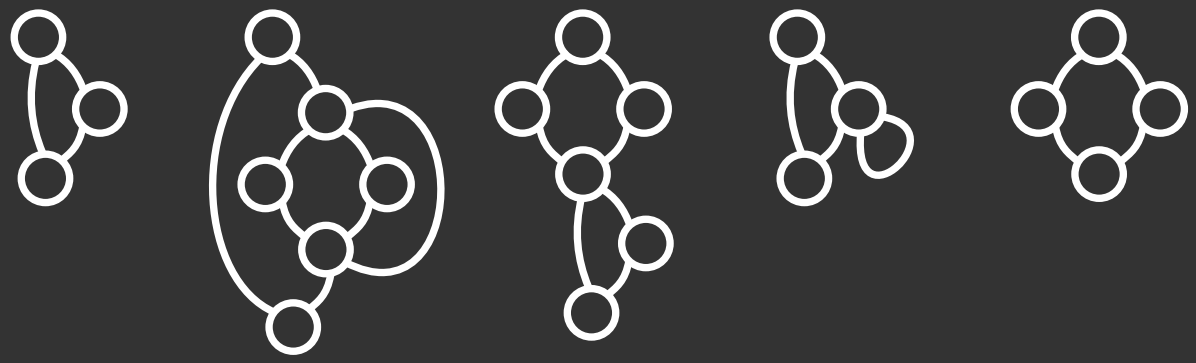


WHAT WE CAN HANDLE

WHAT WE CAN HANDLE

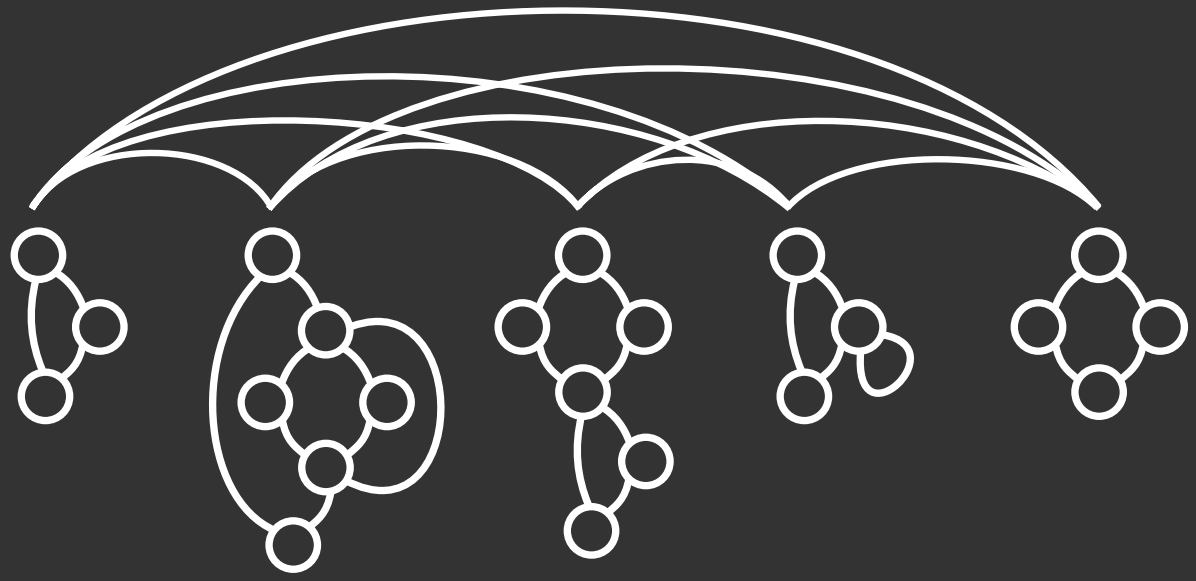
Allowed Differences	LLVM's Identical Only	State of Art	Our Technique (FMSA)
Return Types	✗	✗	✓
List of Parameters	✗	✗	✓
CFGs	✗	✗	✓
Instructions	✗	✓	✓
Operands	✗	✗	✓
Number of Instructions	✗	✗	✓

PROFITABLE MERGES



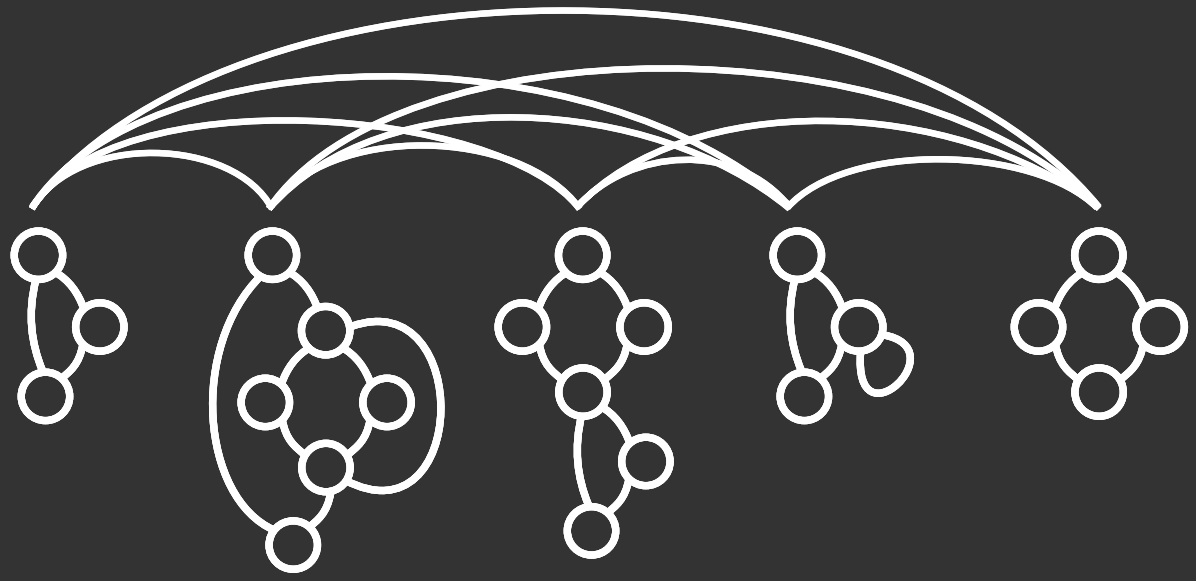
PROFITABLE MERGES

Brute Force

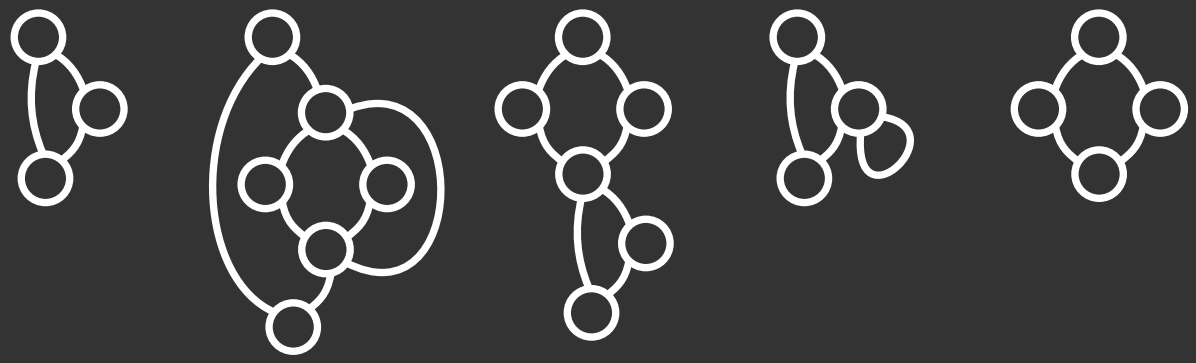


PROFITABLE MERGES

Brute Force: **Infeasible**

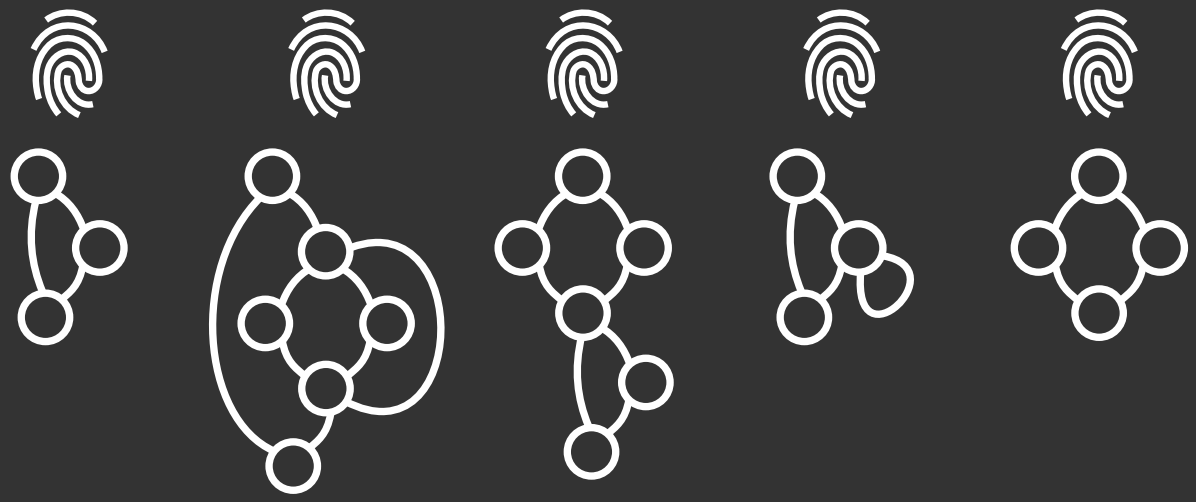


PROFITABLE MERGES



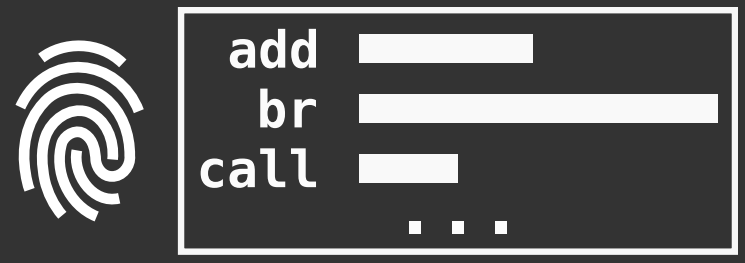
PROFITABLE MERGES

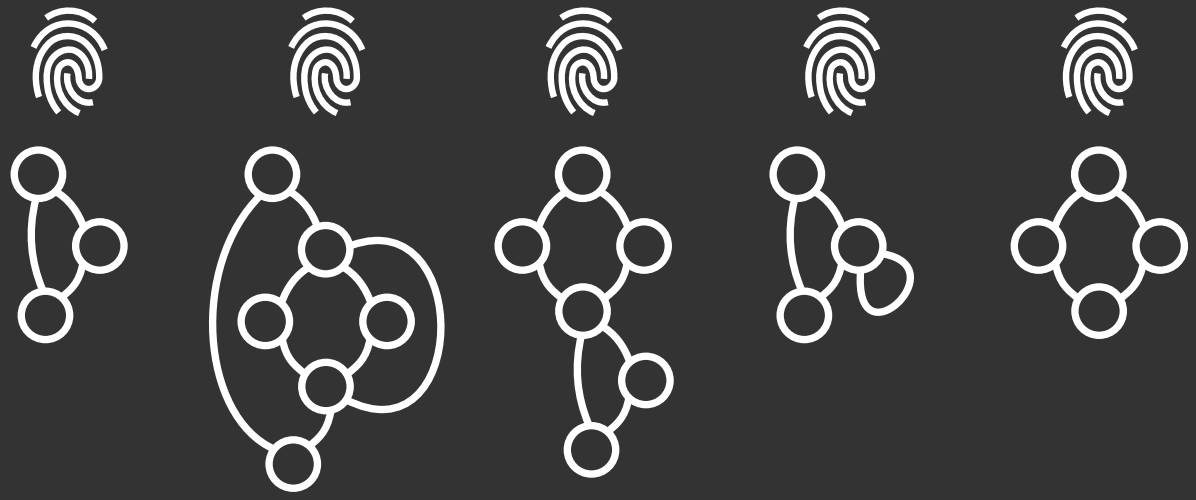
Precompute Fingerprints



PROFITABLE MERGES

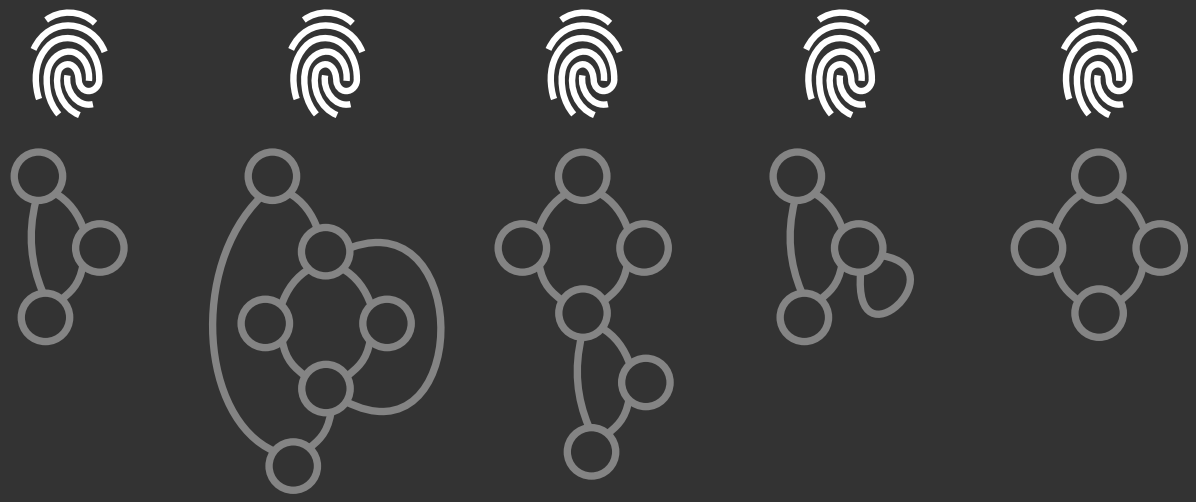
Precompute Fingerprints





PROFITABLE MERGES

Ranking Candidates

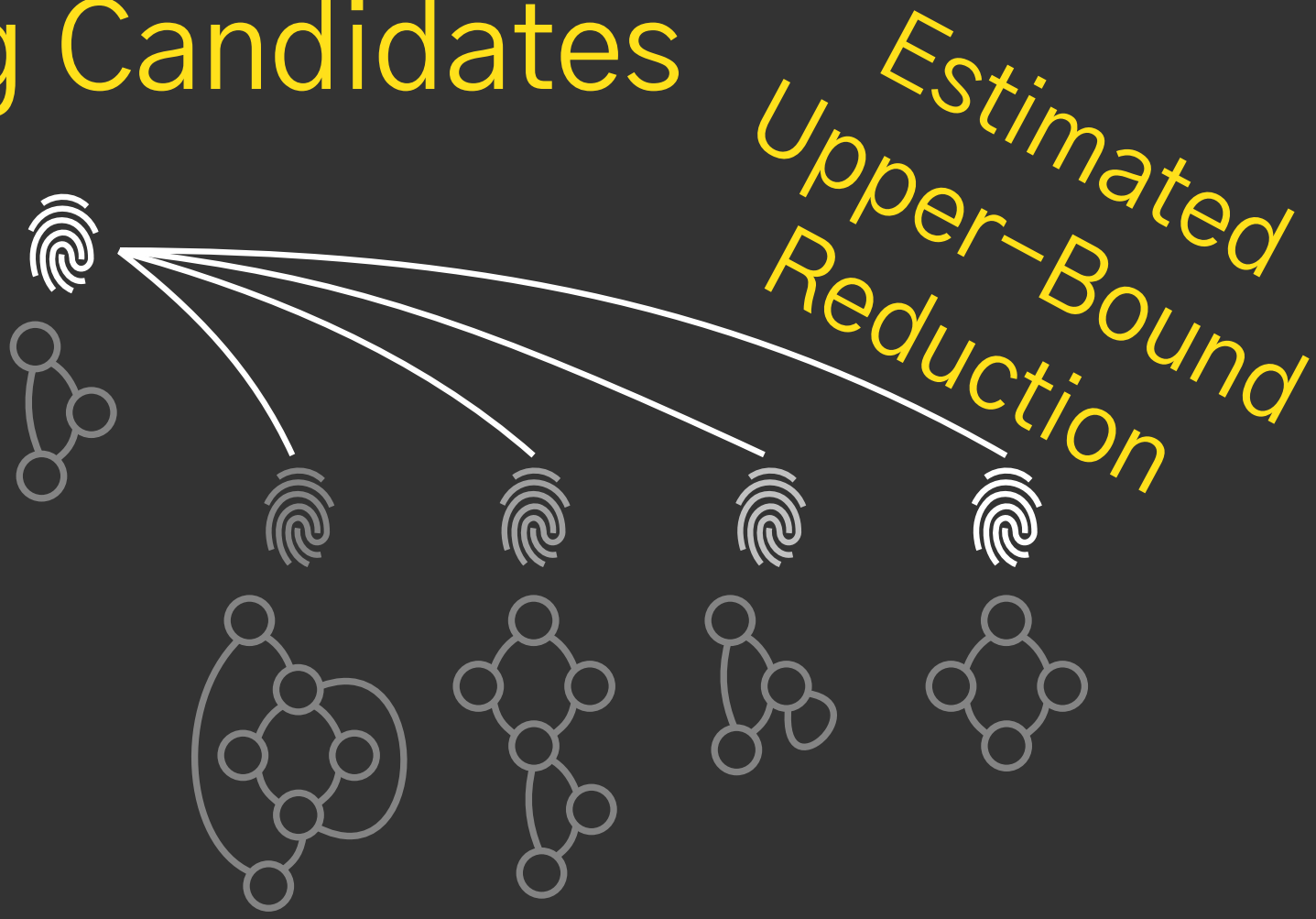


PROFITABLE MERGES

Ranking Candidates

PROFITABLE MERGES

Ranking Candidates



PROFITABLE MERGES

Ranking Candidates

PROFITABLE MERGES

Ranking Candidates

PROFITABLE MERGES

Ranking Candidates

PROFITABLE MERGES

Ranking Candidates

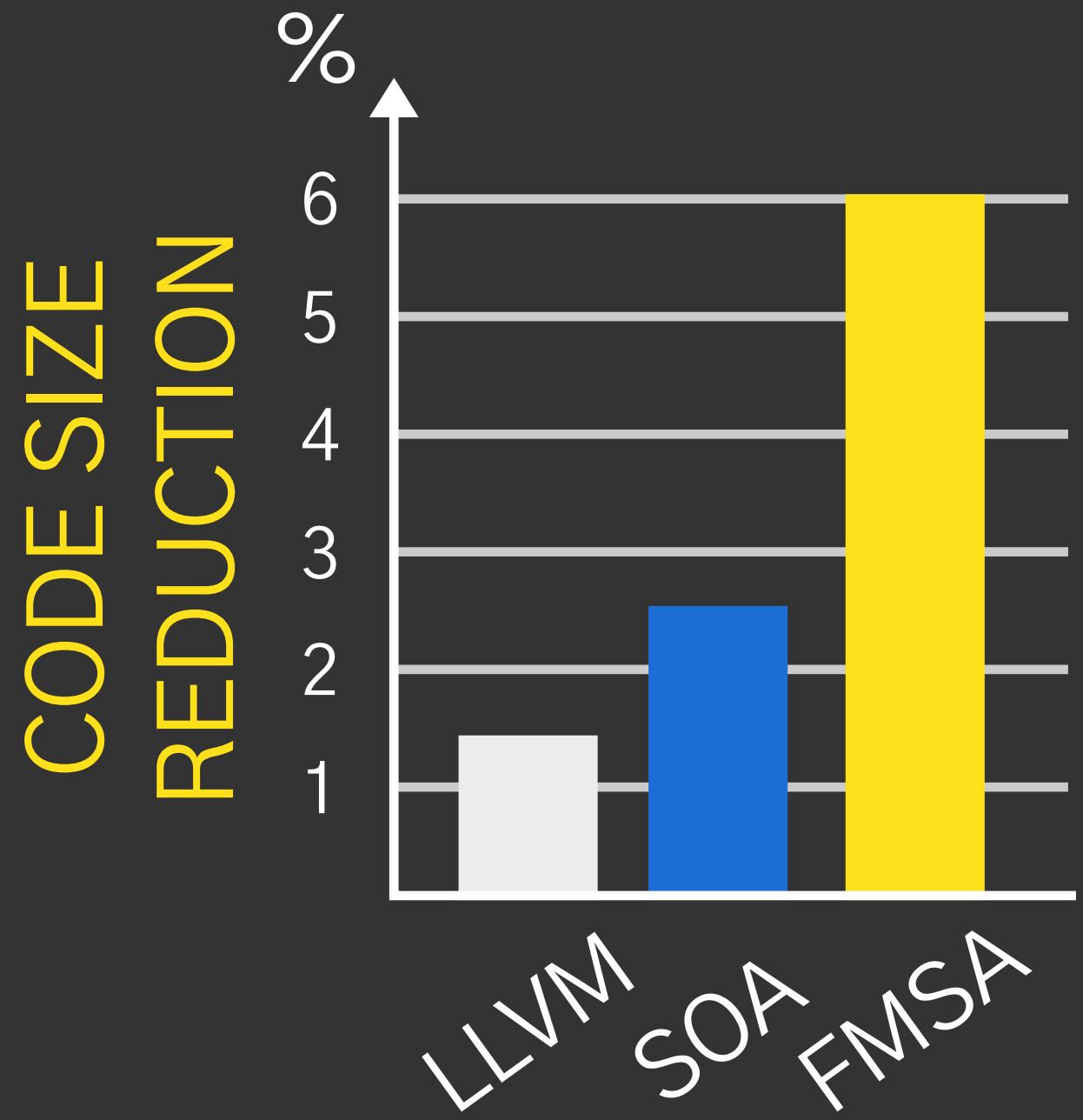
EVALUATION SETUP

LTO with –Os
LLVM: Identical Only
SOA: State of the Art
FMSA: Our Technique

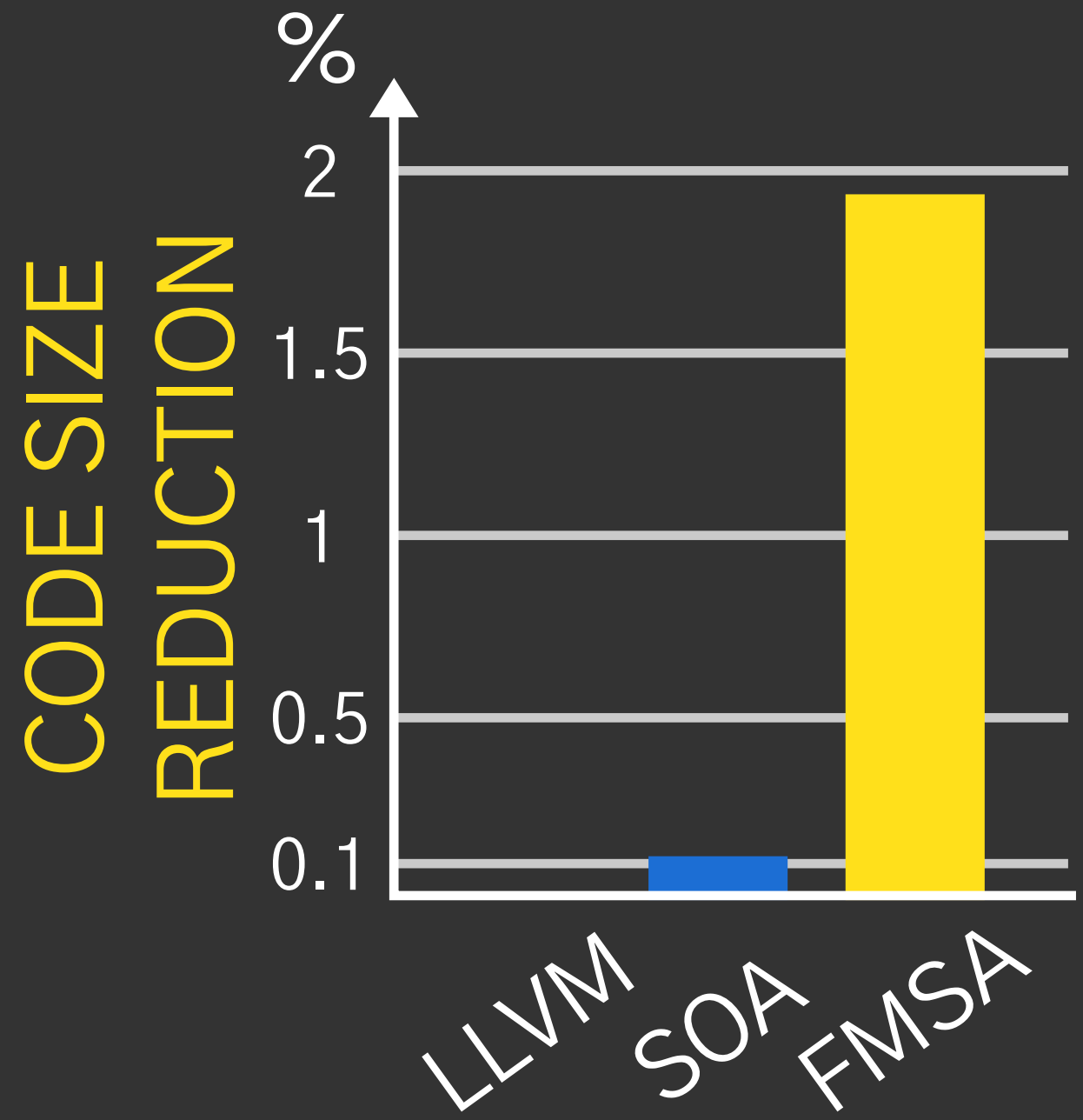
Benchmark Suite:

- SPEC 2006
- MiBench

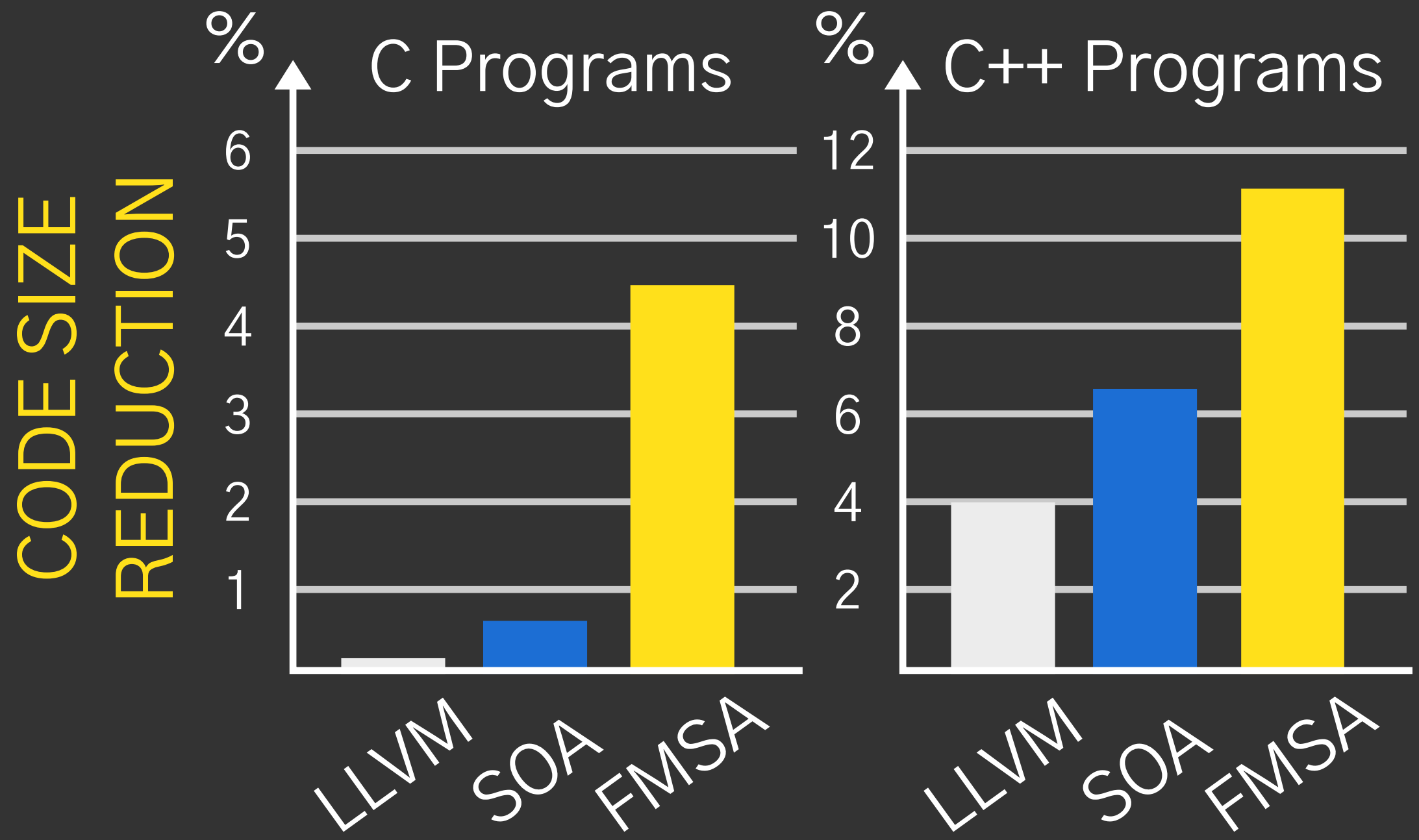
AVERAGE ON SPEC'06



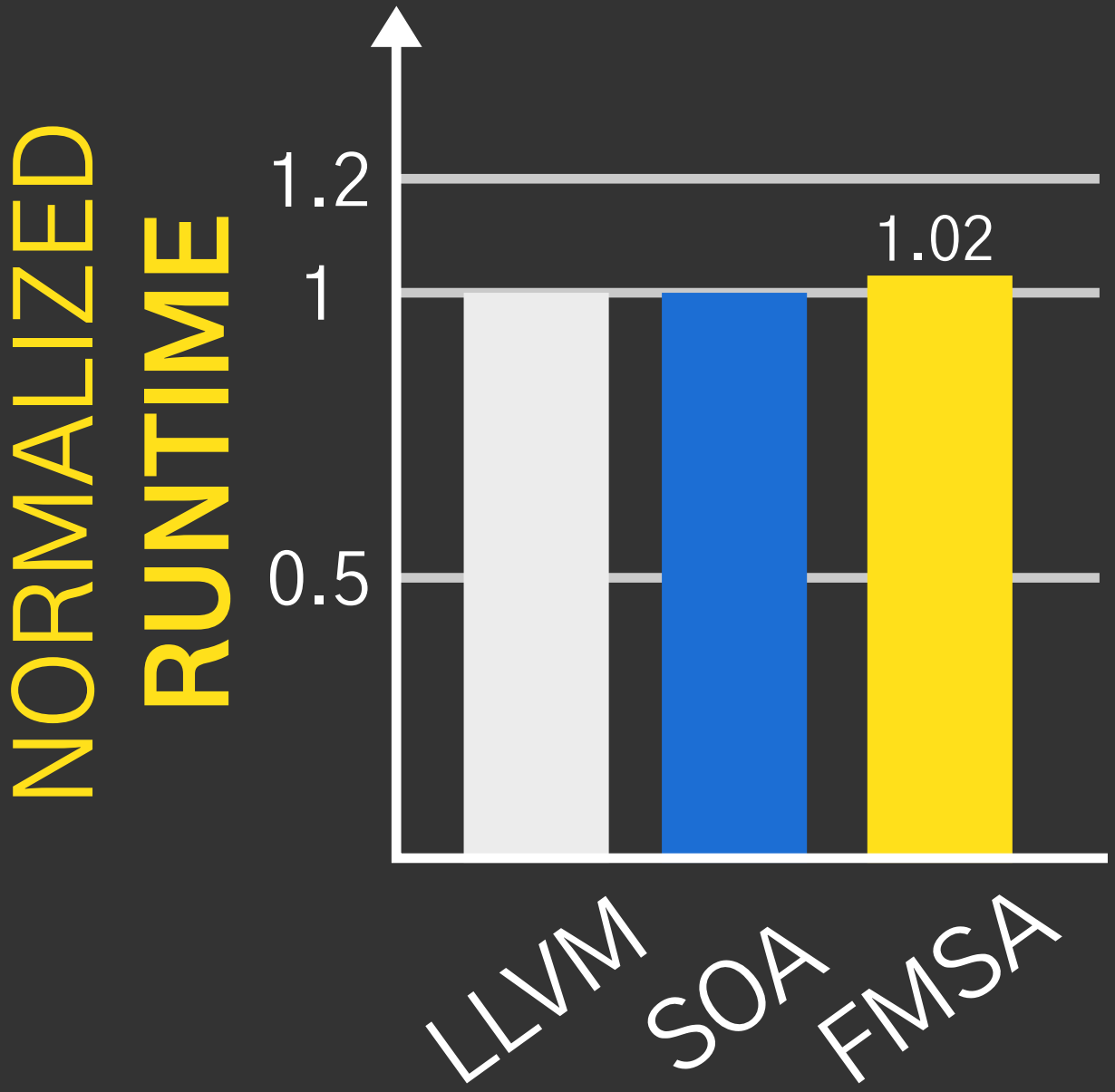
AVERAGE ON MIBENCH



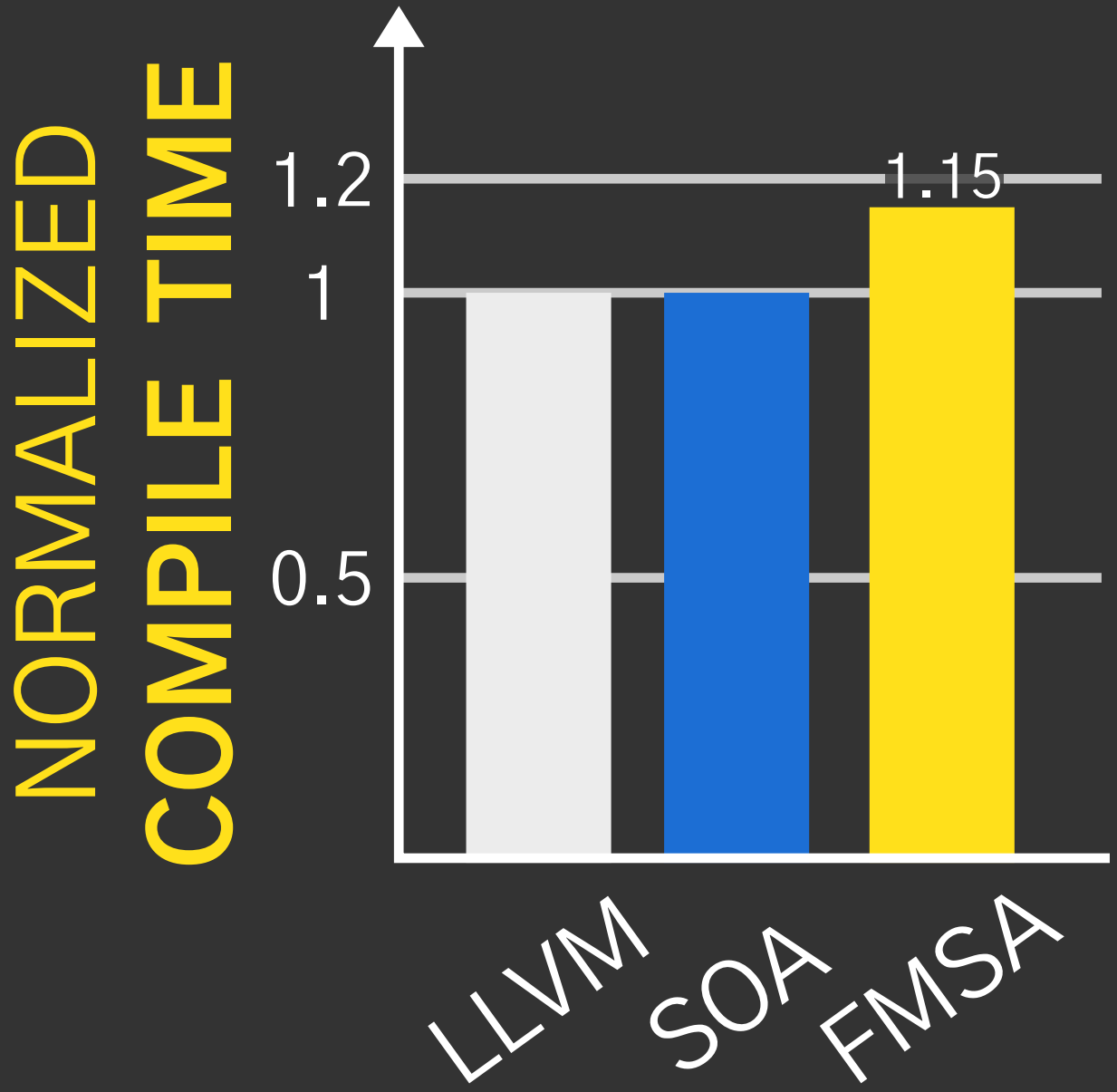
AVERAGE ON SPEC'06



AVERAGE ON SPEC'06



AVERAGE ON SPEC'06



CONCLUSION

NOVEL and **POWERFUL** Technique for Merging **ANY** Pair of Functions

Efficient Exploration Mechanism

LINKS

Paper (CGO'19):

https://doi.org/10.1109/CGO.2019.8661174

Source Code:

https://github.com/rcorcs/fmsa

LLVM Dev RFC:

https://groups.google.com/d/msg/llvm-dev/HQRg58ZCXsM/N7fbgvZgBQAJ

LLVM Patch:

https://reviews.llvm.org/D59442

Function Merging by Sequence Alignment

Rodrigo Rocha

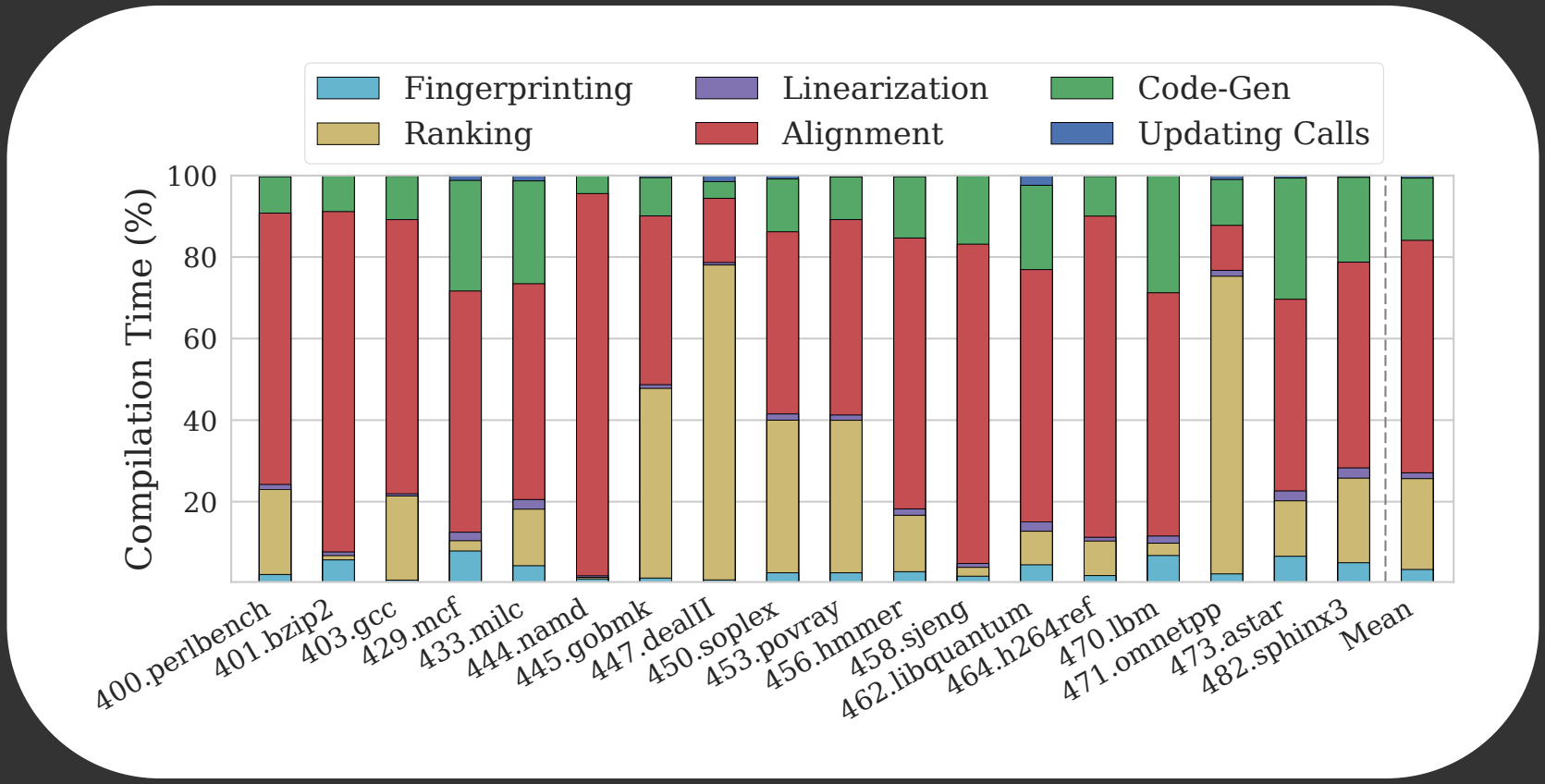
Pavlos Petoumenos

Zheng Wang

Murray Cole

Hugh Leather

Compilation Breakdown



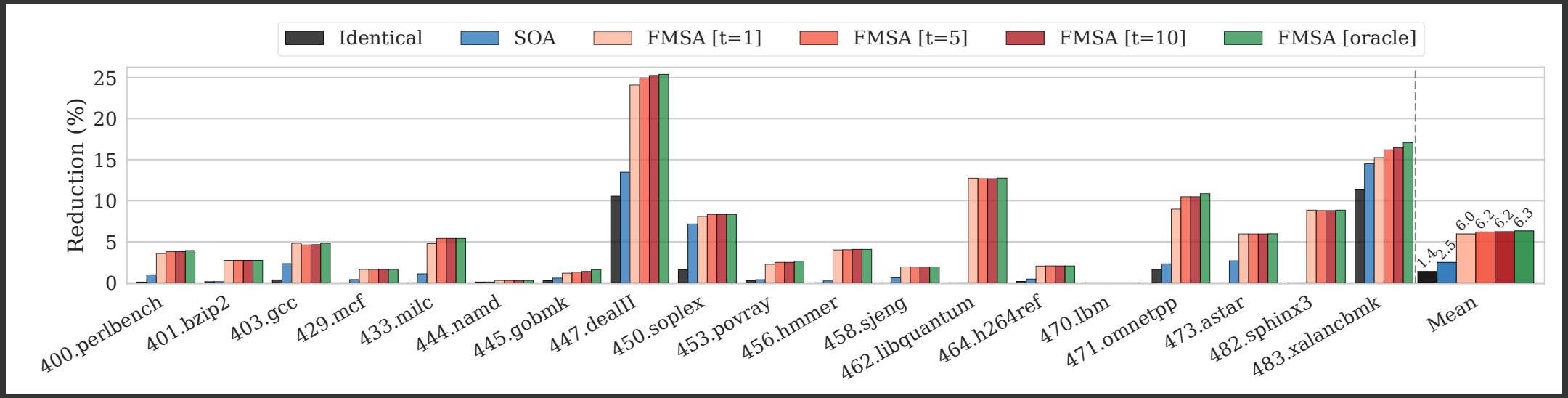
Future Work:

Use a greedy sequence alignment

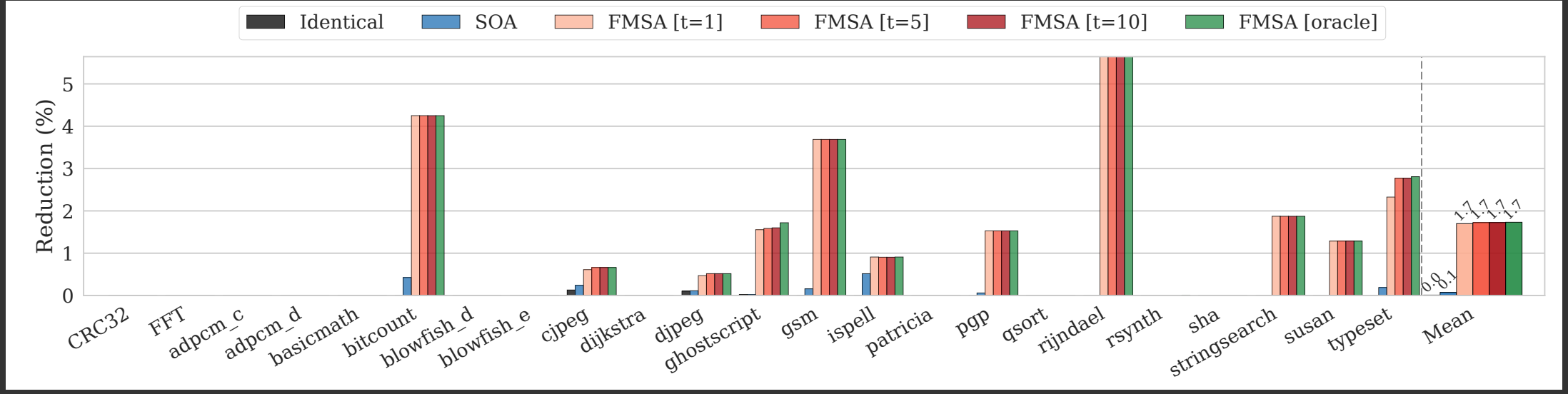
Avoid wasteful merges

Intel

SPEC 2006

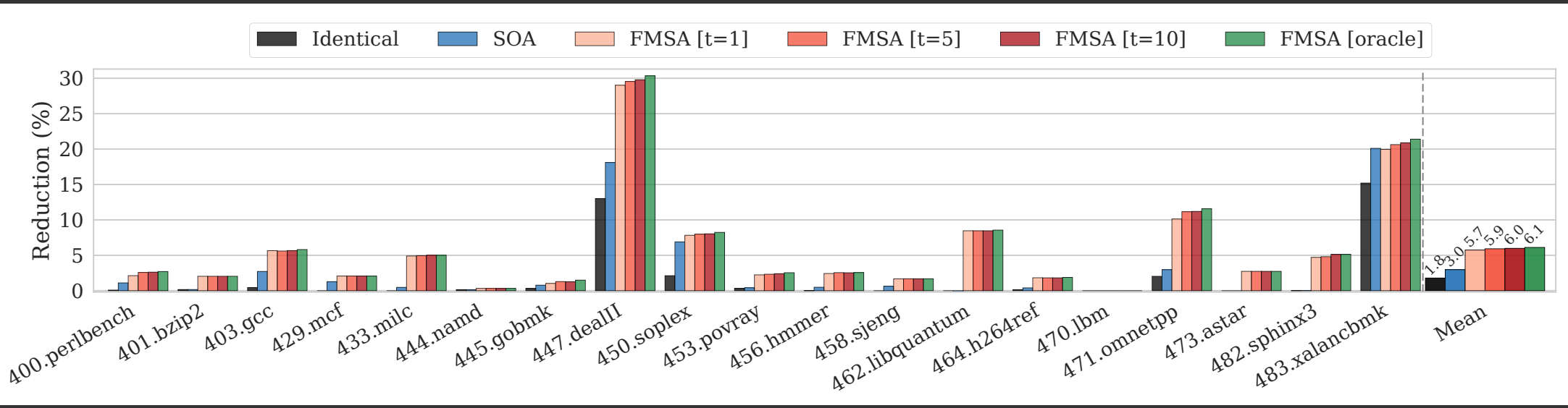


MiBench



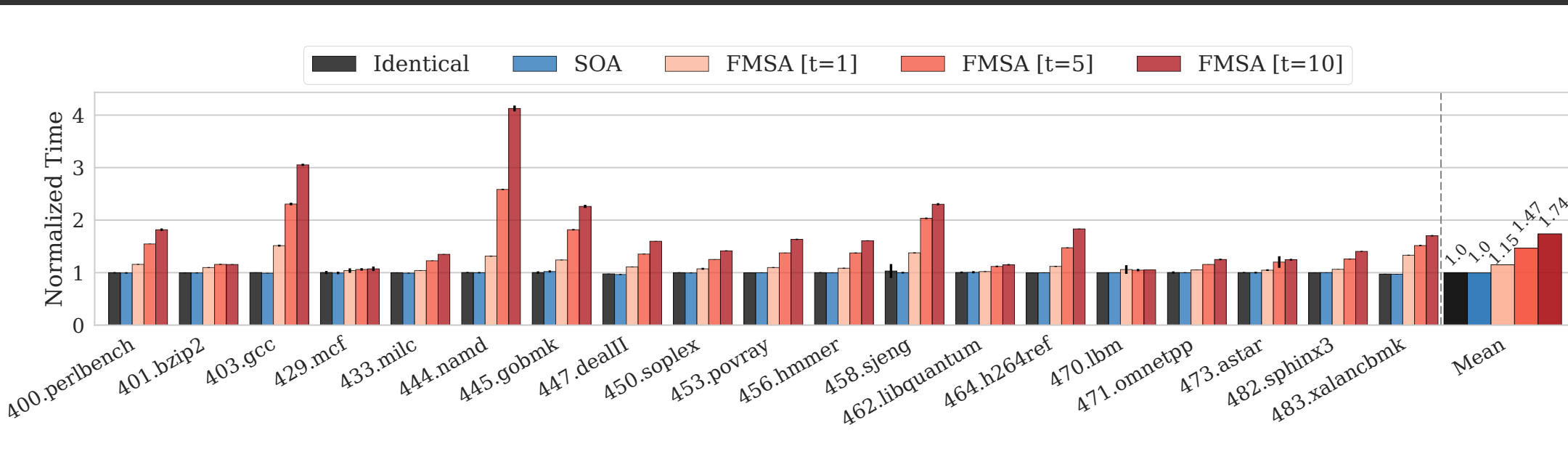
ARM

SPEC 2006



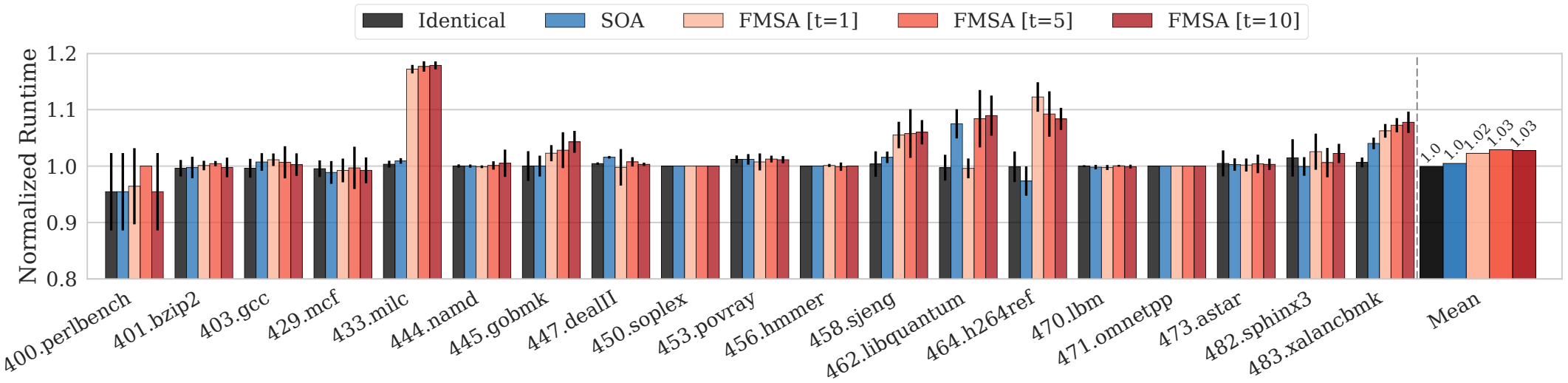
Compilation Time

SPEC 2006



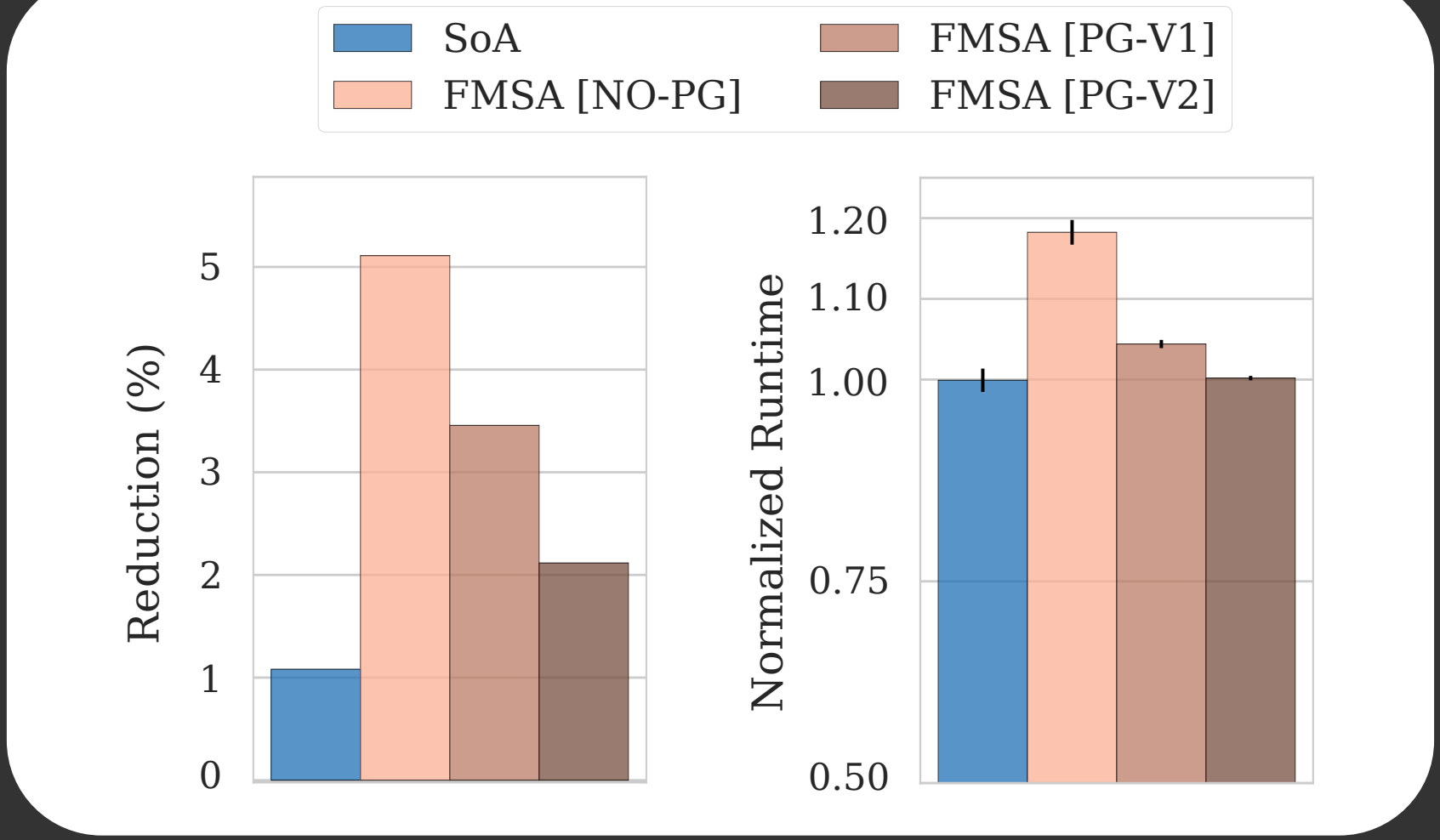
Runtime Overhead

SPEC 2006

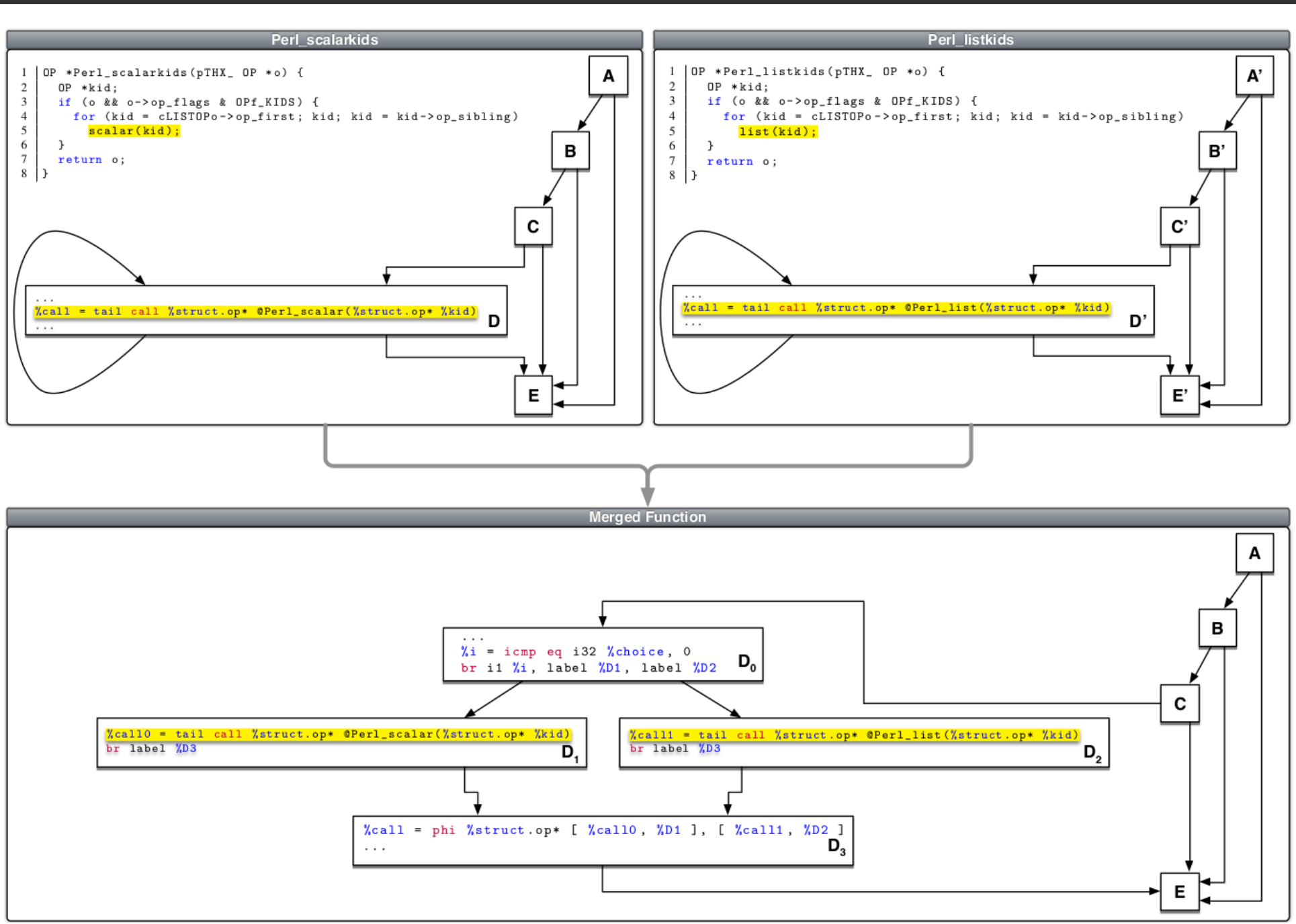


Profile-Guided Results

SPEC 2006: 433.milc



Example: State of the Art



Example of Identical Functions

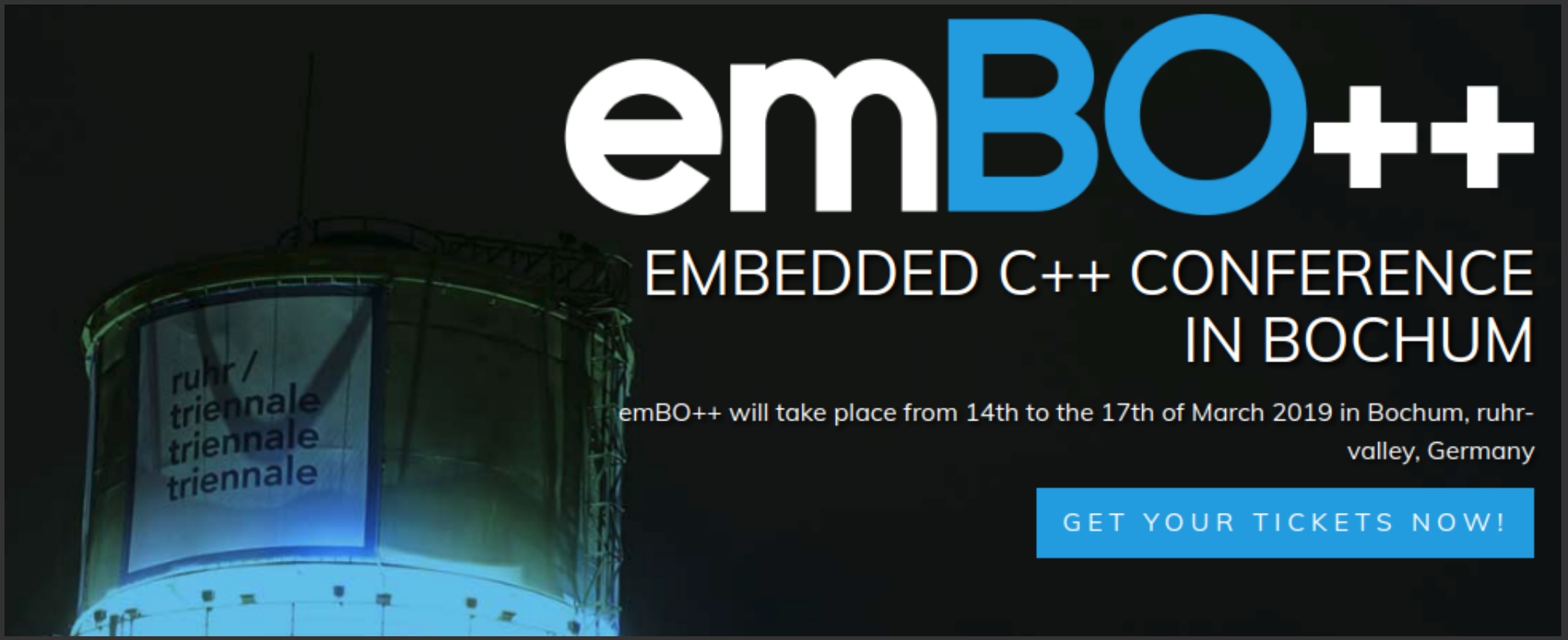
```
template <int dim>
unsigned int PolynomialSpace<dim>::
compute_n_pols (const unsigned int n) {
  unsigned int n_pols = n;
  for (unsigned int i=1; i<dim; ++i) {
 n  _pols *= (n+i);
 n  _pols /= (i+ 1);
 }
  return n_pols;
}
```

```
template <int dim> inline
unsigned int TensorProductPolynomials<dim>::
x_to_the_dim (const unsigned int x) {
  unsigned int y = 1;
  for (unsigned int d=0; d<dim; ++d) {
y    *= x;
}
  return y;
}
```

– – – – – – – – – – After template specialization and applying optimizations:– – – – – – – – – – –

```
unsigned int PolynomialSpace<1>::
compute_n_pols(const unsigned int n) {
  return n;
}
```

```
unsigned int TensorProductPolynomials<1>::
x_to_the_dim(const unsigned int x) {
  return x;
}
```


Follow
Did I mention our embedded engineers are using C++17?
Are you an Embedded Software Engineer looking to drive forwards? Our West of Ireland software hub is looking for a Senior Engineer to help create the future of mobility. #virtualisation #embeddedsoftware #SoftwareDevelopment …
6:00 AM - 1 Feb 2019
emBO++
EMBEDDED C++ CONFERENCE IN BOCHUM
emBO++ will take place from 14th to the 17th of March 2019 in Bochum, ruhr-valley, Germany
GET YOUR TICKETS NOW!
ruhr / triennale triennale triennale

Compilation Pipeline