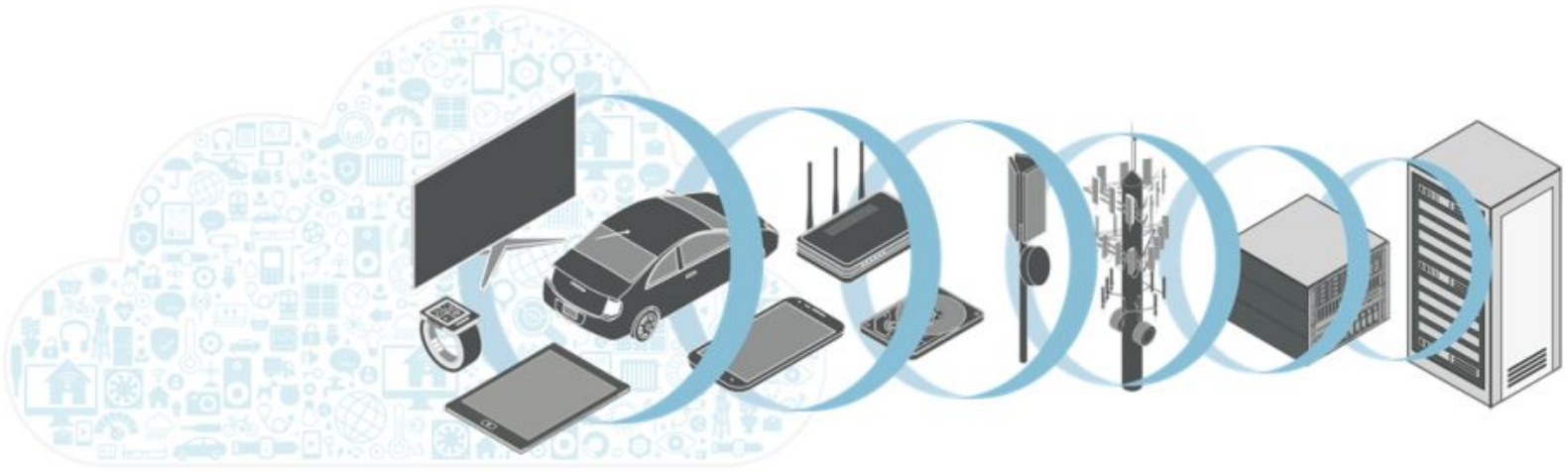
Function Merging for Free

Rodrigo Rocha

P. Petoumenos, Z. Wang, M. Cole, K. Hazelwood, H. Leather

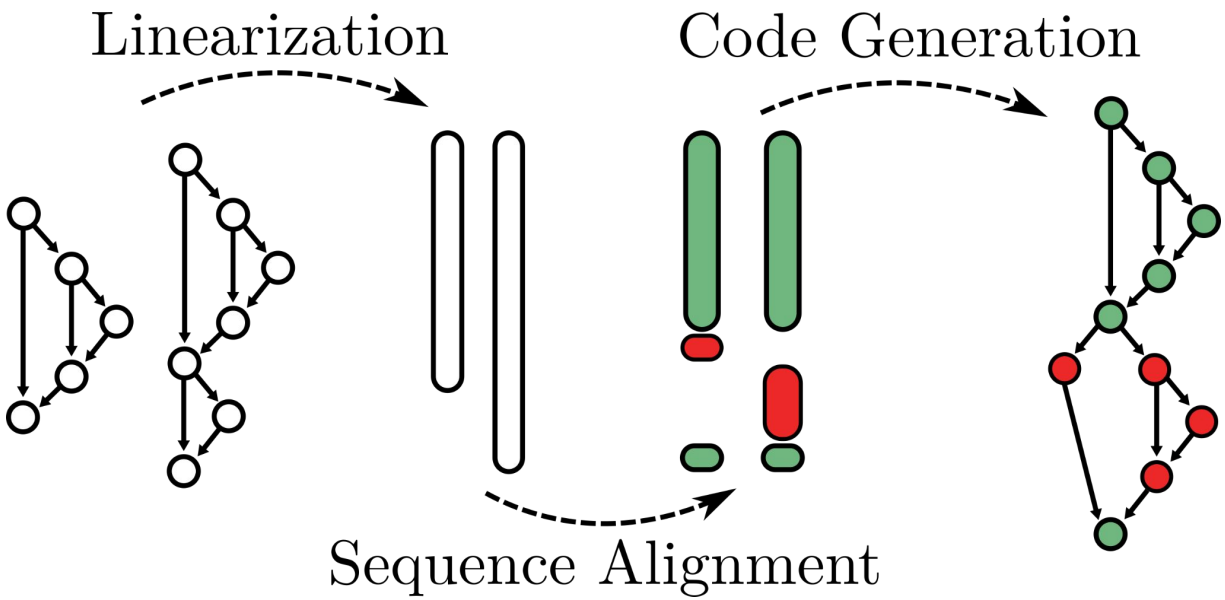
Introduction

Code size is important in many domains -- Large is relative to the constraints



Introduction

Prior function merging technique based on sequence alignment

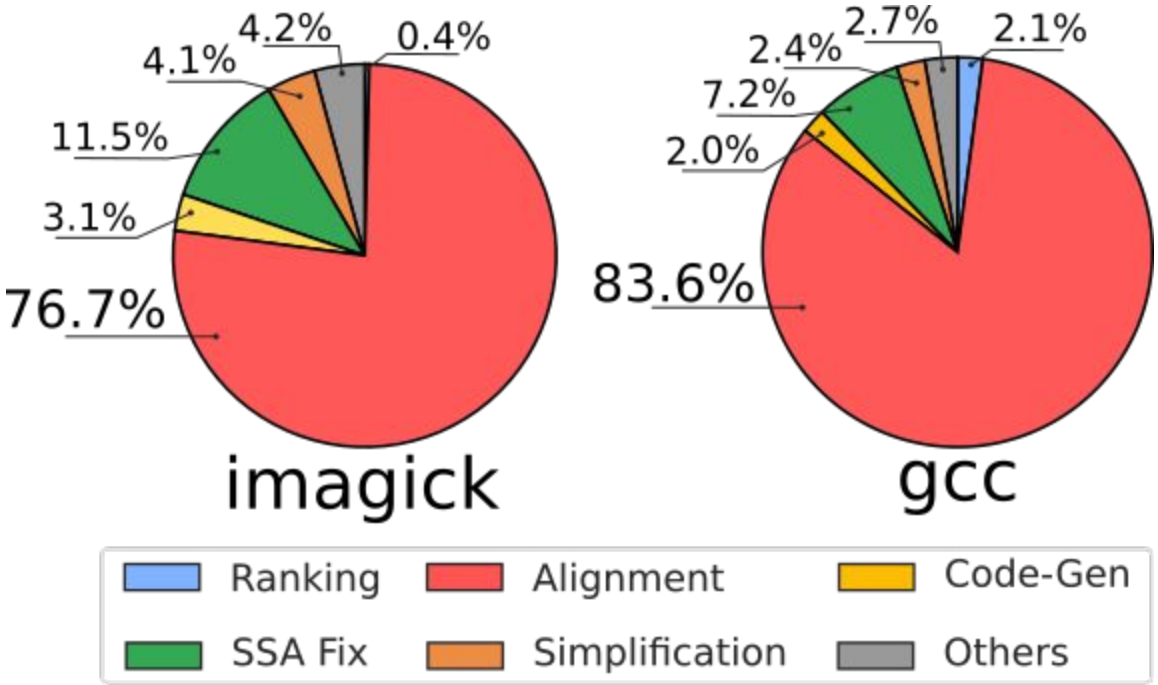


Excessive Memory Usage

The State of the Art can use up to 32 GB of memory to merge two functions

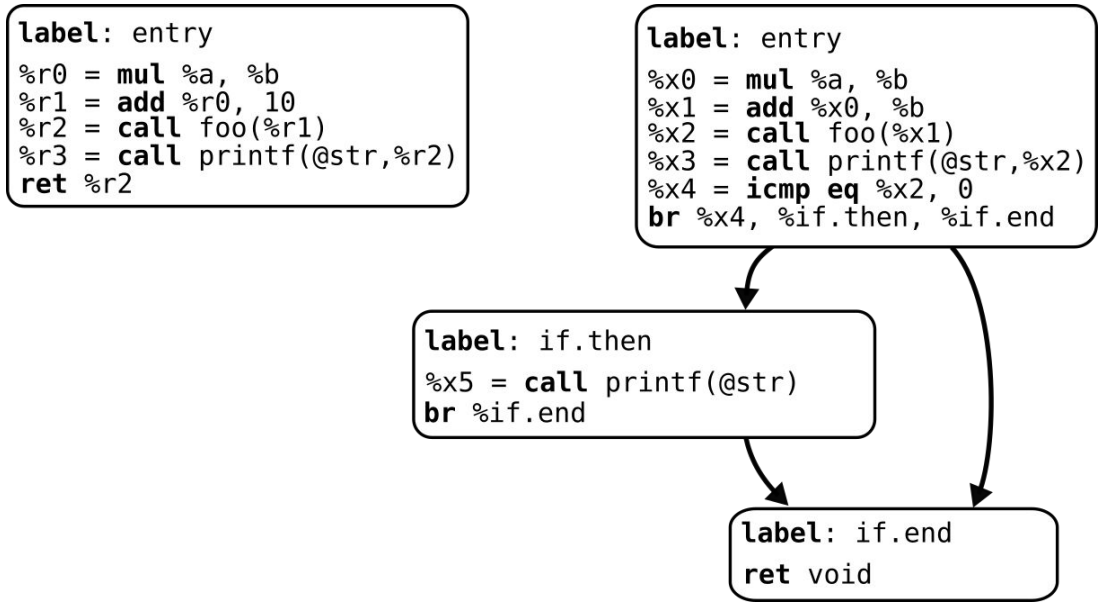
Breakdown of the Compilation Time

Sequence alignment is to blame



State of the Art

Given two input functions



State of the Art

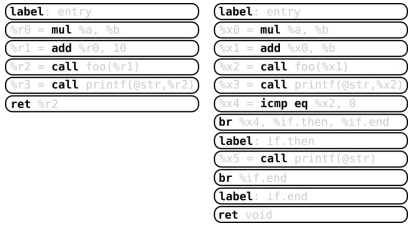
Linearize the input functions

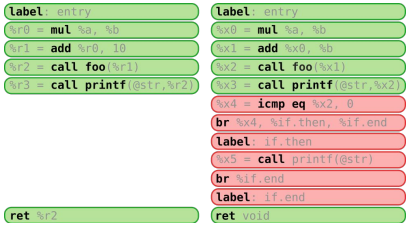
```
label: entry
%r0 = mul %a, %b
%r1 = add %r0, 10
%r2 = call foo(%r1)
%r3 = call printf(@str,%r2)
ret %r2
```

```
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
```

State of the Art

Align the input functions using a quadratic algorithm

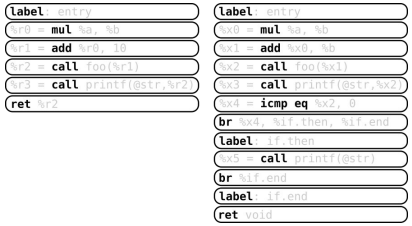




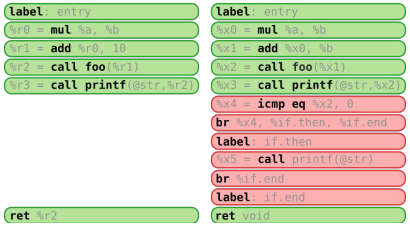
	label	mul	add	call foo	call printf	icmp eq	br	label	call	br	label	ret
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
label	-1	**2**	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
mul	-2	1	**4**	3	2	1	0	-1	-2	-3	-4	-5	-6
add	-3	0	3	**6**	5	4	3	2	1	0	-1	-2	-3
call foo	-4	-1	2	5	**8**	7	6	5	4	3	2	1	0
call printf	-5	-2	1	4	7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

State of the Art

Align the input functions using a quadratic algorithm



	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**



gcc

Largest Functions

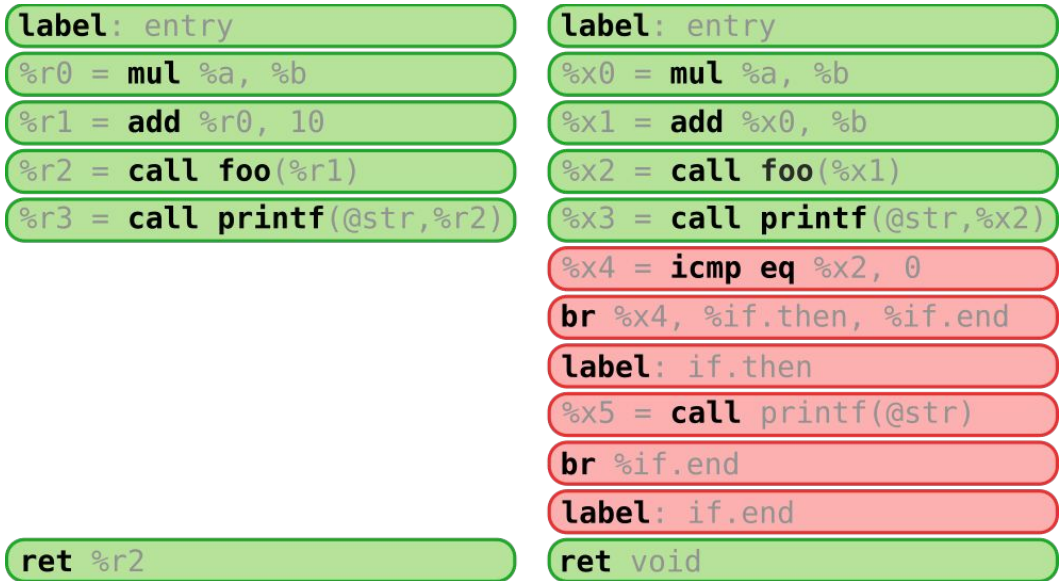
90093

76265

~ 32 GB

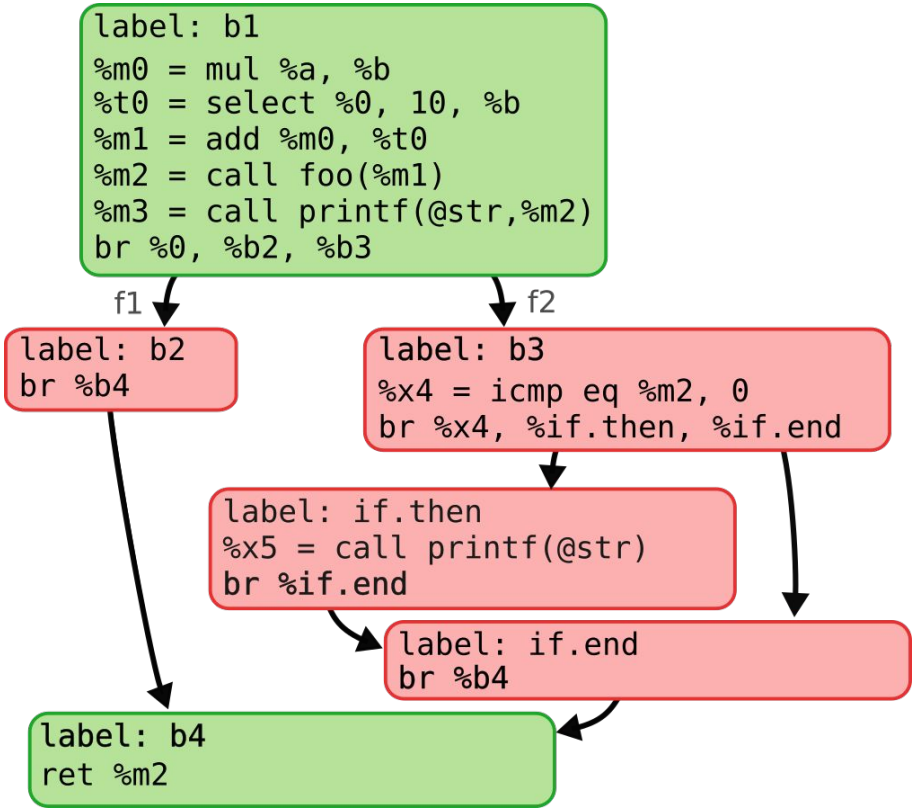
State of the Art

The aligned input functions

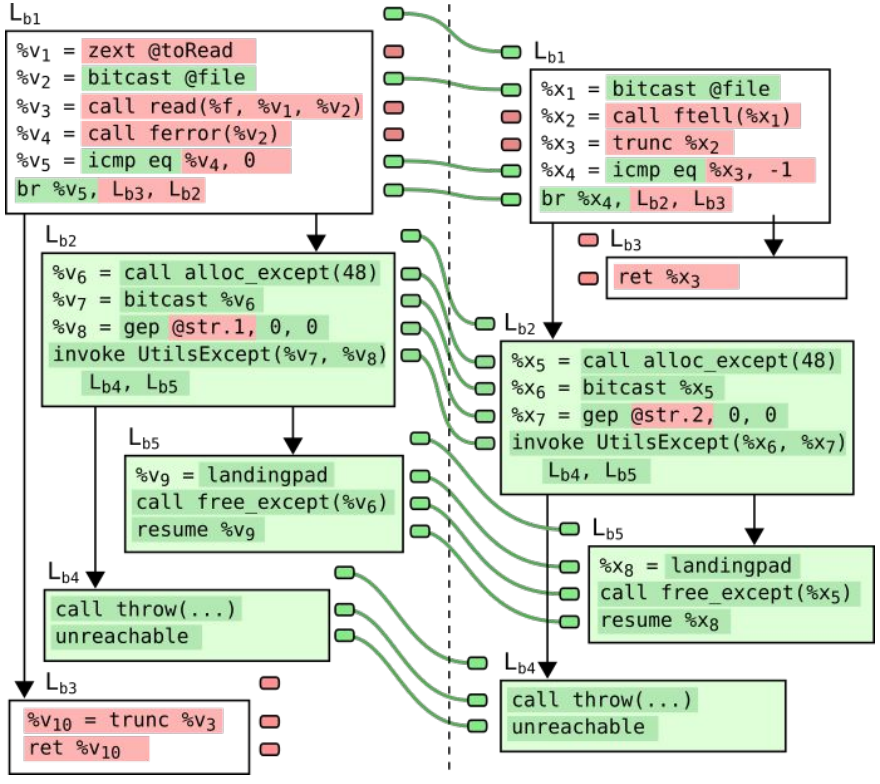


State of the Art

Merged function



Merging Only Highly Similar Blocks



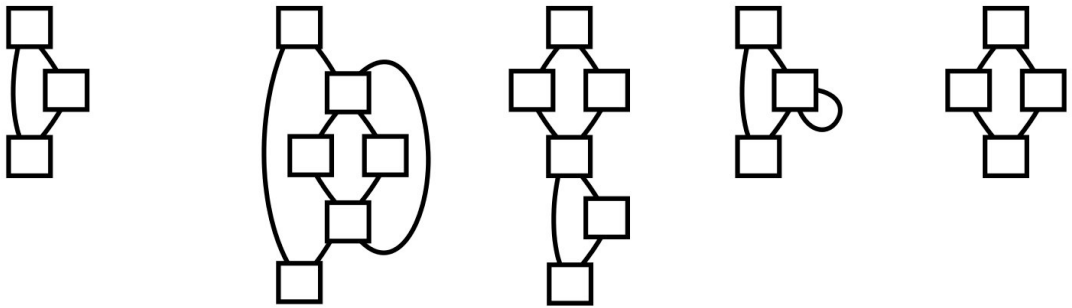
HyFM: Hybrid Function Merging

- Fingerprint-based exploration

- Block-focused merging technique

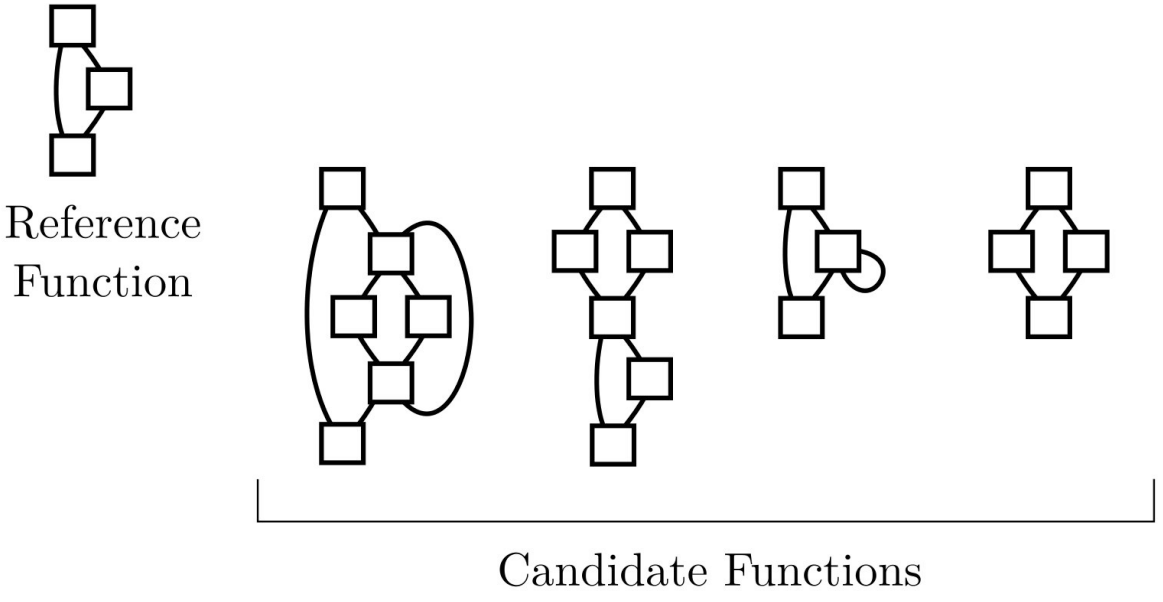
Pairing Functions

Choosing which functions to merge



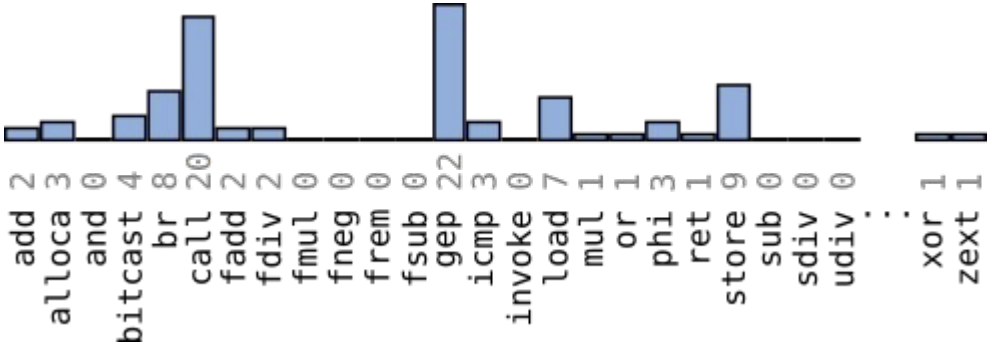
Pairing Functions

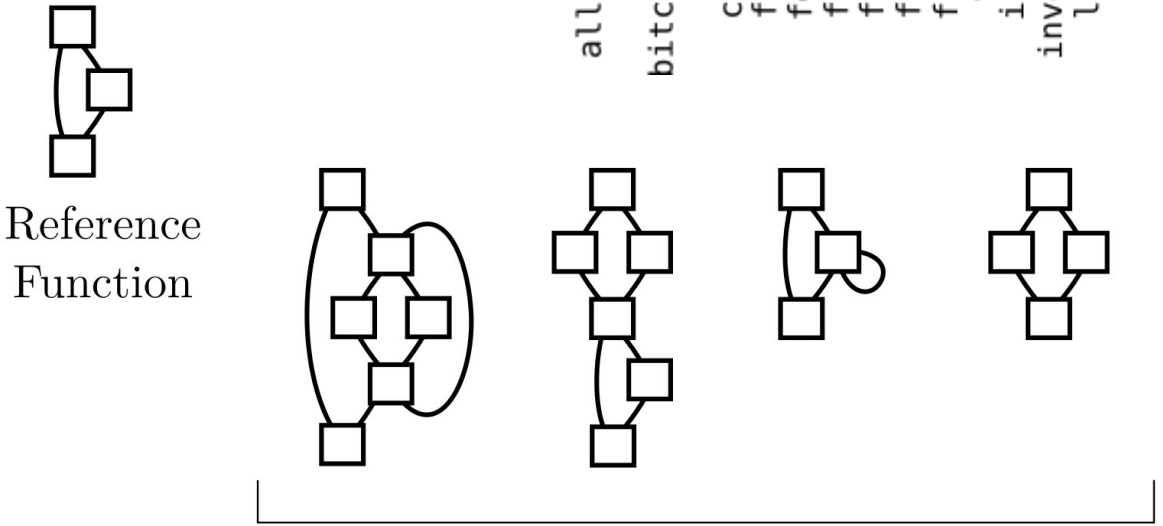
Choosing which functions to merge



Pairing Functions

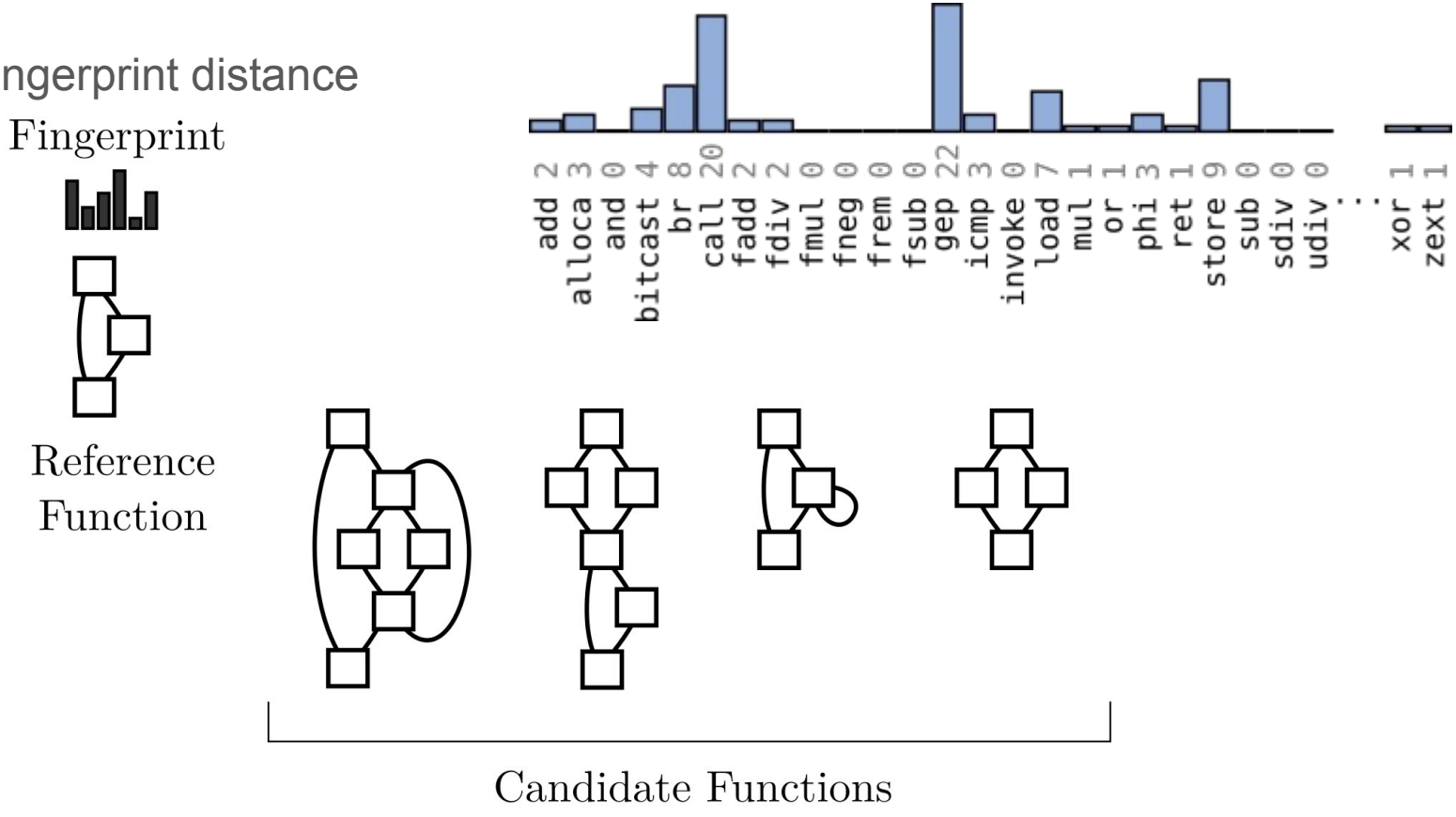
Minimize fingerprint distance





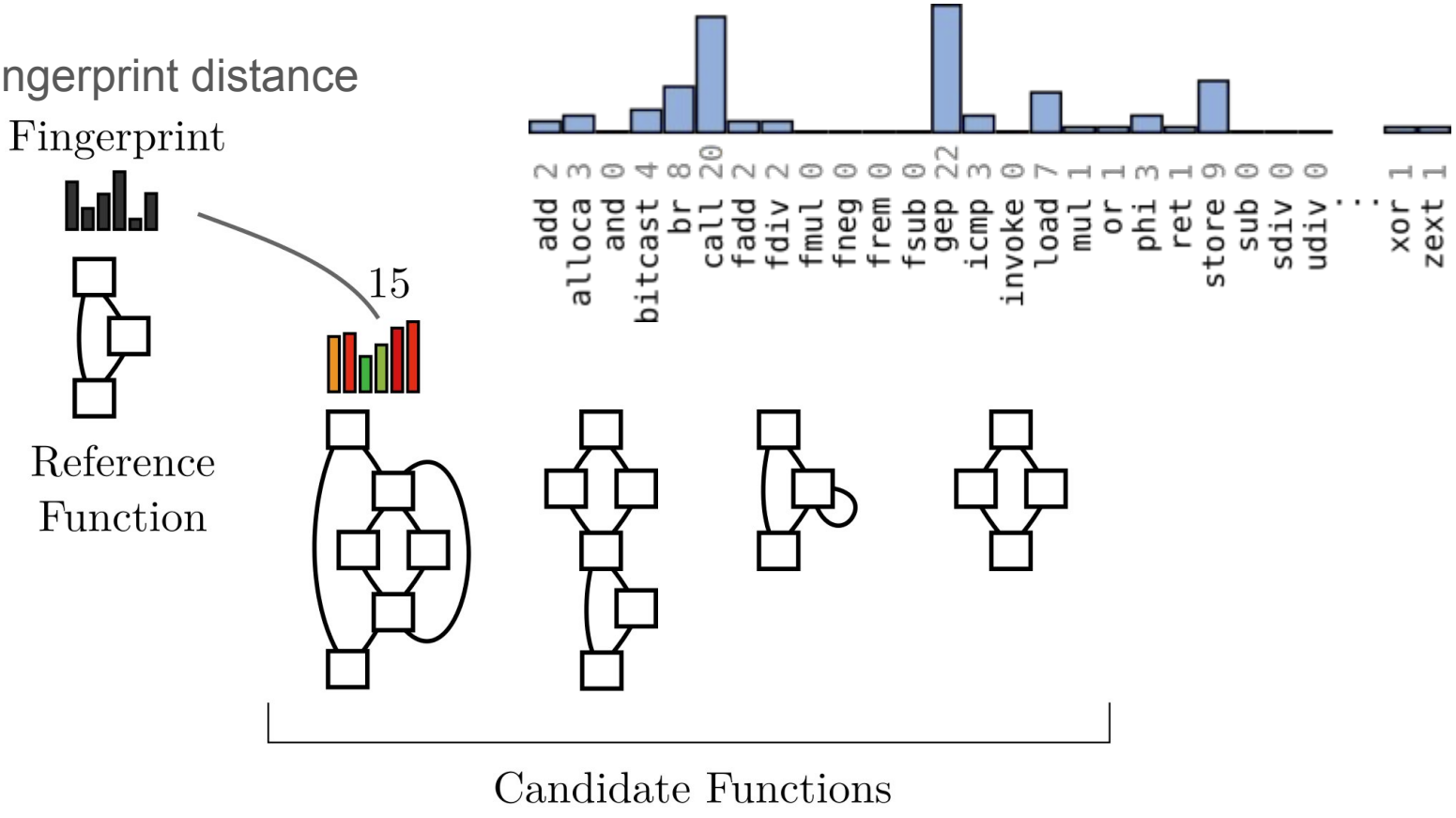
Pairing Functions

Minimize fingerprint distance



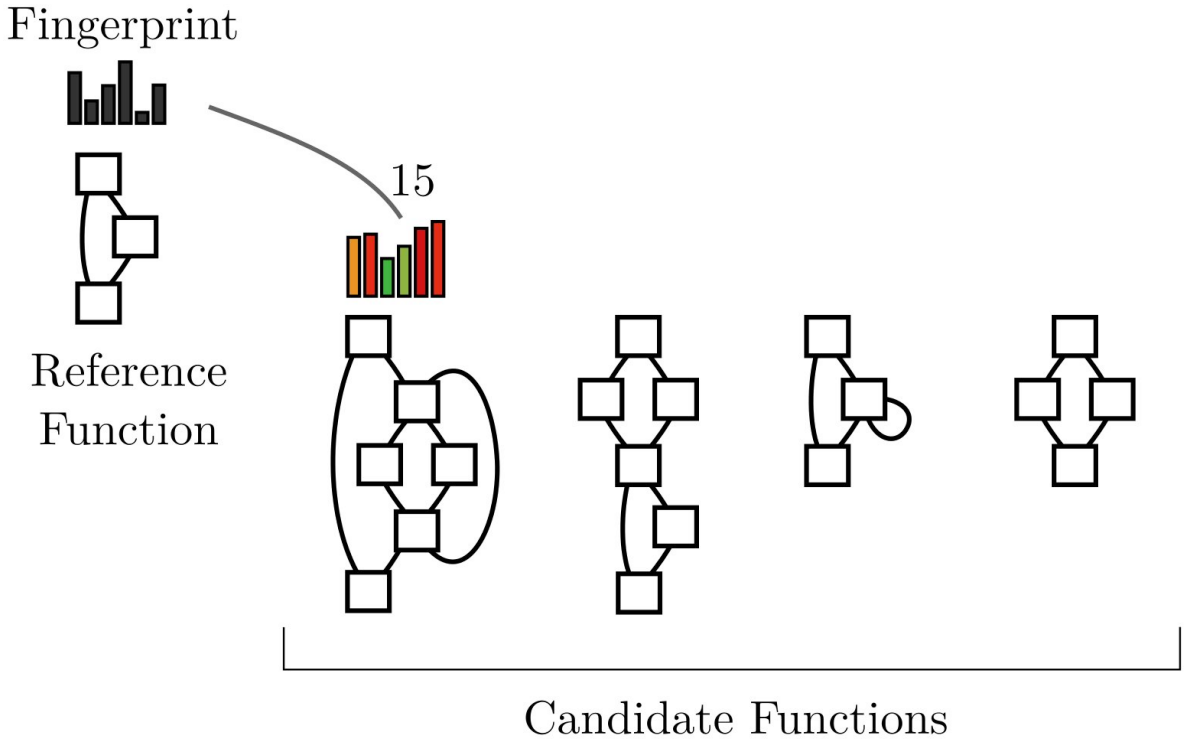
Pairing Functions

Minimize fingerprint distance



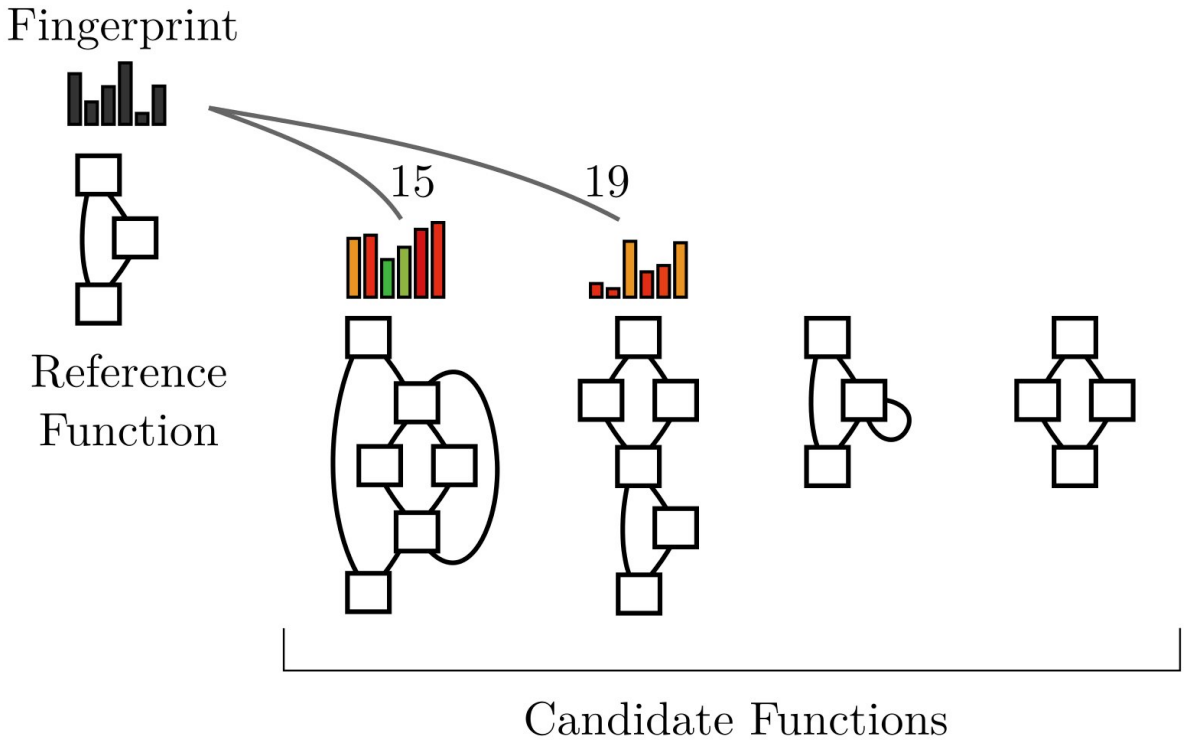
Pairing Functions

Minimize fingerprint distance



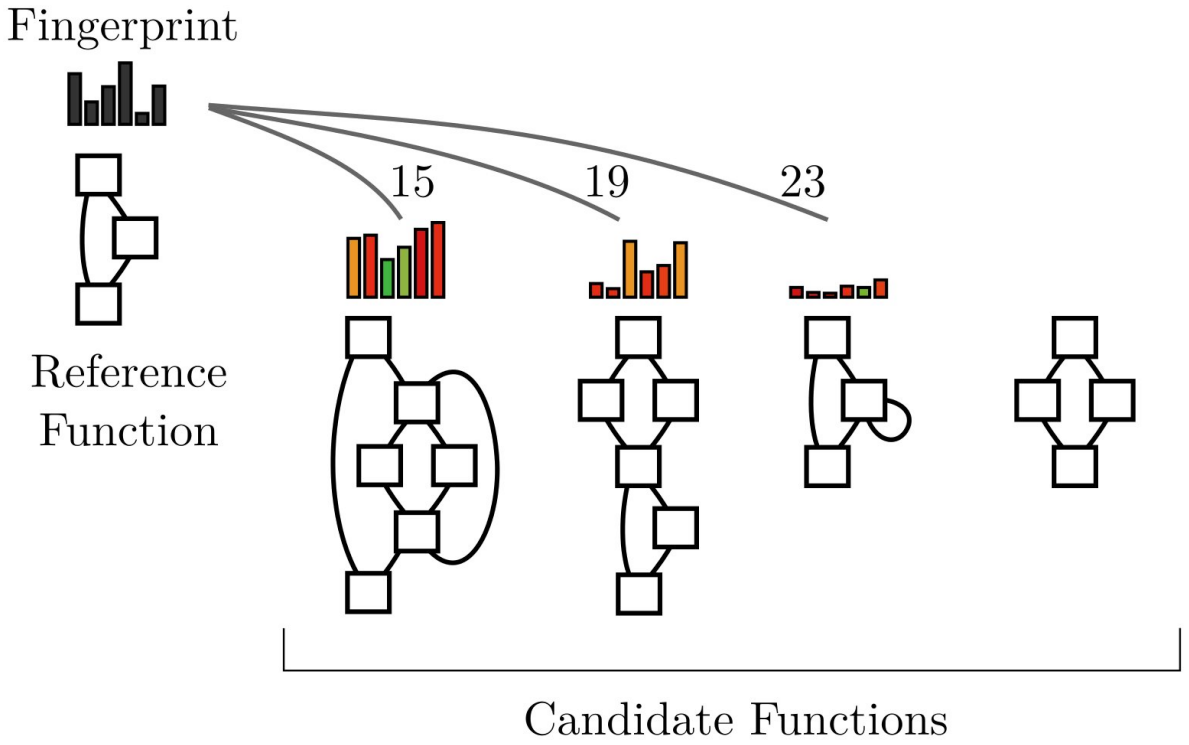
Pairing Functions

Minimize fingerprint distance



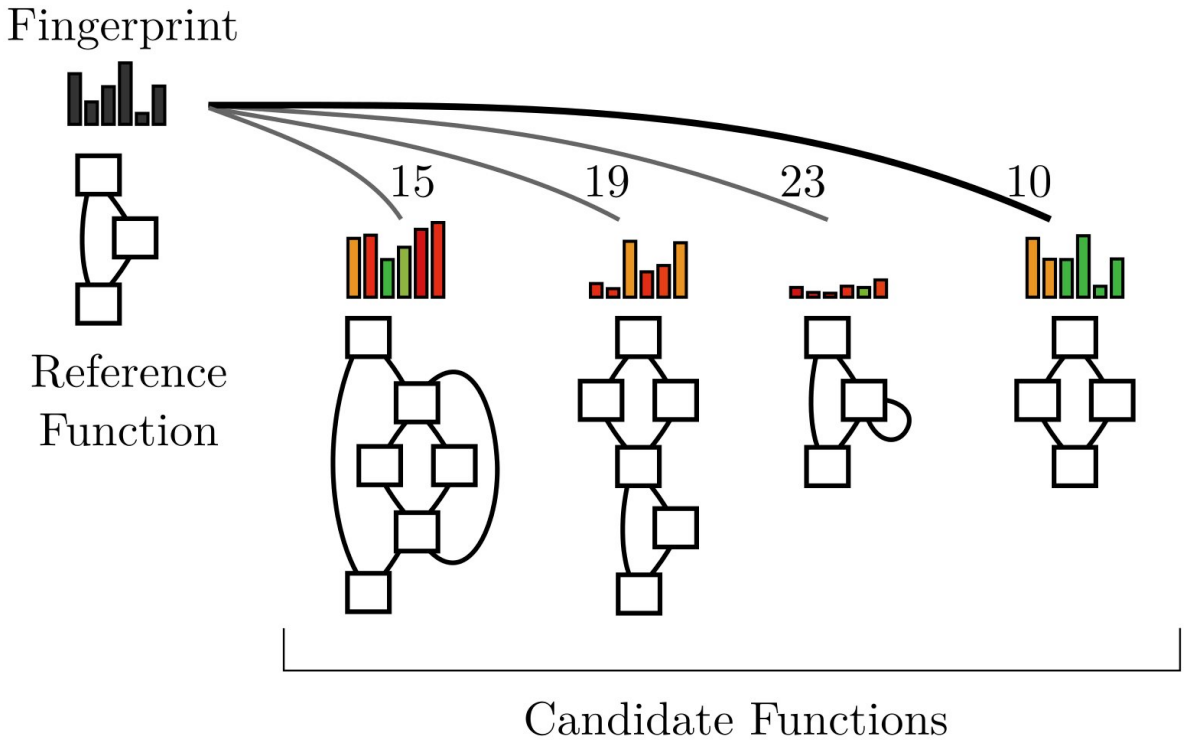
Pairing Functions

Minimize fingerprint distance



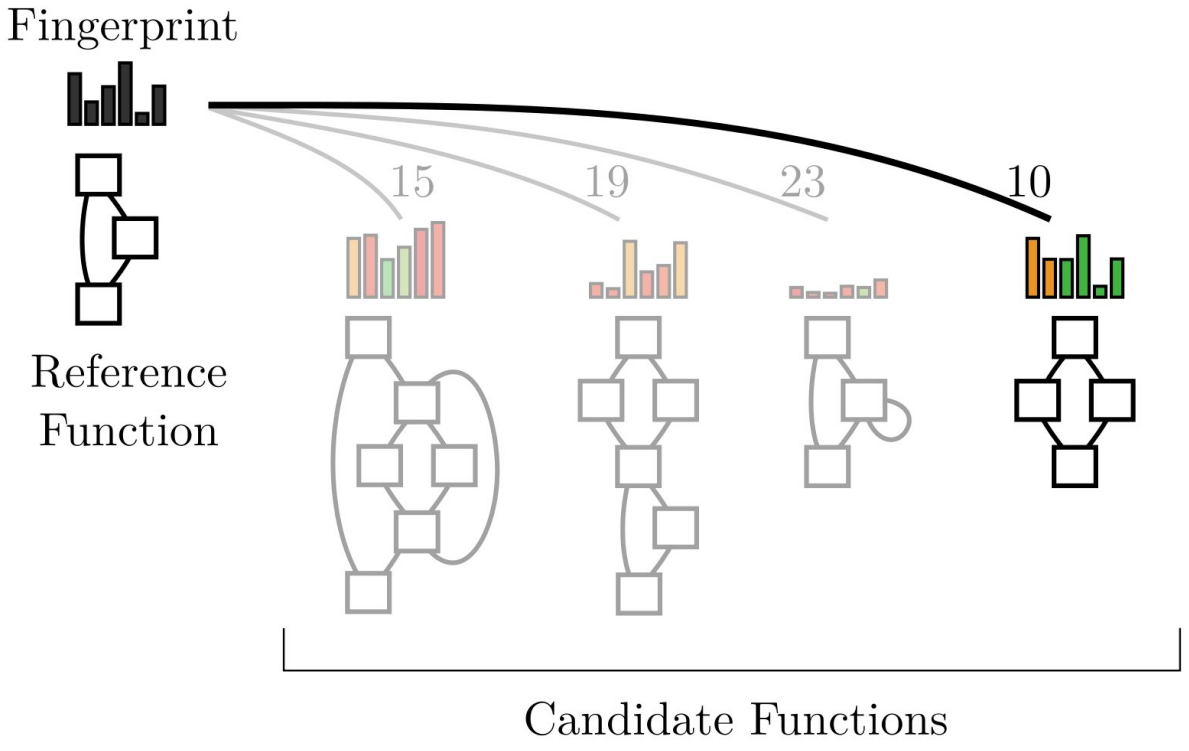
Pairing Functions

Minimize fingerprint distance



Pairing Functions

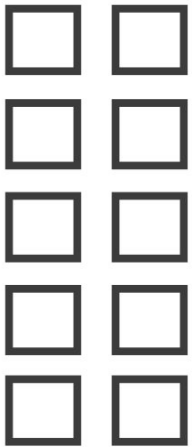
Minimize fingerprint distance



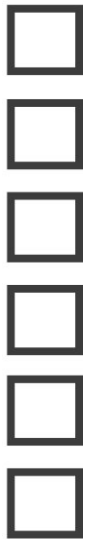
Pairing Blocks

Given two functions

Function 1



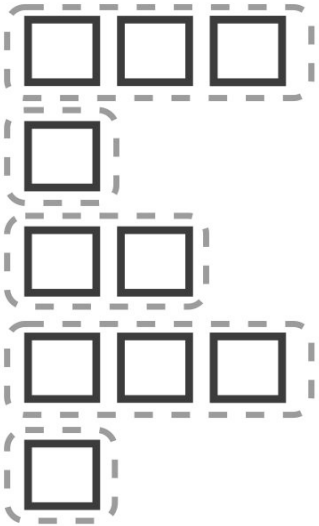
Function 2



Pairing Blocks

Group blocks by their size

Function 1

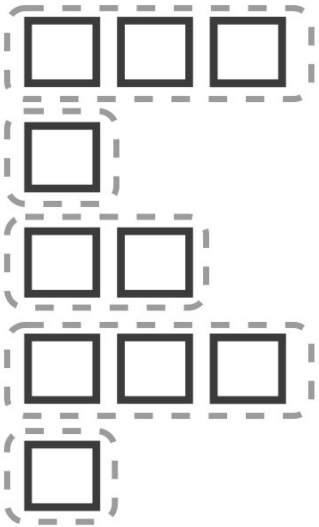


Function 2

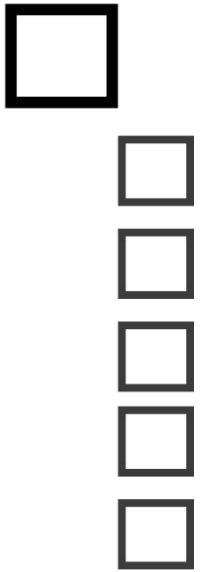
Pairing Blocks

Minimize fingerprint distance

Function 1

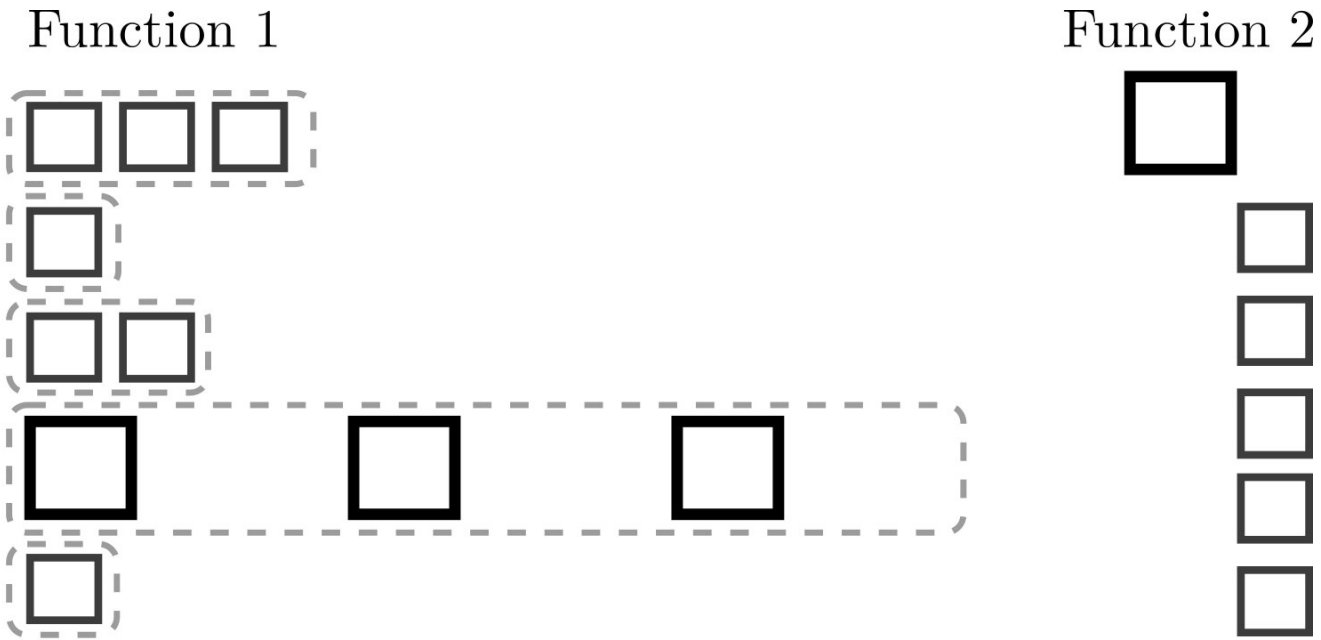


Function 2



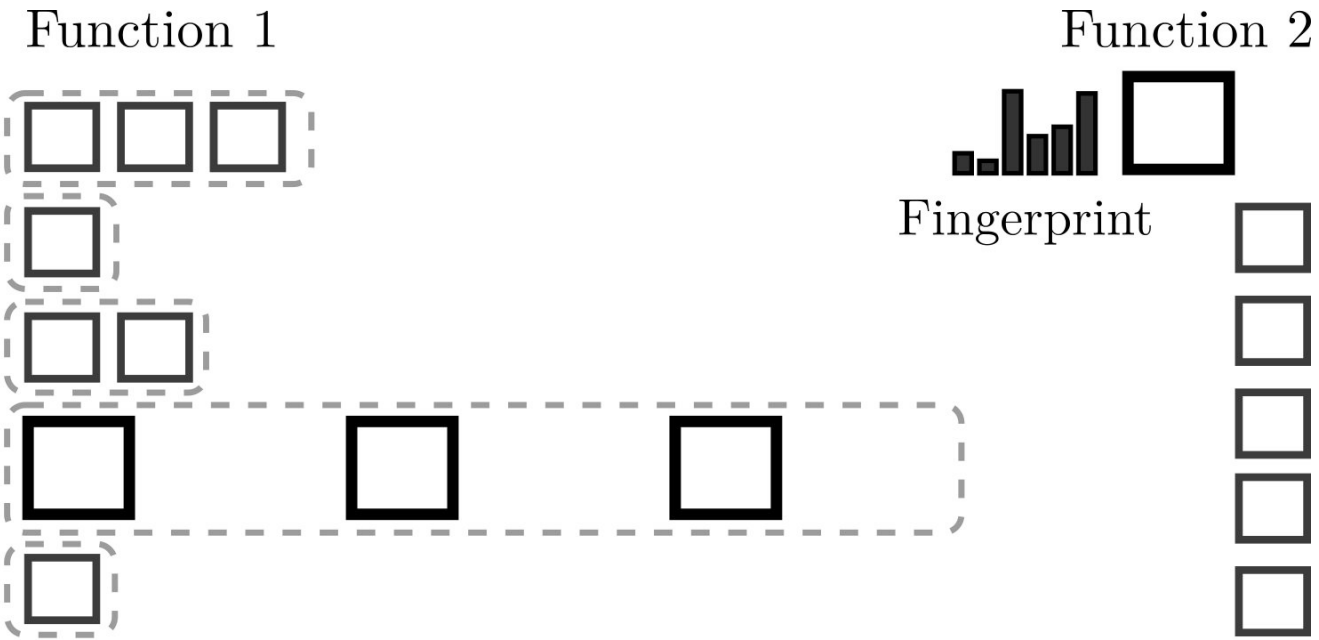
Pairing Blocks

Minimize fingerprint distance



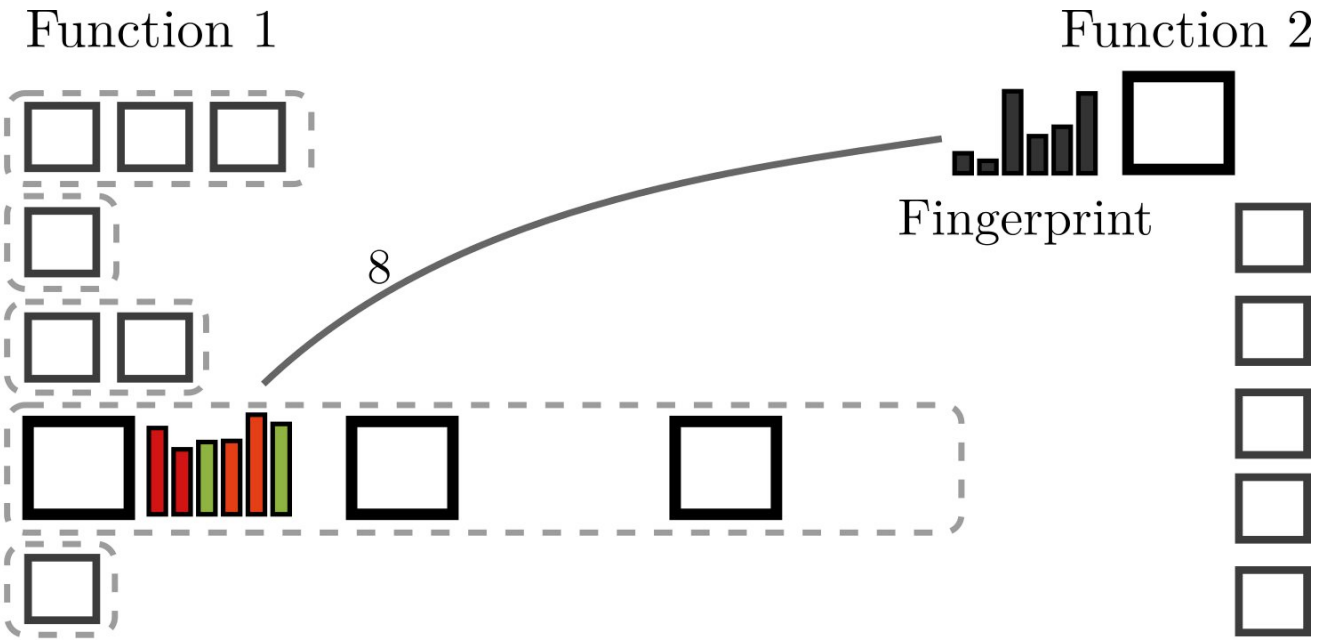
Pairing Blocks

Minimize fingerprint distance



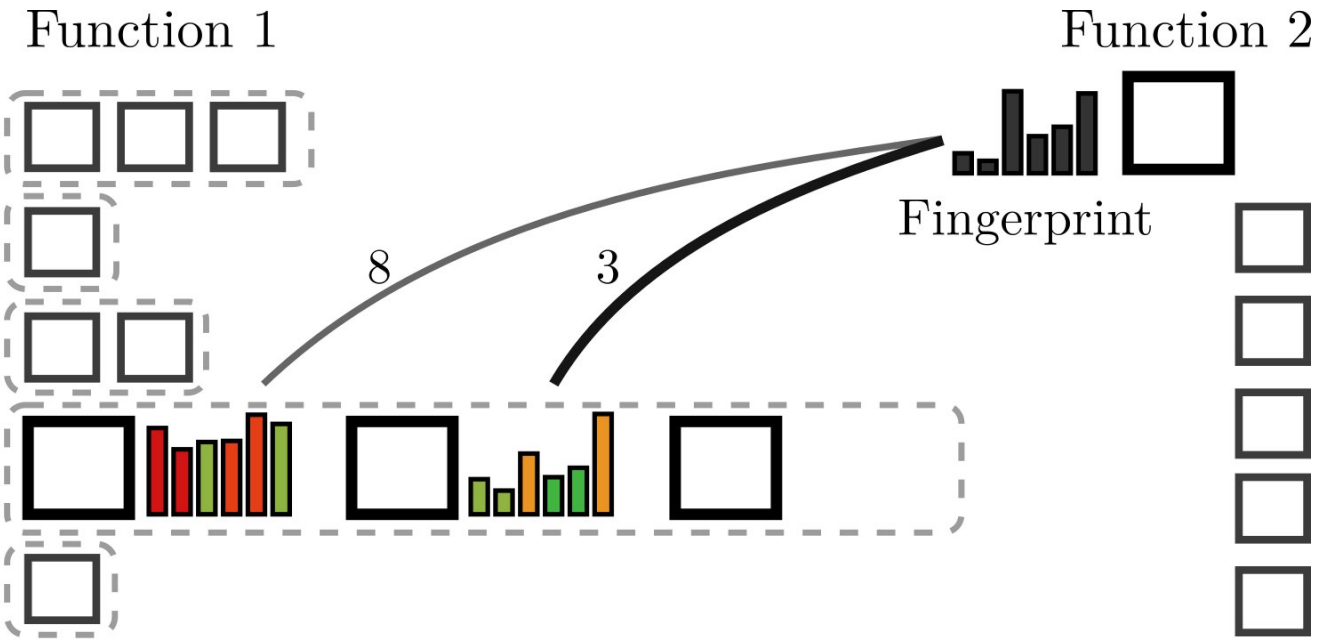
Pairing Blocks

Minimize fingerprint distance



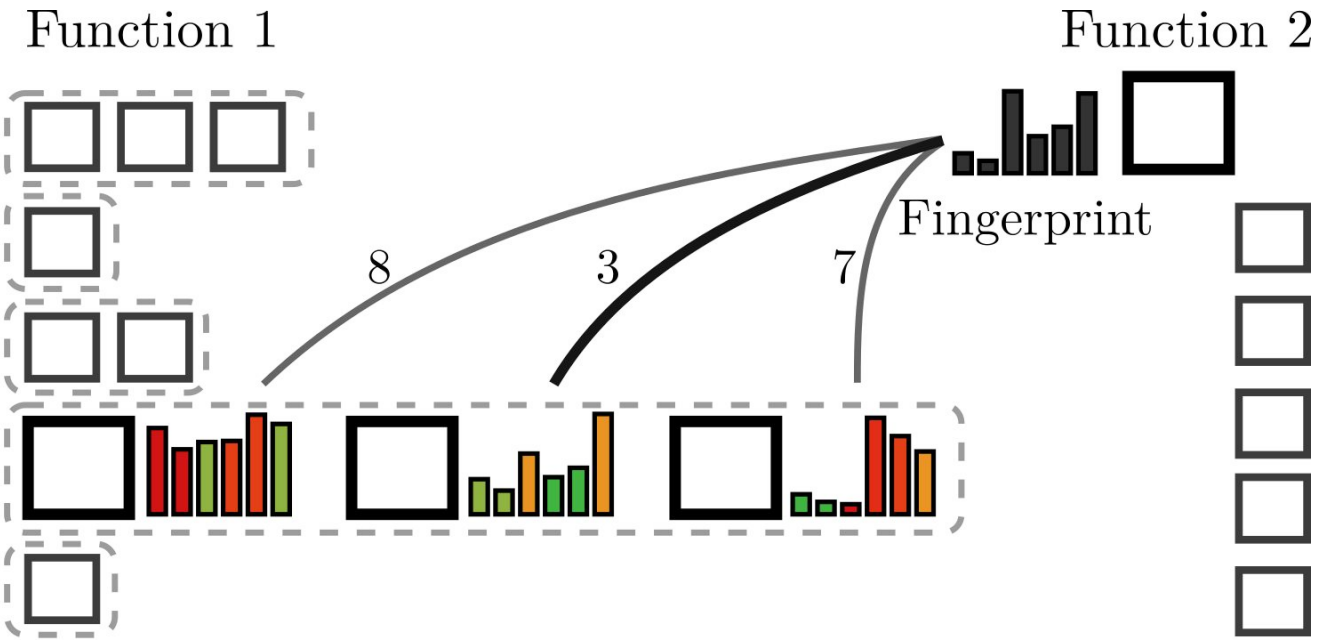
Pairing Blocks

Minimize fingerprint distance



Pairing Blocks

Minimize fingerprint distance



Pairwise Alignment

Match only corresponding instructions

%sw.bb	%entry
%v1 = gep %this, 0, 5	%x1 = alloca
%v2 = bitcast %v1	%x2 = gep %this, 0, 1
$%v_3$ = load %v2	$%x_3$ = load %x2
$%v_4$ = icmp eq %v3, 0	$%x_4$ = icmp eq %x3, 73
br $%v_4$, L_{b3}, L_{b2}	br $%x_4$, L_{b3}, L_{b2}

Pairwise Alignment

Match only corresponding instructions

%sw.bb	%entry
%v1 = gep %this, 0, 5	%x1 = alloca
%v2 = bitcast %v1	%x2 = gep %this, 0, 1
$\%v_3$ = load %v2	$\%x_3$ = load %x2
$\%v_4$ = icmp eq %v3, 0	$\%x_4$ = icmp eq %x3, 73
br $\%v_4$, L_{b3}, L_{b2}	br $\%x_4$, L_{b3}, L_{b2}

Pairwise Alignment

Match only corresponding instructions

%sw.bb	%entry
%v1 = gep %this, 0, 5	%x1 = alloca
%v2 = bitcast %v1	%x2 = gep %this, 0, 1
$\%v_3$ = load %v2	$\%x_3$ = load %x2
$\%v_4$ = icmp eq %v3, 0	$\%x_4$ = icmp eq %x3, 73
br $\%v_4$, L_{b3}, L_{b2}	br $\%x_4$, L_{b3}, L_{b2}

Pairwise Alignment

Match only corresponding instructions

%sw.bb	%entry
%v1 = gep %this, 0, 5	%x1 = alloca
%v2 = bitcast %v1	%x2 = gep %this, 0, 1
$\%v_3$ = load %v2	$\%x_3$ = load %x2
$\%v_4$ = icmp eq %v3, 0	$\%x_4$ = icmp eq %x3, 73
br $\%v_4$, L_{b3}, L_{b2}	br $\%x_4$, L_{b3}, L_{b2}

Pairwise Alignment

Match only corresponding instructions

%sw.bb	%entry
%v1 = gep %this, 0, 5	%x1 = alloca
%v2 = bitcast %v1	%x2 = gep %this, 0, 1
$%v_3$ = load %v2	$%x_3$ = load %x2
$%v_4$ = icmp eq %v3, 0	$%x_4$ = icmp eq %x3, 73
br $%v_4$, L_{b3}, L_{b2}	br $%x_4$, L_{b3}, L_{b2}

Pairwise Alignment

Match only corresponding instructions

%sw.bb	%entry
%v1 = gep %this, 0, 5	%x1 = alloca
%v2 = bitcast %v1	%x2 = gep %this, 0, 1
$\%v_3$ = load %v2	$\%x_3$ = load %x2
$\%v_4$ = icmp eq %v3, 0	$\%x_4$ = icmp eq %x3, 73
br $\%v_4$, L_{b3}, L_{b2}	br $\%x_4$, L_{b3}, L_{b2}

Profitability Analysis

Estimate size of merged block

%sw.bb	%entry	0
%v1 = gep %this, 0, 5	%x1 = alloca	
%v2 = bitcast %v1	%x2 = gep %this, 0, 1	
$\%v_3$ = load %v2	$\%x_3$ = load %x2	
$\%v_4$ = icmp eq %v3, 0	$\%x_4$ = icmp eq %x3, 73	
br $\%v_4$, L_{b3}, L_{b2}	br $\%x_4$, L_{b3}, L_{b2}	

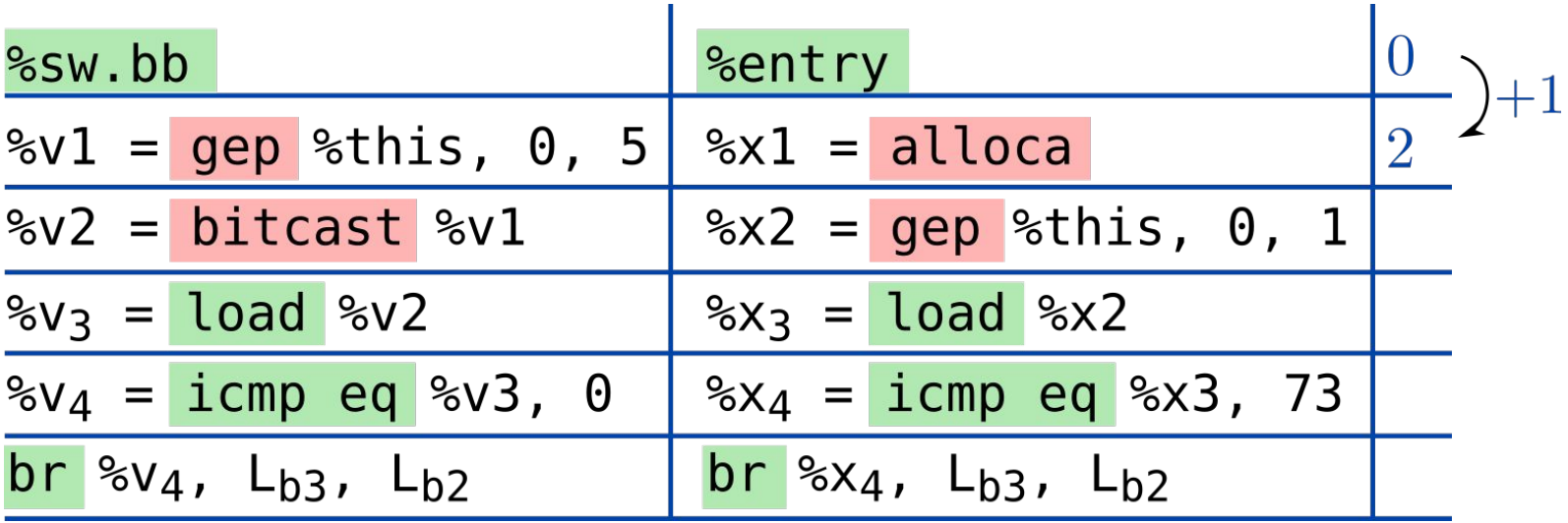
Profitability Analysis

Estimate size of merged block

%sw.bb	%entry	0
%v1 = gep %this, 0, 5	%x1 = alloca	2
%v2 = bitcast %v1	%x2 = gep %this, 0, 1	
$\%v_3$ = load %v2	$\%x_3$ = load %x2	
$\%v_4$ = icmp eq %v3, 0	$\%x_4$ = icmp eq %x3, 73	
br $\%v_4$, L_{b3}, L_{b2}	br $\%x_4$, L_{b3}, L_{b2}	

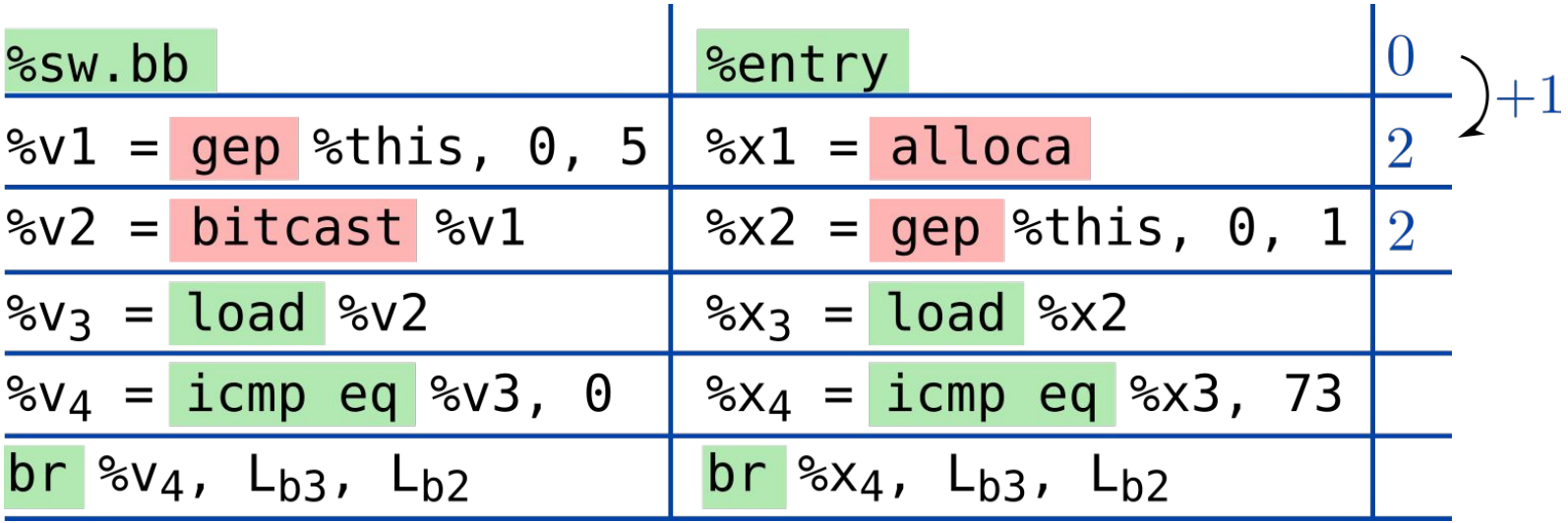
Profitability Analysis

Estimate size of merged block



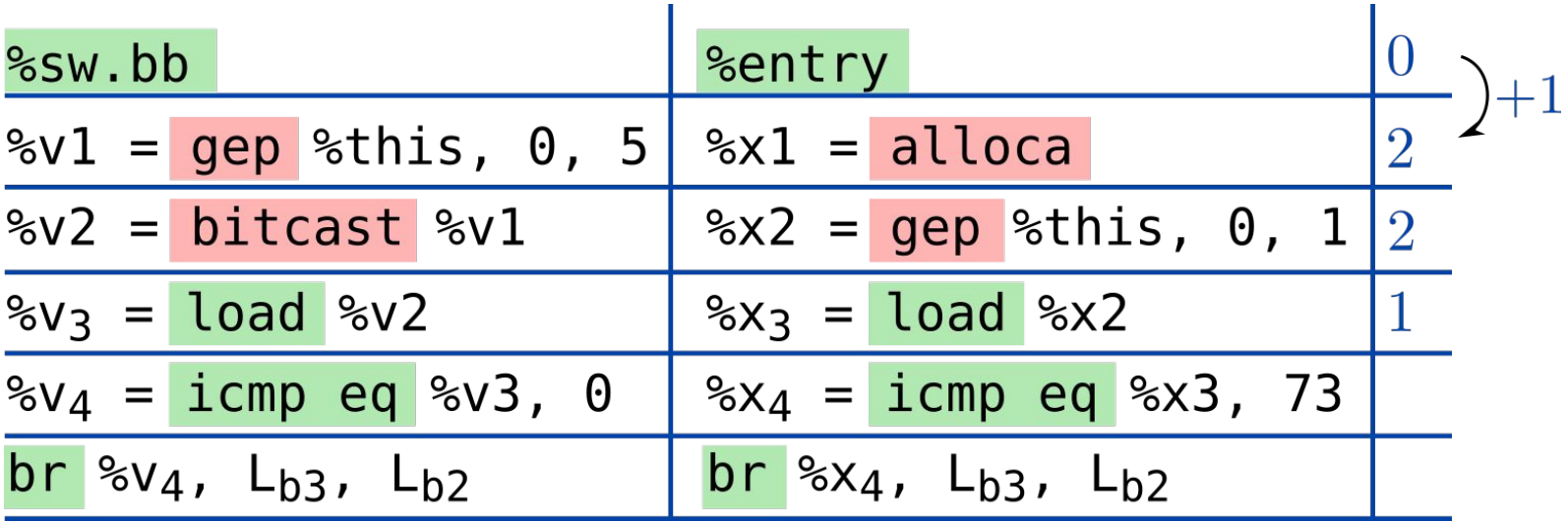
Profitability Analysis

Estimate size of merged block



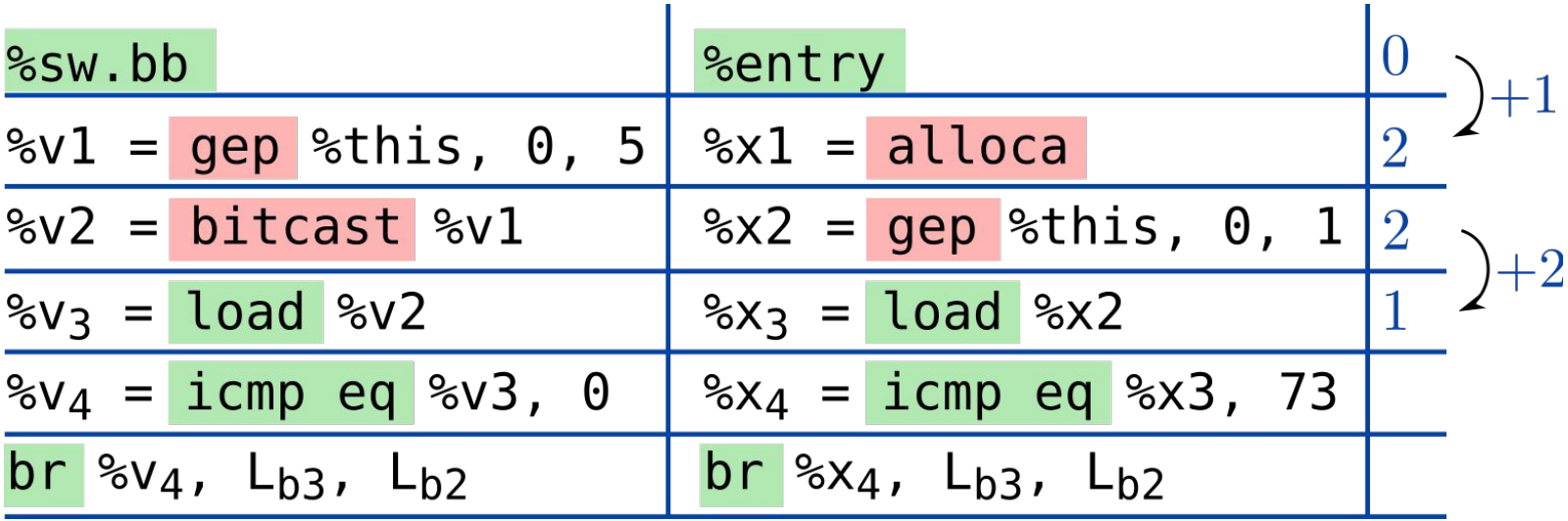
Profitability Analysis

Estimate size of merged block



Profitability Analysis

Estimate size of merged block



Profitability Analysis

Estimate size of merged block

%sw.bb	%entry	0	
%v1 = gep %this, 0, 5	%x1 = alloca	2	+1
%v2 = bitcast %v1	%x2 = gep %this, 0, 1	2	
$\%v_3$ = load %v2	$\%x_3$ = load %x2	1	+2
$\%v_4$ = icmp eq %v3, 0	$\%x_4$ = icmp eq %x3, 73	1	
br $\%v_4$, L_{b3}, L_{b2}	br $\%x_4$, L_{b3}, L_{b2}	1	

Profitability Analysis

Estimate size of merged block

%sw.bb	%entry	0	
%v1 = gep %this, 0, 5	%x1 = alloca	2	+1
%v2 = bitcast %v1	%x2 = gep %this, 0, 1	2	
$\%v_3$ = load %v2	$\%x_3$ = load %x2	1	+2
$\%v_4$ = icmp eq %v3, 0	$\%x_4$ = icmp eq %x3, 73	1	
br $\%v_4$, L_{b3}, L_{b2}	br $\%x_4$, L_{b3}, L_{b2}	1	

Merged Cost: 10 ✓

Independence from Code Layout

All prior techniques fail to merge

```
SPxId id(int i) const {
  if (rep() == ROW) {
    SPxRowId rid = SPxLP::rId(i);
    return SPxId(rid);
  } else {
    SPxColId cid = SPxLP::cId(i);
    return SPxId(cid);
  }
}
```

```
SPxId coId(int i) const {
  if (rep() == ROW) {
    SPxColId cid = SPxLP::cId(i);
    return SPxId(cid);
  } else {
    SPxRowId rid = SPxLP::rId(i);
    return SPxId(rid);
  }
}
```

Independence from Code Layout

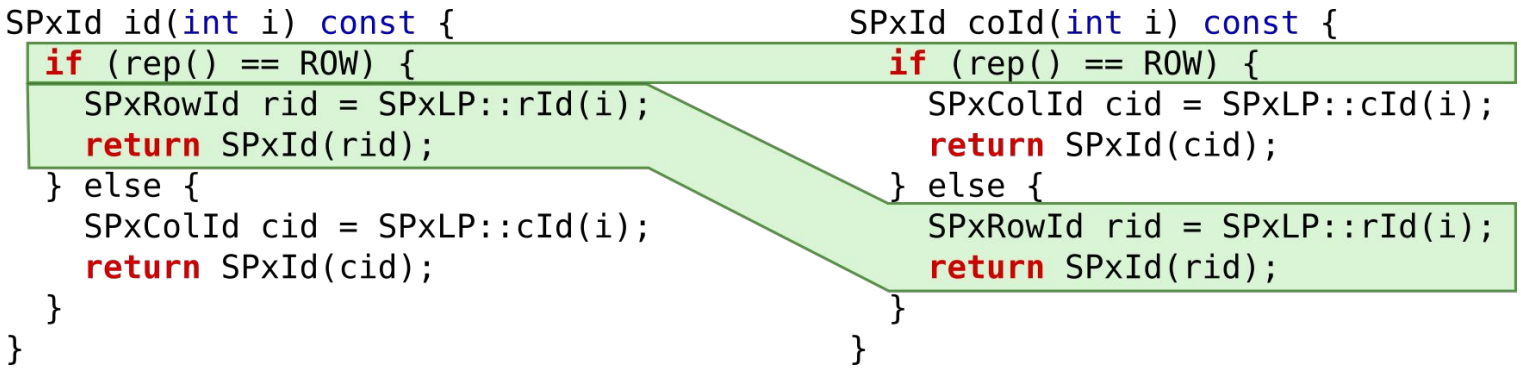
HyFM correctly pair the basic blocks

```
SPxId id(int i) const {
  if (rep() == ROW) {
    SPxRowId rid = SPxLP::rId(i);
    return SPxId(rid);
  } else {
    SPxColId cid = SPxLP::cId(i);
    return SPxId(cid);
  }
}
```

```
SPxId coId(int i) const {
  if (rep() == ROW) {
    SPxColId cid = SPxLP::cId(i);
    return SPxId(cid);
  } else {
    SPxRowId rid = SPxLP::rId(i);
    return SPxId(rid);
  }
}
```

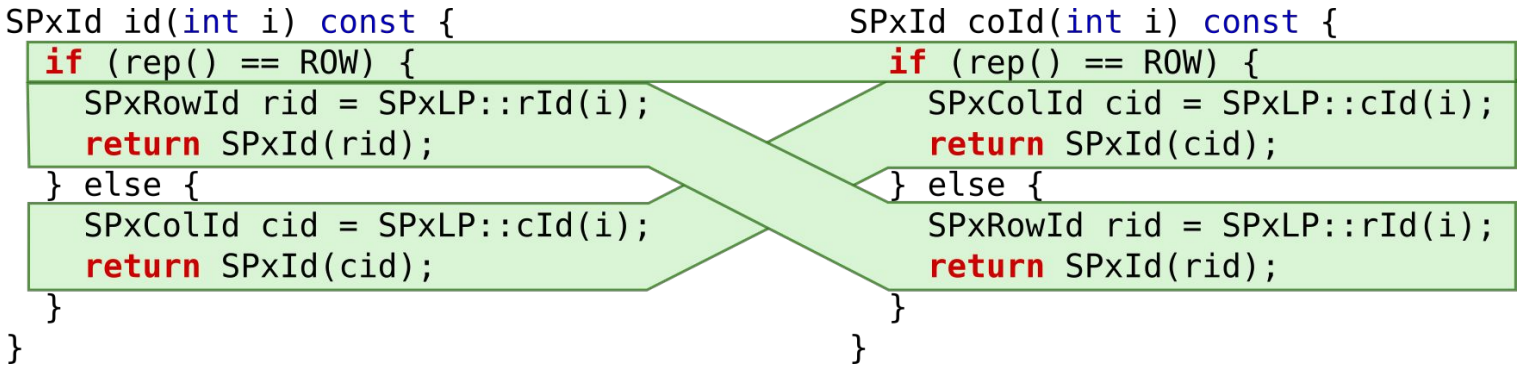
Independence from Code Layout

HyFM correctly pair the basic blocks



Independence from Code Layout

HyFM correctly pair the basic blocks



Evaluation Setup

Link-time optimization with -Os

SalSSA: State of the art

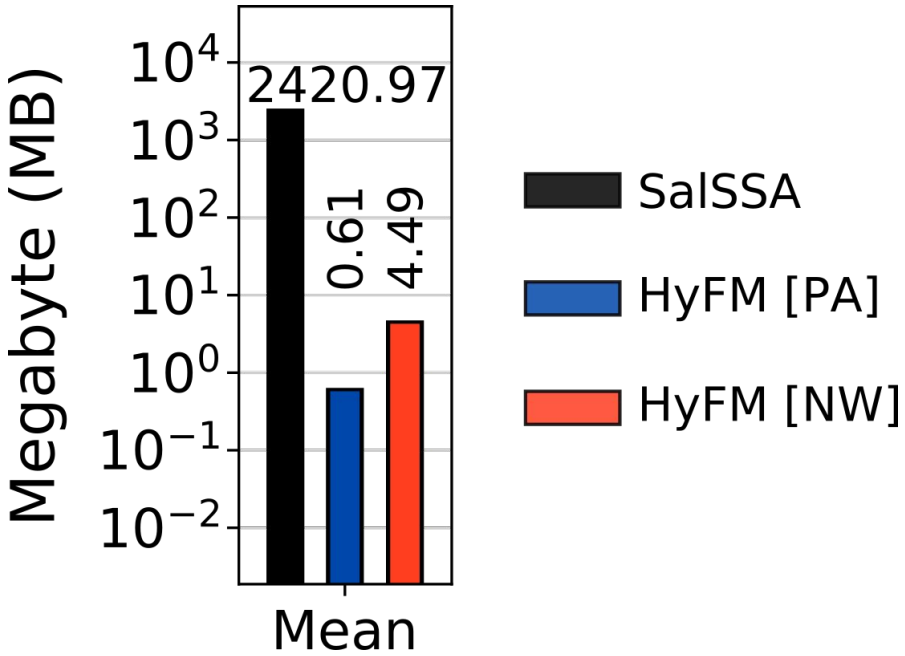
HyFM [PA]: Pairwise alignment

HyFM [NW]: Quadratic sequence alignment per block

Peak Memory Usage

HyFM reduces memory usage by multiple orders of magnitude

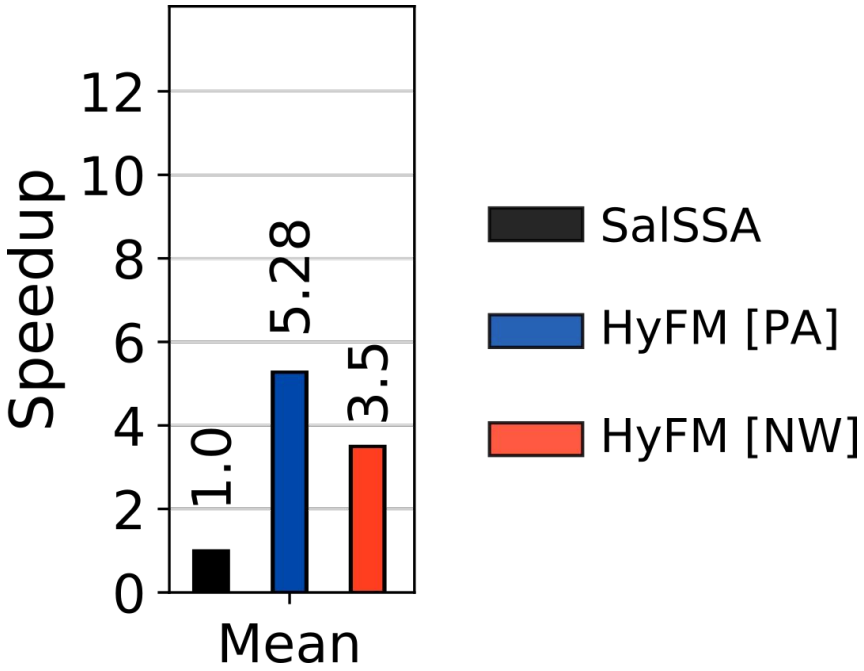
SPEC 2017



Speedup of the Optimization Pass

HyFM reduces memory usage by multiple orders of magnitude

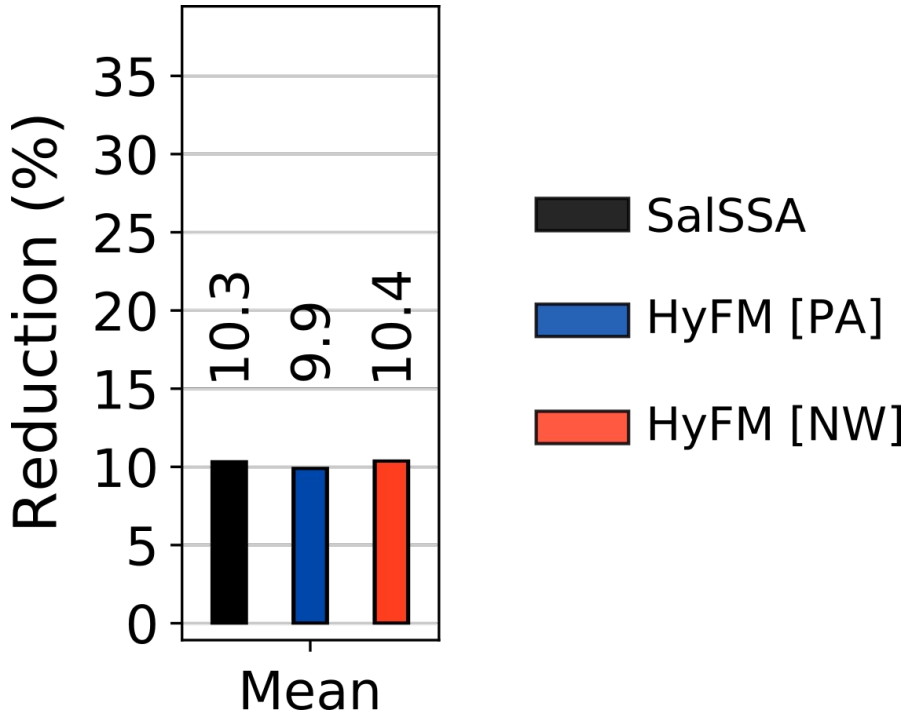
SPEC 2017



Object Size Reduction

HyFM is on a par with the State of the Art

SPEC 2017



Average Reduction Speed

HyFM has the best trade-off between code reduction and compilation time

SPEC 2017