Loop Aware IR-Level Annotation Framework for Performance Estimation in Native Simulation

Omayma matoussi Frédéric Pétrot

TIMA Laboratory – France

01/17/2017

Outline

Introduction

- Multiprocessor system on chip (MPSoC) platforms are becoming more software-centric.
- Software has a considerable impact on the overall performance of the system.
- Hardware/software co-simulation is essential during the co-design process of MPSoC platforms
  - early SW development
  - architecture exploration and HW/SW co-verification
  - performance estimation

Introduction

- Multiprocessor system on chip (MPSoC) platforms are becoming more software-centric.
- Software has a considerable impact on the overall performance of the system.
- Hardware/software co-simulation is essential during the co-design process of MPSoC platforms
  - early SW development
  - architecture exploration and HW/SW co-verification
  - **performance estimation**

Introduction

- Instruction interpretation (ISS, DBT, SBT, etc.)
  - too many details of the software
  - cycle accurate
  - slow simulation speed
- Native simulation
  - Very high abstraction level.
  - very fast.
  - absence of non-functional information.

Introduction

- Instruction interpretation (ISS, DBT, SBT, etc.)
  - too many details of the software
  - cycle accurate
  - slow simulation speed
- Native simulation
  - Very high abstraction level.
  - very fast.
  - **absence of non-functional information.**

Introduction

- How to introduce non-functional information in native simulation?
  - How to extract non-functional information? (software analysis)
  - Where to insert non-functional information? (annotation level + mapping process)

Introduction

- How to introduce non-functional information in native simulation?
  - How to extract non-functional information? (software analysis)
  - **Where to insert non-functional information? (annotation level + mapping process)**

Objective

The objective of this work:

propose a mapping approach that has the following characteristics:

- it is architecture independent (high-level IR).
- it considers all compiler optimizations (IR + mapping).
- it is the least intrusive (both compiler-wise and original code-wise).
- it provides accurate mapping between IR and binary CFGs.
- it yields reasonable simulation time.

Outline

Software back-annotation

Back-annotation

It consists of inserting non-functional information (such as time information, energy metrics, data addresses, instruction count, etc.) into a high-level functional model.

```
for (i = 0; i < n; i++) {
 a = b + c;
T[i] = (i+1)*a;}
```

source code

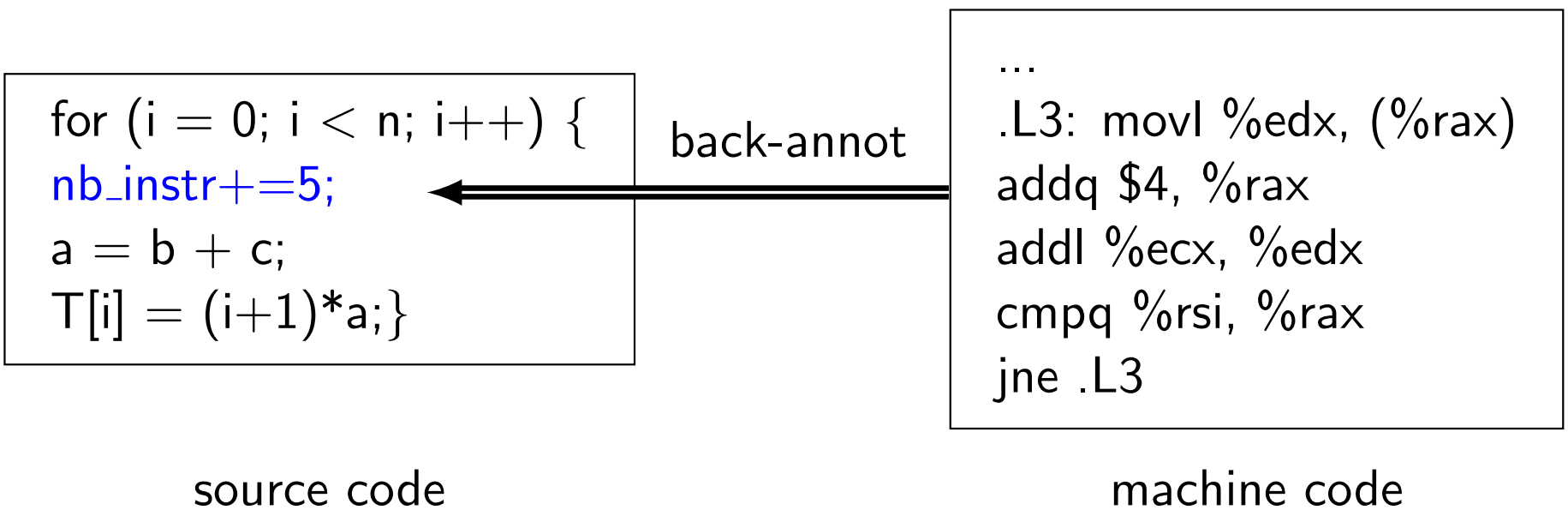
```
...
.L3: movl %edx, (%rax)
addq $4, %rax
addl %ecx, %edx
cmpq %rsi, %rax
jne .L3
```

machine code

Software back-annotation

Back-annotation

It consists of inserting non-functional information (such as time information, energy metrics, data addresses, instruction count, etc.) into a high-level functional model.



Source-level simulation

SLS

Source level simulation considers the source code as a functional model in which annotations will be inserted.

```
1: for (i = 0; i < n; i++) {
2: nb_instr+=5;
 3: a = b + c;
4: T[i] = (i+1)*a;}
```

source code

```
...
 .L3: movl %edx, (%rax)
addq $4, %rax
addl %ecx, %edx
cmpq %rsi, %rax
jne .L3
```

machine code

Source-level simulation

SLS

Source level simulation considers the source code as a functional model in which annotations will be inserted.

```
1: for (i = 0; i < n; i++) {
2: nb_instr+=5;
 3: a = b + c;
4: T[i] = (i+1)*a;}
```

source code

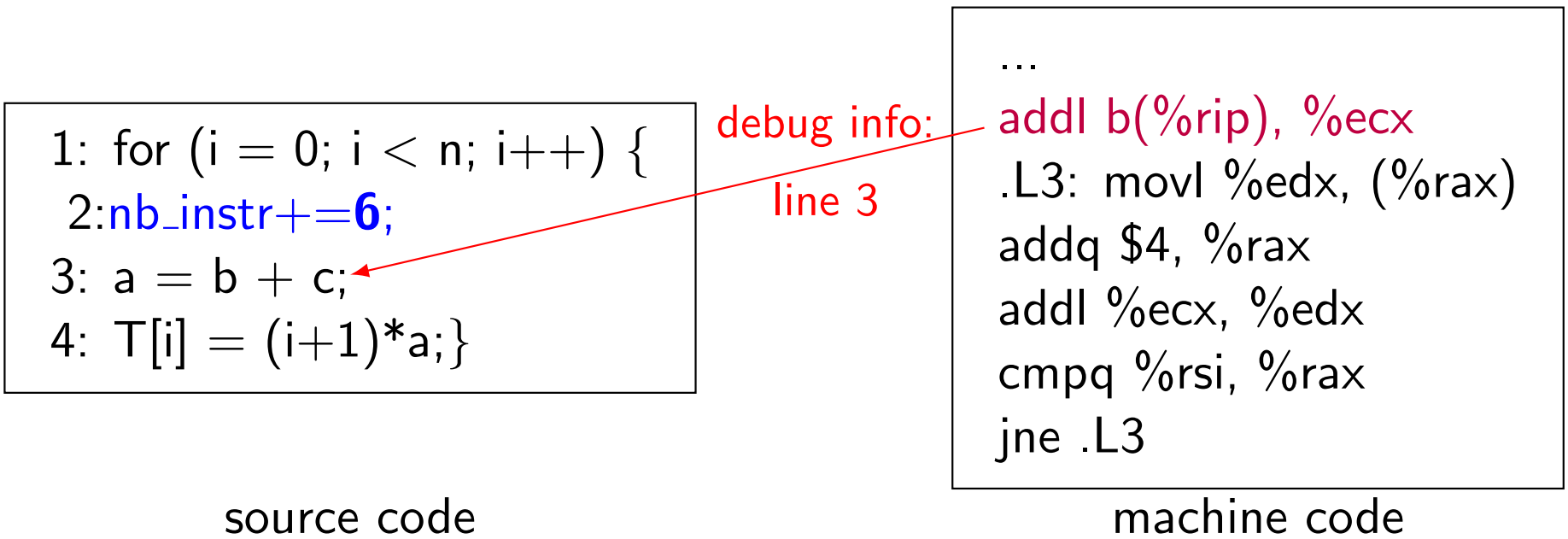
```
...
addl b(%rip), %ecx
.L3: movl %edx, (%rax)
addq $4, %rax
addl %ecx, %edx
cmpq %rsi, %rax
jne .L3
```

machine code

Source-level simulation

SLS

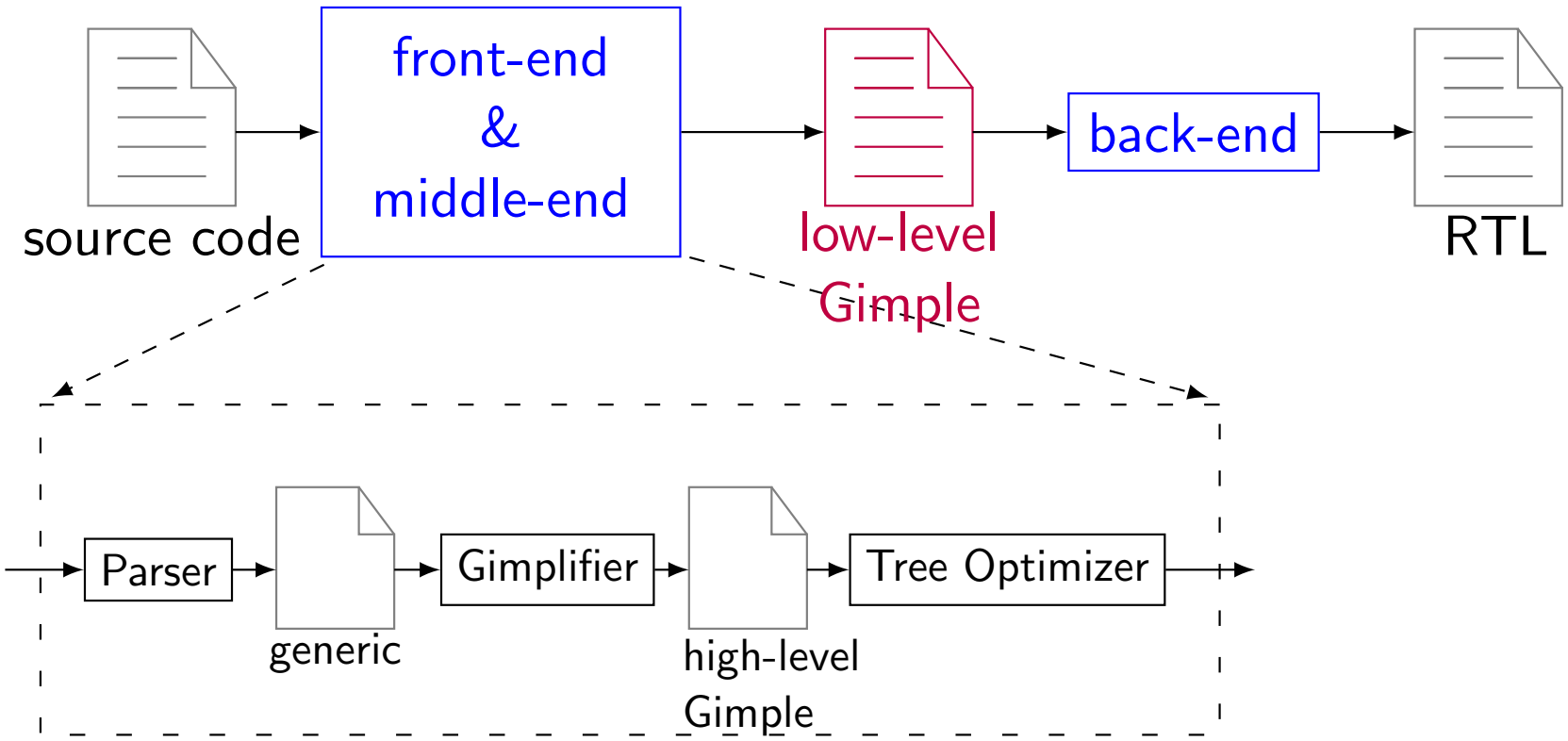
Source level simulation considers the source code as a functional model in which annotations will be inserted.



IR-level simulation

ILS

Intermediate level simulation uses the compiler intermediate representation as a functional model in which non-functional information will be back-annotated.



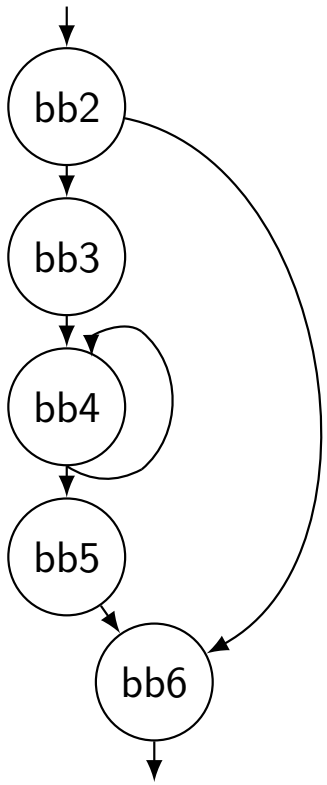
An IR example(1/2)

Control flow graph

A CFG is a directed graph where the nodes are basic blocks and the edges represent jumps between the nodes.

Basic block

A basic block is a sequence of consecutive instructions without any jumps, except for the last instruction, or jump targets, except for the first instruction.



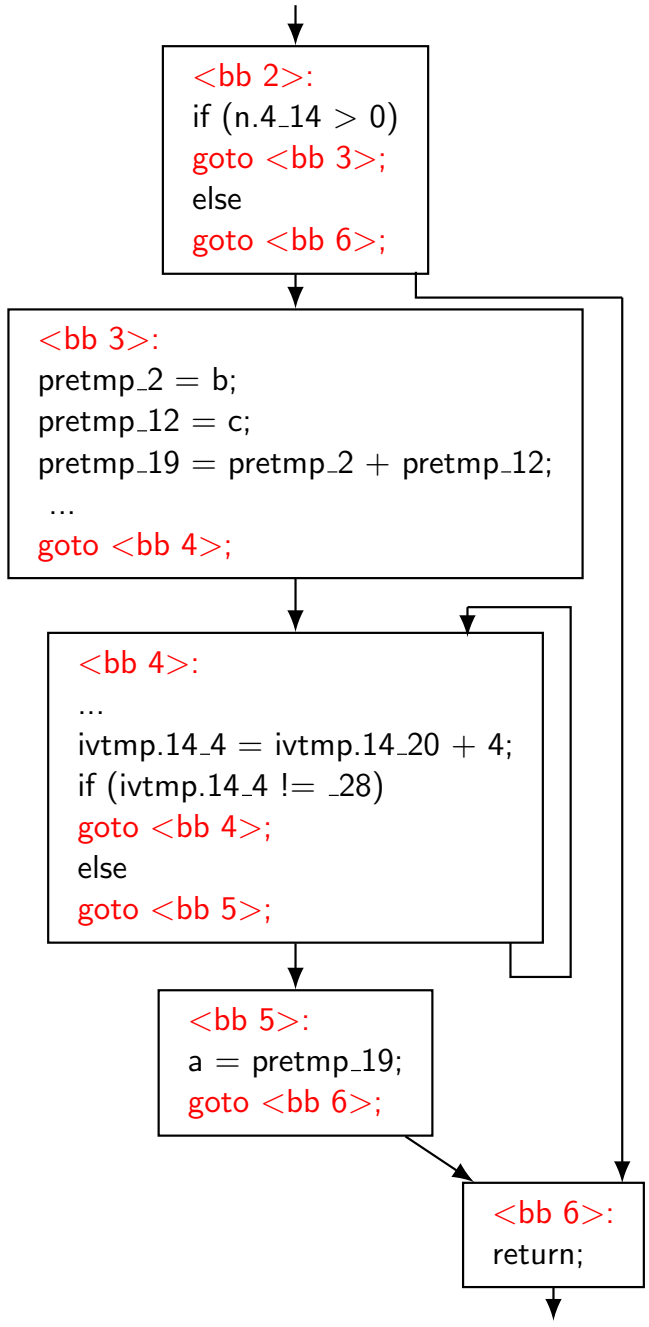
An IR example(2/2)

```
for (i = 0; i < n; i++) {
a = b + c;
T[i] = (i+1)*a;}
```

source code

corresponding

IR CFG



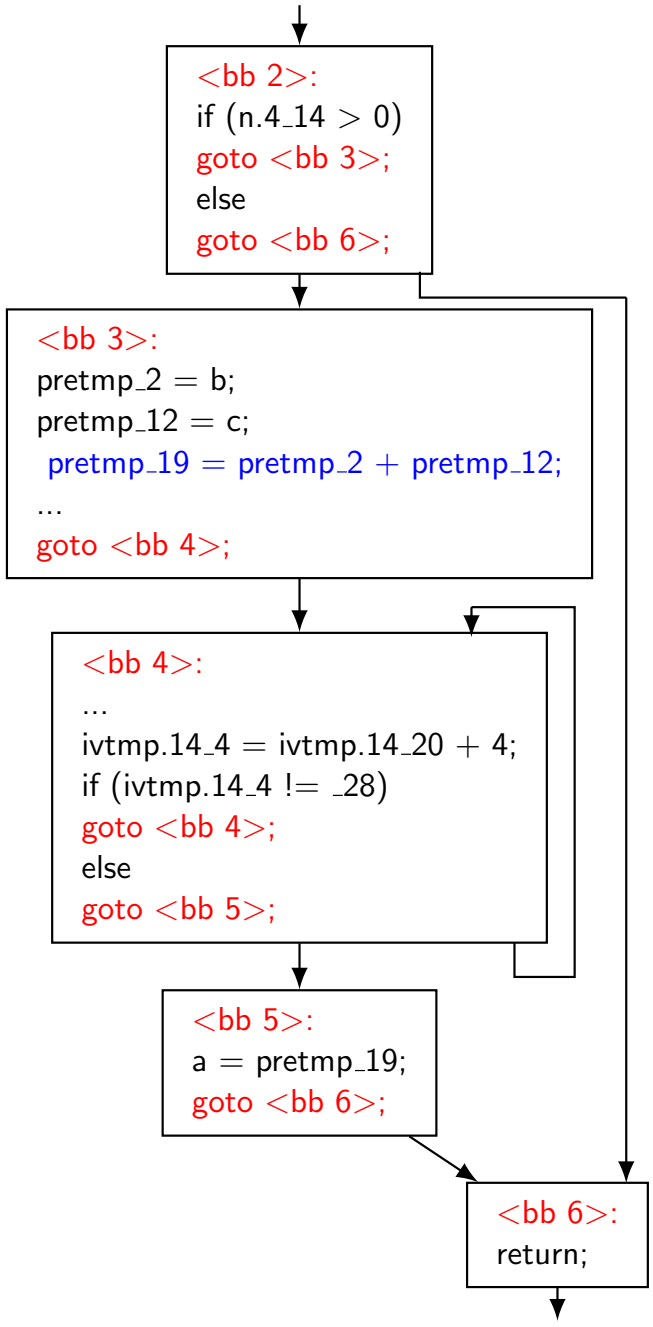
An IR example(2/2)

```
for (i = 0; i < n; i++) {
a = b + c;
T[i] = (i+1)*a;}
```

source code

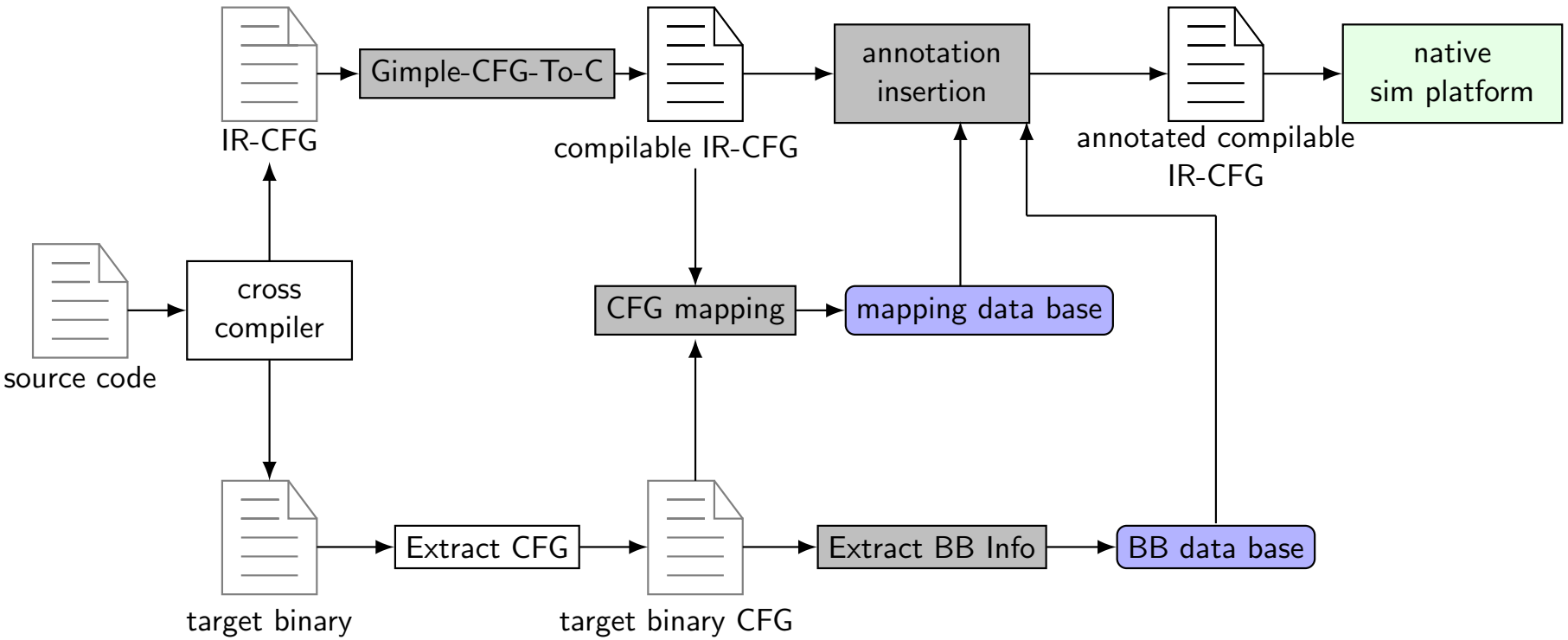
corresponding

IR CFG



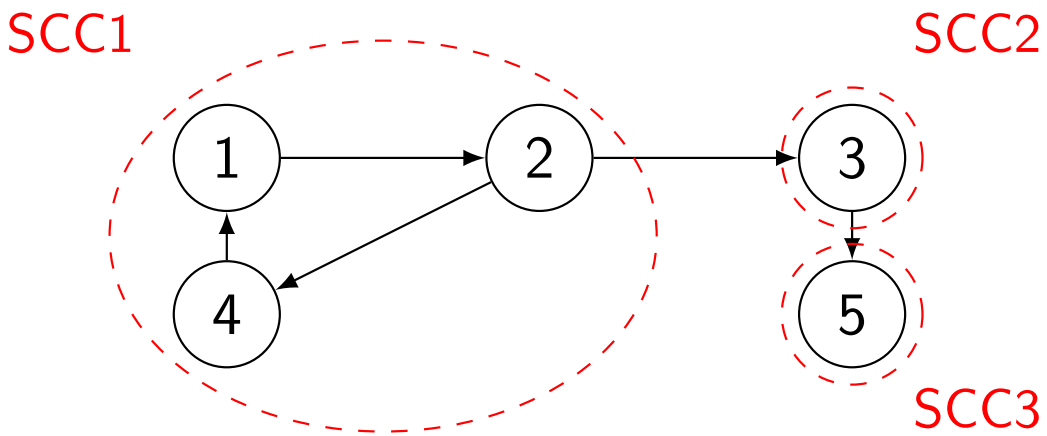
Outline

The proposed IR-annotation framework



Strongly connected component

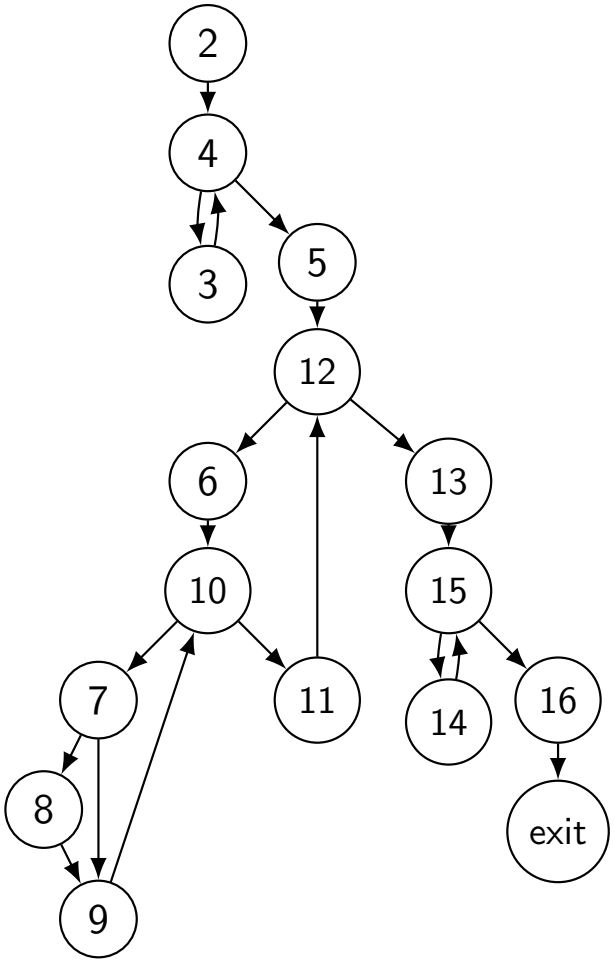
A SCC is a maximal set of vertices such that for every pair of vertices u and v in the set there is a path from u to v and a path from v to u.



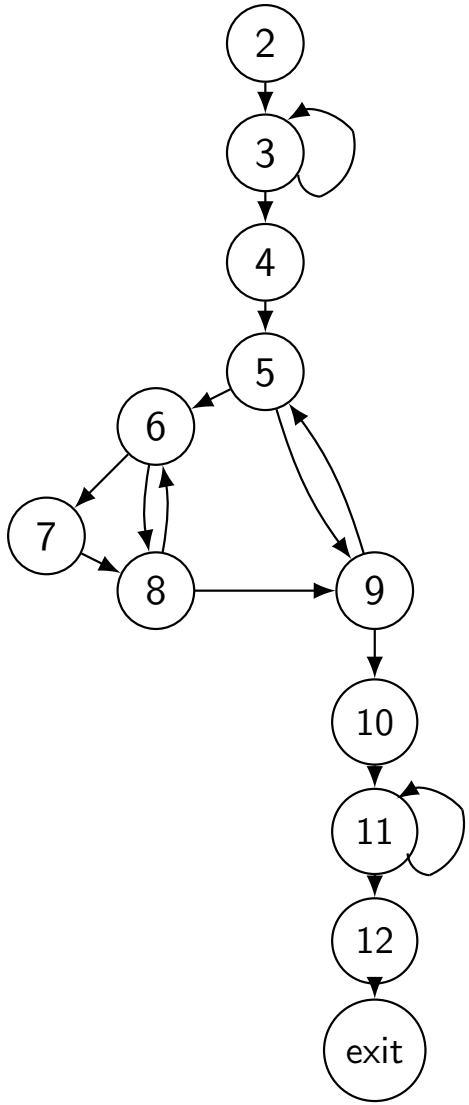
Fixedpoints

Two elements, each from a graph, that are determined to be equivalent, are considered to be fixedpoints.

The proposed mapping algorithm



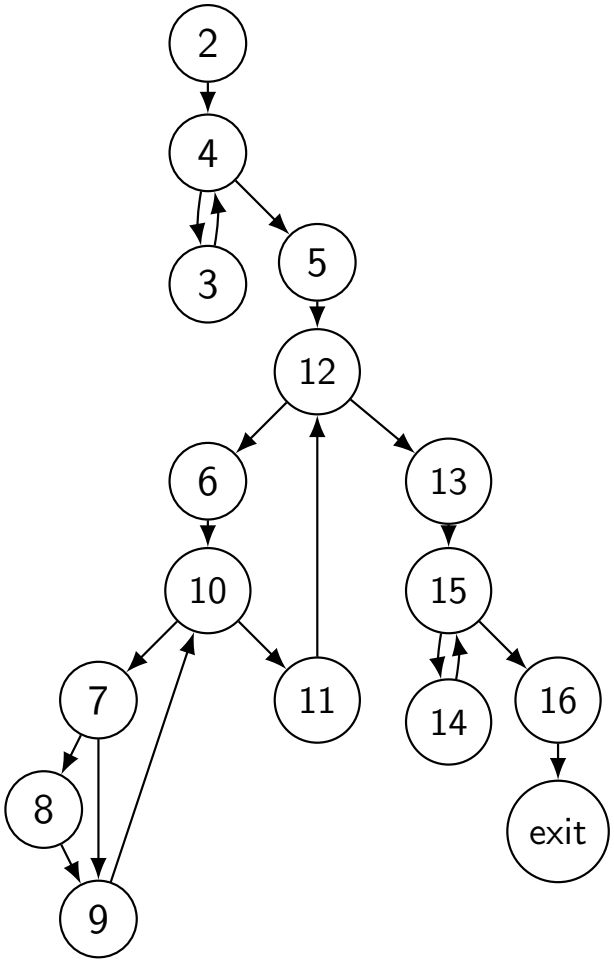
IR CFG



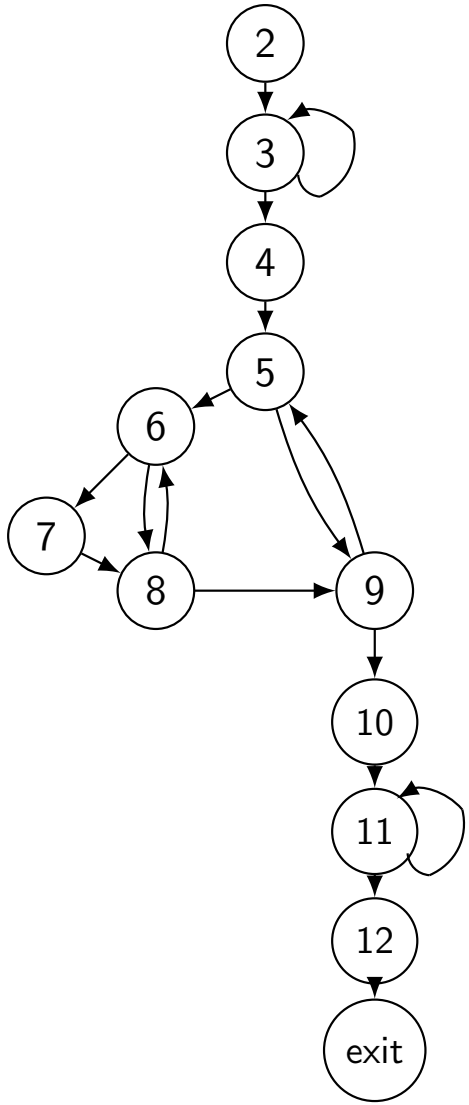
Bin CFG

The proposed mapping algorithm

1. Decompose the CFGs into SCCs.



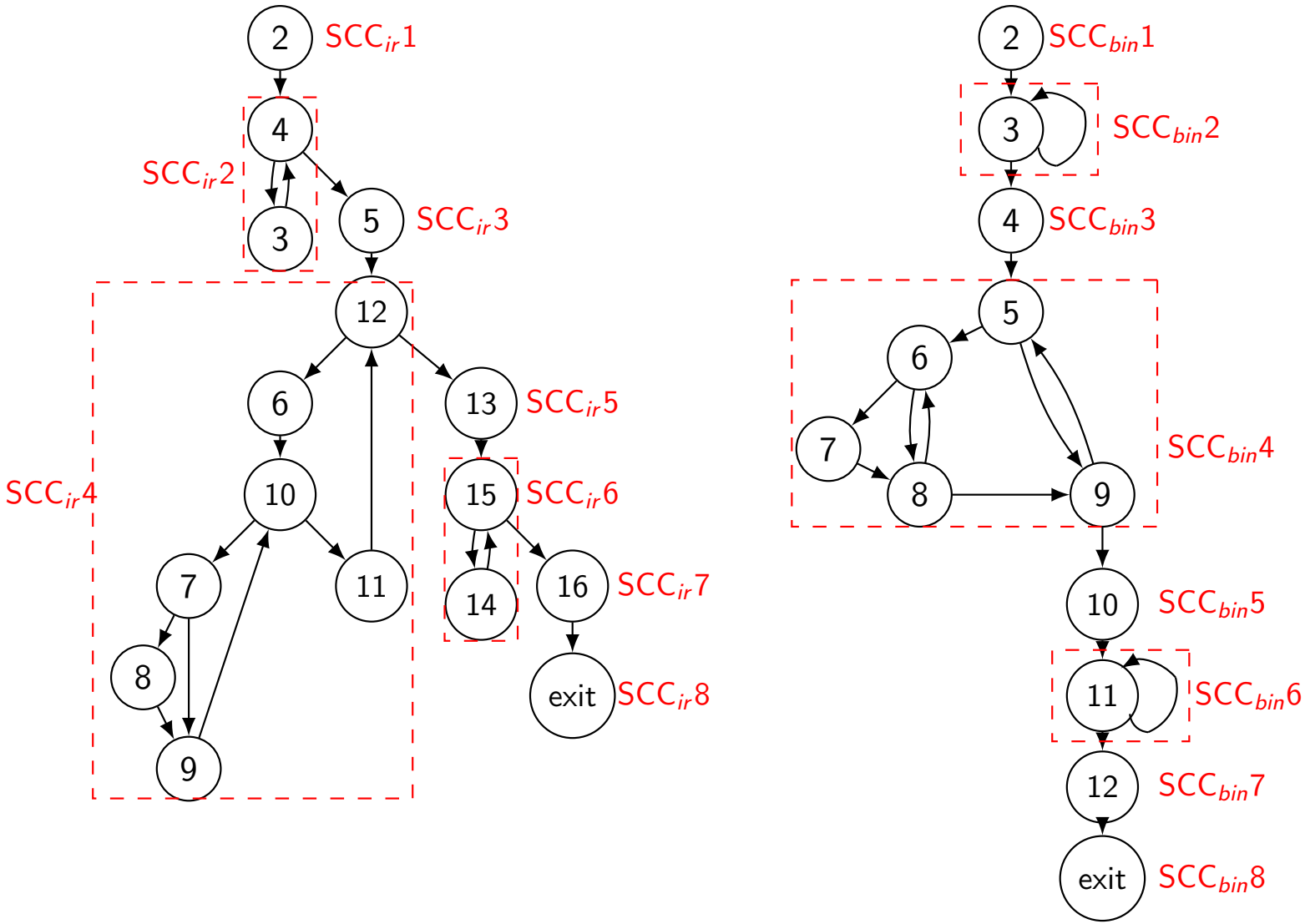
IR CFG



Bin CFG

The proposed mapping algorithm

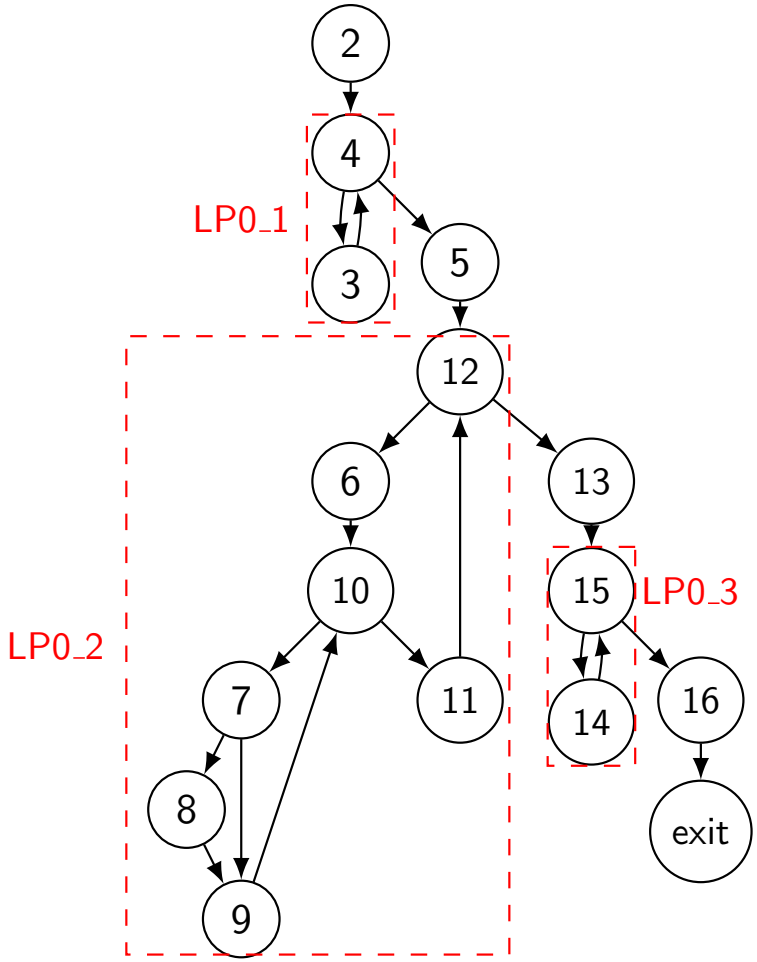
1. Decompose the CFGs into SCCs.



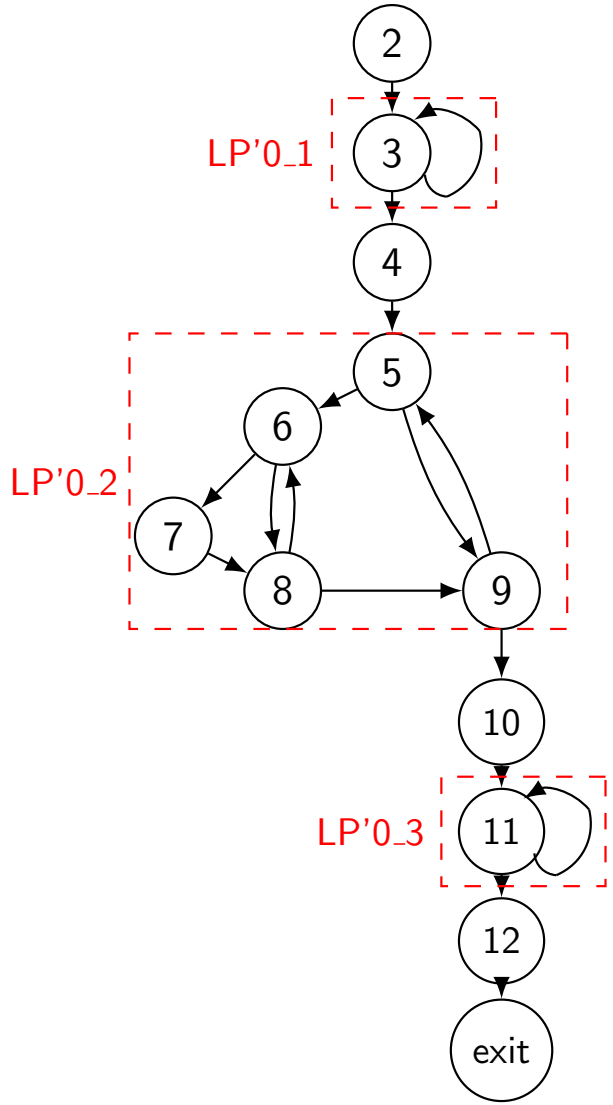
IR CFG **Bin CFG**

The proposed mapping algorithm

- A SCC with at least one arc is a loop block.



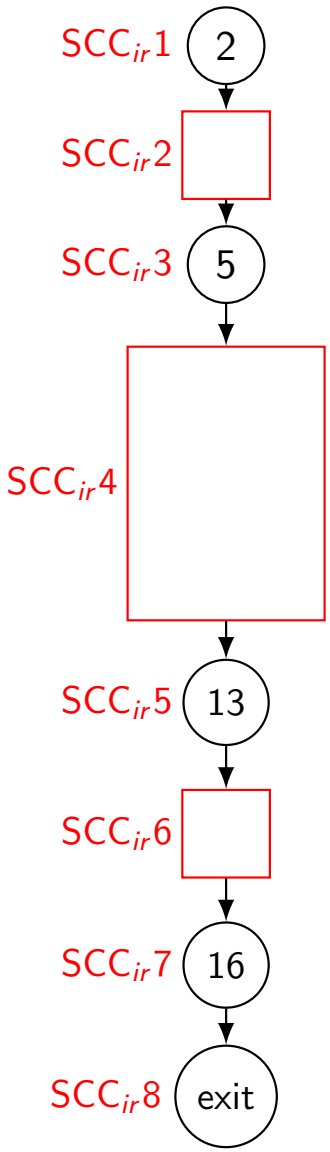
IR CFG



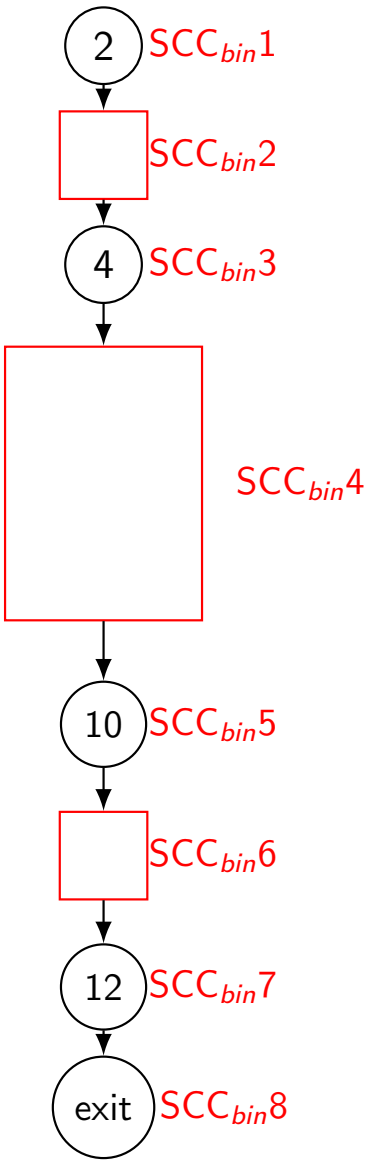
Bin CFG

The proposed mapping algorithm

2. Reconnect the SCCs to form a condensed CFG.



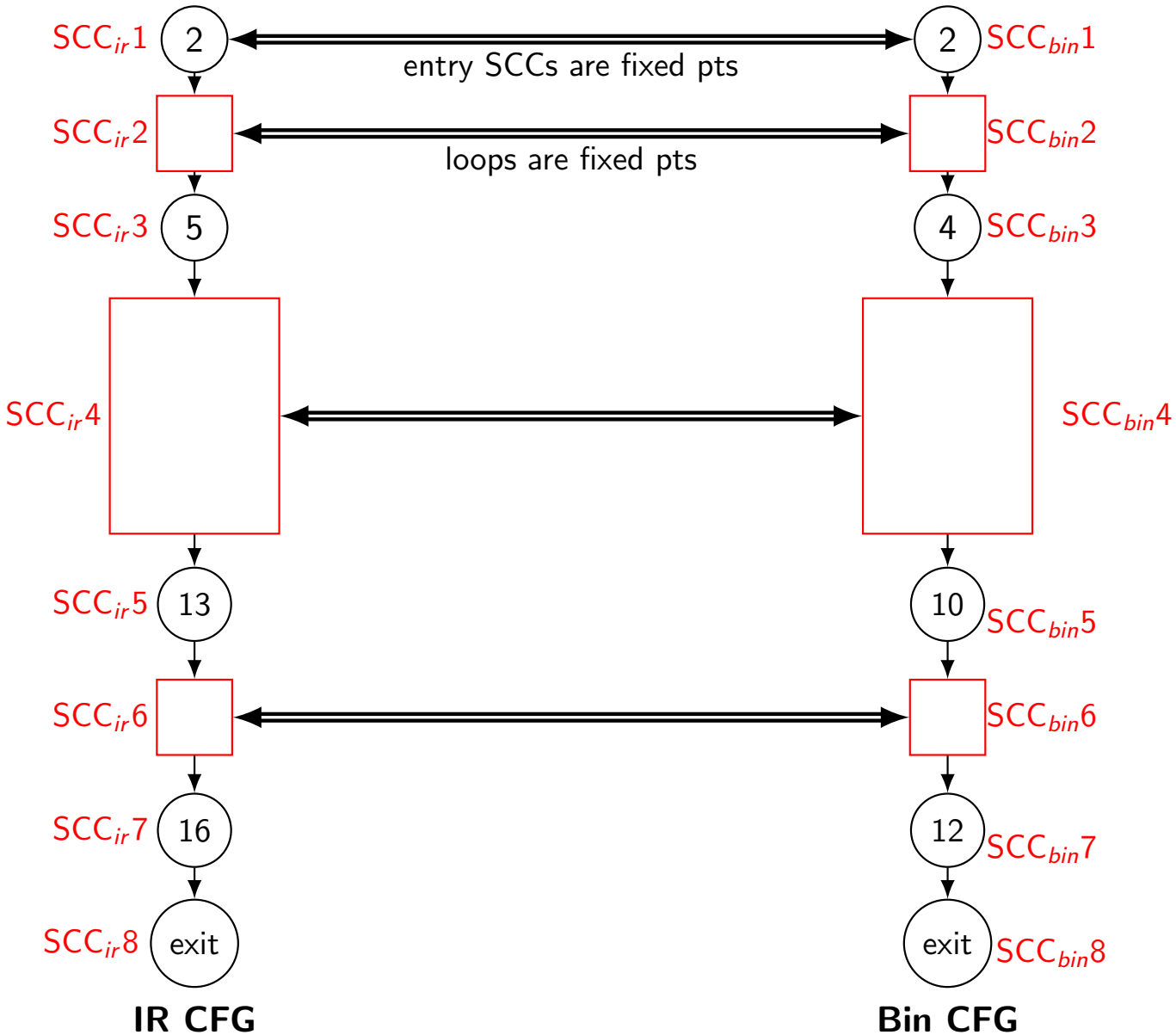
IR CFG



Bin CFG

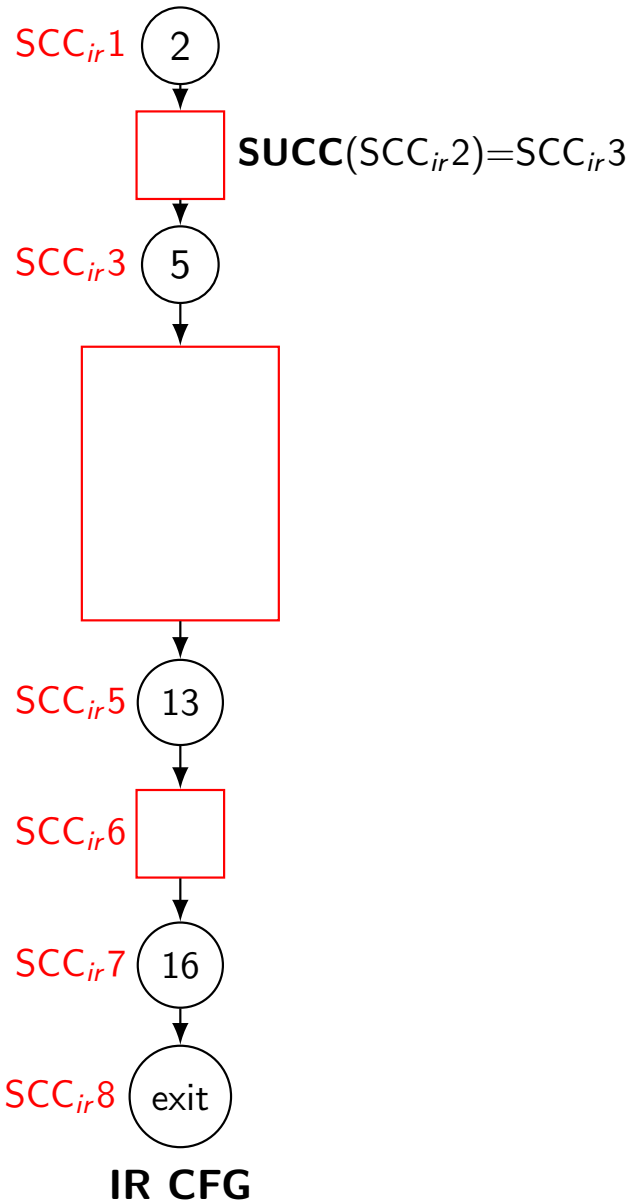
The proposed mapping algorithm

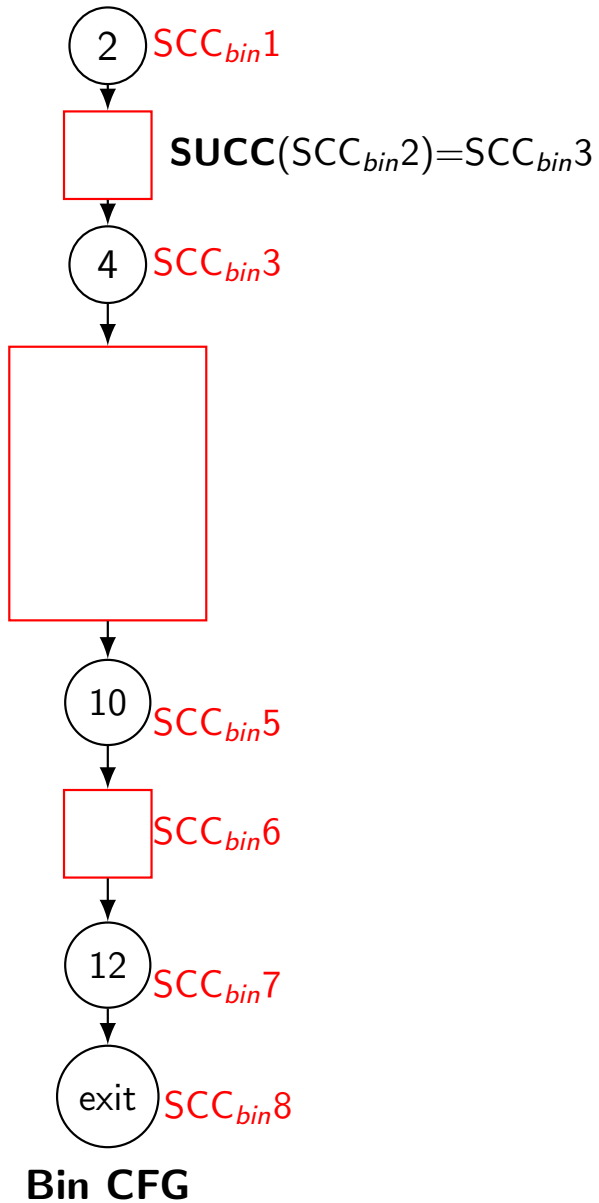
3. Match the condensed CFGs.



The proposed mapping algorithm

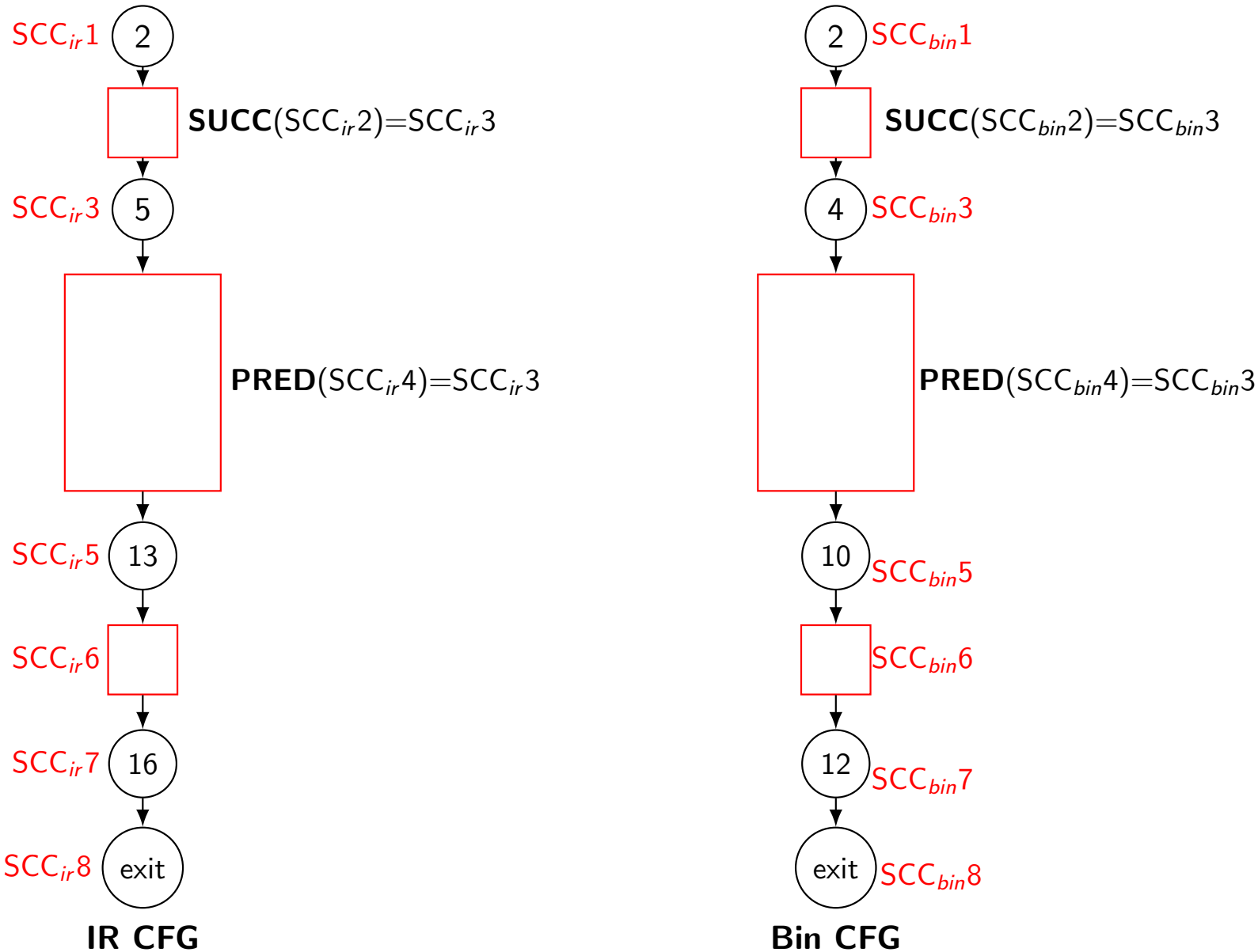
- Propagate fixedpoints using PRED/SUCC relations.





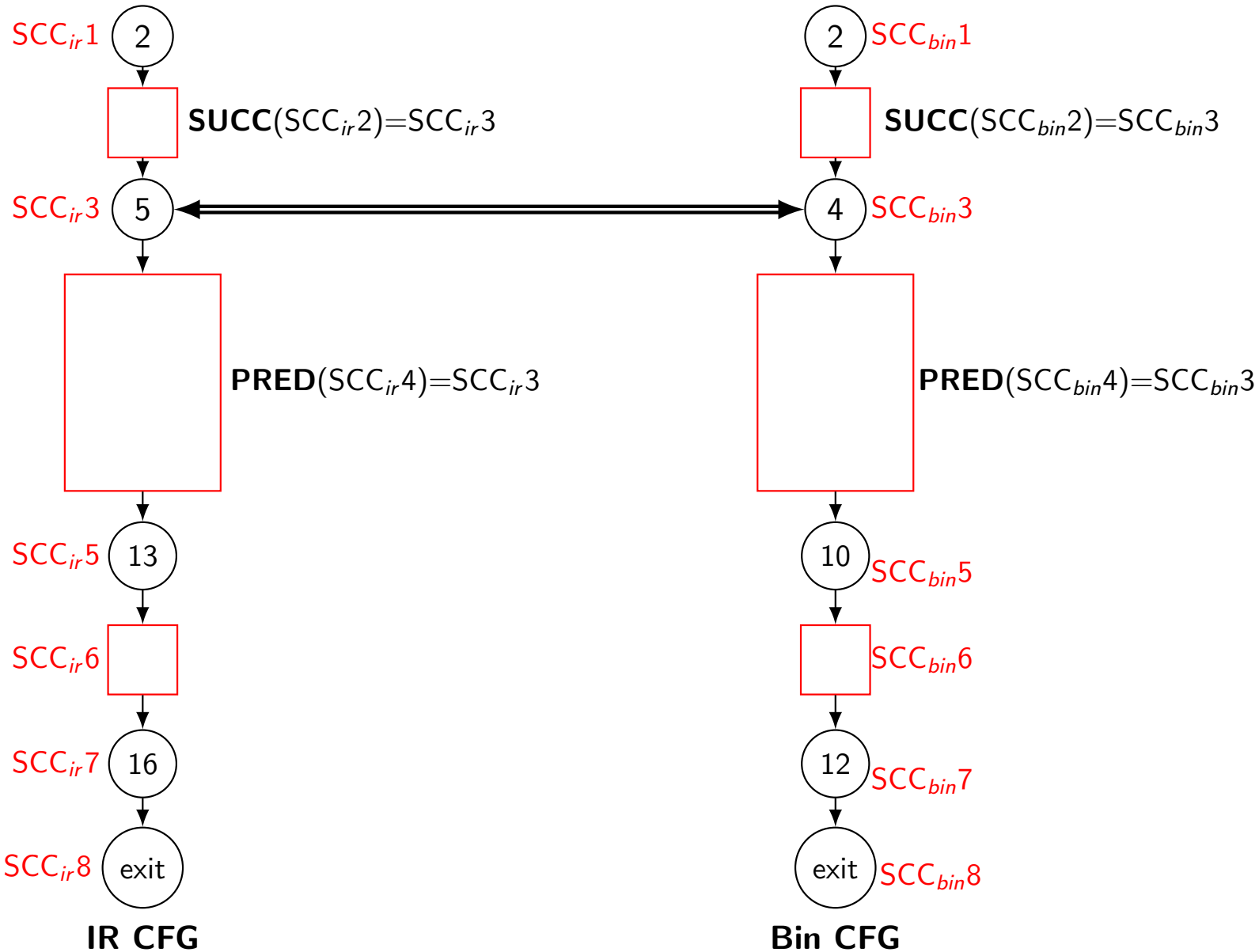
The proposed mapping algorithm

- Propagate fixedpoints using PRED/SUCC relations.



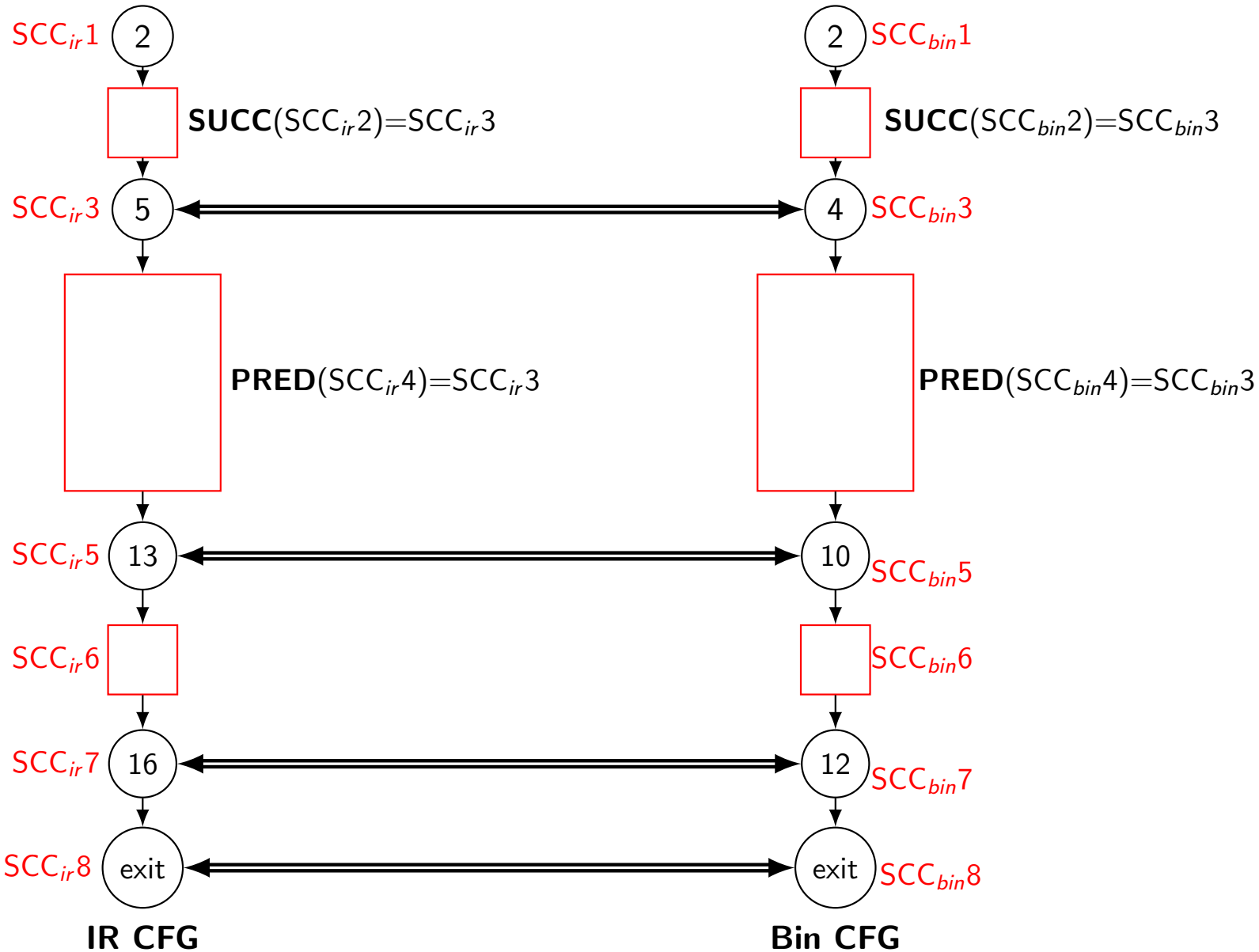
The proposed mapping algorithm

- Propagate fixedpoints using PRED/SUCC relations.



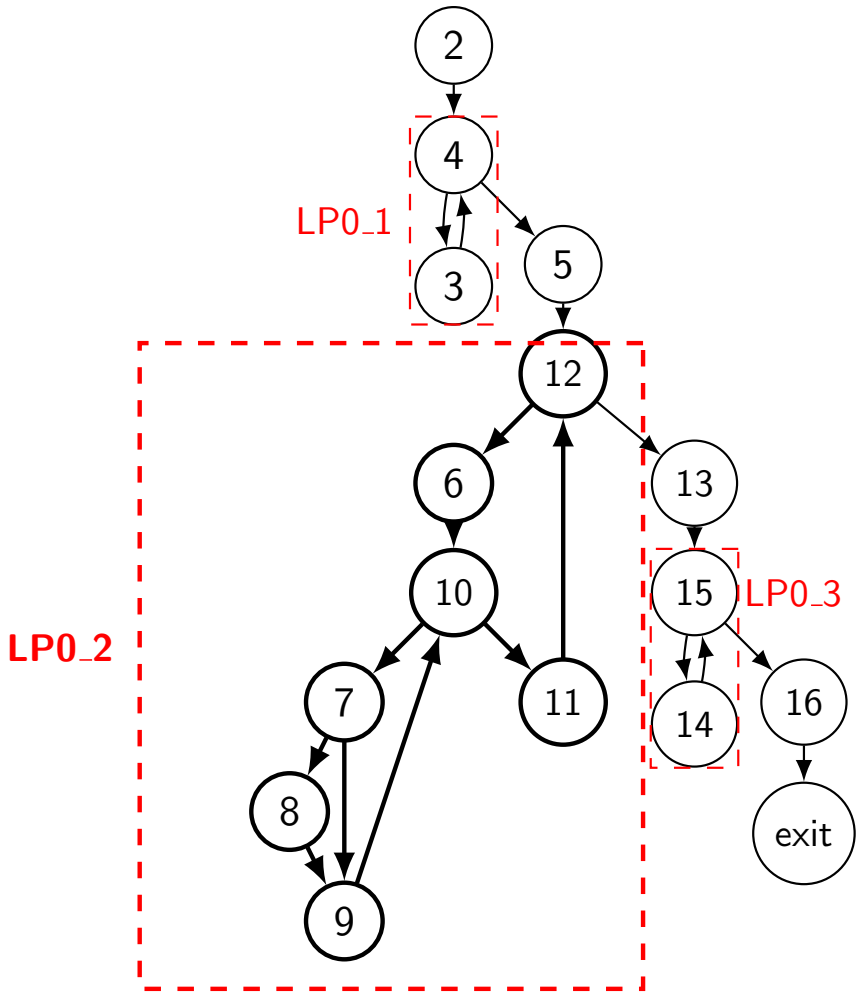
The proposed mapping algorithm

- Propagate fixedpoints using PRED/SUCC relations.

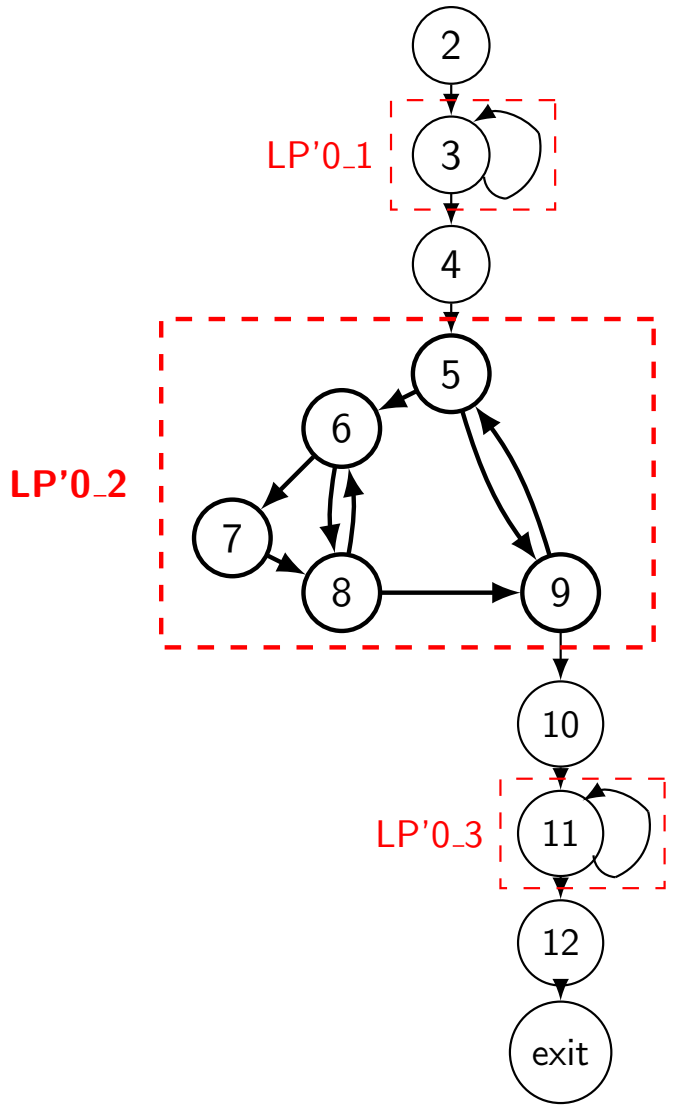


The proposed mapping algorithm

- inside loop blocks.



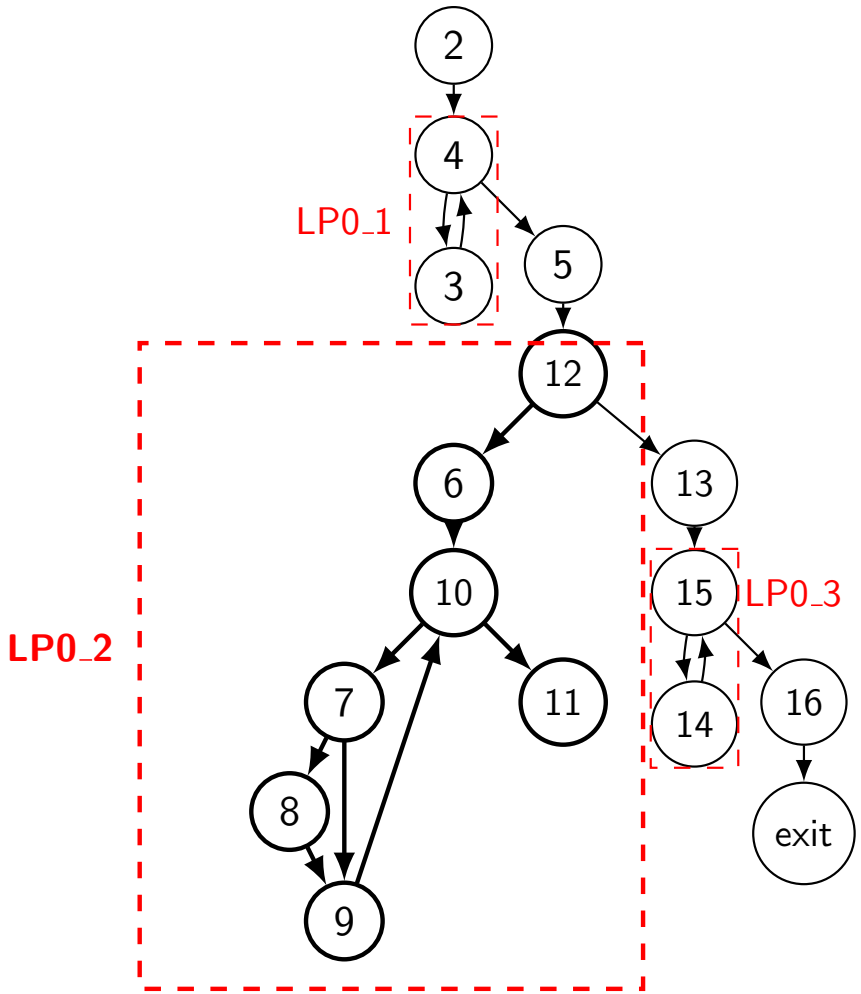
IR CFG



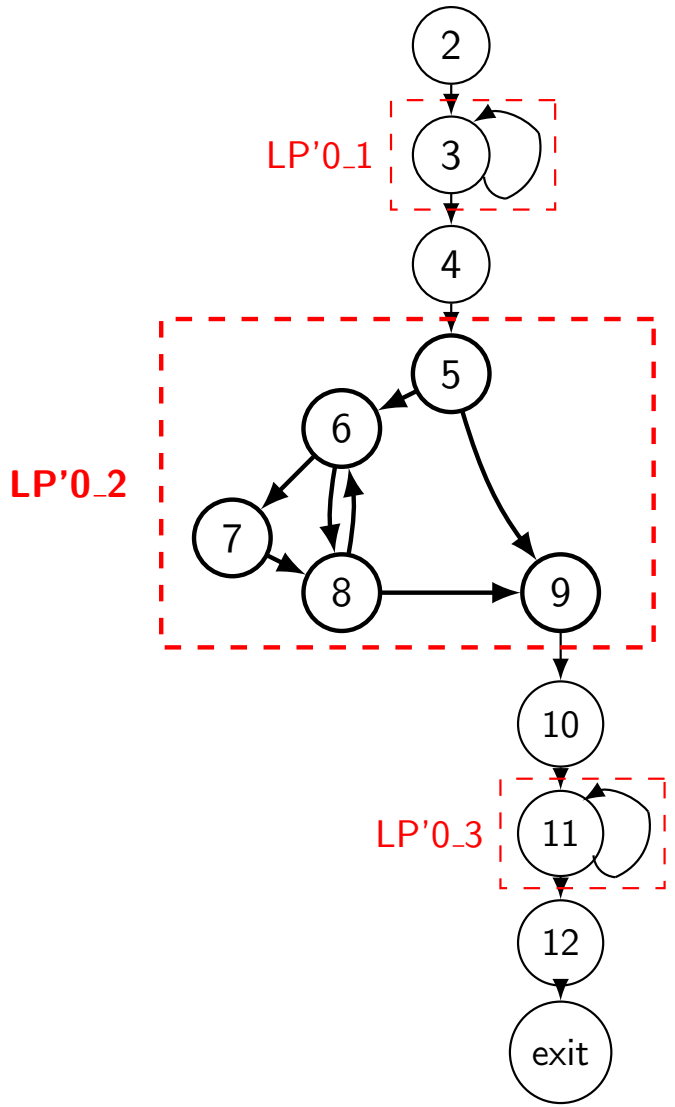
Bin CFG

The proposed mapping algorithm

- remove the back edge.



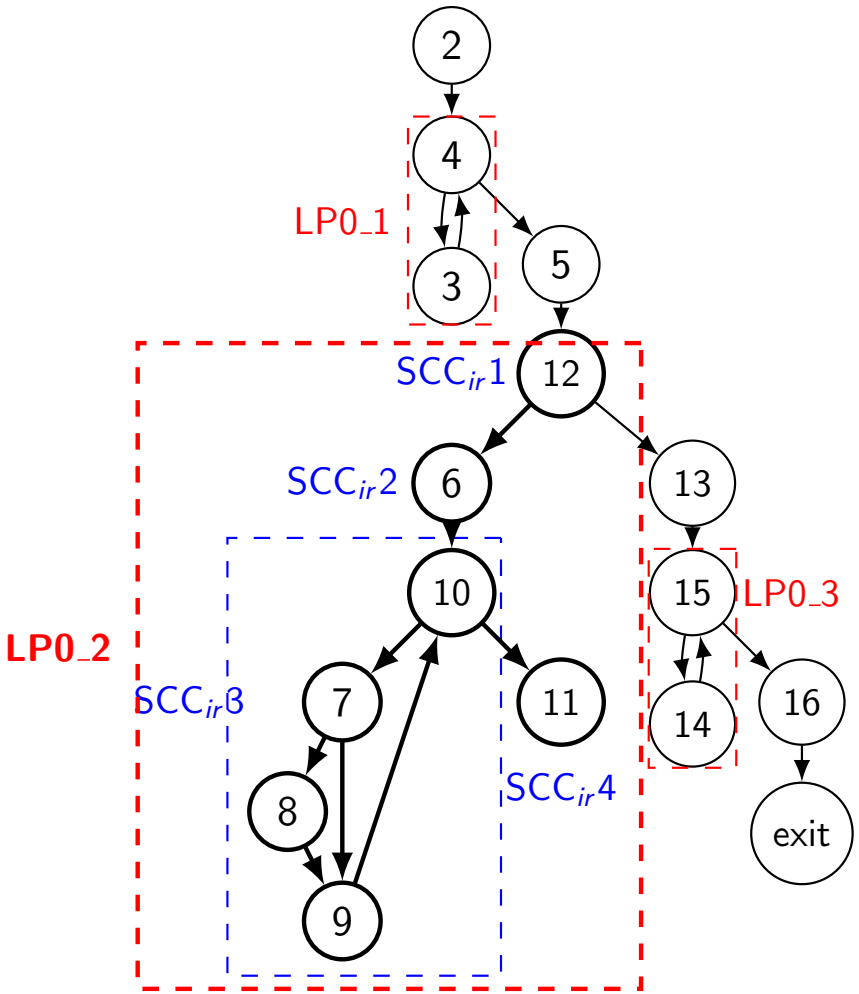
IR CFG



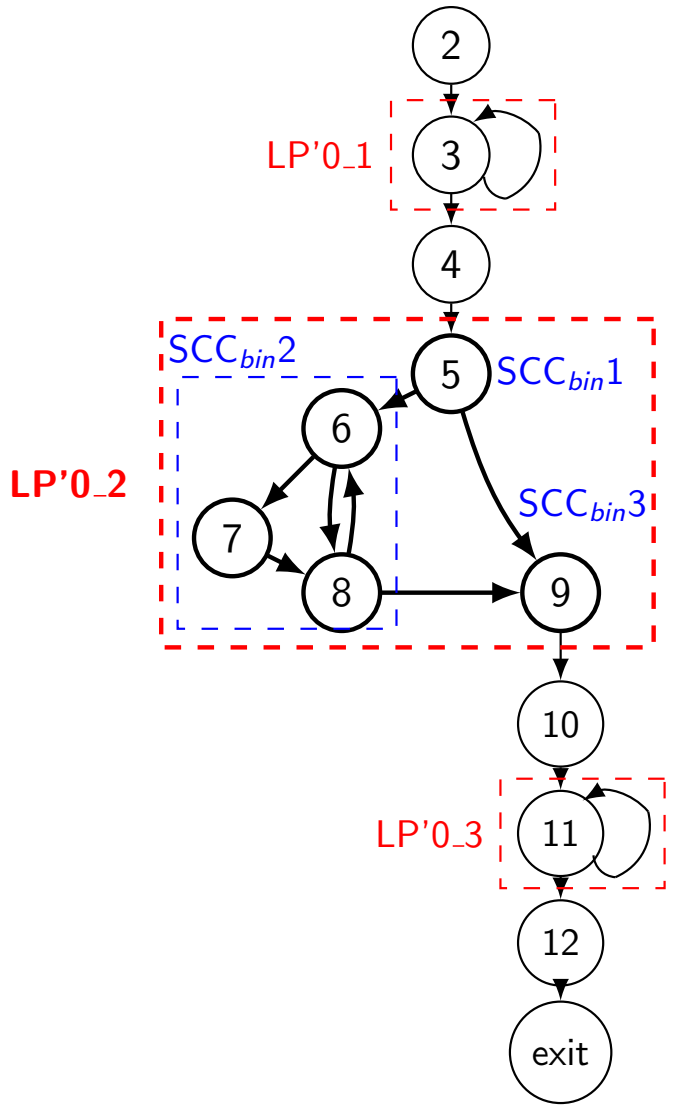
Bin CFG

The proposed mapping algorithm

- apply the mapping process on the loop block.



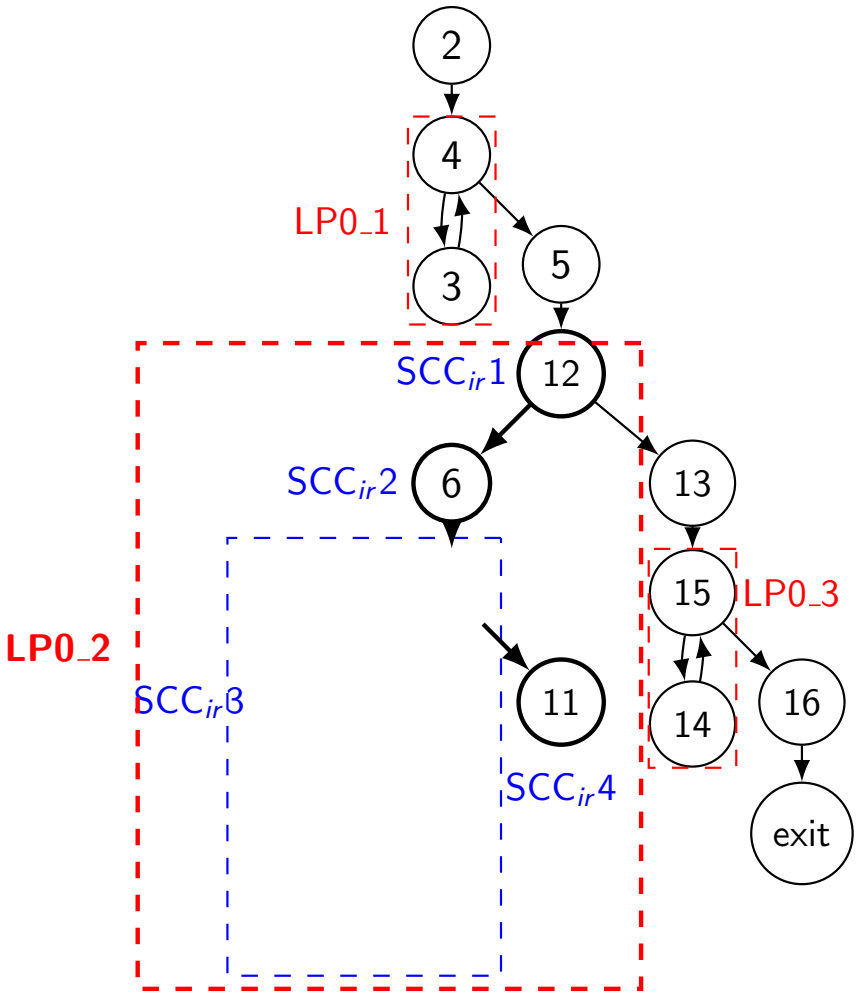
IR CFG



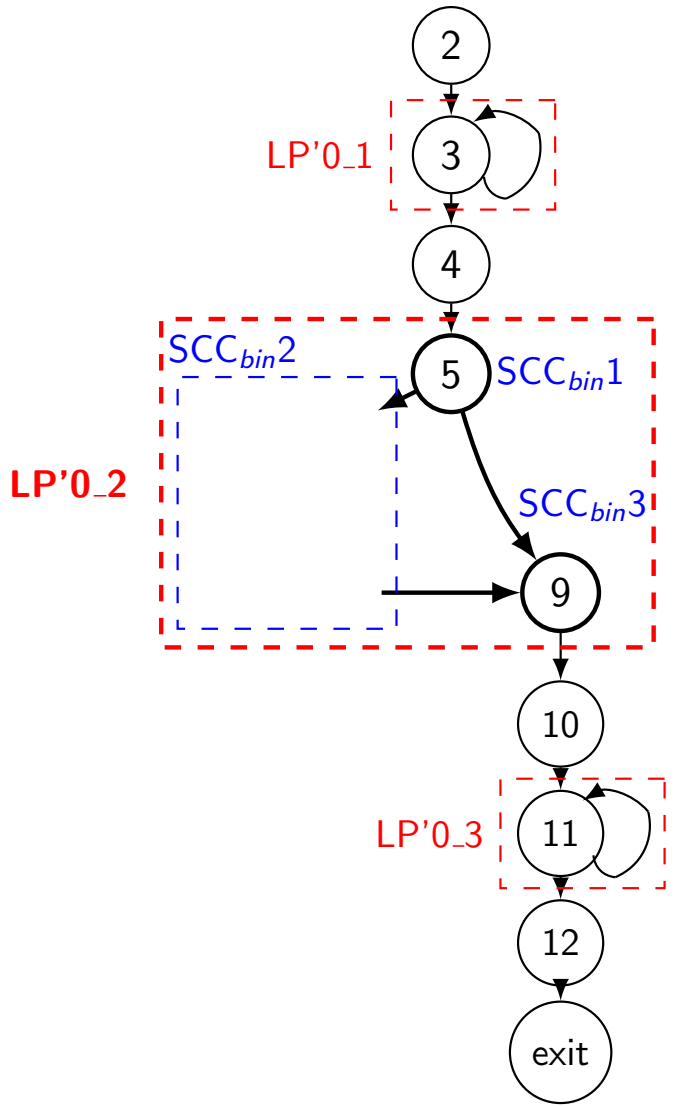
Bin CFG

The proposed mapping algorithm

- apply the mapping process on the loop block.



IR CFG



Bin CFG

Outline

Experimentation

- The target architecture is a 32-bit Kalray k1 core.
- The host processor is a 64-bit Intel x86 CPU.
- The native simulation platform is an in-house HW/SW co-simulation tool based on TLM.
- Our reference is a cycle accurate ISS platform by Kalray.
- The Benchmarks we used are Polybench and Splash.

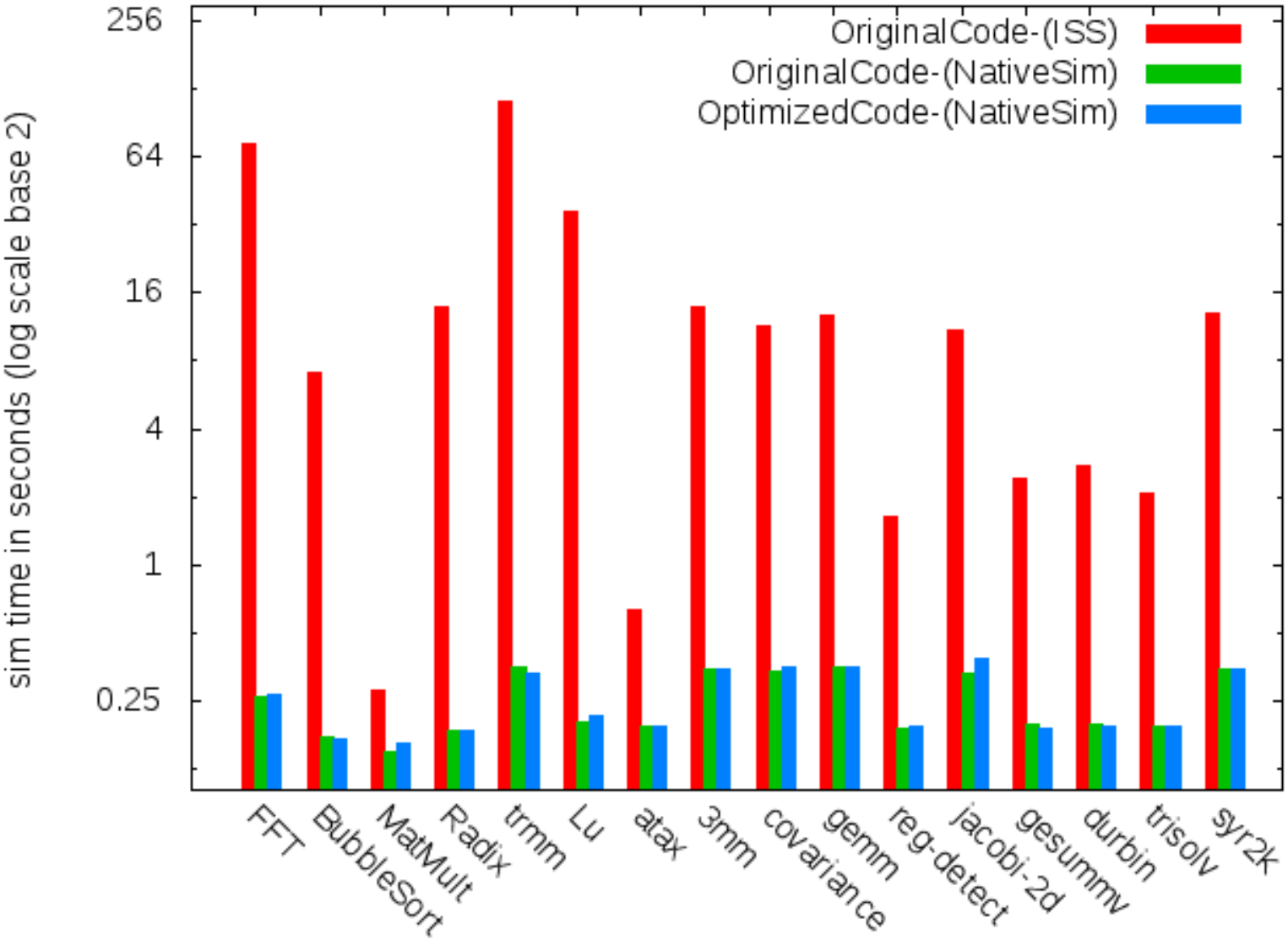
Instruction count estimates

Table 1: Comparison of the number of instructions

Benchmark	FFT	**BbSort**	MMult	Radix	Trmm	Lu	atax
ISS	35399998	**5451974**	79104	6565782	148695	16672930	28251
ILS	35038273	**5451998**	79111	6572218	151822	16568886	28571
ILS-ERROR	-1.02%	**0.00%**	0.00%	0.09%	2.10%	-0.62%	1.13%
SLS-ERROR	-67.08%	**-81.51%**	-21.97%	-39.48%	-28.84%	-41.55%	-53.14%

Benchmark	3mm	covar.	gemm	reg-detect	jacobi	gesu.	durbin
ISS	782941	175205	320192	10251	66719	28279	26224
ILS	806692	179861	318642	10310	68022	29066	27232
ILS-ERROR	3.03%	2.66%	-0.48%	0.58%	1.95%	2.78%	3.84%
SLS-ERROR	-22.4%	-69.98%	-31.05%	-66.22%	-74.97%	-68.2%	-68.67%

Simulation time



Comparison of simulation time

Outline

Conclusion

- We proposed a mapping scheme between the IR and binary CFGs for the purpose of performance estimation.
  - The mapping scheme focuses especially on loops because they are hot spots.
  - Our approach is architecture-independent and takes into account compiler front-end and back-end optimizations.
- Experiments underline the accuracy of the mapping approach and its reasonable simulation time.

Perspectives