
Binary Translation Using Peephole Superoptimizers

Sorav Bansal and Alex Aiken
Stanford University

TALK OUTLINE

- What is Binary Translation?
- What are Peephole Superoptimizers?
- Why Binary Translation Using Peephole Superoptimizers?
- Challenges and Solutions
- Experiments and Results

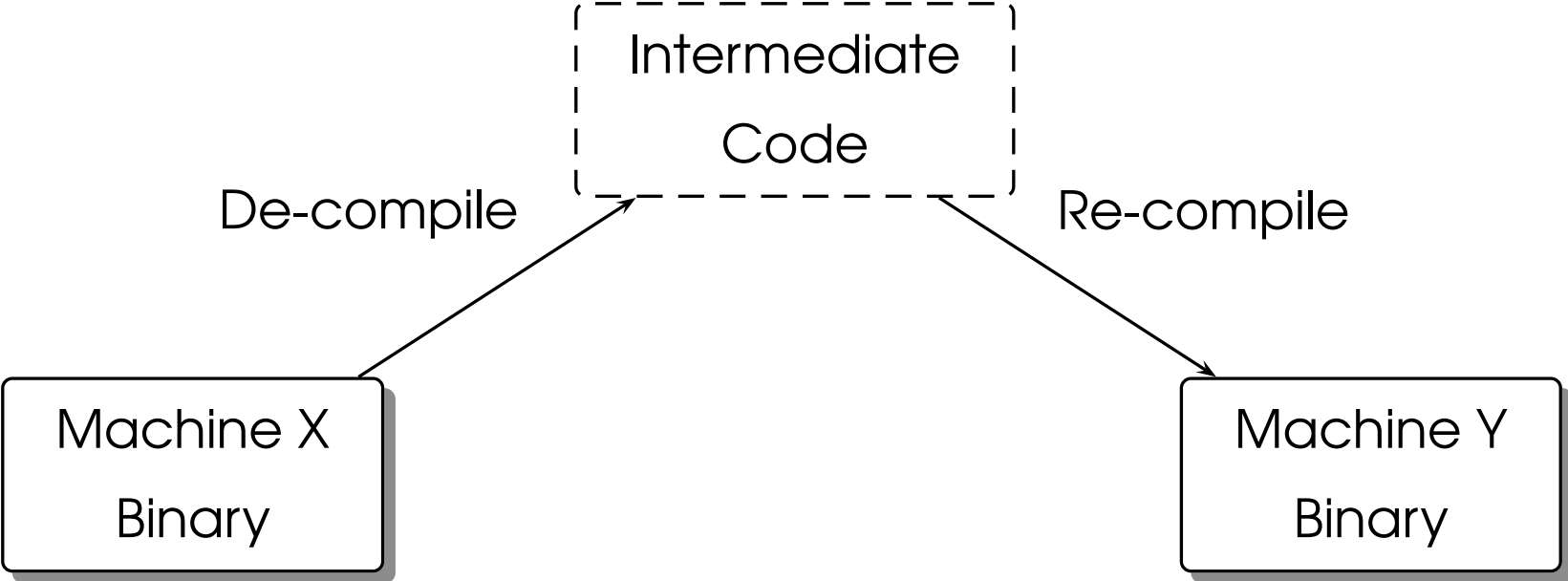
BINARY TRANSLATION

The ability to run code written for one architecture on another without access to source code

Applications

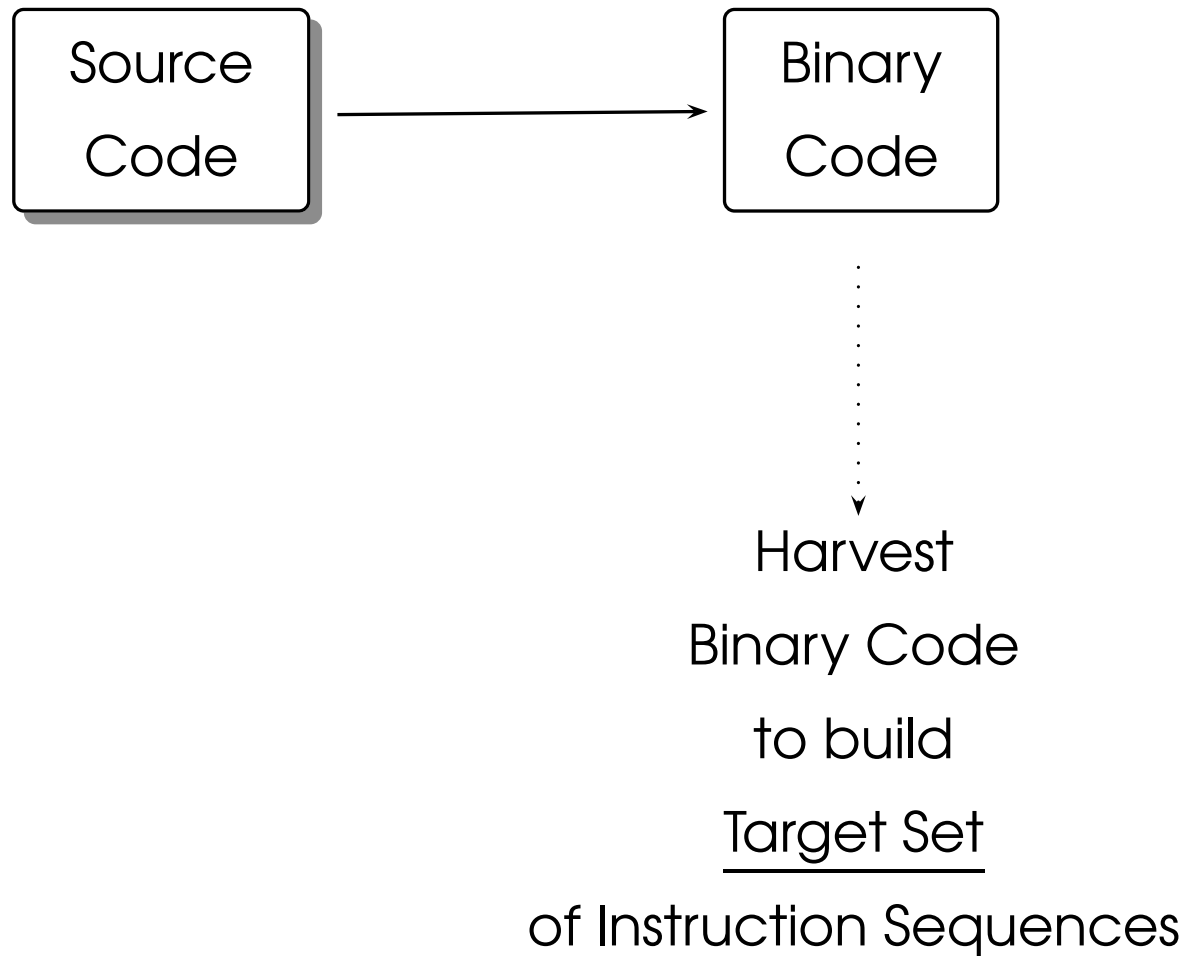
- Portability of Applications (eg. Apple)
- Backward/Forward Compatibility of Architecture Generations (eg. Intel)
- Virtualization (eg. IBM's PAVE)
- Get the "Best-of-Both-Worlds" (eg. iTunes on x86/linux)
- Running Legacy Code
- Instrumentation (Machine X → Machine X)

BINARY TRANSLATION

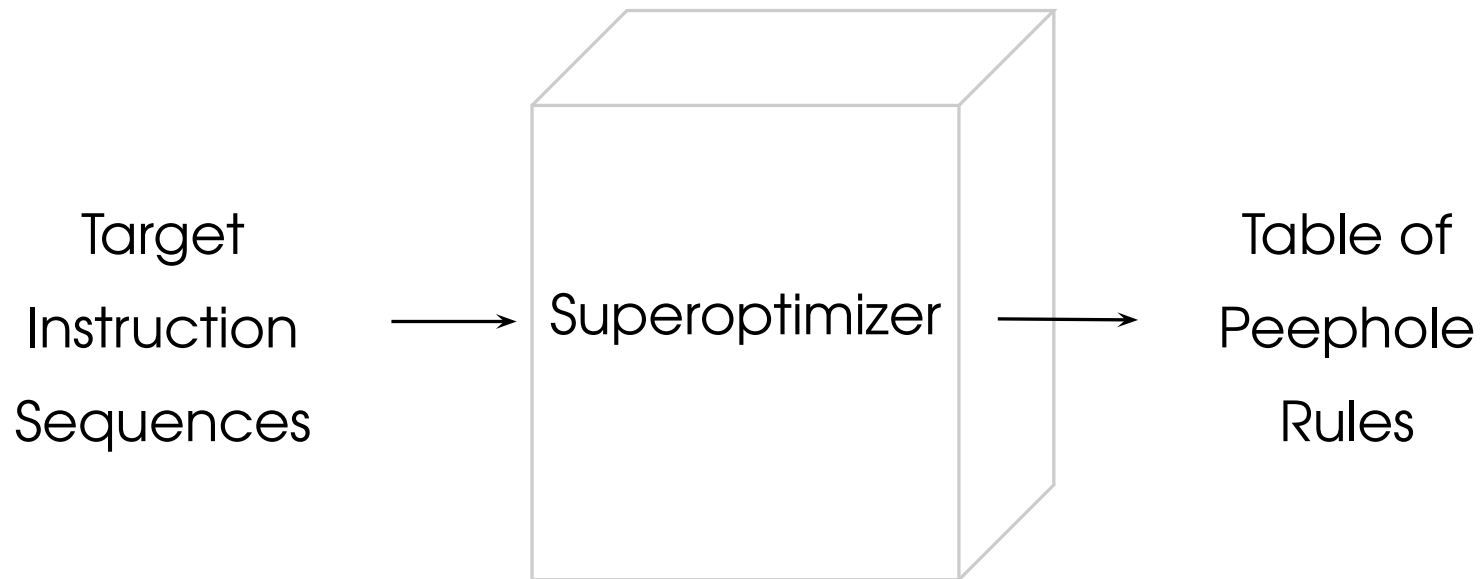


PEEPHOLE SUPEROPTIMIZERS

PEEPHOLE SUPEROPTIMIZERS



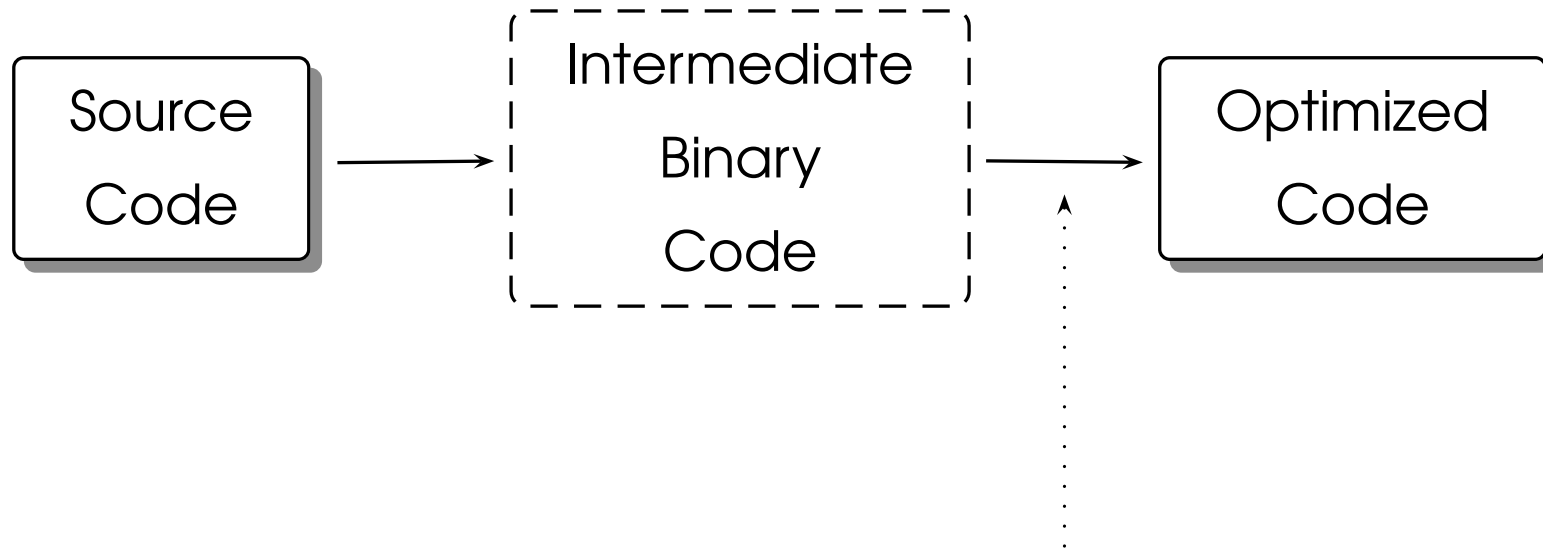
PEEPHOLE SUPEROPTIMIZERS



Peephole Rules

Input	Output
<code>mov %esp, %ebp</code> <code>mov %ebp, %esp</code>	<code>mov %esp, %ebp</code>

PEEPHOLE SUPEROPTIMIZERS

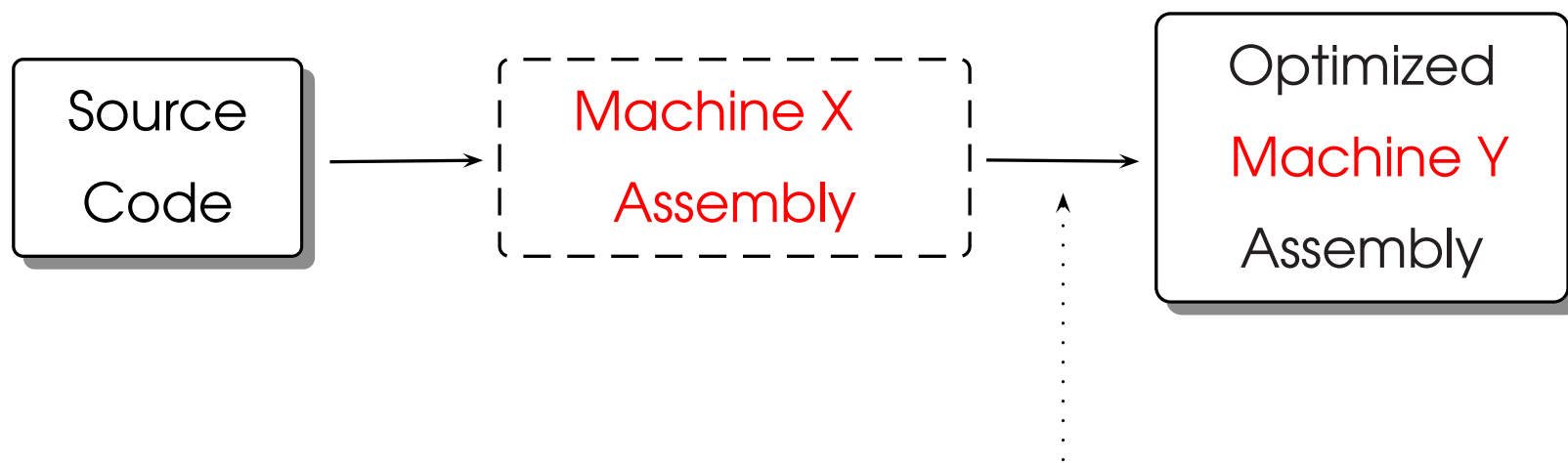


Apply Peephole Rules

Peephole Rules

Input	Output
<code>mov %esp, %ebp</code> <code>mov %ebp, %esp</code>	<code>mov %esp, %ebp</code>

PEEPHOLE TRANSLATORS

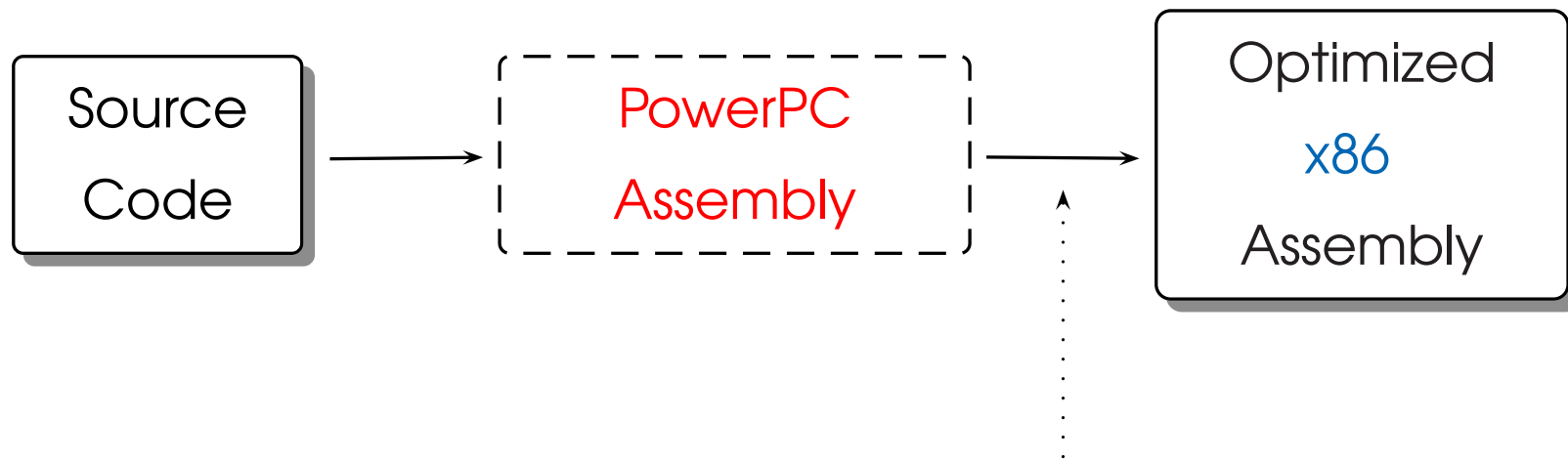


Apply Peephole Rules

Peephole Rules

Input	Map	Output
Machine X Assembly	$rx_1 \rightarrow ry_1$ $rx_2 \rightarrow ry_2$	Machine Y Assembly

PEEPHOLE TRANSLATORS



Apply Peephole Rules

Peephole Rules

Input	Map	Output
<code>mr r1, r2</code> <code>mr r2, r1</code>	<code>r1 → %ebp</code> <code>r2 → %esp</code>	<code>mov %esp, %ebp</code>

CHALLENGES

CHALLENGES

- Register Mapping
 - Dynamic/Static?

CHALLENGES

- Register Mapping
 - Dynamic/Static?
- Control Flow Transfers
 - Direct/Indirect Jumps
 - Function Call/Return

CHALLENGES

- Register Mapping
 - Dynamic/Static?
- Control Flow Transfers
 - Direct/Indirect Jumps
 - Function Call/Return
- Endian-ness

REGISTER MAPPING

PowerPC : 32 GPRs, 3 SPRs, 8 flag registers

x86 : 8 GPRs and other SIMD-support registers

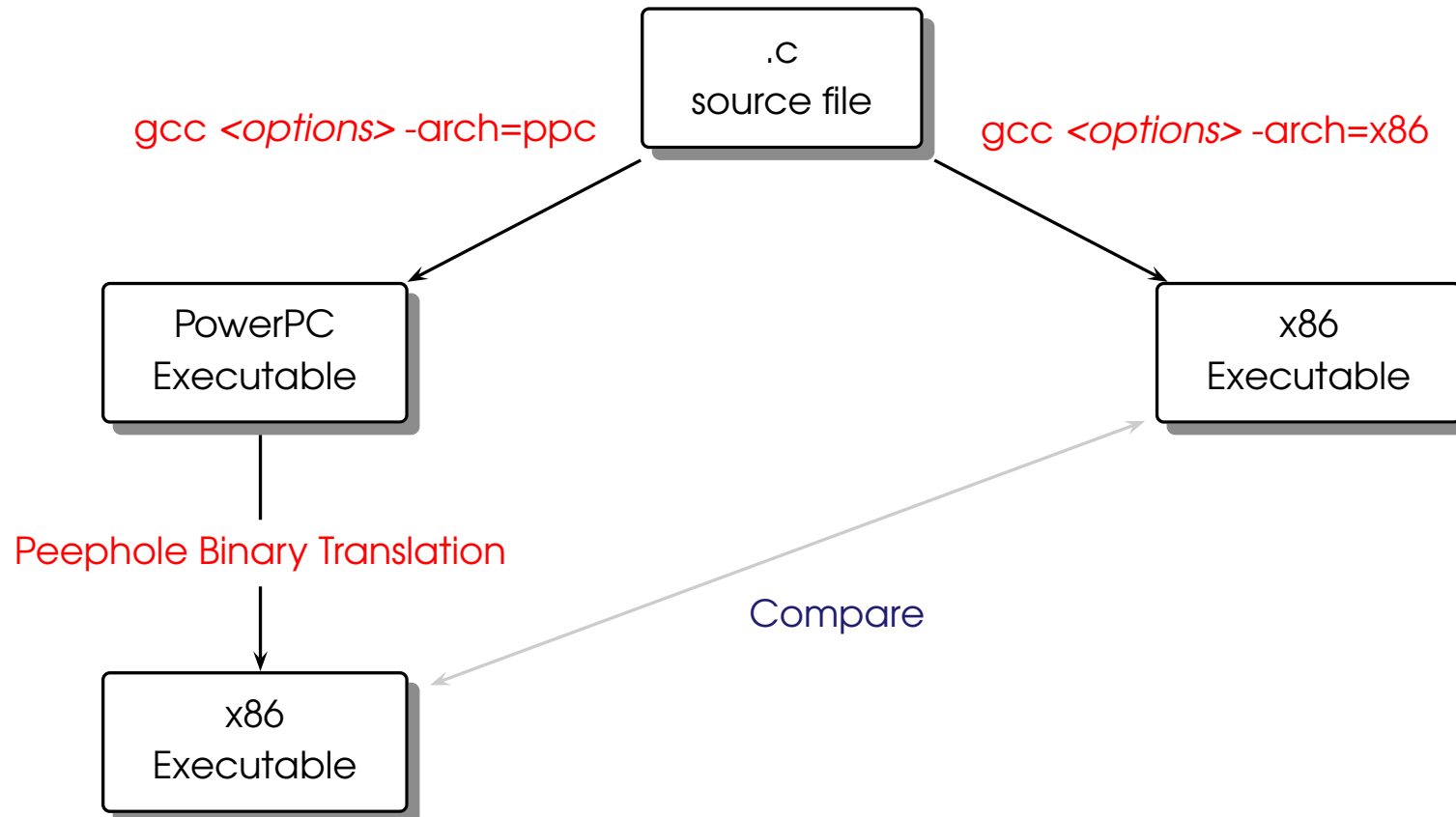
REGISTER MAPPING

Dynamic Register Mapping

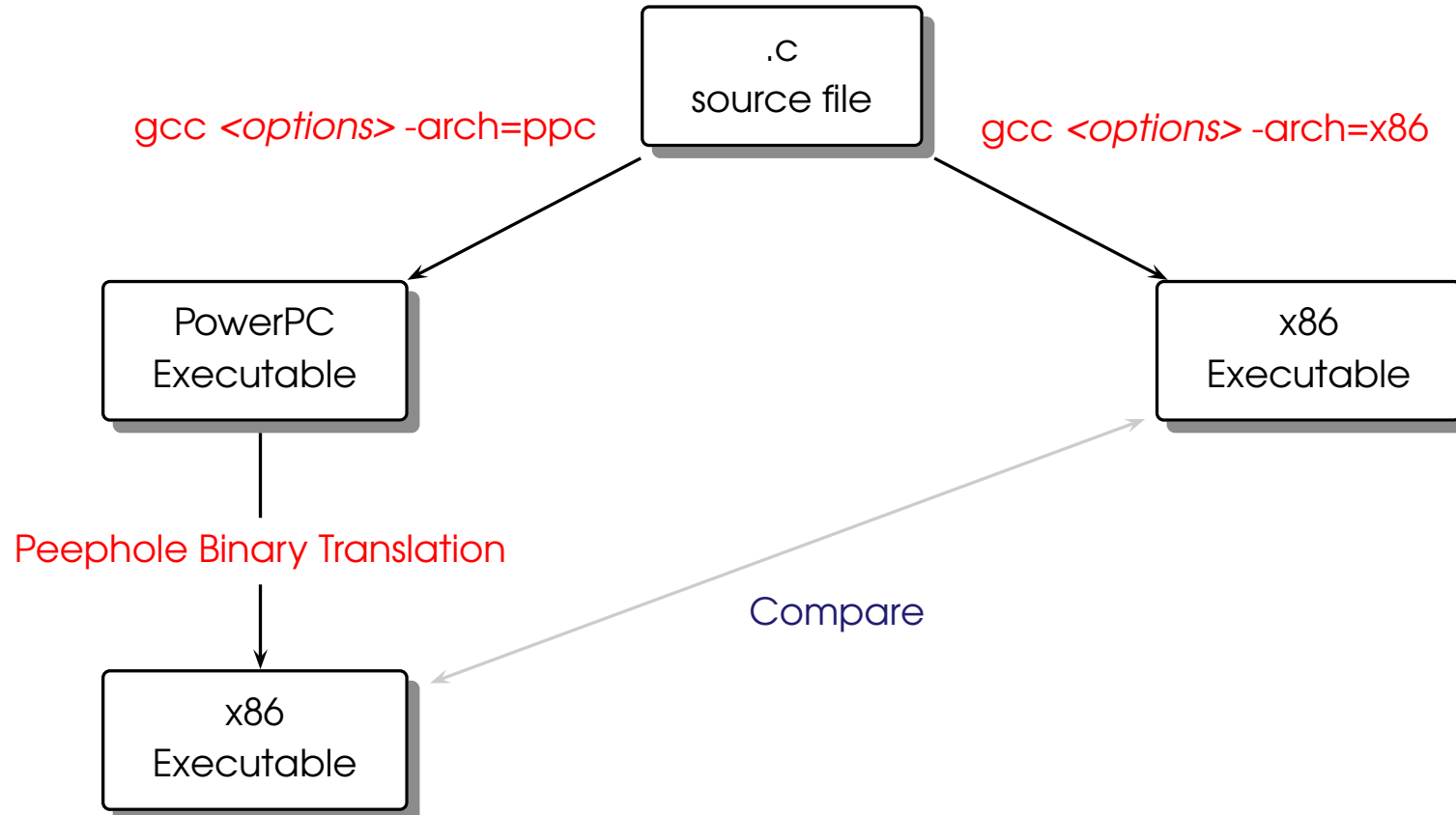
- Use dynamic programming considering all possible register maps. Account for
 - Switching cost
 - Cost of the peephole rule

- Make decisions at function return points

EXPERIMENTAL SETUP



EXPERIMENTAL SETUP



Qemu (Freely available)	:	10-20%
Apple Rosetta (Commercial)	:	70-80% (claim)
Transitive (Commercial)	:	70-80% (claim)

BENCHMARKS

Name	Description
fibonacci64	Compute the first n fibonacci numbers
bubblesort64	Bubble-sort a large array of 64-bit integers
hanoi1	Kolar's Hanoi Tower algorithm no. 1
hanoi2	Er's LLHanoi Hanoi Tower loop less algorithm
hanoi3	Kolar's Hanoi Tower algorithm no. 3

RESULTS

fibonacci64			
bubblesort64			
hanoi1			
hanoi2			
hanoi3			

RESULTS

	○○		
fibonacci64	110%		
bubblesort64	84%		
hanoi1	97%		
hanoi2	103%		
hanoi3	90%		

RESULTS

	O0	O2	
fibonacci64	110%	311%	
bubblesort64	84%	55%	
hanoi1	97%	59%	
hanoi2	103%	159%	
hanoi3	90%	90.5%	

RESULTS

	O0	O2	O2+
fibonacci64	110%	311%	121%
bubblesort64	84%	55%	61.4%
hanoi1	97%	59%	59%
hanoi2	103%	159%	158%
hanoi3	90%	90.5%	67.2%

O2+ = -O2 -fomit-frame-pointer

RESULTS

	O0	O2	O2+
fibonacci64	110%	311%	121%
bubblesort64	84%	<u>55%</u>	<u>61.4%</u>
hanoi1	97%	<u>59%</u>	<u>59%</u>
hanoi2	103%	159%	158%
hanoi3	90%	90.5%	<u>67.2%</u>

O2+ = -O2 -fomit-frame-pointer

RESULTS

	O0	O2	O2+
fibonacci64	110%	(311%)	(121%)
bubblesort64	84%	<u>55%</u>	<u>61.4%</u>
hanoi1	97%	<u>59%</u>	<u>59%</u>
hanoi2	103%	(159%)	(158%)
hanoi3	90%	90.5%	<u>67.2%</u>

O2+ = -O2 -fomit-frame-pointer

CONCLUSIONS AND FUTURE WORK

Conclusions

- Peephole Superoptimization is a good fit for the problem of Binary Translation

Ongoing Work

- Improve performance
- Support all system calls (run larger benchmarks)
- Look at other source-target architecture pairs