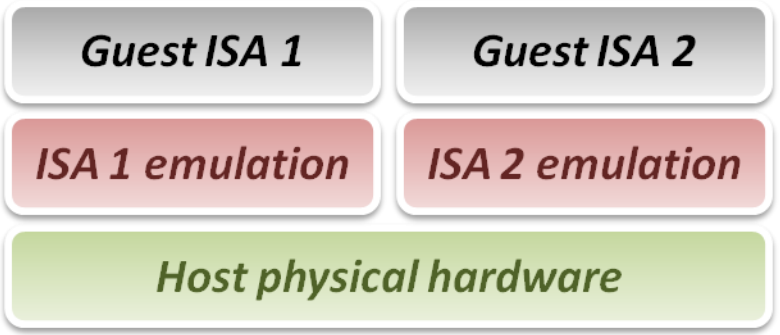
Emulation Technique

- Why do we talk about emulation ?
  - In fact, virtualization technique can be treated as a special case of emulation technique.
  - Many virtualization techniques were developed in or inherited from emulation technique.

- Goal of emulation :
  - Provide a method for enabling a (sub)system to present the same interface and characteristics as another.



Emulation Technique

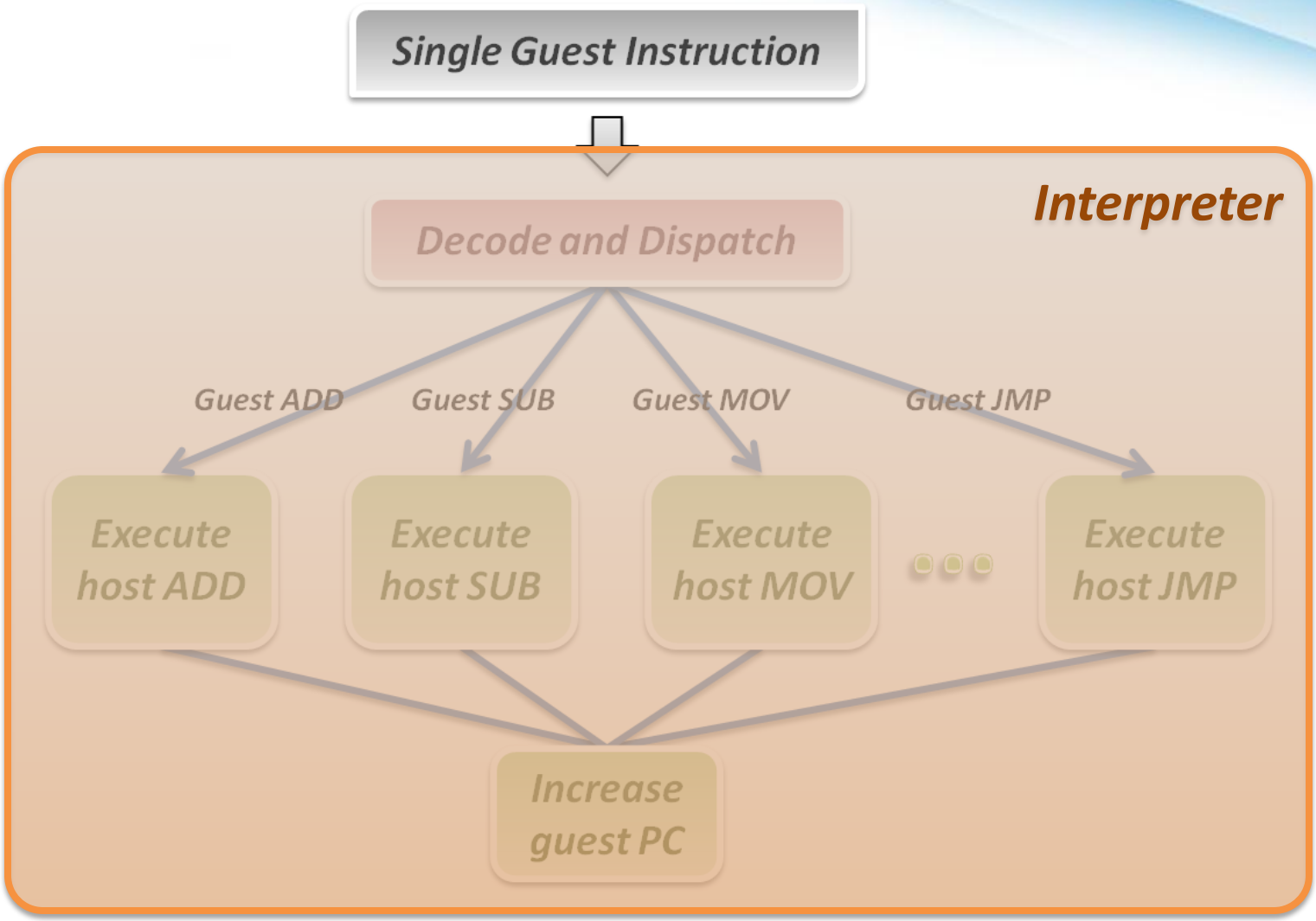
- Three emulation implementations :
  - Interpretation
    - Emulator interprets only one instruction at a time.
  - Static Binary Translation
    - Emulator translates a block of guest binary at a time and further optimizes for repeated instruction executions.
  - Dynamic Binary Translation
    - This is a hybrid approach of emulator, which mix two approaches above.
- Design challenges and issues :
  - Register mapping problem
  - Performance improvement

Interpretation

- Interpreter execution flow :
  1. Fetch one guest instruction from guest memory image.
  2. Decode and dispatch to corresponding emulation unit.
  3. Execute the functionality of that instruction and modify some related system states, such as simulated register values.
  4. Increase the guest PC (Program Counter register) and then repeat this process again.

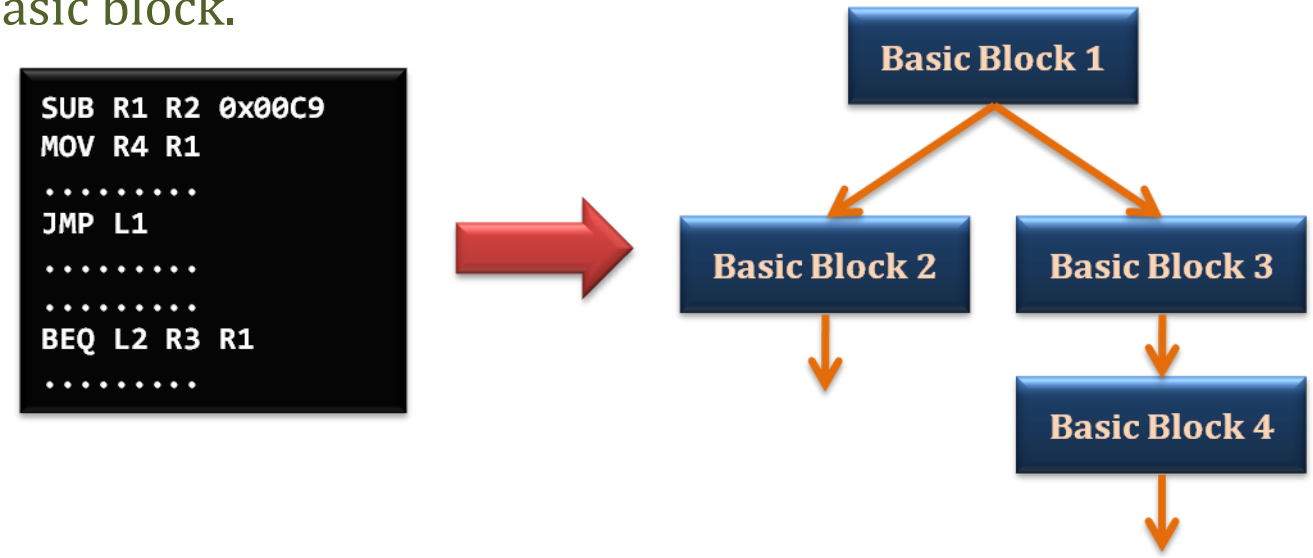
- Pros & Cons
  - Pros
    - Easy to implement
  - Cons
    - Poor performance

Interpretation



Static Binary Translation

- Using the concept of basic block which comes from compiler optimization technique.
  - A basic block is a portion of the code within a program with certain desirable properties that make it highly amenable to analysis.
  - A basic block has only one entry point, meaning no code within it is the destination of a jump instruction anywhere in the program.
  - A basic block has only one exit point, meaning only the last instruction can cause the program to begin executing code in a different basic block.

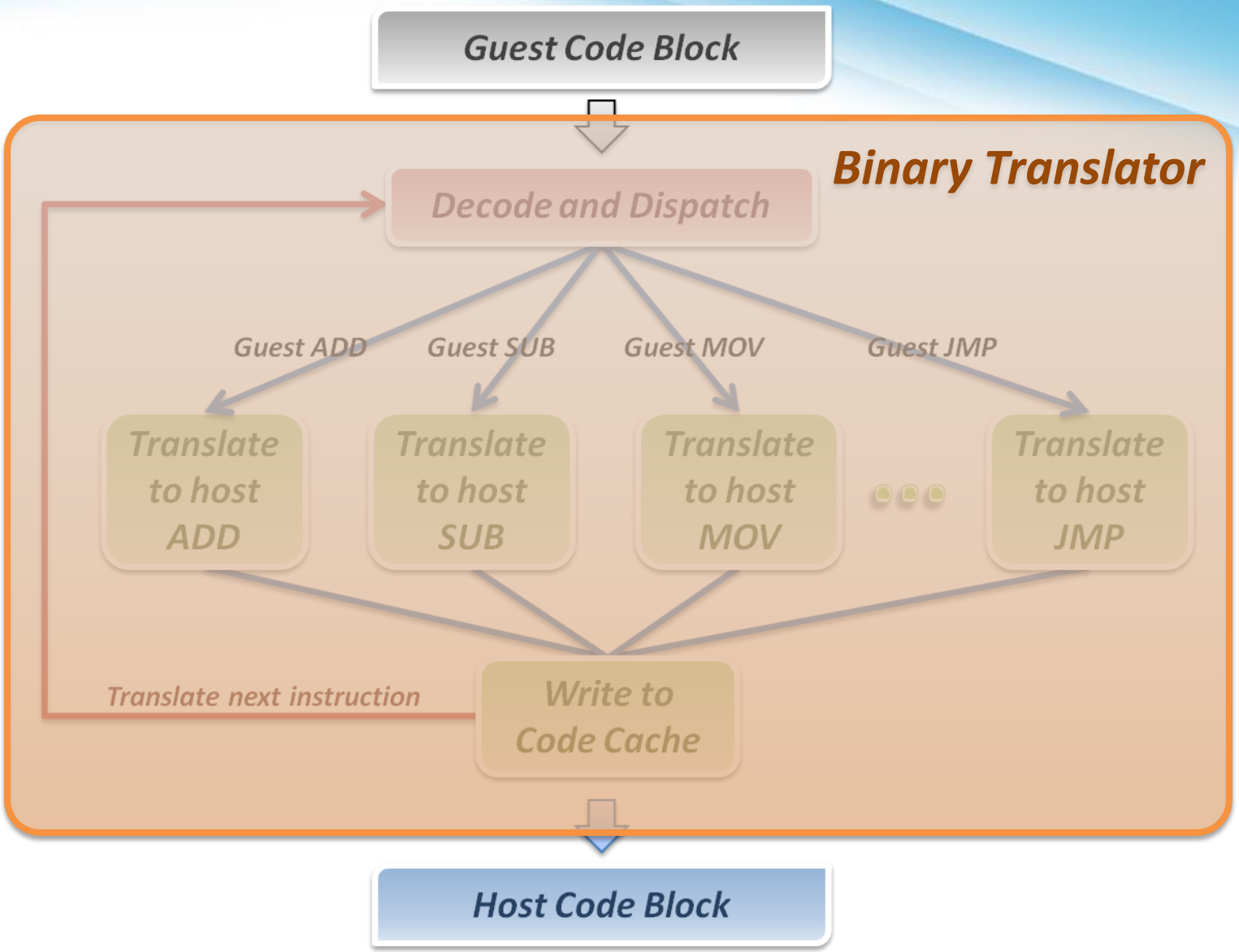


Static Binary Translation

- Static binary translation flow :
  1. Fetch one block of guest instructions from guest memory image.
  2. Decode and dispatch each instruction to the corresponding translation unit.
  3. Translate guest instruction to host instructions.
  4. Write the translated host instructions to code cache.
  5. Execute the translated host instruction block in code cache.

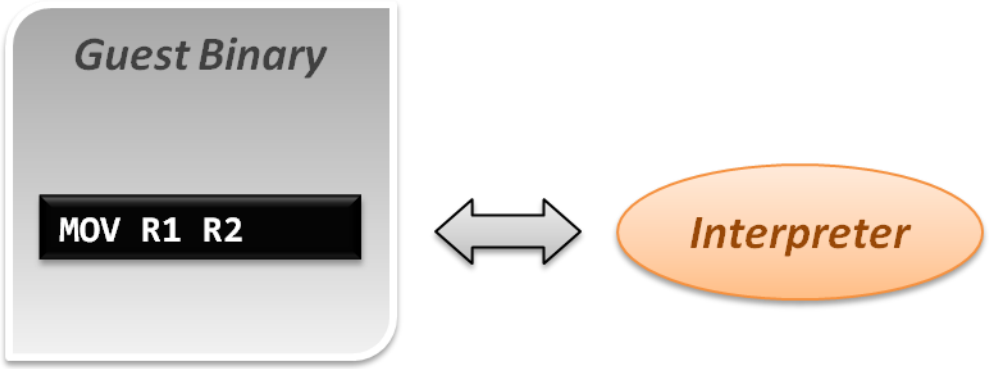
- Pros & Cons
  - Pros
    - Emulator can reuse the translated host code.
    - Emulator can apply more optimization when translating guest blocks.
  - Cons
    - Implementation complexity will increase.

Binary Translation

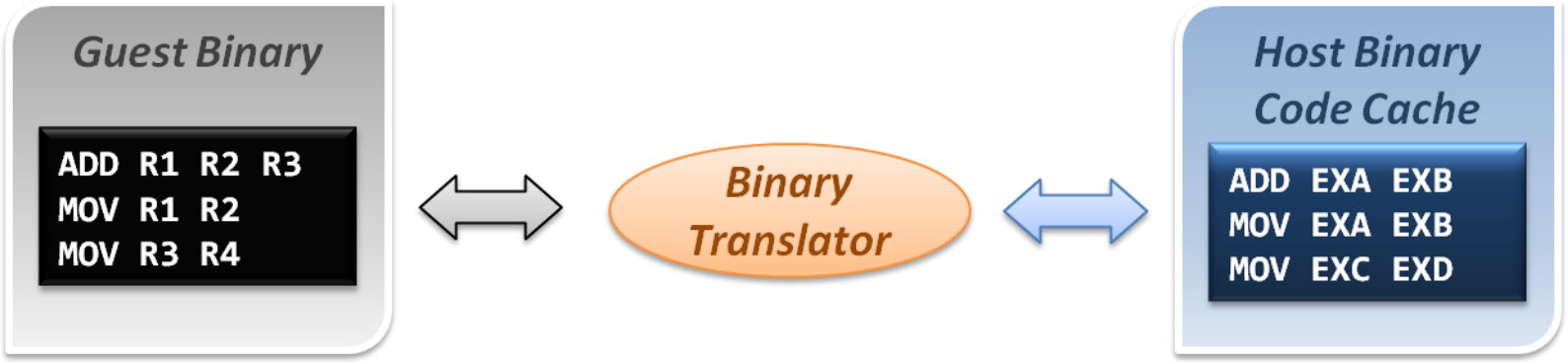


Comparison

- Interpretation implementation



- Static binary translation implementation

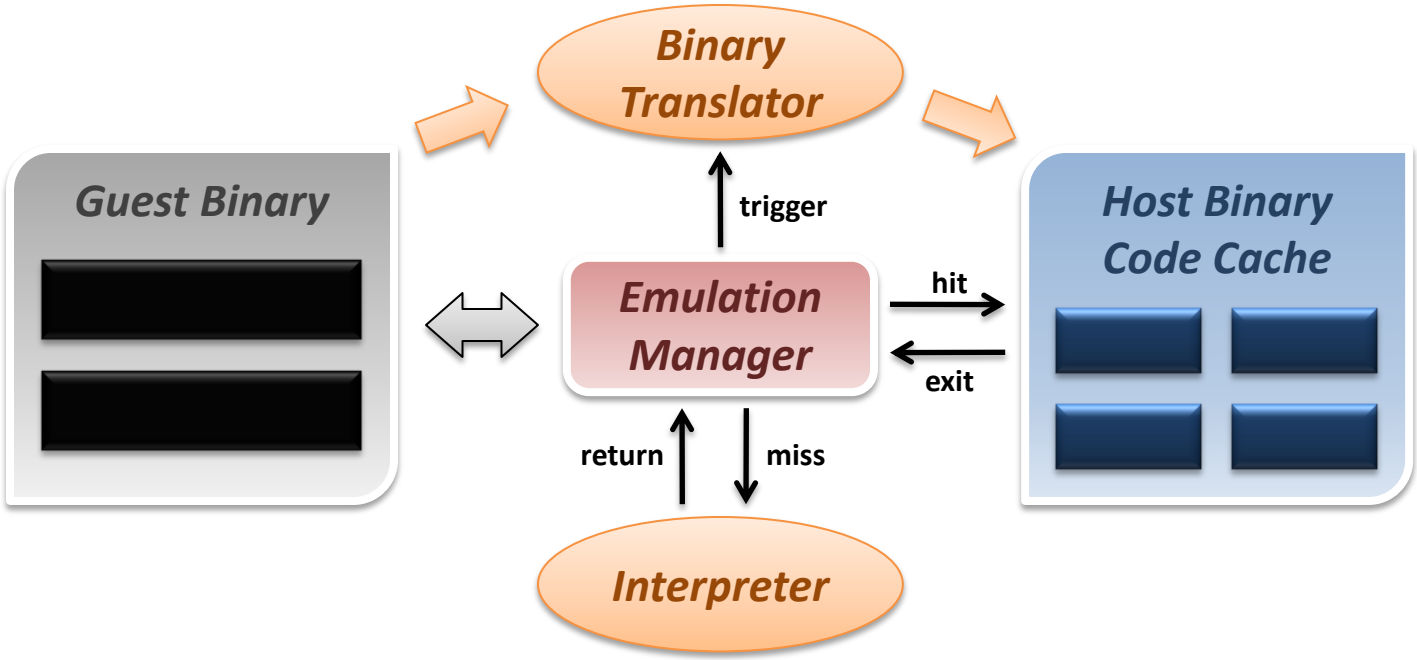


Dynamic Binary Translation

- A hybrid implementation
  - For the first discovered codes, directly interpret by interpreter and record these codes as discovered.
  - As the guest codes discovered, trigger the binary translation module to translate the guest code blocks to host code blocks, and place them into code cache.
  - When execute the translated block of guest code next time, jump to the code cache and execute the translated host binary code.
- Pros & Cons
  - Pros
    - Transparently implement binary translation.
  - Cons
    - Hard to implement.

Dynamic Binary Translation

1. First time execution, no translated code in code cache.
2. Miss code cache matching, then directly interpret the guest instruction.
3. As a code block discovered, trigger the binary translation module.
4. Translate guest code block to host binary, and place it in the code cache.
5. Next time execution, run the translated code clock in the code cache.



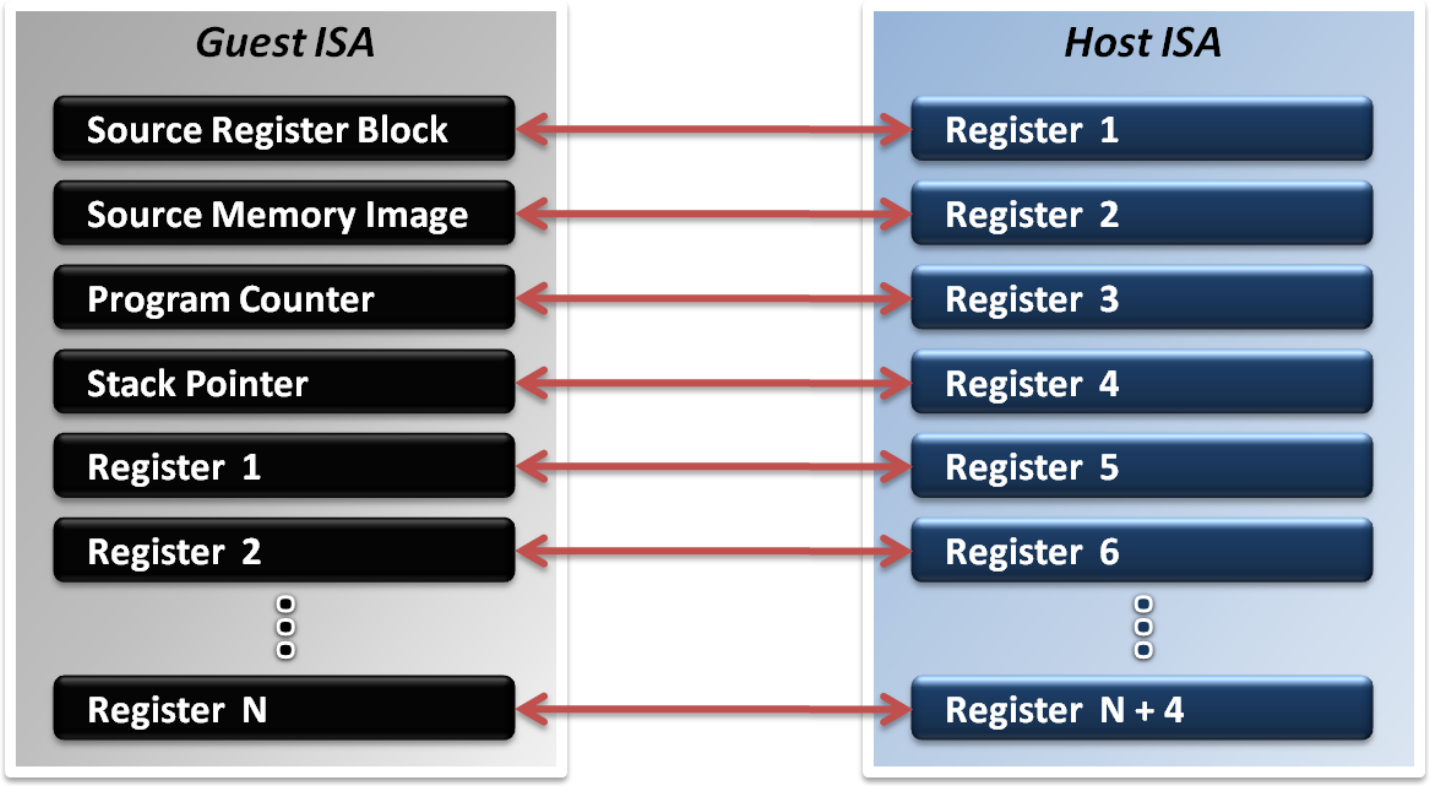
Register mapping problem

Performance improvement

Design challenges and issues

Register Mapping Problem

- Why should we map registers ?
  - Different ISA will define different number of registers.
  - Sometimes guest ISA even require some special purpose register which host ISA does not defined.



Register Mapping Problem

- Mapping different purpose of registers :
  - Map special purpose registers
    - Program Counter Register
    - Stack Pointer Register
    - Page Table Register
    - System Statues Register
    - Special Flags Register
  - Hold guest context and memory image
  - Map general purpose registers
  - Map intermediate values

Register Mapping Problem

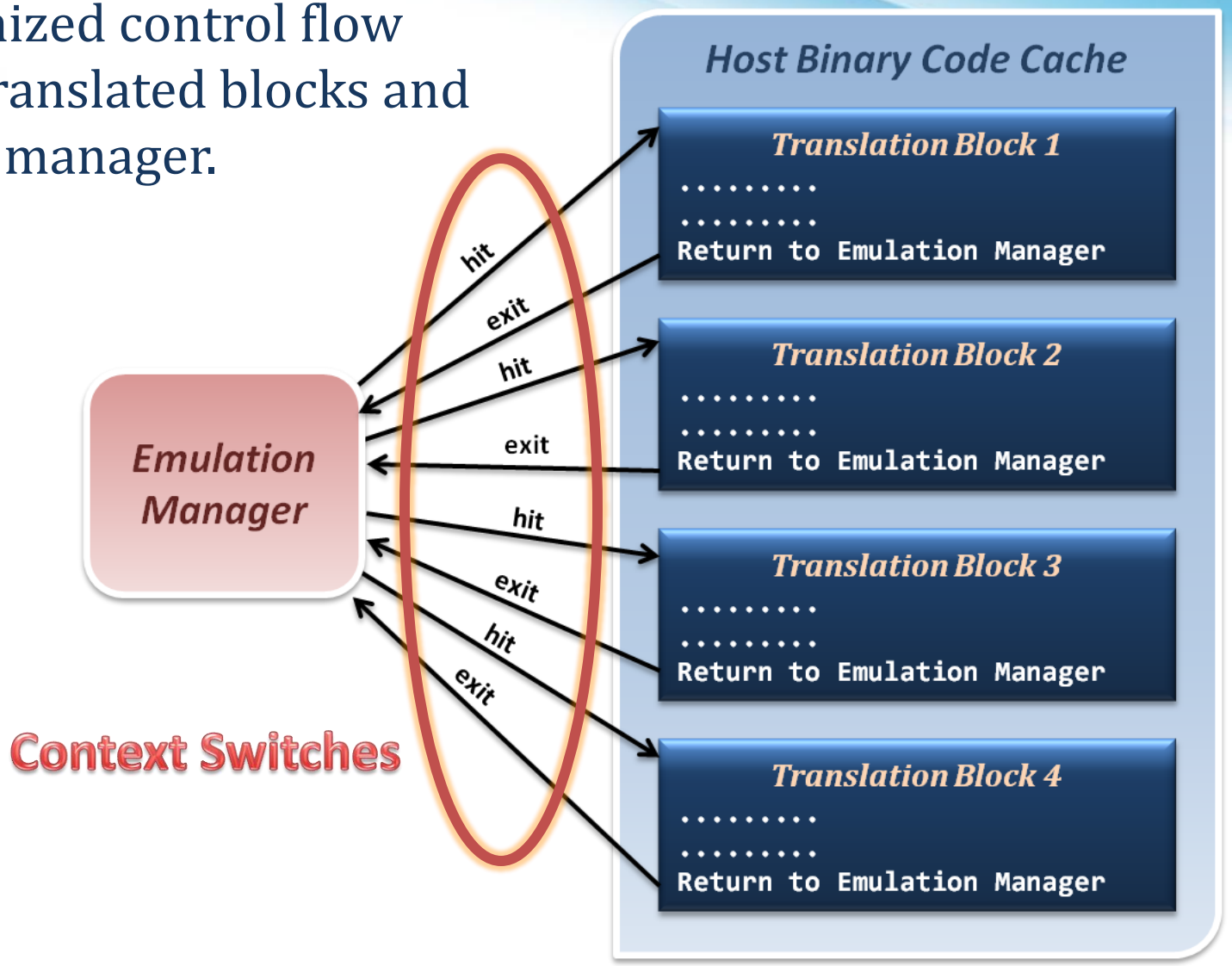
- If number of host registers is larger the guest
  - That will be an easier case for implementation.
  - Directly map one register of guest to one of host, and make use of the rest registers for optimization.
  - Example :
    - Translating x86 binary to RISC
- If number of host registers is not enough
  - That should involve more effort.
  - Emulator may only map some frequently used guest registers to host, and left the unmapped registers in memory.
  - Mapping decision will be critical in this case.

Performance Improvement

- What introduces the performance hit ?
  - Control flow problem
    - Highly frequent context switches between code caches and emulation manager will degrade performance.
  - Target code optimization
    - Translate guest code block in instruction-wise (translate one instruction at a time) will miss many optimization opportunities.
- Solutions :
  - Translation Chaining
  - Dynamic Optimization

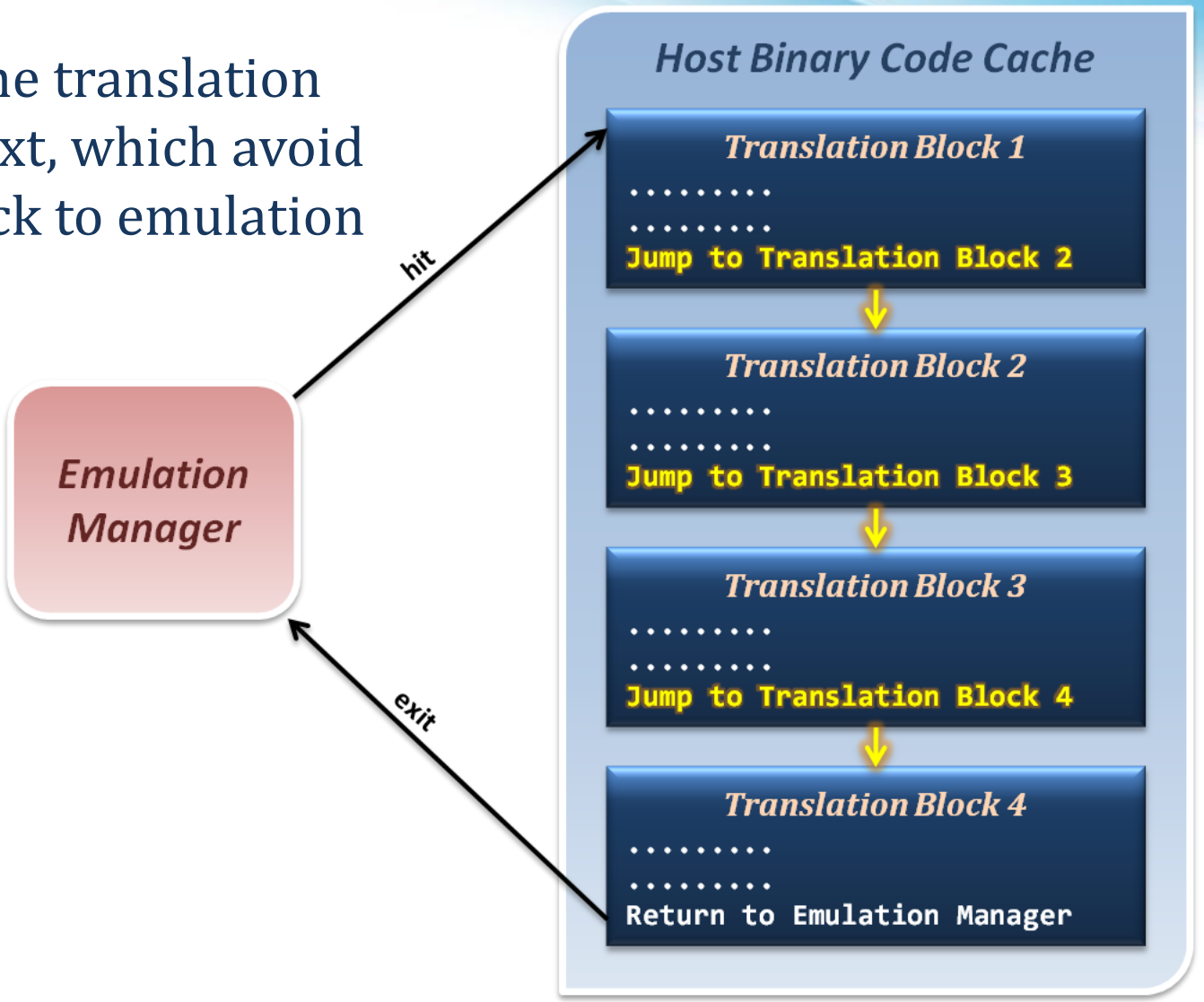
Translation Chaining

- Non-optimized control flow between translated blocks and emulation manager.



Translation Chaining

- Jump from one translation directly to next, which avoid switching back to emulation manager.



Dynamic Optimization

- How to optimize binary codes ?
  - Static optimization (compiling time optimization)
    - Optimization techniques apply to generate binary code base on the semantic information in source code.
  - Dynamic optimization (run time optimization)
    - Optimization techniques apply to generated binary code base on the run time information which relate to program input data.
- Why we use dynamic optimization technique ?
  - Advantages :
    - It can benefit from dynamic profiling.
    - It is not constrained by a compilation unit.
    - It knows the exact execution environment.

Dynamic Optimization

- How to implement dynamic optimization ?
  - Analysis program behavior in run time.
  - Collect run time profiling information based on the input data and host hardware characteristics.
  - Dynamically translate or modify the binary code by reordering instructions or other techniques.
  - Write back the optimized binary into code cache for next execution.

Dynamic Optimization

- How to analyze program behavior and profile ?
  - Collect statistics about a program as it runs
    - Branches (taken, not taken)
    - Jump targets
    - Data values
    - Cache misses
  - Predictability allows these statistics to be used for optimizations to be used in the future
- Profiling in a VM differs from traditional profiling used for compiler feedback.

Dynamic Optimization

- Dynamic binary translation and optimization :