Binary Translation

Redha Gouicem

Towards a heterogeneous CPU landscape

New emerging architectures challenge the x86 dominance

Porting legacy software is not always possible:

- Source code unavailable
- x86-specific assembly code

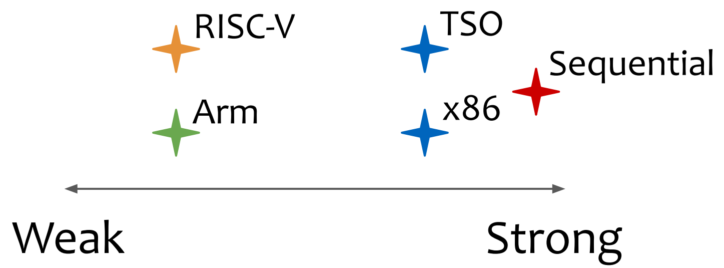
Executing legacy applications on new architectures without porting them is made possible with **binary translation**

Architectures have different memory semantics

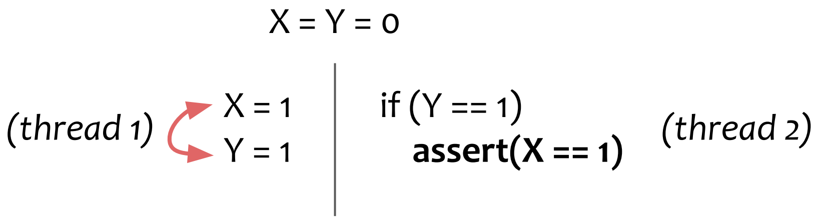
Weak memory model architectures

CPUs can re-order memory accesses

The **weaker** the model, the **more re-orderings** are allowed



When translating from **strong to weak** models, new behaviors can appear



```
X = Y = 0

(thread 1)  X = 1    |  if (Y == 1)             (thread 2)
            Y = 1    |     assert(X == 1)
```

On x86, assert **always succeeds**

On Arm, assert **can fail** if *thread 1*'s instructions are re-ordered

The source memory model must be enforced

Enforcing a stronger memory model

In hardware:

For example, Apple implements both Arm and x86 models in their new M1 chips

- ✓ Efficient
- ✗ Only possible if you have control over the hardware

In software:

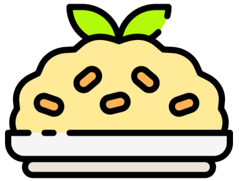
Add memory fences around every memory access

- ✓ Works on any off-the-shelf CPU
- ✗ Hard to correctly match the source model
- ✗ Heavy performance cost (roughly doubles execution time)

How can we efficiently and correctly enforce a memory model?

Binary translation for weak memory models

Lasagne
A static binary translator
[PLDI'22]



Risotto
A dynamic binary translator
[ASPLOS'23]

Credits: Flaticon.com

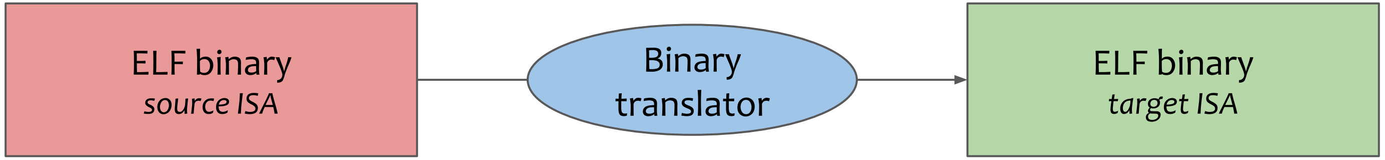
Static binary translation

Translate all the reachable instructions in a source binary **ahead-of-time**

Generate a new executable ELF binary for the target ISA

Usually:

- Parse the ELF symbol table
- Follow direct jumps
- Detect instructions in non-reached code



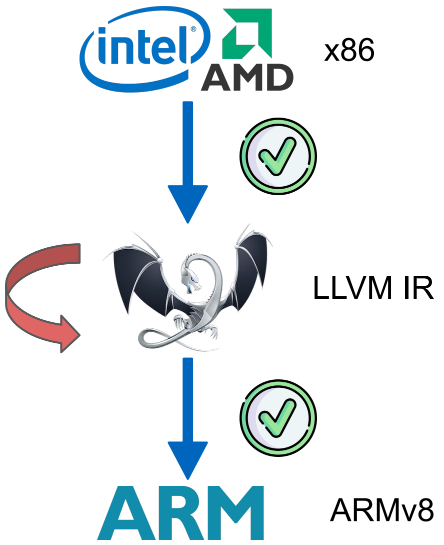
Lasagne: a static binary translator

Statically lift x86 binaries to Arm, using the LLVM IR

Formally verified mappings for memory operations ✓

Implemented in **Microsoft mctoll**

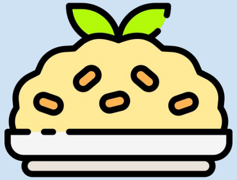
Function type discovery and peephole optimisations



Published at [PLDI'22]
Open source, **77 commits** merged into mctoll

Binary translation for weak memory models

Lasagne
A static binary translator
[PLDI'22]



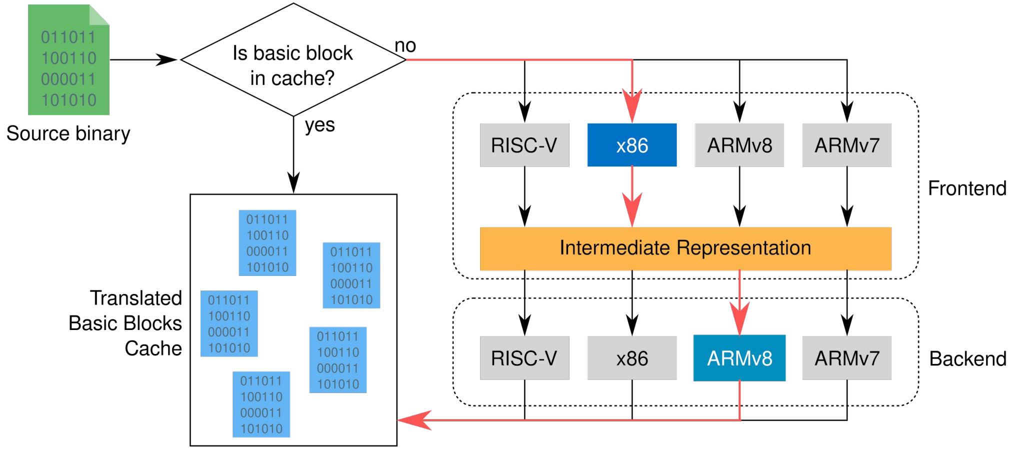
Risotto
A dynamic binary translator
[ASPLOS'23]

Credits: Flaticon.com

Dynamic binary translation

Execute the binary and translate the reached instructions **just-in-time**

The runtime is responsible for switching between execution and translation modes



Risotto: a dynamic binary translator

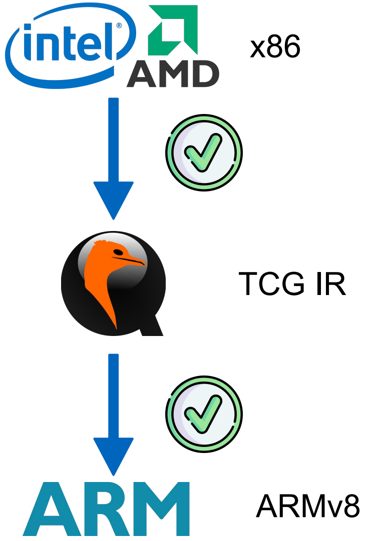
Dynamically translate x86 to Arm via TCG IR

Formally verified mappings for memory operations
Fix proposal for the Arm memory model ✓

Implemented in QEMU

Dynamic host linker for shared libraries + native CAS translation

Published at [ASPLOS'23], artifact evaluation incoming
Open source, patches under review in **QEMU**
Fix to **Arm official memory model** merged



Risotto: Dynamic host linker

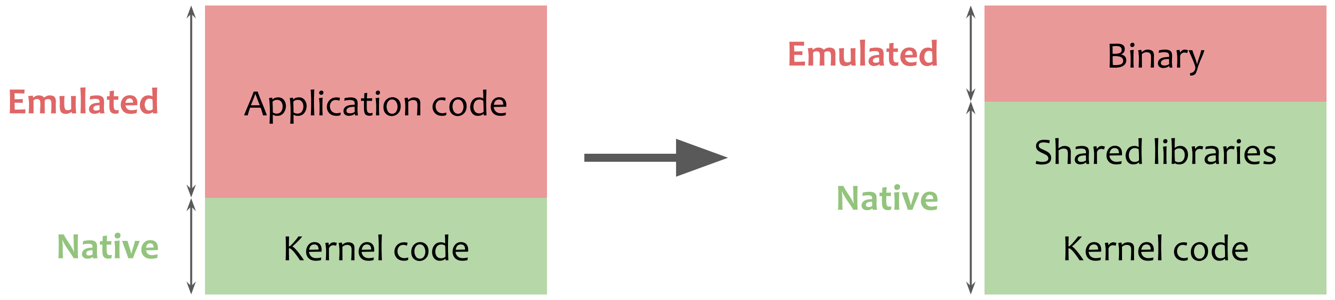
We work in user-mode emulation

i.e., the kernel is not emulated

System calls are the interface between native and emulated code

We can look at shared libraries similarly:

- available in native optimised version
- have a well-defined API



Risotto: Dynamic host linker (2)

At initialisation time:

1. Load user-provided signature *Interface Definition Language* file (IDL)
2. Load the ELF and detect entries in the *Procedure Linkage Table* (PLT)

At run time:

3. If the function is not in the IDL file, translate the function from source ISA
4. If the function is in the IDL file, generate marshalling code and call the native shared library function

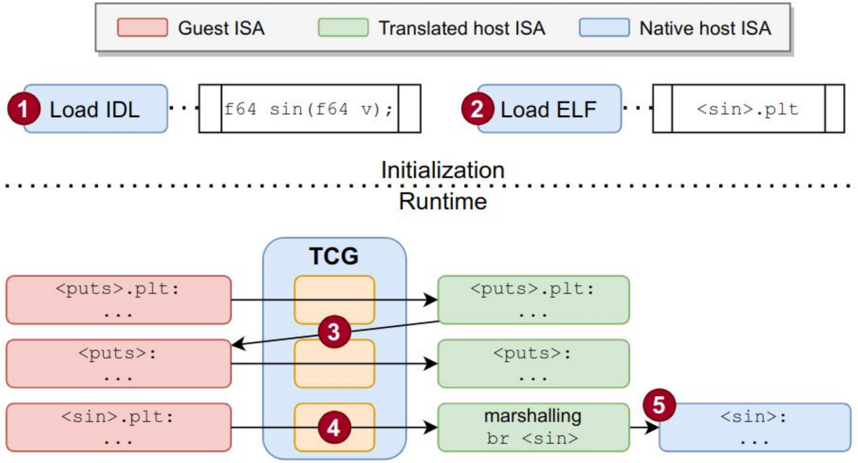
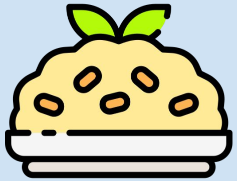


Figure 11: Risotto's dynamic linker workflow.

Binary translation for weak memory models

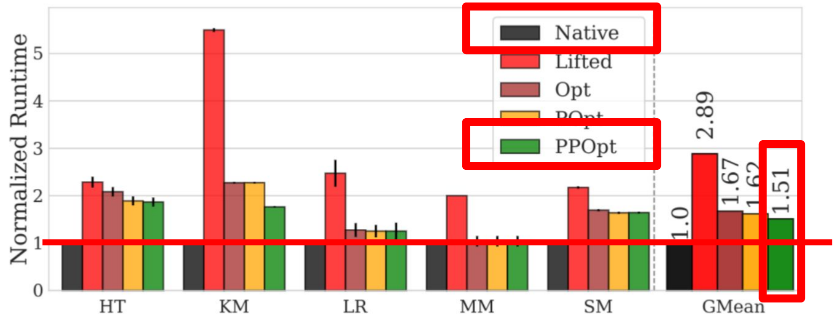
Lasagne
A static binary translator
[PLDI’22]



Risotto
A dynamic binary translator
[ASPLOS’23]

Credits: Flaticon.com

Lasagne: Evaluation



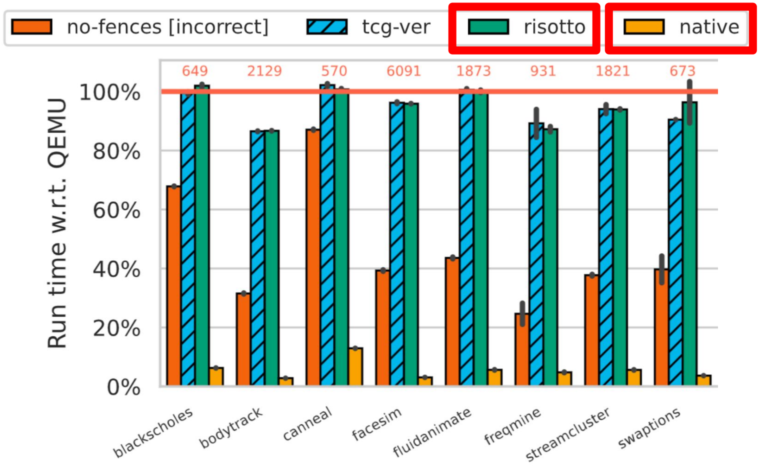
Performance of Lasagne normalised to native Arm execution for the Phoenix 2.0 benchmark suite

Performance is decent when compared to a natively compiled binary for Arm, thanks to our optimisations→ **1.5x slowdown on average**

Static translation has intrinsic problems:

- x86 **cannot be 100% statically translated**
- No support for **dynamic code loading**

Risotto: Evaluation



Performance of Risotto normalised to QEMU for the PARSEC benchmark suite

Good performance w.r.t. QEMU→ **6.7% improvement on average**

The performance overhead is not acceptable in practice →**12x slower than native**

Limits of Lasagne and Risotto

	Lasagne (static)	Risotto (dynamic)	Hybrid?
Performance	✔	✗	✔
Coverage	✗	✔	✔

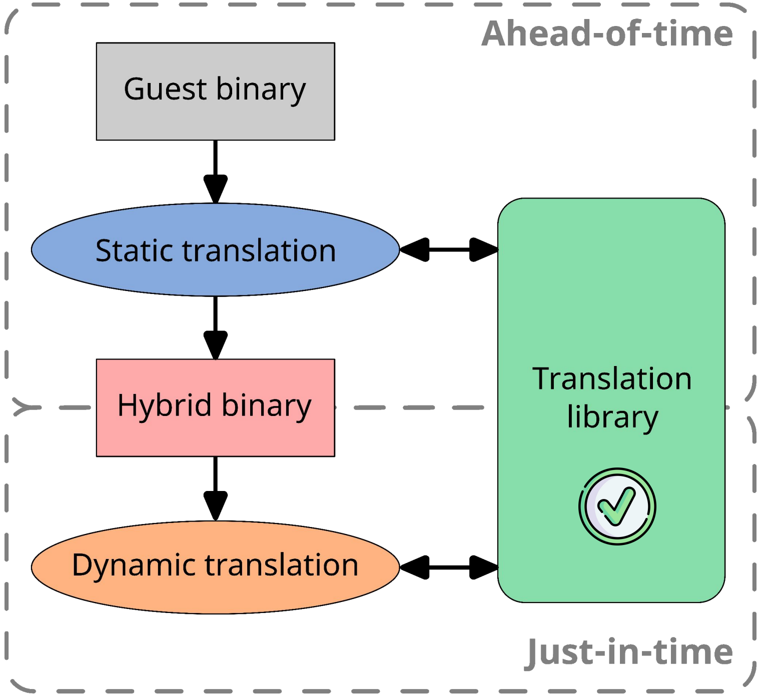
We need a hybrid translator to get the benefits of static and dynamic translation

Arancini: A hybrid binary translator

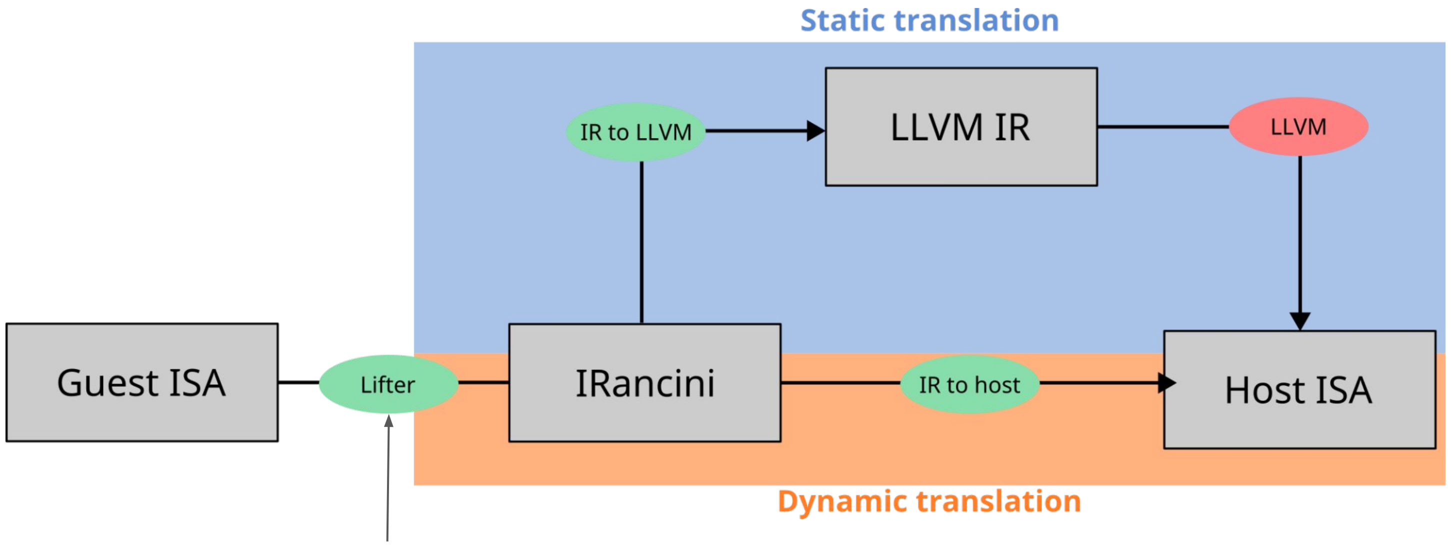
Ahead-of-time:
- Static translation
- Slow but heavily optimised
- Translate all reachable code from the symbol table and direct jumps

Just-in-time:
- Dynamic translation
- Fast with minor optimisation
- Fill the gaps if needed

Still with formally verified mappings ✓



Overview of the translation process



We use Intel xed for instruction decoding

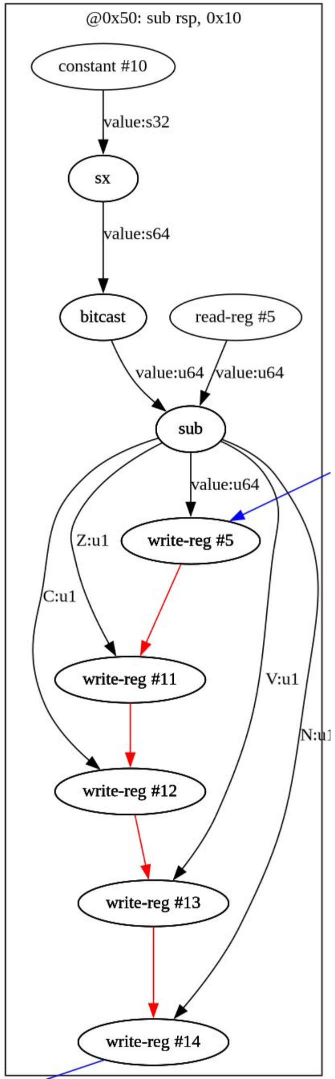
IRancini: Translating an instruction

Each guest instruction is translated as a *packet* containing nodes that represent simple operations

Example: sub rsp, 0x10 → RSP := RSP - 0x10

Read RSP register, subtract 0x10 from it, and write the result back to RSP

- Read the inputs and type them
- Do the subtraction
- Write result and set flags



IRancini: Representing the instruction flow

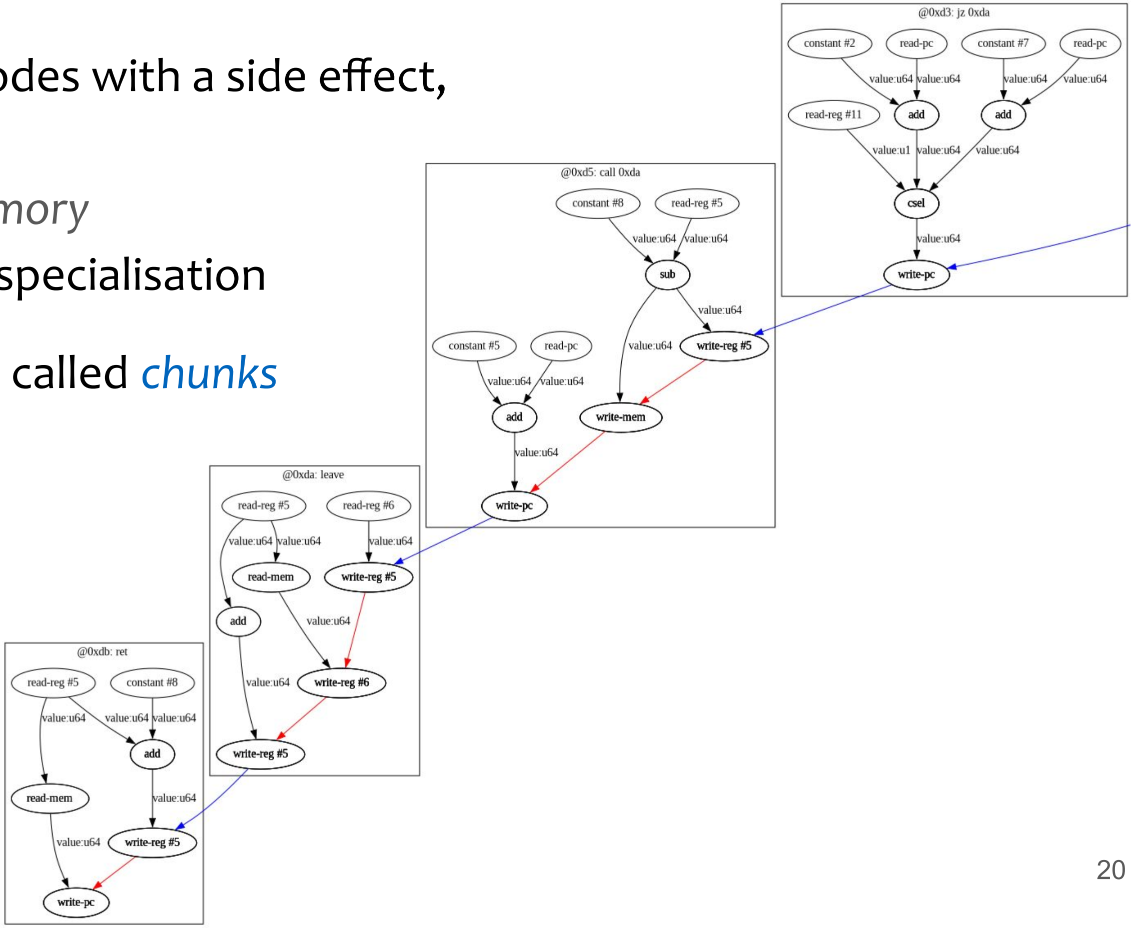
Packets are linked together by nodes with a side effect,
not by control flow

e.g., writing to a register/memory

This simplifies optimisations and specialisation

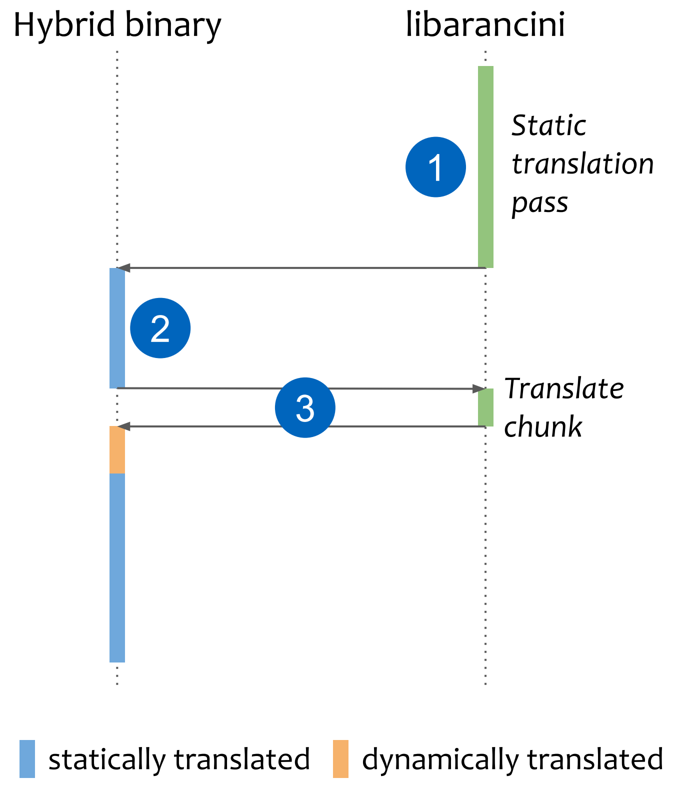
Groups of connected packets are called *chunks*

i.e., a function



The Arancini runtime

1. Static pass generates a hybrid binary containing both guest and host instructions
2. Execute host instructions
3. If an unknown address is reached, jump to our Arancini library, translate the new chunk, and jump back to execution


Hybrid binary
libarancini
1
Static translation pass
2
3
Translate chunk
statically translated
dynamically translated