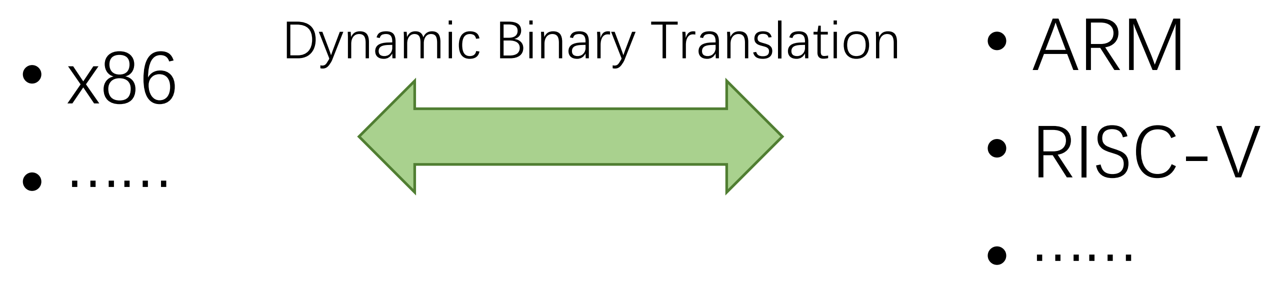
CrossMapping: Harmonizing Memory Consistency in Cross-ISA Binary Translation

Chen Gao, Xiangwei Meng, Wei Li, Jinhui Lai,
Yiran Zhang, and Fengyuan Ren

Dynamic Binary Translation

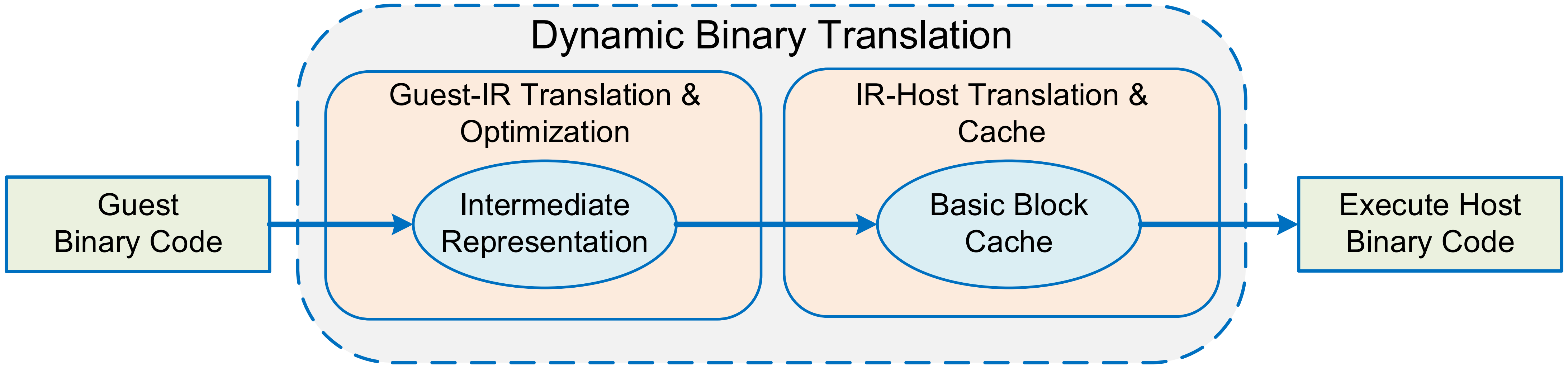


- x86
- ……

- ARM
- RISC-V
- ……

Dynamic Binary Translation

- DBT technology can emulate guest binary programs on the host by translating codes at runtime.


Dynamic Binary Translation
Guest Binary Code
Guest-IR Translation & Optimization
Intermediate Representation
IR-Host Translation & Cache
Basic Block Cache
Execute Host Binary Code

Memory Model

- The memory model describes the behavior of concurrent primitives on shared memory.

Memory Model

- The memory model describes the behavior of concurrent primitives on shared memory.

Memory Access	x86	ARMv8
Load → Load	Ordered	Out-of-order
Load → Store	Ordered	Out-of-order
Store → Load	Out-of-order	Out-of-order
Store → Store	Ordered	Out-of-order

Memory Consistency Issues in DBT

- Message passing test in x86

Initially X=0, Y=0

```
X = 1;   |   a = Y;
         |
Y = 1;   |   b = X;
```

a = 1, b = 0 **Forbidden**

Memory Consistency Issues in DBT

- Message passing test in x86

Initially X=0, Y=0

```
X = 1;   |   a = Y;
Y = 1;   |   b = X;
```

a = 1, b = 0 **Forbidden**

x86 to ARMv8
Naive Translation

- Message passing test in ARMv8

Initially X=0, Y=0

```
X = 1;   |   a = Y;
Y = 1;   |   b = X;
```

a = 1, b = 0 **Observable**

Memory Consistency Issues in DBT

- Message passing test in x86

Initially X=0, Y=0

```
X = 1;   |   a = Y;
Y = 1;   |   b = X;
```

a = 1, b = 0 **Forbidden**

Insert Barriers to Ensure Memory Ordering

- Message passing test in ARMv8

Initially X=0, Y=0

```
X = 1;           |   a = Y;
fence st-st      |   fence ld-ld
Y = 1;           |   b = X;
```

a = 1, b = 0 **Forbidden**

Memory Consistency Issues in DBT

- Message passing test in x86

Initially X=0, Y=0

```
X = 1;   |   a = Y;
Y = 1;   |   b = X;
```

a = 1, b = 0 **Forbidden**

Eliminate Redundant Barriers

- Message passing test in ARMv8

Initially X=0, Y=0

```
X = 1;          |   a = Y;
fence st-st     |   fence ld-ld
Y = 1;          |   b = X;
```

a = 1, b = 0 **Forbidden**

Harmonizing Memory Consistency

- QEMU

Table: QEMU mapping schemes (x86 to ARMv8)

x86		TCG IR		ARMv8
Load	→	Fmr; ld	→	DMB ld; LDR
Store	→	Fmw; st	→	DMB full; STR
RMW	→	call	→	BLR; RMW; RET
MFENCE	→	Fsc	→	DMB full

Harmonizing Memory Consistency

- QEMU

Table: QEMU mapping schemes (x86 to ARMv8)

x86		TCG IR		ARMv8
Load	→	Fmr; ld	→	DMB ld; LDR
Store	→	Fmw; st	→	DMB full; STR
RMW	→	call	→	BLR; RMW; RET
MFENCE	→	Fsc	→	DMB full

Harmonizing Memory Consistency

- QEMU

Table: QEMU mapping schemes (x86 to ARMv8)

x86		TCG IR		ARMv8
Load	→	Fmr; ld	→	DMB ld; LDR
Store	→	Fmw; st	→	DMB full; STR
RMW	→	call	→	BLR; RMW; RET
MFENCE	→	Fsc	→	DMB full

Harmonizing Memory Consistency

- QEMU

Fetch-And-Add litmus test

Initially X=0, Y=0

```
         | a = Y;
X = 1;   | if(a == 1)
Y = 1;   |     FAA(X,1) //LOCK XADD
```

a = 1, X = 1 **Forbidden**

x86 Pseudo-Assembly

Initially X=0, Y=0

```
DMB full;  | DMB full;
  X = 1;   | a = Y;
DMB full;  | if(a == 1)
  Y = 1;   |     FAA(X,1) //LDAXR–STLXR
```

a = 1, X = 1 **Observable**

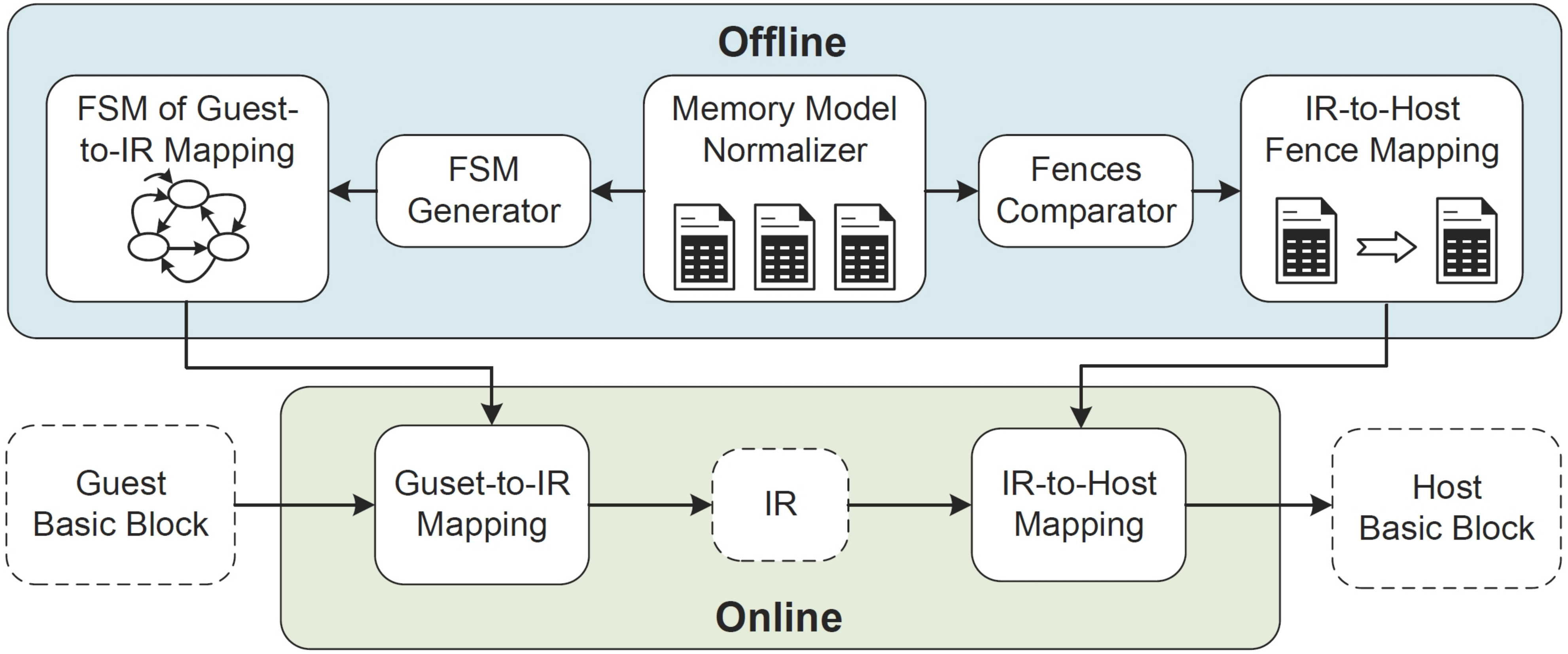
ARMv8 Pseudo-Assembly

Harmonizing Memory Consistency

- Risotto[ASPLOS'23] and Lasagne[PLDI'22]
  - ➢Correctly handle RMW instructions
  - ➢Designed specifically for x86 to ARMv8

- ArMOR [ISCA'15]
  - ➢More efficient
  - ➢Not designed for Cross-ISA DBT systems.

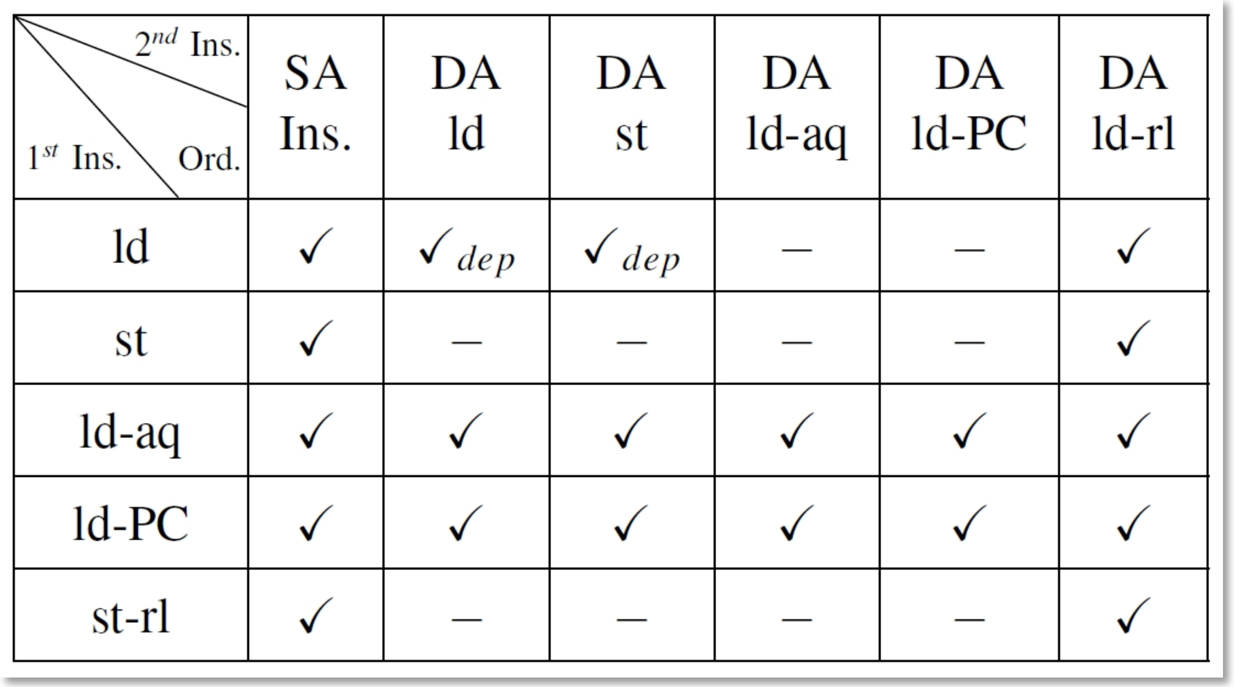
Overview of CrossMapping



Memory Model Normalizer

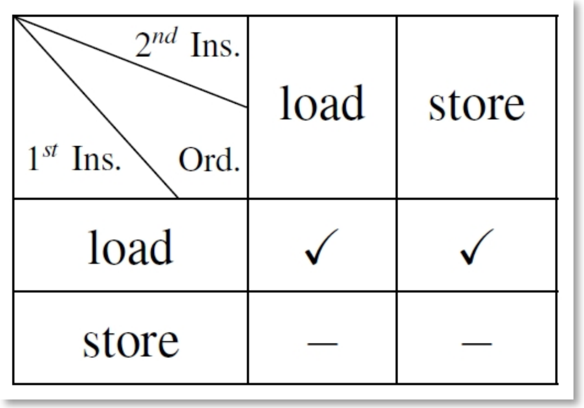
- Definition of specification table

Specification table of ARMv8 memory orderings



2nd Ins. / Ord. / 1st Ins.	SA Ins.	DA ld	DA st	DA ld-aq	DA ld-PC	DA ld-rl
ld	✓	✓ *dep*	✓ *dep*	–	–	✓
st	✓	–	–	–	–	✓
ld-aq	✓	✓	✓	✓	✓	✓
ld-PC	✓	✓	✓	✓	✓	✓
st-rl	✓	–	–	–	–	✓

Specification table of **`DMB ld`**

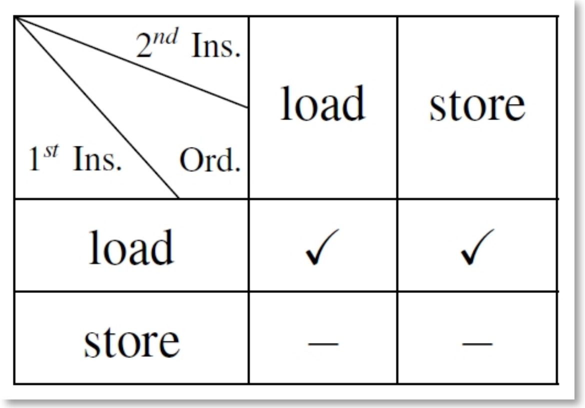


2nd Ins. / Ord. / 1st Ins.	load	store
load	✓	✓
store	–	–

Memory Model Normalizer

- Refinement

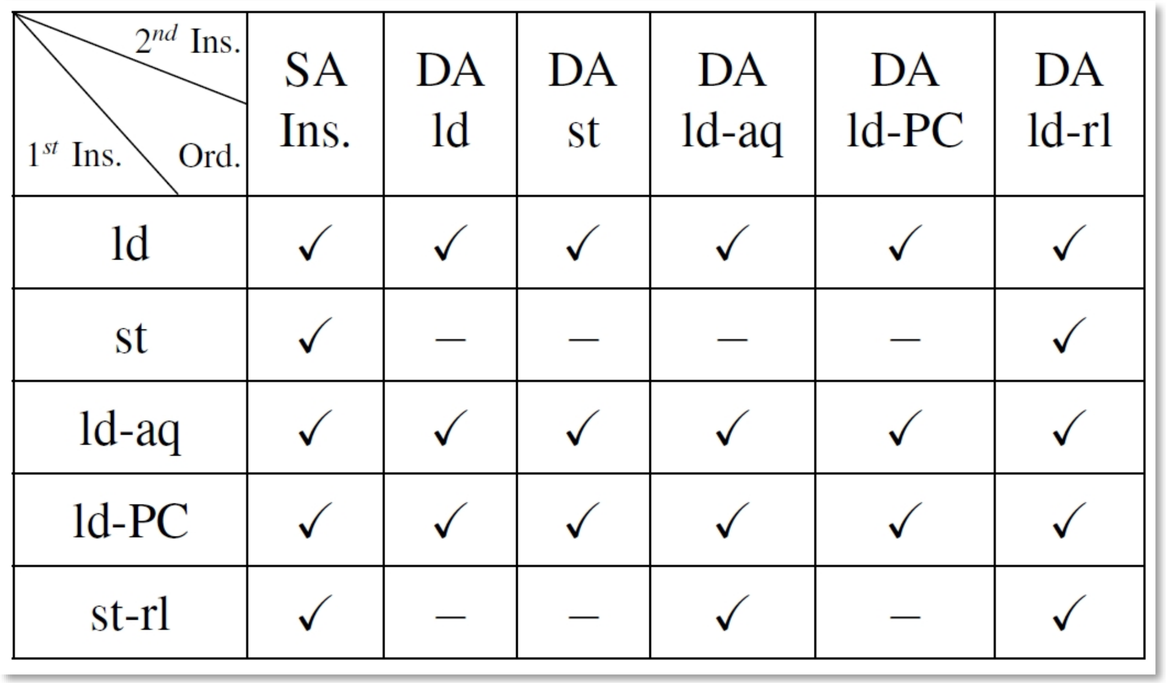
Specification table of **`DMB ld`**



2nd Ins. / 1st Ins. Ord.	load	store
load	✓	✓
store	—	—

Refine

Refined specification table for **`DMB ld`**



2nd Ins. / 1st Ins. Ord.	SA Ins.	DA ld	DA st	DA ld-aq	DA ld-PC	DA ld-rl
ld	✓	✓	✓	✓	✓	✓
st	✓	—	—	—	—	✓
ld-aq	✓	✓	✓	✓	✓	✓
ld-PC	✓	✓	✓	✓	✓	✓
st-rl	✓	—	—	✓	—	✓

Memory Model Normalizer

- Comparison

	Ins. 1	Ins. 2
Ins. 1	✓	✓
Ins. 2	✓	—

$\geqslant$

	Ins. 1	Ins. 2
Ins. 1	✓	✓
Ins. 2	—	—

Memory Model Normalizer

- Union

	Ins. 1	Ins. 2
Ins. 1	✓	✓
Ins. 2	✓	—

$\cup$

	Ins. 1	Ins. 2
Ins. 1	✓	✓
Ins. 2	—	—

=

	Ins. 1	Ins. 2
Ins. 1	✓	✓
Ins. 2	✓	—

- Subtraction

	Ins. 1	Ins. 2
Ins. 1	✓	✓
Ins. 2	✓	—

$-$

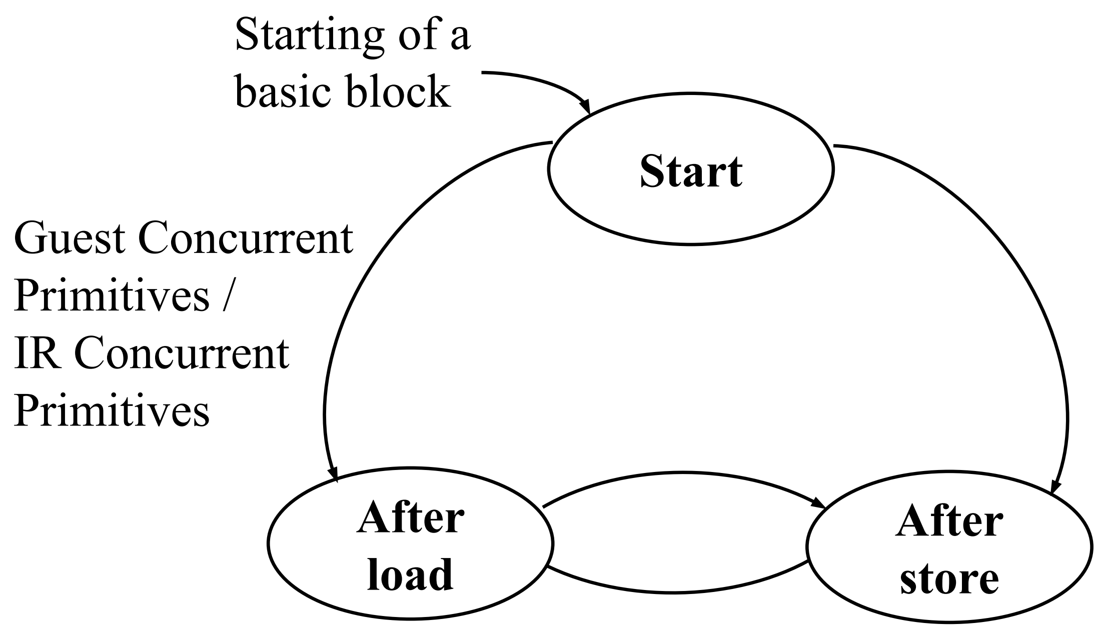
	Ins. 1	Ins. 2
Ins. 1	✓	✓
Ins. 2	—	—

=

	Ins. 1	Ins. 2
Ins. 1	—	—
Ins. 2	✓	—

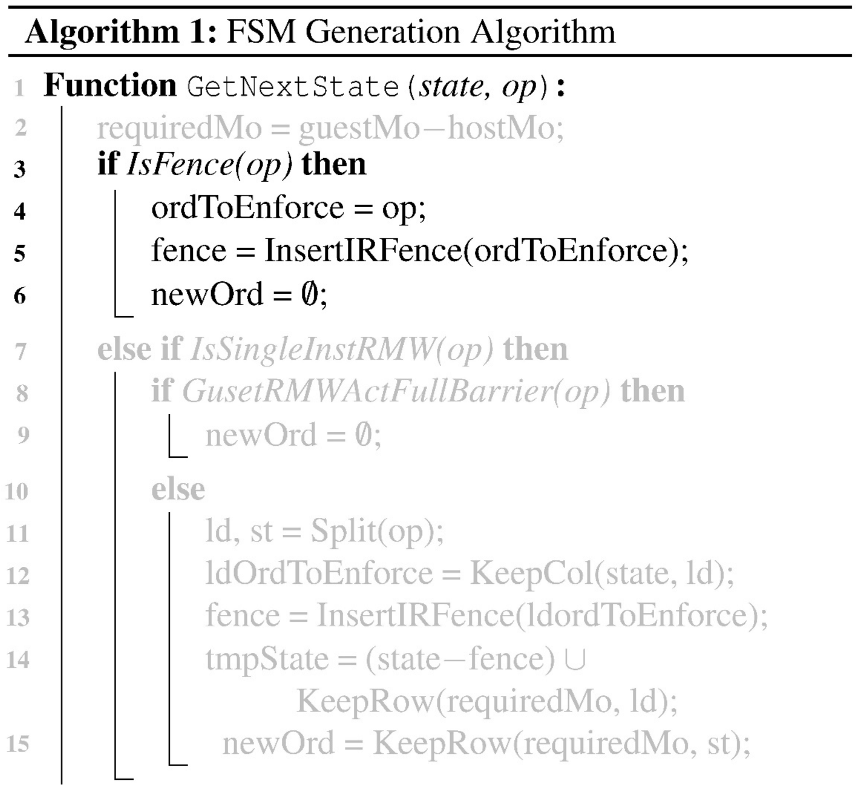
Guest-to-IR Mapping via FSM

- An example of FSM


Starting of a basic block
Start
Guest Concurrent Primitives / IR Concurrent Primitives
After load
After store

Guest-to-IR Mapping via FSM

- FSM Generator
    - ➢Fences
    - ➢Single-instruction RMWs
    - ➢End of the basic block
    - ➢Other memory accesses



Algorithm 1: FSM Generation Algorithm

```
Function GetNextState(state, op):
    requiredMo = guestMo−hostMo;
    if IsFence(op) then
        ordToEnforce = op;
        fence = InsertIRFence(ordToEnforce);
        newOrd = ∅;
    else if IsSingleInstRMW(op) then
        if GusetRMWActFullBarrier(op) then
            newOrd = ∅;
        else
            ld, st = Split(op);
            ldOrdToEnforce = KeepCol(state, ld);
            fence = InsertIRFence(ldordToEnforce);
            tmpState = (state−fence) ∪
                    KeepRow(requiredMo, ld);
            newOrd = KeepRow(requiredMo, st);
```

Guest-to-IR Mapping via FSM

- FSM Generator
  - ➢Fences
  - ➢Single-instruction RMWs
  - ➢End of the basic block
  - ➢Other memory accesses

```
if $IsFence(op)$ then
    ordToEnforce = op;
    fence = InsertIRFence(ordToEnforce);
    newOrd = $\emptyset$;
else if $IsSingleInstRMW(op)$ then
    if $GusetRMWActFullBarrier(op)$ then
        newOrd = $\emptyset$;
    else
        ld, st = Split(op);
        ldOrdToEnforce = KeepCol(state, ld);
        fence = InsertIRFence(ldordToEnforce);
        tmpState = (state$-$fence) $\cup$
                KeepRow(requiredMo, ld);
        newOrd = KeepRow(requiredMo, st);
else if $IsEndOfBasicBlock(op)$ and
  $isAfterRMW(state)$ then
    fence = InsertIRFence(fullfence$-$state);
    newOrd = $\emptyset$;
else
```

Guest-to-IR Mapping via FSM

- FSM Generator
  - ➢Fences
  - ➢Single-instruction RMWs
  - ➢End of the basic block
  - ➢Other memory accesses

```
        if GusetRMWActFullBarrier(op) then
            newOrd = ∅;
        else
            ld, st = Split(op);
            ldOrdToEnforce = KeepCol(state, ld);
            fence = InsertIRFence(ldordToEnforce);
            tmpState = (state−fence) ∪
                    KeepRow(requiredMo, ld);
             newOrd = KeepRow(requiredMo, st);
    else if IsEndOfBasicBlock(op) and
     isAfterRMW(state) then
        fence = InsertIRFence(fullfence−state);
        newOrd = ∅;
    else
        ordToEnforce = KeepCol(state, op);
        InsertIRFence(ordToEnforce);
        newOrd = KeepRow(requiredMo, op);
    nextState = (state−fence) ∪ newOrd;
    return nextState;
```

Guest-to-IR Mapping via FSM

- FSM Generator
  - ➢Fences
  - ➢Single-instruction RMWs
  - ➢End of the basic block
  - ➢Other memory accesses

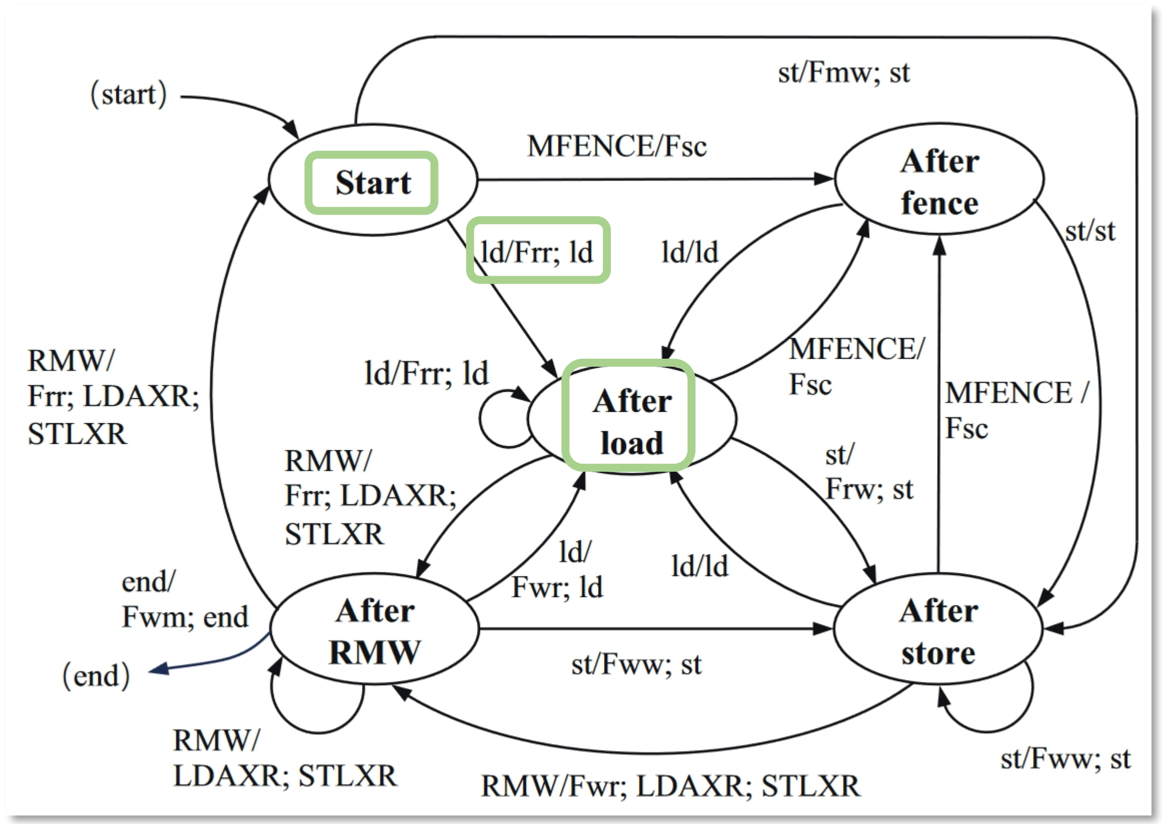
```
    if GusetRMWActFullBarrier(op) then
        newOrd = ∅;
    else
        ld, st = Split(op);
        ldOrdToEnforce = KeepCol(state, ld);
        fence = InsertIRFence(ldordToEnforce);
        tmpState = (state−fence) ∪
                KeepRow(requiredMo, ld);
         newOrd = KeepRow(requiredMo, st);
else if IsEndOfBasicBlock(op) and
 isAfterRMW(state) then
    fence = InsertIRFence(fullfence−state);
    newOrd = ∅;
else
    ordToEnforce = KeepCol(state, op);
    InsertIRFence(ordToEnforce);
    newOrd = KeepRow(requiredMo, op);
nextState = (state−fence) ∪ newOrd;
return nextState;
```

IR-to-Host Mapping

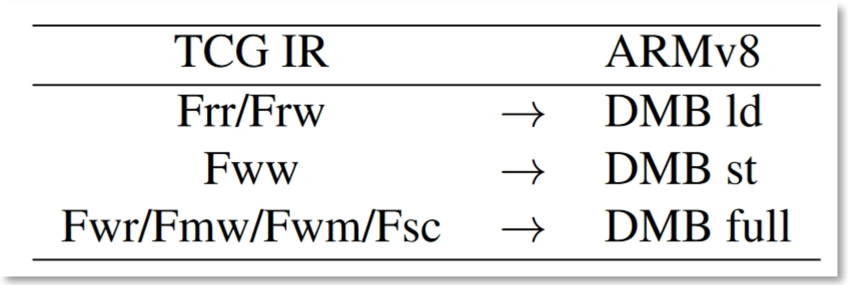
- Fences Comparator
  - ➢For each IR fence, the comparator identifies the weakest host fence to satisfy all the enforced orderings of IR fence.

Case Study: x86 to ARMv8 Mapping

- FSM of mapping from x86 to TCG IR



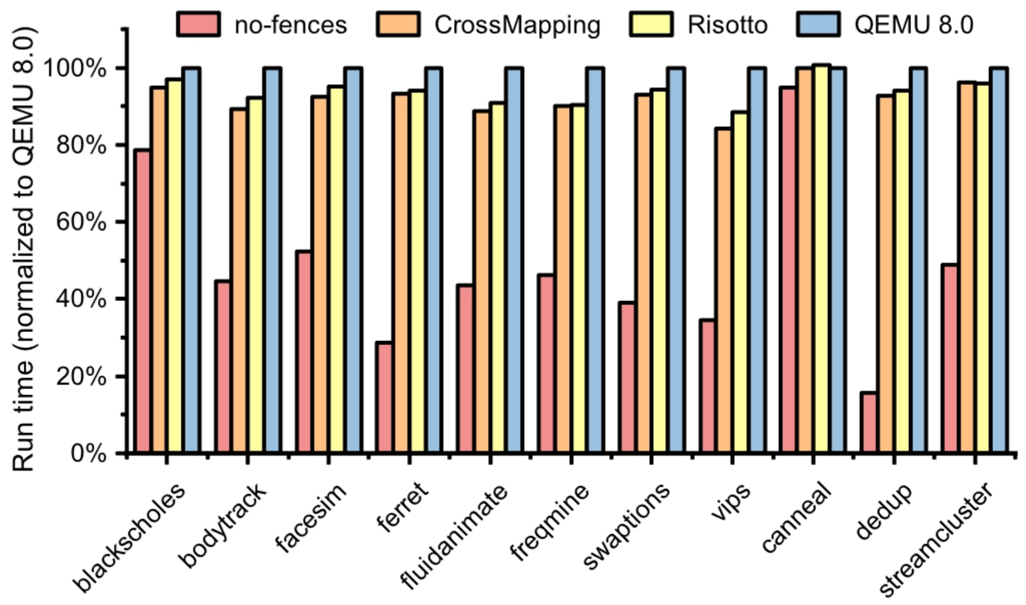
- TCG IR to ARMv8 fence mapping scheme



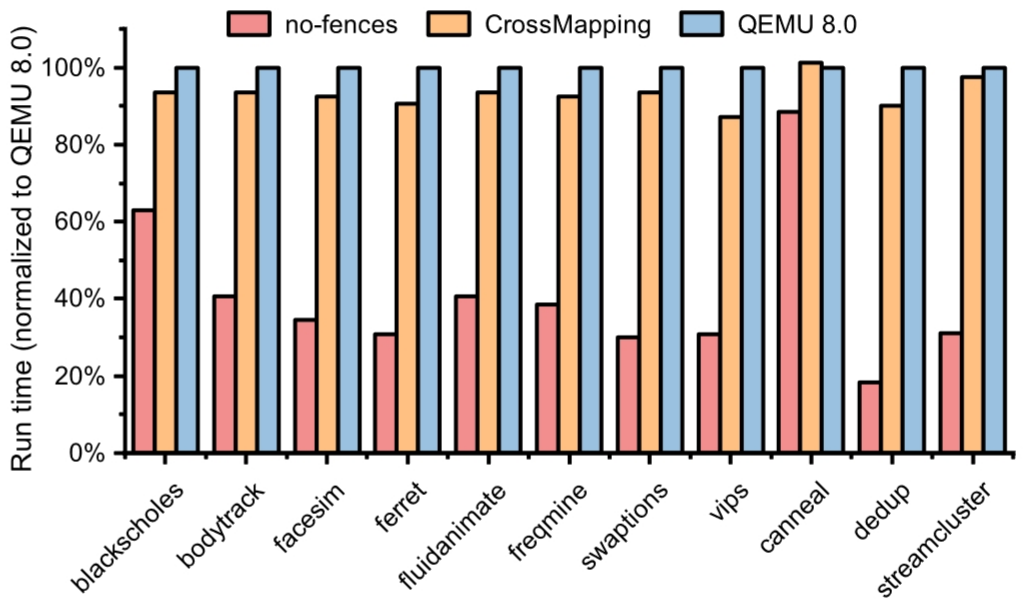
TCG IR		ARMv8
Frr/Frw	→	DMB ld
Fww	→	DMB st
Fwr/Fmw/Fwm/Fsc	→	DMB full

Evaluation

- Strong-on-Weak Architecture Emulation



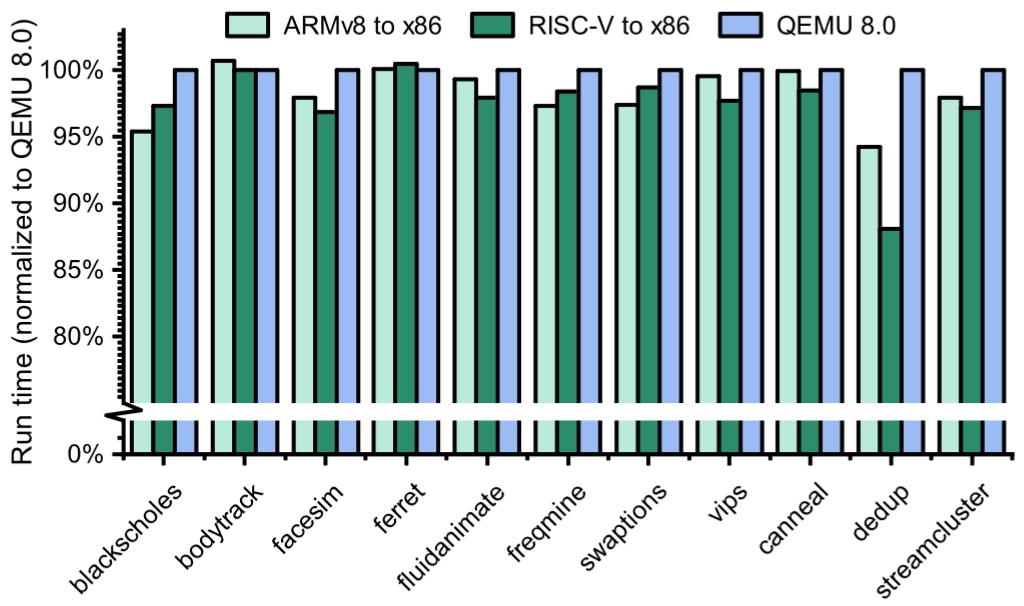
(a) Emulating x86 on ARMv8



(b) Emulating x86 on RISC-V

Evaluation

- Weak-on-Strong Architecture Emulation



Thanks!