Dynamic Binary Translation

Shadow Stack

Block Chaining

Atmoic Instruction

Static Binary Translation

Memory Consistency

Indirect Branch

SoftMMU

Code Cache

Multiple Thread

Self-Modified Code

(Dynamic) Code Optimization

Trace / Region

A Survey on Binary Translation

NiuGen 2021/12/??

IA32-EL

HQEMU

Shade

BOX32

rev.ng

LLBT

QEMU

DQEMU

BOX64

UQBT

PQEMU

ExGear

Captive

Rosetta

LoongsonLab

Embra

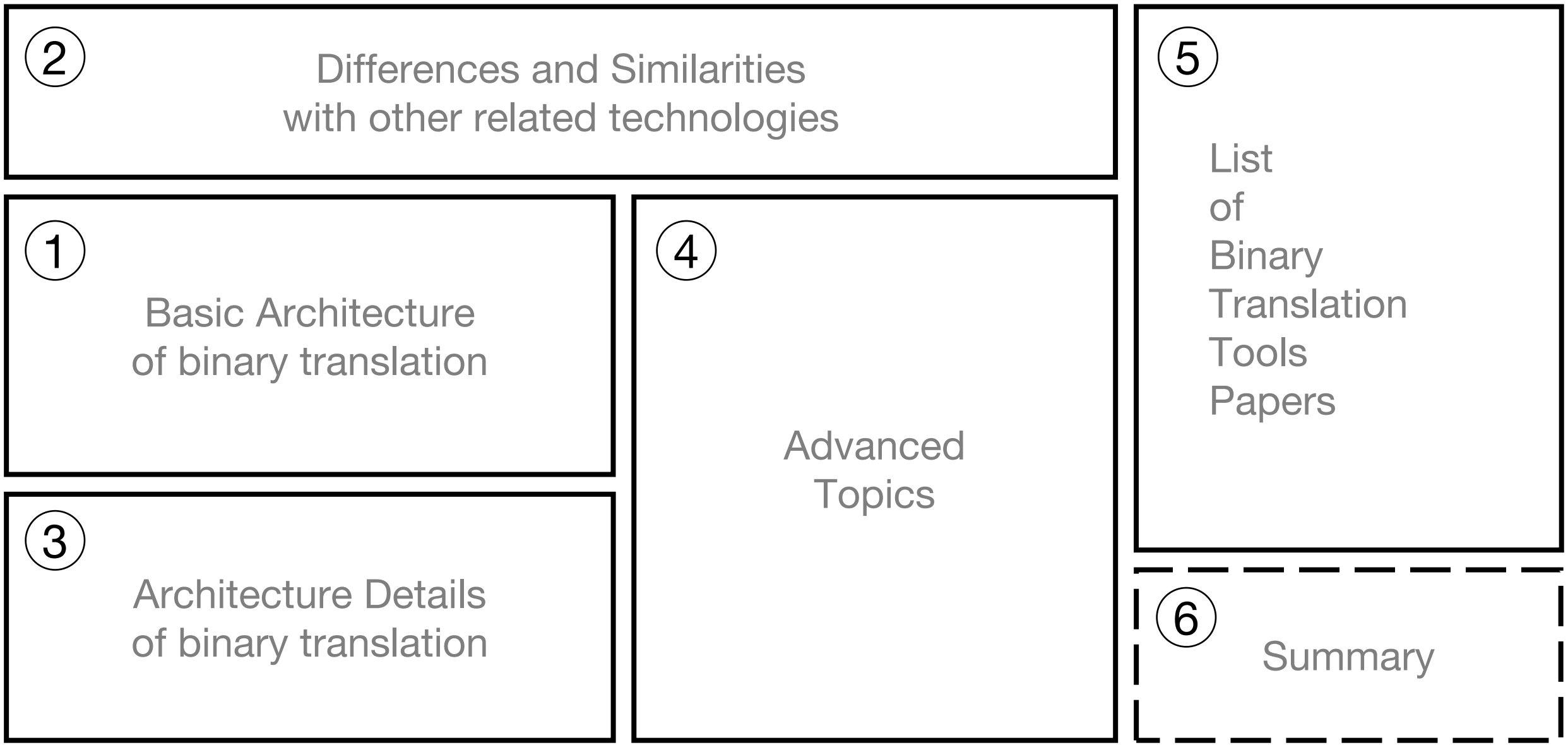
FX!32

DynamoRIO

Transmeta

DAISY

A Survey on Binary Translation



A Survey on Binary Translation

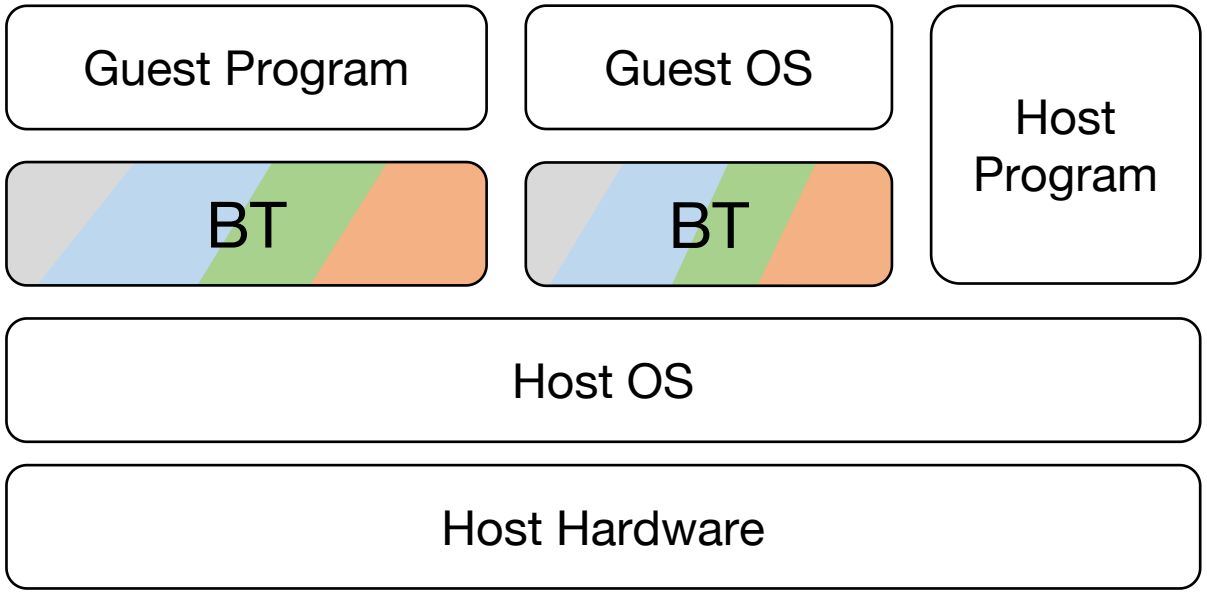
- Introduction ..... basic architecture

- Related Technologies ..... differences and similarities
- Architecture Details ..... and basic optimizations
- Advanced Topics ..... problems, challenges, optimizations, etc.

- List of Binary Translation ..... by year, type, etc.
- Summary

Introduction

executing host binaries which are translated from guest binaries



- Fetch Guest Binaries
- Disassemble
- Translate to Host Binaries
- Executing Host Bianries
  - with Simulation Environment

- Why translate? What is the purpose of binary translation? What can binary translation do?
- How to translate? / How to maintain correctness?
- What is the simulation environment? What is its relationship with translation?

Introduction

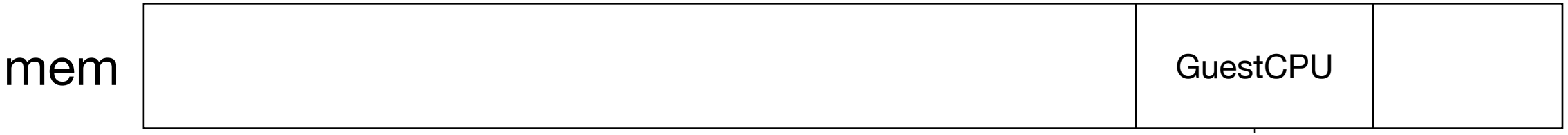
Why translate? What is the purpose of binary translation? What can binary translation do?

- Program Emulation
  - program migration (legacy codes, cross-ISA codes, ...
  - debuging support

- Program Analysis (depends on what you want to do
  - instrument
  - optimization
  - bug detection, security
  - tracing
  - profiling, ...

Introduction

How to translate? / How to maintain correctness?



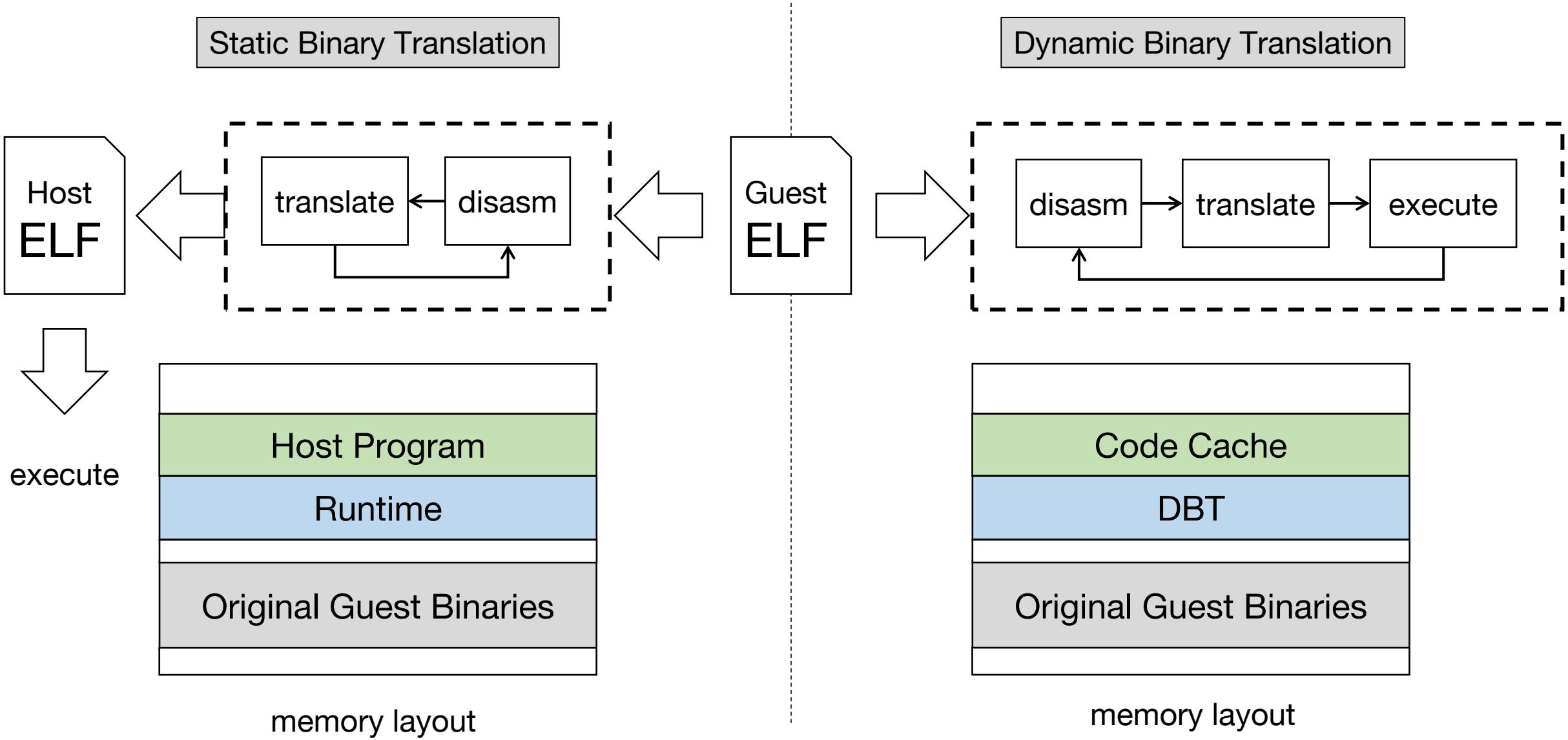
i386

```
add     %eax,   %ebx
```

```
struct GuestCPU {
    reg_t  regs[8];
    reg_t  eip;
    seg_t  segs[6];
    ....
} cpu
```

	i386	loongarch
read	`mov %eax, [ cpu.regs[0] ]`	`ld.w $t0, [ cpu.regs[0] ]`
read	`mov %ebx, [ cpu.regs[3] ]`	`ld.w $t1, [ cpu.regs[3] ]`
add	`add %eax, %ebx`	`add.w $t0, $t1`
write	`mov [ cpu.regs[0] ], %eax`	`st.w $t0, [ cpu.regs[0] ]`
	`# EFLAGS CALCULATION`	`# EFLAGS CALCULATION`

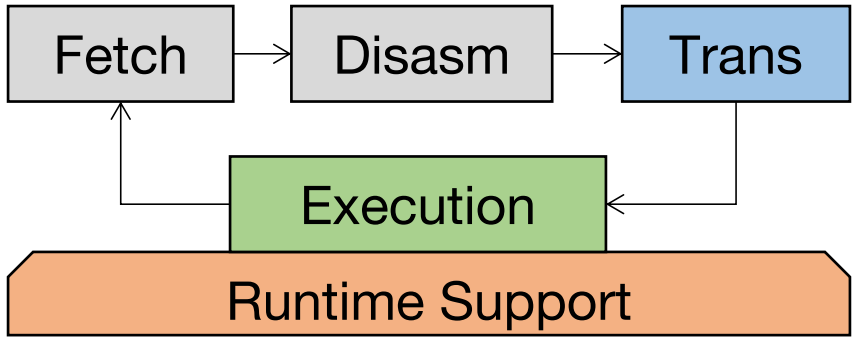
Introduction



A Survey on Binary Translation

2

Differences and Similarities
with other related technologies



3

Architecture Details
of binary translation

4

Advanced
Topics

5

List
of
Binary
Translation
Tools
Papers

6

Summary

A Survey on Binary Translation

- Introduction ..... basic architecture

- Related Technologies ..... differences and similarities
- Architecture Details ..... and basic optimizations
- Advanced Topics ..... problems, challenges, optimizations, etc.

- List of Binary Translation ..... by year, type, etc.
- Summary

Related Technology

Binary Instrument/Optimization ..... Binary Rewrite

- Fetch Guest Binaries
- Disassemble
- Translate / Optimize / Instrument
- Executing and collecting dynamic information

Binary Instrument
- instrument without translation
- translate with instrument

- Binary Rewriting [1]
  - Emulation
  - Optimization
  - Observation
  - Hardending
    - post-compile & link-time

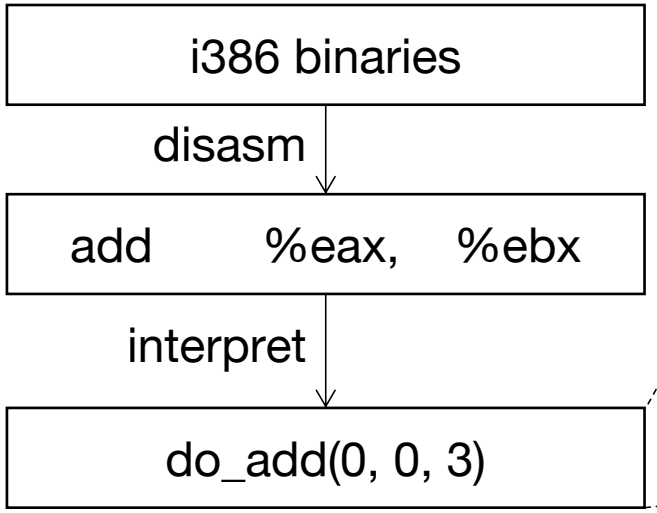
[1] 2019. From Hack to Elaborate Technique - A Survey on Binary Rewriting

Related Technology

Interpretation

- Script language: python, shell, javascript, ...
- No translation
- program to implement guest binaries' function
- similar to helper function in binary translator

```
struct GuestCPU {
    reg_t  regs[8];
    reg_t  eip;
    seg_t  segs[6];
    ....
} cpu
```




```
void do_add ( int dest, int src1, int src2 )
{
        reg_t   a    =   cpu.regs[src1]
        reg_t   b    =   cpu.regs[src2]
        reg_t   res  =   a + b;
        cpu.regs[dest] = res;
}
```

Related Technology

Just-In-Time Compilation (Java)

- JIT dynamically trenslate Intermediate-Representation into host binaries to execute.
  - IR can not executing on hardware directly and
  - could contain high-level information.

- Binary Translation translates one ISA binary to another ISA binary.
  - such as x86, ARM, MIPS, Alpha, ...
  - could executing on hardware directly
- Binary translation is a kind of dynamic compilation

Related Technology

Virtualization (KVM, ...

- Using bianry translation to do virualization in pure software
  - translate sensitive instruction into user-space instruction
- Using binary translation to help virtualization

- Hardware Support Virtualization
  - no binary translation
  - but only for same-ISA
  - direct execution and trap and emulate [1]

```
struct GuestCPU {
    reg_t  regs[8];
    reg_t  eip;
    seg_t  segs[6];
    ....
    cr_t     CR[];
    dr_t     DR[];
    msr_t    MSR[]
} cpu
```

[1] 2017. Edouard Bugnion. Hardware and Software support for virtualization. (Synthesis Lectures)

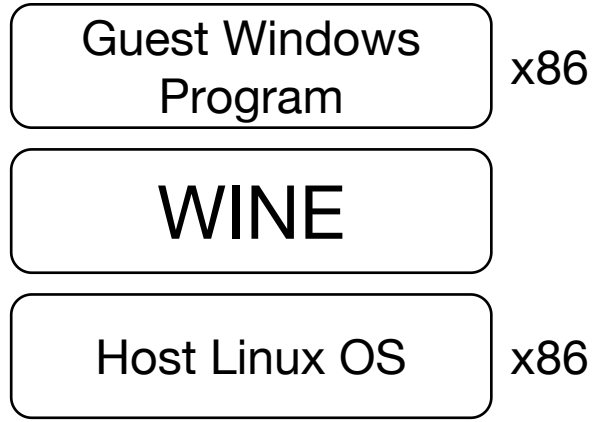
Related Technology

Hardware Simulation (GEM5)

- Focus on microarchitecture simulation
  - NOT emulation (which is functional simulation)
  - Cache, TLB, Pipeline, OoO, ...
- Binary Translation can be used to accelerate simulation [1]

Interface Abstraction

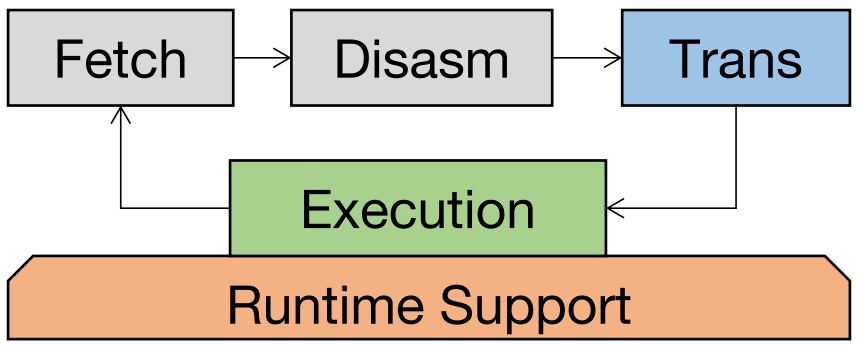
- For different platform with same-ISA
  - Android Runtime: run android app on different platform
  - WINE: run windows program in linux
  - Linux posix interface



[1] 2010. Igor Bohm. Cycle-accurate performance modelling in an ultra-fast just-in-time dynamic binary translation instruction set simulator

A Survey on Binary Translation

Binary Rewrite	=	Translation	,	Optimization	,	Instrumentation
Script Language	=	JIT	Java / JVM	,	Interpret	Python, Matlab
Virtualization	=	KVM	, Wine	,	Android	, Gem5



3

Architecture Details
of binary translation

4

Advanced
Topics

5

List
of
Binary
Translation
Tools
Papers

6

Summary

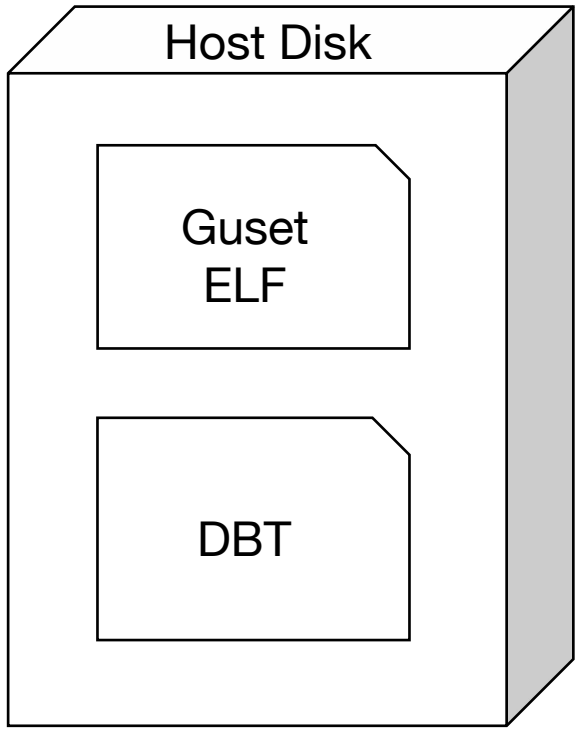
A Survey on Binary Translation

- Introduction ..... basic architecture

- Related Technologies ..... differences and similarities
- Architecture Details ..... and basic optimizations
- Advanced Topics ..... problems, challenges, optimizations, etc.

- List of Binary Translation ..... by year, type, etc.
- Summary

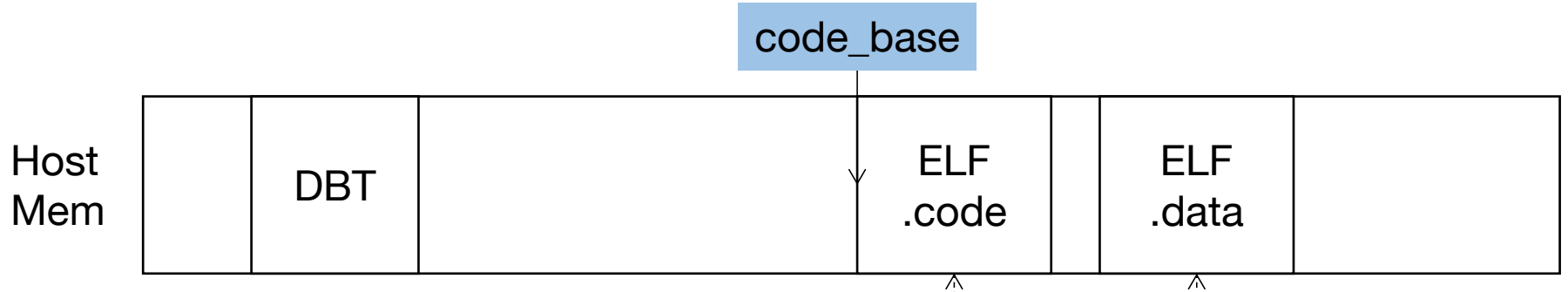
Architecture - Fetch



Entry point address = 0x401bc0 (GVA)

Offset from .code start = 0x401bc0 - 0x4011a0

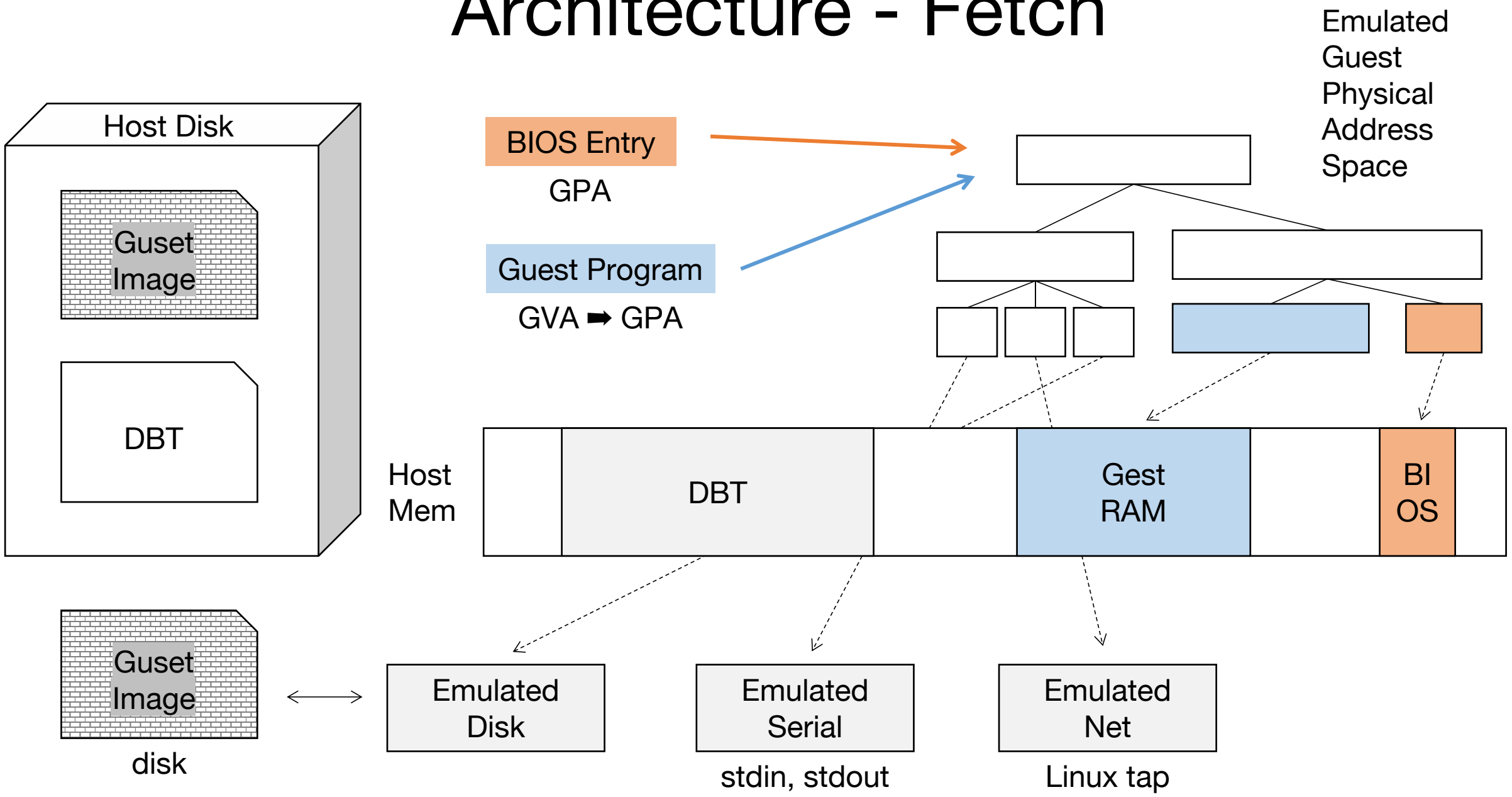
Real address (HVA) = code_base + (0x401bc0 - 0x4011a0)



	offset	GVA	length
.code	0x11a0	0x4011a0	0x91800
.data	0x95000	0x495000	0x1bfcc

Entry point address: 0x401bc0

Architecture - Fetch



Architecture - Disassemble

Binary: 554889E5897DFC8975F88b55FC8b45F801d05dC3

55		push	%rbp	
48 89 e5		mov	%rsp,	%rbp
89 7d fc		mov	%edi,	-0x4(%rbp)
89 75 f8		mov	%esi,	-0x8(%rbp)
8b 55 fc		mov	-0x4(%rbp),	%edx
8b 45 f8		mov	-0x8(%rbp),	%eax
01 d0		add	%edx,	%eax
5d		pop	%rbp	
c3		ret		

prepare stack

get argument from stack

operation

restore stack

return

Architecture - Disassemble

- DiStorm https://github.com/gdabah/distorm
- Capstone https://github.com/capstone-engine/capstone
- Intel XED https://github.com/intelxed/xed
- Zydis https://github.com/zyantific/zydis

- Switch-Case
  - combine with translation

```
switch ( b ) {
case 0x88:
case 0x89: /* mov Gv, Ev */
        ot = mo_b_d(b, dflag)
        modrm = x86_ldub_code(env, s)
        reg = ((modrm >> 3) & 7) | rex_r;

        /* generate a generic store */
        gen_ldst_modrm(env, s, modrm, ot, reg, 1);
...
}
```

disasm

translate

Architecture - Translation

0x401cfa	mov	-0x14(%rbp),	%eax	
0x401cfd	cmp	-0x18(%tbp),	%eax	
0x401d00	jle	0x401d0a		branch
0x401d02	mov	%eax,	-0x4(%rbp)	
0x401d05	jmp	0x401d10		direct jmp
0x401d0a	mov	-0x18(%rbp),	%eax	
0x401d0d	mov	%eax,	-0x4(%rbp)	
0x401d10	mov	-0x4(%rbp),	%eax	
0x401d13	pop	%rbp		
0x401d14	retq			indirect jmp

Architecture - Translation

(1) read from CPU (2) do operation (3) write to CPU

(4) update PC

i386

```
addr  add     %eax,   %ebx
```

loongarch

```
ld.w    $t0,    [ cpu.regs[0] ]
ld.w    $t1,    [ cpu.regs[3] ]
add.w   $t0,    $t1
st.w    $t0,    [ cpu.regs[0] ]
# EFLAGS CALCULATION
# cpu.eip = addr + 2
```

i386

```
addr  jmp     addr2
```

loongarch

```
# cpu.eip = addr2
```

i386

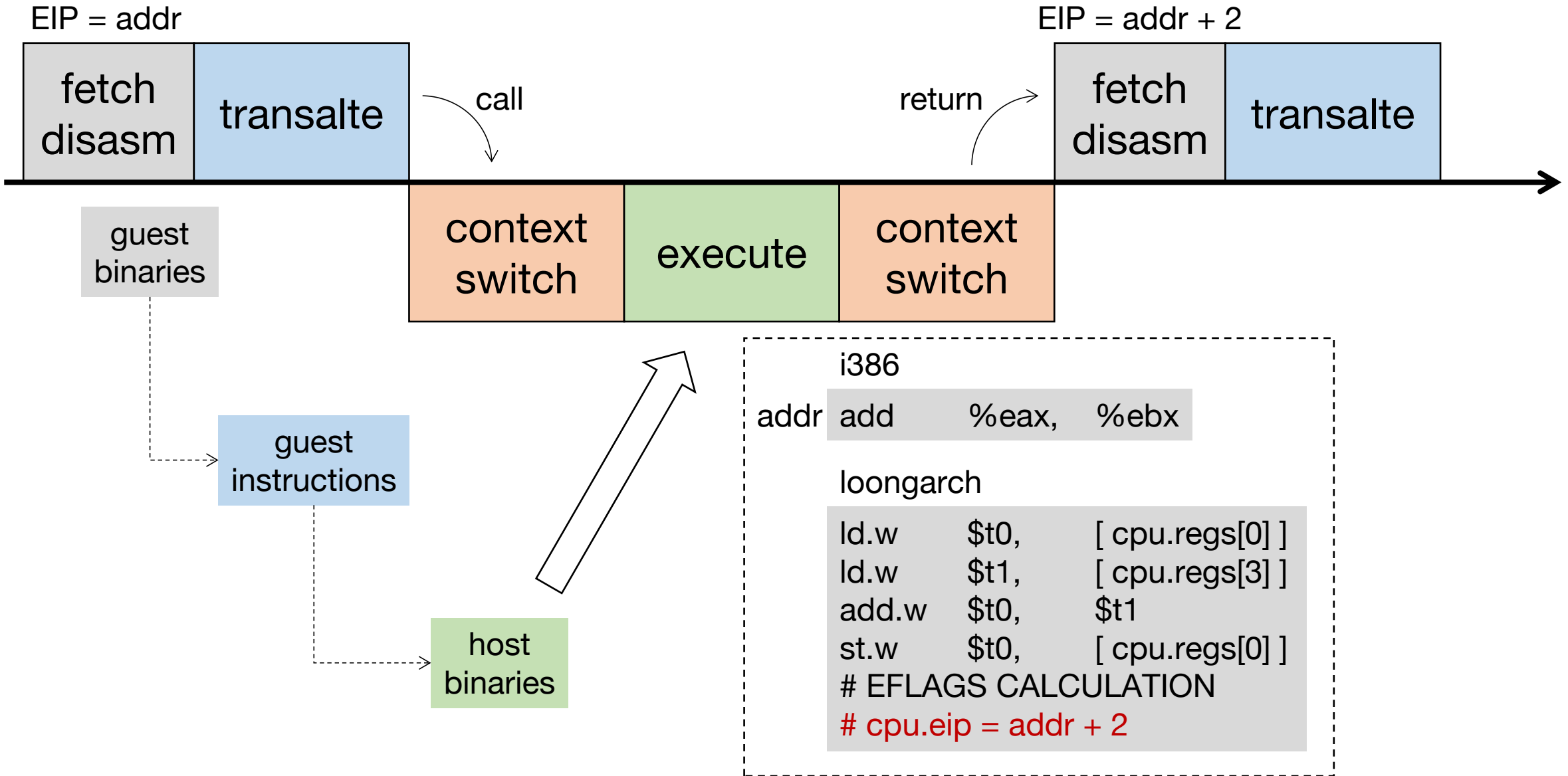
```
addr  je      addr3
```

loongarch

```
        # read EFLAGS.ZF into $t0
        je      $t0,    zero,   L1
        # cpu.eip = addr + 5
        b       NEXT
L1:
        # cpu.eip = addr3
NETX:
```

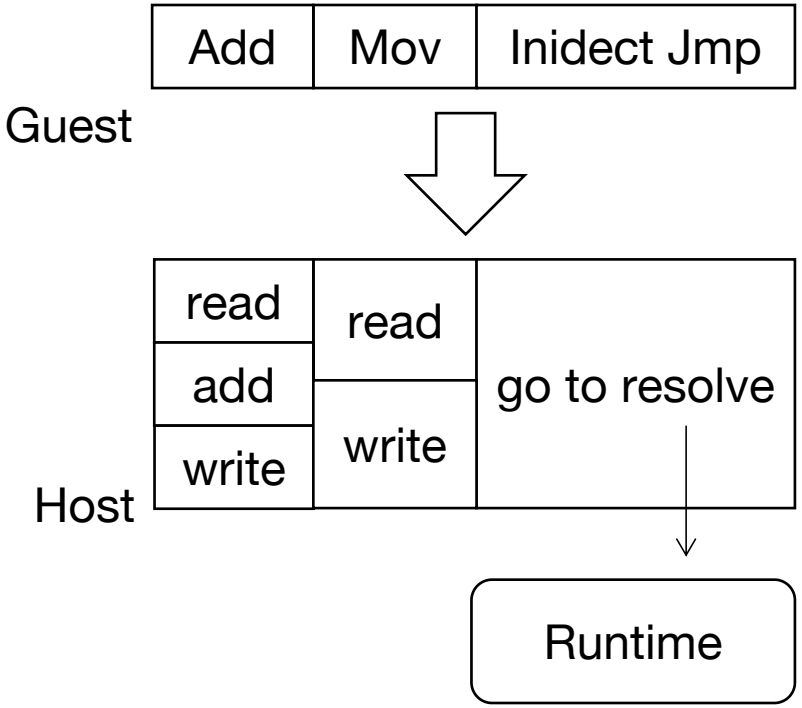
```
struct GuestCPU {
    reg_t  regs[8];
    reg_t  eip;
    seg_t  segs[6];
    ....
} cpu
```

Architecture - Execution

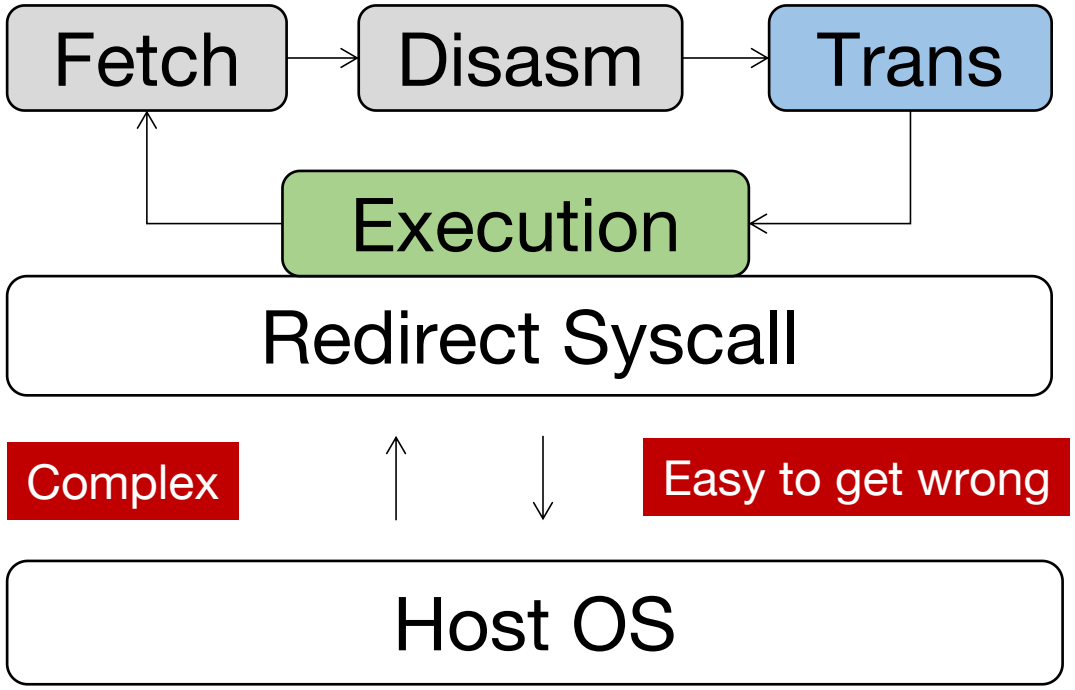


Architecture - user-mode

- Execution Environment (Runtime) Support
  - deal with unknown behavior: indirect branch target, ...
  - deal with behavior not included in program: syscall, ...

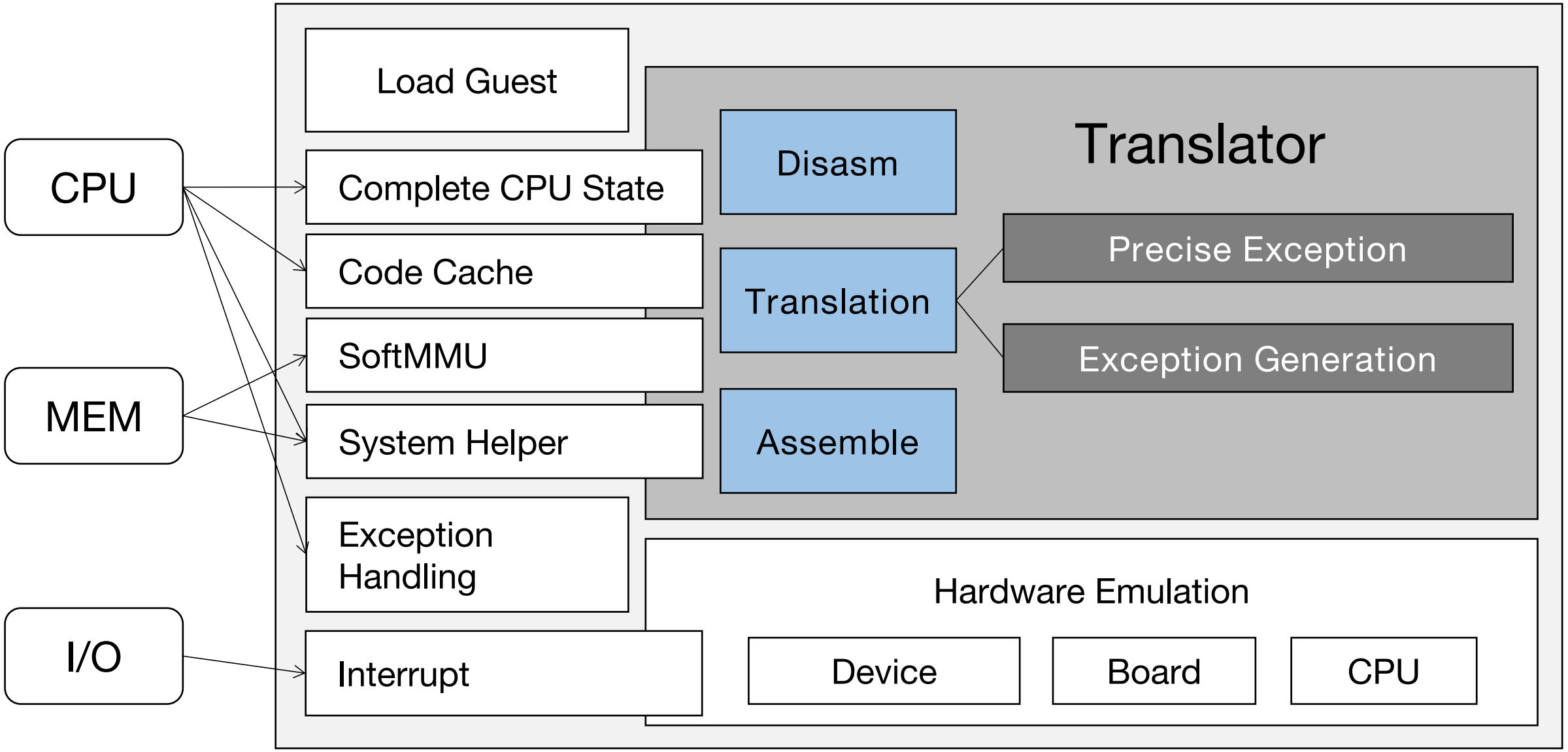


Static Binary Translation



Dynamic Binary Translation

Architecture - system mode

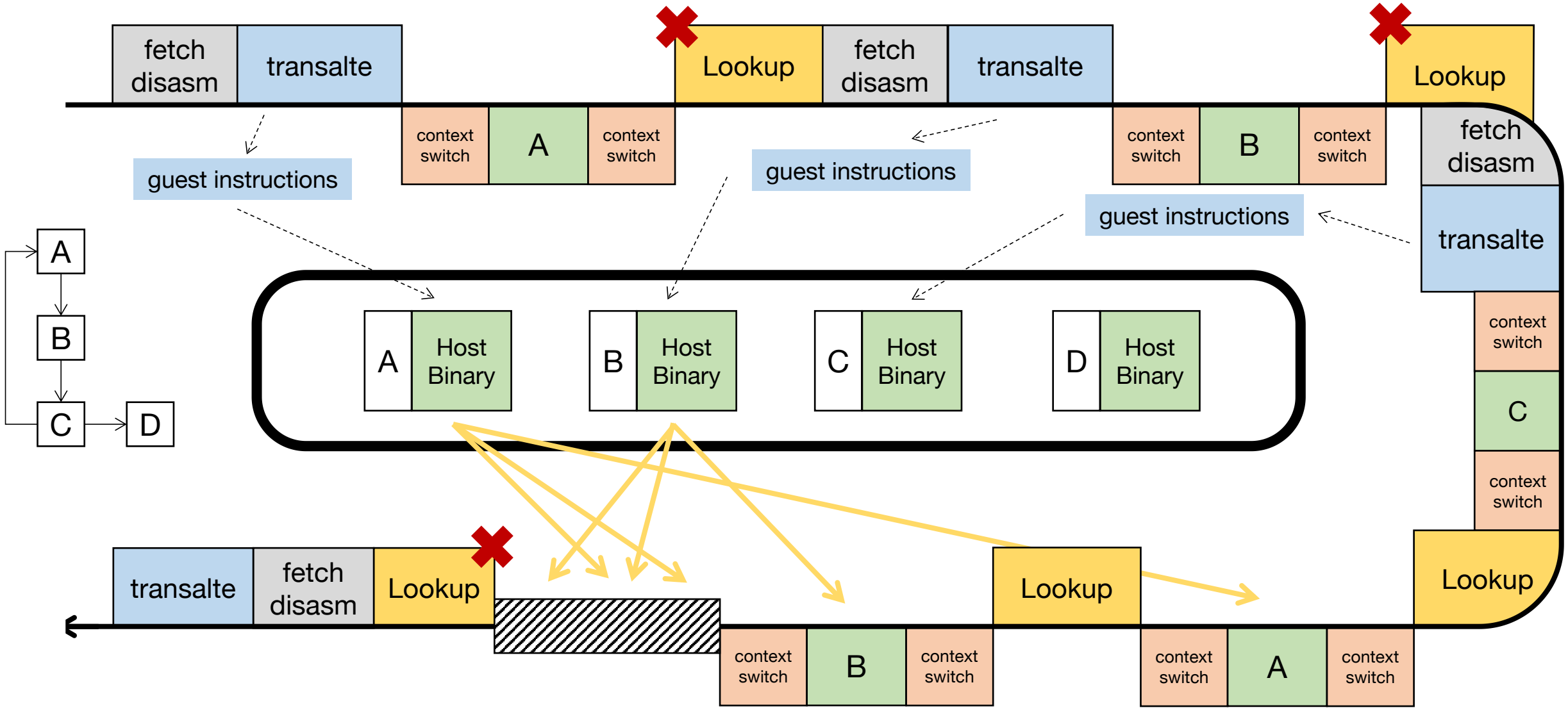


Architecture - Basic Optimizations

- Code Blocks and Code Cache
- Register Mapping
- Block Chaining // for most direct jmp

- Combine with interpretation to filter code codes
- Combine with profiling to generate optimized trace/region

Architecture - Basic Optimizations



Architecture - Basic Optimizations

i386

```
addr    add         %eax, %ebx
```

loongarch

```
ld.w        $t0, [ cpu.regs[0] ]
ld.w        $t1, [ cpu.regs[3] ]
add.w       $t0,         $t1
st.w        $t0, [ cpu.regs[0] ]
# EFLAGS CALCULATION
# cpu.eip = addr + 2
```

i386

```
addr+2  add         %eax, %ecx
```

loongarch

```
ld.w        $t0, [ cpu.regs[0] ]
ld.w        $t1, [ cpu.regs[1] ]
add.w       $t0,         $t1
st.w        $t0, [ cpu.regs[0] ]
# EFLAGS CALCULATION
# cpu.eip = addr + 4
```

BLOCK

i386

```
addr    add     %eax,   %ebx
addr+2  add     %eax,   %ecx
```

loongarch

```
ld.w    $T0,    [ cpu.regs[0] ]
ld.w    $T1,    [ cpu.regs[3] ]
add.w   $T0,    $T1
st.w    $T0,    [ cpu.regs[0] ]
# EFLAGS CALCULATION
ld.w    $T0,    [ cpu.regs[0] ]
ld.w    $T1,    [ cpu.regs[1] ]
add.w   $T0,    $T1
st.w    $T0,    [ cpu.regs[0] ]
# EFLAGS CALCULATION
# cpu.eip = addr + 4
```

Basic Code Block

- continuous instructions
- end with branch / jmp

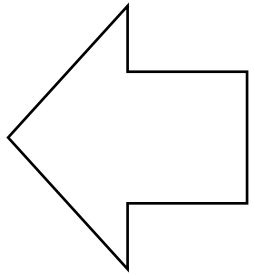
Architecture - Basic Optimizations

load registers

do operations

save registers

```
ld.w    $S0,    [ cpu.regs[0] ]
ld.w    $S1,    [ cpu.regs[1] ]
ld.w    $S3,    [ cpu.regs[3] ]

add.w   $S0,    $S3
# EFLAGS CALCULATION
add.w   $S0,    $S1
# EFLAGS CALCULATION
# cpu.eip = addr + 4

st.w    $S0,    [ cpu.regs[0] ]
st.w    $S1,    [ cpu.regs[1] ]
st.w    $S3,    [ cpu.regs[3] ]
```



i386

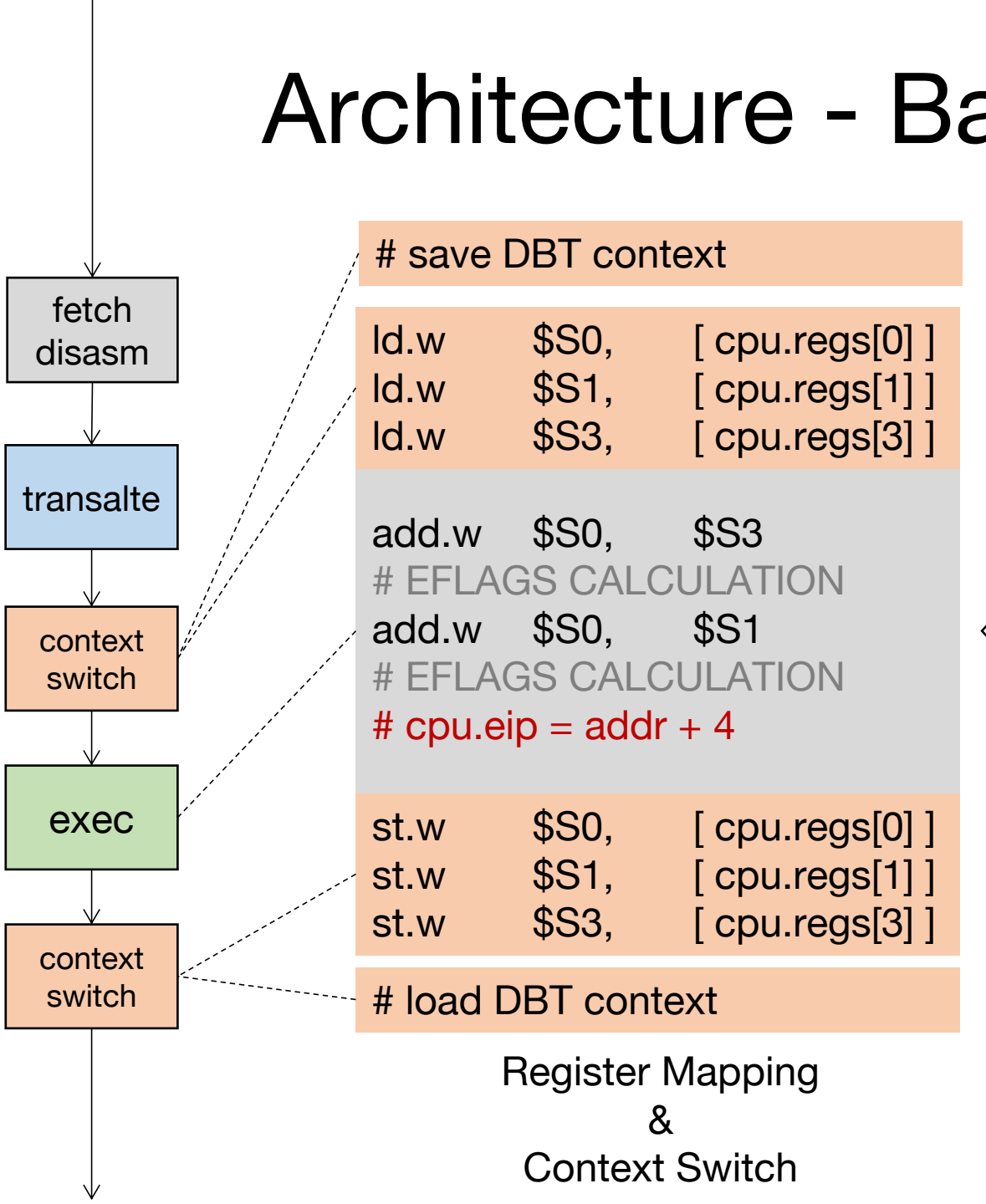
```
addr    add     %eax,   %ebx
addr+2  add     %eax,   %ecx
```

loongarch

```
ld.w    $T0,    [ cpu.regs[0] ]
ld.w    $T1,    [ cpu.regs[3] ]
add.w   $T0,    $T1
st.w    $T0,    [ cpu.regs[0] ]
# EFLAGS CALCULATION
ld.w    $T0,    [ cpu.regs[0] ]
ld.w    $T1,    [ cpu.regs[3] ]
add.w   $T0,    $T1
st.w    $T0,    [ cpu.regs[0] ]
# EFLAGS CALCULATION
# cpu.eip = addr + 4
```

Register Mapping

Architecture - Basic Optimizations




```
# save DBT context

ld.w    $S0,    [ cpu.regs[0] ]
ld.w    $S1,    [ cpu.regs[1] ]
ld.w    $S3,    [ cpu.regs[3] ]

add.w   $S0,    $S3
# EFLAGS CALCULATION
add.w   $S0,    $S1
# EFLAGS CALCULATION
# cpu.eip = addr + 4

st.w    $S0,    [ cpu.regs[0] ]
st.w    $S1,    [ cpu.regs[1] ]
st.w    $S3,    [ cpu.regs[3] ]

# load DBT context
```

Register Mapping
&
Context Switch

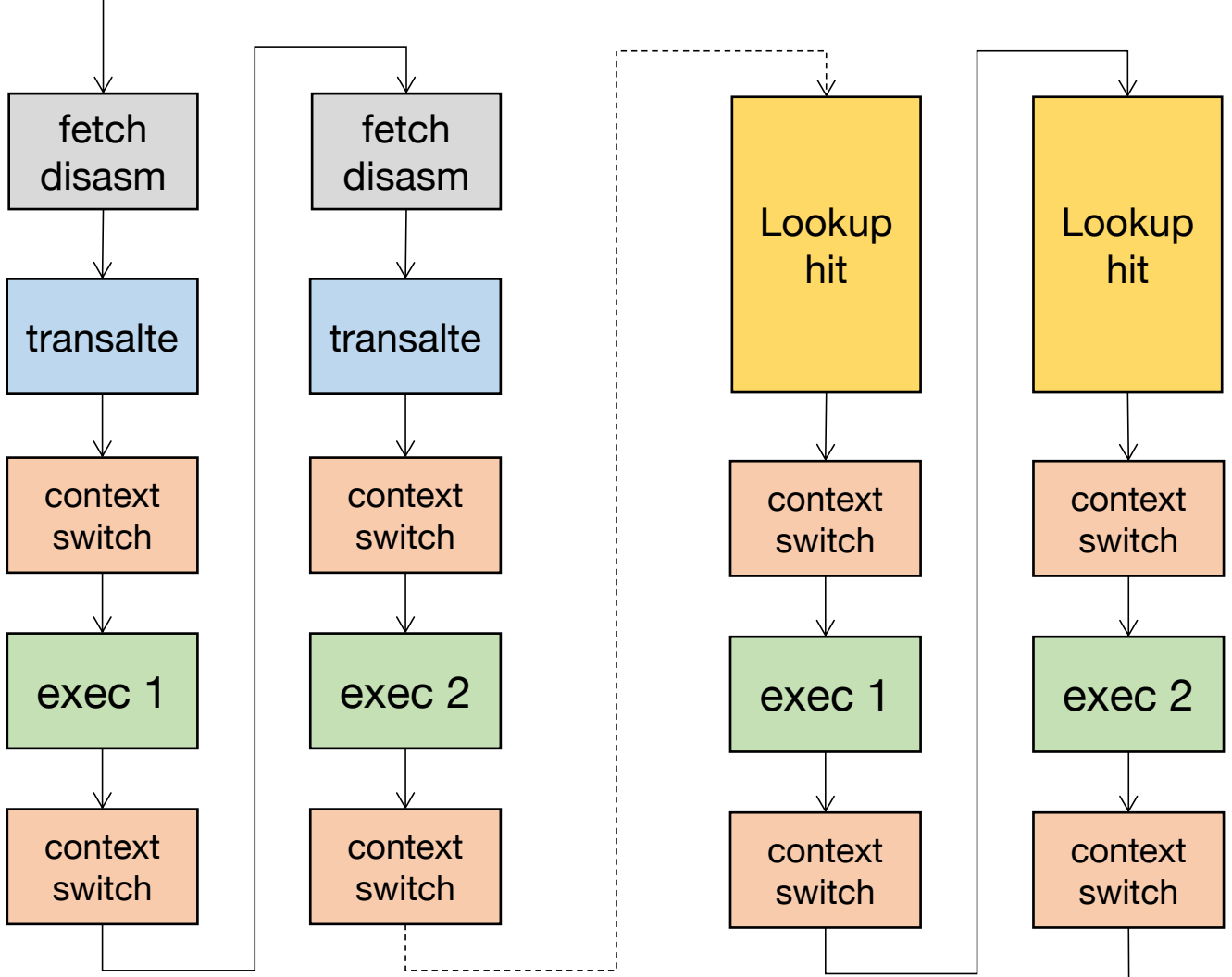
i386

```
addr    add     %eax,   %ebx
addr+2  add     %eax,   %ecx
```

loongarch

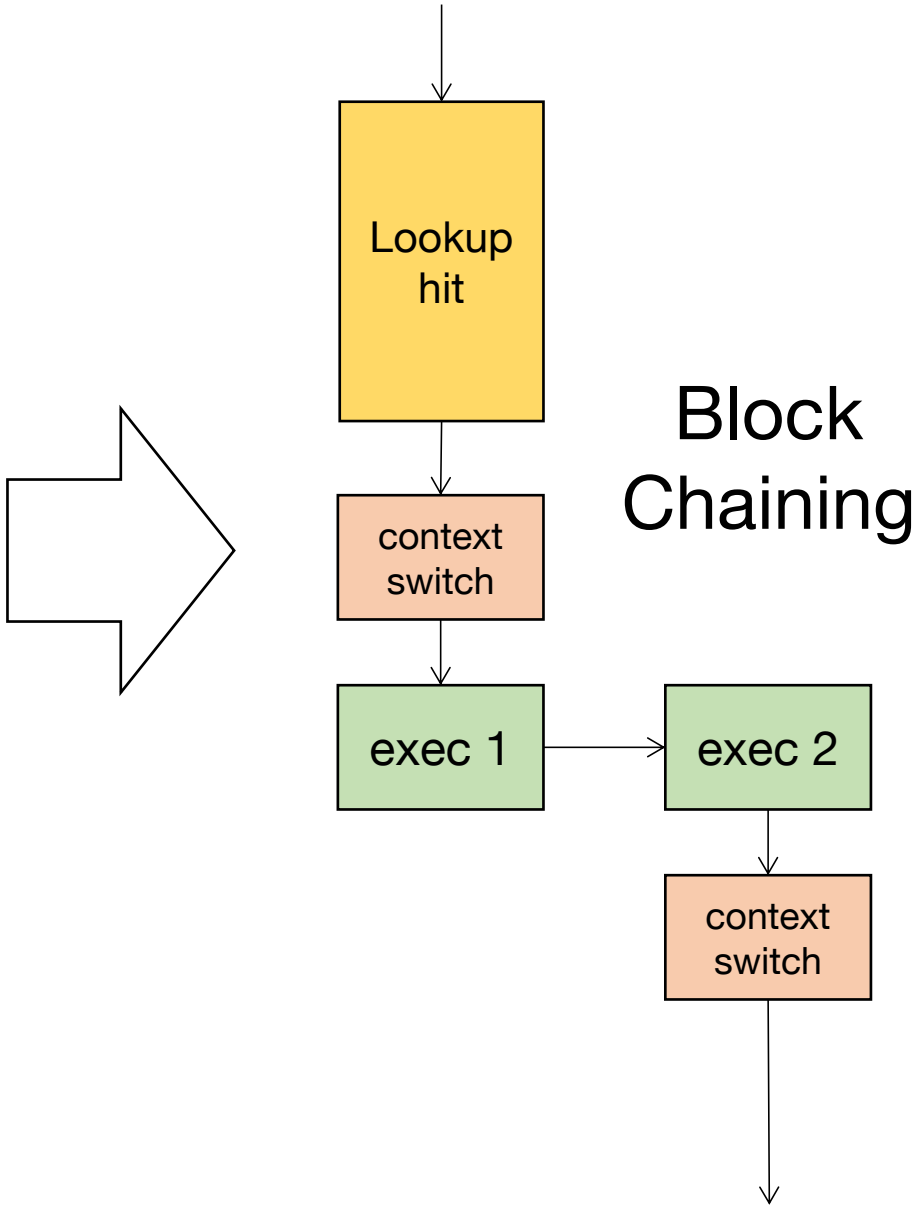
```
ld.w    $T0,    [ cpu.regs[0] ]
ld.w    $T1,    [ cpu.regs[3] ]
add.w   $T0,    $T1
st.w    $T0,    [ cpu.regs[0] ]
# EFLAGS CALCULATION
ld.w    $T0,    [ cpu.regs[0] ]
ld.w    $T1,    [ cpu.regs[3] ]
add.w   $T0,    $T1
st.w    $T0,    [ cpu.regs[0] ]
# EFLAGS CALCULATION
# cpu.eip = addr + 4
```

Architecture - Basic Optimizations



(1) Not translated yet

(2) Lookup hit after translation



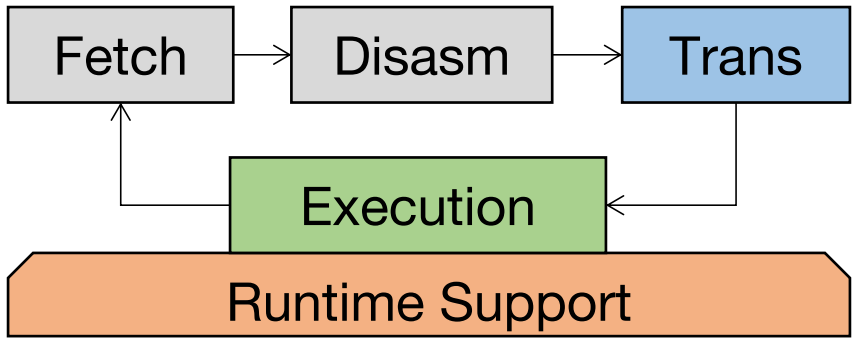
Block Chaining

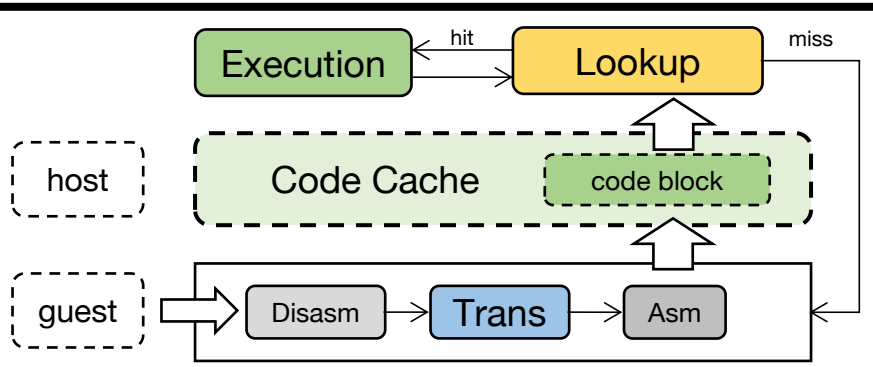
Architecture - Basic Optimizations

- Code Cache ..... remove retranslation
- Code Block ..... reduce PC update
  - Basic Block
  - Translation Block
  - Code Fragment
- Register Mapping ..... reduce load / store
  - context switch
- Block Chaining ..... reduce context switch / lookup
  - direct jmp and branch

A Survey on Binary Translation

Binary Rewrite	=	Translation	,	Optimization	,	Instrumentation
Script Language	=	JIT / Java / JVM	,	Interpret		Python, Matlab
Virtualization	=	KVM	, Wine	, Android	,	Gem5





4

Advanced Topics

5

List of Binary Translation Tools Papers

6

Summary

A Survey on Binary Translation

- Introduction ..... basic architecture

- Related Technologies ..... differences and similarities
- Architecture Details ..... and basic optimizations
- Advanced Topics ..... problems, challenges, optimizations, etc.

- List of Binary Translation ..... by year, type, etc.
- Summary

Advanced Topics

- Optimizations
  - Branch, Code Cache and TB Lookup
  - Optimize Generated Codes
  - Special Guest: FP, SIMD, Atomic, EFLAGS, JIT, Dynamic Library, Syscall, ...
  - System-mode: Memory, Interrupt, vCPU, ...
  - Course-Grained Reconfigurable Architectures
- Problems
  - Performance !
  - SMC
  - Precise Exception
  - ABI
  - Memory Consistency

1. optimize BT to run faster
2. optimize Translation to generate better codes to run faster

correctness

efficiency

sercutiry

Optimization - Indirect Branch

- ... the most of the overhead is caused by RAS/code duplication related overhead and address resolution. [1]
  - RAS: lack of RAD for call and return
  - Code duplication: increased cache miss / branch miss
  - Address Resolution: inlined block lookup (indirect jmp, call, ret, ...
- ... the performance overhead can be attributed to frequent exits from the code cache to ... [2]
  - ... the major causes for code cache exits include block translation, trace formation, and indirect branch resolution. [2]

[1] 2009. Edson Borin. Characterization of DBT overhead

[2] 2015. Surya Tej Nimmakayala. Exploring Causes of Performance Overhead During Dynamic Binary Translation

Optimization - Indirect Branch

- ... indirect branches are optimistically transformed into direct branchs ... referred to as the predicted indirect branch target. [1]
  - If the comparison succeeds, control goes to the preficted target.
  - If the comparison fails, control is directed to a special Dynamo routine that looks up a Dynamo-maintained switch table.
- ... makeing use of run time information to convert each indirect branch to a set to conditional branches. [2]

```
If Rx == #addr_1 goto #target_1
Else if Rx == #addr_2 goto #target_2
Else if Rx == #addr_3 goto #target_3
Else hash_lookup(Rx); do it the slow way
```

[3]

[1] 2000. Dynamo. A Transparent Dynamic Optimization System

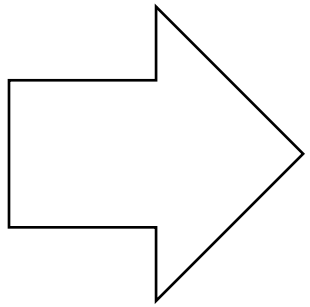
[2] 2001. DAISY. Dynamic Binary Translation and Optimization

[3] 2003. Ho-Seop Kim. Hardware Support for Control Transfers in Code Caches

Optimization - Indirect Branch

- For a given indirect branch, we add a private buffer to cache its all previous target address. [1]
  - The private buffer is a small hash table,

```
if(branch_target==address1) {
        goto TPC1;
}else if(branch_target==address2) {
        goto TPC2;
}else if(branch_target==address3) {
        goto TPC3;
}else goto lookup_module( ) ;
```

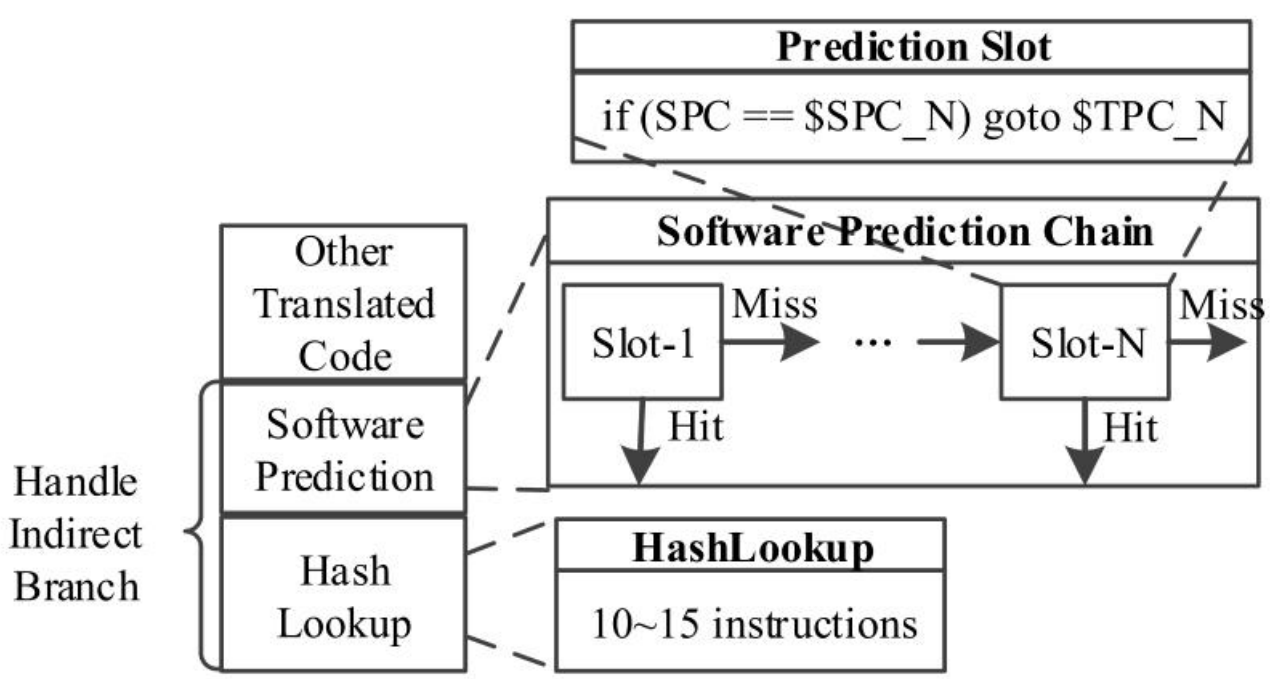


```
case an indirect branch:
    1, load private buffer of current TB.1;
    2, calculate the index by the target address of the
    indirect branch;

    3, load TB.2 from private buffer  of current TB.1by
    the index;

    4, if (target address == first source binary instruction
    address of the TB.2 ){
        go to the code cache address of TB.2;
    }else{
        go to the lookup module.;
    }
```

[1] 2012. Liao Yin. Improve Indirect Branch Prediction with Private Cache in Dynamic Binary Translation

Optimization - Indirect Branch

- This papar analyzes the performance bootleneck of software prediction, and proposes a novel prediction mechanism called Software Prediction with Target Updating (SPTU). [2]



miss counter
-> threshold
-> flush and update

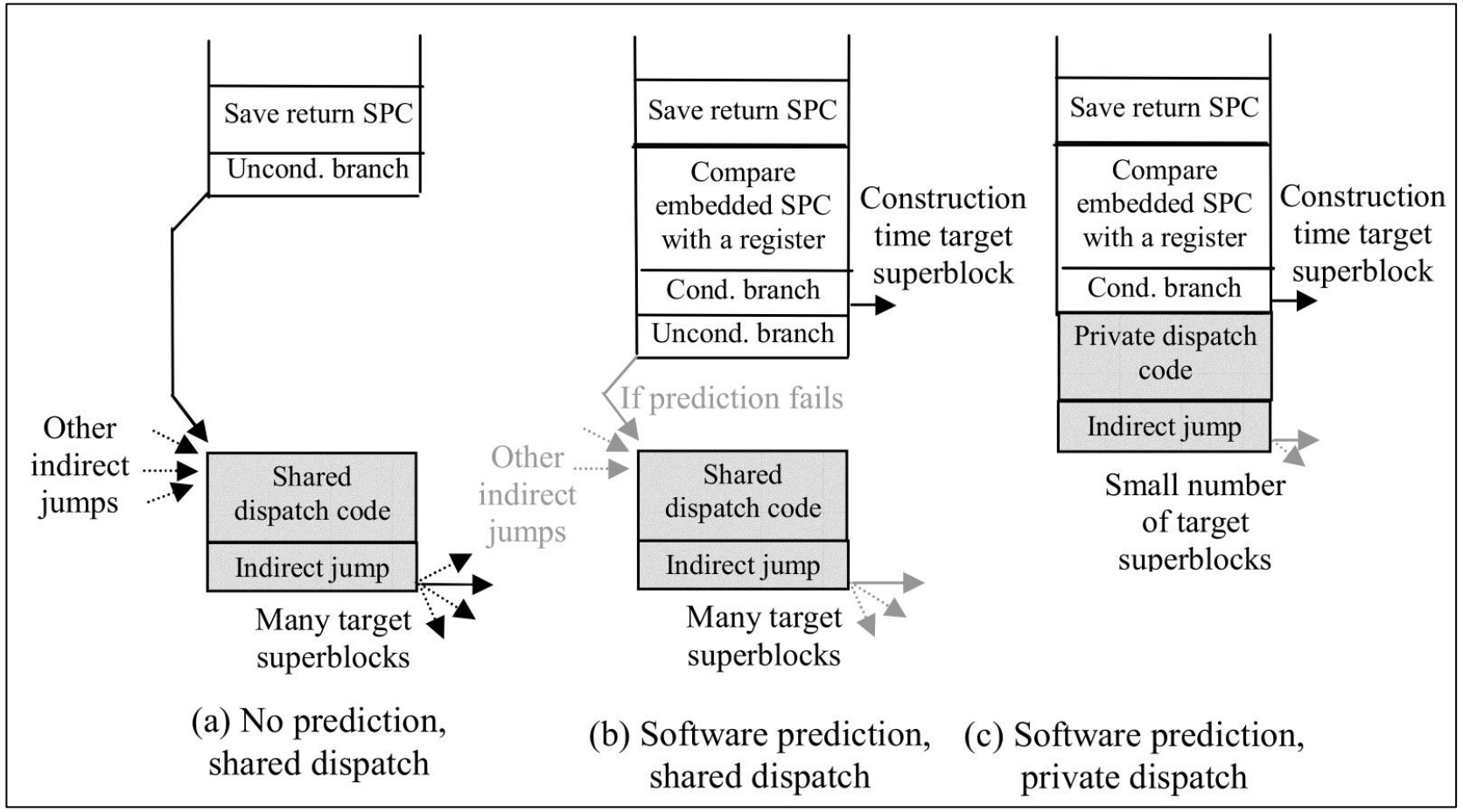
2012. Private Cache:
2013. Software Prediction [1]
=> insert on every miss

[1] 2013. Ning Jia. Software Prediction for Indirect Branch in Dynamic Translation Systems

[2] 2014. Ning Jia. SPTU: Improving Dyanmic Binary Translation through software Prediction with Target Updating

Optimization - Indirect Branch

- Software prefiction private dispatch [1]

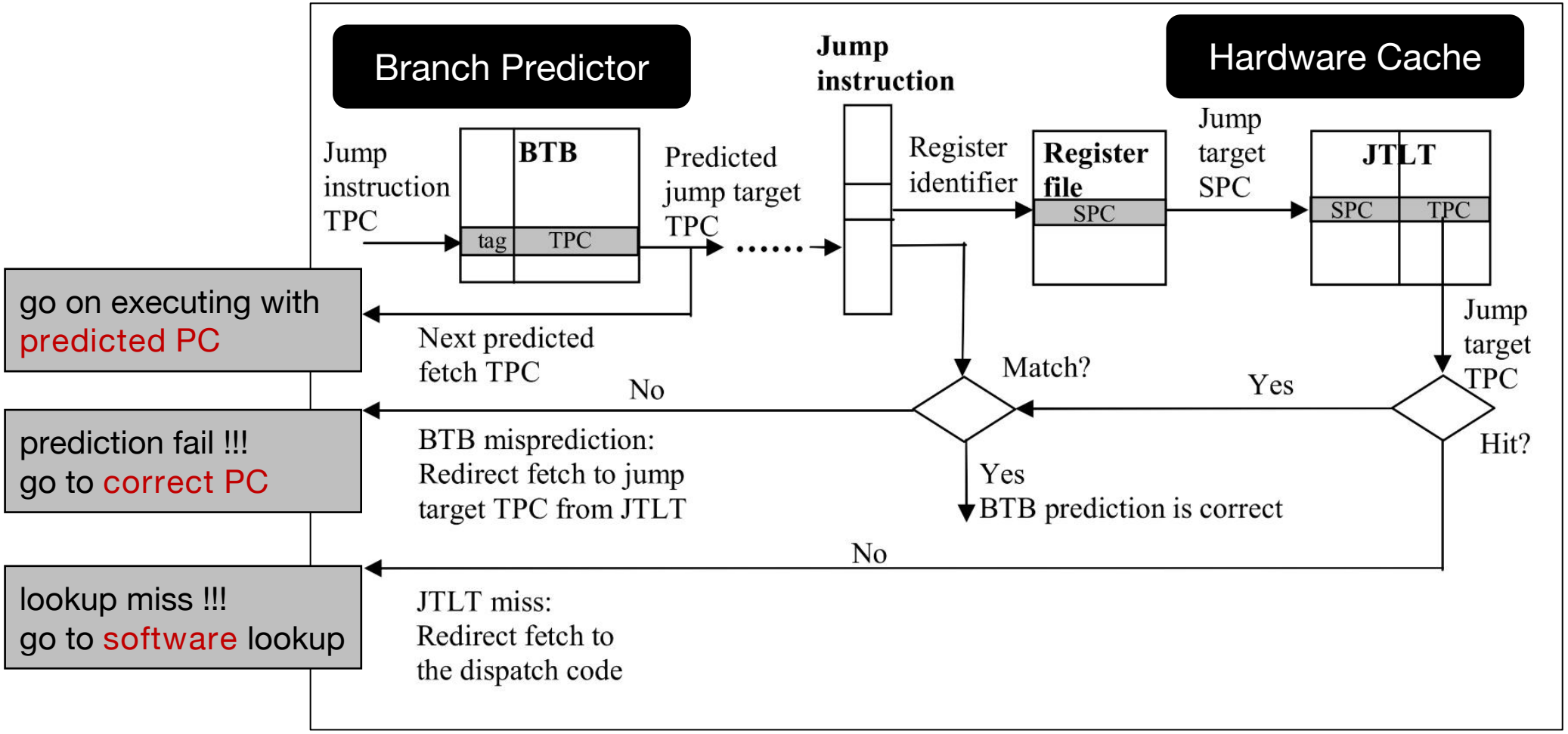


[1]

[1] 2003. Ho-Seop Kim. Hardware Support for Control Transfers in Code Caches

Optimization - Indirect Branch

- ... maintain a hardware cache of dispatch table entries. [1]



[1]

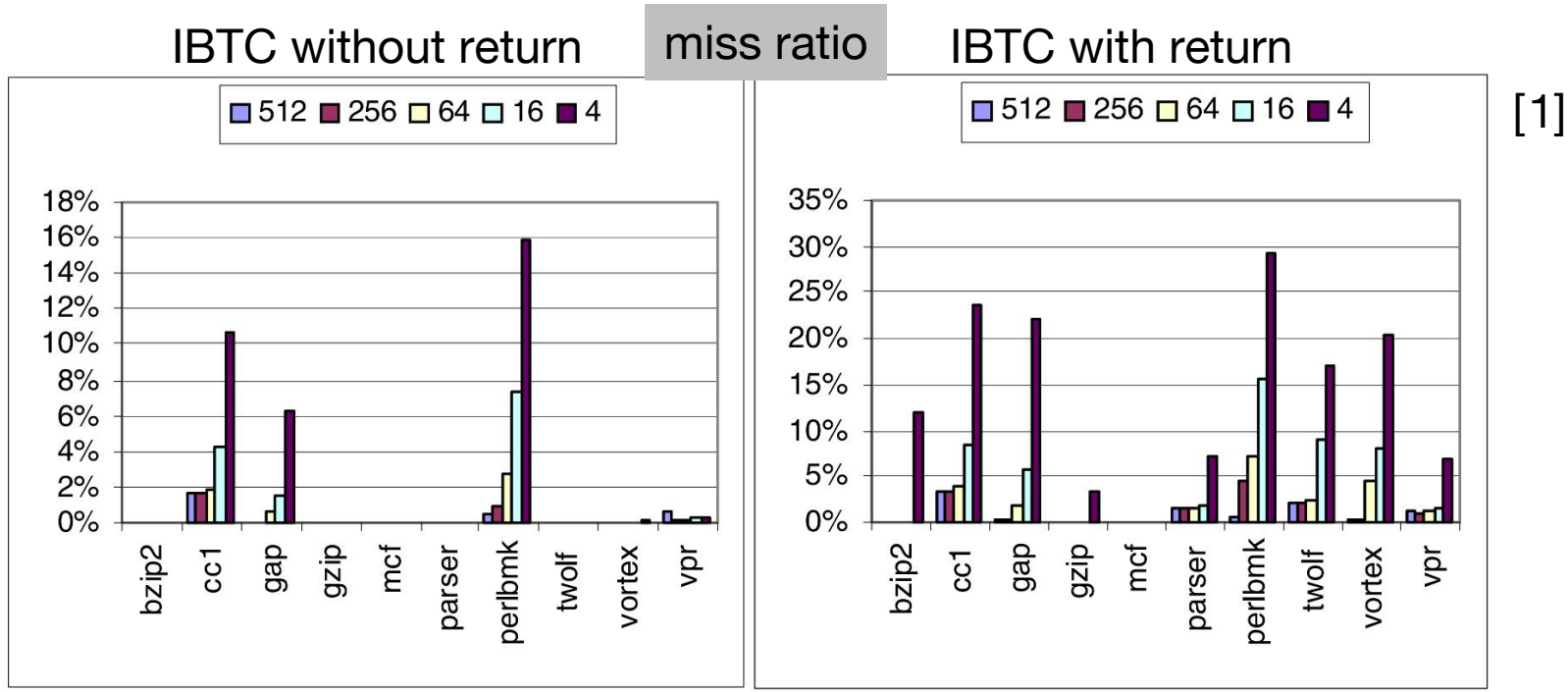
[1] 2003. Ho-Seop Kim. Hardware Support for Control Transfers in Code Caches

Optimization - Indirect Branch

- An IBTC is a small, direct-mapped cache that maps branch-target addresses to their fragment cache locations. [1]
  - an IBTC is a simpler structure, and much faster to consult.

GVA[9:0]

GVA_1	HVA_1
GVA_2	HVA_2
GVA_3	HVA_3
GVA_4	HVA_4
...	...
GVA_n	HVA_n



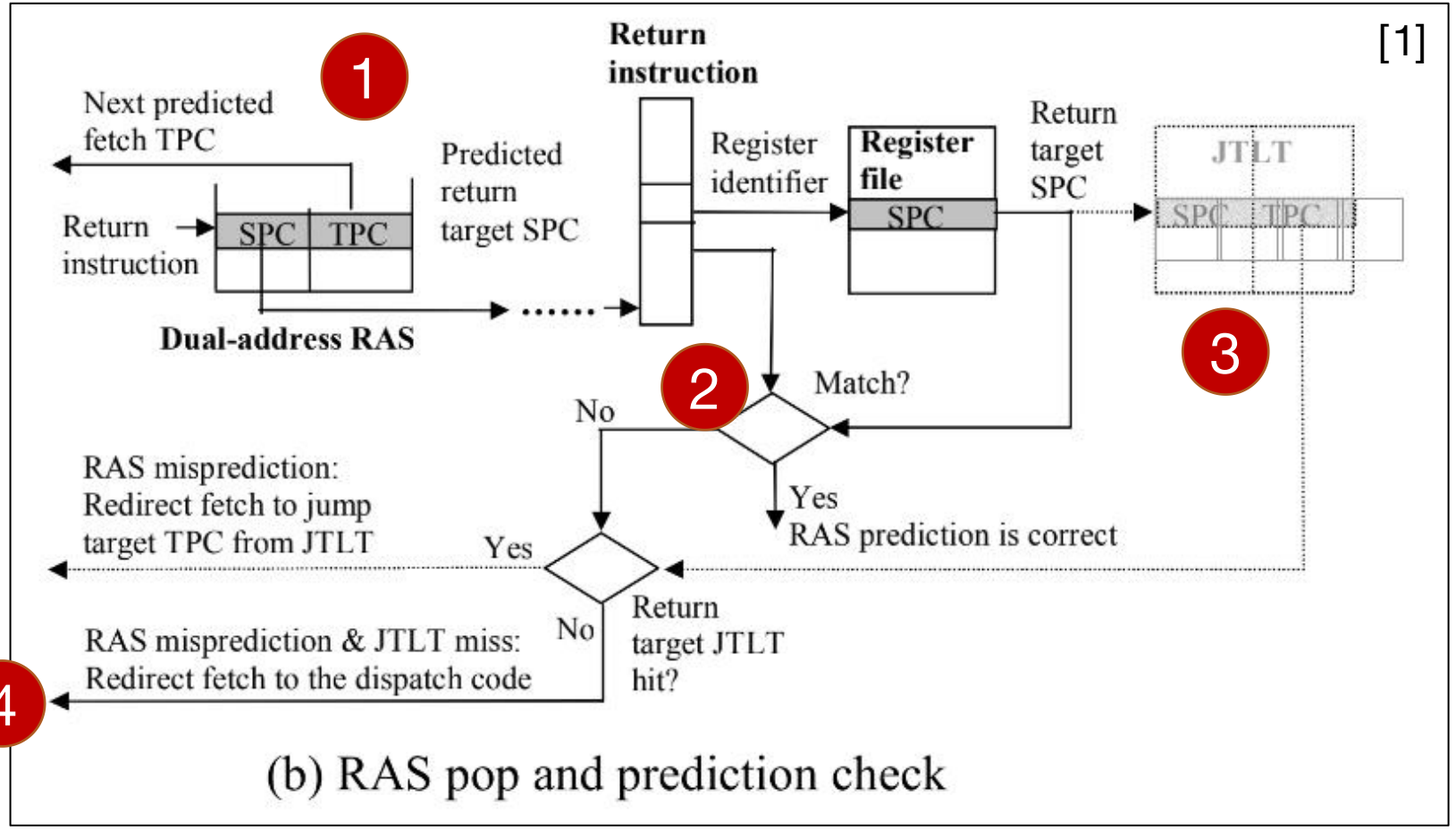
[1]

[1] 2004. K.Scott. Overhead Reduction Techniques for Software Dynamic Translation. (IPDPS)

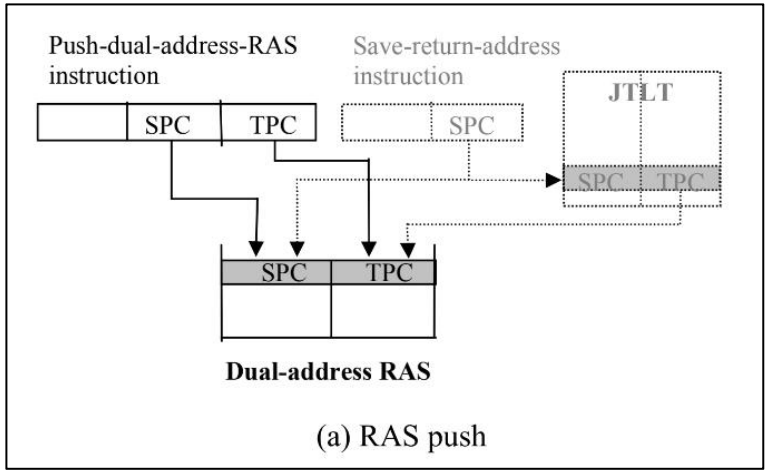
[2] 2007. Jason D.Hiser. Evaluating Indirect Branch Handling Mechanisms is Software Dynamic Translation Systems.

Optimization - Indirect Branch

- Dual-address return address stack [1]



[1]



(1) go on executing with PC poped from stack (Predicted PC!)

(2) compare SPC from stack with SPC from instruction

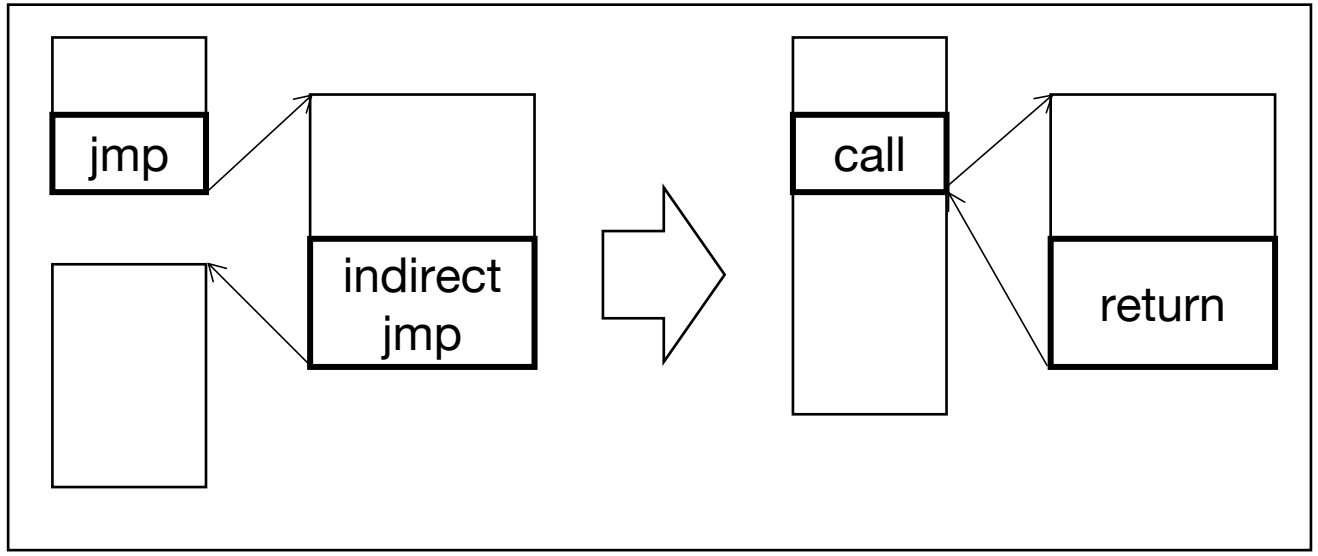
(3) lookup hardware cache

(4) go to software lookup

[1] 2003. Ho-Seop Kim. Hardware Support for Control Transfers in Code Caches

Optimization - Indirect Branch

- Hardware-assisted function returns use a software return address stack to predict the targets of function returns. [1]
  - ... not ending a basic block when a call instruction is encountered.
- Branch table inference, an algorithm for detecting and translating branch tables. [1]
  - C compilers commonly generate branch tables for large switch statements.



Original C code

```
switch (val) {
    case 0: ...
    case 1: ...
    case 2: ...
    case 3: ...
    default: ...
}
```

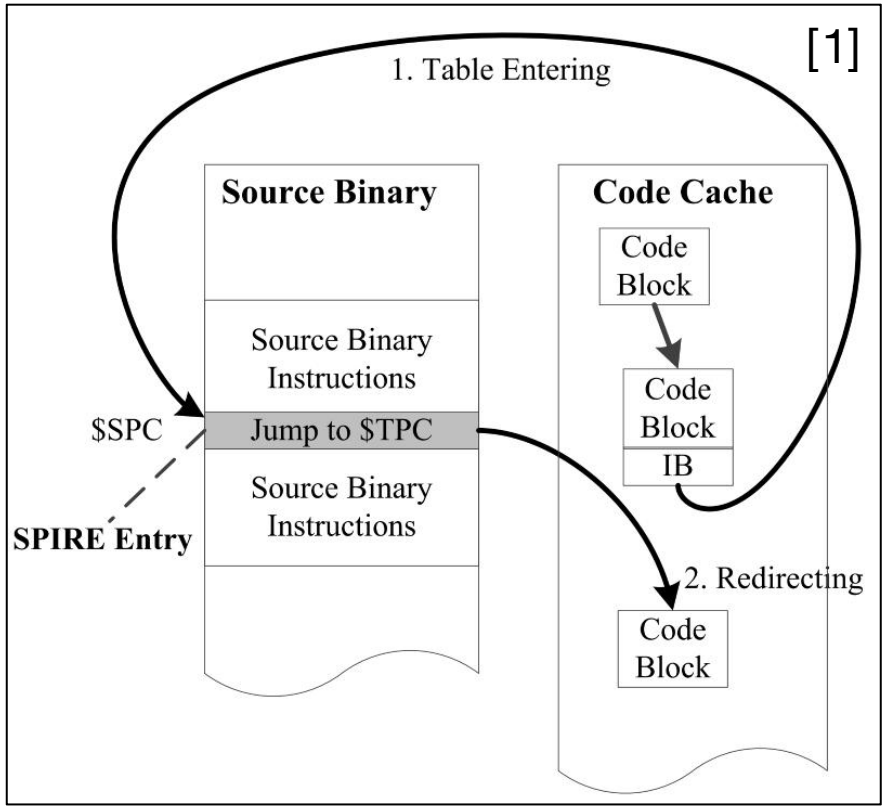
ARM branch table

```
CMP R0, #3
BHI default
ADR R1, table
LSL R0, R0, #2
LDR PC, [R0, R1]
table:
.word case0
.word case1
.word case2
.word case3
```

[1] 2016. Amanieu D'Antras. Optimizing Indirect Branches in Dynamic Binary Translators

Optimization - Indirect Branch

- It reuses the source binary code space to build a SPC-indexed redirecting table. [1]
- ... dynamically write a Virtual Branch Instruction into the Source Binary Blocks ... [2]



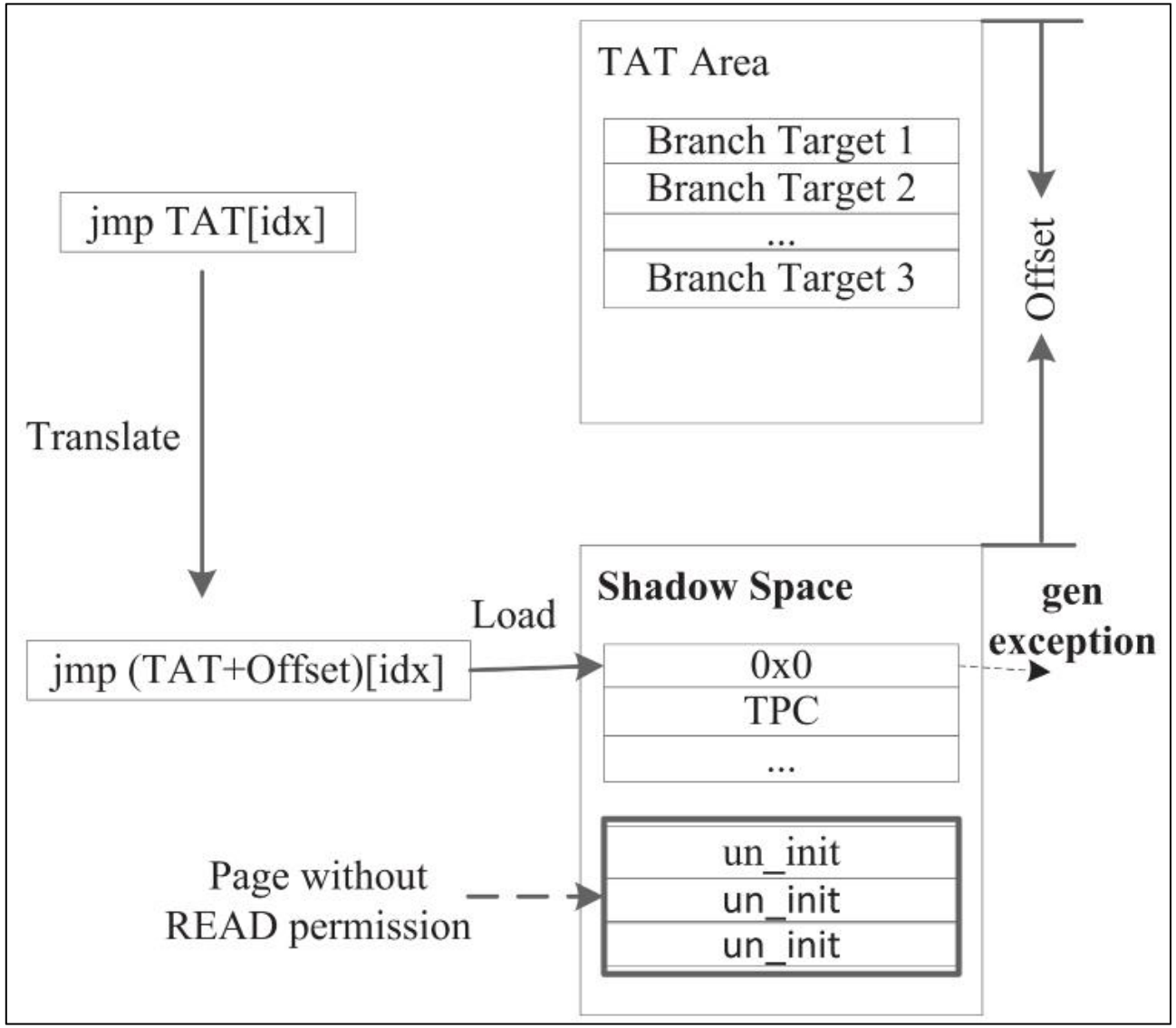
- not use before fill & fill after first use
- overlap in source binary
- source binary correctness
  - self-modified code
  - retranslation

[1] 2013. Ning Jia. SPIRE: Improving Dynamic Binary Translation through SPC-Indexed Indirect Branch Redirecting

[2] 2013. Xiaochun Zhang. VBIW: Optimizing Indirect Branch in Dynamic Binary Translation

Optimization - Indirect Branch

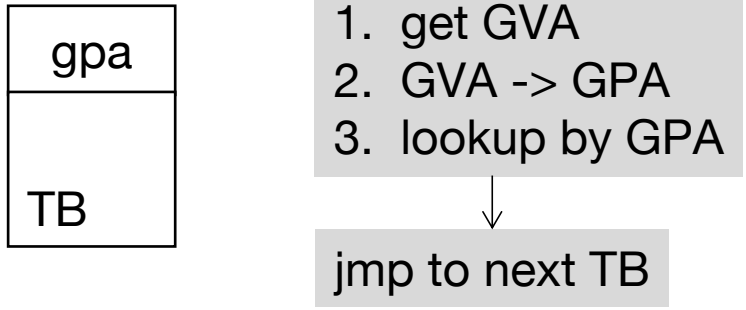
- ... most of the branch targets are prestored in the program's memory as some kind of address tables. [1]
  - switch-case statements
  - virtual function call
  - function pointer
    - as an element of other structure



[1] 2014. Ning Jia. DTT: Program Structure-Aware Indirect Branch Optimization via Direct-TPC-Table in DBT system

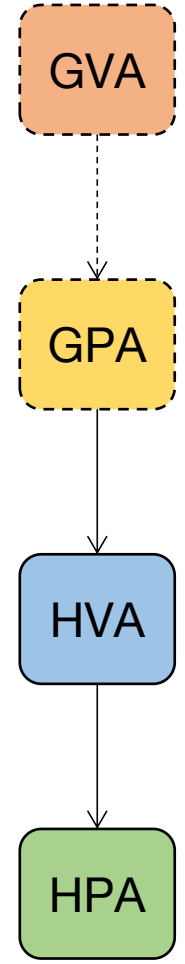
Optimization - Indirect Branch (sys mode)

[1]

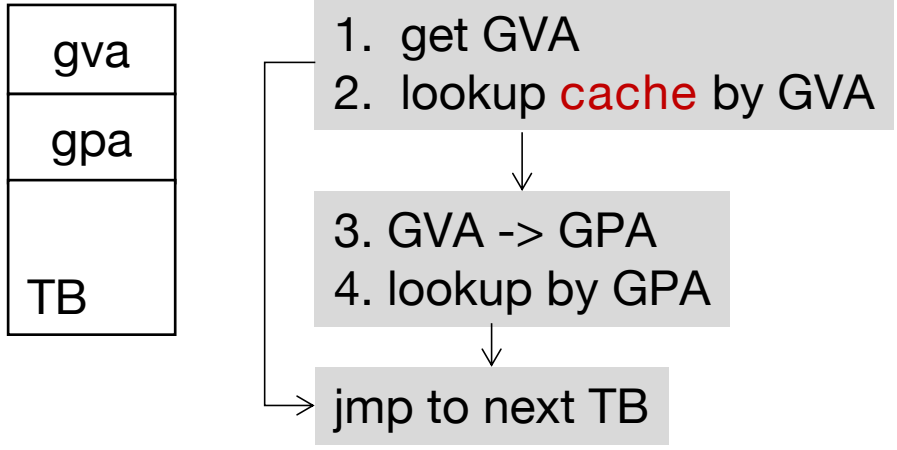


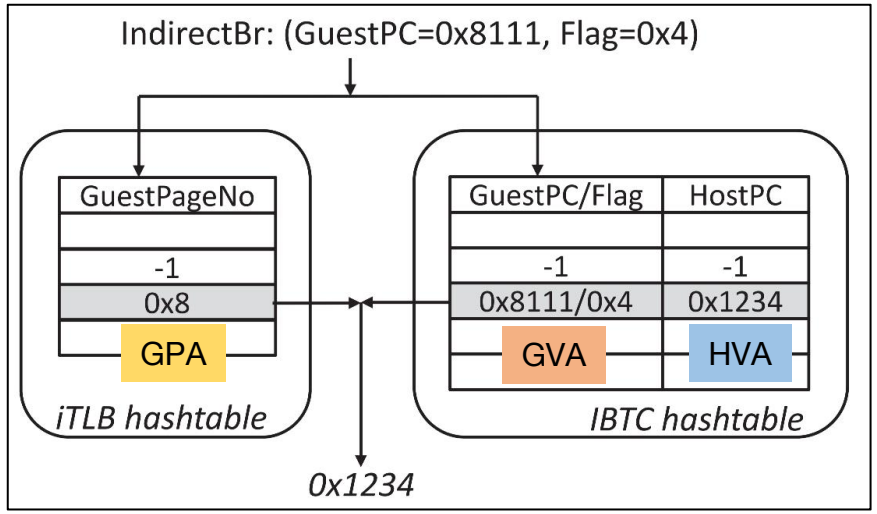

```
# set base address to Rm
# e.g. function entry point
v0x10400: set current-pc to Rm (1)
          ...
          branch to Rm+0x1000 (2)
v0x11400: ...
          branch to Rm+0x2000 (3)
v0x12400: ...
```

[1] ... use an offset from the virtual address of each page that contains a branch ... simplifies the guard code necessary for an indirect branch. [1]



[2]





[2]

[1] 2012. Toshihiko Koju. Optimizing Indirect Branches in a System-level Dynamic Binary Translator. (IBM)

[2] 2015. Optimizing Control Transfer and Memory Virtualization in Full System Emulators. (TACO)

Optimization - Indirect Branch

[01] 2000. Dynamo. A Transparent Dynamic Optimization System

[02] 2001. DAISY. Dynamic Binary Translation and Optimization

[03] 2003. Ho-Seop Kim. Hardware Support for Control Transfers in Code Caches

[04] 2003. Ho-Seop Kim. Hardware Support for Control Transfers in Code Caches

[05] 2007. Jason D.Hiser. Evaluating Indirect Branch Handling Mechanisms is Software Dynamic Translation Systems

[06] 2009. Edson Borin. Characterization of DBT overhead

[07] 2012. Liao Yin. Improve Indirect Branch Prediction with Private Cache in Dynamic Binary Translation

[08] 2012. Toshihiko Koju. Optimizing Indirect Branches in a System-level Dynamic Binary Translator. [IBM]

[09] 2013. Ning Jia. SPIRE: Improving Dynamic Binary Translation through SPC-Indexed Indirect Branch Redirecting

[10] 2013. Xiaochun Zhang. VBIW: Optimizing Indirect Branch in Dynamic Binary Translation

[11] 2014. Ning Jia. DTT: Program Structure-Aware Indirect Branch Optimization via Direct-TPC-Table in DBT system

[12] 2015. Optimizing Control Transfer and Memory Virtualization in Full System Emulators. (TACO)

[13] 2015. Surya Tej Nimmakayala. Exploring Causes of Performance Overhead During Dynamic Binary Translation

[14] 2016. Amanieu D'Antras. Optimizing Indirect Branches in Dynamic Binary Translators

Optimization - Code Cache

- evaluates several alternative cache management schemes that identify and remove only enough traces to make room for a new trace. [1]
  - Full Cache Flush
  - Least-Recently Accessed
  - Least-Frequently Accessed
  - Least-Recently Created (FIFO)
  - Largest Element
  - Best-Fit Element

LRC: 2.88%

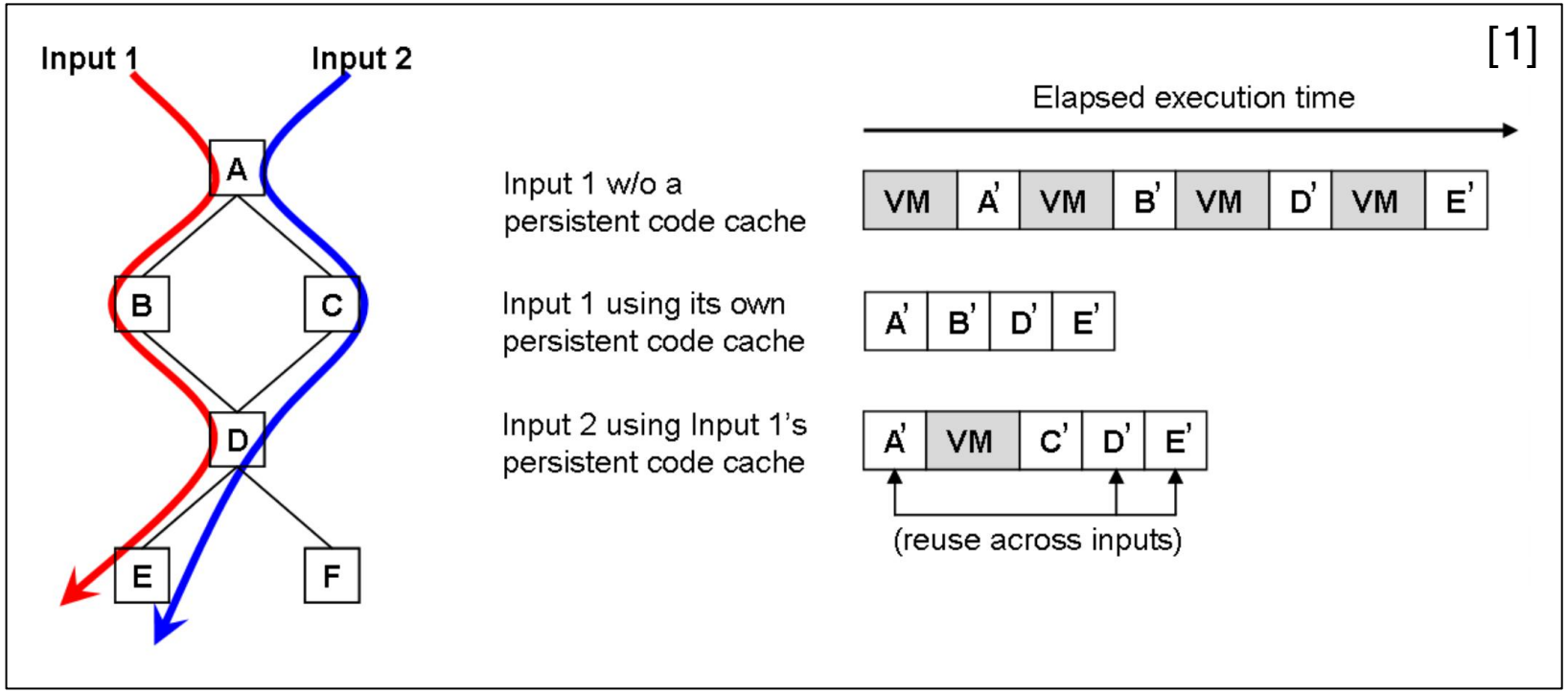
Flush: 4.61%

scheme	fragment-ation	additional victims	miss rate	management
LRA	Yes	71%	2.48%	priority queue
LFA	Yes	41%	9.11%	priority queue
LRC	None	78%	2.88%	pointer
LE	Yes	21%	13.91%	priority queue
BFE	Minimal	16%	20.77%	multiple sorted lists
Flush	None	N/A	4.61%	pointer

[1] 2002. Kim Hazelwood. Code Cache Management Schemes for Dynamic Optimizers

Optimization - Code Cache

- ... intra-execution model of code reuse by storing and reusing translations across executions, [1]
  - base address, size
  - binary path
  - program header
  - timestamps
- ... code cache sharing among processes ... [2]
- ... to redue the translation overhead [3]



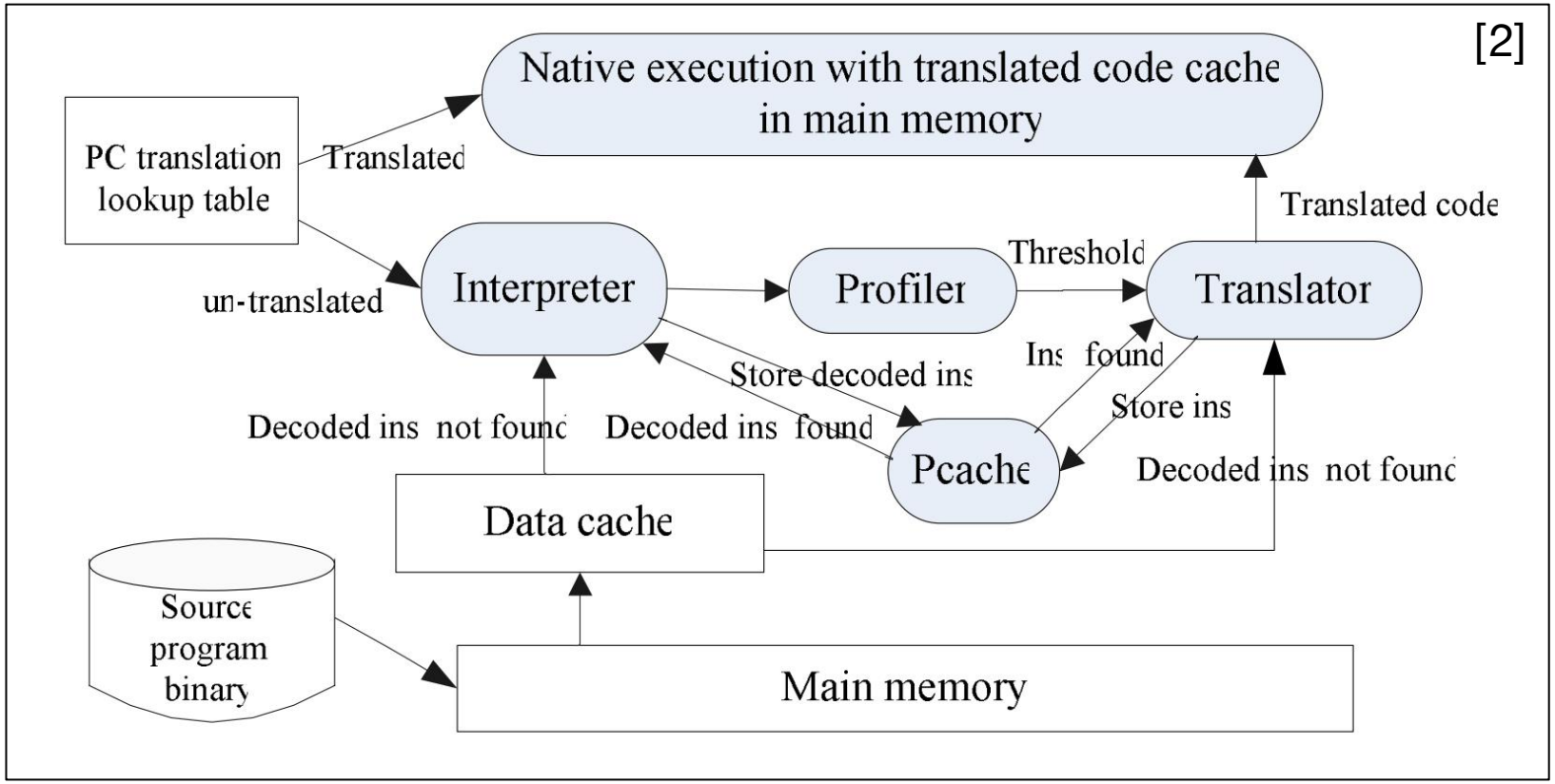
[1] 2007. Dan Connors. Persistent Code Caching: Exploiting Code Reuse Across Executions and Applications

[2] 2008. Derek Bruening. Process-Shared and Persistent Code Caches

[3] 2016. Wenwen Wang. A General Persistent Code Caching Framework for Dynamic Binary Translation

Optimization - Code Cache

- ... interpreted code cache (Pcache), a hardware assist, to save the information of the decoded instruction for reuse. [1]
- ... add the translation routine entry into the Pcache design thus saving most decoding operations during translation. [2]



[1] 2009. Wei Chen. A Hardware Approach for Reducing Interpretaion Overhead. (CIT'09)

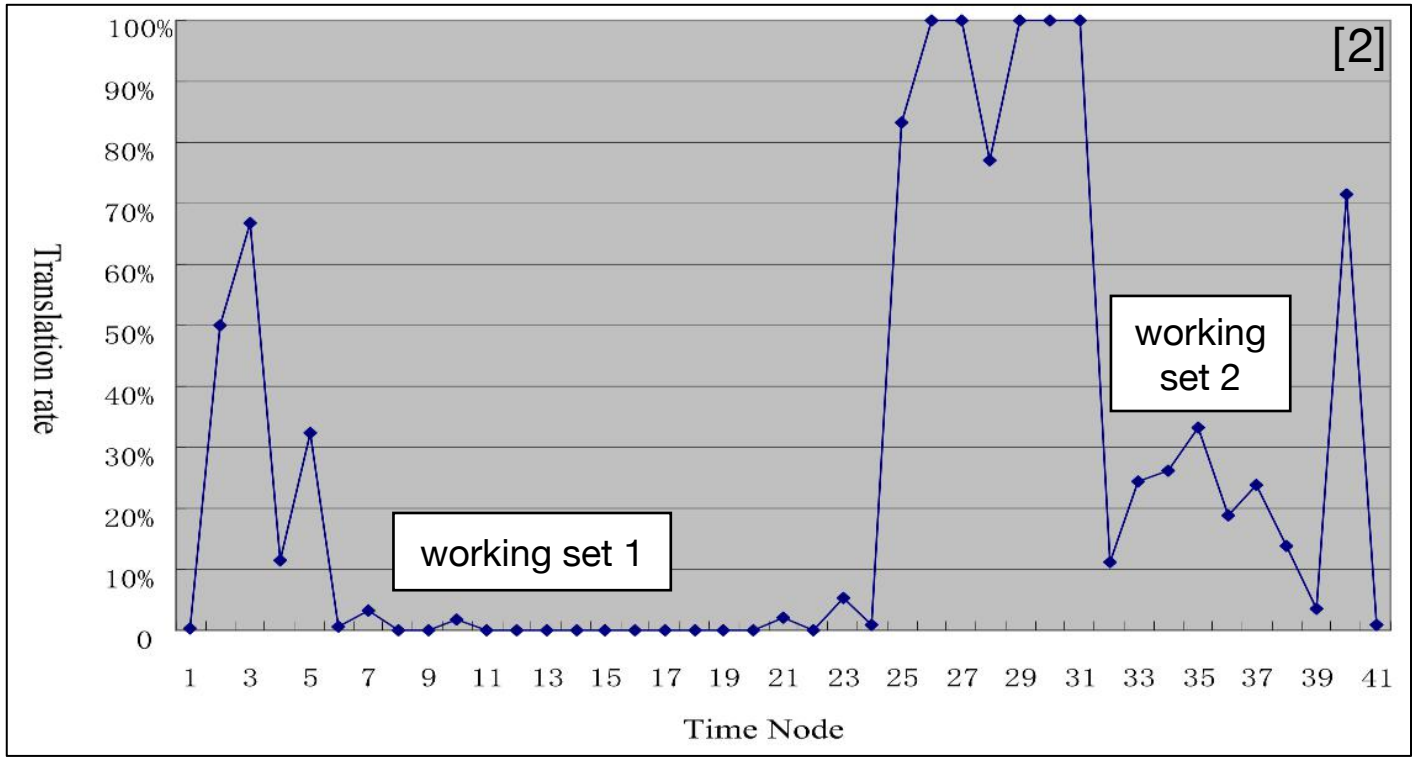
[2] 2009. Wei Chen. A Light-weight Code Cache Design for Dynamic Binary Translation. (ICPADS'09)

Optimization - Code Cache

- ... defined working set as the set of blocks that run recently. [1]
- Static Code Cache (SCC)
- Dynamic Code Cache (DCC)

$$T_{rate}=N_{TranslationBlock} / N_{ExecutionBlock}*100\%$$ [2]

- The size of code cache is assigned about 32 KB
  - embedded system



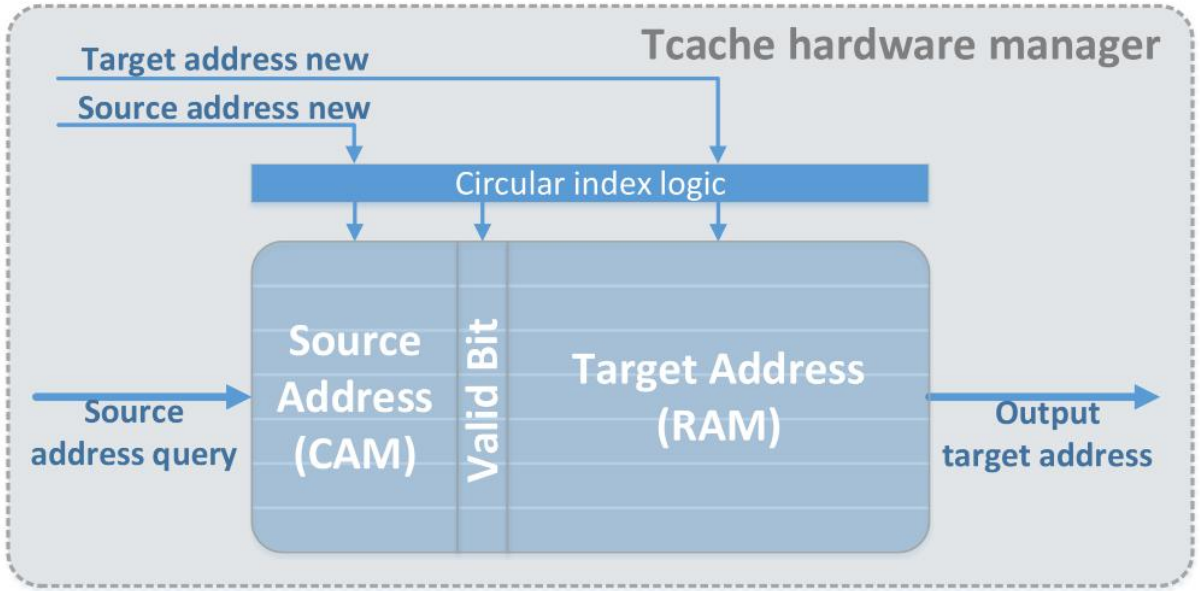
[1] 2001. Banerjia. Preemptive replacement strategy for a caching dynamic translator. (US Patent)

[2] 2011. Ruhui Ma. Code Cache Management Based on Working Set in Dynamic Binary Translator

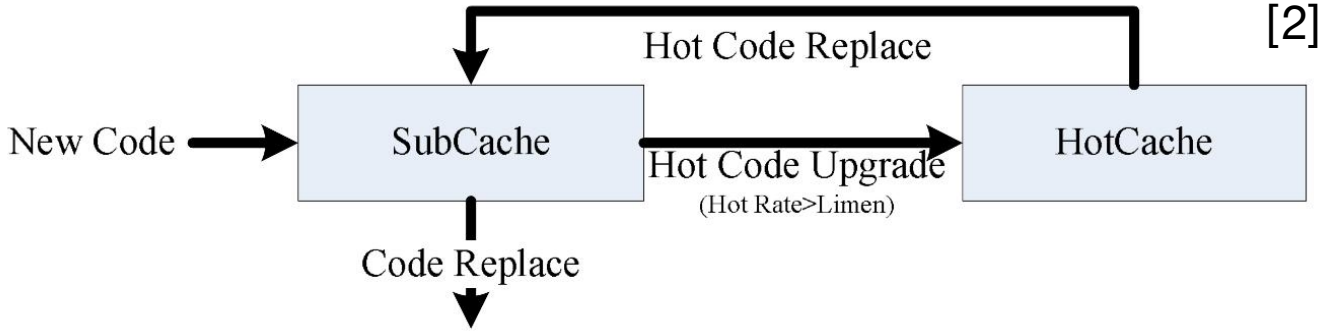
Optimization - Code Cache

[1]

- Hardware-Assisted Transaltion Cache [1]
  - CAM + RAM
- ... considered the temporal and spatial locality of the program execution and the replacement cost of Cache, ... [2]
  - BPMS profiled-based management strategy



[2]



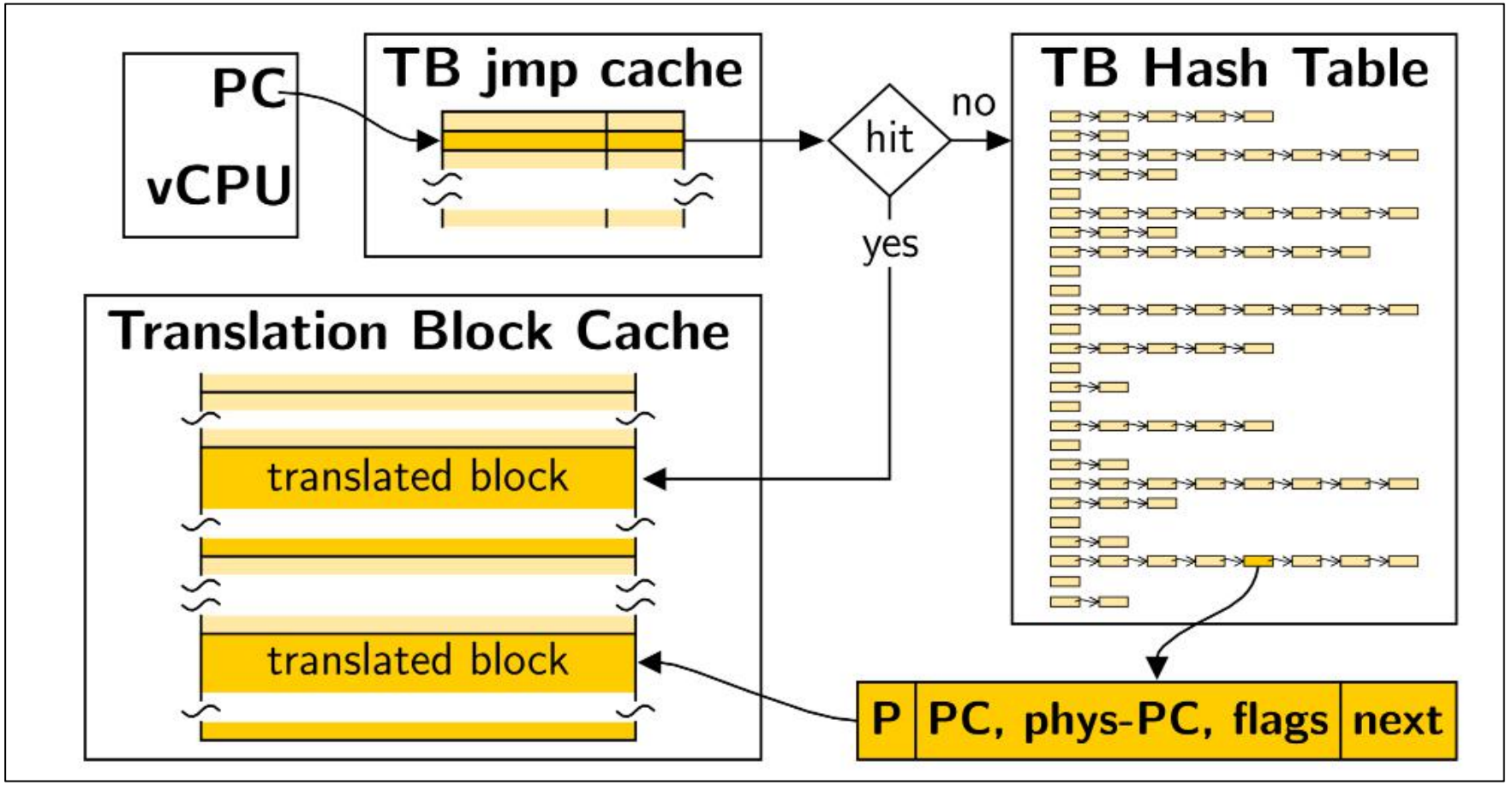
[1] 2018. Filipe Salgado. A hardware-assisted translation cache for DBT in embedded system

[2] 2018. Fei Deng. Research on CodeCache Management Strategy Based on Code Heat in DBT

Optimization - TB Lookup

- Translation Block lookup mechanisms in QEMU [1]
- TB jmp cache
  - index by GVA
- TB Hash Table
  - index by GVA

[1]

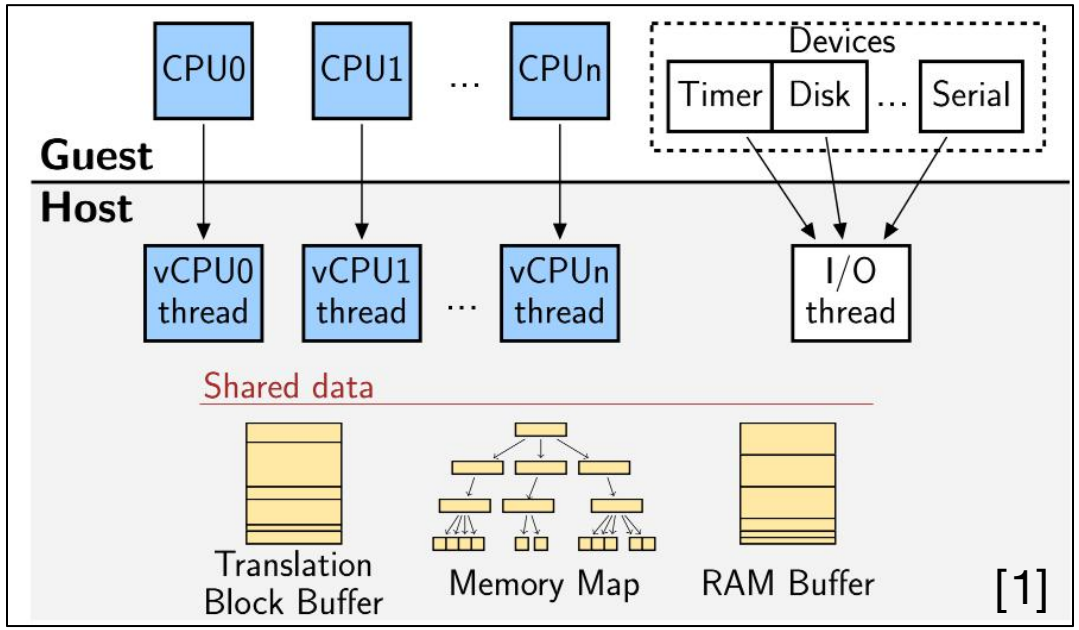


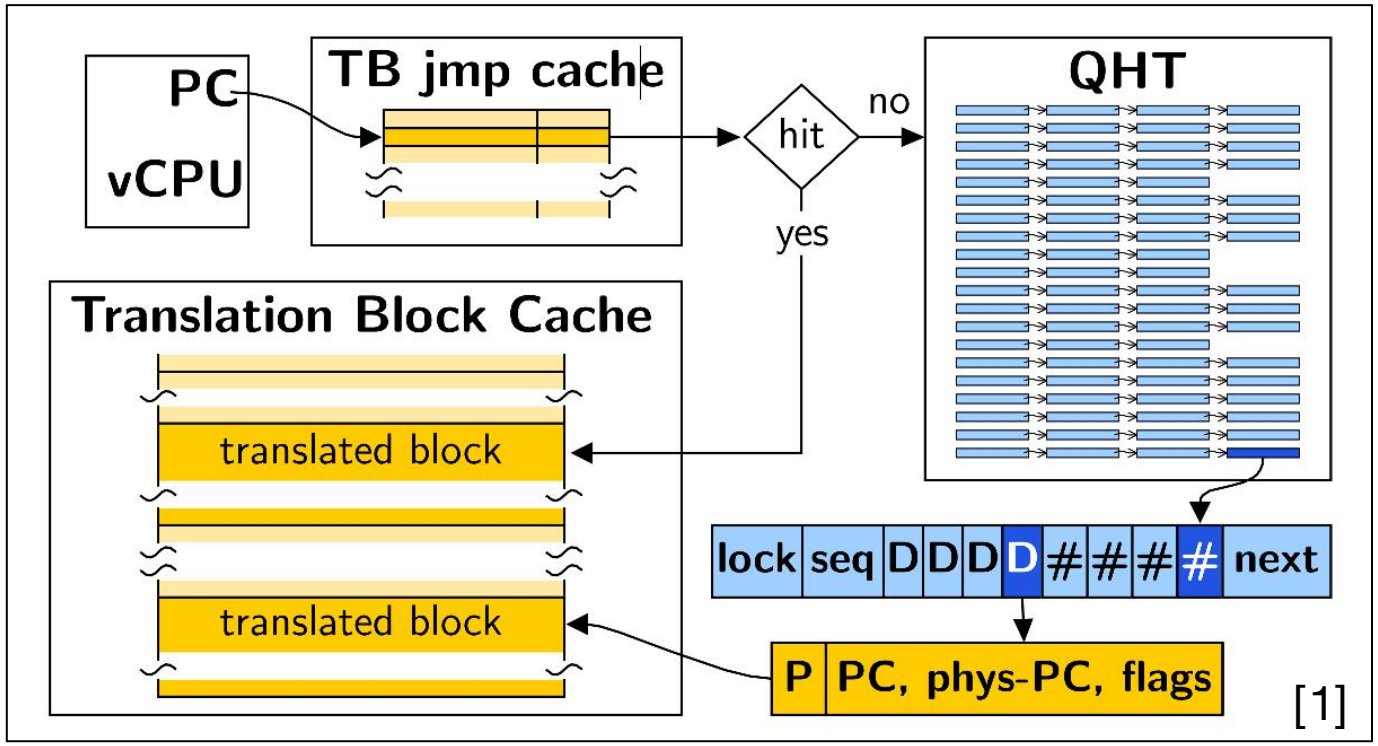
gva
gpa
TB

[1] 2017. Emilio G.Cota. Cross-ISA Machine Emulation for Multicores. CGO'17

Optimization - TB Lookup

- A memory-efficient design of a shared code cache for DBT engines. [1]
  - improve the hashing function used to place TB in the hash table
  - adopt a new hash table design that enables correct, concurrent lookups





[1] 2017. Emilio G.Cota. Cross-ISA Machine Emulation for Multicores. CGO'17

Optimization - Code Cache & TB Lookup

[01] 2001. Banerjia. Preemptive replacement strategy for a caching dynamic translator. (US Patent)

[02] 2002. Kim Hazelwood. Code Cache Management Schemes for Dynamic Optimizers

[03] 2007. Dan Connors. Persistent Code Caching: Exploiting Code Reuse Across Executions and Applications

[04] 2008. Derek Bruening. Process-Shared and Persistent Code Caches

[05] 2011. Ruhui Ma. Code Cache Management Based on Working Set in Dynamic Binary Translator

[06] 2009. Wei Chen. A Hardware Approach for Reducing Interpretaion Overhead. (CIT'09)

[07] 2009. Wei Chen. A Light-weight Code Cache Design for Dynamic Binary Translation. (ICPADS'09)

[08] 2016. Wenwen Wang. A General Persistent Code Caching Framework for Dynamic Binary Translation

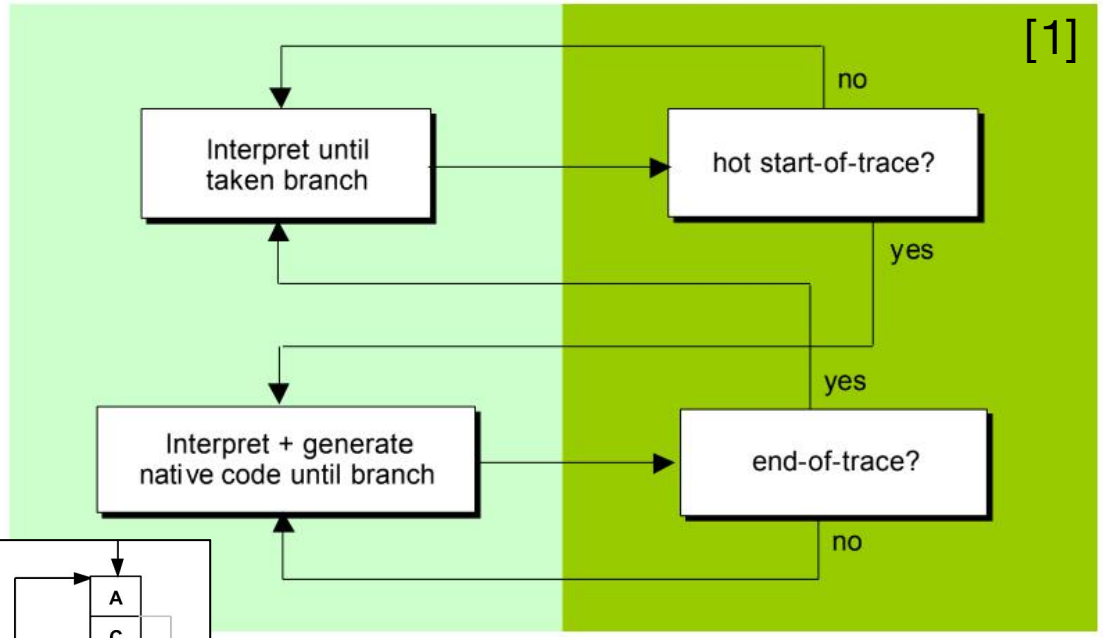
[09] 2017. Emilio G.Cota. Cross-ISA Machine Emulation for Multicores. CGO'17

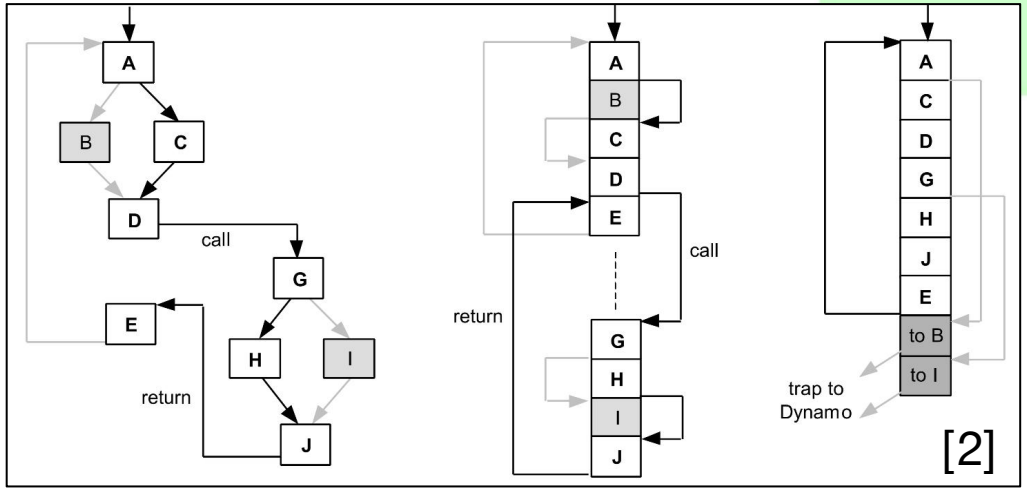
[10] 2018. Filipe Salgado. A hardware-assisted translation cache for DBT in embedded system

[12] 2018. Fei Deng. Research on CodeCache Management Strategy Based on Code Heat in DBT

Optimization - Better Binaries

- ... it is possible to use a piece of software to improve the performance of a native, statically optimized program binary, while it is executing. [1]
  - interpret and code generation
- MERT algorithm [2] (NET)
  - most recently executed tail
  - to pick hot traces





[1] 1999. Vasanth Bala. Transparent Dynamic Optimization: The Design and Implementation of Dynamo

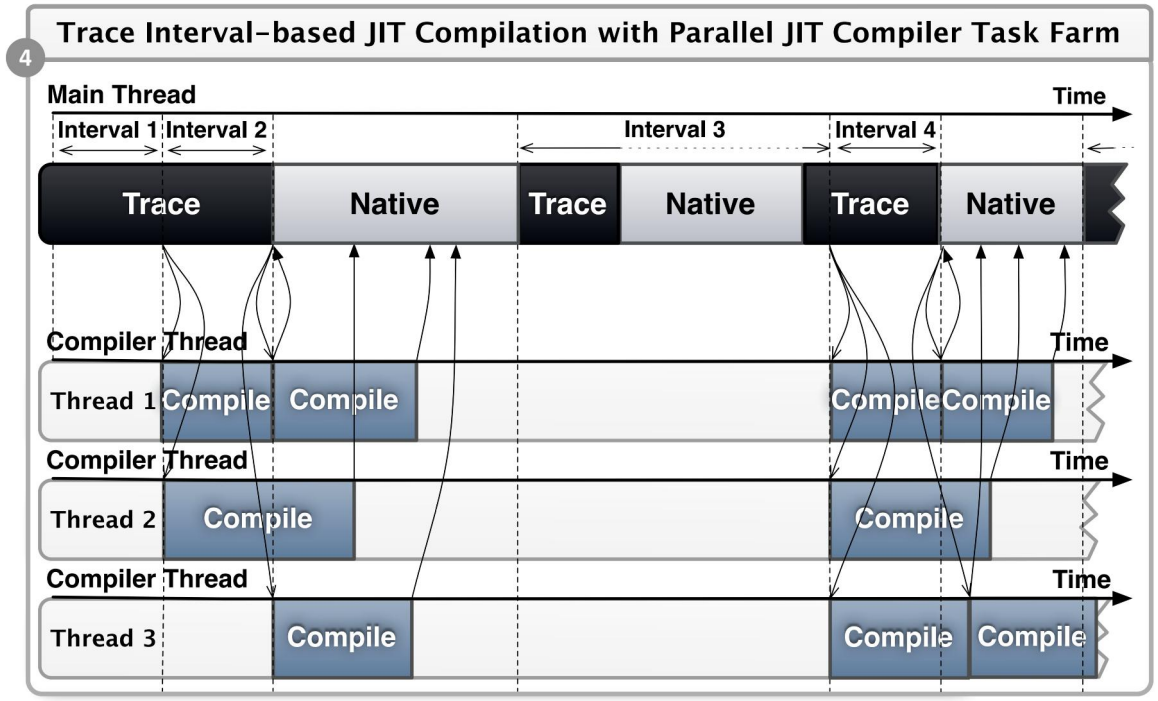
[2] 2000. Vasanth Bala. Dynamo: A Transparent Dynamic Optimization System

Optimization - Better Binaries

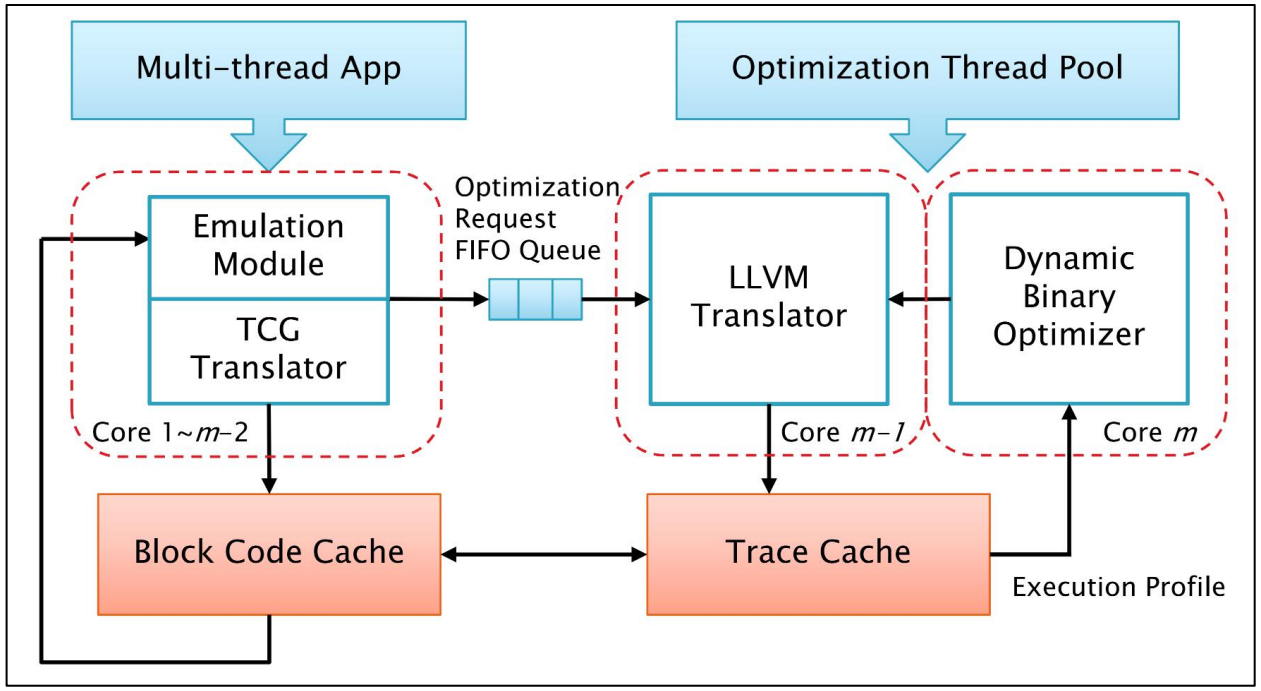
2011. Fast Dynamic Translation Using LLVM On Multi-Core Hosts
2012. An LLVM-based Hybrid Binary Translation System
2013. Efficient and Retargetable Dynamic Binary Translation
2014. Dynamically Translating Binary Code for Multi-Threaded Programs Using Shared Code Cache
2016. A Unidied Static Binary Analysis Framework
2019. Raising binaries to LLVM IR with MCTOLL
2020. Robust Practical Binary Optimization at Run-time using LLVM
2021. Efficient LLVM-based dynamic binary translation

Optimization - Better Binaries

- background optimization thread to off-load the overhead



Parallel JIT compilation task form [1]



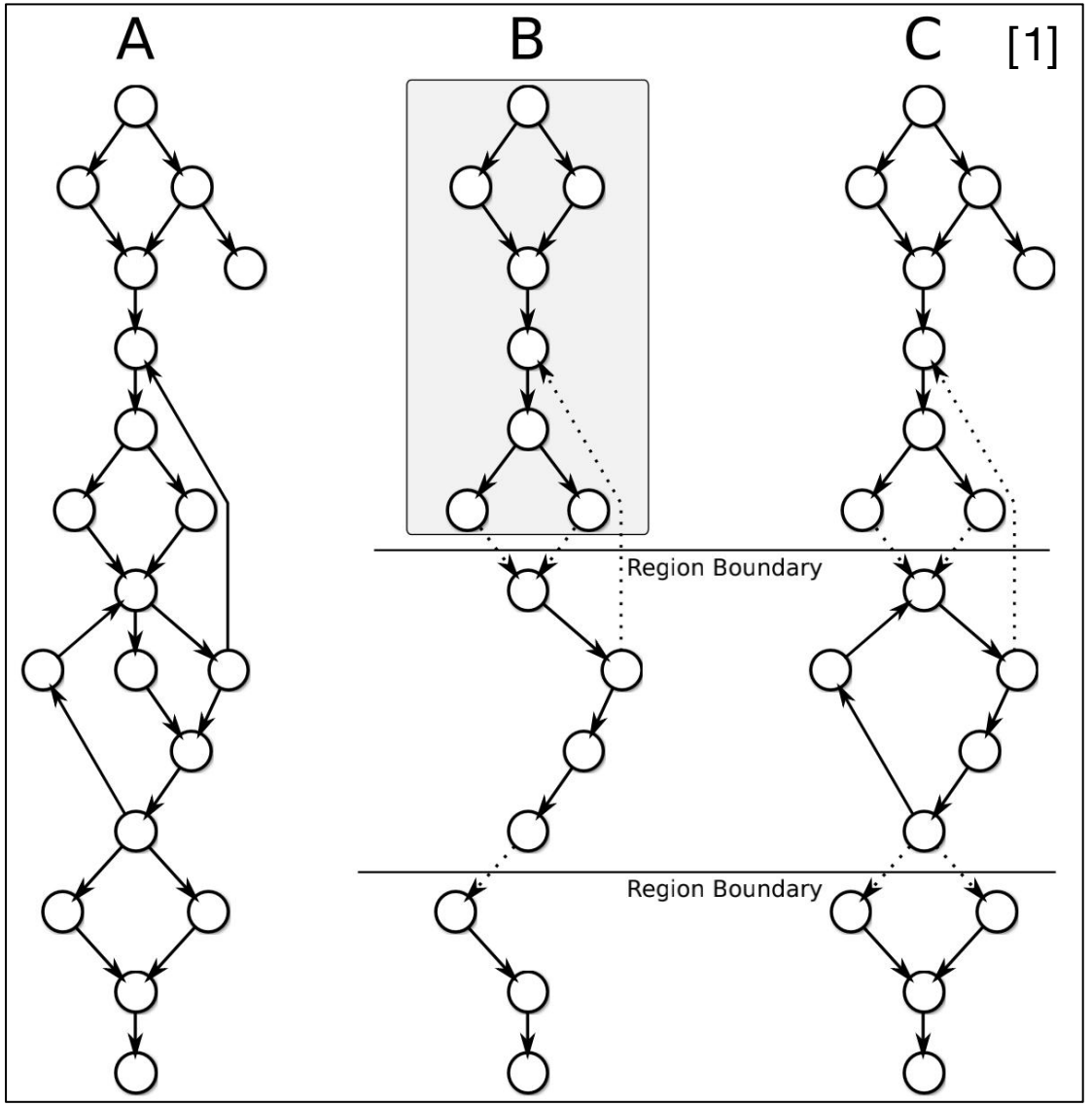
Major components of HQEMU [2]

[1] 2011. Igor Bohm. Generalized Just-In-Time Trace Compilation using a parallel task farm in a DBT

[2] 2014. Ding-Yong Hone. Efficient and Retargetable Dynamic Binary Translation on Multicores

Optimization - Better Binaries

- a complete code generation strategy for a region-based DBT, which exploits branch type and control flow profiling information to improve code quality for the common case. [1]
  - direct branch to same page
  - direct branch to different page
  - indirect branch
  - LLVM optimization passes

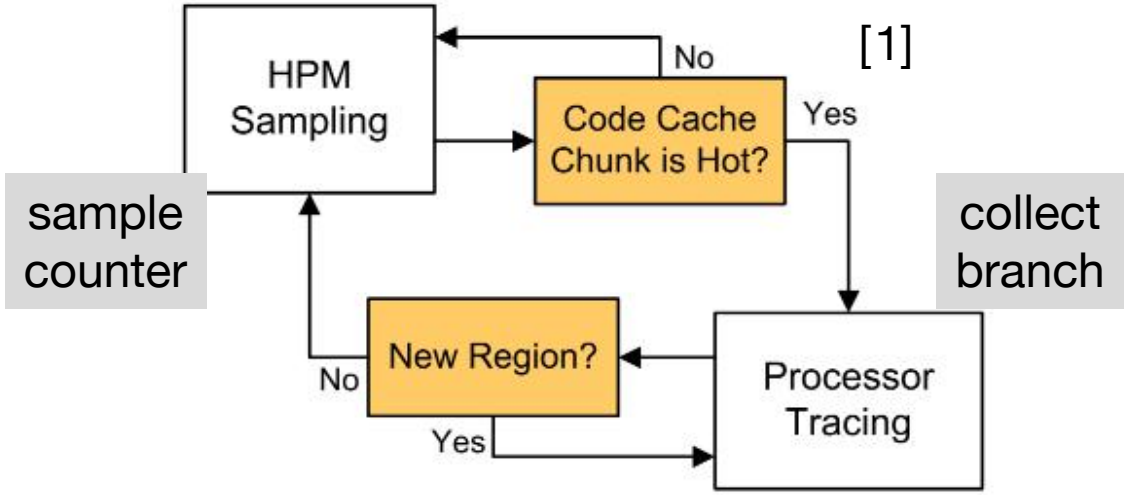


[1] 2014. Tom Spink. Efficient code generation in a region-based dynamic binary translator

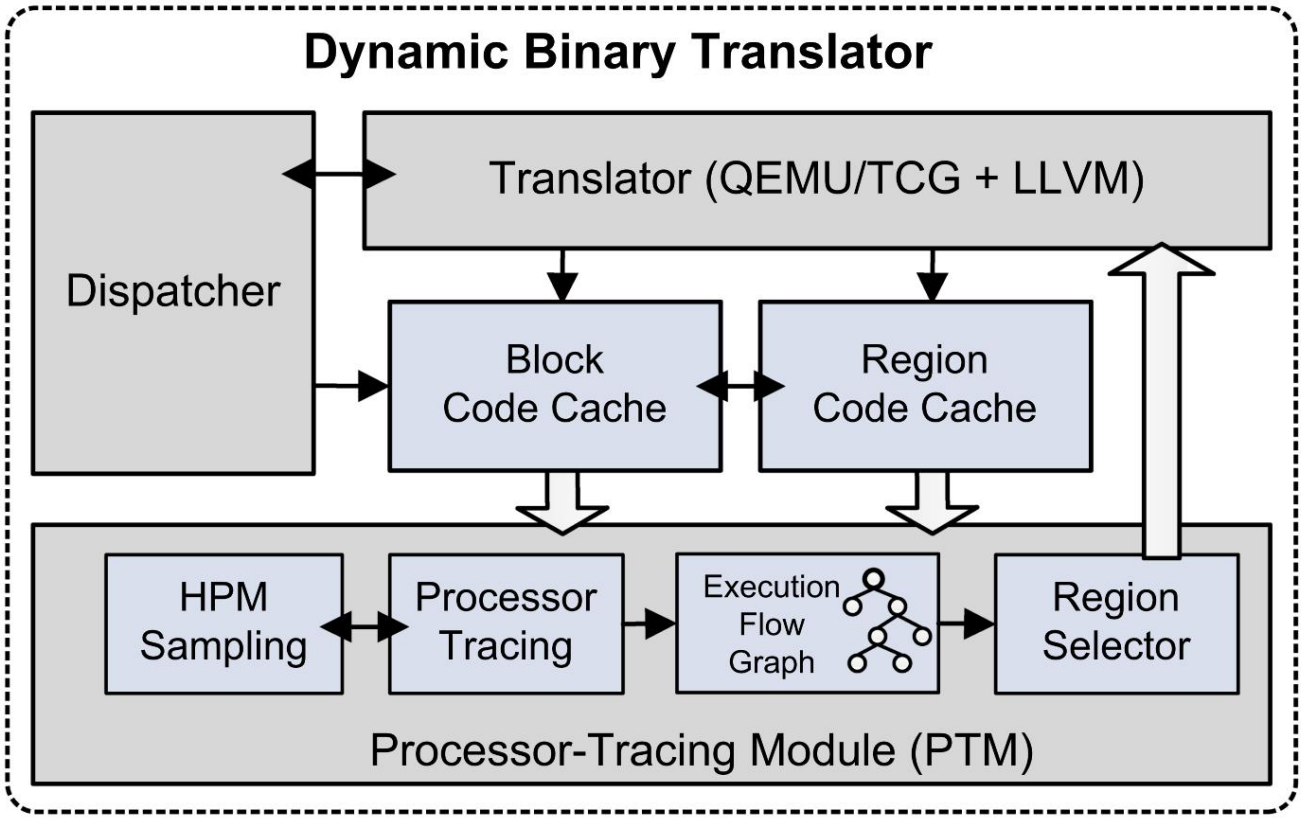
Optimization - Better Binaries

- ... leverage the branch history information stored in the processor to reconstruct the program execution profile and effectively form high-quality regions with low cost. [1]
  - Intel LBT / BTS / PT
  - ARM CoreSight

[1]



[1]

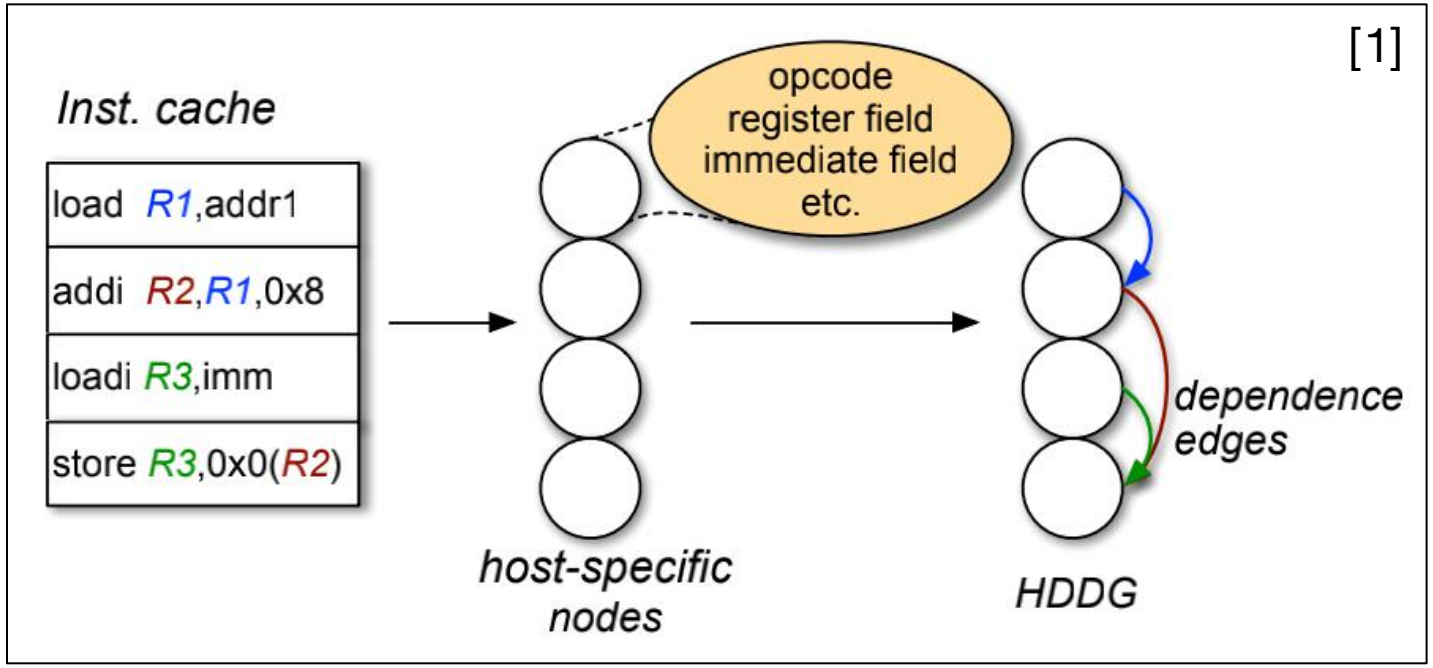


[1] 2018. Ding-Yong Hong .Processor-Tracing Guided Region Formation in Dynamic Binary Translation

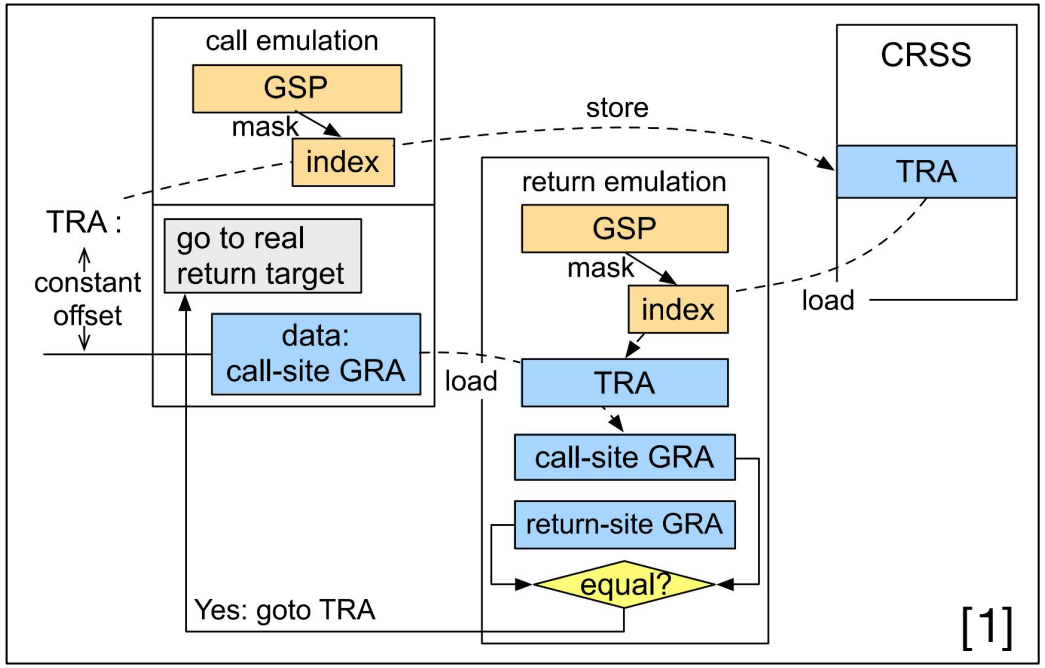
Optimization - Better Binaries

- ... conducting post-optimization on the translated code, rather than on the IR as conventional binary translators do. [1]

Host-specific Data Dependence Graph



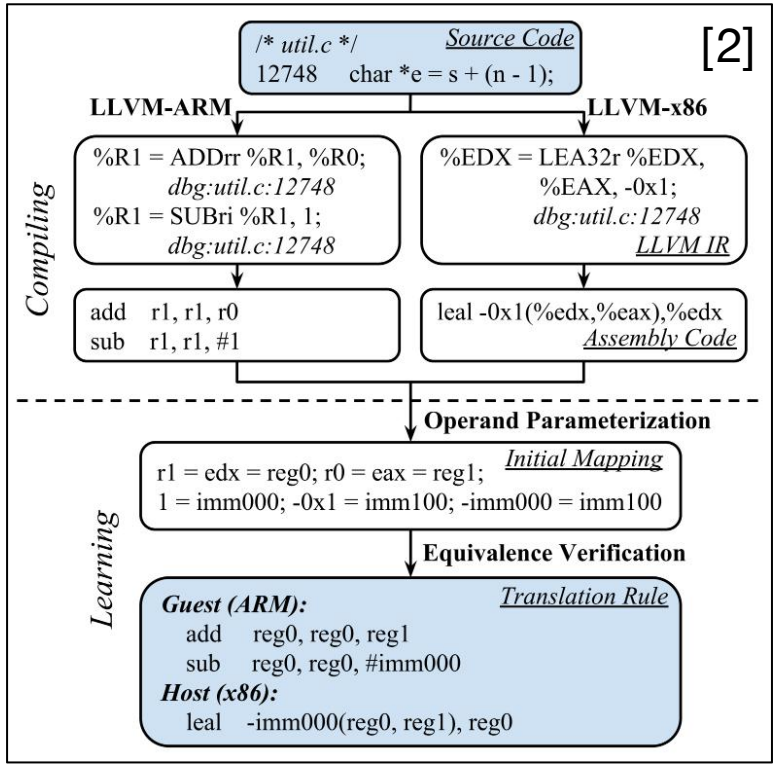
some kind of shadow stack



[1] 2015. Xiaochun Zhang. HERMES: A Fast Cross-ISA Binary Trnaslator with Post-Optimization

Optimization - Better Binaries

- Peephole rules are patern matching rules ... [1]
  - emumerates all possible instruction sequences up to a certain length. [1]
- ... automatically learn translation rules from guest and host binaries compiled from the same source code. [2]
  - Compiling
    - LLVM-ARM
    - LLVM-x86
  - Learning
    - Operand Parameterization
    - Equivalence Verification



Guest (ARM): [2]

add r0, r1, r0, lsl #2

ldr r0, [r0, #-4]

Host (x86):

movl -0x4(%ecx,%eax,4),%eax

[1] 2008. Sorav Bansal. Binary Translation Using Peephole Superoptimizers. (OSDI)

[2] 2018. Wenwen Wang. Enhancing Cross-ISA DBT Through Atumatically Learned Translation Rules. (ASPLOS)

Optimization - Better Binaries

[01] 2008. Sorav Bansal. Binary Translation Using Peephole Superoptimizers. (OSDI)

[02] 2011. Igor Bohm. Generalized Just-In-Time Trace Compilation using a parallel task farm in a DBT

[03] 2014. Tom Spink. Efficient code generation in a region-based dynamic binary translator

[04] 2014. Ding-Yong Hone. Efficient and Retargetable Dynamic Binary Translation on Multicores

[05] 2015. Xiaochun Zhang. HERMES: A Fast Cross-ISA Binary Trnaslator with Post-Optimization

[06] 2018. Wenwen Wang. Enhancing Cross-ISA DBT Through Atumatically Learned Translation Rules. (ASPLOS)

[07] 2018. Ding-Yong Hong .Processor-Tracing Guided Region Formation in Dynamic Binary Translation

[08] 2019. Unleashing the Power of Learning- An Enhanced Learning-Based Approach for Dynamic Binary Translation

[09] 2020. More with Less – Deriving More Translation Rules with Less Training Data for DBTs Using Parameterization

Optimization - Float Point

- ... most FP operations can be correctly emulated by surrounding the use of the host FP unit with a minimal amount of non-FP code. [1]

- FP workloads operate mostly on normal or zero numbers
  - denormals, infinities, NaNs not necessary
- FP flags are rarely cleared
- most FP operations raise the inexact flag
- FP workloads rarely change the rounding mode, which default to round-to-nearest-even

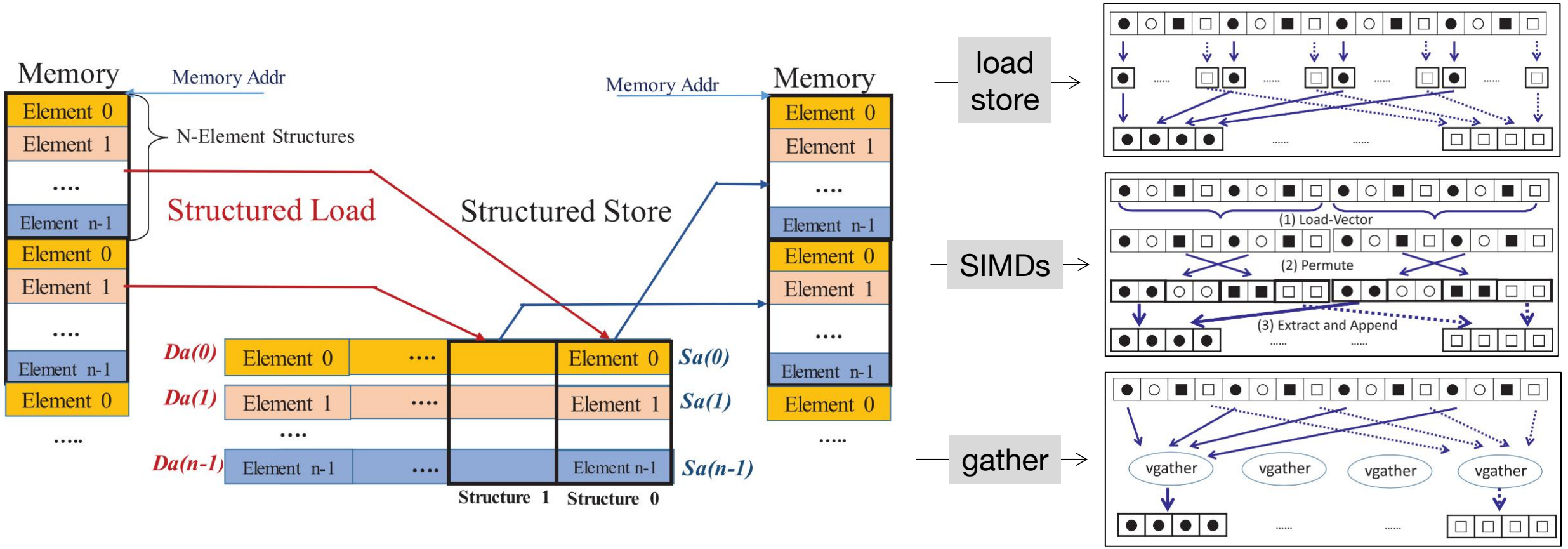
[1]

```
0   float64 float64_mul(float64 a, float64 b, fp_status *st)
1   {
2     float64_input_flush2(&a, &b, st);
3     if (likely(float64_is_zero_or_normal(a) &&
4                float64_is_zero_or_normal(b) &&
5                st->exception_flags & FP_INEXACT &&
6                st->round_mode == FP_ROUND_NEAREST_EVEN)) {
7       if (float64_is_zero(a) || float64_is_zero(b)) {
8         bool neg = float64_is_neg(a) ^ float64_is_neg(b);
9         return float64_set_sign(float64_zero, neg);
10      } else {
11        double ha = float64_to_double(a);
12        double hb = float64_to_double(b);
13        double hr = ha * hb;
14        if (unlikely(isinf(hr))) {
15          st->float_exception_flags |= float_flag_overflow;
16        } else if (unlikely(fabs(hr) <= DBL_MIN)) {
17          goto soft_fp;
18        }
19        return double_to_float64(hr);
20      }
21    }
22  soft_fp:
23    return soft_float64_mul(a, b, st);
24  }
```

[1] 2019. Emilio G.Cota. Cross-ISA Machine Instrumentation using Fast and Scalable DBT. (VEE)

Optimization - SIMD

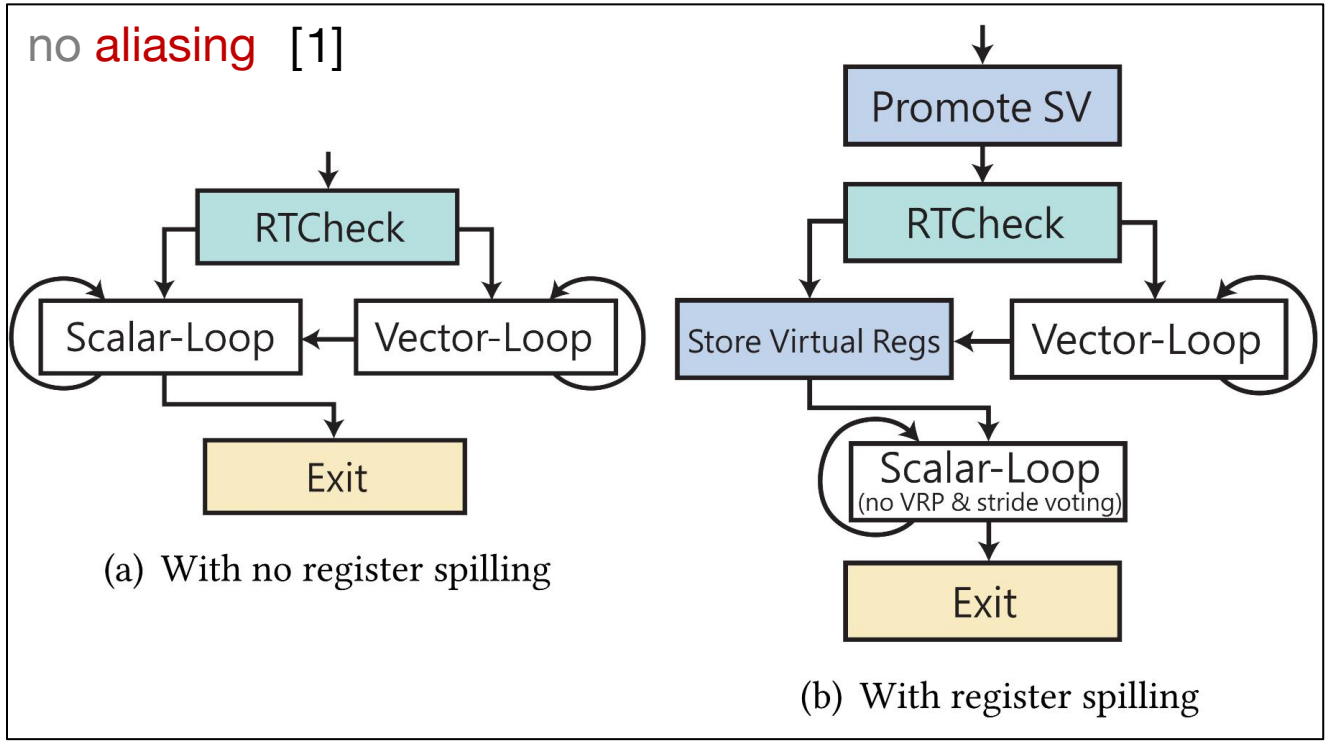
- Structured loads/stores, such as VLDn/VSTn in ARM NEON
  - widely used in signal processing, multimedia, mathematical, and 2D matrix transposition applications [1]

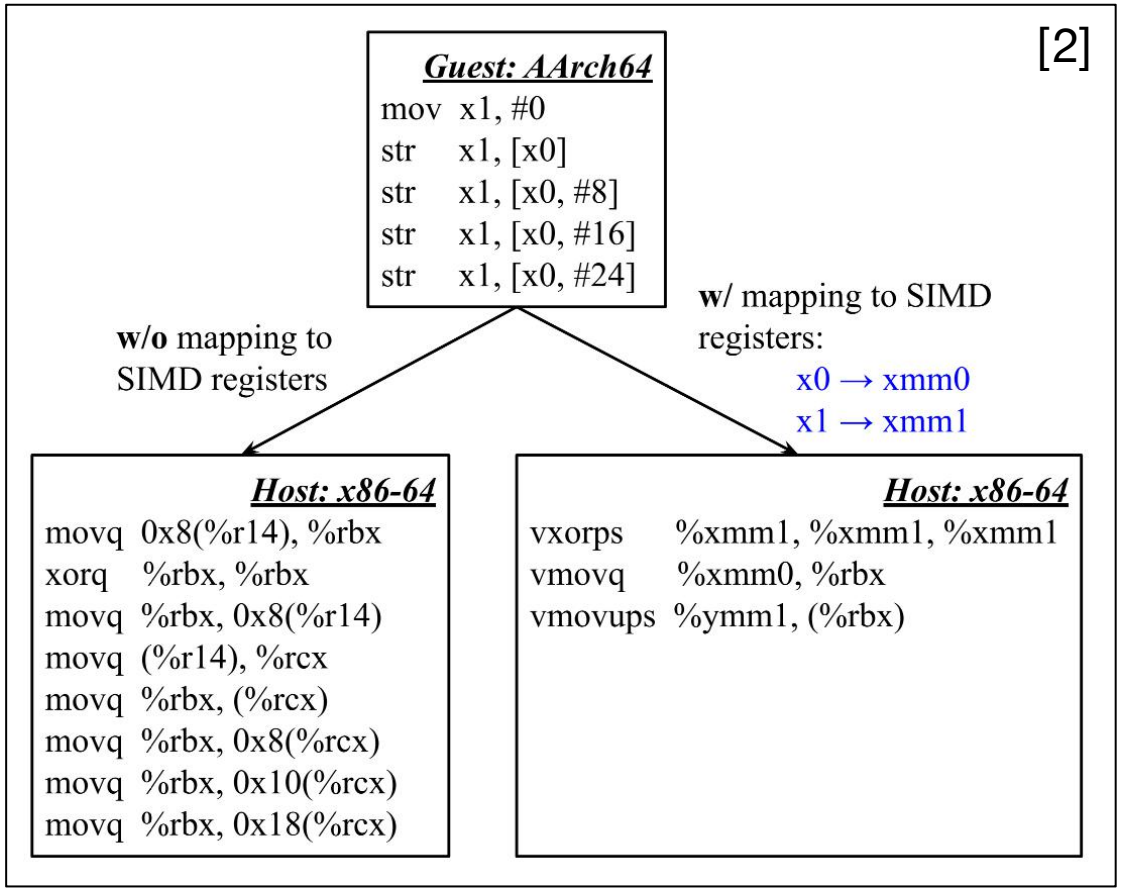


[1] 2019 Sheng-Yu Fu. Optimizing data permutations in structured loads-stores translation and SIMD register mapping for a cross-ISA DBT

Optimization - SIMD

- ... convert non-vectorized loops to vector/SIMD forms ... [1]



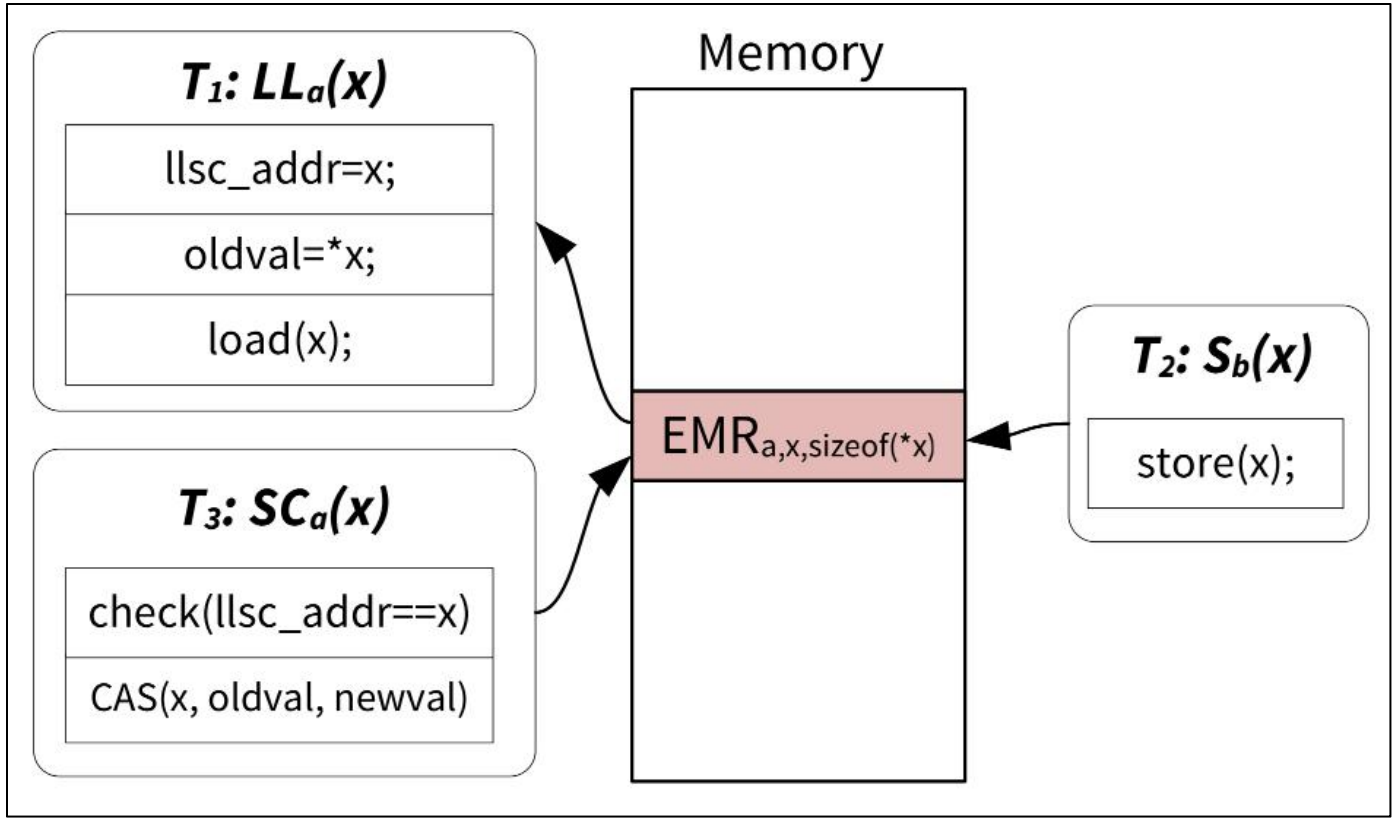


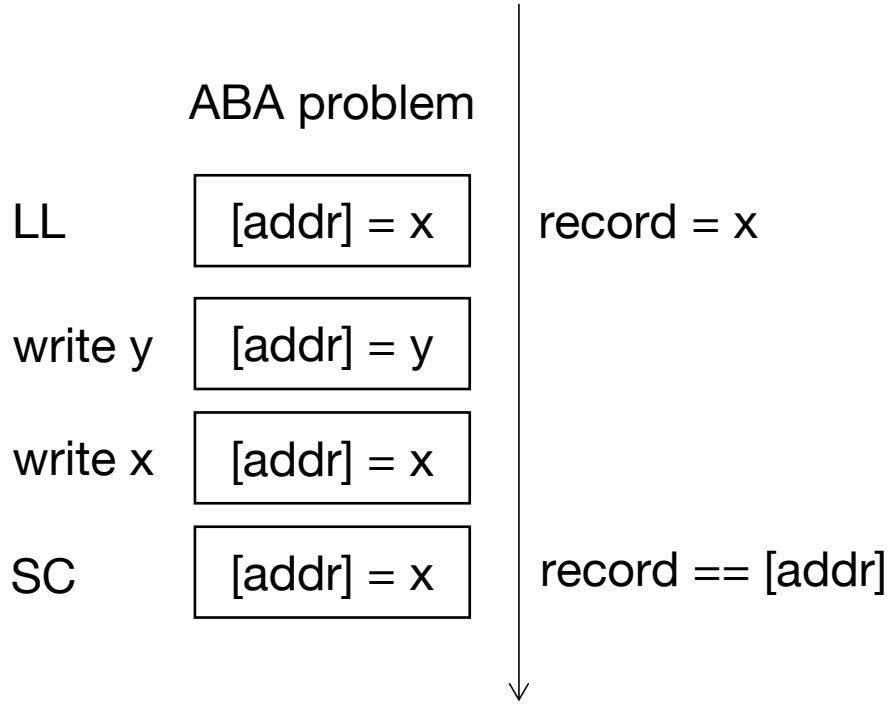
[1] 2019. Chih-Min Lin. Exploiting Vector Processing in Dynamic Binary Translation. (ICPP)

[2] 2021. Jin Wu. Effective Exploitation of SIMD resources in Cross-ISA Virtualization. (VEE)

Optimization - Atomic

- Compare-and-Swao (CAS) & Load-Link / Store-Conditional (LL/SC)
  - easy to emulate CAS with LL/SC
  - more challenging to emulate LL/SC with CAS [1]

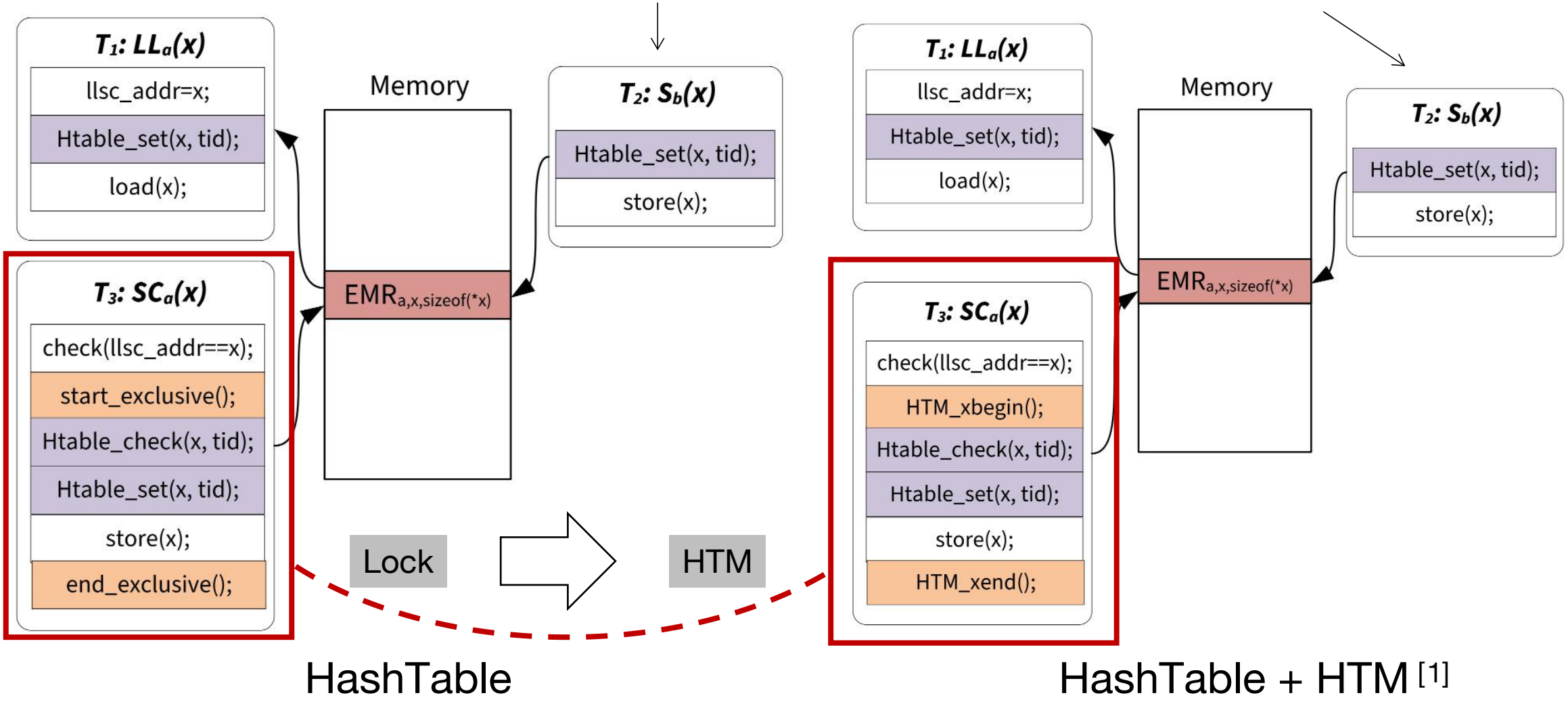




[1] 2021.Ziyi Zhao. Enhancing Atomic Instruction Emulation for Cross-ISA Dynamic Binary Translation

Optimization - Atomic

every normal store need to update hash table: address X is modified by thread TID



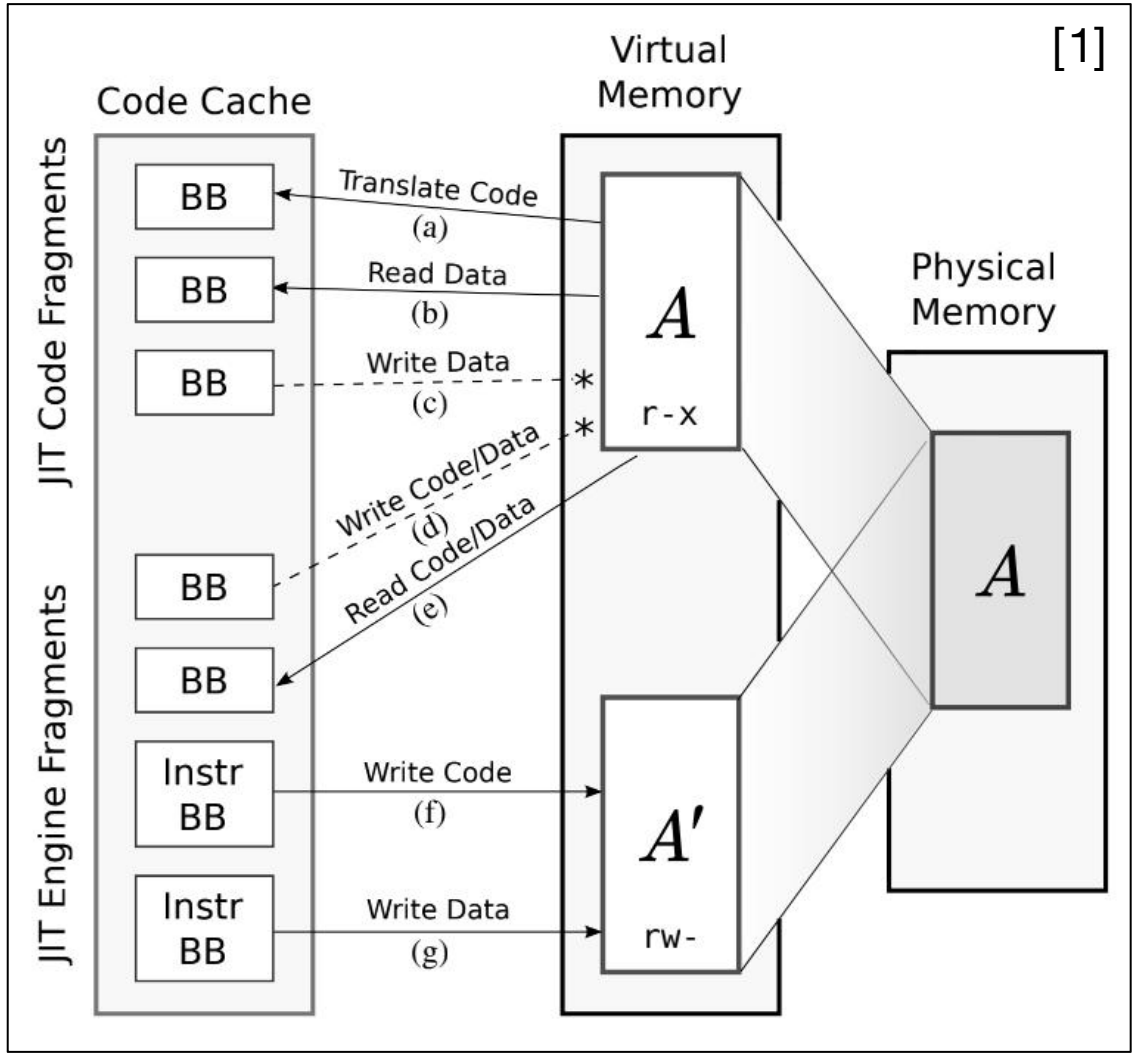
HashTable

HashTable + HTM [1]

[1] 2021.Ziyi Zhao. Enhancing Atomic Instruction Emulation for Cross-ISA Dynamic Binary Translation

Optimization - Guest JIT

- ... efficient source code annotations that allow developers to demarcate dynamic code regions and identify code changes within those regions. [1]
  - code cache
  - flush
- .. automatically inferring the presence of a JIT and instrumenting its write instructions with translation consistency operations [1]



[1] 2015. Byron Hawkins. Optimizating Binary Translation of Dynamic Generated Code. (CGO)

Optimization - Specific Guest

[1] 2019. Emilio G.Cota. Cross-ISA Machine Instrumentation using Fast and Scalable DBT. (VEE)

[2] 2019. Sheng-Yu Fu. Optimizing data permutations in structured loads-stores translation and SIMD register mapping for a cross-ISA DBT

[3] 2019. Chih-Min Lin. Exploiting Vector Processing in Dynamic Binary Translation. (ICPP)

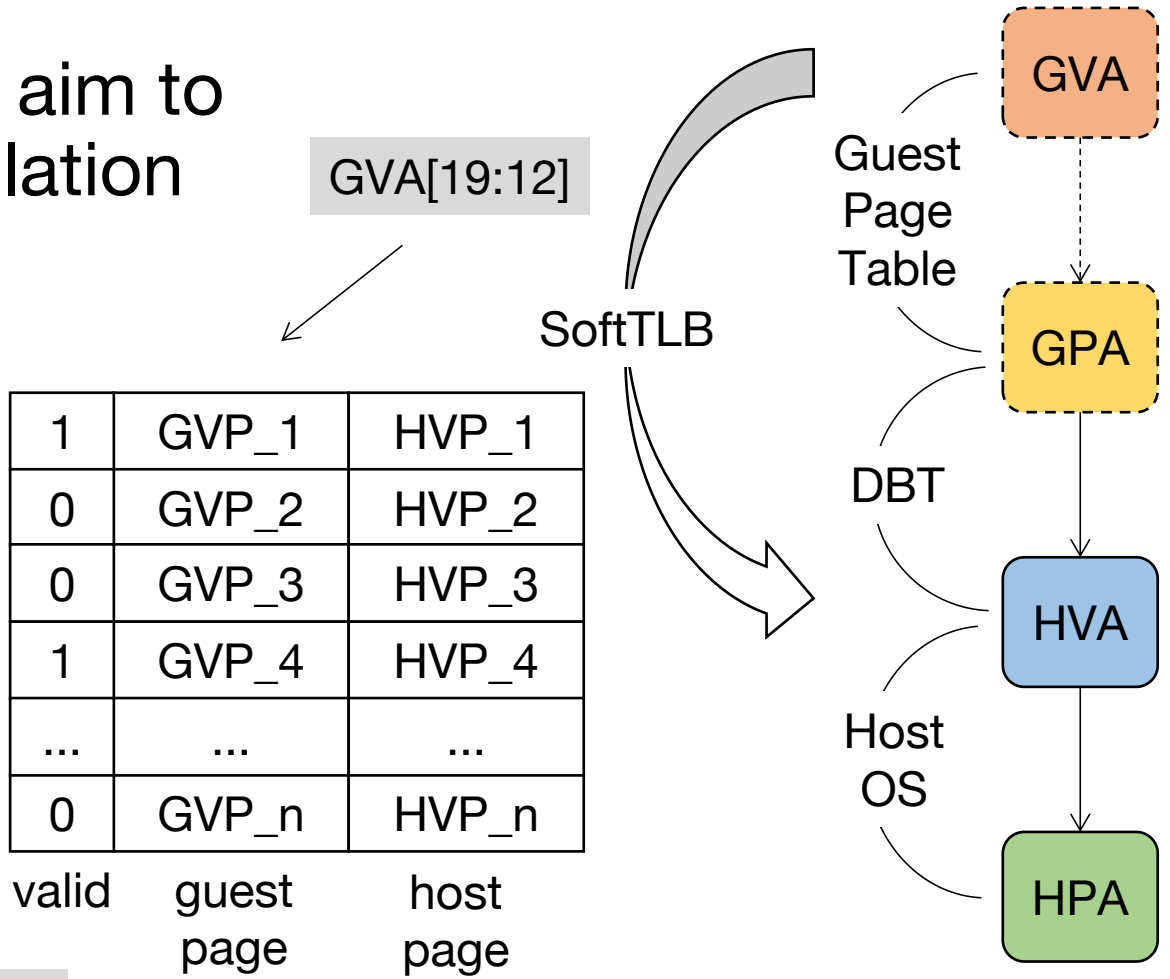
[4] 2021. Jin Wu. Effective Exploitation of SIMD resources in Cross-ISA Virtualization. (VEE)

[5] 2021.Ziyi Zhao. Enhancing Atomic Instruction Emulation for Cross-ISA Dynamic Binary Translation

[6] 2015. Byron Hawkins. Optimizating Binary Translation of Dynamic Generated Code. (CGO)

Optimization - system mode

- ... a series of sTLB optimizations that aim to reducing the address translation emulation overhead. [1]
  - sTLB lookup 13.2%
  - sTLB refills 24.9%
  - code cache 38.3%
  - othres 28.1%
- sTLB Optimization
  - size (number of entries, dynamically resize)
  - victim items
  - set-associative sTLB (SIMD walk)

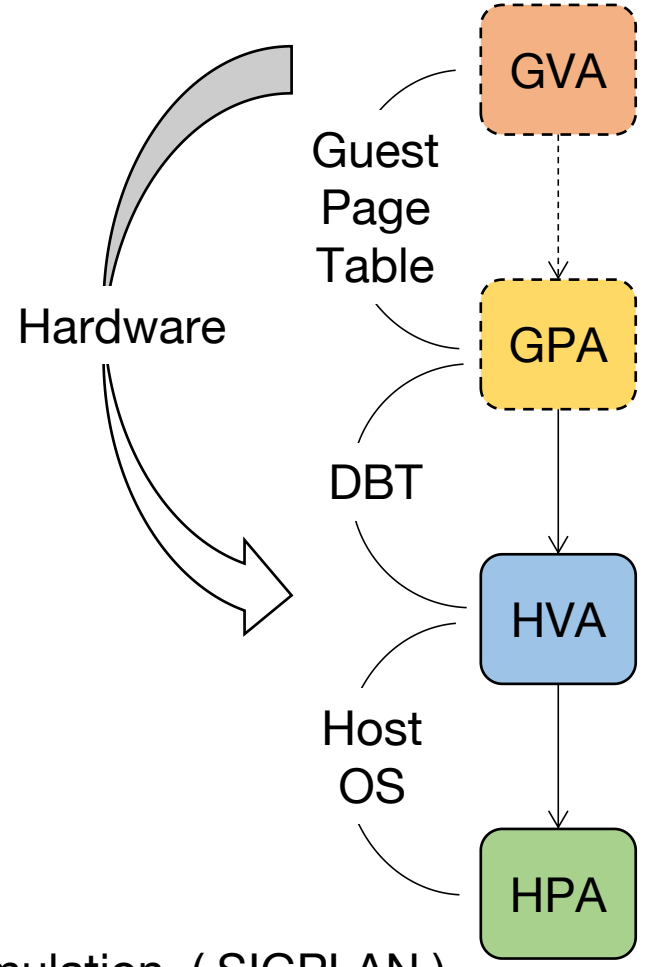


256 Entry STLB + 8 victim + Dynamically Resize

[1] 2015. Xin Tong. Optimizing Memory Translation Emulation in Full System Emulators. [IBM]

Optimization - system mode

- Embedded Shadow Page Tables (ESPT) [1]
  - put GVA -> HVA into host page table
- ESPT without kernel's modification
  - instead of relying on using LKMs, our approach adopts a shared memory mapping scheme [2]
- Using hardware assisted virtualisation for cross-ISA simulation [3]
  - dune framework



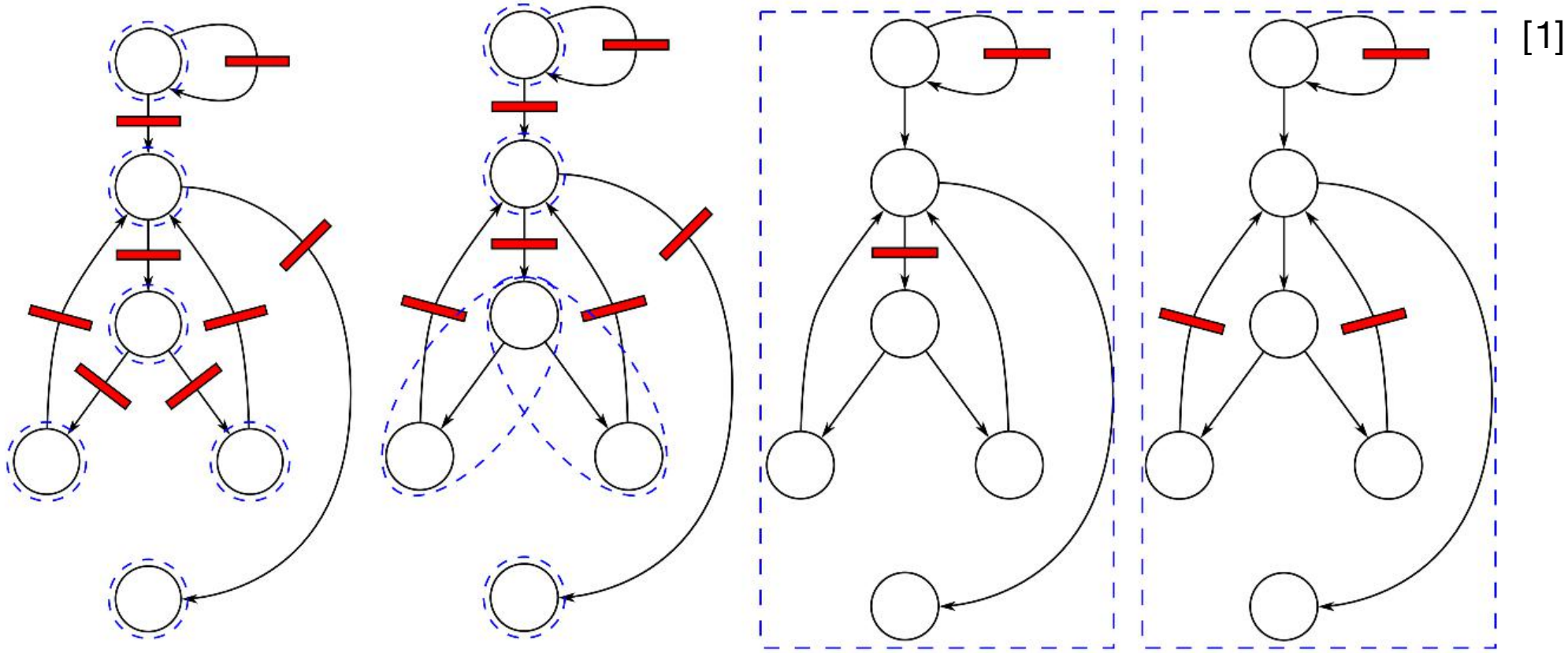
[1] 2014. Jan-Jan Wu. Efficient Memory Virtualization for Cross-ISA System Mode Emulation. (SIGPLAN)

[2] 2015. Zhe Wang. Practical Implementation and Efficient Management of Embedded Shadow Page Tables for Cross-ISA System Virtual Machines

[3] 2018. Antoine Faravelon. Acceleration of Memory Accesses in Dynamic Binary Translation.

Optimization - system mode

- ... a novel scheme for handling of asynchronous interrupts, ... [1]
  - What is the minimum number of interrupt checks that need to be inserted and where to insert them?

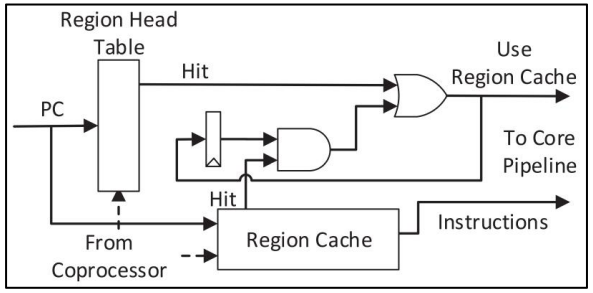


[1]

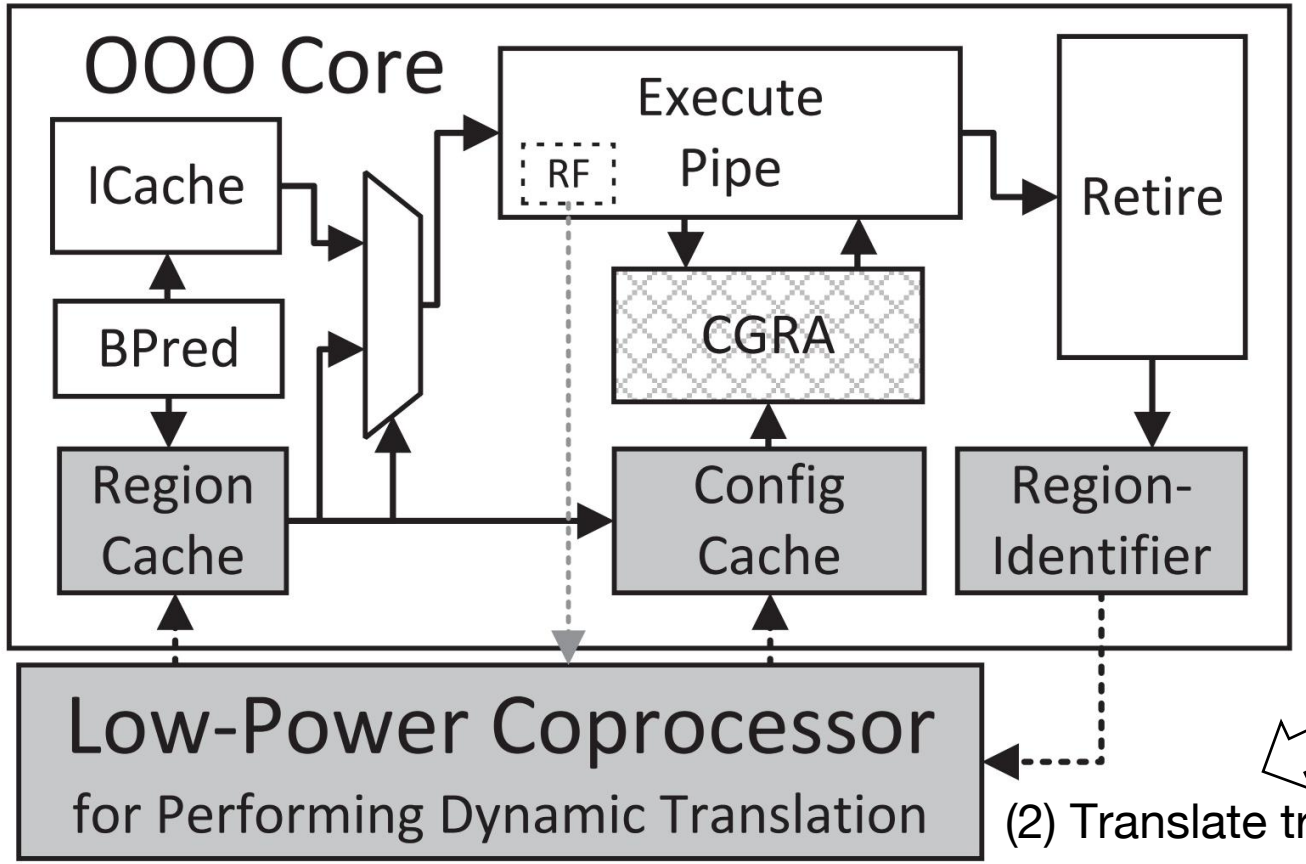
[1] 2016. Tom Spink. Efficient asynchronous interrupt handling in a full-system instruction set simulator. (LCTES)

Optimization - CGRA

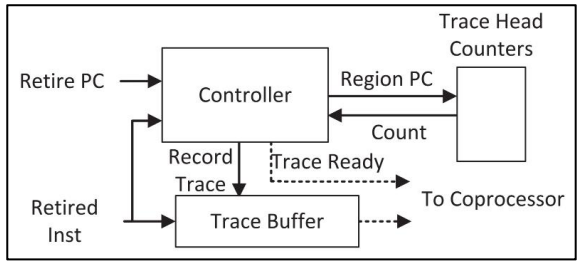
Course-Grained Reconfigurable Architectures



(3) Region Cache



[1]



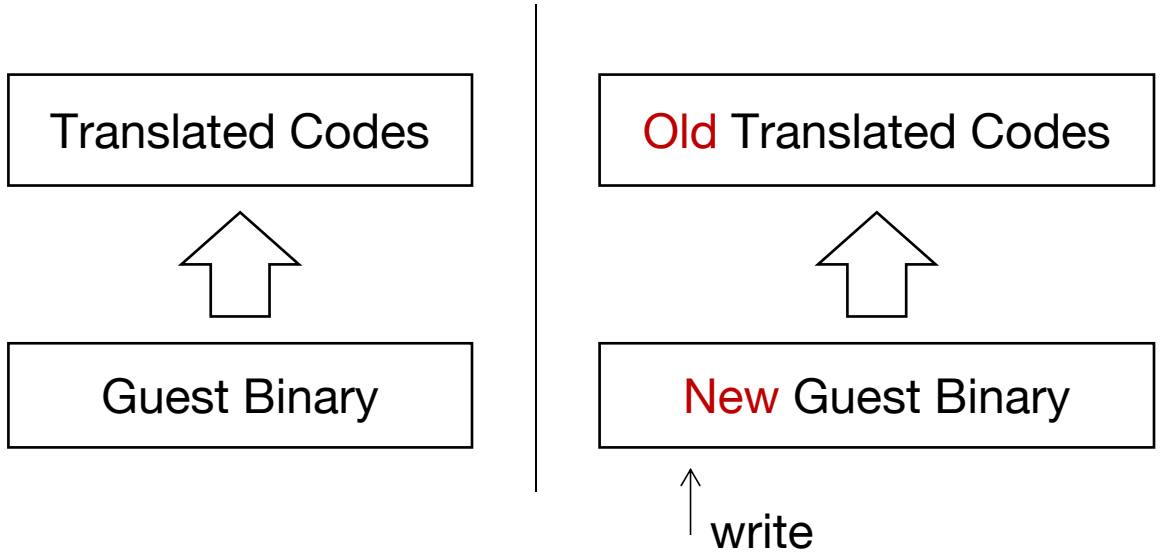
(1) Identify hot trace

(2) Translate trace into CCA config

[1] 2016. Matthew A. Watkins. Software Transparent Dynamic Binary Translation for Coarse-Grain Reconfigurable Architectures. (HPCA)

Advanced Topics - Problems

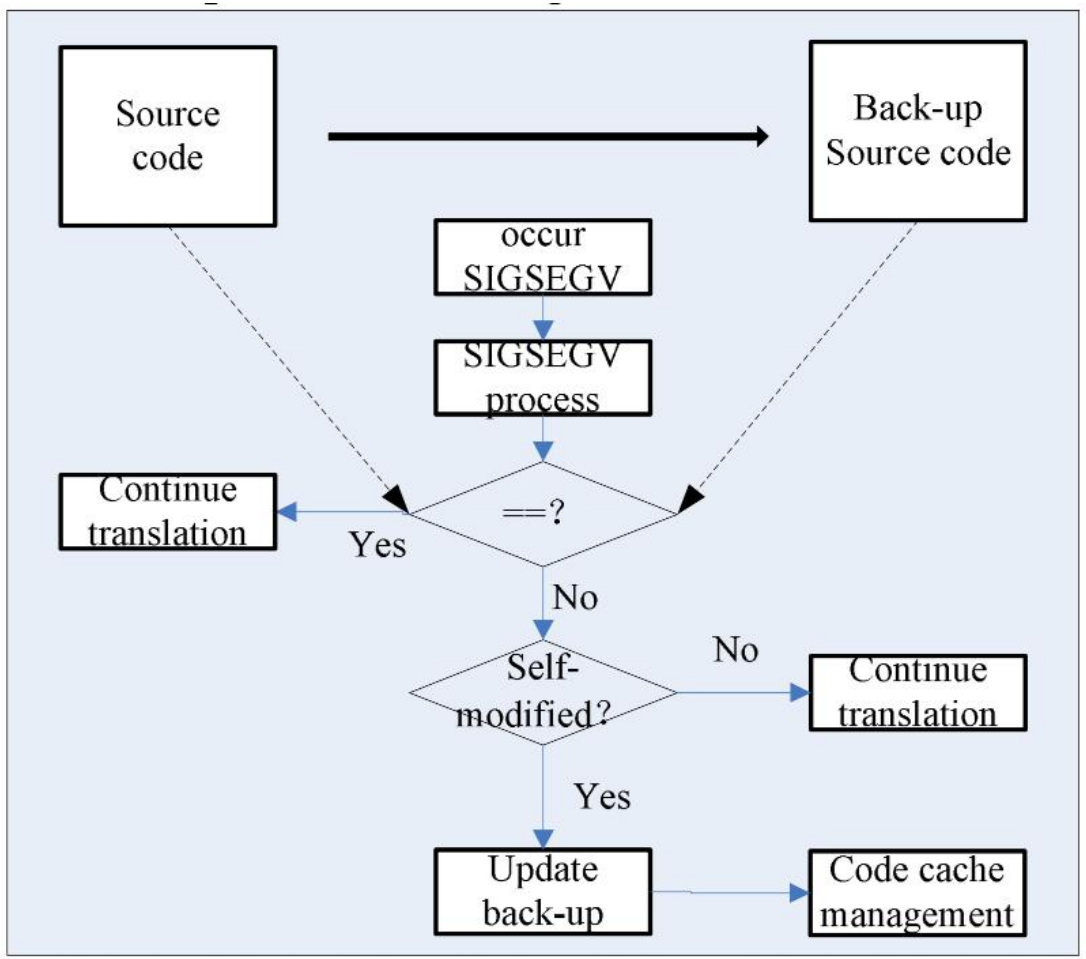
- Biggest Problem: Performance !
- Self-Modified Code: Code Cache
  - page protection in user-mode
  - softmmu in sys-mode
- Precise Exception: one -> many
  - TB boundary
  - signal in user-mode
  - interrupt / exception in sys-mode
- ABI Problem: Cross-ISA
- Memory Consistency: Cross-ISA



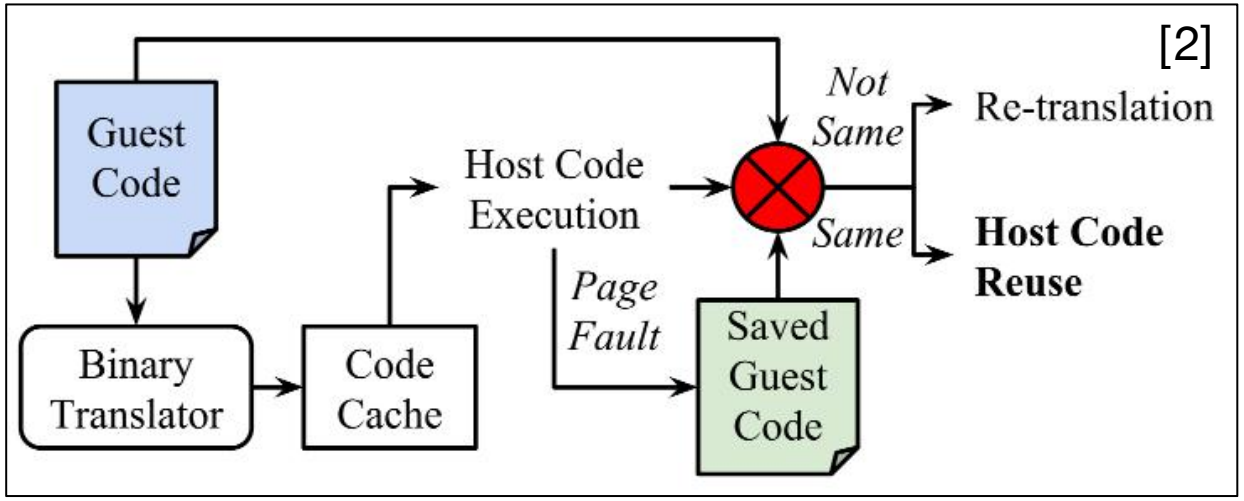
Problems - SMC

- ... ASCMS for self-modifying code cache management. The ASCMS provides a precise positioning to a translated block, not a trace or the whole code cache. [1]
  - through comparing source code and back-up source code, the system can find whether the code is modified by itself

[1]



[2]

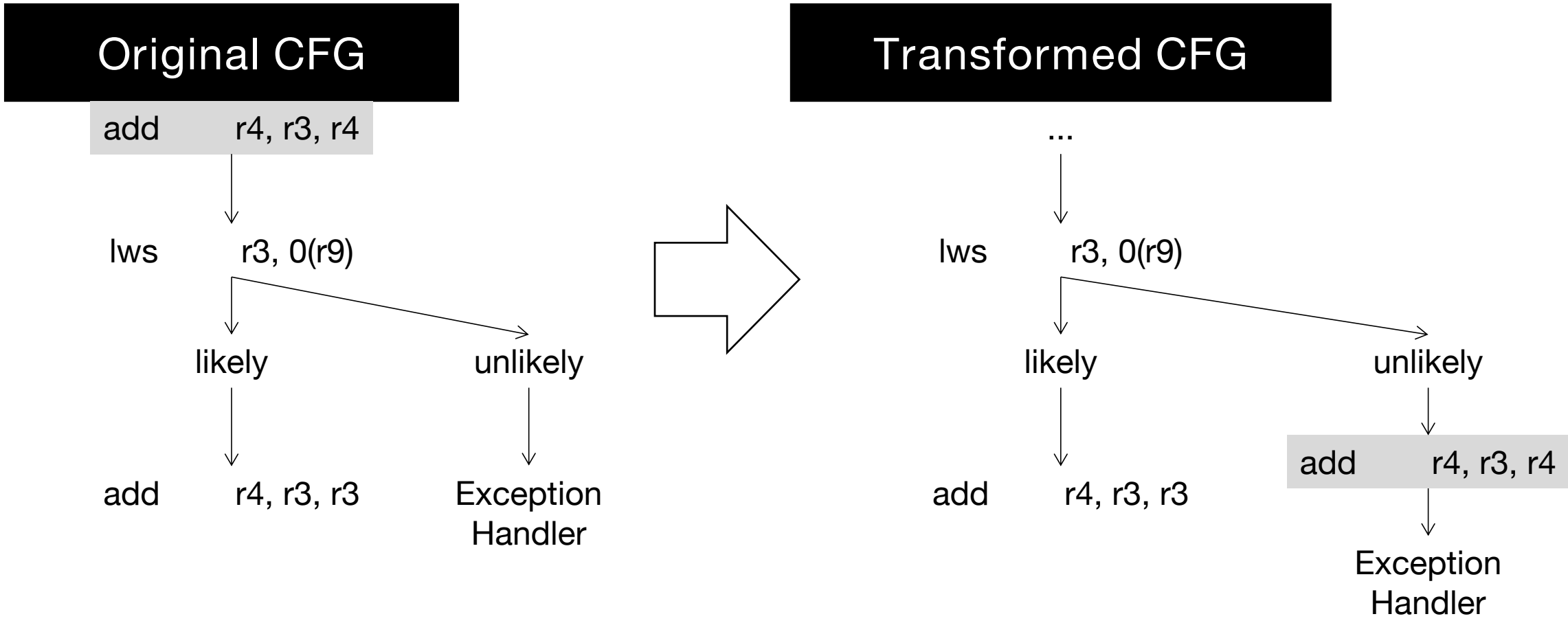


[1] 2013. Anzhan Liu. ASCMS - An Accurate Self-Modifying Code Cache Management Strategy in Binary Translation

[2] 2018. Wenwen Wang. Improving Dynamically-Generated Code Performance on DBTs. (VEE)

Problems - Precise Exception

- ... maintaining enough state to recompute the processor state when an unpredicted event such as a synchronous exception ... [1]



[1] 2001. Michael Gschwind .Optimization and precise exceptions in dynamic compilation

Advanced Topics

1. optimize BT to run faster
2. optimize Translation to generate better codes to run faster

- Optimizations
  - Branch, Code Cache and TB Lookup
  - Optimize Generated Codes
  - Special Guest: FP, SIMD, Atomic, EFLAGS, JIT, Dynamic Library, Syscall, ...
  - System-mode: Memory, branch, Interrupt, vCPU, ...
  - Course-Grained Reconfigurable Architectures
- Problems
  - Performance !
  - SMC
  - Precise Exception
  - Memory Consistency
  - ABI

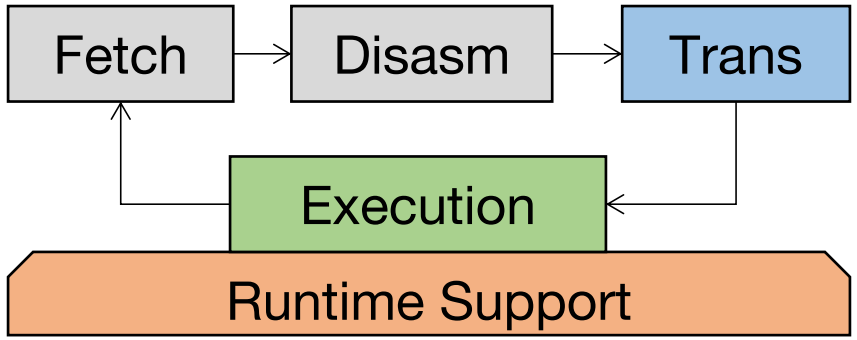
correctness

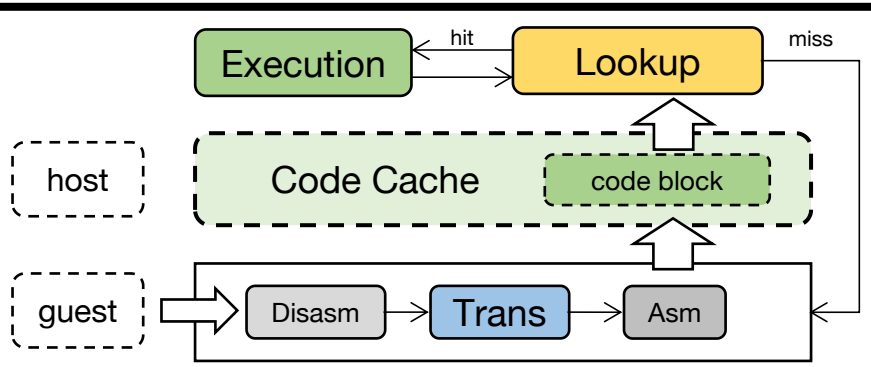
efficiency

sercutiry

A Survey on Binary Translation

Binary Rewrite	=	Translation	,	Optimization	,	Instrumentation
Script Language	=	JIT / Java / JVM	,	Interpret		Python, Matlab
Virtualization	=	KVM	, Wine	, Android	,	Gem5





- Optimization
  - branch
  - code cache
  - block lookup
  - better codes
  - float point
  - simd
  - atomic
  - .....
  - sys
    - address
    - interrupt
    - ...
- Common Problems
  - SMC
  - precise exception
  - cross-ISA: ABI, ...
  - ...

5

List of Binary Translation Tools Papers

6

Summary

A Survey on Binary Translation

- Introduction ..... basic architecture

- Related Technologies ..... differences and similarities
- Architecture Details ..... and basic optimizations
- Advanced Topics ..... problems, challenges, optimizations, etc.

- List of Binary Translation ..... by year, type, etc.
- Summary

List of Binary Translation

1996 Embra

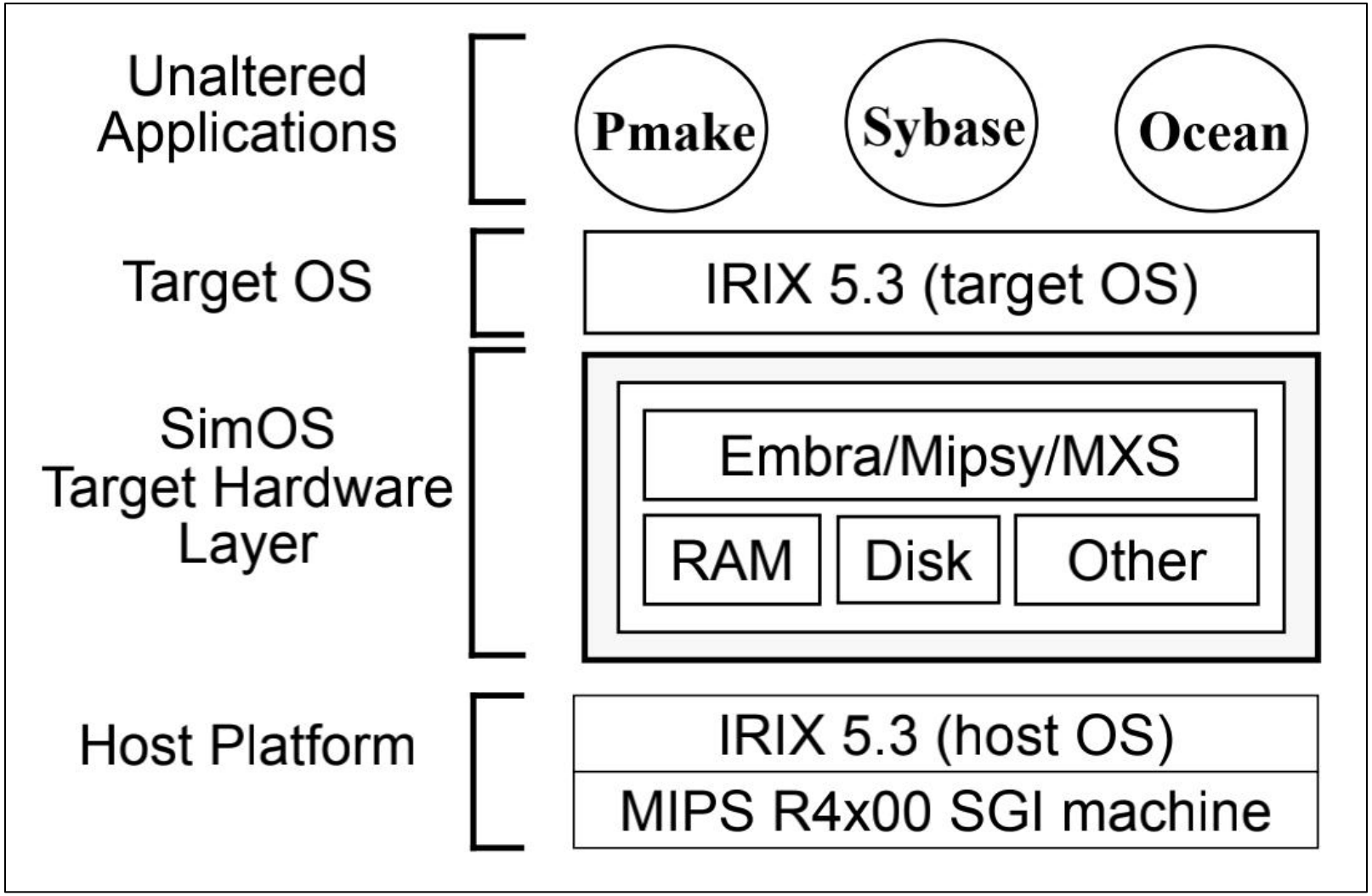
Full-system DBT

Guest MIPS R3000/R4000

Host MIPS

- MMU Address Translation
- Self-Modified Code
- Multiple Processors
- Physical Address Chaining

- Cache Simulation



1996. Embra Fast and Flexible Machine Simulation. (ACM SIGMETRICS)

List of Binary Translation

1997 FX!32

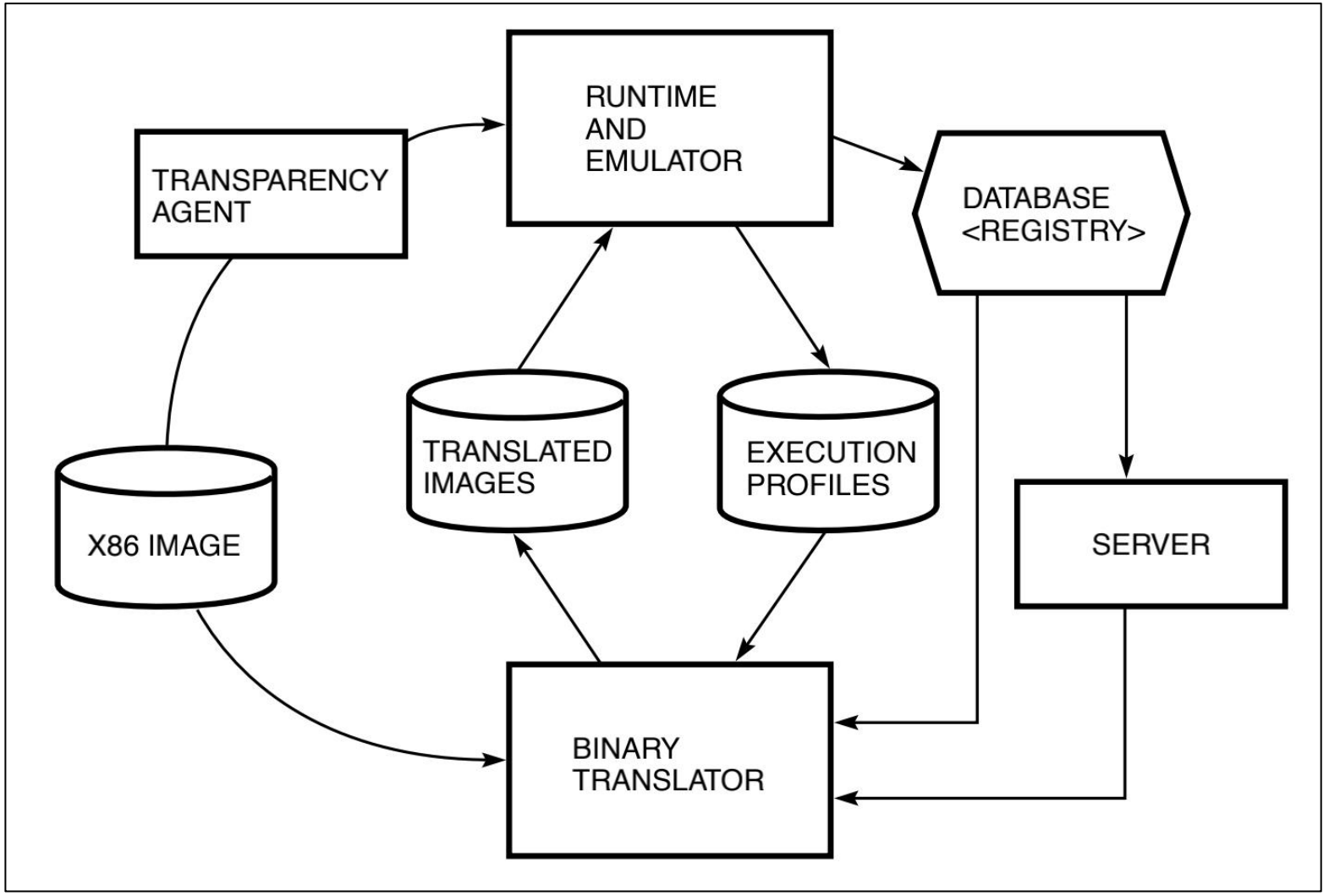
User-Mode DBT Windows NT 4.0

Guest 32-Bit x86

Host Alpha NT

500 MHz Alpha = 200 MHz Pentium-Pro

- Runtime loader
- Emulator Interpretor
- Translator Translator
- Database Profiles
- Server, Manager, ...



1997. DIGITAL FX!32 Combining Emulation and binary translation. (Digital Technical Journal)

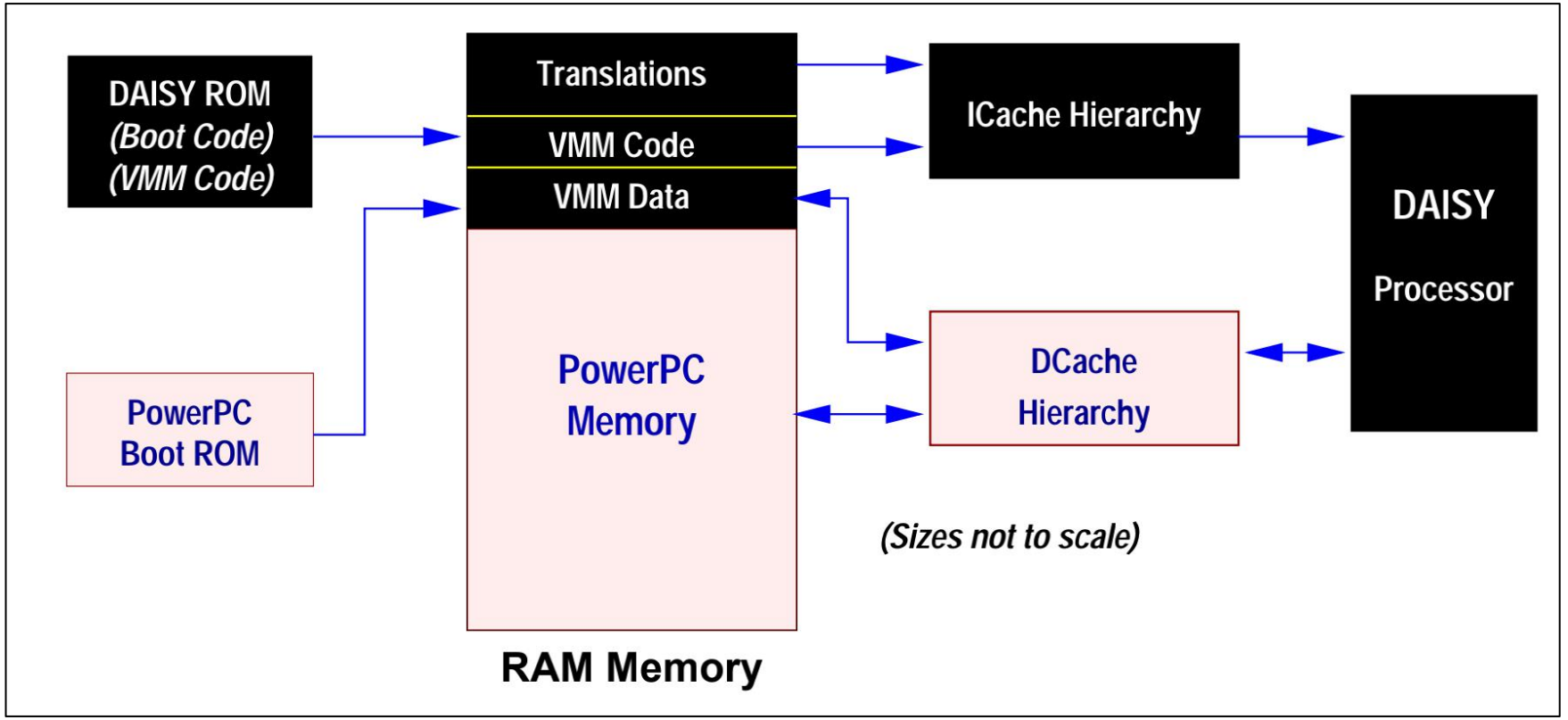
List of Binary Translation

2000 DAISY

Full-system DBT

Guest PowerPC, s390, ...

Host VLIW



- Self-Modified Code
- Precise Exception
- Memory Consistency
- Memory-Maped IO
- Aggressive Reordering
- ...

2000. Full System Binary Translation RISC to VLIW. (IBM Research)

2001. Dynamic Binary Translation and Optimization. (IEEE Computers)

List of Binary Translation

2003 Code Morphing

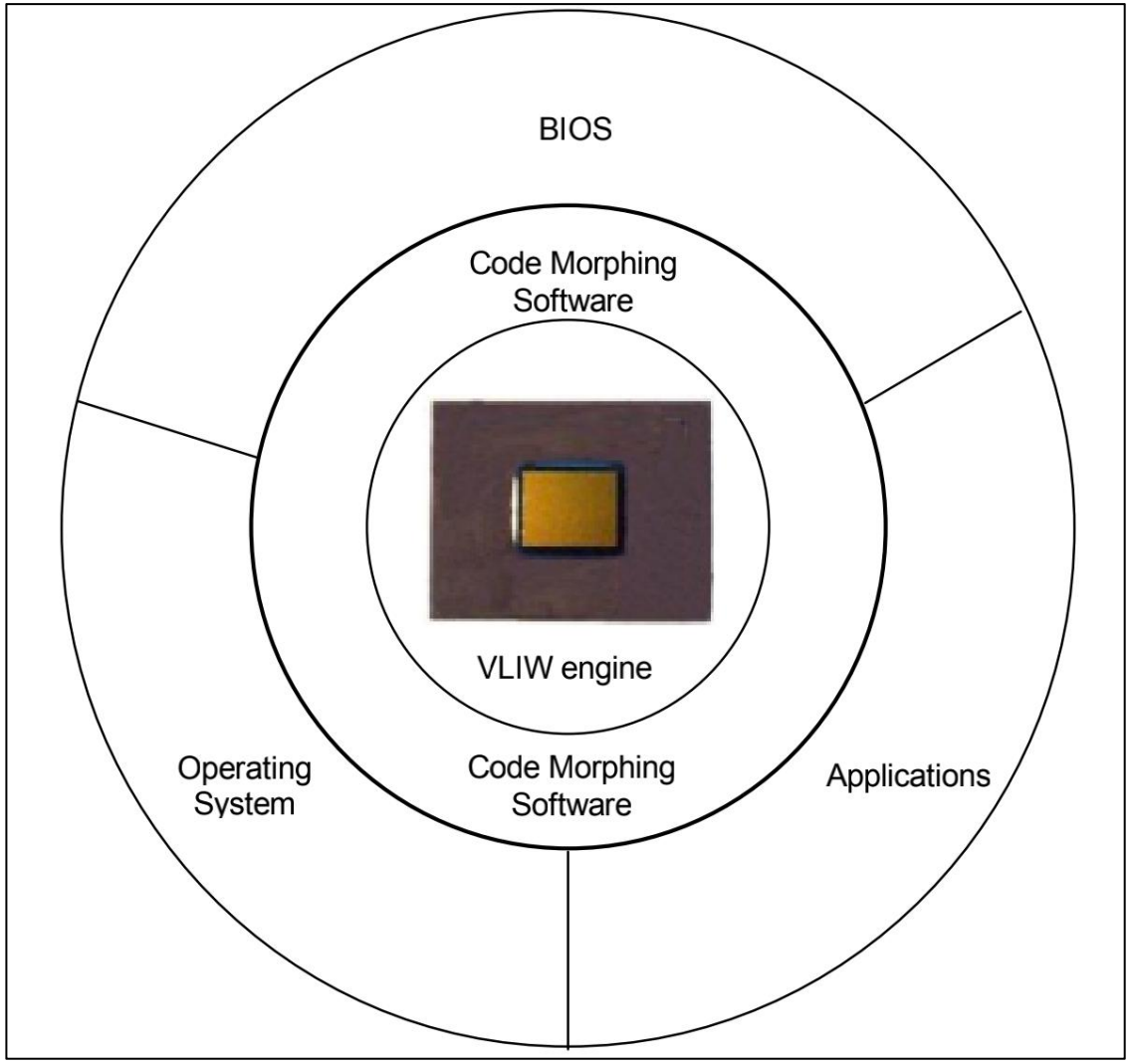
Full-system DBT

Guest x86

Host Curose Processor

Hardware Support

- Precise Exception
- Interrupts
- Memory-Mapped I/O
- Data Speculation
- Self-Modified Code



2003.Code Morphing.using speculation recovery and adaptive retranslation to address real-life challenges.(CGO)

List of Binary Translation

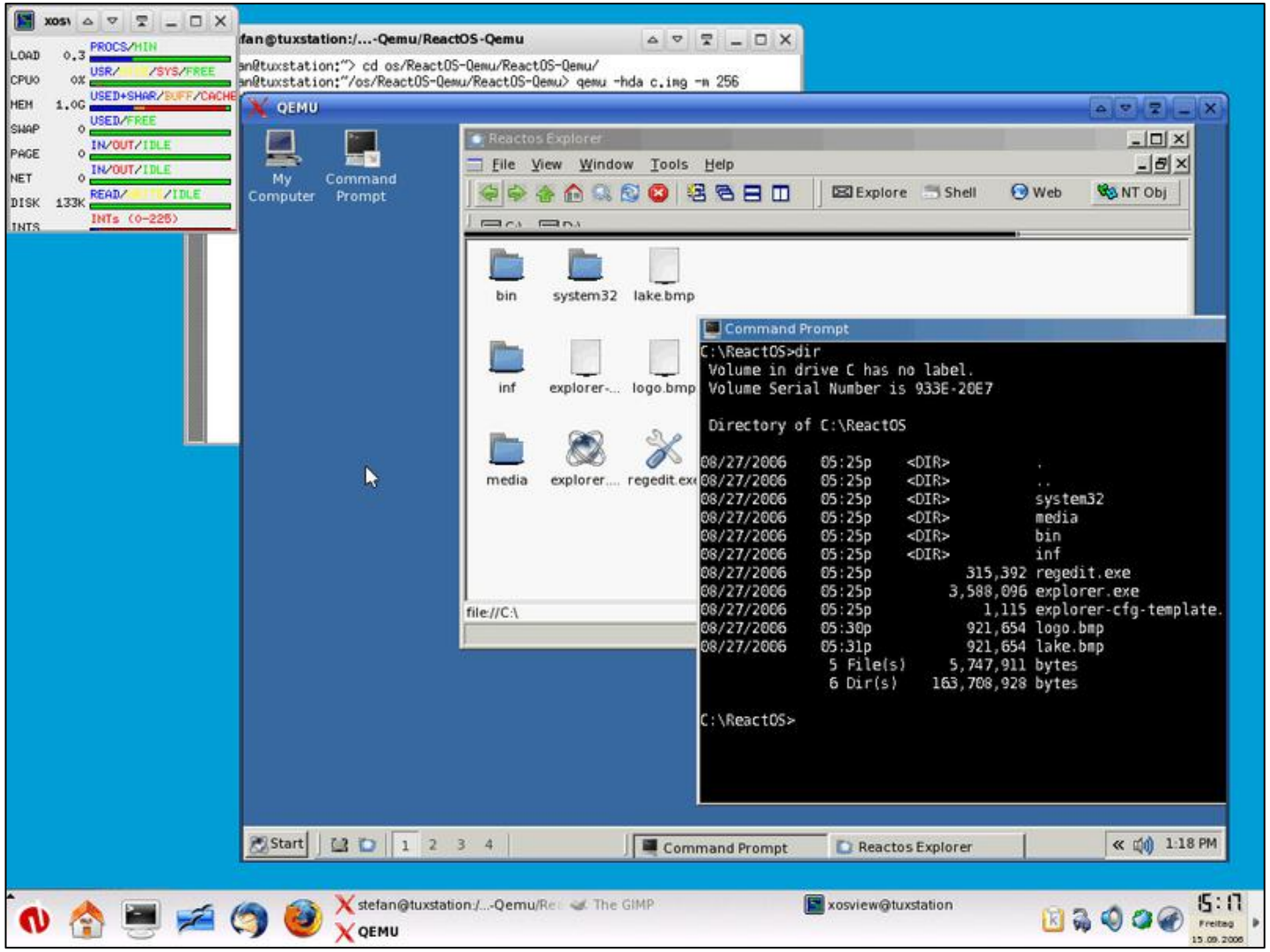
2005 QEMU

User-Mode & Full-system DBT

Guest x86, ARM, ...

Host x86, ARM, ...

- Tiny Code Generator
- Machine Desctiptions
- Devices Emulation
- ...



2005.QEMU.a Fast and Portable Dynamic Translator.(USENIX)

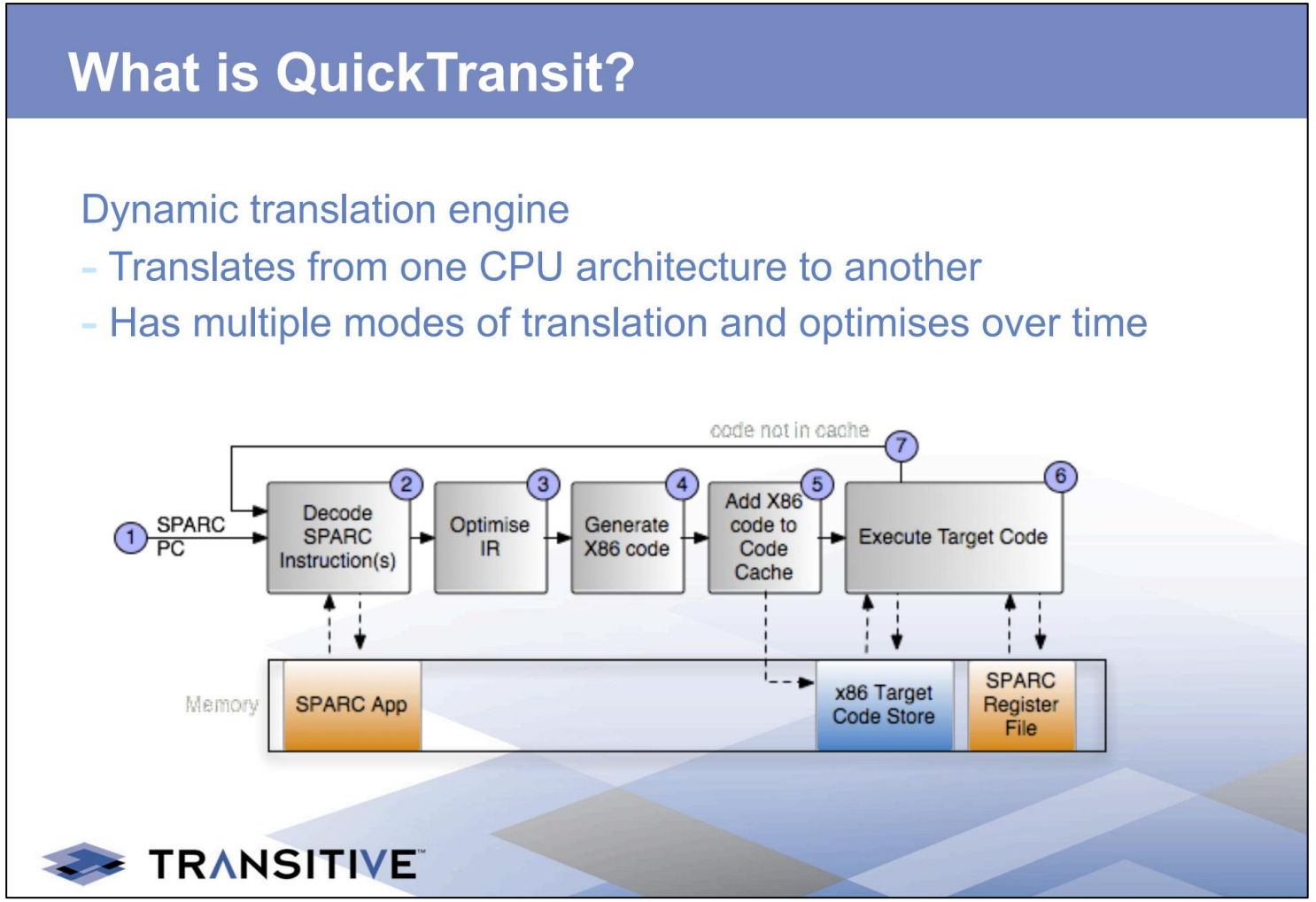
List of Binary Translation

2008 QuickTransit

User-Mode & Full-system DBT

the company behind Apple's Rosetta

- User: IBM PowerVM Lx86
- Sys: Solaris/SPARC to Linux/x86-64
- KVM + QuickTransit
  - Shadow Page Table
  - Paravirtualizing



https://quick-transit.com/

List of Binary Translation

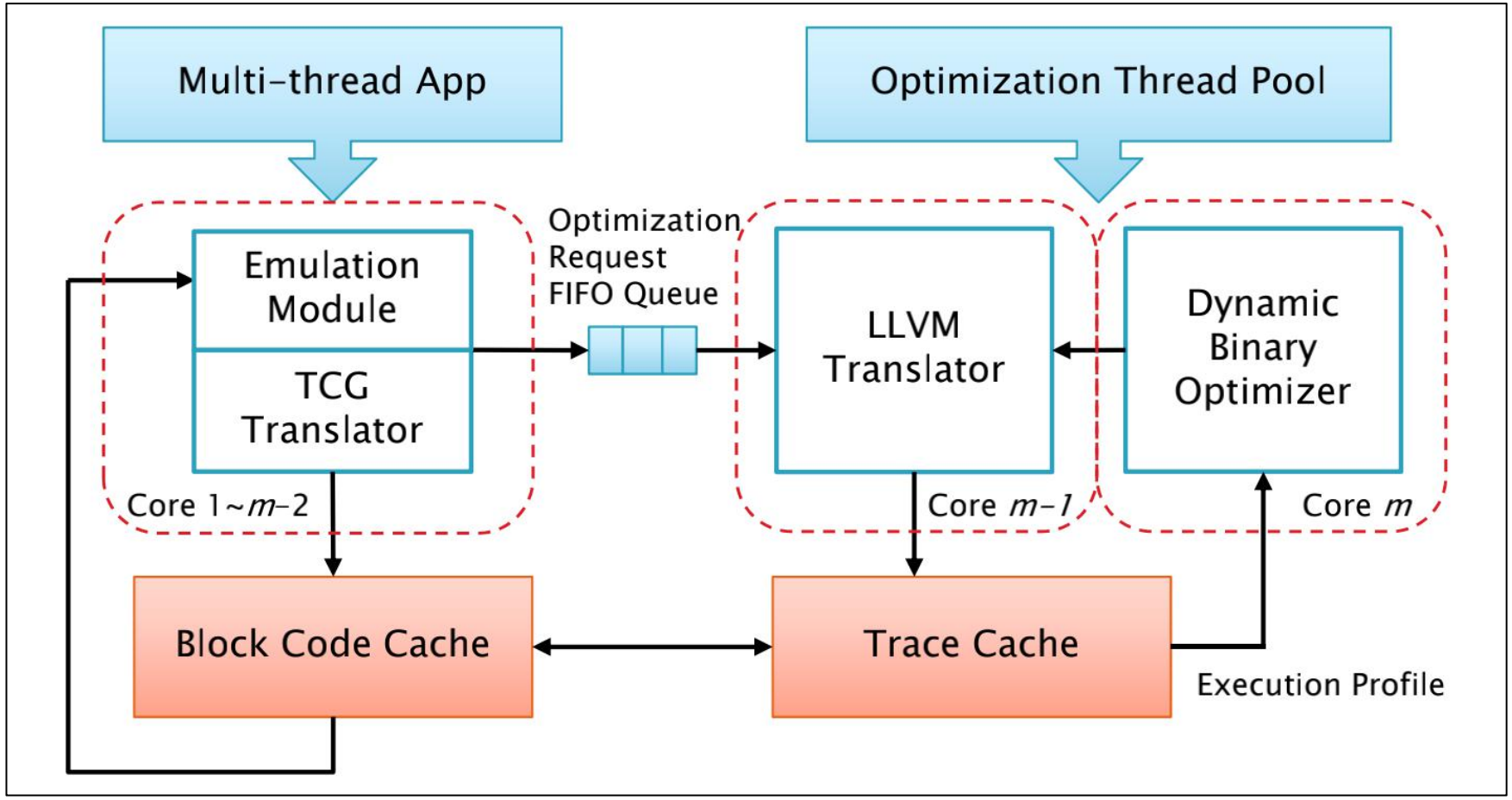
2013 HQEMU

User-Mode DBT

Guest x86, ARM

Host x86, ARM, PPC

- TCG -> LLVM -> Host
- Background Optimization

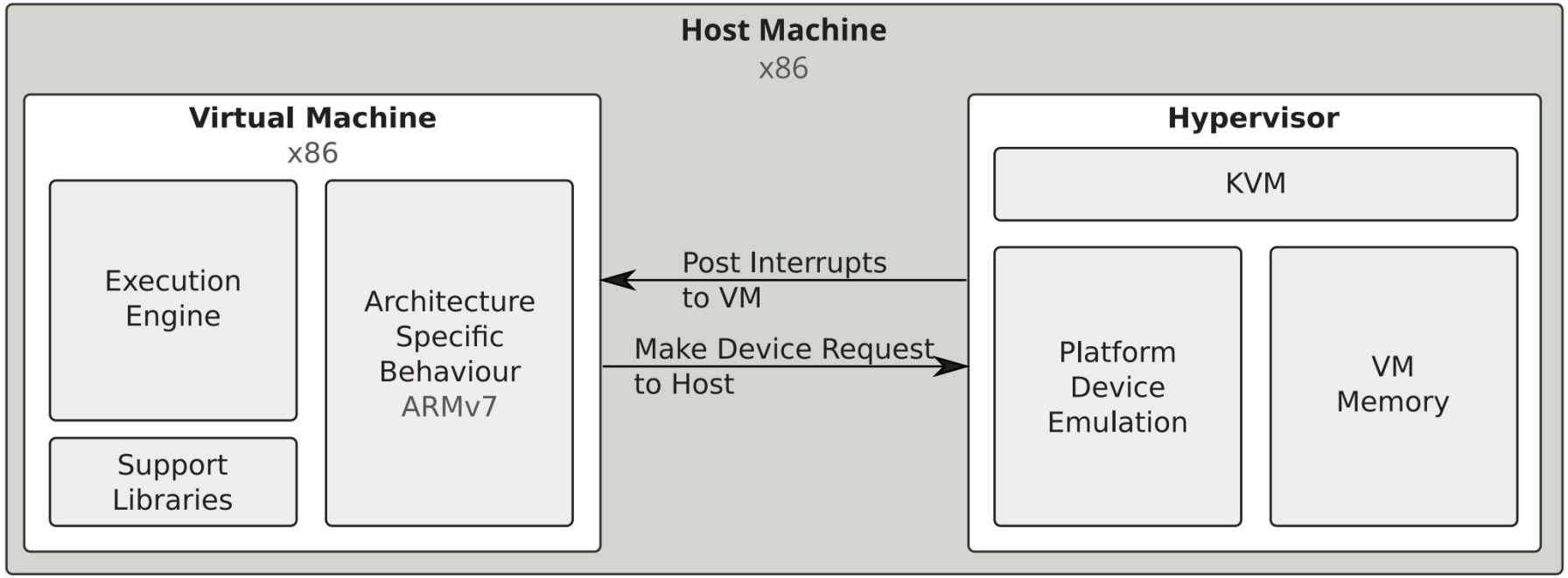


2013. Effificient and Retargetable Dynamic Binary Translation. (清华PhD)

List of Binary Translation

2016 Captive

2019 Captive 2.0



Full-System DBT

Guest ARM-v7

Host x86_64

- Hypervisor
- Execution Engine
- Architectural Implementation

2016. Hardware-Accelerated Cross-Architecture Full-System Virtualization. (ACM TACO)

2019. A Retargetable System-Level DBT Hypervisor. (USENIX)

List of Binary Translation

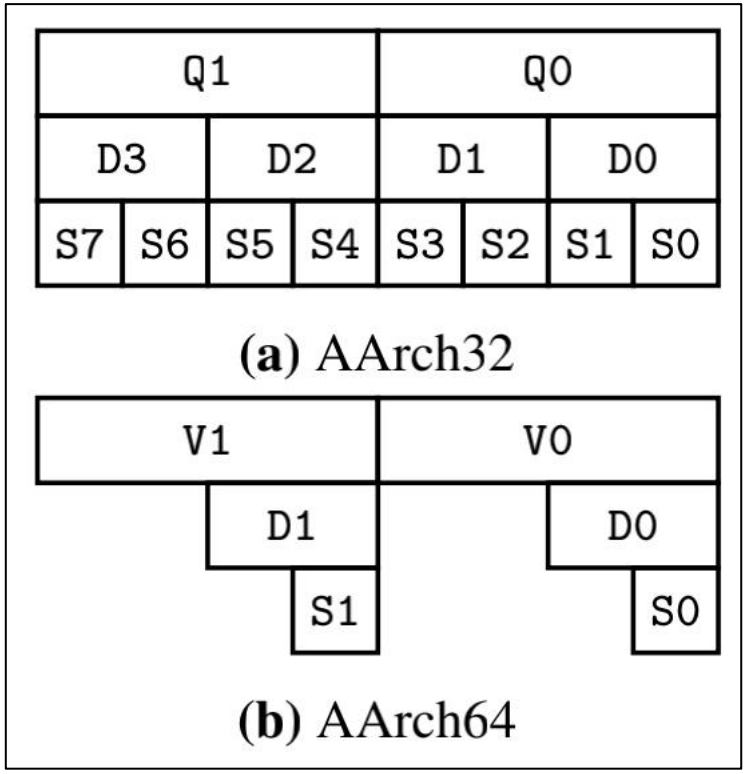
2017 MAMBO-X64

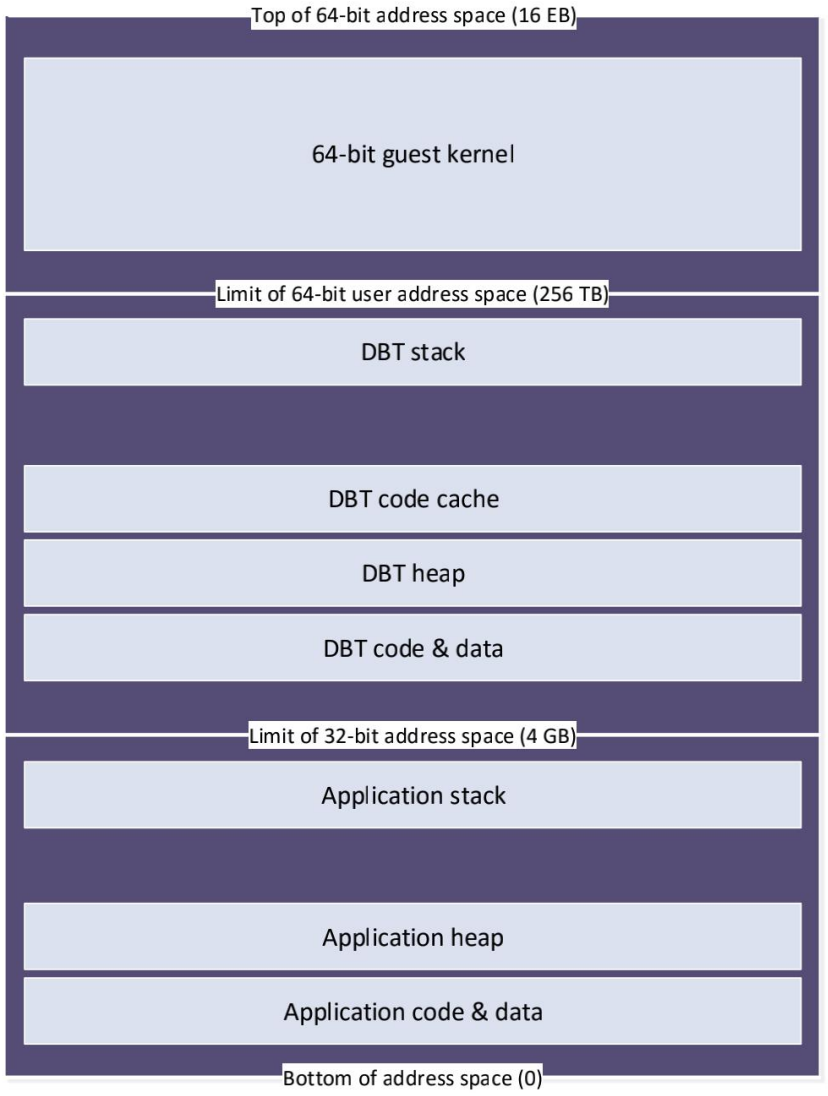
User-Mode DBT

Guest AArch 32

Host AArch 64

- Float Point Register Mapping
- Load/Store Modes
- Trace Compilation Algorithm
- System Signals





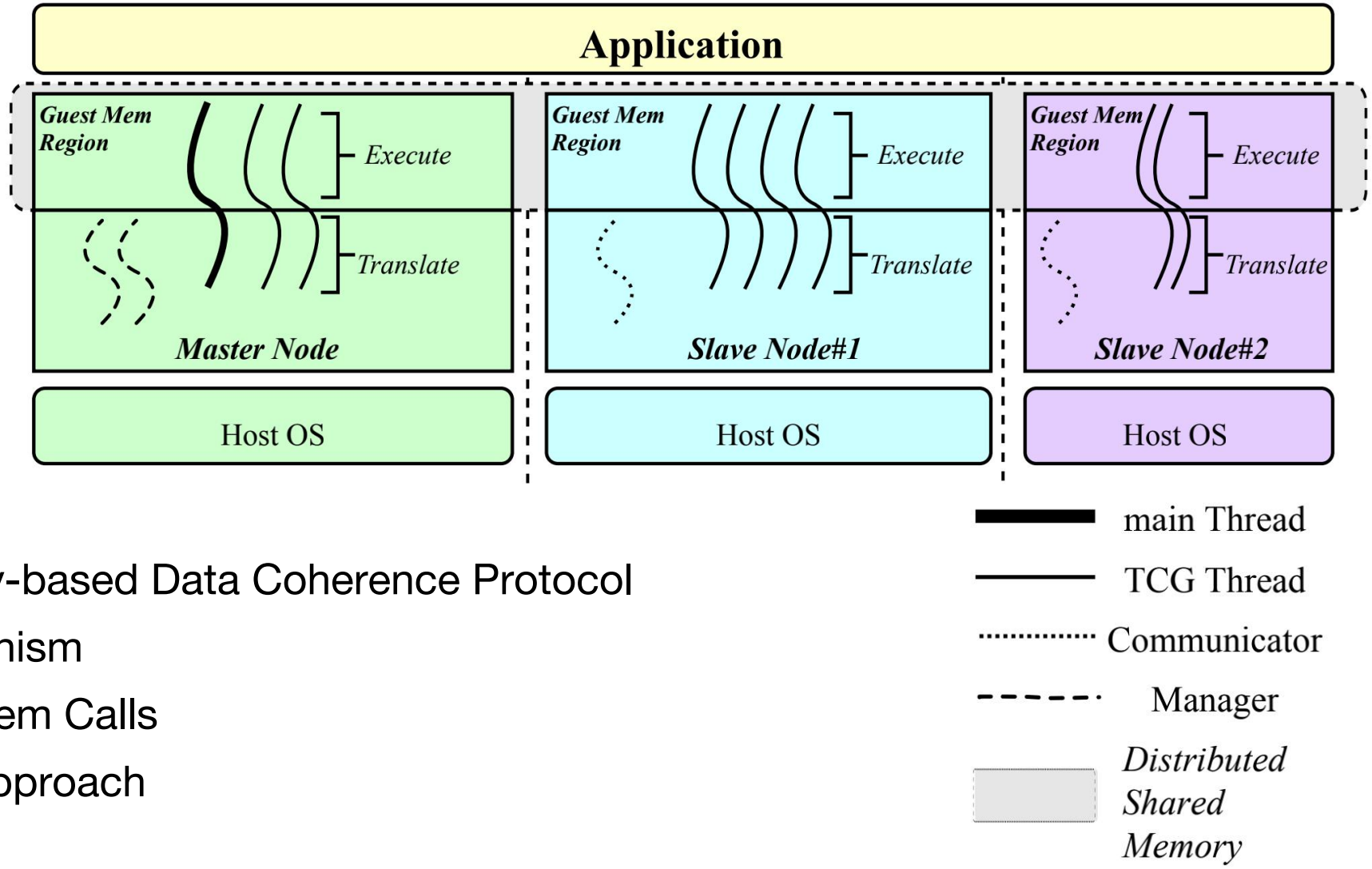
2017. Low Overhead Dynamic Binary Translation on ARM. (ACM SIGPLAN)

List of Binary Translation

2020 DQEMU

User-Mode DBT

Guest Multiple Thread

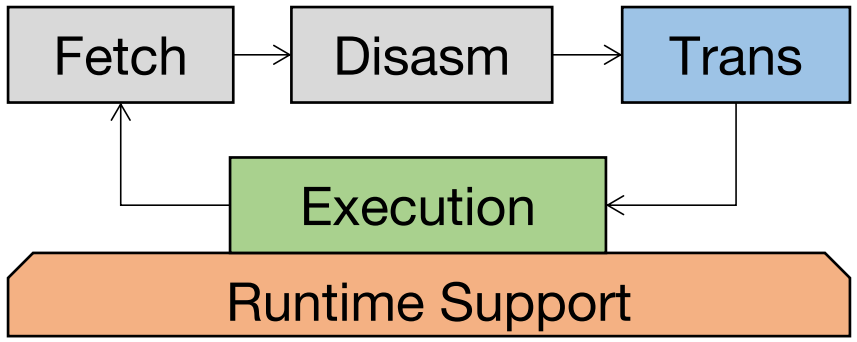


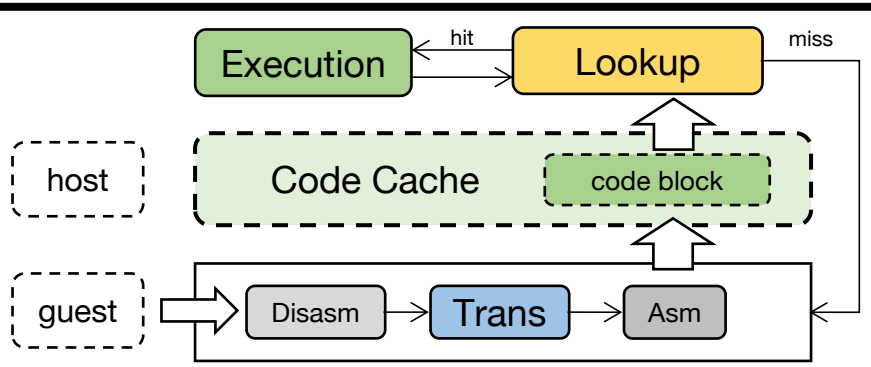
- Page_x0002_Level Directory-based Data Coherence Protocol
- Hierarchical Locking Mechanism
- Delegation Scheme for System Calls
- Remote Thread Migration Approach

2020. DQEMU- A Scalable Emulator with Retargetable DBT on Distributed Platforms. (ICPP)

A Survey on Binary Translation

Binary Rewrite	=	Translation	Optimization	Instrumentation
Script Language	=	JIT / Java / JVM	Interpret	Python, Matlab
Virtualization	=	KVM	Wine	Android	Gem5





- Optimization
  - branch
  - code cache
  - block lookup
  - better codes
  - float point
  - simd
  - atomic
  - .....
  - sys
    - address
    - interrupt
    - ...
- Common Problems
  - SMC
  - precise exception
  - cross-ISA: ABI, ...
  - ...

1994 Shadw
1996 Embra
1997 FX!32
2000 DAISY
2000 Dynamo
2000 HP Aries
2000 UQBT
2003 Code Morphing
2005 QEMU
.....

6 Summary

A Survey on Binary Translation

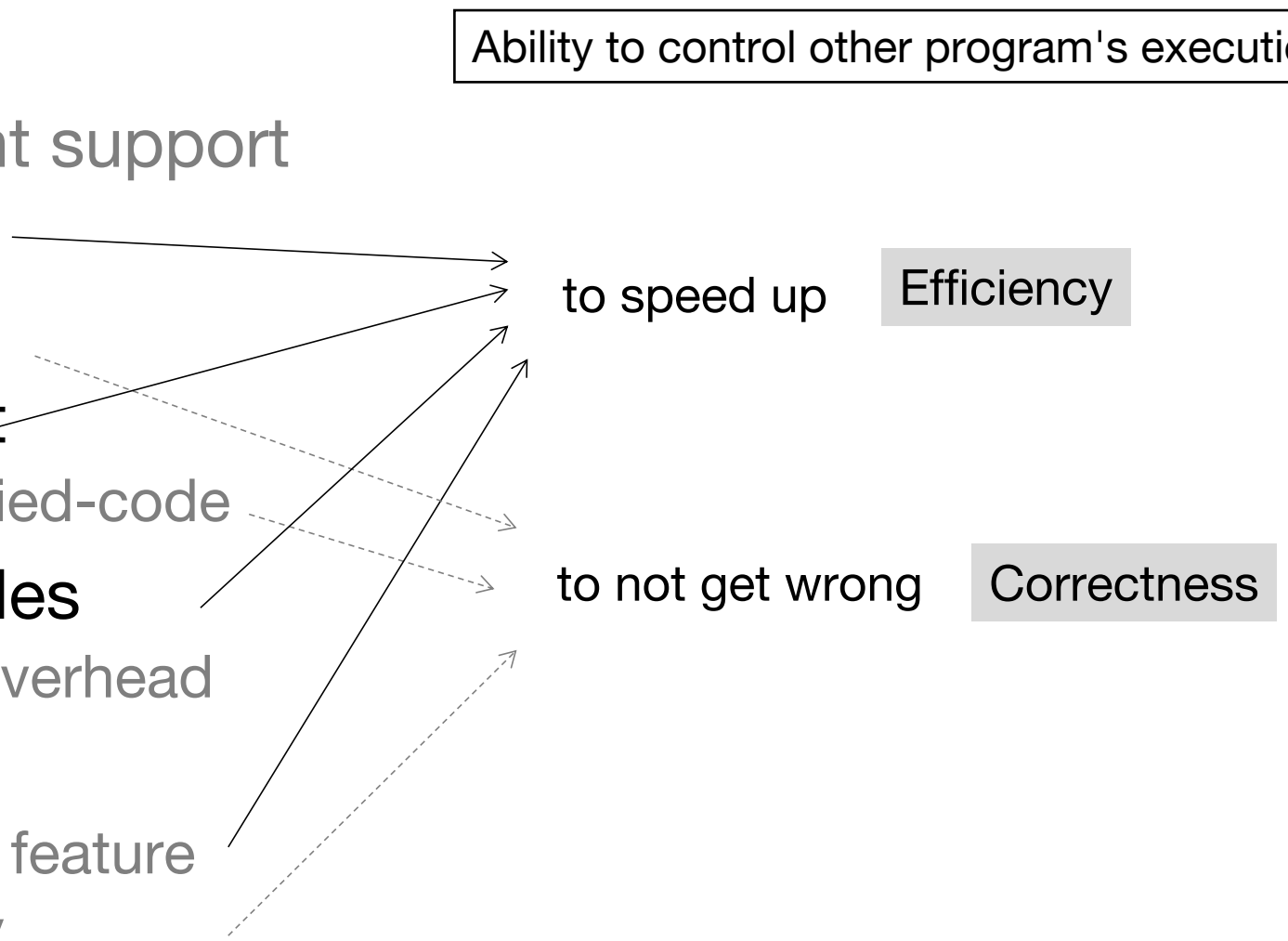
- Introduction ..... basic architecture

- Related Technologies ..... differences and similarities
- Architecture Details ..... and basic optimizations
- Advanced Topics ..... problems, challenges, optimizations, etc.

- List of Binary Translation ..... by year, type, etc.
- Summary

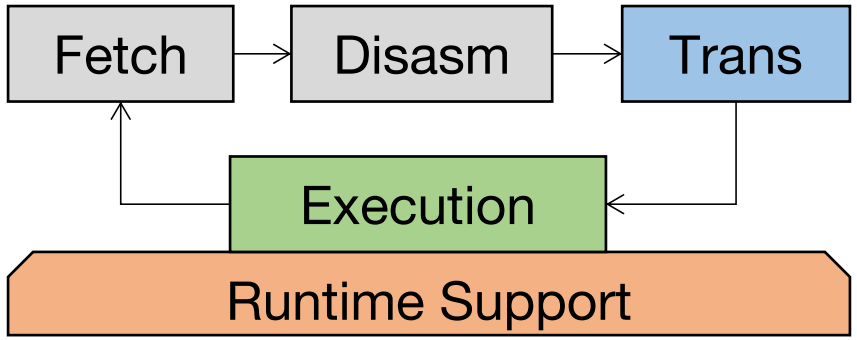
Summary

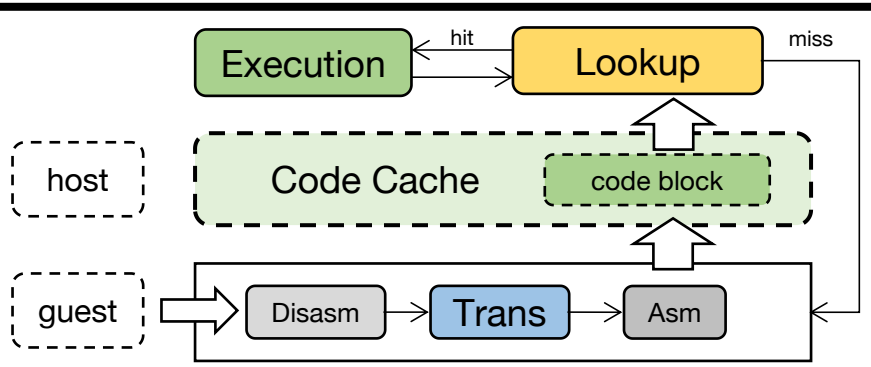
- Interpretation
  - execution environment support
- Translate and Execution
  - precise exception
- Reuse Translation Result
  - copy problem: self-modified-code
- Optimize Generated Codes
  - profile and optimization overhead
- Cross-ISA
  - FP, SIMD, ... architecture feature
  - ABI, memory consistency



A Survey on Binary Translation

Binary Rewrite	=	Translation	Optimization	Instrumentation
Script Language	=	JIT / Java / JVM	Interpret	Python, Matlab
Virtualization	=	KVM	Wine	Android	Gem5





- Optimization
  - branch
  - code cache
  - block lookup
  - better codes
  - float point
  - simd
  - atomic
  - .....
  - sys
    - address
    - interrupt
    - ...
- Common Problems
  - SMC
  - precise exception
  - cross-ISA: ABI, ...
  - ...

1994 Shadw
1996 Embra
1997 FX!32
2000 DAISY
2000 Dynamo
2000 HP Aries
2000 UQBT
2003 Code Morphing
2005 QEMU
.....

Bianry Translation

Efficiency

Correctness