Micro-Policies

Formally Verified Low-Level Tagging Schemes for Safety and Security

Cătălin Hrițcu

INRIA Paris-Rocquencourt

Computer systems are insecure

Computer systems are insecure

- Today’s CPUs are mindless bureaucrats
  - “write past the end of this buffer” ... *yes boss!*
  - “jump to this untrusted integer” ... *right boss!*
  - “return into the middle of this instruction” ... *sure boss!*

Computer systems are insecure

- Today's CPUs are mindless bureaucrats
  - "write past the end of this buffer" ... *yes boss!*
  - "jump to this untrusted integer" ... *right boss!*
  - "return into the middle of this instruction" ... *sure boss!*
- Software bears most of the burden for security

  pervasive security enforcement impractical

  security-performance tradeoff

  just write secure code ... all of it!

Computer systems are insecure

- Today's CPUs are mindless bureaucrats
  - "write past the end of this buffer" ... *yes boss!*
  - "jump to this untrusted integer" ... *right boss!*
  - "return into the middle of this instruction" ... *sure boss!*
- Software bears most of the burden for security

  pervasive security enforcement impractical

  security-performance tradeoff

  just write secure code ... all of it!

Consequence:

  tons of vulnerabilities in every large system

  - violations of known safety and security policies

Micro-Policies

- general efficient dynamic enforcement mechanism for
  - critical invariants of low-level code
  - high-level abstractions and programming models

Micro-Policies

- general efficient dynamic enforcement mechanism for
  - critical invariants of low-level code
  - high-level abstractions and programming models
- add large metadata **tags** to all machine words
  - “this word is an instruction, and this one is a pointer”
    “this word comes from the net, and this one is private”
    tag structure defined entirely by software

Micro-Policies

- general efficient dynamic enforcement mechanism for
  - critical invariants of low-level code
  - high-level abstractions and programming models
- add large metadata **tags** to all machine words
  - "this word is an instruction, and this one is a pointer"
    "this word comes from the net, and this one is private"
    tag structure defined entirely by software

  tags efficiently propagated on each instruction
  - **rules** *defined by software* (fault handler; verified)
  - rule lookup *accelerated by hardware* rule cache

Micro-Policies for ...

We already thoroughly explored: dynamic information flow control (IFC)

Micro-Policies for ...

We already thoroughly explored: dynamic information flow control (IFC)

Currently exploring:

user-kernel distinction

- hardware types
  - int vs. pointer vs. instruction

memory safety

stop all spatial and temporal violations on heap and stack

- pointers become capabilities

control-flow integrity

- call-stack protection
- opaque closures
  - first-class functions (λ)
- linear pointers
  - absence of aliasing

Micro-Policies for ...

We already thoroughly explored: dynamic information flow control (IFC)

Currently exploring:

- user-kernel distinction
- hardware types
  - int vs. pointer vs. instruction
- memory safety
  - stop all spatial and temporal violations on heap and stack
  - pointers become capabilities
- control-flow integrity
- call-stack protection
- opaque closures
  - first-class functions (λ)
- linear pointers
  - absence of aliasing

Longer term plans:

- pointer permissions
  - "readable", "writeable“, "jumpable", or "callable"
- process isolation
  - replacement for virtual memory
- dynamic type tags
  - for C, Scheme, or even OCaml
- dynamic sealing & trademarks
  - cache result of dynamic contracts
- higher-order contracts
- data race detection
- user-defined metadata

Micro-Policies for ...

We already thoroughly explored: dynamic information flow control (IFC)

Currently exploring:

- user-kernel distinction
- hardware types
  - int vs. pointer vs. instruction
- memory safety
  - stop all spatial and temporal violations on heap and stack
  - pointers become capabilities
- control-flow integrity
- call-stack protection
- opaque closures
  - first-class functions (λ)
- linear pointers
  - absence of aliasing

Longer term plans:

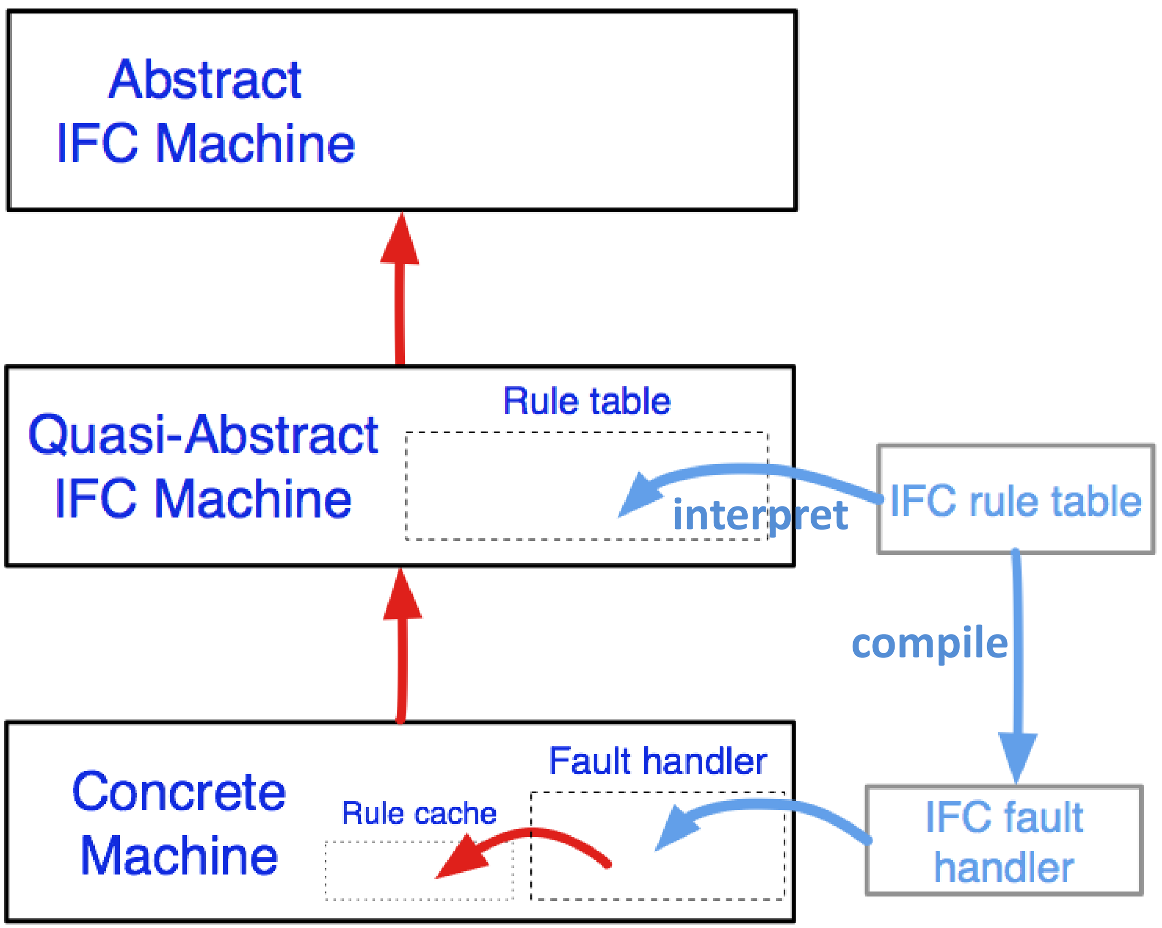
- pointer permissions
  - "readable", "writeable“, "jumpable", or "callable"
- process isolation
  - replacement for virtual memory
- dynamic type tags
  - for C, Scheme, or even OCaml
- dynamic sealing & trademarks
  - cache result of dynamic contracts
- higher-order contracts
- data race detection
- user-defined metadata

IFC Micro-Policy

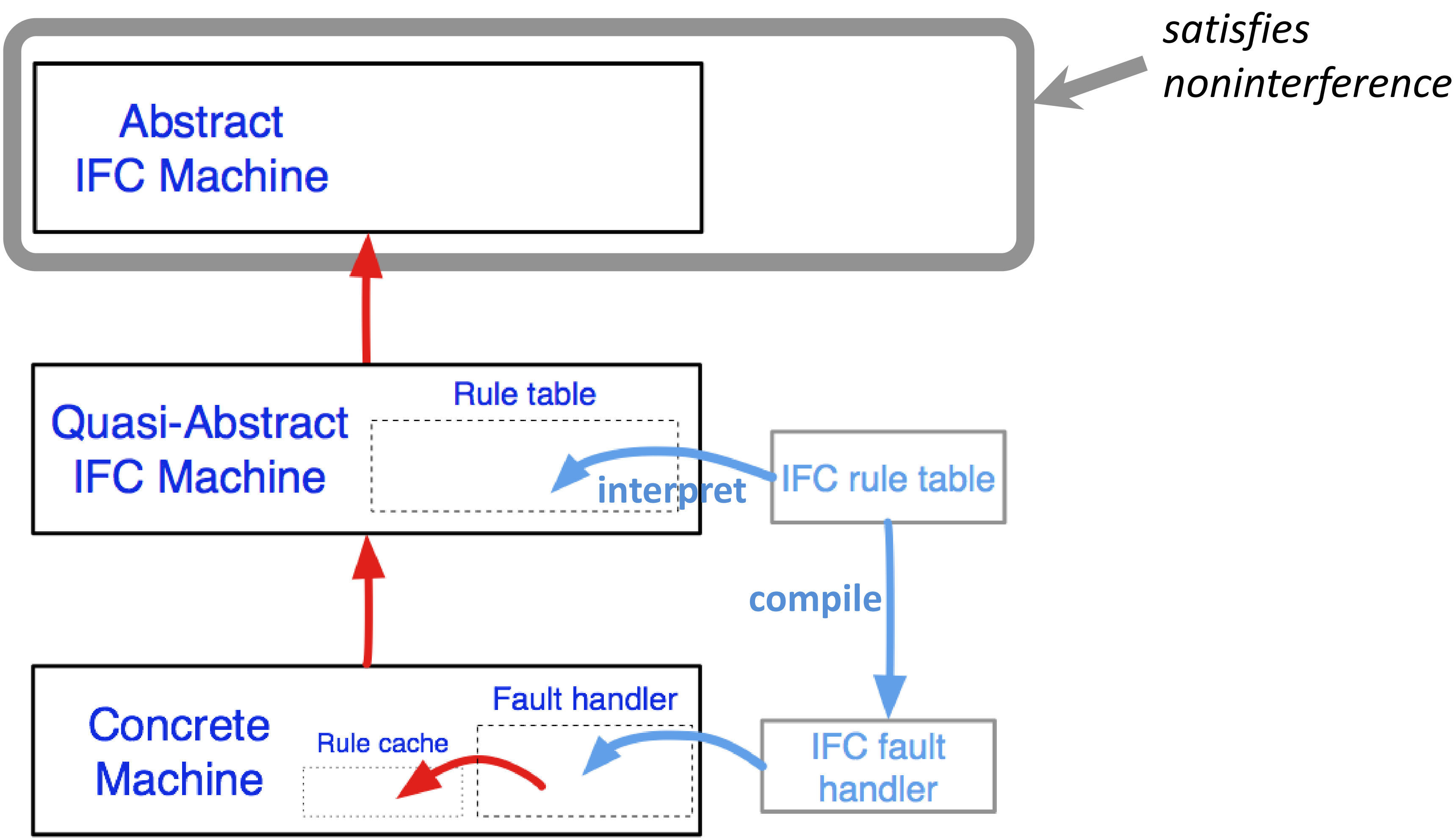
opcode	*allow*	e_{rpc}	e_r
sub	TRUE	$\mathtt{LAB}_{pc}$	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_2$
output	TRUE	$\mathtt{LAB}_{pc}$	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_{pc}$
push	TRUE	$\mathtt{LAB}_{pc}$	BOT
load	TRUE	$\mathtt{LAB}_{pc}$	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_2$
store	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_{pc} \sqsubseteq \mathtt{LAB}_3$	$\mathtt{LAB}_{pc}$	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_2 \sqcup \mathtt{LAB}_{pc}$
jump	TRUE	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_{pc}$	--
bnz	TRUE	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_{pc}$	--
call	TRUE	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_{pc}$	$\mathtt{LAB}_{pc}$
ret	TRUE	$\mathtt{LAB}_1$	--

- A Verified Information Flow Architecture [POPL 2014]
- Testing Noninterference, Quickly [ICFP 2013]
- All Your IFCException Are Belong To Us [S&P 2013]
- A Theory of Information-Flow Labels [CSF 2013]

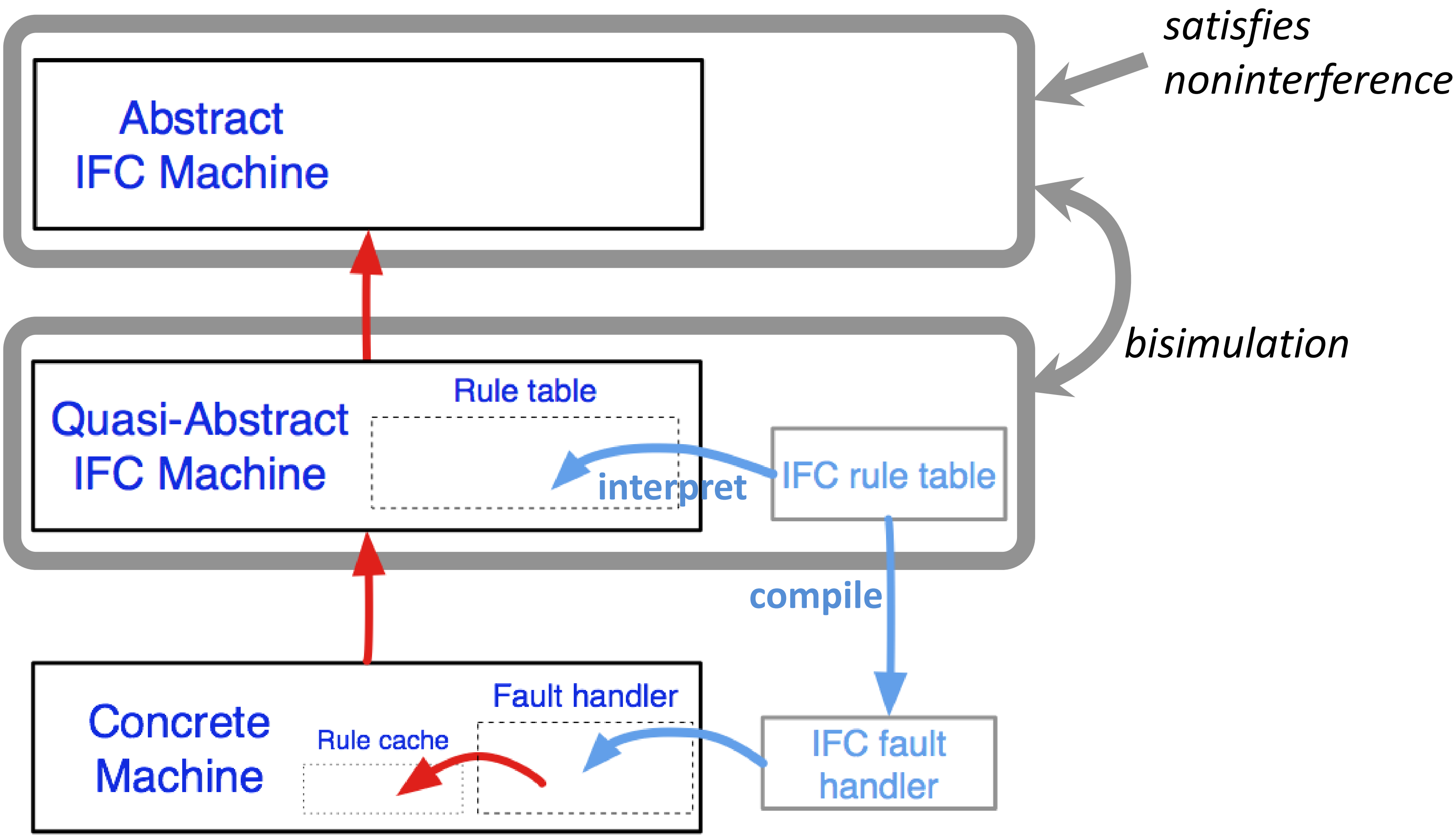
Noninterference proof in Coq [POPL 2014]



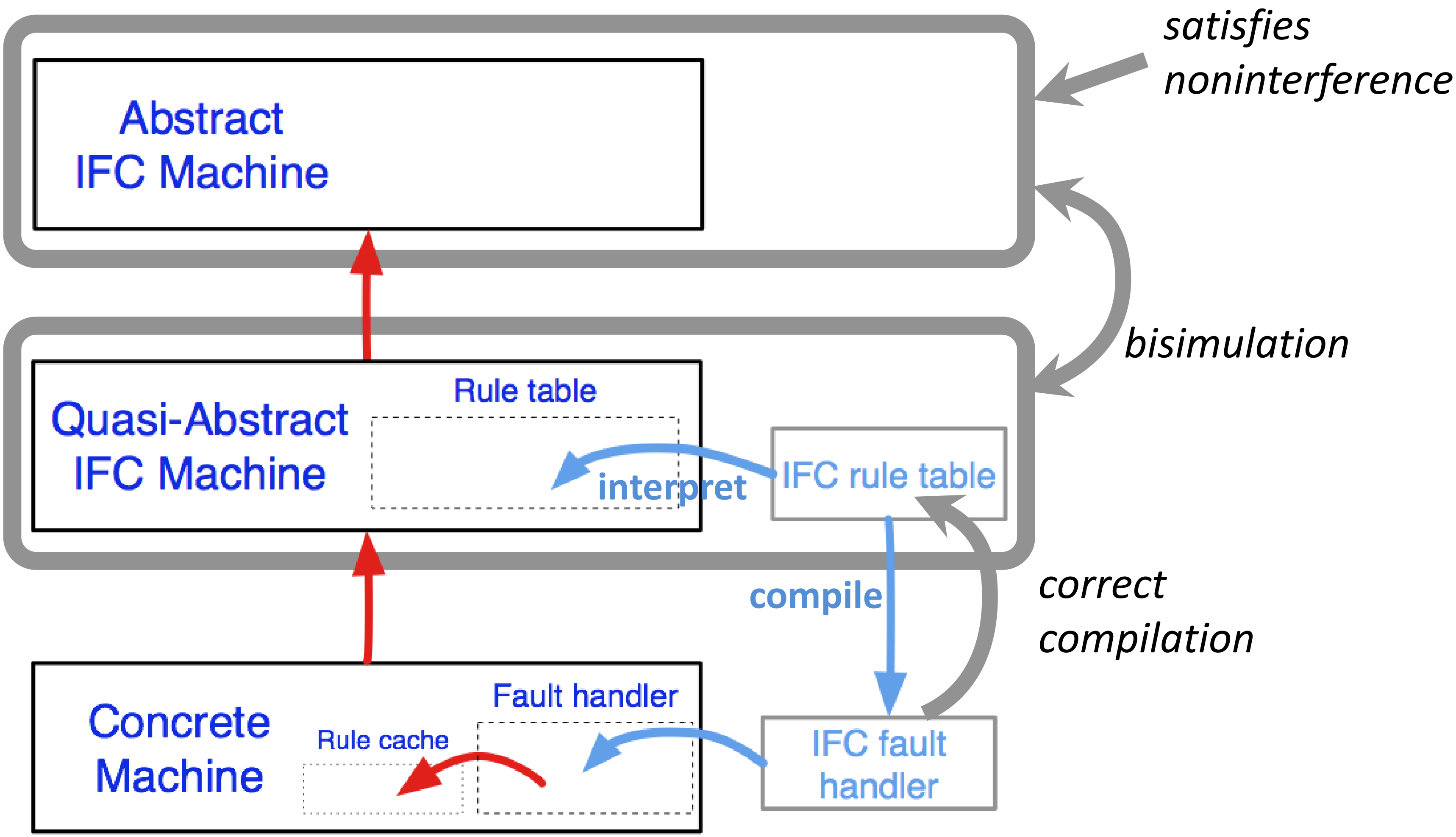
Noninterference proof in Coq [POPL 2014]



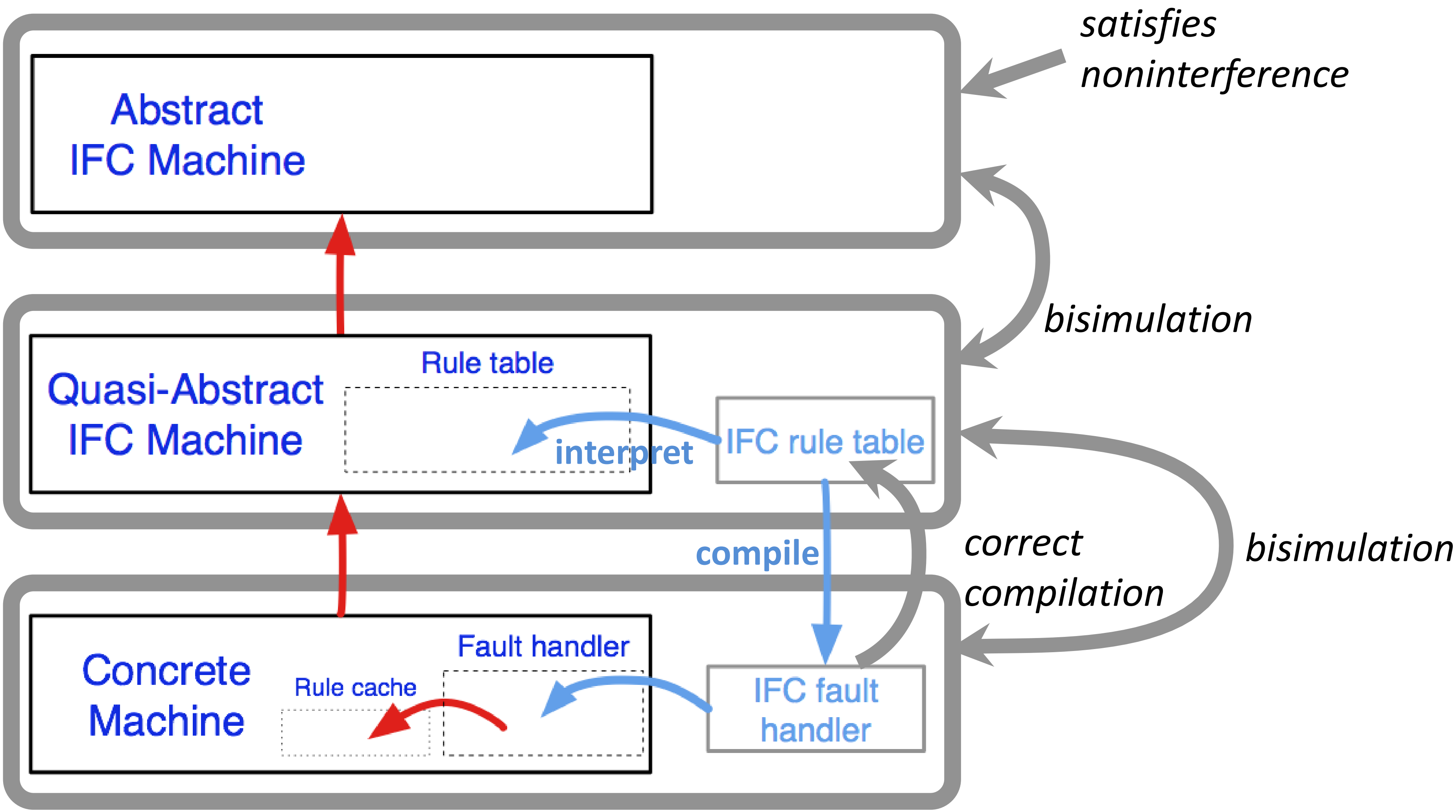
Noninterference proof in Coq [POPL 2014]



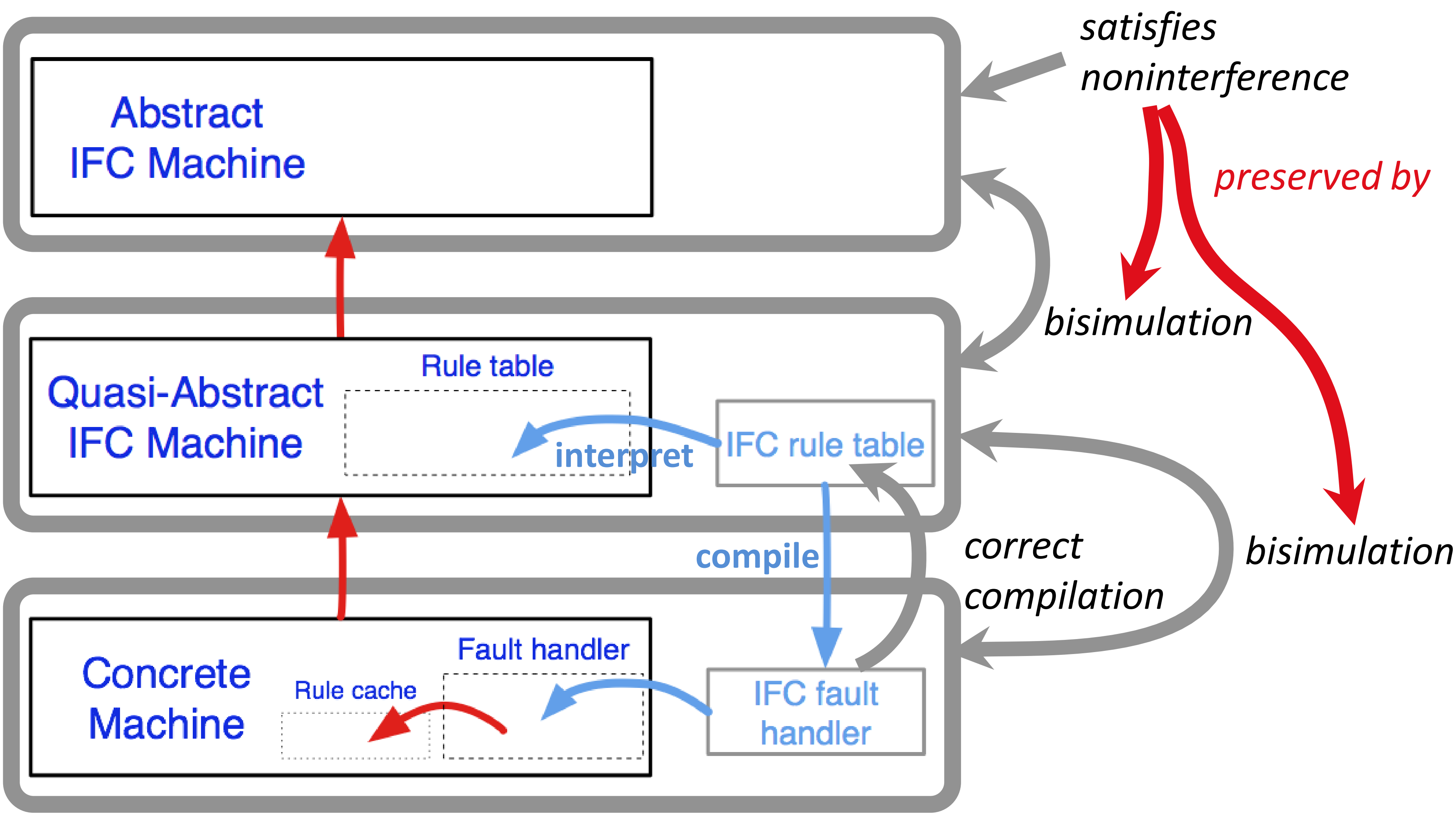
Noninterference proof in Coq [POPL 2014]



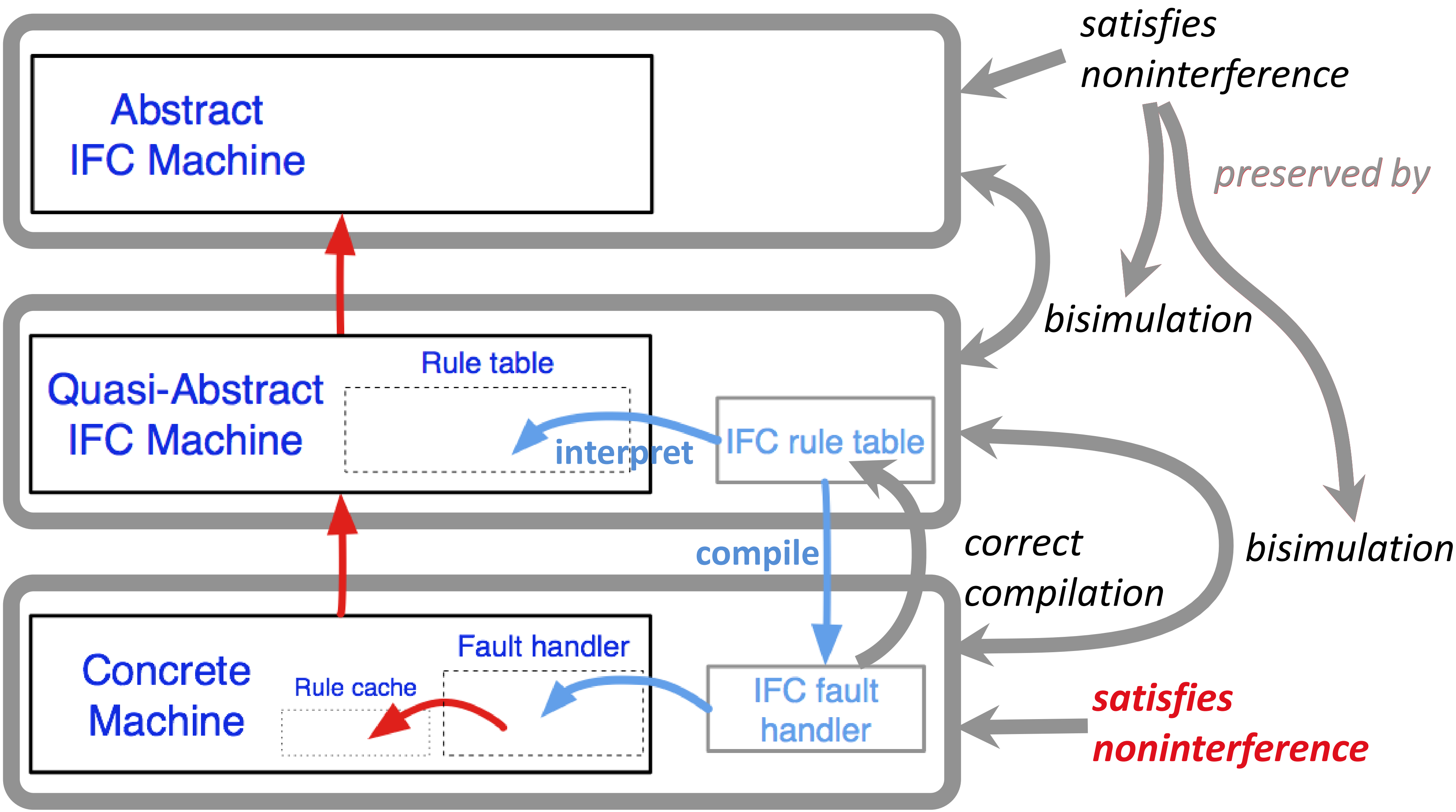
Noninterference proof in Coq [POPL 2014]



Noninterference proof in Coq [POPL 2014]



Noninterference proof in Coq [POPL 2014]



Memory safety

- Goal: prevent all memory safety violations
  - **spatial violations**: accessing arrays out of bounds
  - **temporal violations**:
    dereferencing pointer after its region was freed
  - for simplicity here only for heap-allocated data
    and excluding unpacked C structs
- Pointers become **unforgeable capabilities**
  - can only obtain a valid pointer to a memory region
    - by allocating that region or
    - by copying or offsetting an existing pointer to that region

Micro-Policy for memory safety

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

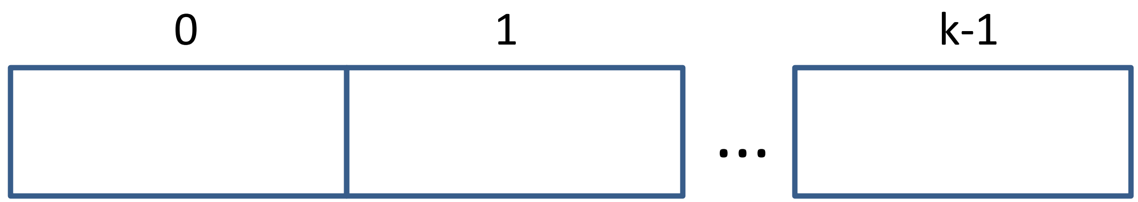
p ← alloc k

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

p ← alloc k

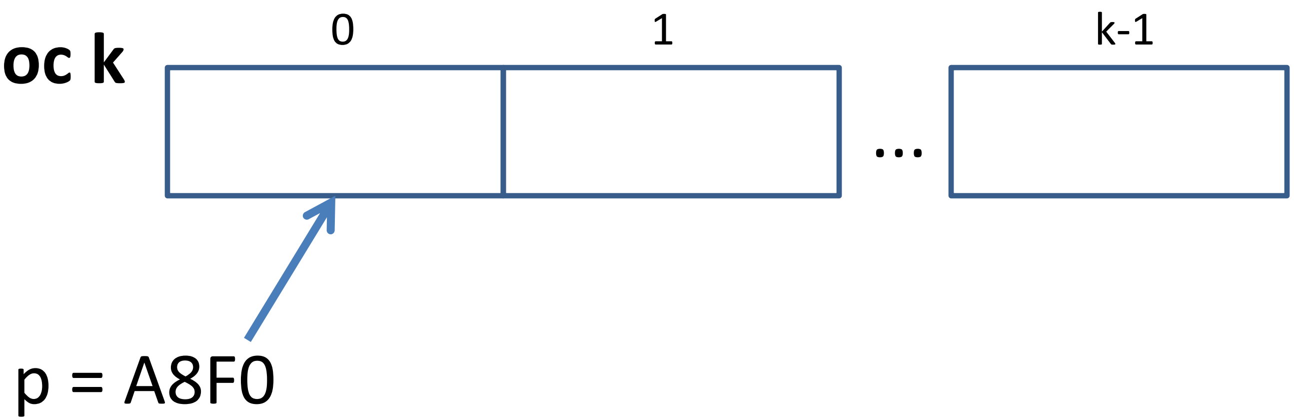


Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

p ← alloc k



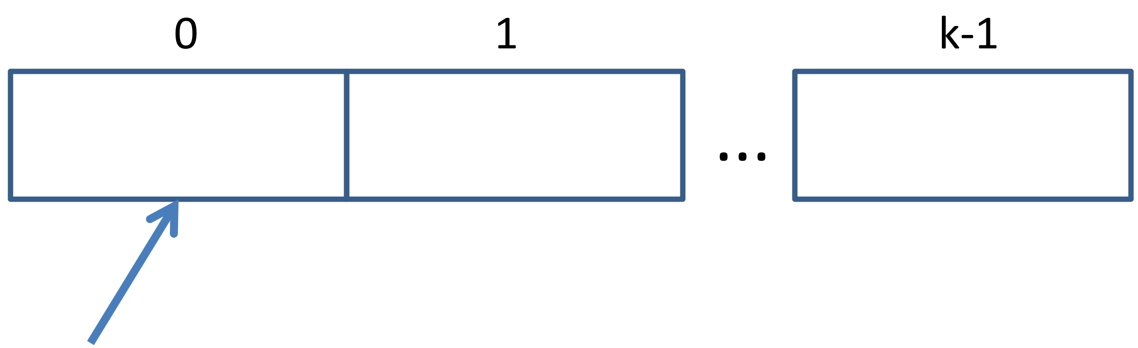
Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

p ← alloc k

fresh n



p = A8F0

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

p ← alloc k

fresh n

0	1		k-1
0@M(n,i)	0@M(n,i)	...	0@M(n,i)

p = A8F0

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

p ← alloc k

fresh n

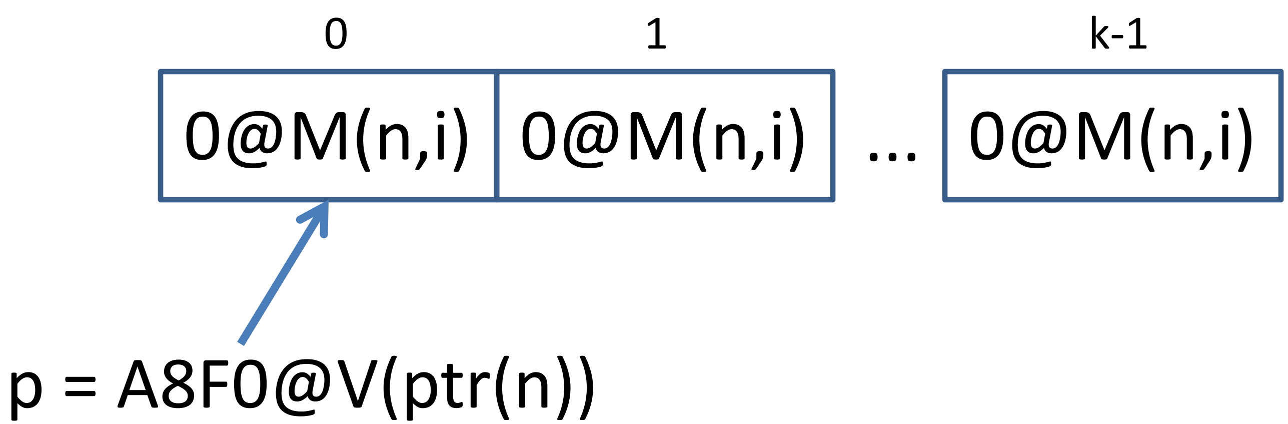
0	1		k-1
0@M(n,i)	0@M(n,i)	…	0@M(n,i)

p = A8F0@V(ptr(n))

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

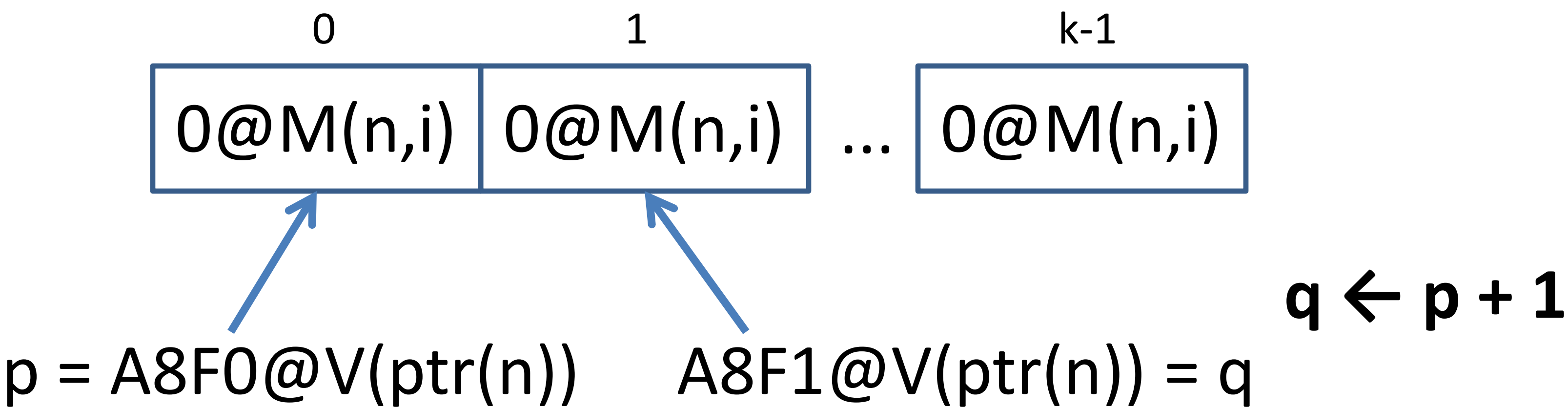


q ← p + 1

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

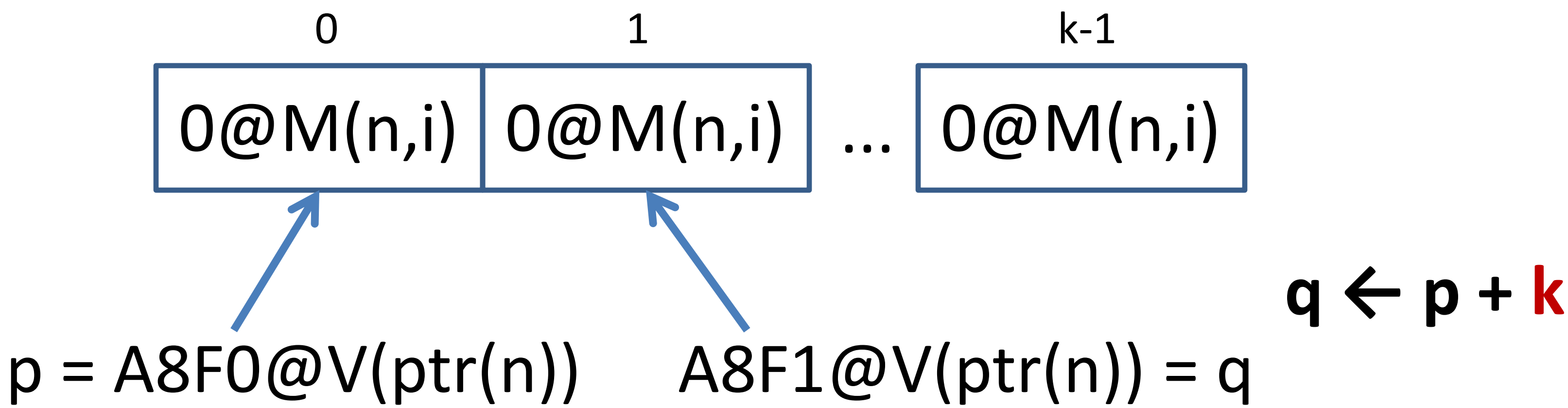
Micro-Policy for memory safety



Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

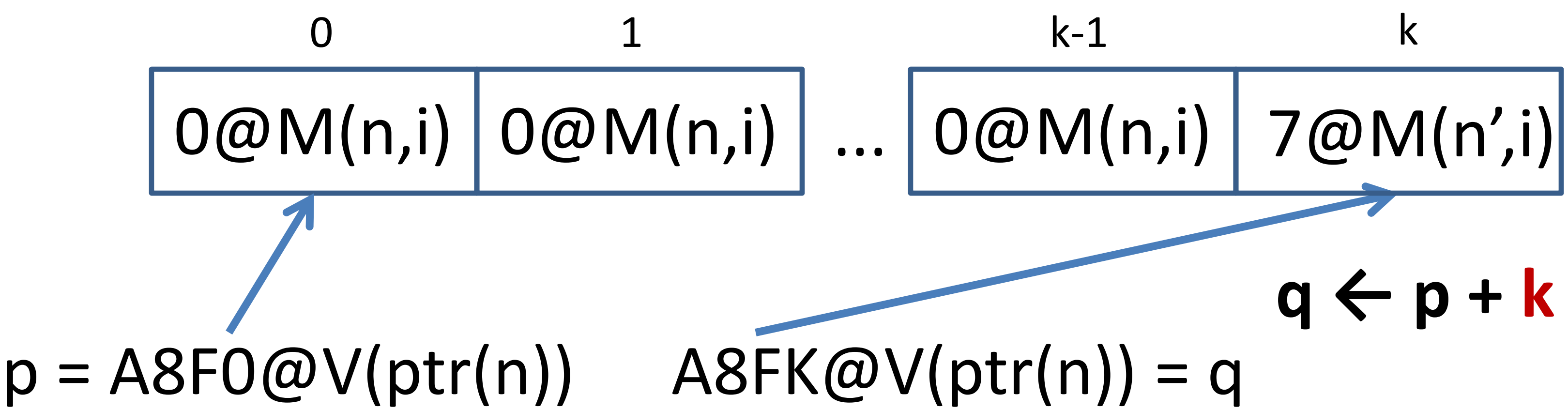
Micro-Policy for memory safety



Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

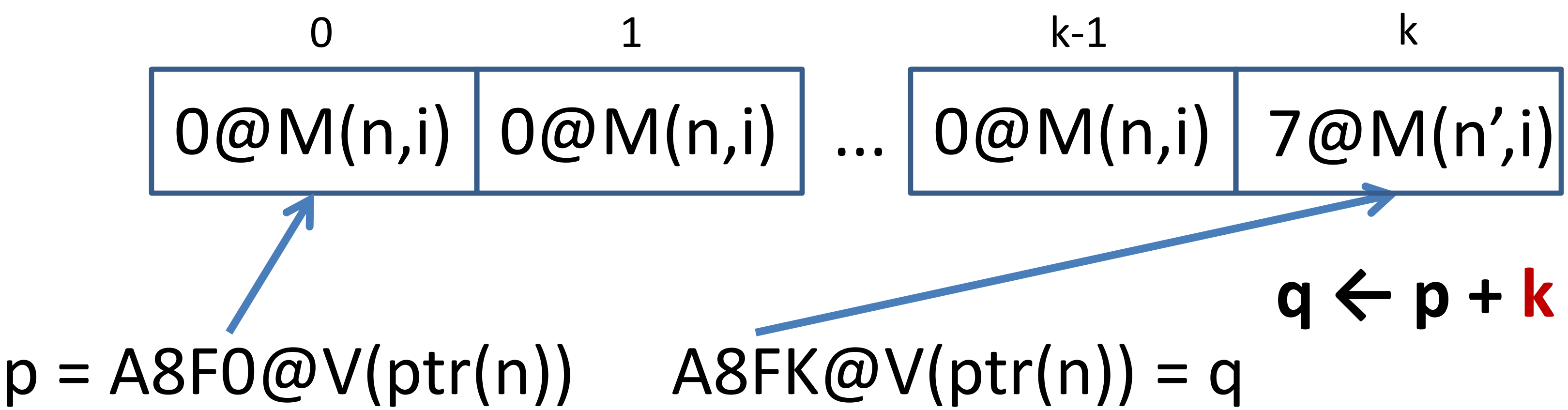
Micro-Policy for memory safety



Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

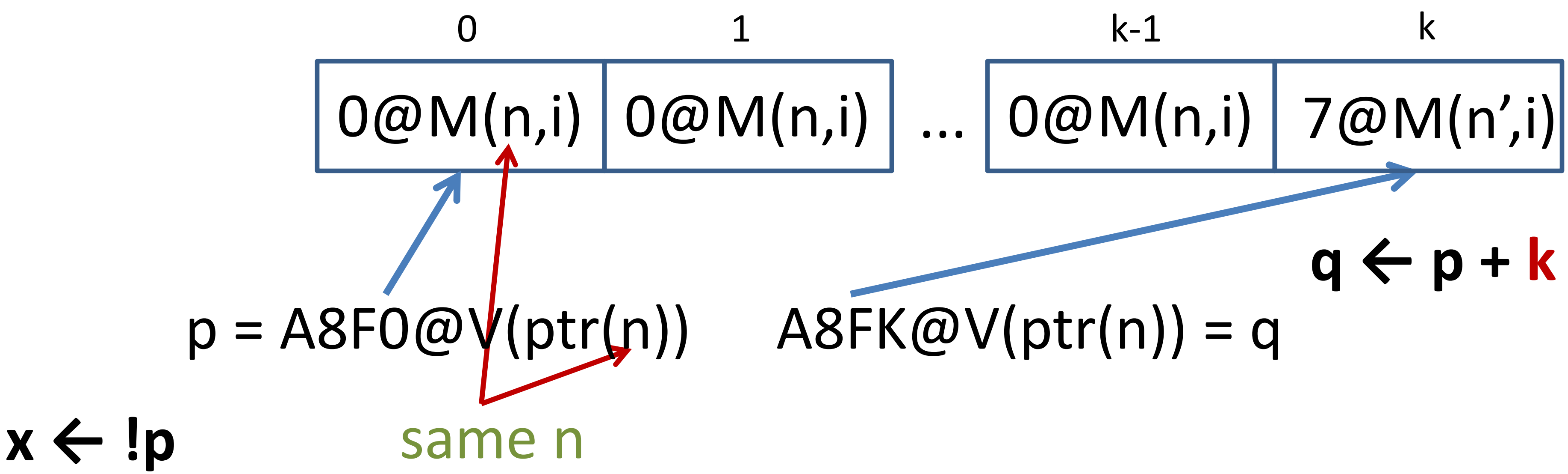


x ← !p

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety



Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

0	1		k-1	k
0@M(n,i)	0@M(n,i)	…	0@M(n,i)	7@M(n’,i)

q ← p + k

p = A8F0@V(ptr(n))   A8FK@V(ptr(n)) = q

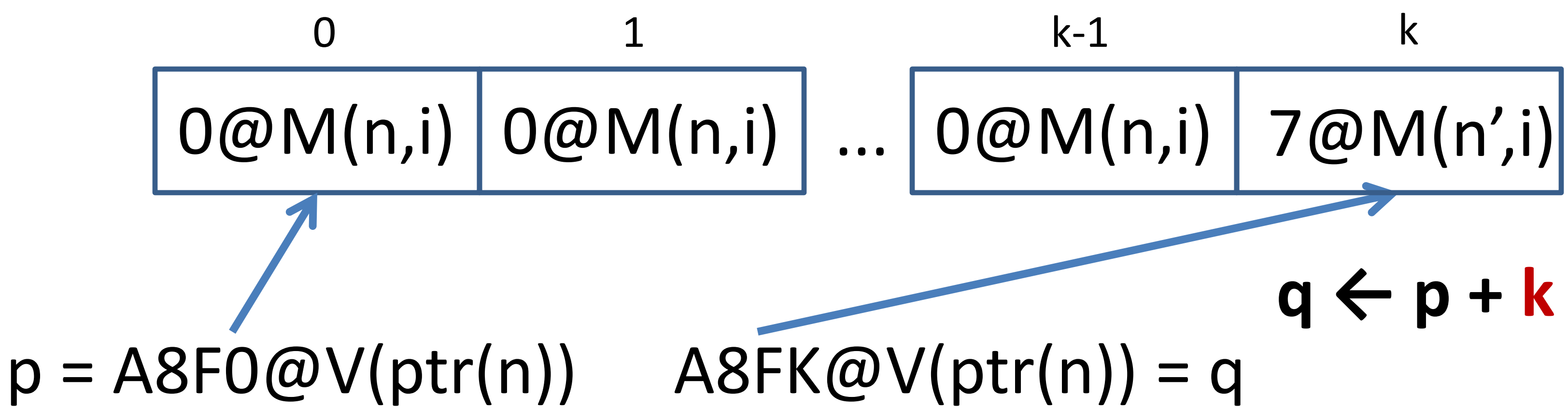
x ← !p   same n

x = 0@V(i)

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety



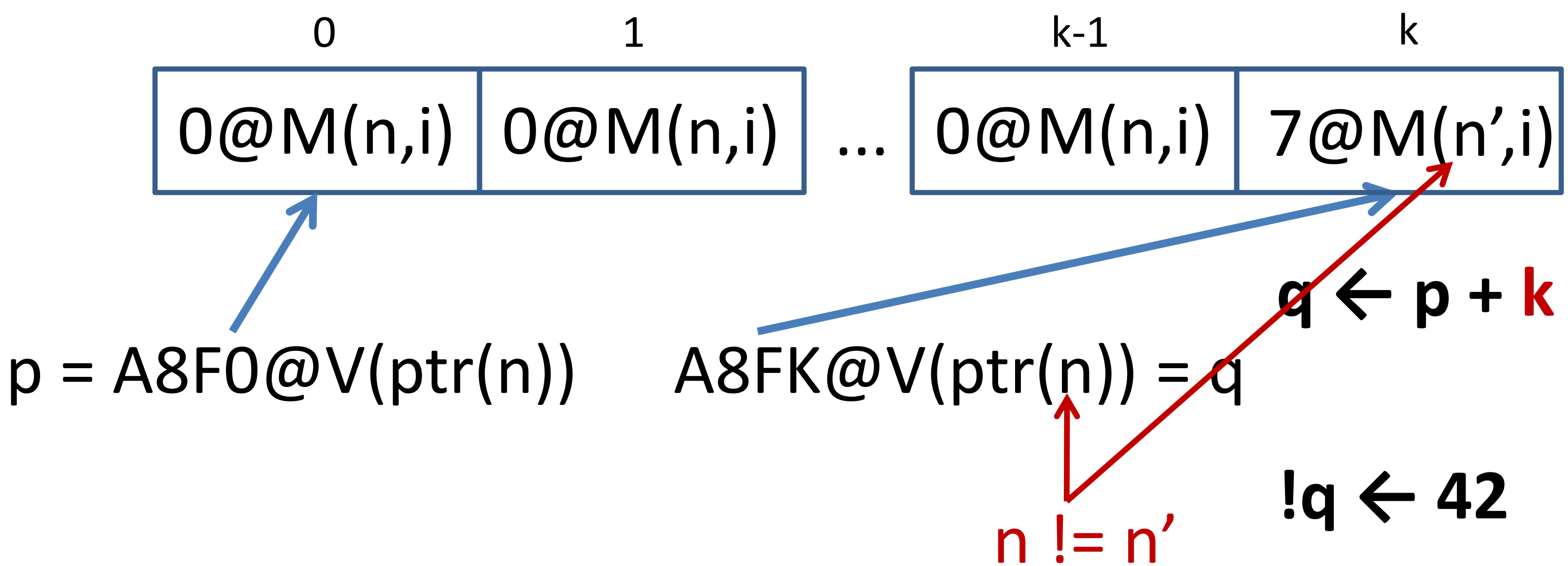
q ← p + k

!q ← 42

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

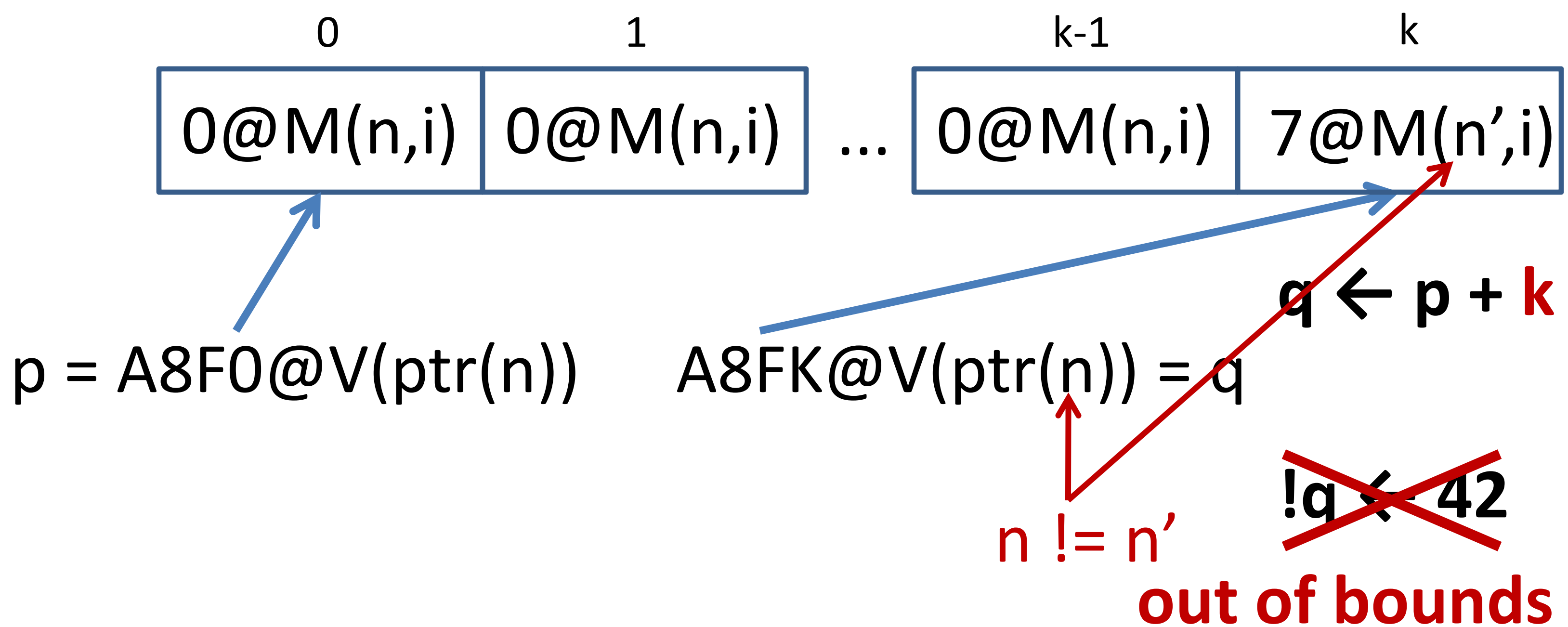
Micro-Policy for memory safety



Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety



Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

0	1		k-1	k
0@M(n,i)	0@M(n,i)	...	0@M(n,i)	7@M(n',i)

q ← p + k

p = A8F0@V(ptr(n))   A8FK@V(ptr(n)) = q

~~!q ← 42~~

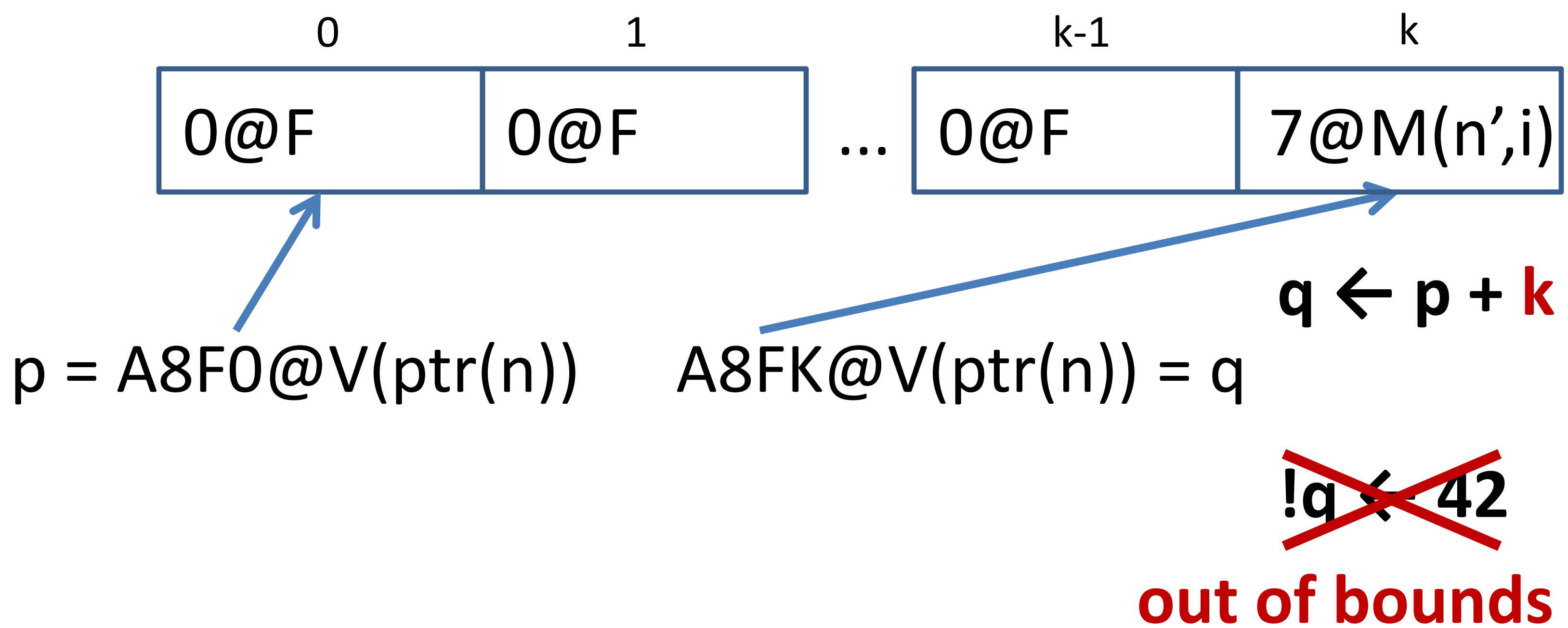
out of bounds

free p

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety

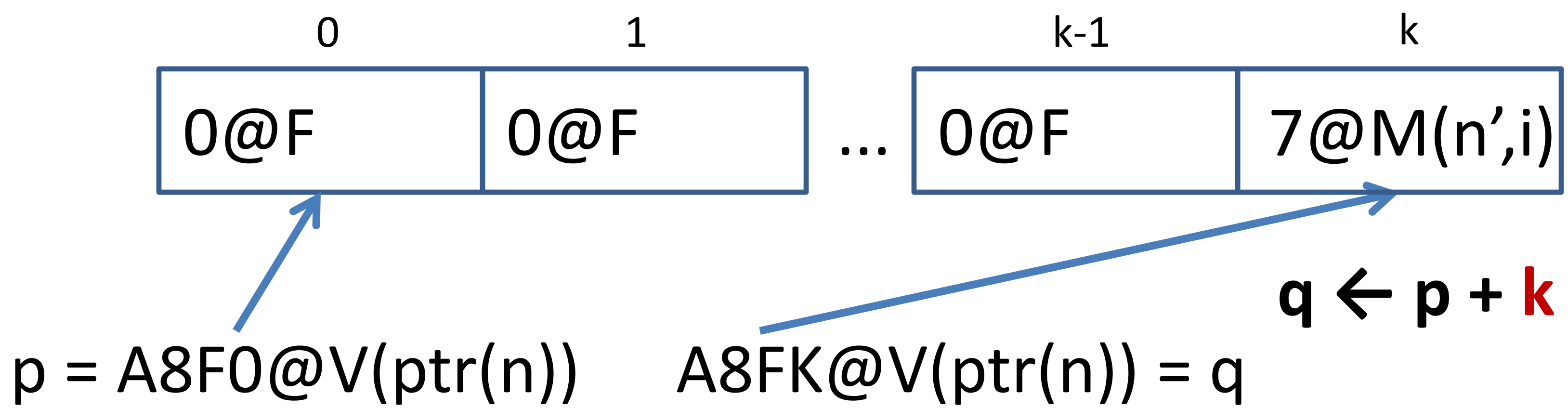


free p

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety



q ← p + k

~~!q ← 42~~

out of bounds

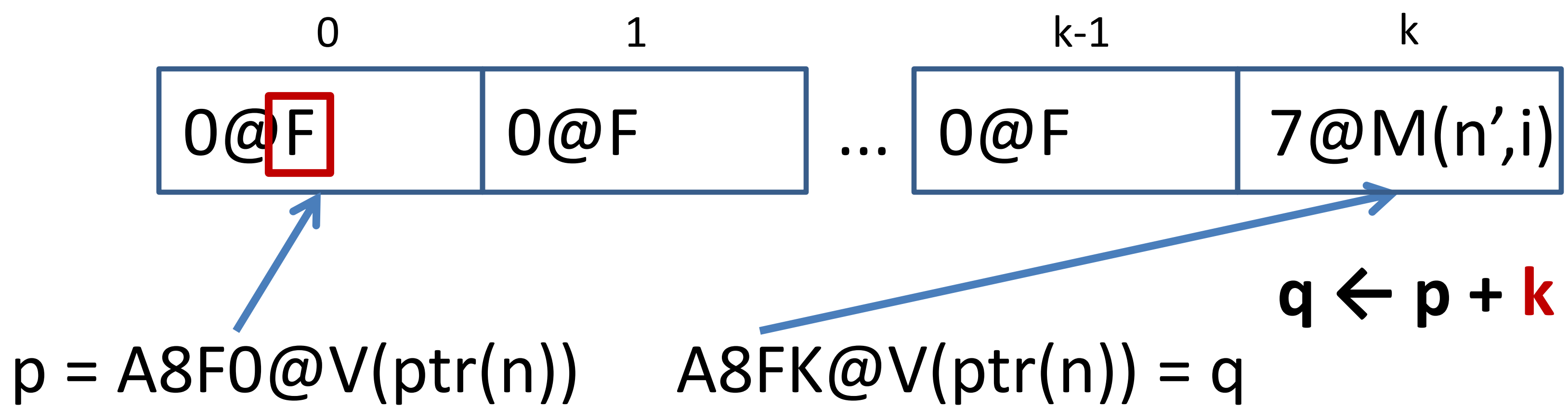
free p

x ← !p

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety



~~**!q ← 42**~~

out of bounds

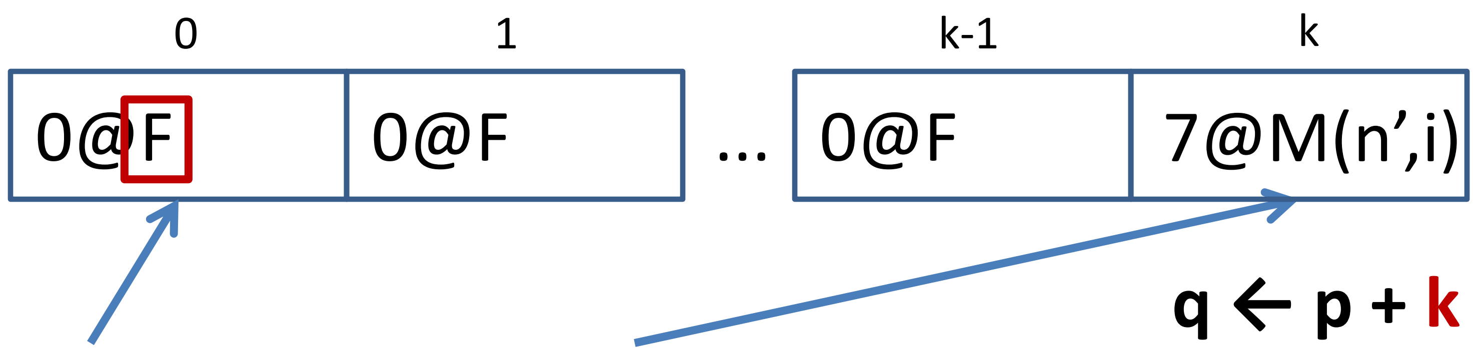
free p

x ← !p

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Micro-Policy for memory safety



q ← p + k

p = A8F0@V(ptr(n))

A8FK@V(ptr(n)) = q

~~!q ← 42~~

out of bounds

free p

~~x ← !p~~

use after free

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

Direction of this project

- **Beyond IFC**:
  – show generality: study diverse set of micro-policies
  – formally verify enforced properties
  – implement and evaluate practical viability
- **Beyond clean-slate (CRASH/SAFE)**:
  – targeting a stock RISC architecture
  – extended with tags and a rule cache
  – legacy software with little or no changes

Future challenges

- **Micro-policy composition**
  - hardware supports compound tags
  - but policies are often not orthogonal
    (e.g. tags can leak information)
  - this is not just reference monitoring / safety properties
    - “micro-calls” into privileged code can inspect tags
    - policy violations are often recoverable
  - sequential (vertical) vs. parallel (cross product)
  - further improve efficiency
- **Meta-language for micro-policies**
  - beyond disparate DSLs

Collaborators on this project*

UPenn

Arthur Azevedo de Amorim**

Maxime Denes

Leonidas Lampropoulos

Benoit Montagu

Benjamin Pierce

Antal Spector-Zabusky

INRIA Paris

Nick Giannarakis**

Cătălin Hrițcu

Portland State

Nathan Collins

Andrew Tolmach

IRISA Rennes

Delphine Demange

David Pichardie

Harvard

Greg Morrisett

Randy Pollack

* Started part of DARPA CRASH/SAFE

** Soon interns at INRIA Paris

Collaborators on this project*

Formal side

UPenn

- Arthur Azevedo de Amorim**
- Maxime Denes
- Leonidas Lampropoulos
- Benoit Montagu
- Benjamin Pierce
- Antal Spector-Zabusky

INRIA Paris

- Nick Giannarakis**
- Cătălin Hrițcu

Portland State

- Nathan Collins
- Andrew Tolmach

IRISA Rennes

- Delphine Demange
- David Pichardie

Harvard

- Greg Morrisett
- Randy Pollack

Architecture side

UPenn

- Andre DeHon
- Udit Dhawan
- Ben Karel
- Nikos Vasilakis
- Jonathan M. Smith

MIT

- Tom Knight
- Howard Shrobe

BAE Systems

- Greg Sullivan

...

* Started part of DARPA CRASH/SAFE

** Soon interns at INRIA Paris

My other two current projects

- QuickChick: Speeding up Formal Proofs with Property-Based Testing
  - General Framework for Polarized Mutation Testing
  - Language for Custom Test-Data Generators
  - Deep Integration with Coq/SSReflect
- νF*: Next Generation Security Type Checker
  - Better refinement type inference (Dijkstra monad)
  - Beyond value-dependency
  - Better control of effects (including termination)
  - Smarter (semantic) termination checking

THANK YOU

BACKUP SLIDES

Computer systems are insecure

Computer systems are insecure

Why?

Computer systems are insecure

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	65.5	CWE-732	Incorrect Permission Assignment for Critical Resource
[18]	64.6	CWE-676	Use of Potentially Dangerous Function
[19]	64.1	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[20]	62.4	CWE-131	Incorrect Calculation of Buffer Size
[21]	61.5	CWE-307	Improper Restriction of Excessive Authentication Attempts
[22]	61.1	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
[23]	61.0	CWE-134	Uncontrolled Format String
[24]	60.3	CWE-190	Integer Overflow or Wraparound
[25]	59.9	CWE-759	Use of a One-Way Hash without a Salt

CWE & SANS Institute TOP 25 MOST DANGEROUS SOFTWARE ERRORS

Source: 2011 CWE/SANS Top 25 Most Dangerous Software Errors

Micro-Policy for memory safety

- Tag := Val(Type) | Mem(n,Type) | Free   Type := Int | Ptr(n)
- allocation:
  - generate fresh n
  - initialize region with 0@Mem(n,Int)
  - return <pointer-to-region>@Val(Ptr(n))
- memory access (read/write):
  - check that pointer tagged @Val(Ptr(n))
  - check that referenced location tagged @Mem(n,Type)
  - on memory read tag result with @Val(Type)
  - when writing w@Val(NType) retag location with @Mem(n,Type)
- reclaiming memory (free):
  - check that pointer and referenced location have the same n
  - overwrite region with 0@Free

Formal verification side

- Verification of low-level code
  - bisimulation/refinement
  - verified structured code generators