Micro-Policies

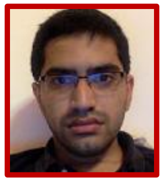
Formally Verified,
Tag-Based Security Monitors

Cătălin Hrițcu

Inria Paris-Rocquencourt, Prosecco team

Micro-Policies collaborators

- **Formal methods** & **hardware architecture**
- **Current team**:
  - *UPenn*: **Arthur Azevedo de Amorim**, **André DeHon**, **Benjamin Pierce**, **Antal Spector-Zabusky**, **Udit Dhawan**
  - *Inria Prosecco*: **Cătălin Hrițcu**, **Yannis Juglaret** (soon DGA-Inria PhD)
  - *Paris Sud (Digiteo) & Portland State*: **Andrew Tolmach**



- **Past: DARPA CRASH/SAFE project**
  - *France*: **Delphine Demange** (*IRISA Celtique*), **Maxime Dénès** (*Inria Gallium*), **Nick Giannarakis** (*Inria Prosecco*), **David Pichardie** (*IRISA Celtique*)

Computer systems are insecure

Computer systems are insecure

- **Today's computers 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
  - bad security-performance tradeoff
  - just write secure code ... all of it!
- **Consequence: vulnerabilities in every system!**
  - **violations of well-studied safety and security policies**

Micro-policies

- add **large tag** to each machine word **unbounded metadata**

$$\boxed{\textbf{word}}\boxed{\textbf{tag}} \longrightarrow \boxed{\textbf{tag[0]}}\boxed{\textbf{tag[1]}}\boxed{\textbf{tag[2]}}$$

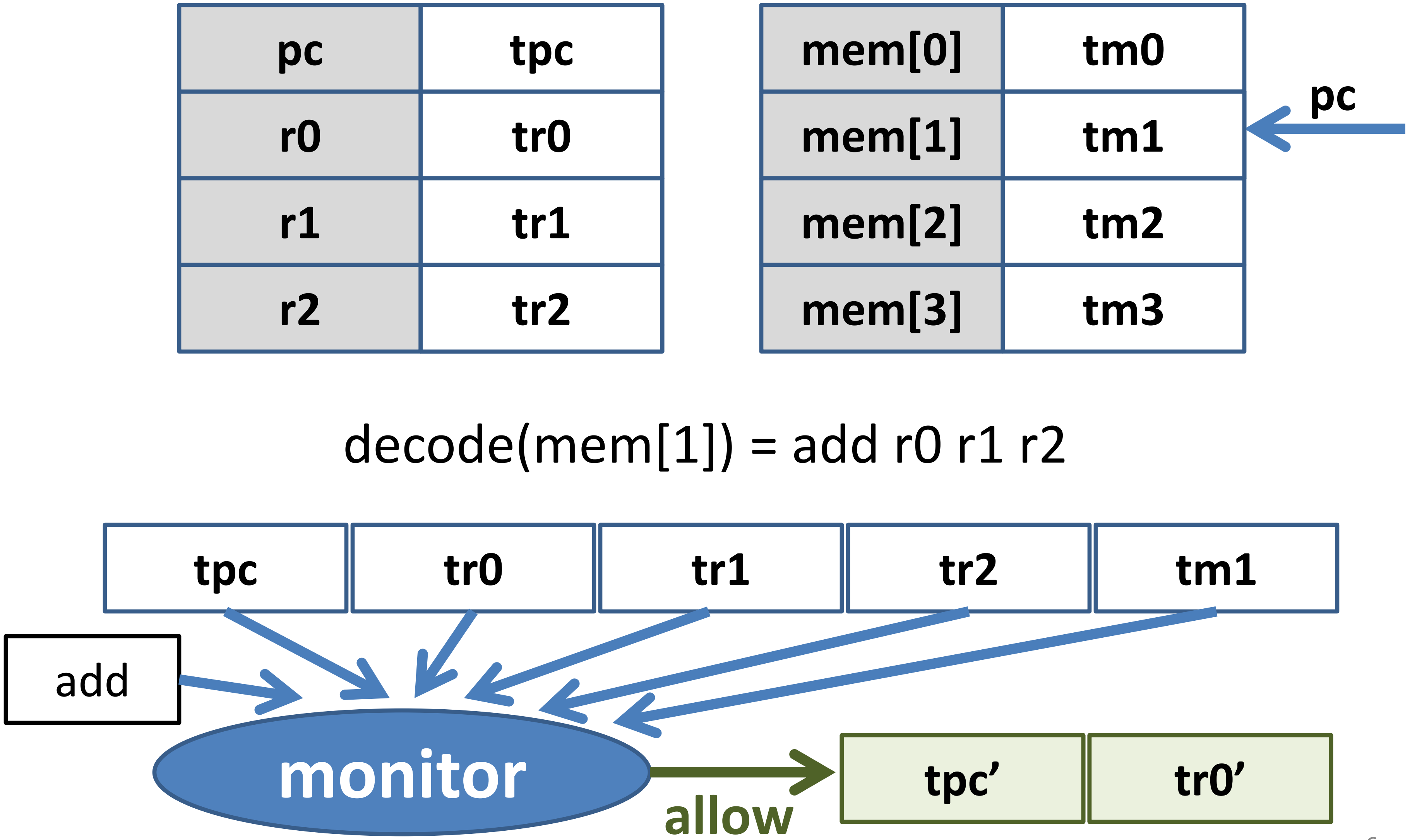
- words in memory and registers are all tagged

pc	**tag**
r0	**tag**
r1	**tag**
r2	**tag**

mem[0]	**tag**
mem[1]	**tag**
mem[2]	**tag**
mem[3]	**tag**

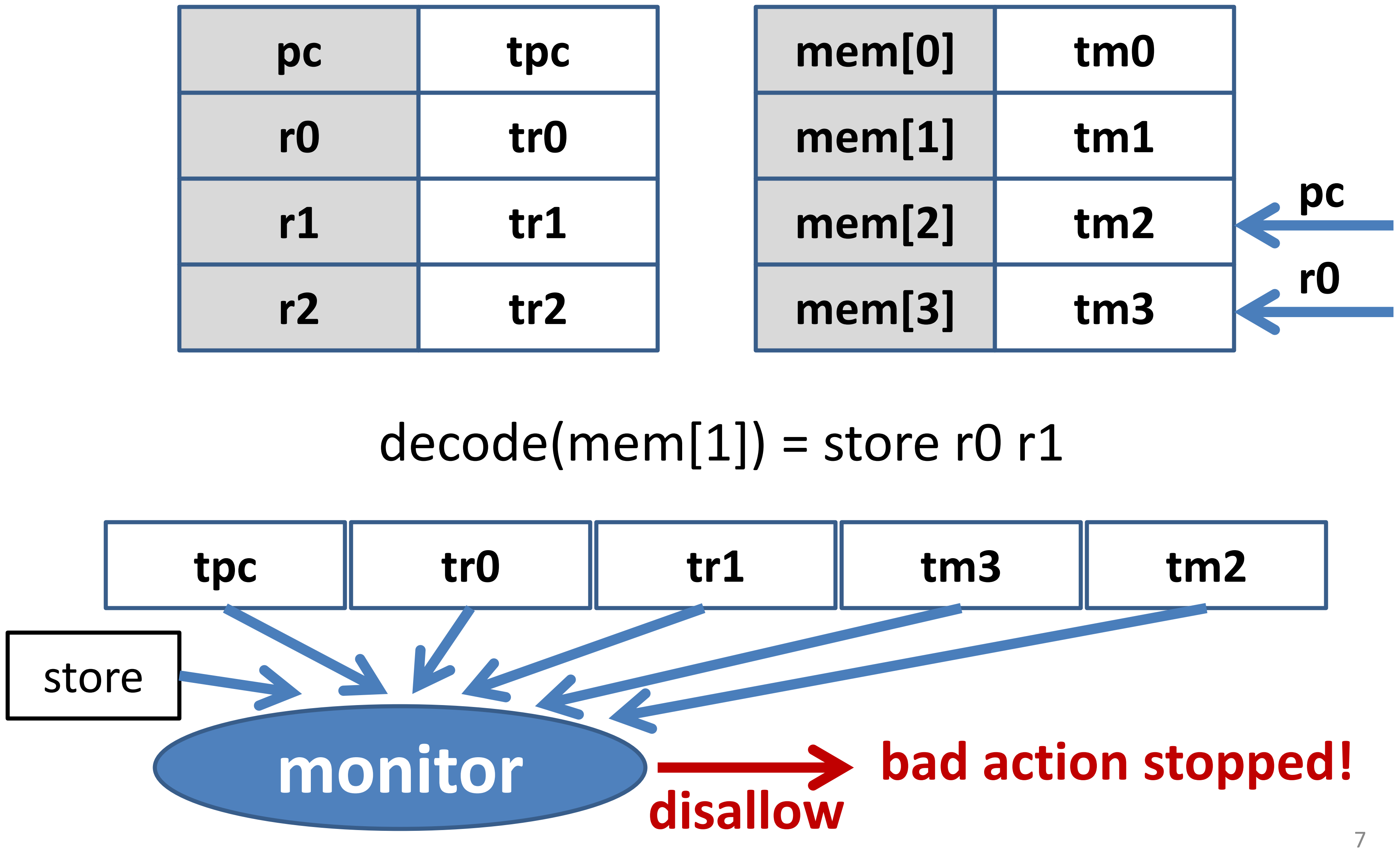
*Conceptual model, the hardware implements this efficiently (more later)

Tag-based instruction-level monitoring



decode(mem[1]) = add r0 r1 r2

Tag-based instruction-level monitoring



Micro-policies are cool!

- **low level + fine grained**: unbounded per-word metadata, checked & propagated on each instruction
- **expressive**: can enforce large number of policies
- **flexible**: tags and monitor defined by software
- **efficient**: accelerated using hardware caching
- **secure**: formally verified to provide security

Expressiveness

- Micro-policy mechanism can efficiently enforce:
  - memory safety
  - code-data separation
  - control-flow integrity
  - compartment isolation
  - taint tracking
  - information flow control
  - monitor self-protection
  - dynamic sealing

and a lot more!

History:

• SAFE machine had separate HW mechanisms for many of these

• micro-policies were only used for IFC [Oakland'13, POPL'14]

•... we only realized later how expressive they are [ASPLOS'15, Oakland'15]

Flexibility by example: **memory safety**

- Our memory safety micro-policy prevents
  - **spatial violations**: reading/writing out of bounds
  - **temporal violations**: use after free, invalid free
  - for **heap-allocated data** (for simplicity)

- Pointers become **unforgeable capabilities**

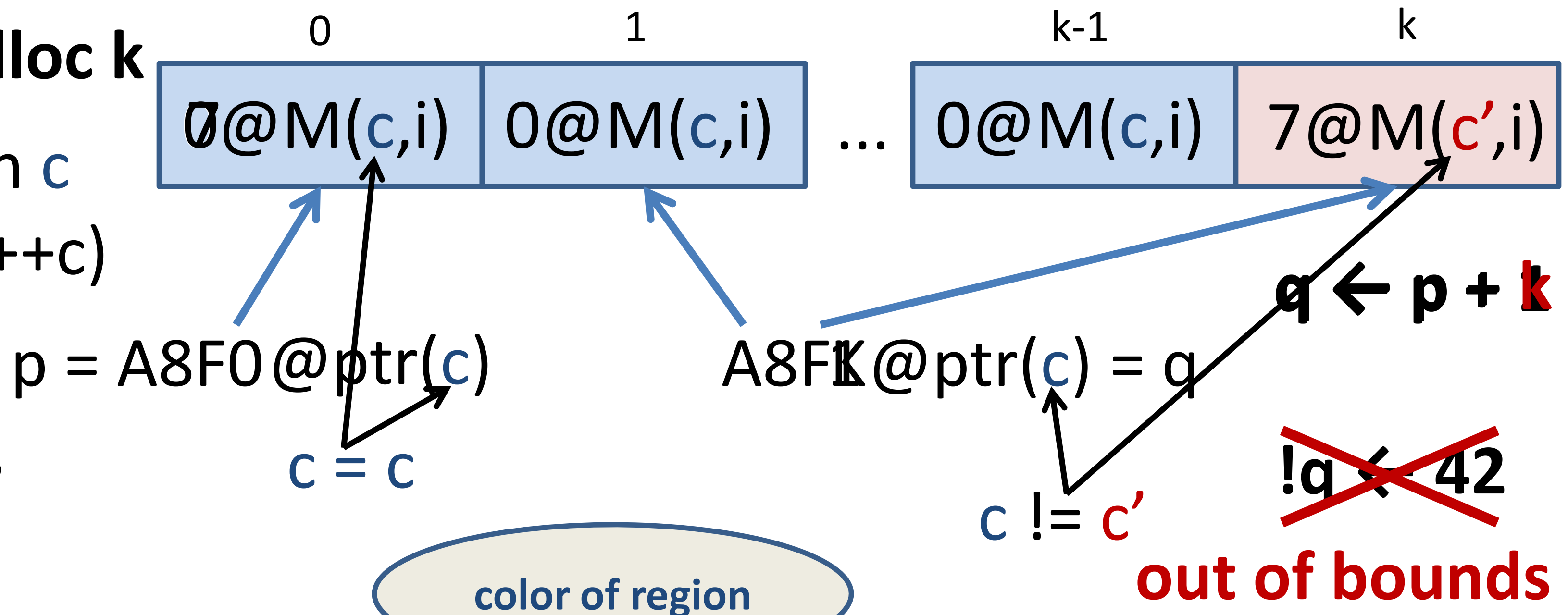
  - can only obtain a valid pointer to a memory region
    - by allocating that region or
    - by copying/offsetting an existing pointer to that region

Memory safety micro-policy

p←malloc k

fresh c

(e.g. ++c)



!p ← 7

free p

color of region

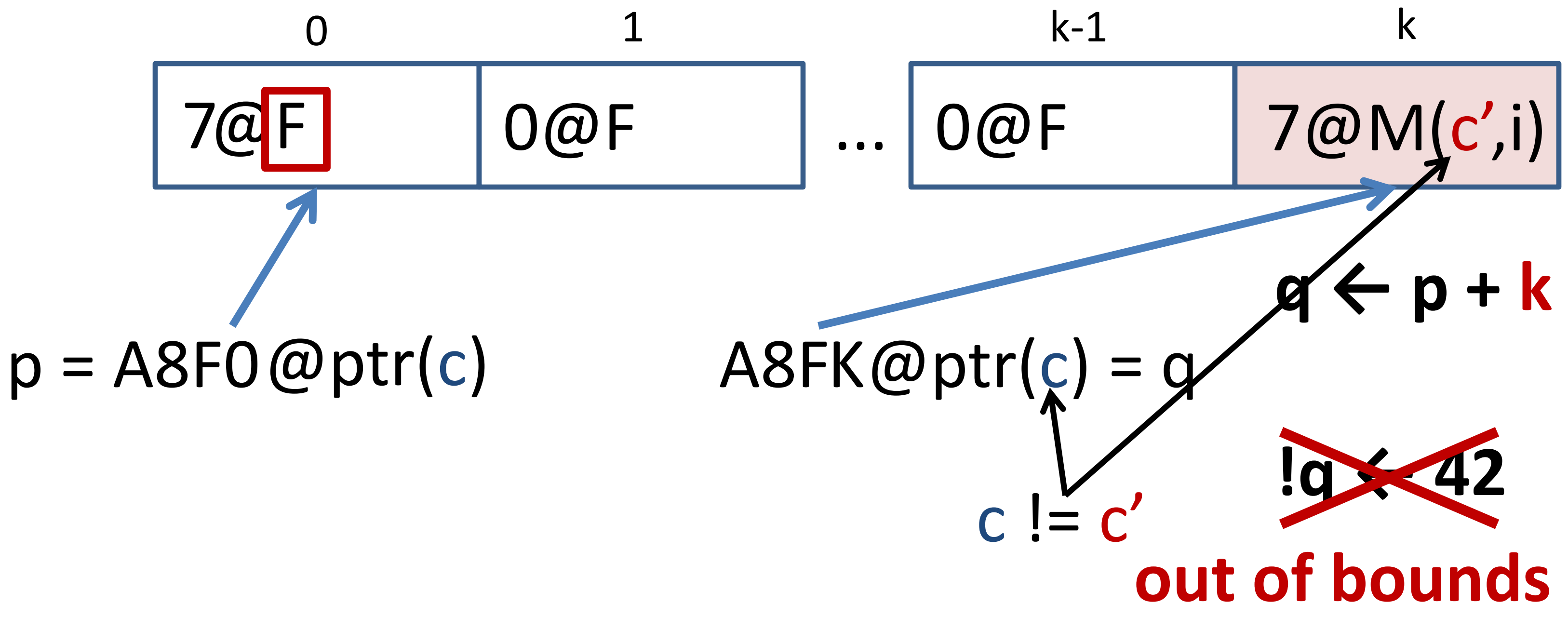
T_v ::= i | ptr(c)   tags on values

T_m ::= M(c,T_v) | F   tags on memory

color of region

tag of content

Memory safety micro-policy



free p

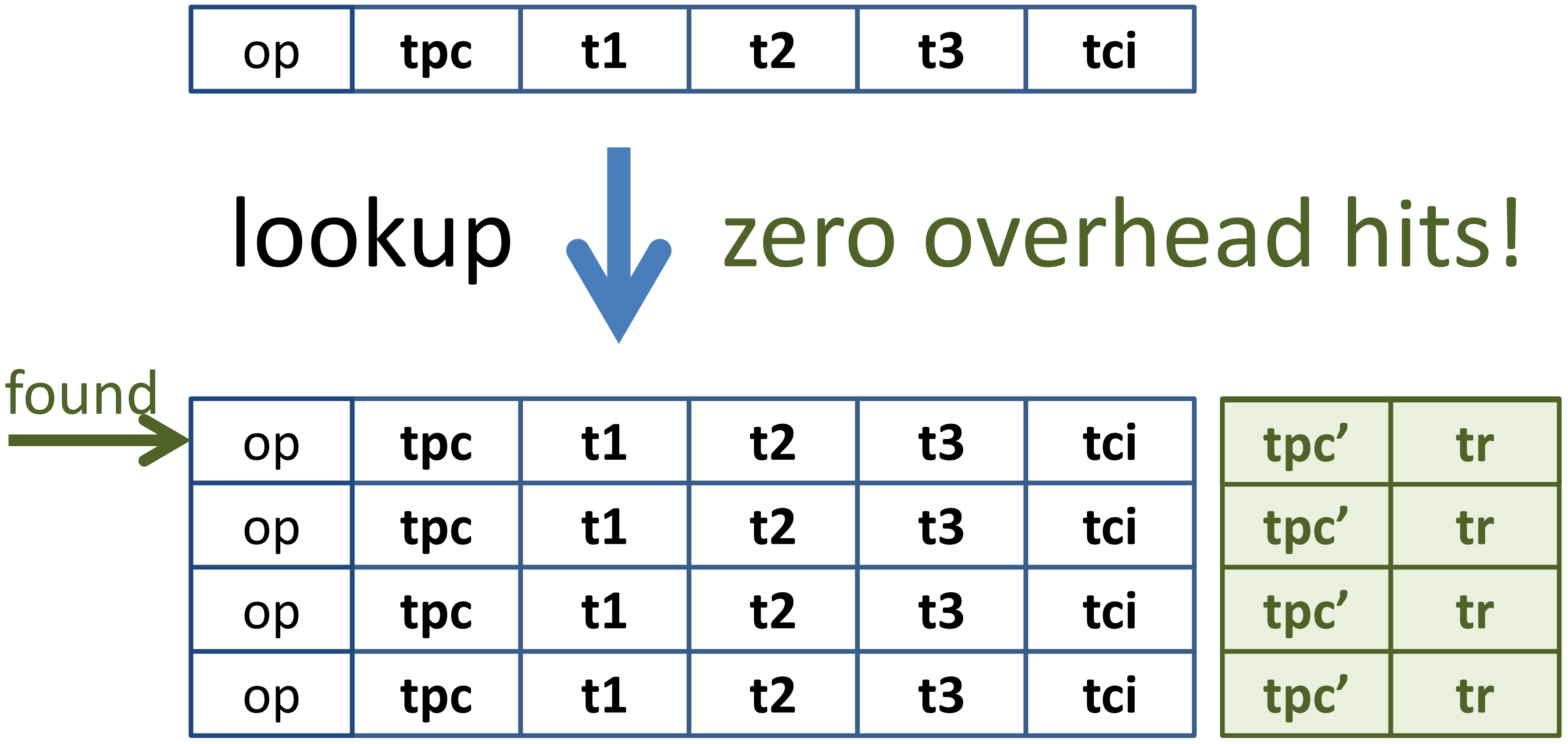
~~**x ← !p**~~

use after free

T_v ::= i | ptr(c) tags on values

T_m ::= M(c,T_v) | F tags on memory

Efficiently executing micro-policies



op	**tpc**	**t1**	**t2**	**t3**	**tci**

op	**tpc**	**t1**	**t2**	**t3**	**tci**	**tpc’**	**tr**
op	**tpc**	**t1**	**t2**	**t3**	**tci**	**tpc’**	**tr**
op	**tpc**	**t1**	**t2**	**t3**	**tci**	**tpc’**	**tr**
op	**tpc**	**t1**	**t2**	**t3**	**tci**	**tpc’**	**tr**

hardware cache

Efficiently executing micro-policies

op	**tpc**	**t1**	**t2**	**t3**	**tci**	**tpc’**	**tr**

lookup

misses trap to software

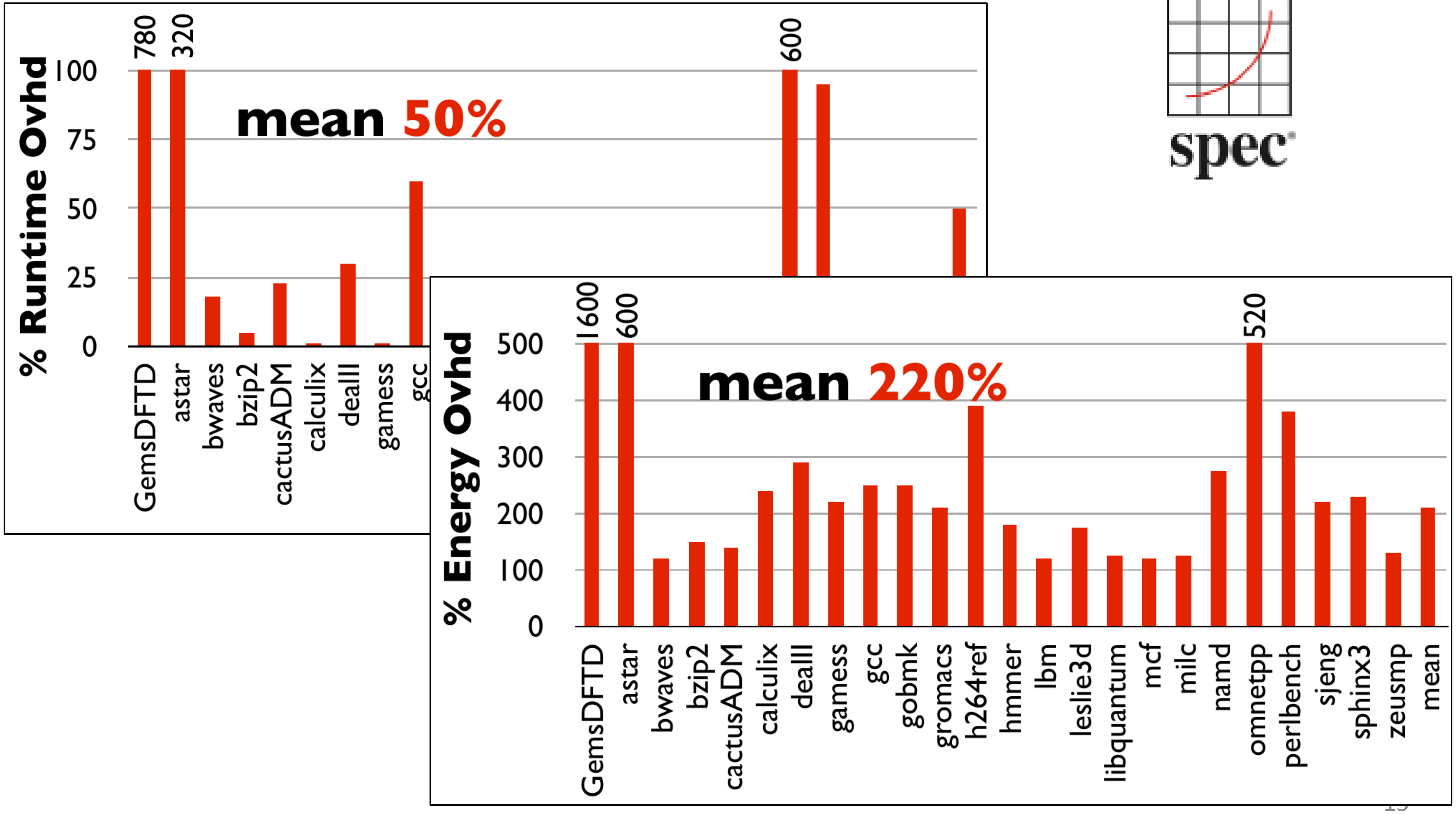
produced “rule” cached

op	**tpc**	**t1**	**t2**	**t3**	**tci**	**tpc’**	**tr**
op	**tpc**	**t1**	**t2**	**t3**	**tci**	**tpc’**	**tr**
op	**tpc**	**t1**	**t2**	**t3**	**tci**	**tpc’**	**tr**
op	**tpc**	**t1**	**t2**	**t3**	**tci**	**tpc’**	**tr**

hardware cache

Simulations for **naive** implementation

memory safety + code-data separation + taint tracking + control-flow integrity
simple RISC processor: single-core 5-stage in-order Alpha



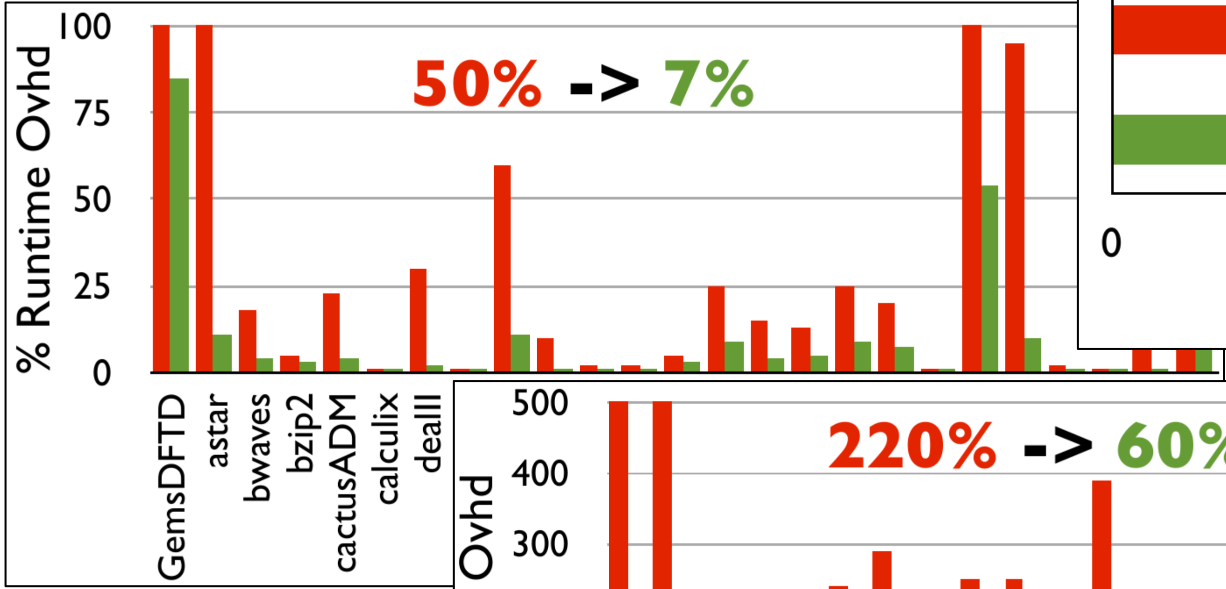
Targeted [micro-]architectural optimizations

[ASPLOS'15]

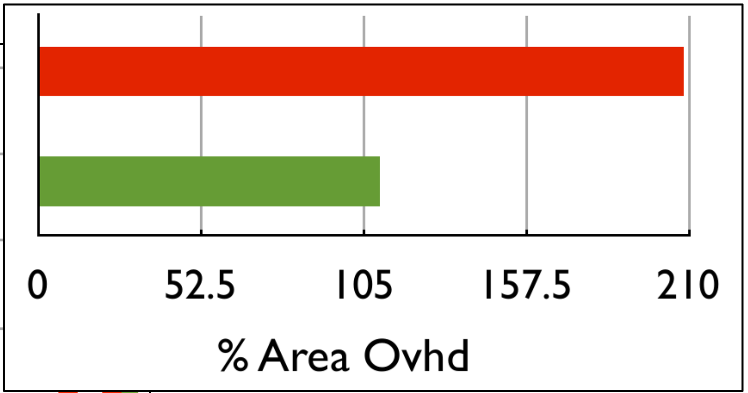
- grouping opcodes and ignoring unused tags
  - **increases effective rule cache capacity**
- transferring only unique tags to/from DRAM
  - **reduces runtime and energy overhead**
- using much shorter tags for on-chip data caches
  - **reduces runtime, energy, and area overhead**
- caching composite policies separately
  - **makes rule cache misses much cheaper**

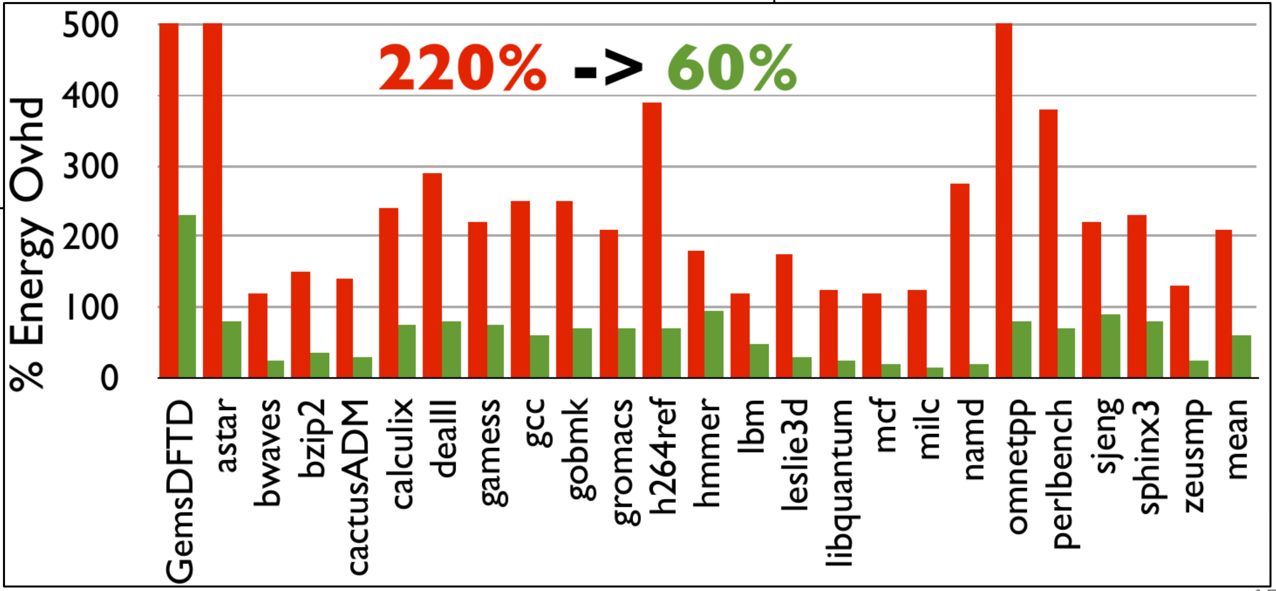
Simulations for **optimized** implementation

memory safety + code-data separation + taint tracking + control-flow integrity
simple RISC processor: single-core 5-stage in-order Alpha



no free lunch

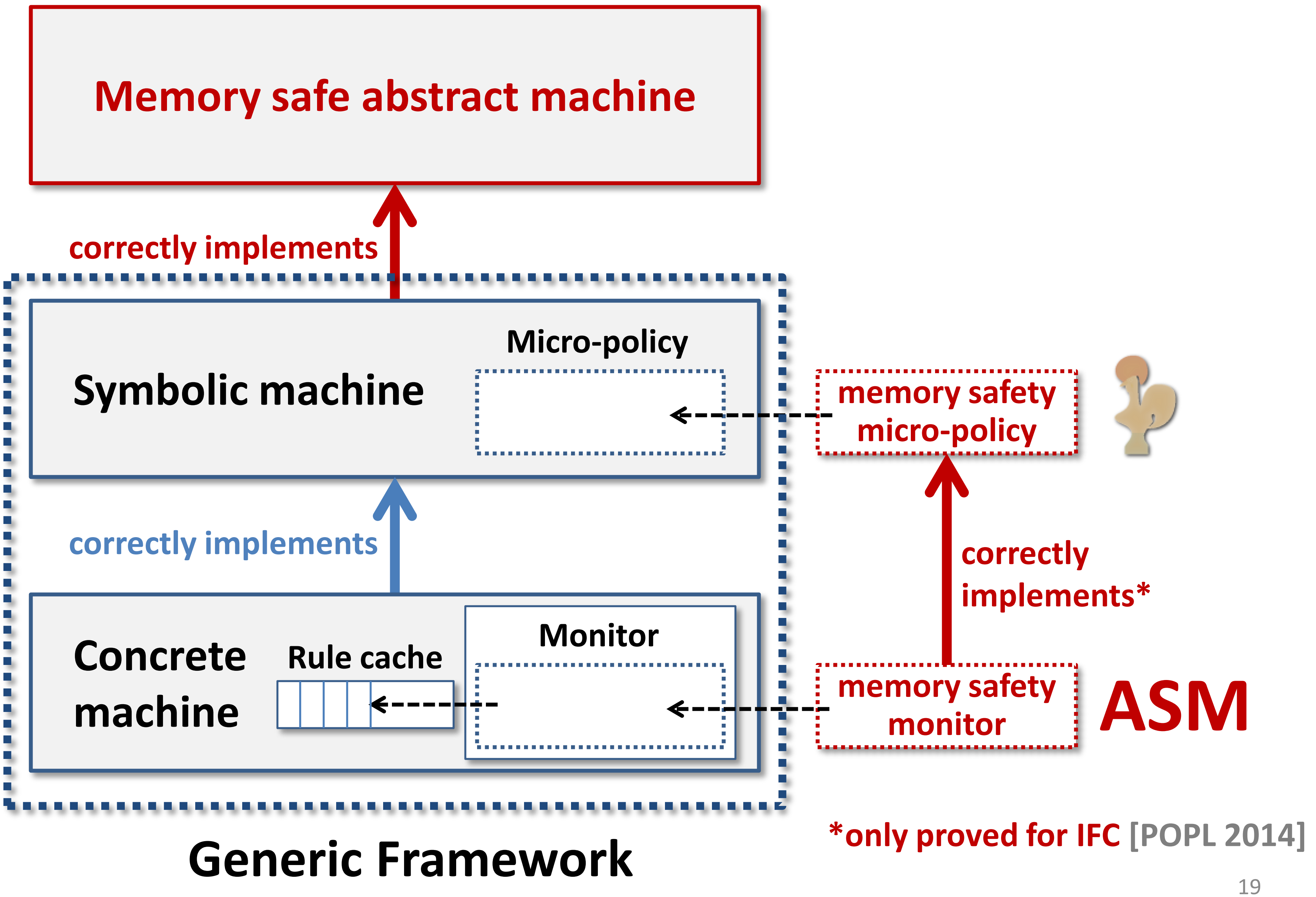




Is it secure?

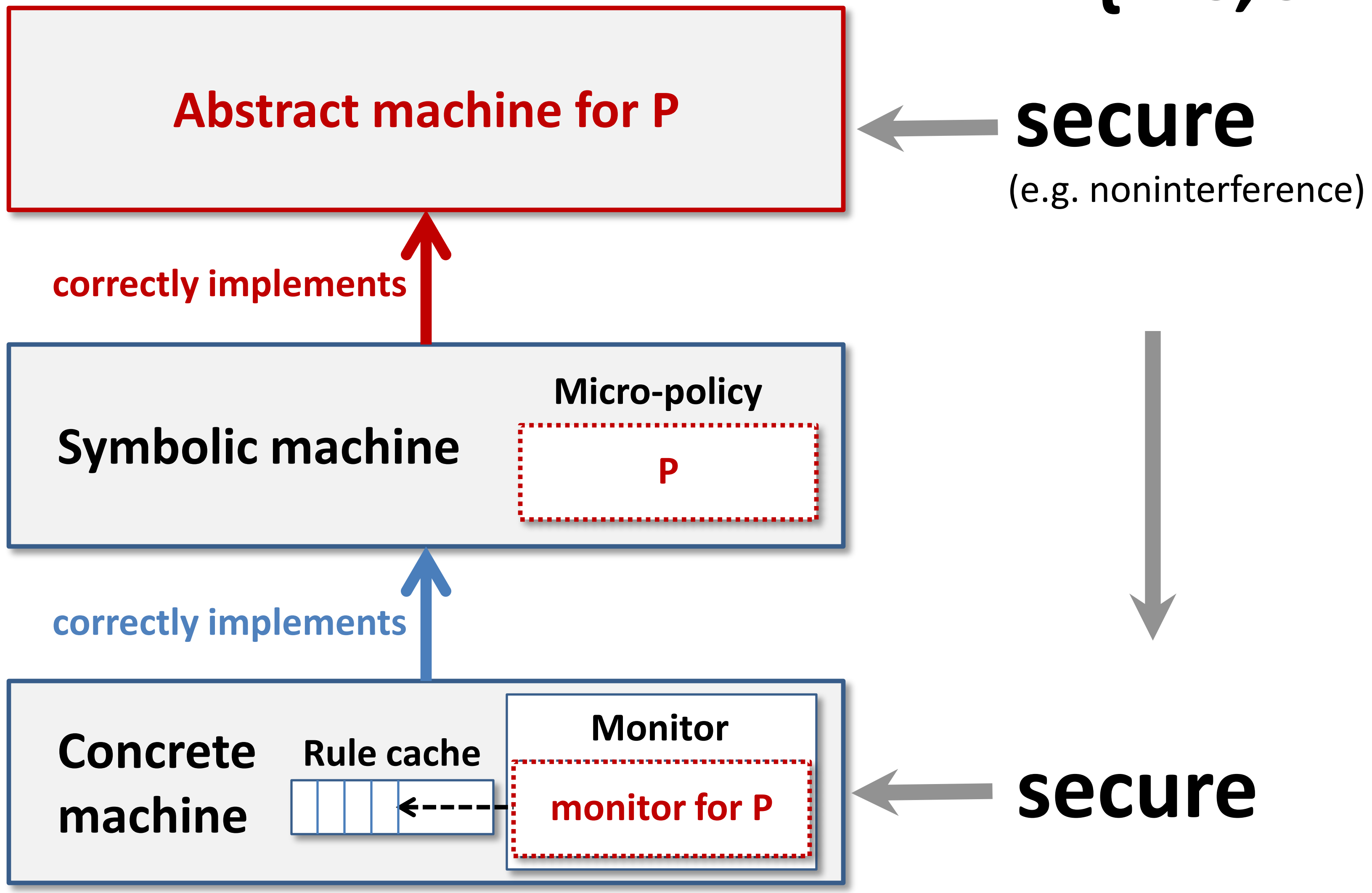
FORMAL VERIFICATION IN COQ

[POPL’14, Oakland’15]



*only proved for IFC [POPL 2014]

P in {IFC,CFI}


Abstract machine for P
secure
(e.g. noninterference)
correctly implements
Micro-policy
Symbolic machine
P
correctly implements
Concrete machine
Rule cache
Monitor
monitor for P
secure

Memory safety micro-policy

1. Sets of tags

$\mathbf{T_v}$ ::= i | ptr(c)

$\mathbf{T_m}$ ::= M(c,T_v) | F

$\mathbf{T_{pc}}$ ::= T_v

2. Transfer function

Record **IVec** := { op:opcode ; t_{pc}:T_{pc} ; t_i:T_m ; ts: ... }

Record **OVec** (op:opcode) := { t_{rpc} : T_{pc} ; t_r : ... }

transfer : (iv:IVec) -> option (OVec (op iv))

Definition **transfer** iv :=

match iv with

| {op=Load; t_{pc}=ptr(c_{pc}); t_i=M(c_{pc},i); ts=[ptr(**c**); M(**c**,$\mathbf{T_v}$)]}

   => {t_{rpc}=ptr(c_{pc}); t_r=$\mathbf{T_v}$}

| {op=Store; t_{pc}=ptr(c_{pc}); t_i=M(c_{pc},i); ts=[ptr(**c**); $\mathbf{T_v}$; M(**c**,T_v')]}

   => {t_{rpc}=ptr(c_{pc}); t_r=M(**c**,$\mathbf{T_v}$)}

...

Memory safety micro-policy

1. Sets of tags

T_v ::= i | ptr(c)

T_m ::= M(c,T_v) | F

T_{pc} ::= T_v

2. Transfer function

Record **IVec** := { op:opcode ; t_{pc}:T_{pc} ; t_i:T_m ; ts: ... }

Record **OVec** (op:opcode) := { t_{rpc} : T_{pc} ; t_r : ... }

transfer : (iv:IVec) -> option (OVec (op iv))

3. Monitor services

Record **service** := { addr : word; sem : state -> option state; ... }

Definition **mem_safety_services** : list service :=

[**malloc; free; base; size; eq**].

Open problems

- **Interaction with compiler, loader, linker, OS**
- **Secure micro-policy composition**
- **Reduce energy + more adaptive usage**
- **Modern RISC** instruction set (e.g. ARM)
- **More realistic processor**
  (our-of-order execution, even multi-core)

Fully abstract compilation

- Golden standard for **secure compilation**
  - P ≈ Q ↔ compile(P) ≈ compile(Q)
  - P ≈ Q = ∀C. C[P] has the same behavior as C[Q]
  - ***intuition:*** low-level machine code contexts can't do more harm than high-level contexts
  - can securely link compiled and untrusted machine code
- **Very strong, but rarely achieved in practice**
  - much stronger than compiler correctness
  - need a compiler & runtime that actually enforce high-level abstractions at the low level
  - **... and that's currently too expensive!**

Targeting micro-policy machine

- Micro-policies can **efficiently protect abstractions**
- **Fully abstract compiler** to **micro-policy machine**
  - Recently started with **Yannis Juglaret**
  - Toy source language: Featherweight Java subset
  - FJ classes protected from native classes they link with
  - Micro-policy combining: protects:

Micro-policy combining:	protects:
• compartment isolation	classes
• linear return capabilities	stack discipline
• dynamic typing	type safety

- **Long term goal**: functional programming language

Take away

- **Micro-policies**, novel security mechanism
  - **low level, fine grained, expressive, flexible, efficient, formally secure**
- **cool research direction** with **many interesting open problems** for us **and others** to solve

BACKUP SLIDES

Other highlights in **Prosecco** team

- **Pro**gramming **sec**urely with **c**rypt**o**graphy
- **Proverif** and **Cryptoverif** protocol analyzers
- **miTLS**: verified reference implementation
- **F***: program verification system for OCaml/F#
- **QuickChick**: property-based testing for Coq
- Permanent researchers:
  - Karthikeyan Bhargavan, Bruno Blanchet, Cătălin Hrițcu, Graham Steel

Cryptosense
Master your Cryptographic Infrastructure