



Micro-Policies

*Formally Verified,
Tag-Based Security Monitors*

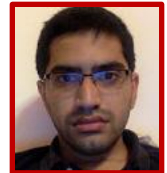
Cătălin Hrițcu

Inria Paris-Rocquencourt, Prosecco team

Inria

Micro-Policies collaborators

- **Formal methods & hardware architecture**
- **Current team:**
 - *UPenn*: Arthur Azevedo de Amorim, **André DeHon**, Benjamin Pierce, Antal Spector-Zabusky, **Udit Dhawan**
 - *Inria Paris*: **Cătălin Hrițcu**, Yannis **Juglaret**
 - *Portland State*: Andrew Tolmach
- **Recent past:**
 - **DARPA CRASH/SAFE** project (2011-2014)



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**



- 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 (a bit more later)

Tag-based instruction-level monitoring

pc	tpc
r0	tr0
r1	tr1
r2	tr2

mem[0]	tm0
mem[1]	tm1
mem[2]	tm2
mem[3]	tm3



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



add



allow



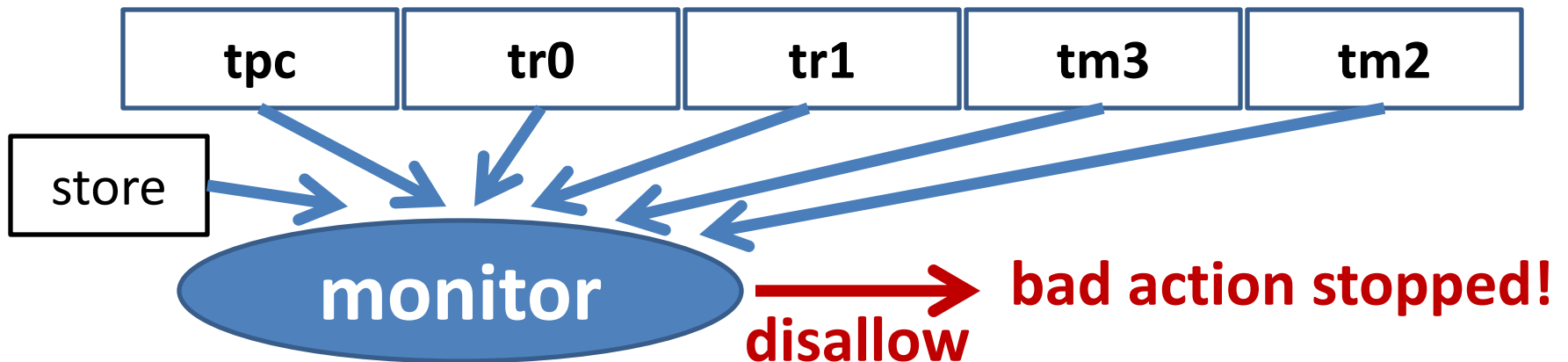
Tag-based instruction-level monitoring

pc	tpc
r0	tr0
r1	tr1
r2	tr2

mem[0]	tm0
mem[1]	tm1
mem[2]	tm2
mem[3]	tm3

← pc
← r0

decode(mem[1]) = store r0 r1



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!

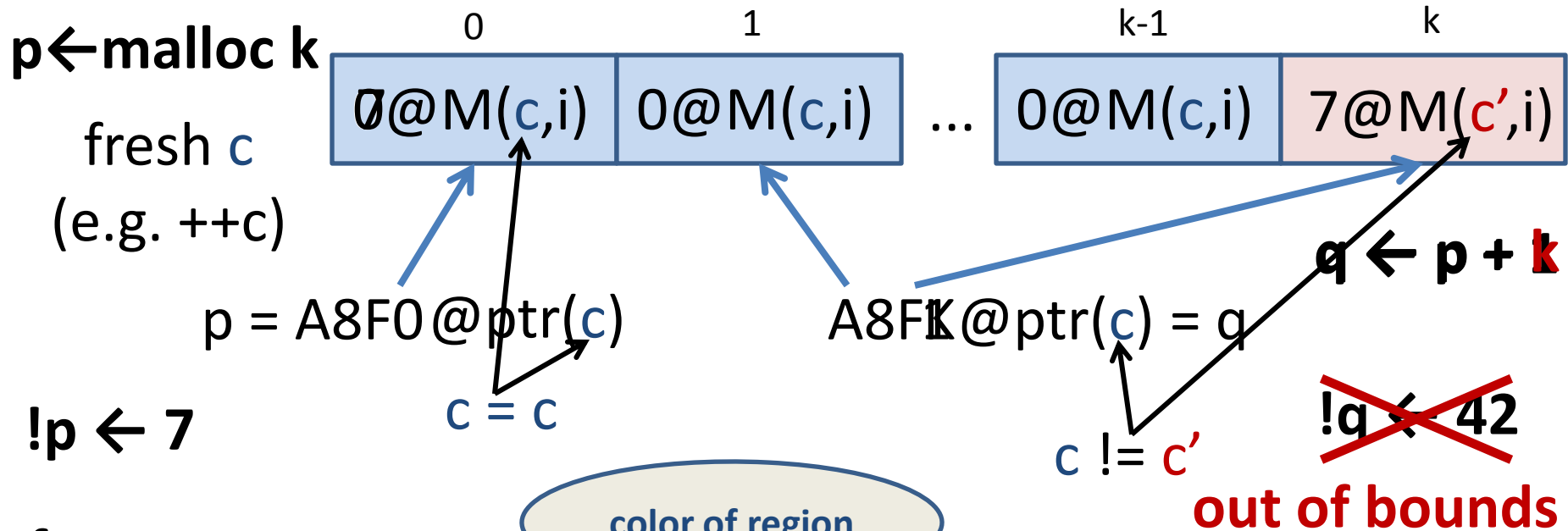
History:

- 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]

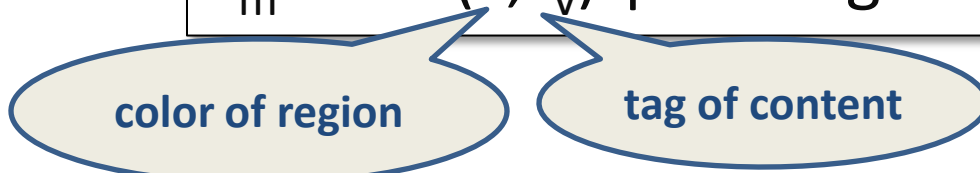
Flexibility by example: **memory safety**

- 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

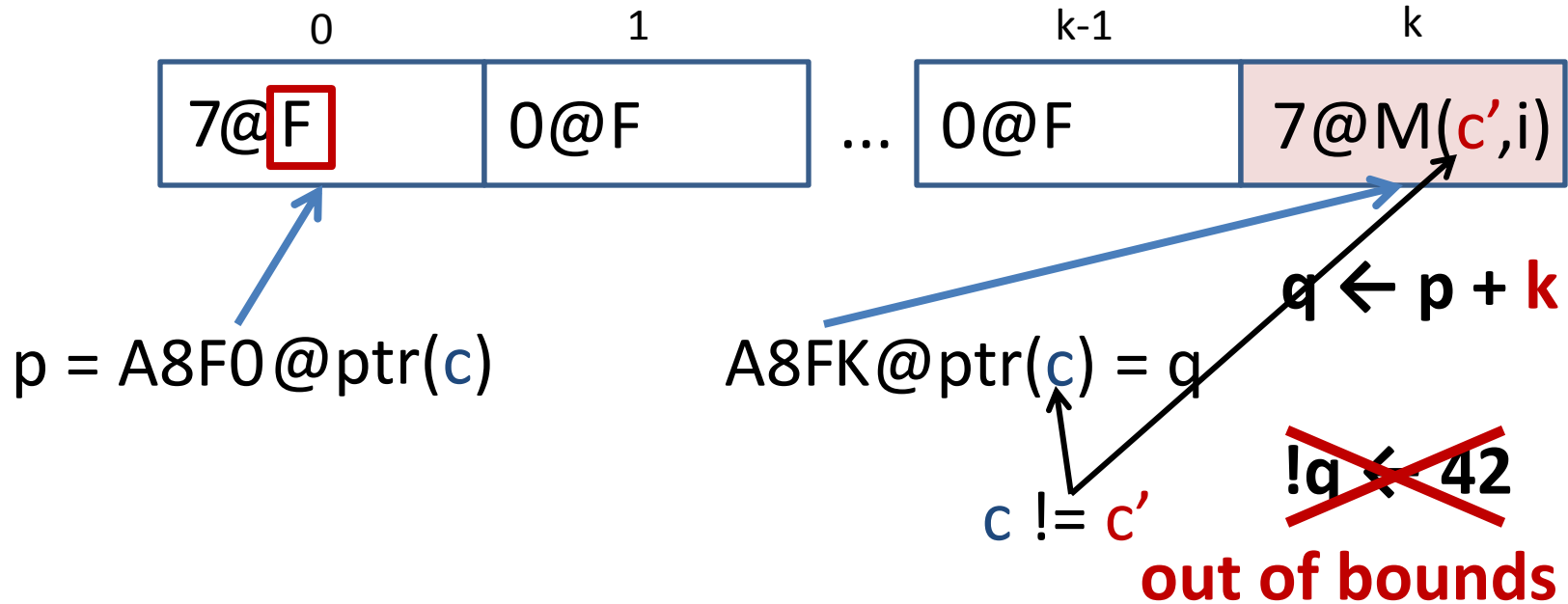
Memory safety micro-policy



$T_v ::= i \mid ptr(c)$	tags on values
$T_m ::= M(c, T_v) \mid F$	tags on memory



Memory safety micro-policy



free p

~~$x \leftarrow !p$~~

use after free

$T_v ::= i \mid \text{ptr}(c)$	tags on values
$T_m ::= M(c, T_v) \mid F$	tags on memory

Is it dead slow?

EFFICIENTLY EXECUTING MICRO-POLICIES

Efficiently executing micro-policies

op	tpc	t1	t2	t3	tci
----	-----	----	----	----	-----

lookup  zero overhead hits!

found 


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

tpc'	tr
tpc'	tr
tpc'	tr
tpc'	tr

hardware cache

Efficiently executing micro-policies

op	tpc	t1	t2	t3	tci	tpc'	tr
----	-----	----	----	----	-----	------	----

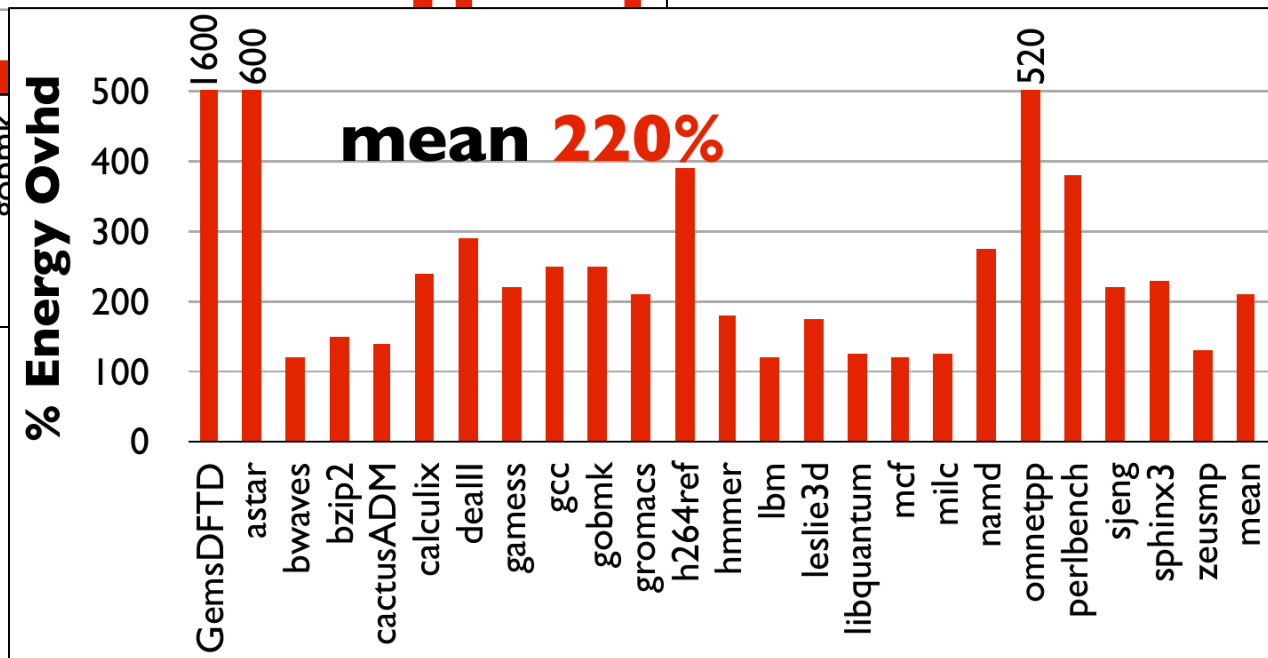
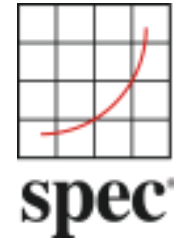
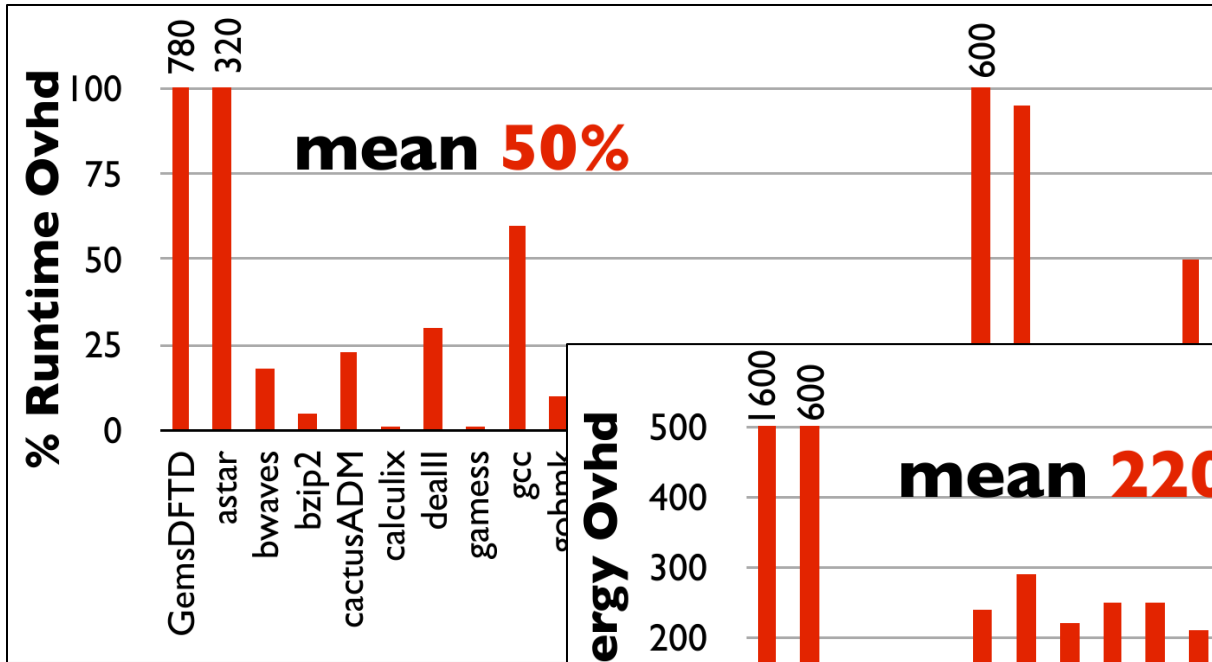
lookup  misses trap to software
produced "rule" cached

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



simulation numbers;
but this naive version also
implemented on FPGA
(part of SAFE machine)
[FPGA '13, TRETs '15]

Targeted [micro-]architectural optimizations

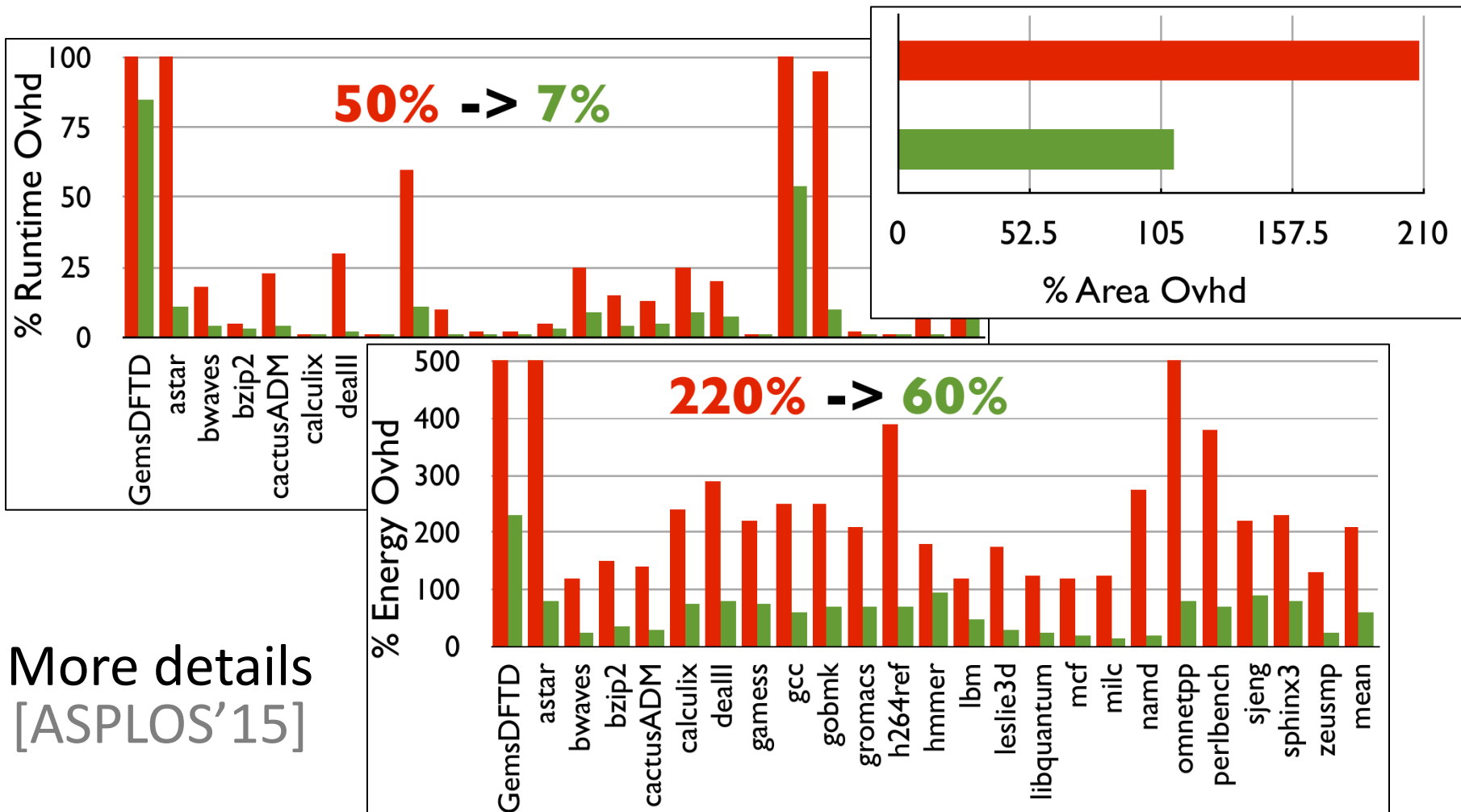
[ASPLOS'15]

- 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



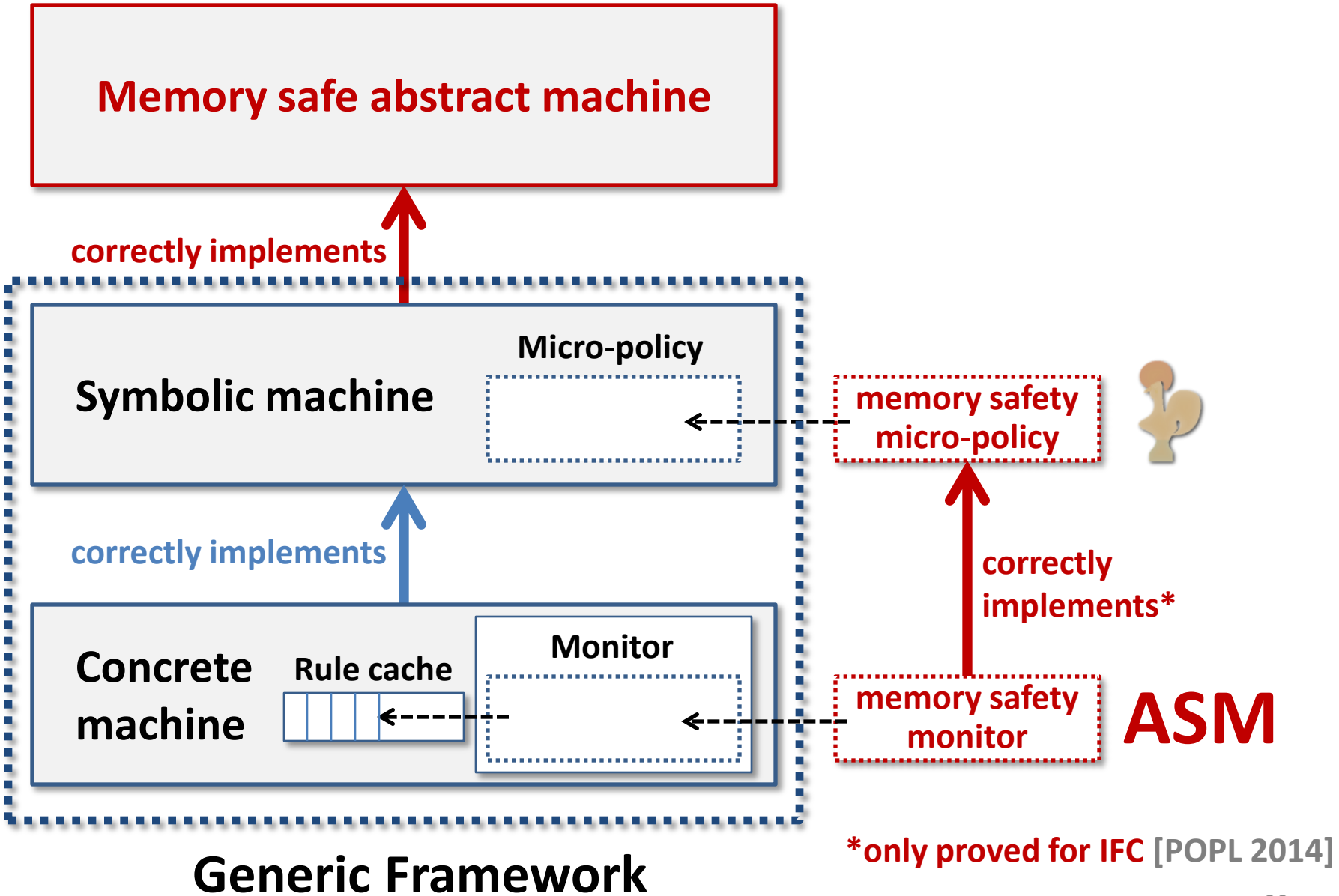
More details
 [ASPLOS'15]

Is it secure?

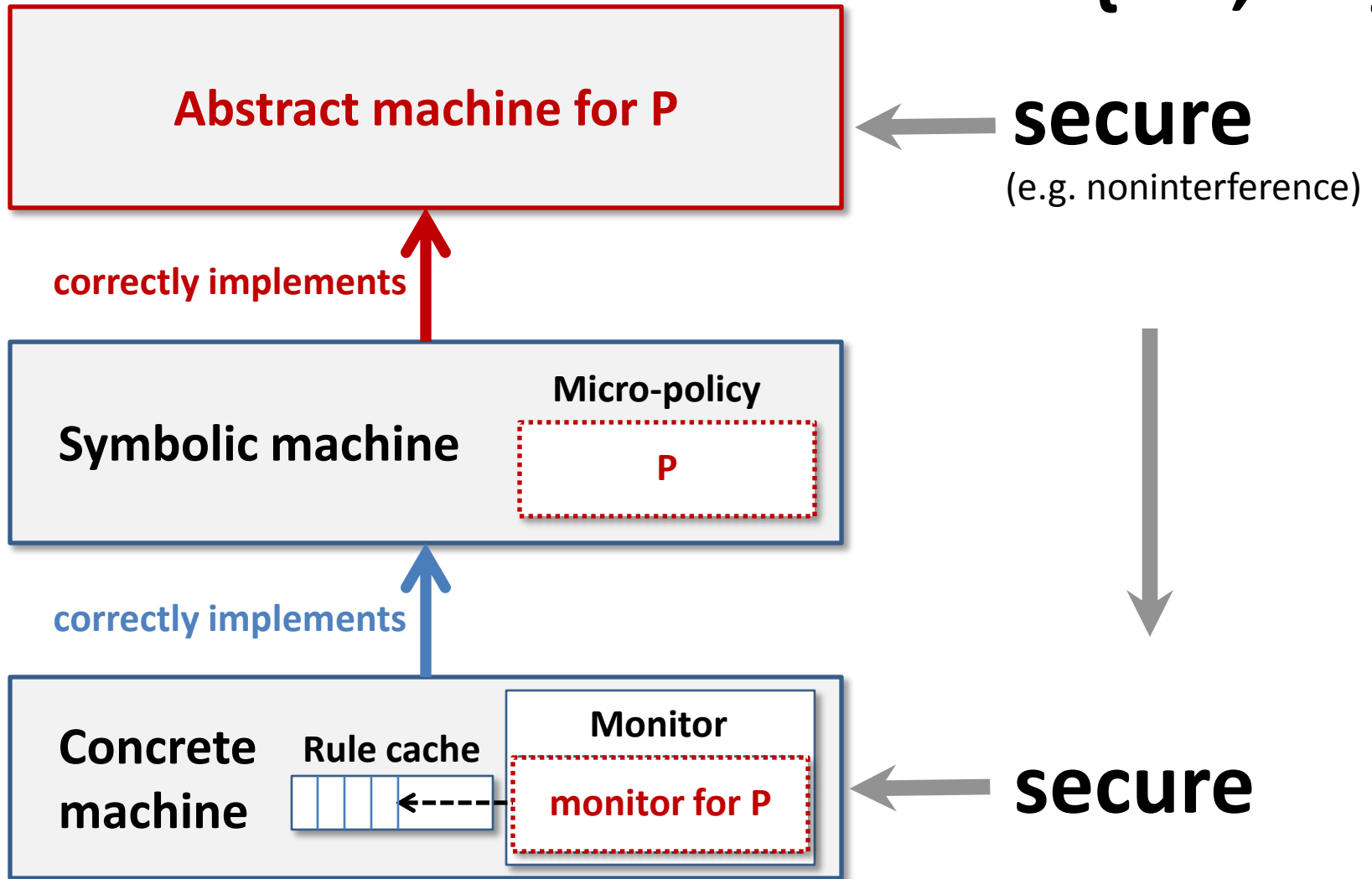
FORMAL VERIFICATION IN COQ



[POPL'14, Oakland'15]



P in {IFC,CFI}



* Currently working on extrinsic definition of memory safety [draft 2015]

Memory safety micro-policy



1. Sets of tags

$T_v ::= i \mid \text{ptr}(c)$

$T_m ::= M(c, T_v) \mid F$

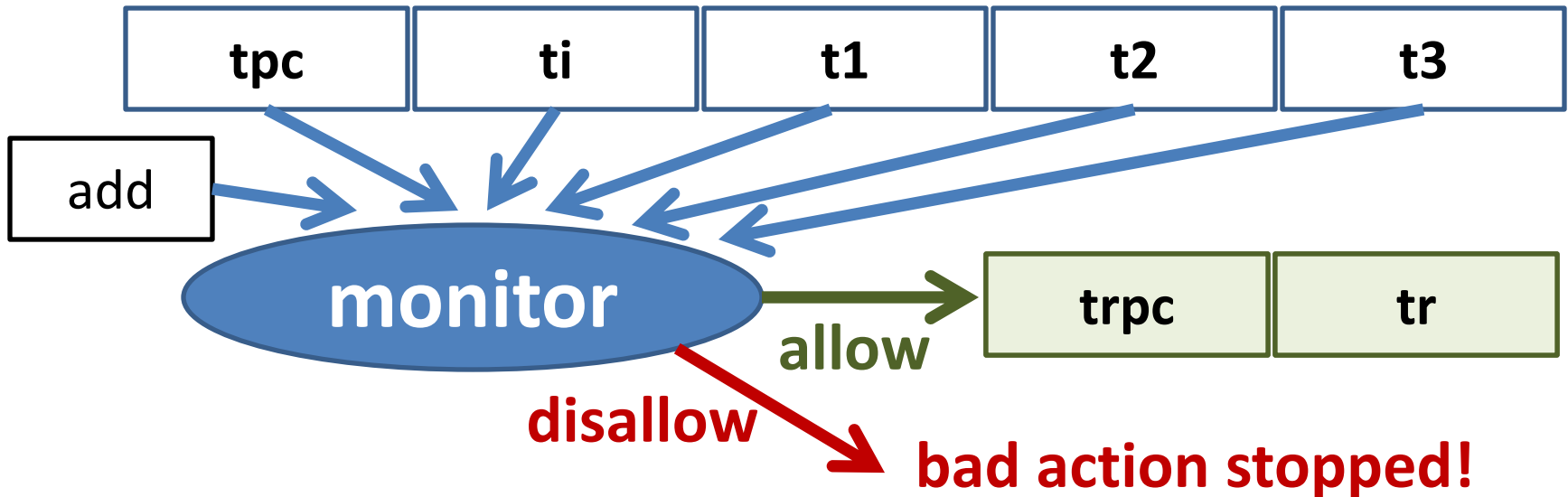
$T_{pc} ::= T_v$

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))



Memory safety micro-policy



1. Sets of tags

$T_v ::= i \mid \text{ptr}(c)$

$T_m ::= M(c, T_v) \mid F$

$T_{pc} ::= T_v$

2. Transfer function

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

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

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

Definition **transfer** iv :=

match iv with

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

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

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

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

...

Memory safety micro-policy



1. Sets of tags

$T_v ::= i \mid \text{ptr}(c)$

$T_m ::= M(c, T_v) \mid F$

$T_{pc} ::= T_v$

2. Transfer function

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

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

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**].

*This takes us beyond “noninterferent” reference monitors (more soon)

Open problems

- **Interaction with compiler, loader, linker, OS**
- **Secure micro-policy composition**
- **Verified optimizing compiler for micro-policies**
- **Reduced/more adaptive energy usage**
- **Modern RISC instruction set (e.g. ARM)**
- **More realistic processor**
(out-of-order execution, even multi-core)
- **Cache side channels**

Full abstraction

- **Golden standard for secure compilation**
 - $P \approx Q \leftrightarrow \text{compile}(P) \approx \text{compile}(Q)$
 - $P \approx Q = \forall C. C[P]$ 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**
 - working on this with **Yannis Juglaret**
 - Toy source language: Featherweight Java subset + **updates**
 - FJ classes protected from native classes they link with
 - Micro-policy combining:
 - isolated compartments, entry points
 - linear return capabilities
 - dynamic typing
- **Long term goal:** functional programming language

protects:

classes, methods

stack discipline

type safety

Open problem: composition

- **composing reference monitors is easy**
 - ... as long as they **can only stop execution**
- **richer interaction** for micro-policies:
 - **monitor services:** malloc, free, ...
classify, declassify, read label, ...
- **secure micro-policy composition is difficult**
 - e.g. composing anything with IFC can leak
 - memory safety + compartmentalization vs.
compartmentalization + memory safety?

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
- other projects:
 - **F***: formal verification of ML programs
 - **QuickChick**: property-based testing for Coq
- talk to me, will be around 2.5 more weeks
- **Thank you!**