Micro-Policies

A Framework for Verified, Tag-Based Security Monitors

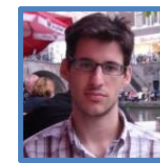
Cătălin Hrițcu

INRIA Paris-Rocquencourt, Prosecco team

Current collaborators on this project

- **Formal verification**
  - Arthur Azevedo de Amorim (UPenn; **INRIA intern 2014**)
  - Maxime Dénès (**INRIA Gallium;** previously UPenn)
  - Nick Giannarakis (ENS Cachan; **INRIA intern 2014**)
  - Cătălin Hrițcu (**INRIA Prosecco;** previously UPenn)
  - Yannis Juglaret (Paris 7; **INRIA intern 2015**)
  - Benjamin Pierce (UPenn)
  - Antal Spector-Zabusky (UPenn)
  - Andrew Tolmach (Portland State)
- **Hardware architecture**
  - André DeHon, Udit Dhawan, ... (UPenn)










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!*
- **Software bears most of the burden for security**
  - pervasive security enforcement impractical
  - security-performance tradeoff
  - just write secure code ... all of it!
- **Consequence: vulnerabilities in every system**
  - **violations of well-studied safety and security policies**

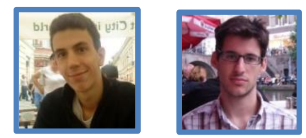
Micro-policies

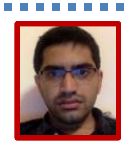
- general **dynamic enforcement mechanism** for
  - critical invariants of **all** machine code
  - high-level abstractions and programming models
- main idea: add **word-sized tag** to each machine word
  - "this word is an instruction, and this one is a pointer"
  - "this word comes from the net, and this is private to A and B"
- **tags propagated on each instruction** ... efficiently
  - tags and rules **defined by software** (**miss handler; verified**)
  - **accelerated by hardware** (rule cache, near-zero overhead hits)

Micro-policies for ...

- information flow control (IFC) [Oakland’13, POPL’14]
- monitor self-protection
- compartmentalization
- dynamic sealing
- memory safety
- code-data separation
- control-flow integrity (CFI)
- taint tracking
- ...

Verified (in Coq) [Oakland’15]



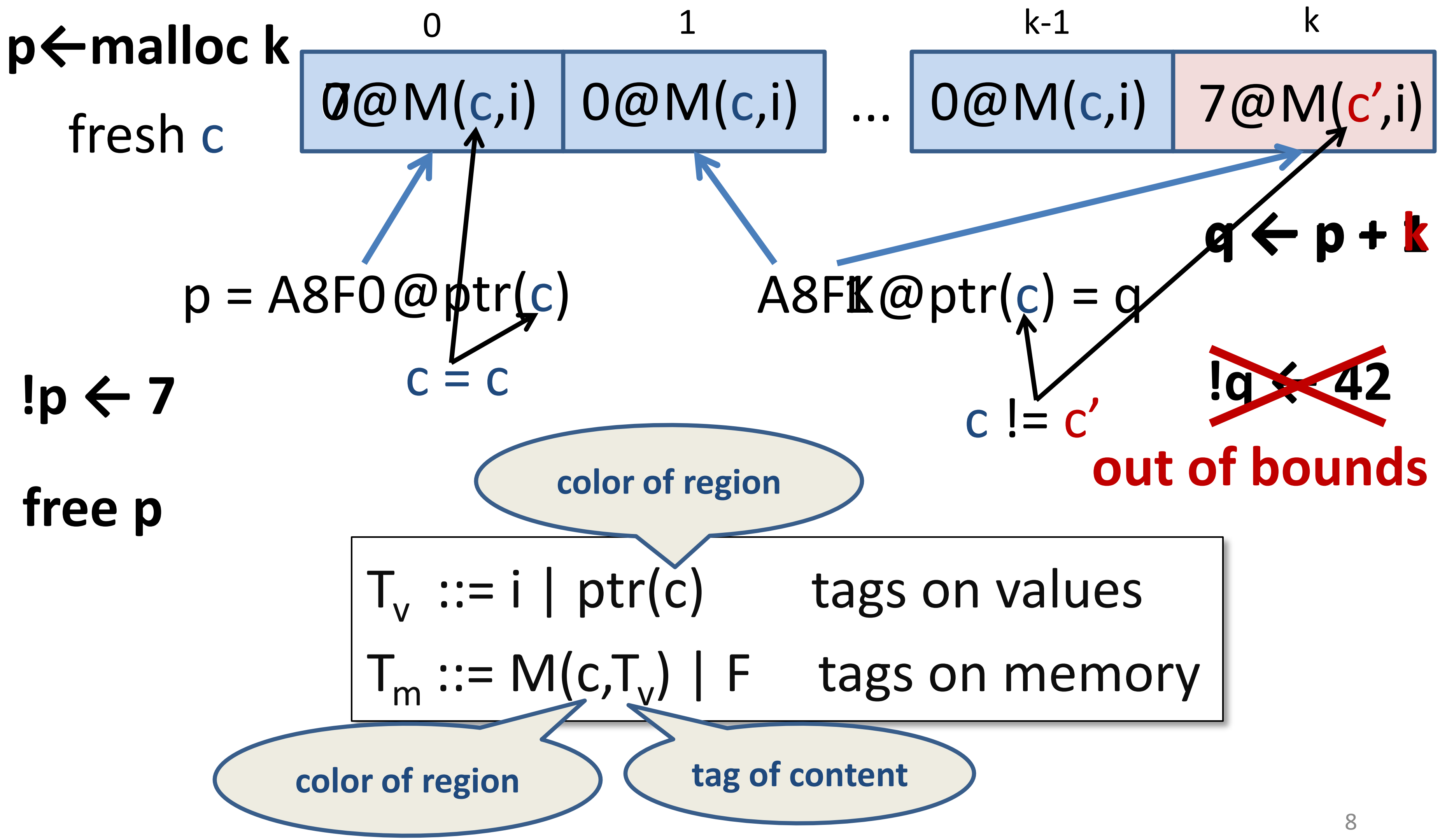


Evaluated (<10% runtime overhead) [ASPLOS’15]

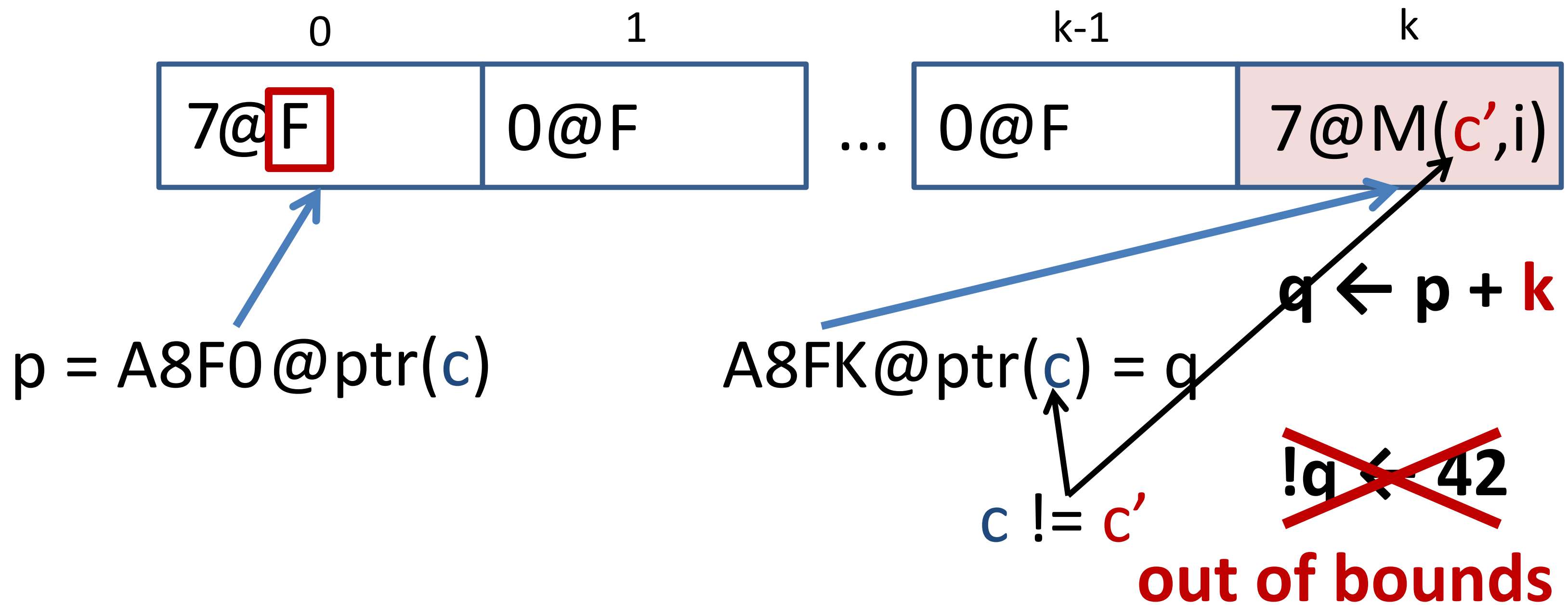
Memory safety

- Prevent
  - **spatial violations**: reading/writing out of bounds
  - **temporal violations**: use after free, invalid free
- 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



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

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))

Definition **transfer** iv :=

match iv with

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

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

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

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

...

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))

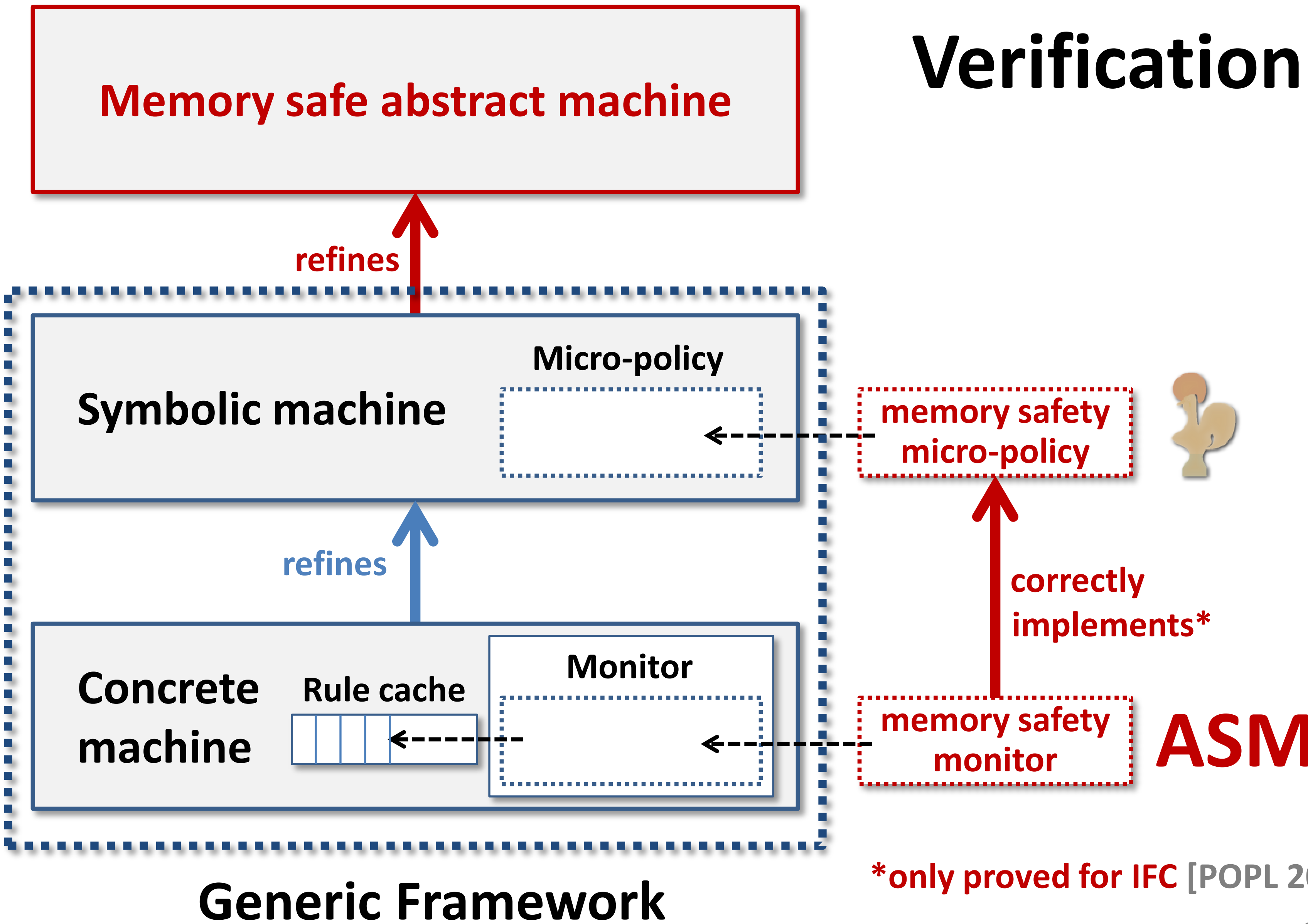
3. Monitor services

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

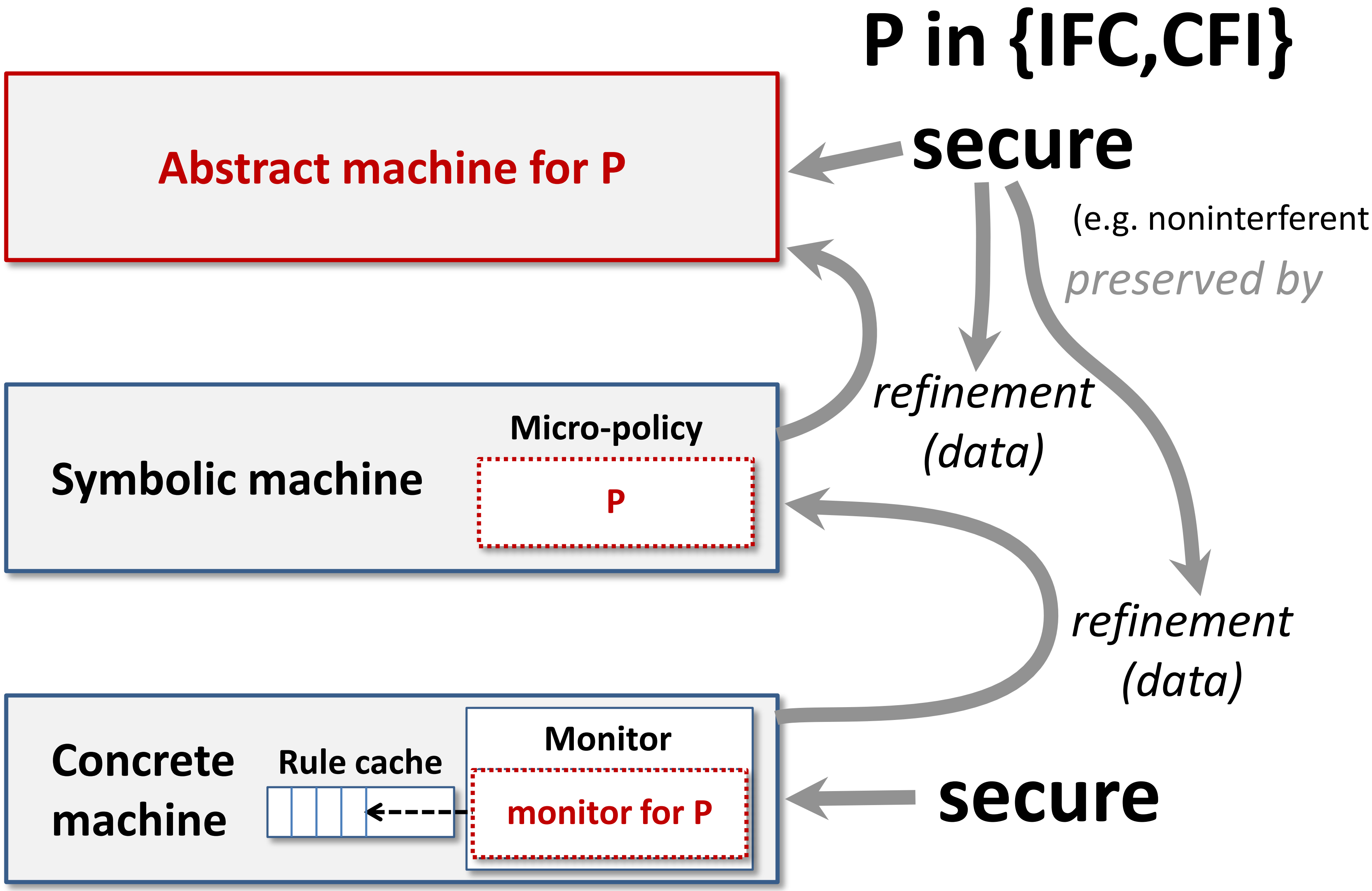
Definition **mem_safety_services** : list service :=

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

Verification



*only proved for IFC [POPL 2014]



P in {IFC,CFI}
Abstract machine for P
secure
(e.g. noninterferent)
preserved by
refinement
(data)
Symbolic machine
Micro-policy
P
refinement
(data)
Concrete
machine
Rule cache
Monitor
monitor for P
secure

Future

- **Interaction with loader and compiler** (static + dynamic) 
  - **Fully abstract compilation to micro-policies (Yannis, intern 2015)**
- ... and **operating system** (e.g. protect the OS itself)
- **Micro-policy composition**, formally
- **Language** for writing micro-policies (symbolic rules)
- **Verification for real RISC** instruction set (e.g. ARM)
- **More realistic processor** (our-of-order execution, multi-core)
- **Concurrency** (big can of worms, data race detection)
- **More micro-policies** (e.g. stack protection, ...)
- **Formally study expressive power** of micro-policies
- **Switch to F*** for the proofs