Micro-Policies

Hardware-Assisted Tag-Based Security Monitors

Cătălin Hrițcu

Inria Paris-Rocquencourt, Prosecco team

Computer systems are insecure

Computer systems are insecure

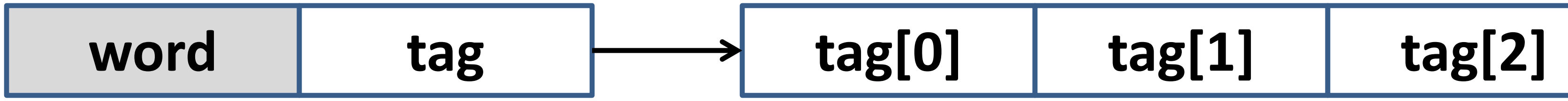
- **Today's computers are mindless bureaucrats**
  - "write past the end of this buffer" ... *yes boss!*
  - "jump to this untrusted integer" ... *right boss!*
  - "return into the middle of this instruction" ... *sure boss!*
- **Software bears most of the burden for security**
  - pervasive security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

HP reinventing the computer

- **opportunity to fix this**:
  - devise a computer that's **not just faster**, but that's also **significantly more secure**
- **it's possible!**
  - new security mechanism called **micro-policies**

Micro-policies

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



- words in memory and registers are all tagged

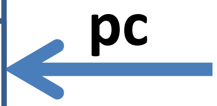
pc	**tag**
r0	**tag**
r1	**tag**
r2	**tag**

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

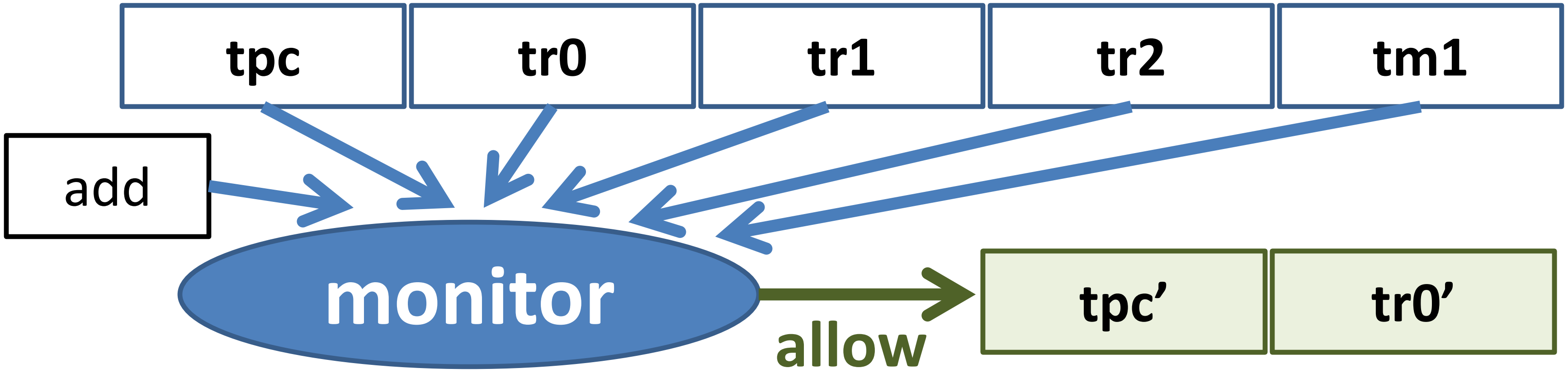
Tag-based instruction-level monitoring

pc	tpc
r0	tr0
r1	tr1
r2	tr2

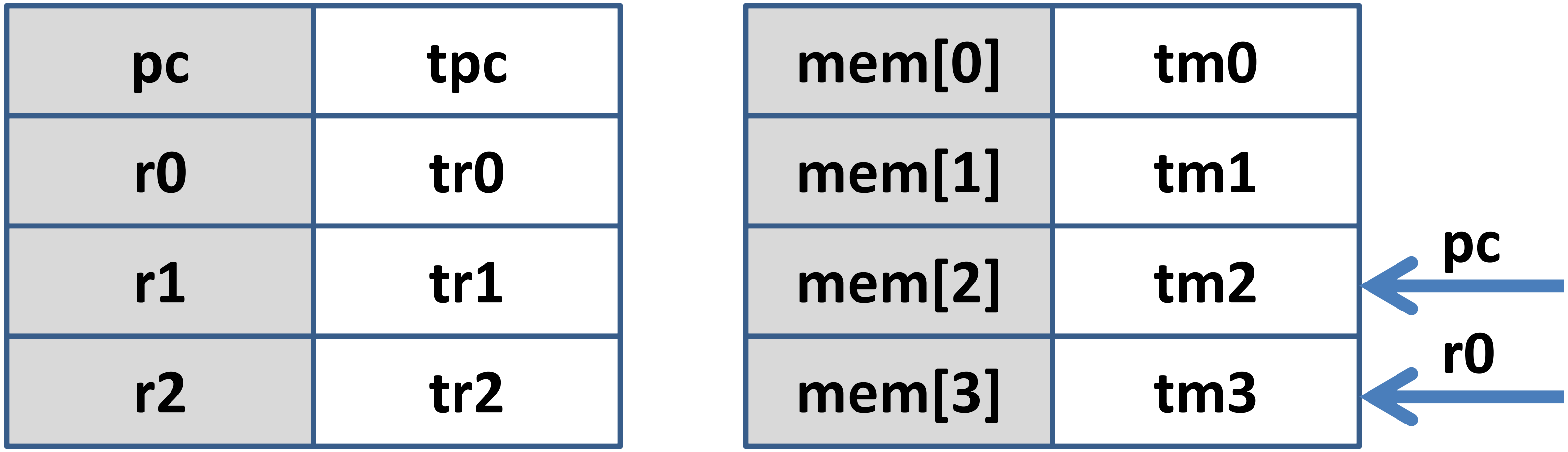
mem[0]	tm0
mem[1]	tm1
mem[2]	tm2
mem[3]	tm3



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



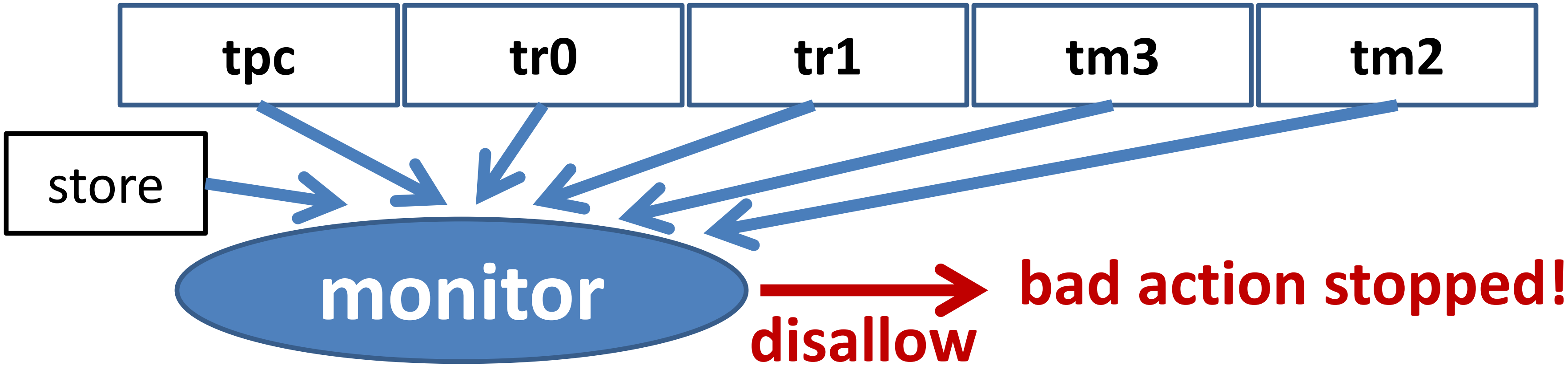
Tag-based instruction-level monitoring



pc	tpc
r0	tr0
r1	tr1
r2	tr2

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

decode(mem[1]) = store r0 r1



Features of micro-policies

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

Expressiveness

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

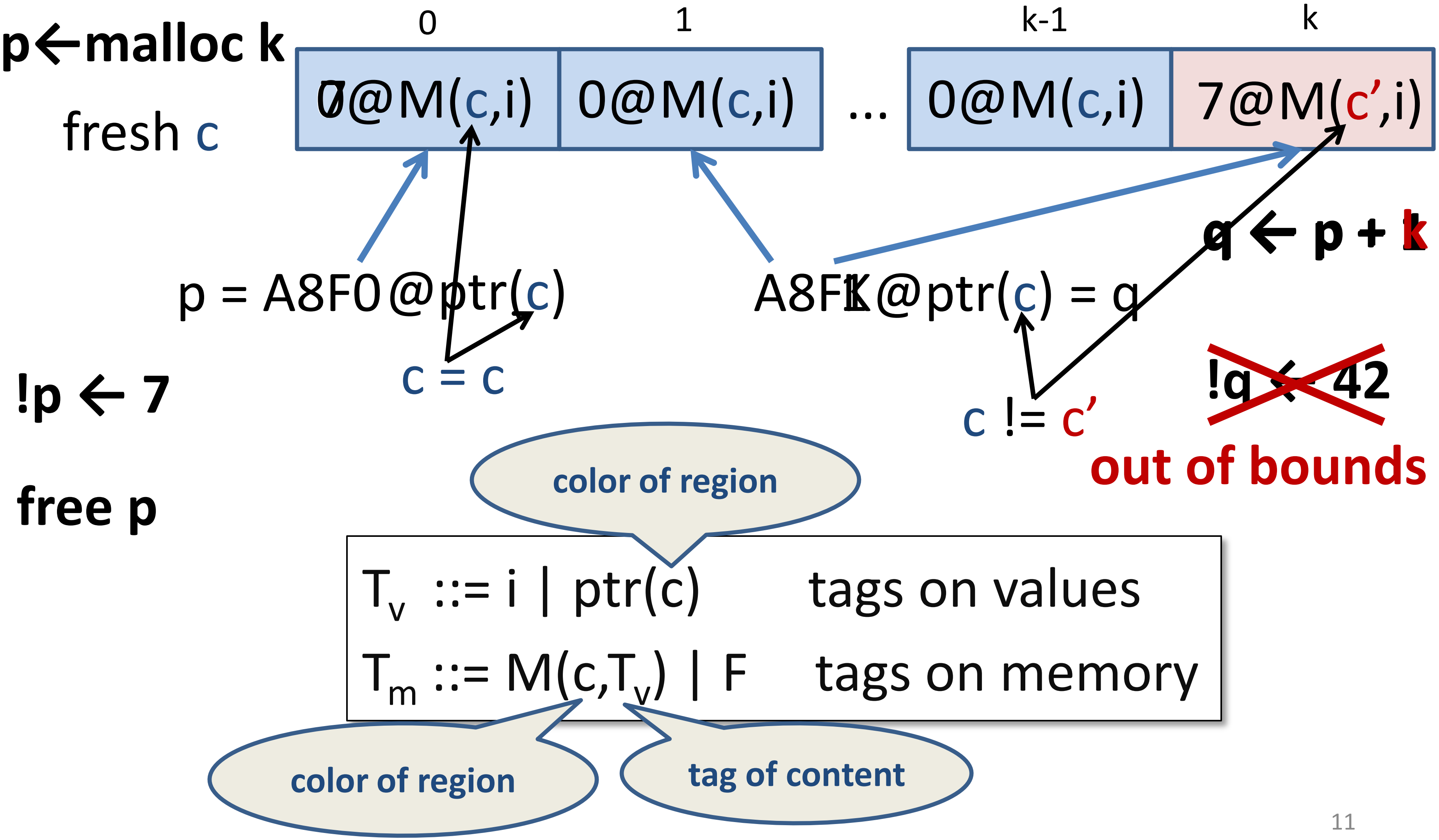
and probably a lot more!

History:
•DARPA CRASH/SAFE project
•different mechanisms for most of these things
•micro-policies were only used for IFC ... but they are a lot more expressive than we realized at first

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 now)
- 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

0 1 k-1 k

7@M(c,i) 0@M(c,i) … 0@M(c,i) 7@M(c’,i)

p = A8F0@ptr(c)

c = c

A8FK@ptr(c) = q

q ← p + k

c != c’

~~**!q ← 42**~~

out of bounds

!p ← 7

free p

color of region

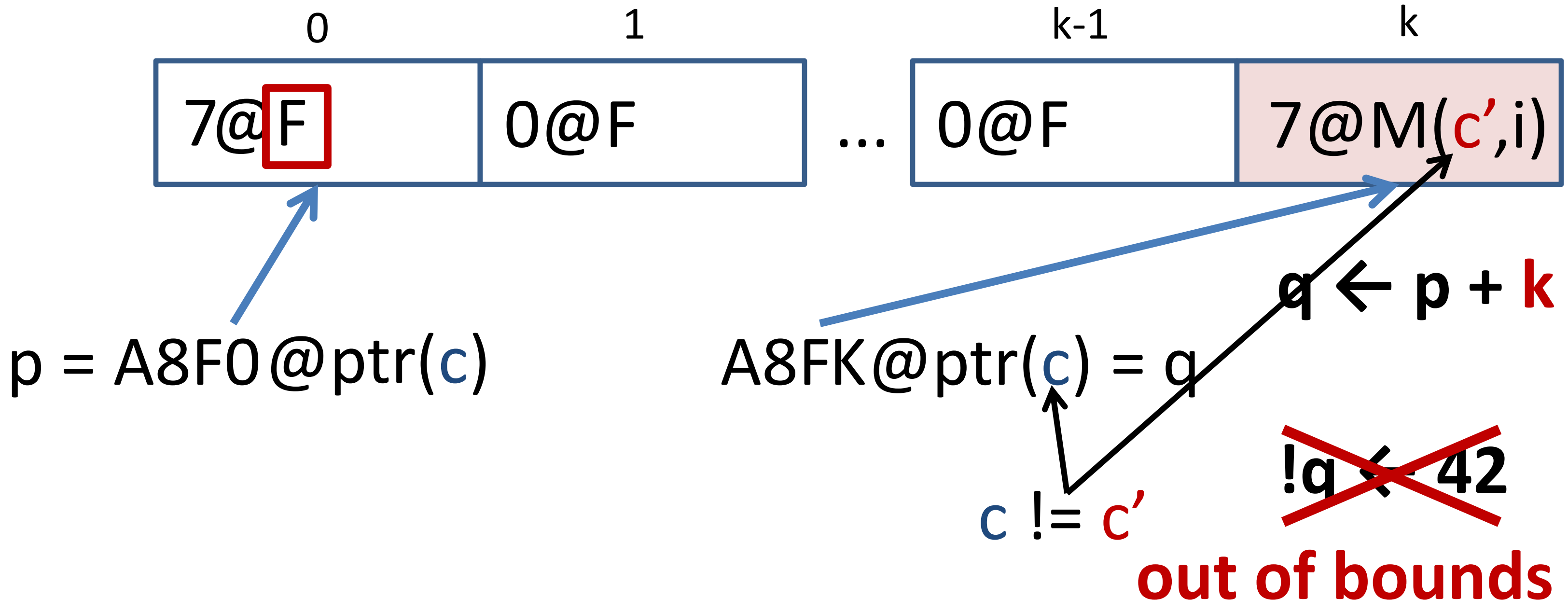
T_v ::= i | ptr(c) tags on values

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

color of region

tag of content

Memory safety micro-policy



free p

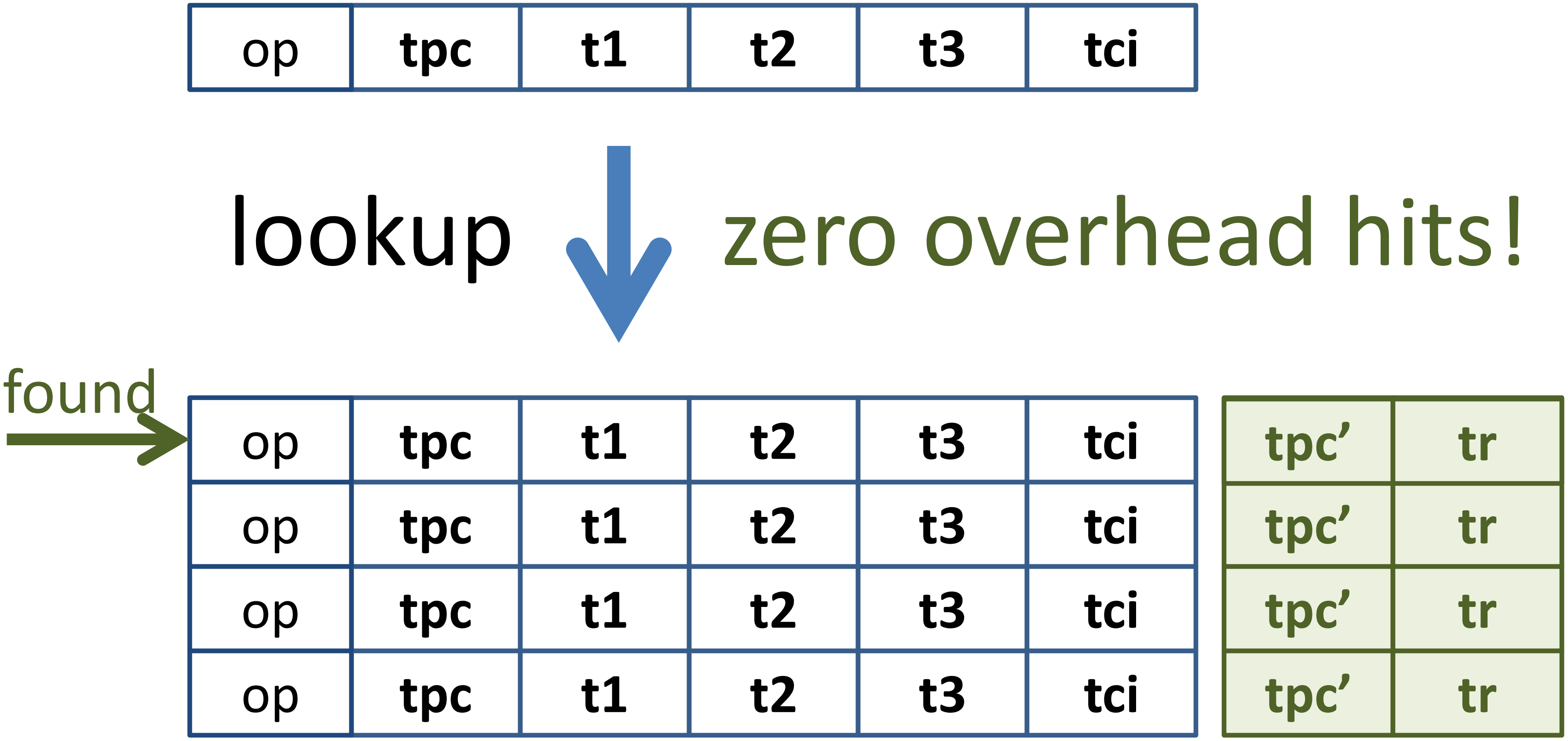
~~**x ← !p**~~

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**

lookup

zero overhead hits!

found

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

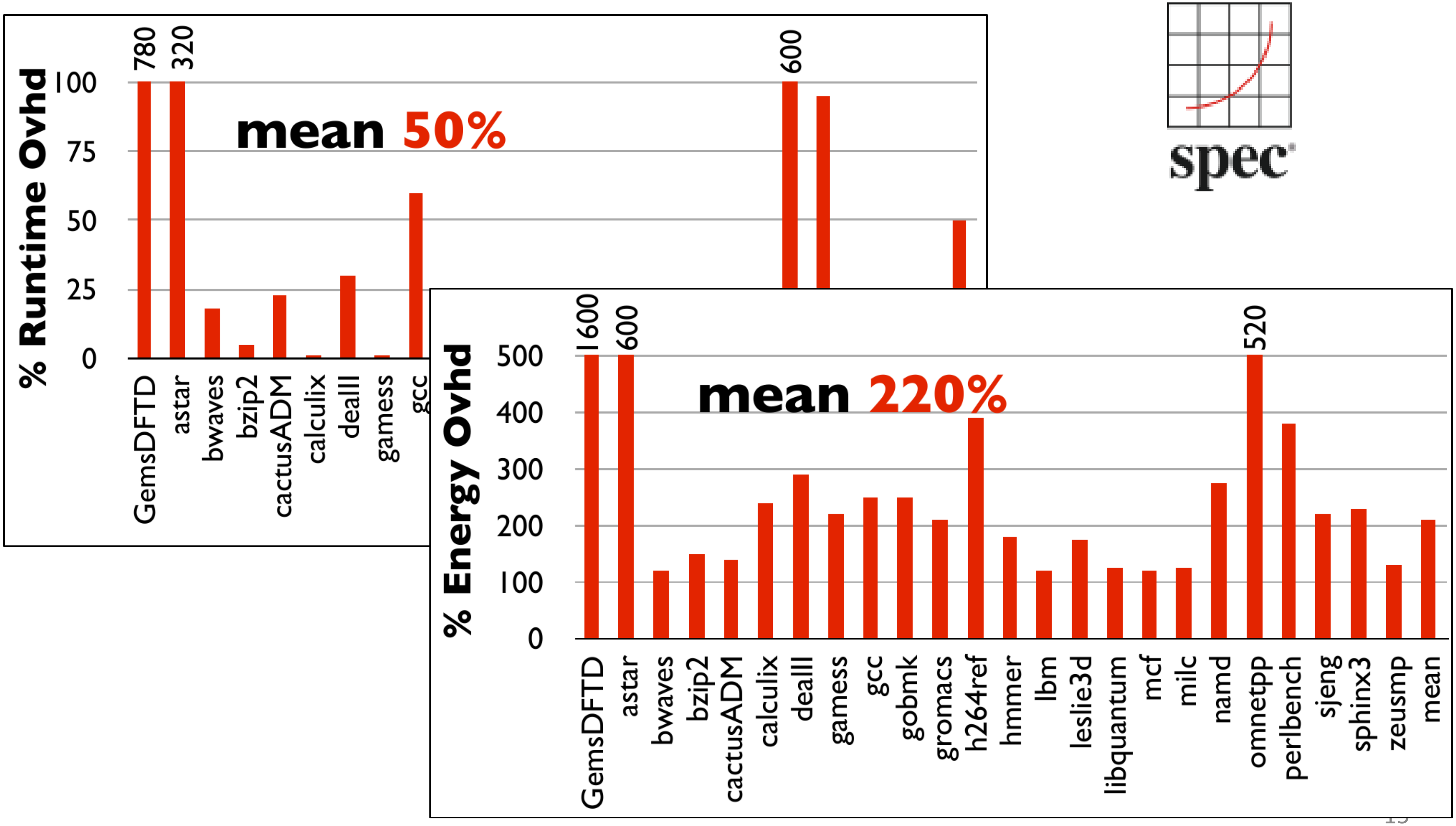
produced “rule” cached

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

Experiments for naive implementation

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



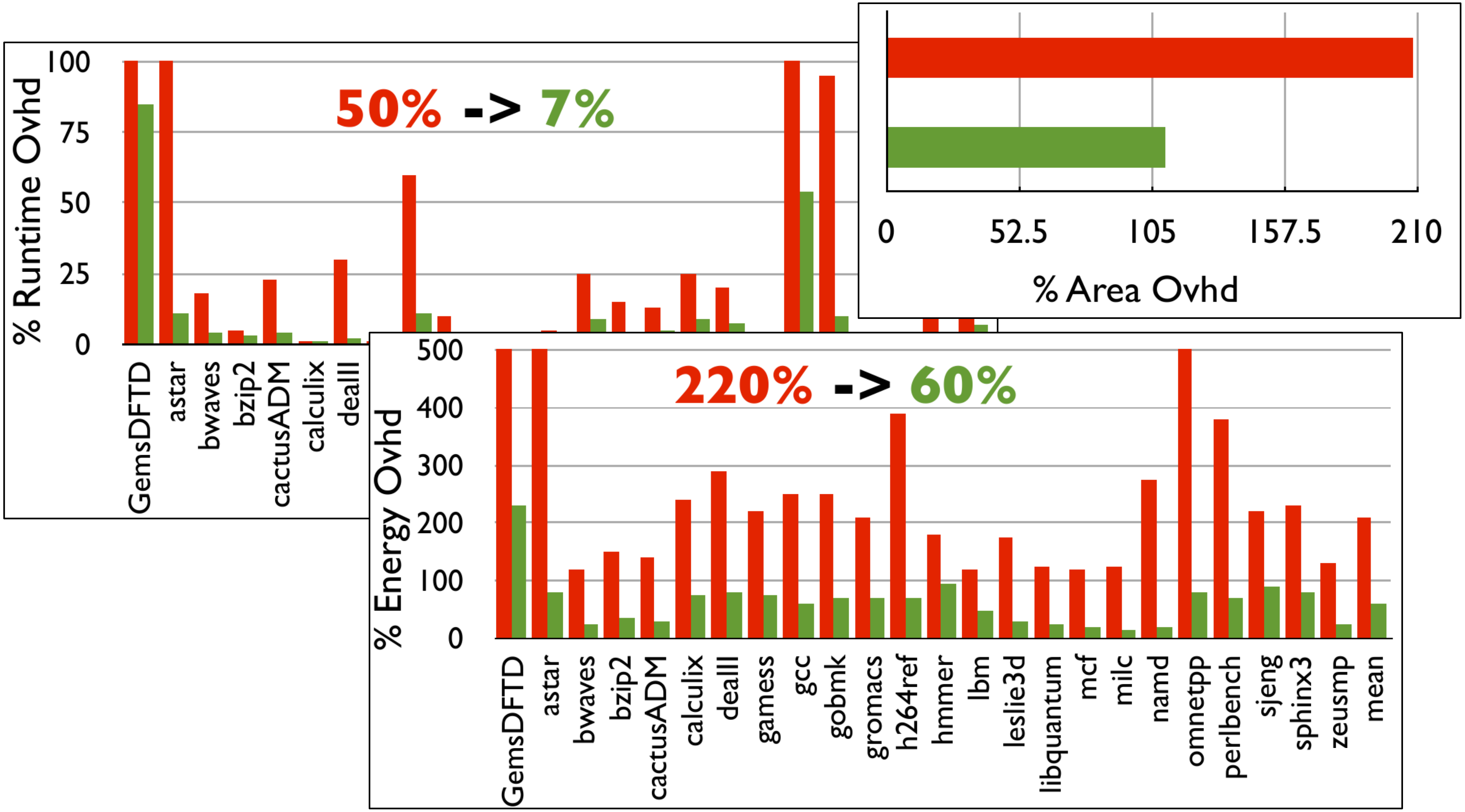
Targeted architectural optimizations

- grouping opcodes and ignoring unused tags
- transferring only unique tags to/from DRAM
- using much shorter tags on-chip
- caching composite policies separately

Experiments for optimized impl.

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

no free lunch

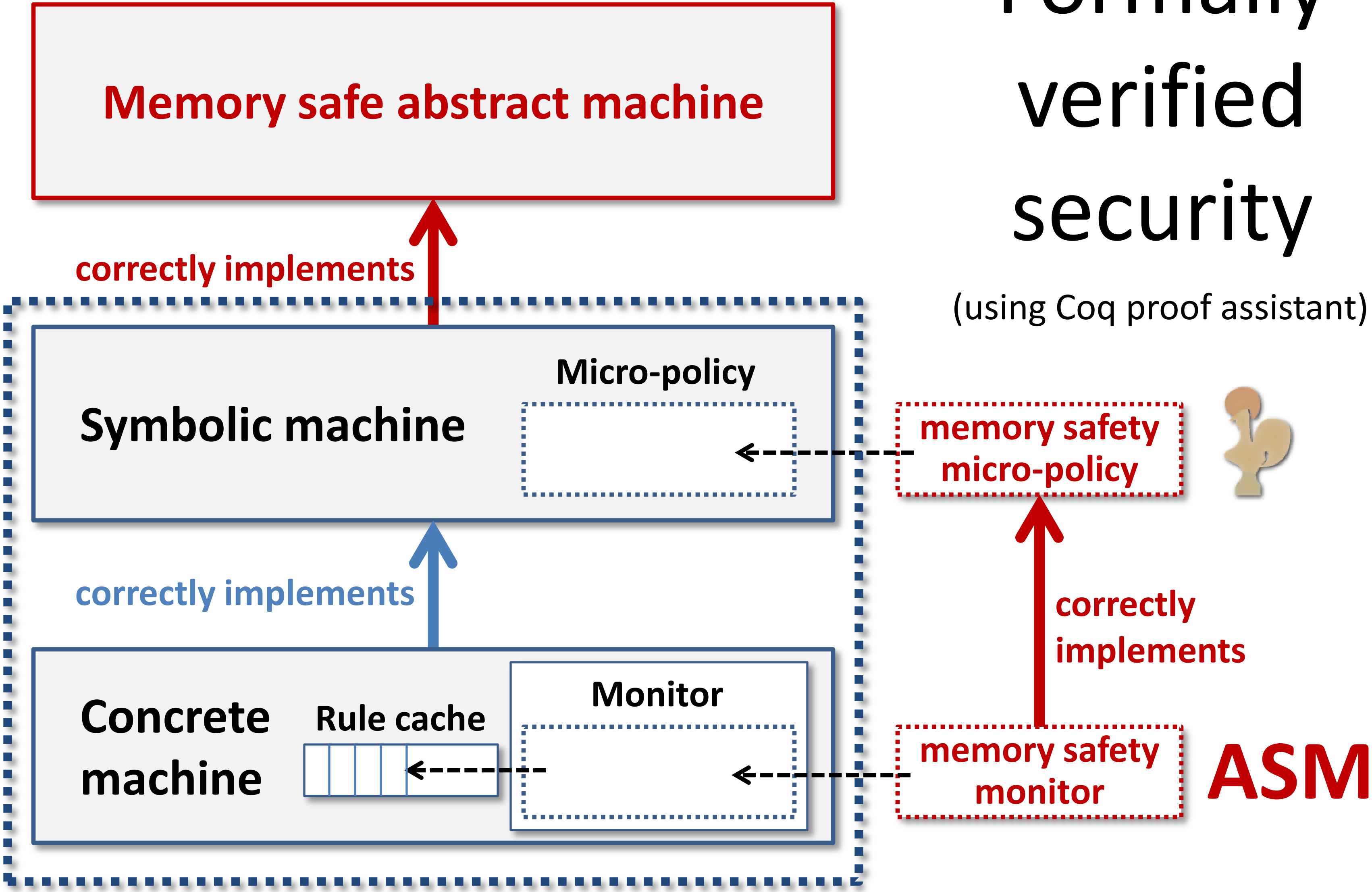


50% -> 7%

220% -> 60%

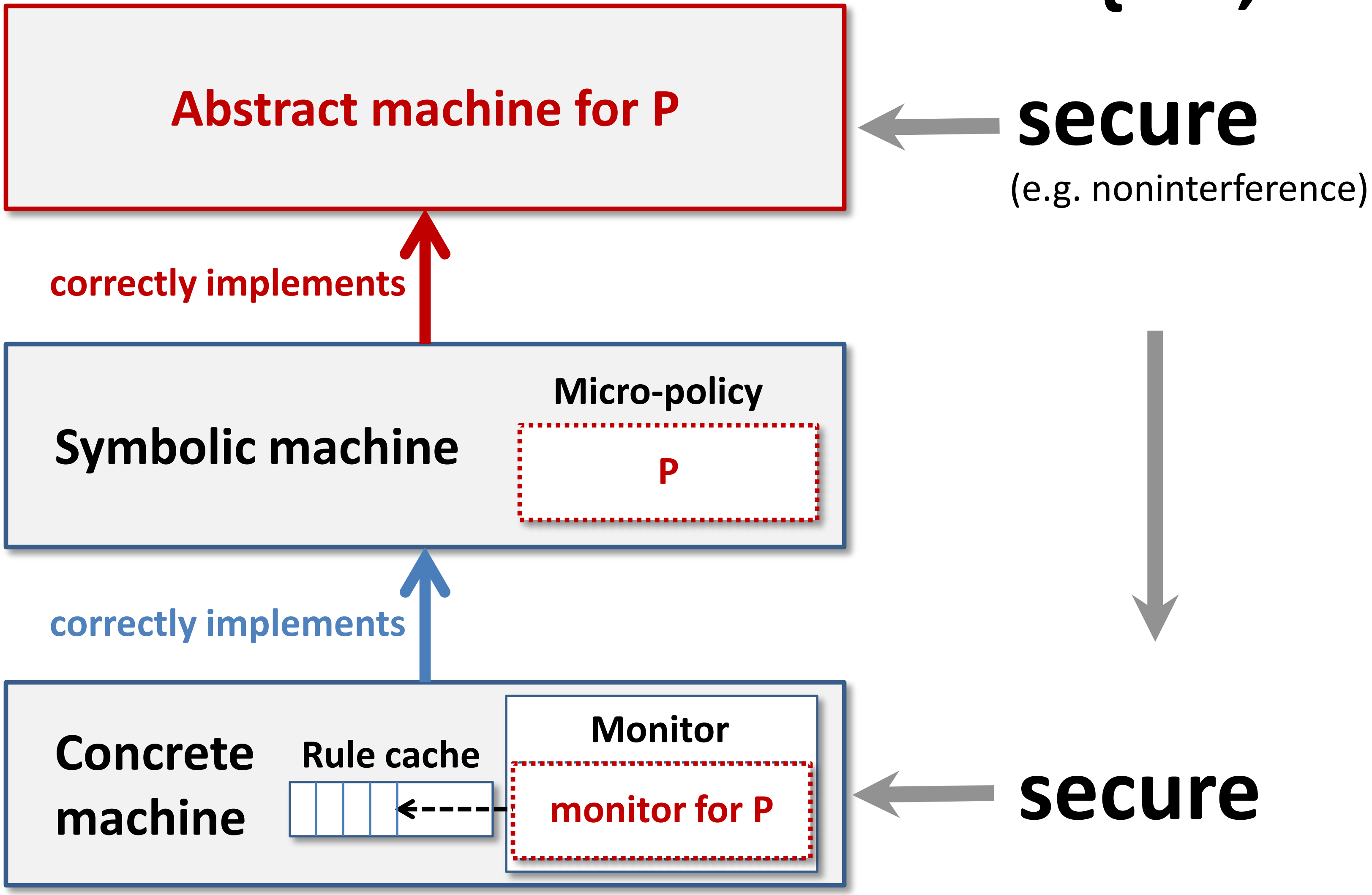
Formally verified security

(using Coq proof assistant)



Generic Framework

P in {IFC,CFI}



Abstract machine for P

secure

(e.g. noninterference)

correctly implements

Symbolic machine

Micro-policy

P

correctly implements

Concrete machine

Rule cache

Monitor

monitor for P

secure

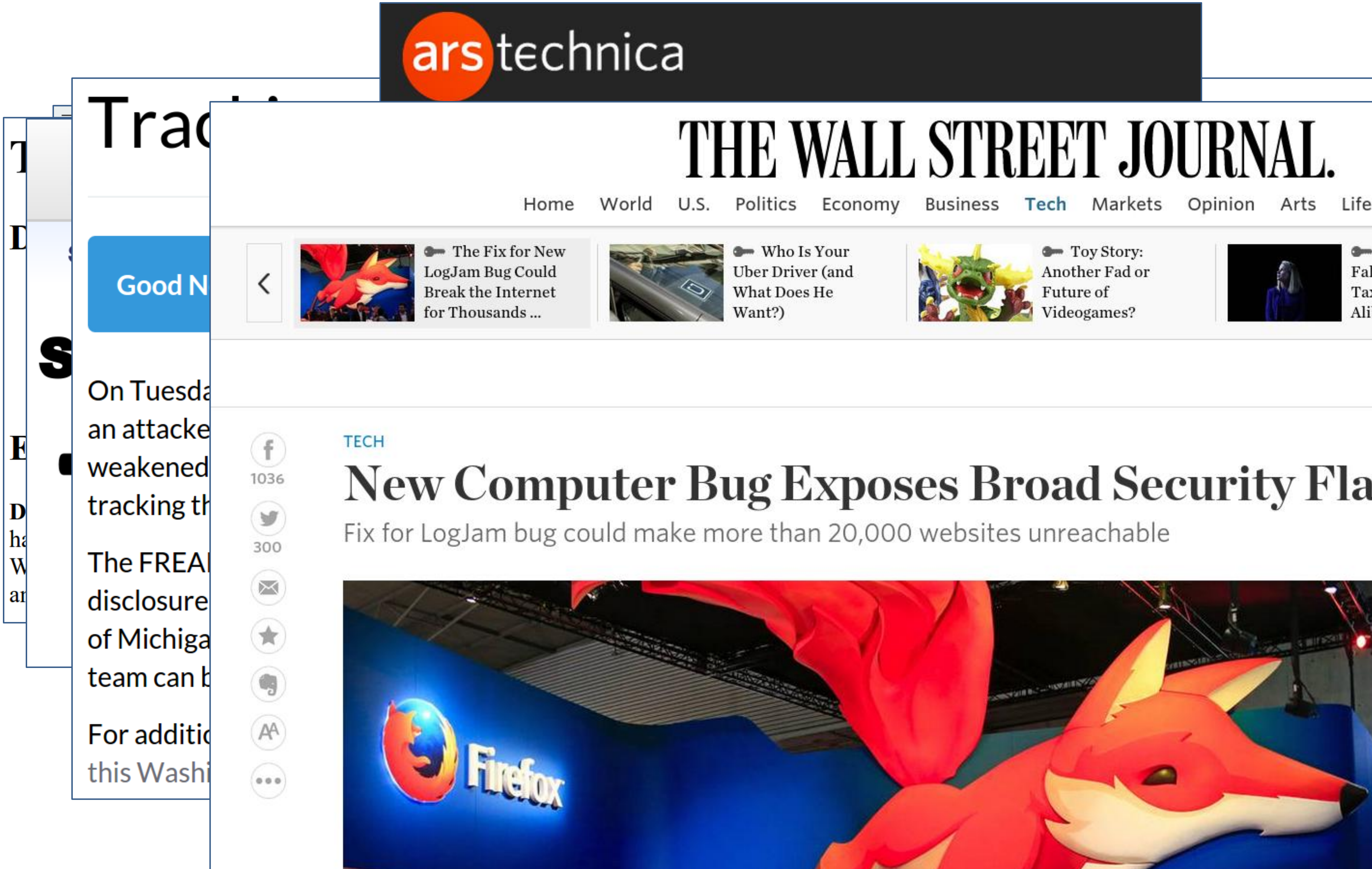
Upcoming

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

Take away

- **Micro-policies**, novel security mechanism that's:
  - **low-level, fine-grained, expressive, flexible, efficient, formally secure**
- Current collaborators (**INRIA** & UPenn):
  - **Arthur Azevedo de Amorim**, André DeHon, **Maxime Dénès**, Udit Dhawan, **Nick Giannarakis**, **Cătălin Hrițcu**, **Yannis Juglaret**, Benjamin Pierce, Antal Spector-Zabusky, Andrew Tolmach, Nikos Vasilakis

Other highlights in Prosecco team



arstechnica

Trac

Good N

On Tuesda
an attacke
weakened
tracking th

The FREA
disclosure
of Michiga
team can b

For additic
this Washi

THE WALL STREET JOURNAL.

The Fix for New LogJam Bug Could Break the Internet for Thousands ...

Who Is Your Uber Driver (and What Does He Want?)

Toy Story: Another Fad or Future of Videogames?

Fal Ta Ali

TECH

New Computer Bug Exposes Broad Security Fla

Fix for LogJam bug could make more than 20,000 websites unreachable

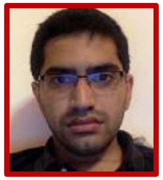
Other highlights in Prosecco team

- programming securely with cryptography
- **Proverif** and **Cryptoverif** protocol analyzers
- **miTLS**: verified reference implementation
- **F***: program verification system for OCaml/F#
- **QuickChick**: property-based testing for Coq
- Prosecco permanent researchers:
  - Karthikeyan Bhargavan (leader), Bruno Blanchet, Cătălin Hrițcu, Graham Steel (Cryptosense startup)

BACKUP SLIDES

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)



The end

- Today’s computer’s were designed long time ago
- **Computer designers from the 50s-90s have a good excuse for getting security wrong** (e.g. horrors like buffer overflows):
  - security wasn't a big issue before the Internet age
  - performance was much more important
- Today the situation is reversed
  - and HP has an opportunity to fix security
  - **but HP will have no excuse if it reinvents the insecure computer**