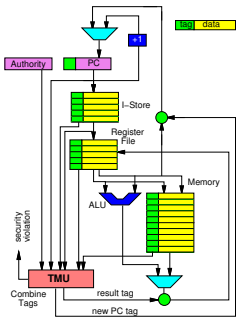
The SAFE Machine

An Architecture for Pervasive Information Flow

Benjamin C. Pierce
University of Pennsylvania

Computer Security Foundations
June 28, 2013



Penn
University of Pennsylvania

Why are computers so insecure?

expensive crossing between protection domains

Single protection boundary (user/kernel)

Memory protection at process/page granularity

One culprit...

monolithic kernel design

Legacy design decisions

embedded in the HW/SW ecosystem

macro instead of micro-kernels

unchecked address arithmetic

manual memory management

"root" user

raw seething bits

numeric issues (under/overflow, implicit promotion to unsigned, etc.)

"uni-typed" semantics

(pointer = integer = instructions)

Crash/
SAFE

BAE SYSTEMS

Shown: Sumit Ray, Howard Reubenstein, Andrew Sutherland, Tom Knight, Olin Shivers, Benjamin Pierce, Ben Karel, Benoit Montagu, Jonathan Smith, Cătălin Hriţcu, Randy Pollack, André DeHon, Gregory Malecha, Basil Krikeles, Greg Sullivan, Greg Frazier, Tim Anderson, Bryan Loyall

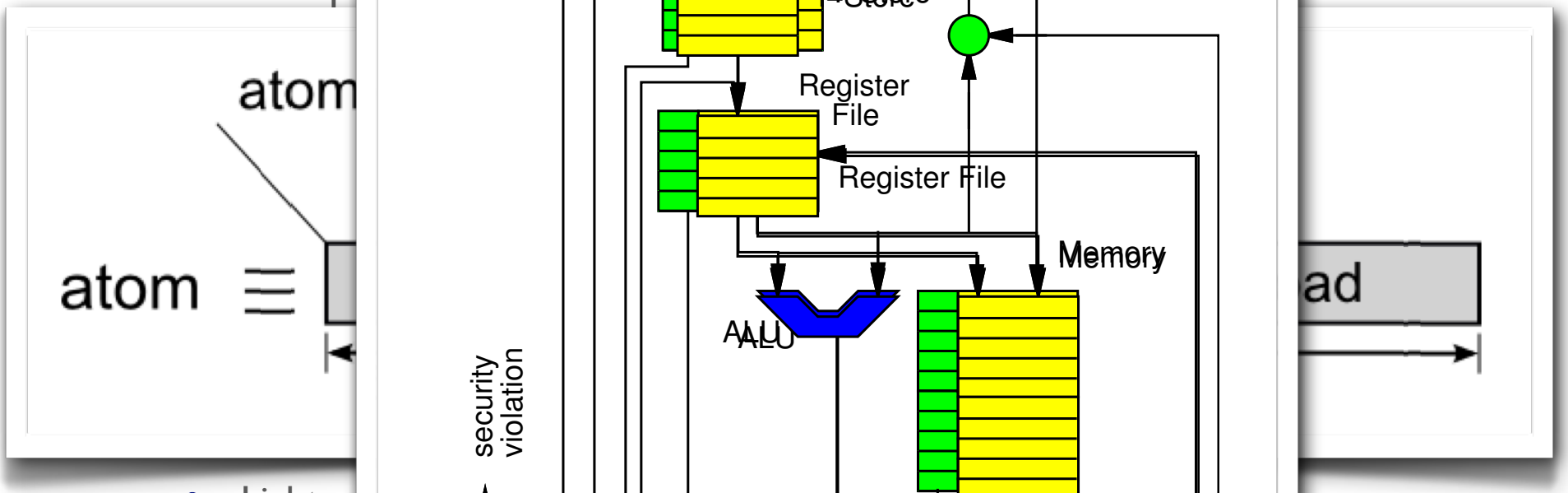
Not shown: Greg Morrisett, Peter Trei, David Wittenberg, Amanda Strnad, Justin Slepak, David Darais, Robin Morisset, Chris White, Anna Gommerstadt, Marty Fahey, Tom Hawkins, Karl Fischer, Hillary Holloway, Andrew Kaluzniacki, Michael Greenberg, Andrew Tolmach, Antal Spector-Zabuski, Leonidas Lampropoulos, Tyler Brown, Ian Nightingale, Udit Dhawan, Albert Kwon, Jesse Tov, Arthur Azevedo de Amorim, Nathan Collins, Arun Thomas, Shannon Spires, ...

SAFE

- Clean-slate redesign of the entire system stack
  - Hardware
  - System software
  - Programming languages
- Support for critical security primitives at all levels (from hardware up)
  - Memory safety
  - Strong dynamic typing
  - Information flow and access control
- Verification of key mechanisms deeply integrated into design process

SAFE:

- "Low-fat"
  - Every
  - Compa
  - Lightwe
  - Linear
  - Hardwa



Why new hardware?

- Explore how to effectively spend hardware resources on security
- Reconsider traditional sources of complexity and vulnerability
- Remove compiler from TCB
  - (at least partly)
- Make security mechanisms available for writing low-level systems code

SAFE: OS level

- "Zero-Kernel OS"
  - OS services organized into a set of mutually suspicious components
  - Goal: No "omniprivileged component"
  - not part of today's story, but central to full SAFE design
- ConcreteWare = runtime services for SAFE
  - process management and scheduling
  - storage allocation and GC
  - virtualization of tagging hardware

Much more on this coming up...!

SAFE: Application level

- Breeze: A mostly functional, security-oriented language
- All type- and security checks are dynamic
  - Static typechecking → dynamic contract checking
- Fine-grained: every value annotated with an IFC label
  - DC label model (cf. LIO system)
- Labels, principals, authorities are first class
- Labels are *public*
  - *bracketing* mechanism prevents flows through labels
  - ... and supports IFC-safe exception handling [Oakland S&P '13]
- Access control (clearance) *á la* Flume, Histar, etc.
- Erlang-style concurrency

Some Related Work...

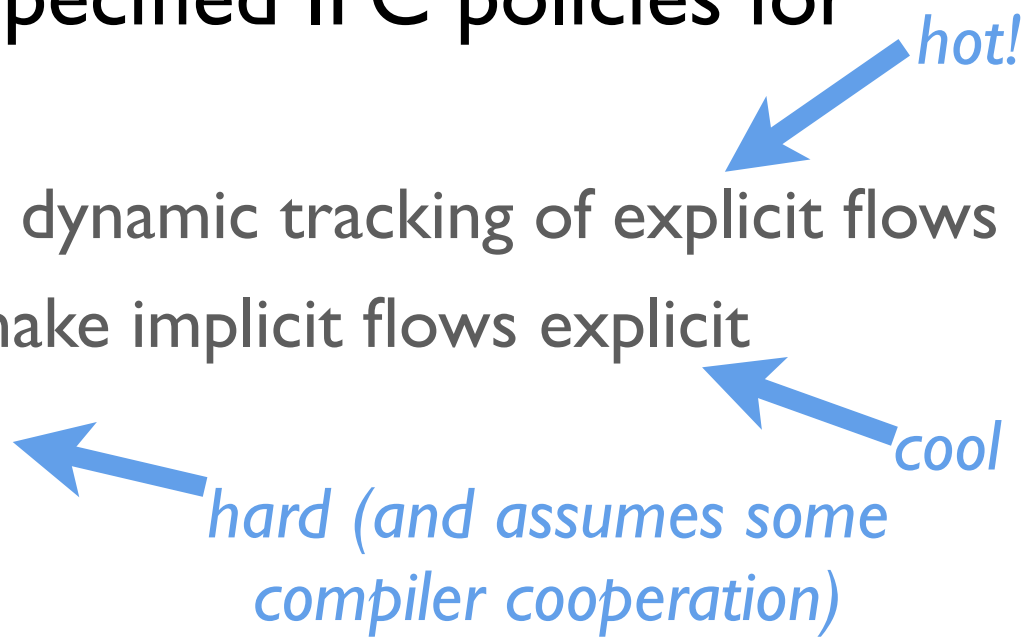
Related Work: LIO

- Haskell library for dynamic IFC
  - Basis for the Stanford HAILS web framework
- Similarities
  - Purely dynamic IFC
  - Same label model ("disjunction categories")
  - Public labels
  - ...
- Differences
  - Coarser-grained than SAFE
    - explicit "labeling nodes" in data structures
  - Pure software approach

[Stefan *et al*, 2011]

Related Work: RIFLE

- Soundly track user-specified IFC policies for unmodified binaries
  - Hardware-supported dynamic tracking of explicit flows
  - Binary rewriting to make implicit flows explicit
  - Heroic static analysis



- Recent Coq formalization and proof of NI
  - for a simplified model

[Vachharajani *et al.*, Micro '04]
[Beringer, APLAS '12]

Related Work: seL4

- Heroic machine-checked correctness proof for a real microkernel
- More recent: machine-checked proof of noninterference for a separation-kernel configuration
- Well-structured proof architecture yields huge reduction in verification effort
  - Prove NI for an abstract specification (for a deterministic notion of NI)
  - Prove refinement between spec and a concrete implementation

for us too!

[Klein *et al*, '09]
[Murray *et al*, '13]

Related Work: ARIES/TIARA

- Direct ancestors of SAFE
- ARIES proposed using a hardware rule cache to speed information-flow tracking
- TIARA proposed the idea of a Zero-Kernel Operating System and outlined a concrete architecture
- Paper designs
  - SAFE is the first concrete realization of these ideas

[Shrobe *et al*, '09]
[Brown and Knight, '01]

Any questions?

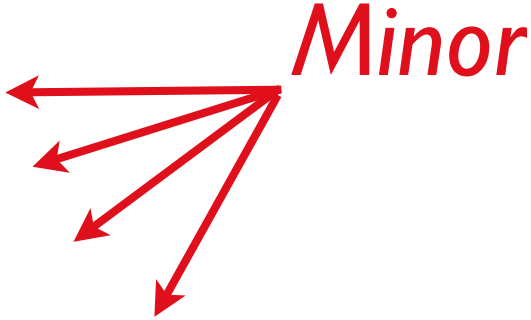
(and formalized)

A Formal^Model of SAFE IFC

Arthur Azevedo de Amorim, Nathan Collins,
André DeHon, Delphine Demange, Cătălin Hriţcu,
David Pichardie, Benjamin C. Pierce,
Randy Pollack, Andrew Tolmach

Simplifications

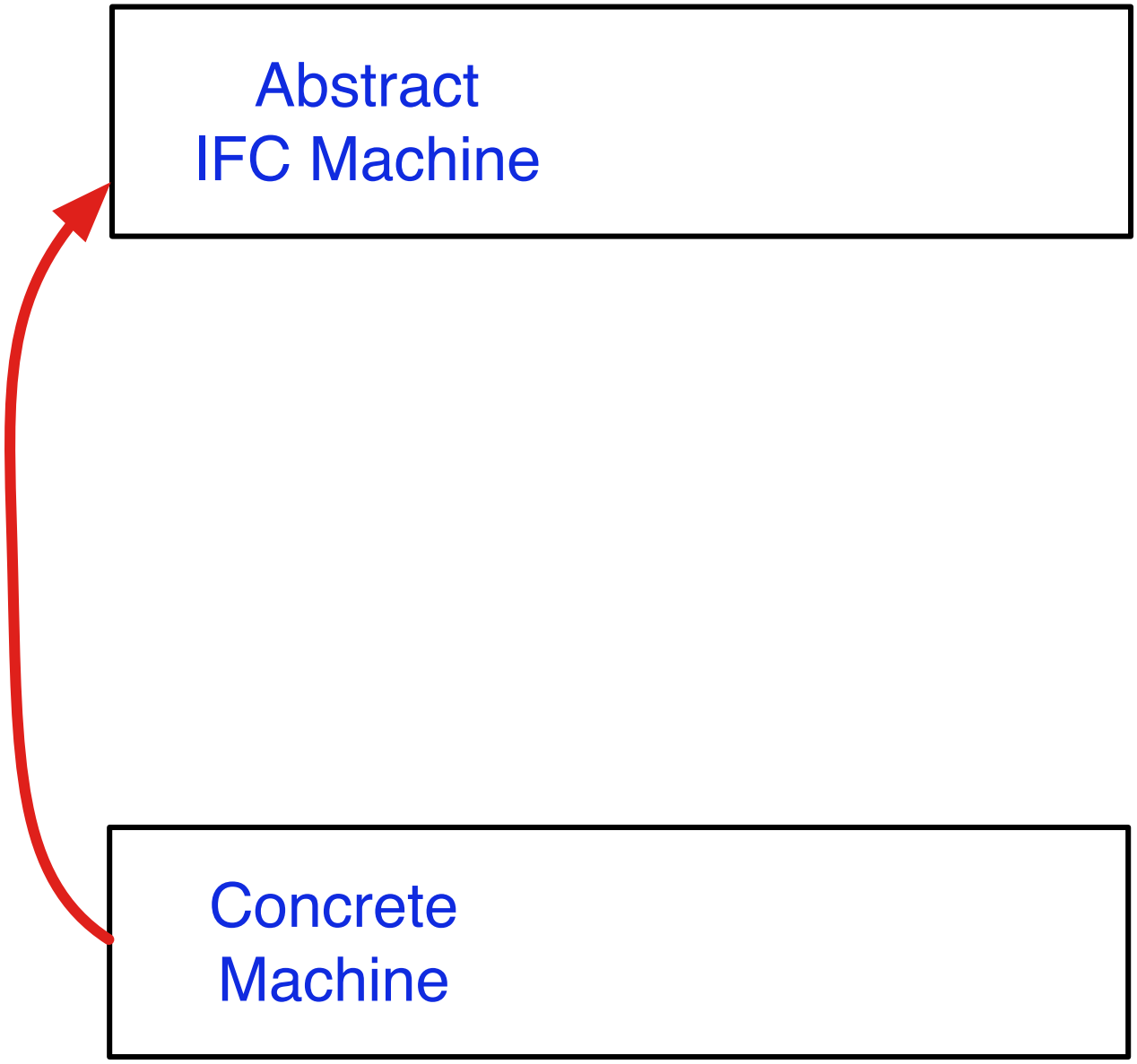
- Deterministic, single-threaded machine
- Conventional memory model
  - pointers are just integers
  - single kernel protection domain

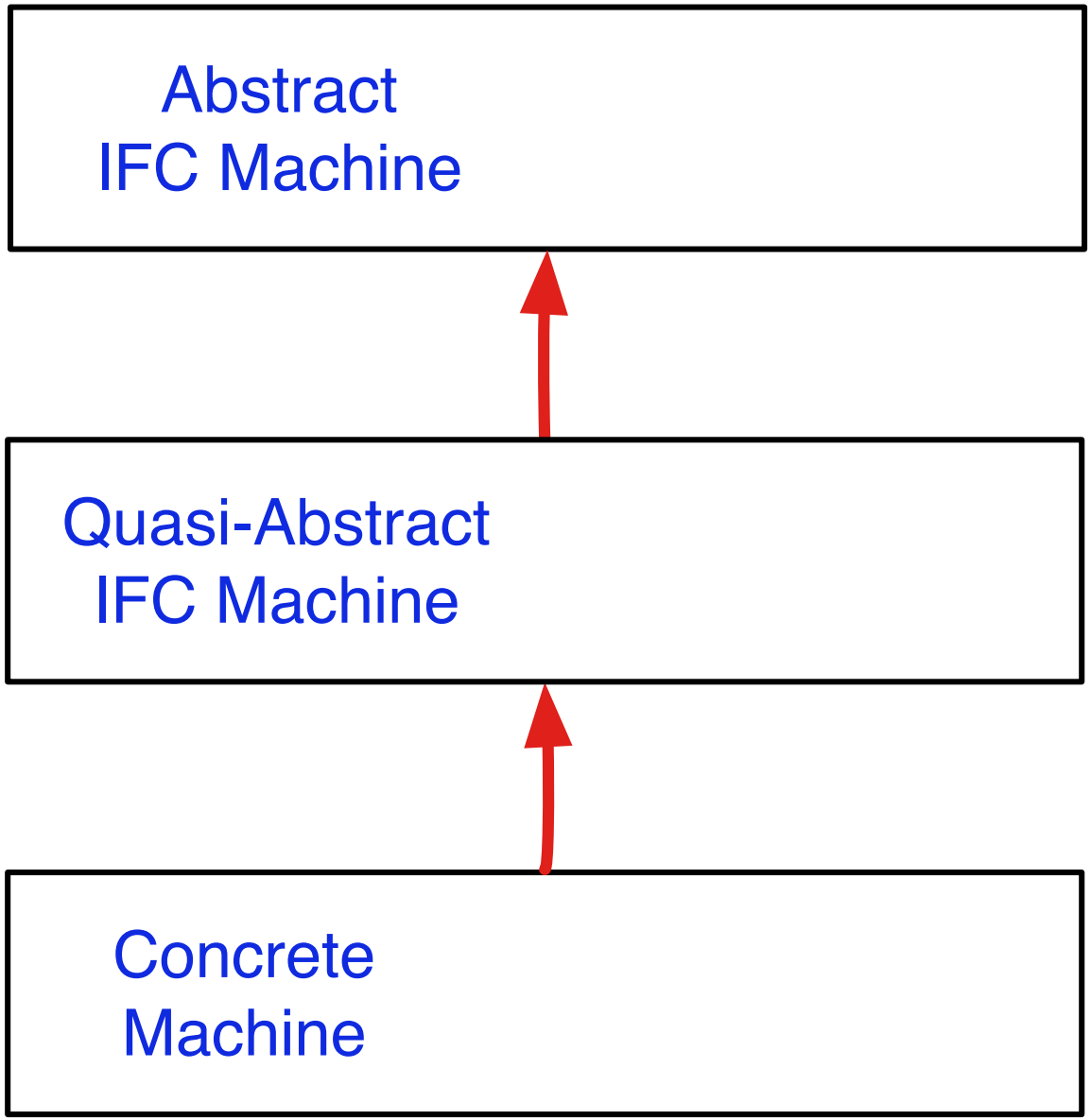


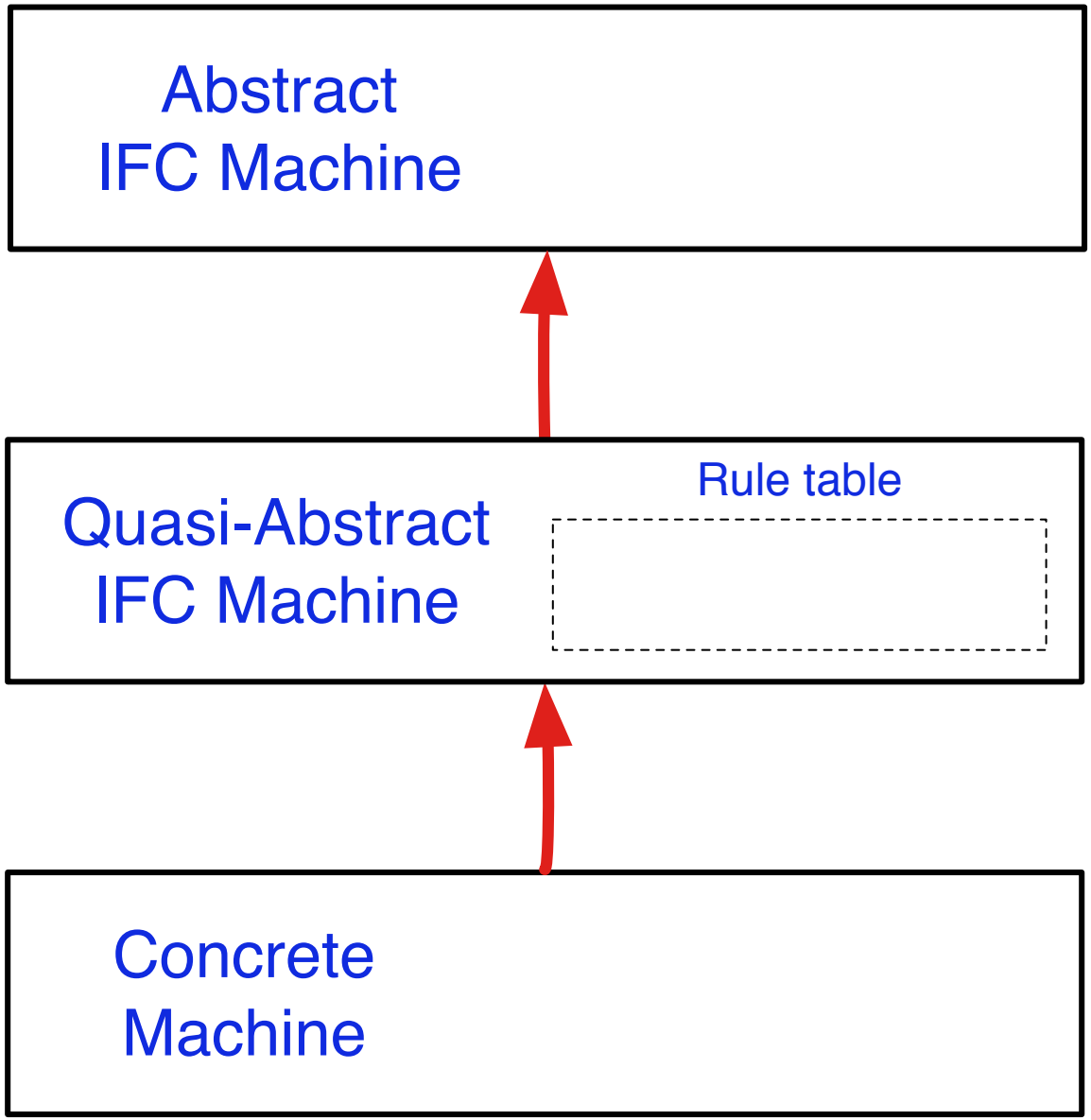
- Stack instead of registers
- No downgrading, public labels, dynamic principal generation, ...
- No exception handling
  - security violation halts the whole machine
- One-line rule cache

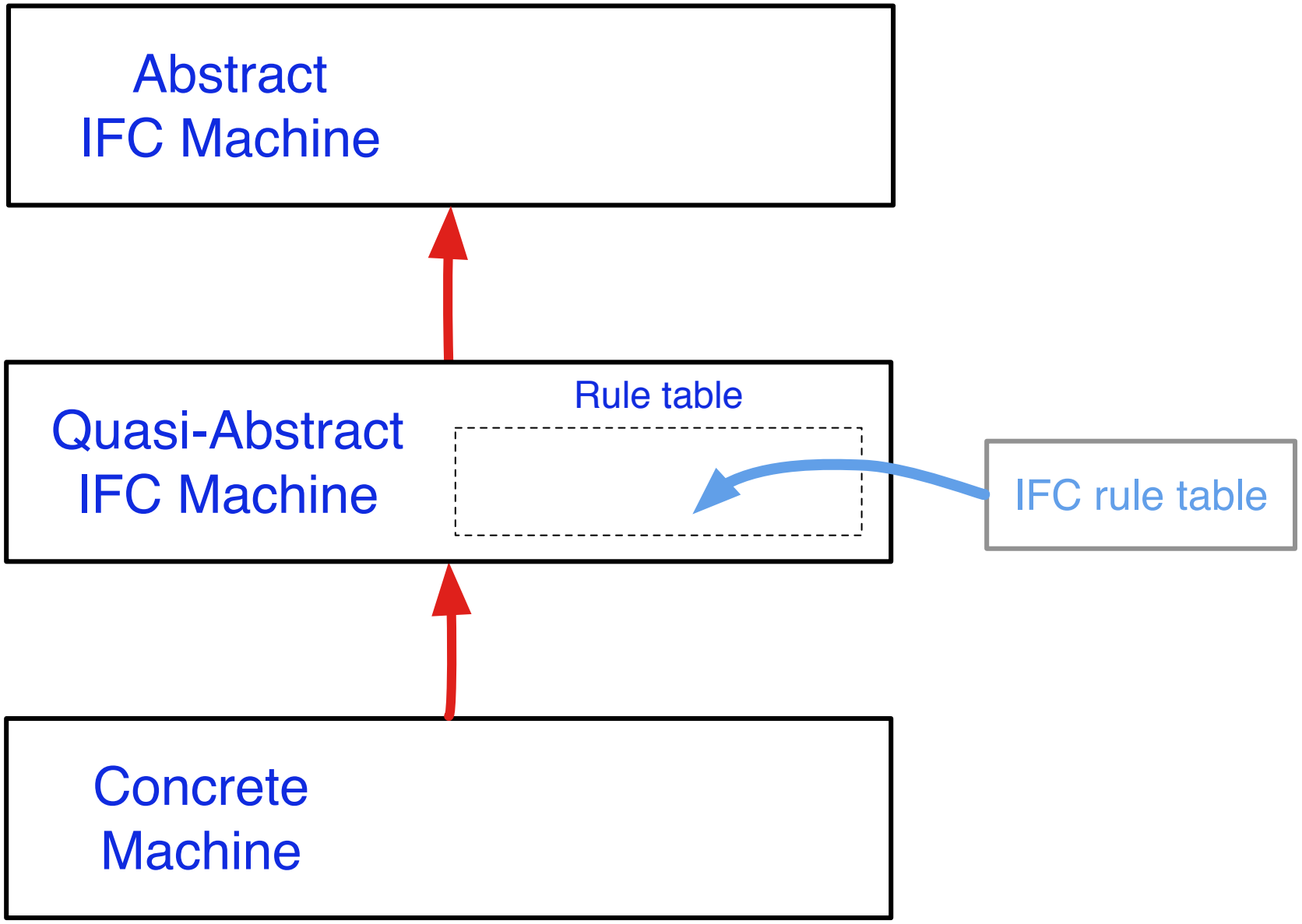
Outline

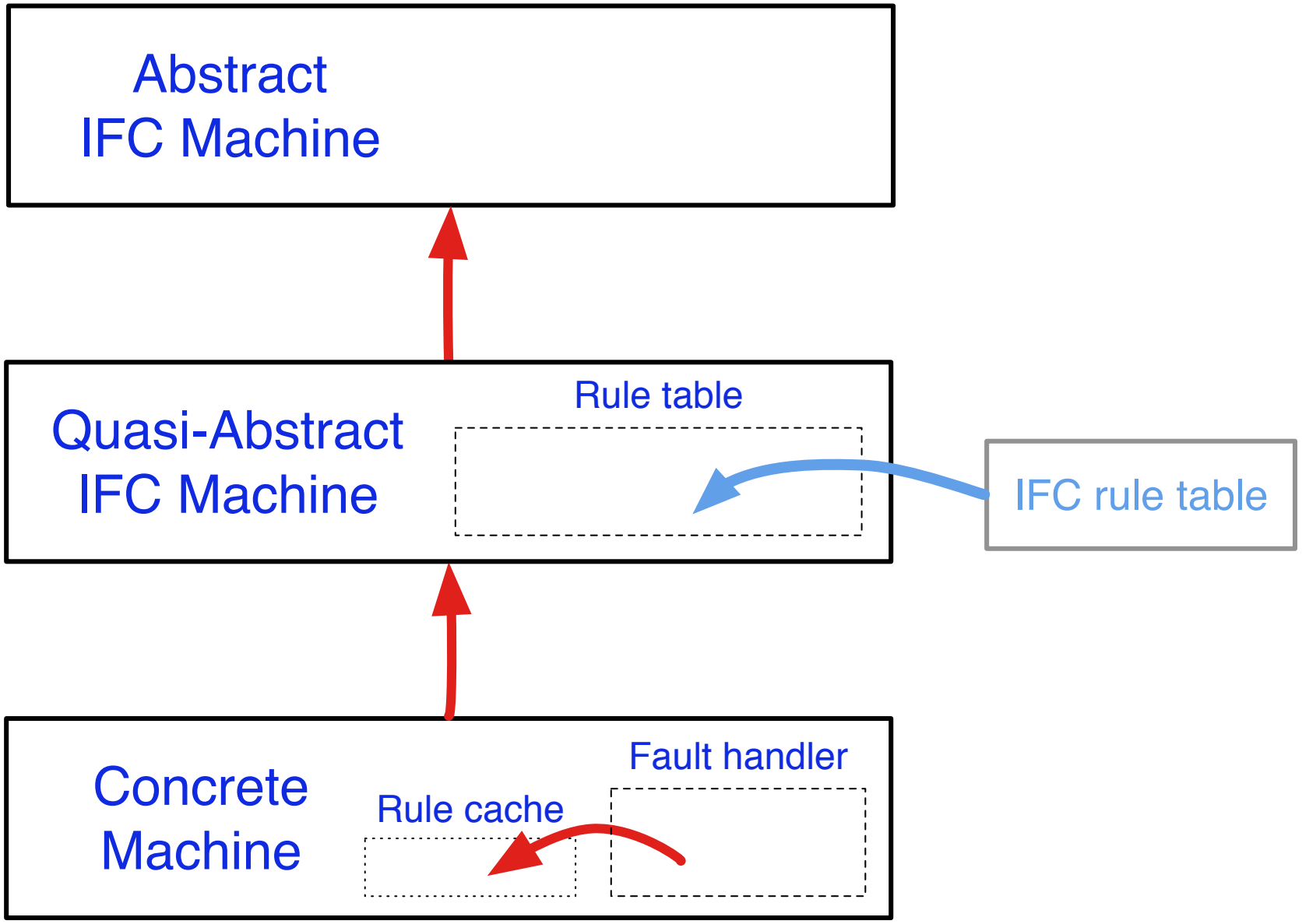
Abstract
IFC Machine

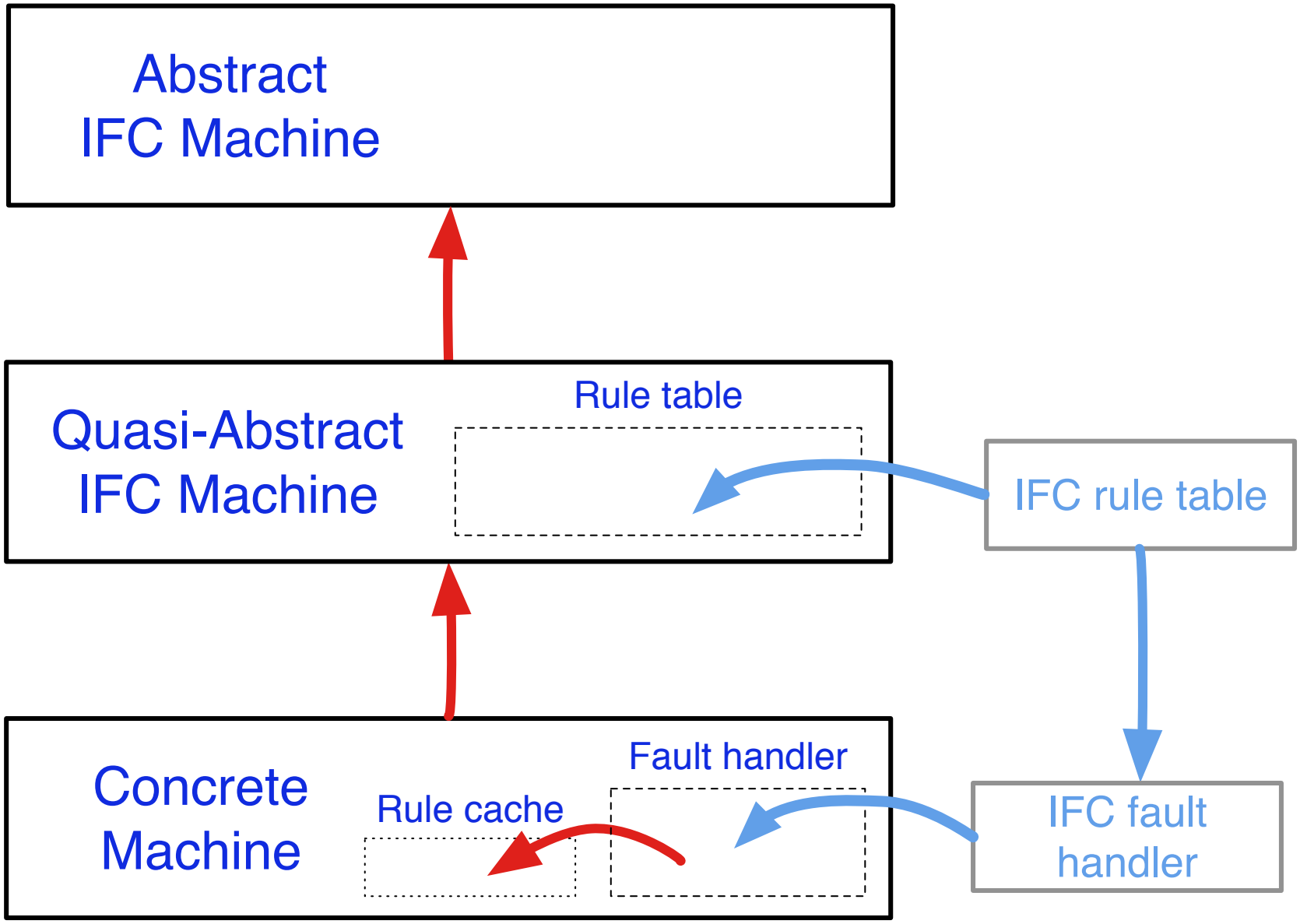

Abstract
IFC Machine
Concrete
Machine


Abstract
IFC Machine
Quasi-Abstract
IFC Machine
Concrete
Machine


Abstract
IFC Machine
Quasi-Abstract
IFC Machine
Rule table
Concrete
Machine


Abstract
IFC Machine
Quasi-Abstract
IFC Machine
Rule table
IFC rule table
Concrete
Machine


Abstract
IFC Machine
Quasi-Abstract
IFC Machine
Rule table
IFC rule table
Concrete
Machine
Fault handler
Rule cache


Abstract
IFC Machine
Quasi-Abstract
IFC Machine
Rule table
IFC rule table
Concrete
Machine
Fault handler
Rule cache
IFC fault
handler

Abstract Machine

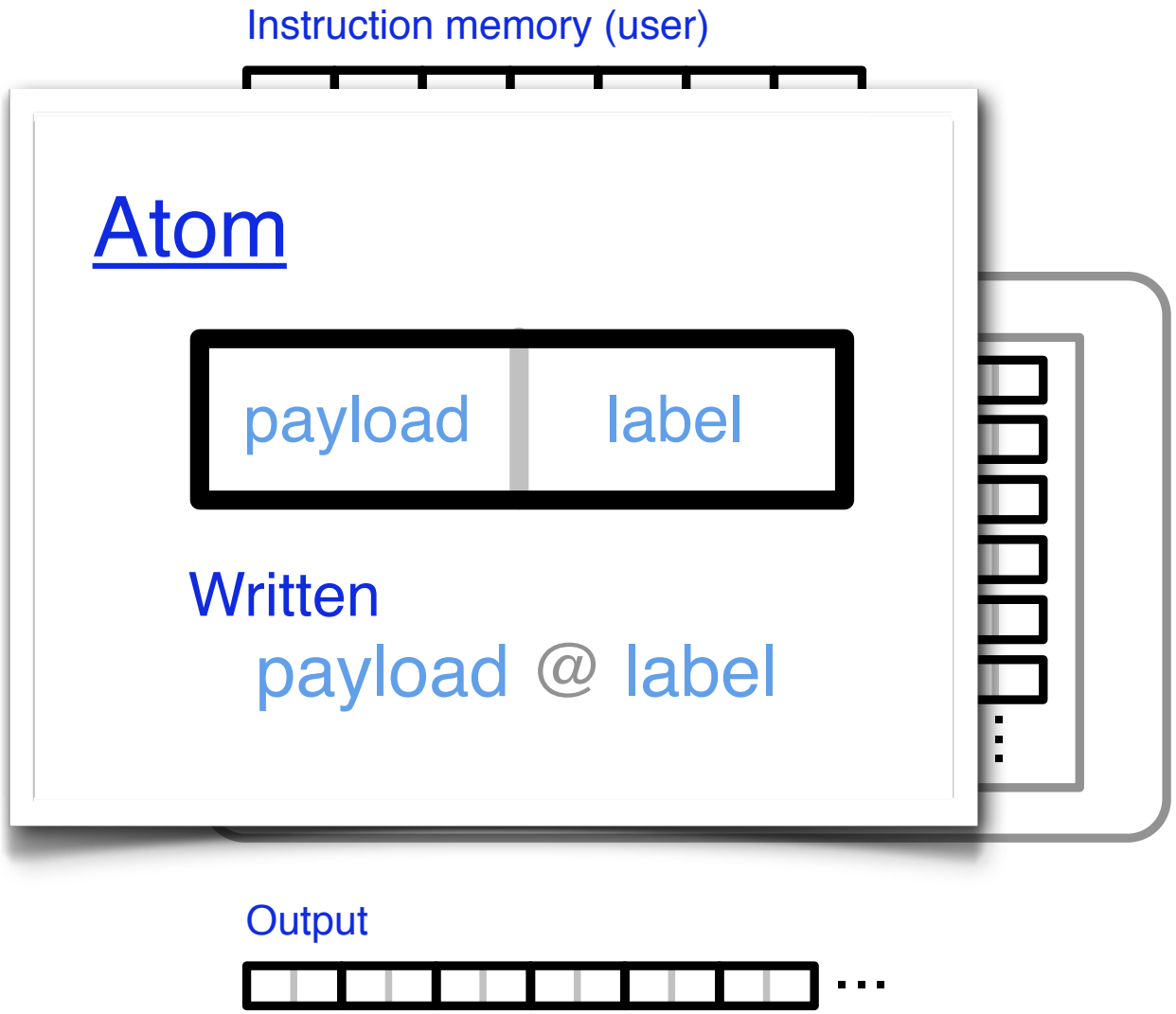
Abstract Machine

Instruction memory (user)

Machine state

$instr$	$::=$		Instructions
	|	Output	output top of stack
	|	Sub	subtract
	|	Push n	push constant integer
	|	Load	indirect load from data memory
	|	Store	indirect store to data memory
	|	Jump	unconditional indirect jump
	|	Bnz n	conditional relative jump
	|	Call	indirect call
	|	Ret	return

Abstract Machine



$$\frac{\iota(n) = \mathsf{Sub}}{\mu\ [n_1@L_1, n_2@L_2, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [(n_1 - n_2)@(L_1 \vee L_2), \sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Output}}{\mu\ [m@L_1, \sigma]\ n@L_{pc} \xrightarrow{m@L_1 \vee L_{pc}} \mu\ [\sigma]\ n{+}1@L_{pc}}$$

$$\mu_1\ [\sigma_1]\ pc_1 \xrightarrow{e} \mu_2\ [\sigma_2]\ pc_2$$

$$\frac{\iota(n) = \mathsf{Store} \qquad \mu(p) = k@L_3 \qquad L_1 \vee L_{pc} \leq L_3 \qquad \mu(p) \leftarrow (m@L_1 \vee L_2 \vee L_{pc}) = \mu'}{\mu\ [p@L_1, m@L_2, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu'\ [\sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Jump}}{\mu\ [n'@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Bnz}\ k \qquad n' = n{+}(m = 0)?1 : k}{\mu\ [m@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Call}}{\mu\ [n'@L_1, a, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [a, n{+}1@L_{pc}; \sigma]\ n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Ret}}{\mu\ [n'@L_1; \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@L_1}$$

output

memory

stack

pc

next state

$$\frac{\iota(n) = \mathsf{Sub}}{\mu \quad [n_1@L_1, n_2@L_2, \sigma] \;\; n@L_{pc} \quad \xrightarrow{\tau} \\ \mu \;\; [(n_1 - n_2)@(L_1 \vee L_2), \sigma] \;\; n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Store} \qquad \mu(p) = k@L_3 \qquad L_1 \vee L_{pc} \leq L_3 \qquad \mu(p) \leftarrow (m@L_1 \vee L_2 \vee L_{pc}) = \mu'}{\mu \; [p@L_1, m@L_2, \sigma] \; n@L_{pc} \xrightarrow{\tau} \mu' \; [\sigma] \; n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Jump}}{\mu \; [n'@L_1, \sigma] \; n@L_{pc} \xrightarrow{\tau} \mu \; [\sigma] \; n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Bnz}\; k \qquad n' = n{+}(m = 0)?1 : k}{\mu \; [m@L_1, \sigma] \; n@L_{pc} \xrightarrow{\tau} \mu \; [\sigma] \; n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Call}}{\mu \; [n'@L_1, a, \sigma] \; n@L_{pc} \xrightarrow{\tau} \mu \; [a, n{+}1@L_{pc}; \sigma] \; n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Ret}}{\mu \; [n'@L_1; \sigma] \; n@L_{pc} \xrightarrow{\tau} \mu \; [\sigma] \; n'@L_1}$$

Example

Suppose:

$$\iota = [\ldots, \mathsf{Sub}, \ldots]$$

Then:

$$\mu \quad [7@\bot, 5@\top] \quad n@\bot \quad \longrightarrow$$

$$\mu \quad [2@\top] \quad (n{+}1)@\bot$$

$$\frac{\iota(n) = \mathsf{Sub}}{\mu \quad [n_1@L_1, n_2@L_2, \sigma] \; n@L_{pc} \quad \xrightarrow{\tau}}$$

$$\frac{\iota(n) = \mathsf{Output}}{\mu\ [m@L_1, \sigma]\ n@L_{pc} \xrightarrow{m@L_1 \vee L_{pc}} \mu\ [\sigma]\ n{+}1@L_{pc}}$$

$$\frac{}{\mu\ [p@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [m@L_1 \vee L_2, \sigma]\ n{+}1@L_{pc}}$$

$$\frac{\begin{array}{l}\iota(n) = \mathsf{Store} \qquad \mu(p) = k@L_3 \qquad L_1 \vee L_{pc} \leq L_3 \\ \mu(p) \leftarrow (m@L_1 \vee L_2 \vee L_{pc}) = \mu'\end{array}}{\mu\ [p@L_1, m@L_2, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu'\ [\sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Jump}}{\mu\ [n'@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Bnz}\ k \qquad n' = n{+}(m = 0)?1 : k}{\mu\ [m@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Call}}{\mu\ [n'@L_1, a, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [a, n{+}1@L_{pc}; \sigma]\ n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Ret}}{\mu\ [n'@L_1; \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@L_1}$$

$$\frac{\iota(n) = \mathsf{Sub}}{\mu \quad [n_1@L_1, n_2@L_2, \sigma] \;\; n@L_{pc} \quad \xrightarrow{\tau} \\ \mu \;\; [(n_1 - n_2)@(L_1 \vee L_2), \sigma] \;\; n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Output}}{\mu\ [m@L_1, \sigma]\ n@L_{pc} \xrightarrow{m@L_1 \vee L_{pc}} \mu\ [\sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Push}\ m}{\mu\ [\sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [m@\bot, \sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Load} \qquad \mu(p) = m@L_2}{\mu\ [p@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [m@L_1 \vee L_2, \sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Store} \qquad \mu(p) = k@L_3 \qquad L_1 \vee L_{pc} \leq L_3 \\ \mu(p) \leftarrow (m@L_1 \vee L_2 \vee L_{pc}) = \mu'}{\mu\ [p@L_1, m@L_2, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu'\ [\sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Call}}{\mu\ [n'@L_1, a, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [a, n{+}1@L_{pc}; \sigma]\ n'@(L_1 \vee L_{pc})}$$

$$\frac{\iota(n) = \mathsf{Ret}}{\mu\ [n'@L_1; \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@L_1}$$

Quasi-Abstract Machine

QA Machine

- Alternate presentation of the abstract machine
  - Same machine states
  - Same step relation
- IFC side conditions factored out into a separate, explicit *rule table*

$$\frac{\iota(n) = \mathsf{Sub} \qquad \vdash_{\mathcal{R}} (L_{pc}, L_1, L_2, \_) \rightsquigarrow_{\mathsf{sub}} L_{rpc}, L_r}{\mu\ [n_1@L_1, n_1@L_2, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [(n_1 - n_2)@L_r, \sigma]\ (n{+}1)@L_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Outpu} \qquad \vdash_{\mathcal{R}} (L_{pc}, \_, \_, \_) \rightsquigarrow_{\mathsf{push}} L_{rpc}, L_r}{\mu\ [m@L_1, \quad n@L_r, \sigma]\ (n{+}1)@L_{rpc}}$$

$$\frac{\iota(n \quad \mu(n) = m@L_2}{\mu\ [m@L_1, \quad n@L_r, \sigma]\ (n{+}1)@L_{rpc}}$$

consult rule table...

for tags...

and opcode...

to obtain result tags...

$$\frac{\iota(n) = \mathsf{Sub} \qquad \vdash_{\mathcal{R}} (L_{pc}, L_1, L_2, \_) \rightsquigarrow_{\mathsf{sub}} L_{rpc}, L_r}{\mu\ [n_1@L_1, n_1@L_2, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [(n_1 - n_2)@L_r, \sigma]\ (n{+}1)@L_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Bnz}\ r \qquad n' = n + (m = 0) ? 1 : r \qquad \vdash_{\mathcal{R}} (L_{pc}, L_1, \_, \_) \rightsquigarrow_{\mathsf{bnz}} L_{rpc}, \_}{\mu\ [m@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@L_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Call} \qquad \vdash_{\mathcal{R}} (L_{pc}, L_1, \_, \_) \rightsquigarrow_{\mathsf{call}} L_{rpc}, L_r}{\mu\ [n'@L_1, a, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [a, (n{+}1)@L_r; \sigma]\ n'@L_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Ret} \qquad \vdash_{\mathcal{R}} (L_{pc}, L_1, \_, \_) \rightsquigarrow_{\mathsf{ret}} L_{rpc}, \_}{\mu\ [n'@L_1; \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@L_{rpc}}$$

IFC Rule Table

is this operation allowed?

new pc label

label for result

$\mathcal{R}$ =

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	$\mathtt{LAB}_1$	--

IFC Rule Table

subtraction is always allowed

pc label is unchanged

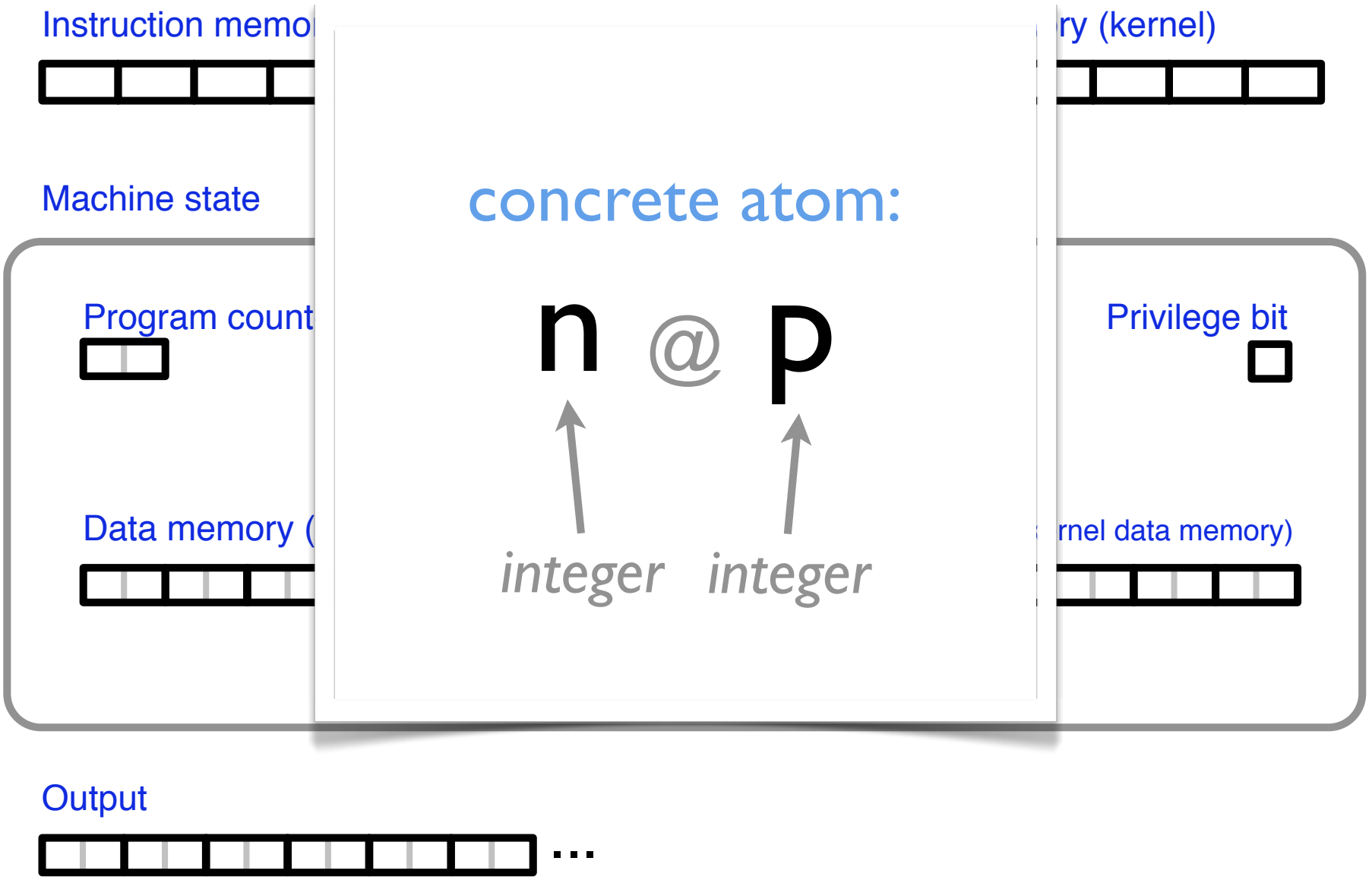
result label is join of arg labels

$\mathcal{R}$ =

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

Concrete Machine

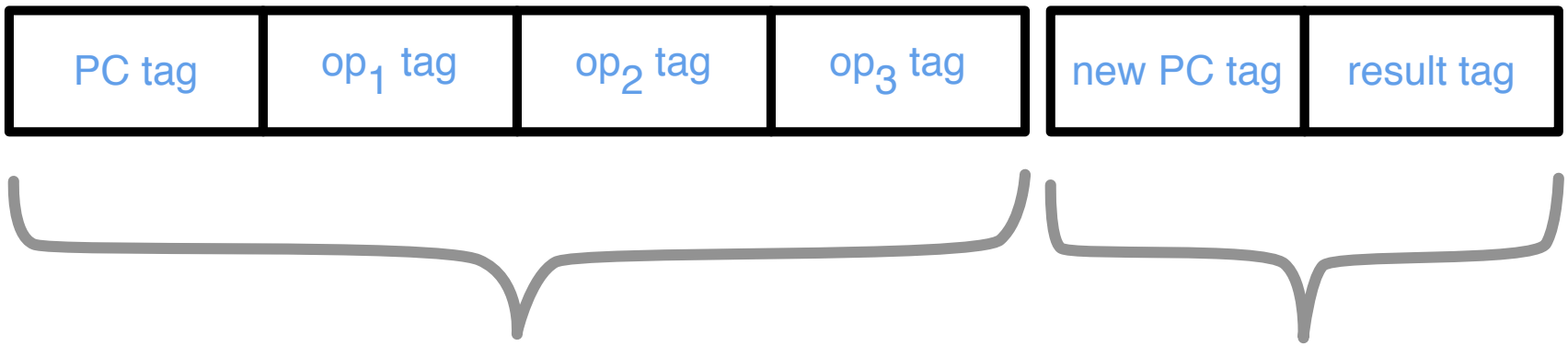
Concrete machine



Concrete machine

(single-line)

Rule cache:



Kernel mode

User mode (cache hit)

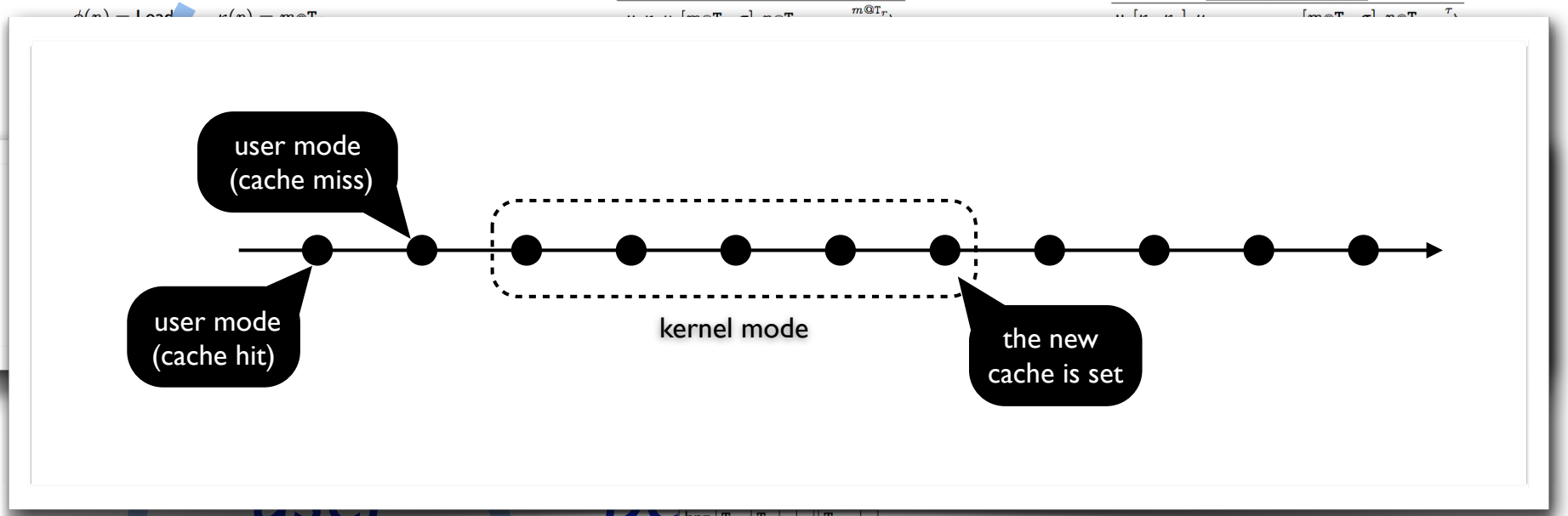
User-to-kernel mode (cache miss)

privilege bit

memory

kernel memory

stack



User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode

$$\iota(n) = \text{Sub} \qquad \kappa = \boxed{\text{sub} \mid \mathrm{T}_{pc} \mid \mathrm{T}_1 \mid \mathrm{T}_2 \mid \_ \parallel \mathrm{T}_{rpc} \mid \mathrm{T}_r}$$

$$\mathsf{u}\ \kappa\ \mu\ [n_1@\mathrm{T}_1, n_2@\mathrm{T}_2, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau}$$
$$\mathsf{u}\ \kappa\ \mu\ [(n_1 - n_2)@\mathrm{T}_r, \sigma]\ n{+}1@\mathrm{T}_{rpc}$$

User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode

$$\frac{\iota(n) = \text{Sub} \quad \kappa = \boxed{\text{sub}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \mathtt{T}_2\mid \_\mid\mid \mathtt{T}_{rpc}\mid \mathtt{T}_r}}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [n_1@\mathtt{T}_1, n_2@\mathtt{T}_2, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ [(n_1 - n_2)@\mathtt{T}_r, \sigma]\ n{+}1@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \text{Output} \quad \kappa = \boxed{\text{output}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_\mid\mid \mathtt{T}_{rpc}\mid \mathtt{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [m@\mathtt{T}_1, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{m@\mathtt{T}_r}}$$

$$\frac{\iota(n) = \text{Sub} \quad \kappa_i \neq \boxed{\text{sub}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \mathtt{T}_2\mid \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [n_1@\mathtt{T}_1, n_1@\mathtt{T}_2, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); n_1@\mathtt{T}_1, n_1@\mathtt{T}_2, \sigma]\ 0@\mathtt{T}_\_\end{array}}$$

$$\frac{\iota(n) = \text{Output} \quad \kappa_i \neq \boxed{\text{output}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_} = \kappa_j}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [n_1@\mathtt{T}_1, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau}}$$

$$\frac{\phi(n) = \text{Sub}}{\mathsf{k}\ \kappa\ \mu\ [n_1@\_, n_1@\_, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [(n_1 - n_2)@\mathtt{T}\_, \sigma]\ n{+}1@\mathtt{T}\_}$$

$$\frac{\phi(n) = \text{Push}\ m}{\mathsf{k}\ \kappa\ \mu\ [\sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [m@\mathtt{T}\_, \sigma]\ n{+}1@\mathtt{T}\_}$$

$$\phi(n) = \text{Load} \qquad \kappa(p) = m@\mathtt{T}_1$$

$$\frac{\iota(n) = \text{Sub} \quad \kappa_i \neq \boxed{\text{sub}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \mathtt{T}_2\mid \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [n_1@\mathtt{T}_1, n_2@\mathtt{T}_2, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); n_1@\mathtt{T}_1, n_2@\mathtt{T}_2, \sigma]\ 0@\mathtt{T}_\_\end{array}}$$

$$\frac{\iota(n) = \text{Jump}\ k \quad \kappa = \boxed{\text{jump}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_\mid\mid \mathtt{T}_{rpc}\mid \_}}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [n'@\mathtt{T}_1, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \text{Jump}\ k \quad \kappa_i \neq \boxed{\text{jump}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [n'@\mathtt{T}_1, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); n'@\mathtt{T}_1, \sigma]\ 0@\mathtt{T}_\_\end{array}}$$

$$\frac{\iota(n) = \text{Bnz}\ k \quad \kappa = \boxed{\text{bnz}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_\mid\mid \mathtt{T}_{rpc}\mid \_} \quad n' = n{+}(m = 0)?1 : k}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [m@\mathtt{T}_1, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \text{Bnz}\ k \quad \kappa_i \neq \boxed{\text{bnz}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [m@\mathtt{T}_1, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); m@\mathtt{T}_1, \sigma]\ 0@\mathtt{T}_\_\end{array}}$$

$$\frac{\iota(n) = \text{Call} \quad \kappa = \boxed{\text{call}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_\mid\mid \mathtt{T}_{rpc}\mid \mathtt{T}_r}}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [n'@\mathtt{T}_1, a, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ [a, (n{+}1@\mathtt{T}_r, \mathsf{u}); \sigma]\ n'@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \text{Call} \quad \kappa_i \neq \boxed{\text{call}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [n'@\mathtt{T}_1, a, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); n'@\mathtt{T}_1, a, \sigma]\ 0@\mathtt{T}_\_\end{array}}$$

$$\frac{\iota(n) = \text{Ret} \quad \kappa = \boxed{\text{ret}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_\mid\mid \mathtt{T}_{rpc}\mid \_}}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [(n'@\mathtt{T}_1, \mathsf{u}); \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \text{Ret} \quad \kappa_i \neq \boxed{\text{ret}\mid \mathtt{T}_{pc}\mid \mathtt{T}_1\mid \_\mid \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [(n'@\mathtt{T}_1, \pi); \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); (n'@\mathtt{T}_1, \pi); \sigma]\ 0@\mathtt{T}_\_\end{array}}$$

User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode

$$\frac{\phi(n) = \mathsf{Sub}}{\mathsf{k}\ \kappa\ \mu\ \ [n_1@\_, n_2@\_, \sigma]\ \ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [(n_1 - n_2)@\mathtt{T}\_, \sigma]\ \ n{+}1@\mathtt{T}\_}$$

$$\frac{\iota(n) = \mathsf{Sub} \quad \kappa = \boxed{\mathsf{sub}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \mathtt{T}_2\ |\ \_\ ||\ \mathtt{T}_{rpc}\ |\ \mathtt{T}_r}}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [n_1@\mathtt{T}_1, n_2@\mathtt{T}_2, \sigma]\ \ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ [(n_1 - n_2)@\mathtt{T}_r, \sigma]\ \ n{+}1@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \mathsf{Output} \quad \kappa = \boxed{\mathsf{output}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \_\ |\ \_\ ||\ \mathtt{T}_{rpc}\ |\ \mathtt{T}_r}}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [m@\mathtt{T}_1, \sigma]\ \ n@\mathtt{T}_{pc} \xrightarrow{m@\mathtt{T}_r} \\ \mathsf{u}\ \kappa\ \mu\ \ [\sigma]\ \ n{+}1@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \mathsf{Bnz}\,k \quad \kappa = \boxed{\mathsf{bnz}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \_\ |\ \_\ ||\ \mathtt{T}_{rpc}\ |\ \_} \quad n' = n{+}(m = 0)?1 : k}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [m@\mathtt{T}_1, \sigma]\ \ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ \ [\sigma]\ \ n'@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \mathsf{Call} \quad \kappa = \boxed{\mathsf{call}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \_\ |\ \_\ ||\ \mathtt{T}_{rpc}\ |\ \mathtt{T}_r}}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ \ [n'@\mathtt{T}_1, a, \sigma]\ \ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ [a, (n{+}1@\mathtt{T}_r, \mathsf{u}); \sigma]\ \ n'@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \mathsf{Ret} \quad \kappa = \boxed{\mathsf{ret}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \_\ |\ \_\ ||\ \mathtt{T}_{rpc}\ |\ \_}}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [(n'@\mathtt{T}_1, \mathsf{u}); \sigma]\ \ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ \ [\sigma]\ \ n'@\mathtt{T}_{rpc}\end{array}}$$

$$\frac{\iota(n) = \mathsf{Sub} \quad \kappa_i \neq \boxed{\mathsf{sub}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \mathtt{T}_2\ |\ \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ \ [n_1@\mathtt{T}_1, n_1@\mathtt{T}_2, \sigma]\ \ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); n_1@\mathtt{T}_1, n_1@\mathtt{T}_2, \sigma]\ \ 0@\mathtt{T}\_\end{array}}$$

$$\frac{\iota(n) = \mathsf{Output} \quad \kappa_i \neq \boxed{\mathsf{output}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \_\ |\ \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ \ [m@\mathtt{T}_1, \sigma]\ \ \_@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); m@\mathtt{T}_1, \sigma]\ \ 0@\mathtt{T}\_\end{array}}$$

$$\frac{\iota(n) = \mathsf{Bnz}\,k \quad \kappa_i \neq \boxed{\mathsf{bnz}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \_\ |\ \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ \ [m@\mathtt{T}_1, \sigma]\ \ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); m@\mathtt{T}_1, \sigma]\ \ 0@\mathtt{T}\_\end{array}}$$

$$\frac{\iota(n) = \mathsf{Call} \quad \kappa_i \neq \boxed{\mathsf{call}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \_\ |\ \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ \ [n'@\mathtt{T}_1, a, \sigma]\ \ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); n'@\mathtt{T}_1, a, \sigma]\ \ 0@\mathtt{T}\_\end{array}}$$

$$\frac{\iota(n) = \mathsf{Ret} \quad \kappa_i \neq \boxed{\mathsf{ret}\ |\ \mathtt{T}_{pc}\ |\ \mathtt{T}_1\ |\ \_\ |\ \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ \ [(n'@\mathtt{T}_1, \pi); \sigma]\ \ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); (n'@\mathtt{T}_1, \pi); \sigma]\ \ 0@\mathtt{T}\_\end{array}}$$

$$\frac{\phi(n) = \mathsf{Sub}}{\mathsf{k}\ \kappa\ \mu\ \ [n_1@\_, n_1@\_, \sigma]\ \ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [(n_1 - n_2)@\mathtt{T}\_, \sigma]\ \ n{+}1@\mathtt{T}\_}$$

$$\frac{\phi(n) = \mathsf{Push}\ m}{\mathsf{k}\ \kappa\ \mu\ [\sigma]\ \ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [m@\mathtt{T}\_, \sigma]\ \ n{+}1@\mathtt{T}\_}$$

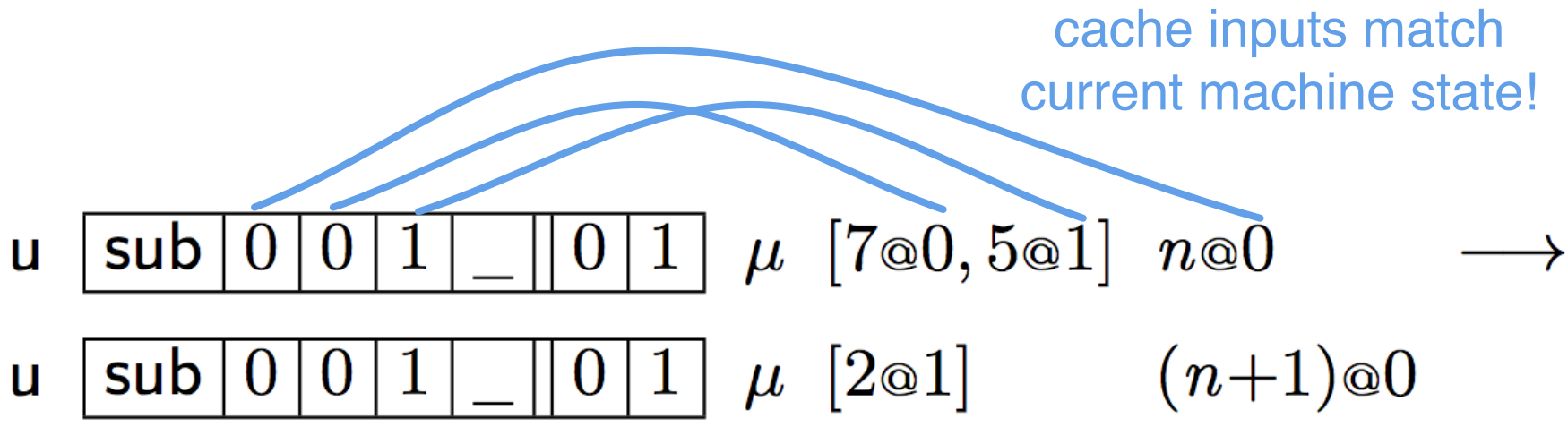
$$\frac{\phi(n) = \mathsf{Load} \qquad \kappa(p) = m@\mathtt{T}_1}{\ldots}$$

Example (cache hit case)

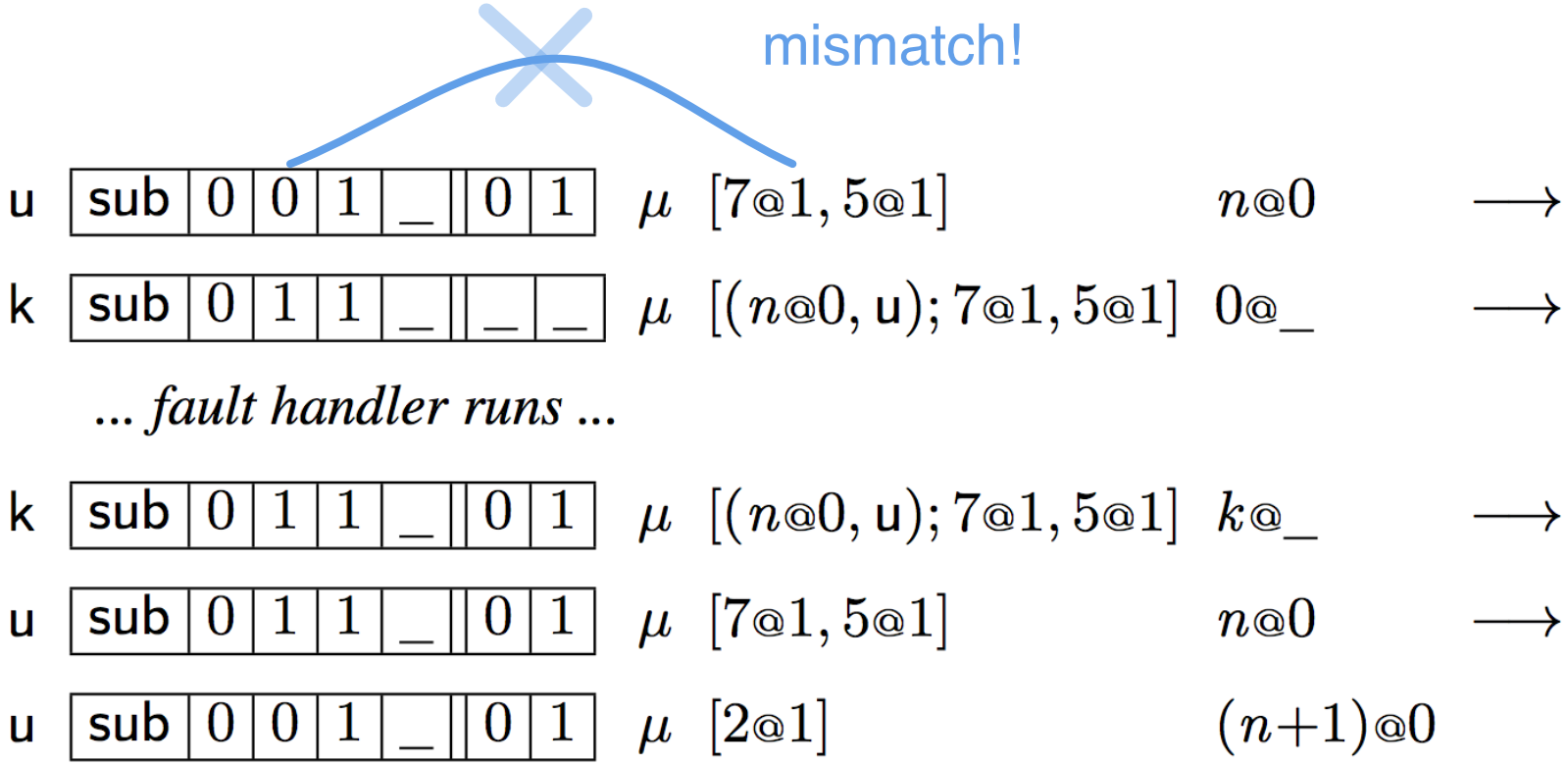
Suppose

tag 0 represents label $\bot$

tag 1 represents label $\top$



Example (cache miss case)



Fault Handler

IFC Rule Table

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

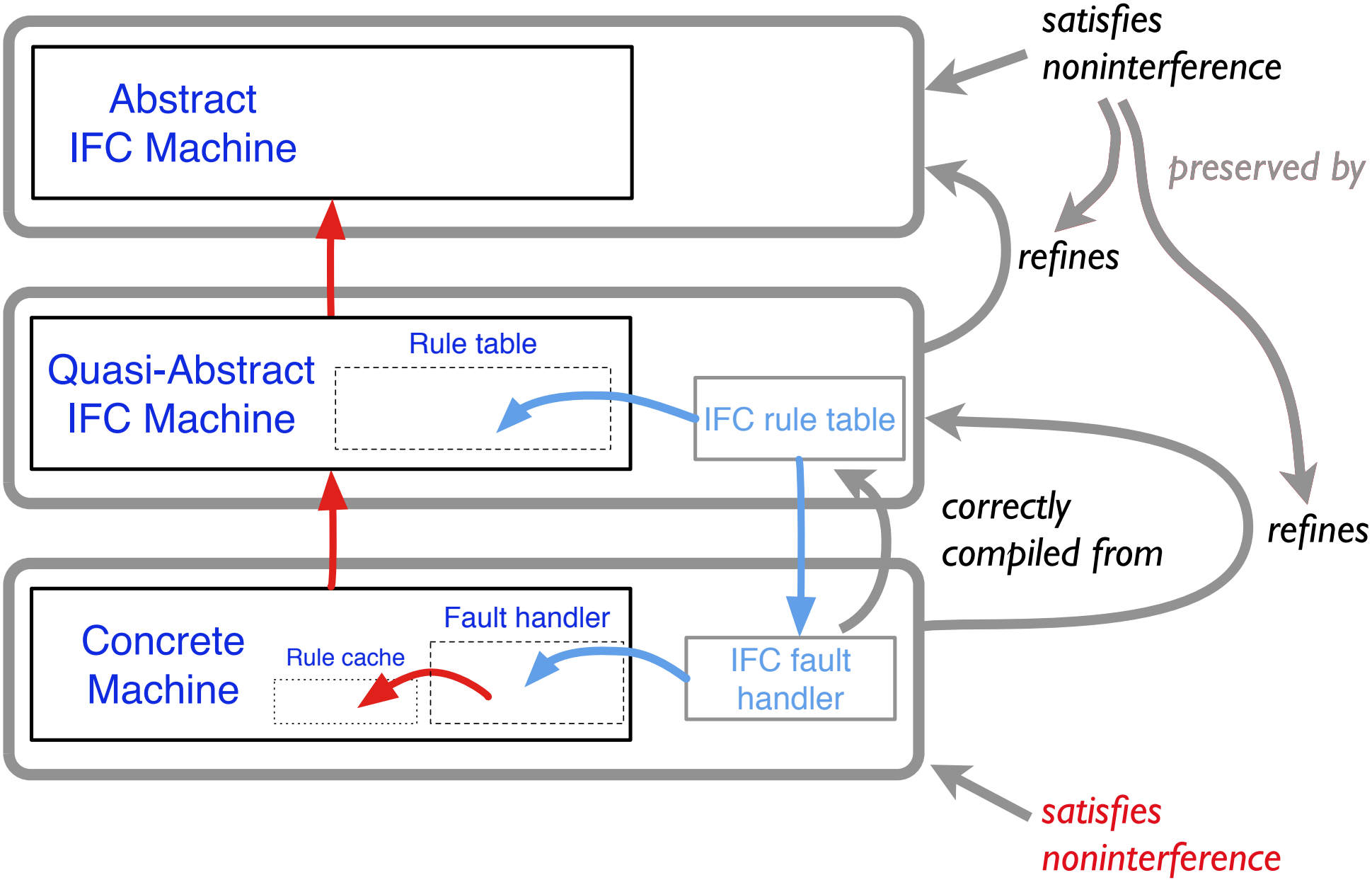
Fault Handler

Properties

Approach

- Goal:
  - Termination-Insensitive Noninterference
    - for concrete machine
    - running this particular fault handler
    - together with arbitrary user code
- Direct brute-force proof?
  - hopelessly complex
  - unmaintainable
- What kind of structure do we need?

The one we've already got!


Abstract
IFC Machine
Quasi-Abstract
IFC Machine
Rule table
IFC rule table
Concrete
Machine
Fault handler
Rule cache
IFC fault
handler
satisfies
noninterference
preserved by
refines
refines
correctly
compiled from
satisfies
noninterference

Points to note

- Refinement framework *very* useful for reasoning
  - start with concrete object
  - propose abstracted version
    - incorporate convenient structure and annotations
  - prove refinement
  - prove interesting property of abstract object
  - automatically follows for concrete object

Points to note

- Need a *generic* notion of noninterference that makes sense for all machines

Definition 10.2 (TINI). A machine $(S, E, I, \cdot \dot{\rightarrow} \cdot, \mathit{Init})$ with a notion of observation $(\Omega, \lfloor \cdot \rfloor_\cdot, \cdot \approx_\cdot \cdot)$ satisfies *termination-insensitive noninterference* (TINI) if, for any observer $o \in \Omega$, pair of indistinguishable initial data $i_1 \approx_o i_2$, and pair of executions $\mathit{Init}(i_1) \xrightarrow{t_1}{}^\star$ and $\mathit{Init}(i_2) \xrightarrow{t_2}{}^\star$, we have $\lfloor t_1 \rfloor_o \approx_o \lfloor t_2 \rfloor_o$.

Some Challenges...

More uses for tags

- SAFE architecture is quite generic
  - Can be used to implement a range of IFC label models just by varying the rule table [see Montagu's CSF 2013 talk!]
- Other potential uses
  - access control (clearance)
  - memory protection
  - linearity
  - dynamic typing

Downgrading

- Essential in real IFC systems
  - *declassification* needed to prevent "label creep"
  - *endorsement* needed to make integrity labels interesting
- Plain noninterference a useful first step, but we want to say something about programs with downgrading
- Should we change to some form of intransitive noninterference? Or something else?
  - *many* possibilities
  - which one do we want?

"Least privilege"

- Real SAFE operating system uses hardware protection mechanisms to organize functionality into mutually suspicious compartments
  - Goal: No "omniprivileged" compartment
  - *Zero-Kernel Operating System (ZKOS)*
- What does this mean, formally? What can/should we prove? What is the appropriate attack model?
- More generally, what would a formal account of "least privilege" look like?

Concurrency

- Real machines / operating systems support many user processes (including SAFE)
- Processes can interact (via streams, in SAFE)
- But communicating threads can leak secrets
  - with high bandwidth, if threads can be created at will
- LIO solution: Never lower PC label
  - "If a thread goes high, it stays high"
  - To operate on secret data without poisoning the PC, fork a new thread
  - Looking at result message from high thread makes receiver high (but it can store it in a data structure without looking)
  - Possible issues:
    - Spawning many many many threads
    - turning everything into futures
- Is this the best we can do???

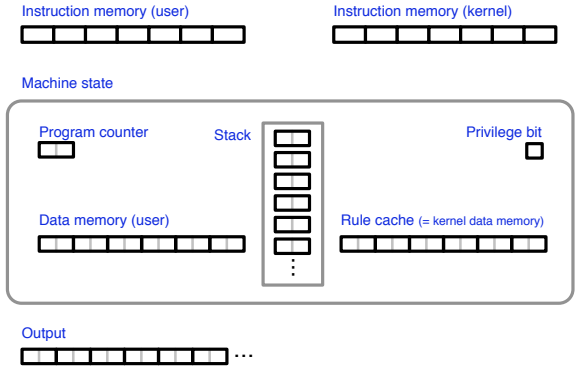
Status

Done

- Definitions and proofs from this talk fully formalized (in Coq)
  - Draft paper nearly done — email me if you'd like a copy
- Full SAFE hardware implemented (in Bluespec) and running (on an FPGA)
  - Pipelined version under construction
- Simple versions of key OS services working in isolation
  - process management and scheduling
  - storage allocation
  - rule cache management

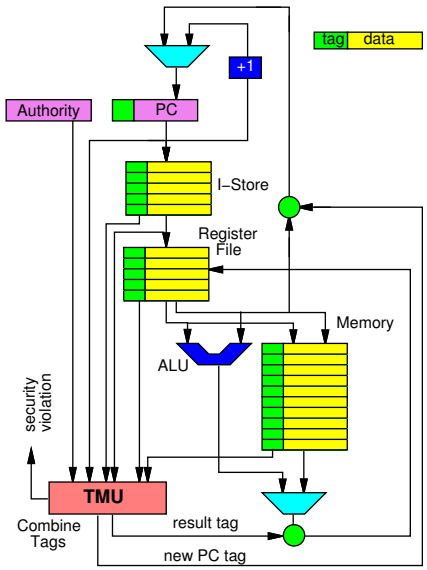
In progress

- Integration of system components into end-to-end demo
- Compilers from high-level languages to SAFE machine code
- Random testing of noninterference
  - see our upcoming ICFP 2013 paper!
- Formal specification and noninterference proofs for larger fragments of full SAFE machine
  - storage allocation
  - public labels
  - dynamic principal generation



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	$\mathtt{LAB}_1$	--

Thank you!



Questions??

$$\frac{\iota(n) = \mathsf{Sub}}{\mu \quad [n_1@L_1, n_2@L_2, \sigma] \quad n@L_{pc} \xrightarrow{\tau} \mu \quad [(n_1 - n_2)@(L_1 \vee L_2), \sigma] \quad n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Output}}{\mu\ [m@L_1, \sigma]\ n@L_{pc} \xrightarrow{m@L_1 \vee L_{pc}} \mu\ [\sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Push}\ m}{\mu\ [\sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [m@\bot, \sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Load} \qquad \mu(p) = m@L_2}{\mu\ [p@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [m@L_1 \vee L_2, \sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Store} \qquad \mu(p) = k@L_3 \qquad L_1 \vee L_{pc} \leq L_3 \qquad \mu(p) \leftarrow (m@L_1 \vee L_2 \vee L_{pc}) = \mu'}{\mu\ [p@L_1, m@L_2, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu'\ [\sigma]\ n{+}1@L_{pc}}$$

$$\frac{\iota(n) = \mathsf{Jump}}{\mu\ [n'@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [\sigma]\ n'@(L_1 \vee L_{pc})}$$