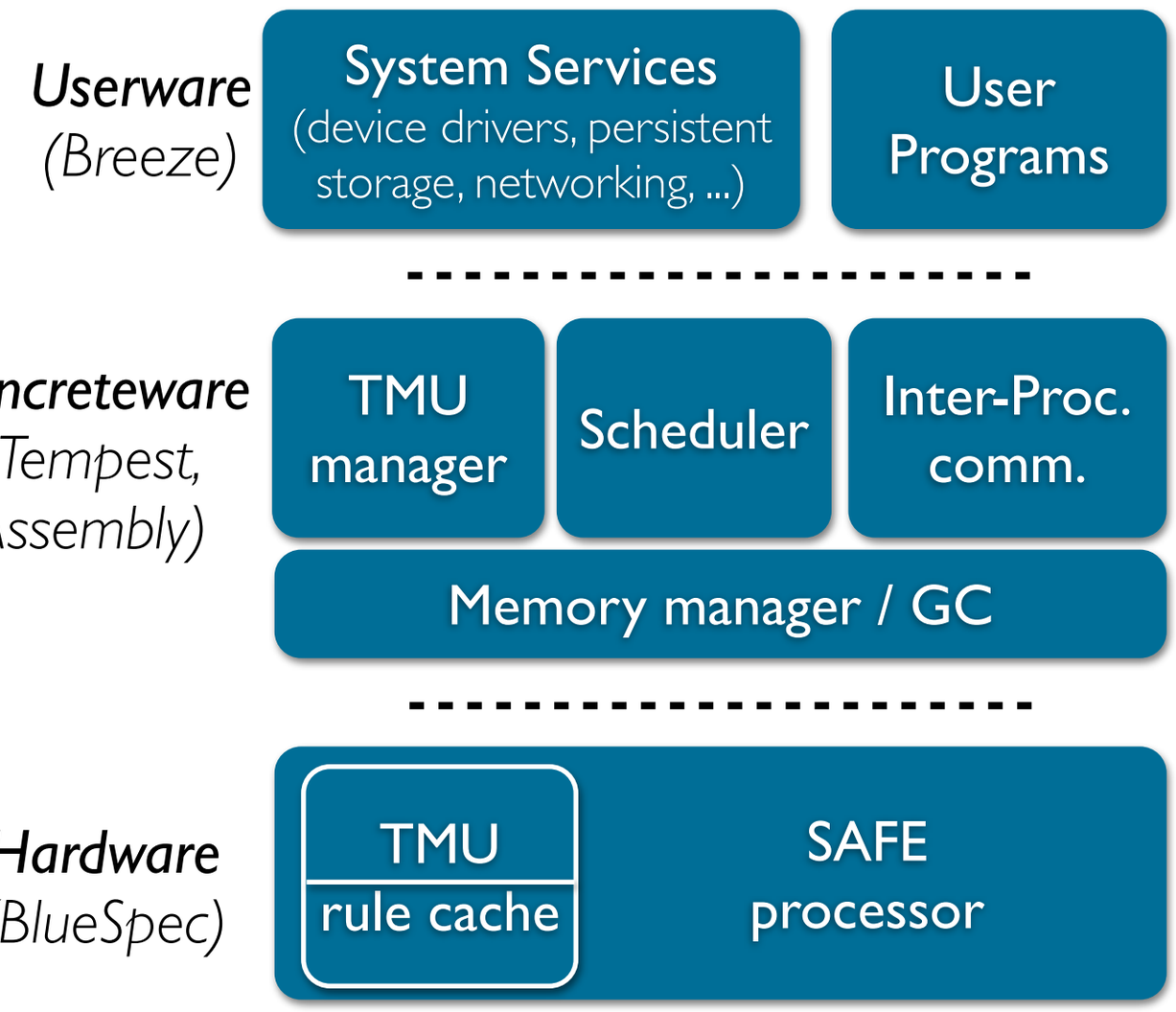
A Verified Information-Flow Architecture for SAFE

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

LTP - Bordeaux - 18 Nov. 2013

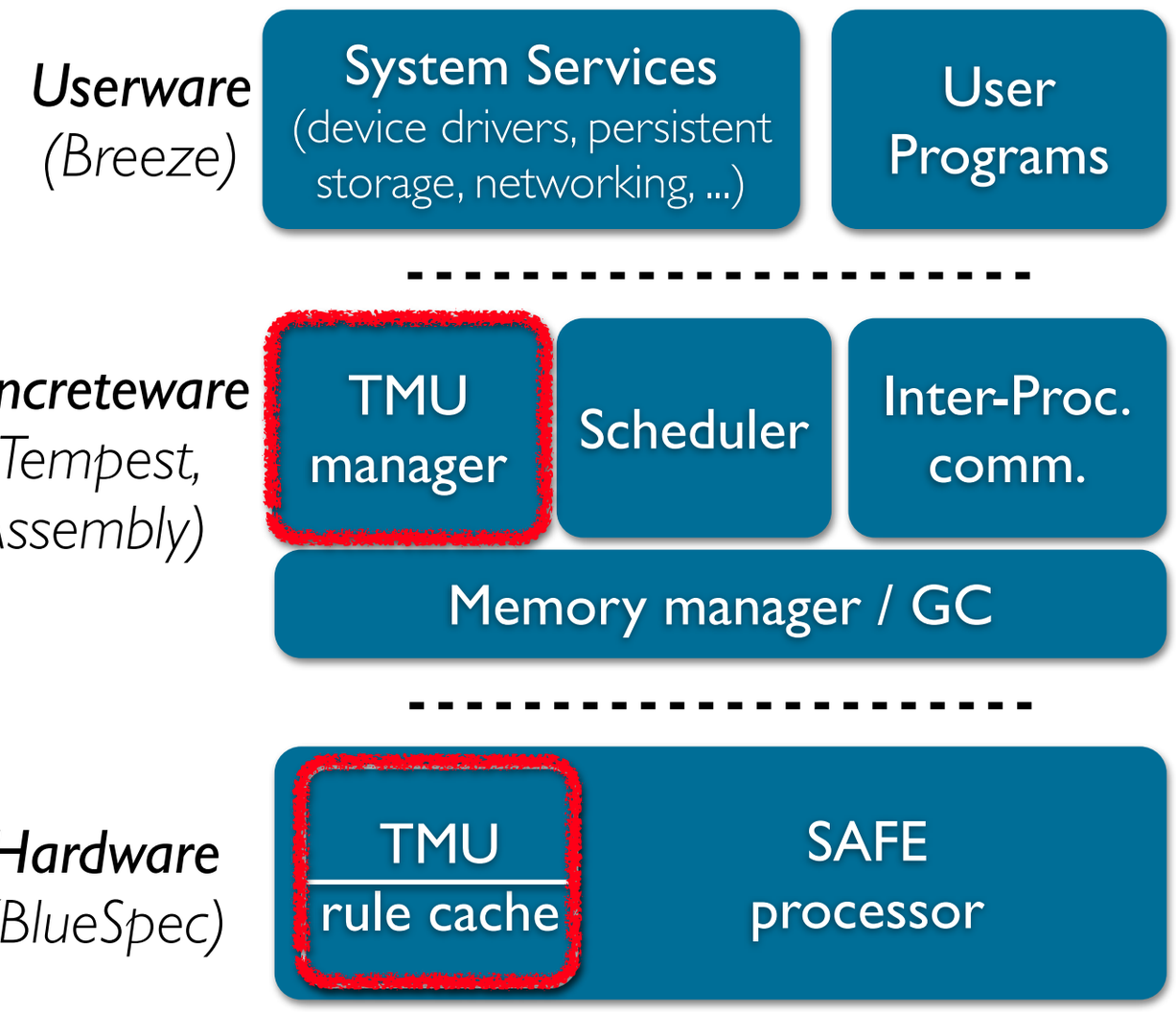
Crash/SAFE

- Clean-slate redesign of entire system stack
  - Hardware, system software, programming languages
- Critical security primitives at all levels
  - Memory safety, strong dynamic typing, information flow and access control
- Verification of key mechanisms
  - deeply integrated into design process



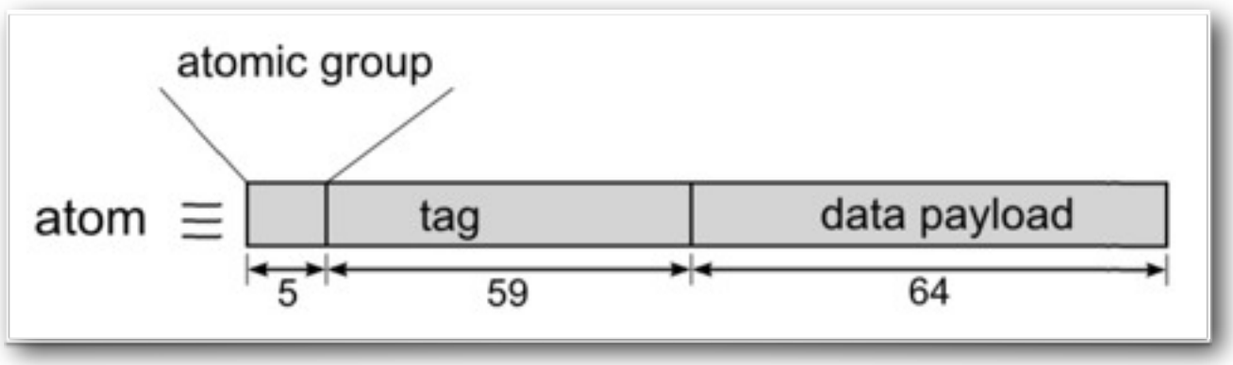
Crash/SAFE

- Clean-slate redesign of entire system stack
  - Hardware, system software, programming languages
- Critical security primitives at all levels
  - Memory safety, strong dynamic typing, information flow and access control
- Verification of key mechanisms
  - deeply integrated into design process



SAFE hardware (glimpse)

- Hardware types ("atomic groups")
  - instruction ≠ integer ≠ pointer
- Hardware tagging
  - *atom* = payload + atomic group + software-defined tag

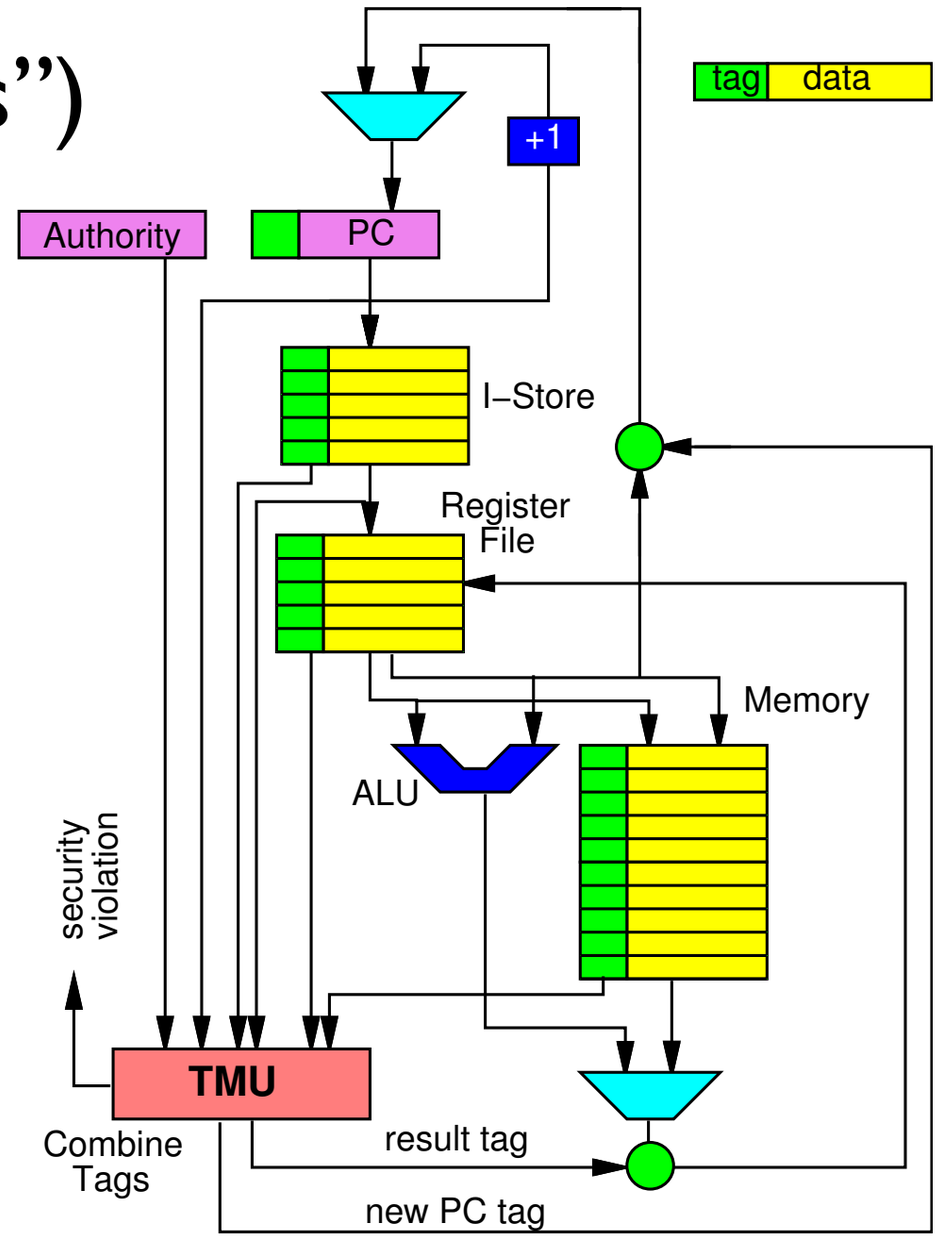


- Hardware *rule cache*: tag propagation with every machine step

SAFE hardware (glimpse)

- Hardware types ("atomic groups")
  - instruction ≠ integer ≠ pointer
- Hardware tagging
  - *atom* = payload + atomic group + software-defined tag

- Hardware *rule cache*: tag propagation with every machine step



This talk: formalizing SAFE's hardware tags for information flow control

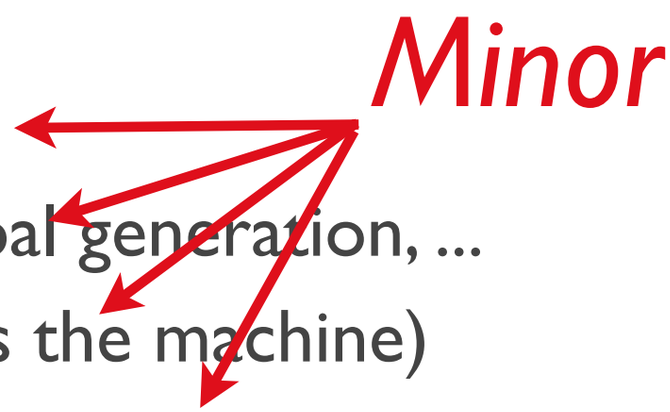
- Abstraction supported by HW tags
  - labels over a IFC lattice
  - dynamic non-interference
- We formally state and verify these properties
  - (hardware + TMU manager) ≈ abstract machine

- Simplifications
  - deterministic, single-threaded, stack machine
  - no downgrading, public labels, dynamic principal generation, ...
  - no exception handling (security violation halts the machine)
  - single-line rule cache

This talk: formalizing SAFE's hardware tags for information flow control

- Abstraction supported by HW tags
  - labels over a IFC lattice
  - dynamic non-interference
- We formally state and verify these properties

  - (hardware + TMU manager) ≈ abstract machine
- Simplifications
  - deterministic, single-threaded, stack machine
  - no downgrading, public labels, dynamic principal generation, ...
  - no exception handling (security violation halts the machine)
  - single-line rule cache



This talk: formalizing SAFE's hardware tags for information flow control

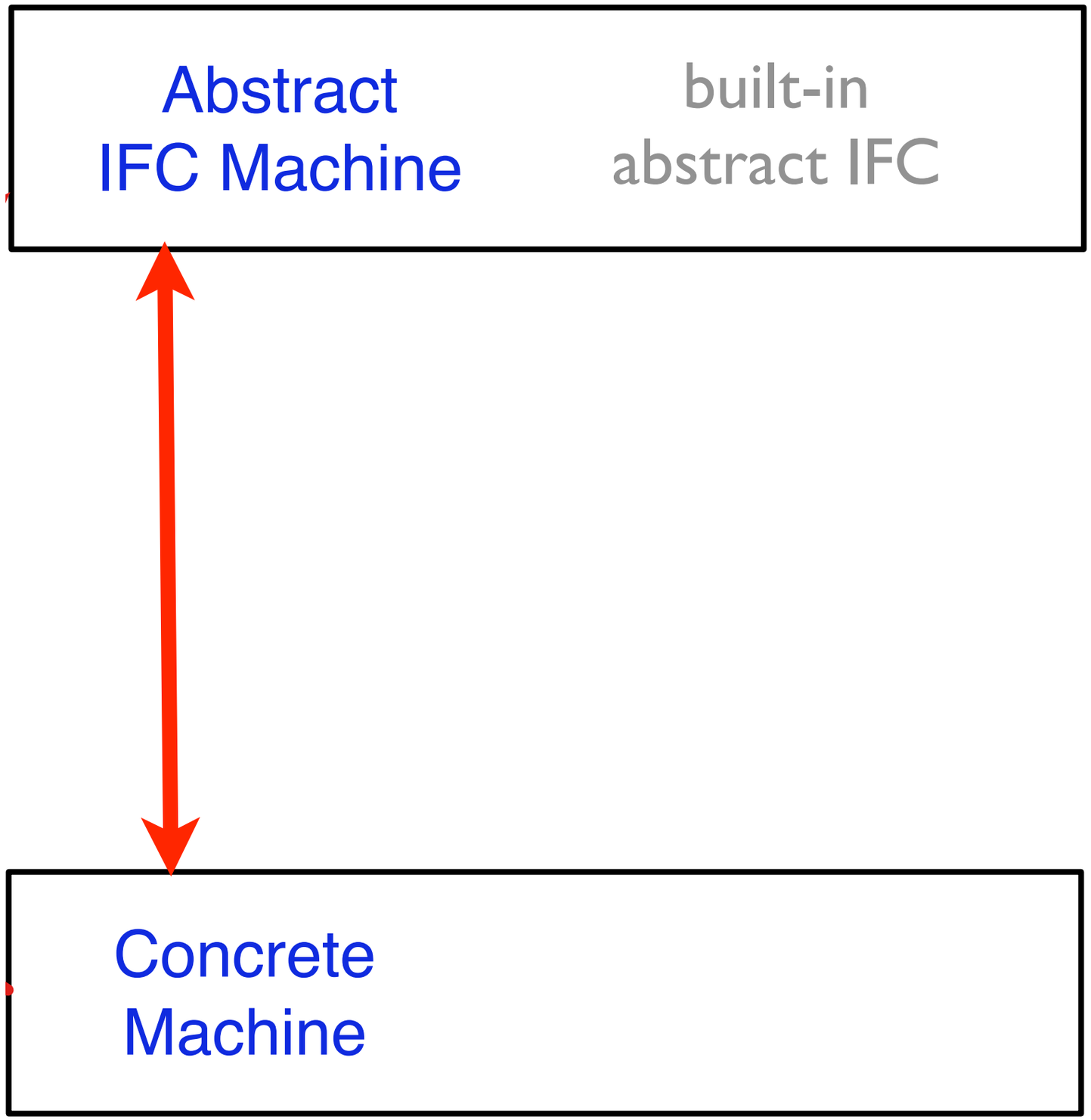
- Abstraction supported by HW tags
  - labels over a IFC lattice
  - dynamic non-interference
- We formally state and verify these properties
  - (hardware + TMU manager) ≈ abstract machine

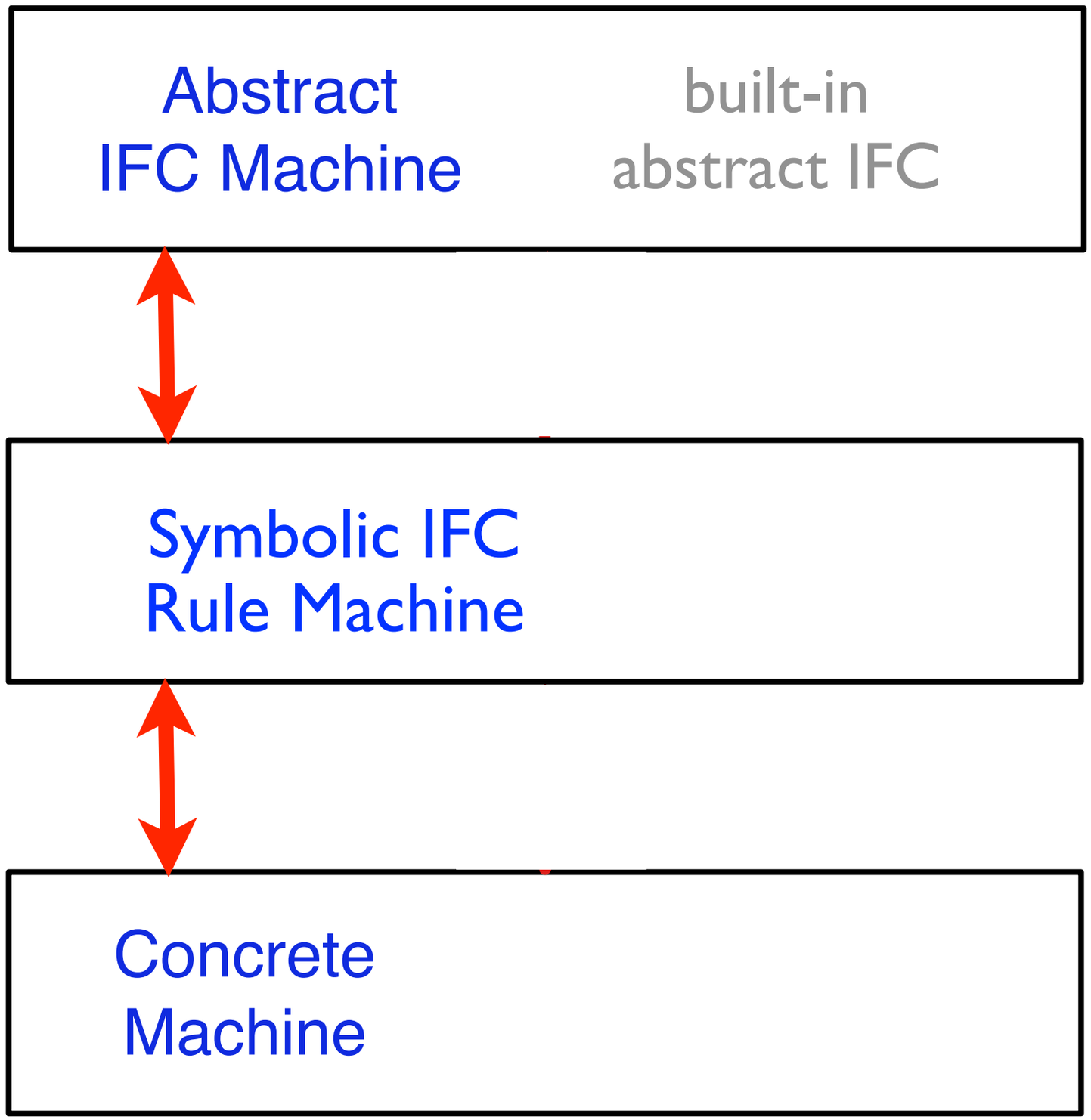
- Simplifications
  - deterministic, single-threaded, stack machine
  - no downgrading, public labels, dynamic principal generation, ...
  - no exception handling (security violation halts the machine)
  - single-line rule cache

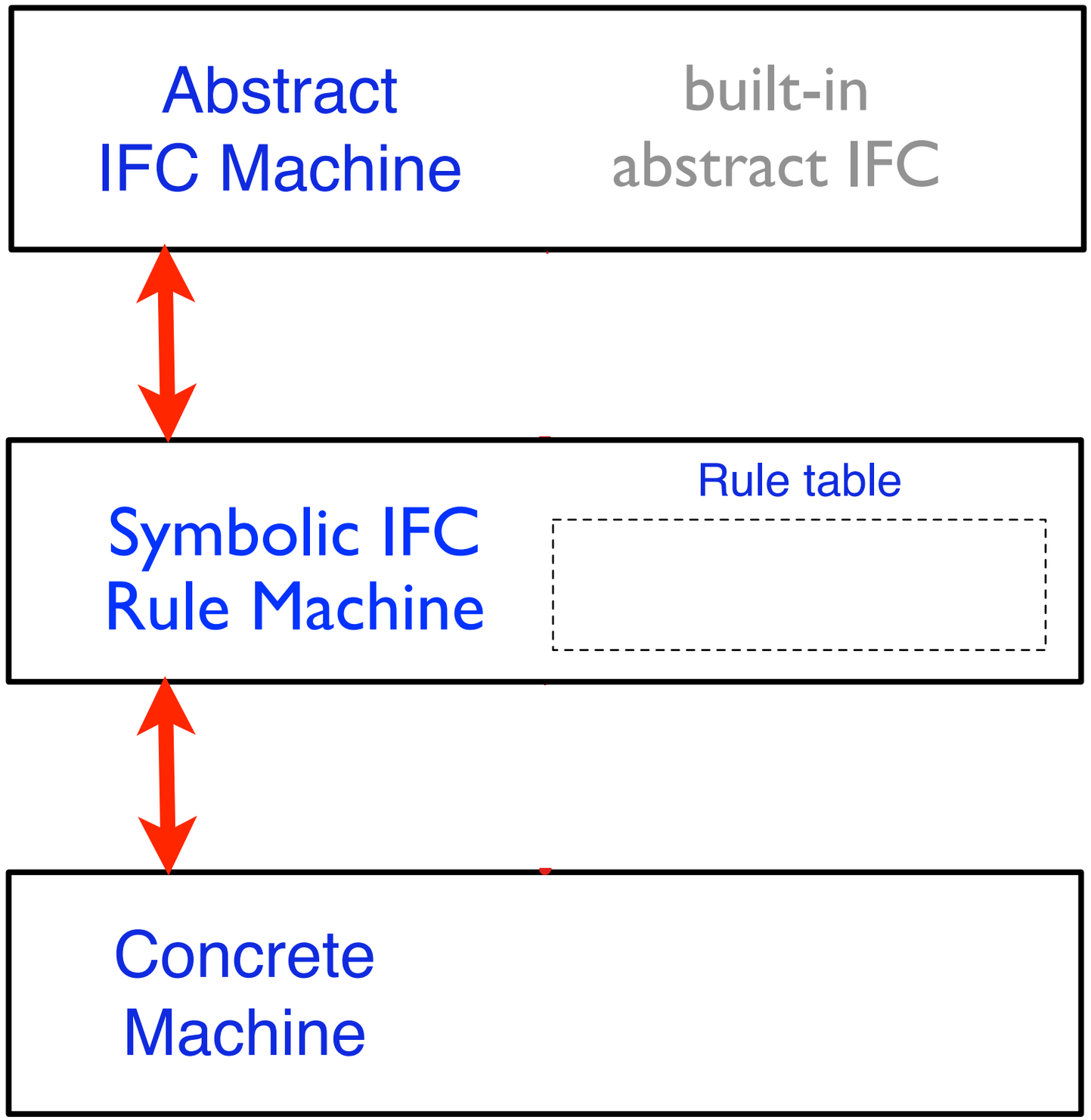
Major

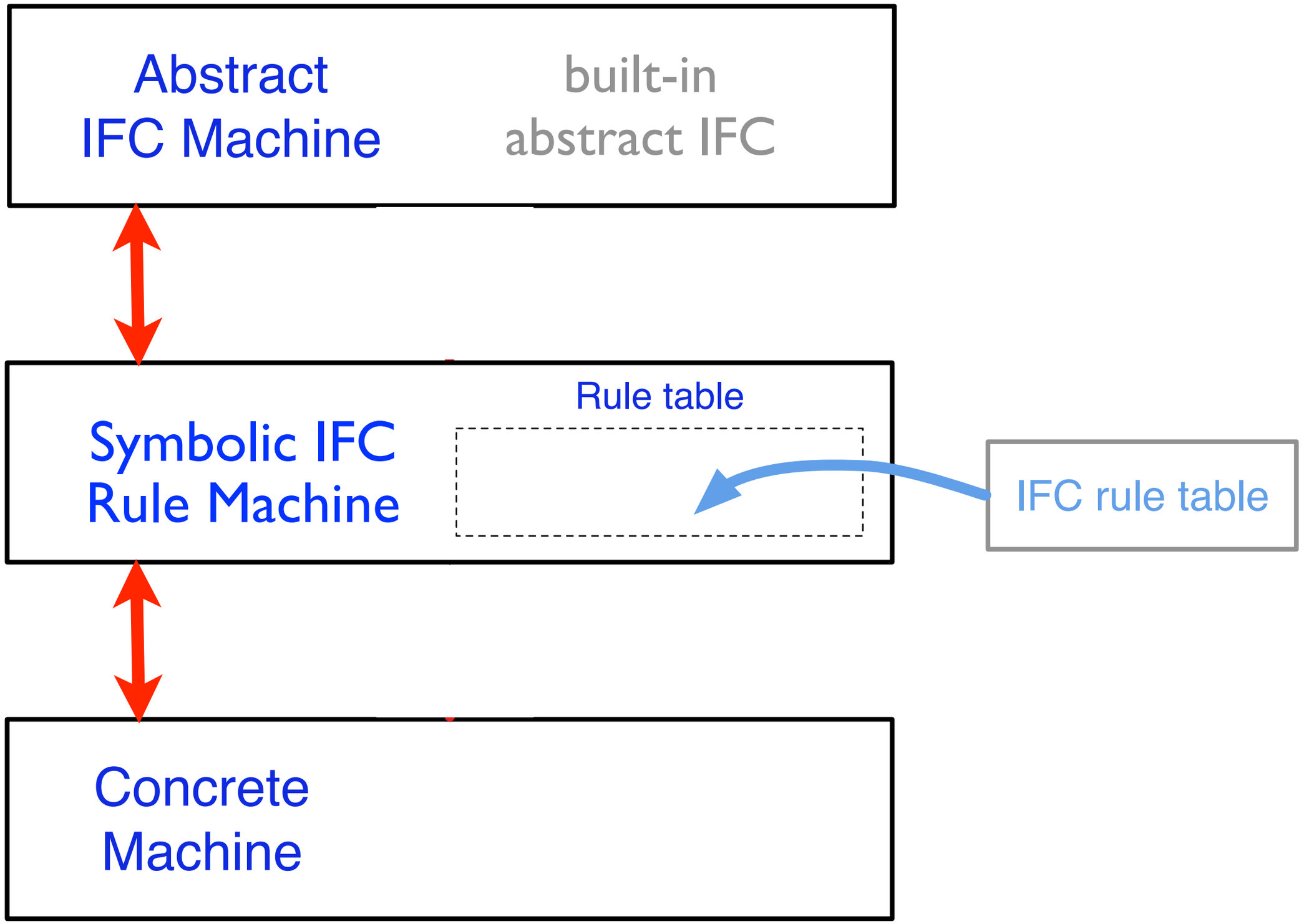
Minor

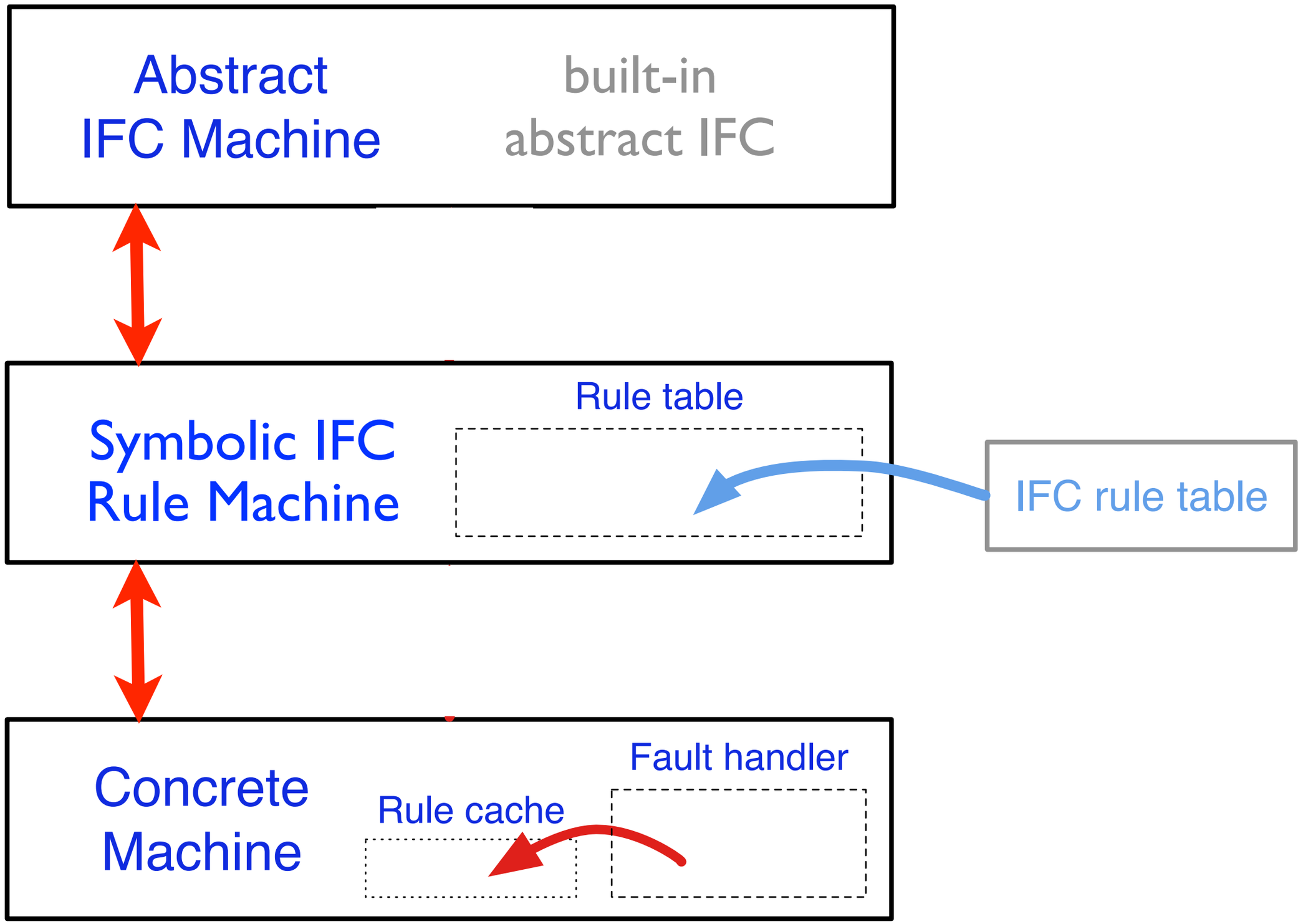
Concrete
Machine

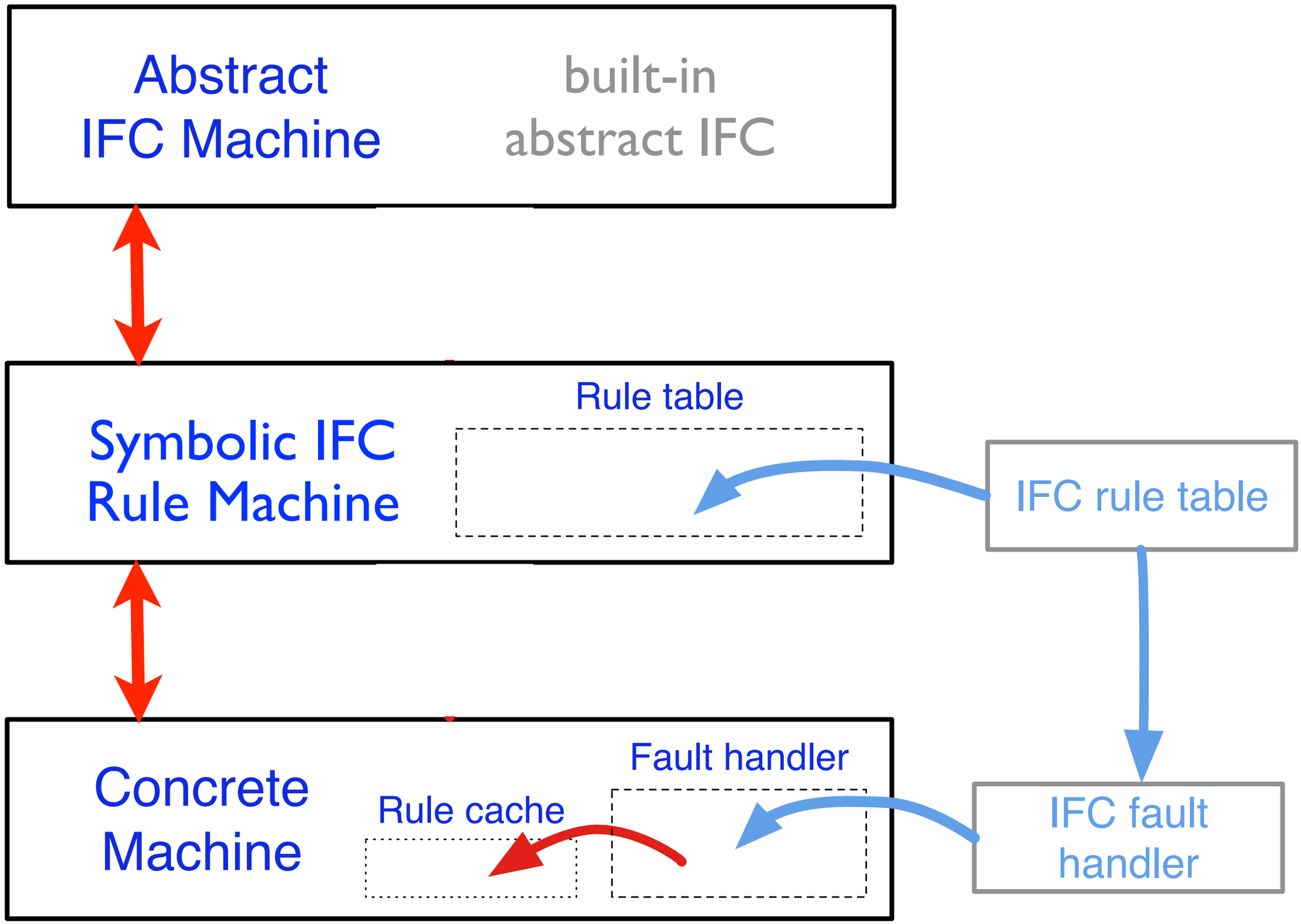

Abstract
IFC Machine
built-in
abstract IFC
Concrete
Machine


Abstract
IFC Machine
built-in
abstract IFC
Symbolic IFC
Rule Machine
Concrete
Machine


Abstract
IFC Machine
built-in
abstract IFC
Rule table
Symbolic IFC
Rule Machine
Concrete
Machine


Abstract
IFC Machine
built-in
abstract IFC
Rule table
Symbolic IFC
Rule Machine
IFC rule table
Concrete
Machine


Abstract
IFC Machine
built-in
abstract IFC
Rule table
Symbolic IFC
Rule Machine
IFC rule table
Concrete
Machine
Fault handler
Rule cache

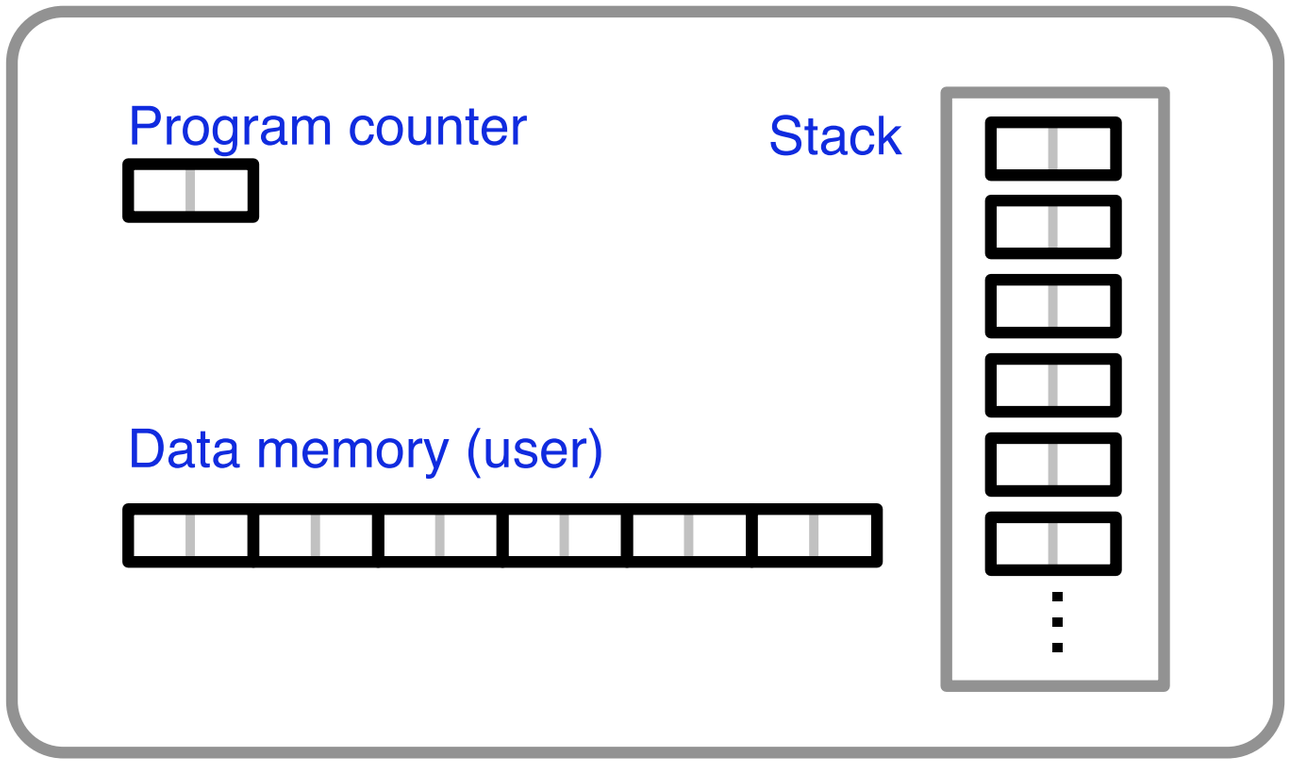

Abstract
IFC Machine
built-in
abstract IFC
Symbolic IFC
Rule Machine
Rule table
IFC rule table
Concrete
Machine
Rule cache
Fault handler
IFC fault
handler

Abstract Machine

Abstract Machine

Instruction memory (user)

Machine state



Output

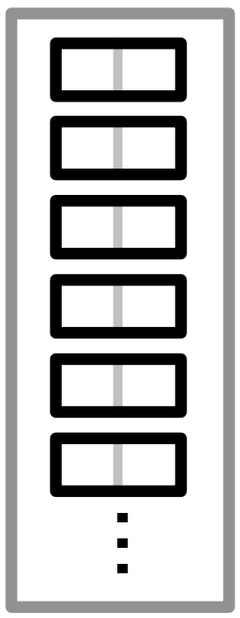
Abstract Machine

Instruction memory (user)

Machine state

Program counter

Stack



Data memory (user)

Output

Atom

Written

payload @ label

Abstract Machine

Instruction memory (user)

Mach

Pr

Da

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

Output

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

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

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

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

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

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

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

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

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

output

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

memory stack pc

next state

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

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

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

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

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

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

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

output

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

memory stack pc

next state

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

output

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

memory stack pc

next state

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

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

output

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

memory stack pc

next state

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

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

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

Non-interference

Non-interference

- We design the abstract machine so that it is easy to prove a non-interference property

Non-interference

- We design the abstract machine so that it is easy to prove a non-interference property
- Roughly: "high" inputs cannot affect "low" outputs.
  - If two executions of a program start with same "low" data, the "low" parts of output traces will be the same

Non-interference

- We design the abstract machine so that it is easy to prove a non-interference property
- Roughly: "high" inputs cannot affect "low" outputs.
  - If two executions of a program start with same "low" data, the "low" parts of output traces will be the same

$$\left.\begin{array}{ll} \forall obs, & \\ & (P, in) \sim_{obs} (P, in') \\ \wedge & (P, in) \Downarrow_{obs} T \\ \wedge & (P, in') \Downarrow_{obs} T' \end{array}\right\} \Rightarrow T \simeq T'$$

Non-interference

- We design the abstract machine so that it is easy to prove a non-interference property
- Roughly: "high" inputs cannot affect "low" outputs.
  - If two executions of a program start with same "low" data, the "low" parts of output traces will be the same

$$\forall obs, \left.\begin{array}{l}(P, in) \sim_{obs} (P, in') \\ (P, in) \Downarrow_{obs} T \\ (P, in') \Downarrow_{obs} T'\end{array}\right\} \Rightarrow T \simeq T'$$

same instructions, but different starting secrets (unknown to observer)

Non-interference

- We design the abstract machine so that it is easy to prove a non-interference property
- Roughly: "high" inputs cannot affect "low" outputs.
  - If two executions of a program start with same "low" data, the "low" parts of output traces will be the same

$$\forall obs,\quad \left.\begin{array}{l}(P, in) \sim_{obs} (P, in') \\ (P, in) \Downarrow_{obs} T \\ (P, in') \Downarrow_{obs} T'\end{array}\right\} \Rightarrow T \simeq T'$$

same instructions, but different starting secrets (unknown to observer)

pointwise equality
cropping to shortest trace
(termination insensitive)

Non-interference

- We design the abstract machine so that it is easy to prove a non-interference property
- Roughly: "high" inputs cannot affect "low" outputs.
  - If two executions of a program start with same "low" data, the "low" parts of output traces will be the same

$$\forall obs,\quad \left.\begin{array}{l}(P, in) \sim_{obs} (P, in') \\ (P, in) \Downarrow_{obs} T \\ (P, in') \Downarrow_{obs} T'\end{array}\right\} \Rightarrow T \simeq T'$$

same instructions, but different starting secrets (unknown to observer)

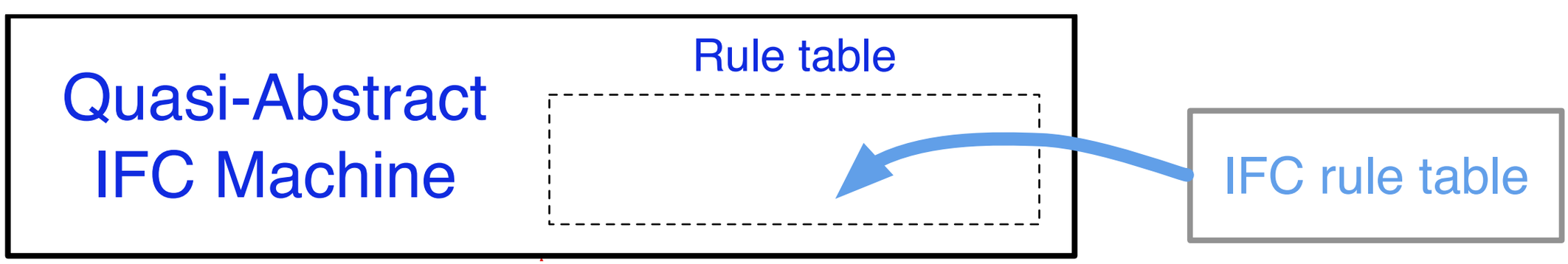
pointwise equality
cropping to shortest trace
(termination insensitive)

standard assumptions: attacker cannot distinguish successful termination, failing with an error, and unproductive infinite loop

Symbolic IFC Rule Machine

Symbolic IFC Rule Machine

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



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

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

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

$$\frac{\begin{array}{c}\iota(n) = \mathsf{Load} \qquad \mu(p) = m@L_2 \\ \vdash_{\mathcal{R}} (L_{pc}, L_1, L_2, \_) \rightsquigarrow_{\mathsf{load}} L_{rpc}, L_r\end{array}}{\mu\ [p@L_1, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu\ [m@L_r, \sigma]\ (n{+}1)@L_{rpc}}$$

$$\frac{\begin{array}{c}\iota(n) = \mathsf{Store} \qquad \mu(p) = k@L_3 \\ \vdash_{\mathcal{R}} (L_{pc}, L_1, L_2, L_3) \rightsquigarrow_{\mathsf{store}} L_{rpc}, L_r \\ \mu(p) \leftarrow m@L_r = \mu'\end{array}}{\mu\ [p@L_1, m@L_2, \sigma]\ n@L_{pc} \xrightarrow{\tau} \mu'\ [\sigma]\ (n{+}1)@L_{rpc}}$$

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

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

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

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

consult rule table...

for tags...

and opcode...

to obtain result tags...

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

consult rule table...

for tags...

and opcode...

to obtain result tags...

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

consult rule table...

for tags...

and opcode...

to obtain result tags...

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

IFC Rule Table

$\mathcal{R}$ =

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

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

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	$\mathtt{LAB}_{pc}$	--
store	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_{pc} \sqsubseteq$ LA	$\mathtt{LAB}_1 \sqcup \mathtt{LAB}_{pc}$	$\mathtt{LAB}_{pc}$
jump	TRUE	$\mathtt{LAB}_1$	--

subtraction is always allowed

pc label is unchanged

result label is join of arg labels

Concrete Machine

Concrete machine

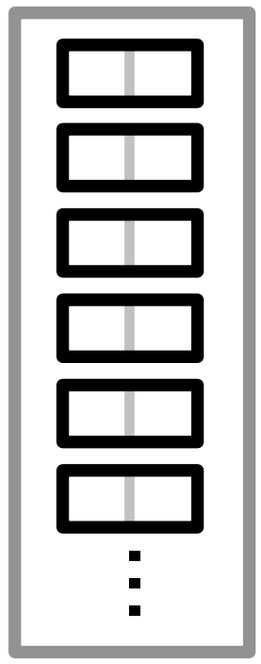
Instruction memory (user)

Instruction memory (kernel)

Machine state

Program counter

Stack



Privilege bit

Data memory (user)

Rule cache (= kernel data memory)

Output

Concrete machine

Instruction memory (user)

Instruction memory (kernel)

Machine state

Program counte

Privilege bit

Data memory (u

nel data memory)

concrete atom:

n @ p

integer integer

Output

...

Concrete machine

Instruction memory (user)

Instruction memory (kernel)

(single-line)

Rule cache:

PC tag	op_1 tag	op_2 tag	op_3 tag	new PC tag	result tag

Inputs

Outputs

Output

...

User mode (cache hit)

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

$$\frac{\iota(n) = \mathsf{Output} \qquad \kappa = \boxed{\mathsf{output}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \_\ \_\ \mathrm{T}_{rpc}\ \mathrm{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [m@\mathrm{T}_1, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{m@\mathrm{T}_r} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n{+}1@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Push}\ m \qquad \kappa = \boxed{\mathsf{push}\ \mathrm{T}_{pc}\ \_\ \_\ \_\ \mathrm{T}_{rpc}\ \mathrm{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [\sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [m@\mathrm{T}_r, \sigma]\ n{+}1@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Load} \qquad \mu(p) = m@\mathrm{T}_2 \qquad \kappa = \boxed{\mathsf{load}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \mathrm{T}_2\ \_\ \mathrm{T}_{rpc}\ \mathrm{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [p@\mathrm{T}_1, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [m@\mathrm{T}_r, \sigma]\ n{+}1@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Store} \qquad \mu(p) = k@\mathrm{T}_3 \qquad \kappa = \boxed{\mathsf{store}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \mathrm{T}_2\ \mathrm{T}_3\ \mathrm{T}_{rpc}\ \mathrm{T}_r} \qquad \mu(p) \leftarrow (m@\mathrm{T}_r) = \mu'}{\mathsf{u}\ \kappa\ \mu\ [p@\mathrm{T}_1, m@\mathrm{T}_2, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu'\ [\sigma]\ n{+}1@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Jump} \qquad \kappa = \boxed{\mathsf{jump}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \_\ \_\ \mathrm{T}_{rpc}\ \_}}{\mathsf{u}\ \kappa\ \mu\ [n'@\mathrm{T}_1, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}_{rpc}}$$

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

$$\frac{\iota(n) = \mathsf{Call} \qquad \kappa = \boxed{\mathsf{call}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \_\ \_\ \mathrm{T}_{rpc}\ \mathrm{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [n'@\mathrm{T}_1, a, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [a, (n{+}1@\mathrm{T}_r, \mathsf{u}); \sigma]\ n'@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Ret} \qquad \kappa = \boxed{\mathsf{ret}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \_\ \_\ \mathrm{T}_{rpc}\ \_}}{\mathsf{u}\ \kappa\ \mu\ [(n'@\mathrm{T}_1, \mathsf{u}); \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}_{rpc}}$$

User-to-kernel mode (cache miss)

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

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

$$\frac{\iota(n) = \mathsf{Push}\ m \qquad \kappa_i \neq \boxed{\mathsf{push}\ \mathrm{T}_{pc}\ \_\ \_\ \_} = \kappa_j}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [\sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathrm{T}_{pc}, \mathsf{u}); \sigma]\ 0@\mathrm{T}_\_}$$

$$\frac{\iota(n) = \mathsf{Load} \qquad \mu(p) = m@\mathrm{T}_2 \qquad \kappa_i \neq \boxed{\mathsf{load}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \mathrm{T}_2\ \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [p@\mathrm{T}_1, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathrm{T}_{pc}, \mathsf{u}); p@\mathrm{T}_1, \sigma]\ 0@\mathrm{T}_\_\end{array}}$$

$$\frac{\iota(n) = \mathsf{Store} \qquad \mu(p) = k@\mathrm{T}_3 \qquad \kappa_i \neq \boxed{\mathsf{store}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \mathrm{T}_2\ \mathrm{T}_3} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [p@\mathrm{T}_1, m@\mathrm{T}_2, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathrm{T}_{pc}, \mathsf{u}); p@\mathrm{T}_1, m@\mathrm{T}_2, \sigma]\ 0@\mathrm{T}_\_\end{array}}$$

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

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

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

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

Kernel mode

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

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

$$\frac{\phi(n) = \mathsf{Load} \qquad \kappa(p) = m@\mathrm{T}_1}{\mathsf{k}\ \kappa\ \mu\ [p@\_, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [m@\mathrm{T}_1, \sigma]\ n{+}1@\mathrm{T}\_}$$

$$\frac{\phi(n) = \mathsf{Store} \qquad \mathsf{store}\ \kappa\ p\ (m@\mathrm{T}_1) = \kappa'}{\mathsf{k}\ \kappa\ \mu\ [p@\_, m@\mathrm{T}_1, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa'\ \mu\ [\sigma]\ n{+}1@\mathrm{T}\_}$$

$$\frac{\phi(n) = \mathsf{Jump}}{\mathsf{k}\ \kappa\ \mu\ [n'@\_, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}\_}$$

$$\frac{\phi(n) = \mathsf{Bnz}\ k \qquad n' = n{+}(m = 0)?1 : k}{\mathsf{k}\ \kappa\ \mu\ [m@\_, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}\_}$$

$$\frac{\phi(n) = \mathsf{Call}}{\mathsf{k}\ \kappa\ \mu\ [n'@\_, a, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [a, (n{+}1@\mathrm{T}\_, \mathsf{k}); \sigma]\ n'@\mathrm{T}\_}$$

$$\frac{\phi(n) = \mathsf{Ret}}{\mathsf{k}\ \kappa\ \mu\ [(n'@\mathrm{T}_1, \pi); \sigma]\ n@\_ \xrightarrow{\tau} \pi\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}_1}$$

User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode

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

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

$$\frac{\iota(n) = \mathsf{Push}\ m \qquad \kappa = \boxed{\mathsf{push}\ \mathsf{T}_{pc}\ \_\ \_\ \_\ \mathsf{T}_{rpc}\ \mathsf{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [\sigma]\ n @ \mathsf{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [m @ \mathsf{T}_r, \sigma]\ n{+}1 @ \mathsf{T}_{rpc}}$$

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

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

$$\frac{\iota(n) = \mathsf{Push}\ m \qquad \kappa_i \neq \boxed{\mathsf{push}\ \mathsf{T}_{pc}\ \_\ \_\ \_} = \kappa_j}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [\sigma]\ n @ \mathsf{T}_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n @ \mathsf{T}_{pc}, \mathsf{u}); \sigma]\ 0 @ \mathsf{T}_\_}$$

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

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

$$\frac{\phi(n) = \mathsf{Load} \qquad \kappa(p) = m @ \mathsf{T}_1}{\mathsf{k}\ \kappa\ \mu\ [p @ \_, \sigma]\ n @ \_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [m @ \mathsf{T}_1, \sigma]\ n{+}1 @ \mathsf{T}_\_}$$

$$\frac{\phi(n) = \mathsf{Store} \qquad \mathsf{store}\ \kappa\ p\ (m @ \mathsf{T}_1) = \kappa'}{\mathsf{k}\ \kappa\ \mu\ [p @ \_, m @ \mathsf{T}_1, \sigma]\ n @ \_ \xrightarrow{\tau} \mathsf{k}\ \kappa'\ \mu\ [\sigma]\ n{+}1 @ \mathsf{T}_\_}$$

$\phi(n) = \mathsf{Jump}$

privilege bit

$$\pi_1\ \kappa_1\ \mu_1\ [\sigma_1]\ pc_1 \xrightarrow{ce} \pi_2\ \kappa_2\ \mu_2\ [\sigma_2]\ pc_2$$

user memory

pc

kernel memory

stack

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

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

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

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

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

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

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

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

User mode (cache hit)

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

$$\frac{\iota(n) = \mathsf{Output} \qquad \kappa = \boxed{\mathsf{output}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \_\ \_\ \mathrm{T}_{rpc}\ \mathrm{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [m@\mathrm{T}_1, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{m@\mathrm{T}_r} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n{+}1@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Push}\ m \qquad \kappa = \boxed{\mathsf{push}\ \mathrm{T}_{pc}\ \_\ \_\ \_\ \mathrm{T}_{rpc}\ \mathrm{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [\sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [m@\mathrm{T}_r, \sigma]\ n{+}1@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Load} \qquad \mu(p) = m@\mathrm{T}_2 \qquad \kappa = \boxed{\mathsf{load}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \mathrm{T}_2\ \_\ \mathrm{T}_{rpc}\ \mathrm{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [p@\mathrm{T}_1, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [m@\mathrm{T}_r, \sigma]\ n{+}1@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Store} \qquad \mu(p) = k@\mathrm{T}_3 \qquad \kappa = \boxed{\mathsf{store}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \mathrm{T}_2\ \mathrm{T}_3\ \mathrm{T}_{rpc}\ \mathrm{T}_r} \qquad \mu(p) \leftarrow (m@\mathrm{T}_r) = \mu'}{\mathsf{u}\ \kappa\ \mu\ [p@\mathrm{T}_1, m@\mathrm{T}_2, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu'\ [\sigma]\ n{+}1@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Jump} \qquad \kappa = \boxed{\mathsf{jump}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \_\ \_\ \mathrm{T}_{rpc}\ \_}}{\mathsf{u}\ \kappa\ \mu\ [n'@\mathrm{T}_1, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}_{rpc}}$$

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

$$\frac{\iota(n) = \mathsf{Call} \qquad \kappa = \boxed{\mathsf{call}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \_\ \_\ \mathrm{T}_{rpc}\ \mathrm{T}_r}}{\mathsf{u}\ \kappa\ \mu\ [n'@\mathrm{T}_1, a, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [a, (n{+}1@\mathrm{T}_r, \mathsf{u}); \sigma]\ n'@\mathrm{T}_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Ret} \qquad \kappa = \boxed{\mathsf{ret}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \_\ \_\ \mathrm{T}_{rpc}\ \_}}{\mathsf{u}\ \kappa\ \mu\ [(n'@\mathrm{T}_1, \mathsf{u}); \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}_{rpc}}$$

User-to-kernel mode (cache miss)

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

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

$$\frac{\iota(n) = \mathsf{Push}\ m \qquad \kappa_i \neq \boxed{\mathsf{push}\ \mathrm{T}_{pc}\ \_\ \_\ \_} = \kappa_j}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [\sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathrm{T}_{pc}, \mathsf{u}); \sigma]\ 0@\mathrm{T}_\_}$$

$$\frac{\iota(n) = \mathsf{Load} \qquad \mu(p) = m@\mathrm{T}_2 \qquad \kappa_i \neq \boxed{\mathsf{load}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \mathrm{T}_2\ \_} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [p@\mathrm{T}_1, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathrm{T}_{pc}, \mathsf{u}); p@\mathrm{T}_1, \sigma]\ 0@\mathrm{T}_\_\end{array}}$$

$$\frac{\iota(n) = \mathsf{Store} \qquad \mu(p) = k@\mathrm{T}_3 \qquad \kappa_i \neq \boxed{\mathsf{store}\ \mathrm{T}_{pc}\ \mathrm{T}_1\ \mathrm{T}_2\ \mathrm{T}_3} = \kappa_j}{\begin{array}{l}\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [p@\mathrm{T}_1, m@\mathrm{T}_2, \sigma]\ n@\mathrm{T}_{pc} \xrightarrow{\tau} \\ \mathsf{k}\ [\kappa_j, \kappa_\_]\ \mu\ [(n@\mathrm{T}_{pc}, \mathsf{u}); p@\mathrm{T}_1, m@\mathrm{T}_2, \sigma]\ 0@\mathrm{T}_\_\end{array}}$$

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

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

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

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

Kernel mode

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

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

$$\frac{\phi(n) = \mathsf{Load} \qquad \kappa(p) = m@\mathrm{T}_1}{\mathsf{k}\ \kappa\ \mu\ [p@\_, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [m@\mathrm{T}_1, \sigma]\ n{+}1@\mathrm{T}\_}$$

$$\frac{\phi(n) = \mathsf{Store} \qquad \mathsf{store}\ \kappa\ p\ (m@\mathrm{T}_1) = \kappa'}{\mathsf{k}\ \kappa\ \mu\ [p@\_, m@\mathrm{T}_1, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa'\ \mu\ [\sigma]\ n{+}1@\mathrm{T}\_}$$

$$\frac{\phi(n) = \mathsf{Jump}}{\mathsf{k}\ \kappa\ \mu\ [n'@\_, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}\_}$$

$$\frac{\phi(n) = \mathsf{Bnz}\ k \qquad n' = n{+}(m = 0)?1 : k}{\mathsf{k}\ \kappa\ \mu\ [m@\_, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}\_}$$

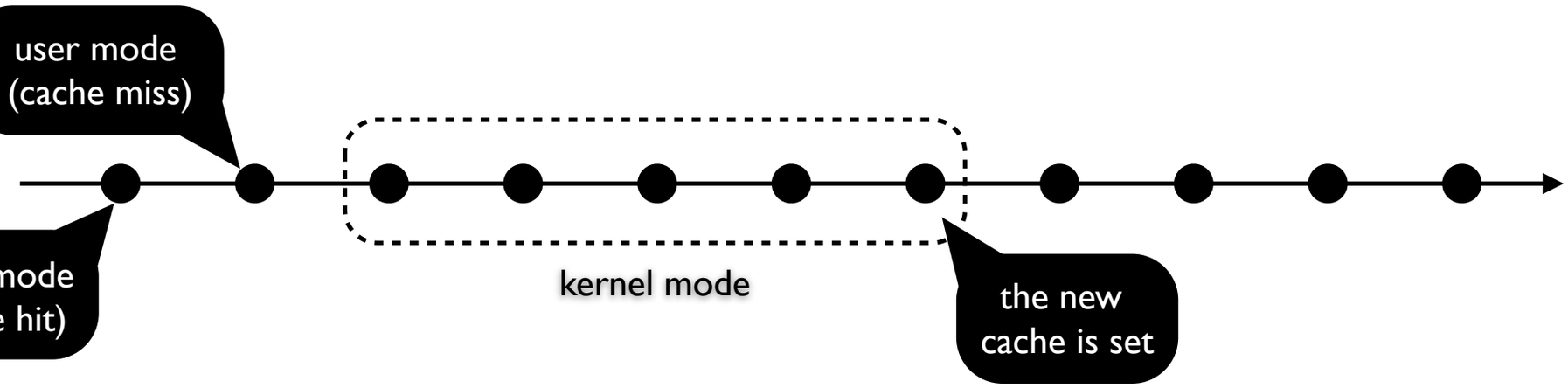
$$\frac{\phi(n) = \mathsf{Call}}{\mathsf{k}\ \kappa\ \mu\ [n'@\_, a, \sigma]\ n@\_ \xrightarrow{\tau} \mathsf{k}\ \kappa\ \mu\ [a, (n{+}1@\mathrm{T}\_, \mathsf{k}); \sigma]\ n'@\mathrm{T}\_}$$

$$\frac{\phi(n) = \mathsf{Ret}}{\mathsf{k}\ \kappa\ \mu\ [(n'@\mathrm{T}_1, \pi); \sigma]\ n@\_ \xrightarrow{\tau} \pi\ \kappa\ \mu\ [\sigma]\ n'@\mathrm{T}_1}$$

User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode



User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode

$$\frac{\iota(n) = \mathsf{Sub} \qquad \kappa = \boxed{\mathsf{sub} \mid T_{pc} \mid T_1 \mid T_2 \mid \_ \,\|\, T_{rpc} \mid T_r}}{\mathsf{u}\ \kappa\ \mu\ [n_1@T_1, n_2@T_2, \sigma]\ n@T_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [(n_1 - n_2)@T_r, \sigma]\ n{+}1@T_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Output} \qquad \kappa = \boxed{\mathsf{output} \mid T_{pc} \mid T_1 \mid \_ \mid \_ \,\|\, T_{rpc} \mid T_r}}{\mathsf{u}\ \kappa\ \mu\ [m@T_1, \sigma]\ n@T_{pc} \xrightarrow{m@T_r} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n{+}1@T_{rpc}}$$

$$\frac{n' = n{+}(m = 0)?1 : k}{\mathsf{u}\ \kappa\ \mu\ [m@T_1, \sigma]\ n@T_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@T_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Call} \qquad \kappa = \boxed{\mathsf{call} \mid T_{pc} \mid T_1 \mid \_ \mid \_ \,\|\, T_{rpc} \mid T_r}}{\mathsf{u}\ \kappa\ \mu\ [n'@T_1, a, \sigma]\ n@T_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [a, (n{+}1@T_r, \mathsf{u}); \sigma]\ n'@T_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Ret} \qquad \kappa = \boxed{\mathsf{ret} \mid T_{pc} \mid T_1 \mid \_ \mid \_ \,\|\, T_{rpc} \mid \_}}{\mathsf{u}\ \kappa\ \mu\ [(n'@T_1, \mathsf{u}); \sigma]\ n@T_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@T_{rpc}}$$

$$\frac{\iota(n) = \mathsf{Sub} \qquad \kappa_i \neq \boxed{\mathsf{sub} \mid T_{pc} \mid T_1 \mid T_2 \mid \_} = \kappa_j}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [n_1@T_1, n_1@T_2, \sigma]\ n@T_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa_{\_}]\ \mu\ [(n@T_{pc}, \mathsf{u}); n_1@T_1, n_1@T_2, \sigma]\ 0@T_{\_}}$$

$$\frac{\iota(n) = \mathsf{Output} \qquad \kappa_i \neq \boxed{\mathsf{output} \mid T_{pc} \mid T_1 \mid \_ \mid \_} = \kappa_j}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [m@T_1, \sigma]\ n@T_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa_{\_}]\ \mu\ [(n@T_{pc}, \mathsf{u}); m@T_1, \sigma]\ 0@T_{\_}}$$

$$\frac{\iota(n) = \mathsf{Bnz}\,k \qquad \kappa_i \neq \boxed{\mathsf{bnz} \mid T_{pc} \mid T_1 \mid \_ \mid \_} = \kappa_j}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [m@T_1, \sigma]\ n@T_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa_{\_}]\ \mu\ [(n@T_{pc}, \mathsf{u}); m@T_1, \sigma]\ 0@T_{\_}}$$

$$\frac{\iota(n) = \mathsf{Call} \qquad \kappa_i \neq \boxed{\mathsf{call} \mid T_{pc} \mid T_1 \mid \_ \mid \_} = \kappa_j}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [n'@T_1, a, \sigma]\ n@T_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa_{\_}]\ \mu\ [(n@T_{pc}, \mathsf{u}); n'@T_1, a, \sigma]\ 0@T_{\_}}$$

$$\frac{\iota(n) = \mathsf{Ret} \qquad \kappa_i \neq \boxed{\mathsf{ret} \mid T_{pc} \mid T_1 \mid \_ \mid \_} = \kappa_j}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [(n'@T_1, \pi); \sigma]\ n@T_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa_{\_}]\ \mu\ [(n@T_{pc}, \mathsf{u}); (n'@T_1, \pi); \sigma]\ 0@T_{\_}}$$

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

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

$$\phi(n) = \mathsf{Load} \qquad \kappa(p) = m@T_1$$

User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode

$$\iota(n) = \mathsf{Sub} \qquad \kappa = \boxed{\mathsf{sub}\ \mathtt{T}_{pc}\ \mathtt{T}_1\ \mathtt{T}_2\ \_\ \|\ \mathtt{T}_{rpc}\ \mathtt{T}_r}$$

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

User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\iota(n) = \mathsf{Sub} \qquad \kappa = \boxed{\mathsf{sub}\,|\,\mathtt{T}_{pc}\,|\,\mathtt{T}_1\,|\,\mathtt{T}_2\,|\,\_\,||\,\mathtt{T}_{rpc}\,|\,\mathtt{T}_r}}{\begin{array}{l}\mathsf{u}\ \kappa\ \mu\ [n'@\mathtt{T}_1, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \\ \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathtt{T}_{rpc}\end{array}}$$

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

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

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

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

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

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

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

User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode

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

User mode (cache hit)

User-to-kernel mode (cache miss)

Kernel mode

$$\phi(n) = \mathsf{Sub}$$

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

$\iota(n) = \mathsf{Bnz}\ k$
$\kappa = [\mathsf{bnz} \mid \mathtt{T}_{pc} \mid \mathtt{T}_1 \mid \_ \mid \_ \mid \mathtt{T}_{rpc} \mid \_]$
$n' = n{+}(m = 0)?1 : k$

$$\frac{}{\mathsf{u}\ \kappa\ \mu\ [m@\mathtt{T}_1, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathtt{T}_{rpc}}$$

$\iota(n) = \mathsf{Bnz}\ k$
$\kappa_i \neq [\mathsf{bnz} \mid \mathtt{T}_{pc} \mid \mathtt{T}_1 \mid \_ \mid \_] = \kappa_j$

$$\frac{}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [m@\mathtt{T}_1, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); m@\mathtt{T}_1, \sigma]\ 0@\mathtt{T}\_}$$

$\iota(n) = \mathsf{Call}$
$\kappa = [\mathsf{call} \mid \mathtt{T}_{pc} \mid \mathtt{T}_1 \mid \_ \mid \_ \mid \mathtt{T}_{rpc} \mid \mathtt{T}_r]$

$$\frac{}{\mathsf{u}\ \kappa\ \mu\ [n'@\mathtt{T}_1, a, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [a, (n{+}1@\mathtt{T}_r, \mathsf{u}); \sigma]\ n'@\mathtt{T}_{rpc}}$$

$\iota(n) = \mathsf{Call}$
$\kappa_i \neq [\mathsf{call} \mid \mathtt{T}_{pc} \mid \mathtt{T}_1 \mid \_ \mid \_] = \kappa_j$

$$\frac{}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [n'@\mathtt{T}_1, a, \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); n'@\mathtt{T}_1, a, \sigma]\ 0@\mathtt{T}\_}$$

$\iota(n) = \mathsf{Ret}$
$\kappa = [\mathsf{ret} \mid \mathtt{T}_{pc} \mid \mathtt{T}_1 \mid \_ \mid \_ \mid \mathtt{T}_{rpc} \mid \_]$

$$\frac{}{\mathsf{u}\ \kappa\ \mu\ [(n'@\mathtt{T}_1, \mathsf{u}); \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \mathsf{u}\ \kappa\ \mu\ [\sigma]\ n'@\mathtt{T}_{rpc}}$$

$\iota(n) = \mathsf{Ret}$
$\kappa_i \neq [\mathsf{ret} \mid \mathtt{T}_{pc} \mid \mathtt{T}_1 \mid \_ \mid \_] = \kappa_j$

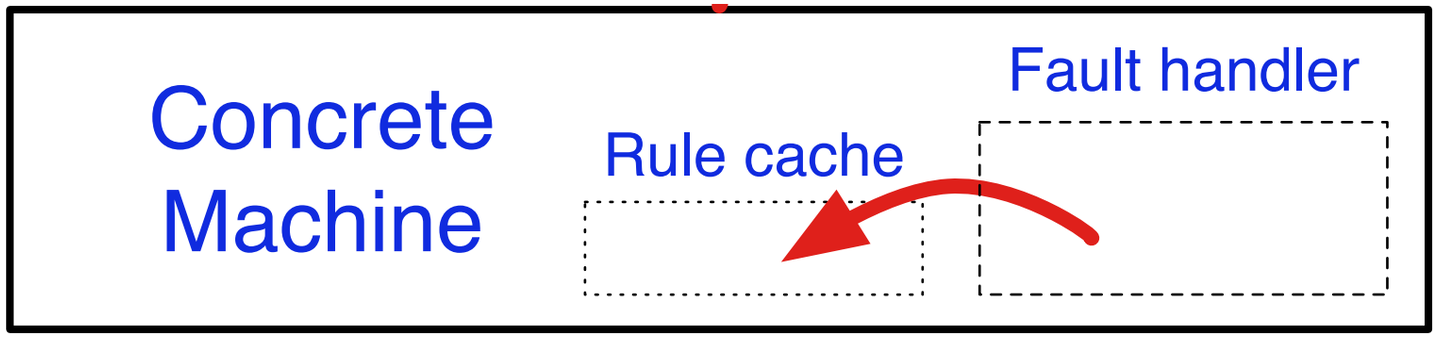
$$\frac{}{\mathsf{u}\ [\kappa_i, \kappa_o]\ \mu\ [(n'@\mathtt{T}_1, \pi); \sigma]\ n@\mathtt{T}_{pc} \xrightarrow{\tau} \mathsf{k}\ [\kappa_j, \kappa\_]\ \mu\ [(n@\mathtt{T}_{pc}, \mathsf{u}); (n'@\mathtt{T}_1, \pi); \sigma]\ 0@\mathtt{T}\_}$$

User mode (cache hit)

User-to-kernel mode (cache miss)

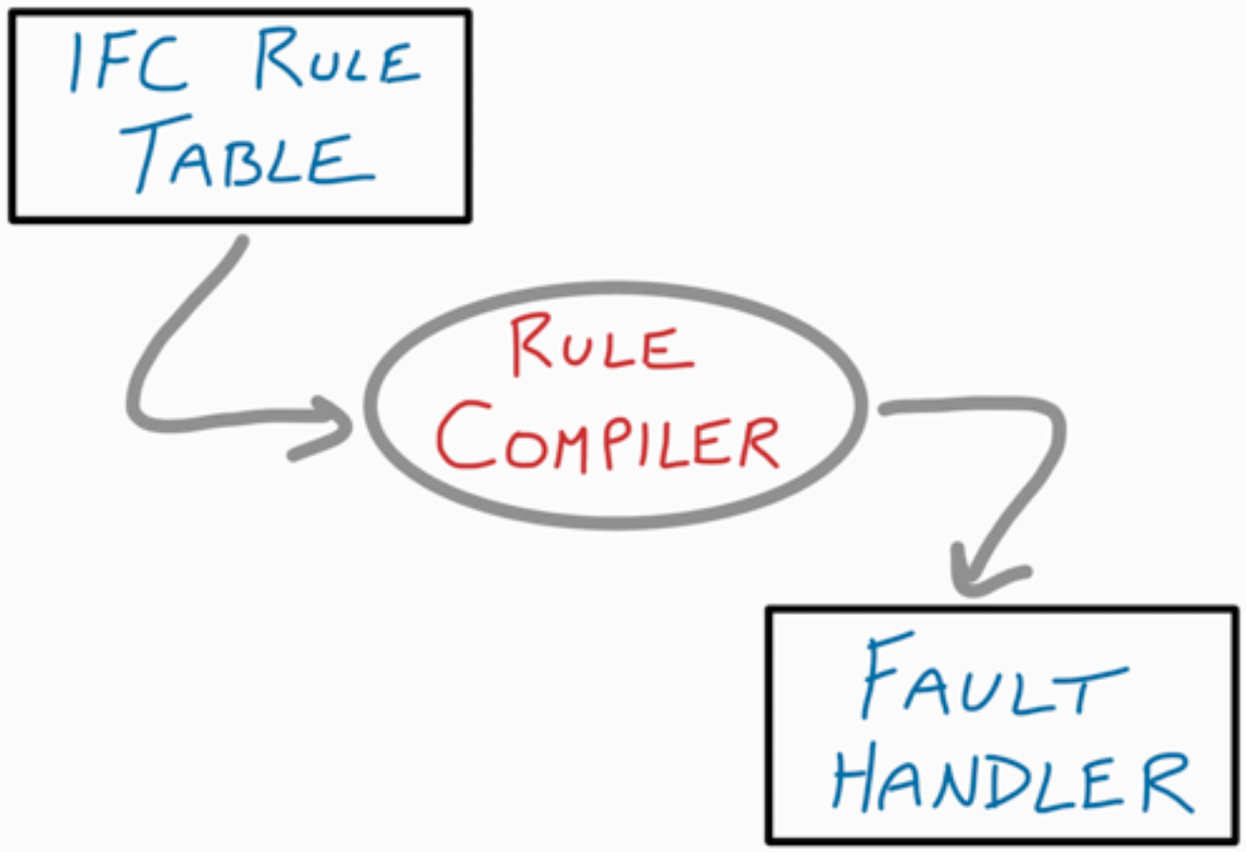
Kernel mode

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



Fault Handler

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	$\mathtt{LAB}_1$	--



Handler Generation

Handler Generation

- IFC rule table entries form a small DSL for computing labels and booleans
  - parameterized over lattice $\bot$, $\sqcup$ and $\sqsubseteq$

Handler Generation

- IFC rule table entries form a small DSL for computing labels and booleans
  - parameterized over lattice $\bot$, $\sqcup$ and $\sqsubseteq$
- The handler is constructed by compiling the DSL into concrete machine instructions

Handler Generation

- IFC rule table entries form a small DSL for computing labels and booleans
  - parameterized over lattice $\bot$, $\sqcup$ and $\sqsubseteq$
- The handler is constructed by compiling the DSL into concrete machine instructions
- Need to encode abstract label lattice into integer tags

  Tag: Label→Int Lab: Int→Label

  genBot, genJoin, genFlows : code sequence computing on integers

Handler Generation

- IFC rule table entries form a small DSL for computing labels and booleans
  - parameterized over lattice $\bot$, $\sqcup$ and $\sqsubseteq$
- The handler is constructed by compiling the DSL into concrete machine instructions
- Need to encode abstract label lattice into integer tags

  Tag: Label→Int Lab: Int→Label

  genBot, genJoin, genFlows : code sequence computing on integers
- We use structured code generators to simplify verification

```
genFaultHandler R = genComputeResults R ++
                    genStoreResults ++
                    genIf [Ret] [Push (-1); Jump]

genComputeResults R =
  genIndexedCases genMatchOp (genApplyRule ∘ Rule_R) opcodes

genMatchOp op =
  [Push op] ++ genLoadFrom addrOpLabel ++ genEqual
genEqual = [Sub] ++ genNot

genApplyRule ⟨allow, e_rpc, e_r⟩ = genBool allow ++
  genIf (genSome (genELab e_rpc ++ genELab e_r)) genNone

genELab BOT          = genBot
        LAB_i        = genLoadFrom addrTag_i
        LE_1 ⊔ LE_2  = genELab LE_2 ++ genELab LE_1 ++ genJoin

genBool TRUE         = genTrue
        LE_1 ⊑ LE_2  = genELab LE_2 ++ genELab LE_1 ++ genFlows

genStoreResults =
  genIf (genStoreAt addrTag_r ++ genStoreAt addrTag_rpc ++ genTrue)
        genFalse
```

Match opcode

Compute results of rule

Update the cache

```
genFalse     =  [Push 0]
genTrue      =  [Push 1]
genAnd       =  genIf [] (genPop ++ genFalse)
genOr        =  genIf (genPop ++ genTrue) []
genNot       =  genIf genFalse genTrue
genImpl      =  genNot ++ genOr
genSome c    =  c ++ genTrue
genNone      =  genFalse

genIndexedCases genDefault genGuard genBody = g
   where g nil       = genDefault
         g (n :: ns) = genGuard n ++ genIf (genBody n) (g ns)

genIf t f          =  genSkipIf (length f') ++ f' ++ t
                        where f' = f ++ genSkip(length t)
genSkip n          =  genTrue ++ genSkipIf n
genSkipIf n        =  [Bnz (n+1)]
genStoreAt p       =  [Push p; Store]
genLoadFrom p      =  [Push p; Load]
genPop             =  [Bnz 1]

opcodes = add :: output :: ... :: ret :: nil
```

```
genFaultHandler R = genComputeResults R ++
                    genStoreResults ++
                    genIf [Ret] [Push (-1); Jump]

genComputeResults R =
  genIndexedCases genMatchOp (genApplyRule ∘ Rule_R) opcodes
```

$$
\begin{array}{lrl}
\text{genELab } \texttt{BOT} & = & \text{genBot} \\
\texttt{LAB}_i & = & \text{genLoadFrom addrTag}_i \\
LE_1 \sqcup LE_2 & = & \text{genELab } LE_2 \mathbin{+\!\!+} \text{genELab } LE_1 \mathbin{+\!\!+} \text{genJoin} \\
 & & \\
\text{genBool } \texttt{TRUE} & = & \text{genTrue} \\
LE_1 \sqsubseteq LE_2 & = & \text{genELab } LE_2 \mathbin{+\!\!+} \text{genELab } LE_1 \mathbin{+\!\!+} \text{genFlows}
\end{array}
$$

```
  genIf (genStoreAt addrTag_r ++ genStoreAt addrTag_rpc ++ genTrue)
        genFalse
```

```
genFalse     =  [Push 0]
genTrue      =  [Push 1]
genAnd       =  genIf [] (genPop ++ genFalse)
genOr        =  genIf (genPop ++ genTrue) []
genNot       =  genIf genFalse genTrue
genImpl      =  genNot ++ genOr
genSome c    =  c ++ genTrue
genNone      =  genFalse

genIndexedCases genDefault genGuard genBody = g
   where g nil       = genDefault
         g (n :: ns) = genGuard n ++ genIf (genBody n) (g ns)

genIf t f          =  genSkipIf (length f') ++ f' ++ t
                        where f' = f ++ genSkip(length t)
genSkip n          =  genTrue ++ genSkipIf n
genSkipIf n        =  [Bnz (n+1)]
genStoreAt p       =  [Push p; Store]
genLoadFrom p      =  [Push p; Load]
genPop             =  [Bnz 1]

opcodes = add :: output :: ... :: ret :: nil
```

genFaultHandler $\mathcal{R}$ = genComputeResults $\mathcal{R}$ ++
genStoreResults ++
genIf [Ret] [Push (-1); Jump]

genComputeResults $\mathcal{R}$ =
genIndexedCases genMatchOp (genApplyRule ∘ $Rule_{\mathcal{R}}$) opcodes

genMatchOp op =
[Push op] ++ genLoadFrom addrOpLabel ++ genEqual
genEqual = [Sub] ++ genNot

Match opcode

genApplyRule $\langle allow, e_{rpc}, e_r \rangle$ = genBool $allow$ ++
genIf (genSome (genELab e_{rpc} ++ genELab e_r)) genNone

Compute results of rule

genELab `BOT` = genBot
`LAB`$_i$ = genLoadFrom addrTag$_i$
$LE_1 \sqcup LE_2$ = genELab LE_2 ++ genELab LE_1 ++ genJoin

genBool `TRUE` = genTrue
$LE_1 \sqsubseteq LE_2$ = genELab LE_2 ++ genELab LE_1 ++ genFlows

genStoreResults =
genIf (genStoreAt addrTag$_r$ ++ genStoreAt addrTag$_{rpc}$ ++ genTrue)
genFalse

Update the cache

genFalse = [Push 0]
genTrue = [Push 1]
genAnd = genIf [] (genPop ++ genFalse)

genIf t f = genSkipIf (length f') ++ f' ++ t
where f' = f ++ genSkip(length t)
genSkip n = genTrue ++ genSkipIf n
genSkipIf n = [Bnz (n+1)]

genStoreAt p = [Push p; Store]
genLoadFrom p = [Push p; Load]
genPop = [Bnz 1]

opcodes = add :: output :: . . . :: ret :: nil

```
genFaultHandler R = genComputeResults R ++
                    genStoreResults ++
                    genIf [Ret] [Push (-1); Jump]

genComputeResults R =
  genIndexedCases genMatchOp (genApplyRule ∘ Rule_R) opcodes

genMatchOp op =
  [Push op] ++ genLoadFrom addrOpLabel ++ genEqual
genEqual = [Sub] ++ genNot

genApplyRule ⟨allow, e_rpc, e_r⟩ = genBool allow ++
  genIf (genSome (genELab e_rpc ++ genELab e_r)) genNone

genELab BOT          = genBot
        LAB_i        = genLoadFrom addrTag_i
        LE_1 ⊔ LE_2  = genELab LE_2 ++ genELab LE_1 ++ genJoin

genBool TRUE         = genTrue
        LE_1 ⊑ LE_2  = genELab LE_2 ++ genELab LE_1 ++ genFlows

genStoreResults =
  genIf (genStoreAt addrTag_r ++ genStoreAt addrTag_rpc ++ genTrue)
        genFalse
```

Match opcode

Compute results of rule

Update the cache

```
genFalse     =  [Push 0]
genTrue      =  [Push 1]
genAnd       =  genIf [] (genPop ++ genFalse)
genOr        =  genIf (genPop ++ genTrue) []
genNot       =  genIf genFalse genTrue
genImpl      =  genNot ++ genOr
genSome c    =  c ++ genTrue
genNone      =  genFalse

genIndexedCases genDefault genGuard genBody = g
   where g nil       = genDefault
         g (n :: ns) = genGuard n ++ genIf (genBody n) (g ns)

genIf t f          =  genSkipIf (length f') ++ f' ++ t
                        where f' = f ++ genSkip(length t)
genSkip n          =  genTrue ++ genSkipIf n
genSkipIf n        =  [Bnz (n+1)]
genStoreAt p       =  [Push p; Store]
genLoadFrom p      =  [Push p; Load]
genPop             =  [Bnz 1]

opcodes = add :: output :: ... :: ret :: nil
```

Handler Correctness

Lemma 7.1 (Fault handler correctness, allowed case)**.** Suppose that $\vdash_{\mathcal{R}} (L_{pc}, \ell_1, \ell_2, \ell_3) \rightsquigarrow_{opcode} L_{rpc}, L_r$ and

$$\kappa_i = \boxed{opcode \mid \mathsf{Tag}(L_{pc}) \mid \mathsf{Tag}(\ell_1) \mid \mathsf{Tag}(\ell_2) \mid \mathsf{Tag}(\ell_3)}\,.$$

$$\begin{aligned} \text{Then } \phi_{\mathcal{R}} \vdash \ & \langle \mathsf{k} \ [\kappa_i, \kappa_o] \ \mu \ [(pc, \mathsf{u}); \sigma] \ 0@\mathtt{T}\_ \rangle \ \rightarrow^{\star}_{\mathsf{k}} \\ & \langle \mathsf{u} \ [\kappa_i, \kappa'_o] \ \mu \ [\sigma] \qquad\qquad\quad pc \rangle \end{aligned}$$

$$\text{with output cache } \kappa'_o = \boxed{\mathsf{Tag}(L_{rpc}) \mid \mathsf{Tag}(L_r)}\,.$$

and similarly for disallowed case

Correctness proof

- Based on custom Hoare logic

$$\{P\}\ c\ \{Q\} \triangleq$$
$$c = \phi(n), \ldots, \phi(n' - 1) \wedge P(\kappa, \sigma) \Longrightarrow$$
$$\exists\, \kappa'\ \sigma'.\quad Q(\kappa', \sigma')$$
$$\wedge\ \phi \vdash \langle \mathsf{k}\ \kappa\ \mu\ [\sigma]\ n@\mathtt{T}\_\rangle \rightarrow^{\star}_{\mathsf{k}} \langle \mathsf{k}\ \kappa'\ \mu\ [\sigma']\ n'@\mathtt{T}\_\rangle$$

Correctness proof

- Based on custom Hoare logic

$$\{P\}\ c\ \{Q\} \triangleq$$
$$c = \phi(n), \ldots, \phi(n' - 1) \wedge P(\kappa, \sigma) \Longrightarrow$$
$$\exists\, \kappa'\ \sigma'.\quad Q(\kappa', \sigma')$$
$$\wedge\ \phi \vdash \langle \mathsf{k}\ \kappa\ \mu\ [\sigma]\ n@\mathtt{T\_}\rangle \rightarrow^{\star}_{\mathsf{k}} \langle \mathsf{k}\ \kappa'\ \mu\ [\sigma']\ n'@\mathtt{T\_}\rangle$$

- Only apply to self-contained code that "falls off end"

Correctness proof

- Based on custom Hoare logic

$$\{P\}\ c\ \{Q\} \triangleq$$
$$c = \phi(n), \ldots, \phi(n' - 1) \wedge P(\kappa, \sigma) \Longrightarrow$$
$$\exists\, \kappa'\ \sigma'.\quad Q(\kappa', \sigma')$$
$$\wedge\ \phi \vdash \langle \mathsf{k}\ \kappa\ \mu\ [\sigma]\ n@\mathtt{T\_}\rangle \rightarrow^{\star}_{\mathsf{k}} \langle \mathsf{k}\ \kappa'\ \mu\ [\sigma']\ n'@\mathtt{T\_}\rangle$$

- Only apply to self-contained code that "falls off end"
  - Usual composition law holds

Correctness proof

- Based on custom Hoare logic

$$\{P\}\ c\ \{Q\} \triangleq$$
$$c = \phi(n), \ldots, \phi(n' - 1) \wedge P(\kappa, \sigma) \Longrightarrow$$
$$\exists\, \kappa'\ \sigma'.\quad Q(\kappa', \sigma')$$
$$\wedge\ \phi \vdash \langle \mathsf{k}\ \kappa\ \mu\ [\sigma]\ n@\mathtt{T\_} \rangle \rightarrow^{\star}_{\mathsf{k}} \langle \mathsf{k}\ \kappa'\ \mu\ [\sigma']\ n'@\mathtt{T\_} \rangle$$

- Only apply to self-contained code that "falls off end"
  - Usual composition law holds
  - We use different form of triple to describe "escaping" code

Correctness proof

- Based on custom Hoare logic

$$\{P\}\ c\ \{Q\} \triangleq$$
$$c = \phi(n), \ldots, \phi(n' - 1) \wedge P(\kappa, \sigma) \Longrightarrow$$
$$\exists\, \kappa'\ \sigma'.\quad Q(\kappa', \sigma')$$
$$\wedge\ \phi \vdash \langle \mathsf{k}\ \kappa\ \mu\ [\sigma]\ n@\mathtt{T}\_ \rangle \rightarrow^{\star}_{\mathsf{k}} \langle \mathsf{k}\ \kappa'\ \mu\ [\sigma']\ n'@\mathtt{T}\_ \rangle$$

- Only apply to self-contained code that "falls off end"
  - Usual composition law holds
  - We use different form of triple to describe "escaping" code
- Imply total correctness, because deterministic

Proof style

- Triples for each instruction, each generator, e.g.

$$\frac{P(\kappa,\sigma) := \exists\, n_1\ \mathtt{T}_1\ n_2\ \mathtt{T}_2\ \sigma'.\quad \sigma = n_1@\mathtt{T}_1, n_2@\mathtt{T}_2, \sigma' \wedge Q(\kappa, ((n_1{+}n_2)@\mathtt{T}\_, \sigma'))}{\{P\}\ [\mathsf{Add}]\ \{Q\}}$$

Proof style

- Triples for each instruction, each generator, e.g.

$$\frac{P(\kappa,\sigma) := \exists\, n_1\ \mathtt{T}_1\ n_2\ \mathtt{T}_2\ \sigma'.\quad \sigma = n_1@\mathtt{T}_1, n_2@\mathtt{T}_2, \sigma' \wedge Q(\kappa, ((n_1{+}n_2)@\mathtt{T}\_, \sigma'))}{\{P\}\ [\mathsf{Add}]\ \{Q\}}$$

$$\frac{P(\kappa,\sigma) := \exists\, n\ \mathtt{T}\ \sigma'.\ \sigma = n@\mathtt{T}, \sigma' \wedge (n \neq 0 \implies P_1(\kappa,\sigma')) \wedge (n = 0 \implies P_2(\kappa,\sigma')) \quad \{P_1\}\ c_1\ \{Q\} \quad \{P_2\}\ c_2\ \{Q\}}{\{P\}\ \mathsf{genIf}\ c_1\ c_2\ \{Q\}}$$

Proof style

- Triples for each instruction, each generator, e.g.

$$\frac{P(\kappa,\sigma) := \exists\, n_1\ \mathtt{T}_1\ n_2\ \mathtt{T}_2\ \sigma'.\quad \sigma = n_1@\mathtt{T}_1, n_2@\mathtt{T}_2, \sigma' \wedge Q(\kappa, ((n_1{+}n_2)@\mathtt{T}\_, \sigma'))}{\{P\}\ [\mathsf{Add}]\ \{Q\}}$$

$$\frac{\begin{array}{l} P(\kappa,\sigma) := \exists\, n\ \mathtt{T}\ \sigma'.\ \sigma = n@\mathtt{T}, \sigma' \wedge (n \neq 0 \implies P_1(\kappa,\sigma')) \\ \qquad\qquad \wedge (n = 0 \implies P_2(\kappa,\sigma')) \\ \{P_1\}\ c_1\ \{Q\} \qquad \{P_2\}\ c_2\ \{Q\} \end{array}}{\{P\}\ \mathsf{genIf}\ c_1\ c_2\ \{Q\}}$$

- Triples for concrete lattice operations (genBot, genJoin, genFlows)

Proof style

- Triples for each instruction, each generator, e.g.

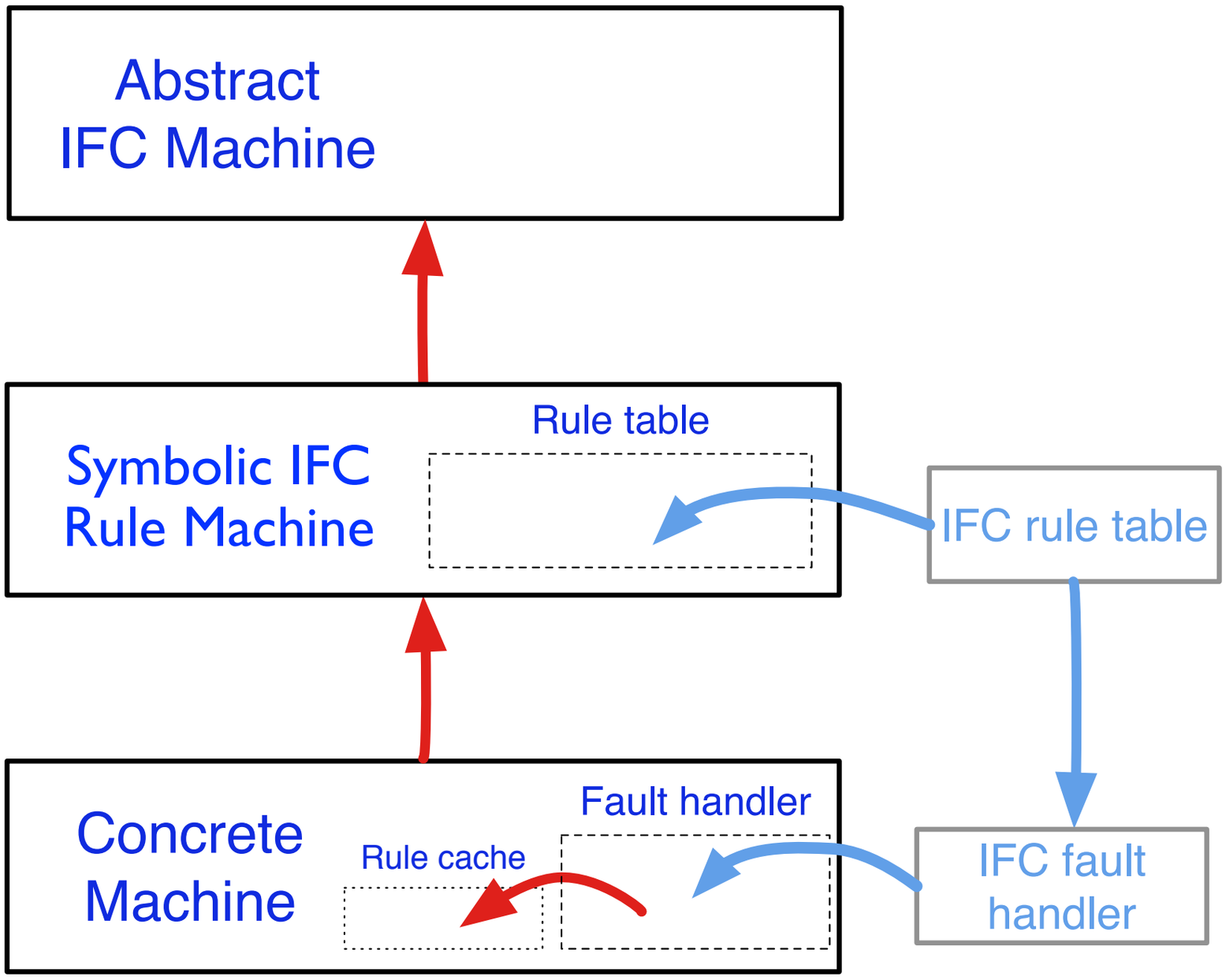
$$\frac{P(\kappa,\sigma) := \exists\, n_1\ \mathtt{T}_1\ n_2\ \mathtt{T}_2\ \sigma'.\quad \sigma = n_1@\mathtt{T}_1, n_2@\mathtt{T}_2, \sigma' \wedge Q(\kappa, ((n_1{+}n_2)@\mathtt{T}\_, \sigma'))}{\{P\}\ [\mathsf{Add}]\ \{Q\}}$$

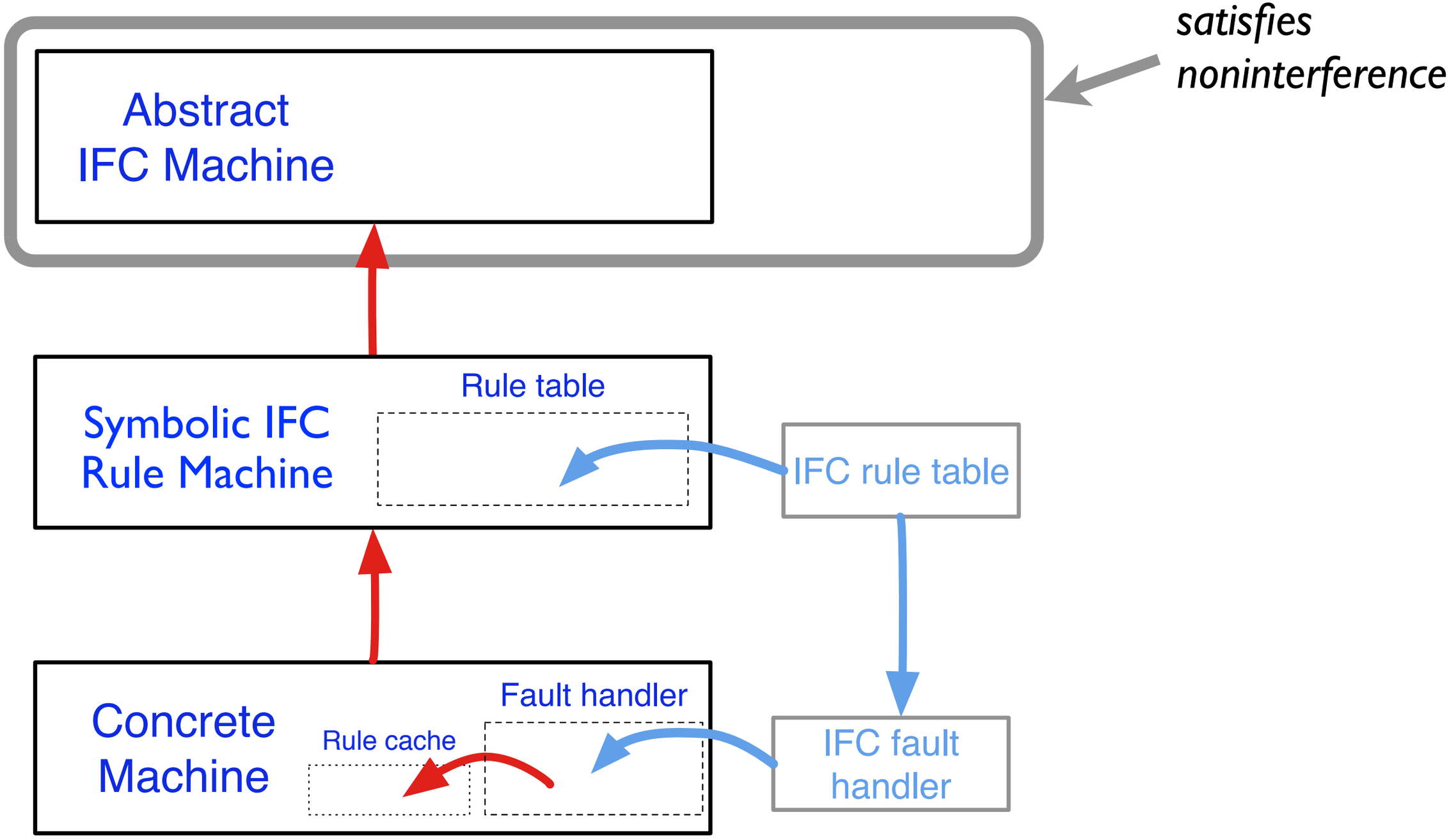
$$\frac{P(\kappa,\sigma) := \exists\, n\ \mathtt{T}\ \sigma'.\ \sigma = n@\mathtt{T}, \sigma' \wedge (n \neq 0 \implies P_1(\kappa,\sigma')) \wedge (n = 0 \implies P_2(\kappa,\sigma')) \quad \{P_1\}\ c_1\ \{Q\} \quad \{P_2\}\ c_2\ \{Q\}}{\{P\}\ \mathsf{genIf}\ c_1\ c_2\ \{Q\}}$$

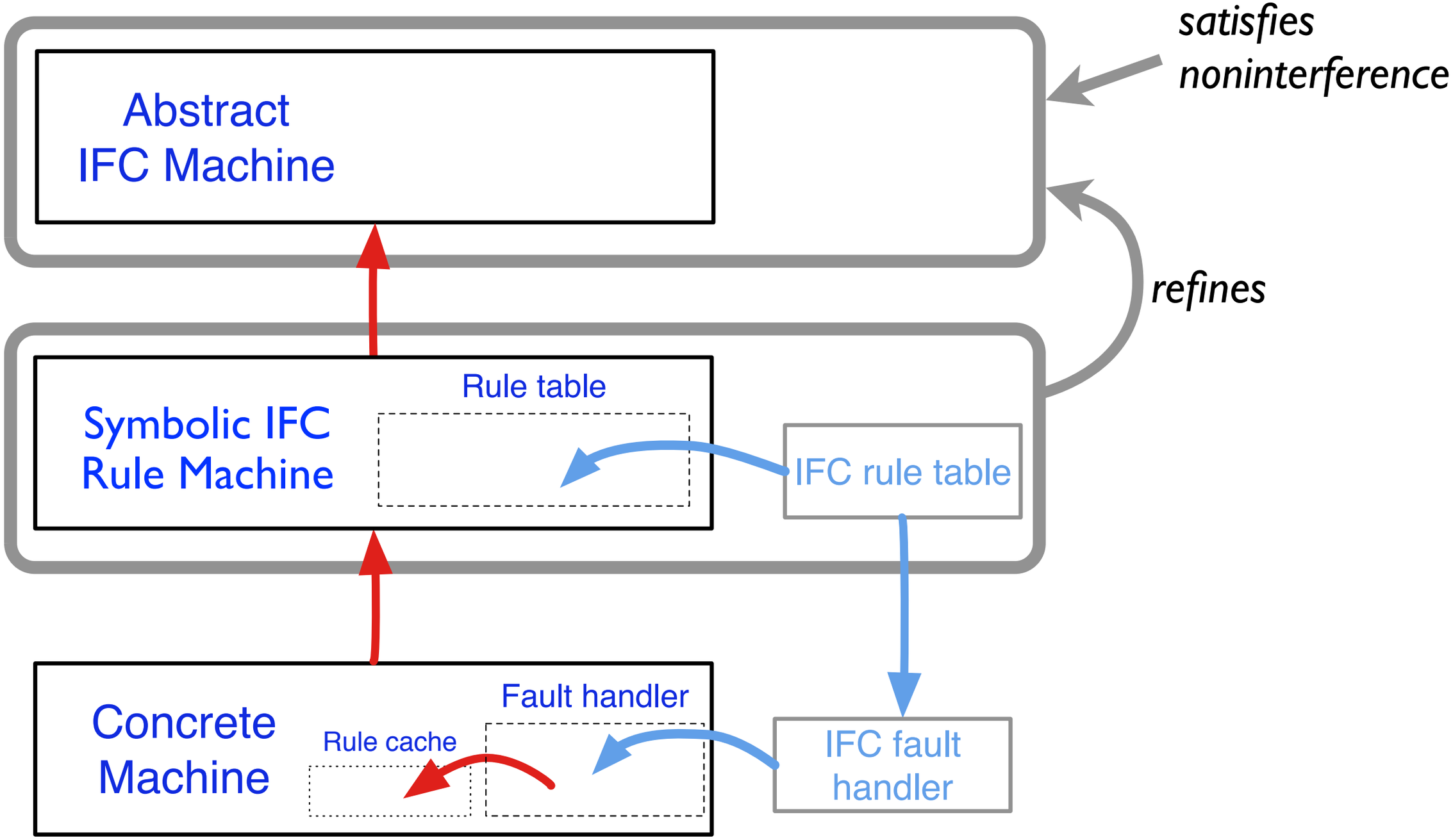
- Triples for concrete lattice operations (genBot, genJoin, genFlows)
- WP style helps automate proofs (straight-line code)

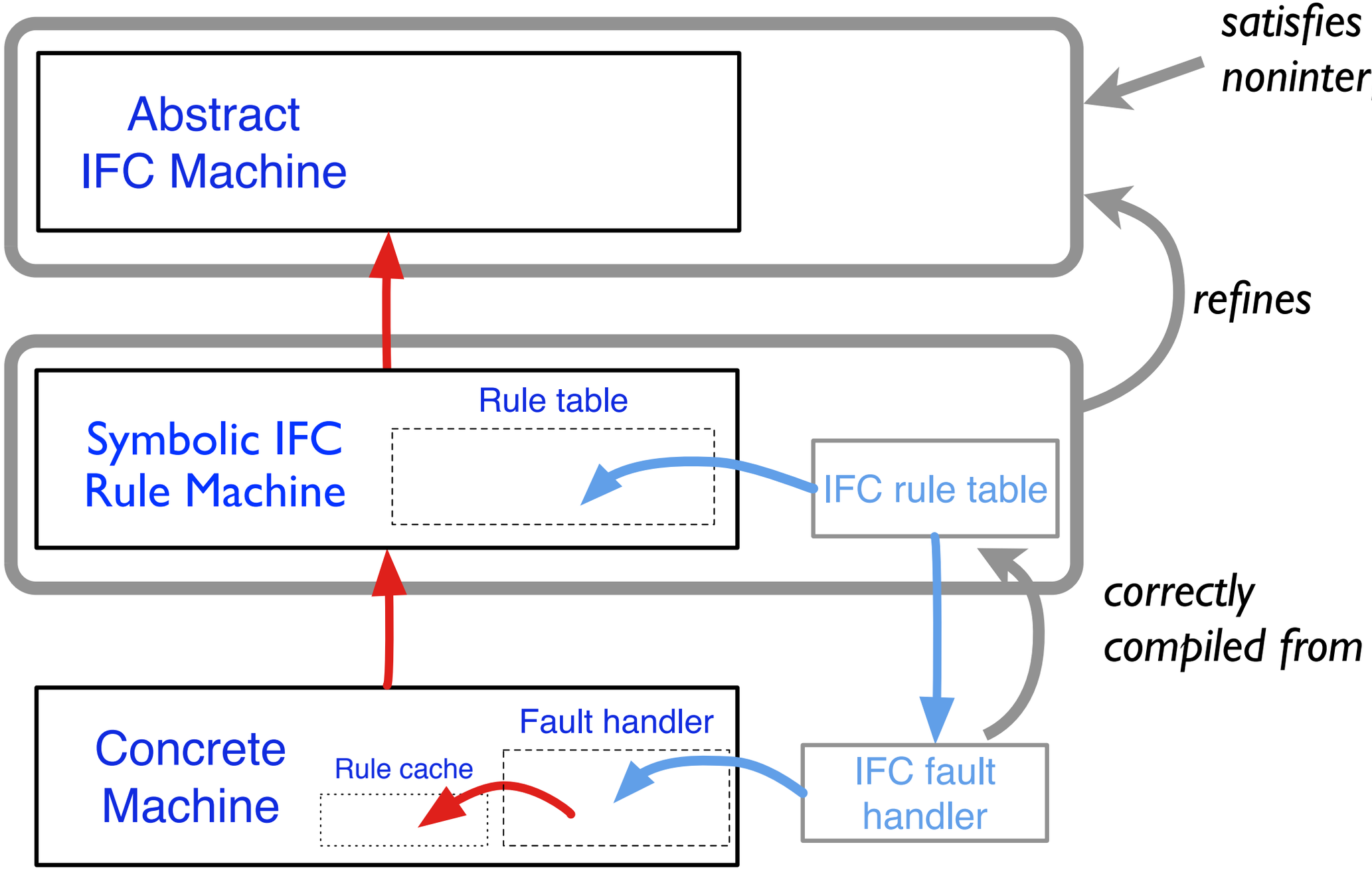
Non-interference for concrete machine

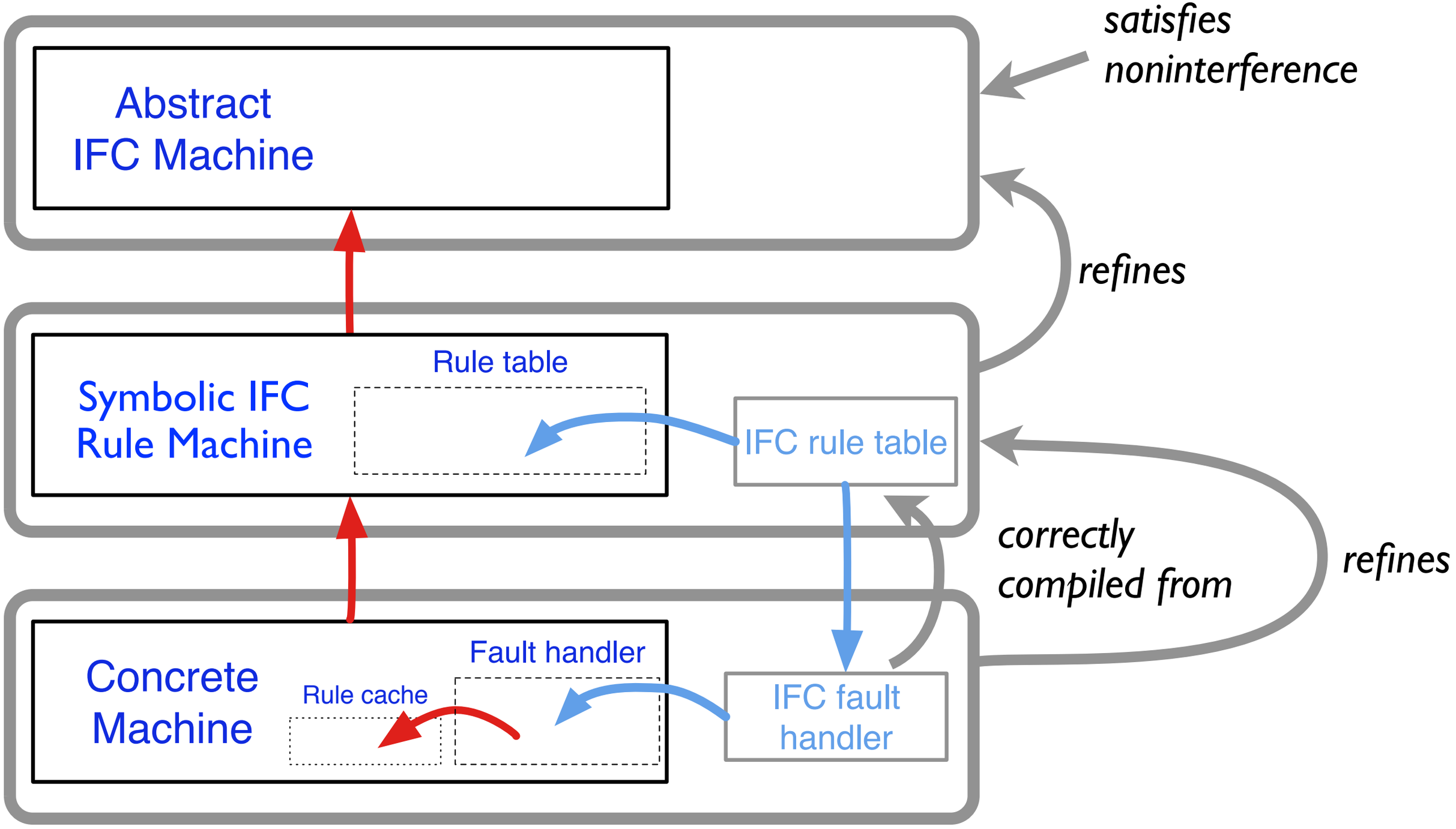
- Running this particular fault handler
- Together with arbitrary user code

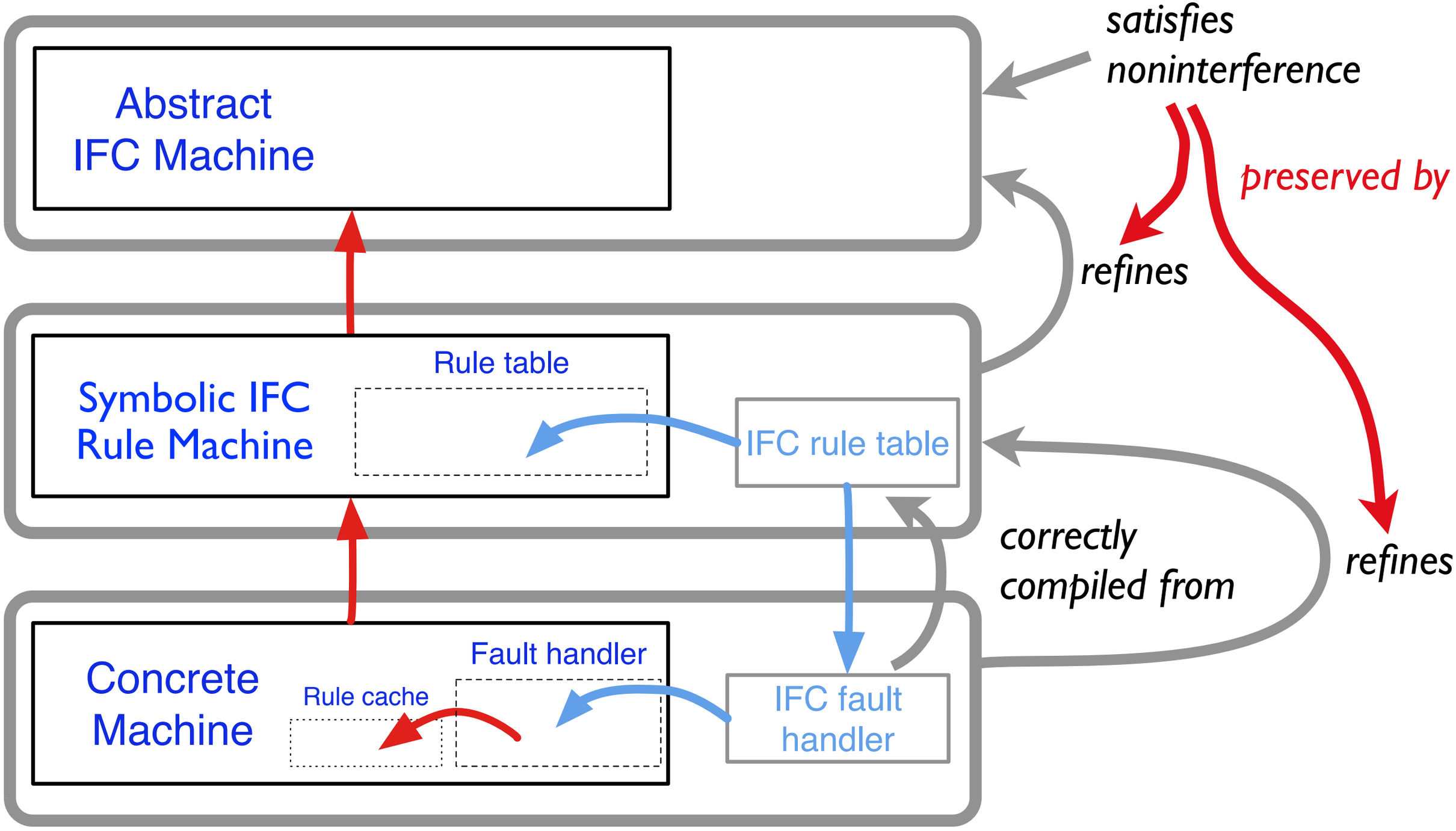

Abstract
IFC Machine
Symbolic IFC
Rule Machine
Rule table
IFC rule table
Concrete
Machine
Rule cache
Fault handler
IFC fault
handler

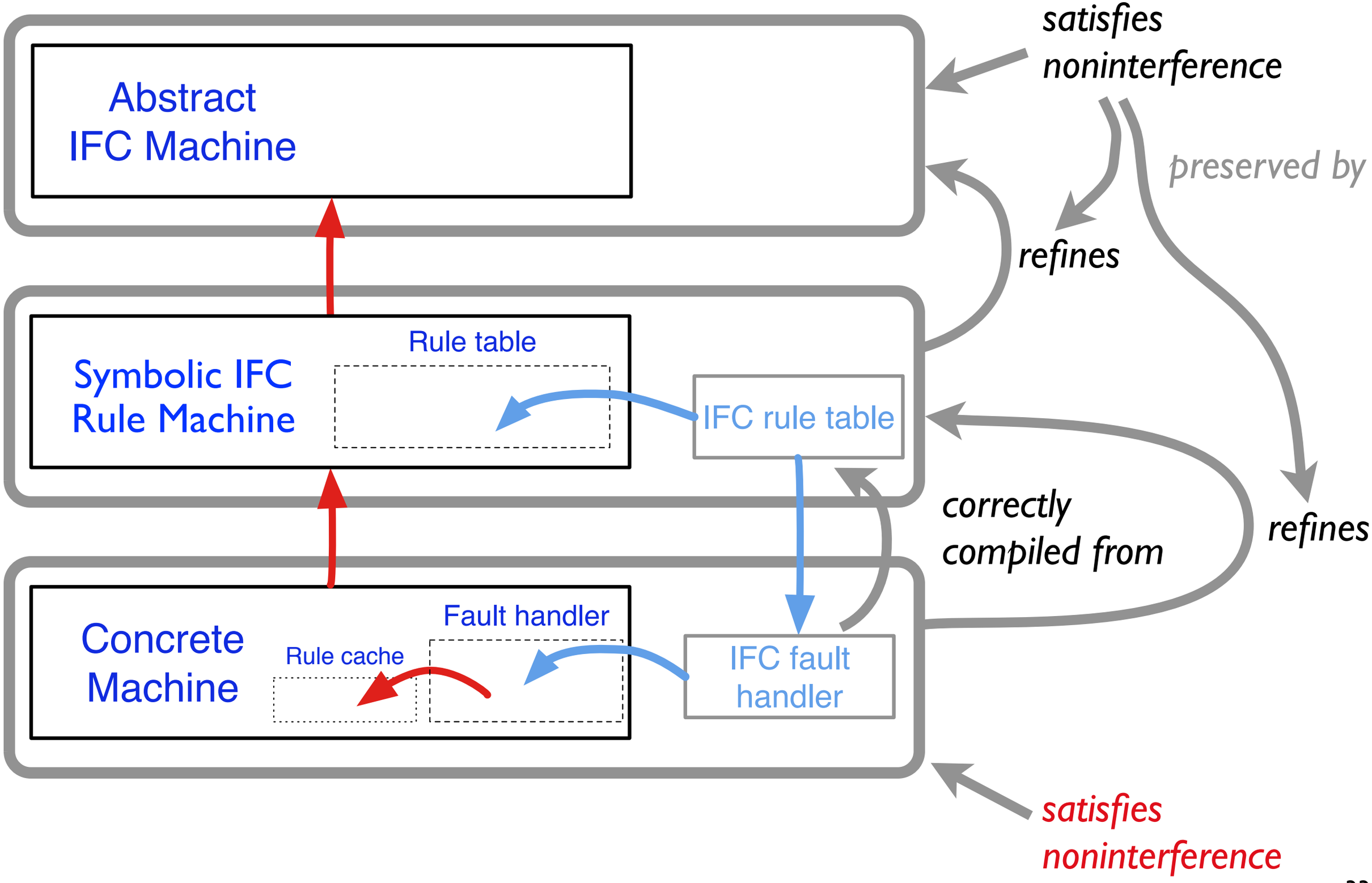

Abstract
IFC Machine
satisfies
noninterference
Symbolic IFC
Rule Machine
Rule table
IFC rule table
Concrete
Machine
Rule cache
Fault handler
IFC fault
handler


satisfies
noninterference
Abstract
IFC Machine
refines
Rule table
Symbolic IFC
Rule Machine
IFC rule table
Concrete
Machine
Fault handler
Rule cache
IFC fault
handler


Abstract
IFC Machine
satisfies
noninterference
refines
Rule table
Symbolic IFC
Rule Machine
IFC rule table
correctly
compiled from
Fault handler
Rule cache
Concrete
Machine
IFC fault
handler


Abstract
IFC Machine
satisfies
noninterference
refines
Rule table
Symbolic IFC
Rule Machine
IFC rule table
correctly
compiled from
refines
Fault handler
Concrete
Machine
Rule cache
IFC fault
handler


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


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

Points to note

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

- 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
- Need a *generic* notion of noninterference that makes sense for all machines
  - Includes a notion of abstracting concrete tags (and associated memory states) into labels

Conclusion

Not presented here : richer IFC tag model

More uses for tags

Conclusion

Not presented here : richer IFC tag model

- Sets of statically known principals

More uses for tags

Conclusion

Not presented here : richer IFC tag model

- Sets of statically known principals
- Memory allocation, and tags are pointers to data structures dynamically allocated in kernel memory

More uses for tags

Conclusion

Not presented here : richer IFC tag model

- Sets of statically known principals
- Memory allocation, and tags are pointers to data structures dynamically allocated in kernel memory

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 [Montagu CSF '13]

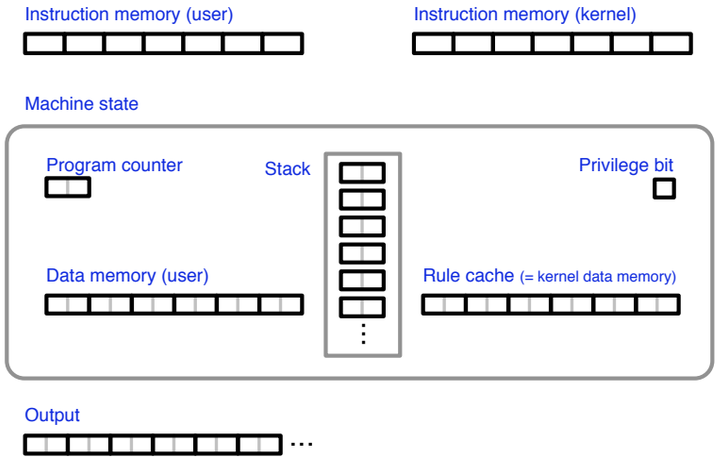
Conclusion

Not presented here : richer IFC tag model

- Sets of statically known principals
- Memory allocation, and tags are pointers to data structures dynamically allocated in kernel memory

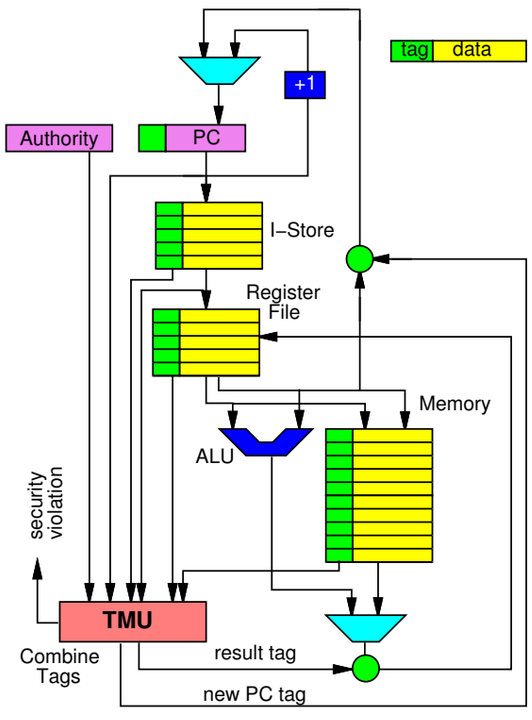
More uses for tags

- SAFE architecture is quite generic
  - Can be used to implement a range of IFC label models just by varying the rule table [Montagu CSF '13]
- Other potential uses
  - access control (clearance), memory protection, linearity, dynamic typing



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??

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

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

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

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

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

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