Domain Separation by Construction

Bill Harrison, Mark Tullsen, & James Hook

Pacific Software Research Center
OGI School of Science & Engineering
Oregon Health & Science University

Domain Separation by Construction

- Approach to language-based security based on monads:
  - Provide a modular algebra of effects,
  - Allow precise control of interaction of threads
  - Expressible in any higher-order functional programming language
- Outline development of a operating system kernel
  - Kernel obeys non-interference style property
    - Called "take separation"
  - Security property follows directly from well-understood (aka, *by construction*) properties of monads aid in its verification

Foundations of our approach

trace based security models

- Goguen/Meseguer 82,84
- McCullough 88
- McClean 94
- Zakathinos, et al., 95,97
- …

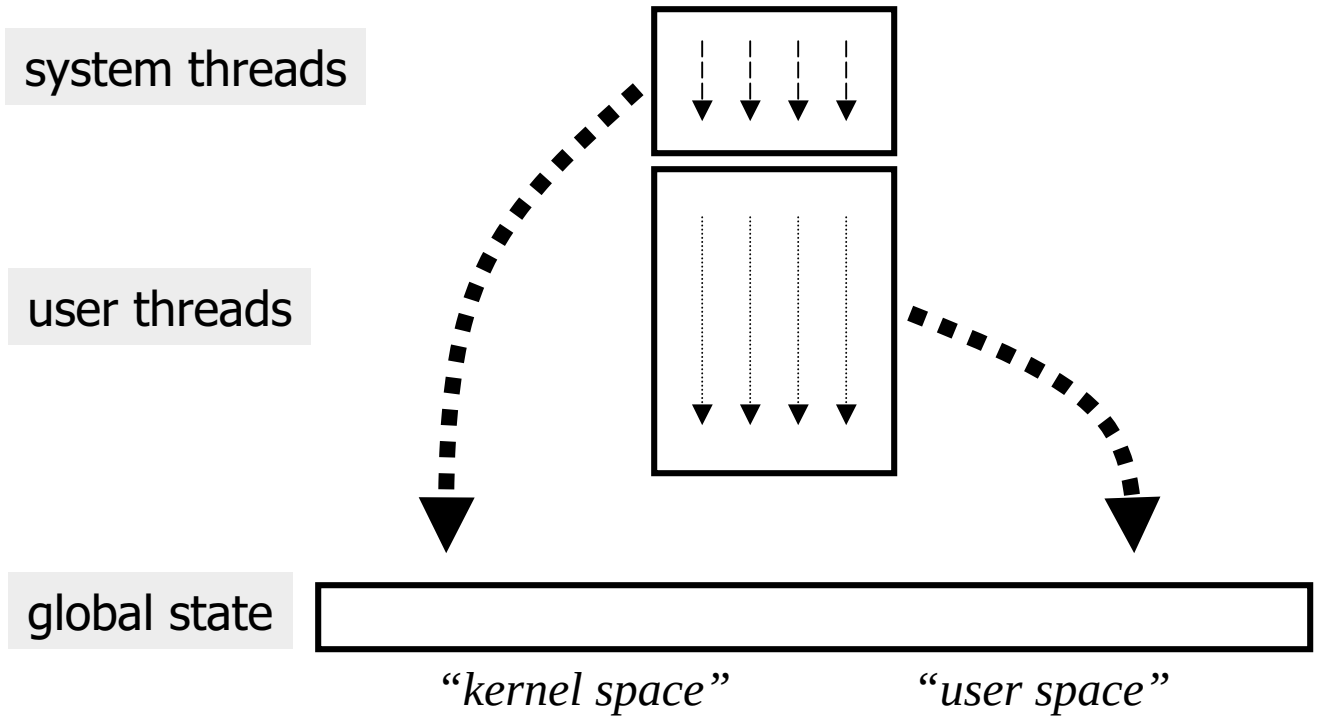
security by design

- Separability & Separation Kernels [Rushby]
- Java Sandboxing
- KSOS 79
- Fox Project 98
- White, et al., 00
- …

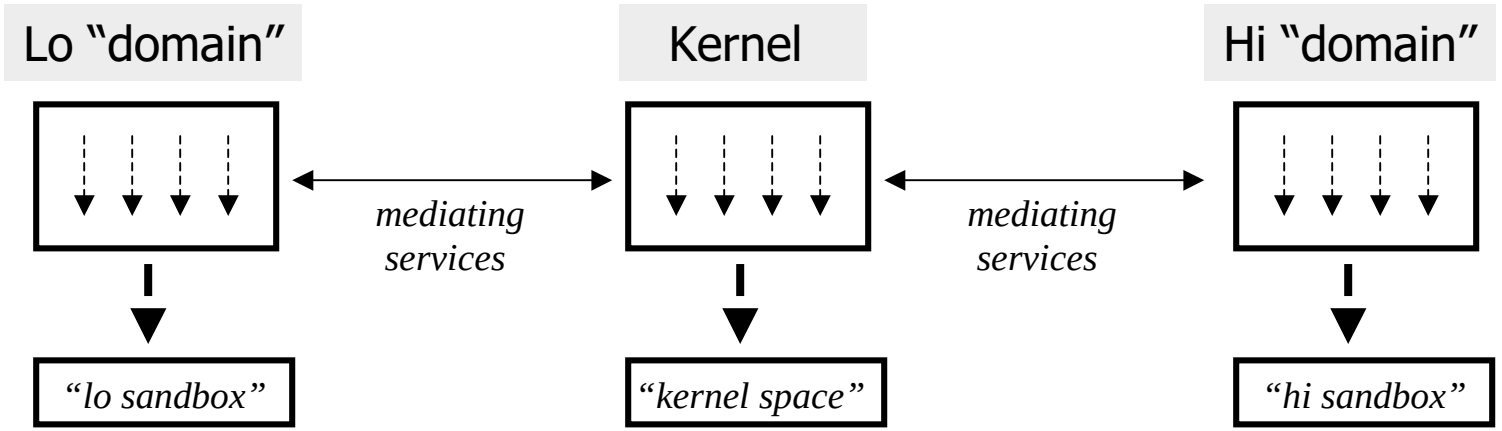
monadic semantics

- LiangHudakJones 95
- Moggi 89
- Harrison 01
- Papaspyrou 00
- …

Shared-state concurrency with global state



Separation Kernel Approach [Rushby82,81,…]



Security through separation:

- relies on tamed effects via state partitioning
  AND controlling interactions through trusted kernel services
- supports a "divide & conquer" approach to verification

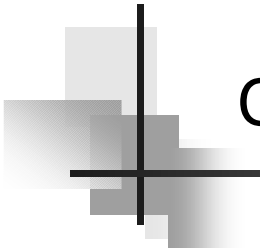
Trace-based models

An **event system** S is a tuple $\langle E, T, I, O \rangle$ where

$$
\begin{array}{rcll}
T & \subseteq & E^* & (T \text{ is the set of permissible traces}) \\
I \cup O & \subseteq & E & \\
I \cap O & = & \phi & (\text{I/O events are distinct}) \\
X \downarrow_Y & = & \text{subseq of } X \text{ with no } y \in Y & (\text{“view” from } Y) \\
E & = & \text{Lo} \cup \text{Hi} \quad \& \quad \text{Lo} \cap \text{Hi} = \phi &
\end{array}
$$

Ex: Generalized Non-interference [GoguenMeseguer]

“Changes in high-level inputs only result in changes to high-level outputs”



Our approach to language based semantics

Instead of traces of abstract Lo and Hi events:

$$h_0, l_0, \ldots, h_n, l_n$$

consider the imperative *program*:

$$h_0 \ ; \ l_0 \ ; \ldots ; \ h_n \ ; \ l_n$$

with (monadic) denotational semantics:

$$[\![-]\!] : (\mathit{Beh}_{\mathsf{Lo}} + \mathit{Beh}_{\mathsf{Hi}}) \to \mathsf{R}()$$

Monad R encapsulates imperative effects and a notion of concurrency called *resumptions* [Plotkin76,Schmidt86,Moggi89].

Take separation (first cut)

For any initial sequence of interleaved Hi *and* Lo *operations, the* Lo *operations should be "oblivious" to the high security operations:*

$$[\![h_0 \; ; \; l_0 \; ; \ldots ; \; h_n \; ; \; l_n]\!] \gg maskHi$$
$$= \quad [\![l_0 \; ; \ldots ; \; l_n]\!] \gg maskHi$$

What is a monad?

A **monad** is an algebra of *effects* (state, exceptions, concurrency,…)

$$\begin{array}{ll} \mathsf{M} & \text{(type constructor)} \\ \eta_{\mathsf{M}} : a \to \mathsf{M}\, a & \text{(unit)} \\ \star_{\mathsf{M}} : \mathsf{M}\, a \to (a \to \mathsf{M}\, b) \to \mathsf{M}\, b & \text{(bind)} \\ \gg_{\mathsf{M}} : \mathsf{M}\, () \to \mathsf{M}\, () \to \mathsf{M}\, () & \text{(sequence)} \end{array}$$

Additionally, a state monad, St, has the **effects**:

$$\begin{array}{ll} \mathsf{u} : (Sto \to Sto) \to \mathsf{St}\, () & \text{(update)} \\ \mathsf{g} : \mathsf{St}\, () & \text{(get)} \end{array}$$

Properties "by construction" of state monads

$$\begin{array}{rcll} \mathsf{u}\, f \star \lambda\_.\mathsf{u}\, g & = & \mathsf{u}\,(g \circ f) & \text{(sequencing)} \\ \mathsf{g} \star \lambda\sigma_0.\mathsf{u}\,(\lambda\_.\sigma_0) & = & \eta() & \text{(cancellation)} \end{array}$$

Monad transformers are constructors for monads

Monad transformer, (StateT *Sto*), adds effects to existing monad M

$$\mathsf{St} = \mathsf{StateT}\ Sto\ \mathsf{M}$$

Monad St2 with two states H, L

$$\mathsf{St2} = \mathsf{StateT}\ L\ (\mathsf{StateT}\ H\ \mathsf{M})$$

$\mathsf{u}_L : (L \to L) \to \mathsf{St2}\ ()$ $\qquad$ $\mathsf{u}_H : (H \to H) \to \mathsf{St2}\ ()$

$\mathsf{g}_L : \mathsf{St2}\ L$ $\qquad$ $\mathsf{g}_H : \mathsf{St2}\ H$

Another by-construction property: "atomic non-interference"

$$(u_L\ f) \gg_{\mathsf{St2}} (u_H\ g) \ = \ (u_H\ g) \gg_{\mathsf{St2}} (u_L\ f)$$

Primer on resumption-based concurrency

- Consider two simple threads: $a = [a_0; a_1]$ and $b = [b_0]$
- *Concurrency as interleaving*: $(a \parallel b)$ means:

$$\{[a_0; a_1; b_0], [a_0; b_0; a_1], [b_0; a_0; a_1]\}$$

- Resumption monad transformer:

$$\begin{array}{lcl} ResT\,\mathsf{M}\,a & = & \mu R.D\,a + P\,(\mathsf{M}(R\ a)) \\ step & : & \mathsf{M}\,a \to \mathsf{ResT}\,\mathsf{M}\,a \\ step\ \varphi & = & P(\varphi \star_{\mathsf{M}} \lambda v.\eta_{\mathsf{M}}(Dv)) \end{array}$$

D = "done"
P = "pause"

- Now, $(a \parallel b)$ means the set of:

$$\begin{array}{l} (step\ a_0) \gg (step\ a_1) \gg (step\ b_0) \\ (step\ a_0) \gg (step\ b_0) \gg (step\ a_1) \\ (step\ b_0) \gg (step\ a_0) \gg (step\ a_1) \end{array}$$

Monadic event systems

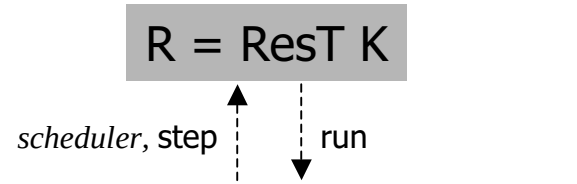
- **Events** are effects in state monads
  - Update operation. $u : (\mathrm{Sto} \to \mathrm{Sto}) \to \mathrm{M}\ ()$
  - Get operation. $g : \mathrm{M}\ \mathrm{Sto}$
  - Monad transformers give us "interaction rules" by construction
- **Traces** represented by "resumption threads"
- **Domain separation** by associating different domains with different state monad transformers
- This approach unifies "security by design" with formal trace-based security models
  - Implementations directly in higher-order typed functional languages
  - Reason about system specs at the level of denotational semantics
  - Promise of the approach: Scalability/Modularity through monads & monad transformers

Monadic Event Systems

(a) *Multi-threading*

(b) *Kernel*

(c) *Separate Domains*



K = StateT L (StateT H M)

liftL liftH

Lo = StateT L M

Hi = StateT H M

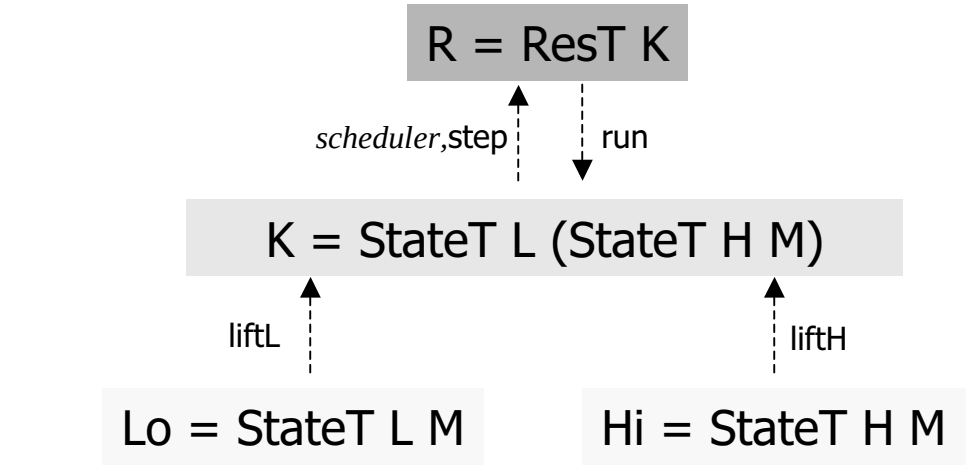
$\mathsf{u}_L \;:\; (L \to L) \to (\mathsf{StateT}\ L\ M)()$

$\mathsf{g}_L \;:\; (\mathsf{StateT}\ L\ M)L$

$\mathsf{u}_H \;:\; (H \to H) \to (\mathsf{StateT}\ H\ M)()$

$\mathsf{g}_H \;:\; (\mathsf{StateT}\ H\ M)$

Monadic Event Systems



$\mathsf{u}_L \;:\; (L \to L) \to (\mathsf{StateT}\ L\ M)\,()$
$\mathsf{g}_L \;:\; (\mathsf{StateT}\ L\ M)\,L$

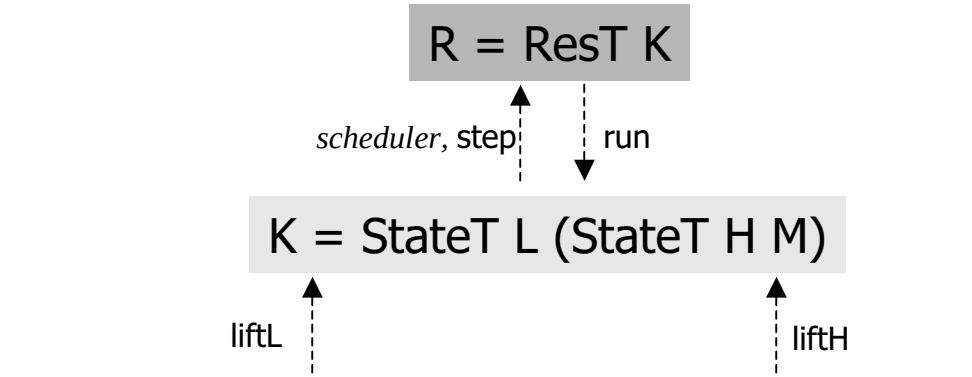
$\mathsf{u}_H \;:\; (H \to H) \to (\mathsf{StateT}\ H\ M)\,()$
$\mathsf{g}_H \;:\; (\mathsf{StateT}\ H\ M)$

Properties "by construction"

$$\mathsf{u}\, f \star \lambda\_.\mathsf{u}\, g \;=\; \mathsf{u}\,(g \circ f) \qquad \text{(sequencing)}$$

$$\mathsf{g} \star \lambda\sigma_0.\mathsf{u}\,(\lambda\_.\sigma_0) \;=\; \eta() \qquad \text{(cancellation)}$$

Monadic Event Systems (more "by construction" properties)



$\mathsf{u}_L \;:\; (L \to L) \to (\mathsf{StateT}\ L\ M)()$
$\mathsf{g}_L \;:\; (\mathsf{StateT}\ L\ M)L$

$\mathsf{u}_H \;:\; (H \to H) \to (\mathsf{StateT}\ H\ M)()$
$\mathsf{g}_H \;:\; (\mathsf{StateT}\ H\ M)$

Interaction rules (aka "atomic non-interference")

$$\mathit{liftL}\ (u_L\ f) \gg_{\mathsf{K}} \mathit{liftH}\ (u_H\ g) \;=\; \mathit{liftH}\ (u_H\ g) \gg_{\mathsf{K}} \mathit{liftL}\ (u_L\ f)$$

Language of behaviors *Beh*

Abstract Syntax for the Behavior Language.

$$
\begin{array}{lcl}
\mathit{Beh} & ::= & \mathit{Var} \texttt{:=} \mathit{Exp} \mid \\
 & & \texttt{skip} \mid \\
 & & \mathit{Beh} \texttt{;} \mathit{Beh} \mid \\
 & & \texttt{ite}\ \mathit{Exp}\ \mathit{Beh}\ \mathit{Beh} \mid \\
 & & \texttt{while}\ \mathit{Exp}\ \texttt{do}\ \mathit{Beh} \\
 & & \\
\mathit{Exp} & ::= & \mathit{Var} \mid \mathit{Integer}
\end{array}
$$

Basic Separation

- Refine monad transformer to reflect separation:

$$
\begin{array}{lcl}
\mathsf{ResT}\,\mathsf{M}\,a & = & \mu R.D\ a + P_{\mathsf{Lo}}\,(\mathsf{M}(\mathsf{R}\,a)) + P_{\mathsf{Hi}}\,(\mathsf{M}(\mathsf{R}\,a)) \\
\mathit{stepL} & : & \mathsf{M}\,a \to \mathsf{ResT}\,\mathsf{M}\,a \\
\mathit{stepH} & : & \mathsf{M}\,a \to \mathsf{ResT}\,\mathsf{M}\,a \\
\mathit{stepL}\ \varphi & = & P_{\mathsf{Lo}}(\varphi \star_{\mathsf{M}} \lambda v.\eta_{\mathsf{M}}(Dv)) \\
\mathit{stepH}\ \varphi & = & P_{\mathsf{Hi}}(\varphi \star_{\mathsf{M}} \lambda v.\eta_{\mathsf{M}}(Dv))
\end{array}
$$

- Create "security conscious" semantics

$$
\begin{array}{lcl}
\mathit{evL} : \mathit{Beh} \to \mathsf{R}\,() & & \\
\mathit{evH} : \mathit{Beh} \to \mathsf{R}\,() & & \\
\mathit{evL}\ (\texttt{x:=1}) & = & \mathit{stepL}\ ((\mathit{liftL} \circ \mathsf{u}_L)[x \mapsto 1]) \\
\mathit{evH}\ (\texttt{x:=1}) & = & \mathit{stepH}\ ((\mathit{liftH} \circ \mathsf{u}_H)[x \mapsto 1])
\end{array}
$$

Create two schedulers

Schedule **with** Hi & Lo events

$$\mathit{withHi} \;:\; \mathit{Beh} \to \mathit{Beh} \to \mathsf{R}\,()$$
$$\mathit{withHi}\ lo\ hi \;=\; \mathit{weave}\ (\mathit{evL}\ lo)\ (\mathit{evH}\ hi)$$

$$\mathit{weave} \;:\; \mathsf{R}\,() \to \mathsf{R}\,() \to \mathsf{R}\,()$$
$$\mathit{weave}\ [l_0, l_1, \ldots]\ [h_0, h_1, \ldots] = [l_0, h_0, l_1, h_1, \ldots]$$

Schedule **without** Hi & Lo events

$$\mathit{withoutHi} \;:\; \mathit{Beh} \to \mathsf{R}\,()$$
$$\mathit{withoutHi}\ lo \;=\; \mathit{evL}\ lo$$

$$\mathit{takeLo} \;:\; \mathit{Integer} \to \mathsf{R}() \to \mathsf{R}()$$
$$\mathit{takeLo}\ n\ [l_0, h_0, \ldots, l_n, \ldots] = [l_0, \ldots, l_n]$$

$$\mathit{run} \;:\; \mathsf{R}\,a \to \mathsf{K}\,a$$
$$\mathit{run}\ [e_0, e_1, \ldots] \;= e_0 \gg_{\mathsf{K}} e_1 \gg_{\mathsf{K}} \ldots$$

Security property

Theorem 5 (Take Separation)

Let $lo, hi \in \mathit{Beh}$, then for all natural numbers n,

$$\mathit{run}\ (\mathit{takeLo}\ n\ (\mathit{withoutHi}\ (\mathit{evL}\ lo)))\ \gg_{\mathsf{K}}\ \mathit{maskHi}$$
$$=\ \ \mathit{run}\ (\mathit{takeLo}\ n\ (\mathit{withHi}\ (\mathit{evL}\ lo)\ (\mathit{evH}\ hi)))\ \gg_{\mathsf{K}}\ \mathit{maskHi}$$

where

$$\mathit{maskHi} = \mathit{liftHi}(\mathsf{u}_H\ (\lambda\_.h_0))$$

for fixed $h_0 \in H$.

Proof Sketch

To show:

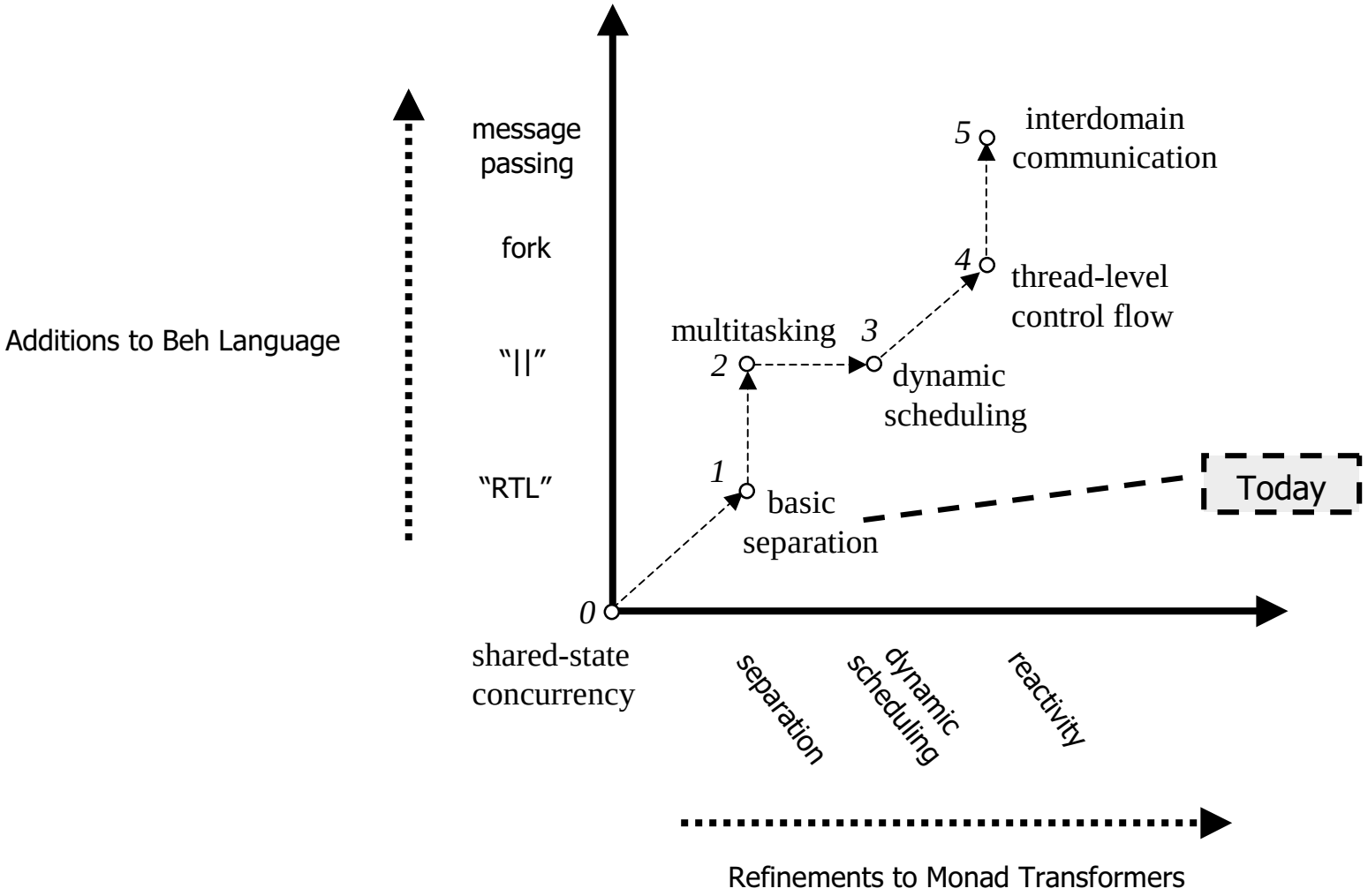
$$\mathit{run}\ (\mathit{takeLo}\ n\ (\mathit{withoutHi}\ (\mathit{evL}\ lo)))\ \gg_{\mathsf{K}}\ \mathit{maskHi}$$
$$=\ \ \mathit{run}\ (\mathit{takeLo}\ n\ (\mathit{withHi}\ (\mathit{evL}\ lo)\ (\mathit{evH}\ hi)))\gg_{\mathsf{K}}\ \mathit{maskHi}$$

Follows from sequencing, atomic non-inter., and monad laws

$$(l_1 \gg h_1 \gg \ldots \gg l_{(k+1)} \gg h_{(k+1)} \gg \eta() \gg\ \mathit{maskHi}$$
$$\{\text{sequencing}\} \quad = l_1 \gg h_1 \gg \ldots \gg l_{(k+1)} \gg \mathit{maskHi}$$
$$\{l_{(k+1)}\#\mathit{maskHi}\} \quad = l_1 \gg h_1 \gg \ldots \gg \mathit{maskHi} \gg l_{(k+1)}$$
$$\{\text{ind. hyp.}\} \quad = \underbrace{l_1 \gg \ldots}_{h_i \text{ excised}} \gg \mathit{maskHi} \gg l_{(k+1)}$$
$$\{l_i\#\mathit{maskHi}\} \quad = l_1 \gg \ldots \gg l_{(k+1)} \gg \mathit{maskHi}$$

* (x # y) means x,y are atomically non-interfering.

Enhancing system by refining the underlying monads



Domain Separation by Construction

- Our approach unifies "security by design" with formal trace-based security models
  - Implementations directly in higher-order typed functional languages
  - Reason about systems at the level of denotational semantics
- Promise of the approach: Scalability/Modularity of systems & their verifications through monads
  - Monads provide an algebraic theory of effects useful in formal specification & verification
    - Verification promoted via "by construction" properties of monads
    - System development through refinement to underlying monads
      - Cost of re-verification of a refined system can be minimal