Parametric Verification of Address Space Separation

Jason Franklin

with Sagar Chaki, Anupam Datta,

Jonathan M. McCune, Arvind Seshadri, and Amit Vasudevan

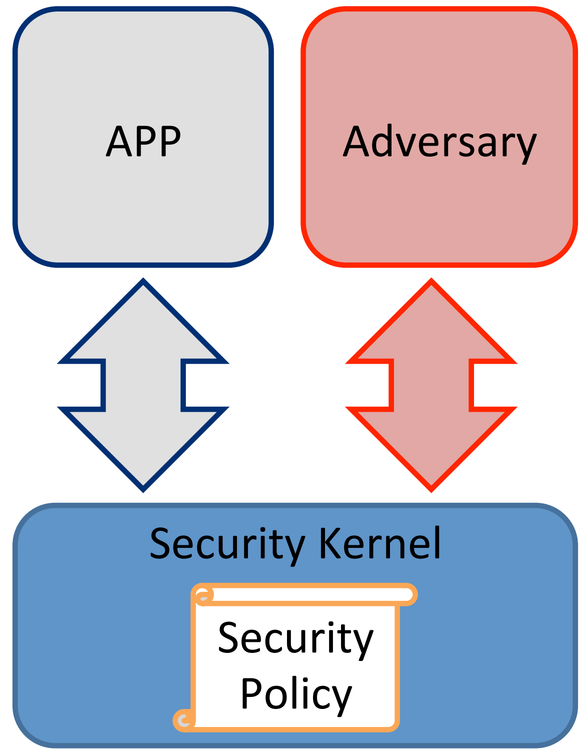
Cylab & SEI @ Carnegie Mellon University

Outline

Security Kernels	• **Definition & Importance**
Interfaces	• **How we model adversaries**
Parametricity	• **Data structure reduction**
Refinement	• **Verification of source code**

Security Kernels

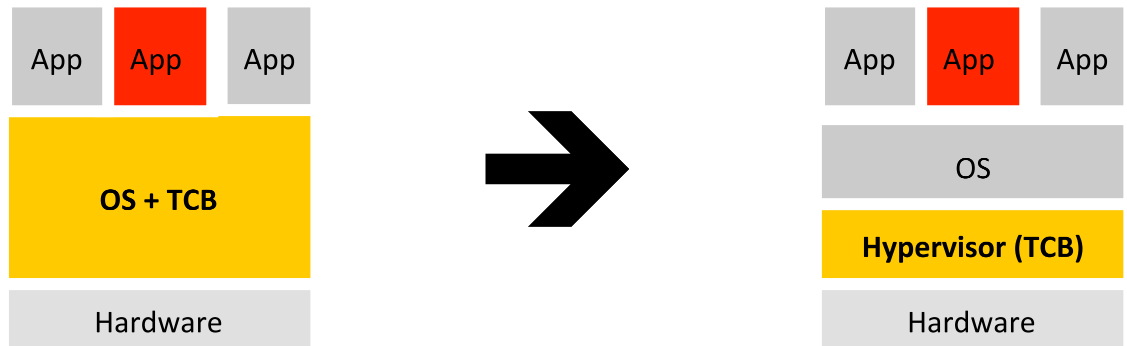
- **Security kernels** utilize **protection mechanisms** to prevent actions that violate a security policy
  - OSes, hypervisors and web browsers
- Uses: cloud computing, online banking, DRM (PS3), malware testing, national security?, etc.
- Critically important to verify absence of security bugs



State of the Art in Security Kernel Verification

- Manual/semi-automated verification
  - Similar goals, costly and time consuming (requires patience)
    - SRI HDM, InaJo, Gypsy, UCLA, PSOS, [Heitmeyer et al.], seL4
- Model checking work for security kernels
  - Study non-parametric verification
    - [Guttman et al.], [Lie et al.], [Mitchell et al.]
- Bug finding with adversaries
  - Unsound or incomplete methods
    - [Kidd et al.], [Emmi et al.], [Bugrara and Aiken]

State of the Art in High-Assurance Systems



- Security-critical components extracted and moved to hypervisor
  - Reduces system code size and interface
    - < 10k L.O.C. and < 10 system calls
  - Promising initial step to reduce complexity of verification
    - TRANGO Virtual Processor, Open Kernel Labs OKL4, VirtualLogix's VLX, **SecVisor (CMU), TrustVisor (CMU)**, and many others

Limitations of State of the Art

Limitations

- Manual verification effort for theorem proving is high
- Small TCBs alone don't enable automated formal verification
- Major source of complexity for model checking is size of data structures

Our Goal

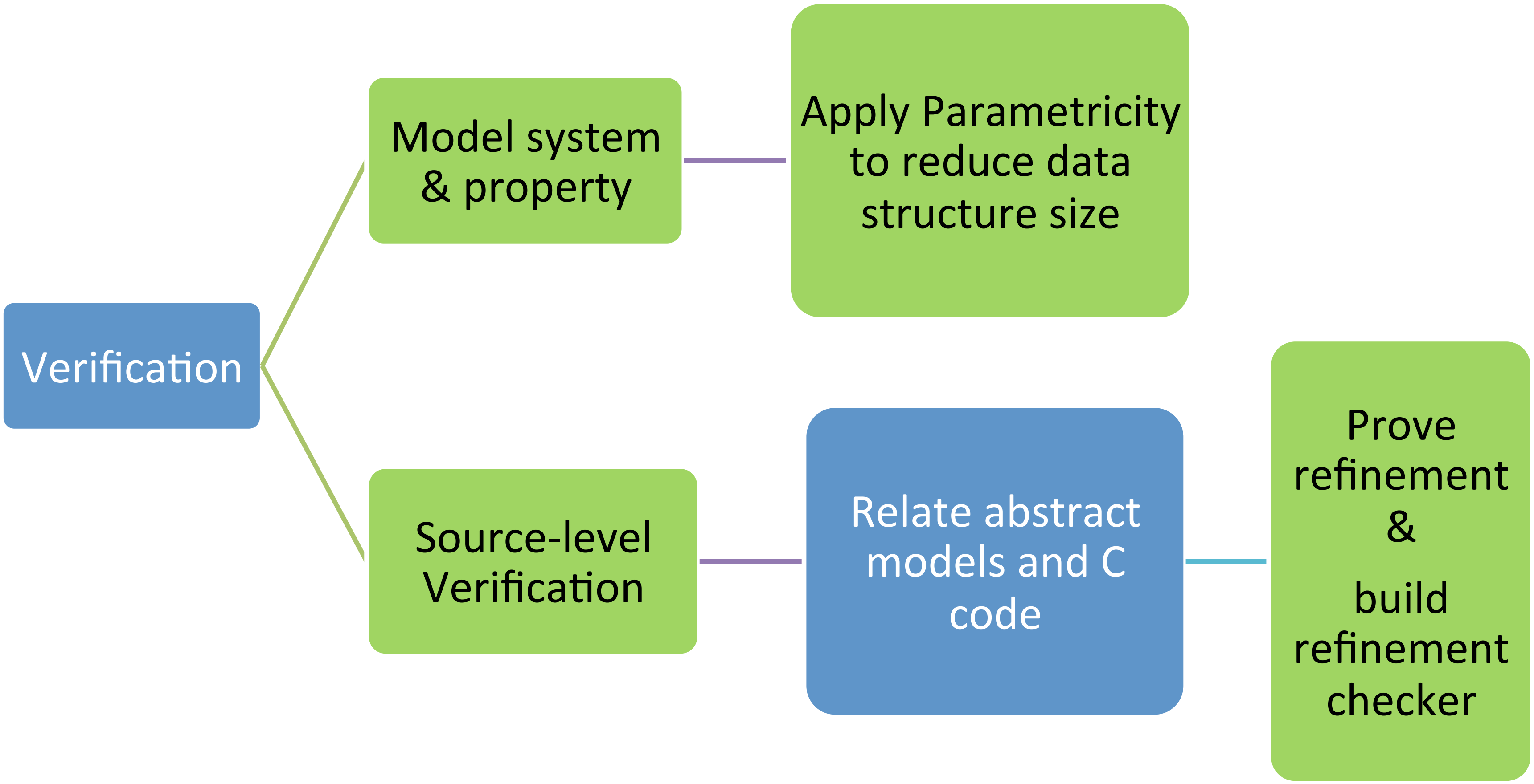
- Overcome limitations by exploiting structure of protection mechanisms

Approach

Automatic source-level verification of OS protection mechanisms can be realized by:

- **Parametrically** reasoning about protection data structures and
- Using **refinement** to carry verification down to source code level

Verification Process



Scope

Systems

- Protection mechanisms of Xen, TrustVisor, SecVisor
  - Manages protection data structures, performs permission and bounds checks

Properties

- Address separation, W xor X, access control

Adversary model

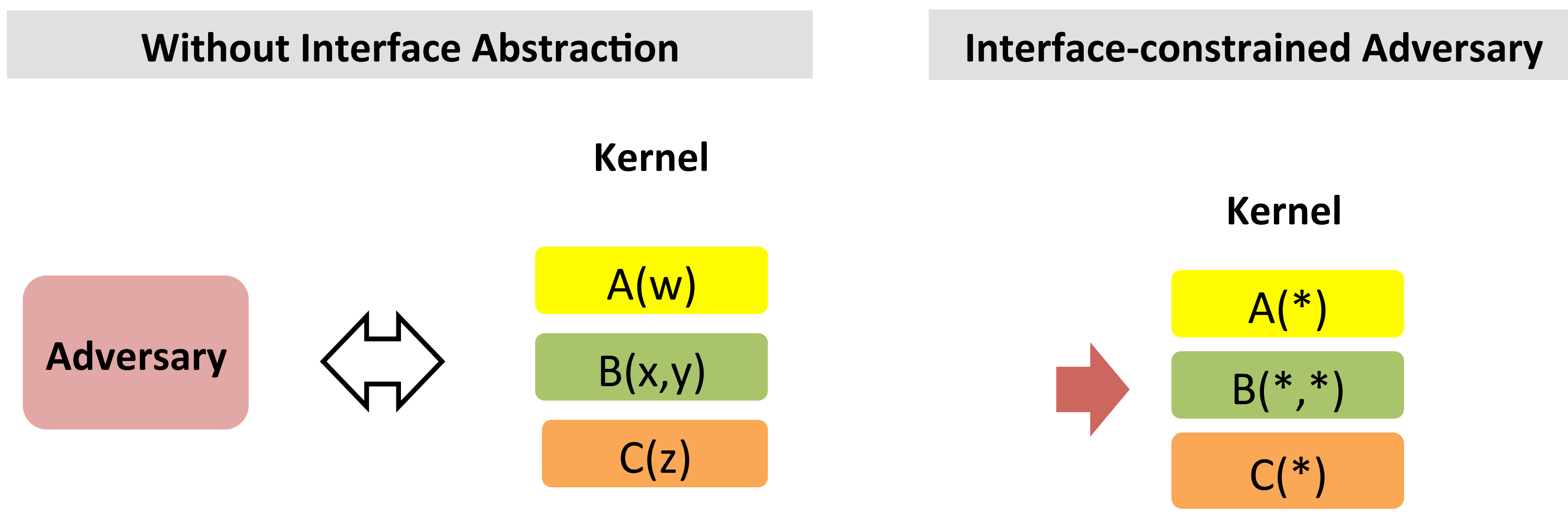
- Adversary constrained to system interface

Outline

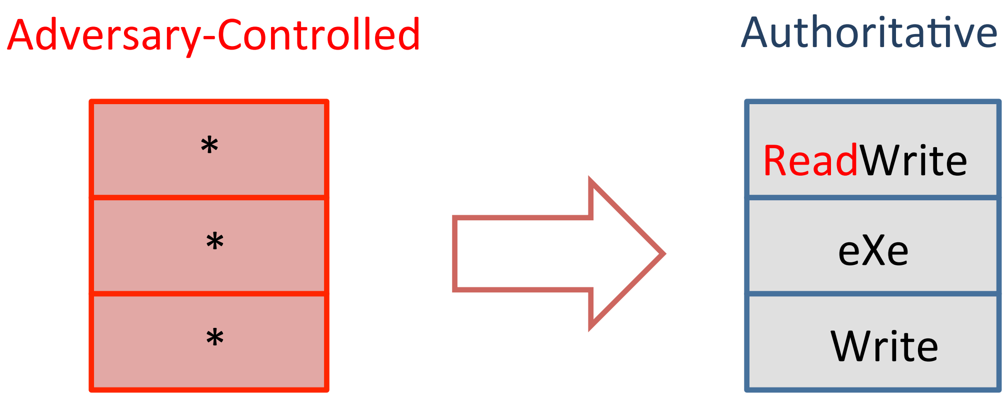
- Security Kernels
- Interfaces
  - **How we model adversaries**
- Parametricity
- Refinement

Interface-Constrained Adversary

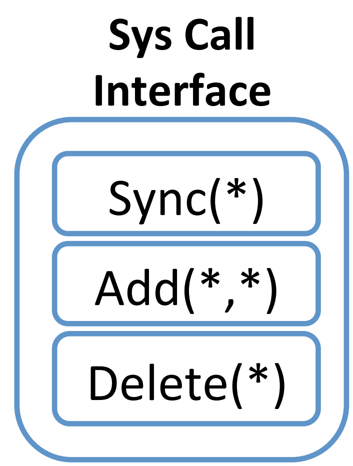
- Adversary model = arbitrary number of calls to system call interface with non-deterministic inputs
  - Reduces complexity of adversary models, eases model checking



Example



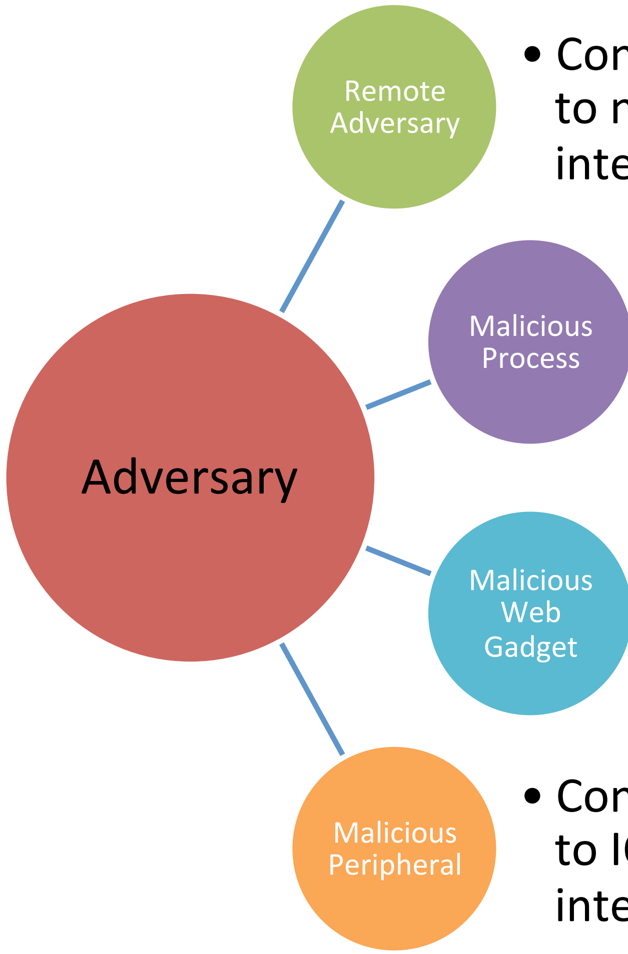

```
Sync ≡

foreach row do
  if (Secure) then
       copy
```



Key Insights:

- Kernel page table is adversary-controlled data structure
- Sync copies adversary-controlled data into memory
- Adversary is constrained to interface
  - I = {Sync(*) , Add(*,*) , Delete(*)}

Common Examples

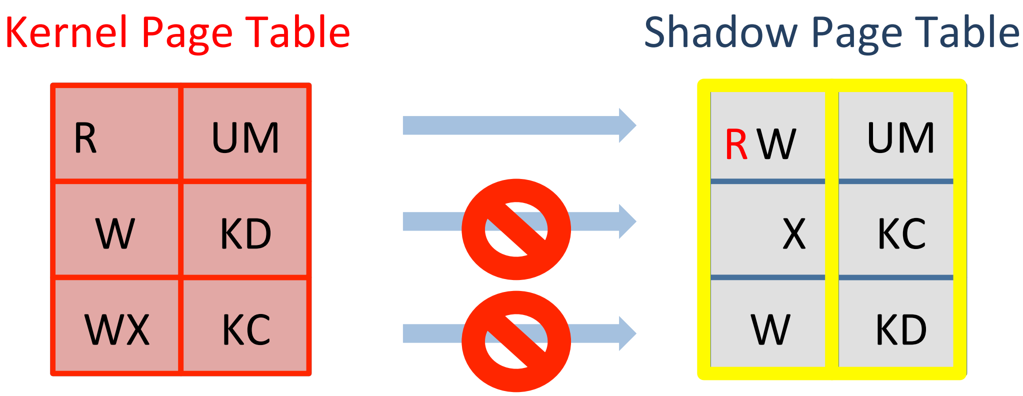


- Constrained to network interface
- Constrained to OS system call interface
- Constrained to browser interface
- Constrained to IOMMU interface

Outline

- Security Kernels
- Interfaces
- Parametricity
  - **Data structure reduction**
- Refinement

Protection Data Structures




```
SecVisor Sync ≡

foreach row do
  if (W XOR X) then
        Sync
```

UM=User Memory   KD=Kernel Data   KC=Kernel Code

Data Structure Size and Verification Complexity

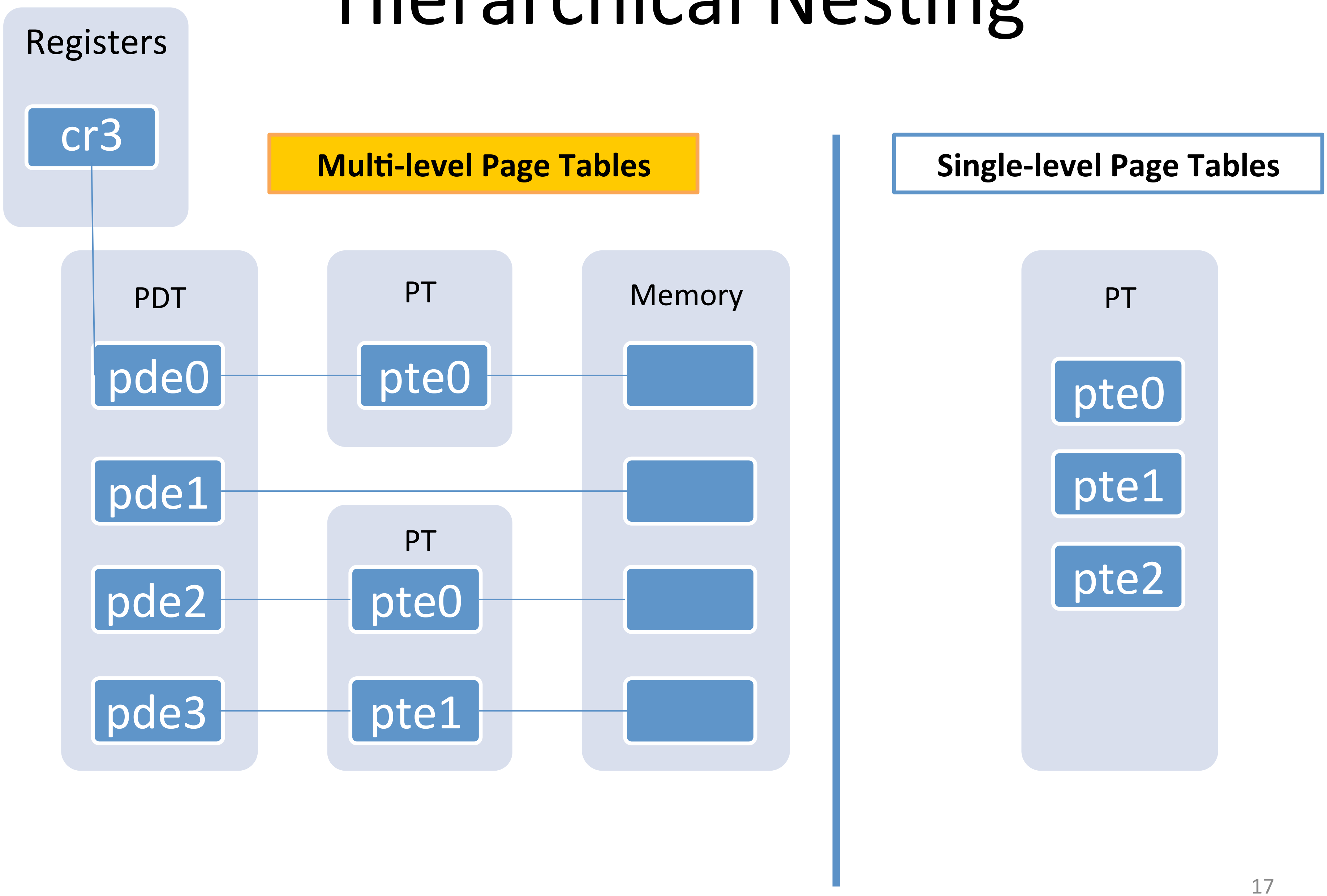
- Securit… …ructures
  - Page … …etc.
- Comple… …reases expone… …ture size

Goal: automated verification techniques that scale gracefully with increase in data structure size

Page Table Entries	States	Space	Time
3	55,000	8MB	2 sec
4	1,700,000	<256MB	360 sec
5	--	Out of Memory	--
1024	--	--	--

Murphi model checking SecVisor security hypervisor with increasing page table size

Hierarchical Nesting

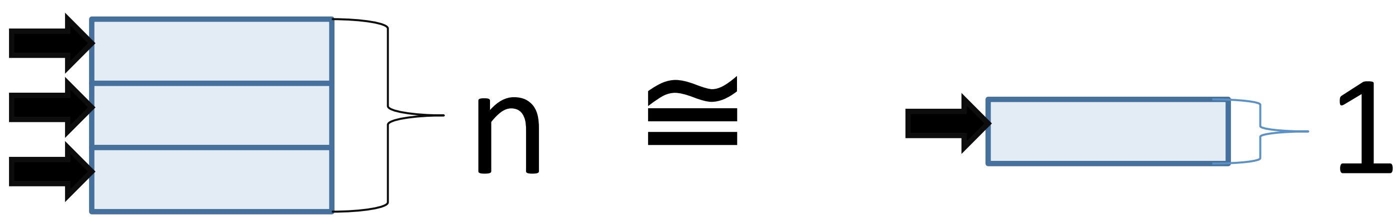


Parametricity of System Data Structures

```
shadow_paging_sync ≡

foreach row do
  if (Row_Check) then
      modify row;
```

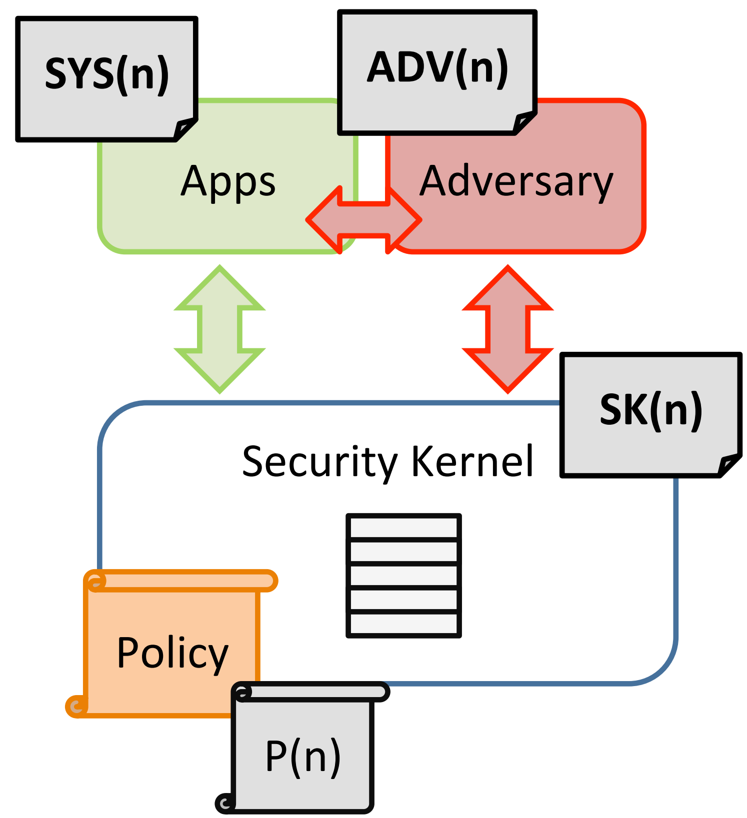
Row uniform

Row independent



Small Model Analysis: Modeling Systems and Properties

- Model kernel and adversaries
  - Parametric Guarded Command Language (PGCL)
- Express security properties
  - Parametric Temporal Specification Logic (PTSL)



Language Design Challenges

- Balancing expressiveness with small model analysis
  - Conditionals
  - Whole array ops
  - Assignment
  - Parallel and sequential composition
  - Non-deterministic update

```
foreach row do

if (Condition) then
   Set row = x;
```

- Distinctive features
  - Modeling systems and adversaries: whole array operations
  - Adversary: Non-deterministic updates

```
Adversary ≡

foreach row do
   row[0] = *;
```

Parametric Programming Language

Natural Numerals	$\mathtt{K}$			
Index Variable	$\mathtt{i}$			
Boolean Variables	$\mathtt{B}$			
Parametric Variable	$\mathtt{n}$			
Expressions	$\mathtt{E}$	$::=$	$\top \mid \bot \mid * \mid \mathtt{B} \mid \mathtt{E} \vee \mathtt{E} \mid \mathtt{E} \wedge \mathtt{E} \mid \neg \mathtt{E}$	
Parameterized Expressions	$\widehat{\mathtt{E}}$	$::=$	$\mathtt{E} \mid \boxed{\mathtt{P_{n,q}[i][K]}} \mid \widehat{\mathtt{E}} \vee \widehat{\mathtt{E}} \mid \widehat{\mathtt{E}} \wedge \widehat{\mathtt{E}} \mid \neg \widehat{\mathtt{E}}$	
Instantiated Guarded Commands	$\mathtt{G}$	$::=$	$\mathtt{GC(K)}$	
Guarded Commands	$\mathtt{GC}$	$::=$	$\mathtt{E}\ ?\ \mathtt{C}$	Simple guarded command
		$\mid$	$\mathtt{GC} \parallel \mathtt{GC}$	Parallel composition
Commands	$\mathtt{C}$	$::=$	$\mathtt{B} := \mathtt{E}$	Assignment
		$\mid$	$\mathtt{for\ i\ :\ P_{n,q}\ do}\ \widehat{\mathtt{E}}\ ?\ \widehat{\mathtt{C}}$	Parametric
		$\mid$	$\mathtt{C;C}$	Sequencing
Parameterized Commands	$\widehat{\mathtt{C}}$	$::=$	$\mathtt{P_{n,q}[i][K]} := \widehat{\mathtt{E}}$	Parameterized array assignment
		$\mid$	$\widehat{\mathtt{C}};\widehat{\mathtt{C}}$	Sequencing

- Language for modeling system & adversary
  - Finite number of Boolean variables
  - System parameter: n
  - Single parametric array: P of size n x q
  - Parametric loop
    - for i : P[n,q] do E ? C

Kernel Entry ≡
¬kernelmode ? kernelmode := ⊤;

for i : Pn,q do
 Pn,q[i][Write] = T ? Pn,q[i][eXe] := ⊥;

SecVisor Model

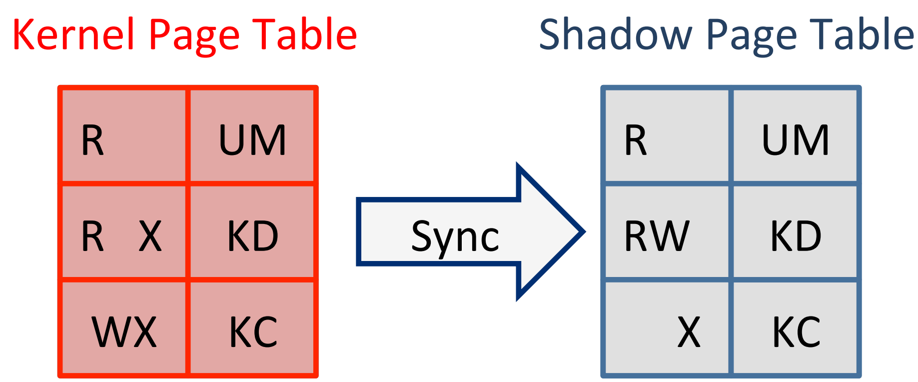
Kernel Entry ≡
¬kernelmode ? kernelmode := ⊤;

for i : Pn,q do
  Pn,q[i][SPTPA] = KC ?
    Pn,q[i][SPTRW] := ⊥;
    Pn,q[i][SPTX] := ⊤;

…

Sync ≡
⊤ ? for i : Pn,q do
  ⊤ ? Pn,q[i][SPTPA] := Pn,q[i][KPTPA]

Attacker ≡
⊤ ? for i : Pn,q do
  Pn,q[i][KPTPA] := *;
  Pn,q[i][KPTRW] := *;
  Pn,q[i][KPTX] := *



Expressive Specification Logic

Basic Propositions	BP	::=	b , b ∈ B
		|	¬BP
		|	BP ∧ BP
Parametric Propositions	PP(i)	::=	$\{P_{n,q}[i][l] \mid \lceil l \rceil \leq \lceil q \rceil\}$
		|	¬PP(i)
		|	PP(i) ∧ PP(i)

Propositio

- Basic range
- Parametric

PTSL Path Formulas	TLPF	::=	TLF	“state formula”
		|	TLF ∧ TLF	“conjunction”
				unction”
				e next state”
				l”

Execution Integrity:
In kernel mode, only kernel code should be executable.
It is stated as follows:

Pexec == MODE=KERNEL⇒(∀ i P[i][eXe]⇒(P[i][CodeType] = KC))

Universal Stat

				their negations”
		|	A TLPF	“for all computation paths”

Existential State Formulas	ESF	::=	BP
		|	∃i.PP(i)
		|	BP ∧ ∃i.PP(i)
Generic State Formulas	GSF	::=	USF
		|	ESF
		|	USF ∧ ESF

Reachability properties

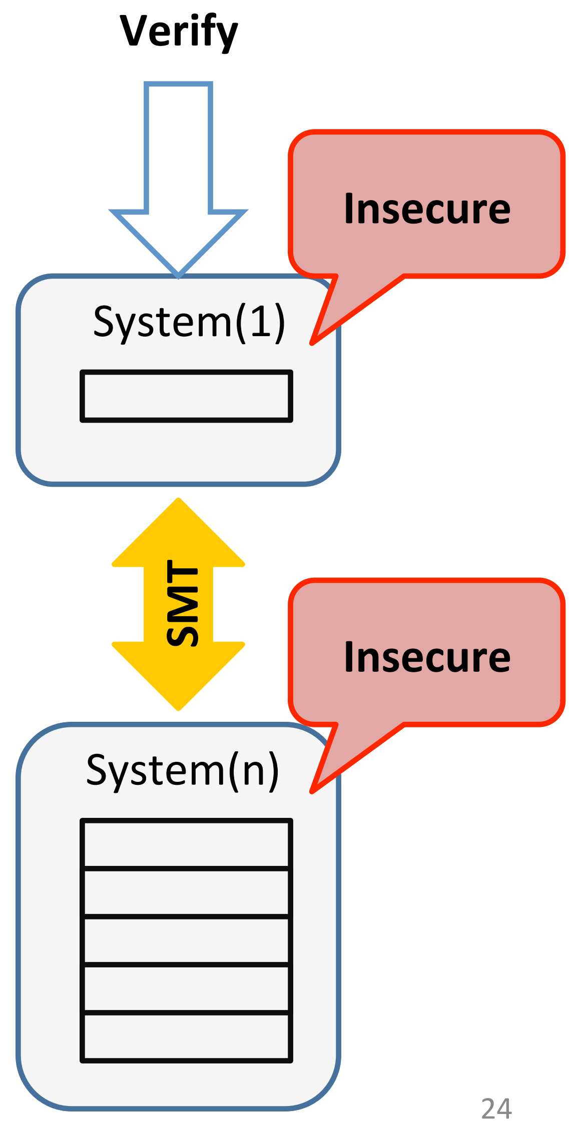
- State formulas
  - Universal, existential, and generic

Temporal logic specifications

- Path formulas
  - Subset of ACTL* with USF as atomic propositions

Small Model Theorems

- Relate properties of System(1) to System(n), for all finite n
  - **Sound:** If small model is secure then large model is secure
  - **Complete:** If small model is insecure then large model is insecure



Small Model Safety Theorem

- System model
  - Let gc(k) be any instantiated guarded command (i.e., any well-formed program)
- Security property
  - Let φ in GSF be any generic state formula
    - Forall i. P(i) , Exists i. P(i), or conjunctions of
- Initial state
  - Let Init in USF be any universal state formula (For all i. P(i))

- Definition: model exhibits φ if contains reachable state that satisfies φ

- **Theorem:** M(gc(k), Init) exhibits φ iff M(gc(1), Init) exhibits φ

- Thms with different initial conditions & properties in [Oakland2010]

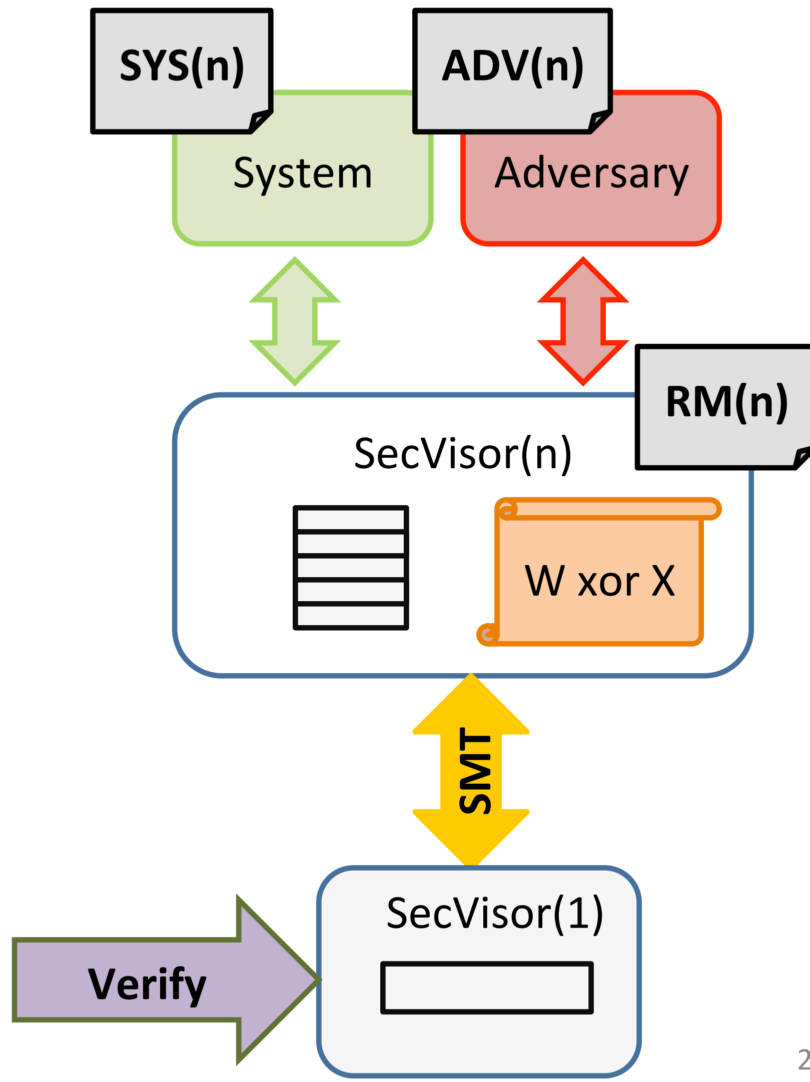
Small Model Analysis

Initial condition:
SecVisor starts in kernel mode and only kernel code is executable

mode = kernel AND
FOREACH page in SPT, if eXe then
page maps kernel code

Execution Integrity:
In kernel mode, only kernel code should be executable.

If mode = kernel then
FOREACH page in SPT, if eXe then
page maps kernel code



Verification Results

- SecVisor (Shadowvisor, sHype, Xen), adversary, and properties expressible
  - Small Model Theorems apply
- Translate to Murphi, verify
  - **Two vulnerabilities**, repaired, verified
    - Two more in ShadowVisor

Sync ≡
⊤ ? for i : Pn,q do
⊤ ?
Pn,q[i][SPTPA] :=Pn,q[i][KPTPA]

Secure Sync ≡
⊤ ? for i : Pn,q do
(¬Pn,q[i][SPTX] ∧ ¬(Pn,q[i][KPTPA] = KC)) ?
Pn,q[i][SPTPA] := Pn,q[i][KPTPA]

Extending Parametricity Results

Complexities and Solutions

- Multiple, linked, parametric arrays
  - Extended PGCL to handle multiple parametric arrays
  - Added nested quantifiers to PTSL
  - New small model theorems

```
page_fault (u32 addr) {

    pdt = get_pdt(addr);

    if (pdt is ADDR) {
        if (pdt < MAX)
            copy;
    } else {
        if (getpde(addr) < MAX)
            copy;
    }
}
```

Related Work

- Parametric verification for correctness
  - Missing whole array operators (adversary) or less efficient
    - [Lazic et al.] and [Emerson and Kahlon]
  - Incomplete methods (environment abstraction)
    - [Clarke et al.] and [Talupur et al.]
- Parametric verification for security
  - Focus on security protocols
    - [Lowe et al.], [Roscoe and Broadfoot], [Durgin et al.], [Millen]

Outline

Security Kernels

Interfaces

Parametricity

Refinement

- **Verification of source code**

Towards Source Level Verification

C code

```
page_fault (u32 addr) {
    pdt = get_pdt(addr);
    if (pdt.ADDR) {
        if (pdt < MAX)
            copy;
    } else {
        if (getpde(addr) < MAX)
            copy;
    }
}
```

Guarded Command Representation

```
page_fault ≡
  pdt.ADDR && pdt < MAX ?
      Pn,k[i][ADDR] := Pn,k[i][ADDR];

             DR && pde(pdt) < MAX ?
             k[i][ADDR] := Pn,k[i][ADDR];
```

1. Convert

Refinement Theorem

2. Check language containment

Language containment ➔ refinement

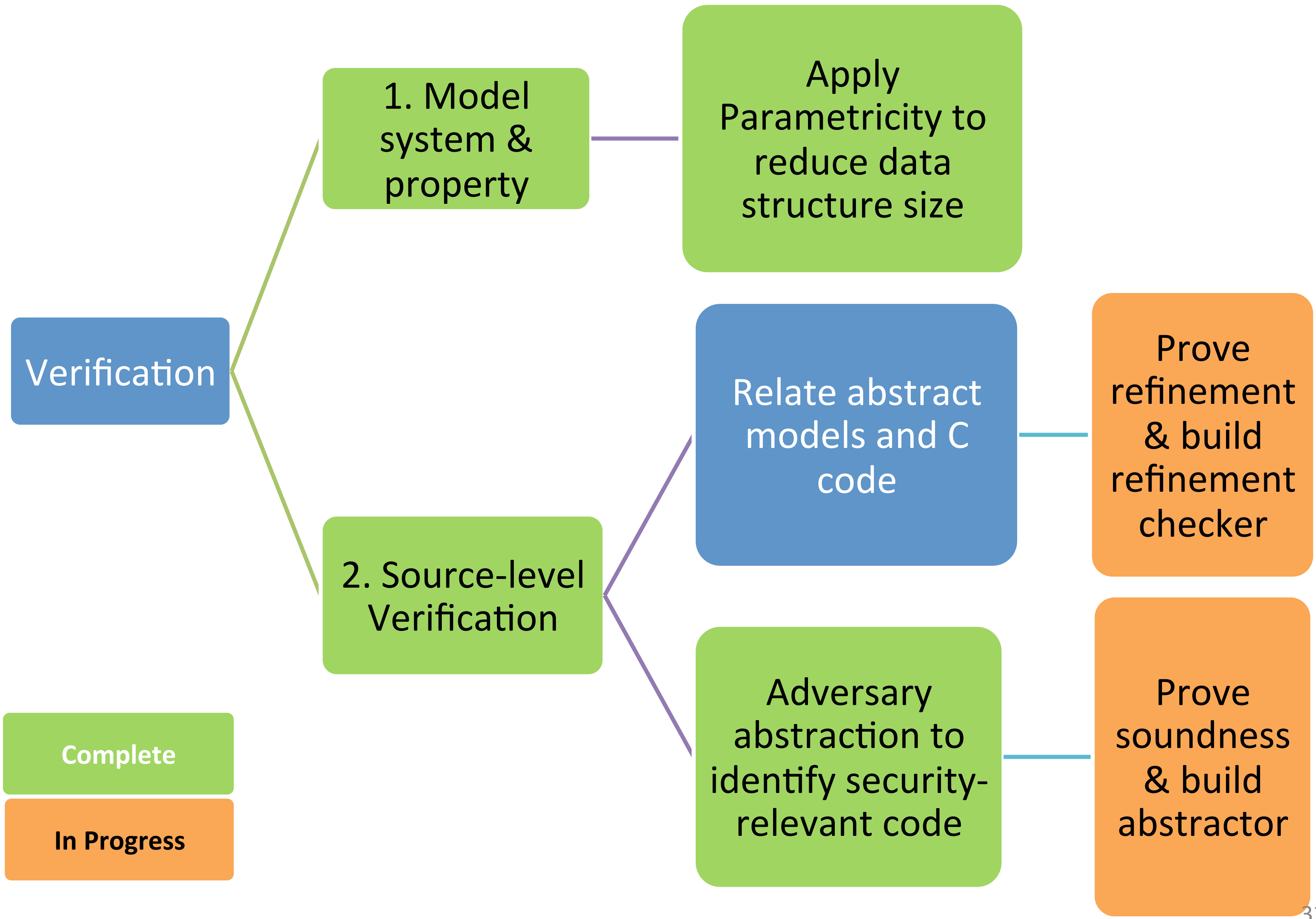
Convert C code to guarded commands (GC)

Check language containment to prove GC is in PGCL

Yes, results apply.

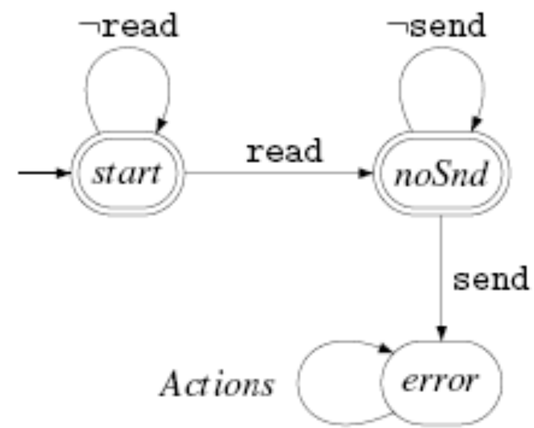
No, modify or results do not apply.

Verification



Expressiveness and Limitations

- PGCL/PTSL can model:
  – Protection mechanisms that are row independent and row uniform
  – Policies that are expressible as FSA over rows (safety properties)



$$
\begin{array}{l}
\texttt{universal\_reference\_monitor} \equiv \\
\quad \top\ ? \\
\quad \texttt{for i : } \mathrm{P_{n,q}} \texttt{ do} \\
\qquad \mathrm{P_{n,q}[i]}[\sigma'_1] := \bot; \ldots; \mathrm{P_{n,q}[i]}[\sigma'_S] := \bot; \\
\quad \texttt{for i : } \mathrm{P_{n,q}} \texttt{ do} \\
\qquad \mathrm{P_{n,q}[i]}[\sigma_1] \wedge \mathrm{P_{n,q}[i][a_1]} \wedge b_{\sigma_1, a_1, \sigma_1}\ ?\ \mathrm{P_{n,q}[i]}[\sigma'_1] := \top; \\
\qquad\qquad \vdots \\
\quad \texttt{for i : } \mathrm{P_{n,q}} \texttt{ do} \\
\qquad \mathrm{P_{n,q}[i]}[\sigma_S] \wedge \mathrm{P_{n,q}[i][a_A]} \wedge b_{\sigma_S, a_A, \sigma_S}\ ?\ \mathrm{P_{n,q}[i]}[\sigma'_S] := \top; \\
\quad \texttt{for i : } \mathrm{P_{n,q}} \texttt{ do} \\
\qquad \mathrm{P_{n,q}[i]}[\sigma_1] := \mathrm{P_{n,q}[i]}[\sigma'_1]; \ldots; \mathrm{P_{n,q}[i]}[\sigma_S] := \mathrm{P_{n,q}[i]}[\sigma'_S];
\end{array}
$$

- Developed compilation to convert FSA policy to PGCL reference monitor

Assumptions and/or Limitations

- Verification
  - Applies to row-uniform, hierarchical-row independent systems (expressible in PGCL)
  - Properties expressible in large subset of ACTL*/X
  - Currently at model level and some source-level guarantees
  - Assumptions about semantics of C, correctness of model checker, translation tools, and of proofs
- Validation
  - Security properties might not be right properties or strong enough
  - Adversary model may not match reality

Related Work

- Model checking for security
  - Study non-parametric verification of secure systems
    - [Guttman et al.], [Lie et al.], [Mitchell et al.]
- Bug finding with adversaries
  - Unsound or incomplete methods
    - [Kidd et al.], [Emmi et al.]
- Operating system verification
  - Manual/semi-automated verification
    - [Walker et al.], [Heitmeyer et al.], [Klein et al.]

Questions?
More info @
http://www.cs.cmu.edu/~jfrankli

- Towards scalable automated source-level verification of protection mechanisms
  - Adversary **abstraction** reduces model complexity
  - **Parametric** reasoning reduces data complexity
    - Small model theorems relate small/large models
    - Application to SecVisor, sHype, Shadowvisor, & Xen
  - **Refinement** theorem pushes guarantees to source
- Outlined path to source level verification
  - Building models is time-consuming, costly, and error-prone
  - In progress, proof of refinement theorem