U.S. Air Force AFRL-SBIR FA8750-07-C-0049

CrossCheck: Improving System Confidence through High-Speed Dynamic Property Checking

Jonathan Springer* (PI), James Ezick*, David Wohlford*, Matthew Craven†, Rick Buskens†

(*) Reservoir Labs
632 Broadway, #803
New York, NY 10012
(212) 780-0527
springer@reservoir.com

(†) Lockheed Martin
3 Executive Campus
Cherry Hill, NJ 08002
(856) 792-9019
matthew.craven@lmco.com

Unclassified, DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited. This material is based upon works supported by the Department of Defense under contract numbers FA8750-06-C-0133 and FA8750-07-C-0049. Any opinions, findings and conclusions expressed in this material are those of Reservoir Labs, and do not necessarily reflect the views of the Department of Defense.

Outline

- Problem area
- Dynamic specification checking approach
- Use cases and applications
- Technology details
- Remarks and conclusions

Problem Area: System Complexity

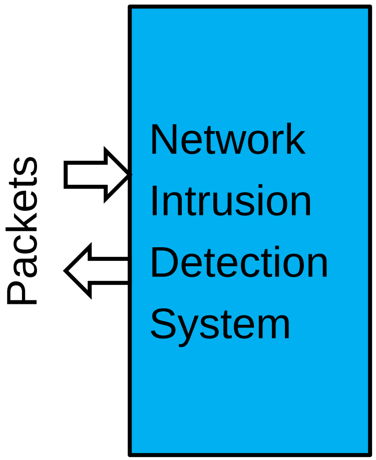
- We rely on increasingly complex systems
  - Large amount of software, large numbers of developers
- Systems are getting more autonomous
  - Scale leads to goal-directed behavior
  - Deployment environment requires goal-directed behavior

→ Increased possibility of defects

→ Increased impact of defects
  - Incremental time and money: failures during development and testing
  - Catastrophic: failures during deployment

A Dynamic Checking Exemplar

- **Well-known problem: Malicious Internet traffic**
- **Well-known solution: Packet-filtering appliances**
  - **Network Intrusion Detection System (NIDS)**
- **NIDS problem area has several characteristic features:**



……Packets of data to process

……Properties expressed as patterns/specifications

……Properties can be complex (e.g. protocols)

……Need for (very) fast matching

……Static or offline checking not appropriate

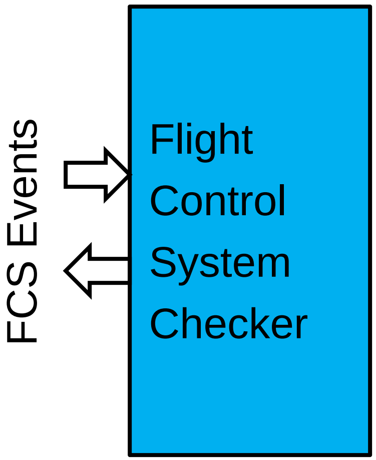
```
alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS zone transfer TCP";
flow:to_server,established; content:"|00 00 FC|";
```

Snort specification

Generalized Dynamic Checking

- **Many problems in different domains parallel this structure**
  - **E.g., verifying the behavior of a Flight Control System**

- **Flight Control System checking problem characteristics:**



……Sensor, actuator, & controller events to process

……Properties expressed as patterns/specifications

……Properties can be complex

……Need for fast matching

……Static checking helps, but often not a solution

```
LongAccel <- AccelHigh ; NoDecel* ; ContinuedAccel ;;
AccelRule := LongAccel,
    group::0, attr::{oldest_only, rollback, match_recover},
    recover::<LongAccel_recover_f>,
    desc::"Check acceleration does not exceed duration limit" ;;
```

CrossCheck specification

CrossCheck

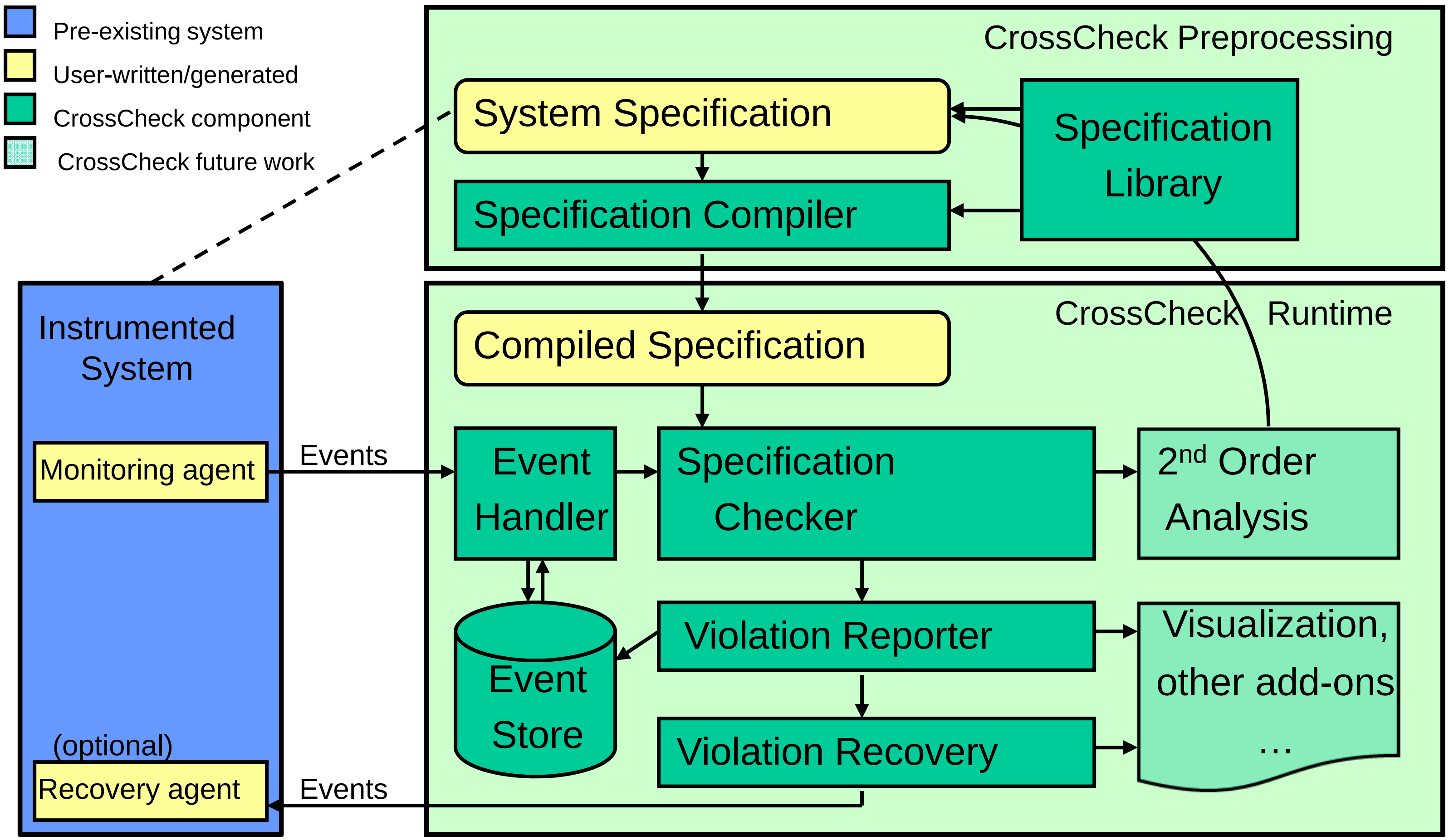
Need a common framework to address these problems

- **We have developed CrossCheck, a platform for dynamic checking of formal specifications**
  - **Specification target is any system of inputs and outputs with behavior complex enough that it does not admit static proof of correctness**
- **Design goals:**
  - **Be applicable in a wide variety of use cases**
  - **Scale to high data rates**
  - **Be flexible and practical for specifying properties of interest**

Specifications and Checking

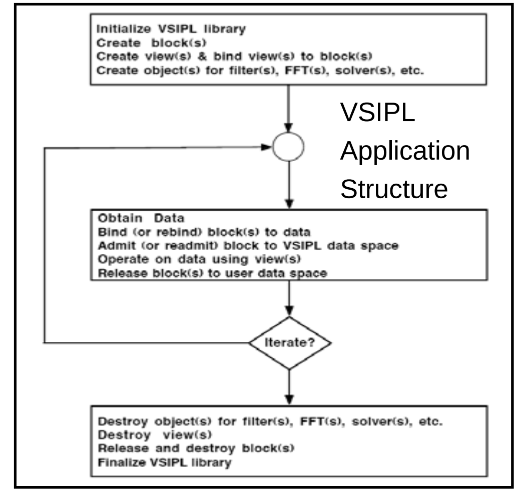
- CrossCheck specifications operate on "Event" abstraction
  - Events are domain-specific
- Specifications are written by a developer for characterizing behavior of a system
  - Written in a formal language (not English)
  - Can come from: requirements documents, expert knowledge, previous failures, …
- Specifications are compiled into a form that can be efficiently checked at system runtime
  - Final form is compiled C code, for platform flexibility and performance
  - Works with a runtime that manages all the common parts of checking
    - Recording events, calling the compiled specification code, reporting violations, etc.

CrossCheck System Architecture



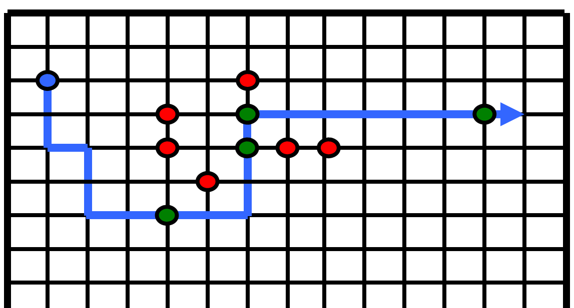
CrossCheck Use Cases

Flight Control System Checking



Software Interface Checking

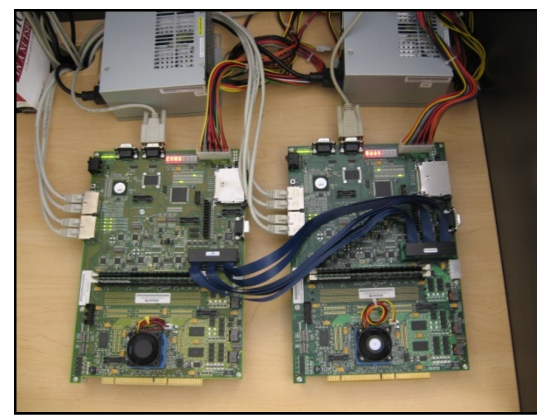
CrossCheck



Cognitive Application Checking

F(3)=5	F(3)=5	F(3)=5
F(3)=5	F(3)=5	F(3)=5
F(3)=5	F(3)=6	F(3)=5
F(3)=5	F(3)=5	F(3)=5

Test System Δ Injection



IDS Protocol Checking

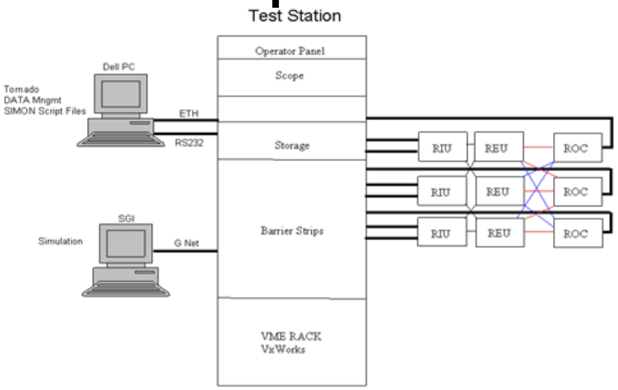
Online Verification of Flight Control System

- Flight control systems offer good use case for dynamic checking
  - Require high reliability
  - Static checking often not feasible (intractable)
- Write specifications of control system behavior
  - Encode requirements as specifications
  - Encode failure modes as (negative) specifications
- CrossCheck offers a means of independent verification, operating outside the FCS
  - Can be useful for formal requirements
    - E.g., RTCA/DO-178B
- Goal is detect designed-in failure modes
  - Orthogonal to hardware redundancy
    - E.g. TMR

Online Verification of Flight Control System

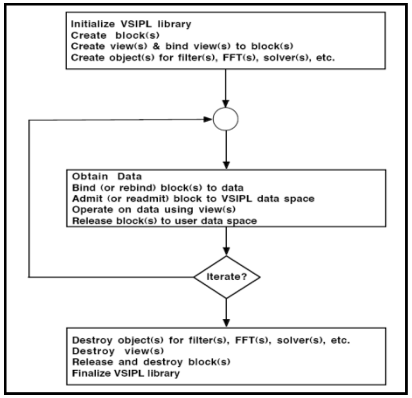
How to apply?

- Control systems involve the interaction of sensors, actuators, and control devices
  - All communicate via formatted data streams
  - Formatting typically reduces to large collections of key/data pairs
  - Easily described as CrossCheck Events
- Emerging flight architectures use standard network interfaces to communicate
  - Simplifies interfacing to CrossCheck runtime component
- Can operate at
  - Development time
  - Test stand (ETS) time
  - Deployment time



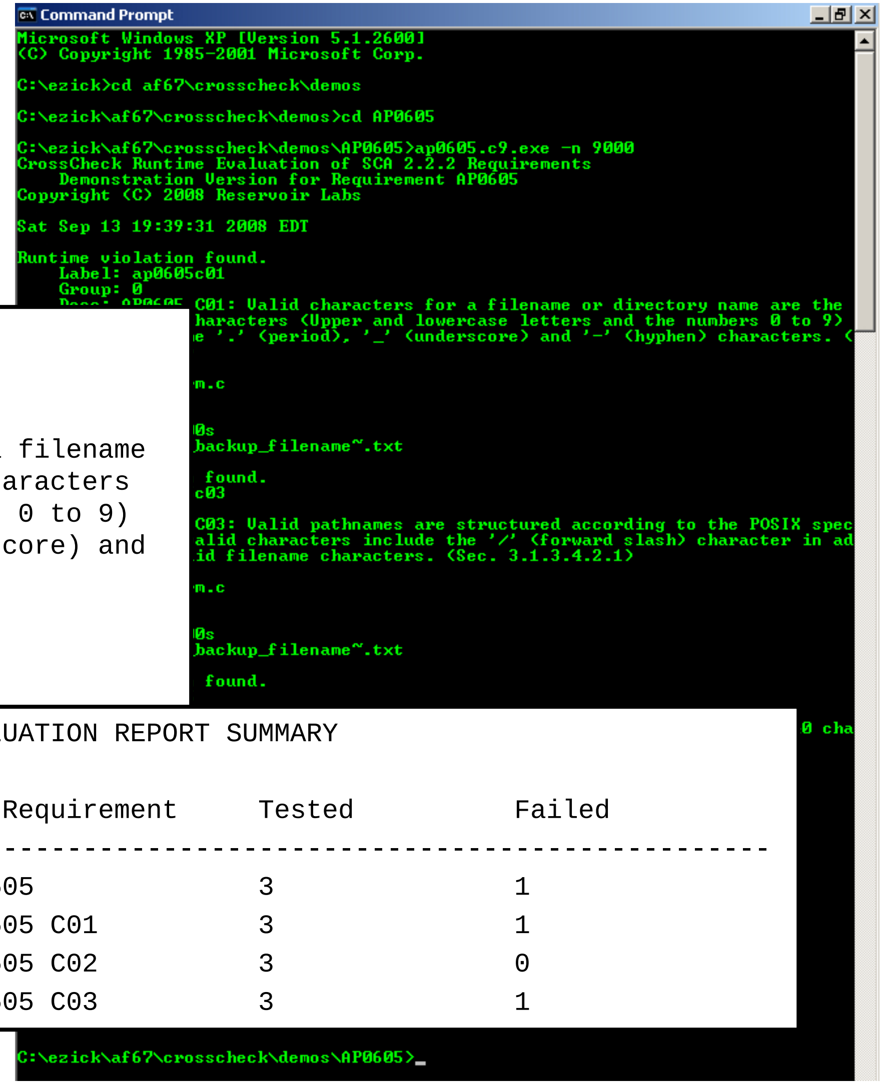
Software Systems Interface Checking

- **Software modules have APIs that must be used properly**
  - **Specific order of procedure calls, parameter constraints, etc.**
- **Existing design-by-contract tools focus on Hoare-style constraints**
  - **E.g. Eiffel/Larch, Java Modeling Language**
  - **Focus on preconditions & postconditions**
  - **Difficult to describe patterns and constraints that span multiple calls**
- **CrossCheck supports more global view of API state**
  - **Patterns of calls, sequencing, iterations, etc.**
- **Examples:**
  - **Malloc/Free usage**
  - **Race condition detection (e.g. Farzan, CAV '08)**
  - **VSIPL API usage**
  - **Software Communications Architecture (SCA)**



Software Systems Interface Checking

- Implemented demonstration specifications to check SCA (Software Communications Arch.) specification
  - E.g. AP0605 requirements



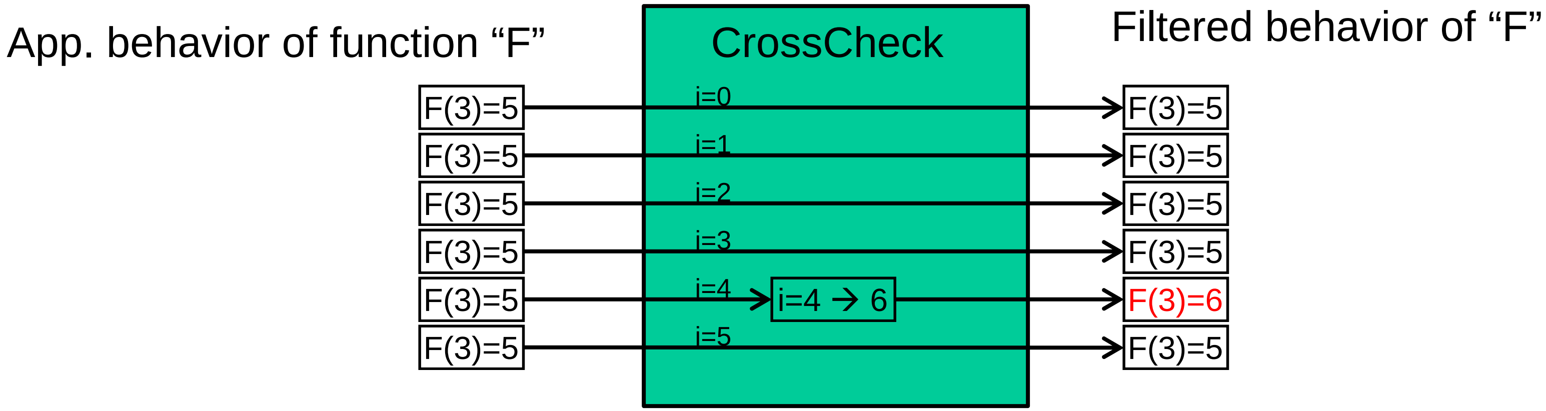

```
Runtime violation found.
    Label: ap0605c01
    Group: 0
    Desc: AP0605 C01: Valid characters for a filename
or directory name are the 62 alphanumeric characters
(Upper and lowercase letters and the numbers 0 to 9)
in addition to the '.' (period), '_' (underscore) and
'-' (hyphen) characters. (Sec. 3.1.3.4.2.1)

    File: waveform.c
    Line: 22
    EID: 1
    Elapsed: 0.000s
    PATHNAME: my_backup_filename~.txt
```

EVALUATION REPORT SUMMARY

SCA Requirement	Tested	Failed
AP0605	3	1
AP0605 C01	3	1
AP0605 C02	3	0
AP0605 C03	3	1

Test System Perturbation Injection

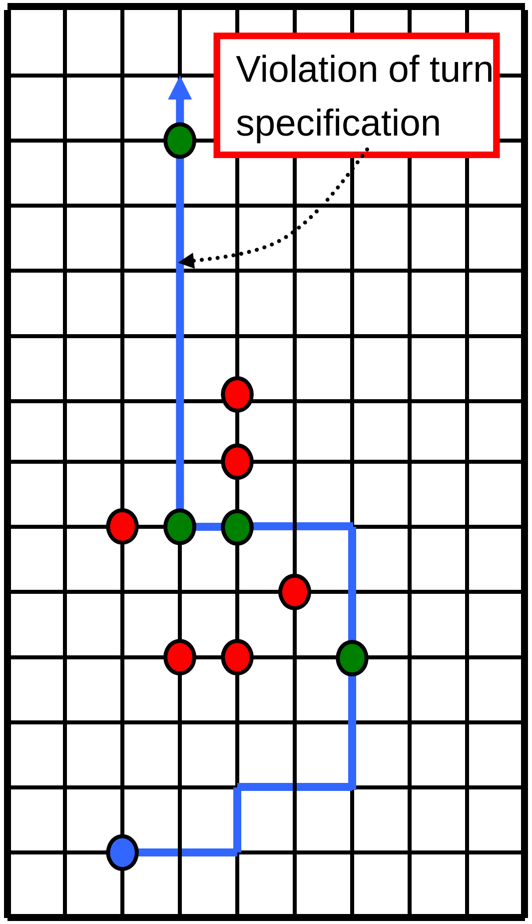
- **In testing, may want to force creation of a rare situation**
  - **E.g., "after 100 calls to procedures A and B, variable X changes"**
- **Can express such perturbation as CrossCheck specification**
  1. **Specification recognizes when necessary conditions are met for injecting the change**
  2. **Recovery action performs the desired change in the system under test**



Checking of Cognitive Systems

- **Cognitive applications are especially difficult to analyze statically**
- **Cognitive applications may rely more on emergent behavior (e.g. subsymbolic systems), for which there is not a strong intuitive notion of correctness**

- **Example: planning application on top of the *Soar* cognitive framework**
  - **Cognitive application is primarily a set of rules matching "facts" to corresponding fact updates**
  - **Facts match well to the Event abstraction**

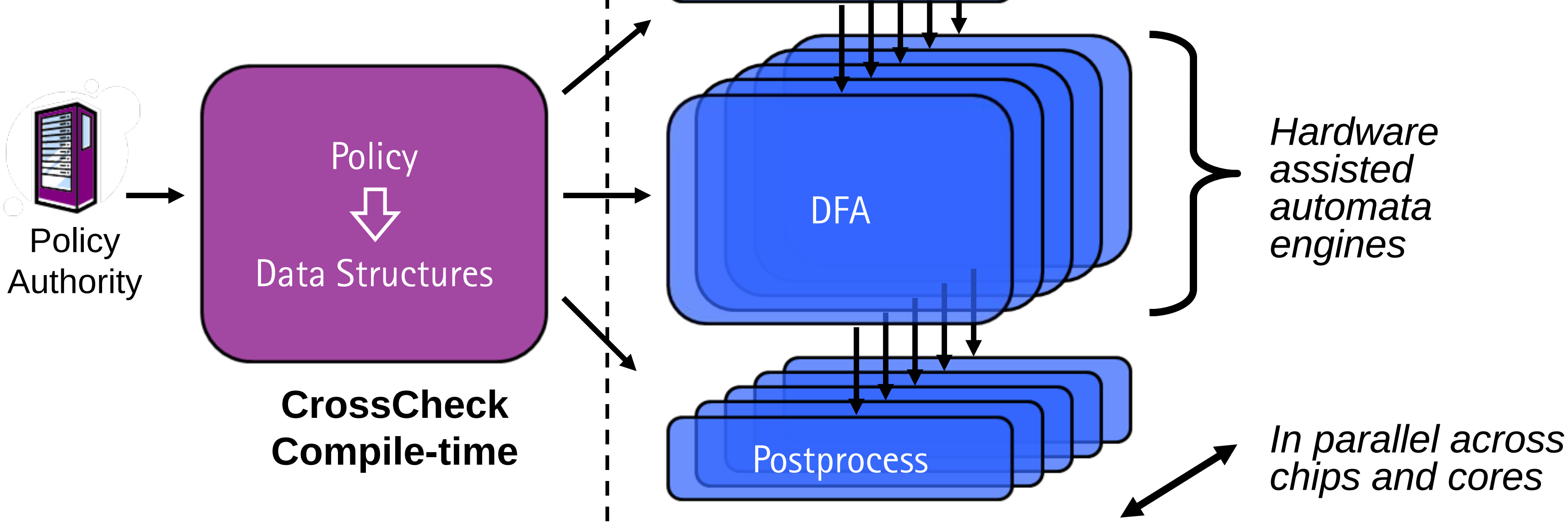


Deep Network Protocol Inspection

- **Network intrusion detection system (NIDS) watches for malicious traffic in network flow passing by at line speed (100Mbps, 1Gbps, 10Gbps, …)**
  - **Traditional NIDSs inspect and verify at the TCP/IP level, but not much at application level protocol**
  - **E.g., existing Reservoir NIDS technology: R-Scope**
- **Protocol verification requires deep content inspection and more sophisticated validity rules**
  - **Rise of protocol specification languages: Bro’s binpac, Microsoft GAPAL**
- **Good match for CrossCheck**
  - **Dynamic, complex rules, event abstractions ↔ protocol abstractions**
  - **Stresses high-speed operation**

CrossCheck as IDS

Layer CrossCheck specification-checking on existing R-Scope hardware-assisted checking infrastructure



CrossCheck Technology

Two distinguishing features of CrossCheck:

1. **Practical specification language**
   - Simple set of primitive operators as basis for specification language
   - Specification language has well-defined semantic basis
   - Close integration with general-purpose C for flexibility (and familiarity)
2. **Efficient execution engine for checking specifications**
   - Avoid explicit state machine graphs to avoid exponential size issues
   - Check state expanded only as needed, as match progresses

CrossCheck Specification Language (CSL) Formalism

- **Four basic operations:**
  1. **Comparison (basic unit of matching)**
  2. **Merge (e.g. “and,” “or”)**
  3. **Concatenation (e.g. sequencing)**
  4. **Repetition (w/ intervals)**
- **CSL defined in terms of an evaluation semantics**
  - **Rules have customizable semantics**
  - **Hierarchical expression language**
- **Each operation can execute arbitrary C code**
  - **Update global or match-local state**
  - **Use CrossCheck environment facilities**

The update operation transitions a single active match by processing an event (s)

$$
\begin{array}{l}
\text{concatenation} \\
\bar{\phi}_{+}.update_{E}(s) := \{ \\
\quad \texttt{let}\ \mathcal{A} = \emptyset; \\
\quad \texttt{let}\ \mathcal{A}_i = combine(\bar{\phi}_{|}.\mathcal{P}_i, s); \\
\quad \texttt{foreach}\ \Gamma_i \in \mathcal{A}_i\ \{ \\
\qquad \texttt{let}\ < \bar{\phi}_j, b_j >= spawn_{E}(\bar{\phi}_{+}.L_j, \Gamma_i); \\
\qquad \bar{\phi}_{+}.\mathcal{P}_j = \bar{\phi}_{+}.\mathcal{P}_j \cup \{\bar{\phi}_j\}; \\
\qquad \texttt{if}\ (b_j)\ \mathcal{A} = \mathcal{A} \cup \{\bar{\phi}_{+}.\phi_{+}.f(\Gamma_i, s)\}; \\
\quad \} \\
\quad \texttt{let}\ \mathcal{A}_j = combine(\bar{\phi}_{|}.\mathcal{P}_j, s); \\
\quad \texttt{foreach}\ \Gamma_j \in \mathcal{A}_j\ \{ \\
\qquad \mathcal{A} = \mathcal{A} \cup \{\bar{\phi}_{+}.\phi_{+}.f(\Gamma_j, s)\}; \\
\quad \} \\
\quad \texttt{let}\ b = \bar{\phi}_{+}.\mathcal{P}_i \neq \emptyset \wedge \bar{\phi}_{+}.\mathcal{P}_i \neq \emptyset; \\
\quad \texttt{return}\ < \mathcal{A} \backslash \{\Gamma_0\}, b >; \\
\}
\end{array}
$$

CSL Semantics Fragment

CrossCheck Specification Language (CSL) Syntax

Event declarations

```
NAV(timestamp:uint64,
    x_acc:double, y_acc:double, z_acc:double,
    align_done:int32, aligning:int32)            ;;
```

%%

Productions / Rules

```
AccelHigh      <- <accel_high_p>?:<accel_high_f> ;;
NoDecel        <- <predicate_p_true>?:<no_decel_f>;;
ContinuedAccel <- <true_p>?:<long_accel_f> ;;
LongAccel      <- AccelHigh ; NoDecel* ; ContinuedAccel ;;

AccelRule := LongAccel,
              <rule_index_f_all>, <rule_init_f_empty>,
              <accel_high_rec_f>, <rule_destroy_f_nop>, 0,
              {oldest_only, no_rollback}, {},
              "Acceleration too high for too long" ;;
```

%%

C code

[user-written C code]

CrossCheck Specification Language (CSL) Syntax

```
NAV(timestamp:uint64,
    x_acc:double, y_acc:double, z_acc:double,
    align_done:int32, aligning:int32)            ;;
```

Event declarations

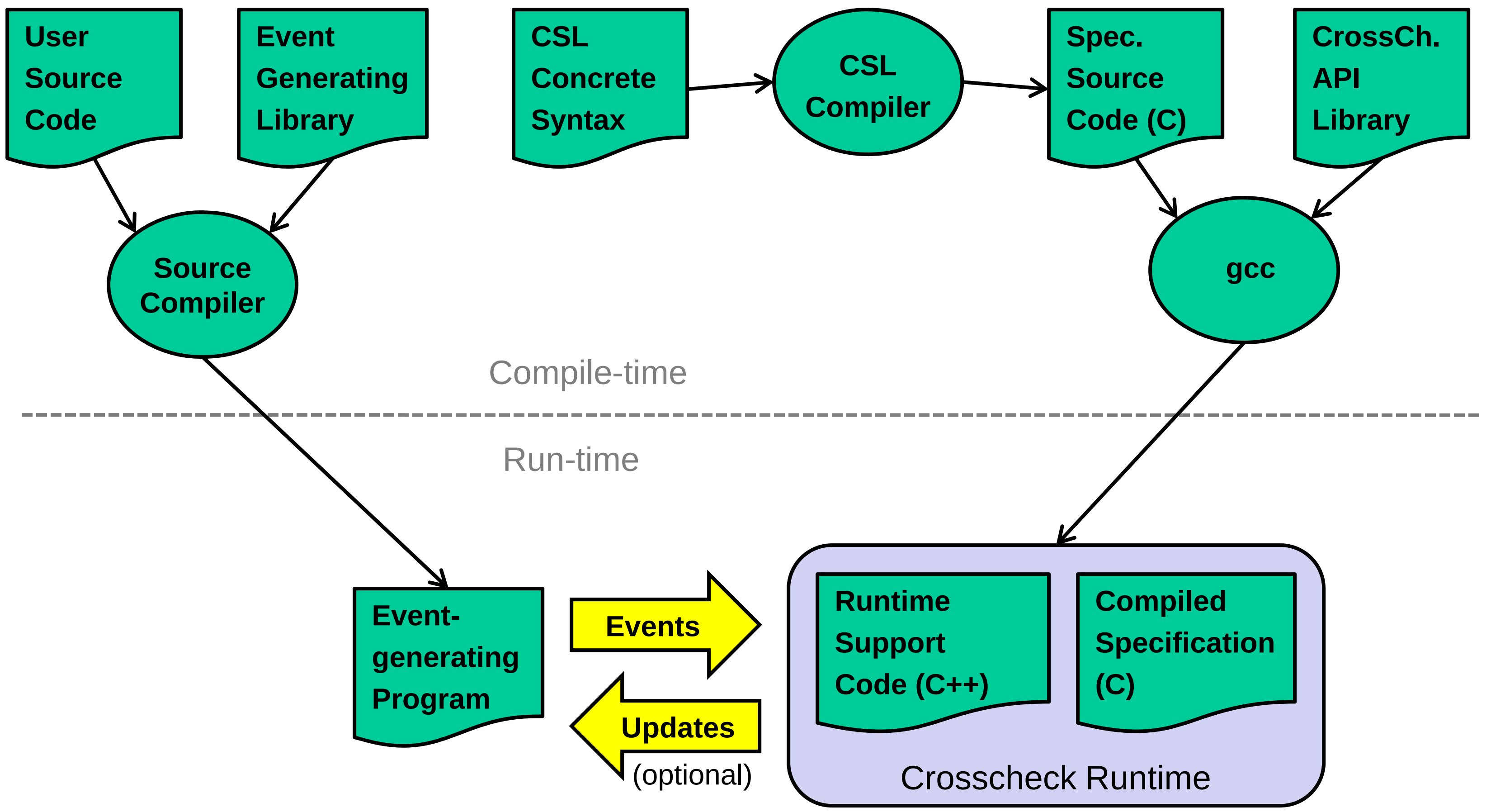
```
%%

AccelHigh      <- <accel_high_p>?:<accel_high_f> ;;
NoDecel        <- <predicate_p_true>?:<no_decel_f>;;
ContinuedAccel <- <true_p>?:<long_accel_f> ;;
LongAccel      <- AccelHigh ; NoDecel* ; ContinuedAccel ;;

AccelRule := LongAccel,
              <rule_index_f_all>, <rule_init_f_empty>,
              <accel_high_rec_f>, <rule_destroy_f_nop>, 0,
              {oldest_only, no_rollback}, {},
              "Acceleration too high for too long" ;;

%%
```

Productions / Rules

[user-written C code]

C code

CrossCheck Implementation Workflow



Future Directions

- **Add usability features**
  - Continue to add to specification library
  - Second-order analysis to jump-start specification writing
  - Compatibility with standardized data exchange formats for Event streams
    - E.g. Data Distribution Service (DDS)
- **Performance features**
  - Integration with hardware support from R-Scope
  - Needed for some use cases (NIDS), but not others
- **Continue to guide CrossCheck progress with use cases**

Second-order Analysis

Feedback can be established between rule matches and specifications, or between matches and the external system

- **Use for specification inference**
  - **System can suggest possible specifications based on training data**
- **Use for model-based recovery**
  - **Recovery operations operate according to a formal model of the system under test**
- **Probabilistic failure detection**
  - **Collections of nonfatal violations, accumulating a probability of error**

Summary

- **Dynamic checking of specifications is broadly applicable**
  - **Works in many cases where static checking not feasible**
- **Useful to abstract dynamic checking support into a framework (CrossCheck)**
- **Simple, orthogonal set of specification language primitives helps simplify specifications**
- **Specification language practicality important**
  - **C integration**