Protecting Critical Infrastructures – Power Grid Case Study

Karthik Pattabiraman

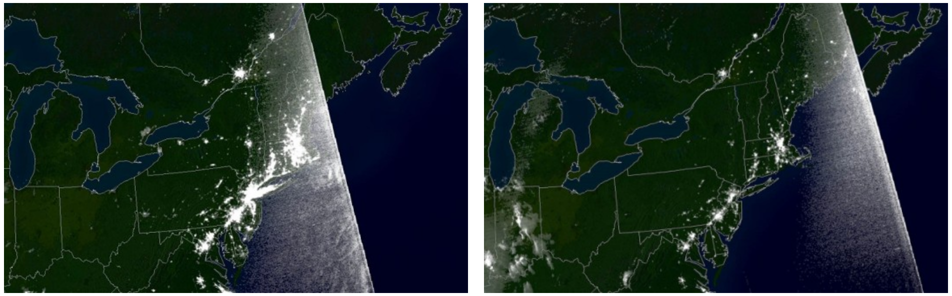
Microsoft Research

Joint work with **Flore Yuan, Peter Klemperer**, **Zbigniew Kalbarczyk and Ravishankar Iyer**

1867

Motivation: Power Grid

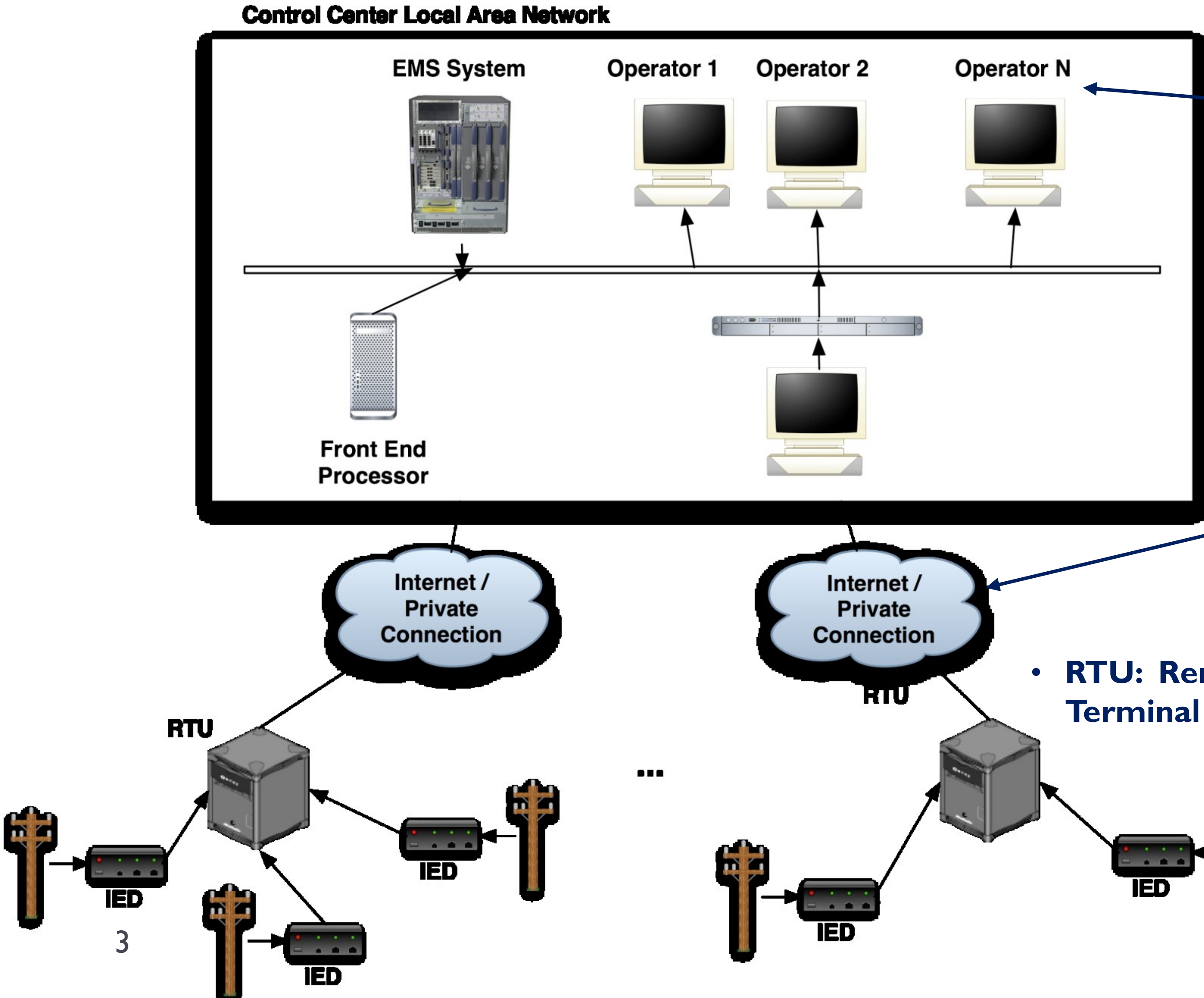
- **Large and complex infrastructure**
  - Multiple producers and consumers with varying needs/demands
- **Critical for national security**
  - Many other essential services depend on power
- **Local failures can cascade leading to massive blackouts**
  - Example: Northeastern blackout of August 2003



North-eastern blackout viewed from space

NYC skyline during blackout

Unprotected Power Grid



- Current Control Systems' OS:
- Windows NT
- Unix-like

- Internet/Intranet connection (may or may not be secure)

- RTU: Remote Terminal Unit

- IEDs:
- Intelligent Electronic Devices

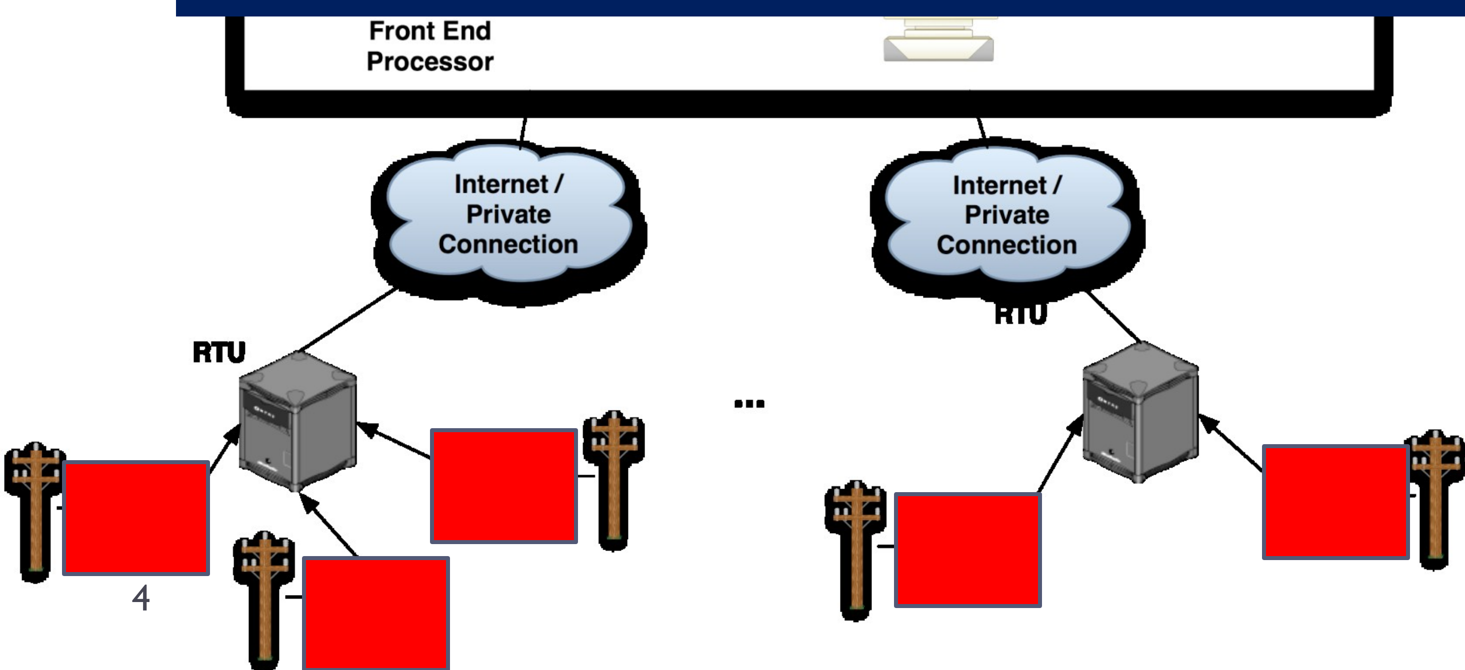
Fault-Models (IED)

Deployed in harsh or even adversarial conditions

- High temperatures, moisture or mechanical stress lead to failures

- May be subject to malicious tampering or physical attacks

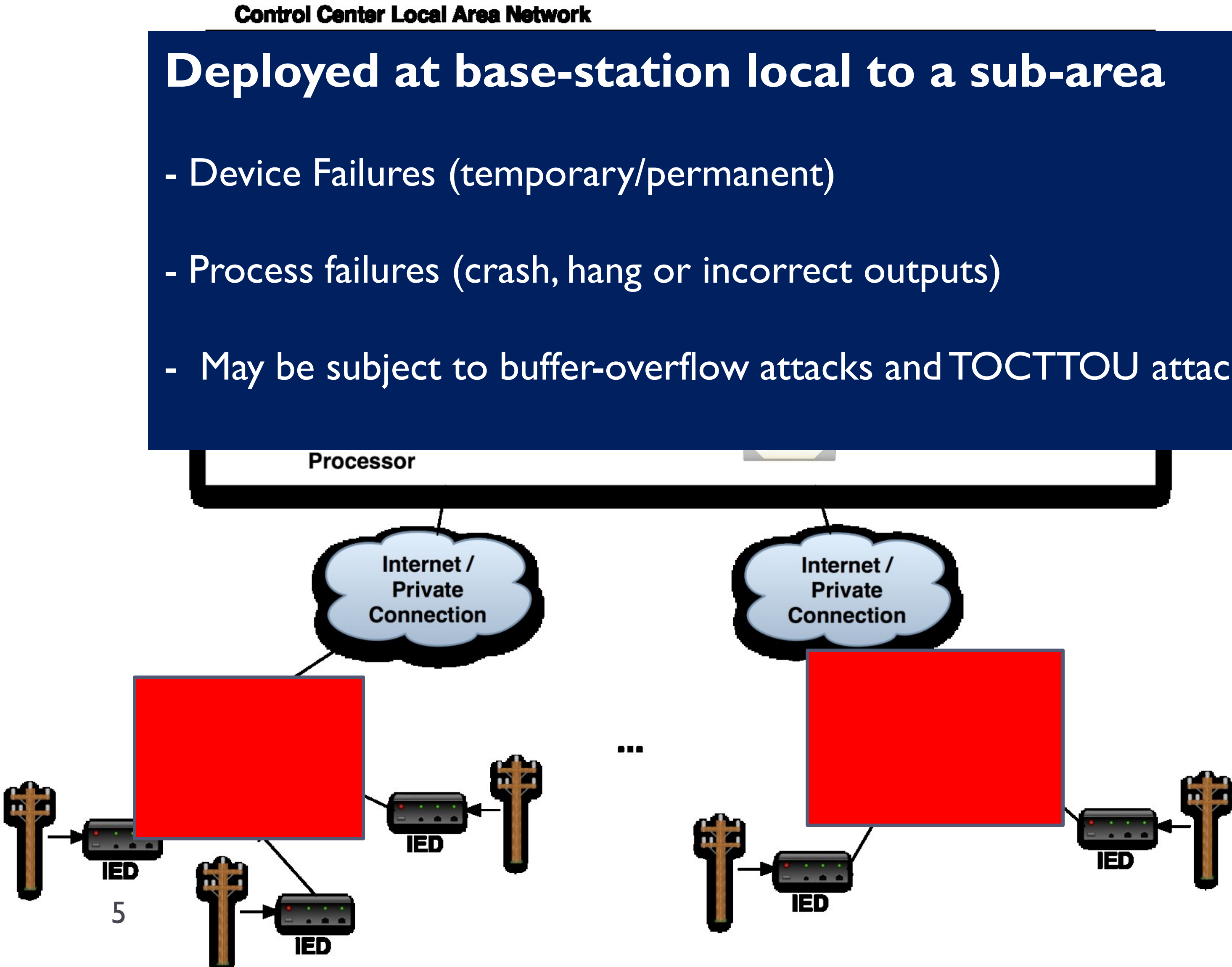
- Fake data injection, data delay attacks



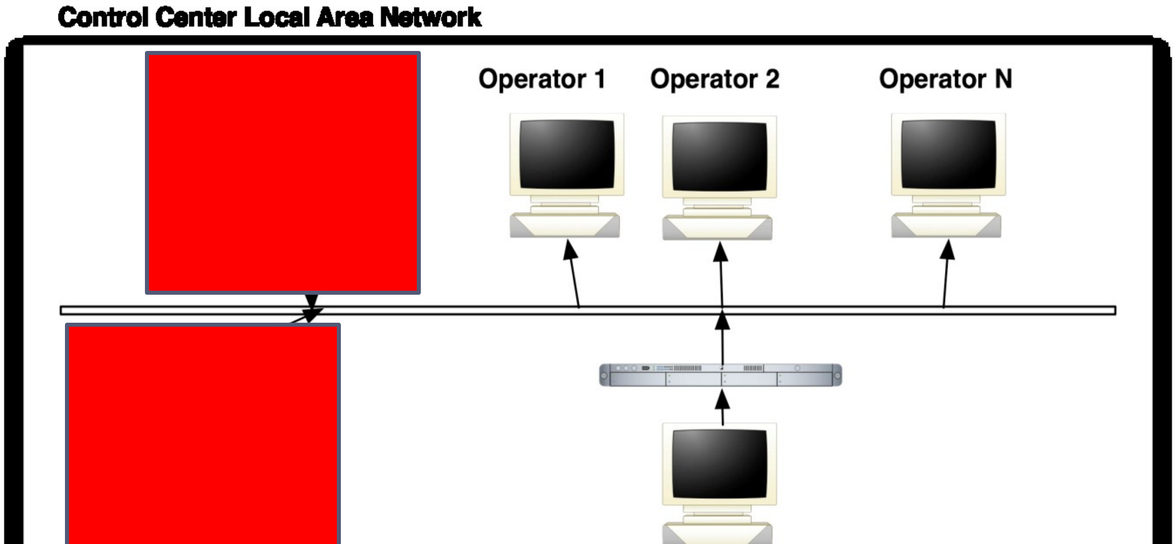
Fault-Models (RTU)

Deployed at base-station local to a sub-area

- Device Failures (temporary/permanent)
- Process failures (crash, hang or incorrect outputs)
- May be subject to buffer-overflow attacks and TOCTTOU attacks



Fault-Models (Front-end Processor/EMS)



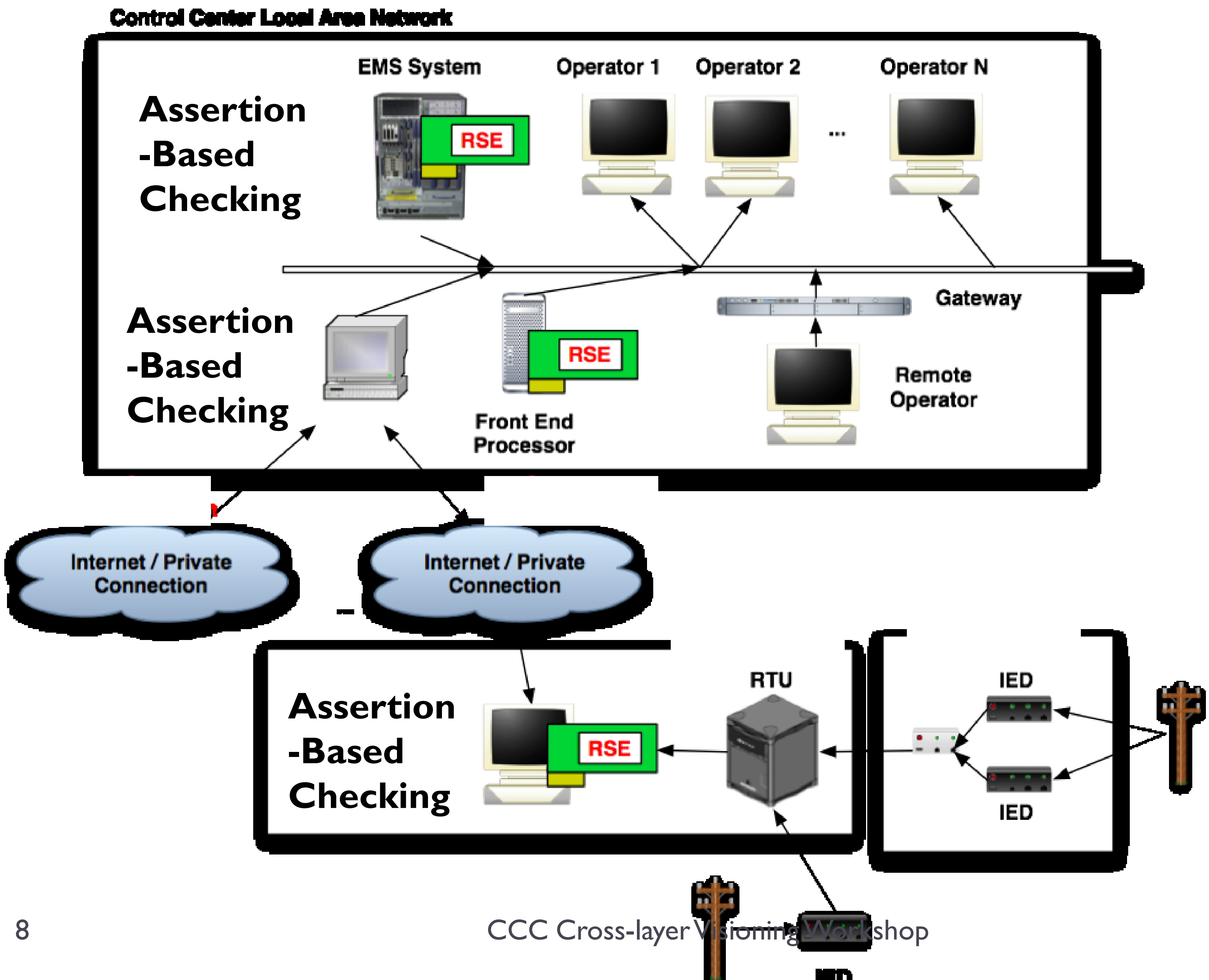
Deployed at control center for an area (multiple sub-areas)

- Device Failures (temporary/permanent)
- Process failures (crash, hang or incorrect outputs)
- Attacks from malicious hosts (e.g., DoS, buffer overflows, data-replay attack)
- Unauthorized access by malicious insiders or external attackers

Constraints for Protection Techniques

- **Low performance overheads**
  - Real-time data processing and decision making
- **The "curse" of legacy**
  - Large installed s/w base often on antiquated h/w
- **Low false-alarm rates**
  - Do not want to trigger recovery actions unnecessarily
- **Prevention of error propagation**
  - Preemption of cascading failures

Protected Power Grid

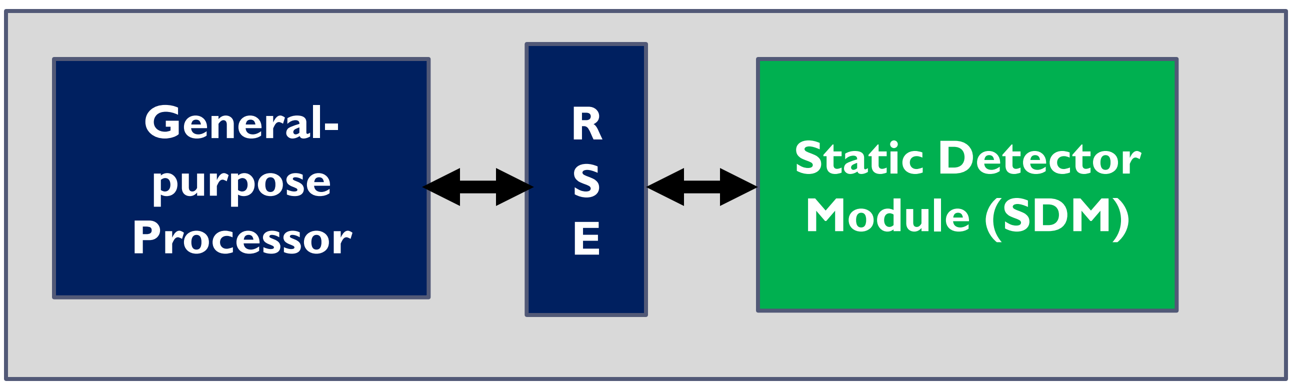


Talk Outline

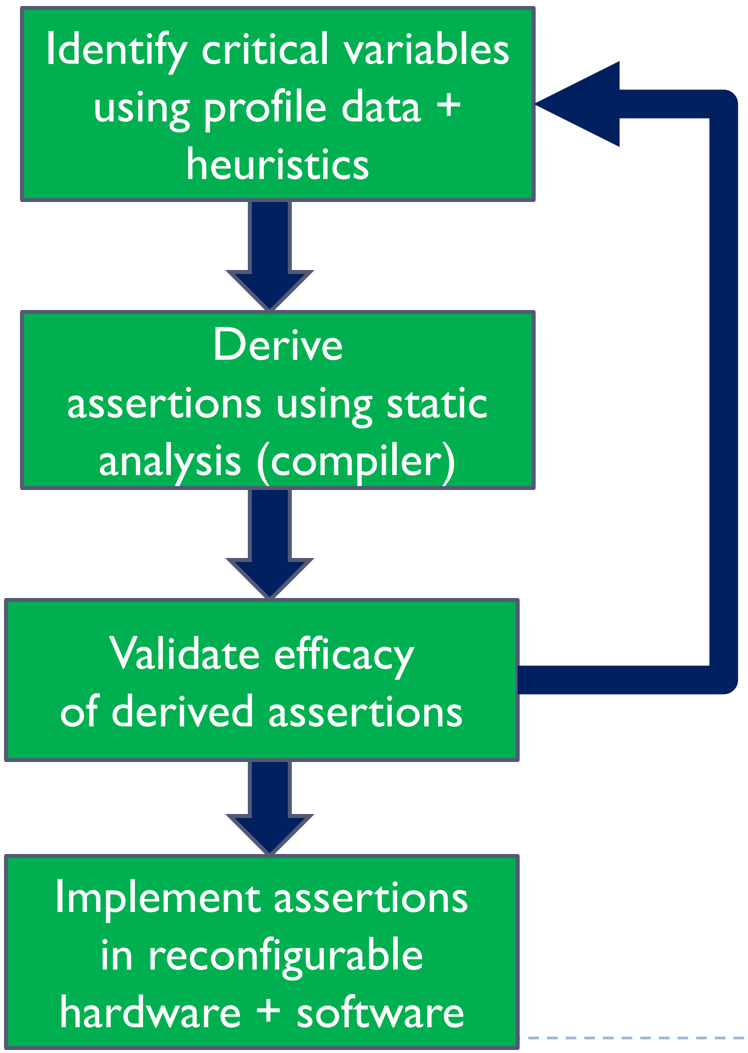
- **Background and Motivation**
- **Assertion-based Checking (ABC)**
  - Derivation of assertions (CVR Technique) [TDSC'09][IOLTS'07]
  - Validation of assertions (SymPLFIED) [DSN'08 – best paper]
- **Case Study: Application of ABC to power grid**
- **Conclusion and Open Questions**

Assertion-based Checking: Overview

- **Assertions/runtime checks specific to program**
  - Focus on protecting program's critical variables
  - Based on static analysis of program source code
- **Execute checks on special-purpose reconfigurable hardware (RSE) in parallel with application**
  - Concurrent, low-latency detection of errors
  - Generic interface to processor's internal state



Assertion-based Checking: Approach



- **Critical Variable Identification**
  - Errors in variables likely to result in failures
- **Assertion Derivation**
  - Based on dependencies of critical variables
- **Assertion Validation**
  - Formal methods to find corner cases that escape
- **Assertion Execution**
  - Execute assertions using reconfigurable h/w or s/w

Assertion-based Checking: Advantages

- **Only detect the errors that matter to application**
  - Many errors do not matter and detecting them violates safety
  - Overheads can be tuned based on application's requirements and the constraints of the hardware platform

- **Fully automated (no programmer intervention)**
  - Important for legacy code and for code evolution

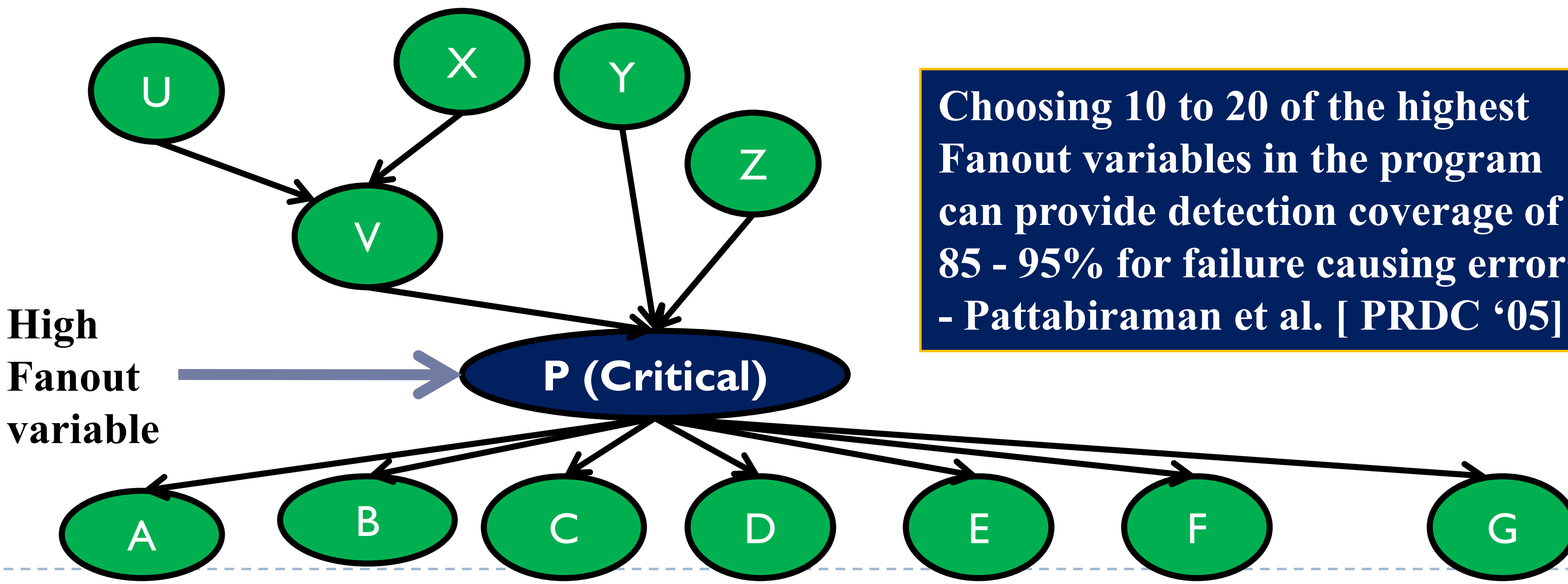
- **Prevent error-propagation (pre-emptive detection)**
  - Low detection latency due to hardware support
  - Formal guarantees on error-containment and detection

Talk Outline

- **Background and Motivation**
- **Assertion-based Checking (ABC)**
  - Derivation of assertions (CVR Technique) [TDSC'09][IOLTS'07]
  - Validation of assertions (SymPLFIED) [DSN'08 – best paper]
- **Case Study: Application of ABC to power grid**
- **Conclusion and Open Questions**

Derivation: Crit. Var. Identification

- **Critical Variable: Highly sensitive to program errors that cause failure**
  - Variables that have a high dynamic use count (**Fanout**)
  - Validated empirically using fault-injection experiments



Choosing 10 to 20 of the highest Fanout variables in the program can provide detection coverage of 85 - 95% for failure causing errors - Pattabiraman et al. [PRDC '05]

Program Fails !

Derivation: Algorithm

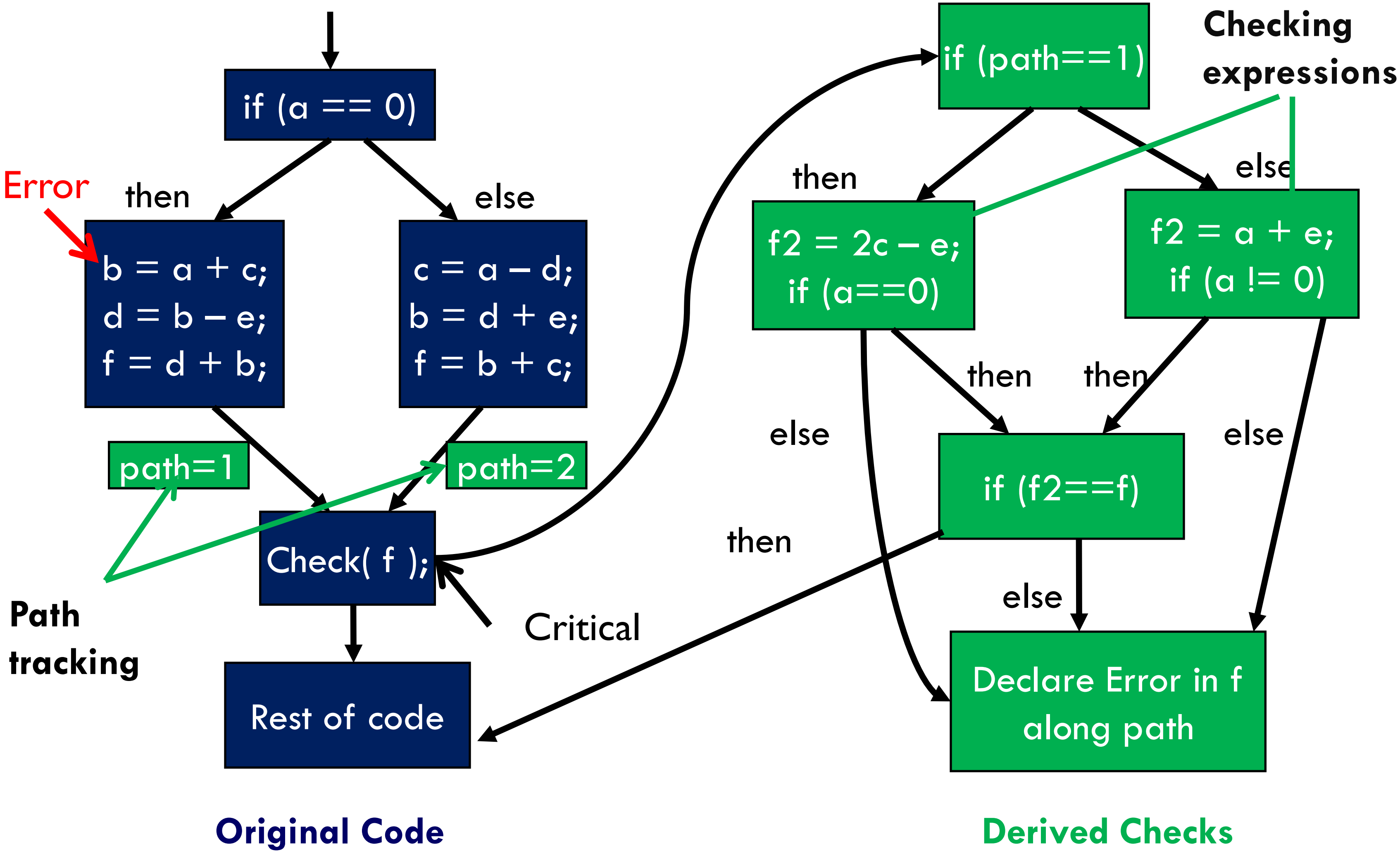
Extract backward slice of critical variable

Specialize slice specific to control paths

Optimize slice to obtain checking expression

Instrument program to track runtime paths, execute corresponding checks

Derivation: Example



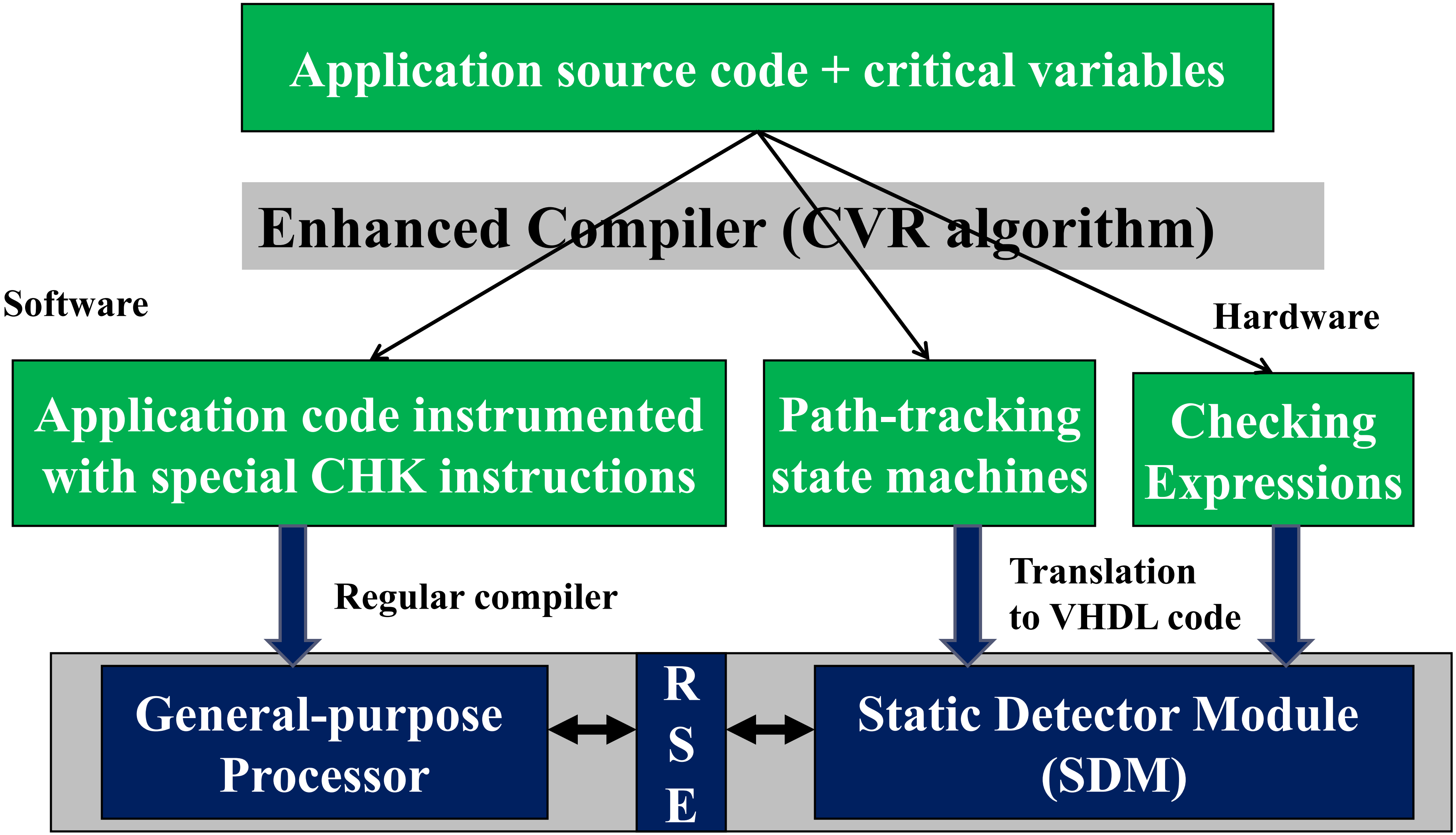
Derivation: Experimental Results

- **Added new sequence of compiler passes**
  - Implemented in LLVM optimizing compiler

- **Performance Evaluation (Pentium 4)**
  - Benchmarks: Stanford programs, Olden suite
  - Average performance overhead = 33 %

- **Coverage Evaluation (Fault-injection)**
  - Detected 77 % of failure-causing errors across programs
  - 68 % of errors were detected before propagation
  - Less than 3 % of errors detected were benign

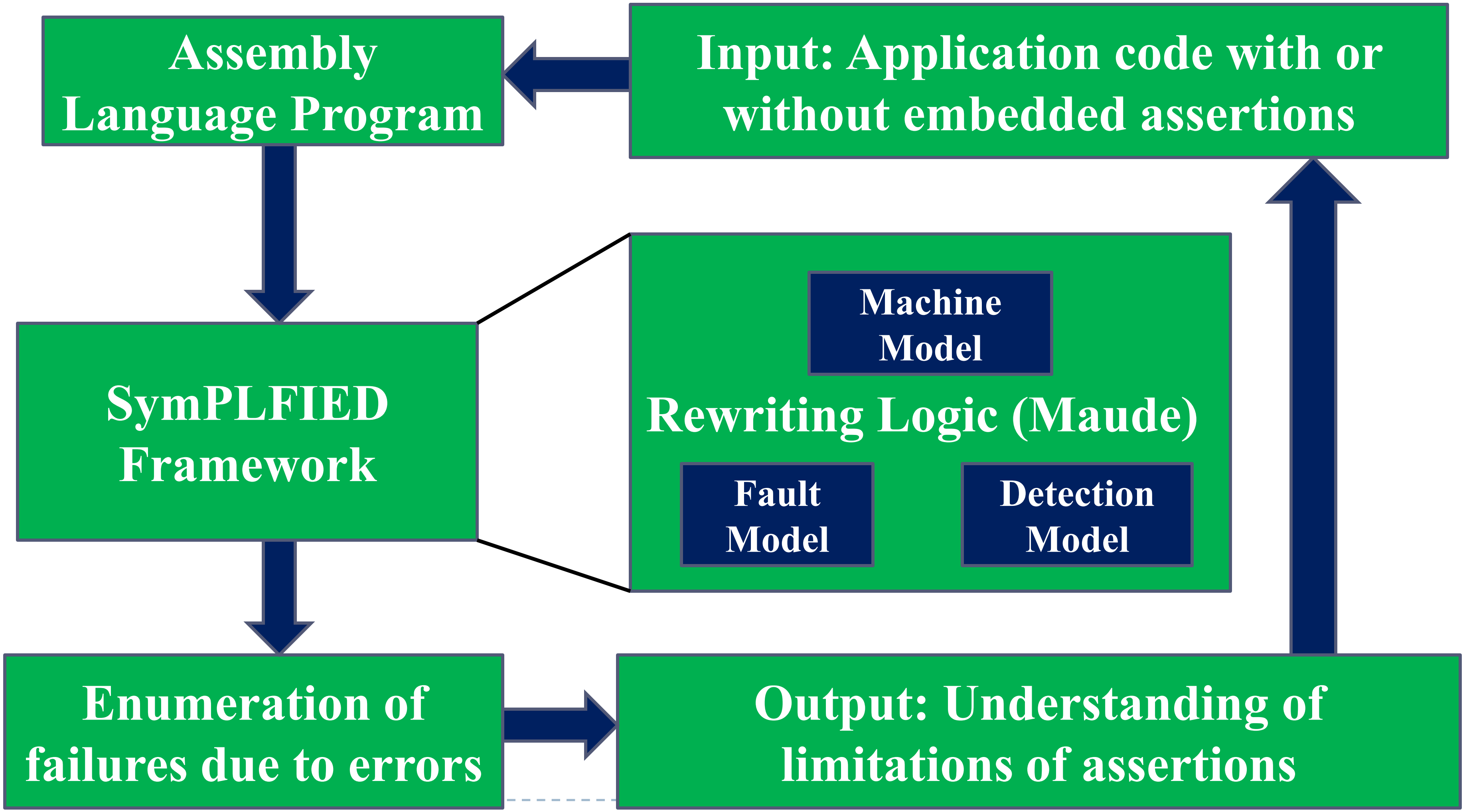
Detection: H/W Implementation



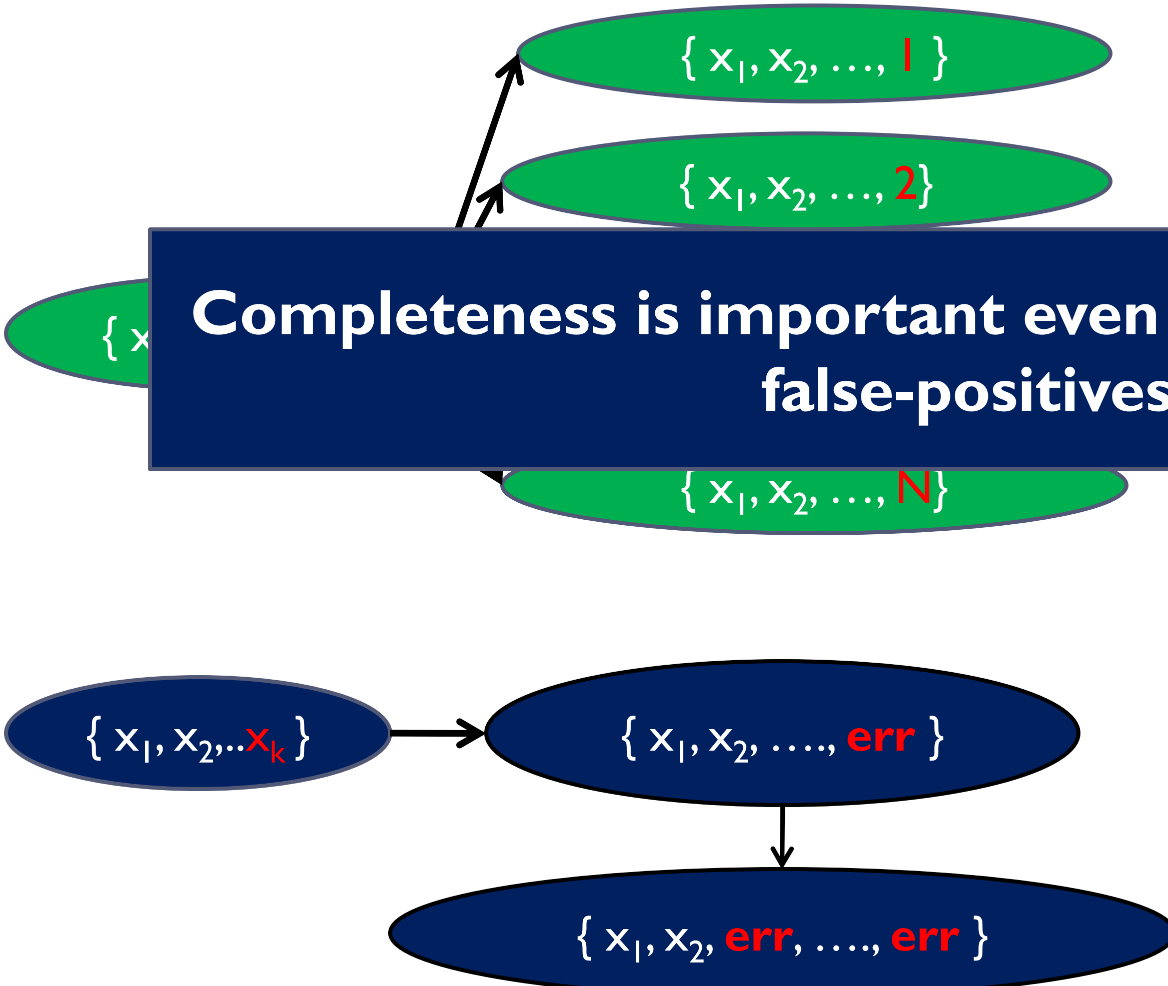
Talk Outline

- **Background and Motivation**
- **Assertion-based Checking (ABC)**
  - Derivation of assertions (CVR Technique) [TDSC'09][IOLTS'07]
  - Validation of assertions (SymPLFIED) [DSN'08 – best paper]
- **Case Study: Application of ABC to power grid**
- **Conclusion and Open Questions**

Validation: SymPLFIED Framework



Validation: Symbolic Execution



- **Exhaustive enumeration leads to state space explosion**
- **program as an abstract symbol**
  - Track propagation of errors symbolically
  - Abstraction may lead to false-positives

Completeness is important even at the cost of a few false-positives

Validation: Example

```
        ori  $2  $0  #1
        read $1
        mov $3, $1
        ori $4 $0 #1
loop:  setgt $5 $3 $4
        beq  $5 0 exit
        mult $2 $2 $3
        subi $3 $3 #1
        beq  $0 #0 loop
exit:   prints "Factorial = "
        print $2
        halt
```

($3 > $4) = true
Exit loop

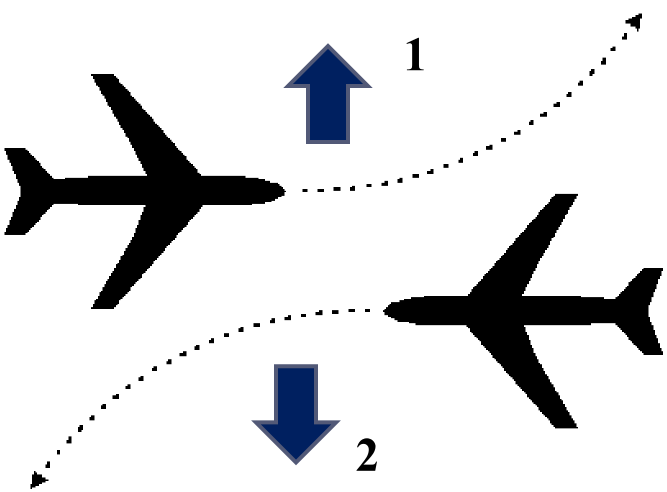
err

($3 > $4) = false
Reenter loop and continue

true => { N ! , (N ! / 2 !) , (N ! / 3 !) ….. 1}
false => err

Validation: Results Summary

- **Tcas: Application Characteristics**
  - FAA mandated Aircraft collision avoidance system
  - Rigorously verified protocol and implementation
  - About 150 lines of C code = 1000 lines of assembly
  - Ran SymPLFIED on a cluster of workstations (in parallel)



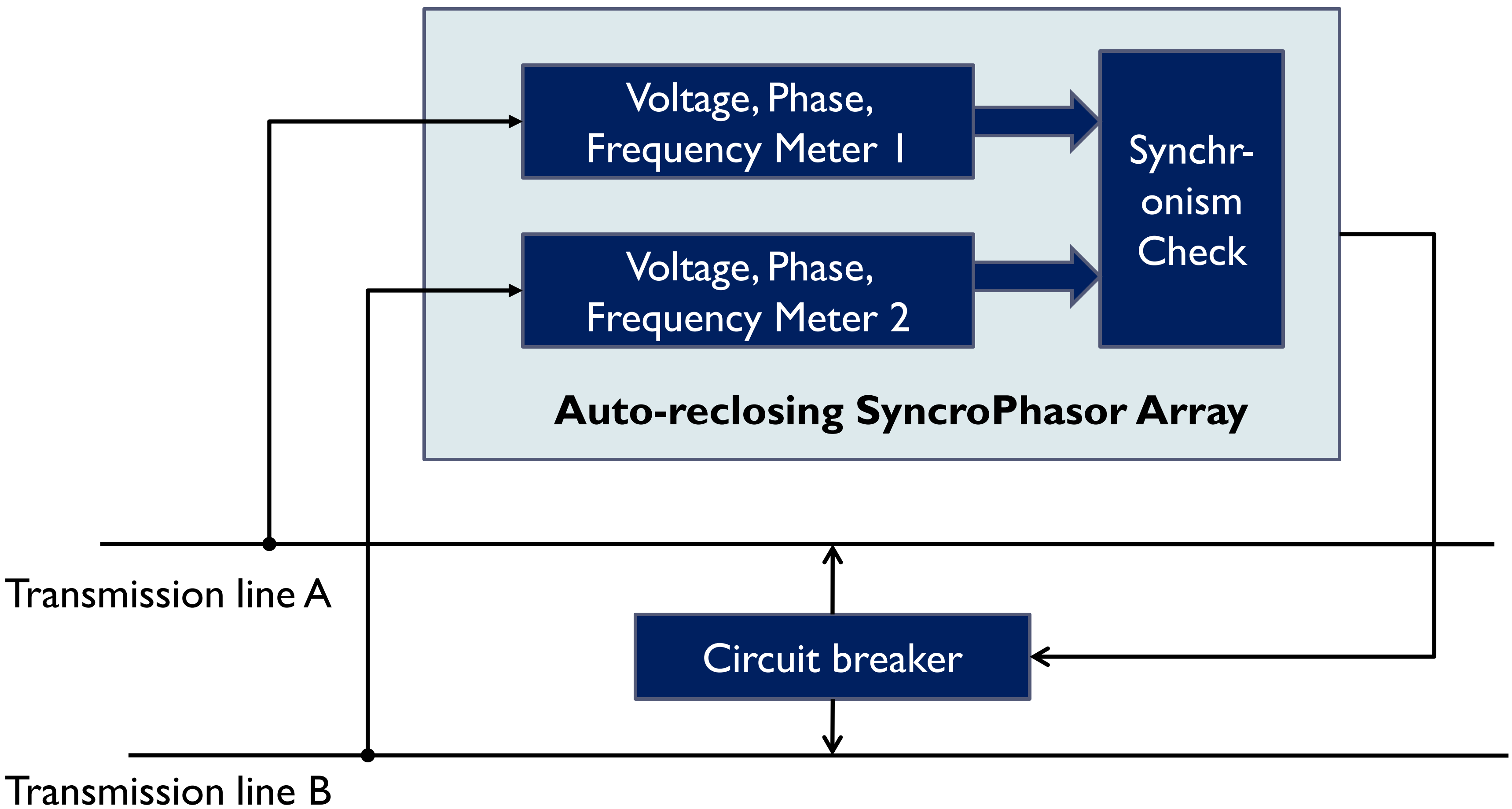
Found a fault causing a safety violation within 5 minutes

1. Injected into a register holding a function's return value
2. Did not find the fault with random fault-injection even when run for 5x the time

Talk Outline

- **Background and Motivation**
- **Assertion-based Checking (ABC)**
  - Derivation of assertions (CVR Technique) [TDSC'09][IOLTS'07]
  - Validation of assertions (SymPLFIED) [DSN'08 – best paper]
- **Case Study: Application of ABC to power grid**
- **Conclusion and Open Questions**

Case Study: SyncroPhasor



Case Study: Check Insertion

```
void checkSync(Meter meterA, Meter meterB, Breaker breaker)
{
    int PhaseA = meterA.Input;
    int PhaseB = meterB.Input;
    int Difference = abs( PhaseB – PhaseA );
     // Stuck-at-Fault: Difference = 180
    if (Difference < 25) {
            breaker.status = reclose;
            Check( abs(meterB.Input – meterA.Input) < 25 );
    } else {
            breaker.status = open;
            Check( abs(meterB.Input – meterA.Input) >= 25 );
    }
```

Safety violation !

Error detected by check failing

Case Study: Error Propagation Example

```
            read $1
            mov $2, $1
            read $1
            mov $3 $1
            sub $2 $2 $3
            setgt $5 $2 #0
            beq $5 #0 decision
            mult $2 $2 #-1
            ori $4 #24
decision :  setgt $5 $2 $4
            beq $5 0  open
            prints "reclose"
open :      prints "open"
            halt
```

err

(\$3-\$2 > 0) == false
Get absolute value

(\$3-\$2 > 0) == true
Goto decision

(\$2 > 24) == false

(\$2 > 0) == true
Goto open

false => reclose

true => open

Case Study: Check Validation

```
              read $1
              mov $2, $1
              mov $6 $2
              read $1
              mov $3 $1
              mov $7 $3
              sub $2 $2 $3
              setgt $5 $2 #0
              beq $5 #0 decision
              mult $2 $2 #-1
              ori $4 #25
decision :    setgt $5 $2 $4
              beq $5 0  open
              prints "reclose"
open :
              check(abs($7-$6)>=25)
              prints "open"
              halt
```

err

($2 > 25) == false

($2 > 25) == true
Goto open

false => detection

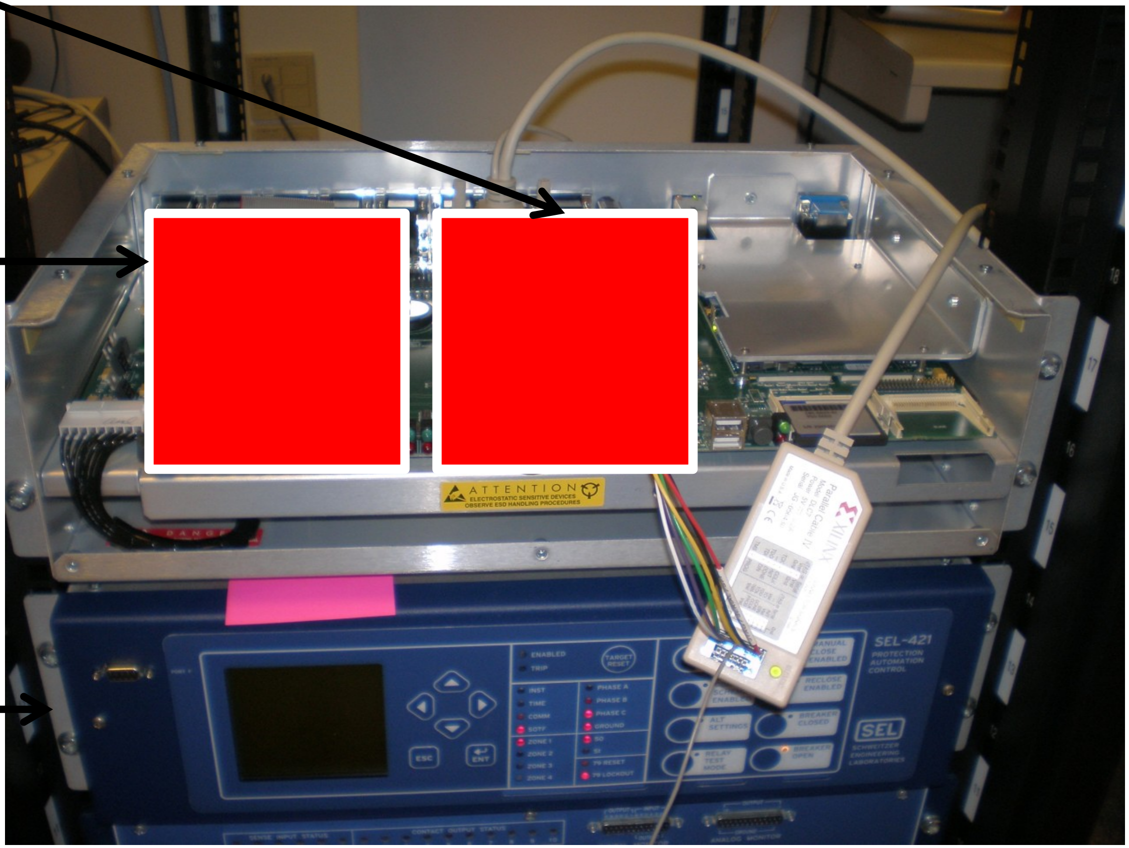
Case Study: Hardware Implementation

Performance Overhead = 2 %
Area Overhead = 2.5 %

Nallatech DIME-2 FPGA with Xilinx FPGA

Schweitzer SEL-3351 data aggregator

Schweitzer SEL-421 relay



Syncro-phasor setup

Talk Outline

- **Background and Motivation**
- **Assertion-based Checking (ABC)**
  - Derivation of assertions (CVR Technique) [TDSC'09][IOLTS'07]
  - Validation of assertions (SymPLFIED) [DSN'08 – best paper]
- **Case Study: Application of ABC to power grid**
- **Conclusion and Open Questions**

Conclusions

- **Power-grid: Example of complex and critical infrastructure with multiple constraints**
  - Range of devices from very small to large (**Customizable**)
  - Prevalence of legacy code (**Backward Compatible**)
  - Real-time processing requirements (**Low overheads**)
  - Containment and isolation of errors (**Formal guarantees**)

- **Example protection technique for power-grid: Assertion-based Checking (ABC)**
  - Automatically derive assertions based on static analysis (CVR)
  - Formally validate efficacy of checks (SymPLFIED)
  - Implement using reconfigurable hardware (RSE)

Open Questions

- **Do the lessons from the power-grid carry over to other critical infrastructures, e.g., water system ?**
  - Can we develop a common characterization of the systems ?
- **At what level should we apply protection techniques ?**
  - Hardware, Operating System, Middleware, Application
- **What kind of guarantees do we need to provide ?**
  - Formal, probabilistic, qualitative, hand-waving ?
- **How does reliability impact security in these systems ?**
  - Should we address both in a unified manner or separately ?
  - Are the two goals in conflict or can they leverage one another ?