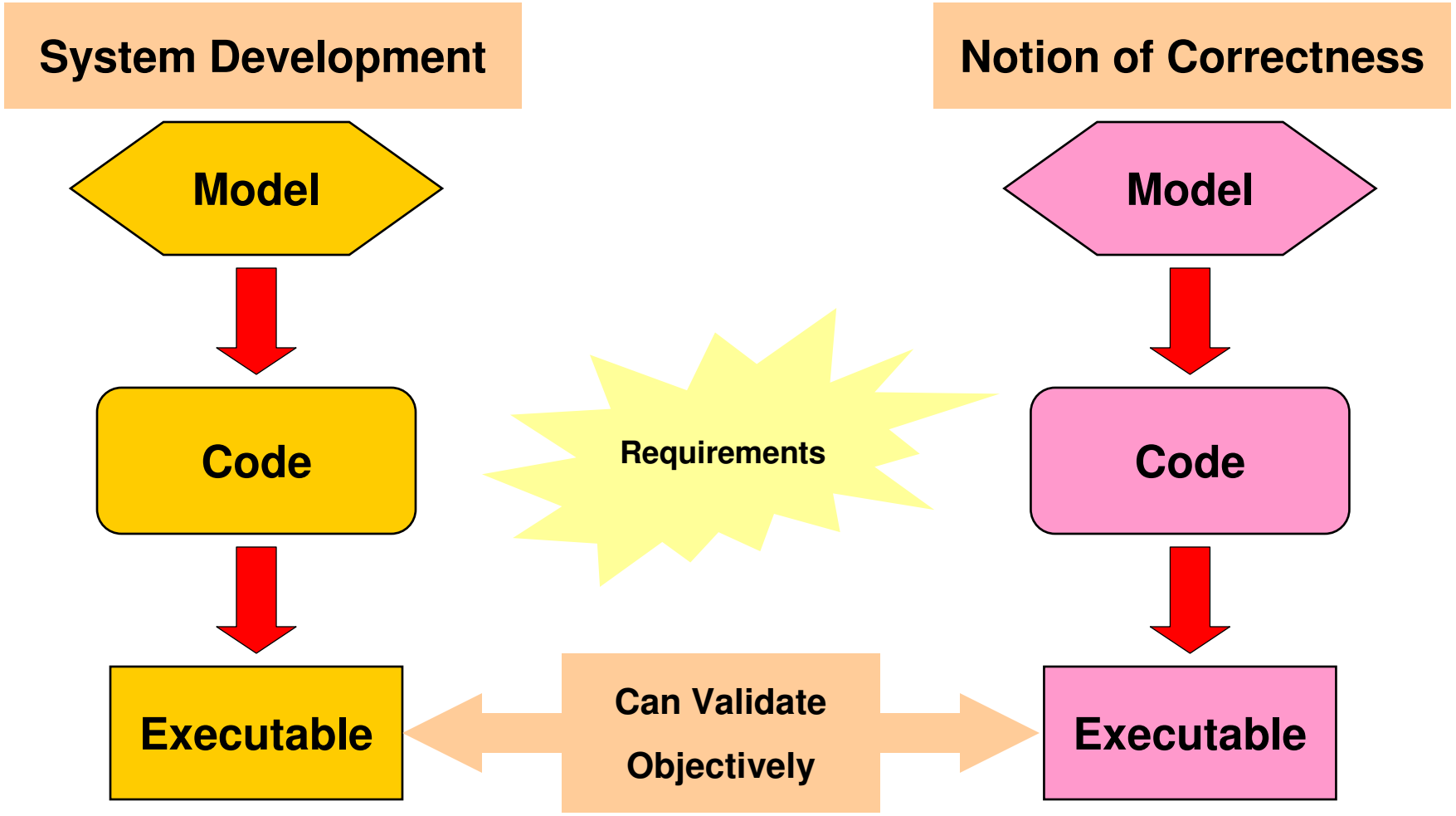
Model-Driven Construction of Certified Binaries

Sagar Chaki[1], James Ivers[1], Peter Lee[2], Kurt Wallnau[1], Noam Zeilberger[2]

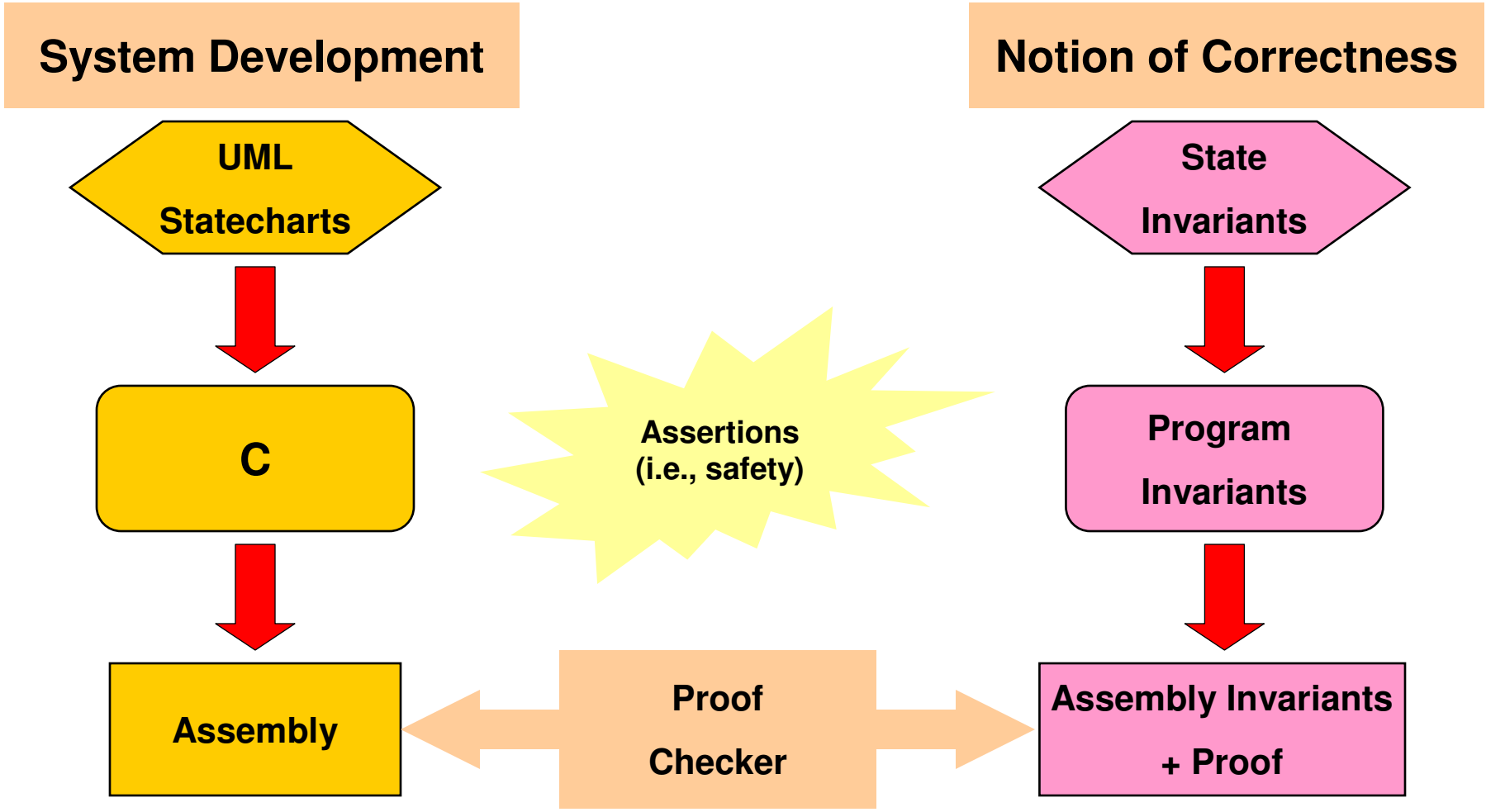
[1]Software Engineering Institute, CMU
[2]Computer Science Department, CMU

Software Engineering Institute | Carnegie Mellon

Overall Vision



Concrete Realization



Application Scenario

Producer

Consumer

Safety Requirements

Certified Binary

Certified Binary Generation

Certificate Validation

- Only release binaries
- Source considered IP

No Obfuscation

- Don't trust producer (tools)
- Don't trust communication channel

Foundations

Certifying Model Checking (CMC)

- Generate the invariants required for certificate construction

Proof-Carrying Code (PCC)

- Generate the certificate given the invariants

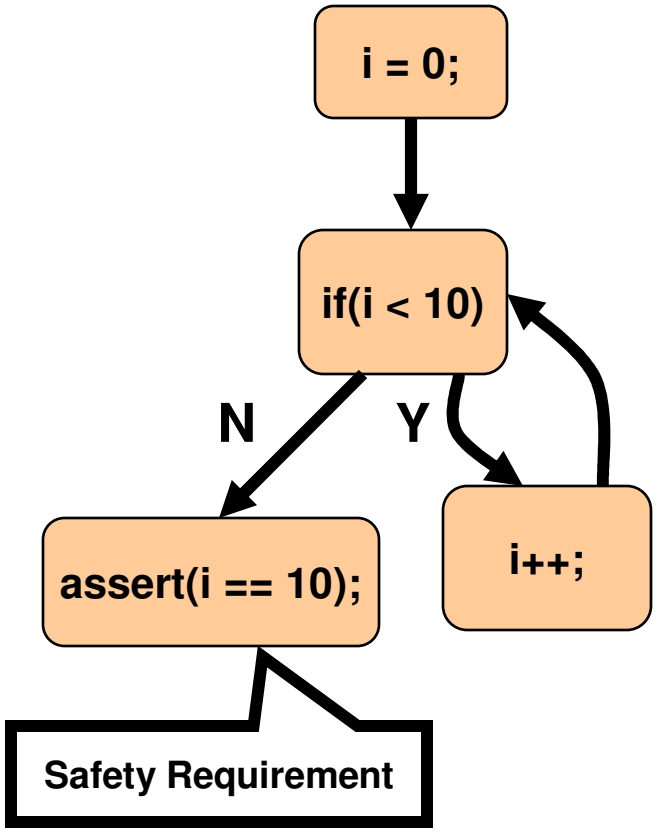
Extends both paradigms

- CMC extended to executables
- PCC extended to a richer class of specifications

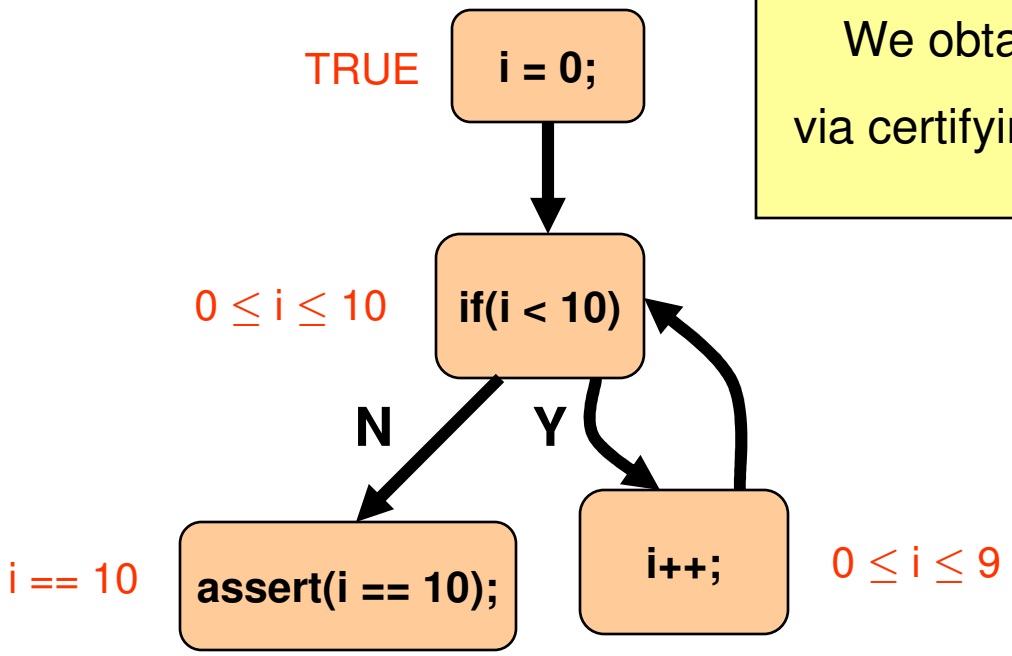
Only talk about safety and invariants here

- Need ranking functions for liveness – see paper

Model Example

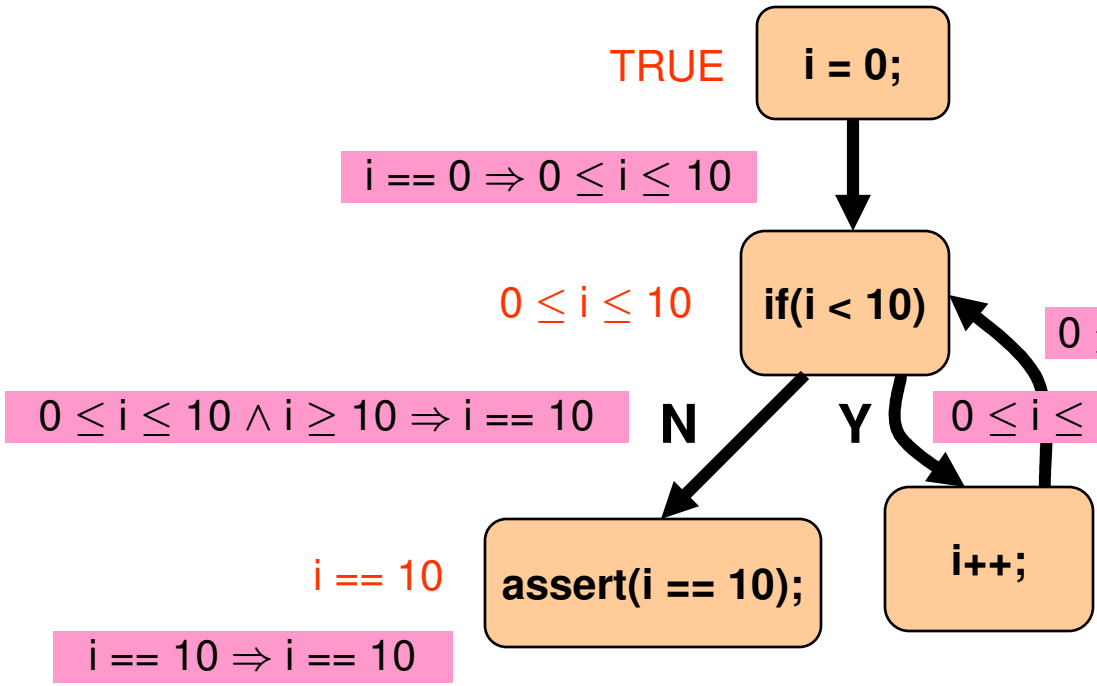


Model Invariants



We obtain the invariants via certifying model checking

Model Certification: Step 1: VC-Gen



VC: Preservation

Strongest Postcondition of

Invariant before Transition

⇒

Invariant after Transition

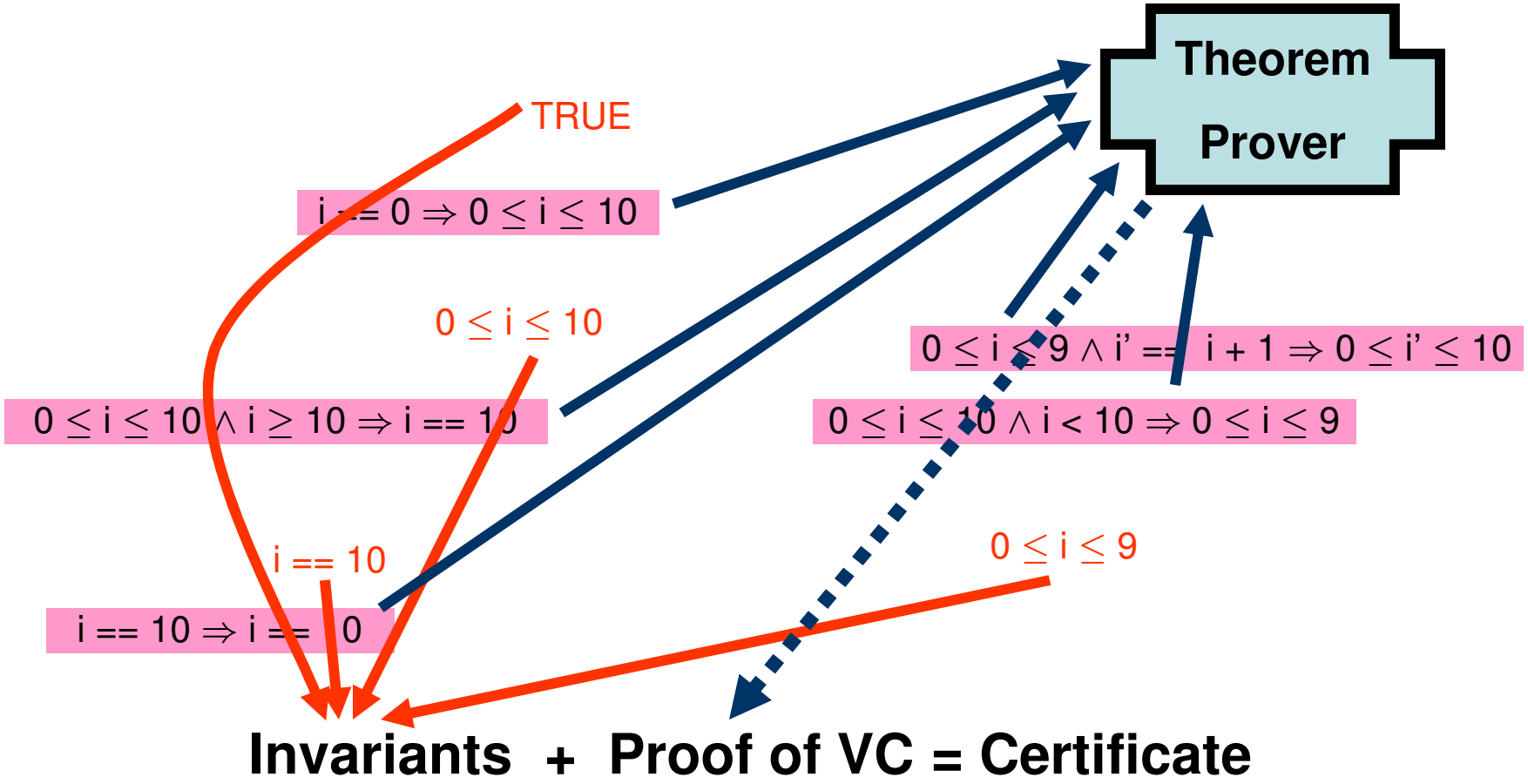
VC: Strength

Invariant at Assertion

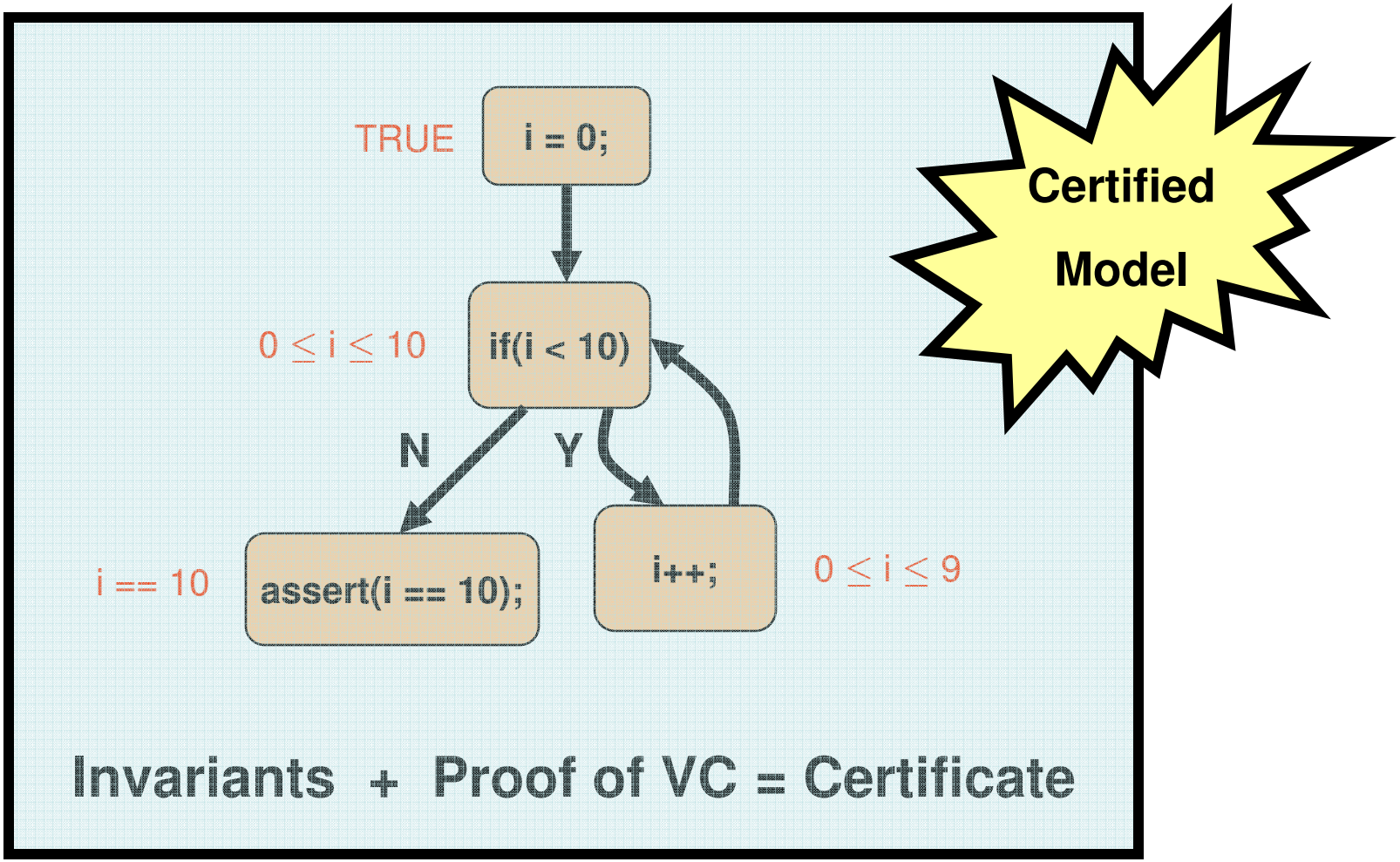
⇒

Assertion Condition

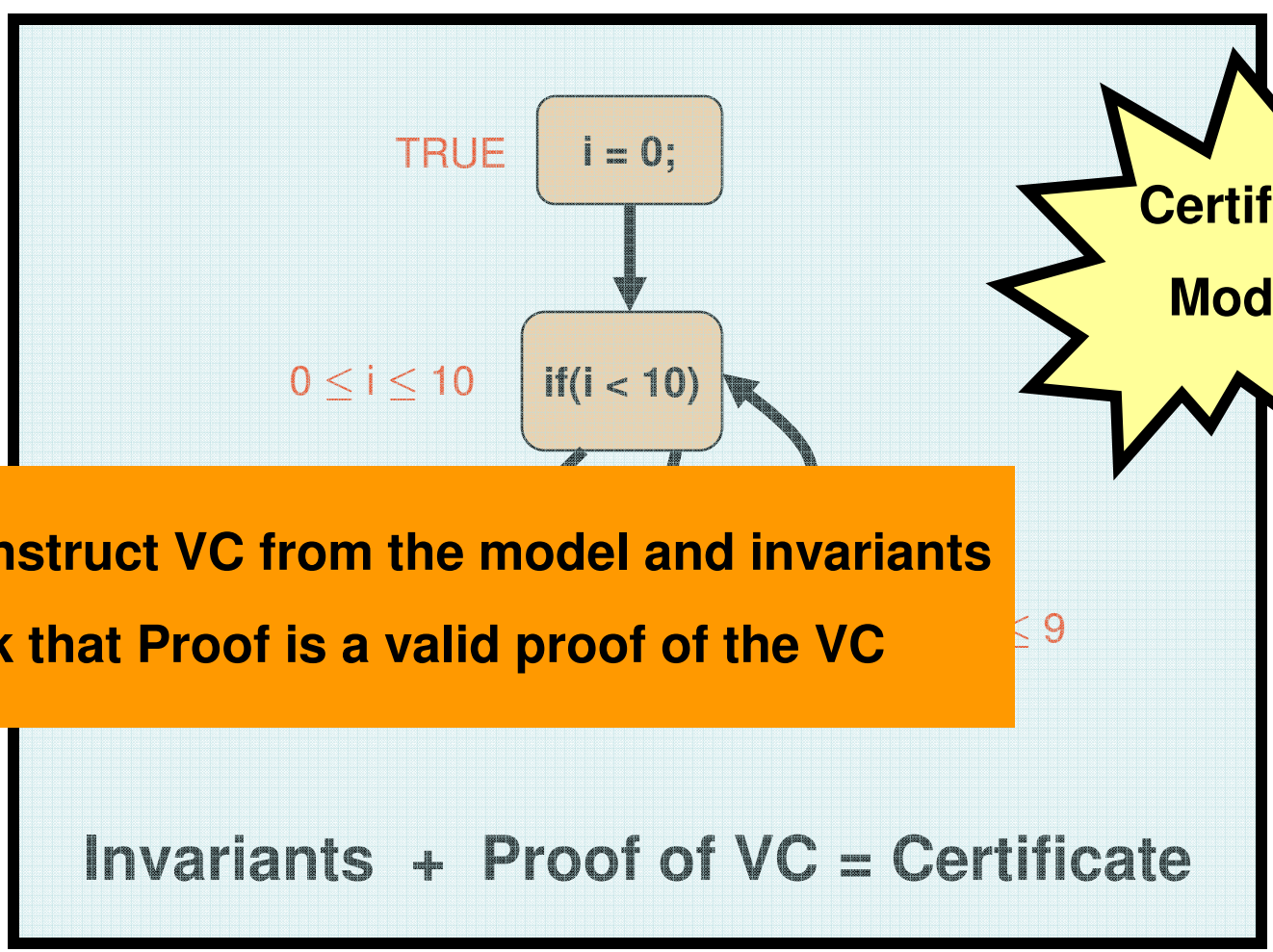
Model Certification: Step 2: Certification



Certified Model



Model Validation

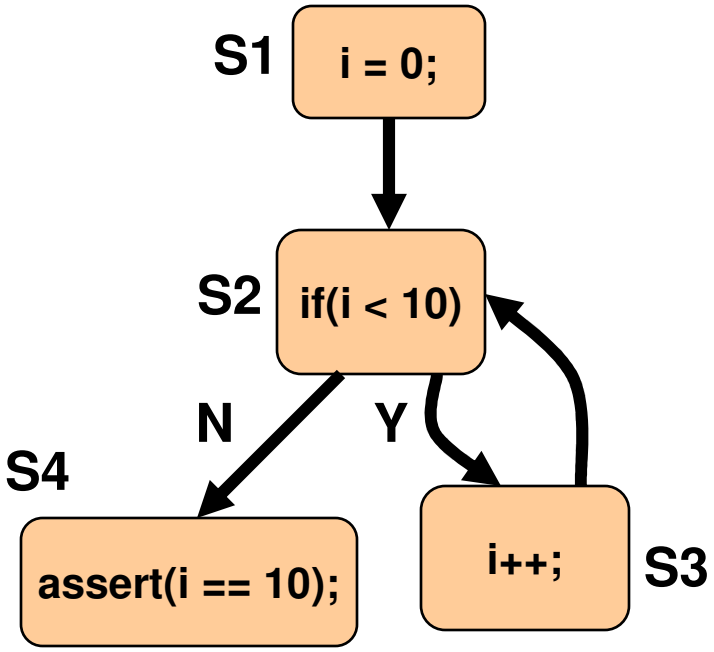


Certified Model

1. Reconstruct VC from the model and invariants
2. Check that Proof is a valid proof of the VC

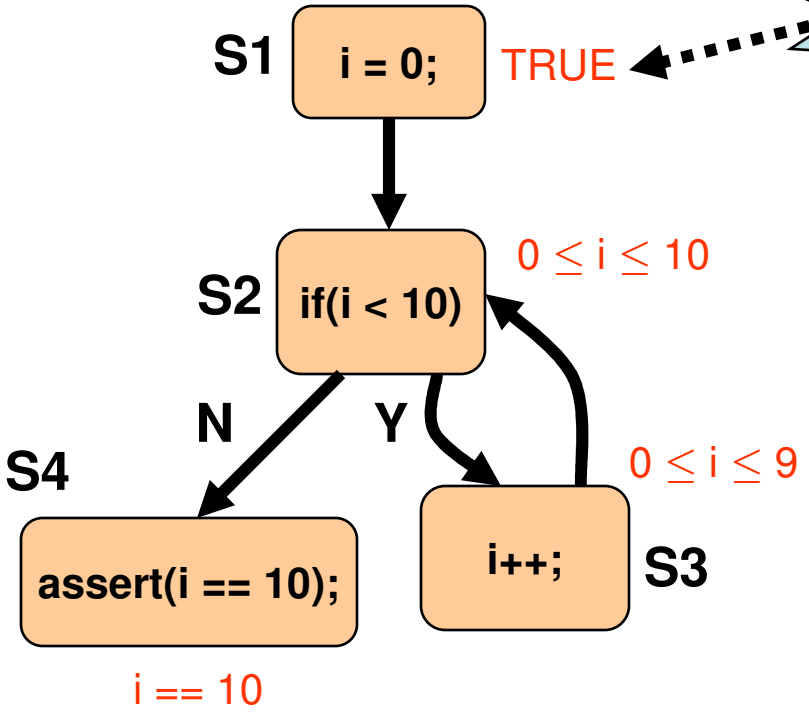
From Model to Code

Code Generation




```
enum State { S1,S2,S3,S4 };
State s = S1;
int i;
while(1) {

  if(s == S1) { i = 0; s = S2; }

  else if(s == S2)
    s = (i < 10) ? S4 : S3;

  else if(s == S3) { i++; s = S2; }

  else assert(i == 10);
}
```

Code Level Invariant Generation

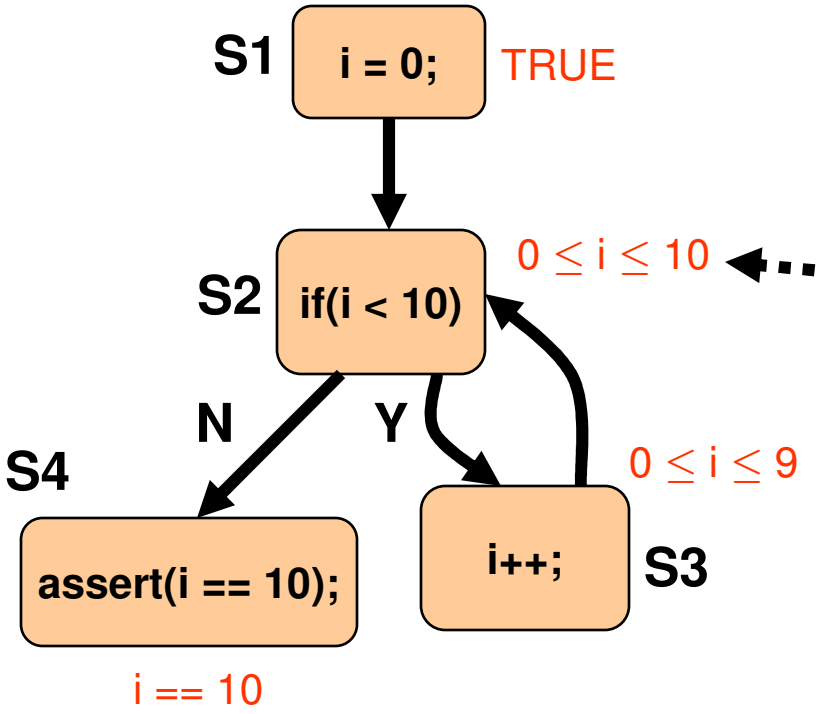


Automated

I1: s == S1 ∧ 1

```
enum State { S1,S2,S3,S4 };
S
  while (1) {

  if(s == S1) { i = 0; s = S2; }

  else if(s == S2)
    s = (i < 10) ? S4 : S3;

  else if(s == S3) { i++; s = S2; }

  else assert(i == 10);
}
```

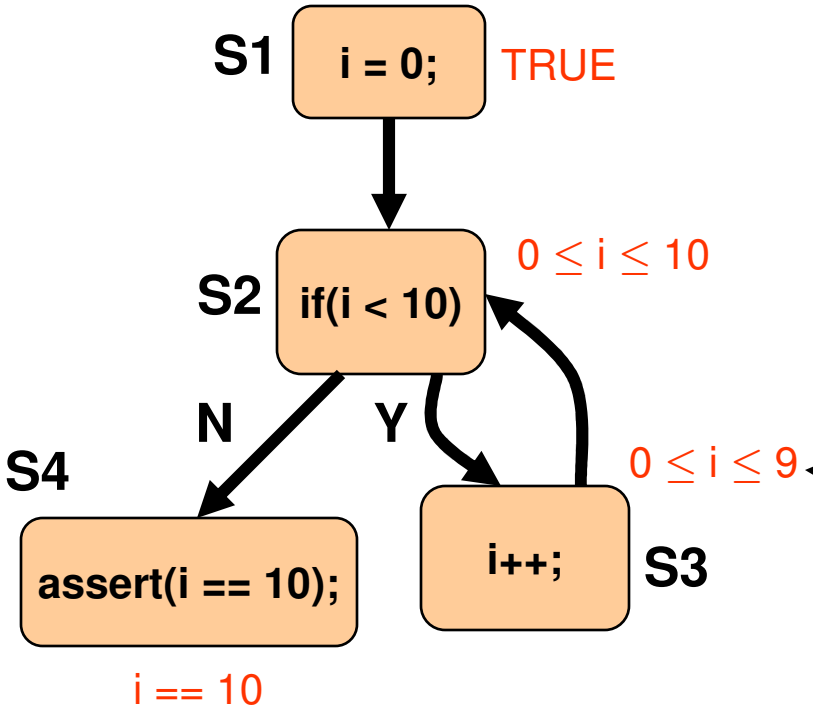
Code Level Invariant Generation




```
enum State { S1,S2,S3,S4 };
State s = S1;
int i;
while(1) {

  if(s == S1) { i = 0; s = S2; }

  else if(s == S2)
    s = (i < 10) ? S4 : S3;

  else if(s == S3) { i++; s = S2; }

  else assert(i == 10);
}
```

I2: s == S2 ∧ 0 ≤ i ≤ 10

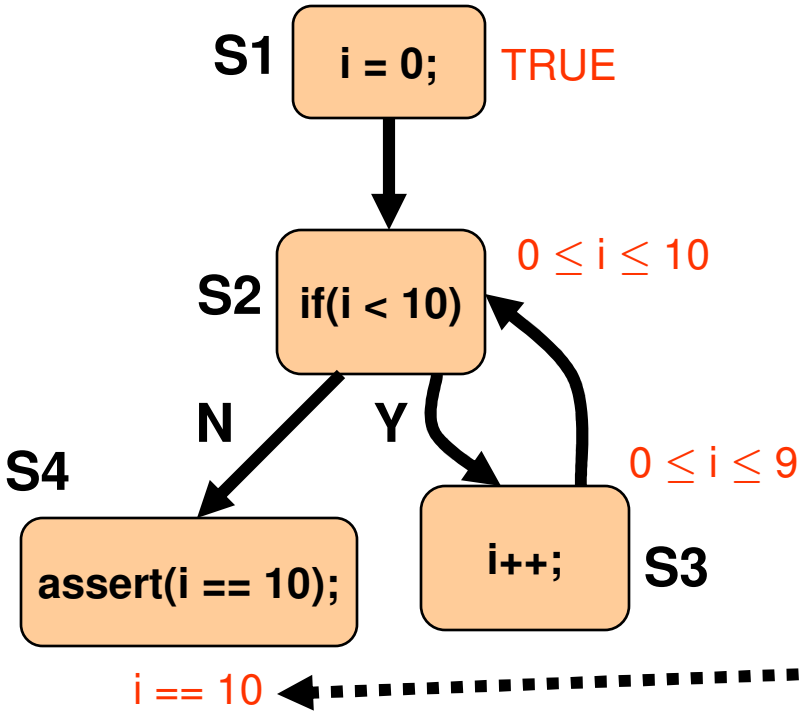
Code Level Invariant Generation




```
enum State { S1,S2,S3,S4 };
State s = S1;
int i;
while(1) {

  if(s == S1) { i = 0; s = S2; }

  else if(s == S2)
    s = (i < 10) ? S4 : S3;

  else if(s == S3) { i++; s = S2; }

  else assert(i == 10);
}
```

I3: s == S3 ∧ 0 ≤ i ≤ 9

Code Level Invariant Generation

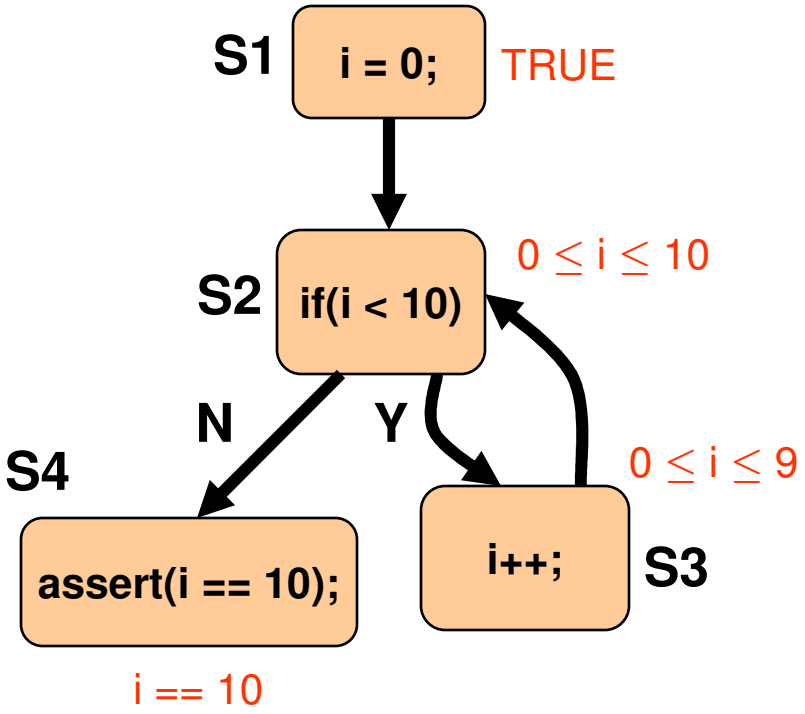



```
enum State { S1,S2,S3,S4 };
State s = S1;
int i;
while(1) {

  if(s == S1) { i = 0; s = S2; }

  else if(s == S2)
    s = (i < 10) ? S4 : S3;

  else if(s == S3) { i++; s = S2; }

  else assert(i == 10);
}
```

I4: s == S4 ∧ i == 10

Code Level Invariant Generation

Certification and Validation are similar to those in the model




```
enum State { S1,S2,S3,S4 };
State s = S1;
int i;
while(1) {

  if(s == S1) { i = 0; s = S2; }

  else if(s == S2)
    s = (i < 10) ? S4 : S3;

  else if(s == S3) { i++; s = S2; }

  else assert(i == 10);
}
```

I1 ∨ I2 ∨ I3 ∨ I4

I1: s == S1 ∧ 1

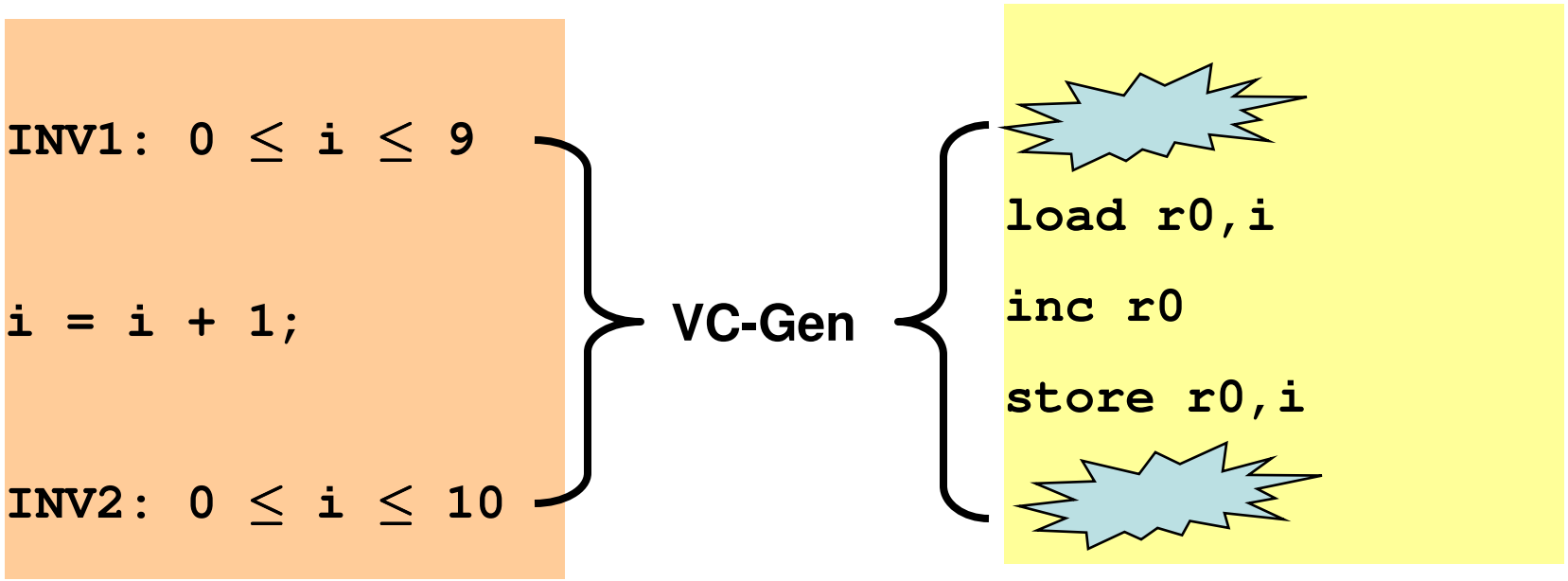
I2: s == S2 ∧ 0 ≤ i ≤ 10

I3: s == S3 ∧ 0 ≤ i ≤ 9

I4: s == S4 ∧ i == 10

From Code to Assembly

Compilation



Problem1: How do you "compile" the invariants?

Problem2: How do you generate the VC at the assembly level?

Compiling the Invariants

```
BEGIN();
INV(0 ≤ i ≤ 9);

i = i + 1;

BEGIN();
INV(0 ≤ i ≤ 10);
```

Compiling the Invariants

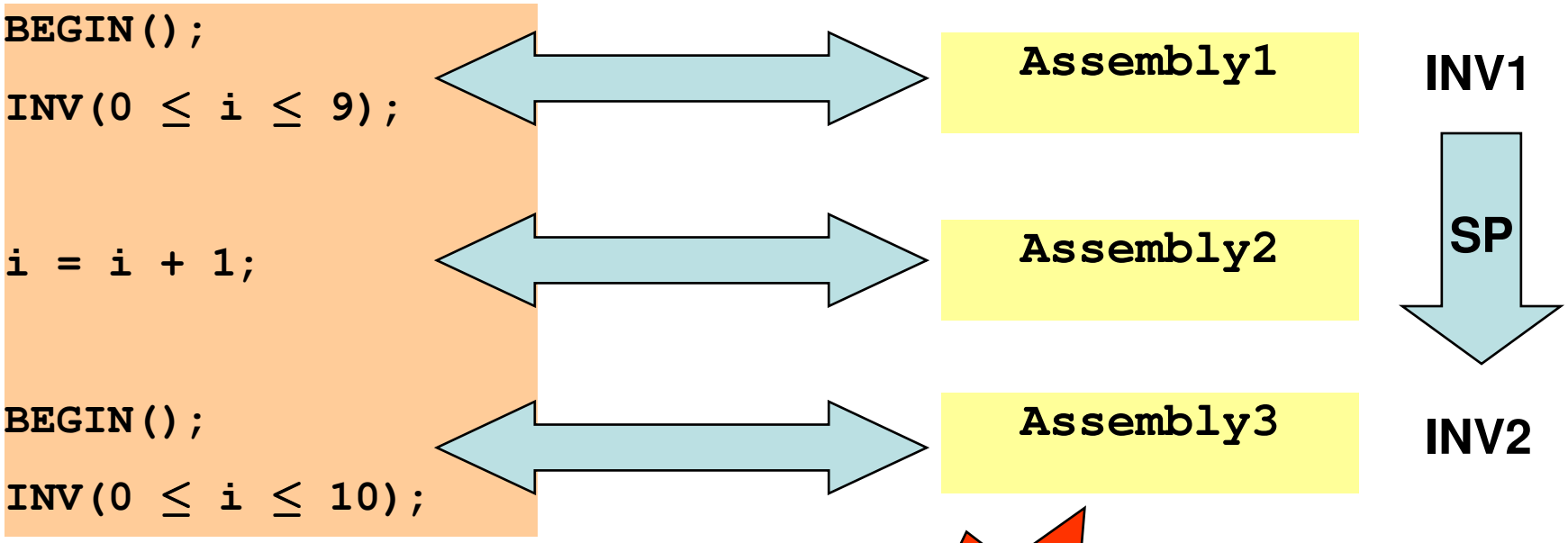
Calling convention: "r0" is used to pass the argument to INV.

```
BEGIN();

INV(0 ≤ i ≤ 9);
```

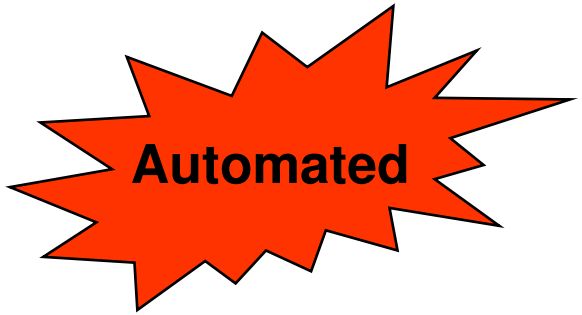
```
call BEGIN

load r0,0

blt i 0 L1

bgt i 9 L1

store r0 1

L1: call INV
```

The weakest precondition of "r0" w.r.t. this program is "0 ≤ i ≤ 9". But that's precisely the invariant we compiled.

VC-Gen at the Assembly Level



Certification and Validation are similar to those in the model



Experimental Evaluation

Full Process from Model to Assembly

- Toy example with a “counter”
- PACC[1] in-house code generator
- gcc PowerPC compiler

From C to Assembly

- Micro-C embedded OS (6000 LOC)
  - certified that a “lock” was being “acquired” and “release” correctly
- Gnu “tar” implementation in the “Plan 9” implementation
  - Certified that a particular buffer will never overflow

[1]Predictable Assembly from Certifiable Components (PACC): http://www.sei.cmu.edu/pacc

Caveats

Obtaining model-level invariants non-trivial

- Manual assistance (annotations)

Code generator and compiler optimizations (reordering)

- Control code-generation and compilation

Non-functional properties

- Need a logical way to reason about them

Domain-specific proof-Engineering decisions

- Apply in industrial setting

What if doing it this way is just too hard?

- Cryptographic approach: *Verification Across Intellectual Property Boundaries*, Chaki, Schallhart, Veith, Proceedings of CAV, 2007

Questions?

chaki@sei.cmu.edu

Software Engineering Institute | Carnegie Mellon