MaC
Monitoring and Checking at Runtime

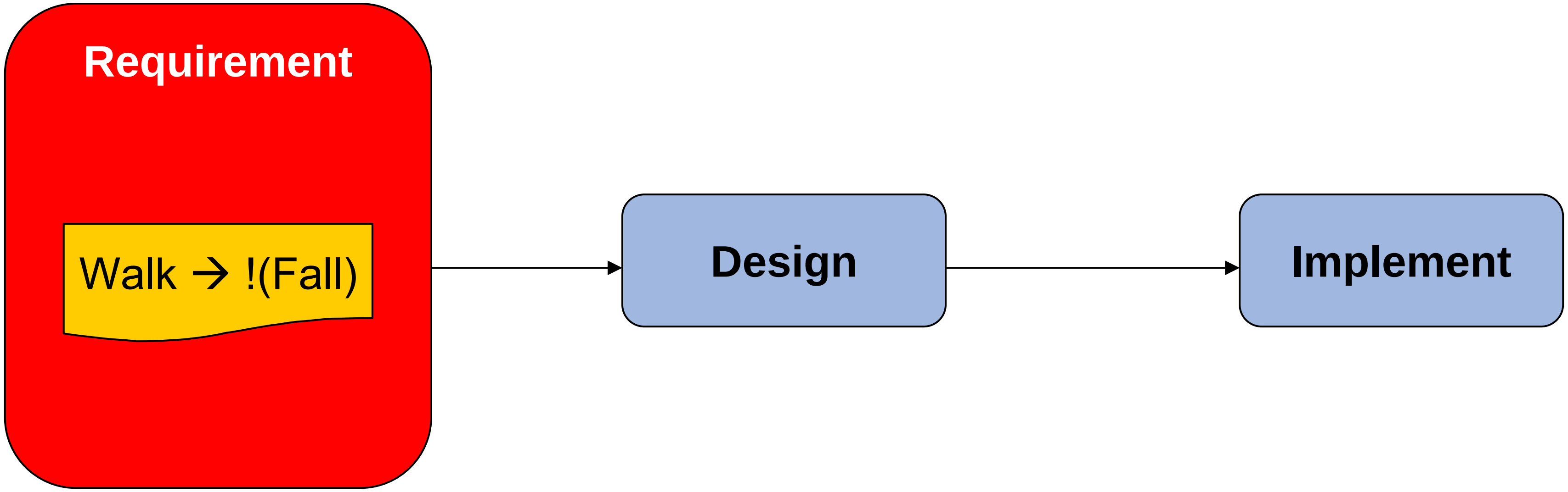
Presented By
Usa Sammapun
CIS 700 Oct 10, 2005

What is MaC?

- A verification technique
  - Goal: **Ensure a software program runs correctly**
- To understand software verification
  - Know how software is developed
  - Know how software is verified

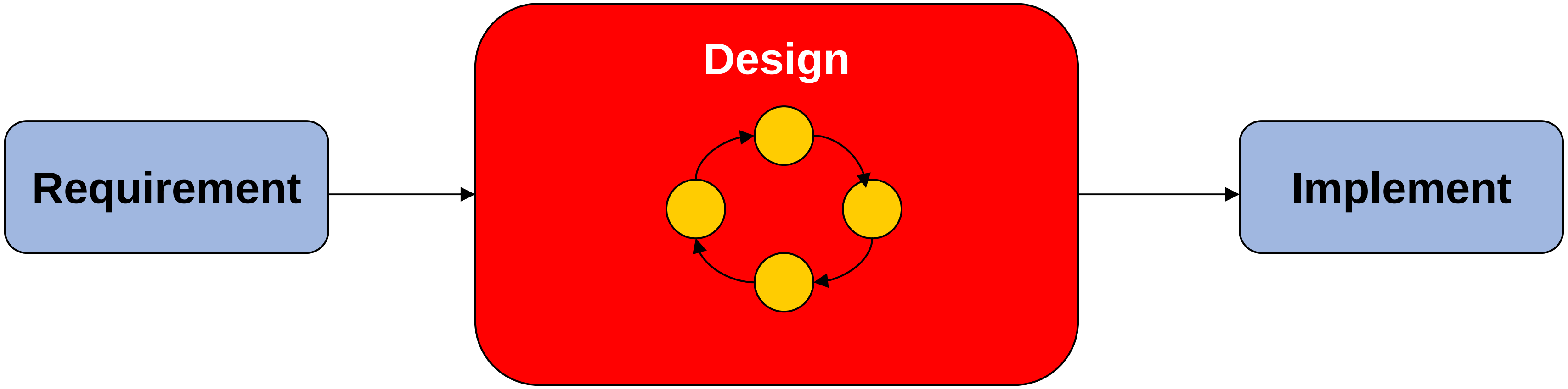
Software Development Process

- Requirement and Properties
  - What program should do
    - **When AIBO dog walks, it must not fall**
  - Informal (English) → Formal (Logic, FSM)



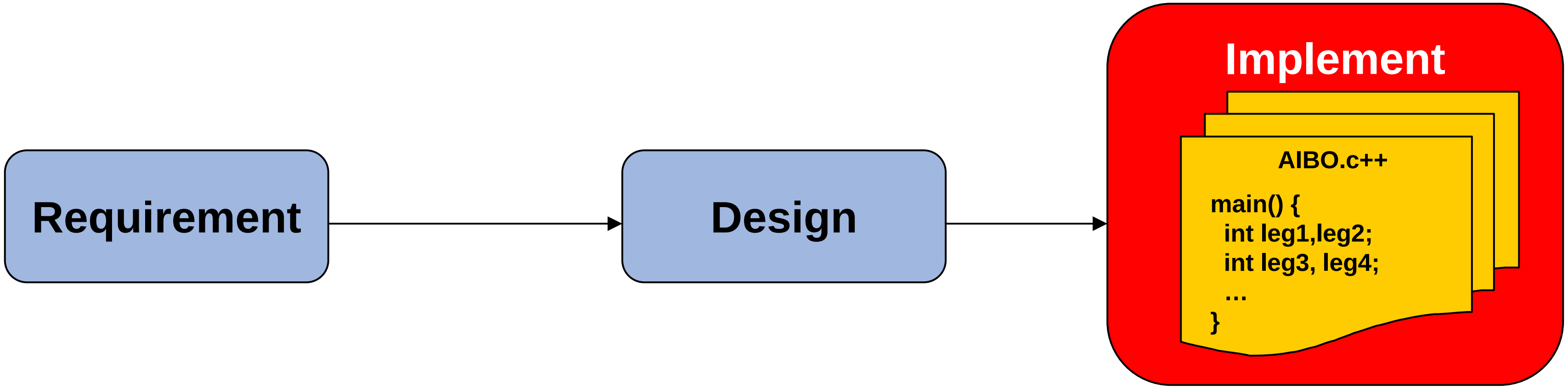
Software Development Process

- Design Specification and Analysis
  - How program fulfill requirements
    - **AIBO dog coordinates his 4 legs**
  - Formal modeling (UML, FSM, Control theory)
  - Analysis
    - Simulation
    - Verification (Model checking)



Software Development Process

- Implementation
  - Actual program (AIBO dog walking program in C++)
  - Varification & Validation
    - Testing
    - Runtime verification



Verification

- Design
  - Model Checking
- Implementation
  - Testing
  - Runtime Verification

MaC
Monitoring and Checking
At Runtime

Penn
University *of* Pennsylvania

Verification

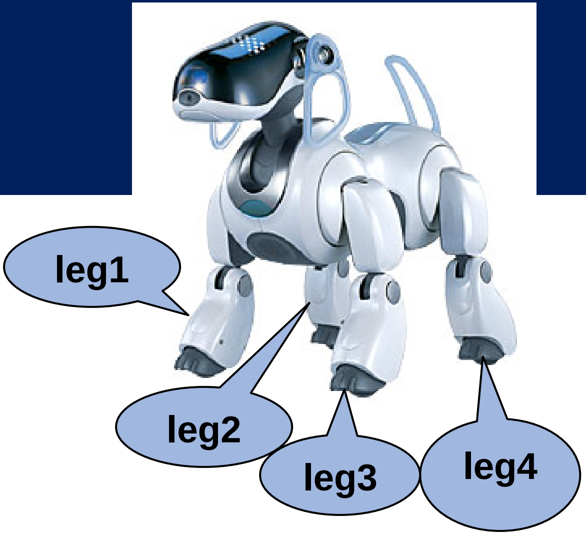
- Design
  - <u>**Model Checking**</u>
- Implementation
  - Testing
  - Runtime Verification

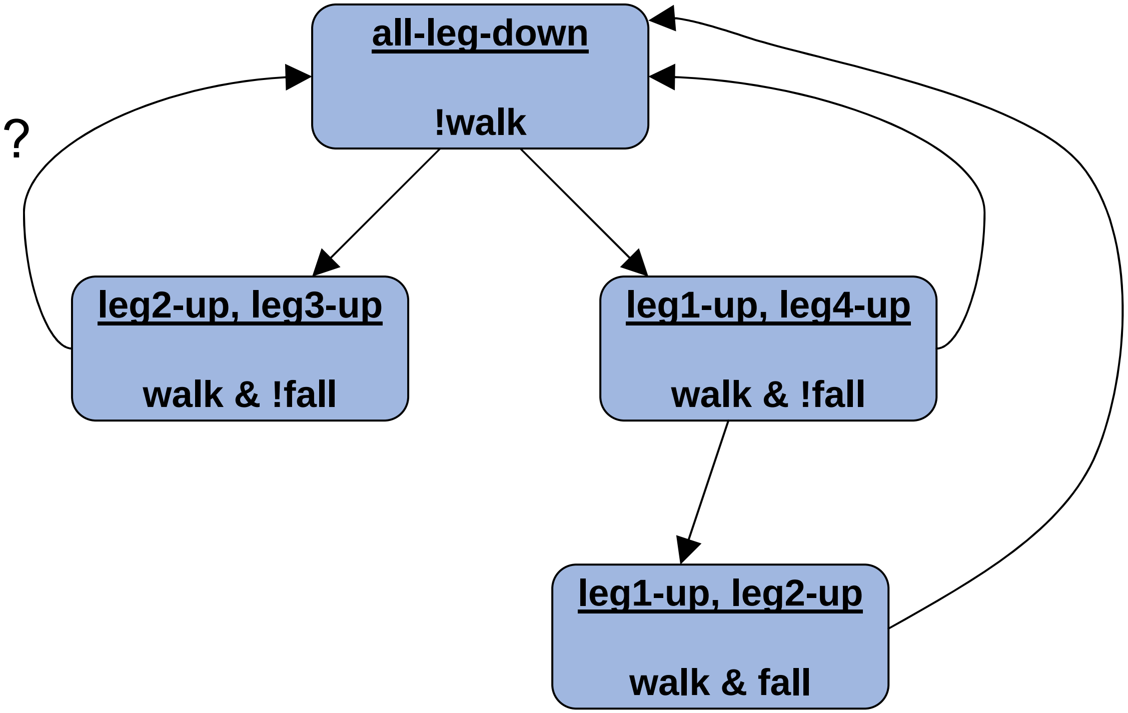
Penn
University *of* Pennsylvania

Model Checking

- **Given**
  - Requirement & Properties
  - Model
- **Verify**
  - Explore all paths
  - Violate requirement ??

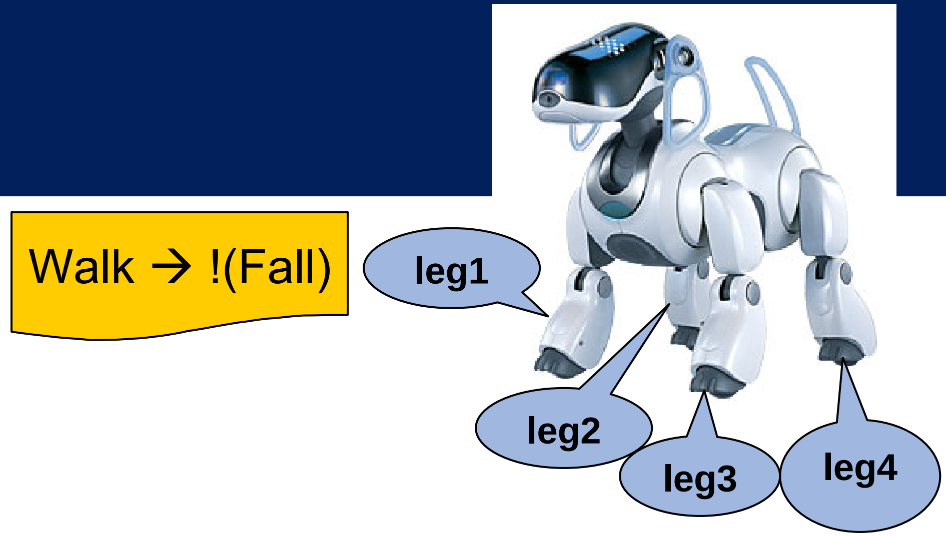
Walk → !(Fall)

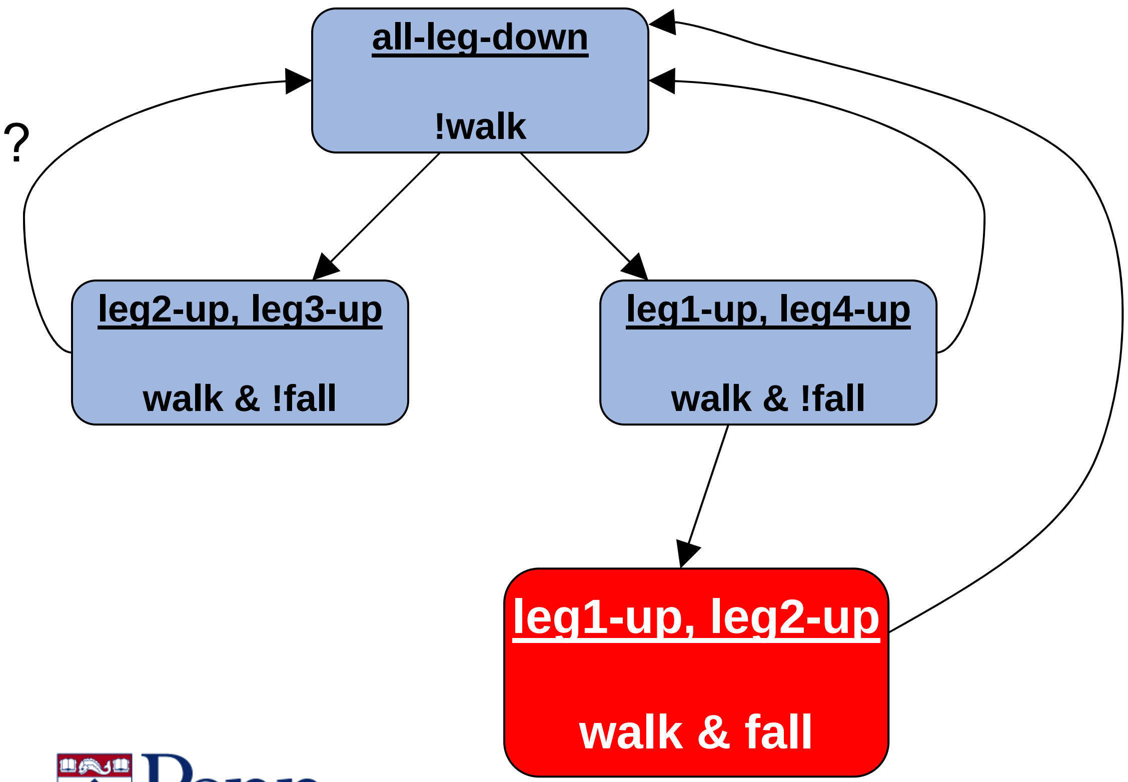




Model Checking

- **Given**
  - Requirement & Properties
  - Model
- **Verify**
  - Explore all paths
  - Violate requirement ??





Model Checking - GOOD

- **Rigorous and Formal**
  - Based on Mathematics
- **Complete**
  - Explore all paths

Model Checking - PROBLEM

- **Check Design, not implementation**
  - What if implementation does not follow model?

- **Not scalable**
  - What if the program is HUGH?
    - Explore all paths might not be feasible

Verification

- Design
  - Model Checking
- Implementation
  - **Testing**
  - Runtime Verification

Penn
UNIVERSITY of PENNSYLVANIA

Testing

- **We've seen it**
  - Run actual program with different inputs
  - See if outputs are what we want
- **Ex. AIBO**
  - Run AIBO dog
  - See whether or not AIBO dog falls
- **Good**
  - Check directly the implementation

Testing – PROBLEM

- **Not rigorous, Not formal**
  - Possibly random inputs

- **Not complete**
  - What if bugs never show up during test ??
  - What if it's not AIBO, but a heart device !?!

Verification

- Design
  - Model Checking

- Implementation
  - Testing
  - **Runtime Verification**

MaC
Monitoring and Checking
At Runtime

Penn
University *of* Pennsylvania

Runtime Verification

Walk → !(Fall)

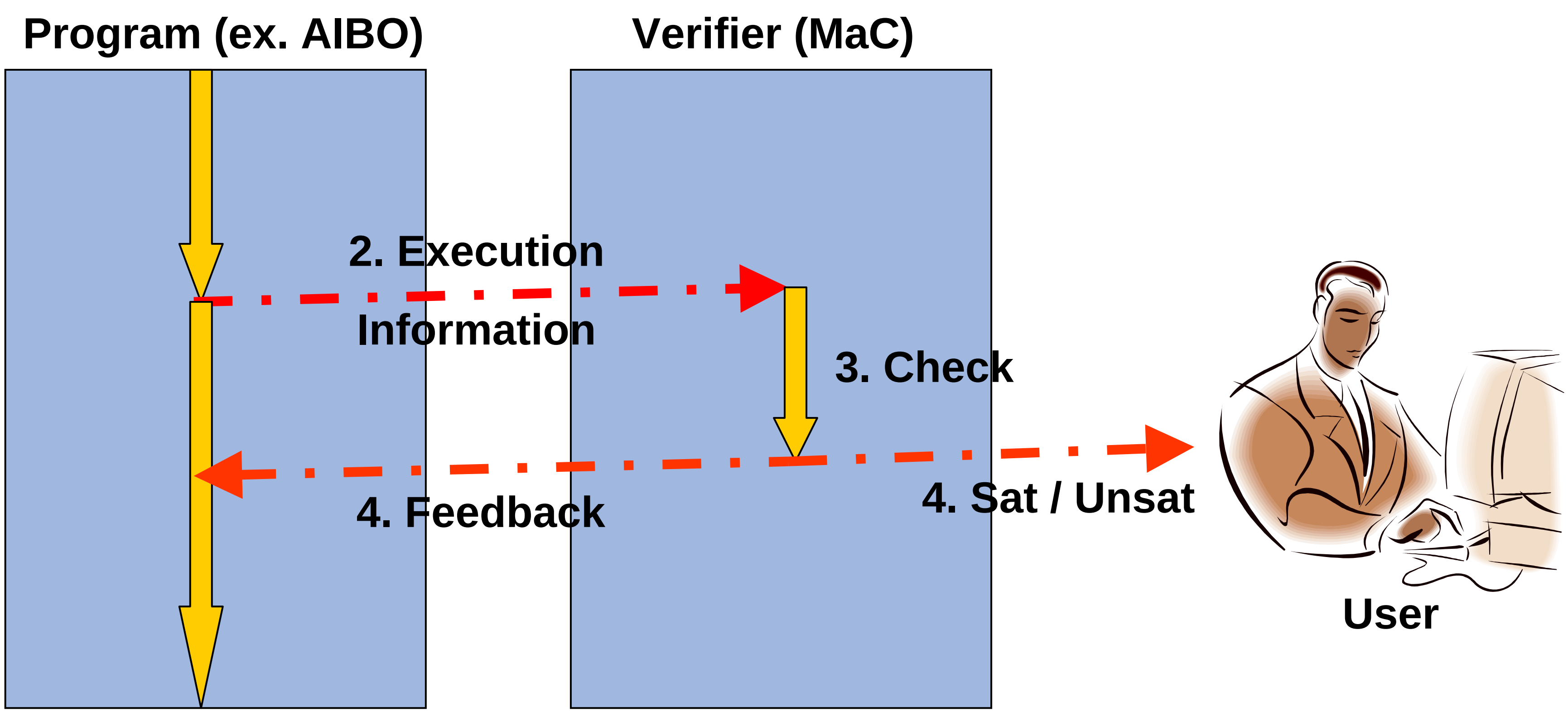
- **Given**
  - Requirement & Properties
  - Implementation

- **Ensures the current program execution follows its formal requirements**

Penn
University *of* Pennsylvania

Runtime Verification

1. Specify formal requirements **(Walk → !Fall)**



Penn
University of Pennsylvania

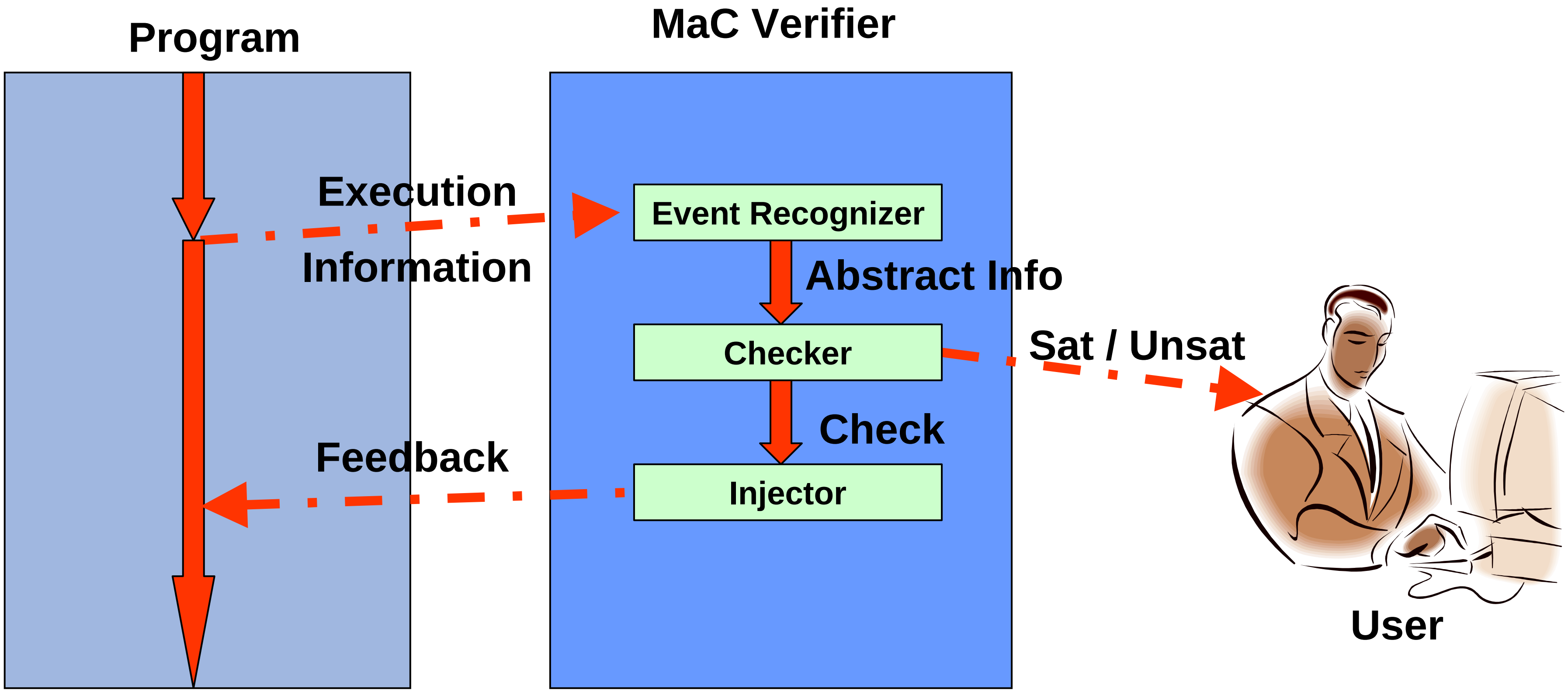
Runtime Verification

- Rigorous and Formal
- Done at implementation
- Not complete
  - Guarantee for current execution

Runtime Verification

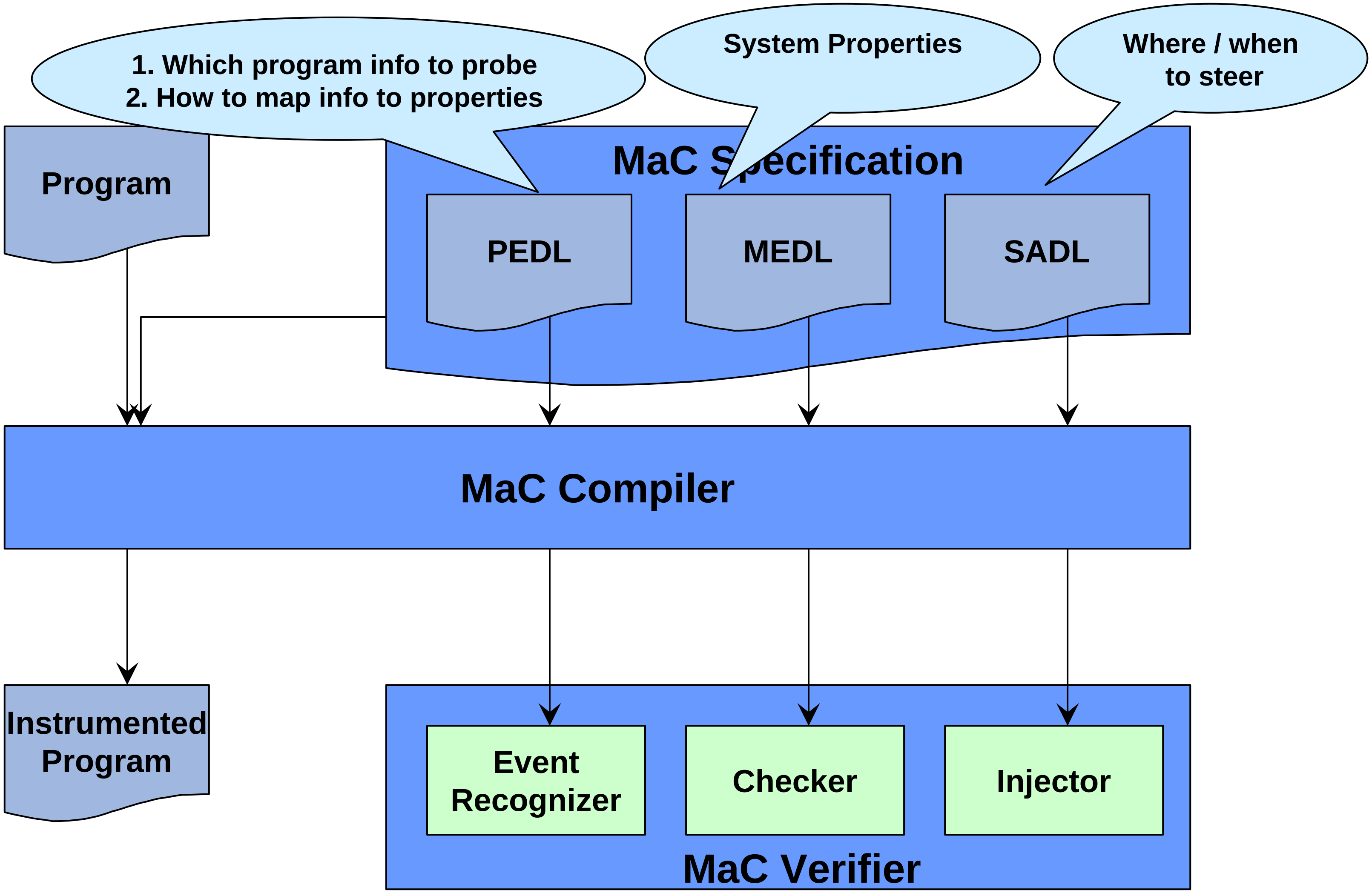
- MaC
  - Monitoring and Checking at Runtime
  - **Components**
    - MaC verifier
    - MaC formal language

MaC Verifier



Penn
University of Pennsylvania

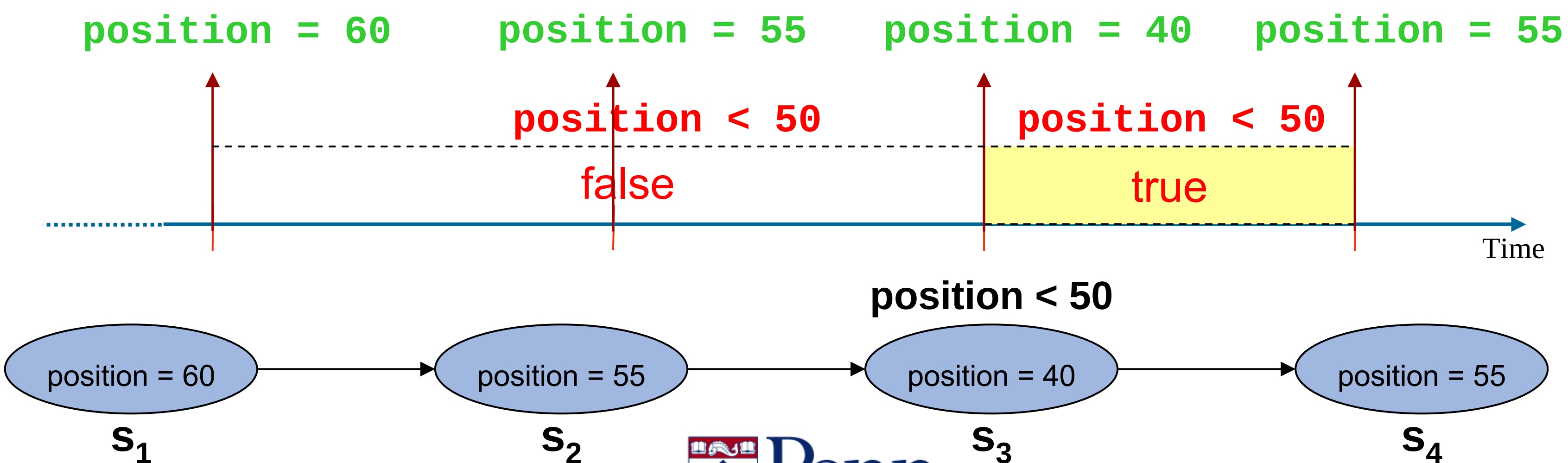
MaC Verifier and Language



Abstract Information

- **To capture roughly and abstractly what the program is doing**

- **Events**
  - Instantaneous incidents
  - such as variable updates **update(position)**
- **Conditions**
  - Proposition about the program that may be ***true/false/undefined*** for a duration of time
  - *such as* ***position < 50***



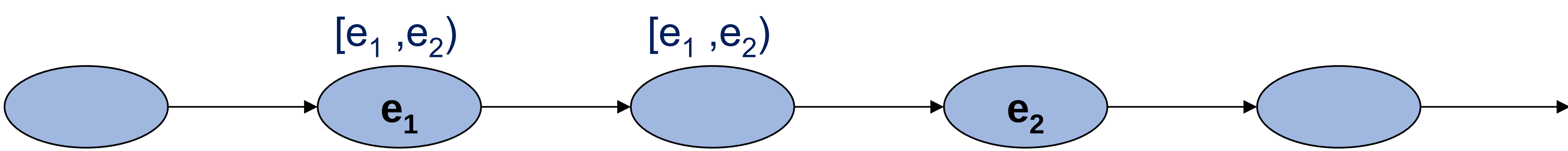
Events

- **e** - variable update, start/end method
- **e1 || e2** - or
- **e1 && e2** - and
- **start(c)** - instant when condition c becomes true
- **end(c)** - instant when condition c becomes false
- **e when c** - e occurs when condition c is true

Conditions

- Conditions interpreted over 3 values: true, false and undefined.

- **c** - boolean expression
- **!c** - not c
- **c_1 || c_2** - or
- **c_1 && c_2** - and
- **c_1 -> c_2** - imply
- **defined(c)** - true when c is defined
- **[e_1, e_2)** - interval

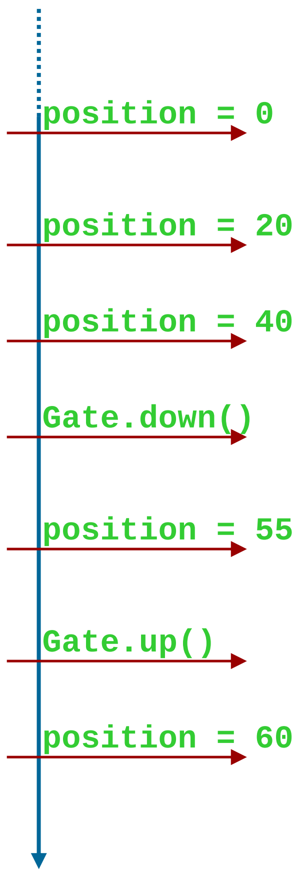


MaC Language

- PEDL
  - How execution information transform into events and conditions
- MEDL
  - Specify properties using events and conditions

PEDL and MEDL

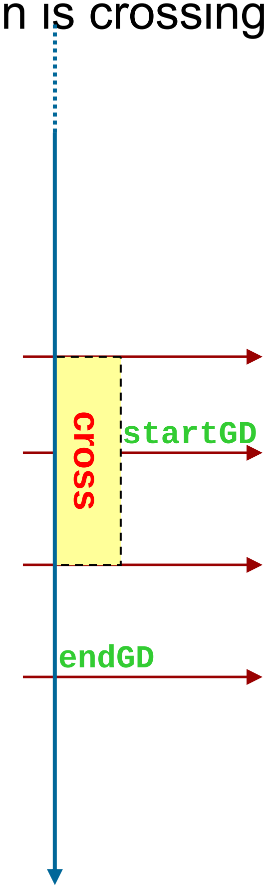
Railroad Crossing Property: - If train is crossing, then gate must be down
- Train is crossing when position is between 30 and 50



Java Program

Abstraction

- When train position is between 30 and 50
- When gate starts/ends being down



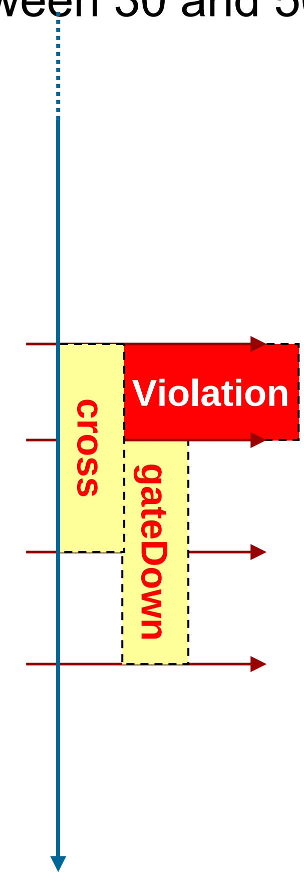
PEDL

Abstraction

- When gate is down

Check

- If train is crossing, then gate must be down



MEDL

Instrumentation

```
class Train {
  int position;
  main() {
    position = 0;
    position = 20;
    position = 40;
    position = 55;
} }
```

+

```
Train.position;
```

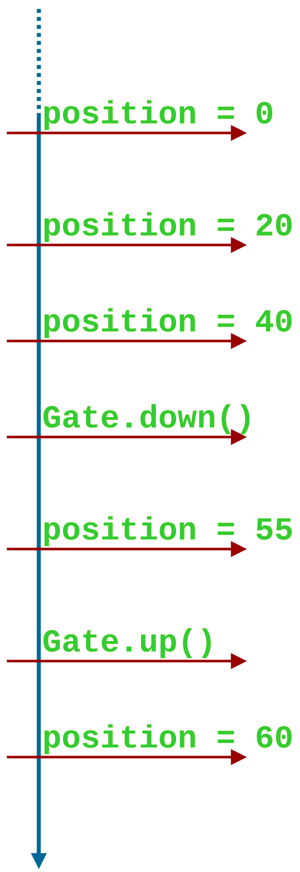
=

```
class Train {
  int position;
  main() {
    position = 0;
    send(x,0);
    position = 20;
    send(x,20);
    position = 40;
    send(x,40);
    position = 55;
    send(x,55);
}  }
```

Sent to Event Recognizer:
[(position,0), (position,20), (position,40), (position,55)]

MaC Language - PEDL

Railroad Crossing Property: - If train is crossing, then gate must be down
- Train is crossing when position is between 30 and 50



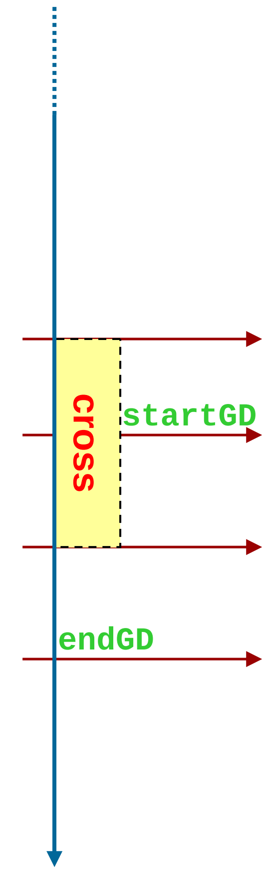
Abstraction

- When train position is between 30 and 50
- When gate starts/ends being down

```
export event startGD, endGD;
export condition cross;

monobj Train.position;
monmeth Gate.up();
monmeth Gate.down();

condition cross = (30 < RRC.position)
&& (RRC.position < 50);
event startGD = endM(Gate.down());
event endGD = startM(Gate.up());
```



Java Program **PEDL**

MEDL – Property Language

- **Composed using**
  - Events
  - Conditions
  - Connectives

- **Properties**
  - ***Alarms:*** events that must never occur

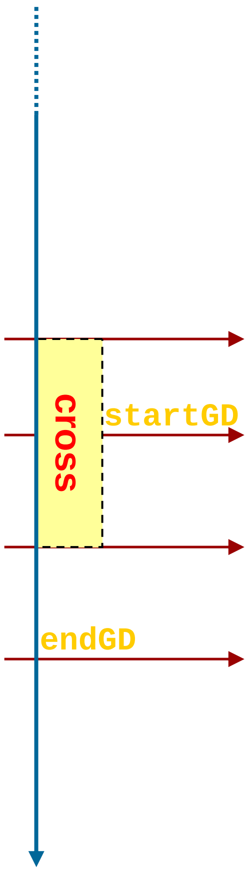
    alarm elevator = door_open when ! floor_level

  - ***Safety Properties:*** conditions that must always hold true

    property rail_road = train_cross → gate_down

MaC Language - MEDL

Railroad Crossing Property: - If train is crossing, then gate must be down
- Train is crossing when position is between 30 and 50



PEDL

Abstraction

- When gate is down

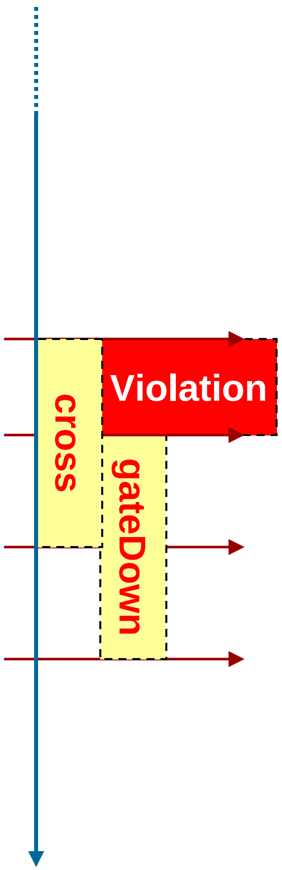
Check

- If train is crossing, then gate must be down

```
import event startGD, endGD;
import conditions cross;

condition gateDown = [startGD, endGD);
property safeRRC = cross -> gateDown;
```

MEDL

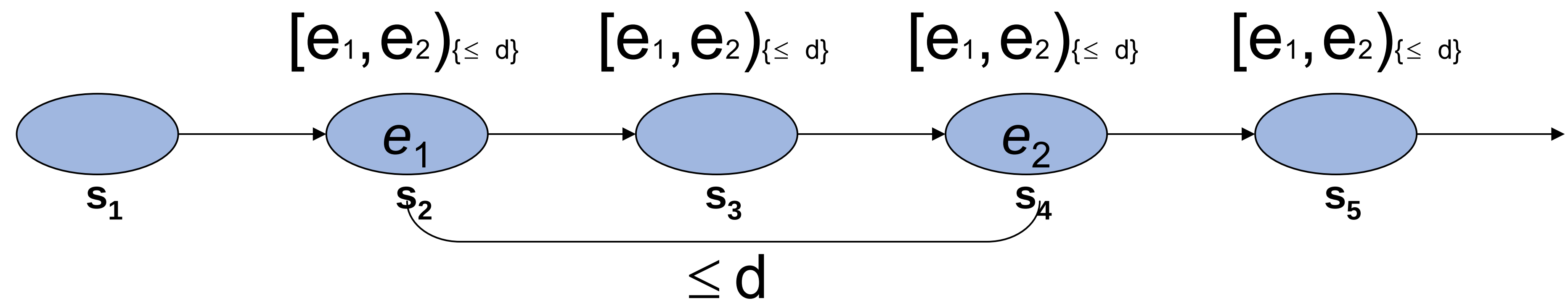


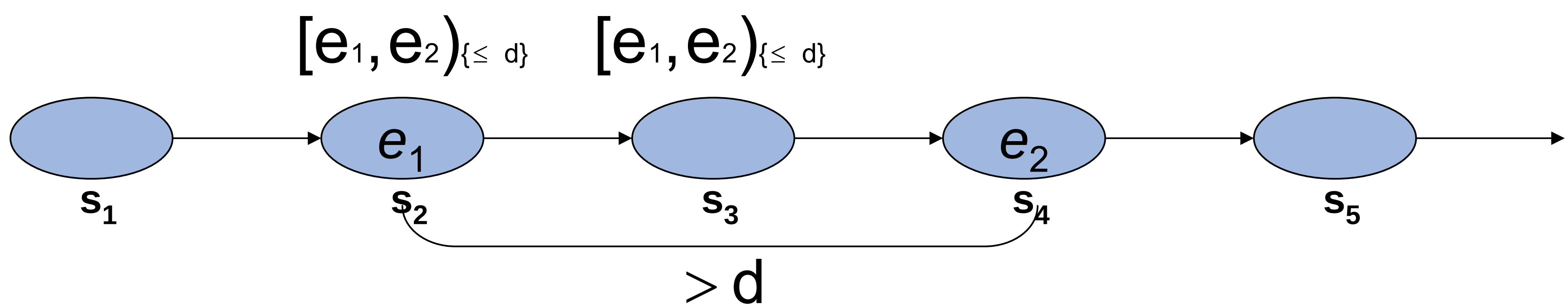
Current Work

- Timing properties
- Probabilistic properties
- Dynamic MaC
- Steering using control theory

Quantitative Properties

- Time bound interval: $[e_1, e_2)_{\{\leq d\}}$ $[e_1, e_2)_{\{< d\}}$ $[e_1, e_2)_{\{= d\}}$





Example

- A real-time task T must finish within 100 time units
  - *startT* – event when task T starts executing
  - *endT* – event when task T finishes executing

$$[startT, endT)_{\{\leq 100\}}$$