# Barrier Slicing for Remote Software Trusting

*Ceccato Mariano[1], Mila Dalla Preda[2],*
*Jasvir Nagra[2],*
*Christian Collberg[3], Paolo Tonella[1]*

*[1]Fondazione Bruno Kessler-IRST, Trento, Italy*
*[2]University of Trento, Italy*
*[3]University of Arizona, USA*

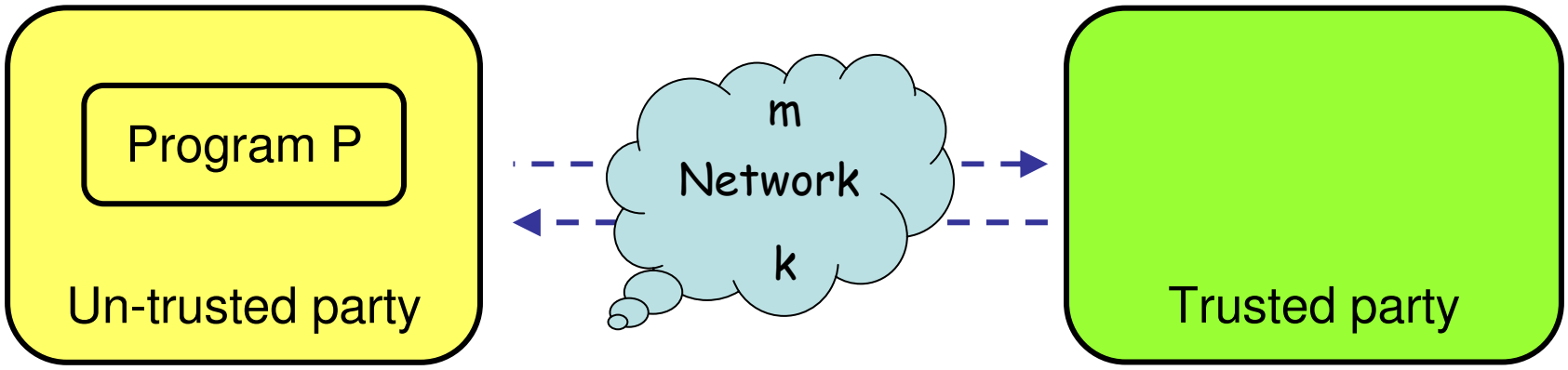

# Outline

- Problem definition
- Attack model
- Barrier slicing
- Preliminary results
- Future works

# Problem definition

- Network application, that needs a services by the trusted party.
- Trusted party means to deliver the services only to clients that can be trustred.

- s: state of the program P
- m = f(s)
- k = g(m)
  = g(f(s))



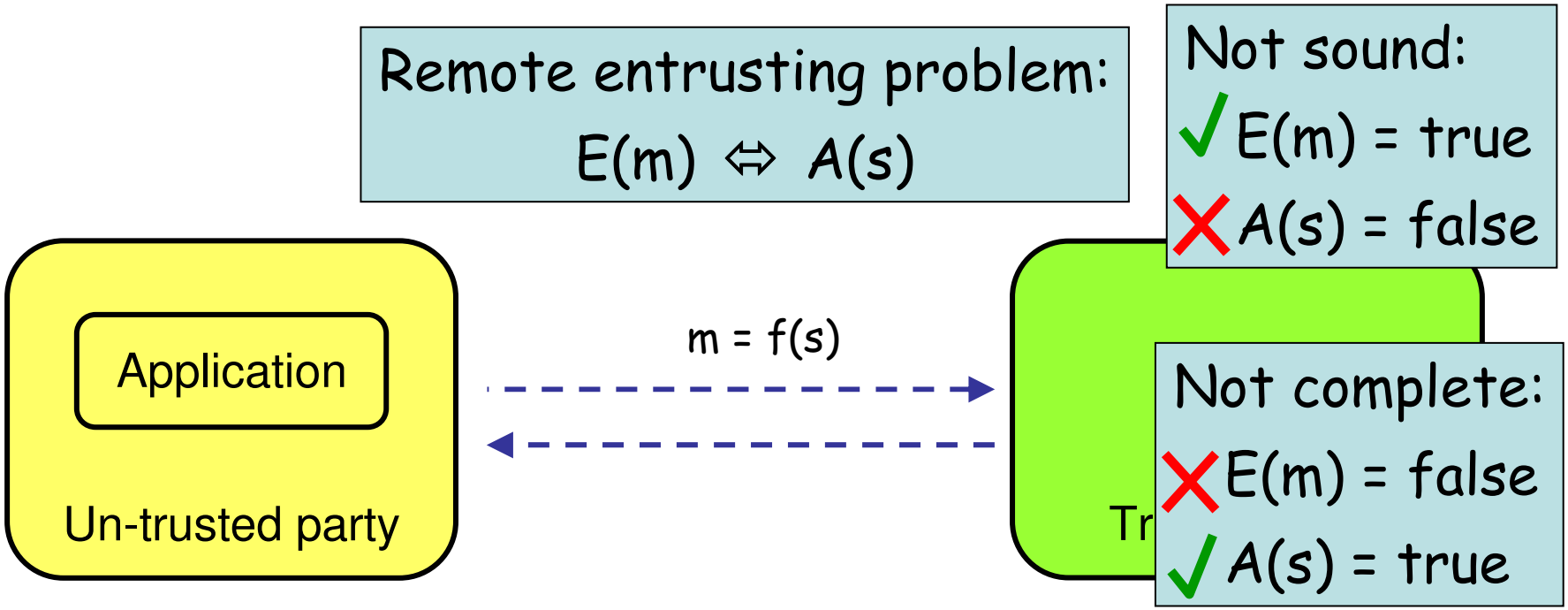
# Problem definition

**P is a valid state:**

A(s) = true

**P is entrusted:**

E(m) = true



# Attack model

Attacker on untrusted host:

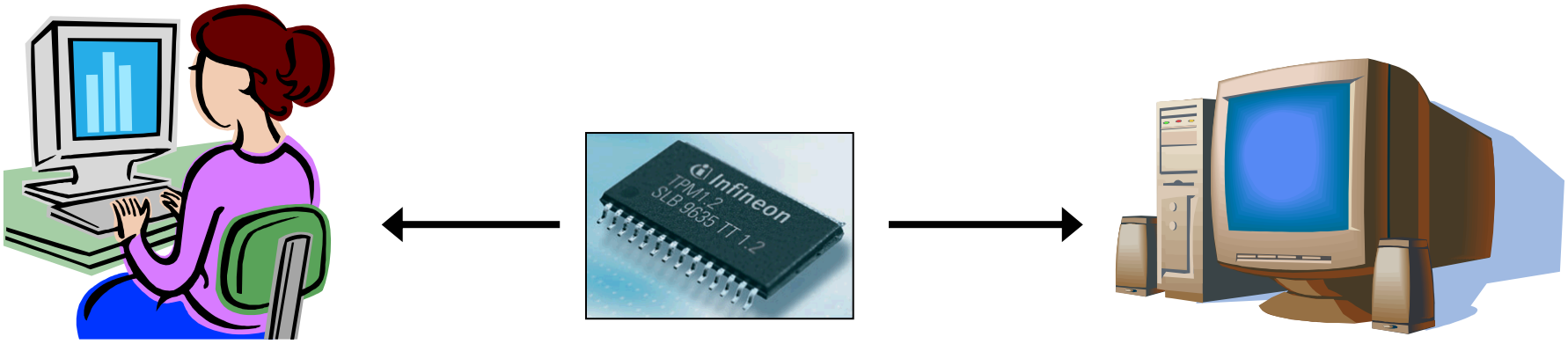
- Any dynamic/static analysis tool
- Any software (buggers, emulators, …)
- Read/write any memory location, register, network message, file.

Attacks:

- Reverse engineer and direct code change.
- Runtime modification of the memory.
- Produce (possibly tampered) copies of P that run in parallel.
- Interception and change of network messages.

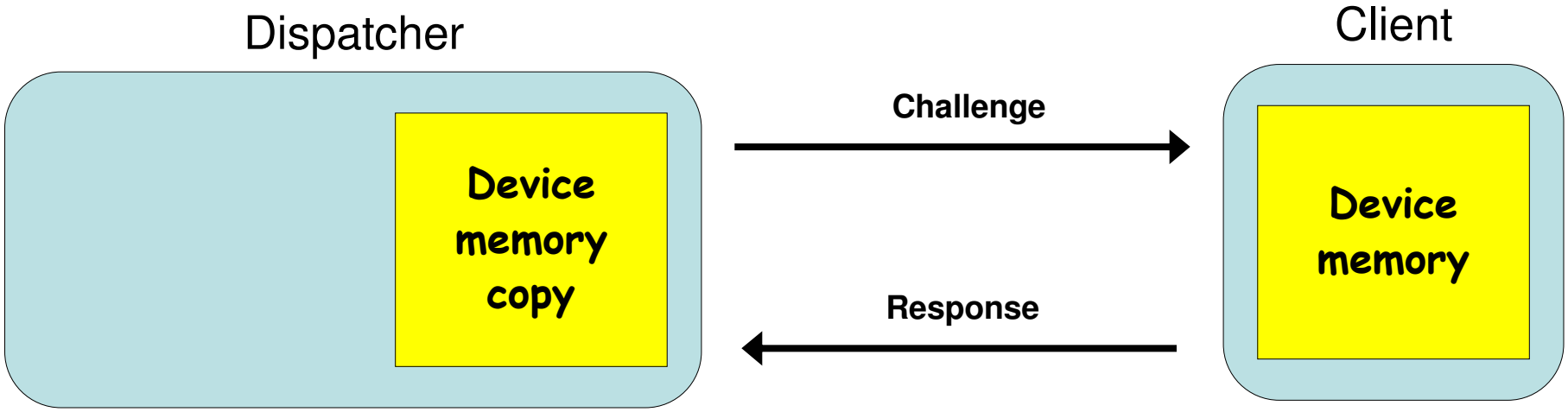
# Hardware based attestation

- Special hardware (TPM) is used to measure the state of the platform during the boot process.
  - Difficult to update
  - Costly
- Malicious code is detected because it causes measurements to deviate from the expected values.



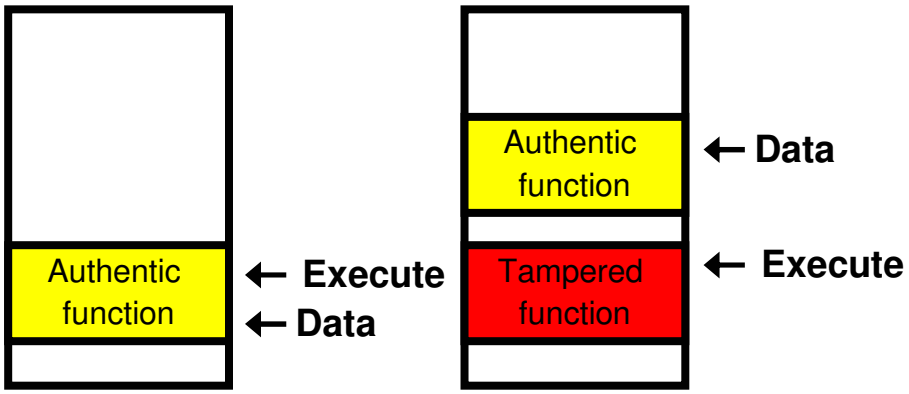
# Software based attestation

- Software based primitive to verify code execution on an un-trusted host
  - It can be updated.
  - No special purpose hardware is required.
  - It provides run-time attestation.
- It is based on
  - Challenge-response protocol.
  - Predictable checker execution time.



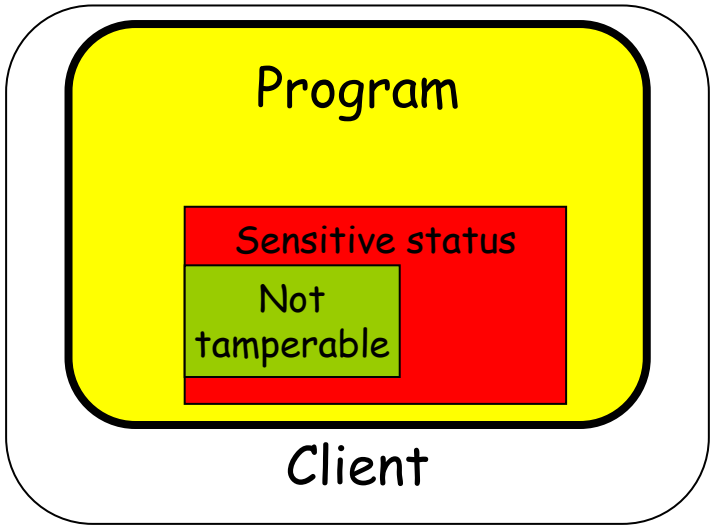
# Vulnerability

- A tampered program is running.
- The attacker computes the checksum on a correct copy.
- This attacks requires a small execution time overhead.
  - Accurate execution time prediction is mandatory to reveal this attack.



# Program state partition

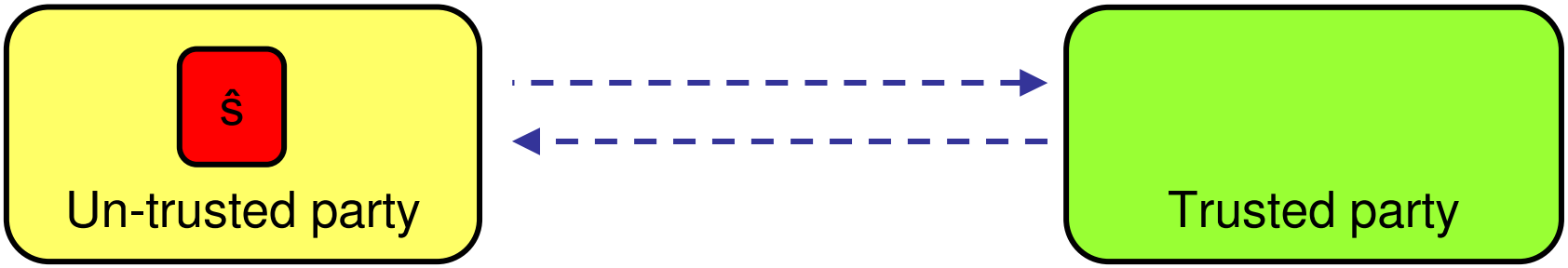
- There is a limited status (set of program variables) in an application that we are interested in protecting.
- A sub-portion of this state ($s_{|safe}$) can not modified by the user, otherwise
  - The client would receive a not-usable service or
  - The server would notice it




$$s = s_{|safe} \cup s_{|unsafe}$$

$$A(s) = A_{safe}(s_{|safe}) \wedge A_{unsafe}(s_{|unsafe})$$

# State tampering




$\hat{s}_{|safe}$ is sent:

- $A_{safe}(\hat{s}_{|safe})$ = false,
- tampering is detected

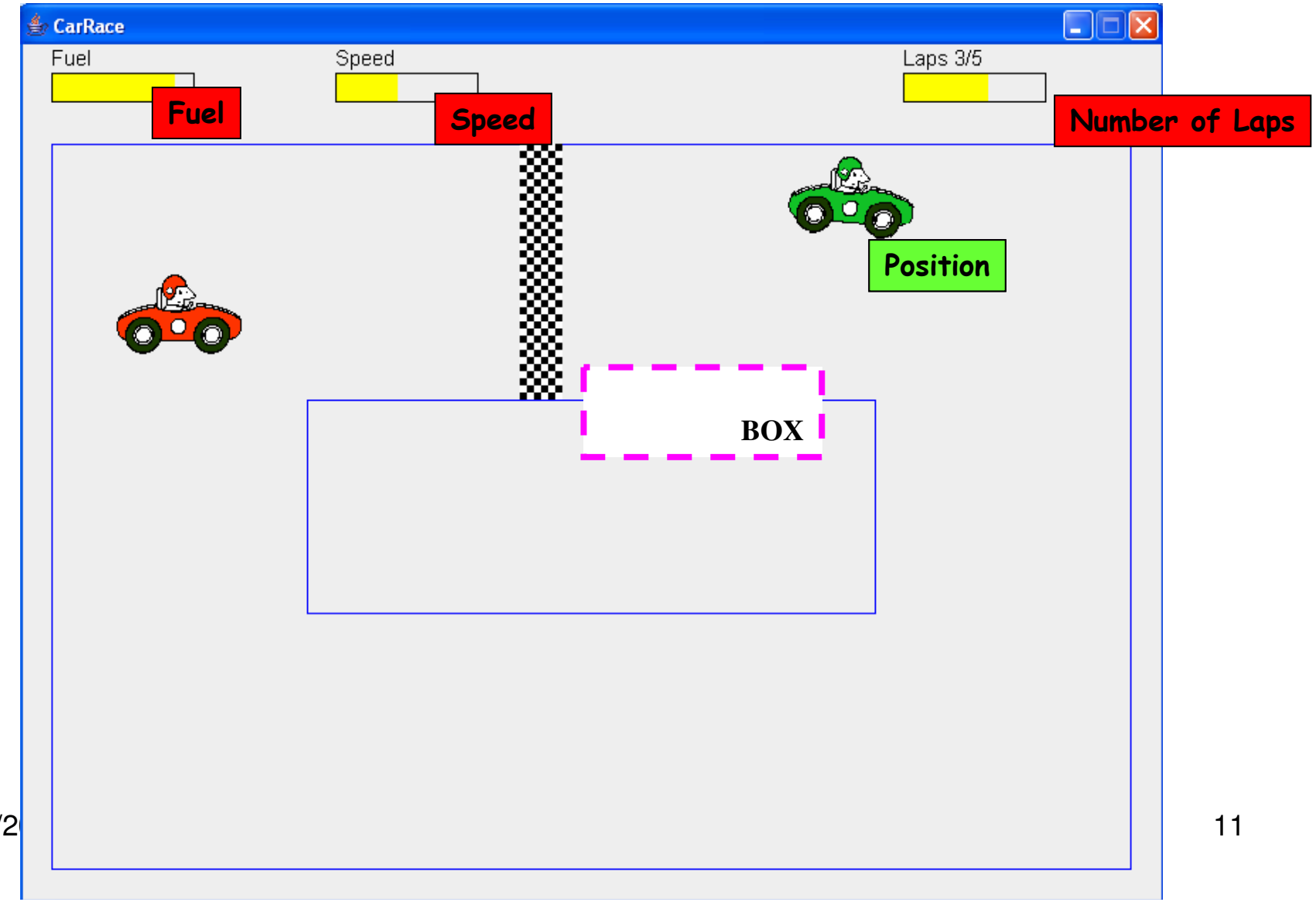
$s_{|safe}$ (!= $\hat{s}_{|safe}$) is sent:

- $A_{safe}(s_{|safe})$ = true,
- Service is not usable
- Tampering is useless

$$\hat{s} = \hat{s}_{|safe} \cup \hat{s}_{|unsafe}$$

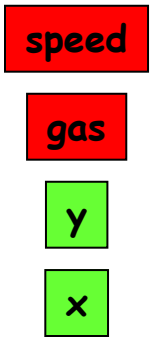
$$A(s) = A_{safe}(s_{|safe}) \wedge A_{unsafe}(s_{|unsafe})$$

# Example


CarRace
Fuel
Speed
Laps 3/5
Fuel
Speed
Number of Laps
Position
BOX

# Example

```
1   time2 = System.currentTimeMillis();
2   double delta = speed * (time2 – time);
3   x = x + delta * cos(direction);
4   y = y + delta * sin(direction);
5   Server.sendPosition(x,y);
6   if (track.isInBox(x, y)){
7           gas = maxGas;
8           lastFuel = time2;
9   }
10  else {
11          gas = maxGas - (int) (time2-lastFuel);
12          if (gas < 0) {
13                  gas = 0;
14                  if (speed > maxSpeed /10)
15                          speed = maxSpeed /10;
16                  else if (speed < minSpeed/10)
17                          speed = minSpeed/10;
            }
    }
18  time = time2;
```



# Program slice

```
1   time2 = System.currentTimeMillis();
2   double delta = speed * (time2 – time);
3   x = x + delta * cos(direction);
4   y = y + delta * sin(direction);
5   Server.sendPosition(x,y);
6   if (track.isInBox(x, y)){
7           gas = maxGas;
8           lastFuel = time2;
9   }
10  else {
11          gas = maxGas - (int) (time2-lastFuel);
12          if (gas < 0) {
13                  gas = 0;
14                  if (speed > maxSpeed /10)
15                          speed = maxSpeed /10;
16                  else if (speed < minSpeed/10)
17                          speed = minSpeed/10;
            }
    }
18  time = time2;
```

# Program slice

```
1   time2 = System.currentTimeMillis();
2   double delta = speed * (time2 – time);
3   x = x + delta * cos(direction);
4   y = y + delta * sin(direction);
5   Server.sendPosition(x,y);
6   if (track.isInBox(x, y)){
7           gas = maxGas;
8           lastFuel = time2;
9   }
10  else {
11          gas = maxGas - (int) (time2-lastFuel);
12          if (gas < 0){
13                  gas = 0;
14                  if (speed > maxSpeed /10)
15                          speed = maxSpeed /10;
16                  else if (speed < minSpeed/10)
17                          speed = minSpeed/10;
            }
    }
18  time = time2;
```

slice(speed, 18) =
{1, 2, 3, 4, 6, 11, 12, 14, 15, 16, 17}

# Barrier slicing

```
1   time2 = System.currentTimeMillis();
2   double delta = speed * (time2 – time);
3   x = x + delta * cos(direction);
4   y = y + delta * sin(direction);
5   Server.sendPosition(x,y);
6   if (track.isInBox(x, y)){
7           gas = maxGas;
8           lastFuel = time2;
9   }
10  else {
11          gas = maxGas - (int) (time2-lastFuel);
12          if (gas < 0) {
13                  gas = 0;
14                  if (speed > maxSpeed /10)
15                          speed = maxSpeed /10;
16                  else if (speed < minSpeed/10)
17                          speed = minSpeed/10;
            }
    }
18  time = time2;
```

Barriers

speed

gas

y

x

# Barrier slicing

```
1   time2 = System.currentTimeMillis();
2   double delta = speed * (time2 – time);
3   x = x + delta * cos(direction);
4   y = y + delta * sin(direction);
5   Server.sendPosition(x,y);
6   if (track.isInBox(x, y)){
7           gas = maxGas;
8           lastFuel = time2;
9   }
10  else {
11          gas = maxGas - (int) (time2-lastFuel);
12          if (gas < 0){
13                  gas = 0;
14                  if (speed > maxSpeed /10)
15                          speed = maxSpeed /10;
16                  else if (speed < minSpeed/10)
17                          speed = minSpeed/10;
            }
    }
18  time = time2;
```

slice(speed, 18) =
{1, 2, 6, 11, 12, 14, 15, 16, 17}

# Client transformation 1

```
time2 = System.currentTimeMillis();
double delta = speed * (time2 – time);
x = x + delta * cos(direction);
y = y + delta * sin(direction);
Server.sendPosition(x,y);
if (track.isInBox(x, y)){
      gas = maxGas;
      lastFuel = time2;
      }
else {
      gas = maxGas - (int) (time2-lastFuel);
      if (gas < 0) {
            gas = 0;
            if (speed > maxSpeed /10)
                  speed = maxSpeed /10;
            else if (speed < minSpeed/10)
                  speed = minSpeed/10;
      }
}
time = time2;
```

```
time2 = System.currentTimeMillis();
double delta = speed * (time2 – time);
x = x + delta * cos(direction);
y = y + delta * sin(direction);
Server.sendPosition(x,y);
if (track.isInBox(x, y)){
      sync( );
      lastFuel = time2;
      }
else {
      sync( );
      if (gas < 0) {
            sync( );
            if (speed > maxSpeed /10)
                  sync( );
            else if (speed < minSpeed/10)
                  sync( );
      }
}
time = time2;
```

# Client transformation 2

```
time2 = System.currentTimeMillis();
double delta = speed * (time2 – time);
x = x + delta * cos(direction);
y = y + delta * sin(direction);
Server.sendPosition(x,y);
if (track.isInBox(x, y)){
      sync( );
      lastFuel = time2;
      }
else {
      sync( );
      if (gas < 0) {
            sync( );
            if (speed > maxSpeed /10)
                  sync( );
            else if (speed < minSpeed/10)
                  sync( );
      }
}
time = time2;
```

```
time2 = System.currentTimeMillis();
double delta = speed * (time2 – time);
x = x + delta * cos(direction);
y = y + delta * sin(direction);
Server.sendPosition(x,y);
if (track.isInBox(x, y)){
      sync( );
      lastFuel = time2;
      }
else {
      sync( );
      if (ask("gas") < 0) {
            sync( );
            if (ask("speed") > maxSpeed /10)
                  sync( );
            else if (ask("speed") <minSpeed/10)
                  sync( );
      }
}
time = time2;
```

# Server transformation

```
time2 = System.currentTimeMillis();
double delta = speed * (time2 – time);
client.receivePosition(x,y);
if (track.isInBox(x, y)){
      gas = maxGas;
      lastFuel = time2;
}
else {
      gas = maxGas - (int) (time2-lastFuel);
      if (gas < 0) {
            gas = 0;
            if (speed > maxSpeed /10)
                  speed = maxSpeed /10;
            else if (speed < minSpeed/10)
                  speed = minSpeed/10;
      }
}
time = time2;
```

```
time2 = System.currentTimeMillis();
double delta = speed * (time2 – time);
client.receivePosition(x,y);
if (A(x,y) == false )
      exit( "Tampering detected");
if (track.isInBox(x, y)){
      gas = maxGas;
      sync( );
      lastFuel = time2;
}
else {
      gas = maxGas - (int) (time2-lastFuel);
      sync( );
      if (gas < 0) {
            gas = 0;
            sync( );
            if (speed > maxSpeed /10) {
                  speed = maxSpeed /10;
                  sync( ); }
            else if (speed < minSpeed/10) {
                  speed = minSpeed/10;
                  sync( ); }
      }
}
time = time2;
```

# Optimizations:

```
time2 = System.currentTimeMillis();
double delta = speed * (time2 – time);
x = x + delta * cos(direction);
y = y + delta * sin(direction);
Server.sendPosition(x,y);
if (track.isInBox(x, y)){
      sync( );
      lastFuel = time2;
      }
else {
      sync( );
      if (ask("gas") < 0) {
            sync( );
            if (ask("speed") > maxSpeed /10)
                  sync( );
            else if (ask("speed") <minSpeed/10)
                  sync( );
      }
}
time = time2;
```

# Preliminary results

- CarRace game:
  - We moved the barrier slice on the server
  - Each time the client needs a value computed on the server, it asks for it from the server (communication overhead, delay).

| Original client | Slice | Barrier slice |
|---|---|---|
| 858 | 185 | 120 (-65%) |
| | 22% | 14% (-35%) |

| | Regular messages | Trust messaged | Increase |
|---|---|---|---|
| **Sent** | 1174 | 5910 | 5.03 |
| **Received** | 1172 | 5910 | 5.04 |

# Open issues

- Does the approach scale on a real size application?
  - Communication overhead.
  - Server overhead.
  - Identification of the security sensitive sub-state (s).
  - Identification of the already-protected sensitive sub-state ($s_{|safe}$).
  - Integration with other techniques.

# Optimizations:

```
time2 = System.currentTimeMillis();
double delta = speed * (time2 – time);
x = x + delta * cos(direction);
y = y + delta * sin(direction);
Server.sendPosition(x,y);
if (track.isInBox(x, y)){
      sync( );
      lastFuel = time2;
      }
else {
      sync( );
      if (ask("gas") < 0) {
            sync( );
            if (ask("speed") > maxSpeed /10)
                  sync( );
            else if (ask("speed") <minSpeed/10)
                  sync( );
      }
}
time = time2;
```