Karlsruhe Institute of Technology

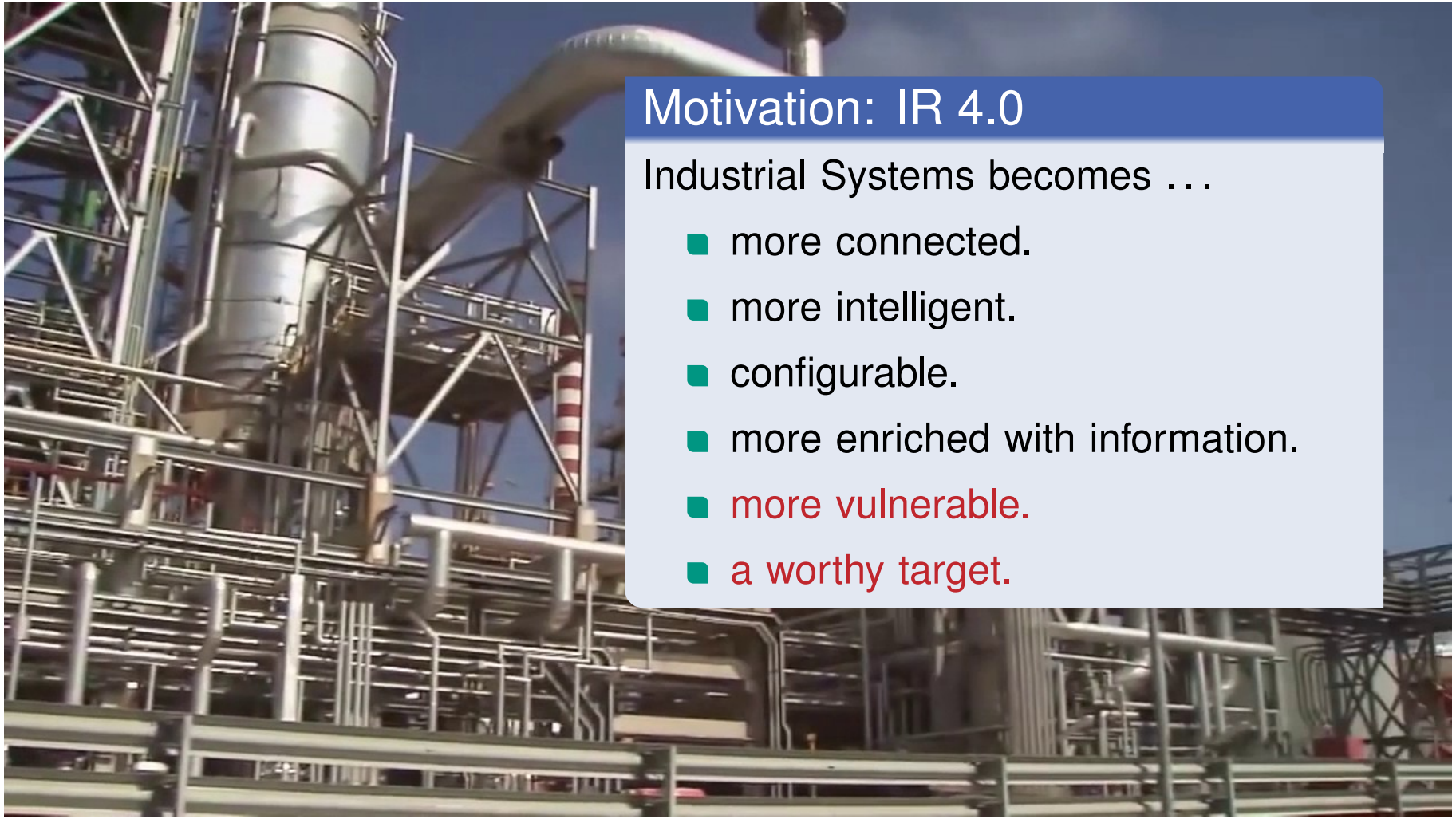
KASTEL Industry 4.0 Demonstrator

Provably Forgetting Information in PLC software

Alexander Weigl | 10. Oct. 2019

INSTITUTE FOR THEORETICAL INFORMATICS – APPLICATION-ORIENTED FORMAL VERIFICATION

Orientation & Story

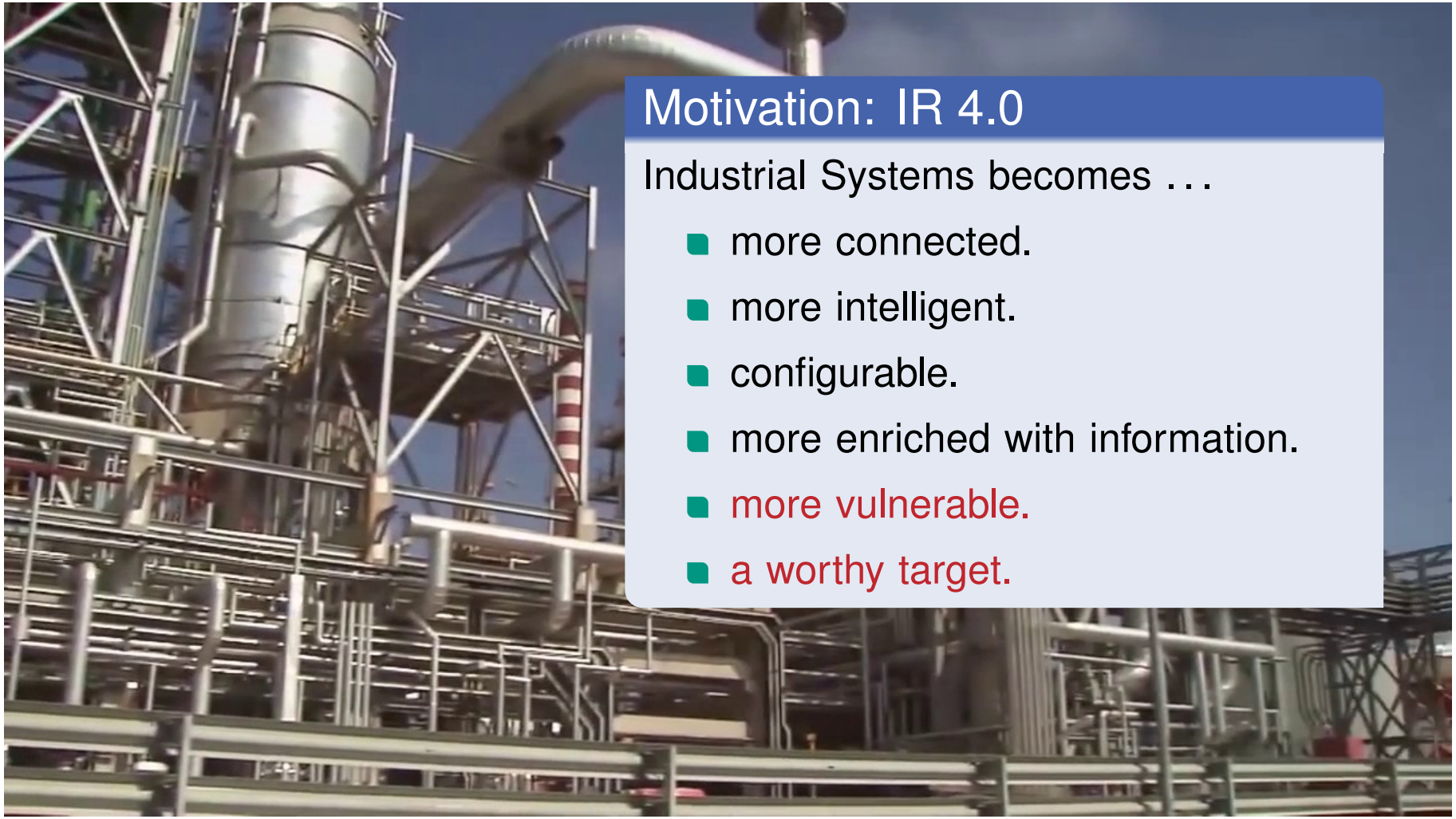


Motivation: IR 4.0

Industrial Systems becomes ...

- more connected.
- more intelligent.
- configurable.
- more enriched with information.
- more vulnerable.
- a worthy target.

Orientation & Story

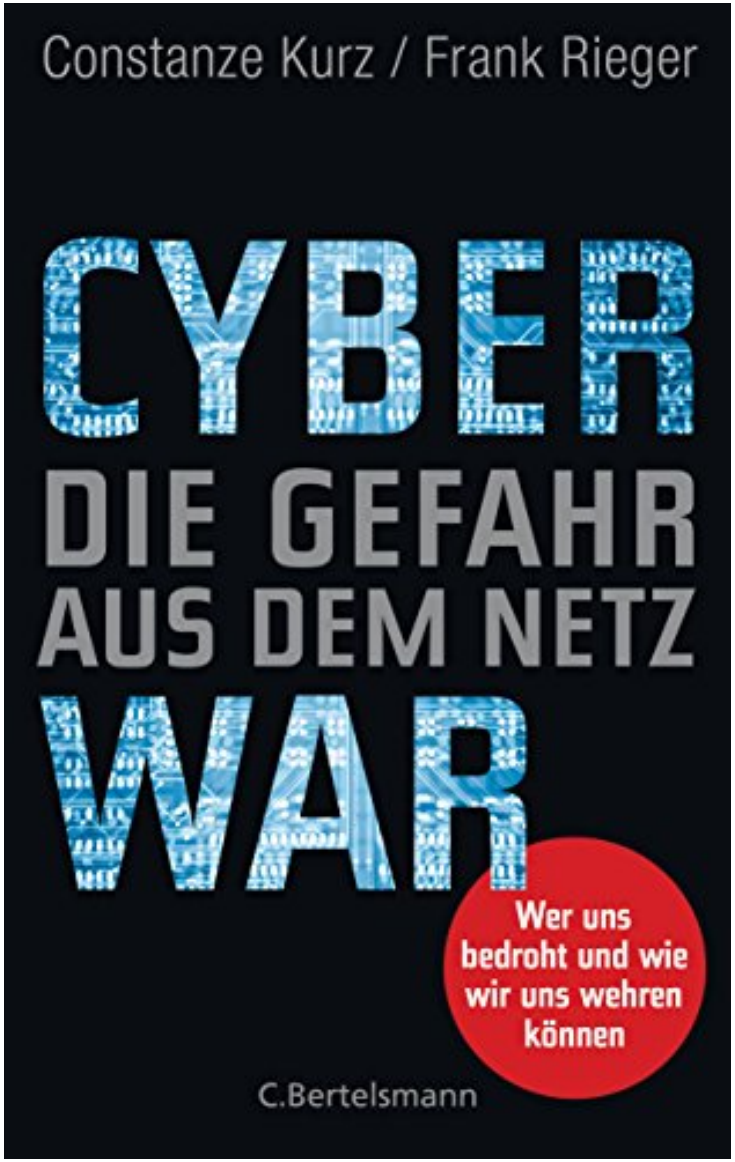


Motivation: IR 4.0

Industrial Systems becomes ...

- more connected.
- more intelligent.
- configurable.
- more enriched with information.
- more vulnerable.
- a worthy target.

Orientation & Story



Constanze Kurz and Frank Rieger
Cyberwar – Die Gefahr aus dem Netz

also in *LNP272: Alles zerfragen*

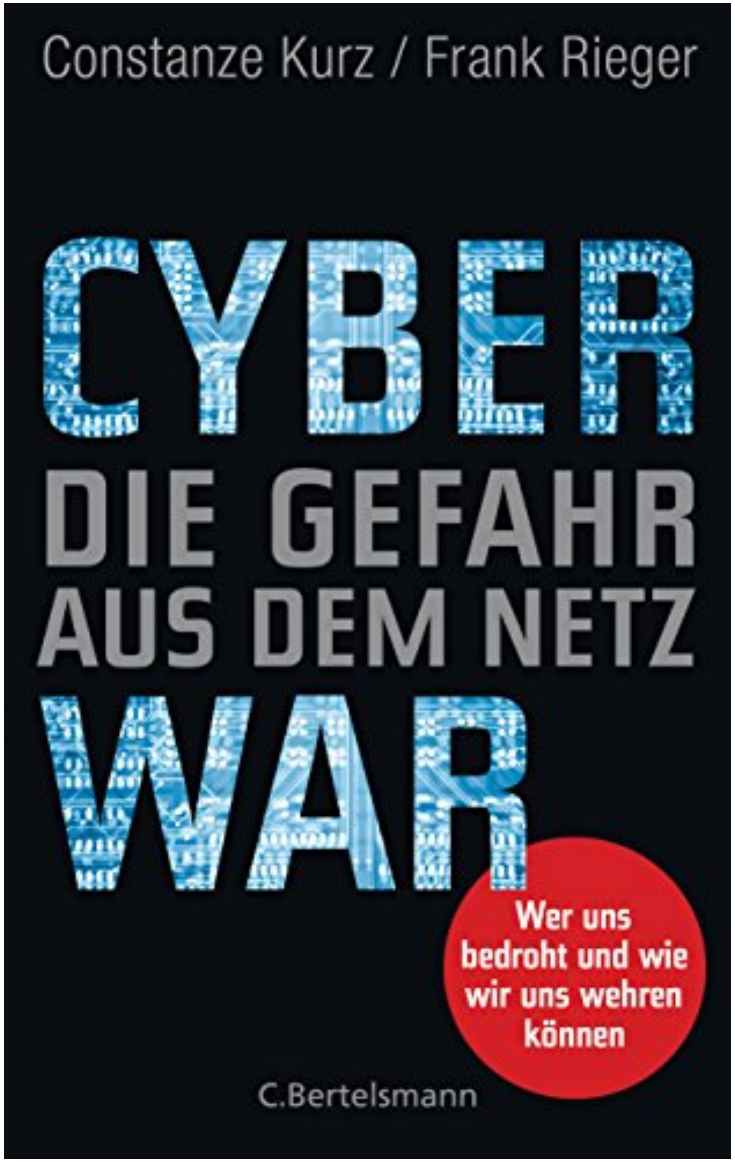
Business Secrets are

confidential information of a company, and protected by **law**.
Protection requires efforts by the owning company to protect their data following the state of the art.

KASTEL

Demonstrator is part of KASTEL SVI (AP 4.6)

Orientation & Story



Constanze Kurz and Frank Rieger
Cyberwar – Die Gefahr aus dem Netz

also in *LNP272: Alles zerfragen*

Business Secrets are

confidential information of a company, and protected by **law**.
Protection requires efforts by the owning company to protect their data following the state of the art.

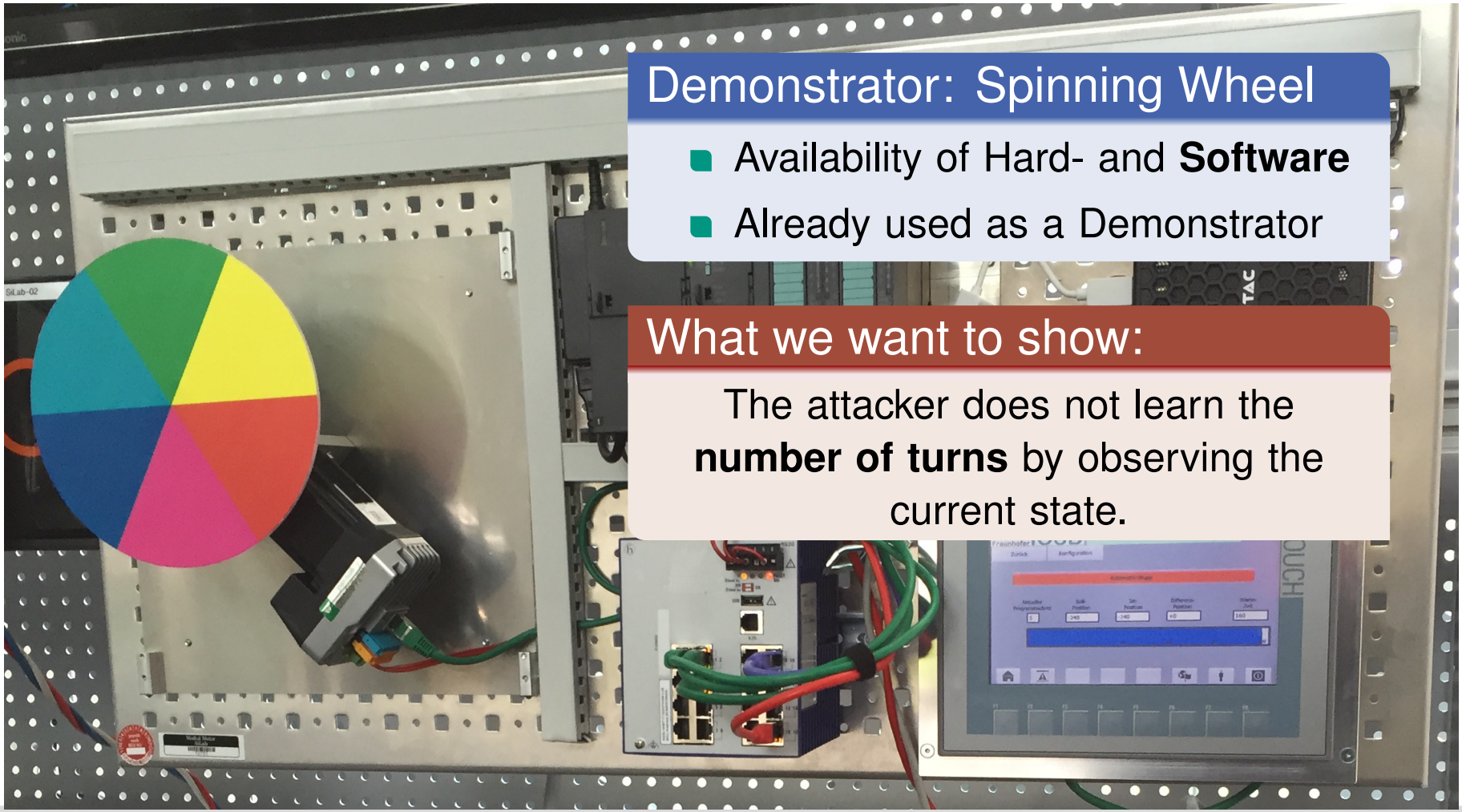
KASTEL

Demonstrator is part of KASTEL SVI (AP 4.6)

The KASTEL Demonstrator

What we demonstrate?

An approach to ensure that no Business Secrets are stored.



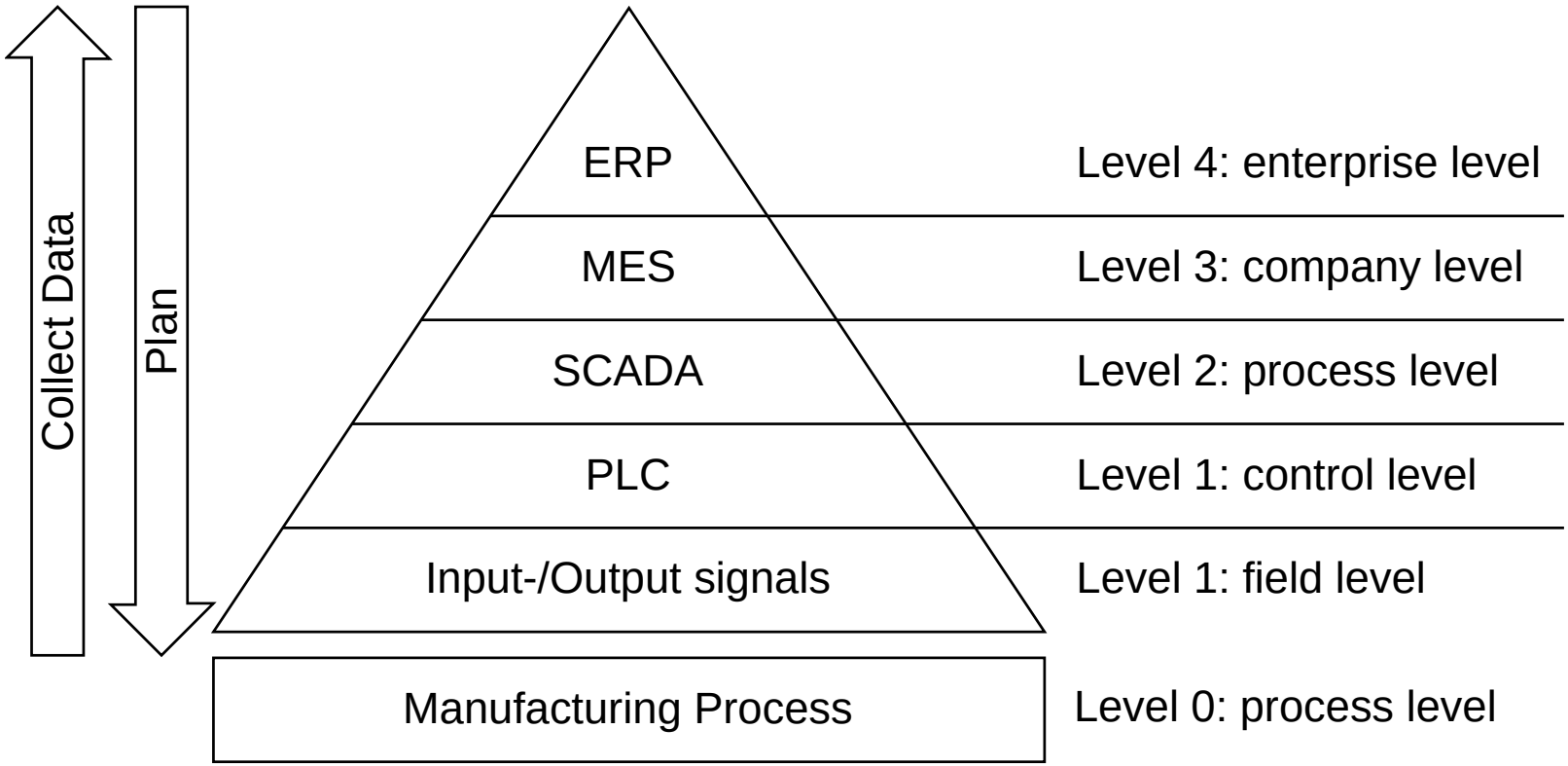
Demonstrator: Spinning Wheel

- Availability of Hard- and **Software**
- Already used as a Demonstrator

What we want to show:

The attacker does not learn the **number of turns** by observing the current state.

System & Attacker Model



Attacker's Environment

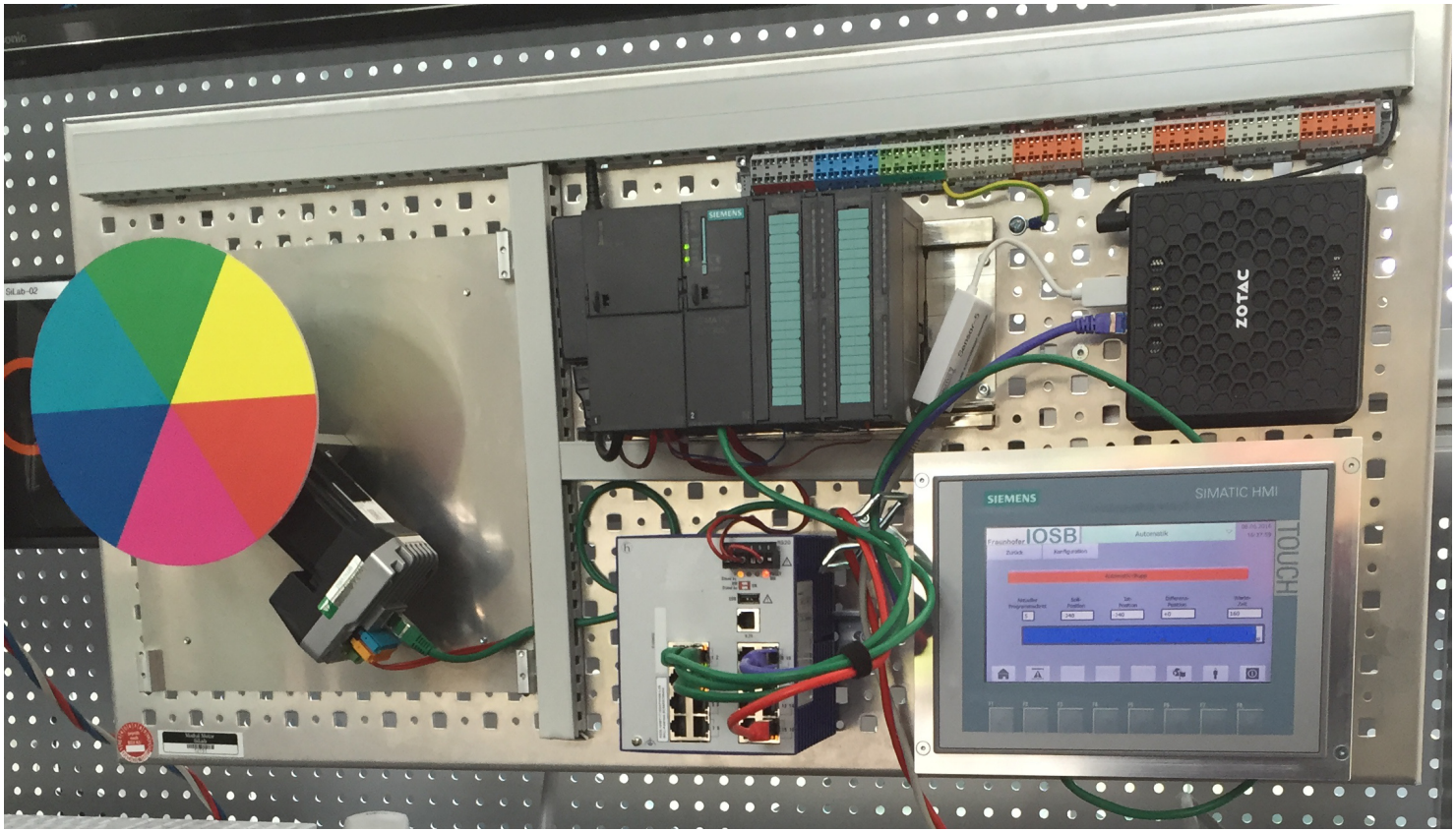
- Focus on the PLC system
- Attacker can observe only one system state

Overview

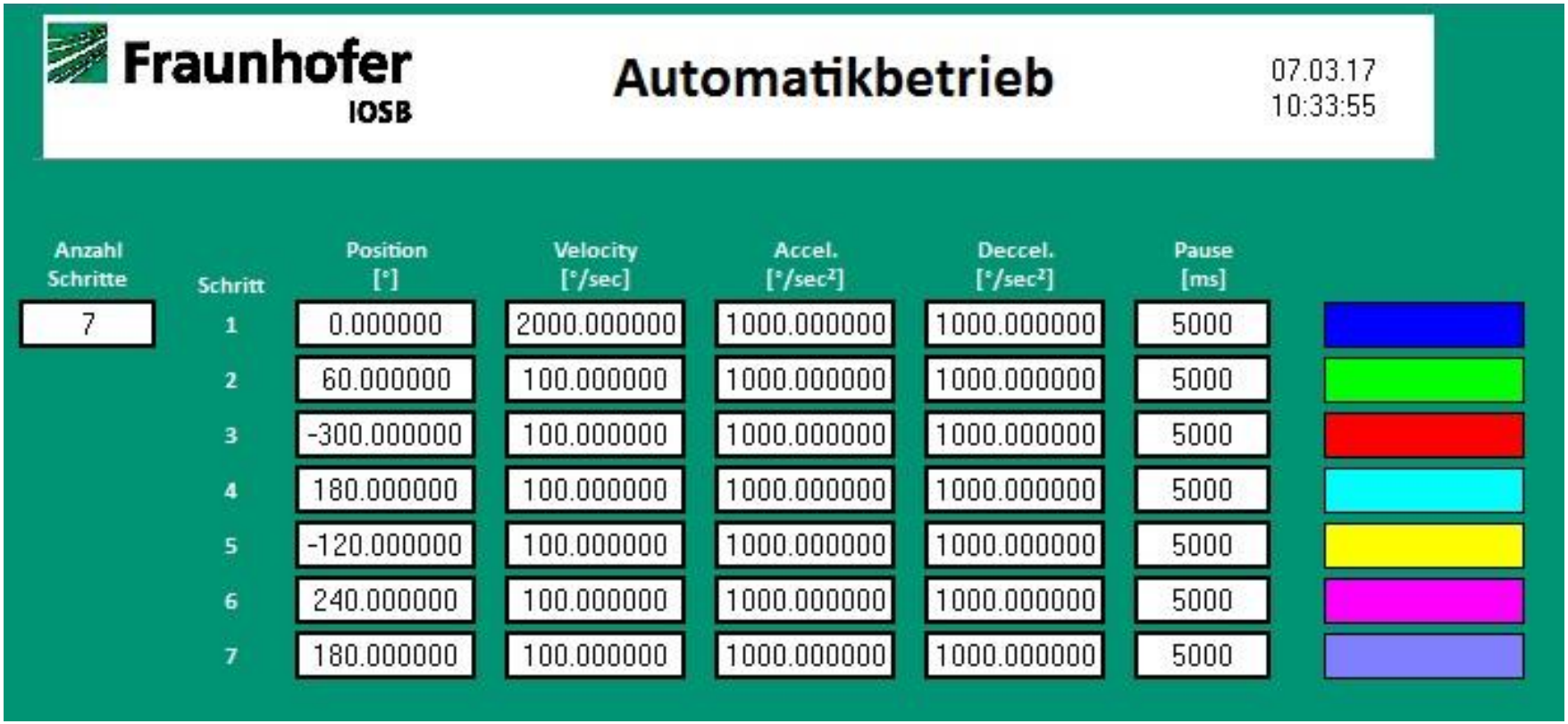
1. The Software
   - Functionality
   - Software Architecture
   - Preparation for Verification
2. The Verification
   - Information Flow
   - Forgetting Information
   - Results
   - Discussion: Validity
3. Closing Remarks
   - Quantification
   - Conclusion

The Software

Operator view



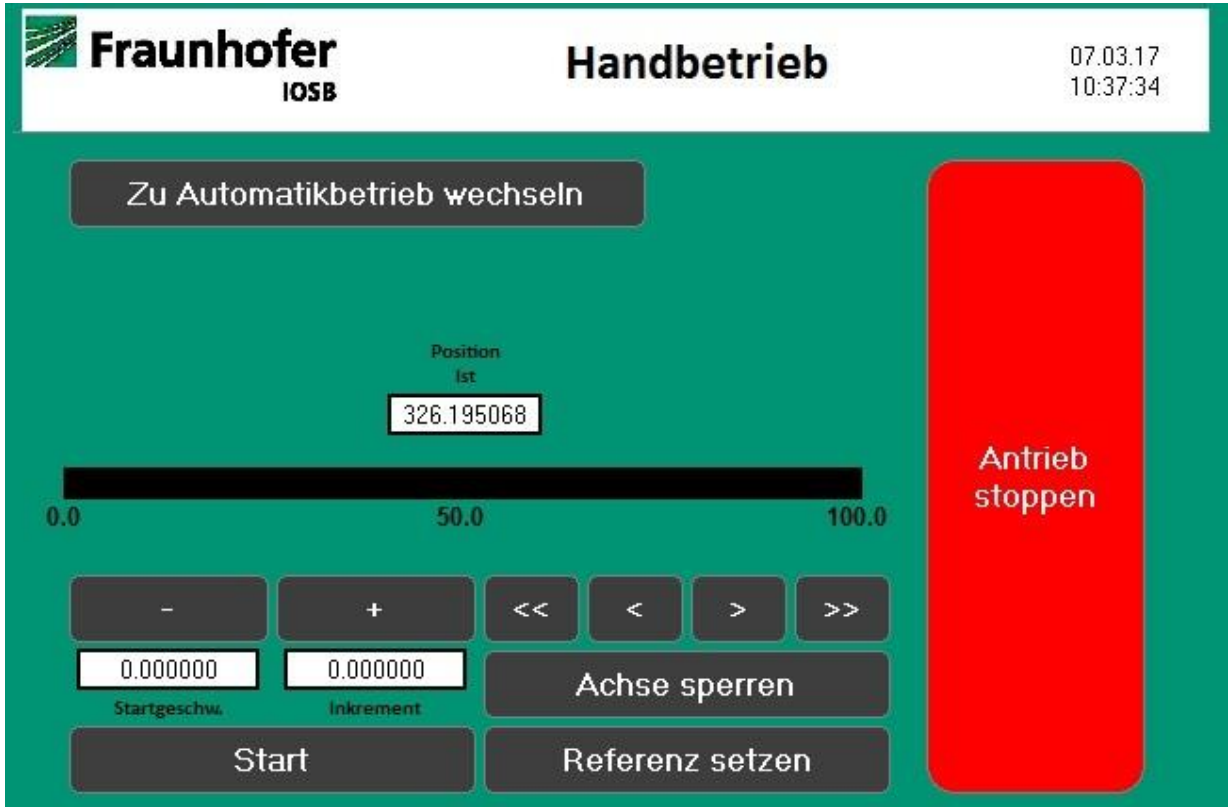
Operator view



Automatic Mode

- PLC drives to *user-defined* segments sequentially
- A segment consists of position, velocity, accel-/decelaration, break time
- Sequence can be repeatedly executed

Operator view

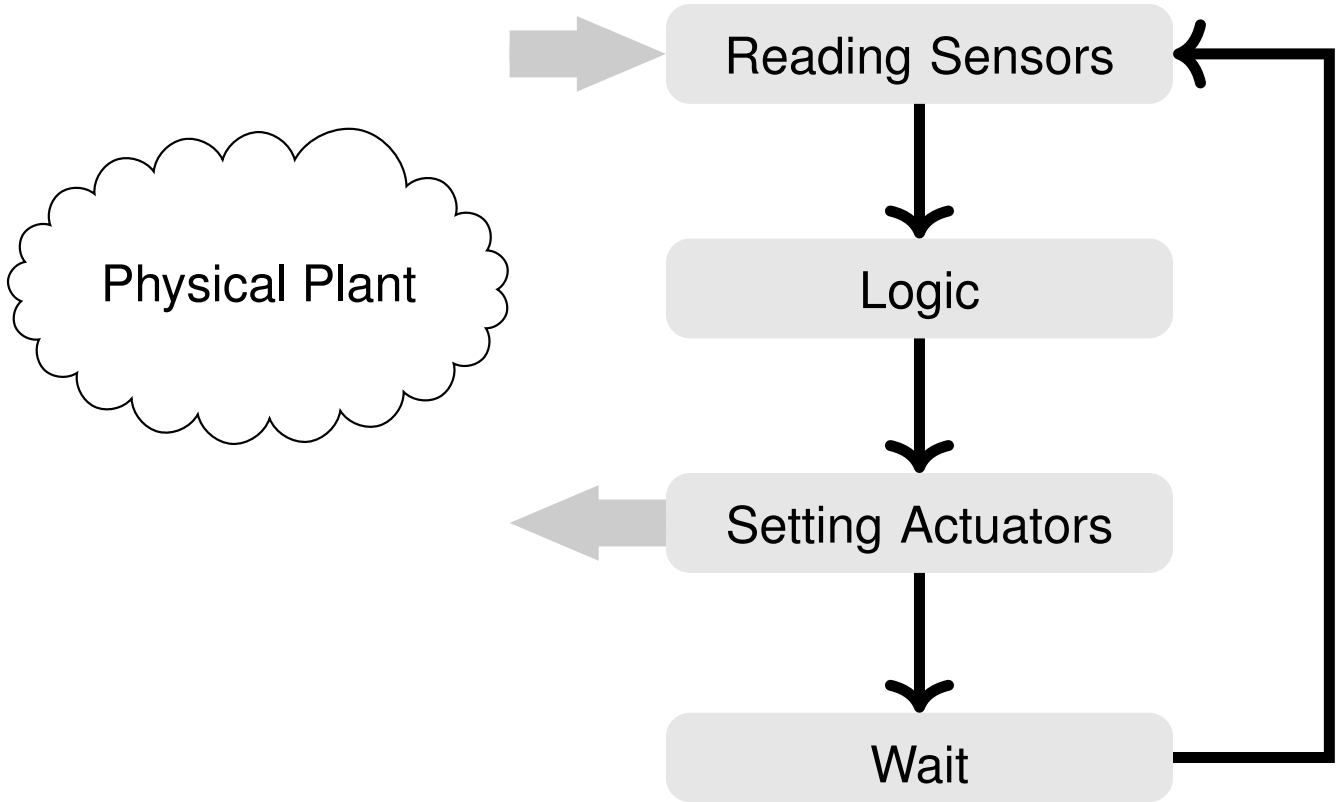


Manual Mode

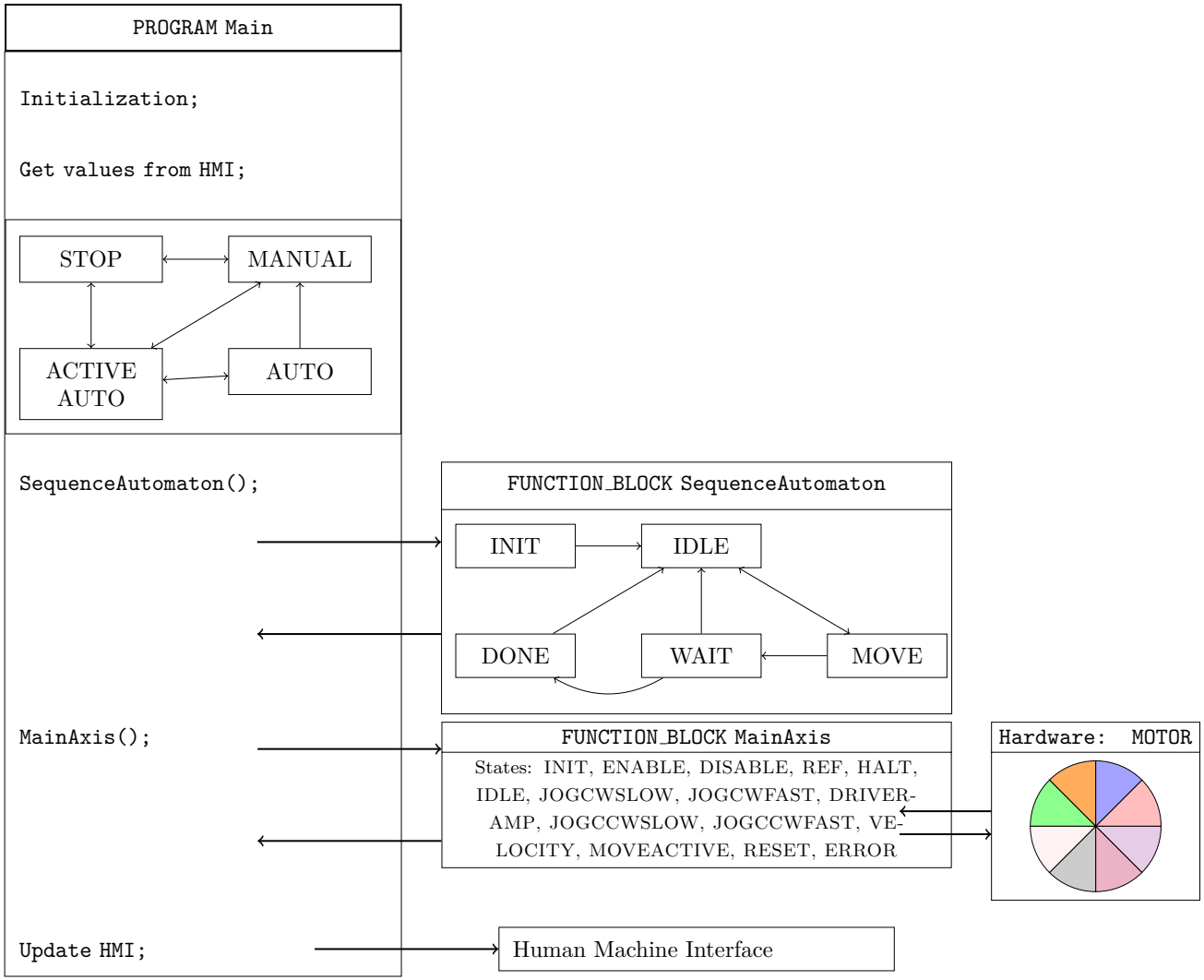
- Operator can manually control velocity, and
- set the reference position

Reactive Software

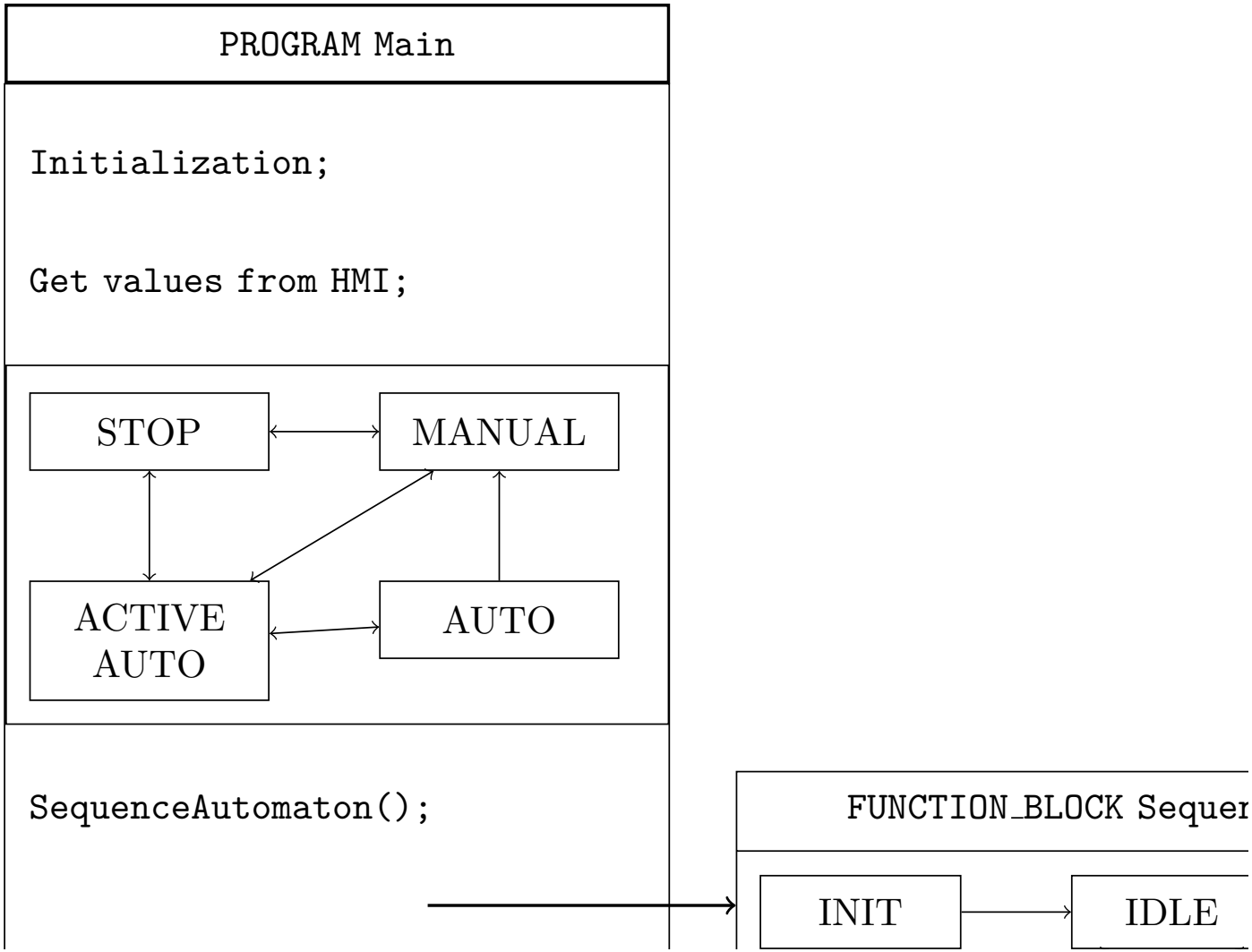
- Executed every n ms
- Feedback loop
- For verification, we focus on **Logic** component



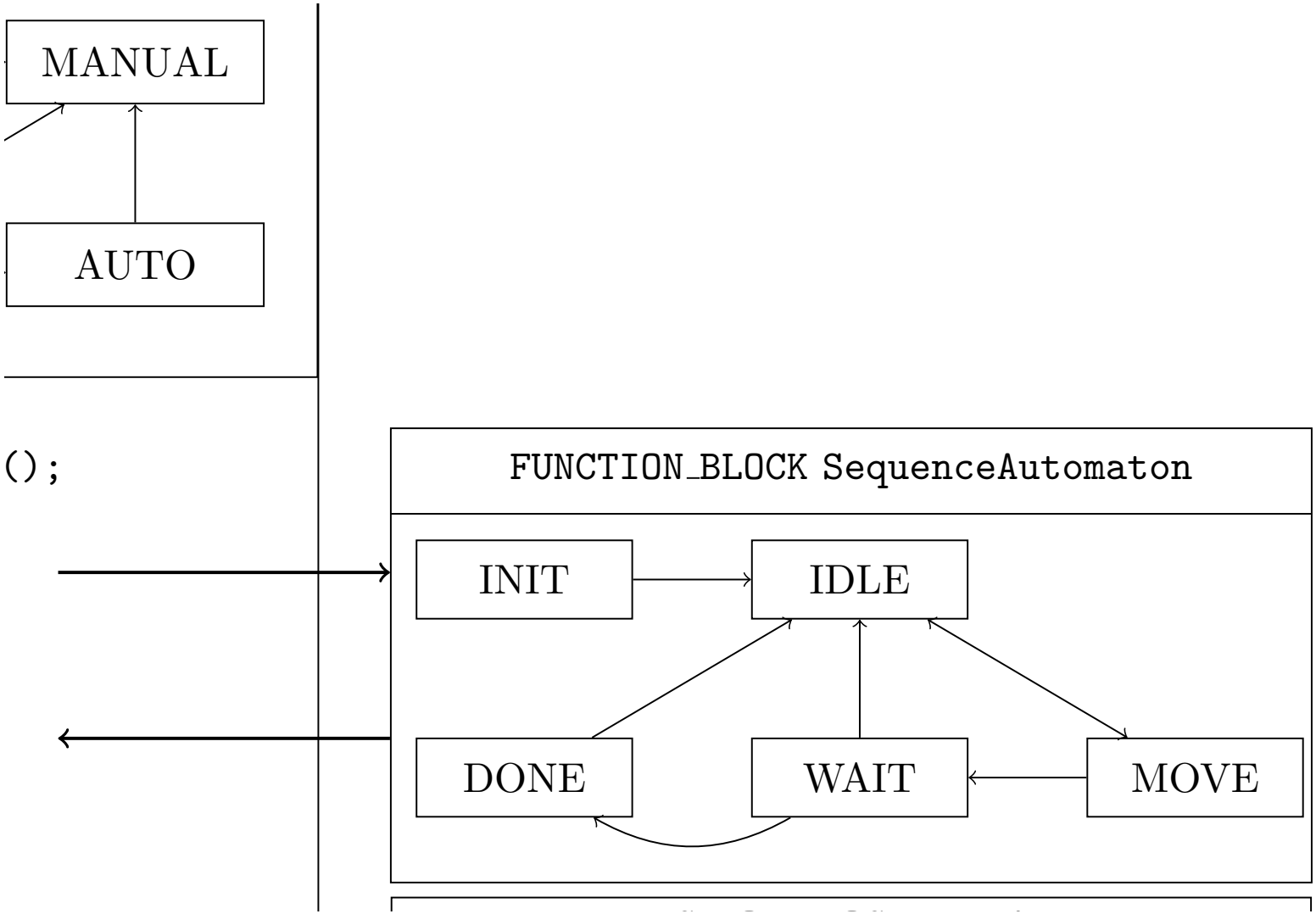
Software Architecture



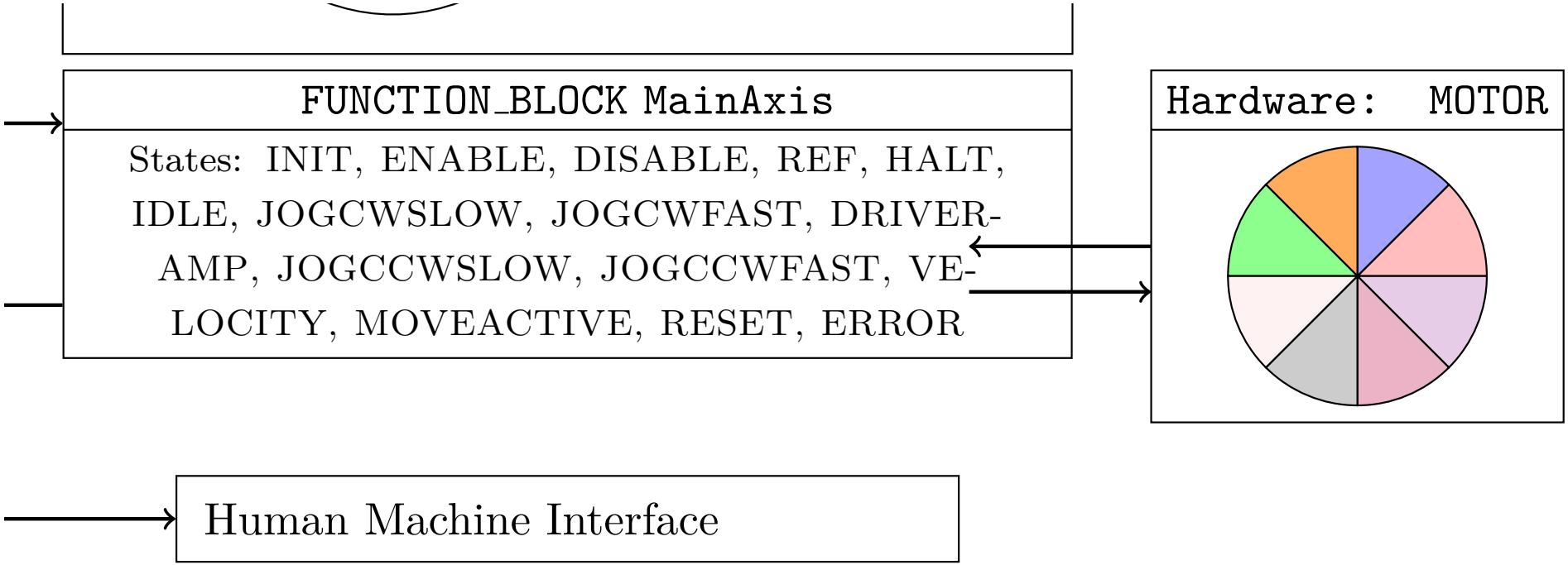
Software Architecture



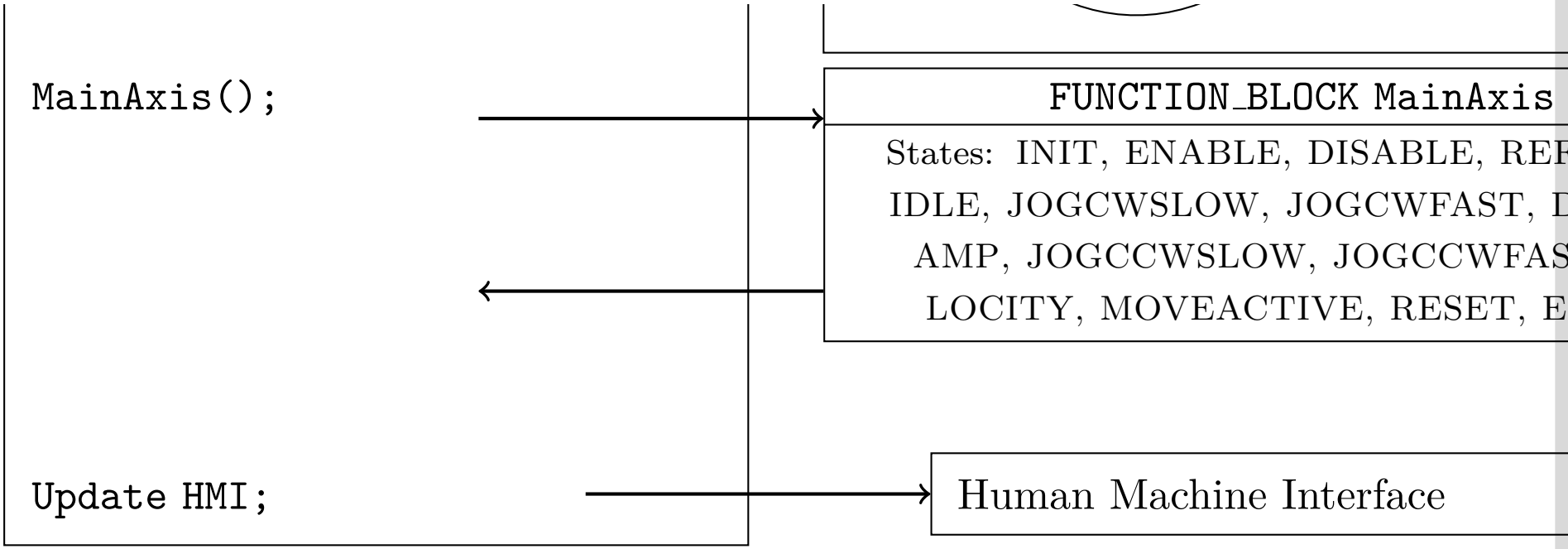
Software Architecture



Software Architecture



Software Architecture



Preparation for Verification

Software not directly usable

- focus on `MainAxis`
- demote floating-point to integers
- reduce state, remove assignment to HMI variables

Are these abstractions valid?

Verification Pipeline

PLC Program to be Verified

- 421 LoC in Structured Text
- 32 states variables, 52 input variables
- 566 bits large (270 bits input, 296 bits state)

Preparation for Verification

Software not directly usable

- focus on `MainAxis`
- demote floating-point to integers
- reduce state, remove assignment to HMI variables

Are these abstractions valid?

Verification Pipeline

PLC Program to be Verified

- 421 LoC in Structured Text
- 32 states variables, 52 input variables
- 566 bits large (270 bits input, 296 bits state)

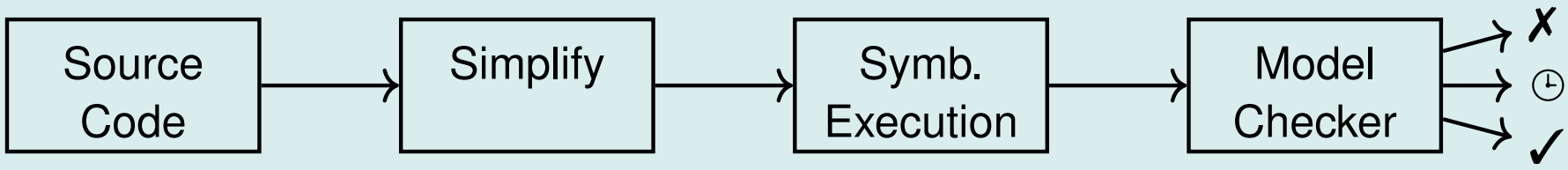
Preparation for Verification

Software not directly usable

- focus on `MainAxis`
- demote floating-point to integers
- reduce state, remove assignment to HMI variables

Are these abstractions valid?

Verification Pipeline



PLC Program to be Verified

- 421 LoC in Structured Text
- 32 states variables, 52 input variables
- 566 bits large (270 bits input, 296 bits state)

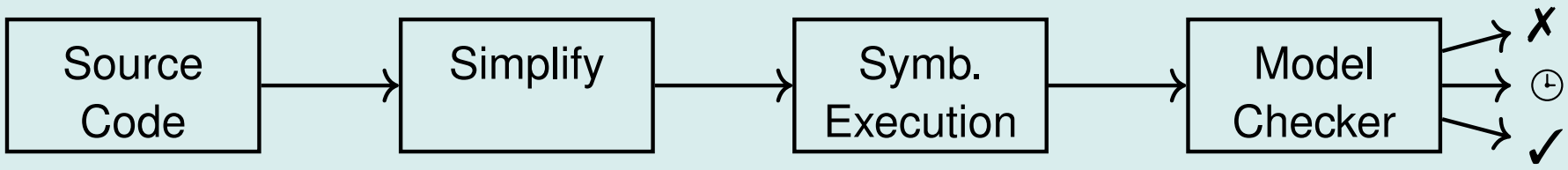
Preparation for Verification

Software not directly usable

- focus on `MainAxis`
- demote floating-point to integers
- reduce state, remove assignment to HMI variables

Are these abstractions valid?

Verification Pipeline



PLC Program to be Verified

- 421 LoC in Structured Text
- 32 states variables, 52 input variables
- 566 bits large (270 bits input, 296 bits state)

The Verification

Information Flow

What we want to show:

The attacker does not learn the **number of turns** since the start of the PLC by observing the current state.

The attacker does not learn the number of turns by observing **one state** σ_{t_0}:

$$\#Turns(t_0) := \left\lfloor \frac{1}{360} \int_0^{t_0} v(t)\, dt \right\rfloor$$

$$\text{Prob}(\#Turns) = \text{Prob}(\#Turns \mid \sigma_{t_0})$$

$v(t)$ – Angular Speed ($\frac{deg}{s}$)

Classical Information Flow

- Property: No influence of $v(t)$ on the state.
- . . . Non-interference is too strong: Velocity is stored internally!
- . . . of course sensors values have influence
- . . . but #Turns is not stored.

Information Flow

What we want to show:

The attacker does not learn the **number of turns** since the start of the PLC by observing the current state.

The attacker does not learn the number of turns by observing **one state** σ_{t_0}:

$$\#Turns(t_0) := \left\lfloor \frac{1}{360} \int_0^{t_0} v(t)\, dt \right\rfloor$$

$$\text{Prob}(\#Turns) = \text{Prob}(\#Turns \mid \sigma_{t_0})$$

$v(t)$ – Angular Speed ($\frac{deg}{s}$)

Classical Information Flow

- Property: No influence of $v(t)$ on the state.
- . . . Non-interference is too strong: Velocity is stored internally!
- . . . of course sensors values have influence
- . . . but #Turns is not stored.

Information Flow

What we want to show:

The attacker does not learn the **number of turns** since the start of the PLC by observing the current state.

The attacker does not learn the number of turns by observing **one state** σ_{t_0}:

$$\#Turns(t_0) := \left\lfloor \frac{1}{360} \int_0^{t_0} v(t)\, dt \right\rfloor$$

$$\text{Prob}(\#Turns) = \text{Prob}(\#Turns \mid \sigma_{t_0})$$

$v(t)$ – Angular Speed ($\frac{deg}{s}$)

Classical Information Flow

- Property: No influence of $v(t)$ on the state.
- . . . Non-interference is too strong: Velocity is stored internally!
- . . . of course sensors values have influence
- . . . but #Turns is not stored.

Information Flow

What we want to show:

The attacker does not learn the **number of turns** since the start of the PLC by observing the current state.

The attacker does not learn the number of turns by observing **one state** σ_{t_0}:

$$\#Turns(t_0) := \left\lfloor \frac{1}{360} \int_0^{t_0} v(t)\, dt \right\rfloor$$

$$\text{Prob}(\#Turns) = \text{Prob}(\#Turns \mid \sigma_{t_0})$$

$v(t)$ – Angular Speed $(\frac{deg}{s})$

Classical Information Flow

- Property: No influence of $v(t)$ on the state.
- . . . Non-interference is too strong: Velocity is stored internally!
- . . . of course sensors values have influence
- . . . but #Turns is not stored.

Forgetting Information

Idea

- Relaxing the information flow
- Allowing the system to react to current sensor values

. . . but forget old information

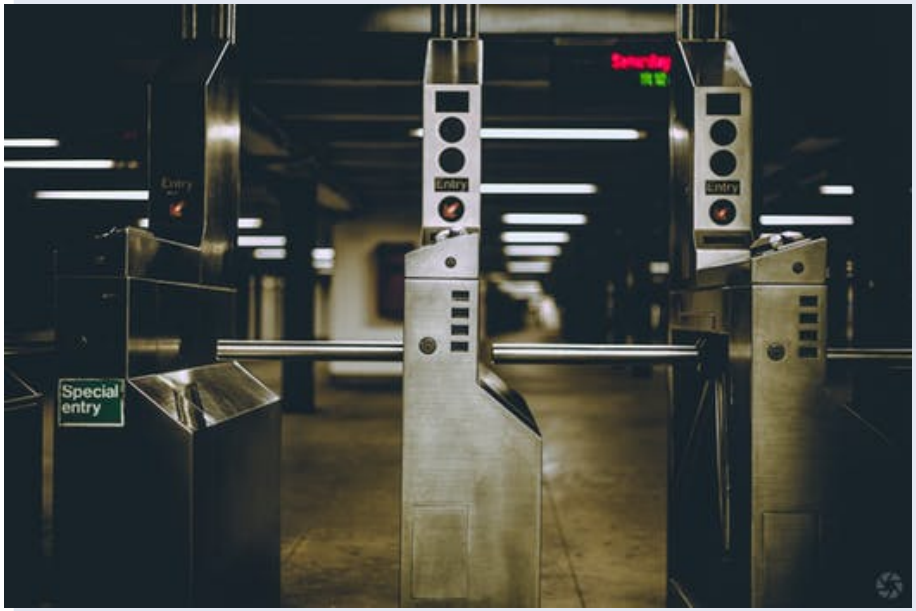
Forgetting Information

Idea

- Relaxing the information flow
- Allowing the system to react to current sensor values

. . . but forget old information

Example: Baffle Gate



- Granting access based on permission
- But does not store amount of passed

Forgetting Information

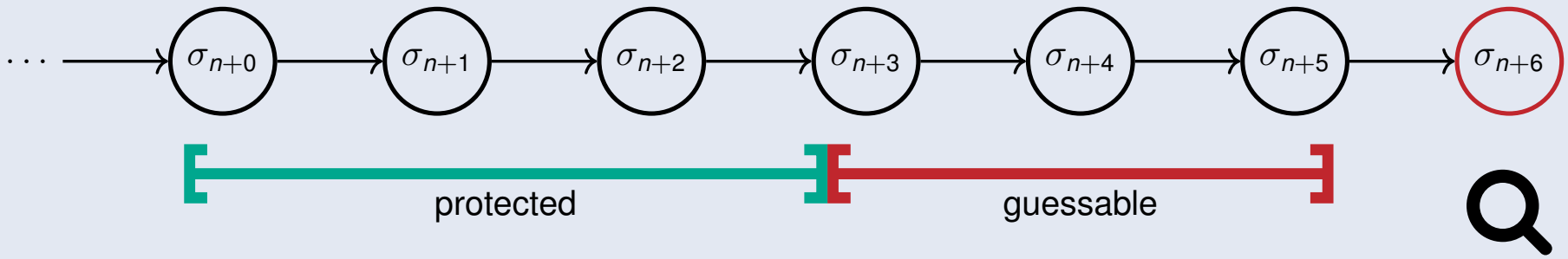
Idea

- Relaxing the information flow
- Allowing the system to react to current sensor values

  . . . but forget old information

Privacy-preservation by forgetting

System is allowed to store secret data of m last steps.



Forgetting Information in Relational Test Tables

#	ASSUME			ASSERT	⏲
	S	I_L	I_H	S	
0	$=$	$=$	—	—	1
1	—	$=$	—	—	—
2	—	$=$	$=$	—	k
3	—	$=$	$=$	$=$	ω

We distinguish between

- state variables ($|S| = 32$)
- uncritical sensor variables ($|I_L| = 51$), and
- protected sensor variable ($|I_H| = 1$, angular velocity).

Syntax

- "—" expresses "DON'T CARE"
- "=" expresses equality in columns variables
- k is the allowed lookbehind

Forgetting Information in Relational Test Tables

#	ASSUME S	ASSUME I_L	ASSUME I_H	ASSERT S	⏲
0	=	=	—	—	1
1	—	=	—	—	—
2	—	=	=	—	k
3	—	=	=	=	ω

We distinguish between

- state variables ($|S| = 32$)
- uncritical sensor variables ($|I_L| = 51$), and
- protected sensor variable ($|I_H| = 1$, angular velocity).

Syntax

- "—" expresses "DON'T CARE"
- "=" expresses equality in columns variables
- k is the allowed lookbehind

Forgetting Information in Relational Test Tables

#	ASSUME S	ASSUME I_L	ASSUME I_H	ASSERT S	⏲
0	=	=	—	—	1
1	—	=	—	—	—
2	—	=	=	—	k
3	—	=	=	=	ω

We distinguish between

- state variables ($|S| = 32$)
- uncritical sensor variables ($|I_L| = 51$), and
- protected sensor variable ($|I_H| = 1$, angular velocity).

Syntax

- "—" expresses "DON'T CARE"
- "=" expresses equality in columns variables
- k is the allowed lookbehind

Forgetting Information in Relational Test Tables

#	ASSUME S	ASSUME I_L	ASSUME I_H	ASSERT S	$\odot$
0	$=$	$=$	—	—	1
1	—	$=$	—	—	—
2	—	$=$	$=$	—	k
3	—	$=$	$=$	$=$	ω

Explanation

For all possible two runs of the systems, starting in

- **arbitrary, but equal, states** and equal uncritical input I_L,
- then **injecting different secrets**,
- after **waiting** k **cycles**
- the **states have to be equal**

Forgetting Information in Relational Test Tables

#	ASSUME S	ASSUME I_L	ASSUME I_H	ASSERT S	$\odot$
0	$=$	$=$	—	—	1
1	—	$=$	—	—	—
2	—	$=$	$=$	—	k
3	—	$=$	$=$	$=$	ω

Explanation

For all possible two runs of the systems, starting in

- **arbitrary, but equal, states** and equal uncritical input I_L,
- then **injecting different secrets**,
- after **waiting k cycles**
- the **states have to be equal**

Forgetting Information in Relational Test Tables

#	ASSUME S	ASSUME I_L	ASSUME I_H	ASSERT S	$\odot$
0	$=$	$=$	—	—	1
1	—	$=$	—	—	—
2	—	$=$	$=$	—	k
3	—	$=$	$=$	$=$	ω

Explanation

For all possible two runs of the systems, starting in

- **arbitrary, but equal, states** and equal uncritical input I_L,
- then **injecting different secrets**,
- after **waiting k cycles**
- the **states have to be equal**

Forgetting Information in Relational Test Tables

#	ASSUME S	ASSUME I_L	ASSUME I_H	ASSERT S	🕒
0	$=$	$=$	—	—	1
1	—	$=$	—	—	—
2	—	$=$	$=$	—	k
3	—	$=$	$=$	$=$	ω

Explanation

For all possible two runs of the systems, starting in

- **arbitrary, but equal, states** and equal uncritical input I_L,
- then **injecting different secrets**,
- after **waiting k cycles**
- the **states have to be equal**

Results

The system does not adhere to information forgetting.
for $k = 2, 5, 7, 10$

Analysation of the counterexample

- *last* velocity is stored internally
- but not *last* velocity is not overwritten forcibly

If we do not consider the internal stored velocity, the system forgets the information.

Results

The system does not adhere to information forgetting.
for $k = 2, 5, 7, 10$

Analysation of the counterexample

- *last* velocity is stored internally
- but not *last* velocity is not overwritten forcibly

If we do not consider the internal stored velocity, the system forgets the information.

Results

The system does not adhere to information forgetting.
for $k = 2, 5, 7, 10$

Analysation of the counterexample

- *last* velocity is stored internally
- but not *last* velocity is not overwritten forcibly

If we do not consider the internal stored velocity, the system forgets the information.

Results

The system does not adhere to information forgetting.
for $k = 2, 5, 7, 10$

Analysation of the counterexample

- *last* velocity is stored internally
- but not *last* velocity is not overwritten forcibly

If we do not consider the internal stored velocity, the system forgets the information.

Results

The system does not adhere to information forgetting.
for $k = 2, 5, 7, 10$

Analysation of the counterexample

- *last* velocity is stored internally
- but not *last* velocity is not overwritten forcibly

If we do not consider the internal stored velocity, the system forgets the information.

Discussion: Validity

Why PLC level?

Protection on . . .

- PLC level is hard
- upper pyramid level easier and known

but also attacks on the sensor/actuator level happened

Single observable state

If an attacker sees a sequence of states, then

- the information of the sequence leak
- information that are k cycles past are still secret

Discussion: Validity

Why PLC level?

Protection on . . .

- PLC level is hard
- upper pyramid level easier and known

but also attacks on the sensor/actuator level happened

Single observable state

If an attacker sees a sequence of states, then

- the information of the sequence leak
- information that are k cycles past are still secret

Discussion: Validity

Only `MainAxis`

- `MainAxis` is the most critical
- HMI also reads the velocity from global state
- An attacker can get the complete user-defined program sequence

Program transformation

- Demoting floating point to integer is critical
  ... justification in each individual case
- Symb. Execution and other simplification are uncritical

Verification

- Starting in arbitrary equal states is an over-abstraction
- Spurious counterexample possible

Discussion: Validity

Only `MainAxis`

- `MainAxis` is the most critical
- HMI also reads the velocity from global state
- An attacker can get the complete user-defined program sequence

Program transformation

- Demoting floating point to integer is critical
  ... justification in each individual case
- Symb. Execution and other simplification are uncritical

Verification

- Starting in arbitrary equal states is an over-abstraction
- Spurious counterexample possible

Discussion: Validity

Only `MainAxis`

- `MainAxis` is the most critical
- HMI also reads the velocity from global state
- An attacker can get the complete user-defined program sequence

Program transformation

- Demoting floating point to integer is critical
  ... justification in each individual case
- Symb. Execution and other simplification are uncritical

Verification

- Starting in arbitrary equal states is an over-abstraction
- Spurious counterexample possible

Closing Remarks

Quantification

In view of KASTEL continuation:

Information Forgetting is a Quantification of Security

Quantifiations

A system that . . .

- forgets information faster
- forgets more information

is more secure.

In the view of *risk assessment*

A system, that forgets faster, decreases the costs when a data breach occurs.

Quantification

In view of KASTEL continuation:

Information Forgetting is a Quantification of Security

Quantifiations

A system that . . .

- forgets information faster
- forgets more information

is more secure.

In the view of *risk assessment*

A system, that forgets faster, decreases the costs when a data breach occurs.

Quantification

In view of KASTEL continuation:

Information Forgetting is a Quantification of Security

Quantifiations

A system that . . .

- forgets information faster
- forgets more information

is more secure.

In the view of *risk assessment*

A system, that forgets faster, decreases the costs when a data breach occurs.

Conclusion

Take away

- We can prove that systems forget information
- Forgetting information is a *quantitative* privacy property
  It does **not prevent attacks**, but the **loot is reduced**.
- Technical Report appears soon
- Verification software available:
  `https://github.com/verifaps/verifaps-lib`