ICS-SEA: Formally Modeling the Conflicting Design Constraints in ICS

Eyasu Getahun Chekole, Huaqun Guo

*Institute for Infocomm Research (I^2R), A*STAR, Singapore*

The 5th Industrial Control System Security Workshop (ICSS'19)
(Co-located with ACSAC'19)

San Juan, Puerto Rico, USA

10 December 2019

Outline

Outline

Industrial Control Systems (ICS)

- An automatic control of industrial processes: *smart grids*, *manufacturing*, *healthcare*, *water treatment*, *transportation*, etc.
- Physical processes integrated with computations via networks.

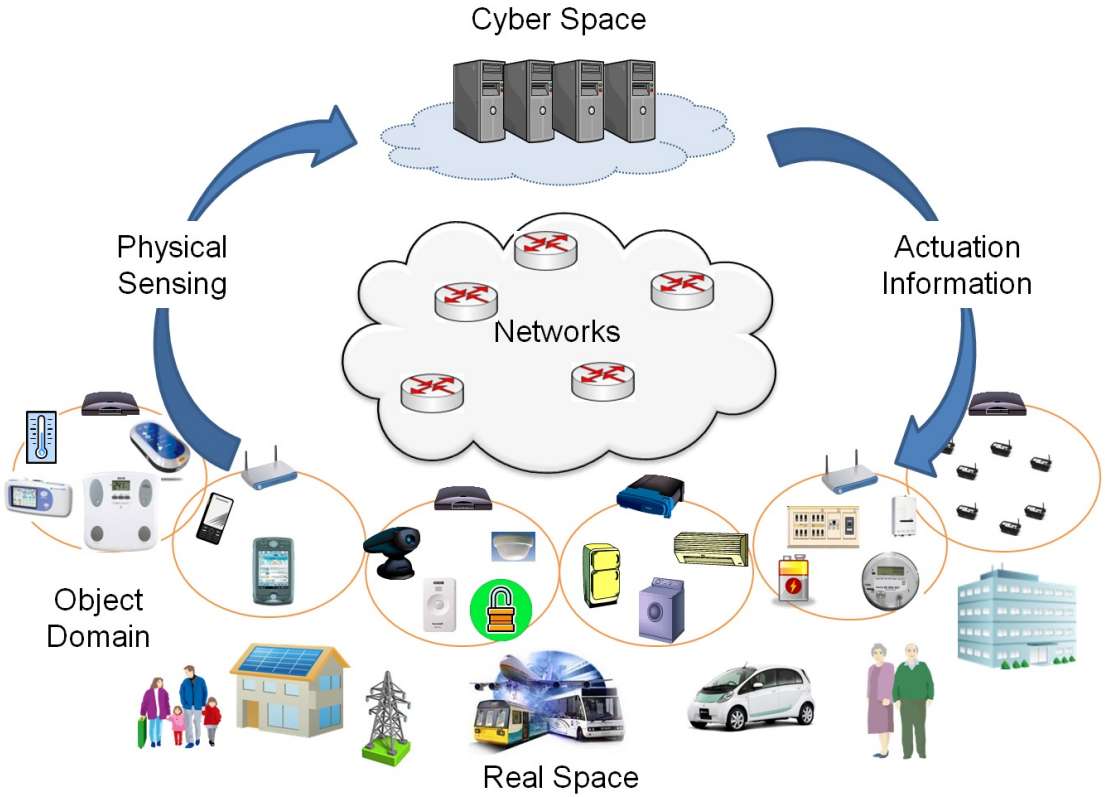


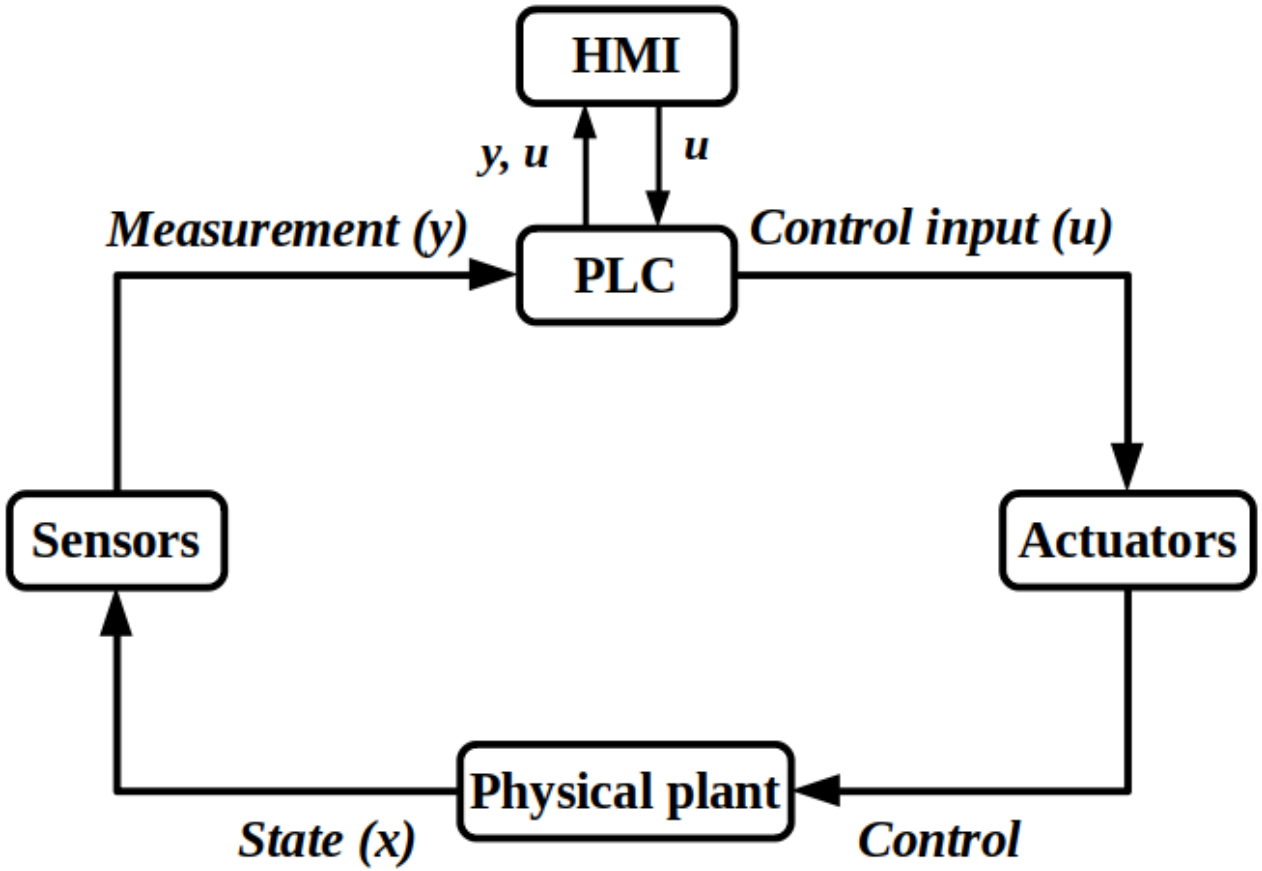
Figure 1: ICS architecture

ICS abstraction

- ICS model with linear-time invariant

$$x_{t+1} = Ax_t + Bu_t$$
$$y_t = Cx_t$$

- Time is discretized. A, B and C are constant matrices.



IT Vs OT

- Unlike IT, ICS (OT)
  - Have long life cycle (15 – 20 years).
  - Contains *resource-constrained devices*, e.g. PLCs, Sensors, Actuators.
  - Are constrained with *hard real-time* requirements.
  - The security priority is *Availability, Integrity and Confidentiality*.

Outline

The conflicting design constraints in ICS

- Security
- Efficiency
- Availability/Resilience

Outline

Security

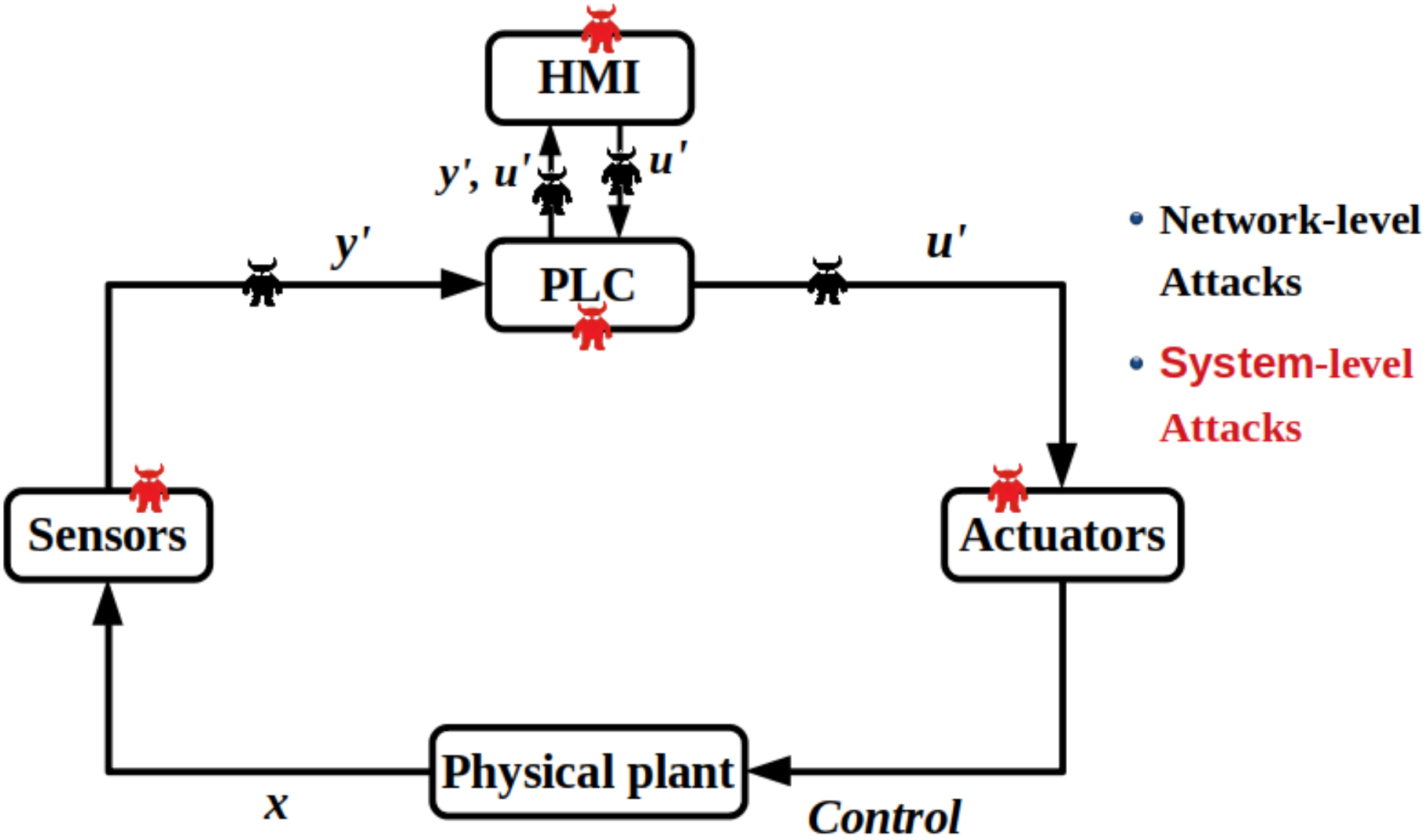


Figure 2: Attack vectors in ICS

Security

- Network-level attacks
  - Man-in-the-middle attacks
  - Reply attacks
  - Dos/DDoS attacks
- System-level attacks
  - Memory-safety attacks
  - Side-channel attacks
  - Malware/Virus
- So, *Security* is a critical concern in ICS.

Outline

Efficiency

- Most security solutions introduce high runtime overheads
  - Cryptographic solutions
  - Machine-learning based solutions
  - Memory-safety solutions
  - Intrusion detection systems
  - Etc.
- These overheads can cause delays (say δ) at runtime.

Efficiency

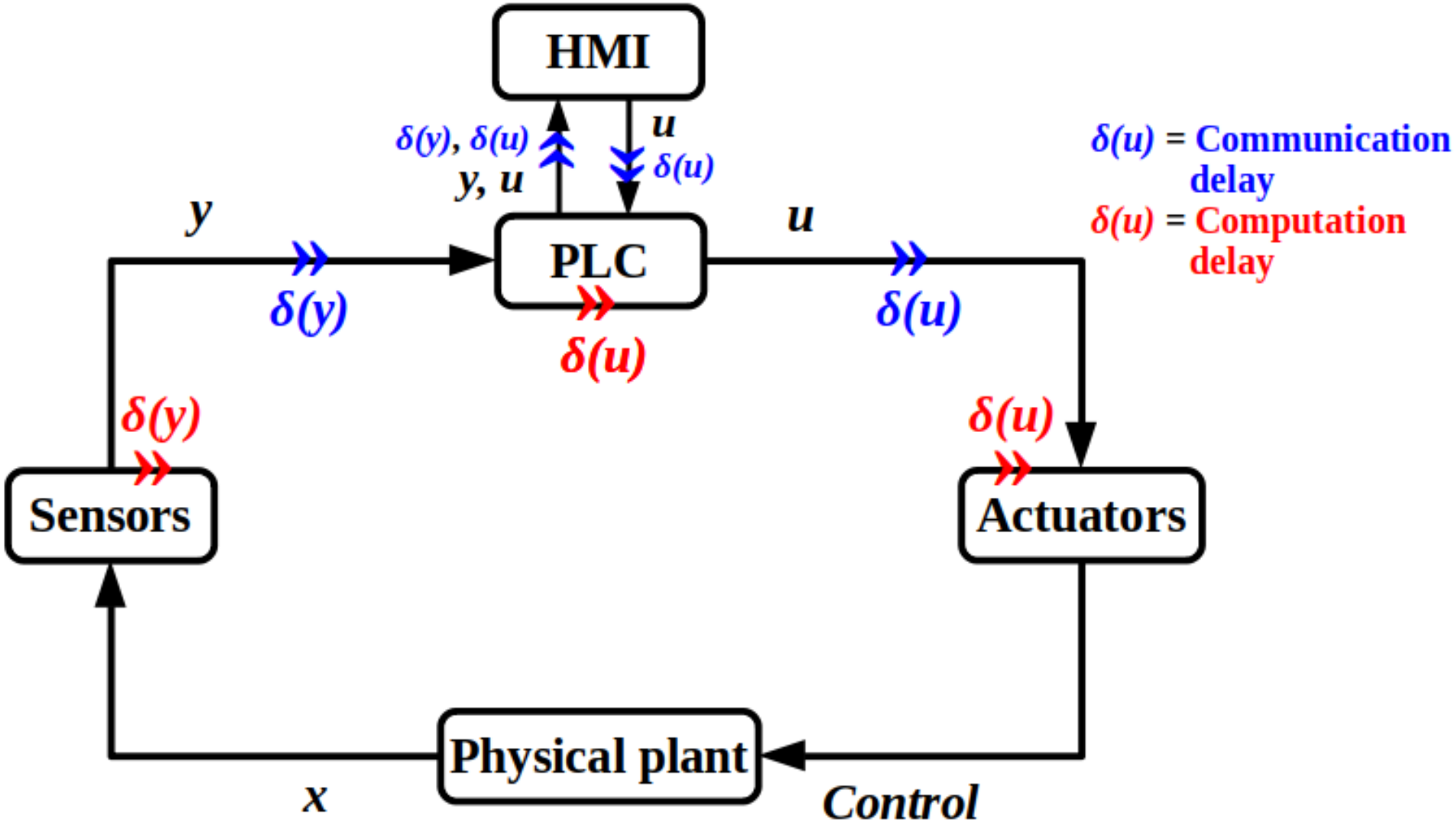


Figure 3: Delay in ICS

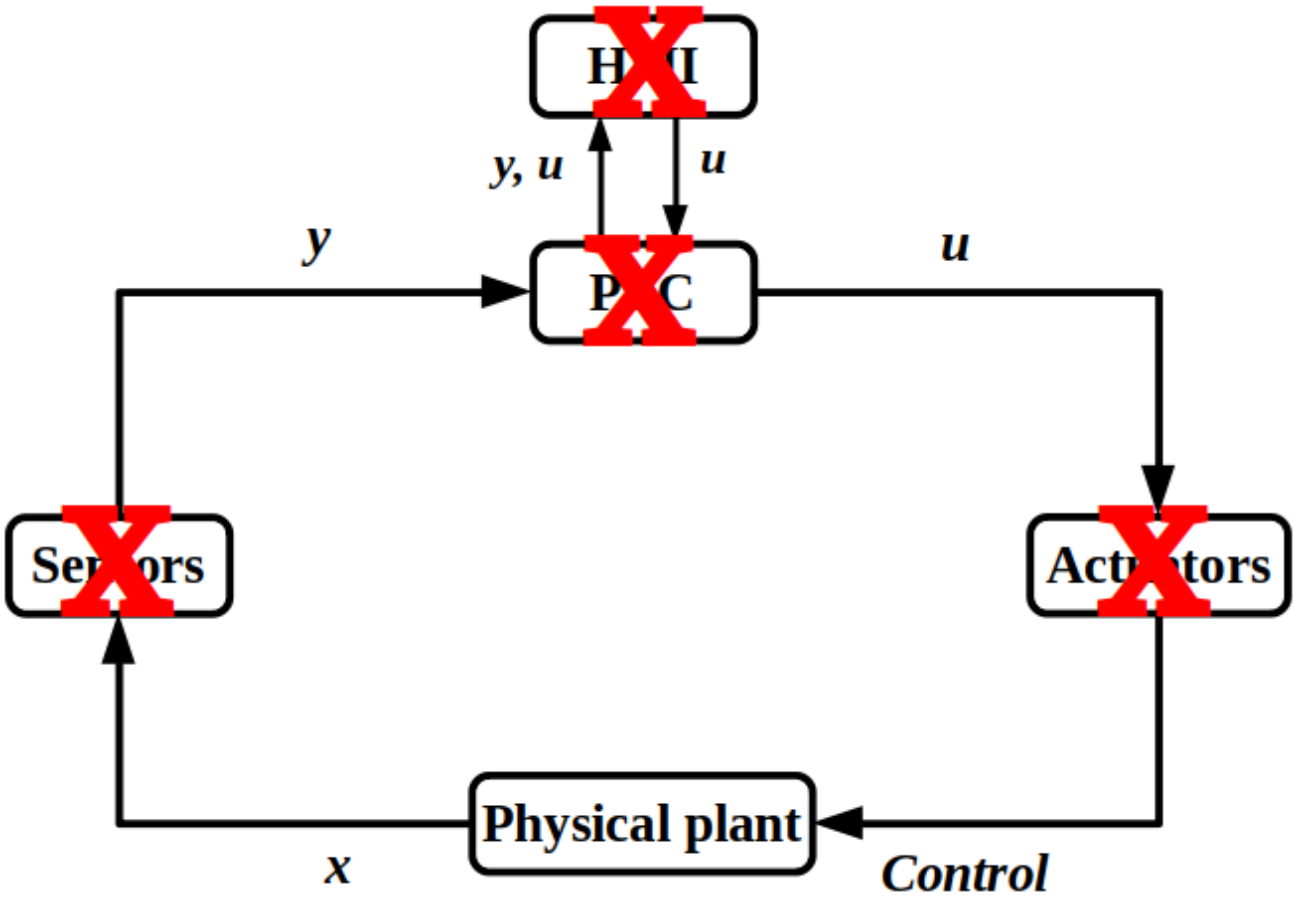
Efficiency

- ICS devices, e.g. PLCs, have limited computational power!
- ICS are highly delay-sensitive systems
  - PLCs have hard real-time constraints.
  - Communications are synchronized by system time.
- Failing to meet the real-time constraints could lead to,
  - Disruption of the control system
  - Damage to the physical plant
- Thus, *Efficiency* is a critical concern in ICS!!

Outline

Availability

- System unavailability could happen due to,
  1. A service delay due to a security overhead (i.e. δ).
  2. The system is down for some reason, e.g., the system is restarted/ aborted to mitigate an attack or it is compromised by an attack.



Availability

- System *unavailability* is a critical concern in ICS, as it leads the control system to unsafe state.

Availability

- System *unavailability* is a critical concern in ICS, as it leads the control system to unsafe state.

In ICS, ***Security***, ***Efficiency*** and ***Availability*** are equally important!

The research problem

- What overhead is considered to be tolerable (efficient) and how it can be quantified?
- What level of unavailability is still acceptable with respect to the process dynamics in ICS? How can we quantify that?
- How can we address the SEA tradeoffs in ICS?

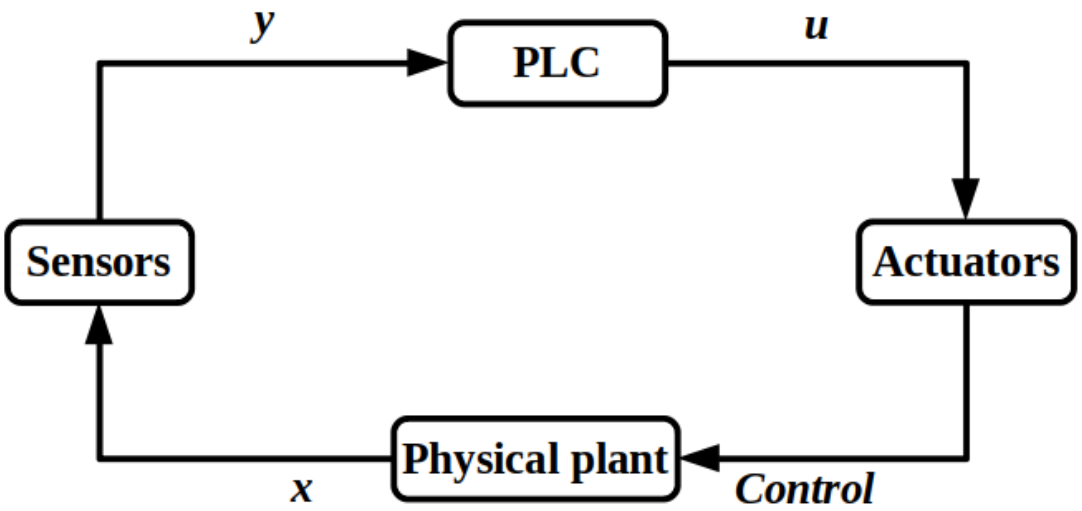
Outline

Modeling the ICS design constraints

- In this work, we model:
  - *Efficiency* based on *real-time constraints* (RTC) in ICS.
  - *Availability* based on *physical-state resiliency* (PSR) in ICS.

Outline

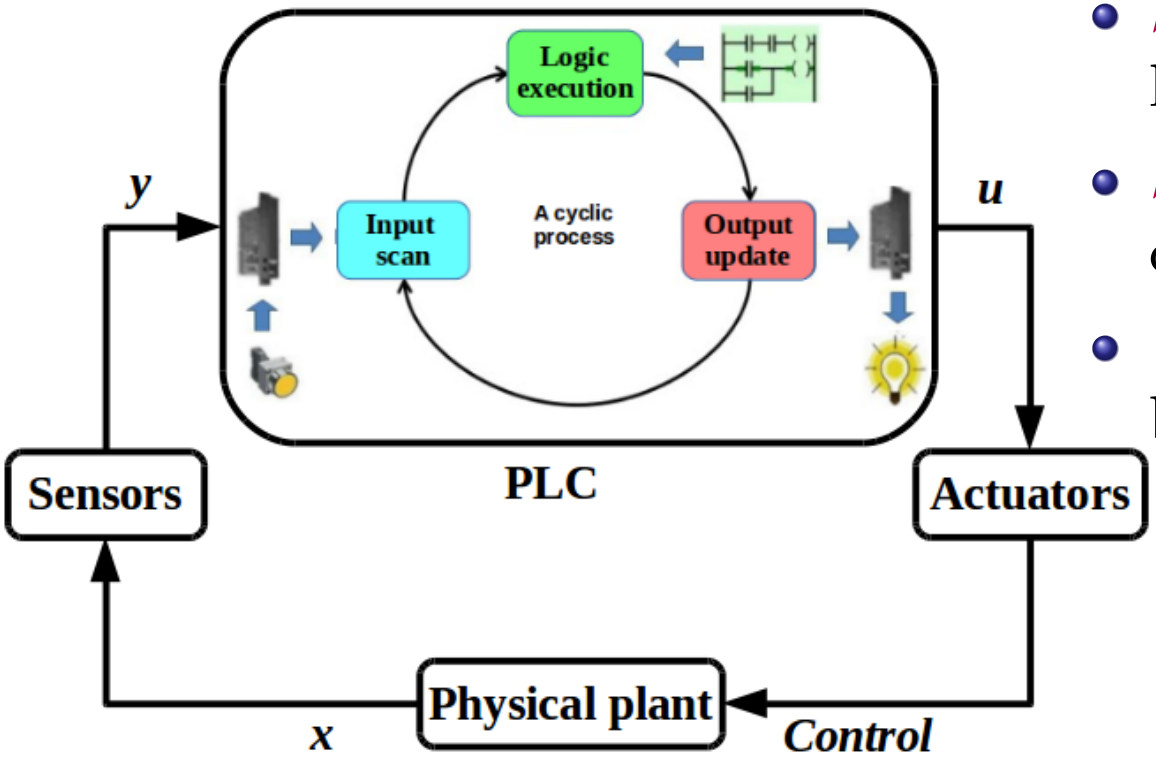
Modeling the real-time constraints



- *Scan cycle*: input scan + logic exec. + output update.
- *Scan time (T_s)*: time taken to complete the PLC scan cycle.
- *Cycle time (T_c)*: an upper bound the PLC scan time.

- The *real-time constraint* of the PLC is $T_s \leq T_c$.

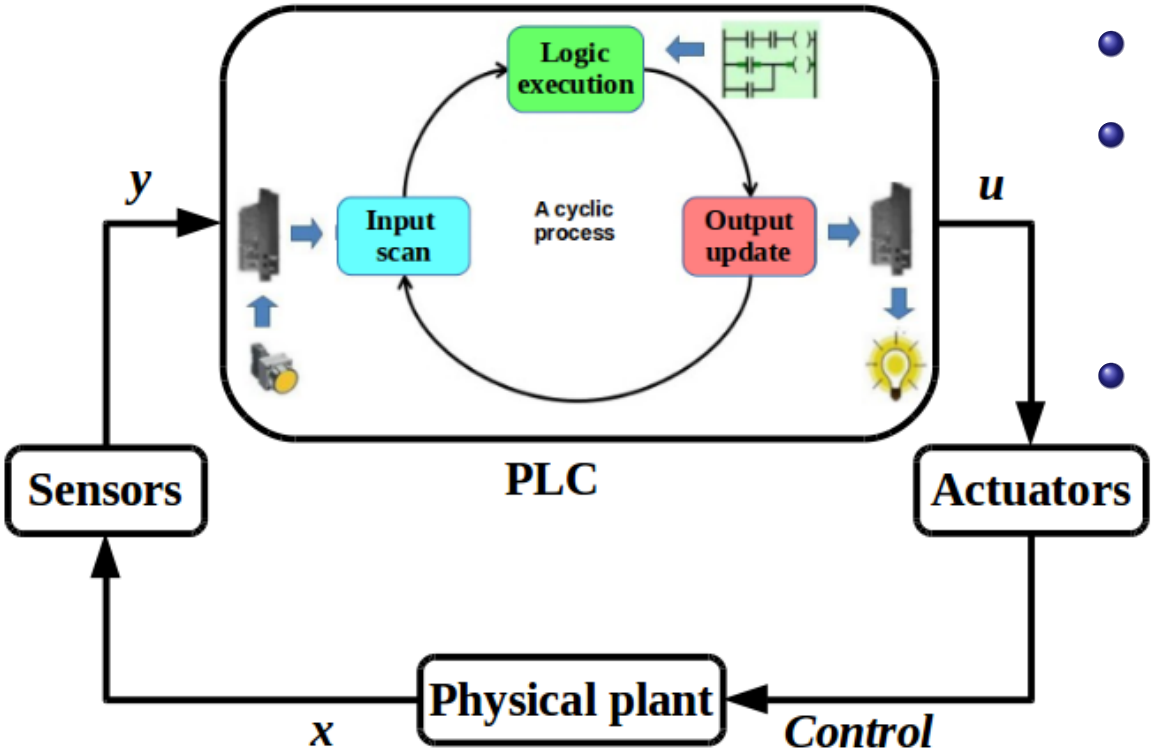
Modeling the real-time constraints



- *Scan cycle*: input scan + logic exec. + output update.
- *Scan time (T_s)*: time taken to complete the PLC scan cycle.
- *Cycle time (T_c)*: an upper bound the PLC scan time.

- The *real-time constraint* of the PLC is $T_s \leq T_c$.

Quantifying tolerability of an overhead



- By design, $T_s \leq T_c$.
- Suppose T_s' is the scan time after enforcing a security solution in the ICS.
- The security overhead is *tolerable* if $T_s' \leq T_c$.

Figure 3: The PLC scan cycle

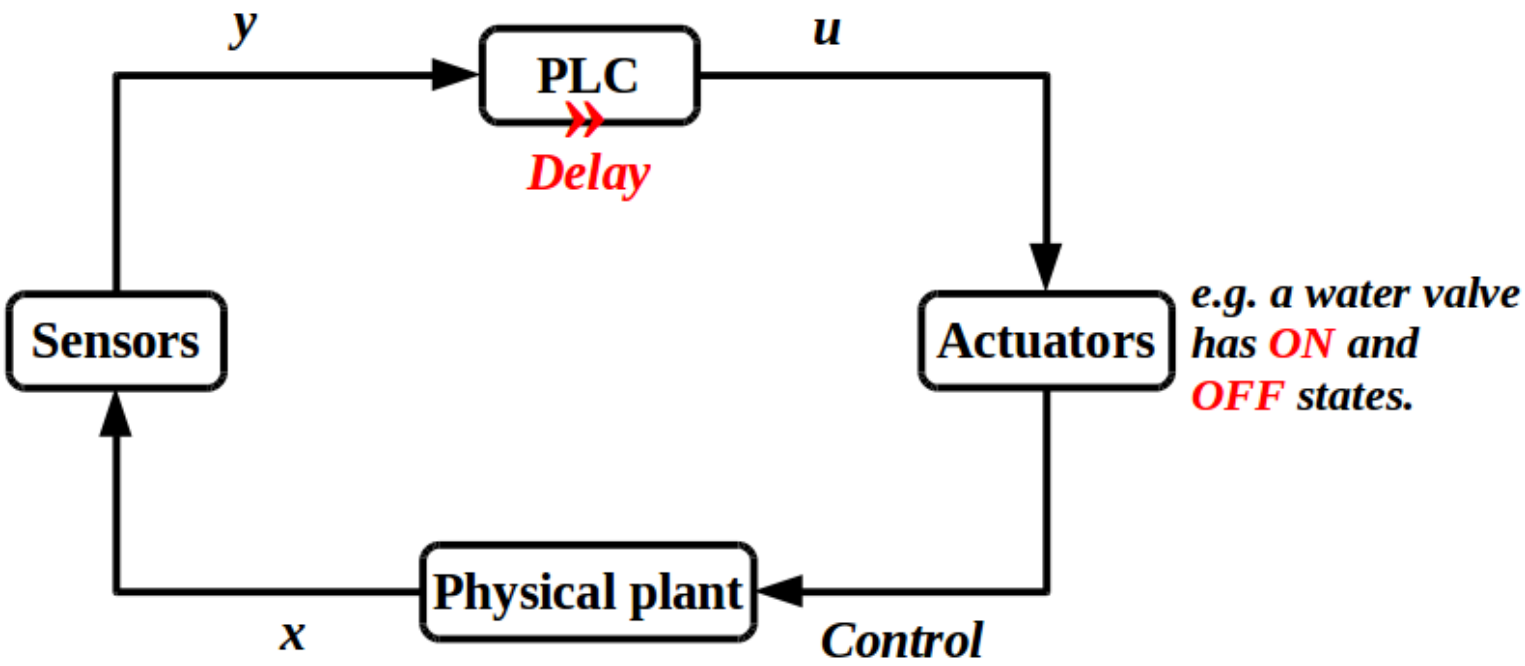
Outline

Modeling physical-state resiliency

- Delay in control input could disrupt the ICS dynamics.

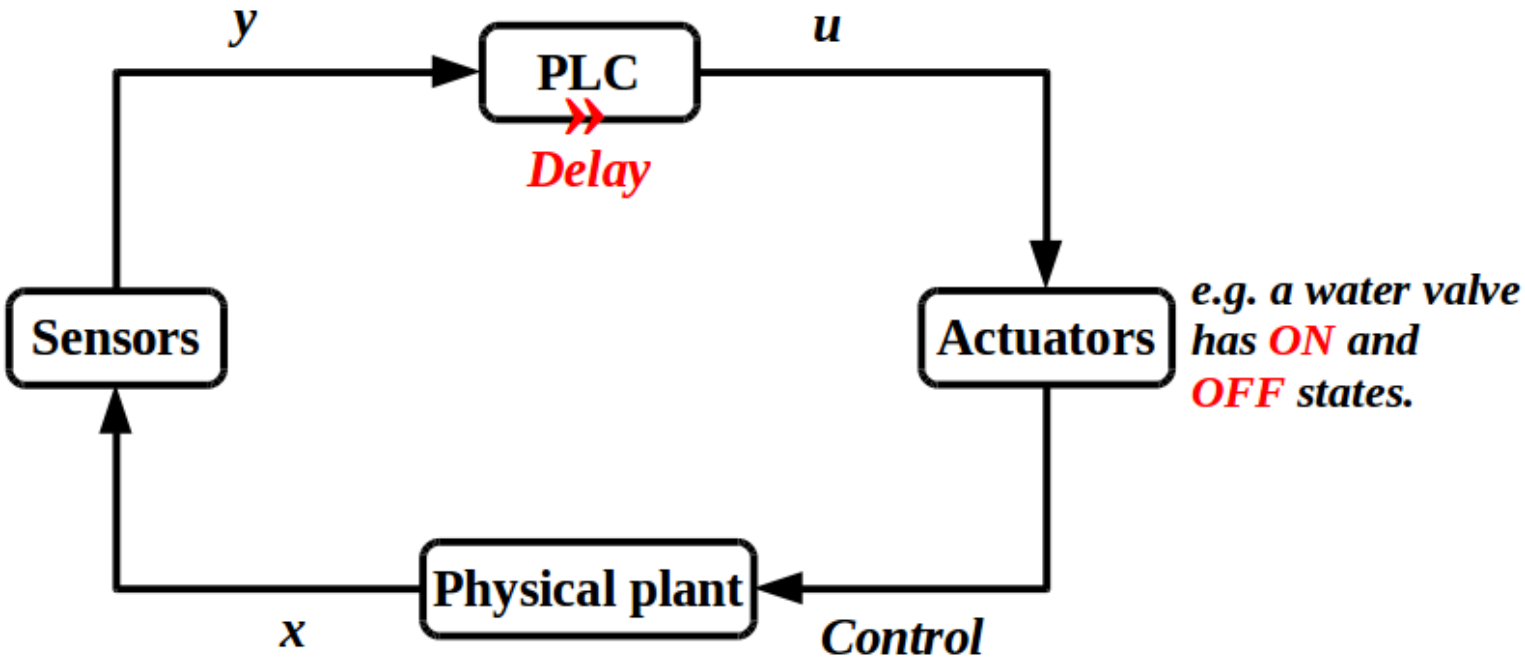
Modeling physical-state resiliency

- Delay in control input could disrupt the ICS dynamics.



Modeling physical-state resiliency

- Delay in control input could disrupt the ICS dynamics.



- The delay (i.e. δ) could happen when,
  1. $T_s' > T_c$ (i.e. due to overhead) $=> \delta = T_s' - T_c$.
  2. The PLC is down for some reason $=> \delta = \infty$.

Modeling physical-state resiliency

- Under what level of delay is the ICS still stable?

Modeling physical-state resiliency

- Under what level of delay is the ICS still stable?
- State estimation of the plant at time t is,

$$x_{t+1} = Ax_t + Bu_t, \text{ without delay.}$$
$$x'_{t+1} = Ax_t + Bu_{t-1}[\![t, t+\tau]\!], \text{ with delay } \tau.$$

Modeling physical-state resiliency

- Under what level of delay is the ICS still stable?
- State estimation of the plant at time t is,

$$x_{t+1} = Ax_t + Bu_t, \text{ without delay.}$$
$$x'_{t+1} = Ax_t + Bu_{t-1}[\![t, t+\tau]\!], \text{ with delay } \tau.$$

- Suppose ω and θ are the upper and lower bounds of x_t.
- The control-loop is stable (with delay τ) if the following holds:

$$\theta \leq x'_{t+1} \leq \omega$$

$$\theta \leq Ax_t + Bu_{t-1}[\![t, t+\tau]\!] \leq \omega$$

Our recommendation

- To set RTC and PSR as a runtime safety properties in ICS.
  - To be checked at runtime.
- to raise alarm in case of violation.

Outline

Testing security solutions

- Memory-safety solutions
  - ASan
  - CIMA
- Cryptographic solutions
  - 2FA
  - LCDA

Overview of memory-safety attacks

- C/C++ languages are vulnerable to memory-safety bugs.
  - Buffer over/underflows
  - Dangling pointers
  - Memory leaks, etc
- Compilers do not have inherent safety/security checks.

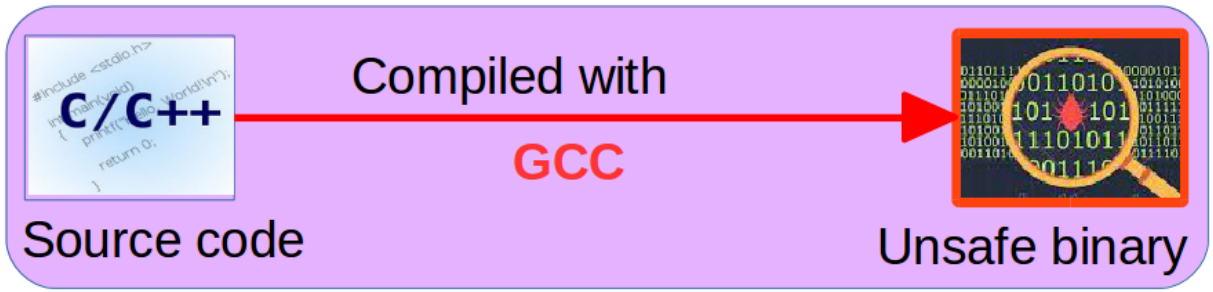


Figure 4: Unsafe compilation

- Unsafe binaries lead to *runtime crashes* or *cyber attacks*.

Crashes and attacks

```
foo(){
    char buffer[16];
    printf("Insert input: ");
    gets(buffer);
}
```

Crashes and attacks

```
foo(){
    char buffer[16];
    printf("Insert input: ");
    gets(buffer);
}
```

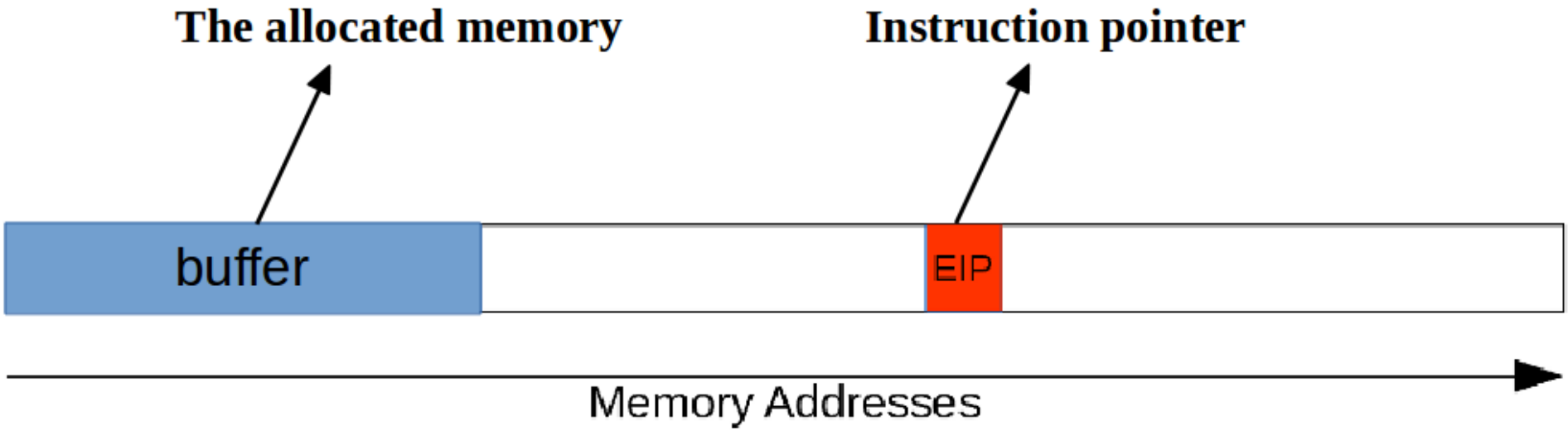


Figure 5: The memory layout

Crashes and attacks

```
foo(){
    char buffer[16];
    printf("Insert input: ");
    gets(buffer);
}
```

Crashes and attacks

```
foo(){
    char buffer[16];
    printf("Insert input: ");
    gets(buffer);
}
```

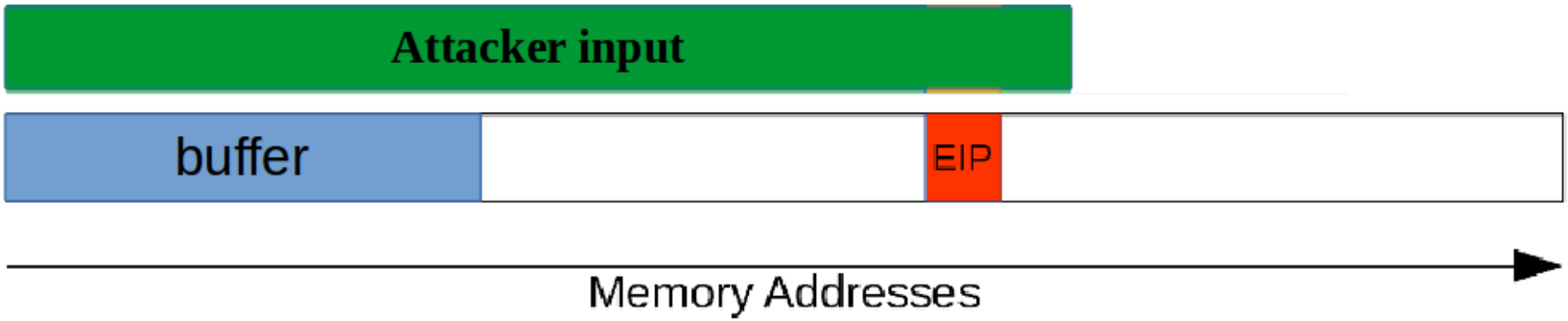


Figure 6: Overflowing the buffer

Crashes and attacks

```
foo(){
    char buffer[16];
    printf("Insert input: ");
    gets(buffer);
}
```

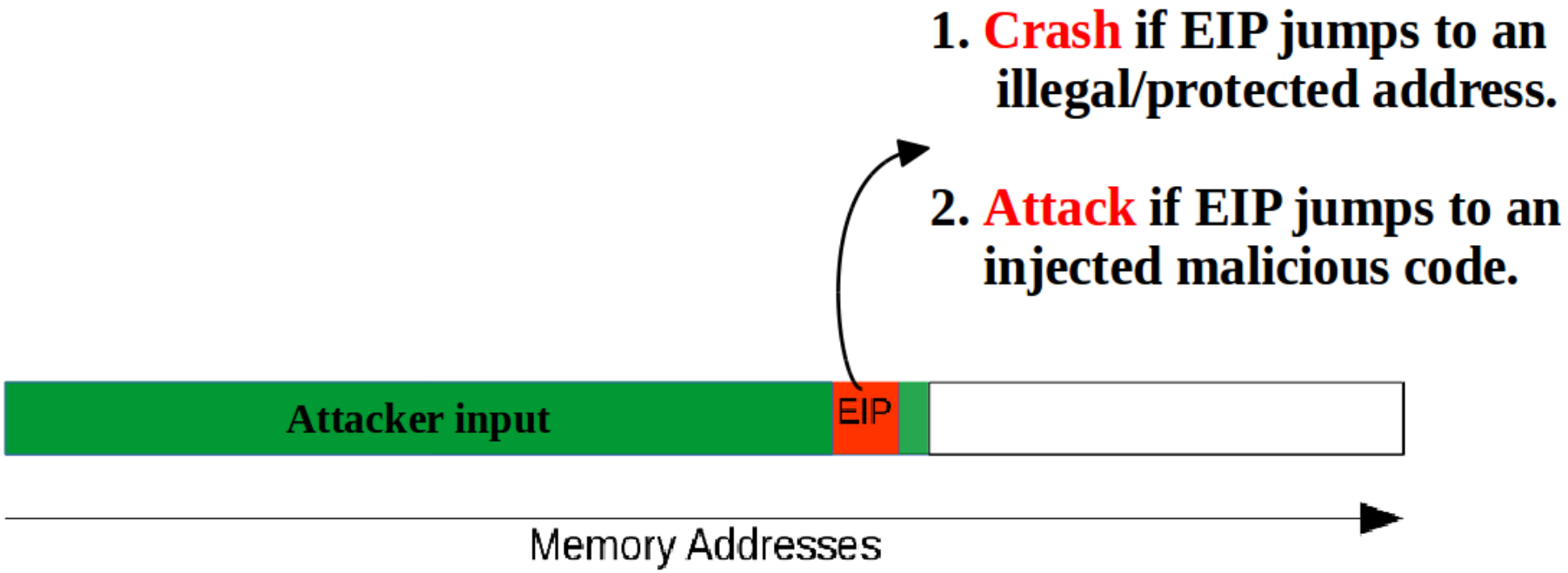


Figure 6: Overflowing the buffer

Memory-safety attacks exploitation

```
foo(){
    char buffer[16];
    printf("Insert input: ");
    gets(buffer);
}
```

- Create a tailored input: overwrite EIP with known address.

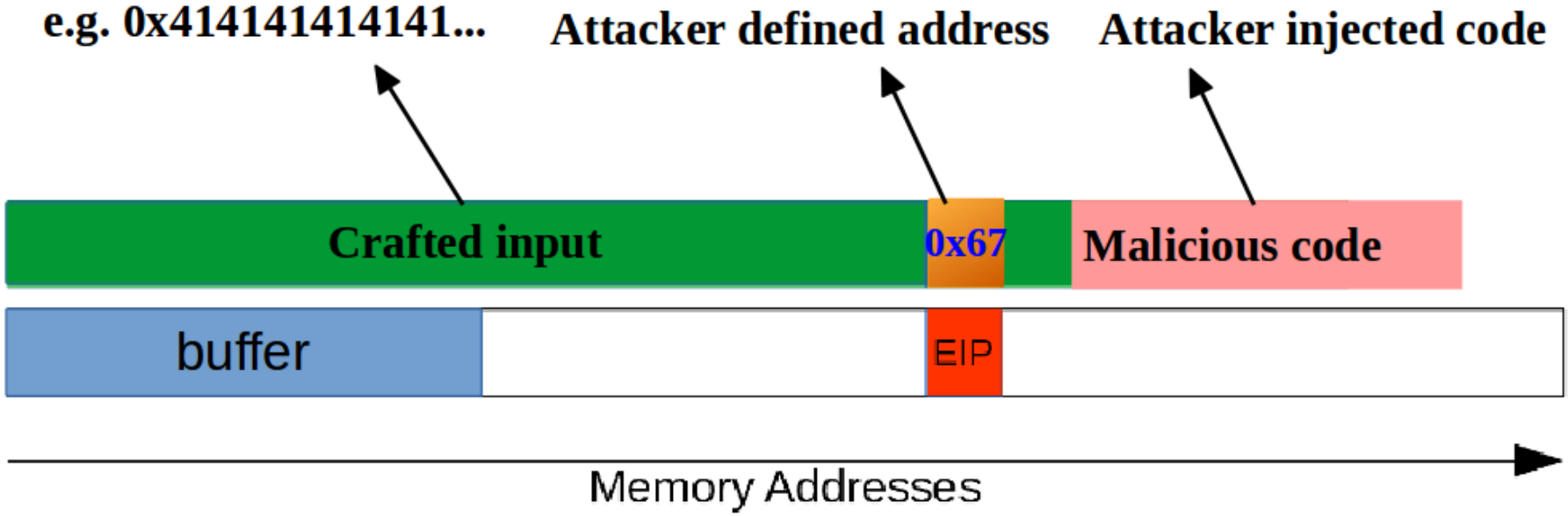


Figure 7: Overwrite EIP with crafted input

Memory-safety attacks exploitation

```
foo(){
    char buffer[16];
    printf("Insert input: ");
    gets(buffer);
}
```

- Divert control to the new address (hijack control flow)

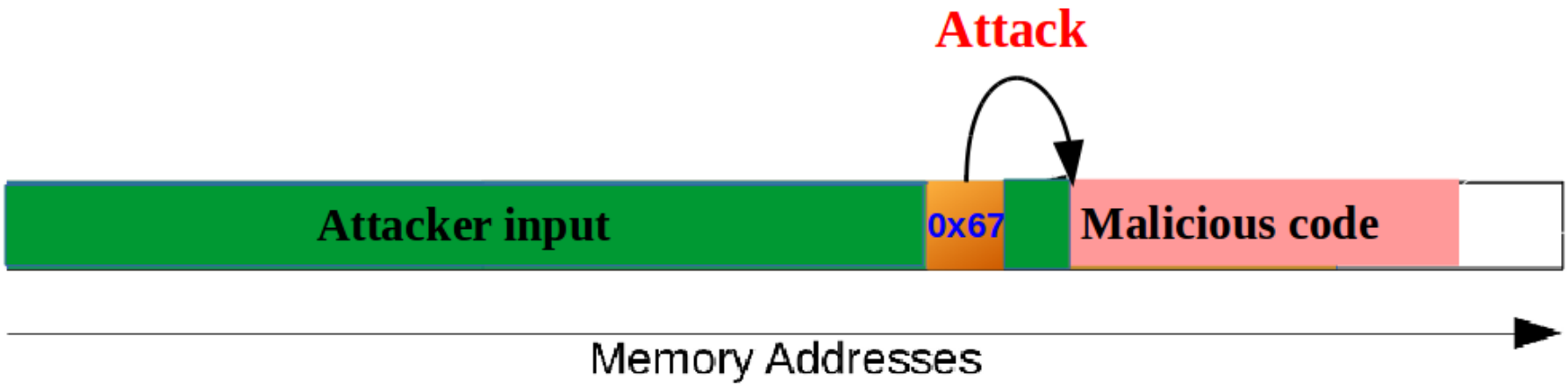


Figure 7: Diverting control to the injected code or existing module

Outline

Overview of ASan

- ASan is a memory-safety tool based on,
  - Compile-time code instrumentation.
  - Shadow memory mapping.
  - Creating poisoned regions, aka redzones.

ASan instrumentation and shadow mapping

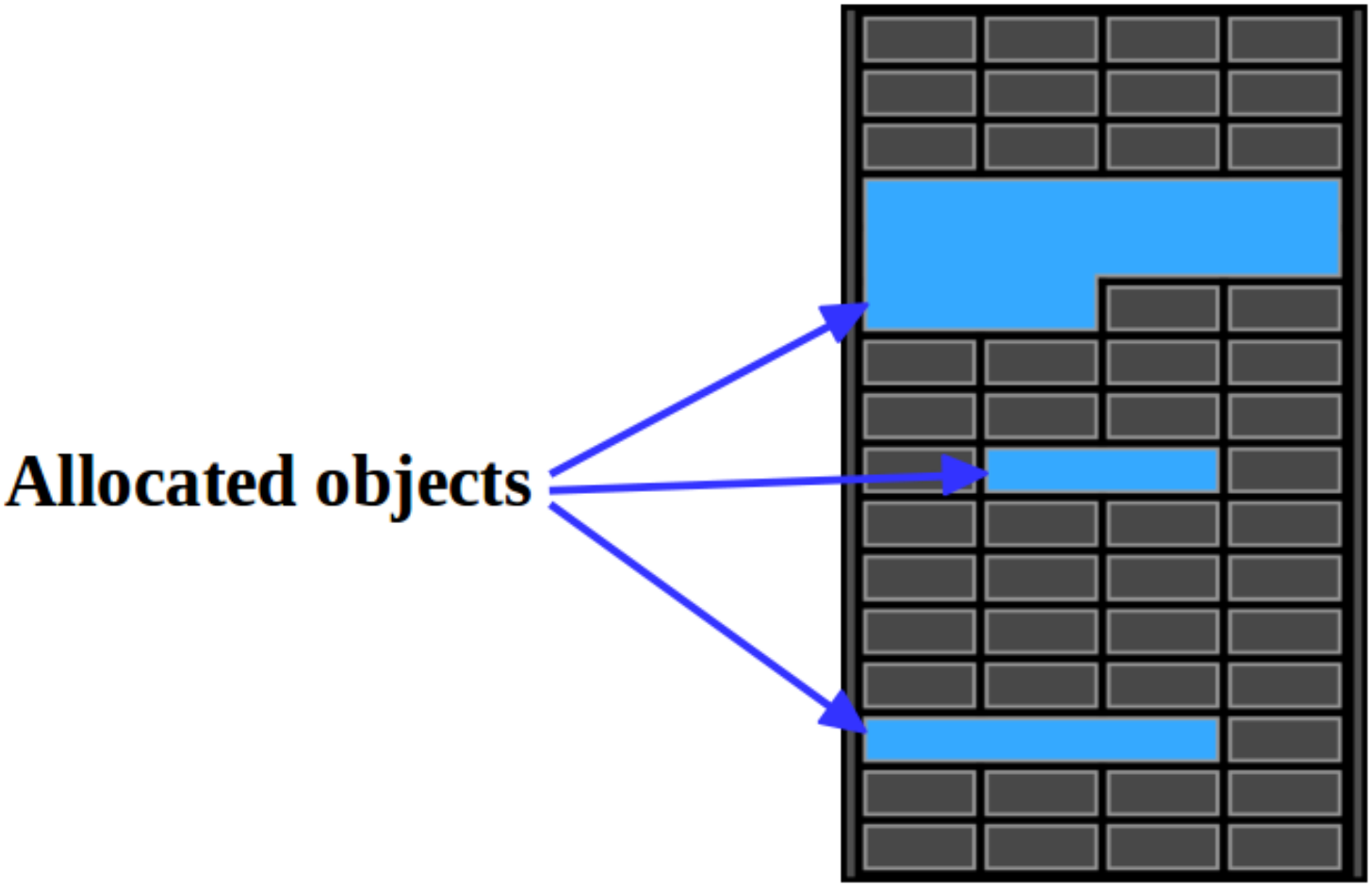
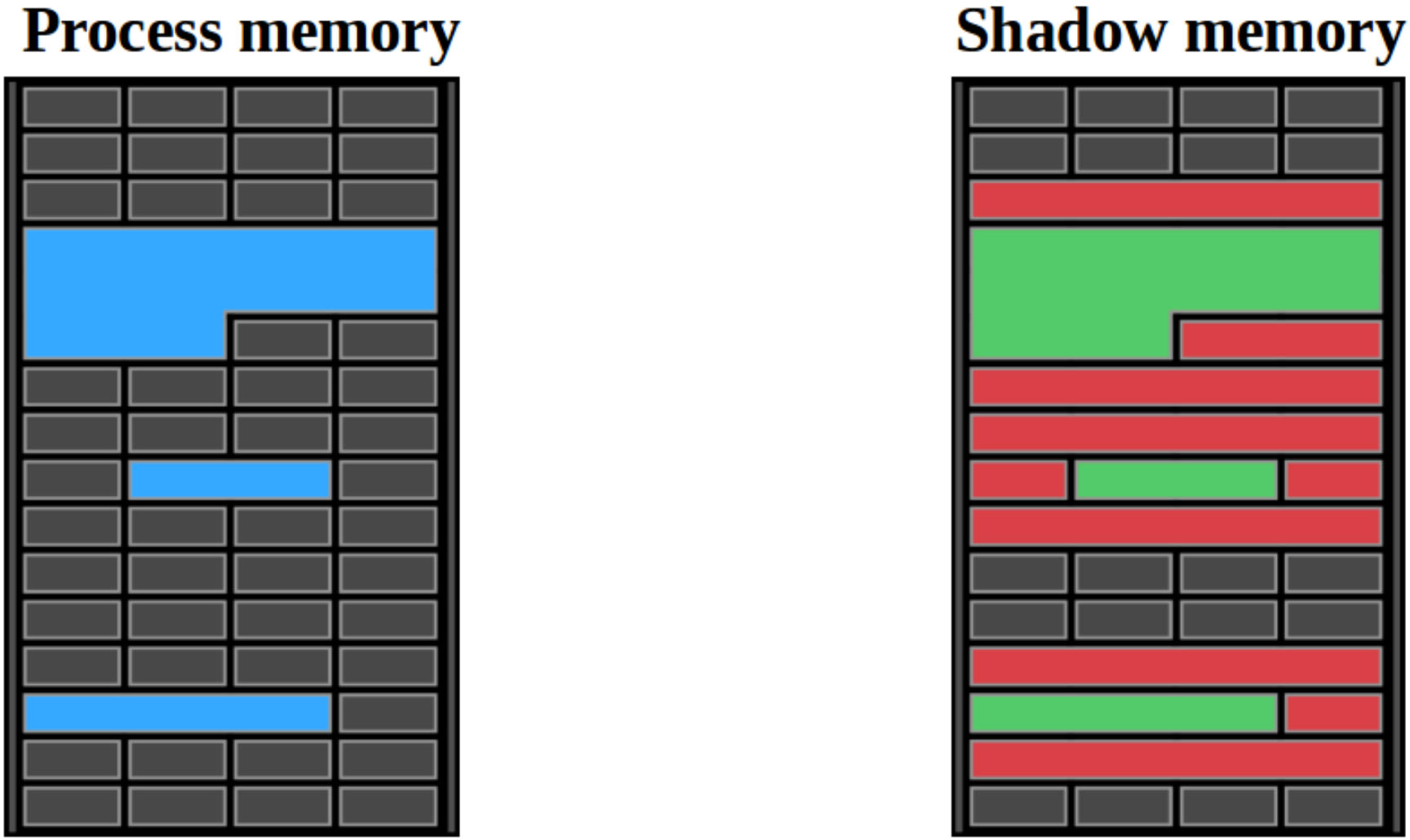
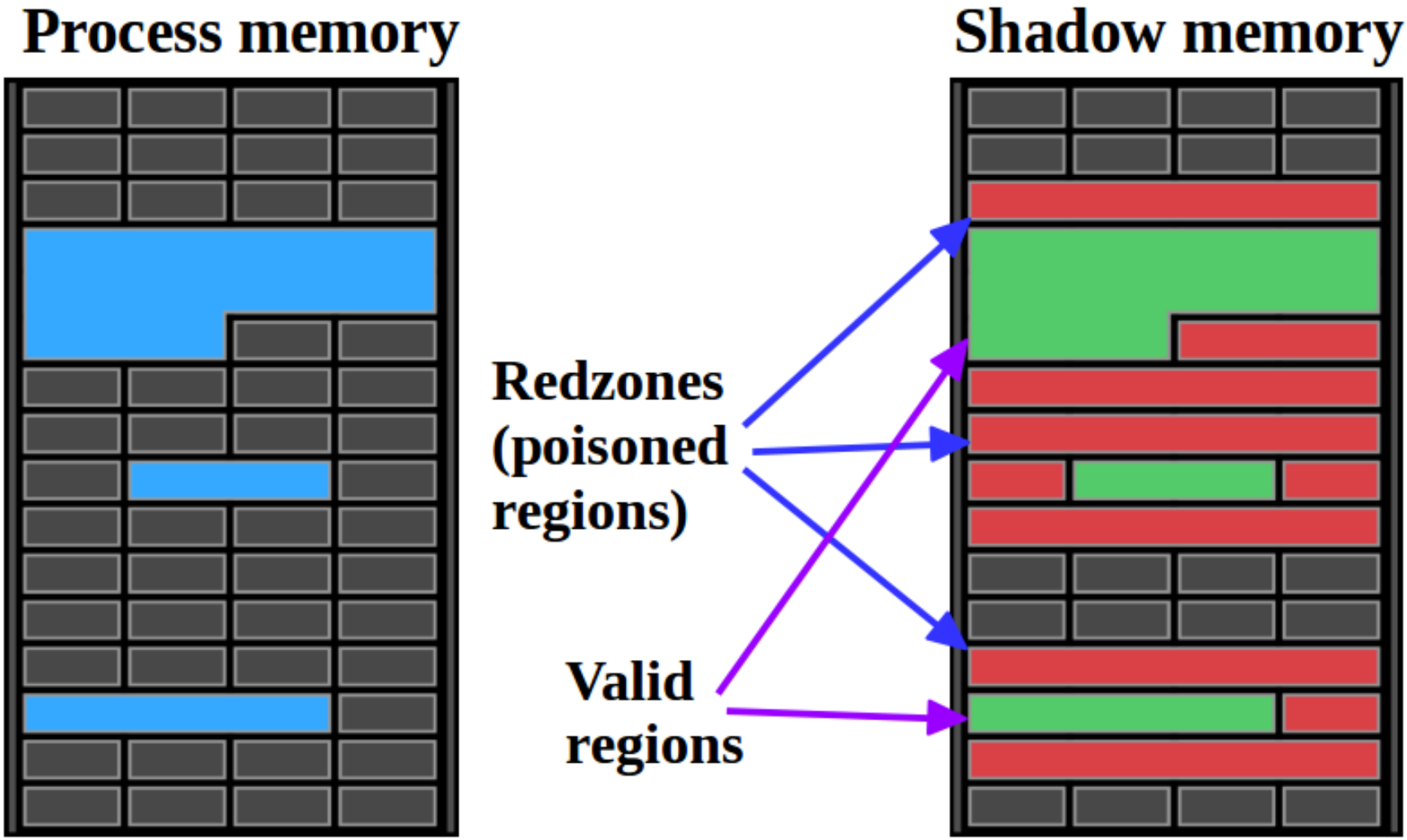


Image source: *Mike Swingler, Anna Zaks. "Advanced Debugging and the Address Sanitizer", WWDC15, Apple Inc.*

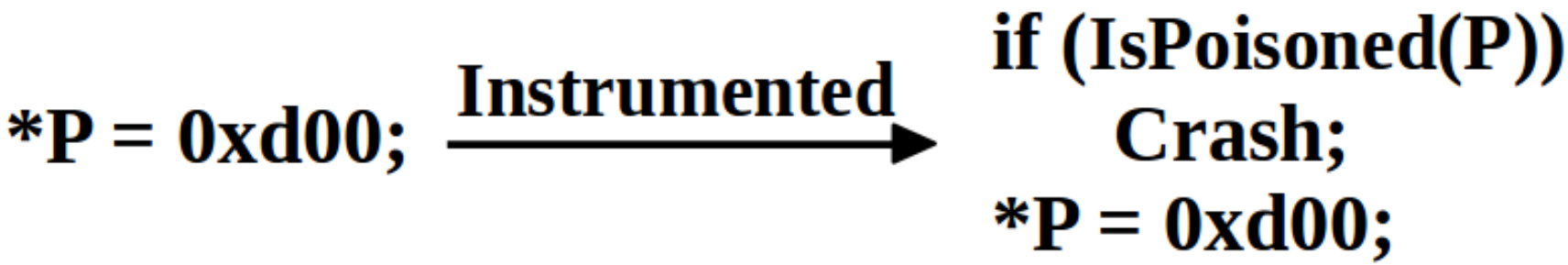
ASan instrumentation and shadow mapping



ASan instrumentation and shadow mapping



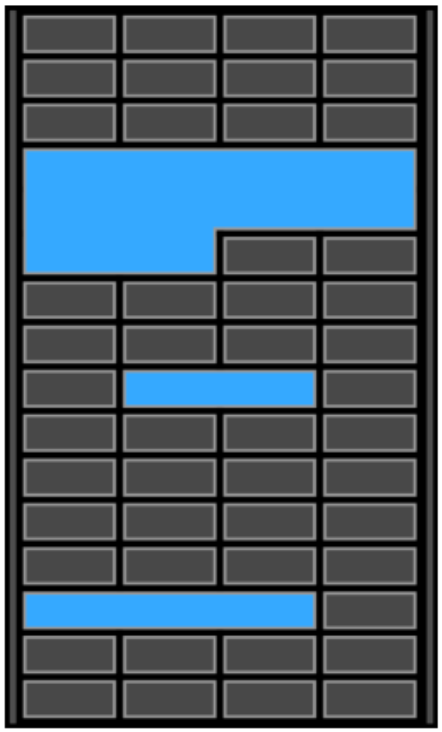
ASan instrumentation and shadow mapping



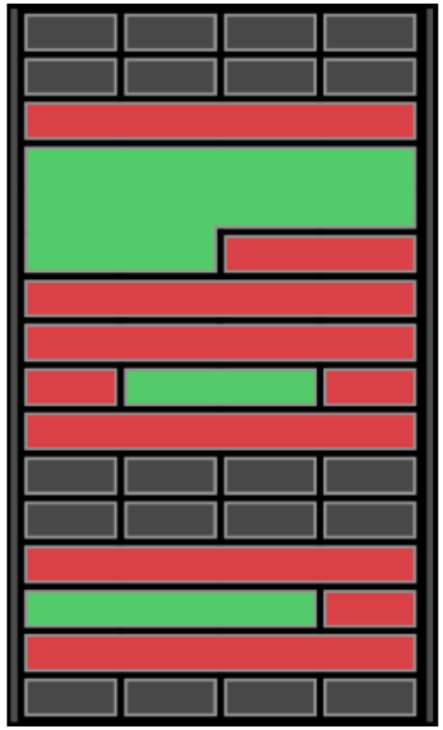
ASan instrumentation and shadow mapping

```
if (IsPoisoned(P))
    Crash;
*P = 0xd00;
```

Process memory

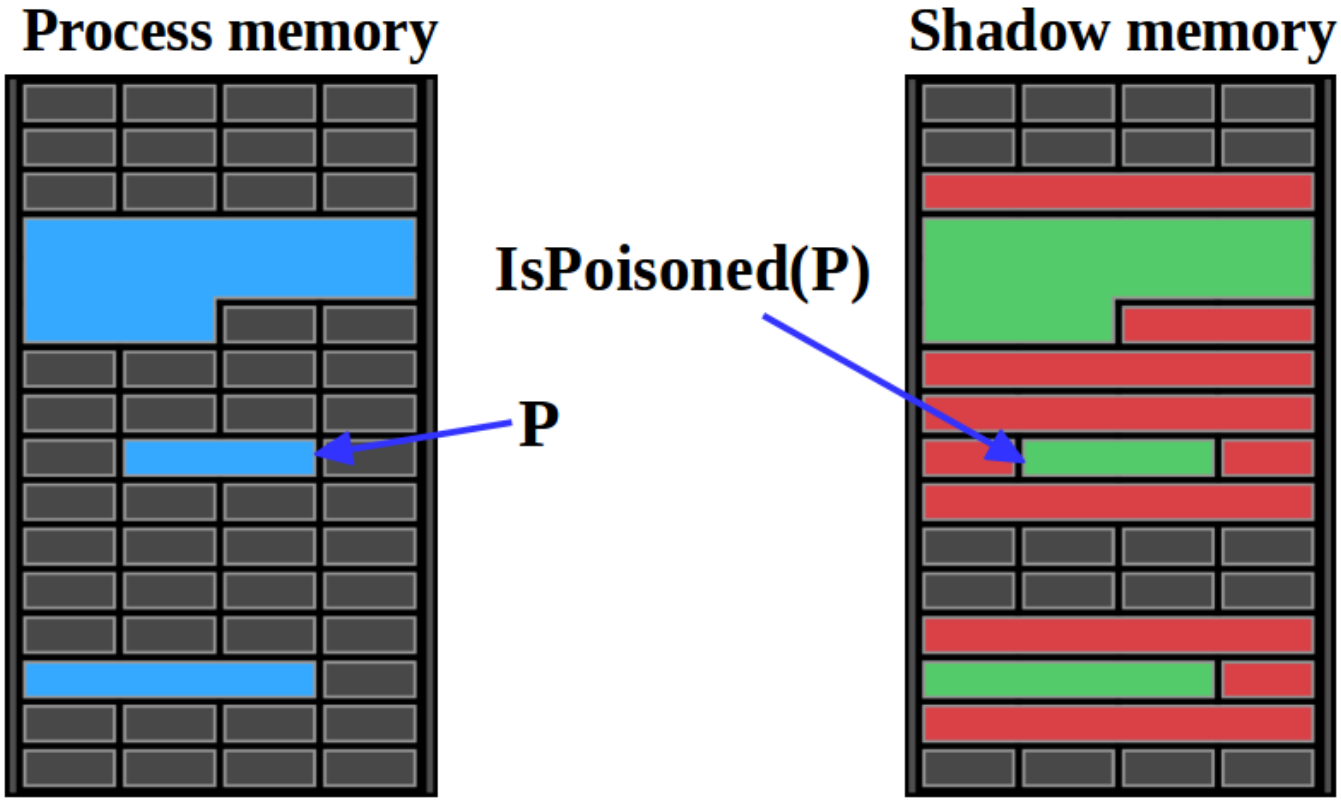


Shadow memory



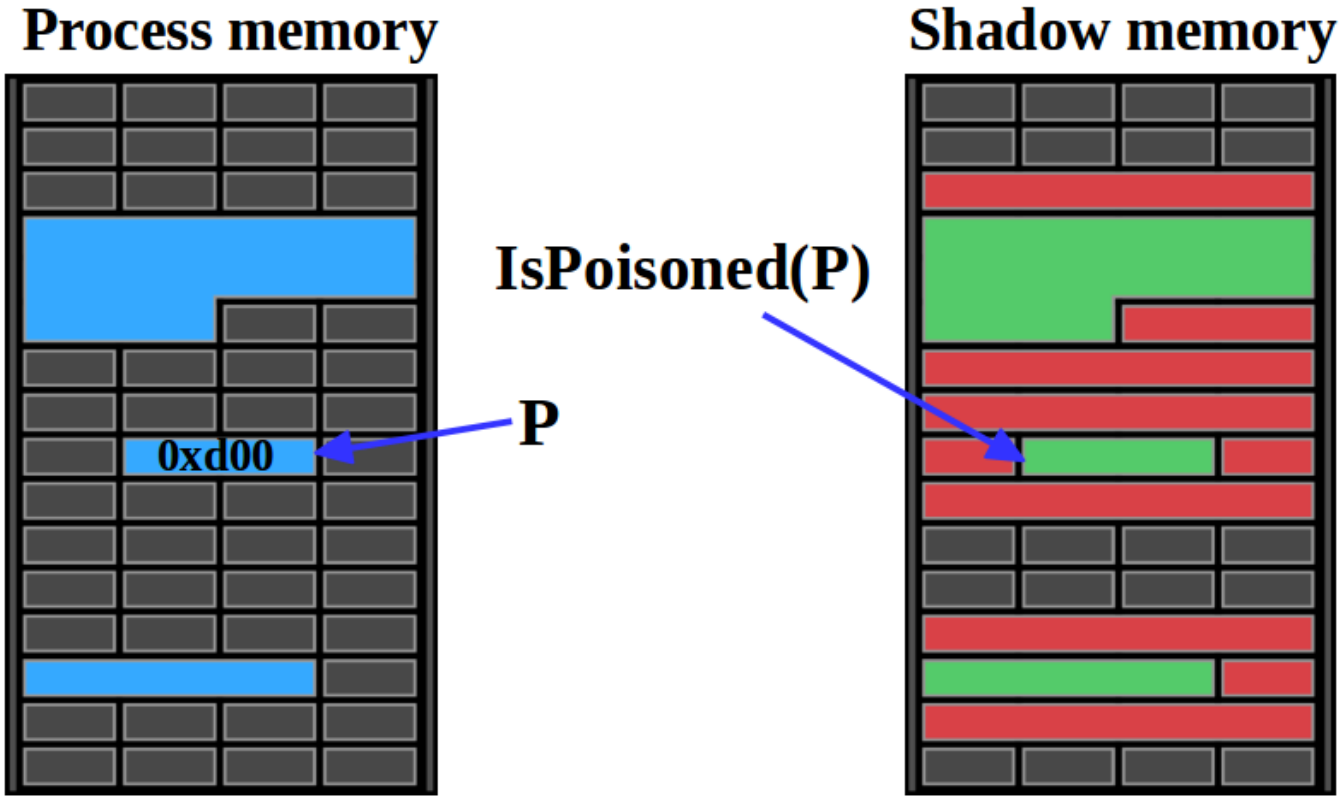
ASan instrumentation and shadow mapping

```
if (IsPoisoned(P))
    Crash;
*P = 0xd00;
```



ASan instrumentation and shadow mapping

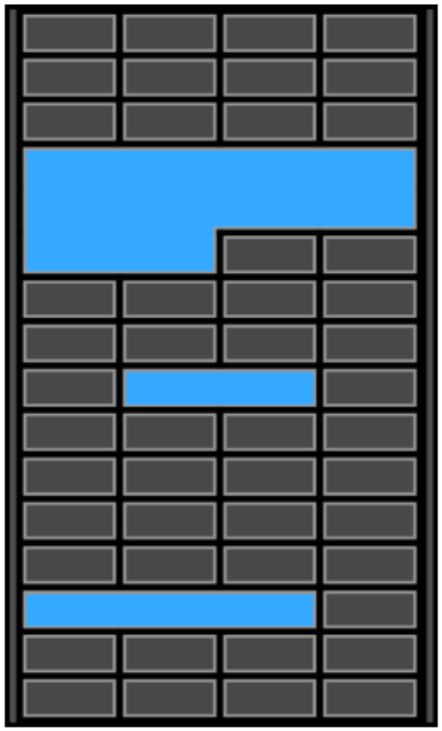
```
if (IsPoisoned(P))
    Crash;
*P = 0xd00;
```



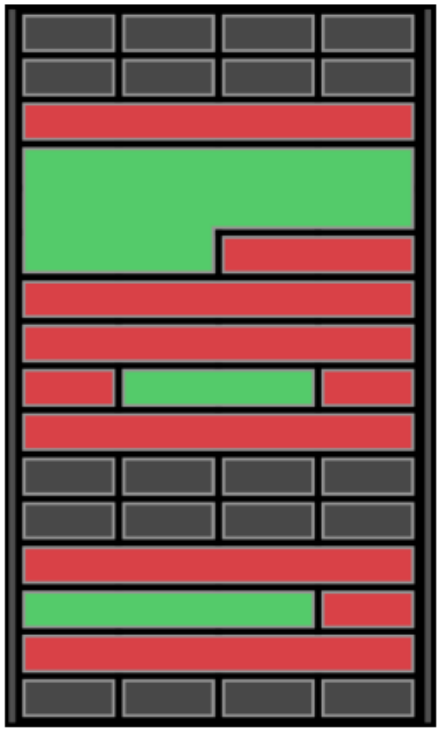
ASan instrumentation and shadow mapping

```
if (IsPoisoned(P))
    Crash;
*P = 0xd00;
```

Process memory

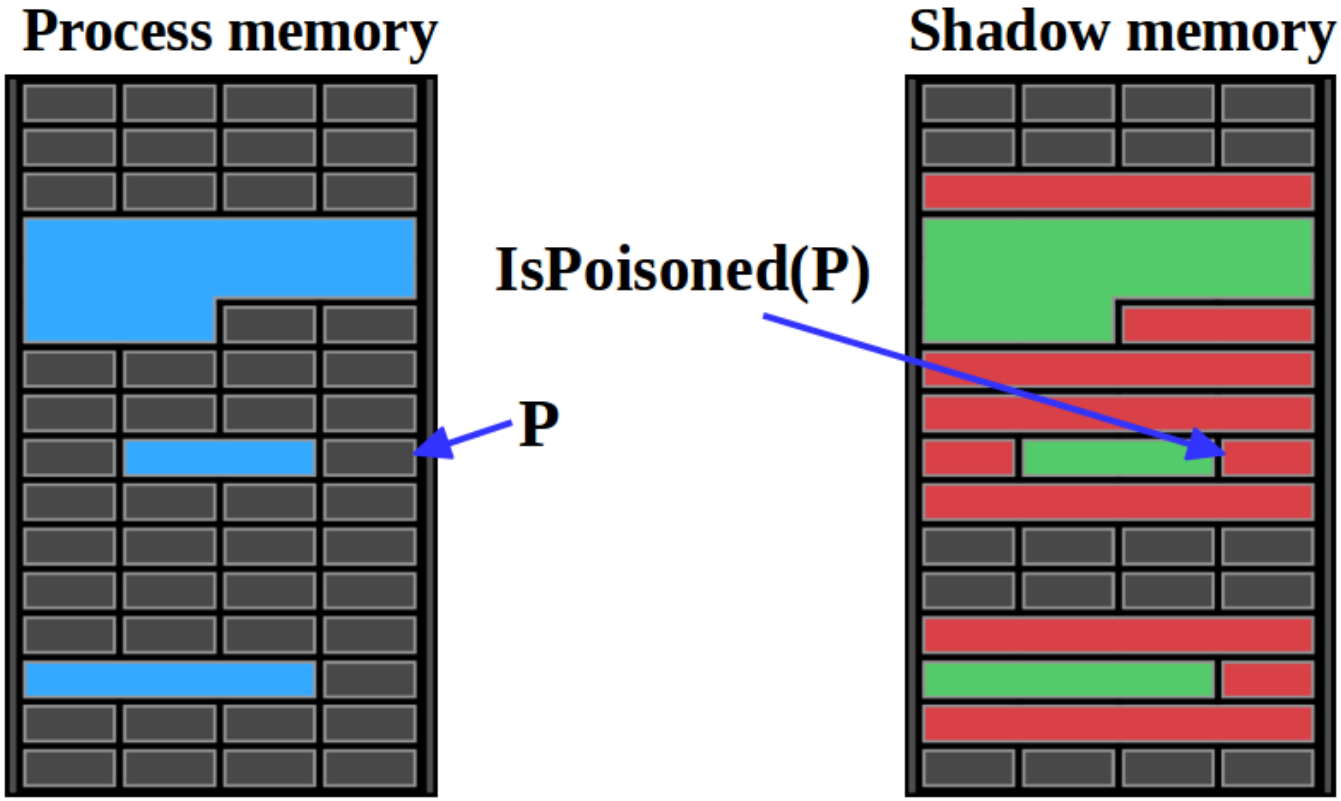


Shadow memory



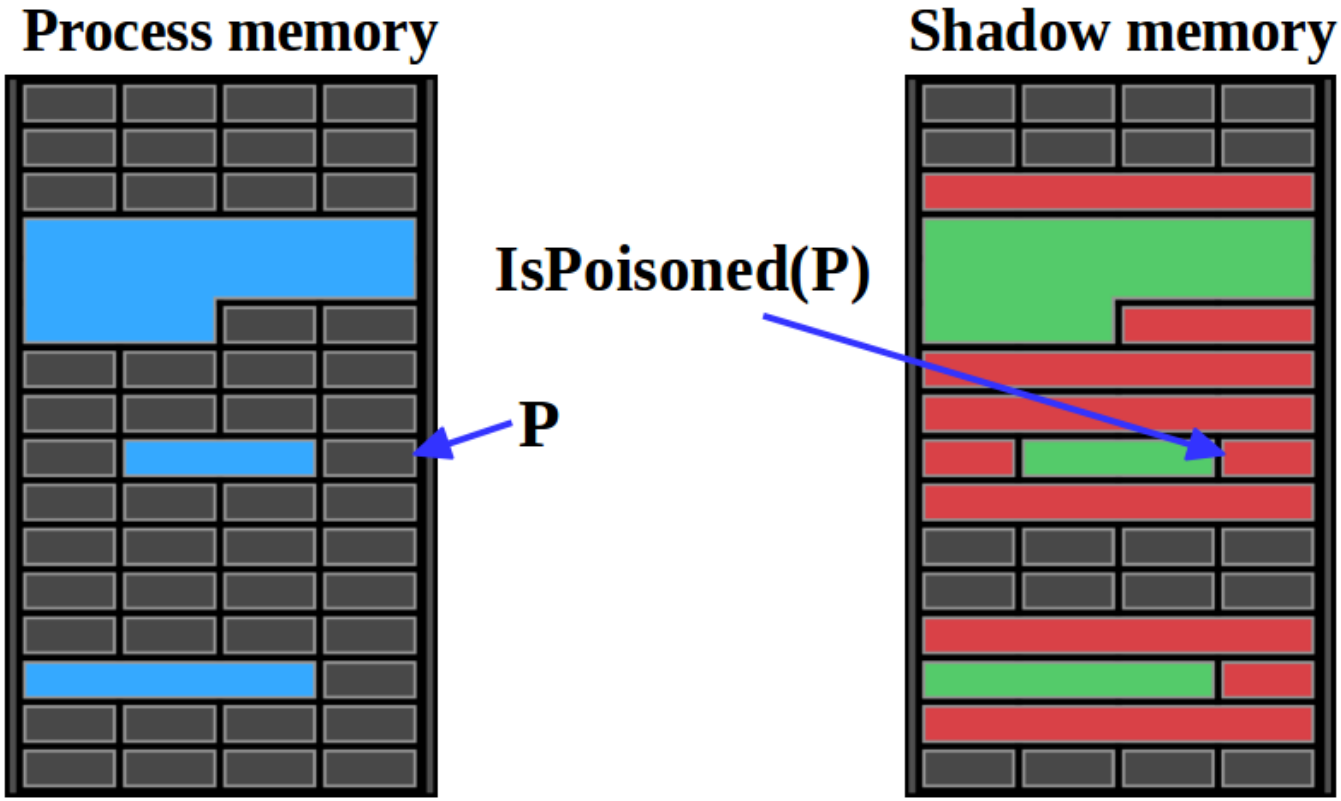
ASan instrumentation and shadow mapping

```
if (IsPoisoned(P))
    Crash;
*P = 0xd00;
```



ASan instrumentation and shadow mapping

```
if (IsPoisoned(P))
    Crash;
*P = 0xd00;
```



Outline

Overview of CIMA

- We developed CIMA to solve the mitigation limitation of ASan.
- CIMA: Countering Illegal Memory Accesses at runtime
- It is a *light-weight*, *efficient* and *proactive* mitigation strategy against memory-safety attacks.

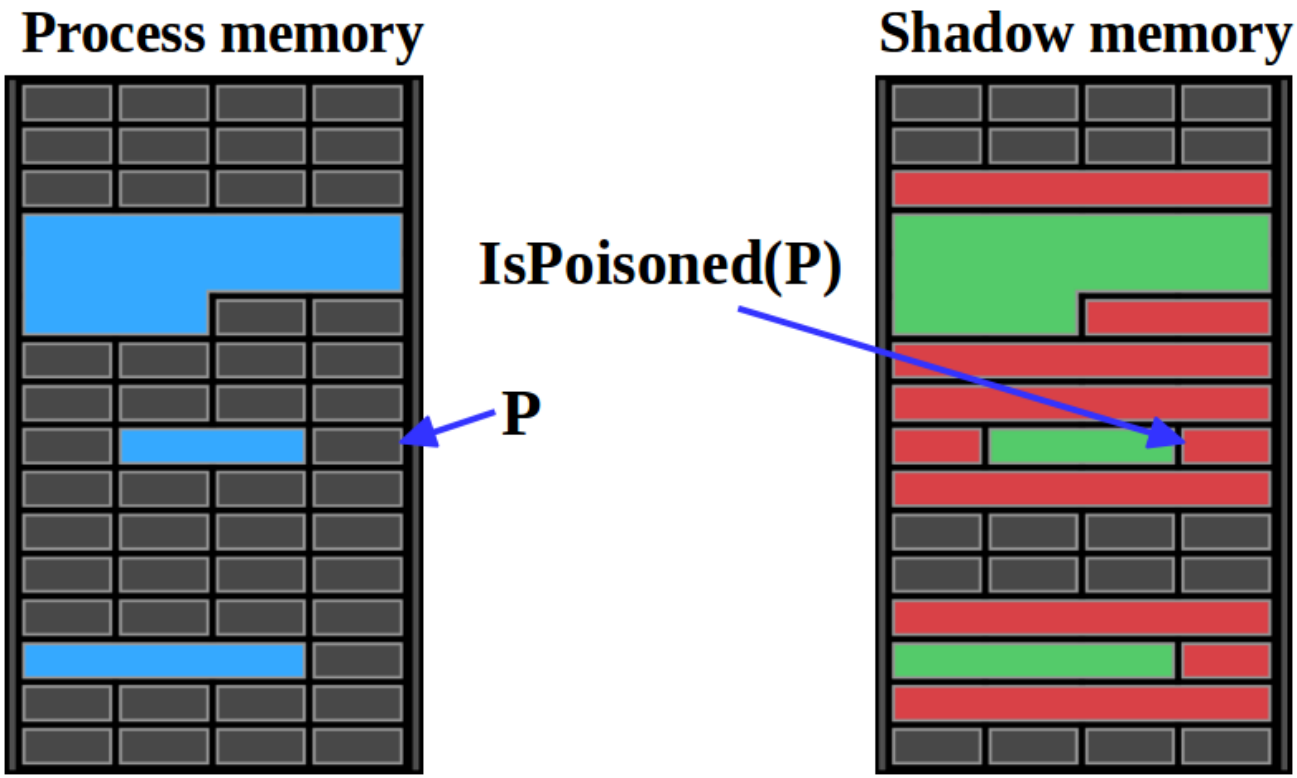
Approach of CIMA

- Based on *bypassing* illegal memory access instructions.

Approach of CIMA

- Based on *bypassing* illegal memory access instructions.

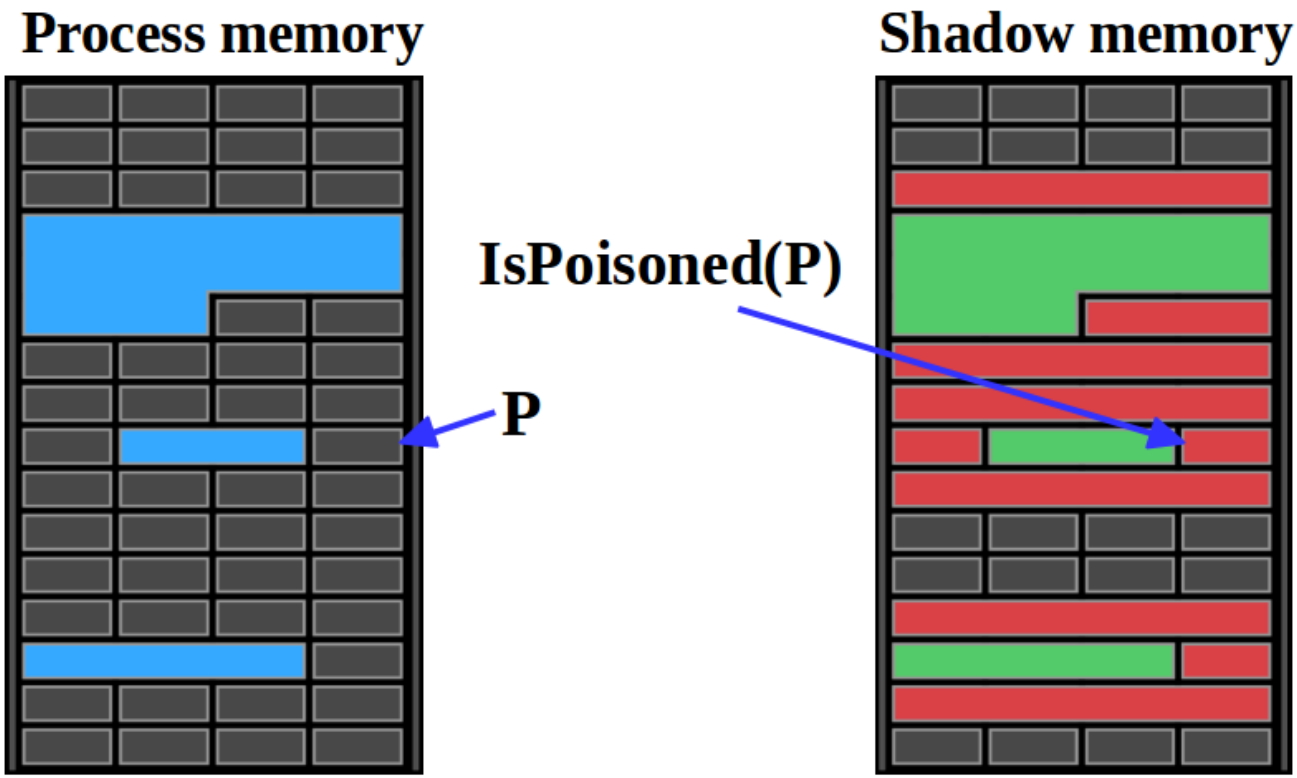
```
if (IsPoisoned(P))
    Crash;
*P = 0xd00;
```



Approach of CIMA

- Based on *bypassing* illegal memory access instructions.

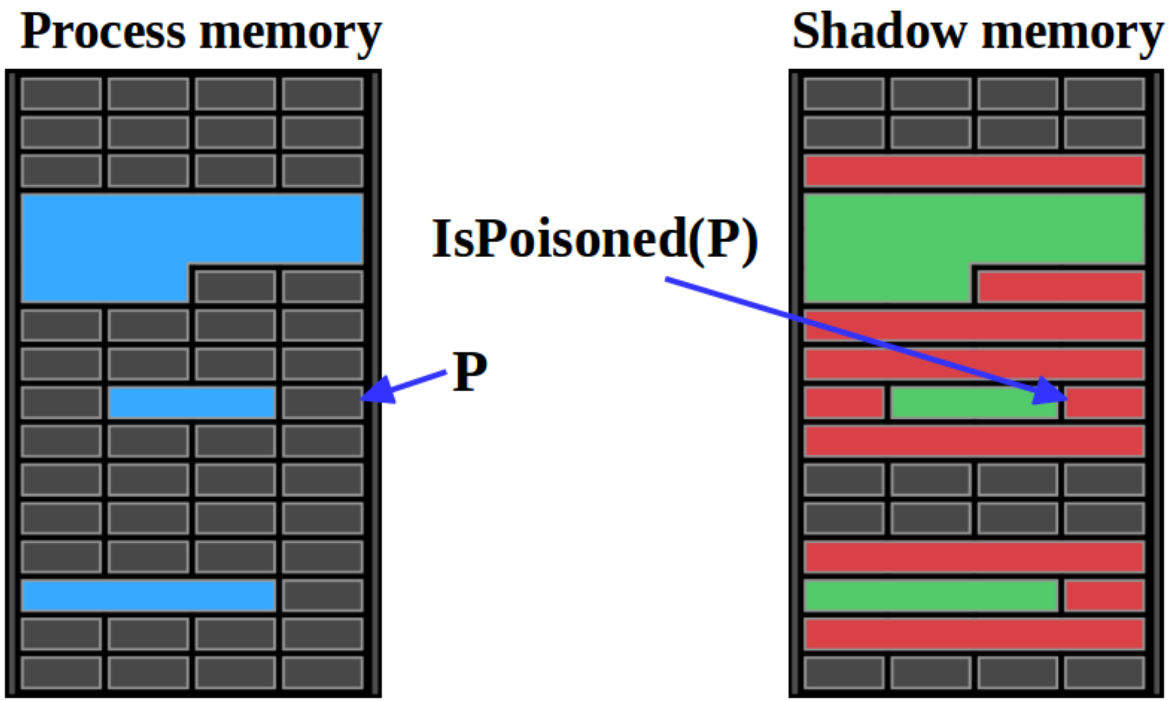
```
if (IsPoisoned(P))
    Crash;
*P = 0xd00;
```



Approach of CIMA

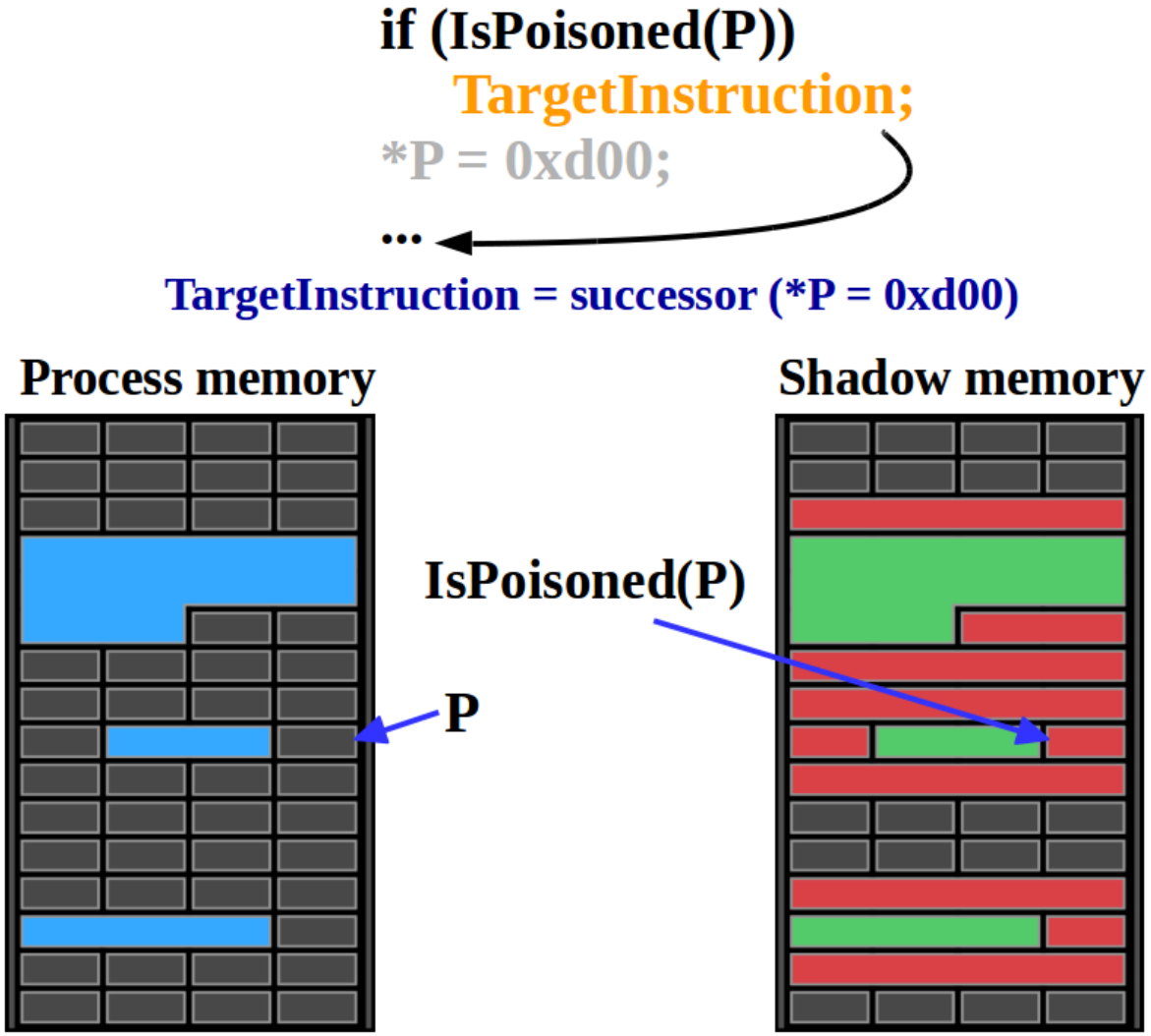
- Based on *bypassing* illegal memory access instructions.

```
if (IsPoisoned(P))
    TargetInstruction;
*P = 0xd00;
```



Approach of CIMA

- Based on *bypassing* illegal memory access instructions.



Outline

2FA: two-factor authentication

- 2FA[1] is a secure communication protocol based on two-factor authentication.
- Historical data from a server is used as a second factor, in addition to a secret key, to authenticate a server communicating with PLCs.
- Developed for a Metro Control system.
- Experimented on SecUTS testbed.
- Its overhead on PLCs is measured.
- No mitigation strategy involved.

[1] H. Guo, E. W. R. Tan, L. Zhou, Z. Zhao, X. Yu. 2FA Communication Protocol to Secure Metro Control Devices. In The IEEE Intelligent Transportation Systems Conference (ITSC). Auckland, New Zealand. 2019

Outline

LCDA: legacy-compliant data authentication

- LCDA[2] is a cryptographic authentication method developed for an ICS.
- Designed to verify authenticity of communication between PLCs in ICS.
- Symmetric and asymmetric signature algorithms were benchmarked for a variety of hardware platforms.
- Experimented on SWaT testbed.
- Its overhead on PLCs is measured.
- No mitigation strategy involved.

[2] J. H. Castellanos, D. Antonioli, N. O. Tippenhauer, M. Ochoa. Legacy-Compliant Data Authentication for Industrial Control System Traffic. In Applied Cryptography and Network Security. 2017

Outline

Outline

Experimental design

- We designed our experiments based on two ICS testbeds:
  - Secure Water Treatment (SWaT)[3] Testbed
  - Secure Urban Transportation System (SecUTS)

[3] https://itrust.sutd.edu.sg/testbeds/secure-water-treatment-swat/

Overview of SWaT

- **SWaT**: a secure water treatment plant at SUTD.

Overview of SWaT

- **SWaT**: a secure water treatment plant at SUTD.

Figure 8: SWaT

Architecture of SWaT

- Has 6 distinct processes controlled by 6 PLCs.

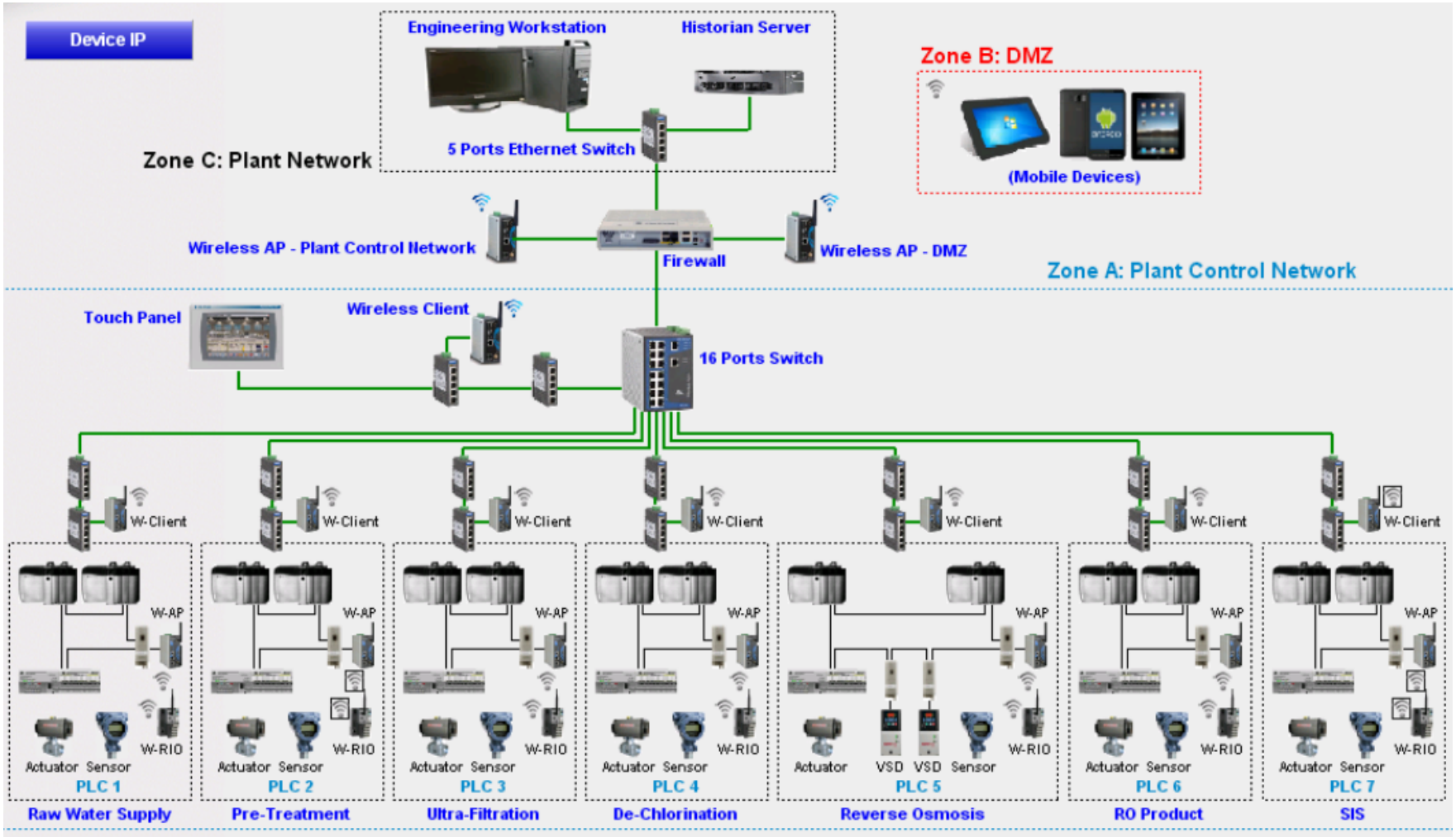


Figure 9: SWaT architecture

The purification process of SWaT

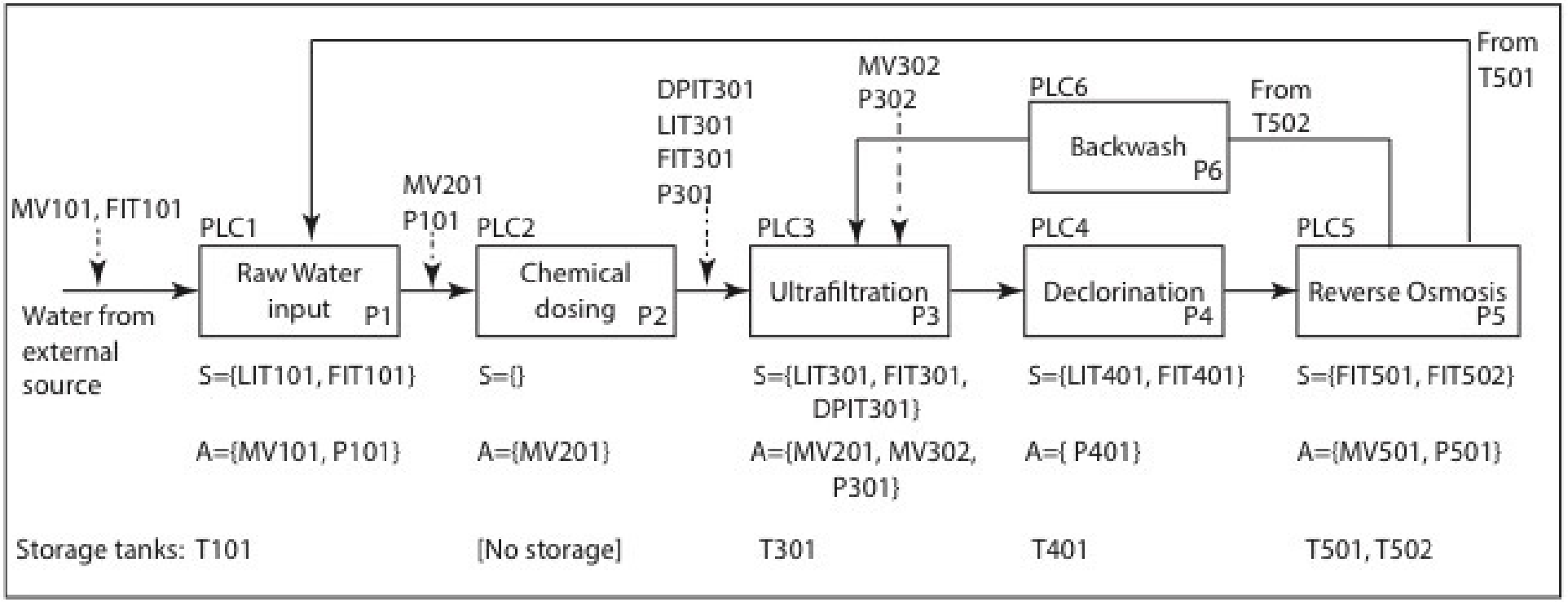


Figure 10: A six sage water purification process

Water inflow process (P1)

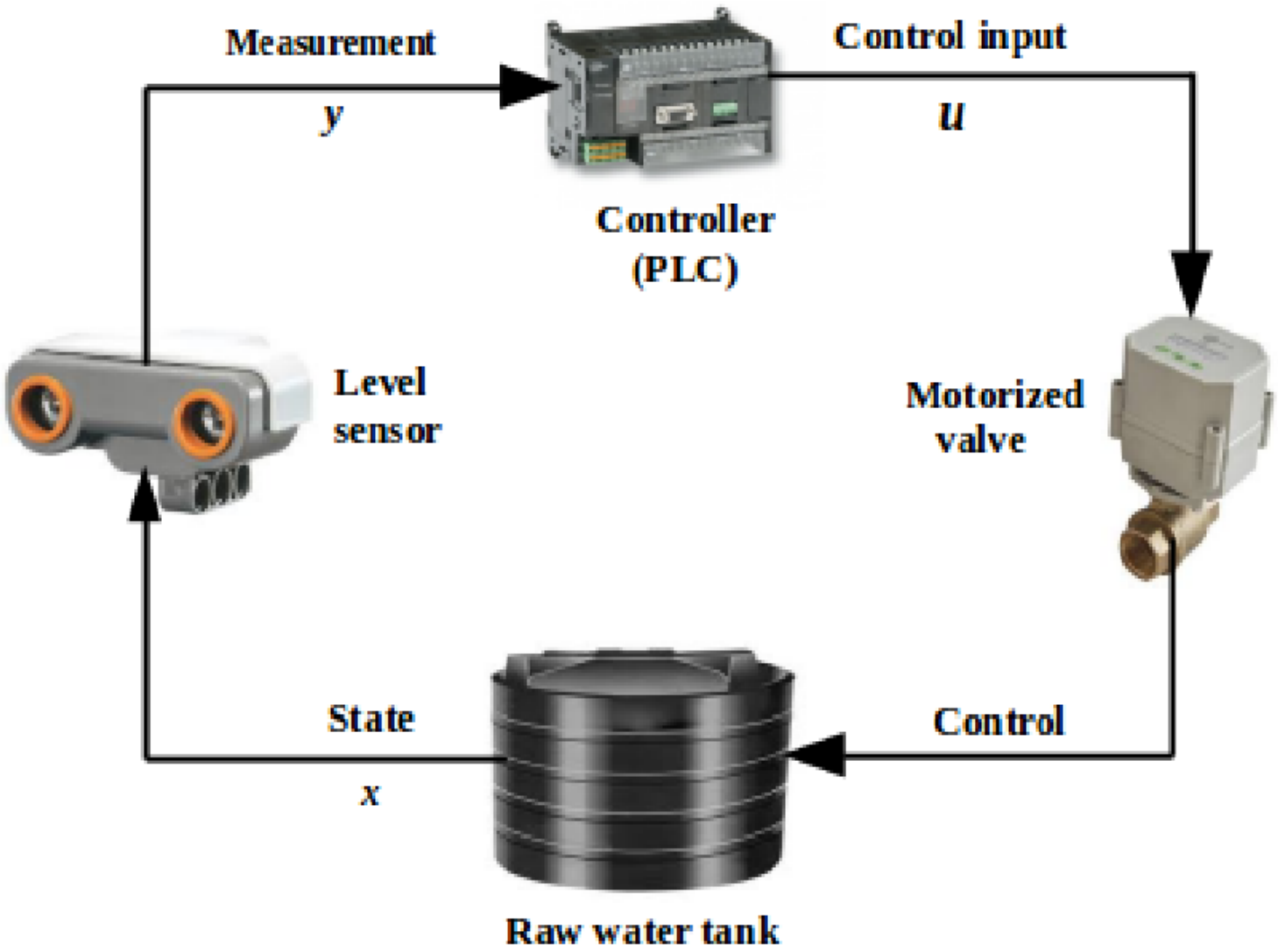


Figure 11: A raw water inflow control

The Open-SWaT design

- The PLCs in SWaT and SecUTS are closed source.
- Thus, we designed Open-SWaT and Open-SecUTS testbeds by mimicking SWaT and SecUTS, respectively.

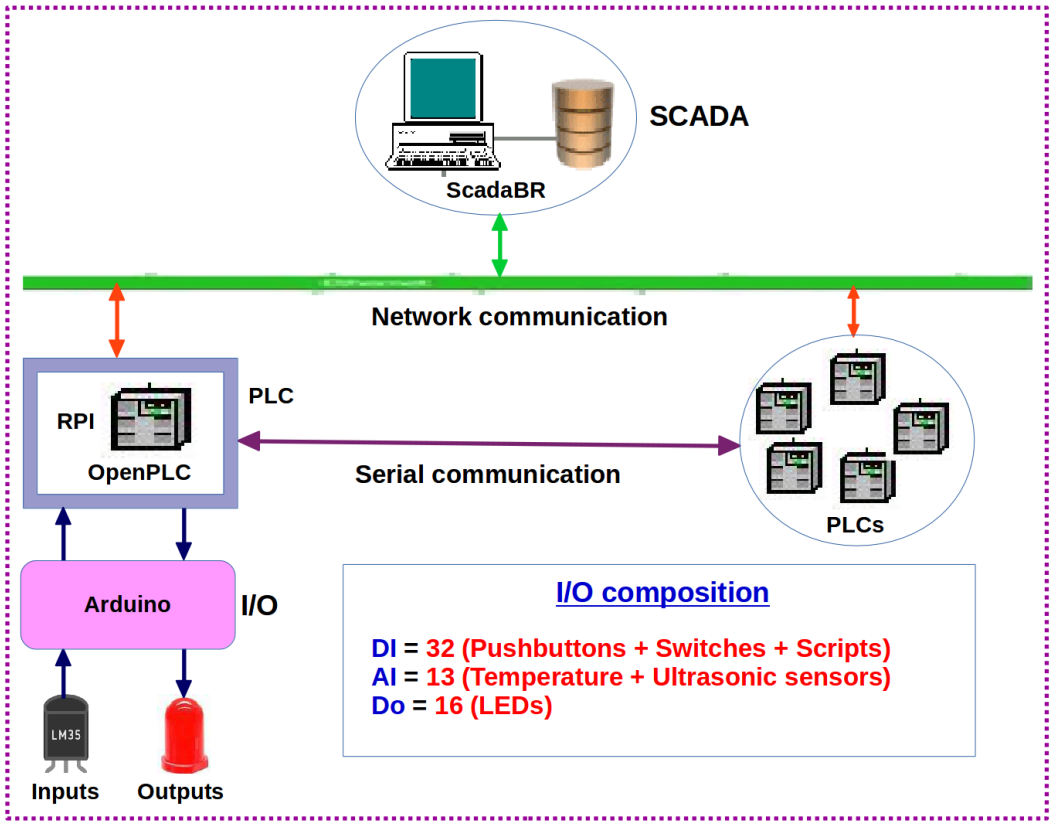


Figure 13: Architecture of Open-SWaT

Open-SWaT details

- Detailed profiles of the testbed.
  - Hosted on Raspberry PI
  - *Processor speed*: 200MHz
  - *Controller*: OpenPLC
  - *Cycle time*: 10MS
  - *PLC program complexity*: 129 instructions
  - *Number of connections*: 7
  - *Communication frequency*: 10MS
  - *I/O terminal*: Arduino
  - *Digital inputs*: 32
  - *Digital outputs*: 16
  - *Analog inputs*: 13
  - *SCADA system*: ScadaBR

SecUTS

- The Secure Urban Transportation System (SecUTS) is an ICS testbed designed to monitor a Metro SCADA system.
- It comprises an Integrated Supervisory Control and a train signaling system.
- Consists of:
  - 6 digital inputs (emergency and control buttons)
  - 9 digital outputs (tunnel and station lightings, ventilators and alarms)
  - The scan cycle is 30ms.

Outline

Evaluating the tools

- We evaluate the tools along three directions:
  - Security guarantee (omited for this presentation!)
  - Efficiency
  - Availability/Resilience

ASan: Efficiency

Table 1: Memory-safety overheads of ASan (Open-SWaT)

Operations	Number of cycles	Network devices	CPU speed (in MHz)	Original (T_s)		ASan ($\hat{T}_s$)			
				Mean (in µs)	Max (in µs)	Mean (in µs)	Max (in µs)	MSO (in µs)	MSO (in %)
Input scan	50000	6	200	59.38	788.12	118.44	1132.32	59.09	99.46
Program execution	50000	6	200	69.09	611.82	115.88	720.36	46.79	67.72
Output update	50000	6	200	145.01	981.09	185.37	1125.45	40.36	27.83
Full scan time	50000	6	200	273.48	2381.03	419.69	2978.13	146.21	53.46

ASan: Efficiency

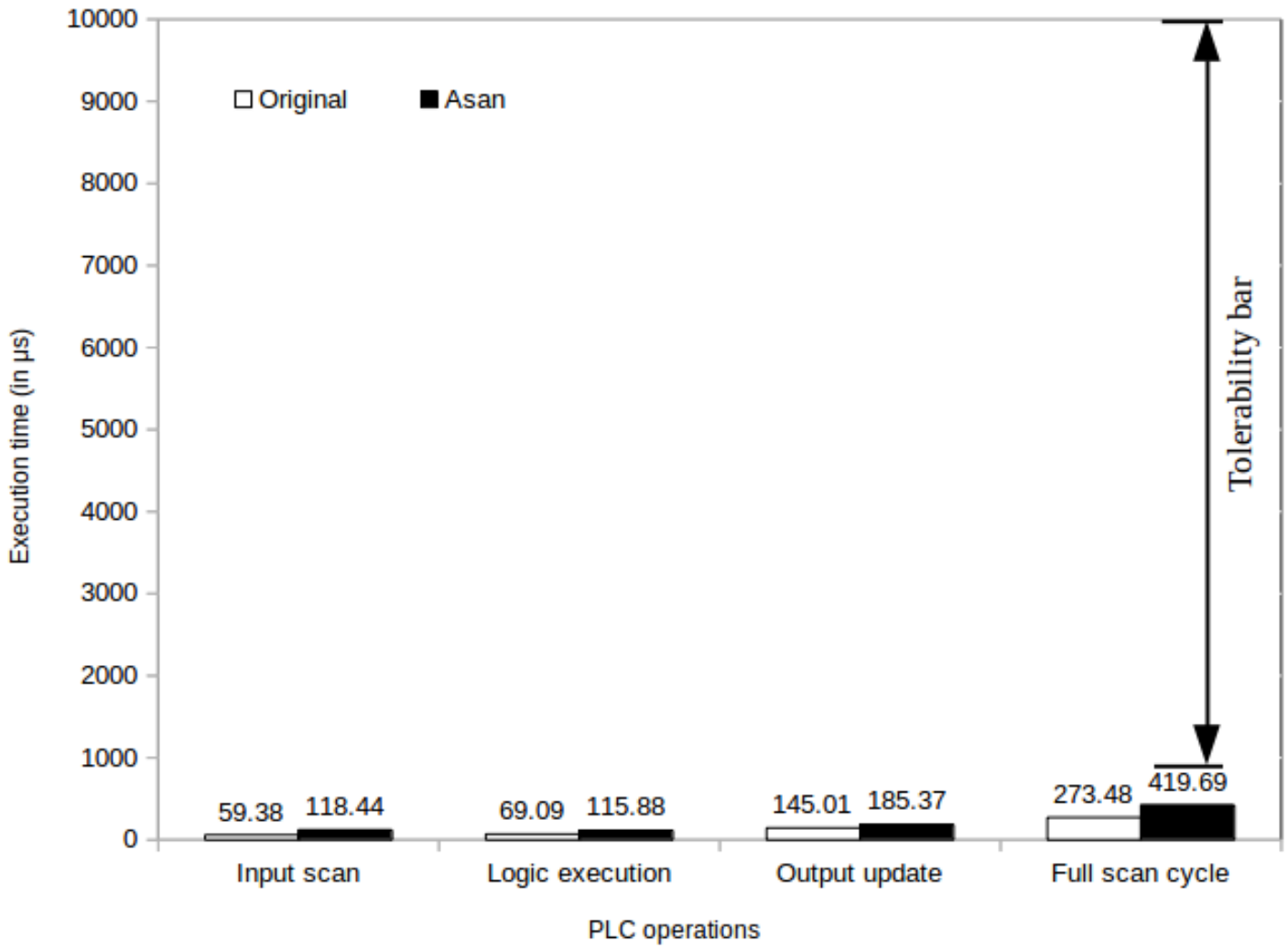


Figure 14: ASan average-case scan time

ASan: Efficiency

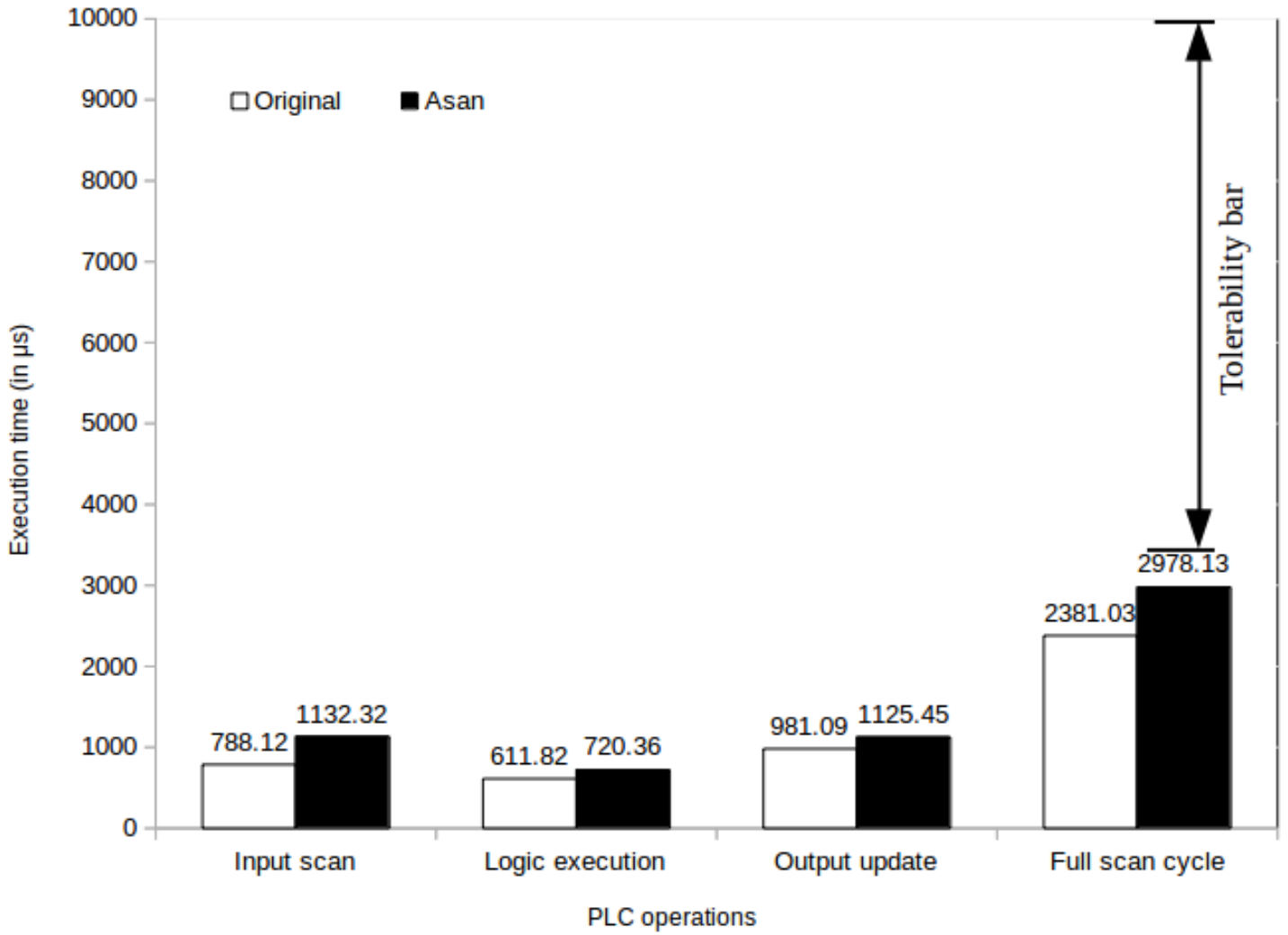


Figure 15: ASan worst-case scan time

ASan: Availability/Resilience

- ASan doesn't ensure PSR in the presence of memory-safety attacks, because,
  - It simply aborts the program when a memory-safety attack is detected, hence, $\delta = \infty$.
- We proposed CIMA to overcome this problem.

CIMA: Efficiency

Table 1: Memory-safety overheads ASan + CIMA (Open-SWaT)

Operations	Number of cycles	Network devices	CPU speed (in MHz)	Original (T_s)		ASan + CIMA ($\hat{T}_s'$)			
				Mean (in µs)	Max (in µs)	Mean (in µs)	Max (in µs)	MSO (in µs)	MSO (in %)
Input scan	50000	6	200	59.38	788.12	122.86	1151.35	63.48	106.9
Program execution	50000	6	200	69.09	611.82	118.97	802.18	49.88	72.2
Output update	50000	6	200	145.01	981.09	199.89	1213.62	54.88	37.85
Full scan time	50000	6	200	273.48	2381.03	441.72	3167.15	168.24	61.52

CIMA: Efficiency

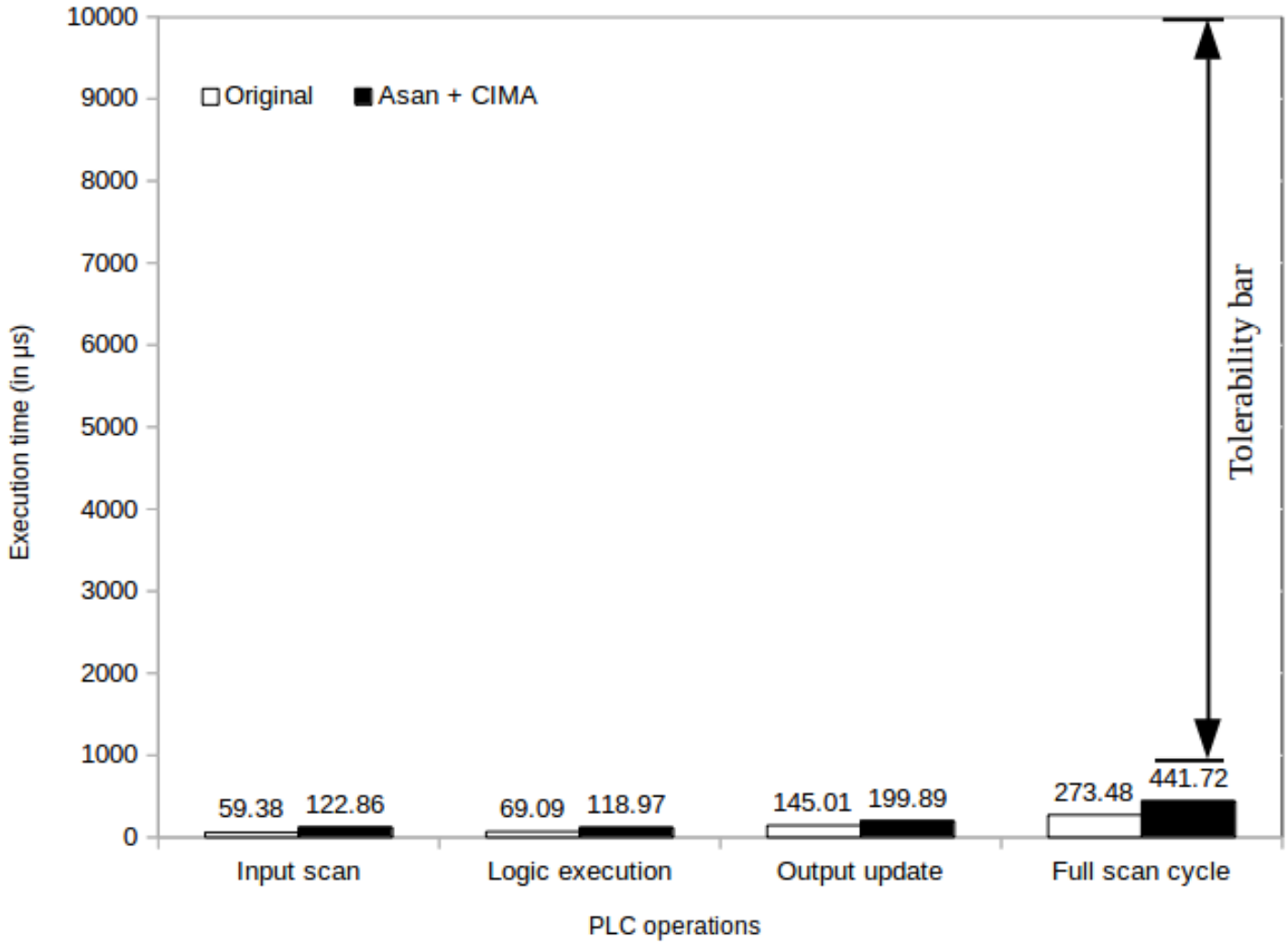


Figure 16: ASan+CIMA average-case scan time

CIMA: Efficiency

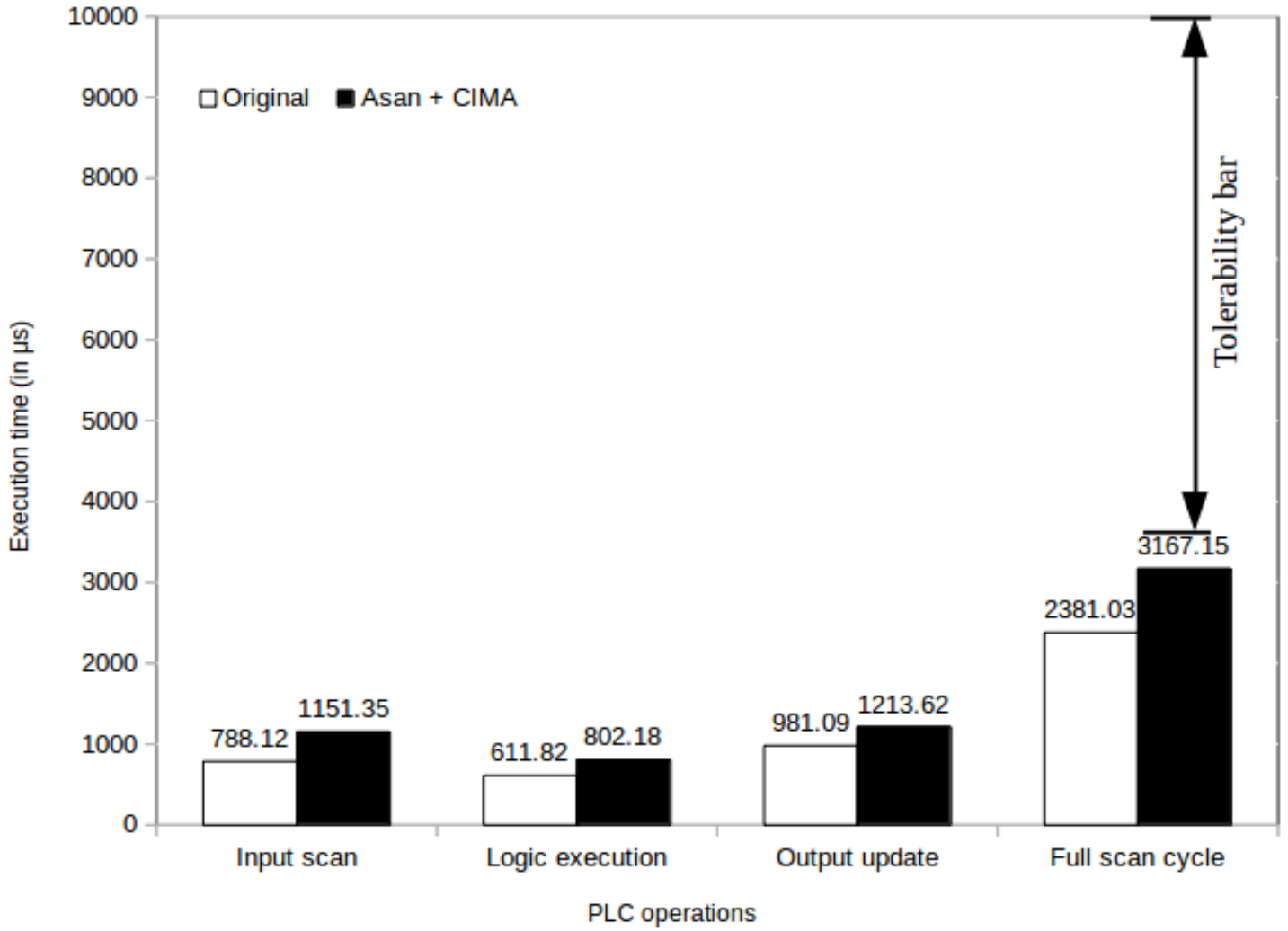


Figure 17: ASan+CIMA worst-case scan time

Breakdown of the overhead

- ASan: 53.46%
- CIMA: 8.06%
- Overall overhead: 61.52%

CIMA: Availability/Resilience

- CIMA ensures physical-state resiliency of the ICS even under the presence of memory-safety attacks, because,
  1. The overall overhead is tolerable, i.e. $T_s' < T_c$.
  2. CIMA doesn't abort the victim program.
  3. Thus, $\delta = 0$!

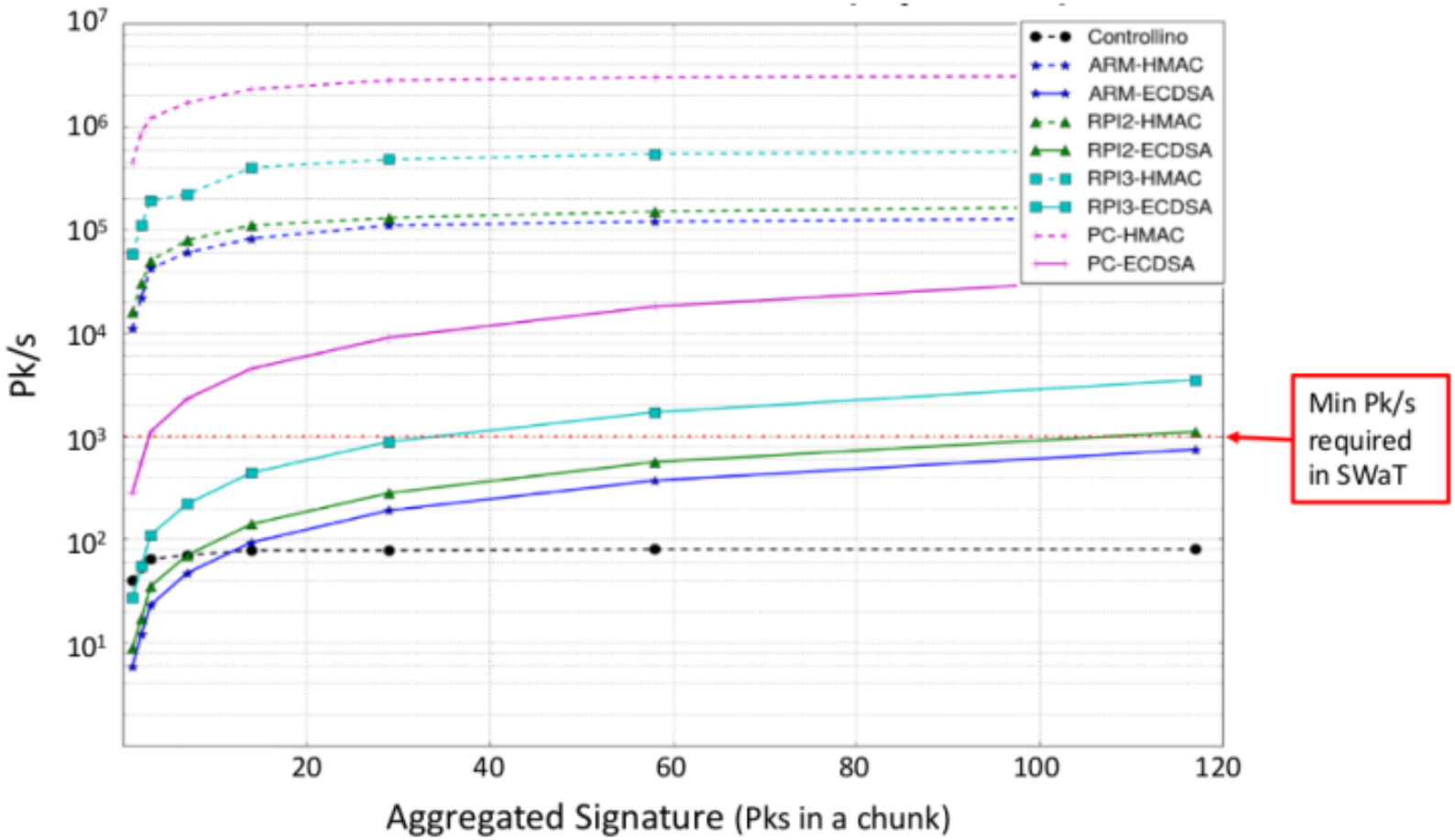
2FA: Efficiency

- 2FA introduced an overhead ranging from 18 to 26ms when tested on different number of historical data.
- It is tolerable since the T_c of SecUTS is 30ms, i.e. $T_s' < T_c$.

2FA: Availability/Resilience

- $T_s' < T_c$, i.e. $\delta = 0$.
- 2FA doesn't restart or abor the system, thus $\delta = 0$ again.
- Therefore, 2FA doesn't violate the PSR.

LCDA: Efficiency



- T'_s is tolerable in some platforms and not in others.

J. H. Castellanos, D. Antonioli, N. O. Tippenhauer, M. Ochoa. Legacy-Compliant Data Authentication for Industrial Control System Traffic. In

LCDA: Availability/Resilience

- Although LCDA doesn't render system restart/abort, its overhead in some platforms could violate the PSR.
- Therefore, its efficiency and resilience is dependant on the platform used.

Conclusion

- In ICS, the hard real-time and availability requirements are equally critical as security.
- We tried to formally model this critical requirements.
- We are also intended to use these requirements as safety properties in ICS.
- We hope other researchers will also improve and use our models in the future.