Sancus 2.0: A Security Architecture for Low-Cost Networked Embedded Devices

Frank Piessens,
imec-DistriNet, KU Leuven

Introduction

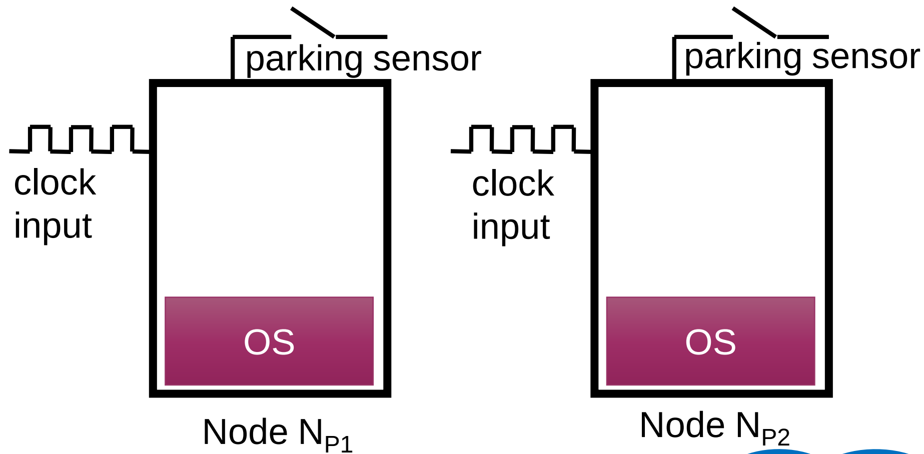
- Long-term objective:

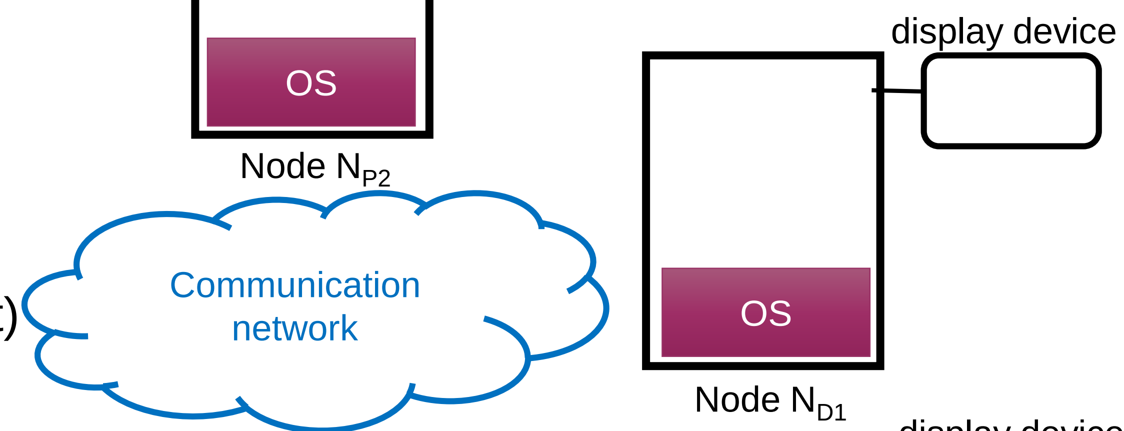
  Secure open software application platforms for distributed IT infrastructure

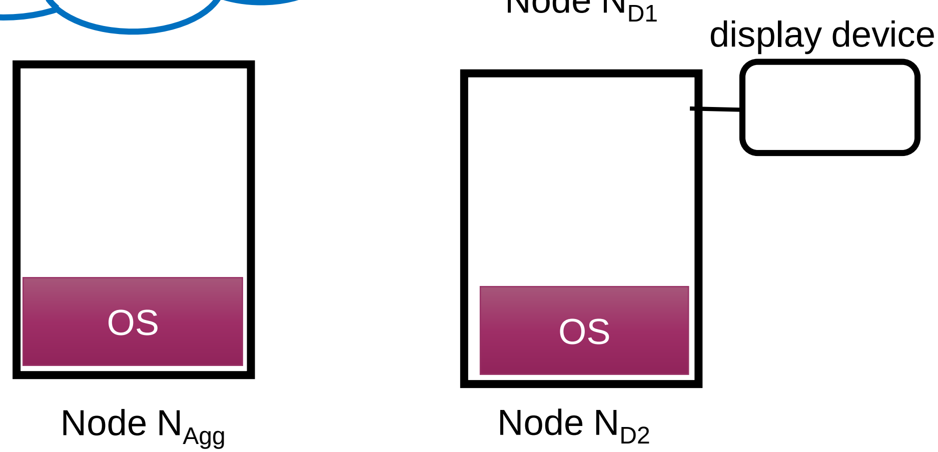
  - Distributed IT infrastructure =
    - Networked hardware and corresponding system software
    - Examples: sensor networks, smart cities, cloud platforms, …
  - Open software application platform =
    - Multiple, mutually distrusting parties install and run applications on such shared infrastructure
  - Secure = ?
    - Software provider viewpoint:
      - My application should run reliably under realistic attacker models

DistriNet iMinds KU LEUVEN

Example: Smart parking system



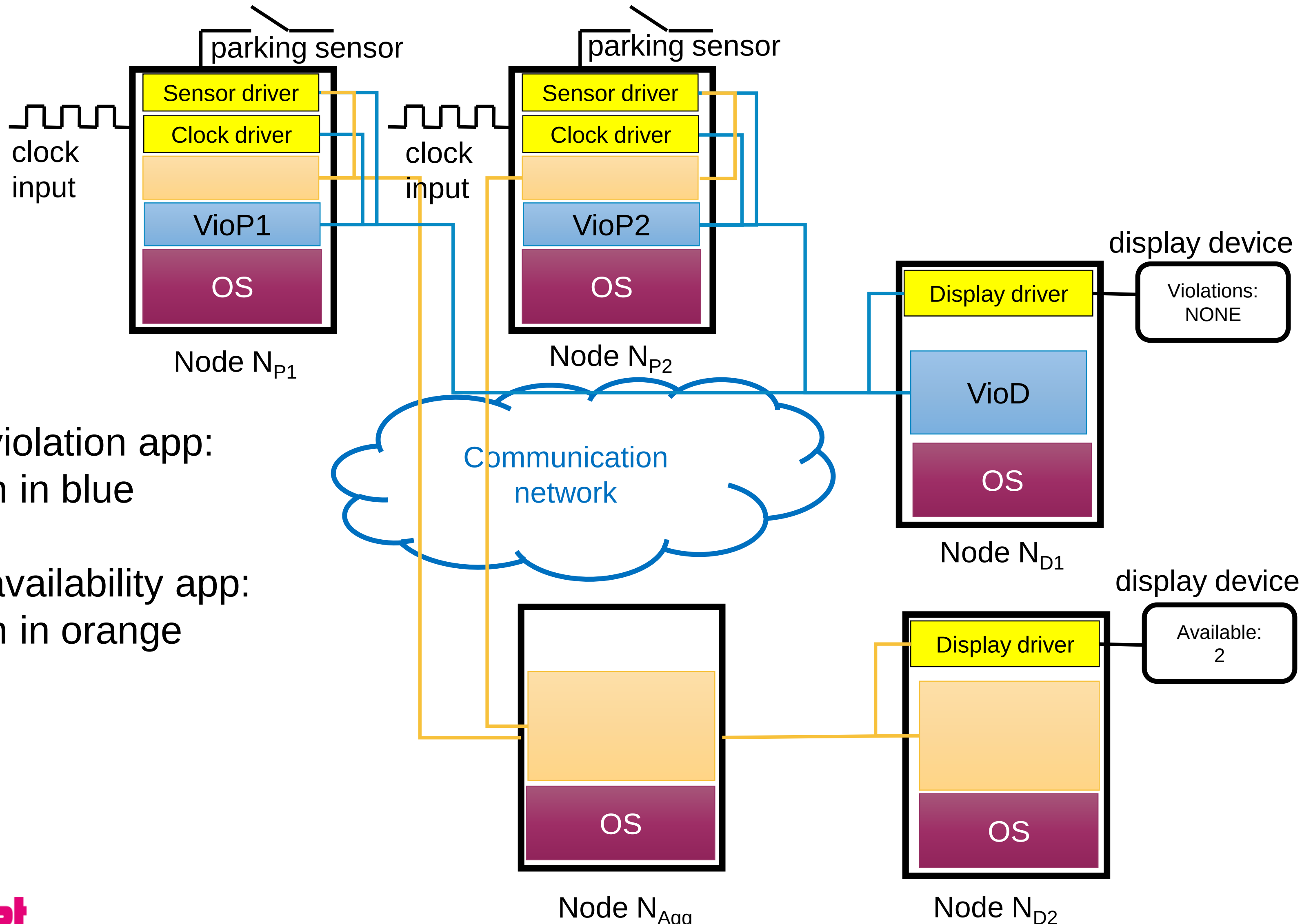




Network of:
- Sensor nodes (1 per spot)
- Display devices
- Aggregation nodes
- …

Possible applications:
- Parking rule violation detection
- Parking guidance
- Availability monitoring
- …

… running some applications



Parking violation app:

- Shown in blue

Parking availability app:

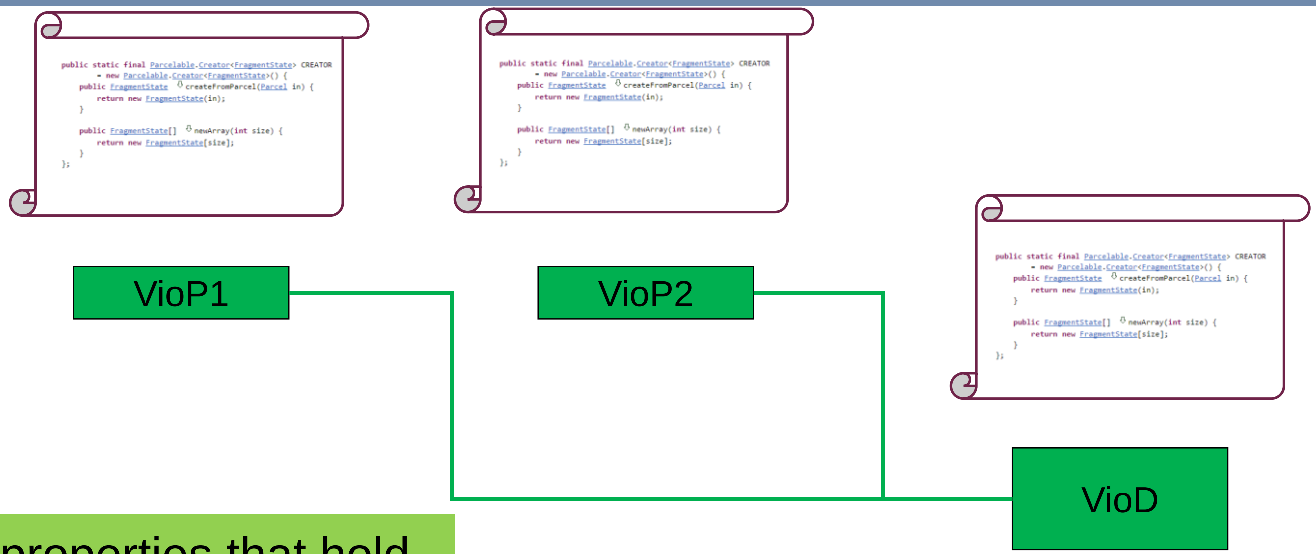
- Shown in orange

DistriNet iMinds KU LEUVEN

Key objective

- Can we build **secure** infrastructure (from the software provider point-of-view)?
  - What are realistic attacker models?
    - Software attacks
      - Other applications
      - System software
    - Network attacks (Dolev-Yao for crypto)
    - Hardware / physical attacks
  - What security properties?
    - Integrity of application state
    - Confidentiality of application state
    - Availability

Defining infrastructure security

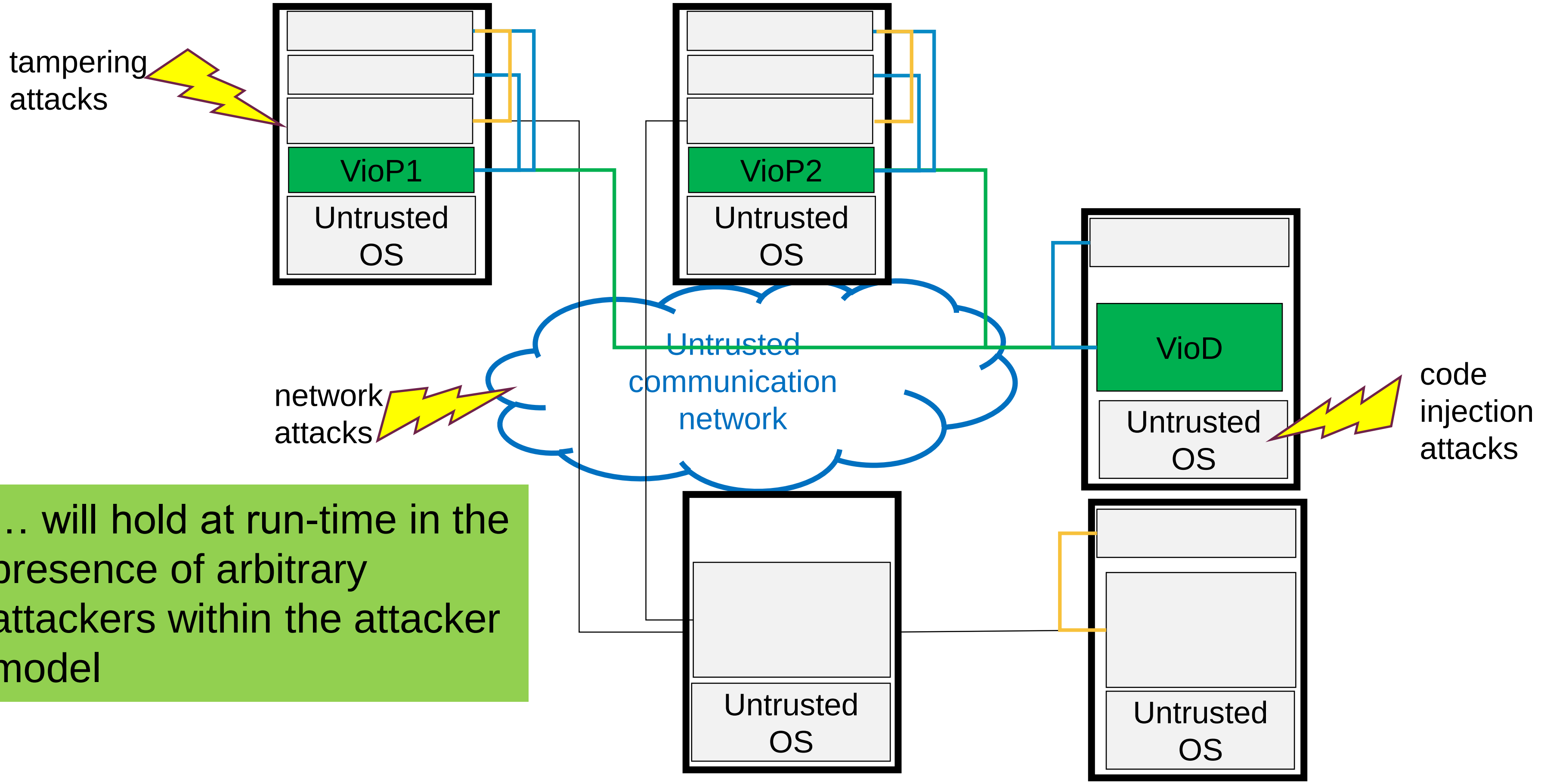


The security properties that hold at the level of abstraction of the source code …

(E.g. "*The display shows a parking violation on spot X only if a car has entered that spot more than 2 hours ago and has since then not left the spot*")

DistriNet iMinds KU LEUVEN

Defining infrastructure security



… will hold at run-time in the presence of arbitrary attackers within the attacker model

Summary

- Infrastructure security =
  "Preserving application security at run time"
- This is parametric in:
  - Attacker model:
    - We will focus in this talk on software (including system software) and network attacks
  - Relevant application security properties
    - We will focus in this talk on integrity:
      - Whatever the application does can be expected from the source code
      - But it might not do anything (availability) and might leak arbitrary application information (confidentiality)

DistriNet
iMinds KU LEUVEN

Outline of this talk

- What hardware/software support is required to realize this notion of "infrastructure security" on small embedded processors?
- Sancus 2.0:
  - A small microprocessor with support for
    - Protected application modules
    - Remote attestation, authentication and secure communication with these modules
    - Publications: Usenix Security 2013 and ACM TOPS 2017
  - A software infrastructure ("operating system") to deploy and run distributed applications on these processors
    - Publication currently under submission

DistriNet iMinds KU LEUVEN

Sancus: System model

Infrastructure provider

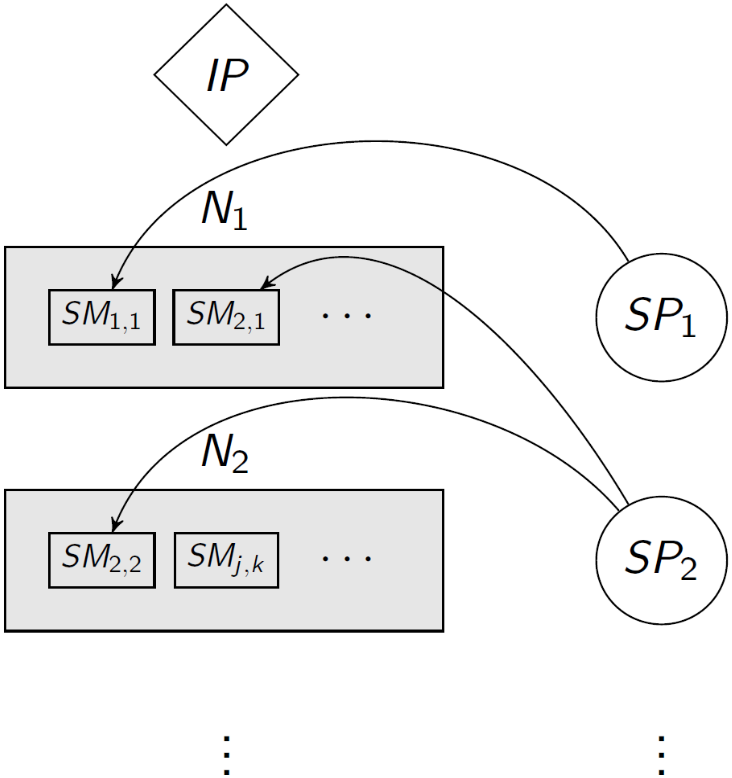
IP owns and administers nodes N_i

Software providers

SP_j wants to use the infrastructure

Software modules

$SM_{j,k}$ is deployed by SP_j on N_i

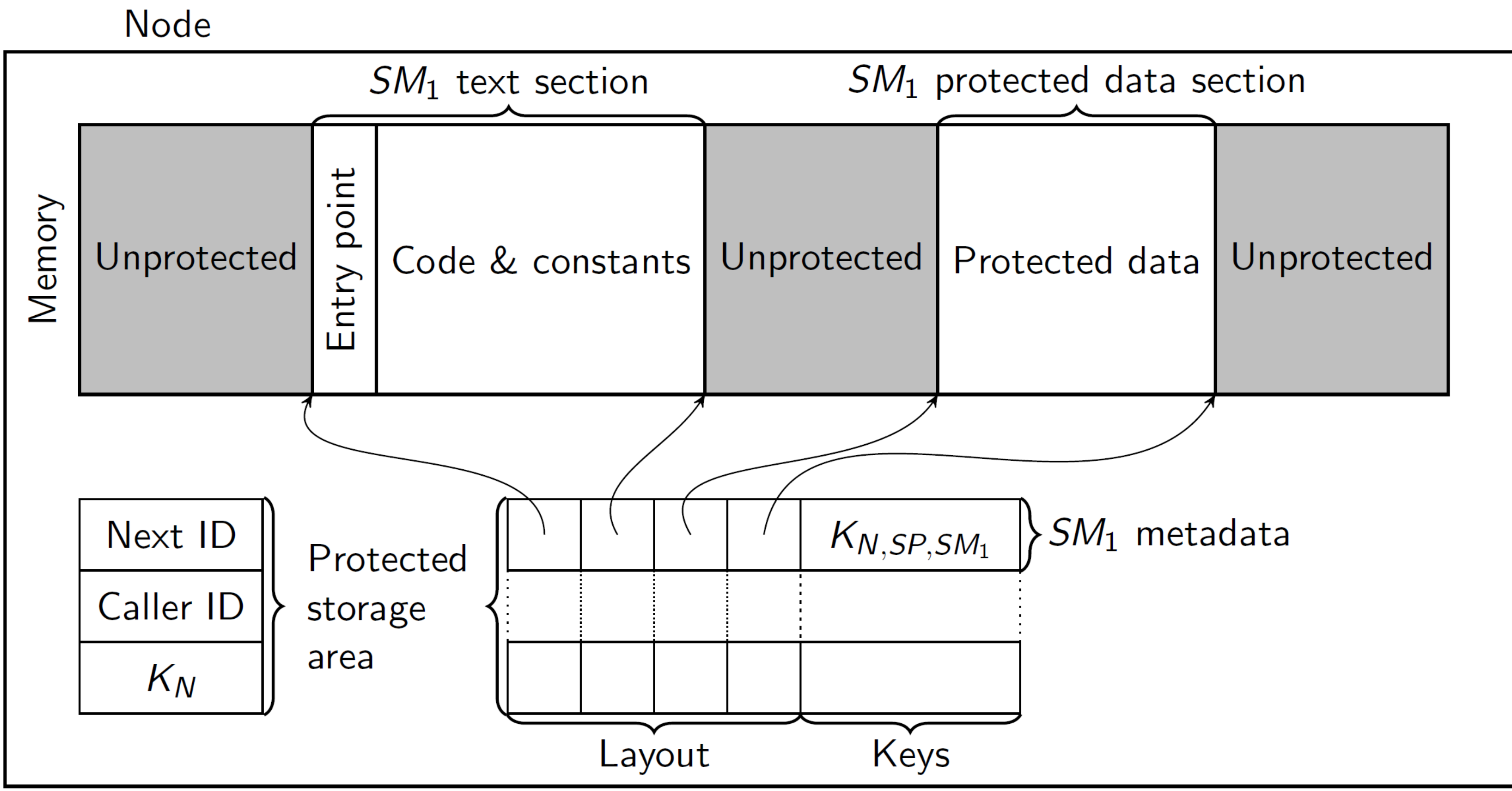


Isolating software modules

Protected software modules

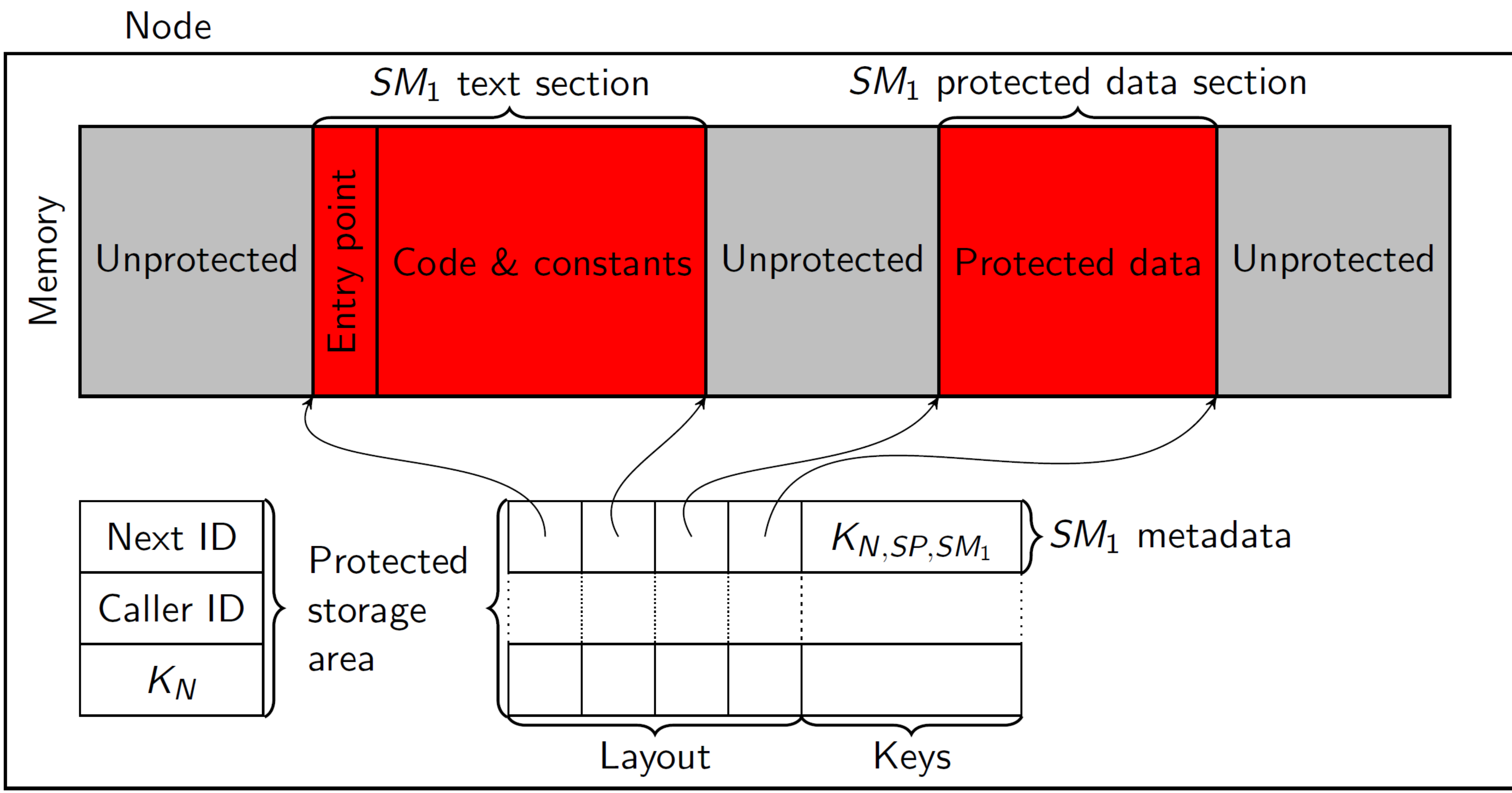
- Standard SW modules, defining memory sections
  - Text section
    - Code and constants
  - Protected data section
    - Runtime data that needs to be protected
  - (Optional unprotected sections)

Node with one software module loaded



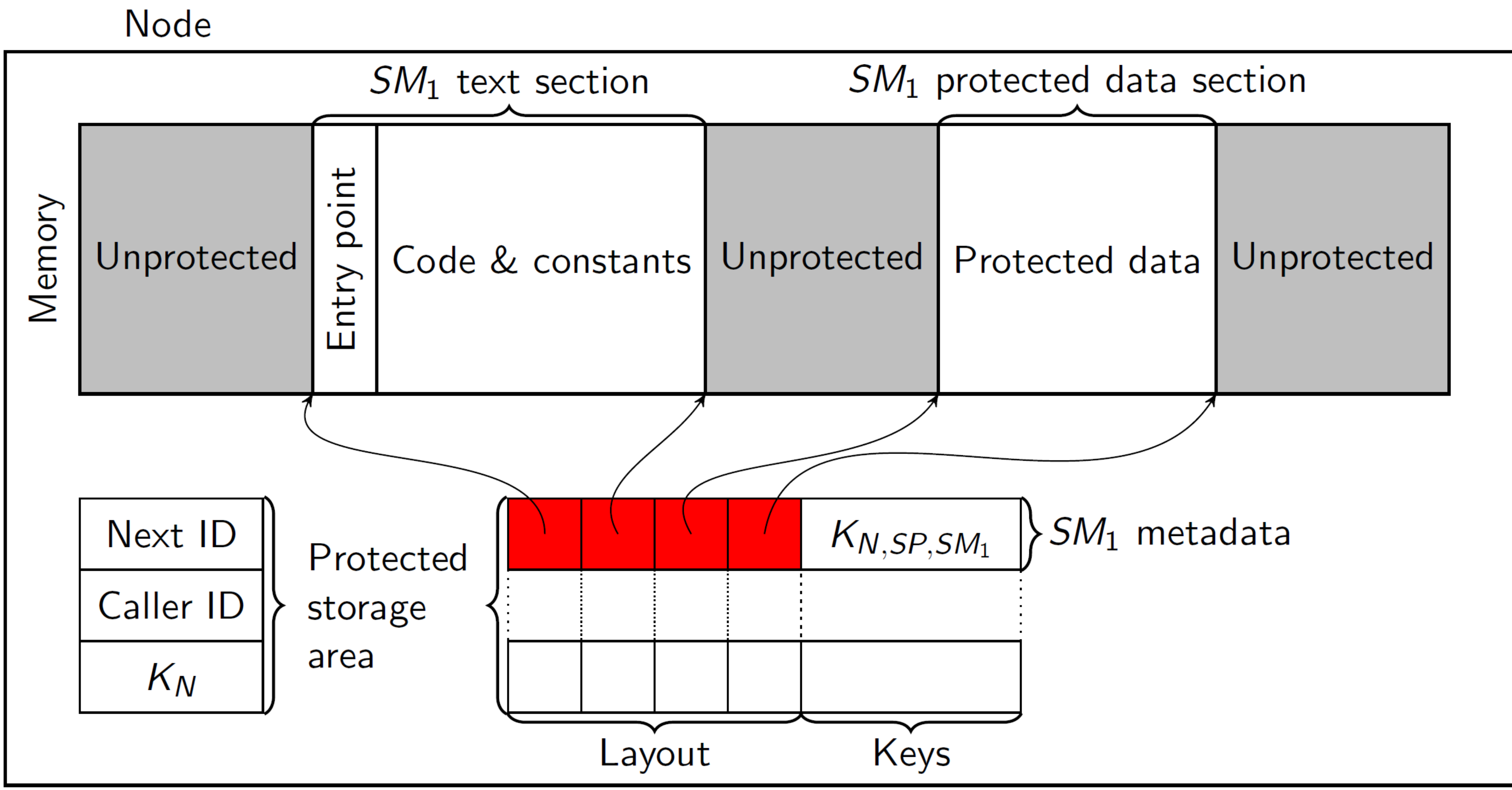
Node with one software module loaded

Text and data sections



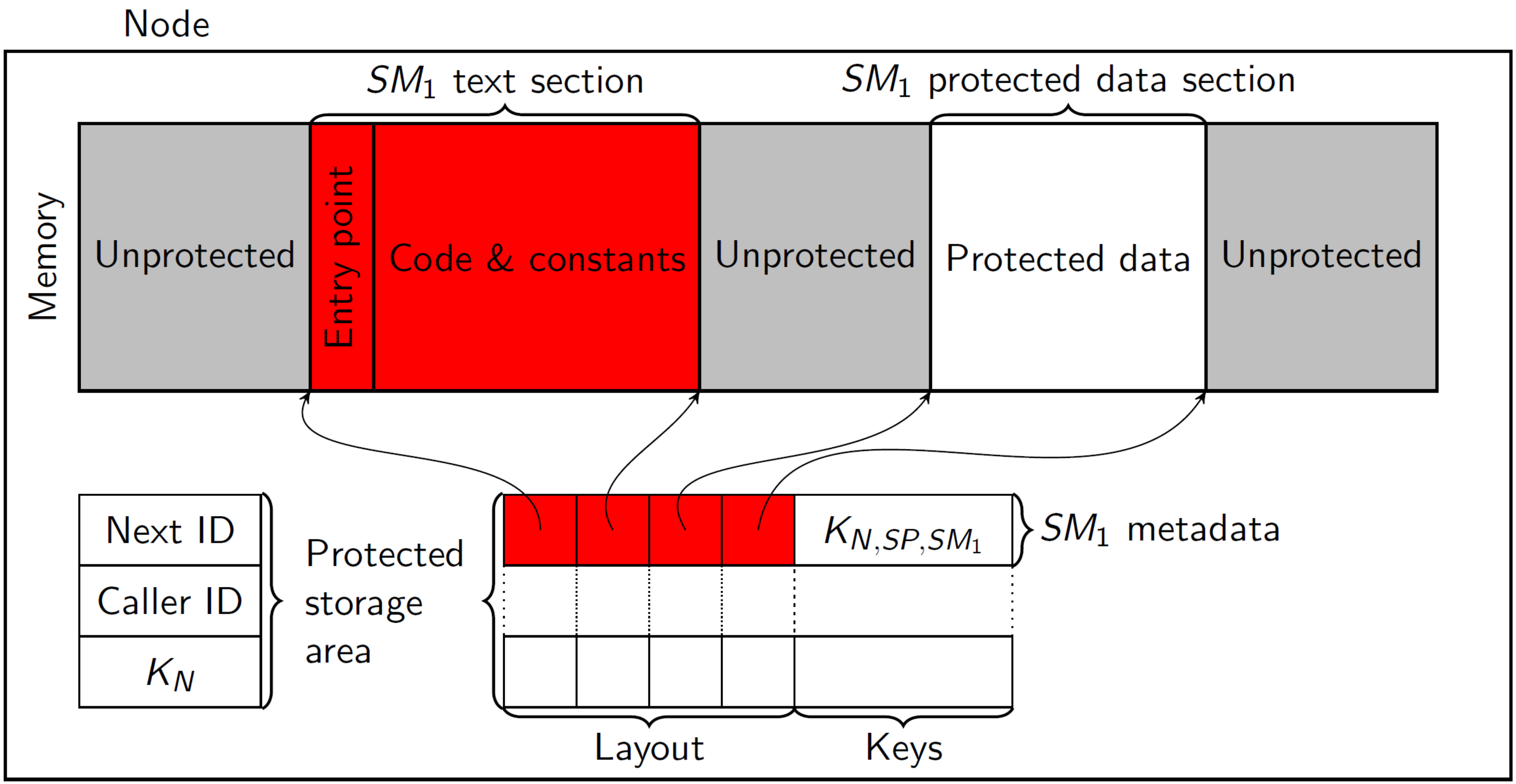
Node with one software module loaded

Module layout



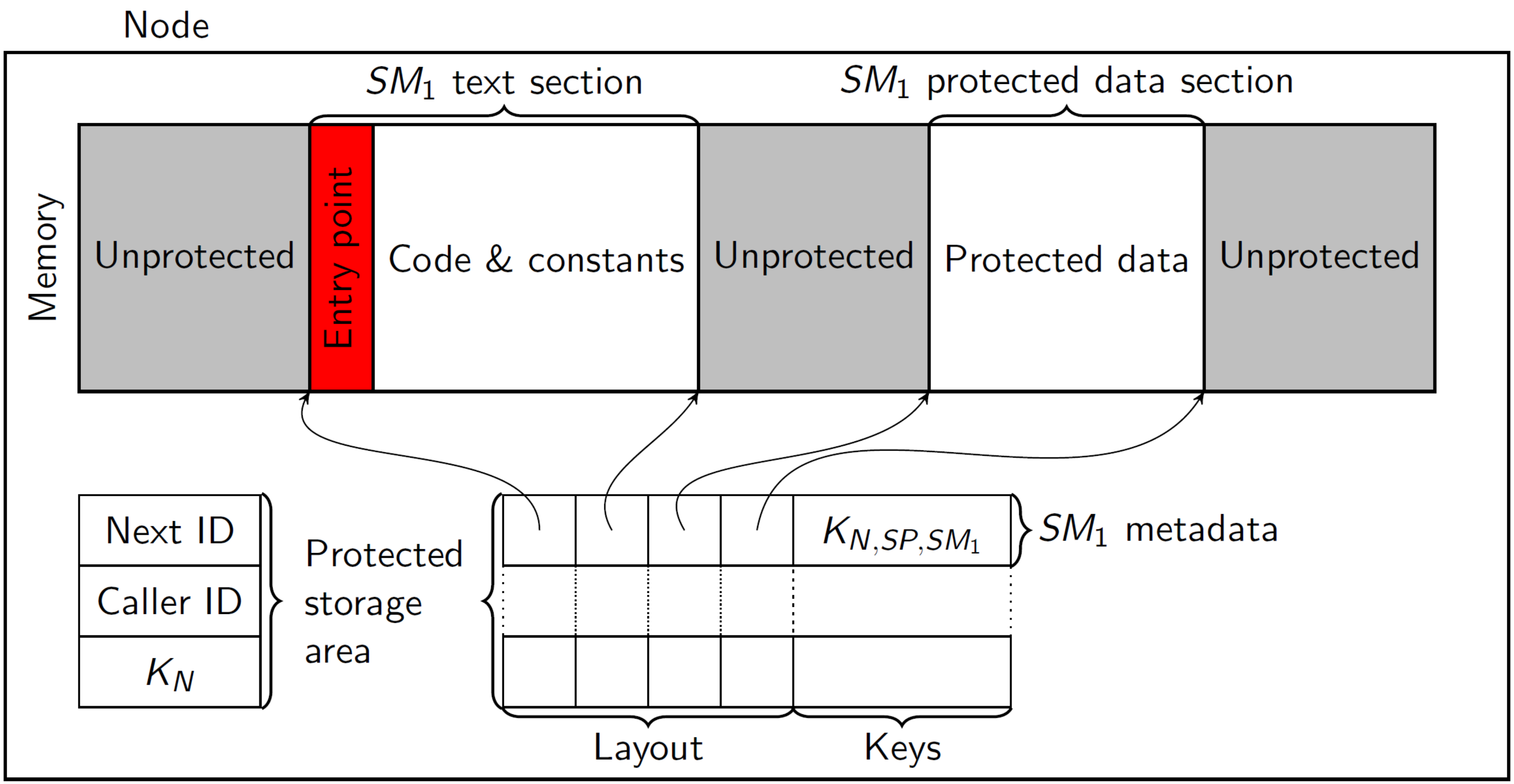
Node with one software module loaded

Module identity



Node with one software module loaded

Module entry point



Isolation on a node

- By program-counter based access control:

From/to	Entry	Text	Data	Unprotected
Entry	`r-x`	`r-x`	`rw-`	`rwx`
Text	`r-x`	`r-x`	`rw-`	`rwx`
Other	`--x`	`---`	`---`	`rwx`

- Enabled / disabled with new instructions
  - `protect` *layout*, *SP*

    Enables isolation at *layout*
  - `unprotect`

    Disables isolation of current SM

DistriNet iMinds KU LEUVEN

Key management

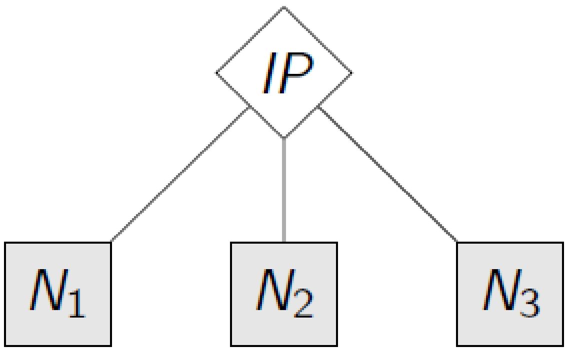
Key derivation scheme allowing both Sancus and *SP*'s to get the same key

Infrastructure provider is trusted party

Able to derive all keys

Every node N stores a key K_N

Generated at random



Key derivation scheme allowing both Sancus and *SP*'s to get the same key

Infrastructure provider is trusted party

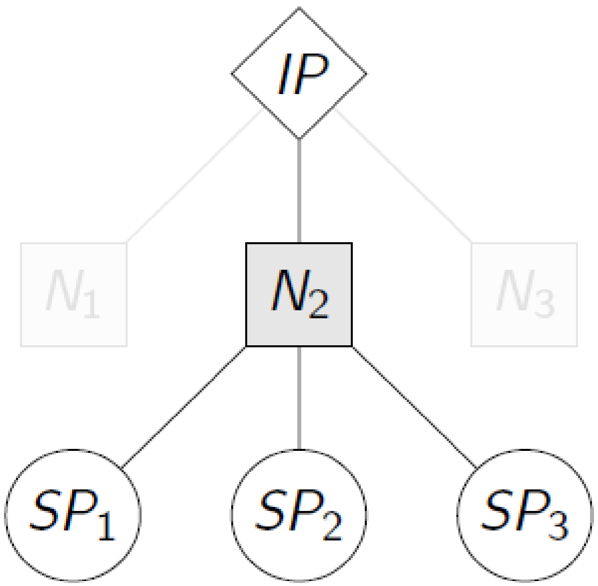
Able to derive all keys

Every node N stores a key K_N

Generated at random

Derived key based on *SP* ID

$K_{SP} = kdf(K_N, SP)$



Key derivation scheme allowing both Sancus and *SP*'s to get the same key

Infrastructure provider is trusted party

Able to derive all keys

Every node N stores a key K_N

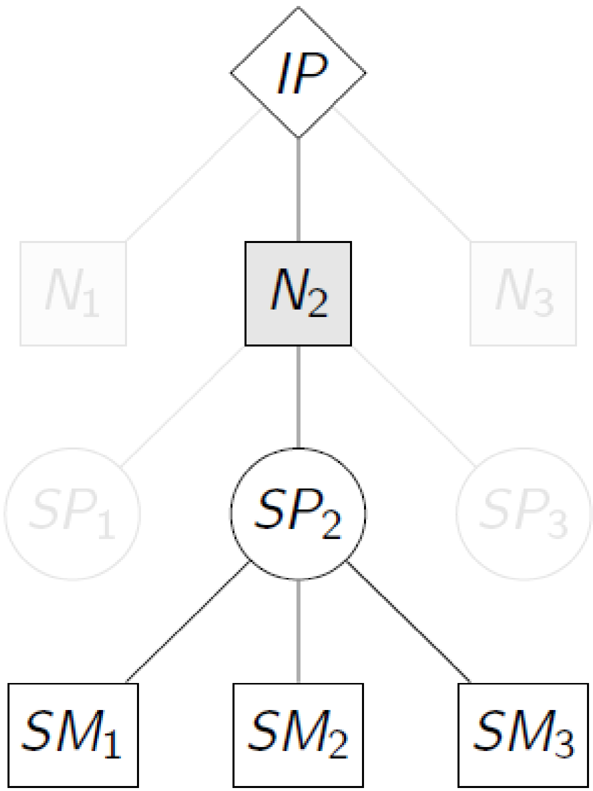
Generated at random

Derived key based on *SP* ID

$K_{SP} = kdf(K_N, SP)$

Derived key based on *SM* identity

$K_{SM} = kdf(K_{SP}, SM)$

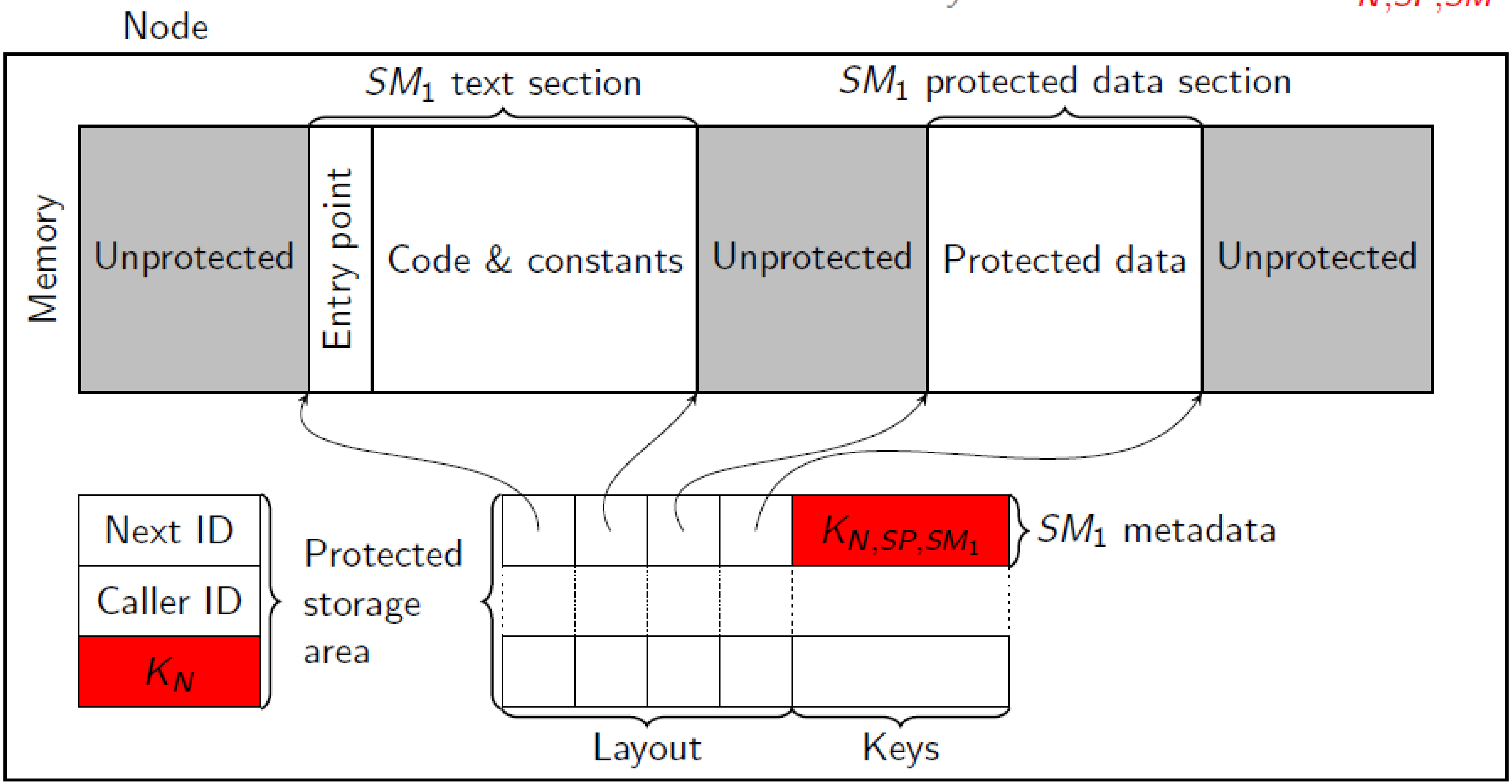


Node with one software module loaded

Module keys

`protect` *layout*, *SP*

Enables isolation at *layout* and calculates $K_{N,SP,SM}$



Remote attestation and secure communication

Attestation and communication

Ability to use $K_{N,SP,SM}$ proves the integrity and isolation of SM deployed by SP on N

Only N and SP can calculate $K_{N,SP,SM}$

N knows K_N and SP knows K_{SP}

$K_{N,SP,SM}$ is calculated *after* enabling isolation

No isolation, no key; no integrity, wrong key

Only SM on N is allowed to use $K_{N,SP,SM}$

Enforced through special instructions

Attestation and communication

Secure communication and remote attestation are provided through AEAD using the module key

Authenticated encryption with associated data

To provide confidentiality, integrity and authenticity

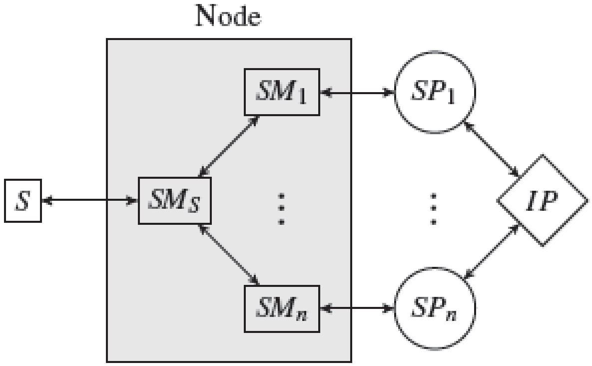
AEAD is provided through the `encrypt`/`decrypt` instructions

Using the key of the calling *SM*

A nonce should be included for freshness

Added to the associated data

An example (simplified) scenario



- Node manages a sensor S by means of an IP provided module SM_S
- Various SP's can install SM's:
  1. The SP contacts IP to get a $K_{N,SP}$
  2. SP creates SM, and calculates $K_{N,SP,SM}$
  3. SM is deployed on N using untrusted OS services
  4. SM is protected with the instruction:
     - **protect** *layout*, SP
     - This creates $K_{N,SP,SM}$ and enables memory protection on SM
  5. SP sends a request to SM (including a nonce No)
  6. SM computes a response (possibly calling SM_S and including No) and protects it:
     - Using the **encrypt** instruction
     - This creates an authenticated encryption of the response using $K_{N,SP,SM}$

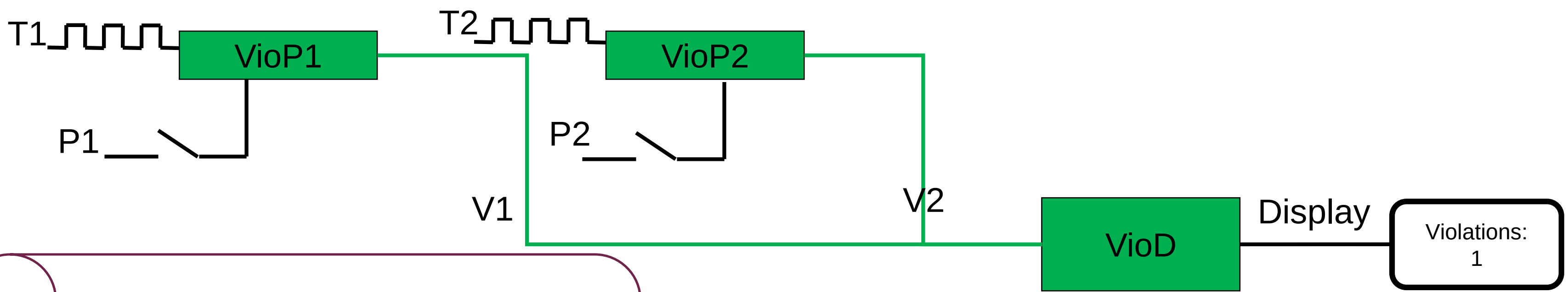
DistriNet iMinds KU LEUVEN

Sancus 2.0: Wrap-up

- Security enhanced microprocessor, with
  - Strong isolation of software modules
  - Remote attestation and secure communication
    - Using hardware implemented crypto
    - And a hierarchical key-derivation scheme
- Open-source
  - https://distrinet.cs.kuleuven.be/software/sancus/
  - Verilog source code based on the OpenMSP430
    - Usable in RTL simulator or on FPGA
  - Software stack
    - Compiler
    - Deployment tools

Distributed applications on Sancus 2.0

Consider again the parking app




```
module VioP1 (P1,T1;Violation1);

on P1(x) {
  if x then { taken = 1 }
  else { taken=0; count=0;
           Violation1(0) }
}

on T1(x) {
  if taken then { count = count + 1 };
  if count > MAX then
     { Violation1(1) }
}
```

```
module VioD(V1,V2;Display);
on V1(x) {
  if x then { c1 = 1 } else { c1 = 0 };
  if c1 then { Display(1) };
  if c2 then { Display(2)  }
}
on V2(x) … similar
```

Deploying the application

- Sancus 2.0 includes a software stack to securely deploy such applications
- SP provides:
  - Source code for each of the modules
  - Deployment descriptor
    - Mapping modules to nodes
    - Mapping unconnected application channels to physical I/O channels

DistriNet iMinds KU LEUVEN

Compiling source modules

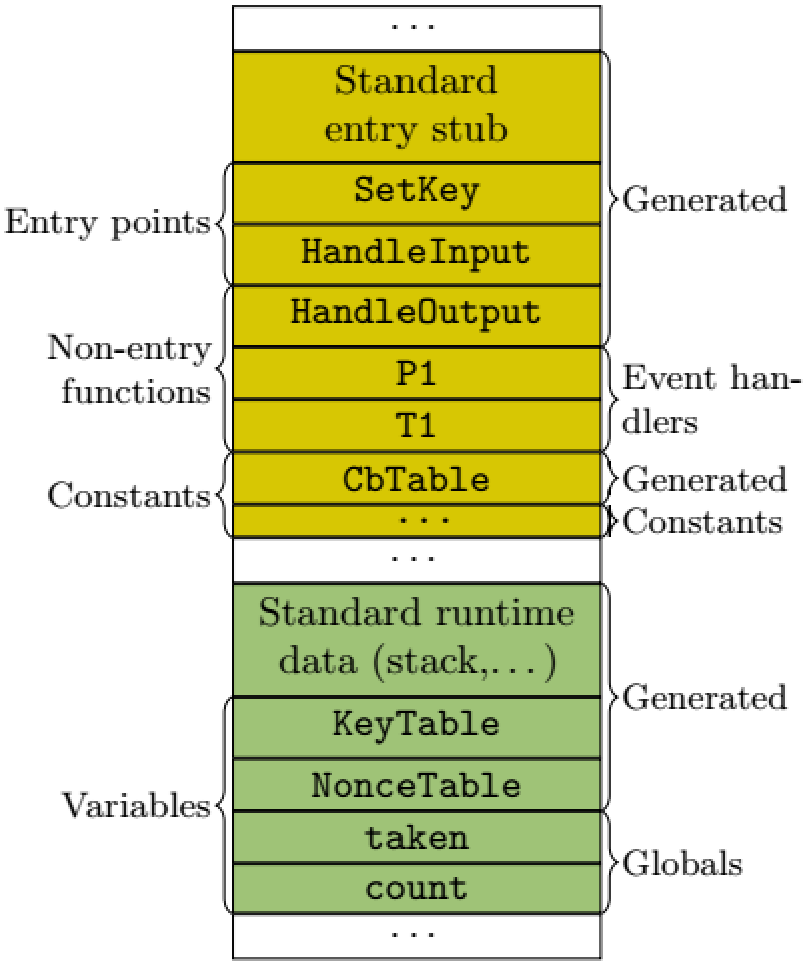
- Generated protected module should make sure to authenticate events:
  - P1 and T1 events must come from the corresponding input sensors
  - Violation1 event should go to the display module
- Done by associating a cryptographic key with every connection

```
module VioP1 (P1,T1;Violation1);

on P1(x) {
  if x then { taken = 1 }
  else { taken=0; count=0;
           Violation1(0) }
}

on T1(x) {
  if taken then { count = count + 1 };
  if count > MAX then { Violation1(1) }
}
```

Compiling source modules




```
def HandleInput(conn_id, payload):
  try:
    key = KeyTable[conn_id]
    if key != 0:
      cb = CbTable[conn_id]
      nonce = NonceTable[conn_id]
      cb(Decrypt(nonce, payload, key))
      NonceTable[conn_id] += 1
  except: pass
```

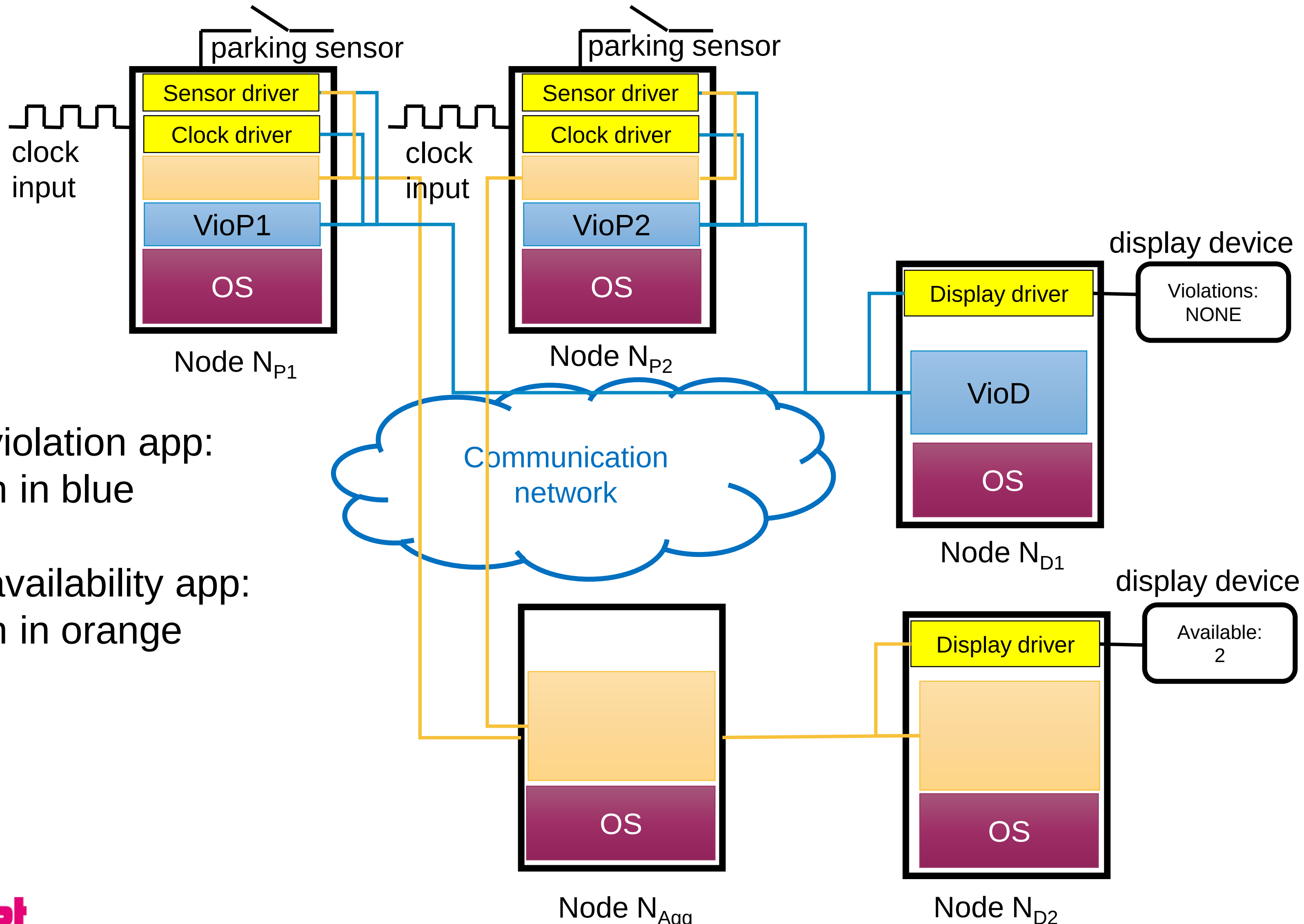
```
def HandleOutput(conn_id, data):
  key = KeyTable[conn_id]
  if key != 0:
    nonce = NonceTable[conn_id]
    NonceTable[conn_id] += 1
    payload = Encrypt(nonce, data, key)
    HandleLocalEvent(conn_id, payload)
```

```
def SetKey(payload):
  try:
    conn_id, key = Decrypt(payload)
    if KeyTable[conn_id] == 0:
      KeyTable[conn_id] = key
  except: pass
```

Deployment algorithm

1. Deployment of application code
   a) Compile all source modules to PMs
   b) Load them on the node specified in the deployment descriptor
   c) Generate cryptographic keys for each connection, and send them to the sending and receiving modules encrypted by the appropriate module keys
2. Connections to physical I/O channels
   a) Generate crypto keys for connections to physical outputs, send them to application module and protected driver module and attest success.
   b) Generate crypto keys for connections to physical inputs and send them to application module and protected driver module

Shared driver modules



Parking violation app:

- Shown in blue

Parking availability app:

- Shown in orange

Protected driver modules

- Driver modules have to satisfy properties (to be checked by the application deployer) that depend on the desired security guarantee:
  - E.g. Integrity:
    - Applications should be able to “take exclusive ownership” of protected driver modules for output devices
    - But protected driver modules for input devices can broadcast input events to all applications with only integrity protection
  - Confidentiality is dual

DistriNet iMinds KU LEUVEN

Security?

- Remember our security objective:
  - Security properties that hold of the source code should hold at run-time in the presence of arbitrary attackers within the attacker model
- This holds for arbitrary safety properties
  - E.g. “No violation is signaled for X on the display unless a car entered in X and stayed there for > N clock ticks”
- But it does not hold for:
  - Arbitrary *confidentiality* properties
    - This can be fixed at the expense of efficiency
  - Availability or real-time properties
    - Handling these is work-in-progress and needs weakening of the attacker model

Conclusion

- Long-term objective:

  Secure open software application platforms for distributed IT infrastructure

- Secure = "preserving application security"
- Achieving security is understood for some combinations of (1) attacker model and (2) relevant application security properties:
  - We discussed preservation of safety properties in the presence of network/software attackers
  - Other papers discuss preservation of module non-interference in the presence of local software attackers
  - Many other interesting combinations remain to be investigated!

Questions?

References

[1] Pieter Agten et al., *Secure Compilation to Modern Processors*. CSF 2012

[2] Job Noorman et al., *Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base*. USENIX Security 2013

[3] Jan Tobias Mühlberg et al., *Lightweight and Flexible Trust Assessment Modules for the Internet of Things*. ESORICS 2015.

[4] Marco Patrignani et al., *Secure compilation to protected module architectures*, ACM TOPLAS 2015

[5] Frank Piessens et al., *Security guarantees for the execution infrastructure of software applications*, IEEE SecDev 2016.

[6] Job Noorman et al., *Sancus 2.0: A Low-Cost Security Architecture for IoT Devices*, ACM TOPS 2017.