Sancus: A Low-Cost Security Architecture for Distributed IoT Applications on a Shared Infrastructure

Job Noorman

Public PhD Defence

19 Apr 2017

Safety considerations when lodging in a hotel

Untrusted guests

Locked room

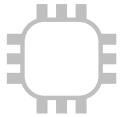
Unknown personnel

Just trust them

A hotel as an analogy for computing devices

Untrusted guests

Locked room



Other applications

Virtual memory, virtual machines,...

Unknown personnel

Just trust them

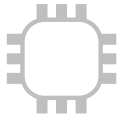
Operating system

Just trust it

A hotel as an analogy for computing devices

Untrusted guests

Locked room



Other applications

Virtual memory, virtual machines,...

Unknown personnel

Just trust them

Operating system

Just trust it

Hostile environment

Guarded transportation

Untrusted network

Cryptography

A hotel nobody would want to go, a good analogy for embedded devices

Untrusted guests

Locked room

Unknown personnel

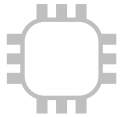
Just trust them

Hostile environment

Guarded transportation

No rooms/walls

Don't go there



Other applications

Virtual memory, virtual machines,...

Operating system

Just trust it

Untrusted network

Cryptography

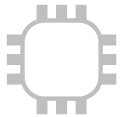
No software isolation

Well, just go there anyway...

Lack of isolation problematic for third-party extensibility

Private hotel

No other guests



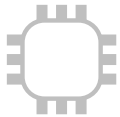
Monolithic application

Single, fixed binary

Lack of isolation problematic for third-party extensibility

Private hotel

No other guests



Monolithic application

Single, fixed binary

Trusted guests

Bouncer at the entrance

First-party extensibility

Authenticated binaries

Lack of isolation problematic for third-party extensibility

Private hotel

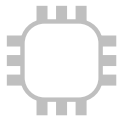
No other guests

Trusted guests

Bouncer at the entrance

Untrusted guests

Rooms for security



Monolithic application

Single, fixed binary

First-party extensibility

Authenticated binaries

Third-party extensibility

Memory isolation for security

Although very relevant, low-end devices lack effective security features

More threats on embedded devices

Due to network connectivity and third-party extensibility

No effective solutions exist

It's "a mess" (Viega and Thompson)

Researchers are exploring this area

E.g., SMART (El Defrawy et al.)

Goal: create a trustworthy hotel for secure lodging

Isolation of guests

Private rooms for the guests

Goal: create a trustworthy hotel for secure lodging

Isolation of guests

Private rooms for the guests

Attestation of well-being

Proof unharmed arrival

Goal: create a trustworthy hotel for secure lodging

Isolation of guests

Private rooms for the guests

Attestation of well-being

Proof unharmed arrival

Secure communication

Call home/with other guests

Goal: create a trustworthy hotel for secure lodging

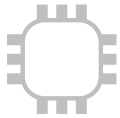
Isolation of guests
Private rooms for the guests

Attestation of well-being
Proof unharmed arrival

Secure communication
Call home/with other guests

Zero-people Trusted Lodging Base
Less people to trust, less can go wrong

Goal: design and implement a low-cost, extensible security architecture



Isolation of software modules

Memory isolation for data and code

Attestation of a module's state

Proof integrity and isolation

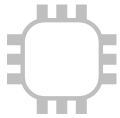
Secure communication

Both locally and remotely

Zero-software Trusted Computing Base

Counteracting attackers with *full* control over infrastructural software

Extended goals



Confidentiality of communication and code

Authenticity often not enough

Guarantees for distributed applications

On an infrastructure shared with the attacker

Securing unaltered legacy code

When changing code is impossible/too expensive

Target: a generic system model

Infrastructure provider

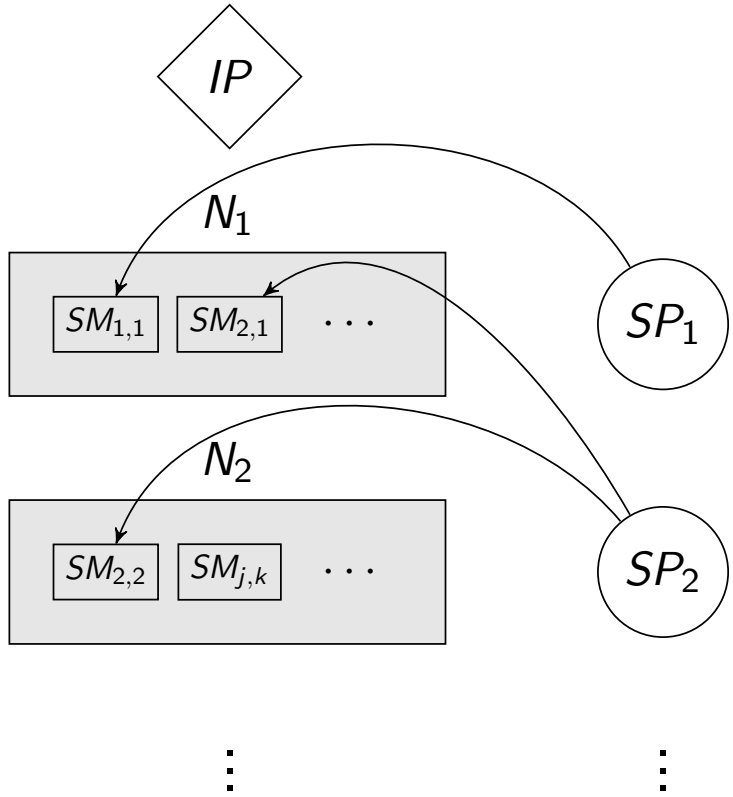
IP owns and administers nodes N_i

Software providers

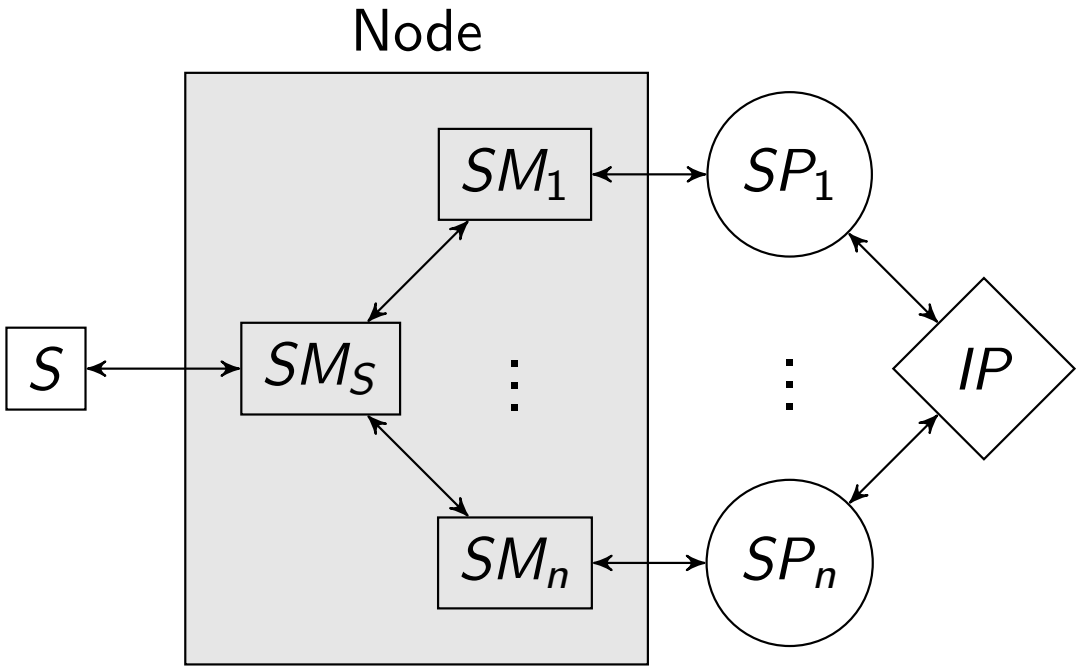
SP_j wants to use the infrastructure

Software modules

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



Example node configuration



Preview

1. Module isolation
2. Key management
3. Remote attestation and secure communication
4. Secure linking
5. Results

Overview

1. Module isolation
   - Hotel analogy
   - Module layout
   - Access rights enforcement
2. Key management
3. Remote attestation and secure communication
4. Secure linking
5. Results

Safe lodging for guests by building rooms

Walls to keep other guests and personnel out

Safeguard possessions, prevent attacks

Door with lock to allow controlled access

We want room service, right?

Modules are bipartite with a text section and a *private* data section

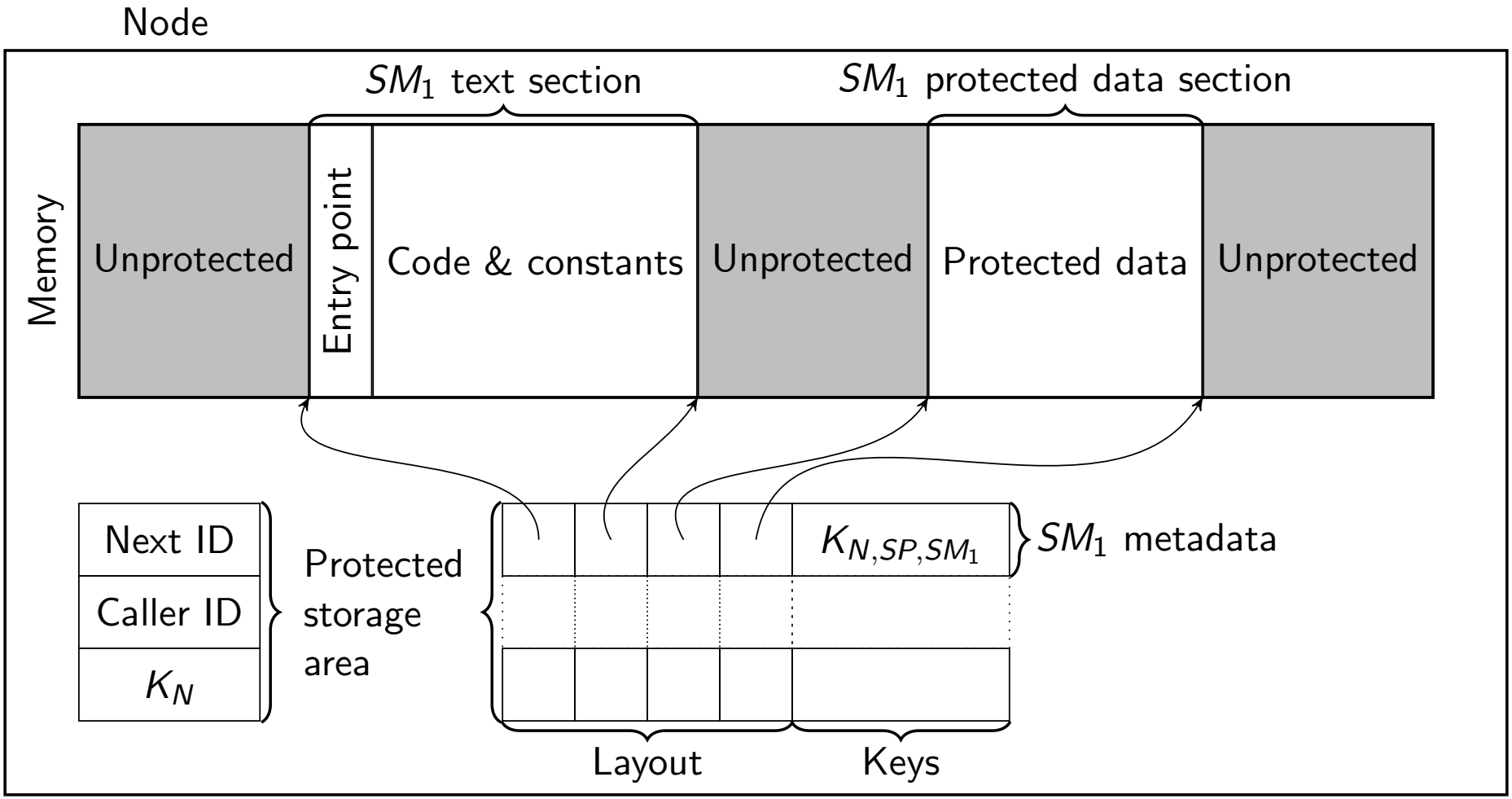
Immutable text section

Containing code and constants

Private data section

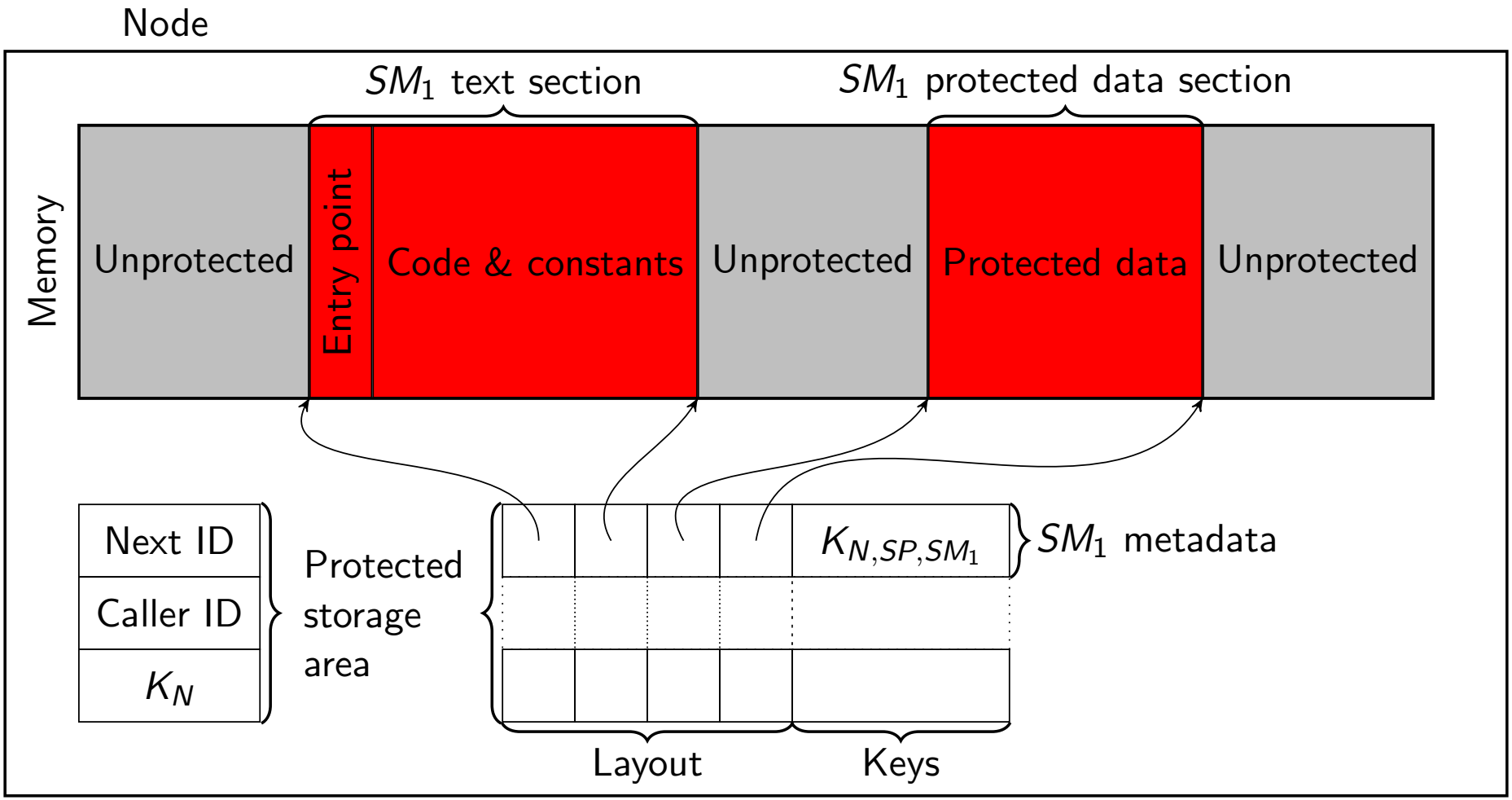
Containing secret runtime data

Node with one software module loaded



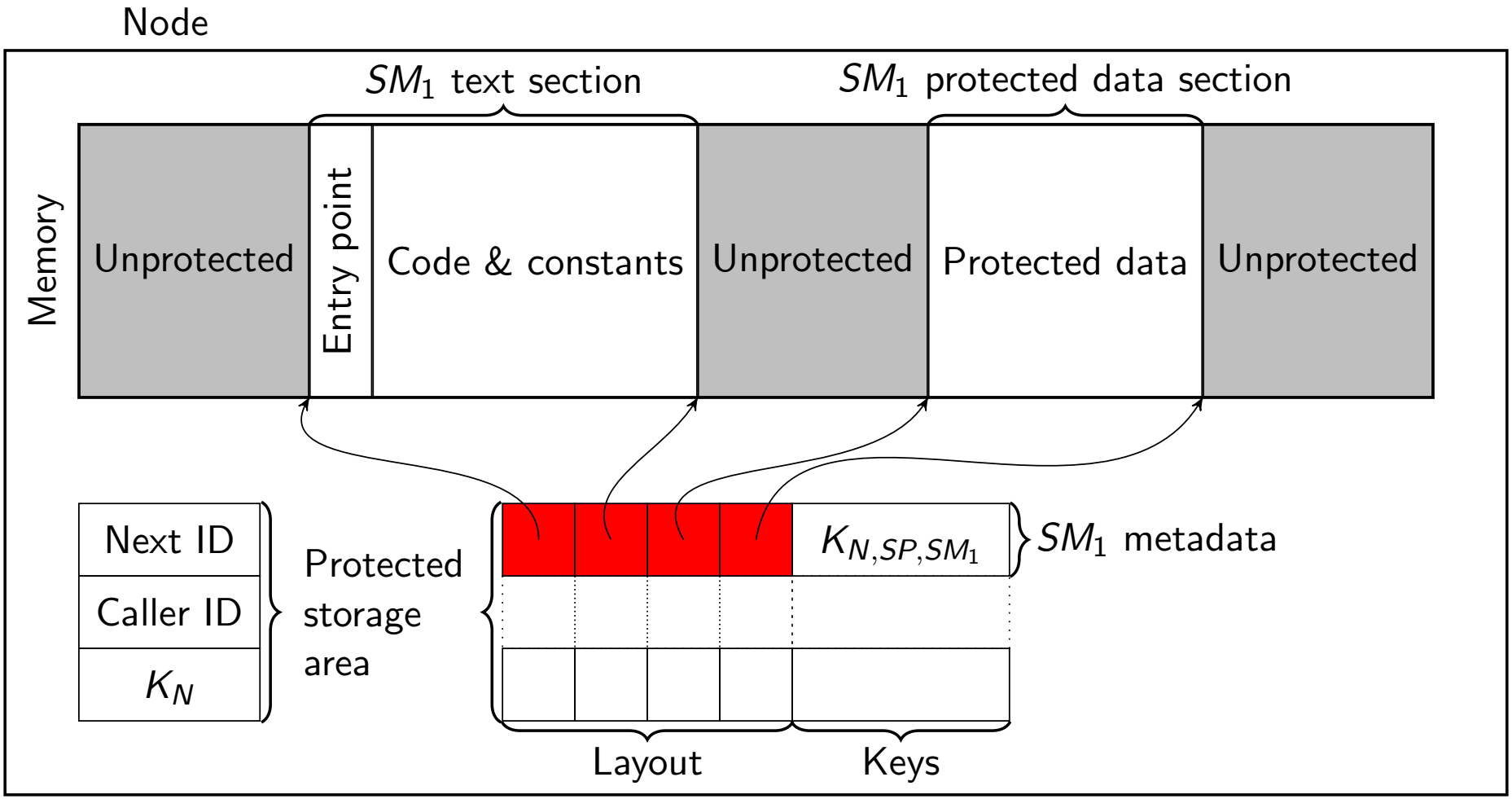
Node with one software module loaded

Text and data sections



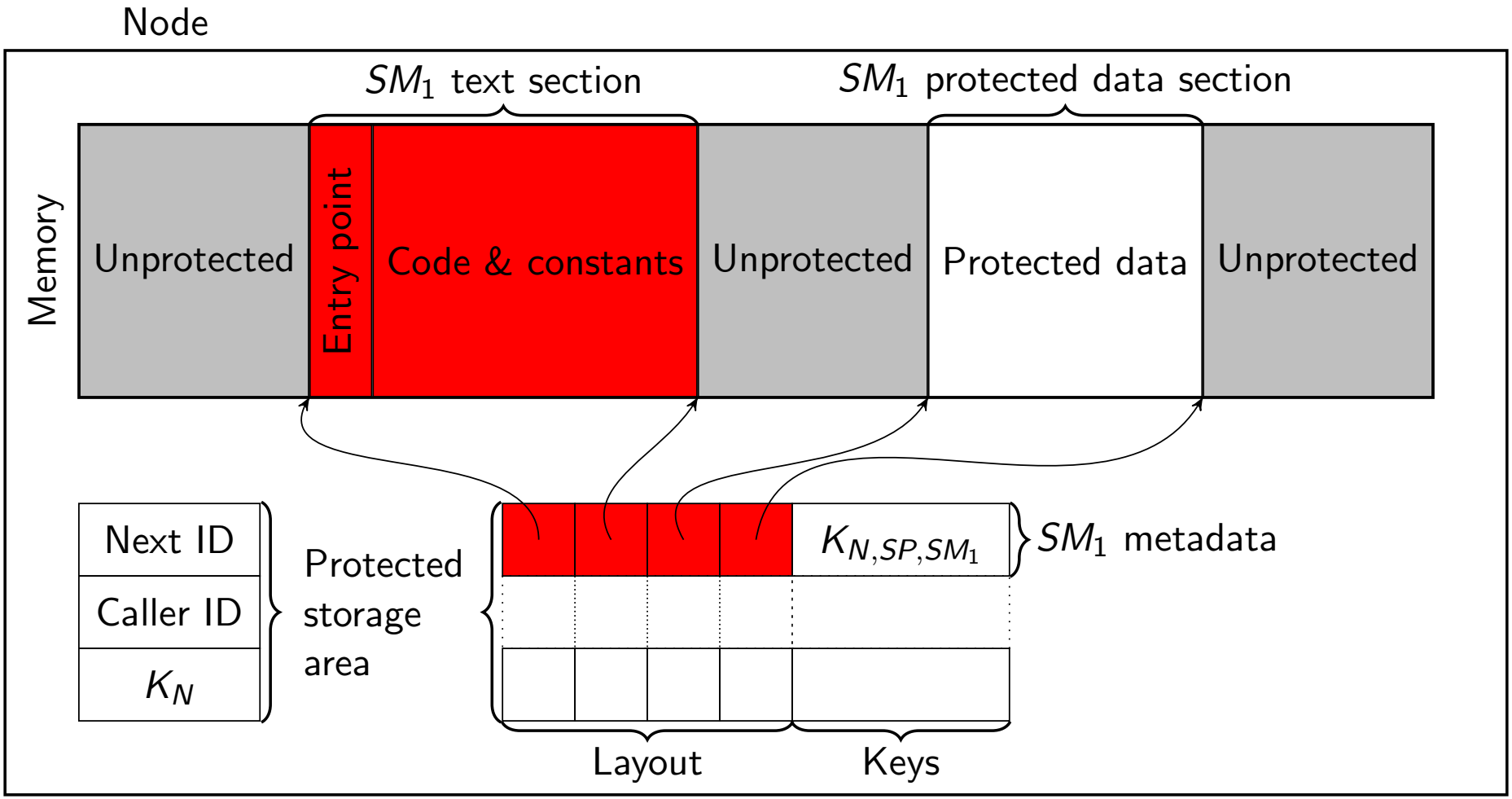
Node with one software module loaded

Module layout



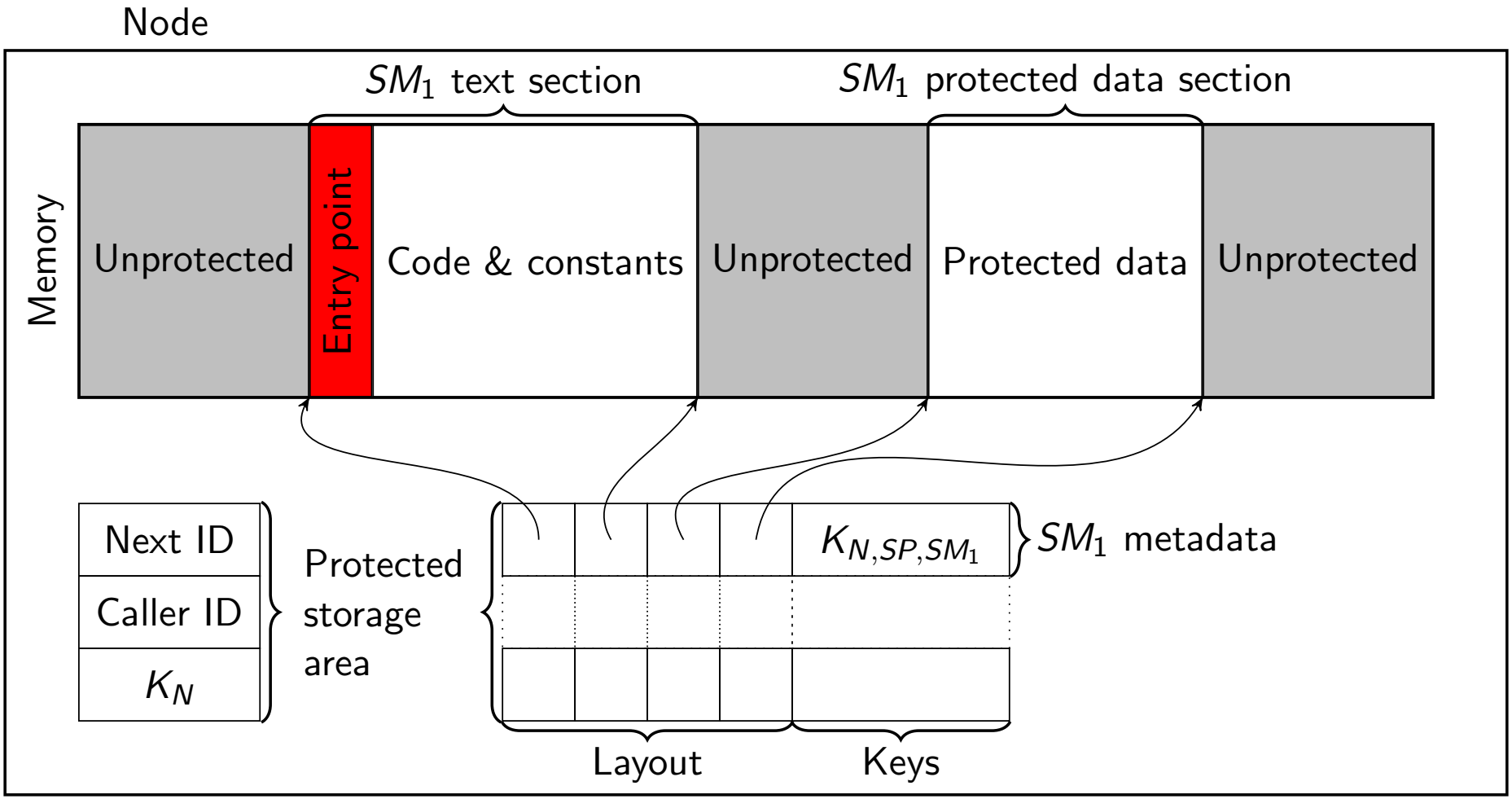
Node with one software module loaded

Module identity



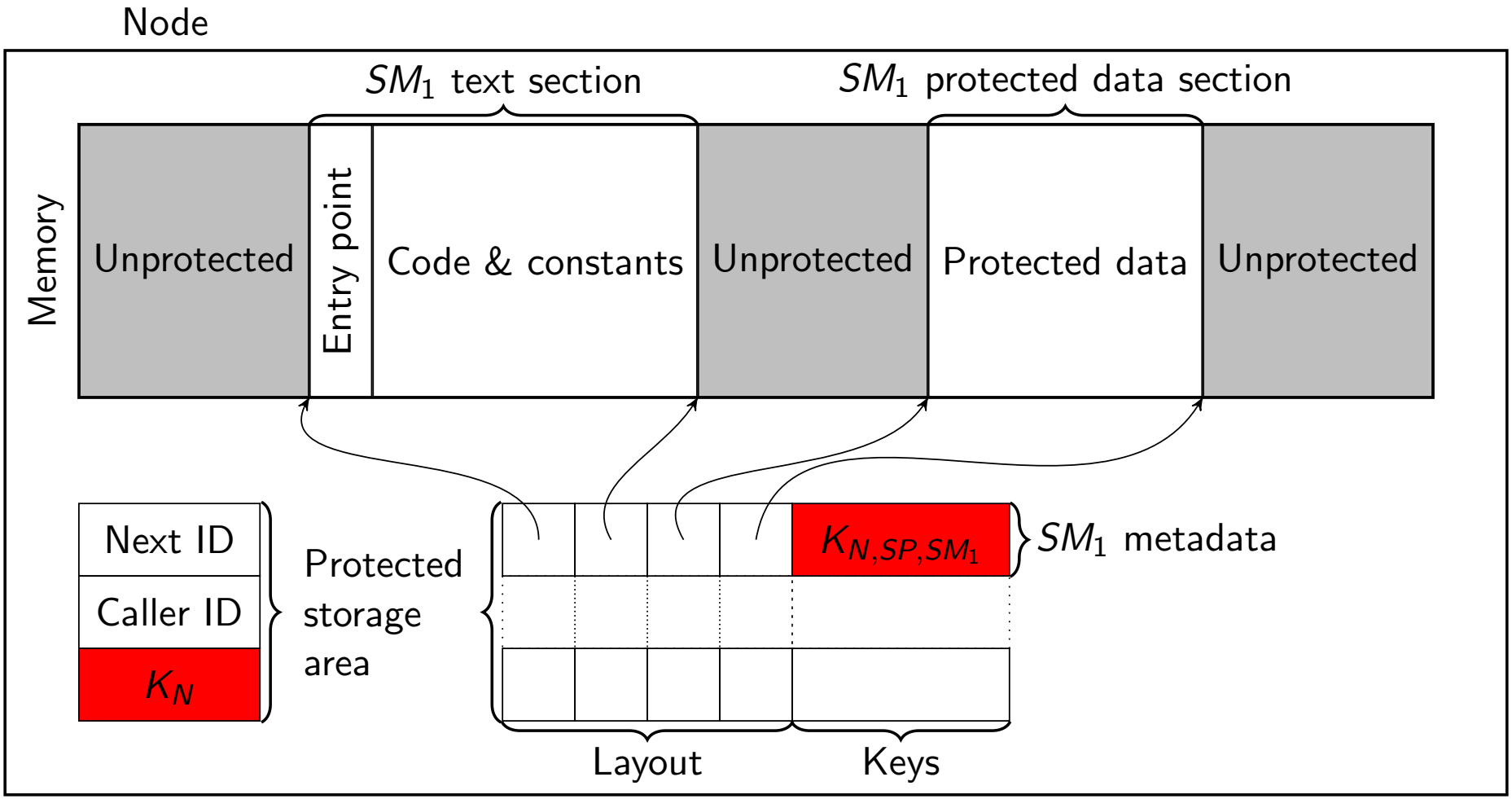
Node with one software module loaded

Module entry point



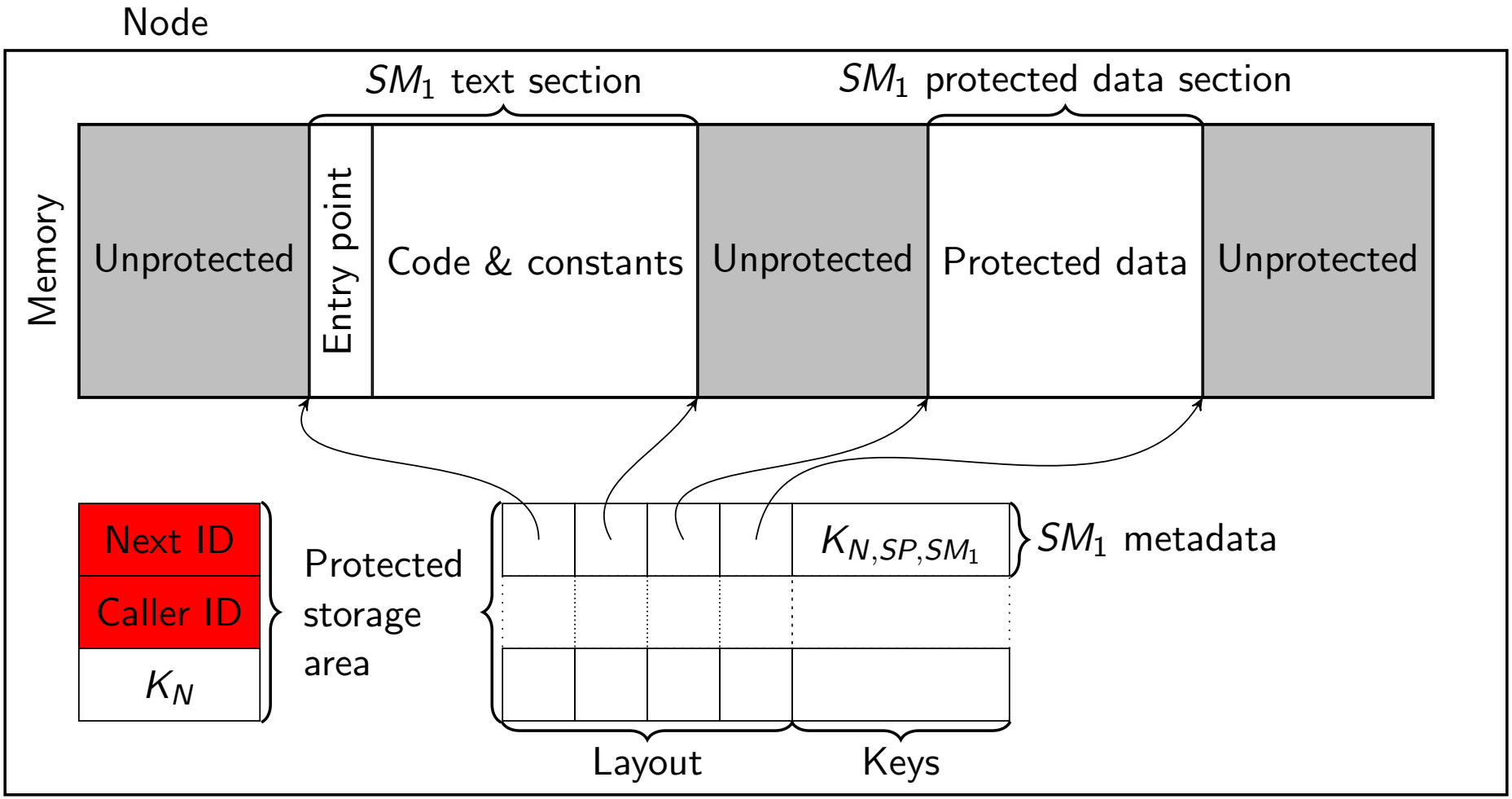
Node with one software module loaded

Module keys



Node with one software module loaded

ID registers



Modules are isolated using *program-counter based memory access control*

- Variable access rights
  - Depending on the current program counter

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

From/to	Text	Data	Unprotected
Text			
Other			

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

From/to	Text	Data	Unprotected
Text			
Other			

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

From/to	Text	Data	Unprotected
Text			
Other			

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

Only accessible from text section

From/to	Text	Data	Unprotected
Text		`rw-`	
Other		`---`	

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

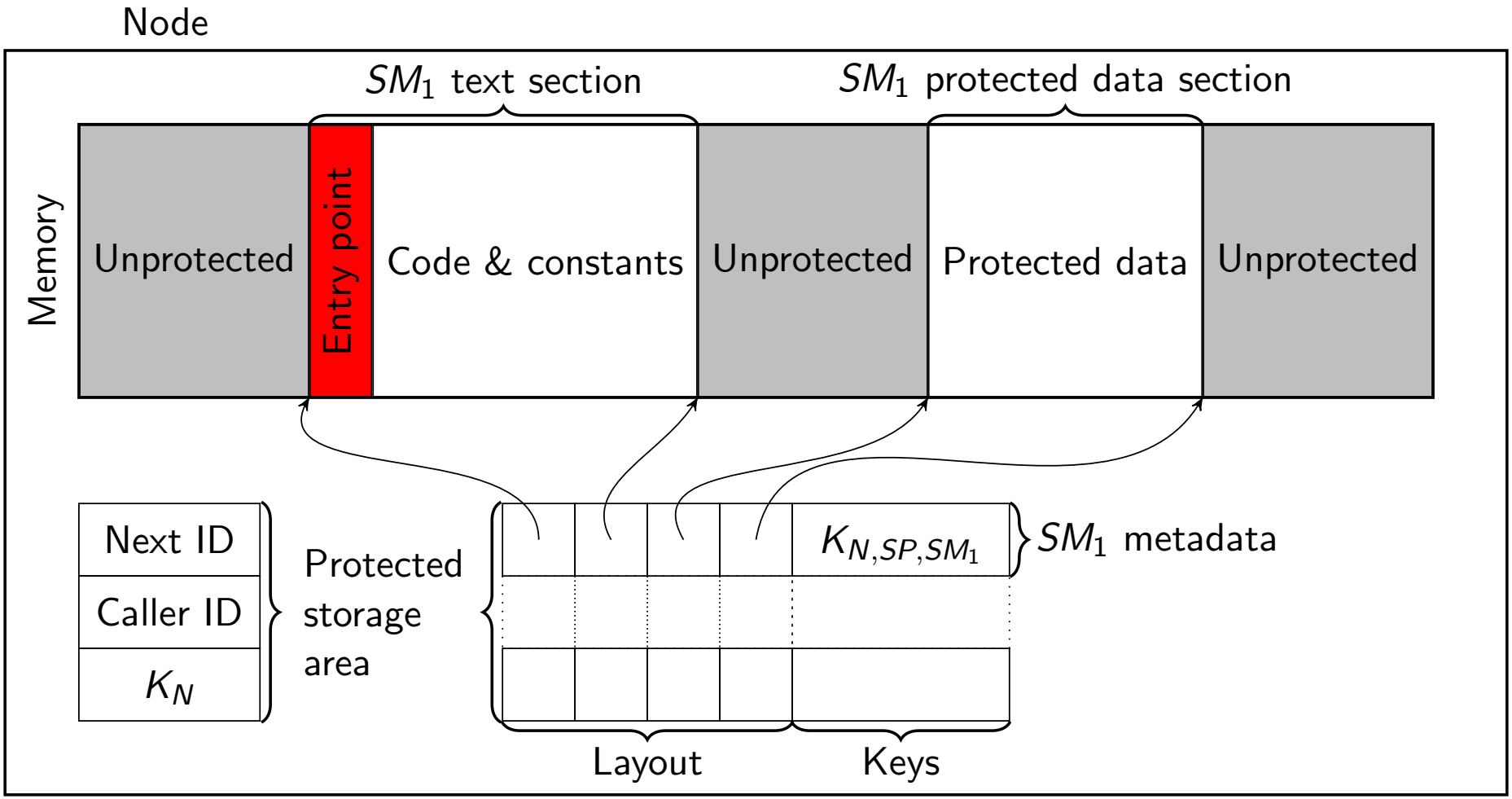
Only accessible from text section

Protection against code misuse (e.g., ROP)

From/to	Text	Data	Unprotected
Text	r-x	rw-	
Other	---	---	

Node with one software module loaded

Module entry point



Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

Only accessible from text section

Protection against code misuse (e.g., ROP)

Enter module through single entry point

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

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

Only accessible from text section

Protection against code misuse (e.g., ROP)

Enter module through single entry point

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

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

Only accessible from text section

Protection against code misuse (e.g., ROP)

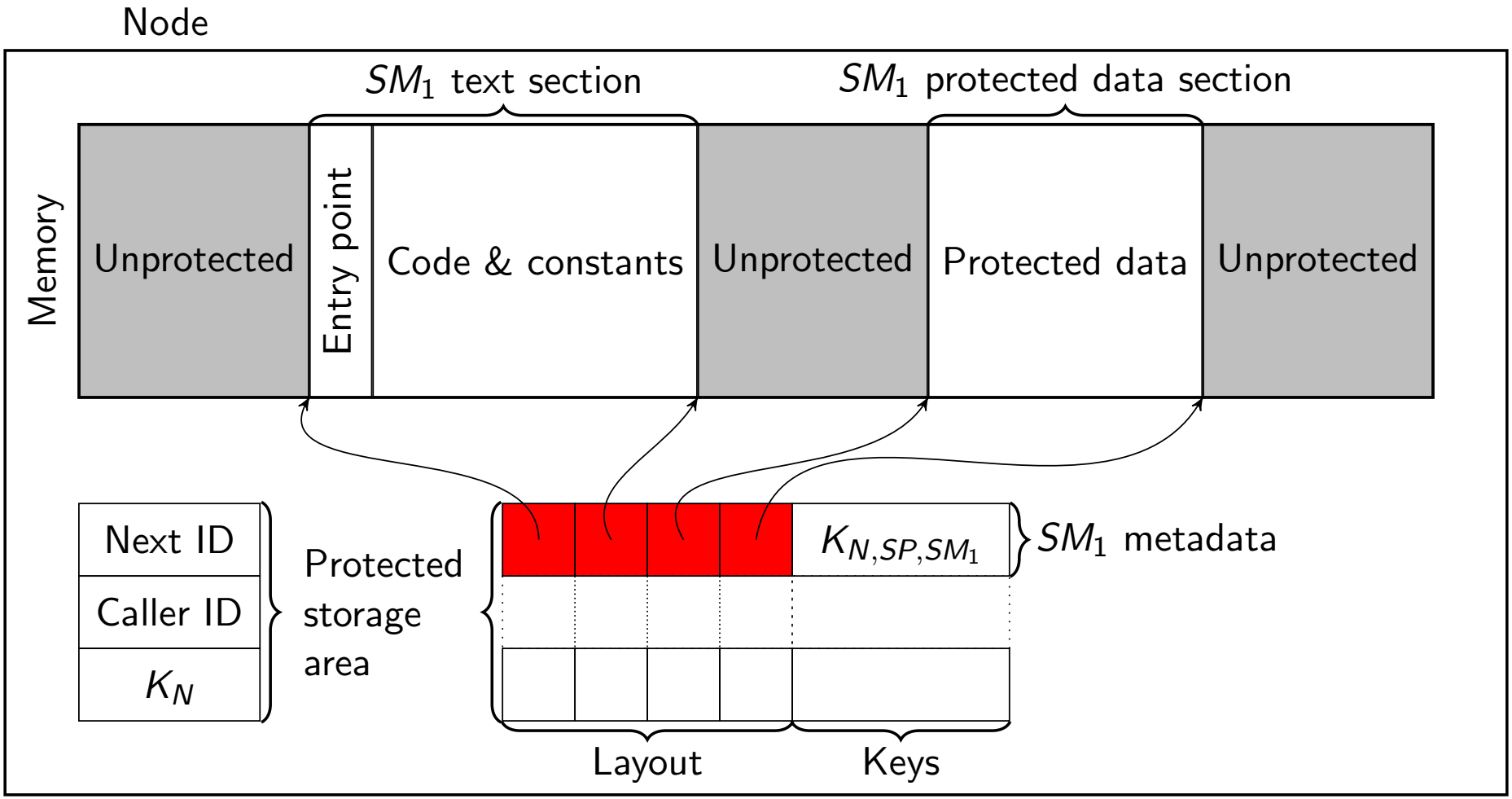
Enter module through single entry point

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

Isolation can be enabled/disabled using new instructions

Node with one software module loaded

Module layout



Isolation can be enabled/disabled using new instructions

`protect` *layout*, *SP*

Enables isolation at *layout*

`unprotect`

Disables isolation of current SM

Overview

Providing a flexible, inexpensive way for secure communication

Establish a shared secret

Between *SP* and its module *SM*

Use symmetric crypto

Public-key is too expensive for low-cost nodes

Ability to deploy modules without *IP* intervening

After initial registration, that is

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

Infrastructure provider is trusted party

Able to derive all keys

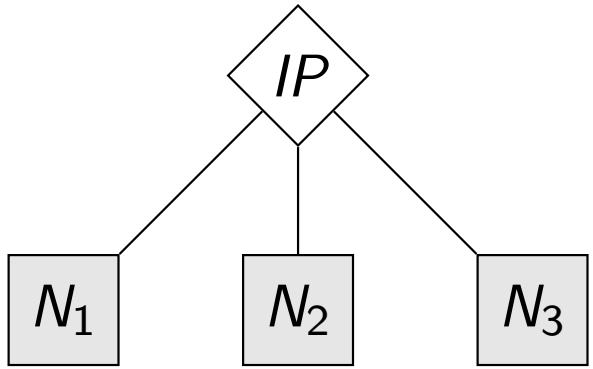
Key derivation scheme allowing both Sancus and *SP*'s to get the same key

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

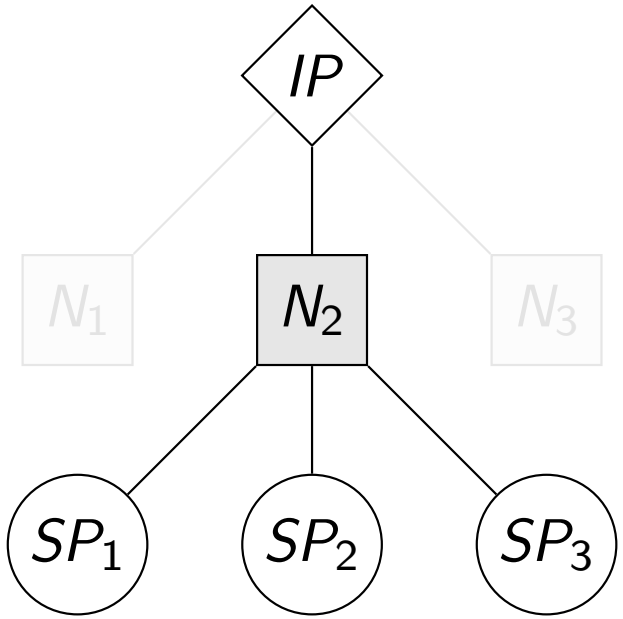
Able to derive all keys

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

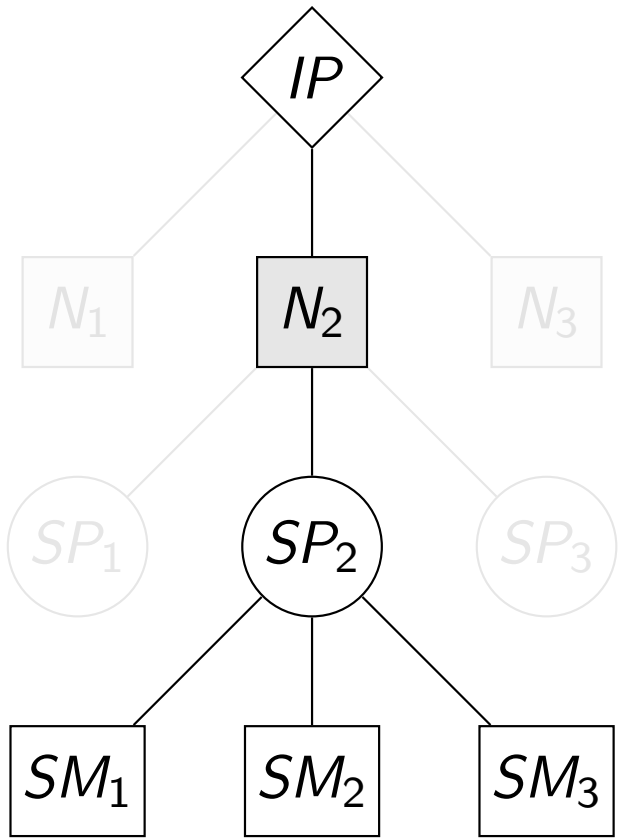
Generated at random

Derived key based on *SP* ID

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

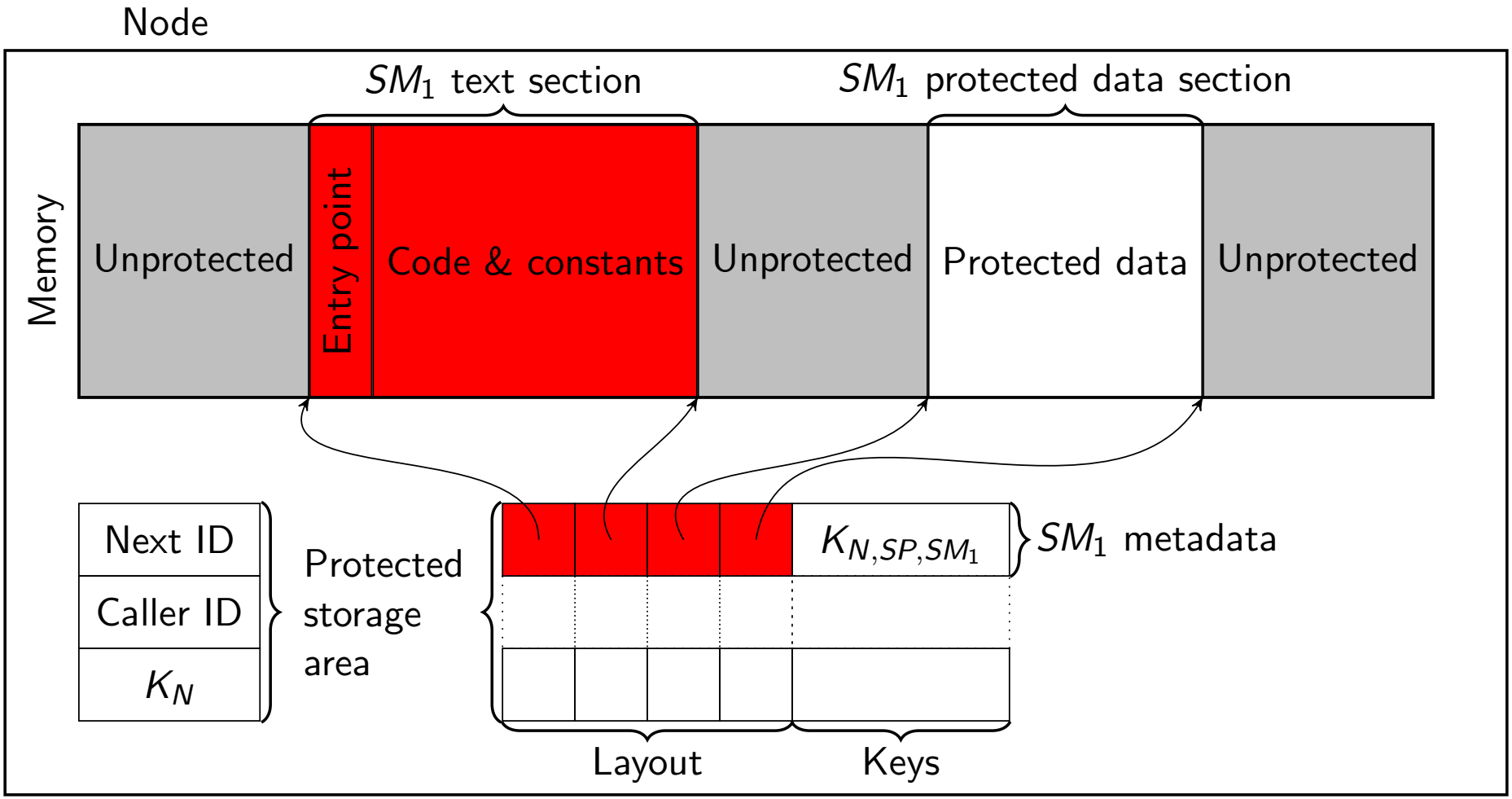
Derived key based on *SM* identity

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



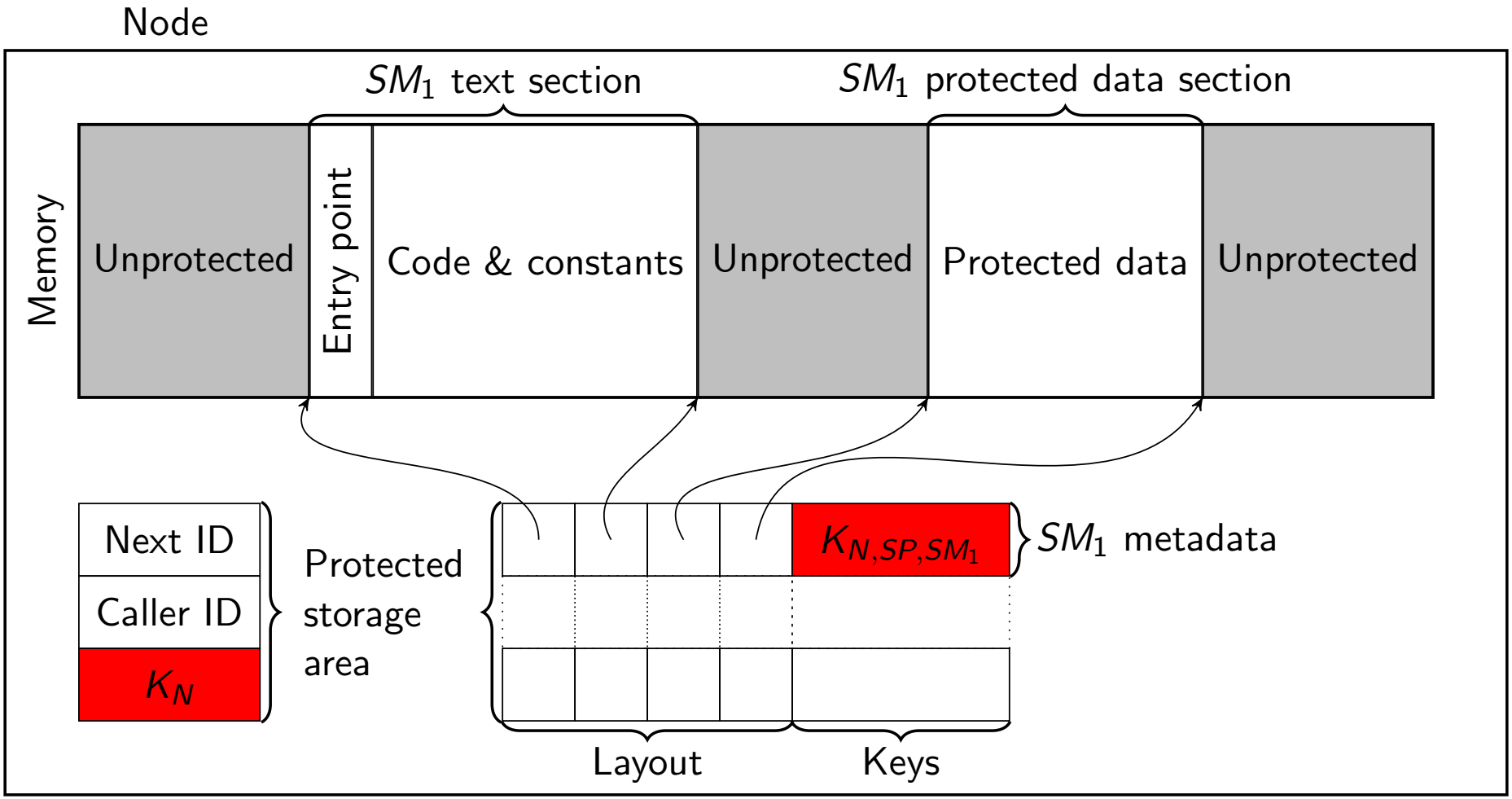
Node with one software module loaded

Module identity



Node with one software module loaded

Module keys



Isolation can be enabled/disabled using new instructions

`protect` *layout*, *SP*

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

`unprotect`

Disables isolation of current SM

Overview

Verify safety of guests and communicate with them

Verify guest has safely arrived in room

A lot can go wrong in our hostile setting

Secure communication between guests and their home

With mutual authentication

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

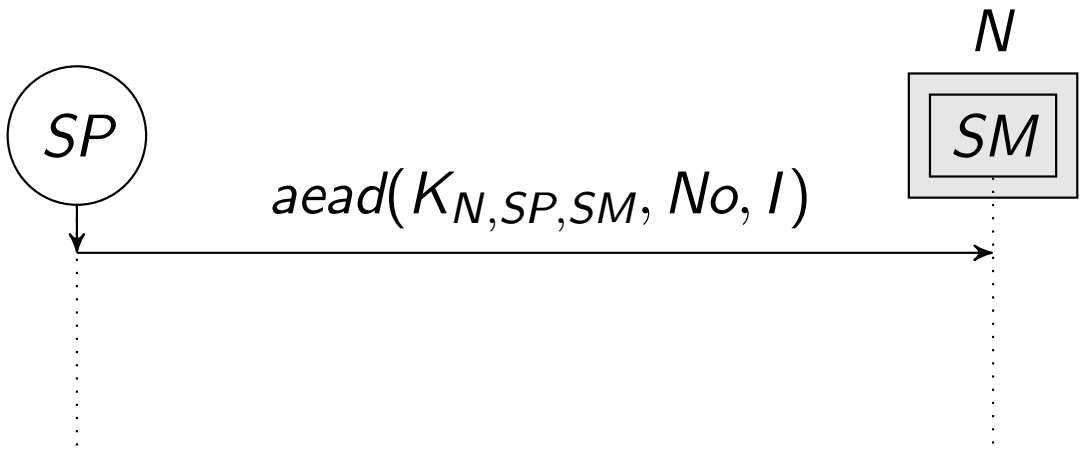
No isolation, no key; no integrity, wrong key

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

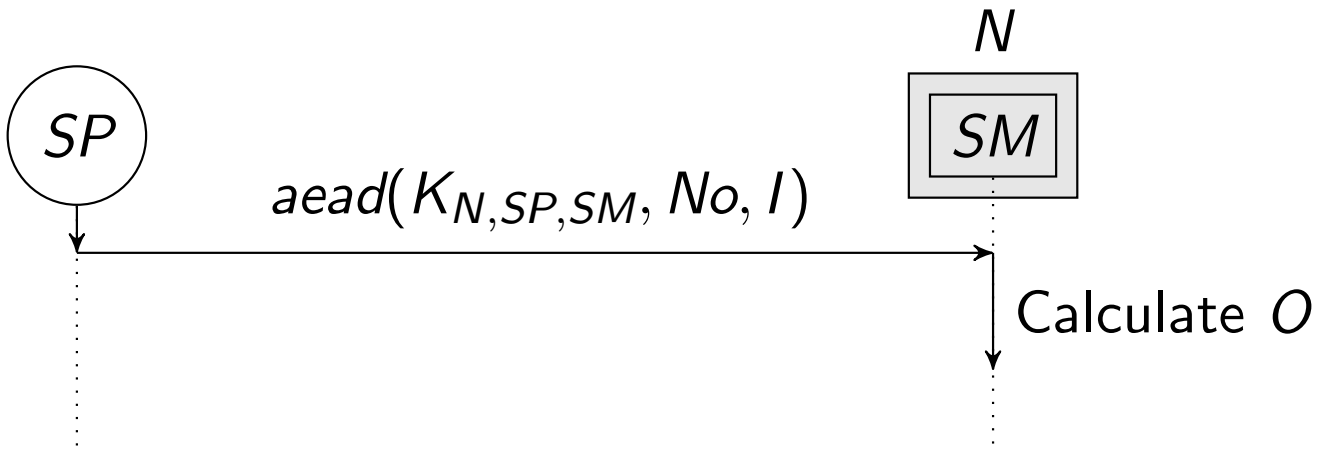
Enforced through special instructions

Secure communication is provided by authenticated encryption using the module key

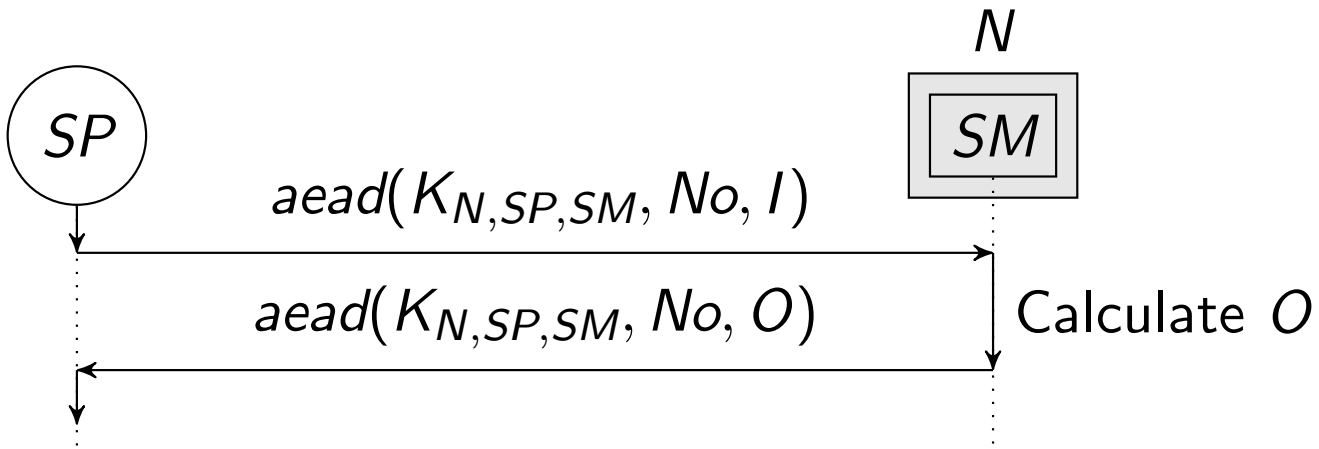
Secure communication is provided by authenticated encryption using the module key



Secure communication is provided by authenticated encryption using the module key



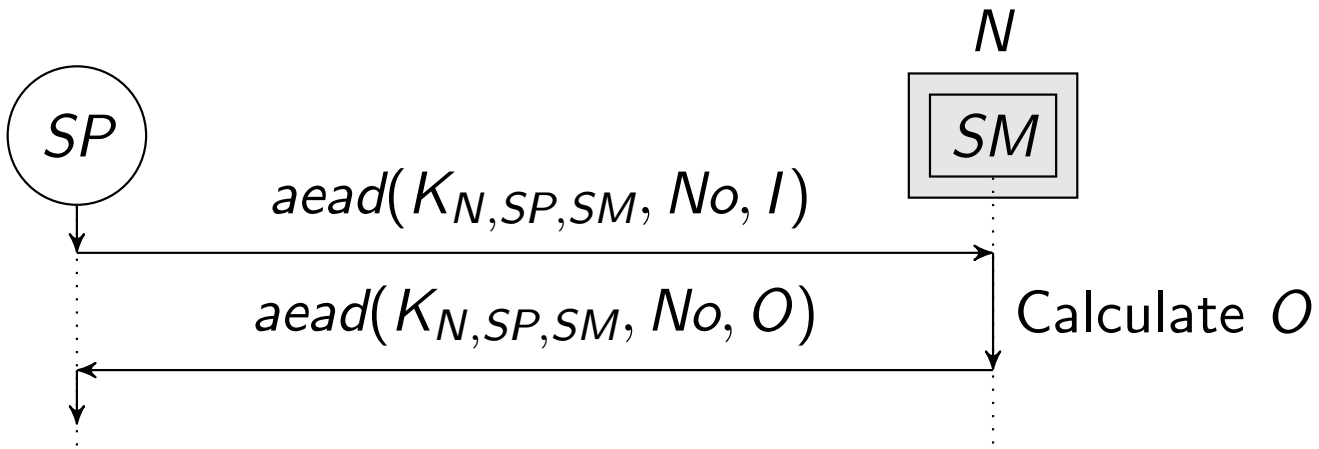
Secure communication is provided by authenticated encryption using the module key



AEAD is done using the `encrypt`/`decrypt` instructions

Using the key of the calling *SM*

Secure communication is provided by authenticated encryption using the module key

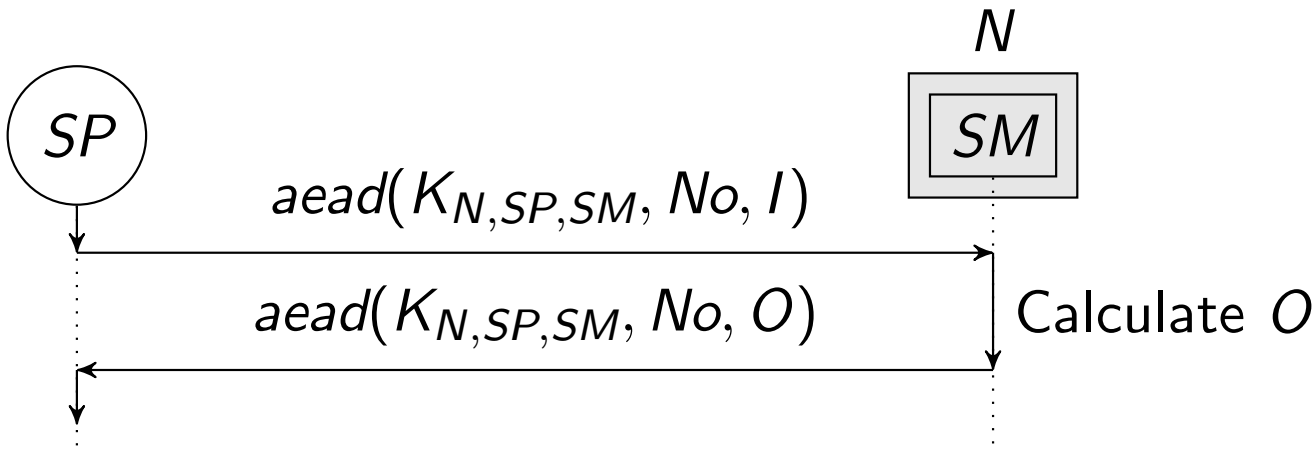


AEAD is done using the `encrypt`/`decrypt` instructions

Using the key of the calling *SM*

This scheme provides trust in the confidentiality, integrity and authenticity of messages

Remote attestation is provided through secure communication



Attest integrity, isolation and liveliness

Of *SM* by *SP*

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

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}

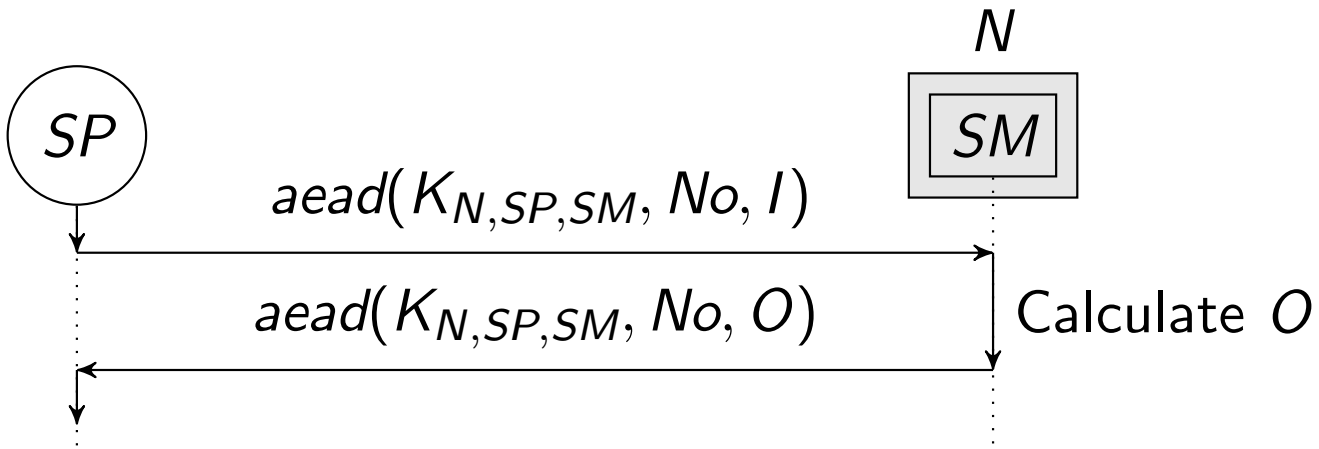
$K_{N,SP,SM}$ is calculated *after* enabling isolation

No isolation, no key; no integrity, wrong key

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

Enforced through special instructions

Remote attestation is provided through secure communication



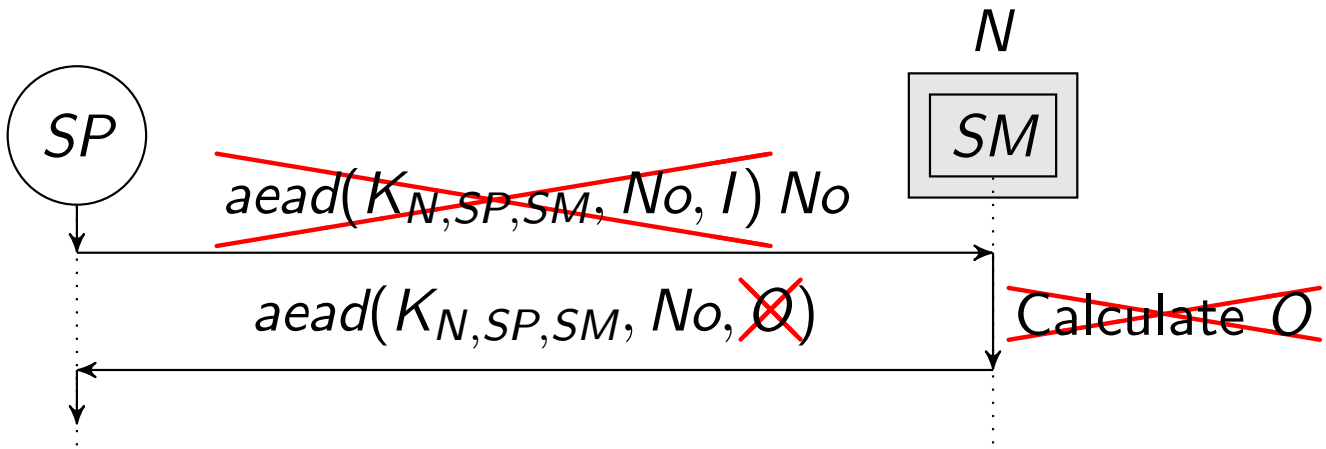
Attest integrity, isolation and liveliness

Of *SM* by *SP*

Integrity and isolation attested by AEAD, liveliness by nonce

Thus included in secure communication

Remote attestation is provided through secure communication



Attest integrity, isolation and liveliness

Of *SM* by *SP*

Integrity and isolation attested by AEAD, liveliness by nonce

Thus included in secure communication

$\Rightarrow$ remote attestation $\subset$ secure communication

So can be achieved more easily

Overview

Let guests safely interact with each other

Safely go to other rooms

To talk with other guests

Know who is in the room/at the door

Do not get in a room with untrusted guests

Enabling efficient and secure local inter-module function calls

Verify the *SM* that is to be called

Is it the correct, isolated *SM*?

Inherently different from secure communication

May belong to different *SP*s; no shared secret

We can rely on protected local state

Gives rise to interesting optimizations

Modules are verified by calculating a cryptographic hash over their identity

Module A wants to call module B

A is deployed with a hash of B's identity

In its text section or, using secure communication, in its data section

Modules are verified by calculating a cryptographic hash over their identity

Module A wants to call module B

A is deployed with a hash of B's identity

In its text section or, using secure communication, in its data section

A calculates the hash of B's *actual* identity

If they match B can safely be called

Modules are verified by calculating a cryptographic hash over their identity

Module A wants to call module B

A is deployed with a hash of B's identity

In its text section or, using secure communication, in its data section

A calculates the hash of B's *actual* identity

If they match B can safely be called

Done through new instruction: `attest`

Need to be ensured of B's isolation

The expensive hash calculation is needed only once

We only need to know if the same module is still there

After initial verification, that is

The expensive hash calculation is needed only once

We only need to know if the same module is still there

After initial verification, that is

Sancus assigns unique IDs to modules

Never reused within a boot-cycle

`attest` returns the ID of the verified module

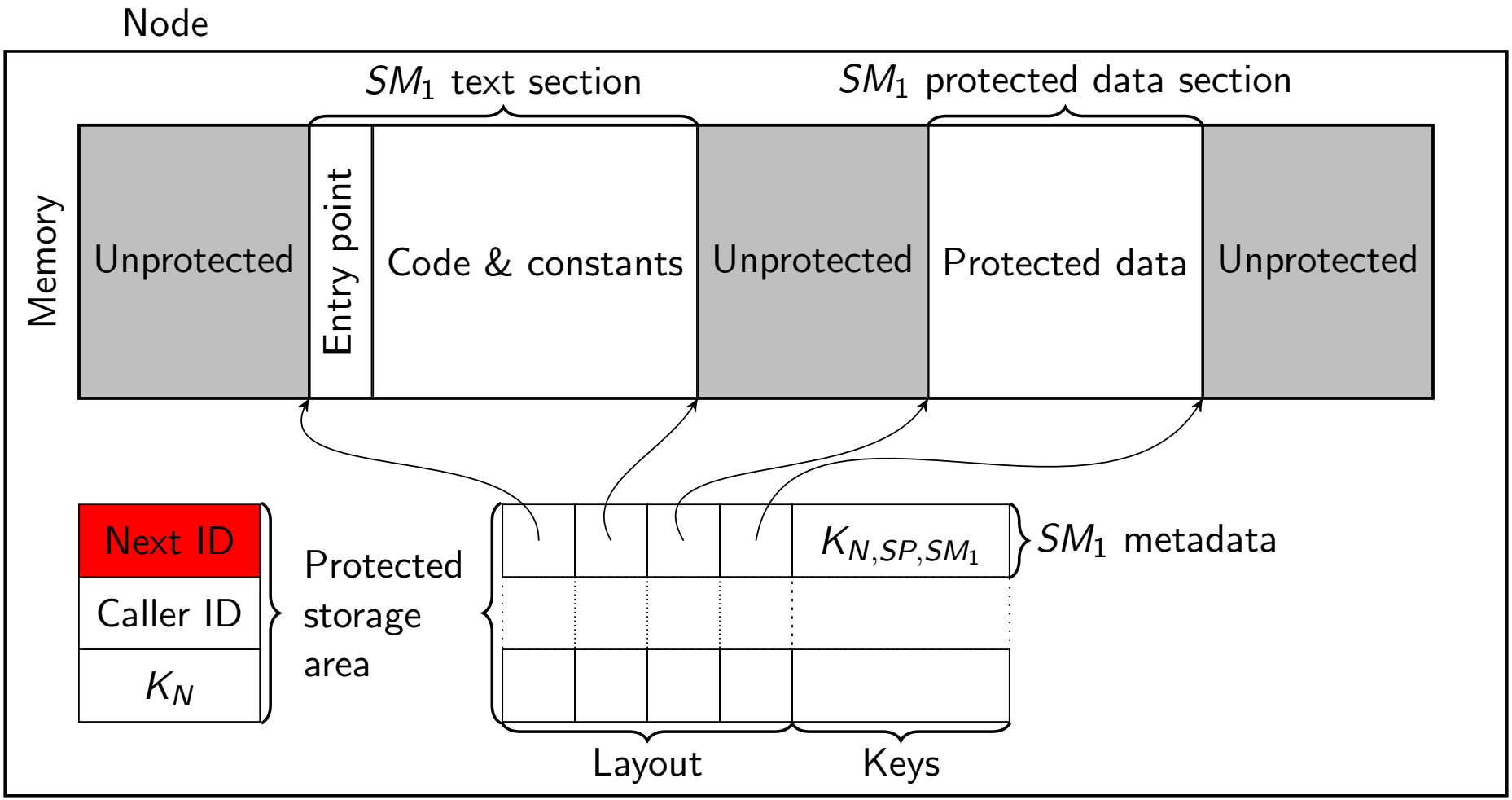
Can be stored in the protected section

Later calls can use a new instruction: `get-id`

Check if the same module is still loaded

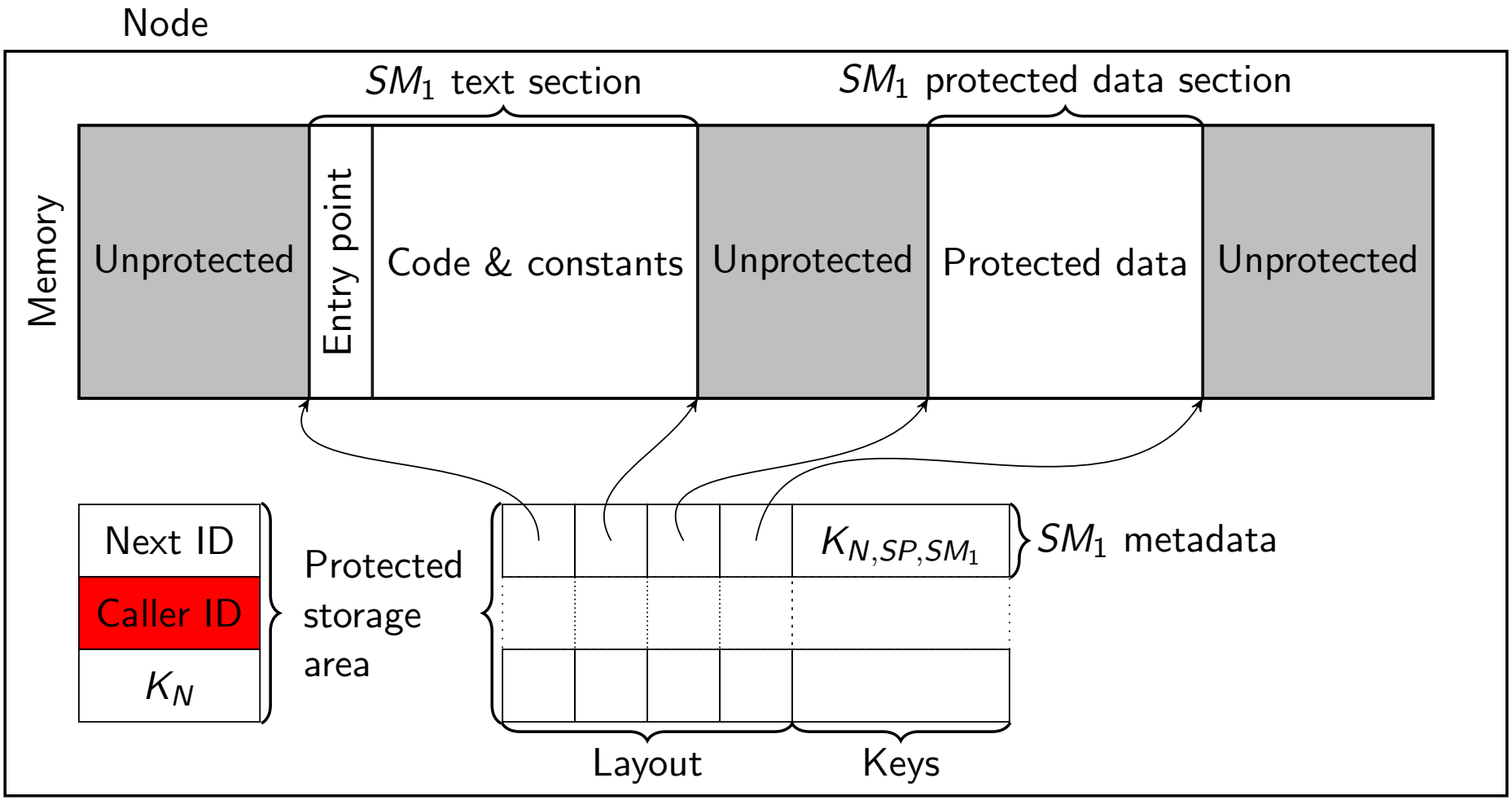
Node with one software module loaded

ID used for next loaded module



Node with one software module loaded

ID of the previously executing module



Overview

Complete implementation of Sancus based on the MSP430 architecture

Based on the openMSP430 project

Very mature open-source MSP430 implementation

Built on existing cryptographic primitives:

- AEAD/KDF: SpongeWrap (Bertoni et al.)
- Sponge/hash: spongent (Bogdanov et al.)

Usable in RTL simulator and FPGA

For easy testability of Sancus

Automatically handling the intricacies of compiling Sancus modules

Placing the runtime stack in the protected section

Prevent access by untrusted code

Clearing registers on module exit

Prevent data leakage

Supporting more than one entry point

Dispatching through a single entry point

Automatically handling the intricacies of compiling Sancus modules

```
#include <sancus/sm_support.h>
#define ID foo

int  SM_DATA(ID)  protected_data;
void SM_FUNC(ID)  internal_function() {/*...*/}
void SM_ENTRY(ID) entry_point() {/*...*/}
```

Review

1. **Module isolation**
   Isolation using *program-counter based access control*

2. **Key management**
   Hierarchical scheme with keys based on module's *identity*

3. **Remote attestation and secure communication**
   Attestation based on the ability to use a key

4. **Secure linking**
   Module verification based on a hash of its identity

5. **Results**
   Simulator, FPGA, and automatic compilation

Sancus: A Low-Cost Security Architecture for Distributed IoT Applications on a Shared Infrastructure

Job Noorman

Public PhD Defence

19 Apr 2017