Applying Theoretical Crypto's Real/Ideal Paradigm to the Security of Ordinary Programs

Alley Stoughton
IMDEA Software Institute

DeepSpec Workshop
Princeton University
June 6–8, 2016

This work was sponsored by DARPA under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors, and are not necessarily endorsed by the Department of Defense or the United States Government.

Program Security as a Specification

- Program security is a kind of specification
- But one that's rather different from the specification of input/output behavior
- I'm going to illustrate how a key definitional framework of theoretical cryptography — the Real/Ideal Paradigm — can be used to define the security of some ordinary programs
- Instead of probabilistic security as in crypto, we use language features like data abstraction to get absolute guarantees
- I'll use the two-player board game Battleship as my example

Defining Program Security

- Surprisingly little work on specifying whole program security
  - More specific than noninterference theorems for information flow control (IFC) languages
- State of the art: employ numerous program security annotations, as in Jif
  - Attempts to capture informal policy
  - Tells an auditor where to focus — but not exactly what do look for

Zdancewic (2004):
“… we do not yet have the tools to easily describe desired security policies. We do not understand the right high-level abstractions for specifying information-flow policies.”

Battleship Rules
Ship Placement

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	c	c	c	c	c	b				
D						b				
E						b				
F										
G			p		s	s	s			
H			p				d			
I							d			
J							d			

Battleship Rules
Shooting

Player's Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	c	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	s			
H			p				D			
I				★			D			
J				★	★	★	d			

Opponent's Shooting Record

	A	B	C	D	E	F	G	H	I	J
A										
B										
C		✚	✚	✚	✚		★			
D		★		★						
E							★			
F										
G		★			✚	✚				
H							✚			
I				★			✚			
J				★	★	★				

Shoot CA –

Battleship Rules
Shooting

Player's Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	s			
H			p				D			
I				★			D			
J				★	★	★	d			

Opponent's Shooting Record

	A	B	C	D	E	F	G	H	I	J
A										
B										
C	C	✚	✚	✚	✚		★			
D		★		★						
E							★			
F										
G		★			✚	✚				
H							✚			
I				★			✚			
J				★	★	★				

Shoot CA – "Sank Carrier"

Battleship Rules
Shooting

Player’s Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	s			
H			p				D			
I				★			D			
J				★	★	★	d			

Opponent’s Shooting Record

	A	B	C	D	E	F	G	H	I	J
A										
B										
C	C	✚	✚	✚	✚		★			
D		★		★						
E							★			
F										
G		★			✚	✚				
H							✚			
I				★			✚			
J				★	★	★				

Position Inference – Carrier

Battleship Rules
Shooting

Player's Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	s			
H			p				D			
I				★			D			
J				★	★	★	d			

Opponent's Shooting Record

	A	B	C	D	E	F	G	H	I	J
A										
B										
C	C	C	C	C	C		★			
D		★		★						
E							★			
F										
G		★			✚	✚				
H							✚			
I				★			✚			
J				★	★	★				

Shoot GG –

Battleship Rules
Shooting

Player's Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	S			
H			p				D			
I				★			D			
J				★	★	★	d			

Opponent's Shooting Record

	A	B	C	D	E	F	G	H	I	J
A										
B										
C	C	C	C	C	C		★			
D		★		★						
E							★			
F										
G		★			✚	✚	S			
H							✚			
I				★			✚			
J				★	★	★				

Shoot GG – "Sank Submarine"

Battleship Rules

Shooting

Player’s Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	S			
H			p				D			
I				★			D			
J				★	★	★	d			

Opponent’s Shooting Record

	A	B	C	D	E	F	G	H	I	J
A										
B										
C	C	C	C	C	C		★			
D		★		★						
E							★			
F										
G		★			✚	✚	S			
H							✚			
I				★			✚			
J				★	★	★				

Shoot JG –

Battleship Rules

Shooting

Player's Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	S			
H			p				D			
I				★			D			
J				★	★	★	D			

Opponent's Shooting Record

	A	B	C	D	E	F	G	H	I	J
A										
B										
C	C	C	C	C	C		★			
D		★		★						
E							★			
F										
G		★			✚	✚	S			
H							✚			
I				★			✚			
J				★	★	★	D			

Shoot JG – "Sank Destroyer"

Battleship Rules
Shooting

Player's Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	S			
H			p				D			
I				★			D			
J				★	★	★	D			

Opponent's Shooting Record

	A	B	C	D	E	F	G	H	I	J
A										
B										
C	C	C	C	C	C		★			
D		★		★						
E							★			
F										
G		★			✚	✚	S			
H							✚			
I				★			✚			
J				★	★	★	D			

Position Inference – Destroyer

Battleship Rules
Shooting

Player's Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	S			
H			p				D			
I				★			D			
J				★	★	★	D			

Opponent's Shooting Record

	A	B	C	D	E	F	G	H	I	J
A										
B										
C	C	C	C	C	C		★			
D		★		★						
E							★			
F										
G		★			✚	✚	S			
H							D			
I				★			D			
J				★	★	★	D			

Position Inference – Submarine

Battleship Rules
Shooting

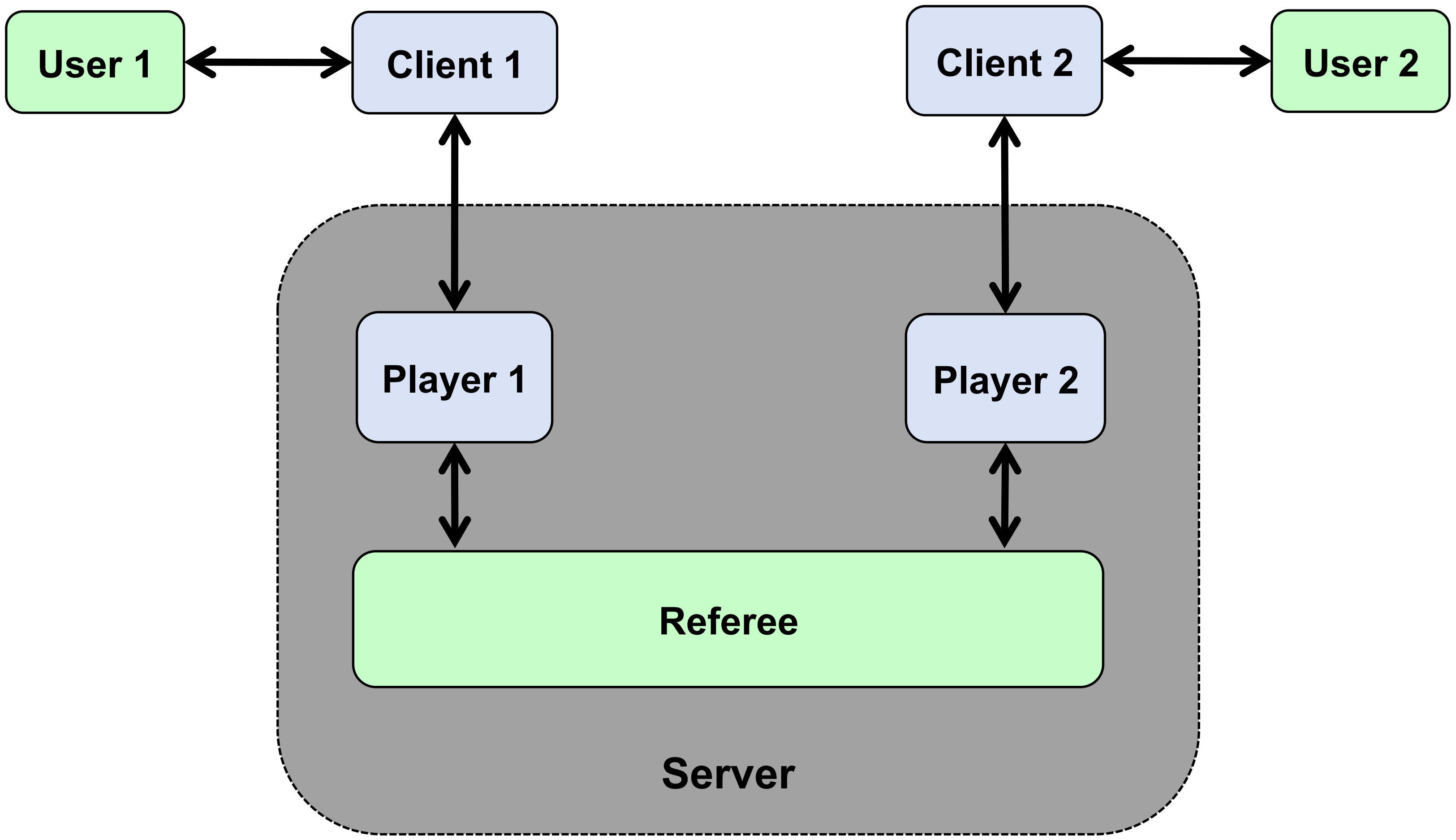
Player's Board

	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	S			
H			p				D			
I				★			D			
J				★	★	★	D			

Opponent's Shooting Record

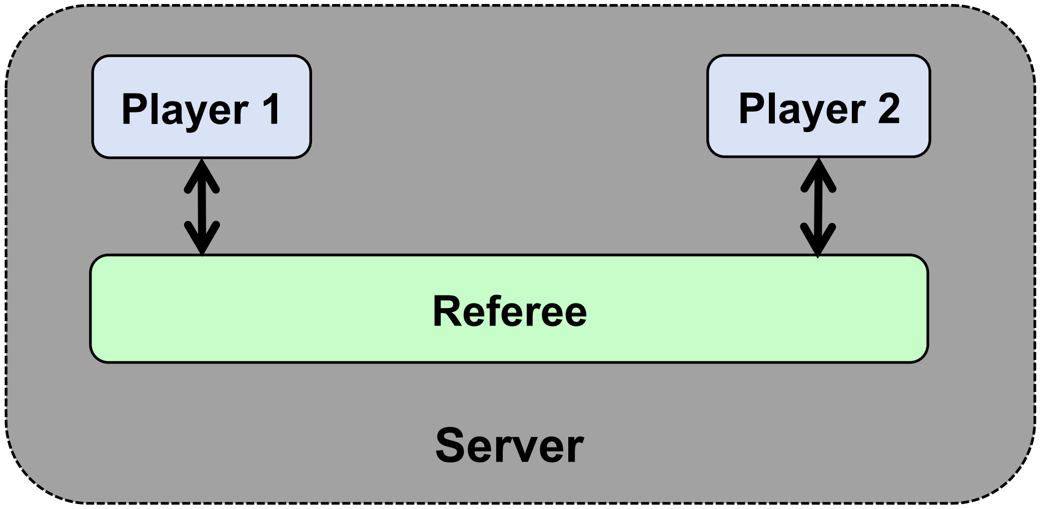
	A	B	C	D	E	F	G	H	I	J
A										
B										
C	C	C	C	C	C		★			
D		★		★						
E							★			
F										
G		★			S	S	S			
H							D			
I				★			D			
J				★	★	★	D			

Program Architecture

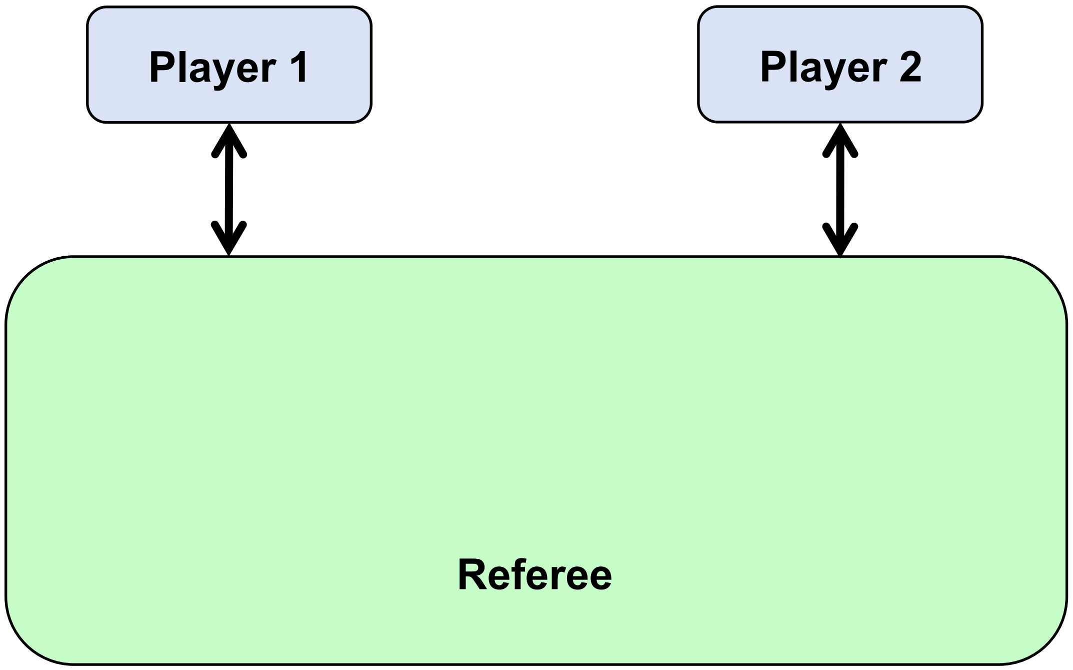


Whole Program Security

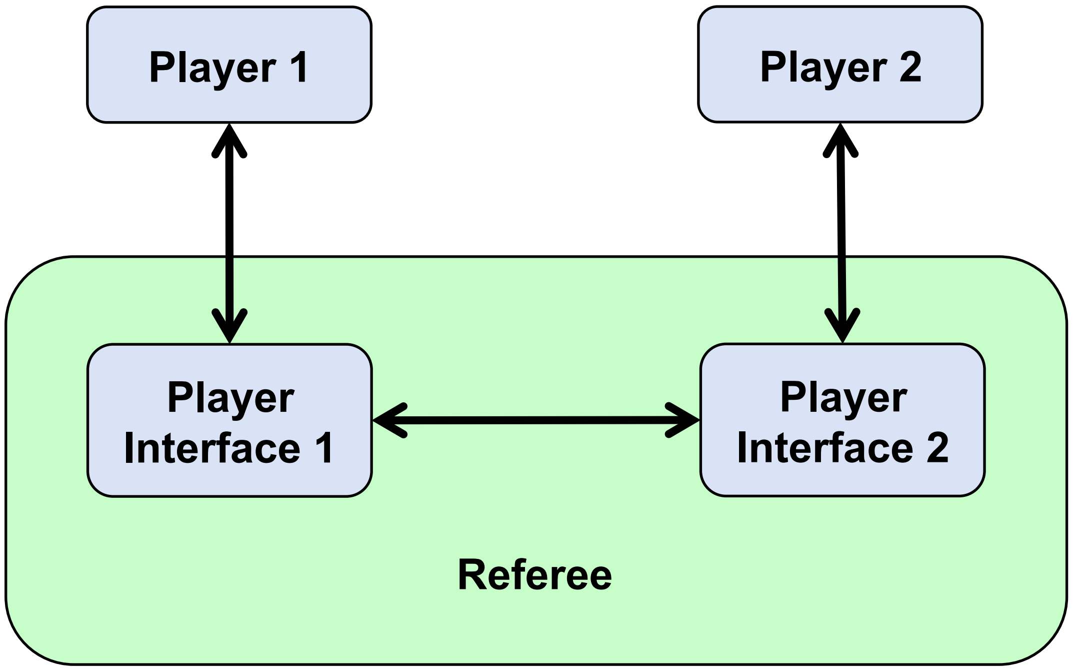
- A program is ***secure*** iff its referee is **indistinguishable** from a **model referee**, from the players' viewpoints
- Players are ***untrusted*** (need not be audited), except for check that they only communicate via interfaces



Splitting Referee into Mutually Distrustful Player Interfaces (PIs)



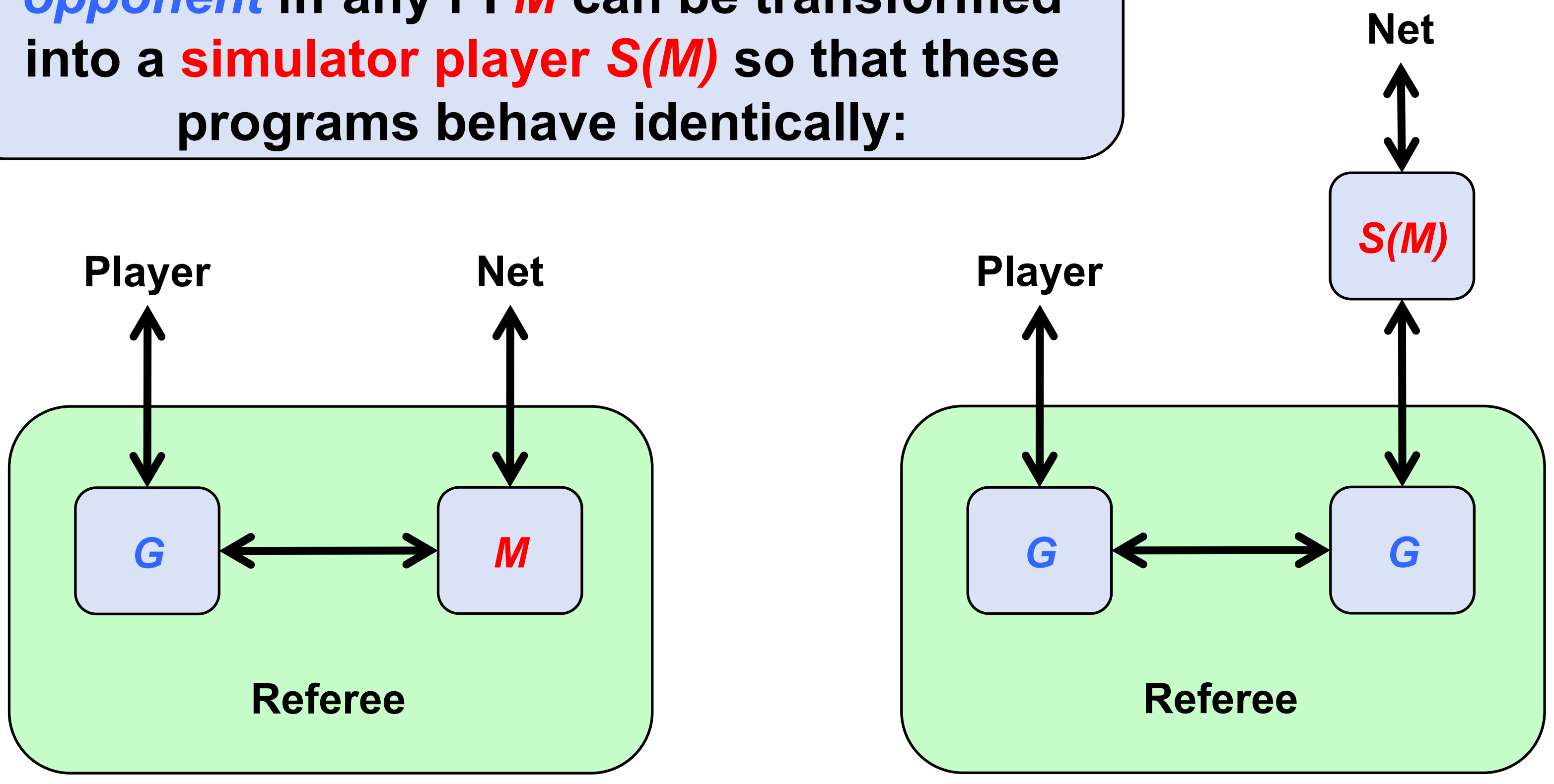
Splitting Referee into Mutually Distrustful Player Interfaces (PIs)



Our normal definition of security applies to a split referee, but we want also security against a malicious opponent PI

Security Against Malicious PI
Real/Ideal Paradigm

G is *secure against any malicious opponent* iff any PI *M* can be transformed into a simulator player *S(M)* so that these programs behave identically:



Real Program

Ideal Program

Implementations

- With a colleague and interns at MIT Lincoln Laboratory, implemented Battleship in Haskell/LIO
  - IFC library built on top of Safe Haskell by David Mazières's group at Stanford
  - Our use of IFC really amounted to access control (AC)
- Implemented in Concurrent ML (CML) using AC
  - I'm going to tell you about the CML + AC version

CML + AC Battleship

- **PIs exchange — using trusted code — immutable, abstract *locked boards*, whose cells can be unlocked using unforgeable *keys* held by originating player:**

```
type key (* key *)
type ck  (* counted key *)
val labelKey : key * int -> ck
type lb  (* locked board *)
datatype lsr =
           Invalid        (* invalid counted key *)
         | Repeat         (* illegal repetition *)
         | Miss           (* missed a ship *)
         | Hit            (* hit an unspecified ship *)
         | Sank of ship   (* sank the given ship *)
val lockedShoot : lb * pos * ck -> lb * lsr
```

CML + AC Example

PI 1

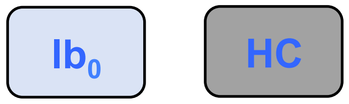
PI 2

CML + AC Example

CML + AC Example

PI 1

PI 2



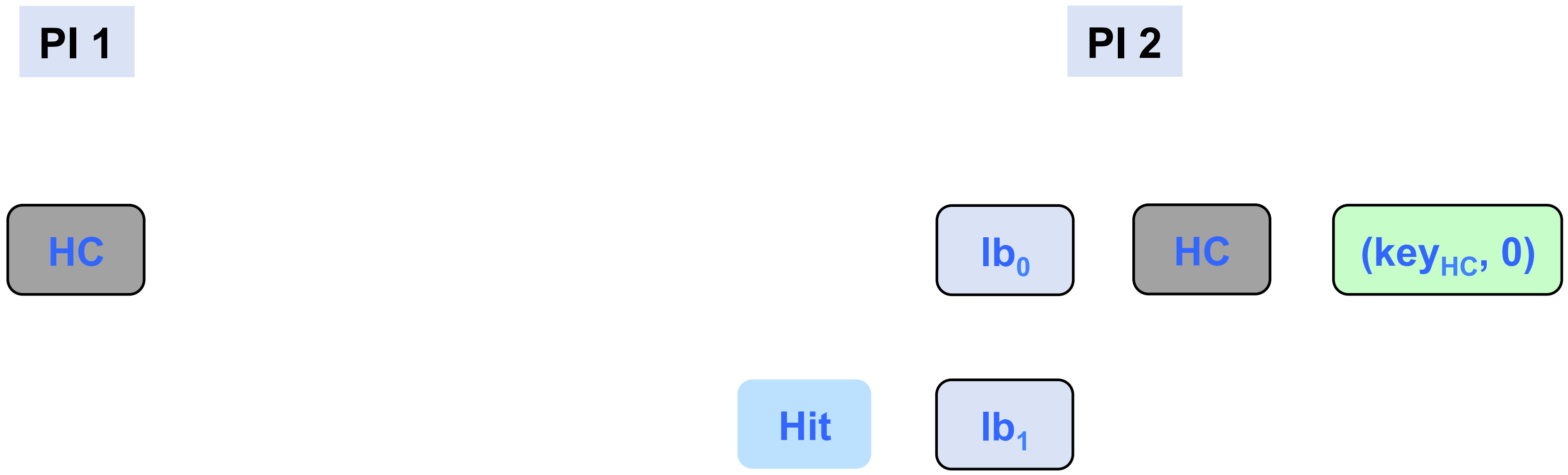
CML + AC Example

PI 1
PI 2
HC
Ib_0
HC

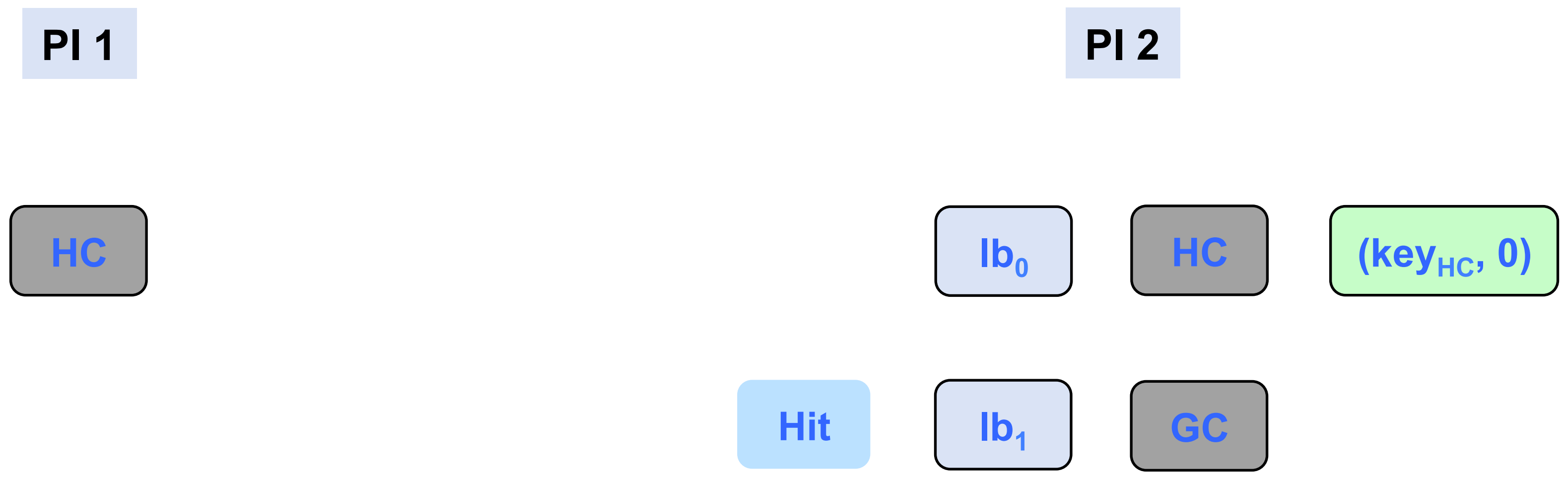
CML + AC Example

PI 1
PI 2
HC
lb_0
HC
$(key_{HC}, 0)$

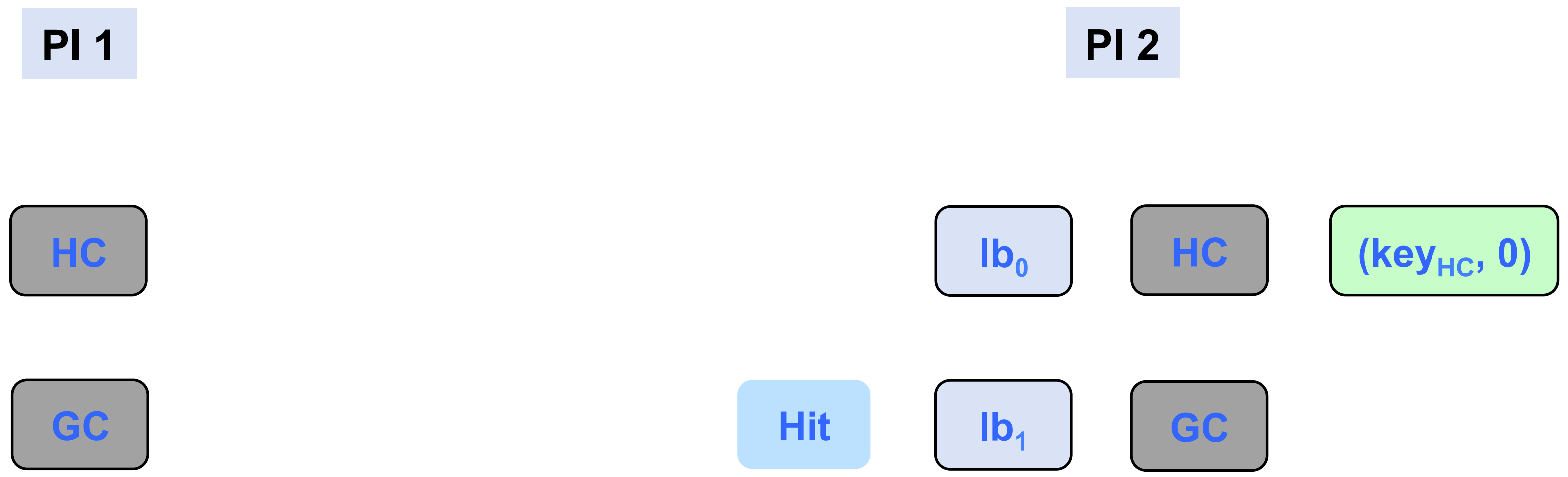
CML + AC Example



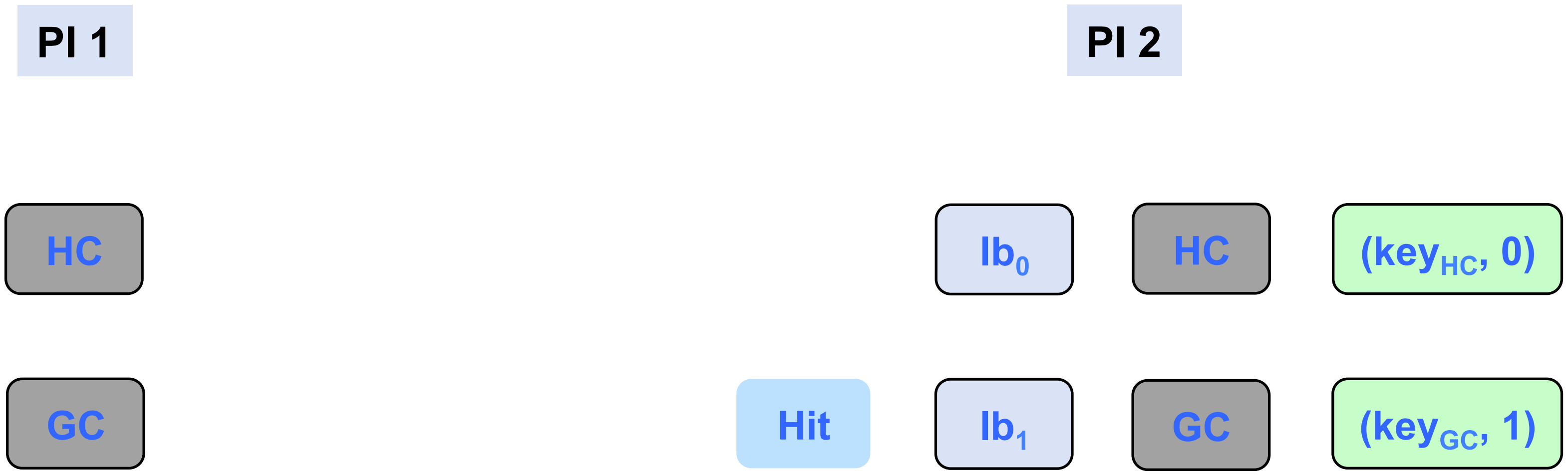
CML + AC Example



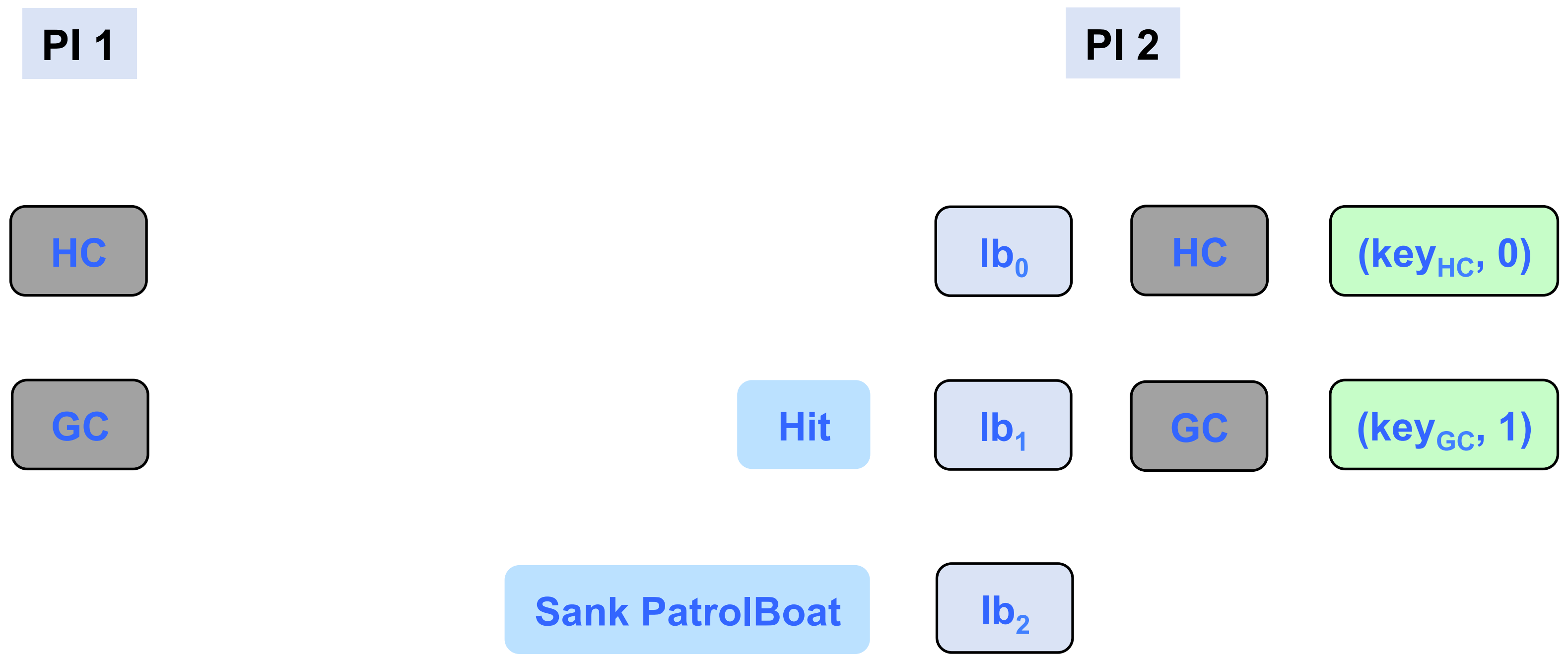
CML + AC Example



CML + AC Example

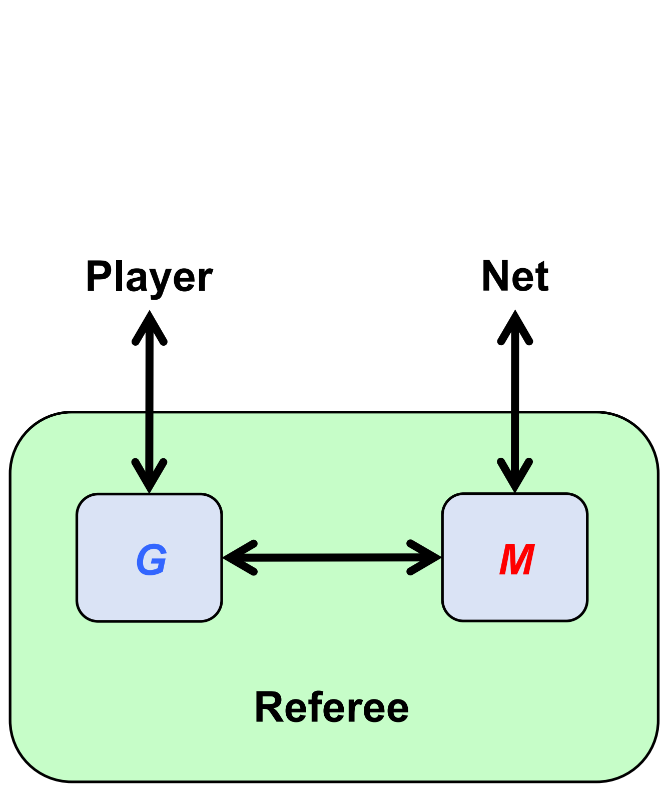


CML + AC Example

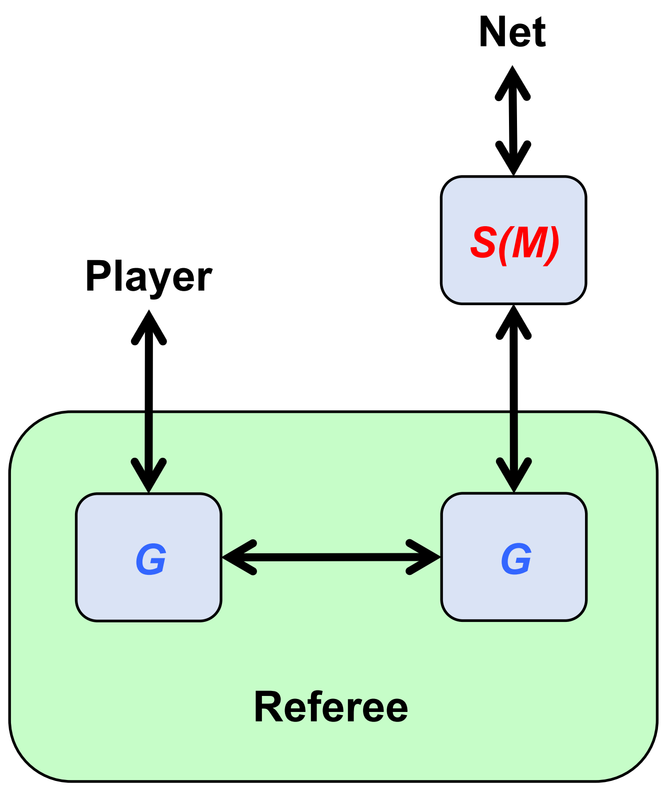


A counted key is only applicable to a single locked board, and can’t be deconstructed

Construction of Simulator Player for CML + AC

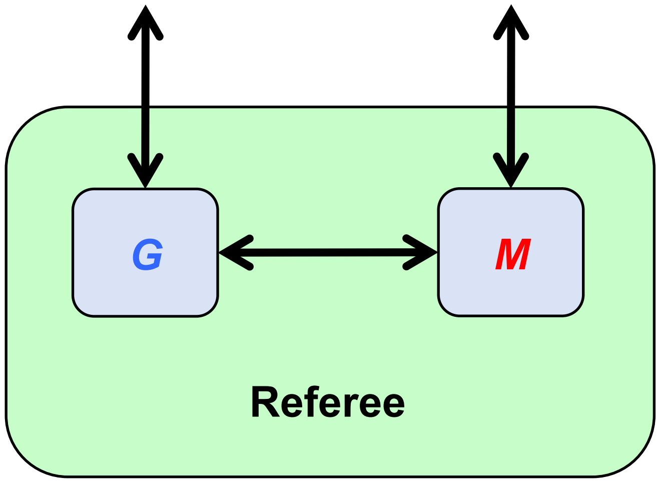


Real Program

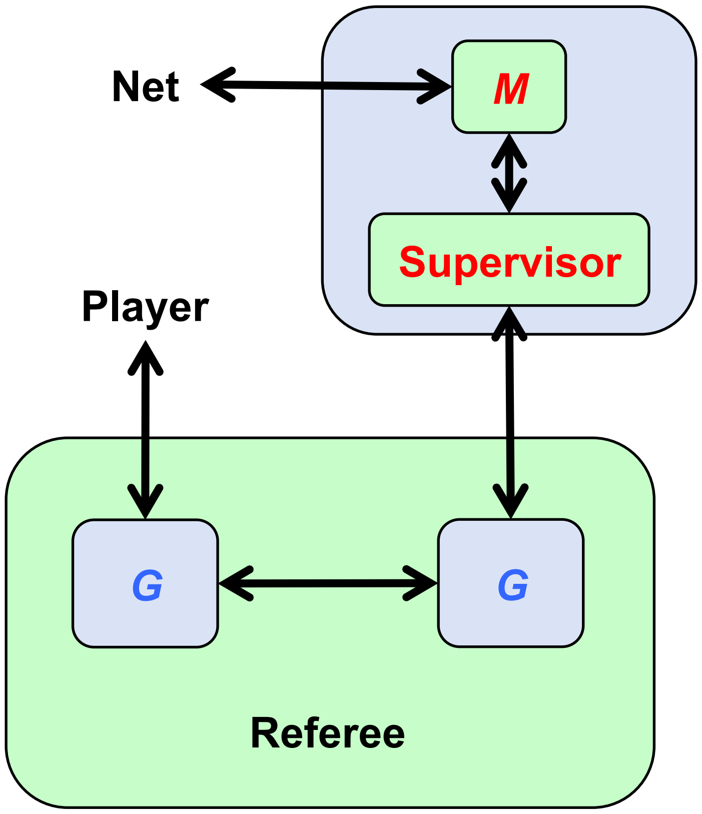


Ideal Program

Construction of Simulator Player

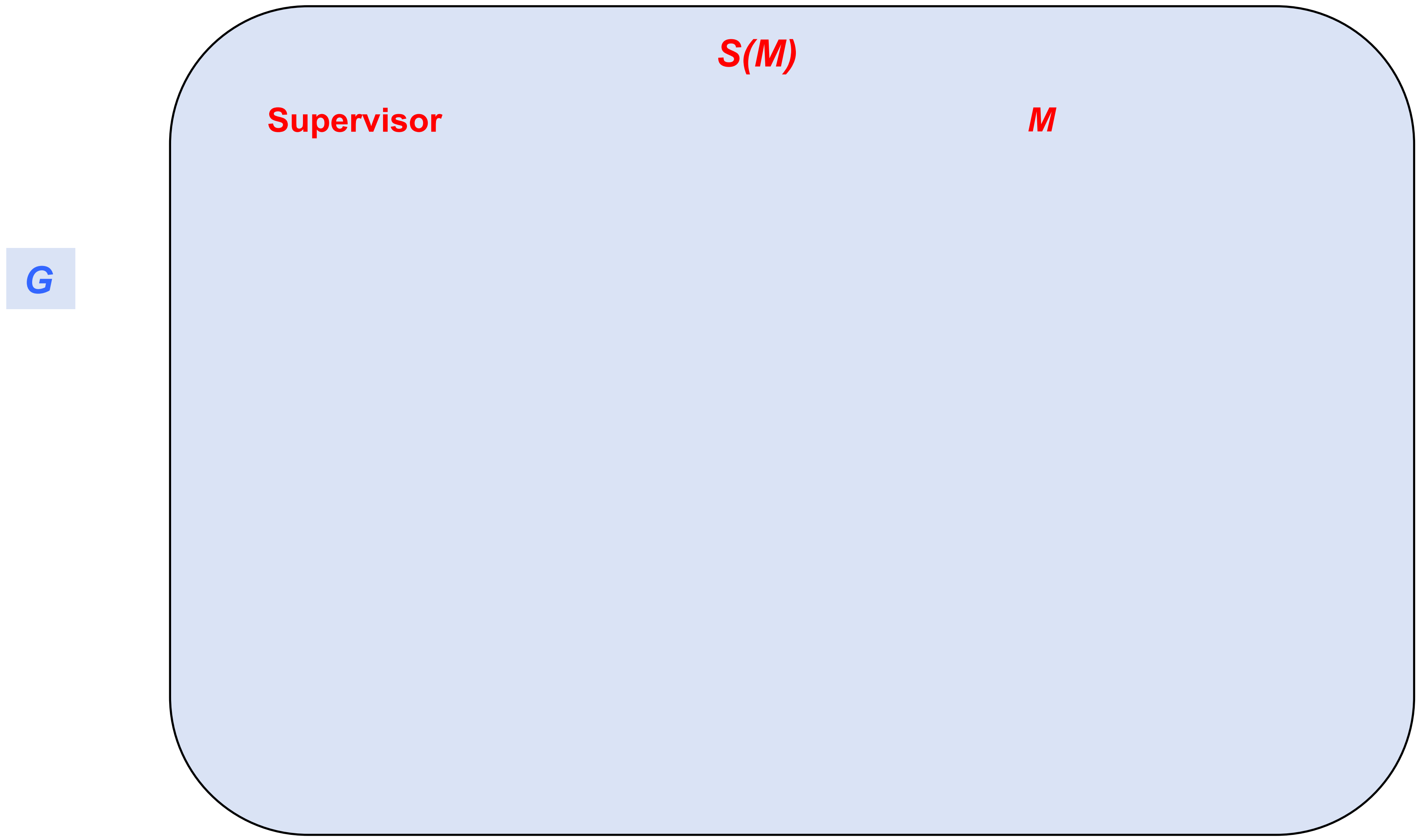


Real Program

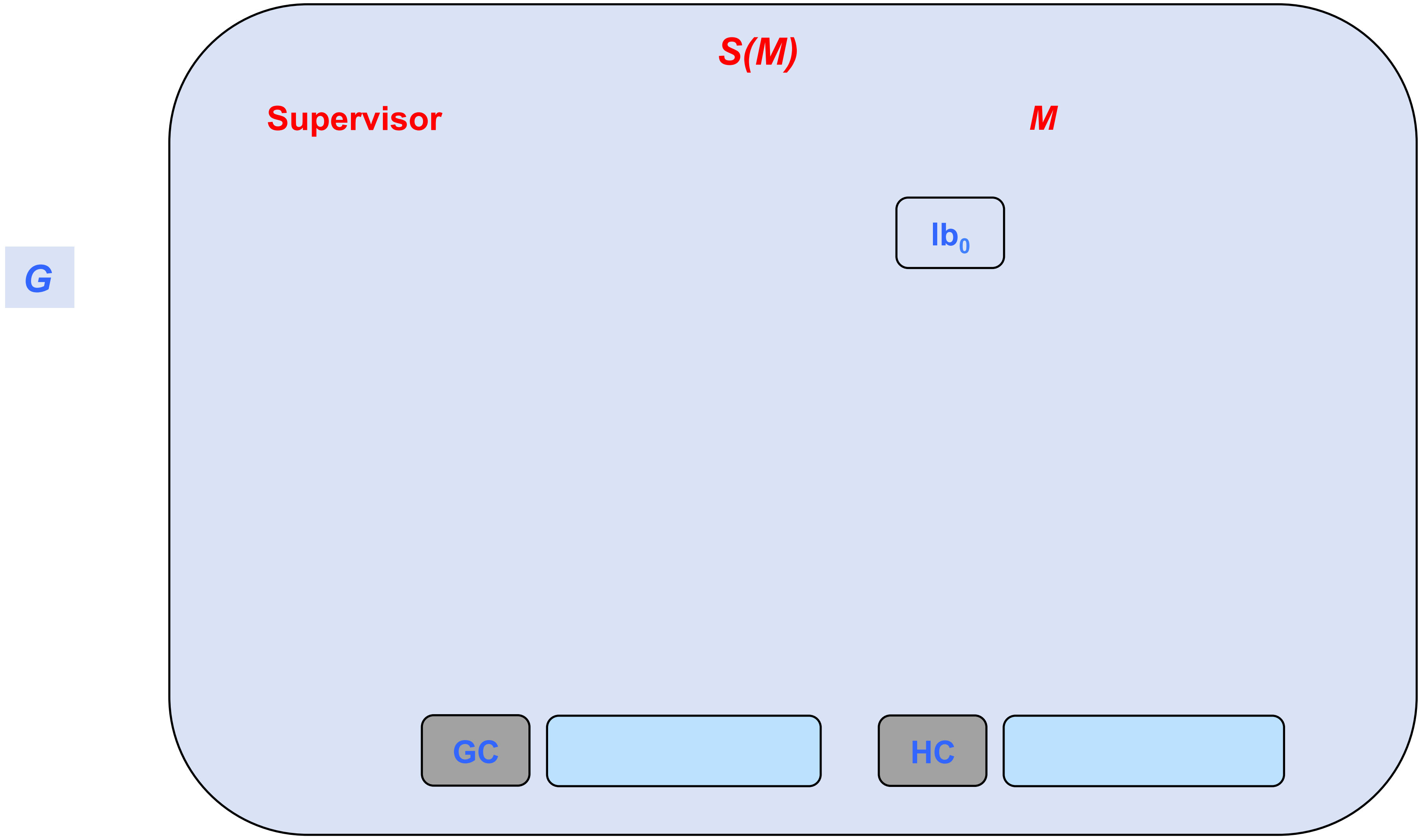


Ideal Program

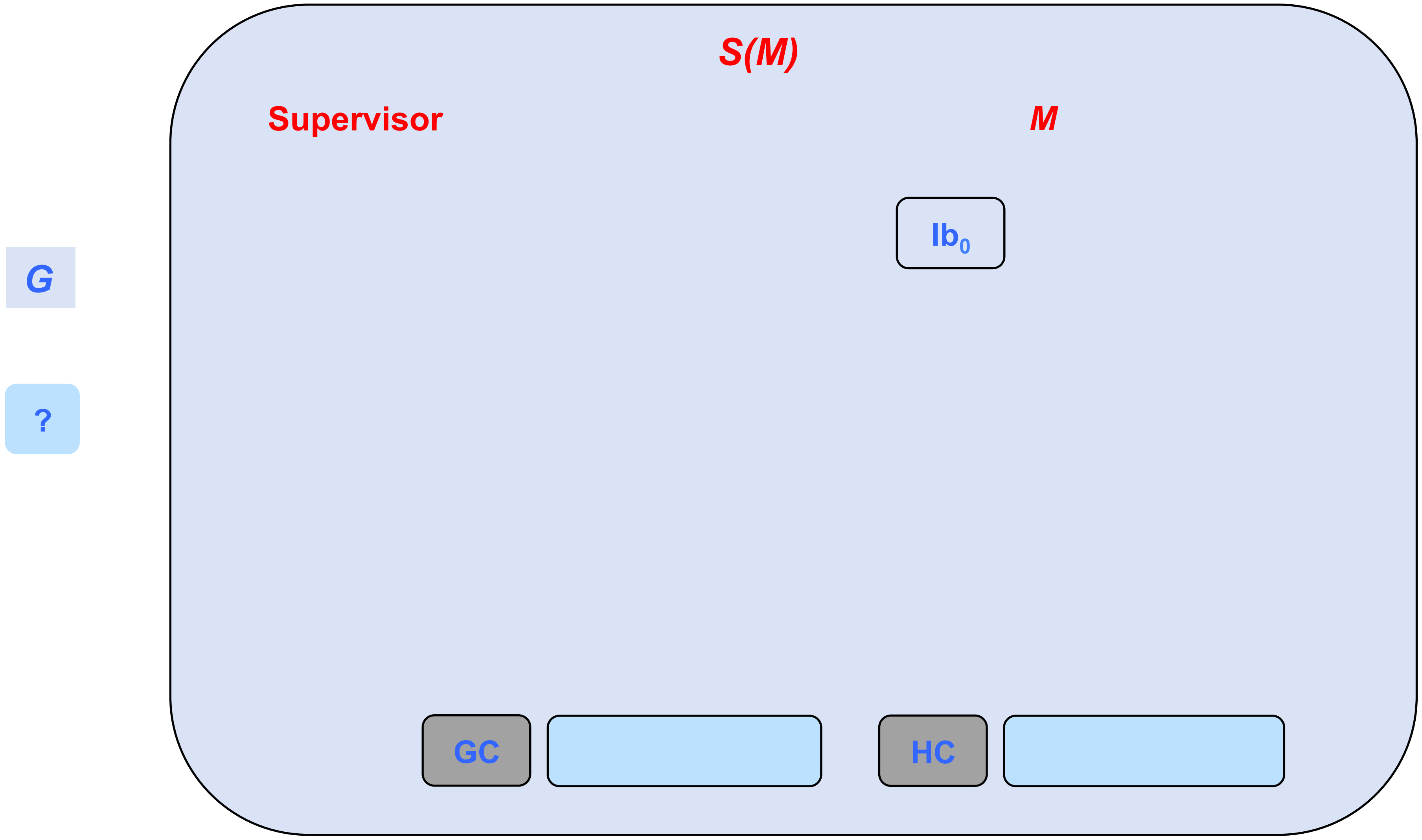
CML + AC Simulator Example



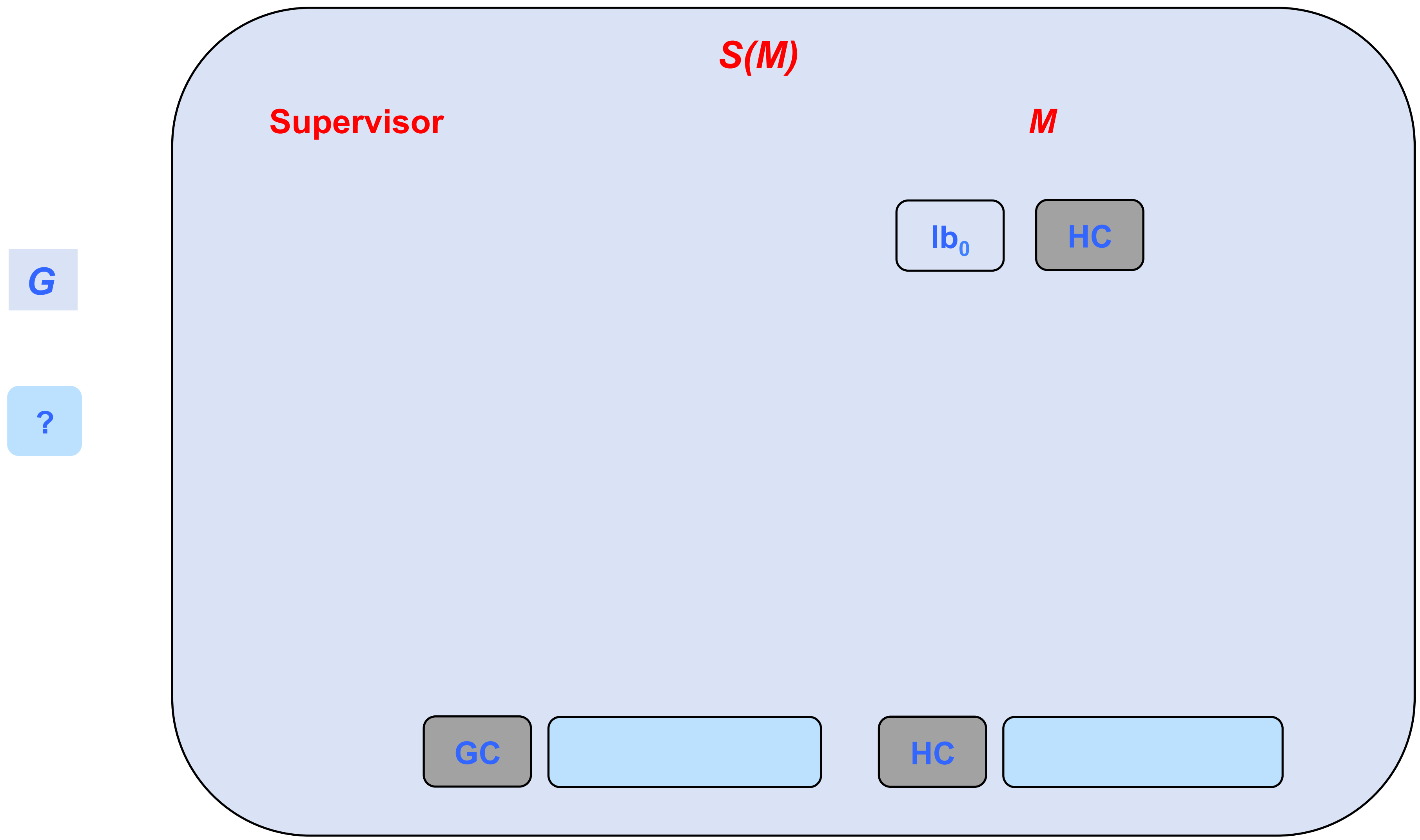
CML + AC Simulator Example



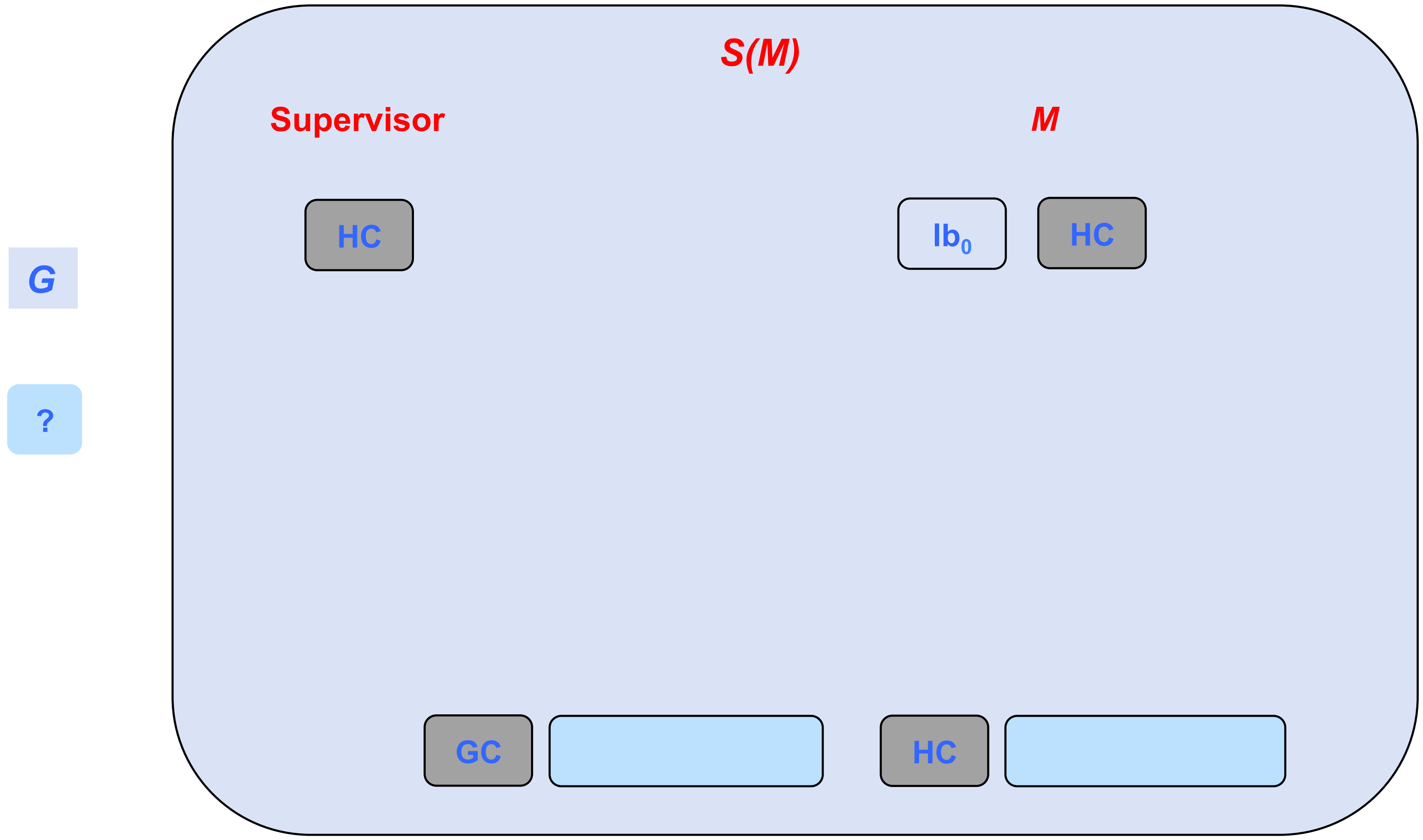
CML + AC Simulator Example



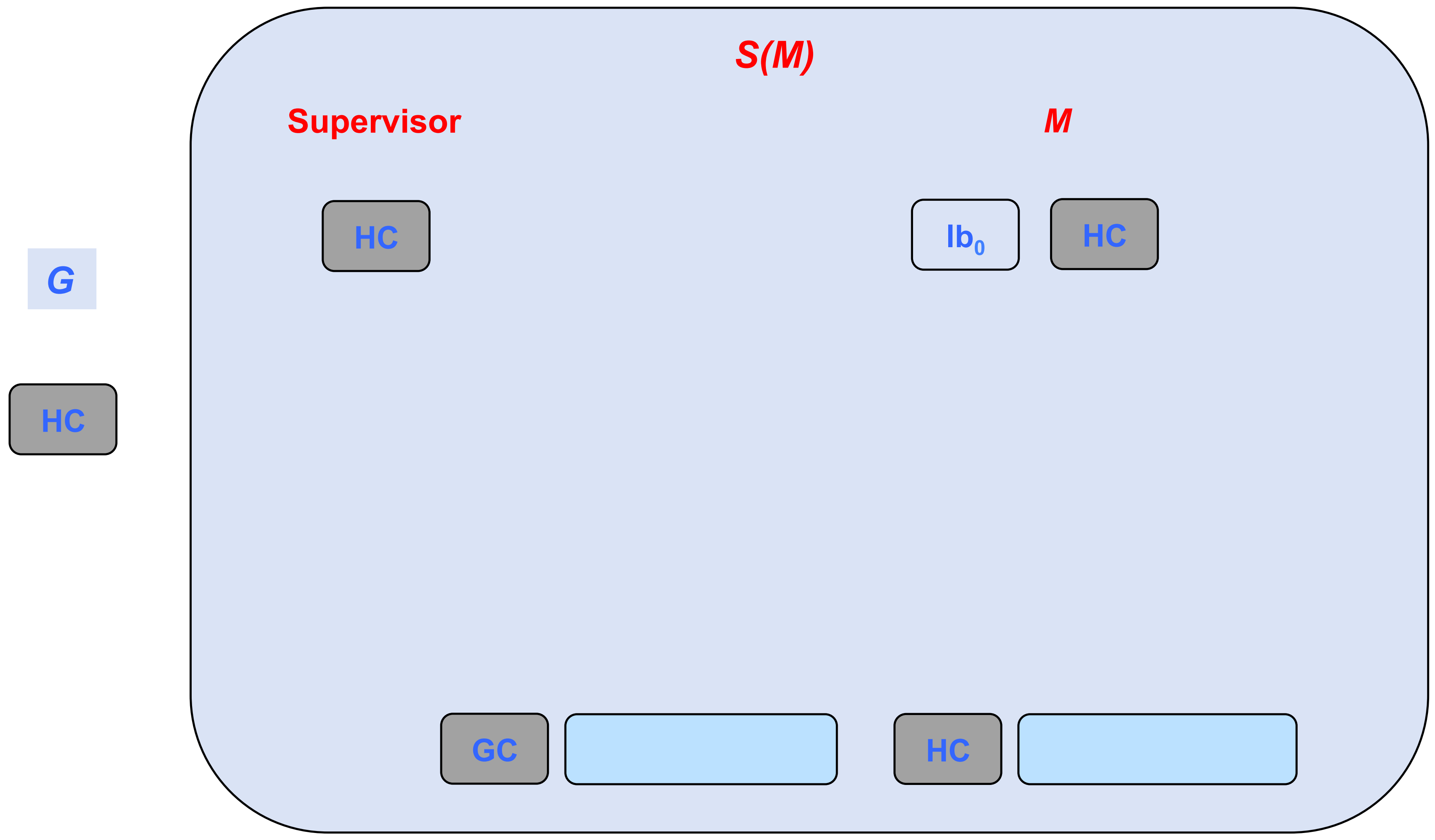
CML + AC Simulator Example



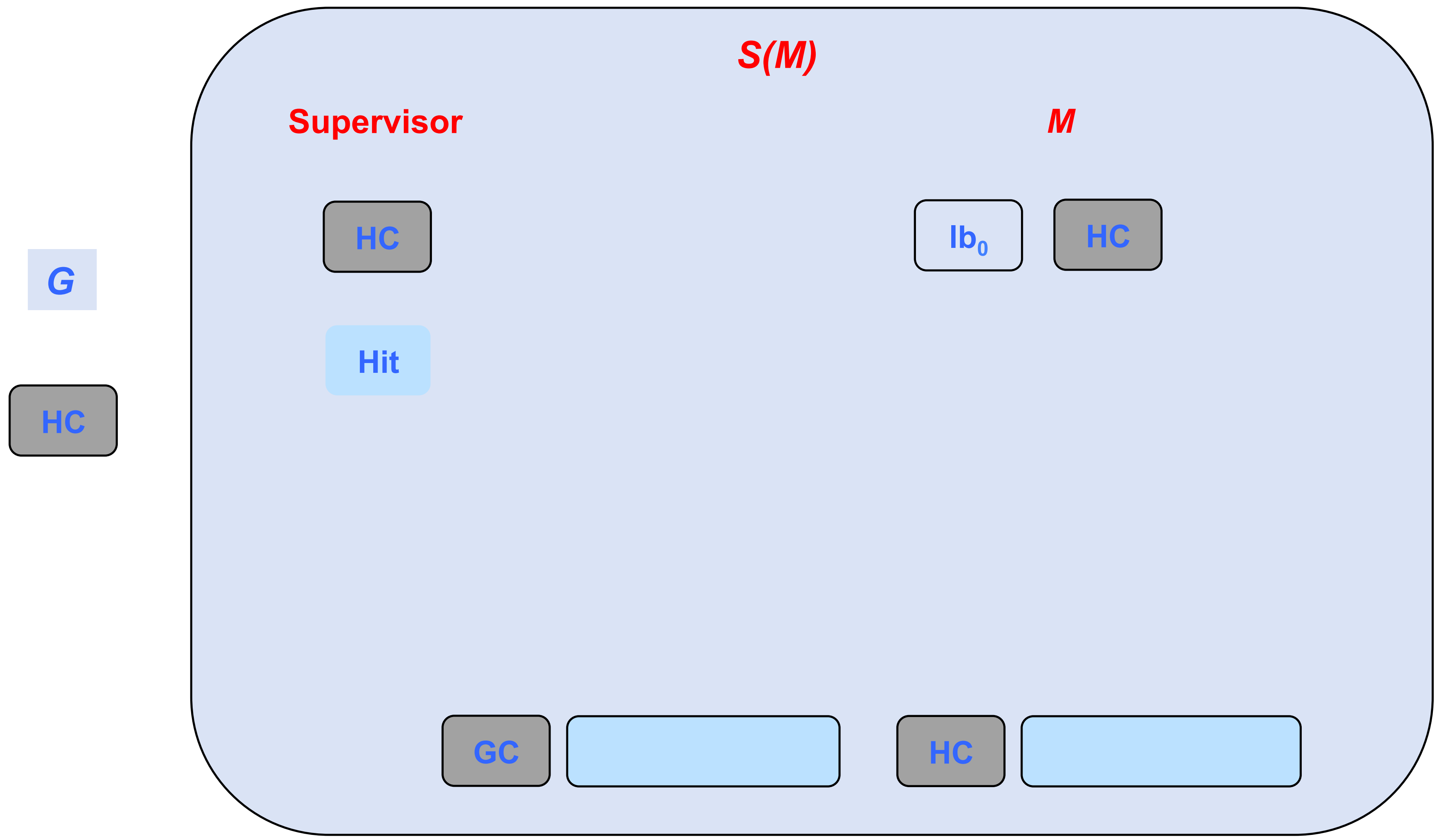
CML + AC Simulator Example



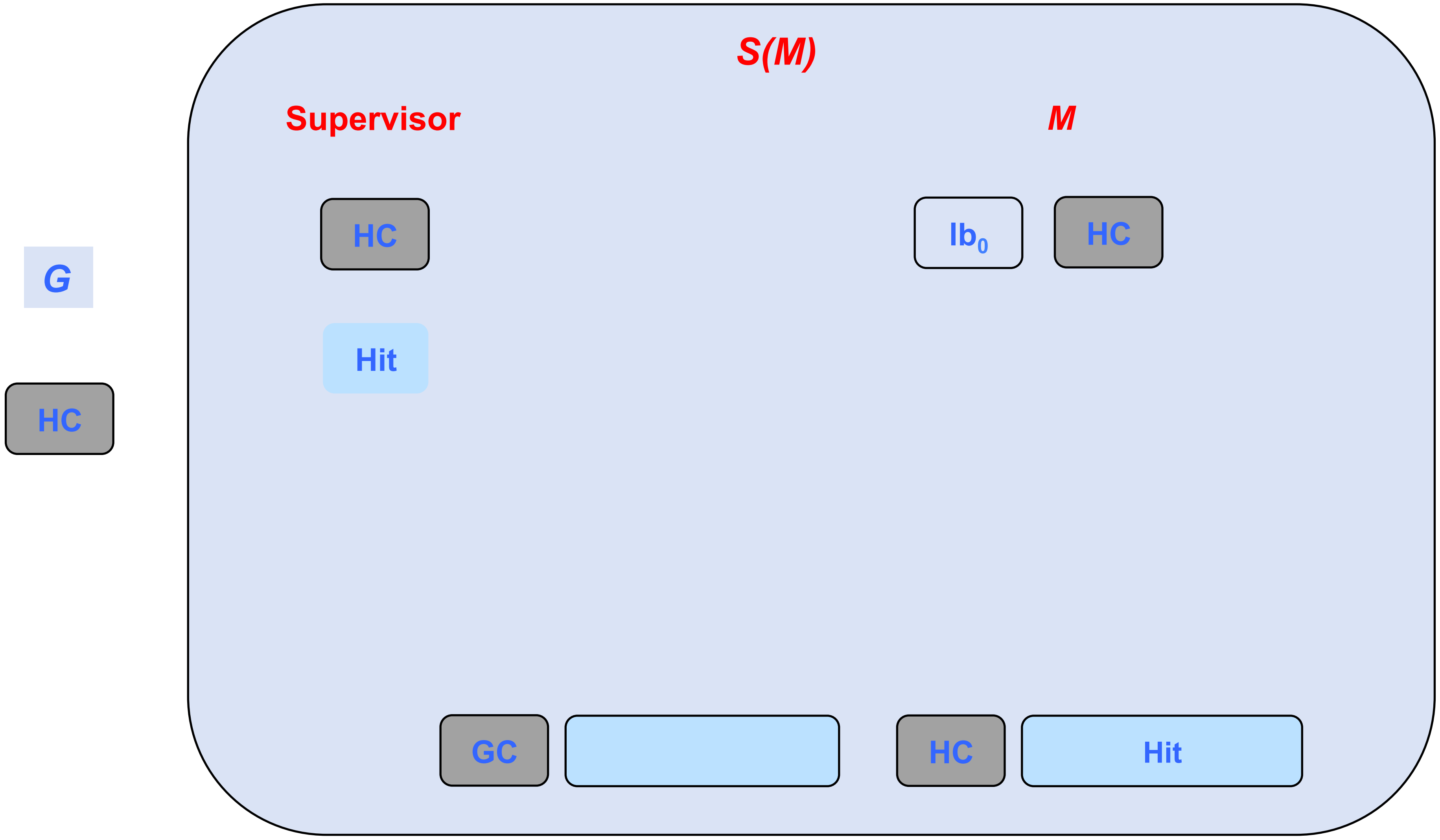
CML + AC Simulator Example



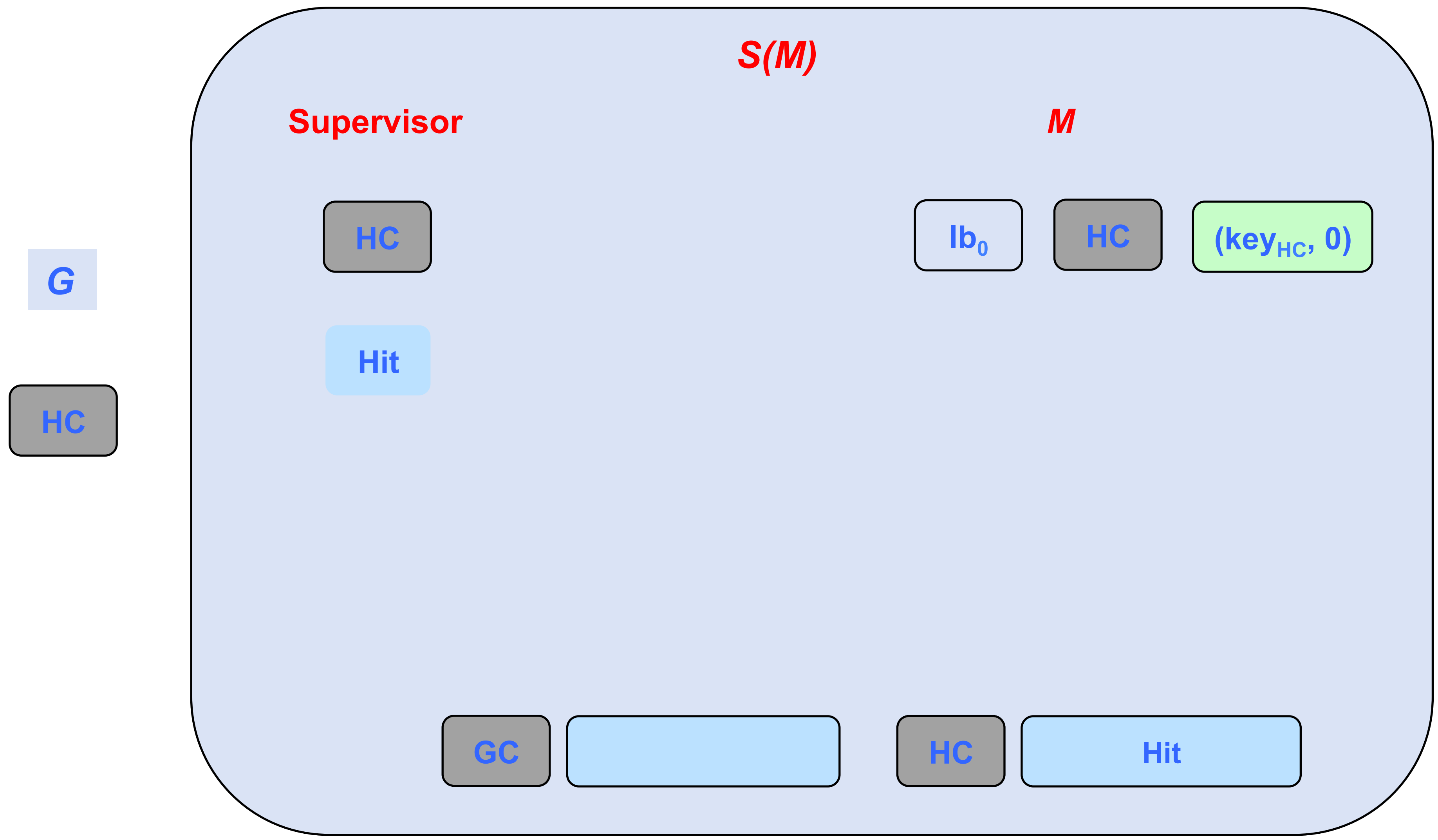
CML + AC Simulator Example



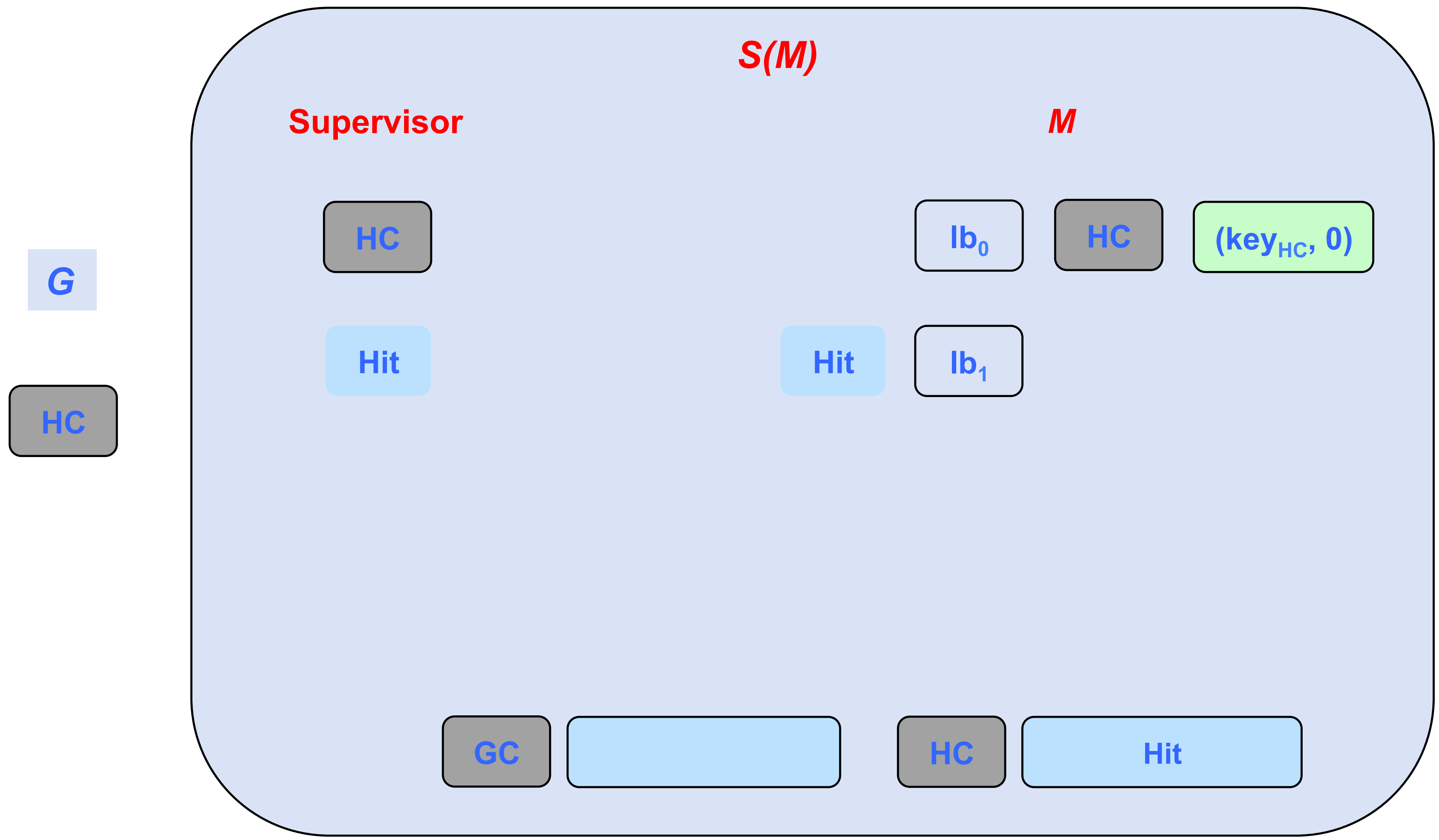
CML + AC Simulator Example



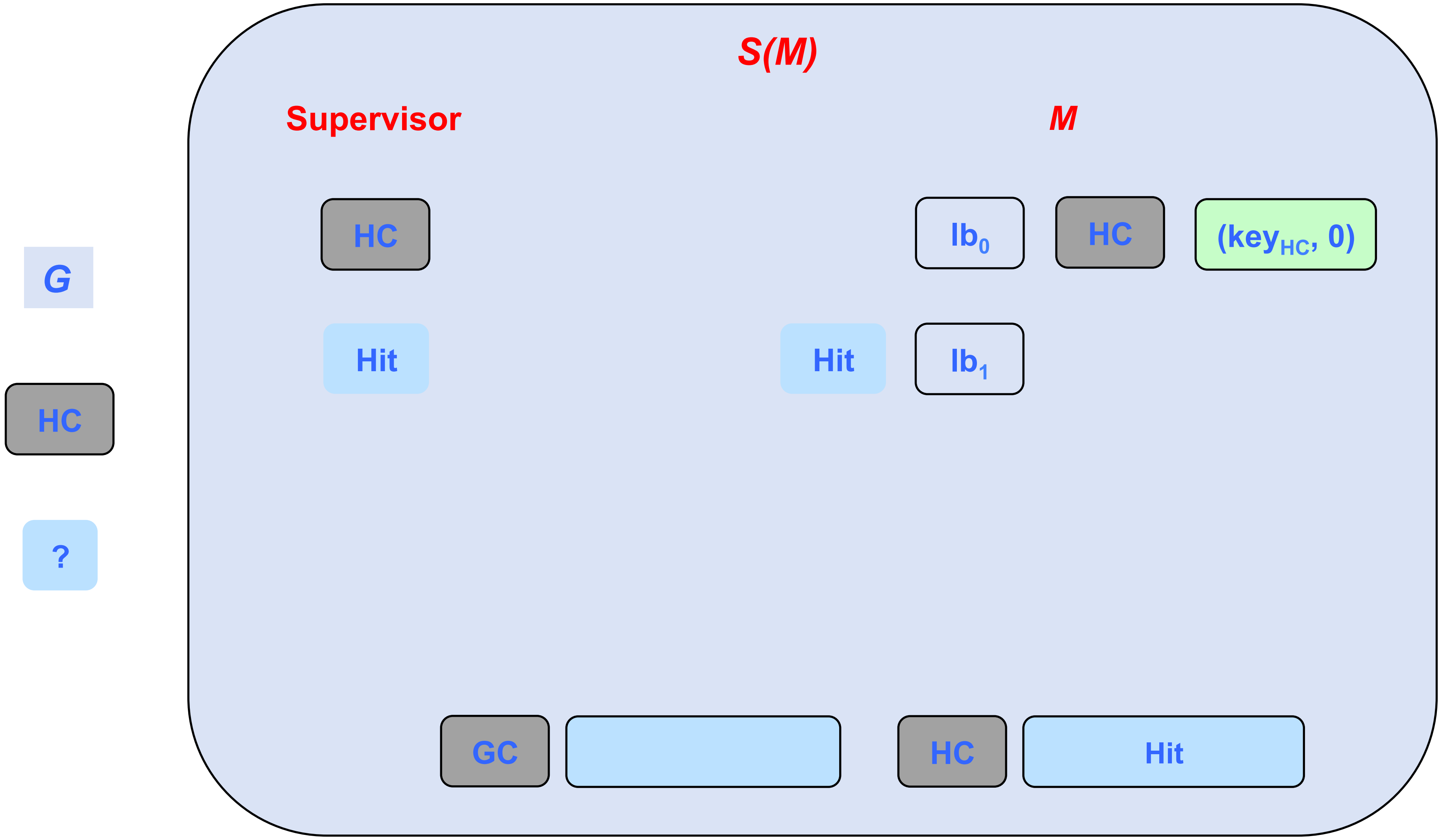
CML + AC Simulator Example



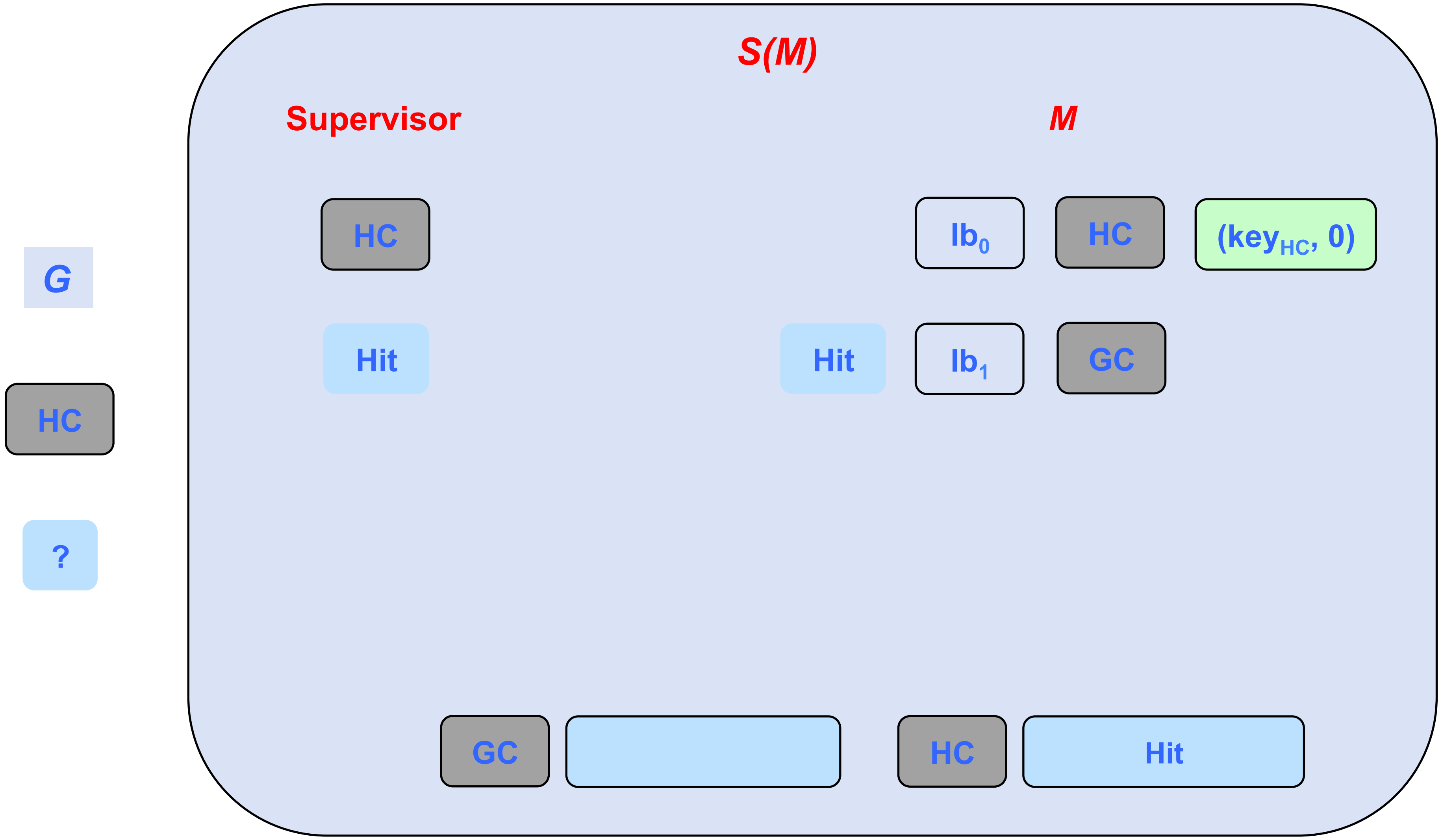
CML + AC Simulator Example



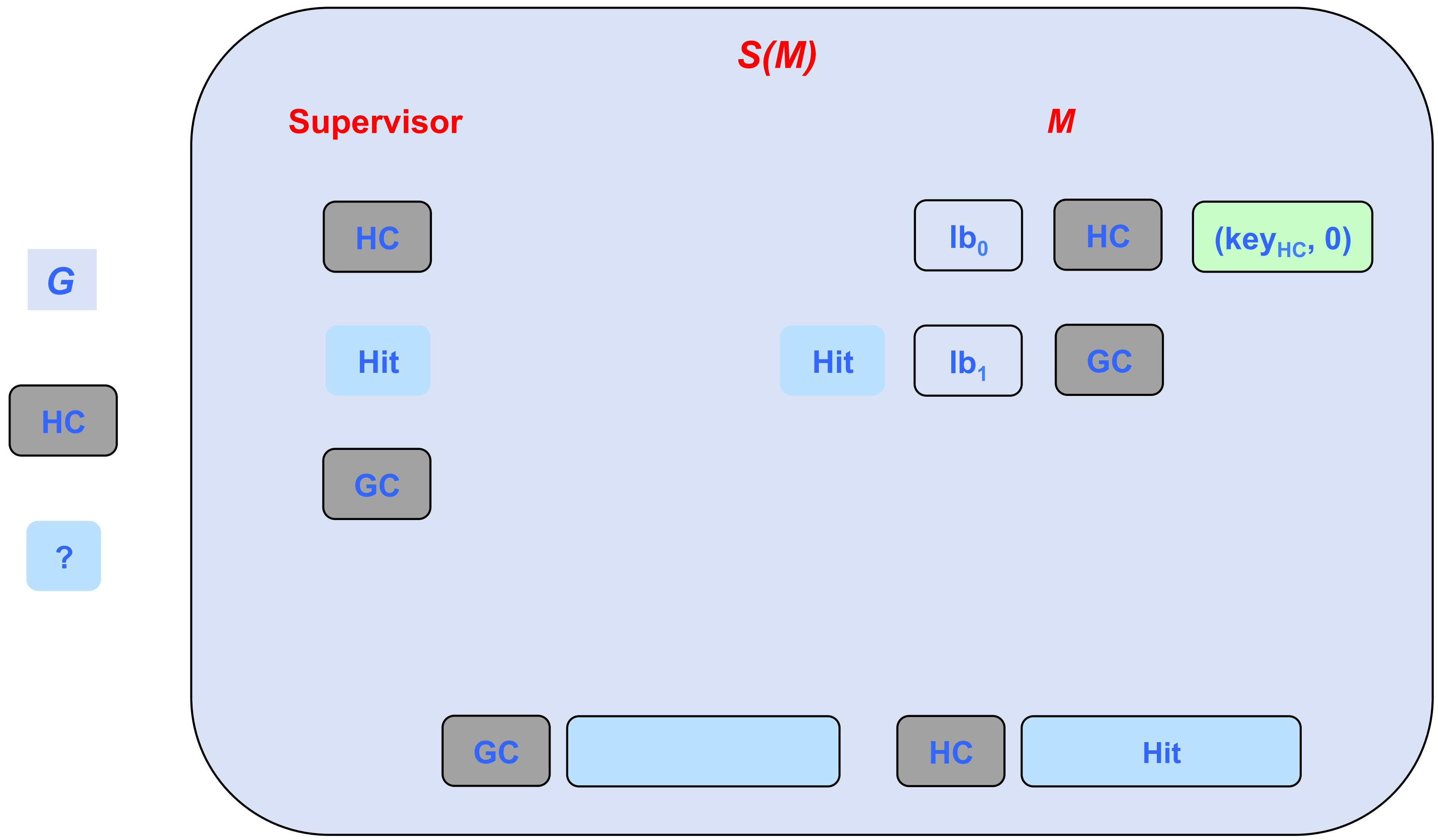
CML + AC Simulator Example



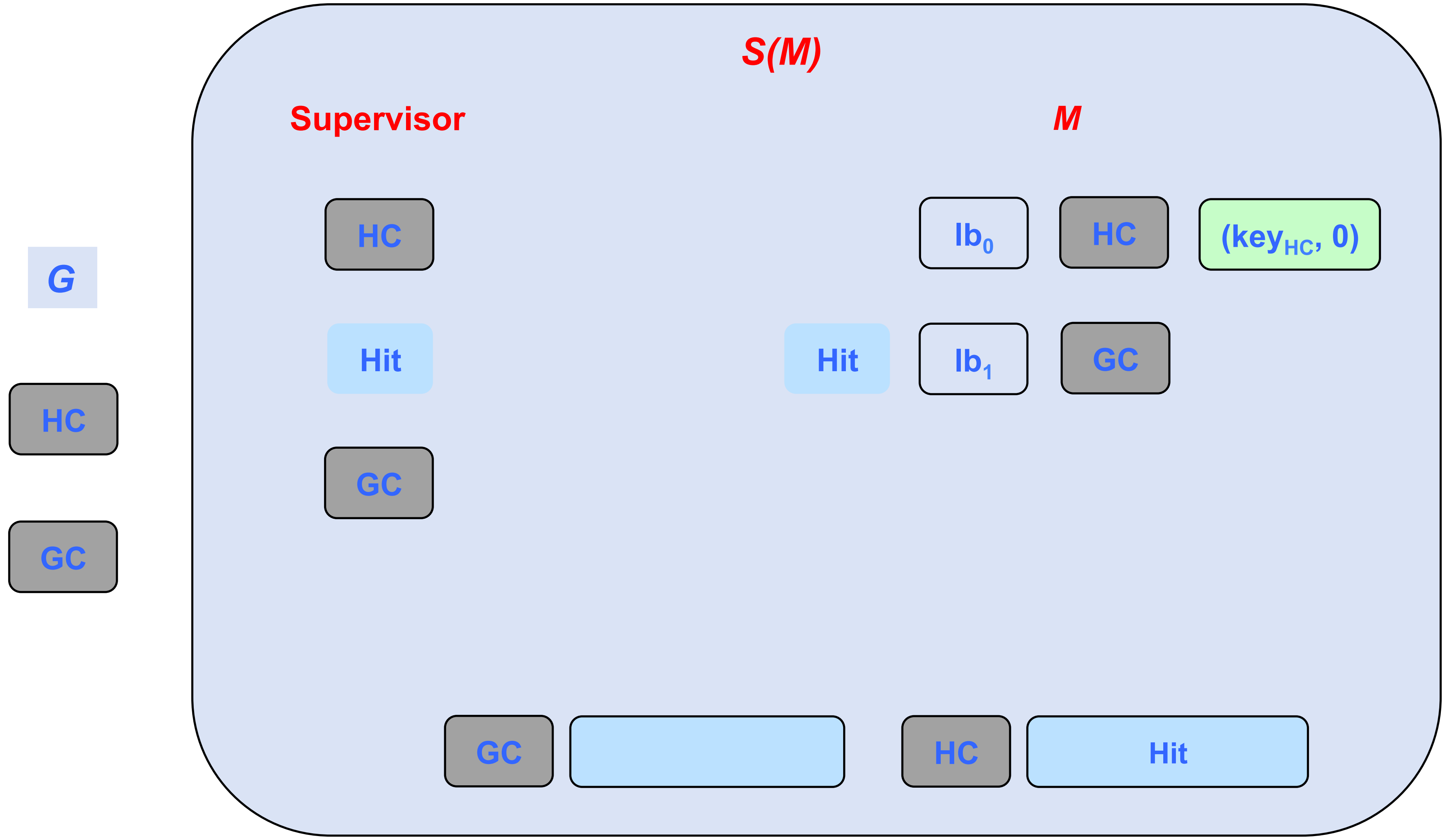
CML + AC Simulator Example



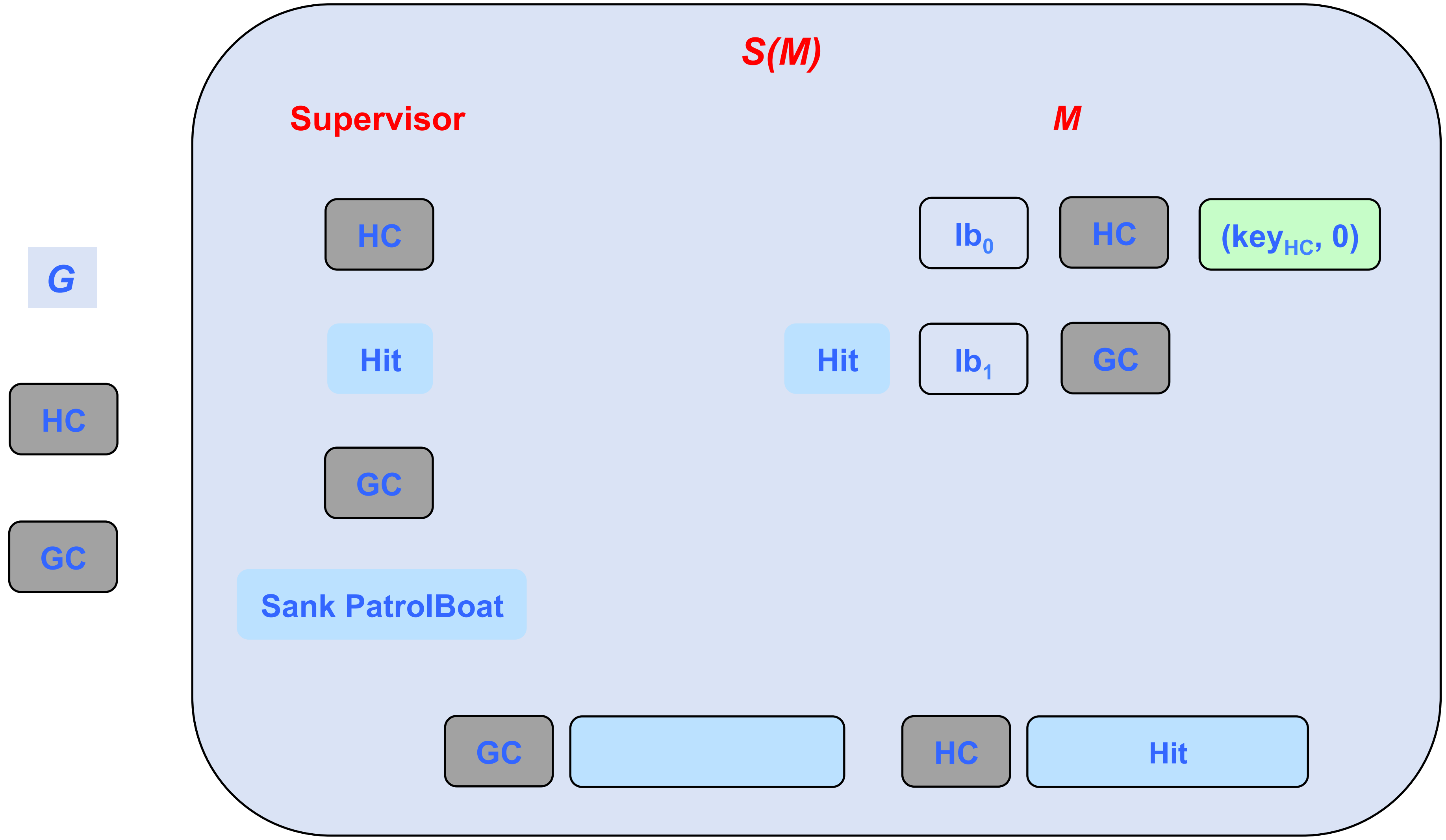
CML + AC Simulator Example



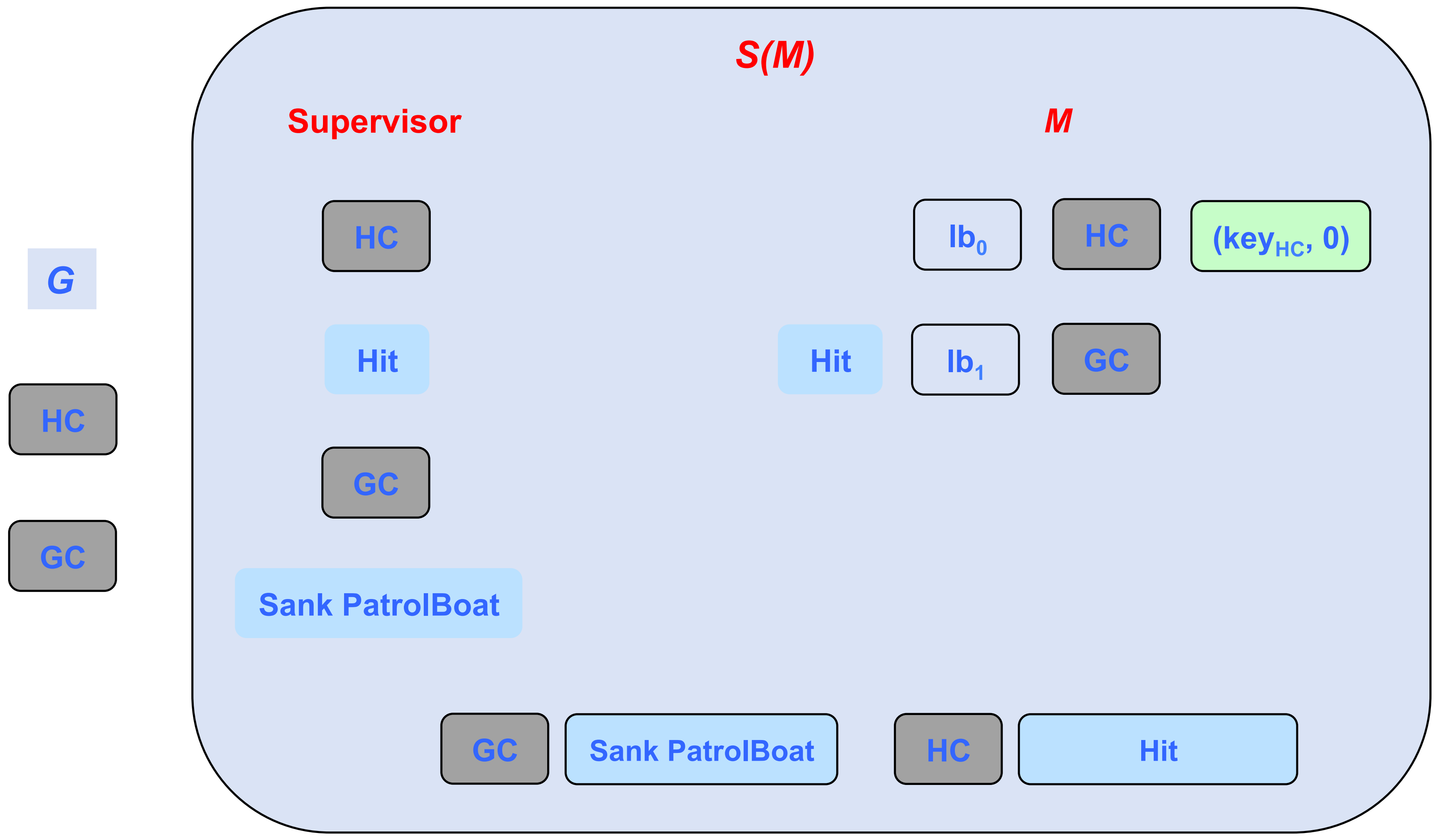
CML + AC Simulator Example



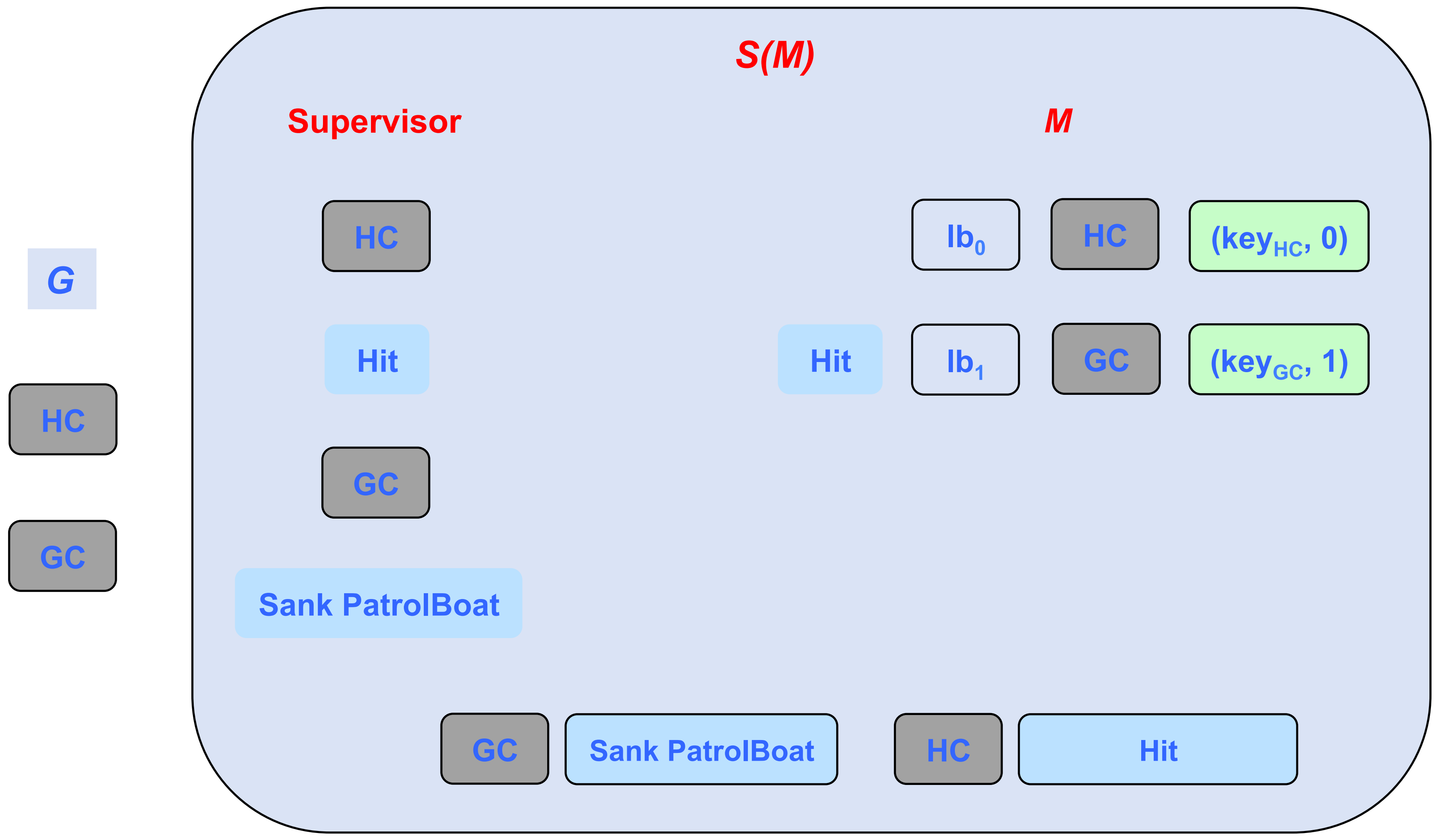
CML + AC Simulator Example



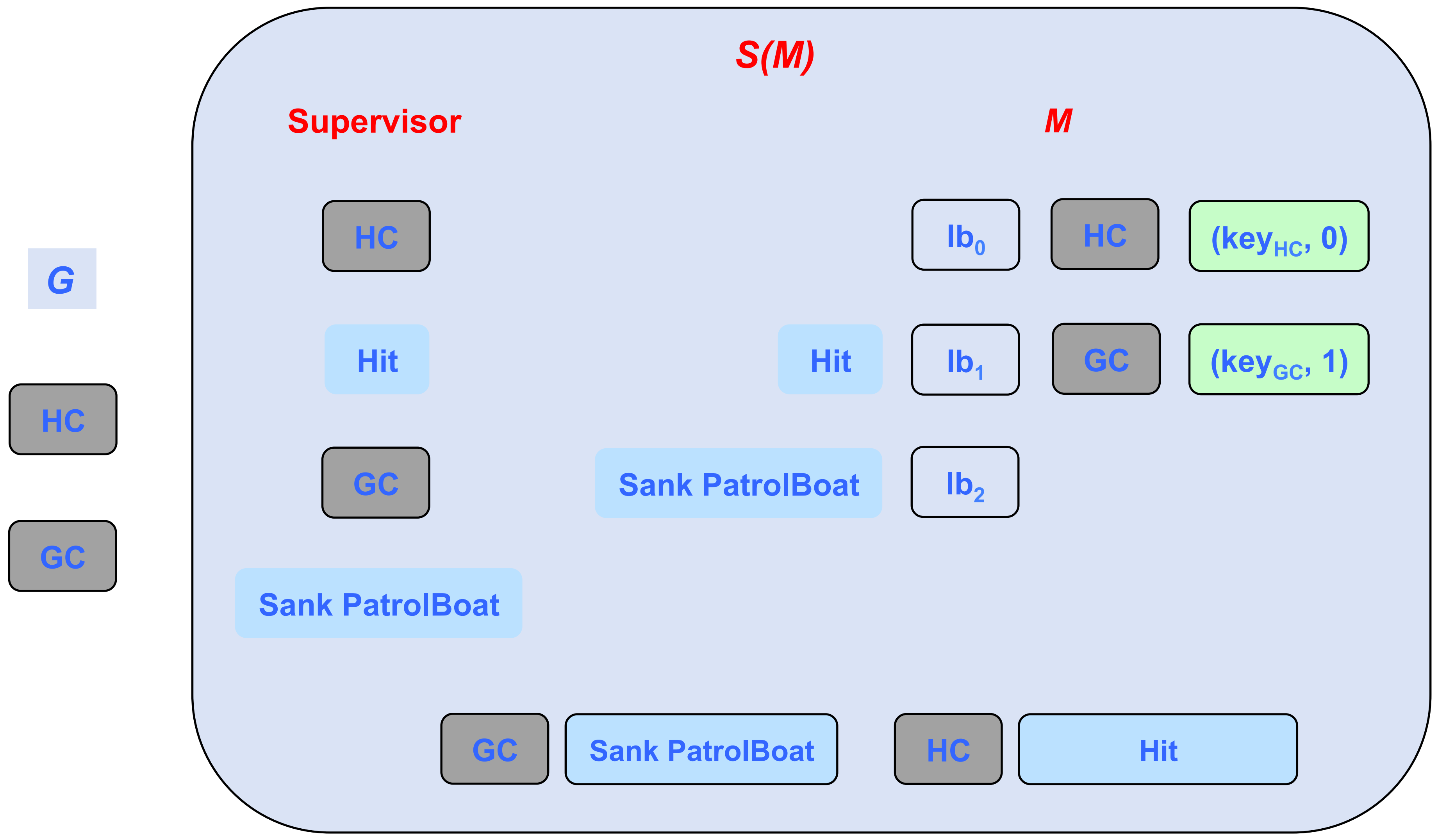
CML + AC Simulator Example



CML + AC Simulator Example



CML + AC Simulator Example



Future Work

- I plan to prove in Coq that both the LIO and CML Battleship implementations are secure
  - Whole program security — *G* composed with itself works as should model referee
  - Security against a malicious Pl — need to show that simulator works correctly for all *M*
  - Ideally start with pre-existing Coq formalization of typed language with both immutable and mutable data structures — suggestions?
- Want to understand how generally applicable the real/ideal paradigm is to ordinary program security
  - How far can TCBs be reduced?