CS 599: Formal Methods in Security and Privacy

Applying Real/Ideal Paradigm to Programming Language-Based Security

Marco Gaboardi
gaboardi@bu.edu

Alley Stoughton
stough@bu.edu

Two Circa 2013 Security Projects at MIT Lincoln Laboratory

RESEARCH-ARTICLE

You Sank My Battleship!: A Case Study in Secure Programming

Authors: Alley Stoughton, Andrew Johnson, Samuel Beller, Karishma Chadha, Dennis Chen, Kenneth Foner, Michael Zhivich Authors Info & Claims

PLAS'14: Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security • July 2014 • Pages 2–14 • https://doi.org/10.1145/2637113.2637115

Published: 28 July 2014 Publication History

Evaluation of Information Flow Control Programming Languages

Home / Proceedings / CSF / CSF 2017

2017 IEEE 30th Computer Security Foundations Symposium (CSF)

Mechanizing the Proof of Adaptive, Information-Theoretic Security of Cryptographic Protocols in the Random Oracle Model

Year: 2017, Pages: 83-99
DOI Bookmark: 10.1109/CSF.2017.36

Authors

Alley Stoughton
Mayank Varia

Formalization in EasyCrypt of Security Proofs for Cryptographic Protocols

Approaches to Secure Programming

- Information Flow Control (IFC)
  - Restricts flow of data, preventing more-classified (lower-integrity) data from influencing less-classified (higher-integrity) results — unless necessary privileges are used
- Access Control (AC)
  - Restricts data access to components holding necessary privileges, without controlling what may happen to data once it is accessed
- Data Abstraction
  - Maintain invariants and limit views of / access to data
  - Can use to implement AC and IFC

Defining Program Security

- Surprisingly little work on specifying program security
  - More specific than noninterference theorems
- 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 to 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 Case Study

- This talk uses a case study involving the two-player board game Battleship to illustrate how security definitions can be separated from enforcement
- Precise definitions of security:
  - Whole program security
  - Security of one player against another — borrowing real/ideal paradigm of theoretical cryptography
- Three Battleship implementations:
  - One in Concurrent ML (CML) with trusted referee
  - One in LIO/Haskell using IFC to avoid need for trusted referee
  - One in CML using AC to avoid need for trusted referee

Battleship Rules
Ship Placement

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

Battleship Rules
Ship Placement

	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										

Carrier

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										
H										
I										
J										

Battleship

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					s	s	s			
H										
I										
J										

Submarine

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					s	s	s			
H							d			
I							d			
J							d			

Destroyer

Battleship Rules
Ship Placement

Patrol Boat

	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										

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 CG –

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 CG – "Miss"

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 CB –

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 CB – "Hit"

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 DB –

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 DB – "Miss"

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 CC –

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 CC – "Hit"

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										

Skipping Ahead …

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

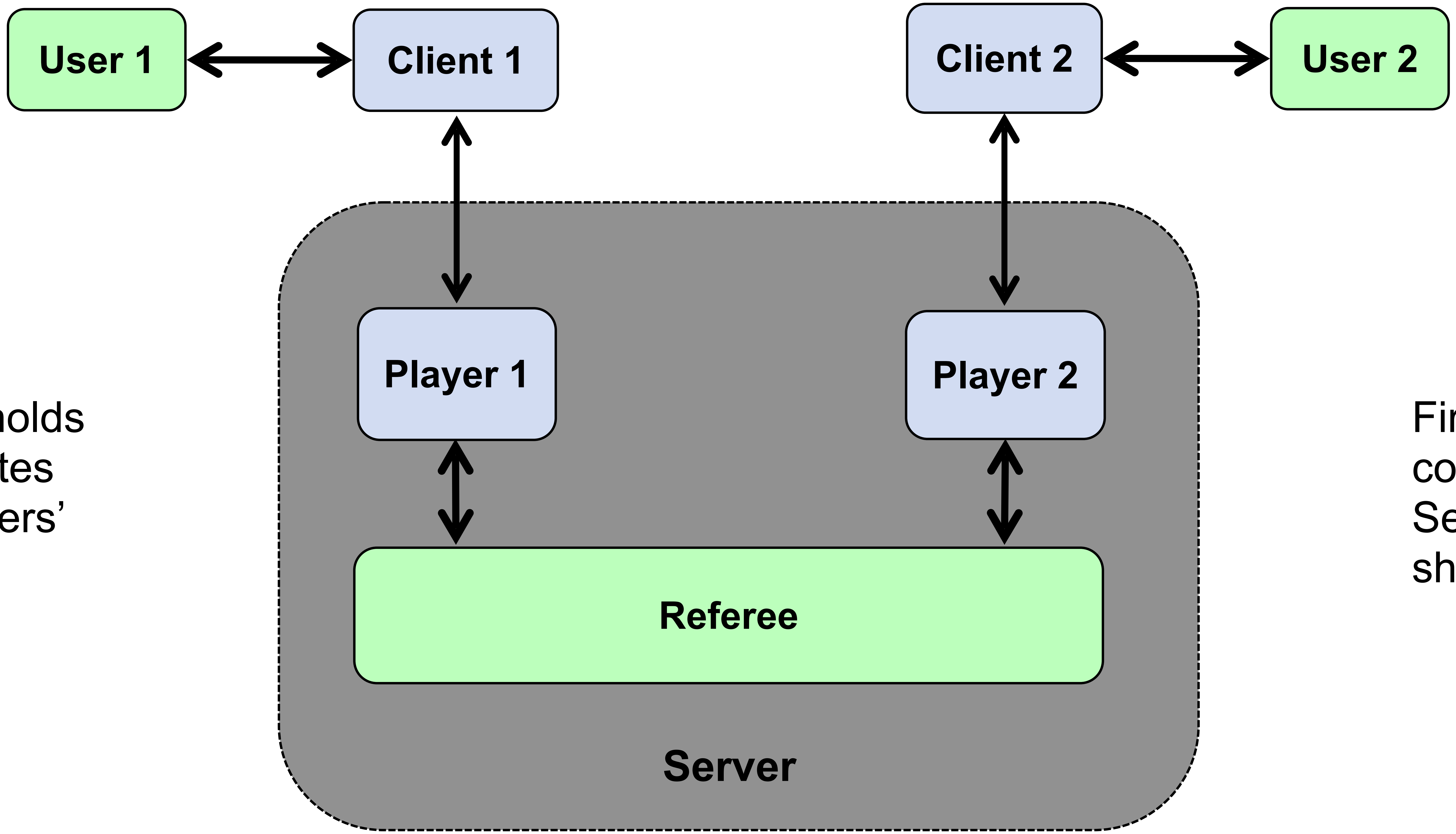
	A	B	C	D	E	F	G	H	I	J
A										
B						b				
C	C	C	C	C	C	b	★			
D		★		★		b				
E						b	★			
F										
G		★	p		S	S	S			
H			p				D			
I				★			D			
J				★	★	★	D			

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 and Behavior

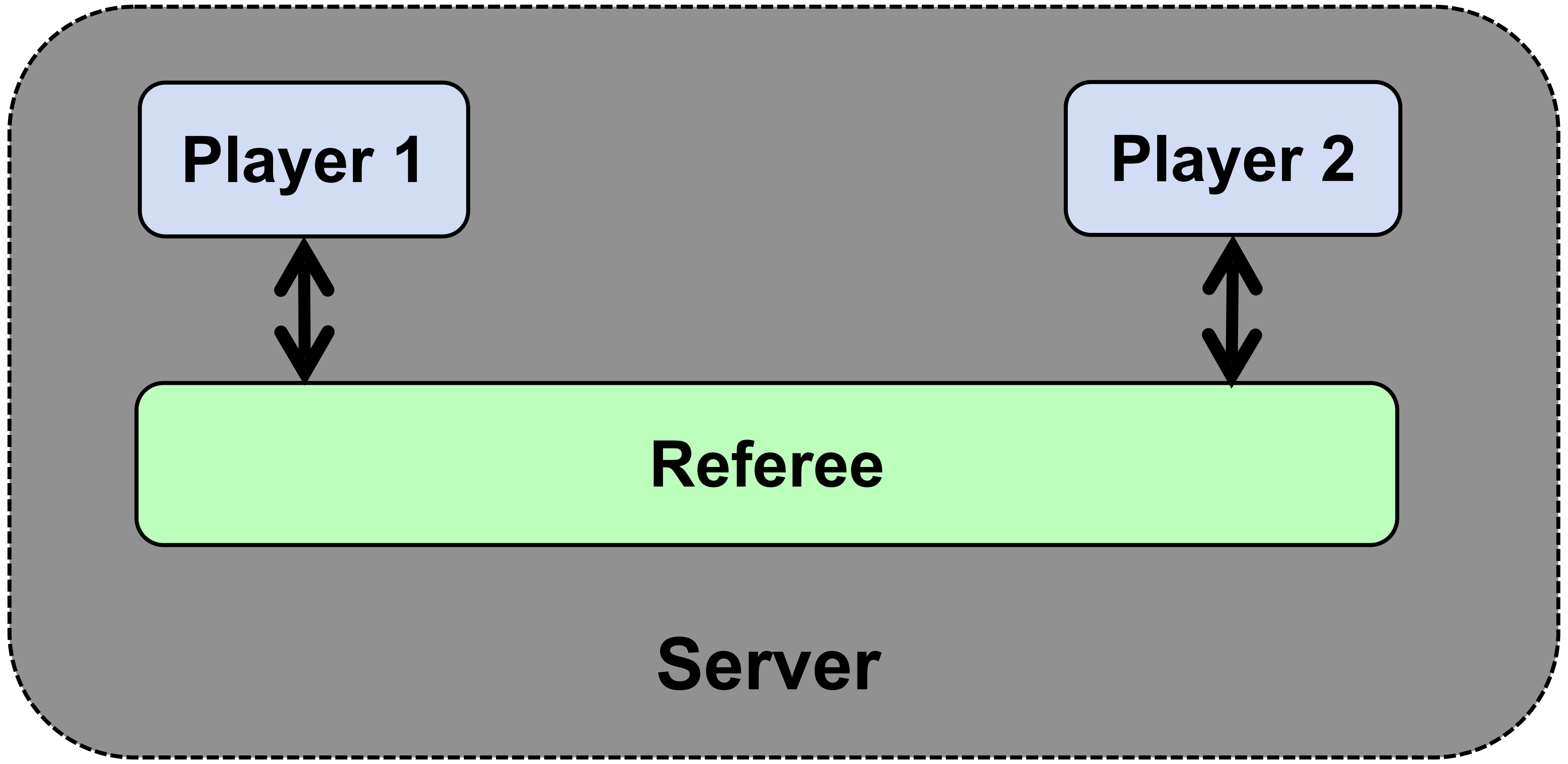
Referee holds and updates both Players' boards



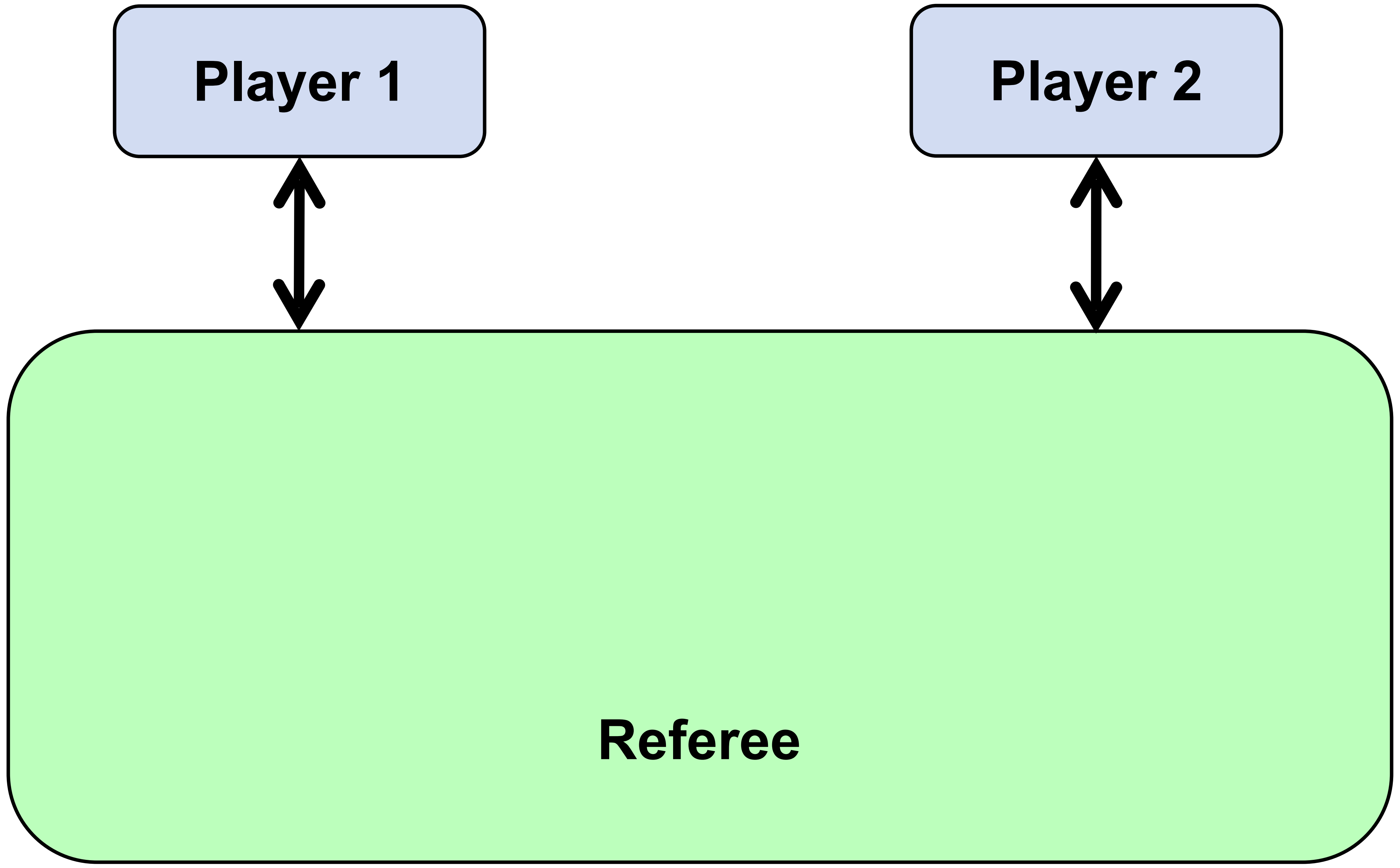
First Client to connect to Server gets to shoot first

Whole Program Security

- A referee is *secure* iff it is indistinguishable from a model referee, from the players' viewpoints
- Players are untrusted
- First CML implementation directly implements the model referee

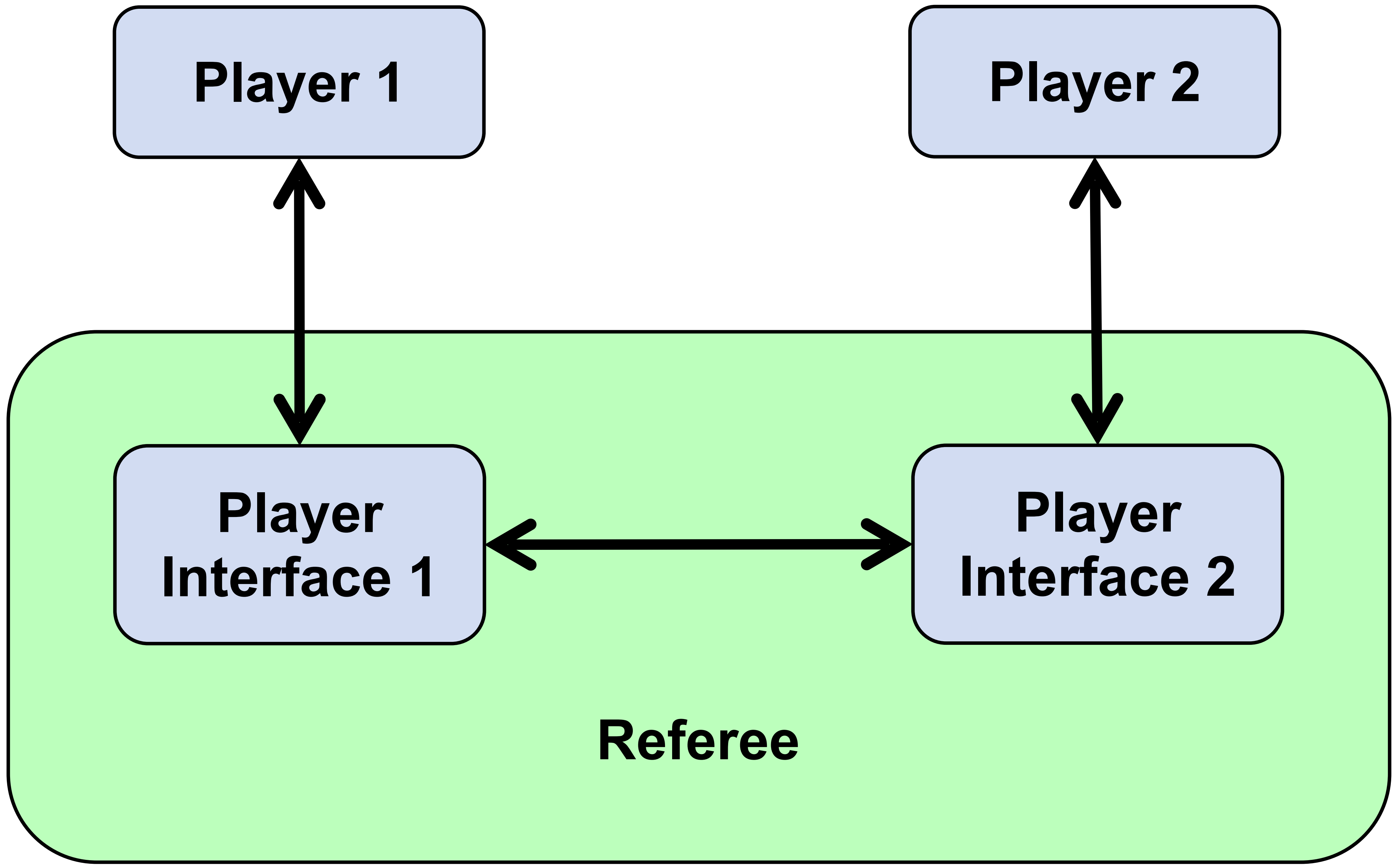


Splitting Referee into Mutually Distrustful Player Interfaces (PIs)



Splitting Referee into Mutually Distrustful Player Interfaces (PIs)

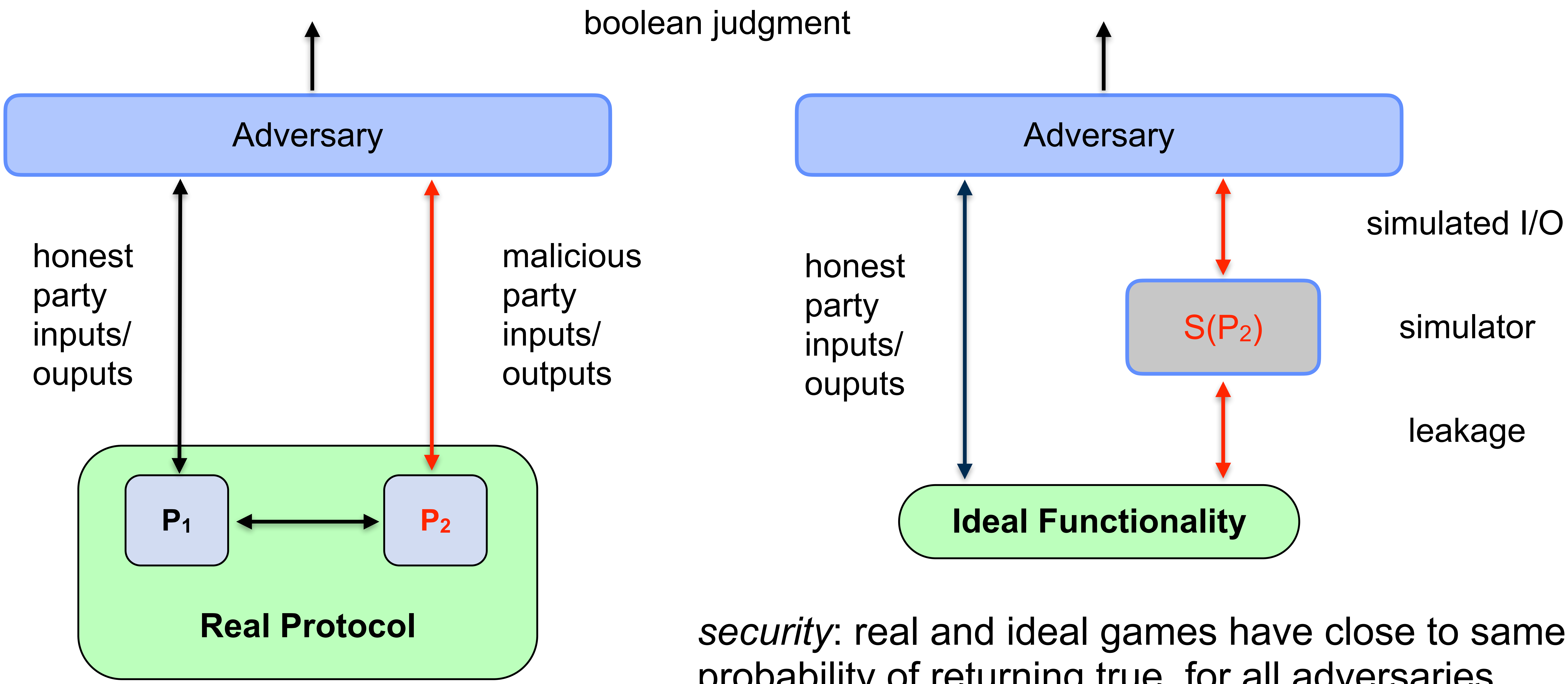
PIs will rely on some *trusted infrastructure*



How do we define security against a malicious PI?

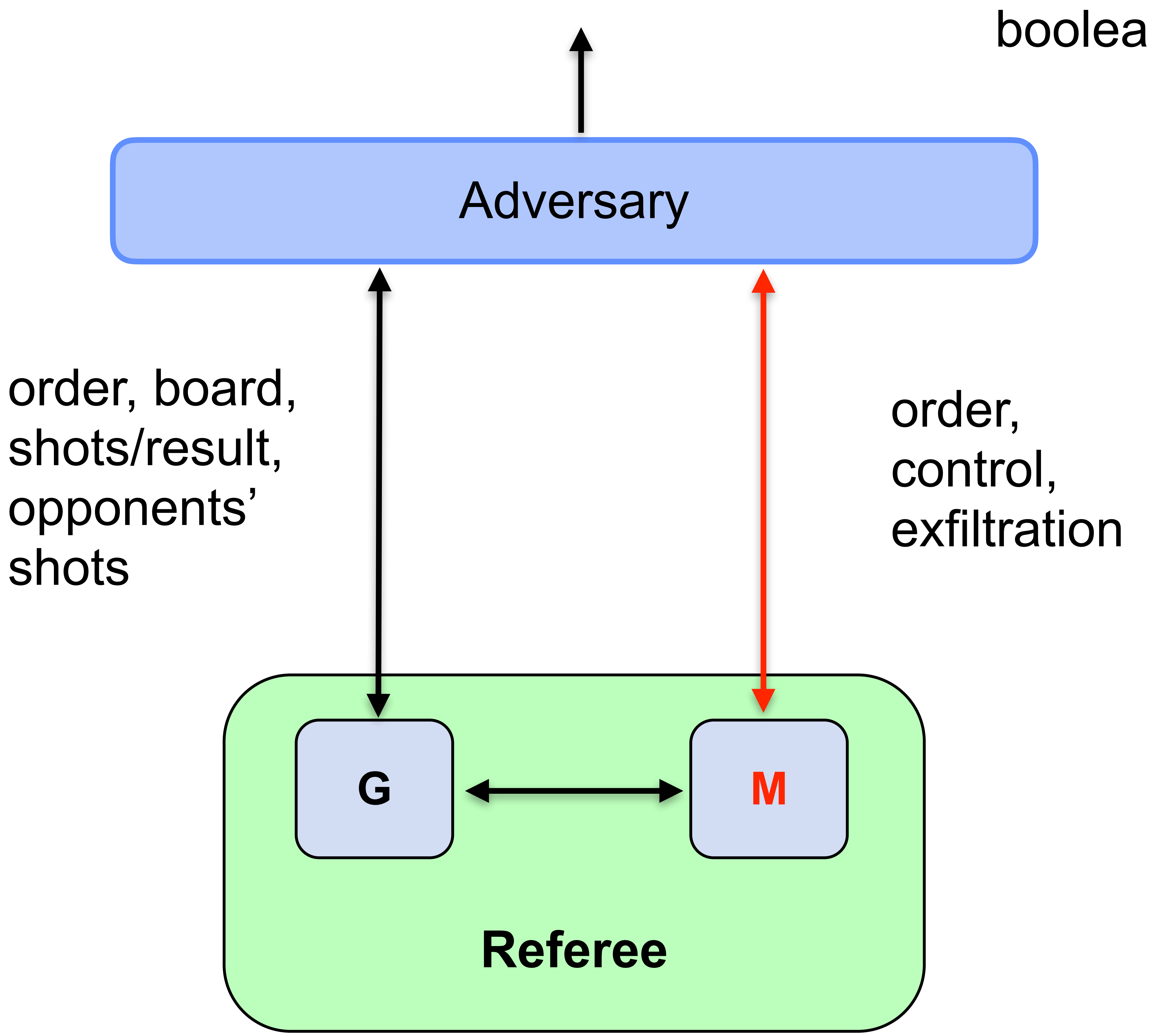
Our normal definition of security applies to a split referee, but we want also security against a malicious opponent PI

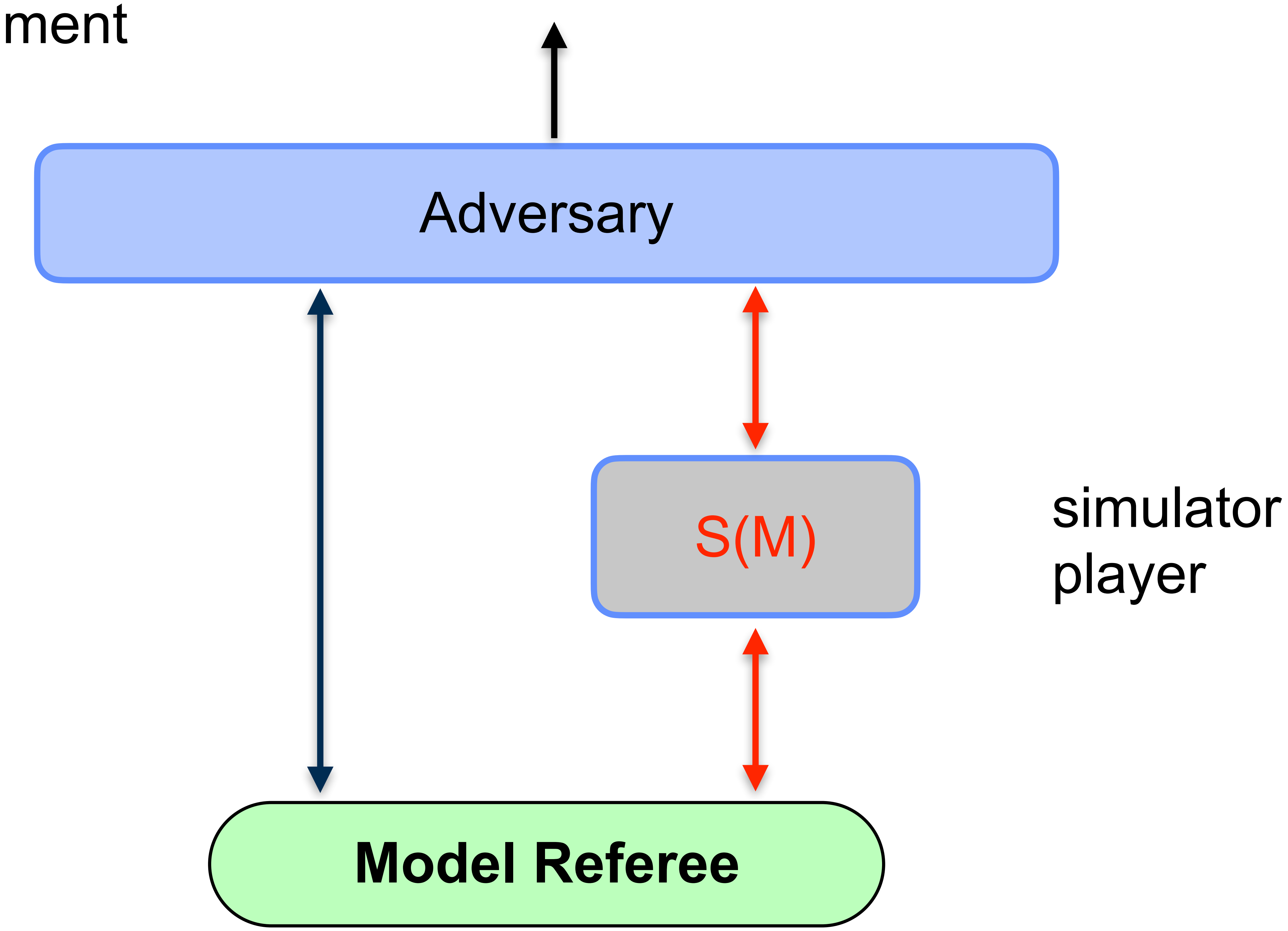
Theoretical Cryptography's Real/Ideal Paradigm



security: real and ideal games have close to same probability of returning true, for all adversaries

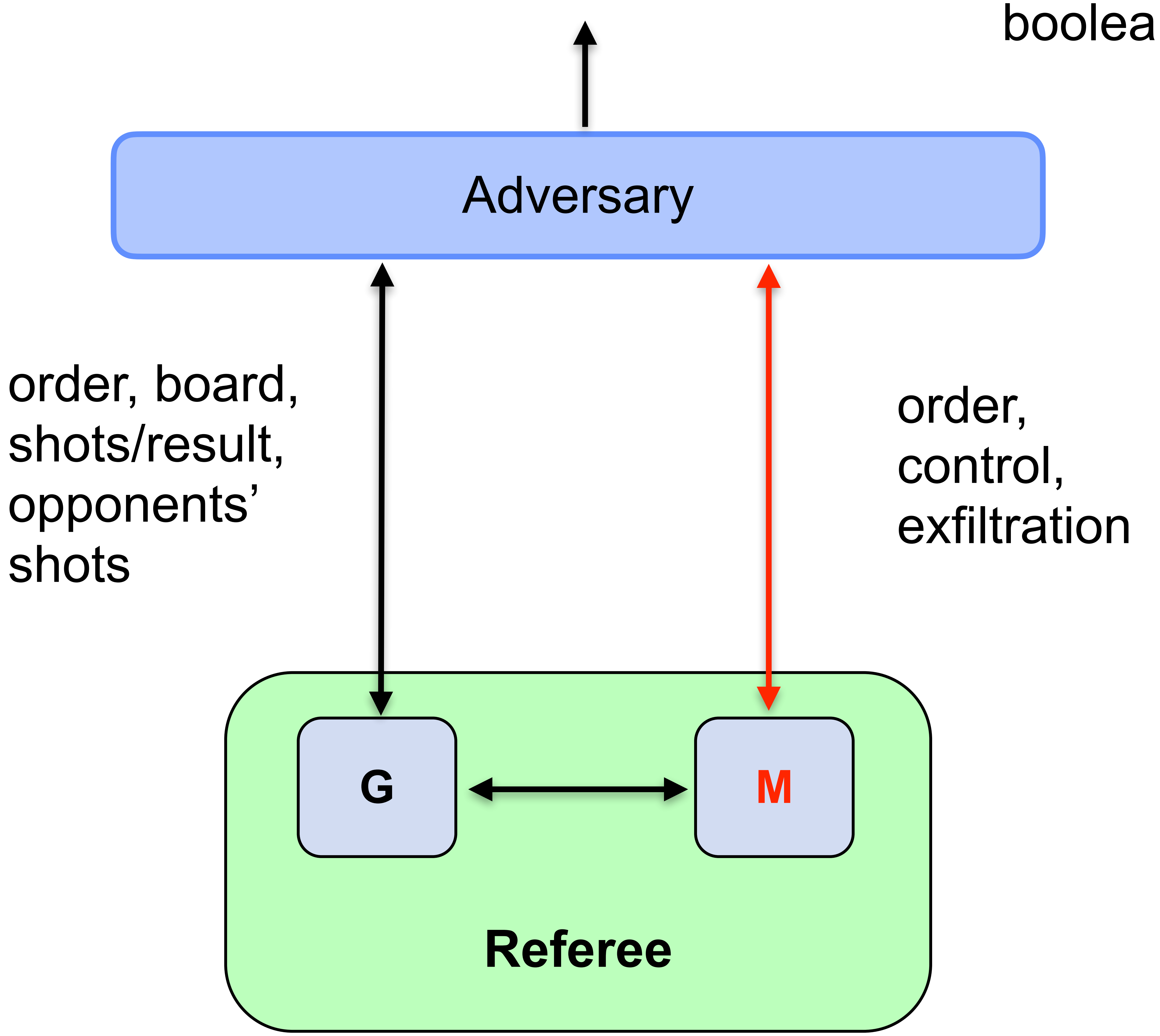
Security Against Malicious PI (Tentative)

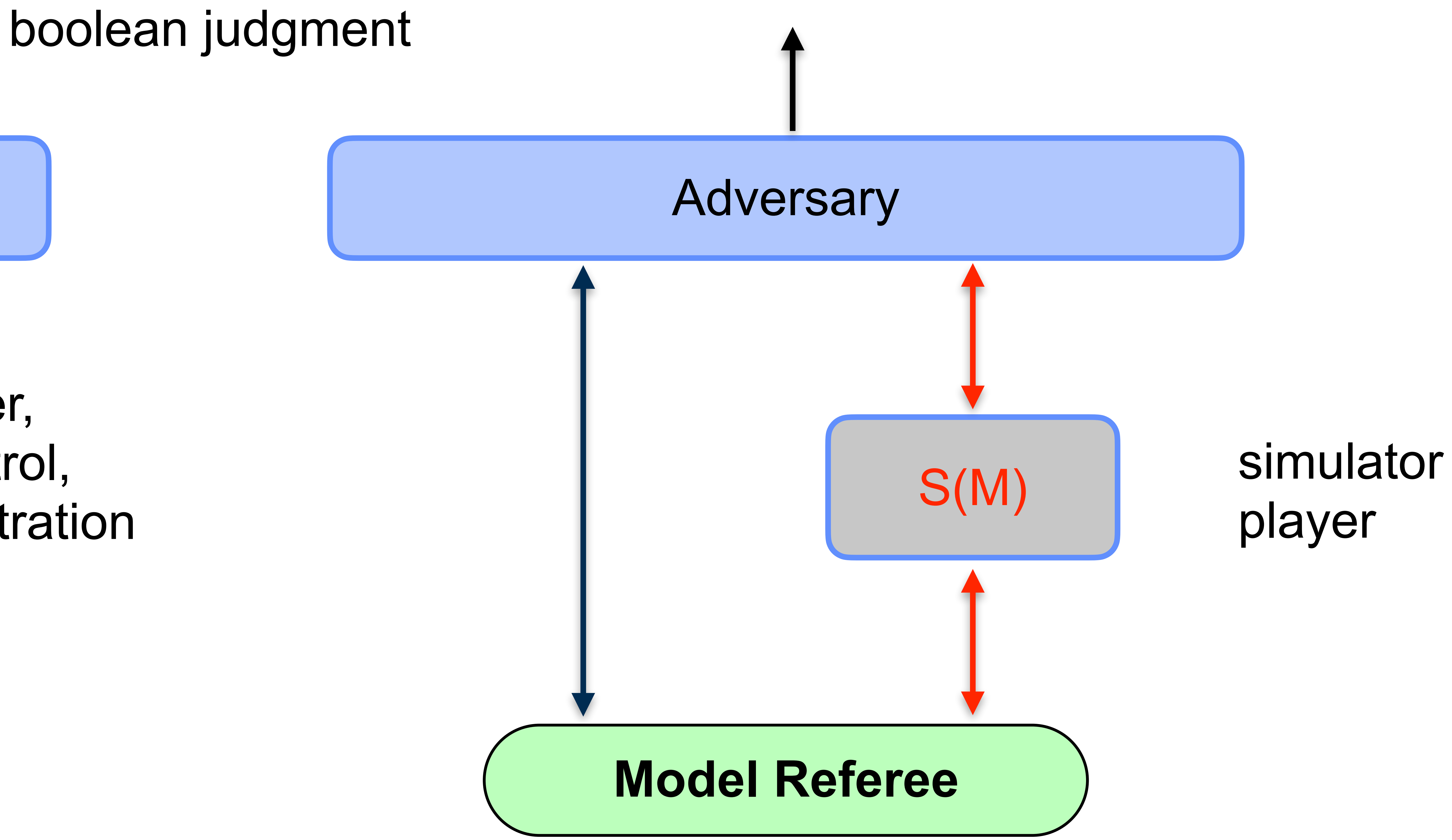




security: as scheduling is *possibilistic*, not *probabilistic*, what do we want for security?

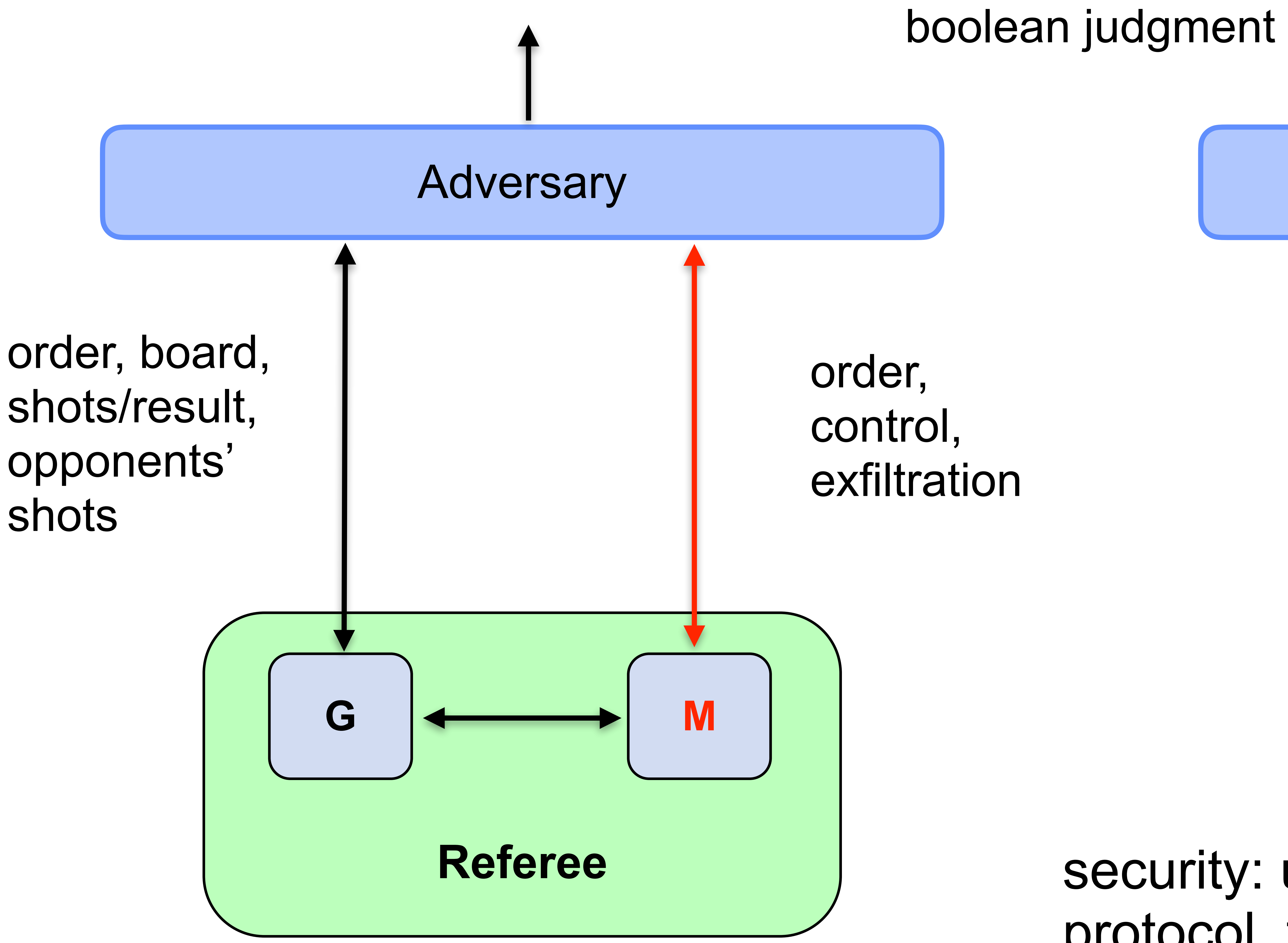
Security Against Malicious PI (Tentative)

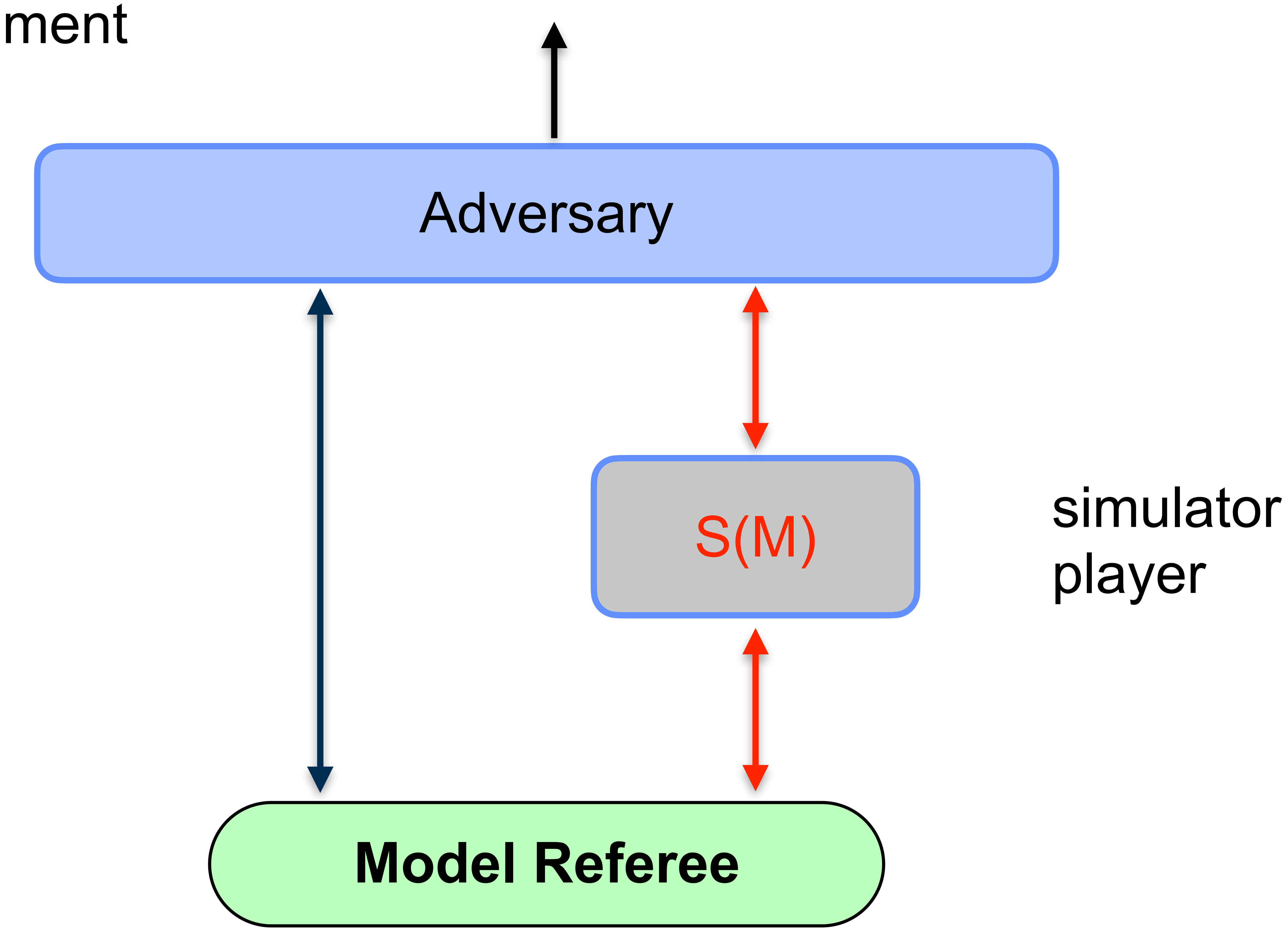




security: if there is an execution on one side resulting in *b*, then there is an execution on the other side also resulting in *b*

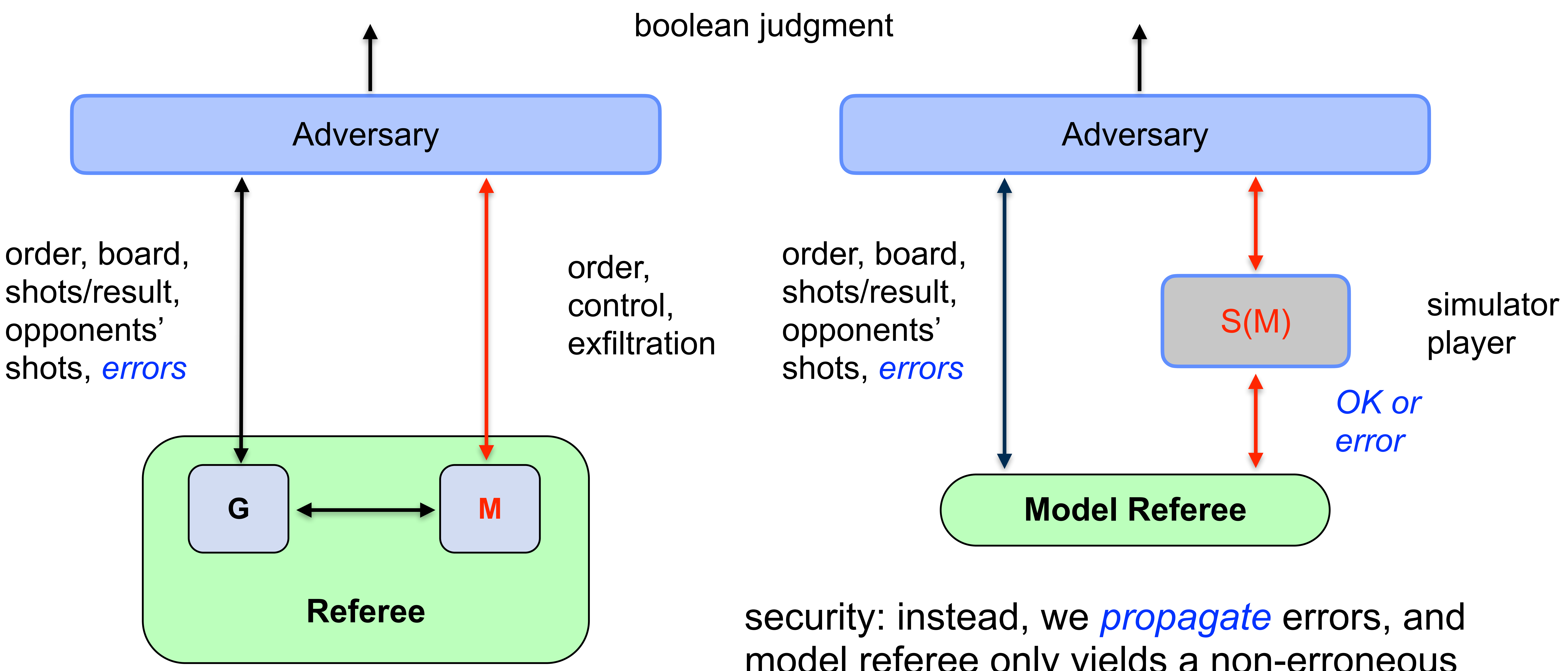
Security Against Malicious PI (Tentative)





security: unfortunately, if M doesn't follow the protocol, the error behavior (*termination*) in the two worlds can be different

Security Against Malicious PI



security: instead, we *propagate* errors, and model referee only yields a non-erroneous result if simulator player says OK

Ambiguity Example: Patrol Boat

	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			

GD
HC
GC

Ambiguity Example: Patrol Boat

	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	p						
H			s				d			
I			s				d			
J			s				d			

GD
HC
GC

LIO

- LIO is a library for Concurrent Haskell with dynamic encorcement of information flow control
- Information flow labels have both secrecy and integrety components
- Provides mutable variables, which can be shared between threads, and used for communication

LIO Battleship

- PIs exchange — using trusted code — labeled boards, made of labeled cells:

```
data LSR =  -- labeled shot result
       Miss       -- a miss
     | Hit        -- hit an unspecified ship
     | Sank Ship  -- sank a specified ship

data LC =  -- labeled cell
  LC
  (DCLabeled
   (Principal,  -- originating player interface
    Principal,  -- receiving player interface
    Pos,        -- position of cell
    DC LSR      -- DC action for shooting cell
   ))
```

LIO Example

PI 1

PI 2

Patrol Boat
MVar

LIO Example

PI 1

PI 2

1 : (1, 2, GC, pb) : 1 $\wedge$ 2

1 : (1, 2, HC, pb) : 1 $\wedge$ 2

Patrol Boat
MVar

LIO Example

PI 1

1 : (1, 2, HC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Patrol Boat MVar

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Patrol Boat MVar

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Patrol Boat MVar

HC

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

: (1, 2, HC, pb) : 1 ∧ 2

Patrol Boat MVar

HC

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Yields Hit

: (1, 2, HC, pb) : 1 ∧ 2

Patrol Boat MVar

HC

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

1 : (1, 2, GC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Yields Hit

: (1, 2, HC, pb) : 1 ∧ 2

Patrol Boat MVar

HC

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

: (1, 2, GC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Yields Hit

: (1, 2, HC, pb) : 1 ∧ 2

Patrol Boat MVar

HC

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

: (1, 2, GC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Yields Hit

: (1, 2, HC, pb) : 1 ∧ 2

Patrol Boat MVar

GC, HC

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

: (1, 2, GC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Yields Hit

: (1, 2, HC, pb) : 1 ∧ 2

: (1, 2, GC, pb) : 1 ∧ 2

Patrol Boat
MVar

GC, HC

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

: (1, 2, GC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Yields Hit

: (1, 2, HC, pb) : 1 ∧ 2

Yields Sank PatrolBoat

: (1, 2, GC, pb) : 1 ∧ 2

Patrol Boat MVar

GC, HC

LIO Example

PI 1

: (1, 2, HC, pb) : 1 ∧ 2

: (1, 2, GC, pb) : 1 ∧ 2

PI 2

1 : (1, 2, GC, pb) : 1 ∧ 2

1 : (1, 2, HC, pb) : 1 ∧ 2

Still Yields Hit

: (1, 2, HC, pb) : 1 ∧ 2

Yields Sank PatrolBoat

: (1, 2, GC, pb) : 1 ∧ 2

Patrol Boat MVar

GC, HC

Concurrent ML

- Concurrent ML is a library for Standard ML (we use the Standard ML of New Jersey implementation)
- It has no special security features
- But the combination of its abstract types (provided by its rich module system) and mutable references can be used to program access control

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

lb_1

CML + AC Example

PI 1

PI 2

Ib_1 HC

CML + AC Example

PI 1

HC

PI 2

Ib_1 HC

CML + AC Example

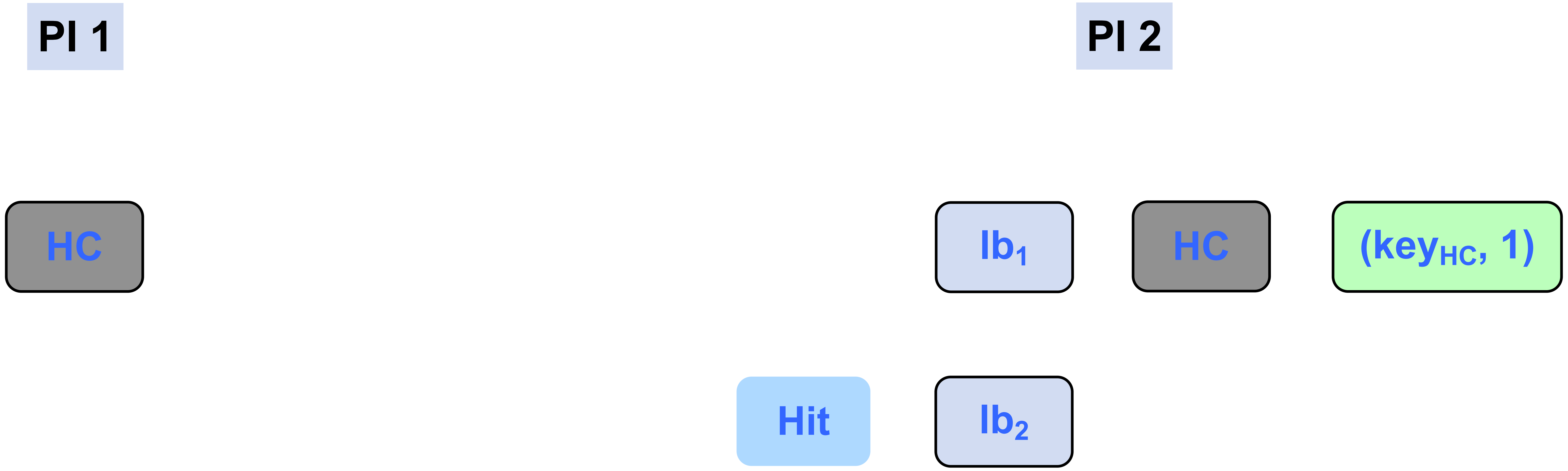
PI 1

HC

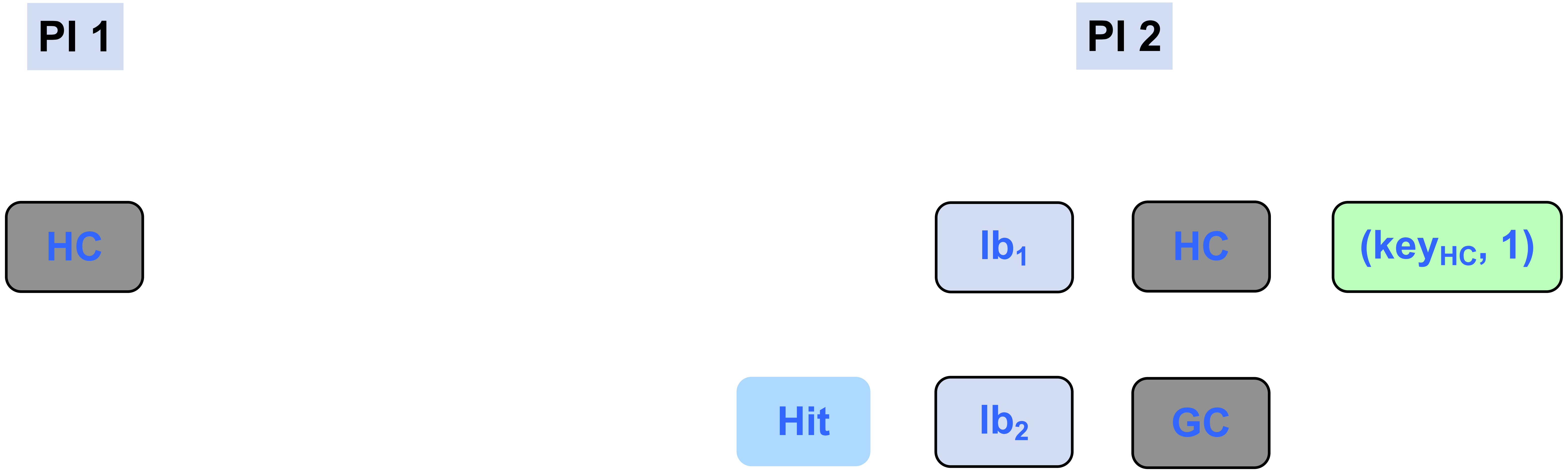
PI 2

lb_1 | HC | $(key_{HC}, 1)$

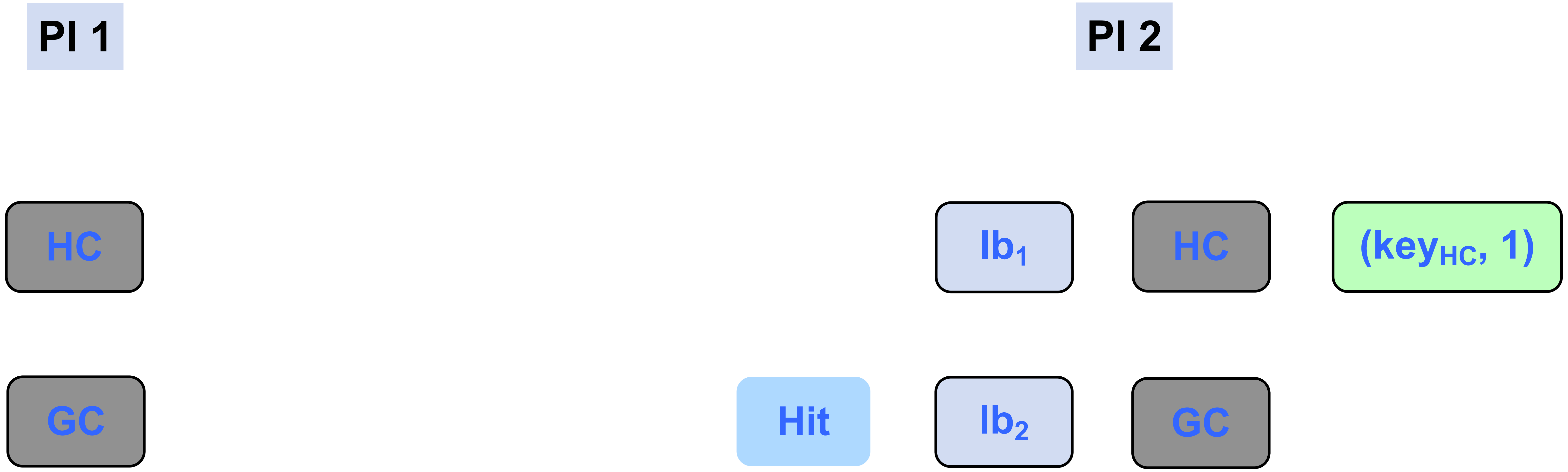
CML + AC Example



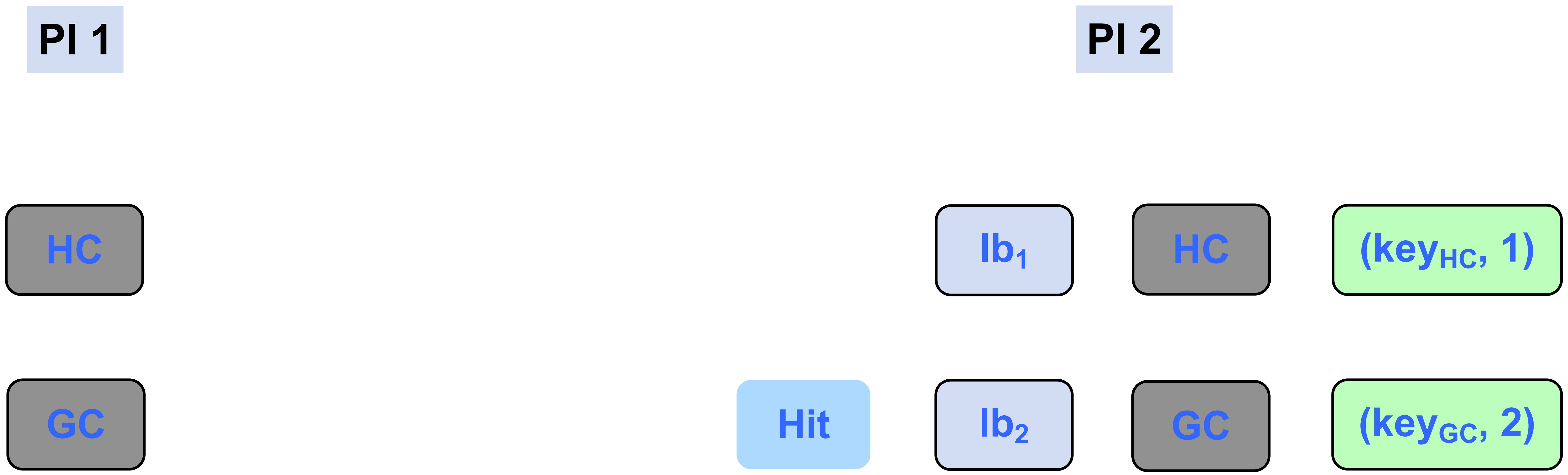
CML + AC Example



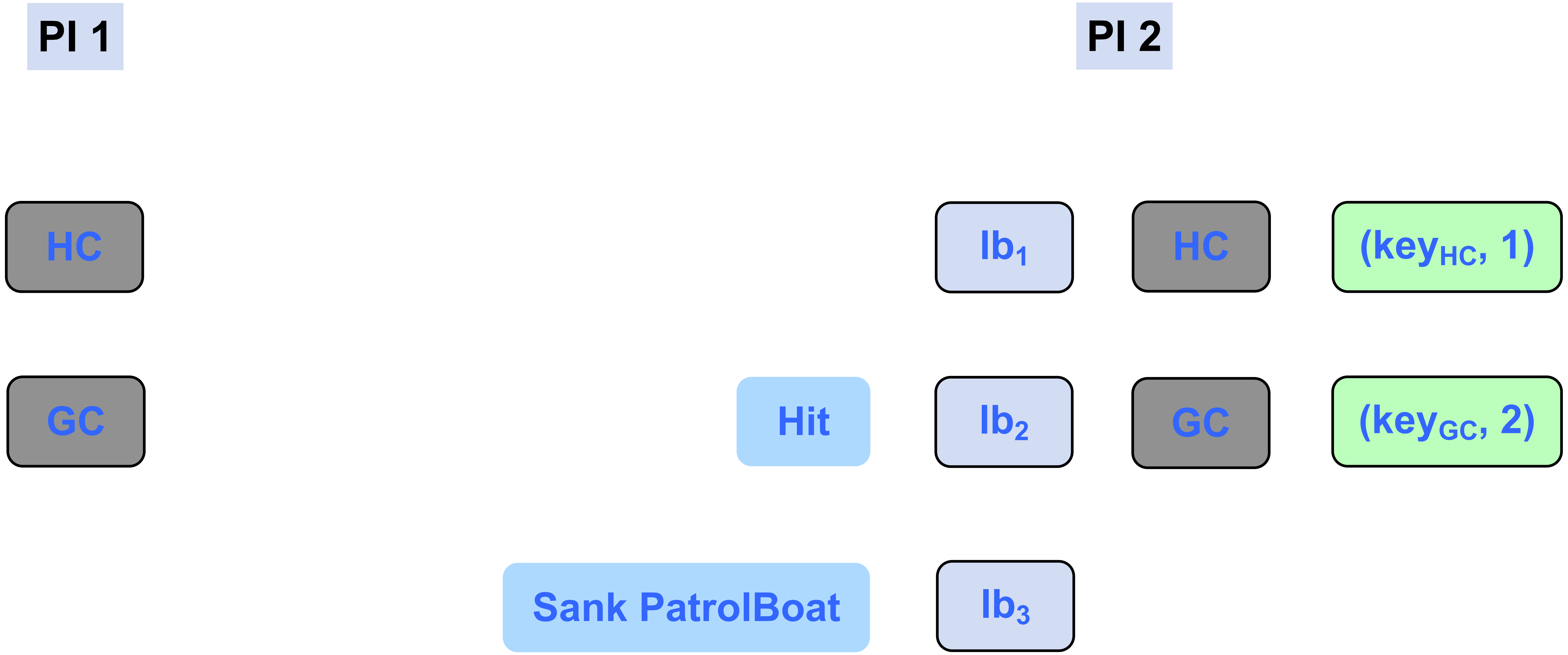
CML + AC Example



CML + AC Example

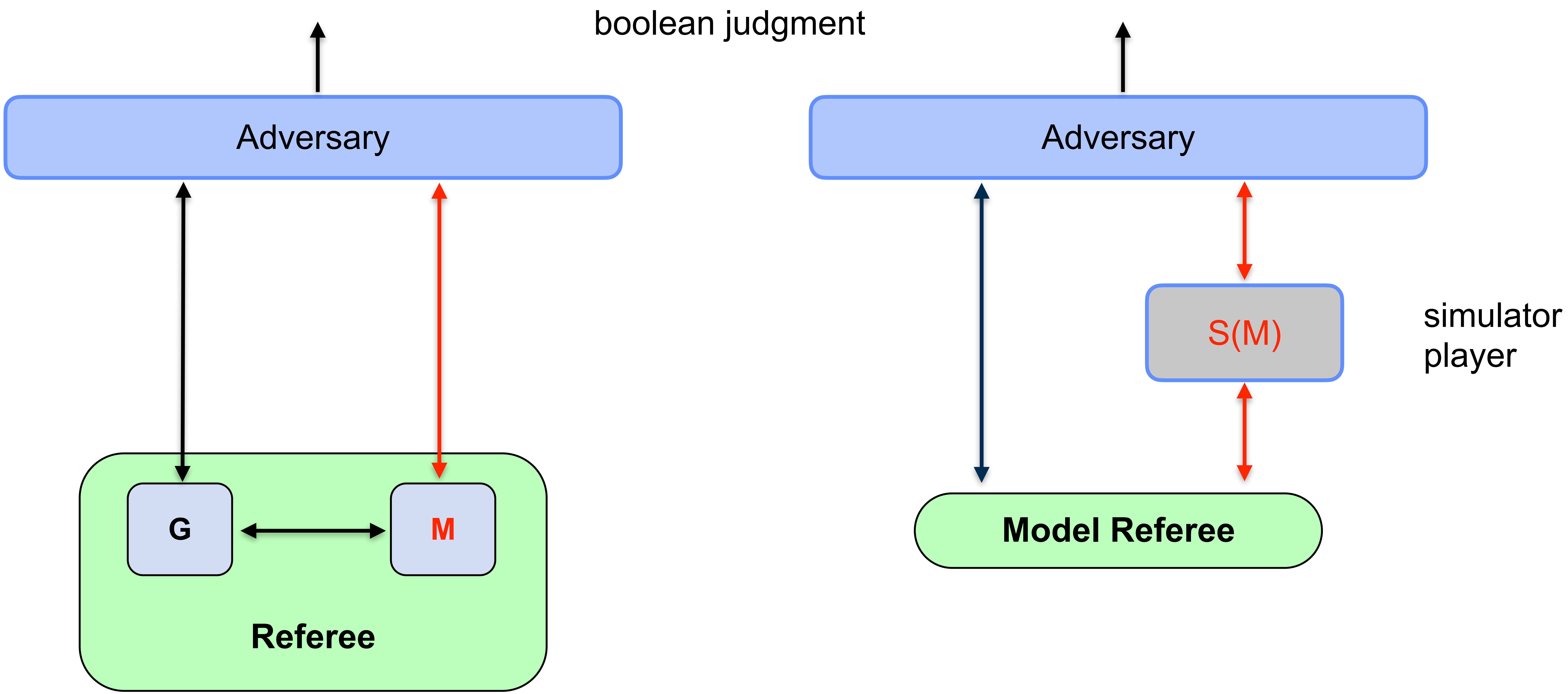


CML + AC Example

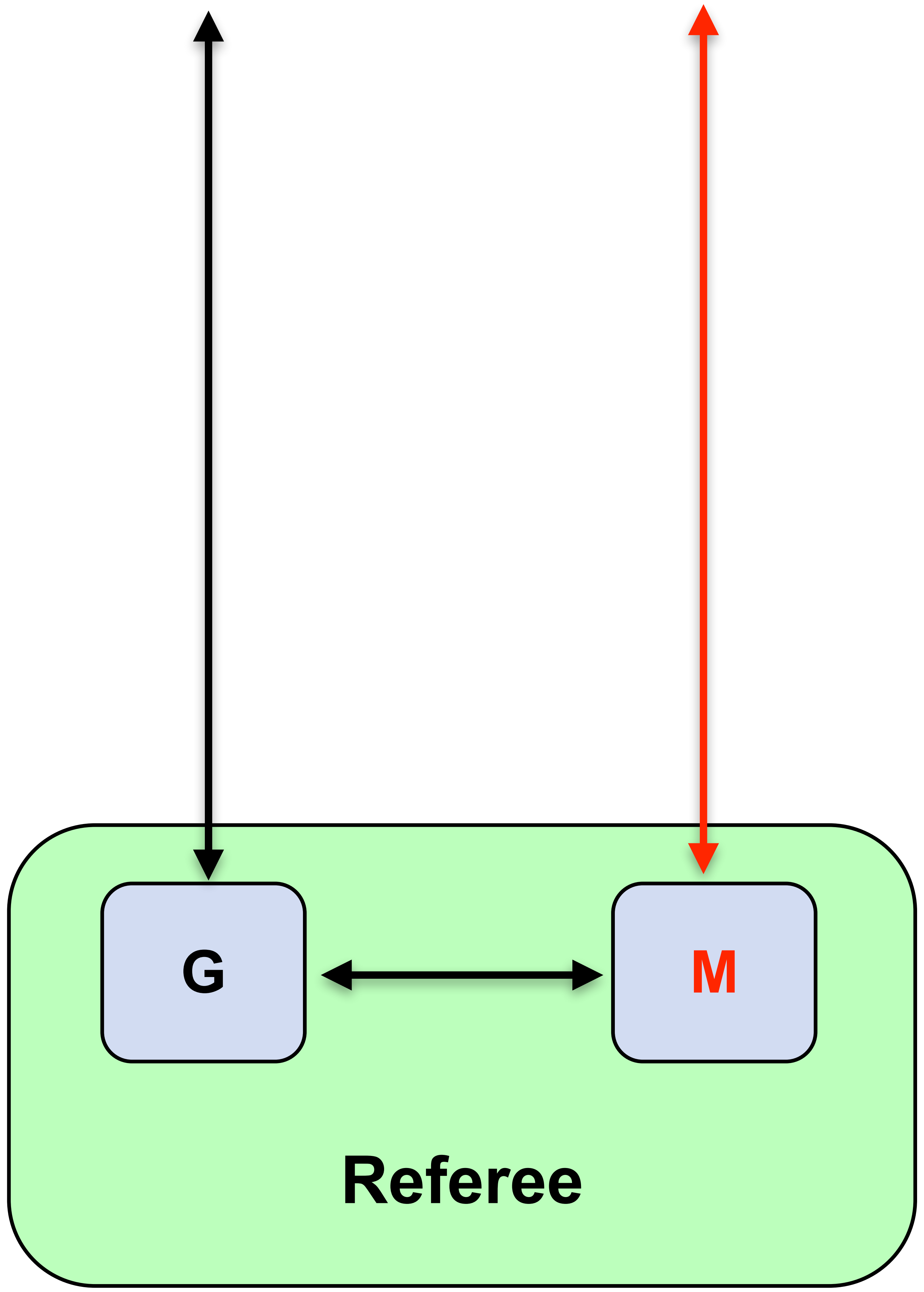


A counted key is only applicable to a single locked board, and can't be deconstructed

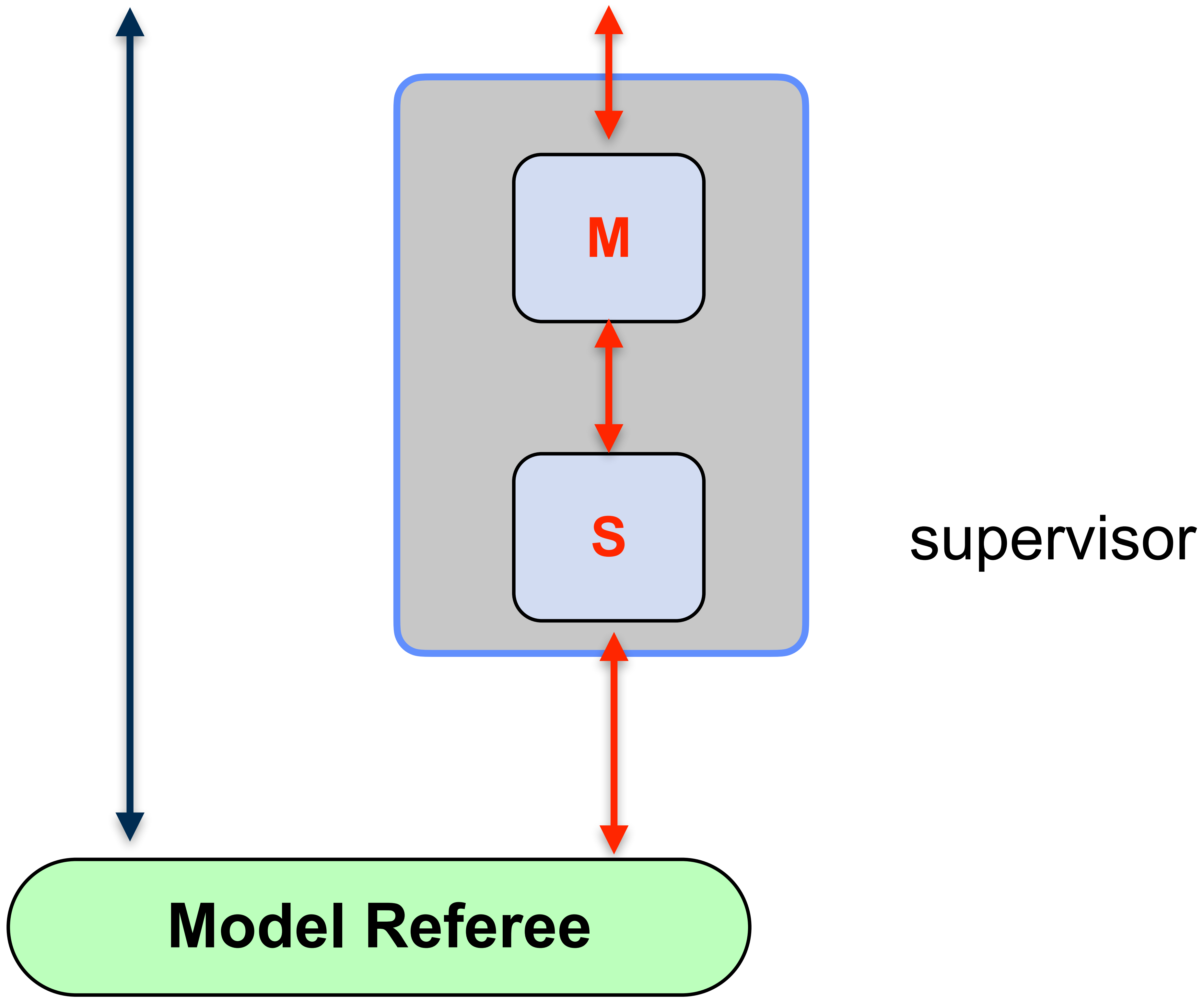
Construction of Simulator Player for CML + AC



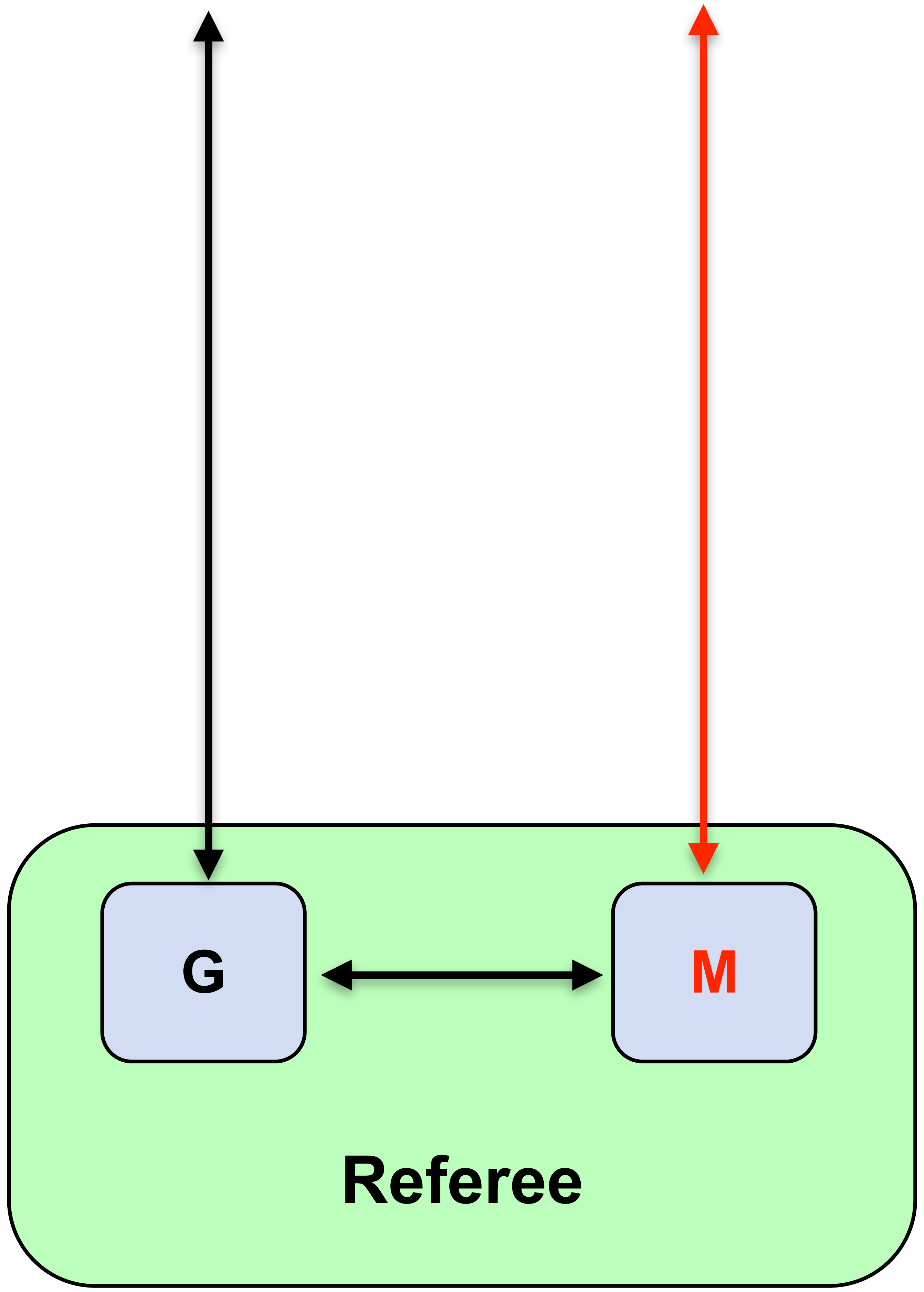
Construction of Simulator Player for CML + AC

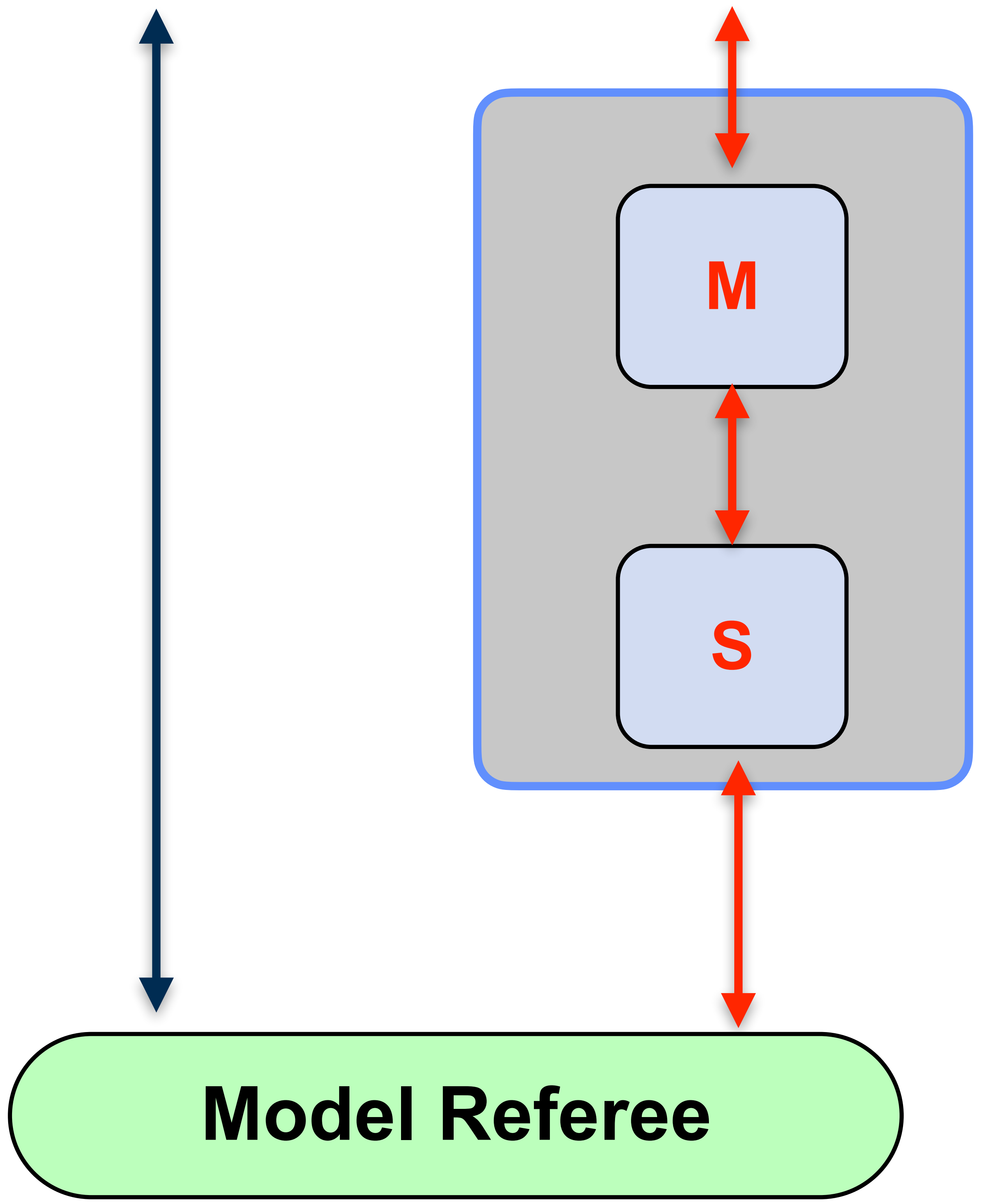


supervisor interacts with M using *reimplementation* of locked board abstract type

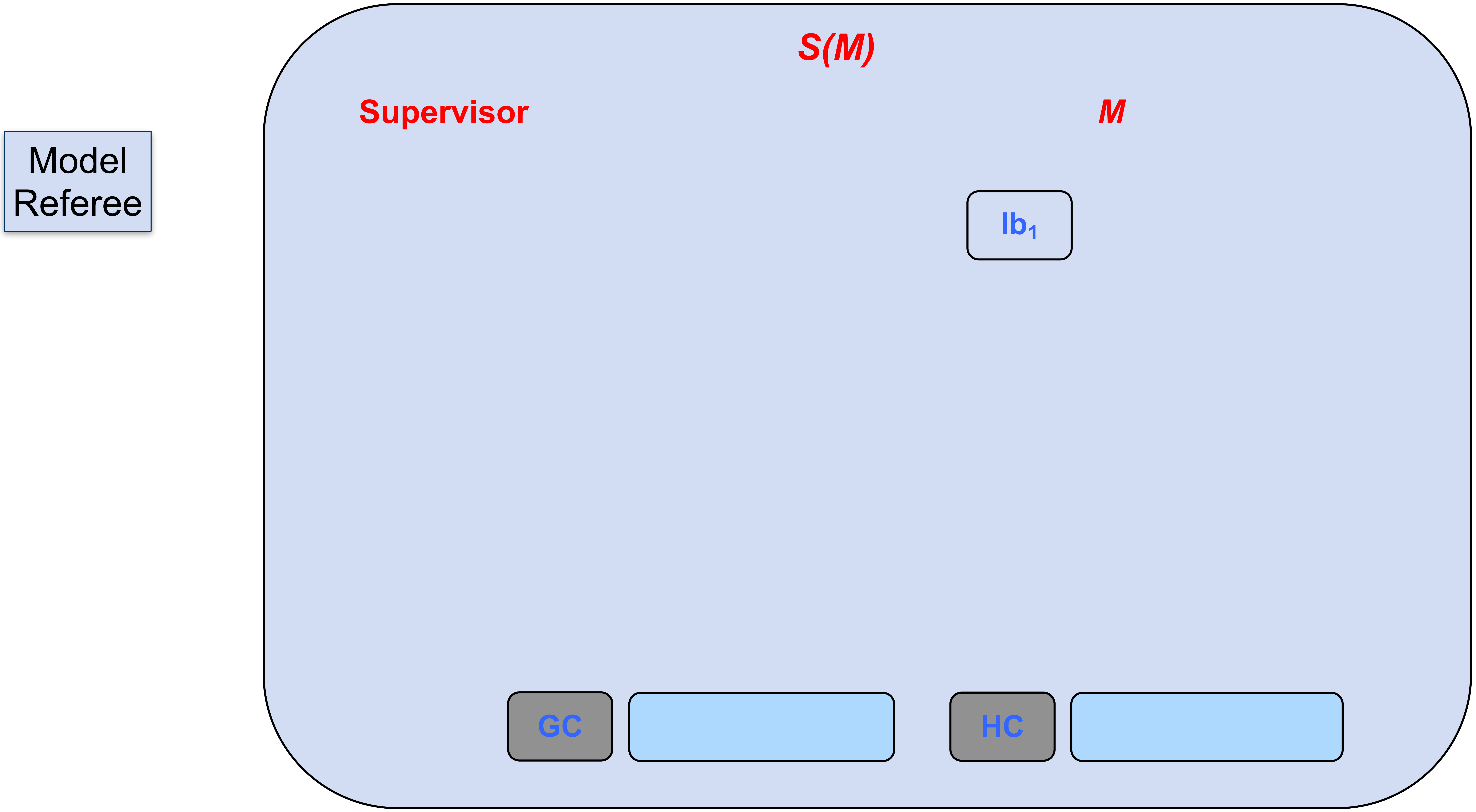


CML + AC: M Doesn’t Learn More Than it Should

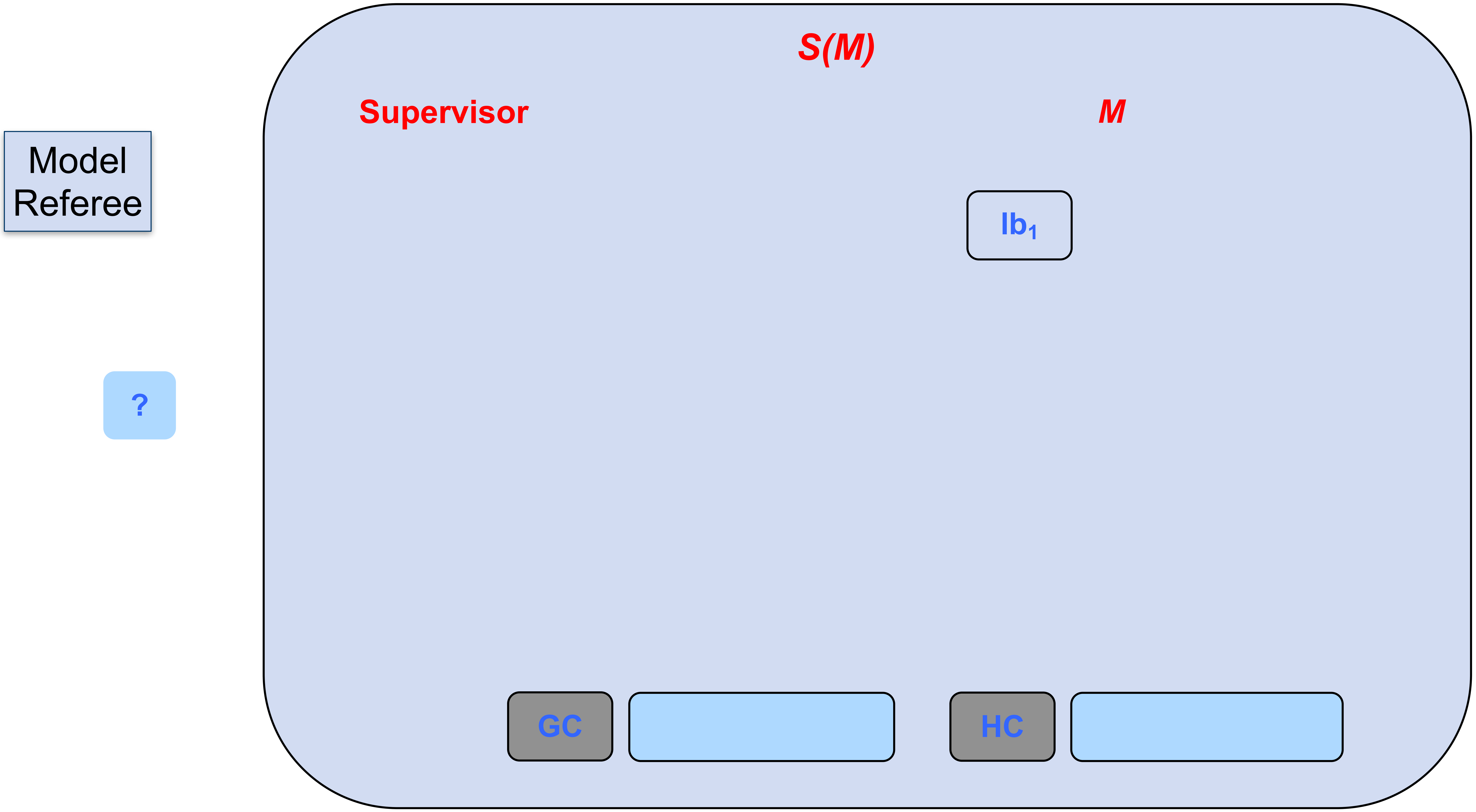




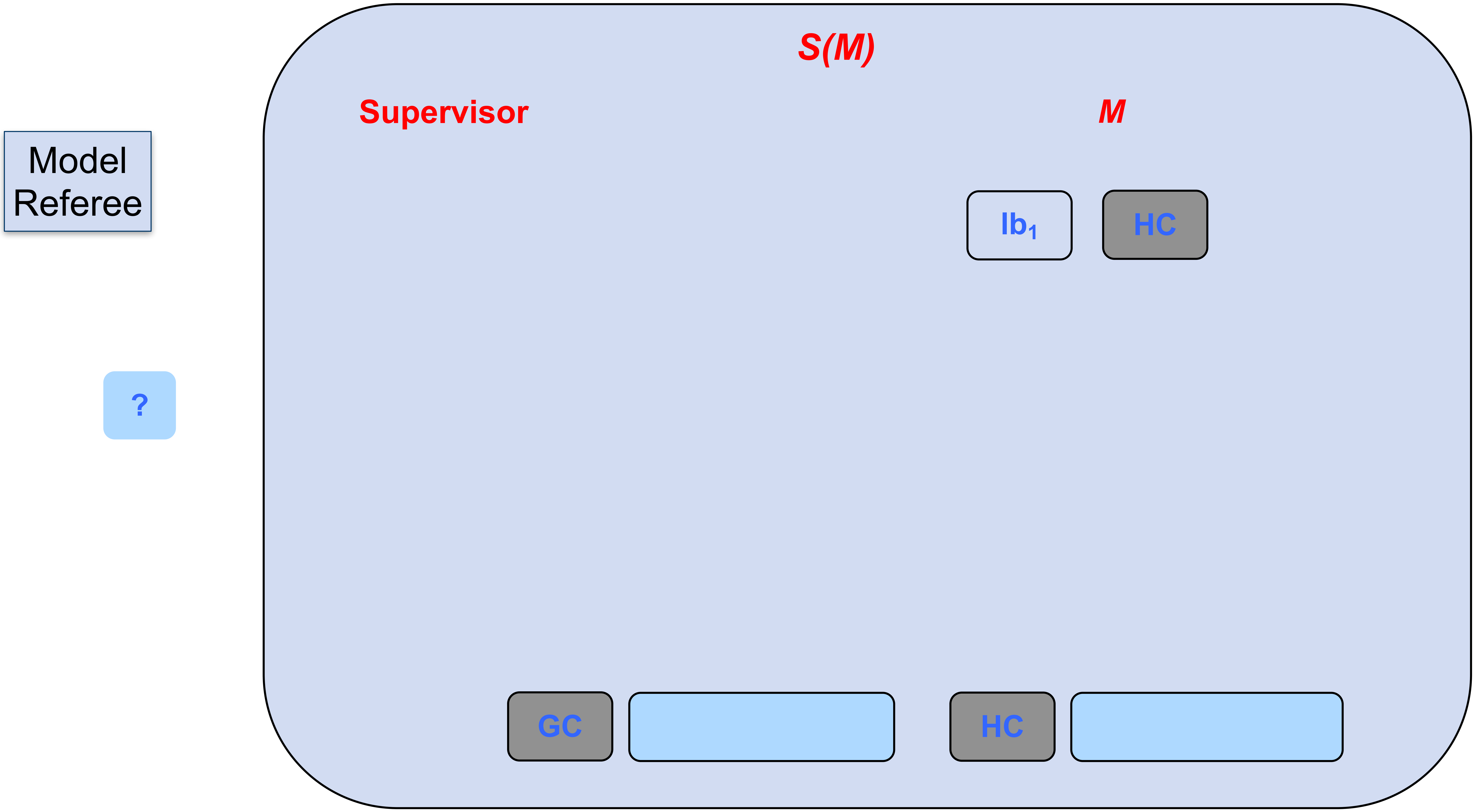
CML + AC Simulator Example



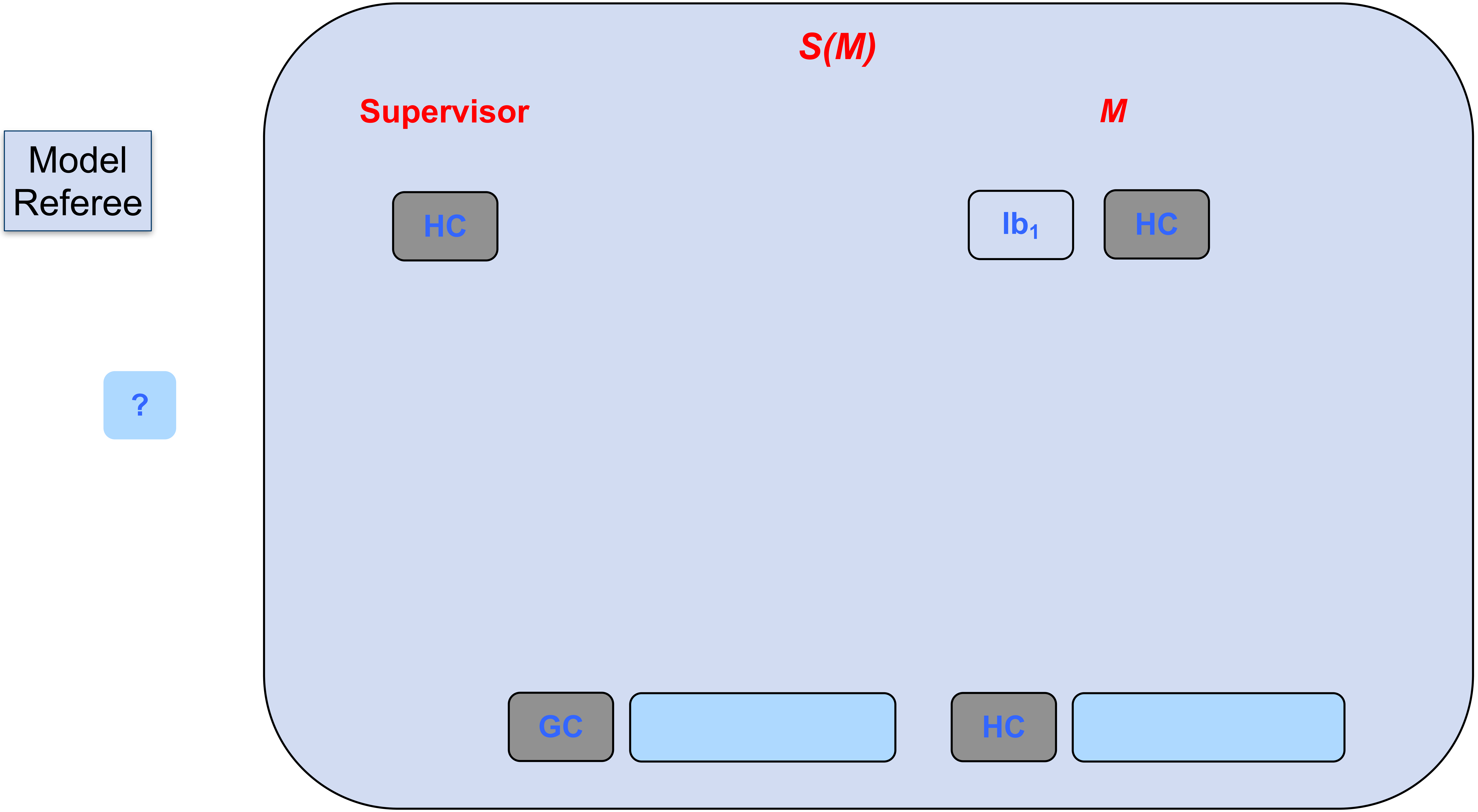
CML + AC Simulator Example



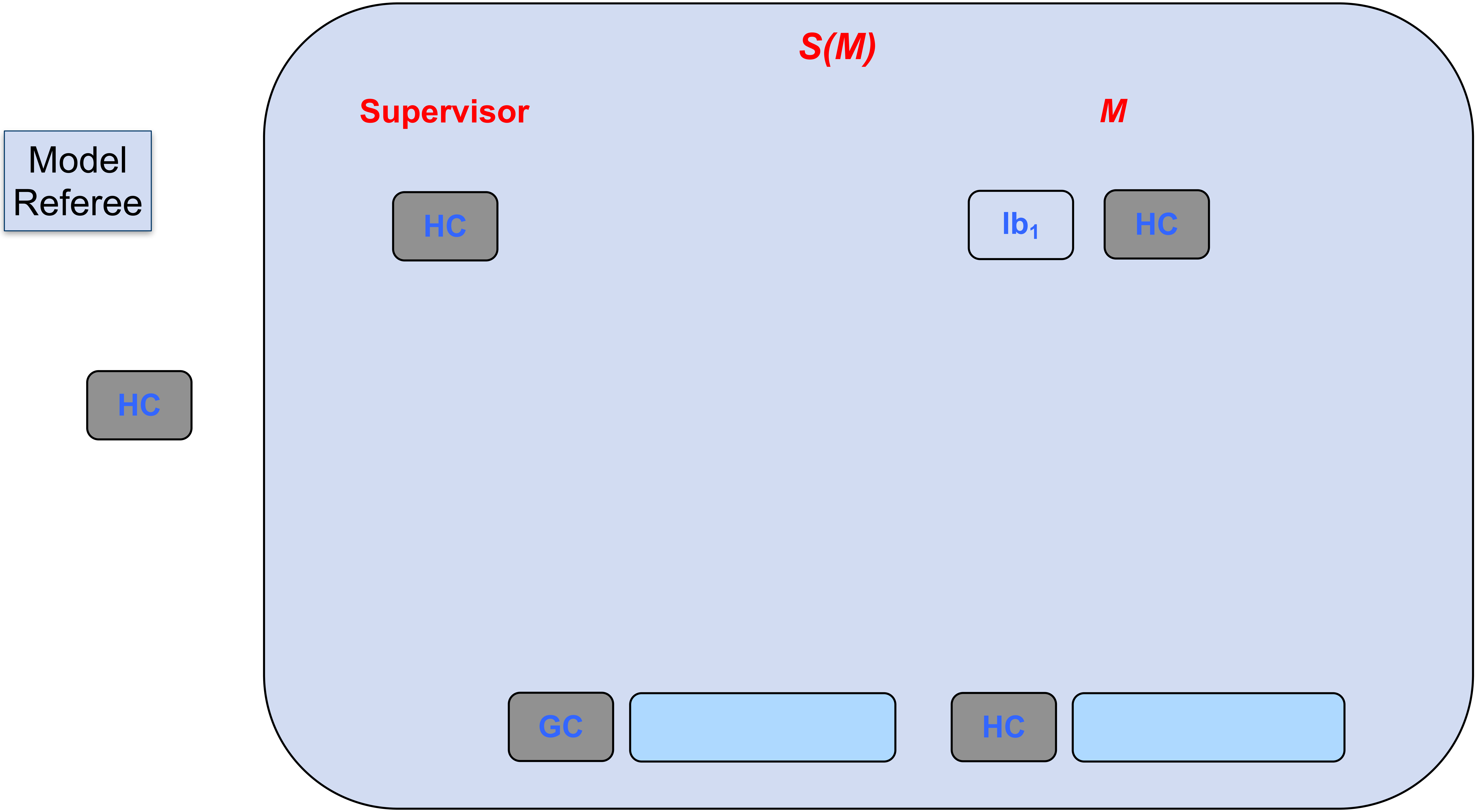
CML + AC Simulator Example



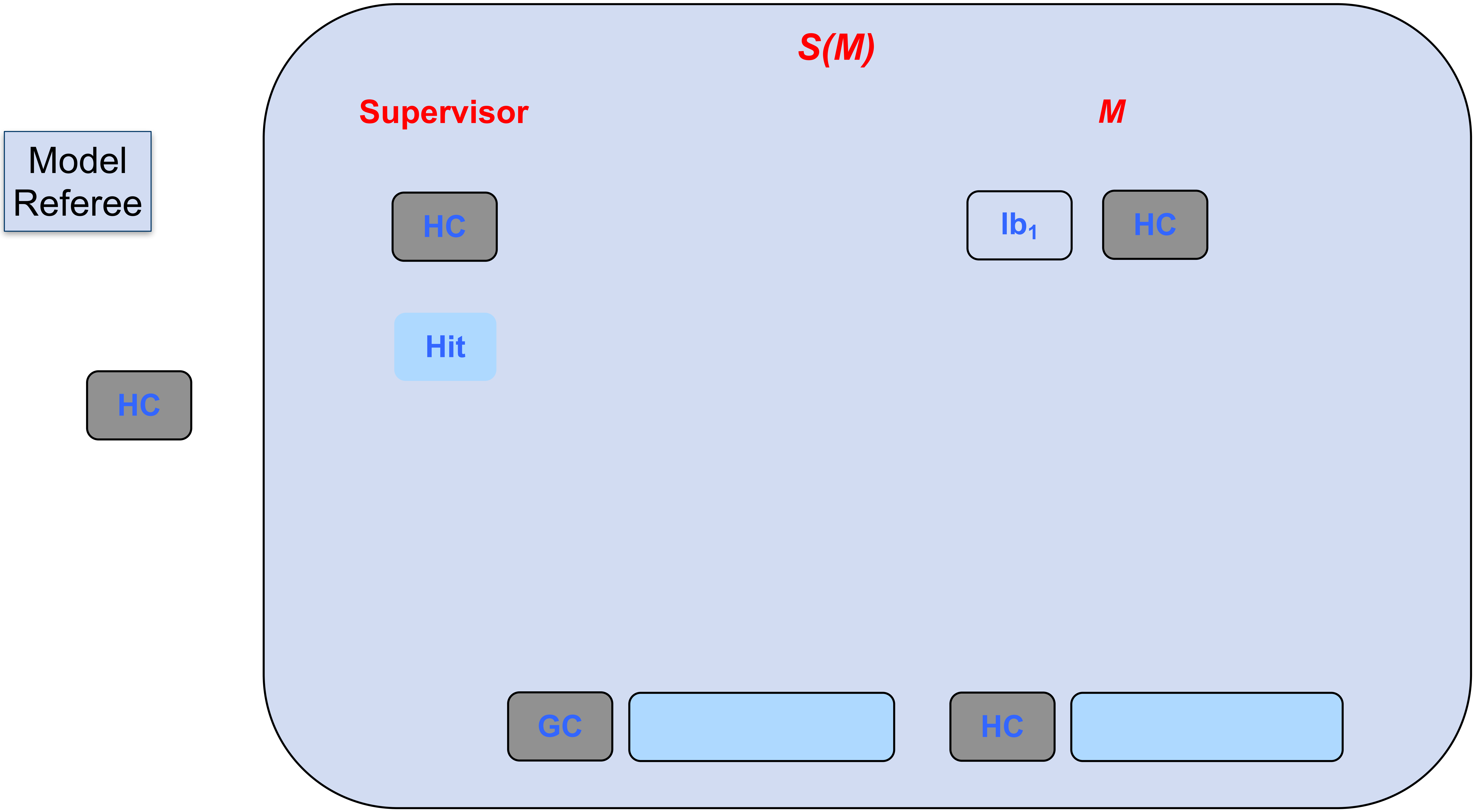
CML + AC Simulator Example



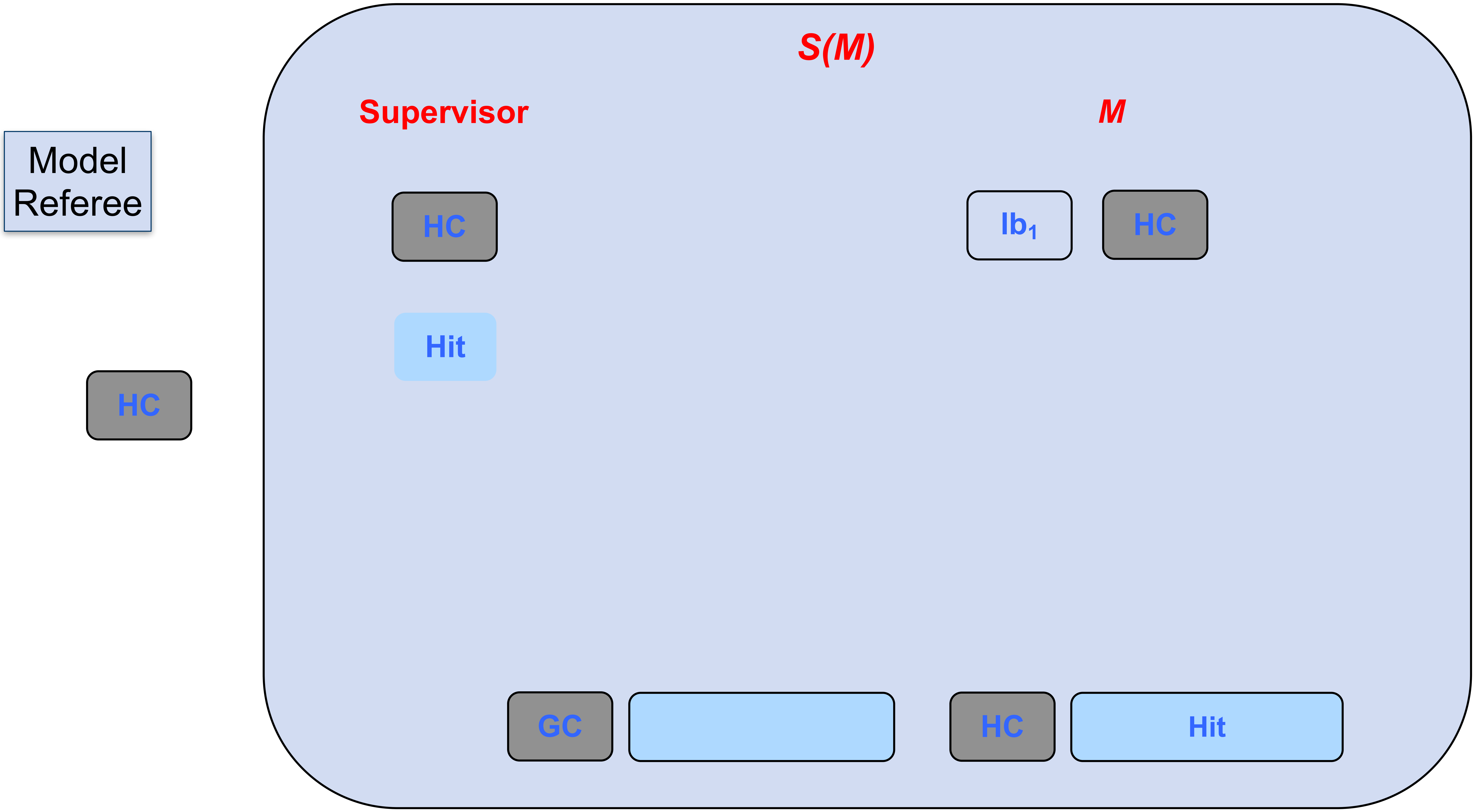
CML + AC Simulator Example



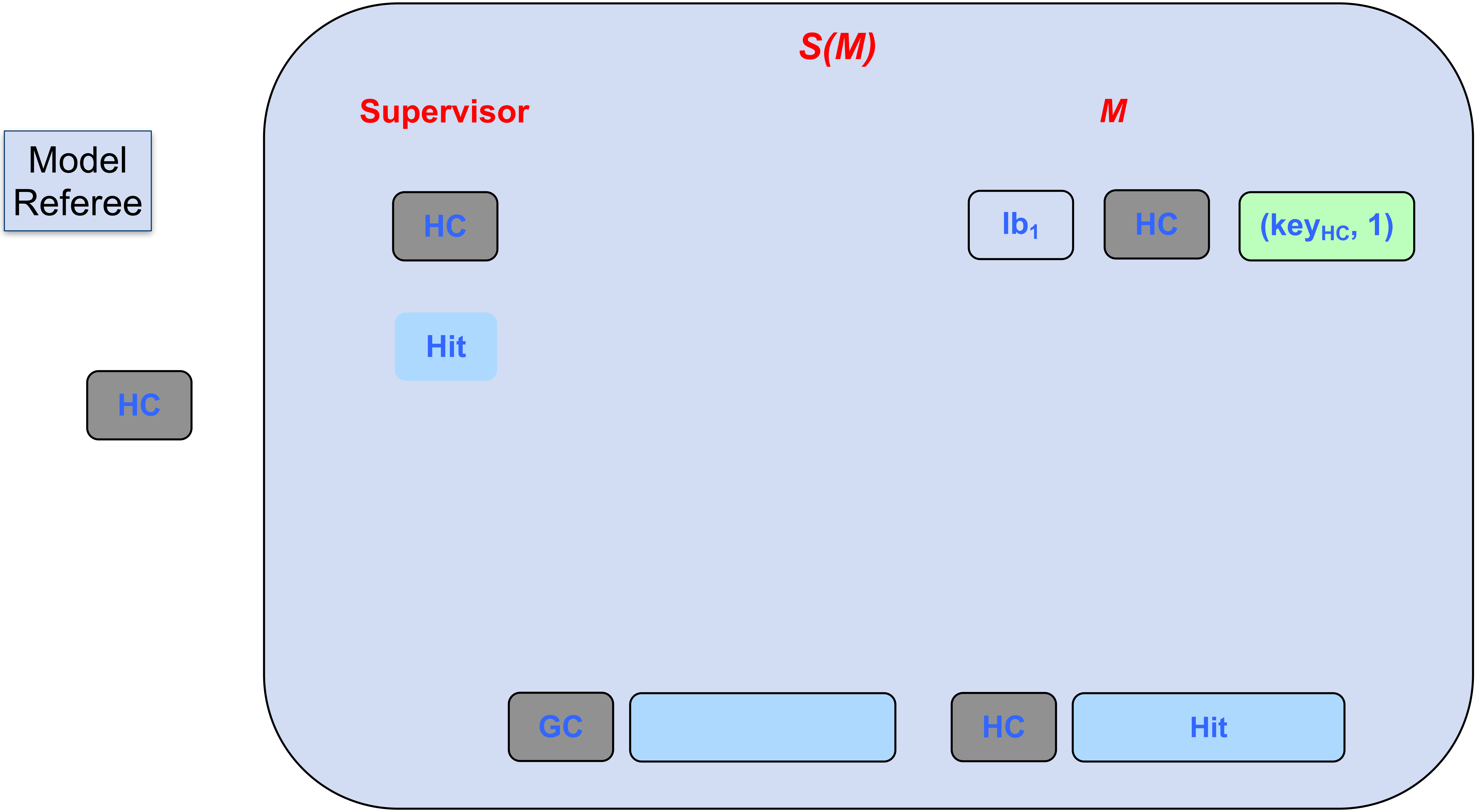
CML + AC Simulator Example



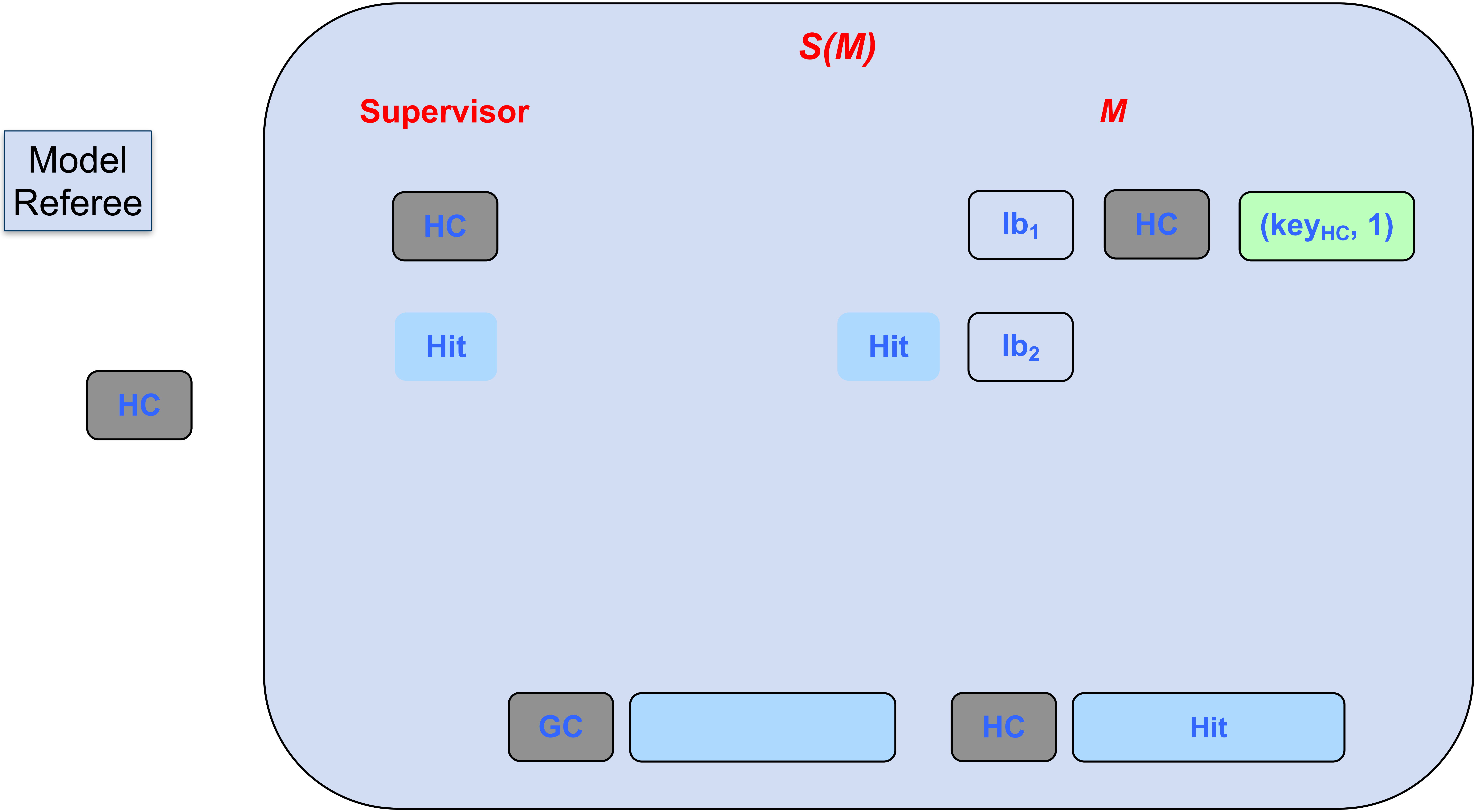
CML + AC Simulator Example



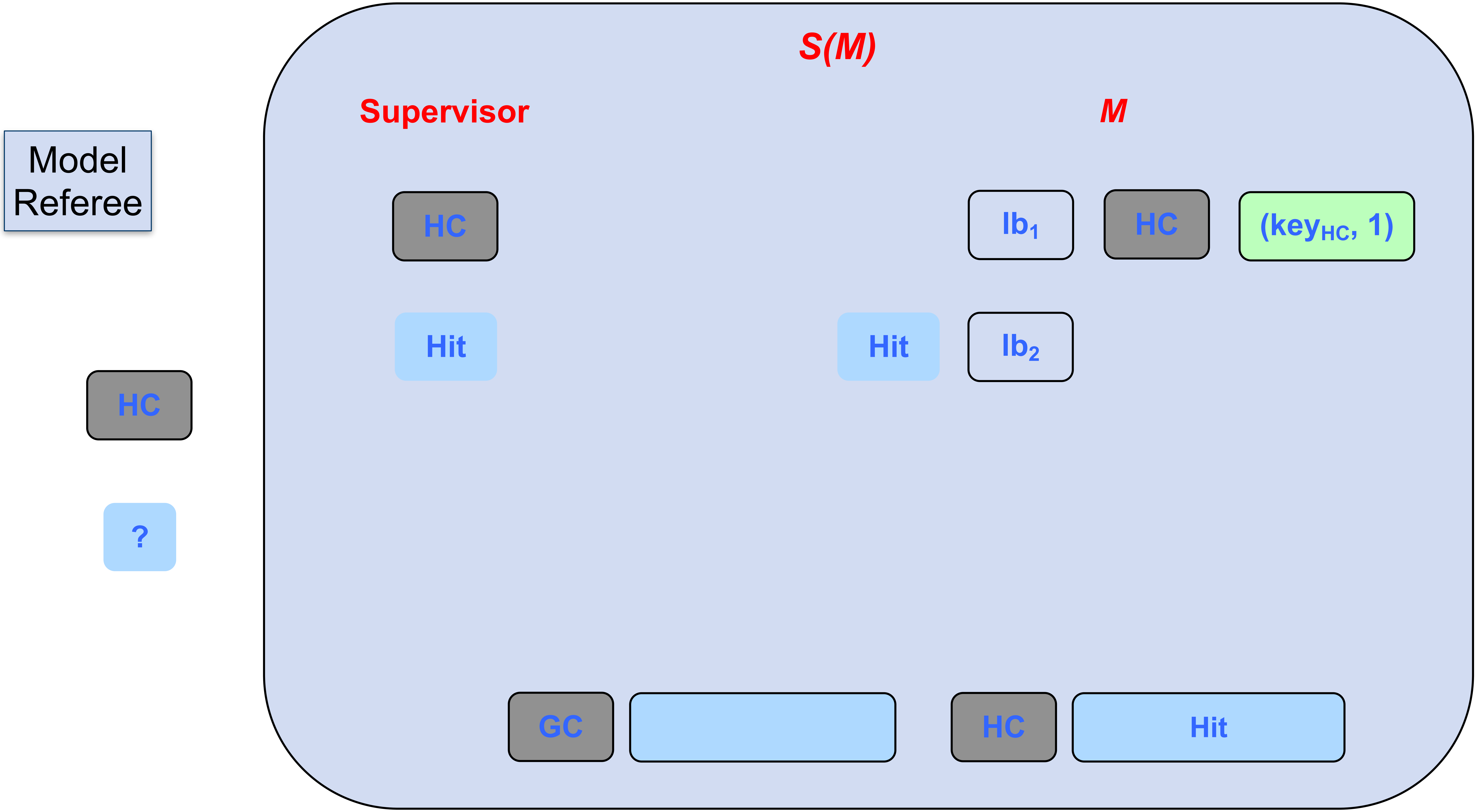
CML + AC Simulator Example



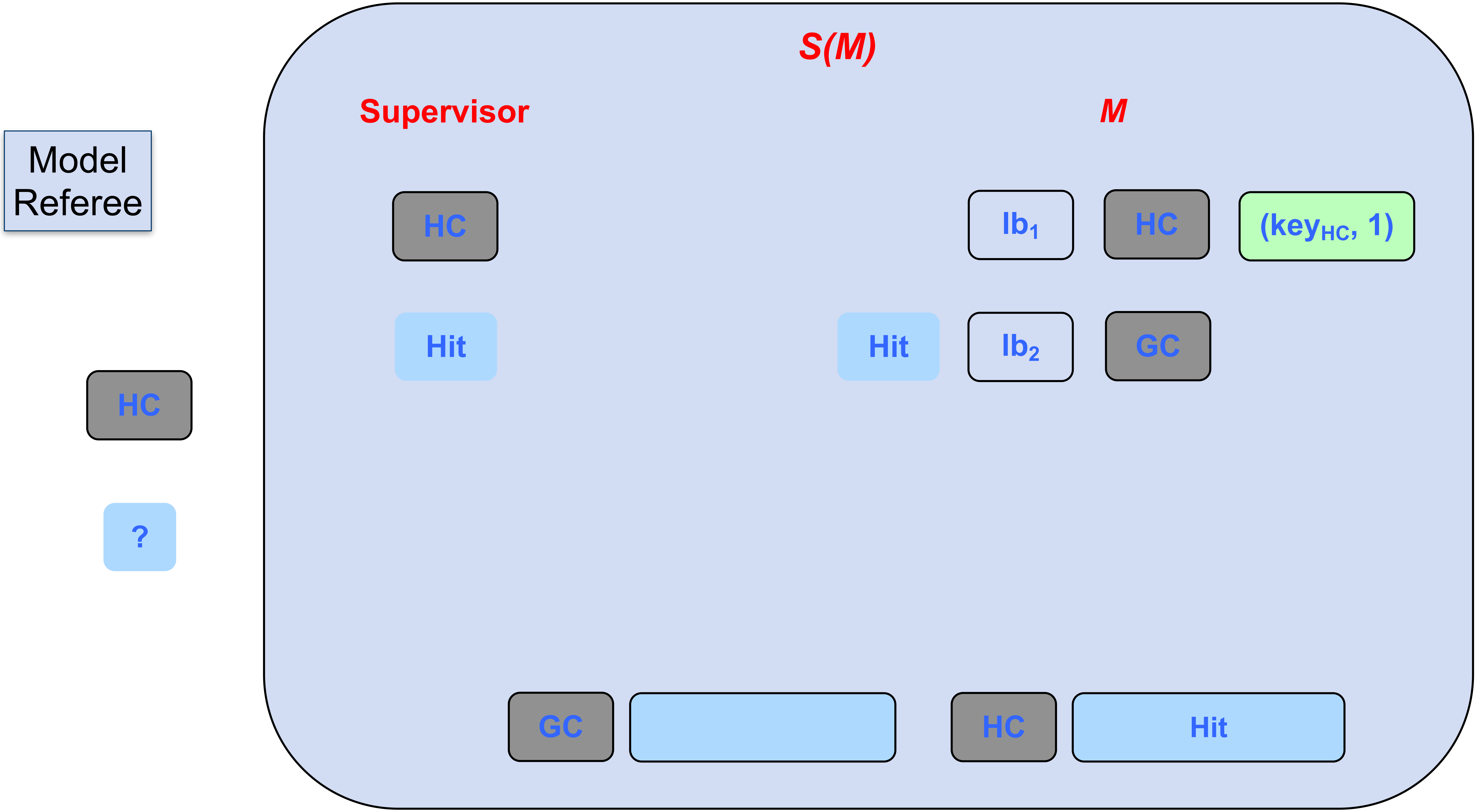
CML + AC Simulator Example



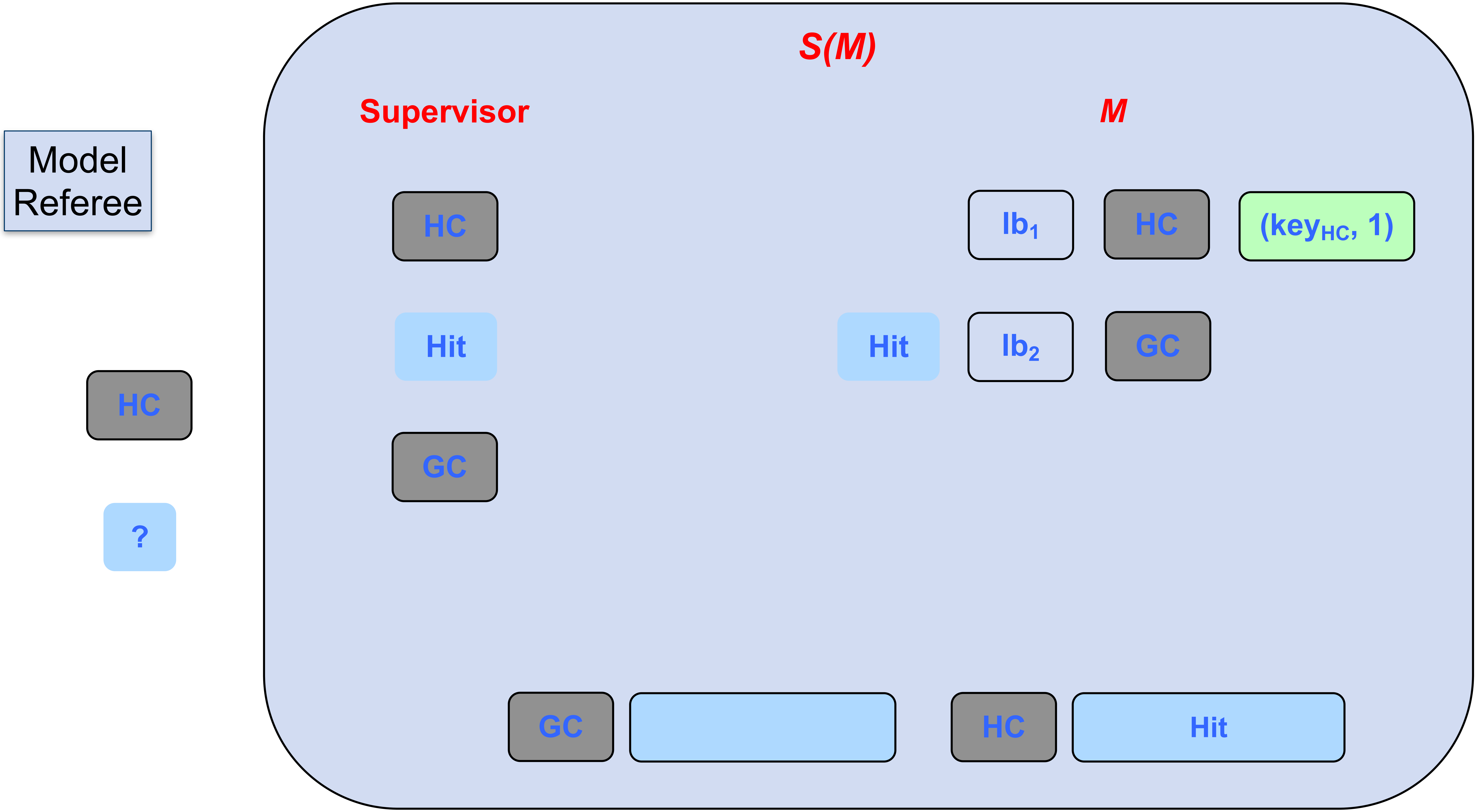
CML + AC Simulator Example



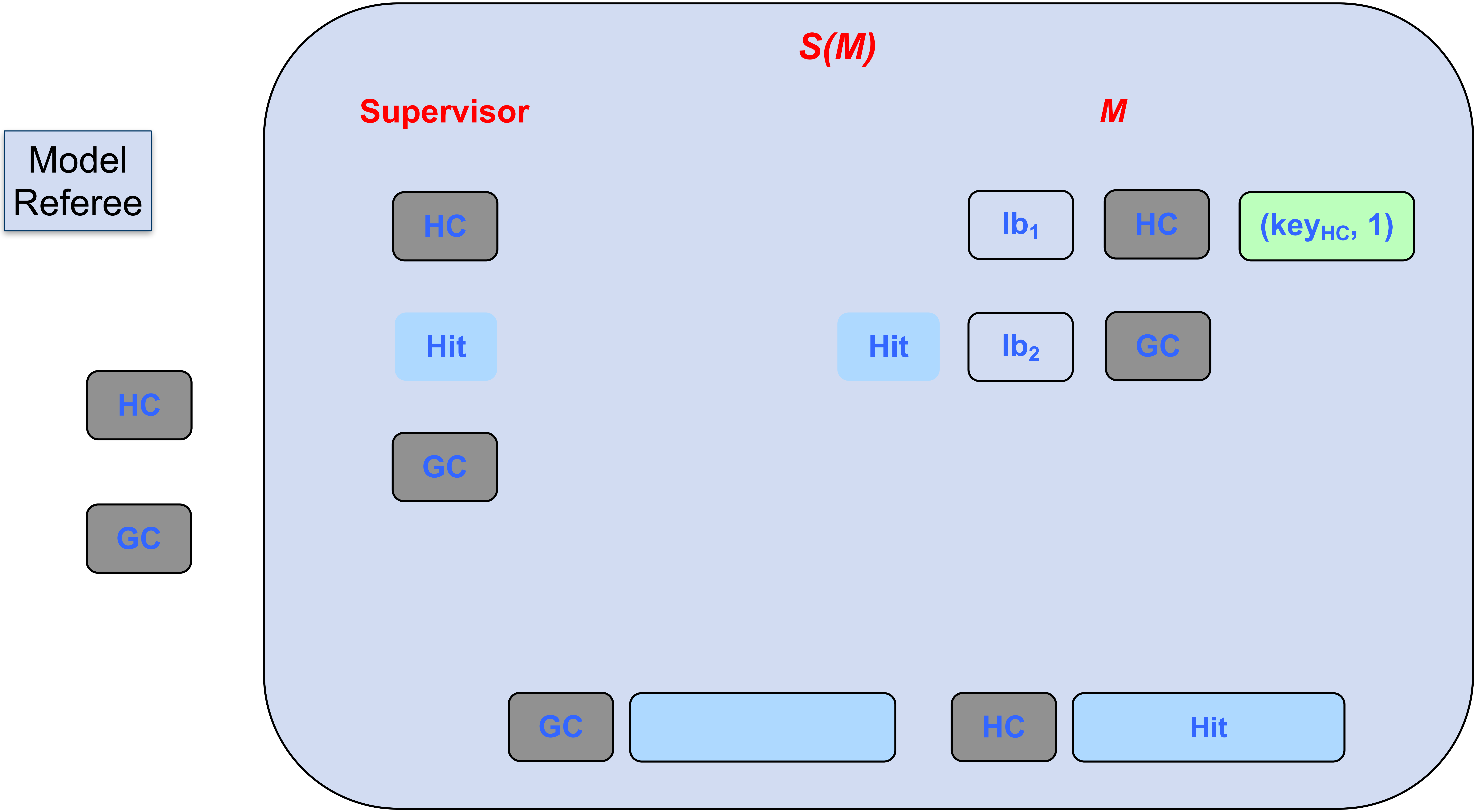
CML + AC Simulator Example



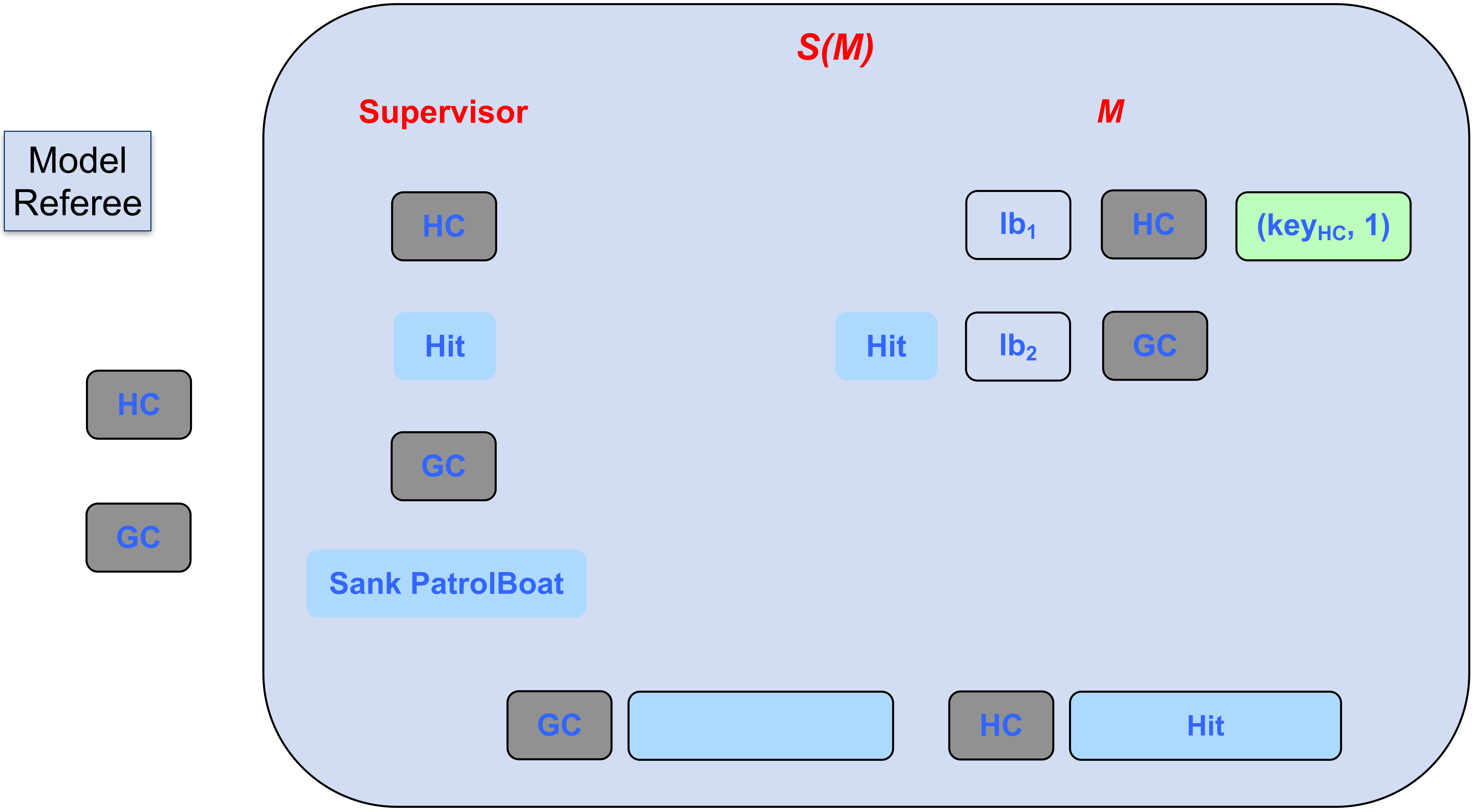
CML + AC Simulator Example



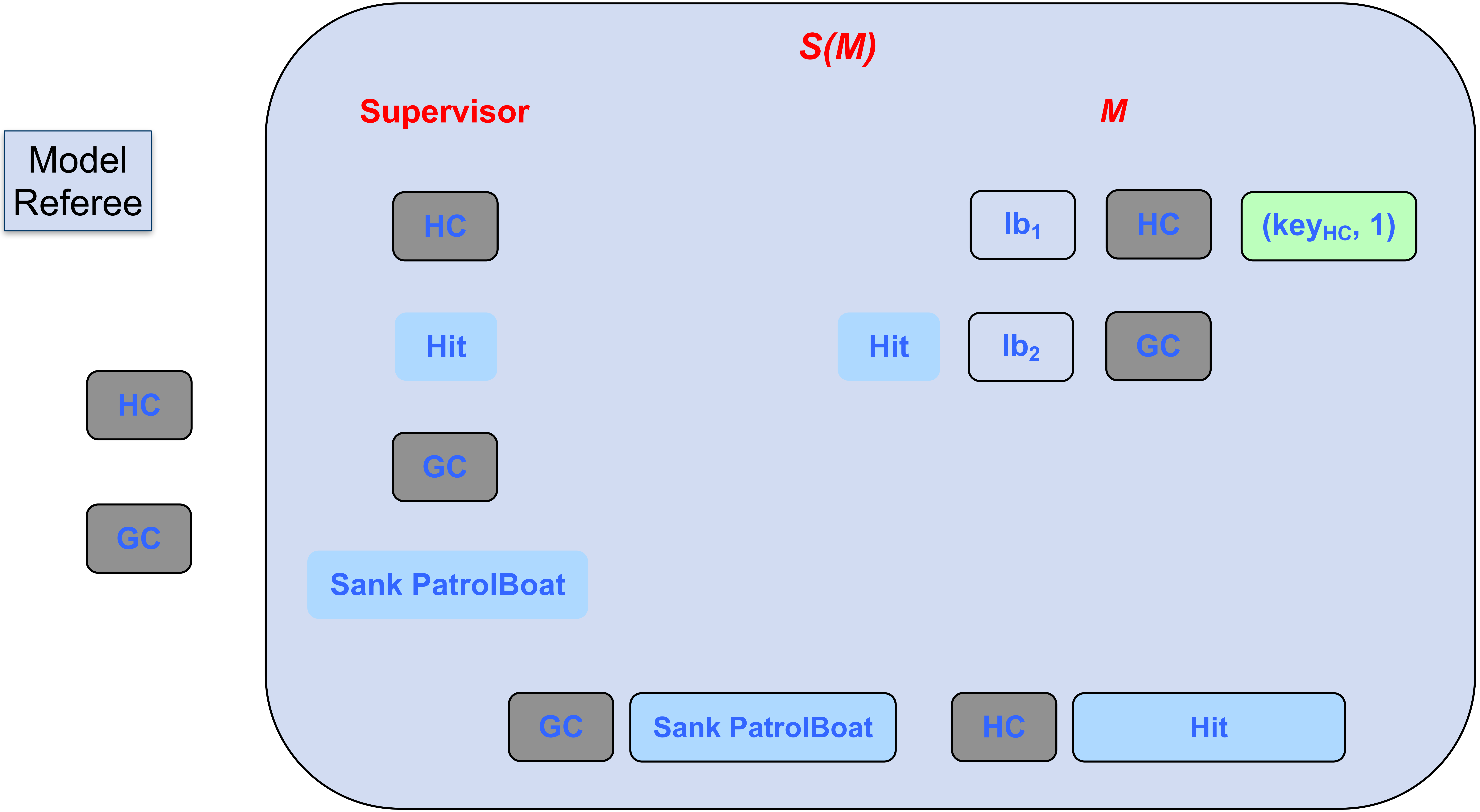
CML + AC Simulator Example



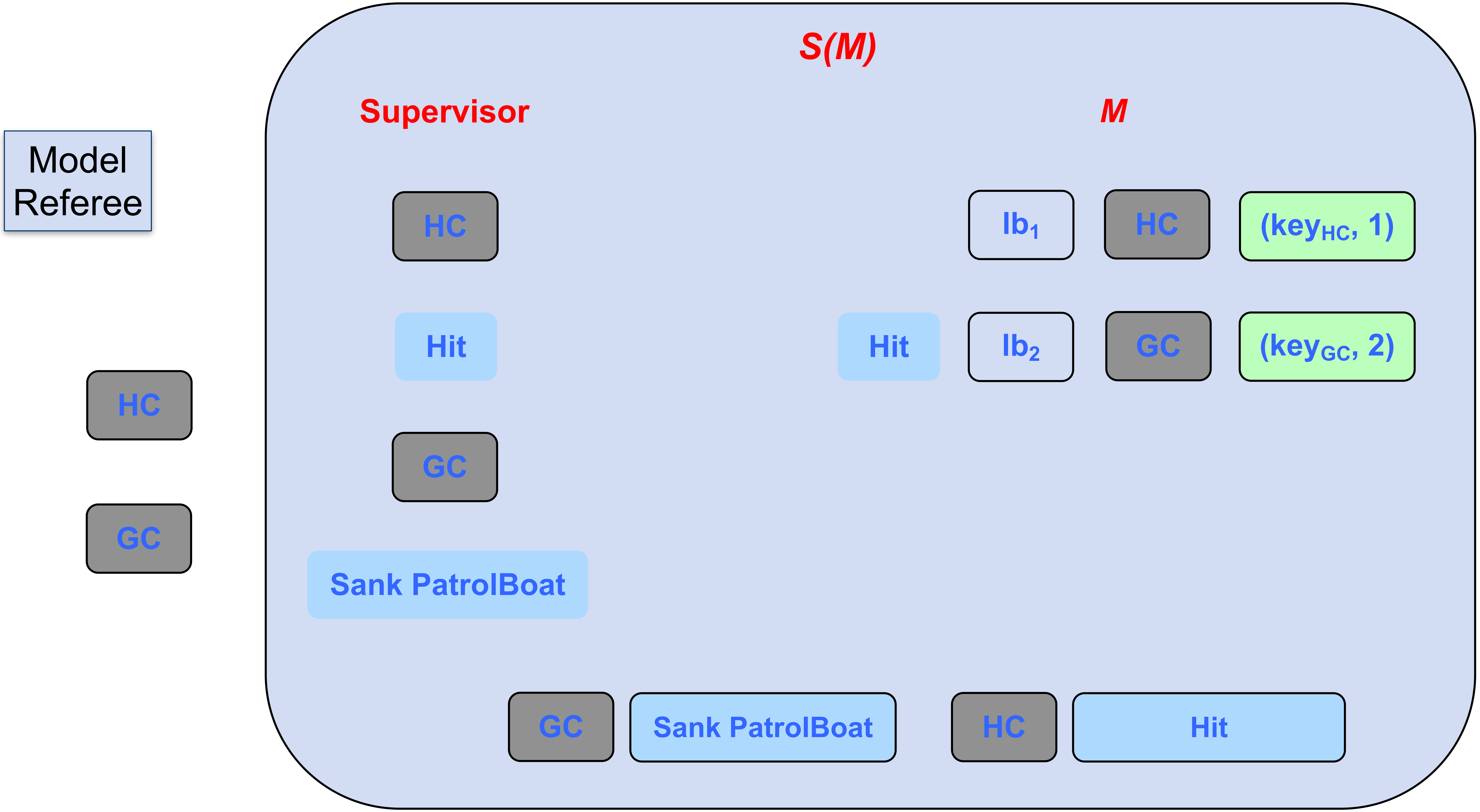
CML + AC Simulator Example



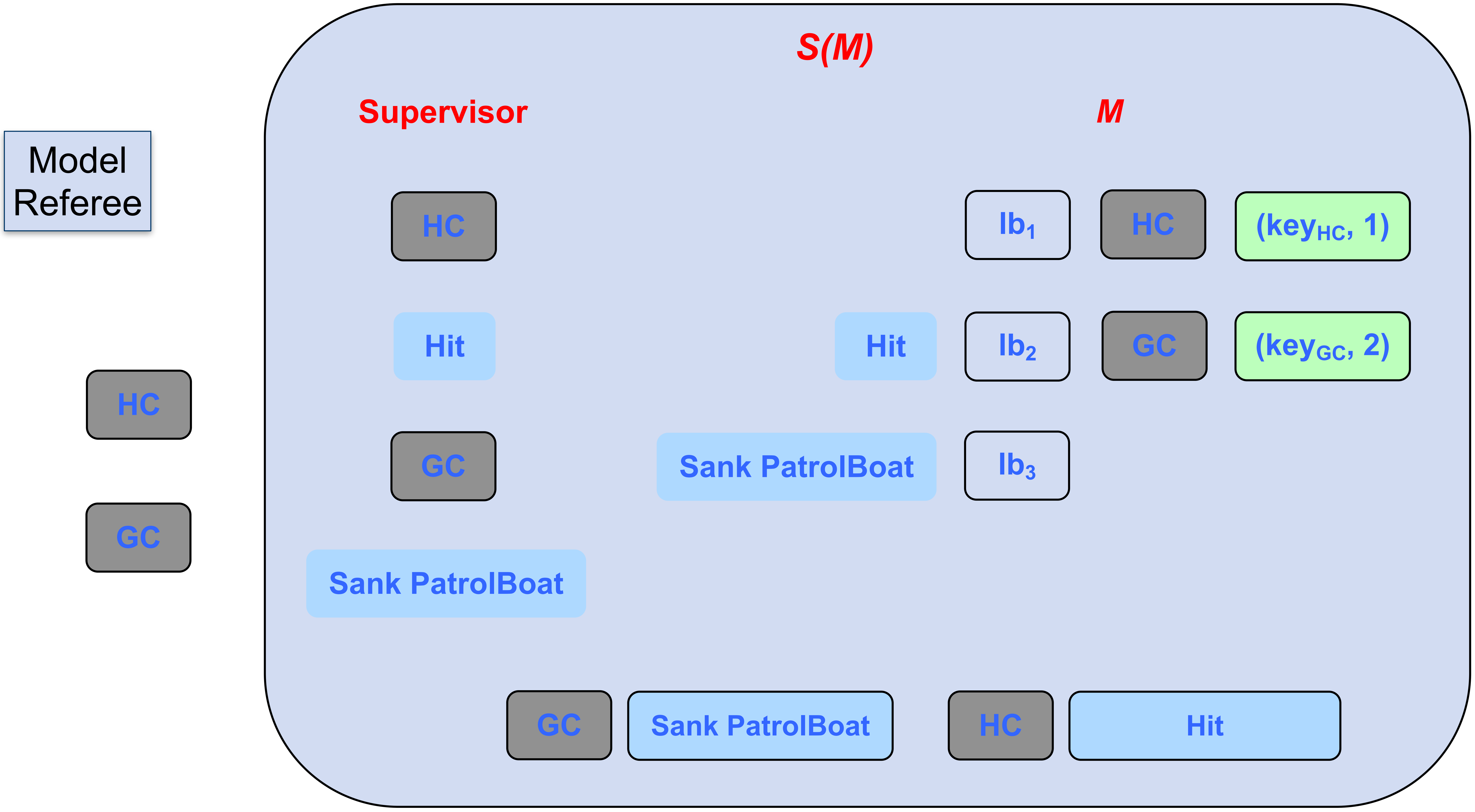
CML + AC Simulator Example



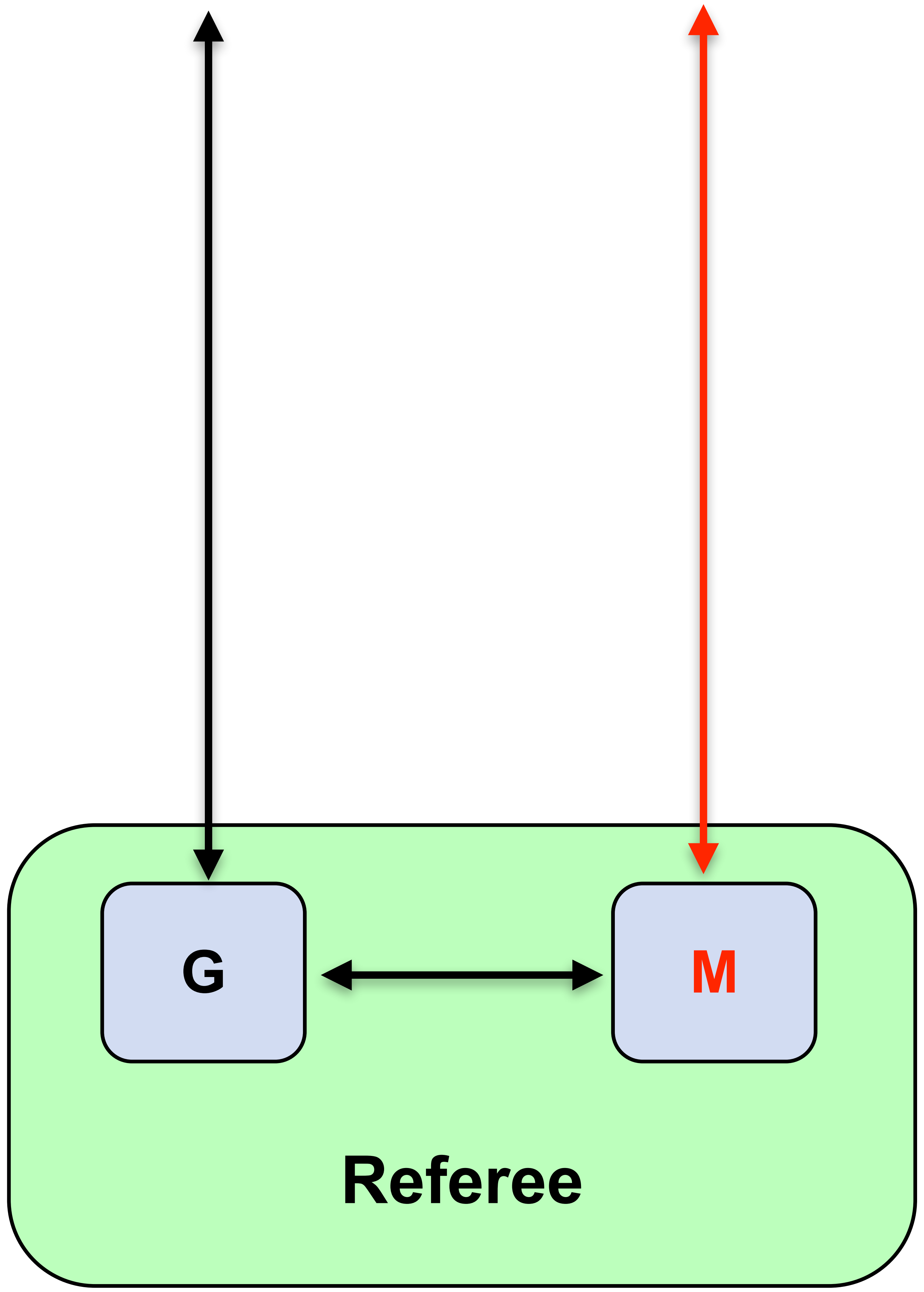
CML + AC Simulator Example



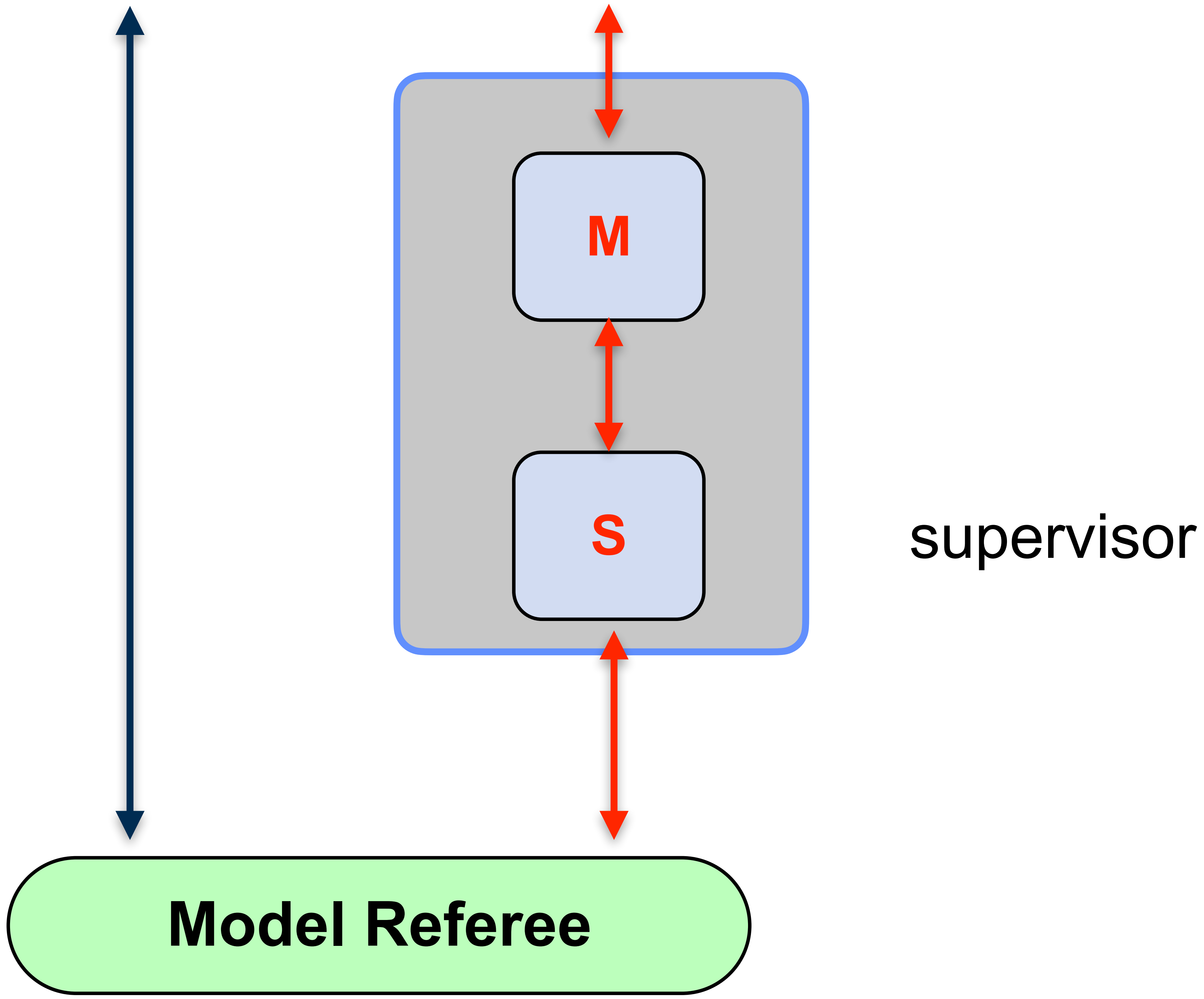
CML + AC Simulator Example



CML + AC: M Commits to a Board

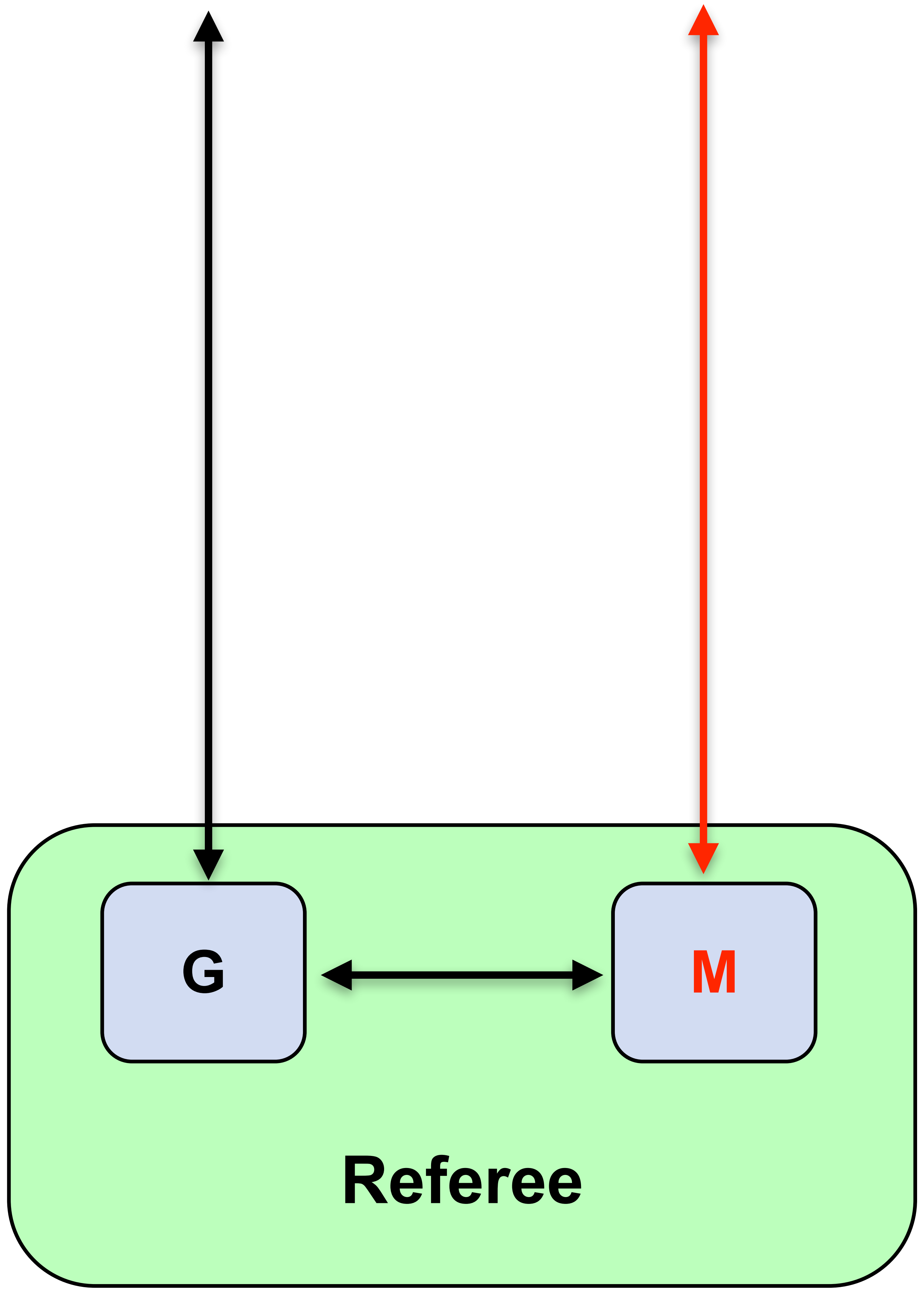


abstract type has *two* kinds of locked boards: one for shooting and one for extraction; S extracts from the locked board M provides its source board, to give to G

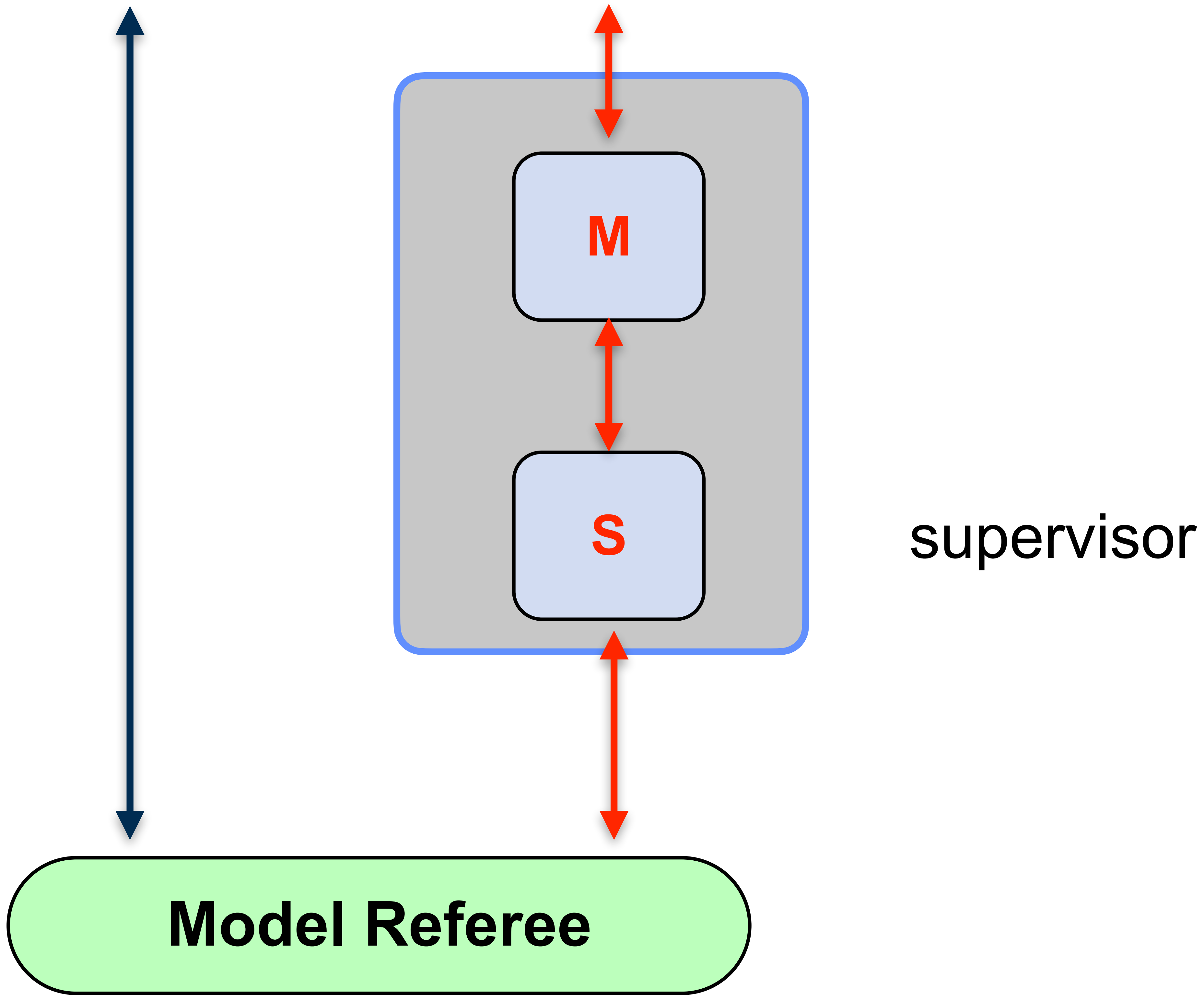


Q: What is the potential pitfall with this approach?

CML + AC: M Commits to a Board



abstract type has *two* kinds of locked boards: one for shooting and one for extraction; S extracts from the locked board M provides its source board, to give to G



A: A replay attack in which M gives G back its own locked board must be prevented

Conclusions

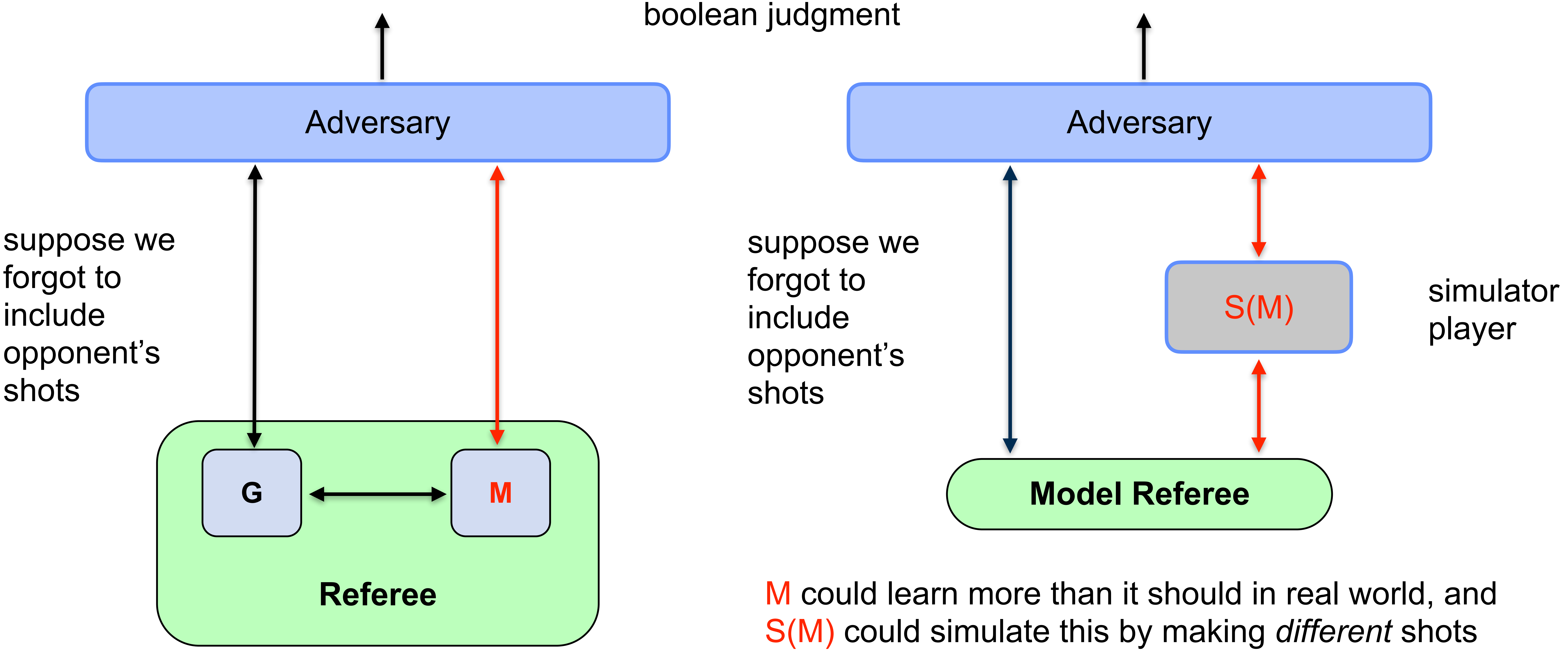
- We used theoretical cryptography's real/ideal paradigm to define when one program interface is secure against a possibly malicious program interface
  - This separates the definition of security from its enforcement
- We gave two secure implementations, using our definition to guide our design and *informally audit* it
  - Using LIO and information flow control
  - Using Concurrent ML + access control
- We found numerous security bugs during our audits

Research Questions

- How do we know that a real/ideal paradigm definition says what we want?
  - Designing ideal functionalities is something of an art, and tools for making their design easier would be useful
  - Tools for helping the designer know they got the correct definition would also be helpful

How Do We Know This Is What We Want?

boolean judgment



M could learn more than it should in real world, and S(M) could simulate this by making *different* shots

Research Questions

- What are alternatives to the real/ideal paradigm for defining the security of one component against another?
- When is it useful to split a trusted component into two mutually distrustful ones?
  - For Battleship, are there solutions relying on smaller trusted computing bases?
- When is information flow control necessary to achieve security?
  - Why did Battleship not require information flow control?

Future Work

- We want to formalize our results using a proof assistant
- It must be possible to formalize and reason about a programming language with
  - A rich module system, supporting abstract types
  - Concurrency
  - Mutable references
- We need to be able to reason about thread scheduling
- We are currently investigating whether the Coq development of the concurrent separation logic Iris would be a good vehicle for this work