Specifying Binding and Commitment with Ghost Outputs and Strong Refinement

Hagit Attiya, Technion

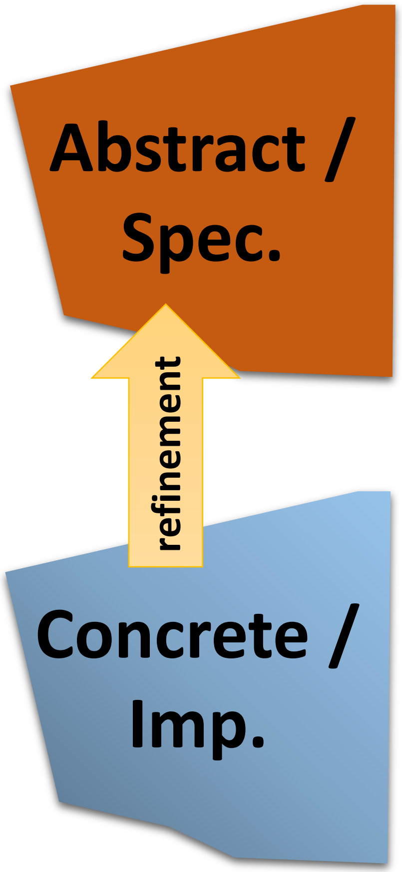
With: Constantin Enea, Jennifer Welch and Itay Flam

Abstraction for Distributed Protocols

Abstraction and refinement are successful tools for specifying and verifying concurrent data structures

Less so for distributed protocols, which are often specified by listing their properties

- ☝ Part of the problem is inadequate specification tools
- ☝ Suggest a path for (partially) dealing with it



Case in Point: Crusader Agreement

[Dolev, 1982]

Agreement A

(A1) All the reliable processors that do not explicitly know that z is faulty agree on the same message.

(A2) If z is reliable, then all the reliable processors agree on its message.

Agreement B

(B1) All the reliable processors agree on the same message.

(B2) If z is reliable, then all the reliable processors agree on its r

Agreement B was named the *Byzantine Generals Problem* by Lampo
[3]. Here it is referred to as the *Byzantine Agreement*. For con
Agreement A is called the *Crusader Agreement*.

Crusader Agreement, More Precisely

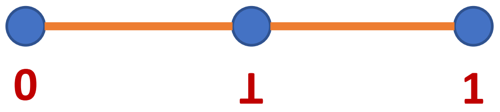
[Abraham, Ben-David, Yandamuri, 2022]

As in standard Crusader Agreement [15], this functionality is similar to agreement, but allows some parties to output a special $\perp$ value. More specifically, it guarantees (1) Validity: when all non-faulty parties have the same input, this is the only output; and (2) Agreement: no two honest parties output two distinct non-$\perp$ values.

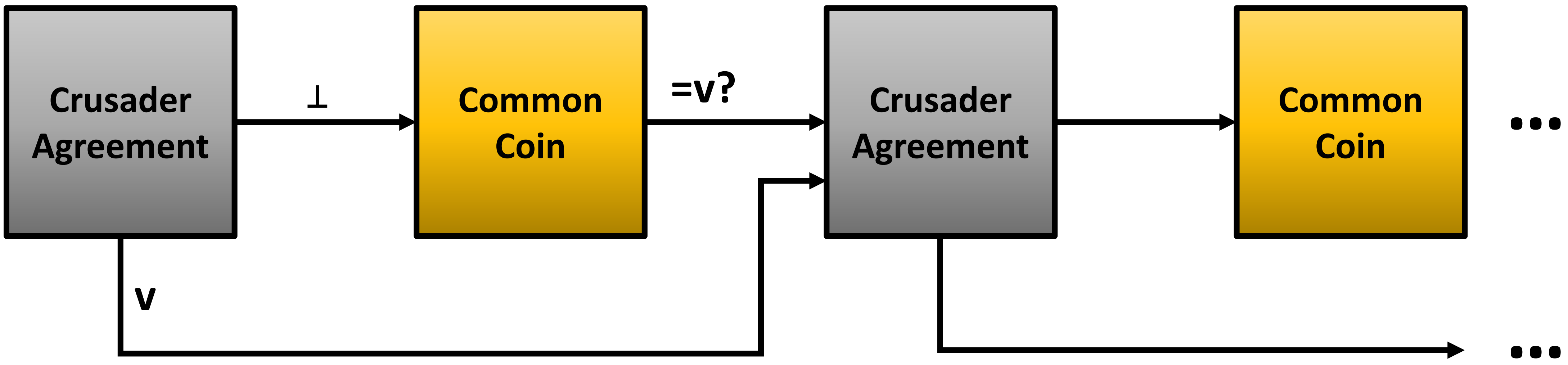
[Attiya, Welch, 2023]

A process starts at one of the end vertexes and decides on a vertex, s.t.

1. If all start at the same vertex ⇨ decide on this vertex (validity)
2. Decided vertexes are adjacent (agreement)

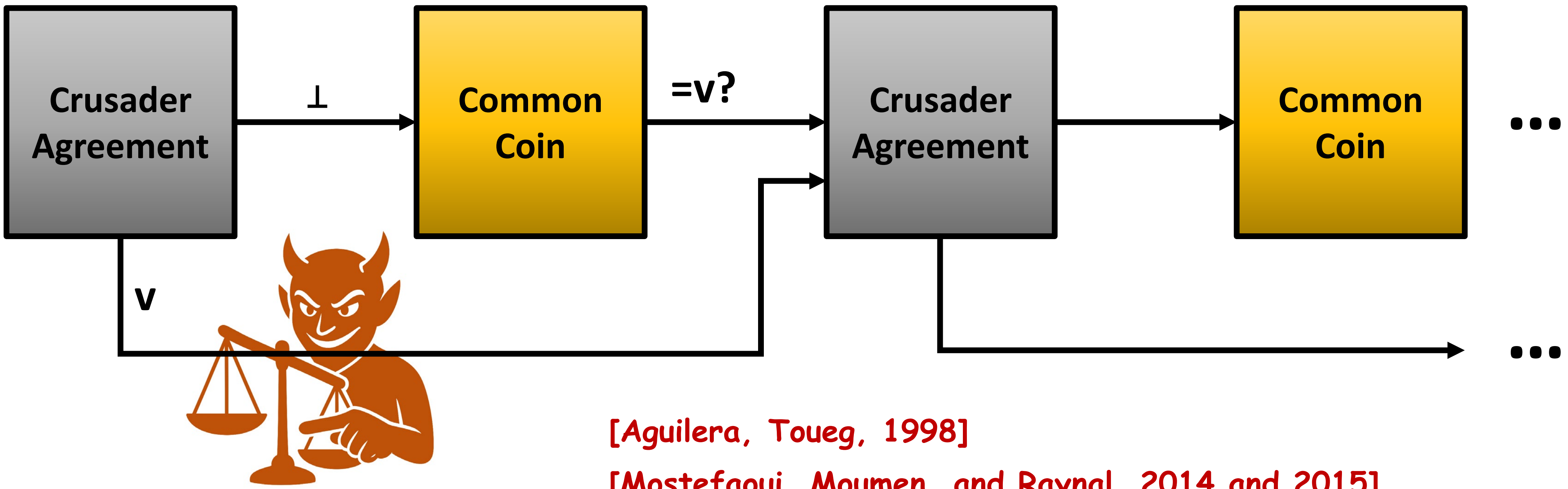


Crusader Agreement in Randomized Consensus (Simplified)



Crusader Agreement in Randomized Consensus: What Could Go Wrong?

Adaptive adversary can exploit the uncertainty to prohibit termination

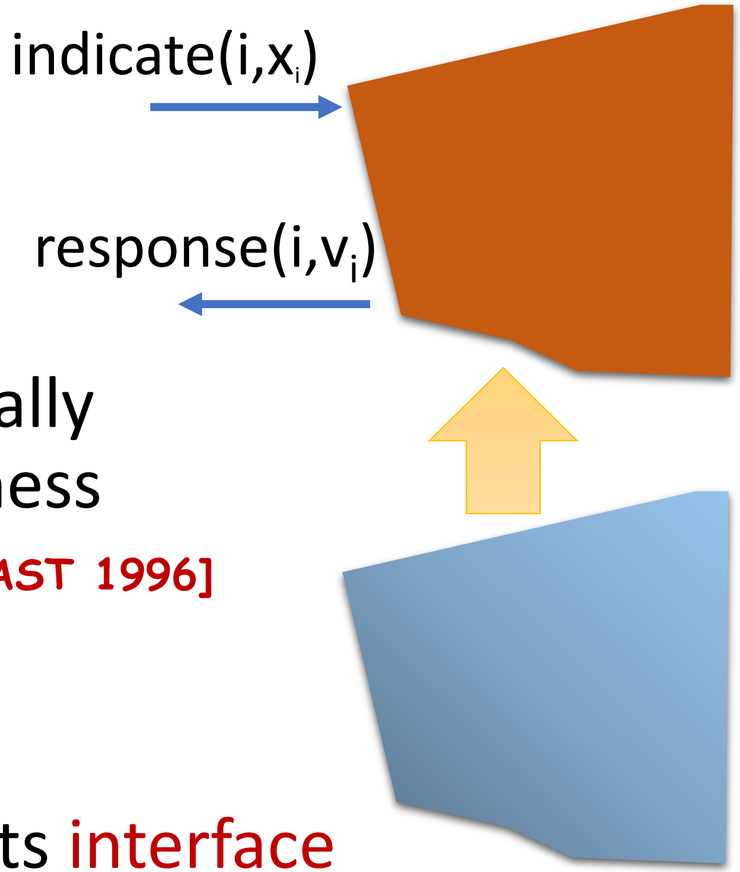


[Aguilera, Toueg, 1998]

[Mostefaoui, Moumen, and Raynal, 2014 and 2015]

Crusader Agreement Specification

Use a **ghost** (auxiliary) variable to capture the non-$\perp$ value



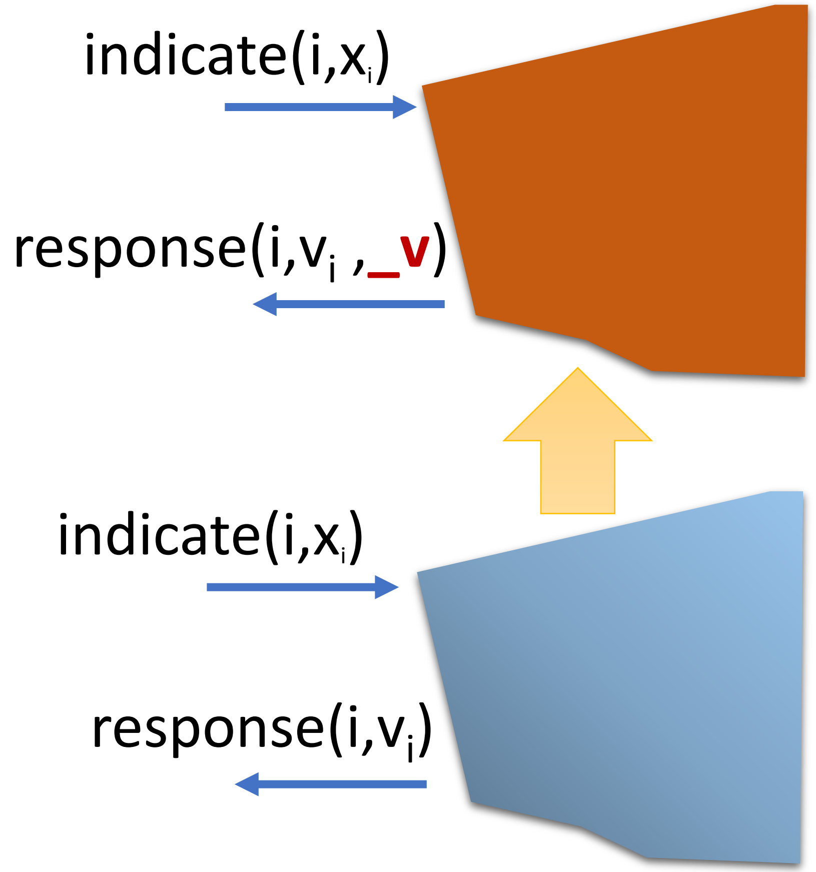
Auxiliary variables (history or prophecy) are typically added to an implementation to prove its correctness

[Abadi, Lamport 1991] [Marcus, Pnueli AMAST 1996]

We'll add them to the specification, in part., its interface

Crusader Agreement with a Ghost Output

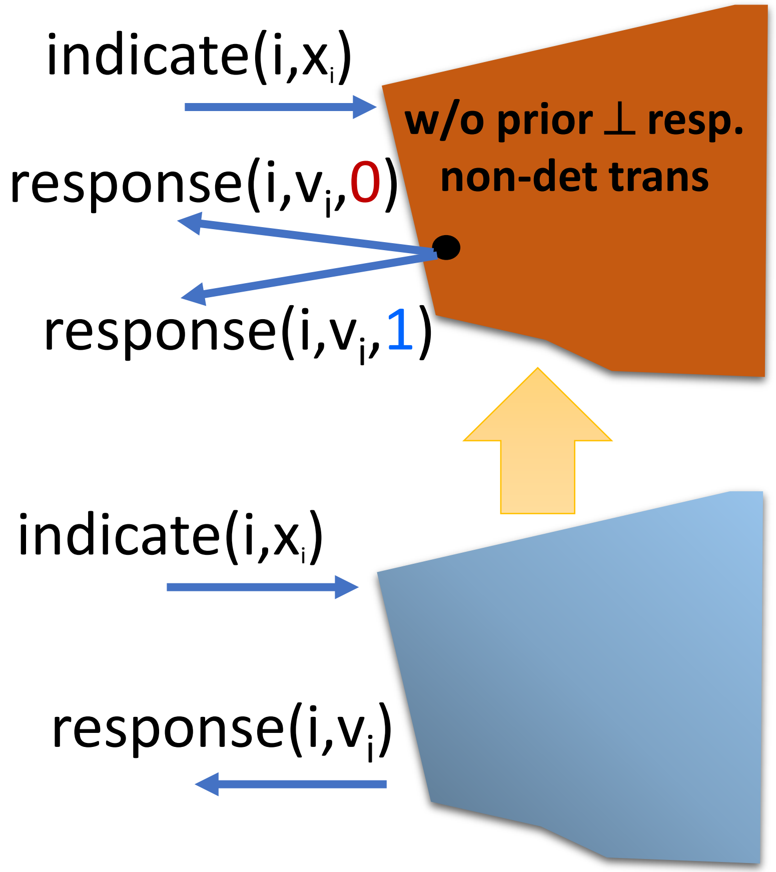
- _v is the same in all responses
- _v is the input of some correct process
- If v_i <> $\perp$ then v_i = _v



Crusader Agreement with a Ghost Output

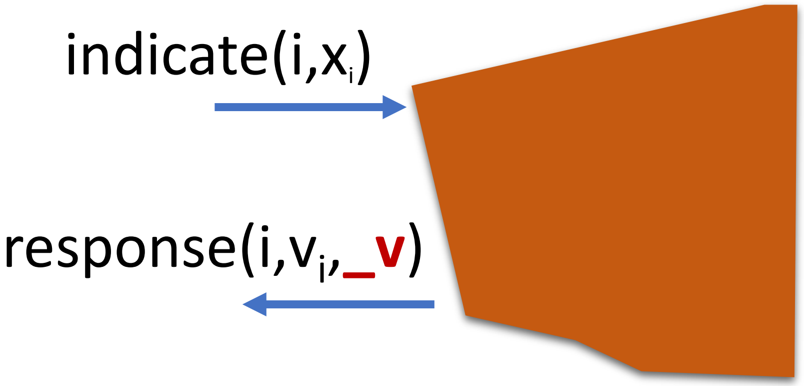
- _v is the same in all responses
- _v is the input of some correct process
- If v_i <> ⊥ then v_i = _v

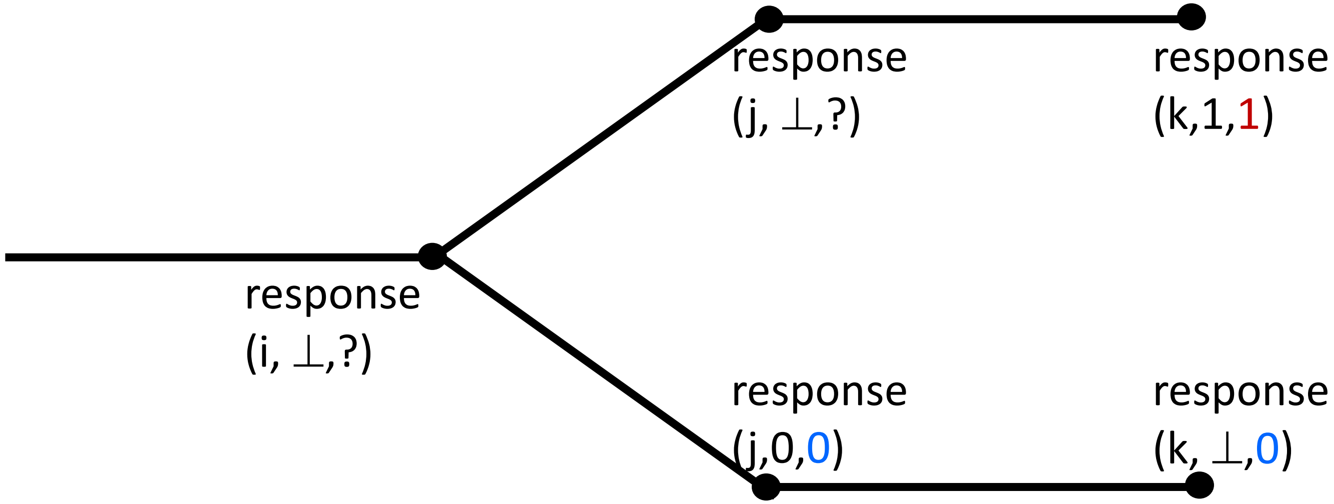
Hides a non-deterministic choice of the ghost output



Crusader Agreement with a Ghost Output

- _v is the same in all responses
- _v is the input of some correct process
- If v_i <> $\perp$ then v_i = _v

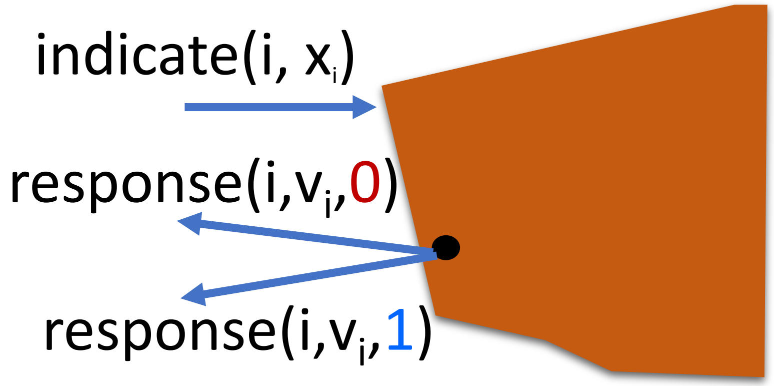


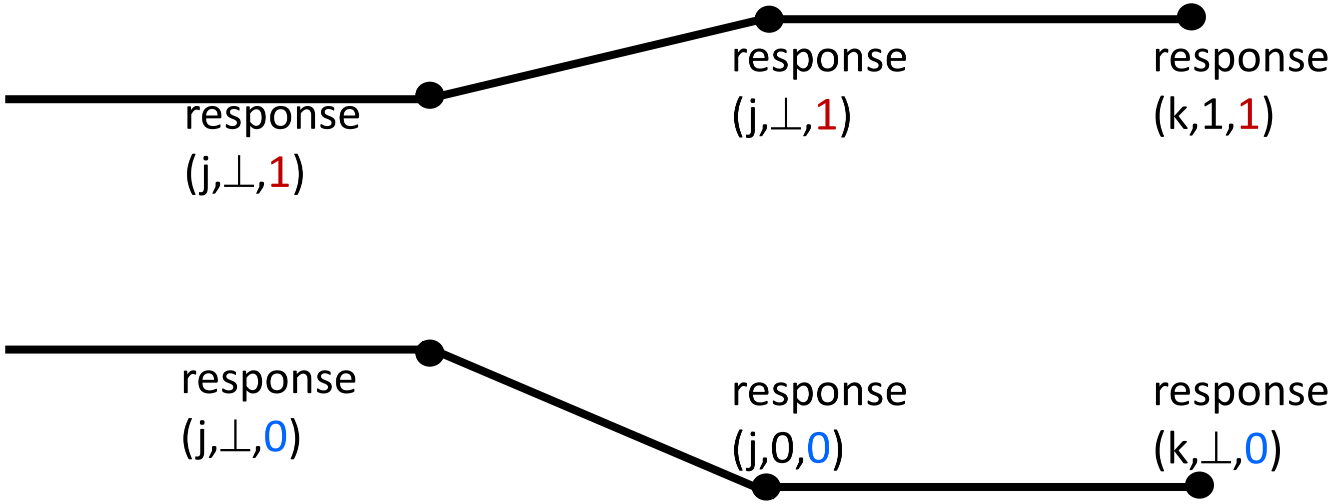


Trace Property

Crusader Agreement with a Ghost Output

In a CA implementation,
_v might be determined by the future,
i.e., it is a prophecy variable

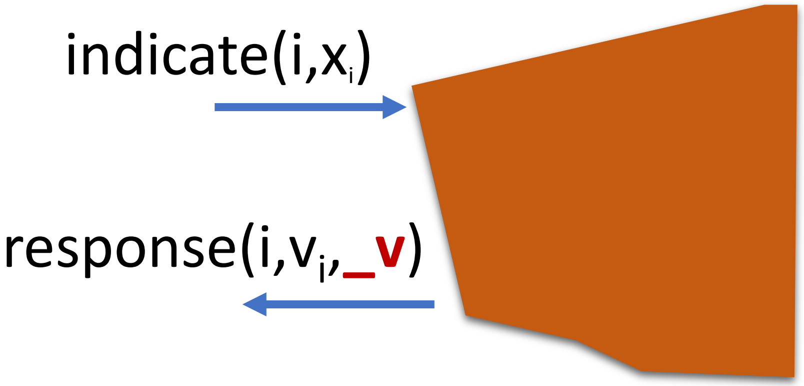




Trace Property

Binding Crusader Agreement

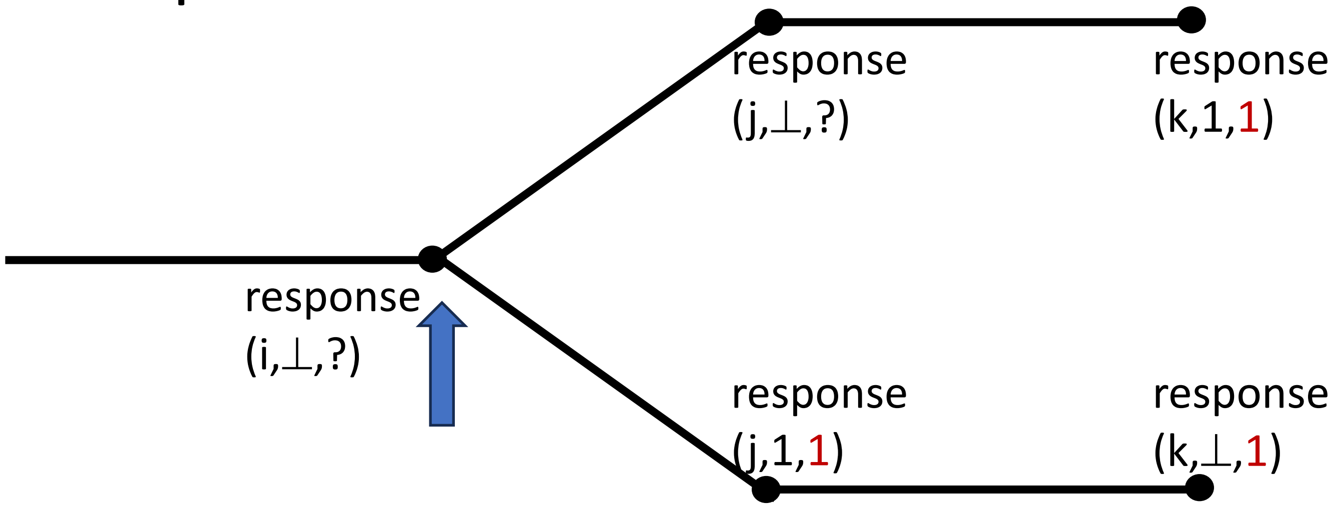
[Abraham, Ben-David, Yandamuri, 2022][Attiya, Welch, 2023]

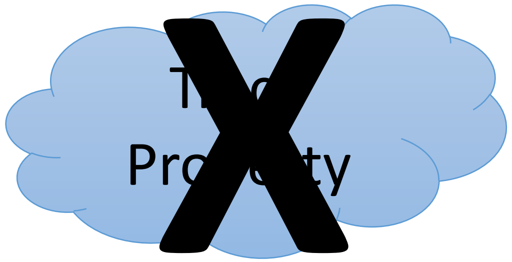


Same non-⊥ value is decided in all extensions
after the first correct process returns

Adversary cannot bias the response

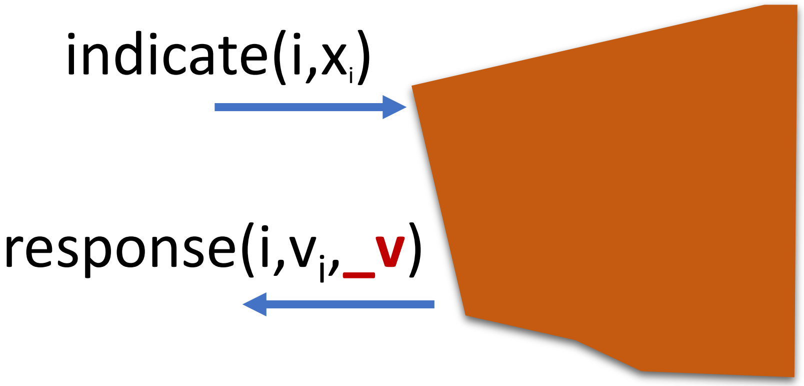
Not a trace property

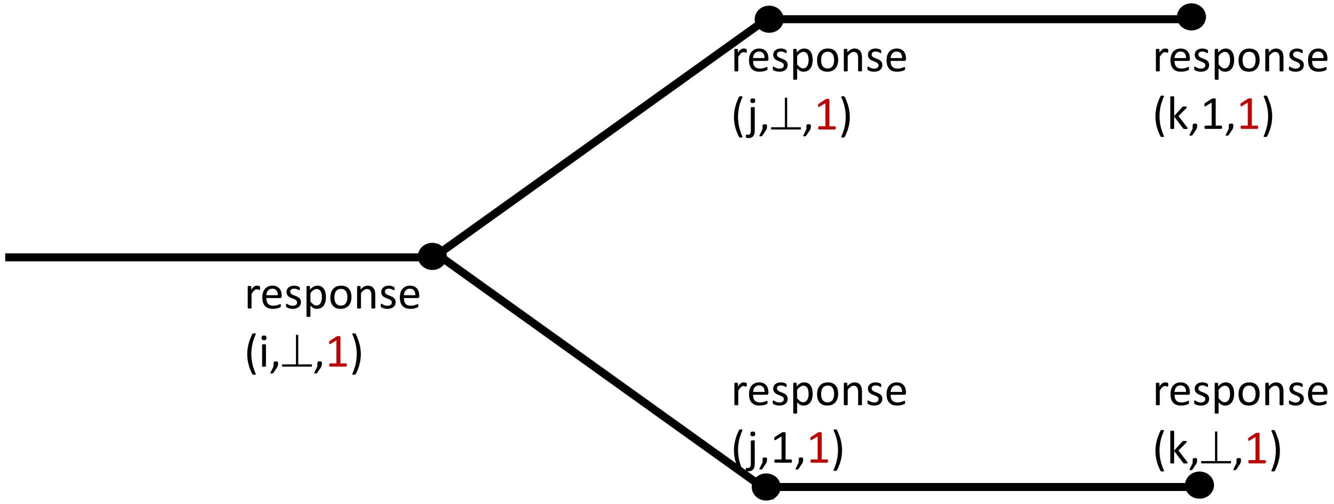


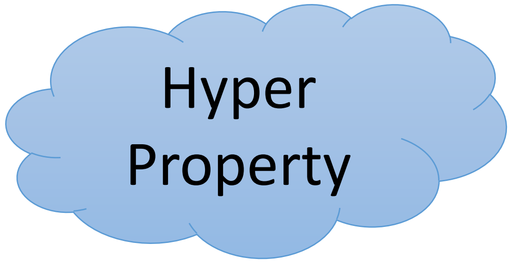


Binding Crusader Agreement

Same non-⊥ value is decided in all extensions
after the first correct process returns

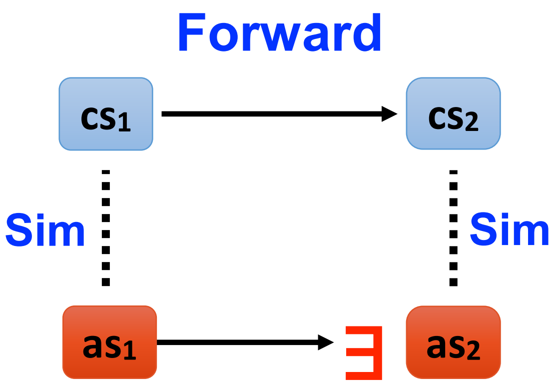


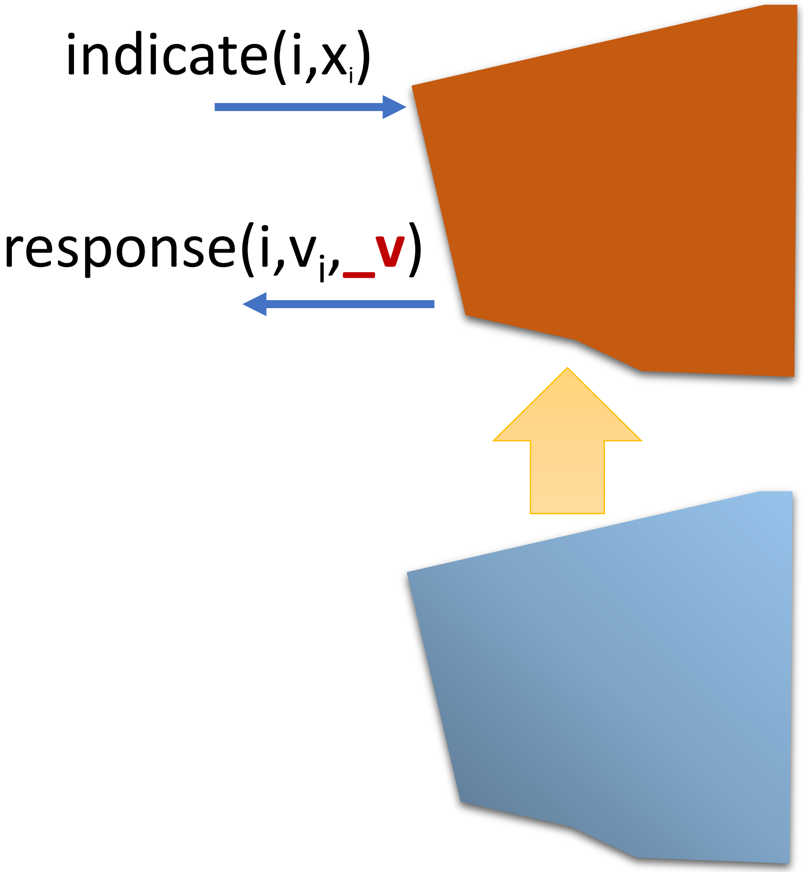




Implementing the CA Specification

Refine the CA specification by relating states of the abstract and concrete objects

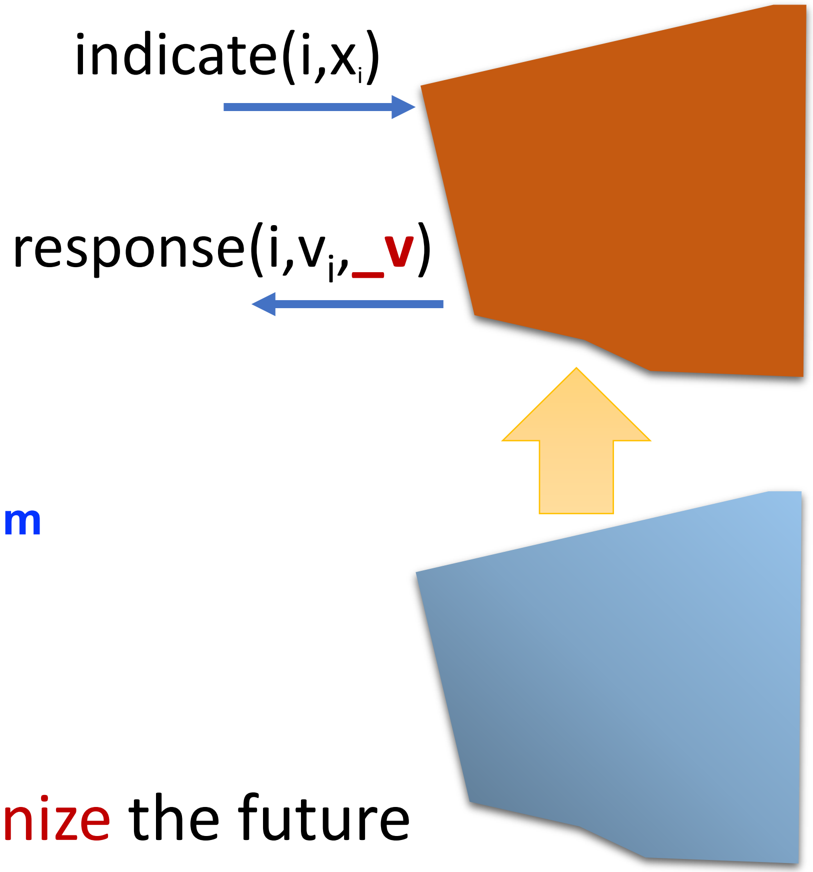




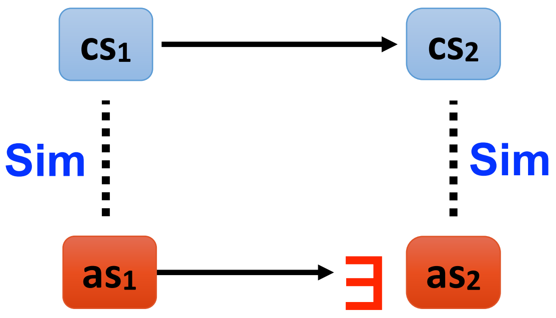
Forward simulations rely only on history leading to the current state

Implementing the CA Specification

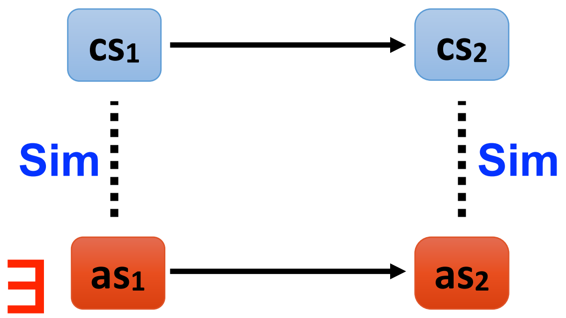
Refine the CA specification by relating states of the abstract and concrete objects



Forward



Backward



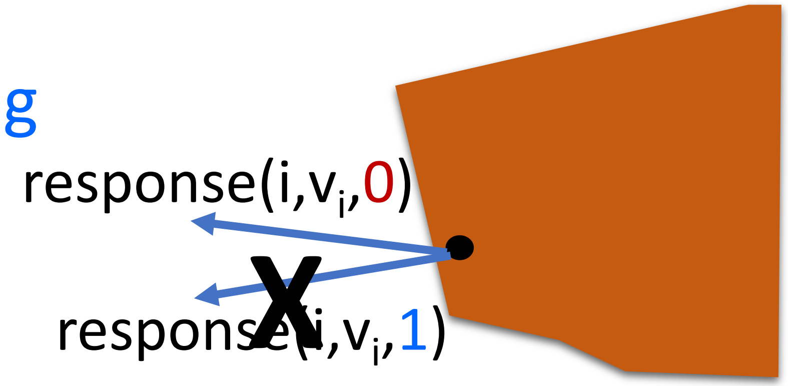
Backward simulations are prophecies that determinize the future

Refinement can be proved by forward & backward simulation

[Lynch, Vaandrager][Jonsson]

Binding CA and Strong Refinement

A crusader agreement implementation is binding
if it is a strong refinement of the specification



$$\text{Obj} \leq_S \text{Spec iff } \forall \text{ program P}, \forall \text{ deterministic scheduler } S_1 \text{ of P X Obj},$$
$$\exists \text{ deterministic scheduler } S_2 \text{ of P X Spec},$$
$$\text{Traces(P X Obj X } S_1) = \text{Traces(P X Spec X } S_2)$$

≡Forward Simulation from the implementation to the specification

[Attiya, Enea, DISC 2019]

_v is a history variable (no non-determinism)

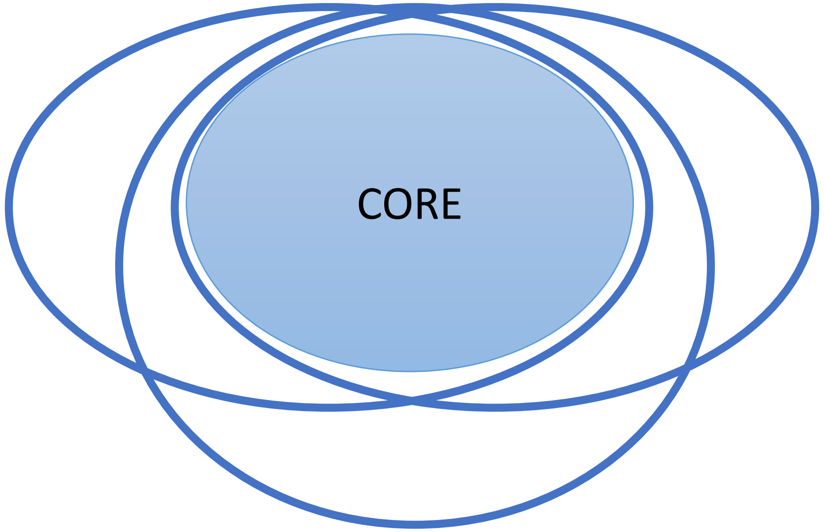
Another Example: Gather

Implicit in [Canetti, Rabin 1993]

[Abraham, Jovanovic, Maller, Meiklejohn, Stern, and Tomescu 2021]

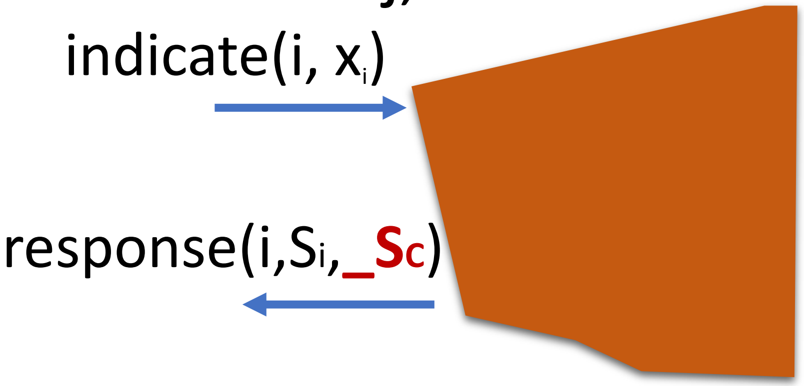
Gather is a natural *multi-dealer* extension of Reliable Broadcast where every party is also a dealer. The output of a gather protocol is a *gather-set*. A gather-set consists of *at least* $n - f$ pairs (j, x), such that $j \in [n]$, $x \in \mathcal{M}$, and each index j appears at most once. For any given gather-set X, we define its index-set $Indices(X) = \{j | \exists (j, x) \in X\}$ to be the set of indices that appear in X.

Intuitively speaking, the goal of Gather is to have some common *core* gather-set such that all parties output a super-set of this core.



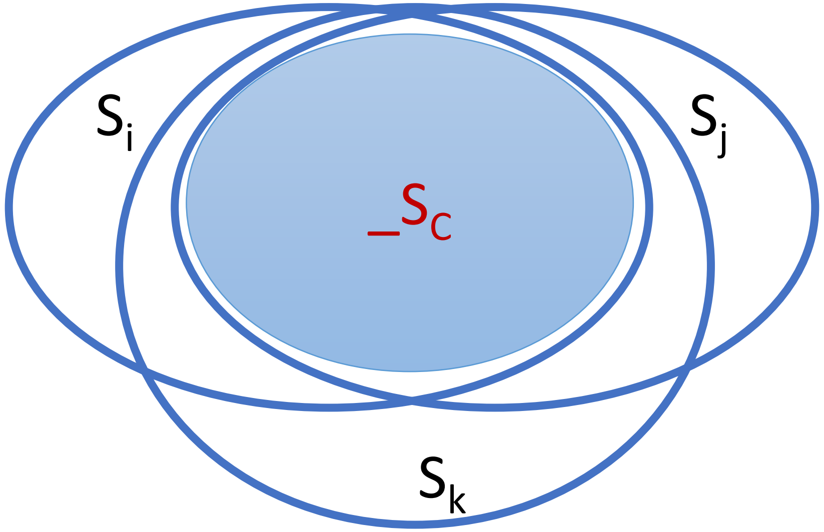
Gather with a Ghost Output

✓For process k, and every pair of nonfaulty processes i and j, if (k,x_i) is in S_i and (k,x_j) is in S_j, then $x_i = x_j$



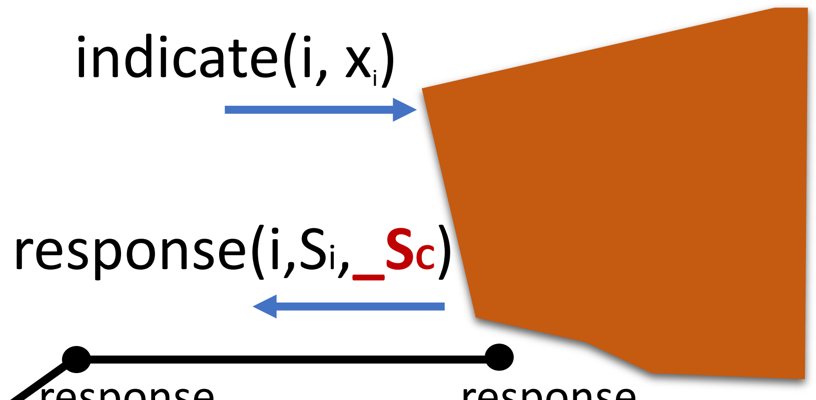
✓For every pair of correct processes i and j, if (j,x) is in S_i, then $x = x_j$

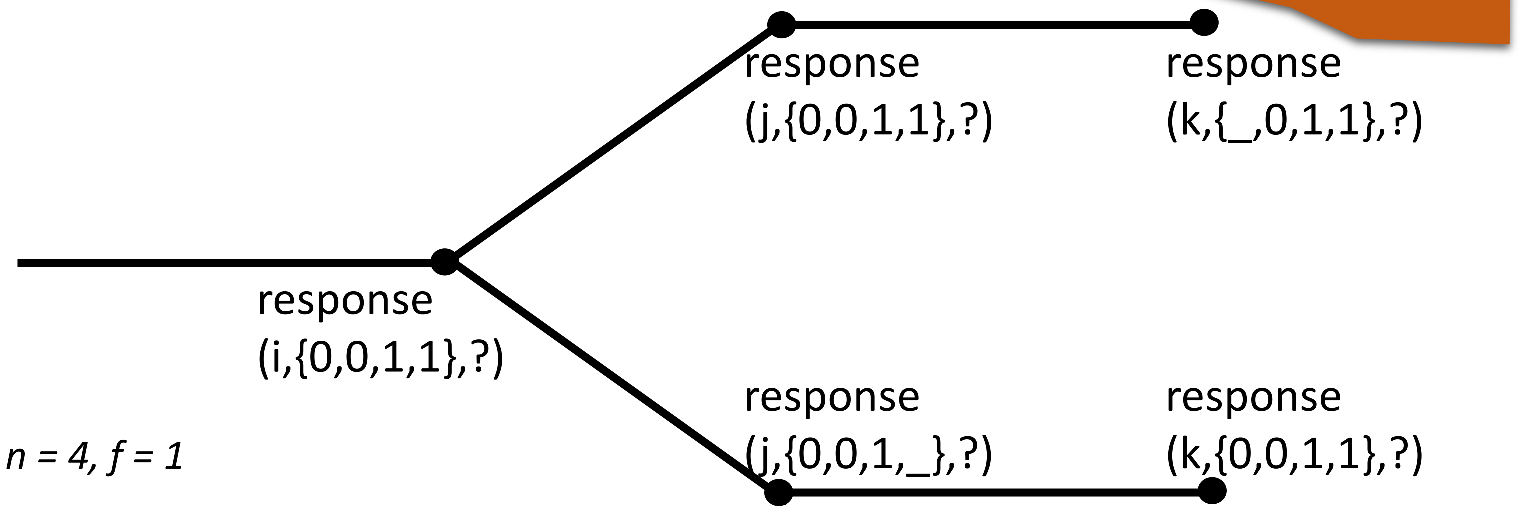
- For every correct process, $S_i \supseteq$ _S_C
- |_S_C| $\geq n - f$ ⇨ $|S_i| \geq n - f$



Common Core

there is a set of n-f processes, whose values appear in the set of every correct process



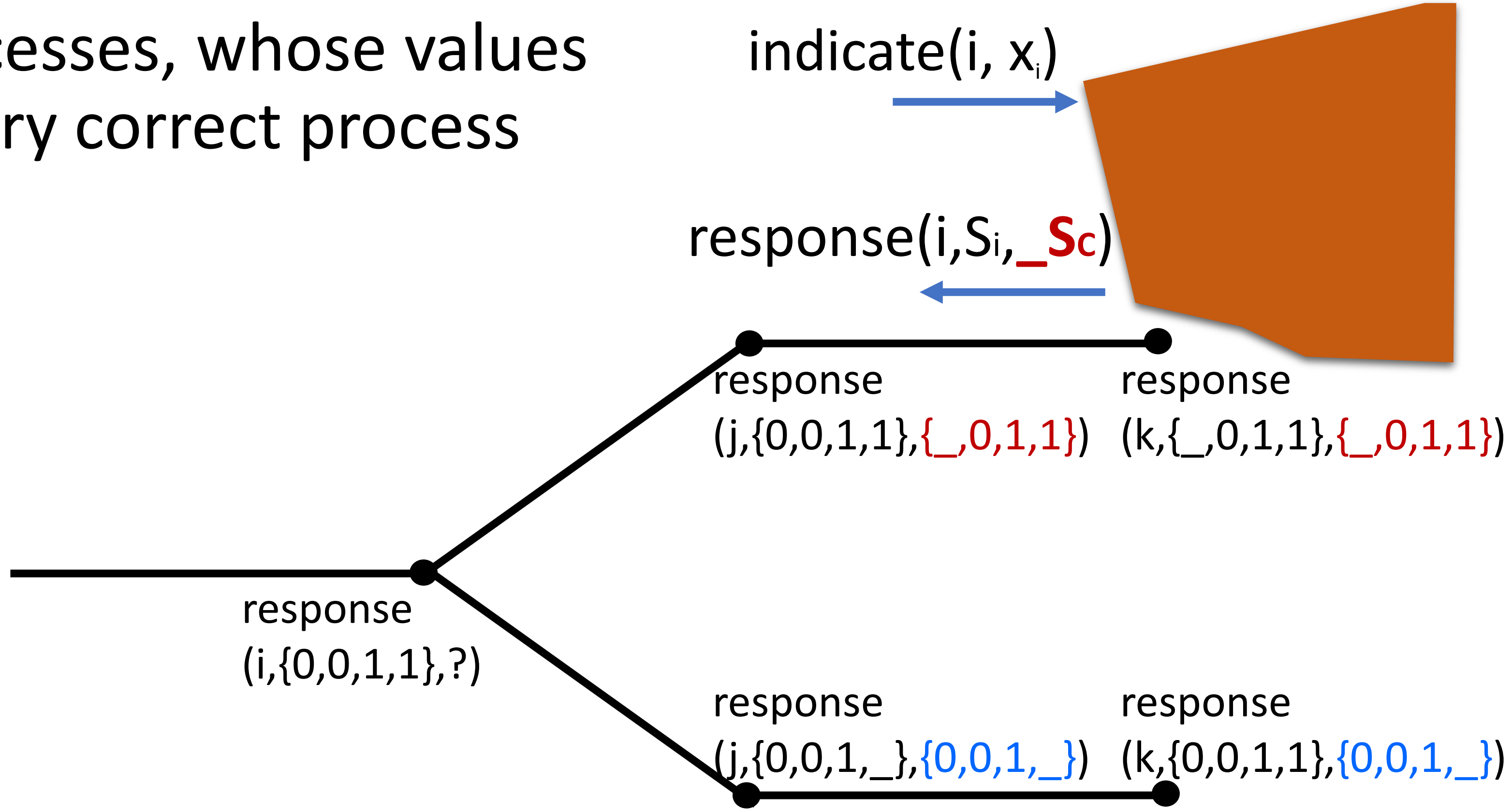


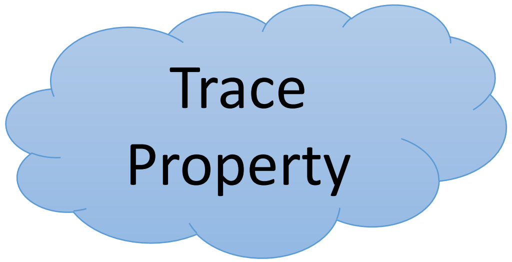
Trace Property

n = 4, f = 1

Common Core

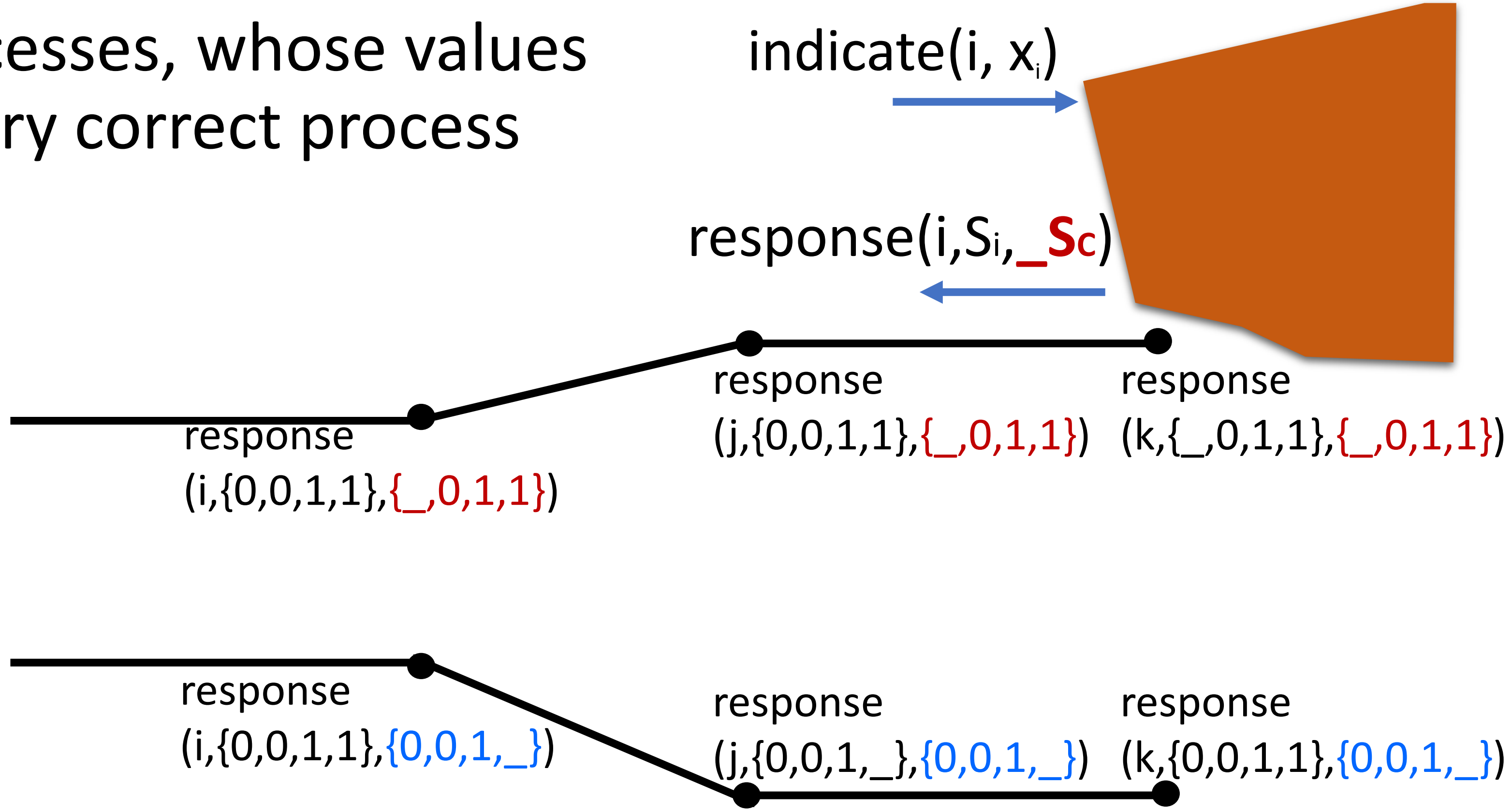
there is a set of n-f processes, whose values appear in the set of every correct process

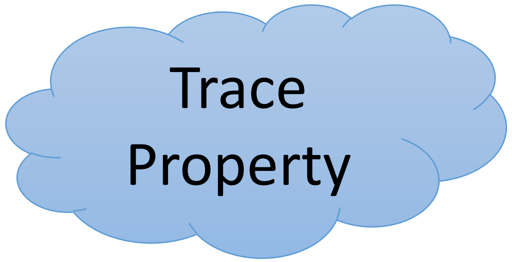




Common Core

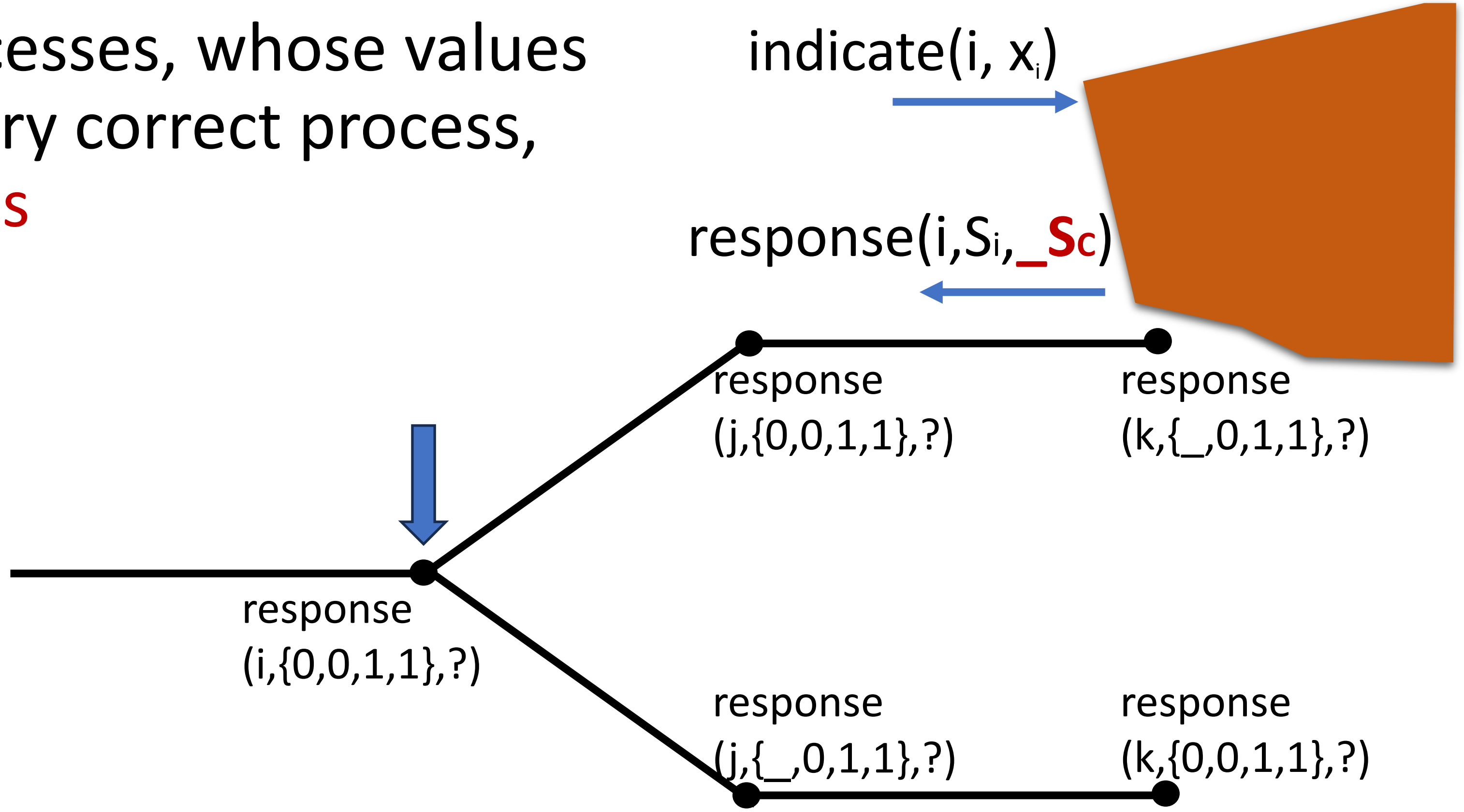
there is a set of n-f processes, whose values appear in the set of every correct process

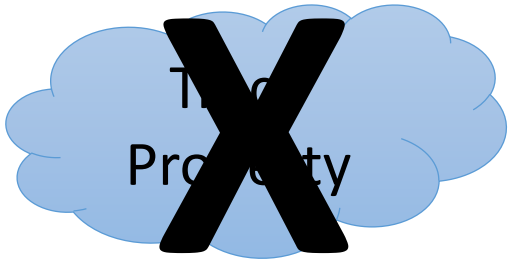




Binding Common Core

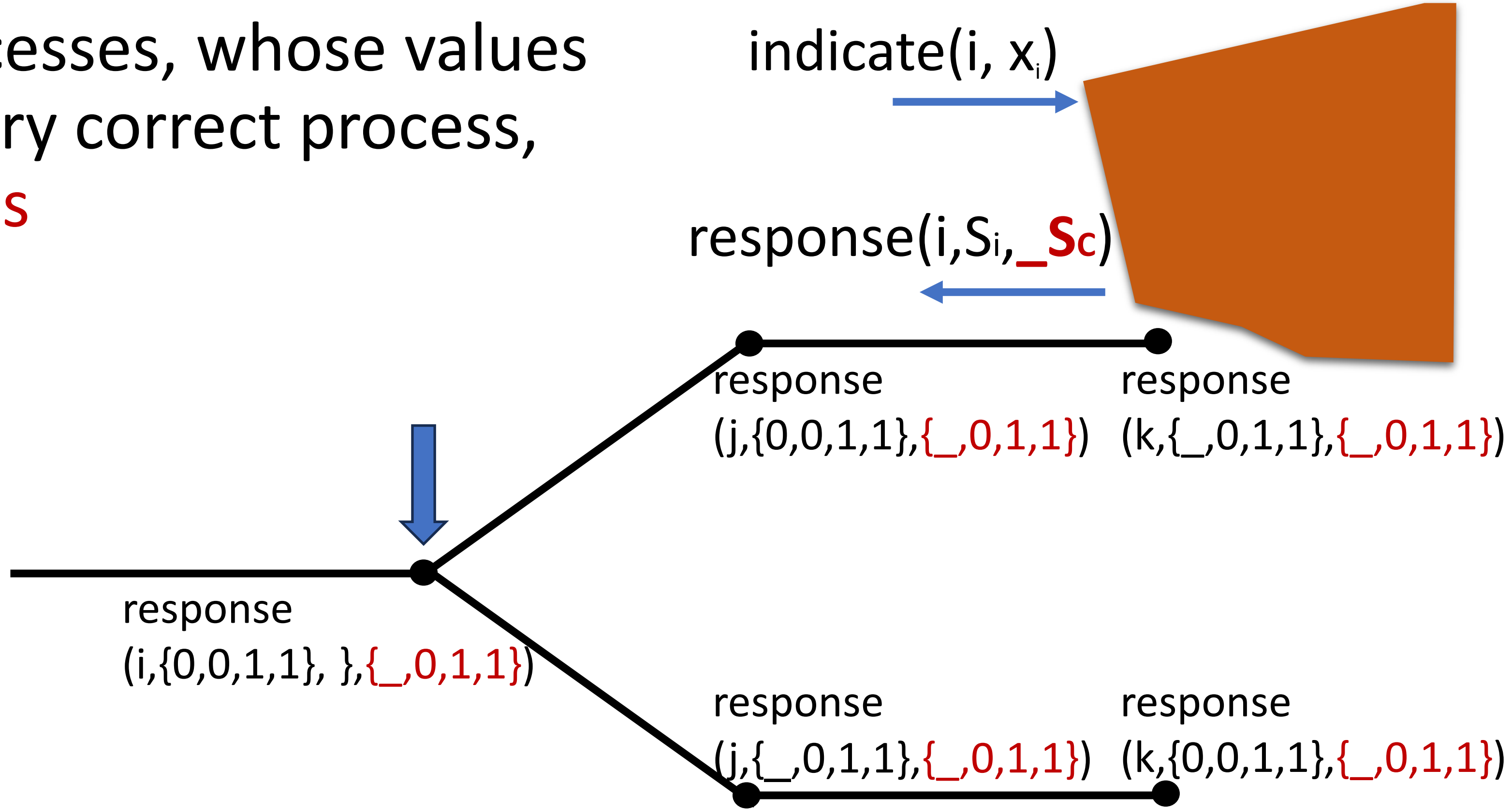
When the first correct process returns
there is a set of n-f processes, whose values appear in the set of every correct process,
in all possible extensions





Binding Common Core

When the first correct process returns
there is a set of n-f processes, whose values appear in the set of every correct process,
in all possible extensions



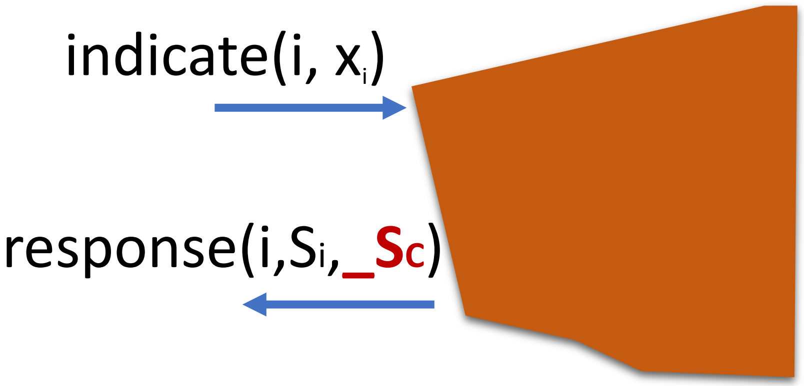
Hyper Property

Binding Gather and Strong Refinement

Implementation is binding if it is a
strong refinement of the gather module

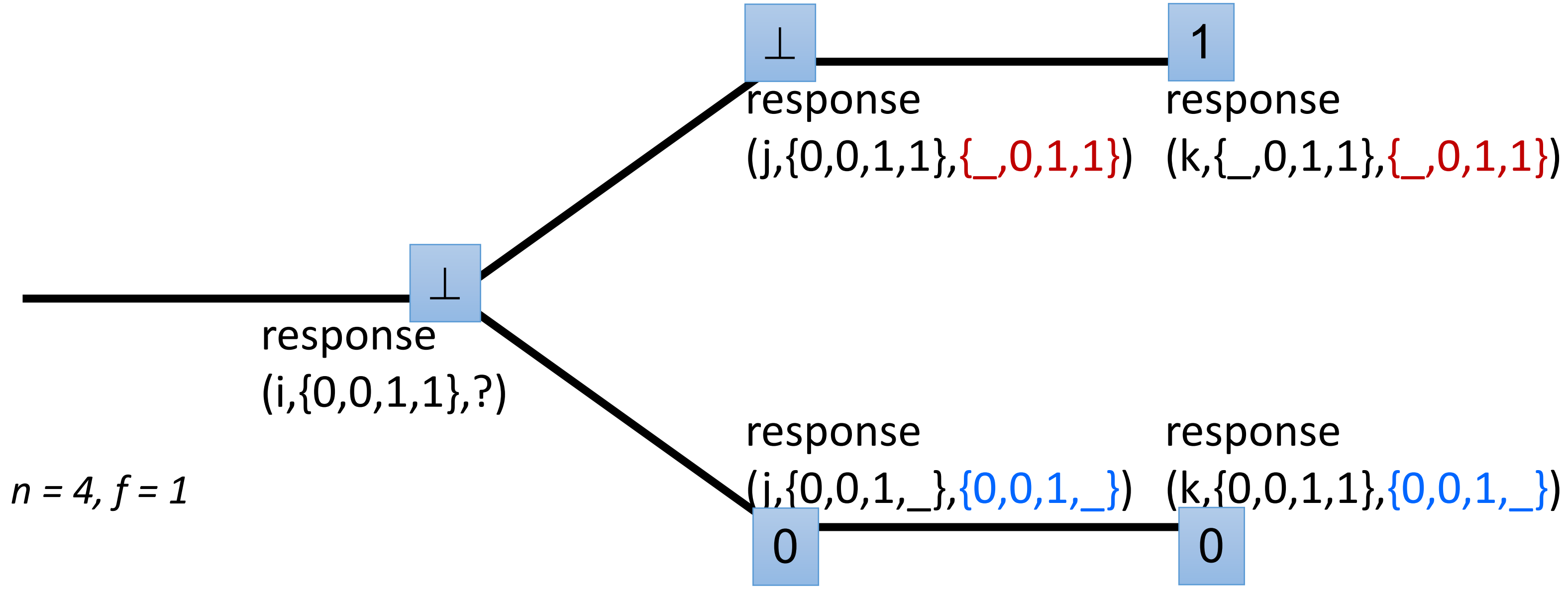
≡Forward Simulation

$\_S_c$ is a history variable



Crusader Agreement from Gather (code for p_i)

```
Sᵢ ← gather(xᵢ)
if Sᵢ contains ≥ |Sᵢ| - f copies of v
      then return v
else return ⊥
```



Crusader Agreement from Gather (key lemma)

```
Sᵢ ← gather(xᵢ)
if Sᵢ contains ≥ |Sᵢ| - f copies of v
      then return v
else return ⊥
```

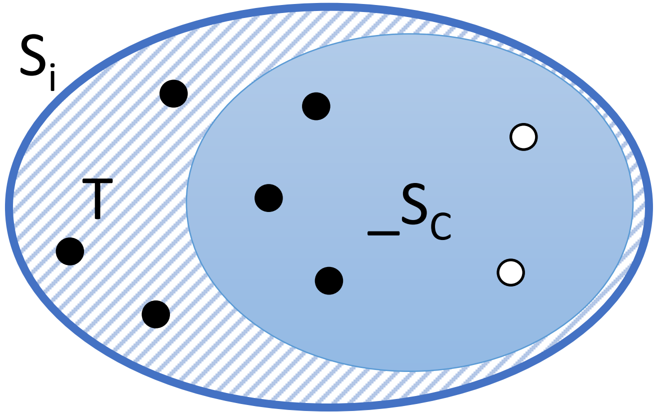
Lemma: If a non-⊥ value v is returned by a correct process, then v appears $\geq |\_S_C| - f$ times in the common core $\_S_C$

This suffices since $|\_S_C| = n - f$ and $n > 3f$

⇨ at most one value appears $|\_S_C| - f$ times in $\_S_C$

⇨ all correct processes that return a non-⊥ value return the same value

Crusader Agreement from Gather (key lemma)




```
Si ← gather(xi)
if Si contains ≥ |Si| – f copies of v
      then return v
else return ⊥
```

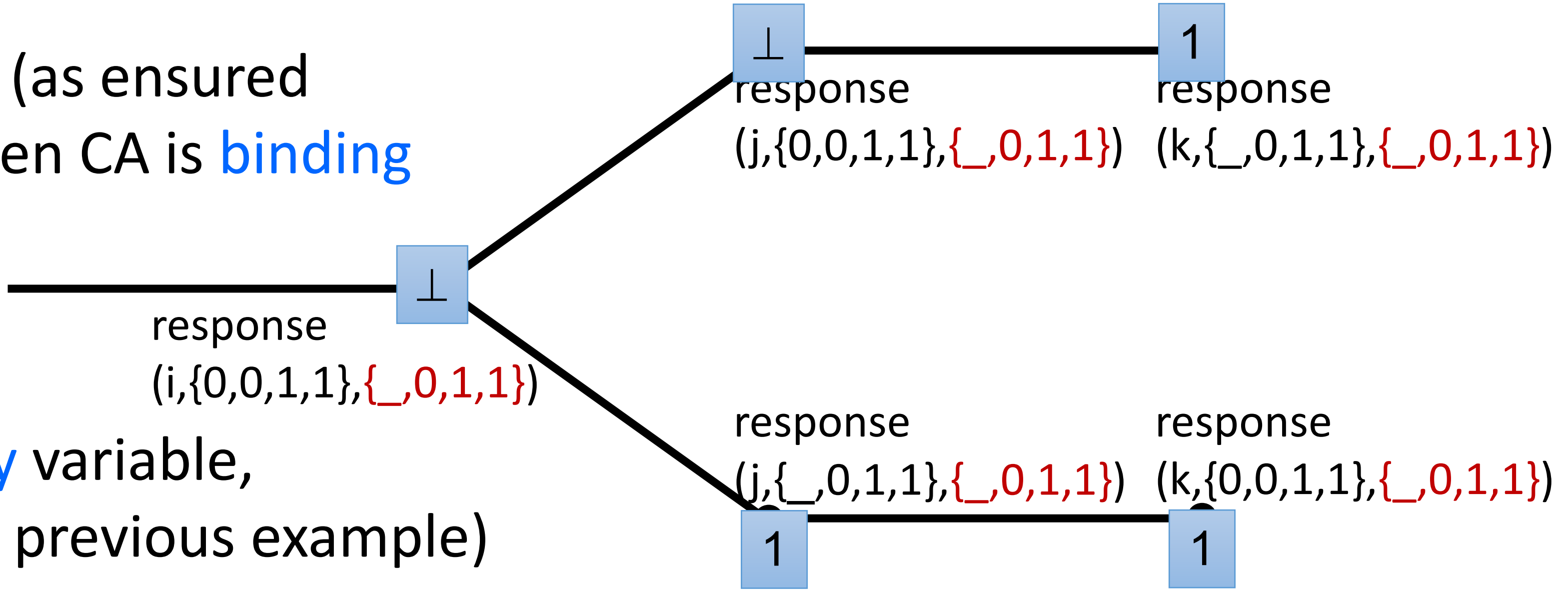
Lemma: If a non-⊥ value v is returned by a correct process, then v appears ≥ $|\_S_C|$-f times in the common core $\_S_C$

Proof of the lemma: correct process p_i returns v appearing $|S_i|$-f times in S_i
Let T be $S_i \setminus \_S_C$. $|T| = |S_i| - |\_S_C| \leq f$
Then the number of times v appears in $\_S_C$ is the number of times it appears in S_i minus the number of times it appears in T, which is $\geq \underbrace{|\cancel{S_i}| - f}\ - \underbrace{(|\cancel{S_i}| - |\_S_C|)}$

Crusader Agreement from Gather: Binding

```
Si ← gather(xi)
if Si contains ≥ |Si| – f copies of v
      then return v
else return ⊥
```

- If _S_C is a history variable (as ensured by strong refinement), then CA is binding



Otherwise, _S_C is a prophecy variable, and CA is not binding (recall previous example)

A Glimpse of What's Next

Commitment is a hyperproperty: a process commits to a value v (often drawn at random), unknown to other processes

In all extensions, only v can be revealed

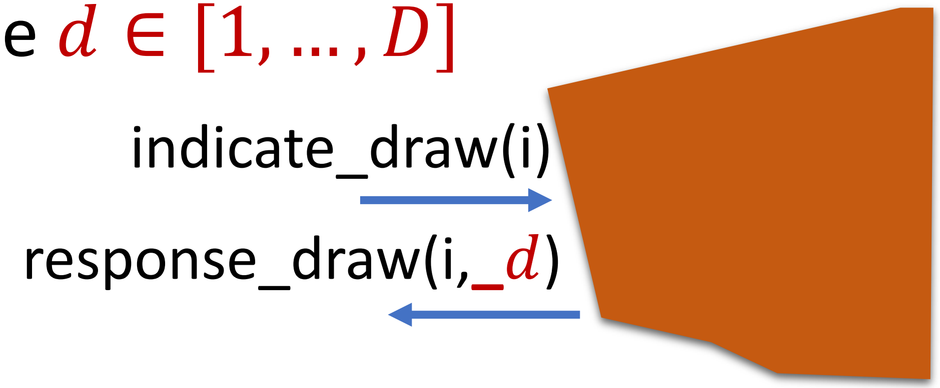
But what about random secret draw?

Implicitly used for a common coin in
[Canetti, Rabin 1993]
[Freitas, Kuznetsov, Tonkikh, DISC 2022]

Process p_i commits to a random value v, unknown to all processes

(Single) Random Secret Draw: Ghost Output

A single process commits to a random value $d \in [1, \dots, D]$



Also, ensure that d stays secret until revealed (using non-interference)

Wrap-Up

- Binding is a hyperproperty that commits the outputs across all extensions
- Ghost outputs can expose hidden commitments
- Strong refinement (≡ forward simulation, based only on the history) enforces binding
- Composition preserves binding:

  E.g., gather ⇨ crusader agreement
- Future research: commitment of probabilistic distributions

THANKS!