University of
Southampton

Generating SPARK from Event-B

Providing Fundamental Safety and Security

Asieh Salehi, Thai Son Hoang, Dana Dghaym, Michael Butler and **Colin Snook**

University of Southampton

Outline of talk

- Background
  - Event-B, CamilleX & Event-B notation extensions (including Records)
  - Spark
- Overview -
  - why generate SPARK from Event-B
  - From abstract record structures to SPARK
- SPARK code generation
  - Overview of translation rules (inc. records)
- SBB Electronic Voting Case study
  - using latest CamilleX and Records
  - SBB final refinement -> SPARK specifications

Background - Event-B

University of Southampton

- Discrete transition systems
  - Variables representing states
  - Guarded events representing transitions
  - Contexts: Static part of the models (carrier sets, constants, etc.)
  - Machines: Dynamic part of the models (variables, events, etc.)
- First-order logic with set theory

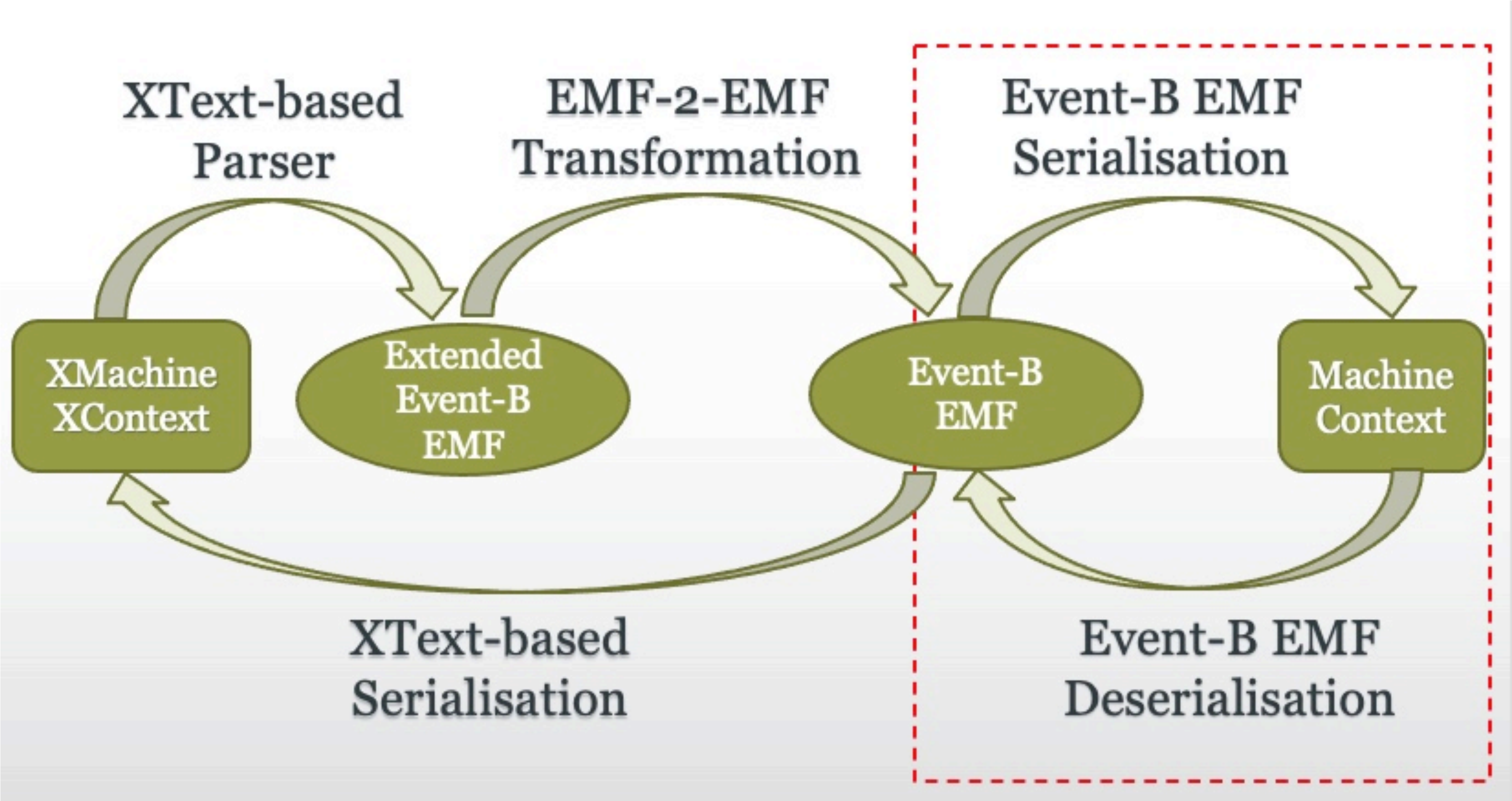
- Refinement
  - Start with a simple abstract model
  - Add detail and design in small steps

- Verification by automatic theorem provers
- Validation by model checking
- Model checker also useful for liveness and debugging

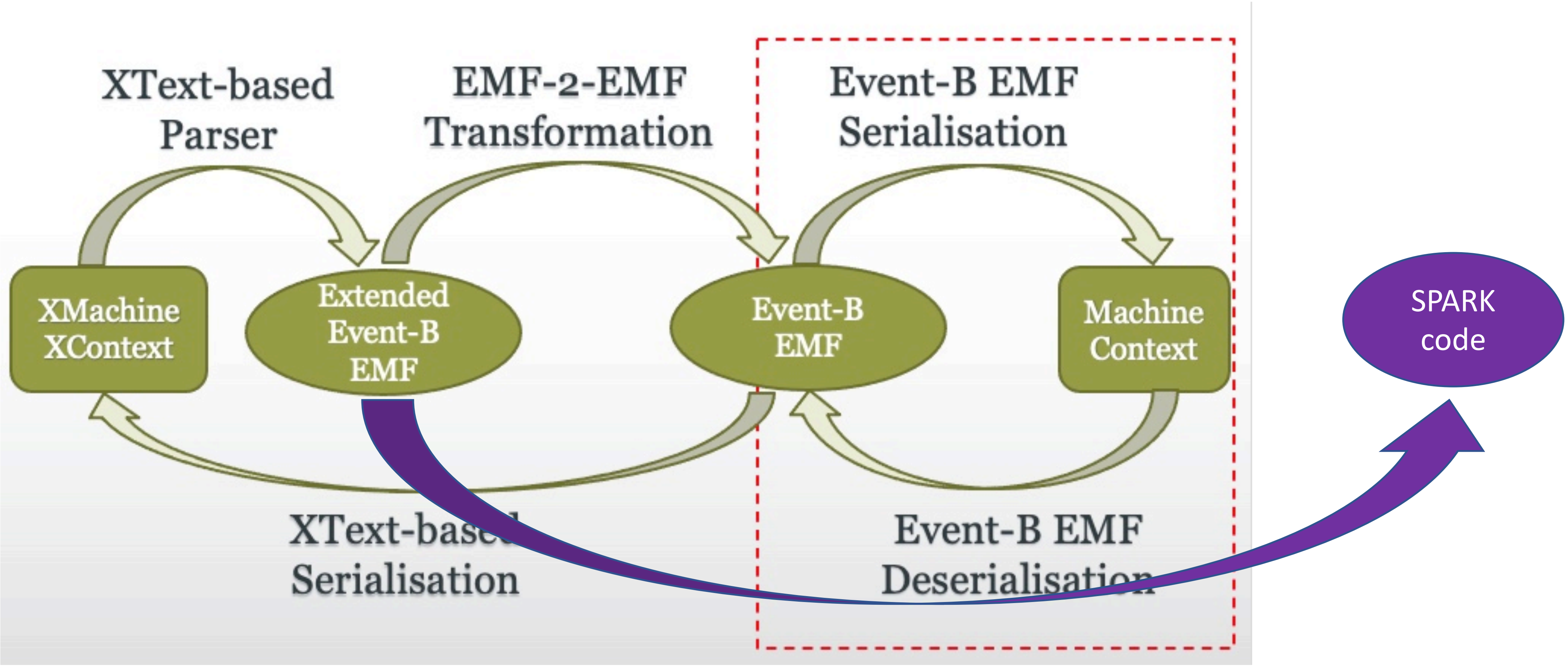
The Need for Textual Representation

- (True) Textual representation helps with teamworking
- Framework (e.g., XText) for developing IDE for DSLs.
- Design Principles:
  1. Reuse the existing Event-B tools of Rodin as much as possible.
  2. Support direct extension of the Event-B syntax to provide additional features.
  3. Provide compatibility with other kinds of ‘higher-level’ models that contribute to the overall model, e.g., UML-B diagrams.
- We make use of the Event-B EMF and EMF-2-EMF framework

The CamilleX Framework



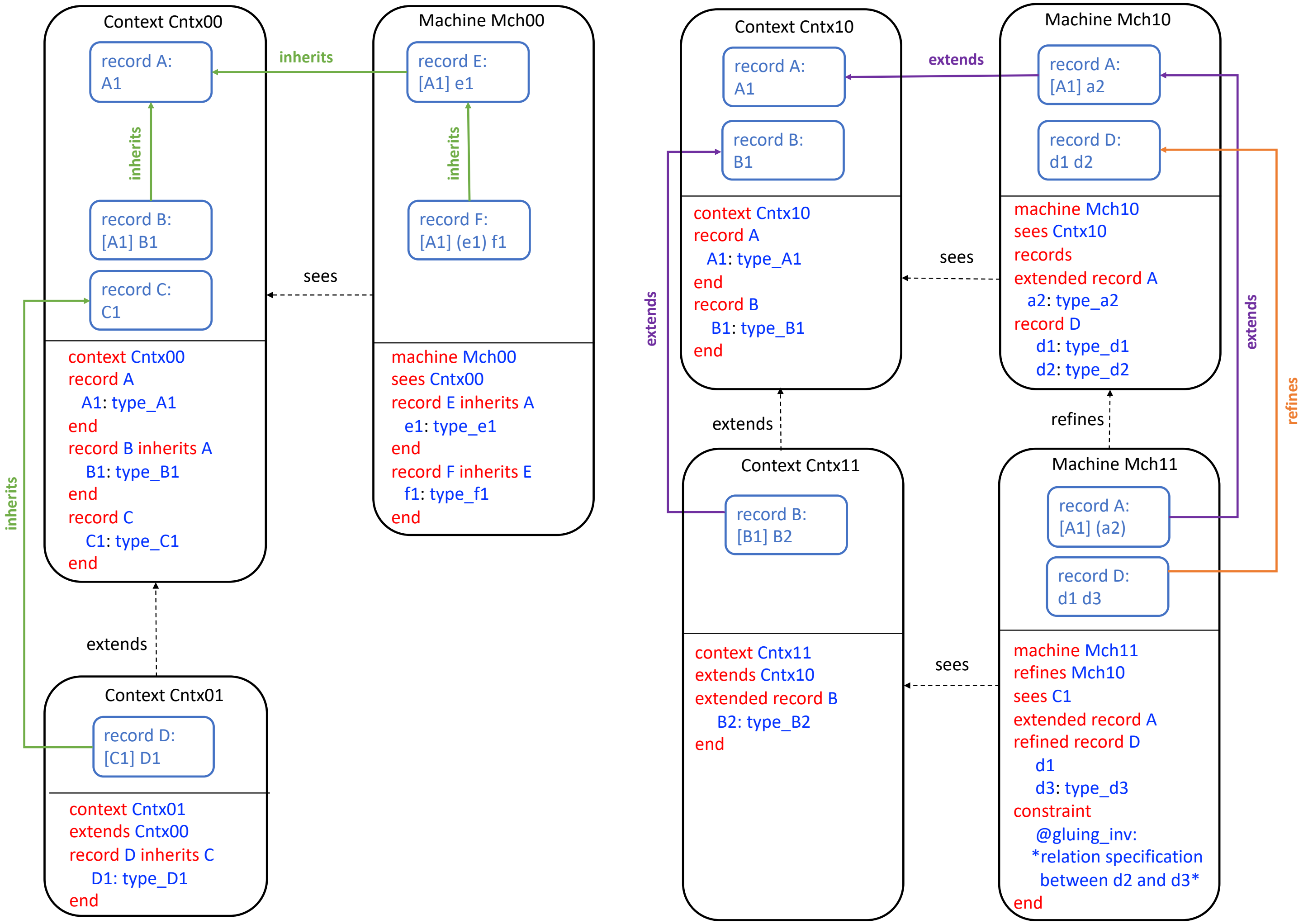
The CamilleX Framework



Records – an important extension to Event-B

- Inherits
  - Subtyping records
  - Implicit fields
- Refines
  - Replacing fields
  - Adding fields
- Extends
  - (Only) Adding fields
- Constraints
  - Properties of record instances

Ongoing research :
decomposition of records to prepare for translation to SPARK records



Background - SPARK

- Subset of Ada programming language
- Assertions, pre-conditions, post-conditions
- Targeted at highly reliable software
- Formal verification to prove the absence of runtime errors:
  - arithmetic overflow, buffer overflow and division-by-zero.
- Applied over many years
  - e.g. aircraft systems, control systems and rail systems.

Motivation

- Abstraction to isolate important properties
- Refinement to add detail and design
- Resulting in Validated and Verified models

- …But how can we ensure the code complies with the models

- Answer: generate Spark outline code
    - With pre/post-conditions that match events of the model
    - Assertions for run time checking

Steps : From abstract concept to Spark Implementation

- Abstract model of concept
- Refinements that introduce more detailed requirements
- Refinements that introduce design decisions
- Decomposition … *Controller* + Environment
- Refine *Controller* to prepare for code generation
- Generate SPARK from *Controller*

Overview of Event-B to SPARK Translation Rules

- Component Translation
  - (All) Context → specification package
    - Context extends → context packages use (and all extended context packages)
  - (Last) Refined Machine → specification and body packages
    - Machine sees context → use contexts packages (and all extended context packages)

```
machine mch_name sees ctx_name
```

```
with ctx_name; use ctx_name;
package mach_name
with SPARK_Mode => On
end mach_name;
```

```
package body mch_name with SPARK_Mode is ...
end mach_name;
```

Overview of Event-B to SPARK Translation Rules

- Constant Translation
  - Non function constants → constant, type depends on the axiom definitions

constants $const\_name$
axioms
$const\_name \in const\_type$
$const\_name = const\_value$

$$const\_name : \mathbf{constant}\ const\_type := const\_val;$$

  - Function type constant → function with return type depending on the range of the function and the function parameters are the domain of the Event-B function.

constants $cnst\_name$
axioms $cnst\_name \in dom \rightarrow ran$

$$\mathbf{function}\ cnst\_name\ (p\_dom :\ \mathbf{in}\ \ dom)\ \mathbf{return}\ ran;$$

Overview of Event-B to SPARK Translation Rules

University of Southampton

- Variable Translation
  - Variable → Global variable,
    - initialised according to the INITIALISATION event actions
- Record Translation
  - CamilleX Record → SPARK record
    - with all Event-B record fields (direct and implicit)

```
record rec_name
field_name : field_type
```

```
type rec_name is
field_name : field_type ;
end record;
```

Overview of Event-B to SPARK Translation Rules

University of Southampton

- Event Translation
  - Event → Procedure *(except Initialisation)*
  - Event Guard → Pre-condition
  - Event Action → Post-condition & Procedure body
  - Event Parameter → Procedure Parameter
    - (where output/input/in_out is deduced from guards and actions)
- Note that we have already proved invariants in the Event-B.. No need to translate invariants to SPARK
  - *Or is there! Some industrial partners have suggested that it may still be useful.. E.g. to catch problems caused by interrupts.*

Refinement of SBB example
From abstract concept to spark

- Ballot
- Paper (voter,vote)
- Paper (voter,vote,time)
- Paper (~~voter,vote~~,time,encrypted)
- Paper (~~voter,vote~~,time,encrypted, mac)
- Decompose -> smart ballot box + voters/attackers
- Refine SBB data towards arrays etc.
- Generate SPARK for SBB

Abstract notion of a ballot as a mapping from voters to vote
We only consider ideal situation of valid votes

Refinement of SBB example
From abstract concept to spark

- Ballot
- Paper (voter,vote)
- Paper (voter,vote,time)
- Paper (~~voter,vote~~,time,encrypted)
- Paper (~~voter,vote~~,time,encrypted, mac)
- Decompose -> smart ballot box + voters/attackers
- Refine SBB data towards arrays etc.
- Generate SPARK for SBB

Replace ballet with its physical representation :
Paper - fields for voter and vote

This introduces the possibility of invalid papers.. Copying faking etc.

Refinement of SBB example
From abstract concept to spark

- Ballot
- Paper (voter,vote)
- Paper (voter,vote,time)
- Paper (~~voter,vote~~,time,encrypted)
- Paper (~~voter,vote~~,time,encrypted, mac)
- Decompose -> smart ballot box + voters/attackers
- Refine SBB data towards arrays etc.
- Generate SPARK for SBB

Introduce new field : time

Voting papers can expire,
Reduces opportunity for validity threats

Refinement of SBB example
From abstract concept to spark

- Ballot
- Paper (voter,vote)
- Paper (voter,vote,time)
- Paper (~~voter,vote~~,time,encrypted)
- Paper (~~voter,vote~~,time,encrypted, mac)
- Decompose -> smart ballot box + voters/attackers
- Refine SBB data towards arrays etc.
- Generate SPARK for SBB

Refine voter,vote with encrypted.

Provide confidentiality

Refinement of SBB example
From abstract concept to spark

- Ballot
- Paper (voter,vote)
- Paper (voter,vote,time)
- Paper (~~voter,vote~~,time,encrypted)
- Paper (~~voter,vote~~,time,encrypted, mac)
- Decompose -> smart ballot box + voters/attackers
- Refine SBB data towards arrays etc.
- Generate SPARK for SBB

Introduce mac (algorithm for hashing)

Enables checking validity of vote..

E.g. if an attacker tries to alter the vote

Refinement of SBB example
From abstract concept to spark

- Ballot
- Paper (voter,vote)
- Paper (voter,vote,time)
- Paper (~~voter,vote~~,time,encrypted)
- Paper (~~voter,vote~~,time,encrypted, mac)
- Decompose -> smart ballot box + voters/attackers
- Refine SBB data towards arrays etc.
- Generate SPARK for SBB

- Event-B model is a closed system
  - Some parts of model are the controller
  - Others the environment being controlled

Future work - How to decompose records sets e.g. only cast_papers are in the SBB system

Refinement of SBB example
From abstract concept to spark

- *Data Refinement from abstract SET into Array*
  - *Array can be modelled as a Total function from 0..n to set*
- *Event-B records can have optional fields SPARK we can only use total functions -*
  - *Define a null value for optional field so that all records are total*

- Ballot
- Paper (voter,vote)
- Paper (voter,vote,time)
- Paper (~~voter,vote~~,time,encrypted)
- Paper (~~voter,vote~~,time,encrypted, mac)
- Decompose -> smart ballot box + voters/attackers
- Refine SBB data towards arrays etc.
- Generate SPARK for SBB

Refinement of SBB example
From abstract concept to spark

University of Southampton

- Ballot
- Paper (voter,vote)
- Paper (voter,vote,time)
- Paper (~~voter,vote~~,time,encrypted)
- Paper (~~voter,vote~~,time,encrypted, mac)
- Decompose -> smart ballot box + voters/attackers
- Refine SBB data towards arrays etc.
- Generate SPARK for SBB

Example: Application to Smart Ballot Box Model

University of Southampton

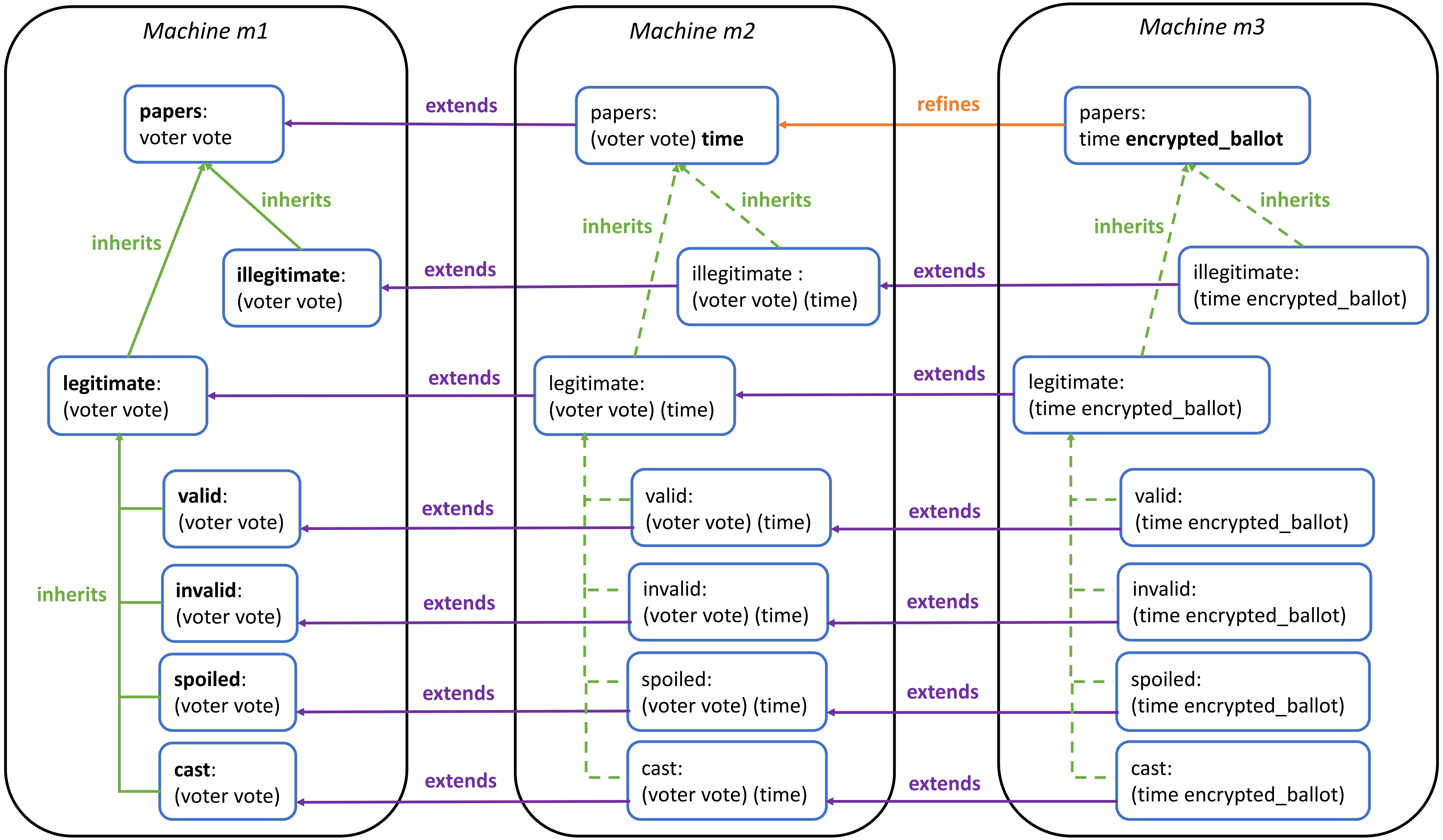
```
event cast_paper
refines cast_paper
any
paper
where
@grd1: paper ∈ BARCODE
@grd2: cast_count ∈ 0 .. max_votes −1
                    ....
then
@act1: cast_arr(cast_count)≔ paper
@act2: cast_count ≔ cast_count + 1
end
```

```
procedure cast(paper : in  barcode) with
Global => (Proof_In => ( spoiled_arr, curr_time, spoil_count),
In_Out => (cast_arr, cast_count)),
Pre => cast_count in 0 .. Max_Votes-1),
  and then not already_cast(paper)
                 ...
Post => already_cast(paper)
   and then cast_count = cast_count' old + 1);
```

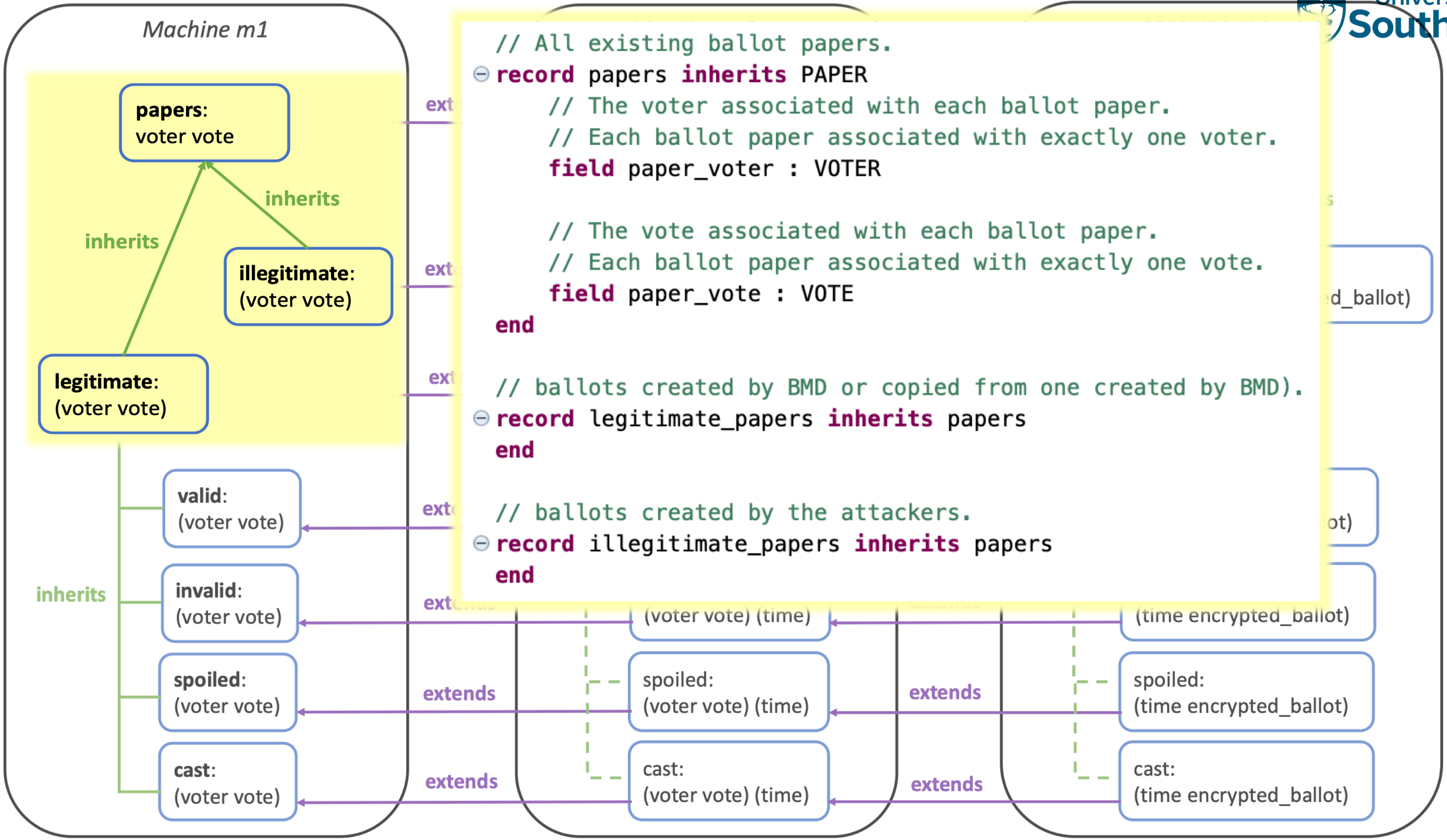
```
procedure cast(paper : in  barcode) is
 begin
   cast_arr(cast_count) := paper;
   cast_count := cast_count + 1;
end cast;
```

QUESTIONS?

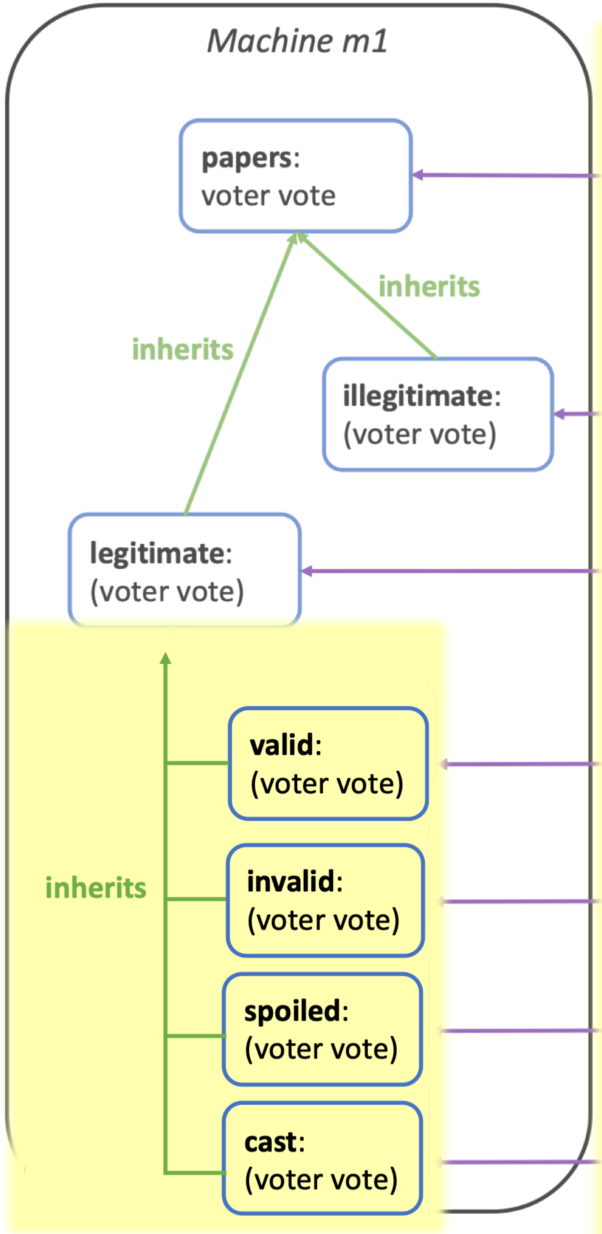
Case Study – SBB Electronic Voting



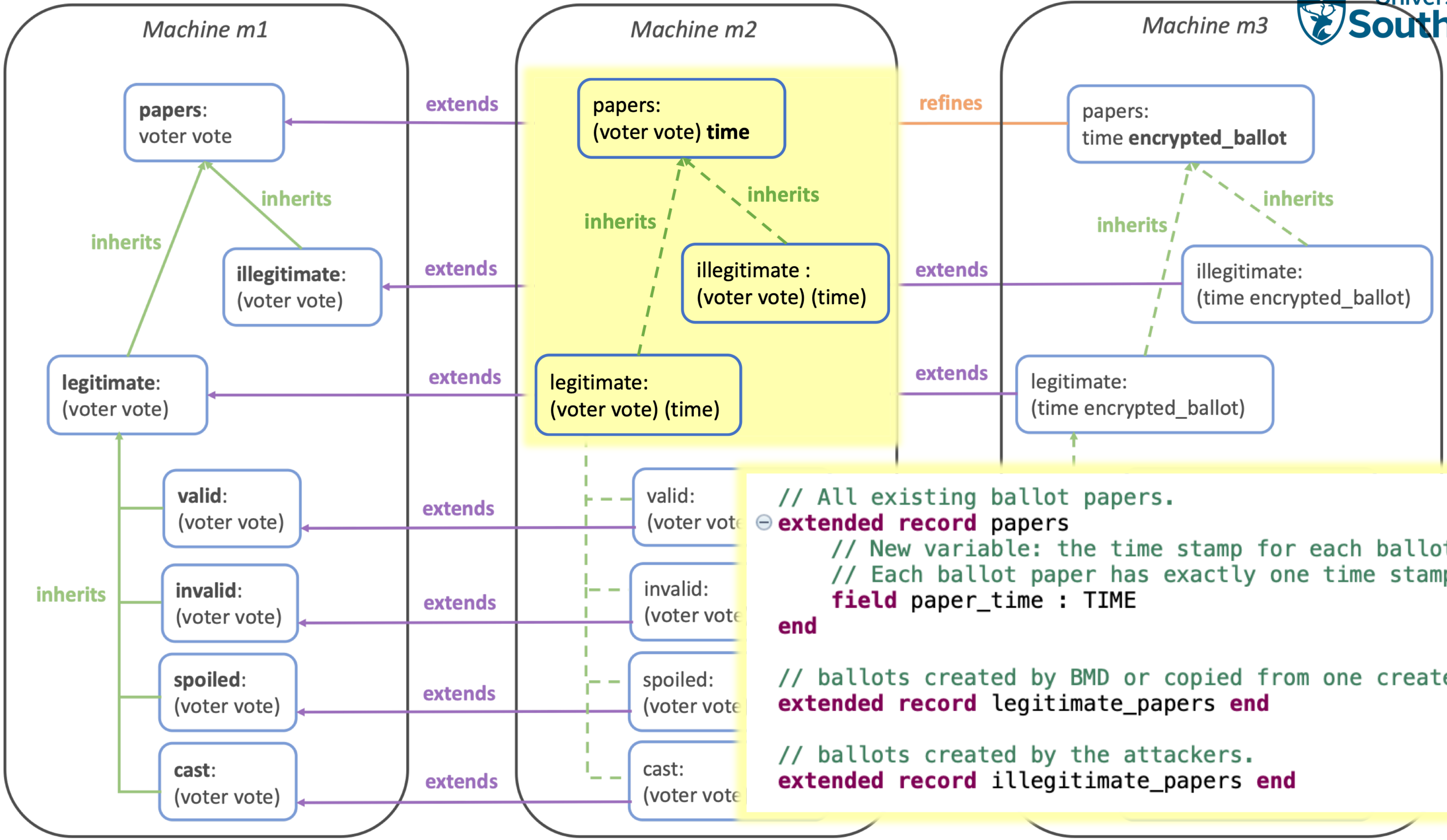
University of Southampton



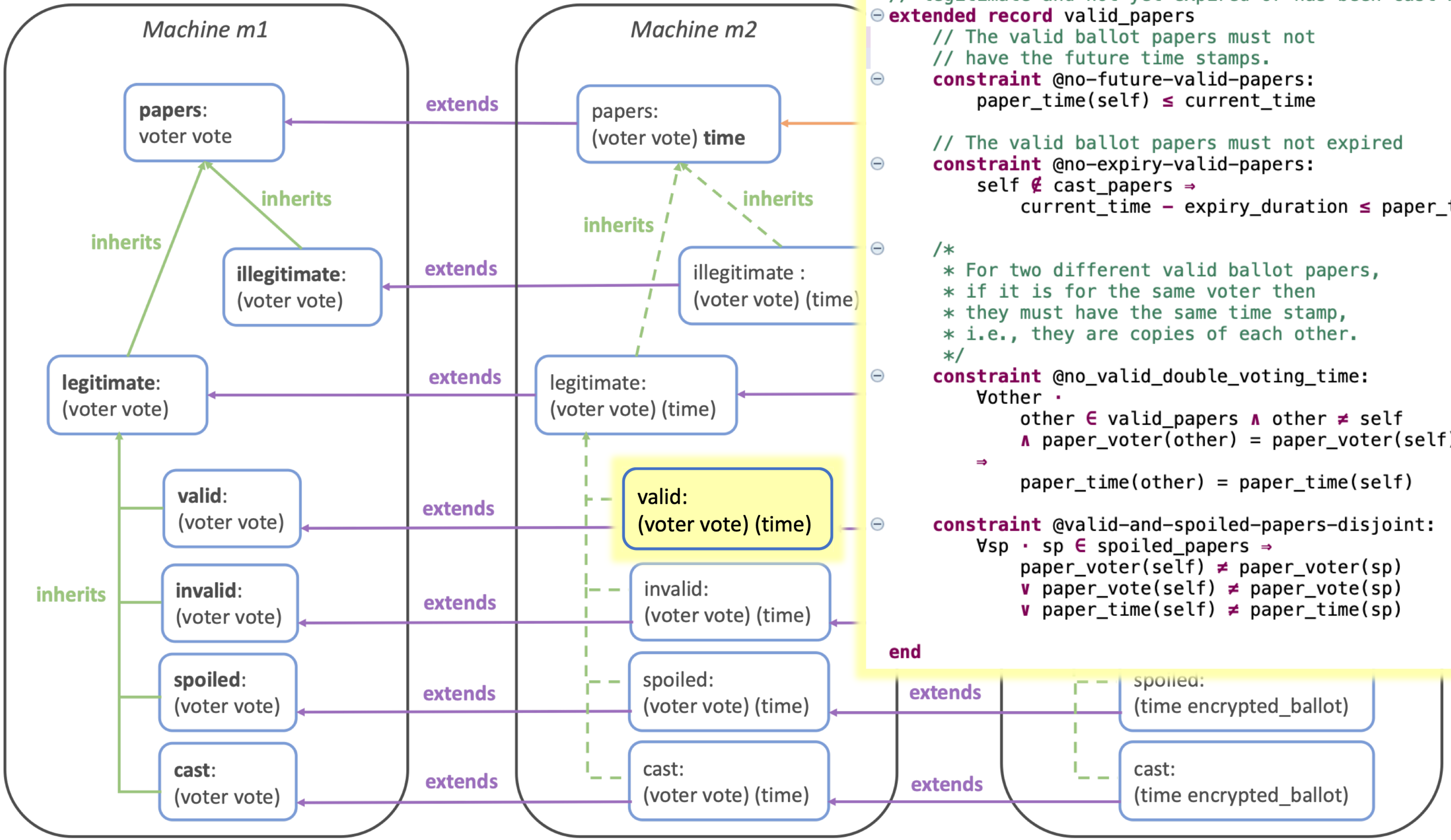

```
// All existing ballot papers.
record papers inherits PAPER
    // The voter associated with each ballot paper.
    // Each ballot paper associated with exactly one voter.
    field paper_voter : VOTER

    // The vote associated with each ballot paper.
    // Each ballot paper associated with exactly one vote.
    field paper_vote : VOTE
end

// ballots created by BMD or copied from one created by BMD).
record legitimate_papers inherits papers
end

// ballots created by the attackers.
record illegitimate_papers inherits papers
end
```



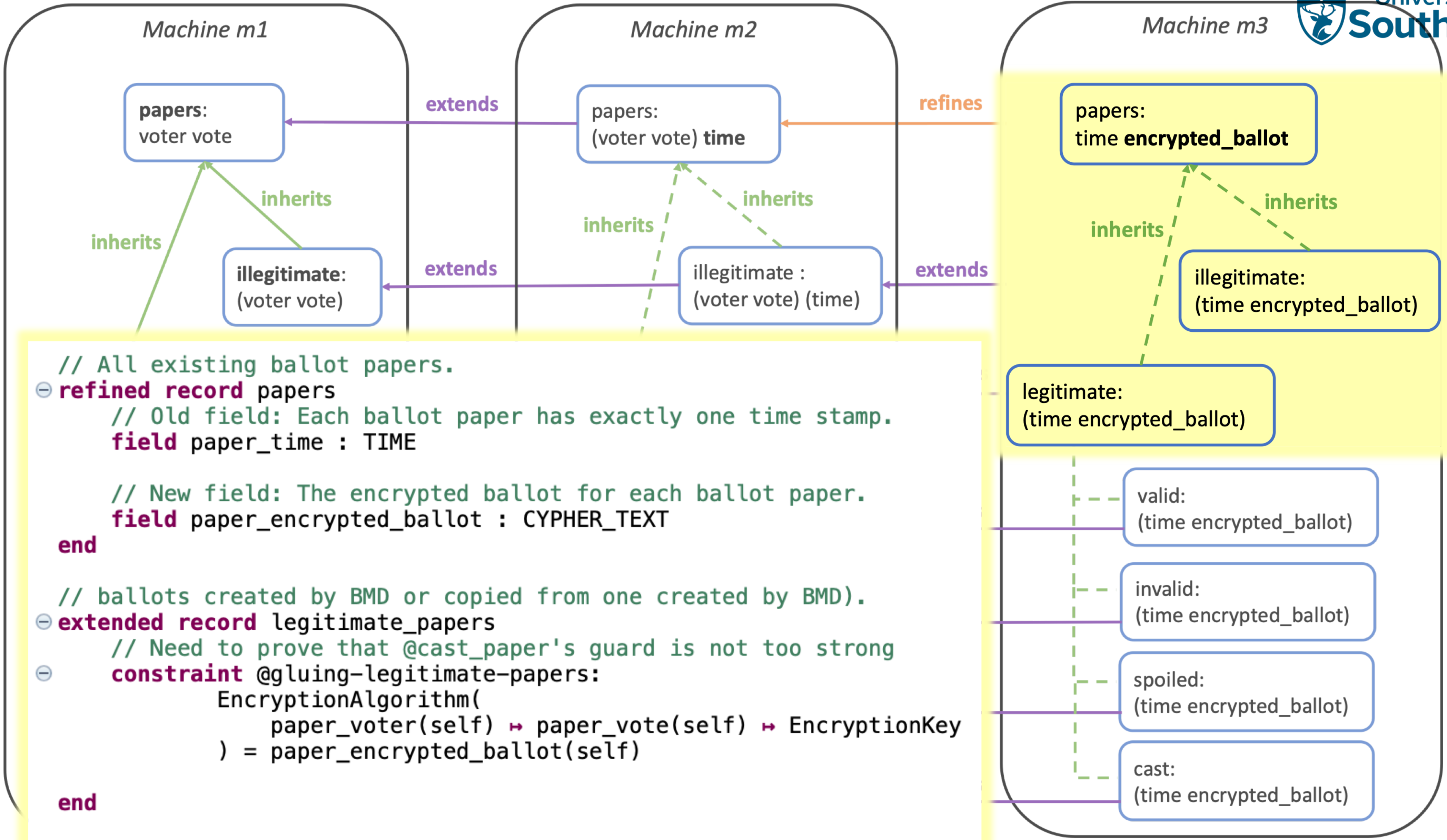

```
// The set of spoiled ballot paper.
record spoiled_papers inherits legitimate_papers
end

// legitimate and not yet expired or has been cast before expired or not spoiled
record valid_papers inherits legitimate_papers
    /*
     * For two different valid ballot papers, if it is for the same voter then
     * they must have the same vote, i.e., they are copies of each other.
     */
    constraint @no_valid_double_voting_vote:
        ∀other ·
            other ∈ valid_papers ∧ other ≠ self
            ∧ paper_voter(other) = paper_voter(self)
        ⇒
            paper_vote(other) = paper_vote(self)
end

// legitimate but becomes invalid since expired or a copy has been cast or
// spoiled.
record invalid_papers inherits legitimate_papers
end

// legitimate and already cast
record cast_papers inherits legitimate_papers
    /*
     * If a voter already cast a ballot paper, they cannot have any valid
     * ballot paper.
     */
    constraint @no_cast_double_voting_vote:
        paper_voter(self) ∉ paper_voter[valid_papers]
end
```




```
// All existing ballot papers.
extended record papers
    // New variable: the time stamp for each ballot paper
    // Each ballot paper has exactly one time stamp.
    field paper_time : TIME
end

// ballots created by BMD or copied from one created by BMD).
extended record legitimate_papers end

// ballots created by the attackers.
extended record illegitimate_papers end
```




```
// legitimate and not yet expired or has been cast before expired.
extended record valid_papers
    // The valid ballot papers must not
    // have the future time stamps.
    constraint @no-future-valid-papers:
        paper_time(self) ≤ current_time

    // The valid ballot papers must not expired
    constraint @no-expiry-valid-papers:
        self ∉ cast_papers ⇒
            current_time − expiry_duration ≤ paper_time(self)

    /*
     * For two different valid ballot papers,
     * if it is for the same voter then
     * they must have the same time stamp,
     * i.e., they are copies of each other.
     */
    constraint @no_valid_double_voting_time:
        ∀other ·
            other ∈ valid_papers ∧ other ≠ self
            ∧ paper_voter(other) = paper_voter(self)
        ⇒
            paper_time(other) = paper_time(self)

    constraint @valid-and-spoiled-papers-disjoint:
        ∀sp · sp ∈ spoiled_papers ⇒
            paper_voter(self) ≠ paper_voter(sp)
            ∨ paper_vote(self) ≠ paper_vote(sp)
            ∨ paper_time(self) ≠ paper_time(sp)

end
```




```
// All existing ballot papers.
refined record papers
    // Old field: Each ballot paper has exactly one time stamp.
    field paper_time : TIME

    // New field: The encrypted ballot for each ballot paper.
    field paper_encrypted_ballot : CYPHER_TEXT
end

// ballots created by BMD or copied from one created by BMD).
extended record legitimate_papers
    // Need to prove that @cast_paper's guard is not too strong
    constraint @gluing-legitimate-papers:
            EncryptionAlgorithm(
                paper_voter(self) ↦ paper_vote(self) ↦ EncryptionKey
            ) = paper_encrypted_ballot(self)

end

// ballots created by the attackers.
extended record illegitimate_papers end
```