GemClassifier, a formally developed smart card

Jean-Louis LANET,
Gemplus Research Lab

Baltimore, March the 6th

GEMPLUS

Agenda

- Motivations
- An experiment: the Java Card byte code verifier
- Analysing the metrics
- Conclusion

Foreword

Do we use formal methods in the development process ?

NO…

… But we use it in the certification process and in the research lab.

Is there any good reason to use formal methods ?

- In 1999 we identified three reasons:
  - reducing the cost of the test:
    - automatic test case generation,
    - but… tools do not merge behavior and data.
  - certifying at a high level smart card:
    - Multos proposed a ITSEC E6 certification,
    - but… the cost of the certification is too high.
  - mastering the complexity:
    - avoid or remove any vulnerability in the code,
    - this is where most benefits can be expected.

Missing items

- What is missing ?
  - prove that formal methods are technically usable in the smart card domain (highly constrained device),
  - demonstration that the overhead is acceptable,
- Provide evidence on a non trivial example,
- Collect metrics for managerial decisions,
- Adapt and/or improve the tools for our domain.

Gemplus experience

- Smart Card components
  - T=1 protocol (1998), T=CL (2000), TCP-IP (2002)
  - Backup (1999)
  - Garbage Collector (2000)
- Java Card
  - Code optimisations correctness (1998)
  - Firewall modelling (1999)
  - Byte Code interpreter (1999)
  - Information flow analysis (2000)

Gemplus experience

- Byte code verification
  - Proof of classical verification algorithm (2000)
  - FACADE lightweight verification (2000)
  - ...

Formal Methods and Smart Cards

- The Multos e-purse and OS

  "…formal model activity was never on the critical path… and never cause a delay in development" … "It's a balance between model clarity and ease of proof."

- dJVM by Cohen

  Very important but unfinished work, Properties to prove: type safe execution

- Java Card Verifier using a Model Checker [Posegga],

  Security properties as temporal formulae are verified with the model checker

Formal Methods and Smart Cards

- And more recent works... :
  - Proof of a verifier on the F&M subset of the byte code using Coq [Bertot], [Jakubietz] using Isabelle [Nipkow],
  - The Loop project at Niemegen University [Poll],
  - Formalisation of a byte code verifier at Kestrel Institute [Qian]...

...and others...

How to introduce FM in product development ?

- Matisse project
  - gathering evidence of the use of formal methods
  - develop methodology and tools.

- We claim that formal methods can be used in developing smart cards in such a way that gains in quality come at acceptable cost.

Evaluation plan

- Hypothesis to be tested:
  - H1: improvement of the quality
    - need a sister development to compare the effectiveness of the formal development
    - collect the errors detected in both development and the cost to correct it

Evaluation plan

- Hypothesis to be tested:
  - H1: improvement of the quality
  - H2: cost overhead is acceptable
    - collect the time needed to achieve both development paying attention to the different phases of the development.

Evaluation plan

- Hypothesis to be tested:
  - H1: improvement of the quality
  - H2: cost overhead is acceptable
  - H3: non specialist engineer can use formal methods
    - will help in the definition of an adequate team
    - evaluate the effectiveness of subcontracting the proof process

Evaluation plan

- Hypothesis to be tested:
  - H1: improvement of the quality
  - H2: cost overhead is acceptable
  - H3: non specialist engineer can use formal methods
  - H4: it will facilitate regulatory requirements
    - from EAL-5 level of the common criteria
    - even if not the same development, the know-how acquired in developing a product could help for modeling the security functions

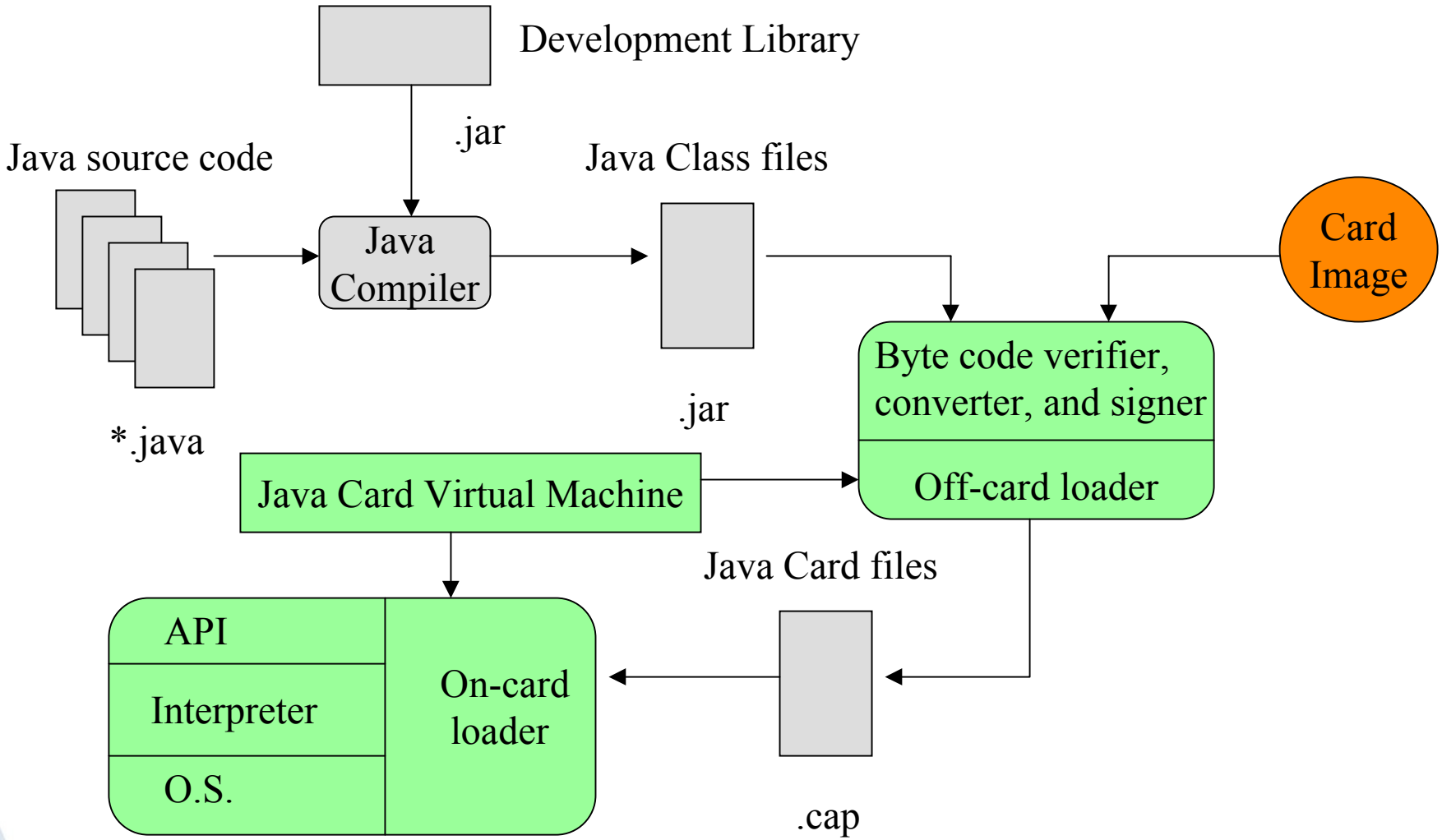
Evaluation plan

- Hypothesis to be tested:
  - H1: improvement of the quality
  - H2: cost overhead is acceptable
  - H3: non specialist engineer can use formal methods
  - H4: it will facilitate regulatory requirements
  - H5: automatically generated code fits SC constraints
    - time will be saved if the code is generated (no unitary test),
    - if the code is automatically generated no rework must be planned to optimize it,
      - the memory footprint overhead is acceptable,
      - the execution time is not slowed.

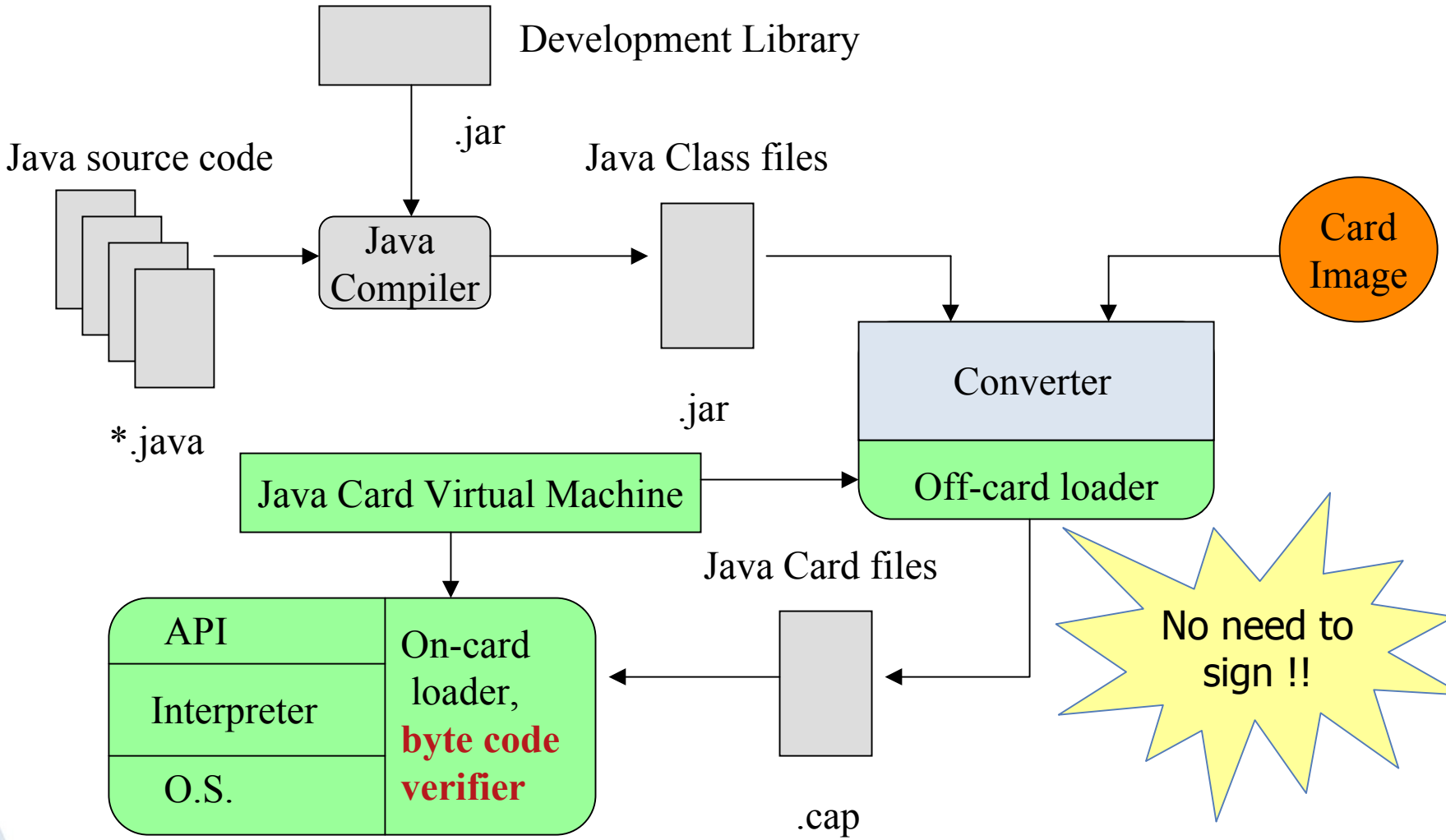
A non-trivial example ?

- Develop formally a significant part of a smart card:
  - operating system or Java Card Virtual Machine,
  - from specification to the code,
  - embed the code INTO a card.
- Define guidelines or methodology for the specification,
- Define rules or tactics to improve the proof process,
- Develop a code translator that complies with the smart card constraints.

Java Card Architecture



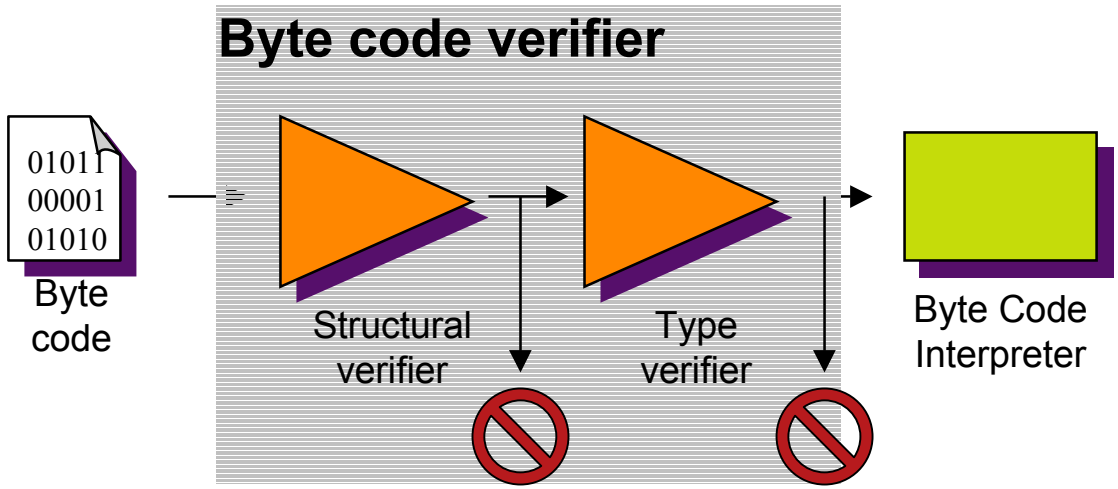
Java Card Architecture



The case study

- Embedding the byte code verifier is a real challenge:
  - verification during load phase only,
  - the verifier is a key point of the security architecture,
  - we need the proof of correct implementation using a formal method.
- For the purpose of the evaluation we have developed two similar algorithms:
  - a PCC-based type verifier,
  - a standalone type verifier.

Byte code verification overview

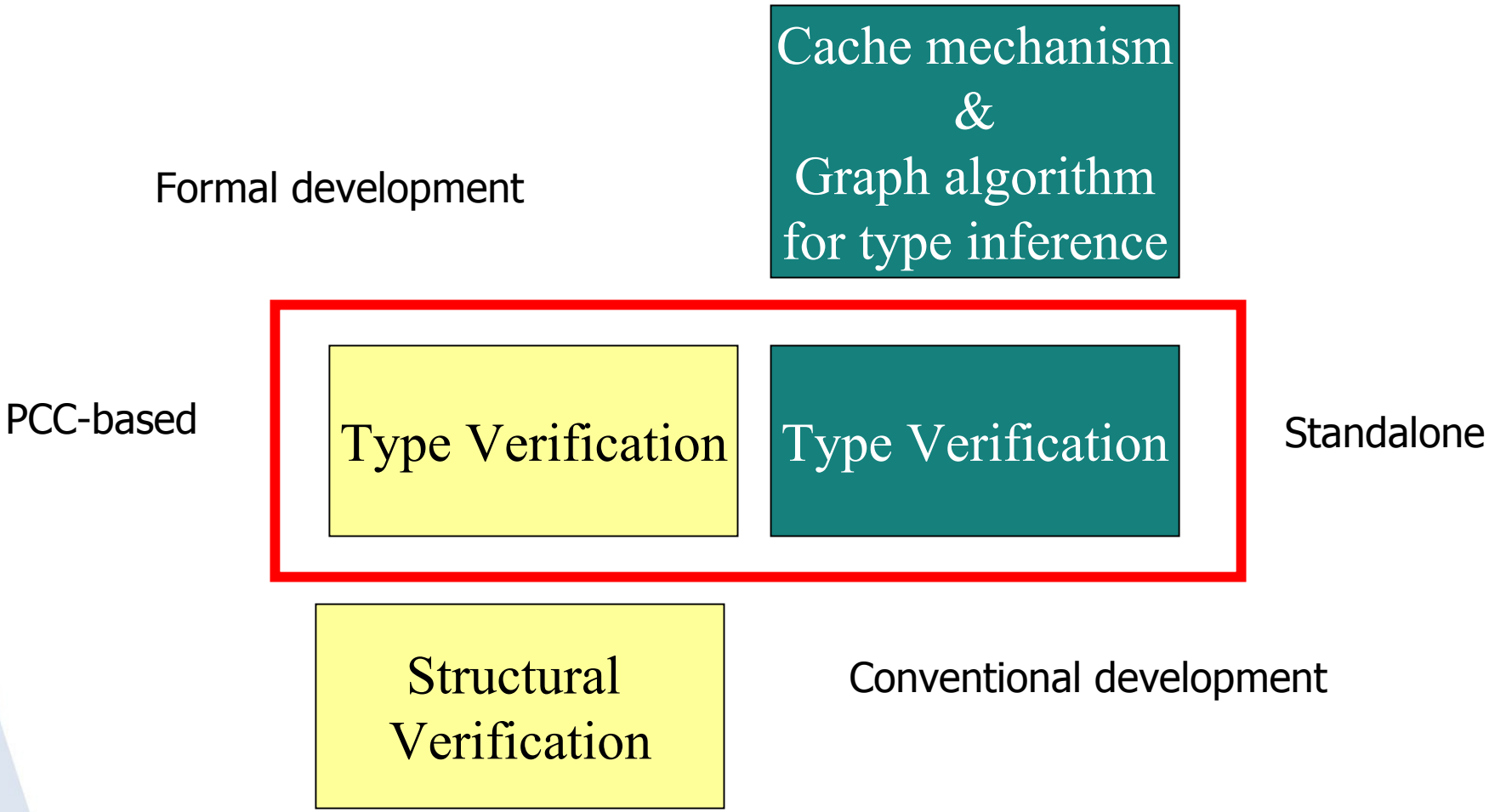


- Byte code verification
  - performs tests ensuring that loaded applets conform to Java typing rules
  - separated in structural and type verification

Byte code verification overview

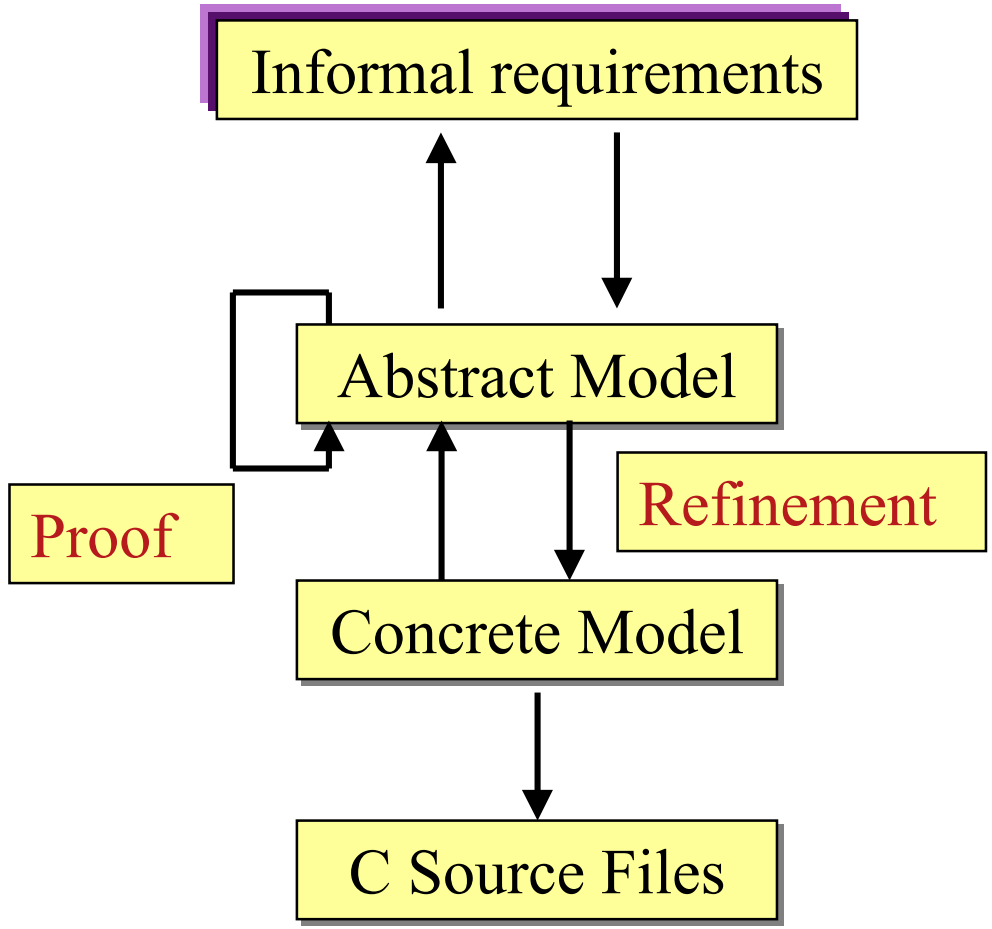
- Check file structure
  - ensure that the file can be correctly parsed by the remaining parts of the virtual machine
  - check internal consistency of components as well as shared information
- Check type correctness
  - consists in a set of tests that must be performed for each instruction of the downloaded applet
    - no forged pointers or illegal data conversions
    - no stack overflow/underflow
    - *etc*…

The common part

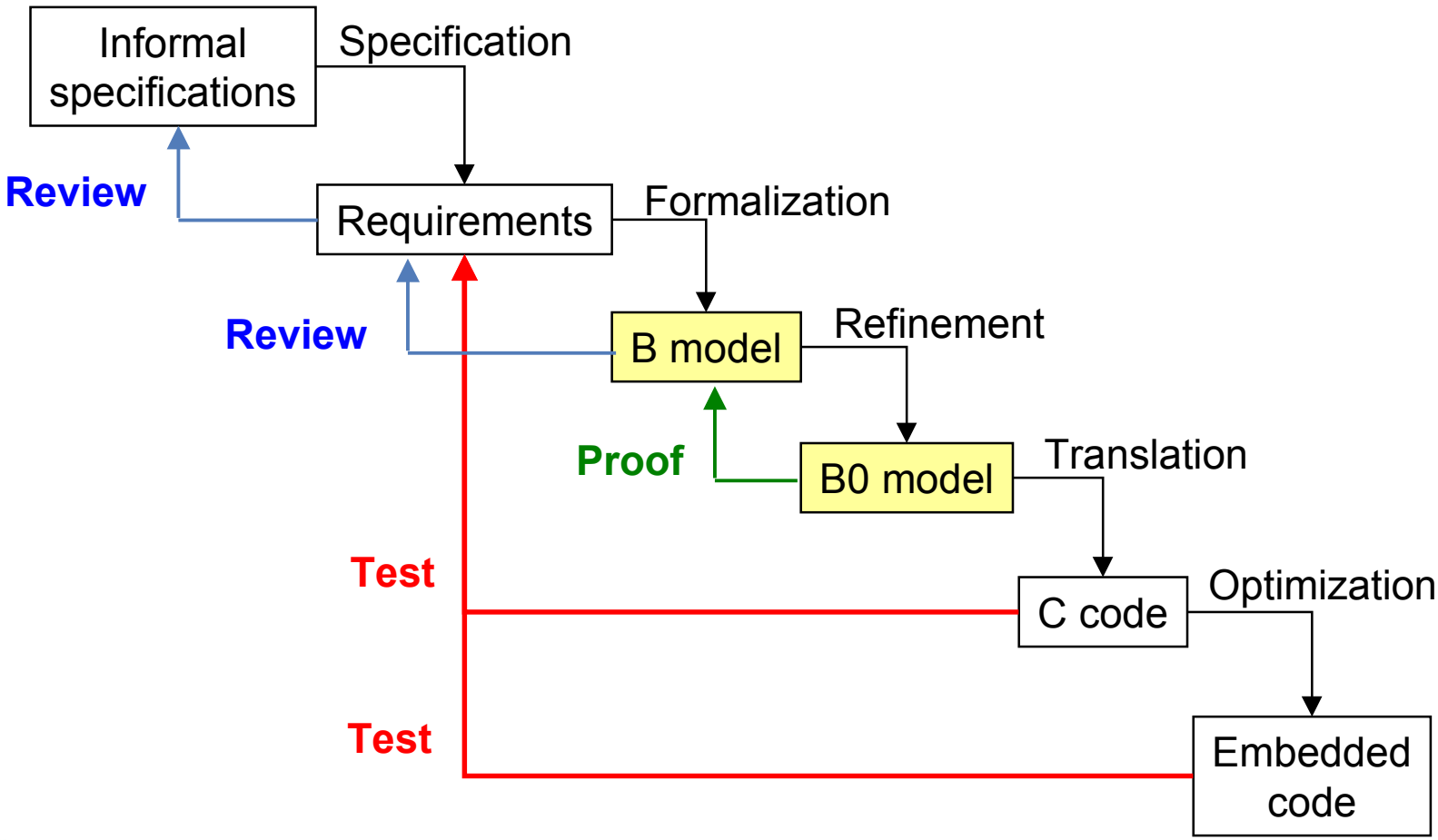


Formal development

- Development with the B formal method
  - definition of the architecture,
  - formalization of the specification in an abstract model,
  - refinement of the abstract model in a concrete model,
  - automatic code generation.



At the boundaries...



Bias in the evaluation

- We developed a prototype not a product,
  - not the same qualification process,
  - no field return,
- Skills of the teams,
  - different skills in type verification, Java Card…,
  - experts in B and C,
- Teams were not physically separated,
  - development, test, integration, review,
  - flow of knowledge,
  - they don’t start exactly at the same time,
- Algorithms are different…
  - jsr, ret are pre-processed

H1: improvement in the quality

- Number of discovered faults

	Formal	Conventional
Review	13	24
Proof	29	NA
Test	32	71
Total	74	95

Faults discovered by test

- In the 32 faults of the formal development, we have to retrieve the errors due to:
  - the translator ……1
  - external tools……8
  - structural verification….9
- 14 errors remain related to the translation from informal to formal specification

- In the 71 errors of the conventional development, two were linked with unification algorithm which is not included in the common part.

Real amount of discovered faults

	Formal	Conventional
Review	13	24
Proof	29	NA
Test	14	69
Total	56	93

Zero default is unreachable !!!

H2: cost overhead is acceptable

- Development duration (in weeks)

	Formal	Conventional
Java understanding	Included in the next phase	4
Development	12	8
Proof	6	NA
Testing	1	3
Integration	1	2
Total	20 (weeks)	17 (weeks)

H3: non specialist can use FM

- The first part of the hypothesis cannot be validated.
- The second part of the hypothesis stated that subcontracting the proof process could drastically reduce the cost.
  - the sub-hypothesis is under investigation. The last 125 lemmas have been provided to Clearsy.

H4: it will facilitate regulatory requirements

- This hypothesis has not been evaluated.

H5: automatically generated code fits the SC constraints

- Memory footprint:

	Formal	Conventional
Type ROM size (kb)	18	16
Structural ROM size (kb)	24	NYI
RAM (byte)	140	128-756*
Applet code overhead (%)	10-20	0

* RAM usage for this verifier is adjustable

H5: automatically generated code fits the SC constraints

- Execution time (ms):

	Formal (6464)	Formal (3232)	Conventional (3232)
Wallet	811	411	318
Utils	2794	1312	1463
PacapInt	241	80	61
TicTacToe	3555	1232	1102

With the 3232, there is a direct access to the memory, while with the 6464, the B model needs a addition to each memory access

Evaluation plan…

Formal methods can be used in developing smart cards in such a way that gains in quality come at acceptable cost.

- **By validating H1, H2 and H5 we can assess that this aim is validated,**
- We cannot draw any conclusion on how a formal development can help to fulfill regulatory requirements.

And what's about the methodology ?

- Easy integration of proved code/non proved code (C code),
- Efficient use of dedicated rules to simplify the proof process,
- **Warning: we did not prove their correctness**
  - need some more weeks to demonstrate them,
  - by the rule prover tool, by hand, with another prover (Coq)...
- We still need to improve the code generation (currently too basic).

...but

- Software architecture constrained by the proof process
  - stronger architectural constraints than classical programming languages
  - modelling choices impact complexity of the proofs
    - requires taking the proof into account early
    - proof can require the specification and/or implementation rewriting
  - difficult to define an abstract model suitable for proving both implementation and specification
- Limit of the tool (wrong lemma, CPU usage,...).

Passing over these limitations

- This experience has demonstrated that the constitution of an expert team is the way to reduce costs:
  - capitalise experience
    - Architecture
    - Proof
  - trains and supports engineers

Formal Methods and Smart Cards

- Smart card is not a specific domain,
  - we obtain the same conclusion as in other domains,
    - overhead,
    - error in the formalization process.
  - tools used for railways run well for SC,
  - we have produced a SMART CARD.

GemClassifier

- A real technological challenge:
  - 2/3 years ago this features was considered as impossible to implement,
  - formal methods are affordable in the smart card domain.
- GemClassifier a technology breakthrough for the Java Card deployment.