June 2013

CLEARSY
SYSTEM ENGINEERING

New York Metro Flushing line

System level formal verification

Authors:
Denis SABATIER
Lilian BURDY

Paris
Lyon
Aix

WWW.CLEARSY.COM

What is “system level formal verification”?

- This is demonstrating wanted properties using only well defined rules and assumptions
  - **System level**: because subparts are represented by properties taken as assumptions
  - **Formal**: because the reasoning from those subpart properties to wanted properties shall use only defined mathematical rules
  - **Verification**: *building the system right* (validation is more a human judgment: *building the right system*)

Subpart: existing sub-system, example relays → Assumed properties

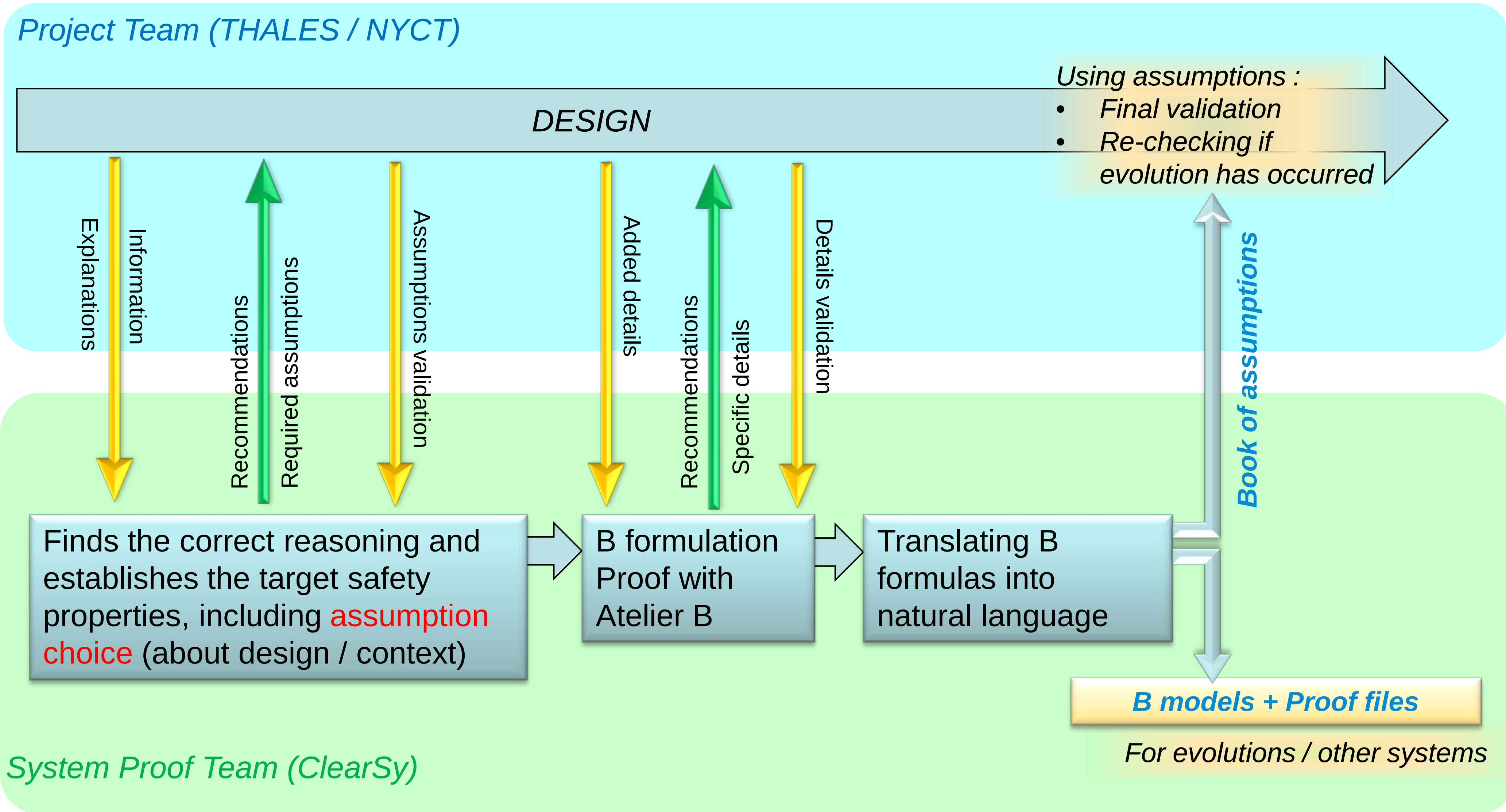
Subpart: on-board computer & software → Assumed properties

Subpart: train rail mechanical behavior → Assumed properties

And other subparts…

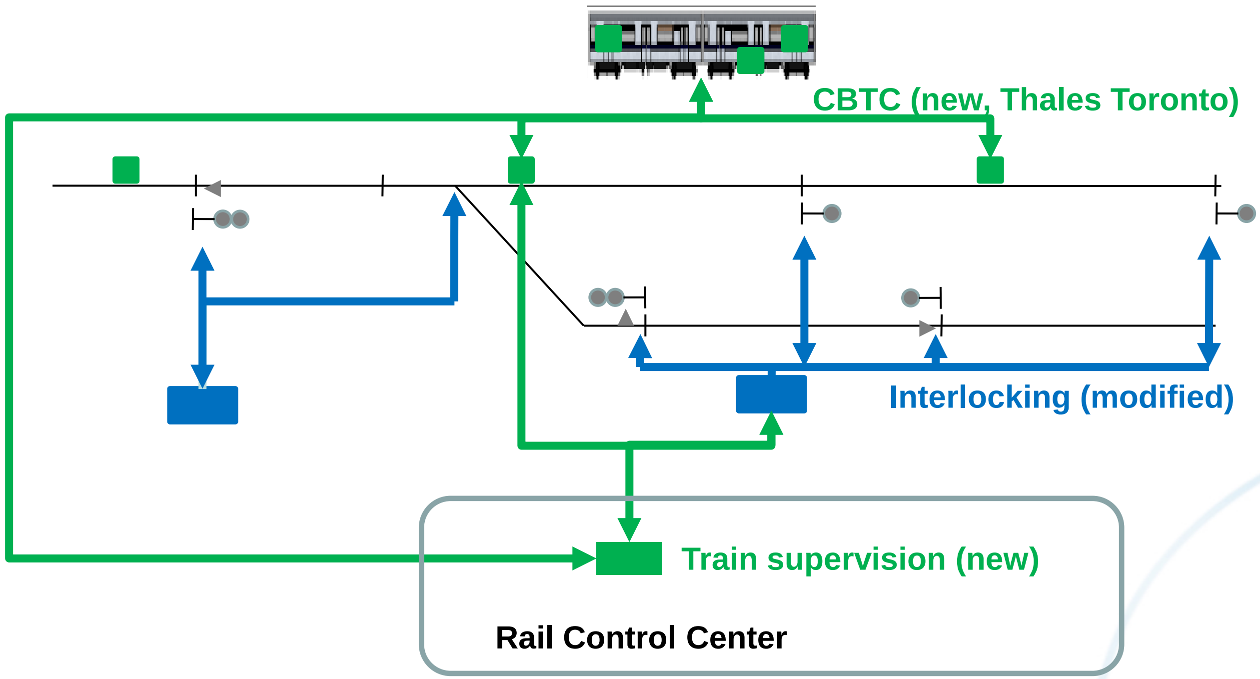
→ Obtaining wanted properties from assumed properties with mathematical formulation and application of known rules only → Wanted properties: no collision, no derailments, no overspeeding

System level formal verification: process for the Flushing project

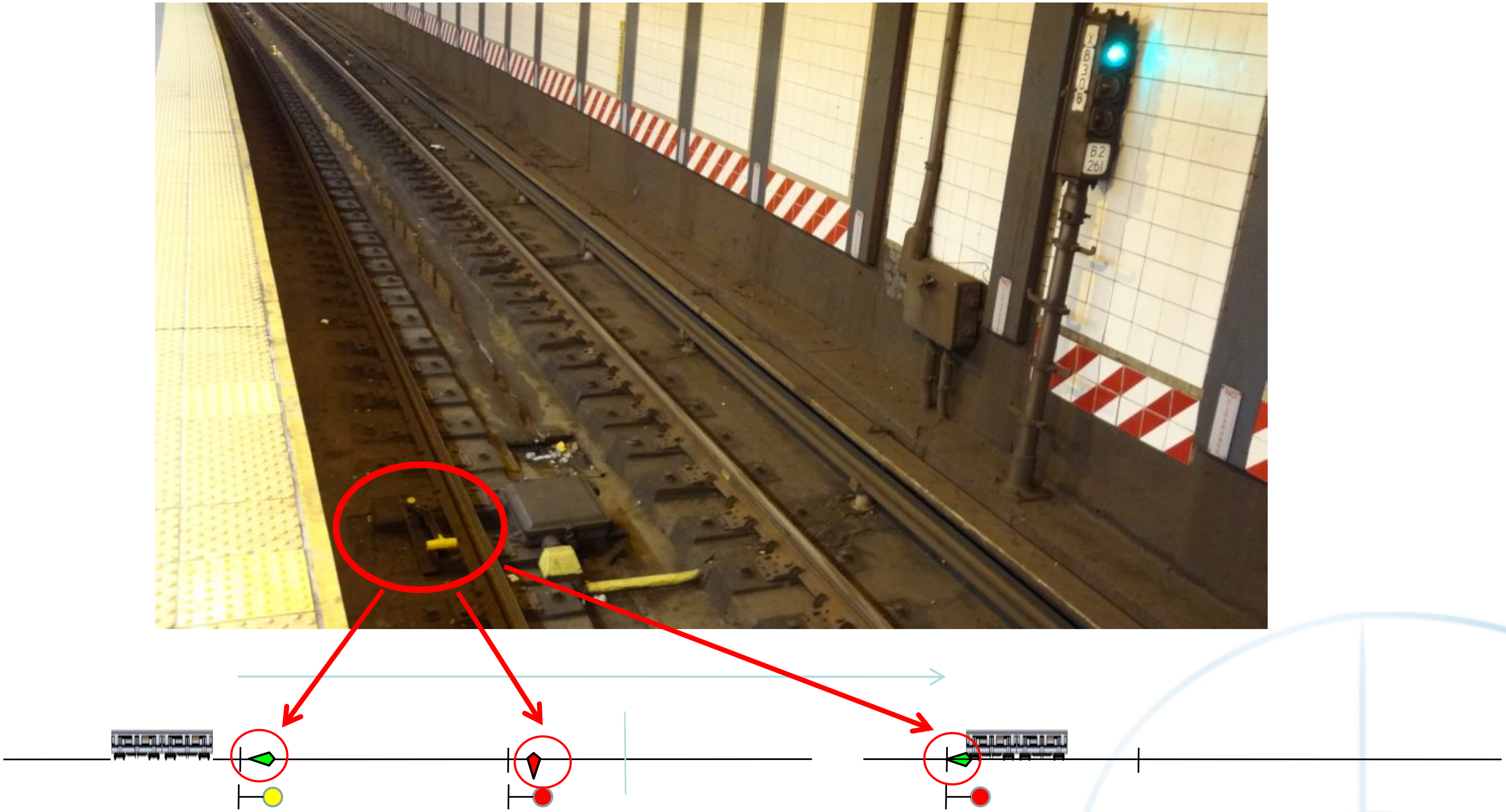


Line 7 CBTC: role & architecture

- CBTC = communication based train control
  - A system with on-board computers and wayside computers
  - Drives trains (automated but not driverless)
  - Interfaced to the interlocking (the system that drives switches and signals)

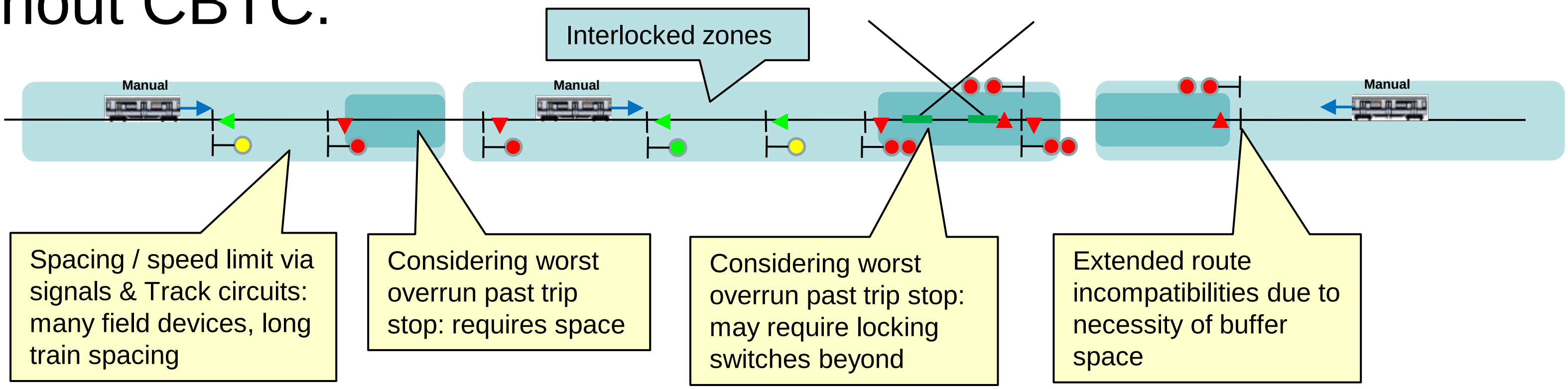


New York specificity: trip stops

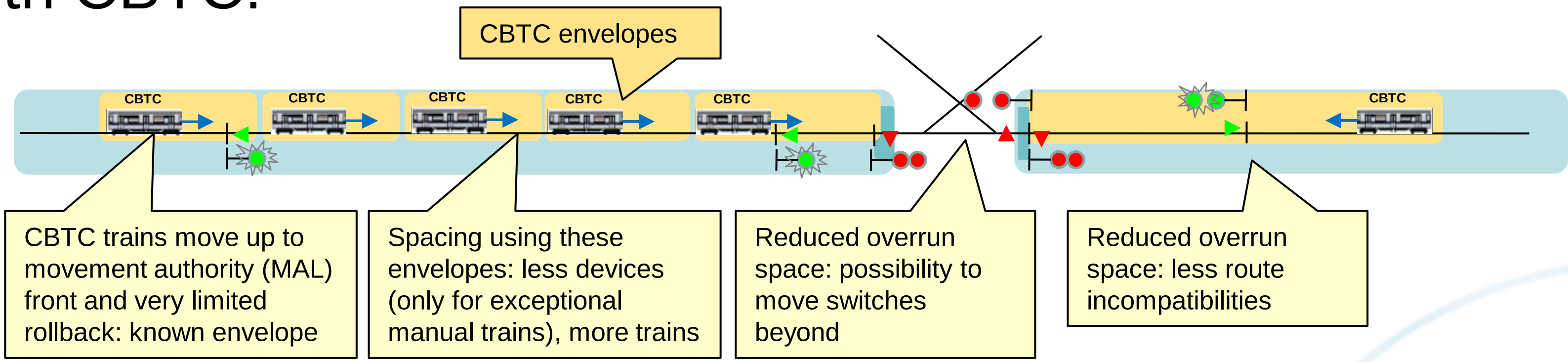


Line 7 CBTC: better train movements in safety

Without CBTC:



With CBTC:



- And more:
  - Possibility to insert virtual signals anywhere for exceptional movements, etc.
- Only: remaining manual trains should be few…

Line 7 CBTC: train positions

- CBTC trains need to determine and communicate their position:
  - Using localization transponders dispatched on the track
    - Mastering Transponder footprint, delays, accuracy, crosstalks, layout & maintenance…
    - First positioning after losing position: orientation determination
  - Using motion determination between transponders
    - Motion sensors (tachometers, accelerometers, beware slipping!)
      - Flushing: 1 free axle + 1 braked only axle, tachometers
      - Accelerometers to determine slips
    - Accuracy is paramount for performance, knowledge of accuracy is paramount **for safety**
  - Using track map
    - Transponder positions
    - Switch position (received)
- CBTC trains need to know their speed
- Radio communications

Line 7 CBTC: safe braking

- CBTC trains guarantee a movement authority limit (MAL) in front:
  - Proposed by the zone controllers
  - When trains accepts MALs: they should never overrun them
    - As long as no MAL beyond is proposed
- Thank to safe braking: worst case braking safe prediction
  - So trains trigger emergency braking when it will still stop them before the MAL
  - But not too early: paramount for good performances!
- How?
  - Guaranteed minimum braking on flat track (worst brake failures, worst slip conditions) known
  - Using safe determination of speed and position
    - To determine distance and **grades**
    - **Grades** are paramount (can double the stopping distance)
    - Using well known physics to predict braking with grades from flat braking
    - Beware kinetic energy hidden in heavy rotating axles
    - Passenger masses not known
  - Taking into account the delay to establish emergency braking
    - Residual acceleration phase, coasting phase (and **grades** during those phases…)
  - *Performance optimization while remaining safe is the game here…*

Chosen target safety properties

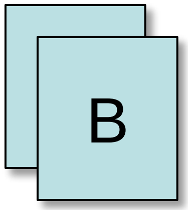
- Main chosen property:
  - **at all time, for each train we can define a protection zone PZ such that:**
    - The train is fully inside PZ, *and will remain inside PZ thanks to its own braking capabilities if PZ remain the same*
      - This one a bit tricky, to be detailed…
    - PZ contains only locked switches and no other obstacle than the train itself
    - PZs do not intersect with each other
- We also need “no over-speeding” (easier to formulate)
  - Because over-speeding derailment are possible
  - Because the PZ proof will need that (in “trains remain inside PZ” sub-proofs)
- We have something well defined to prove
  - If we succeed to define PZ at all time and in all cases
    - Describing all PZ evolutions
    - So that above properties hold,
    - Relying on things matching the design and the actual conditions
  - Then OK.

Obtained final outputs

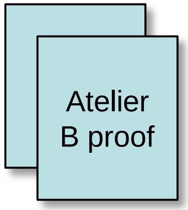
- At the end of the process:

Book(s) of assumptions in plain English

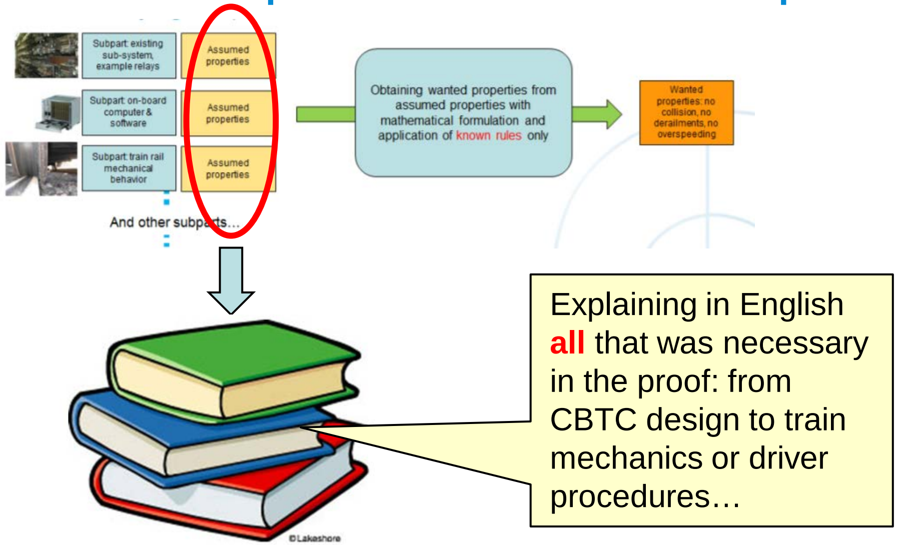
B models



Proof files



- Book of assumptions: the main output



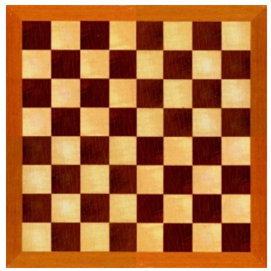
Proof should be verifiable, even without formal methods

- Very often: design and safety are "closed"
  - Relying on expert opinions
    - Final conclusions available, but reasons why not fully available
  - Design: important details & "reasons why" known only by few persons
    - Impossibility to understand without mastering **all**
- Idea here: proof should be verifiable
  - Like a regular mathematical proof: "everybody can read and nobody finds a failure in the logics"
    - Here: simple logics in general, assumptions are paramount
      - Everything needed is called an assumption…
  - Knowing the assumptions (thanks to the book of assumption), with some clues about how to reason, <u>the reader could re-do the proof in its principles</u>
    - How to reason: proof path § in books of assumptions
    - Using Atelier-B tool: for a computer-aided validation of the correct formal definition and correct proof, but this should not prevent a clear, readable proof

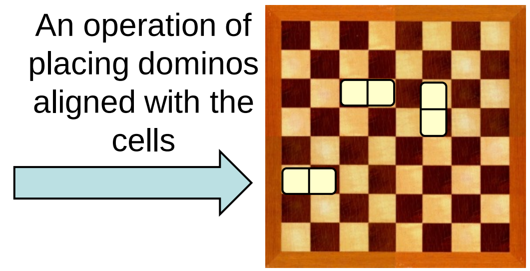
Reasoning with defined rules and assumptions: Dijkstra example

- The famous Dijkstra example:
  - Design:

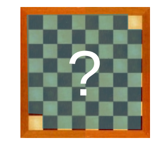
Chessboard



Pile of dominos



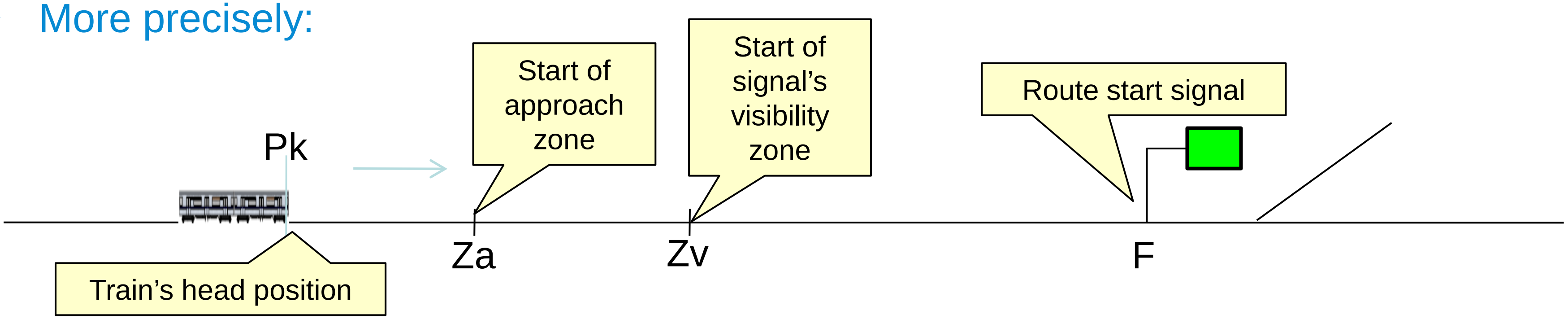
  - Property: placed dominos will never cover all the chessboard less the lower left and upper right cells exactly



  - The key: if B and W are the number of black / white uncovered cells, B = W all the time
    - *Because placing one domino always covers exactly 1 black cell, 1 white cell…*
    - So reaching a state where W = 2 and B = 0 is impossible
- Not a closed expert's opinion…
  - Because the action of a single operation causing B:=B-1, W:=W-1 obviously keep the property B = W
  - Very simple mathematics indeed, all is in the formulation of the assumptions
    - If we define the chessboard / dominos / placing geometry with the important properties (B and W evolutions), obvious!
  - Something everybody can read and verify (and not using so many cases…)

Reasoning with defined rules and assumptions: route cancel example

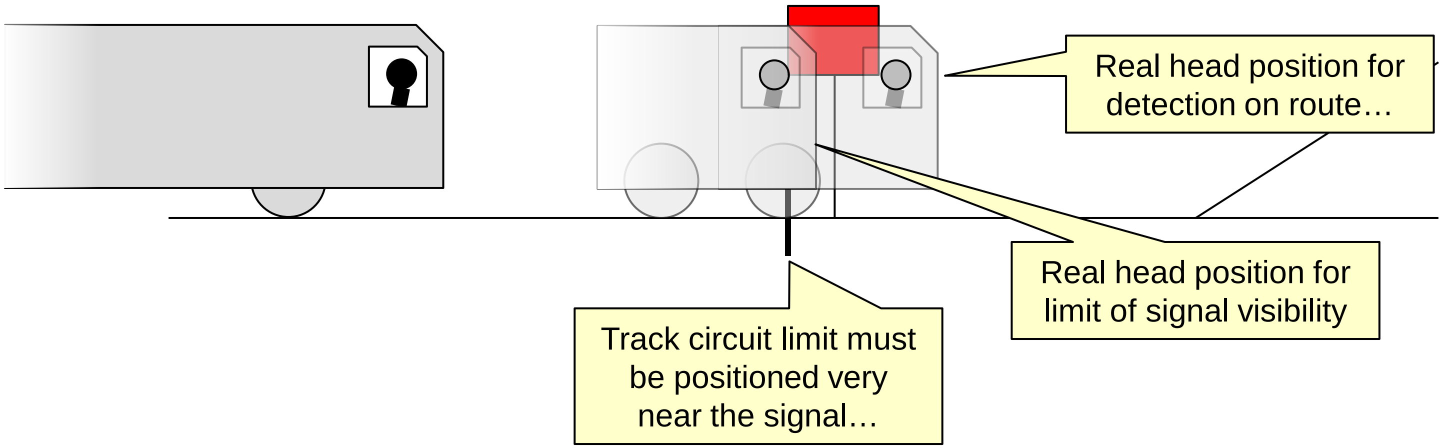
- Cancelling a route (without trip stops / CBTC):
  - If the approach zone is occupied, wait a delay T before actually unlocking the route (and switches)
  - Then no train should be on an unlocked route (wanted property)
- More precisely:



  - Assumptions:
    - If F is red and visible from the train (Pk>Zv), the train stops in less than Ts (delay) and Ds (distance). Including train operator reaction time… And stay stopped after.
      - Ts and Ds are shorter than T, Za or Zv
    - if train is beyond F, route cancel is neutralized (train detected)
    - If train is beyond Za, route cancel delay T is applied
    - Train arrives from left and does not jump (Pk increasing continuously)
  - Property: if Pk>F, then route remained locked
- Reasoning now possible… Simple!

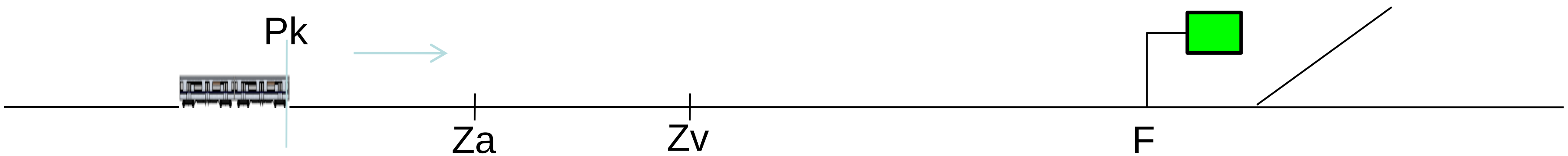
Reasoning with defined rules and assumptions: route cancel example

- Once formulated (assumptions / target properties), things seem simple…
- But assumptions have to be carefully examined in real world:



  - Assumption “route stay locked if Pk>F” is slightly wrong (in fact: if Pk>F+delta1)
  - Assumption about signal visibility ending at F is slightly wrong (in fact: end at F-delta2)
  - Problem if route cancel when train stays in F-delta2, F+delta1
    - Some CBTC drop position if a train stay here too long…
- Well defined, formulated assumptions can and must be confronted with reality
  - Thanks to their precise definition

Reasoning with defined rules and assumptions: route cancel example



- In fact, reasoning for “if Pk>F route not unlocked” very simple (apart from the previous trap):
  - Starting from a situation with green signal and train before Za, Zv
  - Let t0 be the signal cancel time
  - If train before Zv at t0, train stops before Zv+Ds (before signal), Pk never beyond F
  - If train at t0 is beyond Zv: train will stop before t0+Ts, so before t0+T (route not yet unlocked)
    - If the train is stopped beyond F: route never destroyed
    - If the train is stopped before F: Pk never beyond F
- No complex mathematics involved
  - Although involved formal tools to force full formal definition & proof correctness
  - If complex mathematics are needed:
    - Usually means that we are trying to re-prove the scientific result used in the design… => No.
- The most important action: requesting properties to be obtained from well defined assumptions via logical rules only leads to:
  - Well defined (verifiable) assumptions
  - Well known “know why”

Different types of assumptions

- CBTC design assumptions:
  - Software design assumptions
  - Hardware design assumptions
- Context assumptions: all other assumptions
  - Assumptions about external systems (example interlocking)
    - Assume the **global behavior properties** only
      - Such properties could be proven, but only by going into the external system's design
  - Assumptions about how trains or people behave
    - Consequences of physical laws and probabilities
      - Example: both tachometers equipped wheels will not slip together
        - Because 1 free, 1 braked only. OK, but…
        - Proof done under this assumption (even though CBTC design includes this case)
  - Known physical laws
    - Introduced in the proof as assumptions
- *Everything is called "assumption" here…*

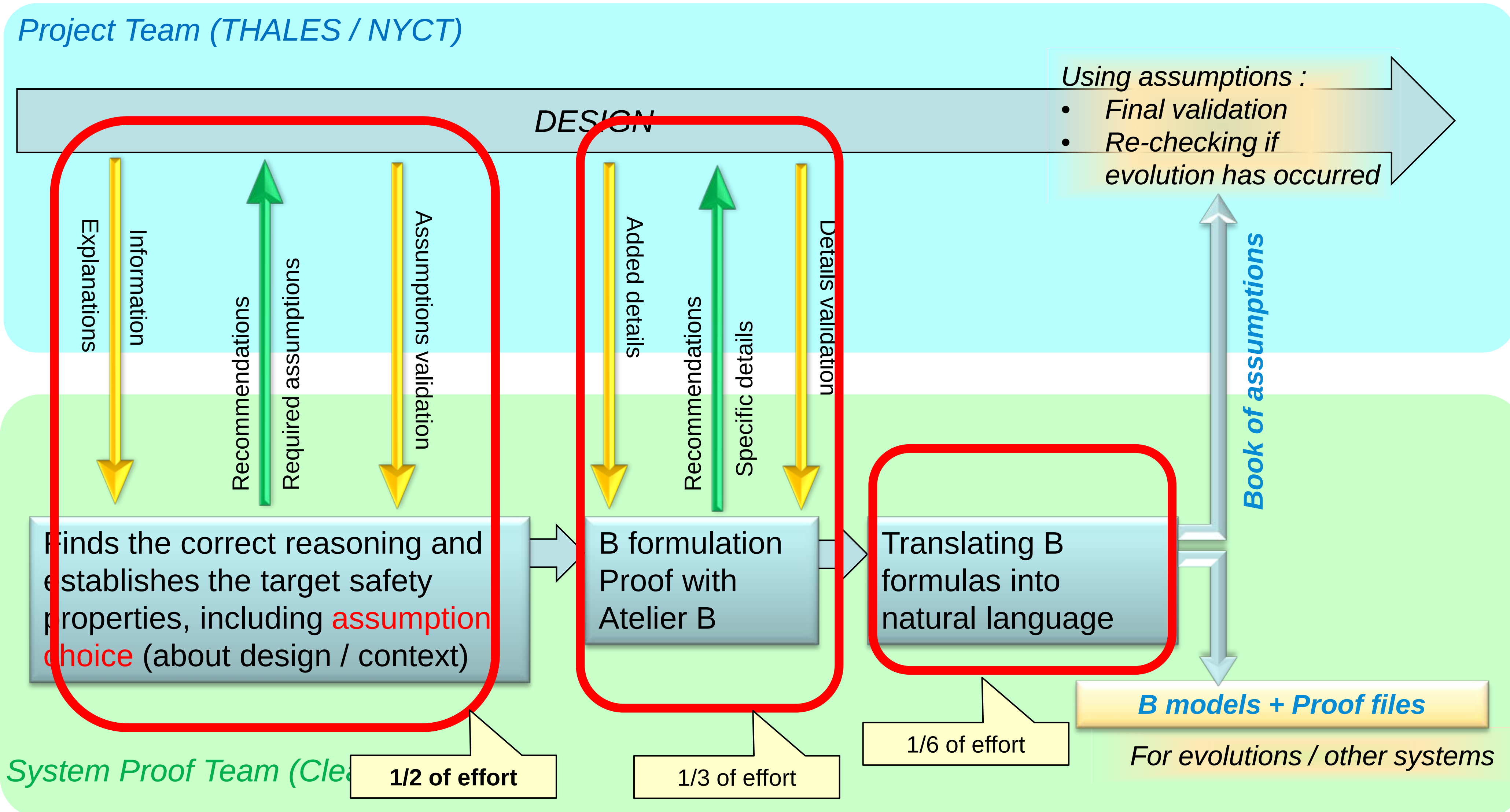
Methodology: choosing assumptions

- Choosing assumptions and finding the correct reasoning are linked processes
  - Realistic assumptions matching the design and conditions: expert knowledge
    - previous example: visibility zone and detection zone
  - Finding “why it works” is replaying the designer’s reasoning (again expert knowledge)
    - Example: dimensioning Za, Zv and T
- Communication with experts is paramount
  - No re-inventing
  - The proof team should add the rigor and well-defineness in existing elements

Finding why it works: methodology

- Methodology to “Find the correct reasoning”:
  - “animate” the system via scenarios, seeking to brake the property (in our example : seeking train collisions)
    - Find out missing design details to do these animations
      - Thus selecting only the relevant details (out of all design details)
  - Find why scenarios leading to collision do not work
    - Find intermediate properties
    - Assumptions to remove collision in context considered unrealistic
  - Prove intermediate properties leading to global property
- **Need for a “natural language proof” phase first**
  - Priority: communication with designers / experts
    - Finding the correct reasoning
    - Finding realistic assumptions
    - Targeting at formulation without spending time at that stage
- Note: formal methods used to force full formulation and to detect any error, *not to find the correct reasoning*

Global methodology



Natural language proof phase how-to

- Do not try to read all documents first
  - Example: explained with relay names and schematics, the route cancel example could be *very* complicated…
- Communication with designers is paramount
  - Using documents only is not fast enough!
  - Assumptions have to be chosen with designers
- "Lightweight" temporary documents for communication
  - Drawings, short texts
  - Meetings (teleconference to avoid losing time in travels)
    - Describing precise understanding and asking confirmation is very efficient
    - Even if things have to be confirmed in written form after, **the amount of information exchanged via discussion is far greater**
- *Need to obtain a formal proof as a motivation*

Natural language proof phase benefits

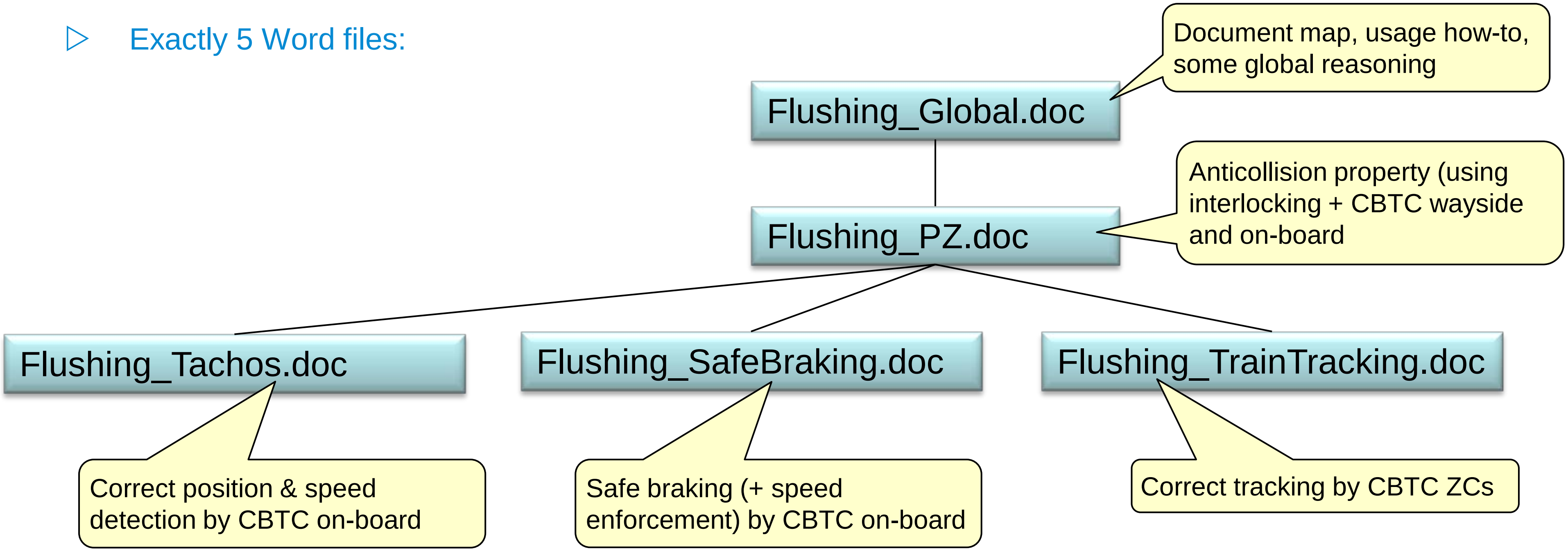
- Building the reasoning with the designers provides an **immediate feedback**
  - Assumptions / reasoning review meetings (teleconference)
  - All participants get familiar with the emerging reasoning
    - Gathering CBTC experts, rail operating experts around common topics
  - Questions about delicate assumptions / special cases known early
    - With enough time to deal with them
- The value of a global reasoning based on defined assumptions is shared as that phase
  - Early in the project → best benefits
- Avoiding any “tunnel effect”
    - Tunnel effect: if the proof team’s work remains invisible too long
- In the Flushing project: ClearSy / NYCT / THALES meetings
  - average teleconferences rate ~4h/2weeks

Final outputs

- At the end of the process: Book of assumptions
  - Main contents: assumptions
    - **Precise definition**
    - Who may validate each assumption (OBCU experts, wheel/rail contact experts, etc.)
    - How to derive tests and verifications for the assumption, method:
      - Link real objects to notions, using explanations given in the documents
      - See hold / not hold cases, use “example if wrong” method, check proposed method and notes in the document
      - Derive concrete verifications to do on the final actual device
  - Usage:
    - Re-validate the assumptions to guarantee the target properties
      - After any project’s twists and turns
      - In case of evolutions, changes
    - Validate the assumptions for other similar systems
      - Or a subset of assumptions corresponding to a sub-property
    - Understand why the property is guaranteed (replay in manual reasoning)
- We also get B formal models and proof files
  - Require B knowledge of course…
  - Usage: after a system evolution, change…

Flushing: book of assumptions

- Exactly 5 Word files:



- In every file (except Flushing_Global):
  - Proof targets § : properties that are guaranteed by proof
  - Assumptions § : assumptions under which the proof holds (for each target property)
  - Sub-proofs § : properties used as assumptions (for each target) that are target properties below
  - Shared notions § : things we had to define to express properties and assumptions
  - Proof path § : clues about how Atelier-B prover proved the target property

Formal modeling phase

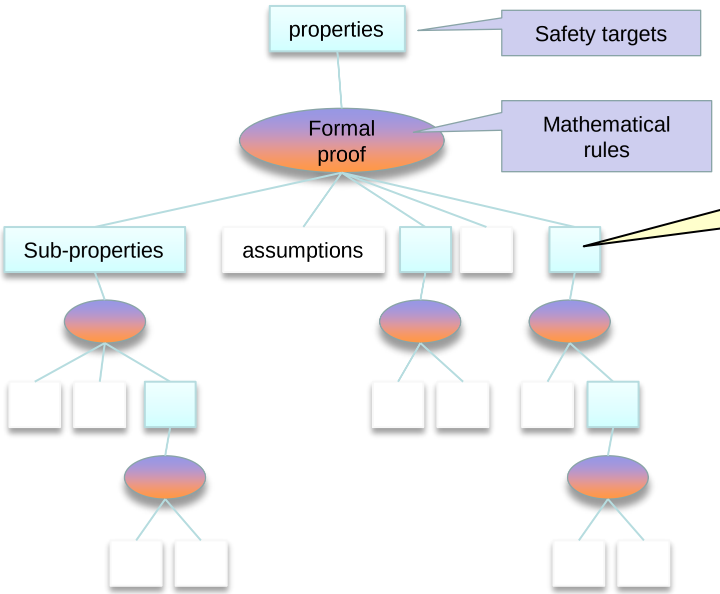
- Convert the previous work into B-models such that the proof of these models are **equivalent** to the previous reasoning
- Why necessary?
  - ***we know it's precise enough to be formalized only if we formalize*** (even if natural language proof was meant to be formal)
- The book of assumptions is obtained from B models
  - Experience: assumptions change shape from how they were explained before B models (during natural language phase)
- Pure B modeling & proof: average 1/3 of global workload

Final phase: redaction, proofreading

- Final phase: from B models after proof, write the **Book of assumptions**
  - Made for direct usage
    - In particular: no B variables and names inside!
- To be done at the end:
  - one redaction costs less than many…
    - Thanks to communications (natural language phase), no “tunnel effect”
- Internal proofreading: paramount
  - We have precise things to verify
    - Each notion shall be well defined
      - Well defined = On any real scenario, interpretation should be undisputable
    - Assumptions: checking how they can be validated (and by who)
  - This proofreading done on the B models before the documents
    - Focusing on B notion to reality links
- *All proofreading / documents : 1/6 of the total workload*

Global price / level of detail

- How to evaluate the global price of such a formal system level verification on a given system?
  - If not done before, by definition the reasoning is *unknown at start*
  - Depends on the complexity of this reasoning
  - Depends on the **level of detail**
- Level of detail: a paramount question
  - Wanted properties are proved using assumptions and sub-properties:



Choosing what is taken as an assumption or as a sub-property (subject to a sub-proof) defines the level of detail

For instance, consider the route cancel example: if delay properties must be proven from the **relay schematics**, additional work & cost. But relay errors causing delay bypass in some case could be found…

  - Level of detail must be decided by a clear criteria

Choosing the level of detail

- Choosing the level of detail determines what is proven
  - Previous example: route cancel proof including / excluding relay schematics
    - Determines if relay mechanisms will be proved
    - Determines the shape of obtained assumptions
      - Excluding relays: “route cancel with occupied approach zone causes T delay…”
      - Including relays: “all cancel circuits are made according to XXX schematic…”
  - Obviously: the deeper we go, the easier assumption validation is…
- Level of detail: a choice involving the customer
  - Agree on a well defined criteria, agree on each particular case afterward
- Flushing: only system level, but with detailed CBTC algorithms
  - Including: (examples)
    - Pulse counting from tachometers (and specific points about direction change or slipping)
    - Kalman filters for the speed measurement
    - How gradients are used in the safe braking model
    - Wayside to on-board communication: messages worst case dating, messages crossings, timeouts
    - Train tracking: exchange of unequipped train suspicion between zone controllers
    - Possible signal overruns (manual trains), associated locking including provisions for returns or mode changes
    - Routes cancel and possible race conditions in the wired interface between CBTC and interlocking.
  - Excluding:
    - Actual code reviews (in particular: not including software track representation)
    - External systems design (example interlocking relay schematics, however used to deduce global properties)

A glance at the Flushing proof

- Target properties and their value
- Top level “Protection Zones” proof
  - Understanding the global reasoning
- Proof decomposition in sub-properties
- For each part of the proof:
  - A glance at the assumptions & sub-properties used
    - To get an approximate understanding
  - *Sorry, only “a glance”, not really replaying the proof*
    - *This was a 4 days presentation to NYCT experts… With confidential details!*

Global property presentation

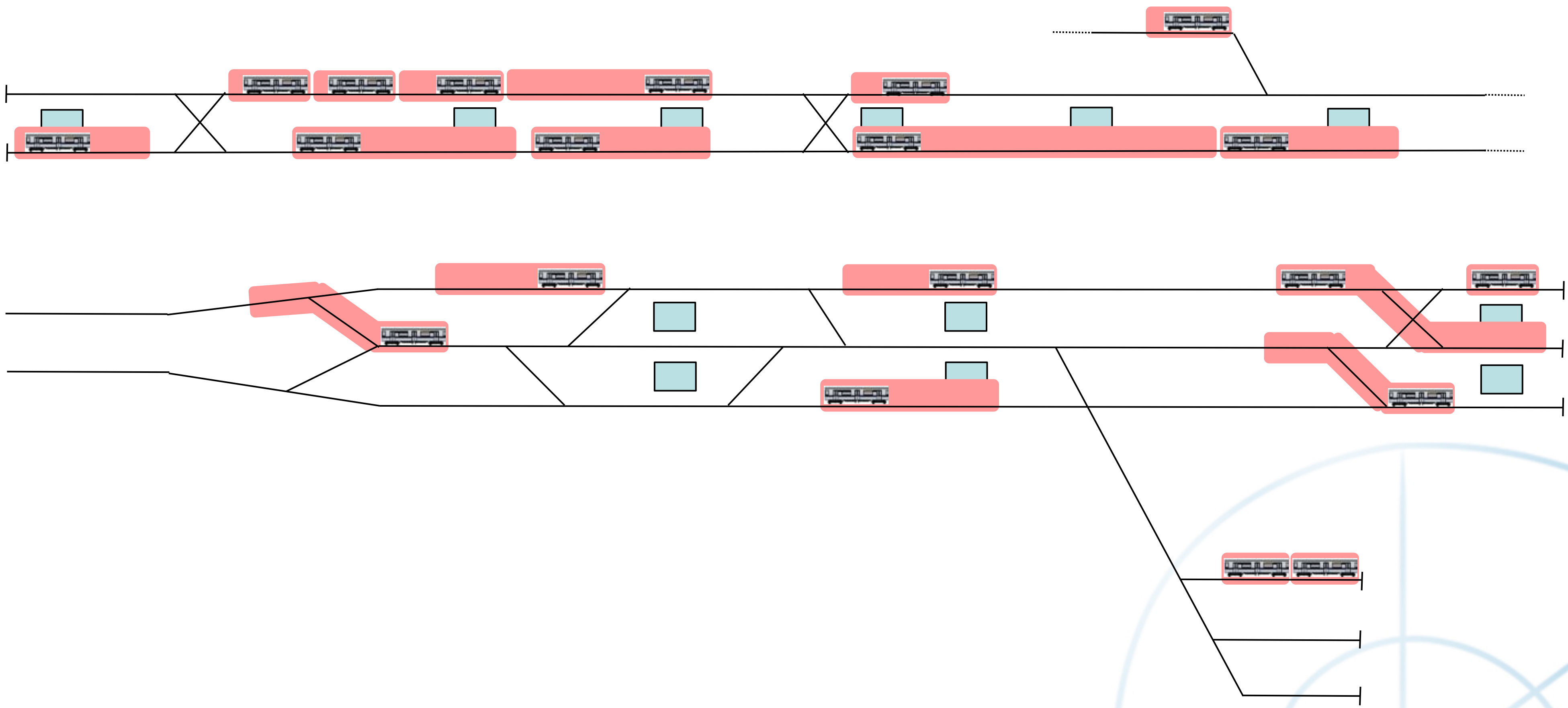
- 1: Train to train collision and train derailment over an incorrectly positioned / unlocked switch are impossible

- 2: CBTC train over-speeding is impossible

  - With these properties, a whole set of accidents are impossible
  - In fact, properties 1 & 2 are means to ensure no injuries on persons
    - Sub-properties of a more (too?) global proof…
    - Using extra assumptions, about other ways to have injuries
      - Fire? Electrocution? Smokes? Aggressions?

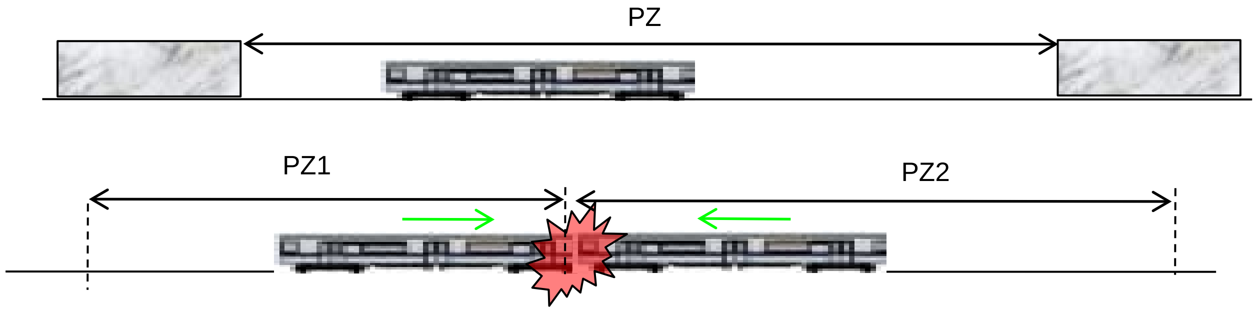
An idea of how we prove no collision

- At all times, there exists a set of disjoint protection zones PZ, such that each train remains inside its PZ under its own braking.

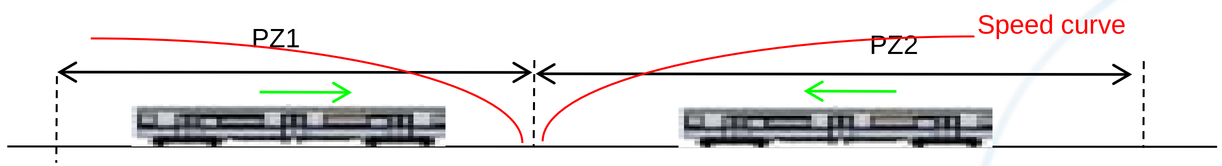


An idea of how we prove no collision

- Why "under its own braking":
  - Examples of collisions with trains remaining inside PZ otherwise:

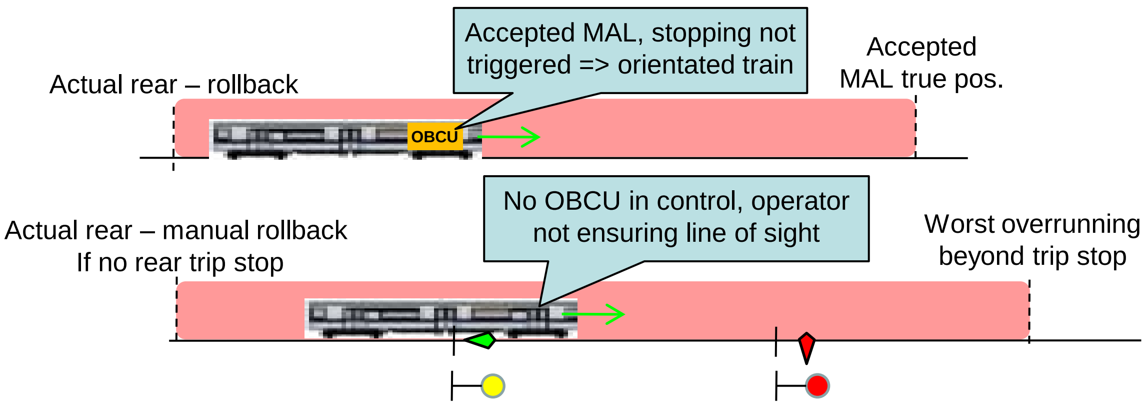


  - So at all time t, if after t the PZ of a train remained unchanged this train should stay inside this PZ and the corresponding proof should rely only on guaranteed braking forces:

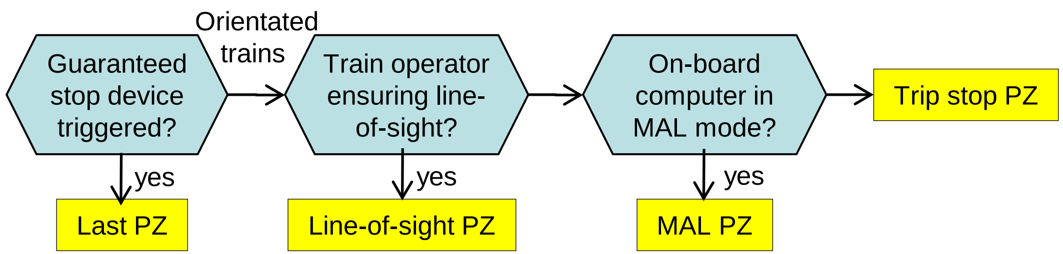


An idea of how we prove no collision

- PZs defined using “well defined” criteria. Examples:



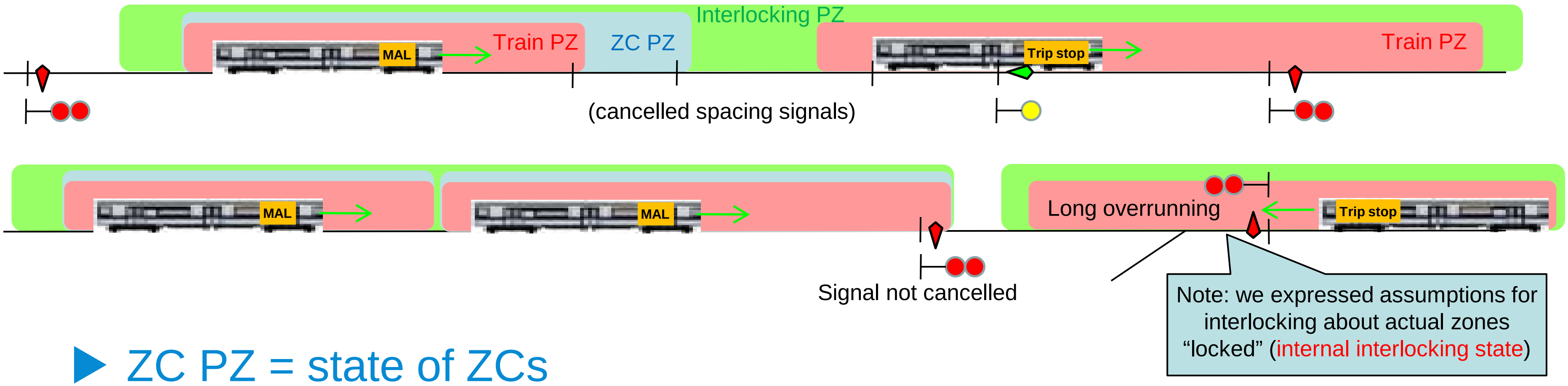
- How to find type of train (as defined in proof)



- Defining PZ in each case to prove their existence. Then:
  - Each train remains in PZ with its own forces (assumptions & sub-proofs)
  - All evolutions keep PZ separated, with locked switches (induction)

An idea of how we prove no collision

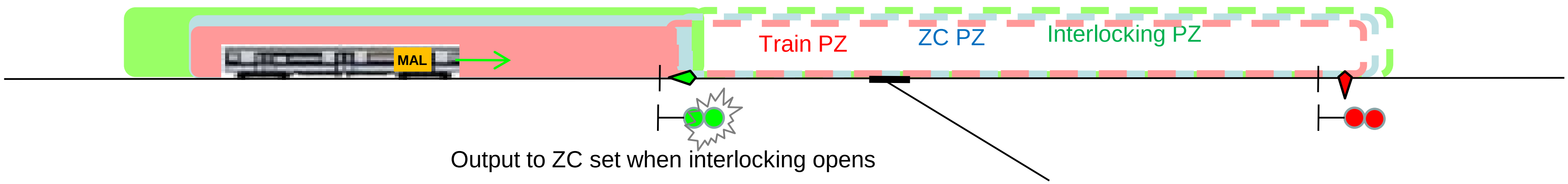
- We define PZ precursors, from which train PZ inherit properties : “zone controller PZ” and “interlocking PZ”:



- ZC PZ = state of ZCs
  - CBTC Controlled trains PZ inherit properties when receiving telegrams from ZCs
- Interlocking PZ = state of interlocking
  - Uncontrolled trains PZ inherit properties thanks to trip stops (and signals…)
  - ZC PZ inherit properties from interlocking zones thanks to interlocking -> ZC inputs
- ZC & Interlocking PZ properties proved by induction also

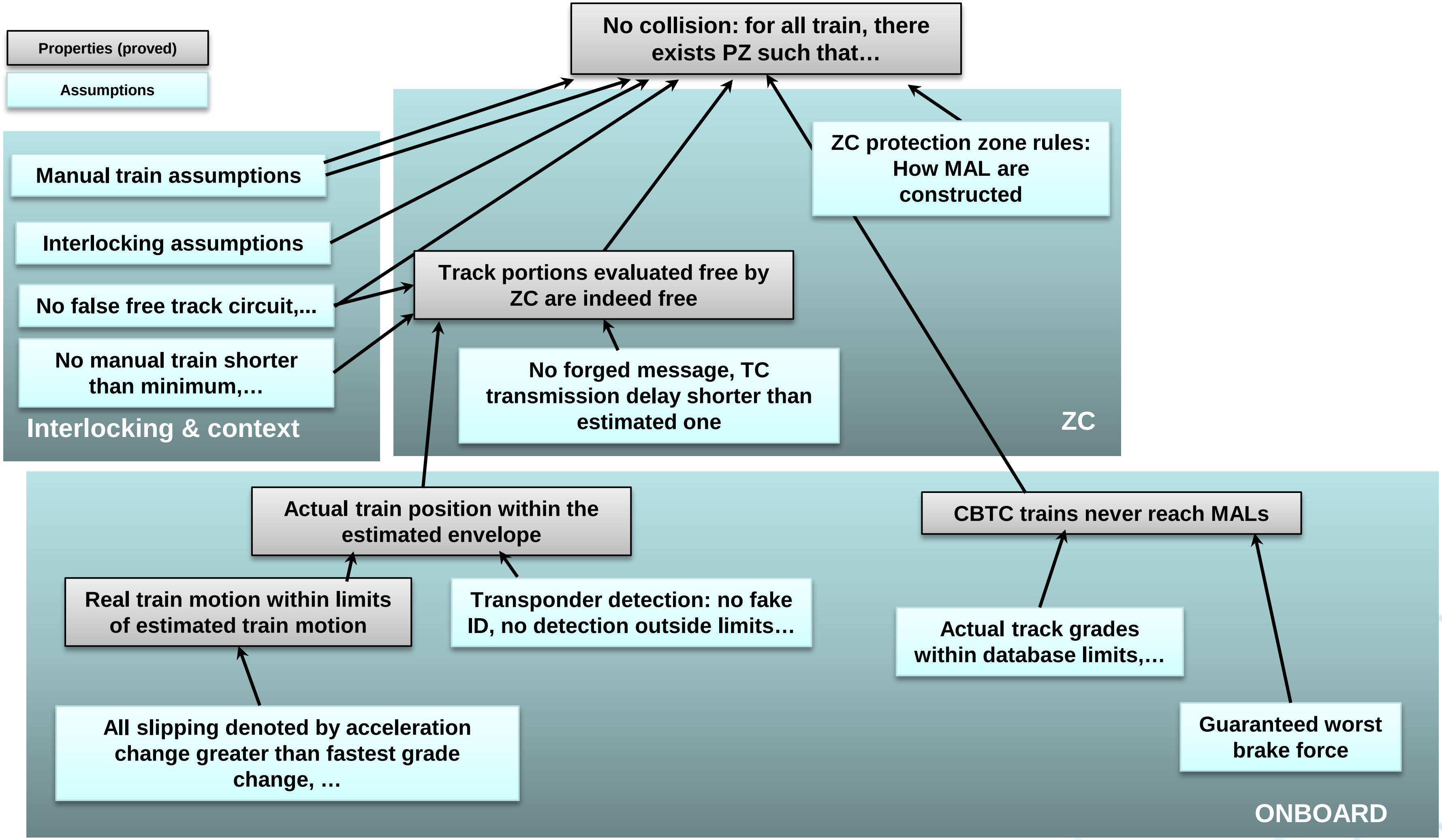
An idea of how we prove no collision

- PZ evolutions: where the proof is…
  - Zones extensions: up to signals or next obstruction



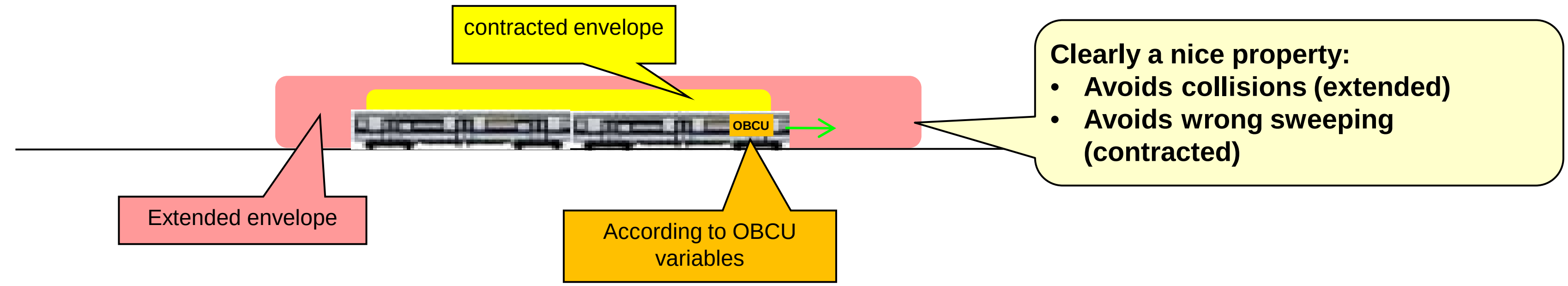
  - Zones rear reduction: freeing inaccessible back space
  - Zone front reduction (or middle = splitting): more delicate
    - Example: for interlocking, only if
      - ZC says next train keep the new limit
      - Or if more than time or distance worst stopping limit
        - Signal cancel at t0, x0 => stopped before t0+T or x0+D
- Output assumptions appear:
  - Example: Interlocking should clear signals only so that corresponding “locked zones” (interlocking PZ) do not intersect

Properties & sub-properties



Position / speed correctness property

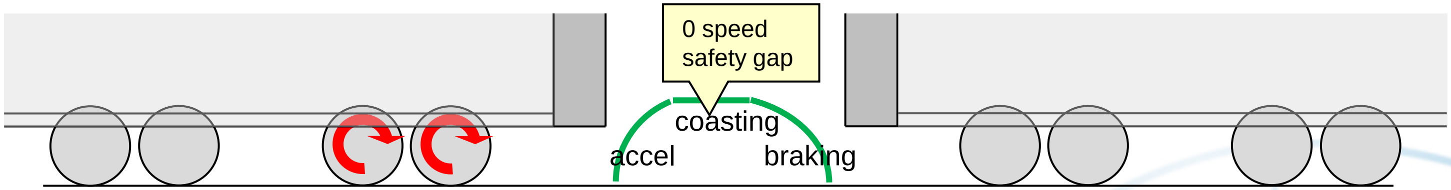
- Target properties:
  - Always: contracted envelope inside real train inside extended envelope



  - Real train speed is always within calculated train speed +/- calculated uncertainty
- Assumptions:
  - *Correct calibration / orientation by localization process*
  - *Slip/slide and sensor failures limitations, track grade limitations (probabilistic)*
  - *Correct train OBCU constants*
  - *Correct OBCU transponder database*
  - *Characteristics of transponder detection (and transponder layout with unique ids and limited crosstalk)*
  - *Limitations of rail (worst turns…) / tachometers errors, counting errors*
  - *OBCU computing assumptions, guaranteed cycle time*
  - *Automation knowledge assumptions (Kalman filters)*
  - *Maximum speed / acceleration (for instance for tachometer ticks counting…)*
- Proof feasible with this: OK!

Safe braking (and CBTC speed correctness)

- Target properties
  - A train in MAL mode never violates its (accepted) MAL
  - A train in MAL mode never overspeeds
- Assumptions
  - *The speed / position determination is correct (sub-proof)*
  - *OBCU constants matching train EB characteristics and masses characteristics*
  - *EB stronger than worst case grade*
  - *Other forces (wind, …) negligible*
  - *Probabilistic assumptions for very odd cases (example max spinning at start train)*



  - *OBCU computing according to wanted formulas (+ guaranteed cycle time)*
  - *Physical laws assumptions (kinetic energy…)*
  - *Correct grades in database*

Train tracking

- Target properties
  - Free zones according to ZCs are indeed free
- Assumptions
  - *Correct ZC constants*
    - *Maximum train acceleration*
    - *Minimum train length, max overhang (length from first axle to train's front)*
    - *Track ends are really ends*
    - *Track circuits longer than shortest truck interval*
    - *Track circuit map correct*
    - *Dead zones shorter than limit*
    - *Known TC acquisition time*
  - *No trains appearing in the middle*
  - *ZC computes according to algorithms, guaranteed cycle time*
    - *Including ZC to ZC communication and ZC to train communication*
  - *Trains move on linear track portions (no incorrect switch reached, cycling sub-proof, proved)*
  - *CBTC Trains give correct envelopes (sub-proof)*
  - *Minimum assumptions for communication layer: no forged messages*
  - *Assumption about OBCU communication (example: calculated envelopes are sent)*

PZ proof (top property)

- Target property
  - At all time, there exists a set of disjoints protection zones PZ, such that each train remains inside its PZ with its own braking. No unlocked switches or obstacles inside PZs
- Assumptions
  - Interlocking assumptions (using locked zones notions)
    - No switch movement in locked zones
    - Unlocking no longer accessible parts
    - Clearing signals only into locked zones
    - Interlocking protection zones extension / reduction compatible with train capabilities
  - Train procedures assumptions
    - Example: when restarting in manual a failed CBTC train, TO must proceed in line-of-sight to next signal
  - *ZC computes according to algorithms, guaranteed cycle time*
    - Including ZC protection zones extension / reduction rules according to train / interlocking
    - Including ZC to ZC communication and ZC to train communication
  - *CBTC Trains give correct envelopes (sub-proof)*
  - *Correct train tracking (sub-proof)*
  - *Correct safe braking (sub-proof)*
  - *Minimum communication assumption: no forged messages*
  - *Assumption about OBCU communication*

End of the “glance at global proof”

- Just a glance, of course!

Safety and proven properties

- Flushing formal verification:
  - Proved properties:
    - No collision and no derailment (“PZ” proof)
    - No over-speeding
  - Level of detail: system
    - Including algorithms, excluding low level design (actual software code)
- Position of this work inside the global safety assessment
  - Among process audits, failure determination, etc.
- *The right balance of formal efforts among other efforts is always to be carefully examined*

Proof and failures determination

- Our assumptions are supposed to hold despite any possible failure or failure accumulation
  - Possible = probability not below what is required for this safety level
- Example:
  - We have assumptions about how a localized OBCU updates envelopes
    - Assumption *: if OBCU localized, then envelope update should conform to…*
    - So : this assumption does not requires anything for a non-localized OBCU
  - If the OBCU has a power failure → no problem, the assumption still holds due to the definition of "localized state" (no longer localized)
    - If OBCU has a memory corruption (always detected): same reasoning.
  - Chosen assumptions are those that are required despite any failure
    - Crash possible when they no longer hold…
- *So possible failure determination / accumulation probability determination is still required!*

Failures

- No property will withstand *any* failure…
  - Example: safety relay do not operate without command
    - Would this withstand "sabotage level" failures?
- Probability considerations to remove extremely unlikely cases: *always needed*
  - Assumptions in the proofs hold, unless those extremely unlikely cases
  - Some assumptions are explicitly probabilistic (example: no undetectable slip)
- If occurrences where target property does not hold must be $<10^{-9}$, then cases where 1 assumption does not hold should be (at least) less than 10^{-9}…
  - With some possibilities to avoid accumulating worst cases too far:
    - Example: positioning proof (real train inside computed envelope) holds under assumptions "input inside worst case bounds"
      - Then put numeric values in this proof only for non-impossible worst cases sets
        - *Reaching worst case bounds on every input: very improbable!*

Risk analysis

- Standards require that safety assessment starts with a risk analysis
  - Considering accidents
  - Deciding what is acceptable and what is not
- Train head-on collision to be avoided with SIL4 compatible level
    - Occurrence less than 10^{-9} per hour, mean time between occurrences 114 000 years
  - How was this decided?
    - Considering the potential number of killed…
    - But there is **an acceptable / not acceptable decision** (human judgment)
- This kind of decision is not a matter of proof
  - Risk analysis are still required, they are the basis of good target properties choice
- As standards say, it is very bad to poorly decide risks…

Safety Integrity Levels & methods

- Standards require appropriate methods to mitigate **design errors**
  - Called systematic failures
  - Probability computation considered irrelevant here because occurrences are systematic under some conditions
  - Standards define appropriate methods to prevent systematic failure, according to the target safety integrity level
- Assumptions from a system level formal proof have an "inherited" SIL level
  - So appropriate methods are applicable for the underlying design
- No design errors in the system level design (covered by the proof): considered SIL4
  - But standards usually consider formal methods at software level ("Highly Recommended" for SIL4 in 50128…)

Formal at system or detailed level?

- Often, formal methods are used only at software level
  - Proof covering the "software specification → software code" step
    - Preventing errors in the code that were not in software specifications
- Here: proof from top level safety properties to system design
- Comparison?
  - System level proof generally dedicated to safety properties only
    - Software proof generally include functional aspects (because included in software specifications)
    - At system level: functional aspects = performance (example: reducing train spacing "as much as safely feasible")
  - System level proofs cover "all aspects"
    - From underlying software algorithms to train procedures
  - In software proof, direct link from lower level models to code (code generation)
    - Direct action against low level coding bugs
    - System level proof provides output assumptions about properties to be ensured by the software: indirect action
- A matter of choice and balance
  - *Putting the lightning rod where the lightning may strike…*

About project processes / organization

- Project success and safety are linked, many possible ways toward successful projects:
  - More process monitoring?
  - More proofreading and quality?
  - More training?
  - More science?
  - More people and means?
  - More testing?
  - More team / customers communications?
- *Nothing should be missing!*
- Formal proofs are to be inserted considering the balance with all this
  - And considering the safety effort required

Project share for safety

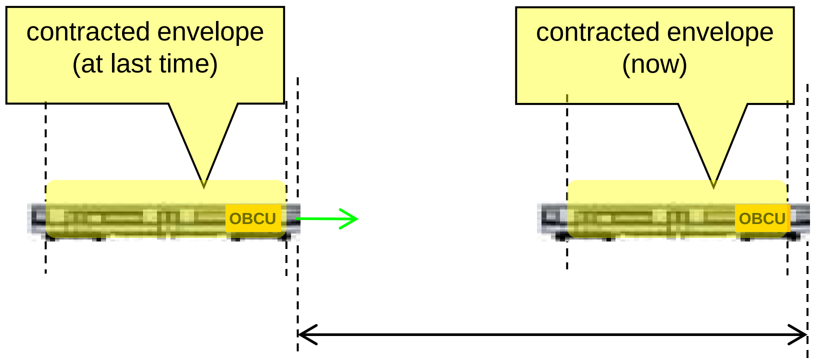
- Safety: a “performance” that does not show up in tests
  - Train spacing reduction thanks to a new CBTC is directly visible
  - Increased safety thanks to a formal proof (for instance) is not visible the same way…
    - And “no accident during X years” is not enough for systems where mean time between unwanted events should be more than 114 000 years…
- Return over investment more difficult to evaluate
  - New functional performances directly visible, not safety improvements
    - Unless very unsafe before!
- Good choice of project share for safety and optimized use of this share must be a constant concern
  - Too little spending on safety:
    - The system may be dangerous (if not blocked by safety assessments…)
  - Too much spending on safety:
    - System too expensive, risk of project failure
- The use of formal proofs have to be considered with this view
  - After an accident, things that should have been done always seem so obvious!

In the design or along the design?

- Formal proof at system level: to be coupled with designer's task or ISA (Independent Safety Assessors) tasks?
- With designers:
  - To favor communications with designers, design understanding and realistic assumptions
  - Shared and early knowledge of issues and special cases
- With ISA:
  - Independence
- Our opinion: whatever the organization,
  1. The proof team must be a specific team
     - Impossible to design a safe system and formalize the reasoning at the same time
  2. The proof team should not start too late (not when the design is finished)

About proof / design communication: those damned “notions”…

- Example: “envelopes update”



- It means that we should have for instance cemax = cemax_r + Scmin
- Will someone find *cemax* = *cemax_r* + *Scmin* in the software code?
  - NO, anyway the software code probably denotes track positions using branched coordinate system
    - Positions denoted by <segment name, abscissas>, not by abscissa only… Lower level design.
- cemax, cemax_r, Scmin = **notions** / notations from the proof
  - Efforts are needed to match them to the real software
  - Define only **unavoidable** notions
  - Use them near their definition (**do not ask people to remember them**!)

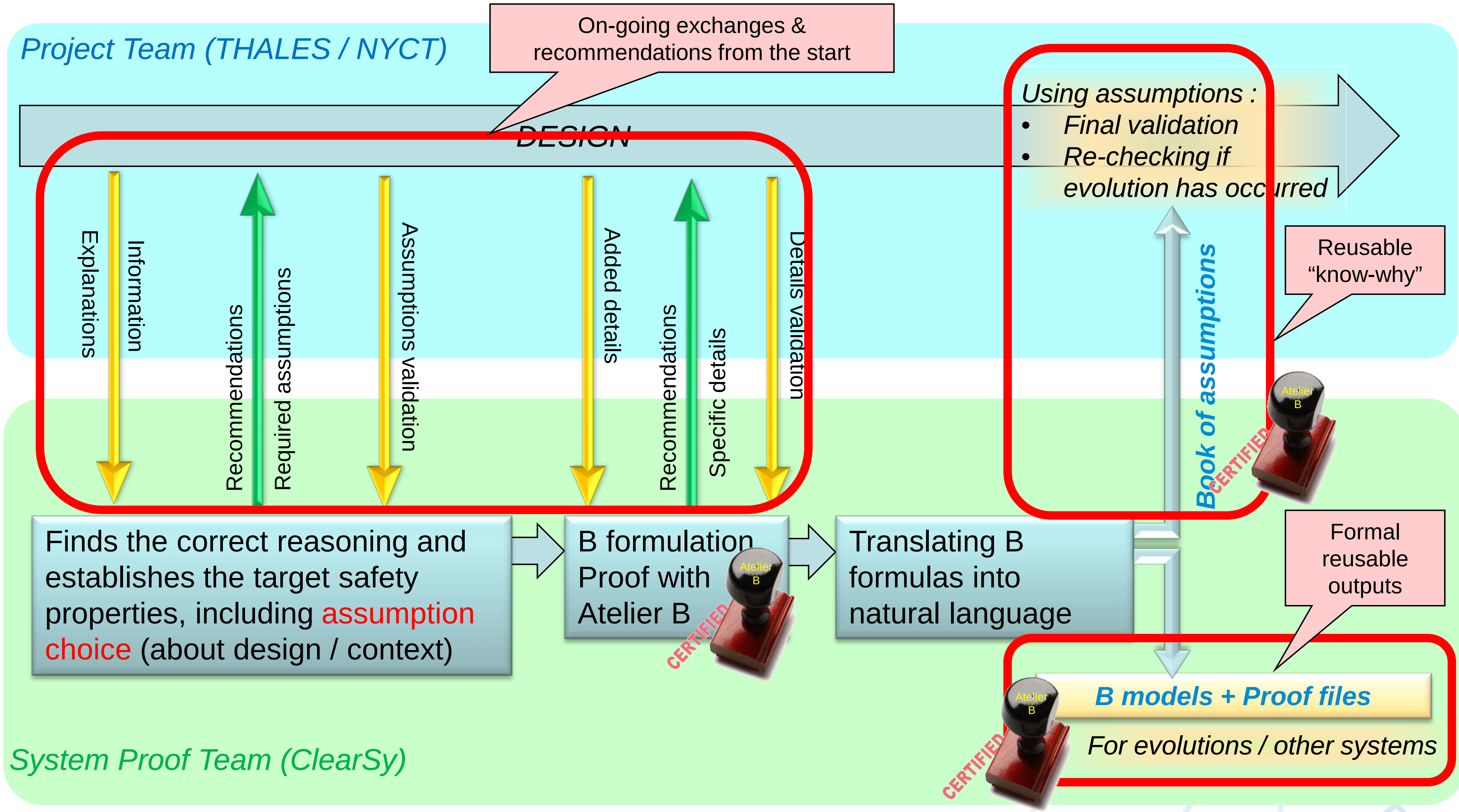
Necessary “notions”… At least should be well defined

- So all assumptions are expressed using words…
- Let’s imagine a fancy assumption: “the house is red”
  - But what is “*the house*”?
    - Walls? Roof? Inside? Outside?
  - And what is “*red*”?
    - Dark orange? Shiny? Striped?
- Defining this means linking words to reality
  - Designing a clear criteria to tell what is part of “the house” and what is not
  - And a criteria to tell what is “red” or not
- Only then can the assumption be correctly TRUE or FALSE
- Method: make precise the notion of “house” and “red” here
  - And use the notions in assumptions

Communication & Optimization of the proof construction process

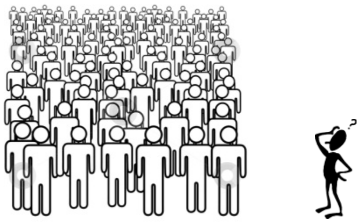
- Again, natural language phase is paramount to “find the way” from realistic assumptions to wanted properties
- Projects documents usually describe “how”
  - With functions names, messages names, etc.
- Bottom-up process:
  - Formalize every low level details
  - Deduce higher properties from this
  - Up to wanted properties
  - Our experience: this process is a **bad idea**
    - Because formalizing unnecessary details
- Our experience: the proof team should have the will to understand how wanted properties are ensured
  - *As fast as possible, as directly as possible*
  - Using contacts with designers in this spirit
  - Reading documents in this spirit
    - Although verification through full documents will be done after in the process
      - Checking that a function exist requires reading documents up to this function. Checking that *it does not exist* requires reading all…

Summary of outputs



System level formal proof: conditions for success

- *According to us…*
  - Proof team really willing to:
    - Understand the system ("plunge" in the domain)
      - But optimize their reasoning (use minimum necessary details)
    - Exchange with the designers (with the will to provide a service)
      - Using extra names and notion knowing that this is a pollution
    - Formalize the optimized reasoning (and only the optimized reasoning)
  - Organization: access to people really knowing the design
    - With enough time
    - Proof team // Designers: neither 1 to 100 nor 100 to 1!



  - Organization allowing easy / lightweight communication
    - Test: will teams exchange hand-written drawings (both directions)?
  - The proof team should master the formal method enough to use it as a tool (knowing what the method can do and cannot do)

Proof team skills

- A team leader is needed to constantly remind the previous "conditions for success" (previous slide)
- Of course, skills with the formal method / tool (B / Atelier B) are needed
  - As a tool, but this is not the main point
- Technical "openness" is paramount
  - Team members need to be willing to cross technical domains' limits

Benefits of a formal system level proof

- Usually: system safe because
  - Safety assessors gave a positive conclusion
  - Supplier has commissioned similar safety systems
  - A complete safety case has been approved
- With a “**replay-able”** system level proof: system safe because *in addition*
  - Impossibility of accidents has been demonstrated
    - Each proof step is verifiable using only pure logic steps
      - Everyone who wants it can check these steps
    - Properties are obtained from well defined assumptions
      - Everyone who wants it can see those assumptions (and understand what they mean in the field)
    - The proof steps correctness is guaranteed by a tool (Atelier B)
  - Anybody could read the proof,
    - maybe discuss some assumption validity in the actual system;
    - **But NEVER doubt that properties are logically deduced from these assumptions**

Deciding for a system level proof

- *Criteria (again according to us…):*
  - Need for a global safety guarantee
    - With a focus balanced on every part of the delimited system
  - Need to have all the necessary conditions at hand
  - Need to have the “reasons why its safe” at hand
    - At hand and re-playable
  - When there is no obvious pitfall to correct first
    - Either technical or organizational
  - *The strength of a chain is that of its weakest ring…*
    - *We can use formal methods to strengthen a ring or to make sure that there are no weak ring*

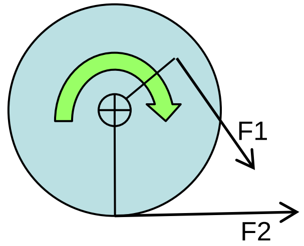
Thanks

- Thank you for your attention…
- And special thanks to NYCT / Thales

- For any extra information, contact:
  - Denis.sabatier@clearsy.com
  - Lilian.burdy@clearsy.com

Explicit, reviewed assumptions

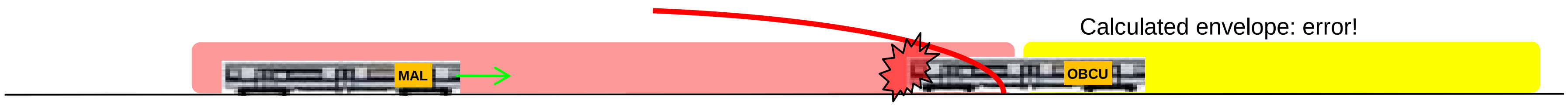
- Example: a “braked only” axle cannot spin
  - Braked only = no traction motor, spin = slipping faster than train speed
  - Useful for a tachometer axle…
  - But is it really true?
    - Should be **sub-provable** using laws (*Change of rotation speed x Inertia moment = Torques*)



    - Not true in all cases… Axle rotating fast, on a slippery rail, if train decelerates strongly
      - But cases of spinning are considered unrealistic
- Example of an apparently obvious assumption that needs Domain experts contact to judge…
  - A proof needs ALL assumptions explicitly
  - Explicitly formulated, the issue can be examined

An idea of how we prove more algebraic sub-properties

- Example: linking real wheel rotation to OBCU outputs
  - Needed for “train inside their envelope” property, example:



  - Assumptions:

R(t): actual wheel rotation → Wheel sensor: N

Using “physical” variables, i.e. infinite accuracy

t, N(t) –

    - At t, Rc may come from N dating back t-2Tc
    - Minimum and maximum R change during 2Tc: using assumptions about greatest train acceleration and such
  - Combining equations: we can prove t, Rc(t) – • ’ • • • • • • • • •
    - OK if uncertainty calculated by OBCU is greater than e’’
    - Proof steps: only simple rules (ex: a<b and b<c implies a<c; a<b and x>0 implies ax<bx…)