Lecture 2 - January 12

Introduction

Safety-Critical vs. Mission-Critical
Formal Methods, Industrial Standards
Verification vs. Validation
Model-Based Development

Consequence:

SCS MCS

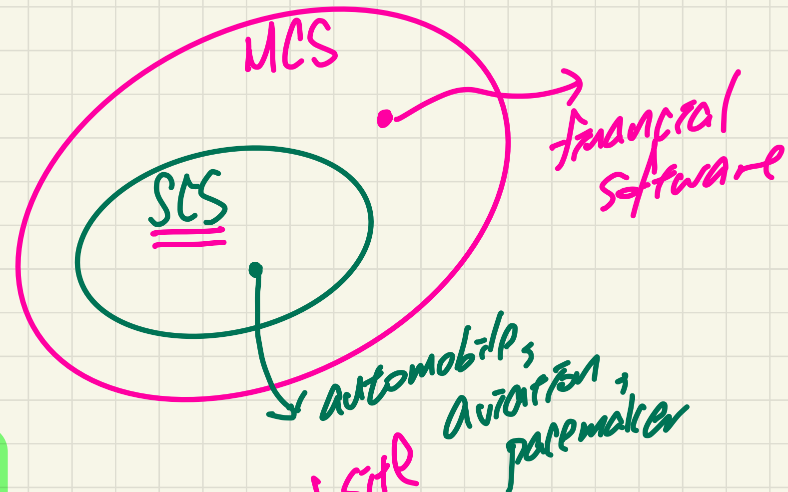
~~(1)~~ SCS ⇔ ≡ MCS

(2) **SCS ⇒ MCS**

(3) MCS ⇏ SCS → shopping website (financial software

MCS → mission

SCS → safety



Mission-Critical vs. Safety-Critical

Safety critical

When defining safety critical it is beneficial to look at the definition of each word independently. Safety typically refers to being free from danger, injury, or loss. In the commercial and military industries this applies most directly to human life. Critical refers to a task that must be successfully completed to ensure that a larger, more complex operation succeeds. Failure to complete this task compromises the integrity of the entire operation. Therefore a safety-critical application for an RTOS implies that execution failure or faulty execution by the operating system could result in injury or loss of human life.

Safety-critical systems demand software that has been developed using a well-defined, mature software development process focused on producing quality software. For this very reason

2342, 4315 (formal method)

the DO-178B specification was created. DO-178B defines the guidelines for development of aviation software in the USA. Developed by the Radio Technical Commission for Aeronautics (RTCA), the DO-178B standard is a set of guidelines for the production of software for airborne systems. There are multiple criticality levels for this software (A, B, C, D, and E).

These levels correspond to the consequences of a software failure:

(most)

- Level A is catastrophic
- Level B is hazardous/severe
- Level C is major
- Level D is minor
- Level E is no effect

(least)

SCS

MCS

Safety-critical software is typically DO-178B level A or B. At these higher levels of software criticality the software objectives defined by DO-178B must be reviewed by an independent party and undergo more rigorous testing. Typical safety-critical applications include both military and commercial flight, and engine controls.

Mission critical

A mission refers to an operation or task that is assigned by a higher authority. Therefore a mission-critical application for an RTOS implies that a failure by the operating system will prevent a task or operation from being performed, possibly preventing successful completion of the operation as a whole.

Mission-critical systems must also be developed using well-defined, mature software development processes. Therefore they also are subjected to the rigors of DO-178B. However, unlike safety-critical applications, mission-critical software is typically DO-178B level C or D. Mission-critical systems only need to meet the lower criticality levels set forth by the DO-178B specification.

Generally mission-critical applications include navigation systems, avionics display systems, and mission command and control.

Source: http://pdf.cloud.opensystemsmedia.com/advancedtca-systems.com/SBS.Jan04.pdf

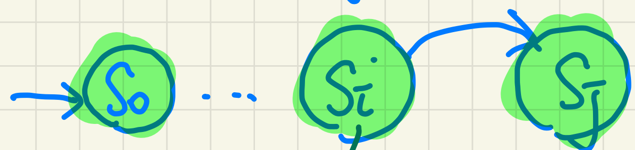
Safety Property

Safety **Property** → predicates

Safety Property → invariant property.

↳ Every possible state of the system should satisfy it.

reactive system is running all the time (infinite # of state)



assume S_i safe

= prove S_j safe. (S_{i+1})

process

– Are we building the product right? (not right e.g. without testing)

right → property?

↓ implicit assumption: given what to build

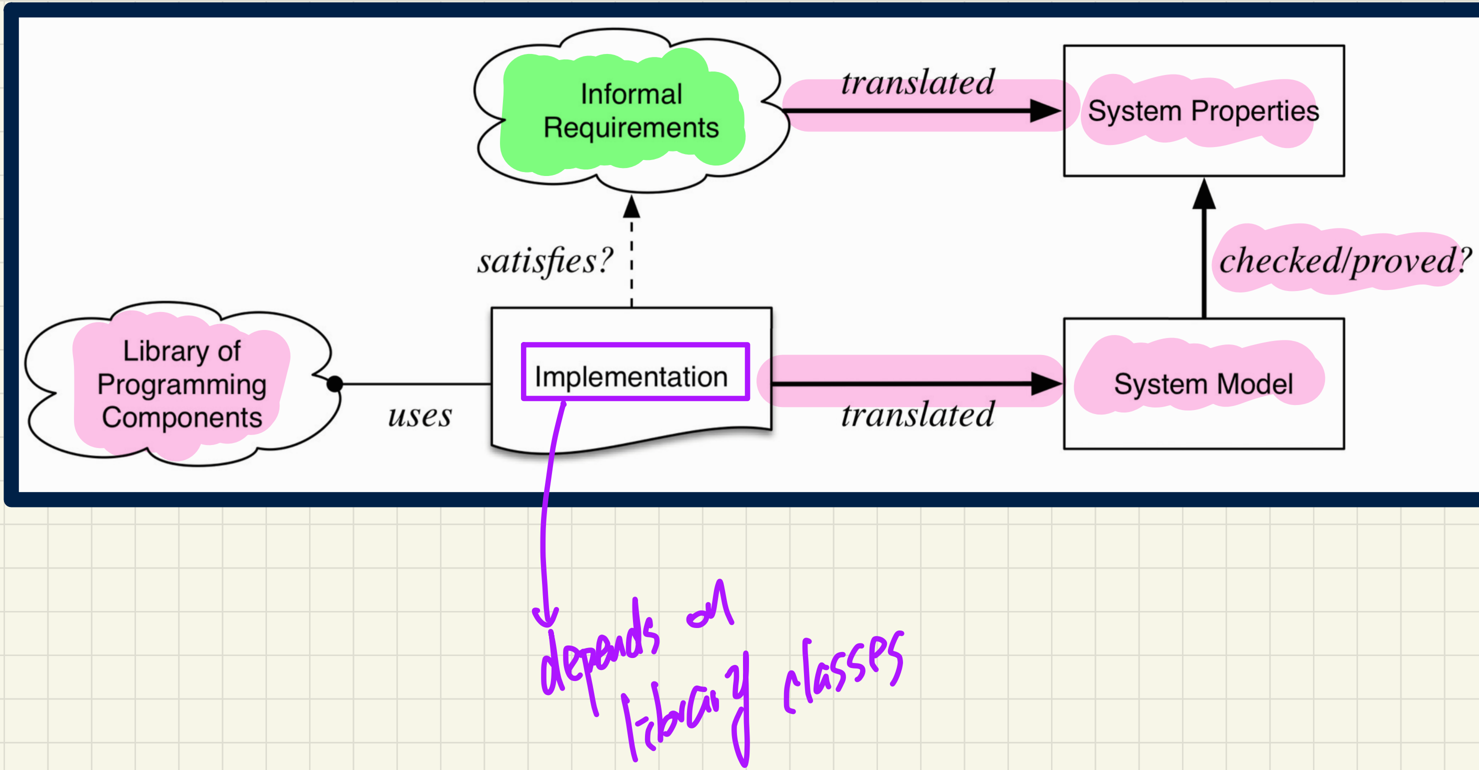
goal

– Are we building the right product?

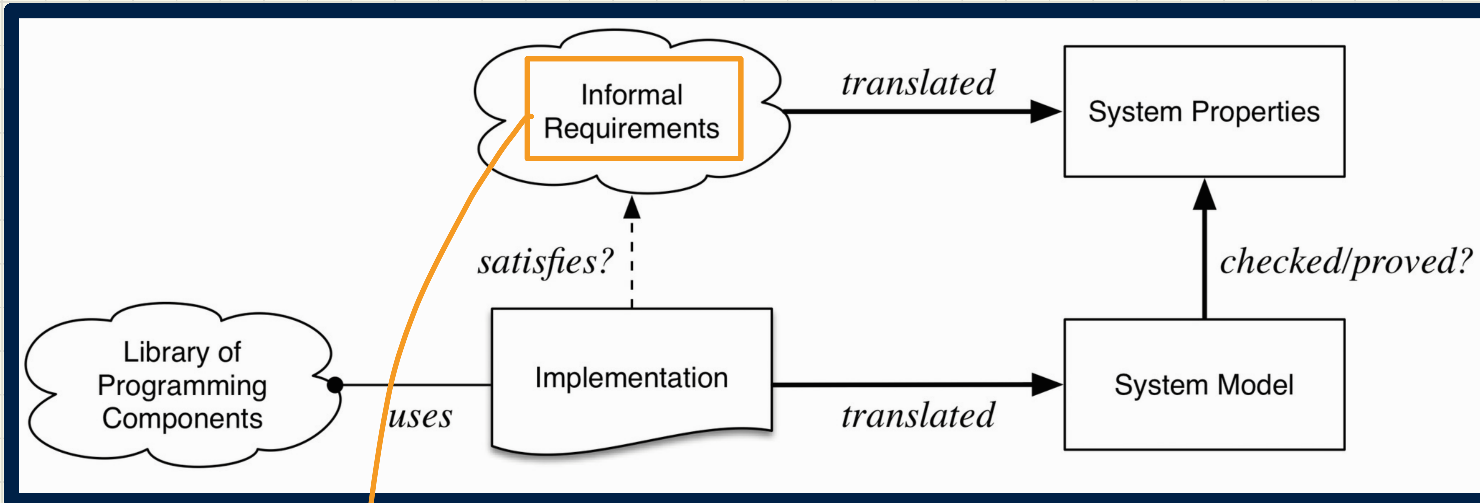
right product → does it deviate from the customer's intended req.

→ EECS4312.

Building the product right?

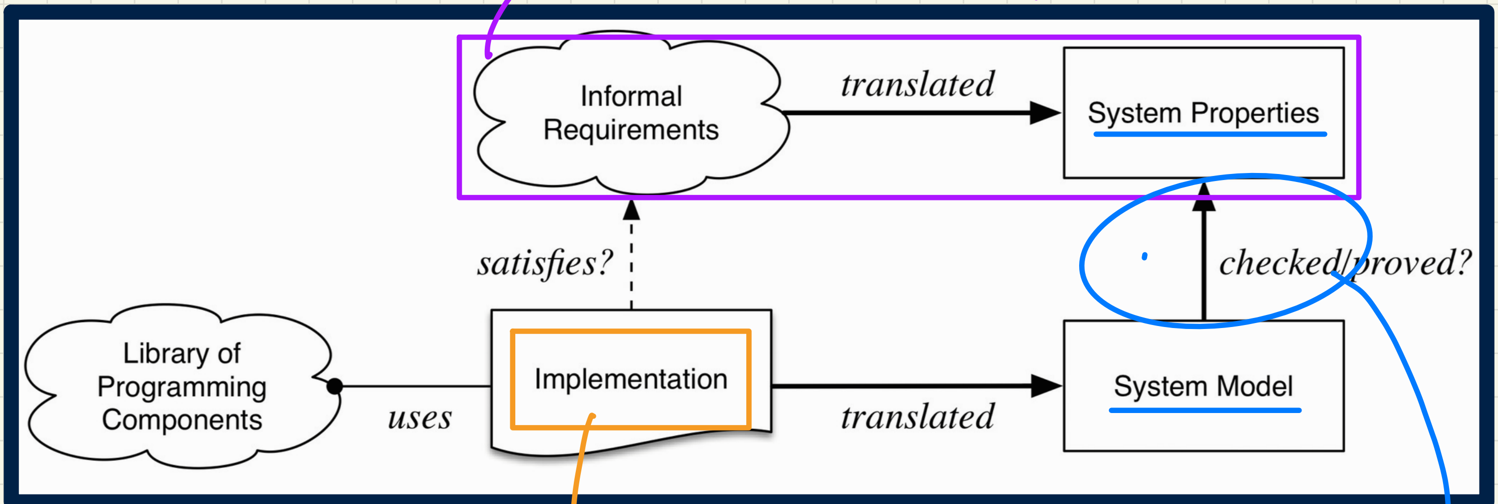


Building the right product?



does the set of requirements really accurately represent the customer's needs.

given req written in NAT, formulate it in predicates



given the NAT descriptions of a strategy, formulate it as abstract state machines

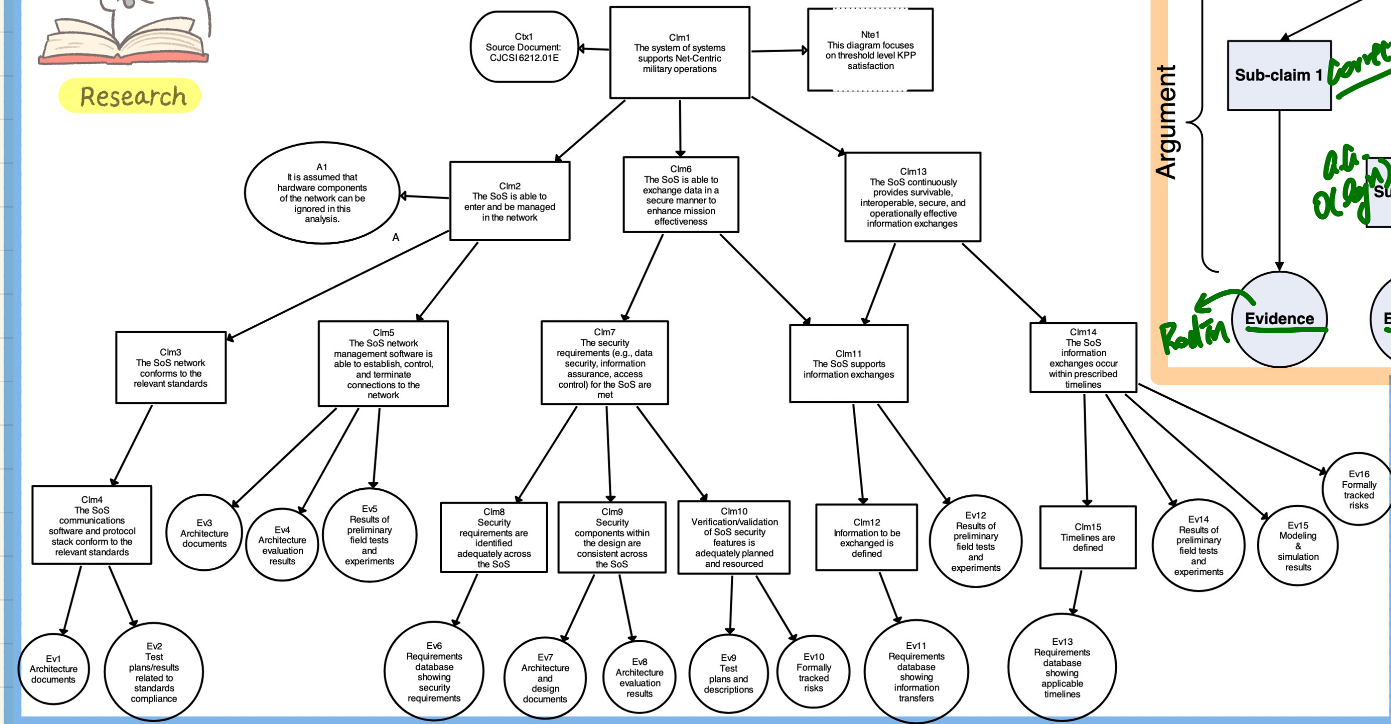
1. constants
2. variables
3. axioms

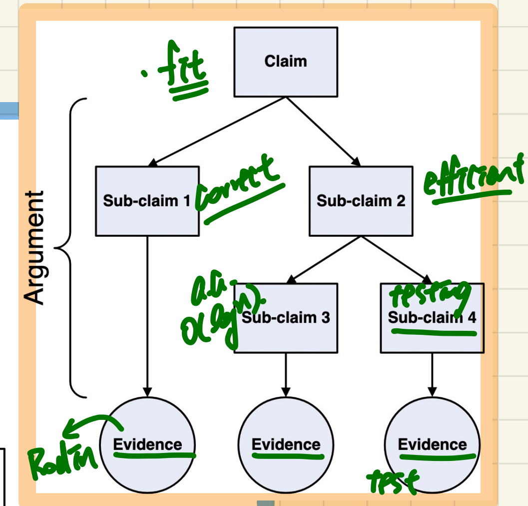
proof obligations

1. prove -
2. generat

Certifying Systems: Assurance Cases

Research on "Assurance Cases" if interested!





Source: https://resources.sei.cmu.edu/asset_files/whitepaper/2009_019_001_29066.pdf

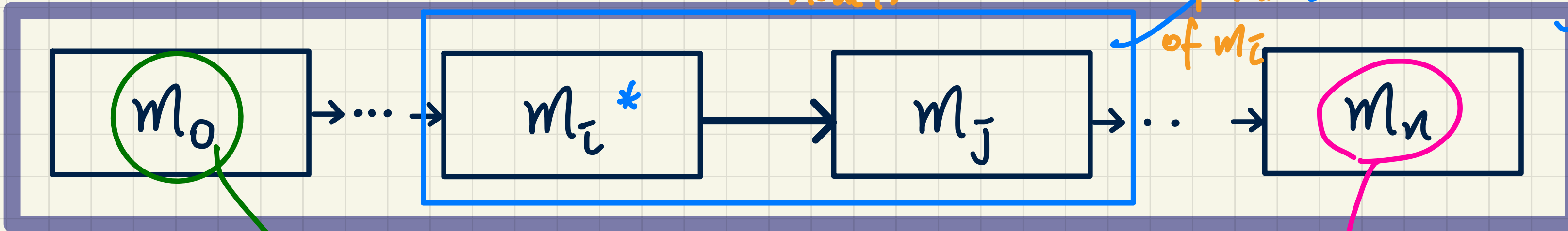
Correct by Construction

2. Instead, distribute different properties to different models

3. Prove m_j is a refinement of m_i

$n+1$ models formal

m_i is more abstract than m_j



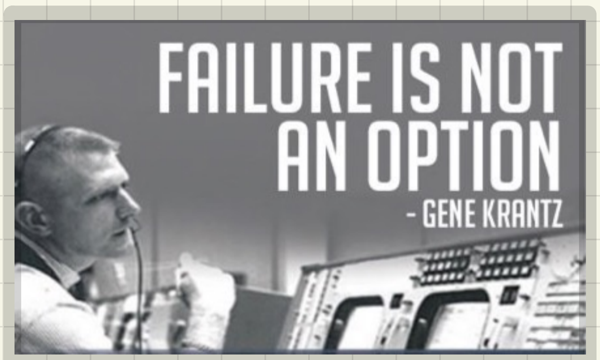
all models describe the same system

Initial, simplest, most abstract model.

1. having a single model and proving all properties on it is infeasible

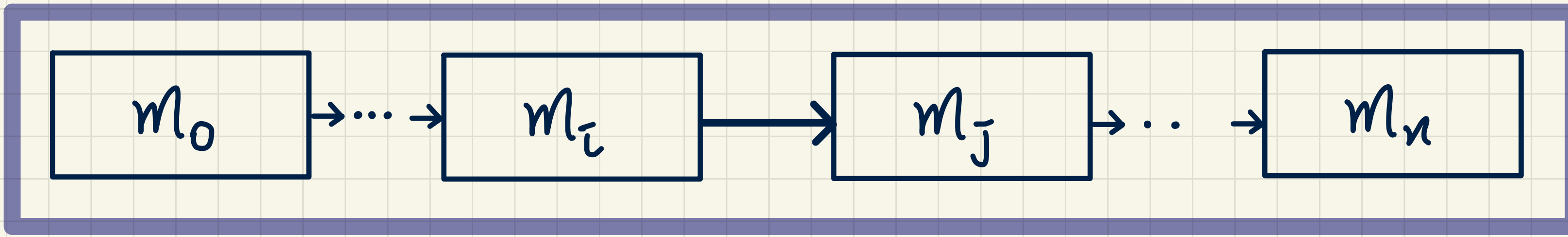
final, most sophisticated, most concrete model

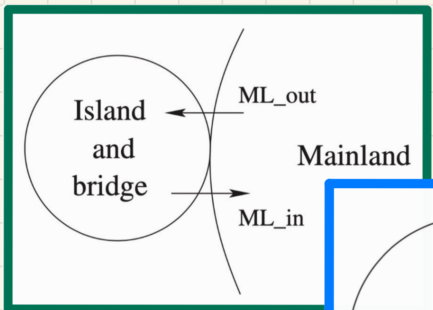
closest for translating into code.



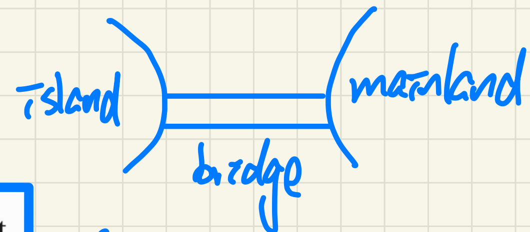
Source: https://audiobookstore.com/audiobooks/failure-is-not-an-option-1.aspx

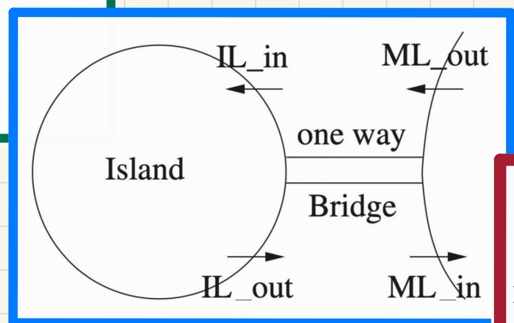
Correct by Construction: Bridge Controller System



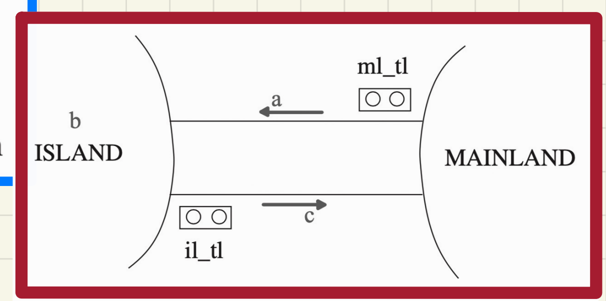


m0



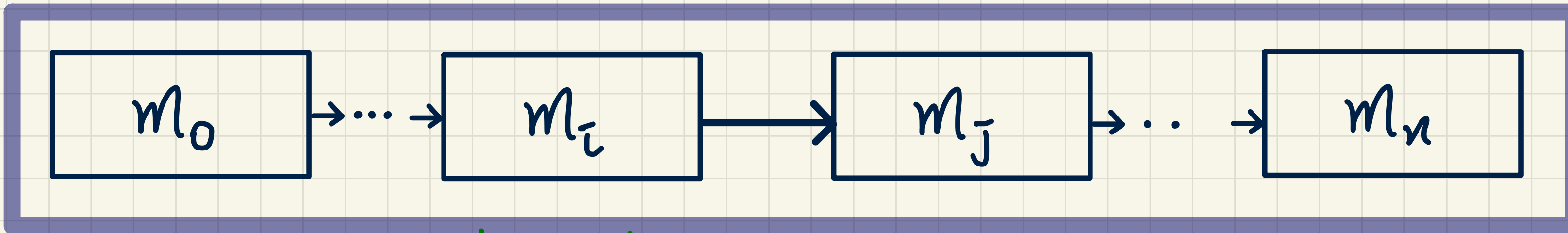


m1



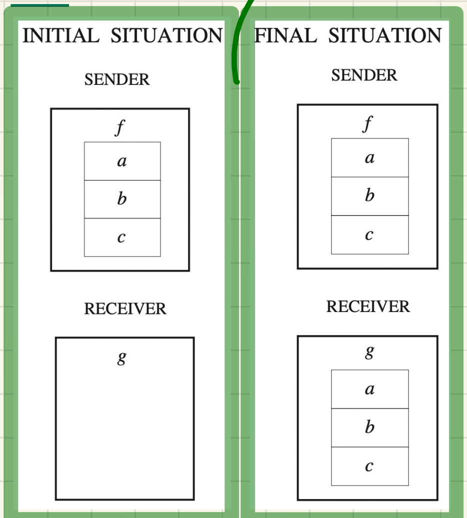
m2

Correct by Construction: File Transfer Protocol



abstracted away delay in sending (instantaneous transfer)

m0



m1: more concrete than m0