



Refinement – basic concepts and ideas

September 12, 2008



Objectives for the lectures on refinement

- **The two lectures on refinement aim to**
 - **to motivate and explain a basic apparatus to define and relate the notions of refinement**
 - **this includes**
 - **representing executions by traces**
 - **explaining the significance of a notion of observation**
 - **outlining the assumption-guarantee paradigm**
 - **introduce and related the following notions of refinement**
 - **supplementing**
 - **narrowing**
 - **detailing**
 - **property refinement**
 - **interface refinement**
 - **Illustrate the use of these notions of refinement**
 - **the interplay between specification and refinement**



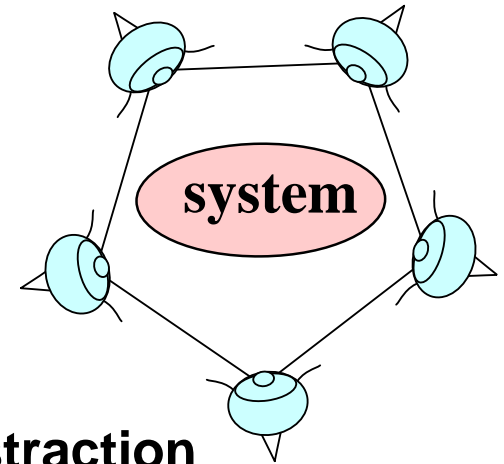
The role of refinement

- **System development makes use of refinement as a means to check and document incremental steps aiming to**
 - **reduce the set of legal implementations**
 - **introduce error handling**
 - **introduce time constraints**
 - **introduce finer granularity of interaction and execution**
 - **introduce implementation dependent data types**
 - **introduce implementation oriented communication protocols**
 - **introduce constraints on unlimited resources**
 - **extend the input domain**



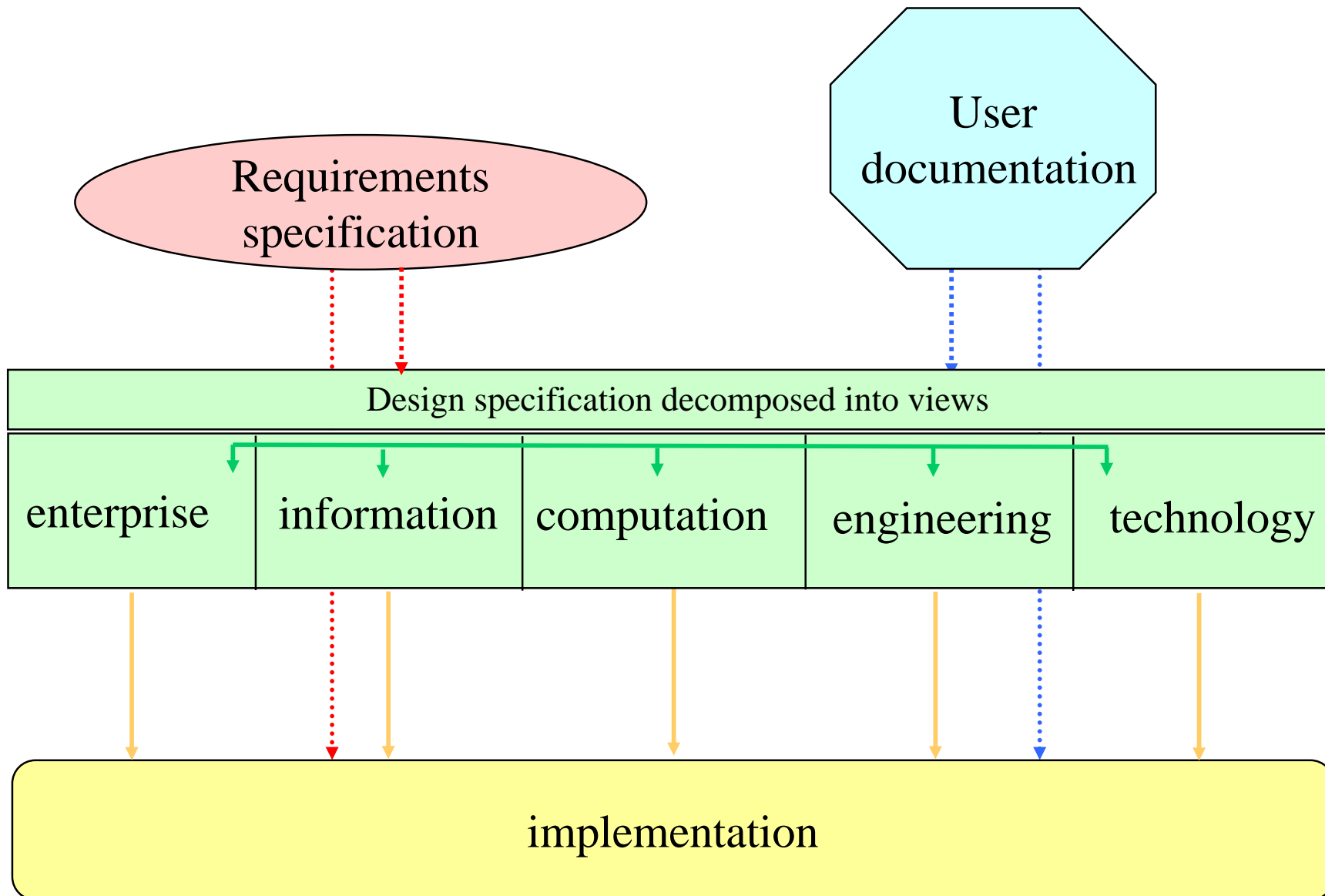
Why refinement is important

- **Systems of today are large and complex – abstraction is a necessary means to**
 - explain what the systems do
 - explain how the systems are built
 - distinguish the essentials from the inessential
 - decompose large and complicated aspects into small more easily understandable entities
 - extract specialized system views
- **Formal documentation gives new possibilities**
- **Refinement**
 - relates system descriptions at different levels of abstraction
 - connects and relates different system views
 - provides a foundation for verifications and validations





Why refinement must be documented





Documenting refinement

- **Precision is just as important when we document refinements as when we write specifications**
- **Refinements can be documented using standard specification languages**
 - in INF 5150 we will use UML for this purpose
- **Formal documentation of refinements facilitates integrated analysis, validation, testing and verification**



Three main concepts of language theory

- **Syntax**

- The relationship between symbols or groups of symbols independent of content, usage and interpretation

- **Semantics**

- The rules and conventions that are necessary to interpret and understand the content of language constructs

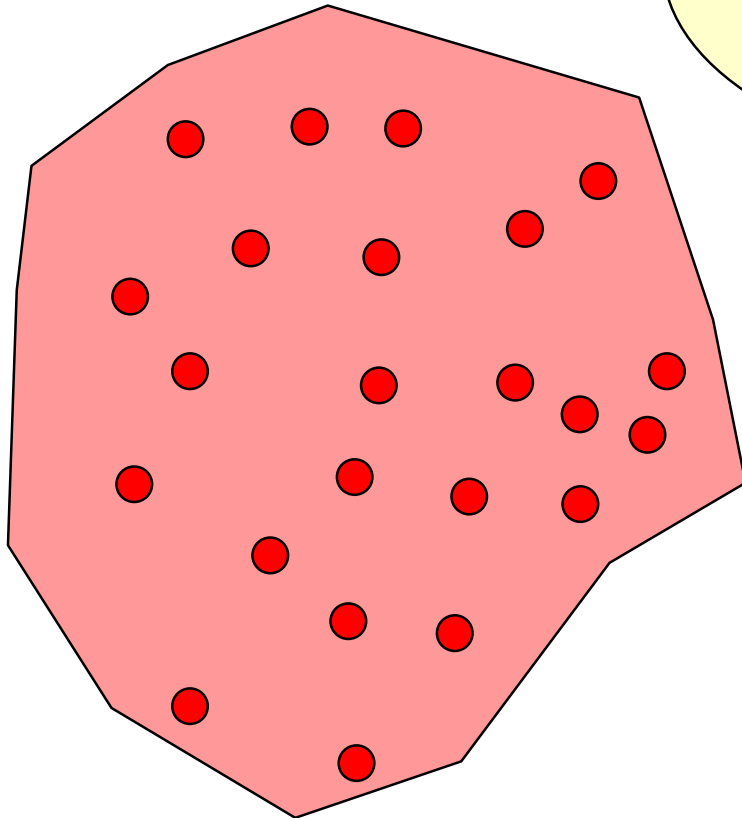
- **Pragmatics**

- The study of the relationship between symbols or groups of symbols and their interpretation and usage



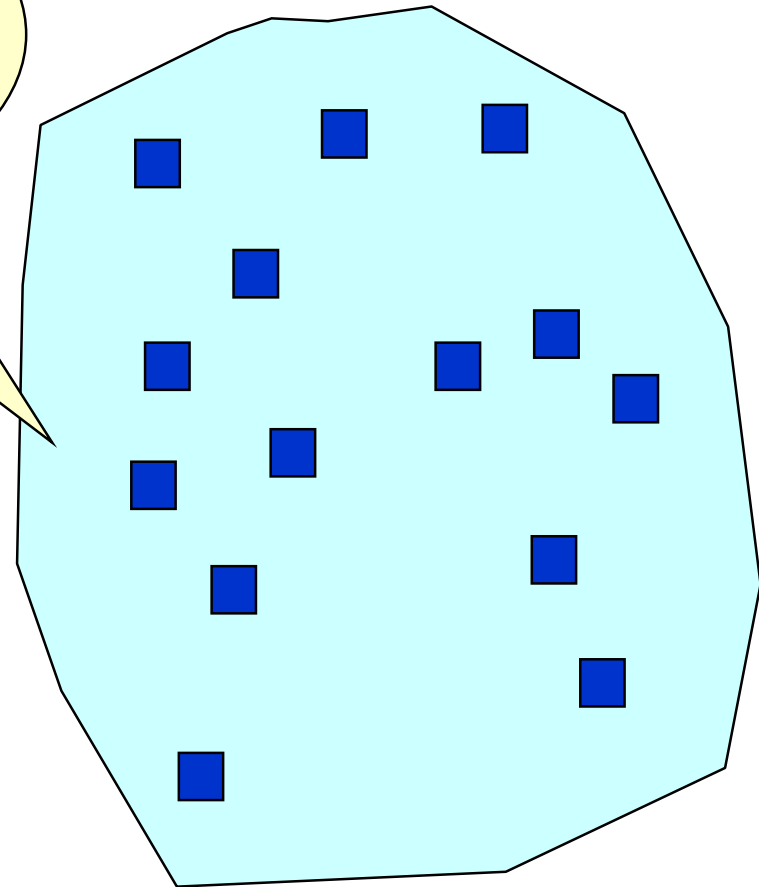
Semantic relation

Set of syntactically correct expressions in the language to be explained



What does it mean that a language is well-understood?

Set of syntactically correct expressions in a language that is well-understood



Semantic relation



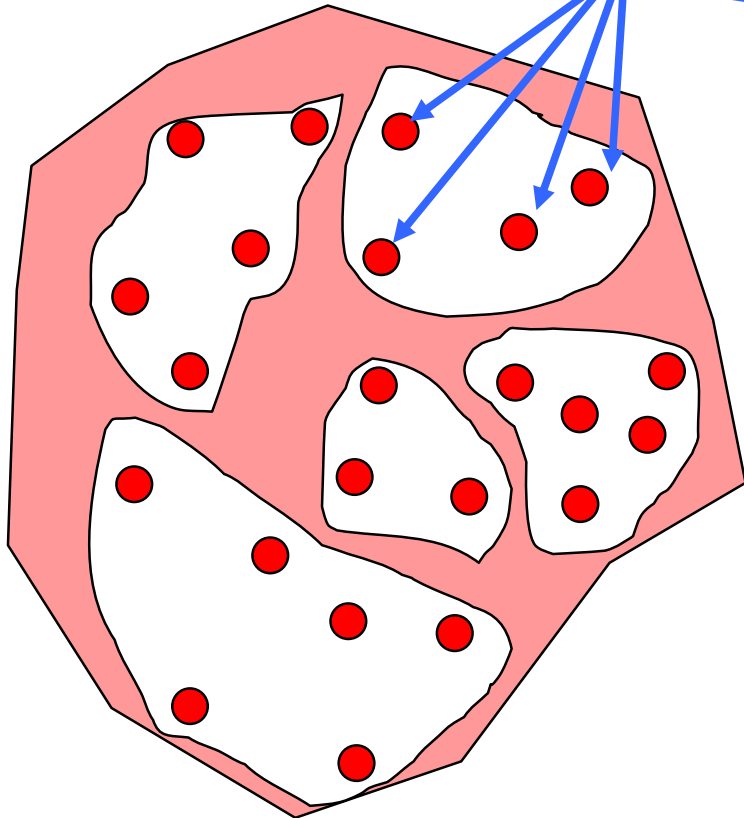
Relates expressions that need interpretation to expressions that are well-understood



The need for a notion of observation

- A semantic relation will define an equivalence relation on the language that should be understood

Of the same meaning



For a specification language these are defined with respect to a notion of observation



Definition of a notion of observation

- May observe only external behavior
- May observe any potential behavior
- May observe time with respect to a global clock
- May observe safety properties
 - Always falsified by a partial execution
- May observe liveness
 - Falsified only by complete executions



May our notion of observation be implemented by a human being?



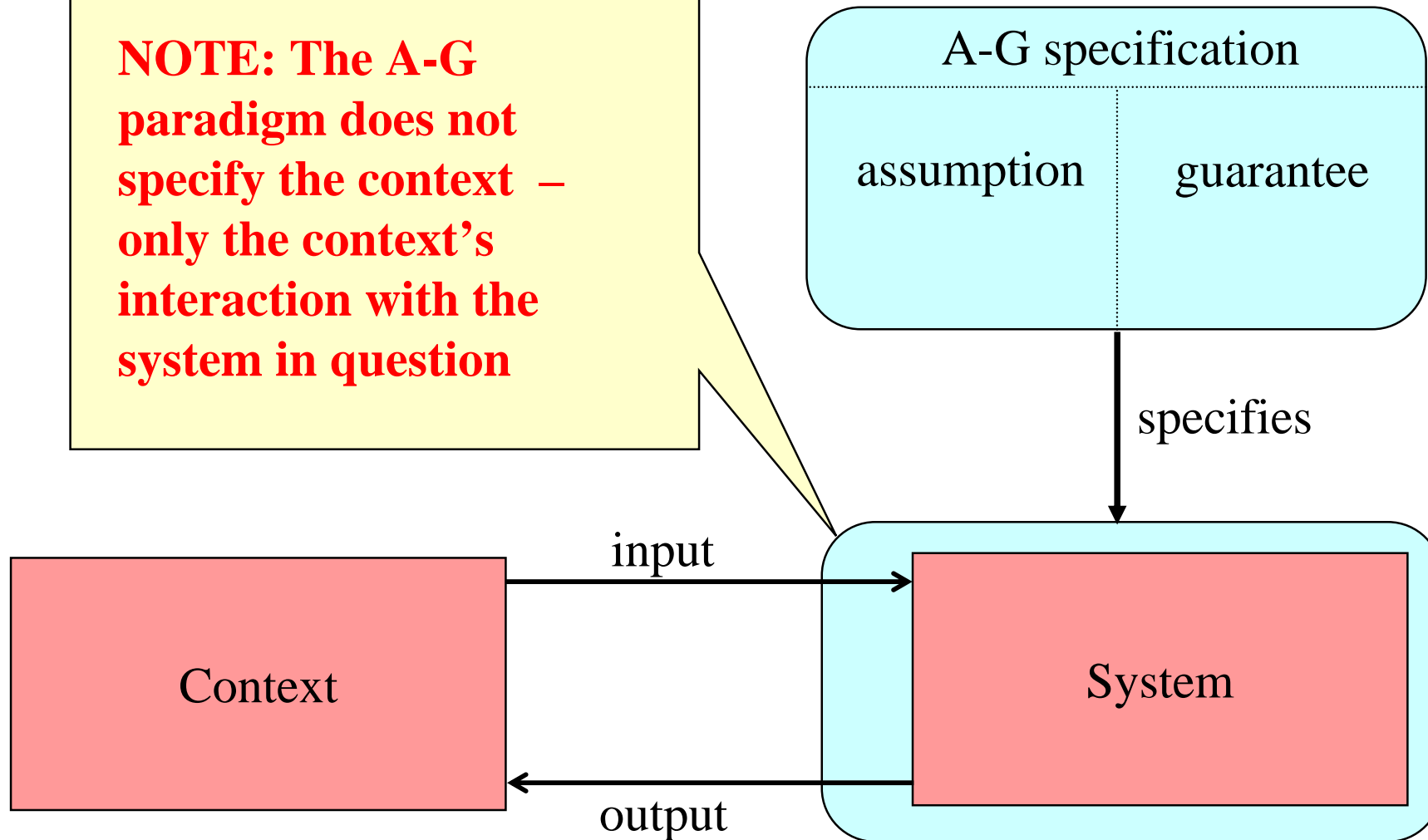
Assumption-guarantee paradigm

- **Well-known specification technique to facilitate modularity**
 - appeared first with pre-post specifications in the 60ies
 - since then taken further and adapted in many directions
 - referred to as: pre-post, rely-guarantee, assumption-commitment, assumption-guarantee, contracts, goal-means-task
- **Motivation:**
 - The behavior of a system component depends on the context it is executed in
 - Not all contexts are equally interesting
- **The assumption describes expected input**
 - The input that can be produced by the relevant contexts
- **The guarantee describes the output the specified component is obligated to produce as long as the context behaves in accordance with the assumption**



Graphical illustration of the A-G paradigm

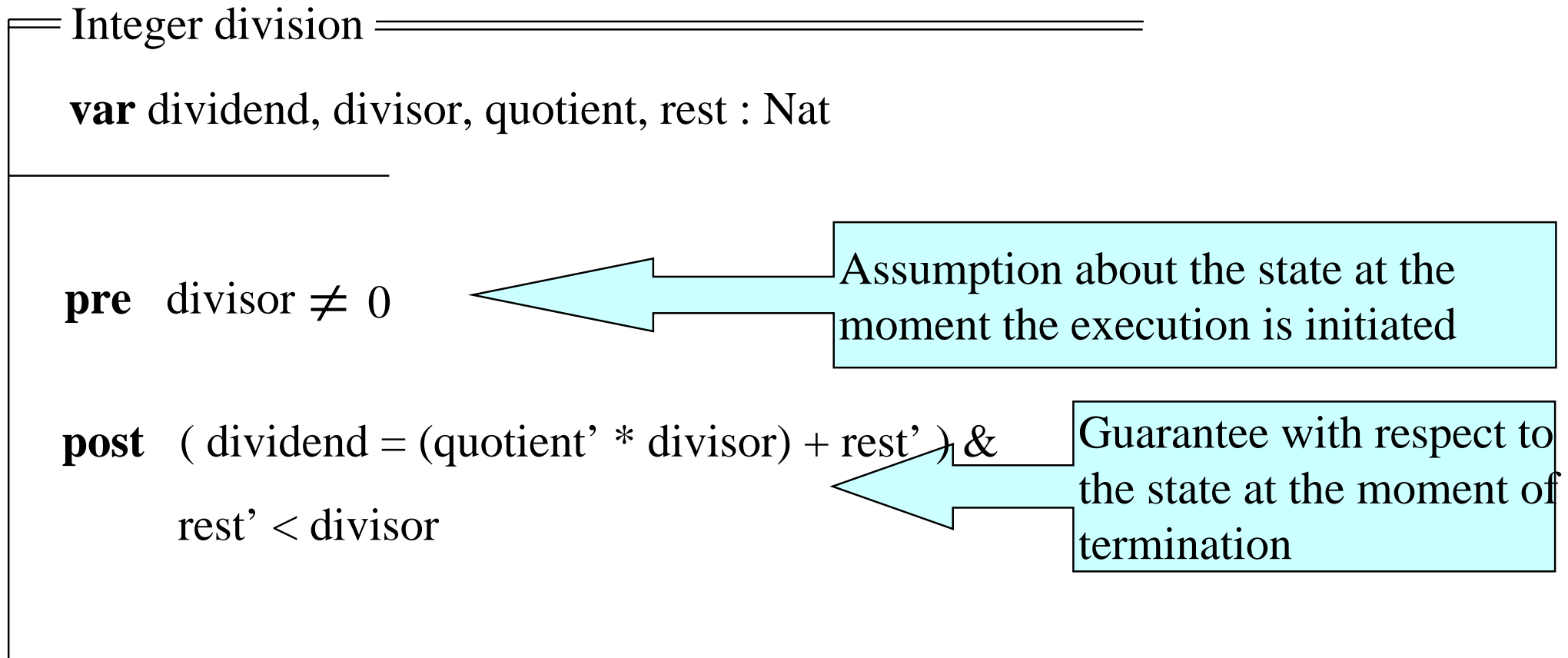
NOTE: The A-G paradigm does not specify the context – only the context's interaction with the system in question





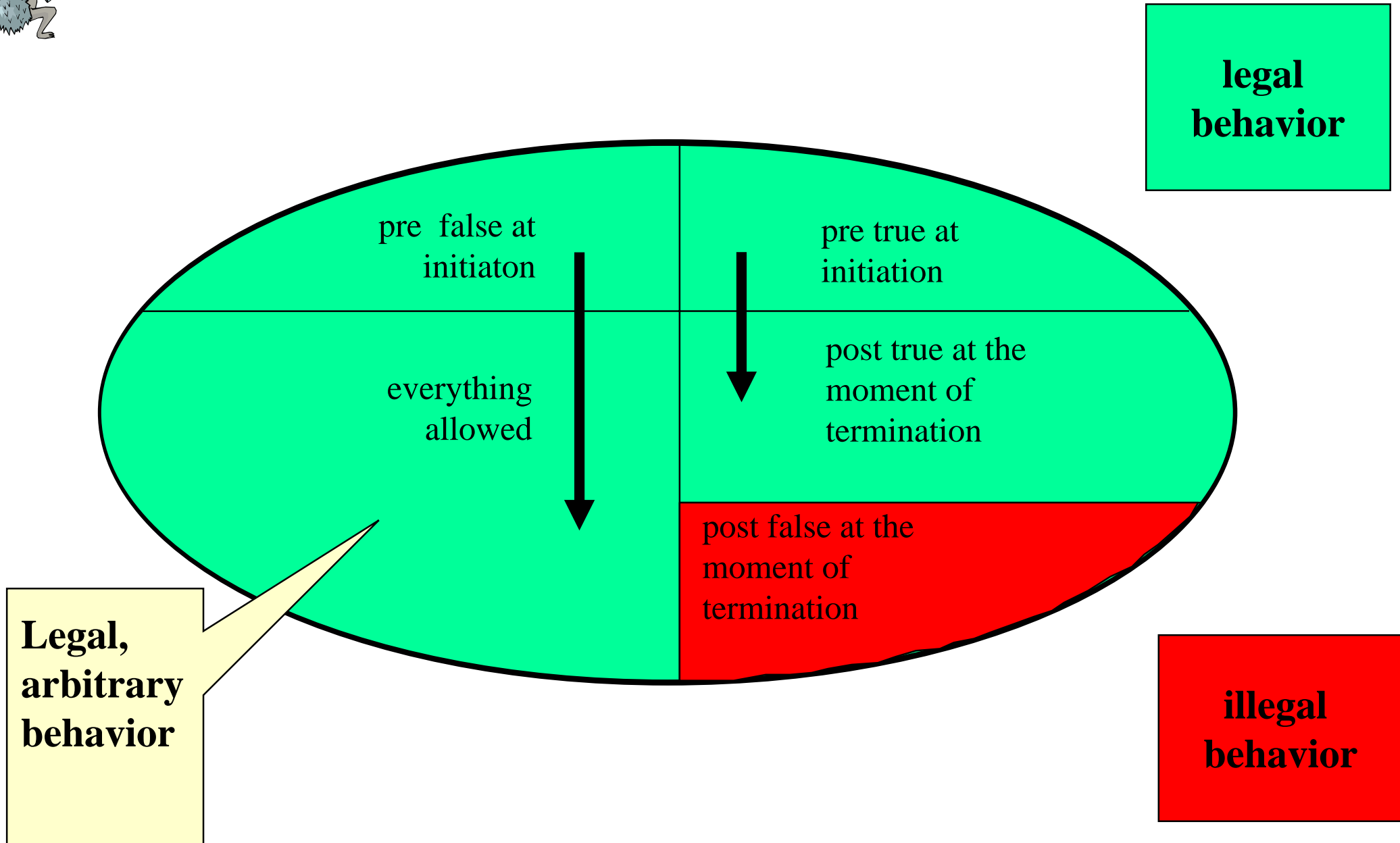
Pre-post specifications

Pre-post specifications are based on the assumption-guarantee paradigm





Semantics for pre-post specification



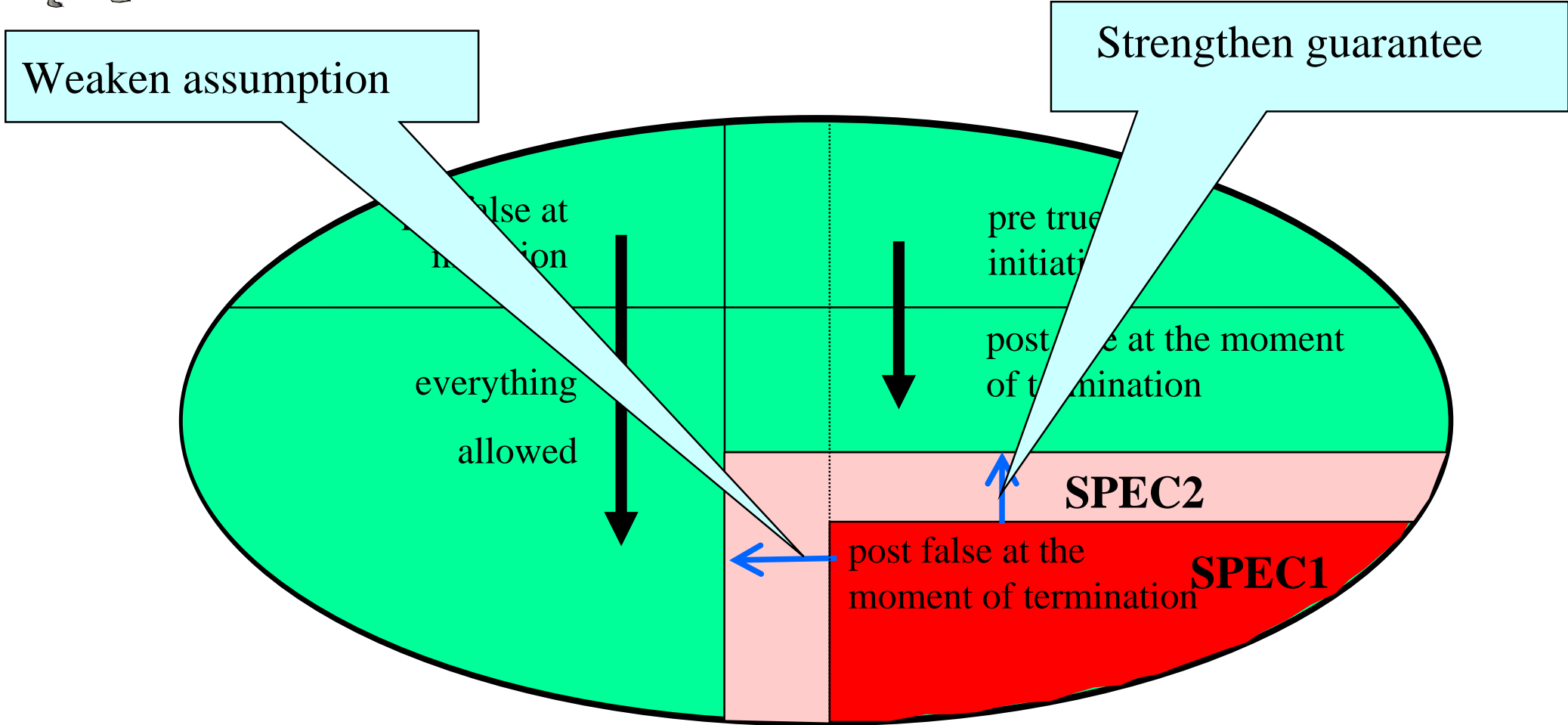


Semantics for pre-post specifications

- A state is a function from the set of variable names to type correct values
 - e.g.,
 - $\text{state}(\text{dividend})=600$
 - $\text{state}(\text{divisor})=6$
 - $\text{state}(\text{quotient})=100$
- A state S satisfies a pre-condition if the condition evaluates to true when for any variable v
 - $S(v)$ is substituted for each occurrence of v in the condition
- A pair of states (S, S') satisfies a post-condition if the condition evaluates to true when for any variable v
 - $S(v)$ is substituted for each occurrence of v in the condition
 - $S'(v)$ is substituted for each occurrence of v' in the condition
- The semantics of a pre-post specification is the set of all pairs of states (S, S') such that
 - S satisfies pre and (S, S') satisfies post, or
 - S does not satisfy pre
 - In other words: $\text{pre}(S) \Rightarrow \text{post}(S, S')$
- We use $[\text{SPEC}]$ to denote the semantics of the pre-post specification SPEC



Property refinement for pre-post specifications



SPEC2 is a property refinement of SPEK1
if [SPEC2] is contained in [SPEC1]
This corresponds to logical implication



Weakening the pre-condition (assumption)

Integer division

var dividend, divisor, quotient, rest : Nat

pre true

post

if divisor \neq 0 then

(dividend = (quotient' * divisor) + rest') & rest' < divisor

else quotient' = 0



Strengthening the post-condition (guarantee)

Integer division

var dividend, divisor, quotient, rest : Nat

pre divisor \neq 0

post (dividend = (quotient' * divisor) + rest') &

rest' < divisor & dividend' = dividend &

divisor' = divisor



The shortcomings of pre-post specifications

- Pre-condition describes only what the context may do before the operation is started up – not what the context may do during the execution of the operation

```
– pre { divisor ≠ 0 }
– <quotient := 0>
– while <dividend > divisor> do
    – <dividend := dividend - divisor>
    – <quotient := quotient + 1>
– od
– <rest:=dividend>
– post { ( dividend' = (quotient * divisor') + rest ) & rest < divisor' }
```

Points of
interference

- “<Statement>” denotes that “statement” is atomic (in the meaning that the context cannot interfere with its execution)



Traces

- **Traces are used to represent system runs matematically**
- **In the literature there are many different kinds of traces**
- **INF 5150 traces are sequences of events**

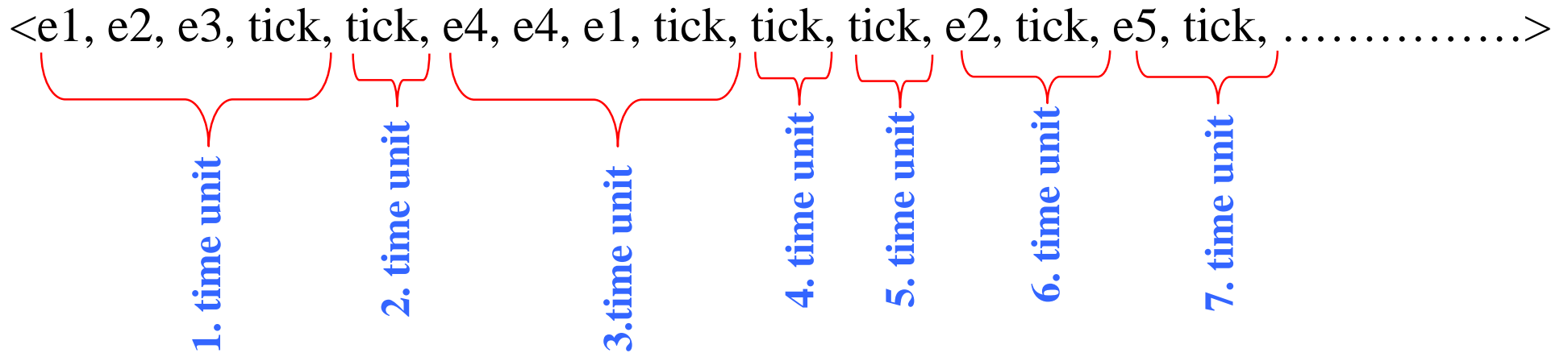
$\langle e1, e2, e3, e4, e4, e1, e2, e5, \dots \dots \dots \rangle$

- **Events are instantaneous**
- **The number of events in a trace may be finite**
 - may be caused by: termination, deadlock, infinite waiting, system crash
- **The number of events in a trace may be infinite**
 - May be cause by: nontermination, livelock, nontermination by purpose



Traces with time ticks

- **Traces are infinite sequences of events and time ticks**



- **Events and time ticks are instantaneous**
- **Each trace contains infinitely time ticks**
 - this reflects that time never halts
- **The number of events in a trace may be finite**



Traces with time stamps

- **Each element of the trace is a pair of an event and a time stamp**

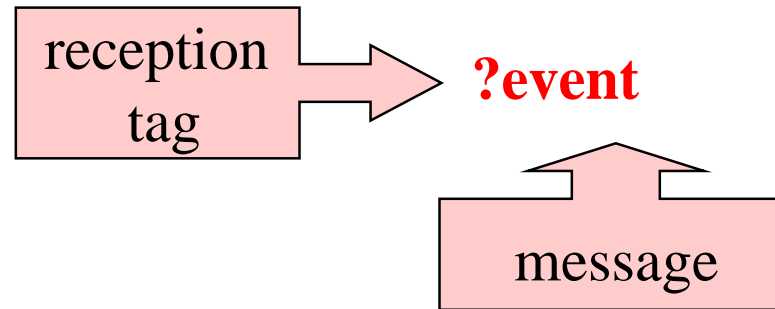
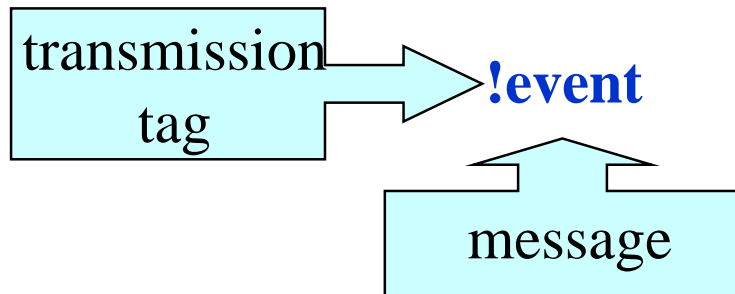
$\langle e1:t1, e2:t2, e3:t3, e4:t4, e4:t5, e1:t6, e2:t7, e5:t8, \dots \rangle$

- **The elements are ordered according to their time stamps**
 - $(t1 \leq t2 \leq t3 \dots)$
- **Events are instantaneous**
- **A trace is either finite or there is for every point in time k an element $n:t$ with time stamp t such that $k < t$**
 - this is necessary to avoid Zenon's paradox



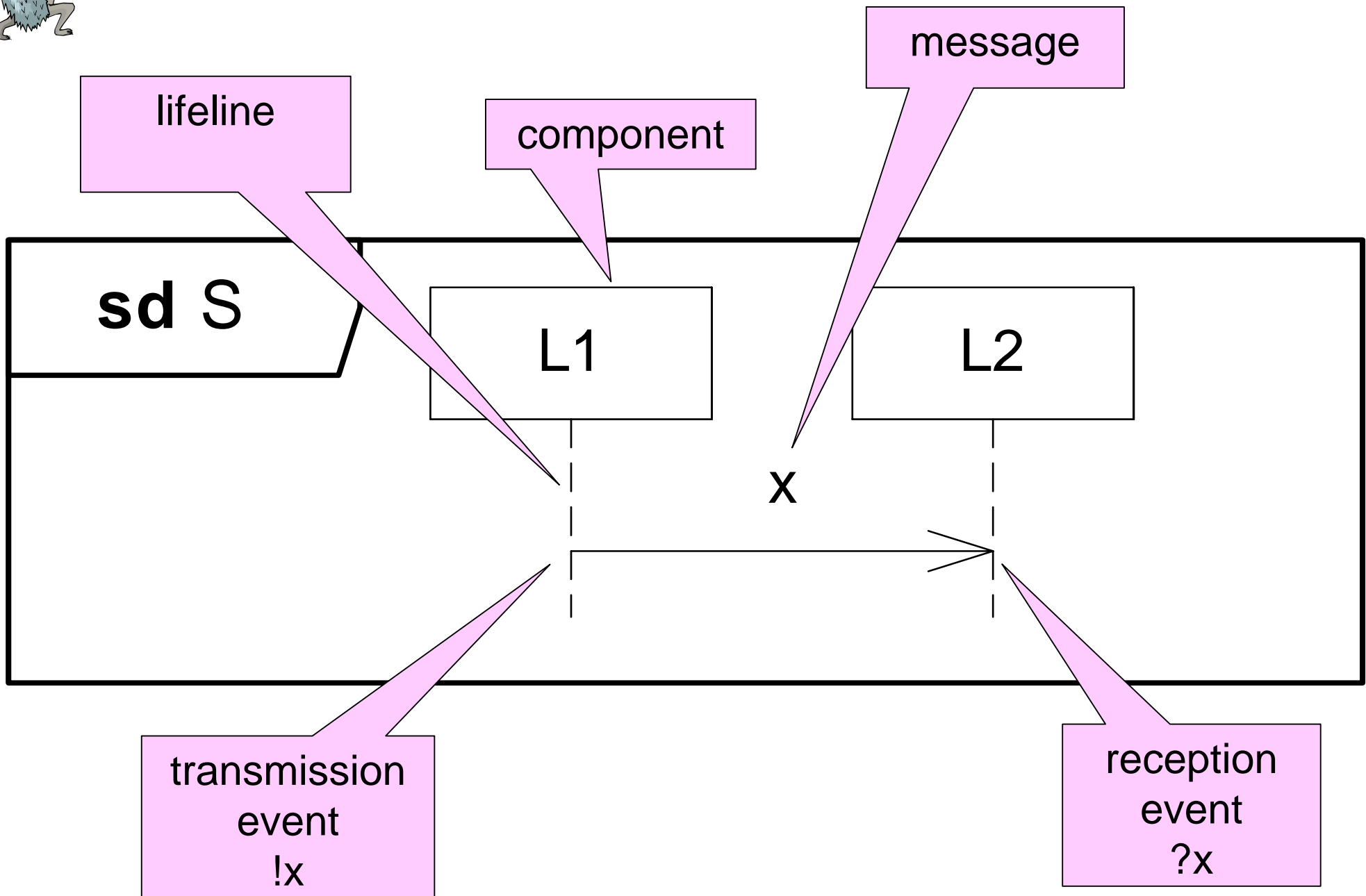
Traces for sequence diagrams

- Two kinds of events:
 - transmission events
 - reception events





Sequence diagram





Causality and weak sequencing

- **Causality:**

- a message can never be received before it has been transmitted
- the transmission event for a message is therefore always ordered before the reception event for the same message

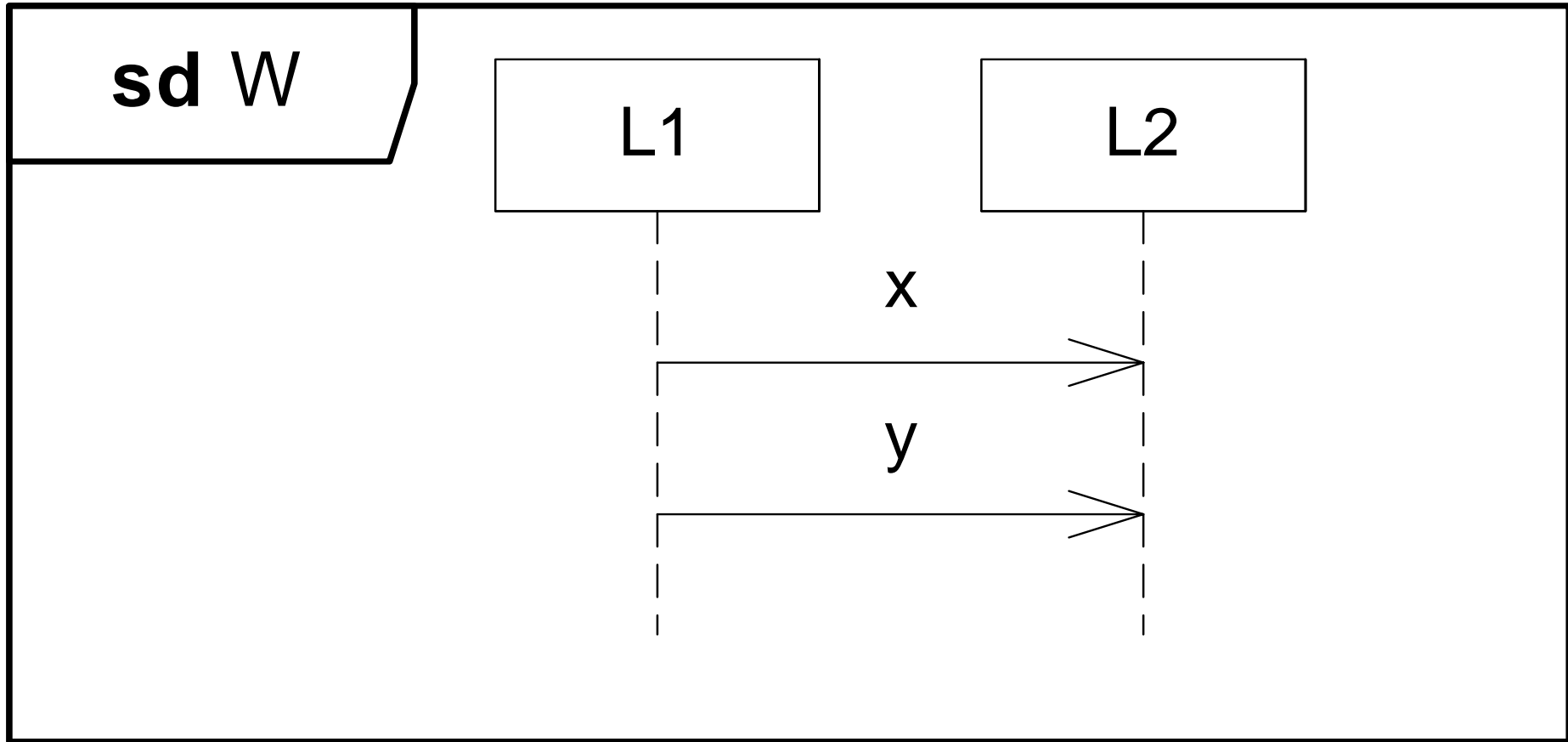
- **Weak sequencing:**

- events from the same lifeline are ordered in the trace in the same order as on the lifeline

- **NOTE: A sequence diagram will normally be represented by more than one trace, and in some cases by infinitely many traces**



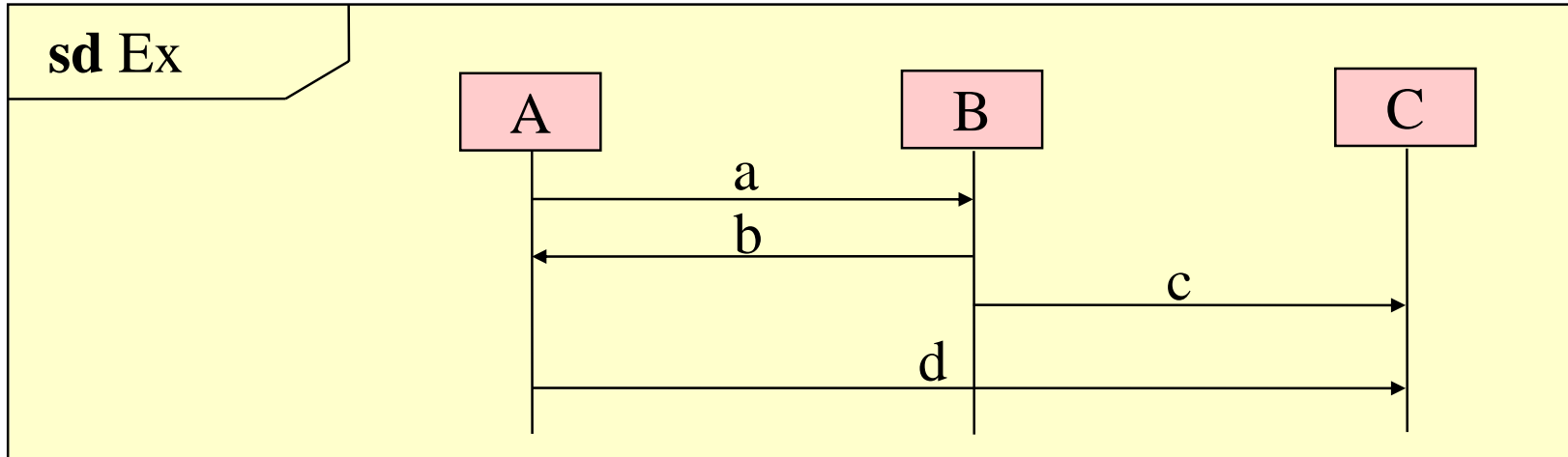
Weak sequencing



$\langle !x, ?x, !y, ?y \rangle$
 $\langle !x, !y, ?x, ?y \rangle$



Example



There are six possible traces if time information is ignored:

- <!a, ?a, !b, ?b, !c, ?c, !d, ?d>
- <!a, ?a, !b, ?b, !c, !d, ?c, ?d>
- <!a, ?a, !b, ?b, !d, !c, ?c, ?d>
- <!a, ?a, !b, !c, ?b, ?c, !d, ?d>
- <!a, ?a, !b, !c, ?b, !d, ?c, ?d>
- <!a, ?a, !b, !c, ?c, ?b, !d, ?d>

Each of these corresponds to infinitely many traces with time information

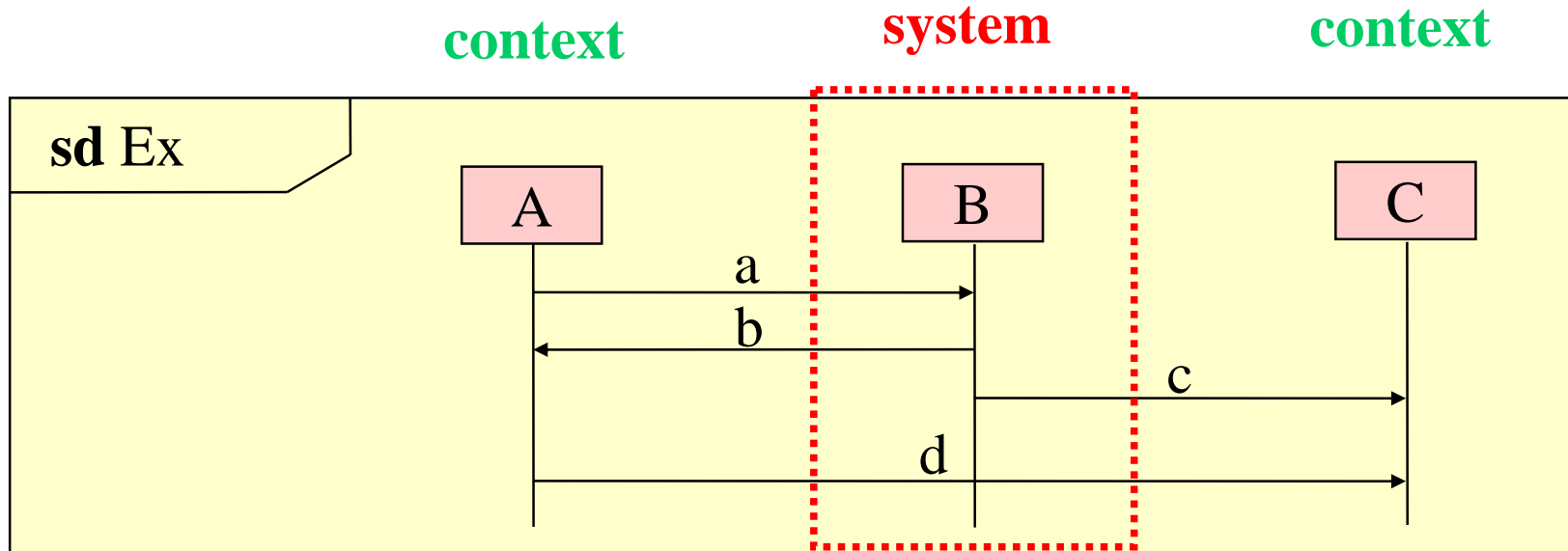


External behavior

- **Property refinement in the classical sense takes only external behavior into consideration**
- **We therefore need a well-defined interface between**
 - the component to be refined, and
 - its context



Projection on B



System has one possible external trace:

$\langle ?a, !b, !c \rangle$

This trace is an abstraction of infinitely many traces with time information



– STAIRS –
**Steps to Analyze Sequence Diagrams
with Refinement Semantics**



Motivation

- **Make use of classical refinement theory in a practical UML setting**
 - From theory to practice, and not the other way around
- **We aim to explain how classical theory of refinement can be used to refine specifications expressed with the help**
- **Sequence diagrams can be used to explain other kinds of UML diagrams**
- **By defining refinement for sequence diagrams we implicitly define refinement for the UML as a whole**



Requirements to STAIRS

- **Should support specification of potential behavior**
 - Means to abstraction
- **Should support specification of mandatory behavior**
 - Important within the security domain
- **Should support specification of negative behavior in addition to positive behavior**
- **Should support classical refinement theory**
- **Should formalize incremental system development**
- **Should facilitate modular analysis, verification and testing**



Next lecture on refinement – September 19

- **Example based introduction to STAIRS**
- **Semantics of sequence diagrams**
- **Refinement in STAIRS**
 - **Supplementing**
 - **Narrowing**
 - **Detailing**
- **Relation to pre-post**