

# System-level modelling with Event-B

B Dissemination Day, GRACE, Tokyo

Michael Butler

[users.ecs.soton.ac.uk/mjb](mailto:users.ecs.soton.ac.uk/mjb)

[www.event-b.org](http://www.event-b.org)

[www.deploy-project.eu](http://www.deploy-project.eu)

School of Electronics and Computer Science  
University of Southampton, UK



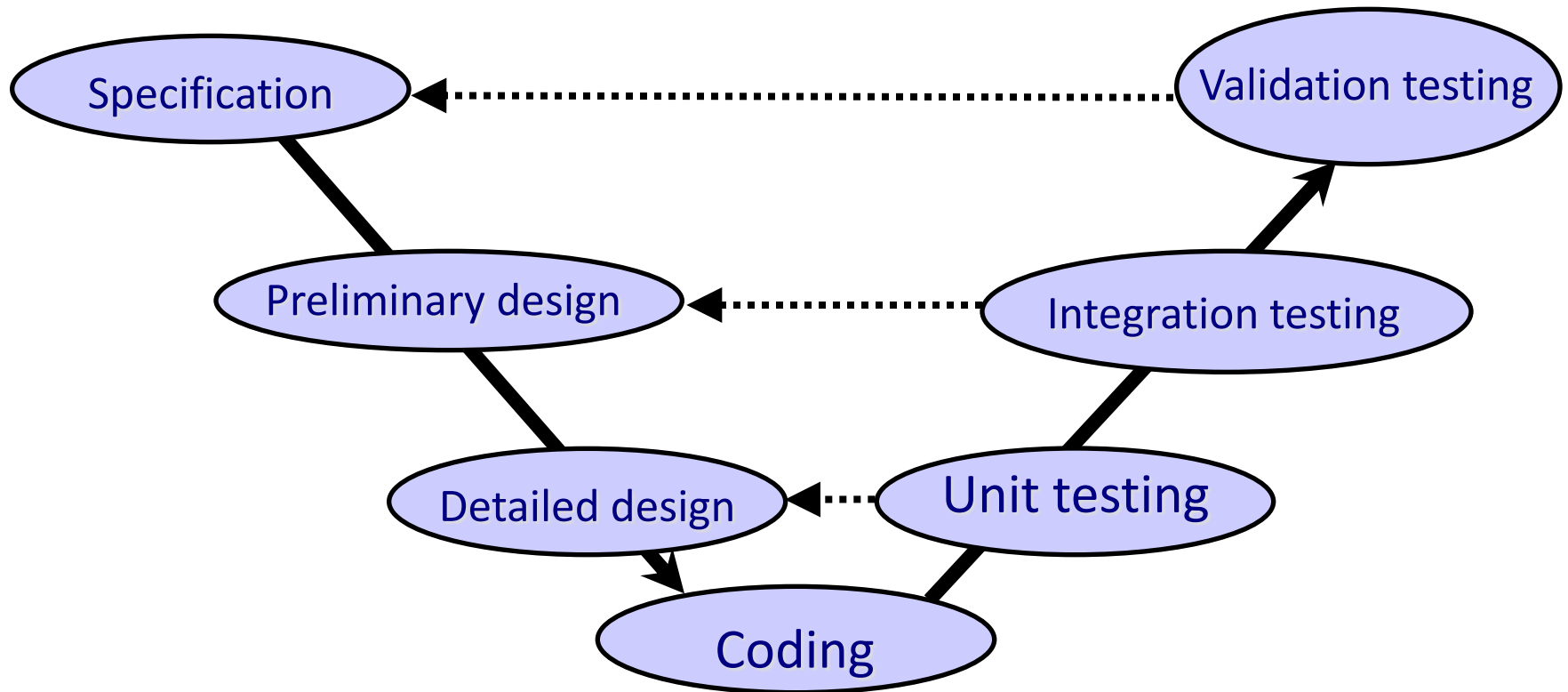
# Contents

- Motivation
- Event-B overview
- *Rational design* with Event-B:
  - abstraction
  - refinement
  - proof and mechanical analysis
- *Decomposition* structures
- Take-home messages

# System level

- Examples of systems:
  - Train signalling system
  - Mechanical press system
  - Access control system
  - Air traffic information system
  - Electronic purse system
  - Distributed database system
  - Cruise control system
  - ...
- System level reasoning:
  - Involves abstractions of *overall* system not just software components

# What's wrong with the V model?



Many errors are introduced early but detected late – such errors are expensive to fix.

# Why is it difficult to detect errors?

- Lack of precision
  - ambiguities
  - inconsistencies
- Too much complexity:
  - complexity of requirements
  - complexity of operating environment
  - complexity of designs

# Need for precise models/blueprints

- Precision from early stages with models
  - Precise descriptions of intent
  - Amenable to analysis by tools
  - Identify and fix ambiguities and inconsistencies as early as possible
- Mastering complexity
  - Encourage abstraction
  - Focus on *what* a system does
  - Early focus on *key / critical* features
  - Incremental analysis and design

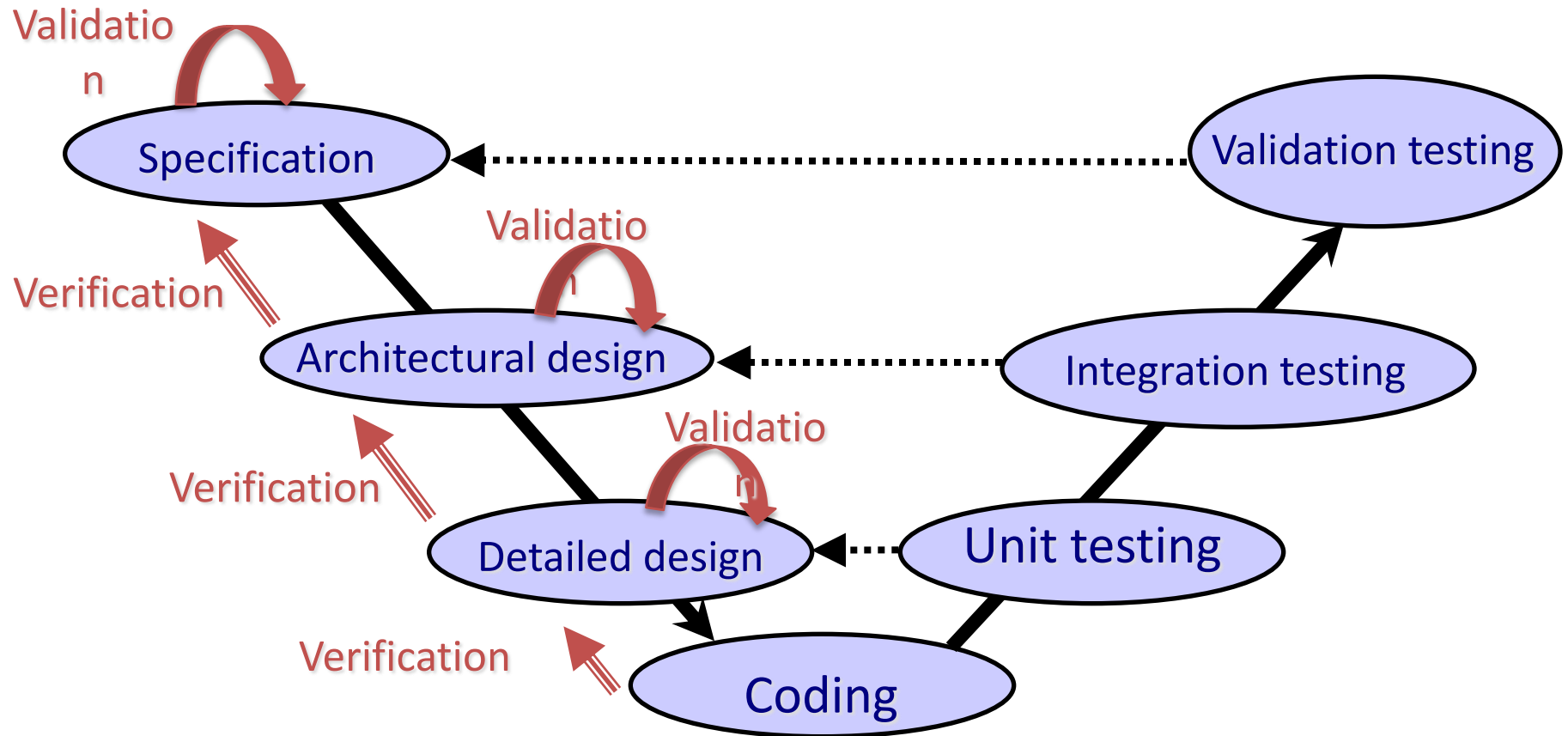
# Formal Methods

- Mathematical techniques for formulation and analysis of systems
- Formal methods facilitate:
  - Clear specifications (contract)
  - Rigorous *validation* and *verification*

*Validation*: does the contract specify the right system?  
– answered informally

*Verification*: does the finished product satisfy the contract?  
– can be answered formally

# Early stage analysis





# Rapid prototyping *versus* modelling

- **Rapid prototyping:** provides early stage feedback on system functionality
  - Plays an important role in getting **user feedback**
  - and in understanding some design constraints
  - But we will see that formal modelling and proof provide a **deep understanding** that is hard to achieve with rapid prototyping
- **Advice: use any approach that improves design process!**

# Event-B (Abrial)

- State-transition model (like ASM, B, VDM, Z)
  - set theory as mathematical language
- Refinement (based on action systems by Back)
  - data refinement
  - one-to-many event refinement
  - new events (stuttering steps)
- Proof method
  - Refinement proof obligations (POs) generated from models
  - Automated and interactive provers for POs

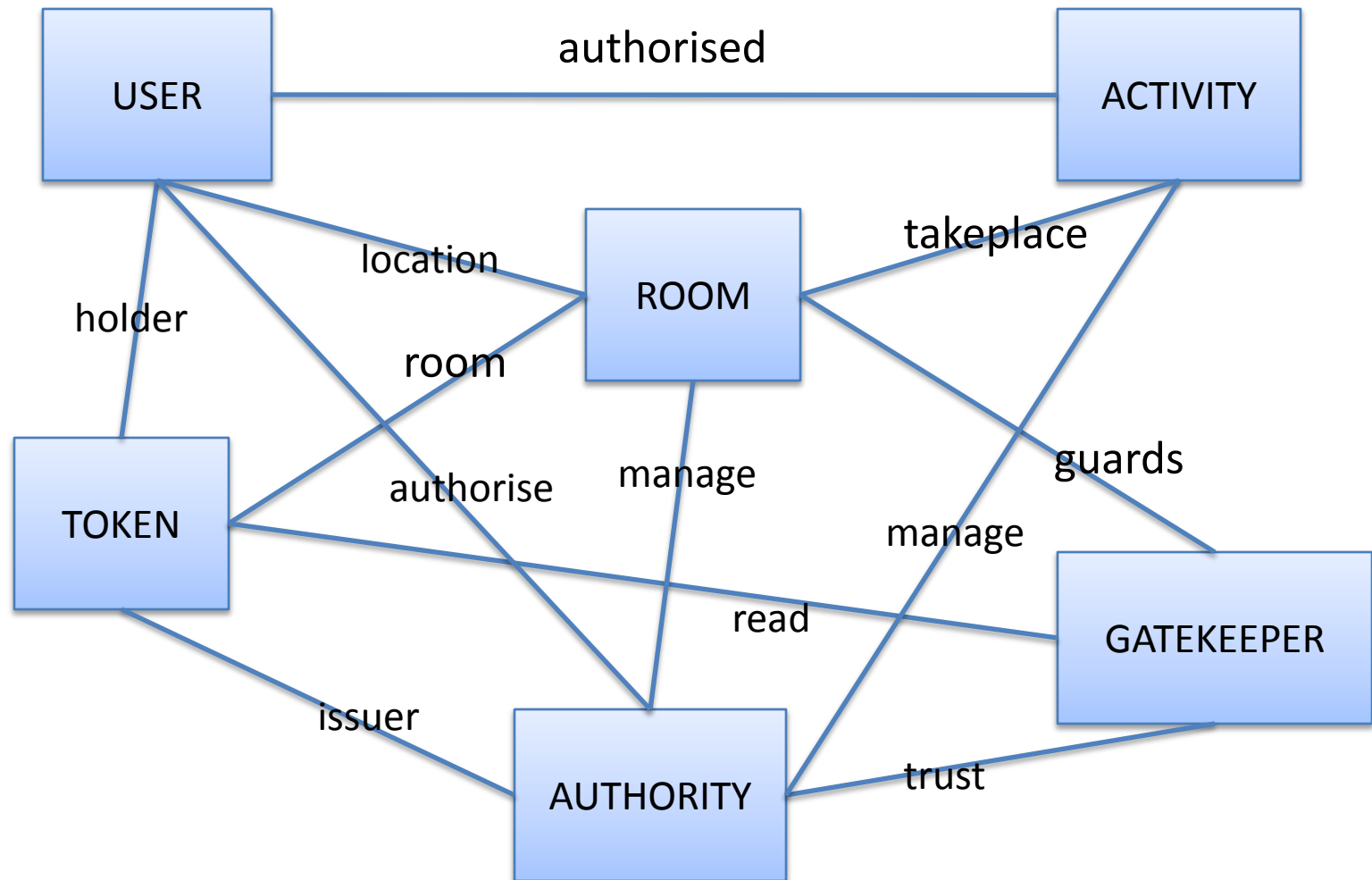
# Rational design, by example

- Example: access control system
- Example intended to give a feeling for:
  - modelling language
  - abstraction and refinement
  - role of verification

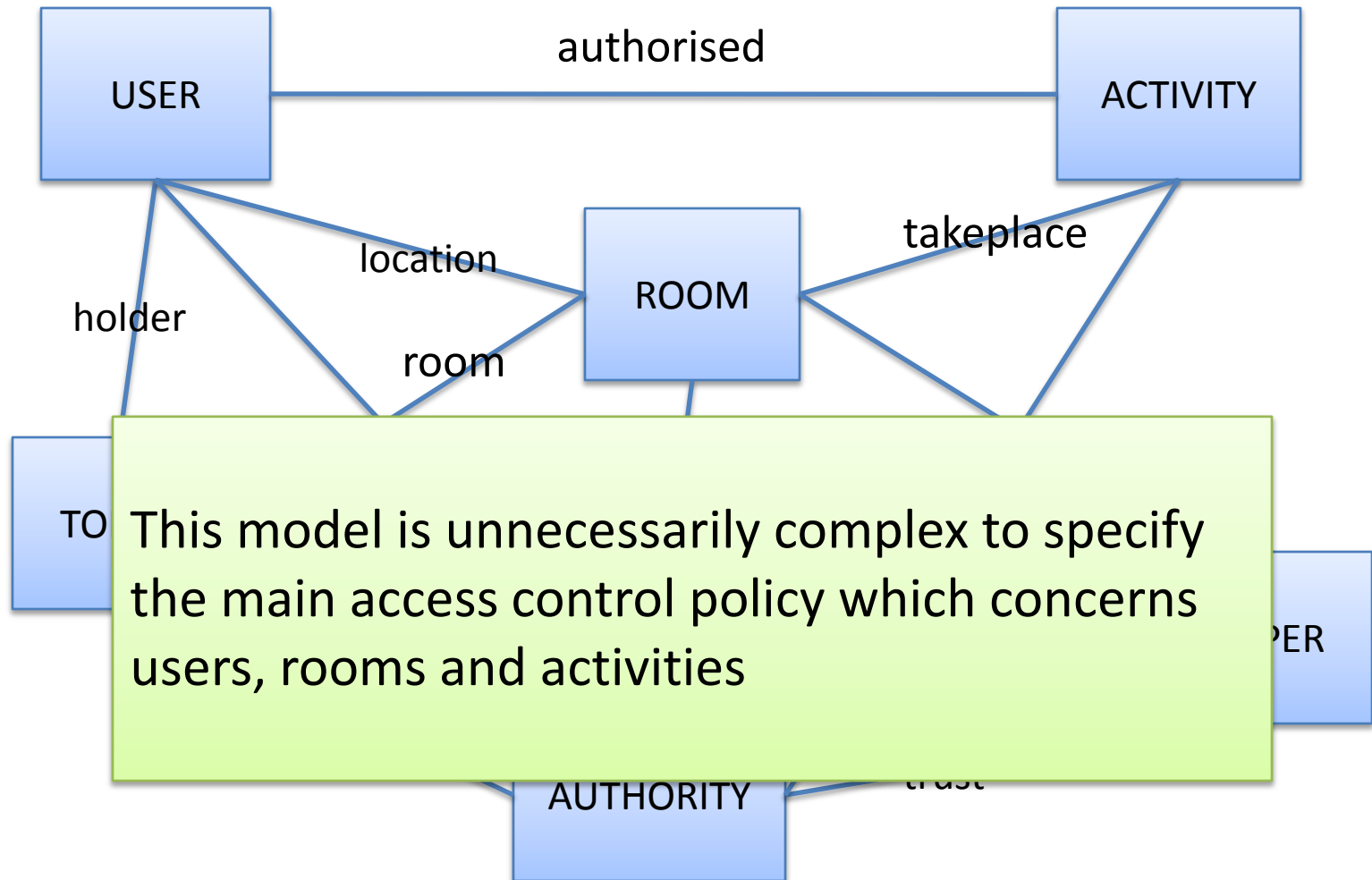
# Access control system

- Users are authorised to engage in activities
- User authorisation may be added or revoked
- Activities take place in rooms
- Users gain access to a room using a one-time token provided they have authority to engage in the room activities
- Tokens are issued by a central authority
- Tokens are time stamped
- A room gateway allows access with a token provided the token is valid

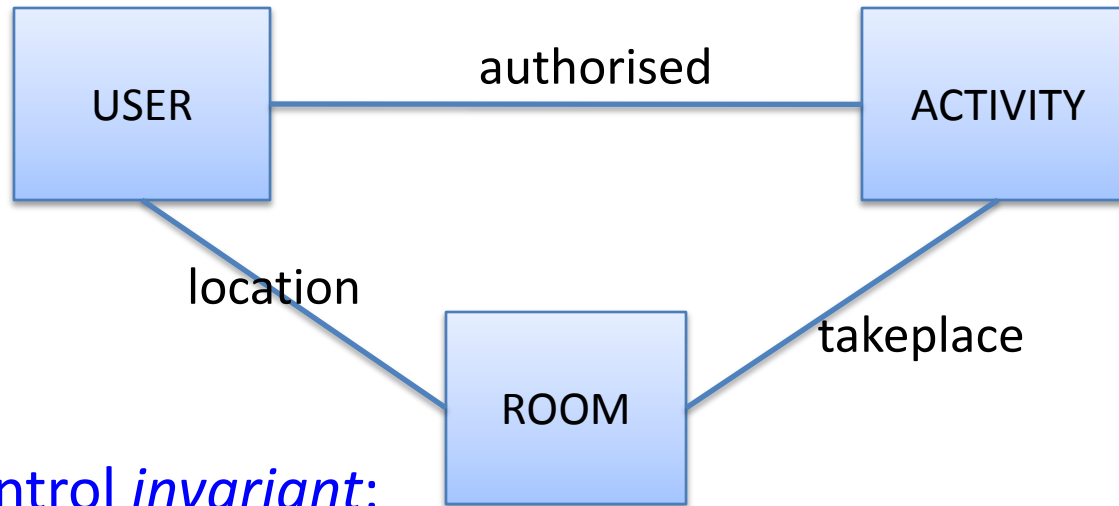
# Entity-relationship diagram



# Entity-relationship diagram



# Simplify / abstract



*Access control invariant:*

**if** user  $u$  is in room  $r$ ,

**then**  $u$  must be authorised to engaged in all activities that can take place in  $\underline{r}$

$$\text{location}(u) = r \Rightarrow \text{takeplace}[r] \subseteq \text{authorised}[u]$$

*Abstraction:* focus on key entities in the problem domain

# Enter a room

Enter  $\triangleq$

when

grd1 :  $u \in \text{User}$

grd2 :  $r \in \text{Room}$

grd3 :  $\text{takeplace}[r] \subseteq \text{authorised}[u]$

then

act1 :  $\text{location}(u) := r$

end

Does this operation maintain the security invariant?



# Remove authorisation

RemoveAuth(u,a)  $\triangleq$

when

grd1 :  $u \in \text{User}$

grd2 :  $a \in \text{Activity}$

grd3 :  $u \mapsto a \in \text{authorised}$

then

act1 :  $\text{authorised} := \text{authorised} \setminus \{ u \mapsto a \}$

end

Does this operation maintain the security invariant?

# Counterexample from model checking with ProB plug-in for Rodin

The screenshot displays the ProB model checker interface. The main window shows a state M1 with the following variables and values:

Name	Value
authorised	$\{(User1 \mapsto Activity1), (User2 \mapsto Activity2)\}$
location	$\{(User1 \mapsto Room2)\}$
takeplace	$\{(Room1 \mapsto Activity1), (Room1 \mapsto Activity2), (Room2 \mapsto Activity1), (Room2 \mapsto Activity2)\}$

The right-hand pane shows the sequence of operations performed during the model checking process:

```
Operations
RemAuth(Activity2, User1)
Enter(Room2, User1)
AddAuth(Activity2, User2)
AddAuth(Activity2, User1)
AddAuth(Activity1, User1)
$initialise_machine({}, {}, {z})
$setup_constants()
(root)
```

At the bottom of the interface, a red banner indicates the result of the model checking: **invariant violated!**

# Failing proof with Rodin

The screenshot displays the Rodin Platform interface for a project named 'Event-B - Rooms1/M1.bum'. The left sidebar, 'Event-B Explorer', shows a tree structure with 'M1' selected, containing 'Variables', 'Invariants', 'Events', and 'Proof Obligations'. The 'Proof Obligations' list includes several items, with 'RemAuth/inv4/INV' marked with a red question mark, indicating a failing proof. The main editor shows the Rodin code for the 'RemAuth' event, which is highlighted in blue. The code defines three events: 'Enter', 'Leave', and 'RemAuth'. The 'RemAuth' event is defined as follows:

```
event RemAuth
  any u a
  where
    @grd3  $u \mapsto a \in \text{authorised}$ 
  then
    @act1  $\text{authorised} := \text{authorised} \setminus \{ u \mapsto a \}$ 
  end
end
```

The bottom status bar indicates '0 errors, 0 warnings, 0 infos'. The bottom panel shows tabs for 'RODIN Keyboard', 'Progress', 'Symbols', 'Rodin Problems', and 'Statistics'.

# Strengthen guard of *RemAuth*

The screenshot displays the Rodin Platform interface for Event-B. The title bar indicates the project is 'Event-B - Rooms1/M1.bum' in the 'Rodin Platform' workspace. The interface is divided into several panes:

- Event-B Explorer (Left):** A tree view showing the project structure. Under 'Proof Obligations', several items are listed with green checkmarks, indicating they are proven. The last item, 'RemAuth/inv4/INV', is selected and highlighted in blue.
- Event-B Code Editor (Right):** Displays the Event-B code for the 'RemAuth' event. The code is as follows:

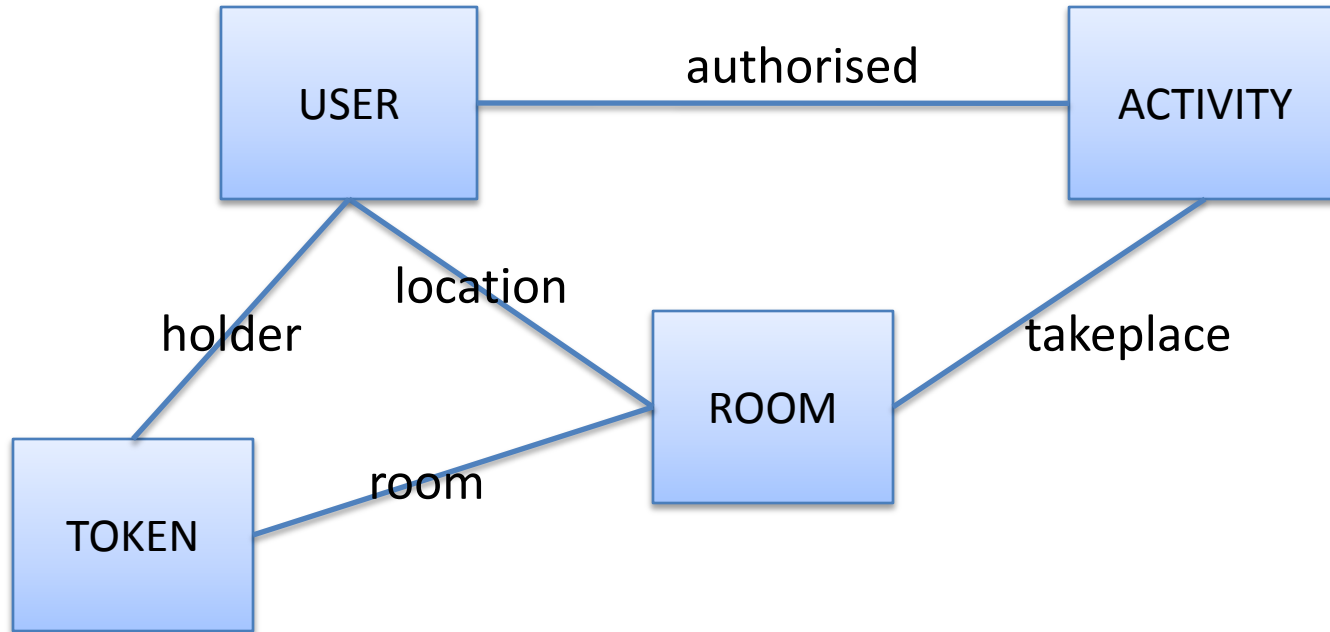
```
@grd2  $r \in \text{Room}$ 
@grd3  $\text{takeplace}[\{r\}] \subseteq \text{authorised}[\{u\}]$ 
then
  @act1  $\text{location} := \text{location} \cup \{u \mapsto r\}$ 
end

event Leave
  any  $u\ r$ 
  where
    @grd1  $u \mapsto r \in \text{location}$ 
  then
    @act1  $\text{location} := \text{location} \setminus \{u \mapsto r\}$ 
  end

event RemAuth
  any  $u\ a$ 
  where
    @grd3  $u \mapsto a \in \text{authorised}$ 
    @grd4  $\forall rr. u \mapsto rr \in \text{location} \Rightarrow rr \mapsto a \notin \text{takeplace}$ 
  then
    @act1  $\text{authorised} := \text{authorised} \setminus \{u \mapsto a\}$ 
  end
end
```

The line containing '@grd4' is highlighted in blue in the original image.
- Bottom Panel:** Contains tabs for 'RODIN Keyboard', 'Progress', 'Symbols', 'Rodin Problems', and 'Statistics'.

# Now refine



Abstract condition on a user and room for entering  
 $\text{takeplace}[r] \subseteq \text{authorised}[u]$

is replaced by a condition on a token

$t \in \text{valid} \wedge \text{room}(t) = r \wedge \text{holder}(t) = u$

# Failing refinement proof

Proving - Rooms1/M2.bps - Rodin Platform - /Users/mjb/Rodin/workspace1.0

Resource ProB Proving Event-B

Proof T Proof I

G

- simplification rewrites
  - type rewrites
    - takeplace[{r}] $\subseteq$ auth

Enter/grd3/GRD

<input type="checkbox"/>	ct	$u \in \text{User} \setminus \text{dom}(\text{location})$
<input type="checkbox"/>	ct	t <sub>etok</sub>
<input type="checkbox"/>	ct	$r = \text{rtok}(t)$
<input type="checkbox"/>	ct	$u = \text{utok}(t)$

State

Goal

ct takeplace[{r}] $\subseteq$ authorised[{u}]

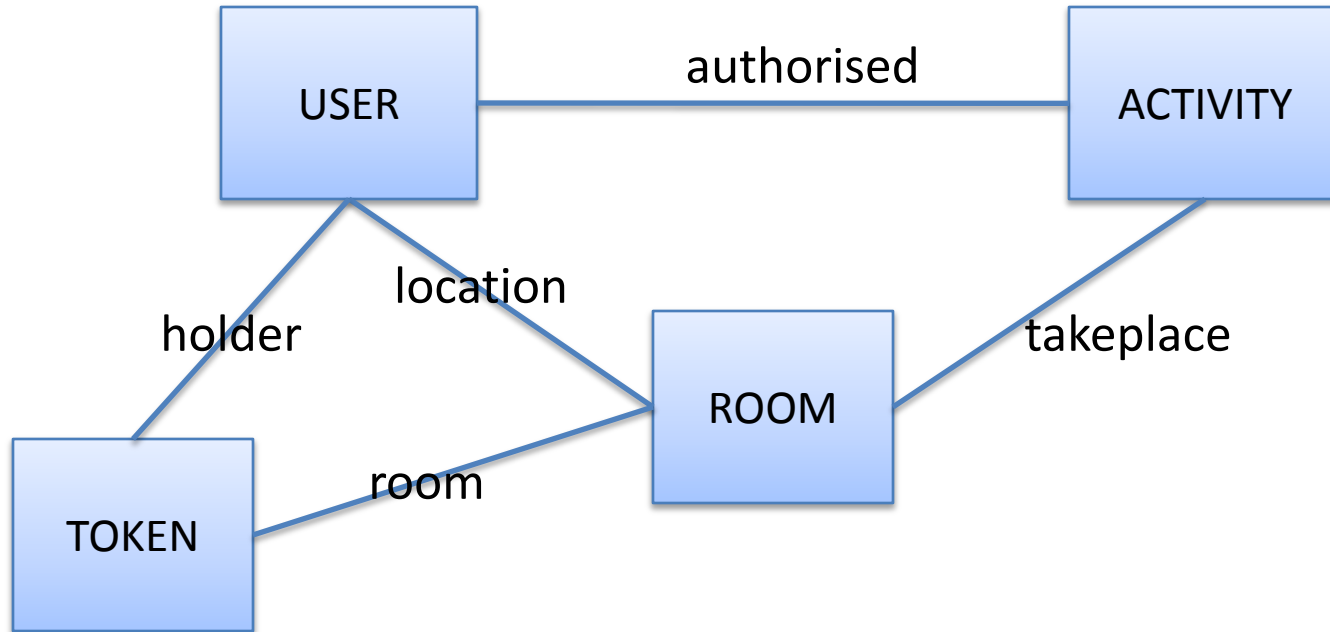
Proof Cont Statistics Rodin Prob

nPP R p0 dc ah ae

Meida Channel3  
mondex  
OwikiGries  
Records  
Rooms1
  - C1
  - C2
  - M1
  - M2
    - Variables
    - Invariants
    - Events
    - Proof Obligations
      - CreateToken/grd5/WD
      - CreateToken/grd6/WD
      - Enter/grd3/WD
      - Enter/grd4/WD
      - Enter/grd3/GRD
      - RemAuth/grd5/WD
      - RemAuth/grd6/WD
      - RemAuth/grd4/GRD

Rules  
SecureDB

# Gluing invariant



To ensure consistency of the refinement we need invariant:

$t \in \text{valid}$

$\Rightarrow$

$\text{takeplace} [\text{room}(t)] \subseteq \text{authorised} [\text{holder}(t)]$

# Invariant enables PO discharge

Proving - Rooms1/M2.bps - Rodin Platform - /Users/mjb/Rodin/workspace1.0

Resource ProB Proving Event-B

Proof T Proof I

G ML

M1 M2 M2

Enter/grd3/GRD

- ☐ ueUser \ dom(location)
- ☐ tetok
- ☐ r=rtok(t)
- ☐ u=utok(t)

State

Goal

takeplace[{r}] ⊆ authorised[{u}]

Proof Cont Statistics Rodin Prob

nPP R p0 dc ah ae

Rules  
SecureDB  
Shadow  
Shared Buffers 20081008

Progr Event Search

M1  
M2

- Variables
- Invariants
- Events
- Proof Obligations
  - inv2 /WD
  - INITIALISATION /inv2 /INV
  - AddAuth /inv2 /INV
  - CreateToken /grd5 /WD
  - CreateToken /grd6 /WD
  - CreateToken /inv2 /INV
  - Enter /grd3 /WD
  - Enter /grd4 /WD
  - Enter /inv2 /INV
  - Enter /grd3 /GRD
  - RescindToken /inv2 /INV
  - RemAuth /grd5 /WD
  - RemAuth /inv2 /INV
  - RemAuth /grd4 /GRD



# But get new failing PO

Proving - Rooms1/M2.bps - Rodin Platform - /Users/mjb/Rodin/workspace1.0

Resource ProB Proving Event-B

Pr Pr

G | + - Home Left Right

simplification rewrites  
type rewrites  
simplification  
goal (fre  
goal  
goal  
tak

RemAuth/inv2/INV

ct  $\forall t$   $\text{tetok}$   
 $\Rightarrow$   
 $\text{takeplace}[\{\text{rtok}(t)\}] \leq \text{authorised}[\{\text{utok}(t)\}]$   
 $u \mapsto a \# \text{authorised}$   
 $u \# \text{dom}(\text{location})$   
 $\Rightarrow$   
 $\text{location}(u) \mapsto a \# \text{takeplace}$

State

Goal

ct  $\text{takeplace}[\{\text{rtok}(t)\}] \leq (\text{authorised} \setminus \{u \mapsto a\})[\{\text{utok}(t)\}]$

Proof Control Statistics Rodin Problems

npp R p0 dc ah ae

Proof Control

ProB Proving Event-B

Variables  
Invariants  
Events  
Proof Obligations

- inv2/WD
- INITIALISATION/inv2
- AddAuth/inv2/INV
- CreateToken/grd5/
- CreateToken/grd6/
- CreateToken/inv2/II
- Enter/grd3/WD
- Enter/grd4/WD
- Enter/inv2/INV
- Enter/grd3/GRD
- RescindToken/inv2/
- RemAuth/grd5/WD
- RemAuth/inv2/INV
- RemAuth/grd4/GRD

Rules  
SecureDB  
Shadow  
ShadowBuffer20081008

# Source of failing PO

Proving - Rooms1/M2.bps - Rodin Platform - /Users/mjb/Rodin/workspace1.0

Resource ProB Proving Event-B

Proof Tree Proof Information

### RemAuth/inv2/INV

- Event in M1
  - RemAuth:
    - ANY  $u, a$  WHERE
      - $\text{grd3: } u \mapsto a \in \text{authorised}$
      - $\text{grd4: } \forall rr. u \mapsto rr \in \text{location} \Rightarrow rr \mapsto a \notin \text{takeplace}$
    - THEN
      - $\text{act1: } \text{authorised} = \text{authorised} \setminus \{ u \mapsto a \}$
    - END
- Event in M2
  - RemAuth:
    - REFINES
      - RemAuth
    - ANY  $u, a$  WHERE
      - $\text{grd3: } u \mapsto a \in \text{authorised}$
      - $\text{grd5: } u \in \text{dom}(\text{location}) \Rightarrow \text{location}(u) \mapsto a \notin \text{takeplace}$
      - $\text{grd6: } \tau$
    - THEN
      - $\text{act1: } \text{authorised} = \text{authorised} \setminus \{ u \mapsto a \}$
    - END
- Invariant in M2
  - $\text{inv2: } \forall t. t \in \text{tok} \Rightarrow \text{takeplace}[\{\text{rtok}(t)\}] \subseteq \text{authorised}[\{\text{utok}(t)\}]$

State

Goal

takepla

Proof Obligations

- inv2 /WD
- INITIALISATION/inv2
- AddAuth/inv2/INV
- CreateToken/grd5/
- CreateToken/grd6/
- CreateToken/inv2/II
- Enter/grd3/WD
- Enter/grd4/WD
- Enter/inv2/INV
- Enter/grd3/GRD
- RescindToken/inv2/
- RemAuth/grd5/WD
- RemAuth/inv2/INV**
- RemAuth/grd4/GRD

Rules

SecureDB

Shadow

ShadowBuffer=20001000

# Strengthen guard of refined *RemAuth*

The screenshot displays the Rodin Platform interface for proving the refinement of the *RemAuth* event. The main editor shows the following code:

```
event RemAuth refines RemAuth
  any  $u\ a$ 
  where
    @grd3  $u \mapsto a \in \text{authorised}$ 
    @grd5  $u \in \text{dom}(\text{location}) \Rightarrow \text{location}(u) \mapsto a \notin \text{takeplace}$ 
    @grd6  $\forall t \cdot t \in \text{tok} \wedge u = \text{utok}(t) \Rightarrow \text{rtok}(t) \mapsto a \notin \text{takeplace}$ 
  then
    @act1  $\text{authorised} := \text{authorised} \setminus \{ u \mapsto a \}$ 
  end
end
```

The goal panel shows the goal to be proved:

```
 $\forall t \cdot t \in \text{tok} \Rightarrow \text{takeplace}[\{\text{rtok}(t)\}] \subseteq (\text{authorised} \setminus \{ u \mapsto a \})[\{\text{utok}(t)\}]$ 
```

The right-hand side of the interface shows a list of proof obligations, all of which are marked as solved (green checkmark):

- inv2 / WD
- INITIALISATI
- AddAuth / inv
- CreateToke
- CreateToke
- CreateToke
- Enter / grd3 /
- Enter / grd4 /
- Enter / inv2 / l
- Enter / grd3 /
- RescindTok
- RemAuth / gr
- RemAuth / gr
- RemAuth / in
- RemAuth / gr

The bottom panel shows the Proof Control, Statistics, and Rodin Problems tabs, along with a toolbar and a status bar.

# Rational design – what, how, why

- *What* does it achieve?

**if** user  $u$  is in room  $r$ ,

**then**  $u$  must be authorised to engaged in all activities that can take place in  $\underline{r}$

- *How* does it work?

Check that a user has a valid token

- *Why* does it work?

For any valid token  $t$ , the holder of  $t$  must be authorised to engage in all activities that can take place in that room

# What, how, why written in B

- *What* does it achieve?

$\text{location}(u) = r$

$\Rightarrow \text{takeplace}[r] \subseteq \text{authorised}[u]$

- *How* does it work?

$t \in \text{valid} \wedge r = \text{room}(t) \wedge u = \text{holder}(t)$

- *Why* does it work?

$t \in \text{valid}$

$\Rightarrow$

$\text{takeplace}[\text{room}(t)] \subseteq \text{authorised}[\text{holder}(t)]$

# Decomposition

- Beneficial to model systems **abstractly** with **little architectural structure** and **large atomic steps**
  - e.g., *file transfer, replicated database transaction*
- **Refinement** and **decomposition** are used to add structure and then separate elements of the structure
- **Atomicity decomposition:** Decomposing large atomic steps to more fine-grained steps
- **Model decomposition:** Decomposing refined models to for (semi-)independent refinement of sub-models
- Towards a **method** for decomposition

# Simple file store example

**sets** FILE, PAGE, DATA

$\text{CONT} = \text{PAGE} \rightarrow \text{DATA}$

**machine** filestore

**variables** file, dsk

**invariant**

$\text{file} \subseteq \text{FILE} \wedge$   
 $\text{dsk} \in \text{file} \rightarrow \text{CONT}$

**initialisation**

$\text{file} := \{ \} \quad || \quad \text{dsk} := \{ \}$

**events**

CreateFile  $\triangleq$  ...

WriteFile  $\triangleq$  // set contents of  $f$  to be  $c$

**any**  $f, c$  **where**

$f \in \text{file}$

$c \in \text{CONT}$

**then**

$\text{dsk}(f) := c$

**end**

ReadFile  $\triangleq$  // return data in page  $p$  of  $f$

**any**  $f, p, d!$  **where**

$f \in \text{file}$

$p \in \text{dom}(\text{dsk}(f))$

$d! = \text{dsk}(f)(p)$

**end**

# Refinement of file store

- Instead of writing entire contents of a file in one atomic step, each page is written separately

**machine**filestore2

**refines**filestore

**variables**            file,dsk,writing,wbuf, sdsk

**invariant**

writing  $\subseteq$  file

wbuf  $\in$  writing  $\rightarrow$  CONT

sdsk  $\in$  writing  $\rightarrow$  CONT            // shadow disk



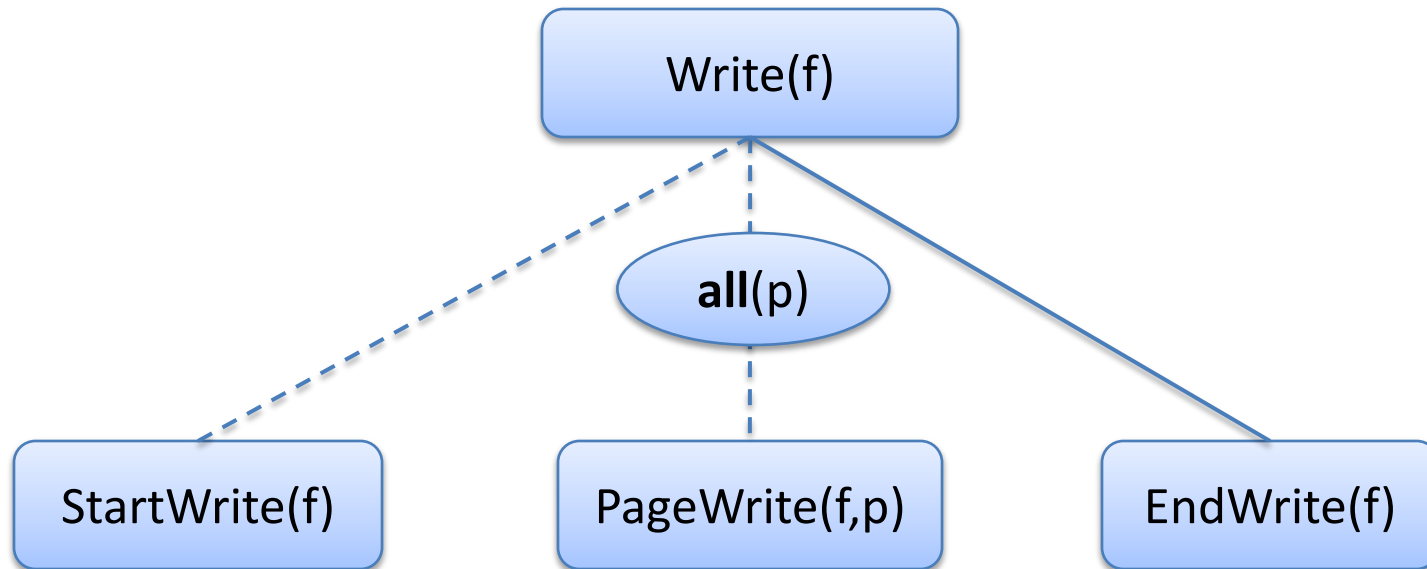
# Breaking atomicity

- Abstract *WriteFile* is replaced by
  - new events: *StartWriteFile*, *WritePage*,
  - refining event: *EndWriteFile*
- Refined events for *different* files may interleave
- Non-interference is dealt with by treating new events as refinements of *skip*
  - new events must maintain gluing invariants
- **But:** refinement rule does not reflect the connection between then new events and the abstract event

# Event refinement diagrams

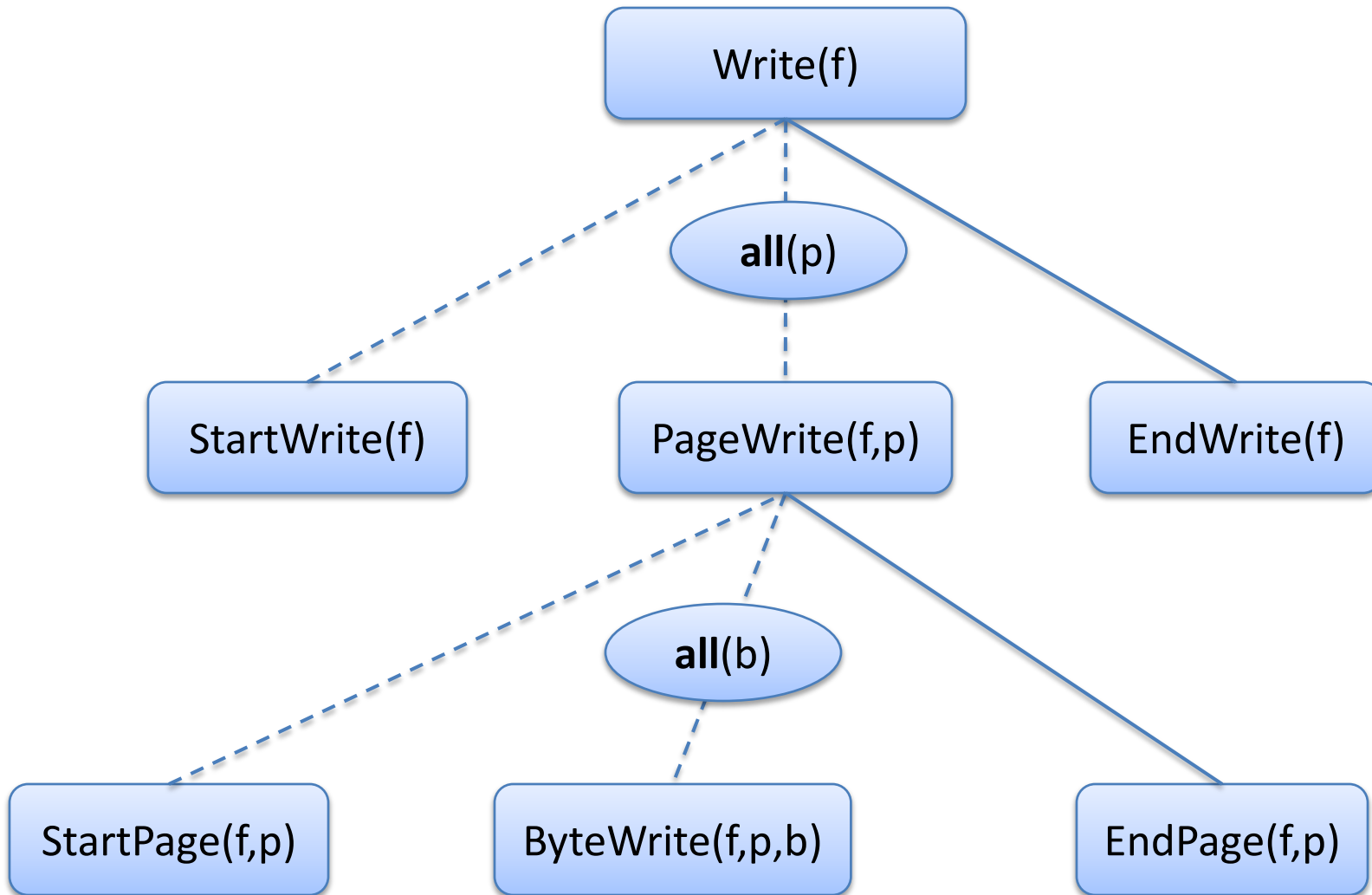
- Based on diagrammatic notation of *Jackson System Development* (JSD)
- Graphical representation of how abstract atomic events are refined
- We can exploit the hierarchical nature of JSD diagrams to represent event refinement
- Adapt JSD notation for our needs

# Event refinement diagram



- Diagram represents **atomicity refinement** explicitly *and*
- Diagram specifies **sequencing constraints** on events

# Hierarchical refinement



# Replicated data base

- Abstract model

$db \in \text{object} \rightarrow \text{DATA}$

Commit =    /\*    update a set of objects  $os$     \*/

**any**  $os$ , update

**where**

$os \subseteq \text{object} \wedge$

$\text{update} \in (os \rightarrow \text{DATA}) \rightarrow (os \rightarrow \text{DATA})$

**then**

$db := db \triangleleft + \text{update}(os \triangleleft db)$

**end**

# Refinement by replicated database

$\text{sdb} \in \text{site} \rightarrow (\text{object} \rightarrow \text{DATA})$

Update is by two phase commit:

PreCommit followed by Commit

Global **commit** if all sites *pre-commit*

Global **abort** if at least one site aborts

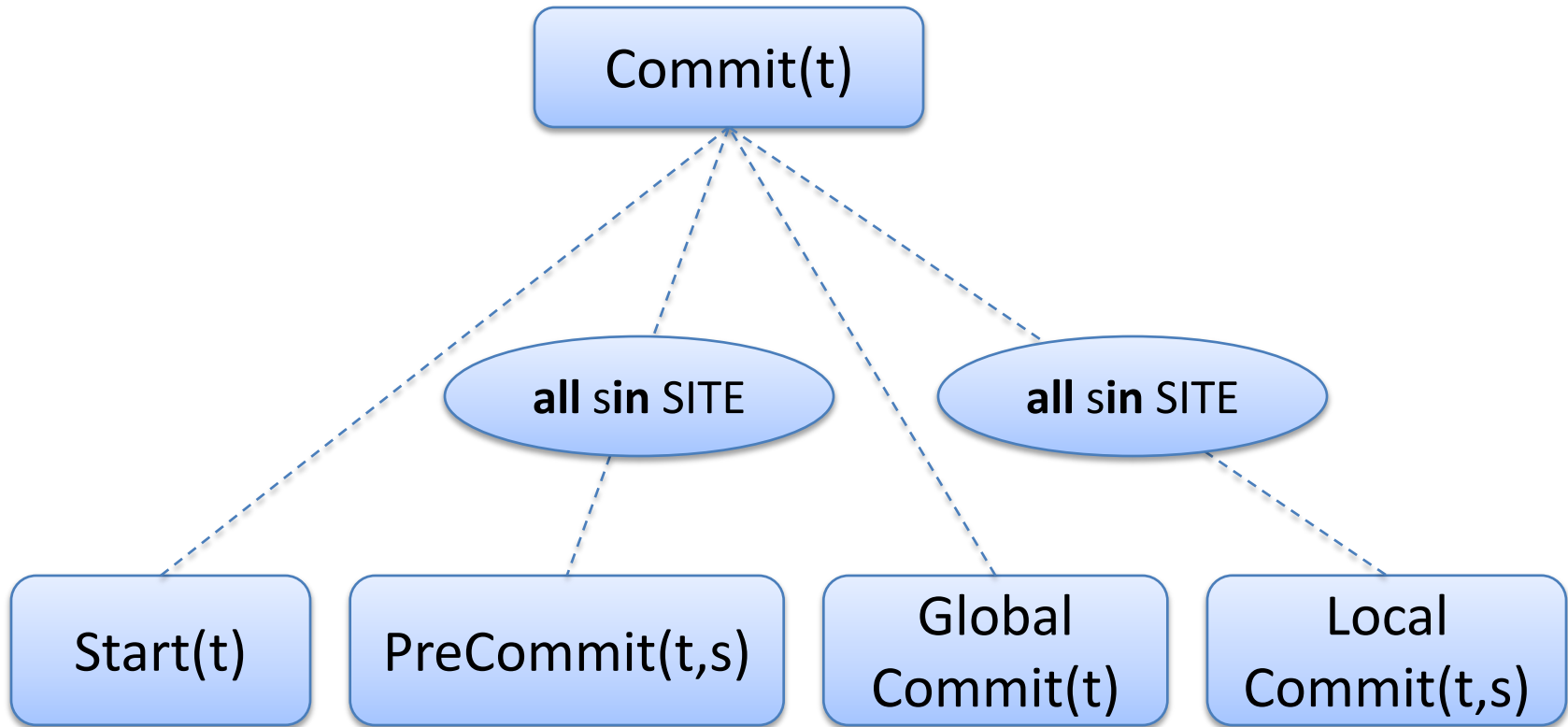
# Mutual Exclusion

At abstract level, update transaction is a choice of 2 atomic events:



Update transaction will commit or abort *but not both*

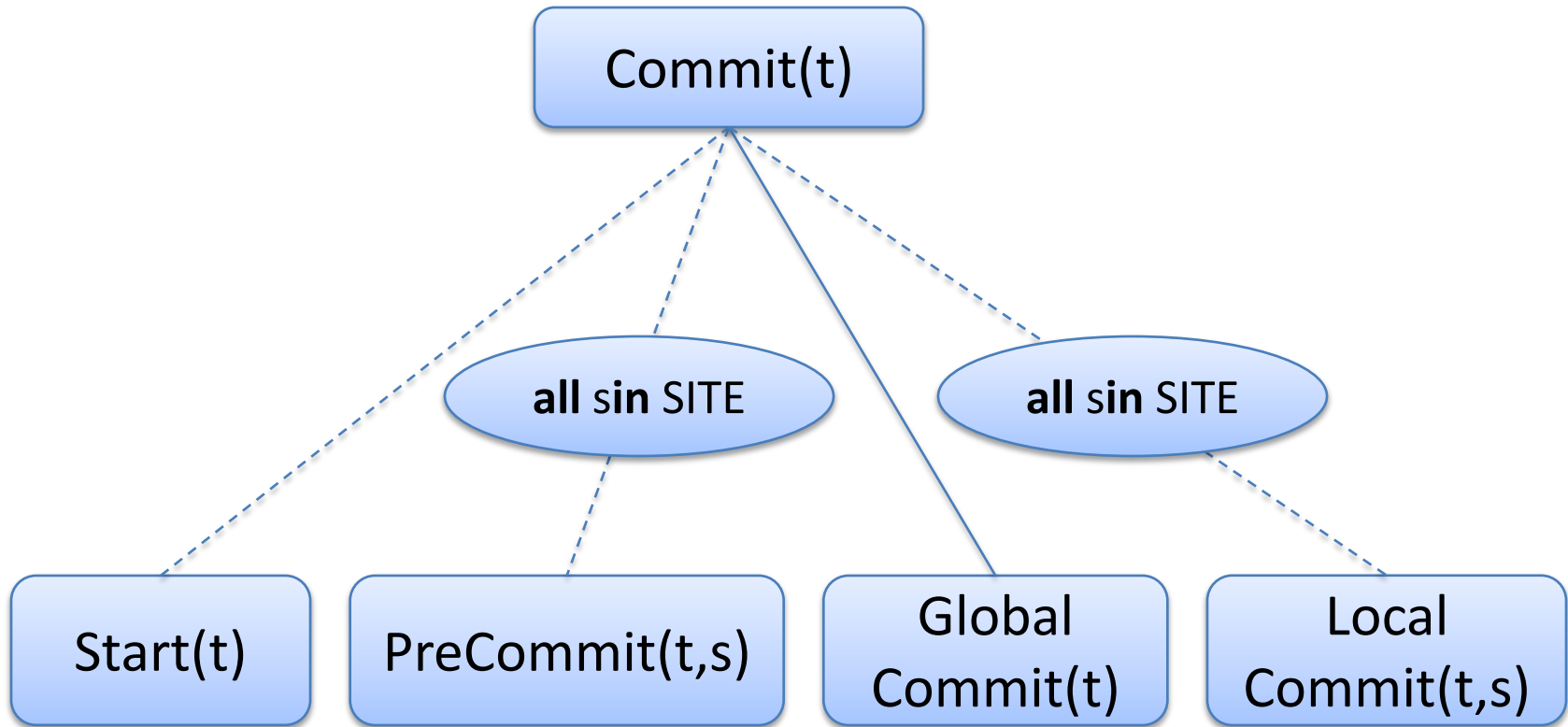
# Event refinement diagram for Commit



Which event refines the abstract *Commit*?

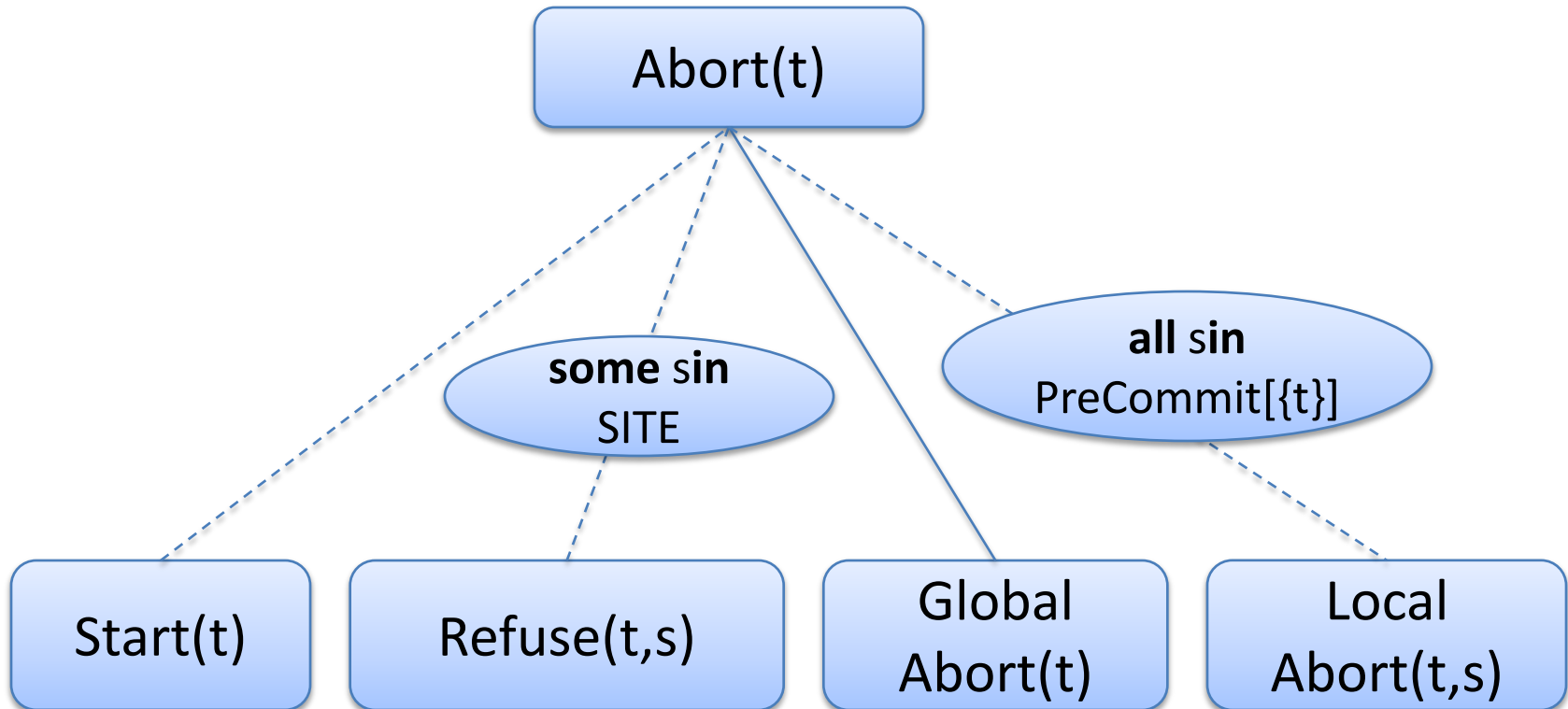


# Event refinement diagram for Commit



Decision to proceed is made by *GlobalCommit*

# Event refinement diagram for **Abort**

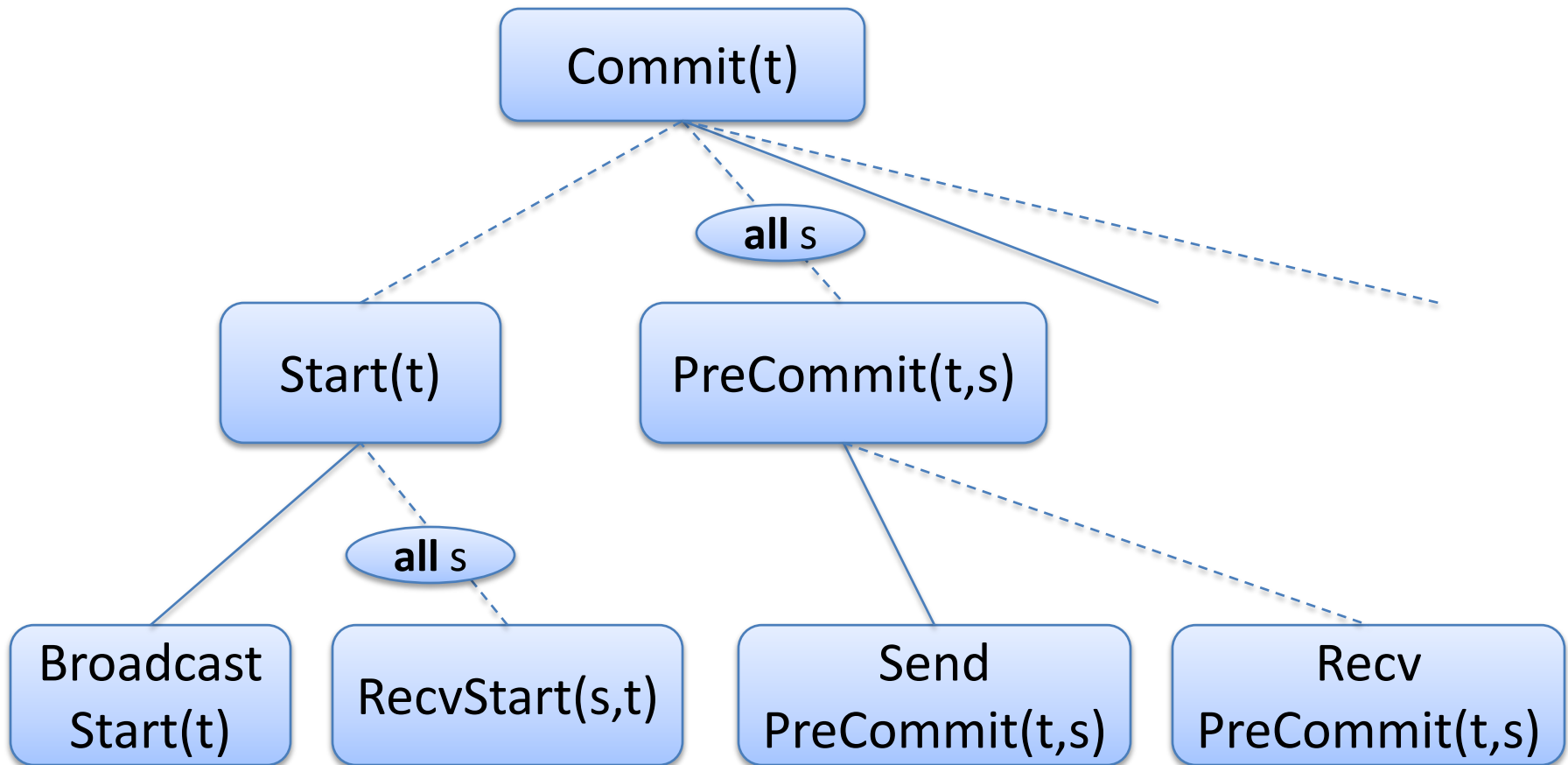


Protocol aborts transaction if some site aborts

# Commit and abort affect object locking

- *PreCommit(t,s)* : locks all objects for transaction *t* at site *s*
- *LocalCommit(t,s)* *LocalAbort(t,s)*: release all objects for transaction *t* at site *s*

# Introducing messaging



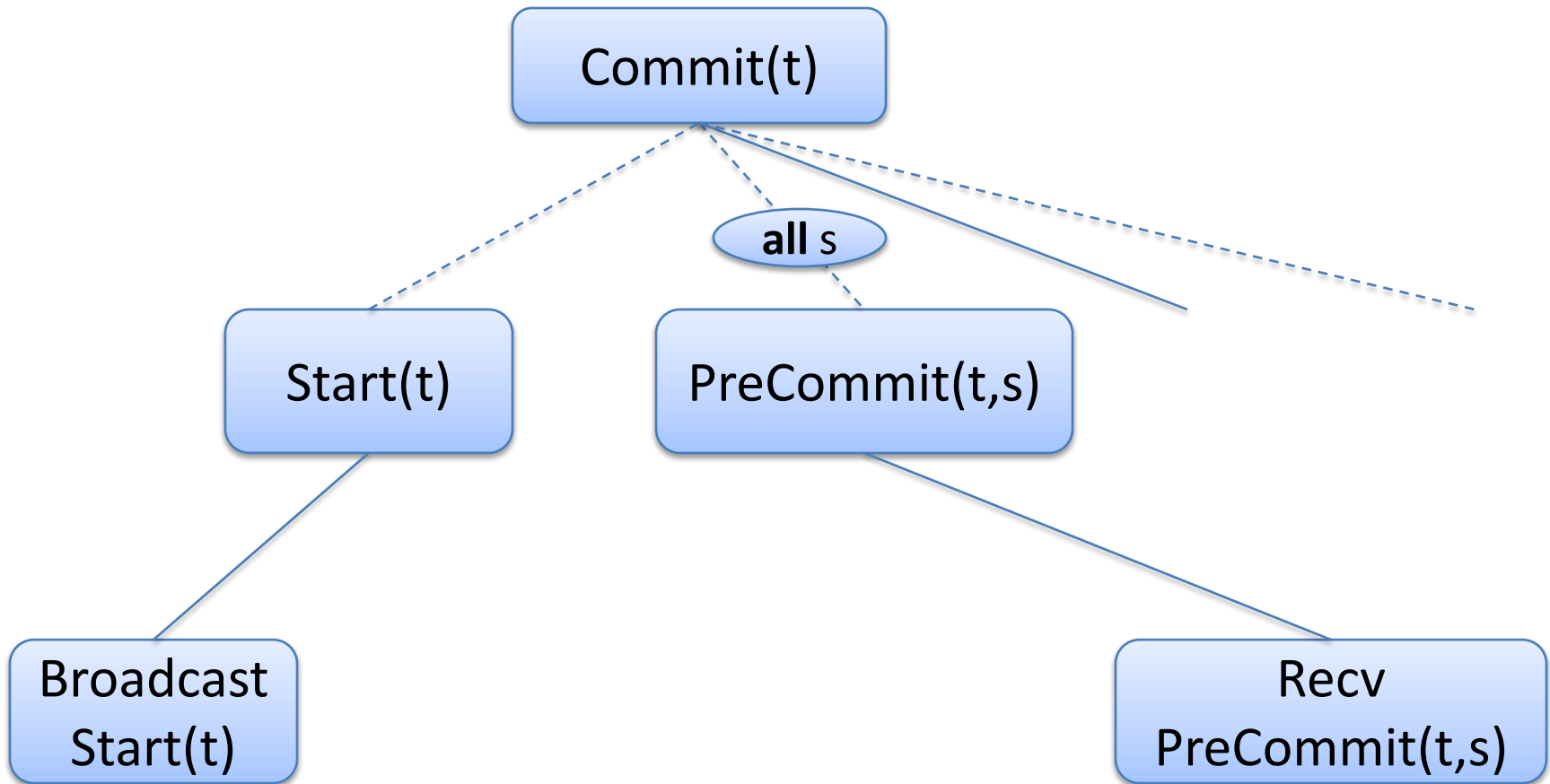
# Where are we going?

- Start with *system-level* model of transaction, independent of architecture/roles
- Then introduced *stages* of a transaction
  - *separation* of normal and error behaviour
- Next we introduce explicit message send/receive
  - this will allow us later to *separate* the requester/responder roles
- Hierarchical diagrams help us to *identify* and *manage* these steps

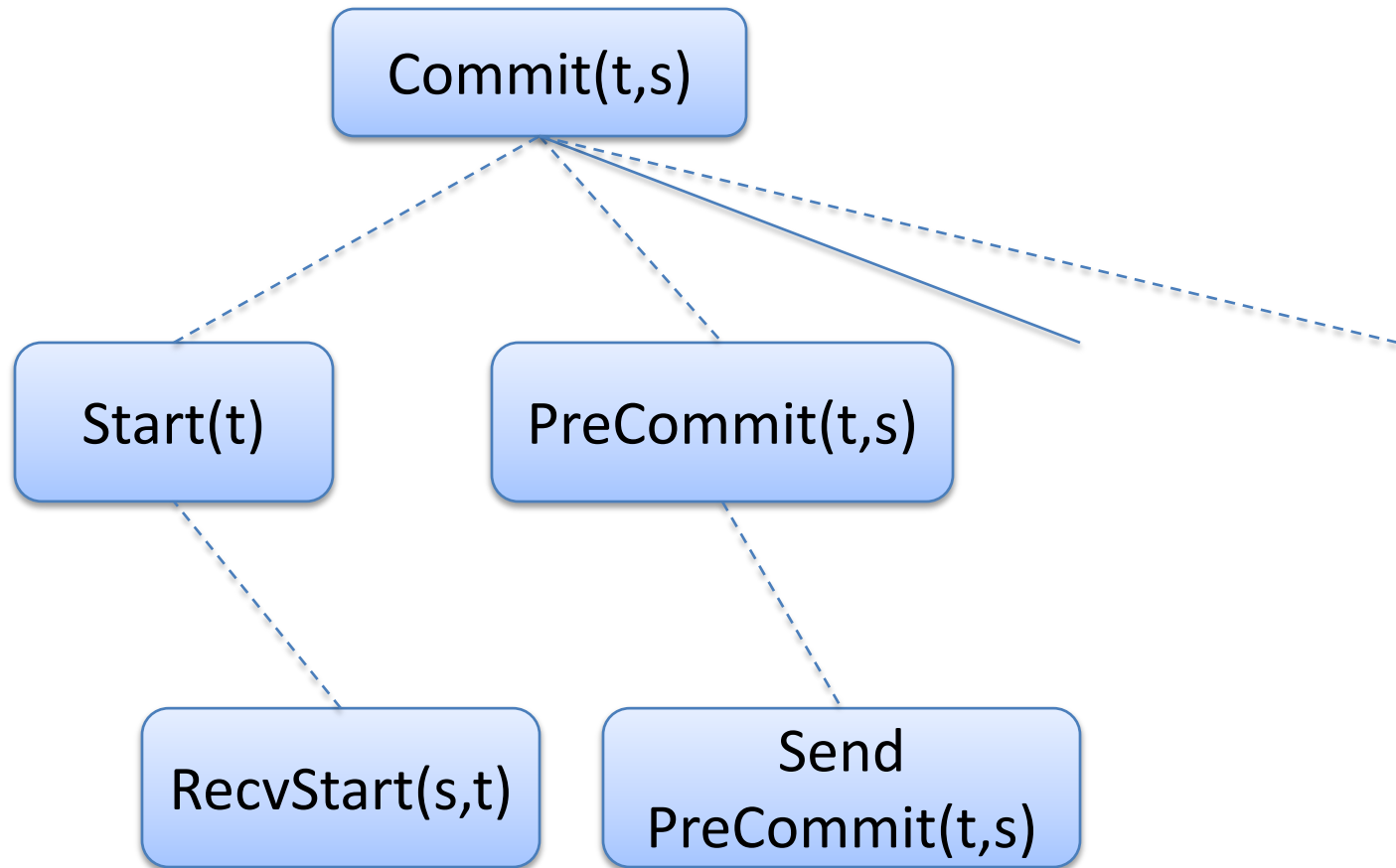
# Architectural/role decomposition

- Explicit message/receive allows to **separate** requester/responder roles
- We do this by **slicing** the diagrams

# Coordinator behaviour for database



# Non-coordinator behaviour for database





# Important Messages

- Formal modelling can be applied to *systems*
- Role of *formal modelling*:
  - increase understanding
  - decrease errors
- Role of *verification*:
  - improve quality of models (consistency, invariants)
- Role of *tools*:
  - make verification as automatic as possible, pin-pointing errors and even *suggesting* improvements
- *Methods needed*:
  - stronger guidelines for abstraction, refinement and decomposition needed
  - good *structures* help to ease their application
- In practice, refinement is *not* top-down!