Specification and design of distributed embedded middleware applications with SDL-2000

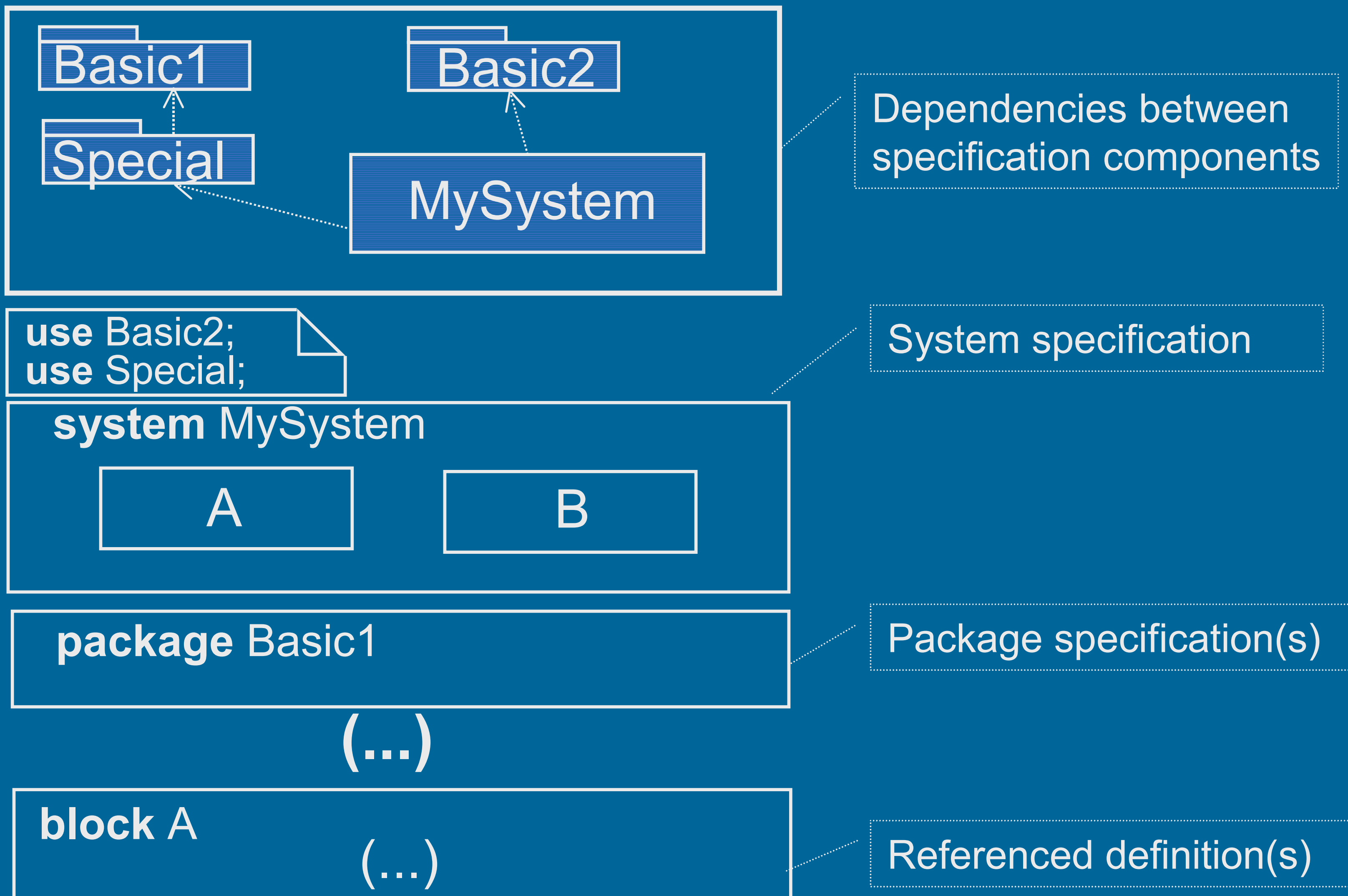
Dr. Eckhardt Holz

Humboldt-Universität zu Berlin

SDL-2000

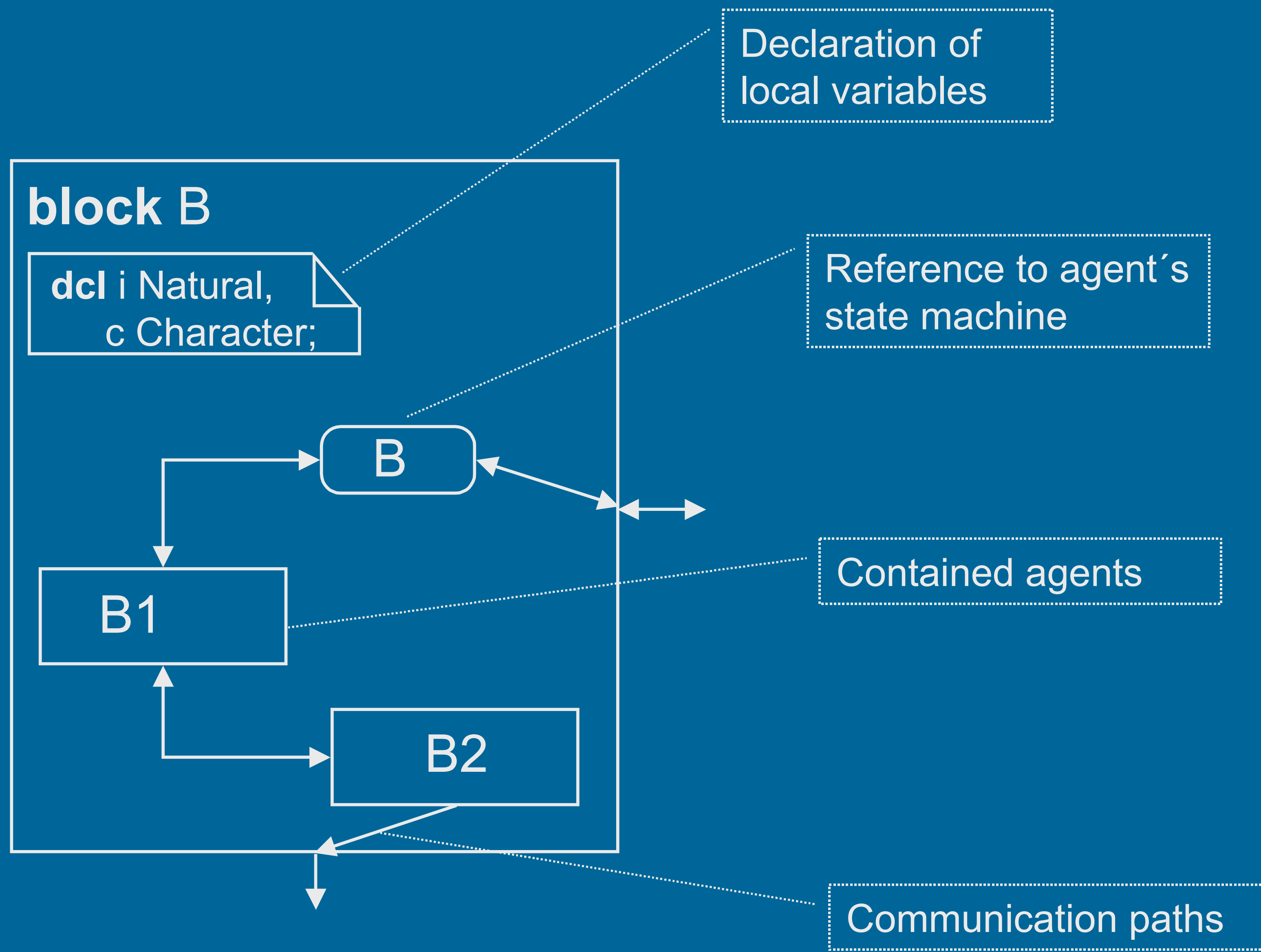
- ITU-T Specification and Description Language
  - graphical language
  - is completely based on object-orientation
  - has a new formal semantics
  - is accompanied by a new standard
    Z.109: SDL-UML-Profil

SDL-2000 Specification Structure

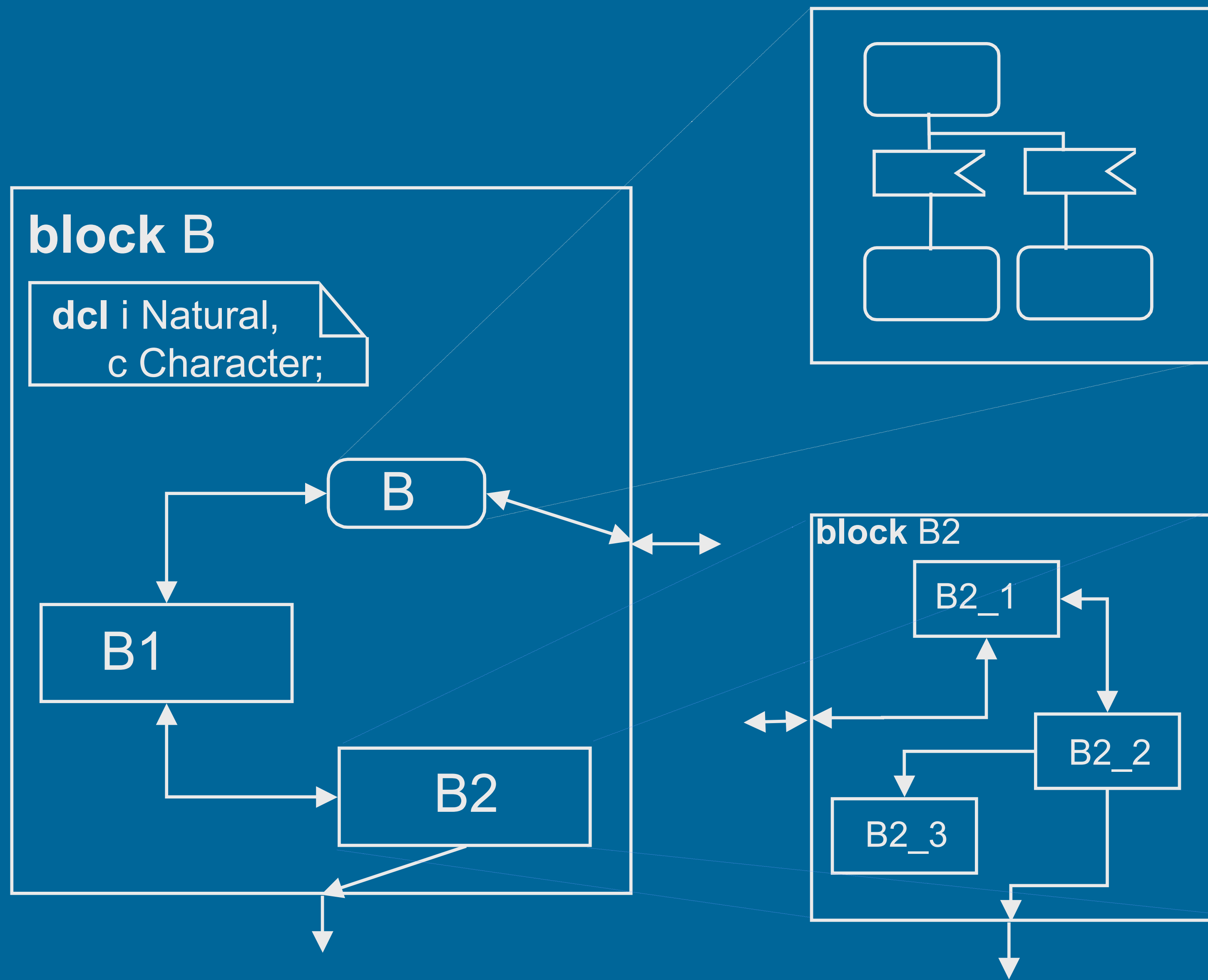


Agents

- basic specification concept
- model active components of a system
- an agent instance is an extended finite communicating state machine that has
  - its own identity
  - its own signal input queue
  - its own life cycle
  - a reactive behaviour specification



Declaration of local variables
block B
dcl i Natural,
c Character;
Reference to agent´s state machine
B
Contained agents
B1
B2
Communication paths


block B
dcl i Natural,
c Character;
B
B1
B2
block B2
B2_1
B2_2
B2_3

De-Composition of Agents

- structural decomposition into internal agents implies also decomposition of behaviour
- container of an agent determines scheduling semantics of its contents
  - concurrent agents: ***block***
  - alternating agents: ***process***

Block Agent

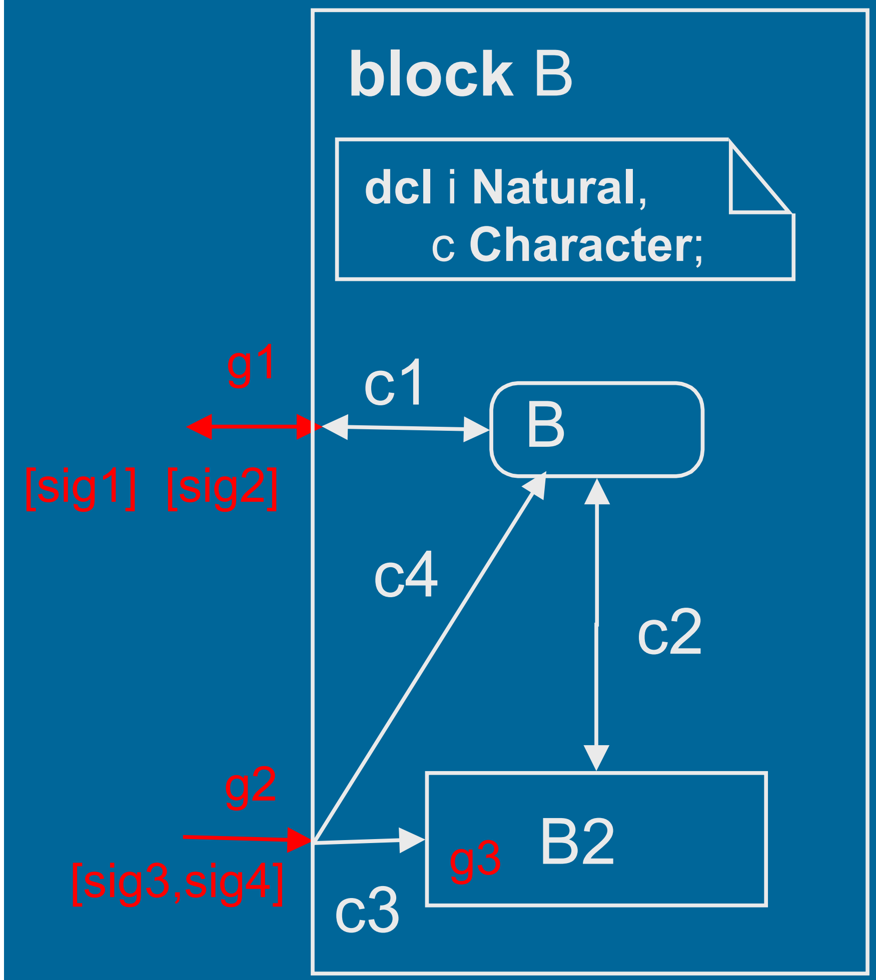
- all contained agents execute concurrently with each other and with the agents state machine
  - multiple threads of control
  - concurrent execution of multiple transitions
  - transitions execute with run-to-completion
- contained agents may be
  - blocks or processes

Process Agent

- all contained agents execute alternating with each other and with the agents state machine
  - at most one transition is executed at any point in time
  - selection is non-determined
  - transitions execute in run-to-completion
- contained agents
  - may be processes only

General Communication

- communication is based on signal exchange
- communication requires a complete path from sender to receiver consisting of
  - gates
  - channels
  - connections
- path may be defined
  - explicitly
  - or implicitly derived




```
block B;
gate g1 in with sig2;
        out with sig1;
gate g2  in with sig3,sig4;

state B referenced;
block B2 referenced;

channel c1        from env via g1 to this;
                  from this to env via g1;
endchannel;
channel c2        from this to B2;
                  from B2 to this;
endchannel;
channel c3        from env via g2
                  to B2 via g3;
endchannel;
channel c4        from env via g2 to this;
endchannel;
endblock B;
```

Advanced Communication

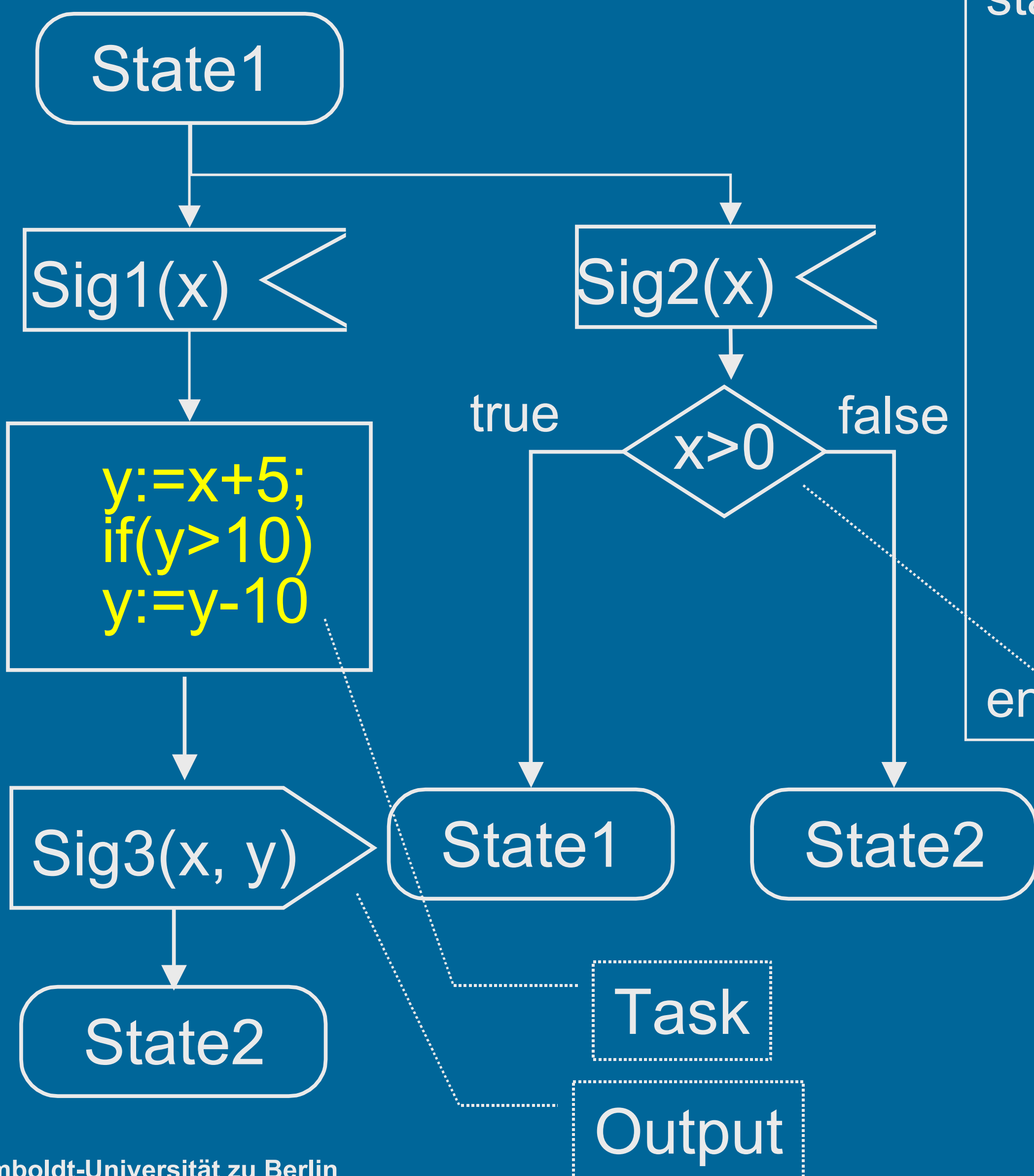
- two-way communication
  - remote variables
    - read access to variables of other agents
    - no containment relation required
  - remote procedures
    - execution of a procedure by a different agent
    - request-reply style

Simple State Machines

- behaviour of an agent
  – is specified by a state machine
- two main constituents:
  – states
    - particular condition in which an agent may consume a signal
  – transitions
    - sequence of activities triggered by the consumption of a signal

Transition Actions

- output
  - generation and addressing of signals (identification of receiver or communication path)
- task
  - sequence of simple or compound statements
  - algorithmic noatation or informal text
- decision
  - branching a transition into a series of alternative paths

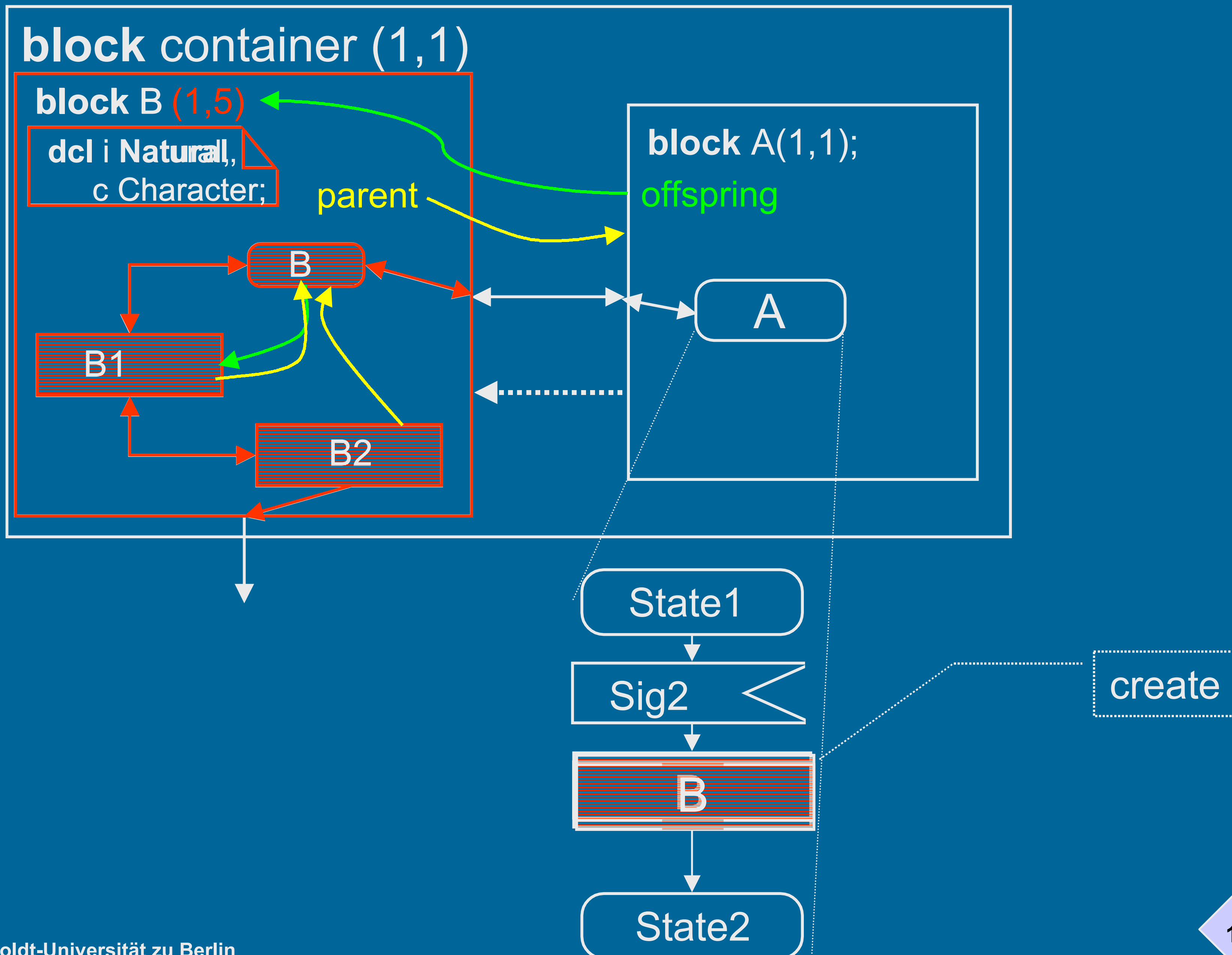



```
state State1;
    input Sig1(x);
        task { y:=x+5;
                 if(y>10)
                 y:=y-10;}
        nextstate State2;
    input Sig2(x);
        decision (x>0);
           true: nextstate State1;
           false: nextstate State2;
        enddecision
endstate;
```

Decision

Create and Stop

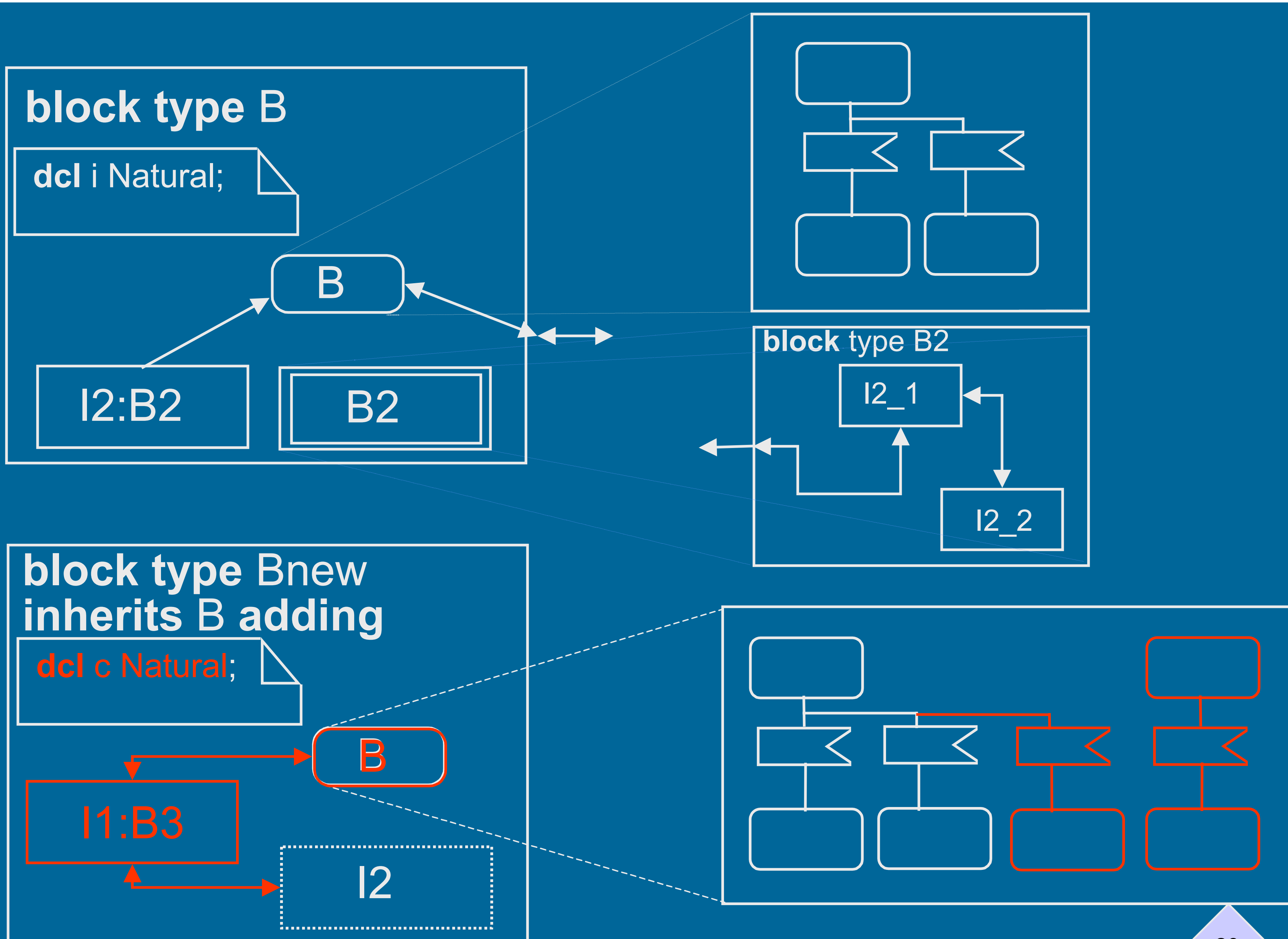
- performance of a create-request action results in the existence of a new agent instance in the indicated agent set
  - creators implicit *offspring* expression refers to new createe
  - createe´s implicit *parent* expression refers to creator
- initial internal structure will be created too

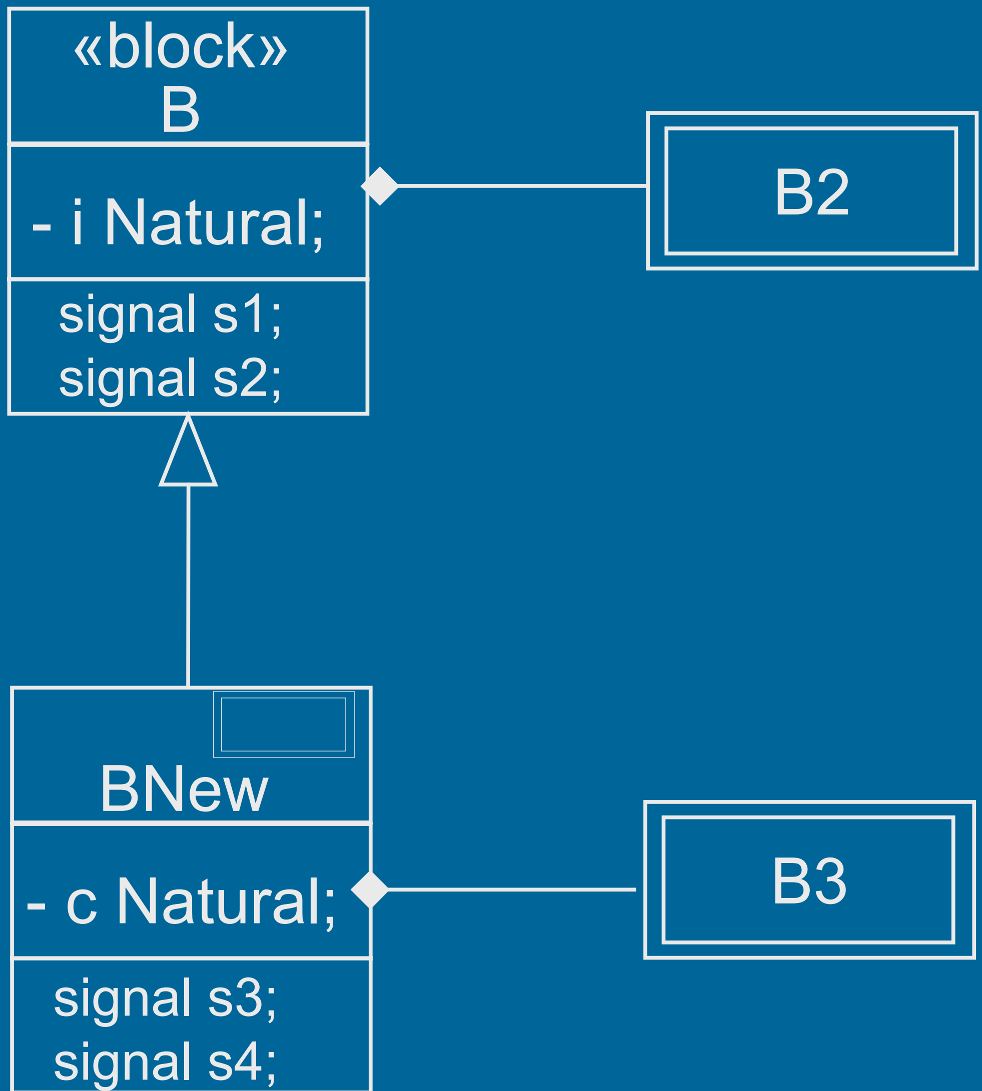

block container (1,1)
block B (1,5)
dcl i Natural,
c Character;
parent
B
B1
B2
block A(1,1);
offspring
A
State1
Sig2
B
State2
create

Object-Orientation in SDL

- structural typing concepts allow to define the properties of a set of specification elements
- kinds of structural types
  - agent type
  - state type
  - signal (type)
  - procedure (type)
  - data types and interfaces

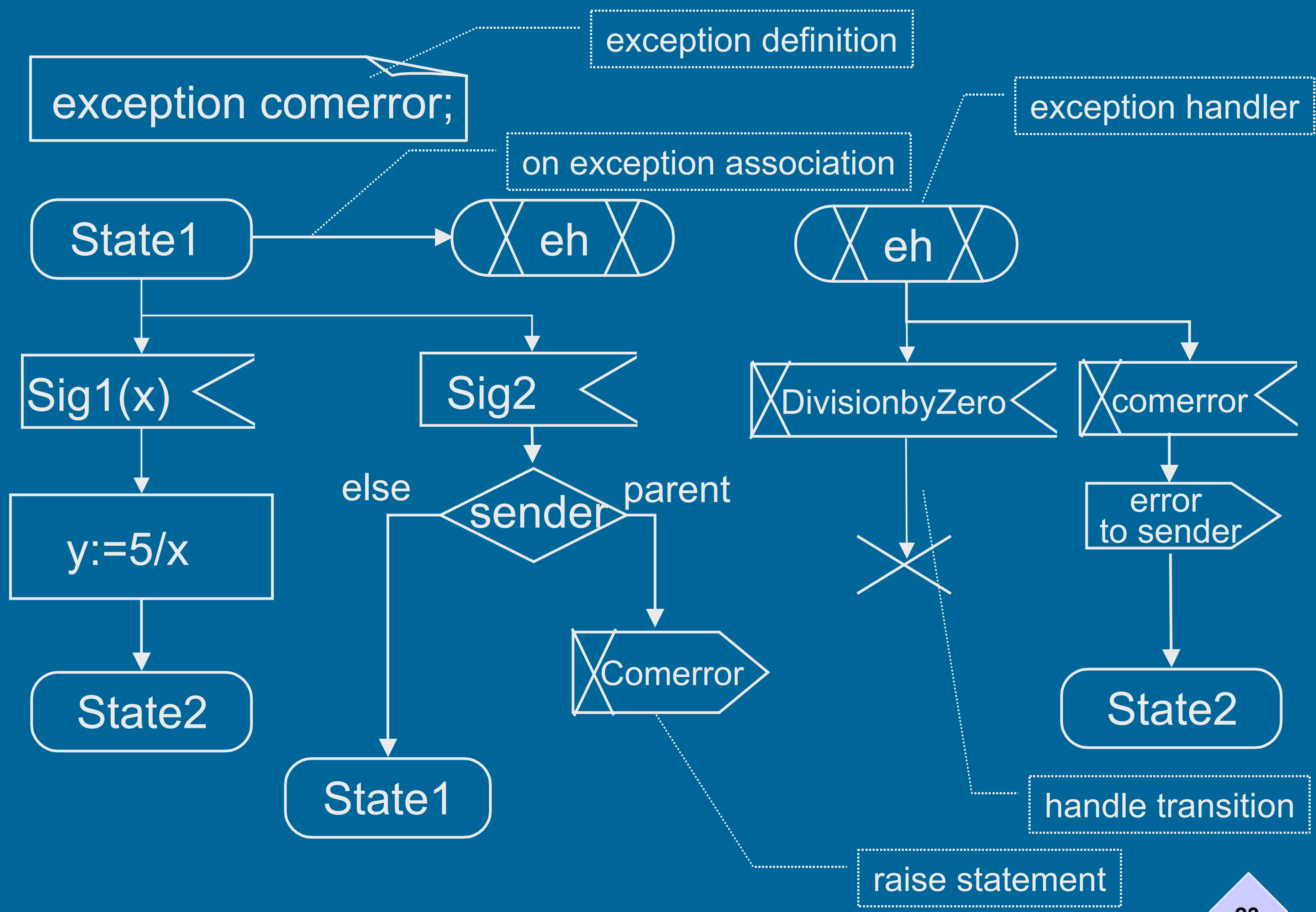
- type concept corresponds to class concept in other OO languages and notations
  - inheritance
  - virtuality
  - abstraction
  - instance definition & creation
- all instance definitions in SDL are either explicitly or implicitly based on a type


block type B
dcl i Natural;
B
I2:B2
B2
block type B2
I2_1
I2_2
block type Bnew
inherits B adding
dcl c Natural;
B
I1:B3
I2


«block»
B
- i Natural;
signal s1;
signal s2;
B2
BNew
- c Natural;
signal s3;
signal s4;
B3

Advanced State Machines

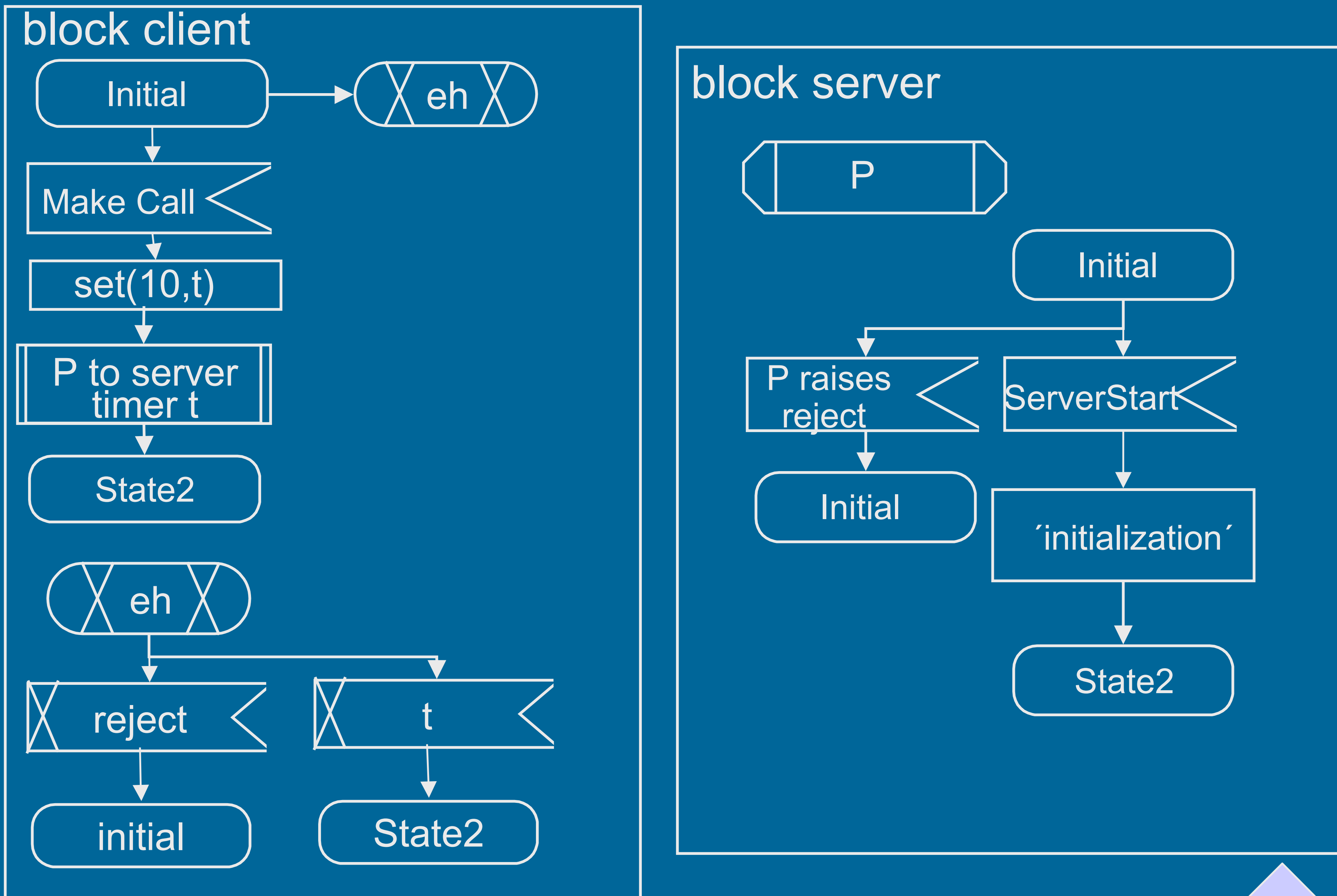
- exceptions are used to denote and handle unexpected or exceptional behaviour
  - exception: the type of cause
  - exception handler:behaviour to occur after an exception (handle-clauses)
  - onexception: attaches exception handler to a behaviour unit
  - raise: forces a transition to throw an exception


exception definition
exception comerror;
exception handler
on exception association
State1
eh
eh
Sig1(x)
Sig2
DivisionbyZero
comerror
else
sender
parent
error
to sender
y:=5/x
Comerror
State2
State1
State2
handle transition
raise statement

Remote Procedures

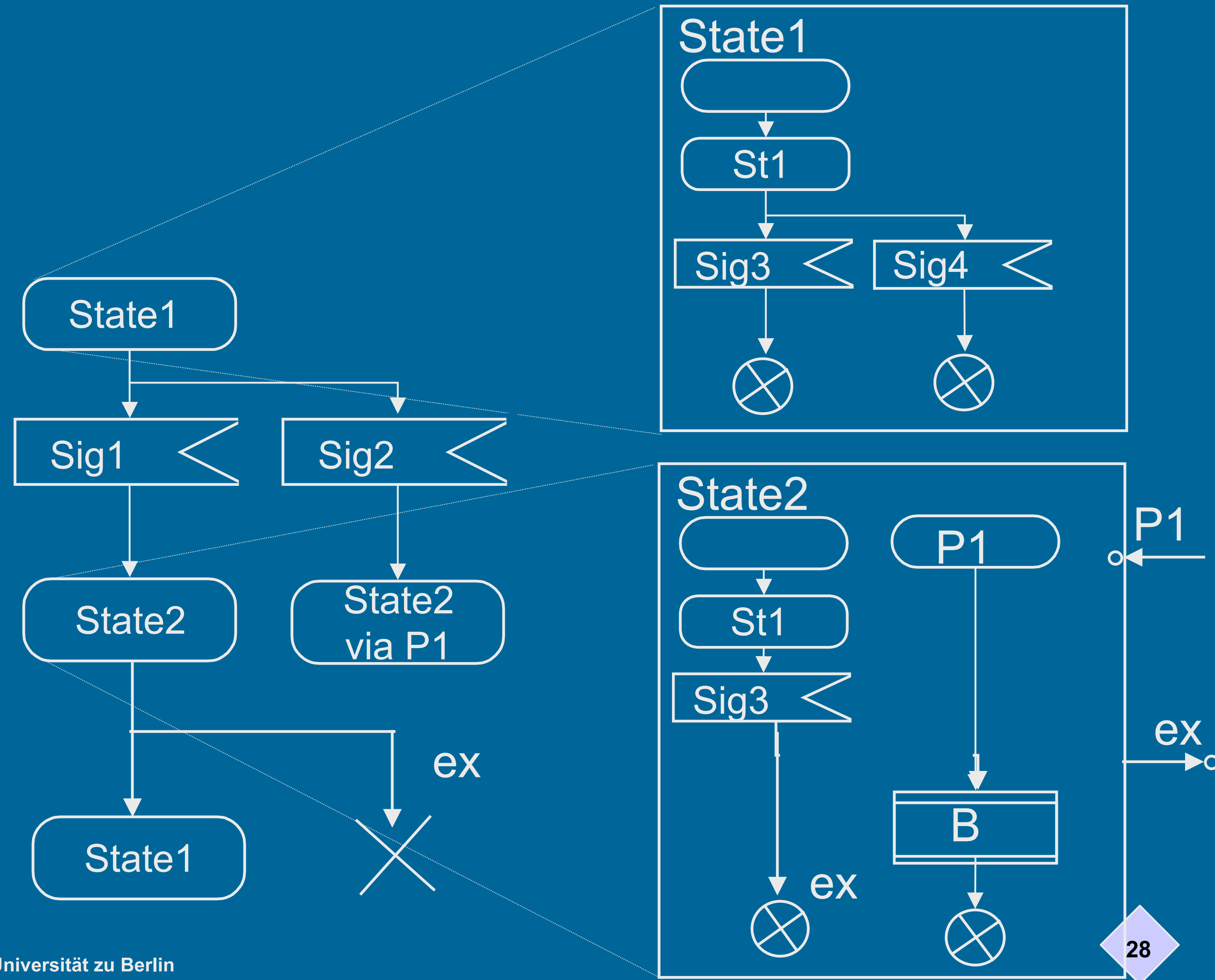
- an agent can make its procedures available for other agents
  – remote procedures
  – realized by two-way communication between caller and server
- after a call to a remote procedure the caller is blocked until he receives the procedure return from the server

- remote procedure call may deadlock
  - can be prevented by an associated timer, which raises an exception
- server accepts calls for remote procedures in any state
  - execution may be deferred by *save*
  - execution may be rejected by *input <p> raise <deny>*
- exceptions raised by the remote procedure are raised at client and server side


block client
Initial
eh
Make Call
set(10,t)
P to server
timer t
State2
eh
reject
t
initial
State2
block server
P
Initial
P raises
reject
ServerStart
Initial
´initialization´
State2

Composite States

- composite states are a means to hierarchically structure state machines
  - nesting of states
  - agent can be in more than one state at a time
  - Harel´s state charts
- composite state is itself a sub-state machine
- state machine of an agent is in fact a top-level composite state


State1
Sig1
Sig2
State2
State2
via P1
ex
State1
State1
St1
Sig3
Sig4
State2
P1
P1
St1
Sig3
ex
ex
B

Virtual Behaviour Elements

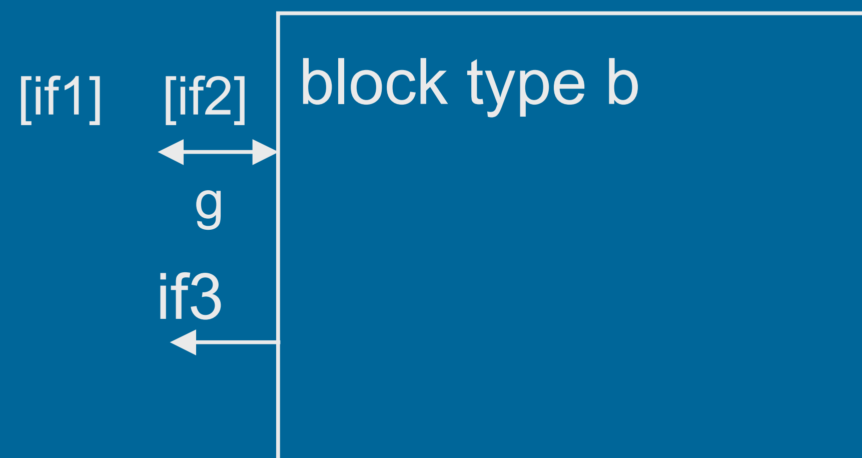
- allow the redefinition or replacement of behaviour elements in a type specialisation
- redefinition and finalisation similar to structural elements
- available for
  - procedures
  - transitions
  - exception handle transitions

Interface

- pure typing concept used for typed communication between agents
- interface definition groups and names a set of
  - remote variable
  - remote procedure
  - signal definitions
- gates and channels paths can be typed by interfaces

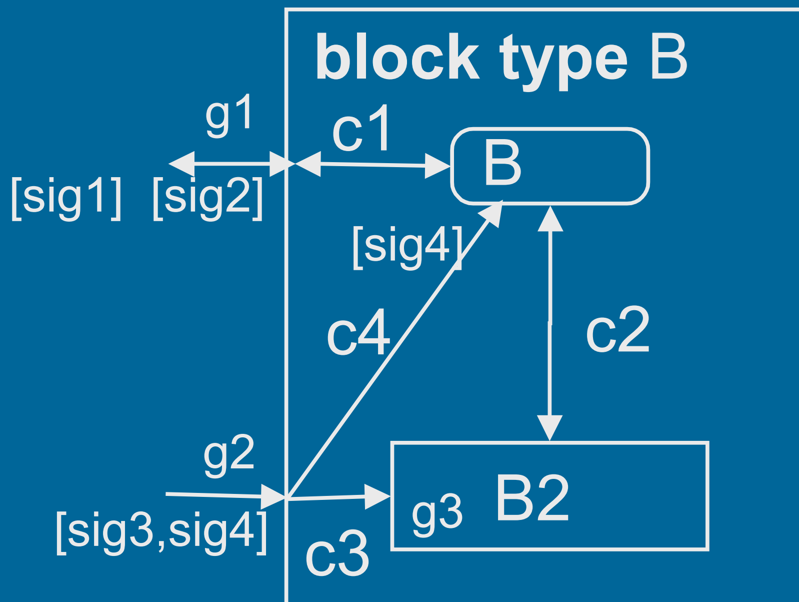
```
interface if1;
    signal sig1;
    procedure P;
    dcl I Natural;
endinterface;

signal sig2,sig3;
interface if2;
    use sig2, sig3;
endinterface;
```

```
interface if3
    inherits if1,if2;
```

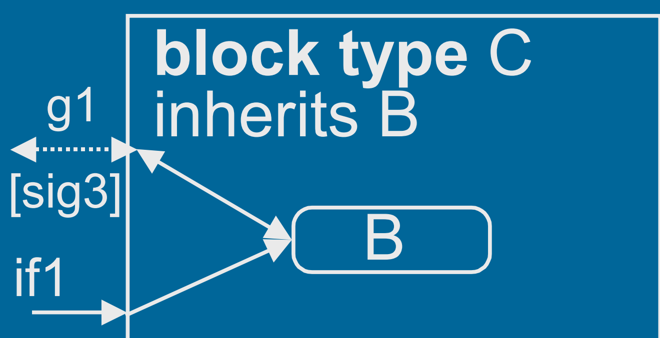


Agent Implicit Interface

- each agent and agent type introduces an implicit interface
  - same name as agent (type)
- contains all
  - signals accepted by the agents state machine
  - remote variables/procedures provided by agents state machine
- inherits all interfaces on gates connected to agents state machine

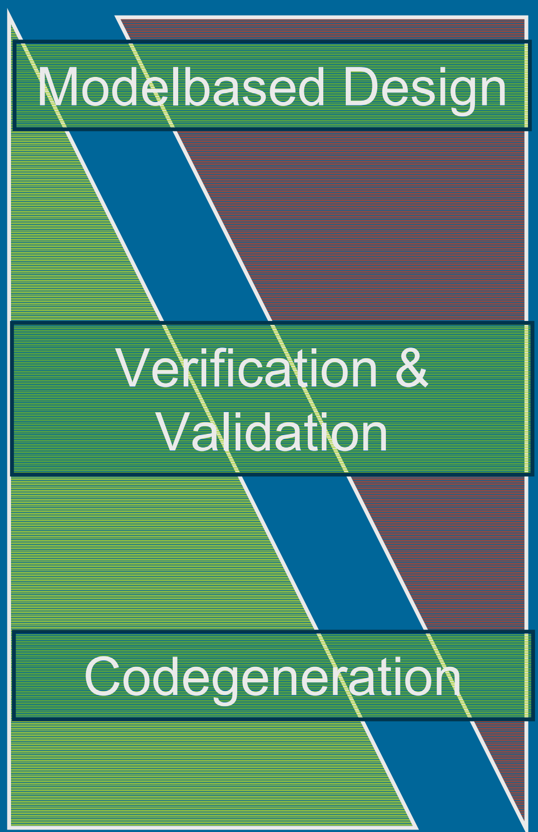


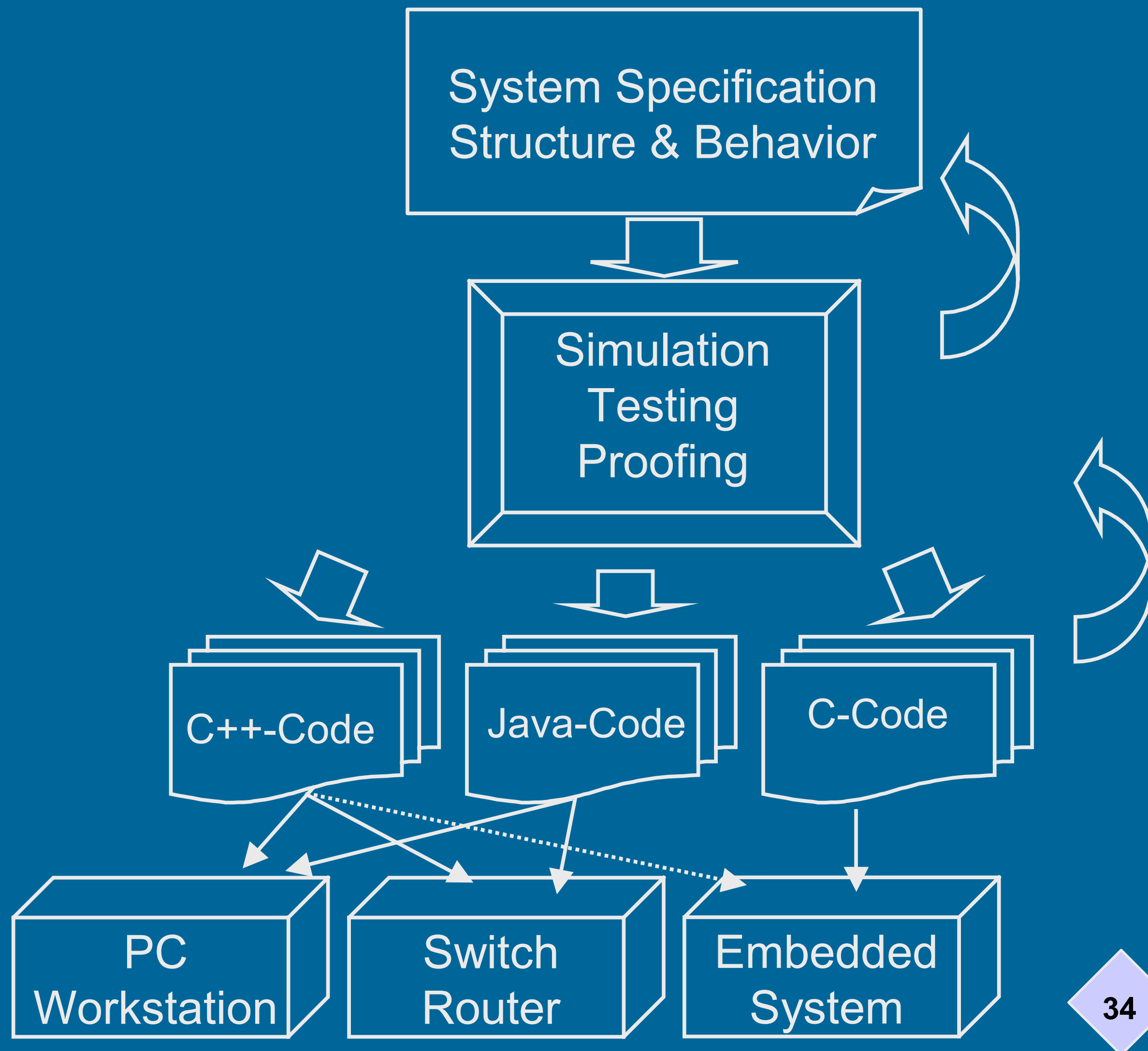

```
interface B
    use sig2,sig4;
endinterface
```



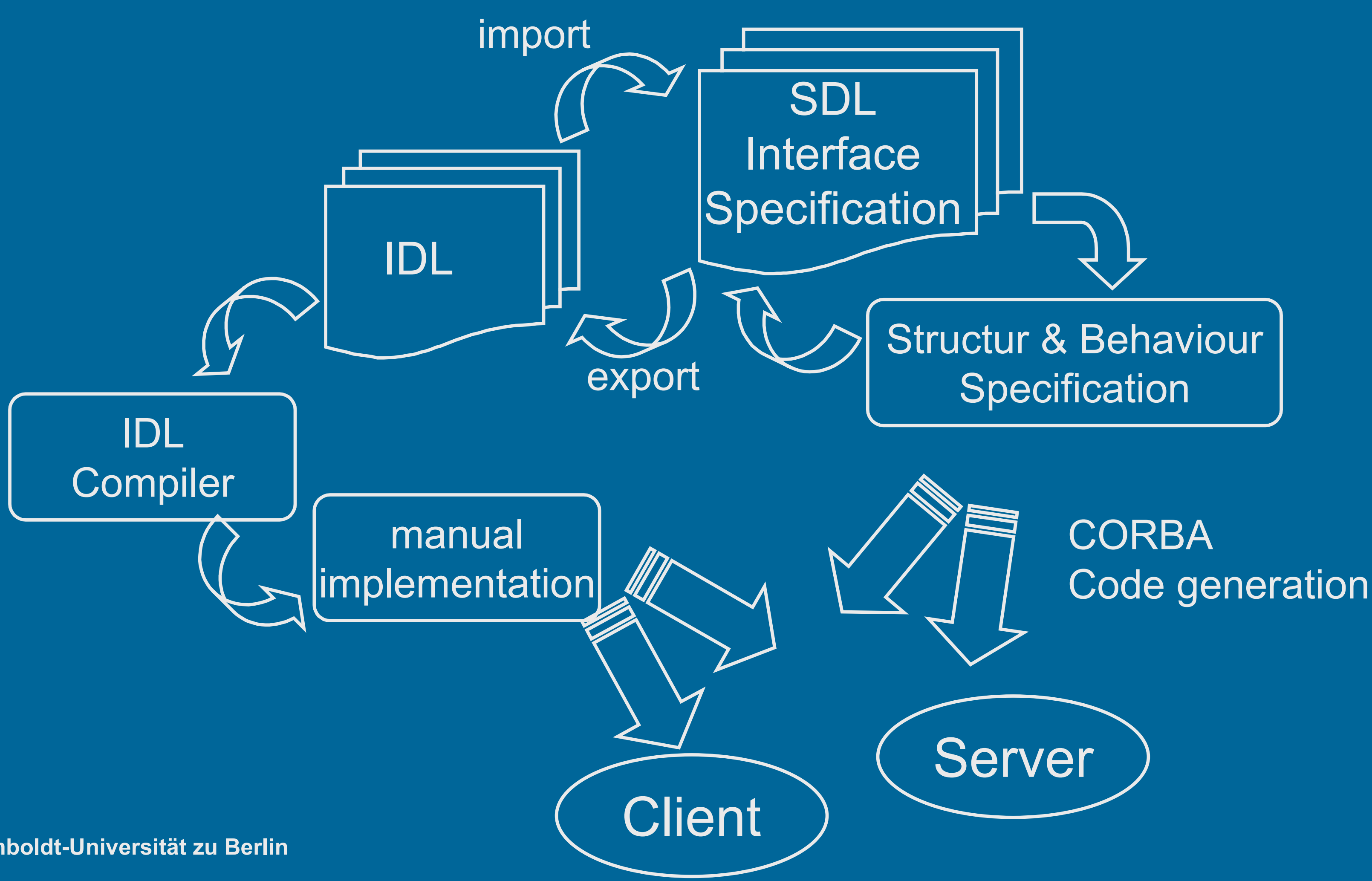

```
interface C
inherits B, if1;
    use sig3;
endinterface
```

Development with SDL



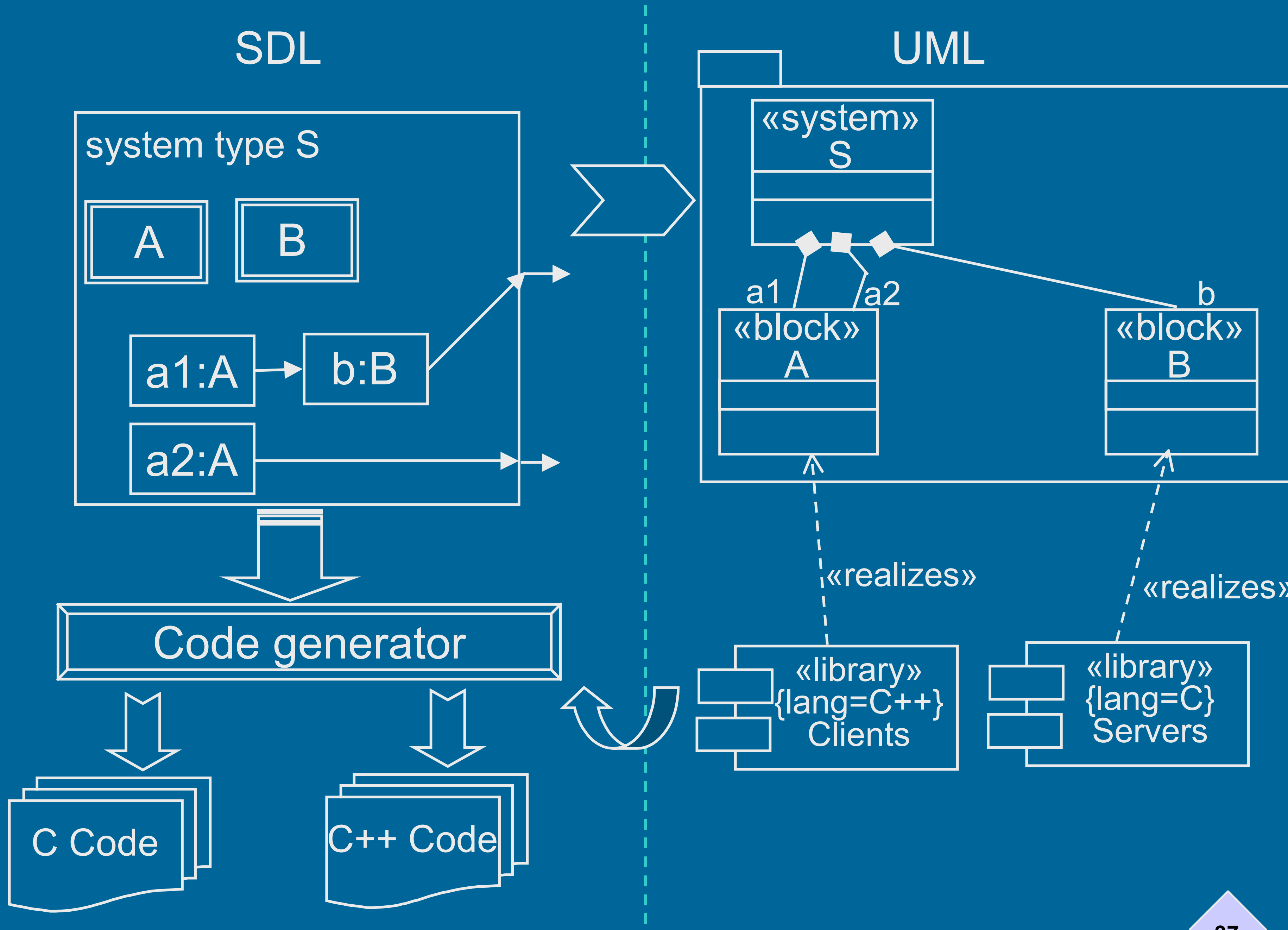


Integrating with CORBA



SDL-UML-Profile

- provides a reflection of SDL-concepts in UML
- defines a series of stereotypes
  - «block», «process», «signal»,...
- defines a series of wellformednes rules
- allows a mapping of SDL type hierarchies to UML structure and behaviour diagrams
- enables combined usage of SDL and UML


SDL
UML
system type S
A
B
a1:A
b:B
a2:A
«system»
S
a1
a2
b
«block»
A
«block»
B
«realizes»
«realizes»
Code generator
«library»
{lang=C++}
Clients
«library»
{lang=C}
Servers
C Code
C++ Code

Tool Support and Application

- commercial tools
  - initial support of SDL-2000
  - integrate design, test and codegeneration
    - Telelogic TAU (SDT, Geode), Cinderella
- proprietary and academic tools
  - code generation and verification tools
    - SITE,...
- applications
    - telecommunication systems (ISDN, GSM, UMTS)
    - software for mobil phones, car radios,
    - automotive and aerospace systems

Summary

- SDL is well positioned in reactive systems design (esp. telecommunications systems)
- SDL-2000 opens up to new application domains
  - extended communication means and interfaces for CORBA systems
  - advanced structural and behavioural concepts for embedded systems design
  - smooth modelbased integration with UML allows to choose adequate design technology
  - enhances CORBA and UML based design with verification and validation technologies

Further Information

- SDL-2000 tutorial slides

  www.informatik.hu-berlin.de/~holz/SDLTutorial/SAMTutorialFinal.htm
  holz@informatik.hu-berlin.de
- ITU standards and recommendations
  - www.itu.ch or www.itu.int/itudoc/itu-t/approved/z/index.html
- SDL Forum Society
  - www.sdl-forum.org
- conferences and workshops
  - bi-annual SDL-Forum - next Copenhagen June 2001
  - bi-annual SAM-Workshops – next 2002