Institute for Software Integrated Systems

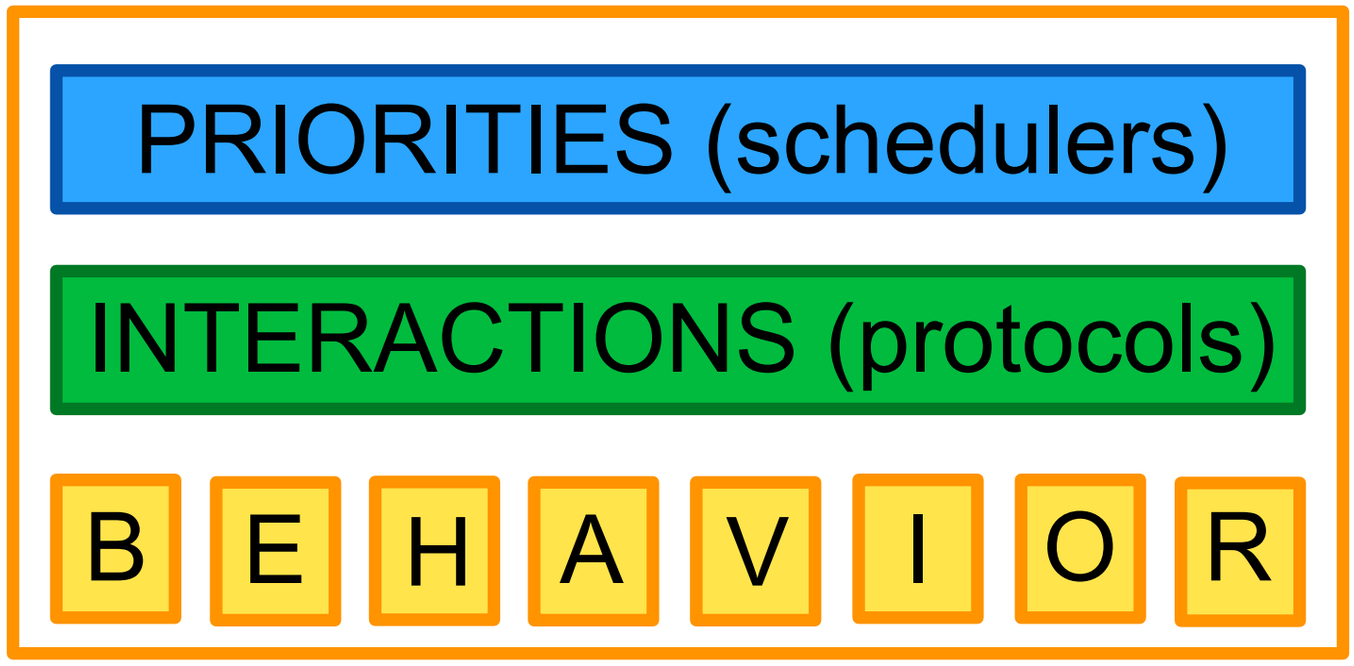
VANDERBILT UNIVERSITY

A Design Studio for Modeling, Analyzing, and Generating Systems with BIP

Anastasia Mavridou

October 24, 2017

Behavior-Interaction-Priority



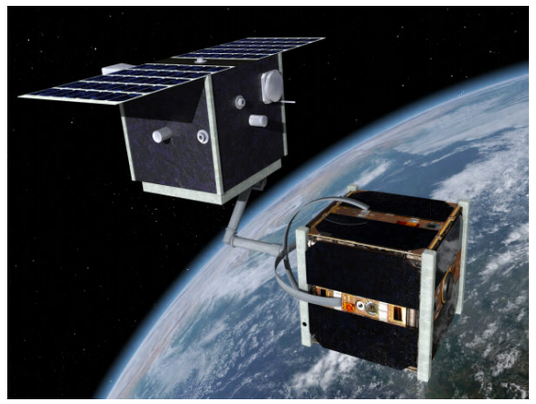
BIP allows to **compositionally**

- develop correct-by-construction applications
- analyze existing applications

Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. "Rigorous component-based system design using the BIP framework." IEEE software 28, no. 3 (2011): 41-48.

BIP Application Examples

- Development of correct-by-construction satellite software
  - 49 safety properties enforced by construction
  - compositional verification of deadlock-freedom with D-Finder:
    - State space size: $> 3^{10} \times 4$
    - Verification time: < 2 minutes

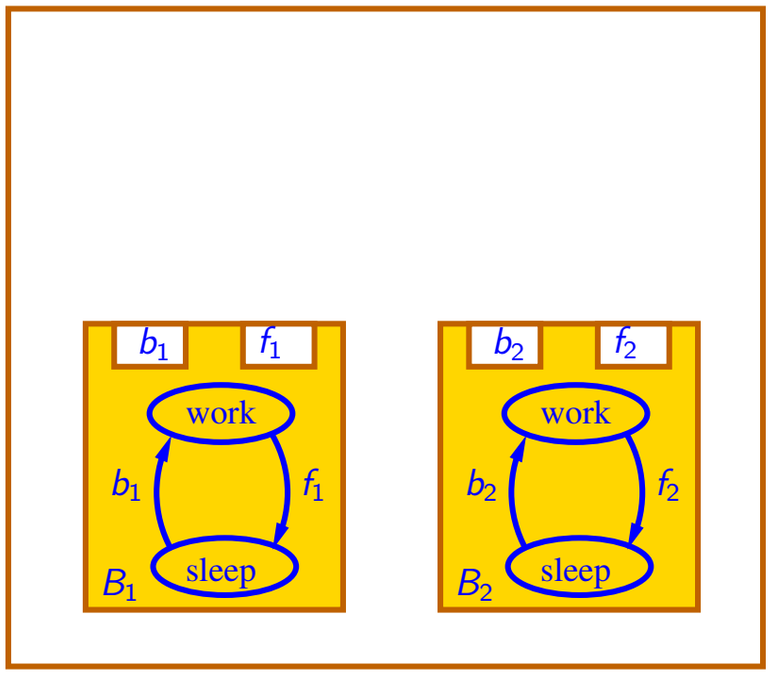


- Development of the Dala robot controller
  - > 250,000 lines of code
  - example results of deadlock-freedom analysis with D-Finder:

Module	BIP LoC	C/C++ LoC	Estimated state space size	Verification time (minutes)
LaserRF	5,343	51,653	$2^{20} \times 3^{29} \times 34$	1:22
Rflex	8,244	57,442	$2^{34} \times 3^{35} \times 1045$	9:39
Antenna	1,645	16,501	$2^{12} \times 3^{9} \times 13$	0:14

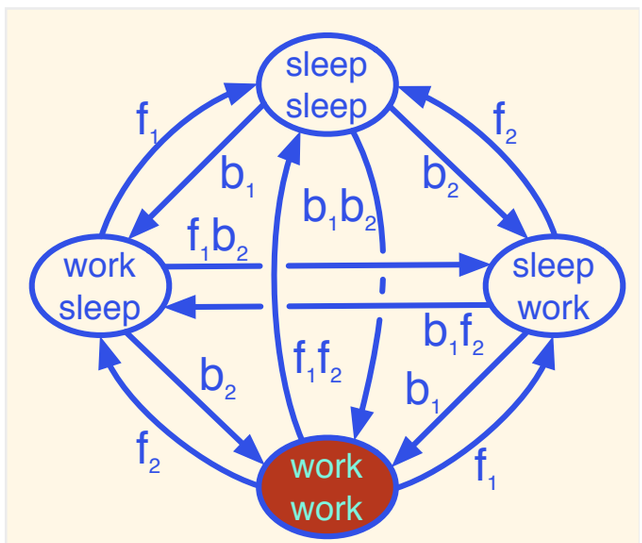
BIP-by-example

Safety property:
Mutual exclusion



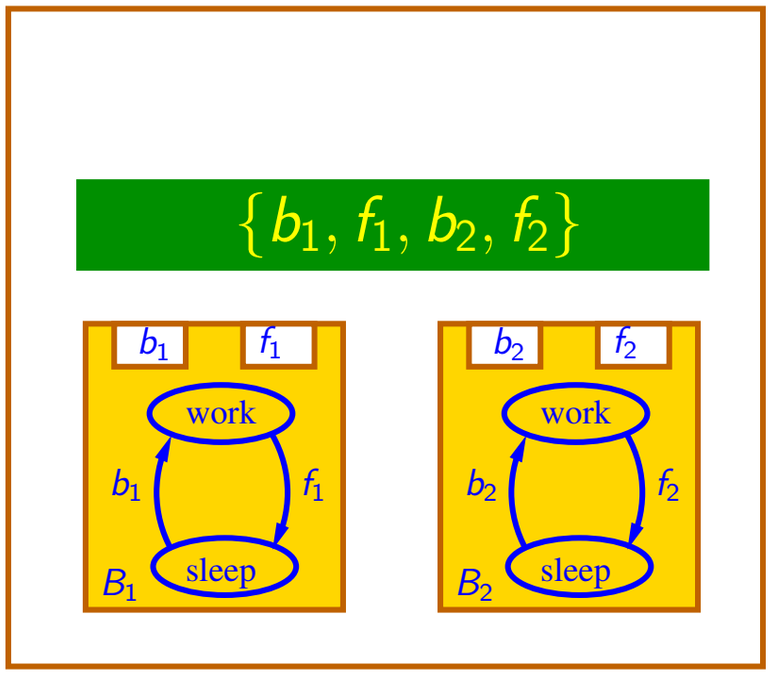
$$\{b_1, b_2, f_1, f_2, b_1b_2, b_1f_2, f_1b_2, f_1f_2\}$$

No restrictions



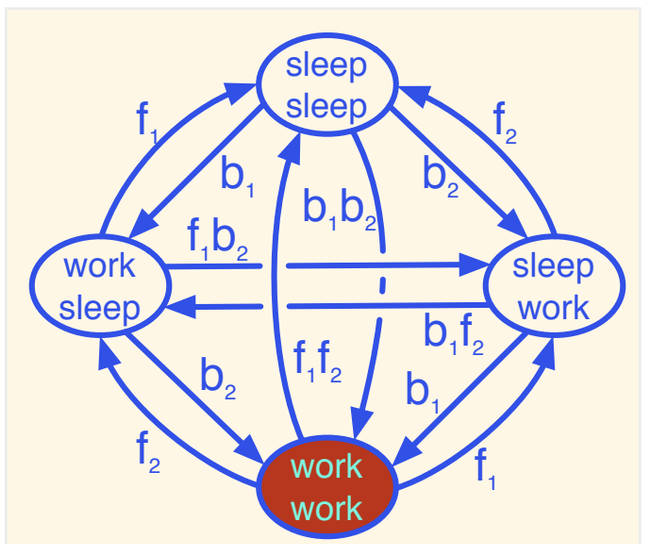
BIP-by-example

Safety property:
Mutual exclusion

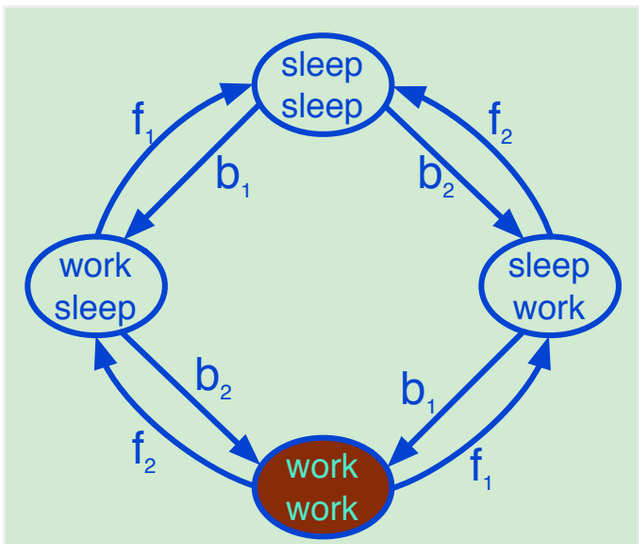


$$\{b_1, b_2, f_1, f_2, b_1b_2, b_1f_2, f_1b_2, f_1f_2\}$$

No constraints

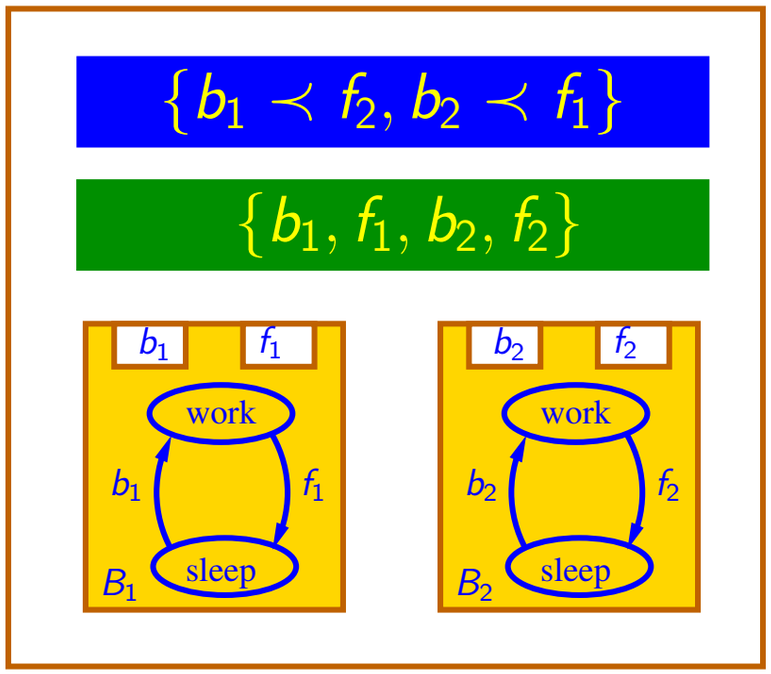


Interaction Constraints



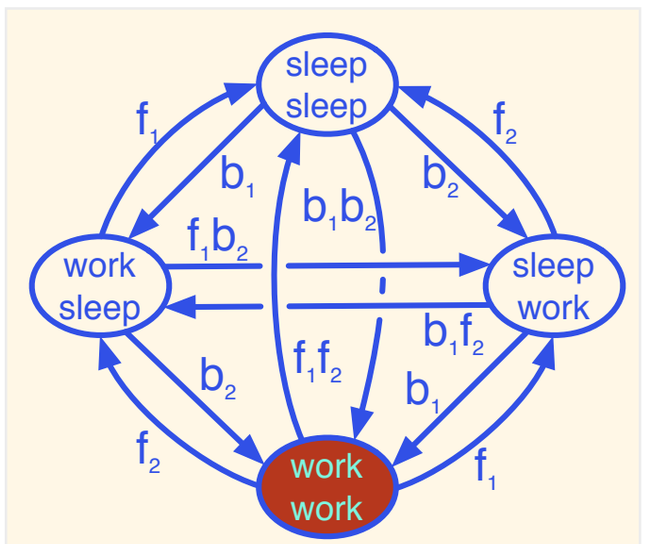
BIP-by-example

Safety property:
Mutual exclusion

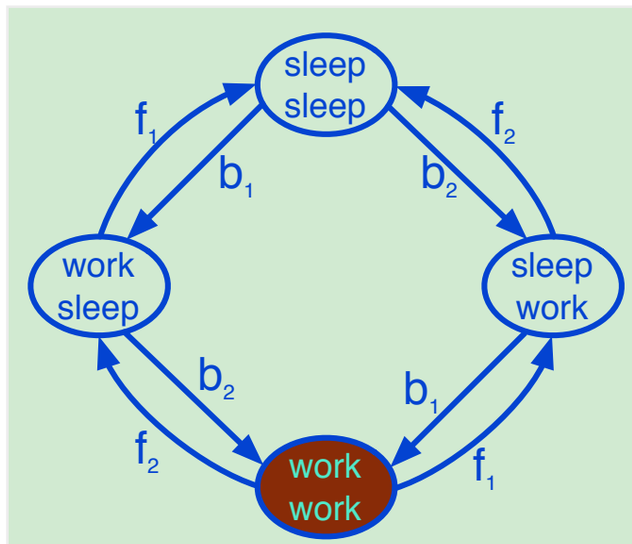


$$\{b_1, b_2, f_1, f_2, b_1b_2, b_1f_2, f_1b_2, f_1f_2\}$$

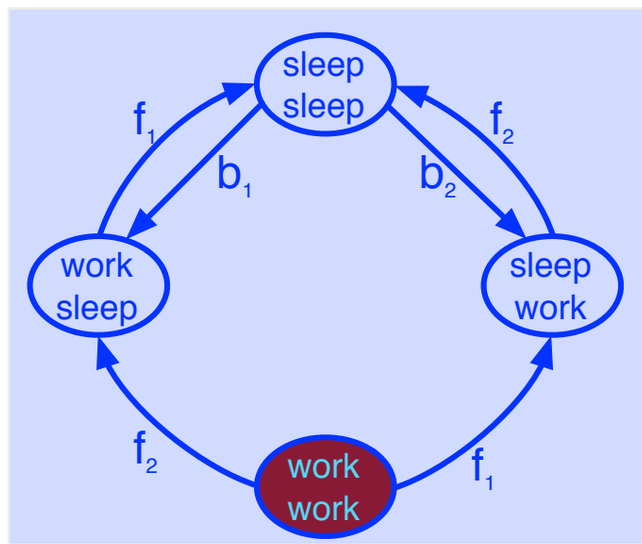
No constraints



Interaction Constraints



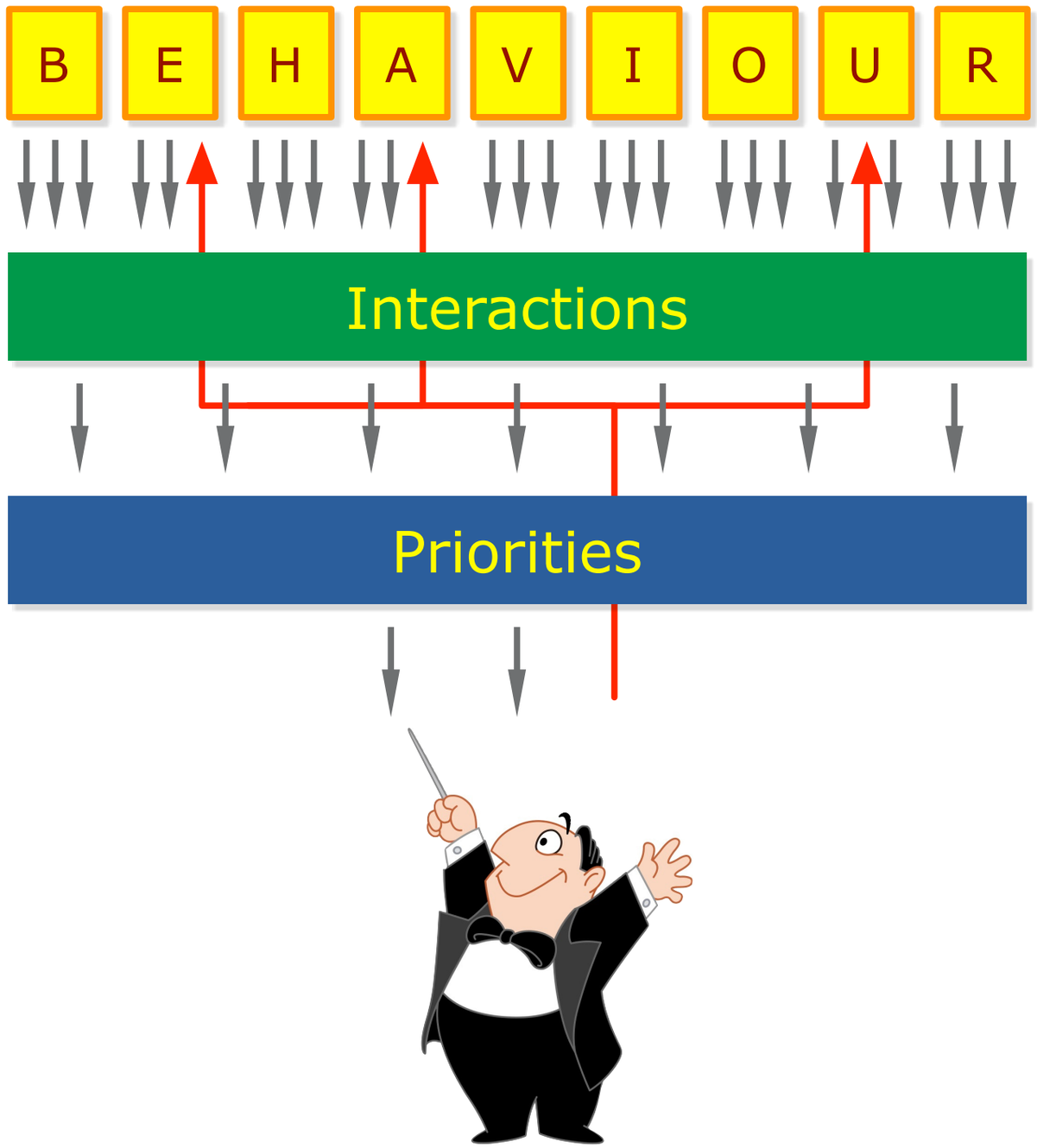
Priority Constraints



Engine-based Execution

1. Components notify the BIP-engine about enabled transitions

2. The BIP-engine picks an interaction and instructs the components

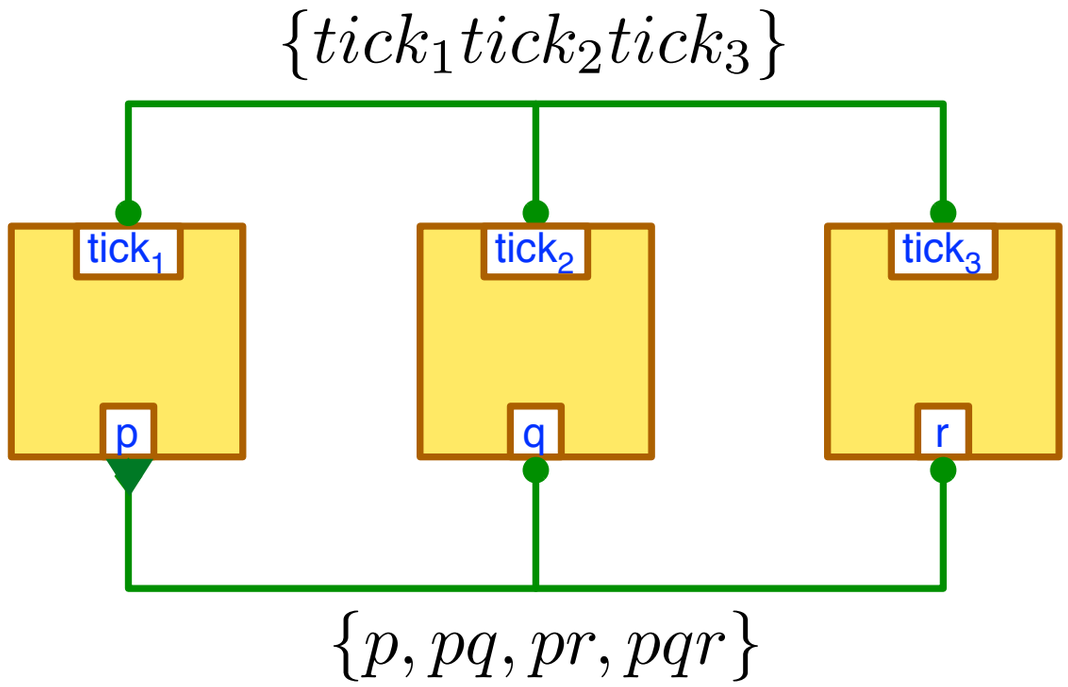


BIP Code Example

```
initial to  WRITE_BUFFER do{  }

 on write from WRITE_BUFFER to WAIT
 on wait from WAIT to STATUS_WRITE
 on contin from STATUS_WRITE to WRITE_BUFFER
 on fail from STATUS_WRITE to WRITE_BUFFER
 on ok_write from STATUS_WRITE to DONE
 on finish from DONE to WRITE_BUFFER


end

atomic type memory_library
        export port  syncPort  setWrite
        export port  syncPort  checkCRC
        export port  syncPort  setRead

        place S0

initial to  S0 do{  }
 on setWrite from S0 to S0
 on setRead from S0 to S0
 on checkCRC from S0 to S0
end

compound type CubETH

        component sMutex MEM_MUX
        component flash_memory_readActionFlowWithAbort MEMRD_ACTFLAB
        component flash_memory_writeActionFlowWithAbort MEMWR_ACTFLAB
        component memory_library MEMLIB
        component flash_memory_readModeManager MEMRD_MODMNG
        component flash_memory_writeModeManager MEMWR_MODMNG

        connector RDV3 flash_memory_setRead2 ( MEM_MUX.take ,MEMLIB.setRead ,MEMRD_MODMNG.read   )
        connector RDV2 flash_memory_setRead1 ( MEM_MUX.take ,MEMLIB.setRead   )
        connector RDV2 flash_memory_read ( MEMRD_MODMNG.contin ,MEMRD_ACTFLAB.read   )
        connector RDV3 flash_memory_read_fail2 ( MEMRD_ACTFLAB.fail ,MEM_MUX.release ,MEMRD_MODMNG.done   )
        connector RDV2 flash_memory_read_fail1 ( MEMRD_ACTFLAB.fail ,MEM_MUX.release   )
        connector RDV3 flash_memory_ok_read2 ( MEMRD_ACTFLAB.ok_read ,MEM_MUX.release ,MEMRD_MODMNG.done   )
        connector RDV2 flash_memory_ok_read1 ( MEMRD_ACTFLAB.ok_read ,MEM_MUX.release   )
        connector RDV2 flash_memory_checkCRC ( MEMRD_ACTFLAB.check_CRC ,MEMLIB.checkCRC   )
        connector SINGLE flash_memory_bad_CRC ( MEMRD_ACTFLAB.bad_CRC   )
        connector RDV2 flash_memory_write ( MEMWR_MODMNG.contin ,MEMWR_ACTFLAB.write   )
        connector RDV3 flash_memory_setWrite2 ( MEM_MUX.take ,MEMLIB.setWrite ,MEMWR_MODMNG.write   )
        connector RDV2 flash_memory_setWrite1 ( MEM_MUX.take ,MEMLIB.setWrite   )
        connector RDV3 flash_memory_write_fail2 ( MEMWR_ACTFLAB.fail ,MEM_MUX.release ,MEMWR_MODMNG.done   )
        connector RDV2 flash_memory_write_fail1 ( MEMWR_ACTFLAB.fail ,MEM_MUX.release   )
        connector RDV3 flash_memory_ok_write2 ( MEMWR_ACTFLAB.ok_write ,MEM_MUX.release ,MEMWR_MODMNG.done   )
        connector RDV2 flash_memory_ok_write1 ( MEMWR_ACTFLAB.ok_write ,MEM_MUX.release   )
        connector SINGLE MEMRD_ACTFLAB_finish ( MEMRD_ACTFLAB.finish   )
        connector SINGLE MEMWR_ACTFLAB_wait ( MEMWR_ACTFLAB.wait   )
        connector SINGLE MEMWR_ACTFLAB_contin ( MEMWR_ACTFLAB.contin   )
        connector SINGLE MEMWR_ACTFLAB_finish ( MEMWR_ACTFLAB.finish   )

        priority flash_memory_setRead1_after_flash_memory_setRead2          flash_memory_setRead1 < flash_memory_setRead2
        priority flash_memory_read_fail1_after_flash_memory_read_fail2      flash_memory_read_fail1 < flash_memory_read_fail2
        priority flash_memory_ok_read1_after_flash_memory_ok_read2          flash_memory_ok_read1 < flash_memory_ok_read2
        priority flash_memory_setWrite1_after_flash_memory_setWrite2        flash_memory_setWrite1 < flash_memory_setWrite2
        priority flash_memory_write_fail1_after_flash_memory_write_fail2        flash_memory_write_fail1 < flash_memory_write_fail2
```

Design Studio: Graphical Language

Graphical connectors to represent component interaction

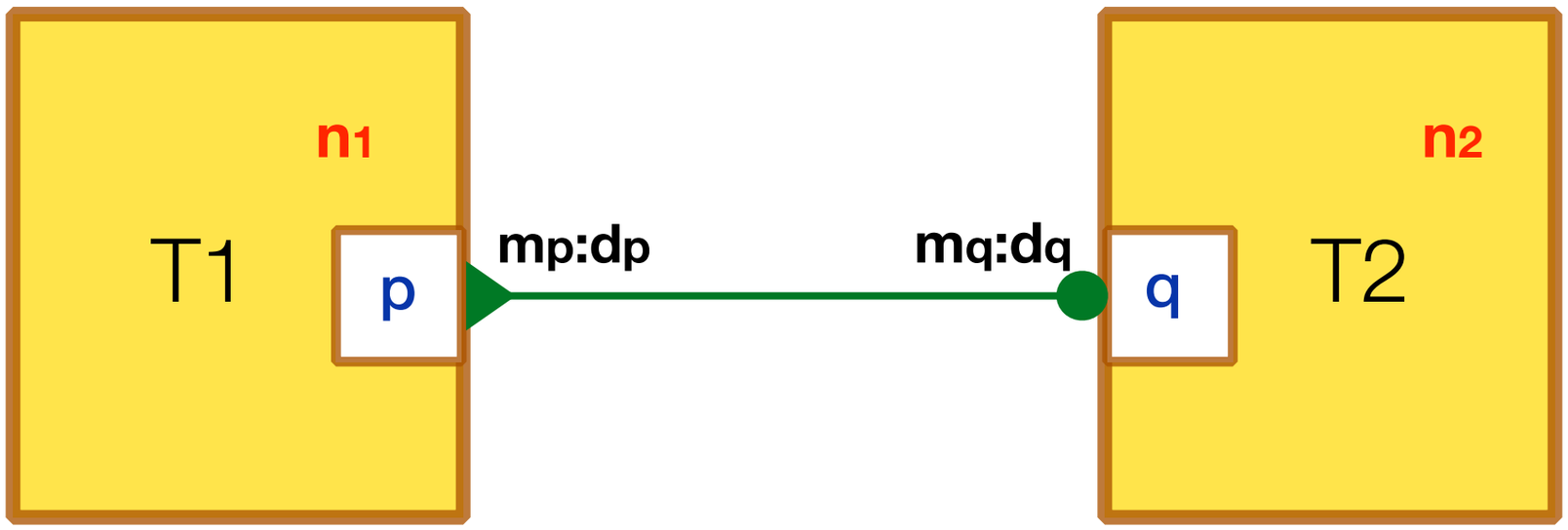


- Connectors are tree-like structures
  - connector ends of two types
    - *Triggers* (triangles) — nodes that do not require interaction
    - *Synchrons* (bullets) — nodes that require interaction with others

Design Studio: Parameterized Models

Systems are built from multiple instances of the same type

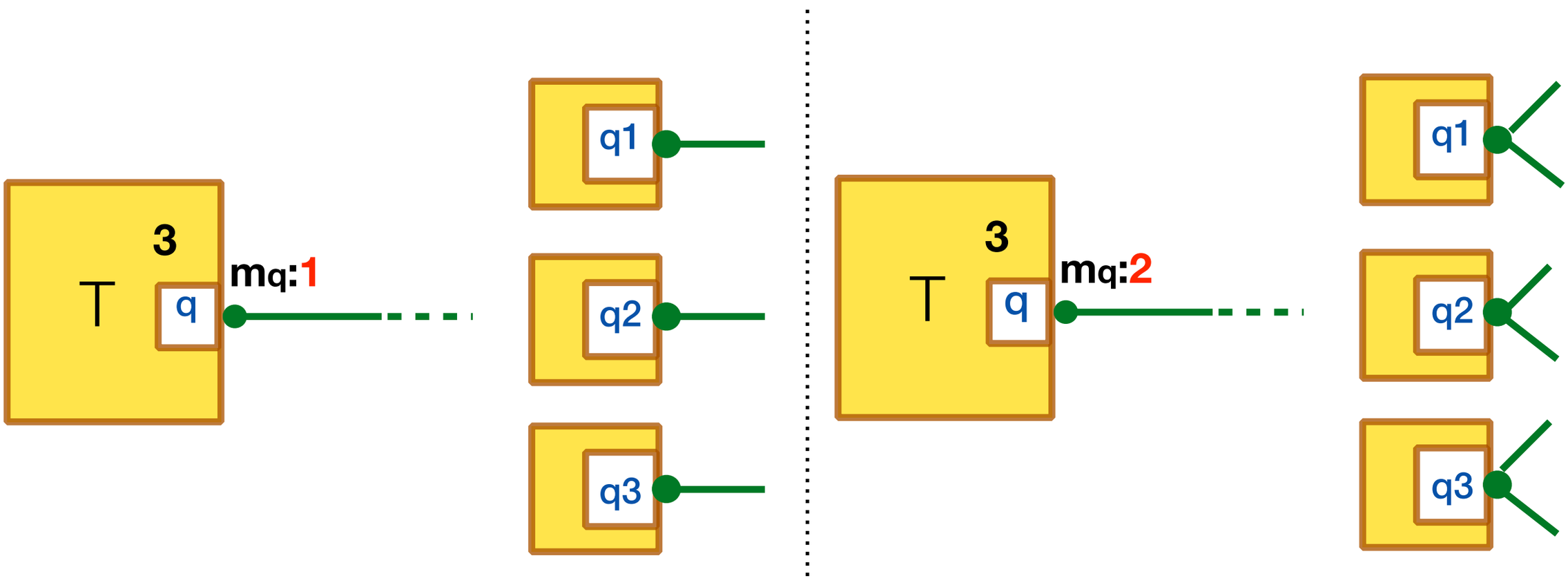
- Component types
- Define interactions between component types



cardinality: number of instances of each component type

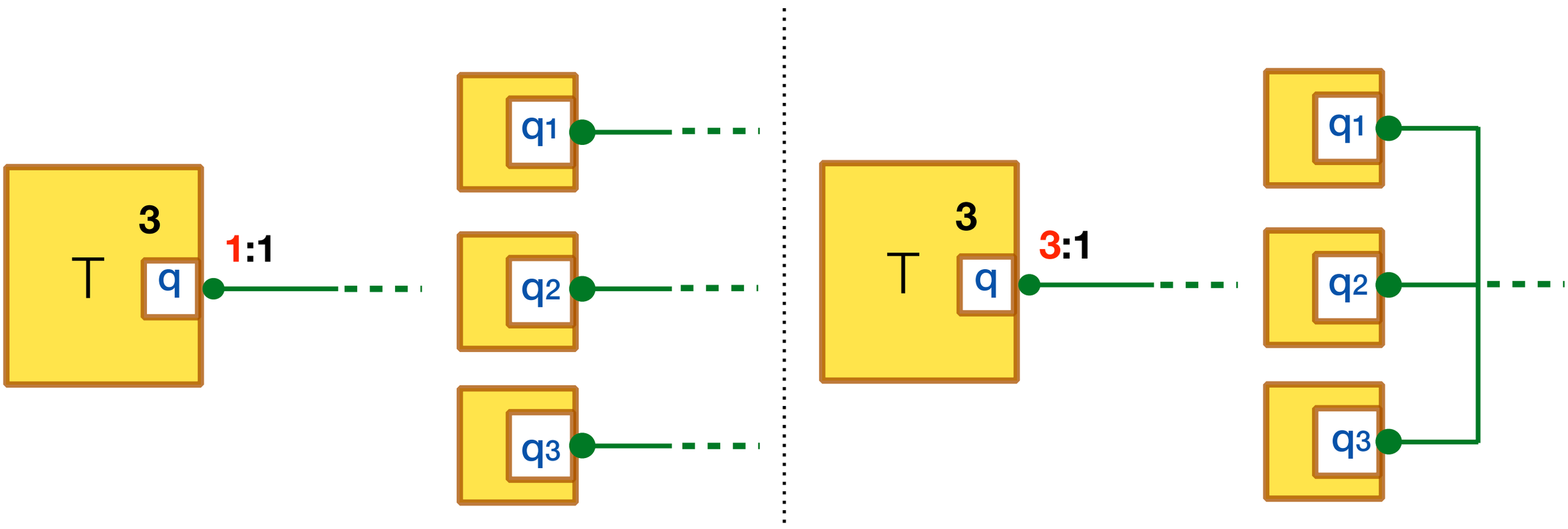
Design Studio: Parameterized Models

Degree: number of connector instances attached to each port instance

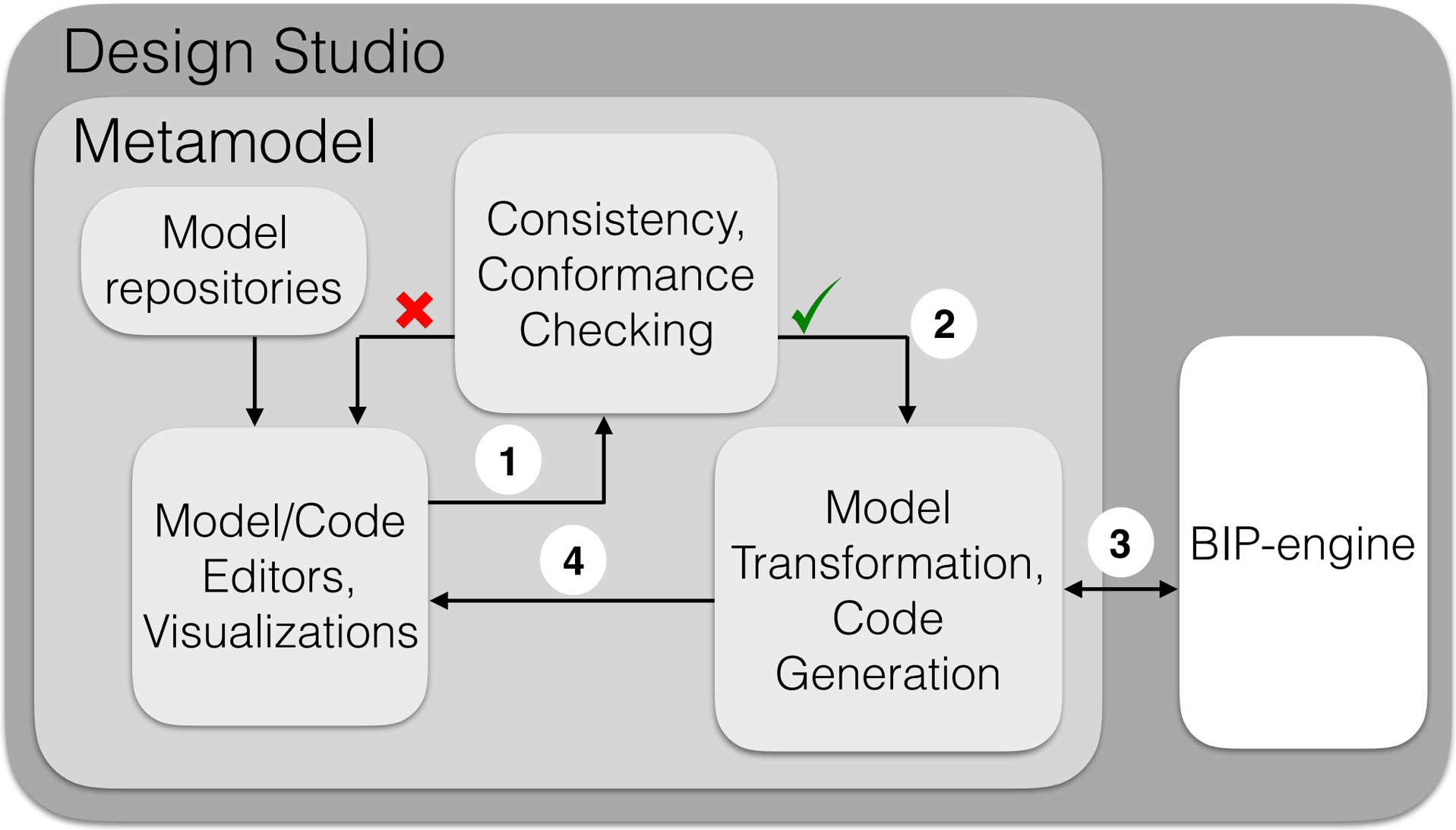


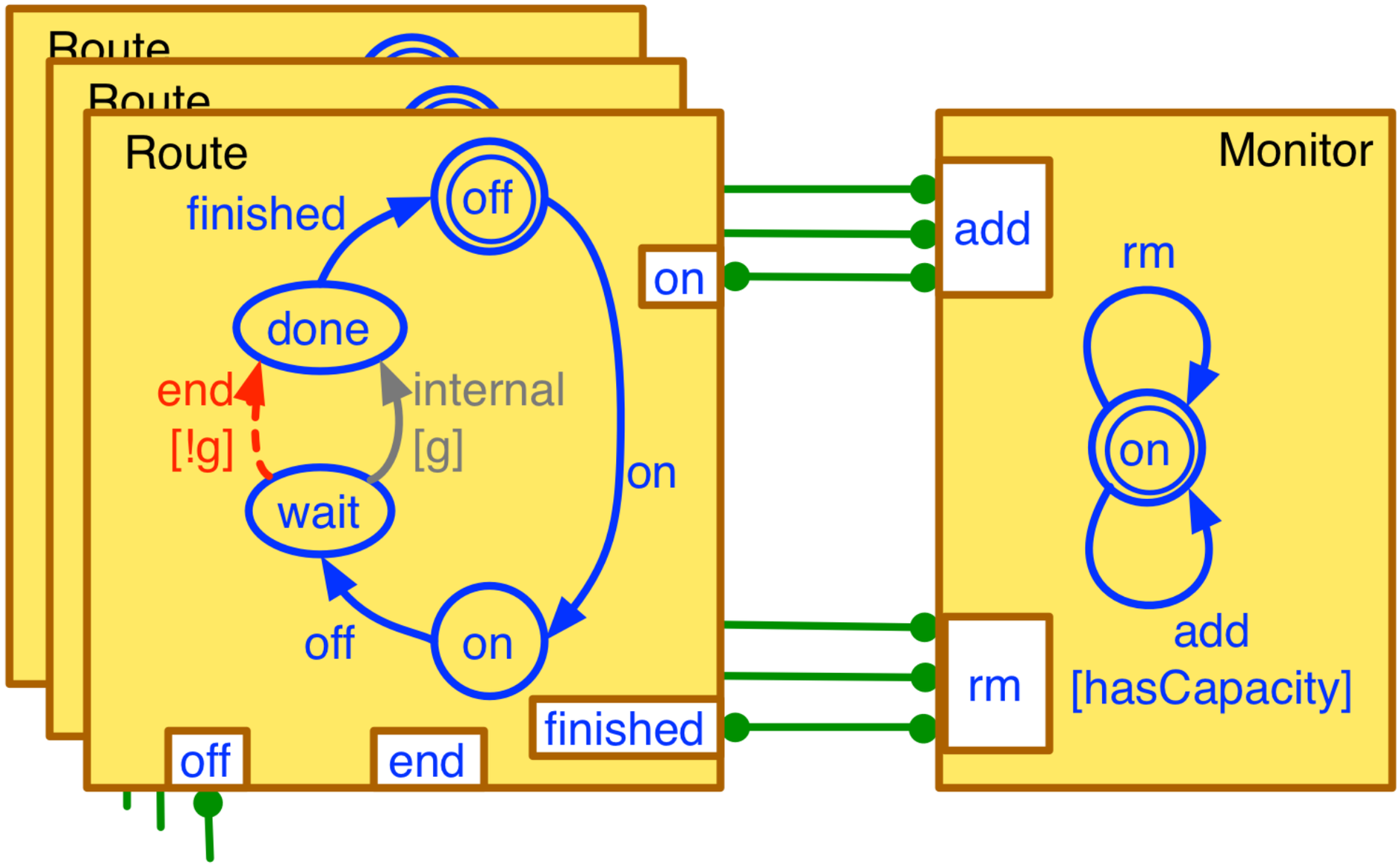
Design Studio: Parameterized Models

Multiplicity: number of port instances of the same type that participate in a connector instance



BIP Design Studio





Hands-on BIP

Modeling Camel Routes

Conclusion

- The BIP design studio
  - web-based, version-controlled, collaborative
  - open source: github.com/anmavrid/webgme-bip
  - allows coping with modeling complexity and size
  - formal semantics
  - includes:
    - dedicated editors for code, interaction and behavior editing
    - code generation plugins
    - consistency, conformance checking mechanisms
    - integration of the BIP-engine and visualization of its output

Related Bibliography

- Mavridou, Anastasia, Joseph Sifakis, and Janos Sztipanovits. “A WebGME design studio for architecture-based design with BIP.” 17th High Conference Software and Systems Conference (HCSS), Annapolis, MD (2017).

- Mavridou, Anastasia, Joseph Sifakis, and Janos Sztipanovits. “Architecture-based design and analysis with BIP.” Safe and Secure Systems and Software Symposium (S5). Dayton, OH (2017).

- Basu, Ananda, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. "Rigorous component-based system design using the BIP framework." IEEE software 28, no. 3 (2011): 41-48.

- Bliudze, Simon, Anastasia Mavridou, Radoslaw Szymanek, and Alina Zolotukhina. "Exogenous coordination of concurrent software components with JavaBIP.” Software: Practice and Experience (2017).

- Mavridou, Anastasia, Eduard Baranov, Simon Bliudze, and Joseph Sifakis. "Architecture diagrams: A graphical language for architecture style specification.” 9th Interaction and Concurrency (2016).