Programming Dynamic Reconfigurable Systems with DR-BIP

Rim El Ballouli Saddek Bensalem Marius Bozga Joseph Sifakis

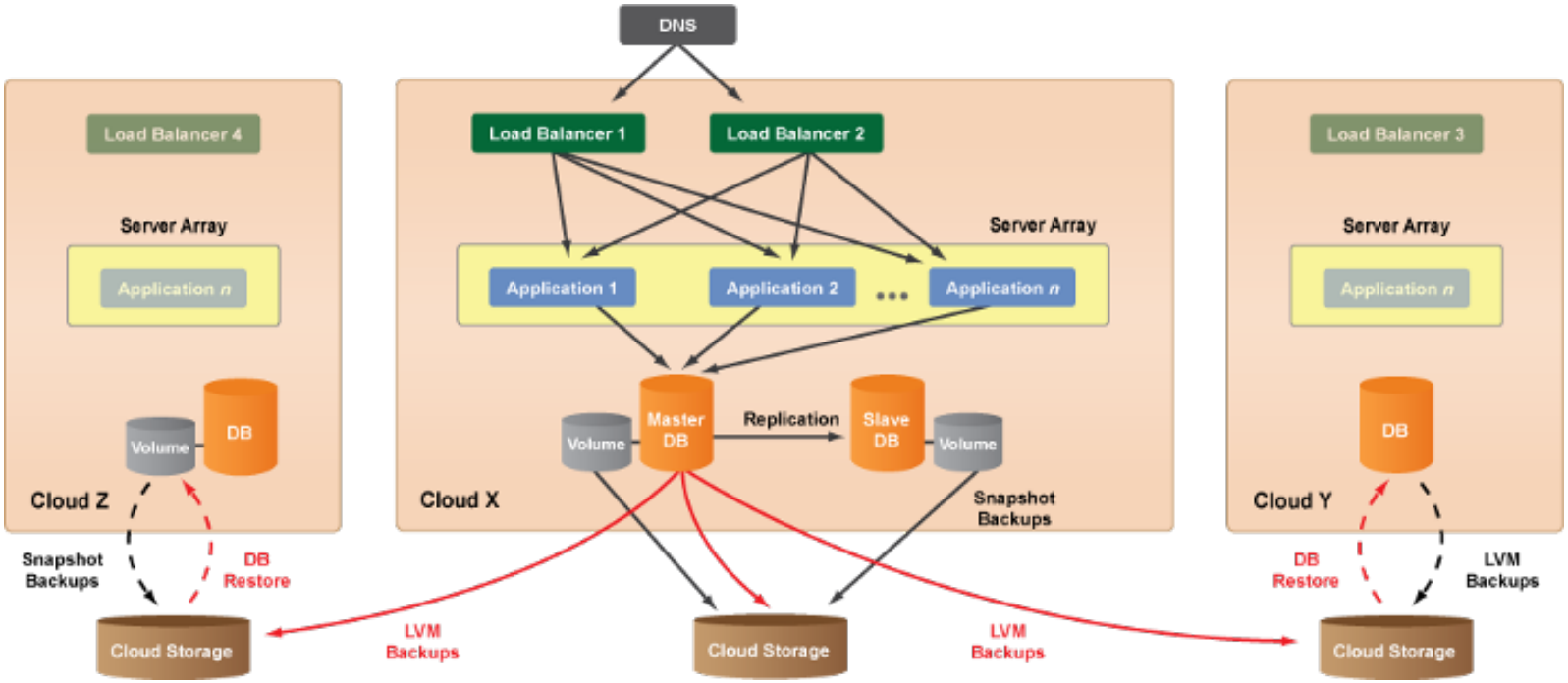
VERIMAG
Univ. Grenoble Alpes, CNRS, Grenoble INP, 38000 Grenoble, France

2019, April 6

MeTRiD 2019: Methods and Tools for Rigorous System Design
Satellite workshop of ETAPS 2019 - 6th of April 2019, Prague, Czech Republic

Dynamic architectures

- *Architectures* are organization principles, recipes for composing components in order to build complex systems meeting given properties



©*RightScale Docs: Hybrid Cloud Site Architecture Diagram*

- *Dynamic architectures* have the ability to adapt so as to meet given properties depending on a changing environment

Dynamic architecture for autonomous systems

Autonomous systems are very important in modern applications

- e.g. transportation, robotics, mobile, resilient, self-* systems
- are *reactive*, that is, they continuously interact with their environment
- consist of *interacting components* that provide basic functionality
- exhibit *dynamicity in different forms* understood as the change of the coordination structure and the number and type of components involved

Is there a possible classification proposing basic types of coordination changes whose combination encompasses any other kind of dynamic coordination ?

1. We study a rigorous modeling methodology and develop a framework for the description of dynamic architectures
2. We show that the framework encompasses various dynamic coordination mechanisms

Dynamic architecture for autonomous systems

Autonomous systems are very important in modern applications

- e.g. transportation, robotics, mobile, resilient, self-* systems
- are *reactive*, that is, they continuously interact with their environment
- consist of *interacting components* that provide basic functionality
- exhibit *dynamicity in different forms* understood as the change of the coordination structure and the number and type of components involved

Is there a possible classification proposing basic types of coordination changes whose combination encompasses any other kind of dynamic coordination ?

1. We study a rigorous modeling methodology and develop a framework for the description of dynamic architectures
2. We show that the framework encompasses various dynamic coordination mechanisms

Dynamic architectures - a classification

Endogenous vs., Exogenous

- *Endogenous*: the coordination mechanisms are *integrated* in the description of components, as in programming languages that use synchronization primitives, function calls etc.
- *Exogenous*: respects a strict *separation* between coordination and internal behavior i.e., components have interface *ports* to which coordination mechanisms are attached

Declarative vs., Imperative

- *Imperative*: the coordination mechanisms are *explicitly* built from elementary mechanisms e.g semaphores, function call, connectors
- *Declarative*: using logics to specify the coordination mechanisms like interaction or configuration logic, etc - Usually coordination mechanisms are constraints that must be met

	endogenous	*exogenous* ✓
imperative	process algebra programming languages	architecture definition languages graph-based formalisms
declarative ✓	[modal logics]	graph logics configuration logics

Dynamic architectures - a classification

Endogenous vs., Exogenous

- *Endogenous*: the coordination mechanisms are *integrated* in the description of components, as in programming languages that use synchronization primitives, function calls etc.
- *Exogenous*: respects a strict *separation* between coordination and internal behavior i.e., components have interface *ports* to which coordination mechanisms are attached

Declarative vs., Imperative

- *Imperative*: the coordination mechanisms are *explicitly* built from elementary mechanisms e.g semaphores, function call, connectors
- *Declarative*: using logics to specify the coordination mechanisms like interaction or configuration logic, etc - Usually coordination mechanisms are constraints that must be met

	endogenous	*exogenous* ✓
imperative	process algebra programming languages	architecture definition languages graph-based formalisms
declarative ✓	[modal logics]	graph logics configuration logics

Dynamic architectures - a classification

Endogenous vs., Exogenous

- *Endogenous*: the coordination mechanisms are *integrated* in the description of components, as in programming languages that use synchronization primitives, function calls etc.
- *Exogenous*: respects a strict *separation* between coordination and internal behavior i.e., components have interface *ports* to which coordination mechanisms are attached

Declarative vs., Imperative

- *Imperative*: the coordination mechanisms are *explicitly* built from elementary mechanisms e.g semaphores, function call, connectors
- *Declarative*: using logics to specify the coordination mechanisms like interaction or configuration logic, etc - Usually coordination mechanisms are constraints that must be met

	endogenous	*exogenous* ✓
imperative	process algebra programming languages	architecture definition languages graph-based formalisms
declarative ✓	[modal logics]	graph logics configuration logics

Dynamic architectures - beyond parametric architectures

Static architectures: static coordination

- a fixed number of component instances and interactions
- e.g., a Master/Slave architecture with 2 masters and 5 slaves

Parametric architectures: parametric coordination rules

- arbitrary number of instances of types of components
- e.g., a parameterized Ring architecture involving N components

Beyond parametric architectures: dynamic coordination

- dynamic, self organizing systems adapting their structure to achieve some goal
- e.g., a pipeline may be configured into a ring and then into a star architecture
- e.g., robots or autonomous agents may adopt various coordination schemes

Our framework: express dynamic architectures as a set of *architectural motifs*, each motif being governed by proper coordination rules

Dynamic architectures - beyond parametric architectures

Static architectures: static coordination

- a fixed number of component instances and interactions
- e.g., a Master/Slave architecture with 2 masters and 5 slaves

Parametric architectures: parametric coordination rules

- arbitrary number of instances of types of components
- e.g., a parameterized Ring architecture involving N components

Beyond parametric architectures: dynamic coordination

- dynamic, self organizing systems adapting their structure to achieve some goal
- e.g., a pipeline may be configured into a ring and then into a star architecture
- e.g., robots or autonomous agents may adopt various coordination schemes

Our framework: express dynamic architectures as a set of *architectural motifs*, each motif being governed by proper coordination rules

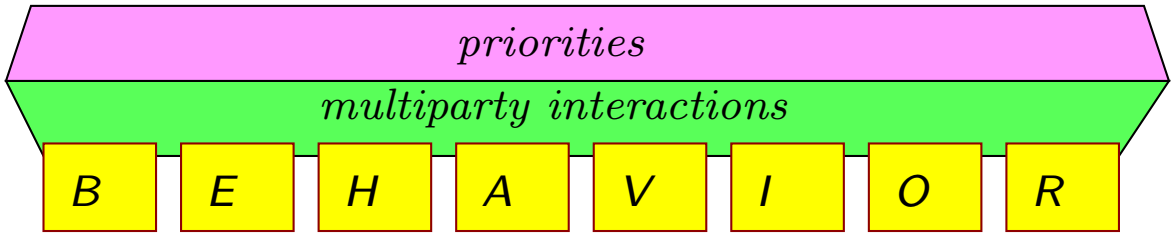
Outline

Outline

The BIP Framework

Component-based systems = layered composition of

- Behavior, atomic functional units (automata, code)
- Interactions, cooperation between actions of behavior
- Priorities, scheduling between interactions



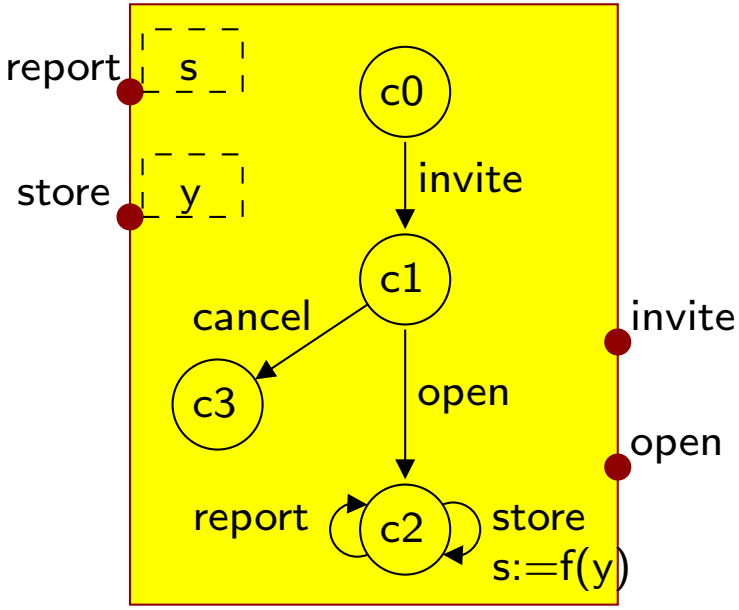
Model-based and component-based design

- providing support for validation and performance analysis
- providing support for implementation on given platforms

BIP: atomic components

extended finite state machines

- a set of *control locations* e.g., c0, c1, ...
- a set of *variables* e.g., s, y
- a set of interface *ports* e.g., report, store
  - attached with variables
- a set of *labelled transitions*
  - port e.g., store
  - *guard* condition e.g., true
  - *assignments* to variables e.g., s:=f(y)
- ...

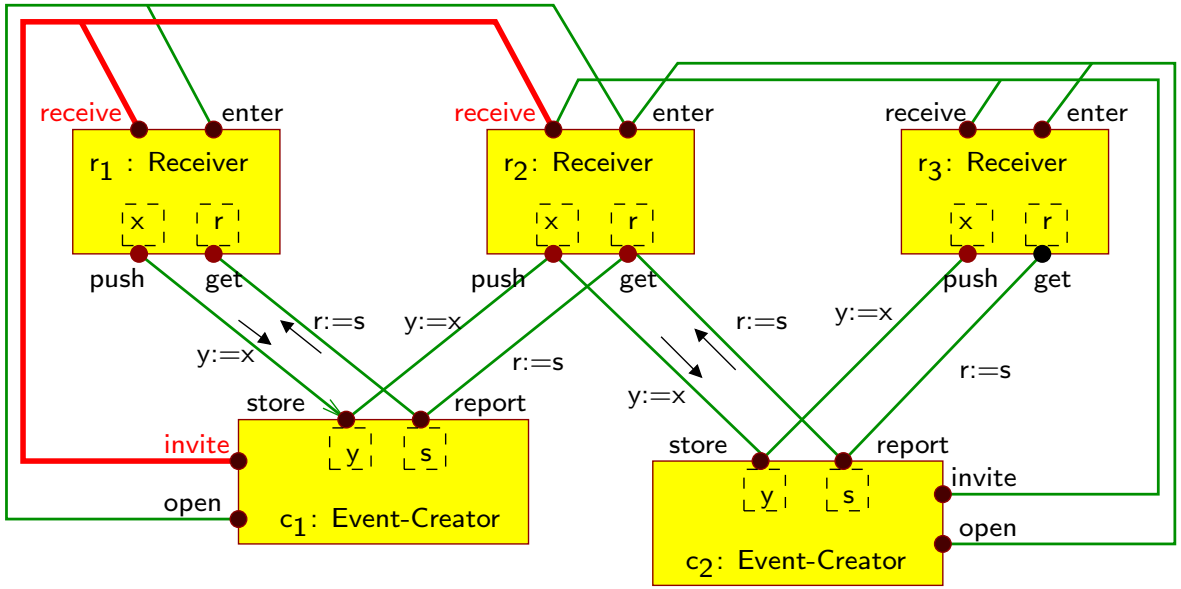


example: an "Event-Creator" component

BIP: multiparty interactions

composition using *n*-ary multiparty interactions

- sets of ports which are *synchronized* at execution
  e.g., (c_1.invite, r_1.receive, r_2.receive), (r_1.push, c_1.store), ...
- guard and assignment of variables from different components

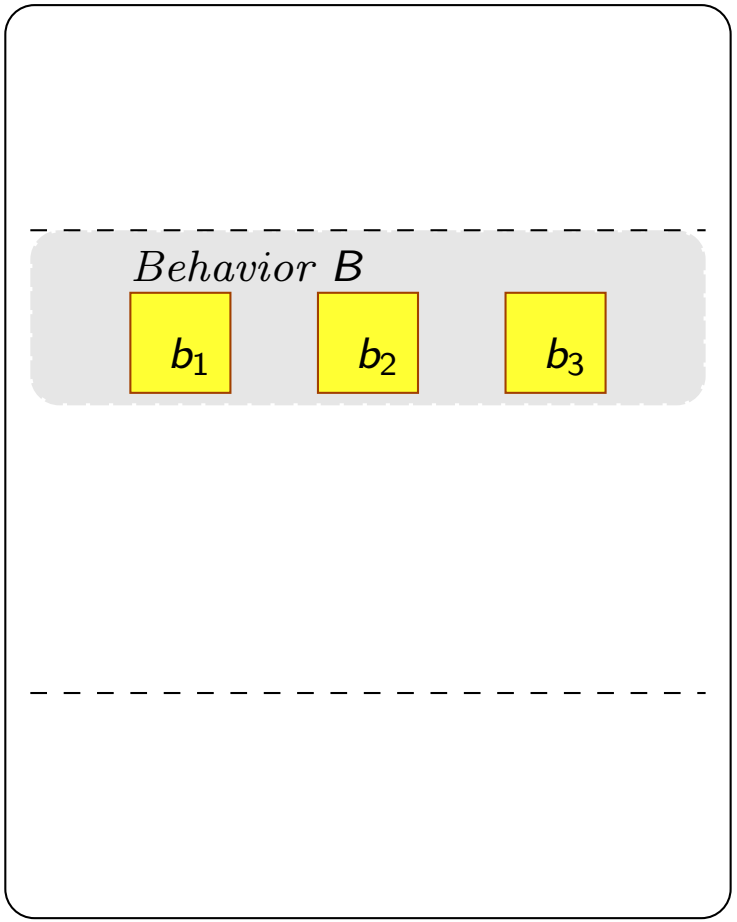


operational semantics of a system

atomic execution of enabled interactions

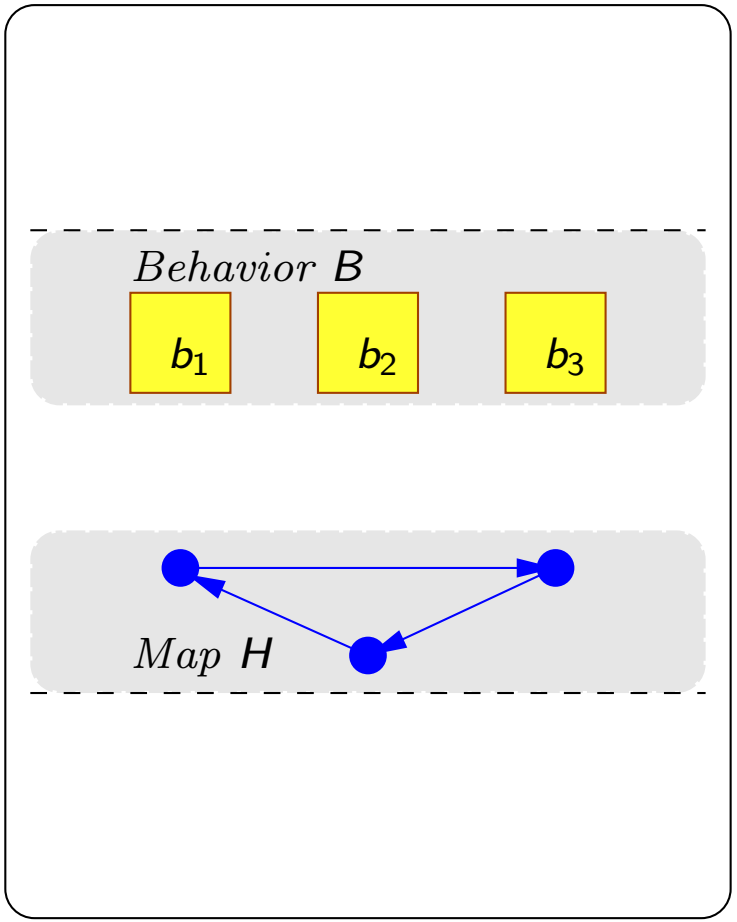
The concept of motif in DR-BIP

- a set of *component instances* B
- parameterized using a logical *map* $H = (N, E)$ and an adressing / *deployment* function $D : B \to N$
- a set of parameterized *interaction rules*
- a set of parameterized *reconfiguration rules*



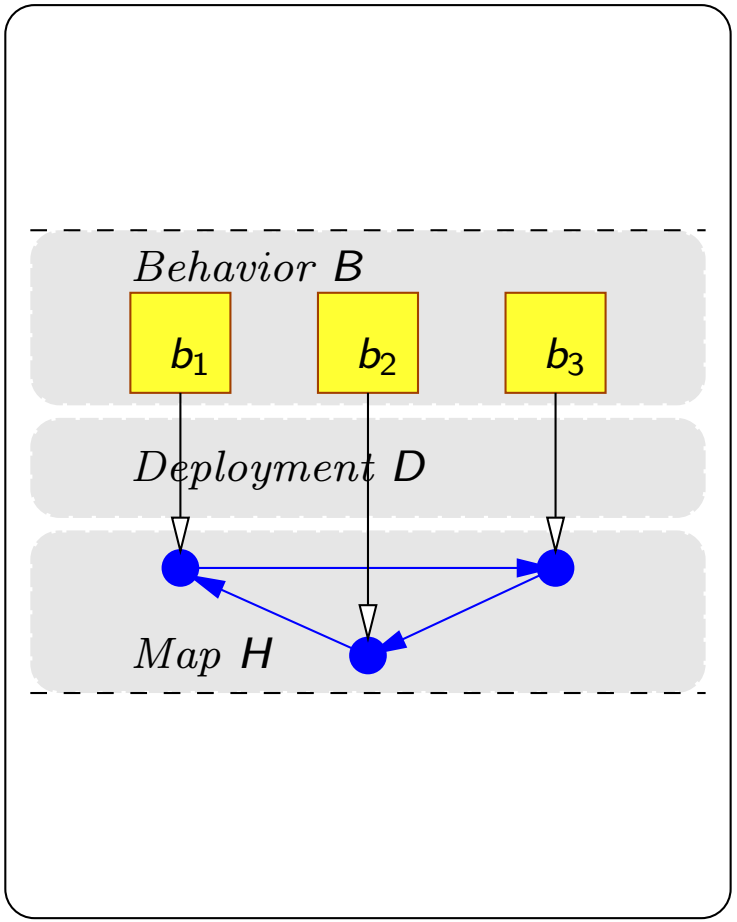
The concept of motif in DR-BIP

- a set of *component instances* B
- parameterized using a logical *map* $H = (N, E)$ and an adressing / *deployment* function $D : B \rightarrow N$
- a set of parameterized *interaction rules*
- a set of parameterized *reconfiguration rules*



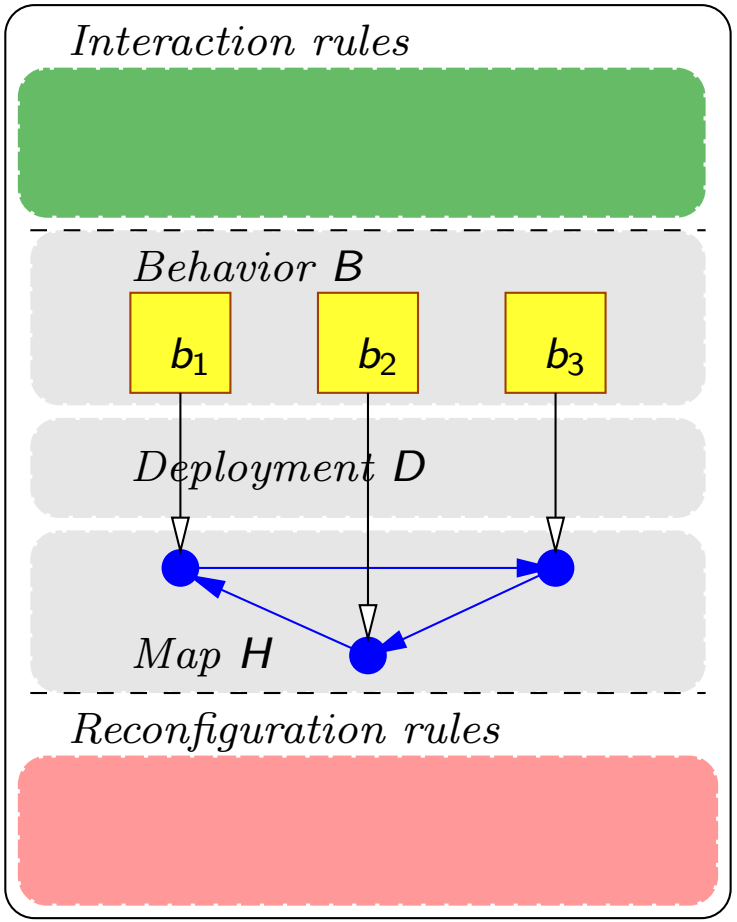
The concept of motif in DR-BIP

- a set of *component instances* B
- parameterized using a logical *map* $H = (N, E)$ and an adressing / *deployment* function $D : B \to N$
- a set of parameterized *interaction rules*
- a set of parameterized *reconfiguration rules*



The concept of motif in DR-BIP

- a set of *component instances* B
- parameterized using a logical *map* $H = (N, E)$ and an adressing / *deployment* function $D : B \to N$
- a set of parameterized *interaction rules*
- a set of parameterized *reconfiguration rules*

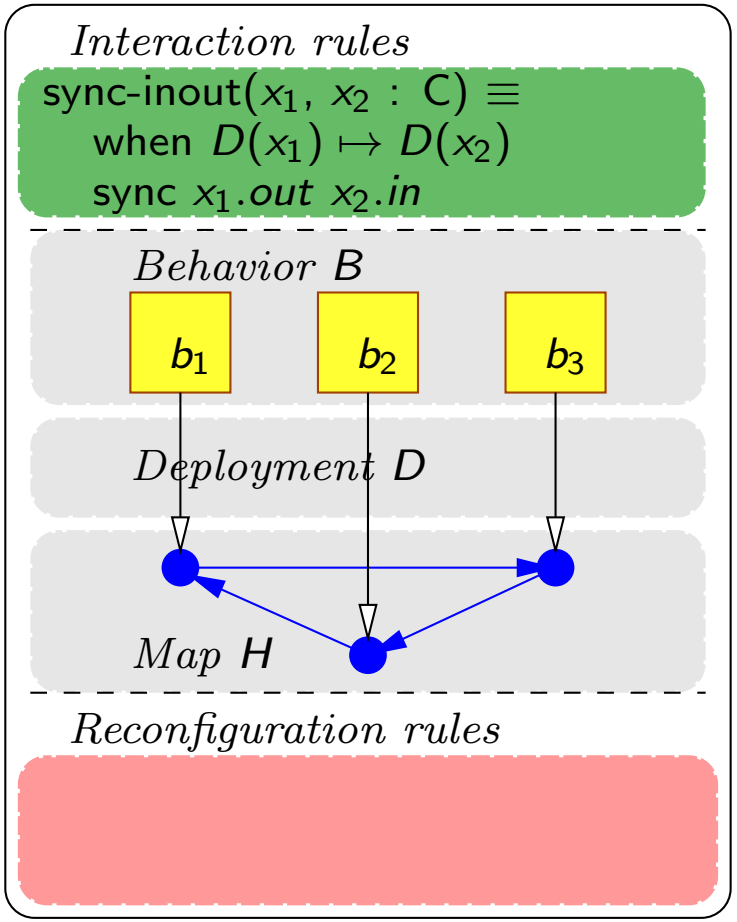


The concept of motif: interaction rules

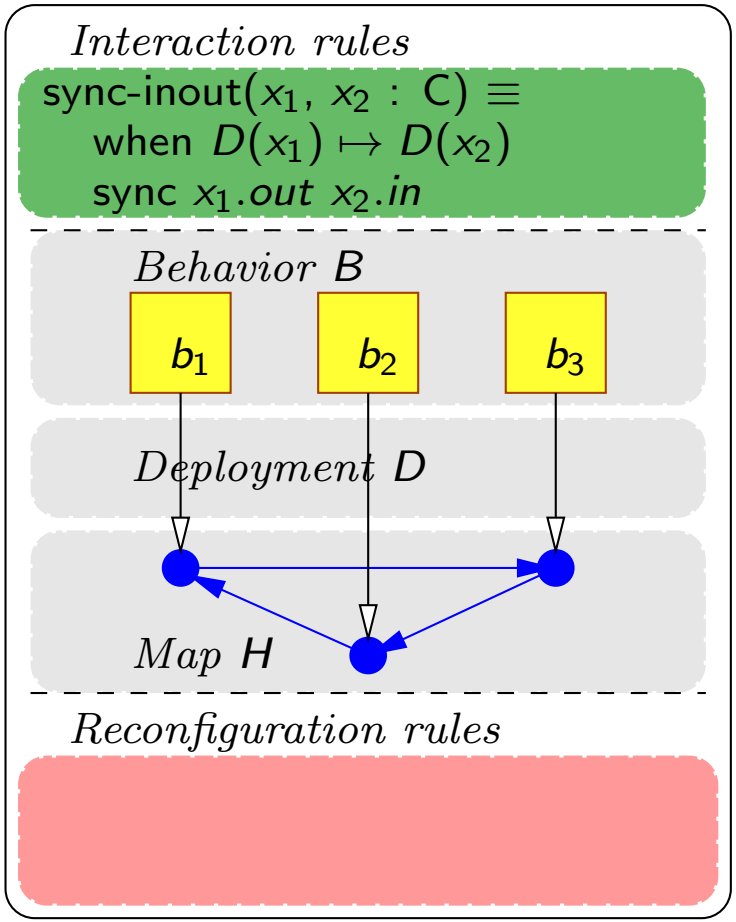
- component's interactions parameterized by the motif content B, H, D

  sync-*rule-name*(*formal-args*) $\equiv$
  when *rule-constraint*
  sync *interaction-ports*
  [*interaction-guard* $\rightarrow$
  interaction-action$^+$]

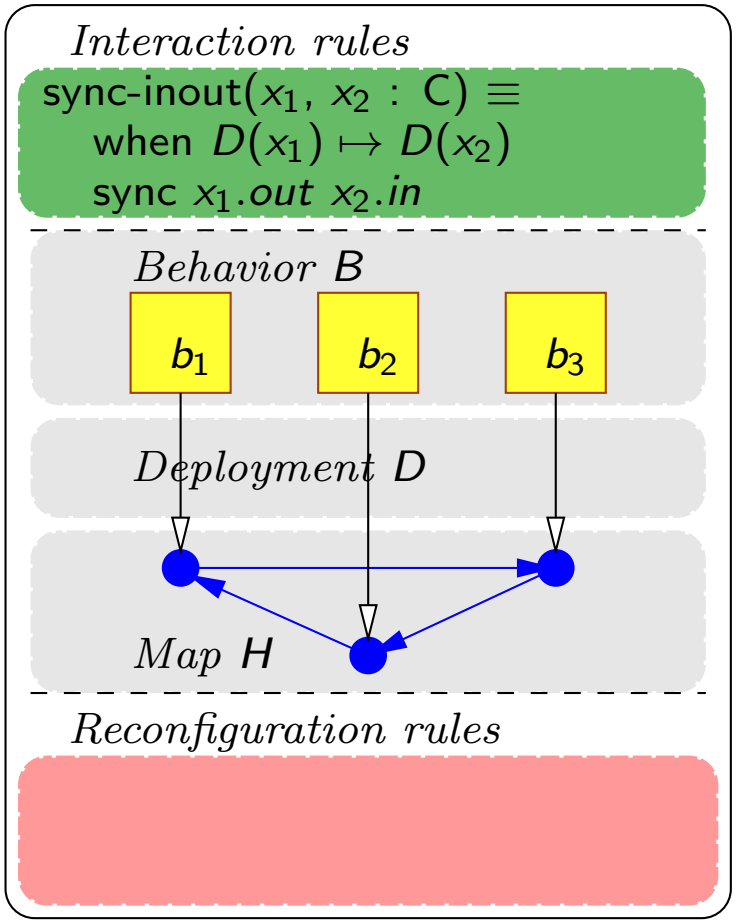
- an interaction defined for every matching of an interaction rule
- the architectural motif defines a regular BIP system

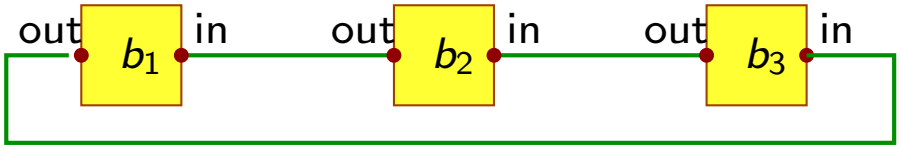


The concept of motif: interaction rules



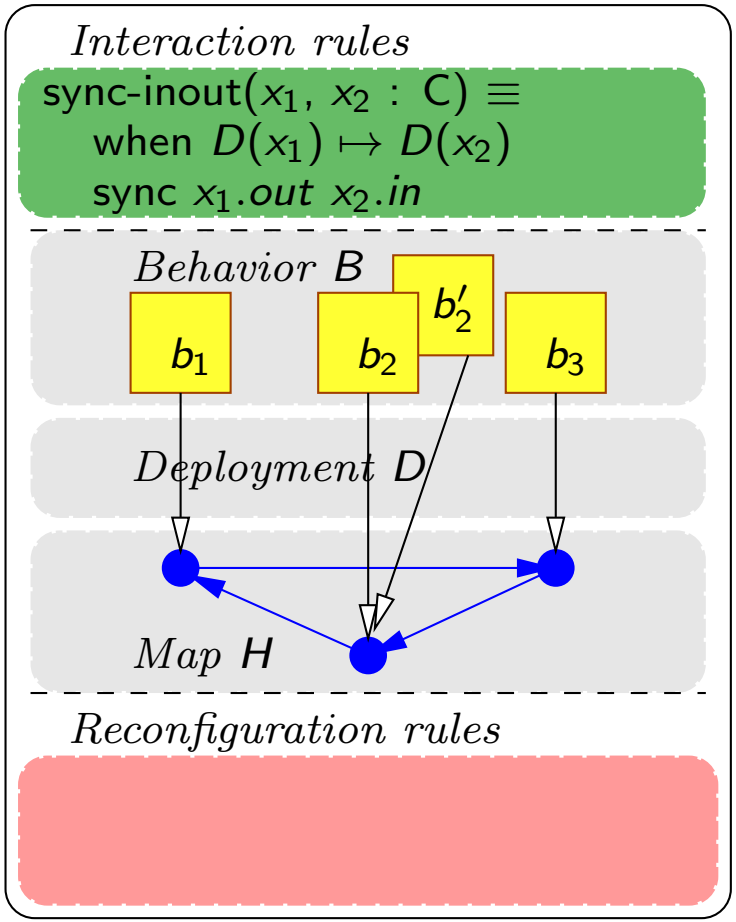
The concept of motif: interaction rules

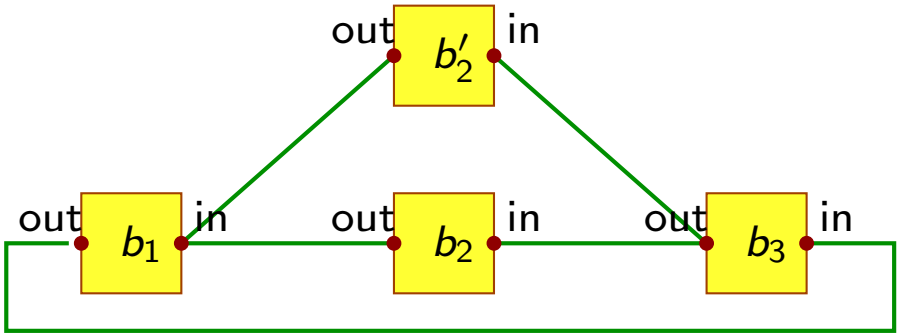




set of interactions between components defined by the interaction rule

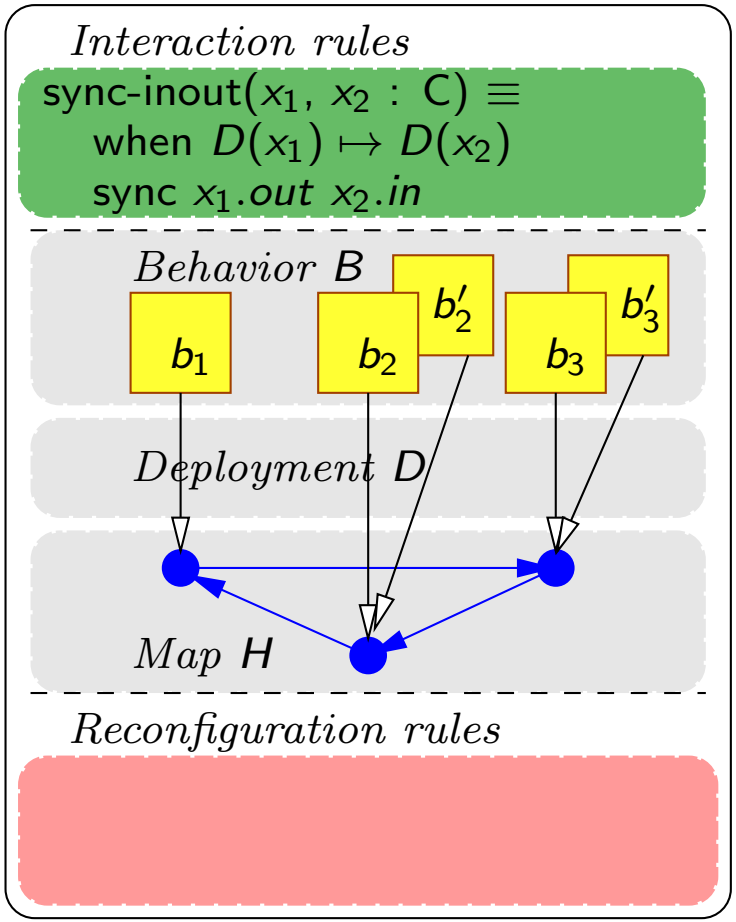
The concept of motif: interaction rules

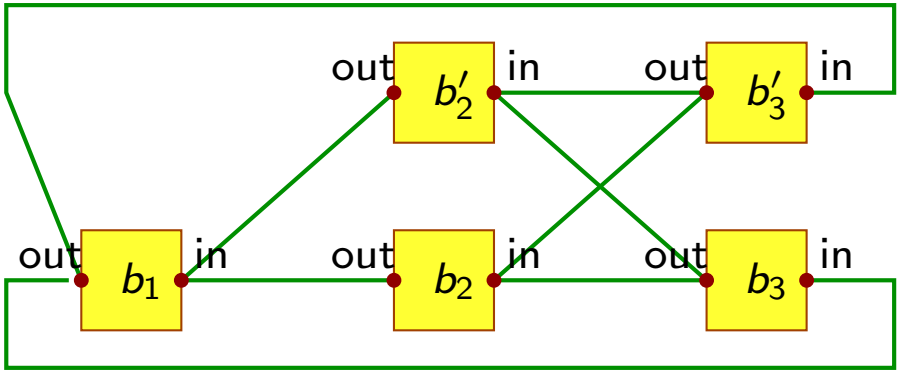




set of interactions between components defined by the interaction rule

The concept of motif: interaction rules





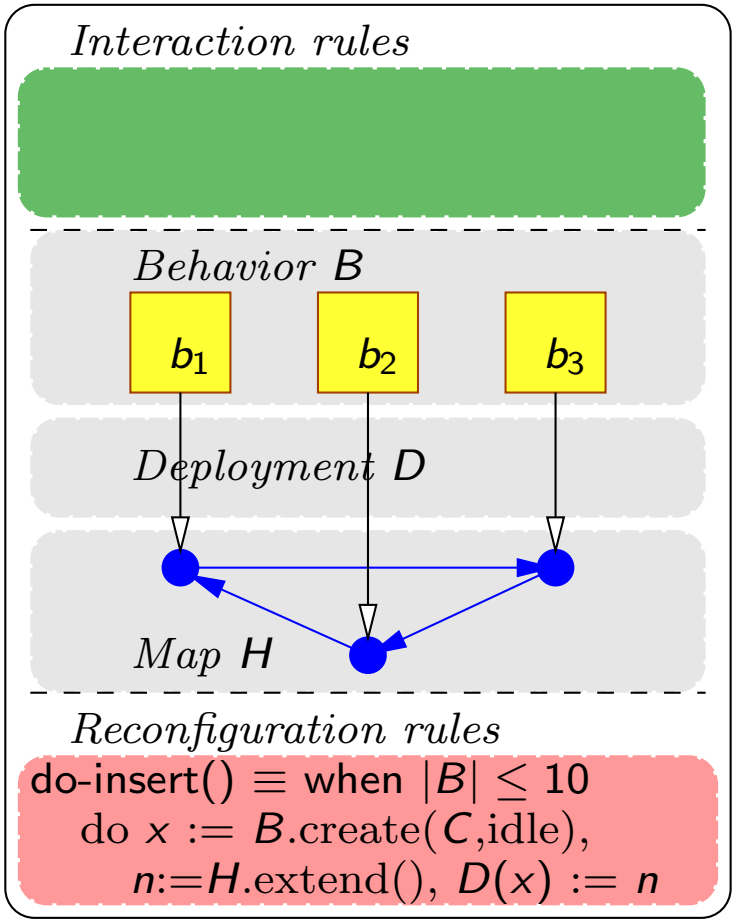
set of interactions between components defined by the interaction rule

The concept of motif: reconfiguration rules

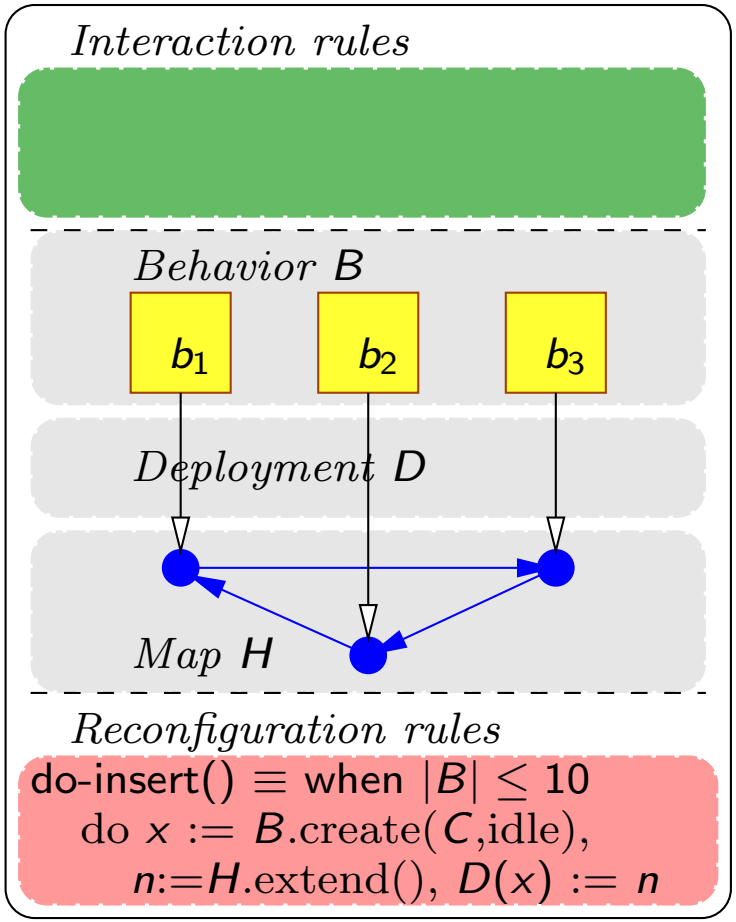
- reconfiguration actions parameterized by the motif content B, H, D

  do-*rule-name*(*formal-args*) $\equiv$
  when *rule-constraint*
  do *reconfiguration-action*$^+$

- a potential reconfiguration action defined for every matching of a reconfiguration rule
- the execution of reconfiguration actions is atomic

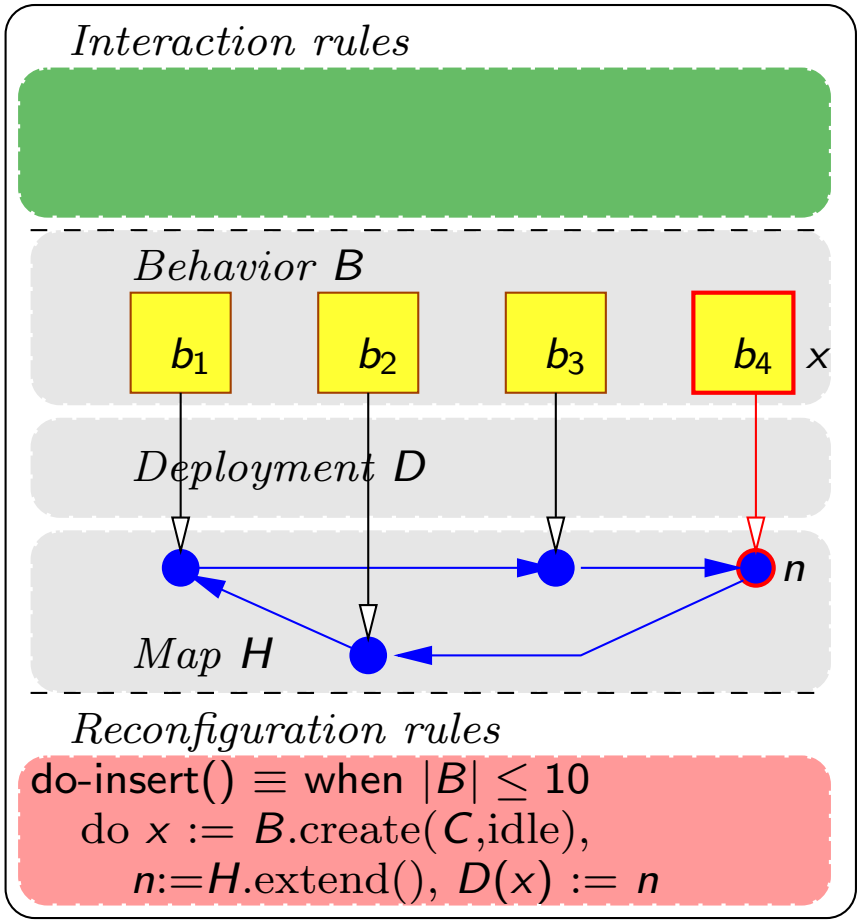


The concept of motif: reconfiguration rules



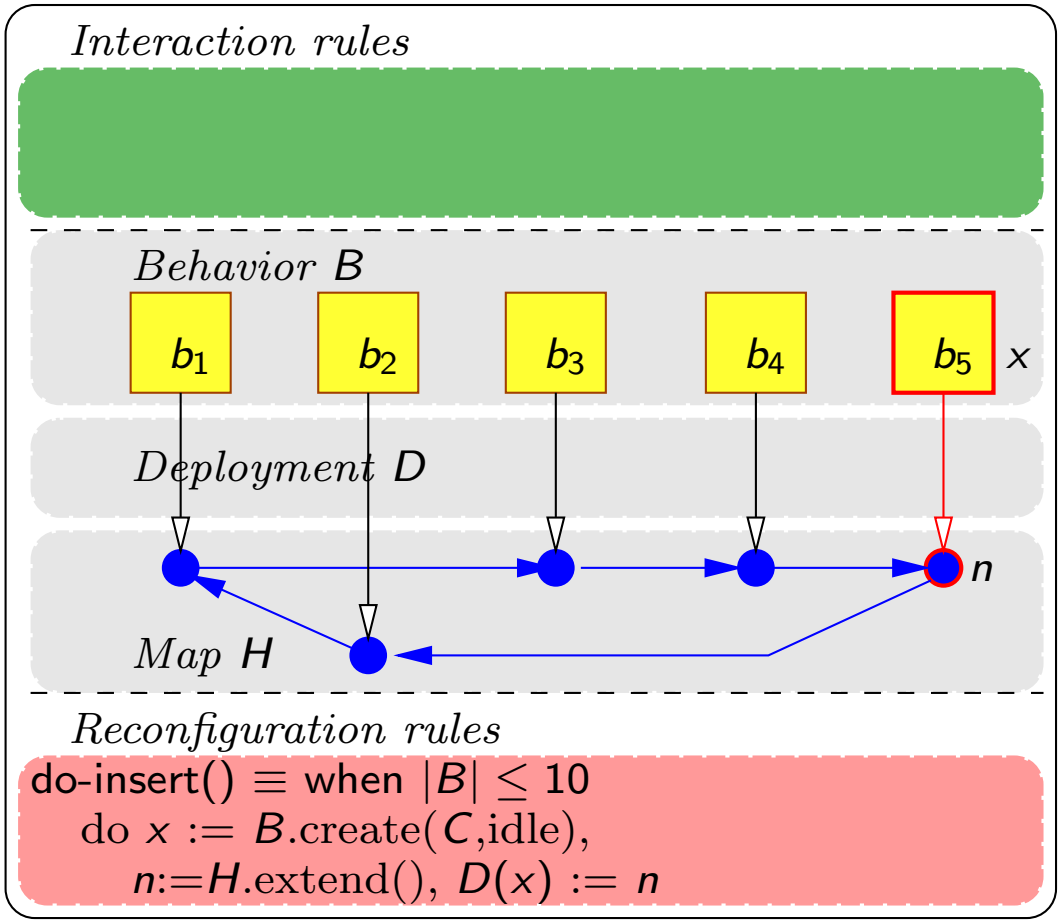
define different forms of architecture dynamism through updates of component instances, map and/or deployment

The concept of motif: reconfiguration rules



define different forms of architecture dynamism through updates of component instances, map and/or deployment

The concept of motif: reconfiguration rules

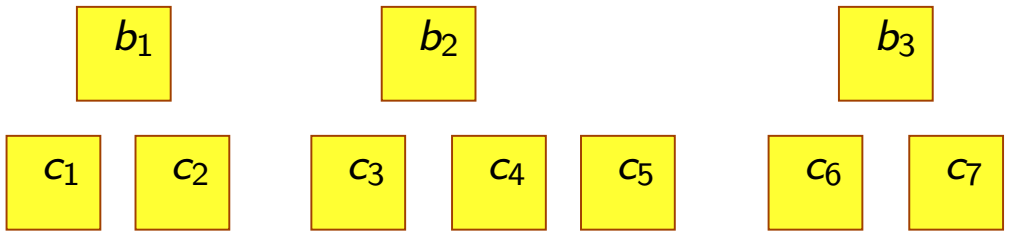


...

define different forms of architecture dynamism through updates of component instances, map and/or deployment

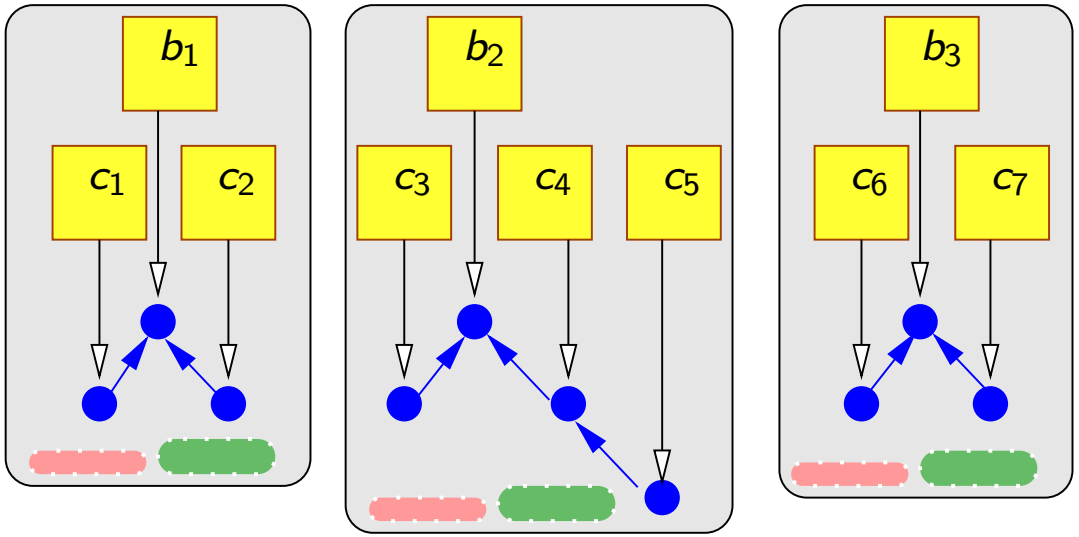
The concept of motif-based system

- a set of *components instances*
- *organized* according to one or more *architectural motifs*
  - an instance may belong to several motifs
- a set of parameterized *inter-motif reconfiguration rules*



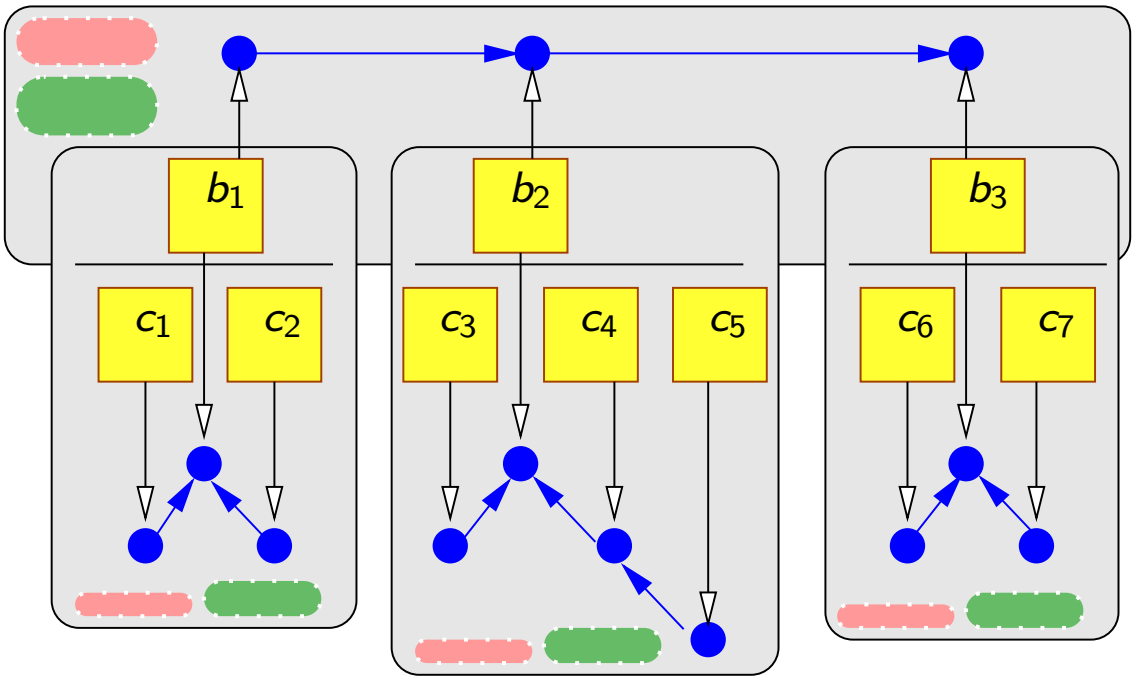
The concept of motif-based system

- a set of *components instances*
- *organized* according to one or more *architectural motifs*
  - an instance may belong to several motifs
- a set of parameterized *inter-motif reconfiguration rules*



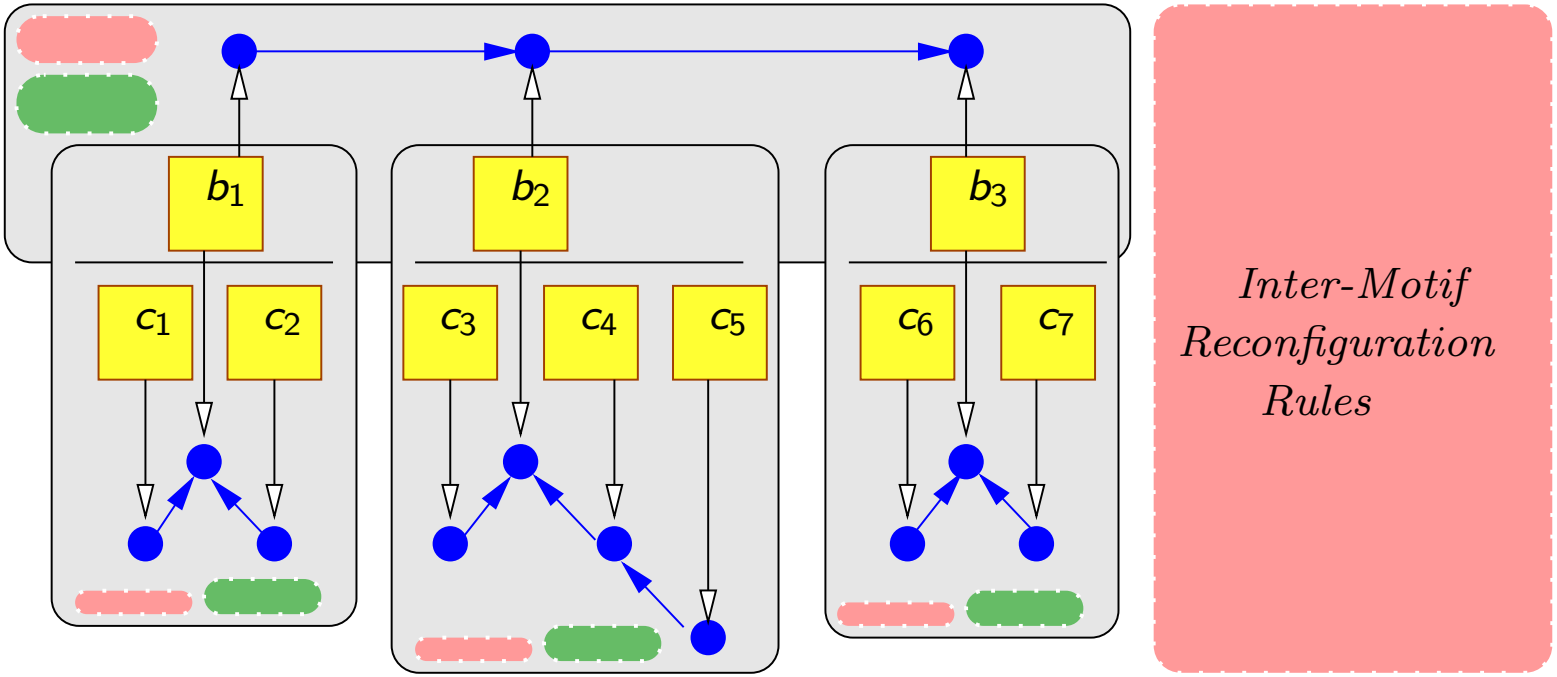
The concept of motif-based system

- a set of *components instances*
- *organized* according to one or more *architectural motifs*
  - an instance may belong to several motifs
- a set of parameterized *inter-motif reconfiguration rules*



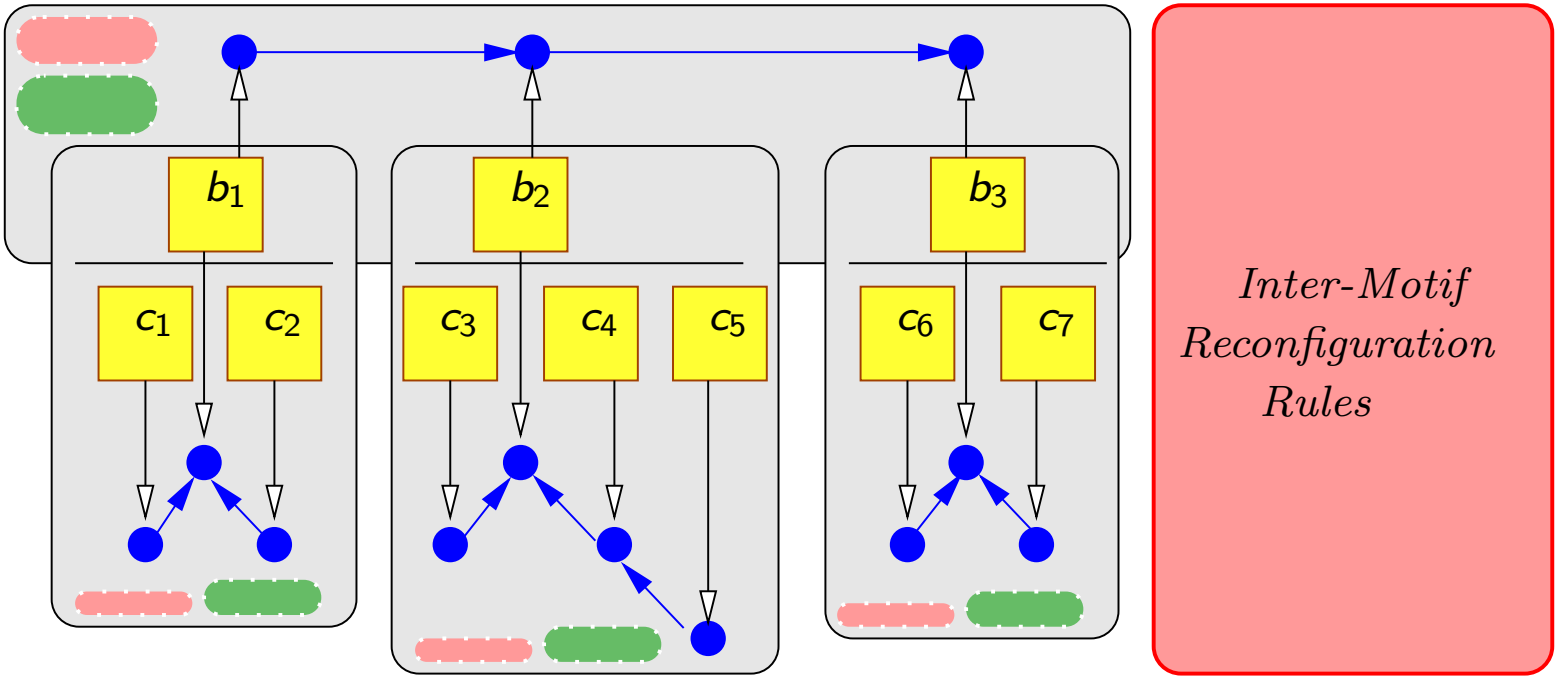
The concept of motif-based system

- a set of *components instances*
- *organized* according to one or more *architectural motifs*
  - an instance may belong to several motifs
- a set of parameterized *inter-motif reconfiguration rules*



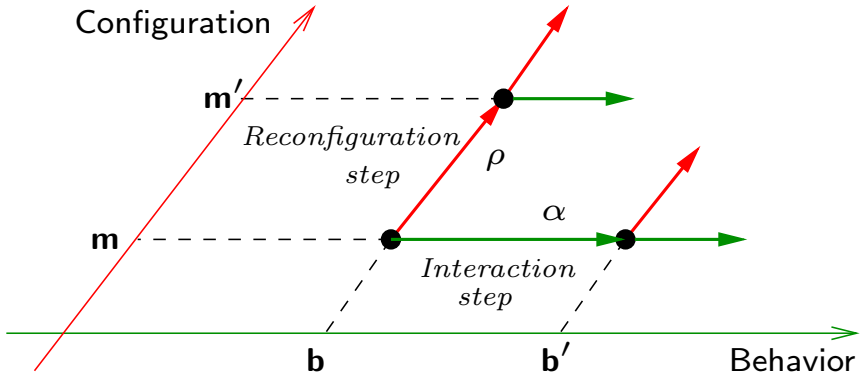
Inter-motif reconfiguration rules

- global reconfiguration actions parameterized by the system content
- define architecture dynamism at system level through joint execution of
  - *intra-motif reconfiguration* actions
  - component instance *migration* from a motif to another
  - architectural motif *creation* and *deletion*



Motif-based systems: operational semantics

- operational semantics is defined incrementally for components, motifs and systems
- a motif-based system *state* is a couple $(\mathbf{m}, \mathbf{b})$ including motifs configuration $\mathbf{m}$ and behaviour / component states $\mathbf{b}$



- separation of concerns between interaction and configuration steps:
  - interactions α preserve motifs configuration i.e., architecture
  - reconfiguration ρ preserve component states

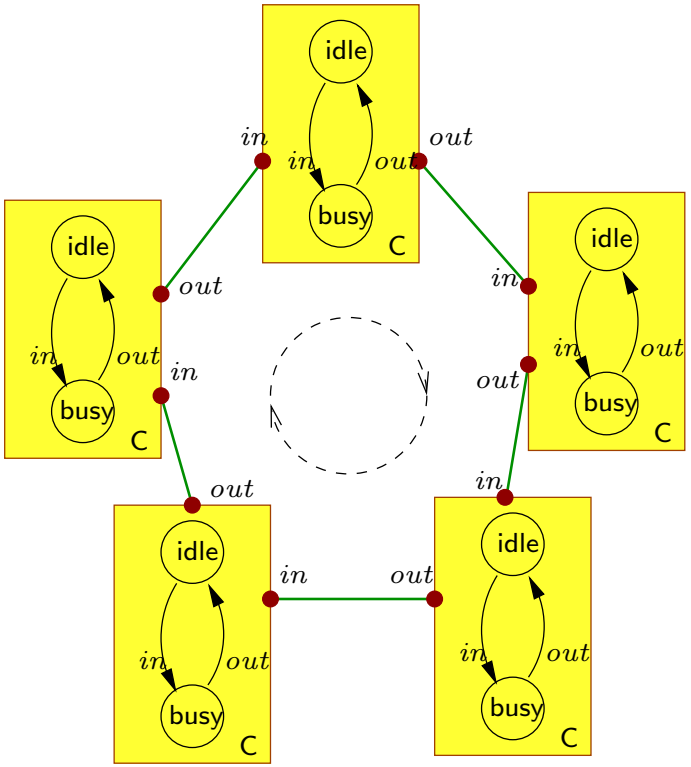
Outline

Exercises

1. *Dynamic Token Ring System*
   - component creation/deletion, motif reconfiguration
2. *Dynamic Multicore Task System*
   - component creation/deletion, component migration
3. *Automated Highway Traffic System*
   - Bergenhem, *Approaches for facilities layer protocols for platooning*, 2015
   - component migration, motif creation/deletion, inter-motif reconfiguration
4. *Self-organized Robot Colonies*
   - Dorigo et al, *Teamwork in self-organized robot colonies*, 2008
   - component migration, inter-motif reconfiguration

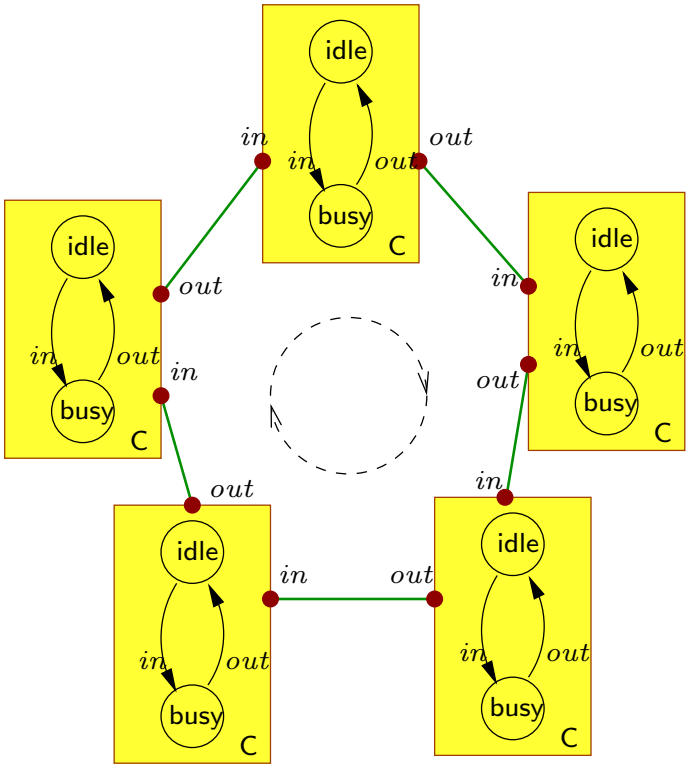
Dynamic token ring system

- a token-passing ring system consisting of identical component instances
- the token is send / received through synchronization with the corresponding neighbor
- *new component instances are allowed to enter and existing instances to leave the ring at any time*



Dynamic token ring system

- a token-passing ring system consisting of identical component instances
- the token is send / received through synchronization with the corresponding neighbor
- *new component instances are allowed to enter and existing instances to leave the ring at any time*



Dynamic token ring system

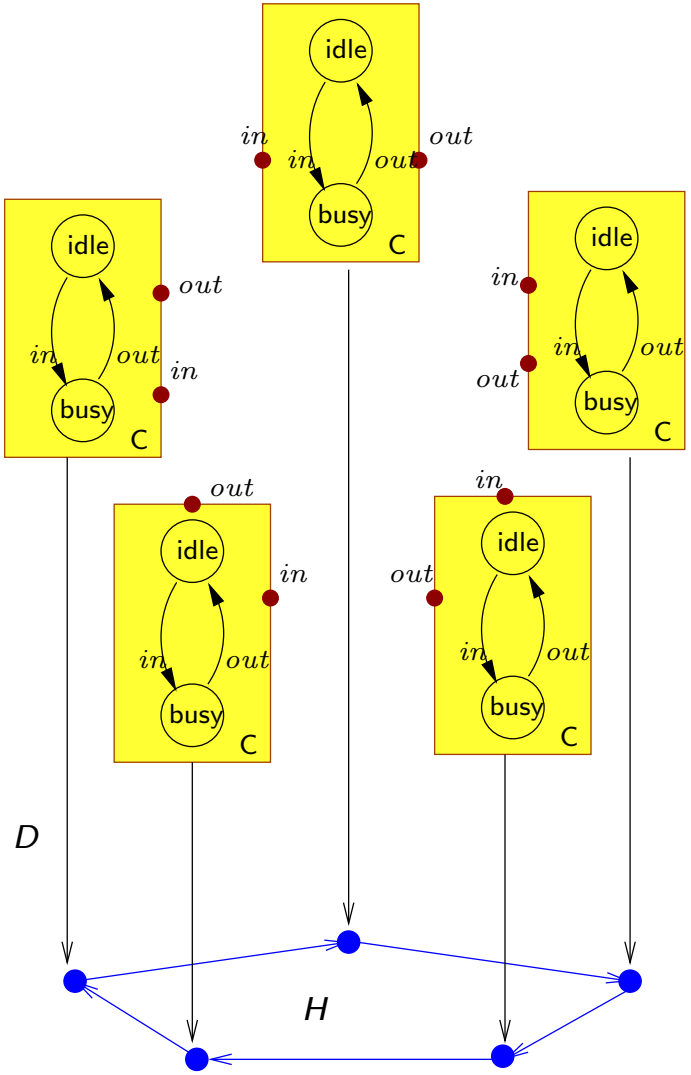
The **Ring** motif

- the map is a cyclic graph with additional update primitives
- components are deployed 1-to-1 into map nodes
- interaction rules

sync-ring-inout(x_1, x_2 : C) $\equiv$
  when D(x_1) $\mapsto$ D(x_2) sync x_1.out x_2.in

- reconfiguration rules

do-ring-insert() $\equiv$
  do x := B.create(C, idle),
    n := H.extend(), D(x) := n
do-ring-remove(x : C) $\equiv$
  when $|$B$| \geq 3 \wedge x$.idle
  do n := D(x), B.delete(x), H.remove(n)



Dynamic token ring system

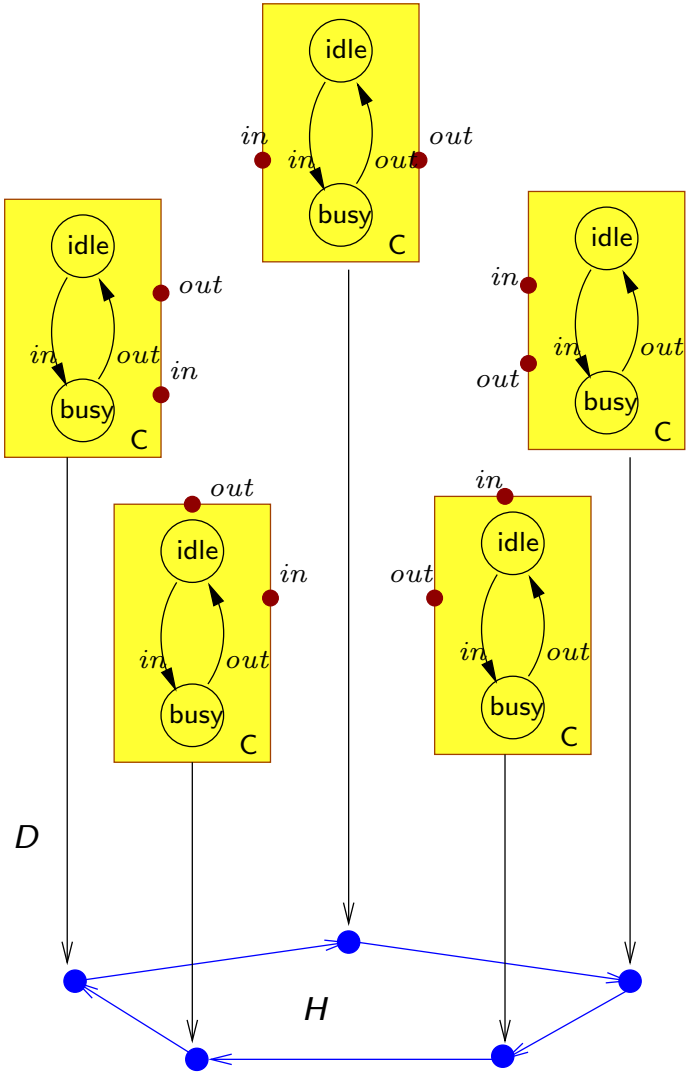
The **Ring** motif

- the map is a cyclic graph with additional update primitives
- components are deployed 1-to-1 into map nodes
- interaction rules

sync-ring-inout(x_1, x_2 : C) $\equiv$
  <u>when</u> D(x_1) $\mapsto$ D(x_2) <u>sync</u> x_1.out x_2.in

- reconfiguration rules

do-ring-insert() $\equiv$
  <u>do</u> x := B.create(C, idle),
    n := H.extend(), D(x) := n
do-ring-remove(x : C) $\equiv$
  <u>when</u> $|$B$| \geq 3 \wedge x$.idle
  <u>do</u> n := D(x), B.delete(x), H.remove(n)



Dynamic token ring system

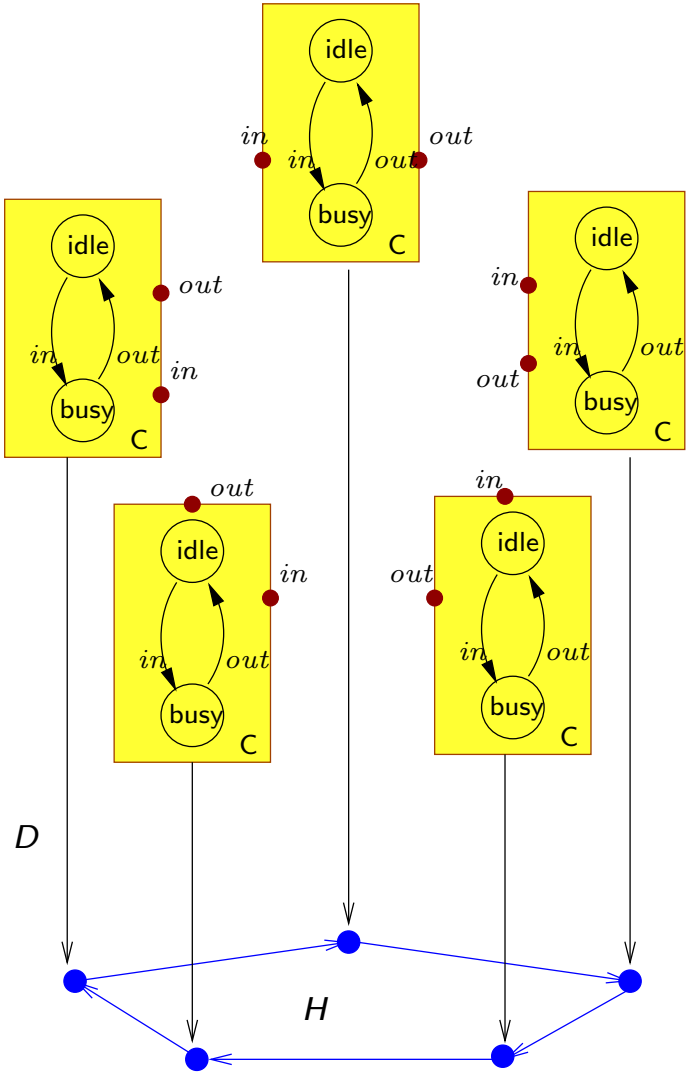
The **Ring** motif

- the map is a cyclic graph with additional update primitives
- components are deployed 1-to-1 into map nodes
- interaction rules

sync-ring-inout(x_1, x_2 : C) $\equiv$
  <u>when</u> D(x_1) $\mapsto$ D(x_2) <u>sync</u> x_1.out x_2.in

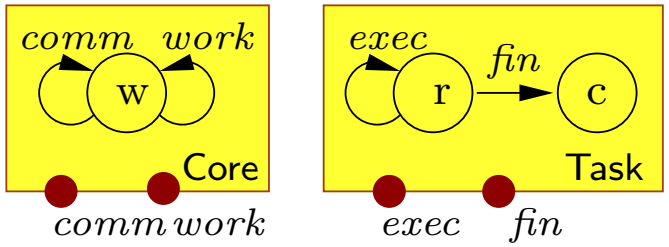
- reconfiguration rules

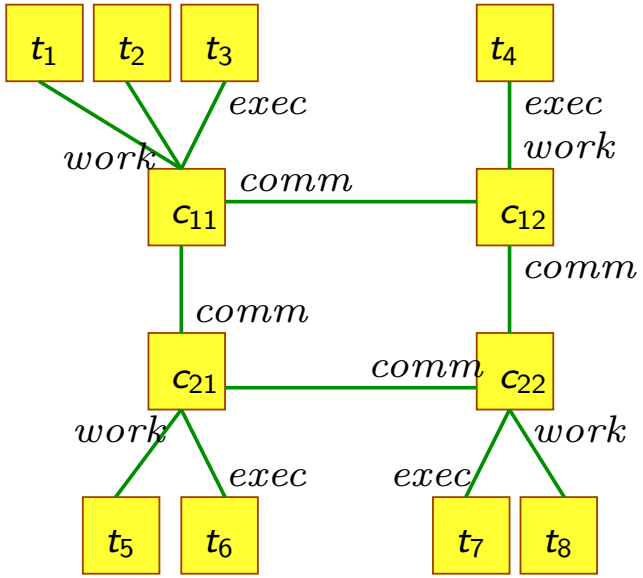
do-ring-insert() $\equiv$
  <u>do</u> x := B.create(C, idle),
    n := H.extend(), D(x) := n
do-ring-remove(x : C) $\equiv$
  <u>when</u> |B| $\geq$ 3 $\wedge$ x.idle
  <u>do</u> n := D(x), B.delete(x), H.remove(n)



Dynamic multicore task system

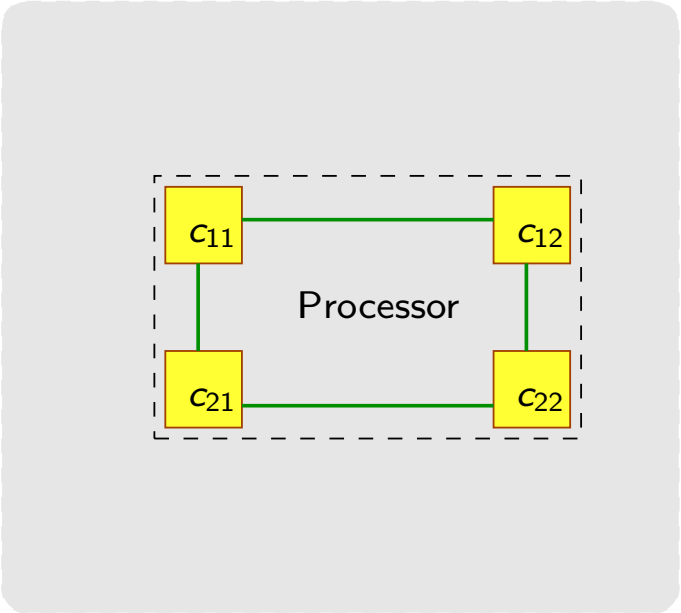
- $n \times n$ multicore grid, n fixed
- every core runs several tasks
- *tasks can migrate such that to balance the load amongst cores*
- *new tasks created, completed tasks removed*





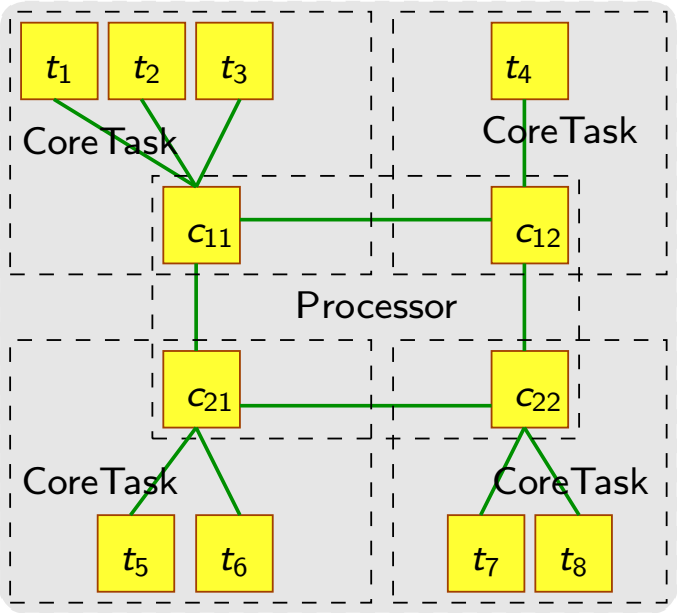
Dynamic multicore task system

- *$n \times n$ multicore grid*
  - the **Processor** motif captures the static architecture
  - grid-like map and 1-to-1 deployment of core components onto map locations
- *every core runs several tasks*
  - the **CoreTask** motif captures the current task allocation
  - map and deployment are not relevant



Dynamic multicore task system

- *$n \times n$ multicore grid*
  - the **Processor** motif captures the static architecture
  - grid-like map and 1-to-1 deployment of core components onto map locations
- *every core runs several tasks*
  - the **CoreTask** motif captures the current task allocation
  - map and deployment are not relevant

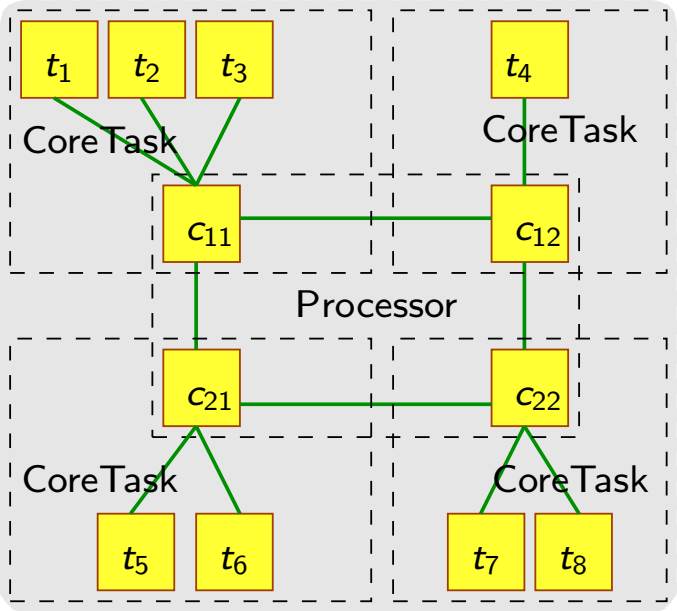


Dynamic multicore task system

- *tasks can migrate such that to balance the load amongst cores*
- inter-motif reconfiguration rule

$$
\begin{array}{l}
\text{do-migrate}(y_1, y_2 : \text{CoreTask},\ y_3 : \text{Processor}, \\
\qquad x_1, x_2 : \text{Core},\ x_3 : \text{Task}) \equiv \\
\quad \underline{\text{when}}\ \langle\ y_1 : x_1 \in \text{B}\ \rangle \wedge \langle\ y_2 : x_2 \in \text{B}\ \rangle \wedge \\
\qquad \langle\ y_3 : \text{D}(x_1) \mapsto \text{D}(x_2)\ \rangle \wedge \\
\qquad |y_1.\text{B}| > |y_2.\text{B}| + \text{K} \wedge x_3 \in y_1.\text{B} \\
\quad \underline{\text{do}}\ y_2.\text{migrate}(x_3),\ y_1.\text{delete}(x_3)
\end{array}
$$

- *new tasks created, completed tasks removed, etc*
  - additional reconfiguration rules within the **CoreTask** motifs

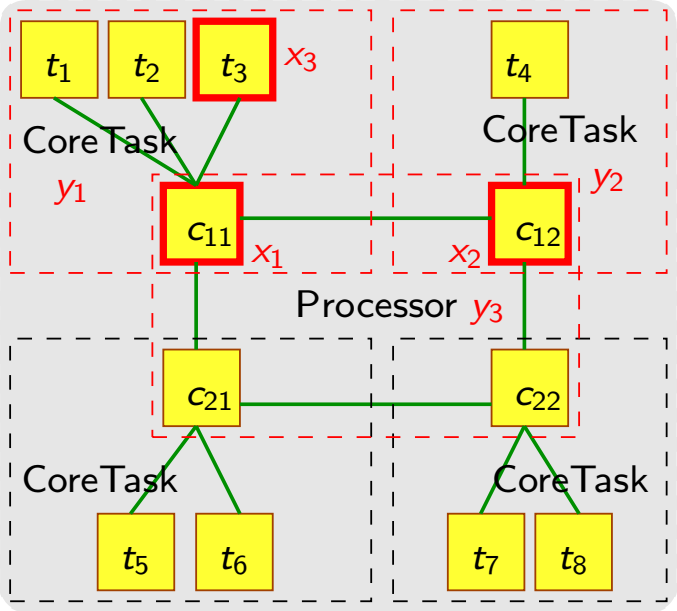


Dynamic multicore task system

- *tasks can migrate such that to balance the load amongst cores*
- inter-motif reconfiguration rule

$$
\begin{array}{l}
\text{do-migrate}(y_1, y_2 : \text{CoreTask},\ y_3 : \text{Processor}, \\
\qquad x_1, x_2 : \text{Core},\ x_3 : \text{Task}) \equiv \\
\quad \underline{\text{when}}\ \langle\ y_1 : x_1 \in \text{B}\ \rangle \wedge \langle\ y_2 : x_2 \in \text{B}\ \rangle \wedge \\
\qquad \langle\ y_3 : \text{D}(x_1) \mapsto \text{D}(x_2)\ \rangle \wedge \\
\qquad |y_1.\text{B}| > |y_2.\text{B}| + \text{K} \wedge x_3 \in y_1.\text{B} \\
\quad \underline{\text{do}}\ y_2.\text{migrate}(x_3),\ y_1.\text{delete}(x_3)
\end{array}
$$

- *new tasks created, completed tasks removed, etc*
  - additional reconfiguration rules within the **CoreTask** motifs

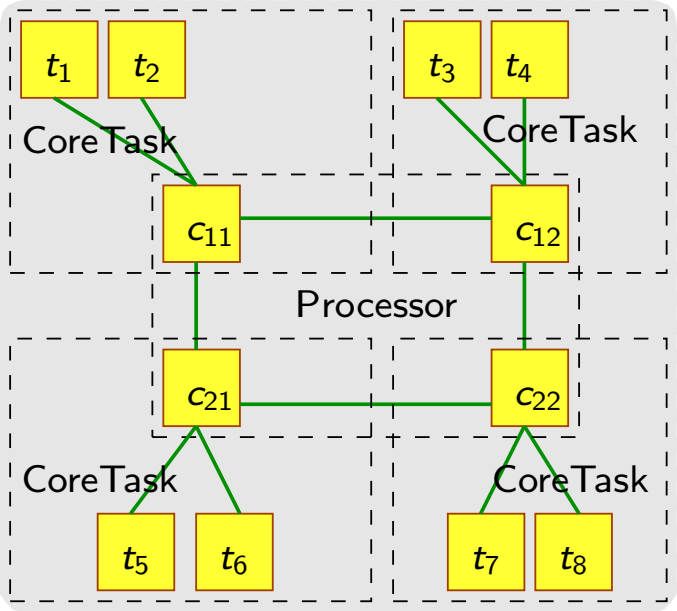


Dynamic multicore task system

- *tasks can migrate such that to balance the load amongst cores*
- inter-motif reconfiguration rule

$$
\begin{array}{l}
\text{do-migrate}(y_1, y_2 : \text{CoreTask},\ y_3 : \text{Processor}, \\
\qquad x_1, x_2 : \text{Core},\ x_3 : \text{Task}) \equiv \\
\quad \underline{\text{when}}\ \langle\ y_1 : x_1 \in \text{B}\ \rangle \wedge \langle\ y_2 : x_2 \in \text{B}\ \rangle \wedge \\
\qquad \langle\ y_3 : \text{D}(x_1) \mapsto \text{D}(x_2)\ \rangle \wedge \\
\qquad |y_1.\text{B}| > |y_2.\text{B}| + \text{K} \wedge x_3 \in y_1.\text{B} \\
\quad \underline{\text{do}}\ y_2.\text{migrate}(x_3),\ y_1.\text{delete}(x_3)
\end{array}
$$

- *new tasks created, completed tasks removed, etc*
  - additional reconfiguration rules within the **CoreTask** motifs

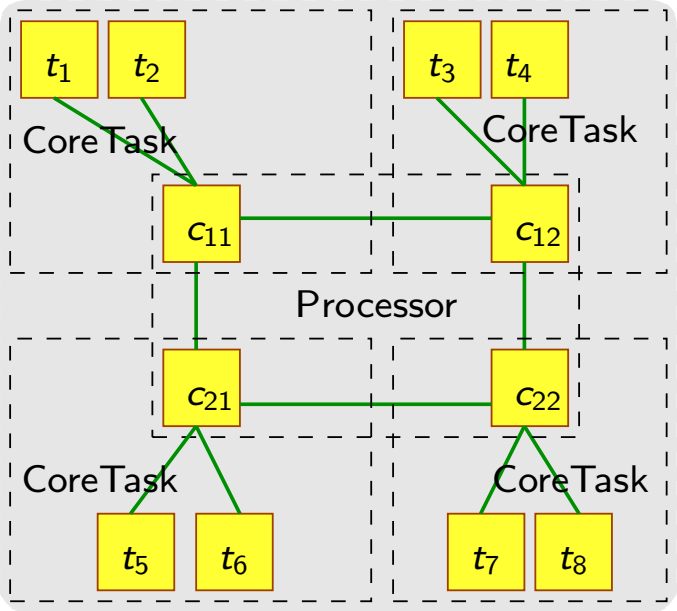


Dynamic multicore task system

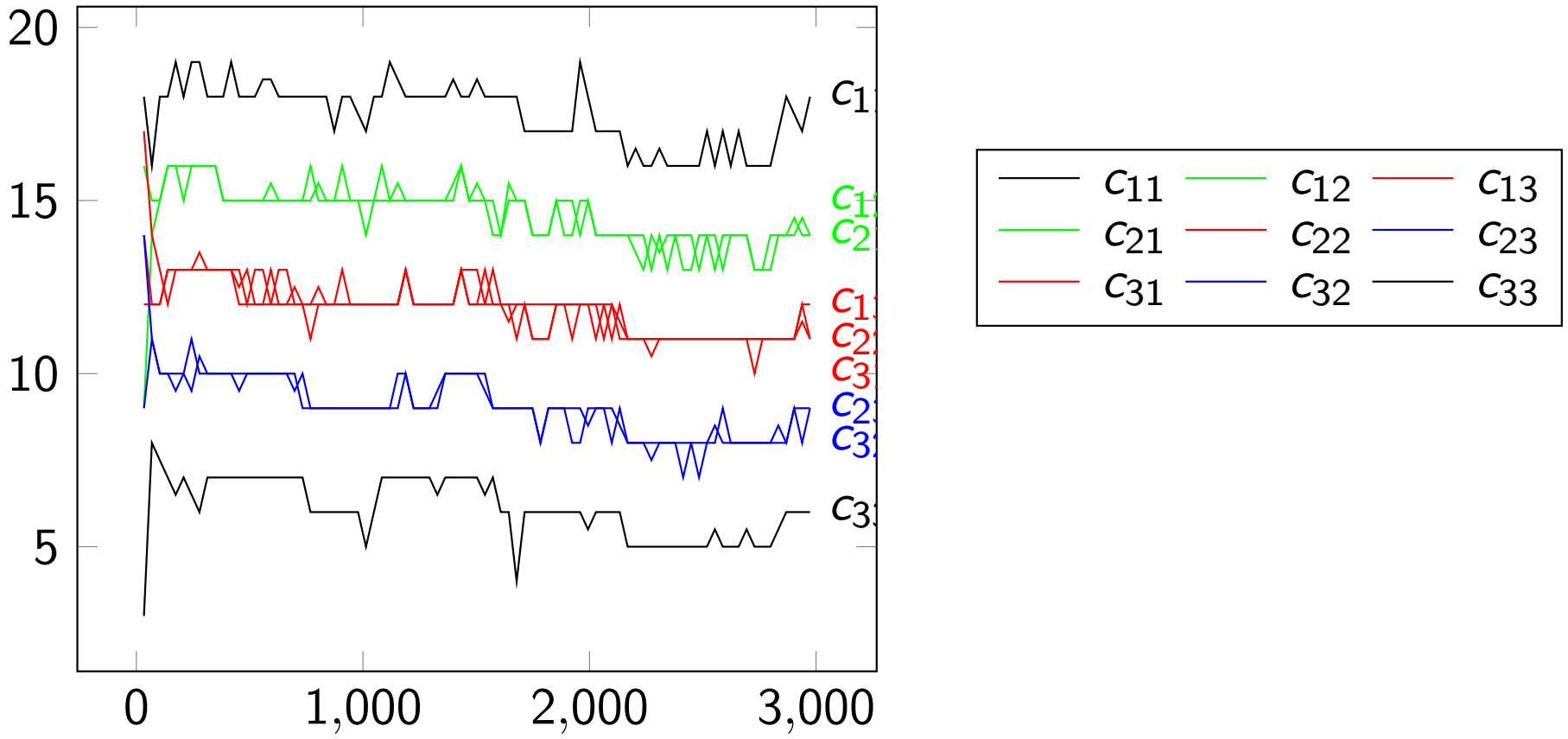
- *tasks can migrate such that to balance the load amongst cores*
- inter-motif reconfiguration rule

$$
\begin{array}{l}
\text{do-migrate}(y_1, y_2 : \text{CoreTask},\ y_3 : \text{Processor}, \\
\qquad x_1, x_2 : \text{Core},\ x_3 : \text{Task}) \equiv \\
\quad \underline{\text{when}}\ \langle\ y_1 : x_1 \in \text{B}\ \rangle \wedge \langle\ y_2 : x_2 \in \text{B}\ \rangle \wedge \\
\qquad \langle\ y_3 : \text{D}(x_1) \mapsto \text{D}(x_2)\ \rangle \wedge \\
\qquad |y_1.\text{B}| > |y_2.\text{B}| + \text{K} \wedge x_3 \in y_1.\text{B} \\
\quad \underline{\text{do}}\ y_2.\text{migrate}(x_3),\ y_1.\text{delete}(x_3)
\end{array}
$$

- *new tasks created, completed tasks removed, etc*
  - additional reconfiguration rules within the **CoreTask** motifs



Dynamic multicore task system



Simulation results of a 3×3 dynamic multicore task system for 3000 steps (new tasks are created at c_{11}, completed tasks are removed at c_{33})

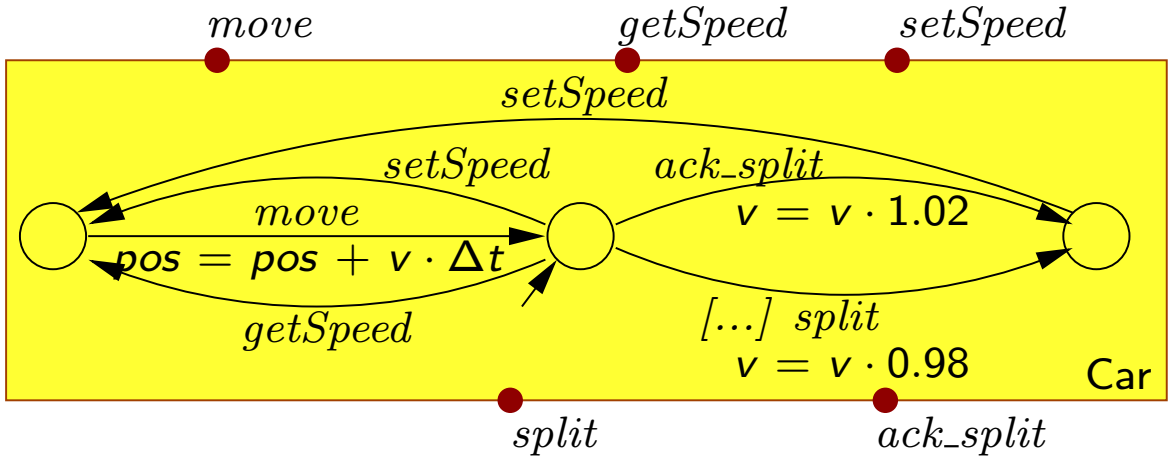
Automated highway traffic system

- a single-lane one-way road
- an arbitrary number of autonomous homogeneous self-driving cars are moving in the same direction, at different cruising speeds
- cars are organized into *platoons*
  i.e. groups of cars cruising at the same speed and closely following a leader car
- platoons may dynamically merge or split
  - a merge takes place if two platoons are close enough
  - a split takes place when an arbitrary car requests to leave the platoon

[C. Bergenhem, *Approaches for facilities layer protocols for platooning*, 18th IEEE International Conference on Intelligent Transportation Systems, ITSC 2015]

Automated highway traffic system

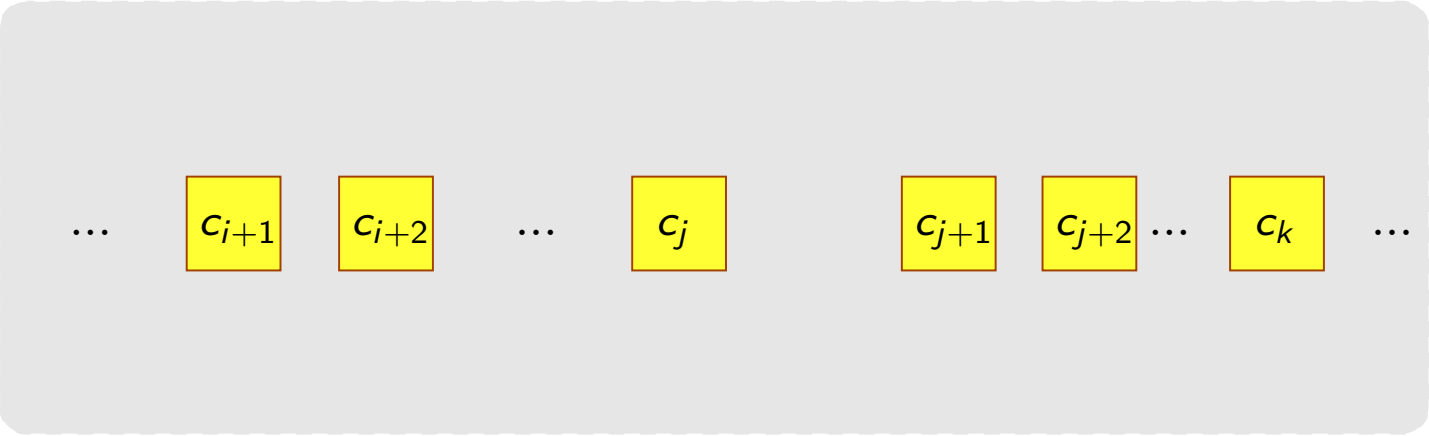
- the behavior of **Car** components



- includes both platoon leader and follower functionality

Automated highway traffic system

- the **Road** motif captures the movement of *all* cars

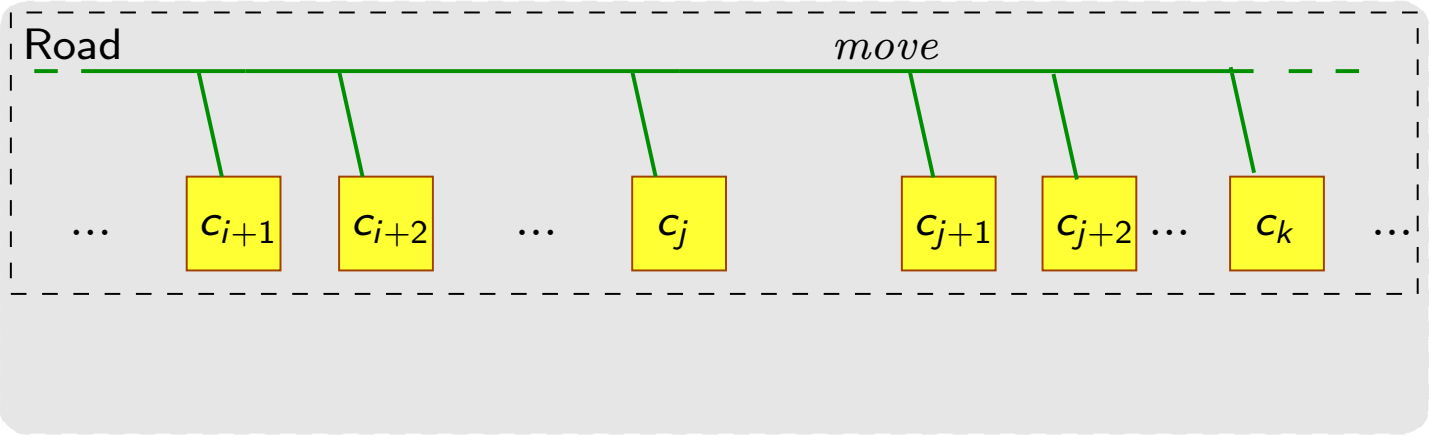


- single move interaction involving all cars

sync-road-move(X : Car) $\equiv$ <u>when</u> X=B <u>sync</u> X.move

Automated highway traffic system

- the **Road** motif captures the movement of *all* cars

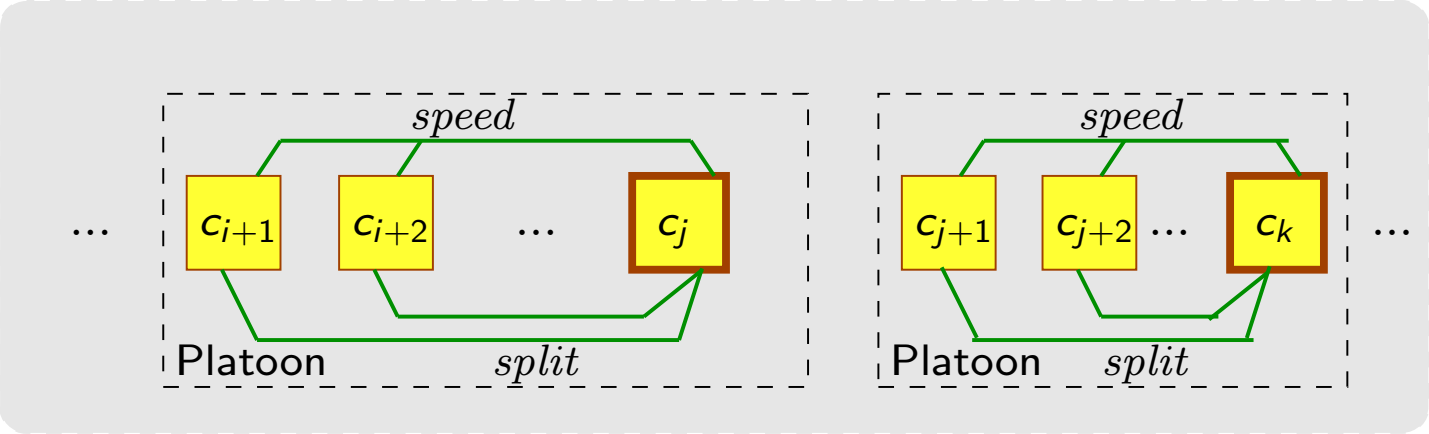


- single move interaction involving all cars

sync-road-move(X : Car) $\equiv$ <u>when</u> X=B <u>sync</u> X.move

Automated highway traffic system

- the **Platoon** motif captures the coordination in *one* platoon

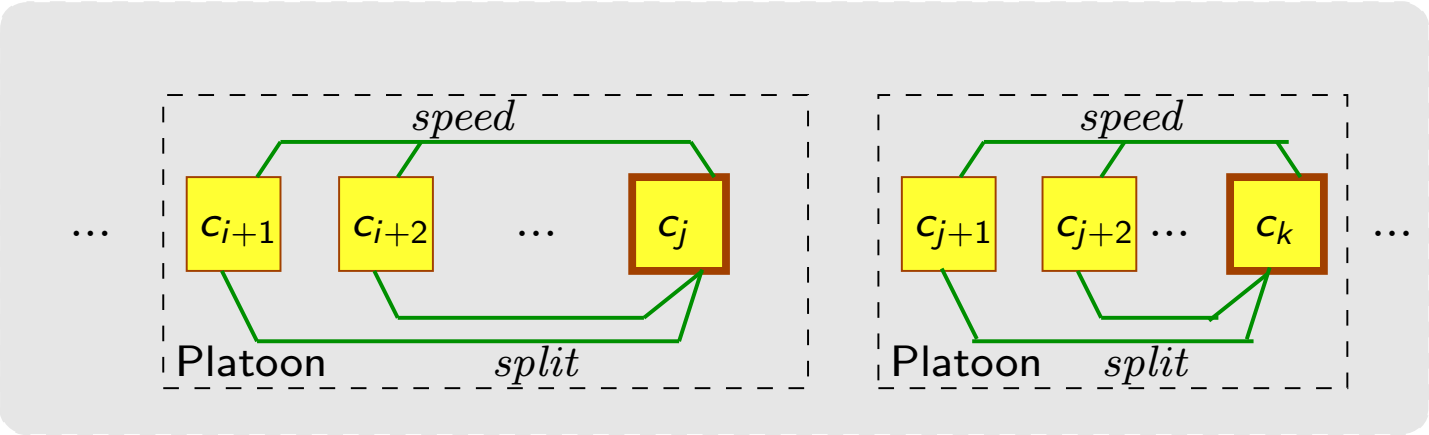


- interactions for synchronizing speed and notifying split manouver

sync-platoon-speed(x : Car, X : Car) $\equiv$ <u>when</u> X=B $\setminus$ x $\wedge$ D(x) = H.head
  <u>sync</u> x.setSpeed X.getSpeed <u>do</u> $X.v = x.v$
sync-platoon-split(x_1, x_2 : Car) $\equiv$ <u>when</u> D(x_1) = H.head $\wedge$ $x_1 \neq x_2$
  <u>sync</u> x_1.ack_split x_2.split

Automated highway traffic system

- *Platoons merge and split dynamically ...*

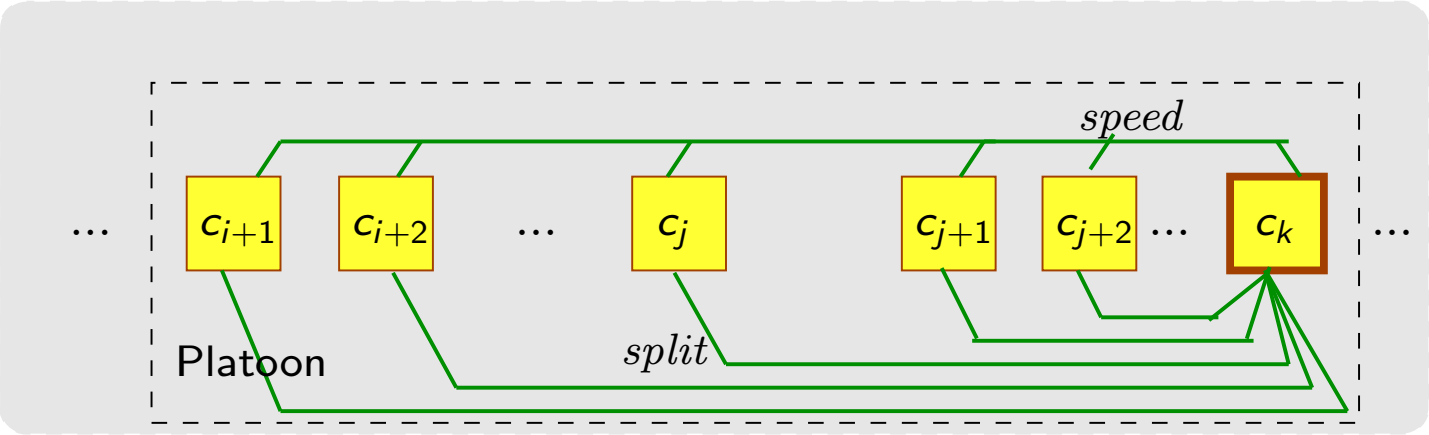


- inter-motif reconfiguration rules for merging

do-platoon-merge(y_1, y_2 : Platoon, x_1, x_2 : Car) $\equiv$
  <u>when</u> $\langle y_1 : \mathrm{D}(x_1) = \mathrm{H.tail}\rangle \wedge \langle y_2 : \mathrm{D}(x_2) = \mathrm{H.head}\rangle \wedge |x_1.\mathrm{pos} - x_2.\mathrm{pos}| < K$
  <u>do</u> $\mathrm{B} := y_1.\mathrm{B} \cup y_2.\mathrm{B}$, $\mathrm{H} := \mathrm{append}(y_2.\mathrm{H}, y_1.\mathrm{H})$, $\mathrm{D} := y_1.\mathrm{D} \cup y_2.\mathrm{D}$,
    create(P, (B, H, D)), delete(y_1), delete(y_2)

Automated highway traffic system

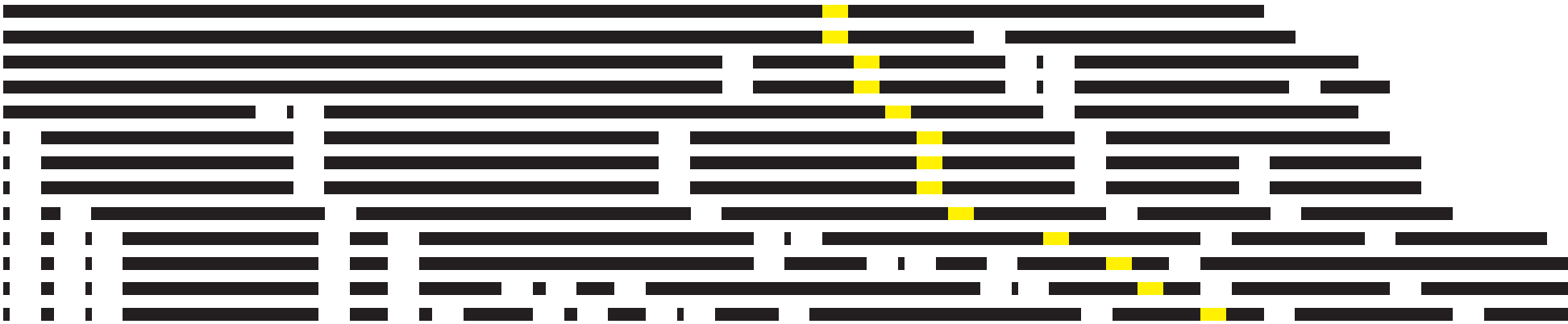
- *Platoons merge and split dynamically ...*



- inter-motif reconfiguration rules for splitting

do-platoon-split(y : Platoon, x : Car) $\equiv$
  <u>do</u> $\langle y$: $H_1 := H.\text{sublist}(0, D(x))$, $B_1 := D^{-1}(H_1)$, $D_1 := D.\text{restrict}(H_1)$,
    $H_2 := H.\text{sublist}(D(x), H.\text{length})$, $B_2 := D^{-1}(H_2)$, $D_2 := D.\text{restrict}(H_2)$ $\rangle$,
    create(P, (B_1, H_1, D_1)), create(P, (B_2, H_2, D_2)), delete(y)

Automated highway traffic system



Simulation results of the highway traffic with 200 cars along 2000 steps (the yellow rectangle designates some fixed car)

Self-organized robot colonies

- a number of identical robots are randomly deployed on a field and have a mission to locate an object (the prey) and to bring it near another object (the nest)
- the robots know neither the position of the nest nor the position of the prey; they have limited communication and sensing capabilities

Self-organized robot colonies

Two phase swarm algorithm proposed by [Dorigo et al, *Teamwork in self-organized robot colonies*, IEEE Transactions on Evolutionary Computation, 2009]

1. the robots self-organize into an *exploration path* starting at the nest
   - the first robot detecting the nest initiates the path, i.e. stops moving and displays a specific status
   - any robot that detects the path, begins moving along the path towards its tail, explores a bit further its neighborhood and gets connected
2. the robots push the prey back to the nest along the *exploration path*
   - additional robots converge near the prey
   - when enough robots have converged, they start pushing the prey along the path towards the nest

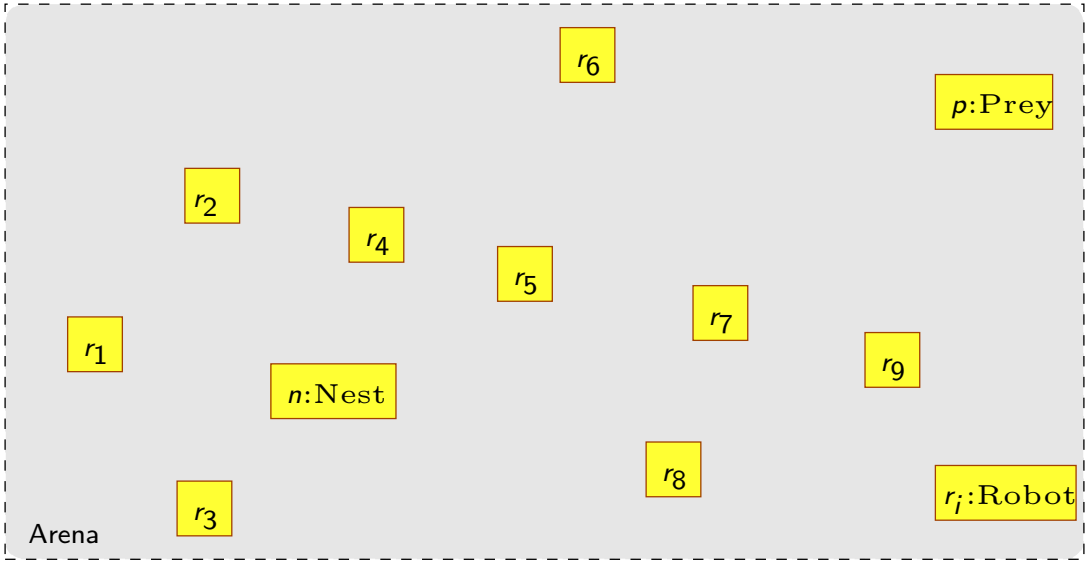
Self-organized robot colonies

Two phase swarm algorithm proposed by [Dorigo et al, *Teamwork in self-organized robot colonies*, IEEE Transactions on Evolutionary Computation, 2009]

1. the robots self-organize into an *exploration path* starting at the nest
   - the first robot detecting the nest initiates the path, i.e. stops moving and displays a specific status
   - any robot that detects the path, begins moving along the path towards its tail, explores a bit further its neighborhood and gets connected
2. the robots push the prey back to the nest along the *exploration path*
   - additional robots converge near the prey
   - when enough robots have converged, they start pushing the prey along the path towards the nest

Self-organized robot colonies

- the **Arena** pattern contains all the components (robots, nest, prey)

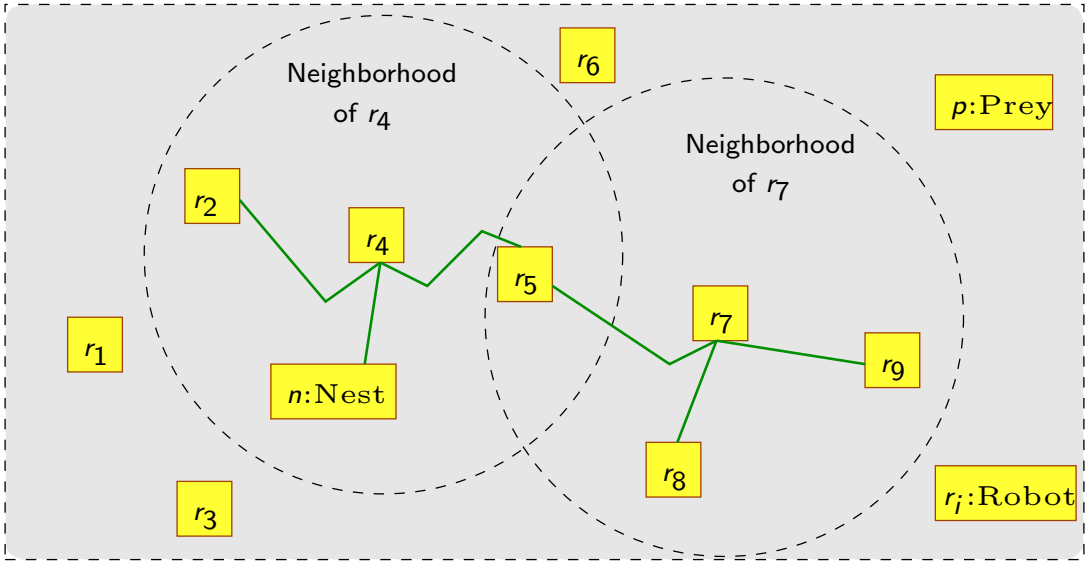


- no particular map/deployment structure
- defines a global *tick* interaction for modeling synchronous time progress
- all robots move i.e. update their position at every tick

sync-arena-tick(X: Robot) $\equiv$ <u>when</u> X = B.Robot <u>sync</u> X.tick

Self-organized robot colonies

- the **Neighborhood** pattern represents the visibility range for a given robot

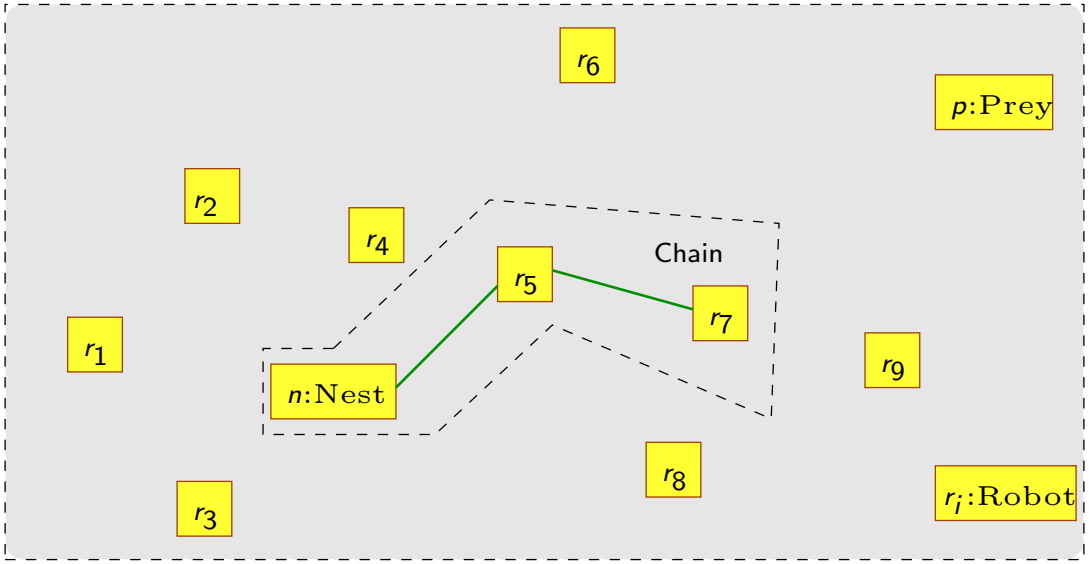


- inter-motif reconfiguration rules keep the neighborhood updated

do-neighborhood-enter(y_1 : Neighborhood, y_2 : Arena, x_1, x_2: Robot) $\equiv$
  <u>when</u> $\langle y_1 : \mathrm{D}(x_1) = \mathrm{H.center} \wedge x_2 \notin \mathrm{B} \rangle \wedge \langle y_2 : x_2 \in \mathrm{B} \rangle \wedge \mathrm{dist}(x_1, x_2) \leq R_{min}$
    <u>do</u> $y_1.\mathrm{migrate}(x_2)$, $\langle y_1 : n := \mathrm{H.extend}(), \mathrm{D}(x_2) := n \rangle$
do-neighborhood-leave(y_1 : Neighborhood, x_1, x_2: Robot) $\equiv$
  <u>when</u> $\langle y_1 : \mathrm{D}(x_1) = \mathrm{H.center} \wedge x_2 \in \mathrm{B} \rangle \wedge x_1 \neq x_2 \wedge \mathrm{dist}(x_1, x_2) \geq R_{max}$
    <u>do</u> $\langle y_1 : n := \mathrm{D}(x_2), \mathrm{B.delete}(x_2), \mathrm{H.remove}(n) \rangle$

Self-organized robot colonies

- the **Chain** pattern represents the *exploration path* constructed dynamically

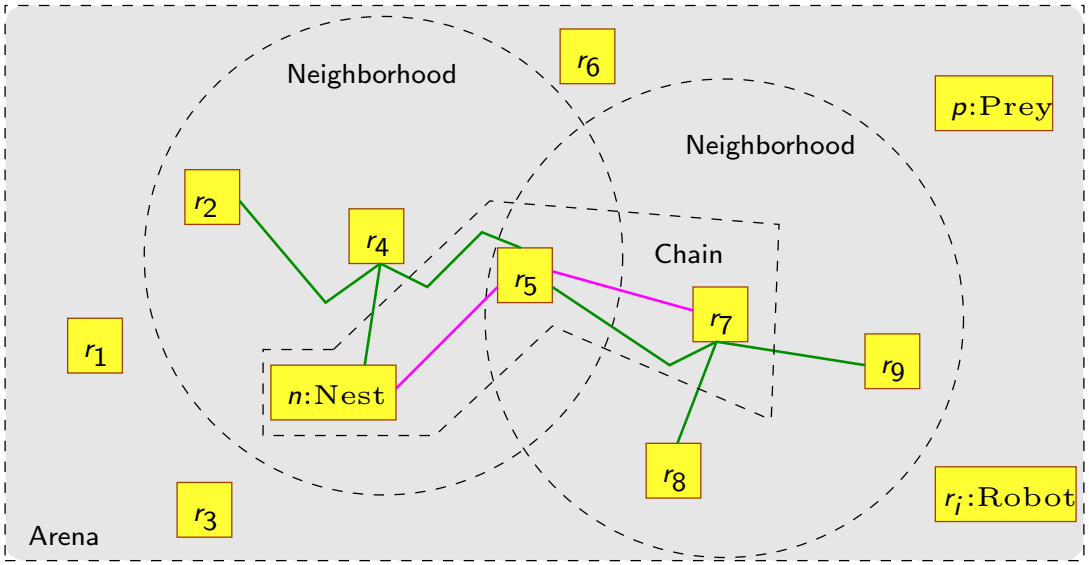


- inter-motif reconfiguration let every robot join/leave the chain

do-chain-connect(y_1 : Chain, y_2 : Neighborhood, x_1, x_2 : Robot) $\equiv$
  <u>when</u> $\langle y_1 : \mathrm{D}(x_1) = \mathrm{H.tail} \wedge x_2 \notin \mathrm{B} \rangle \wedge \langle y_2 : \mathrm{D}(x_1) = \mathrm{H.center} \wedge x_2 \in \mathrm{B} \rangle$
    <u>do</u> y_1.migrate(x_2), $\langle y_1 : n = \mathrm{H.extend}(), \mathrm{D}(x_2) := n \rangle$
do-chain-disconnect(y_1 : Chain, x_1 : Robot) $\equiv$
  <u>when</u> $\langle y_1 : \mathrm{D}(x_1) = \mathrm{H.tail} \rangle \wedge \langle y_1 : x_1.timeout = \mathrm{true} \rangle$
    <u>do</u> $\langle y_1 : n := \mathrm{D}(x_1), \mathrm{B.delete}(x_1), \mathrm{H.remove}(n) \rangle$

Self-organized robot colonies

- overall view of the motif-based system



- dynamic architecture and behaviour fully defined as the superposition of **Arena**, **Neighborhood**(s) and **Chain** motifs

Outline

Conclusion

DR-BIP provides architectural motifs:

- an *exogeneous framework* for dynamic architectures
  - based on elementary structuring concepts of *map* and *deployment*
  - fully *declarative* for interaction definition
  - mixed declarative/imperative for reconfiguration
- three basic types of architectural dynamism
  - behavior dynamism: component *instances creation* and *deletion*
  - deployment dynamism: *mobility* of component instances along map
  - map dynamism: map updates according to specific primitives
- operational semantics
  - built on operational semantics for static BIP systems
  - prototype implementation available

Future Work

Towards automated verification of DR-BIP models

- *correctness of parametric interacting systems*
  - extract parametric interaction invariants for static systems with arbitrary (yet fixed) number of components
  - combine with component invariants and prove system level properties
  - e.g., show that the dynamic ring system is deadlock-free regardless the number of components
- *correctness of reconfiguration operations*
  - prove that architectural properties are preserved by statements changing components, the map and the deployment
  - e.g., show that the ring property is preserved by every reconfiguration, that is, an architectural invariant

Beyond simulation,
implementation for existing middleware platforms e.g., OSGI

Future Work

Towards automated verification of DR-BIP models

- *correctness of parametric interacting systems*
  - extract parametric interaction invariants for static systems with arbitrary (yet fixed) number of components
  - combine with component invariants and prove system level properties
  - e.g., show that the dynamic ring system is deadlock-free regardless the number of components
- *correctness of reconfiguration operations*
  - prove that architectural properties are preserved by statements changing components, the map and the deployment
  - e.g., show that the ring property is preserved by every reconfiguration, that is, an architectural invariant

Beyond simulation,
implementation for existing middleware platforms e.g., OSGI

Future Work

Towards automated verification of DR-BIP models

- *correctness of parametric interacting systems*
  - extract parametric interaction invariants for static systems with arbitrary (yet fixed) number of components
  - combine with component invariants and prove system level properties
  - e.g., show that the dynamic ring system is deadlock-free regardless the number of components
- *correctness of reconfiguration operations*
  - prove that architectural properties are preserved by statements changing components, the map and the deployment
  - e.g., show that the ring property is preserved by every reconfiguration, that is, an architectural invariant

Beyond simulation,
implementation for existing middleware platforms e.g., OSGI

Thank you for your attention!

Questions ?

The research leading to these results has received funding from the European Union Horizon 2020 Research and Innovation Programme under Grant Agreement no. 700665 CITADEL - Critical Infrastructure Protection using Adaptive MILS.