STKM on SCA: A Unified Framework with Components, Workflows and Algorithmic Skeletons

Marco Aldinucci
Computer Science Dept.
University of Torino - Italy

Hinde-Lilia Bouziane
Project-team GRAAL
INRIA Rhône-Alpes - ENS Lyon - France

Marco Danelutto
Computer Science Dept.
University of Pisa - Italy

Christian Pérez
Project-team GRAAL
INRIA Rhône-Alpes - ENS Lyon - France

EuroPar 2009 - 27th Aug 2009 - Delft

Outline

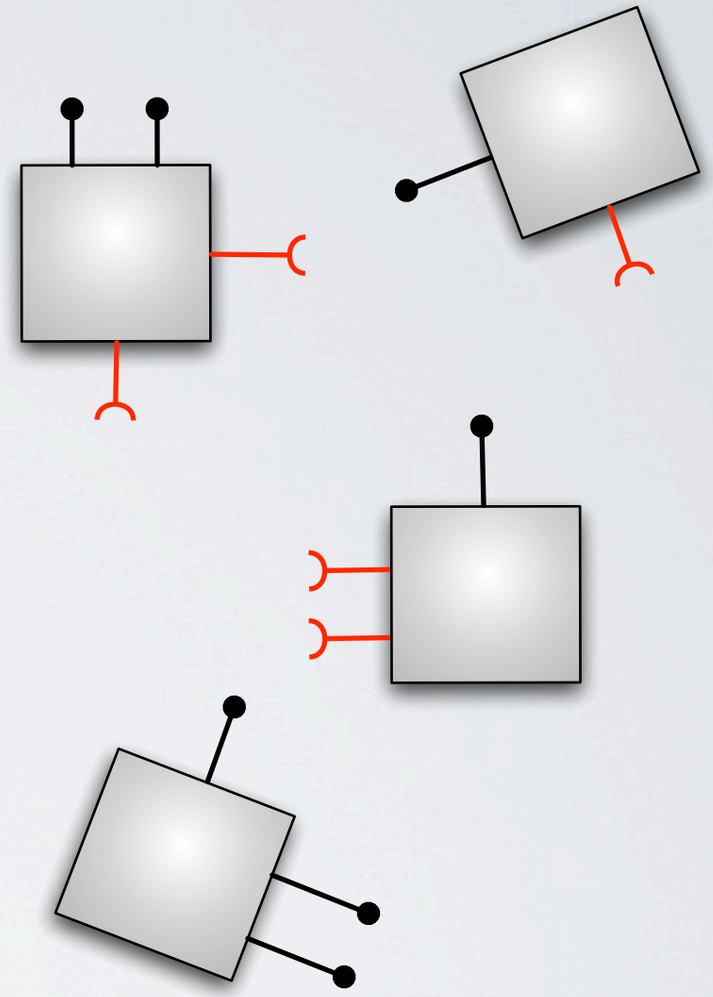
- Introduction
  - Software component models
- Existing models
  - STCM: a spatio-temporal component model
  - Skeletons based models for parallel programming
- **STKM**: a proposal of skeletons introduction in STCM
- A SCA-based implementation of STKM
  - Overview of SCA
  - A projection of STKM on SCA
- Experiments
- Conclusions and future works

Context & Approach

- Programming simply and independently from execution resources
- Complex applications
  - Size, Heterogeneity
  - Different applicative domains
  - Reuse
  - Reuse of legacy code
  - Explicit control of dependencies
- Approach: component models
  - Examples: CCM (OMG), GCM (NoE CoreGrid), SCA (OSOA), CCA (CCA Forum-USA), etc.

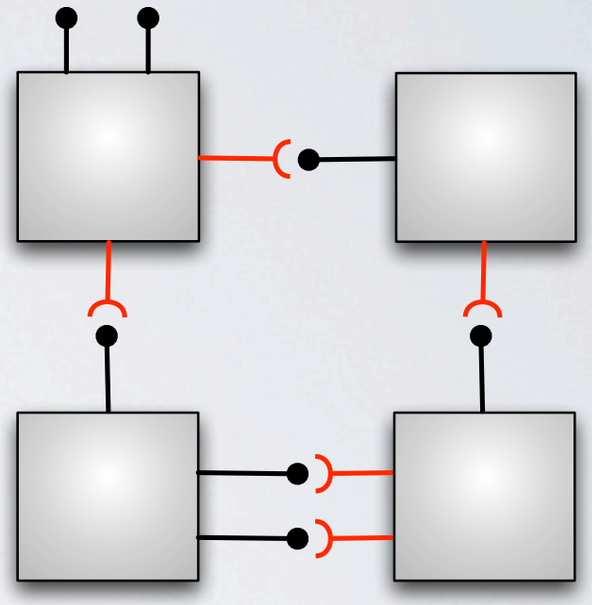
Context & Approach

- Programming simply and independently from execution resources
- Complex applications
  - Size, Heterogeneity
  - Different applicative domains
  - Reuse
  - Reuse of legacy code
  - Explicit control of dependencies
- Approach: component models
  - Examples: CCM (OMG), GCM (NoE CoreGrid), SCA (OSOA), CCA (CCA Forum-USA), etc.



Context & Approach

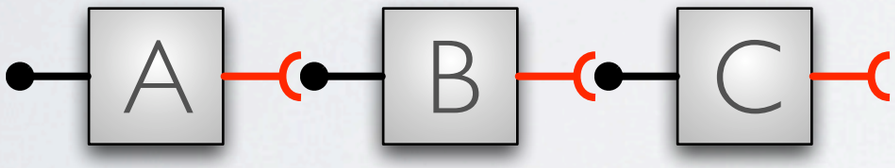
- Programming simply and independently from execution resources
- Complex applications
  - Size, Heterogeneity
  - Different applicative domains
  - Reuse
  - Reuse of legacy code
  - Explicit control of dependencies
- Approach: component models
  - Examples: CCM (OMG), GCM (NoE CoreGrid), SCA (OSOA), CCA (CCA Forum-USA), etc.



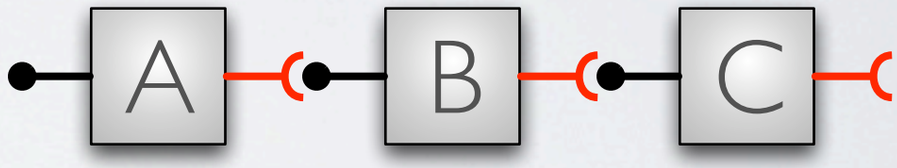
Components & Workflows

- Low-level of abstraction
  - Behavior is hidden in the assembly
  - Component assembly model very close to computing resources

Spatial dependency
(resource dependency - complex design)



Temporal dependency
(resource usage - resource overcomsumption)



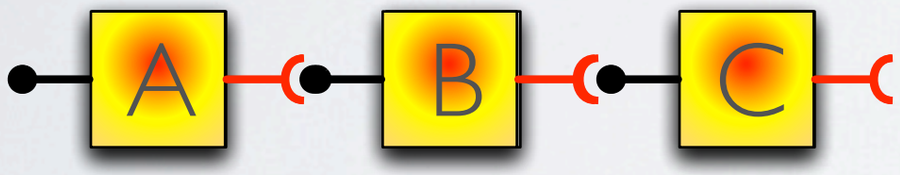
Component Assembly

Workflow

Components & Workflows

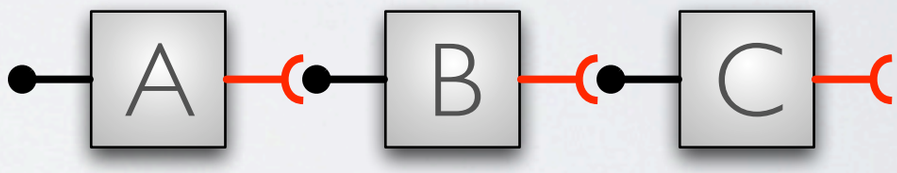
- Low-level of abstraction
  - Behavior is hidden in the assembly
  - Component assembly model very close to computing resources

Spatial dependency
(resource dependency - complex design)



Component Assembly

Temporal dependency
(resource usage - resource overcomsumption)



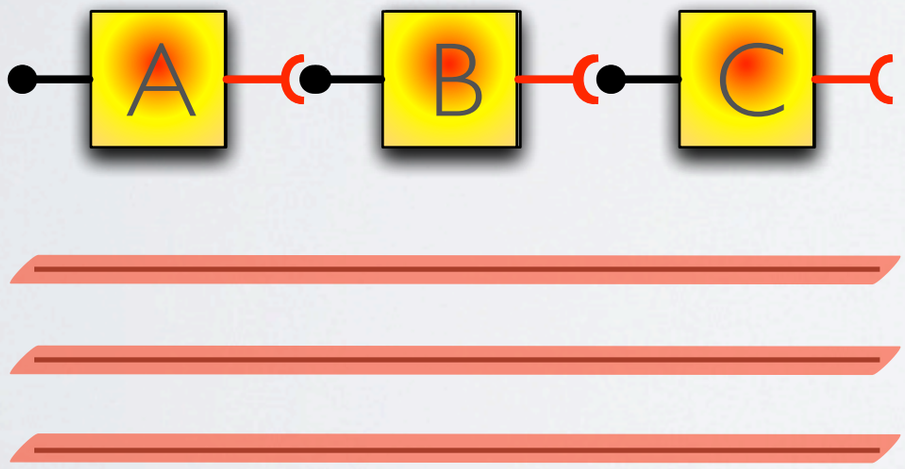
time

Workflow

Components & Workflows

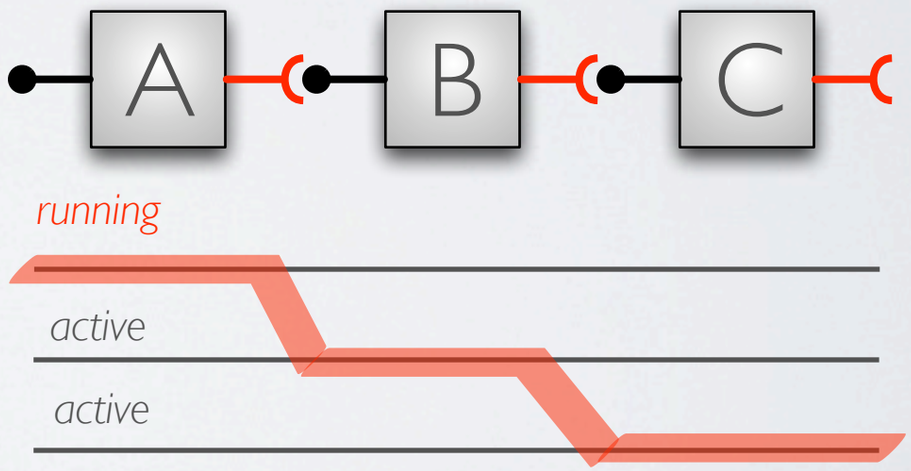
- Low-level of abstraction
  - Behavior is hidden in the assembly
  - Component assembly model very close to computing resources

Spatial dependency
(resource dependency - complex design)



Component Assembly

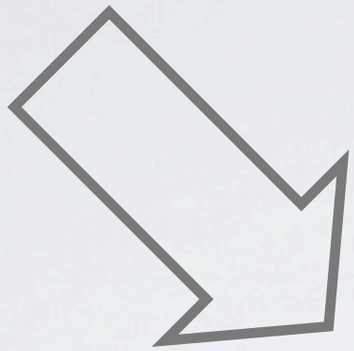
Temporal dependency
(resource usage - resource overcomsumption)



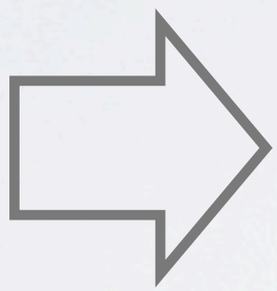
Workflow

Big Picture

Components
spatial dependencies
require co-allocation



Workflows
temporal dependencies
admit scheduling



STCM
spatio-temporal
component model

Big Picture

Components
spatial dependencies
require co-allocation

Behavioral Skeletons
parametric assemblies
component factories
autonomic QoS management

Workflows
temporal dependencies
admit scheduling

STCM
spatio-temporal
component model

STKM
spatio-temporal
model with skeletons

Big Picture

Components
spatial dependencies
require co-allocation

Behavioral Skeletons
parametric assemblies
component factories
autonomic QoS management

Workflows
temporal dependencies
admit scheduling

STCM
spatio-temporal
component model

Services
SCA/Tuscany
components as services

STKM
SCA
implementation

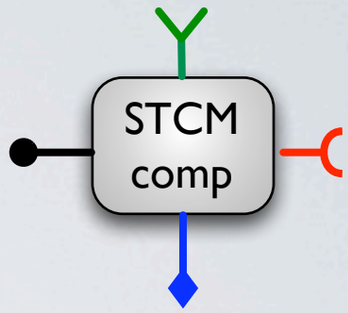
STKM
spatio-temporal
model with skeletons

Outline

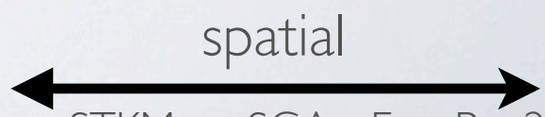
- Introduction
  - Software component models
- **Existing models**
  - STCM: a spatio-temporal component model
  - Skeletons based models for parallel programming
- **STKM**: a proposal of skeletons introduction in STCM
- A SCA-based implementation of STKM
  - Overview of SCA
  - A projection of STKM on SCA
- Experiments
- Conclusions and future works

STCM (EuroPar 08)

- Combination of component and workflow models
  - Spatial and temporal dimensions at the same level of assembly
- Component-task
  - Spatial ports (classical ones)
  - Input and output ports (temporal)
  - Task
- Assembly model
  - Adaptation of a workflow language

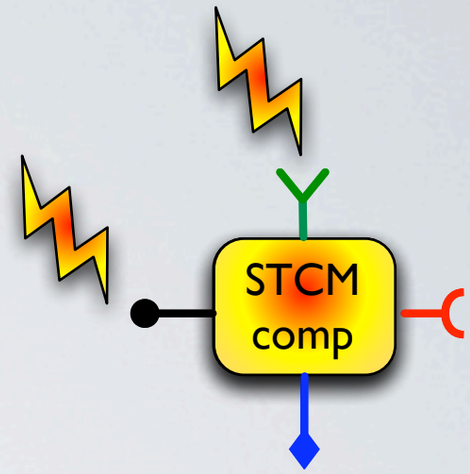


Bouziane, H., Pérez, C., Priol, T.: **A Software Component Model with Spatial and Temporal Compositions for Grid Infrastructures**. In: Proc. of the 14th Intl. Euro-Par Conference. Vol. 5168., Las Palmas de Gran Canaria, Spain, Springer (August 2008) 698–708

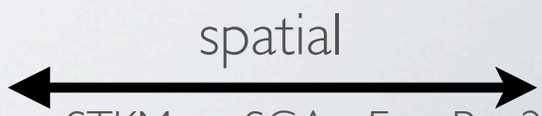


STCM (EuroPar 08)

- Combination of component and workflow models
  - Spatial and temporal dimensions at the same level of assembly
- Component-task
  - Spatial ports (classical ones)
  - Input and output ports (temporal)
  - Task
- Assembly model
  - Adaptation of a workflow language

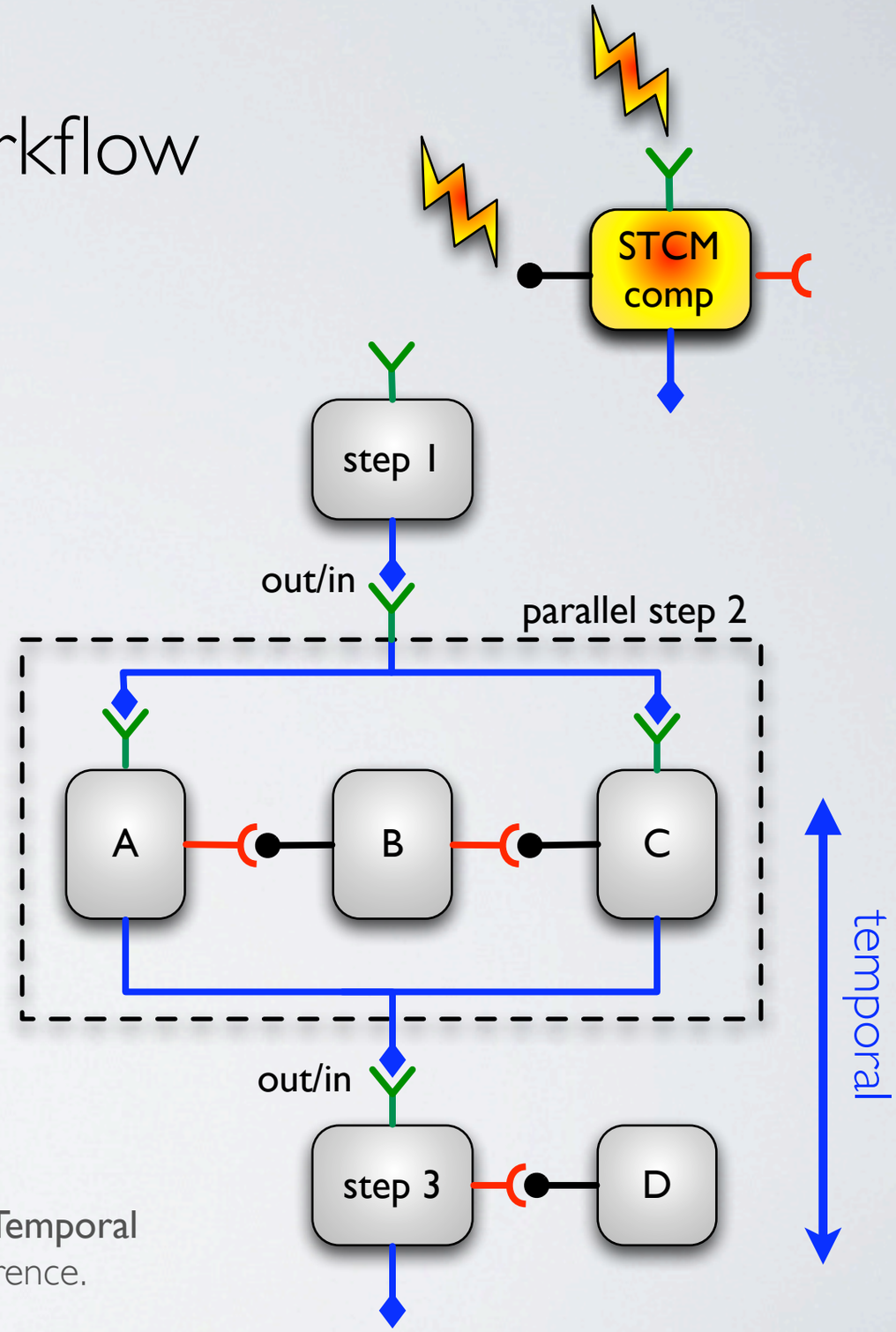


Bouziane, H., Pérez, C., Priol, T.: **A Software Component Model with Spatial and Temporal Compositions for Grid Infrastructures**. In: Proc. of the 14th Intl. Euro-Par Conference. Vol. 5168., Las Palmas de Gran Canaria, Spain, Springer (August 2008) 698–708



STCM (EuroPar 08)

- Combination of component and workflow models
  - Spatial and temporal dimensions at the same level of assembly
- Component-task
  - Spatial ports (classical ones)
  - Input and output ports (temporal)
  - Task
- Assembly model
  - Adaptation of a workflow language

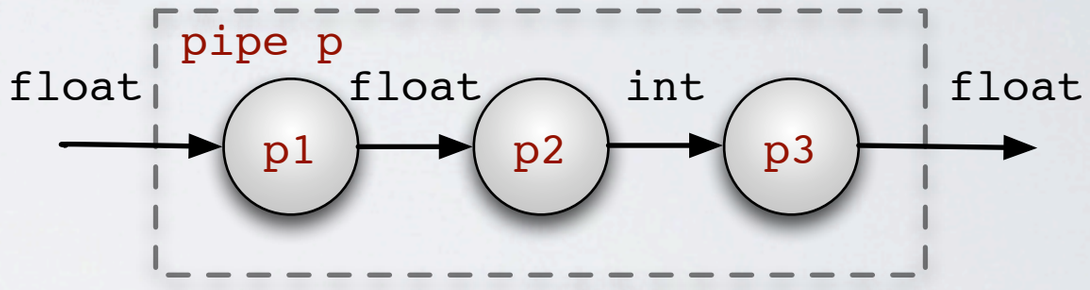


Bouziane, H., Pérez, C., Priol, T.: **A Software Component Model with Spatial and Temporal Compositions for Grid Infrastructures**. In: Proc. of the 14th Intl. Euro-Par Conference. Vol. 5168., Las Palmas de Gran Canaria, Spain, Springer (August 2008) 698–708

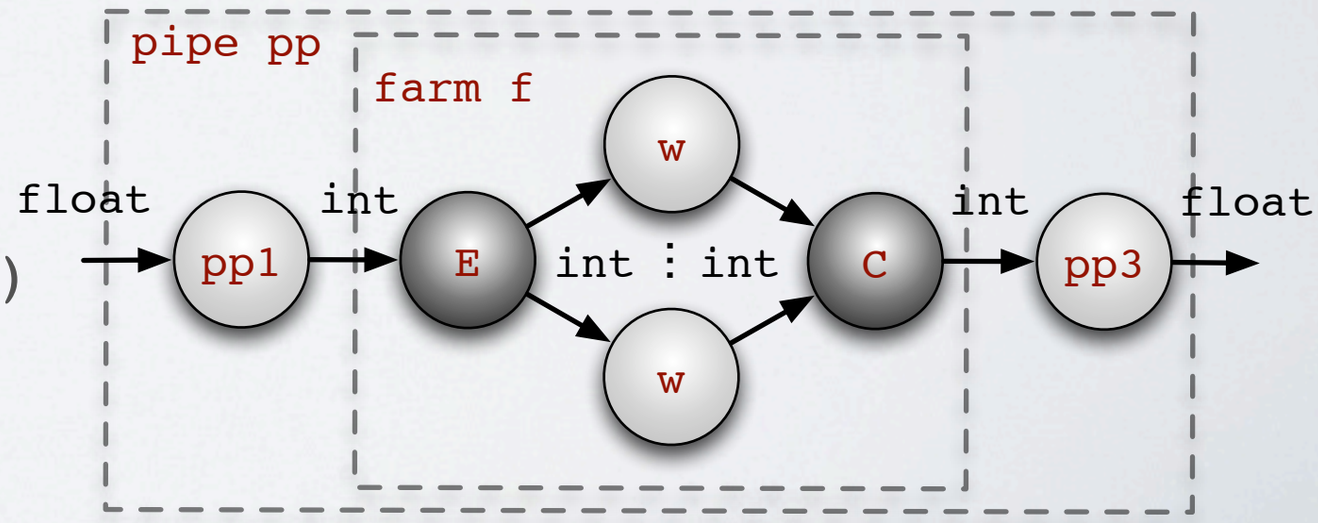
Algorithmic Skeletons (Cole 1989)

- Structured programming (simplicity/correctness of programs)
- Hide the complexity of parallelism setup and data distribution
- Behavioral skeletons (advanced management for adaptation)

```
pipe p in (int a) out (float b)
       p1 in (a)  out (float b1)
       p2 in (b1) out (int b2)
       p3 in (b2) out (b)
end pipe
```




```
farm f in (int af) out(int bf)
        w in (af) out(bf)
end farm
pipe pp in (float a) out(float b)
        pp1 in (b)  out (int b1)
        f   in (b1) out (b2)
        pp3 in (b2) out (b)
end pipe
```



Outline

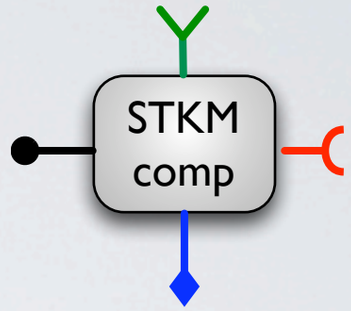
- Introduction
  - Software component models
- Existing models
  - STCM: a spatio-temporal component model
  - Skeletons based models for parallel programming
- **STKM: a proposal of skeletons introduction in STCM**
- A SCA-based implementation of STKM
  - Overview of SCA
  - A projection of STKM on SCA
- Experiments
- Conclusions and future works

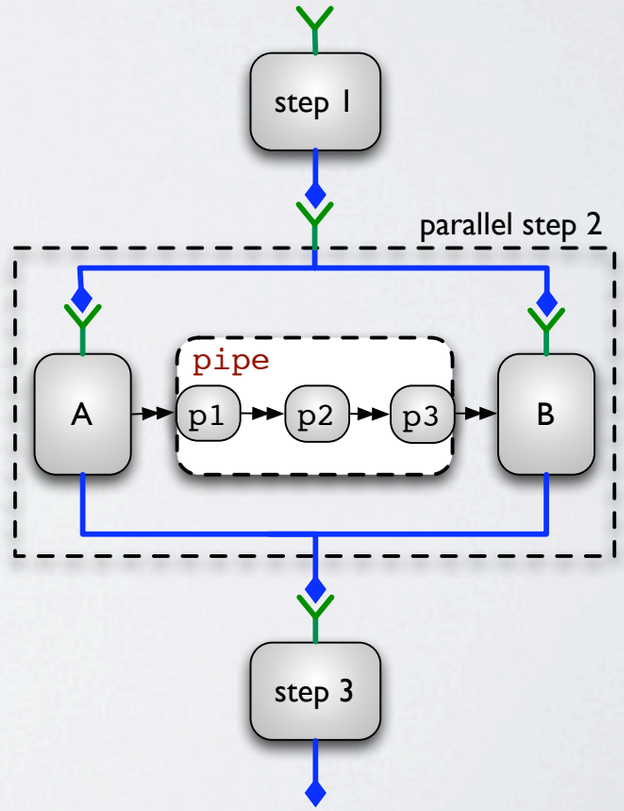
Objectives

- Bringing together suited properties
  - Code reuse facility (Component models)
  - Capability of resources usage optimization (Workflows)
  - Simplicity of programming parallel parts of an application (Skeletons)
- Portability on different execution resources
  - Code reuse
  - Efficiency

STKM (CBHPC 08) - Assembly model 1/2

- Extension of STCM
  - Assembly language with skeletons constructs (**pipe** and **farm**)
- An STKM skeleton construct
  - Is a composite with a predefined behavior
  - Parameters
    - Skeleton's input/output data
    - User's components
  - Can be exploited in **spatial** and **temporal** dimension

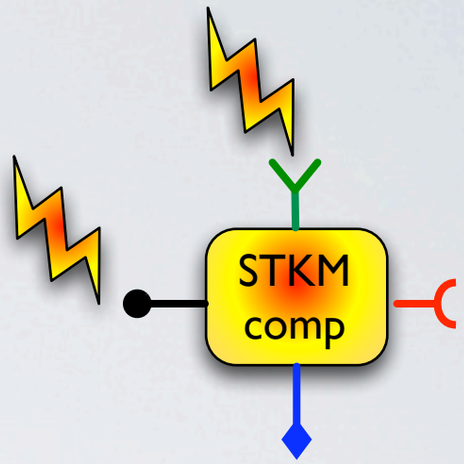


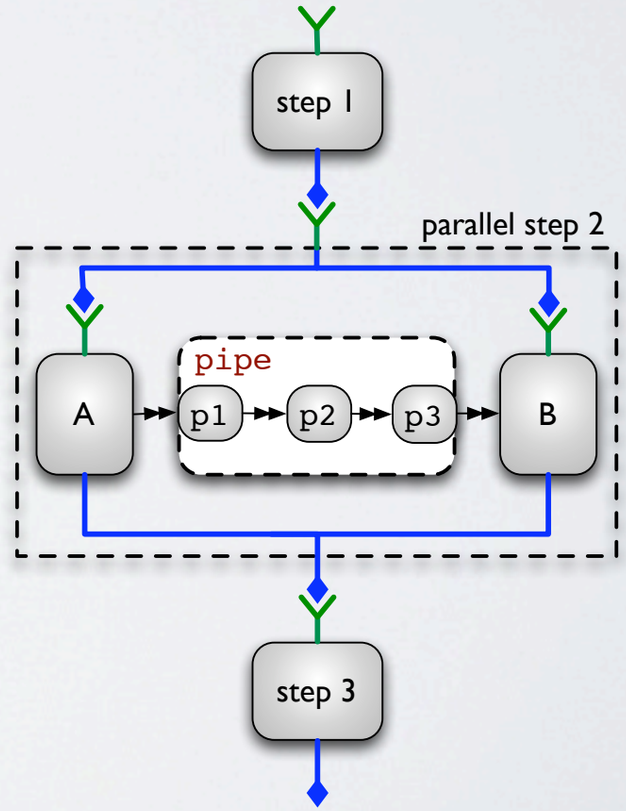


M. Aldinucci, M. Danelutto, H. L. Bouziane, and C. Pérez. **Towards software component assembly language enhanced with workflows and skeletons.** In Proc. of the ACM SIGPLAN Component-Based High Performance Computing (CBHPC), pages 1–11, New York, NY, USA, Oct. 2008. ACM.

STKM (CBHPC 08) - Assembly model 1/2

- Extension of STCM
  - Assembly language with skeletons constructs (**pipe** and **farm**)
- An STKM skeleton construct
  - Is a composite with a predefined behavior
  - Parameters
    - Skeleton's input/output data
    - User's components
  - Can be exploited in **spatial** and **temporal** dimension

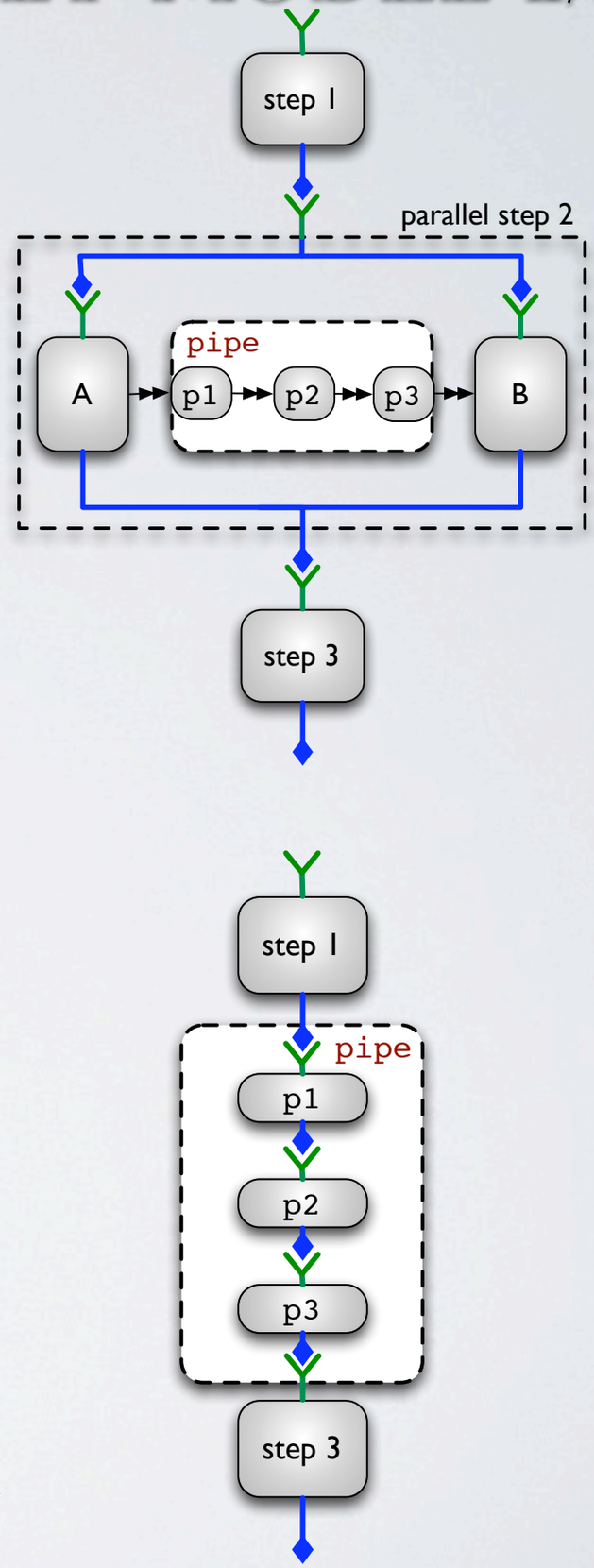




M. Aldinucci, M. Danelutto, H. L. Bouziane, and C. Pérez. **Towards software component assembly language enhanced with workflows and skeletons.** In Proc. of the ACM SIGPLAN Component-Based High Performance Computing (CBHPC), pages 1–11, New York, NY, USA, Oct. 2008. ACM.

STKM (CBHPC 08) - Assembly model 1/2

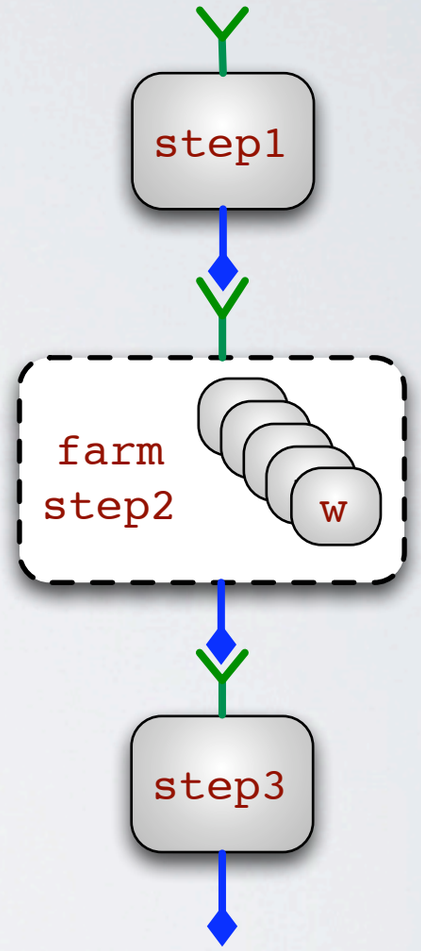
- Extension of STCM
  - Assembly language with skeletons constructs (**pipe** and **farm**)
- An STKM skeleton construct
  - Is a composite with a predefined behavior
  - Parameters
    - Skeleton's input/output data
    - User's components
  - Can be exploited in **spatial** and **temporal** dimension



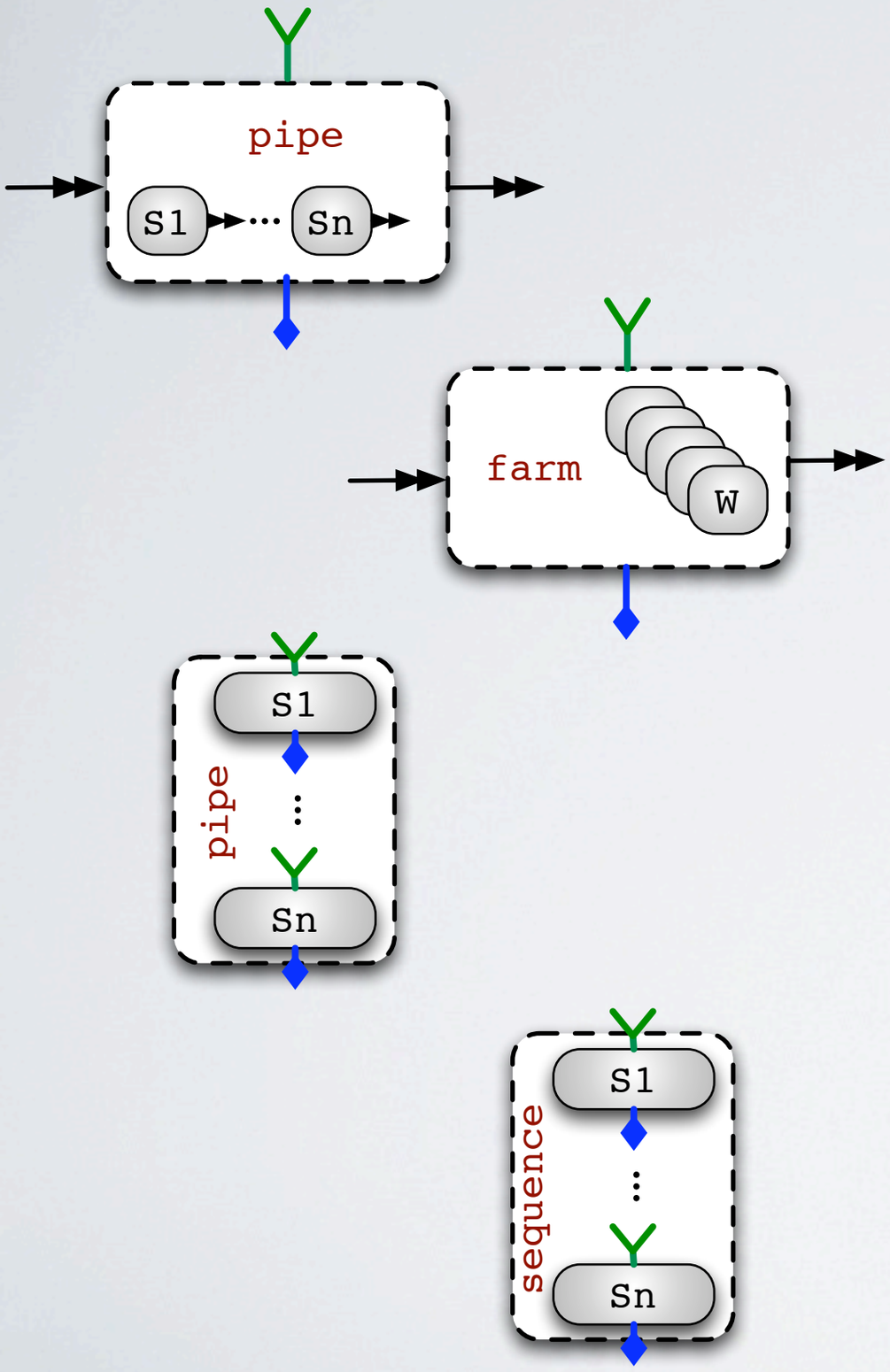
M. Aldinucci, M. Danelutto, H. L. Bouziane, and C. Pérez. **Towards software component assembly language enhanced with workflows and skeletons.** In Proc. of the ACM SIGPLAN Component-Based High Performance Computing (CBHPC), pages 1–11, New York, NY, USA, Oct. 2008. ACM.

STKM (CBHPC 08) - Assembly model 2/2

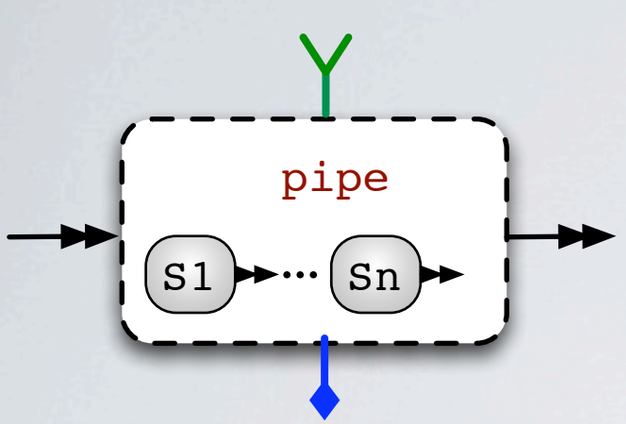
```
component Example{
   … Step1 and Step3 components…

   farm step2{
      inputSkel double inS2;
      outputSkel string outS2;

      worker sequential w {
         inputSkel double inW;
         outputSkel string outW;
         component Worker{streamIn double inW;
                          streamOut string outW;
         };
         connect outW to Worker.outW;
         connect Worker.inW to inW;
   };

instances:  Step1 step1; Step2 step2; Step3 step3;
 … Connections step1 <=> step2 <=> step3 …
sequence ApplMain{
   exectask(step1); exectask(step2); exectask(step3);
};
```

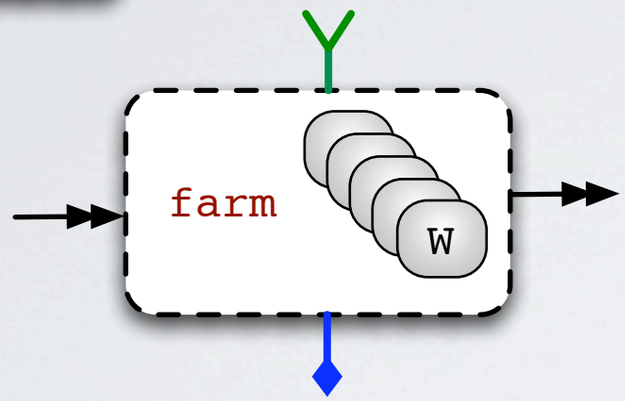


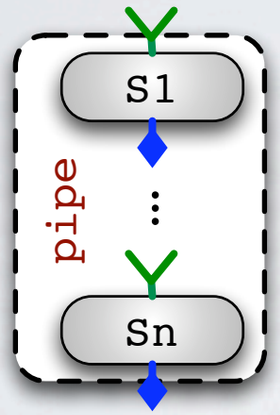
Skeletons: User's view & Implementation

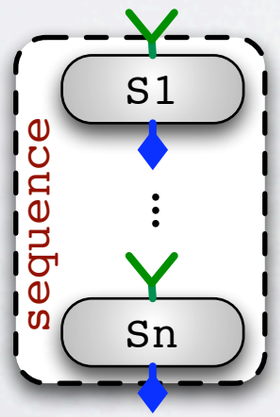


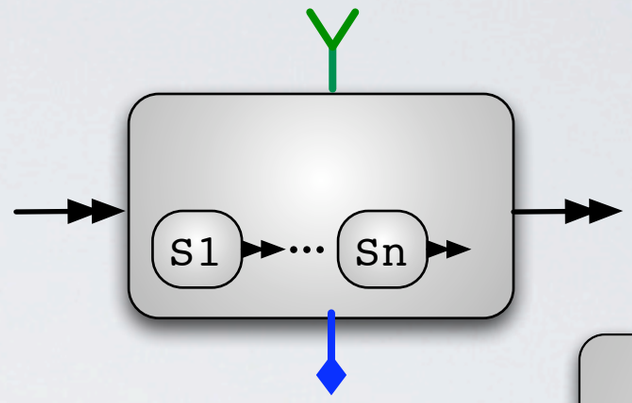
Skeletons: User's view & Implementation

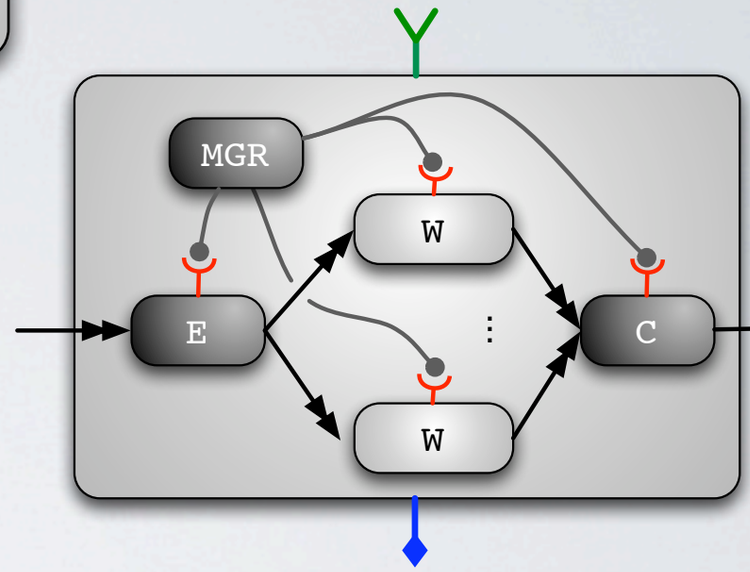


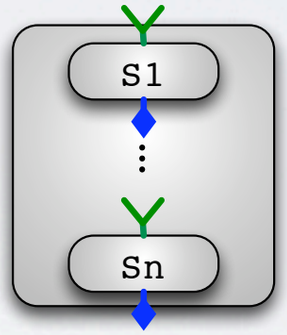


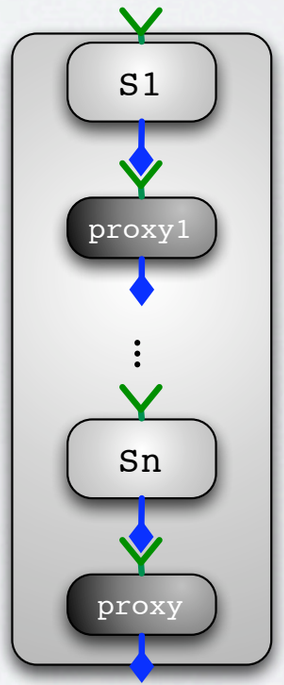




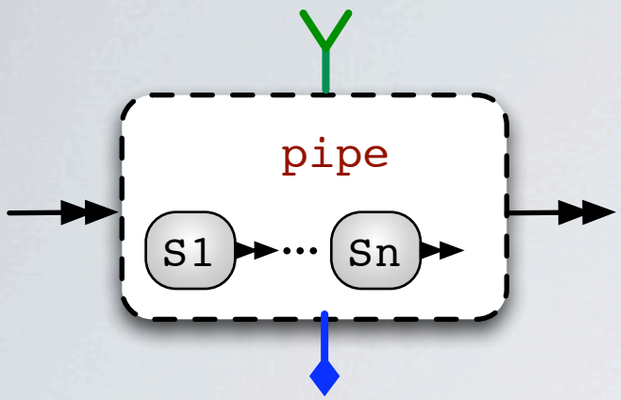


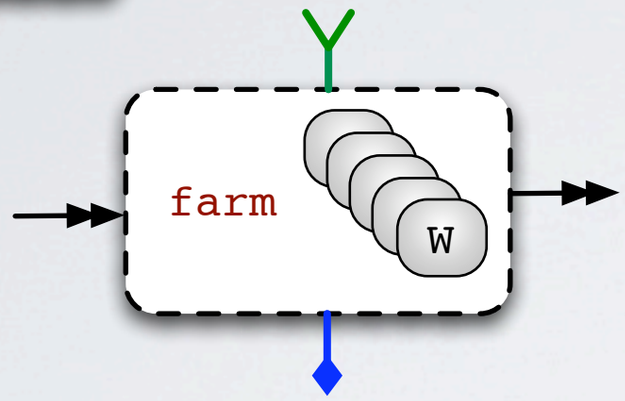


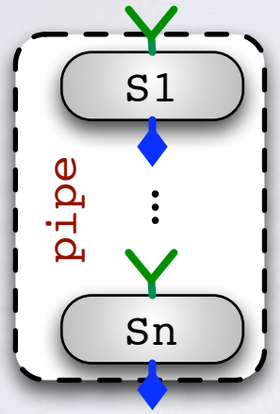


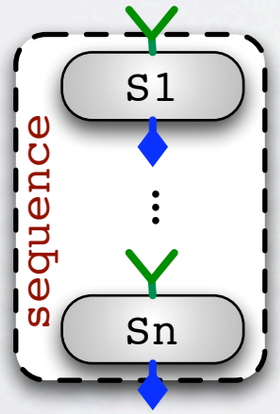


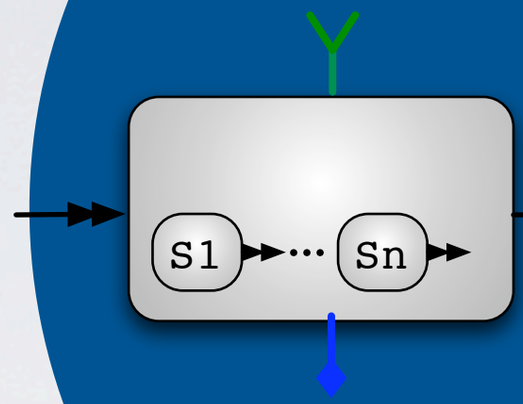
Skeletons: User's view & Implementation

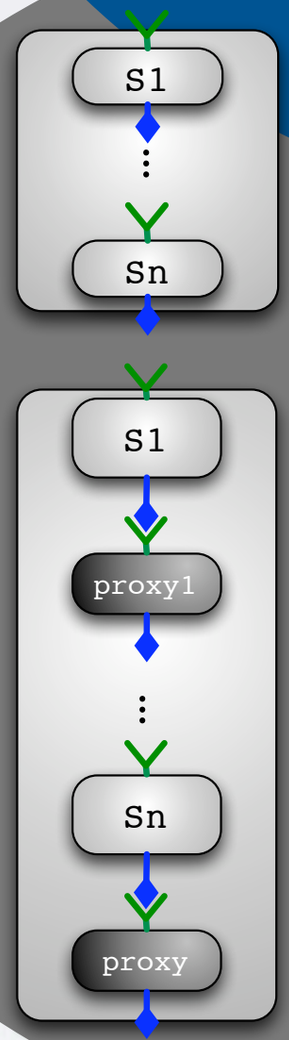


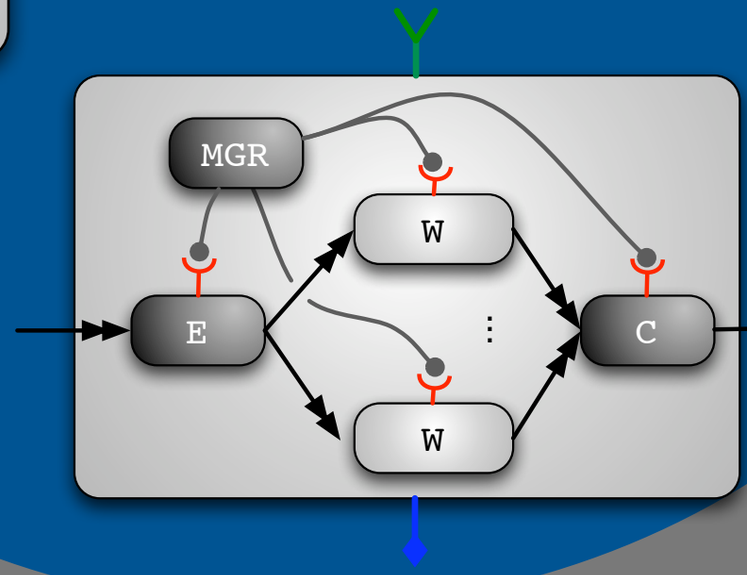










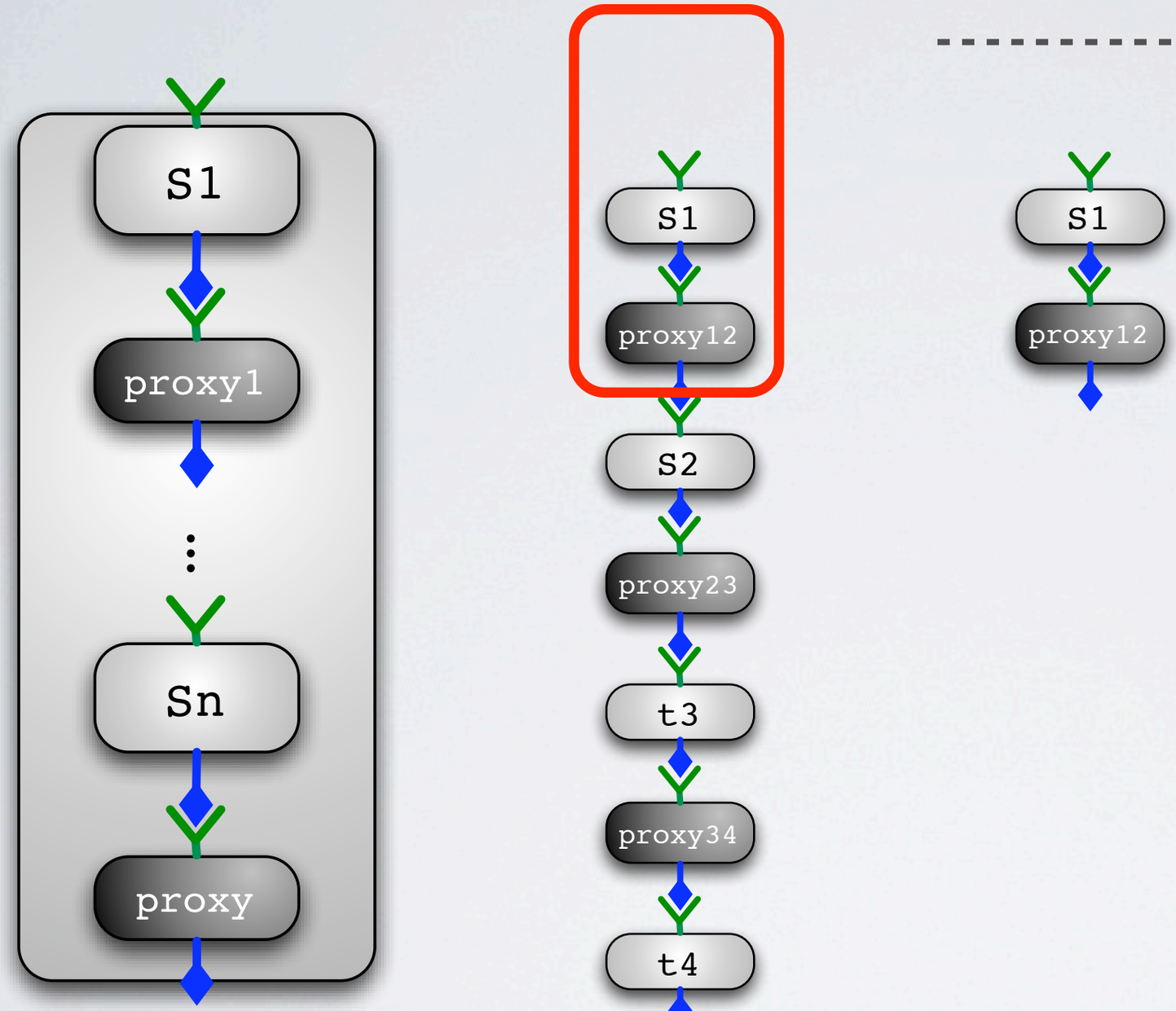


all internal components are created at once

internal components are created when needed, then either destroyed or not

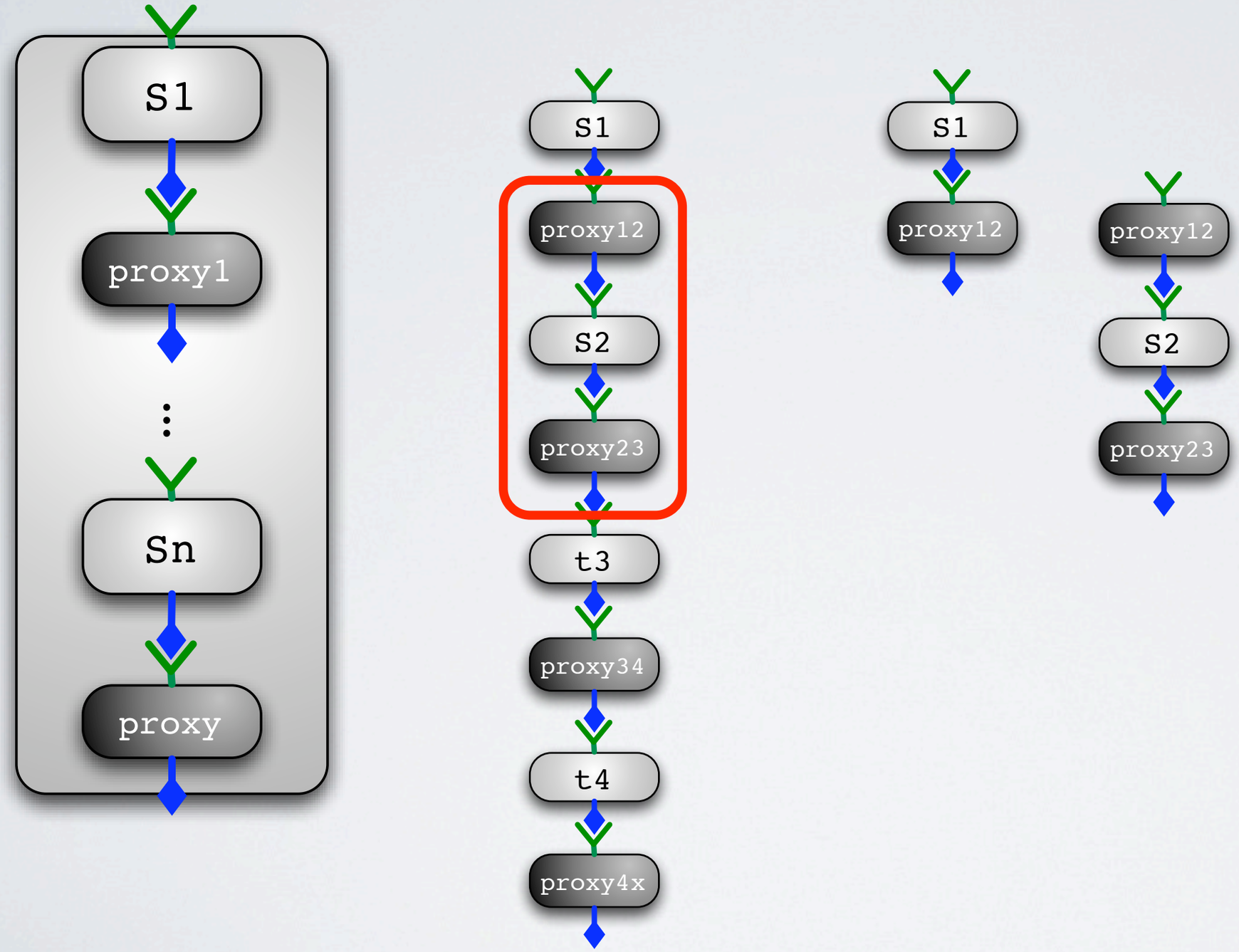
Execution Example: Sequence

time



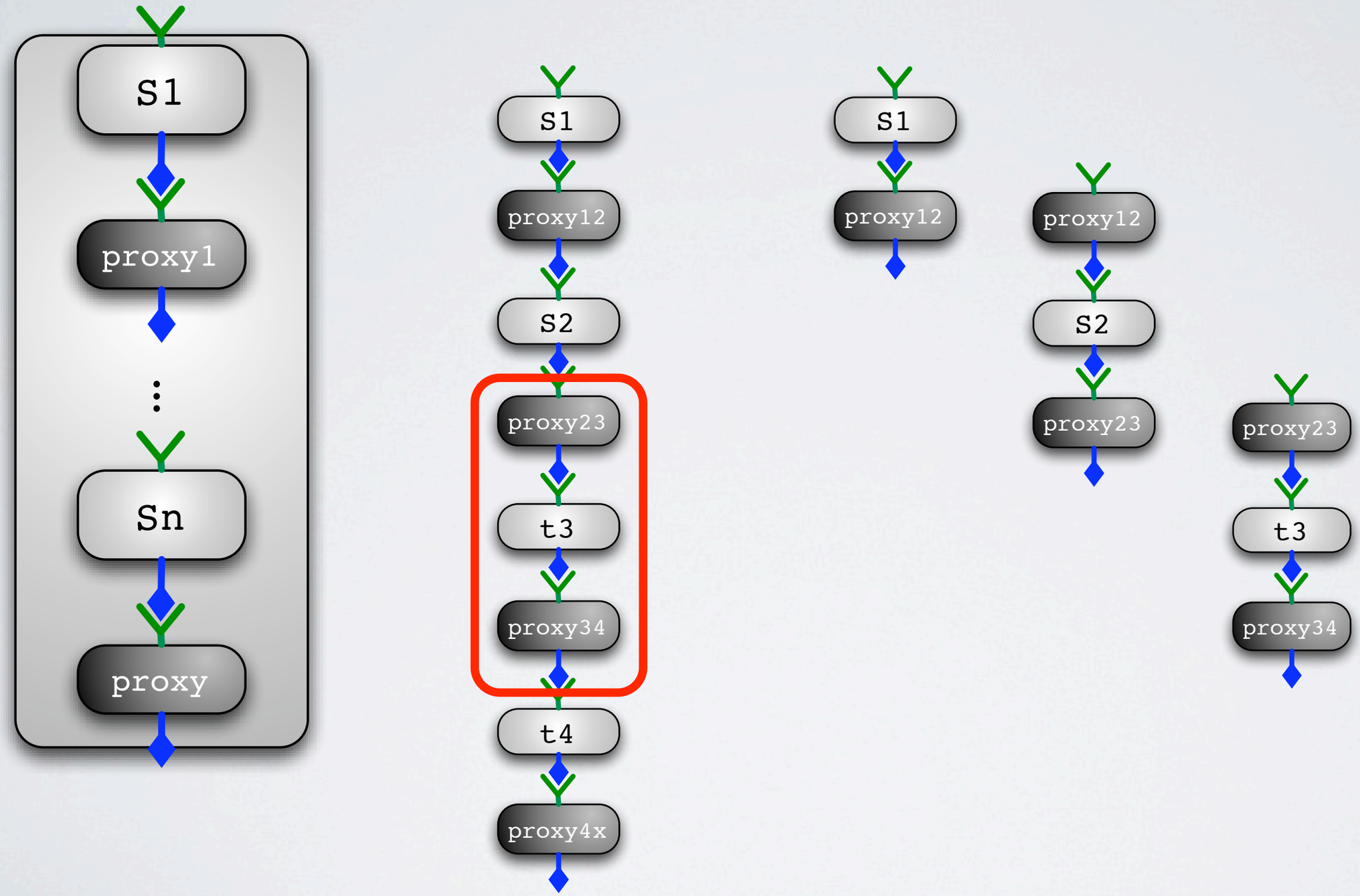
Execution Example: Sequence

time



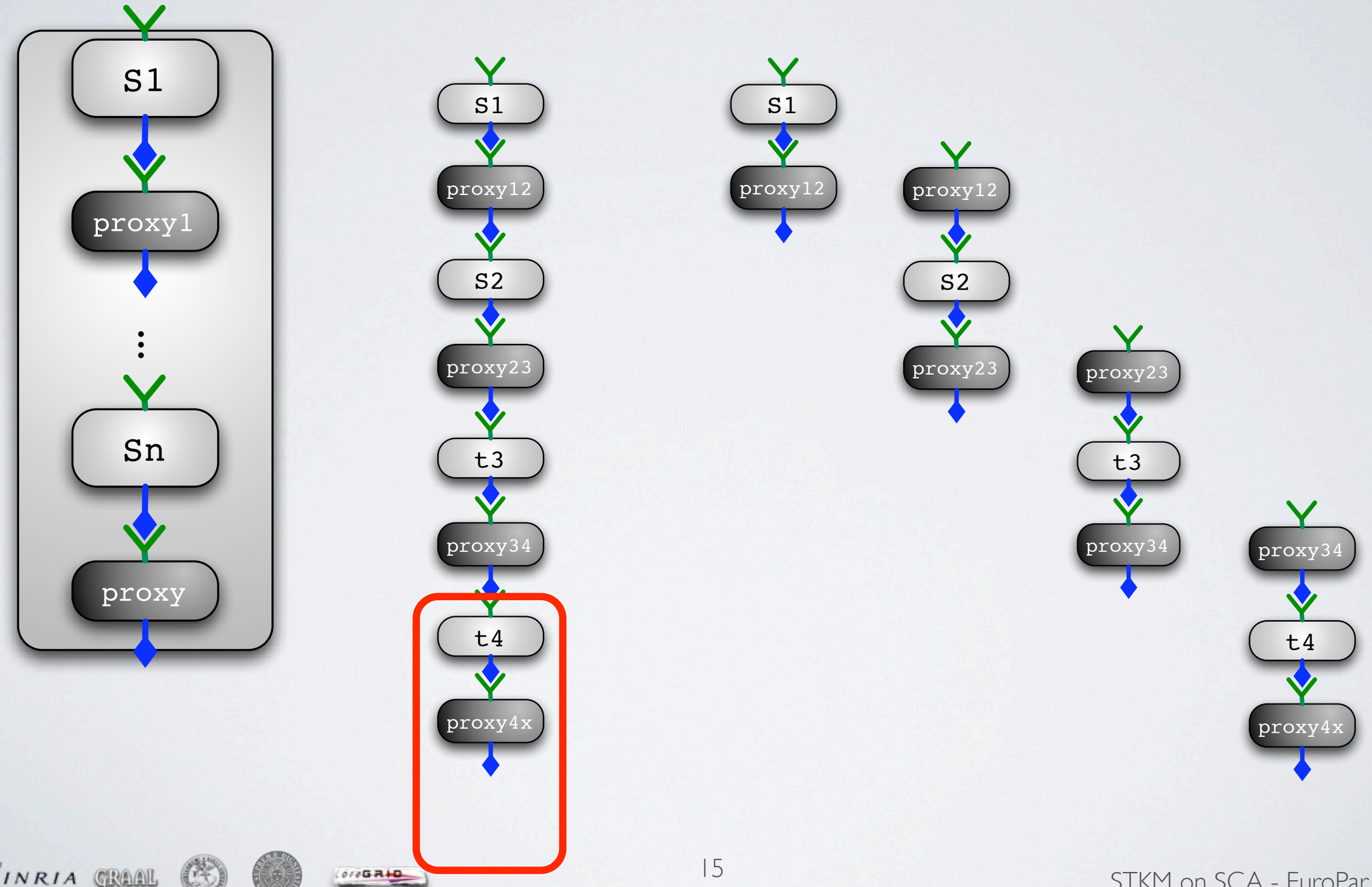
Execution Example: Sequence

time



Execution Example: Sequence

time

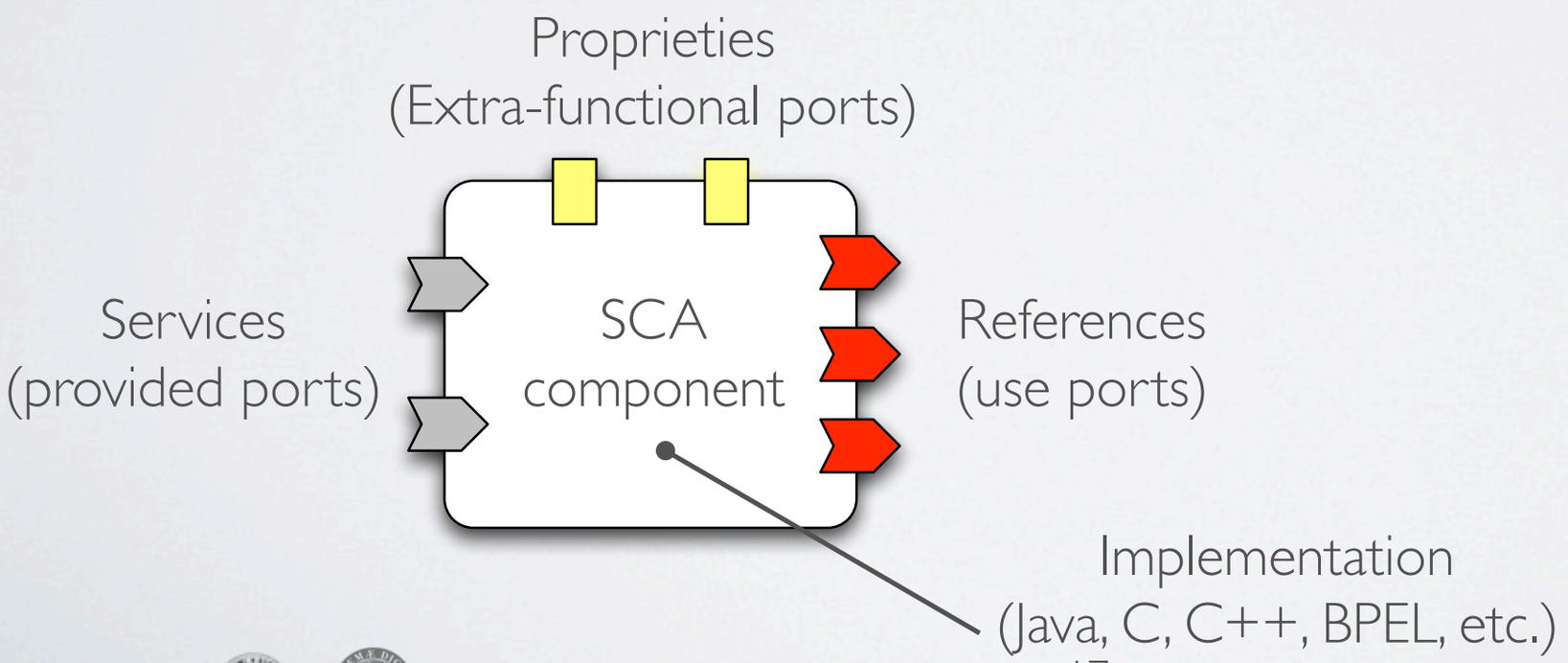


Outline

- Introduction
  - Software component models
- Existing models
  - STCM: a spatio-temporal component model
  - Skeletons based models for parallel programming
  - **STKM**: a proposal of skeletons introduction in STCM
- **A SCA-based implementation of STKM**
  - Overview of SCA
  - A projection of STKM on SCA
- Experiments
- Conclusions and future works

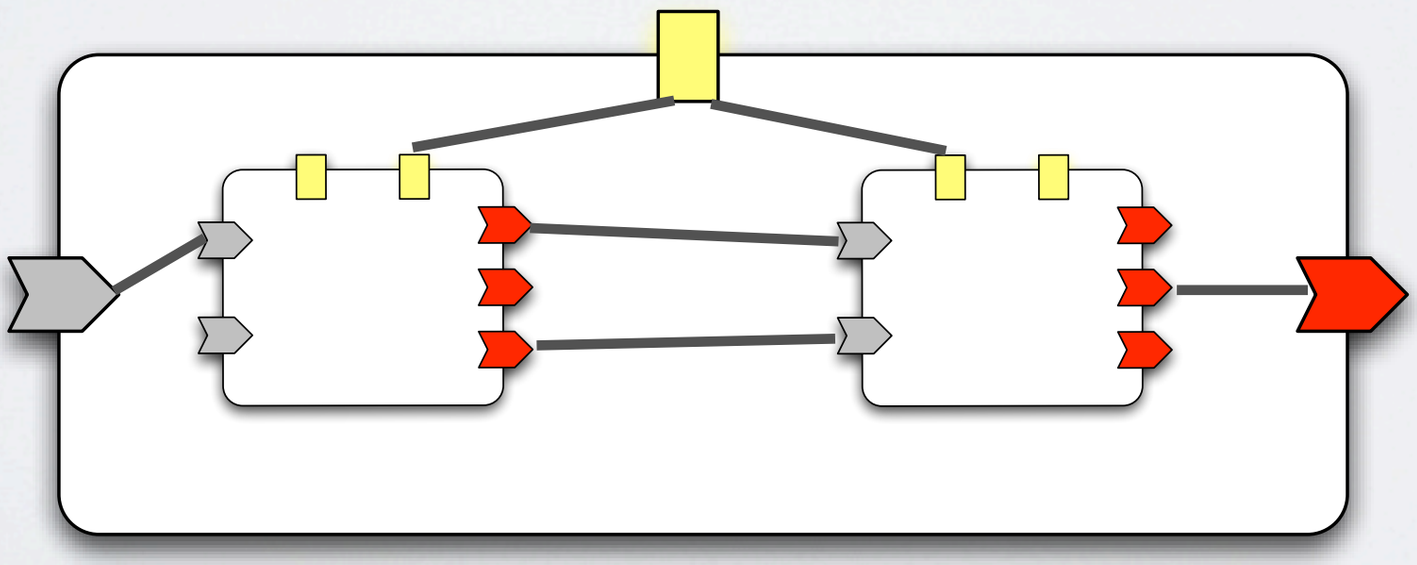
Service Component Architecture

- Open SOA collaboration 2007
- A component model for services composition
  - Independent from any technology
- Models specifications for
  - Assembly, client, component implementation, packaging and deployment



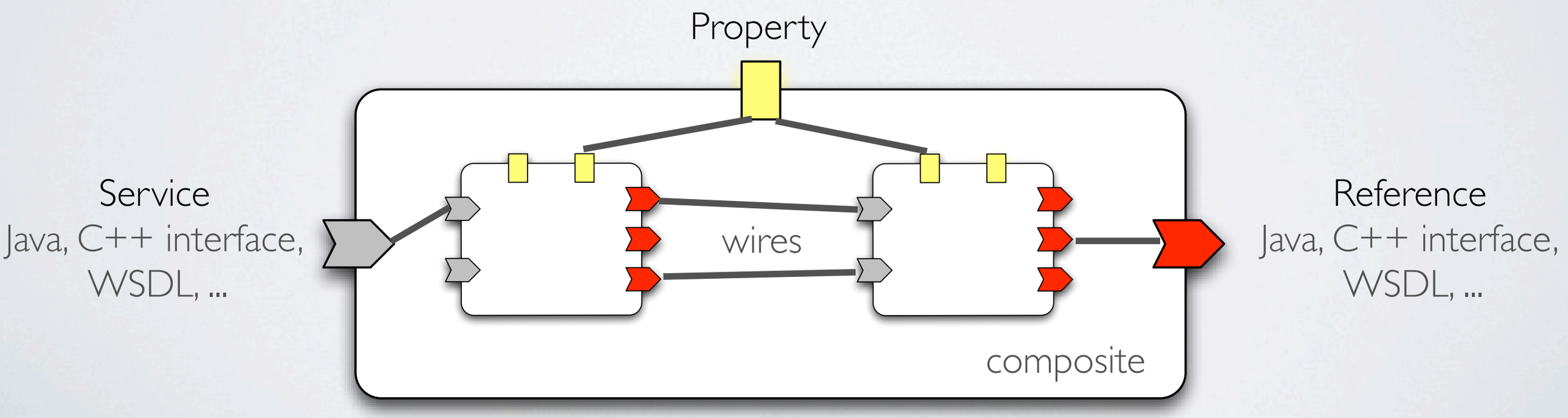
Service Component Architecture

- Open SOA collaboration 2007
- A component model for services composition
  - Independent from any technology
- Models specifications for
  - Assembly, client, component implementation, packaging and deployment

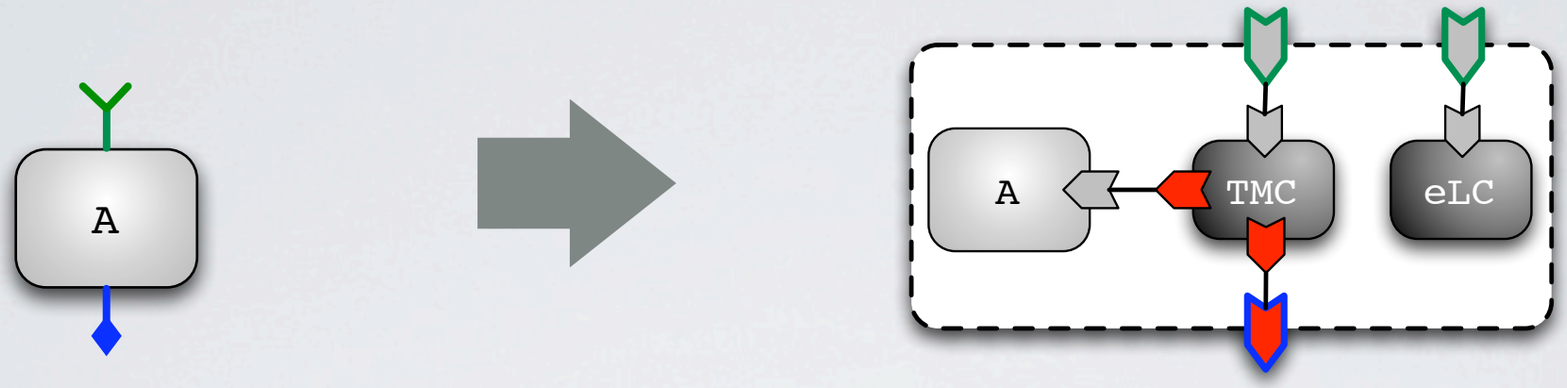


Service Component Architecture

- Open SOA collaboration 2007
- A component model for services composition
  - Independent from any technology
- Models specifications for
  - Assembly, client, component implementation, packaging and deployment

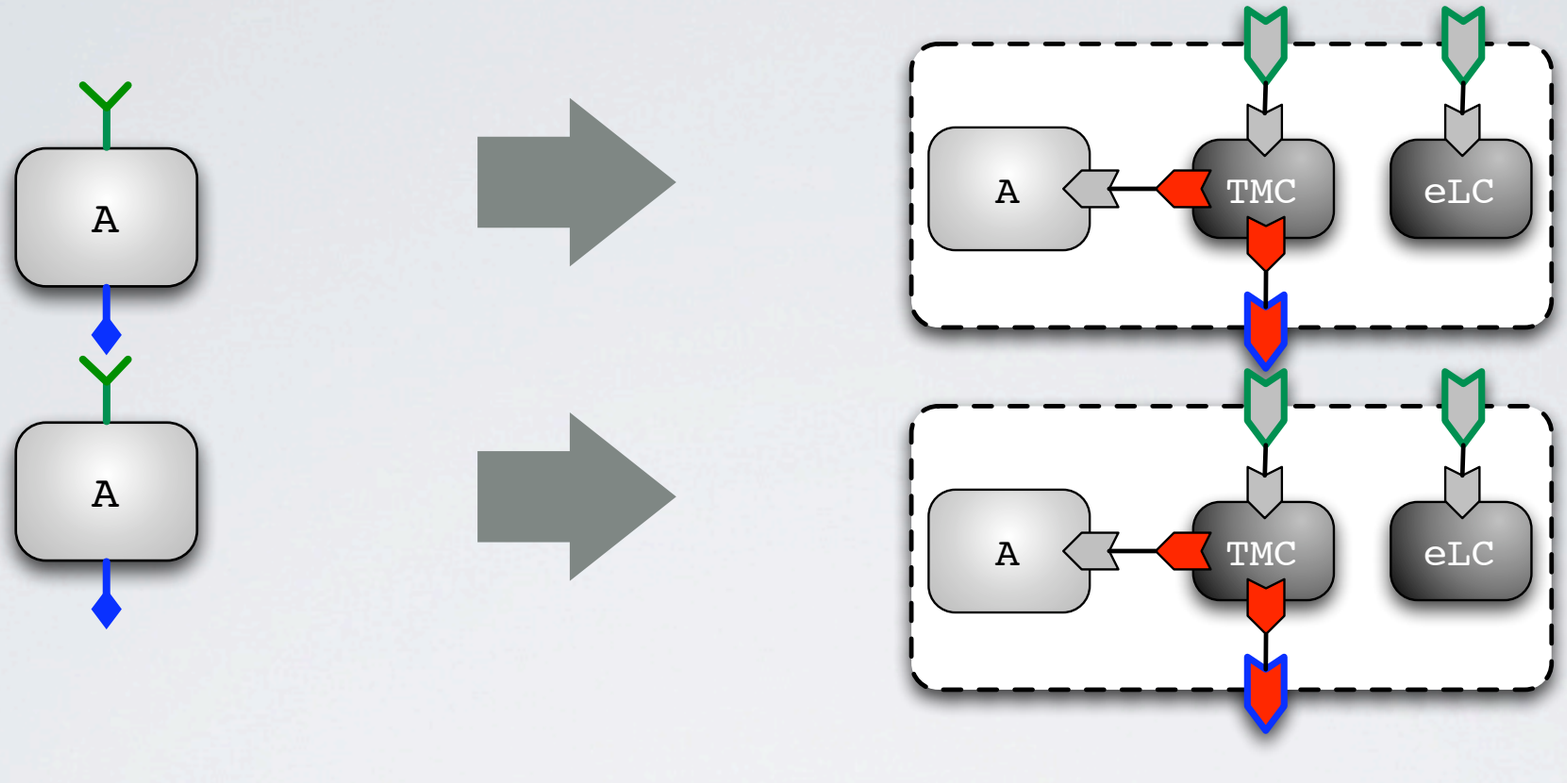


Mapping STKM onto SCA



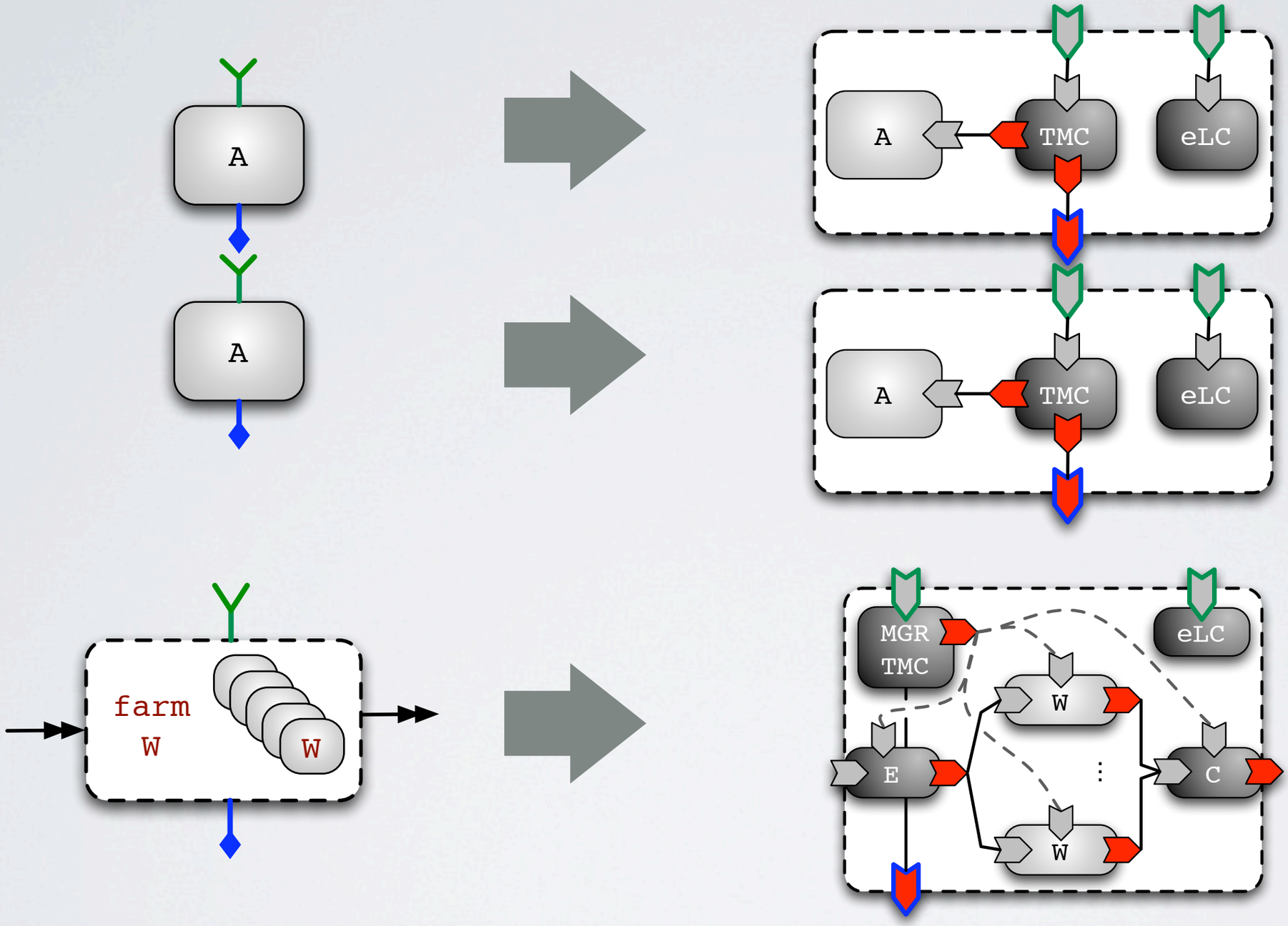
TMC = Task Management Controller
eLC = Extended Lifecycle Controller
MGR = Autonomic Manager

Mapping STKM onto SCA



TMC = Task Management Controller
eLC = Extended Lifecycle Controller
MGR = Autonomic Manager

Mapping STKM onto SCA



TMC = Task Management Controller
eLC = Extended Lifecycle Controller
MGR = Autonomic Manager

Outline

- Introduction
  - Software component models
- Existing models
  - STCM: a spatio-temporal component model
  - Skeletons based models for parallel programming
  - **STKM**: a proposal of skeletons introduction in STCM
  - A SCA-based implementation of STKM
    - Overview of SCA
    - A projection of STKM on SCA
- Experiments
- Conclusions and future works

Experimental Evaluation

- Aim
  - **Test the feasibility of the approach and measure basic overheads of SCA implementation**
- SCA implementation
  - Java 1.5 + Tuscany Java SCA version 1.2.1
  - Adaptation for dynamicity requirements
- Benchmark
  - Sequence, loop, pipeline and nested composition of pipeline and functional replication behavioral skeleton
  - Different types and sizes of tasks (time, data)
- Resources
  - Cluster of 24 Intel Pentium 3, 800MHz, 1GB RAM, 100MBit/s Ethernet

Metrics

Dynamic deployment overheads (S)

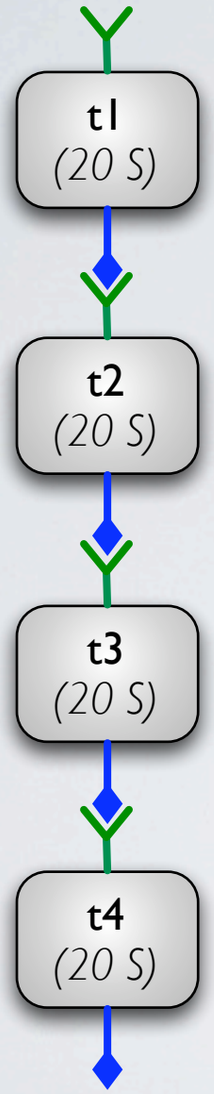
	Time(S)
Remote node launching (with ssh + common daemon library)	45.56
Programmed port connection (ad_hoc API)	3.20

Round Trip Time depending on components' placement (mS)

	Intra-node Inter-component	Inter-node Intra-host	Inter-node inter-host
Default SCA protocol	0.076	20.35	20.17
Web Service protocol	22.66	24.23	24.11

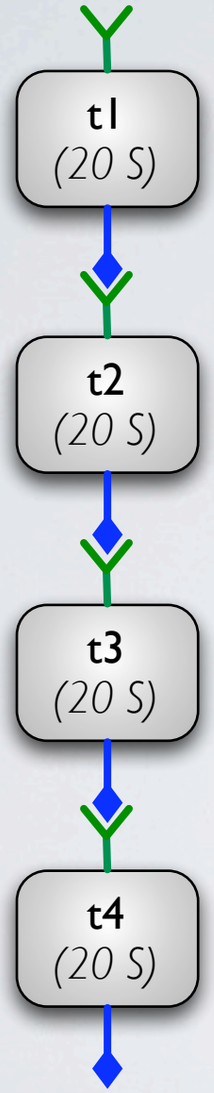
Adaptation Capability: Sequence

sequence	4 Nodes (4 PEs) requires co-allocation	8 Nodes (8 PEs)	4 Nodes (4 PEs)	2 Nodes (2 PEs)	1 Node (1 PE)

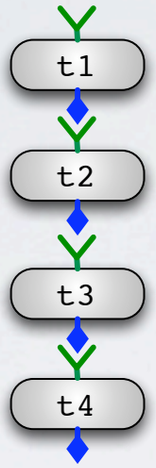


Adaptation Capability: Sequence

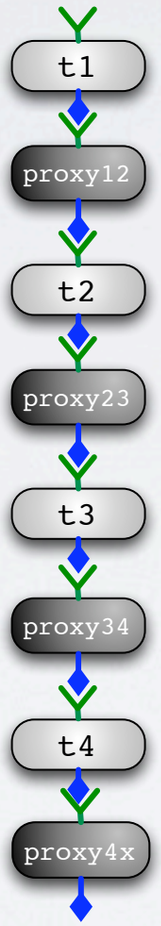
sequence



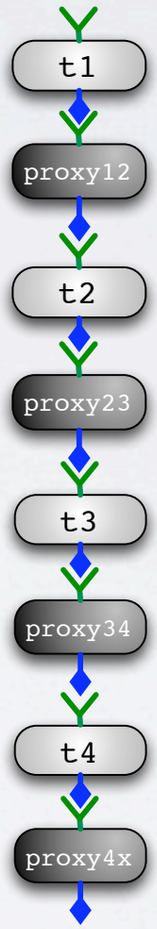
4 Nodes (4 PEs) requires co-allocation



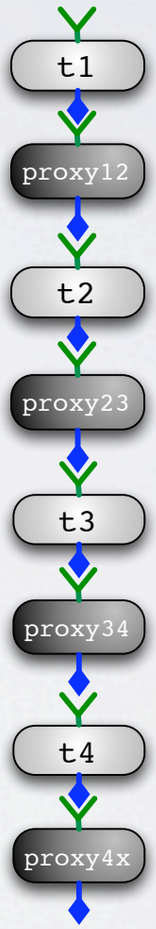
8 Nodes (8 PEs)



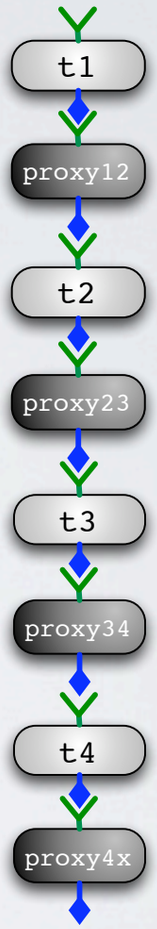
4 Nodes (4 PEs)



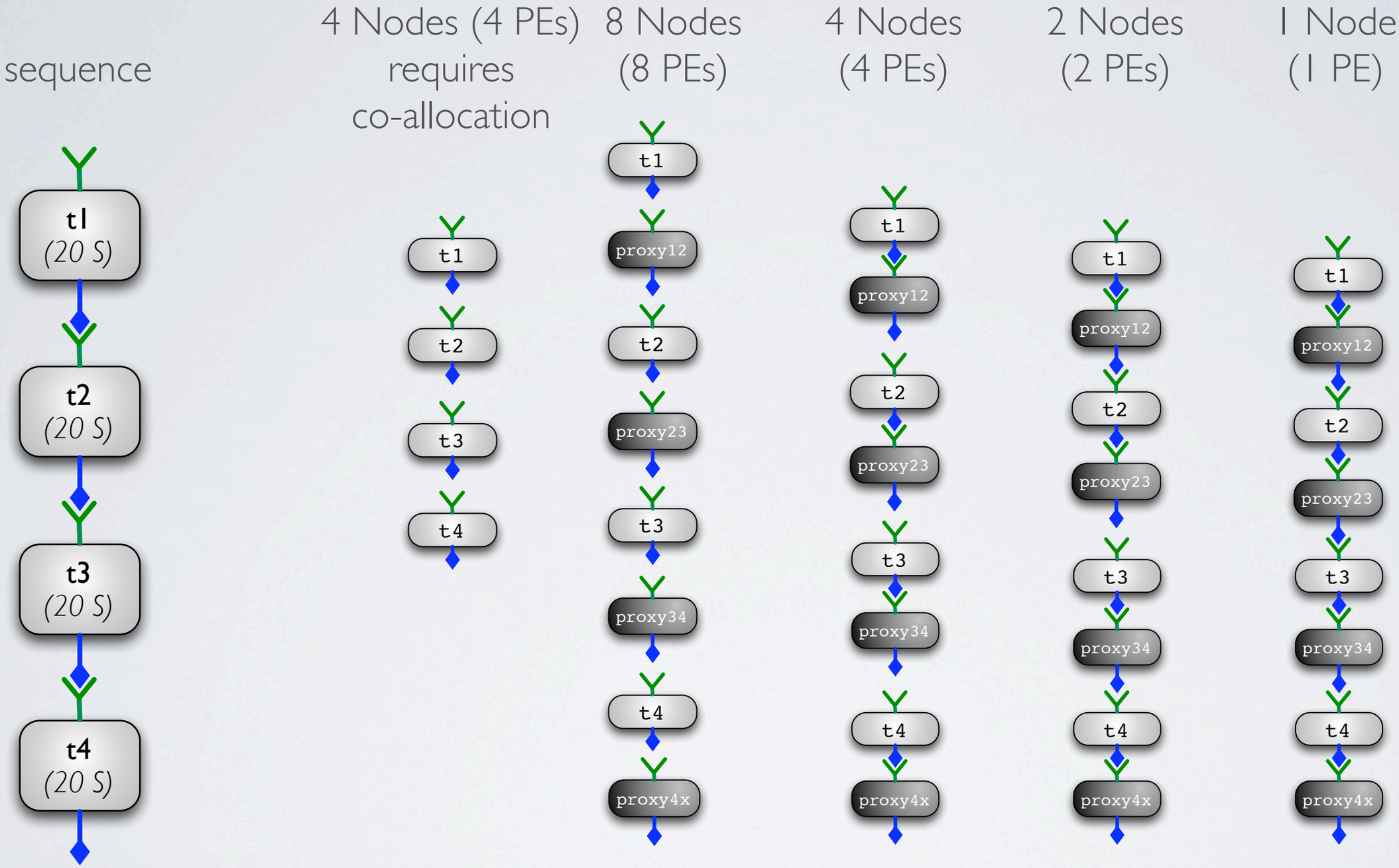
2 Nodes (2 PEs)



1 Node (1 PE)



Adaptation Capability: Sequence

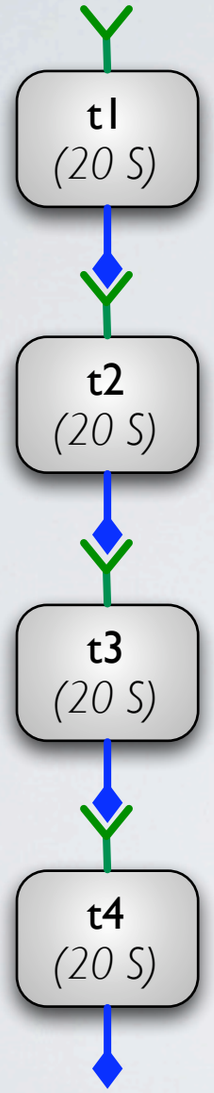


Eff (%)	51.14	27.03	20.13	28.38	28.50

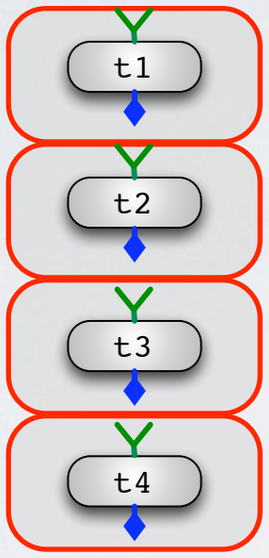
Eff = Computation time - setup overheads

Adaptation Capability: Sequence

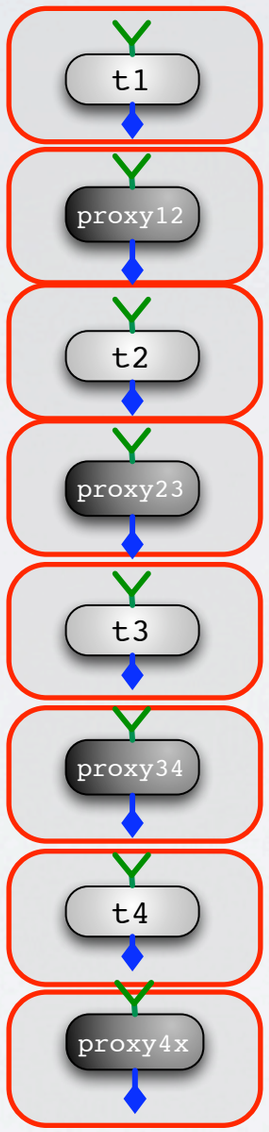
sequence



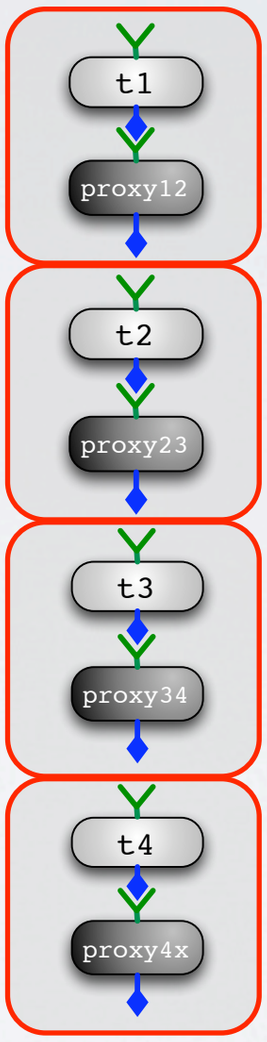
4 Nodes (4 PEs) requires co-allocation



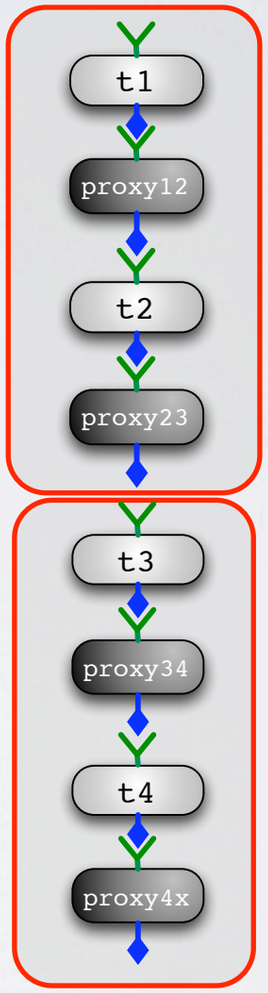
8 Nodes (8 PEs)



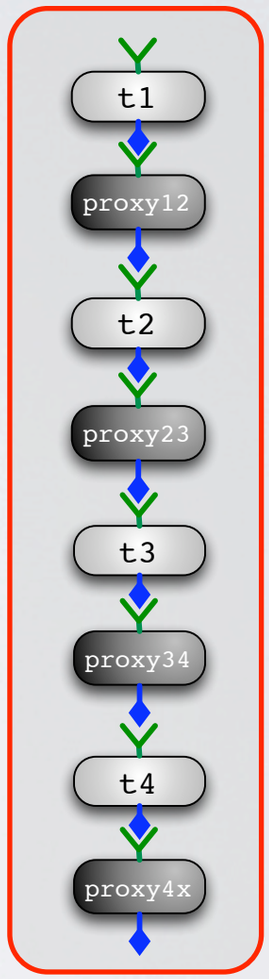
4 Nodes (4 PEs)



2 Nodes (2 PEs)



1 Node (1 PE)

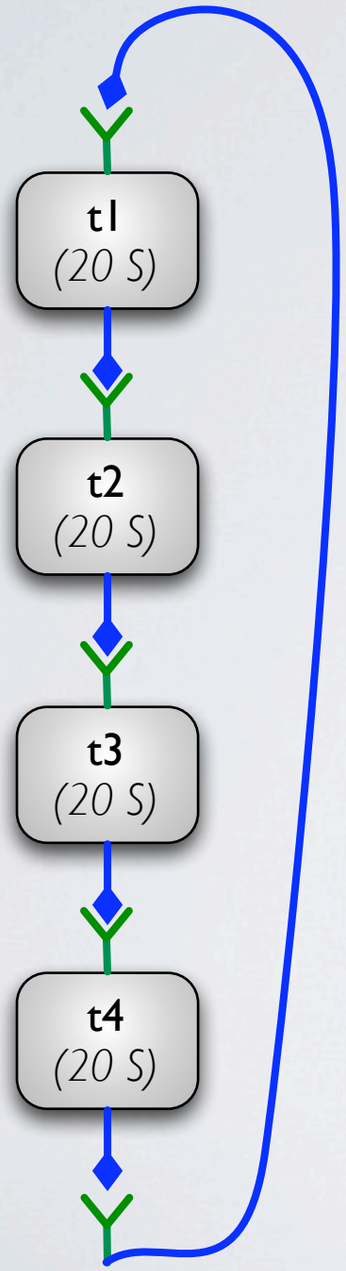


Eff (%)	51.14	27.03	20.13	28.38	28.50

Eff = Computation time - setup overheads

Idiom Recognition (loop)

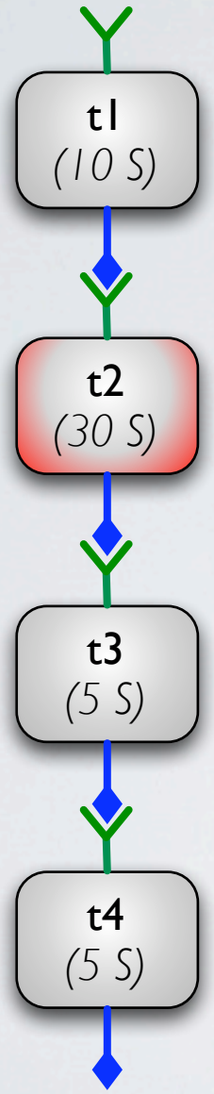
```
for (i=0; i<size; i++)
```



Efficiency = Computation time - overheads		
size	10	100
sequence with creation and destruction components are created on demand	28.40	28.40
sequence with creation component are created once on demand	56.16	90.68
pipelined all components are created at startup	76.96	95.90

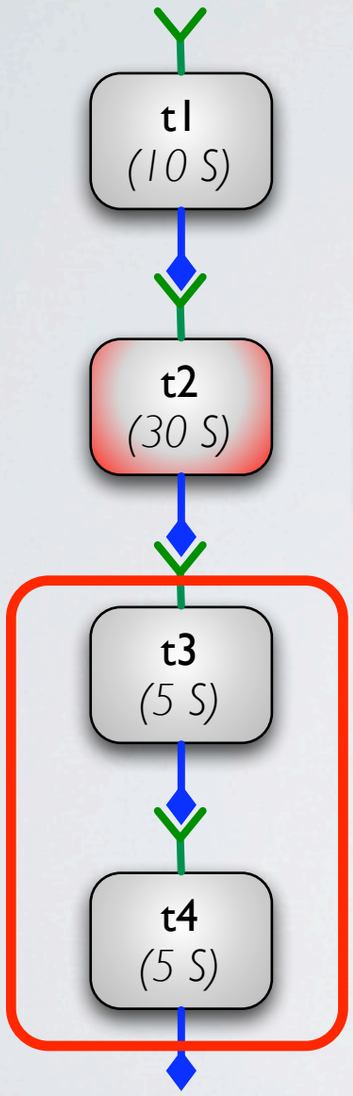
Dynamic Adaption

a b c

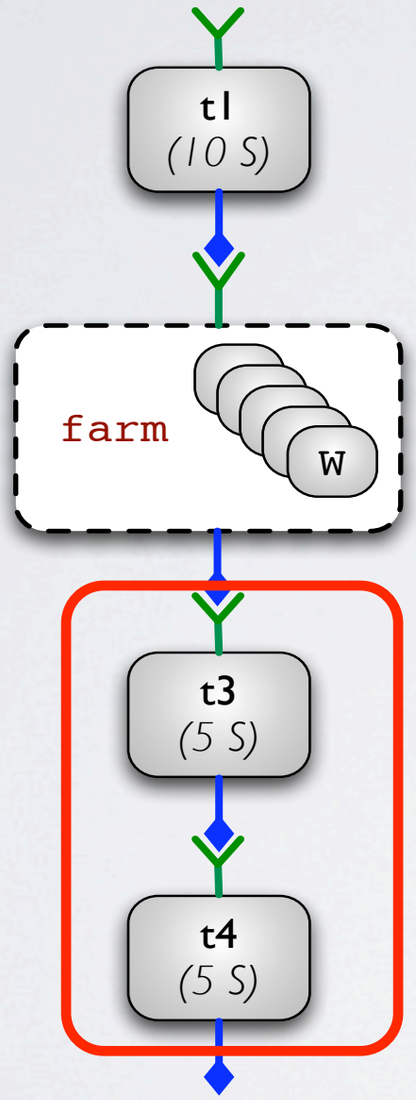


Dynamic Adaption

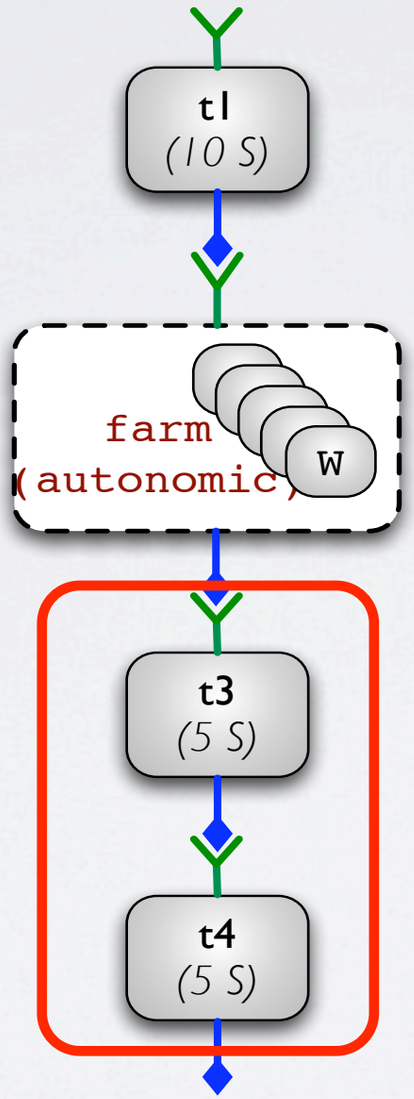
a



b

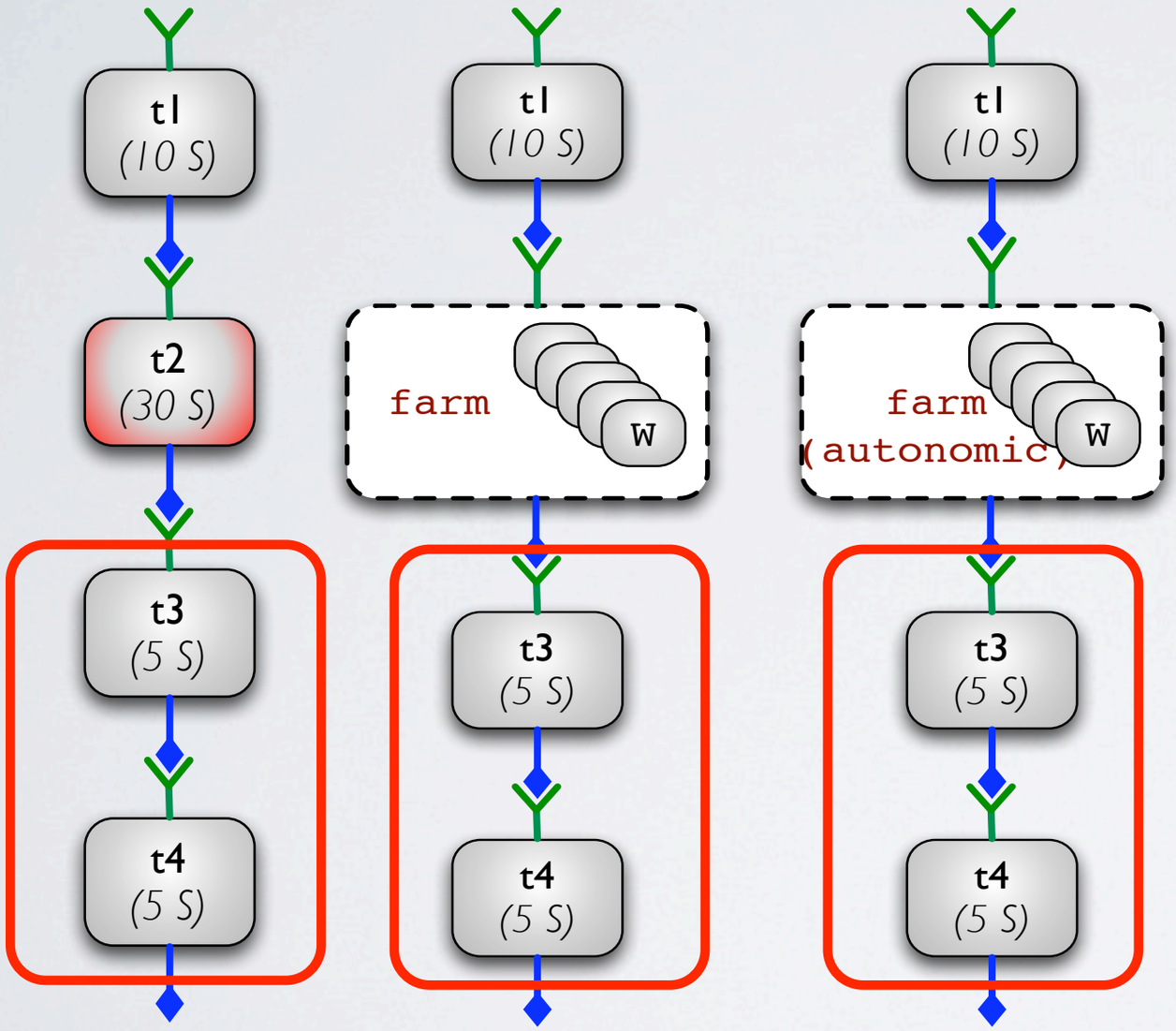


c

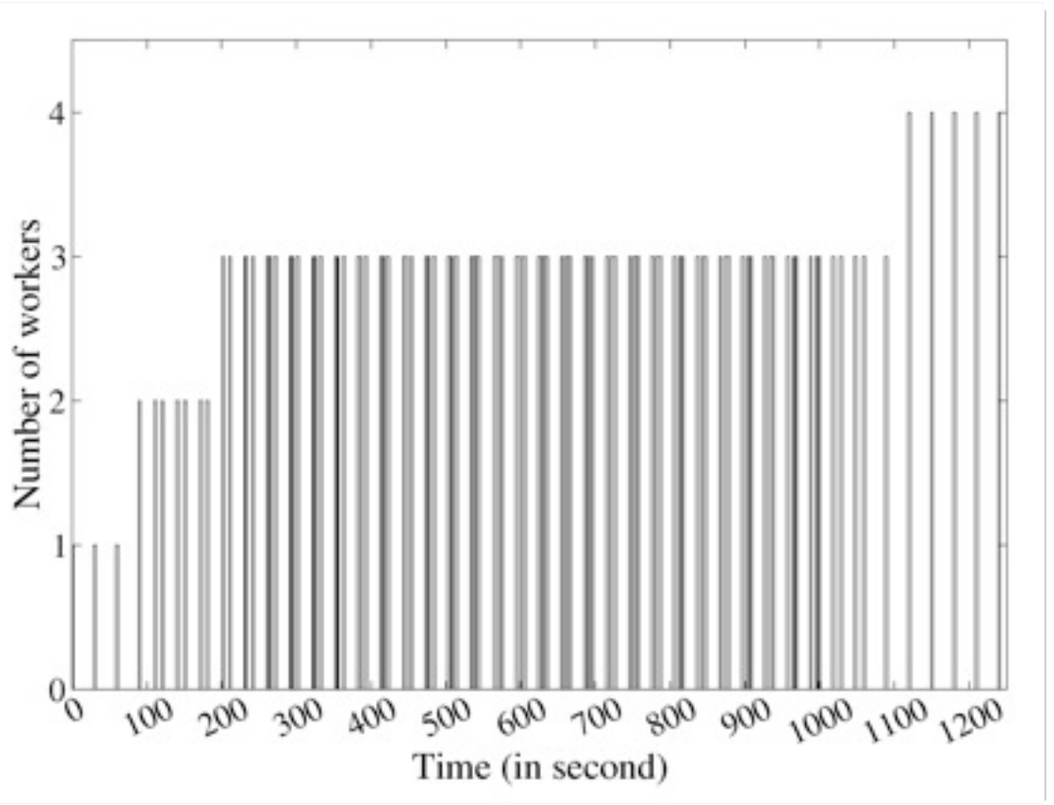


Dynamic Adaption

a b c



case: loop with size 100	time (S)	
pipeline	3105	a
pipeline with farm (3 workers)	1182	b
pipeline with autonomic farm	1403	c



Adaptation Capability: Farm

- Can dynamically change the number of workers as result of the monitored state
  - Workers can be created, destroyed, stopped, started, migrated
  - Can support stateful workers via ports supporting internal data serialization

- Widely experimented with CoreGRID GCM components
  - M. Aldinucci, S. Campa, M. Danelutto, M. Vanneschi, P. Dazzi, D. Laforenza, N. Tonellotto, and P. Kilpatrick. **Behavioural skeletons in GCM: autonomic management of grid components**. In D. E. Baz, J. Bourgeois, and F. Spies, editors, Proc. of Intl. Euromicro PDP 2008: Parallel Distributed and network-based Processing, pages 54–63, Toulouse, France, Feb. 2008. IEEE.
  - M. Aldinucci, M. Danelutto, and P. Kilpatrick. **Autonomic management of non-functional concerns in distributed and parallel application programming**. In Proc. of Intl. Parallel & Distributed Processing Symposium (IPDPS), pages 1–12, Rome, Italy, May 2009. IEEE.

Conclusions

- A combination of component models, workflows and skeletons
- Previous works
  - STCM: merging component models with workflows
  - Skeleton models
  - STKM proposal (theoretical study)
- Contributions
  - STKM prototype on SCA
  - Performance evaluation
- Simplicity of design and adaptation capabilities

Perspectives

- Generic skeletons constructs for easy extension with new skeletons
  - Ongoing work (PhD in the GRAAL project-team)
- Framework implementation for automatic generation of assemblies at execution
  - Model driving engineering
  - Choice and decisions techniques
- Applications implementation
  - Using more recent and advanced frameworks

Thank you

Marco Aldinucci
Computer Science Dept.
University of Torino - Italy

Marco Danelutto
Computer Science Dept.
University of Pisa - Italy

Hinde-Lilia Bouziane & Christian Pérez
Project-team GRAAL
INRIA Rhône-Alpes - ENS Lyon - France

Christian Pérez
Project-team GRAAL
INRIA Rhône-Alpes - ENS Lyon - France