Emergent Software Systems

Summer School

Barry Porter & Roberto Rodrigues Filho
School of Computing and Communications
Lancaster University

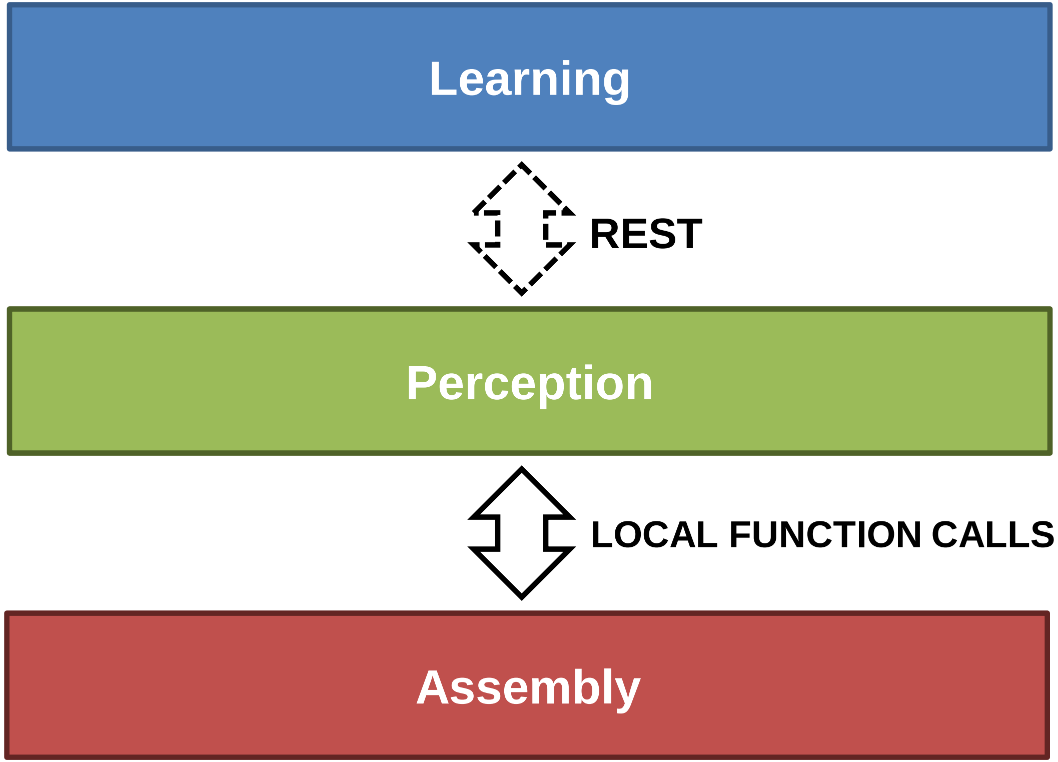
Funded by The Royal Society Newton Fund

Perception, Assembly and Learning (PAL)

- **Framework:**
  - Assembly Module;
  - Perception Module;
  - Learning Module.

Framework

Perception, Assembly and Learning (PAL)



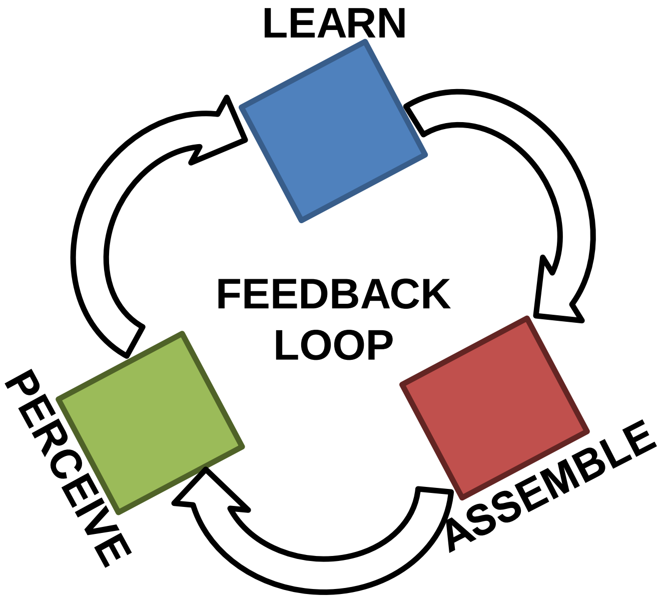
Learning

REST

Perception

LOCAL FUNCTION CALLS

Assembly



LEARN

FEEDBACK LOOP

PERCEIVE

ASSEMBLE

Assembly

The Assembly module is responsible to implement the entire assembly process: given a root component, it extracts the required interfaces from the root component object file, searches for components that provide the required interfaces, and apply this process recursively to all discovered components until it has every possible composition for the system’s local architecture. The Assembly module provides a list of methods that make the process of adaptation, adding new components, removing components, assembling local systems transparent.

Assembly

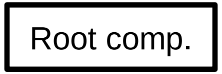
- **Assembly API**
  - *void setMain(char path[], char args[])*
  - *char[] getConfig()*
  - *String[] getAllConfigs()*
  - *bool setConfig()*
  - *void addComp(String comPaths[])*
  - *void removeComp(String compPaths[])*

Assembly

- **Assembly Process (setMain())**

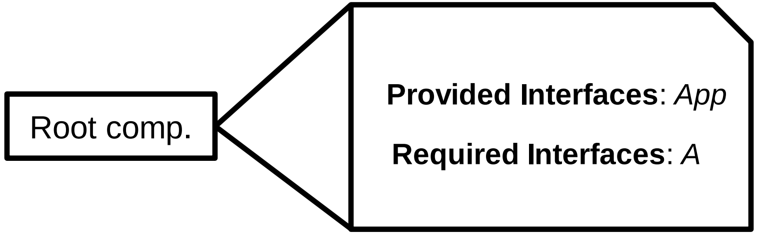
Assembly

- **Assembly Process (setMain())**



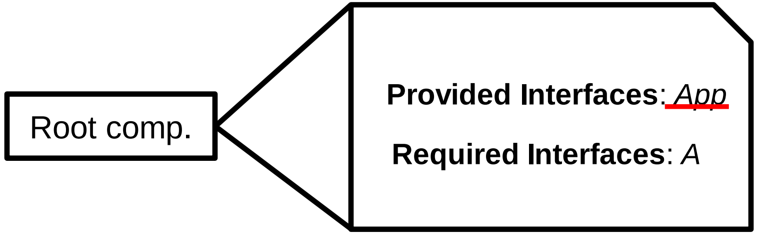
Assembly

- **Assembly Process (setMain())**



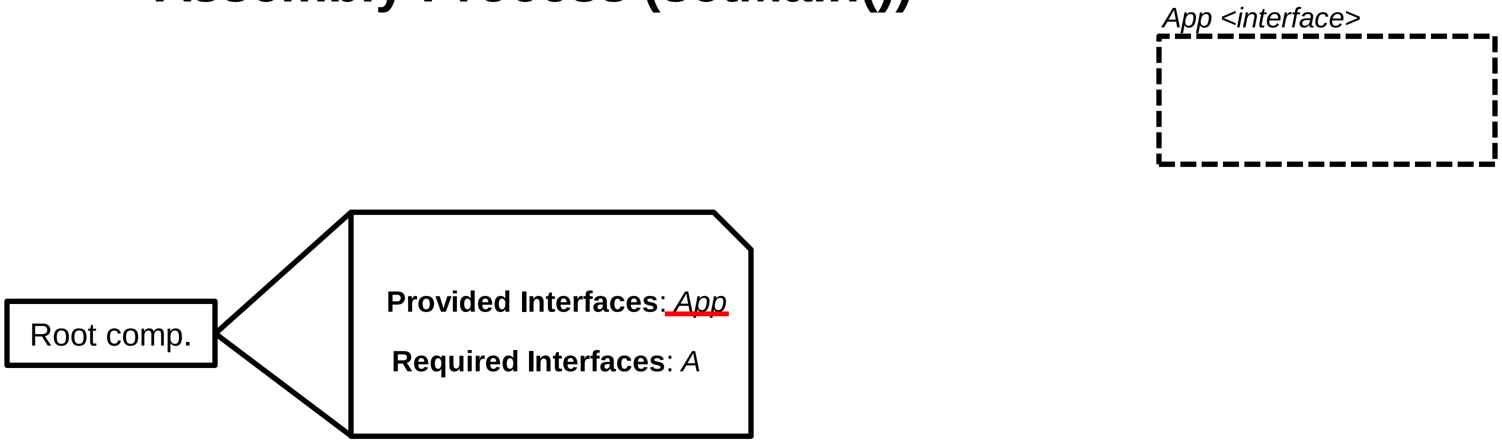
Assembly

- **Assembly Process (setMain())**



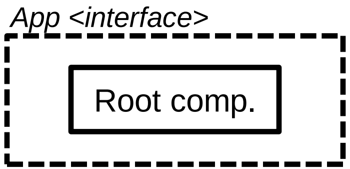
Assembly

- **Assembly Process (setMain())**



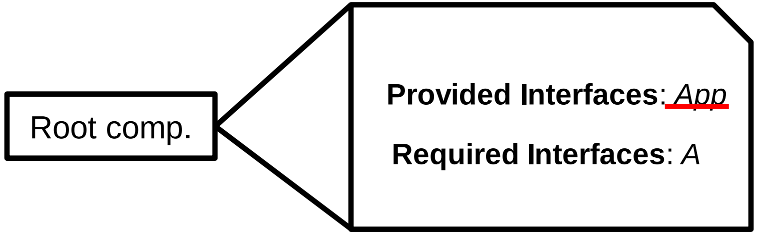
Assembly

- **Assembly Process (setMain())**



App <interface>

Root comp.



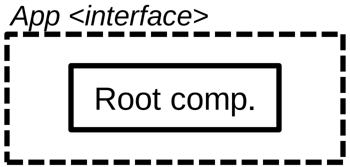
Root comp.

Provided Interfaces: *App*

Required Interfaces: *A*

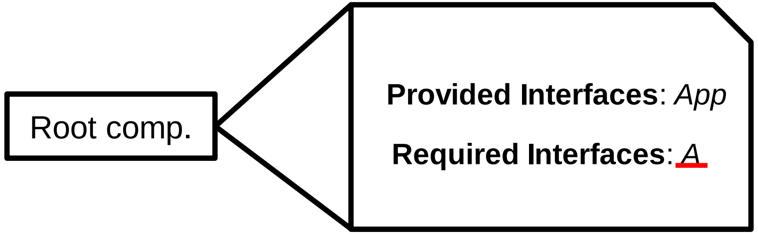
Assembly

- **Assembly Process (setMain())**



App <interface>

Root comp.



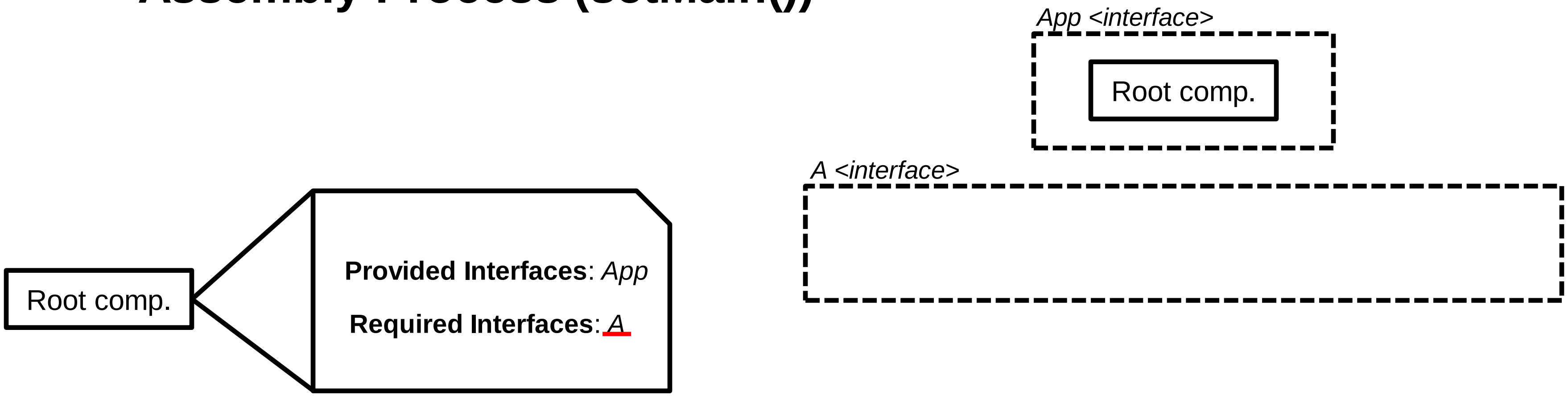
Root comp.

Provided Interfaces: *App*

Required Interfaces: *A*

Assembly

- **Assembly Process (setMain())**



Root comp.

Provided Interfaces: *App*

Required Interfaces: *A*

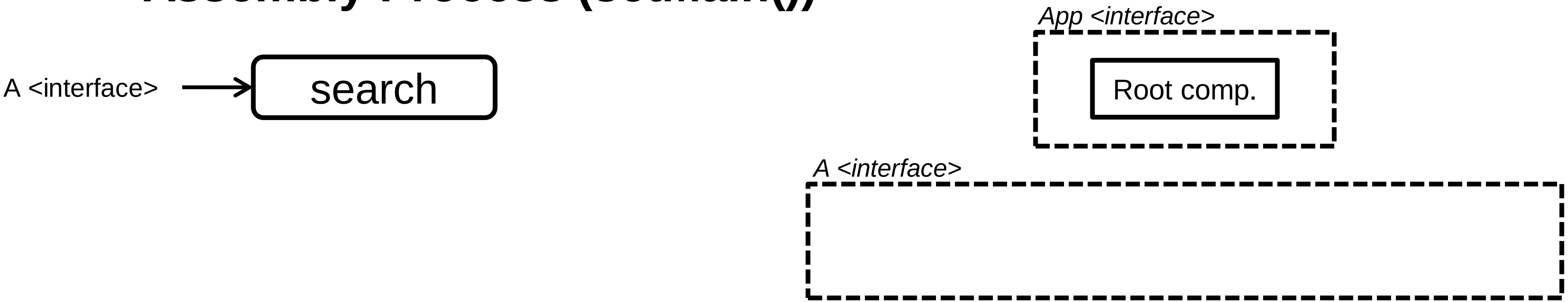
App <interface>

Root comp.

A <interface>

Assembly

- **Assembly Process (setMain())**



A <interface>

search

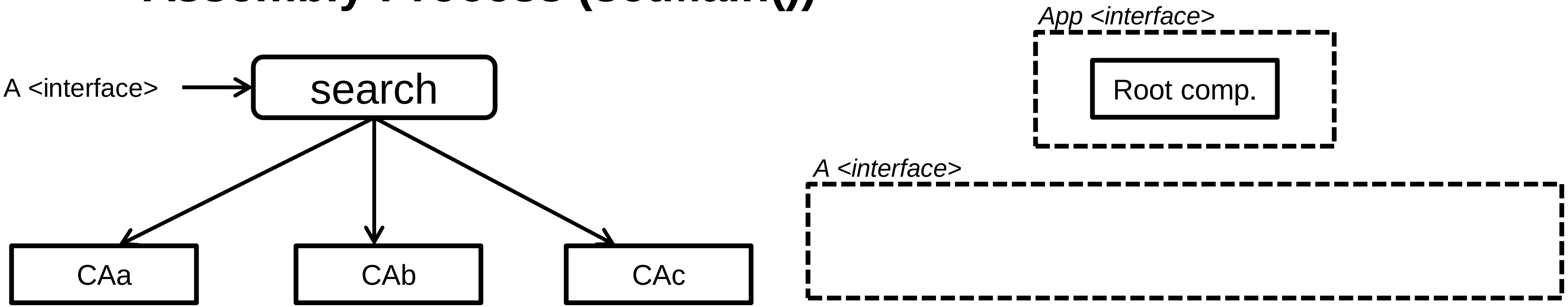
App <interface>

Root comp.

A <interface>

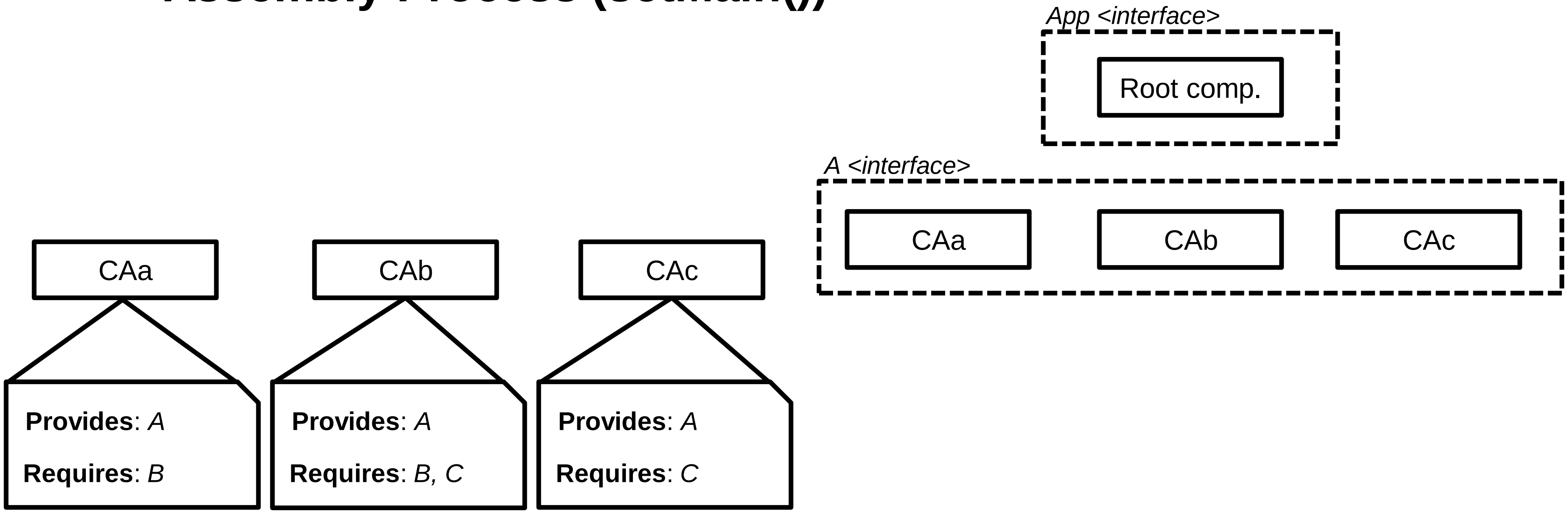
Assembly

- **Assembly Process (setMain())**



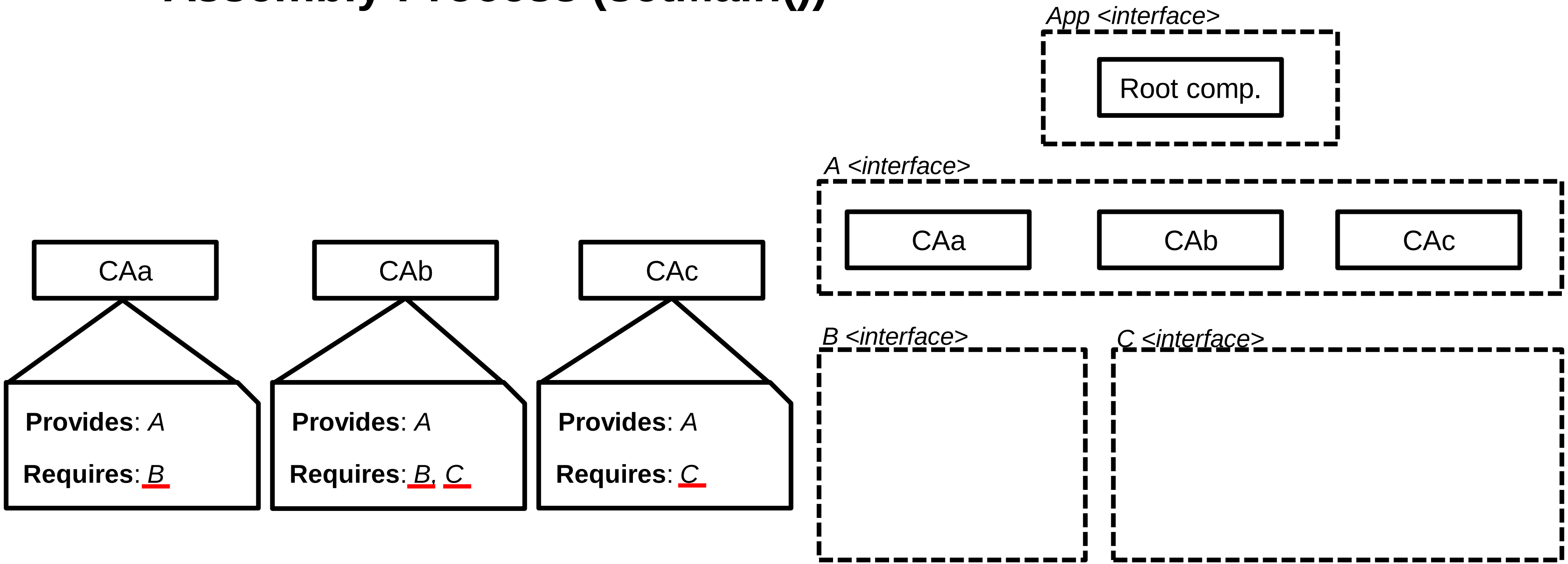
Assembly

- **Assembly Process (setMain())**



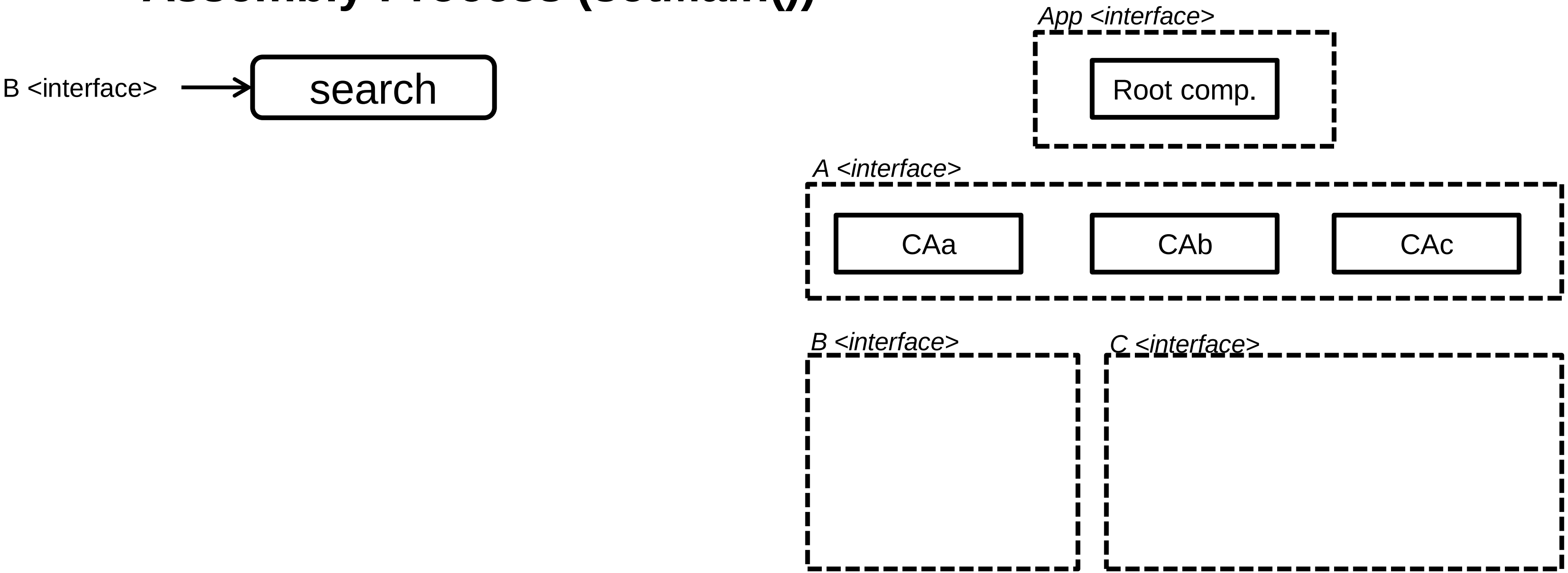
Assembly

- **Assembly Process (setMain())**



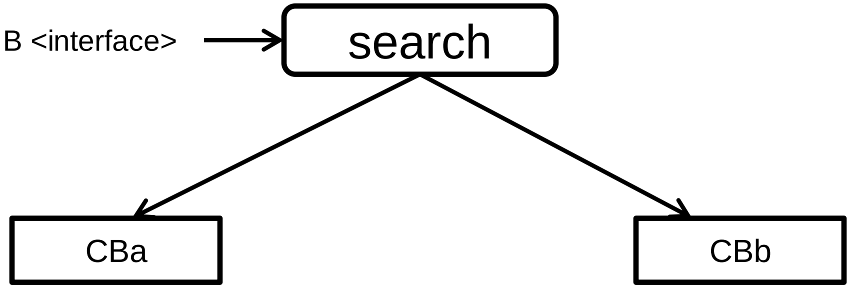
Assembly

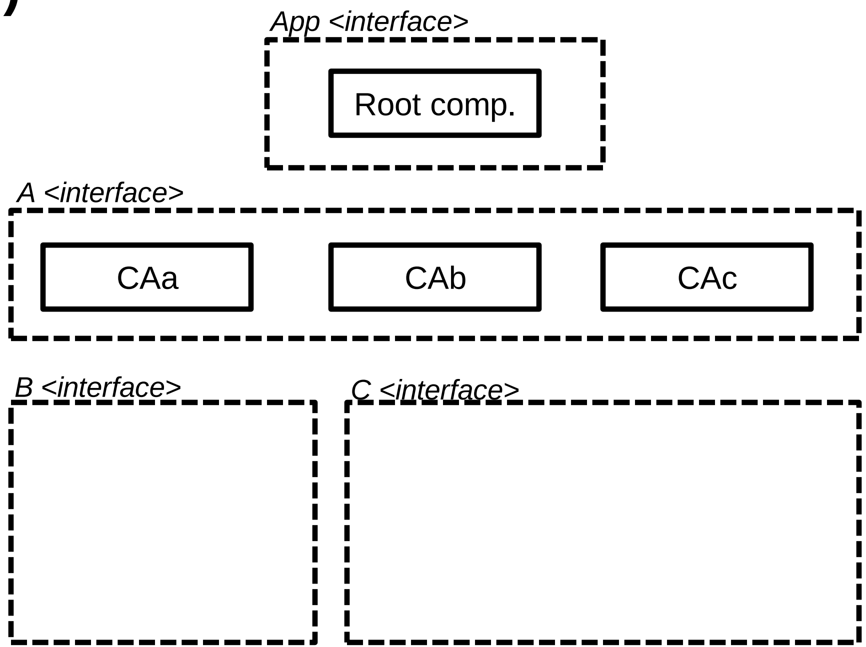
- **Assembly Process (setMain())**



Assembly

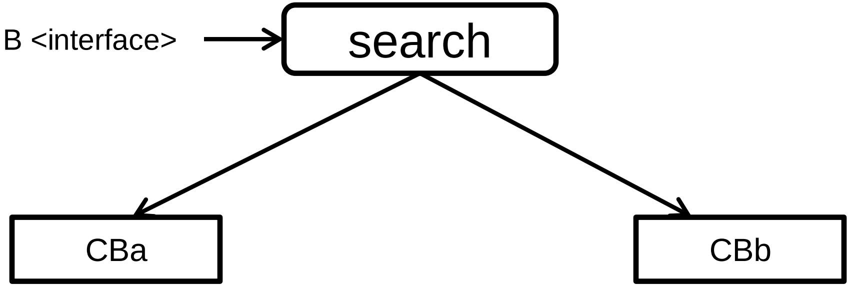
- **Assembly Process (setMain())**

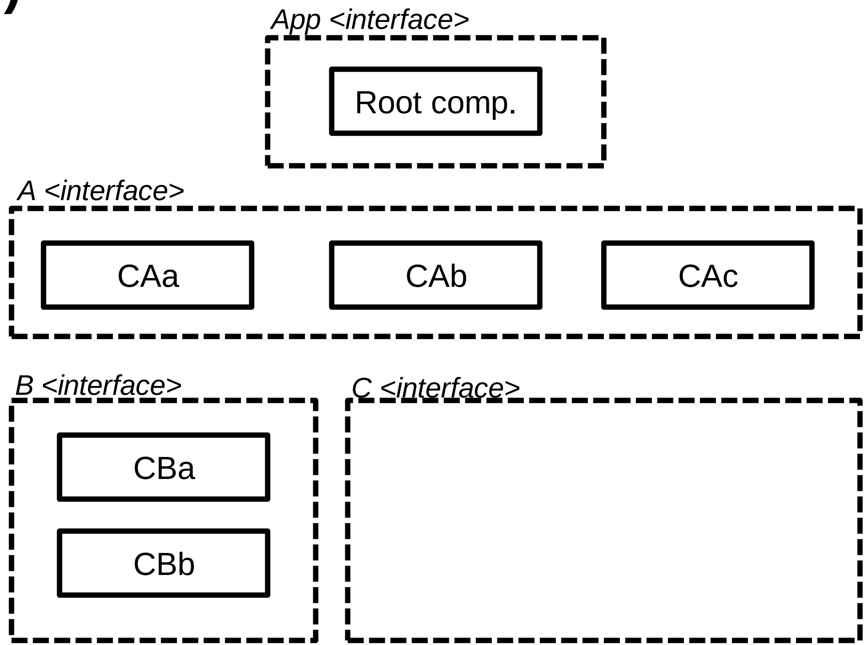




Assembly

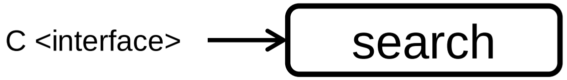
- **Assembly Process (setMain())**

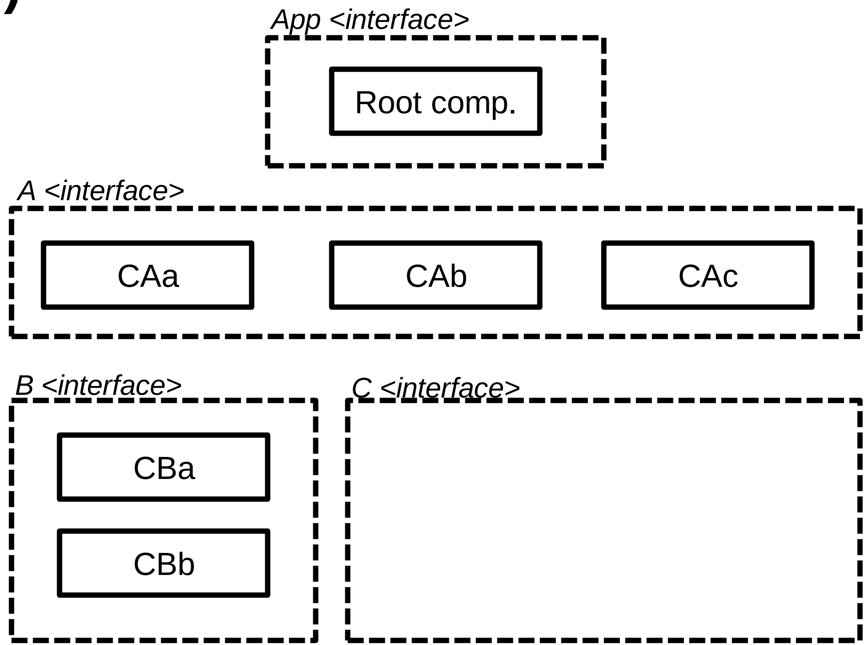




Assembly

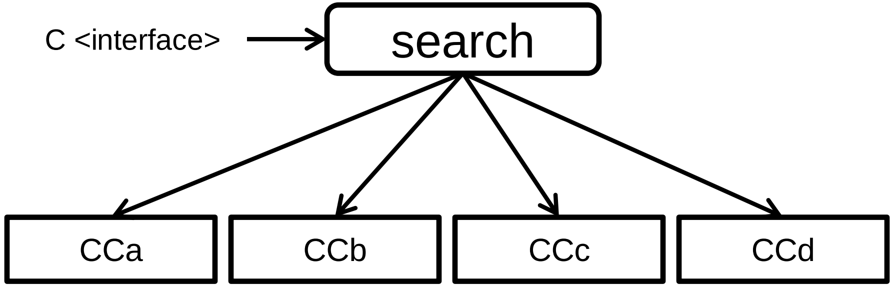
- **Assembly Process (setMain())**

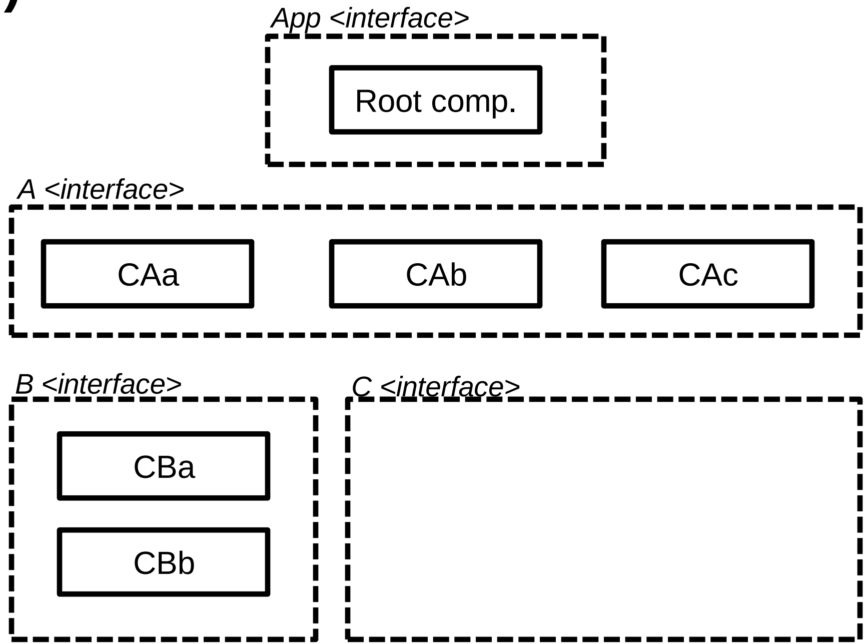




Assembly

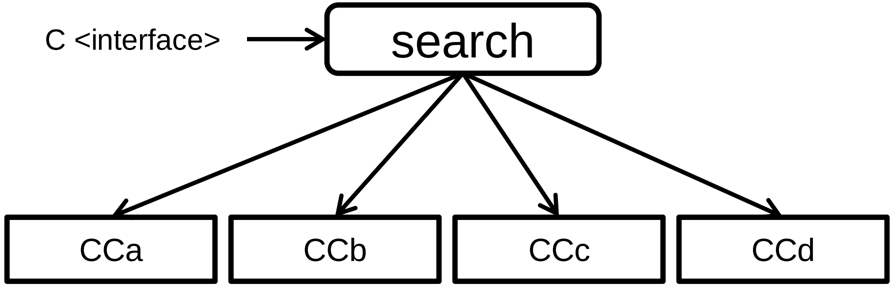
- **Assembly Process (setMain())**

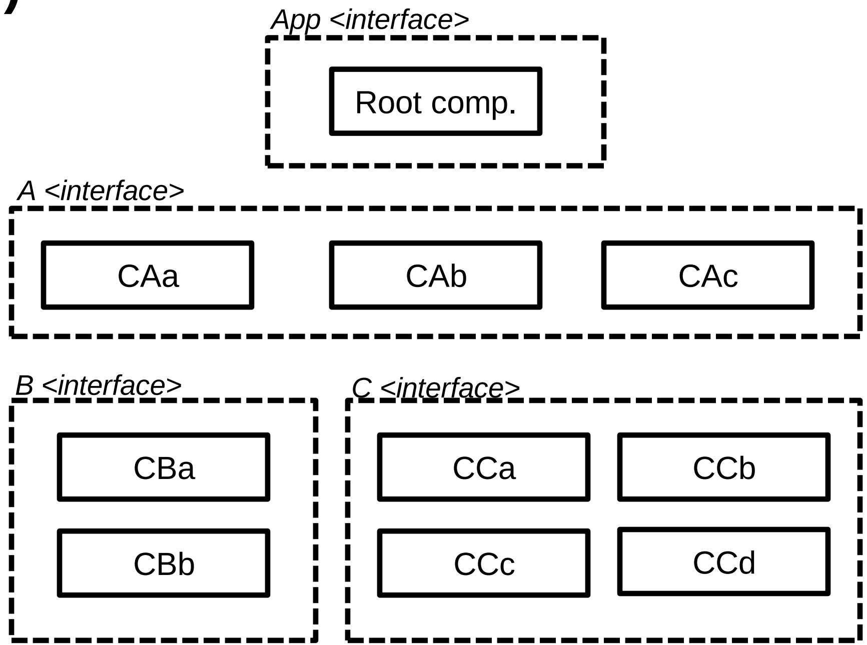




Assembly

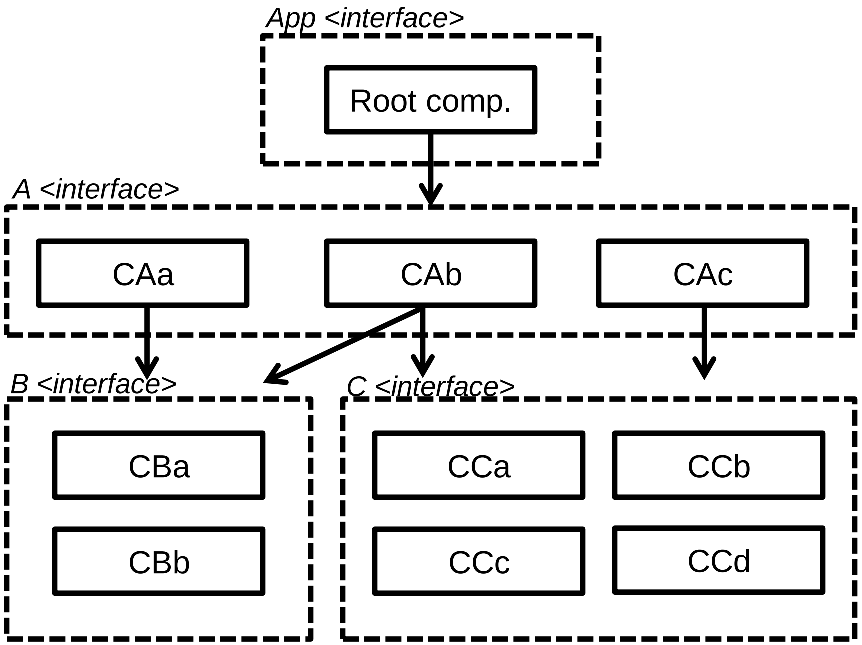
- **Assembly Process (setMain())**





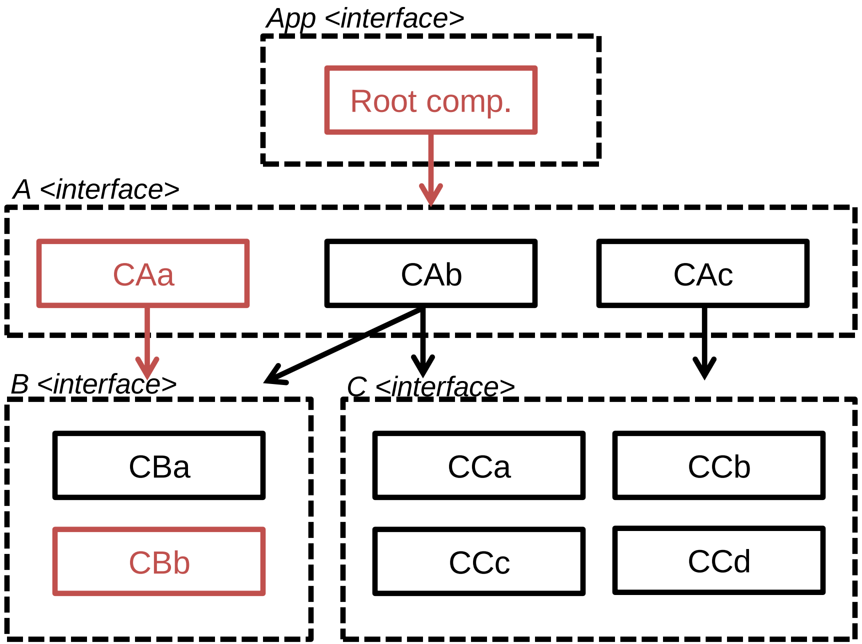
Assembly

- **Assembly Process (setMain())**



Assembly

- **Assembly Process (setMain())**

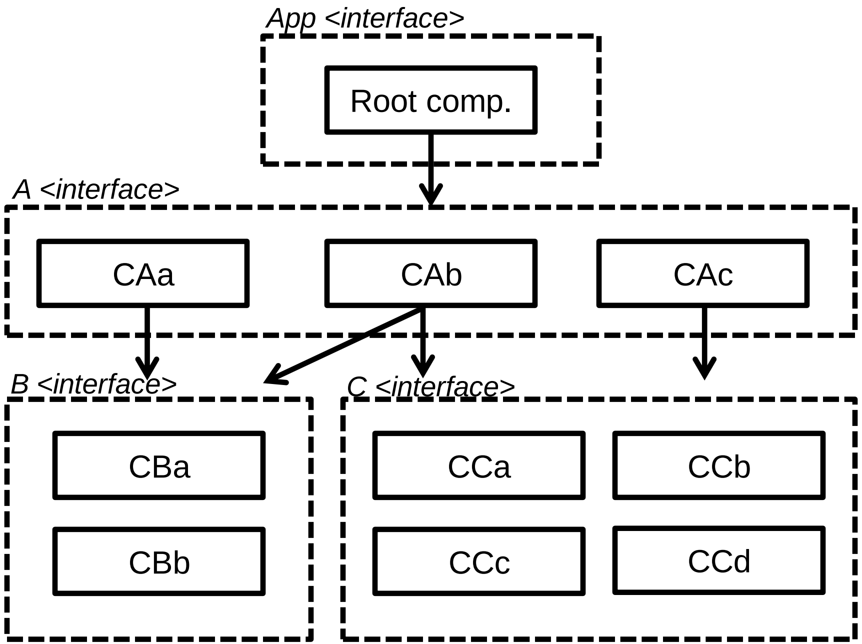


Assembly

- **Architectural Description (getAllConfigs(), getConfig())**

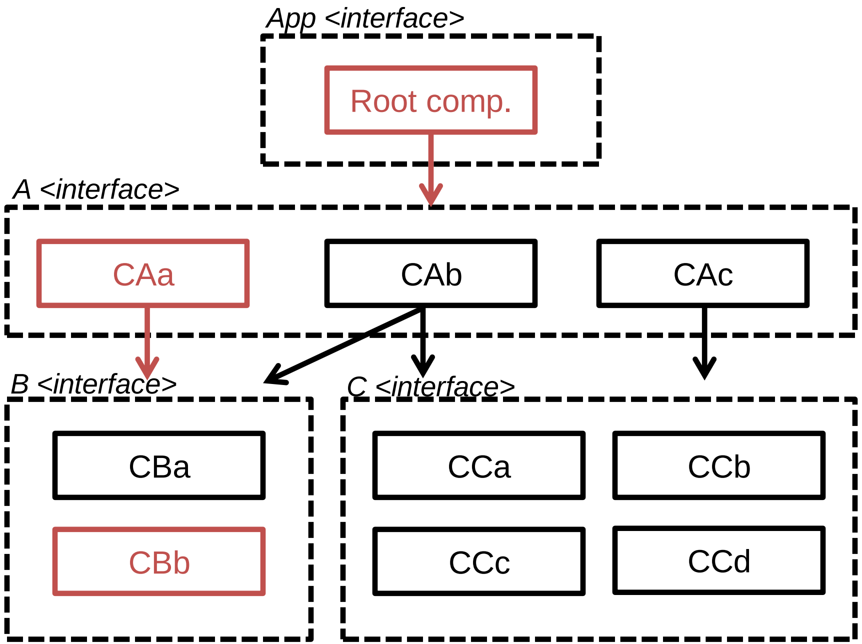
Assembly

- **Architectural Description (getAllConfigs(), getConfig())**



Assembly

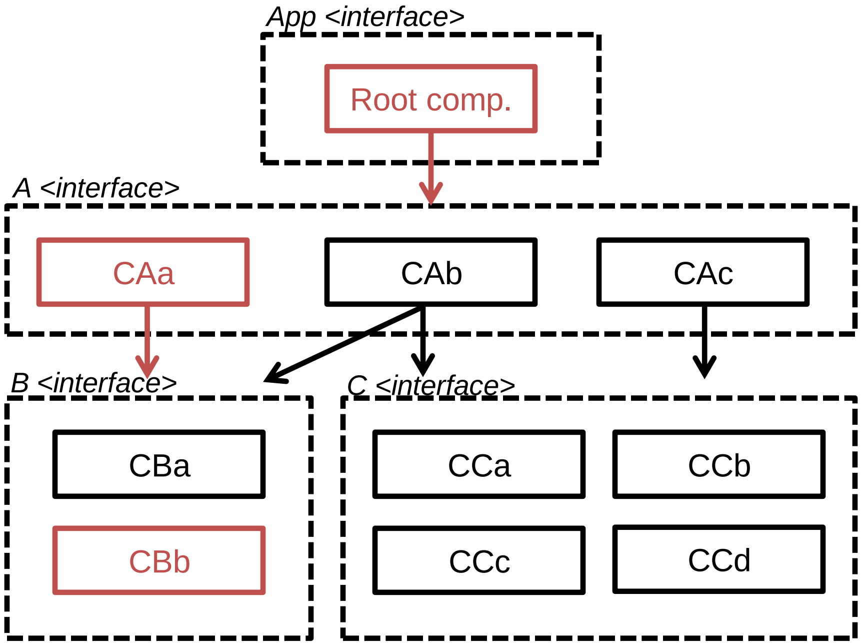
- **Architectural Description (getAllConfigs(), getConfig())**



Assembly

- **Architectural Description (getAllConfigs(), getConfig())**

Architectural descriptions are full descriptions of the architecture, and it works as an unique identification code for the architecture.

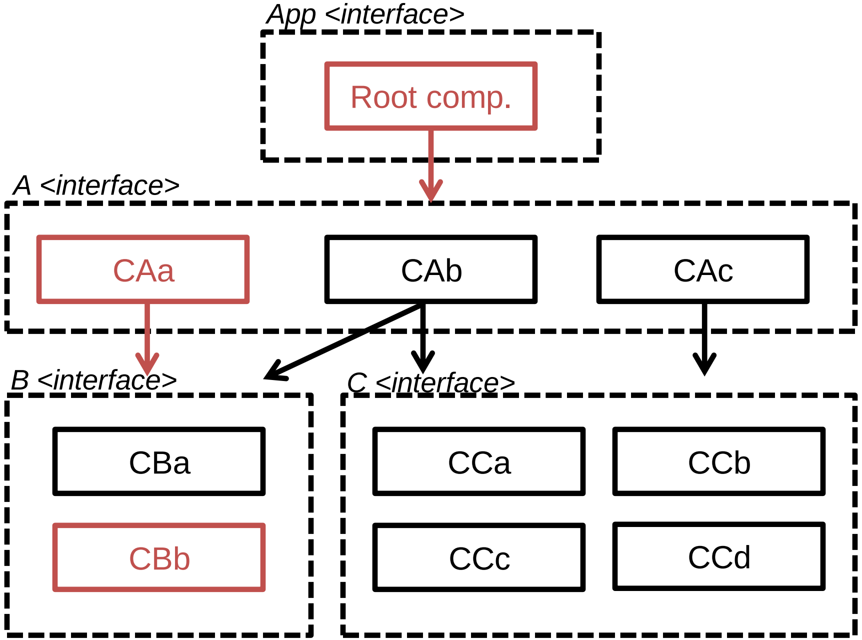


Assembly

- **Architectural Description (getAllConfigs(), getConfig())**

Architectural descriptions are full descriptions of the architecture, and it works as an unique identification code for the architecture.

|*comps*|*connections*|

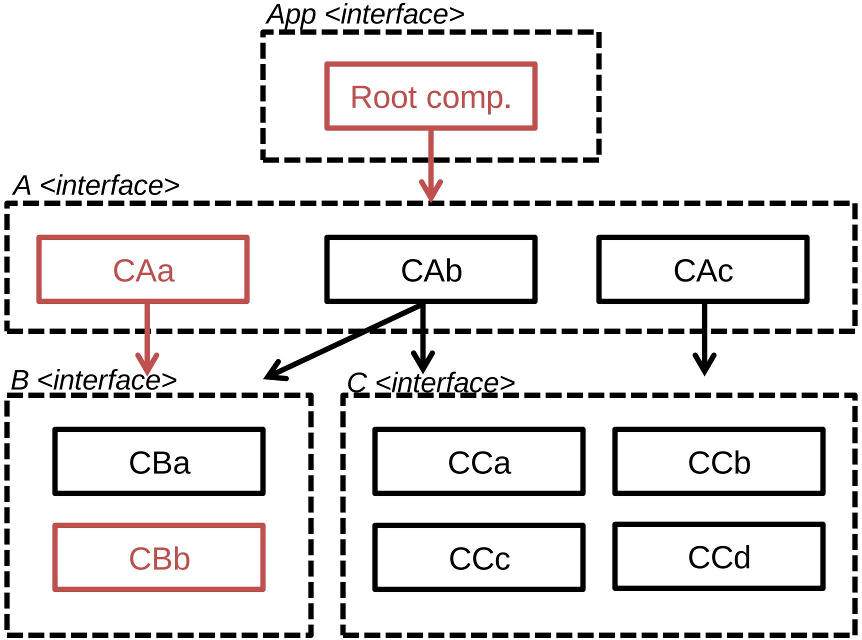


Assembly

- **Architectural Description (getAllConfigs(), getConfig())**

Architectural descriptions are full descriptions of the architecture, and it works as an unique identification code for the architecture.

|Root comp., CAa, CBb|connections|



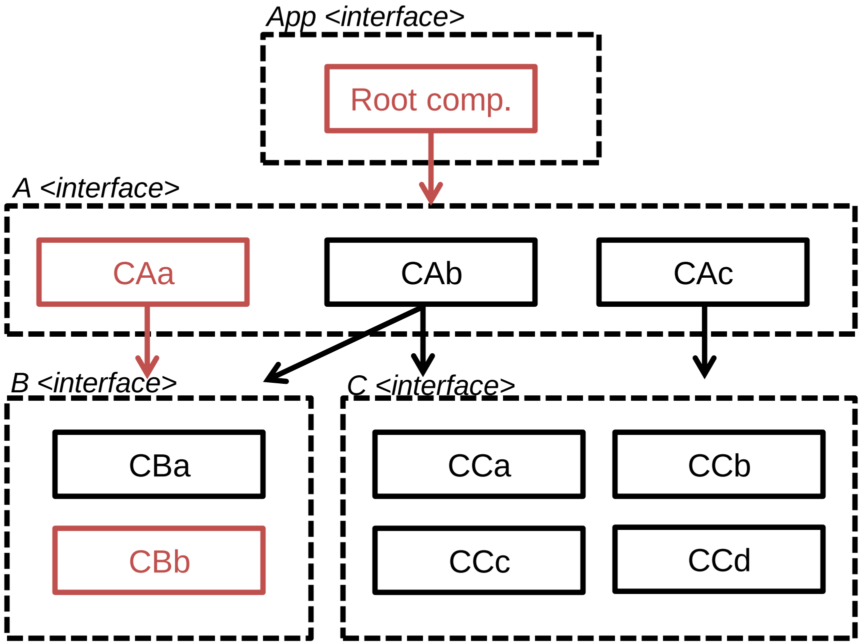
Assembly

- **Architectural Description (getAllConfigs(), getConfig())**

Architectural descriptions are full descriptions of the architecture, and it works as an unique identification code for the architecture.

|*Root comp., CAa, CBb*|

<component>:<interface>:<component>|



Assembly

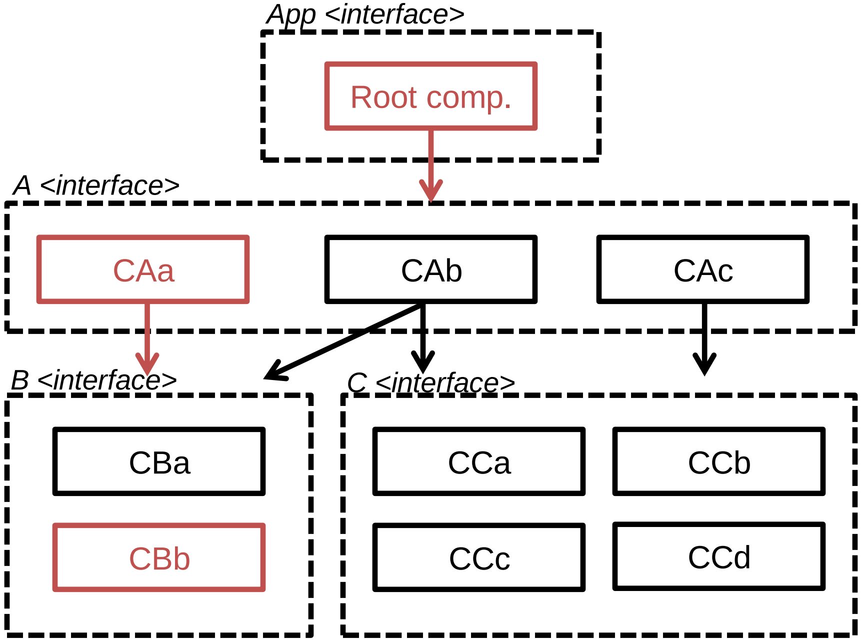
- **Architectural Description (getAllConfigs(), getConfig())**

Architectural descriptions are full descriptions of the architecture, and it works as an unique identification code for the architecture.

|*Root comp., CAa, CBb*|

Root comp.:A:CAa,

CAa:B:CBb|



Assembly

- **Architectural Description (getAllConfigs(), getConfig())**

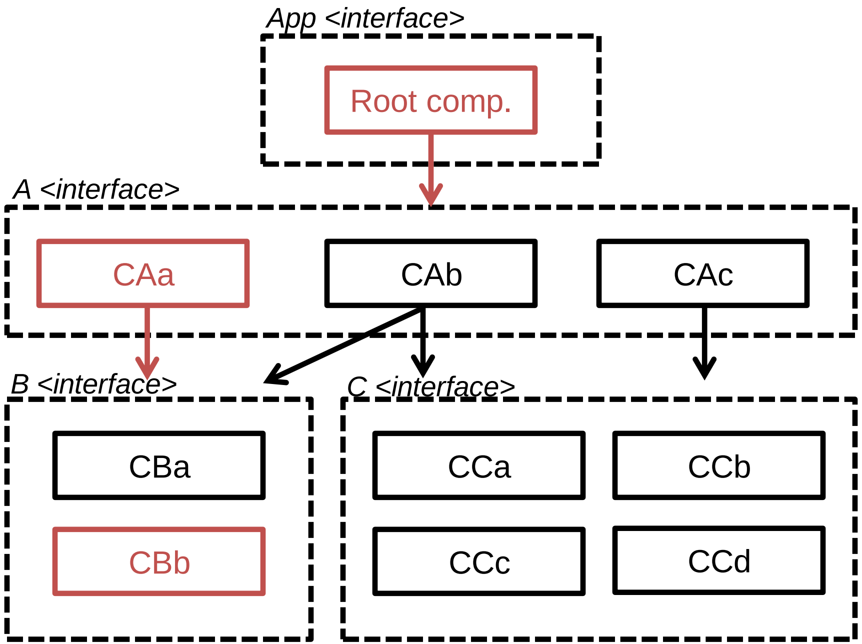
Architectural descriptions are full descriptions of the architecture, and it works as an unique identification code for the architecture.

getConfig():

|Root comp., CAa, CBb|

Root comp.:A:CAa,

CAa:B:CBb|



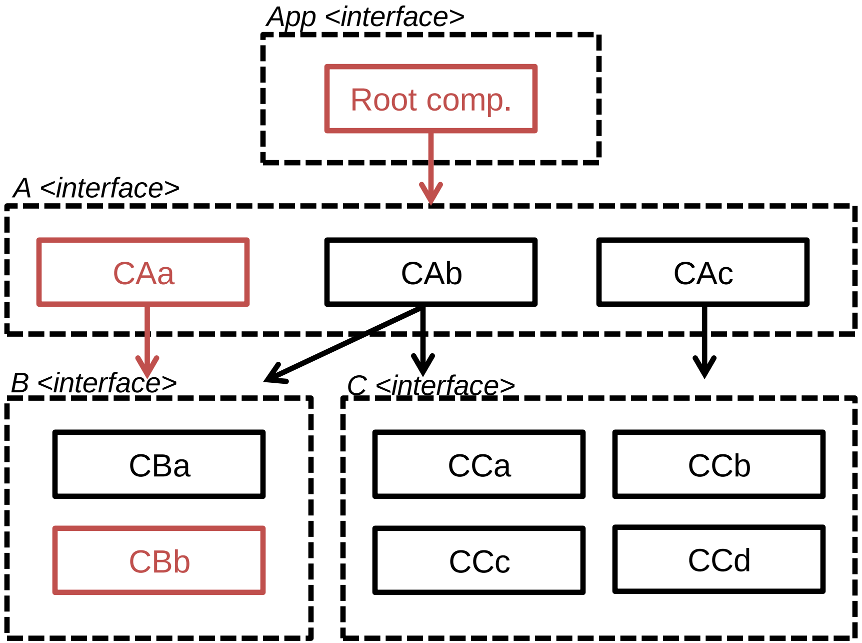
Assembly

- **Architectural Description (getAllConfigs(), getConfig())**

Architectural descriptions are full descriptions of the architecture, and it works as an unique identification code for the architecture.

getAllConfigs():

|Root comp., CAa, CBb|
Root comp.:A:CAa,
CAa:B:CBb|,
|Root comp., CAa, CBa|
Root comp.:A:CAa,
CAa:B:CBa|,
…



Assembly

- **Adaptation (setConfig())**

Assembly

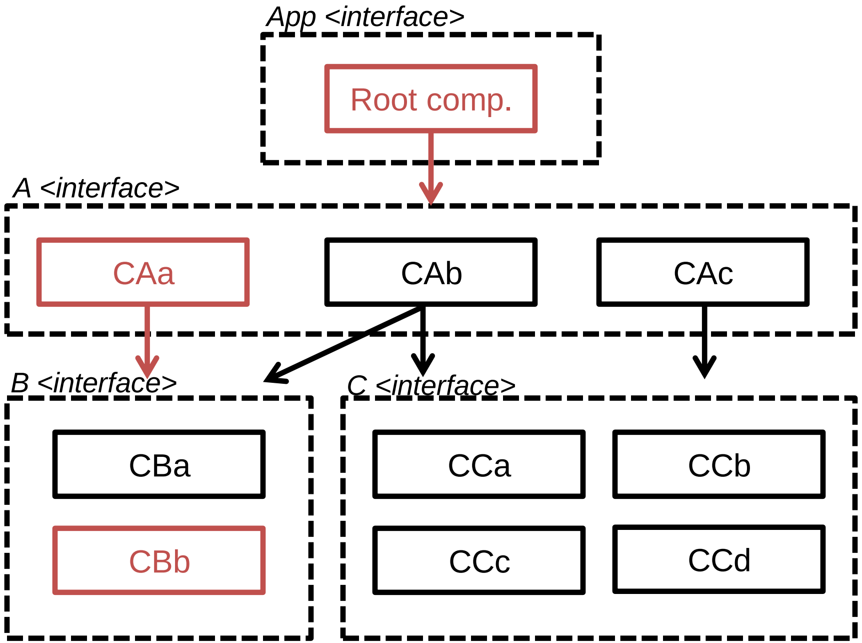
- **Adaptation (setConfig())**

Current composition:

|*Root comp., CAa, CBb*|

Root comp.:A:CAa,

CAa:B:CBb|



Assembly

- **Adaptation (setConfig())**

Current composition:

|Root comp., CAa, CBb|

Root comp.:A:CAa,

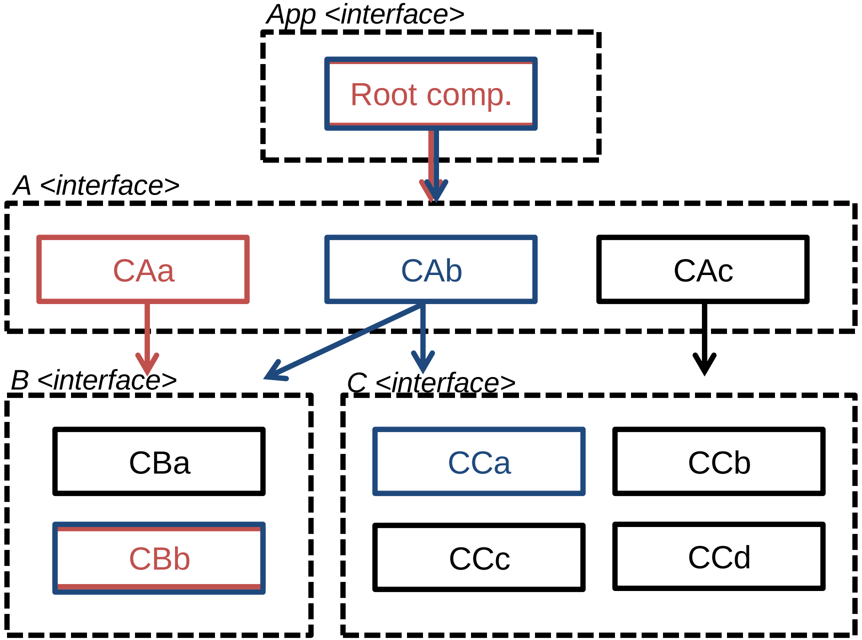
CAa:B:CBb|

New composition:

|Root comp., CAb, CBb, CCa|

Root comp.:A:CAb,

CAb:B:CBb, Cab:C:CCa|



Assembly

- **Adaptation (setConfig())**

Current composition:

|Root comp., CAa, CBb|

Root comp.:A:CAa,

CAa:B:CBb|

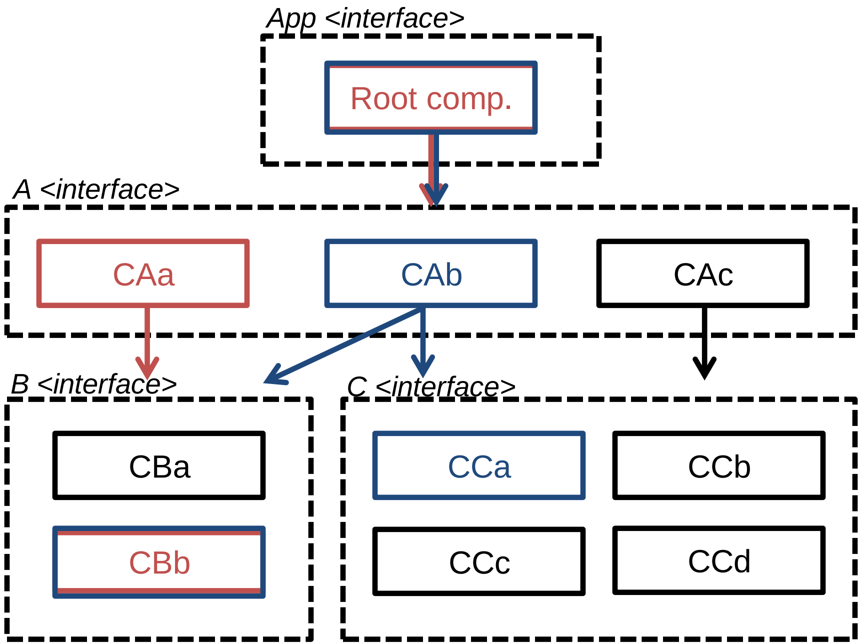
New composition:

|Root comp., CAb, CBb, CCa|

Root comp.:A:CAb,

CAb:B:CBb, Cab:C:CCa|

1. Identify common components



Assembly

- **Adaptation (setConfig())**

1. Identify common components

Current composition:

|Root comp., CAa, CBb|

Root comp.:A:CAa,

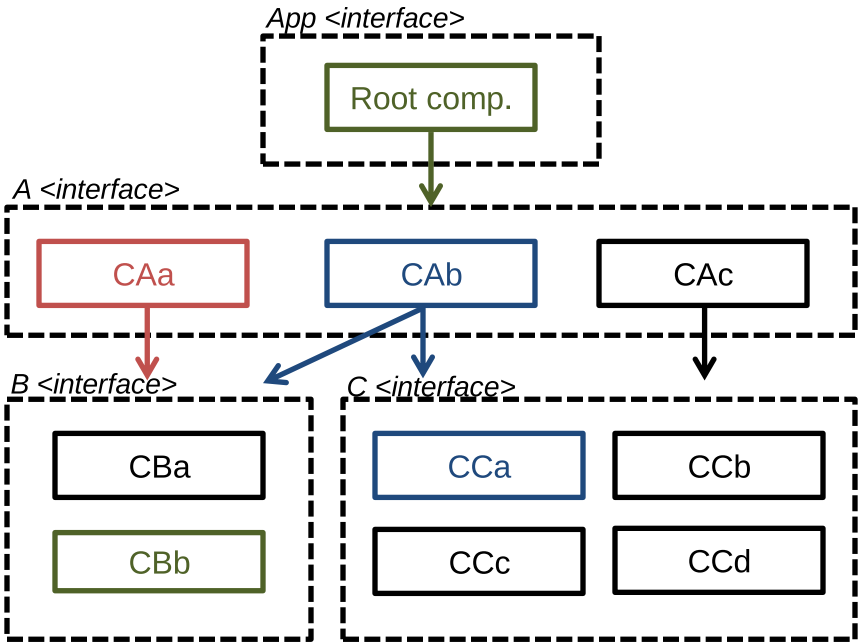
CAa:B:CBb|

New composition:

|Root comp., CAb, CBb, CCa|

Root comp.:A:CAb,

CAb:B:CBb, Cab:C:CCa|



Assembly

- **Adaptation (setConfig())**

Current composition:

|Root comp., CAa, CBb|

Root comp.:A:CAa,

CAa:B:CBb|

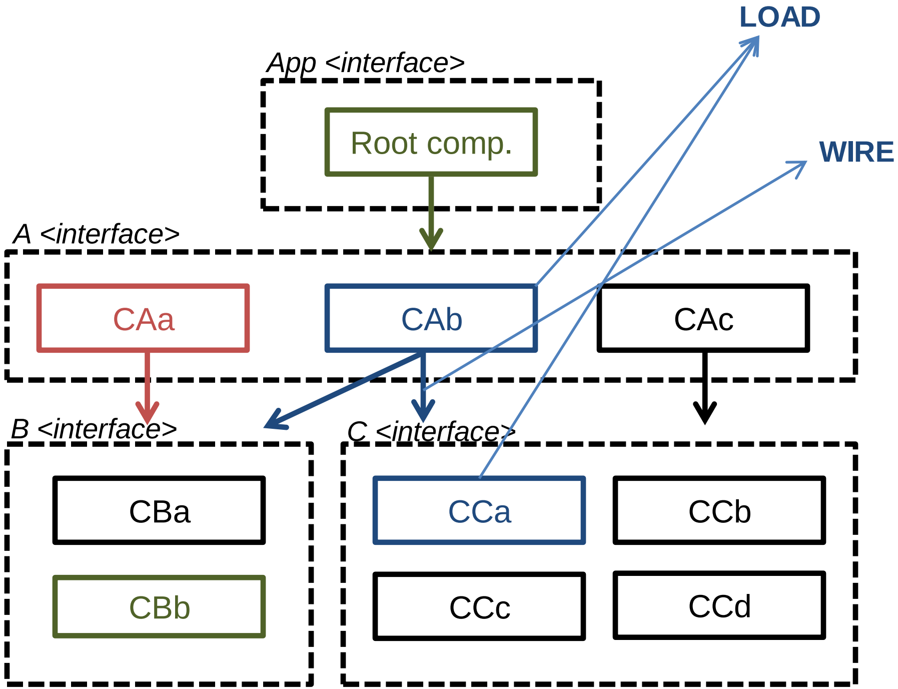
New composition:

|Root comp., CAb, CBb, CCa|

Root comp.:A:CAb,

CAb:B:CBb, Cab:C:CCa|

2. Load and wire the new components.



Assembly

- **Adaptation (setConfig())**

Current composition:

|Root comp., CAa, CBb|

Root comp.:A:CAa,

CAa:B:CBb|

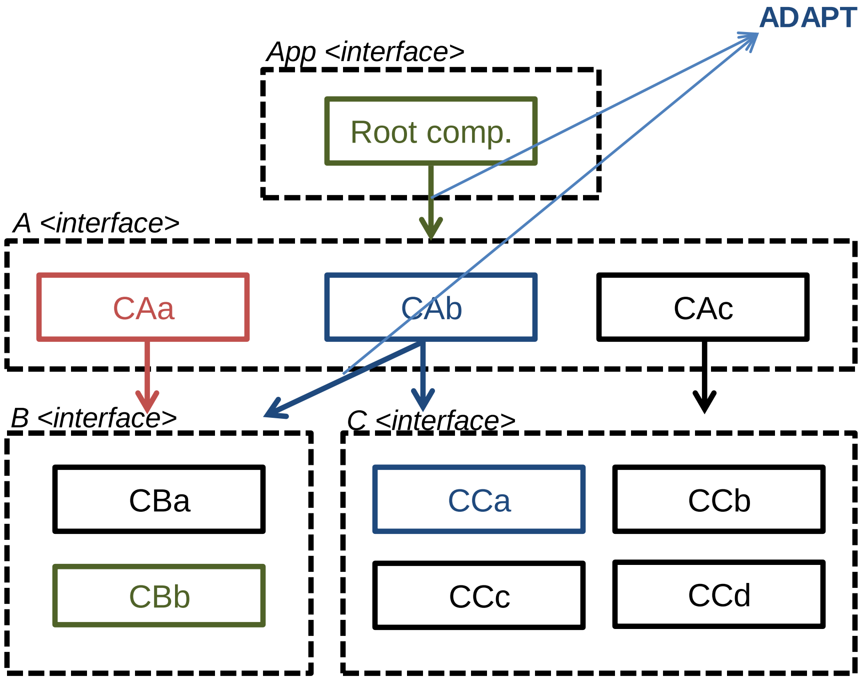
New composition:

|Root comp., CAb, CBb, CCa|

Root comp.:A:CAb,

CAb:B:CBb, Cab:C:CCa|

3. Adapt components.



Assembly

- **Adaptation (setConfig())**

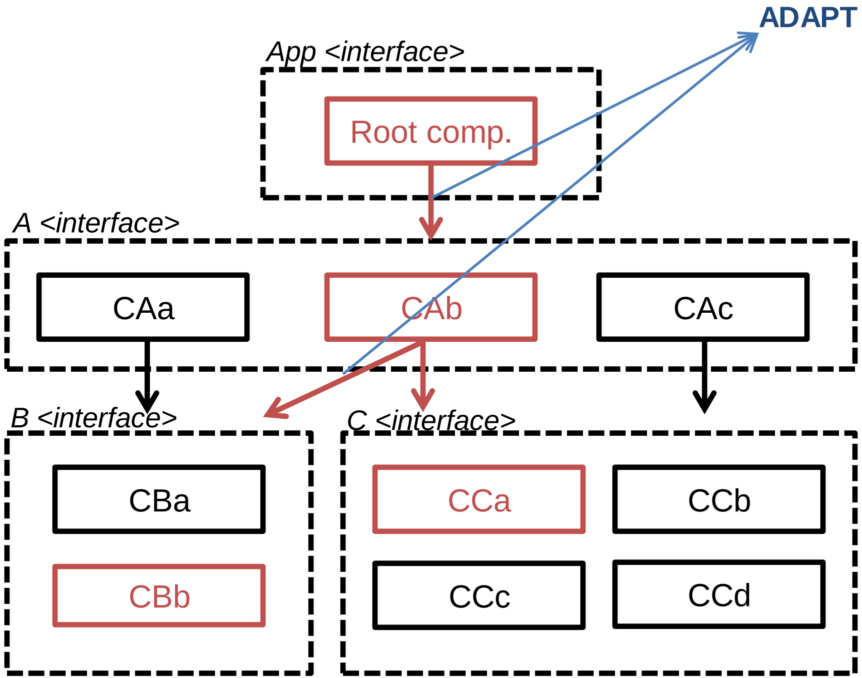
Old composition:

|Root comp., CAa, CBb|
Root comp.:A:CAa,
CAa:B:CBb|

Current composition:

|Root comp., CAb, CBb, CCa|
Root comp.:A:CAb,
CAb:B:CBb, Cab:C:CCa|

3. Adapt components.



Assembly

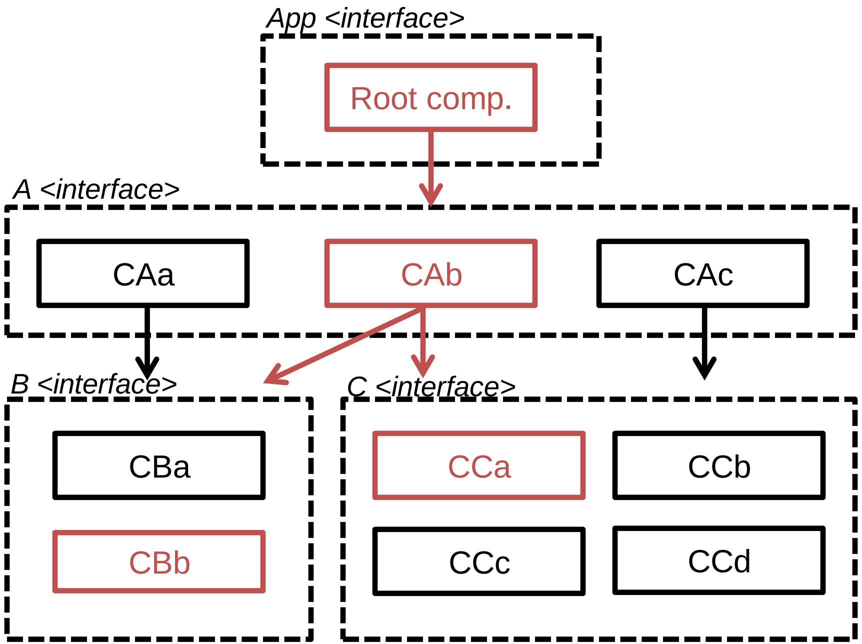
- **Adaptation (setConfig())**

Old composition:

|Root comp., CAa, CBb|
Root comp.:A:CAa,
CAa:B:CBb|

Current composition:

|Root comp., CAb, CBb, CCa|
Root comp.:A:CAb,
CAb:B:CBb, Cab:C:CCa|

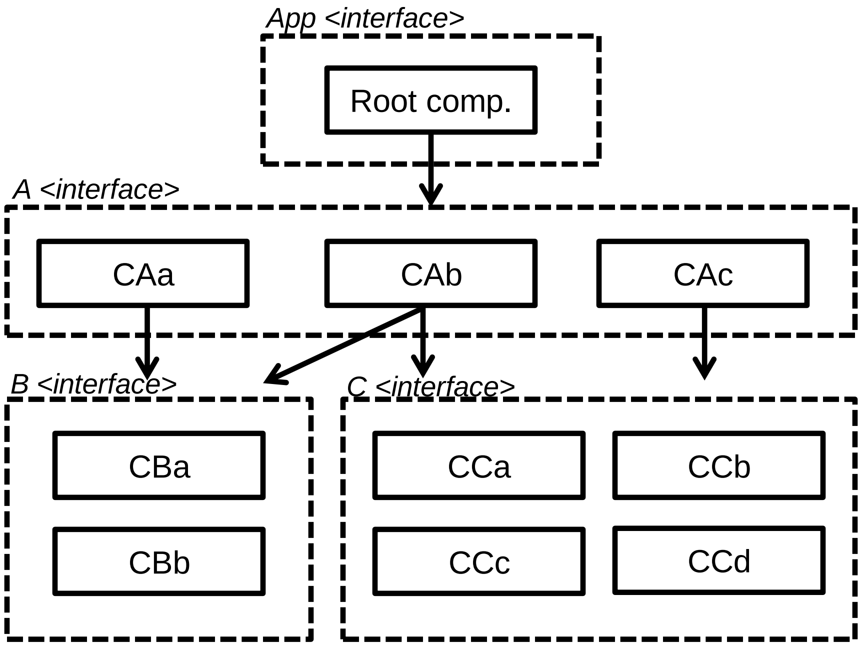


Assembly

- **Adding components**
  - Adding new components is important for ES concept because of its best effort optimisation strategy.

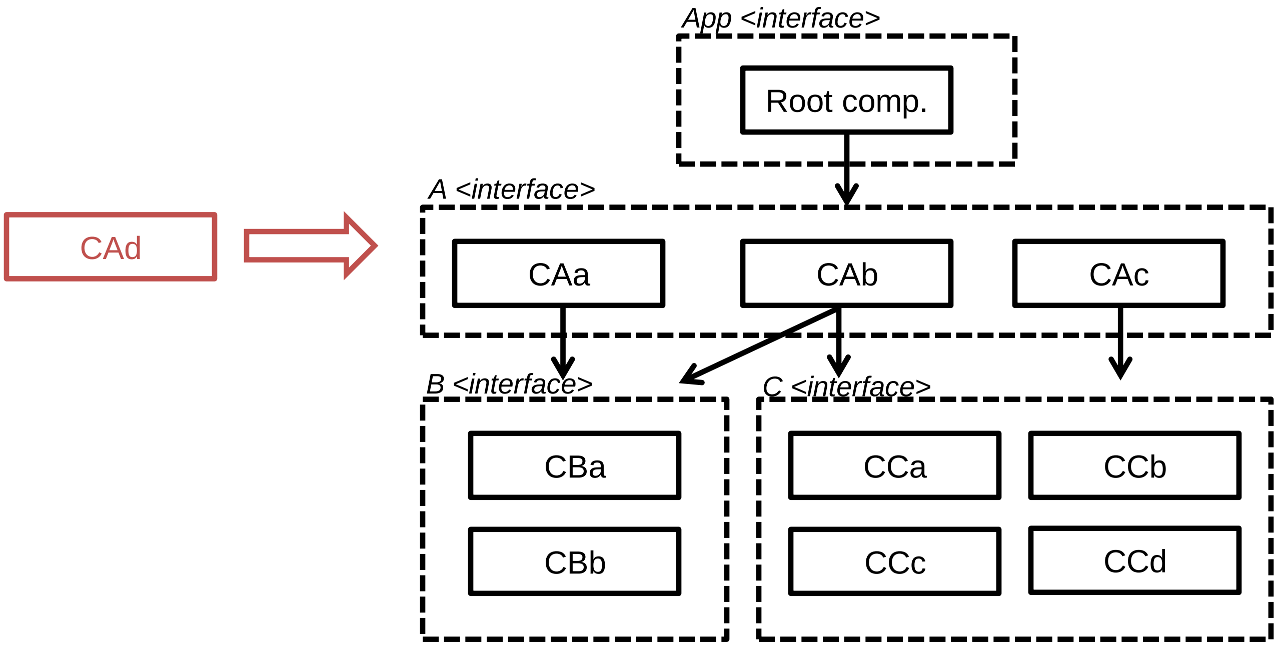
Assembly

- **Adding components**



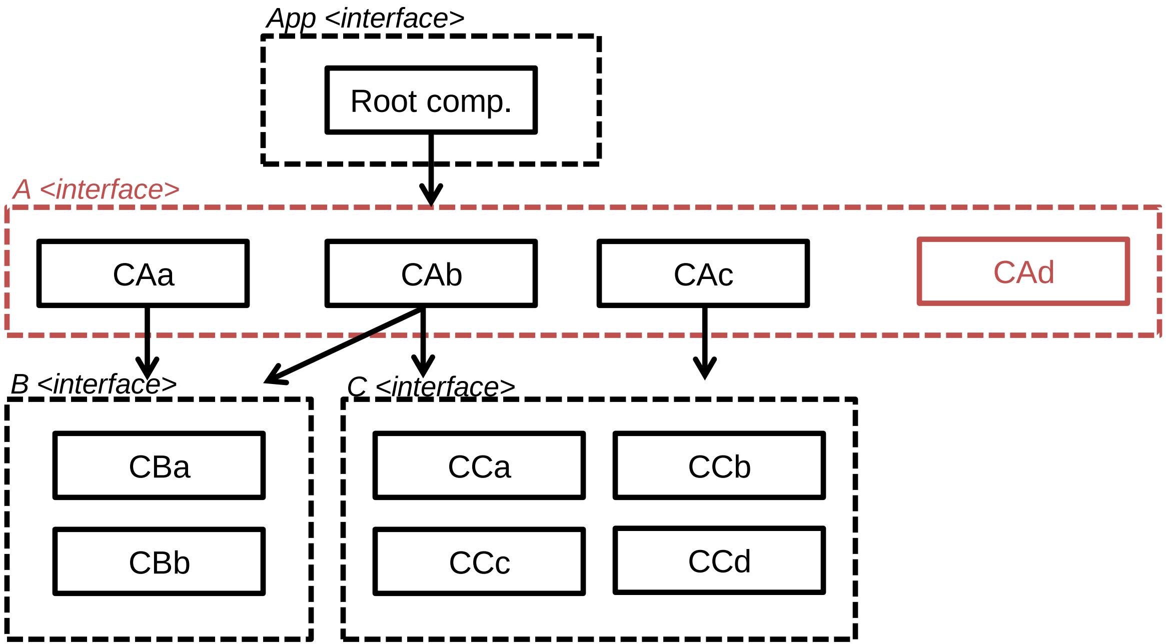
Assembly

- **Adding components**



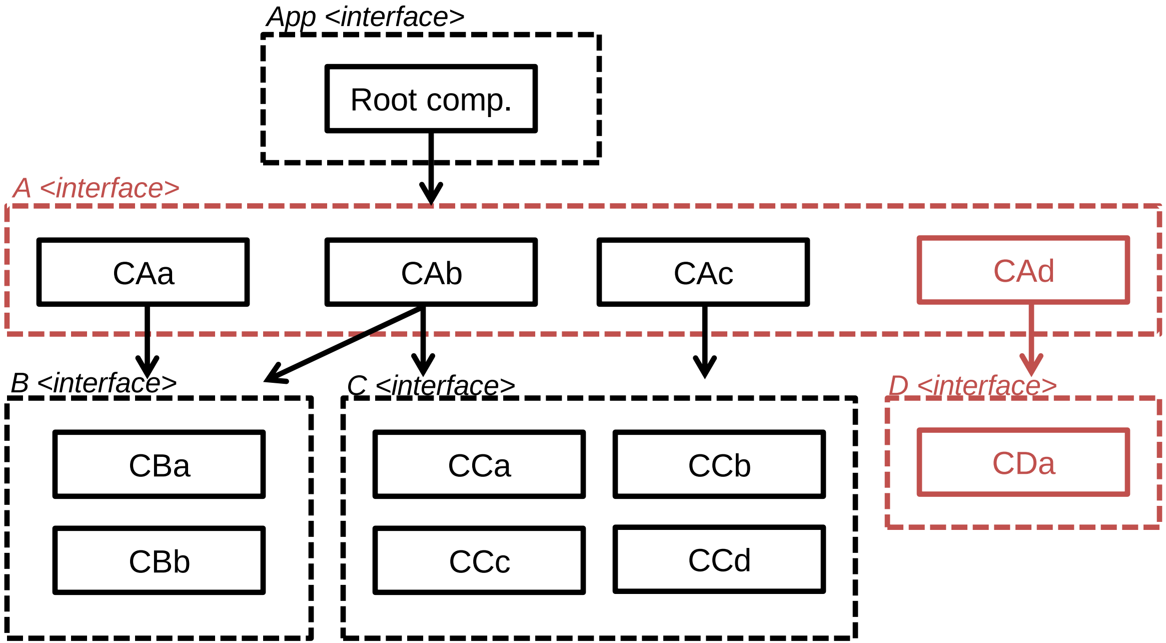
Assembly

- **Adding components**



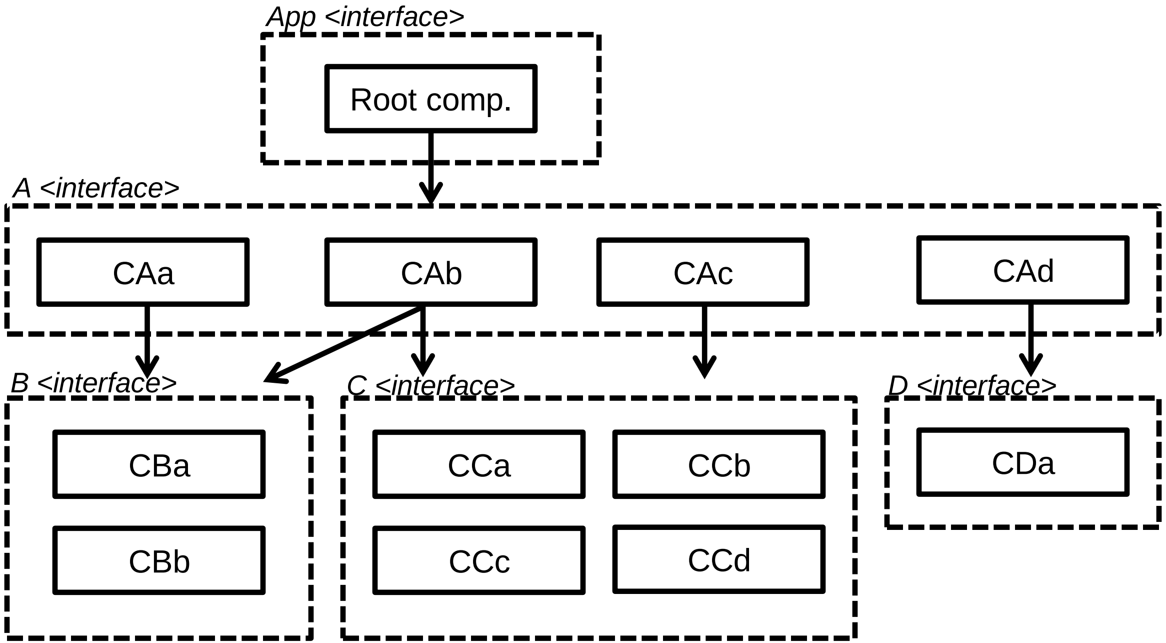
Assembly

- **Adding components**



Assembly

- **Adding components**

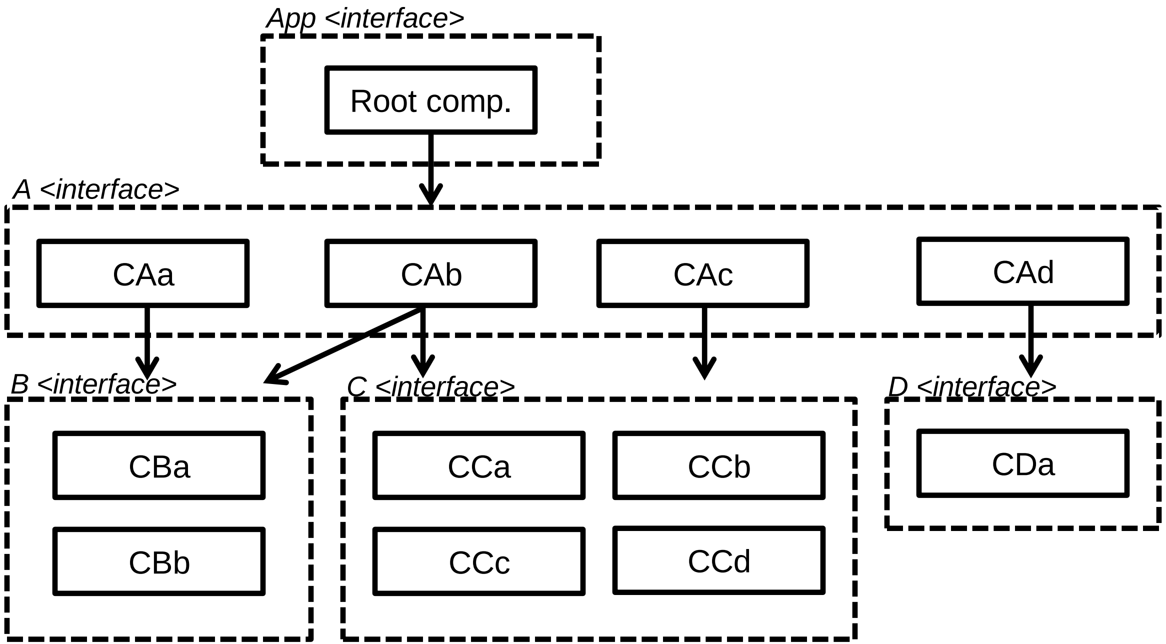


Assembly

- **Removing components**
  - Removing faulty components or components that do not contribute to the satisfaction of the systems goals, etc;

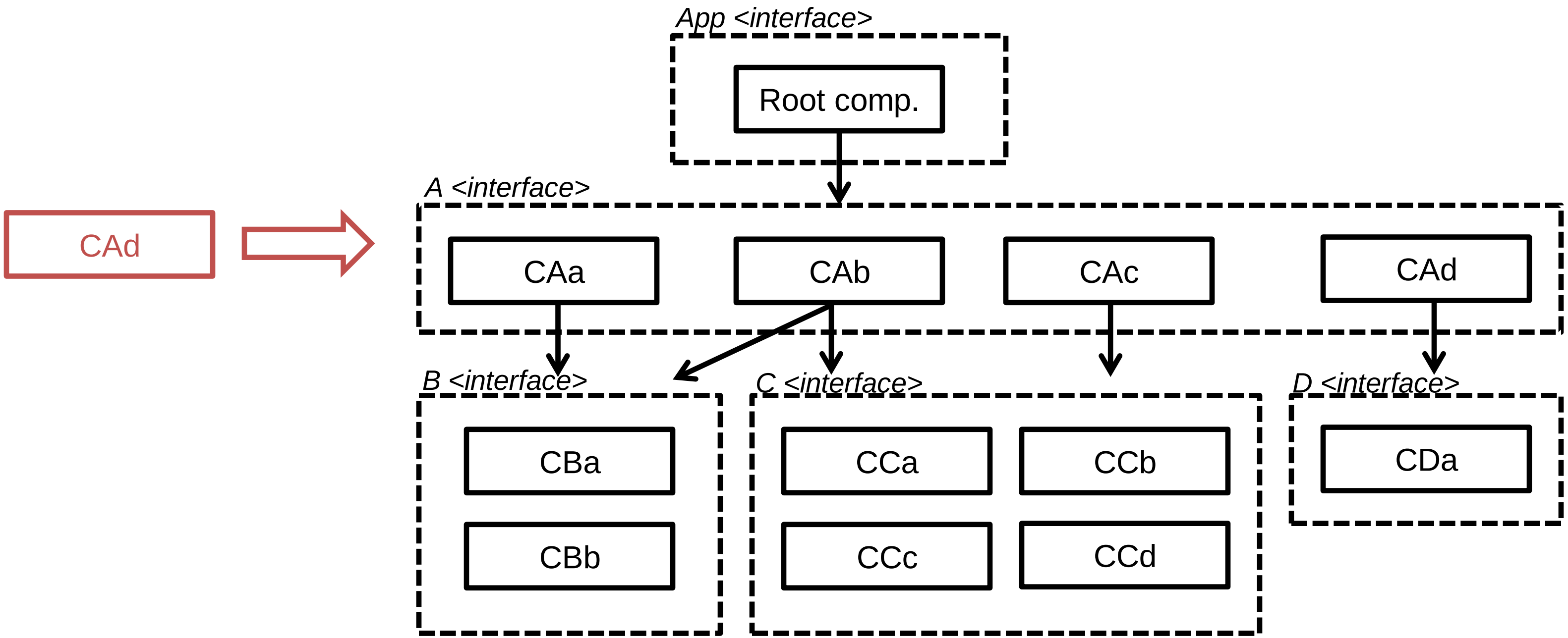
Assembly

- **Removing components**



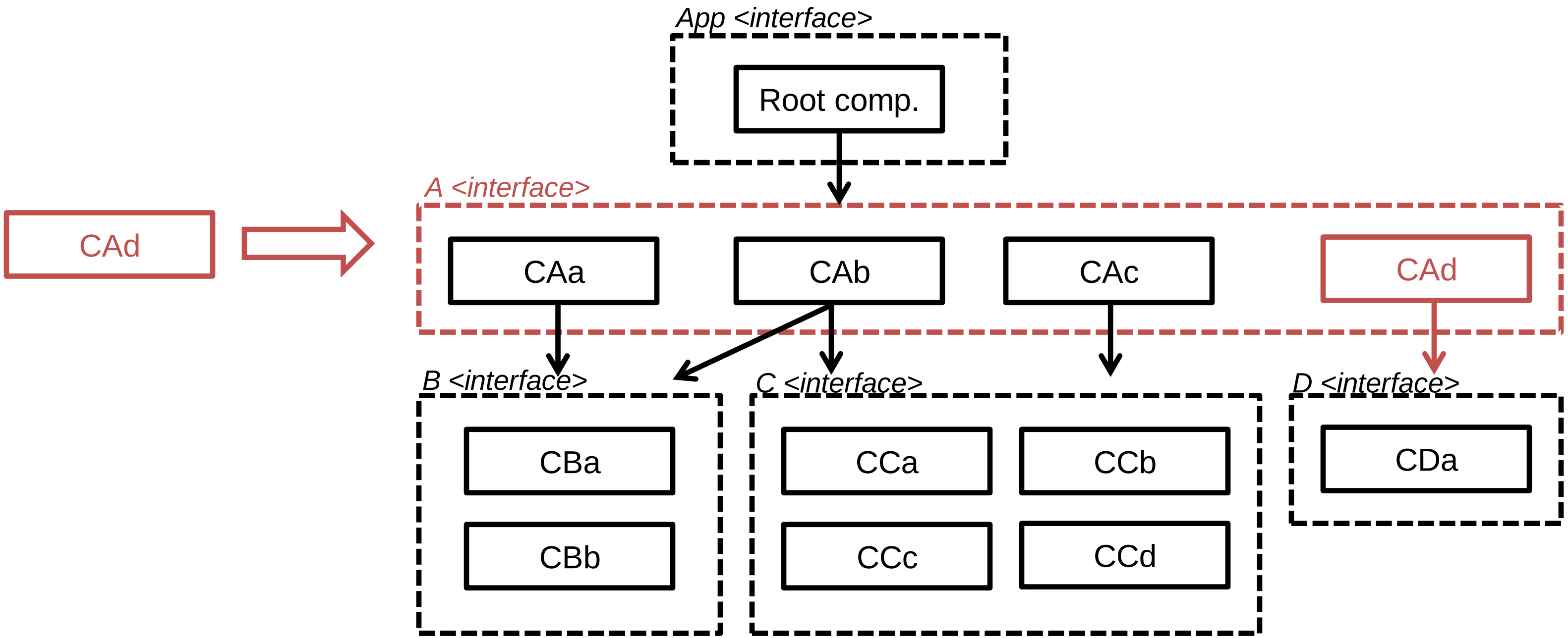
Assembly

- **Removing components**



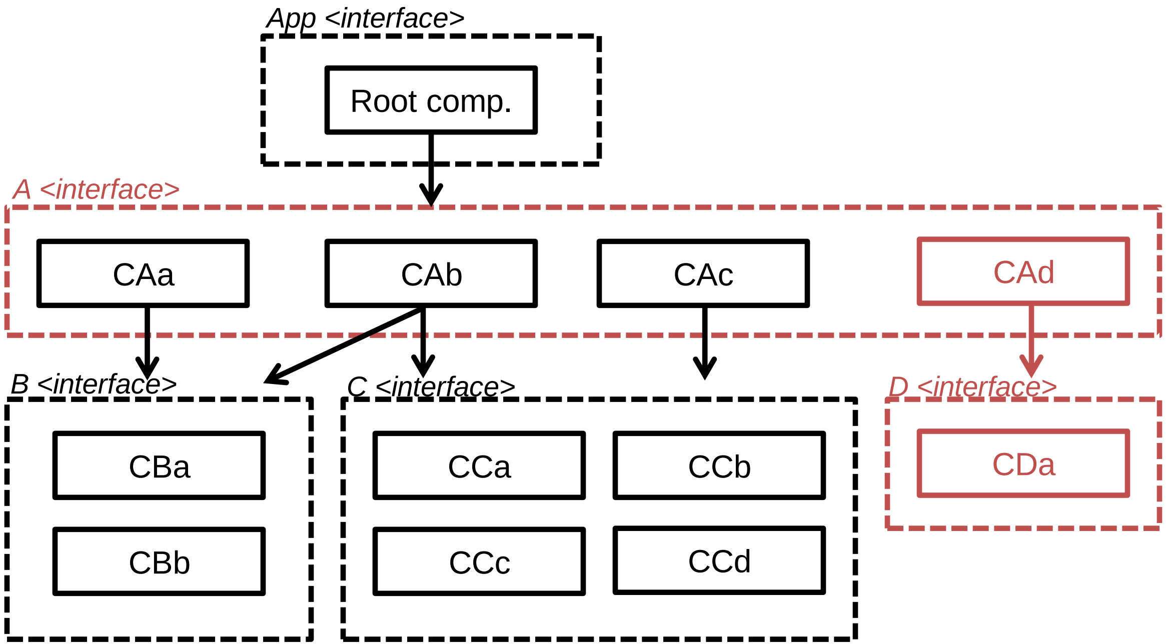
Assembly

- **Removing components**



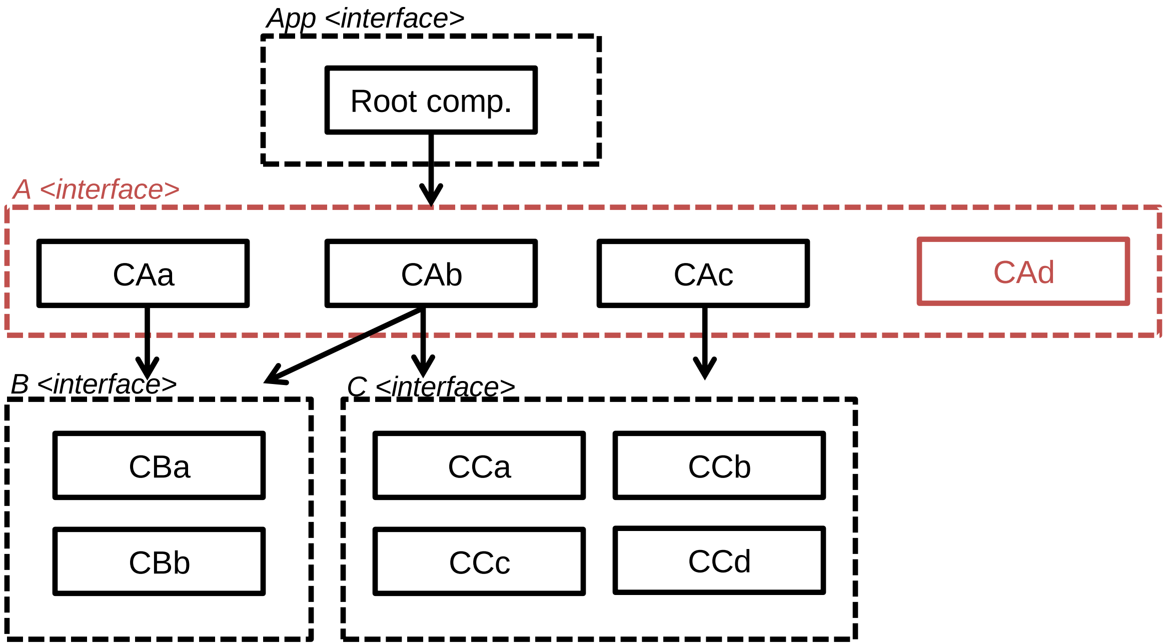
Assembly

- **Removing components**



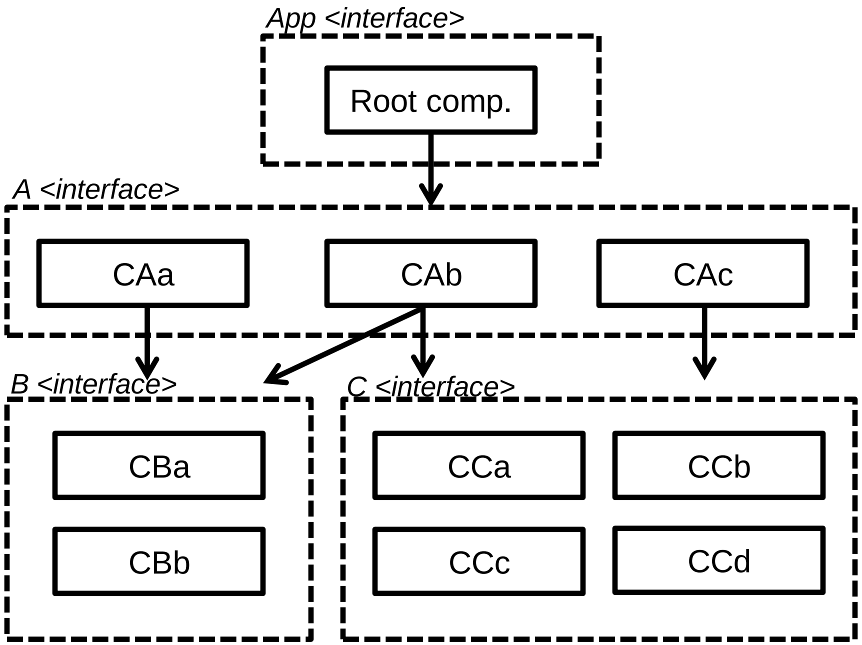
Assembly

- **Removing components**



Assembly

- **Removing components**



Perception

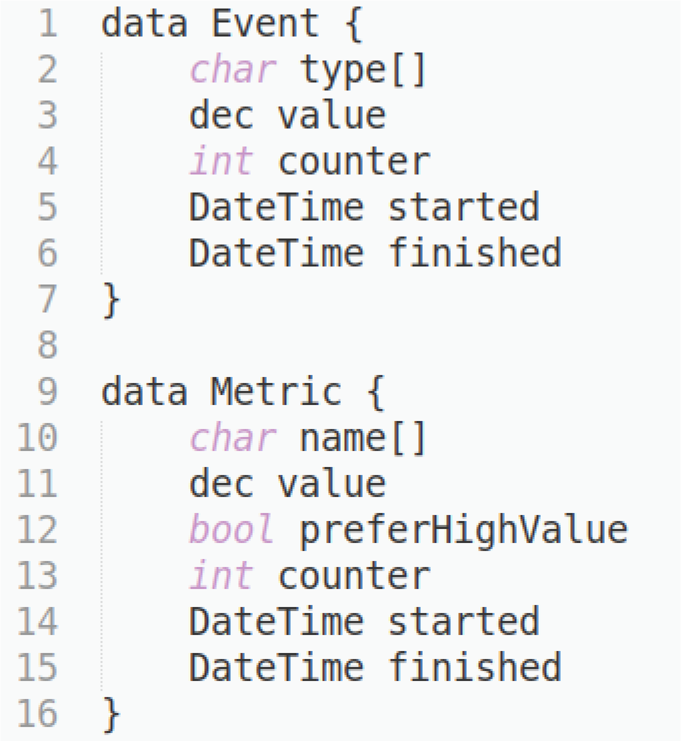
The Perception module handles the insertion of perception proxies into the target's application architectural composition to extract information on the application health and the operating environment on which the application is executing. The perception module is built on top of the Assembly module and provides the Assembly module's function and its own function to be accessible in a RESTful API.

Perception

- **Metrics**
  - A set of labels and numbers;
  - Used to monitor the health status of the systems;
- **Events**
  - A set of labels and numbers;
  - Used to characterise the operating environment;

Perception

- **Metrics and Events**



```
data Event {
    char type[]
    dec value
    int counter
    DateTime started
    DateTime finished
}

data Metric {
    char name[]
    dec value
    bool preferHighValue
    int counter
    DateTime started
    DateTime finished
}
```

Perception

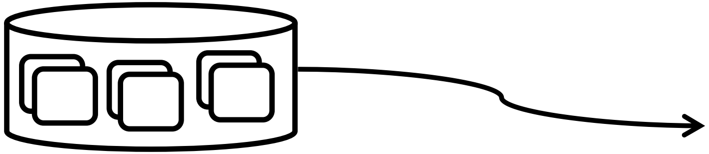
- **Perception API**
  - *String[] getProxies()*
  - *void addProxy(char exp[])*
  - *void removeProxy(char exp[])*
  - *char[] getPerceptionData()*

Perception

- **Get list of available proxy (*getProxies()*)**

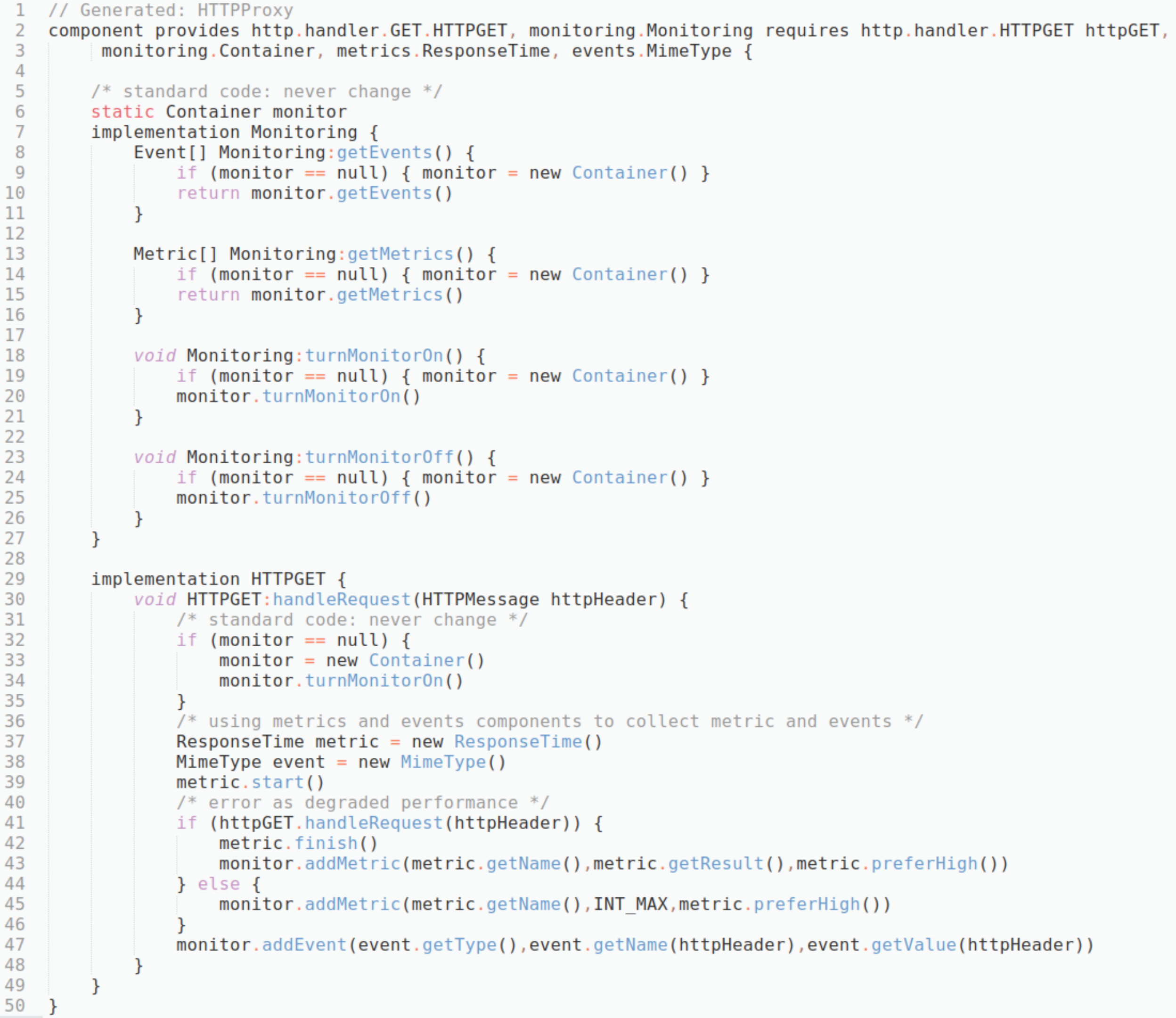
Perception

- **Get list of available proxy (*getProxies()*)**



Proxy Repository (Files System)

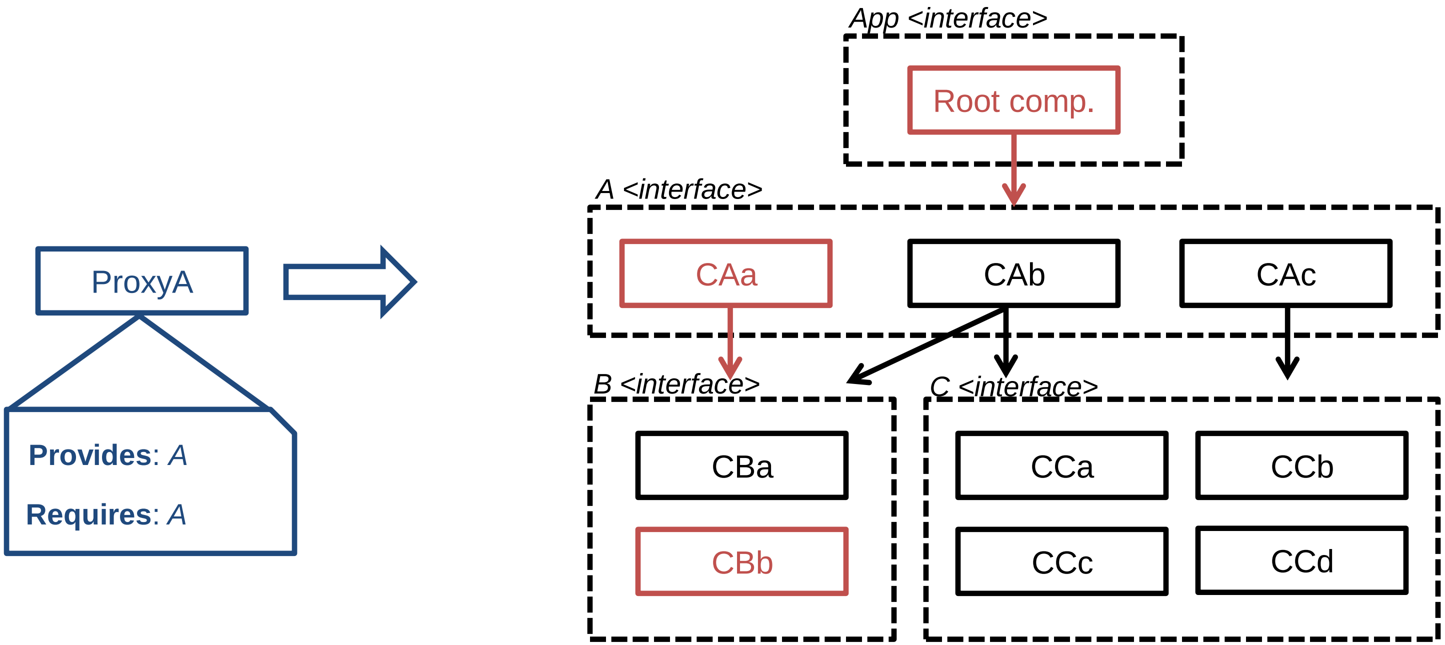
monitoring/HTTPProxy.o,
monitoring/RequestHandlerProxy.dn,
monitoring/CacheHandlerProxy.dn,
...



```
// Generated: HTTPProxy
component provides http.handler.GET.HTTPGET, monitoring.Monitoring requires http.handler.HTTPGET httpGET,
     monitoring.Container, metrics.ResponseTime, events.MimeType {

    /* standard code: never change */
    static Container monitor
    implementation Monitoring {
        Event[] Monitoring:getEvents() {
            if (monitor == null) { monitor = new Container() }
            return monitor.getEvents()
        }

        Metric[] Monitoring:getMetrics() {
            if (monitor == null) { monitor = new Container() }
            return monitor.getMetrics()
        }

        void Monitoring:turnMonitorOn() {
            if (monitor == null) { monitor = new Container() }
            monitor.turnMonitorOn()
        }

        void Monitoring:turnMonitorOff() {
            if (monitor == null) { monitor = new Container() }
            monitor.turnMonitorOff()
        }
    }

    implementation HTTPGET {
        void HTTPGET:handleRequest(HTTPMessage httpHeader) {
            /* standard code: never change */
            if (monitor == null) {
                monitor = new Container()
                monitor.turnMonitorOn()
            }
            /* using metrics and events components to collect metric and events */
            ResponseTime metric = new ResponseTime()
            MimeType event = new MimeType()
            metric.start()
            /* error as degraded performance */
            if (httpGET.handleRequest(httpHeader)) {
                metric.finish()
                monitor.addMetric(metric.getName(),metric.getResult(),metric.preferHigh())
            } else {
                monitor.addMetric(metric.getName(),INT_MAX,metric.preferHigh())
            }
            monitor.addEvent(event.getType(),event.getName(httpHeader),event.getValue(httpHeader))
        }
    }
}
```

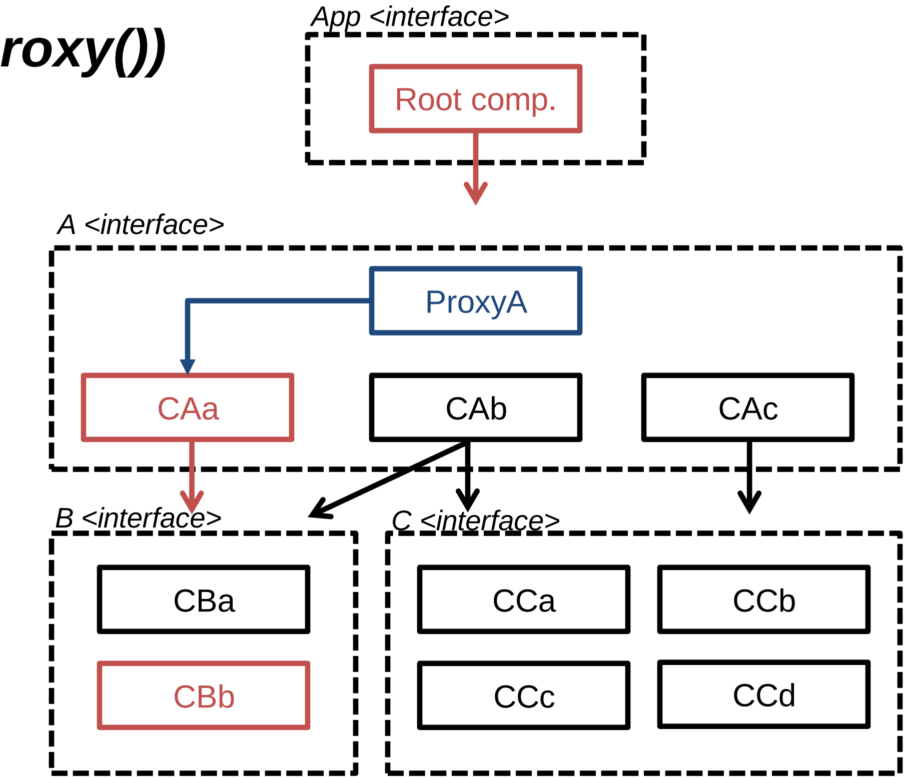
Perception

- **Adding proxy *(addProxy())***



Perception

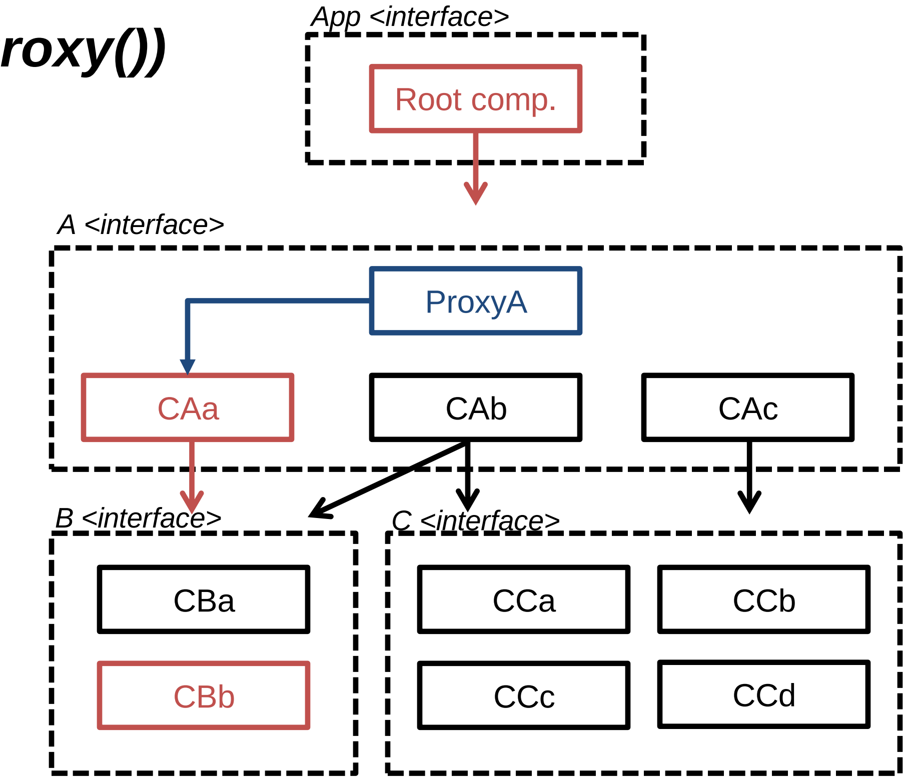
- **Adding proxy *(addProxy())***



Perception

What if we want flexibility in placing proxies?

- **Adding proxy *(addProxy())***



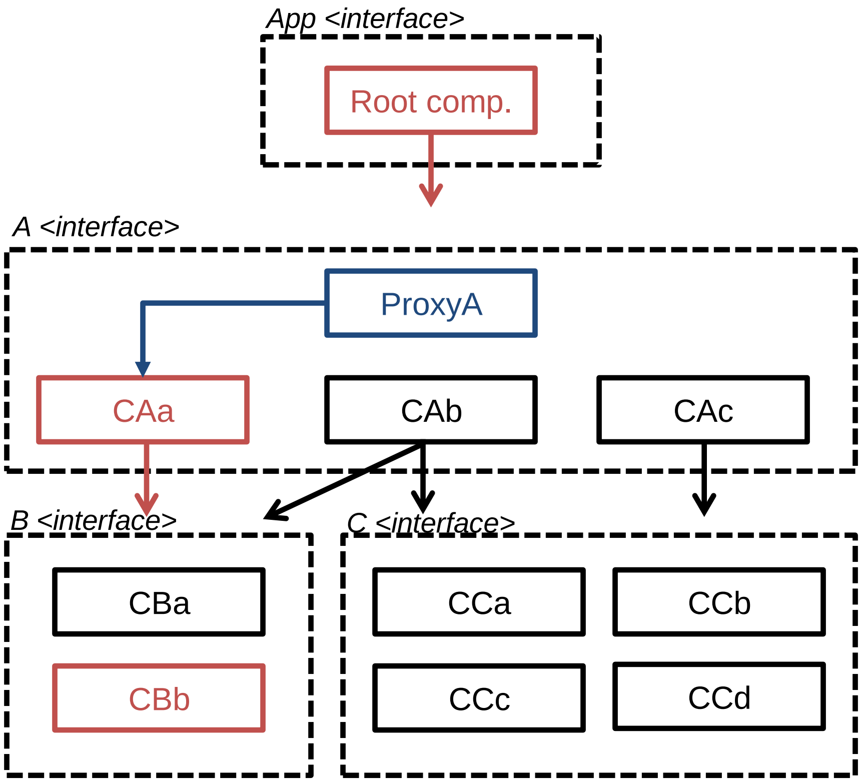
Perception

Proxy Expression Language!

- **Adding proxy *(addProxy())***

Expression:

|components list|expression|



Perception

- **Adding proxy *(addProxy())***

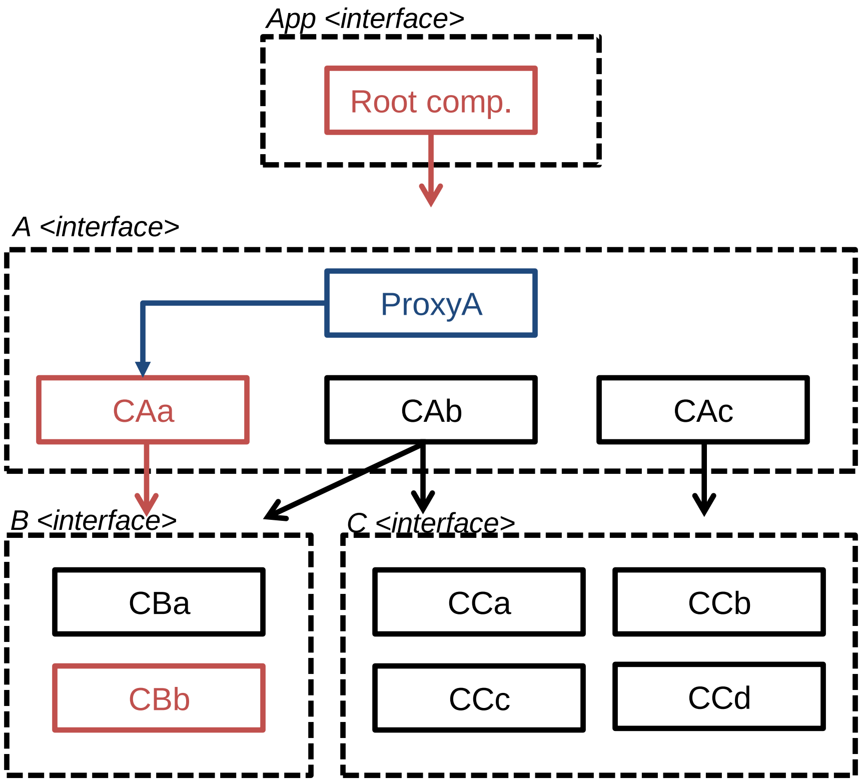
Expression:

|ProxyA|(*:A[0]:*)|*

**: any component*

[]: proxy

**(): apply the rule everywhere*



Perception

- **Adding proxy *(addProxy())***

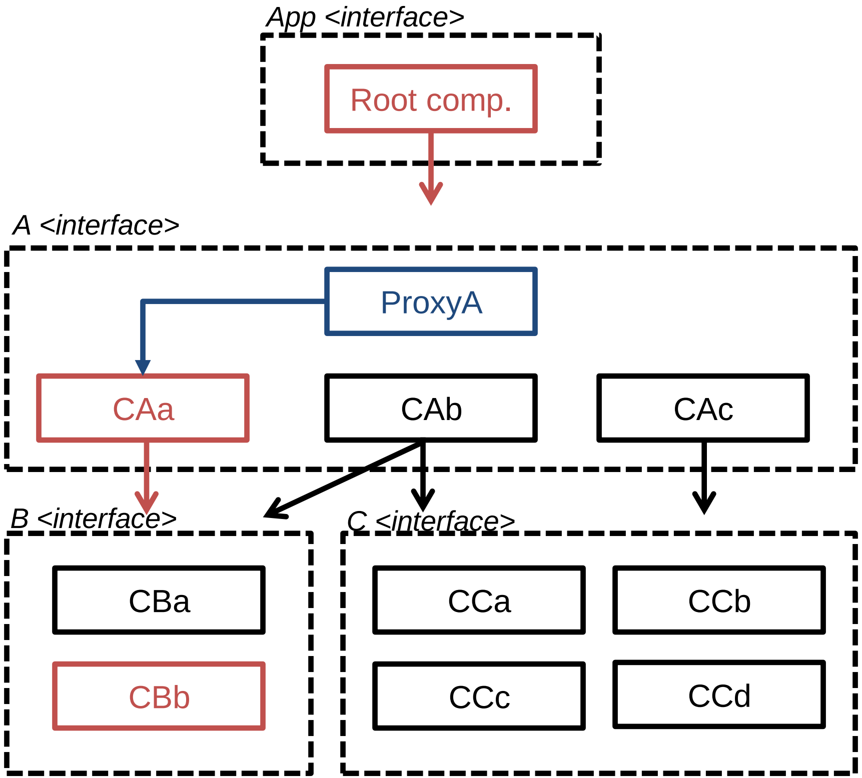
Expression:

|Root comp.,ProxyA|1(0:A[1]:)|*

**: any component*

[]: proxy

n(): apply rules n times



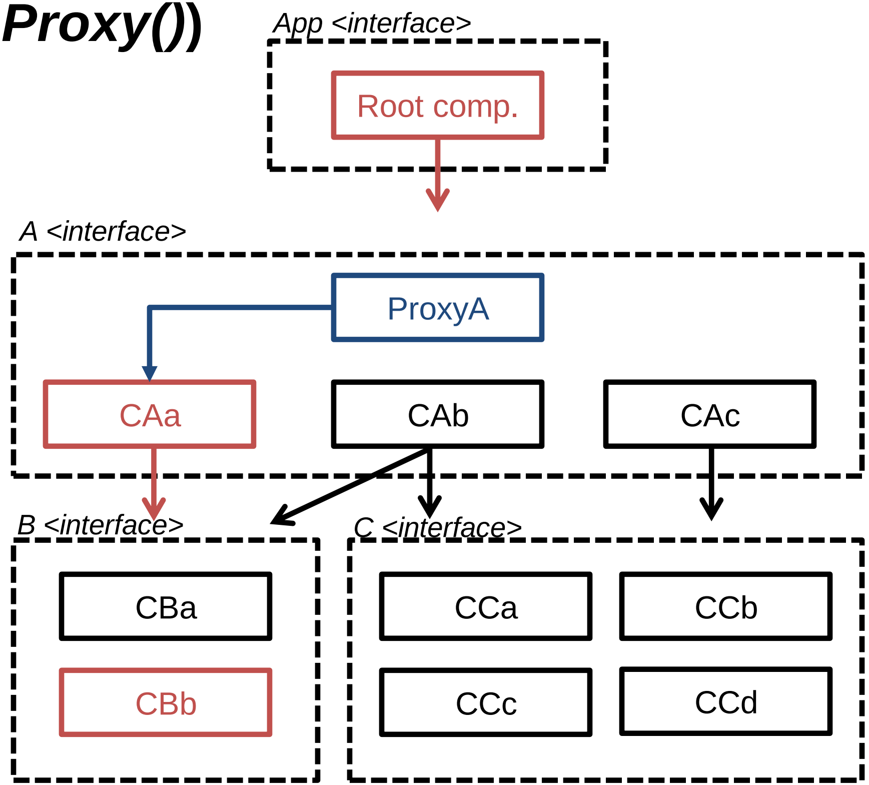
Perception

- **Removing proxy (*removeProxy()*)**

Perception

- **Removing proxy (*removeProxy()*)**

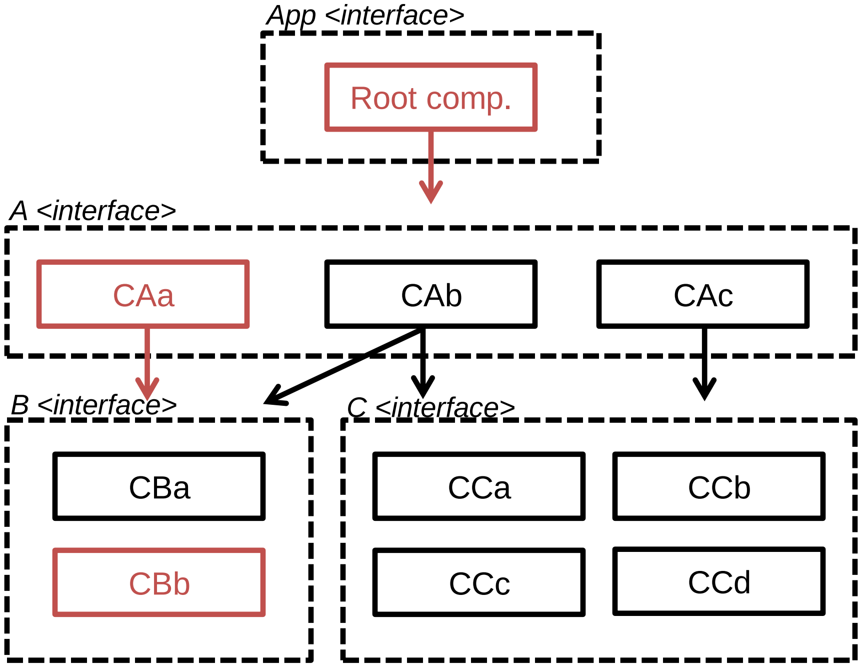
|Root comp.,ProxyA|1(0:A[1]:)|*



Perception

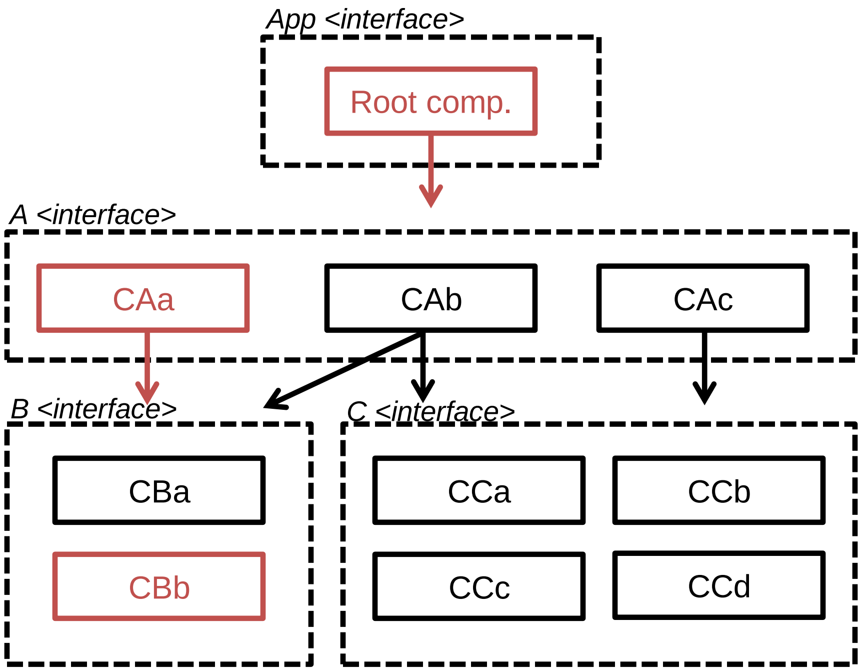
- **Removing proxy (*removeProxy()*)**

|Root comp.,ProxyA|1(0:A[1]:)|*



Perception

- **Removing proxy (*removeProxy()*)**



Perception

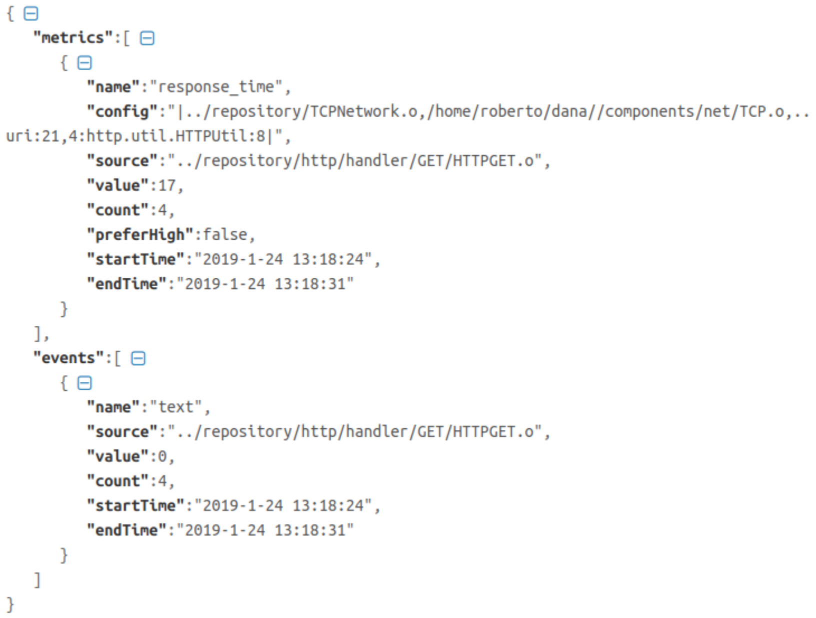
- **Obtaining metrics and events (*getPerceptionData()*)**

Perception

- **Obtaining metrics and events (*getPerceptionData()*)**
  - *Returns metrics and events in a JSON fomart*
  - *Used by the Learning module to collect information on the system*

Perception

- **Obtaining metrics and events (*getPerceptionData()*)**
  - *Returns metrics and events in a JSON fomart*
  - *Used by the Learning module to collect information on the system*



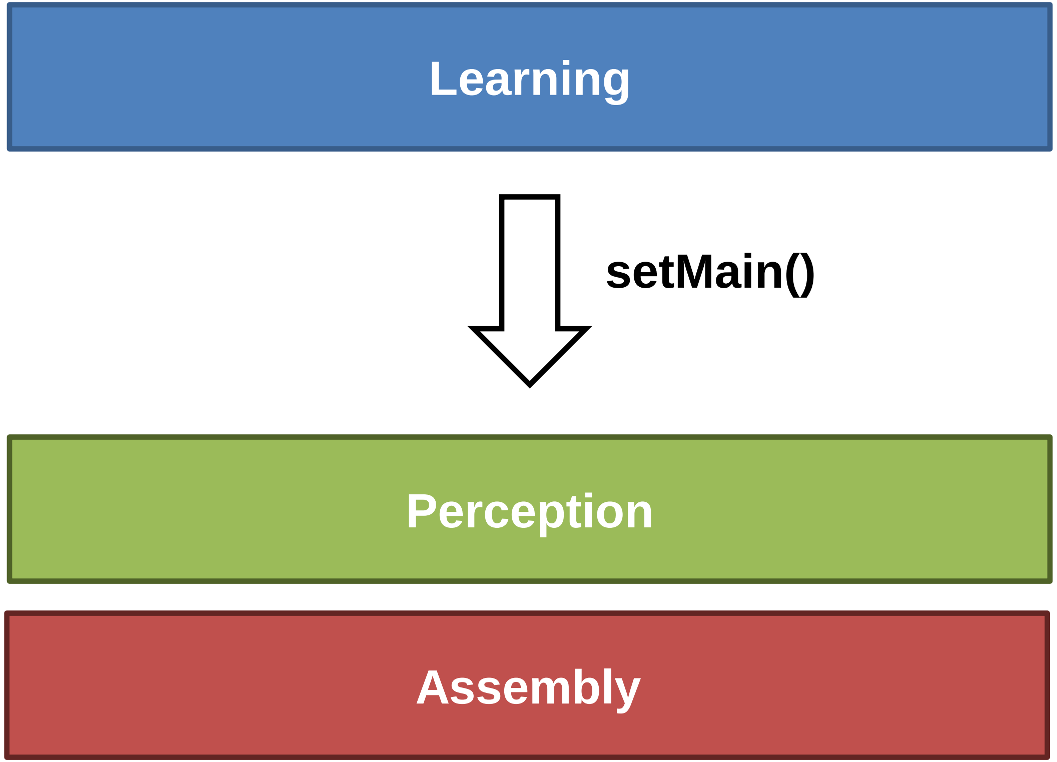
```
{ ⊟
   "metrics":[ ⊟
      { ⊟
         "name":"response_time",
         "config":"|../repository/TCPNetwork.o,/home/roberto/dana//components/net/TCP.o,..
uri:21,4:http.util.HTTPUtil:8|",
         "source":"../repository/http/handler/GET/HTTPGET.o",
         "value":17,
         "count":4,
         "preferHigh":false,
         "startTime":"2019-1-24 13:18:24",
         "endTime":"2019-1-24 13:18:31"
      }
   ],
   "events":[ ⊟
      { ⊟
         "name":"text",
         "source":"../repository/http/handler/GET/HTTPGET.o",
         "value":0,
         "count":4,
         "startTime":"2019-1-24 13:18:24",
         "endTime":"2019-1-24 13:18:31"
      }
   ]
}
```

Learning

The Learning module uses the RESTful API made available by the Perception and Assembly modules to compose a fully functioning system, experiment with the different architectural composition, and based on the perception data the Learning extracts from the target system through the Perception module, the Learning can identify the most suitable architectural composition for the target application goal and the executing environment.

Learning

Setting up



Learning

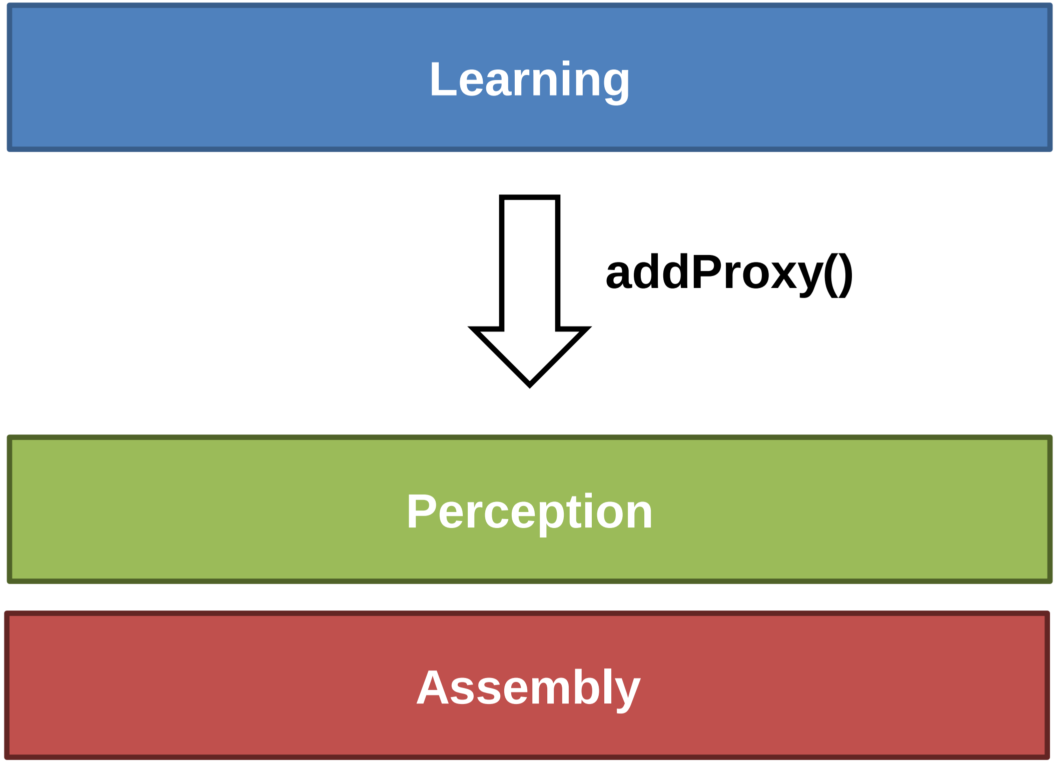
setMain()

Perception

Assembly

Learning

Setting up



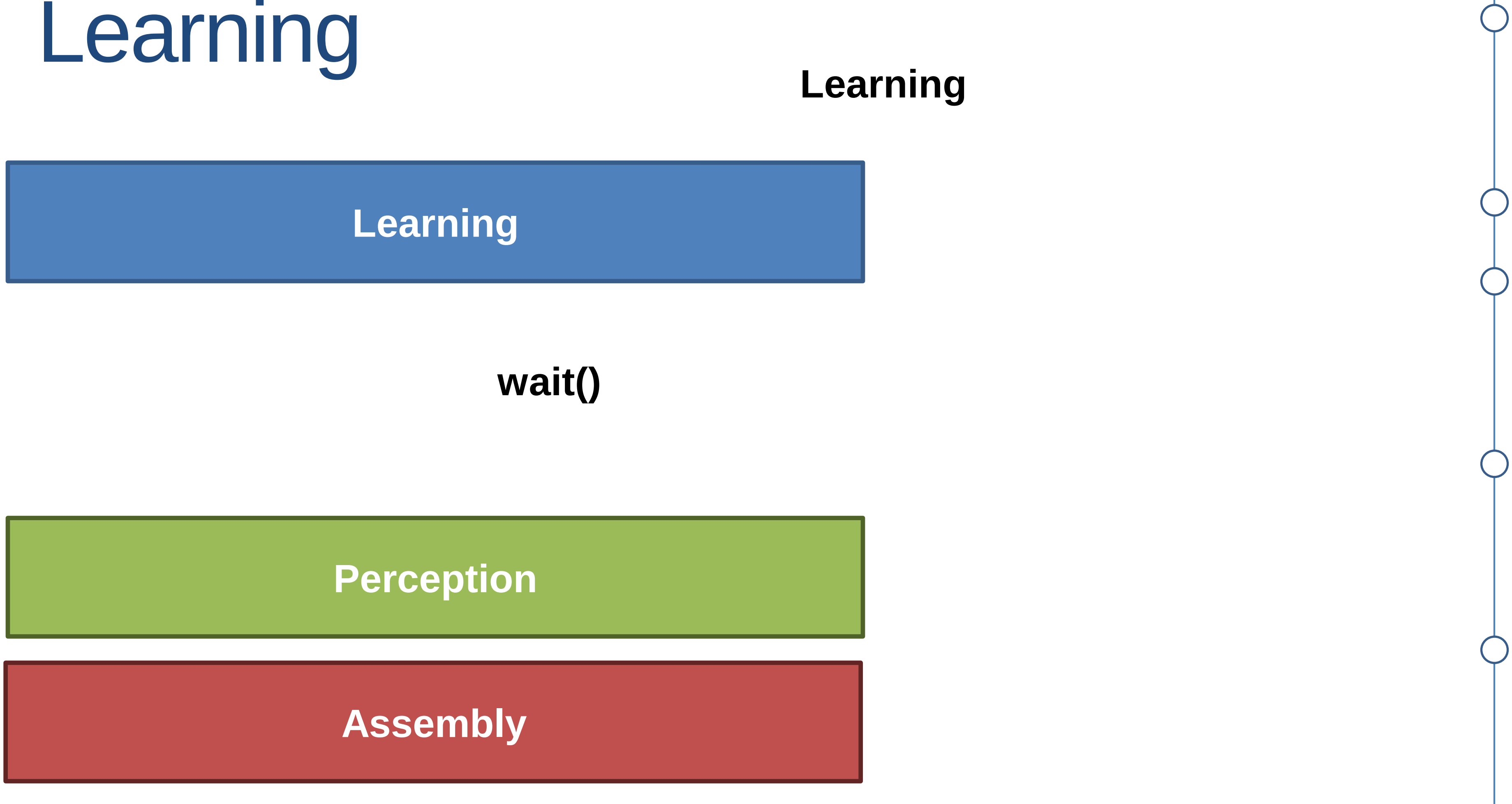
Learning

addProxy()

Perception

Assembly

Learning



Learning

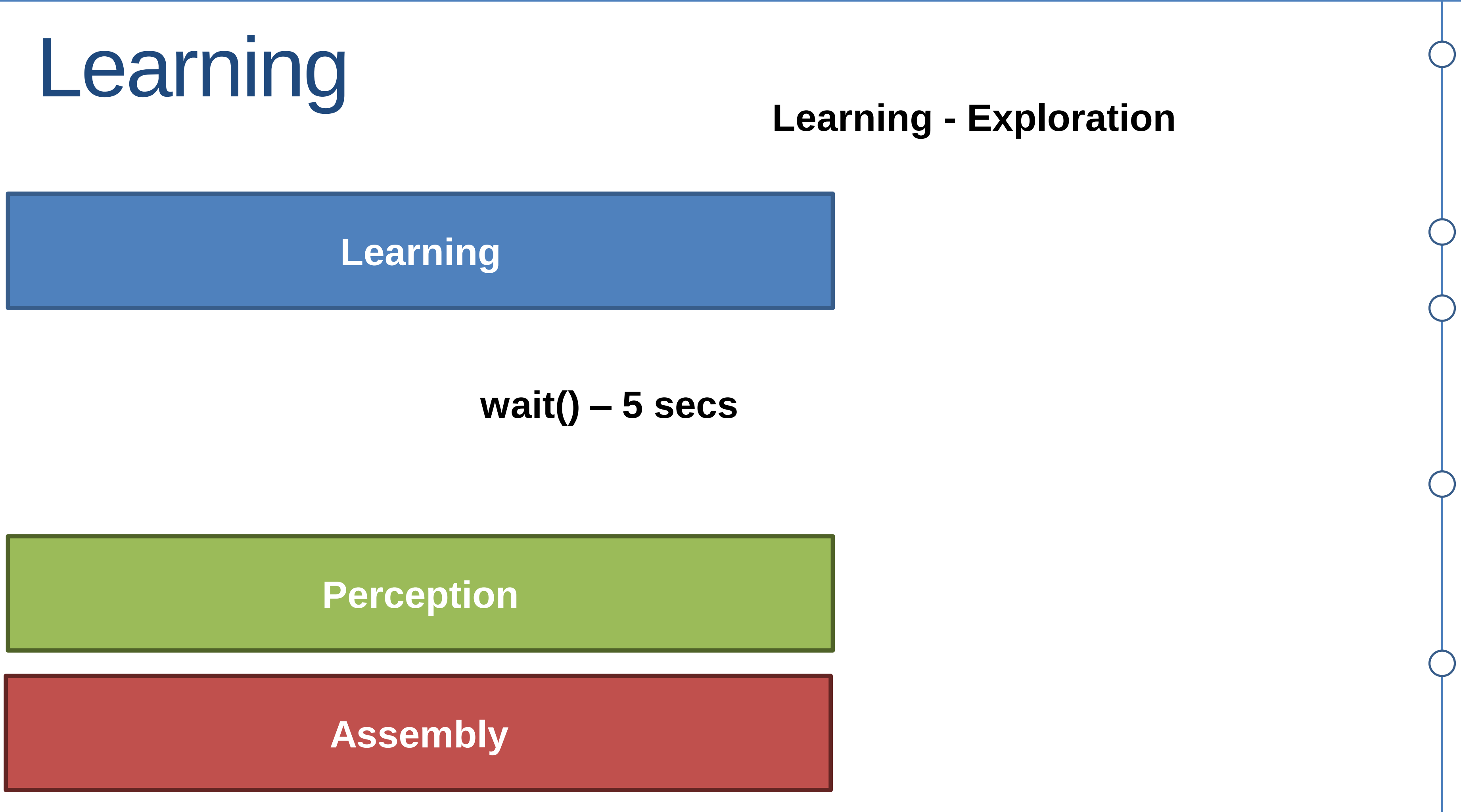
Learning

wait()

Perception

Assembly

Learning



Learning - Exploration

Learning

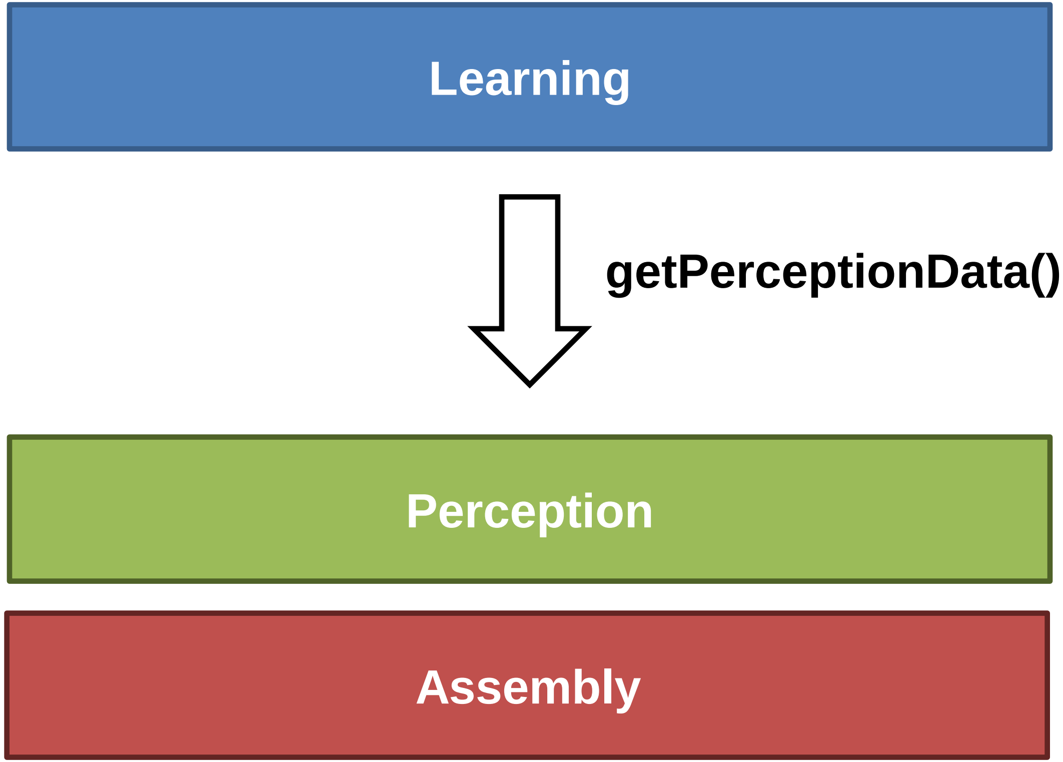
wait() – 5 secs

Perception

Assembly

Learning

Learning - Exploration



Learning

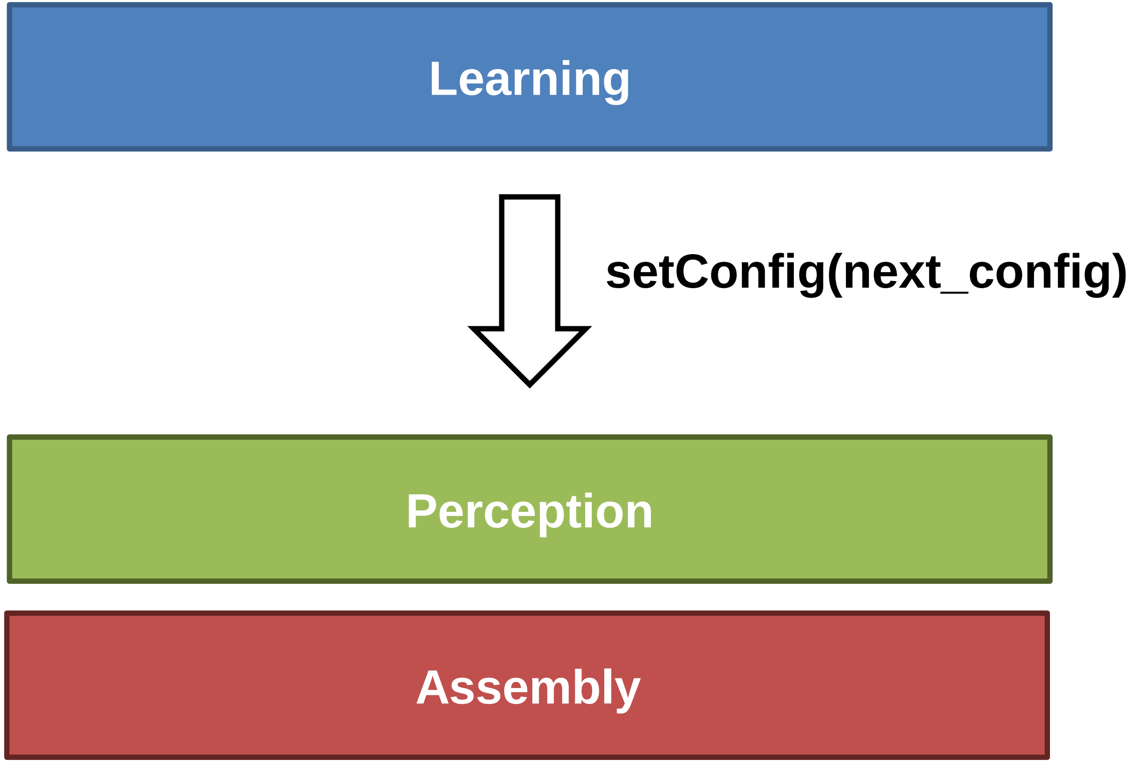
getPerceptionData()

Perception

Assembly

Learning

Learning - Exploration



Learning

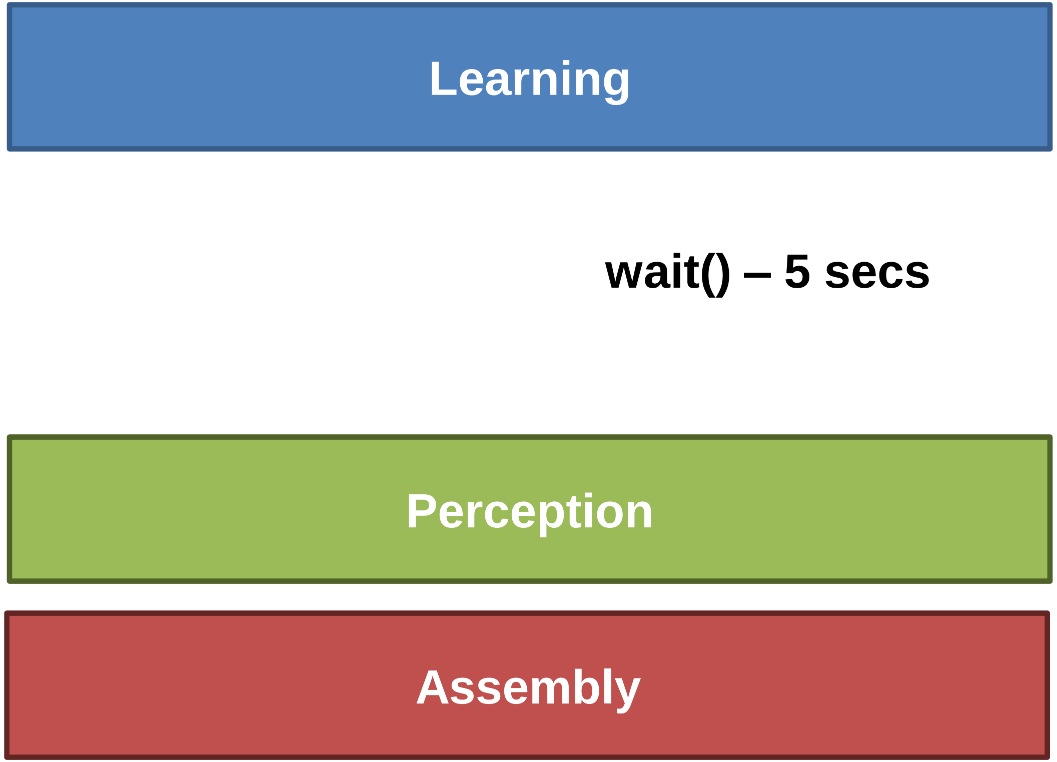
setConfig(next_config)

Perception

Assembly

Learning

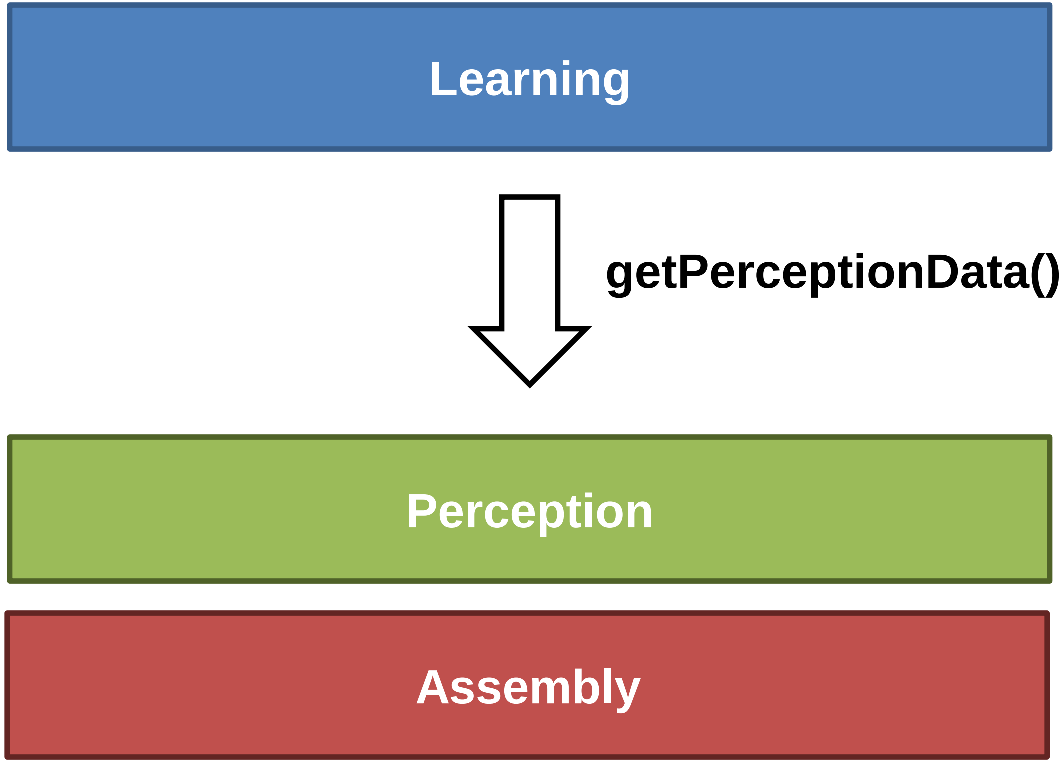
Learning - Exploration



wait() – 5 secs

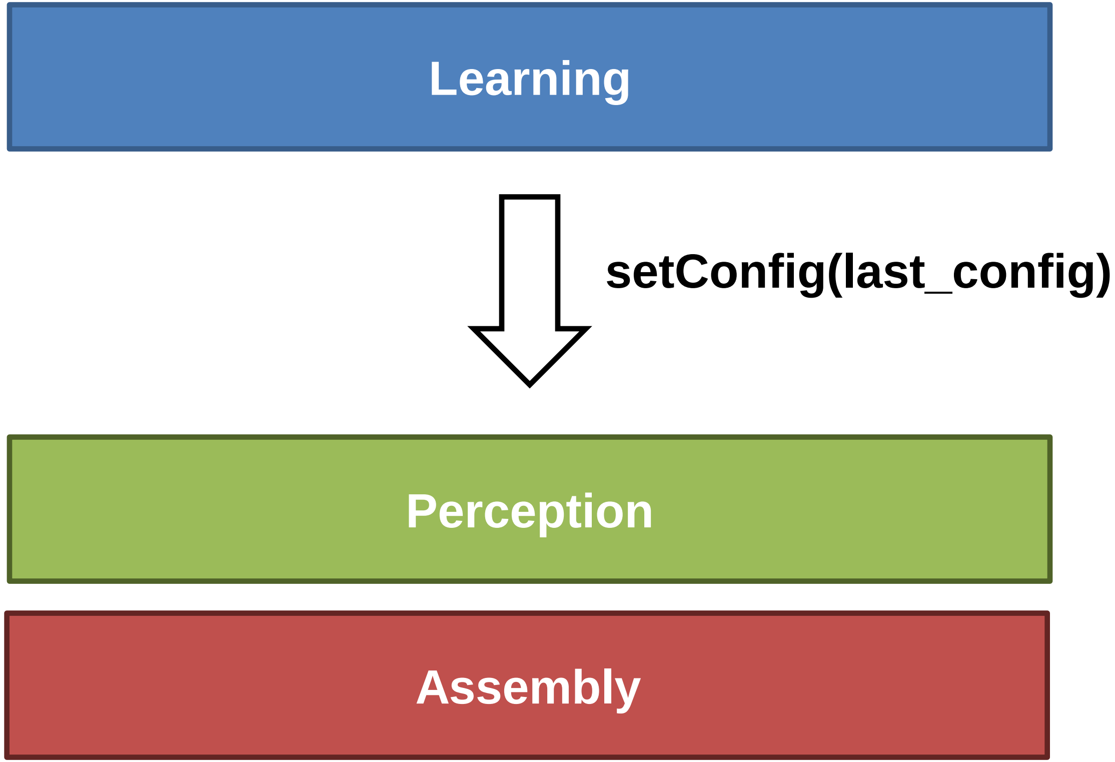
Learning

Learning - Exploration



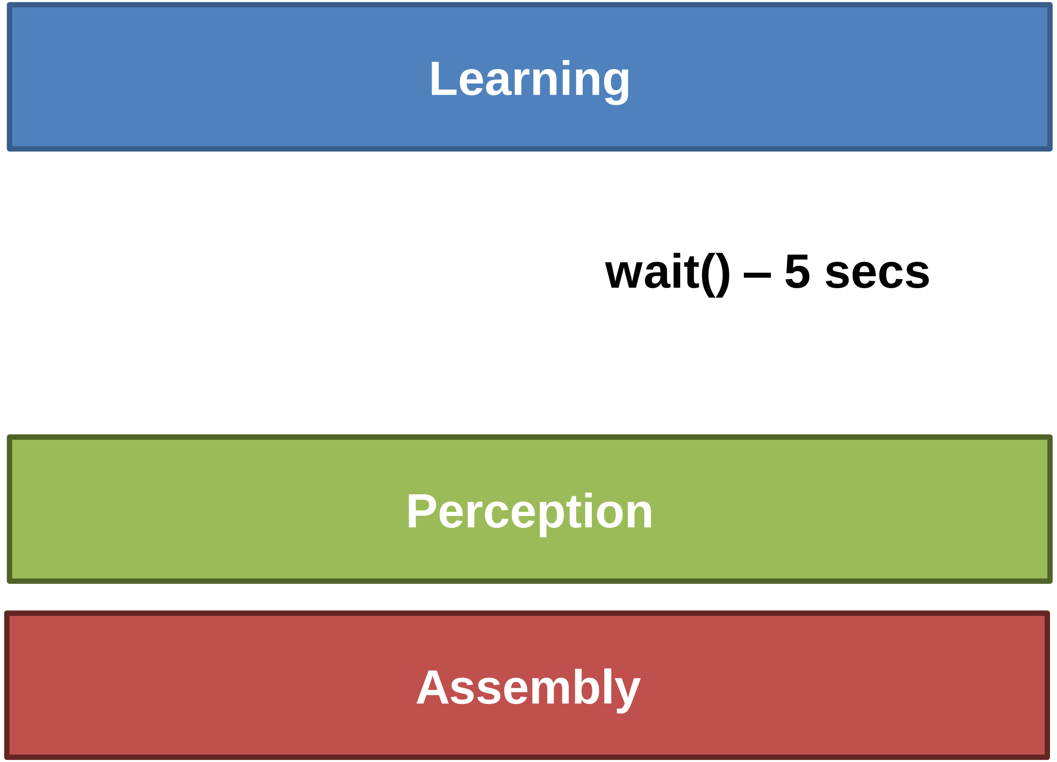
Learning

Learning - Exploration



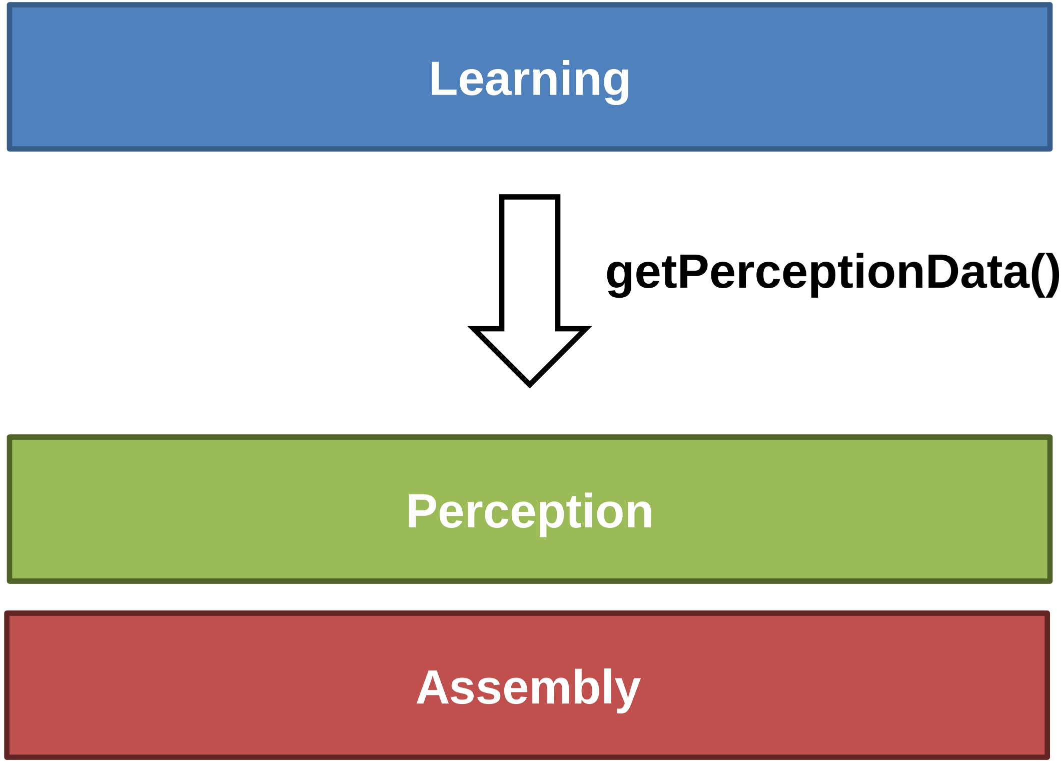
Learning

Learning - Exploration



Learning

Learning - Exploration



Learning

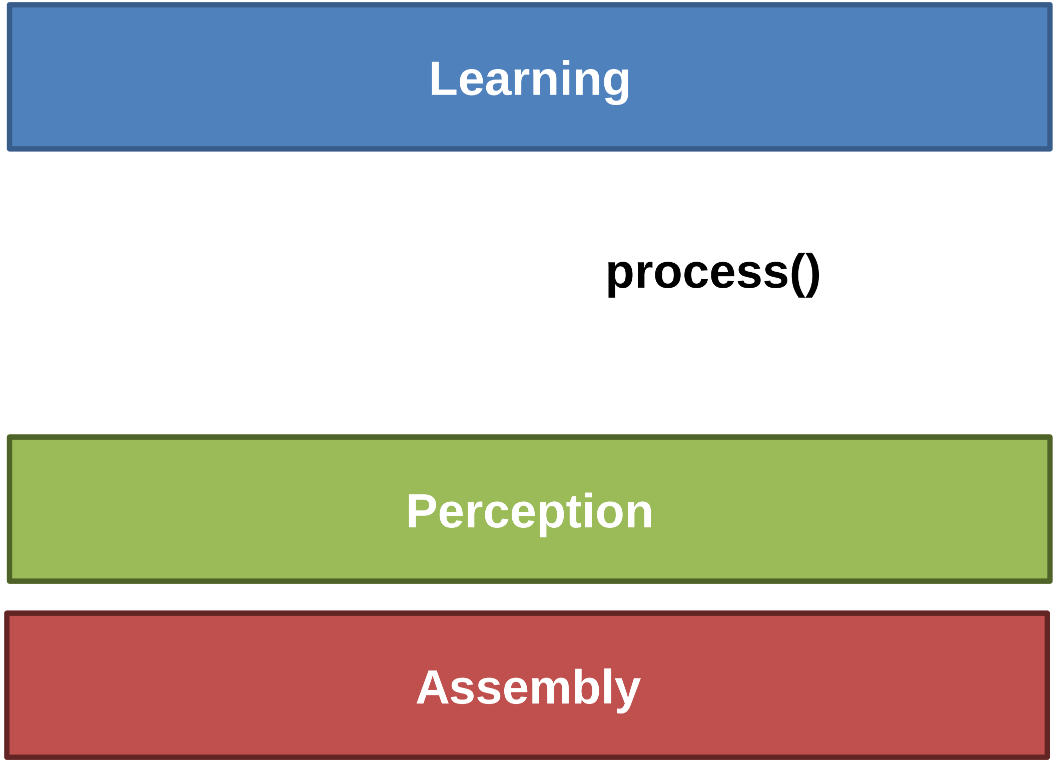
getPerceptionData()

Perception

Assembly

Learning

Learning - Exploration



Learning

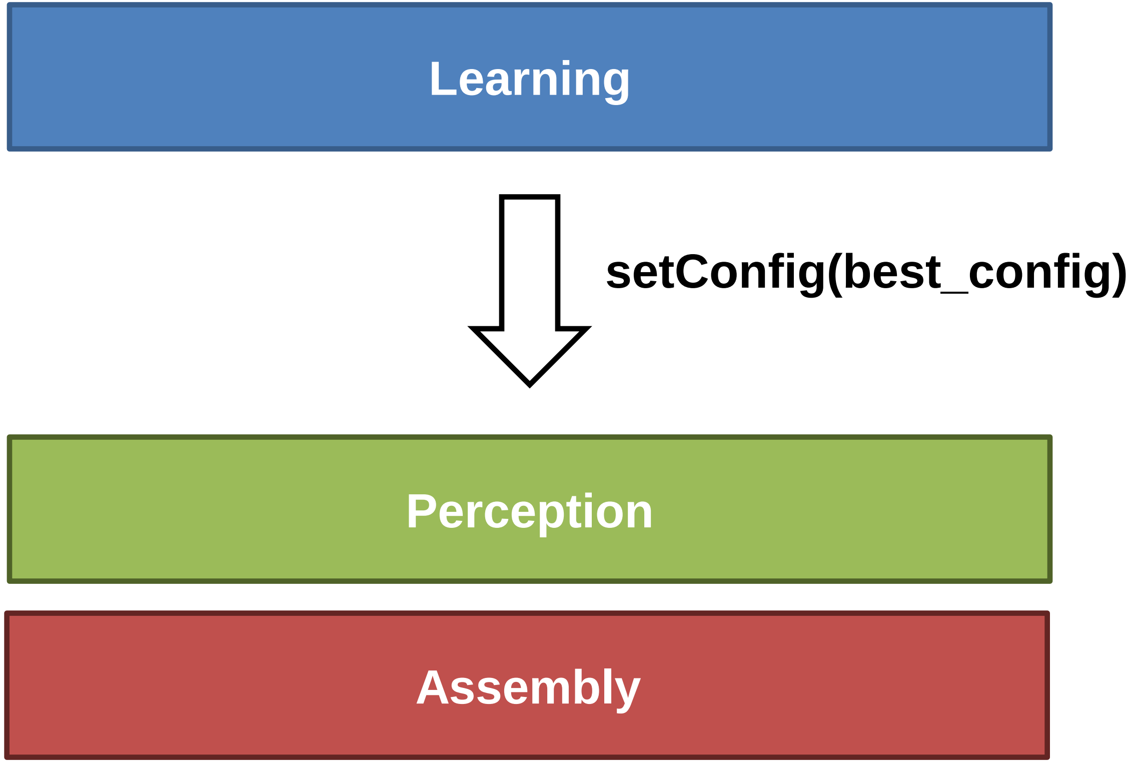
process()

Perception

Assembly

Learning

Learning - Exploitation



Learning

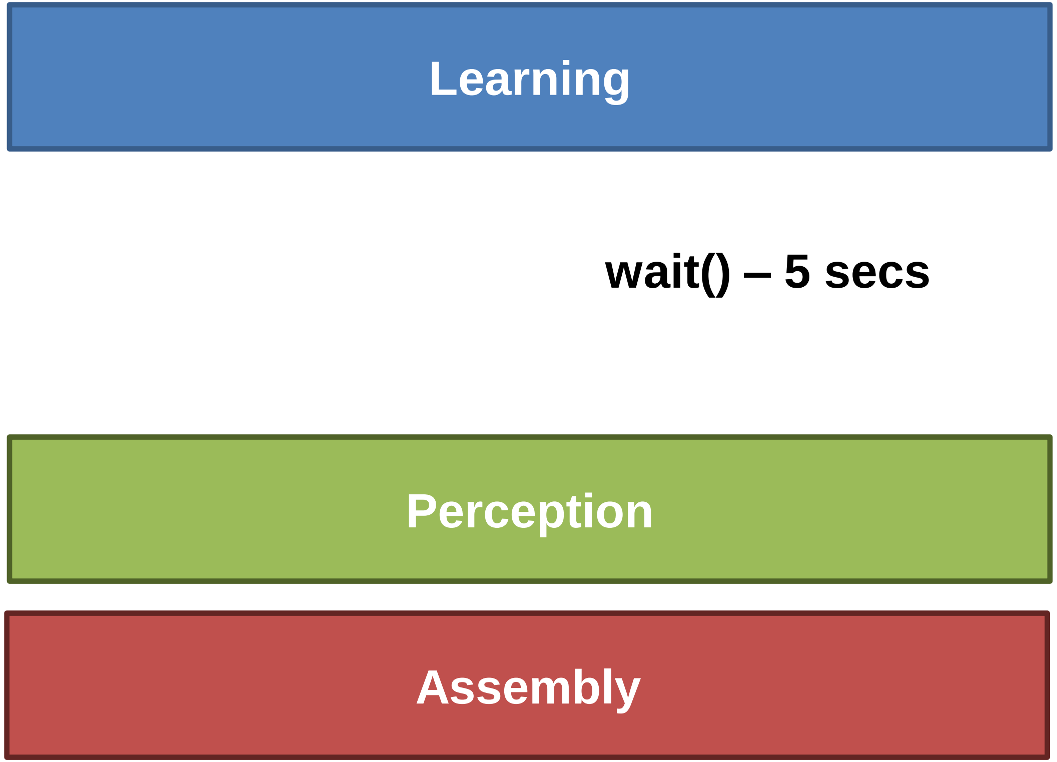
setConfig(best_config)

Perception

Assembly

Learning

Learning - Exploration



Learning

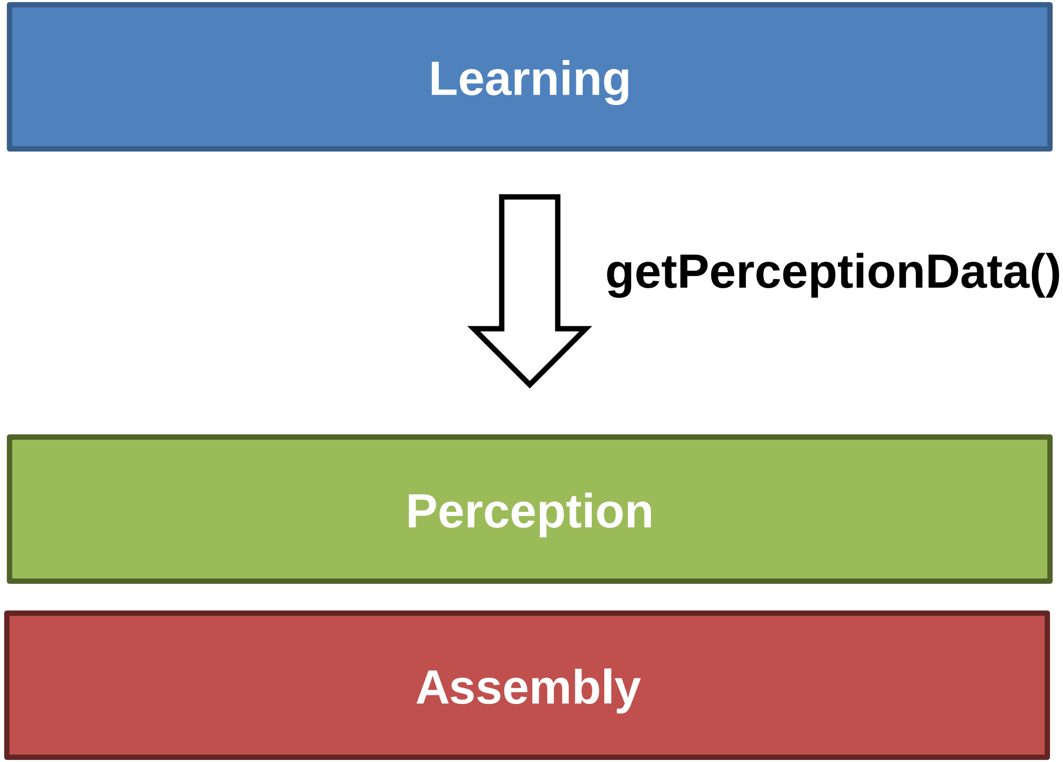
wait() – 5 secs

Perception

Assembly

Learning

Learning - Exploration



Learning

Learning - Exploration

Learning

process() – if nothing changed continue exploiting, otherwise explore.

Perception

Assembly

Summary

- PAL framework supports the concept of Emergent Systems;
- The framework is composed of the Assembly, Perception and Learning module;
- The Assembly module is responsible to compose functioning system and adapt them at runtime;
- The Perception module is responsible to monitor the system and provide real time data of the system to support learning;
- The Learning module uses the Assembly and Perception modules to learn which architectural composition maximises the level of satisfaction of the system goals.

Practical Assignment

For this practical assignment, we expect you to use the PAL framework provided to you on our github repository, and apply it on a Web Server software that is also on our github repo. Your goal is to define the metrics and events to feed the PAL framework, as well as to create different workload patterns (operating environments) where different compositions of the web server will be optimal.