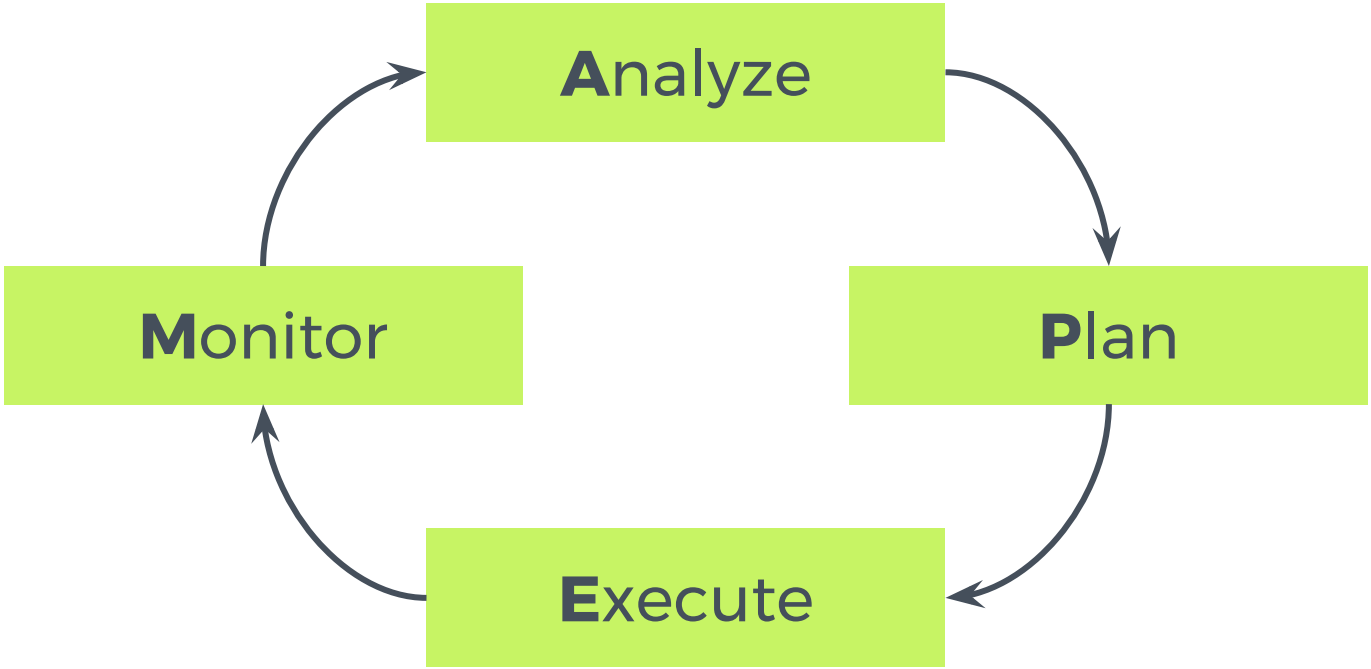
Nornir: A Customisable Framework for Autonomic and Power-Aware Applications

Daniele De Sensi,
Tiziano De Matteis, Marco Danelutto,

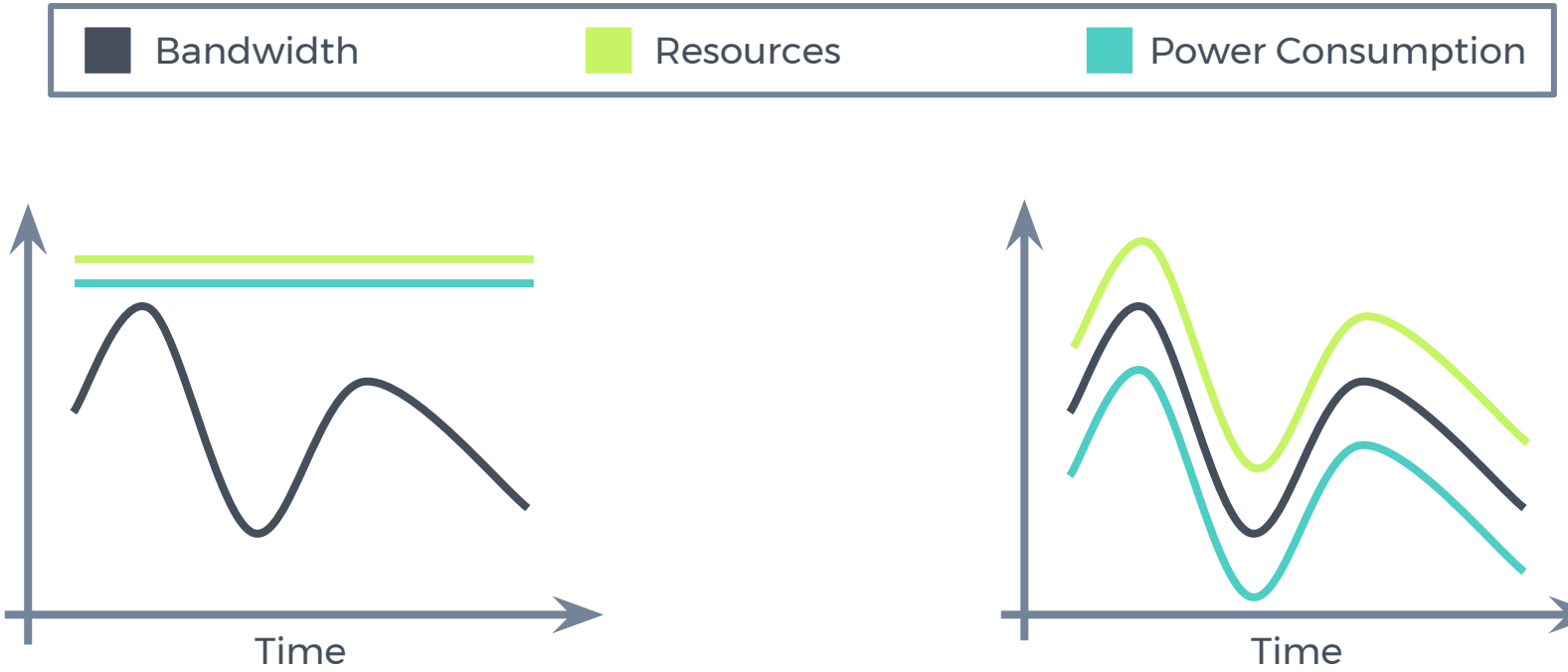
Computer Science Department,
University of Pisa, Italy

BACKGROUND & MOTIVATION

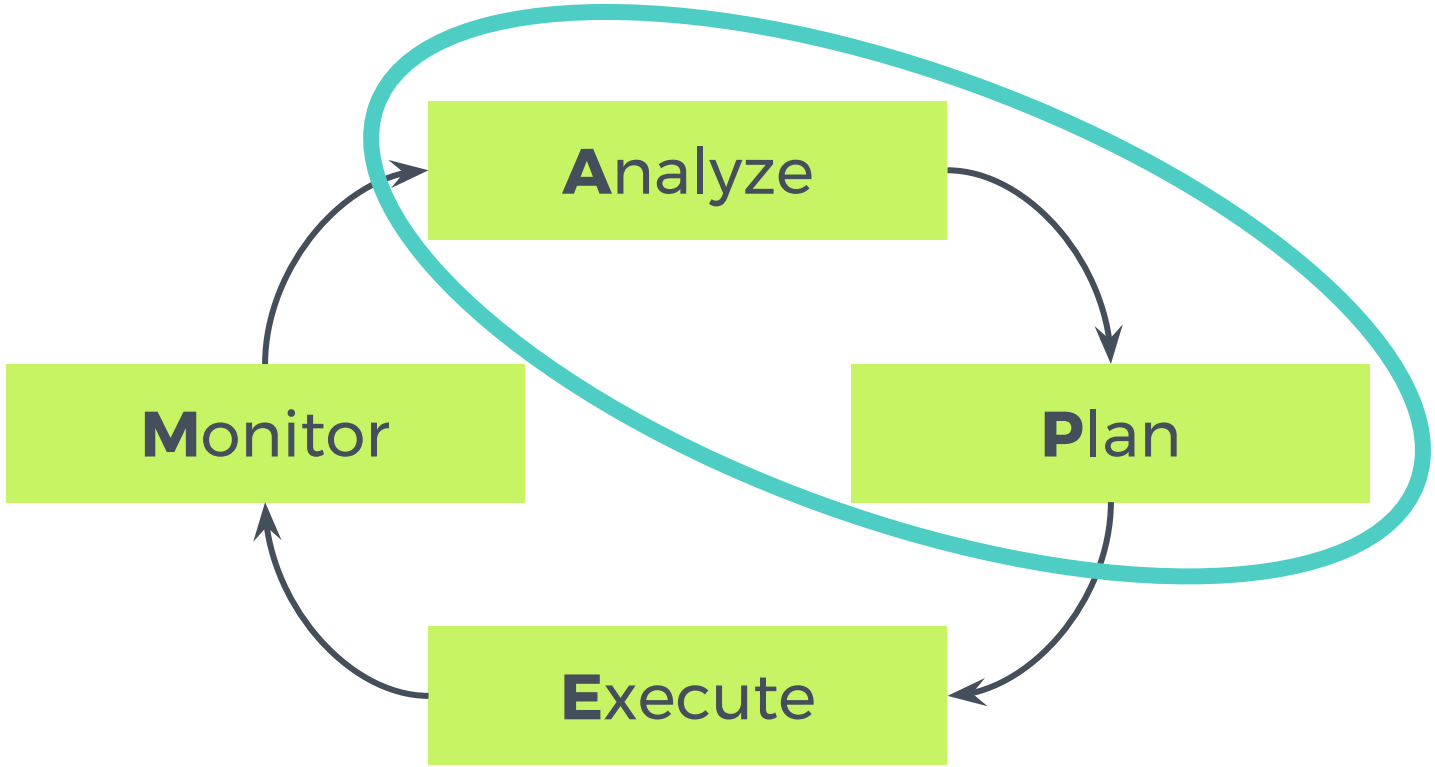
QUALITY OF SERVICE



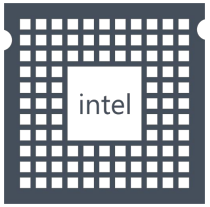
ELASTICITY

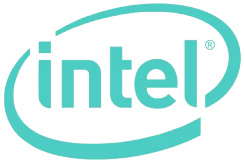


MAPE LOOP

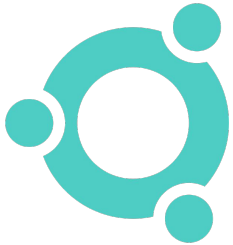


MONITORING AND ACTUATORS



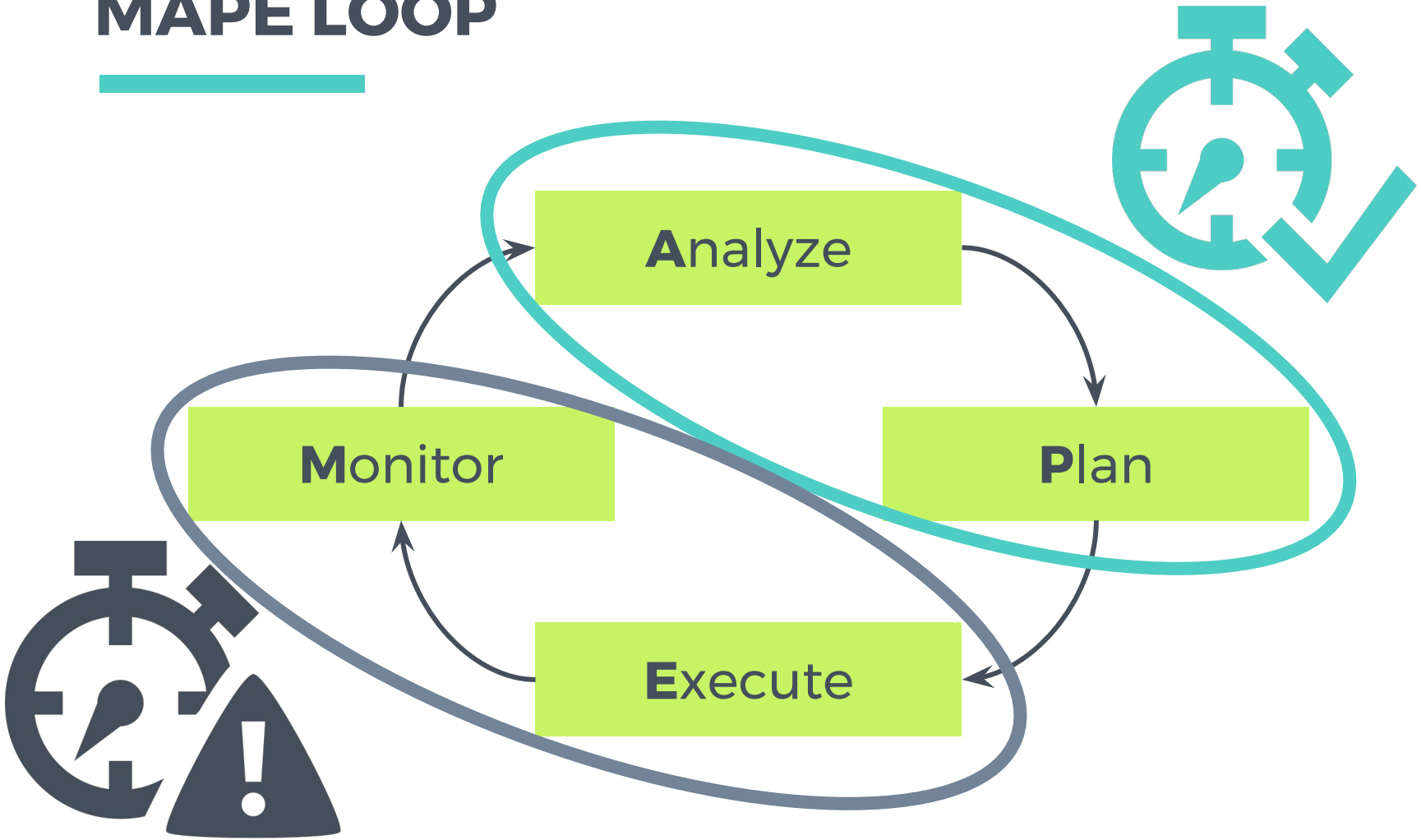


ARM



iOS

MAPE LOOP



PROBLEM

Many strategies are only **simulated** or hardwired on a **specific** application

Difficult to **compare** new strategies with old ones

TARGET

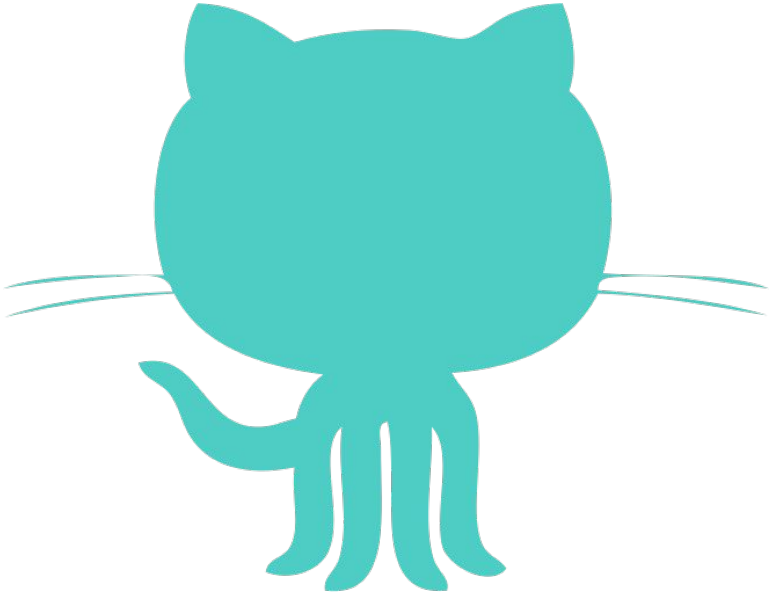
Provide a **customizable** C++ framework for building **autonomic** and **power-aware** reconfiguration algorithms on shared memory multicores

Designer focuses on the **algorithm**, exploiting monitoring infrastructure and executors provided by the framework

Easily **compare** new strategies with state of the art ones

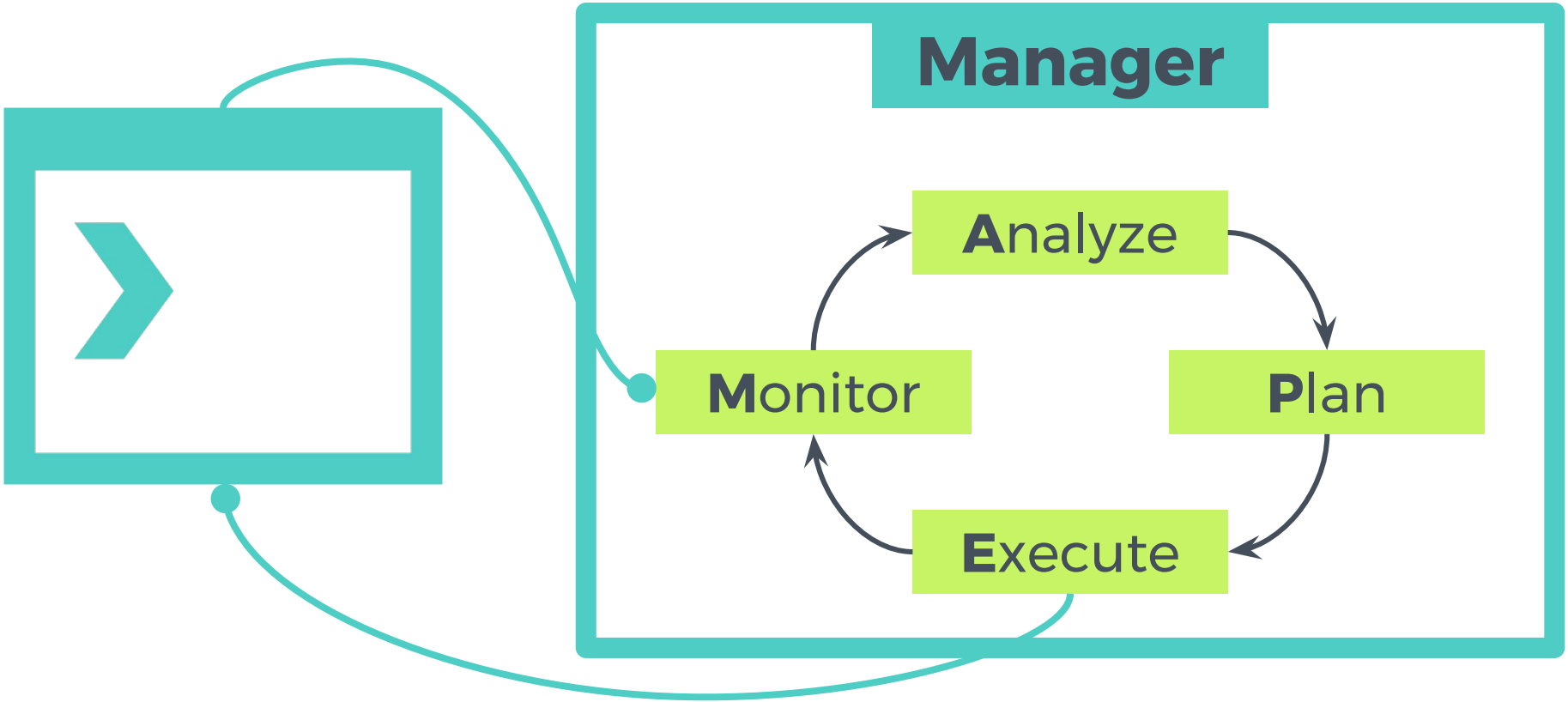
Wide set of applications provided as **testbed**

NORNIR

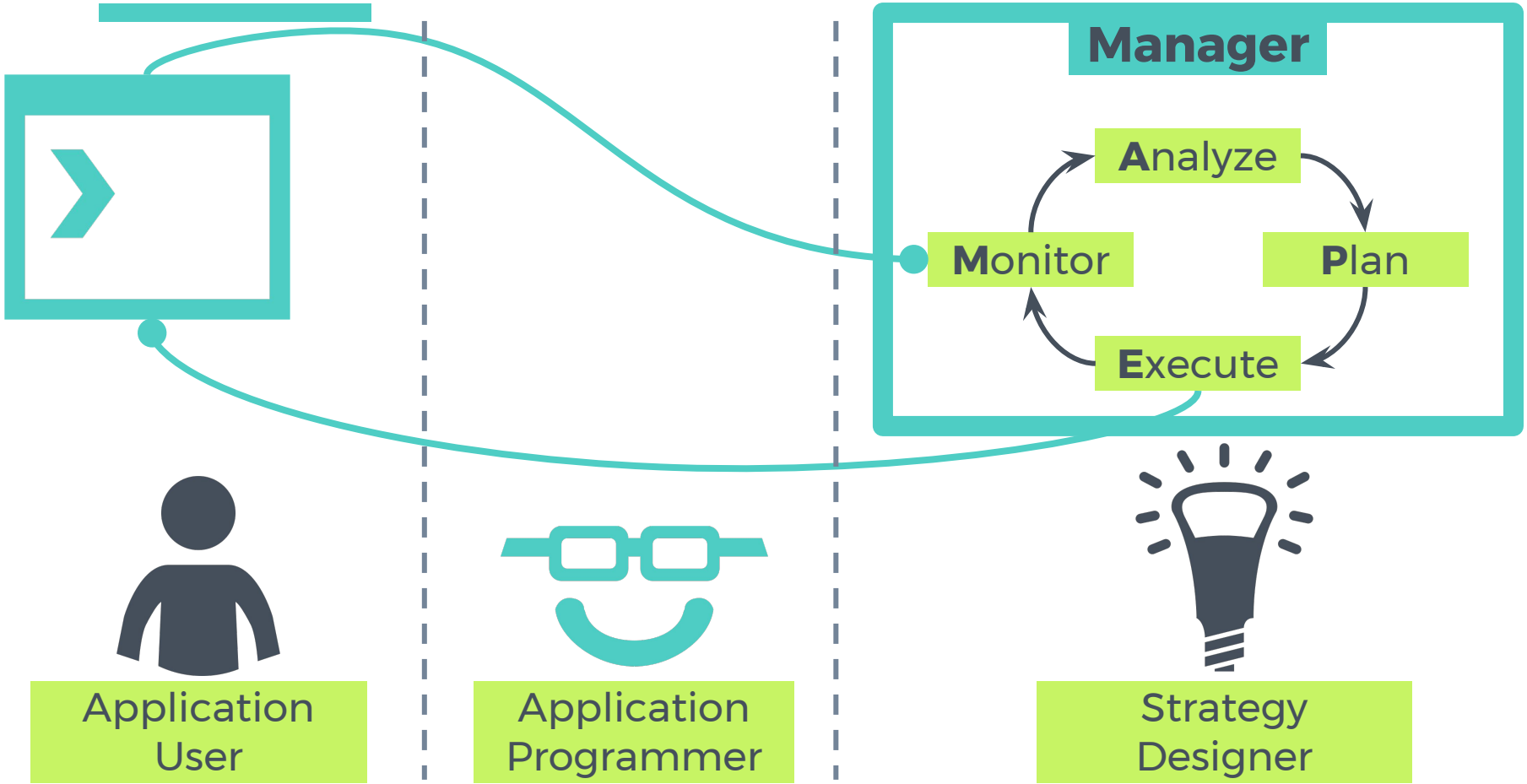


http://danieledesensi.github.io/nornir/

MAPE LOOP

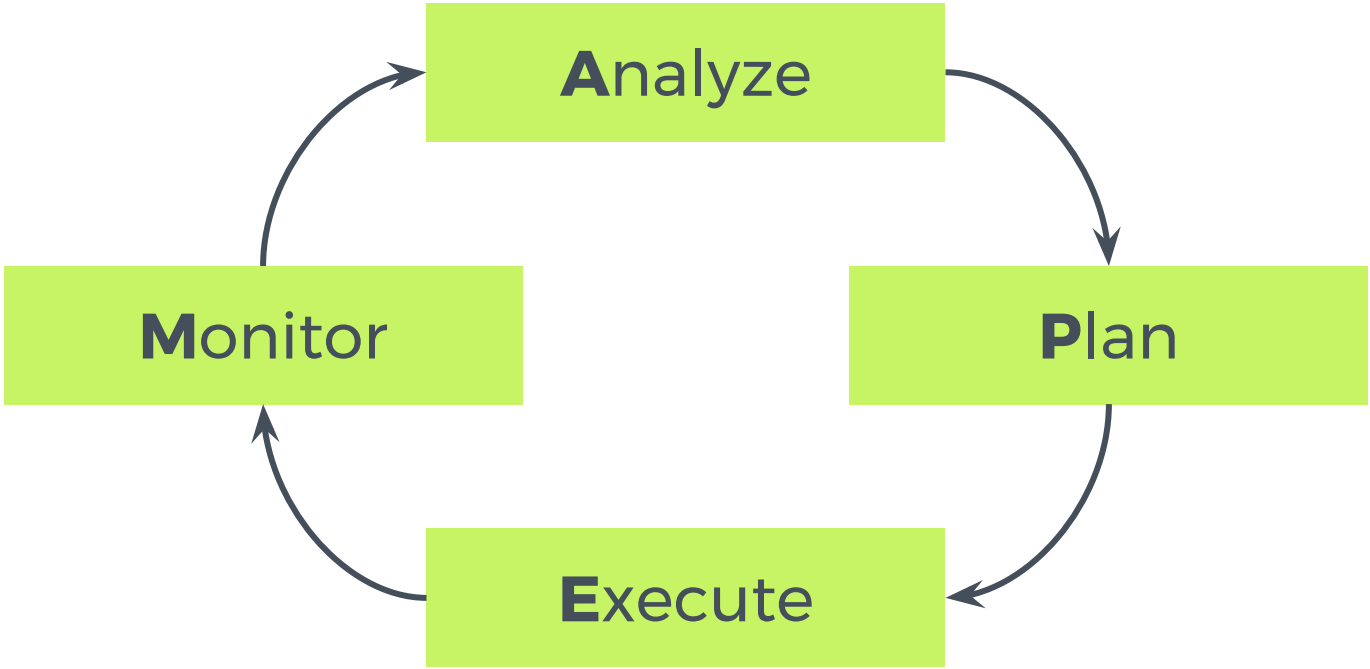


MAPE LOOP


Manager
Analyze
Monitor
Plan
Execute
Application User
Application Programmer
Strategy Designer

STRATEGY DESIGNER

MAPE LOOP



ANALYZE & PLAN

```
class SelectorDummy: public Selector{
    ...
};
```

ANALYZE & PLAN

```
class SelectorDummy: public Selector{
    ...
    KnobsValues getNextKnobsValues(){









    }
};
```

ANALYZE & PLAN

```
class SelectorDummy: public Selector{
    ...
    KnobsValues getNextKnobsValues(){

        if(_samples->average().latency < _p.requirements.latency){



        }else{



        }
        return k;
    }
};
```

ANALYZE & PLAN

```
class SelectorDummy: public Selector{
    ...
    KnobsValues getNextKnobsValues(){
        KnobsValues k(KNOB_VALUE_REAL);
        if(_samples->average().latency < _p.requirements.latency){
            k[KNOB_VIRTUAL_CORES] = 8;
            k[KNOB_FREQUENCY] = 1.2; // GHz
            ...
        }else{



        }
        return k;
    }
};
```

ANALYZE & PLAN

```
class SelectorDummy: public Selector{
    ...
    KnobsValues getNextKnobsValues(){
        KnobsValues k(KNOB_VALUE_REAL);
        if(_samples->average().latency < _p.requirements.latency){
            k[KNOB_VIRTUAL_CORES] = 8;
            k[KNOB_FREQUENCY] = 1.2; // GHz
            ...
        }else{
            k[KNOB_VIRTUAL_CORES] = 16;
            k[KNOB_FREQUENCY] = 2.4; // GHz
            ...
        }
        return k;
    }
};
```

ANALYZE & PLAN

```
class SelectorDummy: public Selector{
    ...
    KnobsValues getNextKnobsValues(){
        KnobsValues k(KNOB_VALUE_REAL);
        if(_samples->average().latency < _p.requirements.latency){
            k[KNOB_VIRTUAL_CORES] = 8;
            k[KNOB_FREQUENCY] = 1.2; // GHz
            ...
        }else{
            k[KNOB_VIRTUAL_CORES] = 16;
            k[KNOB_FREQUENCY] = 2.4; // GHz
            ...
        }
        return k;
    }
};
```

ANALYZE & PLAN

```
class SelectorDummy: public Selector{
    ...
    KnobsValues getNextKnobsValues(){
        KnobsValues k(KNOB_VALUE_RELATIVE);
        if(_samples->average().latency < _p.requirements.latency){
            k[KNOB_VIRTUAL_CORES] = 8;
            k[KNOB_FREQUENCY] = 1.2; // GHz
            ...
        }else{
            k[KNOB_VIRTUAL_CORES] = 16;
            k[KNOB_FREQUENCY] = 2.4; // GHz
            ...
        }
        return k;
    }
};
```

ANALYZE & PLAN

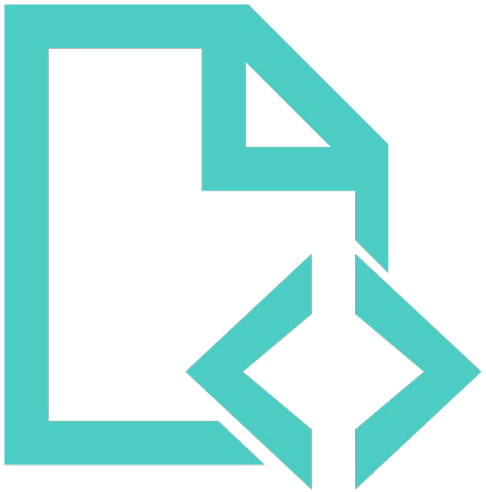
```
class SelectorDummy: public Selector{
    ...
    KnobsValues getNextKnobsValues(){
        KnobsValues k(KNOB_VALUE_RELATIVE);
        if(_samples->average().latency < _p.requirements.latency){
            k[KNOB_VIRTUAL_CORES] = 8;
            k[KNOB_FREQUENCY] = 1.2; // GHz
            ...
        }else{
            k[KNOB_VIRTUAL_CORES] = 16;
            k[KNOB_FREQUENCY] = 2.4; // GHz
            ...
        }
        return k;
    }
};
```

ANALYZE & PLAN

```
class SelectorDummy: public Selector{
    ...
    KnobsValues getNextKnobsValues(){
        KnobsValues k(KNOB_VALUE_RELATIVE);
        if(_samples->average().latency < _p.requirements.latency){
            k[KNOB_VIRTUAL_CORES] = 25; // %
            k[KNOB_FREQUENCY] = 25; // %
            ...
        }else{
            k[KNOB_VIRTUAL_CORES] = 75; // %
            k[KNOB_FREQUENCY] = 75; // %
            ...
        }
        return k;
    }
};
```

APPLICATION USER

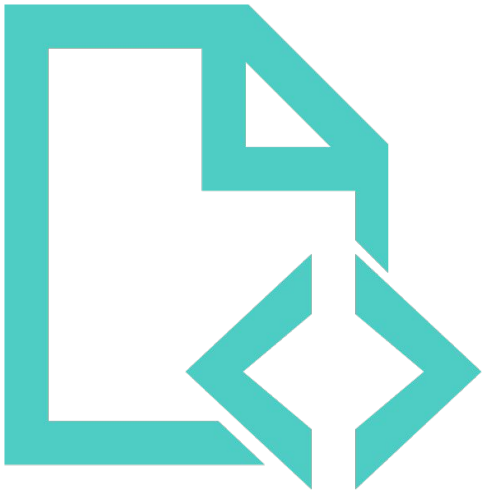
APPLICATION USER



Specifies requirements (QoS)

- Throughput
- Latency
- Completion Time (∞)
- Utilization Factor
- Power Consumption
- Energy (∞)

APPLICATION USER



Other Parameters

- Algorithm
- Control Step Length
- Actuators To Be Used
- ...

EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<nornirParameters>
   <requirements>
      <powerConsumption>50</powerConsumption>
      <throughput>MAX</throughput>
   </requirements>
   <controlStep>500</controlStep>
</nornirParameters>
```

DASHBOARD


denHome
Zoom Out
Last 1 hour
Uptime
11.65 days
Lab Power
144 watts
Lab Montly Cost Estimate
$ 10.65
Used Space
10.30 TiB
Total Space
14.30 TiB
Disk Health
OK
Plex Transcodes
0
ESXI CPU and Memory
min max avg current
1 3% 20% 7% 7%
2 2% 13% 5% 7%
3 4% 20% 7% 12%
4 3% 20% 6% 6%
5 3% 18% 6% 7%
6 2% 17% 6% 6%
Host Temps
min max avg current
CPU Temp 45 62 49 50
System Temp 34 36 35 36
Peripheral Temp 47 49 47 48
PCH Temp 49 52 51 52
VRM Temp 42 45 43 45
DIMMA1 Temp 34 36 35 36
Fan Speeds
Fan 2
Fan 3
Fan 4
Fan A
Internet Traffic
current
WAN Rx 3.1 Mbps
WAN Tx -73 kbps
DMZ Rx 10 kbps
DMZ Tx 0 bps
UPS Voltage/Load
Load
Voltage
min max avg current
Load (Watts) 72 W 99 W 74 W 72 W
Input Voltage (right-y) 119 V 123 V 121 V 121 V
Esxi Load (Watts) 63 W 108 W 70 W 72 W
Esxi Input Voltage (right-y) 119 V 123 V 121 V 121 V
UPS Capacity/Load/Runtime
Load
Voltage
min max avg current
Capacity % 100.00% 100.00% 100.00% 100.00%
Runtime (right-y) 1.02 hour 1.68 hour 1.50 hour 1.63 hour
Load % 8.00% 11.00% 8.22% 8.00%
Esxi Capacity % 100.00% 100.00% 100.00% 100.00%
Docker Stats CPU
current
Plex 5%
Sonarr 0%
Couchpotato 0%
NZBGet 0%
Grafana 0%
Muximux 0%
InfluxDB 0%
Docker Stats Memory
current
Total 81%
Plex 59%
Sonarr 6%
Couchpotato 1%
NZBGet 11%
Grafana 0%
Muximux 0%
Synology DS1512+
Metric Min Max Avg Current
Disk_1 31 °C 32 °C 32 °C 32 °C
Disk_2 31 °C 32 °C 31 °C 32 °C
Disk_3 31 °C 32 °C 31 °C 32 °C
Disk_4 31 °C 32 °C 31 °C 32 °C
Disk_5 30 °C 32 °C 31 °C 32 °C
Synology DX513+
Metric Min Max Avg Current
Disk_1_(DX513-1) 33 °C 34 °C 33 °C 34 °C
Disk_2_(DX513-1) 33 °C 34 °C 33 °C 34 °C
Disk_3_(DX513-1) 33 °C 34 °C 33 °C 34 °C
Disk_4_(DX513-1) 32 °C 33 °C 32 °C 33 °C

APPLICATION PROGRAMMER

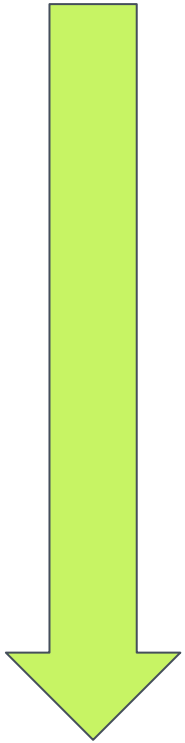
ALTERNATIVES

Black-Box

Instrumentation

Runtime Interaction

Nornir's Programming Interface



More Programming Effort
More Control (Better Solutions)

INSTRUMENTATION

Already existing application, which can be modified and recompiled

```
$ ./manager-external
```

INSTRUMENTATION

Already existing application, written with a generic programming framework

```
StreamElement* s;
while(s = receive()){
    process(s);
}
```

INSTRUMENTATION

Already existing application, written with a generic programming framework

```
#include <nornir.hpp>
nornir::Monitor r("parameters.xml");
StreamElement* s;
while(s = receive()){
    process(s);
}
```

INSTRUMENTATION

Already existing application, written with a generic programming framework

```
#include <nornir.hpp>
nornir::Monitor r("parameters.xml");
StreamElement* s;
while(s = receive()){
    r.begin();
    process(s);
    r.end();
}
```

INSTRUMENTATION

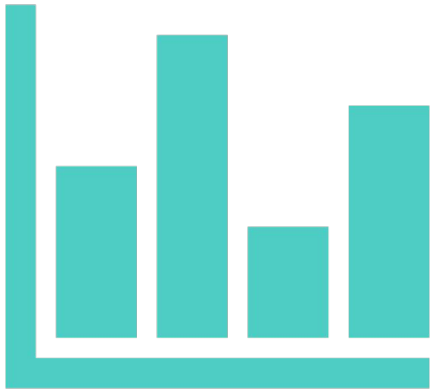
Already existing application, written with a generic programming framework

```
#include <nornir.hpp>
nornir::Monitor r("parameters.xml");
StreamElement* s;
while(s = receive()){
    r.begin();
    process(s);
    r.end();
}
r.terminate();
```

TESTBED

To have a **testbed** for new reconfiguration strategies, we instrumented all the applications in the **PARSEC** benchmark

EVALUATION

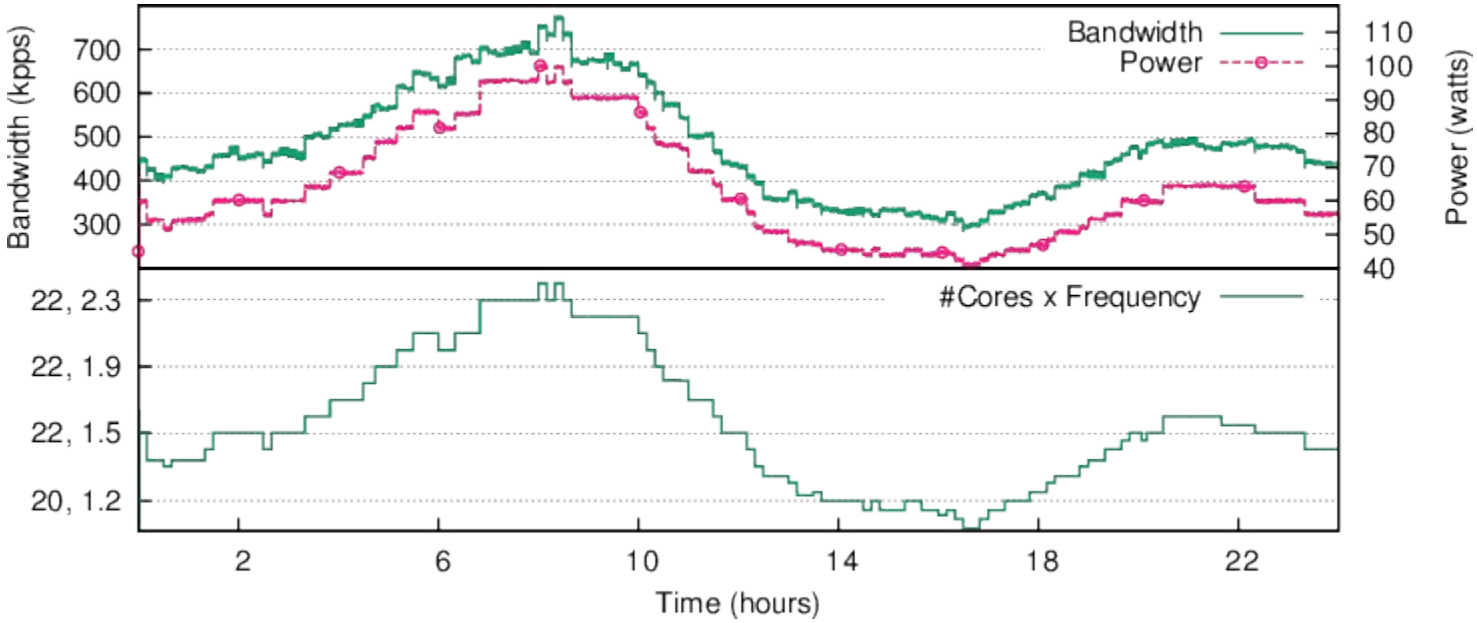


FLEXIBILITY

Currently available state of the art strategies:

- 2 **online learning** algorithms to enforce requirements on throughput and power consumption
- 1 **online+offline** algorithm
- 1 **heuristic** to enforce throughput requirements
- 2 **heuristics** to enforce throughput and utilization requirements

RUNNING EXAMPLE



CONCLUSIONS & FUTURE WORK

CONCLUSIONS

We designed and implemented a flexible and customizable framework for:

- Implementing **autonomic** and **power-aware** reconfiguration strategies with low programming effort
- **Validating** them on a wide set of real applications
- **Comparing** them with state-of-the art existing strategies

FUTURE WORK

Extend to **multiple** operators/applications

Support **distributed memory** architectures

Introduce a **plug-in** system for new strategies

Thanks for your attention

http://danieledesensi.github.io/nornir/

Daniele De Sensi

Tiziano De Matteis

Marco Danelutto

Backup Slides

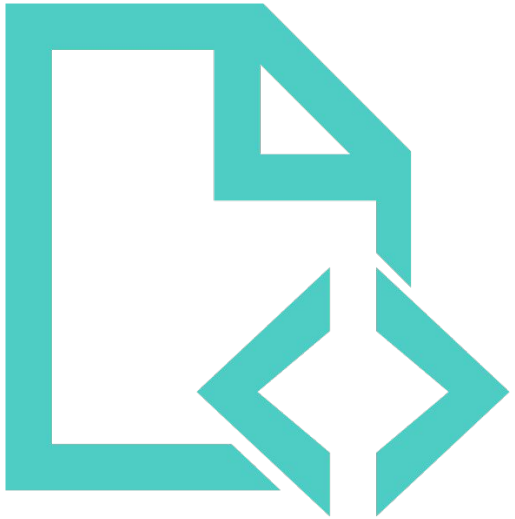
ACTORS

Application User

Application Programmer

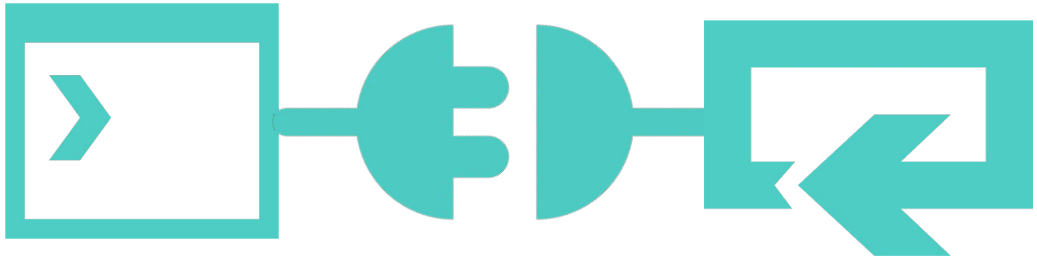
Strategy Designer

APPLICATION USER



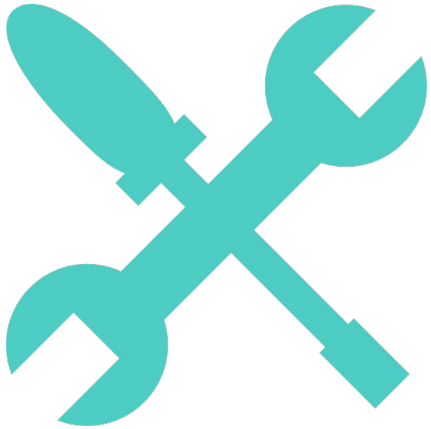
Specifies
requirements (QoS)

APPLICATION PROGRAMMER



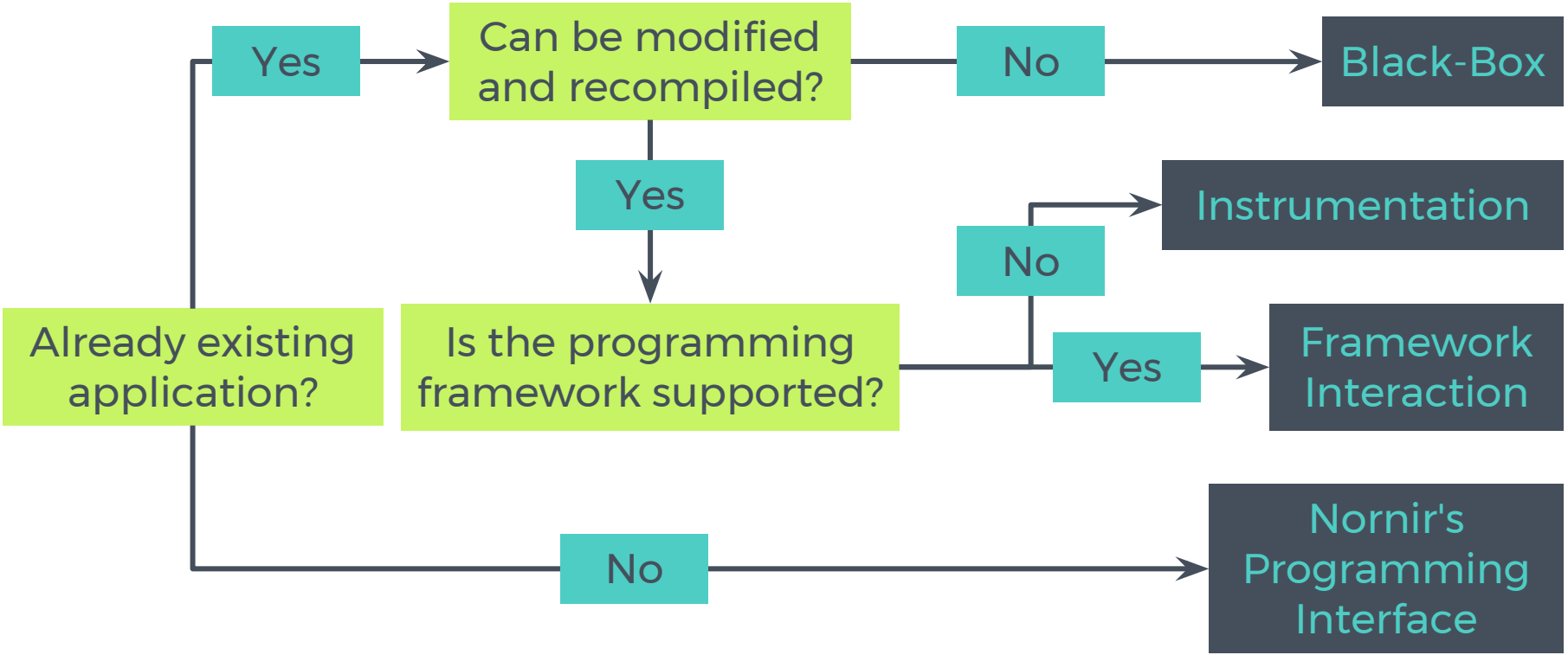
Connects application to manager

STRATEGY DESIGNER



Customizes existing strategies or **creates** new ones

ALTERNATIVES



MAPE LOOP

```
class Manager{
    ...
    void run(){
        while(isRunning()){
            sleep(_parameters.samplingInterval);









        }
    }
};
```

MONITOR

```
class Manager{
    ...
    void run(){
        while(isRunning()){
            sleep(_parameters.samplingInterval);
            // Monitor
            ApplicationSample s = getSample();
            storeSample(s);

        }
    }
};
```

Can be customized to collect framework/application specific information

ANALYZE & PLAN

```
class Manager{
    ...
    void run(){
        while(isRunning()){
            sleep(_parameters.samplingInterval);
            // Monitor
            ApplicationSample s = getSample();
            storeSample(s);

            // Analyze & Plan
            KnobsValues k = _selector->getNextKnobsValues();



        }
    }
};
```

EXECUTE

```
class Manager{
    ...
    void run(){
        while(isRunning()){
            sleep(_parameters.samplingInterval);
            // Monitor
            ApplicationSample s = getSample();
            storeSample(s);

            // Analyze & Plan
            KnobsValues k = _selector->getNextKnobsValues();

            for(uint i = 0; i < KNOB_NUM; i++){
                _knobs[i]->changeValue(k[i]); // Execute
            }
        }
    }
};
```

MAPE LOOP

```
class Manager{
    ...
    void run(){
        while(isRunning()){
            sleep(_parameters.samplingInterval);
            // Monitor
            ApplicationSample s = getSample();
            storeSample(s);

            // Analyze & Plan
            KnobsValues k = _selector->getNextKnobsValues();

            for(uint i = 0; i < KNOB_NUM; i++){
                _knobs[i]->changeValue(k[i]); // Execute
            }
        }
    }
};
```

```
class DummyKnob: public Knob{
    void changeValue(double v){
        ;
    }
};
```

COLOR SCHEME

C7F464

4ECDC4

738498

454F5B