Continuous reasoning over the Cloud-IoT continuum

Antonio Brogi, Stefano Forti

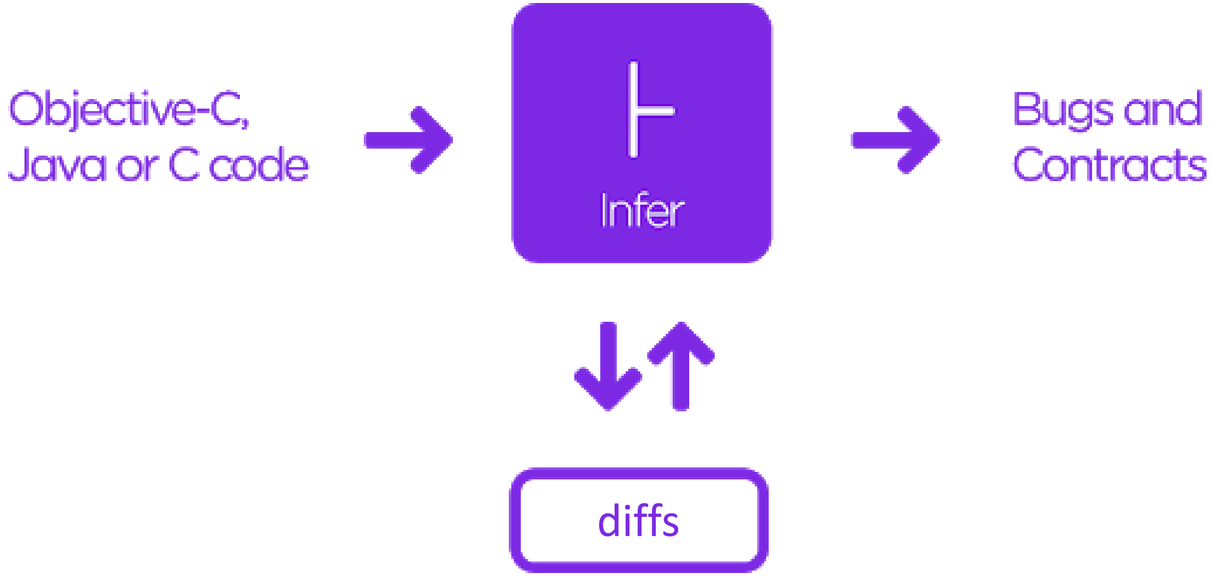
Department of Computer Science

University of Pisa

Continuous Reasoning

Exploit compositionality to differentially analyse a large-scale system:
- by mainly **focussing on the latest changes** introduced in the system, and
- by **re-using previously computed results** as much as possible

- Successful in supporting iterative software development at large IT companies, e.g. FB Infer



Separation logic

Extension of Hoare logic

$$\{precondition\}code\ \{postcondition\}$$

to model in-place update of memory during execution in terms of preconditions and postconditions on the heap

$$
\begin{array}{l}
\{x \mapsto 0 * y \mapsto 0\} \\
\quad [x] = y; \\
\quad [y] = x \\
\{x \mapsto y * y \mapsto x\}
\end{array}
$$

Concurrent separation logic for modular reasoning about threads that share storage and other resources

Continuos reasoning for application placement

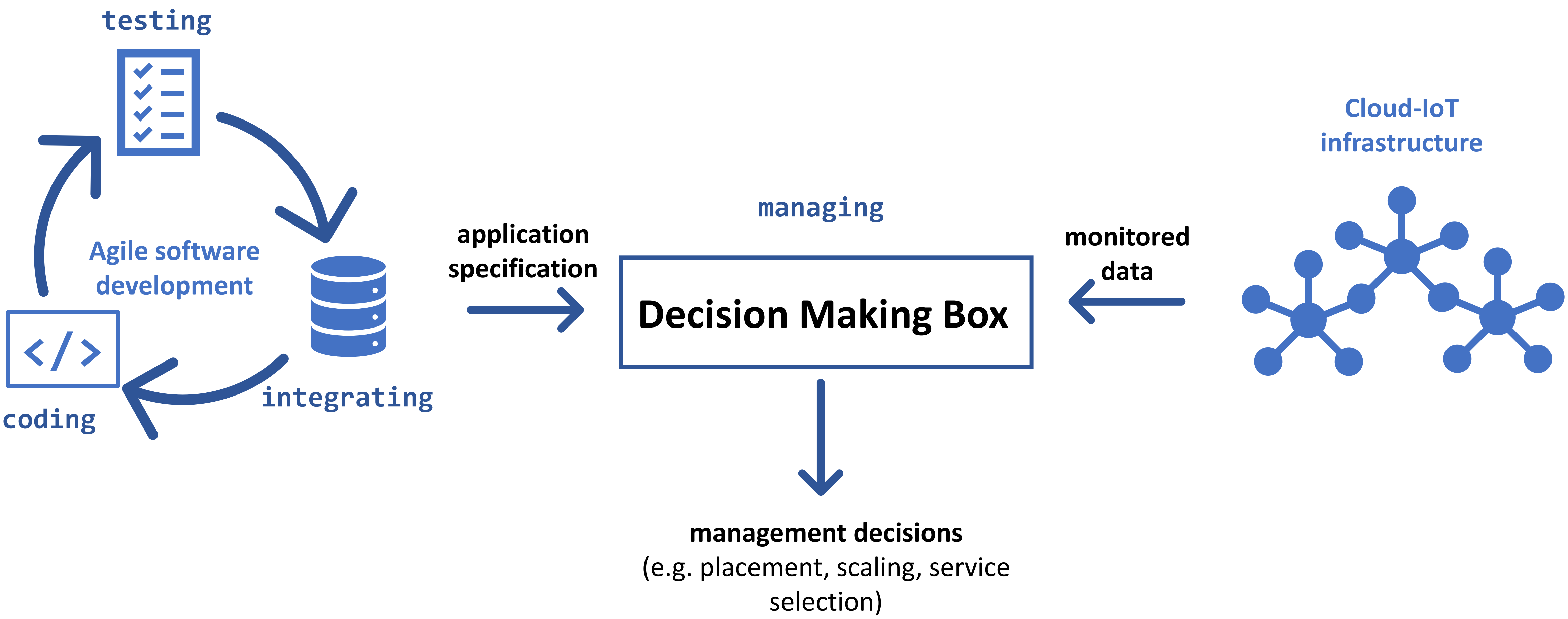
What for?

- **Scale** to larger instances of the placement problem
- **Reduce time needed** to make placement decisions at runtime (faster reaction times!)
- Possibly **reduce** the number of **management operations** (stop, undeploy, deploy, start)

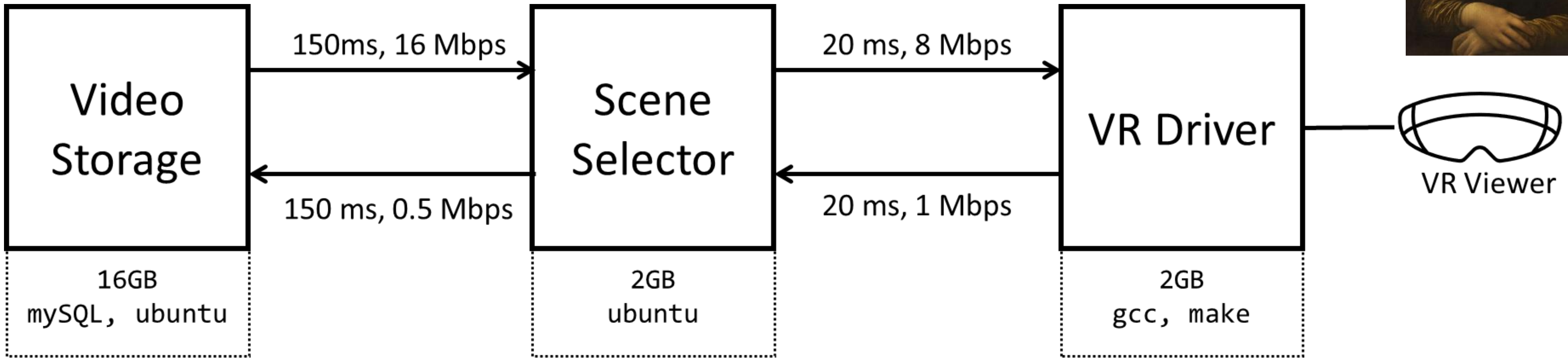
How?

- By trying to **re-place only** those **services affected by**
  - **infrastructure changes** (e.g. node crash, degraded network QoS between communicating services)
  - **changes from CI/CD pipeline** (e.g. addition/removal of services, updated requirements)

The Big Picture

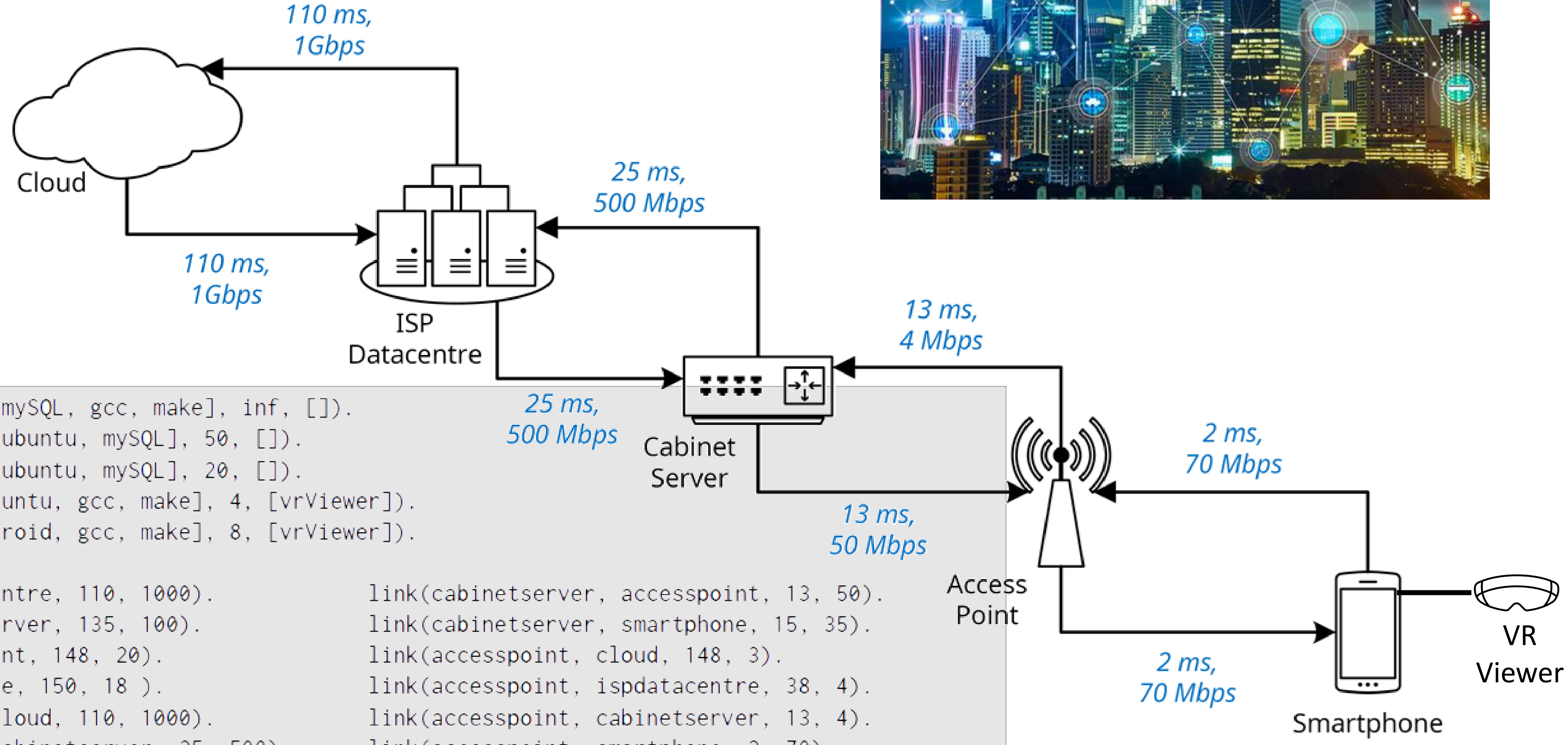


A VR Application

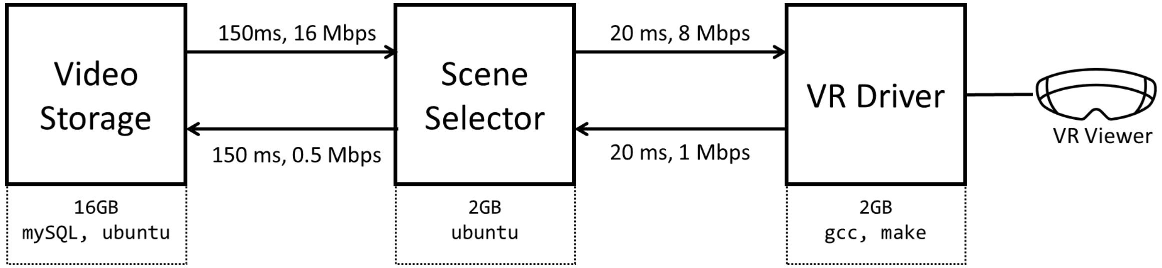


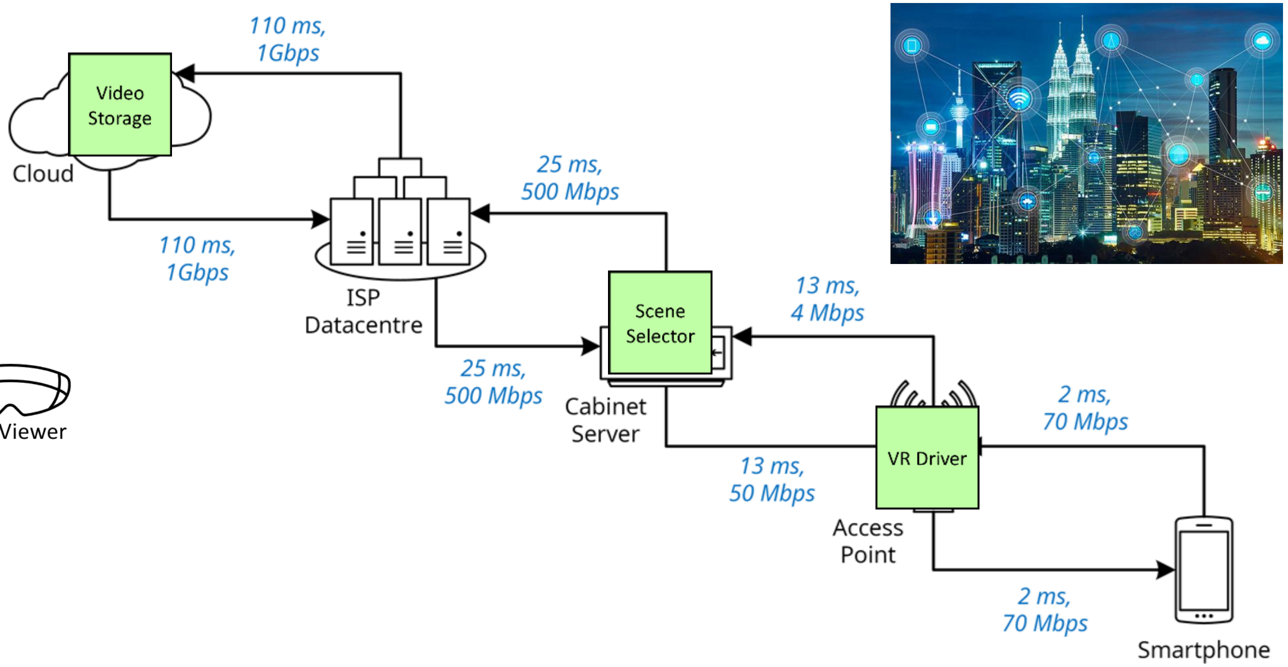

```
application(vrApp, [videoStorage, sceneSelector, vrDriver]).
service(videoStorage, [mySQL, ubuntu], 16, []).
service(sceneSelector, [ubuntu], 2, []).
service(vrDriver, [gcc, make], 2, [vrViewer]).
s2s(videoStorage, sceneSelector, 150, 16).
s2s(sceneSelector, videoStorage, 150, 0.5).
s2s(sceneSelector, vrDriver, 20, 8).
s2s(vrDriver, sceneSelector, 20, 1).
```

A Cloud-IoT Infrastructure

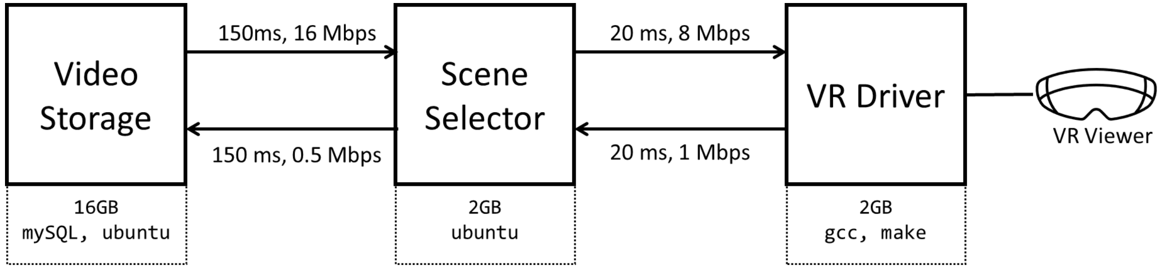


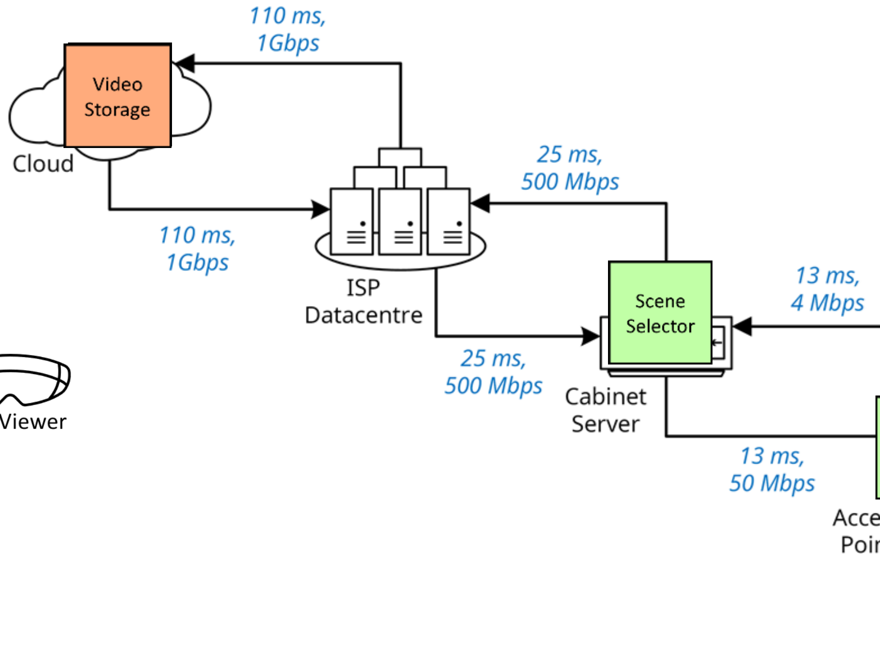

```
node(cloud, [ubuntu, mySQL, gcc, make], inf, []).
node(ispdatacentre, [ubuntu, mySQL], 50, []).
node(cabinetserver, [ubuntu, mySQL], 20, []).
node(accesspoint, [ubuntu, gcc, make], 4, [vrViewer]).
node(smartphone, [android, gcc, make], 8, [vrViewer]).

link(cloud, ispdatacentre, 110, 1000).
link(cloud, cabinetserver, 135, 100).
link(cloud, accesspoint, 148, 20).
link(cloud, smartphone, 150, 18 ).
link(ispdatacentre, cloud, 110, 1000).
link(ispdatacentre, cabinetserver, 25, 500).
link(ispdatacentre, accesspoint, 38, 50).
link(ispdatacentre, smartphone, 40, 35).
link(cabinetserver, cloud, 135, 100).
link(cabinetserver, ispdatacentre, 25, 500).
link(cabinetserver, accesspoint, 13, 50).
link(cabinetserver, smartphone, 15, 35).
link(accesspoint, cloud, 148, 3).
link(accesspoint, ispdatacentre, 38, 4).
link(accesspoint, cabinetserver, 13, 4).
link(accesspoint, smartphone, 2, 70).
link(smartphone, cloud, 150, 2).
link(smartphone, ispdatacentre, 40, 2.5).
link(smartphone, cabinetserver, 15, 3).
link(smartphone, accesspoint, 2, 70).
```






```
1 ?- cr(vrApp,P).
P = [on(vrDriver, accesspoint), on(sceneSelector, cabinetserver), on(videoStorage, cloud)]
```





```
2 ?- cr(vrApp,NewP).
```

- detects that `videoStorage` needs to be migrated
- builds partially ground query to determine new placement `NewP` to migrate `videoStorage` while keeping the rest of the placement as is

```
NewP = [on(vrDriver, accesspoint), on(sceneSelector, cabinetserver), on(videoStorage, ispdatacentre)]
```

Key ideas of `cr/2`: When A is not deployed

```
cr(A,Placement) :- \+ deployment(A,_,_), placement(A,Placement).

placement(A,P) :-
   application(A,Services),
   InitP=[], InitAlloc=([],[]), placement(Services, InitP, InitAlloc, P),
   allocatedResources(P,Alloc), assert(deployment(A,P,Alloc)).

placement([S|Ss],P,(AllocHW,AllocBW),Placement) :-
    nodeOk(S,N,P,AllocHW),       % checks SW, IoT and cumulative hw requirements
    linksOk(S,N,P,AllocBW),      % checks latency and cumulative BW requirements
    placement(Ss,[on(S,N)|P],(AllocHW,AllocBW),Placement).
placement([],P,_,P).

nodeOk(S,N,P,AllocHW) :-
    service(S,SWReqs,HWReqs,IoTReqs),
    node(N,SWCaps,HWCaps,IoTCaps),
    swReqsOk(SWReqs,SWCaps),
    thingReqsOk(IoTReqs,IoTCaps),
    hwOk(S,N,HWCaps,HWReqs,P,AllocHW). % checks cumulative hw requirements on N
```

Key ideas of `cr/2`: When A is already deployed

First try re-placing only “what needs to be re-placed”. Ow, re-place everything.

```
cr(A,NewPlacement) :-
    deployment(A,P,Alloc),
    newServices(P,NewServices),
    crStep(P,Alloc,ServicesToMove,StablePlacement),
    append(NewServices,ServicesToMove,ServicesToPlace),
    placement(ServicesToPlace,StablePlacement,Alloc,NewPlacement),
    allocatedResources(NewPlacement,NewAlloc),
    retract(deployment(A,_,_)), assert(deployment(A,NewPlacement,NewAlloc)).
cr(A, NewPlacement) :-
    deployment(A,_,Alloc),
    application(A, Services),
    InitPlacement=[], placement(Services,InitPlacement,Alloc,NewPlacement),
    allocatedResources(NewPlacement,NewAlloc),
    retract(deployment(A,_,_)), assert(deployment(A,NewPlacement,NewAlloc)).
```

Your turn now ☺

Try to define the predicate

```
crStep(P, Alloc, ServicesToMove, StablePlacement) :- …
```

which:

- given the current placement **P** and the corresponding allocated resources **Alloc**,
- determines the list **ServicesToMove** of the services that need to be re-placed and the partial placement **StablePlacement** that can be kept as is.

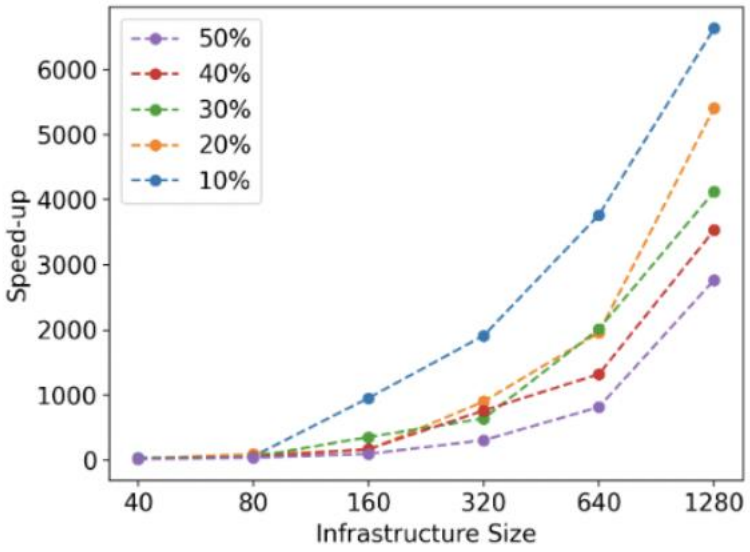
Recall that placement P is a list of the form

```
[on(s1,n1), on(s2,n1), on(s3,n2),…]
```

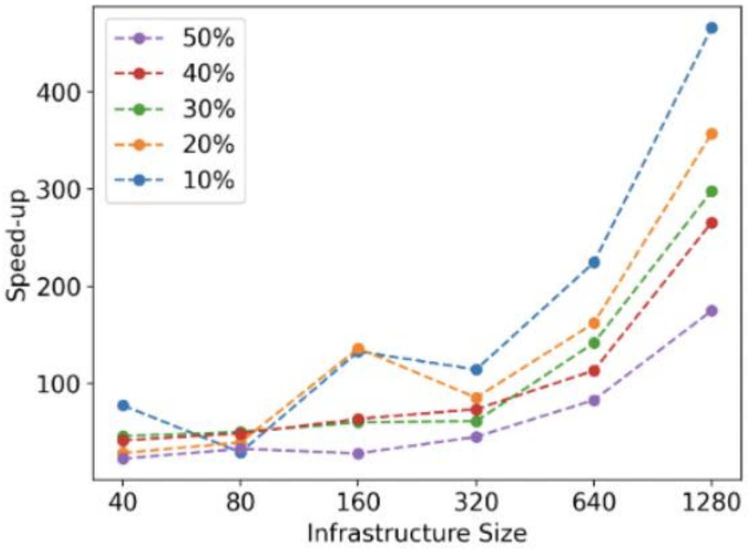
Continuous Reasoning Step (Solution)

```
crStep([on(S,_)|Ps],(AllocHW,AllocBW),ServicesToMove,StableP) :-
    \+ service(S,_,_,_), % removed service
    crStep(Ps,(AllocHW,AllocBW),ServicesToMove,StableP).
crStep([on(S,N)|Ps],(AllocHW,AllocBW),ServicesToMove,[on(S,N)|StableP]) :-
    crStep(Ps,(AllocHW,AllocBW),ServicesToMove,StableP),
    nodeOk(S,N,StableP,AllocHW),linksOk(S,N,StableP,AllocBW). % ok service
crStep([on(S,_)|Ps],(AllocHW,AllocBW),[S|ServicesToMove],StableP) :-
    crStep(Ps,(AllocHW,AllocBW),ServicesToMove,StableP),
    \+ (nodeOk(S,N,StableP,AllocHW), linksOk(S,N,StableP,AllocBW)). % ko service
crStep([],_,[],[]). % base case
```

Experimental Results



Blind search



Heuristic search

- Discrete simulation at
  - varying infrastructure conditions (from 10% to 50% probability of node/link change)
  - varying application spec (10 lifelike commits)

Conclusions

- FogBrainX is a methodology and prototype to support next-gen application management via continuous reasoning.
- Declarative, explainable, scalable.
- Average **speedup > 50**× wrt non-incremental reasoning
- **65 lines of code** vs 1000+ of existing procedural solutions

FogBrainX

Stefano Forti, Giuseppe Bisicchia and Antonio Brogi. Declarative Continuous Reasoning in the Cloud-IoT Continuum. Journal of Logic and Computation. 2021.