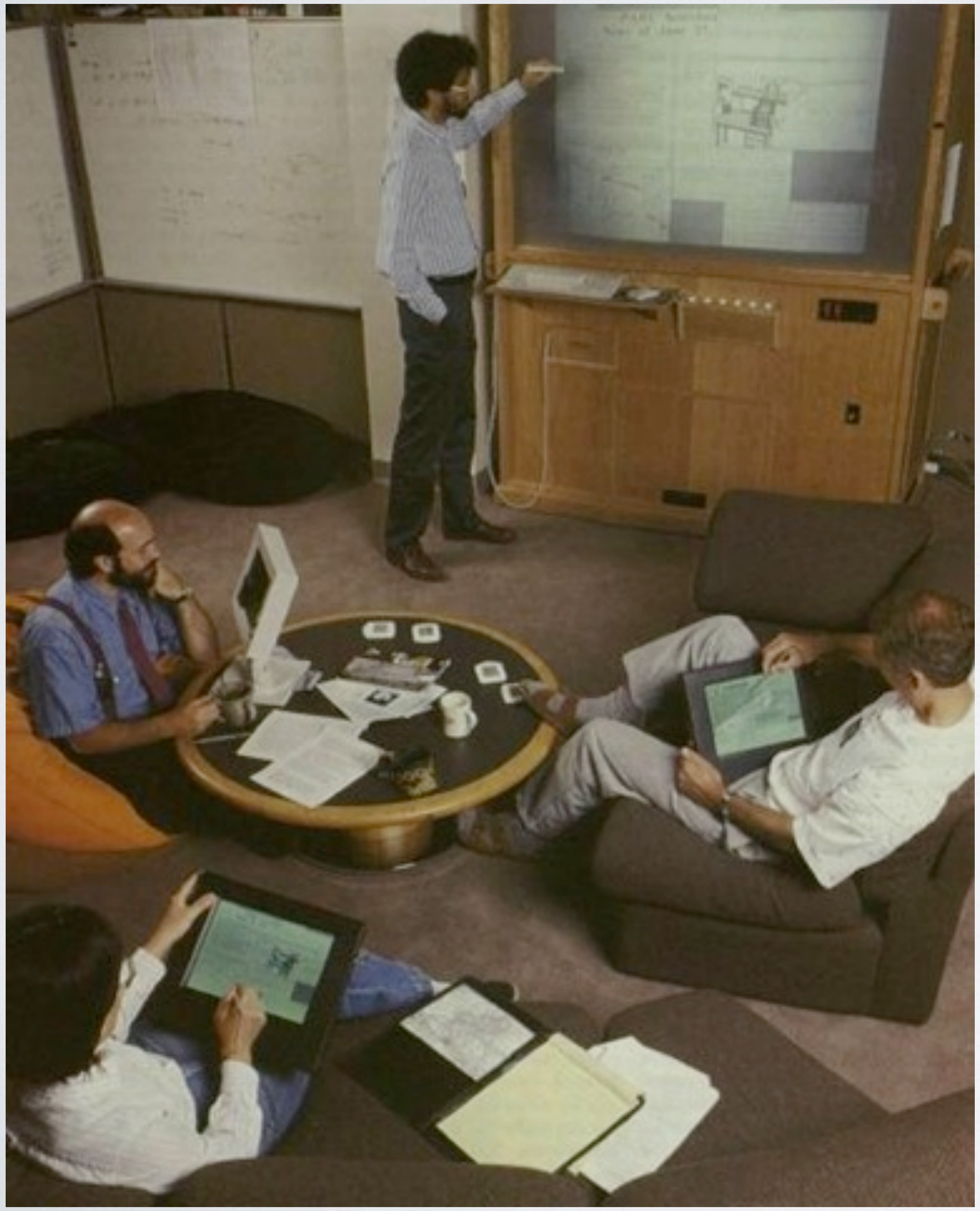
Shared Substance:
Developing Flexible Multisurface Applications

Tony Gjerlufsen [1, 2, 3]
Clemens N. Klokmose [2, 3]
James R. Eagan [2, 3]
Clément Pillias [3, 2]
Michel Beaudouin-Lafon [2, 3]

|in|**situ**| - insitu.lri.fr

[1] Univ. Aarhus (Denmark) - [2] Univ. Paris-Sud (France) - [3] INRIA (France)

How to develop multisurface applications ?

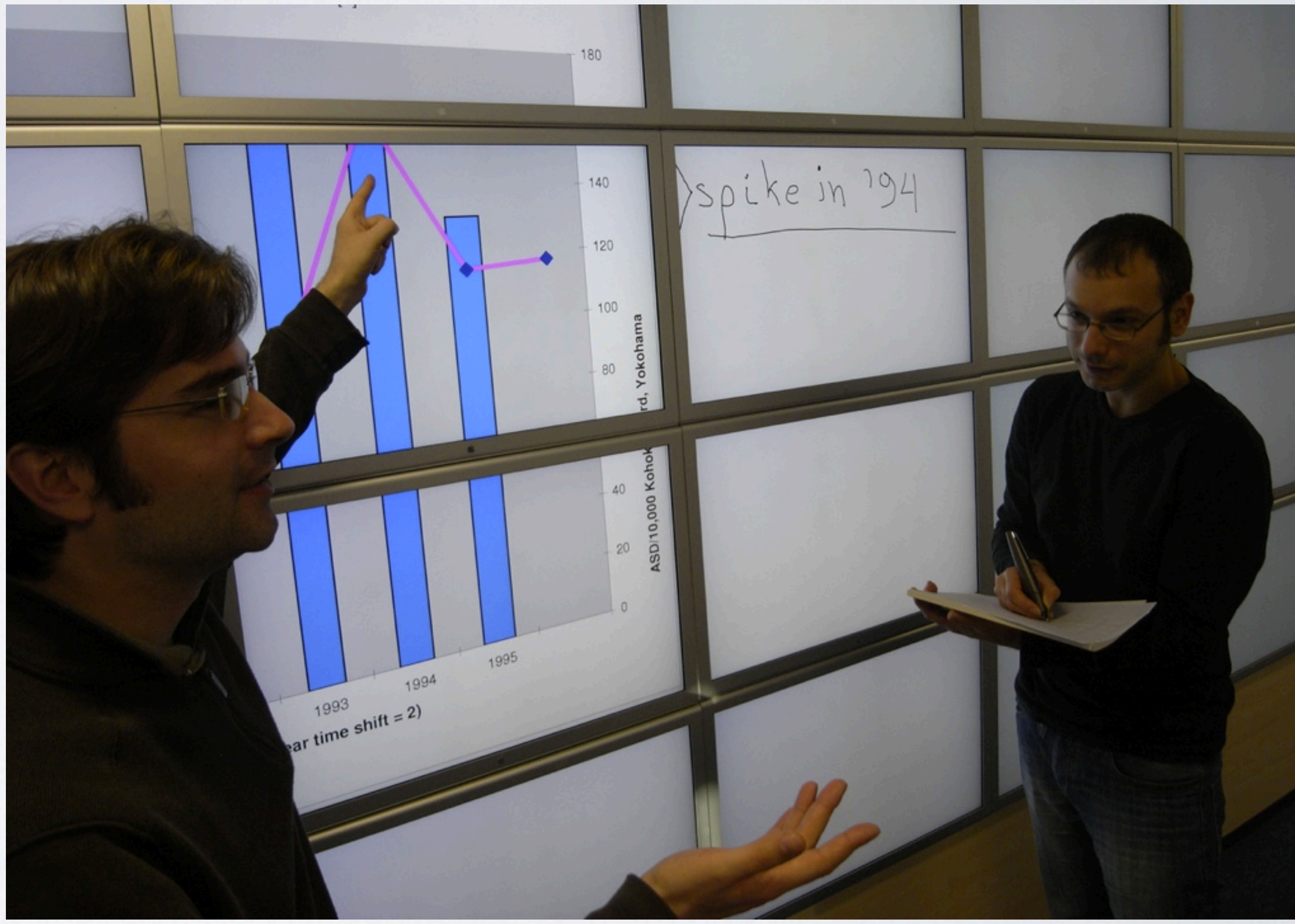


Ubicomp (Weiser, 1991)

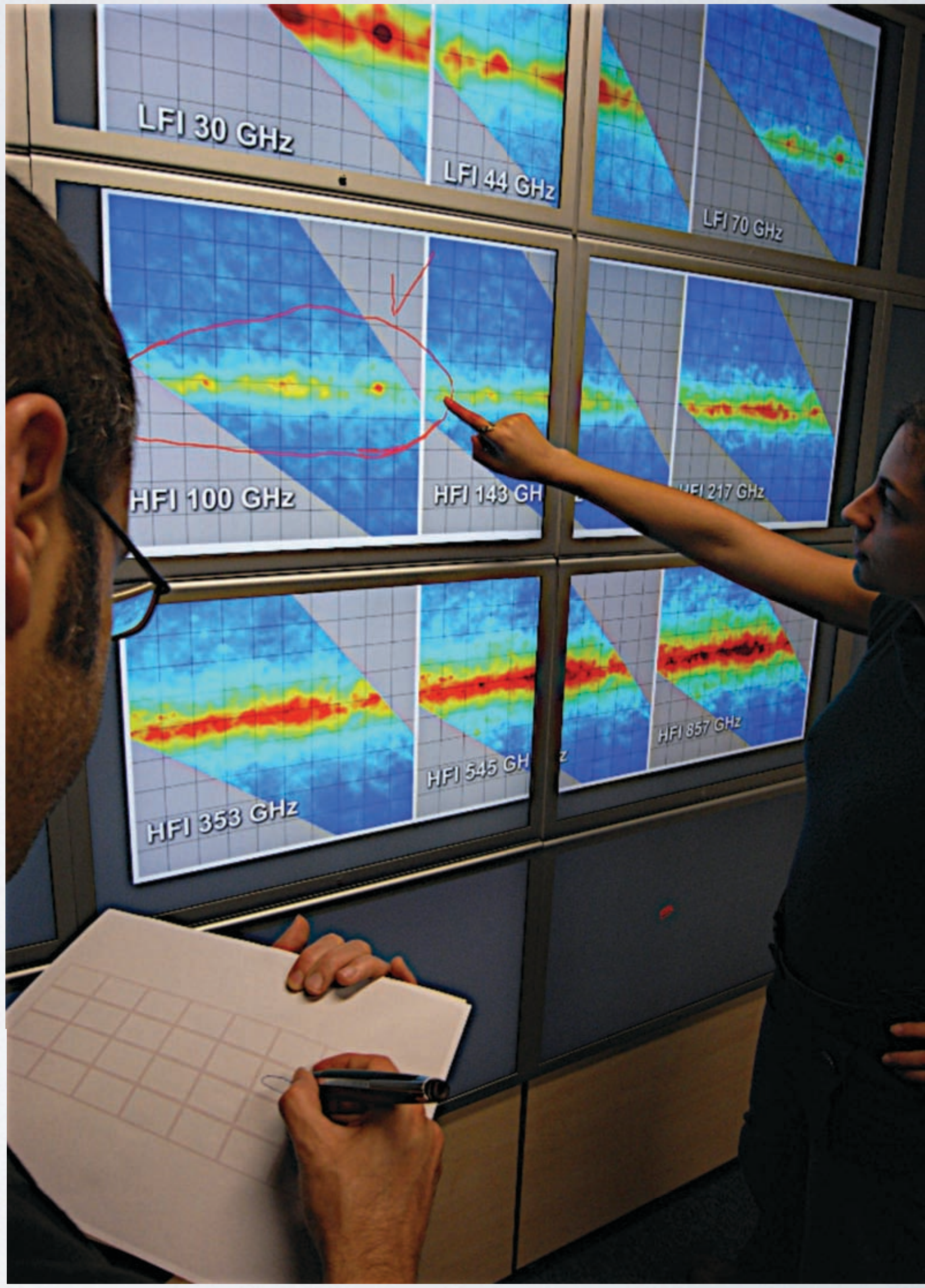
pick-and-drop (Rekimoto, 1997)

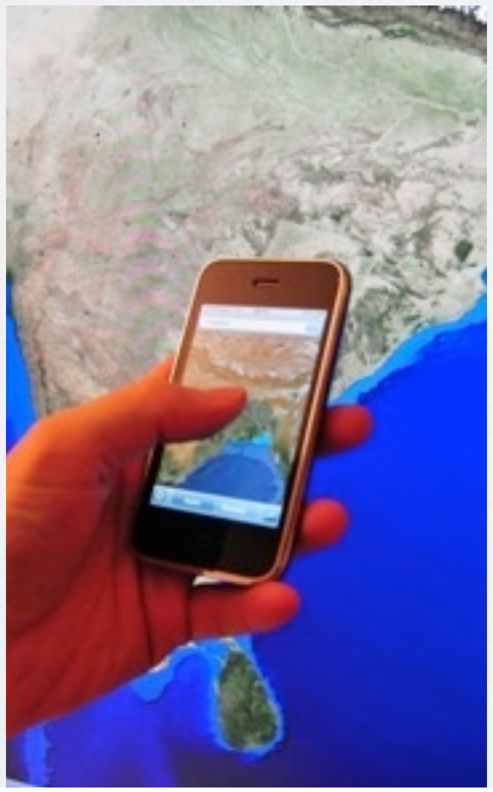
An experimental platform: The WILD room

- Lead users: scientists who analyze big data



Participatory design





Conception :
qu'est-ce qu'on fait ?
Evaluation :

Two key ideas

- **Flexible sharing**
- **Instruments**

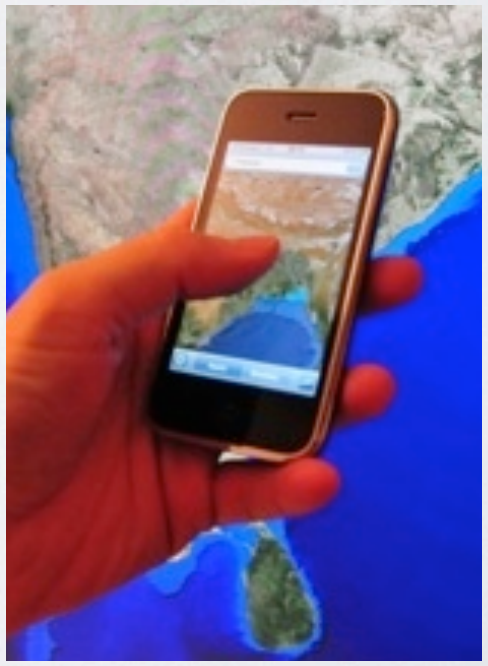
Two key ideas

- **Flexible sharing**
  - Content
  - Application state
  - Physical resources
  - System resources
- **Instruments**

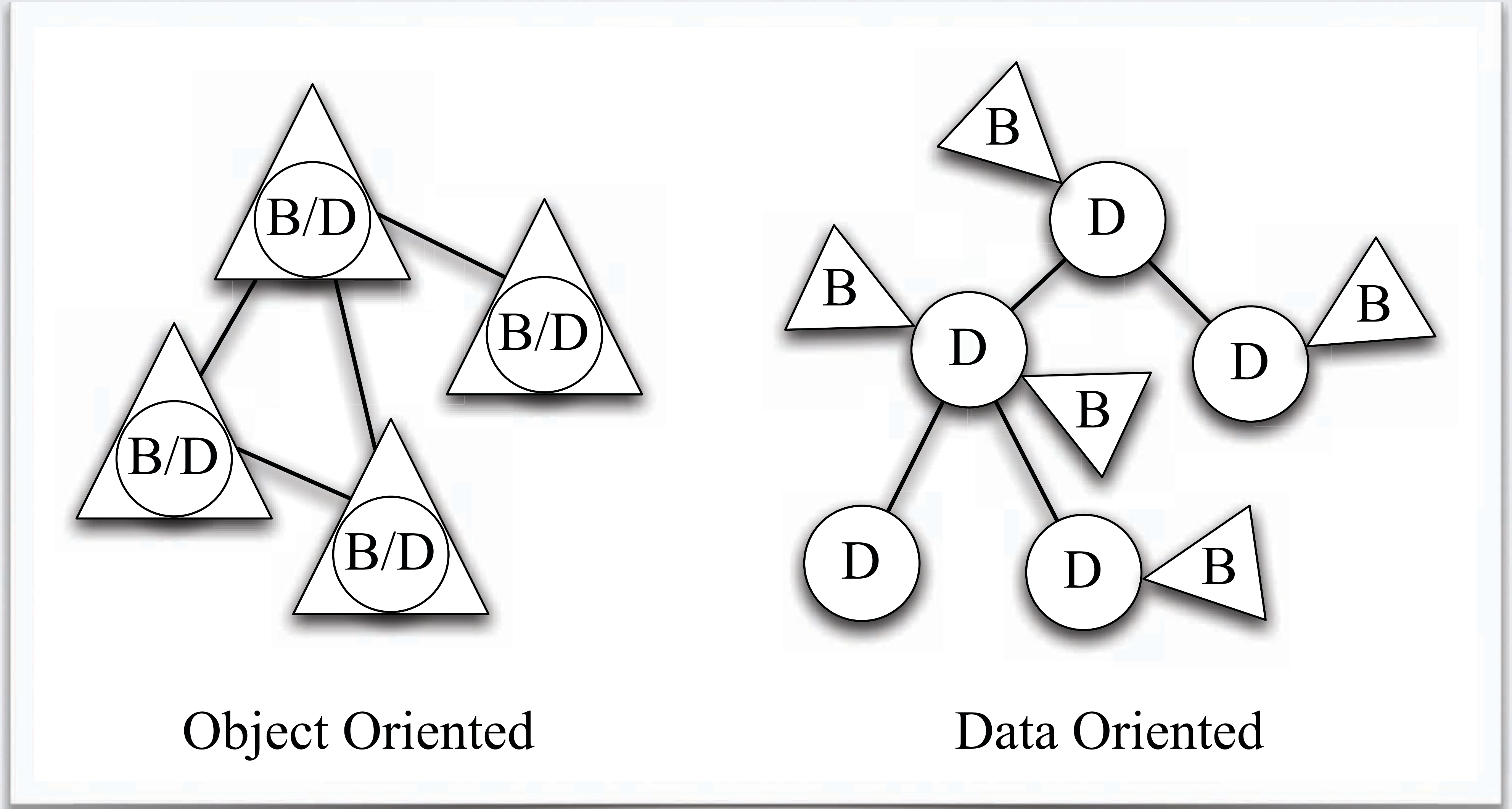
Two key ideas

- **Flexible sharing**
  - Content
  - Application state
  - Behavior
  - Physical resources
  - System resources

- **Instruments**
  - Separate from objects
  - Generic & specialized

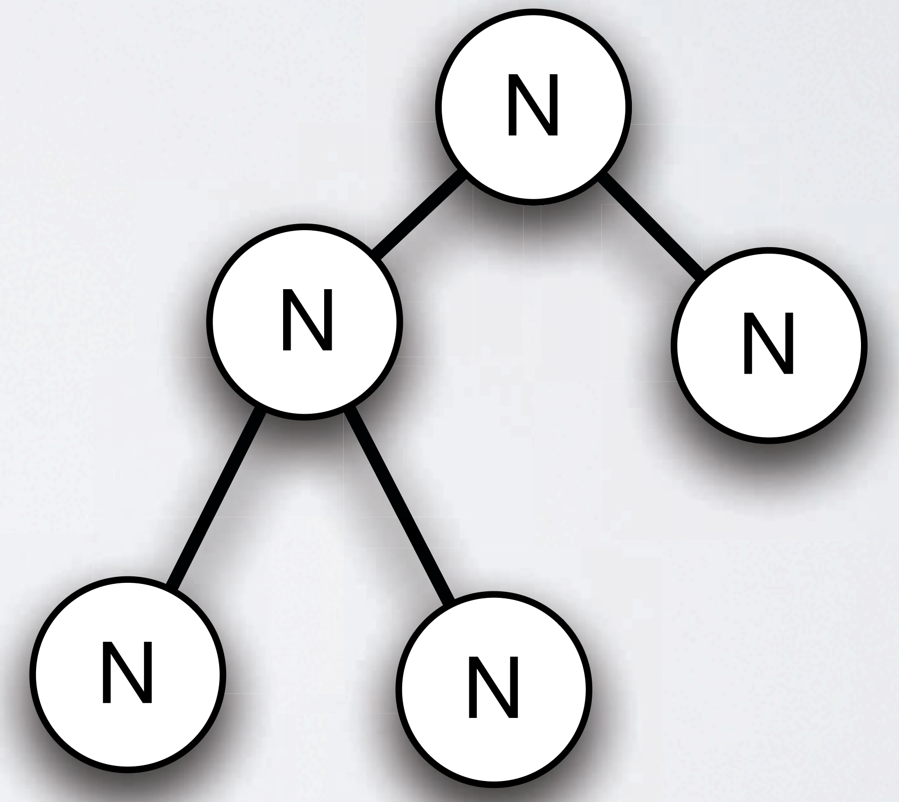


Data-orientation vs. Object-orientation



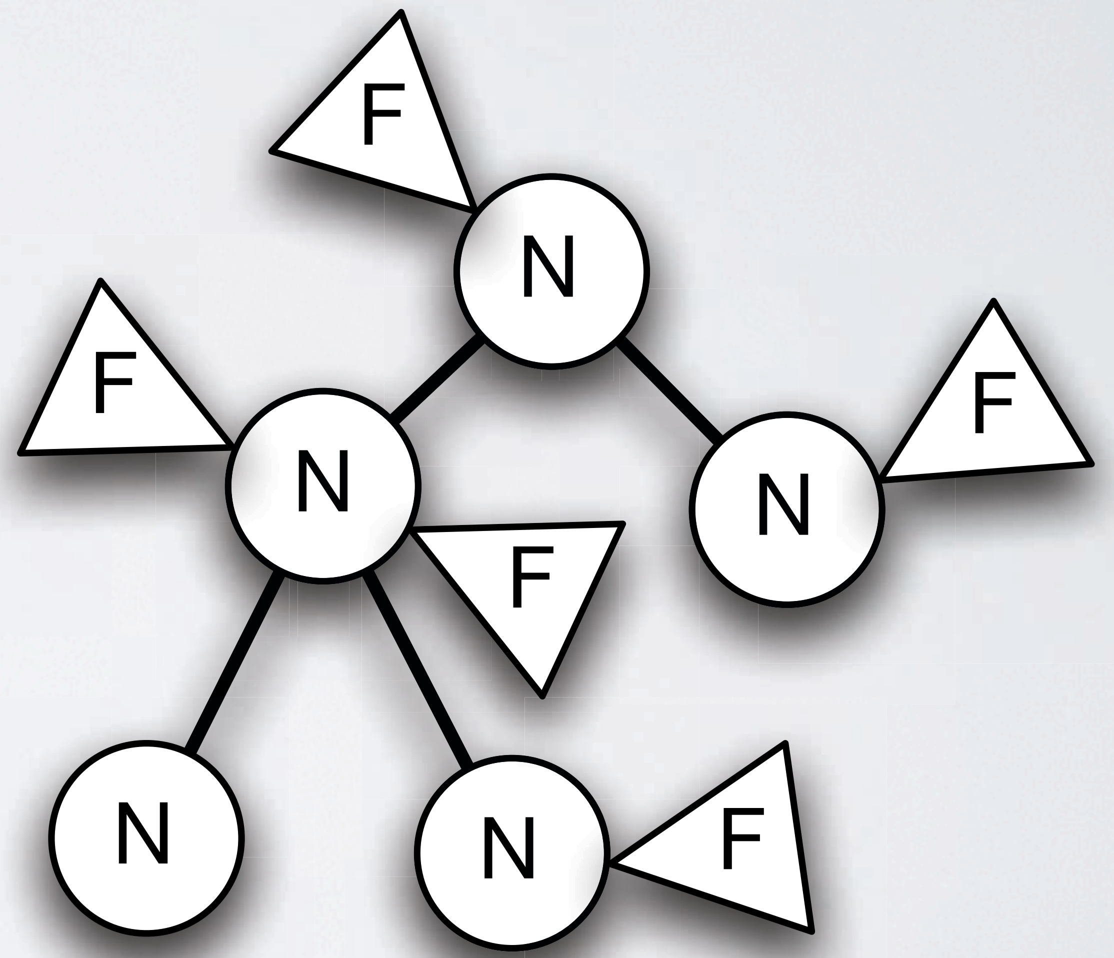
Data-oriented model

- **Nodes** (data)
  - Organized in a tree
  - Values and children



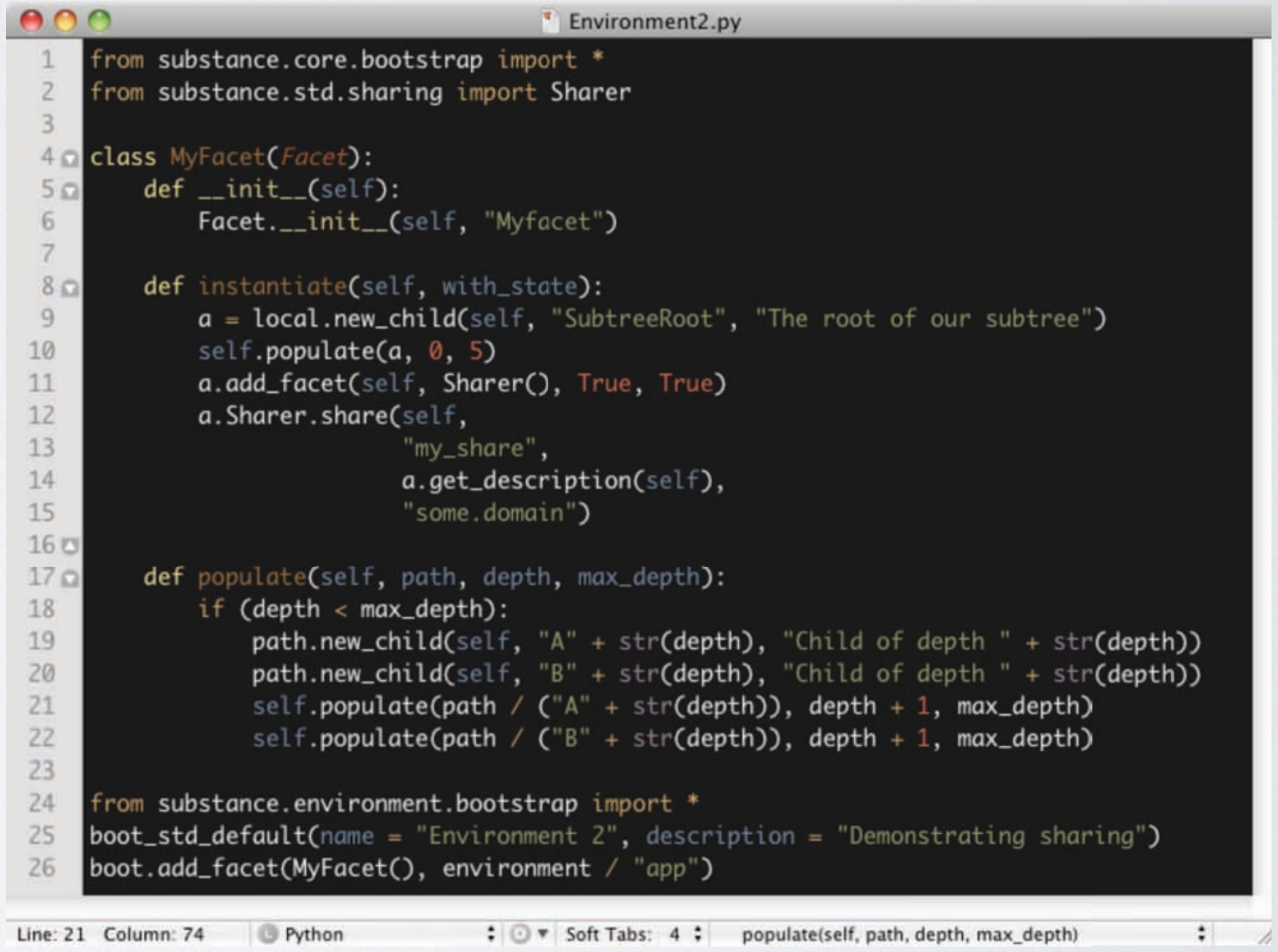
Data-oriented model

- **Nodes** (data)
  - Organized in a tree
  - Values and children
- **Facets** (behavior)
  - Local to nodes
  - Dynamically added/removed



Substance

- Implementation of the data-oriented model in Python
  - Event-driven
  - Reactive and Imperative programming styles
- Participatory design workshops

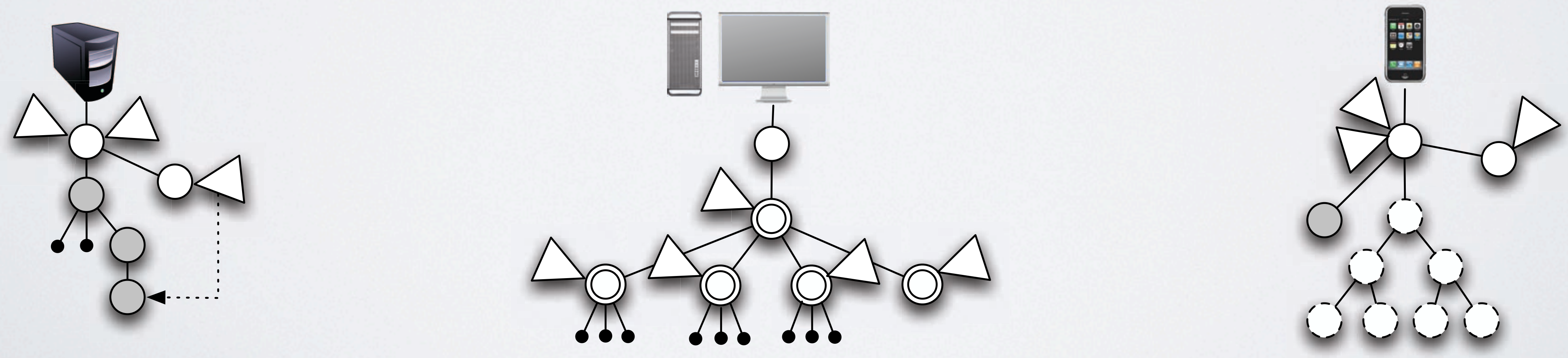

Environment2.py

```python
from substance.core.bootstrap import *
from substance.std.sharing import Sharer

class MyFacet(Facet):
    def __init__(self):
        Facet.__init__(self, "Myfacet")

    def instantiate(self, with_state):
        a = local.new_child(self, "SubtreeRoot", "The root of our subtree")
        self.populate(a, 0, 5)
        a.add_facet(self, Sharer(), True, True)
        a.Sharer.share(self,
                       "my_share",
                       a.get_description(self),
                       "some.domain")

    def populate(self, path, depth, max_depth):
        if (depth < max_depth):
            path.new_child(self, "A" + str(depth), "Child of depth " + str(depth))
            path.new_child(self, "B" + str(depth), "Child of depth " + str(depth))
            self.populate(path / ("A" + str(depth)), depth + 1, max_depth)
            self.populate(path / ("B" + str(depth)), depth + 1, max_depth)

from substance.environment.bootstrap import *
boot_std_default(name = "Environment 2", description = "Demonstrating sharing")
boot.add_facet(MyFacet(), environment / "app")
```

Line: 21 Column: 74 | Python | Soft Tabs: 4 | populate(self, path, depth, max_depth)

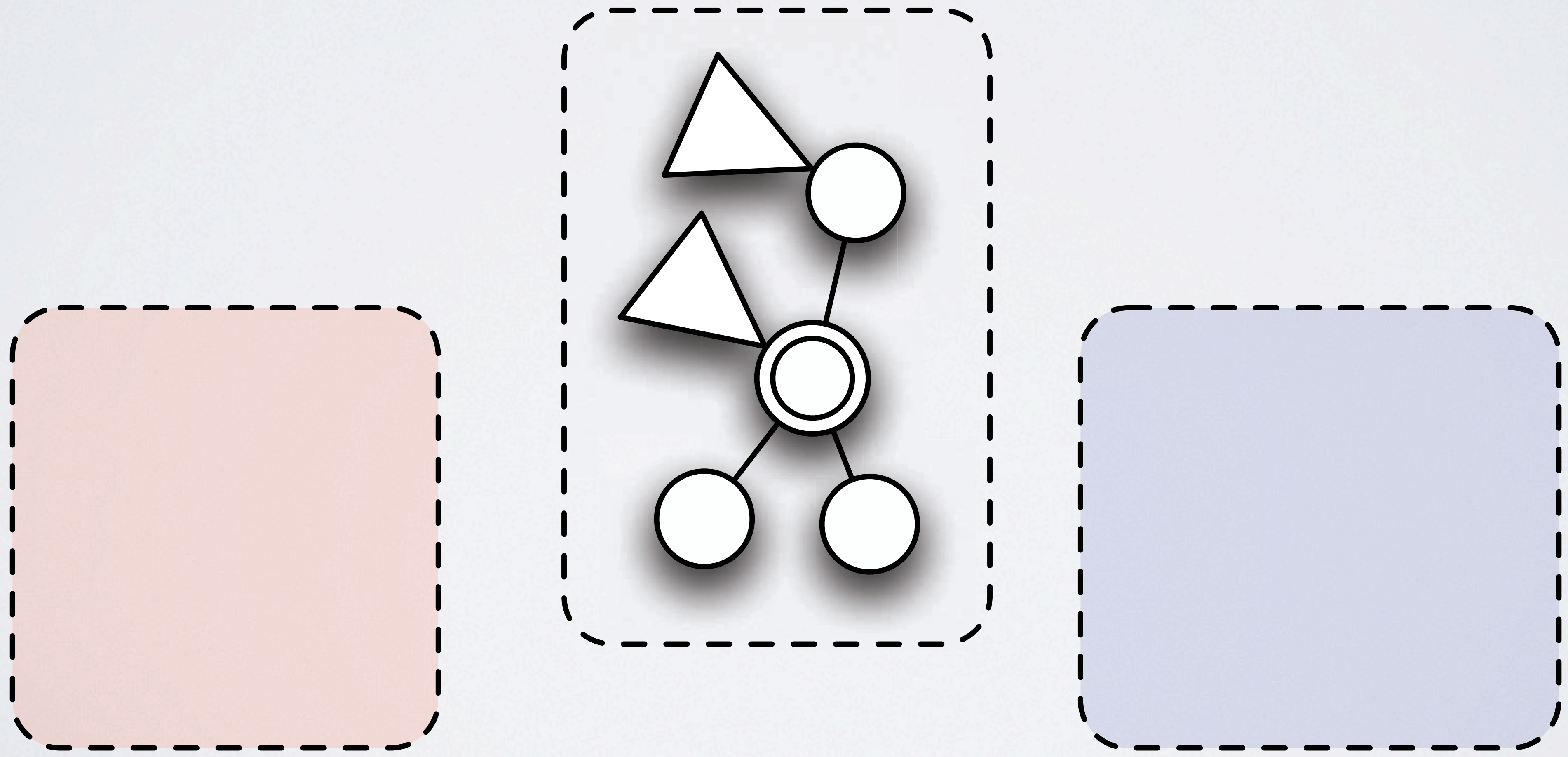
Shared Substance

- Collection of **Environments**
  - Dynamic network discovery to find available environments
  - Sharing through **Mounting** and **Replication**



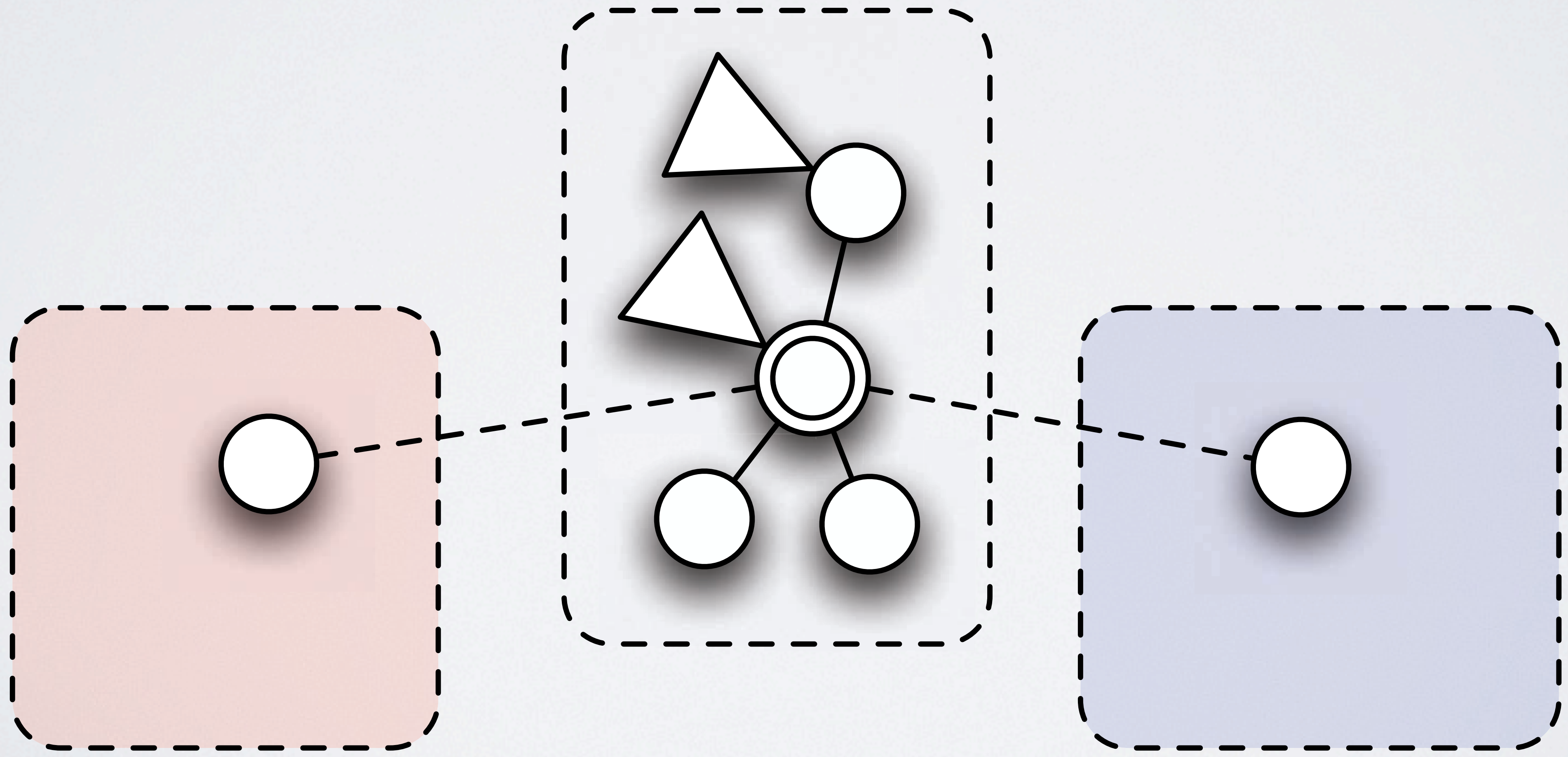
Mounting

- Remote access to a subtree in another environment



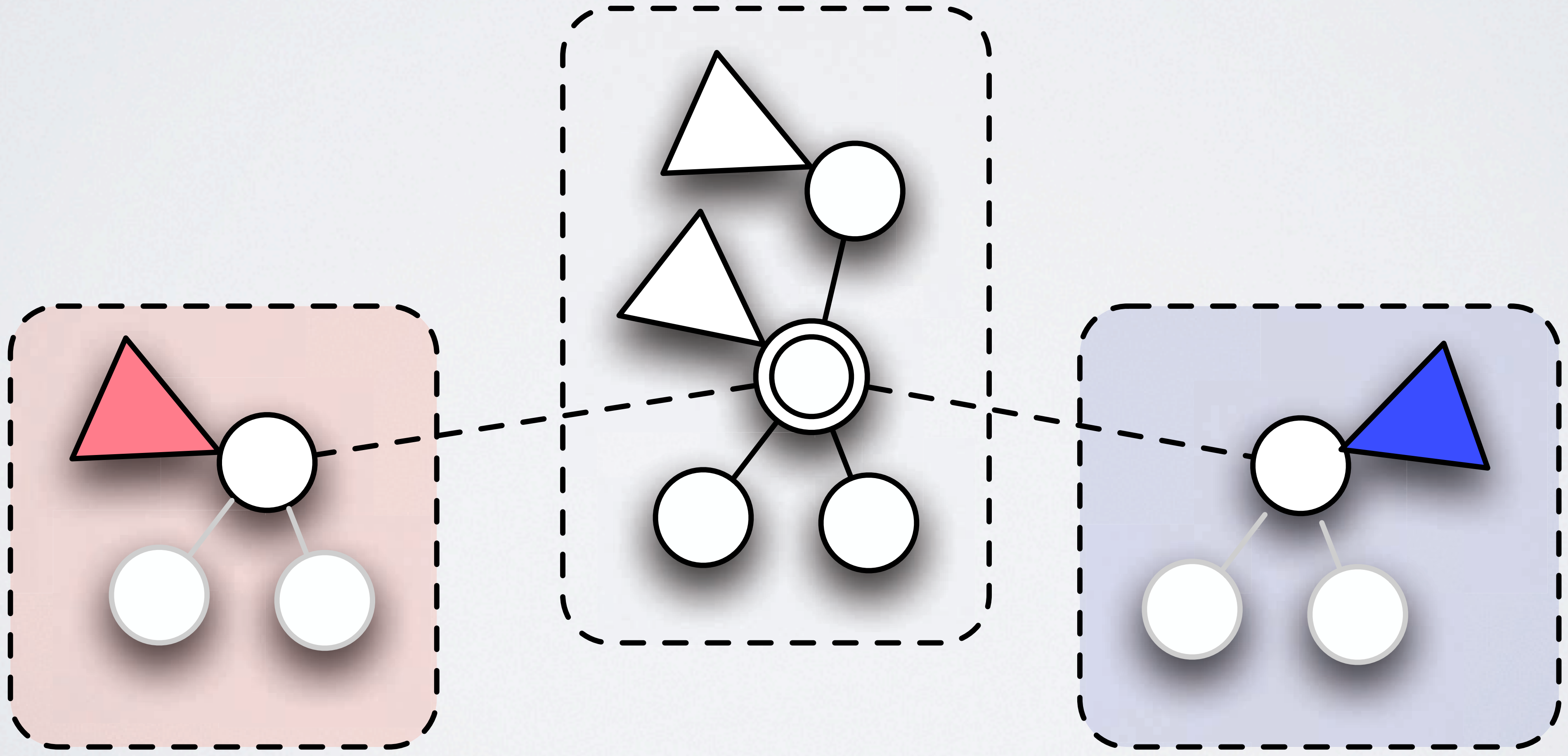
Mounting

- Remote access to a subtree in another environment



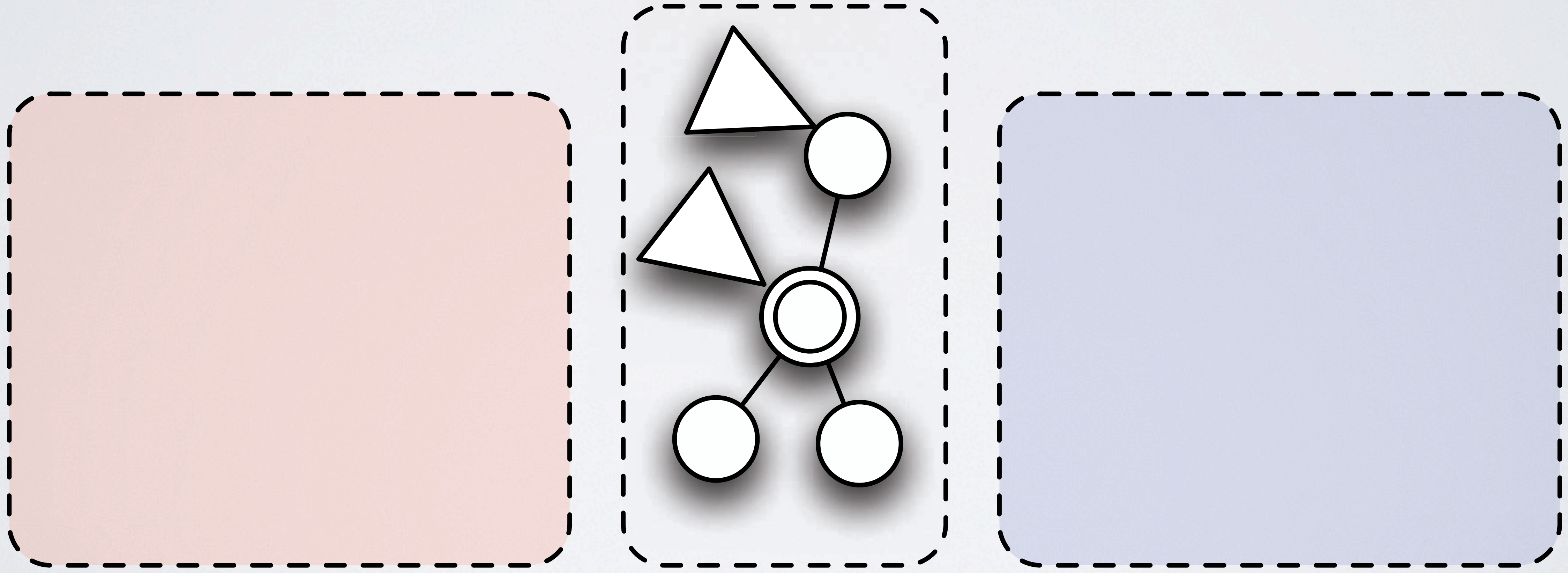
Mounting

- Remote access to a subtree in another environment



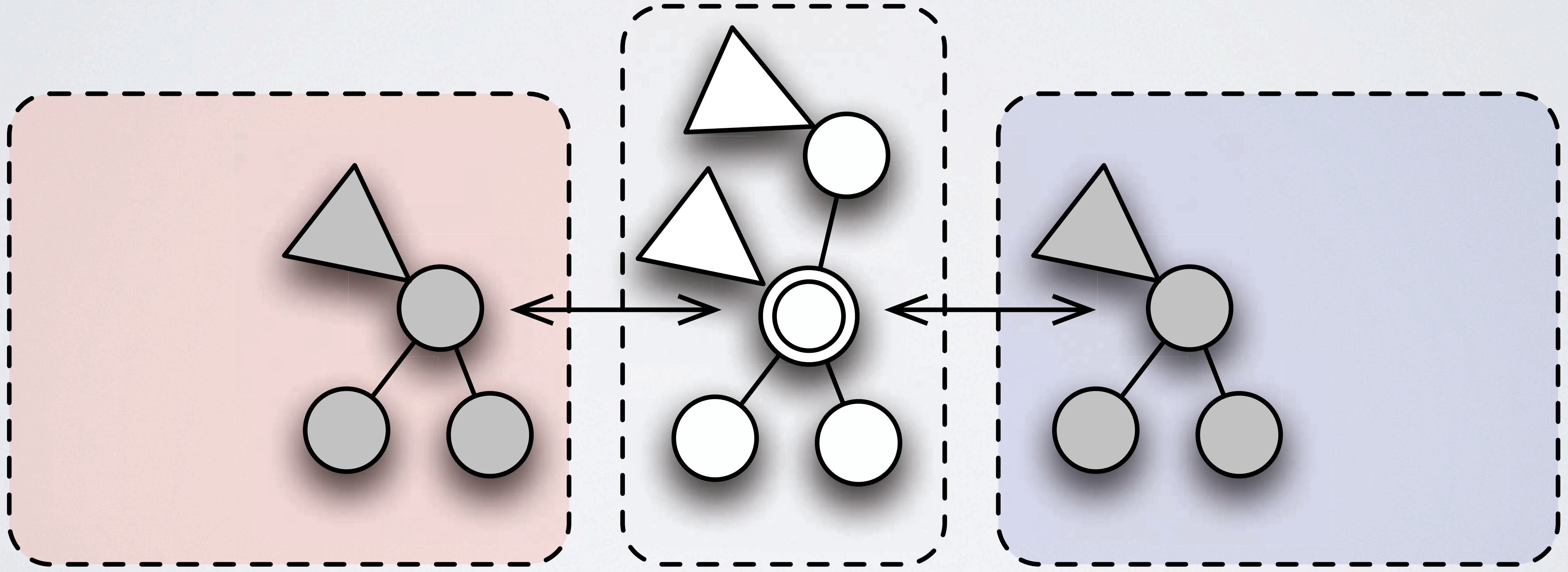
Replication

- Duplicate synchronized subtree



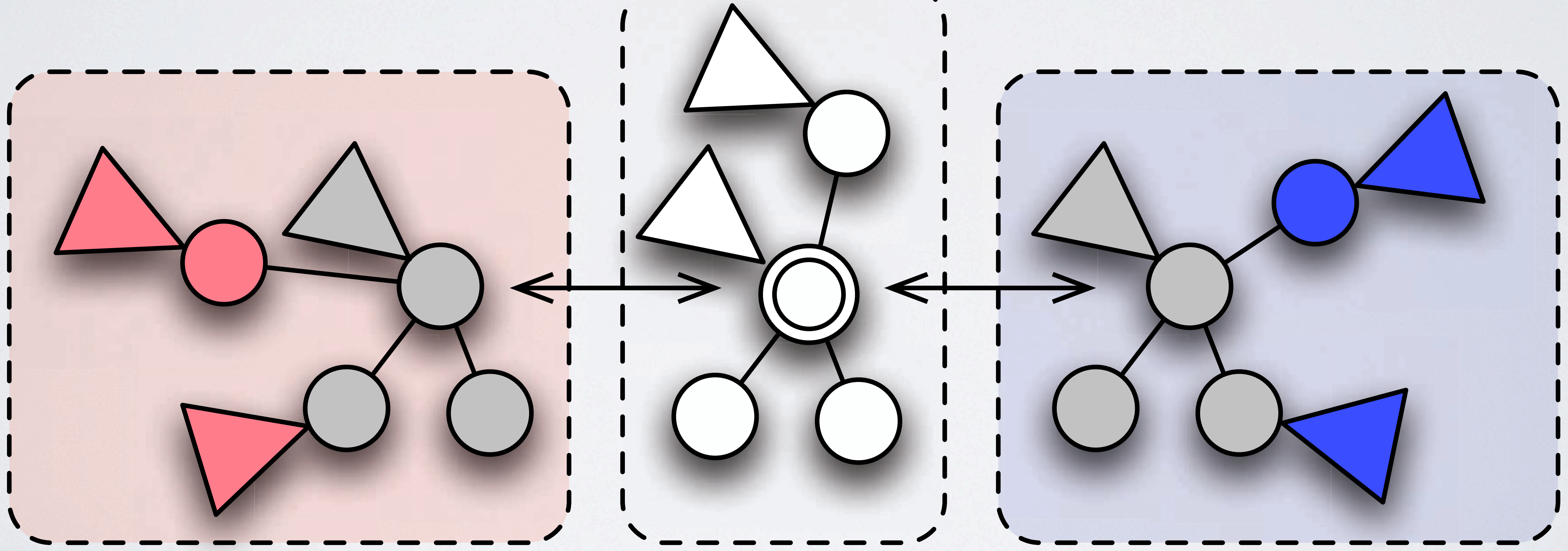
Replication

- Duplicate synchronized subtree

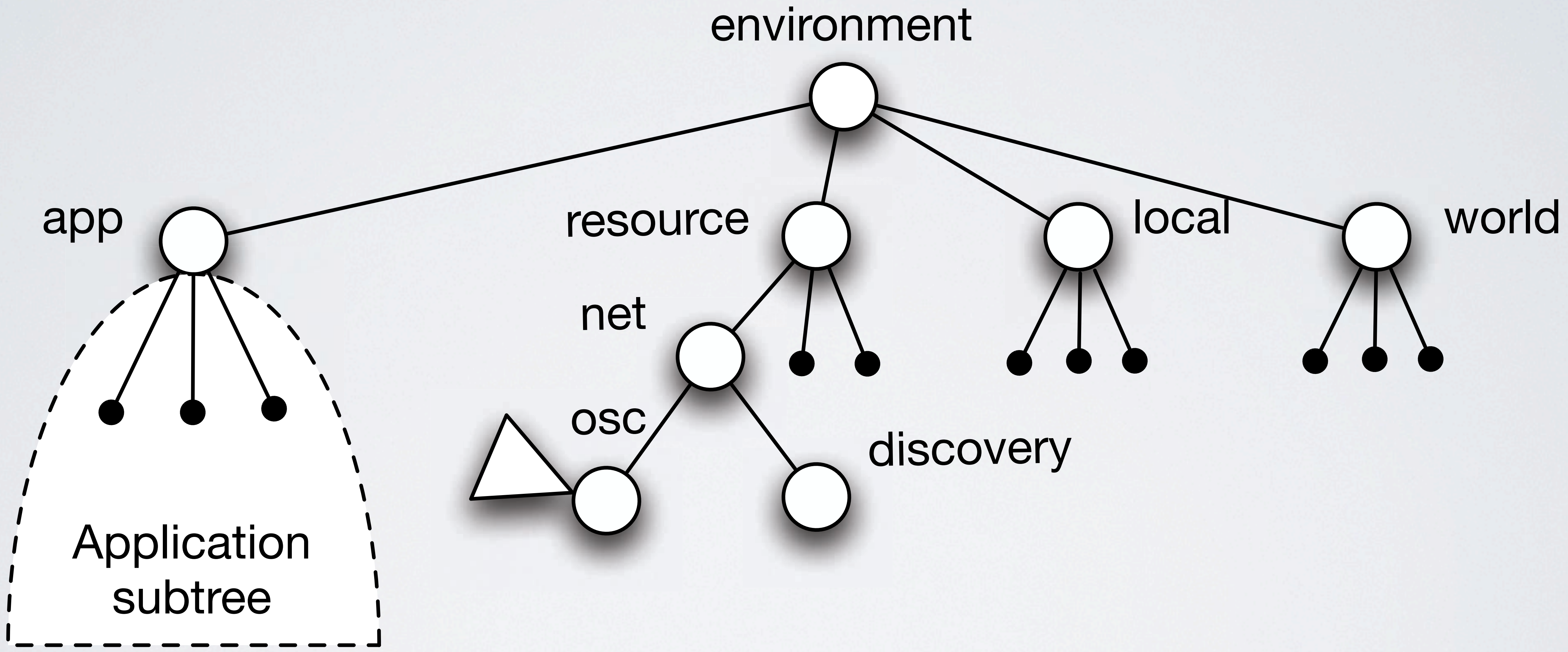


Replication

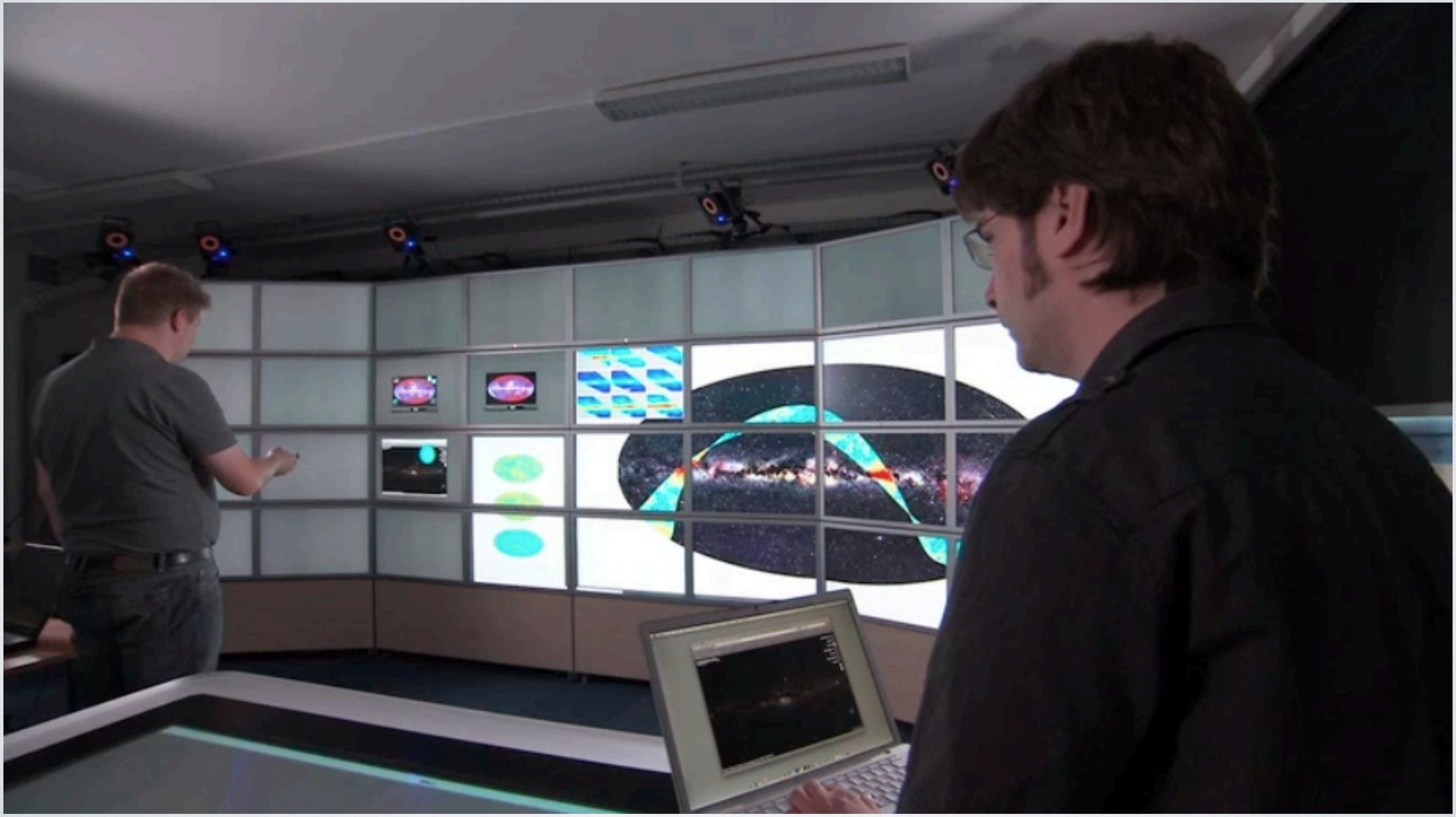
- Nodes and facets can be added to the replicates



Standard Substance Environment

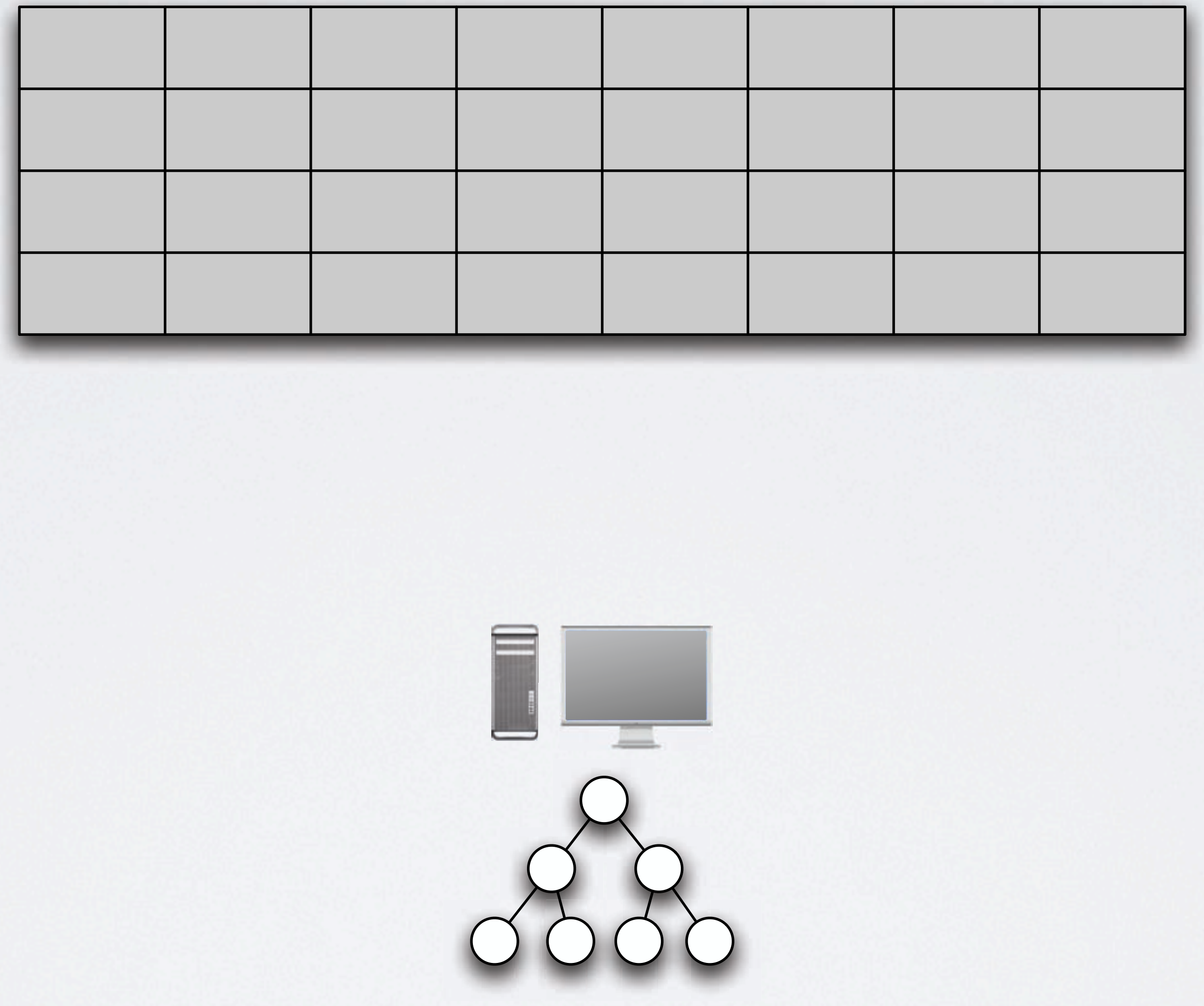


Substance canvas

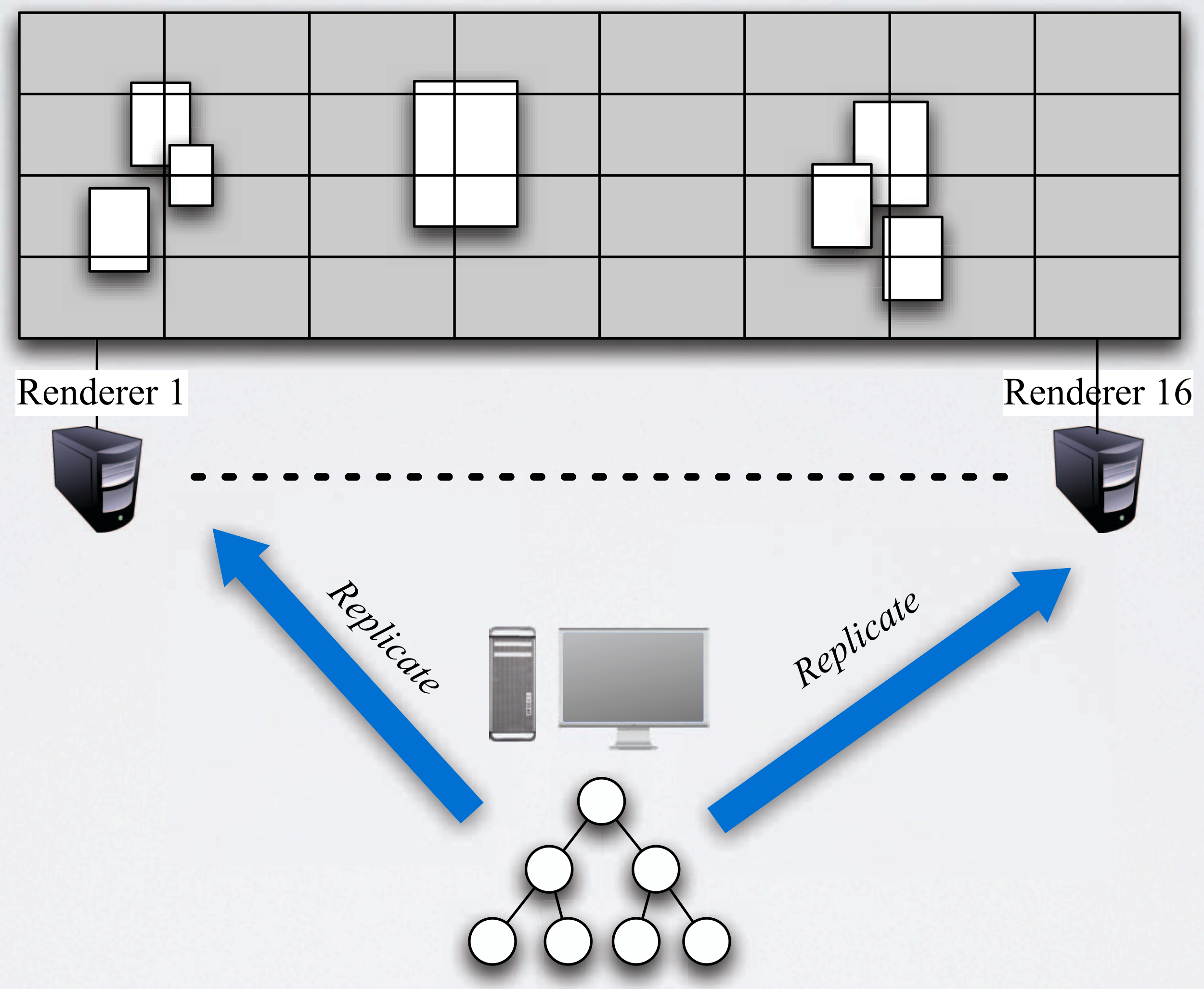


A scenario: aggregating content

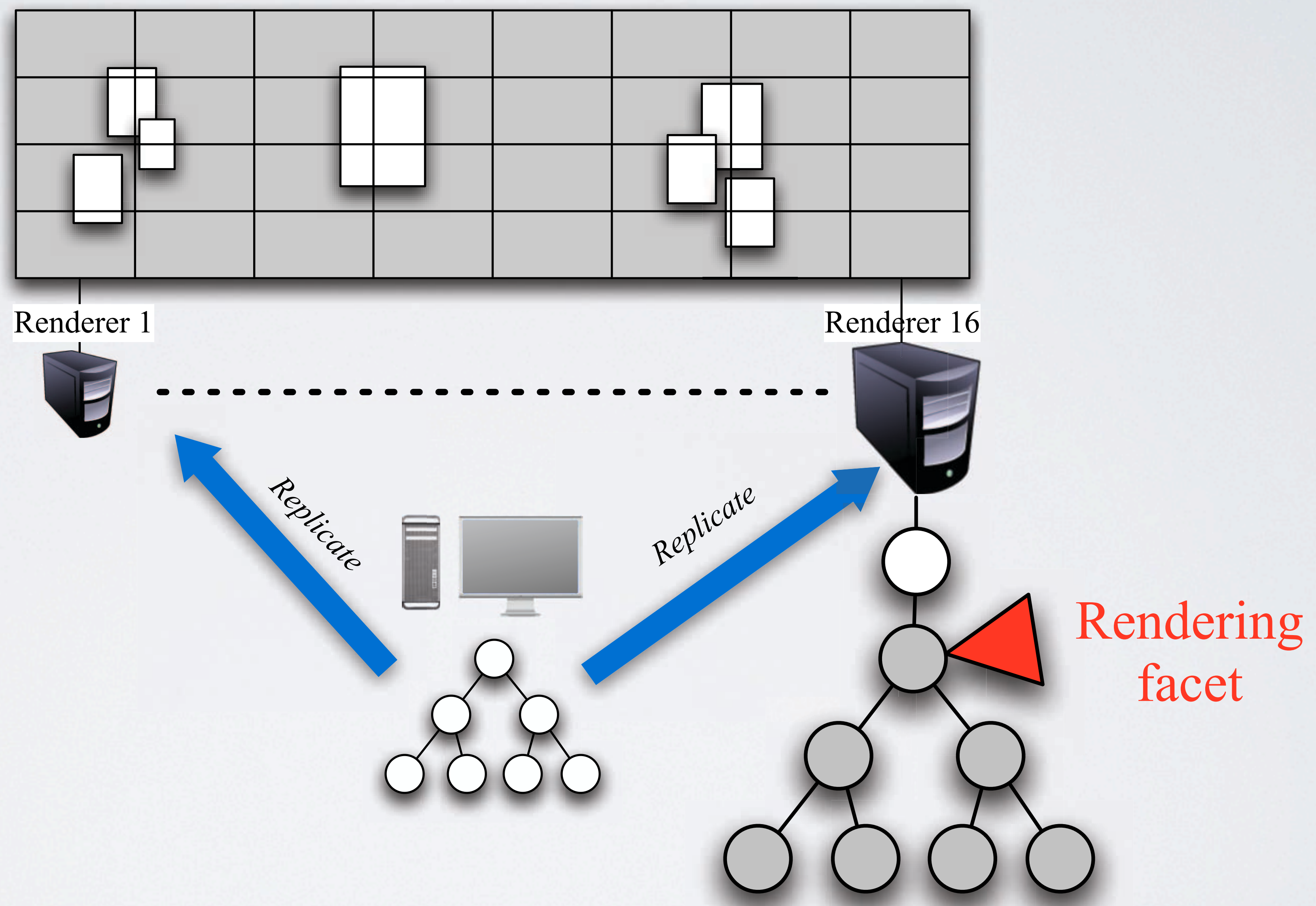
Shared scene graph



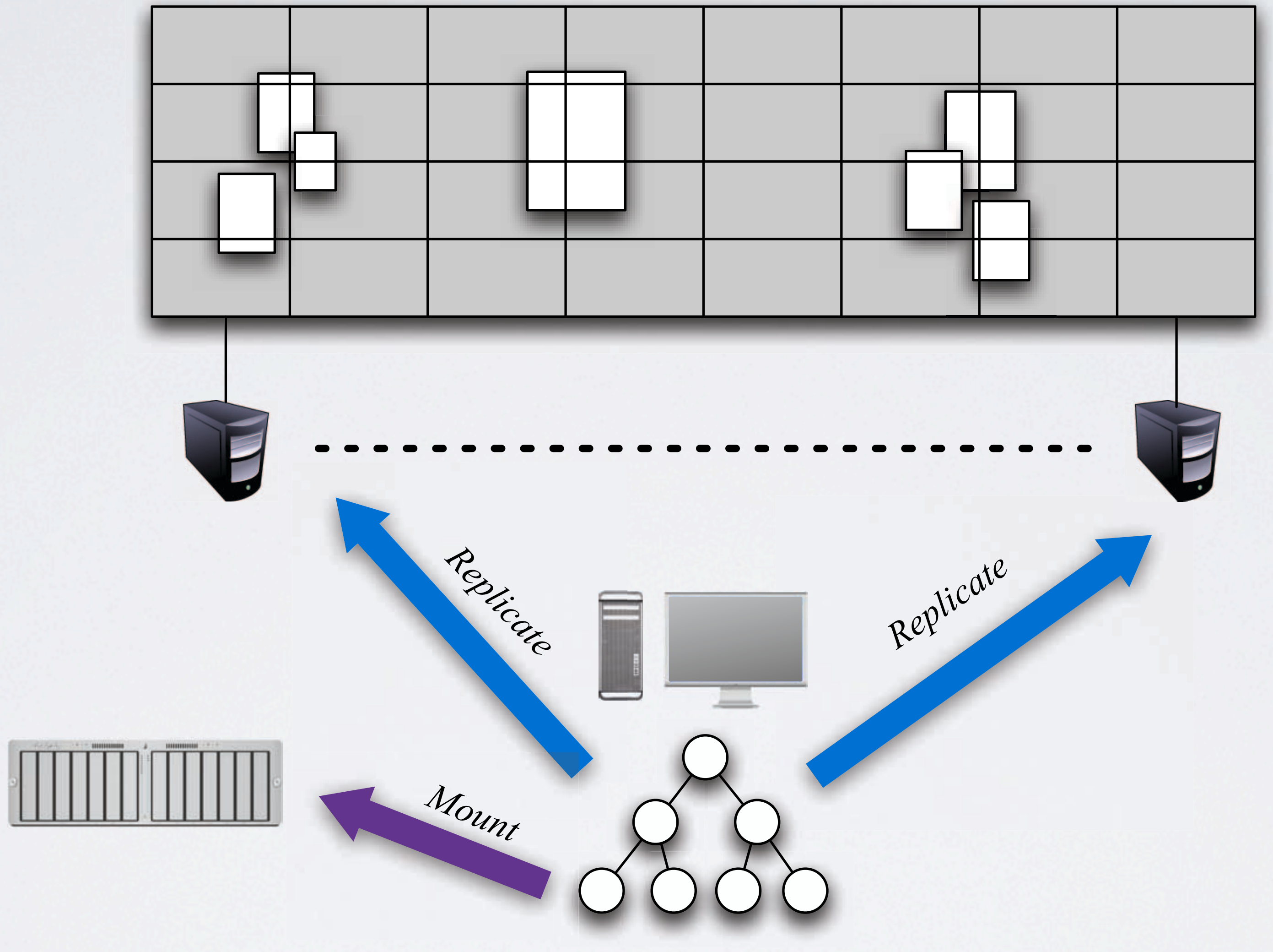
Rendering on the wall



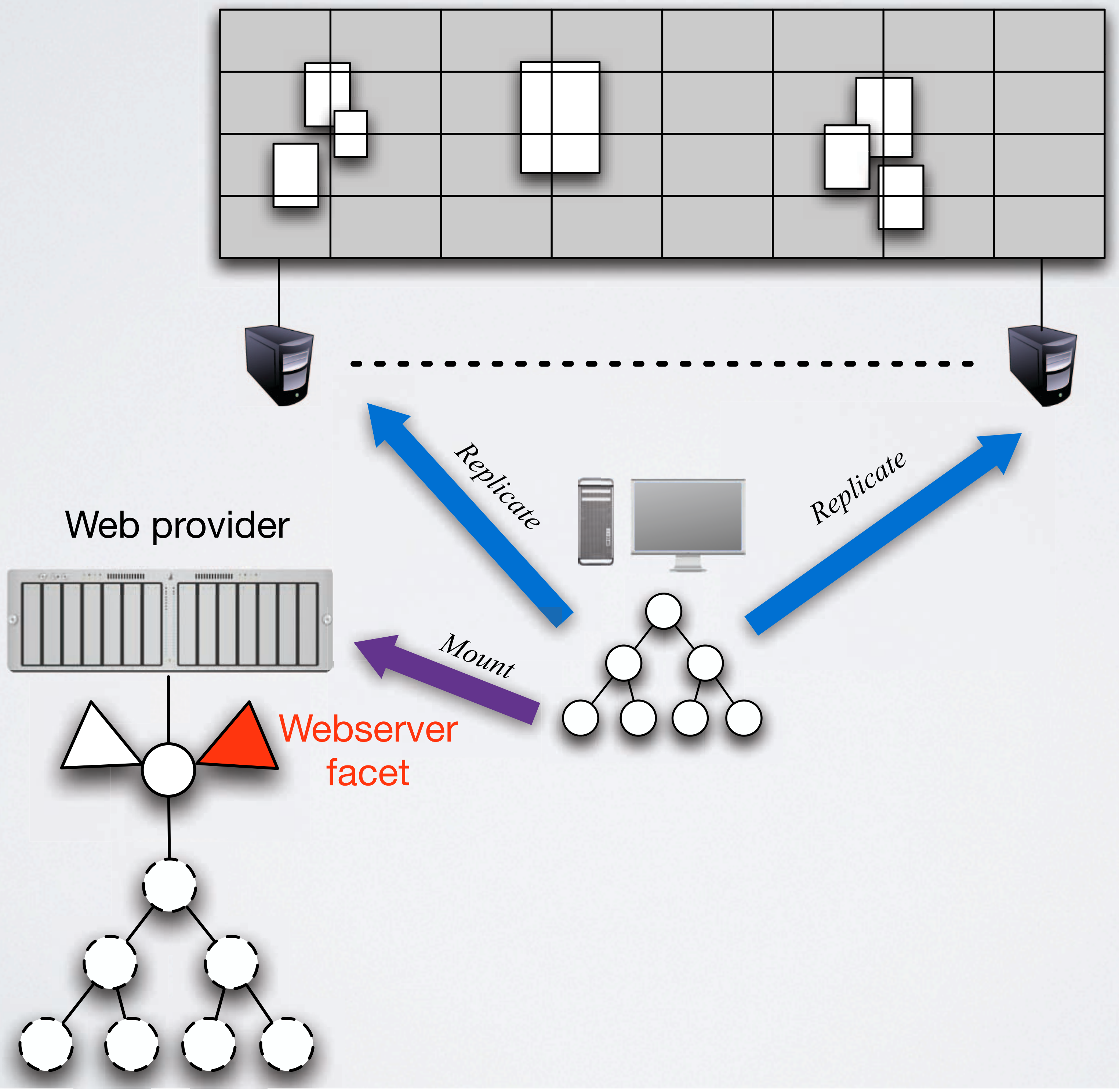
Rendering on the wall



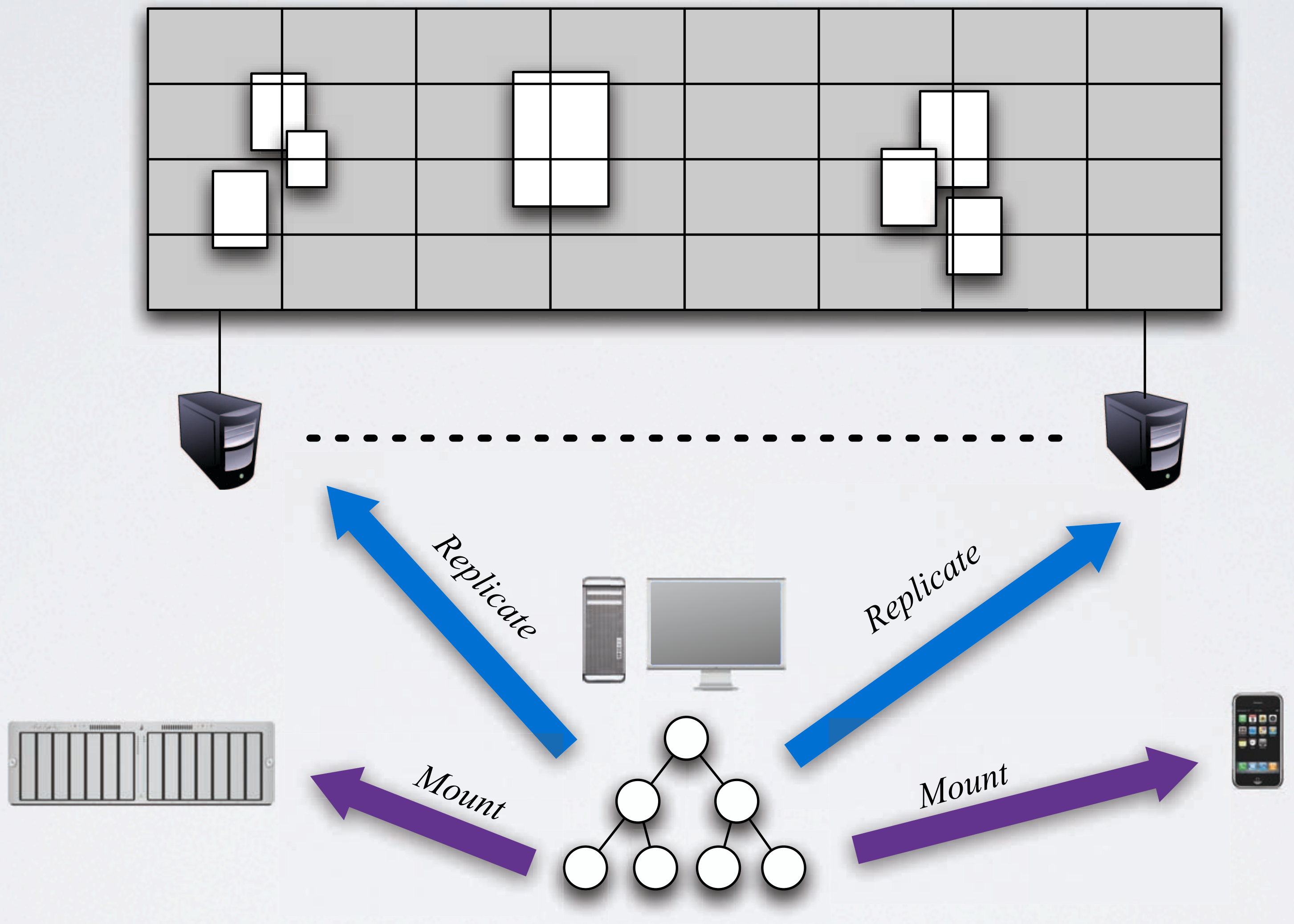
Content providers



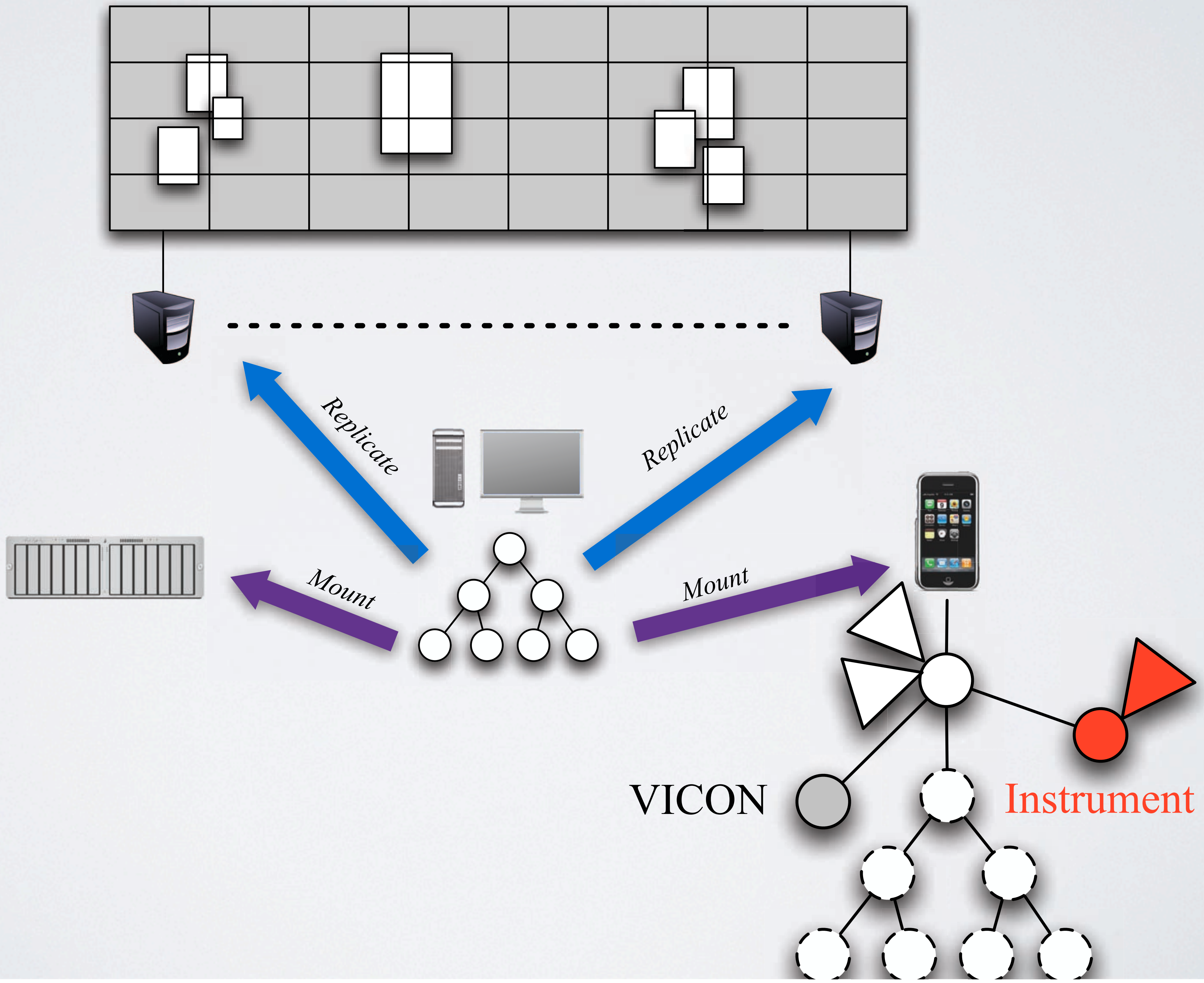
Content providers



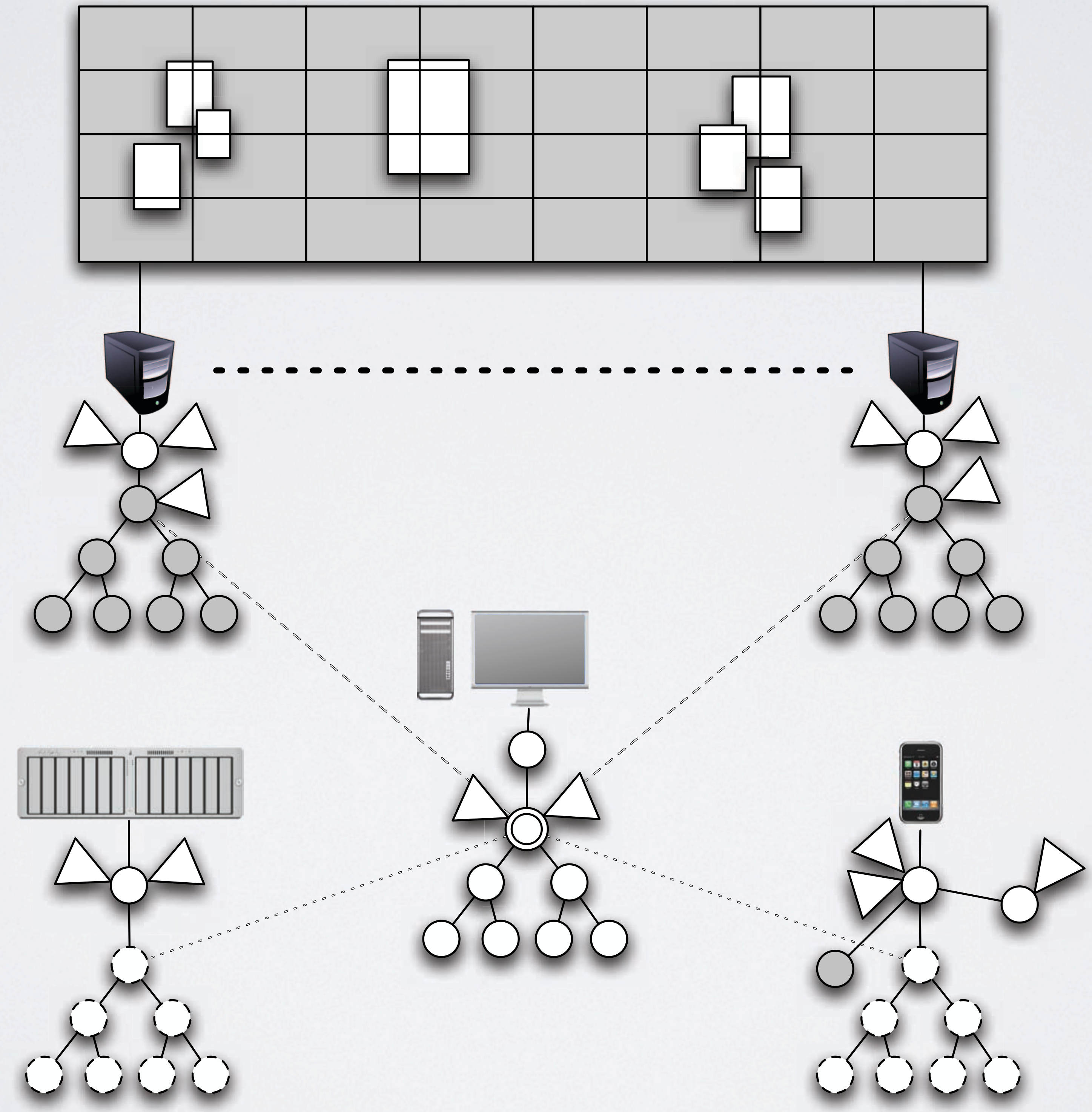
Instruments



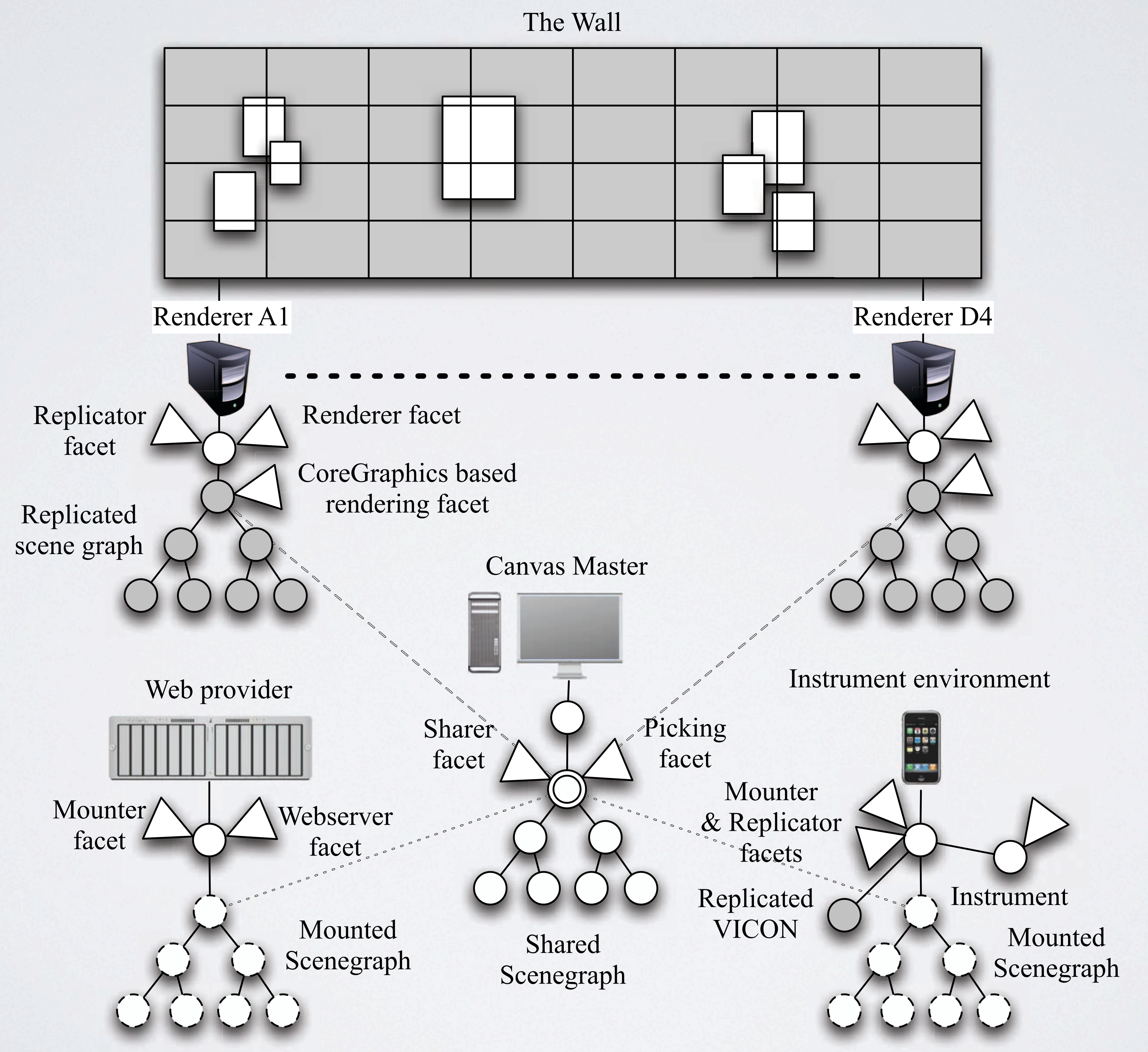
Instruments



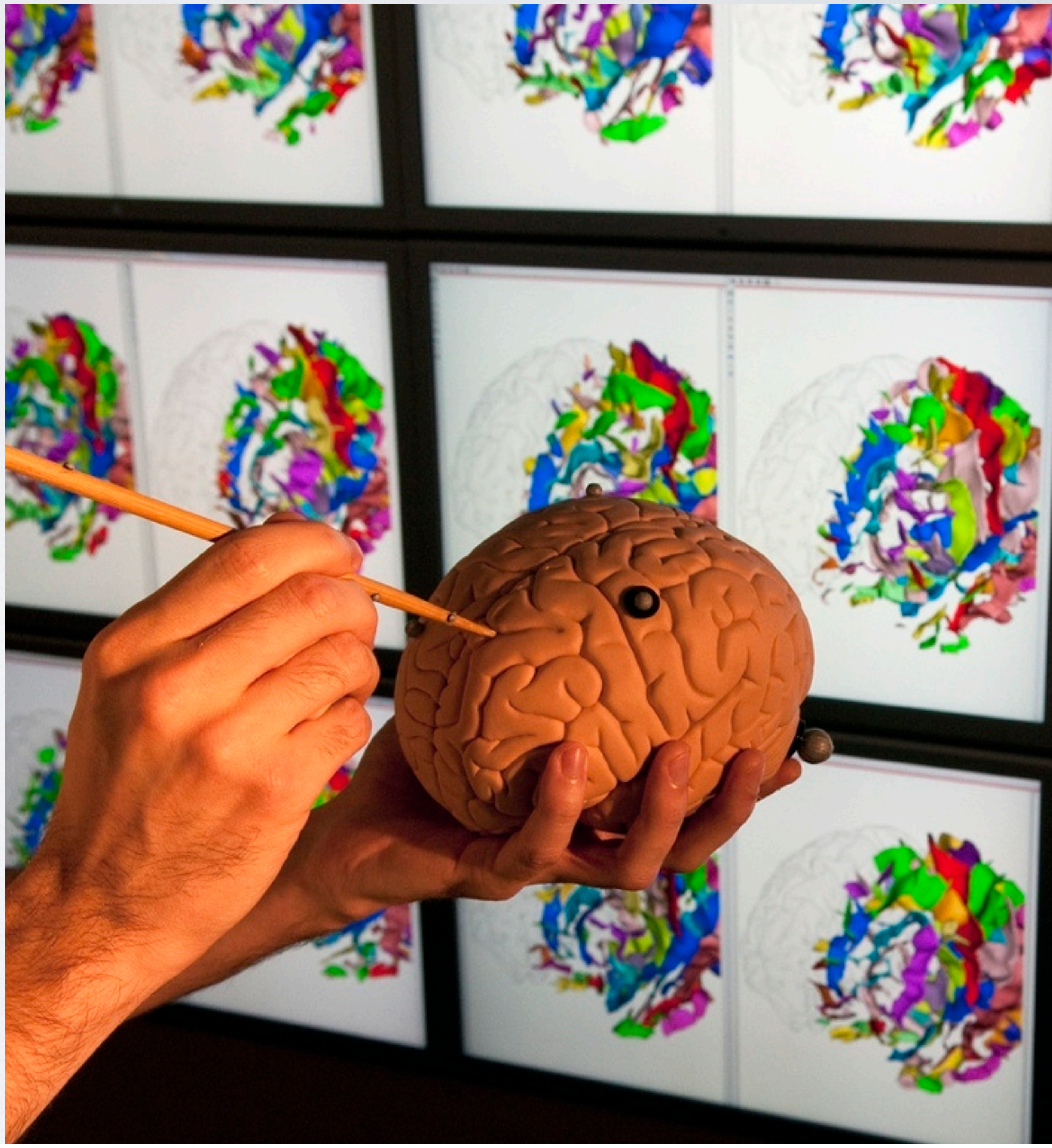
Substance Canvas



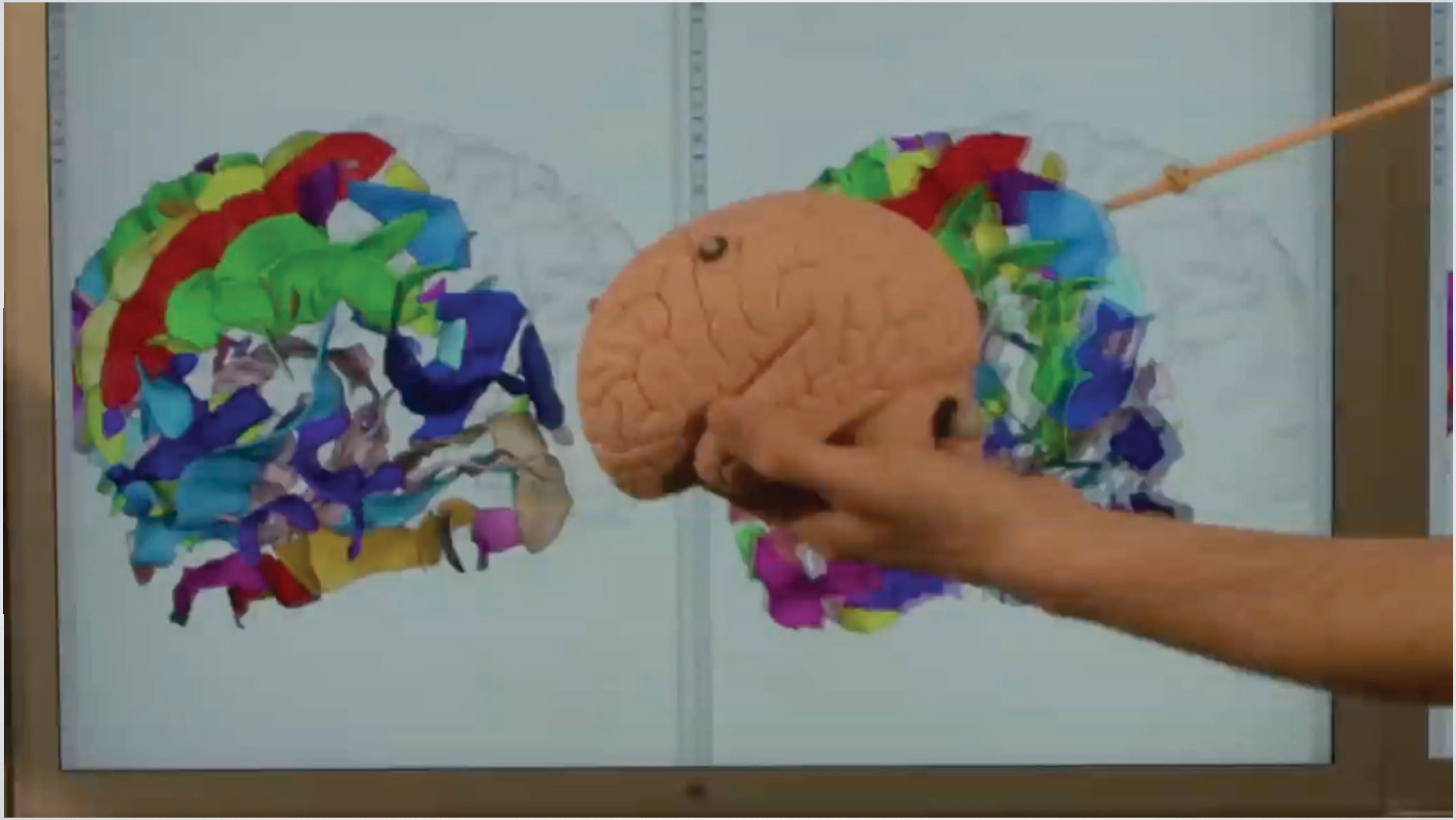
Substance Canvas



Substance Grise

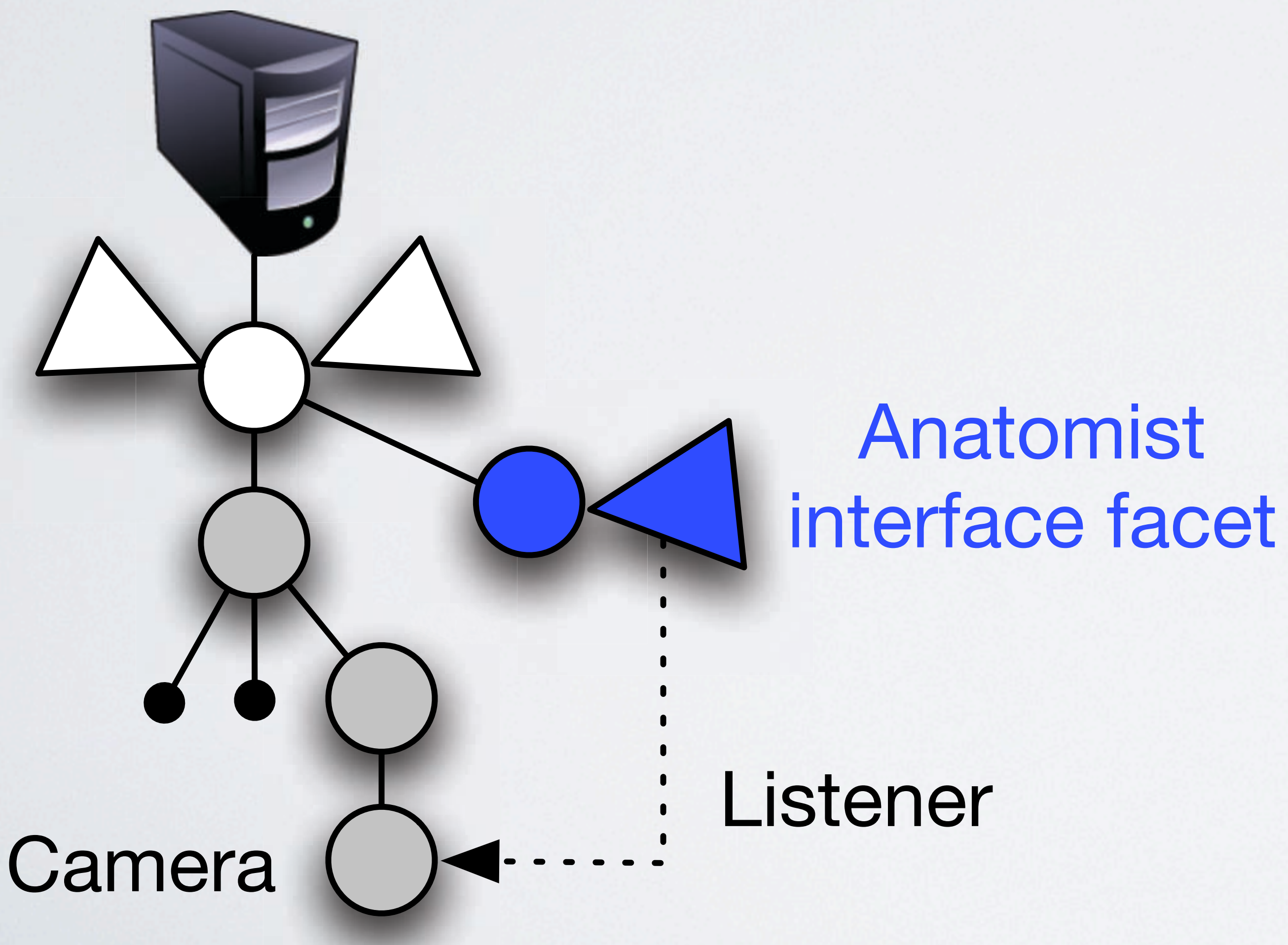


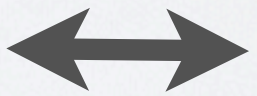
Substance Grise

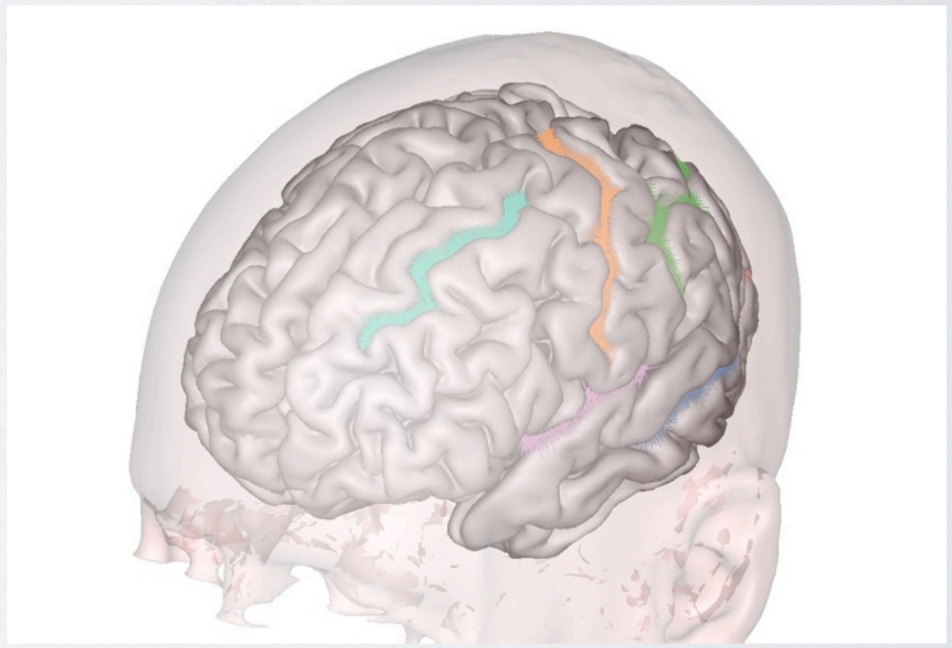


Wrapping legacy applications

- Wrap an existing application in a Substance Environment

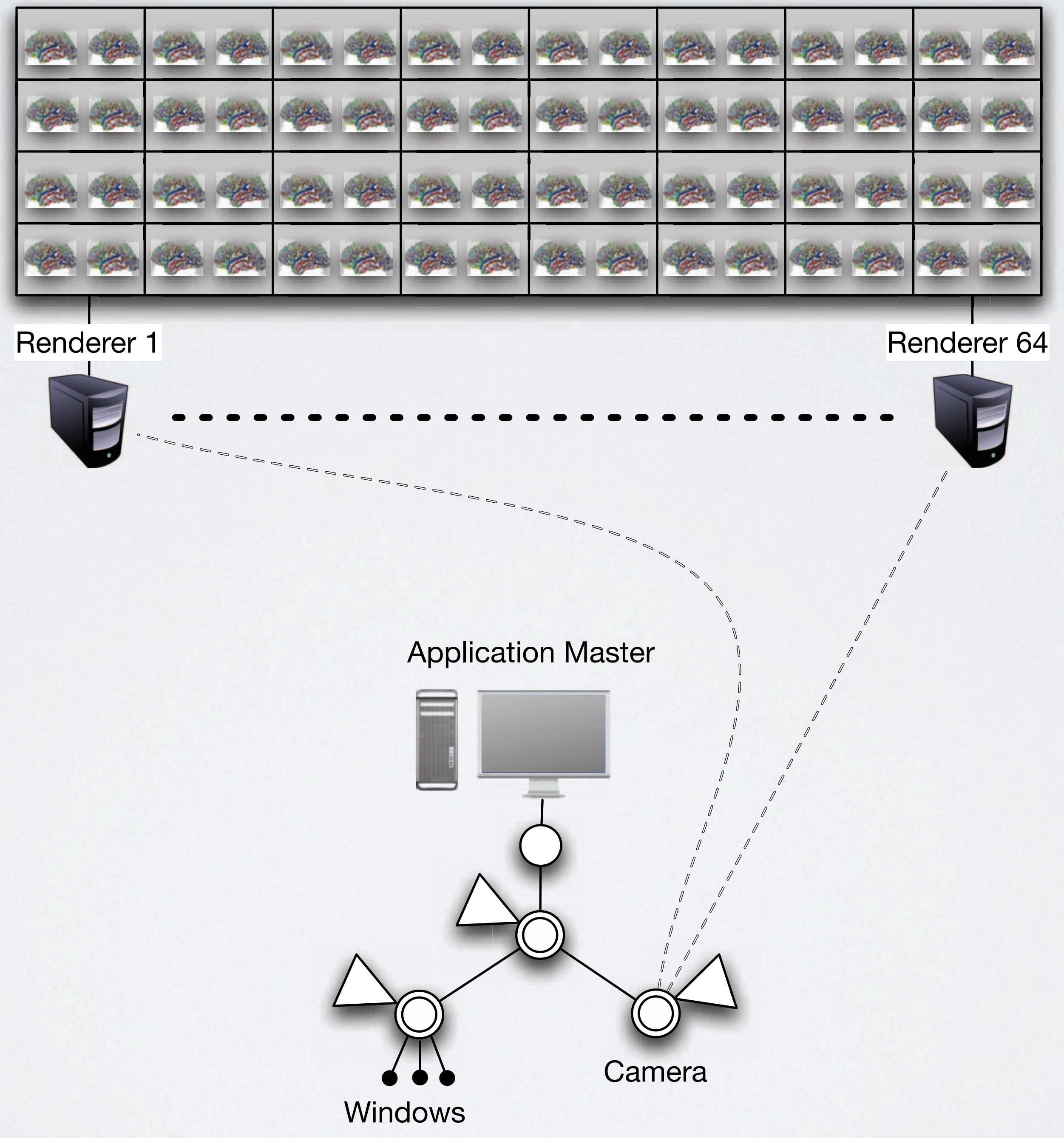




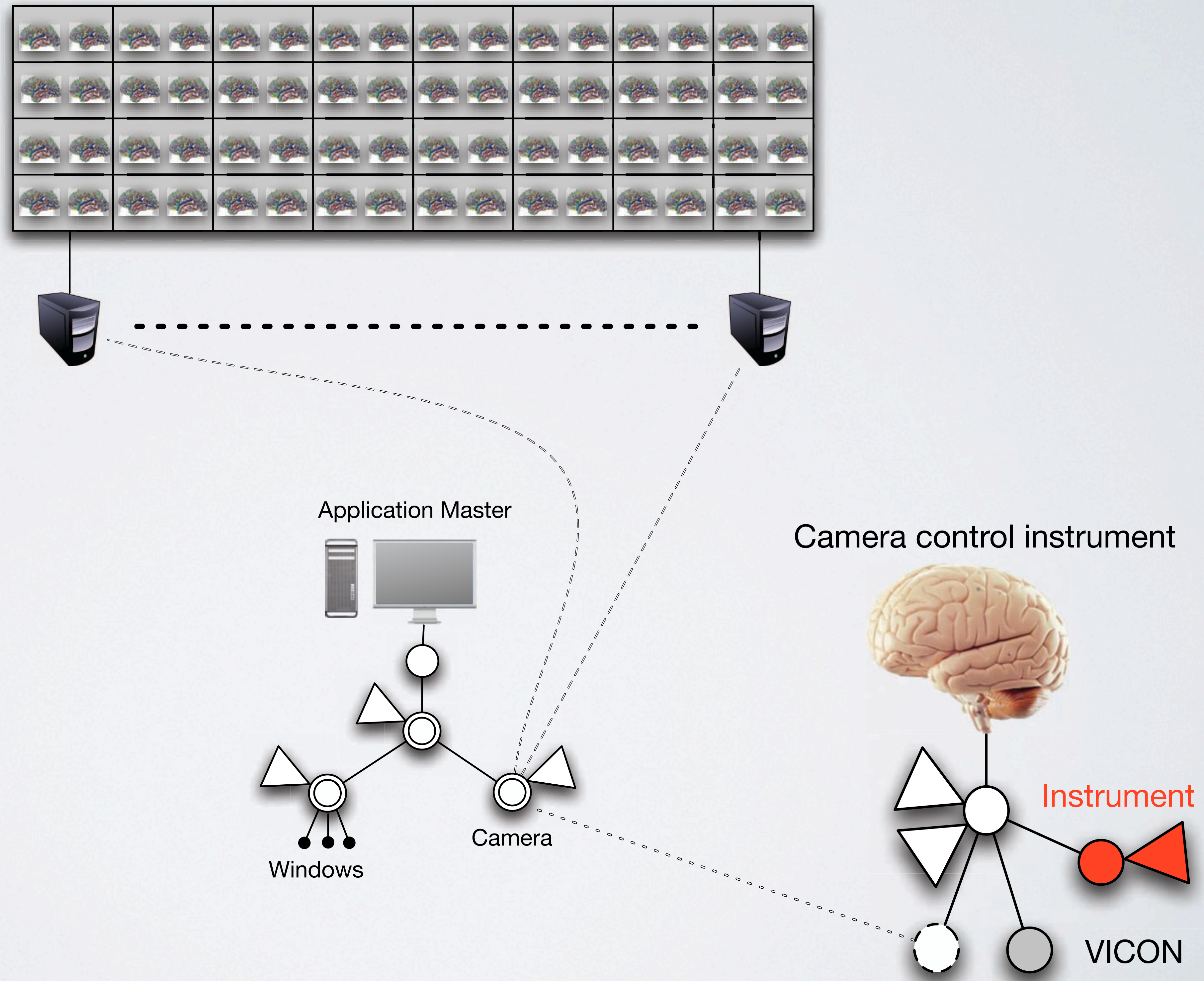


Anatomist application

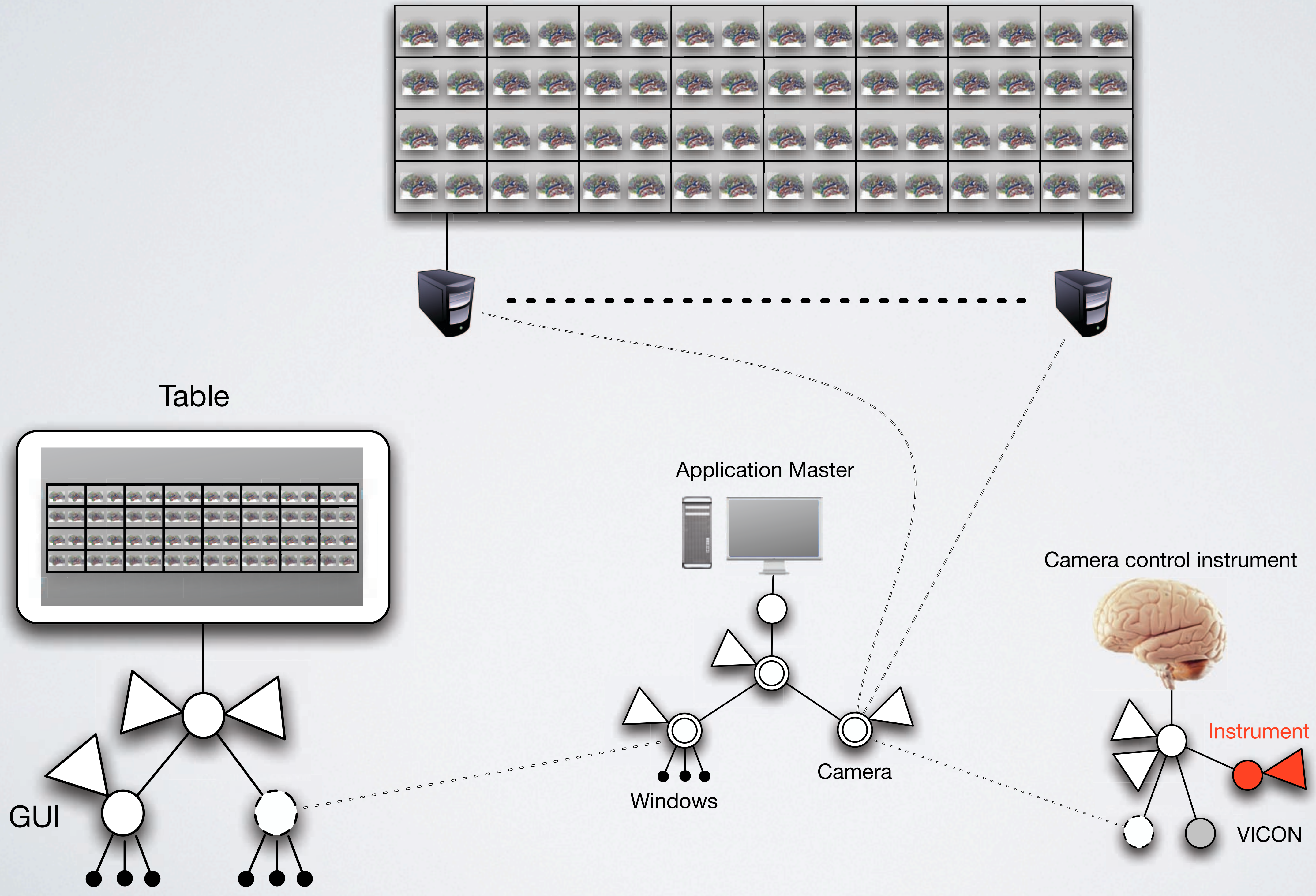
Controlling all the brains at once



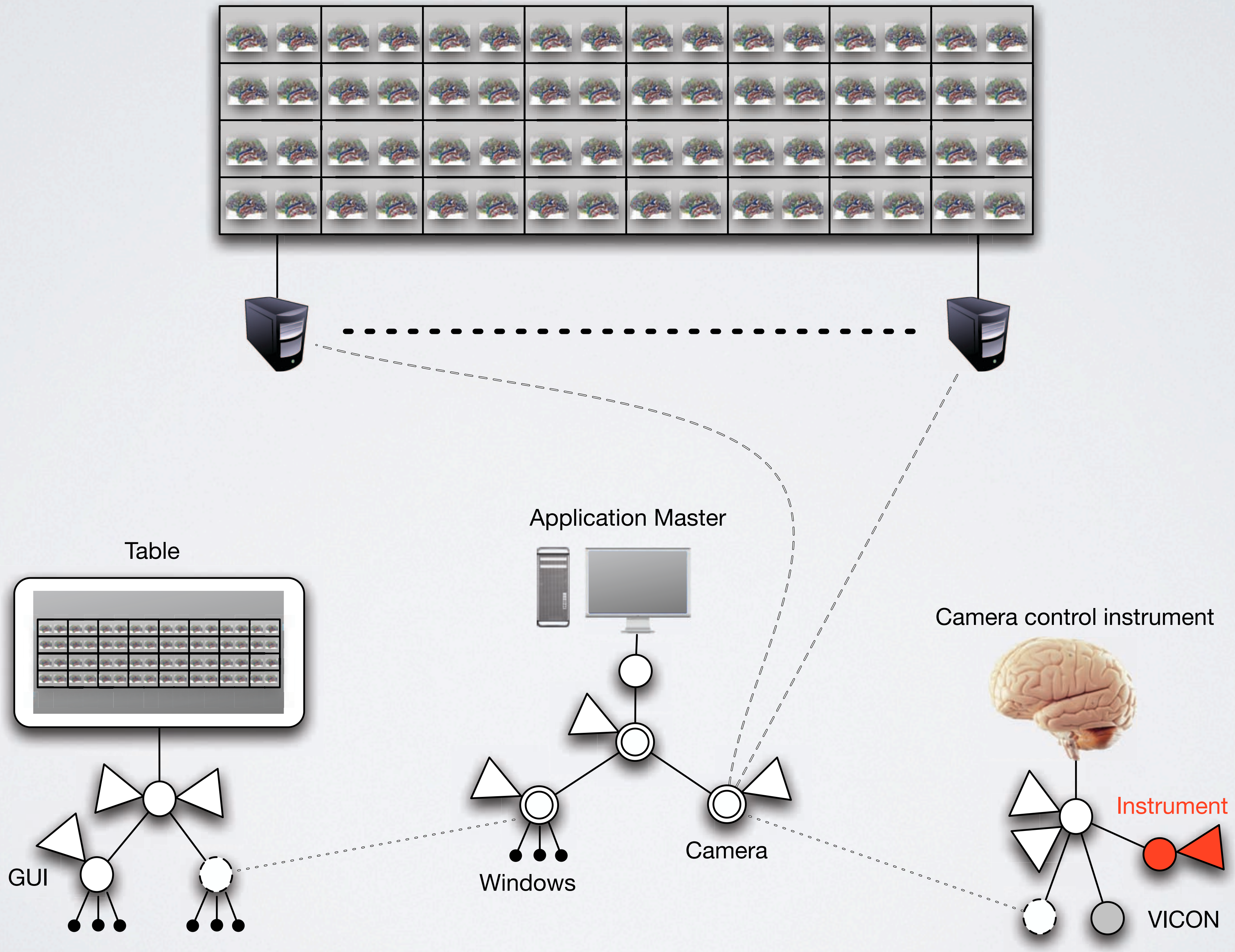
Tangible UI: BCI revisited



Rearranging brains on the table



Substance Grise



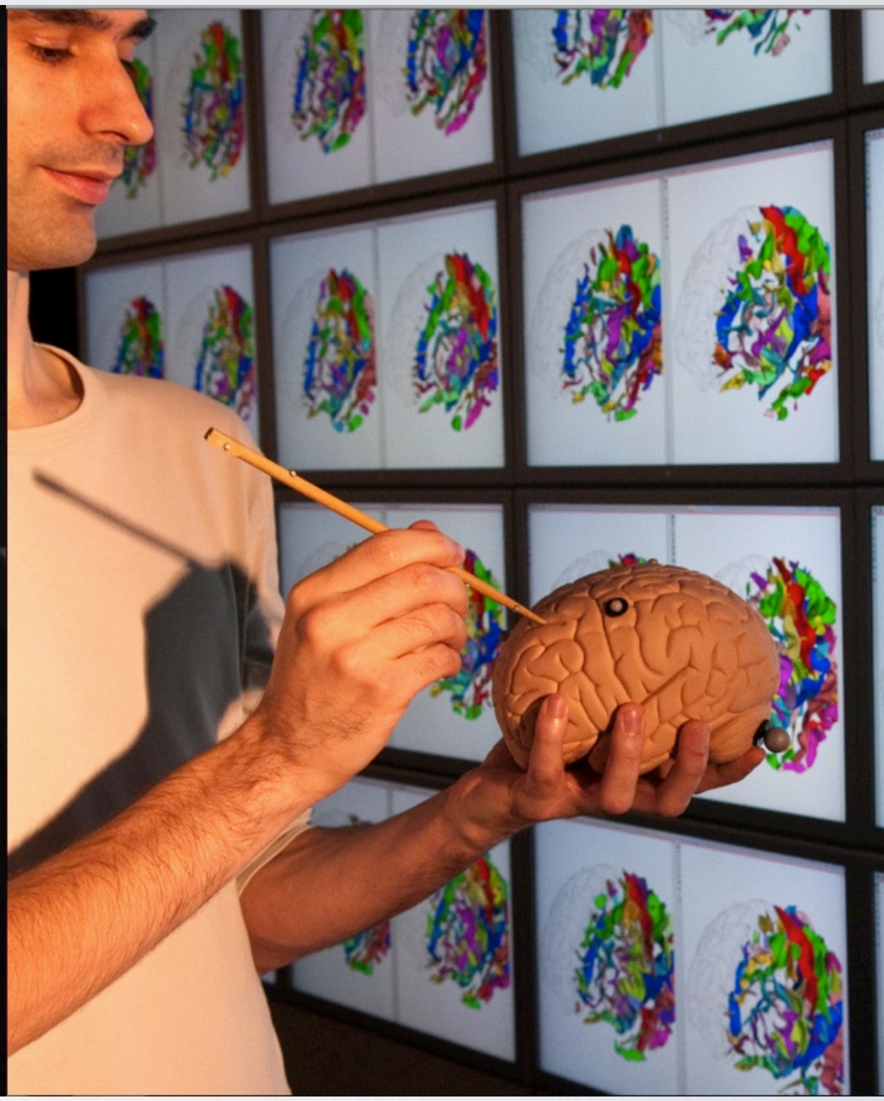
Summary & Contributions

- New programming style: data-orientation
    - Separating data from functionality
- Flexible sharing through replication *and* mounting
    - Supports both a service-oriented approach and a shared-state approach
- Separating instruments from the objects they operate on

Next steps

- Toolkit with generic instruments and content management tools
- Scalability to large distributed systems
- Other application areas
- DIGISCOPE project: remote collaboration across 9 interactive rooms

Questions?



|in|**situ**| - insitu.lri.fr