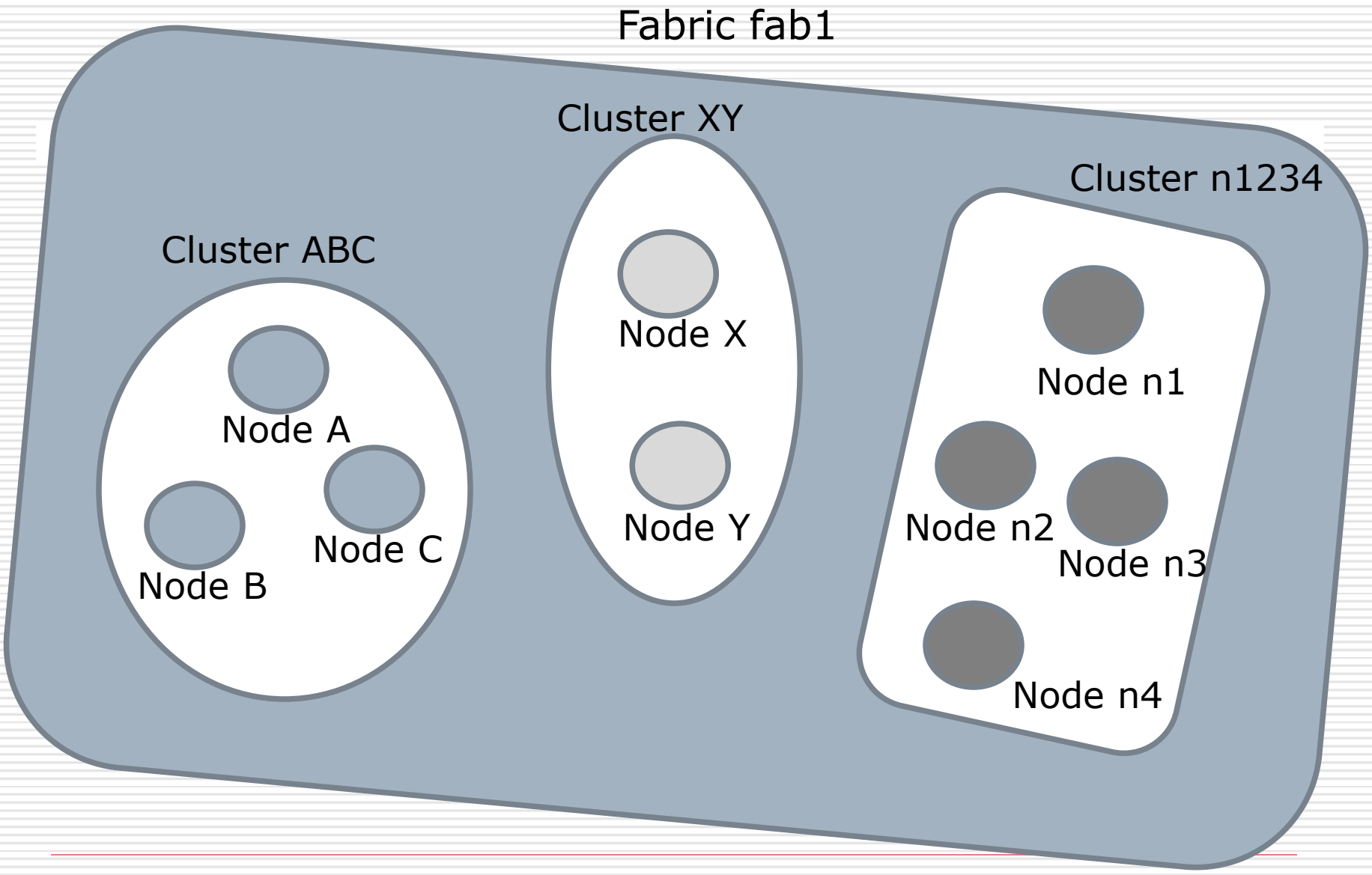
# Language Fabric

Waseda University
Maruyama Fujio

# Agenda

- Node, Cluster and Fabric
- Pipe and Queue
- Weave a Fabric
- Service
- Service on a Fabric
- Deploy a Servic
- Pattern
- Describe distributed system in Fabric using Pattern

# Node, Cluster and Fabric


Fabric fab1
Cluster XY
Cluster n1234
Cluster ABC
Node X
Node n1
Node A
Node n2
Node C
Node n3
Node B
Node Y
Node n4

# Node, Cluster and Fabric

- Node is a Node (Atomic)
- Cluster is a collection of nodes.
- Fabric is a collection of clusters and Nodes.

# Node definition

```
Node node-name {
    node-name();  // Node Constructor
    ……
    ……
}
```

# Cluster definition

```
Cluster cluster-name {
    Node node-name1;
    Node node-name2;
    Node node-name3;
    ……
    cluster-name();  // Cluster Constructor
}
```

# Fabric definition

**Fabric** fabric-name {

**Cluster** cluster-names1 | cluster-def;

**Cluster** cluster-names2 | cluster-def;

**Cluster** cluster-names3 | cluster-def;

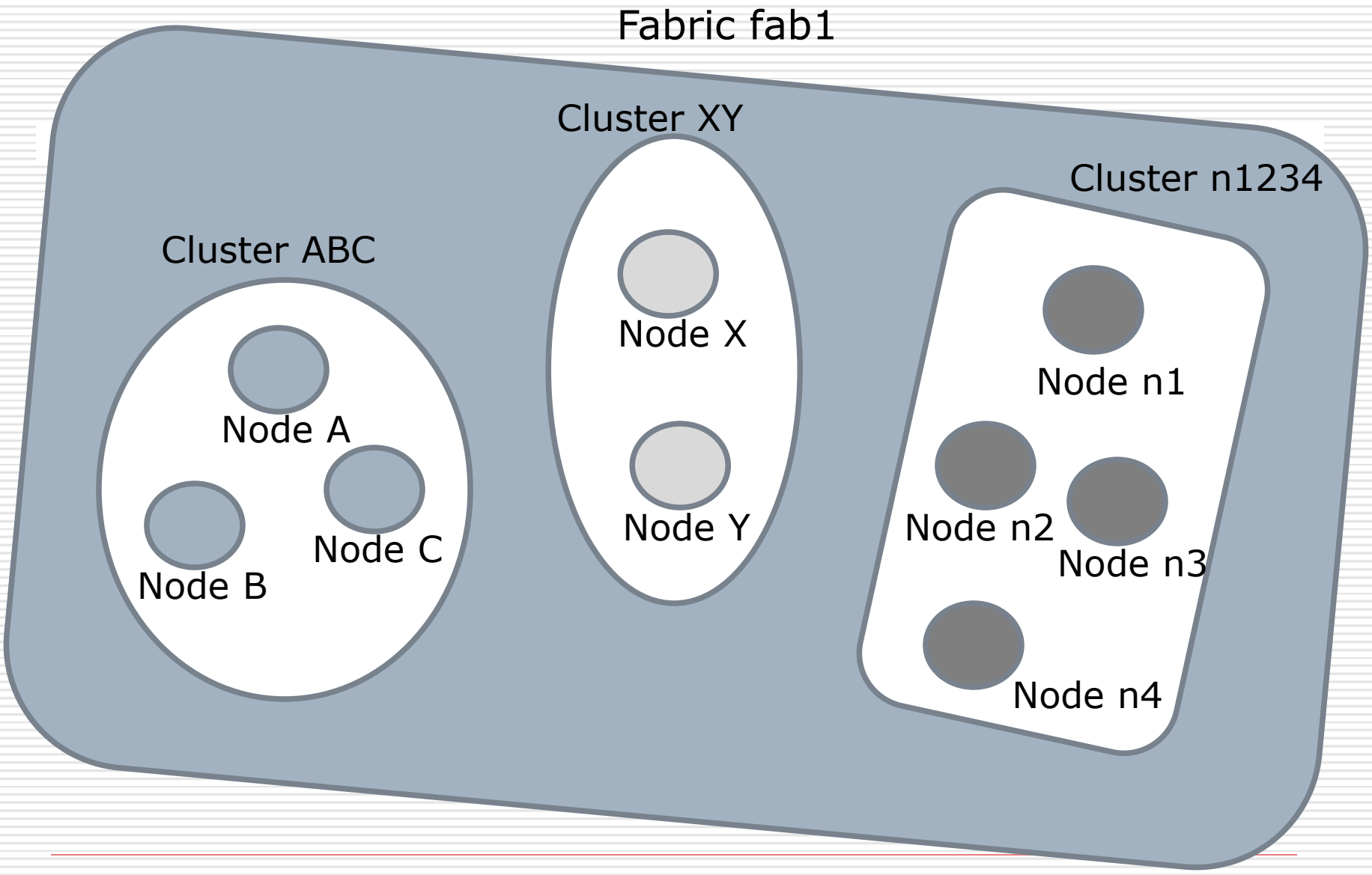
……

**Node** node-name | node-def;

……

fabric-name(); // Fabric Constructor

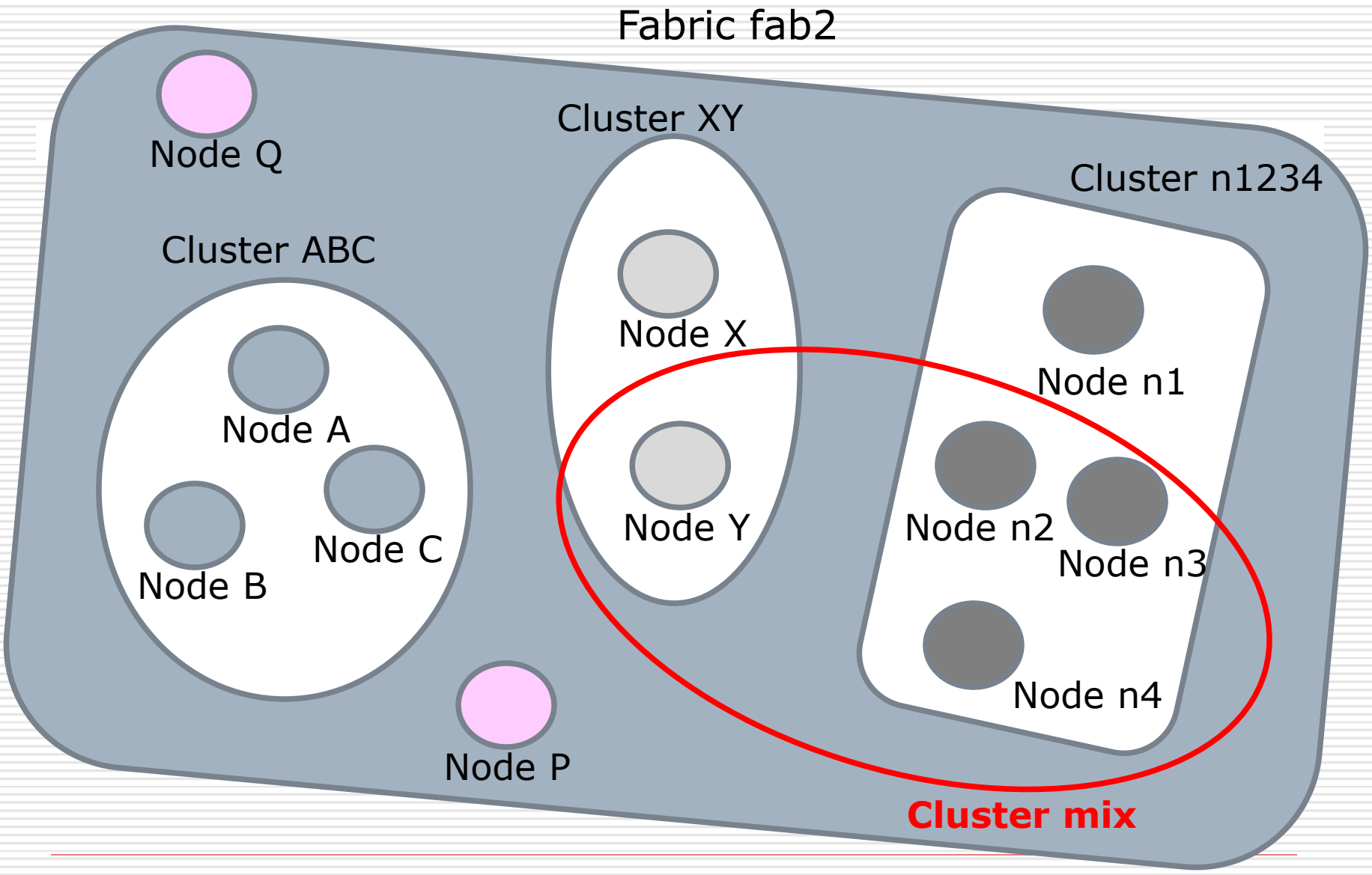
}


Fabric fab1
Cluster XY
Cluster n1234
Cluster ABC
Node X
Node n1
Node A
Node n2
Node C
Node n3
Node B
Node Y
Node n4

# Example1

*Node-definition of A,B,C,……,n3,n4*

```
Fabric fab1 {
  Cluster ABC {
     Node A,B,C; ….
  }
  Cluster XY {
     Node X,Y; …
  }
  Cluster n1234 {
     Node n1,n2,n3,n4; …
  }
}
```


Fabric fab2
Node Q
Cluster XY
Cluster n1234
Cluster ABC
Node X
Node n1
Node A
Node Y
Node n2
Node C
Node n3
Node B
Node n4
Node P
Cluster mix

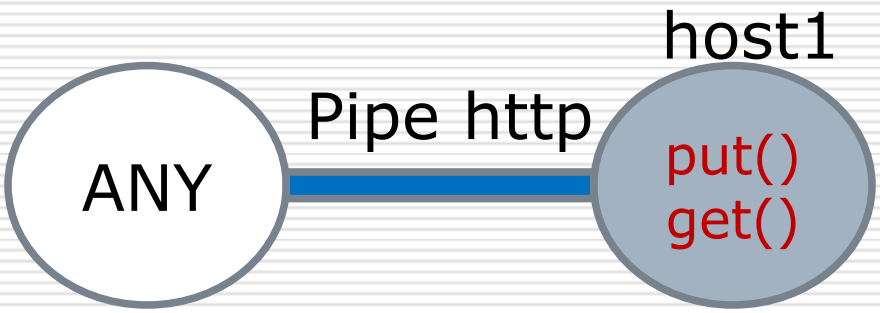
# Example2

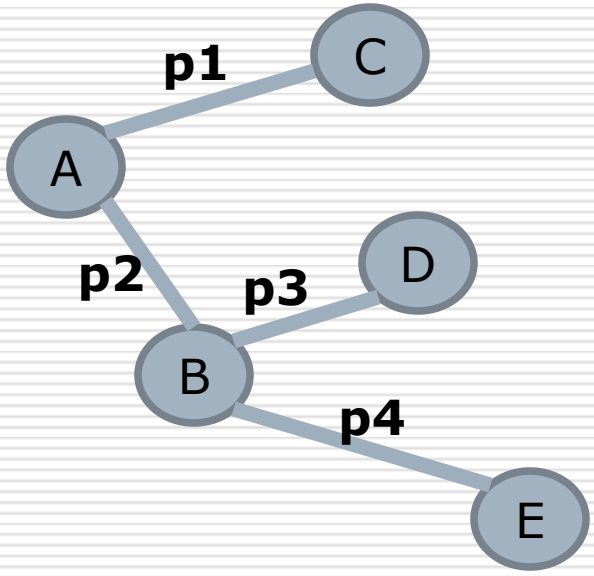
Node definition  A,B, …,P,Q

Cluster ABC { Node A,B,C; … }
Cluster XY { Node X,Y; … }
Cluster n1234 { Node n1,n2,n3,n4; … }

Fabric fab2 {
  Cluster ABC, XY, n1234;
  Cluster mix
    Node Y, n2,n3,n4
   }
  Node P,Q;
}

# Pipe and Queue



# Pipe

- Node can have a Pipe.
- Node can have many Pipes.
- Pipe is One-to-One bi-directional connection to other Node.
- Node must have two methods.
  - get(pipe)
  - put(pipe,data)

# Pipe definition

**[Lasy] Pipe** pipe-name {

**between** node-name1 (caller)

**and** node-name2 (callee)

pipe-name-**c**(); // Constructor caller

pipe-name-**s**(); // Constructor callee

}

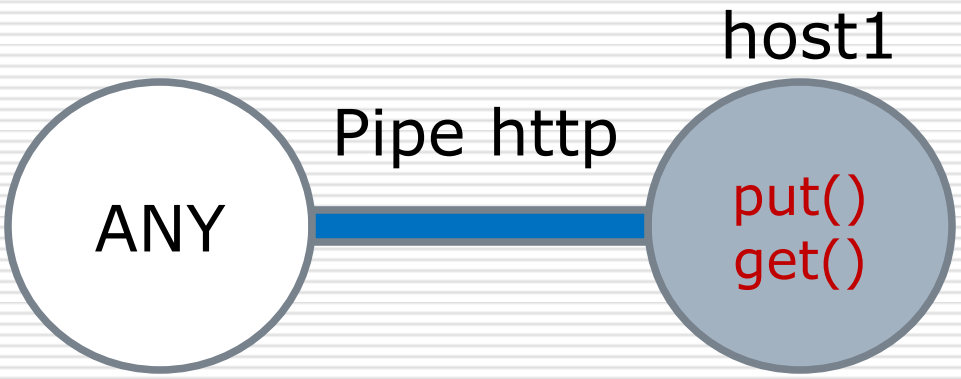
Specail Node name …… **ANY**

# Node’s Method

- get(*pipe*)
- put(*pipe,data*)

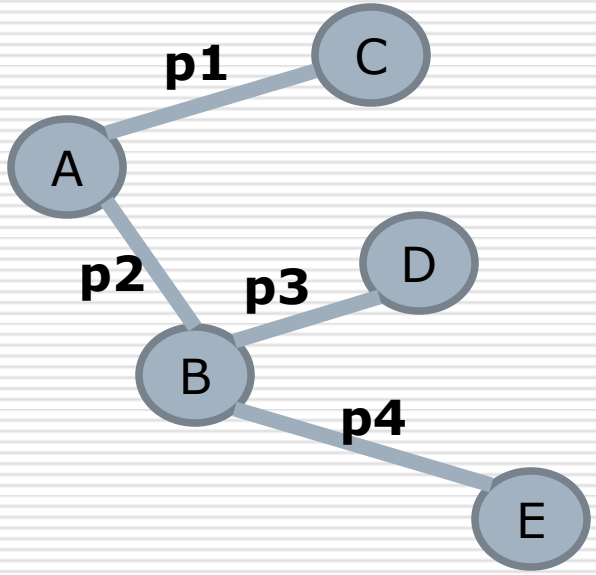
# Example1

```
Node host1 {
  host1();
  get(){……}
  put(pipe){……}
}
Lazy Pipe http {
  between ANY and host1;
  http(…);
}
Fabric example1 {
  Node Any,host1;
  Pipe  http;
}
```



# Example2

```
Fabric example2 {
  Node A,B,C,D;
  Pipe  p1{ between A and C}
  Pipe  p2{ between A and B}
  Pipe  p3{ between B and D}
  Pipe  p4{ between B and E}
}
```



# Queue

- Cluster can have a Queue.
- Cluster can have many Queues.
- Queue is One-to-One one-directional connection to other Cluster.
- Cluster must have following methods.
  - put(queue) : From Cluster
  - get(queue,data) : To Cluster

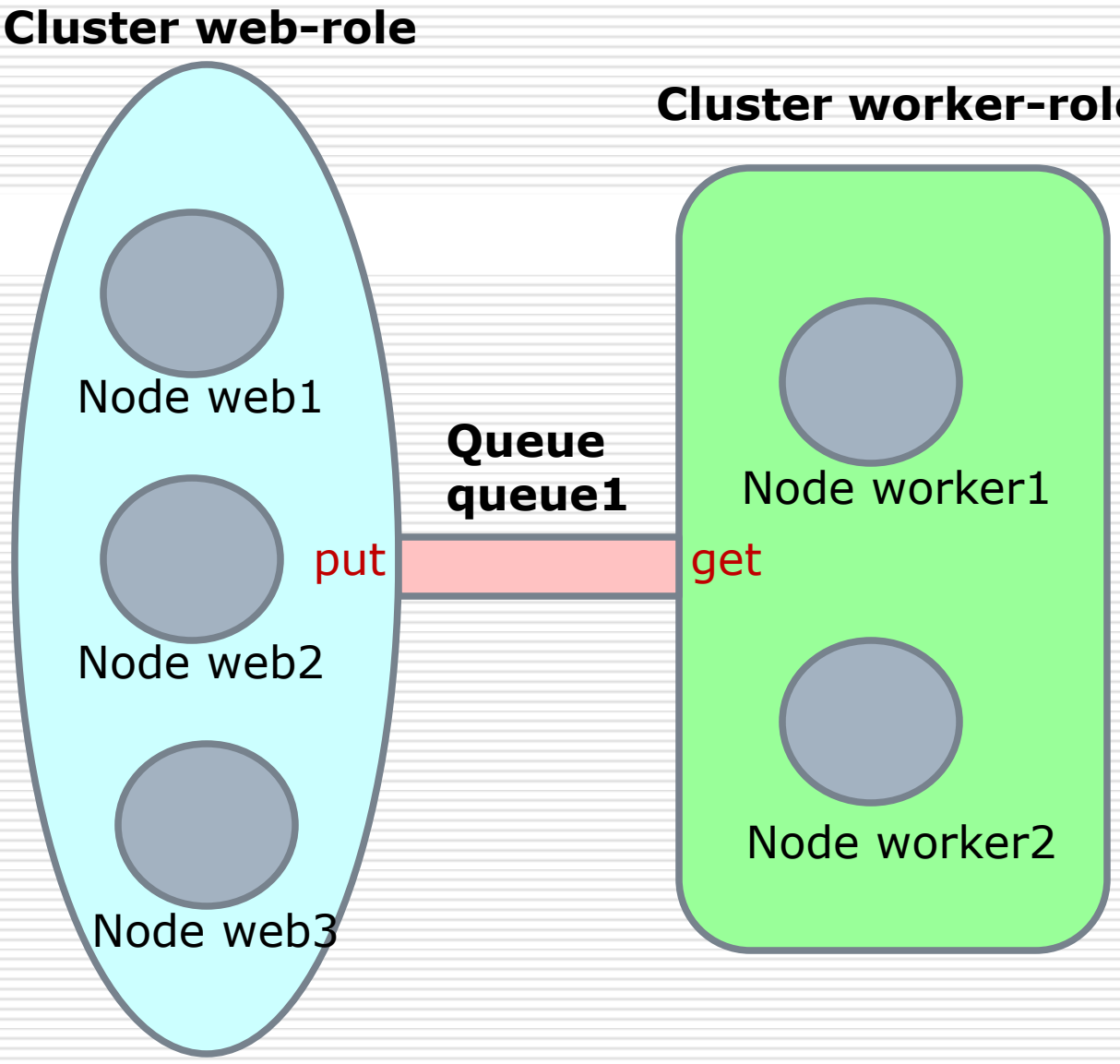
# Queue definition

```
Queue queue-name {
        from    cluster-name1
        to      cluster-name2

        queue-name()  // Constructor
}
```

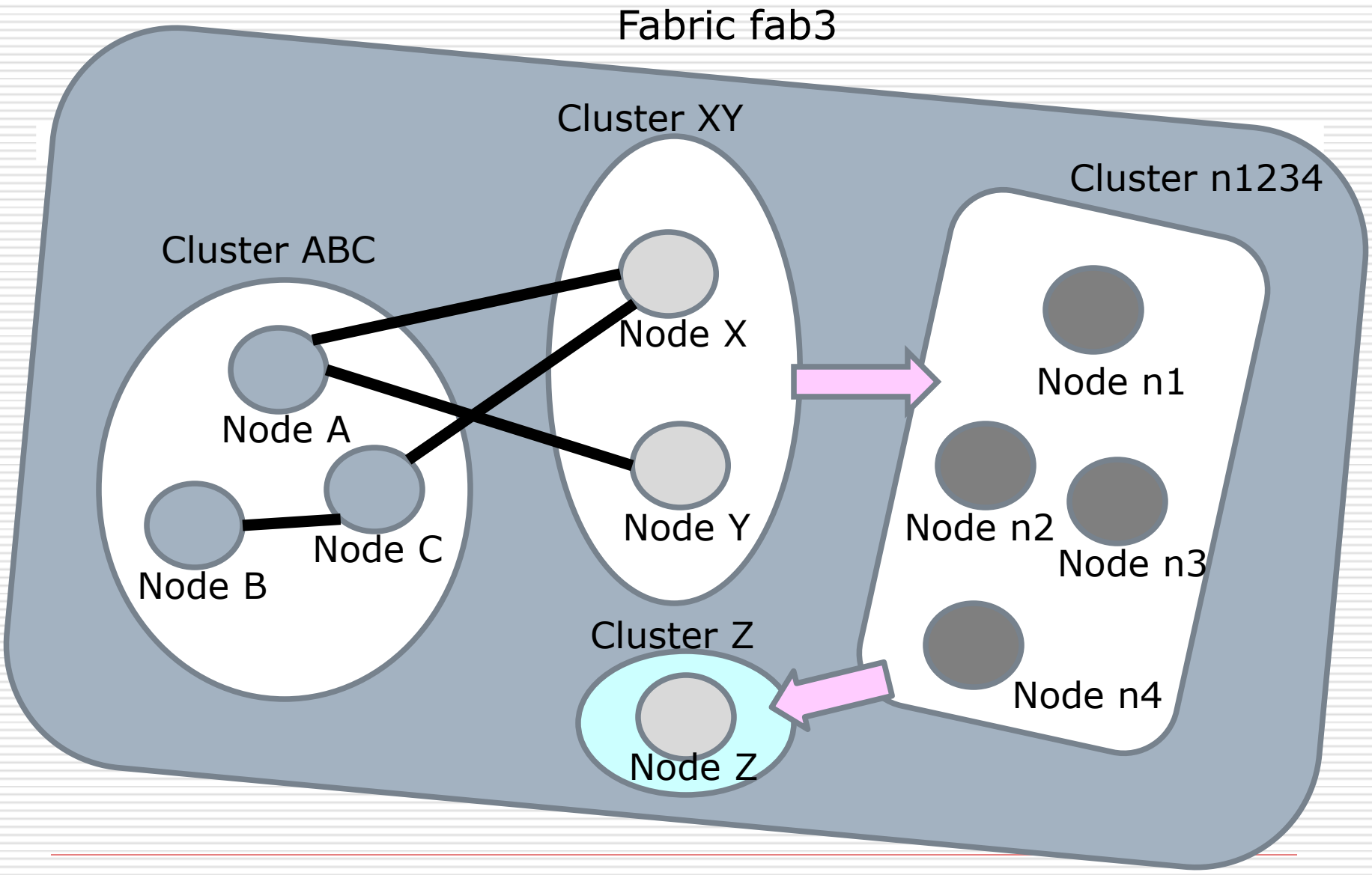
# Cluster's Method

- get(*queue*)
- put(*queue,data*)


Cluster web-role
Cluster worker-role
Node web1
Node web2
Node web3
Queue
queue1
put
get
Node worker1
Node worker2

# Example3

```
Fabric example3 {
  Cluster web-role {
    Node web1,web2,web3;
    web-role();……
  }
  Cluster worker-role {
    Node worker1, worker2;
    worker-role();……
  }
  Queue queue1{
     from web-role to worker-role;
     queue1();……
  }
}
```

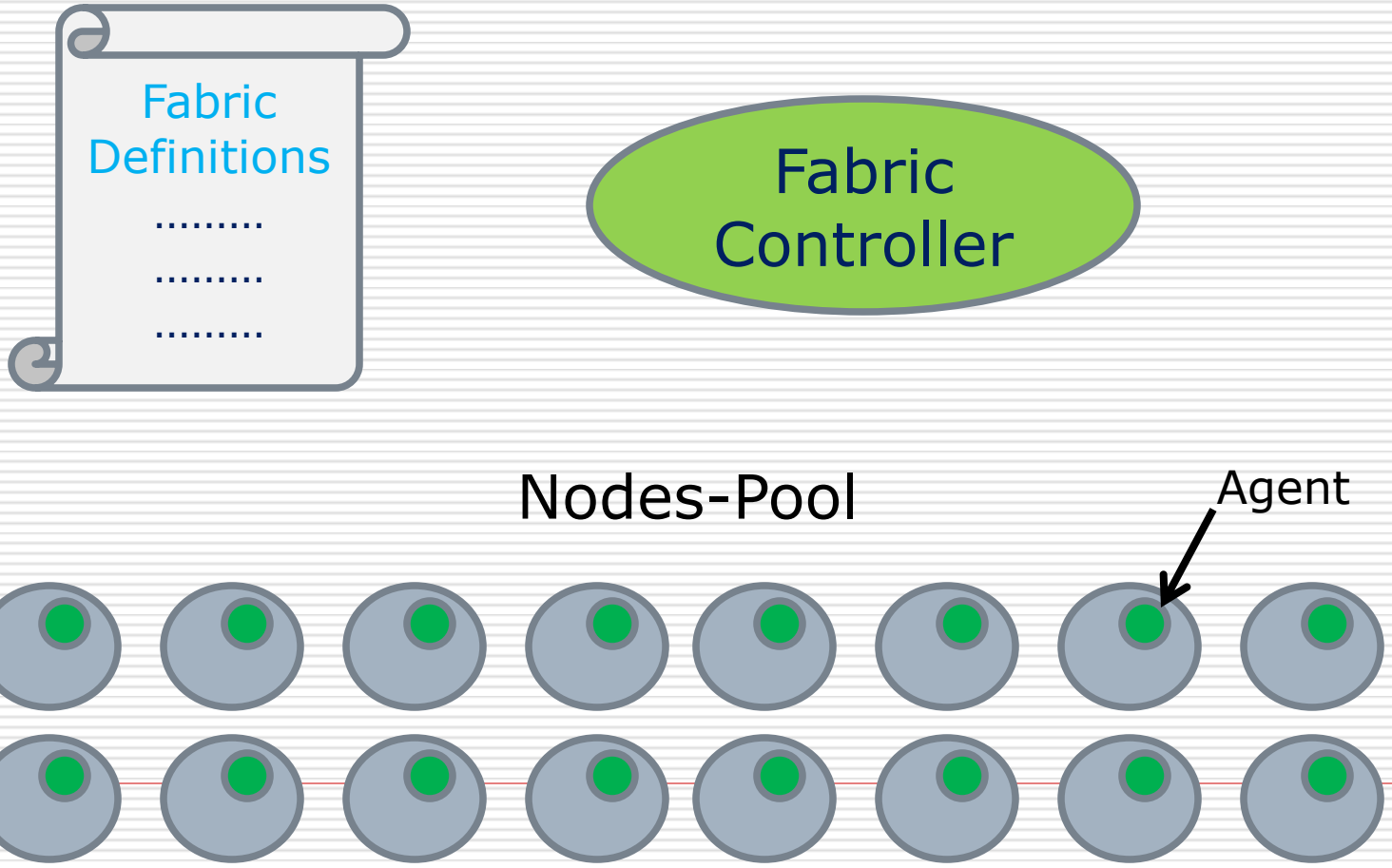

Fabric fab3
Cluster XY
Cluster n1234
Cluster ABC
Node X
Node n1
Node A
Node n2
Node C
Node n3
Node B
Node Y
Cluster Z
Node n4
Node Z

# Example4

```
Node A {  } …. // Definition of Nodes ;
 ……
Pipe p1 { between B and C; … }  // Definition of Pipes ;
 ……
Cluster Z { Node Z; …. } // Definition of Clusters ;
 ……
Queue q1 { from XY to n1234; … } // Definition of Queue ;
 ……
Fabric fab3 {
  Cluster ABC,XY,Z,n1234;
  Pipe p1,p2,p3,p4;
  Queue q1,q2;
}
```

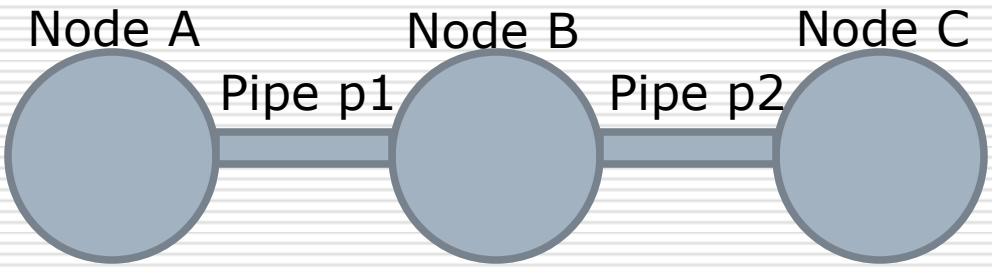
# Weave a Fabric

# Fabric definition, Fabric Controller and Nodes-Pool



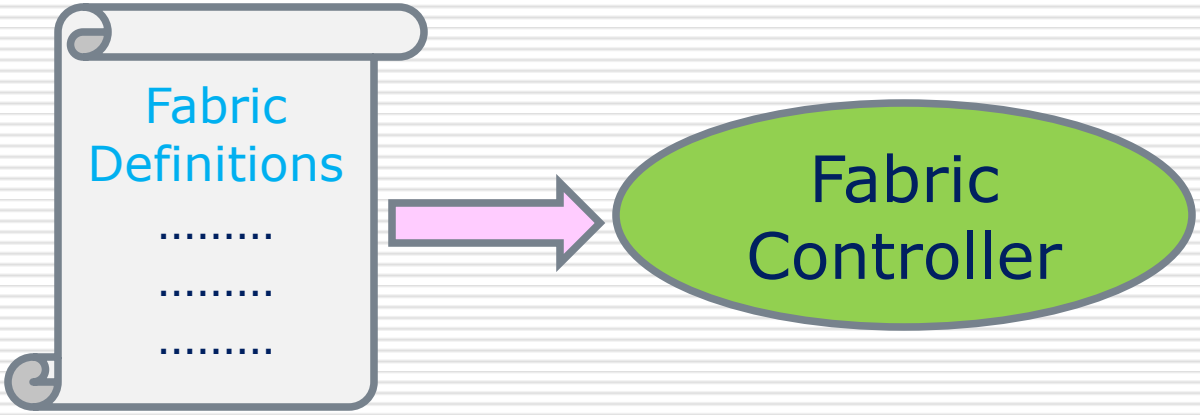
# 3-nodes Sample

```
Fabric 3-nodes {
   Node A { A(){….} // Constructor of Node A }
   Node B { B(){….} // Constructor of Node B }
   Node C { C(){….} // Constructor of Node C }
   Pipe p1 { between A and B ; p1x(){….} // Constructor of Pipe p1 }
   Pipe p2 { between B and C ; p2x(){….} // Constructor of Pipe p2 }
}
```

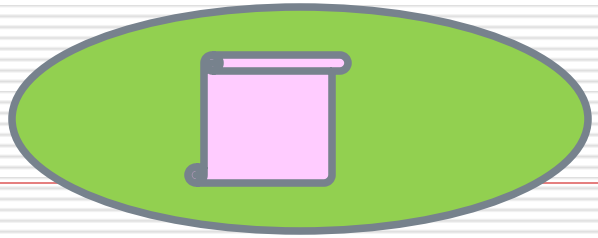


# Weaving (Fabric Controller)

- Read the Fabric definition



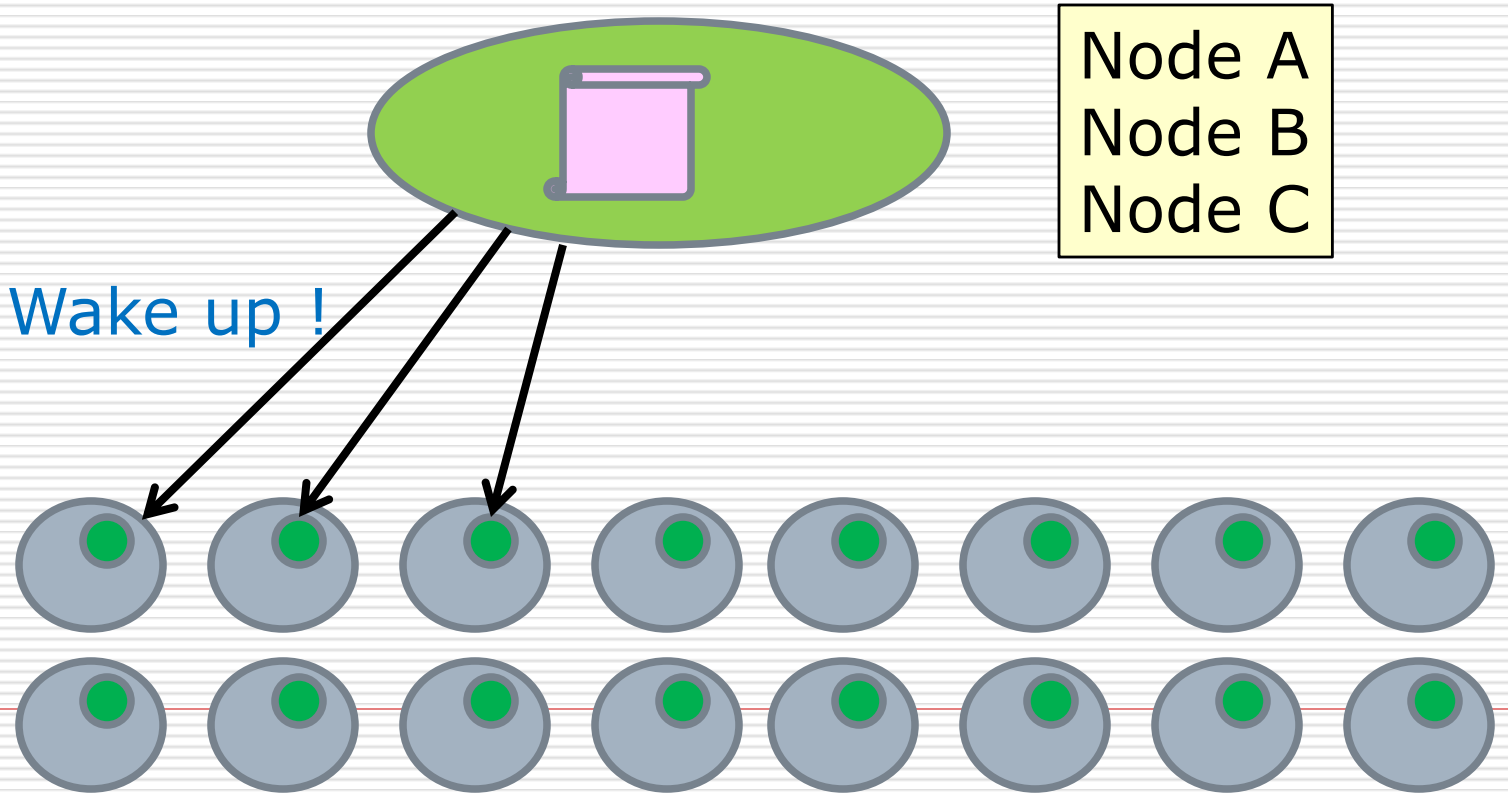

- Enumerate all the nodes in the fabric



Node A
Node B
Node C

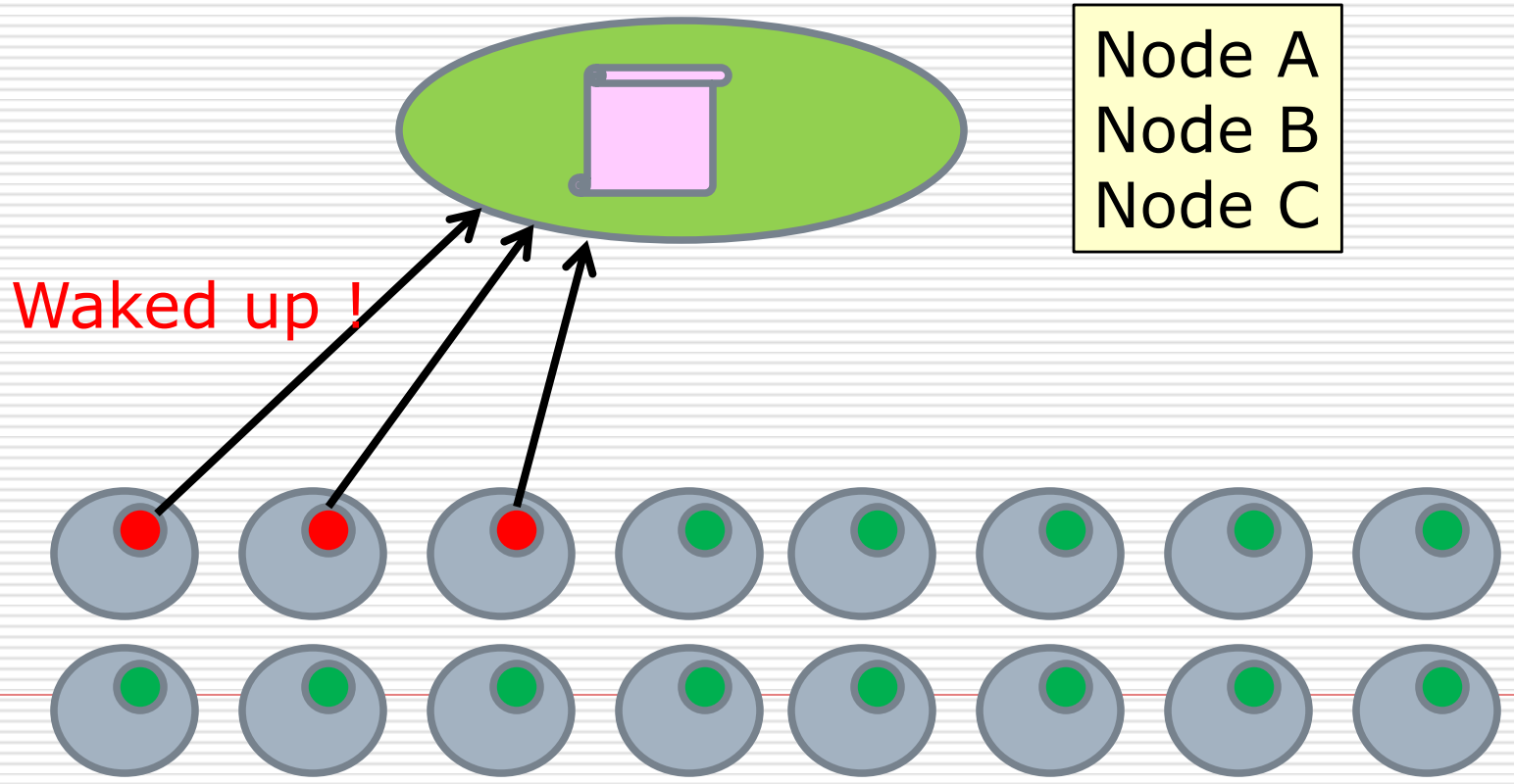
# Weaving (Fabric Controller)

- For nodes in nodes-pool, send ‘Wake up’ Message



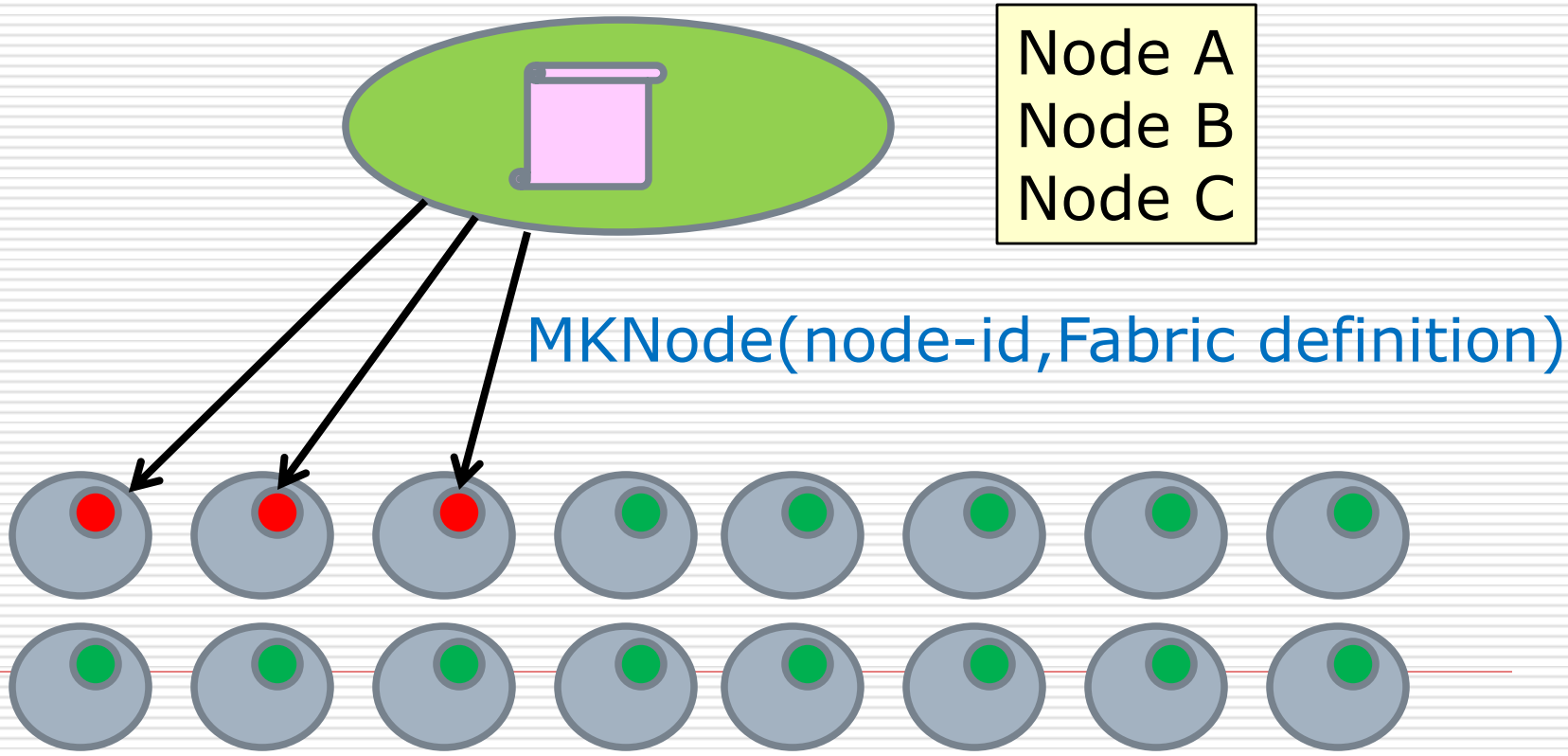
# Weaving (Fabric Controller)

- Node send the 'Waked up' Message to Fabric Controller



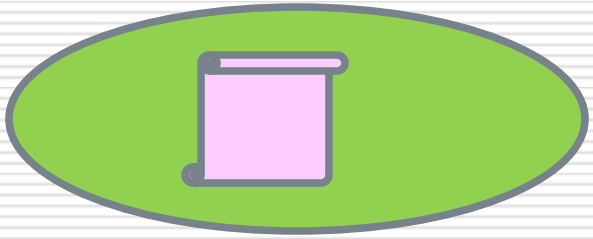
# Weaving (Fabric Controller)

- FC sents to Nodes MKNode (node-id,Fabric-definition) Message.

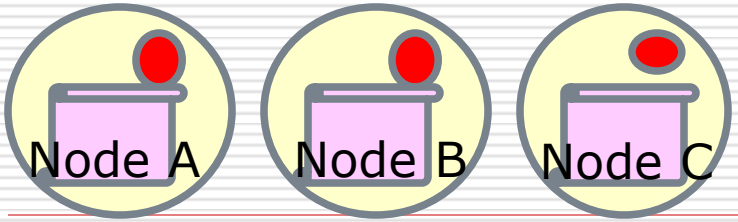


# Weaving

- Nodes receive their node-id and Fabric-definition

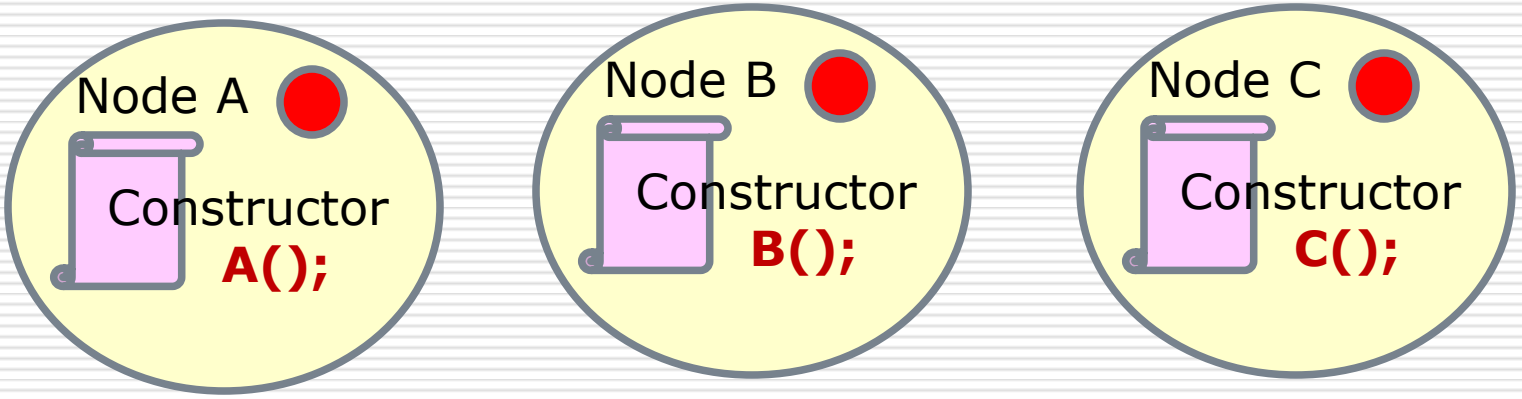


Node A
Node B
Node C



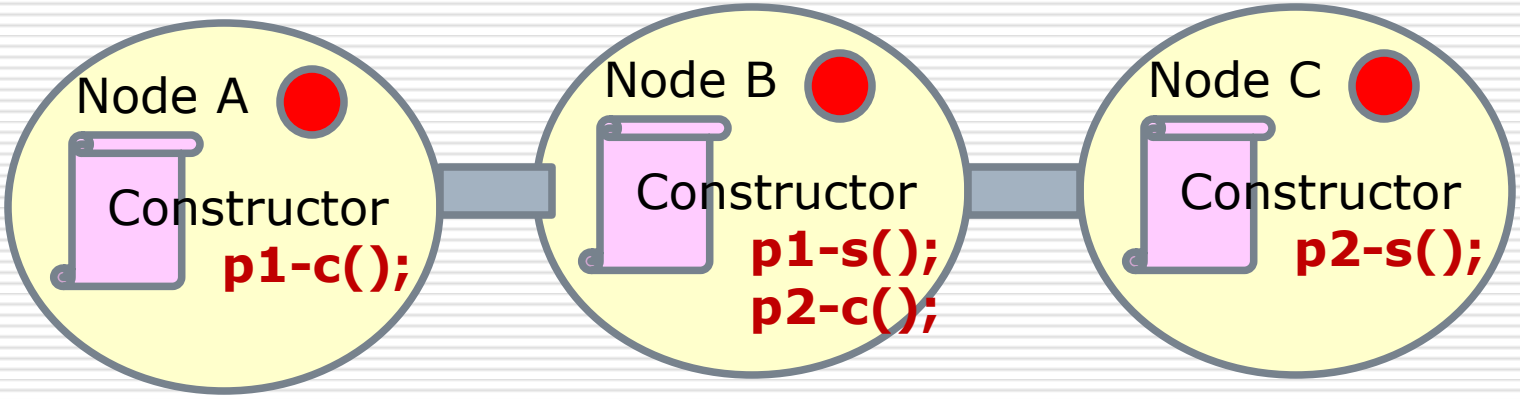
# Weaving (Node Agent)

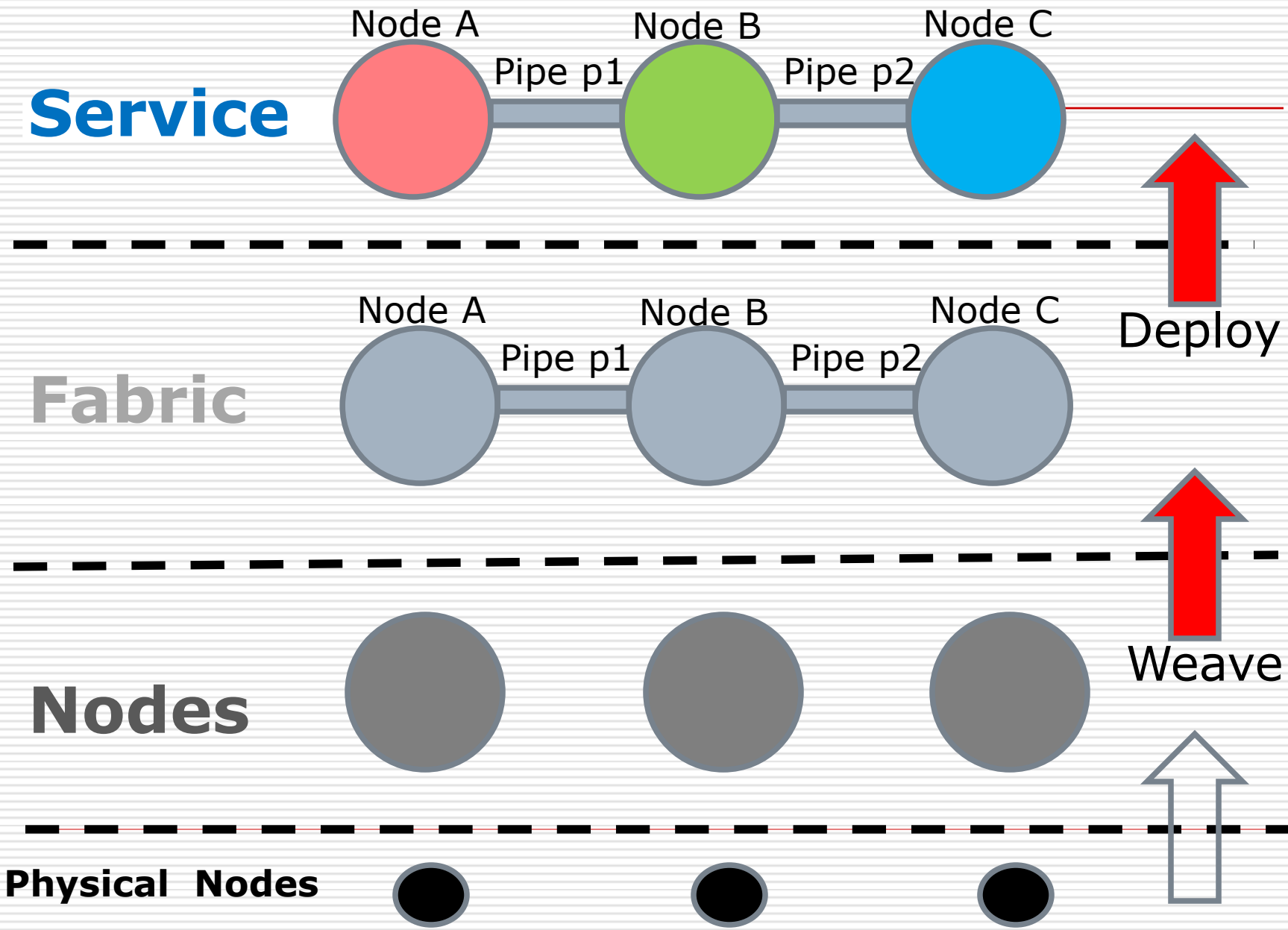
- Interpreting Fabric-dfinition, executes their own node constructer.



# Weaving (Node Agent)

- And make Pipes between nodes, executing Pipe constructor.



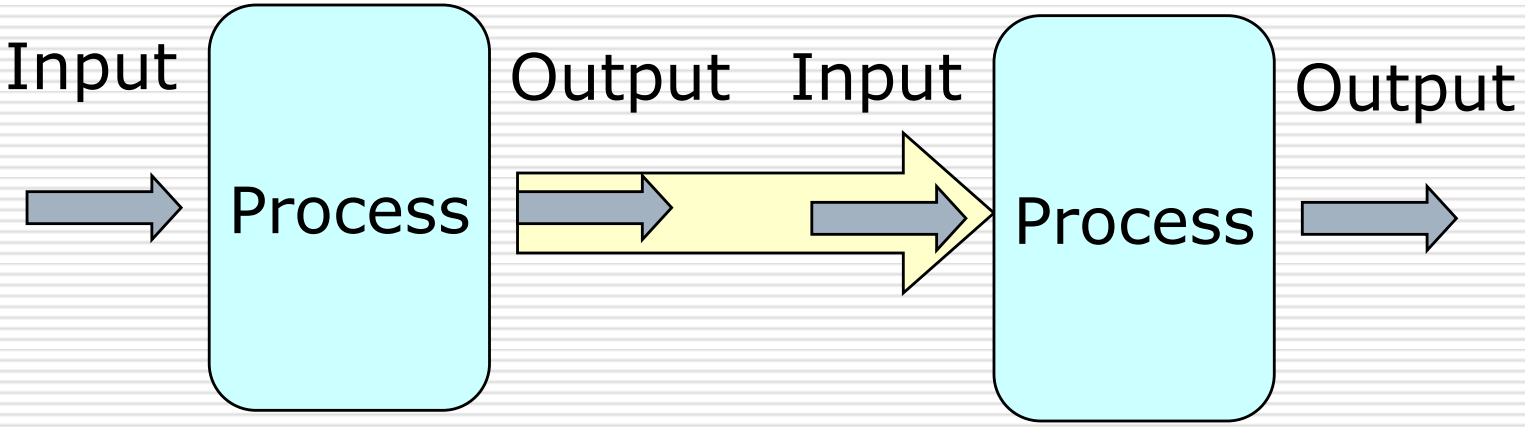

Service
Node A
Pipe p1
Node B
Pipe p2
Node C
Deploy
Fabric
Node A
Pipe p1
Node B
Pipe p2
Node C
Weave
Nodes
Physical Nodes

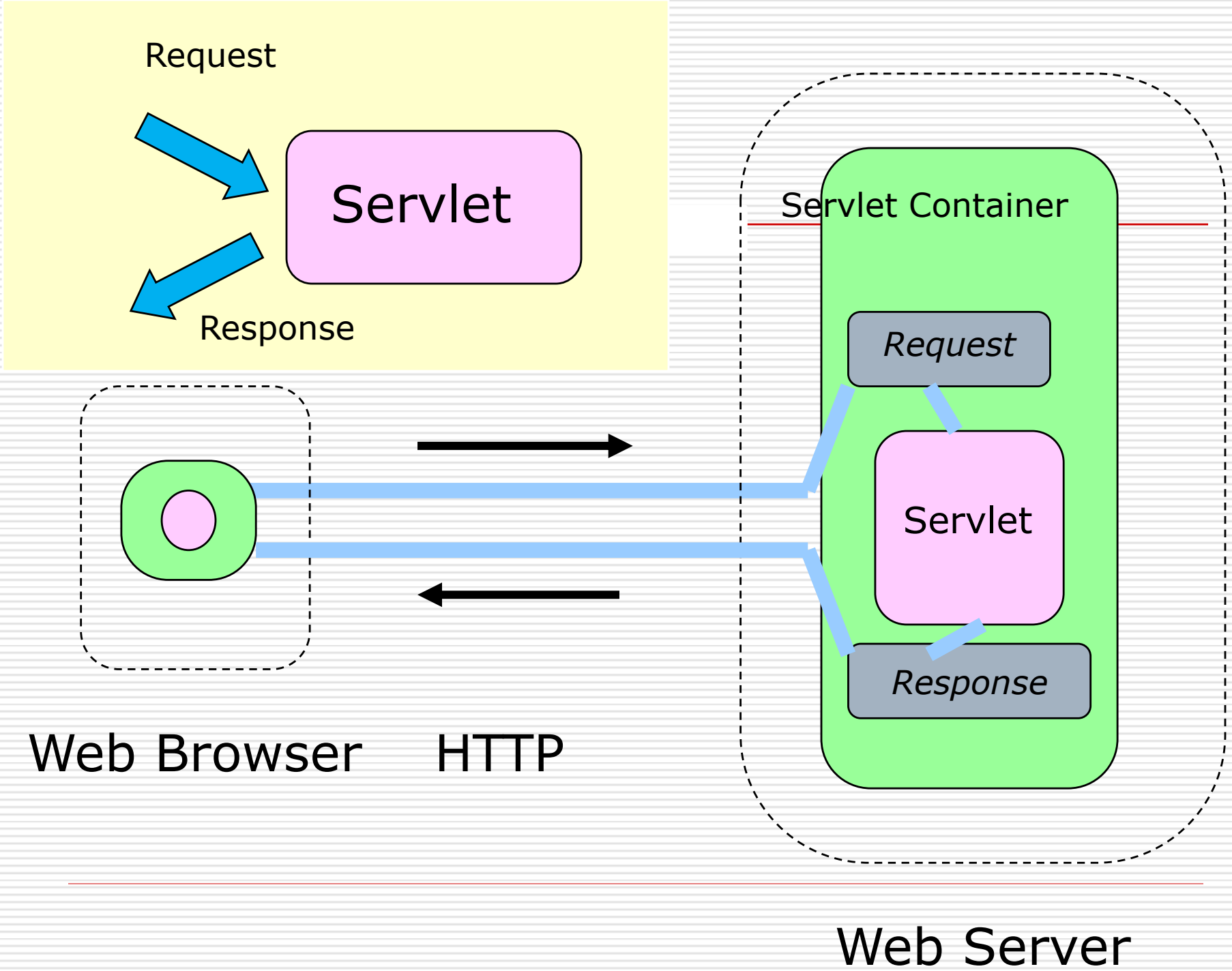
# Service

# What is a Service ?

- Service is a transformation of a message.

# UNIX's Pipe




Request
Servlet
Response
Servlet Container
Request
Servlet
Response
Web Browser
HTTP
Web Server

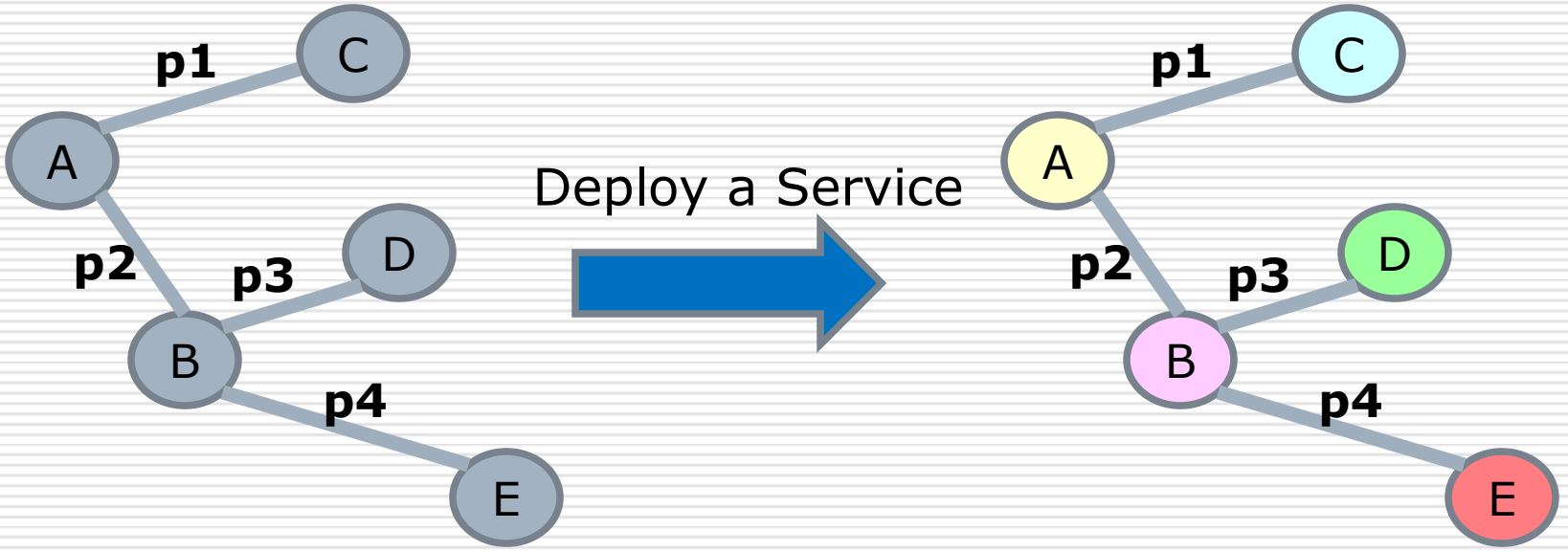
# Functional Language

```
(defun our-length (list)
  (if (null list)    0
     (+ (our-length (rest list)) 1)))
 …..
(our-length ‘(2 3 4))
3

(mapcar
     #'(lambda(num) (+ num 3))
           '(2 3 4))
(5 6 7)
```

# Service on a Fabric

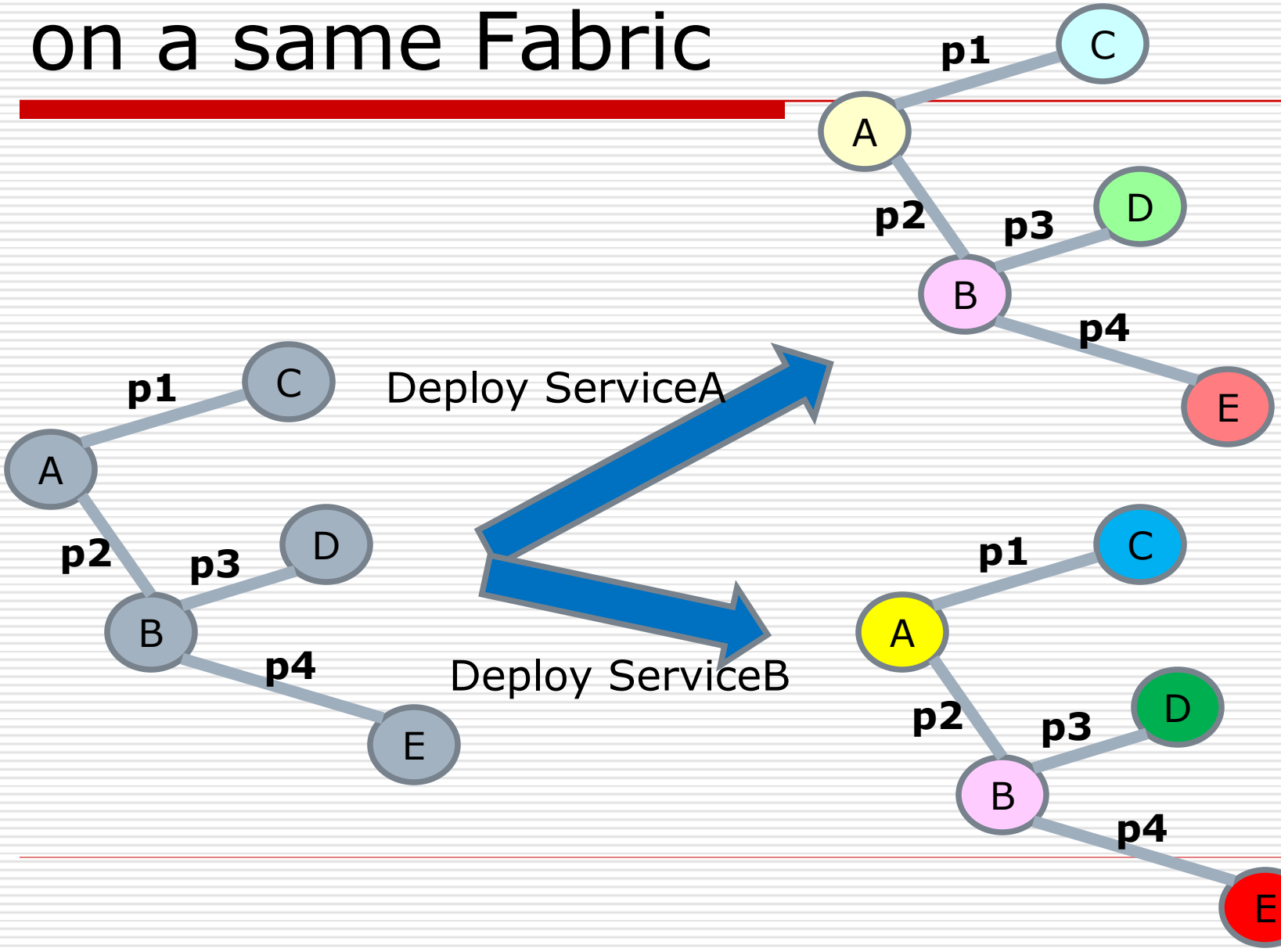
# Service on a Fabric




Fabric

Service on the Fabric

# Two Services on a same Fabric



# Service definition on Node and Cluster

**NodeService** service-name
  **on** node-name {
    service{…}
}
**ClusterService** service-name
  **on** cluster-name {
    service{…}
}

# Service on Fabric

**Service** service-name

**on** fabric-name {

**NodeService** nodeservice-name-lists;

nodeservice-definition-lists**;**

**ClusterService** clusterservice-name-lists ;

clusterservice-definition-lists**;**

# Service's method

**Service** service-name **on** fabric-name {

*Service-list-or-its-definitions;*

weave();

deploy();

start();

end();

undeploy();

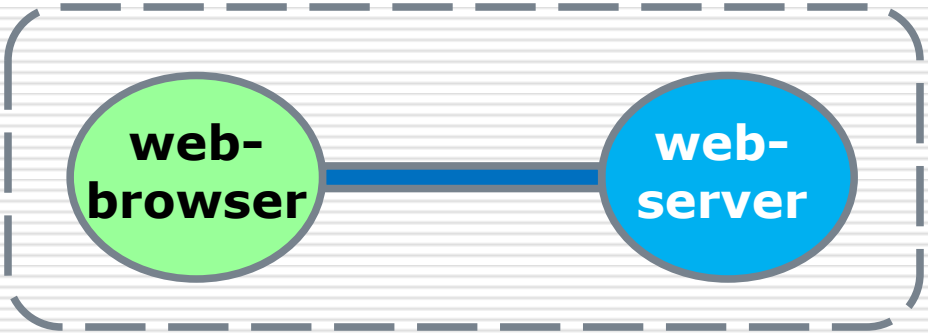
unweave();

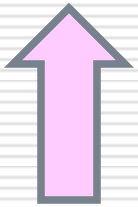
}

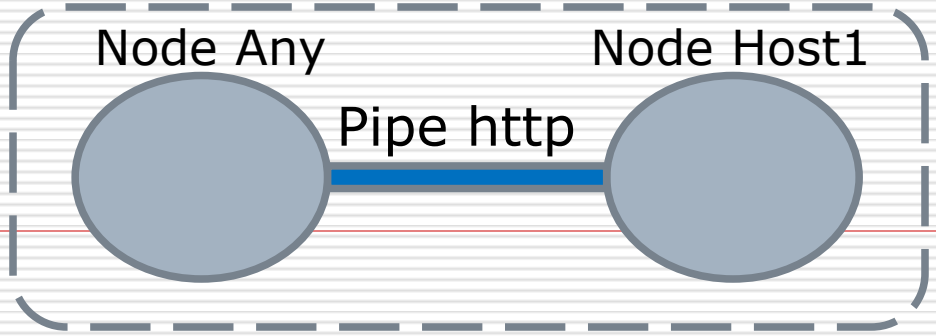
# Example1

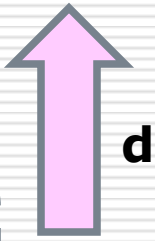
```
Service Web on Web-fabric {
   NodeService web-browser on ANY {…}
   NodeService web-server on Host1 {…}
}
```

**Service** :
**Web**





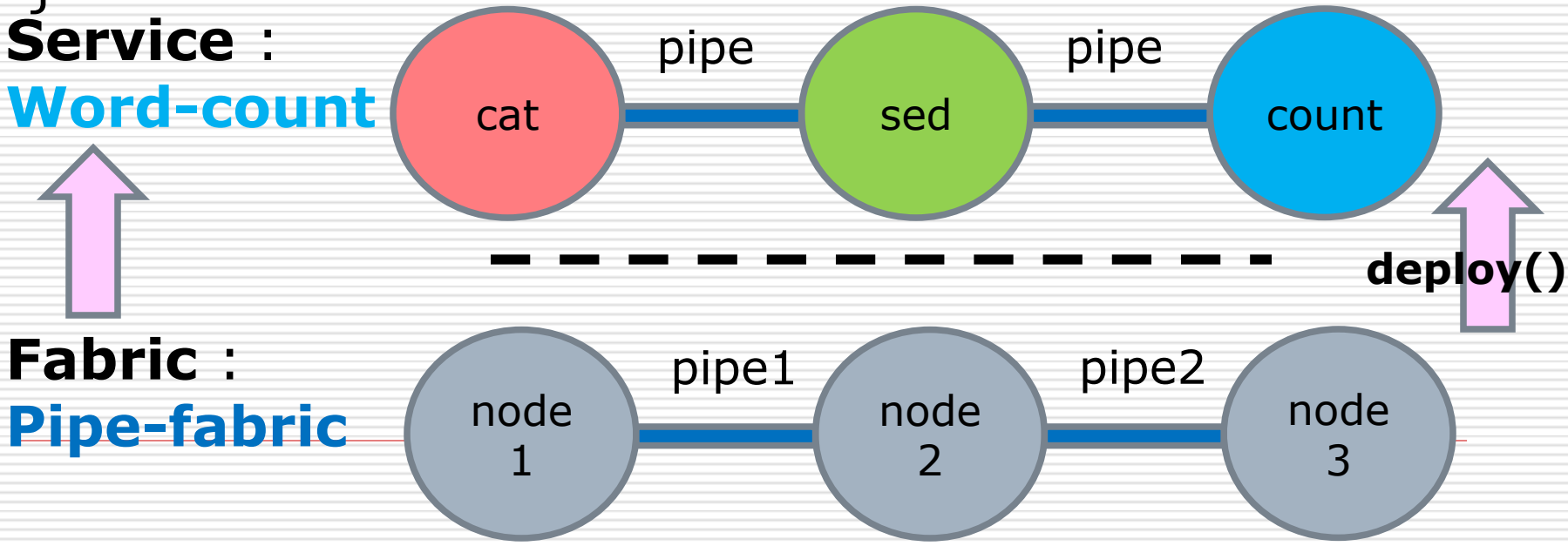



**deploy()**

**Fabric** :
**Web-fabric**

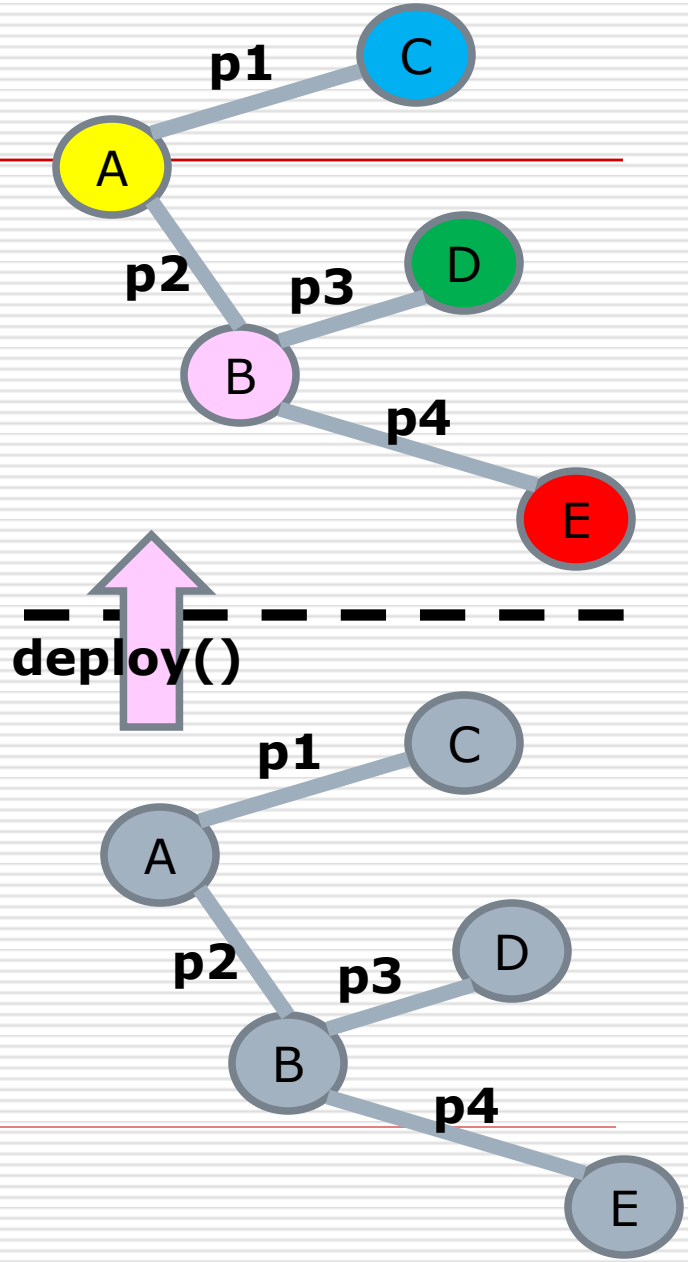
# Example2

```
Service Word-count on Pipe-fabric {
  NodeService cat on node1 {…}
  NodeService sed on node2 {…}
  NodeService count on node3 {…}
}
```



# Example3

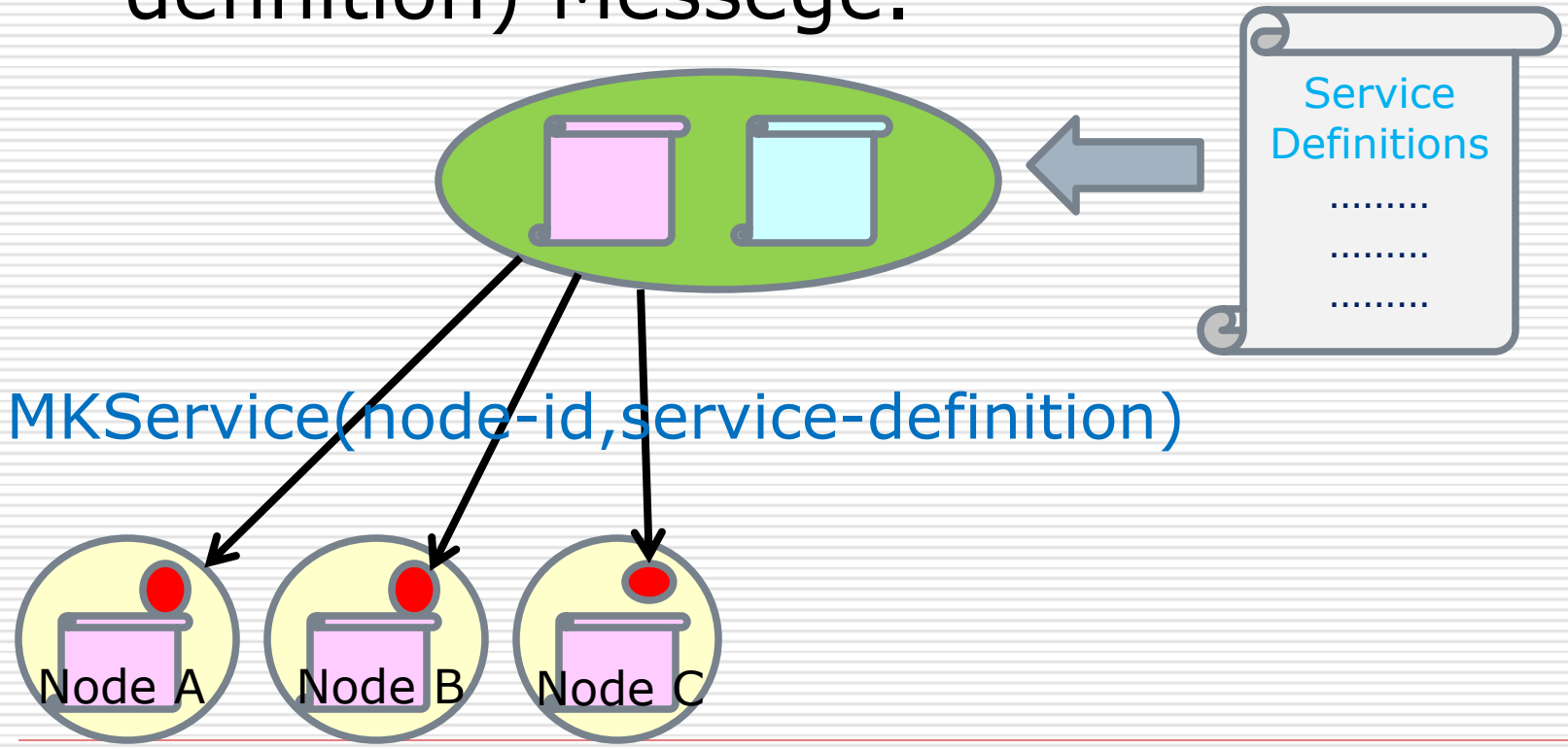
```
Service colours on fabric3 {
  NodeService yellow on A {…}
  NodeService pink on B {…}
  NodeService blue on C {…}
  NodeService green on D {…}
  NodeService red on E {…}
}
```



# Deploy

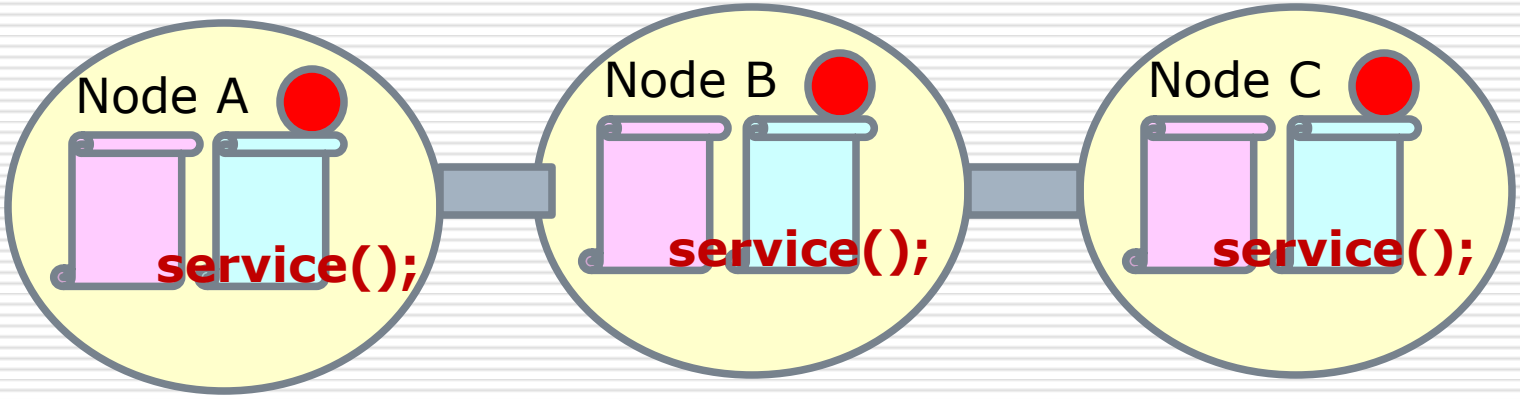
# Deploying (Fabric Controller)

- Send MKService(node-id,service-definition) Messege.



# Deploying (Node Agent)

- Interpreting Service definition, inject service method.



# Extension of definisions

# Array notation

Node node-name[#ofReplica] { …… }

Example

```
Node A[1000]{ A(){…} }
Pipe p[i] { between  node[i] and node[i+1] }
Cluster cluster {
   Node X[3];
   Node Y[100];
   ……
}
```

# Extension of Node definition

**Node** node-name **as Process** | **VM** | **Host**

{

node-name();

}

**Node** node-name[i,j,k] as Process[i]

in VM[j] in Host[k] { …… }

# Extension of Node definition

**Node** node-name **as Process** | **VM** | **Core** | **Host**

```
{
    node-name();
    node-name(node-ID);
    node-name(index);
    ……
}
```

# Extension of Cluster definition

```
Cluster cluster-name {
    Node node-name-lists |
            node-definition-lists ;
    cluster-name();  // Constructor
    get(queue);
    put(queue,data);
    add(node-ID);
    remove(node-ID);
}
```

# Fabric

**Fabric** fabric-name {

**Cluster** cluster-names-list;

**Cluster-definition-lists**;

**Queue** queue-names-list;

**Queue-definition-lists ;**

# Fabric

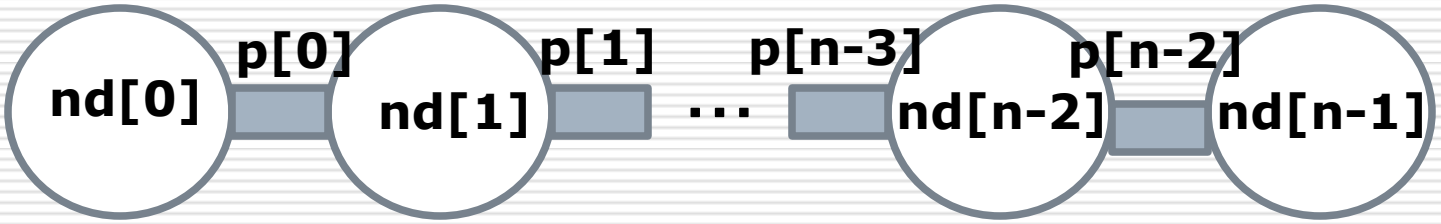
**Node** node-name-list;
**Node-definition-lists**;
**Pipe** pipe-names-list;
**Pipe-definition-lists**;

fabric-name(); // Constructer;
}

# Patterns

Cluster follows a Pattern

# Sequential Pattern



# Sequential Pattern definition

Cluster **Sequential** {

    Node nd[N];

}

Pipe p[0]  { between nd[0] and nd[1] }

Pipe p[1]  { between nd[1] and nd[2] }

……..

Pipe p[N-2] {

  between nd[N-2] and nd[N-1] }

# Generate Definitions Macro

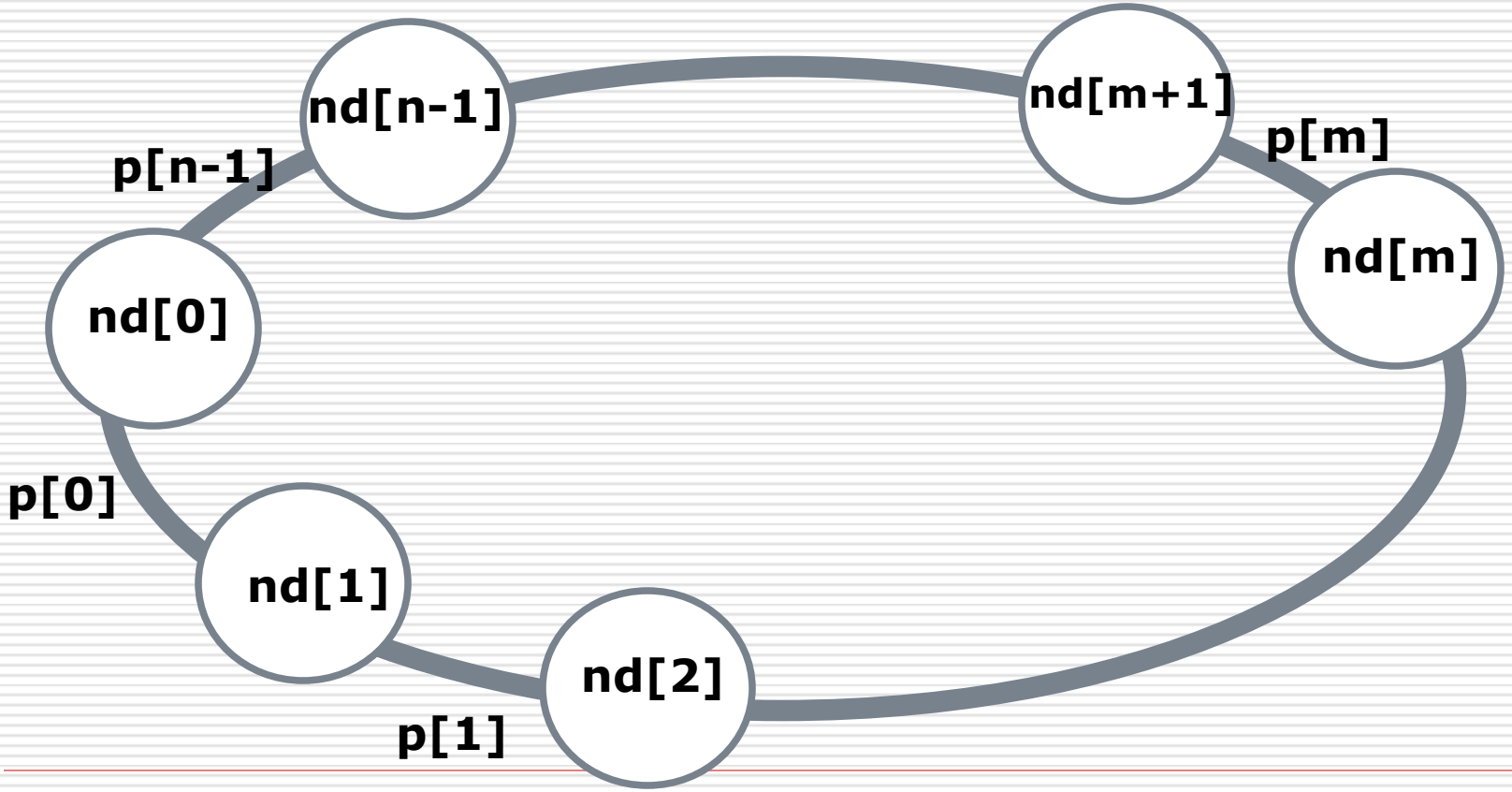
To describe following definitions,

```
Pipe p[0]  { between nd[0] and nd[1] }
Pipe p[1]  { between nd[1] and nd[2] }
  ……..
Pipe p[N-2] { between nd[N-2] and nd[N-1] }
```

We can use **GenerateDefs** Macro.

```
GenerateDefs(i=0;i<N-1;i++) {
   Pipe p[i] { between nd[i] and nd[i+1]; }
}
```

# Circular Pattern

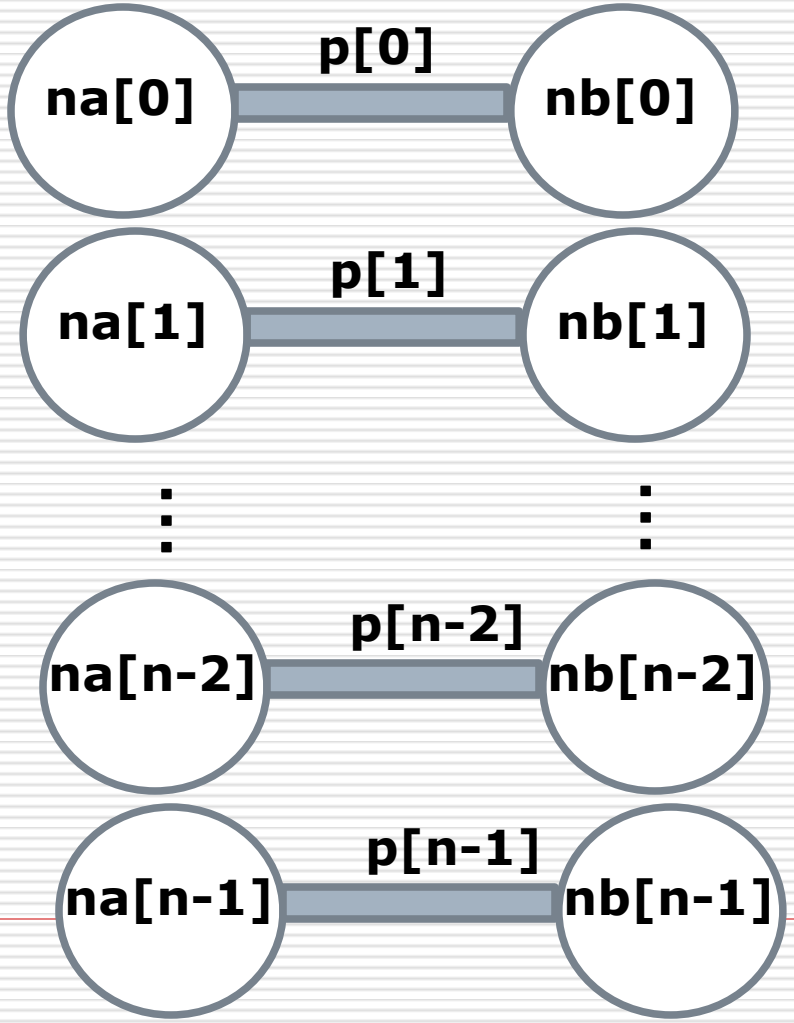


# Circular Pattern definition

```
Cluster Circular {
    Node nd[N];
}

GenerateDefs (i=0;i<N;i++) {
   Pipe p[i]  { between nd[i mod N]
                        and nd[i+1 mod N] }
}
```

# Parallel Pattern

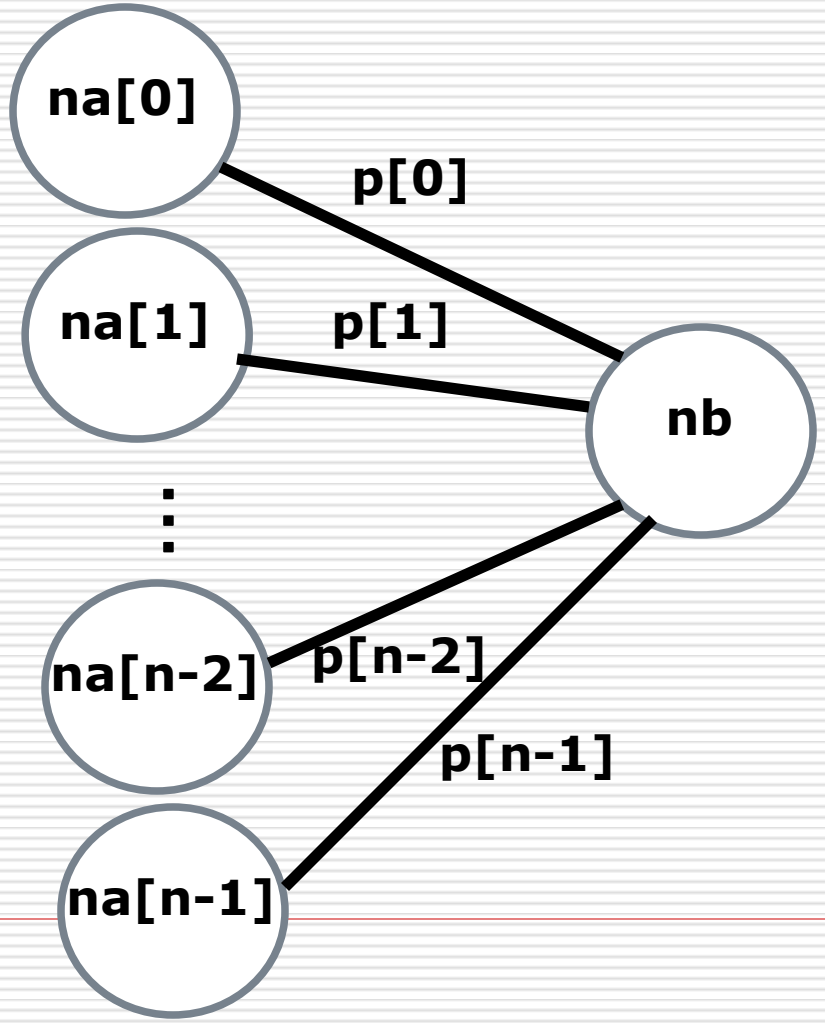


# Parallel Pattern definition

```
Cluster Parallel {
    Node na[N], nb[N];
}

GenerateDefs(i=0;i<N;i++) {
   Pipe p[i]  { between na[i] and nb[i] }
}
```

# FanIn Pattern

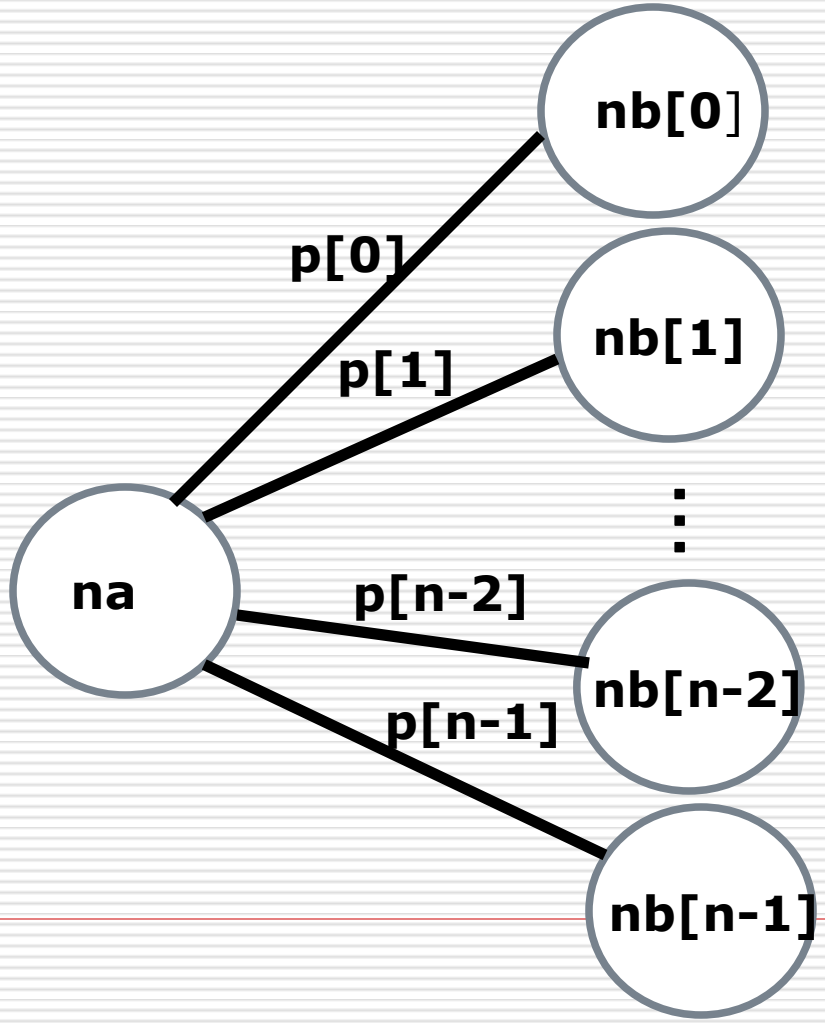


# FanIn Pattern definition

```
Cluster FanIn {
    Node na[N], nb;
}

GenerateDefs(i=0;i<N;i++) {
   Pipe p[i]  { between na[i] and nb }
}
```

# FanOut Pattern

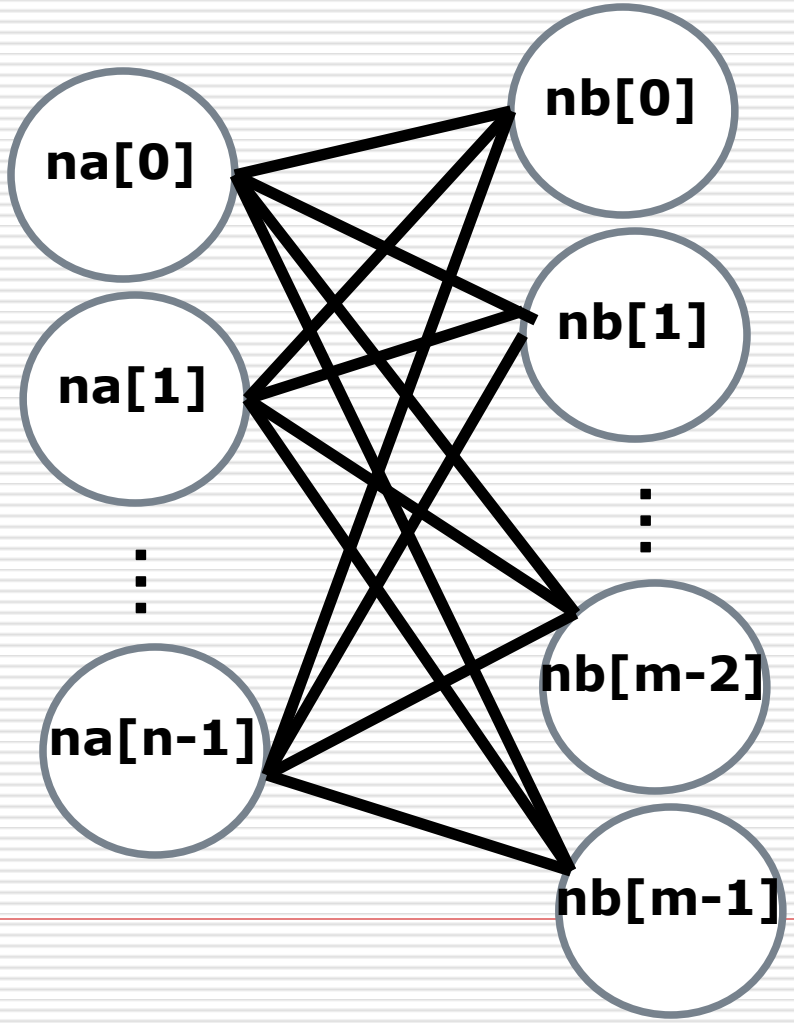


# FanOut Pattern definition

```
Cluster FanOut {
    Node na, nb[N];
}

GenerateDefs(i=0;i<N;i++) {
   Pipe p[i]  { between na and nb[i] }
}
```

# AnyToAny Patern



# AnyToAny Pattern definition

```
Cluster AnyToAny {
    Node na[M], nb[N];}

GenerateDefs(i=0,j=0;i<N,j<N;i++,j++) {
   Lazy Pipe p[i,j]  {
     between na[i] and nb[j] }
}
```

# Clusters follow a Pattern

Cluster cluster-name **follows** aPattern {

Node node-name-list |

node-vector |

node-definition ;

}

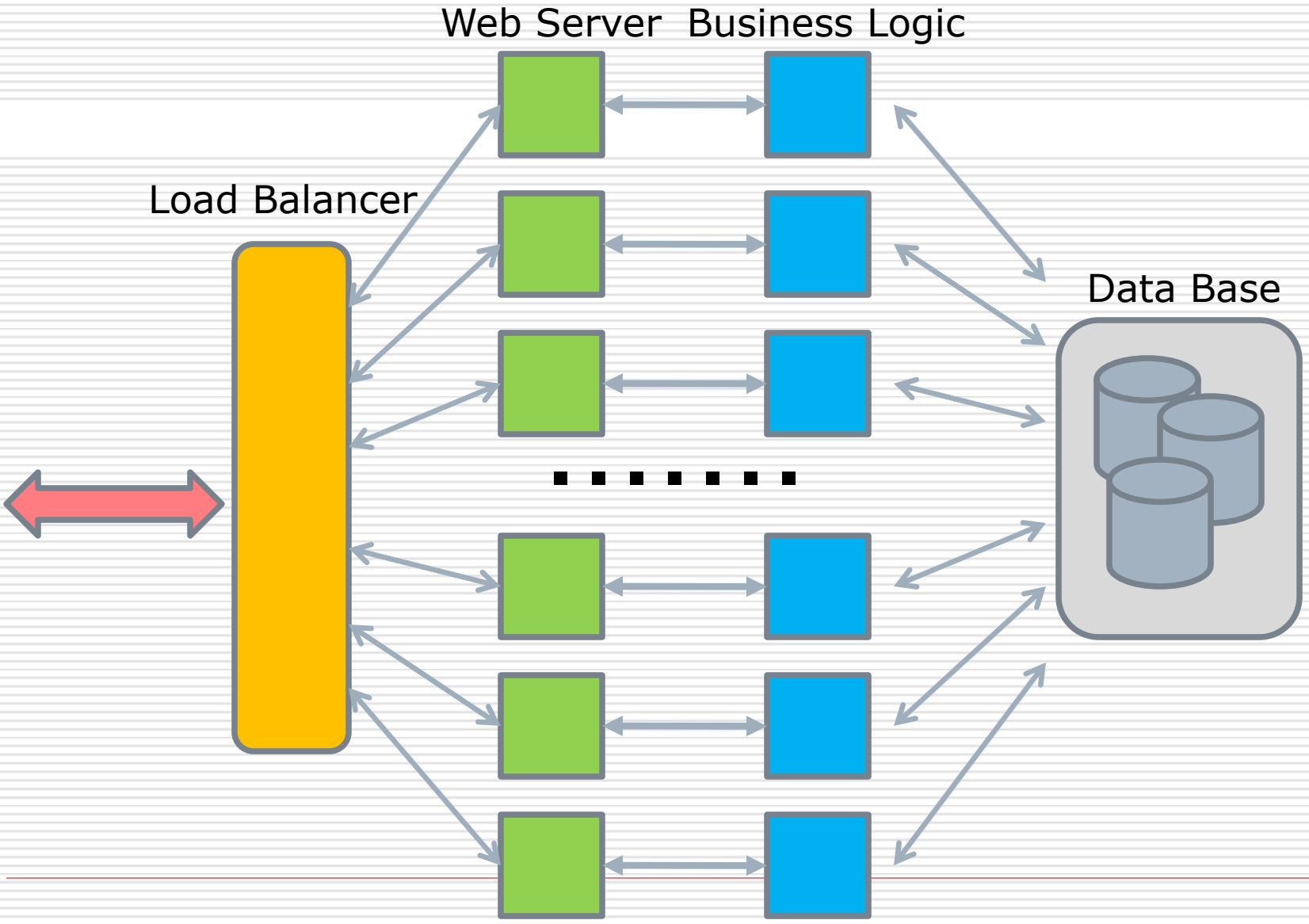
Predefined Pattern = **Sequential** |

**Circular** | **Prallel** | **FanIn** |

**FanOut** | **AnyToAny**

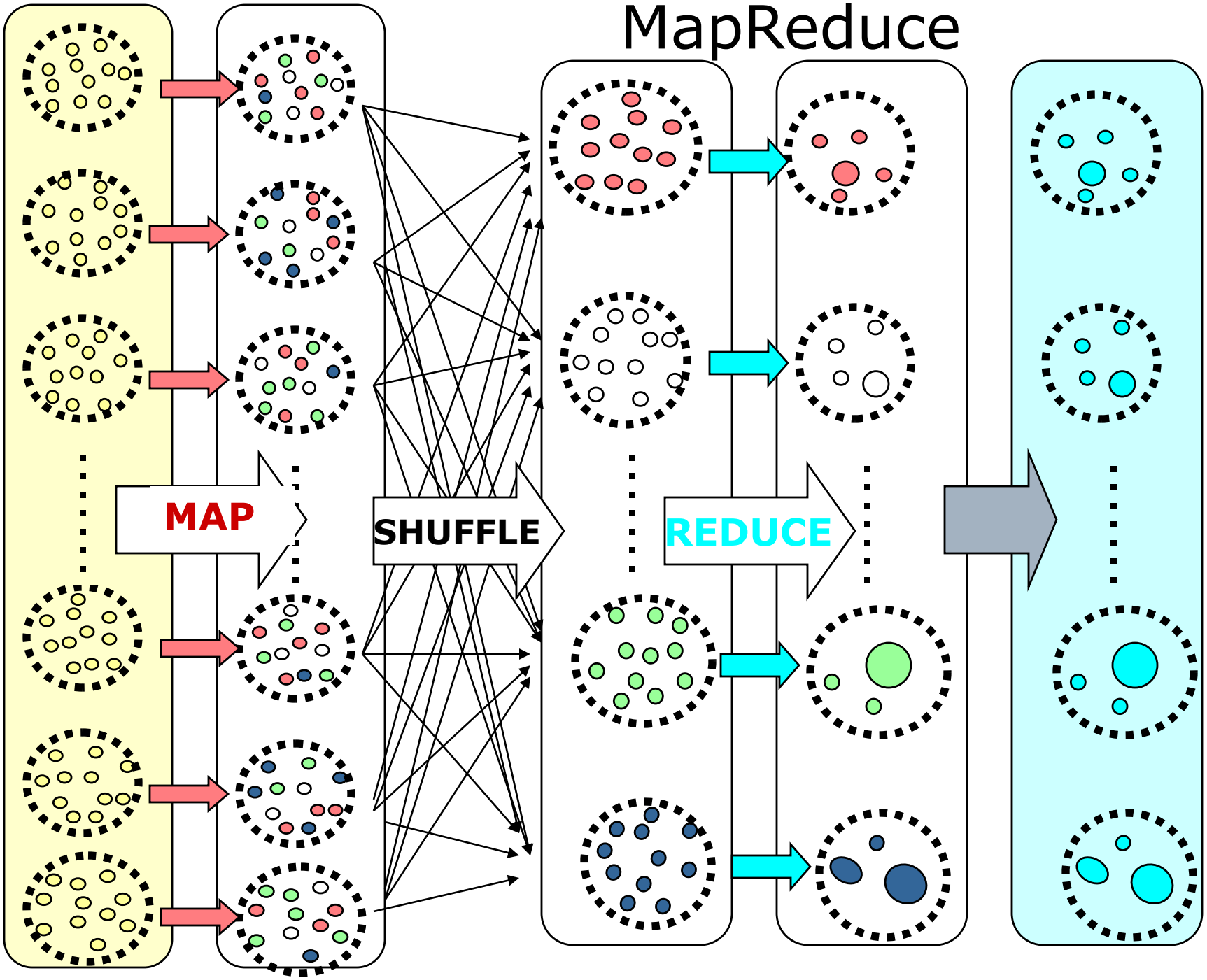
# Describe distributed system in Fabric using Pattern

# 3-tier Web Application



# Example1 3-Tier

```
Fabric 3-Tier-fab {
  Cluster Web-tier follows FanOut {
     Node LB, Web[N];
   }
  Cluster Business-tier follows Parallel {
     Node Web[N], Business[N]
   }
  Cluster Data-tier follows FanIn {
     Node  Business[N], DataBase
  }
  …..
}
```


MapReduce
MAP
SHUFFLE
REDUCE

# Example2 MapReduce

```
Fabric  MapReduce-Fab {
  Cluster Mappers follows Parallel {
     Node Map[M],Mapped[M] ;
  }
  Cluster Shuffling follows AnyToAny {
     Node Mapped[M],Reduce[R];
  }
  Cluster Reducer follows Parallel {
     Node Reduce[R], Reduced[R];
  }
```

# Example2 MapReduce

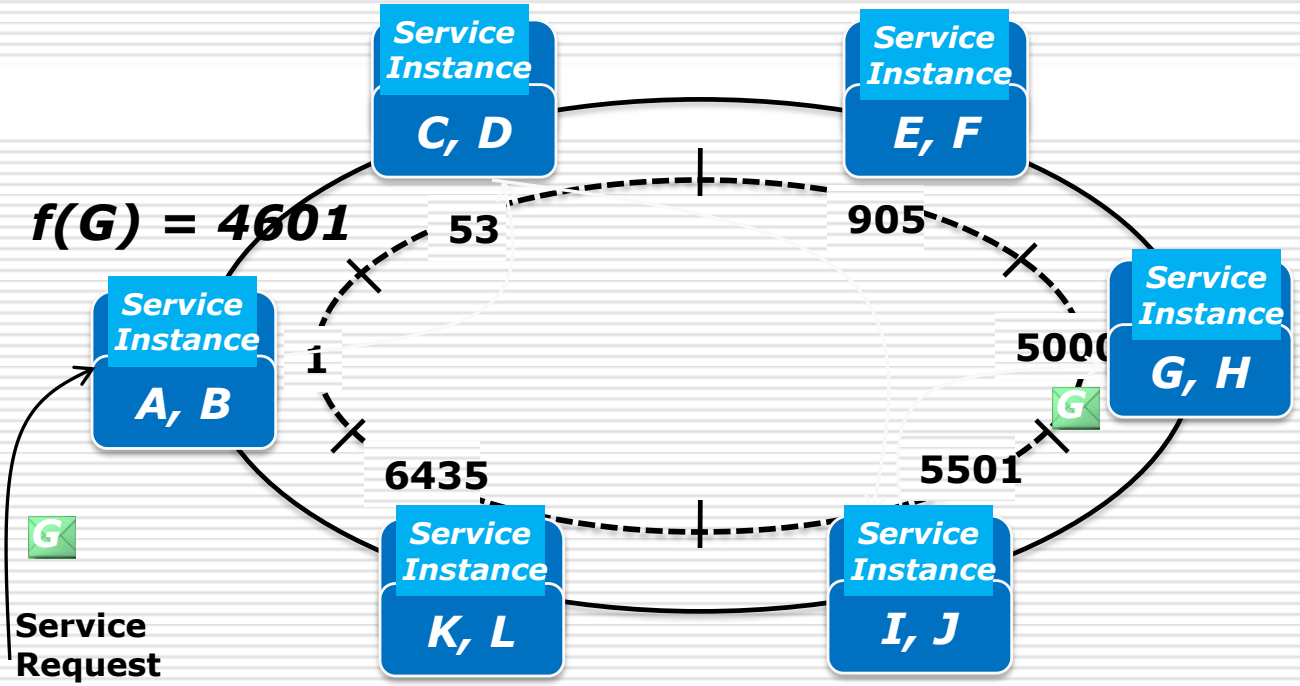
```
Service map on Node Map[x] {
    service(){ put(Map[x].Pipe[x], map(get())) }
}
Service groupByKey on Node Mapped[x] {
    service(){
       <key,value>=get();
       put( Mapped[x].Pipe[x , key], <key,value>);
    }
}
Service reduce on Node Reduce[x] {
    service(){
       put(Reduce[x].Pipe[x],
       reduce(get())) }
}
```

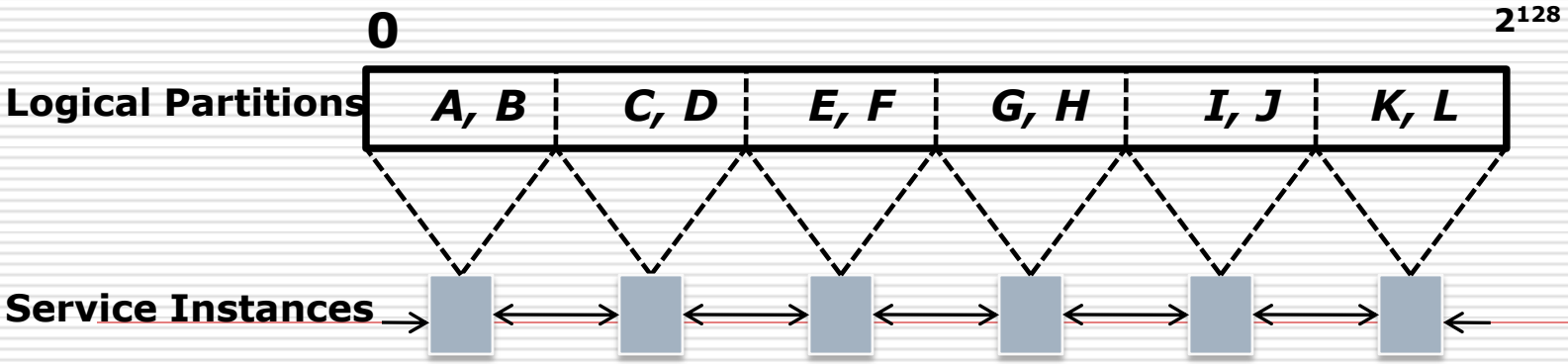
# Example2 MapReduce

```
Service MapReduce
   on Fabric MapReduce-fab {

    main(){
       weave(); // MapReduce Fabric
       deploy(); // Map, Shuffle and Reduce
       deploy-data(); // ?
       start();
       end();
       undeploy();
       unweave();
}
```

# DHT


Service Instance
C, D
Service Instance
E, F
f(G) = 4601
53
905
Service Instance
A, B
1
5000
Service Instance
G, H
6435
5501
Service Instance
K, L
Service Instance
I, J
Service Request


0
2128
Logical Partitions
A, B
C, D
E, F
G, H
I, J
K, L
Service Instances

# Example3 DHT

```
Fabric DHT-fab {
  Cluster DHT-srv follows Lazy FanOut {
      Node server , node[1000];
  }
  Cluster DHT follows Circular {
      Node node[1000];
  }
  Cluster DHT2 follows AnyToAny {
      Node node[1000];
  }
```

# Controll Pattern

# Controll Pattern

- For each occurrence of '**as** controll-pattern' in cluster definition, we add one Controll node to the cluster.

  **Cluster** cluster-name
  **as** controll-pattern

- Node n,…,m

  Node n,…,m,**controll-node**

# Available Pattern

Cluster c { Node n[N];…} **as Available**

```
Cluster c follow FanOut{
    Node cn, n[N]; // cn : controll-node
    service() {       // cluster service work on cn
        for(i=0;i<N;i++){
            sleep(sometime);
            status = get(pipe[i]);
            if (status==EveryThingOK) continue;
            if (status ==NoReply) waitForAWhile();
            if (status==SomethinMustBeWrong){
                remove(n[i]);
                add(new n[i]);
             }
```

# Scalable Pattern

Cluster c { Node n[N];…} **as Scalable**

```
Cluster c follow FanOut{
    Node cn, n[N]; // cn : controll-node
    service() {       // cluster service work on cn
        for(i=0;i<N;i++){
            sleep(sometime);
            status = get(pipe[i]);
            if (status==EveryThingOK) continue;
            if (status ==NoReply) waitForAWhile();
            if (status==TooHeavy){add(new n[++i]); reweave();}
            if (status==Toolight){remove(n[i]); reweave();}
```

# Stateful Pattern (node-controller)

Cluster c { Node n[N];…} **as Stateful**

```
Cluster c follow FanOut{
    Node cn, n[N]; // cn : controll-node
    service() {       // cluster service work on cn
        for(i=0;i<N;i++){
            state = get(pipe[i]);
            store(i,state);
        }
    }
```

# Stateful Pattern (for each node)

```
Node n {
    n(){…}
    get(){…}
    put(){…}
    putSTATE(state){…}
    getSTATE(){…}
    rollback(){…}
}
```

# Replica Data Pattern

# Node node-name with replica[N]

Node node-name **with replica[N]**

```
Cluster node-name as available {
    Cluster n {Node node-name;}
    Cluster replica {Node replica[N];}
    Queue q { From n to replica }
    get(){
       data=Node.node-name.get();
       put(q,data);
    }
    put(){Node.node-name.put();}
}
```