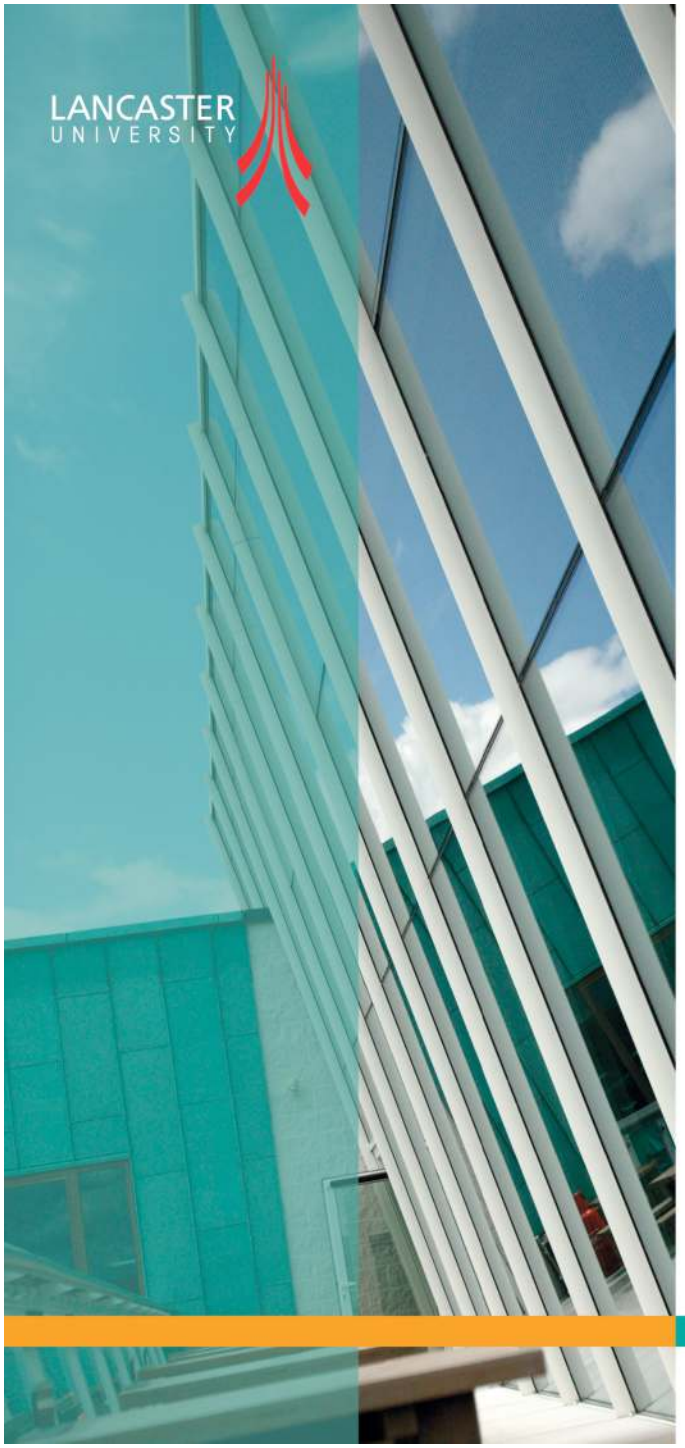


Transparent Componentisation

Multi-Paradigm Programming for Evolving Distributed Systems

Shen Lin[1], François Taïani[1,2], Gordon Blair[1], Marin Bertier[2], Anne-Marie Kermarrec[2]

(2) INRIA RENNES

(1) School of Computing and Communications

Motivation

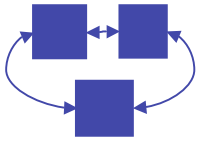
Today's Distributed Systems

- **Heterogeneous**
  - ➔ resources, mobility, context
- **Dynamic**
  - ➔ churns, failures
- **Large and Complex**
  - ➔ SoS, P2P, cloud

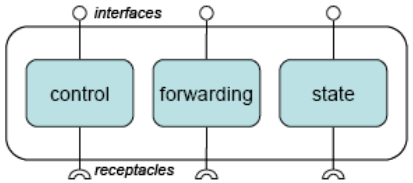
Need for solutions that are **flexible**, **evolvable**, **expressive**

The rise of components

- Component successfully applied to distributed systems
  - ➔ industry: EJB, CCM, OSGi
  - ➔ research: Fractal, OpenCOM
  - ➔ middleware Frameworks: GridKit, Rapidware, Ensemble, Cactus, Open Overlays

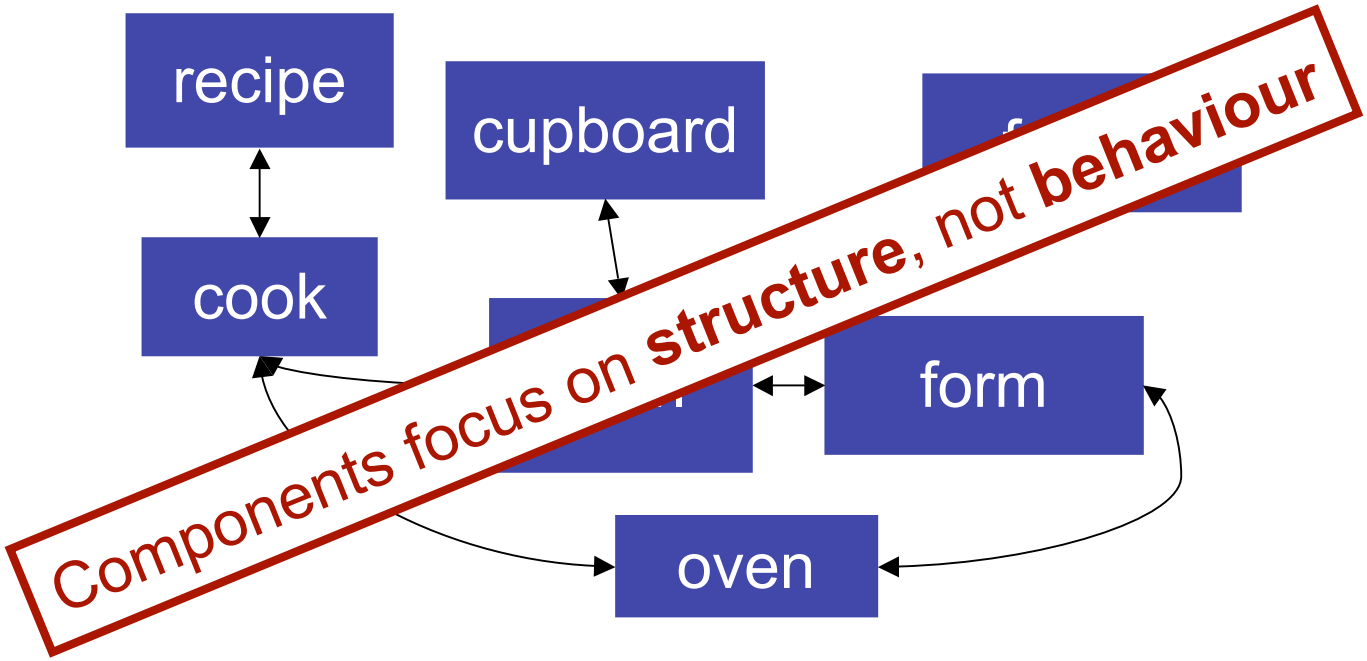


- Clear **structure**, explicit **dependencies**



- Benefits
  - ☺ promote **reuse**
  - ☺ easily **composable** and **configurable** (SPL..)
  - ☺ lend themselves to **runtime reconfiguration**

The problem with components



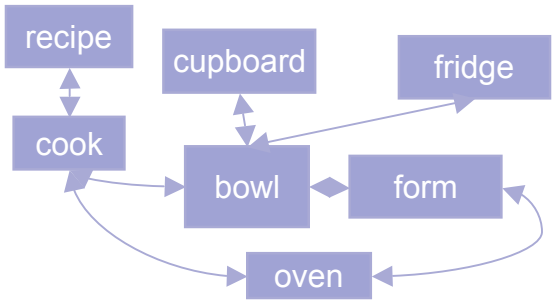
- Drawbacks
  - ☹ low **intelligibility** (where is the intent?)
  - ☹ conceptual **mismatch** for developers focusing on behaviour
  - ☹ high **learning** curve for unfamiliar frameworks

High-level distributed languages

- **Spec. lang. and DSL:** High-level per node description
  - ➔ Lotos, Estelle, PLAN-P, Mace …
- **Marco-programming**: system as one entity
  - ➔ E.g. Kairos, Regiment, TinyDB, MIT-Proto
  - ➔ centralised shared-memory parallel abstraction
  - ➔ main program compiled into code for each node
- Benefits
  - ☺ high level of **abstraction** (in particular for macro-prog)
  - ☺ **intelligible**
  - ☺ good conceptual **match** for developers looking at behaviour

bake

Behaviour rather than structure




```
add(yohourt,1)
add(milk,2)
add(flour,3)
add(butter,1)
bowl.pour(form)
form.putIn(oven)
bake()
```

Can we build a hybrid approach that combines the strengths of both?

- Drawbacks
  - ☹ we loose the benefits of components (reuse, …)

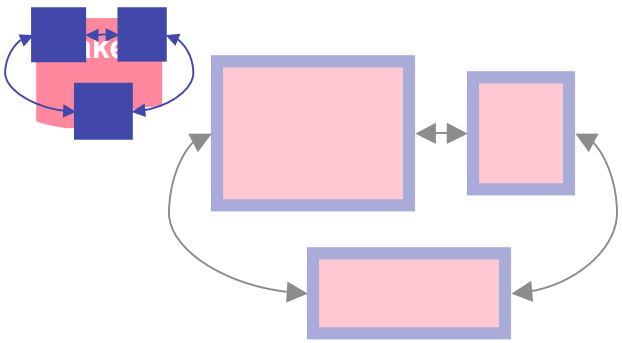
Outline

- **concept**: transparent componentisation
- application to gossip: **WhispersKit**
- **evaluation**: simplicity and performance

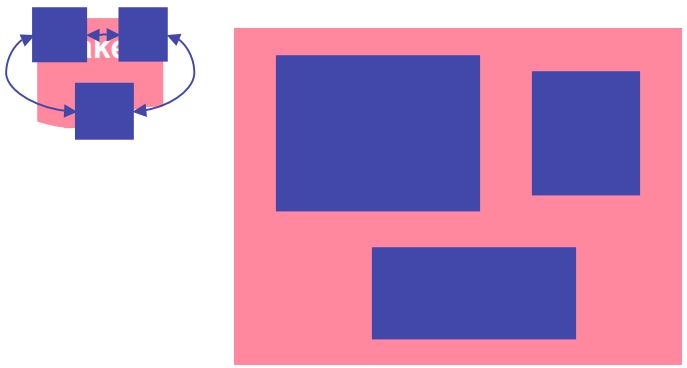
Outline

- **concept**: transparent componentisation
- application to gossip: **WhispersKit**
- **evaluation**: simplicity and performance

structure + behaviour = ?



encapsulation



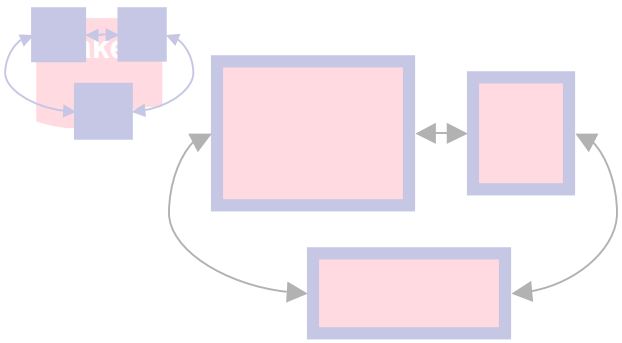
orchestration

- 😐 **tangling** behaviour & structure
- 😐 'breaks' **encapsulation**
- 😐 tension **granularity** vs. **scattering**

- 😐 **complex** composition
- 😐 tension **structural** needs vs. **programmatic** ones

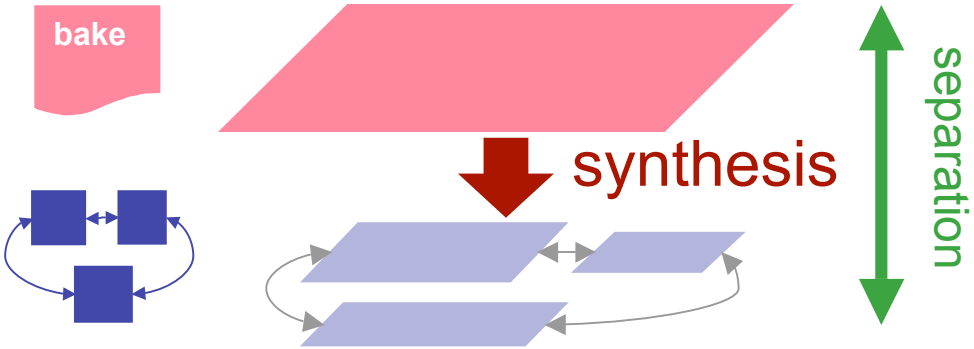
LANCASTER UNIVERSITY
School of Computing and Communications

structure + behaviour = ?



encapsulation

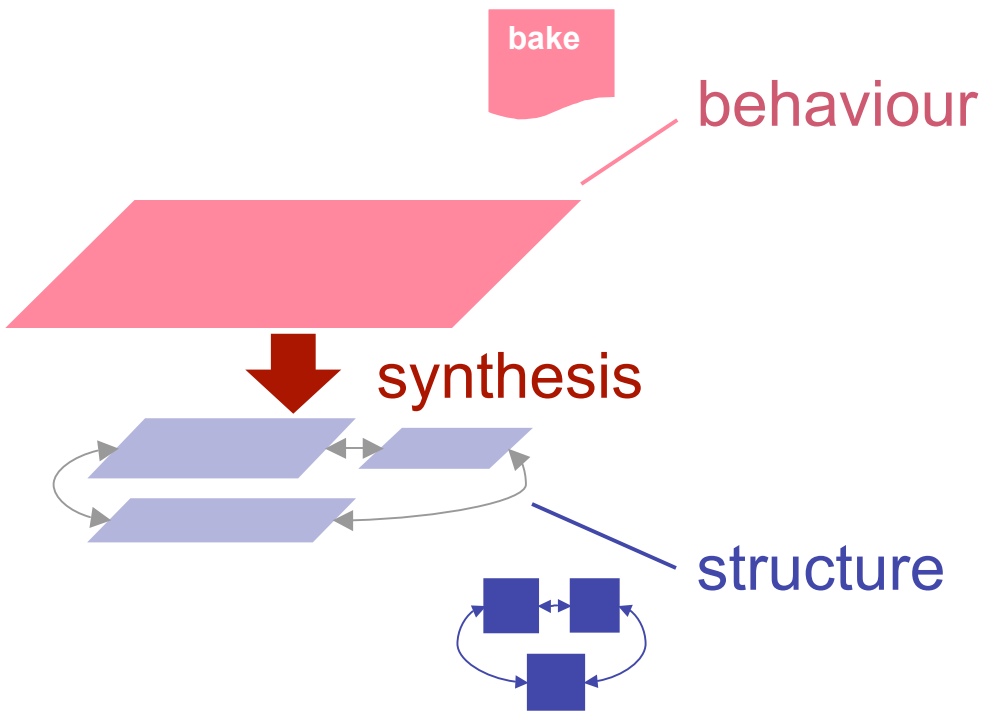
orchestration



transparent componentisation

LANCASTER UNIVERSITY
School of Computing and Communications

Transparent Componentisation



☺ simple
☺ concise
☺ high-level

☺ modular
☺ reusable
☺ (re)configurable

- **Separation of concern** between behaviour / structure
- **Developers** can focus on **high level logic**
- **Systems** takes care of **modularity**, reuse, and evolution

LANCASTER UNIVERSITY
School of Computing and Communications

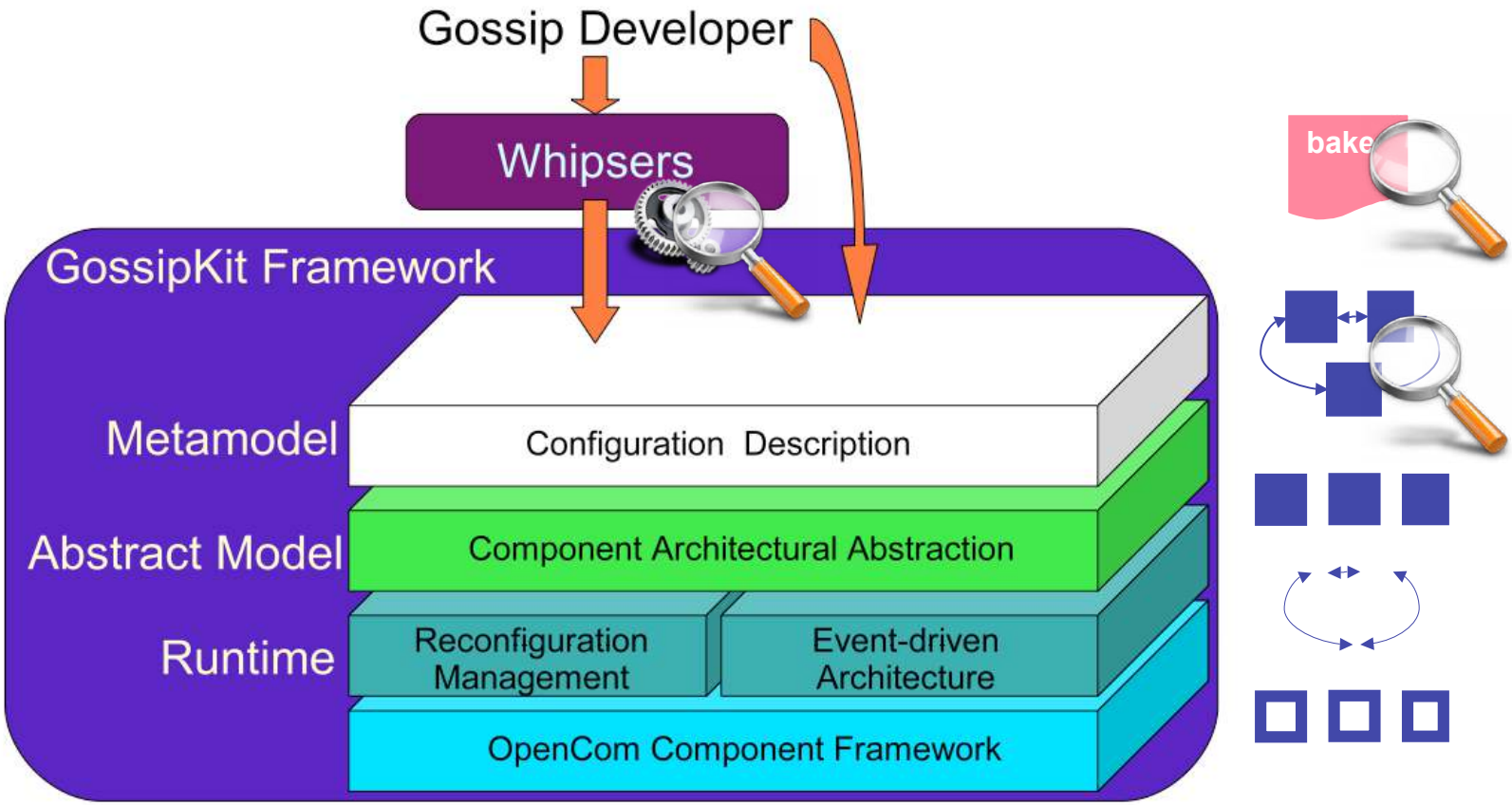
Outline

- **concept**: transparent componentisation
- application to gossip: **WhispersKit**
- **evaluation**: simplicity and performance

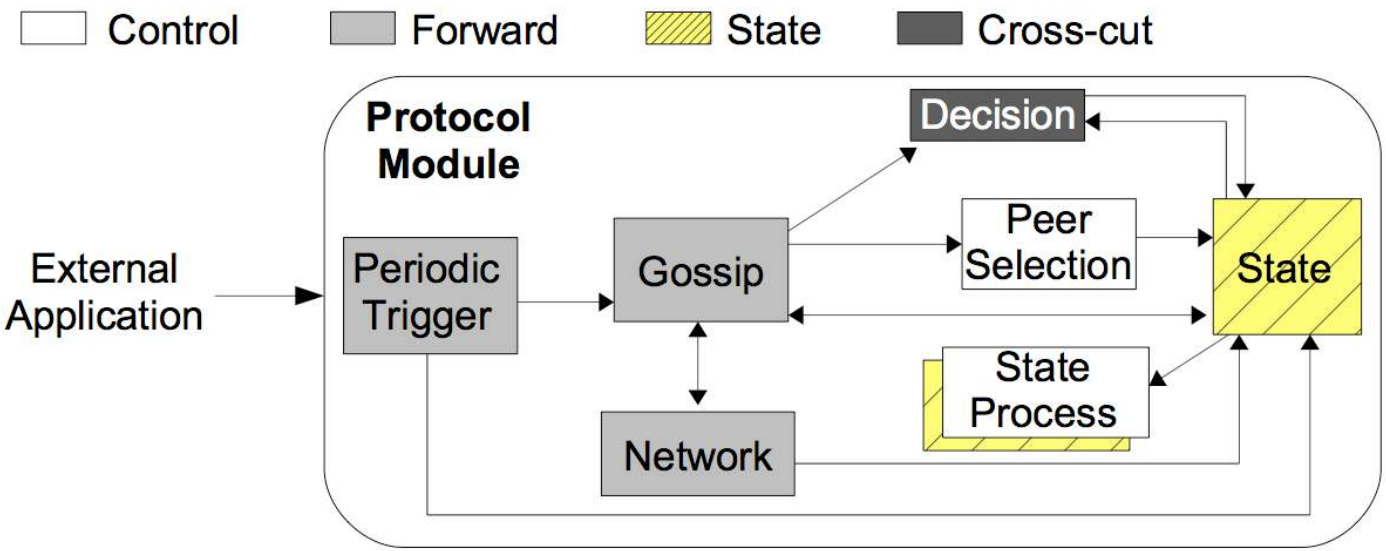
Application to Gossip Protocols

- Gossip (aka epidemic) protocols
  - ➔ leverage **rumour-like** propagation of information
  - ➔ large applicability: aggregation, broadcast, clustering
  - ➔ often **composed** to realised higher-level services
- Conceptually simple
  - ➔ typically symmetric behaviour
  - ➔ key notions of **state**, information **flows**, and **decisions**
- But implementation can be time consuming
  - ➔ multithreading, distributed coordination, network intricacies, co-existence

The WhispersKit Architecture



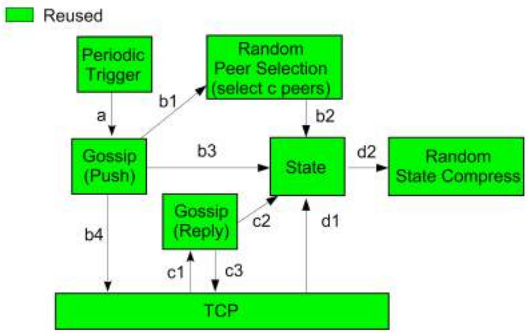
GossipKit



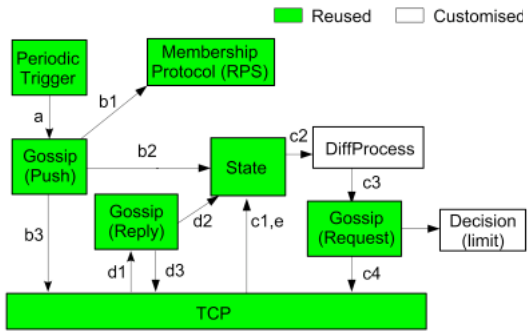
- A component framework for epidemic protocols
  - based on analysis of 30 gossip protocols
  - event-based
  - XML-based configuration for component composition
  - targets abstraction, modularity, reuse, evolvability

LANCASTER UNIVERSITY
School of Computing and Communications

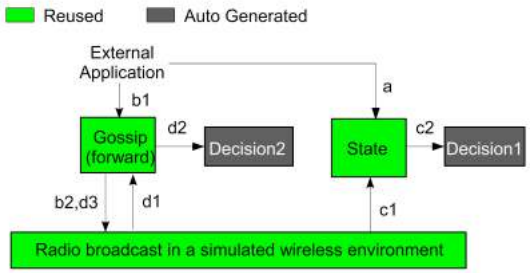
GossipKit Examples



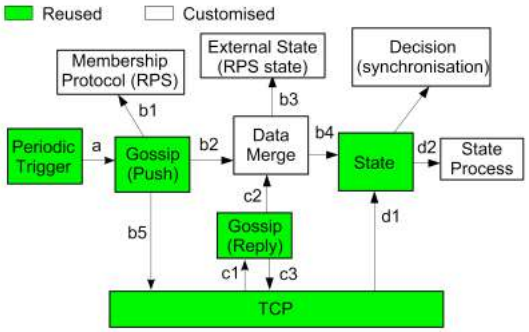
RPS



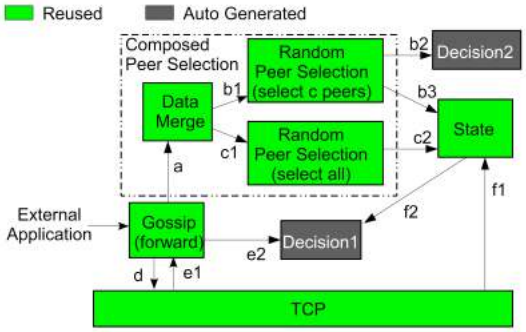
Anti-Entropy



Wireless broadcast



T-Man



SCAMP

Whispers

- macro-programming language for gossip protocols
  - ➔ system as one entity
- primitives

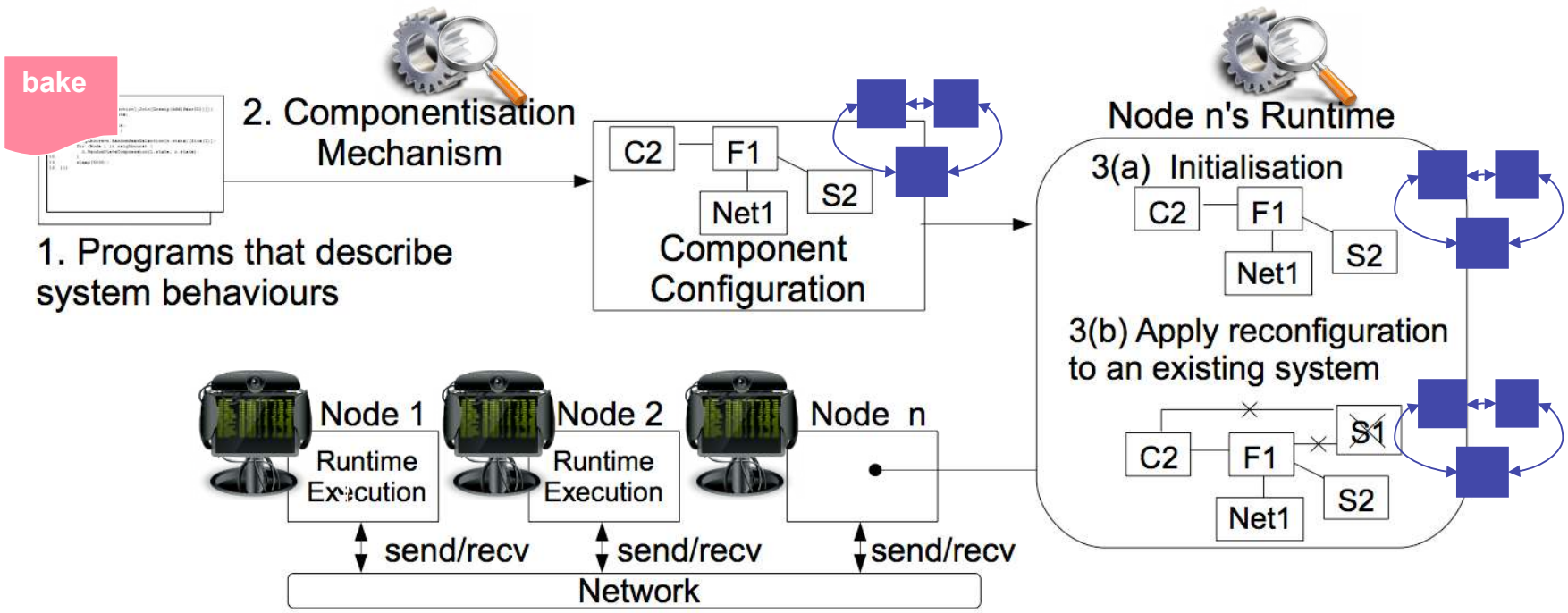
```
protocol {..}                    // protocol block
every (time) {..}                // periodic behaviours
wait (Event e type T) {..}       // reactive behaviours
foreach(n in nodeSet)            // distribution
synchronised {..}                // pairwise data exchange
State state = new State[fields][size] ; // state decl.
state.field ;                    // get a column of data
state.add([fields])              // add
state.remove(row_ID)             // remove
i.RandomStateCompress(...)       // library call
```


LANCASTER UNIVERSITY
School of Computing and Communications

Whispers Example: RPS

```
RPS {
  State sample = new State[Node:PeerID][Size=5];
  Node n, i;
  every (5000) { // do the following every 5000 ms
    foreach (n in AllNodes) { // for each node n
      i=n.RandomPeerSelection(n.sample)[Size=1];
      n.sample.add([n]);
      i.RandomStateCompress(i.sample,n.sample)[Size=5];
      n.RandomStateCompress(i.sample,n.sample)[Size=5];
    } // end of foreach
  } // end of every
} // end of RPS protocol block
```

Deployment Process



Compilation

```
RPS {
  State sample = new State[Node:PeerID][Size=5];
  Node n, i;
  every (5000) { // do the following every 5000 ms
    foreach (n in AllNodes) { // for each node n
      i=n.RandomPeerSelection(n.sample)[Size=1];
      n.sample.add([n]);
      i.RandomStateCompress(i.sa
      n.RandomStateCompress(i.sa
    } // end of foreach
  } // end of every
} // end of RPS protocol block
```

Periodic thread

1 **every** 5 seconds

2 Node neighbour = RandomPeerSelection

3 **retrieve** local_sample

4 **push** local_sample to neighbour

Reactive thread on receipt of a message

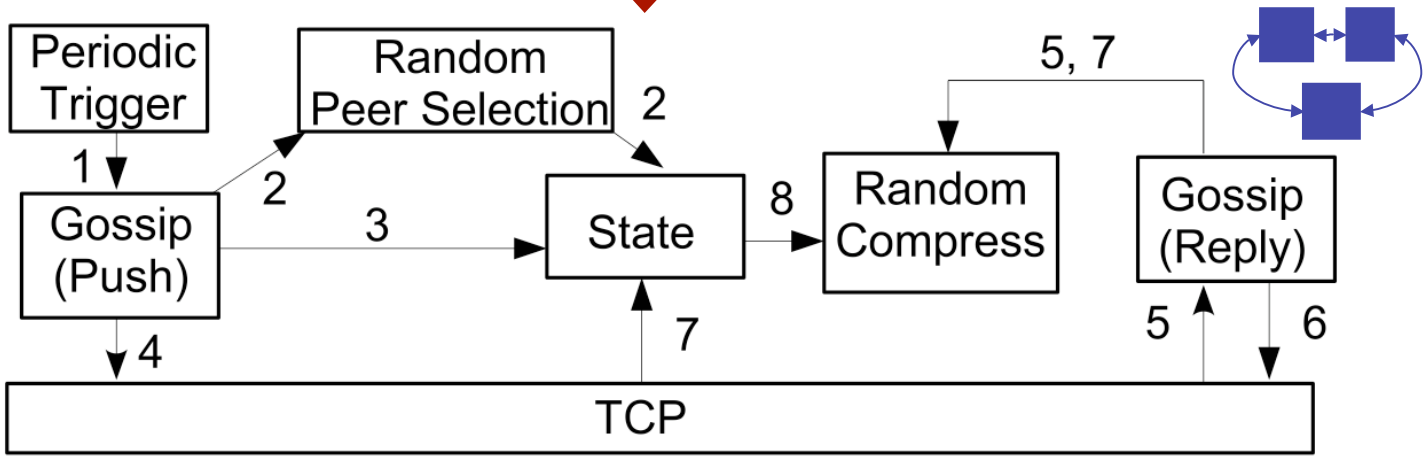
message **push** | message **reply**

5 **retrieve** local_sample

6 **reply** local_sample

7 **extract** remote_sample from message

8 RandomCompress (local_sample, remote_sample)



Distributed Reconfiguration

- Developers describes new behaviour in Whispers
- Platform uses component representation
  - ➔ to compute minimal set of change
  - ➔ to propagate and enact reconfiguration

System Behaviour A

bake

```
1.  RPS (GetPeers[PeerSelection],Join[Gossip[Add(PeerID)]]{
2.    State[PeerID][5] state;
3.    Node n;
4.    List<Node> neighbours;
5.    for(n in ALL_NODES) {
6.      for [;;] {
7.        neighbours=n.RandomPeerSelection(n.state)[Size(1)];
8.        for (Node i in neighbours) {
9.          n.RandomStateCompression(i.state, n.state);
10.       }
11.       sleep(5000);
12. }}}
```

Transparent reconfiguration

System Behaviour B

cook

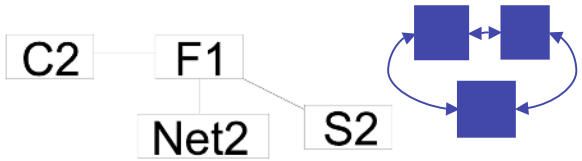
```
1.  RPS (GetPeers[PeerSelection],Join[Gossip[Add(PeerID)]]{
2.    State[PeerID][5] state;
3.    Node n;
4.    List<Node> neighbours;
5.    for(n in ALL_NODES) {
6.      for [;;] {
7.        neighbours=n.RandomPeerSelection(n.state)[Size(1)];
8.        for (Node i in neighbours) {
9.          n.RandomStateCompression(i.state, n.state);
10.       }
11.       sleep(5000);
12. }}}
```

Component mapping

Component Configuration A

Unbind C1 and S1
Unload S1
Replace C1 by C2
Replace Net1 by Net2

Component mapping



Component Configuration B

Distributed Reconfiguration

- Example: RPS → T-Man(Ring) → T-Man(Grid)

coarse grained *fine grained*

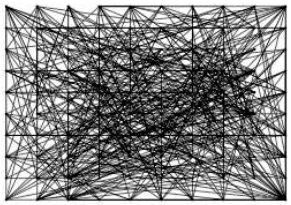


Figure 5.6: Initial random graph maintained by RPS

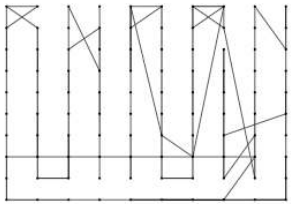


Figure 5.7: 5th rounds since 1st reconfiguration

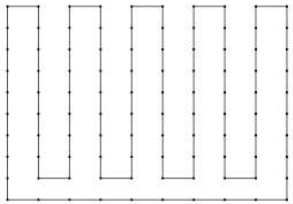


Figure 5.8: Ring constructed at the 11th round

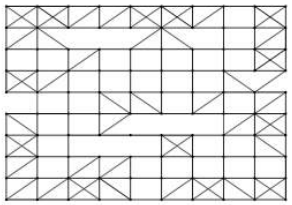


Figure 5.9: Topology at the 20th round

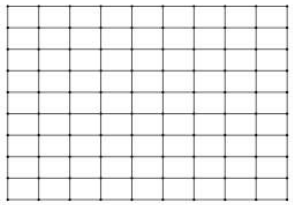


Figure 5.10: Grid constructed at the 23rd round

Outline

- **concept**: transparent componentisation
- application to gossip: **WhispersKit**
- **evaluation**: simplicity and performance

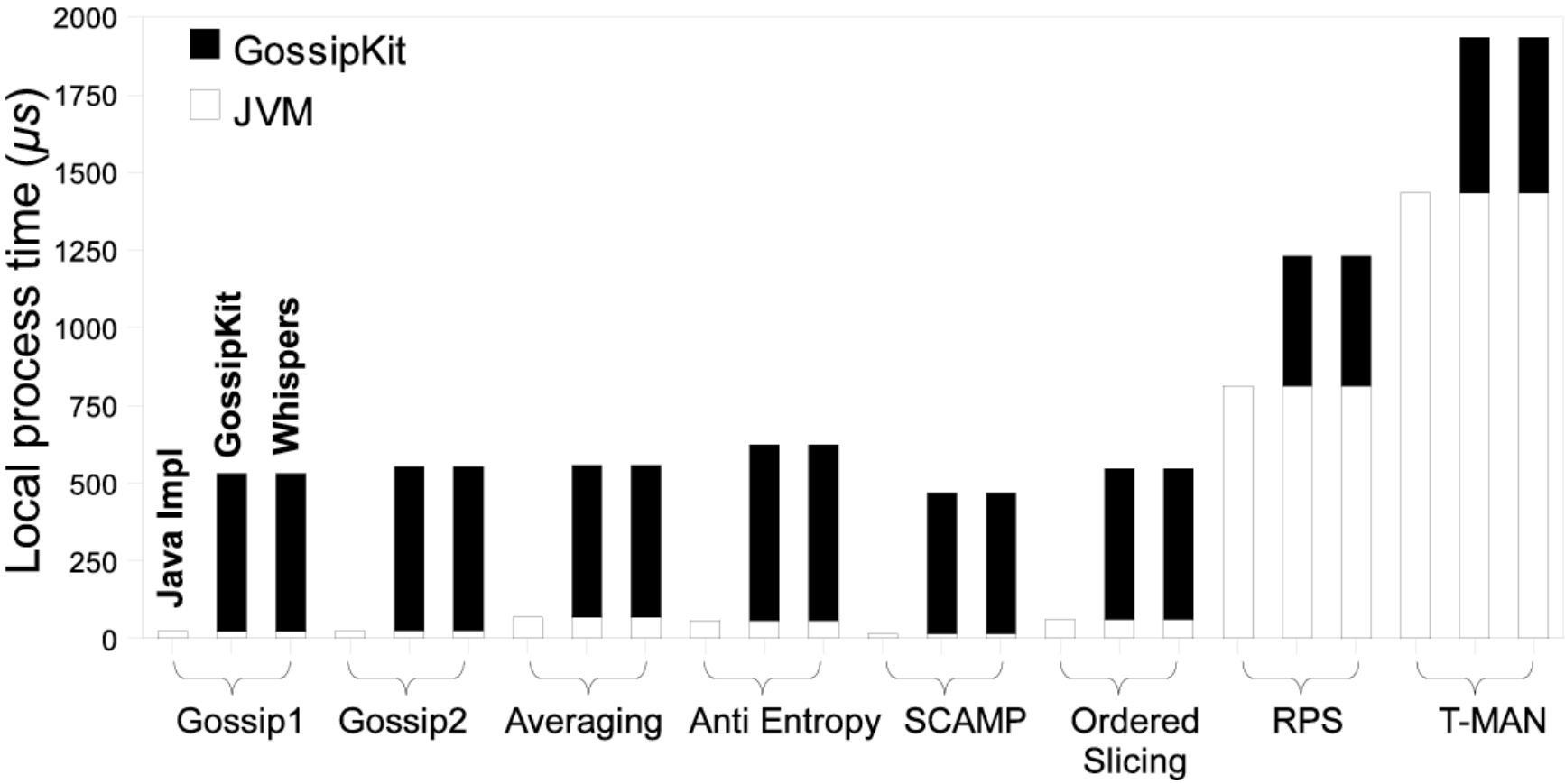
Evaluation: Simplicity (1)

Protocol	WHISPERS	Java	GOSSIPKIT XML Configuration
`Gossip1`	14	277	39
`Gossip2`	14	279	39
Anti Entropy	16	544	100
Averaging	14	466	85
Ordered Slicing	14	471	85
RPS	12	439	81
SCAMP	19	463	88
T-Man	20	491	93
Average	15.4	424	76.3

Evaluation: Simplicity (2)

Protocol	WHISPERS	Java	GOSSIPKIT configuration		
	Cyclomatic	Comp.	Component	Parameter	Connection
Gossip1	2	11	5	6	7
Gossip2	2	11	5	6	7
Anti Entropy	3	10	9	15	13
Averaging	3	6	8	12	11
Ordered Slicing	3	11	8	12	11
RPS	2	12	7	15	10
SCAMP	3	20	8	10	12
T-Man	3	11	8	15	12
Average	2.6	11.5	7.3	11.4	10.4

Performance Overhead



Conclusion

- **Transparent componentisation**
  - ➔ separates the behavioural concerns from structural ones
  - ➔ developers can entirely focus on their code's logic
  - ➔ while harnessing the benefits of a modular approach
- **Whispers/GossipKit**
  - ➔ combines component framework and high-level language
  - ➔ Simple and high-level expressions
  - ➔ Macro-level description
  - ➔ Automatic mapping from programs onto components
- **Available on line:** http://ftaiani.ouvaton.org/GossipKit/

Future Work

- **Extend** Whispers/GossipKit beyond Gossip
- **Exploit** Transparent Componentisation for
  - ➔ reasoning about interferences and synergies
  - ➔ self-adaptation
- **Link** to Model-driven Engineering
- More generally **explore** link between structure / behaviour / and intelligibility in dist. systems

Thank you

Questions?

Composite Example: Averaging

```
Averaging {
  RPS rps = new RPS();
  State state = new State[float:Temp][Size=1];
  Node n, i;
  every (5000) {
    foreach (n in AllNodes) {
      i = n.rps.RandomPeerSelection(n.rps.sample)[Size=1];
      synchronised {
        i.Average(i.state.Temp, n.state.Temp)[];
        n.Average(i.state.Temp, n.state.Temp)[];
      } // end of synchronised
    } // end of every
  } // end of foreach
} // end of Averaging protocol block
```