Journées GDR GPL

Engineering Scale: Software and Distribution for Tomorrow's World

François Taïani

UMR IRISA
UNIVERSITÉ DE RENNES 1
Inria INVENTEURS DU MONDE NUMÉRIQUE
esir ECOLE SUPERIEURE D'INGENIEURS DE RENNES

Ideal software artefact

- ➔ structured, predictable, open, evolvable

A Distributed System Today …

Standards

JSON
JavaScript Object Notation

External developers

External services

facebook

twitter

bit.ly

foursquare™

Geosocial app, est. 2009

45M Users

Middleware

Today's distributed systems

- ➔ sprawling, chaotic, complex, unmanageable?

Outline

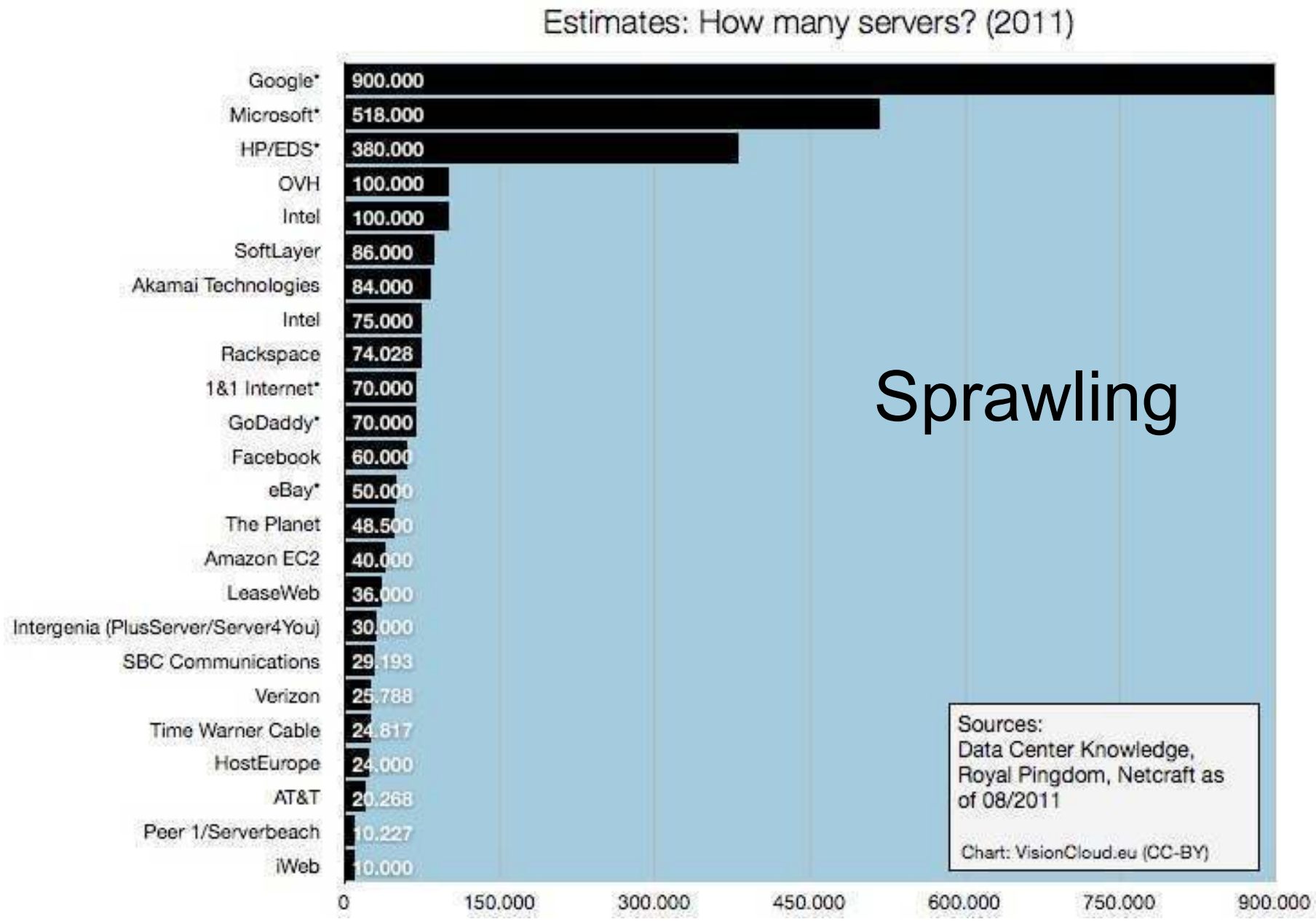
- A call to arms: engineering large scale
- Examples of ways forward

Outline

- A call to arms: engineering large scale
- Examples of ways forward

Today's distributed systems

- ➔ sprawling, chaotic, complex, unmanageable?



Source: "How Many Servers Worldwide?", Mirko Lorenz, http://www.visioncloud.eu/content.php?s=191,324

http://www.google.com/about/datacenters/gallery/#/

Complex

Portability

Interoperability

Transparency

...

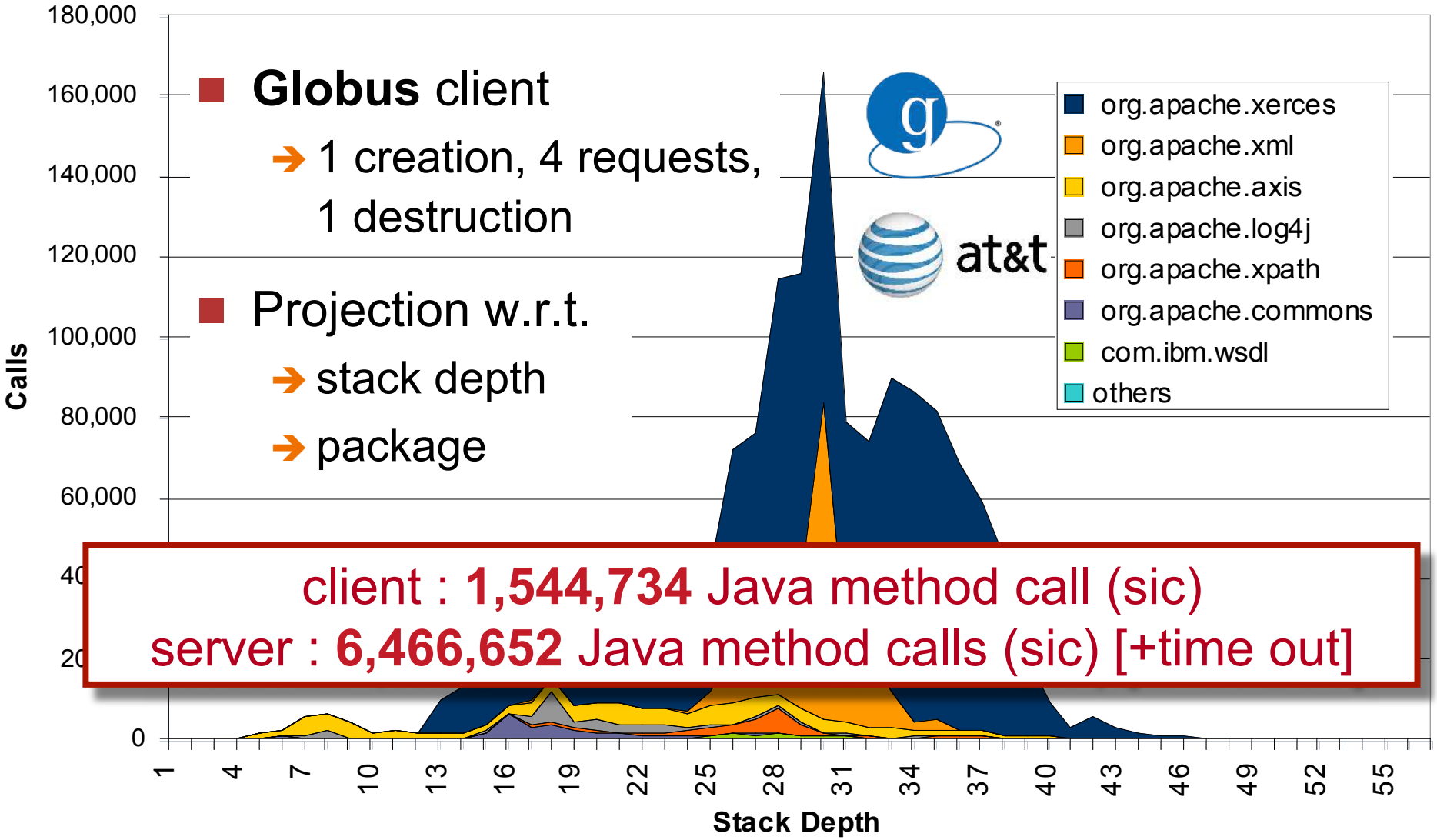
one RPC request,

- **2065** individual invocations
- > **50** C-functions
- > **140** C++ classes

Source: [TKF2009]

Unmanageable?

- **Globus** client
  - 1 creation, 4 requests, 1 destruction
- Projection w.r.t.
  - stack depth
  - package



client : **1,544,734** Java method call (sic)
server : **6,466,652** Java method calls (sic) [+time out]

The Impact of Web Service Integration on Grid Performance. Taïani, Hiltunen, Schlichting, HPDC-14, 2005

Unmanageable?

ars technica

ALL APPLE ASK ARS BUSINESS GADGETS GAMING MICROSOFT OPEN SOURCE SCIE

TECH POLICY

NEWS GUIDES REVIEWS FUTURE OF CARS

› Gear & Gadgets Essential toys, tools, and hardware

Netflix never used its $1 million algorithm due to engineering costs

By Casey Johnston | Published April 13, 2012 4:25 PM

Netflix awarded a $1 million prize to a developer team in 2009 for an algorithm that increased the accuracy of the company's recommendation engine by 10 percent. But today it doesn't use the million-dollar code, and has no plans to implement it in the future, Netflix announced on its blog Friday. The post goes on to explain why: a combination of too much engineering effort for the results, and a shift from movie recommendations to the "next level" of personalization caused by the transition of the business from mailed DVDs to video streaming.

LATEST TO

Unmanageable?

ars technica

ALL APPLE ASK ARS BUSINESS GADGETS GAMING MICROSOFT OPEN SOURCE SCIE

TECH POLICY

NEWS GUIDES REVIEWS FUTURE OF CARS

› Gear & Gadgets

Essential toys, tools, and hardware

Netflix never used its $1 million algorithm due to engineering costs

By Casey Johnston | Published April 13, 2012 4:25 PM

Netflix awa **$1 million prize** veloper team in 2009 for an algorithm that increased the accuracy of the compar **recommendation** engine by 10 percent. But today it doesn't use the million-dollar code, and has no plans to implement it in the future, Netflix announced on its blog Friday. The post goes on to explain why: a combination of too much **too much engineering effort** n movie recommendations to the "next level" of personalization caused by the transition of the business from mailed DVDs to video streaming.

LATEST TO

NETFLIX

amazon webservices™

Apache

Large **Dynamic** **Complex**

› Gear & Gadgets

Essential toys, tools, and hardware

Netflix never used its $1 million algorithm due to engineering costs

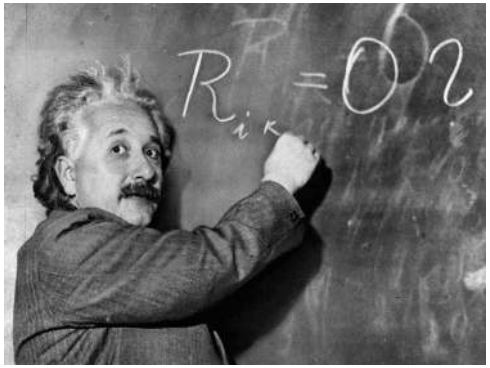
By Casey Johnston | Published April 13, 2012 4:25 PM

Netflix awa **$1 million prize** veloper team in 2009 for an
algorithm that increased the accuracy of the compar **recommendation**
engine by 10 percent. But today it doesn't use the million-dollar code, and
has no plans to implement it in the future, Netflix announced on its blog
Friday. The post goes on to explain why: a combination of too much
too much engineering effort n movie recommendations
to the "next level" of personalization caused by the transition of the
business from mailed DVDs to video streaming.

LATEST TO

Why is distribution hard?

- **Information** takes **time** to travel
  - ➔ Some DS protocols inspired from general relativity



- Machines and networks **fail**
  - ➔ If MTTF 4 years: 1M machines → 1 failure every 2 minutes

Impossibility Results

Asynchronous system with **crash** failures

- **Consensus** impossible (even if only one node crashes)
- **C**onsistency + **A**vailability + **P**artition tol. Impossible

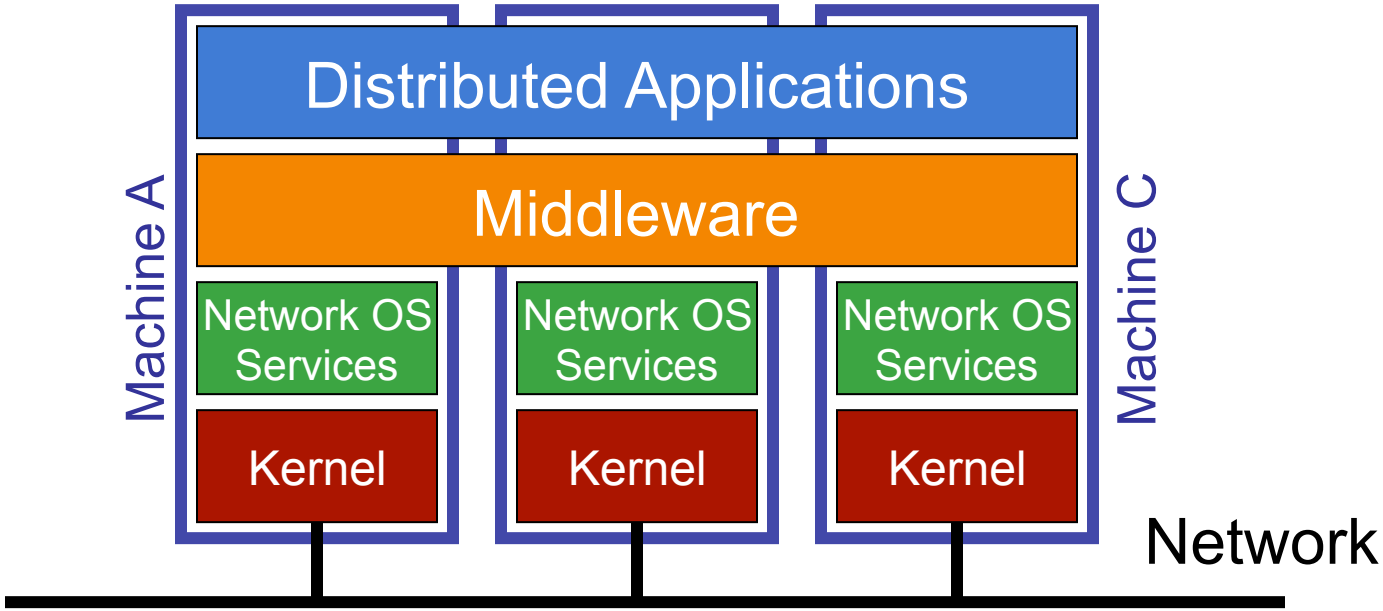
Consequences

- N **crash prone** machines **not** Turing complete

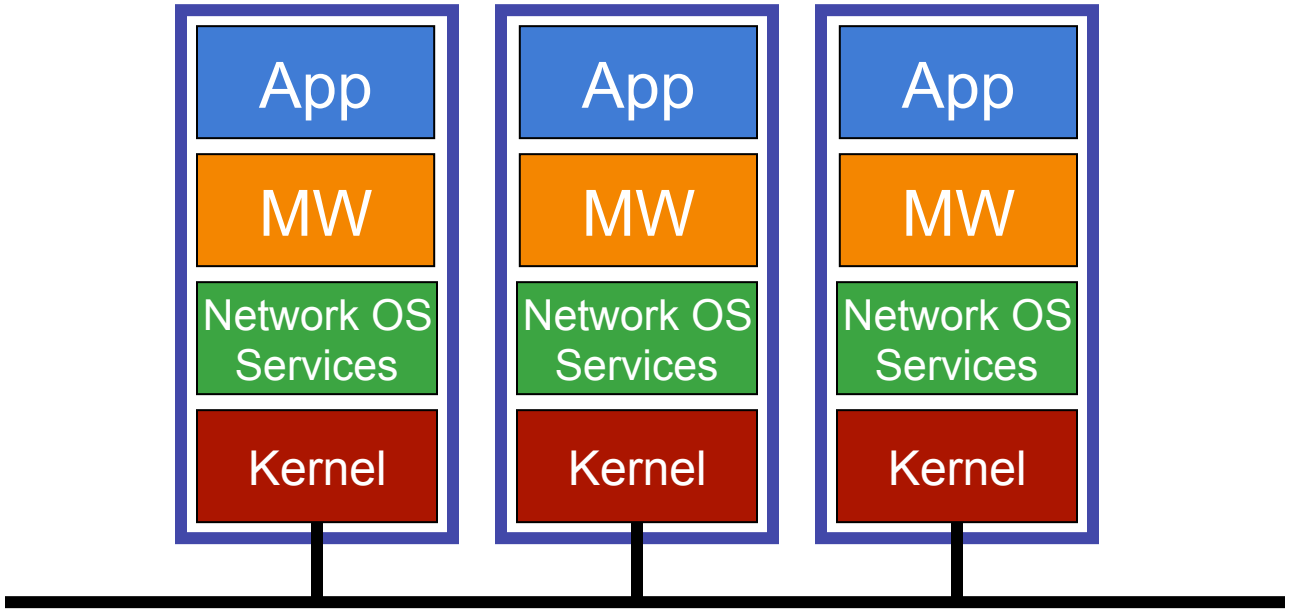
- Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. "Impossibility of distributed consensus with one faulty process." *Journal of the ACM (JACM)* 32.2 (1985): 374-382.
- Gilbert, Seth, and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." *ACM SIGACT News* 33.2 (2002): 51-59.
- Herlihy, Maurice, Sergio Rajsbaum, and Michel Raynal. "Computability in distributed computing: a tutorial." *ACM SIGACT News* 43.3 (2012): 88-110.

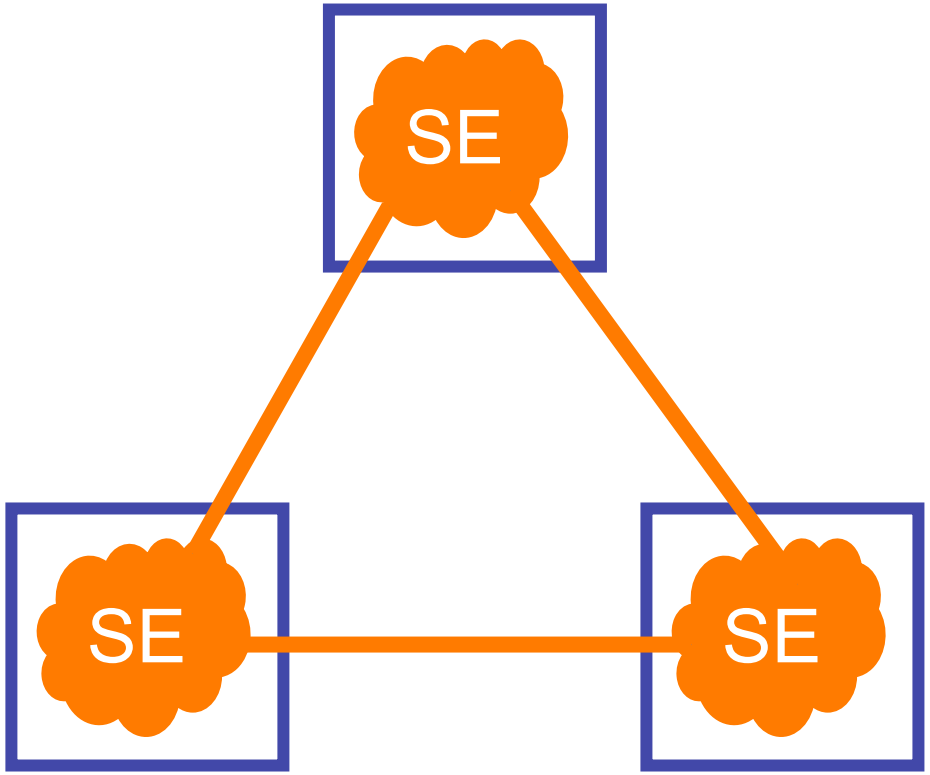
Progress so far: Middleware

- Goal: **"nice"** programming abstractions
  - ➔ Challenge: to hide or not to hide distribution?



In Practice

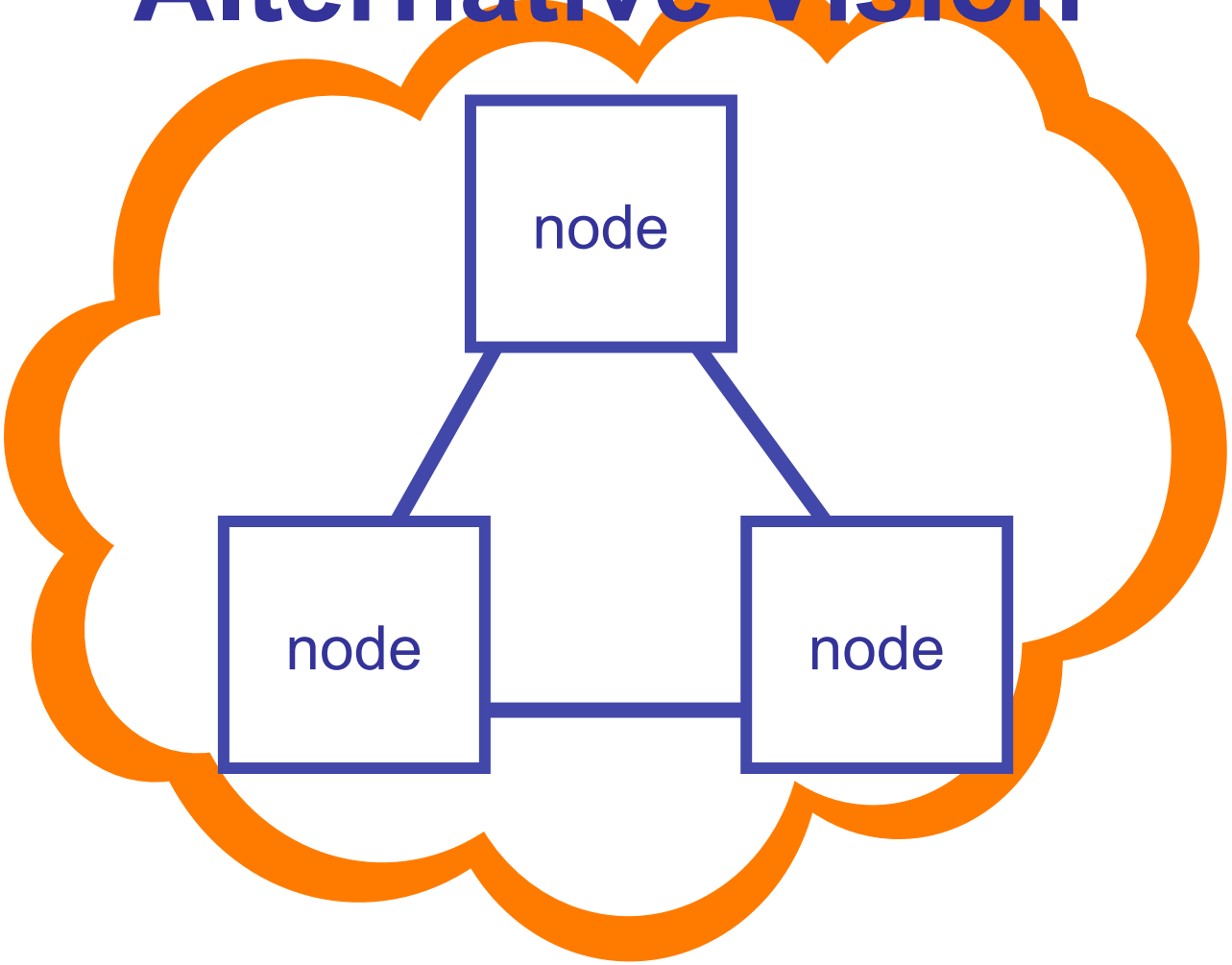

App
MW
Network OS Services
Kernel
App
MW
Network OS Services
Kernel
App
MW
Network OS Services
Kernel



Most of today's effort centred on programming **nodes**

Alternative Vision

SE



Tomorrow's systems will require a **holistic** approach.

The Holistic Challenge

- (Strong) **consistency** is very **costly**
  - ➔ The **one-entity** metaphor only goes so far.
- **Large scale**: embrace an **inconsistent** world
  - ➔ Co-existence of past and present in the same system
  - ➔ Partial adaptation
  - ➔ Emerging behaviour
- **Challenges**
  - ➔ Programming Models
  - ➔ Interoperability
  - ➔ Safety
  - ➔ Security

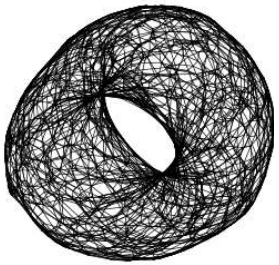
Outline

- A call to arms: engineering large scale
- Examples of ways forward

Example 1

Dionasys project (2014-2017)

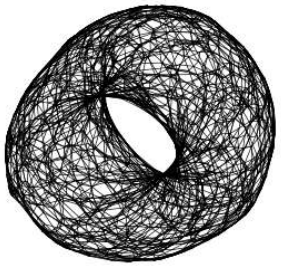
- Target
  - **Large scale**, **heterogeneous** systems
  - E.g. IoT + cloud + VANETs + mobiles
- Aim
  - Principled **holistic** SE approach
- Tools
  - **Self-stabilizing** overlays
  - **Declarative** language
  - **Components**



Jelasity et al [JMB09]

Example 2

- Application of **components** + **DSL** to **gossip** protocols
  - ➔ Whisper + GossipKit

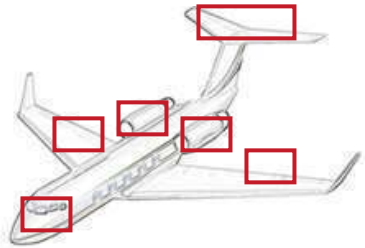


Jelasity et al [JMB09]

- Lin S., Taiani F., Bertier M., Blair G. S., Kermarrec A.-M. (2011). Transparent componentisation: high-level (re)configurable programming for evolving distributed systems. ACM SAC ’11
- GossipKit: A Unified Component Framework for Gossip, François Taïani, Shen Lin, Gordon S. Blair, IEEE TSE, Feb. 2014

Gossip Protocols

- "Traditional" Distributed Systems
  - ➔ Deterministic with strong guarantees
  - ➔ Does not scale well



- Gossip (aka epidemic) Protocols
  - ➔ Introduce some '**chaos**'
  - ➔ Goal: system to **converge** to a desirable outcome
  - ➔ But some nodes might be left out

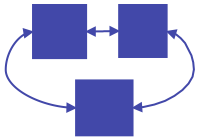
- Trading determinism for **scalability** & **robustness**

Gossip Protocols (cont.)

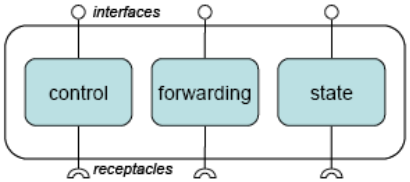
- Principles
  - ➔ leverage **rumour-like** propagation of information
  - ➔ large applicability: aggregation, broadcast, clustering
  - ➔ often **composed** to realised higher-level services
- Conceptually simple
  - ➔ typically symmetric behaviour
  - ➔ key notions of **state**, information **flows**,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

Applying Components to Gossip

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

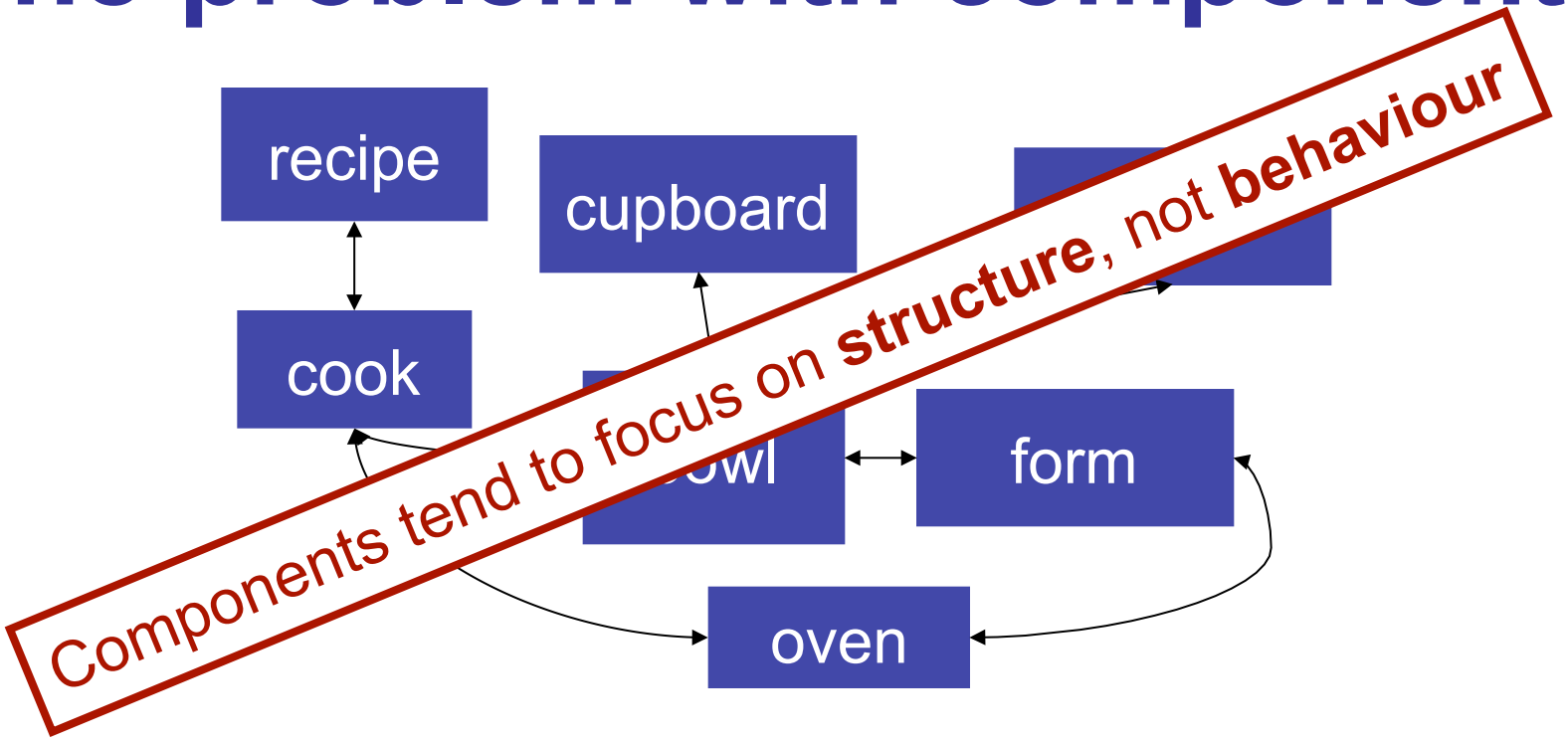


- Clear **structure**, explicit **dependencies**



- 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

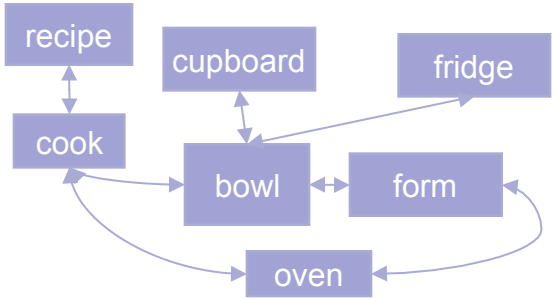
Applying SDL to Gossip

- **Spec. lang. and DSL:** High-level per node description
  - ➔ Lotos, Estelle, PLAN-P, Mace …

bake

- **Macro-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

Behaviour rather than structure



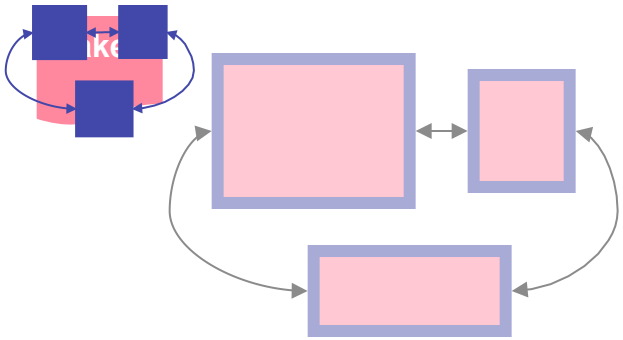

```
add(yohourt,1)
add(milk,2)
add(flour,3)
add(butter,1)
```

Can we build a hybrid approach that combines the strengths of components & high-level languages?

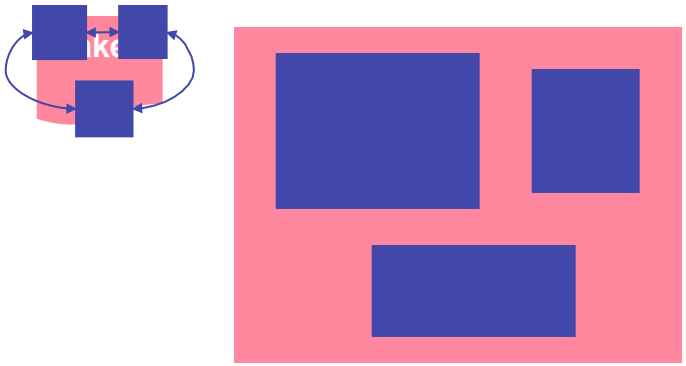
```
form.putIn(oven)
bake()
```

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

structure + behaviour = ?



encapsulation

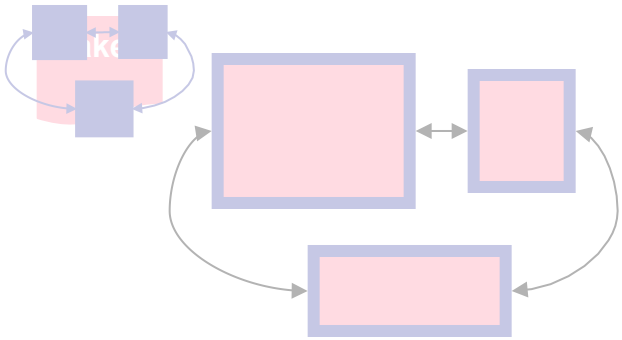


orchestration

- 😐 **tangling** behaviour & structure
- 😐 ‘breaks’ **encapsulation**
- 😐 tension **flexibility** vs. **scattering**

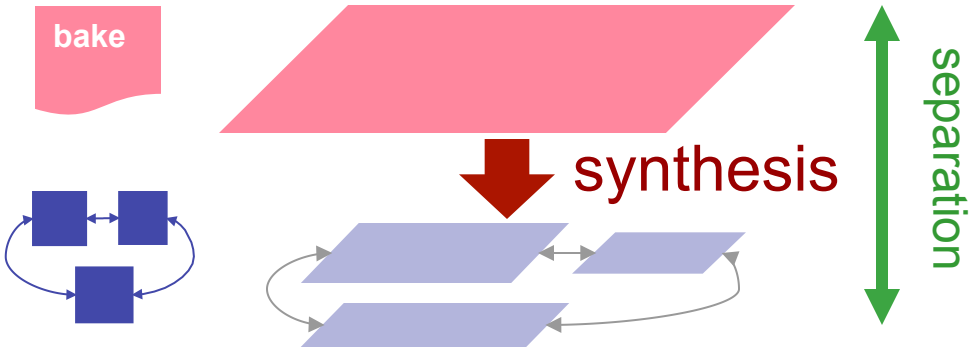
- 😐 **complex** composition
- 😐 tension **structural** needs vs. **programmatic** ones

structure + behaviour = ?



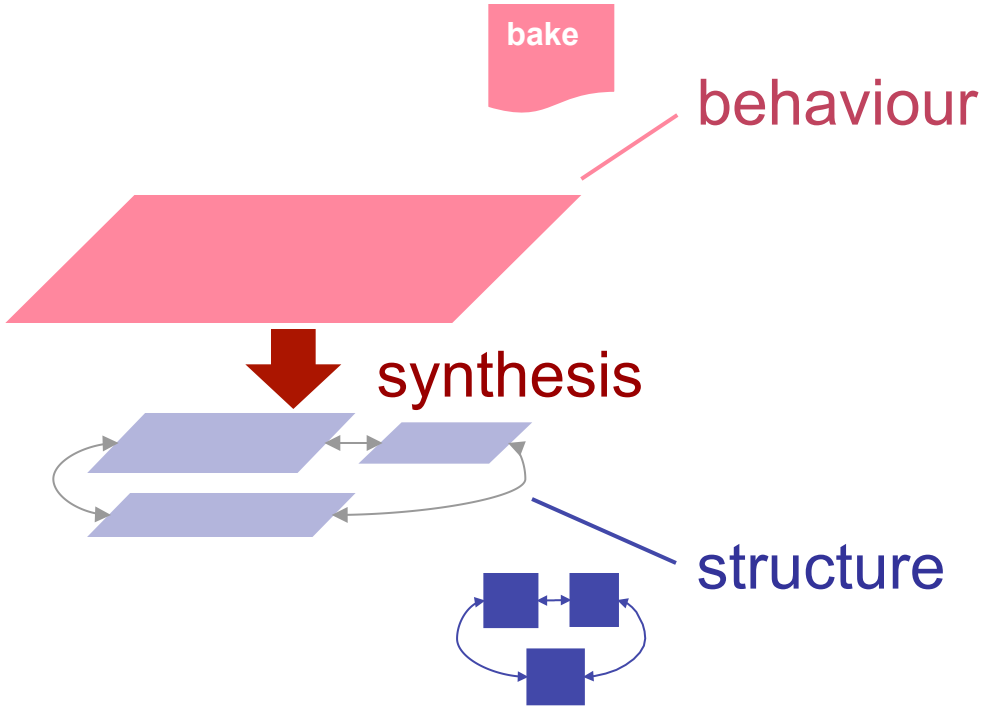
encapsulation

orchestration



transparent componentisation

Transparent Componentisation

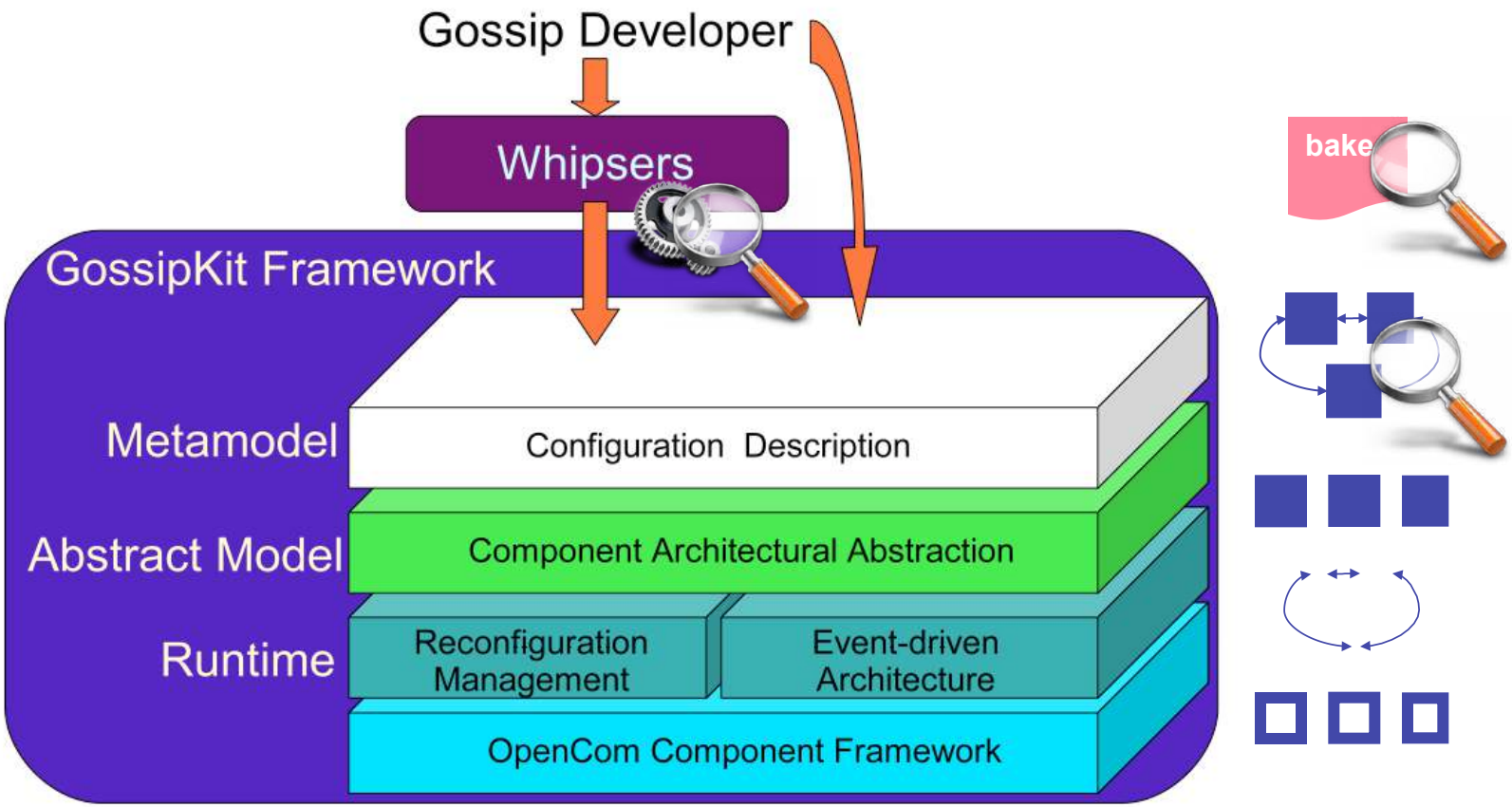


☺ simple
☺ concise
☺ high-level

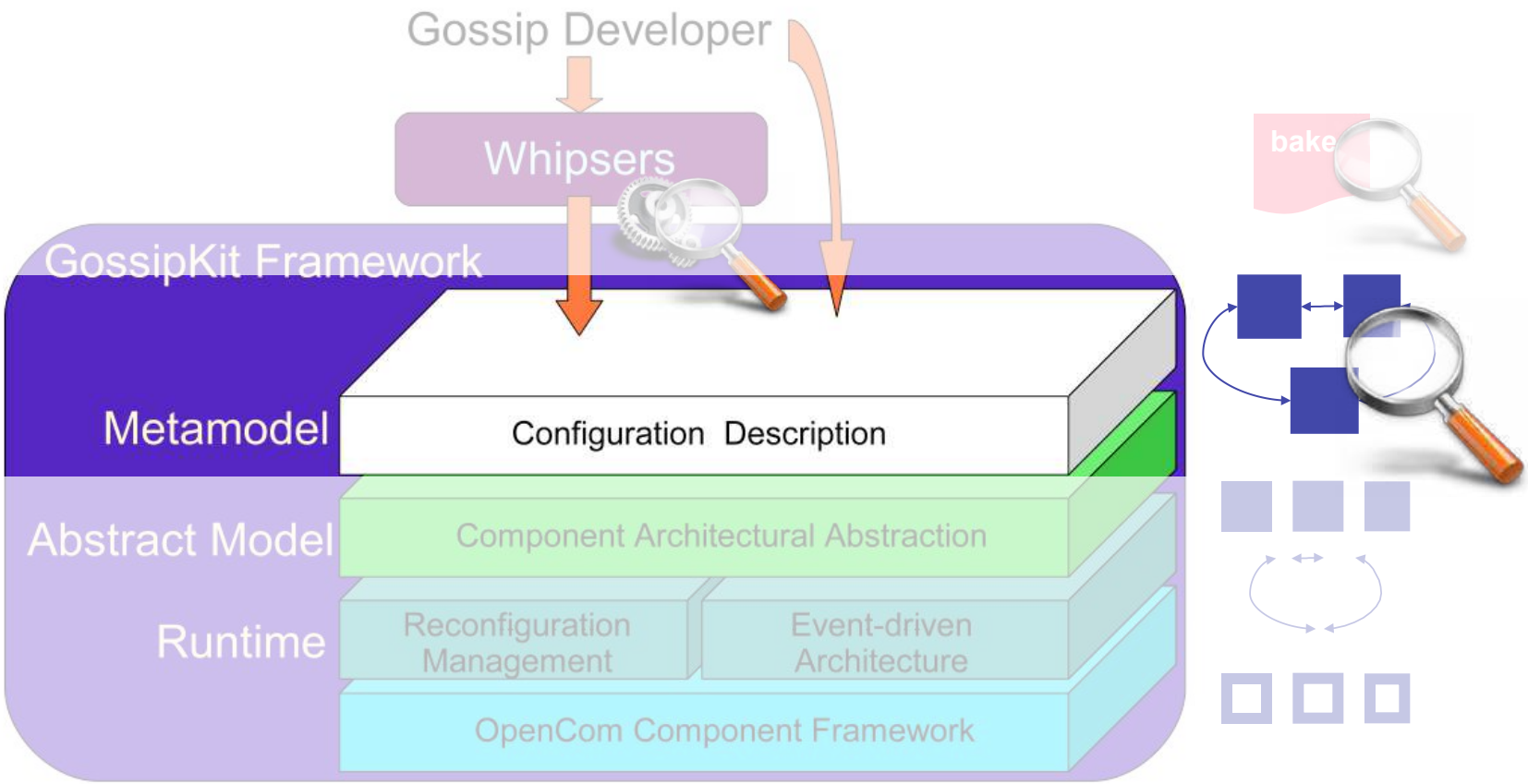
☺ modular
☺ reusable
☺ (re)configurable

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

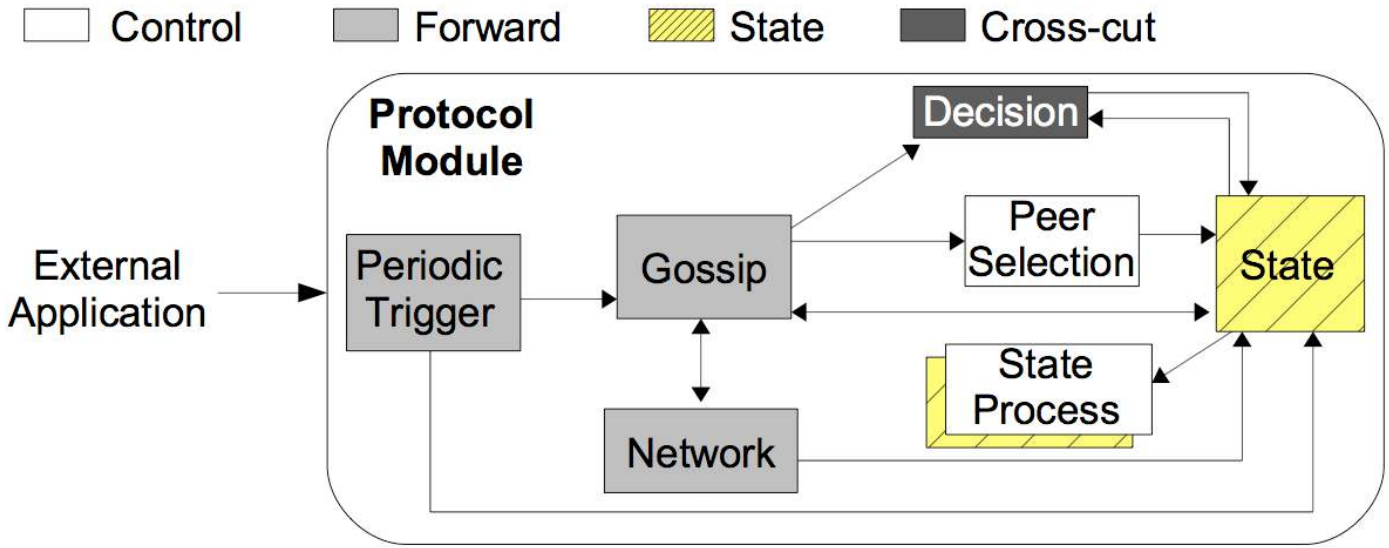
The WhispersKit Architecture



The WhispersKit Architecture

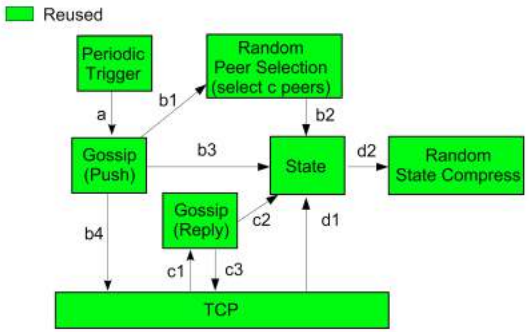


GossipKit

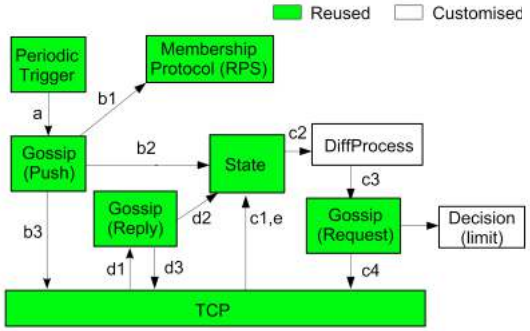


- A component **framework** for **epidemic** protocols
  - based on analysis of 30 gossip protocols
  - **event-based**
  - XML-based **configuration** for component composition
  - targets abstraction, modularity, reuse, evolvability

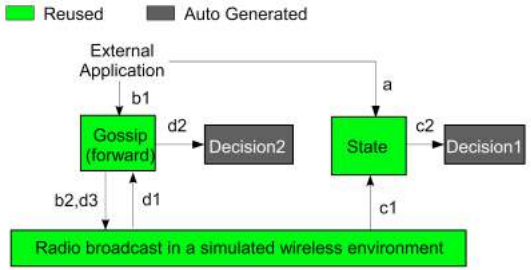
GossipKit Examples



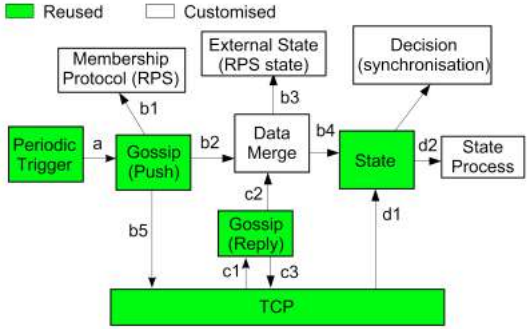
RPS



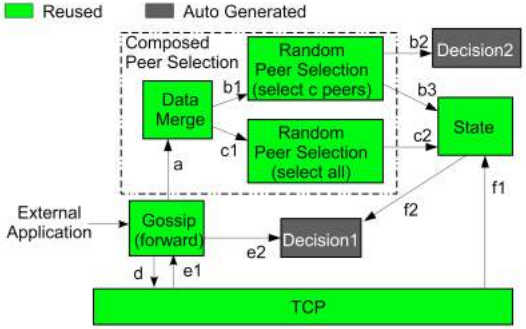
Anti-Entropy



Wireless broadcast

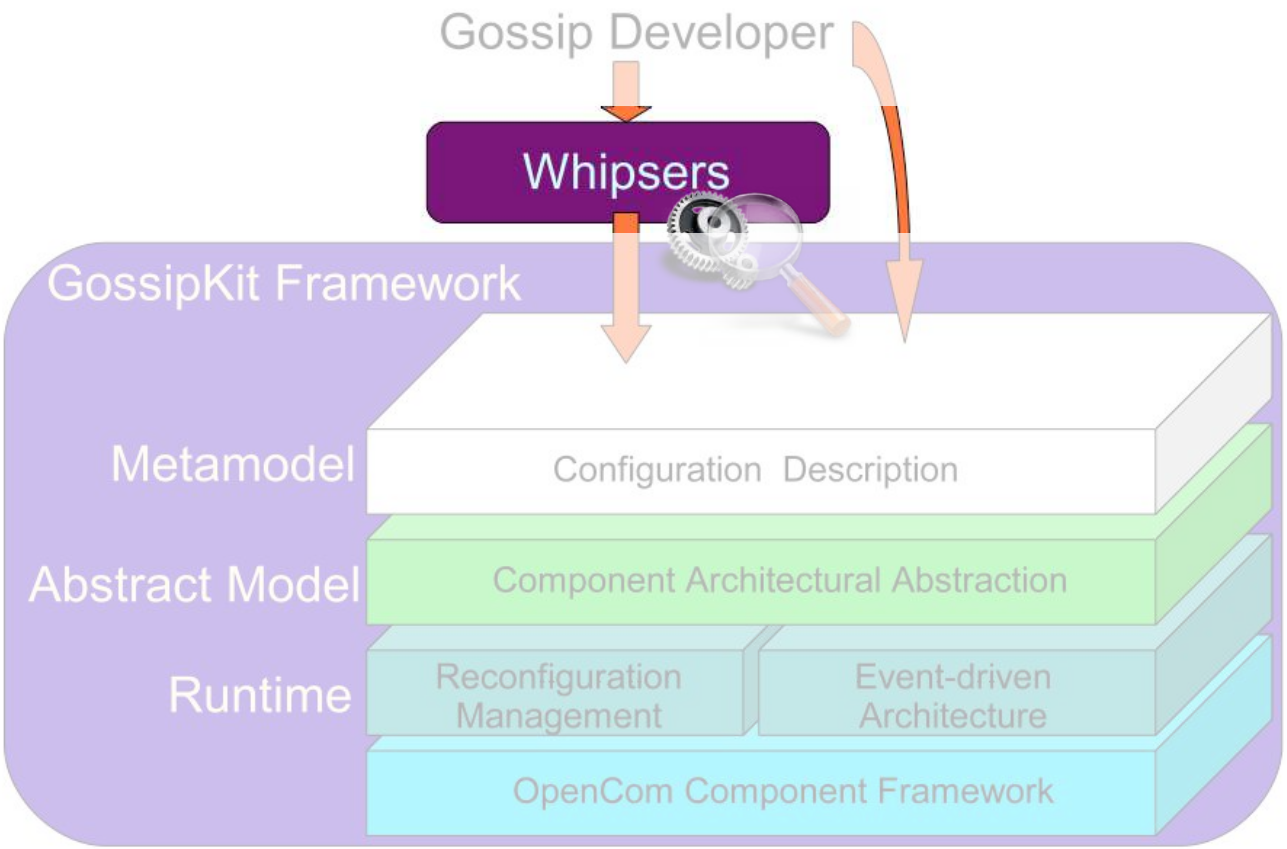


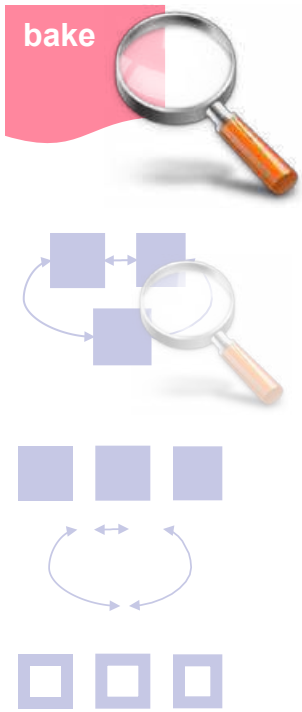
T-Man



SCAMP

The WhispersKit Architecture





Whispers

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

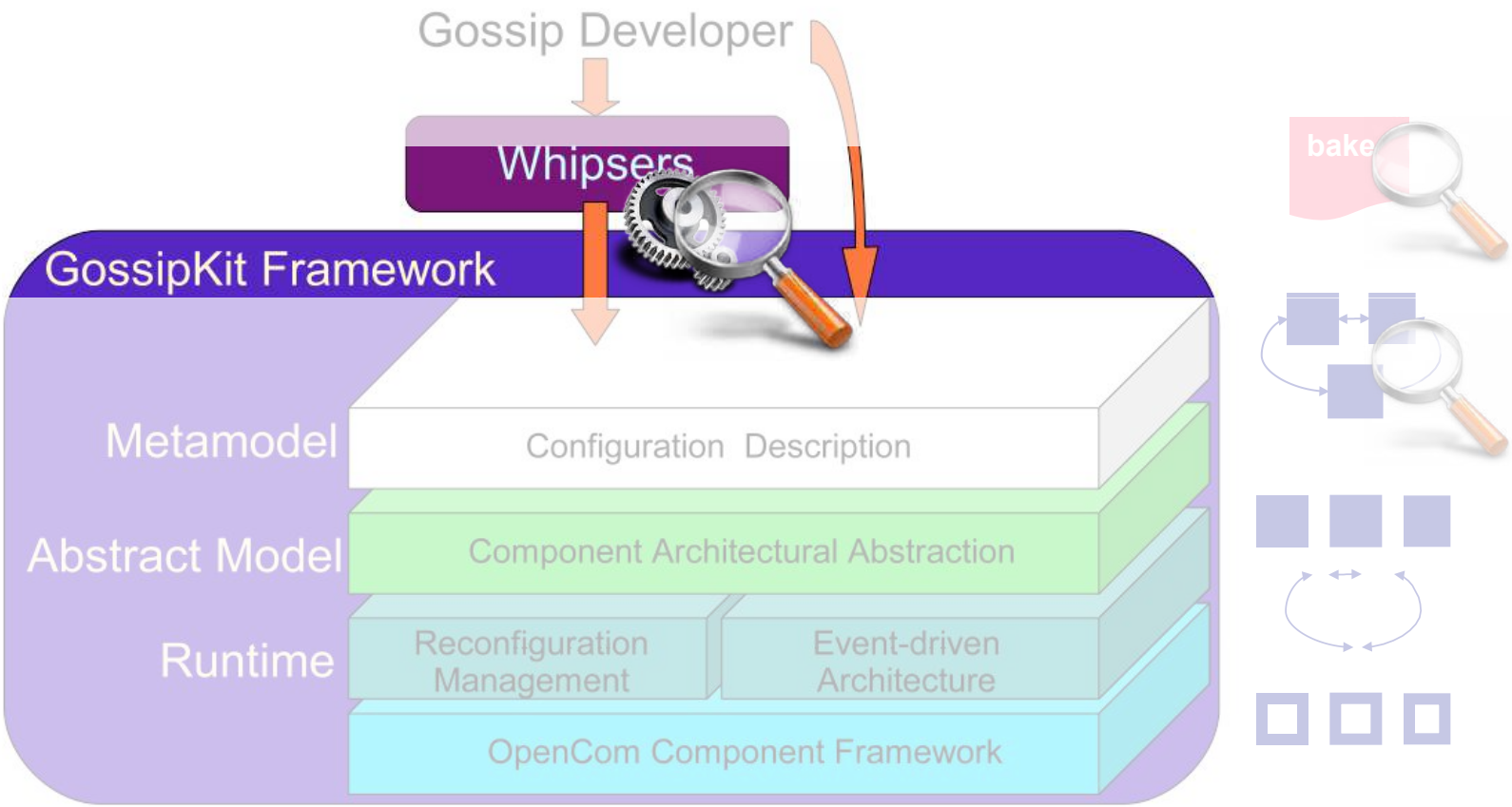
```
protocol {..}                    // protocol block
every (time) {..}                // periodic behaviours
wait (Event e type T) {..}       // reactive behaviours
foreach(n in nodeSet)            // distribution
synchronised {..}                // pairwise data exchange
State state = new State[fields][size]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

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
```

Jelasity, M., Guerraoui, R., Kermarrec, A.-M., and van Steen, M. (2004). The peer sampling service: experimental evaluation of unstructured gossip-based implementations. Middleware ’04

The WhispersKit Architecture



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
```

bake

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**

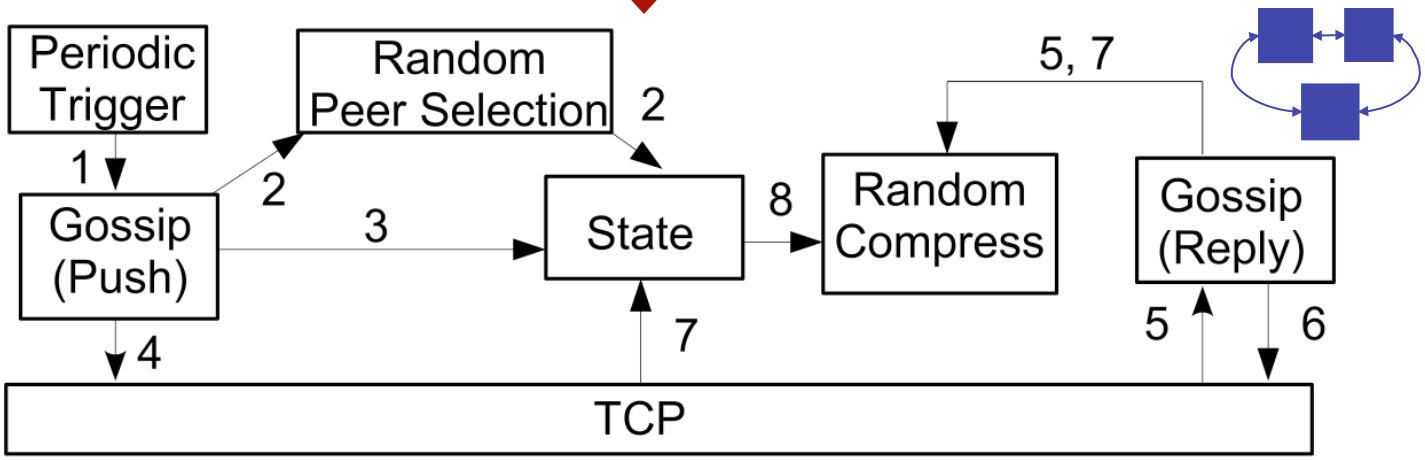
message **reply**

5 **retrieve** local_sample

6 **reply** local_sample

7 **extract** remote_sample from message

8 RandomCompress (local_sample, remote_sample)



Distributed Reconfiguration

- A developer describes new behaviour in Whispers.
- The platform uses component representation
  - ➔ to compute minimal set of changes;
  - ➔ 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

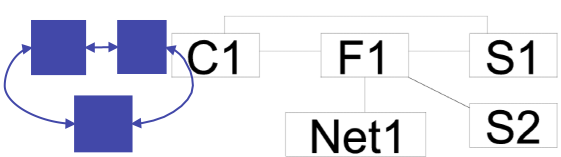
System Behaviour B

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. }}}
```

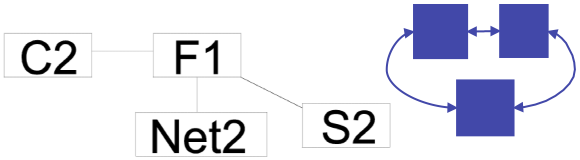
Component mapping

Component mapping



Component Configuration A

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



Component Configuration B

Distributed Reconfiguration

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

coarse grained *fine grained*

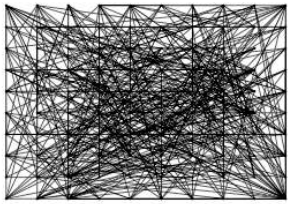


Figure 5.6: Initial random graph maintained by RPS

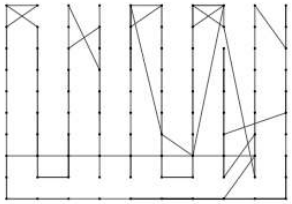


Figure 5.7: 5th rounds since 1st reconfiguration

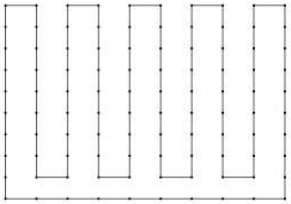


Figure 5.8: Ring constructed at the 11th round

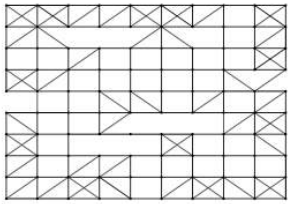


Figure 5.9: Topology at the 20th round

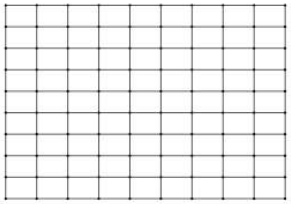


Figure 5.10: Grid constructed at the 23rd round

Conclusion

- The world is **distributed**, the world is **large**
- **Distribution** is more than concatenation
  - ➔ **Failures** and **uncertainties**
- **Large-scale** distributed systems even more so
  - ➔ **Information** takes **time** to travel
- Novel **software engineering** approaches needed
  - ➔ Away from node-centric view
  - ➔ Holistic yet loosely coupled approaches ideal


Thank you

Task Failures at Google

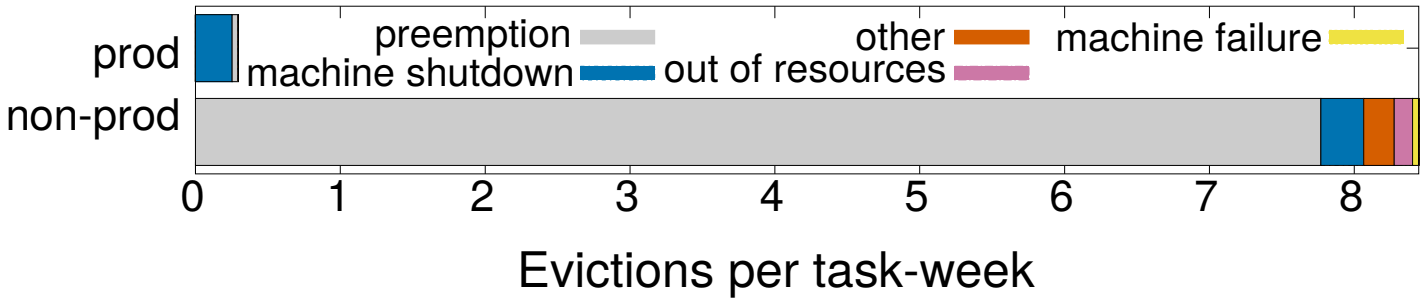


Figure 3: Task-eviction rates and causes for production and non-production workloads. *Data from August 1st 2013.*

- Source: Large-scale cluster management at Google with Borg
  Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, John Wilkes
  EuroSys'2015, Bordeaux, France (2015)

(Some) References

- **[TKF09] CosmOpen: Dynamic reverse-engineering on a budget (journal version)** *François Taïani, Marc-Olivier Killijian, Jean-Charles Fabre*, SP&E, 39(18): (Dec. 2009) pp. 1467-1514 (48p.), doi: http://dx.doi.org/10.1002/spe.943.
- **[THS05] The Impact of Web Service Integration on Grid Performance** *François Taïani, Matti Hiltunen, Rick Schlichting*, The 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), 2005, pp.14-23 (10 p.), doi: http://doi.ieeecomputersociety.org/10.1109/HPDC.2005.1520929.
- [**FLP85]** Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. **Impossibility of distributed consensus with one faulty process.** Journal of the ACM (JACM) 32.2 (1985): 374-382.
- **[GL02]** Gilbert, Seth, and Nancy Lynch. **Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.** ACM SIGACT News 33.2 (2002): 51-59.

(Some) References

- **[HRR12]** Herlihy, Maurice, Sergio Rajsbaum, and Michel Raynal. **Computability in distributed computing: a tutorial.** ACM SIGACT News 43.3 (2012): 88-110.
- **[JMB09]** Mark Jelasity, Alberto Montresor, and Ozalp Babaoglu. 2009. **T-Man: Gossip-based fast overlay topology construction**. *Comput. Netw.* 53, 13 (August 2009), 2321-2339.
- **[LTBBK11]** Lin S., Taiani F., Bertier M., Blair G. S., Kermarrec A.-M. (2011). **Transparent componentisation: high-level (re)configurable programming for evolving distributed systems**. ACM SAC ’11, pp. 203–208
- **[TLG14] GossipKit: A Unified Component Framework for Gossip** *François Taïani, Shen Lin, Gordon S. Blair*, IEEE TSE, vol 40, Issue 2 (Feb. 2014), pp. 123-136 (17p)doi: http://dx.doi.org/10.1109/TSE.2013.50.

(Some) References

- **[JGK04]** Jelasity, M., Guerraoui, R., Kermarrec, A.-M., and van Steen, M. (2004). **The peer sampling service: experimental evaluation of unstructured gossip-based implementations**. Middleware ’04