CUSO Doctoral School

Nov. 26-27, 2015, University of Neuchâtel

Large-Scale Systems And Self-Organization

François Taïani

UMR IRISA

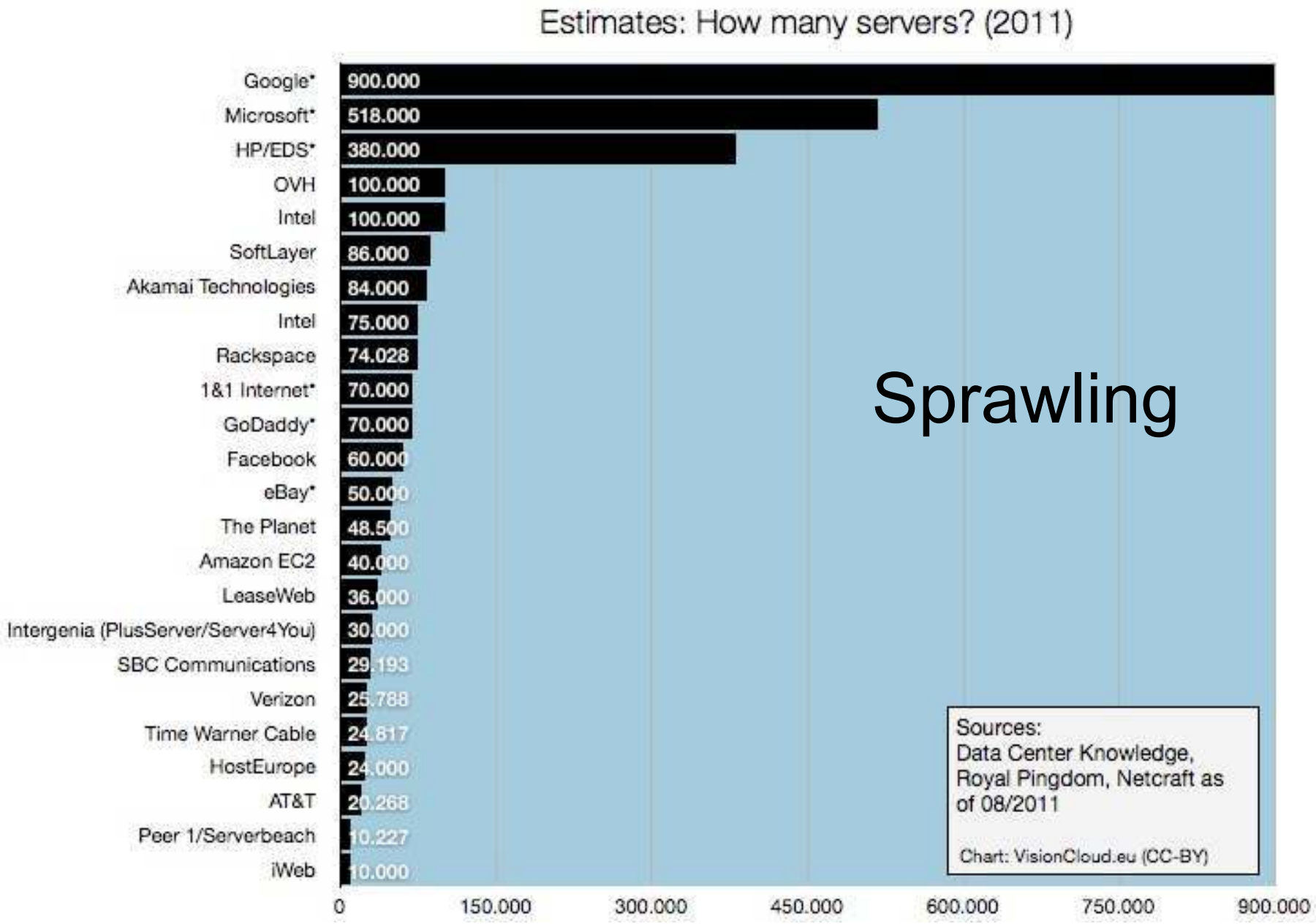
UNIVERSITÉ DE RENNES 1

Inria INVENTEURS DU MONDE NUMÉRIQUE

esir ECOLE SUPERIEURE D'INGENIEURS DE RENNES

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

- **Orbacus**: CORBA middleware (interoperable RPC)
  - ➔ Implemented in C++
  - ➔ On top of Linux (POSIX)
  - ➔ One “Hello world” request processed

- How many C++ classes and C functions involved (server)?

- How many local invocations (server)?

Complex

Portability

Interoperability

Transparency

...

one RPC request,

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

Source: [TKF2009]

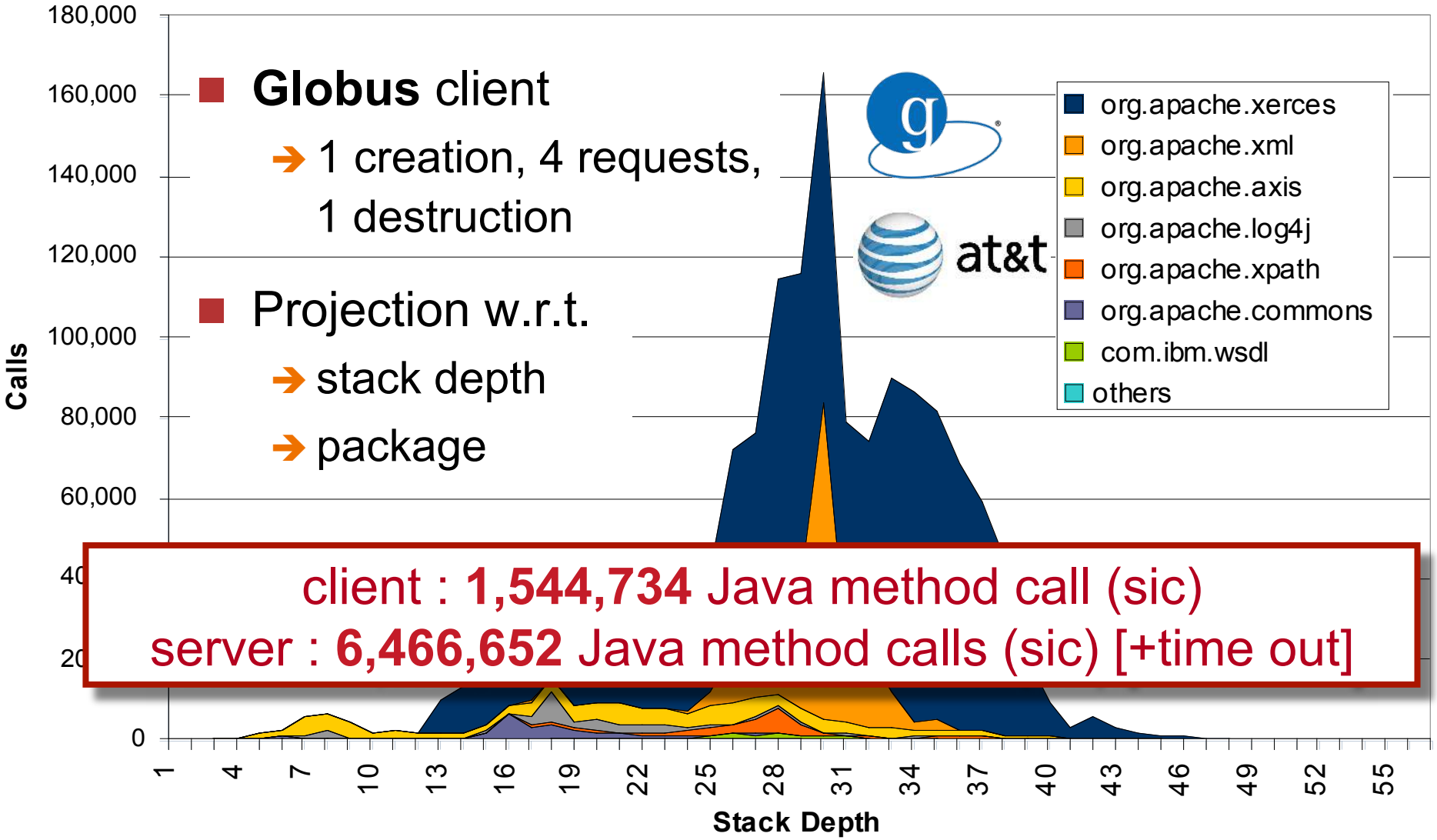
Unmanageable?

- **Globus** client
  - ➔ 1 creation, 4 requests, 1 destruction

- How many local Java invocations?
  - ➔ On the client?
  - ➔ On the server?

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

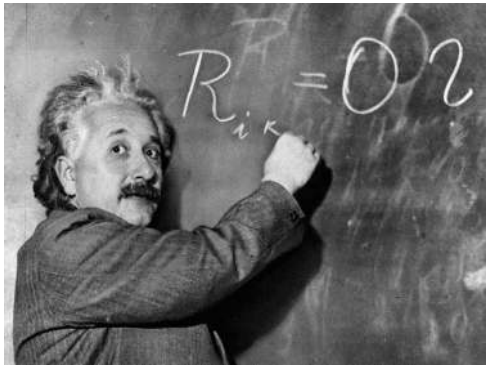
By Casey Johnston | Published April 13, 2012 4:25 PM

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 [FLP85]
  - ➔ even if only one node crashes
  - ➔ (and this assumes we know **all** the nodes beforehand)
- **C**onsistency + **A**vailability + **P**artition tol. Impossible
  - ➔ Brewer's theorem [GL02]

Consequences

- N **crash prone** machines **not** Turing complete [HRR12]

Architecting Scale & Distribution

- **Scale + Distribution + Openness + Dynamicity**
  - ➔ Impossible to envisage all runtime scenarios
  - ➔ Very hard to stop system for evolution (long running)

- What developers need
  - ➔ **Abstraction** (intent rather than mechanisms)
  - ➔ Ability to **change system on the fly** (dynamic evolution)

- Two properties provided by **self-organizing systems**

Self-Organization

- Goal: **emergence** of **structure** in a distributed systems
  - ➔ nodes, software components, policies, ...
- Needs to way to **tell** system what is "**desirable**"
  - ➔ should often be a function of system's context
  - ➔ Ideal specification: high-level, generic, expressive, holistic
- Needs **mechanisms** to get there
  - ➔ Ideal mechanisms: scalable, robust, fault-tolerance

Concrete example with most of these properties:
Self-organizing overlays

Self-Organization

- Goal: **emergence** of **structure** in a distributed systems
  - ➔ nodes, software components, policies, ...
- Needs to way to **tell** system what is "**desirable**"
  - ➔ should often be a function of system's co
  - ➔ Ideal specification: high-level, generic, [illegible]sive, holistic
- Needs **mechanisms** to get there
  - ➔ Ideal mechanisms: scalable, robus[illegible]lt-tolerance

Concrete example with most of these properties:
Self-organizing overlays

Outline

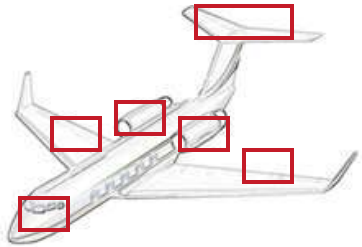
- Gossip Protocols
- The case of self-organizing overlays
- Self-organization++: collective survival
- Conclusion

Outline

- **Gossip Protocols**
- The case of self-organizing overlays
- Self-organization++: collective survival
- Conclusion

Traditional Distribution

- Typical distributed protocols:
  - ➔ do not rely on ‘chance’
  - ➔ provide **deterministic** guarantees



- But
  - ➔ either rely on a **central** core of servers for coordination
    - e.g. Zookeeper, Map Reduce frameworks (Hadoop, Spark)
  - ➔ or **not very scalable** ($O(n^k)$, k= 2,3)

Gossip Protocols

- Alternative: **probabilistic** approach

- 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**

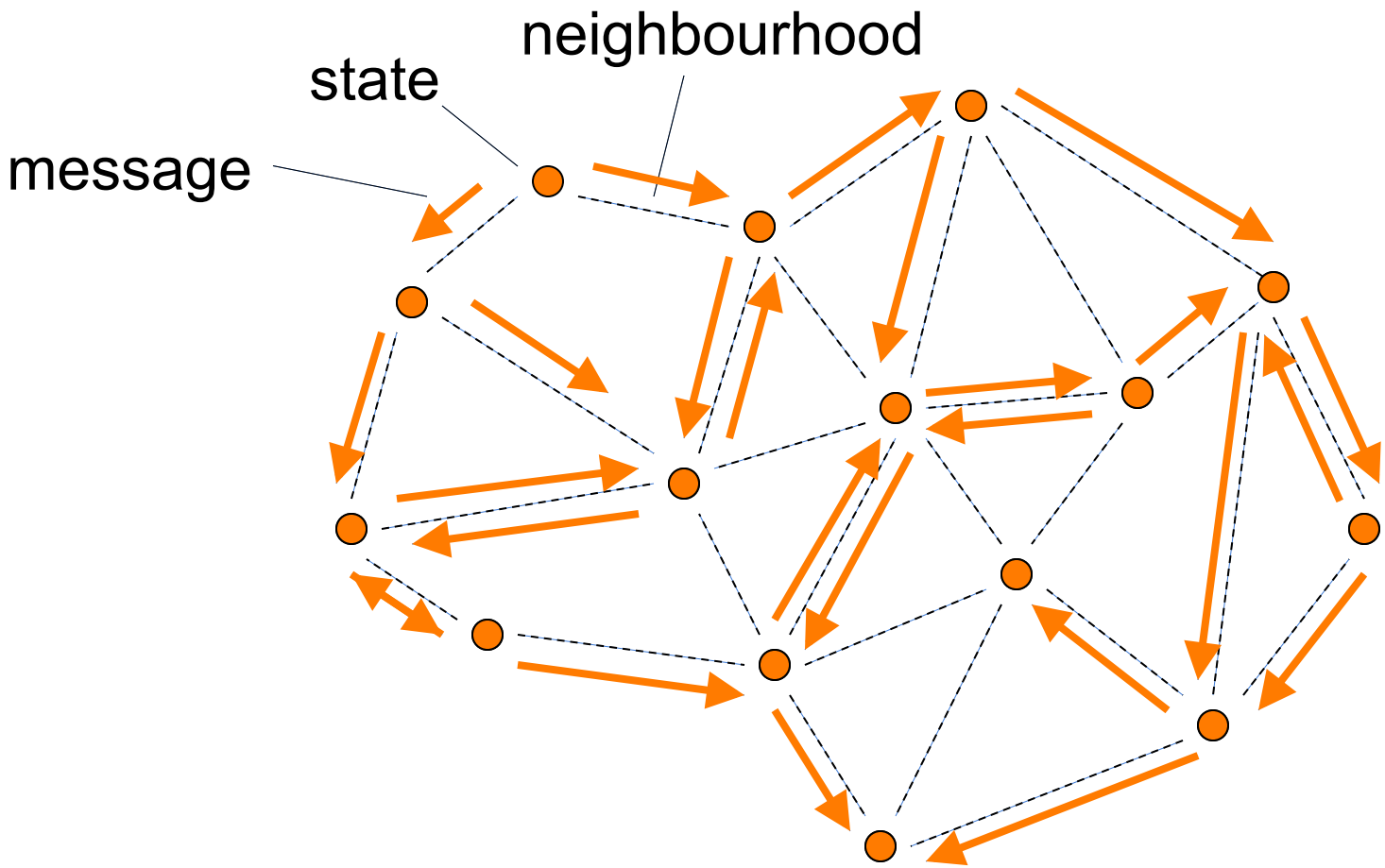
Example 1: Overlay Multicast

- **Overlays:** "logical network" on top of TCP/IP
  - ➔ Foundation of many P2P systems

- **Deterministic multicast**
  - ➔ Flooding → highly inefficient
  - ➔ Spanning tree → very efficient, but complex, costly, brittle

- **Gossip multicast**: for each new broadcast, each node
  - ➔ on first reception randomly pick k targets in the rest of the system (**k** = ‘fanout’), and forward msg to them
  - ➔ (discard duplicate reception)

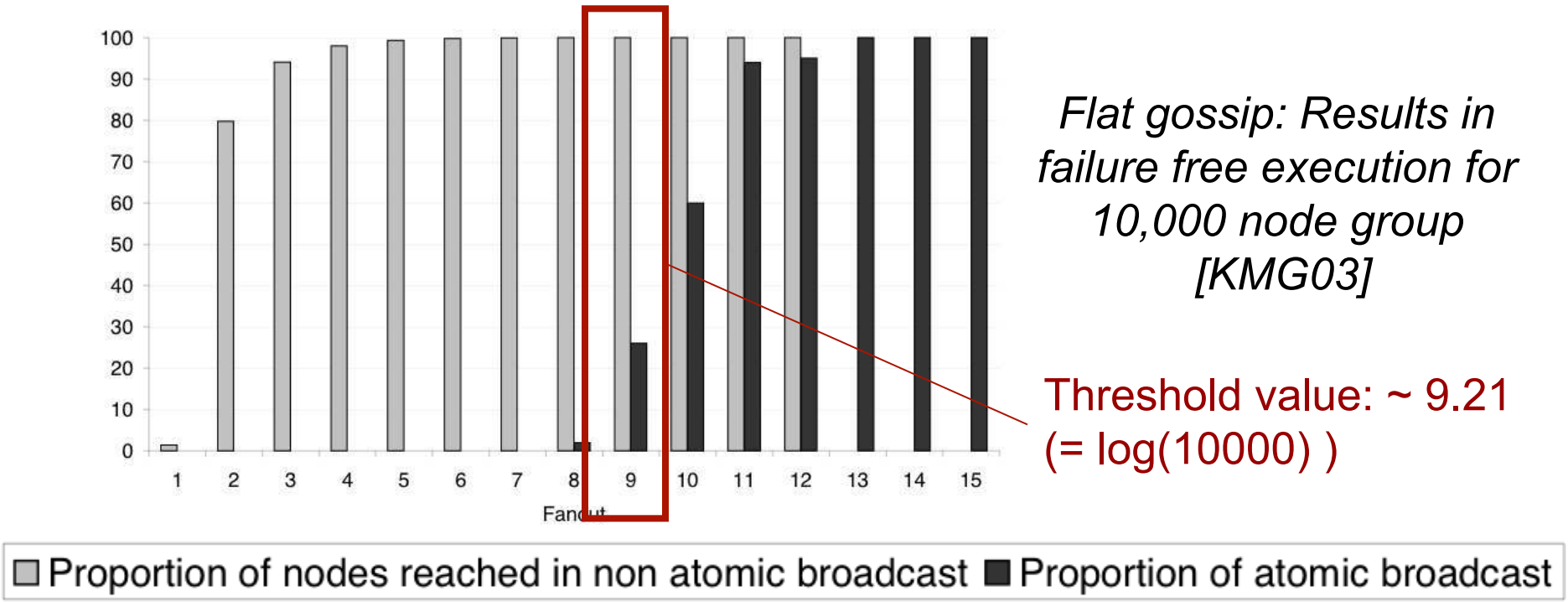
Example 1: Multicast



- How to choose **k** (fanout) to reach everyone?

Which k? Bimodal behaviour

- (Kemarrec, Massoulie, Ganesh / Microsoft Research)



Flat gossip: Results in failure free execution for 10,000 node group [KMG03]

Threshold value: ~ 9.21 (= log(10000))

Gossip: Key Ingredients

State ➔ Stochastic Data exchange ➔ New State

- What **state** to maintain?
  - ➔ msg already seen (multi-cast)
  - ➔ sensor value (aggregation, averaging)
  - ➔ neighbours list (topology construction, membership)
  - ➔ user profile (social networks)
- **When** to gossip? (periodic vs. event based)
- With **which probability** and **with whom** to gossip?
- Which information to **exchange**?
- How to **compute** new state?

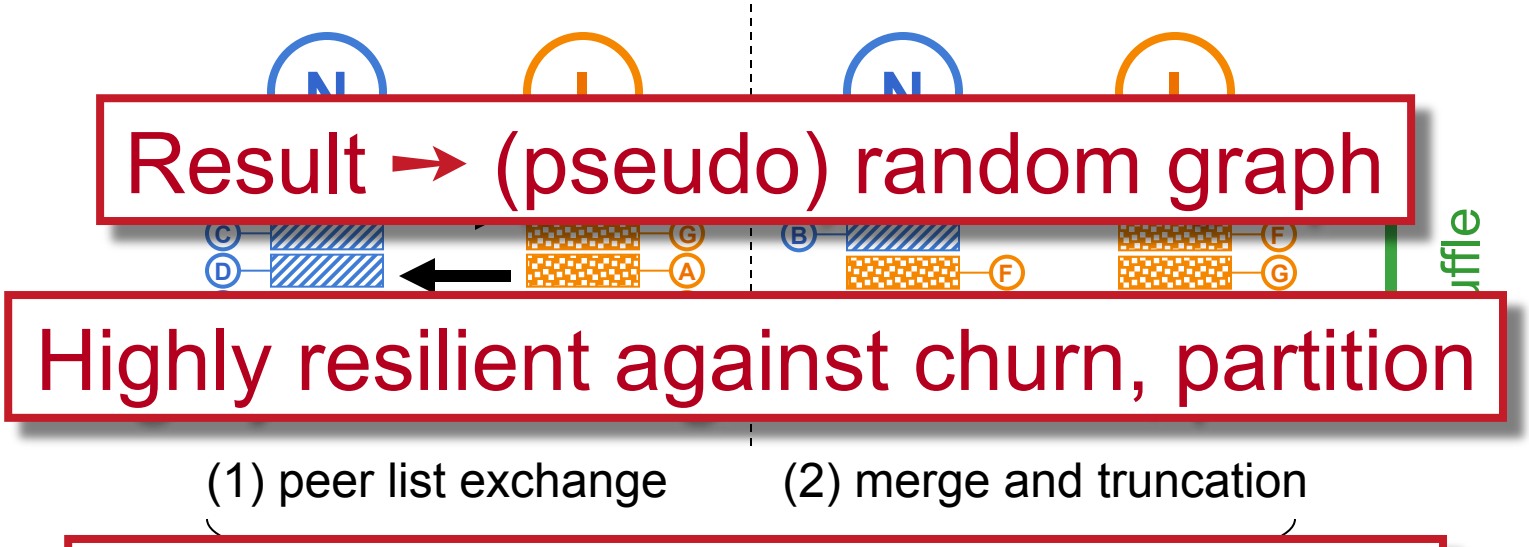
Ex. 2: Peer Sampling

- "Random Peer Sampling Service" [JGK04]
  - ➔ Membership service: who is in the system?

- **State**: list of k neighbours (k = system param)

- Main idea: **Periodically**: each node **n**
  - ➔ picks one neighbour **i**
  - ➔ sends own list of neighbours, receives that of **i**
  - ➔ **i** and **n merge**, **shuffle**, **truncate** the 2 lists

Random Peer Sampling (cont.)



Result → (pseudo) random graph

Highly resilient against churn, partition

(1) peer list exchange (2) merge and truncation

Small diameter (good for multicast)

Other Uses of Gossip

- Self-organisation in WSN: Who should do what?
- Knowledge aggregation: e.g. average temperature
- Information dissemination (failure detection, AWS S3)
- Self-organizing overlays

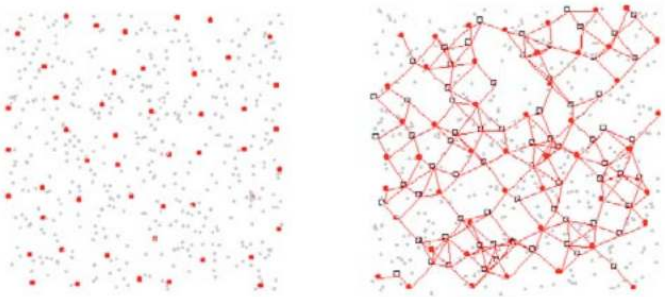


Figure 2: A 300×300 m area network with an average of 20 neighbors per node representing: (a) the router selection, (b) the energy state distribution.

From [LGKVB06]

Outline

- Gossip Protocols
- **The case of self-organizing overlays**
- Self-organization++: collective survival
- Conclusion

From Chaos to Self-organization

From gossip protocols → self-organizing overlays

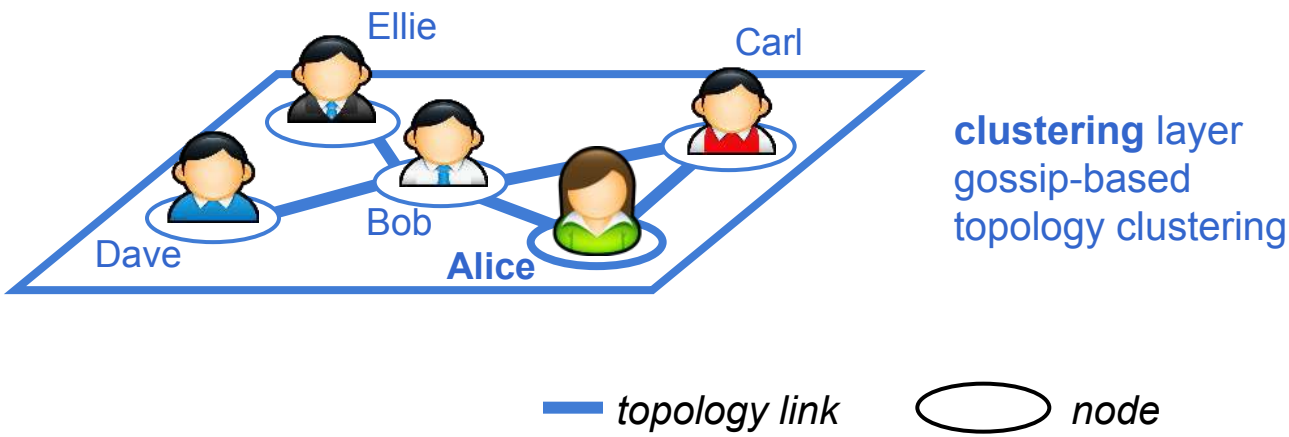
- **Intuition** of scalable self-organization :
  - → Local greedy procedure
  - → Global emergent properties (= organization)

- **Greedy** → Self-stabilizing, fault-tolerant
- **Local** → Decentralized, scalable
- **Randomized** → Flexible, fault-tolerant

Self-Organizing Overlays

Vicinity [VS05], **T-Man** [JMB09], **Gossple** [BFGKL10]

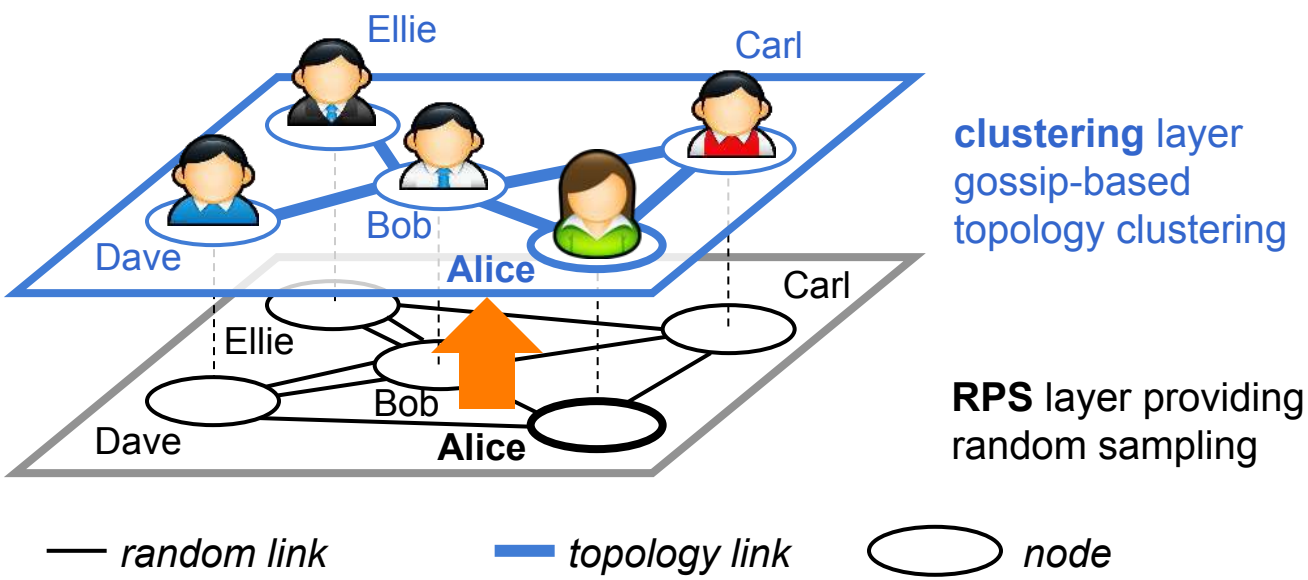
- State: each node has a position (e.g. in R^3)
- Metrics: Euclidian distance between node
- Goal: find *n* closest neighbours



Self-Organizing Overlays

Vicinity [VS05], **T-Man** [JMB09], **Gossple** [BFGKL10]

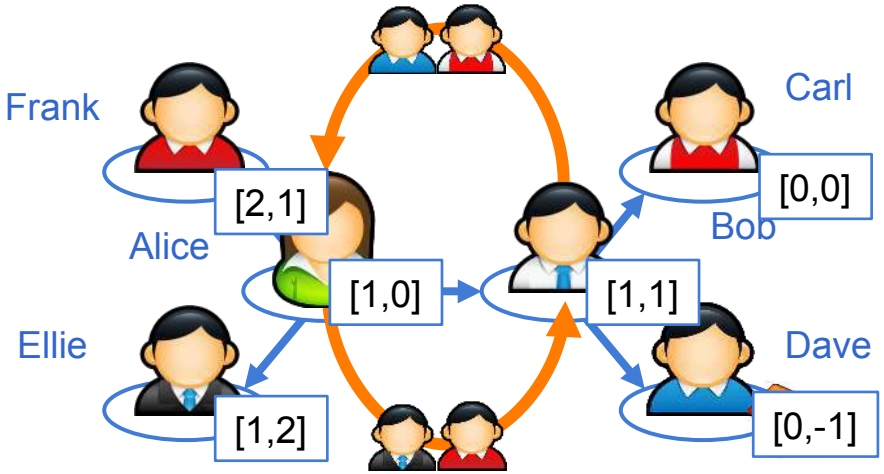
- Greedy local optimisation: in each round
  - ➔ each peer tries to **improve** its local neighbourhood
  - ➔ look for **potential new peers: (1) from random layer**



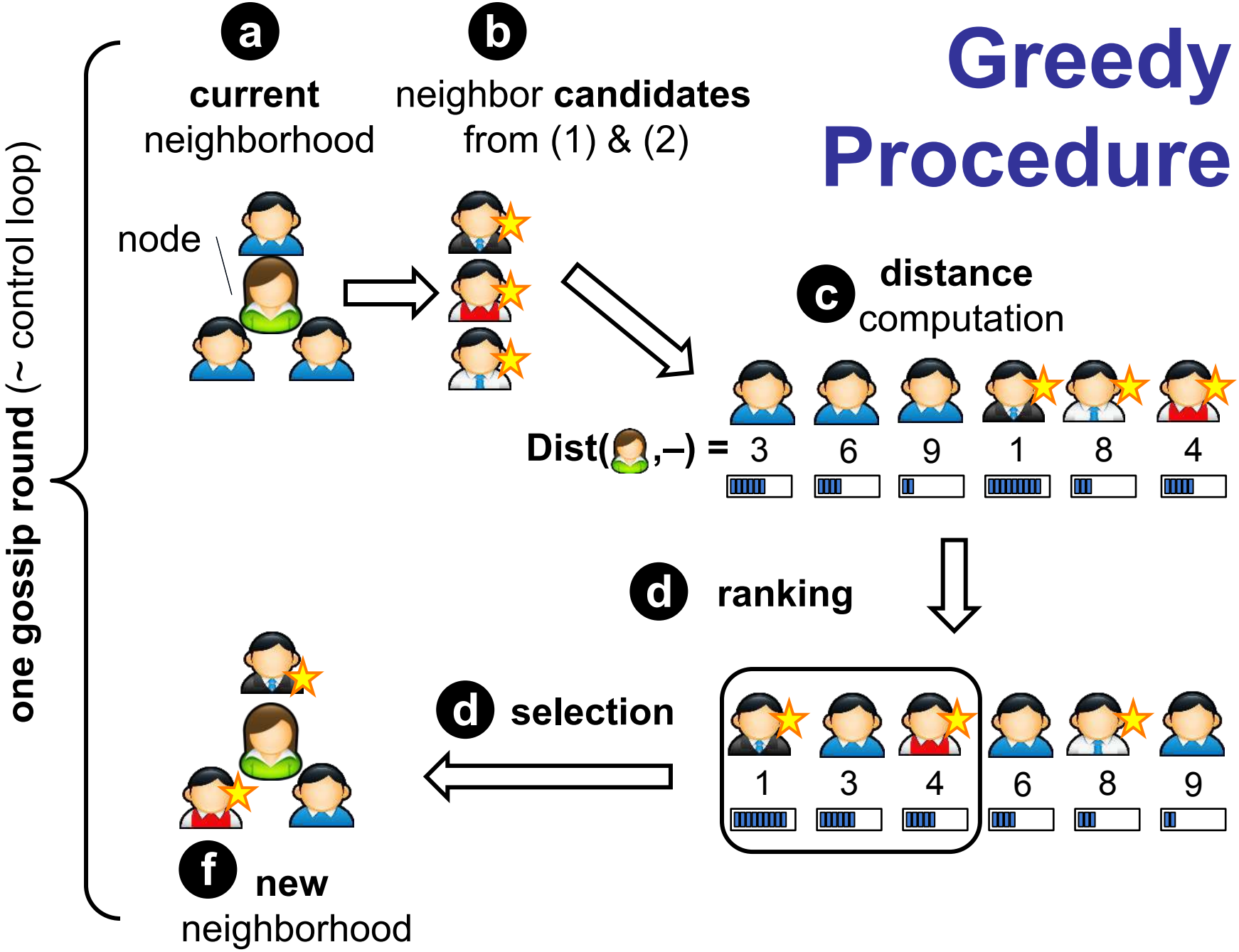
Self-Organizing Overlays

Vicinity [VS05], **T-Man** [JMB09], **Gossple** [BFGKL10]

- Greedy local optimisation: in each round
  - ➔ each peer tries to **improve** its local neighbourhood
  - ➔ look for **potential new peers: (2) from friends of friends**

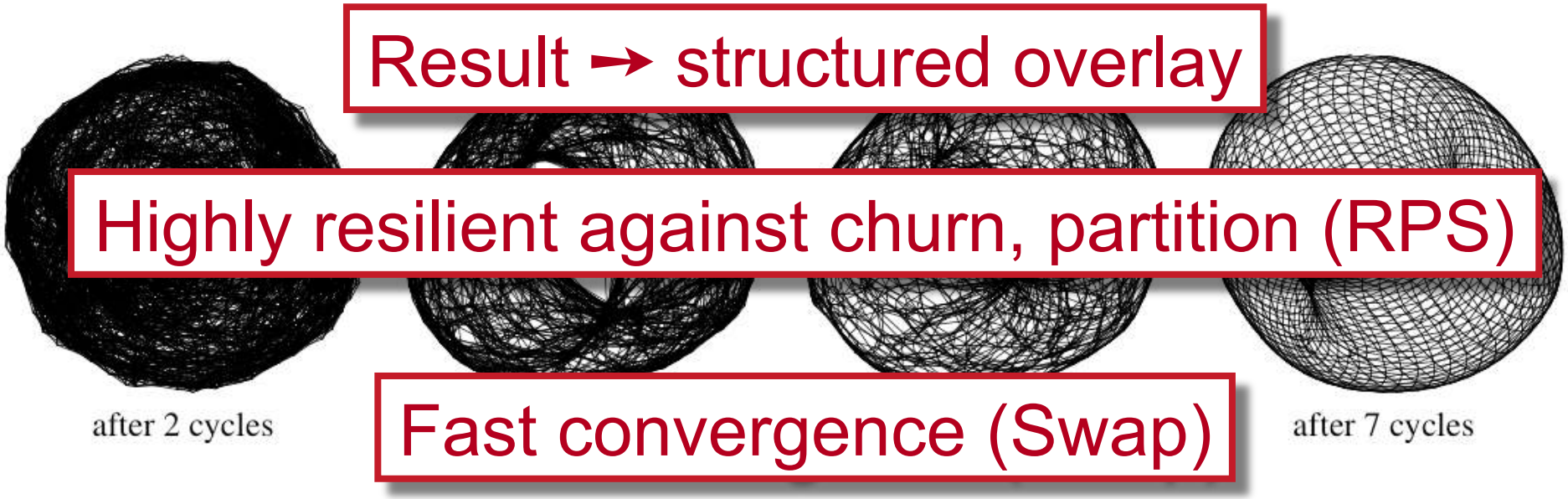


Greedy Procedure



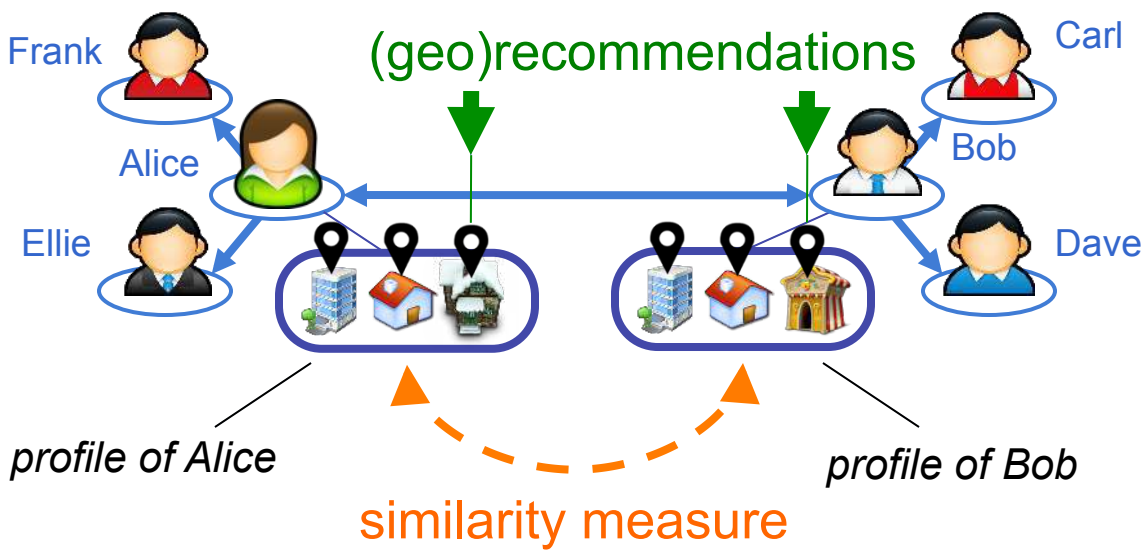
Self-Organization in Action

- (taken from [JMB09], T-Man)



Not limited to Euclidian space

- E.g. with user profiles (here metric = overlap)

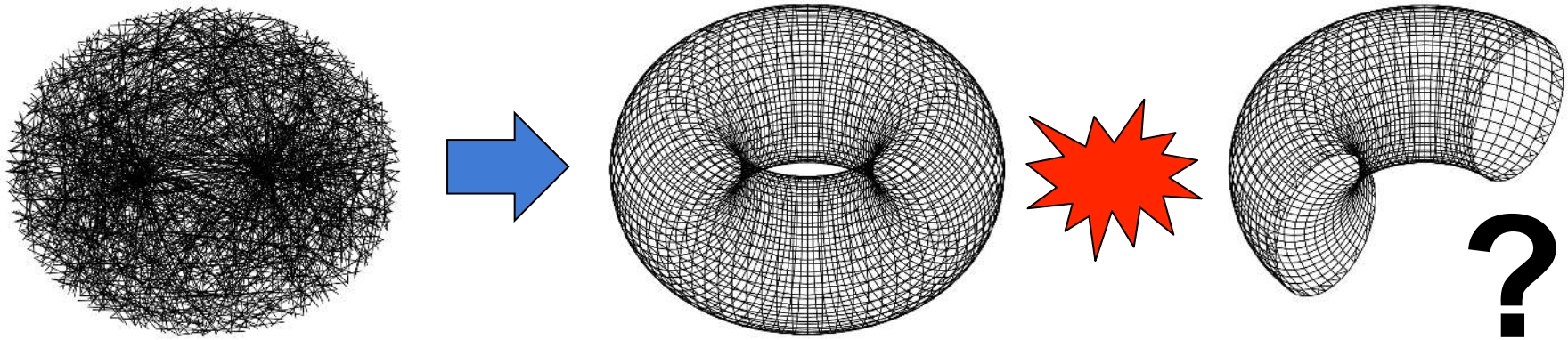


- Perform **decentralised KNN graph construction**
  - ➔ Intuition: **similar users** can **learn from each other**

Outline

- Gossip Protocols
- The case of self-organizing overlays
- **Self-organization++: collective survival**
- Conclusion

Towards Collective Survival

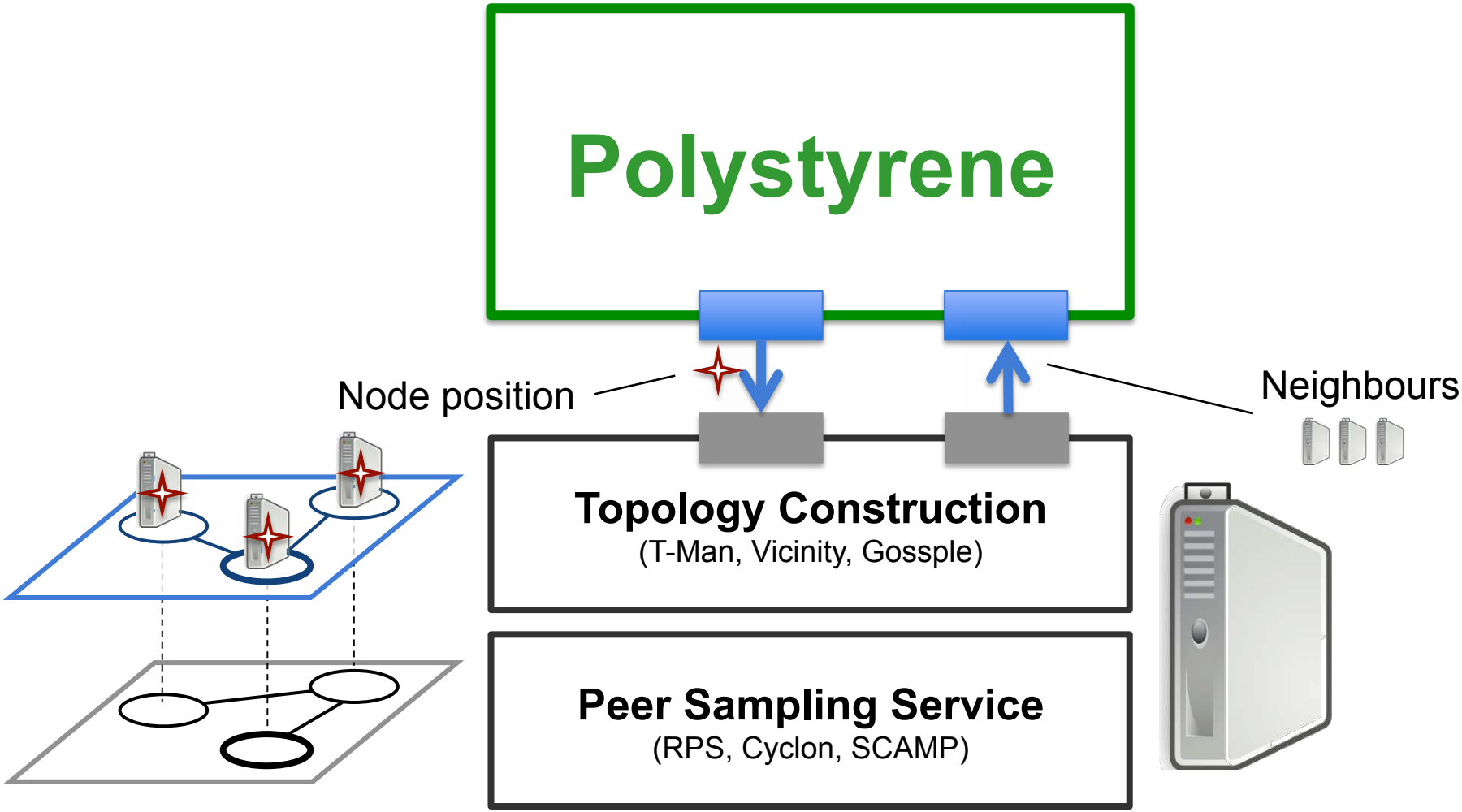


Catastrophic Failure on Self-Organizing Overlay

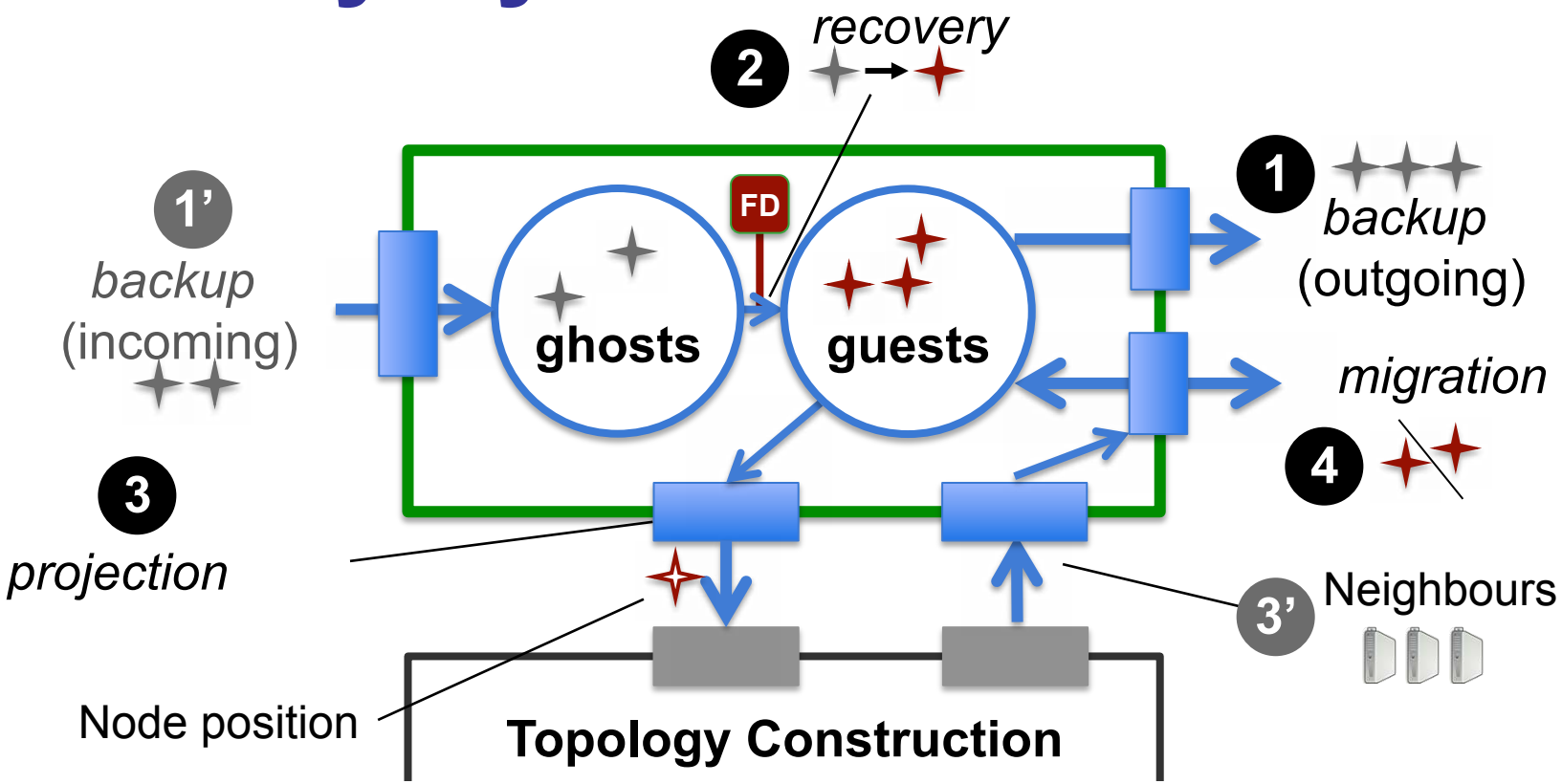
- The topology **heals**
- But the overall **shape** is **lost**

How to recreate whole shape from surviving nodes?

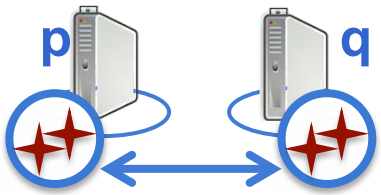
Polystyrene's Architecture

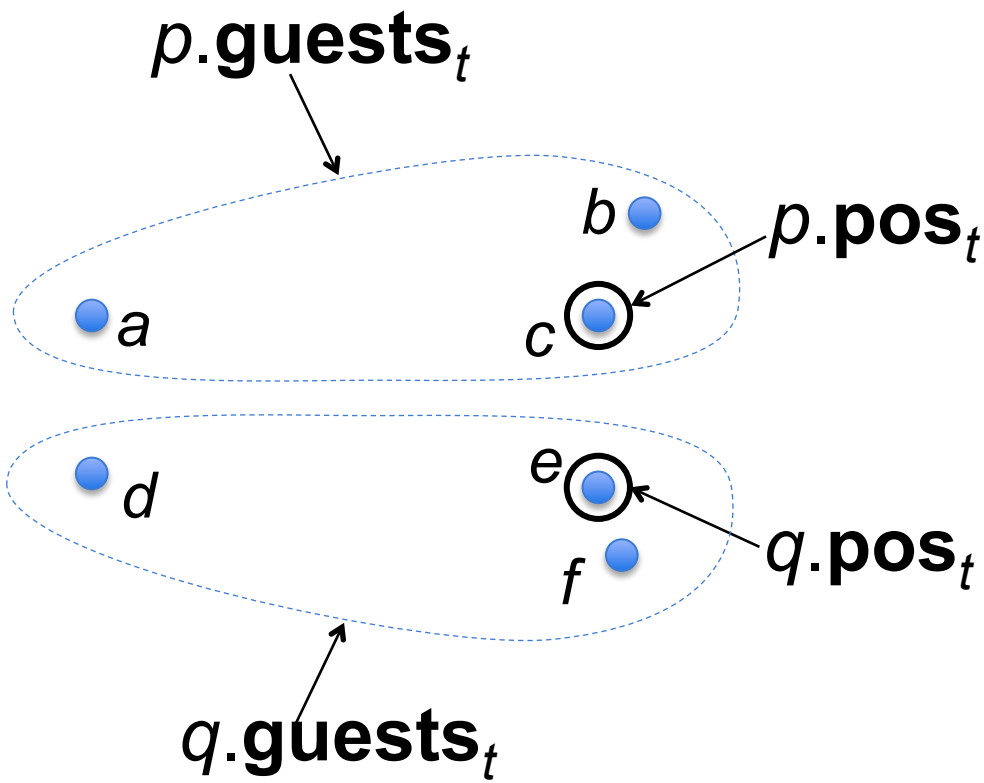


Polystyrene Protocol

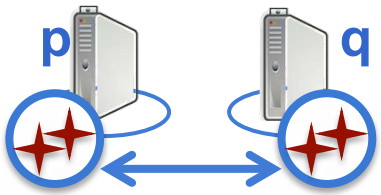


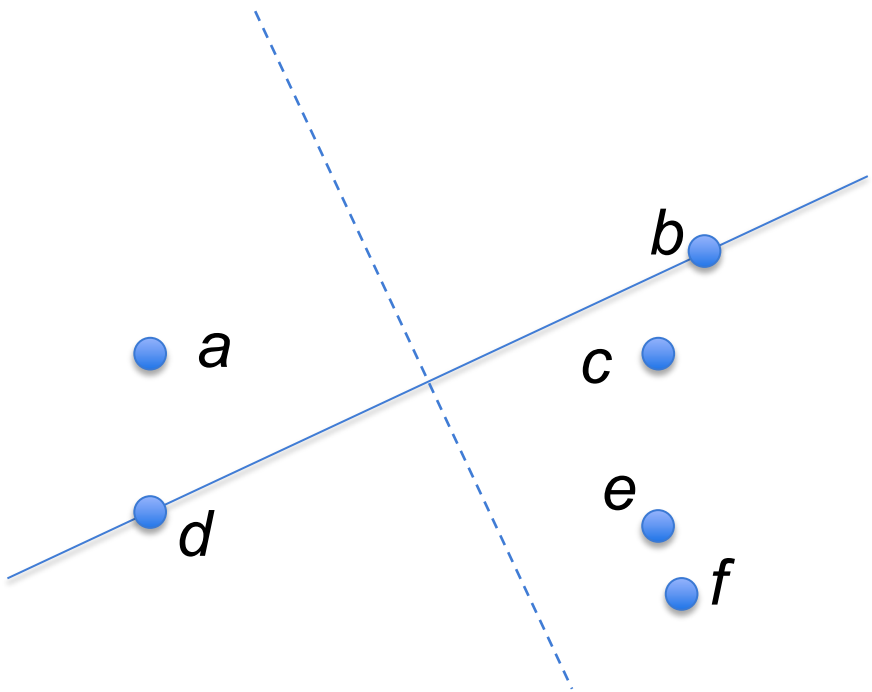
The Migration Process





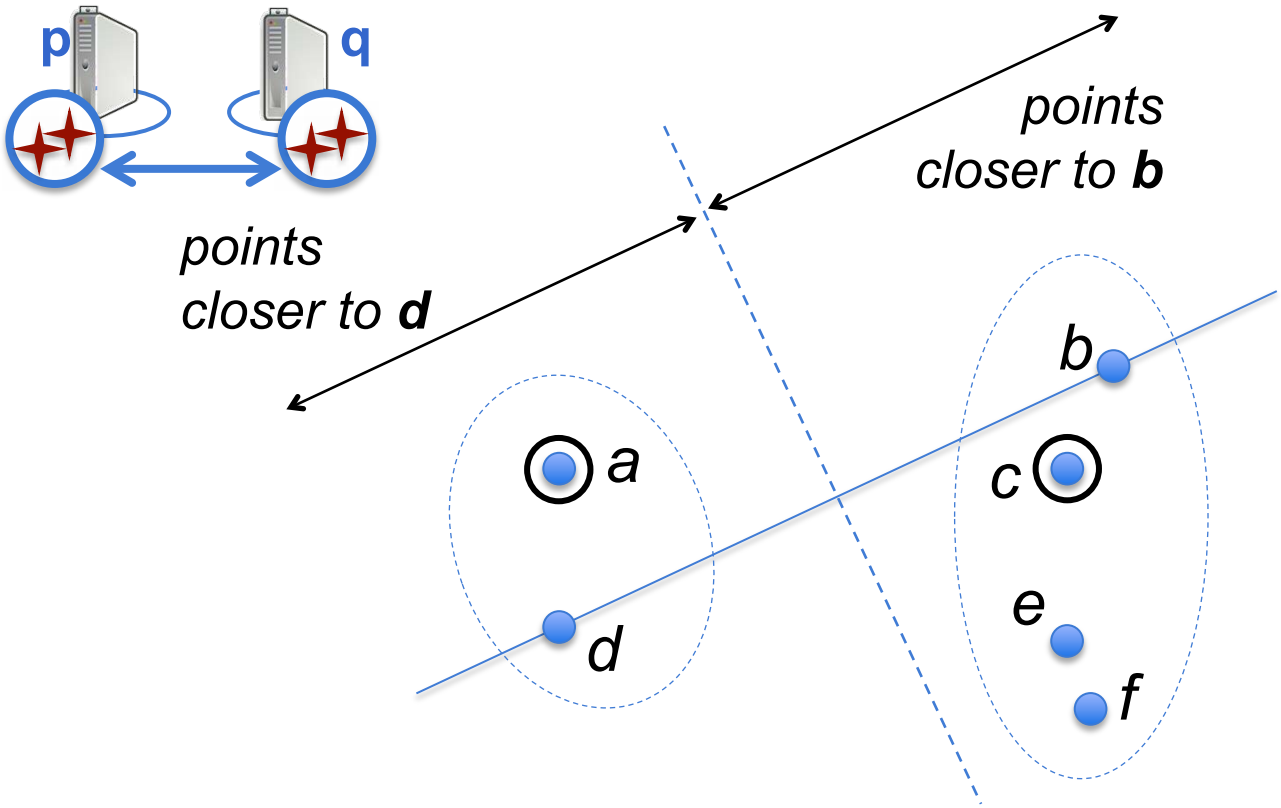
The Migration Process





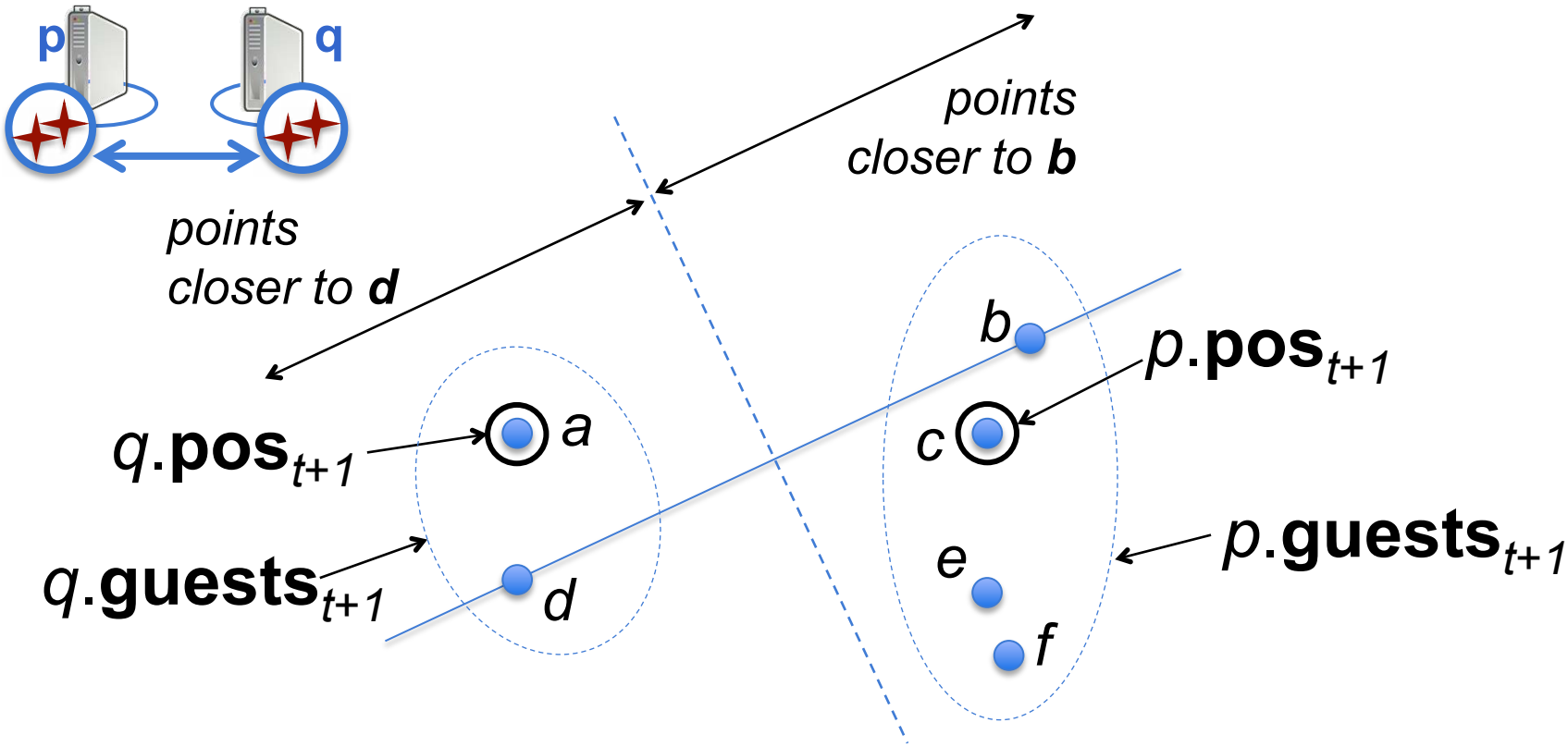
- Bi-clustering of guest points
  - ➔ Heuristics : diameter

The Migration Process



- Bi-clustering of guest points
  - ➔ Heuristics : diameter

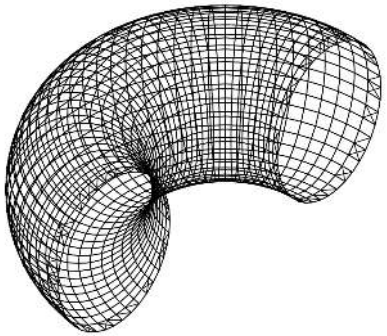
The Migration Process



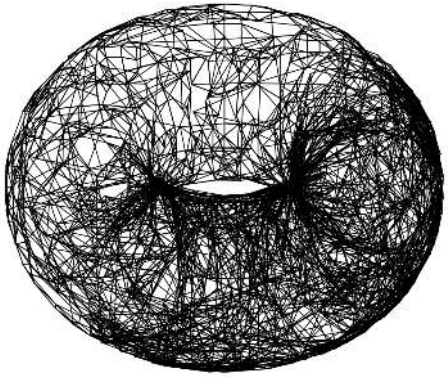
- Bi-clustering of guest points
  - ➔ Heuristics : diameter + minimum move

Evaluation

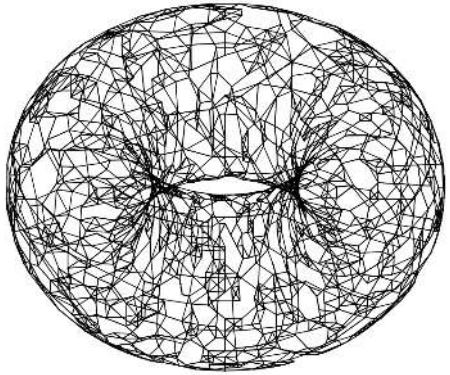
- Shape : 3D torus (replication K = 4)
- Round 20 : 50% correlated node crashes
- Round 100 : 50% node reinjection (repair)



After failure (r=20)



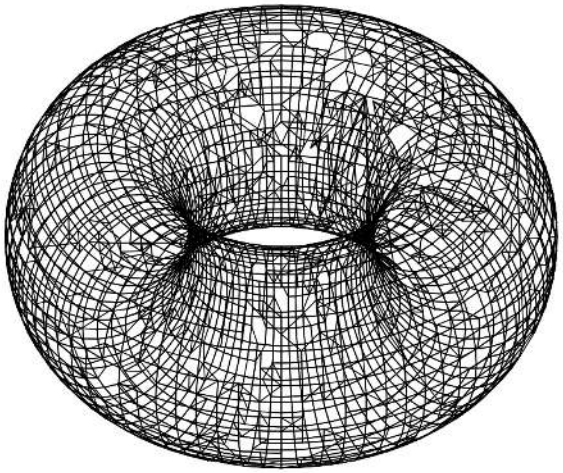
(r=22)



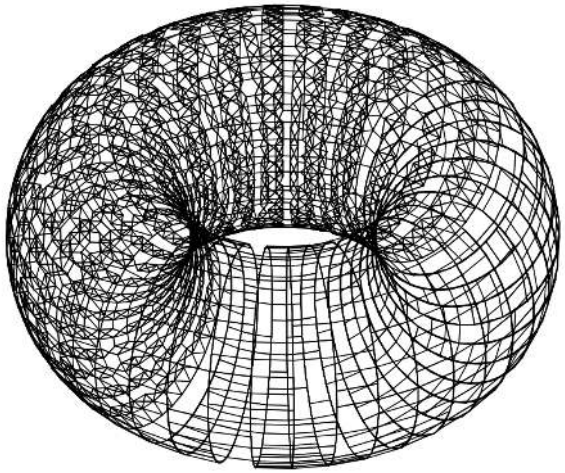
(r=28)

Polystyrene recreates shape with surviving nodes

Evaluation : Node Reinjection



Polystyrene, r = 125

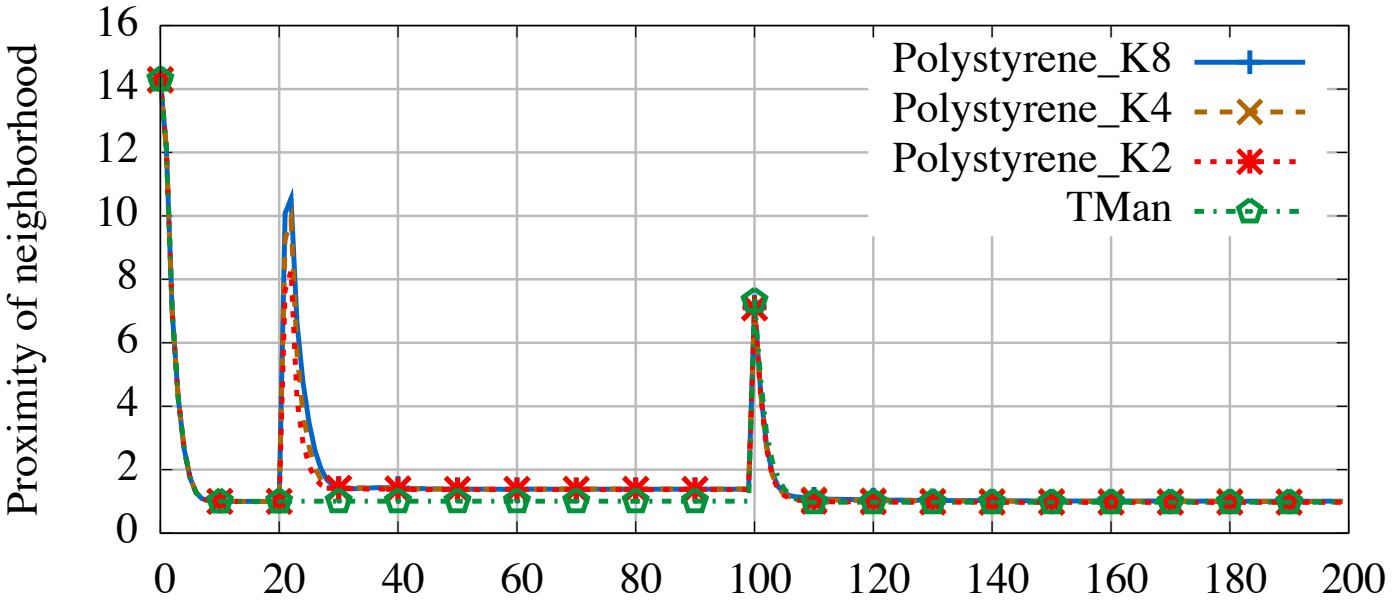


T-Man, r = 125

Polystyrene recreates uniform shape after repair

Eval: Quality of Neigborhoods

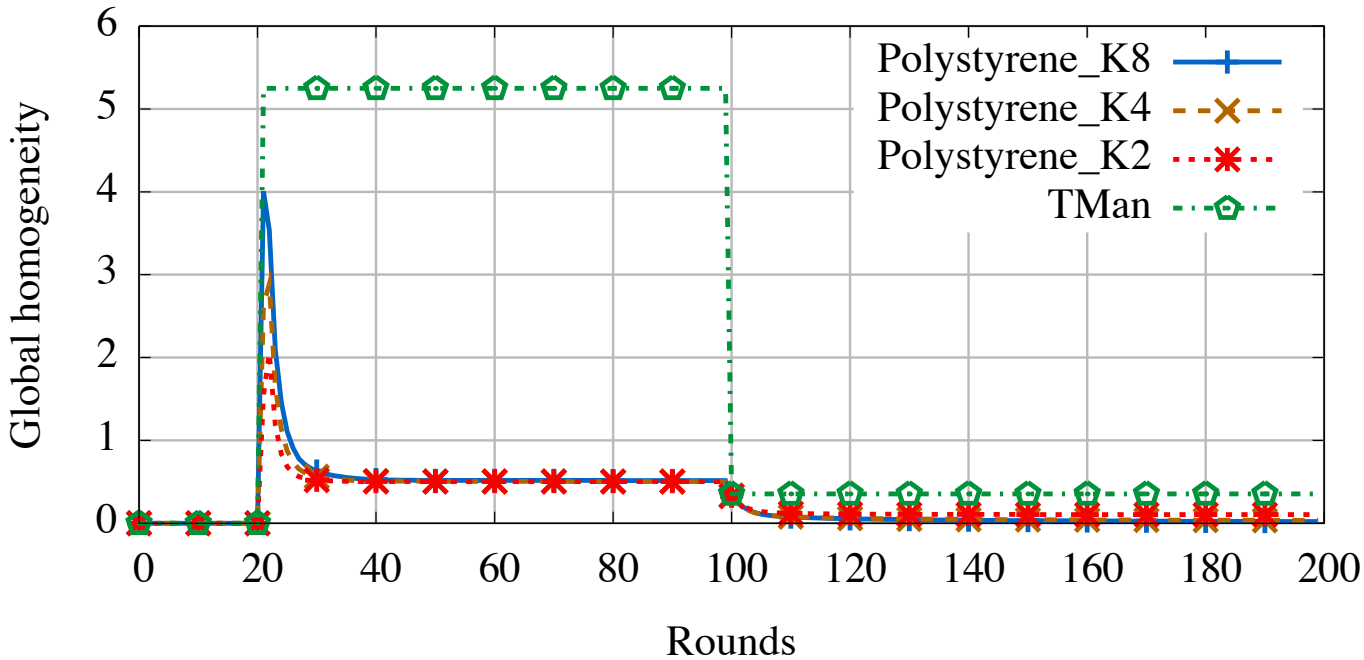
$$proximity = \mathop{\mathbb{E}}_{n_i \in \text{nodes}} \left(\mathop{\mathbb{E}}_{n_j \in n_i.\text{neighbors}} d\big(n_i.\mathbf{pos}, n_j.\mathbf{pos}\big) \right)$$

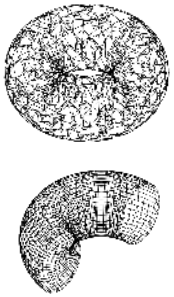


Polystyrene maintains good neighborhoods

Eval: Quality of Shape

$$homogeneity = \mathop{\mathbb{E}}_{\mathbf{x} \in \text{data-points}} \left(\min_{n \in \hat{\text{guests}}^{-1}(\mathbf{x})} \{ d(\mathbf{x}, n.\mathbf{pos}) \} \right)$$

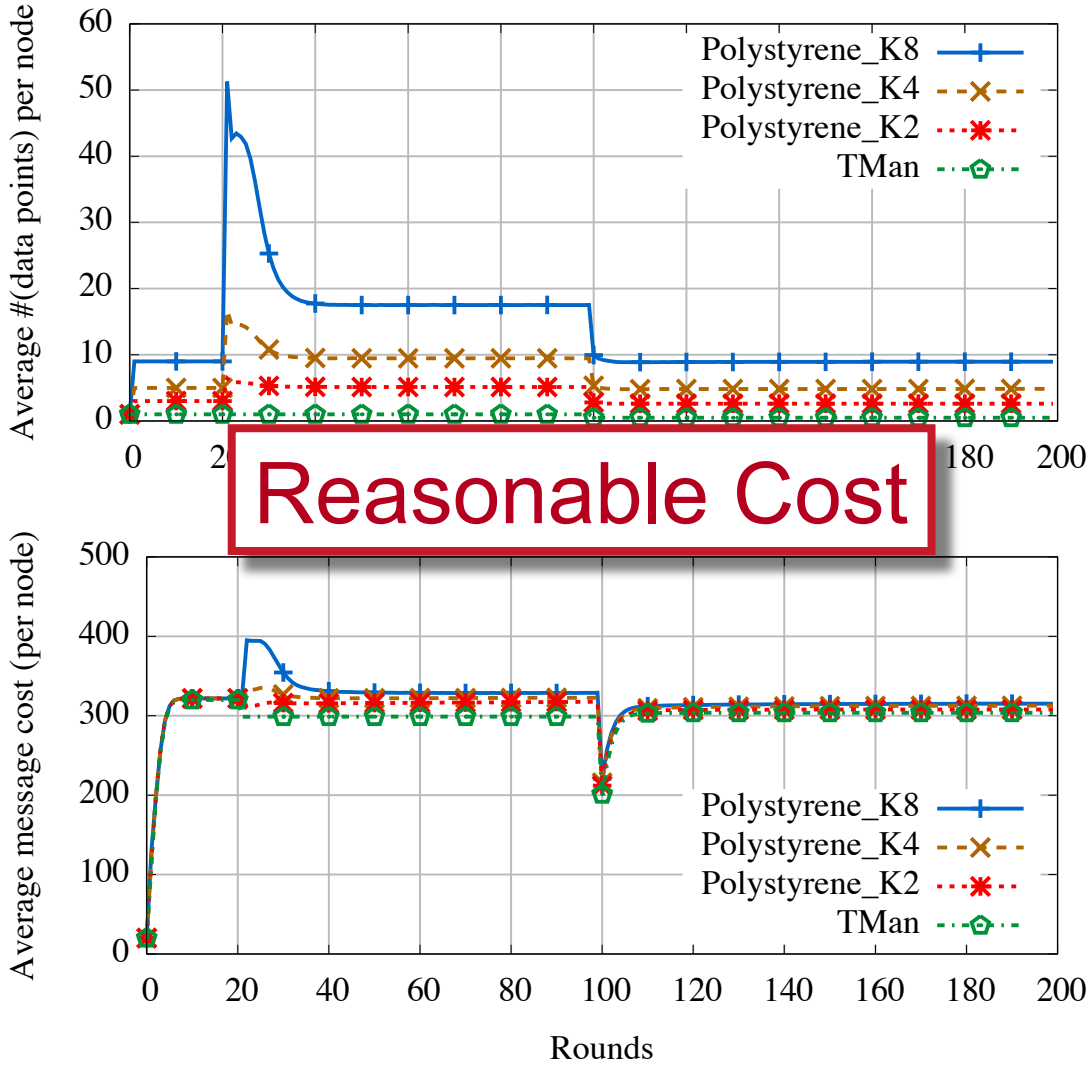




And the torus gets restored!

Eval: Overheads

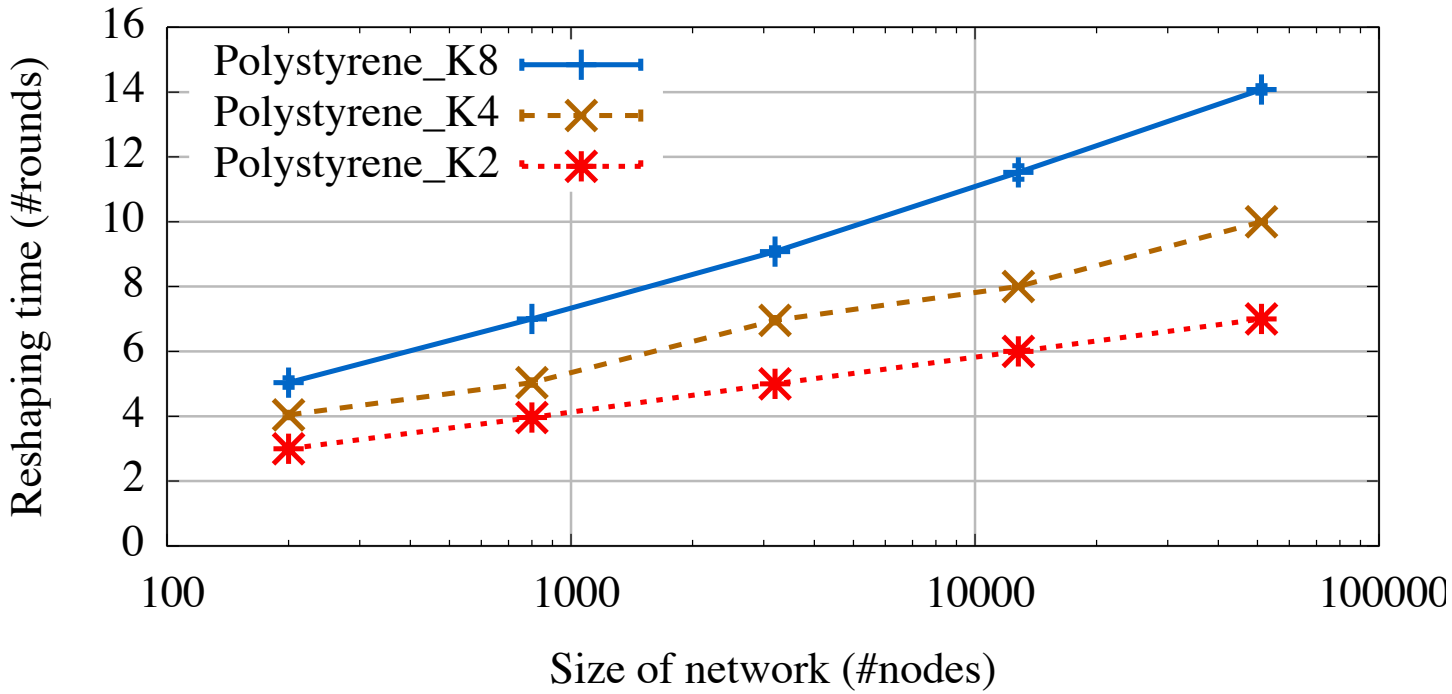
- Message + Memory



Eval: Scalability (1)

- Time (rounds) until homeogeneity less than

$$\mathcal{H}_{\mathcal{A}}^{|N|} = \frac{1}{2}\sqrt{\frac{\mathcal{A}}{|N|}}$$



Logarithmic convergence!

Outline

- Gossip Protocols
- The case of self-organizing overlays
- Self-organization++: collective survival
- **Conclusion**

Conclusion

- Growth in scale / complexity / openness / dynamicity
  - ➔ **Automation of structuring tasks** required
- Self-organisation is possible in
  - ➔ **Large scale** systems
  - ➔ Under very **harsh conditions** (catastrophic failure)
  - ➔ With **very good performance** (log(n))
- Price to pay: **No deterministic guarantees** (usually)
- Next challenge: **holistic programmability** of overlays
  - ➔ Link with "self-awareness"

Thank you
Questions?

(Some) Refs: Computability

- [FLP85] Michael J. Fischer, 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] Seth Gilbert, and Nancy Lynch. **Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.** ACM SIGACT News 33.2 (2002): 51-59.
- [HRR12] Maurice Herlihy, Sergio Rajsbaum, and Michel Raynal. **Computability in distributed computing: a tutorial.** ACM SIGACT News 43.3 (2012): 88-110.

(Some) References: Protocols

- [BHO99] Birman, K. P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., and Minsky, Y. (1999). **Bimodal multicast.** ACM Trans. Comput. Syst., 17:41–88.
- [GRB01] Gupta, I., Renesse, R. v., and Birman, K. P. (2001). **Scalable fault-tolerant aggrega- tion in large process groups.** DSN ’01, pages 433– 442
- [KMG03] Kermarrec A.-M., Massoulie L., Ganesh, A.J., **Reliable Probabilistic Communication in Large-Scale Information Dissemination Systems**, *IEEE Transactions on Parallel and Distributed Systems*, March 2003, (14:3)
- [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, pages 79–98, New York, NY, USA. Springer- Verlag New York, Inc.

(Some) References: Protocols

- [HHL06] Haas, Z. J., Halpern, J. Y., and Li, L. (2006). **Gossip-based ad hoc routing.** IEEE/ACM Trans. Netw., 14:479–491.
- [JK06] Jelasity, M. and Kermarrec, A.-M. (2006). **Ordered slicing of very large-scale overlay networks.** In Proceedings of the Sixth IEEE International Conference on Peer-to- Peer Computing, pages 117–124, Cambridge, United Kingdom. IEEE Computer Society.
- [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.
- [LGKVB06] Erwan Le Merrer, Vincent Gramoli, Anne-Marie Kermarrec, Aline C. Viana, and Marin Bertier. 2006. **Energy aware self-organizing density management in wireless sensor networks**. *1st Int. workshop on Decentralized resource sharing in mobile computing and networking* (MobiShare '06). ACM

(Some) References: Protocols

- [BFGKL10] Bertier, M., Frey, D., Guerraoui, R., Kermarrec, A.-M., Leroy, V.**: The gossple anonymous social network.** In: Proc. of the ACM/ IFIP/USENIX 11th Int. Conf. on Middleware. Middleware'10, pp. 191–211 (2010)

(Some) Refs: Frameworks

- [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
- [LTB08] Lin, S., Taiani, F., and Blair, G. S. (2008). **Facilitating gossip programming with the gossipkit framework**, DAIS’08, pages 238–252
- [KS07] Kermarrec, A.-M. and van Steen, M. (2007). **Gossiping in distributed systems.** SIGOPS Oper. Syst. Rev., 41:2–7
- [EFL07] Eugster, P., Felber, P., and Le Fessant, F. (2007). **The ”art” of programming gossip-based systems.** SIGOPS Oper. Syst. Rev., 41:37–42.

Ideal software artefact

- ➔ structured, predictable, open, evolvable

A Distributed System Today …

Standards

JSON
JavaScript Object Notation

External developers

External services

facebook

twitter

foursquare™

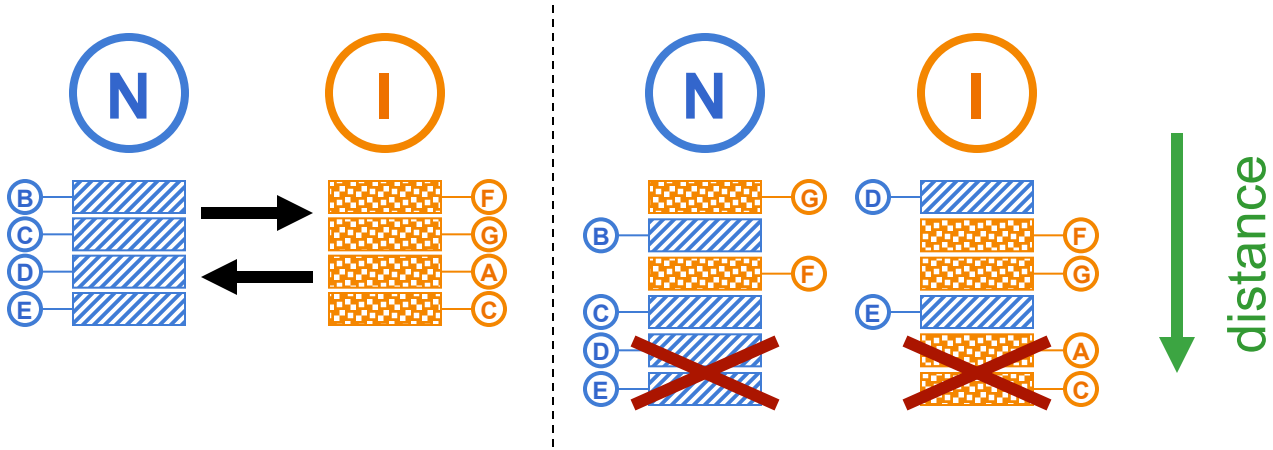
Geosocial app, est. 2009

45M Users

Middleware

T-Man: Swap Mechanism

- To speed convergence each node **n**
  - ➔ picks one node **i** in clustering neighbours
  - ➔ sends own list of neighbours, receives that of **i**
  - ➔ **i** and **n** merge, **sort**, keep k closest neighbours



(1) peer list exchange (2) merge and truncation

Eval: Scalability (2)

- Strong influence of splitting mechanism