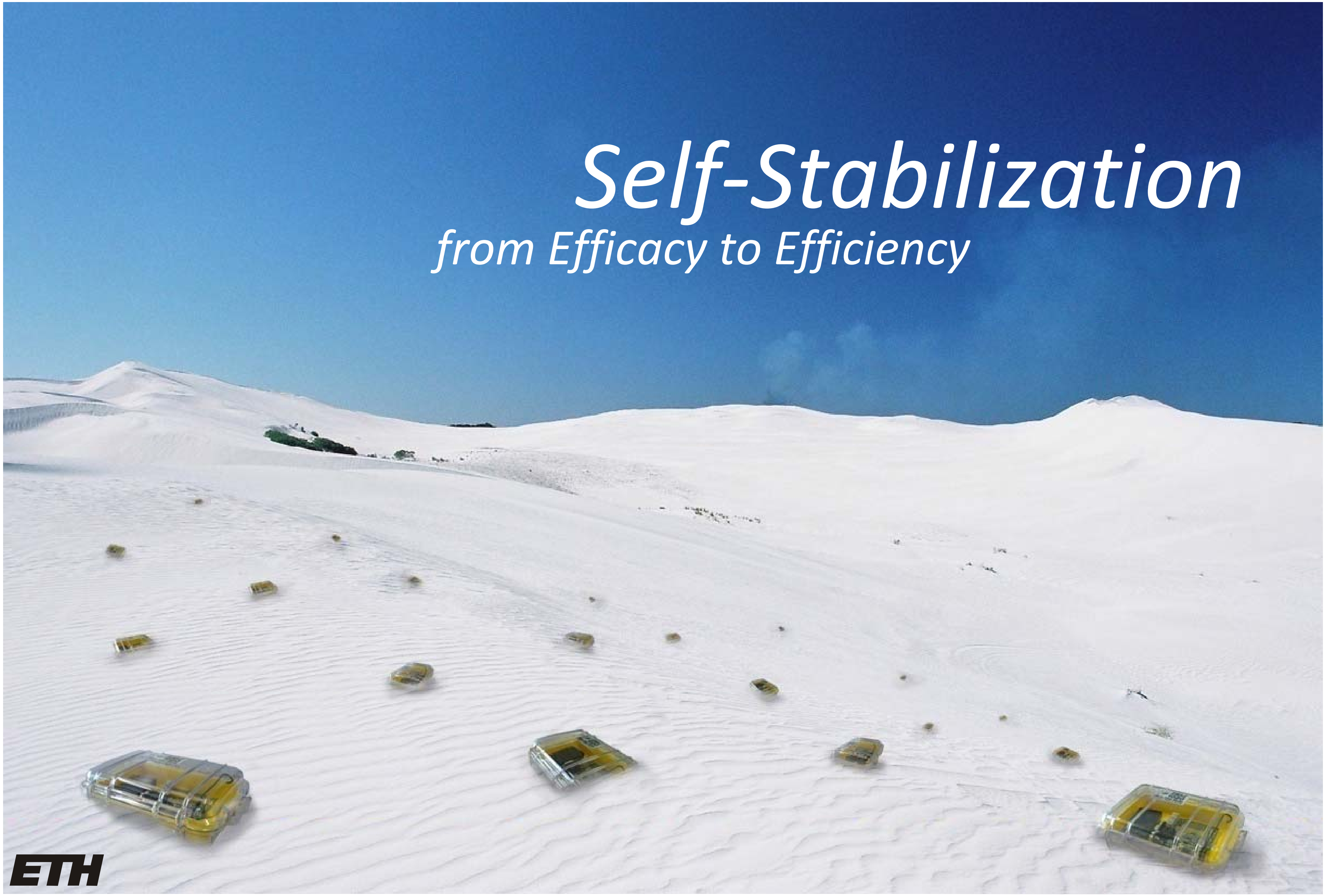
Self-Stabilization

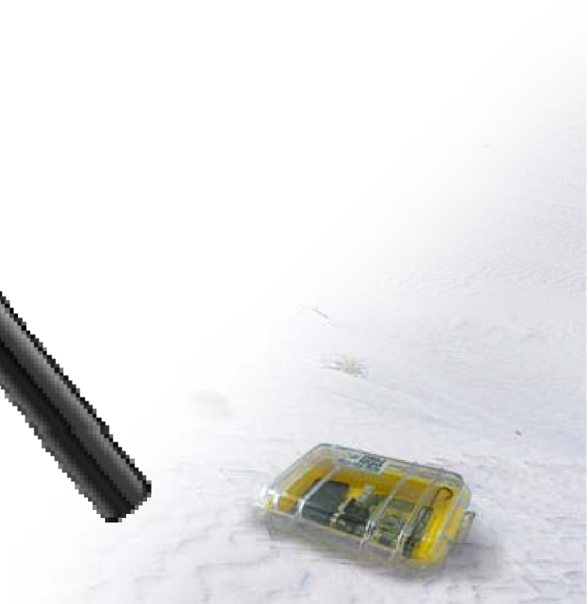
from Efficacy to Efficiency



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

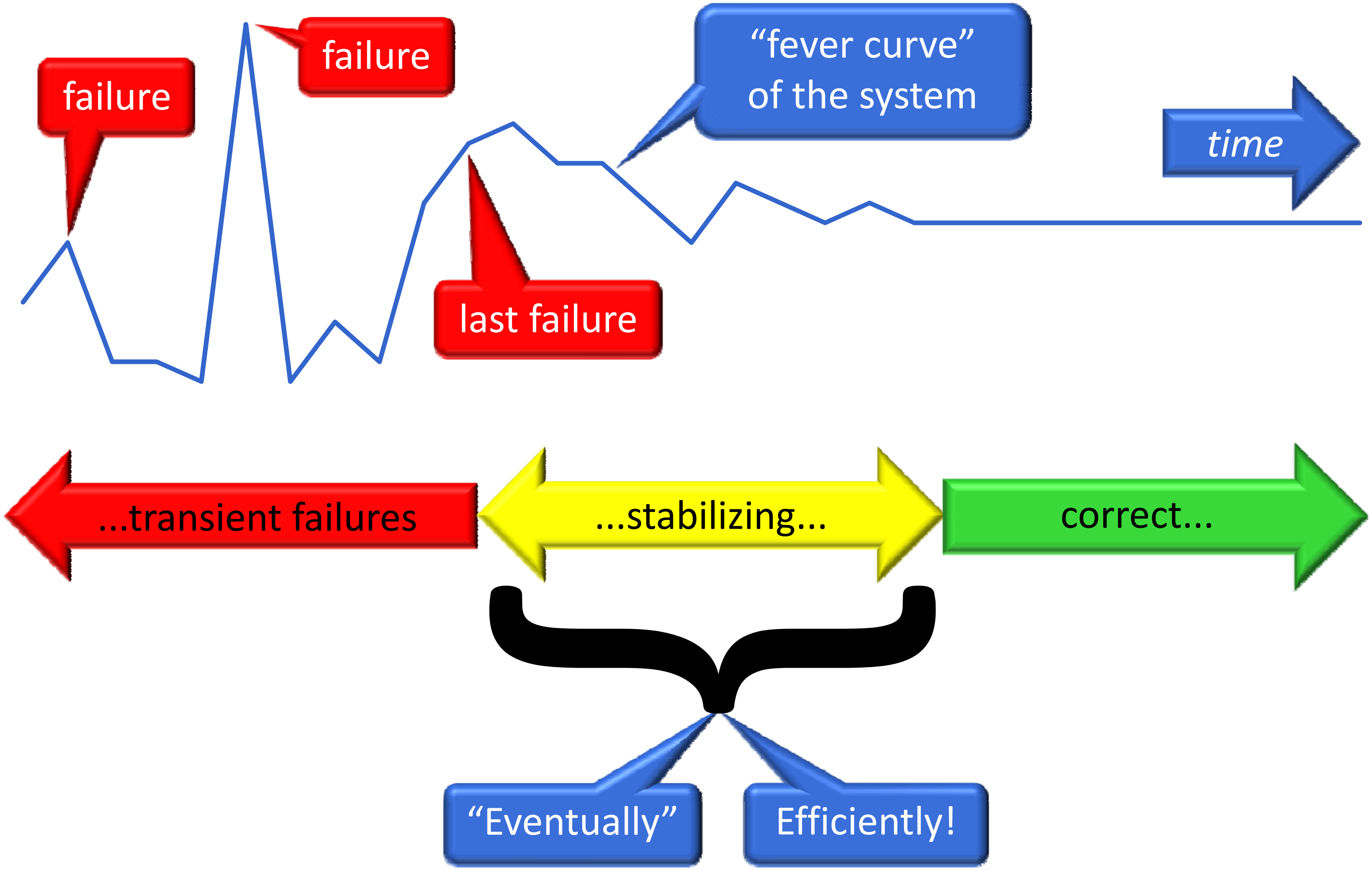
Mea Culpa!

I would like to apologize in advance for everything you may find obvious *or* offensive!



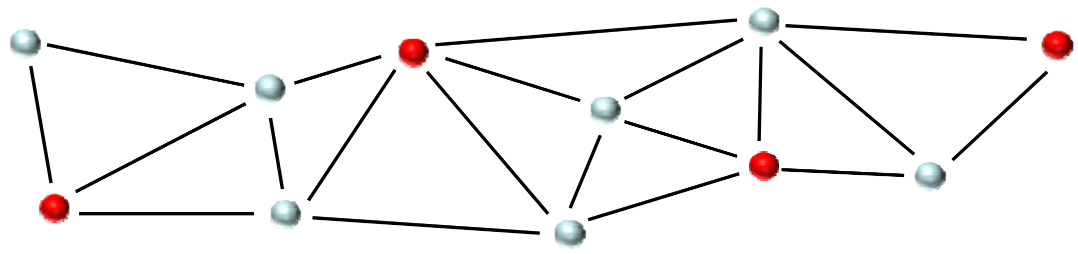
- Frog's eye view, frog is outside(r)!
- Frog may be pretty ignorant,
  but doesn't stop frog from being curious,
  (or even cocky)

Self-Stabilization: Frog's Eye View



Example: Maximal Independent Set (MIS)

- Input: Given a graph (network), nodes with unique IDs.
- Output: Find a Maximal Independent Set (MIS)
  - a non-extendable set of pair-wise non-adjacent nodes



- A self-stabilizing algorithm:

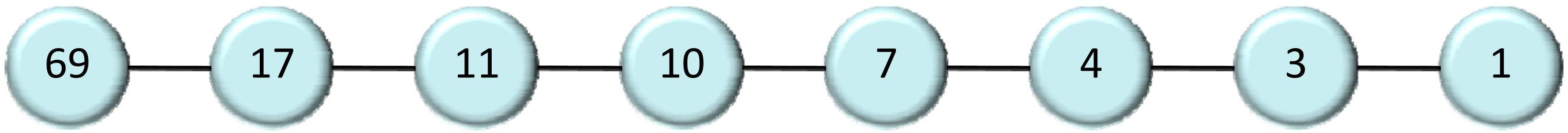
`IF no higher ID neighbor is in MIS → join MIS`

`IF higher ID neighbor is in MIS → do not join MIS`

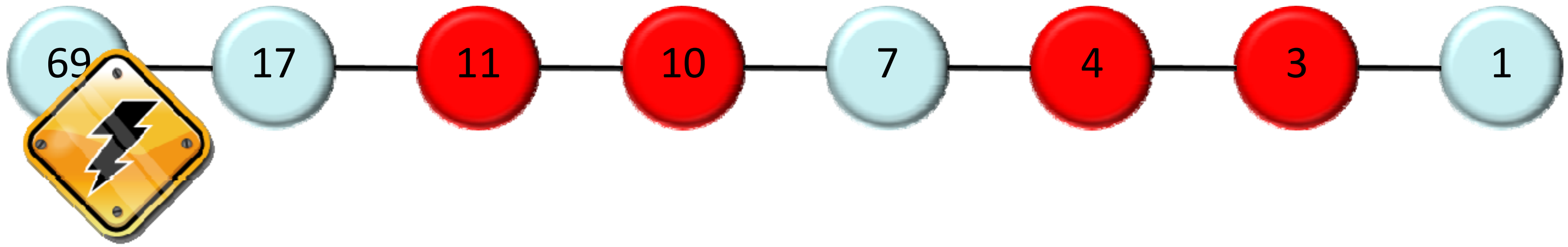
- Can be implemented by constantly sending (ID, in MIS or not in MIS)
- This algorithm has all the beauty of a typical self-stabilizing algorithm: It is simple, and it will eventually stabilize!

Example

```
IF no higher ID neighbor is in MIS → join MIS
IF higher ID neighbor is in MIS → do not join MIS
```



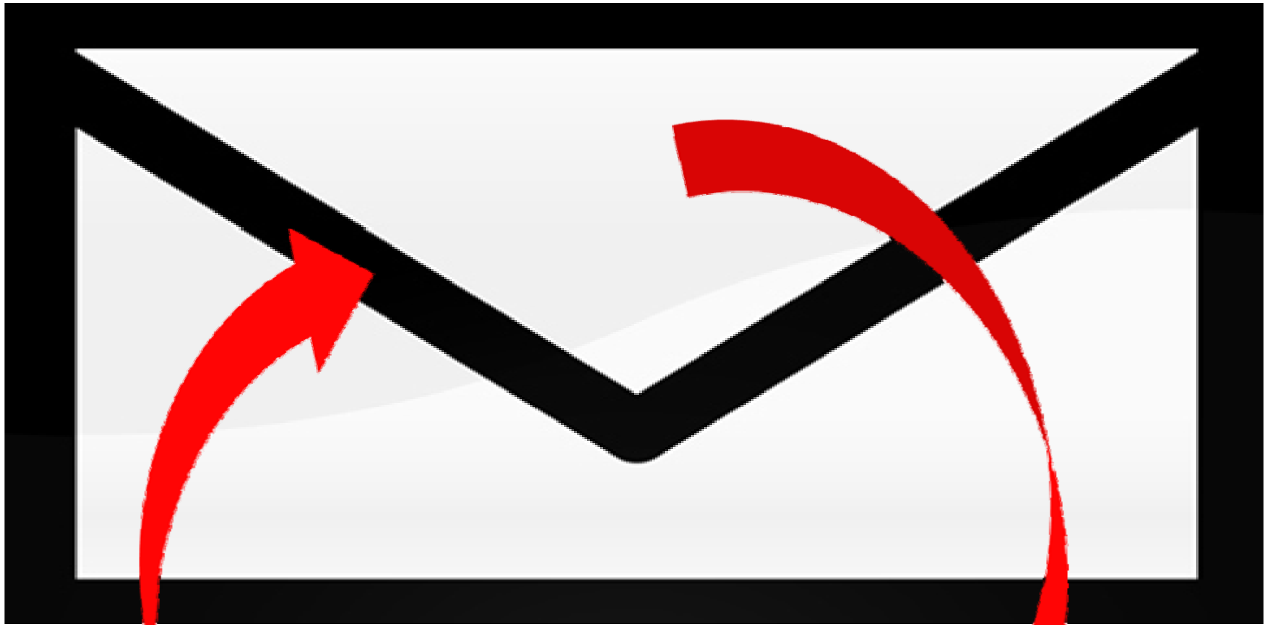
- What about transient failures?



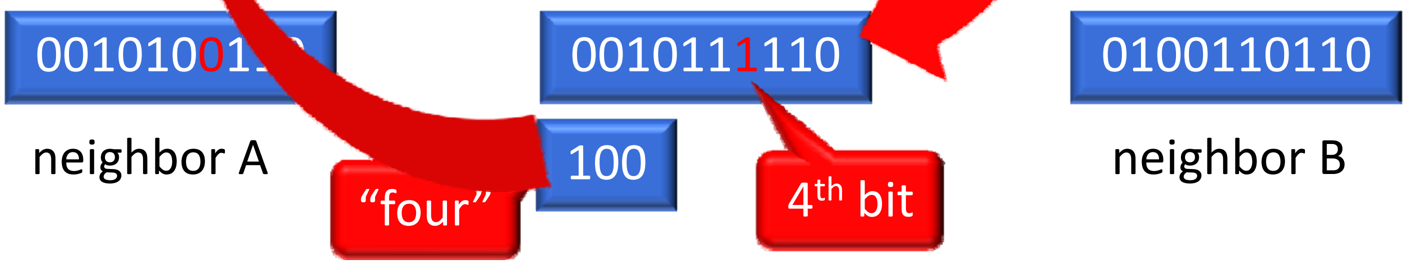
- Proof by animation: Stabilization time is linear in the diameter of the network
  - We need an algorithm that does not have linear causality chain („butterfly effect")

An Efficient Algorithm

- Nodes constantly send the following message



- Blue box: At which position does your „parent“ box differ from the neighbor with the lowest value in the same parent box? (Cole/Vishkin)



An Efficient Algorithm (2)

- In the first box (left-right, then top-bottom) where your value is smaller than that of any of your neighbors, you declare to be in the MIS
- If any neighbor declares to be in the MIS, you declare not to be in the MIS
- Algorithm is much more difficult; I cheated extensively…

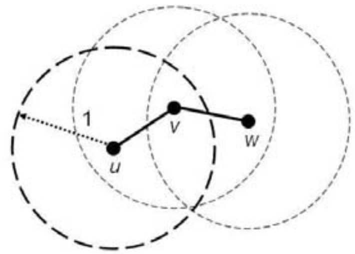
It can be shown…

- „Eventually“ a MIS will emerge, not depending on graph or node IDs
- In fact, for an important class of graphs, so-called bounded-independence graphs (well-suited for practical networks), the message will only have O(1) columns, in other words

> Message size is O(log n)
>
> Stabilization time is O(log*n)

- Stabilization Proof: As soon as there are no more transient failures, each node will recompute the correct message in O(log*n) time.
- Results basically taken from [Schneider et al., 2008]

Connectivity Models for Wireless Networks: Overview



General Graph

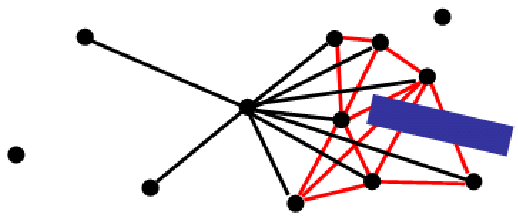
UDG

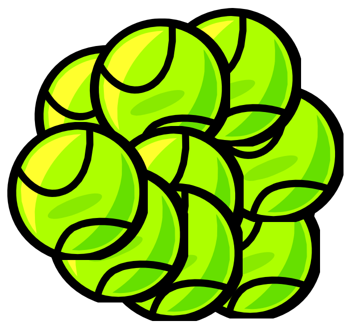
too pessimistic ⟷ **too optimistic**

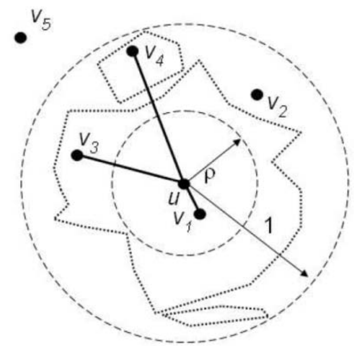
Bounded Independence

Unit Ball Graph

Quasi UDG



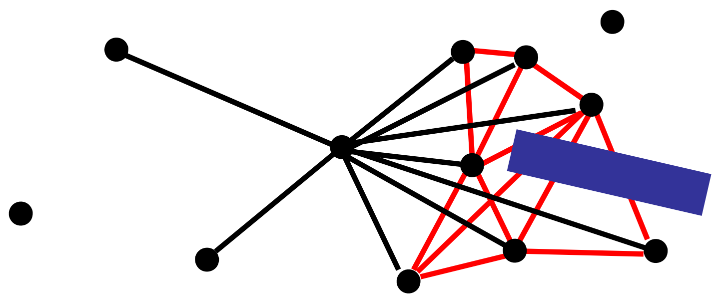




Bounded Independence Graph (BIG)

- Size of any independent set grows polynomially with hop distance *r*
- e.g., $f(r) = O(r^2)$ or $O(r^3)$
- A set S of nodes is an independent set, if there is no edge between any two nodes in S.

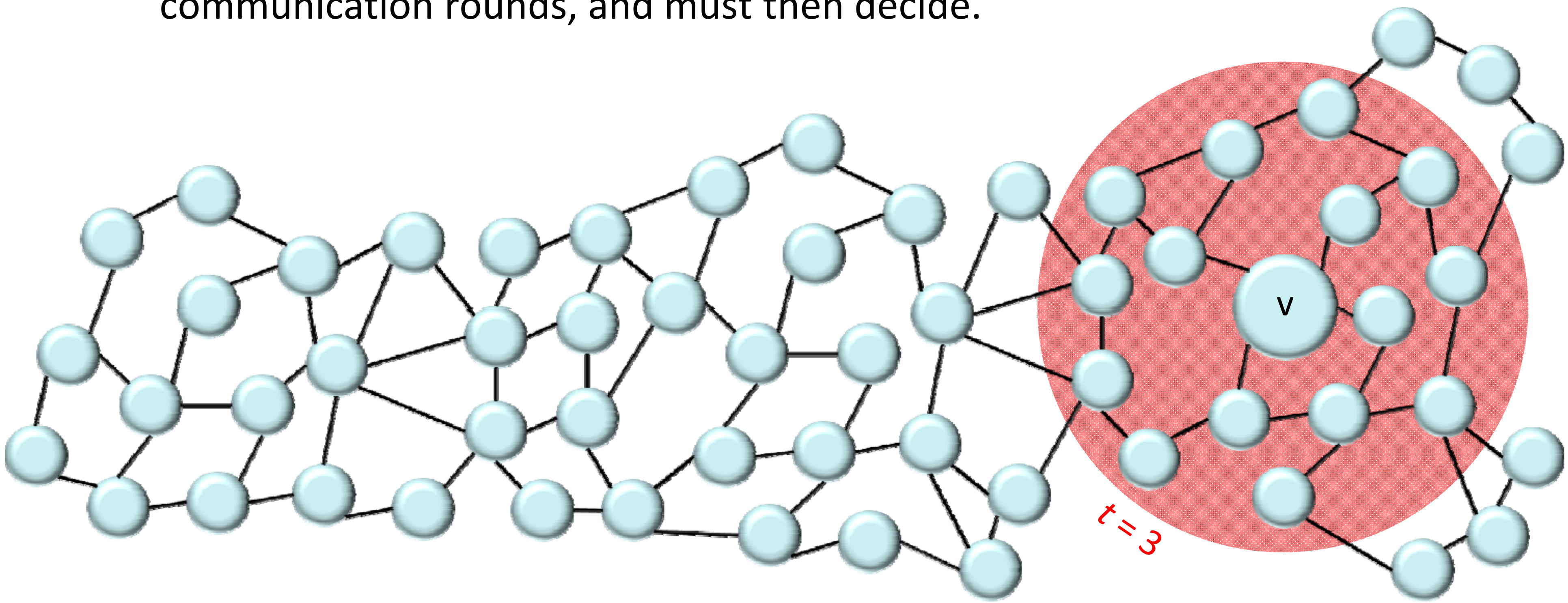
- BIG model also known as bounded-growth
  - Unfortunately, the term bounded-growth is ambiguous



Local Algorithm

- Given a graph, each node must determine its decision (e.g., in MIS or not in MIS) as a function of the information available within radius *t* of the node.

- Alternatively: Given a synchronous algorithm, no failures whatsoever, each node can exchange a message with all neighbors, for *t* communication rounds, and must then decide.



Self-Stabilization vs. Local Algorithms

Self-Stabilization

[Dijkstra, 1974]

Trans. Byz. Faults

Long-Lived

Asynchronous

Local Algorithms

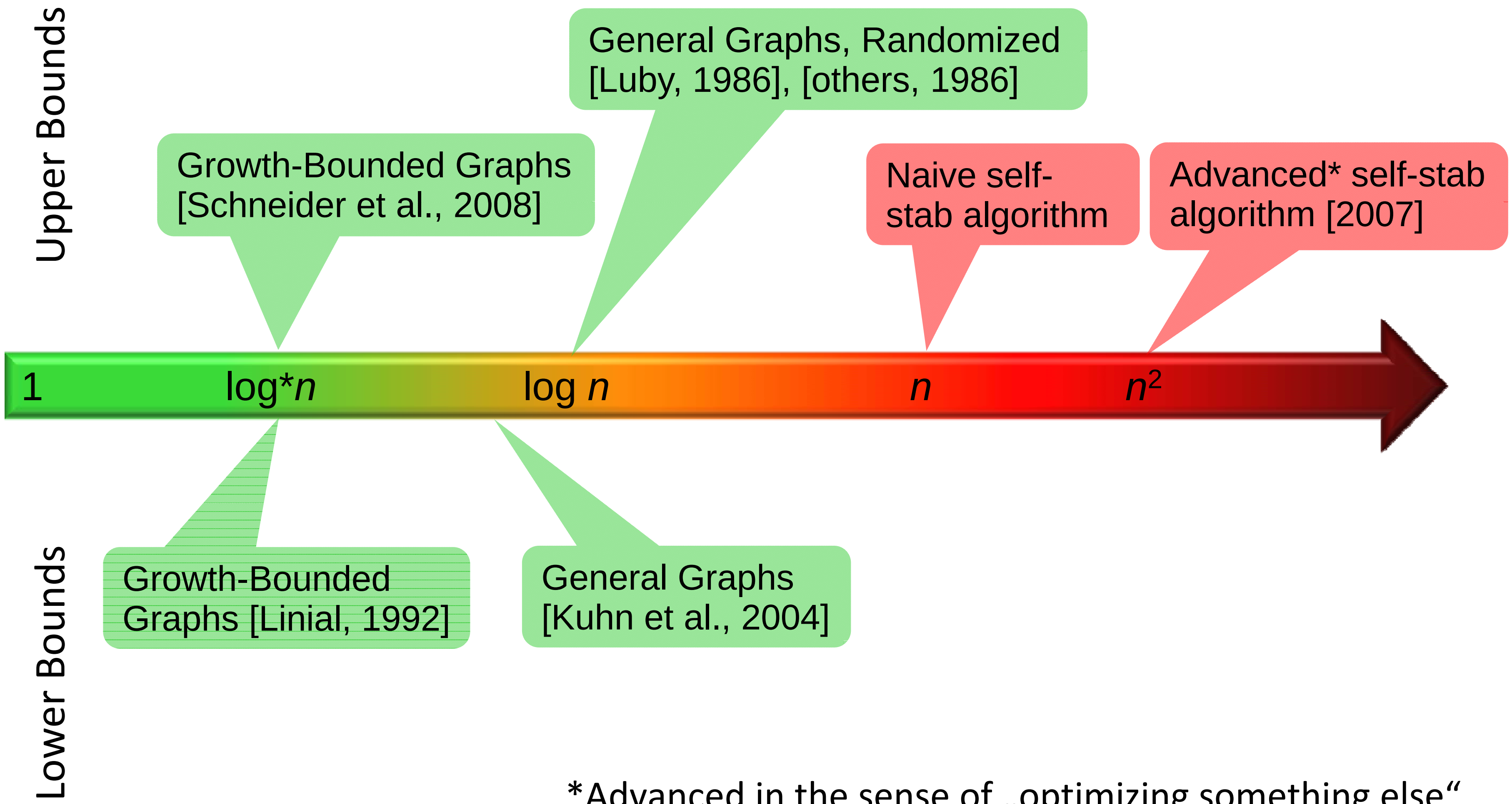
[1980s]

No Faults

One-Shot

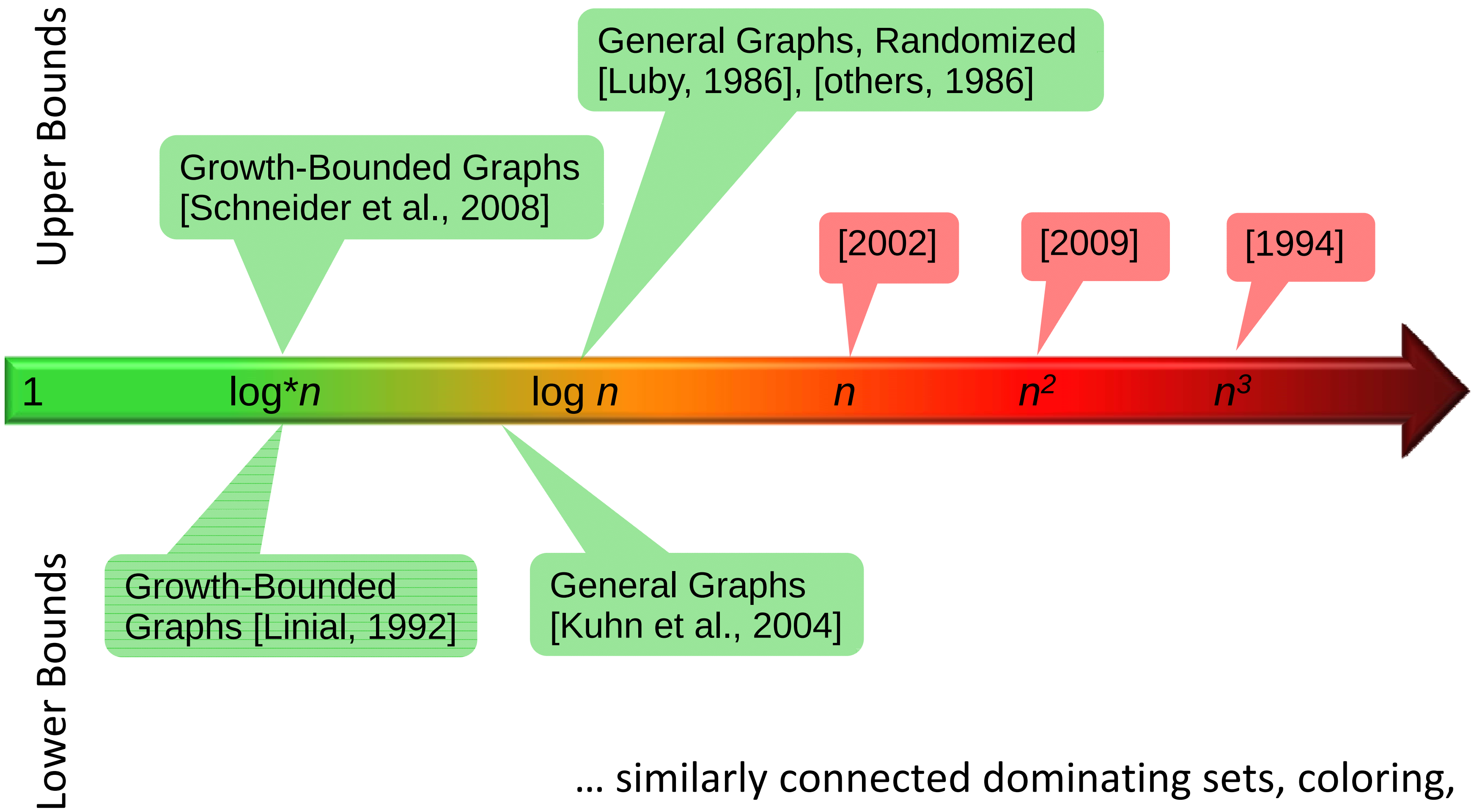
Synchronous

Results: MIS, Local Algorithms vs. Self-Stabilization



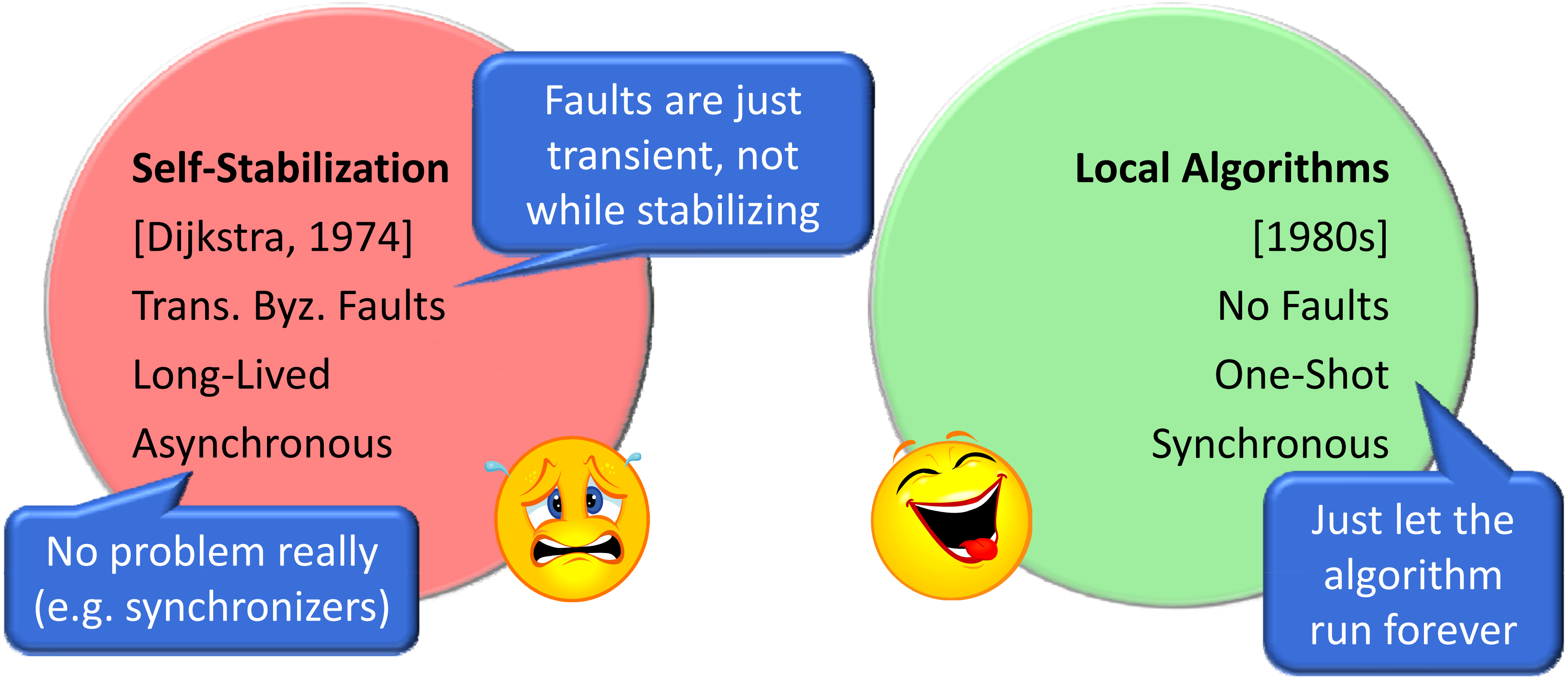
*Advanced in the sense of „optimizing something else"

Results: Maximal Matching, Local Algorithms vs. Self-Stabilization



… similarly connected dominating sets, coloring, covering, packing, max-min LPs, etc.

Self-Stabilization vs. Local Algorithms



Theorem: Self-Stabilization = Local Algorithms

In other words: Self-Stabilization „Re-Invented“ by Local Algorithms

Self-Stabilization = Local Algorithms

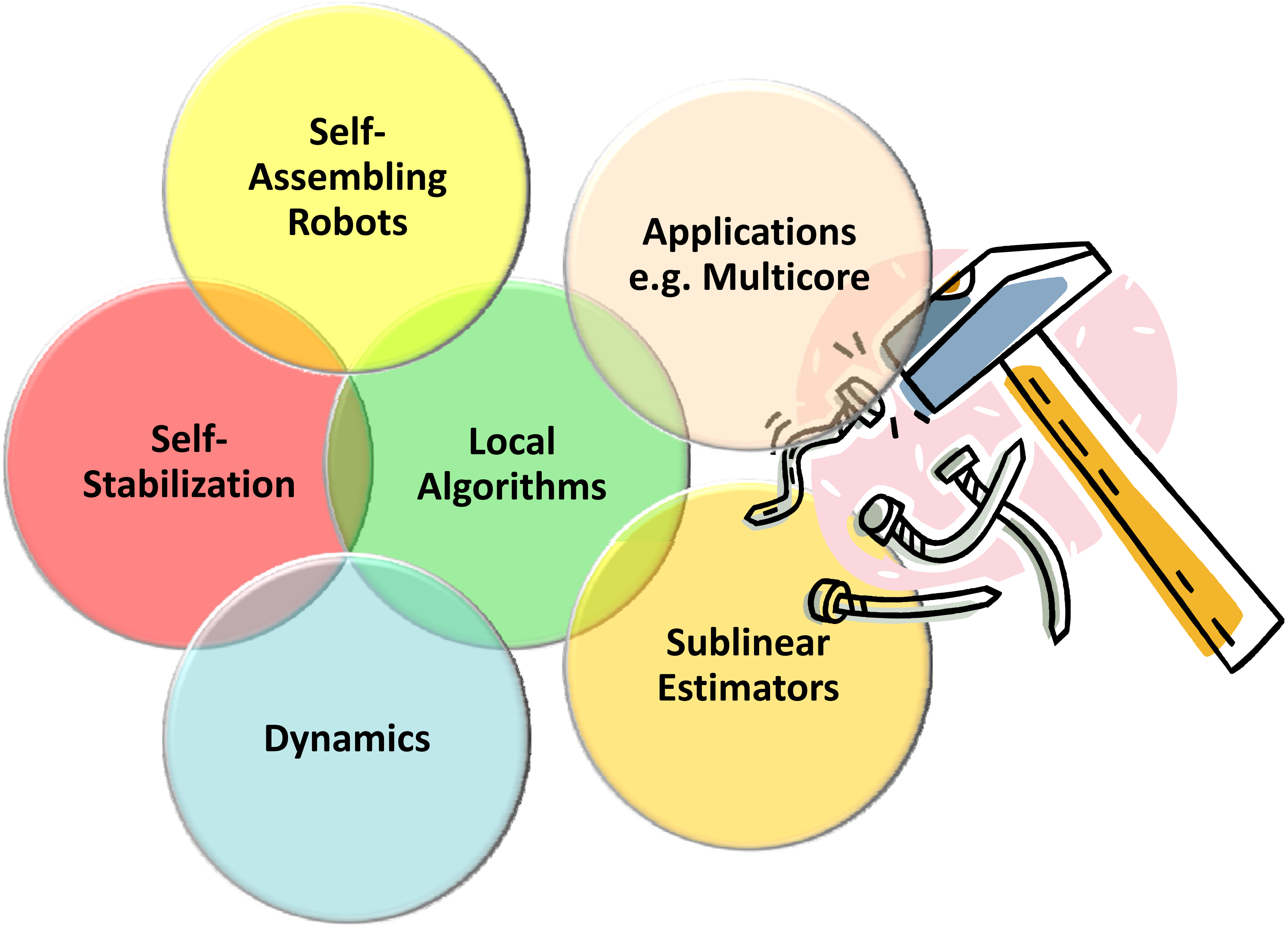
← This direction is known for a very long time, and considered to be a folk theorem, e.g. [Afek, Kutten & Yung 1990], [Awerbuch & Varghese, 1991].

The general idea is to let nodes simulate the local algorithm forever. Nodes do notice a transient failure because the information of a neighbor does not correspond to the local simulation („local checking“); nodes then simply (and automatically) adapt their solution.

→ This direction is even simpler. Lower bounds for local algorithms also hold in the self-stabilization model because the self-stabilization model is „harder“.

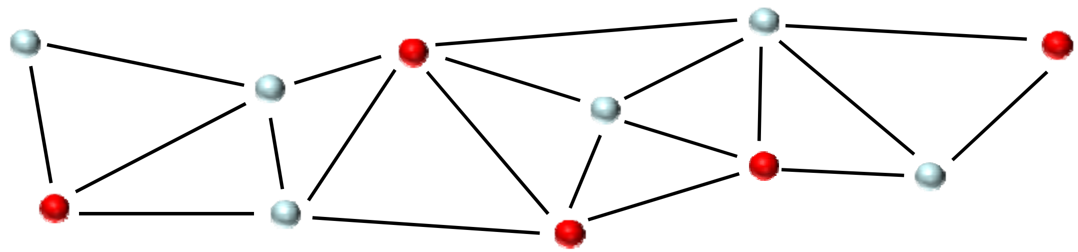
Theorem (just a bit more detail): Every local algorithm with quality guarantee *q* and time complexity *t* can be turned into a self-stabilizing algorithm with quality guarantee *q*, stabilizing efficiently in time *t*; transient faults will at most affect nodes in radius *t*. The very same holds for lower bounds. [Details in SSS 2009 paper]

Relations!

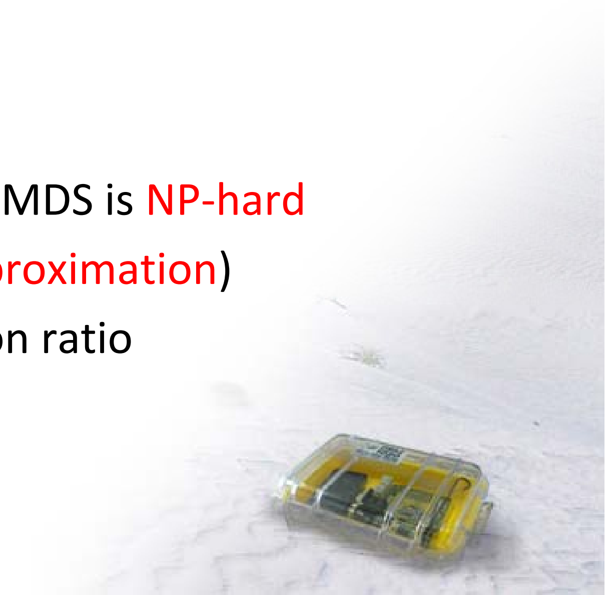

Self-
Assembling
Robots
Applications
e.g. Multicore
Self-
Stabilization
Local
Algorithms
Sublinear
Estimators
Dynamics

Lower Bound Example: Minimum Dominating Set (MDS)

- Input: Given a graph (network), nodes with unique IDs.
- Output: Find a Minimum Dominating Set (MDS)
  - Set of nodes, each node is either in the set itself, or has neighbor in set

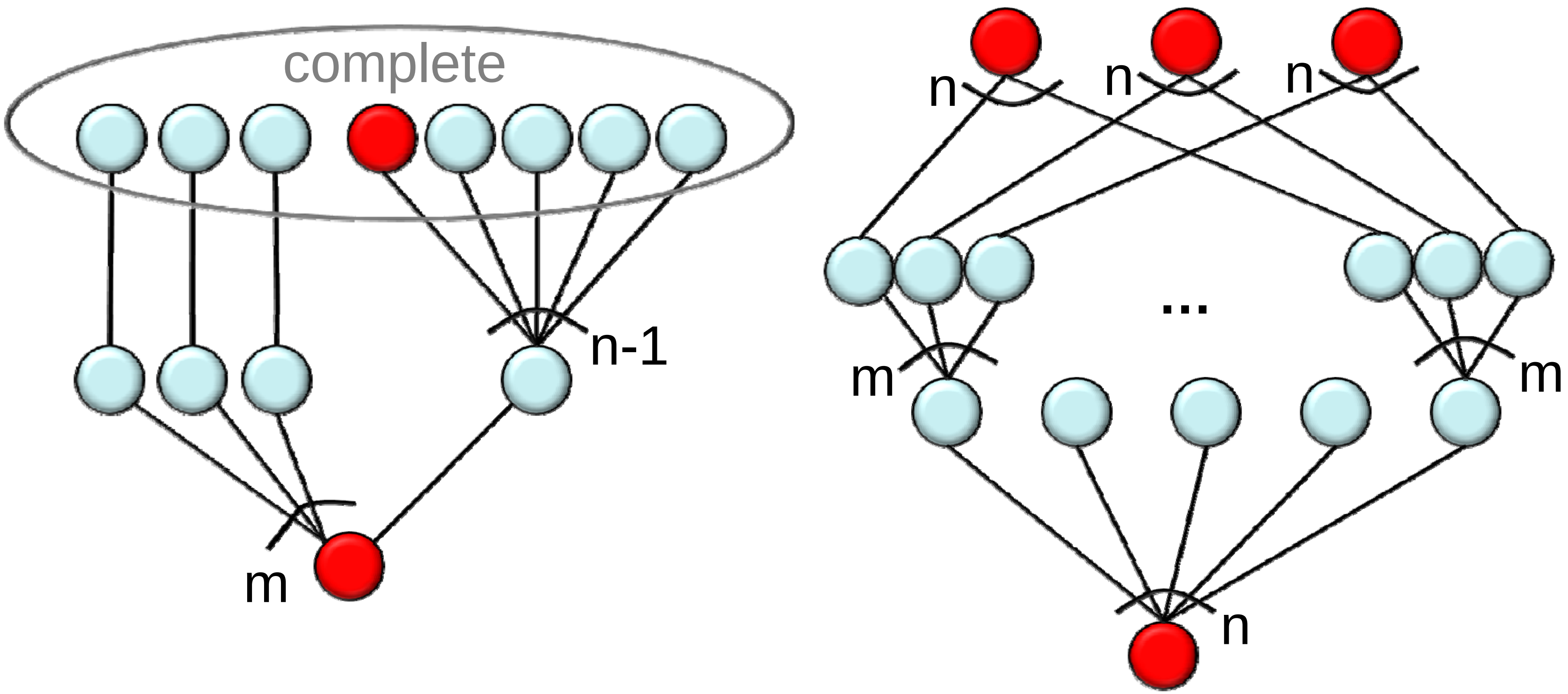


- Differences between MIS and MDS
  - Central (non-local) algorithms: MIS is trivial, whereas MDS is NP-hard
  - Instead: Find an MDS that is “close” to minimum (approximation)
  - Trade-off between time complexity and approximation ratio



Lower Bound for MDS: Intuition

- Two graphs (m << n). Optimal dominating sets are marked red.

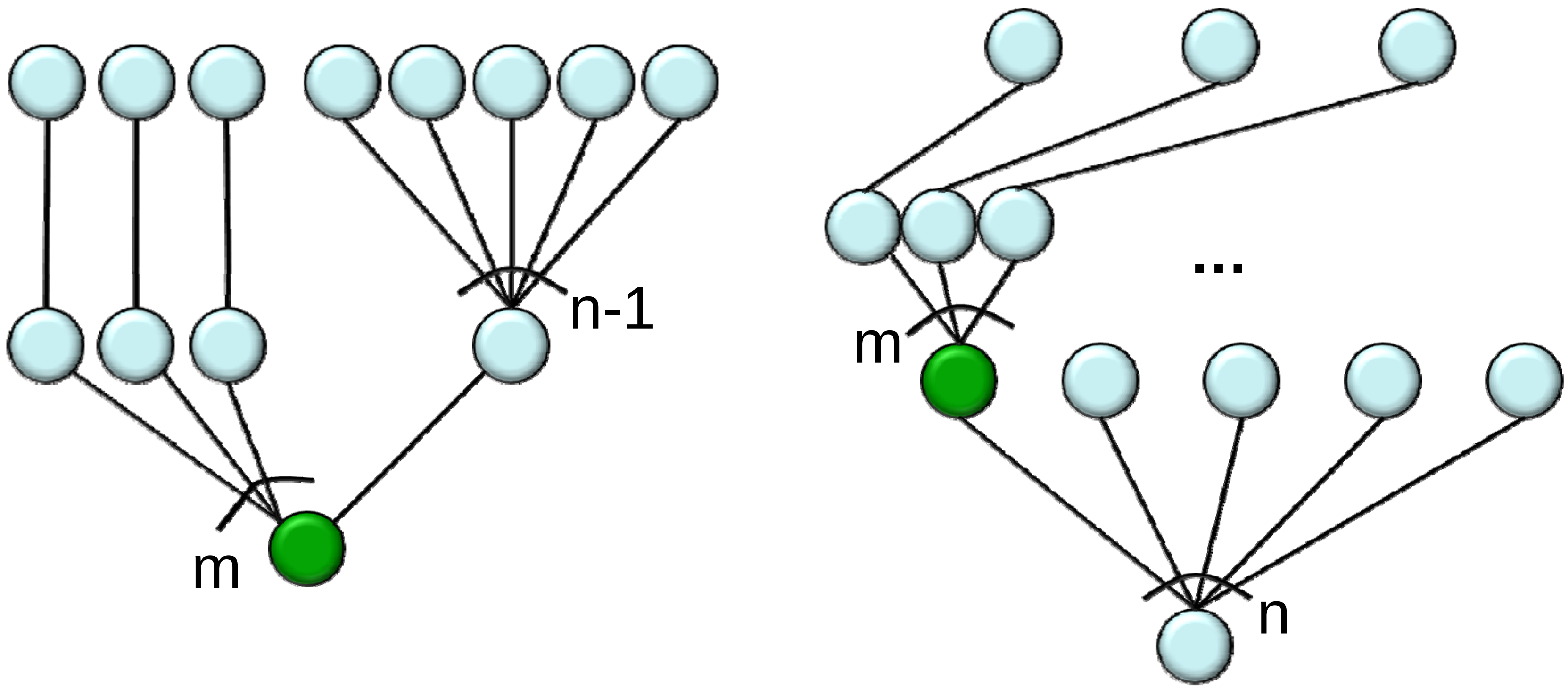


$|DS_{OPT}| = 2.$

$|DS_{OPT}| = m+1.$

Lower Bound for MDS: Intuition (2)

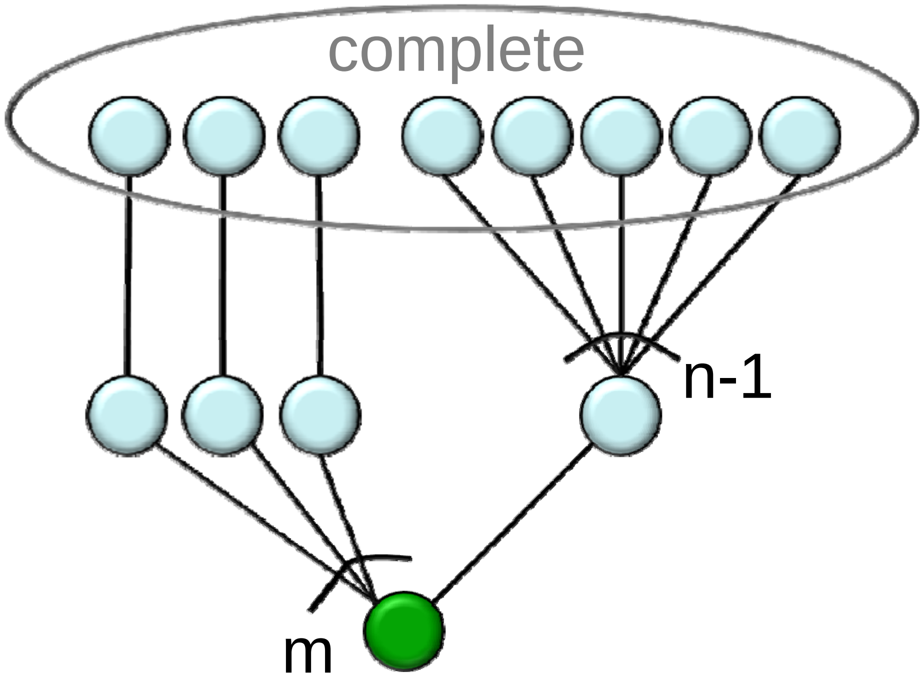
- In local algorithms, nodes must decide only using local knowledge.
- In the example green nodes see exactly the same neighborhood.



- So these green nodes must decide the same way!

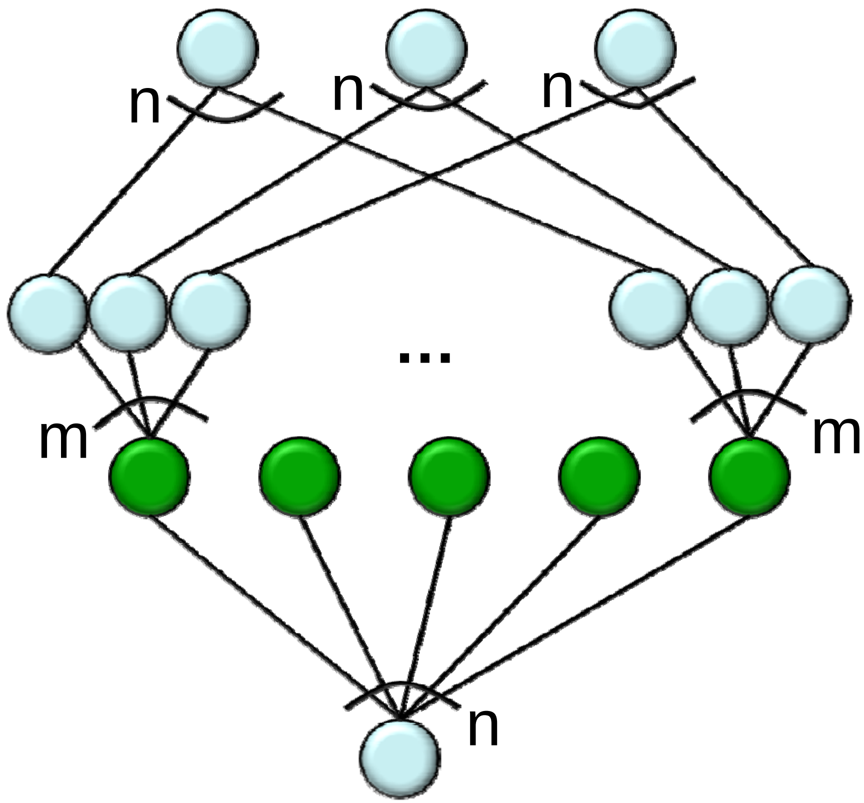
Lower Bound for MDS: Intuition (3)

- But however they decide, one way will be devastating (with $n = m^2$)!



$|DS_{OPT}| = 2.$

$|DS_{OPT\ \text{without green}}| \geq m.$

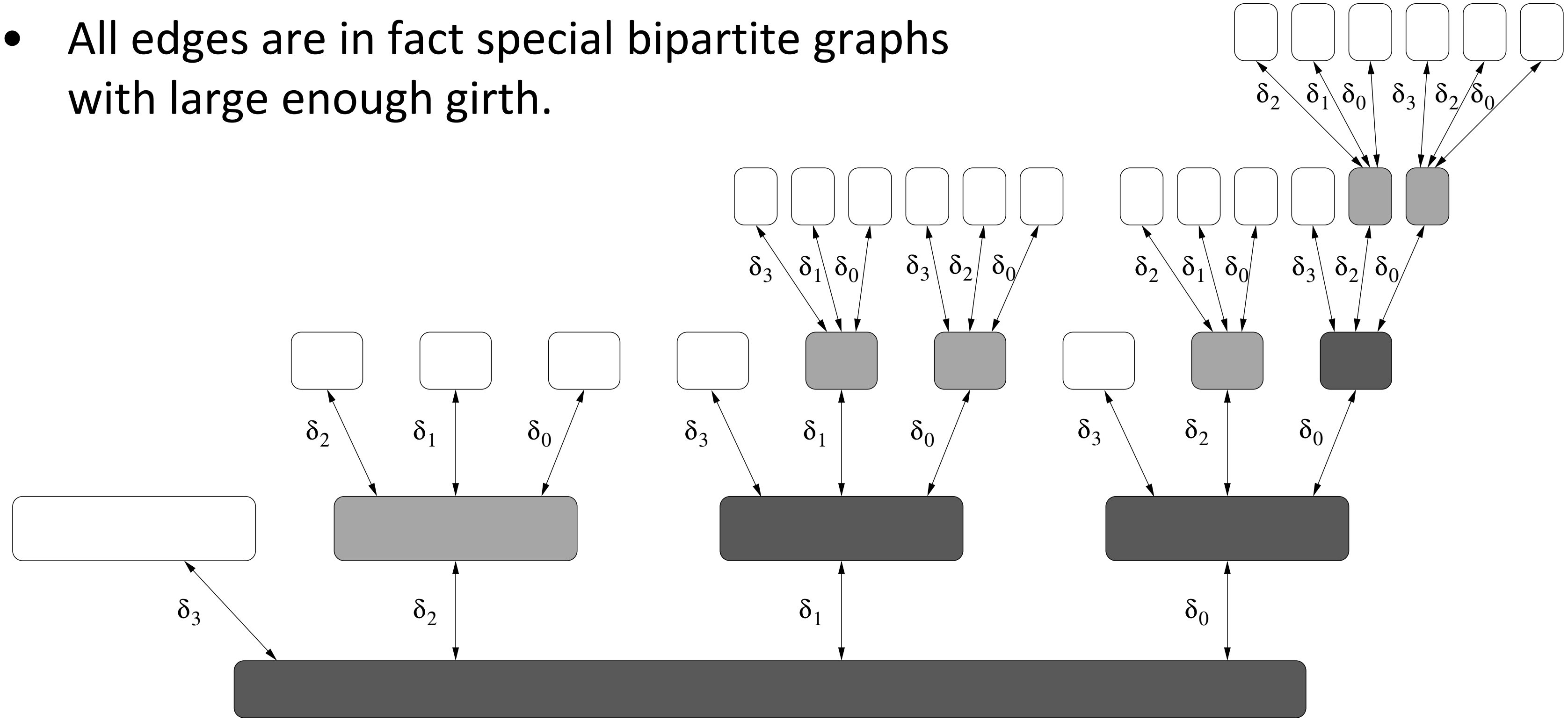


$|DS_{OPT}| = m+1.$

$|DS_{OPT\ \text{with green}}| > n$

Graph Used in the Lower Bound

- The example is for $t = 3$.
- All edges are in fact special bipartite graphs with large enough girth.



The Lower Bound

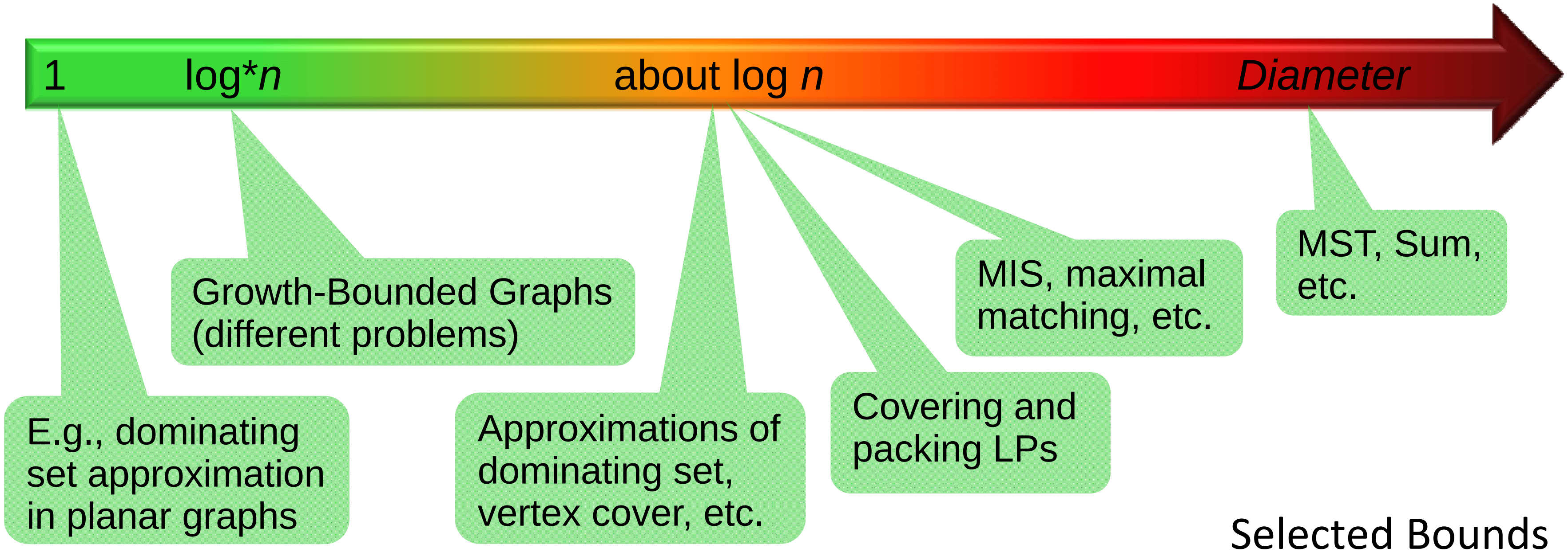
- Lower bounds (Kuhn et al., PODC 2004, SODA 2006):
  - Local model: In a network/graph *G*, each node can exchange a message with all its neighbors for *t* rounds. After *t* rounds, node needs to decide.
  - We construct the graph such that there are nodes that see the same neighborhood up to distance *t*. We show that node ID's do not help, and using Yao's principle also randomization does not.

  - Results: Many problems (vertex cover, dominating set, matching, etc.) can only be approximated by factors $\Omega(n^{c/t^2} / t)$ and/or $\Omega(\Delta^{1/t} / t)$.
  - It follows that a polylogarithmic dominating set approximation (or a maximal independent set, etc.) needs at least $\Omega(\log \Delta / \log\log \Delta)$ and/or $\Omega((\log n / \log\log n)^{1/2})$ time.

Self-Stabilization & Local Algorithms (Lower & Upper Bounds)

Theorem: Self-Stabilization = Local Algorithms

Corollary: Local algorithm lower bounds apply to the self-stabilization model as well.



Selected Bounds

The "Gretchen" Question

Theorem: Self-Stabilization = Local Algorithms

Is this known?!?

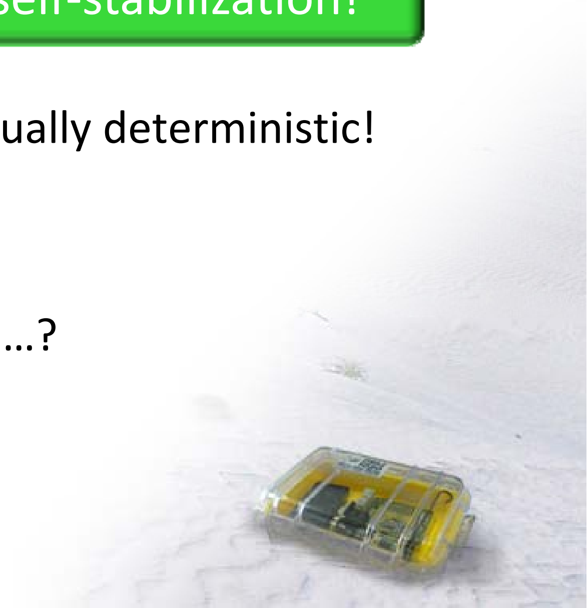
Is „Self-Stab = Local Algos“ Known?

If I ask my friends that are into self-stabilization, the answer is „sure!“

However, if I search „self-stabilization XYZ“ in Google Scholar, I always find published papers (some very recently) that are exponentially worse than the state-of-the-art local algorithm, and that do not cite any local algorithms or lower bounds.

My friends in self-stabilization say “There is more to self-stabilization!”

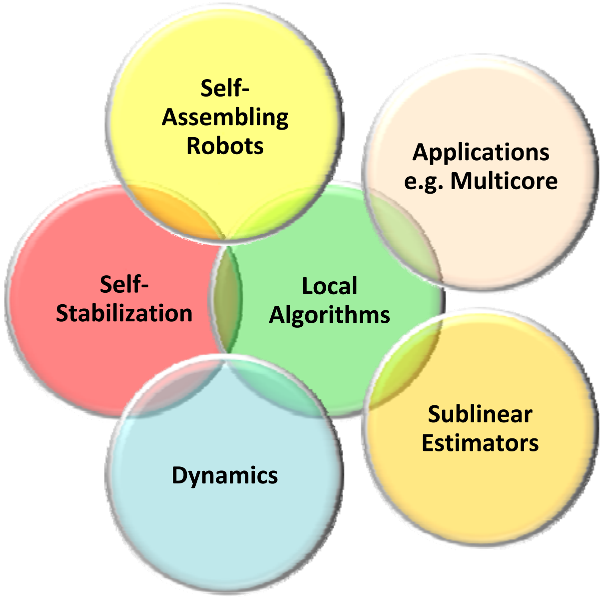
- But your algorithms are often randomized, ours are usually deterministic!
- But what about bit complexity?
- But what about asynchronous systems?
- But what about snap-stabilization, super-stabilization, …?

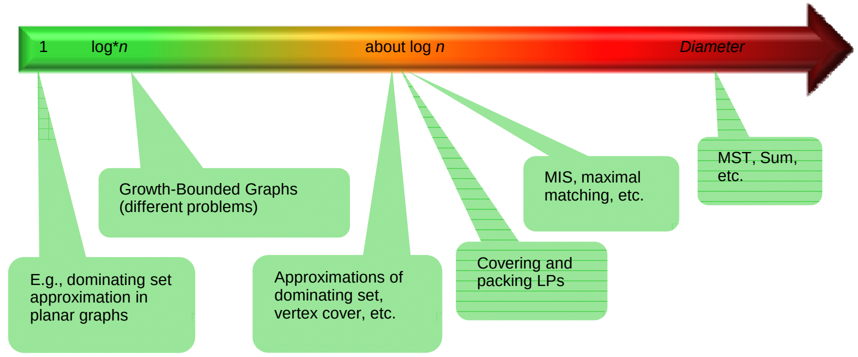


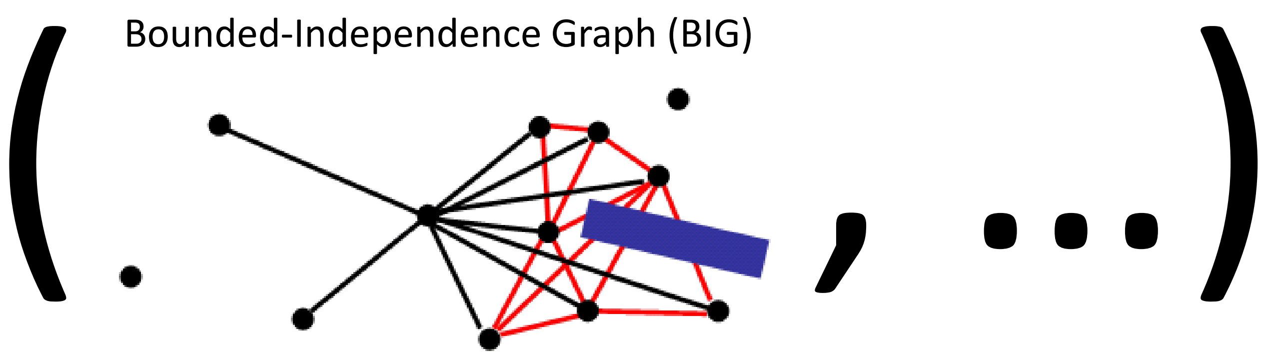
„But…“

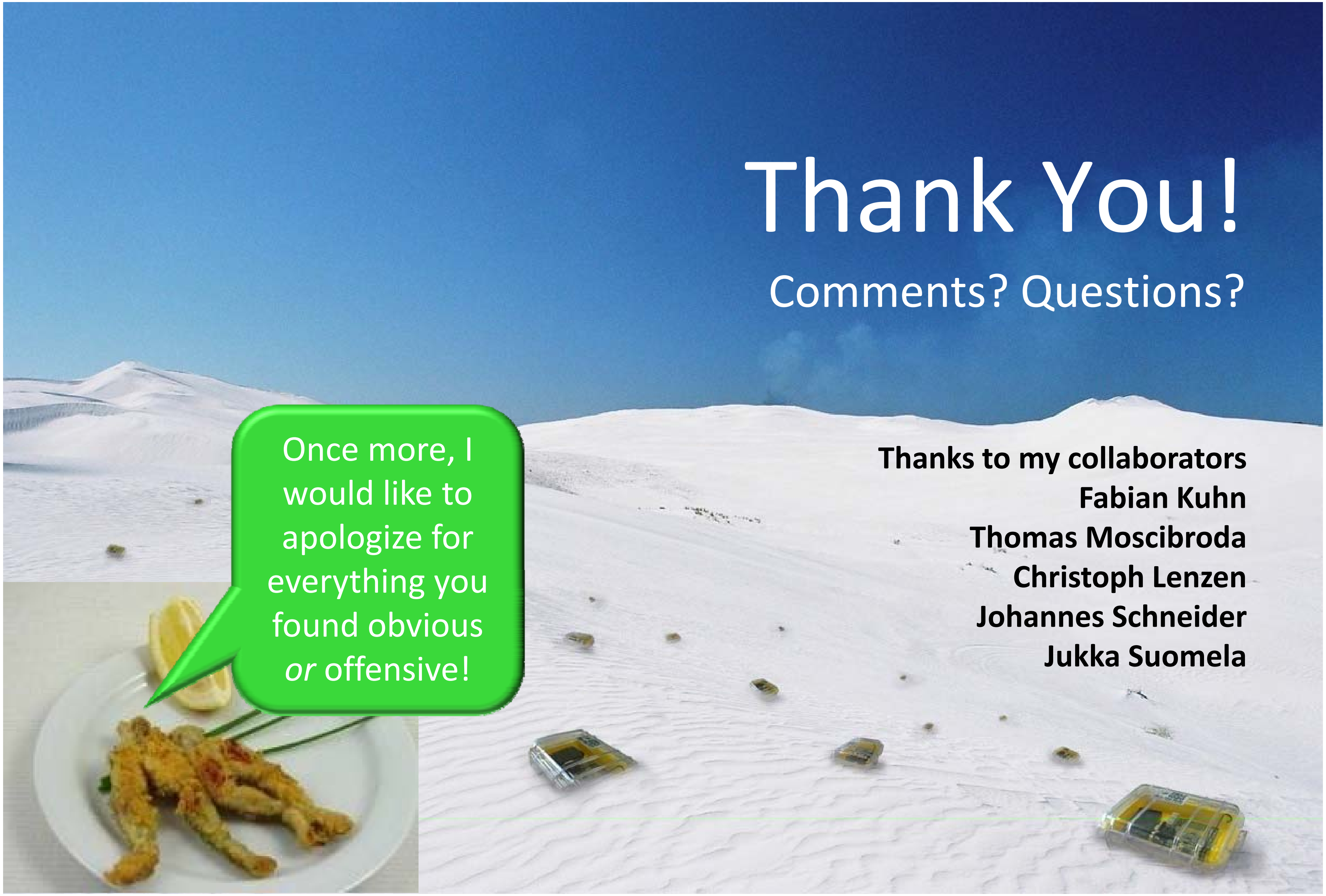
- Randomization
  - There are some pretty fast deterministic local algorithms.
  - One simple idea is to store random seed in ROM. Any self-stabilizing algorithm needs some kind of storage (for code) that cannot be tampered.
- Bit Complexity
  - Local algorithms often just need (poly)logarithmic many rounds, during which they often exchange just a few bits. In addition, information may be compressed, so that all in all, messages are usually of (poly)logarithmic size.
- Asynchronous Systems
  - When turning a local algorithm into a self-stabilizing algorithm using the technique presented on slides 6 and 7, it will automatically be asynchronous, as there is no notion of time. In other words, no synchronizer is needed.
- Snap-Stabilization, Super-Stabilization, Silent Stabilization, etc.
  - I cannot claim that local algorithms solve everything; for that I am not familiar enough with the area (frog‘s eye view!).

Summary & Open Problems








Thank You!
Comments? Questions?
Once more, I would like to apologize for everything you found obvious *or* offensive!
Thanks to my collaborators
Fabian Kuhn
Thomas Moscibroda
Christoph Lenzen
Johannes Schneider
Jukka Suomela