Designing robust distributed systems with weakly interacting feedback structures

THE ADVENTURES OF SELFMAN

EU FP6 Project

LIGHTKONE

EU H2020 Project

CodeMesh 2017

Nov. 9, 2017

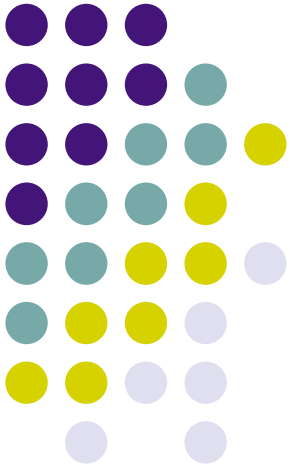
Peter Van Roy

ICTEAM Institute
Université catholique de Louvain
Louvain-la-Neuve, Belgium

Overview

- Motivation
  - Large systems are too complicated for human management
  - Can we make systems self managing?
- Feedback loops
  - One is easy, but how can we combine them?
- Feedback structures
  - Examples of biological and computer systems
  - An interesting design pattern appears!
- Architecture of scalable self-managing systems
  - Weakly interacting feedback structures
  - How to add intelligent behavior
- Conclusion and the future

Designing robust distributed systems

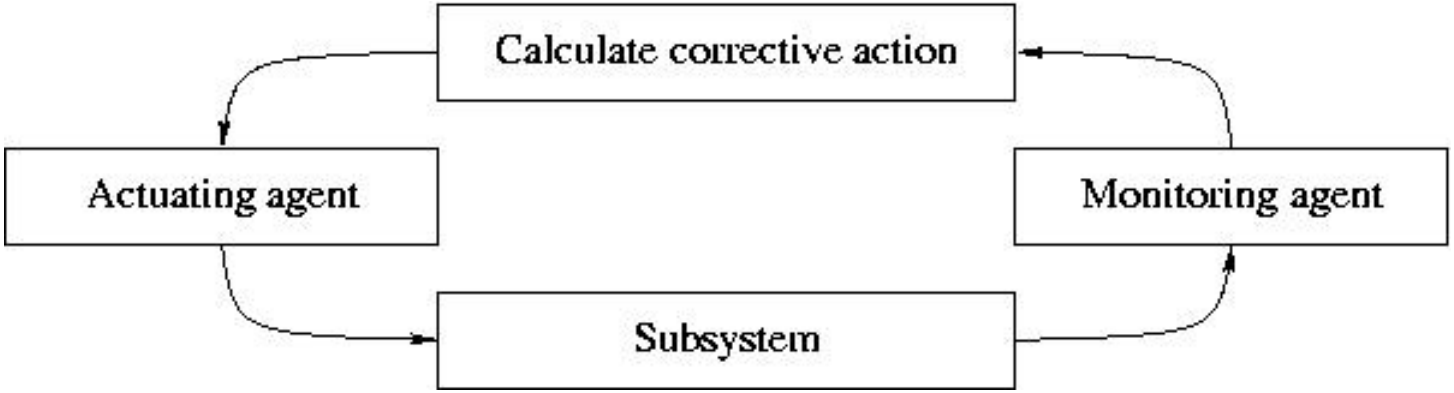


Designing robust distributed systems

- A system is a set of components (called subsystems) that are connected together to form a coherent whole
  - A distributed system is a set of computing nodes, connected by a network, that appears to its users as a single coherent system
- Building large distributed systems is today's bread and butter
  - System execute on many nodes and provide services in unreliable environments
  - Complex libraries are patched together, stress tested, and then require continual maintenance by human experts
  - Since Wiener's work on cybernetics [Wiener 1948], research on self management
- Famously, autonomic computing [IBM 2001] simplifies system management by means of a MAPE feedback loop (Monitor-Analyze-Plan-Execute)
  - But real systems need many feedback loops: how do we combine them?
  - This is the question addressed by this talk!

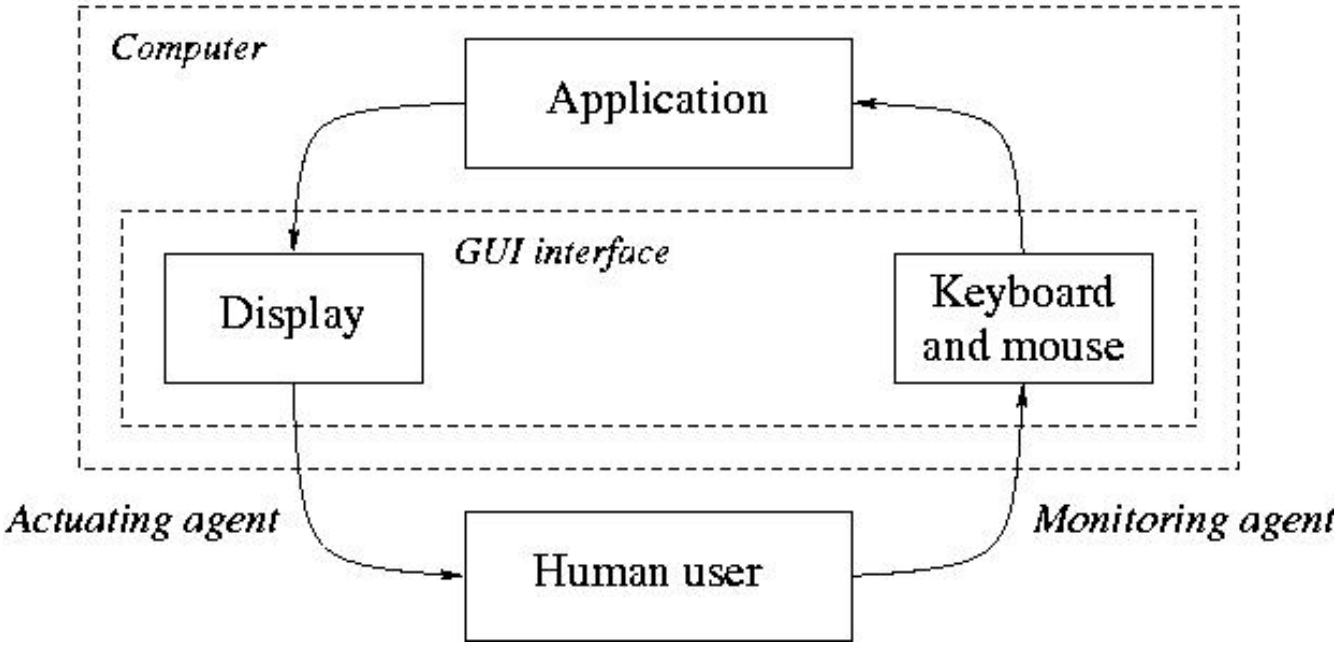
Feedback loops

Feedback loops



- A feedback loop consists of a monitoring agent, a correcting agent, and an actuating agent, interacting with a subsystem in a loop
  - All agents and the subsystem are concurrent components that communicate through asynchronous message passing
  - The subsystem can be hierarchical (itself built of agents)

Feedback loops are everywhere



- Feedback loops are literally everywhere, if you look at a system with the right mindset
- Human-computer interfaces are a simple example

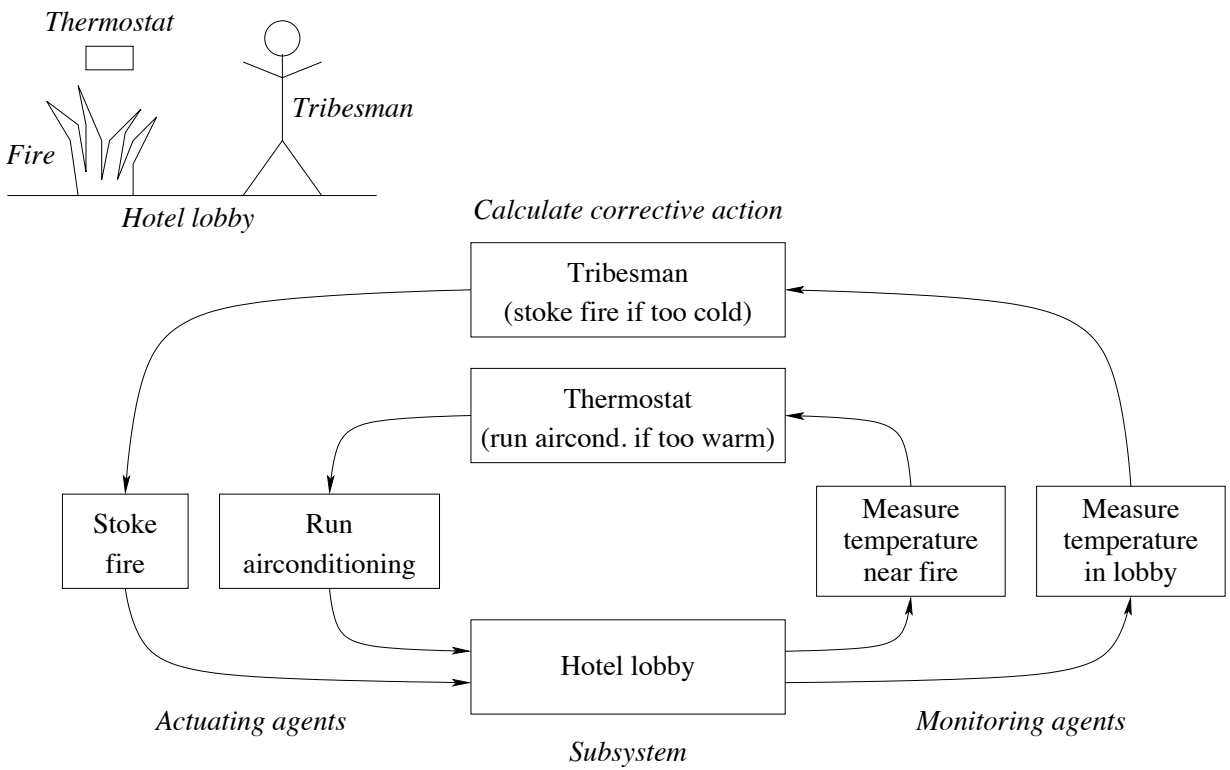
Limits of feedback loops

- Feedback loops are used everywhere
  - Optimizing feedback loops is well understood (control theory)
- But feedback loops are a low-level concept: realistic systems need many of them working together
  - How should they be combined together?
  - How do they relate to the system functionality?
- Combining feedback loops is the topic of this talk
  - We show how to combine feedback loops and connect them to the system specification
- Let's first look at some examples
  - And then we will propose a general pattern

Example 1: Hotel lobby

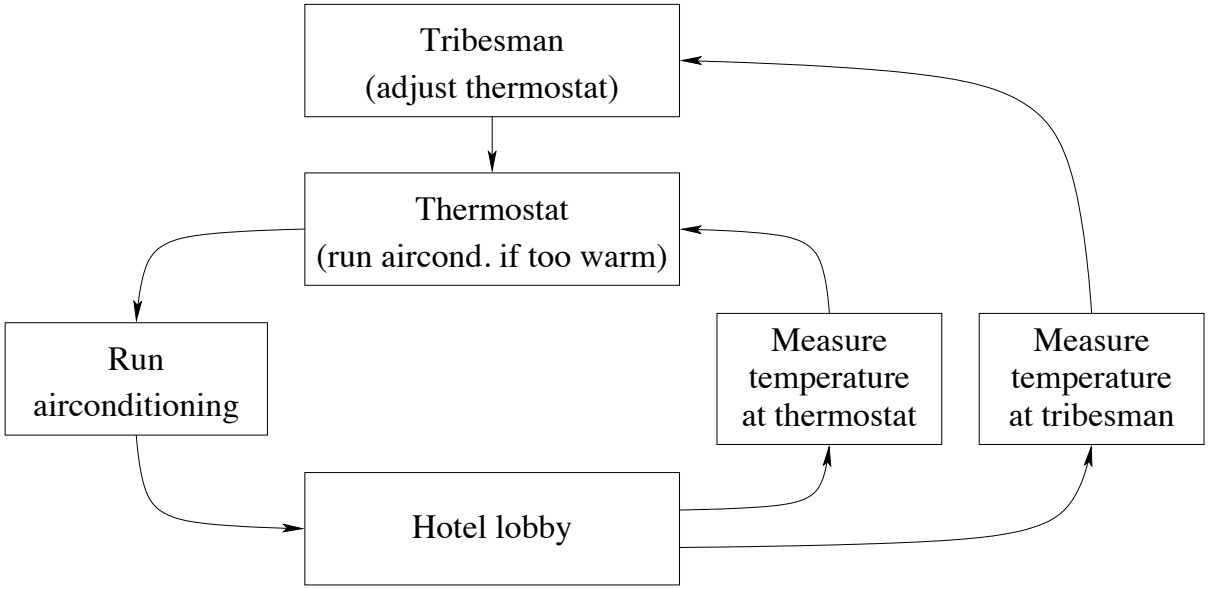
Simple example: hotel lobby

- Two loops interacting through a common subsystem (stigmergy)
- This example comes from N. Wiener's book on Cybernetics (1948)



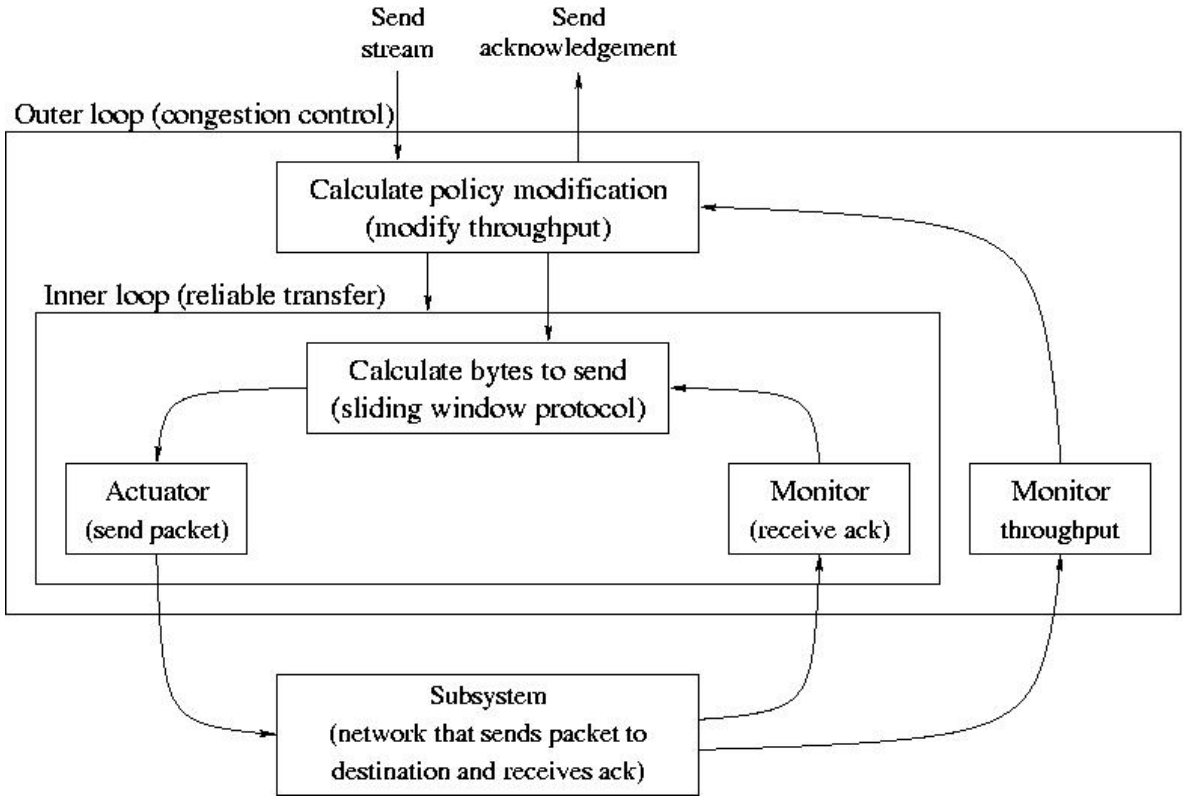
- A primitive tribesman in a modern hotel lobby
  - The tribesman stokes the fire but gets colder and colder because the air conditioning works harder and harder
  - This is unstable!
- Wiener leaves the fix as homework for the reader (!)
- How would you debug this system?

Hotel lobby solution



- Instead of stoking a fire, the tribesman simply adjusts the thermostat. The resulting system is stable.
- This uses management (one loop controls another) instead of stigmergy (two loops interact through the environment)
- Design pattern: use the system, don't try to bypass it

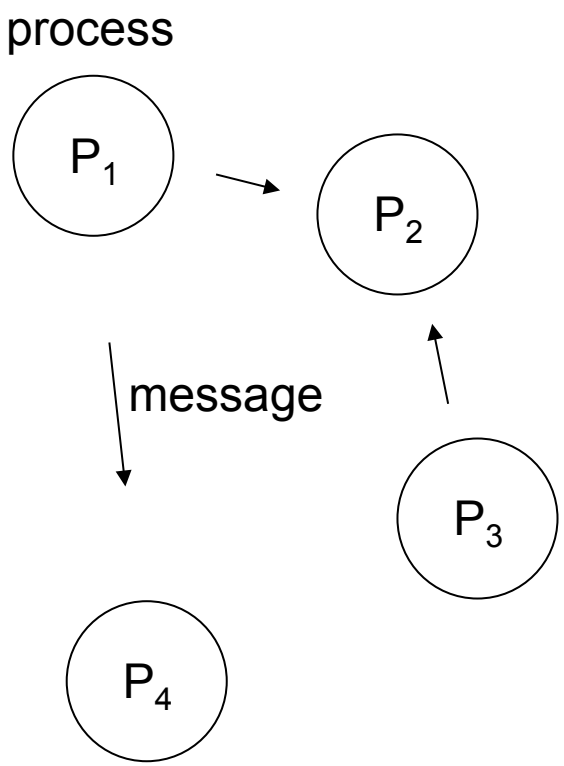
TCP uses a similar approach



- This example shows a reliable byte stream protocol with congestion control (a part of TCP)
  - This diagram is for the sending side
- The congestion control loop manages the reliable transfer loop
  - By changing the sliding window's buffer size

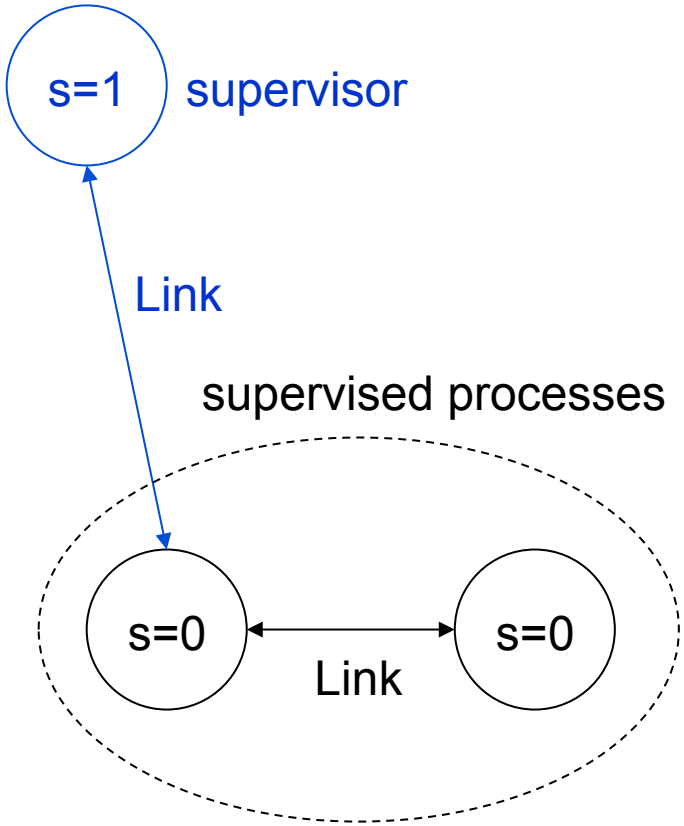
Example 2: Erlang's supervisor trees

Erlang execution model



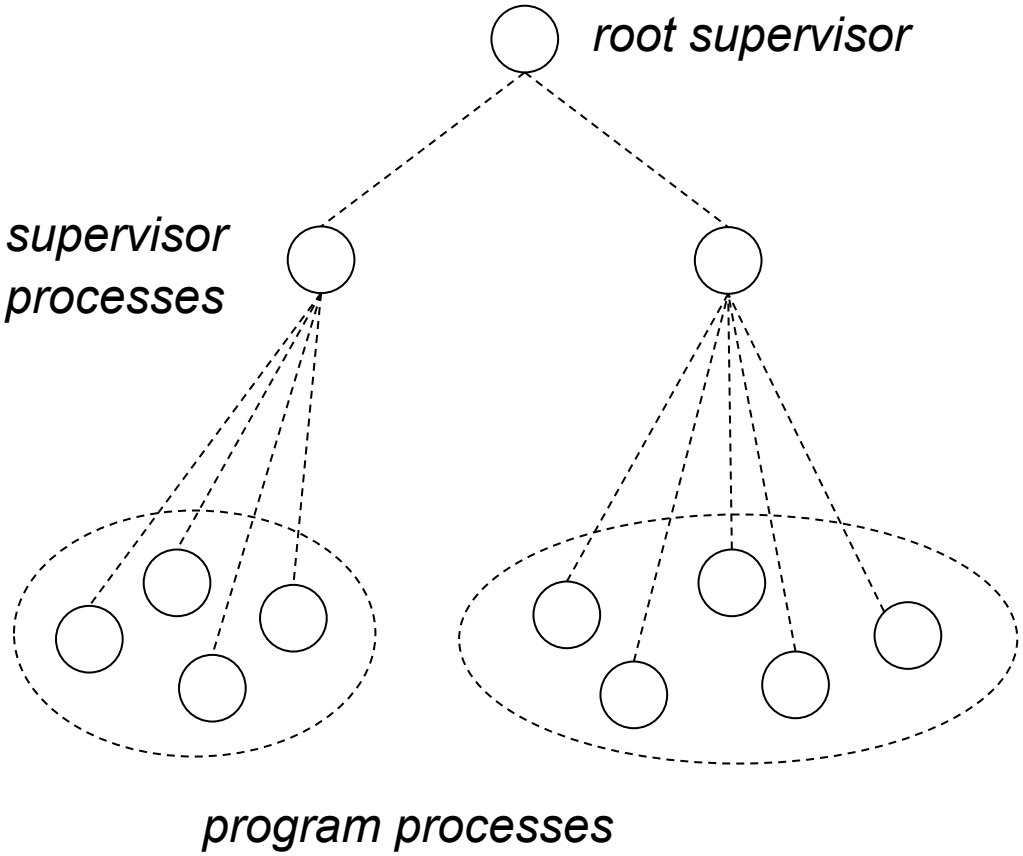
- Erlang is a language used to develop highly reliable software systems
- An Erlang program consists of a set of running processes (lightweight threads with independent address spaces) that send messages asynchronously
- Fault tolerance in three levels:
  - Primitive failure detection through process linking: when one process fails, another is notified
  - Supervisor trees to structure the program
  - Stable storage (database) to restore consistent state after crashes

Primitive failure detection



- Two processes can be linked: if one fails then both are terminated
  - Failure is a permanent crash failure, detected by the run-time system
  - "Let it fail" philosophy: if anything goes wrong, just crash and let another process correct the problem
- If a linked process has its supervisor bit set, then it is sent a message instead of failing
- This primitive failure detection does monitoring in a feedback loop

Supervisor trees



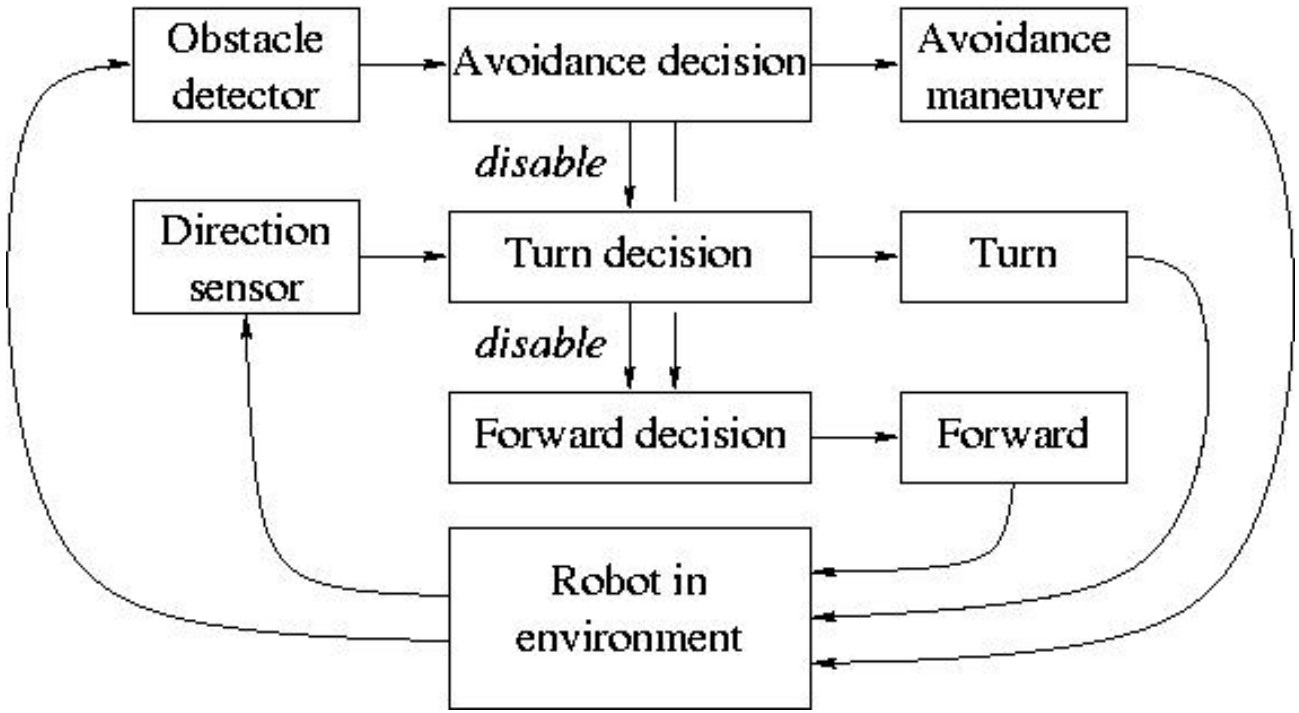
- The program consists of a large number of processes
- Program processes are organized in pools
  - Each pool is observed by a supervisor process linked to all of them
  - An AND supervisor stops and restarts all its children if one crashes
  - An OR supervisor restarts just the crashed child
- Supervisors are themselves observed by a root supervisor
- Each internal node in the supervisor tree corresponds to one feedback loop

Example 3: Brooks' subsumption architecture

Subsumption architecture

- The subsumption architecture is a way to implement complex, “intelligent” behaviors by decomposing them into simpler behaviors
- The system consists of layers where each layer provides a simple ability
- Layers are given priorities: when a layer can act, it disables the lower layers
- Layers interact through stigmergy

An obstacle-avoiding robot



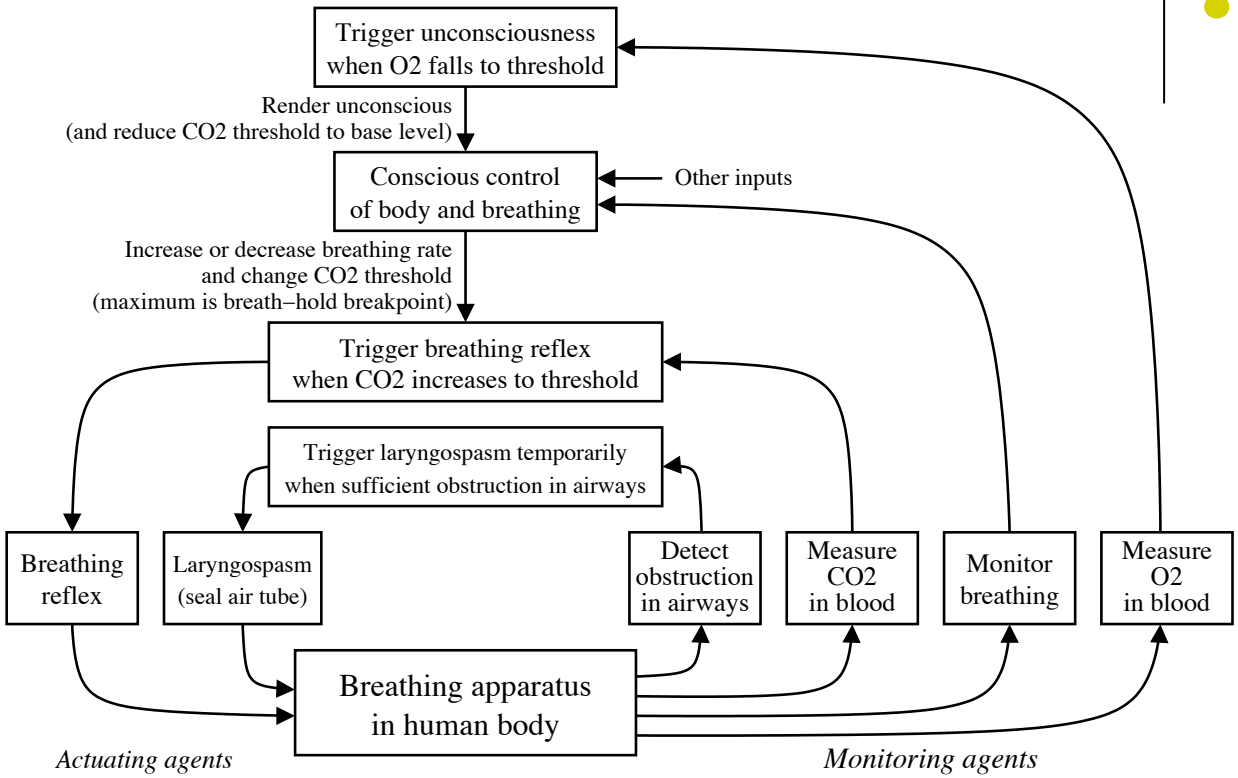
- Each layer provides a competence
- Each layer can override the lower layers
- If a higher layer fails, some competence remains

Example 4: Human respiratory system

Human respiratory system

The operation of the human respiratory system is given as one feedback structure, inferred from a precise medical description of its behavior

It has been debugged by billions of years of natural selection!



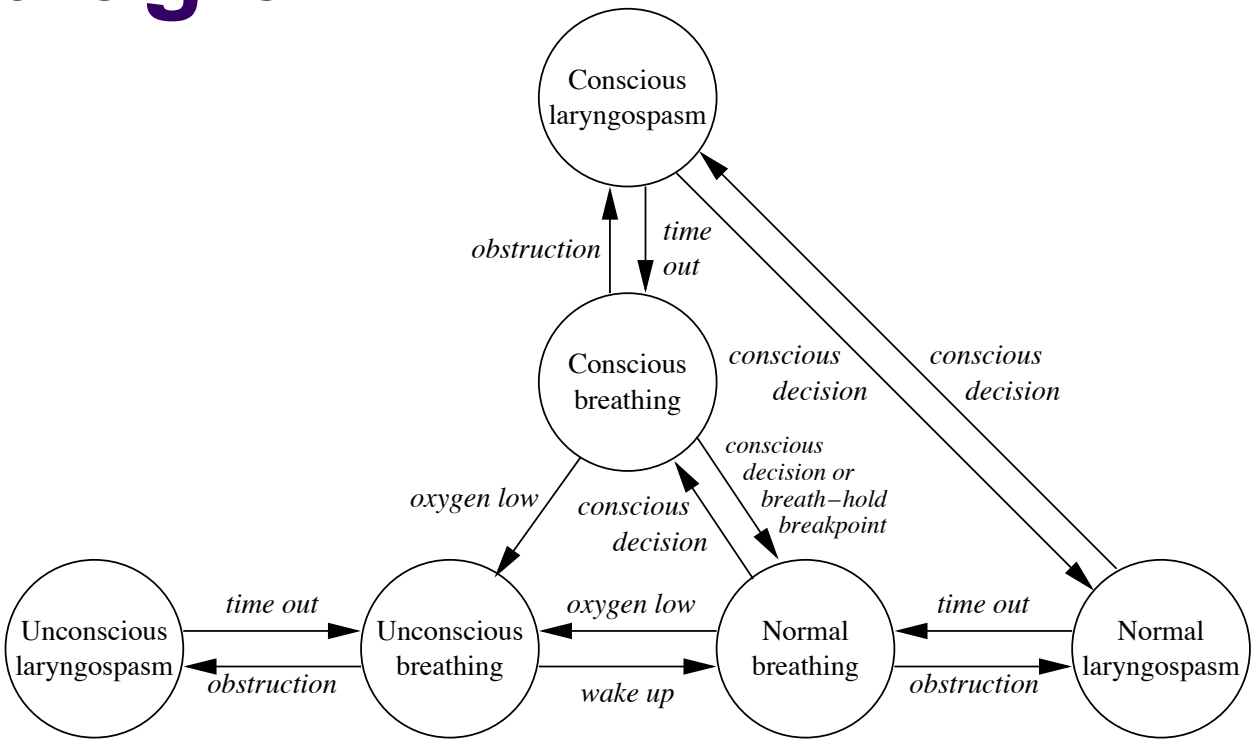
How it works:

- Default behavior: rhythmic breathing reflex
- Complex component: conscious control can override and plan lifesaving actions
- Abstraction: conscious control does not need to know details of breathing reflex
- Fail-safe: conscious control can itself be overridden (falling unconscious)
- Time scales: laryngospasm is a quick action that interrupts slower breathing reflex

Design of respiratory system

- Four feedback loops: two inner loops (breathing reflex and laryngospasm), a loop controlling the breathing reflex (conscious control), and an outer loop controlling the conscious control (falling unconscious)
  - This design is derived from a precise textual medical description [Wikipedia Entry "Drowning"]
- Holding your breath can have two effects
  - Breath-hold threshold is reached first and breathing reflex happens
  - O_2 threshold is reached first and you fall unconscious, which reestablishes the normal breathing reflex
- Some plausible design rules inferred from this system
  - Conscious control is sandwiched in between two simpler loops: the breathing reflex provides abstraction (consciousness does not have to understand details of breathing) and falling unconscious provides protection against instability
  - Conscious control is a powerful problem solver but it needs to be held in check

Respiratory system state diagram



- The behavior of the human respiratory system modeled as a state diagram
- Each state corresponds to a subset of active feedback loops
  - Each state can handle a well-defined range of operating conditions

Feedback structures

Feedback structures

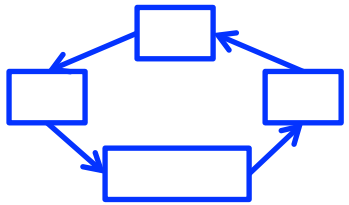
- What do all these examples have in common?
  - They use a combination of feedback loops to maintain a system property such as desirable hotel lobby temperature, fault-tolerant execution, efficient communication, moving while avoiding obstacles, or proper human breathing.
  - We call such a combination a feedback structure

- A feedback structure is a set of feedback loops that work together to maintain one global system property

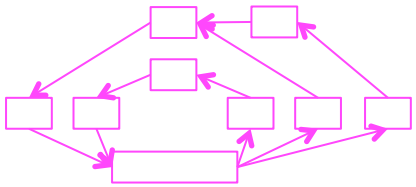
A scalable architecture in four steps

- Concurrent component
  - An active entity communicating with its neighbors through asynchronous messages

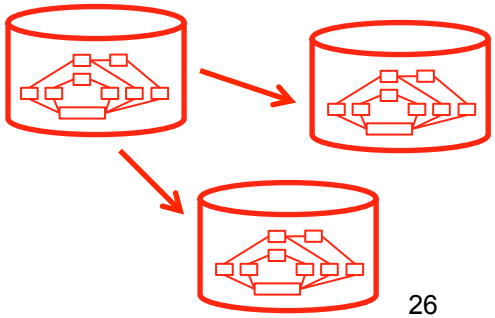
- Feedback loop
  - Monitor, corrector, and actuator components connected to a subsystem and continuously maintaining one local goal



- Feedback structure
  - A set of feedback loops that work together to maintain one global system property

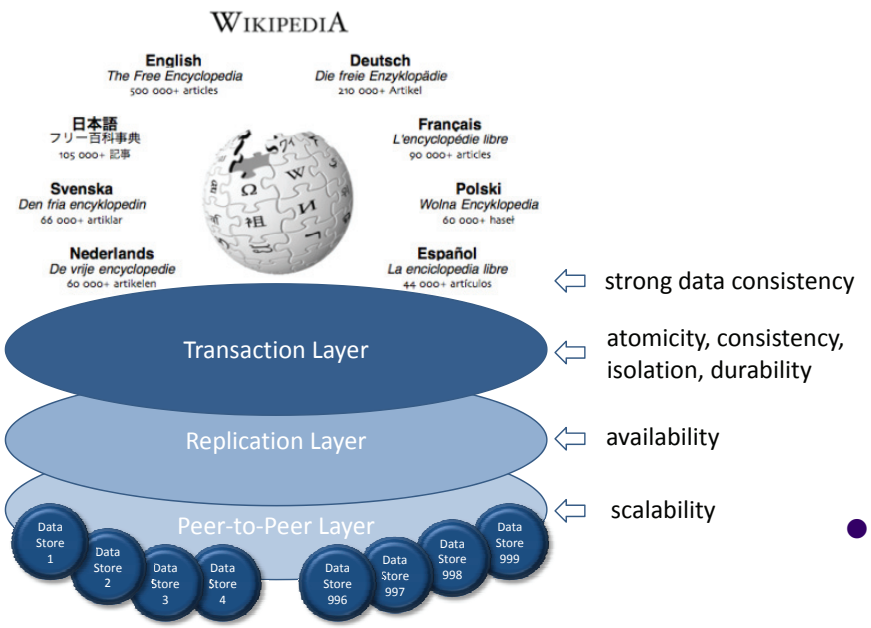


- Weakly interacting feedback structures
  - The complete system is a conjunction of global properties, each maintained by one feedback structure
  - The feedback structures are part of a dependency graph



Design with feedback structures: Scalaris

Scalaris key/value store



- Scalaris is a high-performance transactional self-managing key/value store that is built on top of a structured overlay network
  - SELFMAN project result (www.ist-selfman.org)
  - 4000 read-modify-write transactions per second on two dual-core Intel Xeon at 2.66 GHz

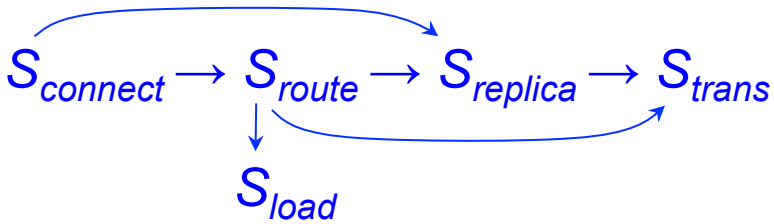
- Scalaris won the IEEE International Scalable Computing Challenge 2008 by building a scalable Wikipedia

Scalaris design uses 5 WIFS

- The Scalaris specification is conjunction of six properties. Each non-functional property is implemented by one feedback structure.

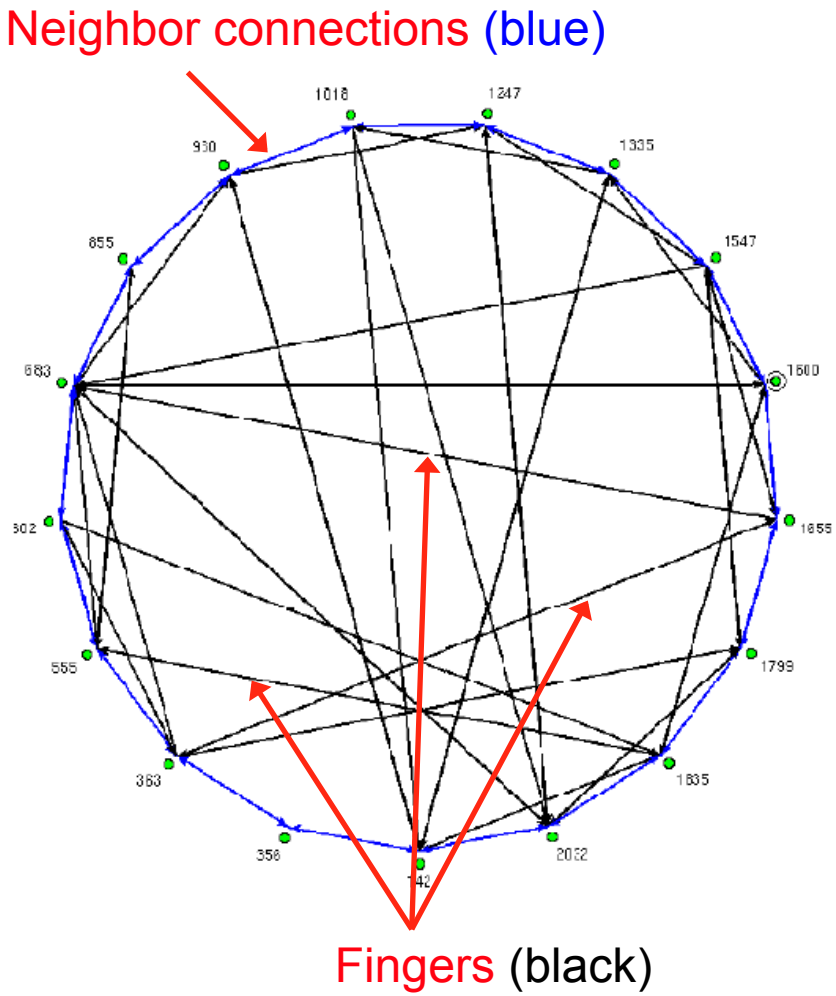
$$S_{scalaris} = S_{key\text{-}value} \wedge S_{connect} \wedge S_{route} \wedge S_{load} \wedge S_{replica} \wedge S_{trans}$$

- Scalaris design uses five feedback structures: connectivity management ($S_{connect}$), efficient routing (S_{route}), load balancing (S_{load}), replica management ($S_{replica}$), and transaction management (S_{trans})
- We arrange the feedback structures in a dependency graph that shows how each needs the functionality of the others:



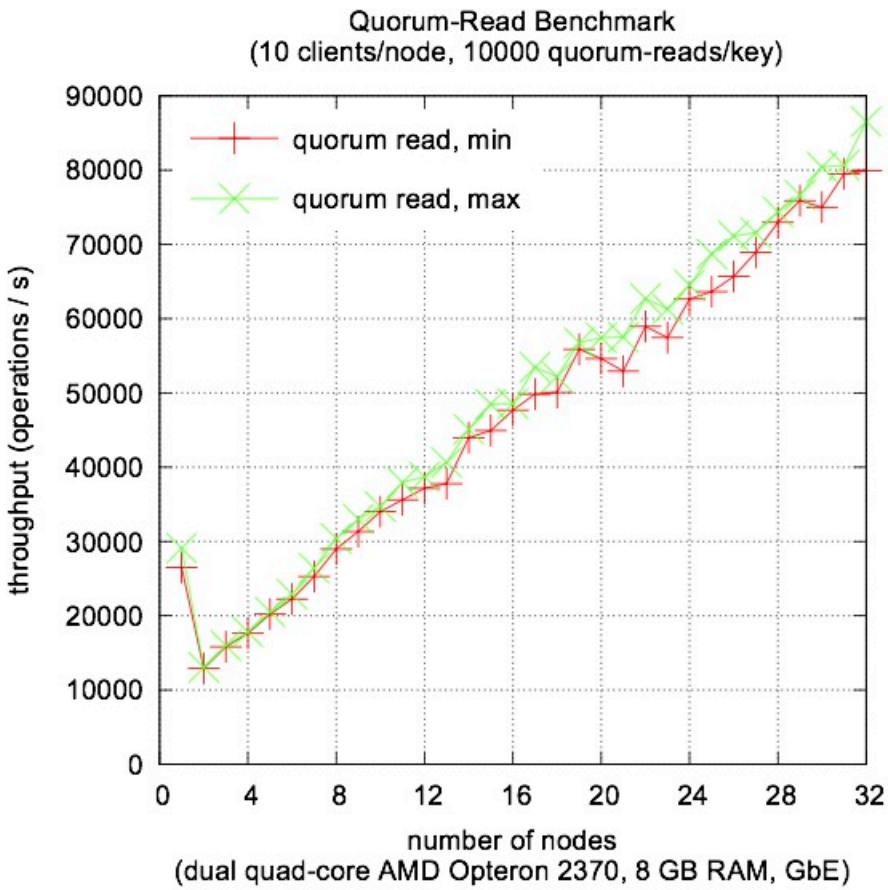
- Since all feedback structures live in the same system, they influence each other ("weakly interacting feedback structures" – WIFS)

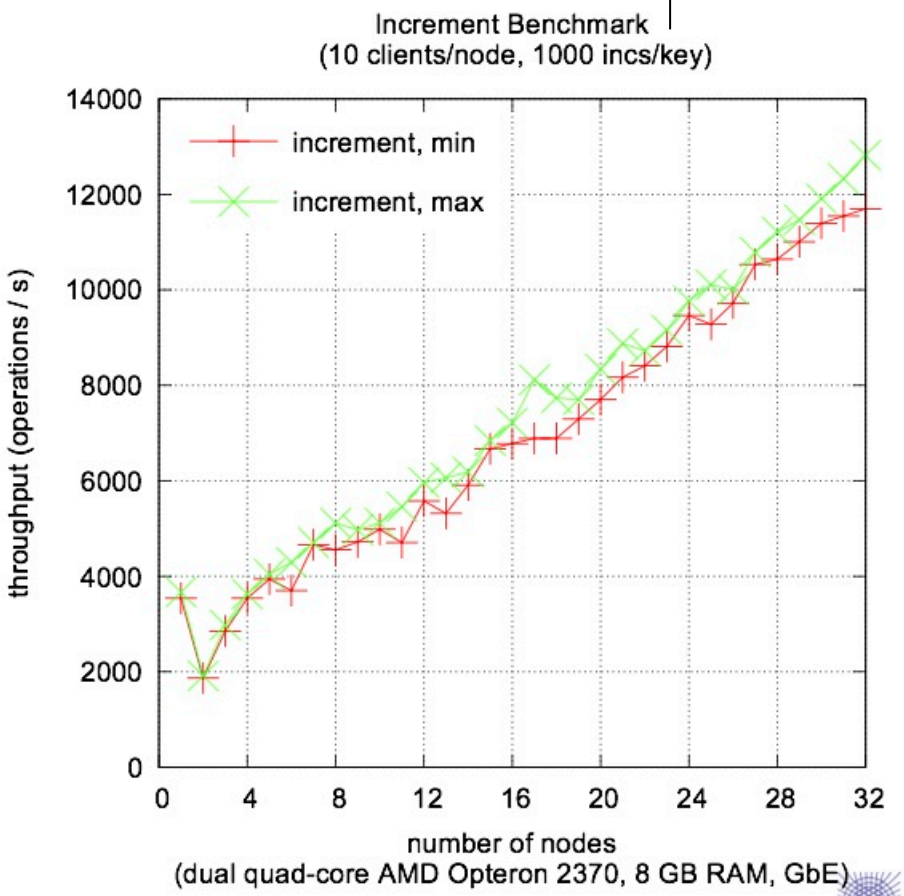
Scalaris is based on a structured overlay network



- Structured overlay networks are often based on a ring
  - By far the most popular structure, it has many variants and has been extensively studied
- Communication is organized by two feedback structures:
  - The neighbor connections ensure connectivity: despite node joins, leaves, and failures
  - The fingers provide efficient routing: they can be temporarily in an inconsistent state

Scalaris scalability



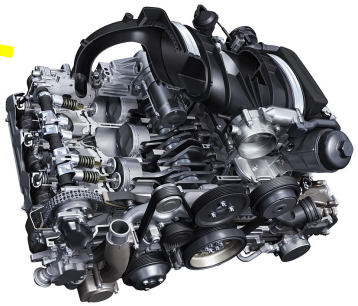


Smart components

Smart components are essential

- The WIFS approach is a framework for robust adaptive systems
  - Each feedback structure corresponds to one global property; a subset of the feedback structure is active at any given time
  - But a feedback structure is just a framework; we still need to design the components themselves
- Intelligence is added by means of “smart” components
  - If the system needs intelligence, it needs at least one smart component
  - For example, Conscious Control in the human respiratory system
- Smart components *completely solve* a problem inside a specific (small) part of the system operating space
  - For example, conscious control, a deep learning algorithm, and a compiler are extremely smart *within their operating spaces*
  - Outside of this space, they are very stupid and should be inactive

Power is built in, not added on



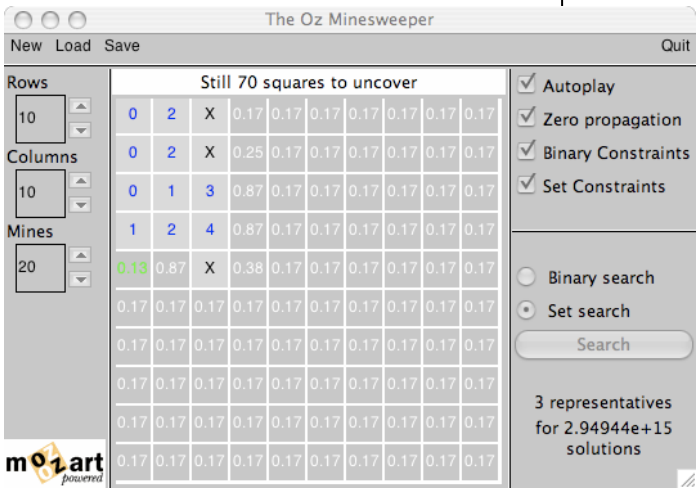
3.6-Liter Biturbo Motor with 353 kW (480 HP)

Porsche Carrera GT

- The power of a system depends on the strength of its smart components
  - The human respiratory system uses conscious control (e.g., to avoid drowning!)
  - Scalaris uses uniform consensus to implement fast transactions
  - AlphaGo uses deep learning and Monte Carlo tree search
  - Google Search uses eigenvector calculation of the Web link matrix
  - What does *your* system use? ☺

Intelligence is a combination of many smart components

- Human intelligence
  - Main strength: adaptability (dynamic creation of new feedback loops)
- Program intelligence
  - Can easily go beyond human intelligence in many areas!
    - **Turing test is irrelevant**: complex components are already replacing humans in more and more areas
  - Deep learning: backpropagation to give human-level or better performance on real-world tasks
  - Compiler: human-readable program translated into executable form
  - Libraries: many libraries distill some form of intelligence (e.g., consensus)



Conclusion and the future

Conclusion and the future

- Design of large robust distributed systems is difficult
  - Because they provide many services and the environment can be hostile
- How can we design them?
  - We learn from existing systems that work: biological and computing systems
  - We observe a useful design pattern: feedback structure
    - A set of feedback loops that together maintain one global system property
- Large systems consist of multiple dependent feedback structures
  - Architecture: weakly interacting feedback structures
  - Intelligence: complex components add intelligence where it is needed
- Future work
  - General-purpose edge computing (LightKone project)
    - Convergent computation in a hostile distributed environment
  - Intelligent components (deep learning)
  - Large-scale evaluation

LIGHTKONE

References

References

- Norbert Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine*. MIT Press, Cambridge, MA, 1948.
- **Peter M. Senge *et al*. *The Fifth Discipline Fieldbook: Strategies and Tools for Building a Learning Organization*. Nicholas Brealey Publishing, 1994.**
- Gerald M. Weinberg. *An Introduction to General Systems Thinking*. Dorset House Publishing, 1975 (Silver Anniversary Edition 2001).
- J. O. Kephart and D. M. Chess. "The Vision of Autonomic Computing", IEEE Computer, 36(1), Jan. 2003, pp. 41-50.
- Joe Armstrong. *Making Reliable Distributed Systems in the Presence of Software Errors*. Ph. D. dissertation, Royal Institute of Technology (KTH), Kista, Sweden, Nov. 2003.
- Joseph Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, Aug. 2004.
- Ahmad Al-Shishtawy, Joel Höglund, Konstantin Popov, Nikos Parlavantzas, Vladimir Vlassov, and Per Brand. "Distributed Control Loop Patterns for Managing Distributed Applications", Workshop on Decentralized Self Management for Grids, P2P, and User Communities, 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008), Oct. 2008, pp. 260-265.
- SELFMAN: Self Management for Large-Scale Distributed Systems Based on Structured Overlay Networks and Components. European 6th Framework Programme, www.ist-selfman.org (2009).
- Alexandre Bultot. *A Survey of Systems with Multiple Interacting Feedback Loops and Their Application to Programming*. Master's thesis, Dept. Comp. Sci. and Eng., UCL, Aug. 2009.
- Florian Schintke, Alexander Reinefeld, Seif Haridi, and Thorsten Schütt. "Enhanced Paxos Commit for Transactions on DHTs", 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010), May 17-20, 2010, Melbourne, Australia.
- Nicolas Varotto. *Designing Robust Adaptive Software with Feedback Structures*. Master's thesis, Dept. Comp. Sci. and Eng., UCL, June 2013.
- **Peter Van Roy, Seif Haridi, and Alexander Reinefeld. "Software Design with Weakly Interacting Feedback Structures and Its Application to Distributed Systems", Research Report, Dept. Comp. Sci. and Eng., UCL, 2011.**
- Francesco Cesarini and Steve Vinoski. *Designing for Scalability with Erlang/OTP: Implementing Robust Fault-tolerant Systems*. O'Reilly, 2016.

Appendices

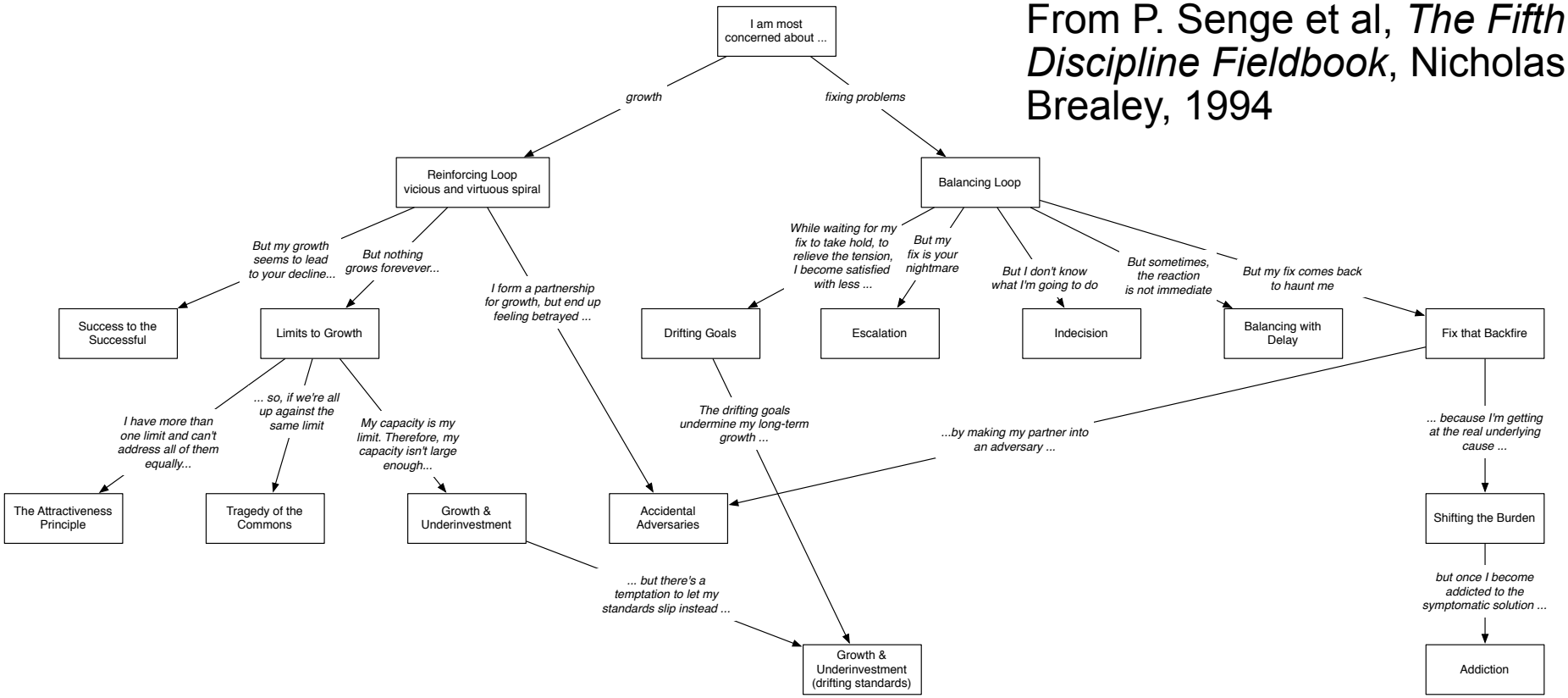
Libraries of feedback structures

Finding the right feedback structure for your design

- Feedback structures are bread and butter of adaptive system design
  - Erlang provides some in OTP, but there exist many more!
  - We don't want to invent them from scratch
- There exist several very useful collections of feedback structures
  - Peter M. Senge *et al*. *The Fifth Discipline Fieldbook: Strategies and Tools for Building a Learning Organization*. Nicholas Brealey Publishing, 1994.
  - Alexandre Bultot. *A Survey of Systems with Multiple Interacting Feedback Loops and their Application to Programming*. Master's thesis, UCL, 2009.
  - Nicolas Varotto. *Designing Robust Adaptive Software with Feedback Structures*. Master's thesis, UCL, 2013.

Feedback structures in human organizations

From P. Senge et al, *The Fifth Discipline Fieldbook*, Nicholas Brealey, 1994



- Human organizations use feedback structures to adapt themselves to changing circumstances; these can be arranged in a tree

Feedback loops are everywhere

Feedback loops are really everywhere!

- Real life is literally filled with variations on the feedback principle.
  - Bending a plastic ruler: single stable state. The ruler resists with a force that increases with degree of bending, until equilibrium (or until ruler breaks: change of phase)
    - The ruler is a simple self-adaptive system.
    - The feedback loop: force imposed on ruler, ruler reacts with counteracting force, this may affect the force, etc.
  - Clothes pin: one stable and one unstable state. Can be kept temporarily in the unstable state by pinching. Release the force and it will go back to (a possibly more complex) stable state.
  - Safety pin: two stable states, open and closed. Within each stable state, the system is adaptive like the ruler. Example of feedback loop with management: the outer control chooses the stable state.
- Anything with duration is managed by a feedback loop
  - Lack of feedback means there is a runaway reaction (explosion)

Eating dinner

- Eating dinner, as a simple activity, has at least two feedback loops at its core
  - Inner loop: the actions of ingesting the food bite by bite “chew, swallow, get next morsel”
  - Outer loop: monitoring the condition “am I full yet?”
  - The outer loop controls the inner loop with management
- This activity, as part of daily life, will be inside other loops

Civilization relies on feedback loops

- Most products of human civilization use an implicit management feedback loop, called "maintenance", done by a human
  - Changing lightbulbs, replacing broken windows, filling up a car
- Each human mind is at the center of an enormous number of these feedback loops
  - Most require very little conscious thinking, since they have become "habits": programmed into the brain below consciousness
  - Each human being creates huge numbers of such habit programs
- If there are too many feedback loops to manage the human complains that "life is too complicated"!
  - "Civilization advances by reducing the number of feedback loops that have to be explicitly managed" (Van Roy's corollary to A. N. Whitehead's dictum)
  - A dishwashing machine reduces work of washing dishes, but it needs to be bought, maintained, replaced, etc. Is it worth it? Is the total effort reduced?

Why is conscious control so smart?

- Cognitive science and neuroscience try to understand why
  - The brain uses brute force, but in a very smart way
- Conscious control is a bricklayer: it continuously builds and organizes new components on top of existing components
  - This process is continuous from birth with compound interest effect, which is why humans are so smart in common-sense tasks
- It continuously brings the most useful concepts to the top (cache organization combined with "grandfather cells")
  - Manipulating common concepts is made easy
- "Mirror neurons": it can use its own components to simulate other humans, which is why humans can empathize so well with others
- It can efficiently execute up to two complex programs at once ("driving and conversation"), because the human brain has double pathways for all complex tasks

A more complicated example: human endocrine system

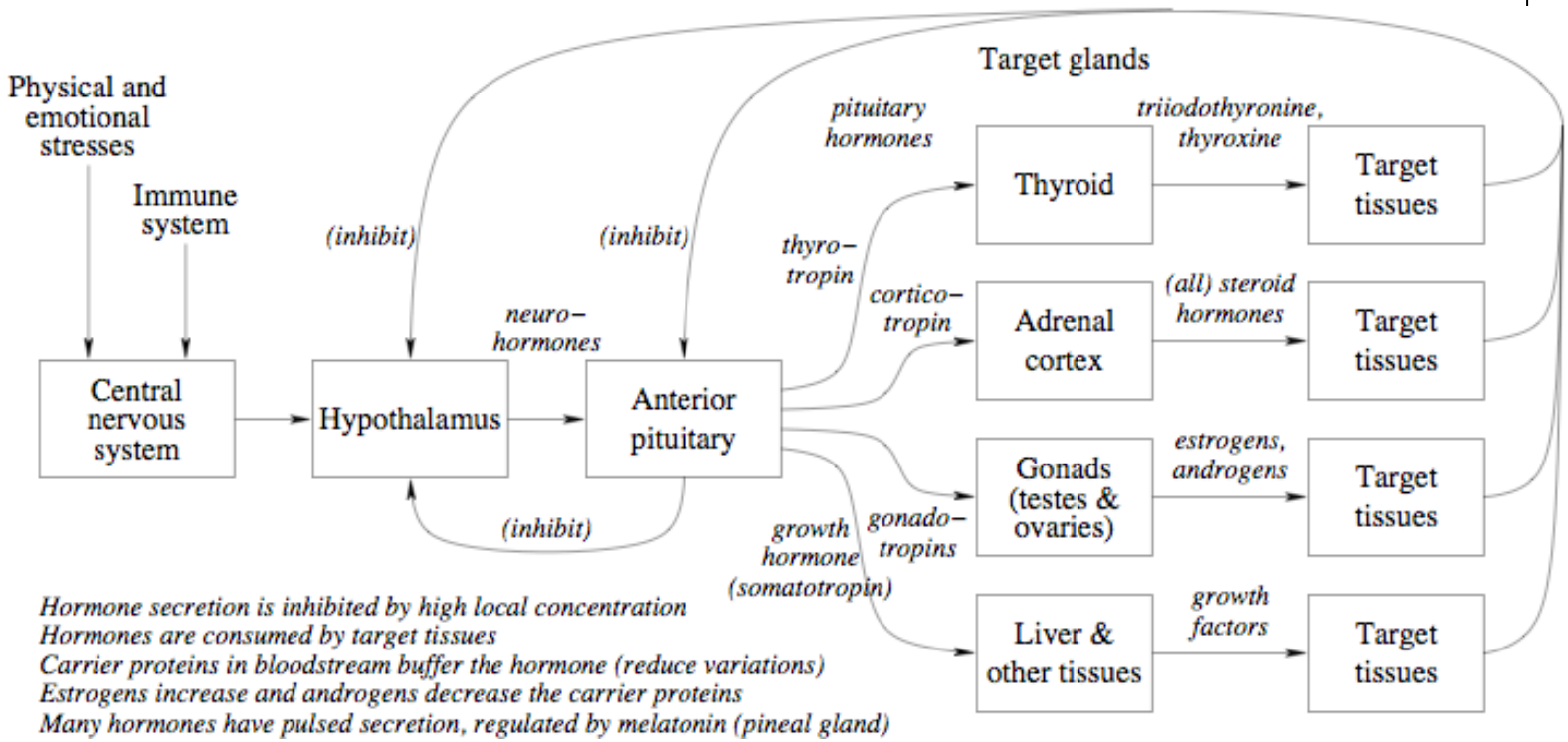
Human endocrine system

- The endocrine system regulates many quantities in the human body
- It uses chemical messengers called hormones which are secreted by specialized glands and which exercise their action at a distance, using the blood stream as a diffusion channel
- By studying the endocrine system, we can obtain insights in how to build large-scale self-regulating distributed systems

Feedback loops in the endocrine system

- There are many feedback loops and systems of interacting feedback loops in the endocrine system
  - Provides homeostasis (stability) and reaction to stresses
- Much regulation is done by simple negative feedback loops
  - Glucose level in blood is regulated by hormones glucagon & insulin. In the pancreas, A cells secrete glucagon and B cells secrete insulin. Increase in glucose in blood causes decrease in glucagon and increase in insulin. These hormones act on the liver, which releases glucose in the bloodstream.
  - Calcium level in blood is regulated by parathyroid hormone (parathormone) and calcitonine (also in opposite directions), which act on the bone
- More complex regulatory mechanisms exist, e.g., hypothalamus-pituitary-target organ axis
- There is interaction between nervous transmission and hormonal transmission

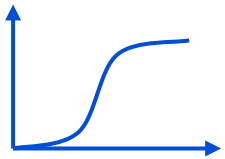
Hypothalamus-pituitary-target organ axis (endocrine system)



- Two superimposed groups of negative feedback loops, a third short negative loop, a fourth loop from the central nervous system [Encyclopaedia Britannica 2005]
- This diagram shows only the main components and their interactions; there are many more parts giving a much more complex full system

Discussion of endocrine system

- This system is quite complex
  - Many interacting feedback loops, many “short circuits”, many special cases, much interaction with other systems (nervous, immune)
    - Negative feedback for most, also saturation (logistic curve)



    - Evolution is not always a parsimonious designer!
      - Only criterion: it has to work
  - Several feedback loops are channeled through a single point, the hypothalamus-pituitary complex in the brain
    - So that the central nervous system can manage these loops
    - Different time scales are used: the loops are slow; the central nervous system is fast

Computational architecture of human endocrine system

- Local and global components
  - Local: gland, organ, or clumps of cells
  - Global (diffuse): large part of the body
- Point-to-point and broadcast channels
  - Fast point-to-point: nerve fiber, e.g., from spinal chord to muscle
  - Slower broadcast: hormone diffused by blood circulation
    - With buffering (reducing variations): carrier proteins
- Regulatory mechanisms can be modeled by interactions between components and channels
  - There are often intermediate links (short circuits)
  - Abstraction (encapsulation) is almost always leaky