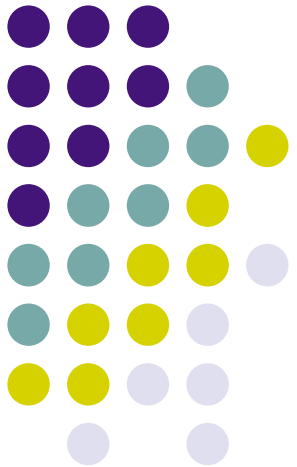
Scale and Design for Peer-to-Peer and Cloud

Modified version of talk given at TTI-Vanguard conference « Matters of Scale » on July 20, 2010

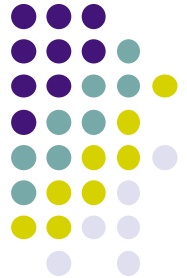
Feb. 12, 2012

Peter Van Roy

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

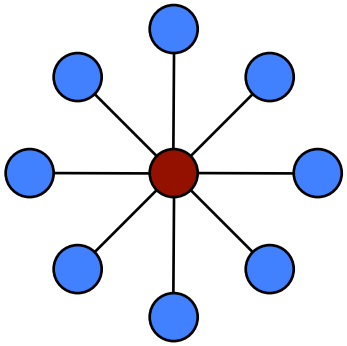


Three Laws of Scalability

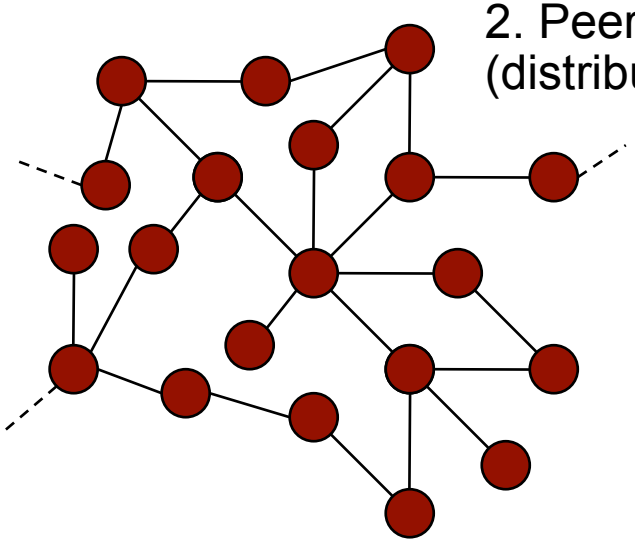


- The First Law:
  - New things happen at each new scale
  - Suggests a path toward future Internet structure
  - Emergence of elastic computing and Heisenberg applications
- The Second Law:
  - In the limit of increasing scale, large systems have only local control
  - Implies concurrency, asynchrony, and nondeterminism
- The Third Law (The CAP Theorem):
  - Pick any two of consistency, availability, and partition tolerance
  - Gives a map for navigating in the design space of scalability
- Designing for scalability
  - Mostly independent parts with carefully designed interactions
  - Weakly interacting feedback structures, complex components, and phases
  - Some scalable computing systems: Scalaris and Beernet

The Three Major Distribution Structures

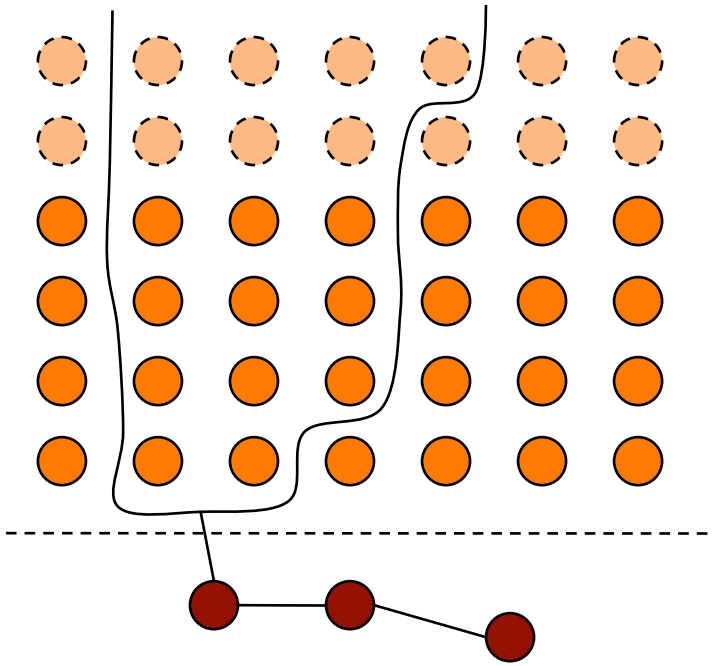


1. Client/server (distributed)



2. Peer-to-peer (distributed,scalable)

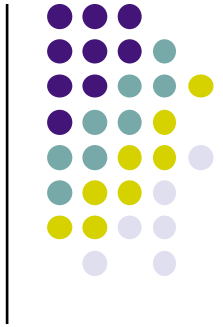
3. Cloud (distributed,scalable,elastic)



Elasticity: the ability to ramp resource usage up and down according to instantaneous demand

Elasticity opens up the new world of Heisenberg applications that we are just starting to exploit

What is Scalability?



- A system is *scalable* if it is able to handle growing amounts of work in an acceptable manner (adapted from Wikipedia)
  - Desired system properties (such as performance) are "acceptable" functions of system size *n*
- We consider systems that consist of *n* equivalent nodes connected through a communication network
  - Ideally, performance (number of operations / second) *p(n)* = O*(n)*, where *n* increases as work increases
  - May not be achievable because of an inherent bottleneck: nodes need to communicate and each message needs to choose its destination, which introduces a logarithmic factor log*(n)* per message
- For many useful tasks, with proper design there are few messages, they have small delay, and they are rarely on the critical path, so O(n) is often achievable

What is Elasticity? (The Mind of Palador)

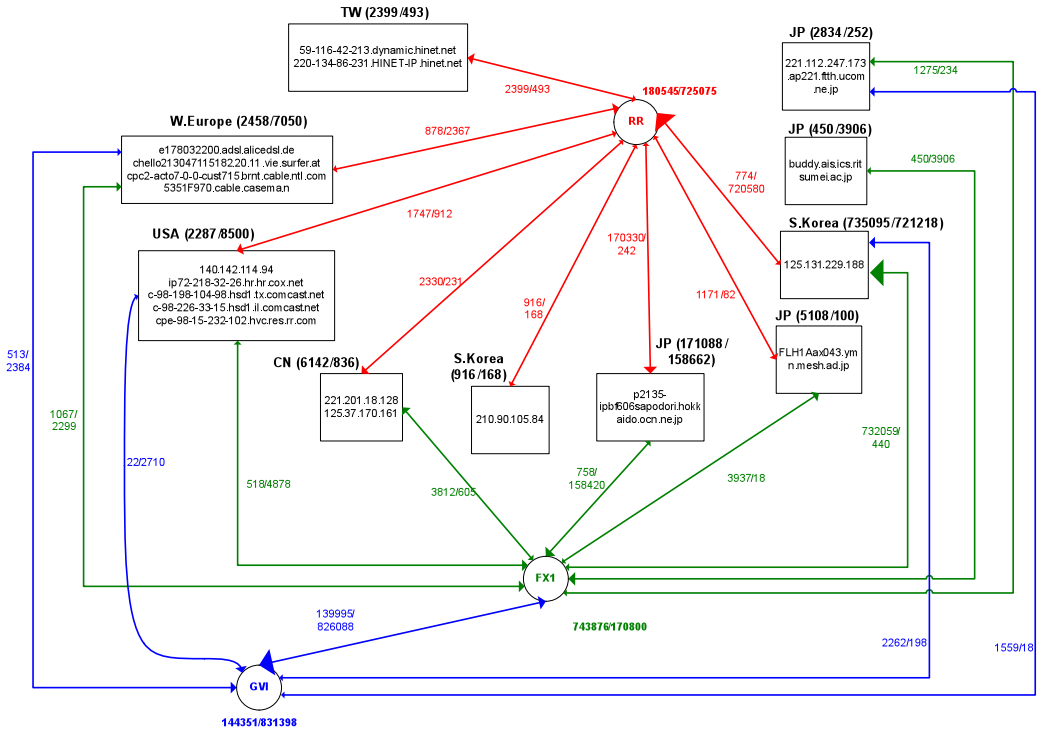
- "Last came one of the strange beings from the system of Palador. It was nameless, like all its kind, for it possessed no identity of its own, being merely a mobile but still dependent cell in the consciousness of its race. Though it and its fellows had long been scattered over the galaxy in the exploration of countless worlds, some unknown link still bound them together as inexorably as the living cells in a human body."
- "In moments of crisis, the single units comprising the Paladorian mind could link together in an organization no less close than that of any physical brain. At such moments they formed an intellect more powerful than any other in the Universe. All ordinary problems could be solved by a few hundred or thousand units. Very rarely, millions would be needed, and on two historic occasions the billions of cells of the entire Paladorian consciousness had been welded together to deal with emergencies that threatened the race. The mind of Palador was one of the greatest mental resources of the Universe; its full force was seldom required, but the knowledge that it was available was supremely comforting to other races."

- From the short story "Rescue Party" by Sir Arthur C. Clarke. First published in *Astounding Science Fiction* in May 1946. Written in March 1945 while Clarke was in the Royal Air Force. It is the first story that Clarke sold. Many of the themes in this story recur in Clarke's later work.

Peer-to-Peer Versus Cloud

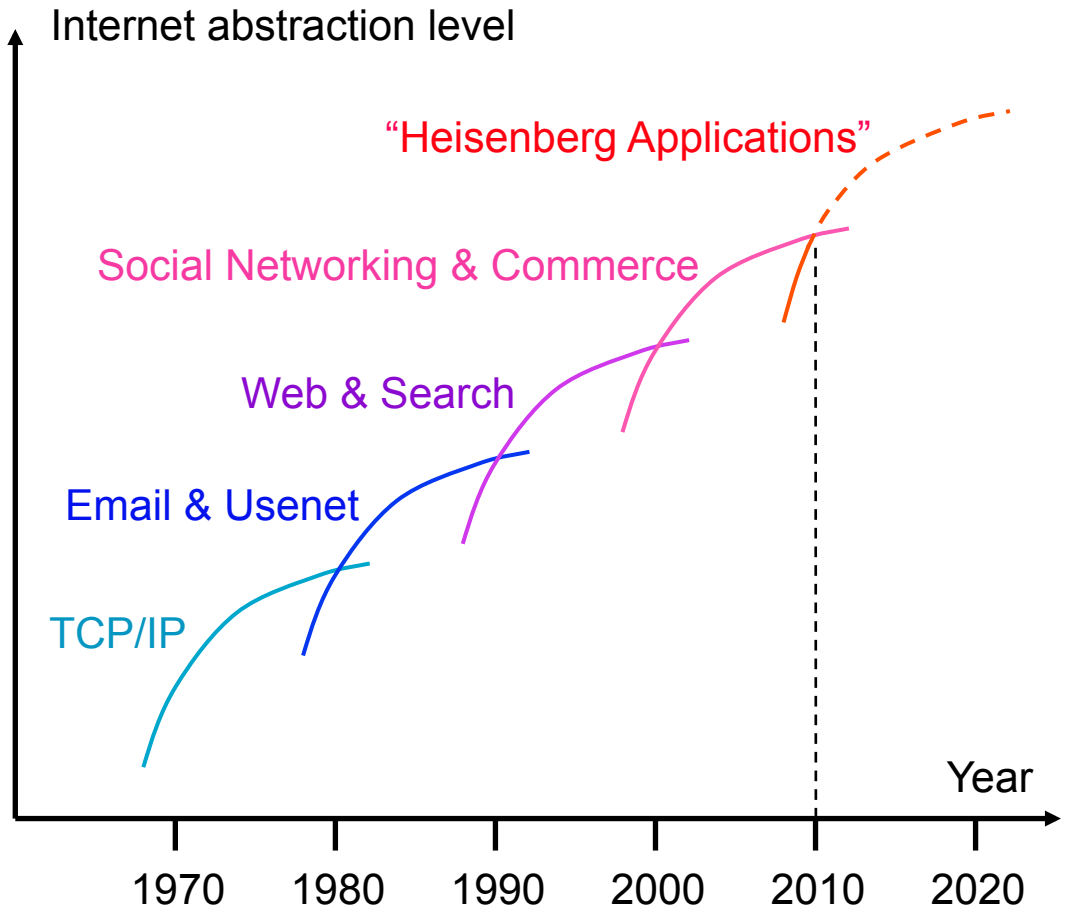
Simple example of Skype P2P routing

Nodes involved in 3-way conference call (FX1, GVI inside NUS, RR outside NUS)



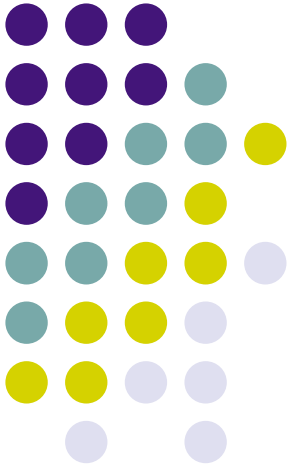
- Both P2P and cloud computing are scalable, but there is a fundamental difference between them
- Suppose Skype would like to add real-time language translation ability to its phone connections
  - Skype is based on a dynamic peer-to-peer architecture
  - Real-time language translation needs elasticity: huge resources (data and computation), but just for the person calling
- It can't be done on Skype's own P2P architecture because it's not elastic
  - The resources are just not there
  - It needs to be hosted on a cloud, as an extension of the P2P structure

The “Next Internet Revolution”



- The Internet has gone through four revolutions since its inception
  - Each revolution takes about ten years to be internalized
  - Old timers like me saw many of them (I started using it in 1983)
- We are now on the brink of a fifth revolution fueled by elasticity and based on a combination of cloud computing and data-intensive algorithms
  - Applications that use massive resources in short bursts, at a constant cost

The First Law (Novelty at Each Scale)



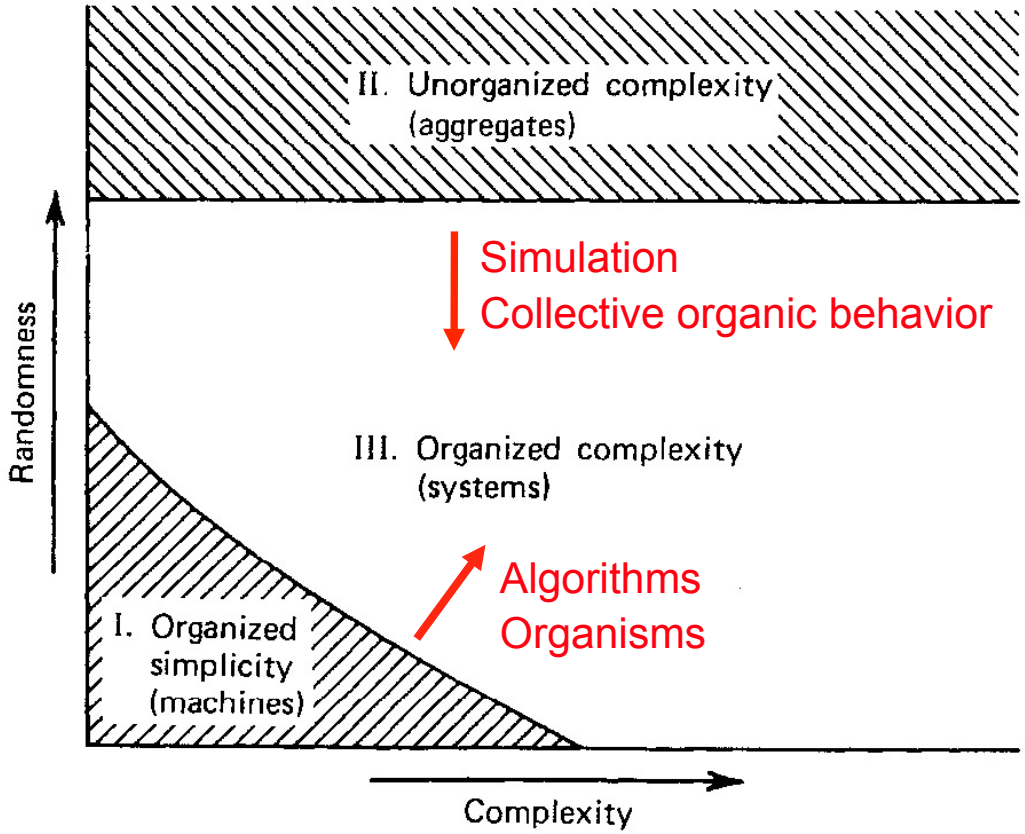
The First Law of Scalability

- At each new scale, the situation changes…

Sam Spade: "Ten thousand? We were talking about a lot more money than this."
Kasper Gutman: "Yes, sir, we were, but this is genuine coin of the realm. With a dollar of this, you can buy ten dollars of talk."
– The Maltese Falcon

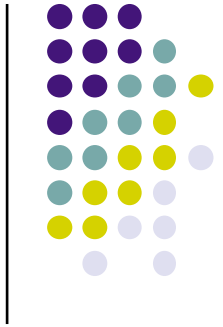
  - It's like physics: at each higher energy level, new physics appears
    - **No problem is ever solved for all scales** (despite claims to the contrary)
- It's a basic law of scalability that even physics cannot get around
  - In large systems, we see this every day
    - Not just computing systems, but any kind of system that can get big, e.g., organizations, skyscrapers, etc., needs new ideas at each level of scale
    - Biological systems take the lead in complexity and the more we look the more we find (e.g., see [Michal 1999] *Atlas of Biochemical Pathways*)
    - Computing systems take the lead for man-made systems
- Let's see what happens when we scale up…

New Scales and New Worlds



- This diagram is adapted from [Weinberg 1975] *An Introduction to General Systems Thinking*
- The disciplines of computing (invention) and biology (discovery) are pushing the boundaries of the two shaded areas inwards
- We are barely starting to investigate the surprising and novel phenomena in the white area

Alps Viewed From Space



- This amazing sight was never seen by humans until spaceflight was invented
- But it has always existed!
  - Nature obeys the First Law

Scalability in Nature

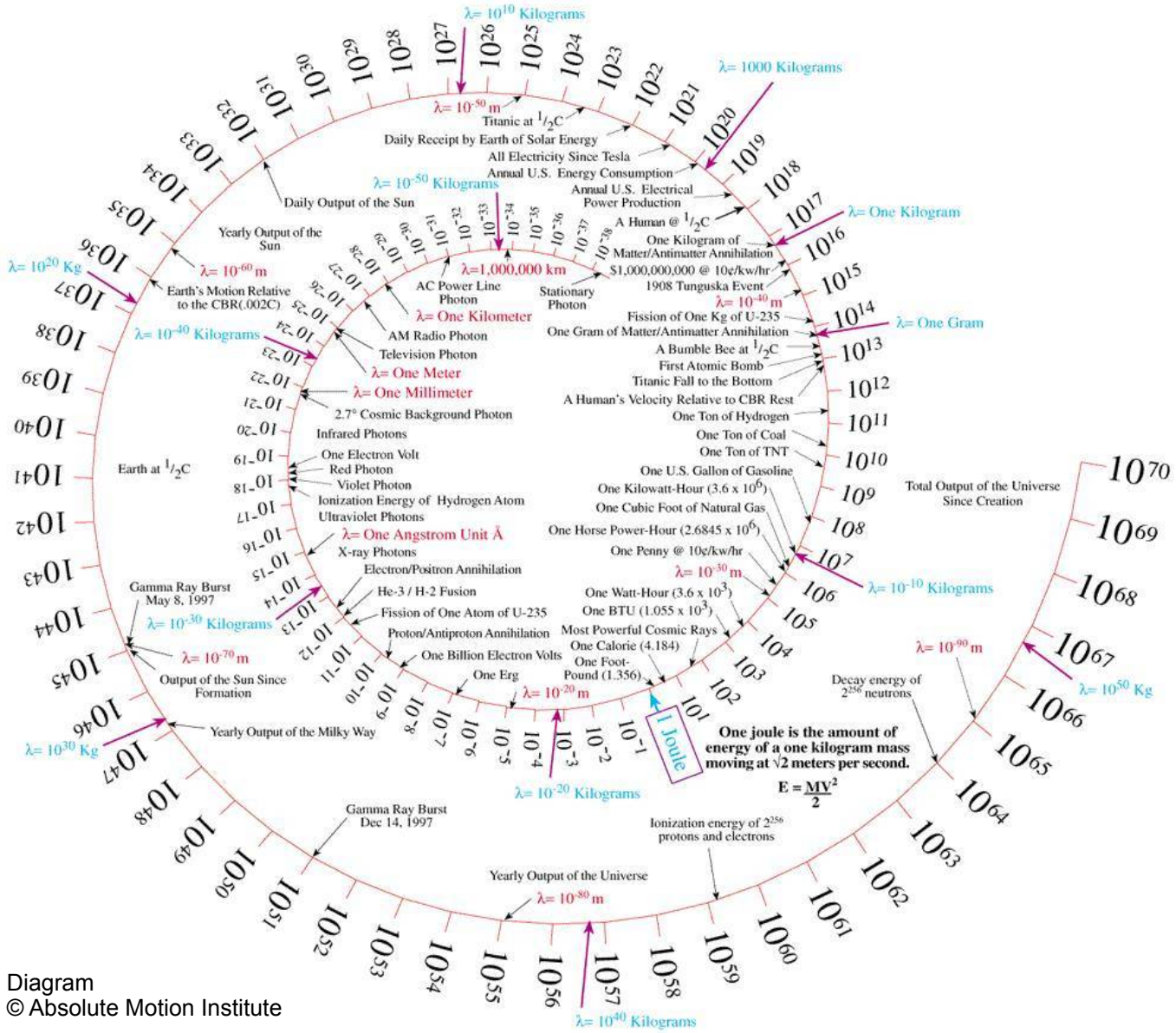
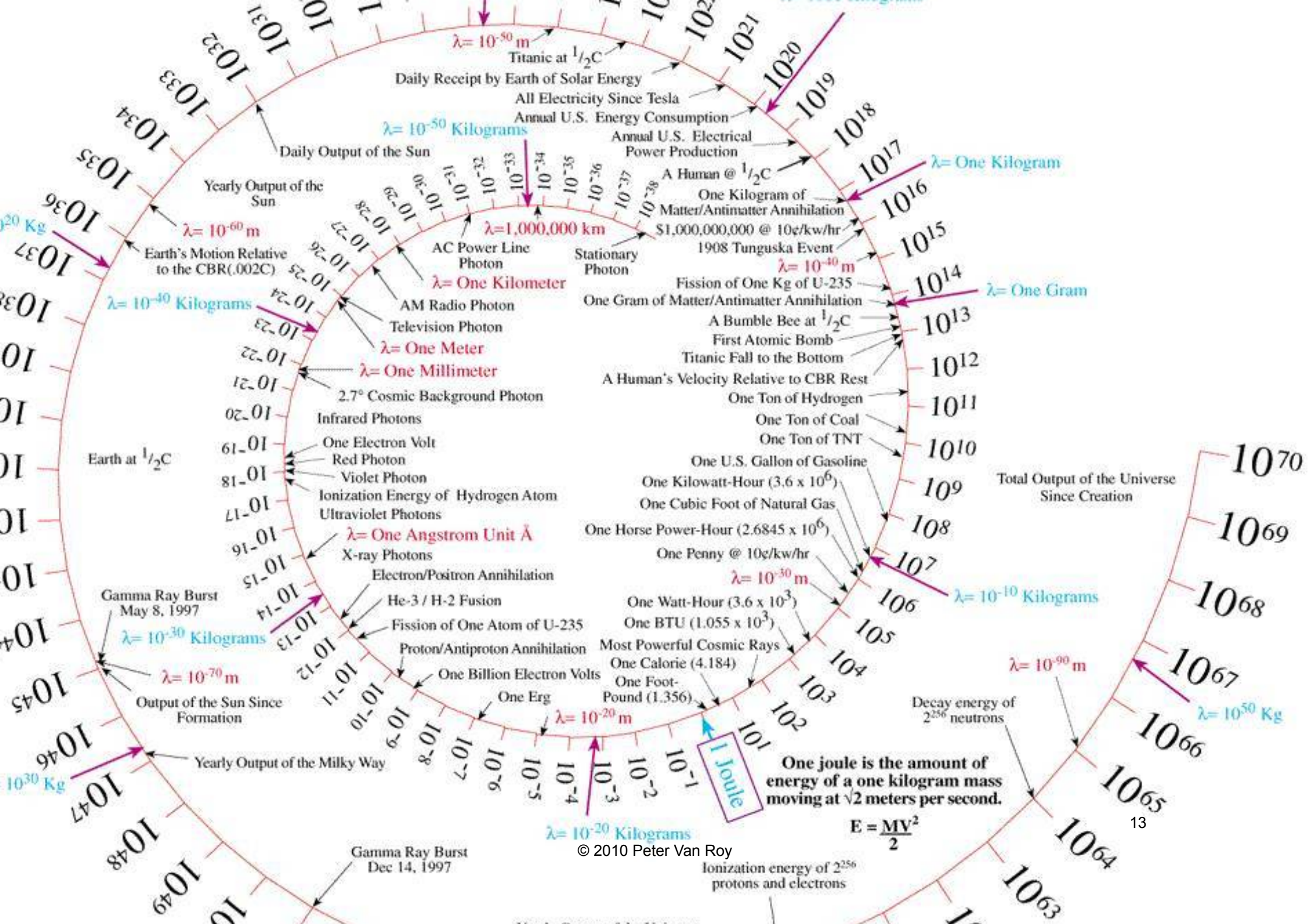


Diagram
© Absolute Motion Institute

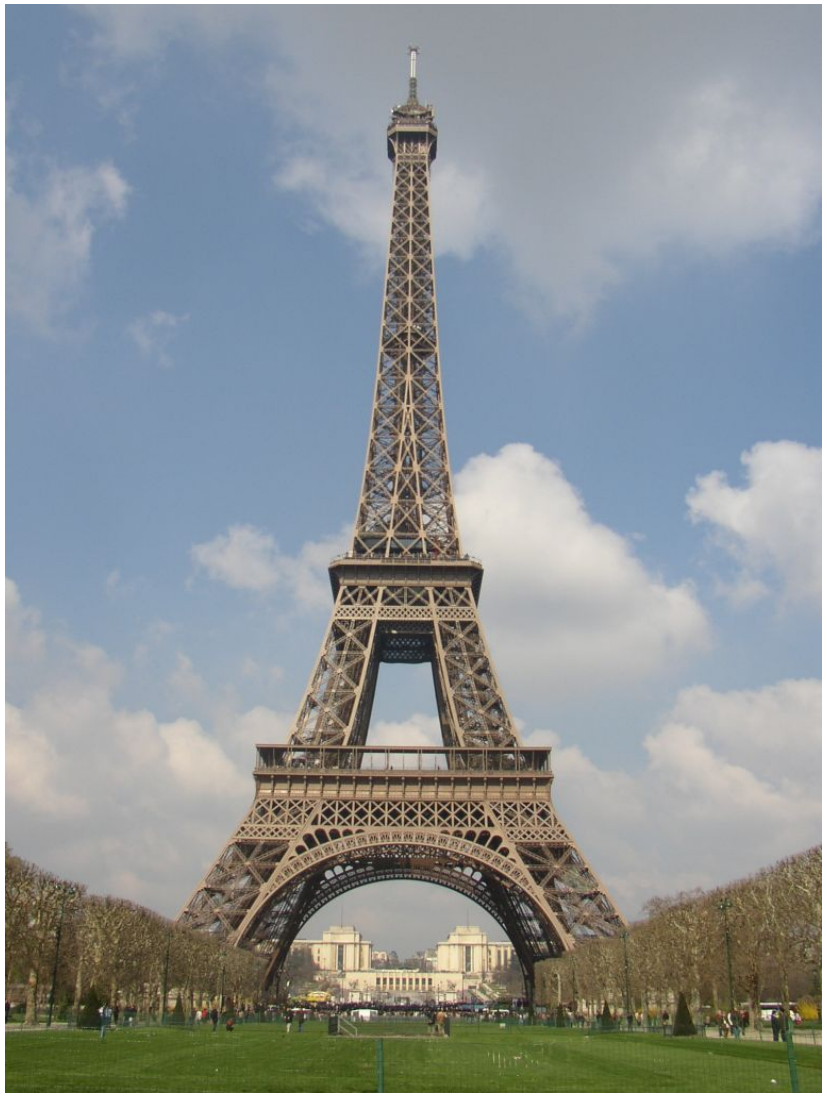
- Energy is a basic component of the universe
- Energy obeys conservation and linear superposition properties
- We observe 100 orders of magnitude in energy levels
  - From a stationary photon to the total output of the universe since creation
- Something new and interesting happens at every energy level



Nazca Figures

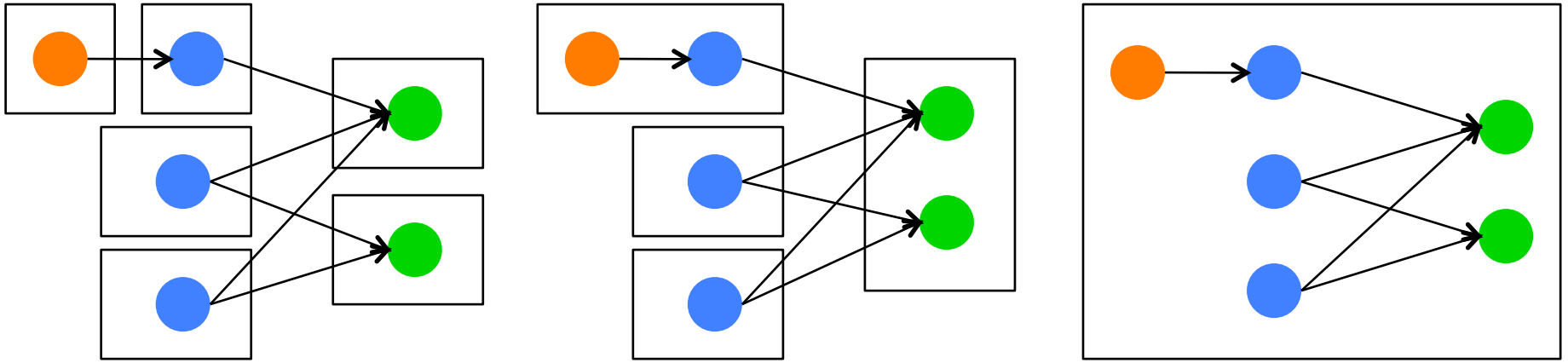
- Humans strive to obey the First Law too
  - These giant figures can only be seen from the sky: intended for the gods?

Striving for the First Law



- Successful complex structures built by humans are successful precisely because they obey the essential laws of complexity
- It is therefore worthwhile to try to understand them in a scientific way

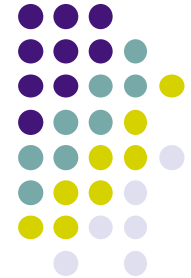
Scalability and Transparent Distribution (A Personal Experience)



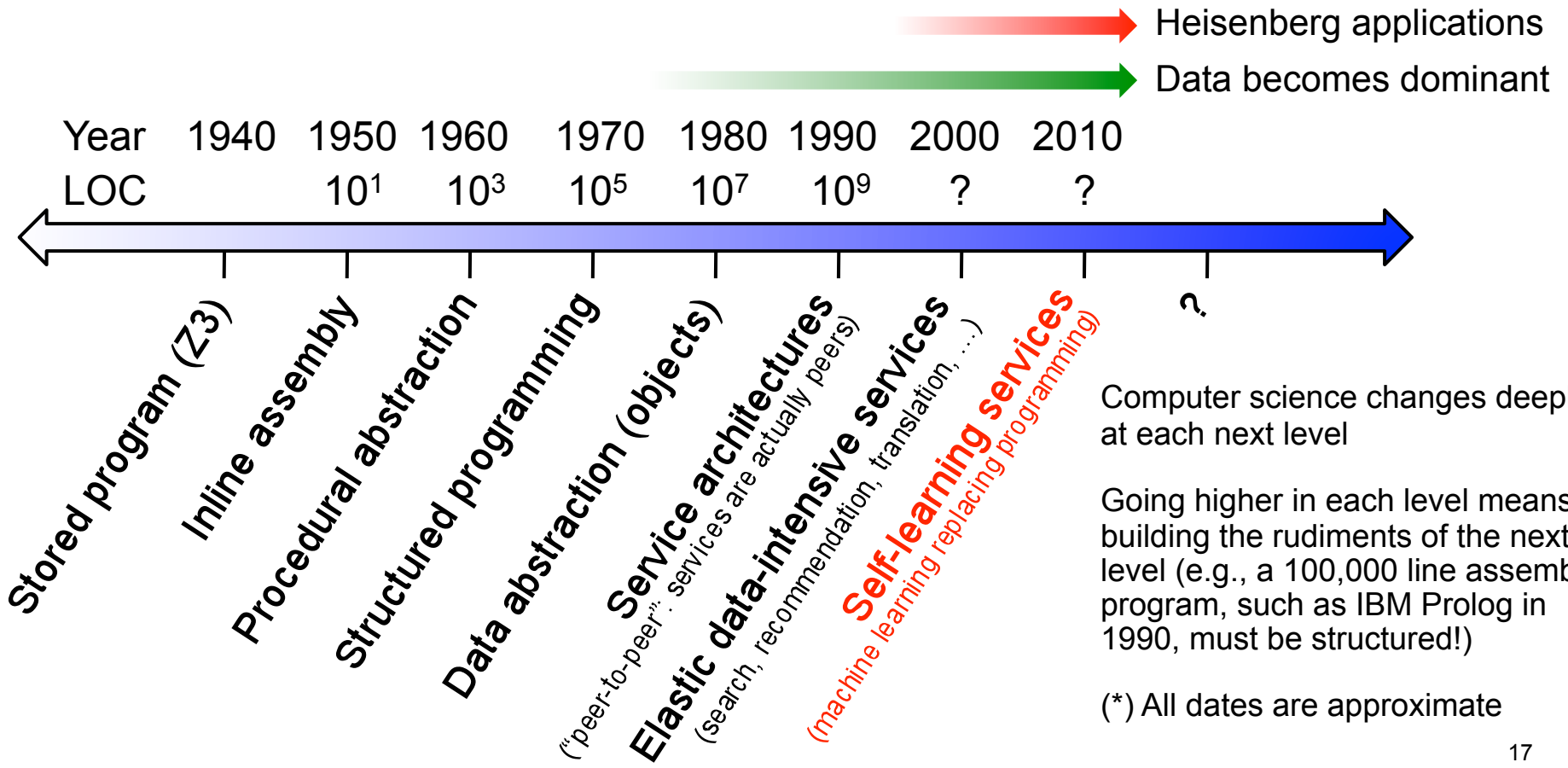
1 pipe, 3 clients, 2 servers on six machines → Same code with another distribution structure → Same on one machine (during development)

- Goal: make the accidental complexity of distributed programming disappear, leaving only the essential complexity
- Achieved by the Mozart system in 1999 (www.mozart-oz.org)
- But the First Law is not so easily vanquished: beyond ~10 machines, the application structure needs to change!

Scalability in Programming



Program complexity timeline (*)



Computer science changes deeply at each next level

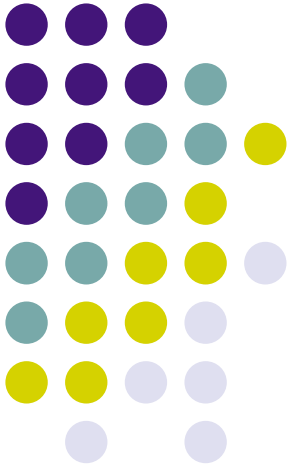
Going higher in each level means building the rudiments of the next level (e.g., a 100,000 line assembly program, such as IBM Prolog in 1990, must be structured!)

(*) All dates are approximate

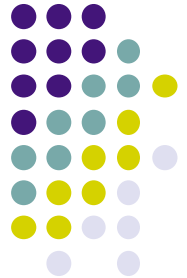
A Glimpse of the Future…

- Today's Internet…
  - Internet ≈ 800,000,000 hosts (2010, www.isc.org); biggest cloud ≈ 1,000,000 hosts (2010, Google)
  - Even the biggest clouds cannot meet the demand
  - Organizations will build clouds of different sizes
    - All clouds will be elastic, limited only by their size
    - Pressure to increase elasticity will cause them to federate (peer-to-peer clouds)
- The future Internet will consist only of clouds
  - The word "cloud" will cease to have special meaning
  - Virtualization and elasticity will be omnipresent
- It will be elastic, data-dominant, and self-learning
  - Elasticity will be used at all scales
  - Programs will use learning to improve themselves
  - Typical example: real-time audio language translation

Elastic Computing and Heisenberg Applications

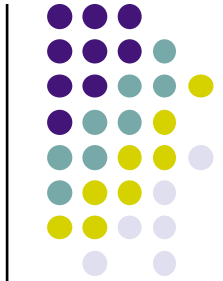


Elastic Computing



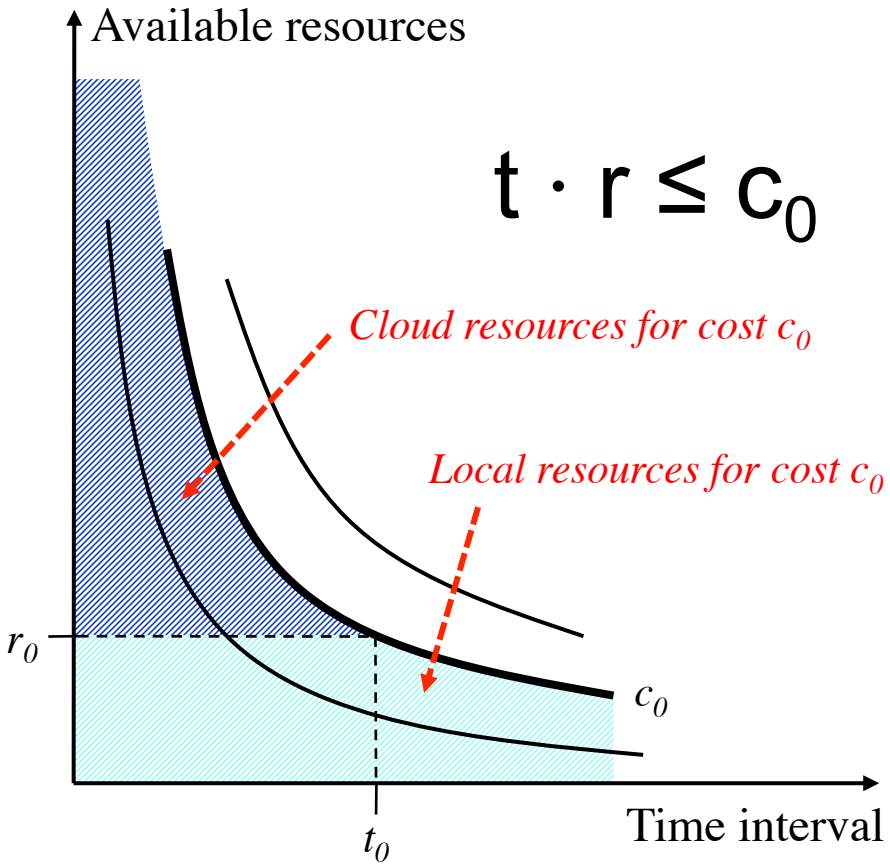
- Two main infrastructures for scalable computing
  - Peer-to-peer: use of client machines (my current expertise)
  - Cloud-based: use of datacenters (my future research)
- Cloud is elastic; peer-to-peer is not
  - Elasticity: the ability to scale resource usage up and down rapidly according to instantaneous demand
  - Elasticity is a new property that did not exist before clouds
- Elasticity makes possible Heisenberg applications
  - Applications that use enormous computational and storage resources for short times, but at constant (low) cost
  - A new kind of application that did not exist before clouds

Computational Heisenberg Principle (1)



- A cloud has two key properties:
  - Pay per use: pay only for the resources actually used
  - Elasticity: ability to scale resource usage up (and down) rapidly
- For a fixed cost, as the time interval decreases more resources can be made available:
  - **For a given maximum cost, the product of resource amount and usage time is less than a constant**
  - Analogy with Heisenberg's Uncertainty Principle in physics: the product of uncertainty in time and uncertainty in energy is equal to (or greater than) a constant. This increases the probability of events that use arbitrarily high energies if the time period is short enough. As long as the high energies are less than the uncertainty, then they are allowed!
    - This is a property of the system itself, not a limitation of measurement!
    - $\Delta t \cdot \Delta E = c$ and $t_{allow} \leq \Delta t$ and $E_{allow} \leq \Delta E$ implies $t_{allow} \cdot E_{allow} \leq c$
- This opens the door to new applications that could not be done before

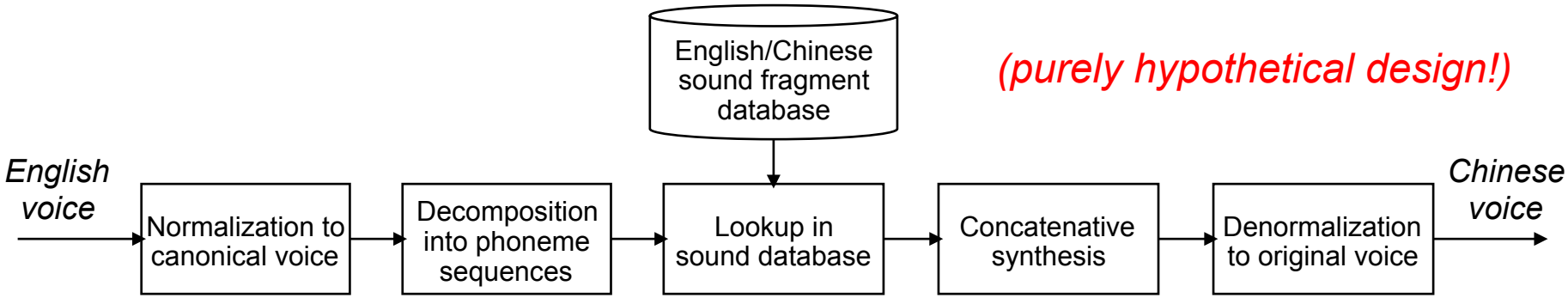
Computational Heisenberg Principle (2)



- For given fixed resource cost c_0, what kinds of applications can run?
- Before clouds: all applications lived in **light blue area** which gives local resources for maximum cost c_0 ($r \leq r_0$)
- With clouds: **dark blue area** becomes available for the same cost ($r > r_0$)
- The dark blue area is the home of Heisenberg applications
  - Like a data-intensive application combined with machine learning techniques

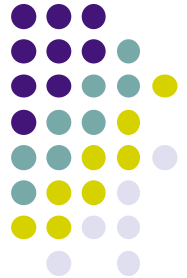
A Heisenberg Application (1): Real-Time Voice Translation

- The pieces of this application already exist; for example the IRCAM research institute has implemented many of them
- It requires combining domain knowledge (in sound and language) with an enormous sound fragment database, hosted on a cloud



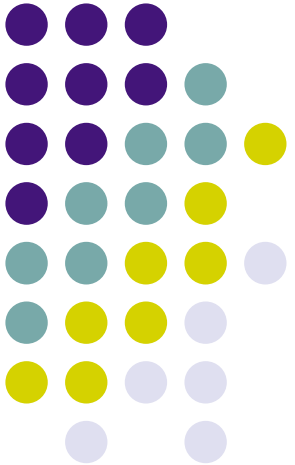
- Performance will be gradually improved through feedback from bilingual speakers and speech recognition technology
- Franz Och, head of translation services at Google, announced recently that they are working on something similar (Feb. 10, 2010)

A Heisenberg Application (2): Ubiquitous Augmentation



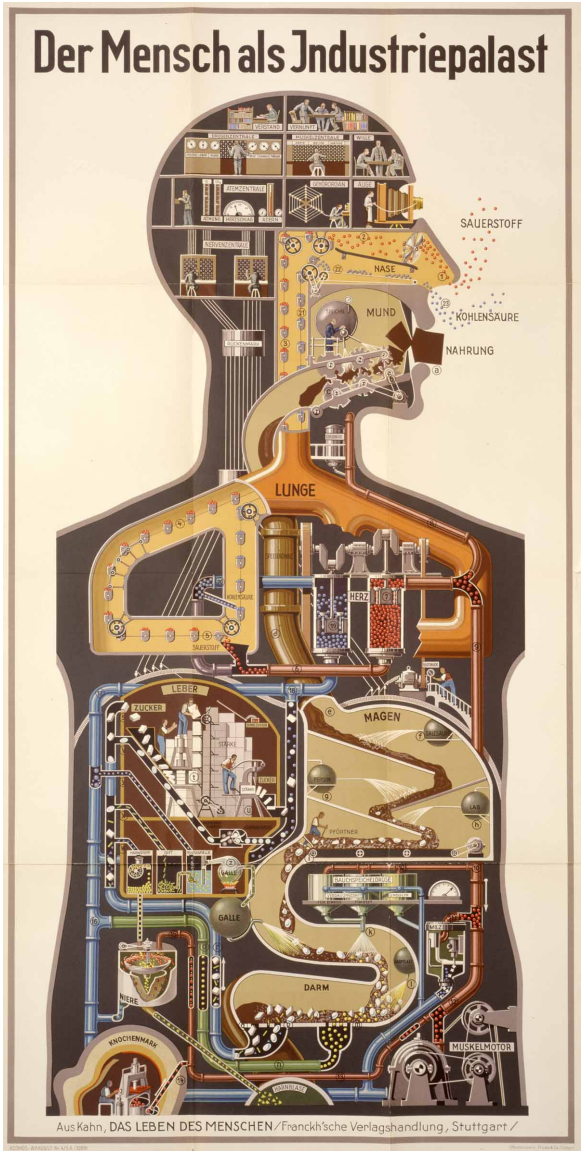
- Your sensory input will be "augmented" in real-time
  - Faces, objects, and names you see will be recognized
  - Selected relevant information will be given spontaneously
  - Foreign languages (text, audio, visual) will be translated
  - When doing an activity, you will be guided to do it expertly
  - When confronted with a problem, solutions will be suggested
- The augmentation will be good enough that it can be always enabled (it doesn't get in your way)
  - It will learn to mesh with your thinking processes productively
  - On the rare occasions that it is disabled, you will feel helpless
    - As if half of your brain just stopped working
    - Like today's Internet addictions, but much worse!

The Second Law (Only Local Control)



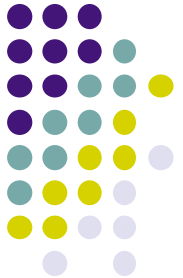
The Second Law

- In the limit of increasing scale, large systems have only local control
  - The system is concurrent and nondeterministic by default
  - Messages can take arbitrary time to arrive (asynchrony) and failures are hard to detect
  - Global control must be programmed and it can be very expensive or impossible
- Sometimes global control is just impossible
  - In a purely asynchronous system, consensus is impossible to achieve even if just one process can crash [FLP 1985]
  - Consensus can be achieved by adding synchrony or randomness, both of which may be too drastic
- But not all is bad news
  - Failures are local too
  - Some global control is possible, but less and less as the scale increases



A typical large system

The Internet is Treacherous As the Sea



Any application or algorithm

Asynchrony: many failure suspicions (algorithm's goal: don't crash!)

Synchrony: known delays (algorithm does its job)

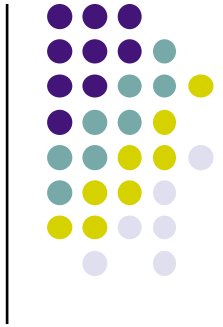
Clipper

Heavy storms at sea (clipper's goal: don't sink!)

Sea calms down (clipper reaches shore)

- Asynchronous system: messages take arbitrary (but finite) time
- Synchronous system: messages take fixed maximum time
- What about the Internet?
  - It starts out asynchronous (stormy) but eventually becomes synchronous (calm)
  - But we don't know how long this will take or what the message delays are!

Coping with Asynchrony



- The perennial dilemma
  - Asynchrony is natural and has higher performance
  - Synchrony is easier to use (each operation is finished before the next)
- Two extremes
  - Extreme 1: Push the asynchrony into the lower layers (e.g., libraries) for performance, and keep the user layers synchronous
  - Extreme 2: Rewire the user's brain to adjust to asynchrony (e.g., use notifications and keep work state external to user's brain)
    - Only works up to a point, because asynchrony is fundamentally harder for human conscious since it needs many context switches
  - Compromise: Use asynchrony by default and insert synchronous operations occasionally to simplify the system
  - Let us see how this works out in a real system…

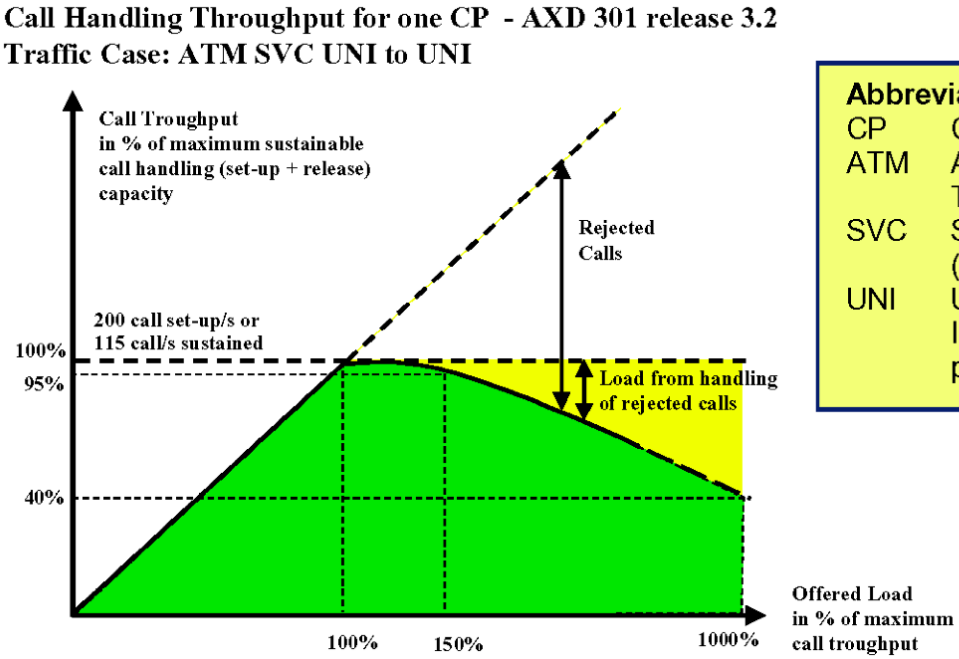
The Right Way and the Wrong Way

- Ericsson AXD 301 ATM Switch: >1 million lines of Erlang
- **Erlang**: Concurrent and independent by default, asynchronous messages, multi-agent programs

✓

- **Java**: Sequential and monolithic by default, synchronous RMI, shared-data programs

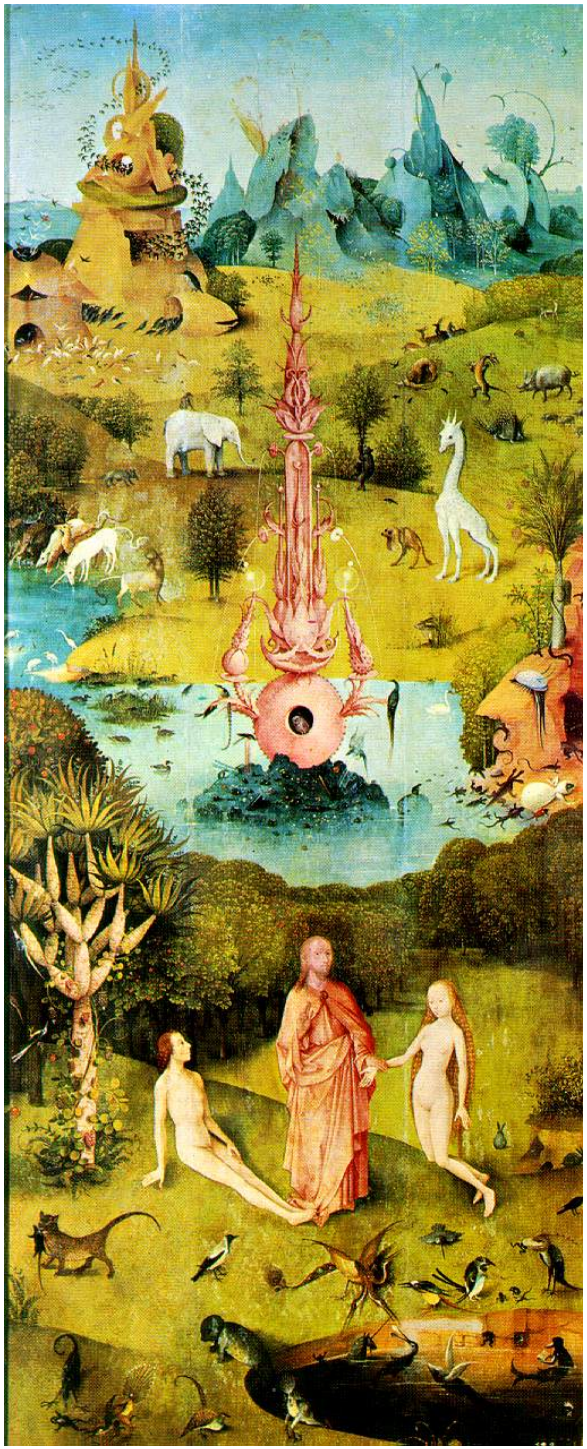
✗



Abbreviations:

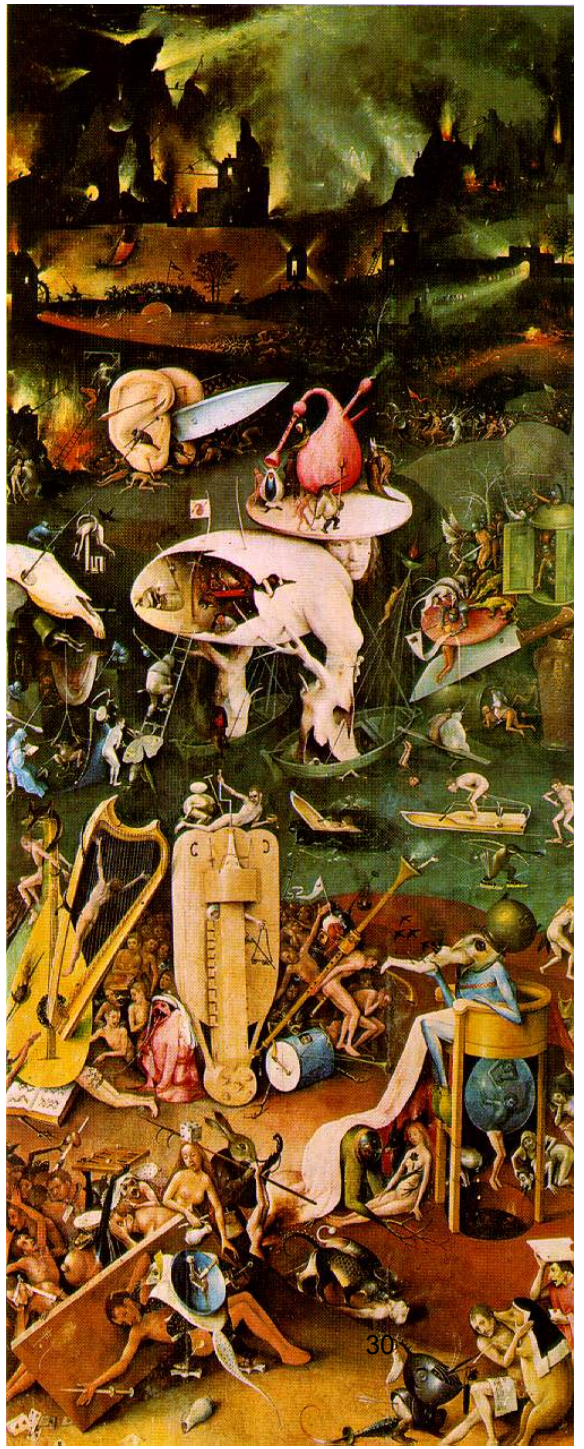
CP	Control Processor
ATM	Asynchronous Transfer Mode
SVC	Switched Virtual (ATM) Channel
UNI	User-Network Interface signaling protocol

- A heresy: object-oriented programming is irrelevant for the Internet!
  - Important: isolation, concurrency, asynchronous messages, higher-order programming
  - Unimportant: inheritance, classes, methods, UML diagrams, monitors

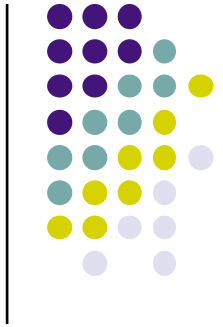


Nondeterminism

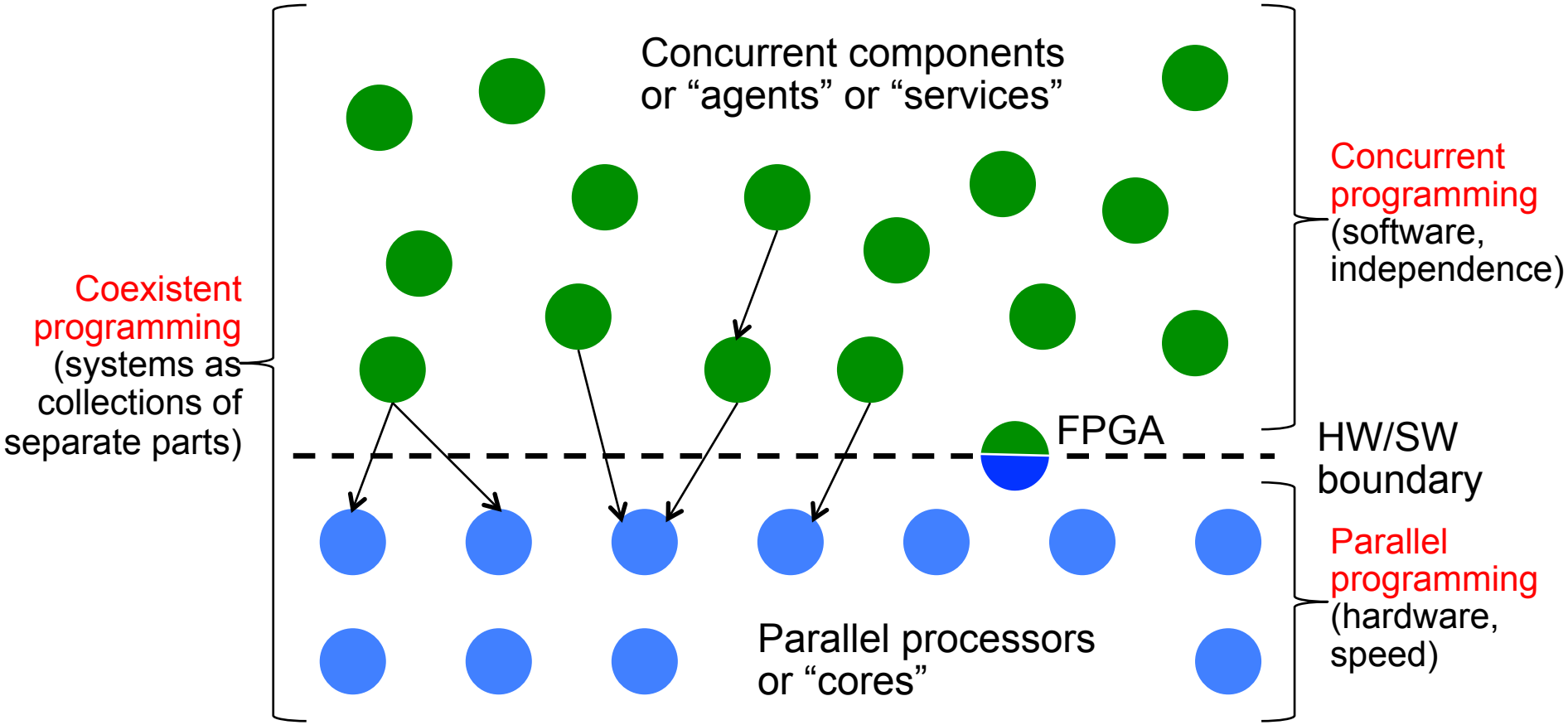
- The system makes choices
  - The user has almost no influence on these choices (message arrivals, process scheduling)
  - The choices may or may not affect the results
- Good nondeterminism: choice does not affect result (benign)
  - Choose path (to same destination)
  - Choose order of independent operations (client A or client B)
- Evil nondeterminism: choice affects result (race condition)
  - Choose destination
  - Choose order of dependent operations (credit or debit)



Concurrency



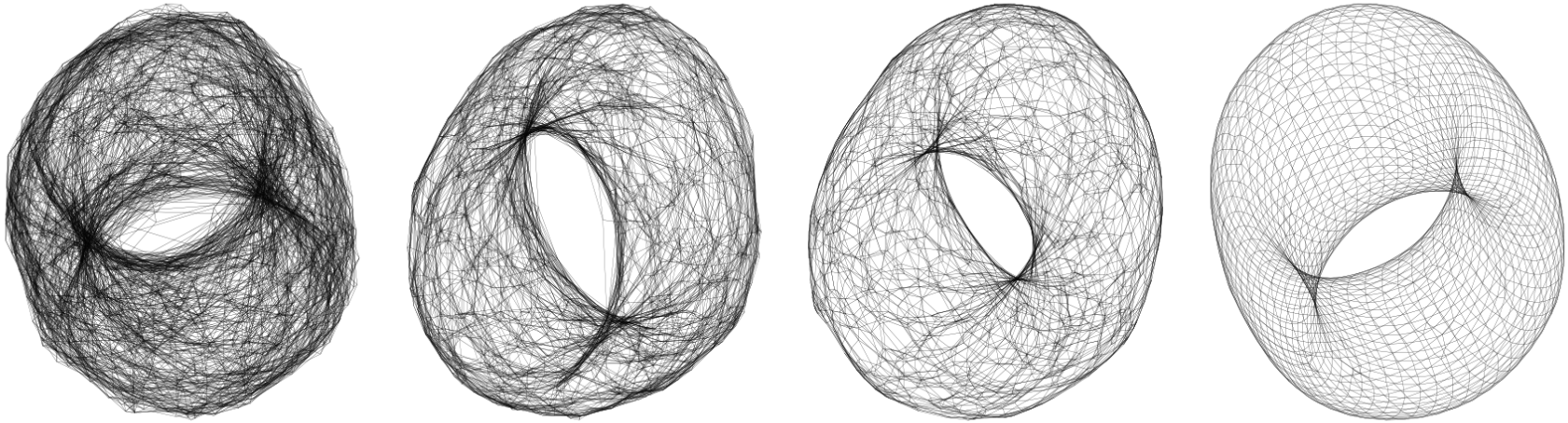
Concurrency is hard, so let's not fight it head-on, but use its power…



“Mostly” Independent Parts

- Large systems consist of mostly independent parts
  - Gas in a box: molecules mostly independent, occasional interaction when two molecules collide.
  - Peer-to-peer network: peers mostly independent, occasional interaction between neighbors only. Can provide efficient and robust communication and storage infrastructure (see later).
  - Gossip algorithm: nodes mostly independent, occasional interaction between random pairs. Can efficiently solve many global problems such as diffusion, search, aggregation, monitoring, and topology management.
  - Swarm intelligence: collaborative behavior among large numbers of simple agents (e.g., flocking and swarming). Each agent interacts with only a small number of neighbors.

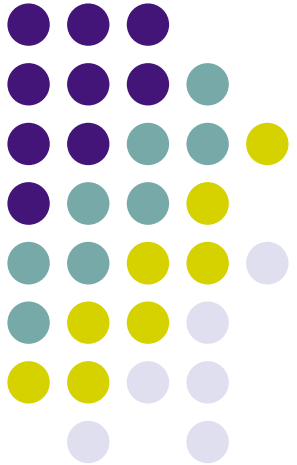
Gossip Algorithm: Topology Management



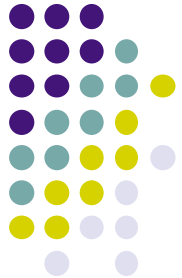
after 3 cycles | after 5 cycles | after 8 cycles | after 15 cycles

- The T-Man algorithm does topology management using a gossip algorithm
  - Each node periodically picks a random node and exchanges information with it
  - Each node has a ranking function that knows what distances nodes are supposed to have in the desired topology (i.e., a torus emerging from a random graph)
- The topology emerges in a few cycles (one cycle = one update per node)
- The algorithm is efficient, extremely robust, and can track changes

The Third Law (The CAP Theorem)

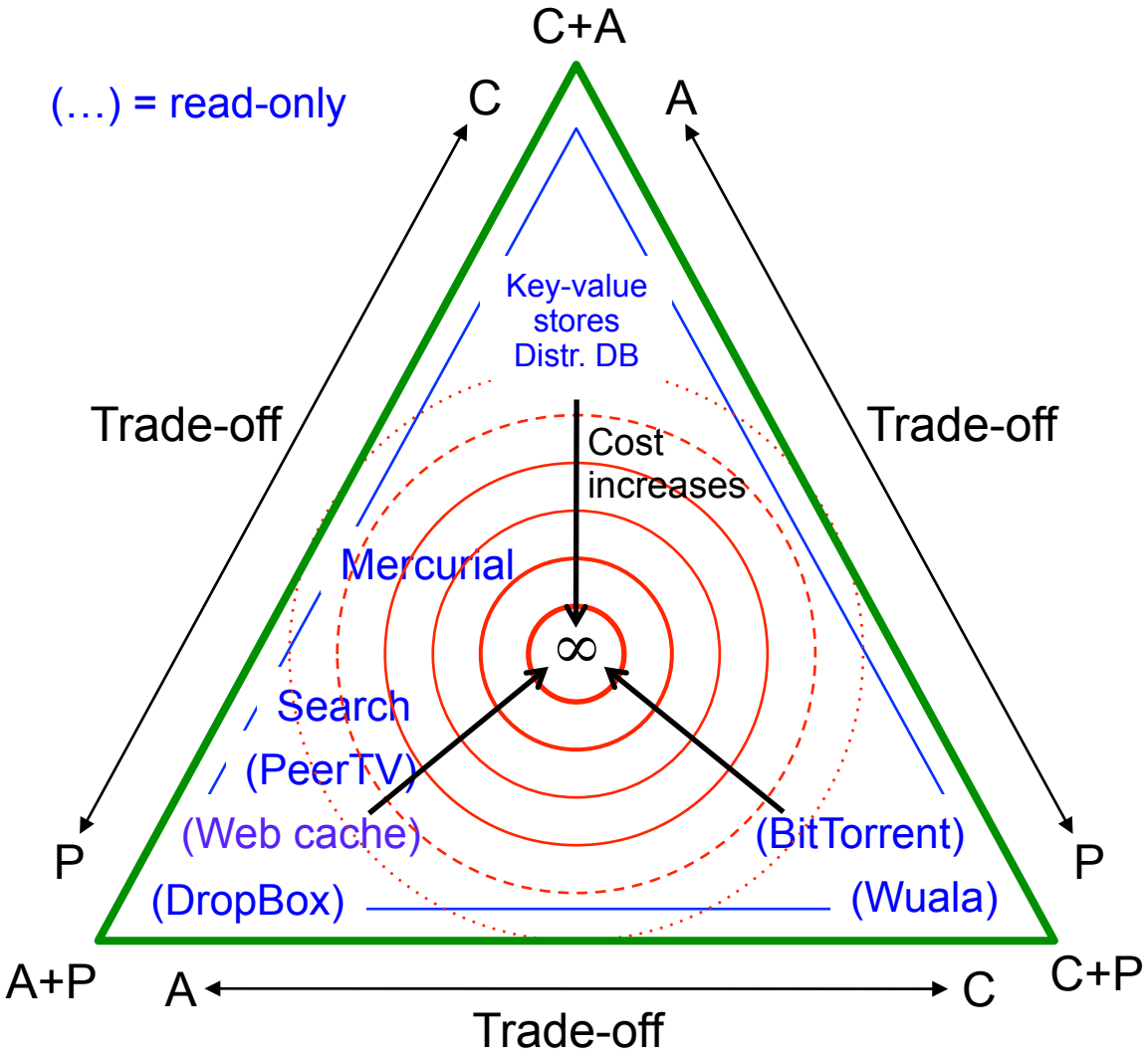


The Third Law



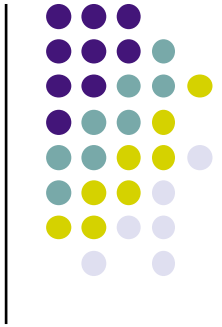
- The CAP theorem was conjectured by Eric Brewer at PODC in 2000 and proved by Seth Gilbert and Nancy Lynch in 2002
- For an asynchronous network, it is impossible to implement an object that guarantees the following properties in all fair executions:
  - Consistency: all operations are atomic (totally ordered)
  - Availability: every request eventually returns a result
  - Partition tolerance: any messages may be lost
- The CAP Theorem applies for all systems, at all levels of abstraction, and at all sizes
  - It can be applied in many places in the same system
  - The whole system is a rainbow of interacting instances of CAP

The CAP Triangle



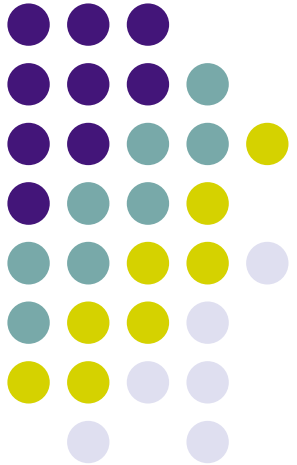
- The CAP space hugs the edges of the triangle
  - Cost increases toward the center
  - The center itself is empty!
- All parts of the CAP space have their uses
- We have arranged some applications around the triangle according to perceived functionality
  - Very little systematic study has been done about navigating in this triangle

Designing with CAP

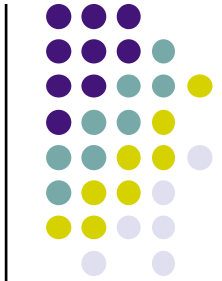


- C is hard to achieve (Second Law) → (P+A, no C) is the default
  - Consistency requires global coordination
- Avoid needing C if possible
  - We can achieve robustness (P) and performance (A)
    - DropBox and Web cache give P and A, but not C
    - Wuala and BitTorrent are read-only, achieve C easily
    - Mercurial is consistent if connected (C+A), but is still usable if disconnected (P+A)
- But if we really need C
  - Give up A → Waiting sometimes needed
  - Give up P → Fragile system
    - Distributed database guarantees C but will block if there is a partition
- We can have our cake and eat it too, if we pay the price
  - Highly reliable communication channels and fault tolerance
  - We get C and A, and we "seem" to get P as well (actually, we just have less partitions)
    - Scalaris, Beernet: peer-to-peer with majority consensus (Paxos) gives robustness
    - Cassandra: run on cloud, not peer-to-peer (does not support loose coupling)

Designing for Scalability

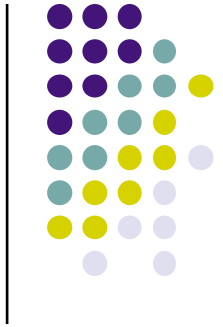


Design Rules for Scalability?



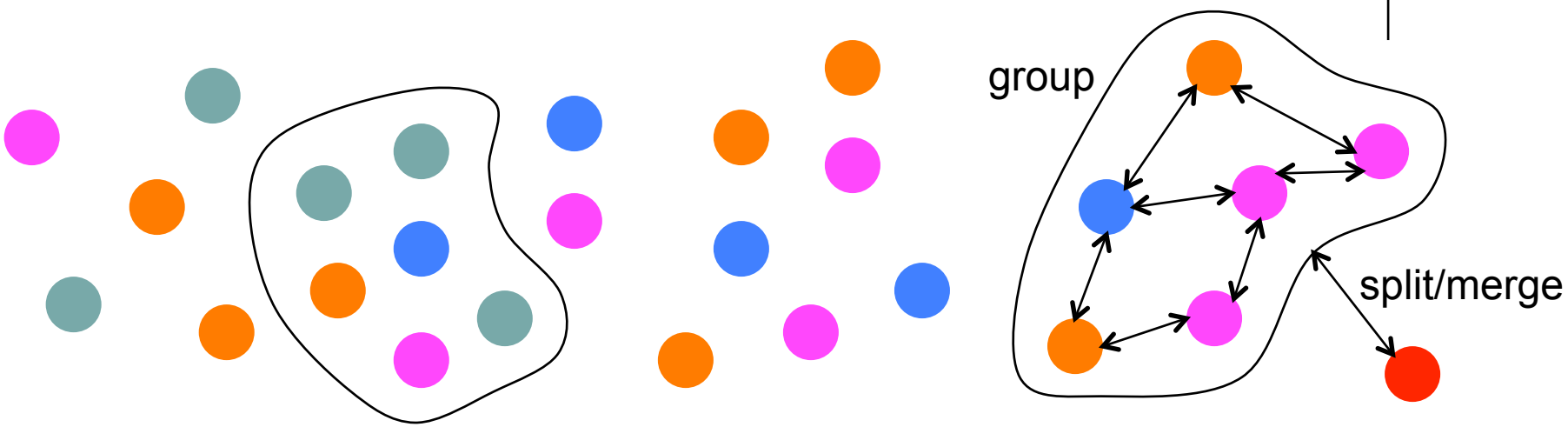
- How can we learn how to build scalable systems?
  - The First Law says new ideas are needed as the system grows
  - But finding new ideas requires blood, sweat, and tears
- Short-cut: study existing systems that work
  - Biological systems have already treaded this path and are suitably huge (see [Michal 1999] *Atlas of Biochemical Pathways*)
  - Some computing systems have treaded this path as well, especially Internet protocols and applications
- Learn lessons from both kinds of systems
  - And maybe come up with some general principles?

First Step: Decide on the Scale



- Centralized systems are much easier to design than decentralized systems
  - But the degree of centralization that's possible depends on the scale: larger scales support less centralization (Second Law)
    - LAN: centralized control
    - Internet: centralized address assignment, decentralized routing
    - Internet on the scale of the solar system
- Decide on the desired scale, and introduce the maximum possible centralization that's possible at that scale
  - Note that your design will not work for larger scales (First Law)

Second Step: Add Consistency

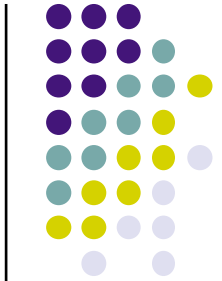


- Every scalable design starts as a decentralized system (P+A, no C)
  - A coexistent system of independent pieces (Second & Third Laws)
- Nodes occasionally interact (add some C) → collaboration, emergence
  - Split protocol: what happens when a node leaves a group (may be abrupt)
  - Merge protocol: what happens when a node joins a group
- Merge is based on data coherence and may need input from highest level
- Many examples: biology, peer-to-peer, map-reduce, gas/liquid/solid, …

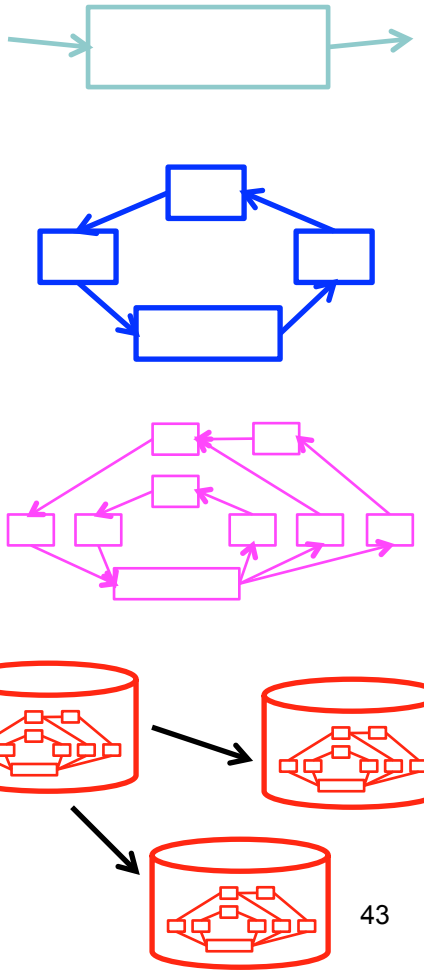
Third Step: General Design Principles

- We start with a decentralized system (P+A, no C)
  - The problem: how much C do we need and how do we add it?
- The rest of the talk explores how to add C
  - Human respiratory system (biology)
  - Decentralized transactional store (computing)
    - Scalaris and Beernet peer-to-peer structured overlay networks
  - More examples in the stub slides: TCP, hotel lobby, human endocrine system
- These examples motivate general design principles
  - We present two: complex components and phase behavior
- Main design principle:
  weakly interacting feedback structures

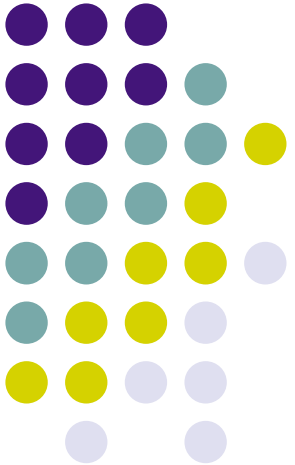
Weakly Interacting Feedback Structures



- Concurrent component
  - An active entity communicating with its neighbors through asynchronous messages (Second Law)
  - "Intelligence" concentrated in core components
- 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 have dependencies based on the operating conditions (Third Law)

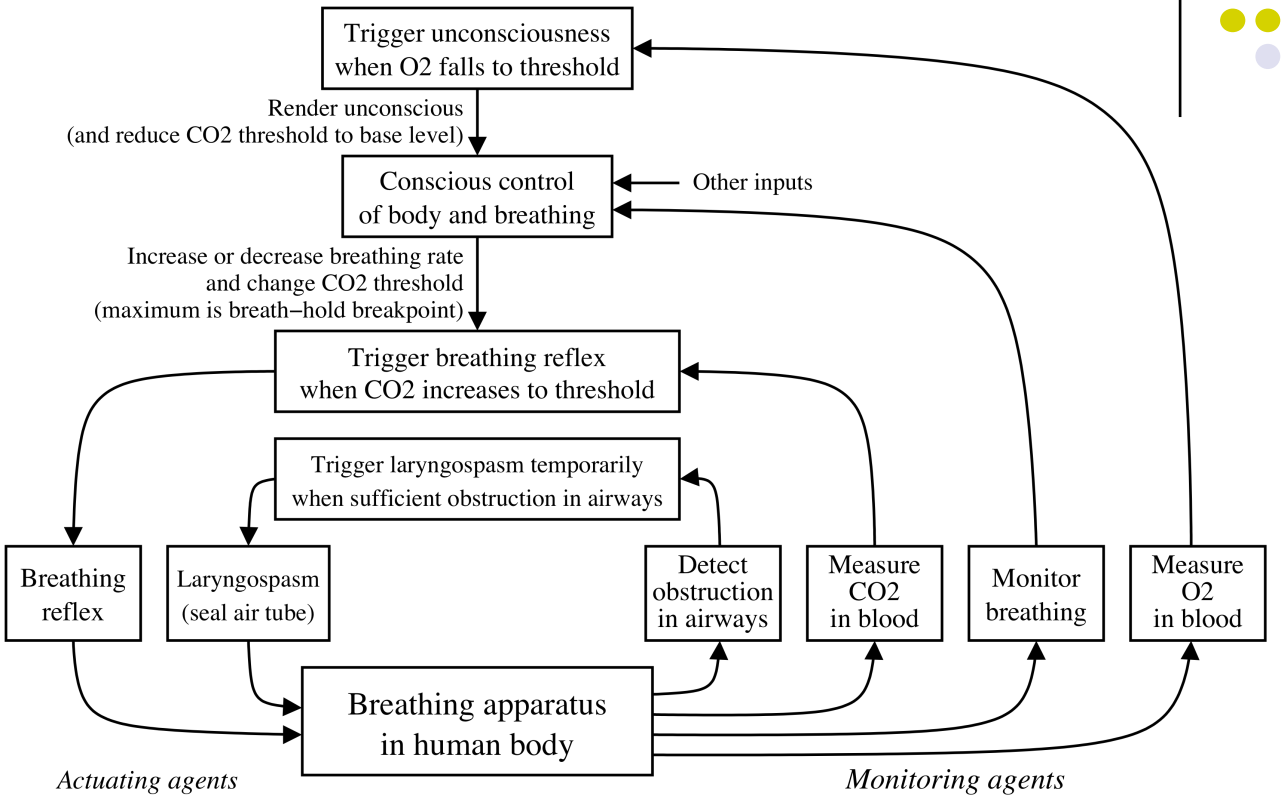


Human Respiratory System and Complex Components



Human Respiratory System

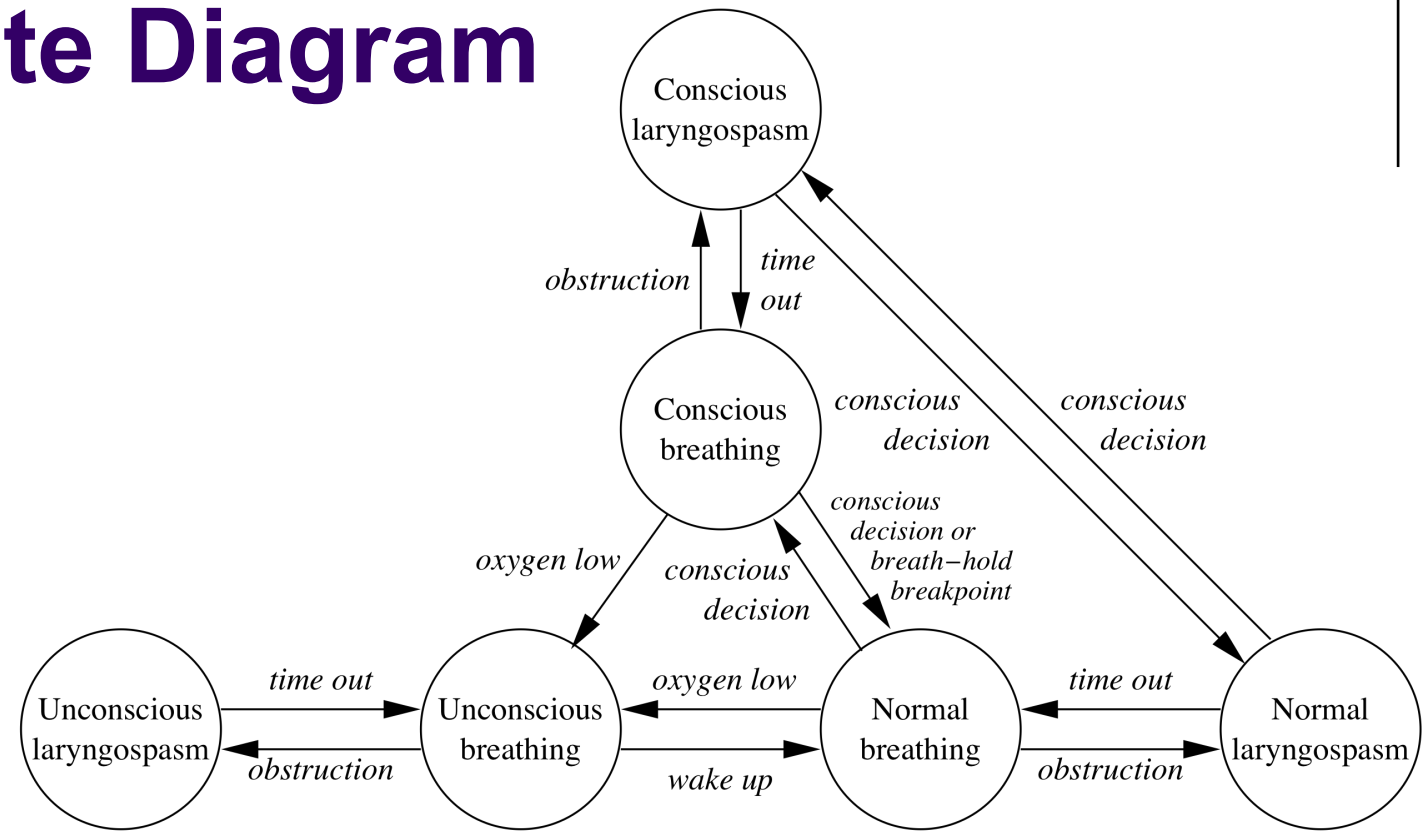
The operation of the human respiratory system is given as one feedback structure, inferred from a precise medical description of its behavior



Some design rules:

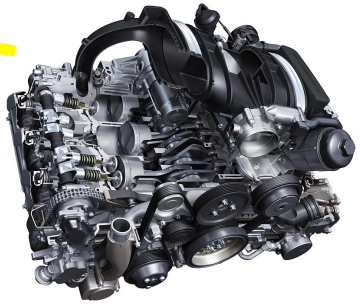
- Default behavior: rhythmic breathing reflex
- Complex component: conscious control can override and plan lifesaving actions
- Abstraction: conscious control does not need to know details of breathing reflex
- Fail-safe: conscious control can itself be overridden (falling unconscious)
- Time scales: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

State Diagram



- The human respiratory system can be seen as a state diagram
- Dominant subset = active subset of feedback loops = state
  - At any time, one subset is active, depending on operating conditions
  - Each subset corresponds to a state in the state diagram

Power is Built In, Not Added On

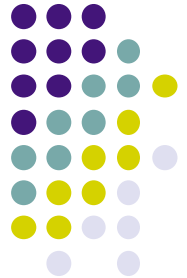


3.6-Liter Biturbo Motor with 353 kW (480 HP)

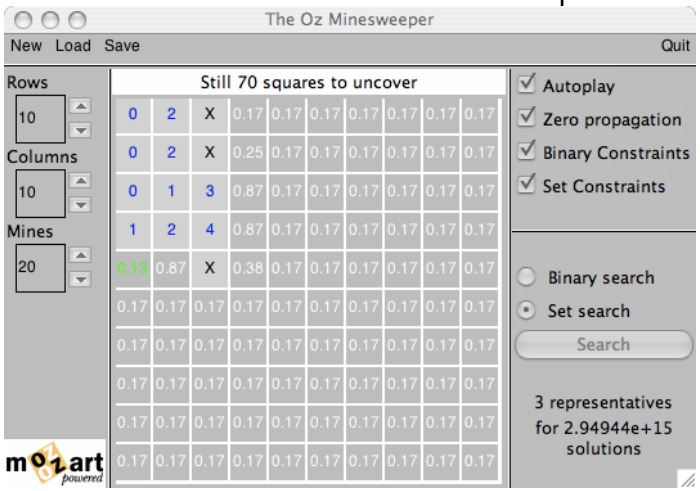
Porsche Carrera GT

- The power of a system depends on the strength of its complex components
  - The human respiratory system uses conscious control (e.g., to avoid drowning!)
  - Erlang OTP uses supervisor trees and a database to implement robustness
  - Scalaris uses Paxos consensus and replication to implement fast transactions
  - Google Search uses eigenvector calculation of the Web link matrix
  - What does *your* system use?

Some Complex Components



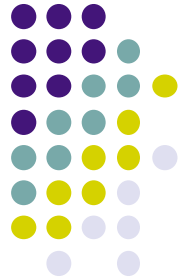
- Human intelligence
  - Main strength: adaptability (dynamic creation of new feedback loops)
- Program intelligence
  - Can easily go beyond human intelligence in many areas!
    - **Turing test is irrelevant**: complex components are already replacing humans in more and more areas
  - Minesweeper digital assistant: uses constraints (easy to program!)
  - Chess: uses alpha-beta search with heuristics
  - Compiler: translates human-readable program into executable form



More on Complex Components

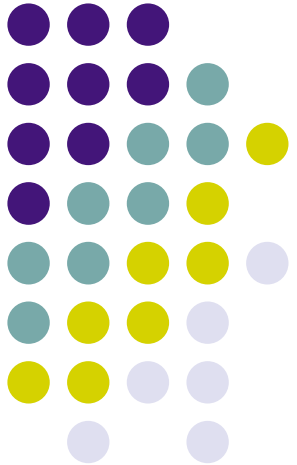
- Complex components *completely solve* a problem inside a specific (small) part of the space of system operating conditions (from the viewpoint of the rest of the system)
  - Conscious control, a chess program, and a compiler are extremely smart *within their operating space*
  - Outside of this space, they can be very stupid and should be inactive (on their own accord or forced)
- Complex components are *completely unpredictable* when viewed from the outside
  - If it were not so, they would not be needed!
  - They can be highly nonlinear and unstable; the rest of the system has to trust them (up to some hardwired fail-safe)

How Come Conscious Control is So Smart?



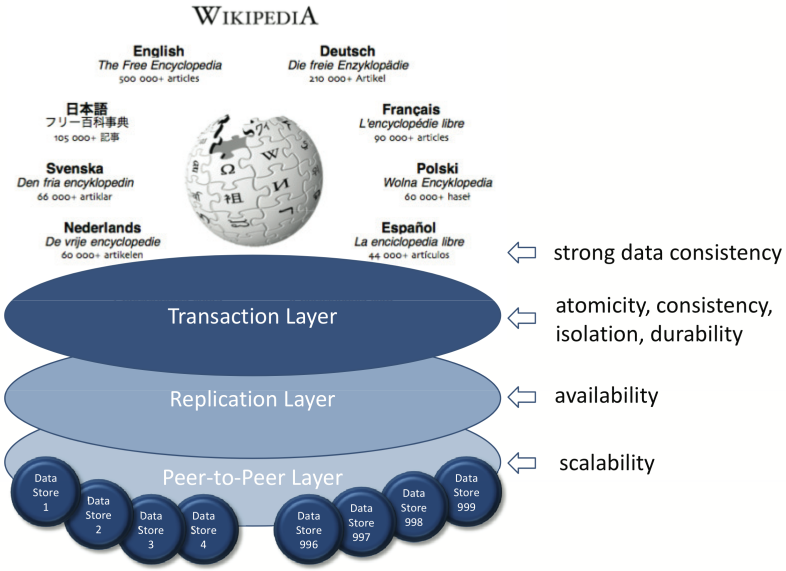
- Cognitive science and neurology try to understand why
  - The brain uses brute force, but in a very smart way
- Conscious control is a bricklayer: it continuously builds and organizes new components on top of existing components
  - This process is continuous from birth with compound interest effect, 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 cell")
  - 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 ("walking and chewing gum"), because of the two-lobed structure of the brain

Transactional Store and Phase Behavior



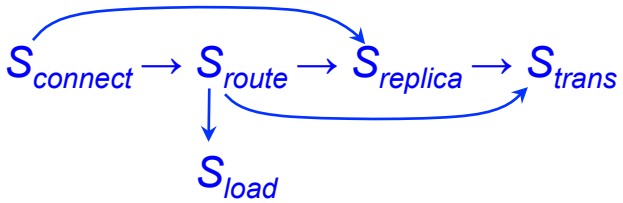
THE ADVENTURES OF SELFMAN

A Peer-to-Peer Key/ Value Store: Scalaris



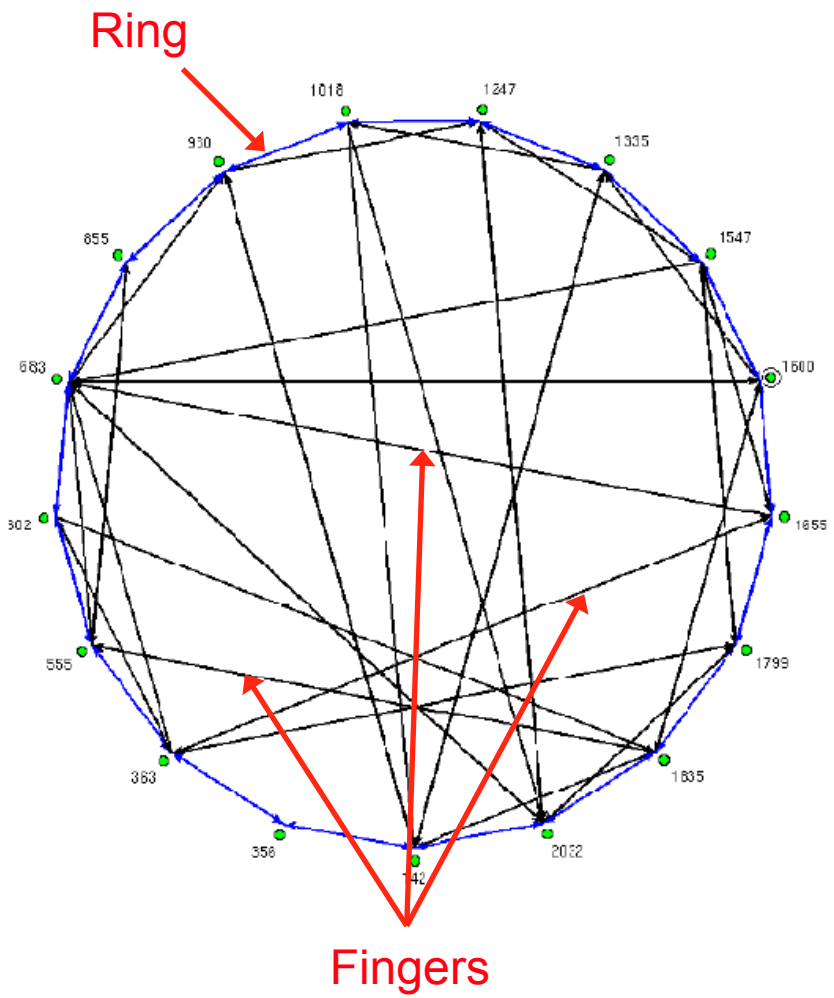
$S_{scalaris} = S_{key\text{-}value} \wedge S_{connect} \wedge S_{route} \wedge S_{load} \wedge S_{replica} \wedge S_{trans}$

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



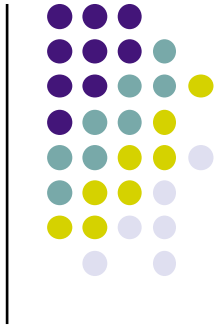
- Scalaris is a high-performance self-managing key/value store that provides transactions and is built on top of a structured overlay network
  - A major result of the European SELFMAN project (www.ist-selfman.org)
  - 4000 read-modify-write transactions per second on two dual-core Intel Xeon at 2.66 GHz
- Scalaris has five WIFS: connectivity management ($S_{connect}$), routing (S_{route}), load balancing (S_{load}), replica management ($S_{replica}$), and transaction management (S_{trans})

Structured Overlay Networks

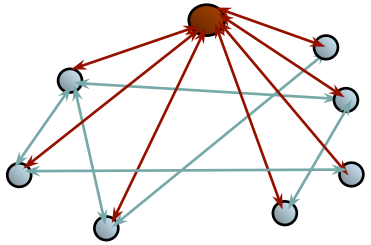


- Structured overlay networks are often based on a ring
  - By far the most popular structure, it has many variants and has been extensively studied
- Self organization is done at two levels:
  - The ring ensures connectivity: it must always exist despite node joins, leaves, and failures
  - The fingers provide efficient routing: they can be temporarily in an inconsistent state

Structured Overlay Networks: Inspired by Peer-to-Peer

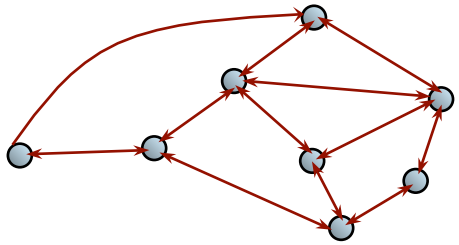


- Hybrid (client/server)
  - Napster



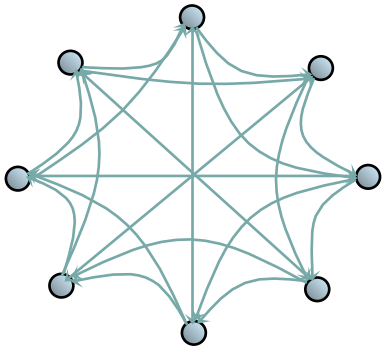
R = *N*-1 (hub)
R = 1 (others)
H = 1

- Unstructured overlay
  - Gnutella, Kazaa, Morpheus, Freenet, …
  - Uses flooding



R = ? (variable)
H = 1…7
(but no guarantee)

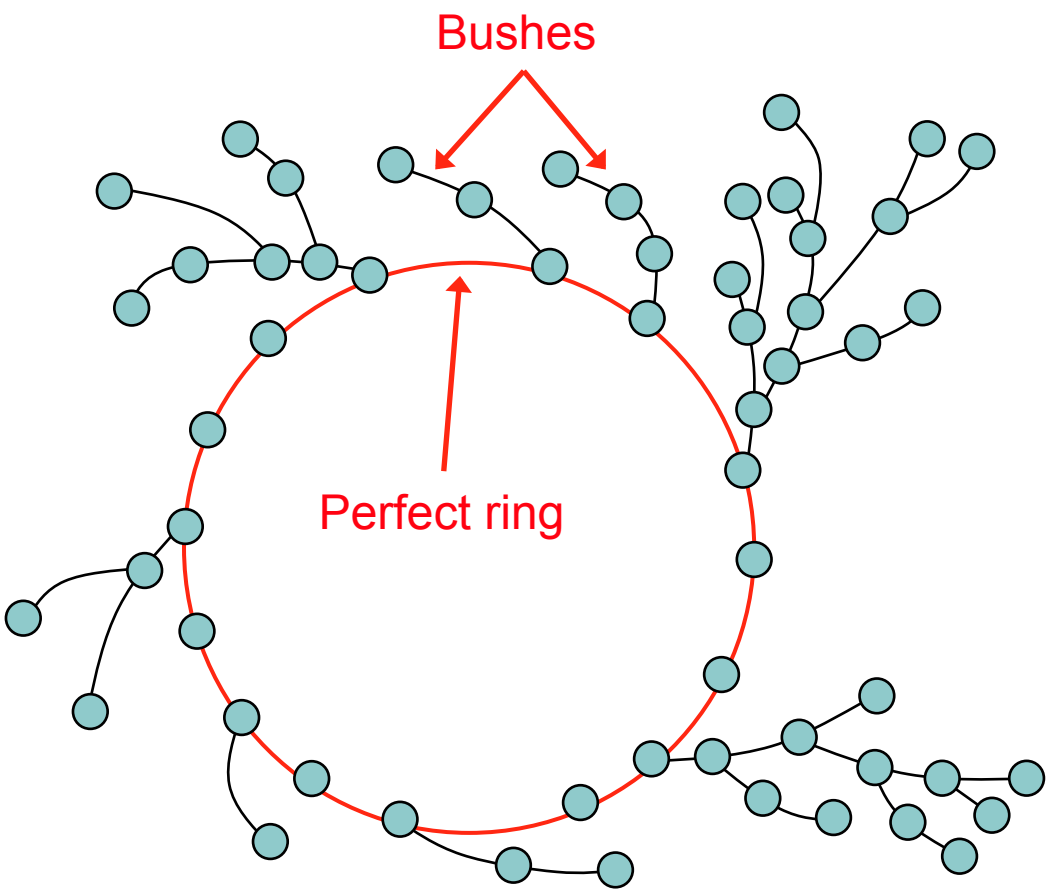
- Structured overlay
  - Exponential network
  - DHT (Distributed Hash Table), e.g., Chord, DKS, Scalaris, Beernet, etc.



R = log *N*
H = log *N*
(with guarantee)

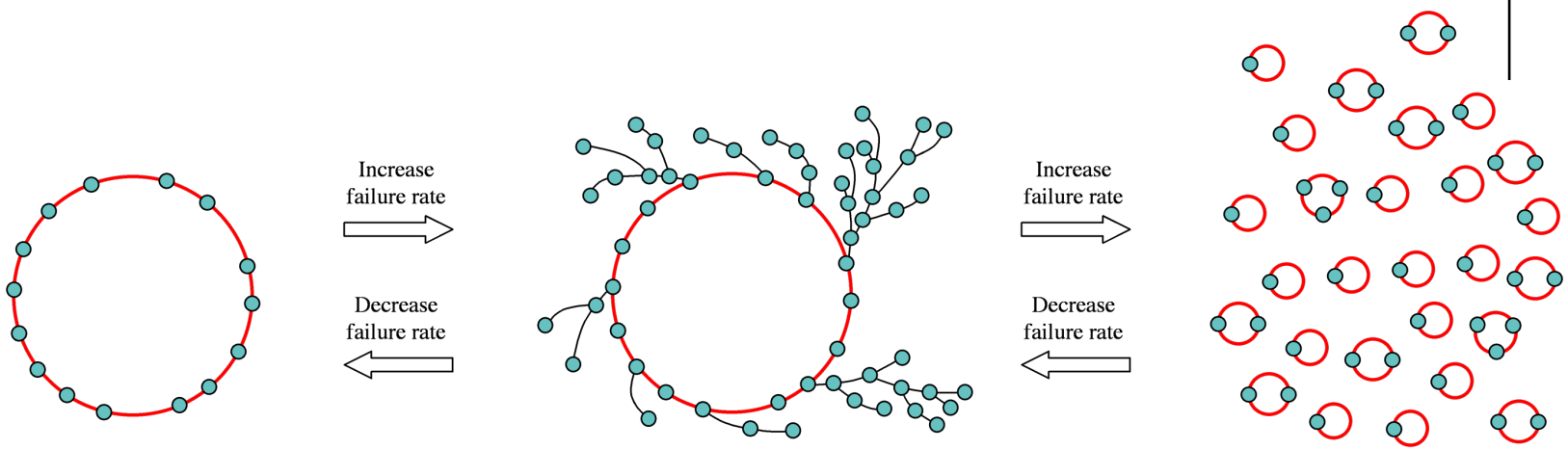
THE ADVENTURES OF SELFMAN

A "Relaxed" Ring: Beernet



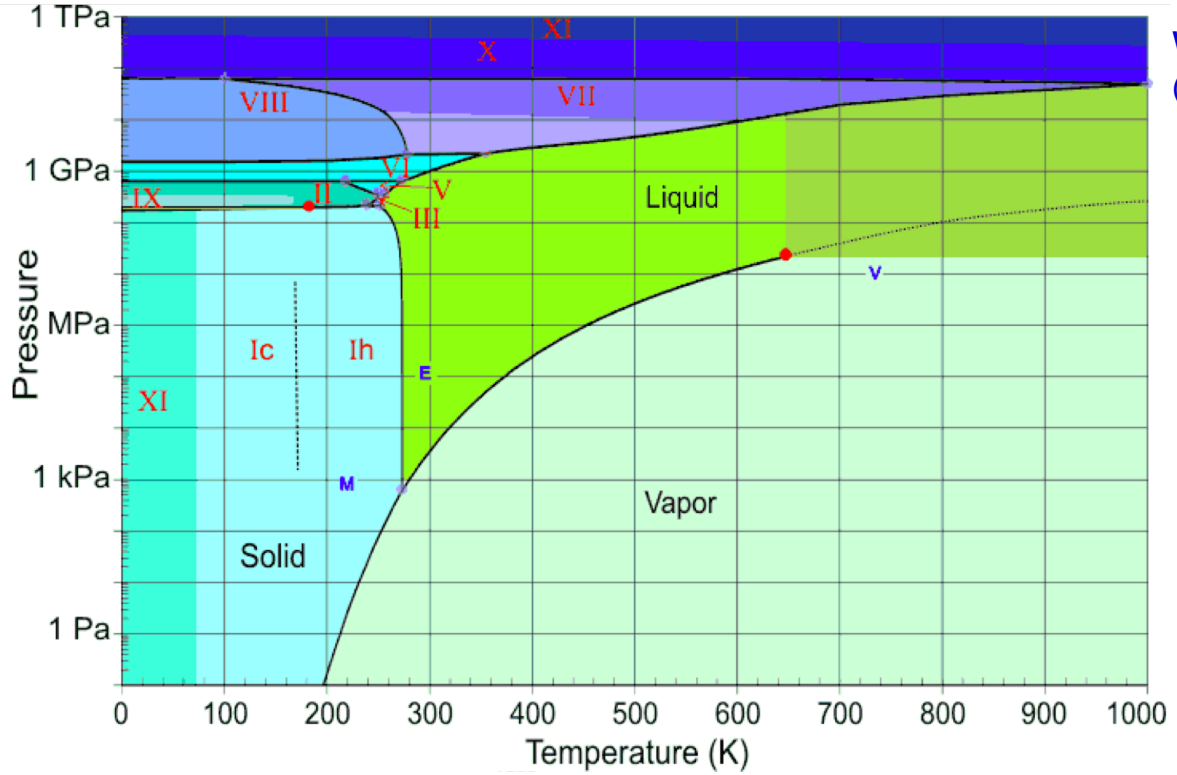
- The relaxed ring is completely asynchronous
  - Join and leave are completely asynchronous
  - The bushes appear only if there are failure suspicions
  - Beernet implements the relaxed ring (SELFMAN)
- There is a perfect ring (in red) as a subset of the relaxed ring
- The relaxed ring is always converging to a perfect ring
  - The bushiness depends on churn (rate of change of the ring, leaves/joins) and failure suspicion rate (communication delays)

Phases in the Relaxed Ring



- The relaxed ring has (at least) three phases
  - Uses ring merge algorithm developed in SELFMAN
  - We are studying how the ring reacts to external stress (phase transitions)
- Key questions:
  - How do the phases show up at the application layer? ("qualitative changes")
  - How do we know when we are near a phase transition? ("early bubbling")

Phases in Large Systems

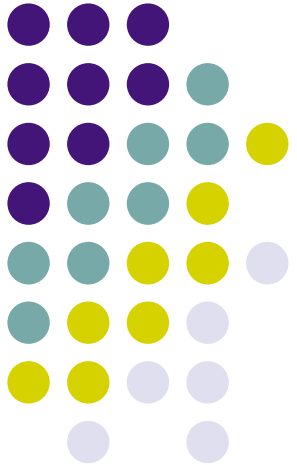


Water phase diagram
(Copyright © Martin Chaplin)

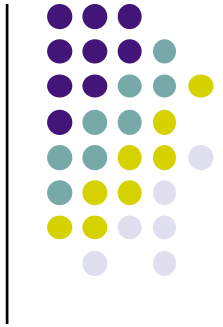
- A phase is a concise characterization of an aggregate behavior in a system consisting of many interacting components
- Phases appear in many large systems
  - Not just physical systems (water) but also computing systems (like peer-to-peer)

- Different parts of the system can be in different phases (no global synchronization!)
  - Depending on the local operating conditions (environment)
  - Boundaries between phases can be sharp or diffuse
  - Phase transitions and critical points can occur if operating conditions change

Conclusions and Prospects

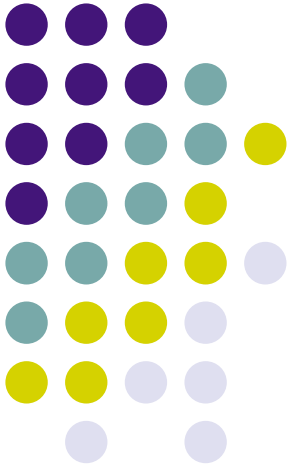


Conclusions and Prospects

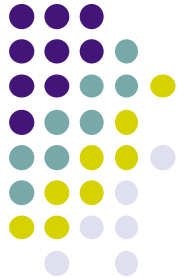


- Laws of scalability
  - First Law: new phenomena appear at each scale
  - Second Law: as scale increases, systems have only local control
  - Third Law: pick two of consistency, availability, partition tolerance
- Clouds are a key part of the next Internet revolution
  - Elasticity leads to Heisenberg applications
  - Demand will cause proliferation of federated clouds
- Design for scalability: a research agenda
  - Weakly interacting feedback structures with dominant subsets
  - Complex components to solve the problem in limited conditions
  - Phases to define behavior over all possible operating conditions

References for Further Reading



References (1)

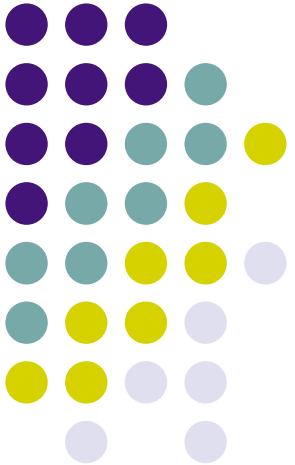


- Joe Armstrong. *Making Reliable Distributed Systems in the Presence of Software Errors*, Ph. D. dissertation, Royal Institute of Technology (KTH), Kista, Sweden, Nov. 2003.
- Ken Birman, Gregory Chockler, and Robbert van Renesse. "Toward a Cloud Computing Research Agenda", 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, ACM SIGACT News, 40(2): 68-80 (June 2009).
- Alexandre Bultot. *A Survey of Systems with Multiple Interacting Feedback Loops and Their Application to Programming*, Master's report, Dept. of Comp. Sci. and Eng., UCL, Aug. 2009.
- Rick Cattell. "High Performance Scalable Data Stores", Feb. 22, 2010.
- Raphaël Collet. *The Limits of Network Transparency in a Distributed Programming Language*, Ph. D. dissertation, Dept. of Comp. Sci. and Eng., UCL, Dec. 2007.
- Michael Fischer, Nancy Lynch, and Michael Paterson. "Impossibility of Distributed Consensus with One Faulty Process", Journal of the ACM, 32(2): 374-382 (April 1985).
- Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services", ACM SIGACT News, 33(2): 51-59 (2002).
- Rachid Guerraoui and Luís Rodrigues. *Introduction to Reliable Distributed Programming*, Springer-Verlag, 2006.
- Márk Jelasity and Özalp Babaoglu. "T-Man: Gossip-based Overlay Topology Management", Proc. 3rd Int. Workshop on Engineering Self-Organising Systems (ESOA 2005), Springer-Verlag LNCS volume 3910, 2006, pp. 1-15.
- Boris Mejías. *A Relaxed Ring for Self-Managing Decentralized Systems with Transactional Replicated Storage*, Ph. D. dissertation, Dept. of Comp. Sci. and Eng., UCL, Oct. 2010 (in preparation).
- Gerhard Michal and Dietmar Schomburg. *Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology*, Wiley-Blackwell, 1999 (first edition), 2011 (second edition, to appear).

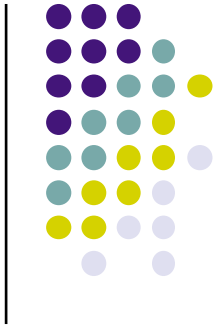
References (2)

- 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.
- SELFMAN: Self Management for Large-Scale Distributed Systems Based on Structured Overlay Networks and Components. European 6th Framework Programme, www.ist-selfman.org (2009).
- Peter M. Senge *et al*. *The Fifth Discipline Fieldbook: Strategies and Tools for Building a Learning Organization*, Nicholas Brealey Publishing, 1994.
- Tallat M. Shafaat, Ali Ghodsi, and Seif Haridi. "Dealing with Network Partitions in Structured Overlay Networks", Journal of Peer-to-Peer Networking and Applications, 2(4): 334-347 (2009).
- Steven Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering (Studies in Nonlinearity)*, Perseus Books, 1994.
- Nassim Taleb. *The Black Swan: The Impact of the Highly Improbable*, Penguin Books, 2008.
- Peter Van Roy, Seif Haridi, and Alexander Reinefeld. "Software Design with Weakly Interacting Feedback Structures and Its Application to Distributed Systems", Research Report RR2011-01, ICTEAM Institute, UCL, Jan. 2011.
- Peter Van Roy. "Programming Paradigms for Dummies: What Every Programmer Should Know", chapter in New Computational Paradigms for Computer Music, G. Assayag and A. Gerzso (eds.), IRCAM/Delatour France, June 2009.
- Gerald M. Weinberg. *An Introduction to General Systems Thinking*, Dorset House Publishing, 1975 (Silver Anniversary Edition 2001).
- Norbert Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine*, MIT Press, Cambridge, MA, 1948.
- Ulf Wiger. "Four-fold Increase in Productivity and Quality – Industrial Strength Functional Programming in Telecom-Class Products", Ericsson Telecom AB, 2001.

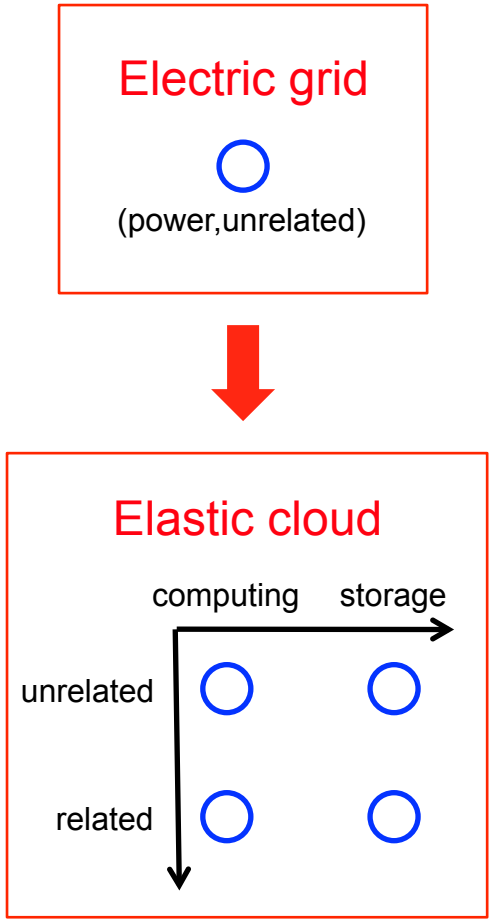
Stub Slides



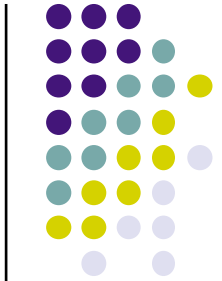
The Structure of Elasticity



- Elasticity of clouds has been compared to an electric grid
  - This is a reasonable comparison, but elasticity in clouds is more complex than in electric grids (for example, often the storage must survive since it is shared by many tasks)
- Elasticity in clouds has two dimensions: computing/storage vs. unrelated/related
  - Elastic computing: often amortization between unrelated tasks
    - But computing can also involve related tasks (solution sharing)
  - Elastic storage: often amortization between related tasks
    - But storage can also involve unrelated tasks (temporary storage)
- Elastic tasks are grouped depending on whether they are related or not
  - Storage tasks are related when they share storage
  - Computing tasks are related when they share solutions



Scalability Implies Long Life

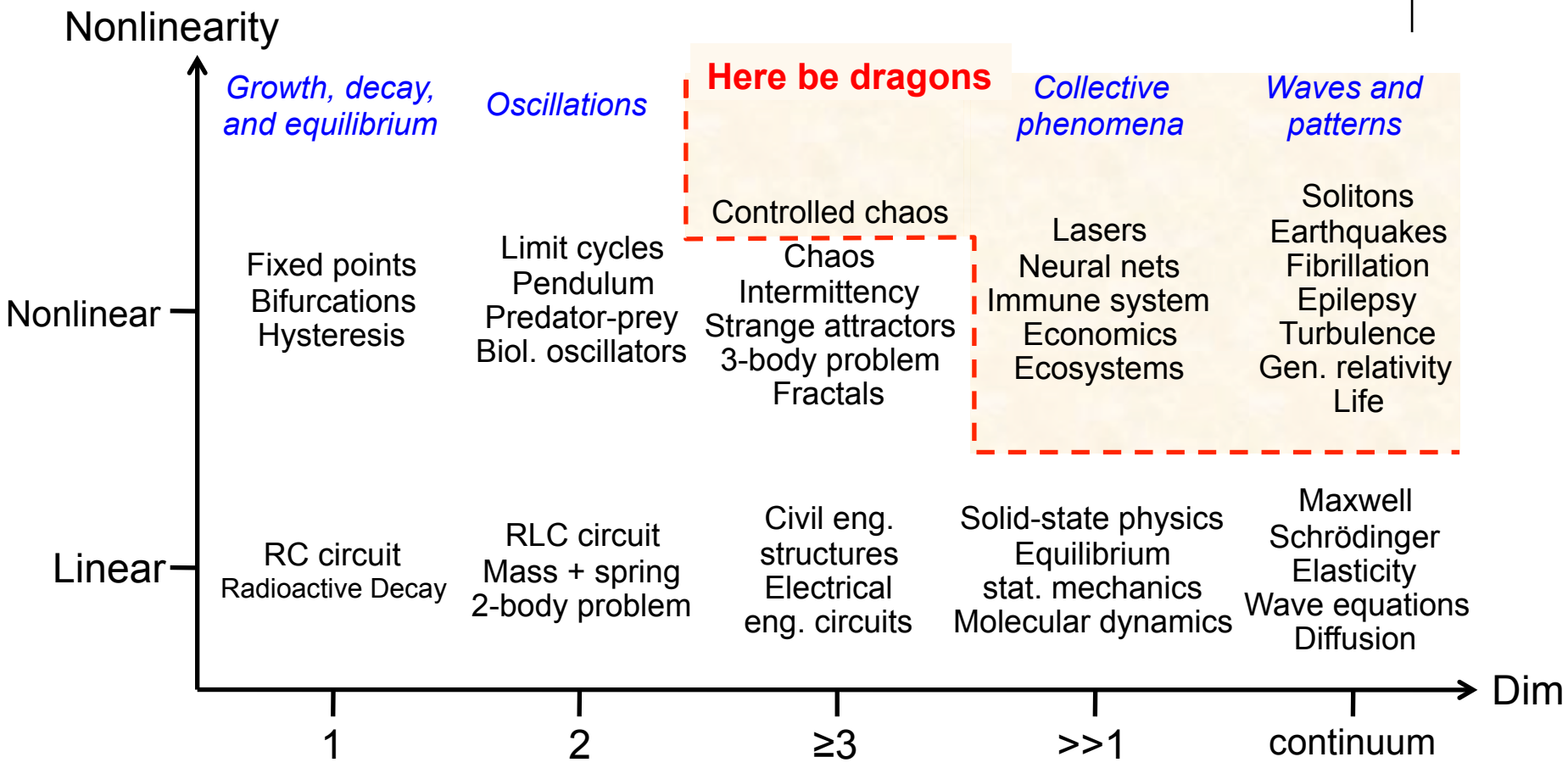


- A scalable system is not just large, it is also long-lived
- Memory leaks
  - Memory leaks are hard to find in distributed systems because of remote references and failures. There is no practical algorithm for true distributed garbage collection.
  - The best technique is still distributed reference counting, with time-lease references and program management of distributed cycles. This crosses all abstraction layers.
- Partial failures
  - Failures of parts of the system are frequent and can be fixed by redundancy
- Software rejuvenation
  - Periodically restart the system with a valid state recovered from the previous incarnation. This solves both memory leaks and partial failures.
  - Used by biological systems for eons: it's why we are not immortal. A fertilized egg is a newly initialized process. The older we get, the more defects accumulate.

Scalability and Concurrency

- The Second Law implies concurrency (independence) by default
- Concurrency and parallelism are often confused, so let us define their common core, "coexistence"
  - Concurrent = consisting of logically independent parts (programming concept)
  - Parallel = executing on separate processors (hardware concept)
  - Coexistent = "existing together" (dictionary definition)
    - Coexistent design: the discipline of building systems as collections of separate parts (at all levels, including hardware and software)
- Concurrency has always existed in computing
  - All programs can be decomposed into almost-independent parts
- Parallelism was a fringe area until recently
  - Multicore processors since 2001 (IBM POWER4 dual-core)
  - Distributed programming mostly client/server until 1990s
- Now parallelism is mainstream and concurrency is embracing it
  - For multicore: add dataflow ideas to programming languages (sociological!)
  - For Internet: techniques from distributed algorithmics (still very technical)

Scalability in Dynamics

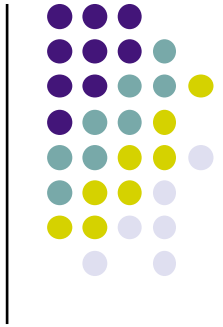


- From [Strogatz 1994] *Nonlinear Dynamics and Chaos*

Simple Forms of Concurrency are the Right Defaults

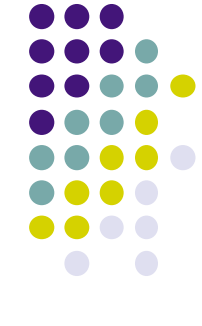
1. The simplest paradigms for concurrent programming are deterministic dataflow concurrency and message-passing concurrency
   - Compare the simplicity of *Concurrent Programming in Erlang* with the complexity of *Concurrent Programming in Java*
   - Deterministic concurrency is the key to simplifying concurrent programming. All forms of deterministic concurrency are explained in [Van Roy 2009].
2. The Erlang language and system is used successfully for building highly available systems; it uses message-passing concurrency with independent agents
3. The E language and system is used successfully for building secure distributed systems; it uses deterministic concurrency to avoid the covert channels of nondeterminism

Civilization Relies on Feedback Loops

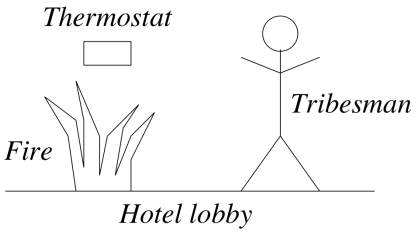


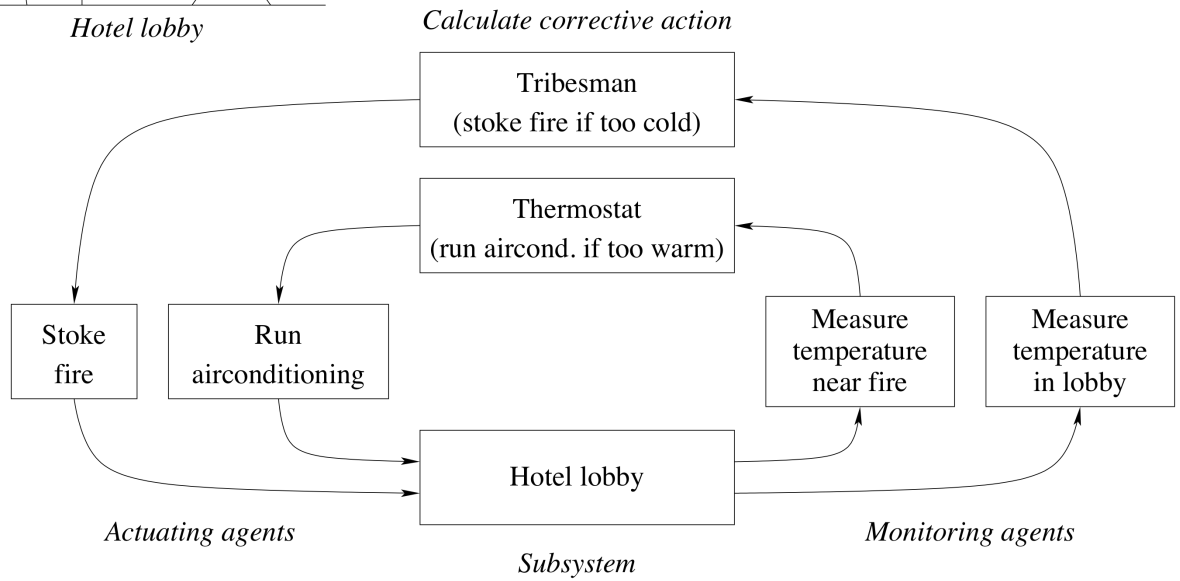
- Most products of human civilization use an implicit management feedback loop, called “maintenance”, done by a human
  - Changing lightbulbs, replacing broken windows, filling up a car
- Each human mind is at the center of many such feedback loops
  - Most require very little conscious thinking, since they have become “habits”: programmed into the brain below consciousness
  - Each human being creates huge numbers of such habit programs
- But if there are too many feedback loops to manage then 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?

Hotel Lobby Example (from [Wiener 1948])



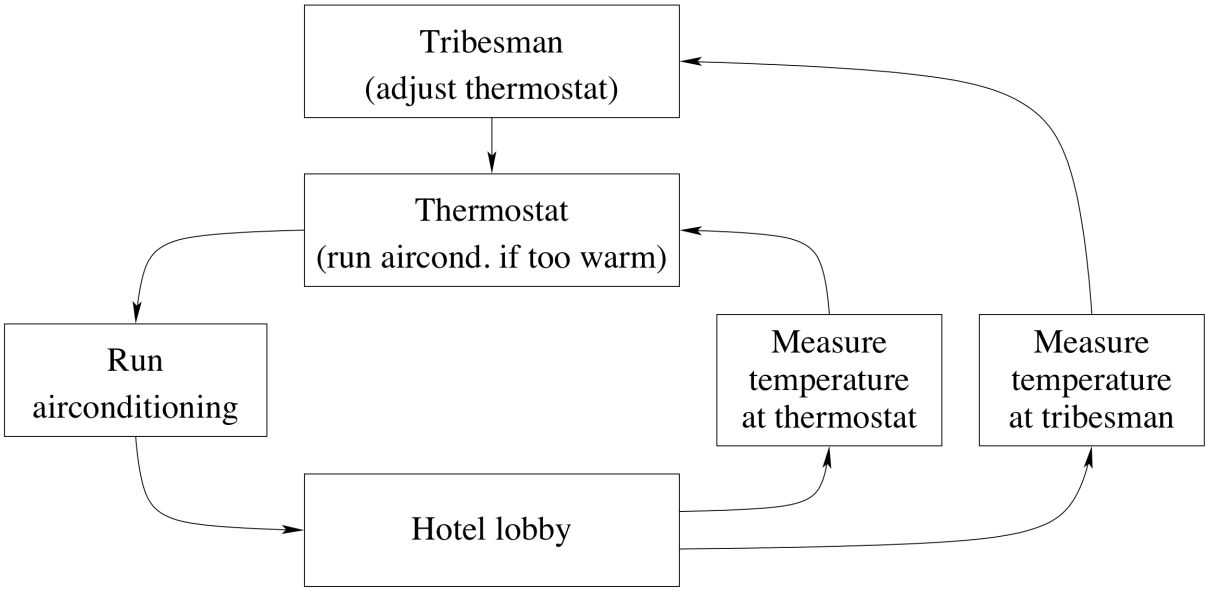
- Two loops interacting through a common subsystem (stigmergy)





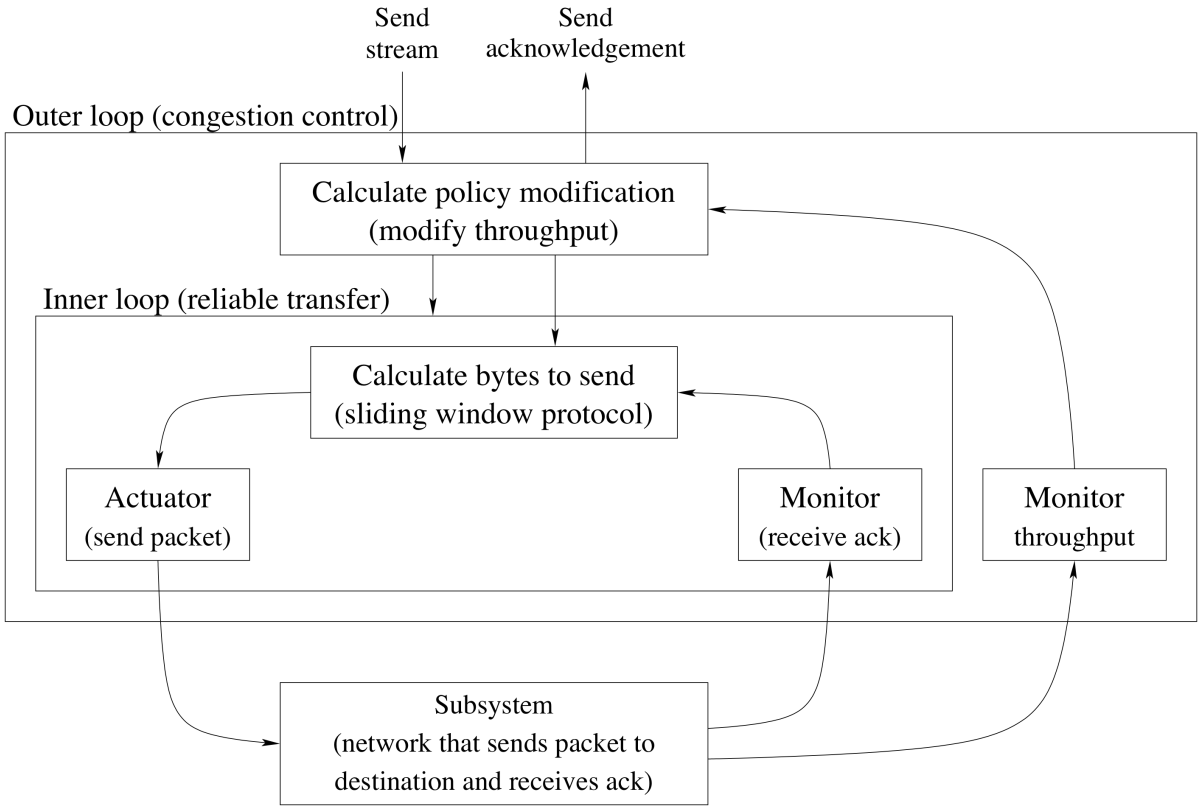
- This is unstable!
  - The tribesman stokes the fire but gets colder and colder because the airconditioning works harder and harder
- Wiener leaves the fix as homework for the reader
- One possible solution: outer loop (tribesman) controls the other by simply adjusting the thermostat
  - One loop controls the other

Correct Solution Uses Management



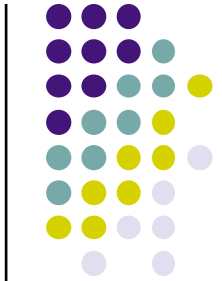
- Instead of stoking a fire, the tribesman simply adjusts the thermostat. The resulting system is stable.
- This uses management instead of stigmergy
- Design pattern: use the system, don't try to bypass it

TCP Feedback Structure



- This example shows a reliable byte stream protocol with congestion control (a variant 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
- With *n* connections there are *n* feedback structures interacting through a shared network (stigmergy)
  - This is an example of a system with *n* WIFS

PageRank in One Slide



- Each Web page holds a quantity of stuff called its "importance"
- At each step, the "importance" flows out along the outgoing links
  - And new stuff comes in through the incoming links
  - Not all flows out (damping factor $d \approx 0.85$) since paths are not infinite
- We iterate until the amount is the same for all pages
  - The final value gives an indication of how important a page is: a page is more important when there are more links from pages that are themselves important
- This is a global fixpoint calculation: the PageRank values are the entries of the dominant eigenvector of the Web adjacency matrix with damping factor

$$\mathbf{R} = \begin{bmatrix} PR(p_1) \\ PR(p_2) \\ \vdots \\ PR(p_N) \end{bmatrix}$$

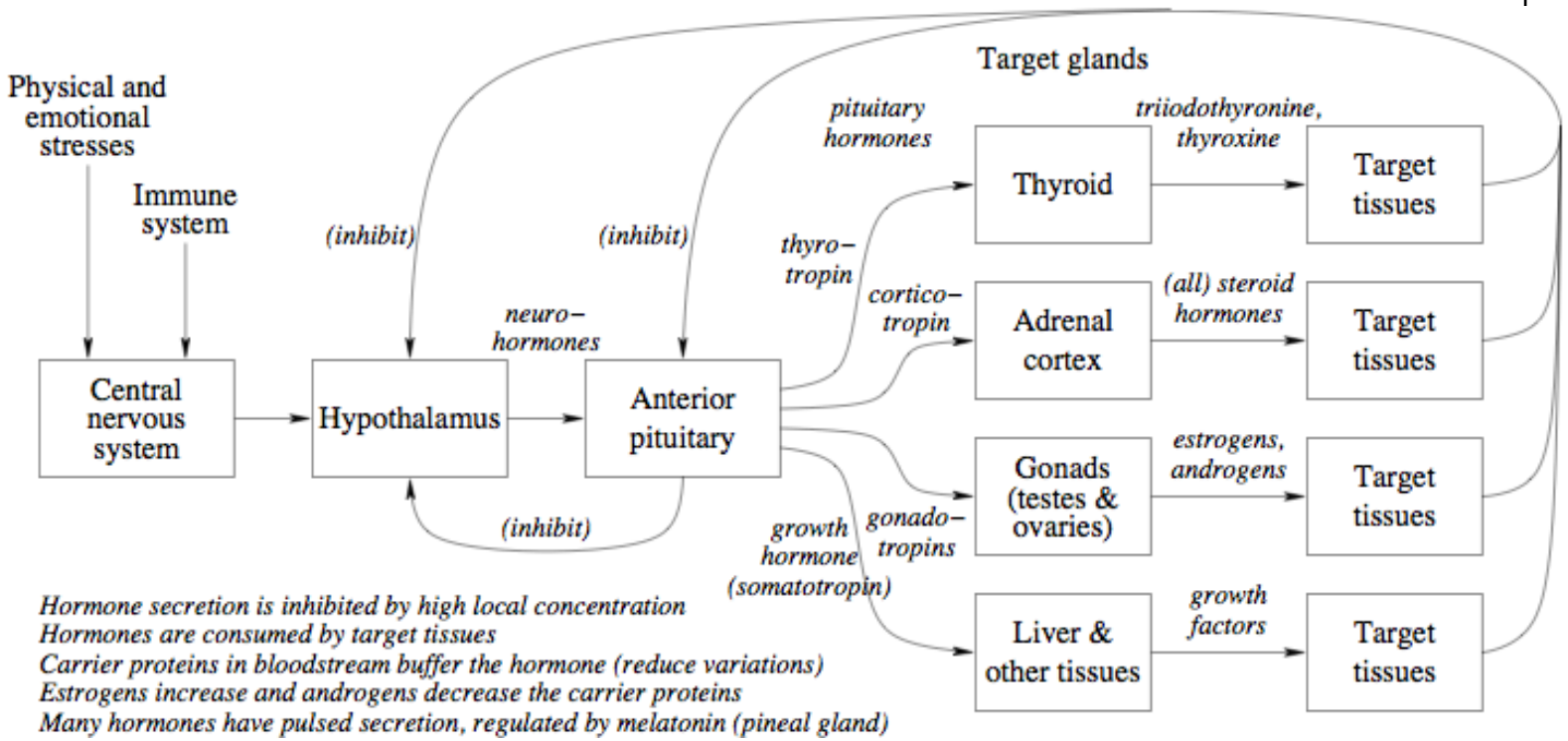
PageRank vector

Normalized Web adjacency matrix

$$\mathbf{R} = \begin{bmatrix} (1-d)/N \\ (1-d)/N \\ \vdots \\ (1-d)/N \end{bmatrix} + d \begin{bmatrix} \ell(p_1,p_1) & \ell(p_1,p_2) & \cdots & \ell(p_1,p_N) \\ \ell(p_2,p_1) & \ddots & & \vdots \\ \vdots & & \ell(p_i,p_j) & \\ \ell(p_N,p_1) & \cdots & & \ell(p_N,p_N) \end{bmatrix} \mathbf{R}$$

PageRank equation: multiply R by adjacency matrix and adjust with damping factor

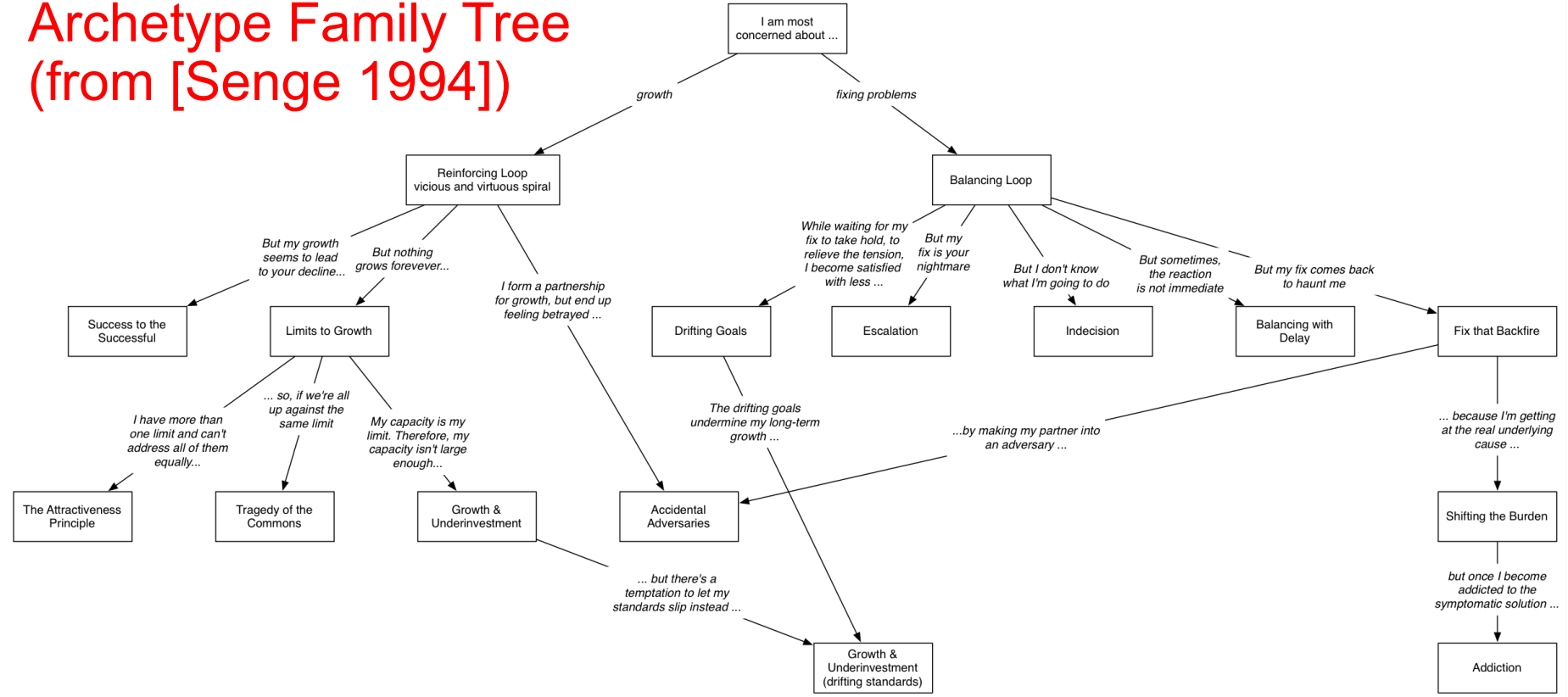
Hypothalamus-pituitary-target organ Axis (Endocrine System)



- Two superimposed groups of negative feedback loops, a third short negative loop, a fourth loop from the central nervous system (from [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

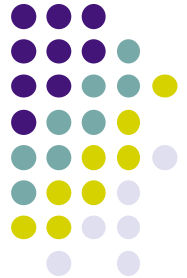
Design Patterns for Feedback Structures

Archetype Family Tree (from [Senge 1994])



- We can arrange feedback structures in a tree according to their relationships and the problems they solve

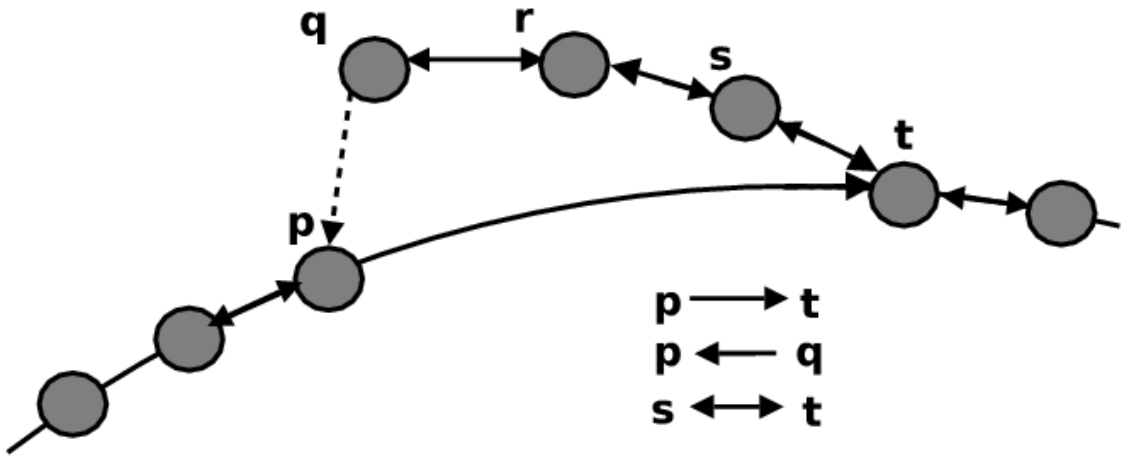
What About Levels of Abstraction?



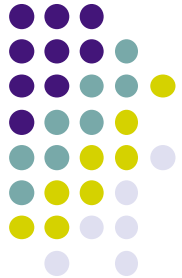
- WIFS architecture seems to imply a single level, yet novelty is observed at all levels
  - How can we reconcile this with the First Law?
- Solution: WIFS structure exists at all levels, organized according to Second and Third Laws (asynchrony and CAP)
- For example, in a multicellular organism:
  - Single cell contains many WIFS, cells communicate following CAP constraints
  - Organs uses WIFS to maintain its operation
  - Complete organism uses WIFS to survive in its environment

More on the Relaxed Ring

- False failure suspicions are common on the Internet
  - We do not want to eject the node from the ring when this happens
- The relaxed ring solves this by doing ring maintenance in asynchronous fashion [Mejias 2008]
  - Nodes communicate through message passing
  - For a join, instead of one step involving 3 peers (as in Chord or DKS), we have two steps each with 2 peers → we do not need locking or a periodic stabilization algorithm
- Invariant: Every peer is in the same ring as its successor

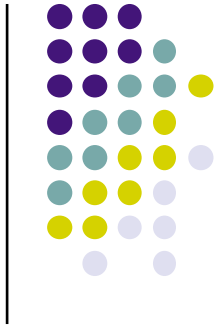


Nonlinearity



- The world is a curious combination of linearity and nonlinearity
  - Linearity = independent parts = whole equals the sum of the parts
  - Nonlinearity = interacting parts = whole is more than the sum of the parts
- Why are nonlinear systems so much harder to analyze quantitatively than linear ones?
  - Because in linear systems, the parts can be analyzed separately and then combined (superposition principle, compositional systems)
    - But there is a surprising twist: many nonlinear systems can be analyzed qualitatively (with a combination of geometrical reasoning and some analysis), which is often good enough
    - See [Strogatz 1994] *Nonlinear Dynamics and Chaos*
  - We need nonlinearity for "intelligent" behavior, but…
    - Too much nonlinearity makes the system fragile
  - That's why biological systems are made of *weakly* interacting subsystems
- What about nonlinearity and scalability?

Nonlinearity and Scalability



- Large systems must be mostly linear
  - Large systems consist of parts that can be superposed
    - Basic physical quantities are additive (mass, force, momentum, energy)
  - Because they can be superposed, the system is linear
- They can't be completely linear, though
  - Because we need nonlinearity for all nontrivial behavior
  - Interaction of two feedback structures is nonlinear
  - State change of a feedback structure is nonlinear
  - Complex components are nonlinear
- Therefore we should add nonlinearity where needed but no more
  - Current computing systems are far too nonlinear and discontinuous
  - They should be mostly linear with a smidgen of nonlinearity

Degrees of Increasing Irregularity in a Large System

1. Existence of probability distribution
   - Statistical physics holds, all microstates have equal probability, behavior is thermodynamic (describable by macroscopic state variables)
   - Unfortunately, most simulations and models are stuck here!
2. Critical point
   - Minor fluctuations can be amplified without bounds
   - The limit of statistical physics
   - Many computing systems have critical points (garbage collectors, dynamic hash tables, wide-area routing, virtual memory)
3. No probability distribution exists ("Black Swans")
   - We know only the range of behavior, frequency limits do not exist
     - Dijkstra's guarded commands have this behavior
   - Complex systems, program verification, distributed algorithmics