

# **Software (Systems) Architecture Foundations**

Lecture #8  
(Boldly) To the Future ...

Alar Raabe

# Recap of Last Lecture

If a builder builds a house for someone, and does not construct it properly, and the house falls and kills its owner, then that builder shall be put to death

Hammurabi, King of Babylon (1780 BC)

- Architect has overall responsibility for ensuring
  - the **completeness** and fitness-for-purpose of the architecture, in terms of adequately addressing all the concerns of its stakeholders
  - the **integrity** of the architecture, satisfactorily reconciling the conflicting concerns of different stakeholders, and showing the trade-offs made in doing so
- A key element in a successful Architecture Governance is a cross-organization **Architecture Board**
  - representative of all the key stakeholders in the architecture, and typically comprising a group of executives responsible for the review and maintenance of the overall architecture
- Architect role in
  - traditional development is to **design the architecture** – to make the decisions
  - agile development is to **enable/support/facilitate team to design the architecture** – not to make the decisions

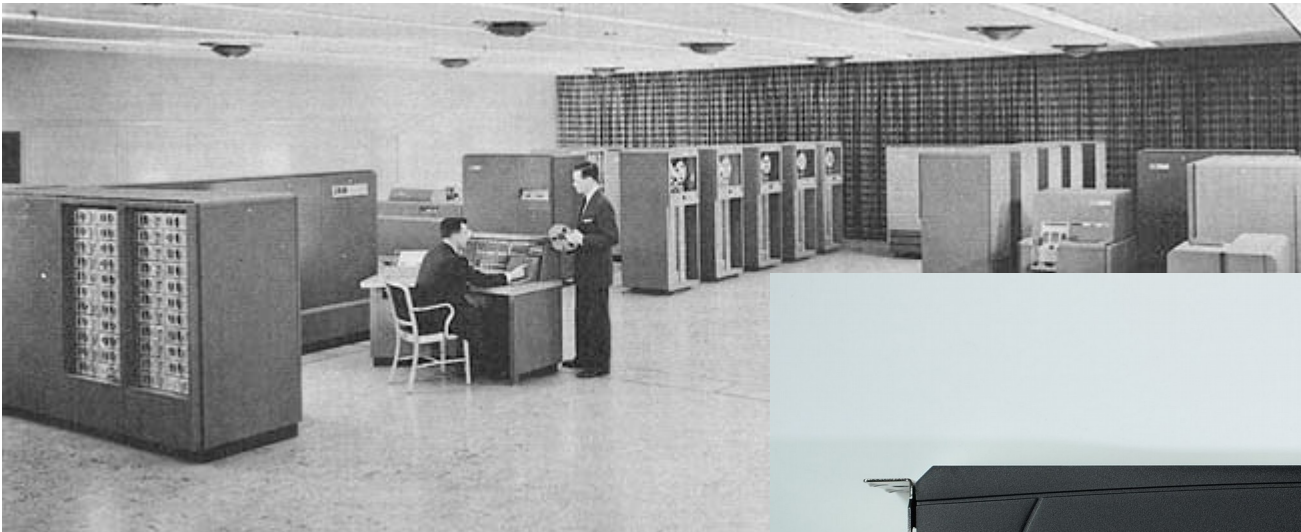
# Content

- Architecting for cloud
  - principles
  - design patterns
- Architecture of adaptive systems
  - reference model of adaptive systems
  - self-awareness
- Architecture of AI
  - machine learning systems
  - cognitive system architecture (example)
  - neural networks (architecture styles of neural networks)
- Conclusions

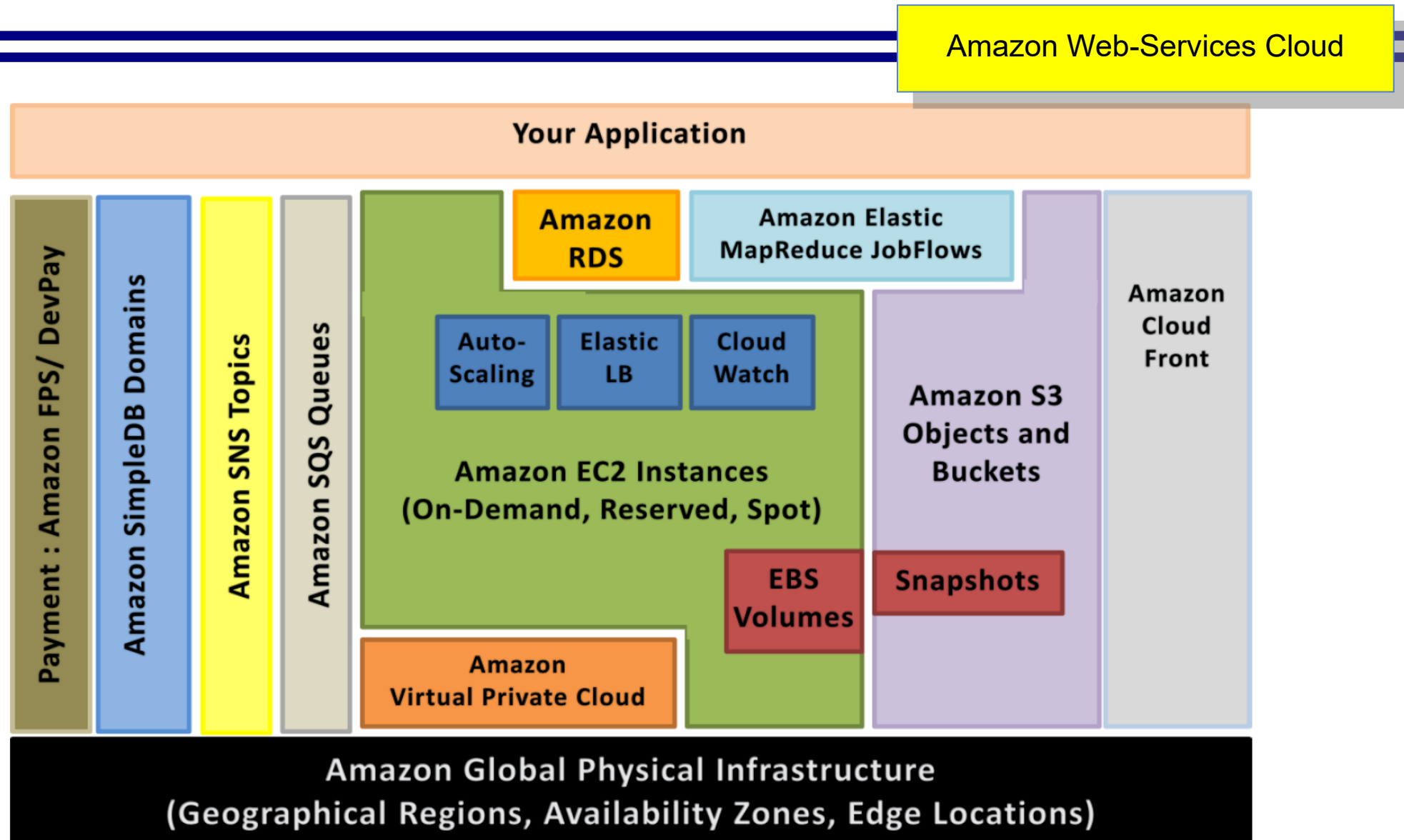
45. The architect allows things to happen. He shapes events as they come. He steps out of the ways and let the design speak for itself.

Lao Tsu (by Philippe Kruchten)

# Change ...



# Architecting for Cloud



# Architecting for Cloud

## Design Principles for AWS (EC2)

Amazon Web-Services Cloud

- Scalability
  - Scaling vertically – resizing resource – very easy, but there's hard limit
  - Scaling horizontally – increasing number of resources
    - separate Stateless Components (no knowledge of previous interactions) from Stateful Components
    - Distributed Processing – divide task and data into small fragments
- Disposable Resources Instead of Fixed Servers
  - Instantiating Compute Resources – Bootstrapping or Golden Images (snapshots for faster start times) or Hybrid (both)
  - Infrastructure as Code – all infrastructure assets are programmable – same techniques apply as for code
- Loose Coupling
  - Well Defined Interfaces – hide technical implementation
  - Service Discovery – avoid hard-coding deployment details
  - Asynchronous Integration – use message queues
  - Graceful Failure – handle failures
- Services, Not Servers
  - Managed Services – provided managed services as building blocks for applications
  - Serverless Architectures – remote execution of mobile code

# Architecting for Cloud

## Design Principles for AWS (EC2)

Amazon Web-Services Cloud

- Automation – react to various life-cycle and out of order events and specific situations
- Databases – use right database technology for each workload
  - Relational Databases
    - Scalability – can scale horizontally by replication for read and data partitioning or sharding for write
    - High Availability – automatic failover to replicated standby instance
    - Anti Patterns – if you don't need complex queries (e.g. joins), consider NoSQL database or for large objects the storage service
  - NoSQL Databases
    - Scalability – can scale horizontally by data partitioning or replication for both read and write
    - High Availability – synchronous replication
    - Anti-Patterns – if you need complex queries or transactions, consider relational database or for large objects the storage service
  - Data Warehouse
    - Scalability – use columnar data storage and massively parallel multiprocessing
    - High Availability – use replication
    - Anti-Patterns – don't do OLTP in DW
  - Search
    - Scalability – use data partitioning and replication
    - High Availability – store data redundantly

# Architecting for Cloud

## Design Principles for AWS (EC2)

Amazon Web-Services Cloud

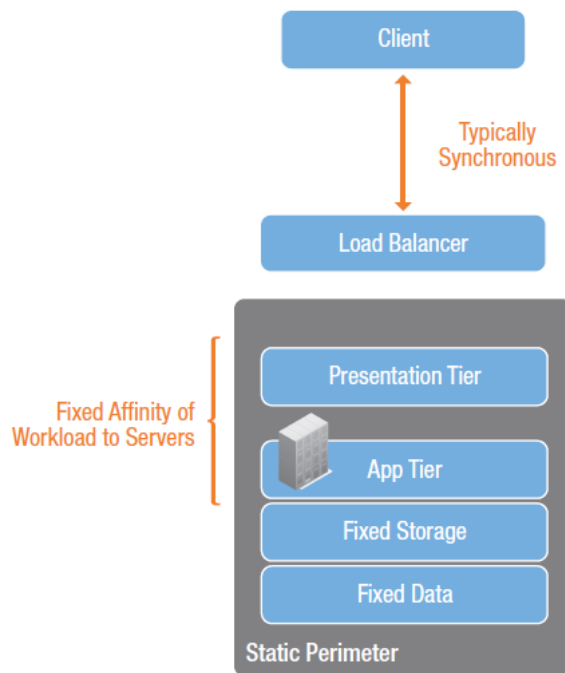
- Removing Single Points of Failure – build fault tolerant applications
  - Introduce Redundancy – either standby (using failover) or active redundancy
  - Detect Failure – automatically detect and react to failure (design health checks)
  - Durable Data Storage – synchronous replication for integrity, asynchronous replication, or quorum-based replication
  - Automated Multi-Data Center Resilience – availability zones
  - Fault Isolation and Traditional Horizontal Scaling – use Shuffle Sharding – group the instances of your system
- Caching
  - Application Data Caching
  - Edge Caching – use content delivery network for static content
- Security
  - Utilize AWS Features for Defense in Depth
  - Offload Security Responsibility to AWS
  - Reduce Privileged Access
  - Security as Code
  - Real-Time Auditing – do continuous monitoring and logging



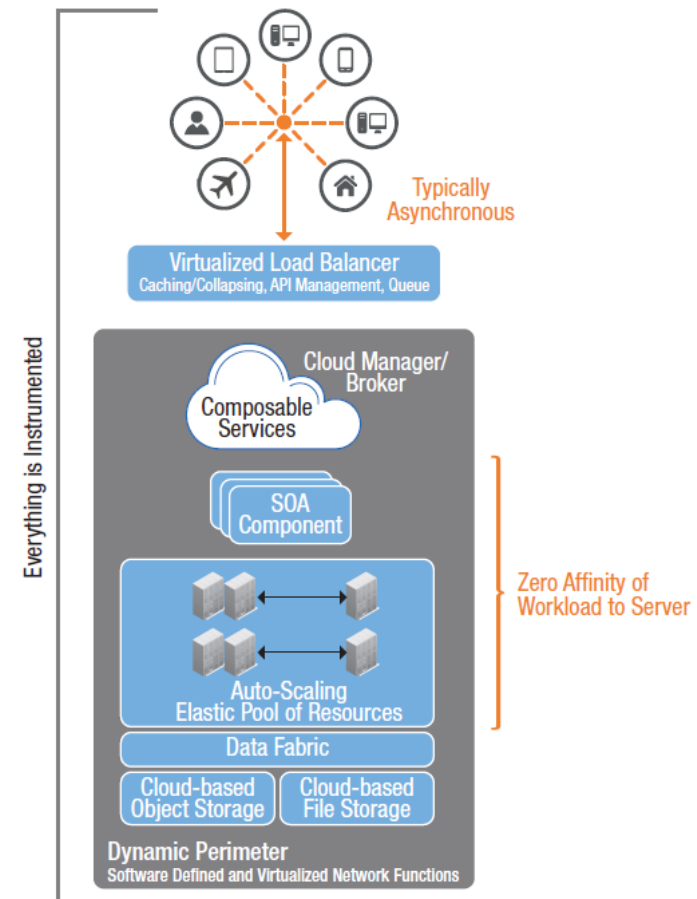
# Architecting for Cloud

Open Data Center Alliance

### Traditional Multi-Tiered App



### Cloud-Aware App



Pictures © Open Data Center Alliance

# Principles of Cloud Application Architecture

Open Data Center Alliance

- Resilient to failure
  - Resiliency is designed into the application, rather than wrapped around it after the fact
  - Failures in cloud infrastructure are handled fluidly without interruption of service
- Resilient to latency
  - Applications adapt gracefully to latency rather than timing out/failing
- Secure
  - Applications are based on secure life-cycle standards and include built-in security
  - Data at rest and in transit is encrypted. APIs are protected by authentication and authorization
- Location independent
  - Applications discover services dynamically rather than relying on hard-coded dependencies
- Elastically scalable
  - Applications respond to demand levels, growing and shrinking as required, in and among clouds
- SOA/Compose-ability
  - Applications consume and expose web services with APIs discoverable at runtime
  - The structure incorporates small, stateless components designed to scale out

# Principles of Cloud Application Architecture

Open Data Center Alliance

- Designed for manageability
  - Applications are instrumented and expose metrics and management interfaces
- Infrastructure independent
  - Applications make no assumptions about the underlying infrastructure, using abstractions in relation to the operating system, file system, database, and so on
- Defined tenancy
  - Each application should have a deliberate, defined single tenancy or multitenancy model
- Available end-user self-service
  - Users should be able to register themselves to use the app through a self-service registration interface, without entering an IT service request
- Bandwidth aware
  - APIs and application protocols are designed to minimize bandwidth consumption
- Cost and resource consumption aware
  - Application architecture is designed to minimize costs due to bandwidth, CPU, storage consumption, and I/O requests

# Cloud Application Design Patterns

Open Data Center Alliance

- Circuit Breaker – detect a fault in a circuit and trip to a safe fallback state, or open
- Request Queuing – use the queue as a buffer between the requests and processing service(s) preventing a heavy load from impacting the service causing it to fail or the request to time out
- Request Collapsing – collapse multiple requests over a given time interval into a single request, to reduce network bandwidth and load on the API end point, allowing it to scale to support a larger number of concurrent users
- Object Change Notification – (or Observer) use implicit invocation to enables an application to scale elastically (since components are no longer tightly coupled, additional instances can be added dynamically to handle increased load)
- Service Discovery – use service registry through which only healthy service instances are located at request time
- Microservices – to improve elasticity, decompose monolithic services functionally into fine-grained microservices, each with a single function
- Stateless Services – to provide Reliability (simple retry), Scalability, Visibility (both request and response payloads contain all the information needed to understand the transaction), Simplicity (no need to manage data or locks)
- Configuration Service – use External Configuration Store and Runtime Reconfiguration (handle configuration change events)
- Authorization Pattern – in a cloud environment, the network cannot be assumed to be secure because it is not under the application owner's control, use API Authorization to ensure that only trusted parties can use API

# Content

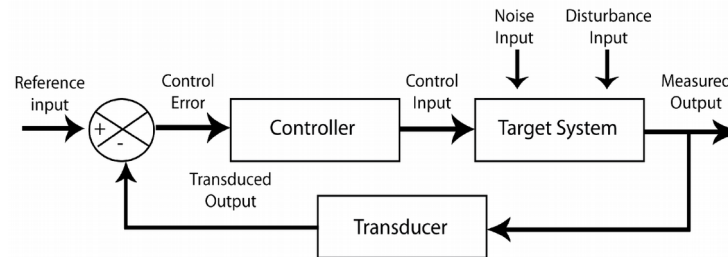
- Architecting for cloud
  - principles
  - design patterns
- Architecture of adaptive systems
  - reference model of adaptive systems
  - self-awareness
- Architecture of AI
  - machine learning systems
  - cognitive system architecture (example)
  - neural networks (architecture styles of neural networks)
- Conclusions

45. The architect allows things to happen. He shapes events as they come. He steps out of the ways and let the design speak for itself.

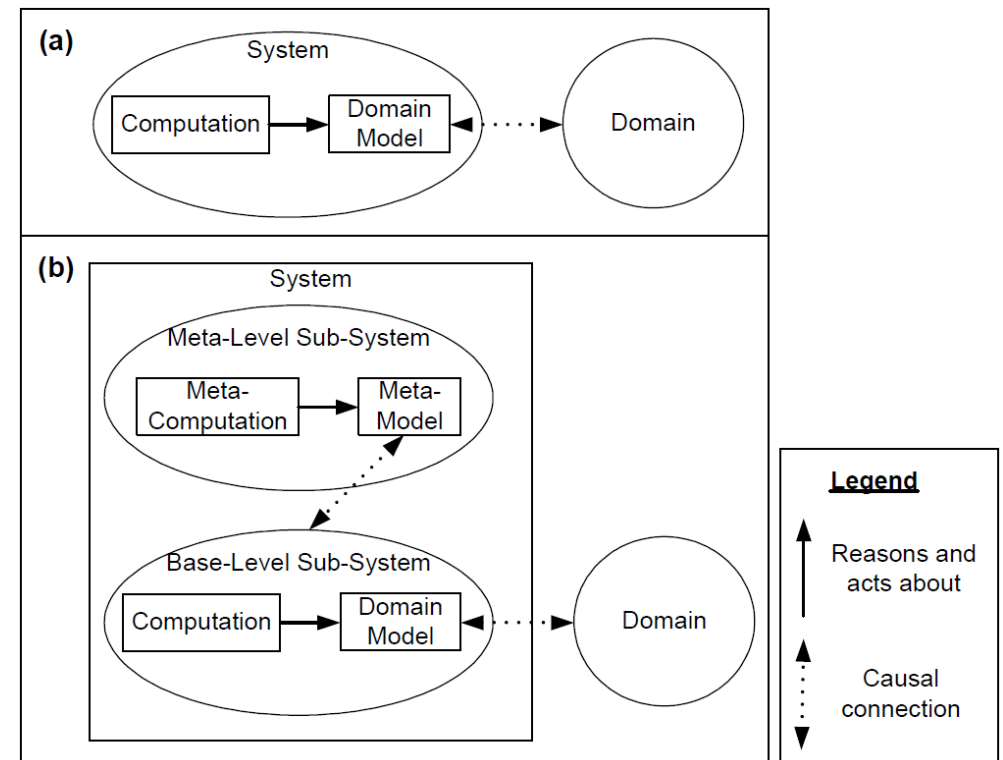
Lao Tsu (by Philippe Kruchten)

# Architecture of Adaptive Systems

- Control Loop

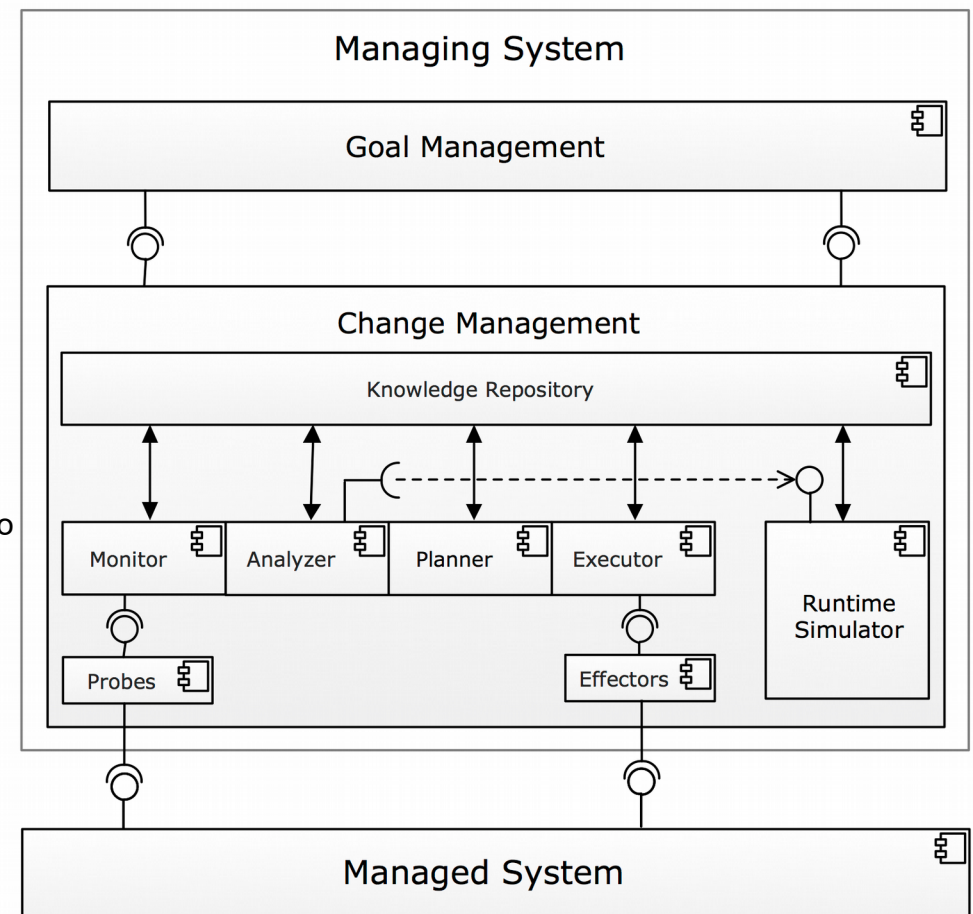


- Reflection/Introspection
  - Traditional (a) vs. Reflective System (b)

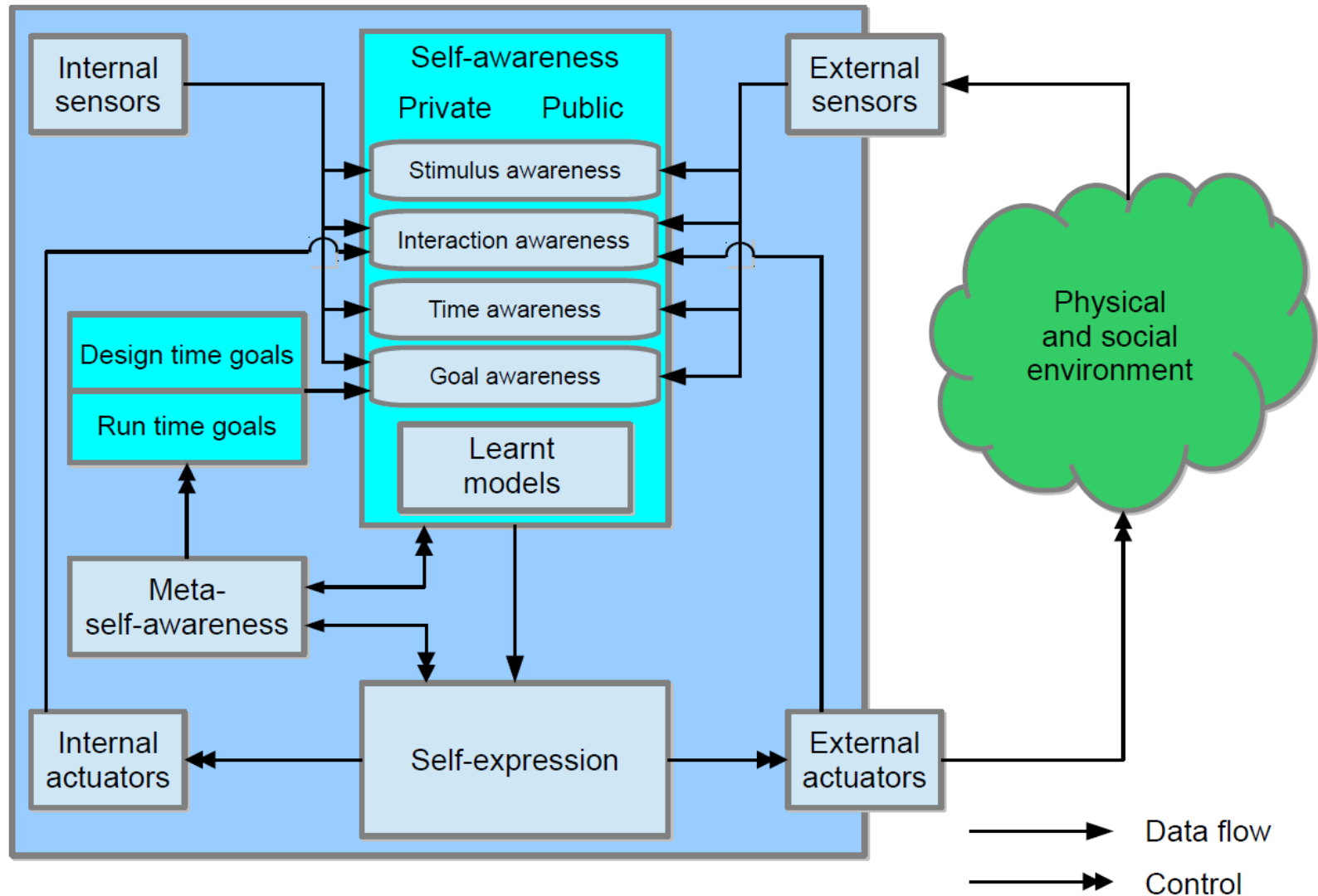


# Architecture of Adaptive Systems Reference Model (MAPE-K)

- Monitor
  - Collects the details from the managed resources e.g. topology information, metrics (e.g. offered capacity and throughput), configuration property settings etc., aggregates, correlates and filters these until it determines a symptom that needs to be analyzed
- Analyze
  - Perform complex data analysis and reasoning on the symptoms provided by the monitor function, is influenced by stored knowledge data, if changes are required, a change request is passed to the plan function
- Plan
  - Structures the actions needed to achieve goals and objectives (to enact a desired alteration in the managed resource)
- Execute
  - Changes the behavior of the managed resource using effectors, based on the actions recommended by the plan function
- Knowledge Management
  - Standard data shared among the monitor, analyze, plan and execute functions (incl. data such as topology information, historical logs, metrics, symptoms and policies), created by the monitor part while execute part might update the knowledge



# Architecture of Adaptive Systems (Self-Awareness)





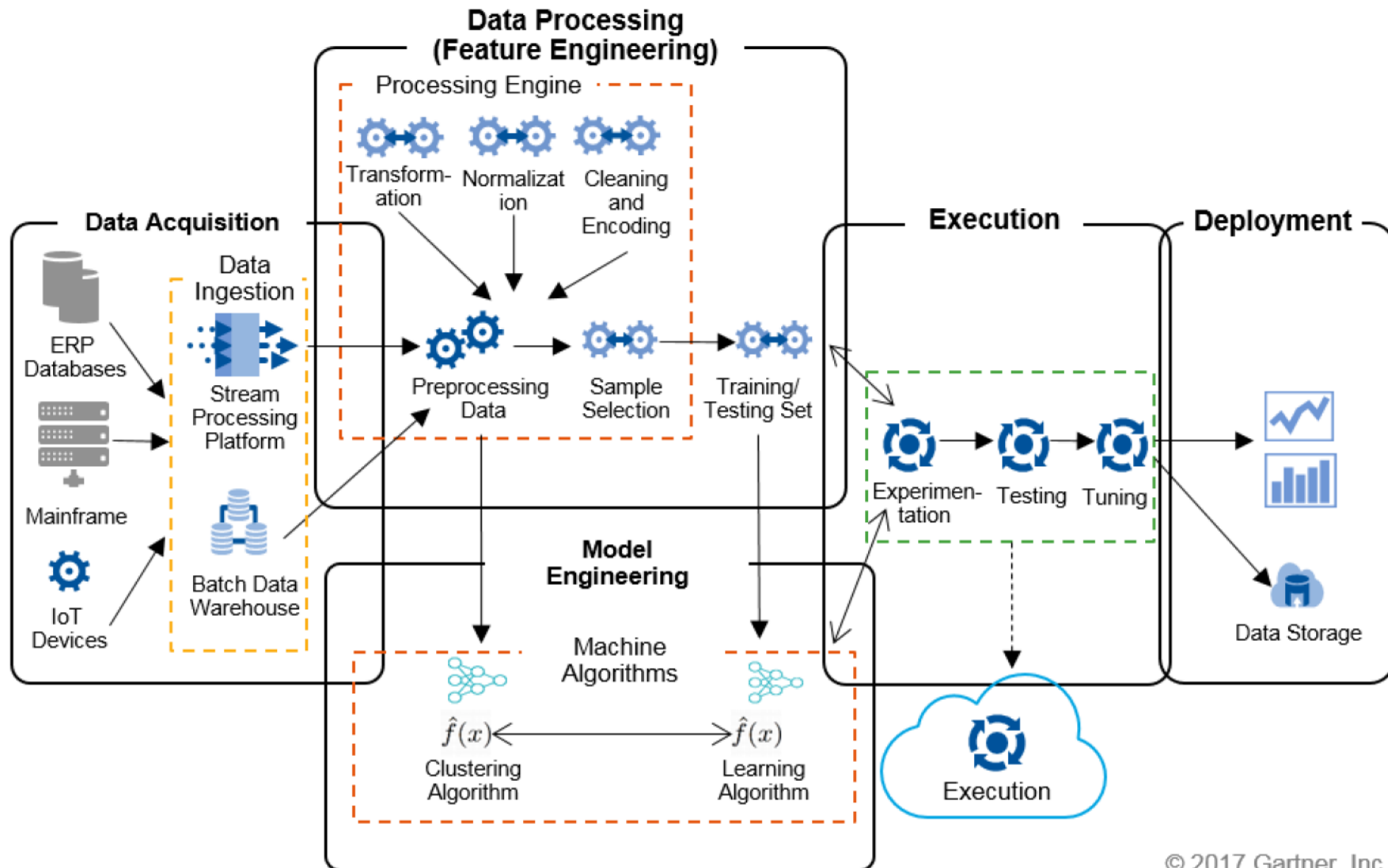
# Content

- Architecting for cloud
  - principles
  - design patterns
- Architecture of adaptive systems
  - reference model of adaptive systems
  - self-awareness
- Architecture of AI
  - machine learning systems
  - cognitive system architecture (example)
  - neural networks (architecture styles of neural networks)
- Internet-of-Things
  - reference architecture
- Conclusions

45. The architect allows things to happen. He shapes events as they come. He steps out of the ways and let the design speak for itself.

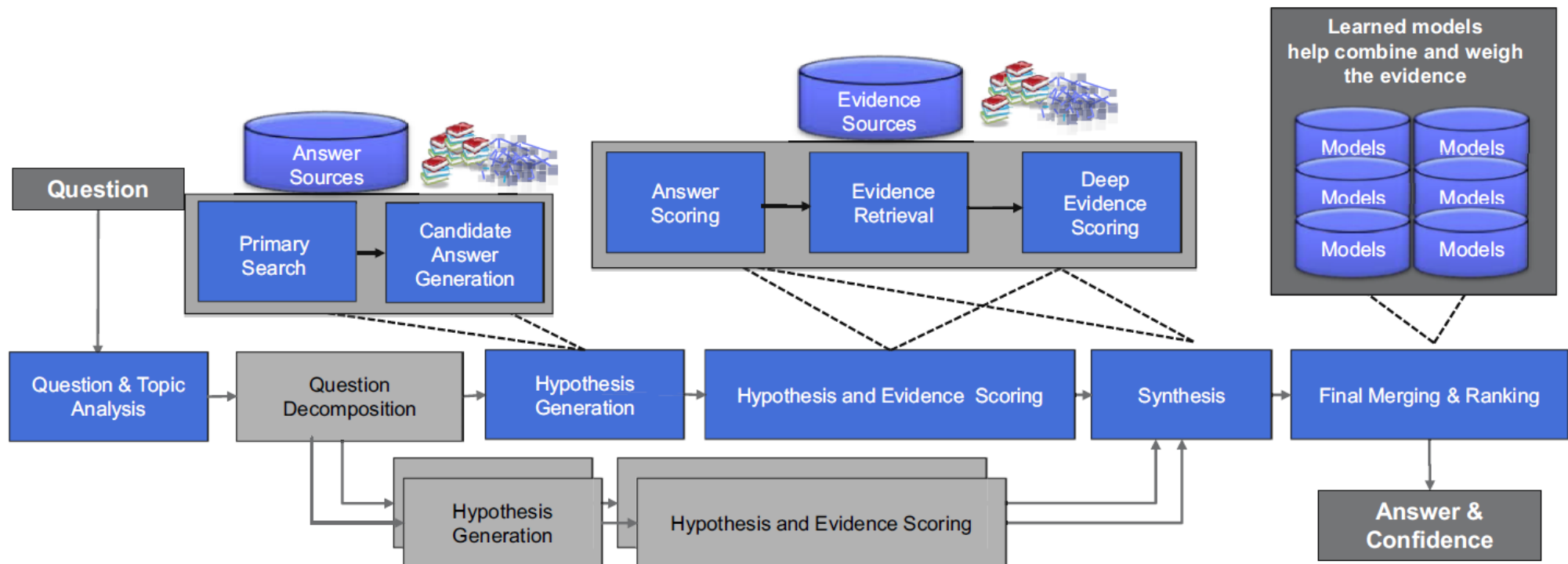
Lao Tsu (by Philippe Kruchten)

# Architecture of Machine Learning (Gartner)



# Cognitive System Architecture

## DeepQA Architecture (IBM Watson)



# Architecture of AI Systems

## Problems in Machine Learning Systems

Machine Learning: The High-Interest  
Credit Card of Technical Debt

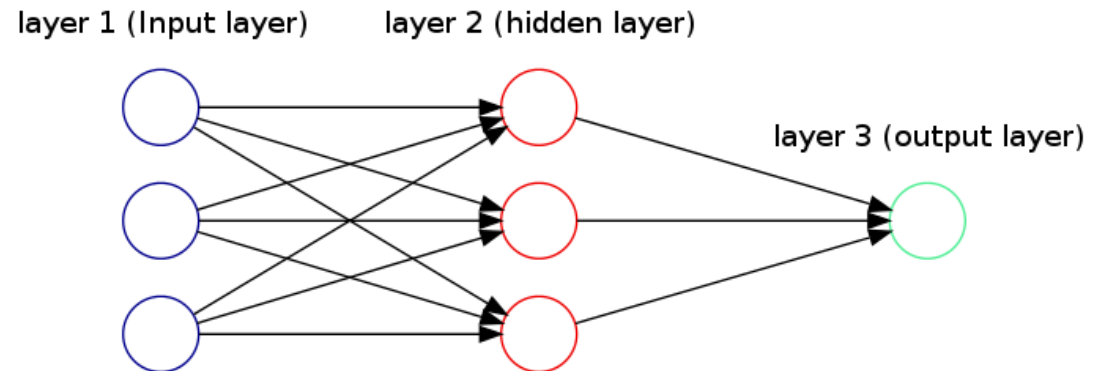
Google

- Complex Models Erode Boundaries
  - Entanglement – data and models are entangled
  - Hidden Feedback Loops – not all feedback loops and their effects are easily seen
  - Undeclared Consumers – predictions might be used in places where they shouldn't
- Data Dependencies Cost More than Code Dependencies
  - Unstable Data Dependencies – relationships in data inevitably change over time
  - Underutilized Data Dependencies – some features in data do not affect the accuracy of the outcome
  - Static Analysis of Data Dependencies – it is difficult to track data usage in the system
  - Correction Cascades – using different model with the correction creates dependency of models
- System Level Spaghetti
  - Glue Code
  - Pipeline Jungles
  - Dead Experimental Codepaths
  - Configuration Debt
- Dealing with Changes in the External World
  - Fixed Threshold in Dynamic Systems – manually selected thresholds become wrong when data changes
  - When Correlations No Longer Correlate
  - Monitoring and Testing – testing is not enough because world is changing

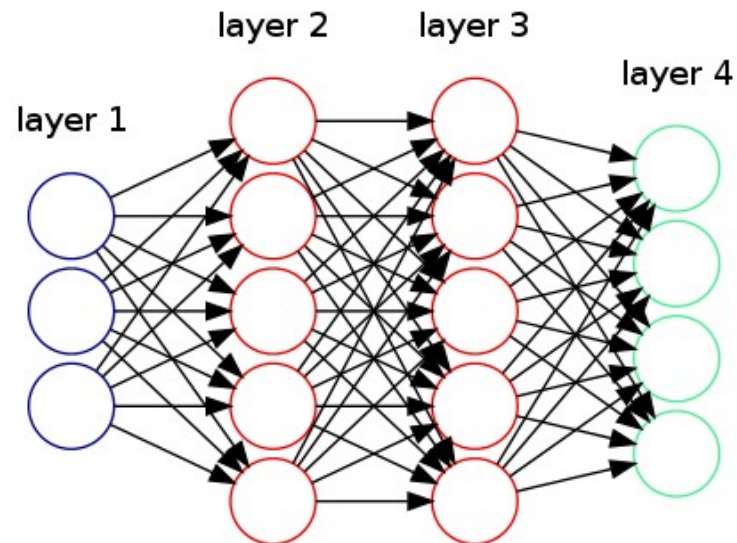
# Architecture of AI Systems

## Neural Networks

- Binary Classification



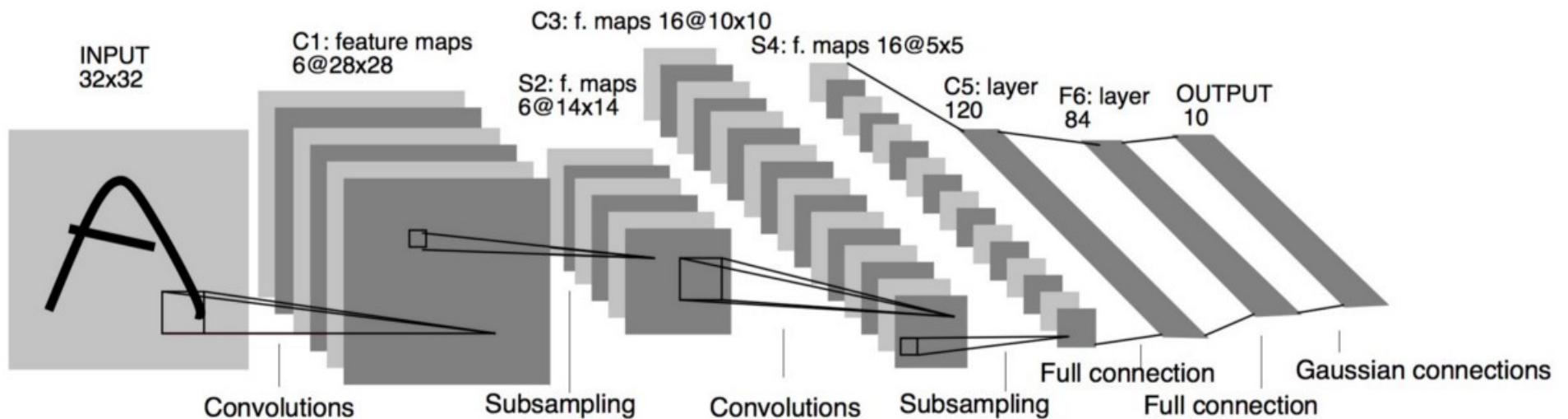
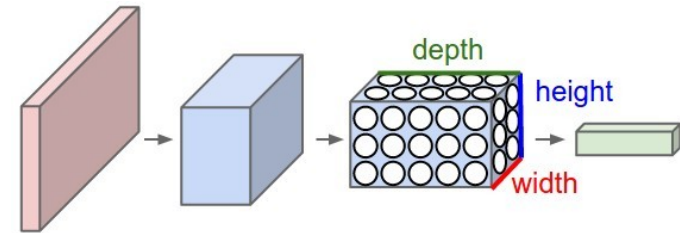
- Multi-Class Classification



# Architecture of AI Systems

## Neural Networks

- Deep Learning for Image Recognition (LeNet5, 1994)
  - convolutional (with input neurons arranged in 3 dimensions: width, height, depth) neural network with sequence of 3 layers:
    - convolution, pooling, non-linearity
  - use convolution to extract spatial features
  - subsample using spatial average of maps
  - multi-layer neural network (MLP) as final classifier
  - sparse connection matrix between layers to avoid large computational cost
















# Architecture Styles of Neural Networks

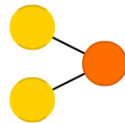
A mostly complete chart of

## Neural Networks

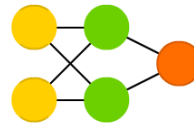
©2016 Fjodor van Veen - asimovinstitute.org

-  Backfed Input Cell
-  Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Different Memory Cell
-  Kernel
-  Convolution or Pool

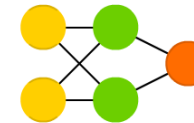
Perceptron (P)



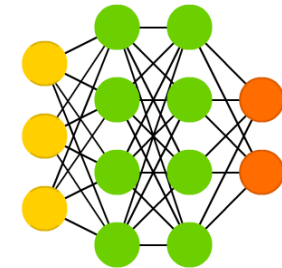
Feed Forward (FF)



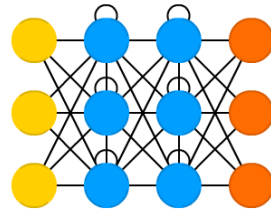
Radial Basis Network (RBF)



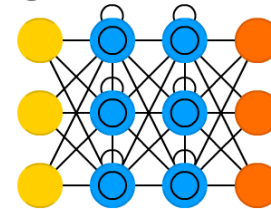
Deep Feed Forward (DFF)



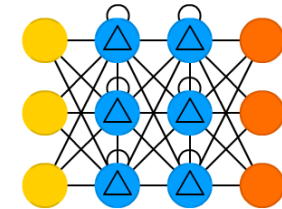
Recurrent Neural Network (RNN)



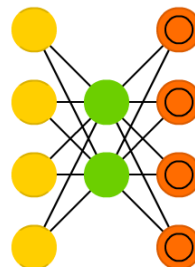
Long / Short Term Memory (LSTM)



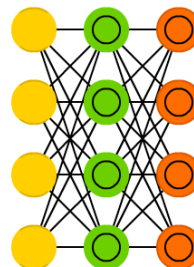
Gated Recurrent Unit (GRU)



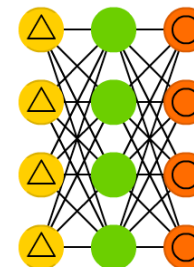
Auto Encoder (AE)



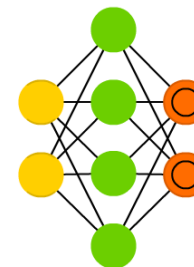
Variational AE (VAE)



Denosing AE (DAE)



Sparse AE (SAE)



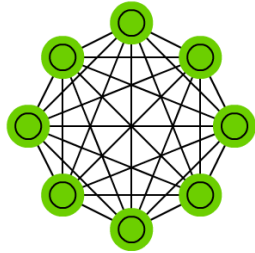
Pictures © Asimov Institute

Copyright © Alar Raabe 2018

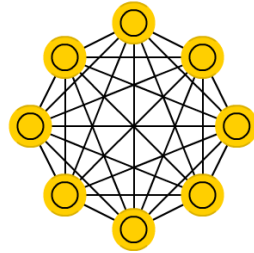


# Architecture Styles of Neural Networks

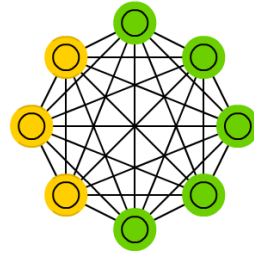
Markov Chain (MC)



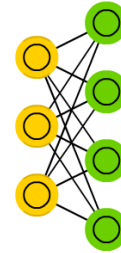
Hopfield Network (HN)



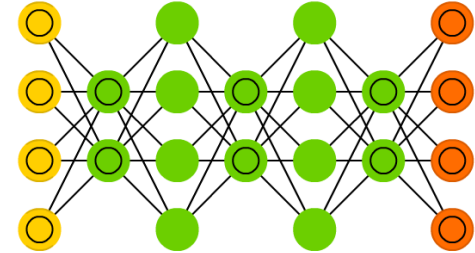
Boltzmann Machine (BM)



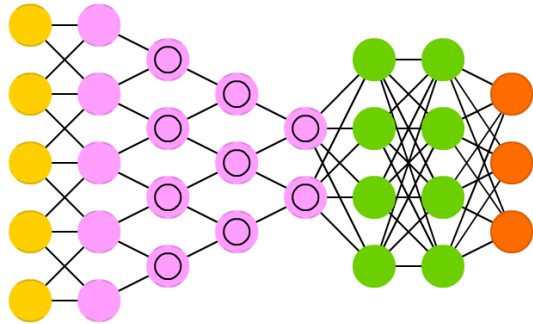
Restricted BM (RBM)



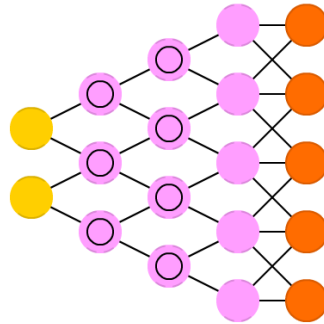
Deep Belief Network (DBN)



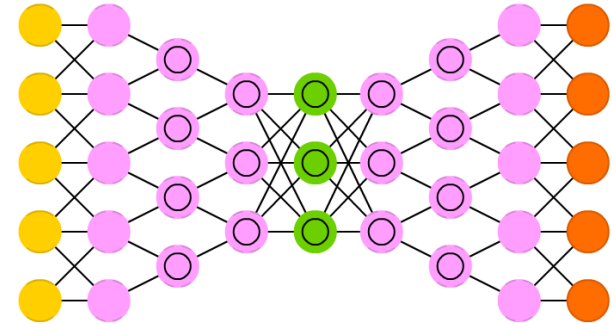
Deep Convolutional Network (DCN)



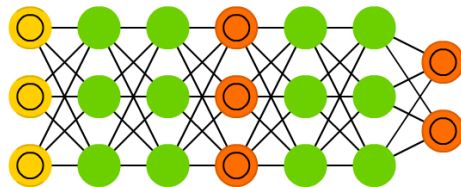
Deconvolutional Network (DN)



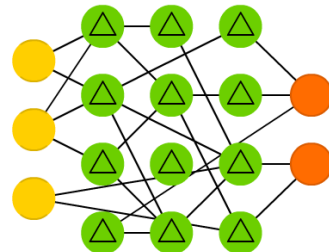
Deep Convolutional Inverse Graphics Network (DCIGN)



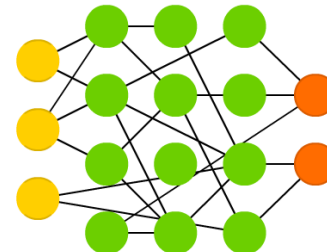
Generative Adversarial Network (GAN)



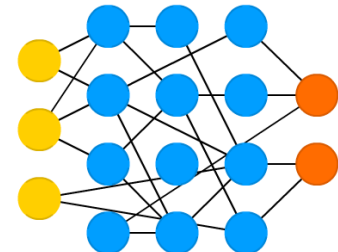
Liquid State Machine (LSM)



Extreme Learning Machine (ELM)



Echo State Network (ESN)

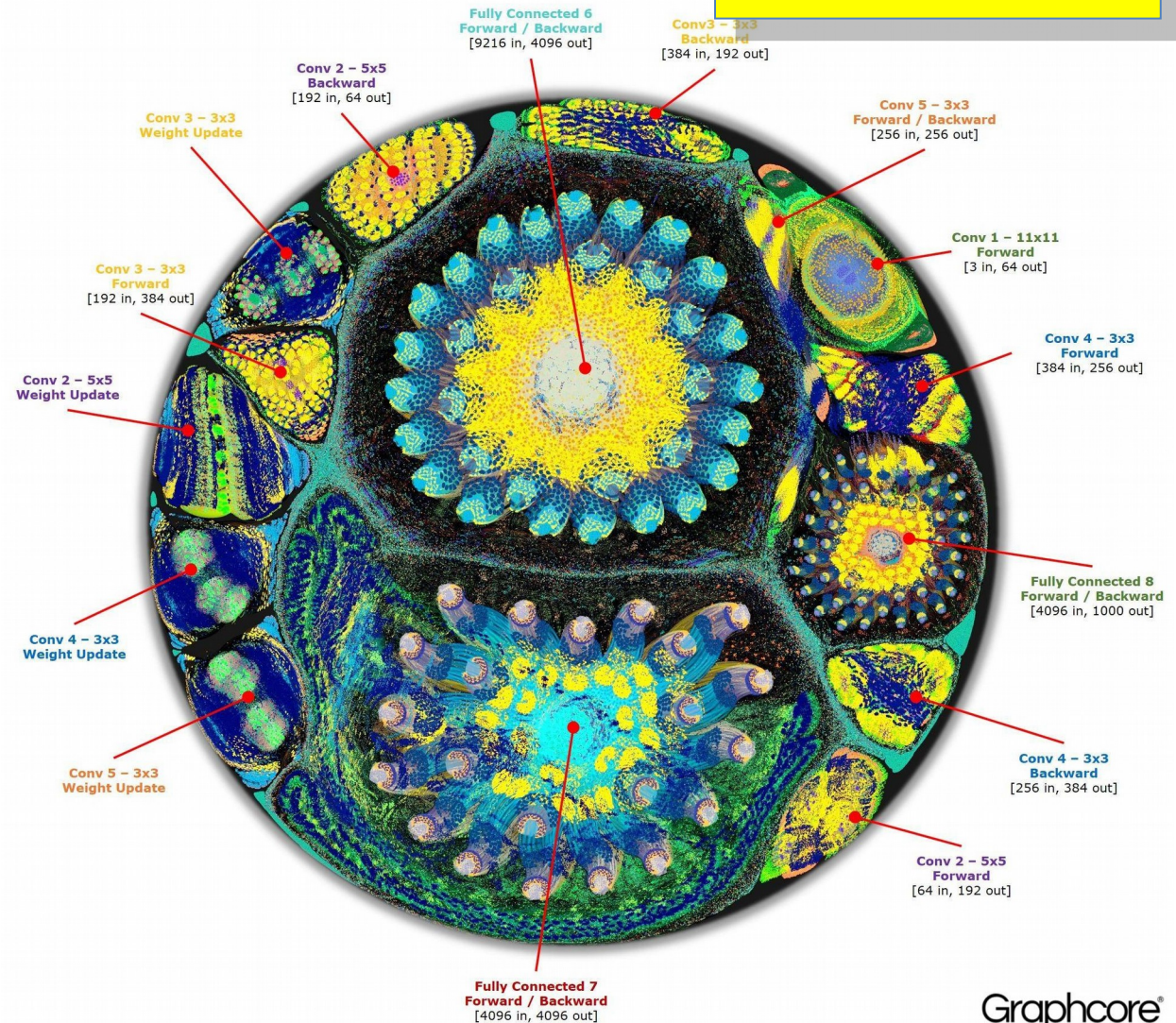




# Visualizing Architecture of Neural Networks

“MRI” of Robot !

- Visualization of Machine Learning Process



Graphcore

# Content

- Architecting for cloud
  - principles
  - design patterns
- Architecture of adaptive systems
  - reference model of adaptive systems
  - self-awareness
- Architecture of AI
  - machine learning systems
  - cognitive system architecture (example)
  - neural networks (architecture styles of neural networks)
- Conclusions

45. The architect allows things to happen. He shapes events as they come. He steps out of the ways and let the design speak for itself.

Lao Tsu (by Philippe Kruchten)

# Conclusions

The best way to predict the future is to invent it

Alan Kay, 1971

- Cloud is not under your control
  - build for unattended working in unknown, volatile and unsecure environment
- Adaptive systems need to be self-aware
- In machine learning systems data and models are entangled
- External world is changing – artificial intelligence models get out of phase

# Conclusions for the whole Course

Architecture is the important stuff – whatever that is !

Ralph Johnson

- **Software Architecture** is a
  - *fundamental conception* of a software system in its **environment** embodied in **elements**, their
  - **relationships** to each other and to the environment, and **principles** guiding software system design and evolution
- **Architecture** with desirable properties doesn't emerge itself, it **needs to be designed**
- Value of Software Systems Architecture
  - **Designing architecture allows us to address concerns** and to achieve required and desirable properties of software systems
  - Architecture **allows us to reason** (i.e. answer questions) about the software system and its properties beforehand (without building and testing the actual system)
- Value of Software Systems Architecture Description
  - As a document, it provides guidance for constructing and evolving the software system, and allows us to **record and communicate our knowledge and decisions** about the software system architecture
- Software Architecture creates choices/options, which have value – **designing and building an architecture is an investment activity**
- Understand the larger context and **isolate your system from the environment**

55. The architect lets all things come and go effortlessly, without desire. He never expect results; thus he is never disappointed. He is never disappointed, thus his spirit never grows old.

Lao Tsu (by Philippe Kruchten)

**Thank You!**

# Questions

---

---

- What are main design constraints for the cloud (ready) software systems?
- List main principles for architecting software systems for cloud?
- ...
- What are the additional parts of adaptive (software) systems? How adaptive (software) systems architecture differs?
- What are the main parts of learning systems?
- How to integrate artificial intelligence (component) into a software system?
- ...

# Literature

---

---

- [https://www.gartner.com/binaries/content/assets/events/keywords/catalyst/catus8/preparing\\_and\\_architecting\\_for\\_machine\\_learning.pdf](https://www.gartner.com/binaries/content/assets/events/keywords/catalyst/catus8/preparing_and_architecting_for_machine_learning.pdf)
- <http://www.redbooks.ibm.com/redbooks/pdfs/sg248387.pdf>
- [https://medium.com/@james\\_aka\\_yale/the-8-neural-network-architectures-machine-learning-researchers-need-to-learn-2f5c4e61aeeb](https://medium.com/@james_aka_yale/the-8-neural-network-architectures-machine-learning-researchers-need-to-learn-2f5c4e61aeeb)
- <http://www.asimovinstitute.org/neural-network-zoo/>
- <https://www.graphcore.ai/posts/what-does-machine-learning-look-like>
- <https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/6255/file/tbhpi66.pdf>
- <https://www.hpi.uni-potsdam.de/giese/public/mdelab/mdelab-projects/software-engineering-for-self-adaptive-systems/morisia/>
- <http://homepage.lnu.se/staff/daweaa/ActivFORMS/Model-based-simulation.htm>
- <http://prlewis.com/files/WICSA-final.pdf>
- [https://d1.awsstatic.com/whitepapers/AWS\\_Cloud\\_Best\\_Practices.pdf](https://d1.awsstatic.com/whitepapers/AWS_Cloud_Best_Practices.pdf)
- [https://slidelegend.com/architecting-cloud-aware-applications-open-data-center-alliance\\_59ba4f921723dd2ca91c7991.html](https://slidelegend.com/architecting-cloud-aware-applications-open-data-center-alliance_59ba4f921723dd2ca91c7991.html)
- <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43146.pdf>
- 
- ... Google “architecting cloud” + “adaptive systems” + “artificial intelligence” + “machine learning” ...



MEMORANDUM

August 28, 1963

Memorandum To: Messrs. A. L. Williams  
T. V. Learson  
H. W. Miller, Jr.  
E. R. Piore  
O. M. Scott  
M. B. Smith  
A. K. Watson

Last week CDC had a press conference during which they officially announced their 6800 system. I understand that in the laboratory developing this system there are only 34 people, "including the janitor." Of these, 14 are engineers and 4 are programmers, and only one person has a Ph.D., a relatively junior programmer. To the outsider, the laboratory appeared to be cost conscious, hard working and highly motivated.

Contrasting this modest effort with our own vast development activities, I fail to understand why we have lost our industry leadership position by letting someone else offer the world's most powerful computer. At Jenny Lake, I think top priority should be given to a discussion as to what we are doing wrong and how we should go about changing it immediately.

TJW, Jr:jmc

T. J. Watson, Jr.

cc: Mr. W. B. McWhirter