

# The business of software



*Prof. Paolo Ciancarini*  
*Software Architecture*  
*CdL M Informatica* □  
*Università di Bologna*

# Agenda

- Software as a product vs sw as a service
- The software industry
- Software standards
- Architectural and engineering issues

# The economy depends on sw

- Software allows a boy in Bologna to call a girl in NewYorkCity
- Software drives the imaging systems that allow the early detection of breast cancer and other illnesses
- Software controls the antilock breaking systems that save people lives in automobiles
- Software powers digital TV and MP3 players
- Software allows to study the human genome
- Software supports teaching and learning activities
- Software allows us to explore and understand our universe

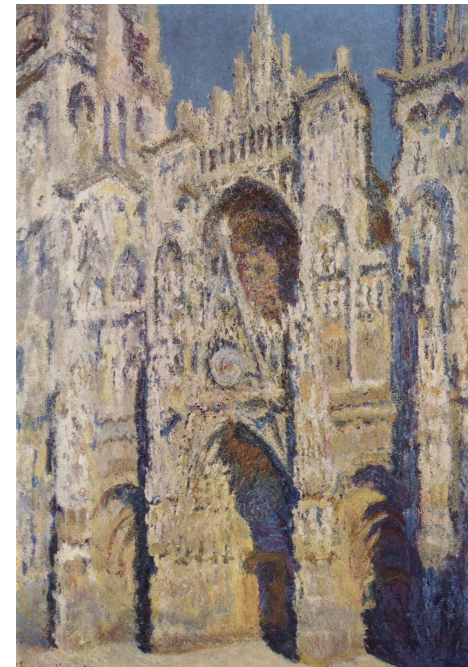
# Software can kill people?





Software and cathedrals  
are much the same—  
first we build them, then  
we pray

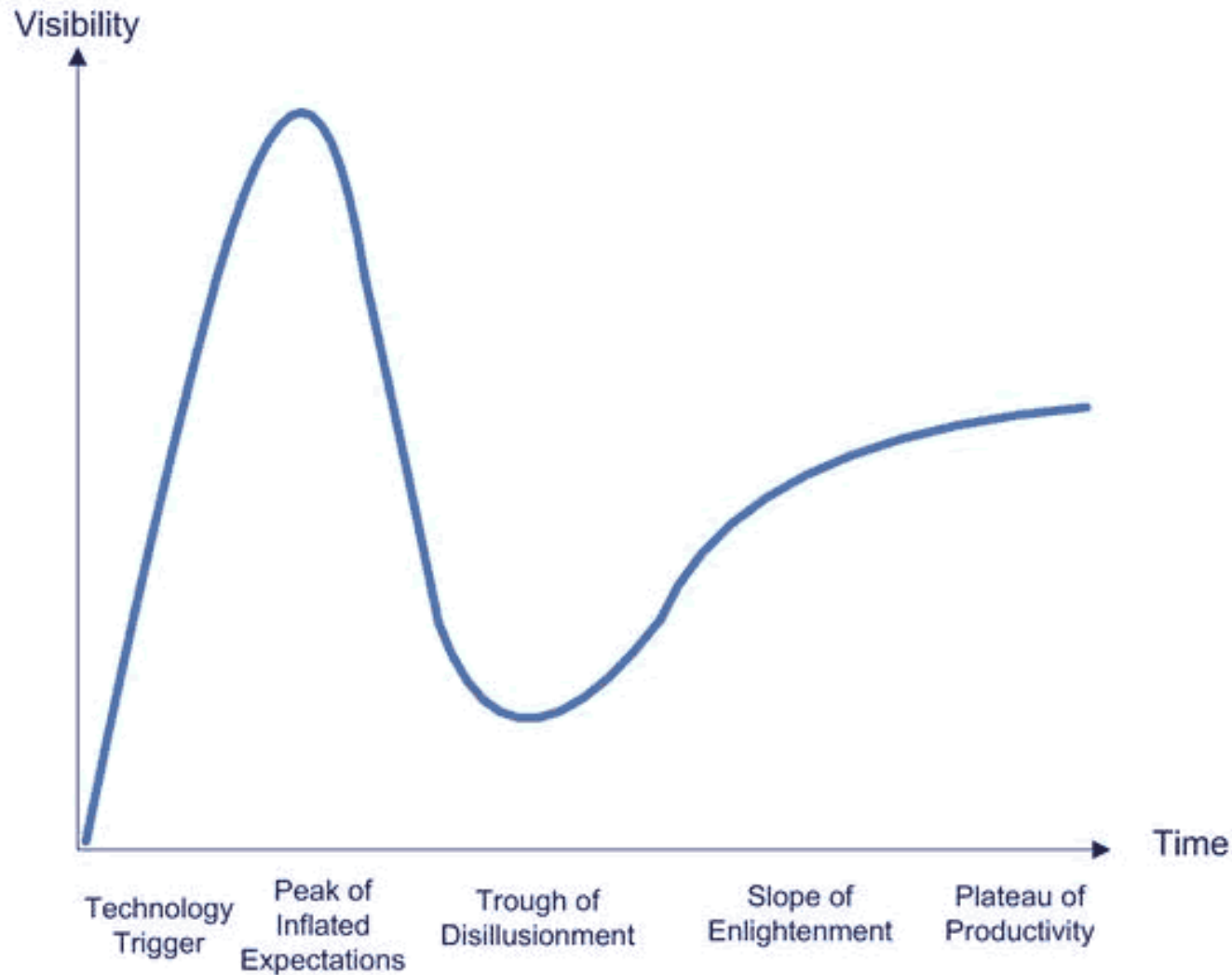
[Sam Redwine]



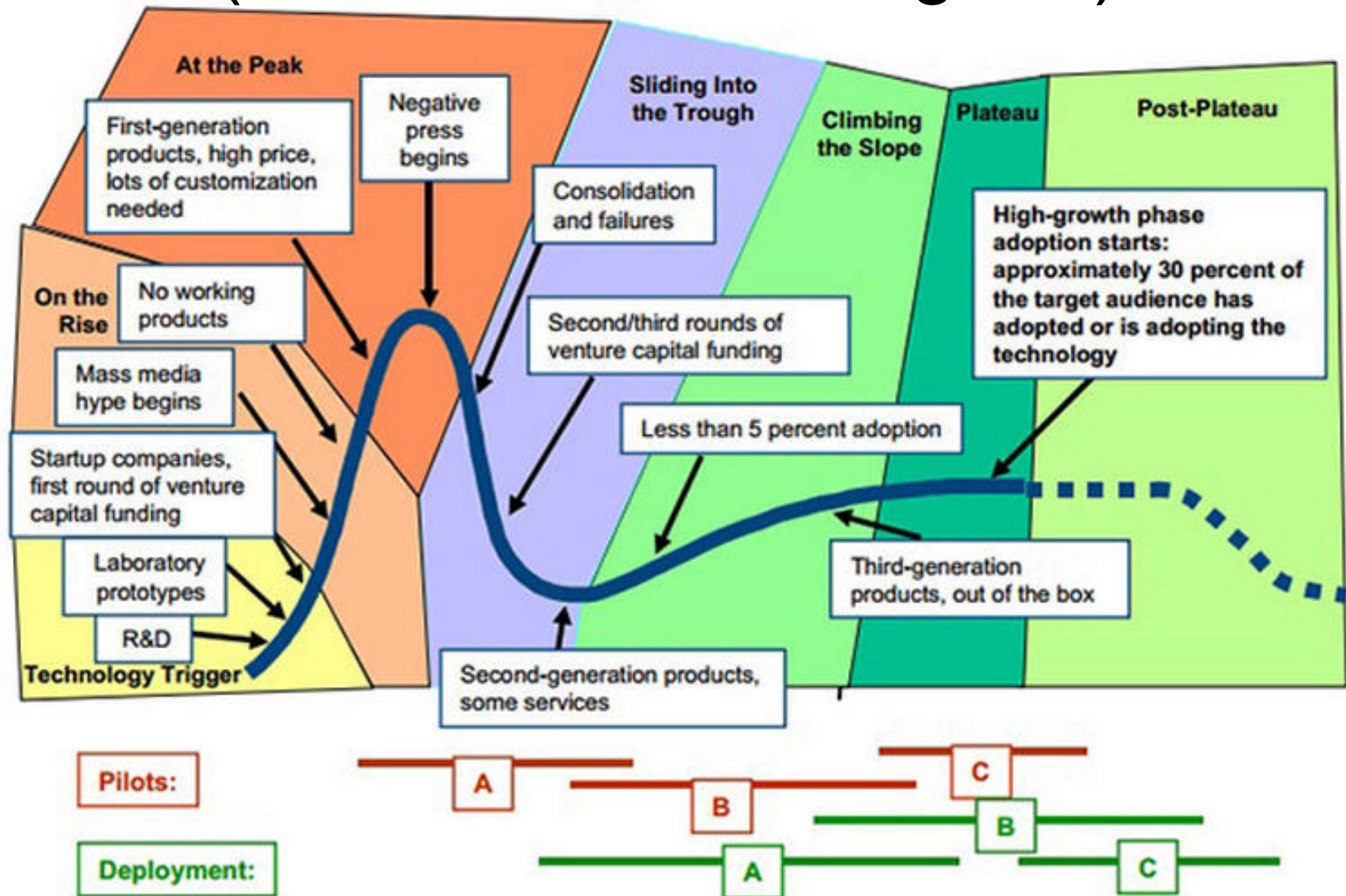
# Why is software important?

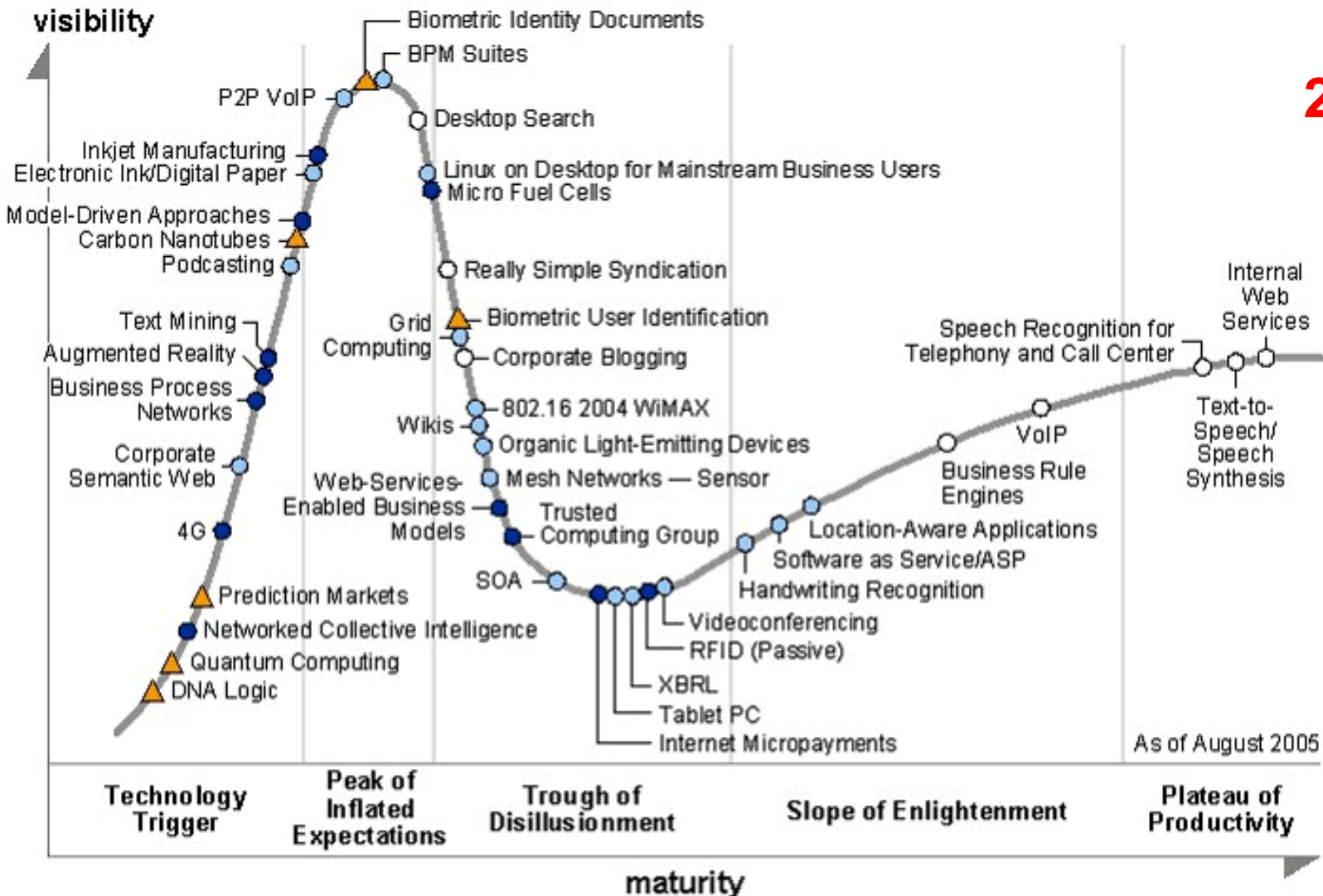
- The economy depends on software
- Software is a key component in modern products, especially when they include innovative, emerging technologies
- In the next few slides we report the **Gartner hype cycle** for the emerging technologies from 2005 to 2018

# Hype Cycle of emerging technologies (according to Gartner)



# Hype Cycle of emerging technologies (how to read the diagram)





**Plateau will be reached in:**

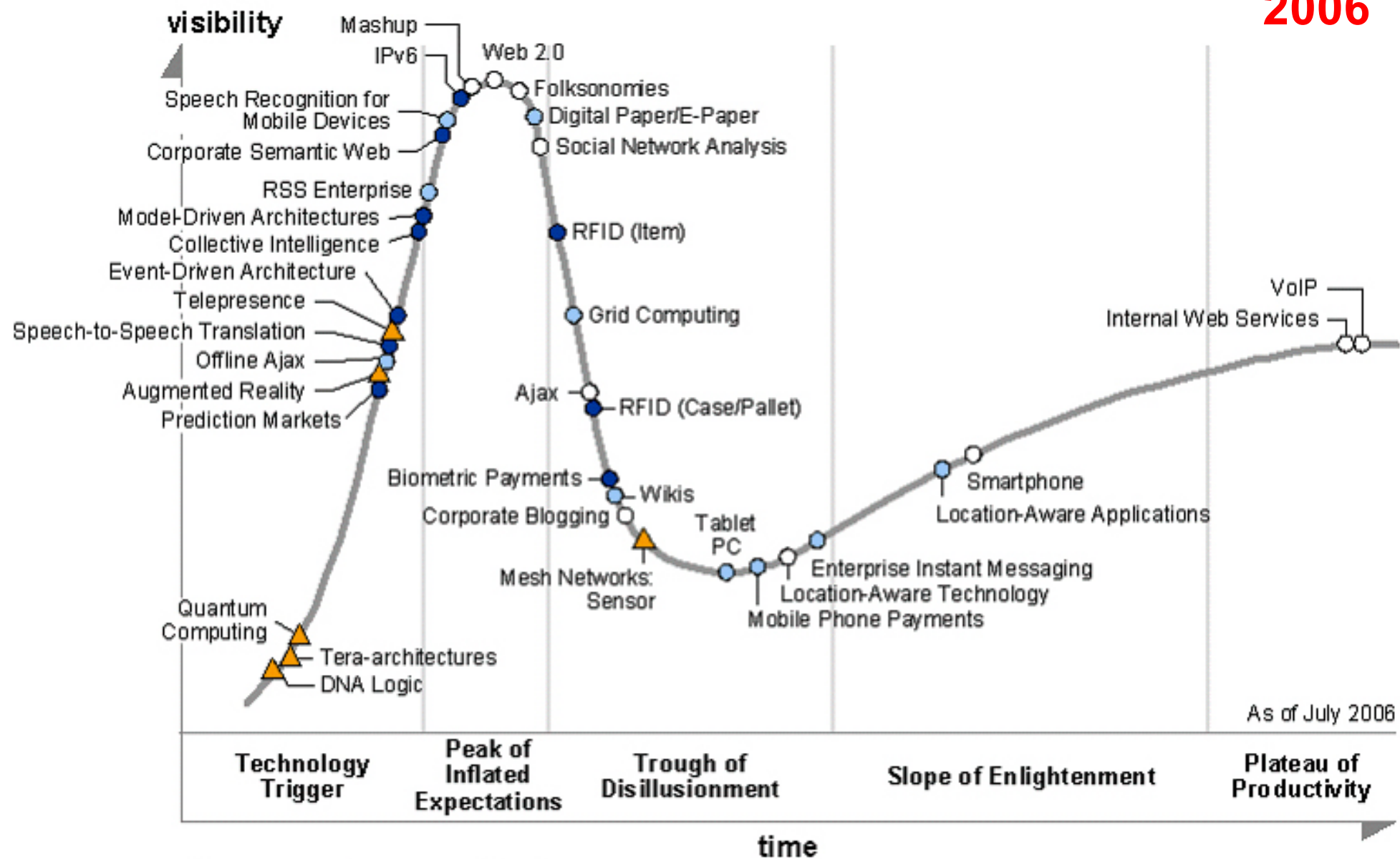
- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ obsolete before plateau

**Acronym Key**

<b>4G</b>	fourth generation	<b>SOA</b>	service-oriented architecture
<b>ASP</b>	application service provider	<b>VoIP</b>	voice over Internet Protocol
<b>BPM</b>	business process management	<b>WiMAX</b>	Worldwide Interoperability for Microwave Access
<b>P2P</b>	peer to peer	<b>XBRL</b>	Extensible Business Reporting Language
<b>RFID</b>	radio frequency identification		



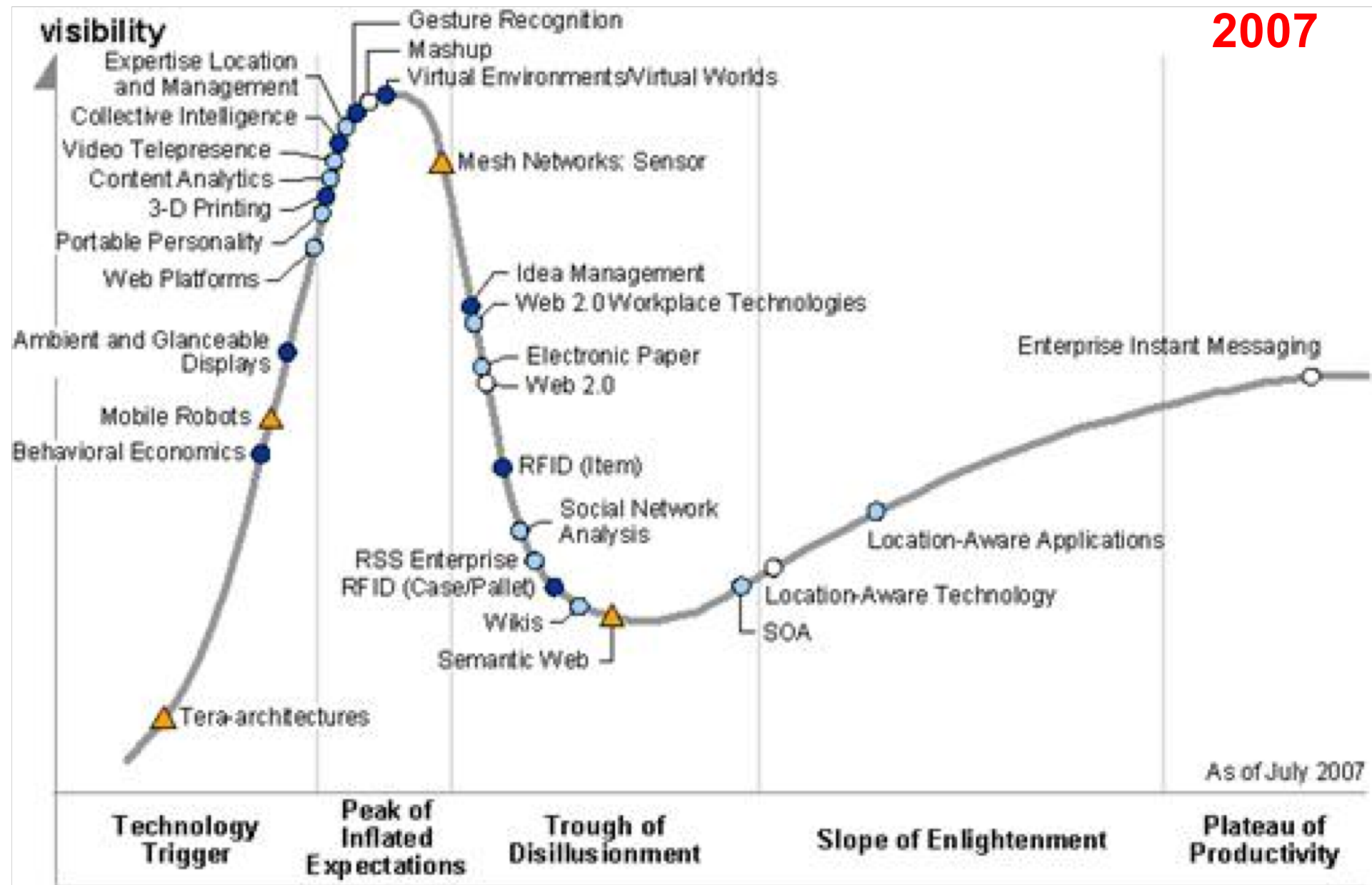
2006



**Years to mainstream adoption:**

- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ obsolete before plateau

2007

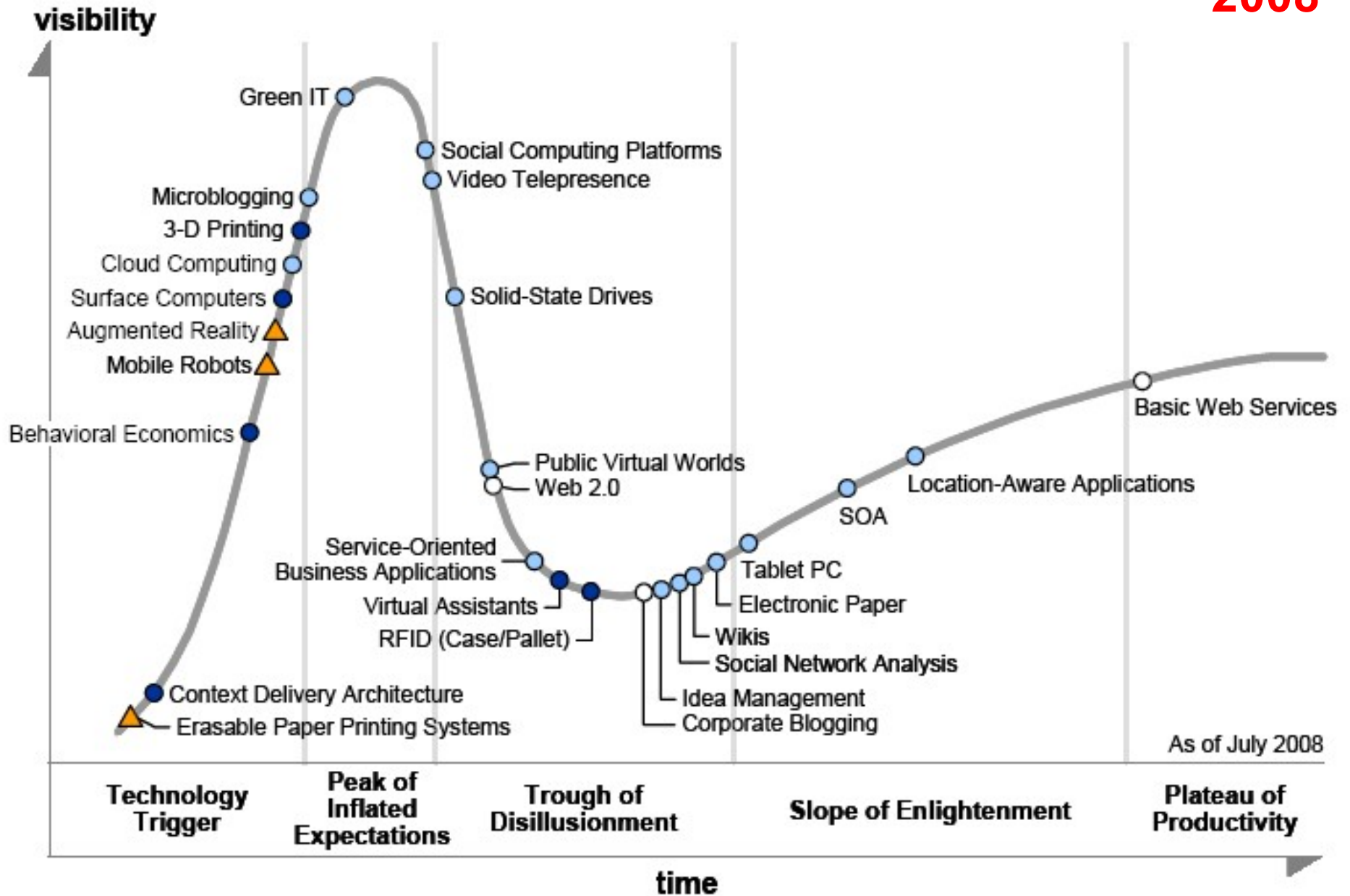


**Years to mainstream adoption:**

○ less than 2 years    ● 2 to 5 years    ● 5 to 10 years    ▲ more than 10 years    ⊗ obsolete before plateau

Figure 1. Hype Cycle for Emerging Technologies, 2008

2008



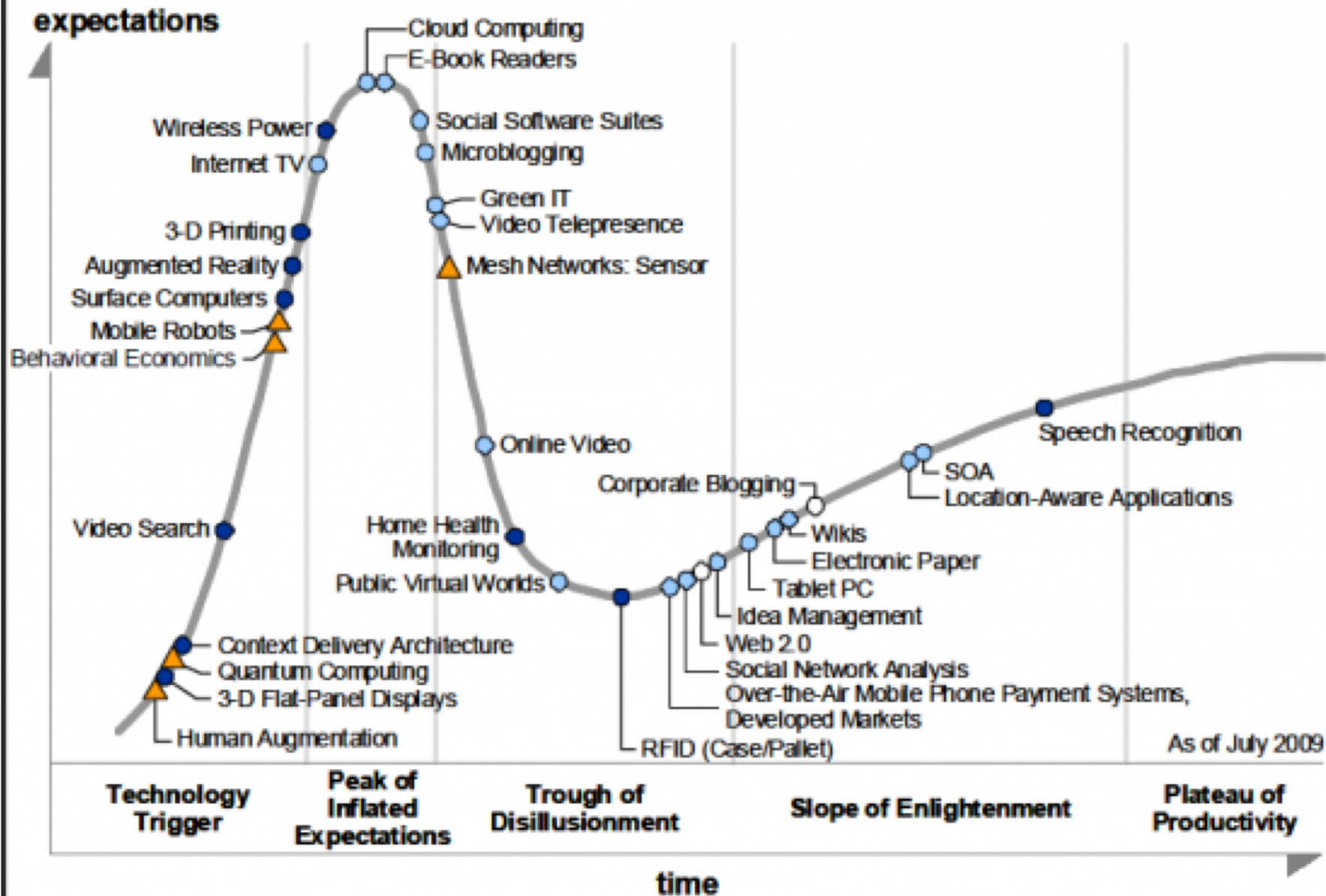
**Years to mainstream adoption:**

- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ obsolete before plateau

Source: Gartner (July 2008)



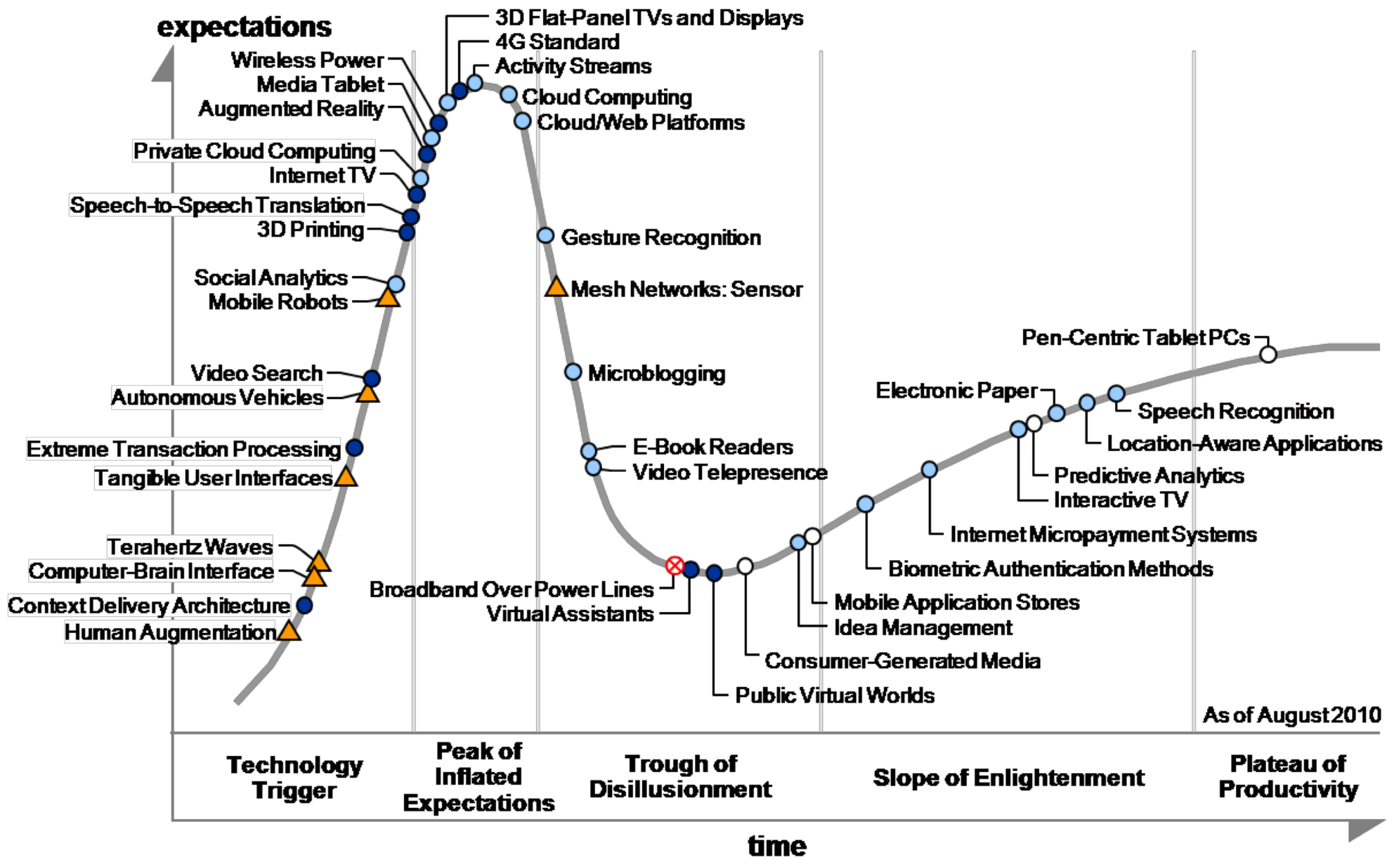
Figure 1. Hype Cycle for Emerging Technologies, 2009



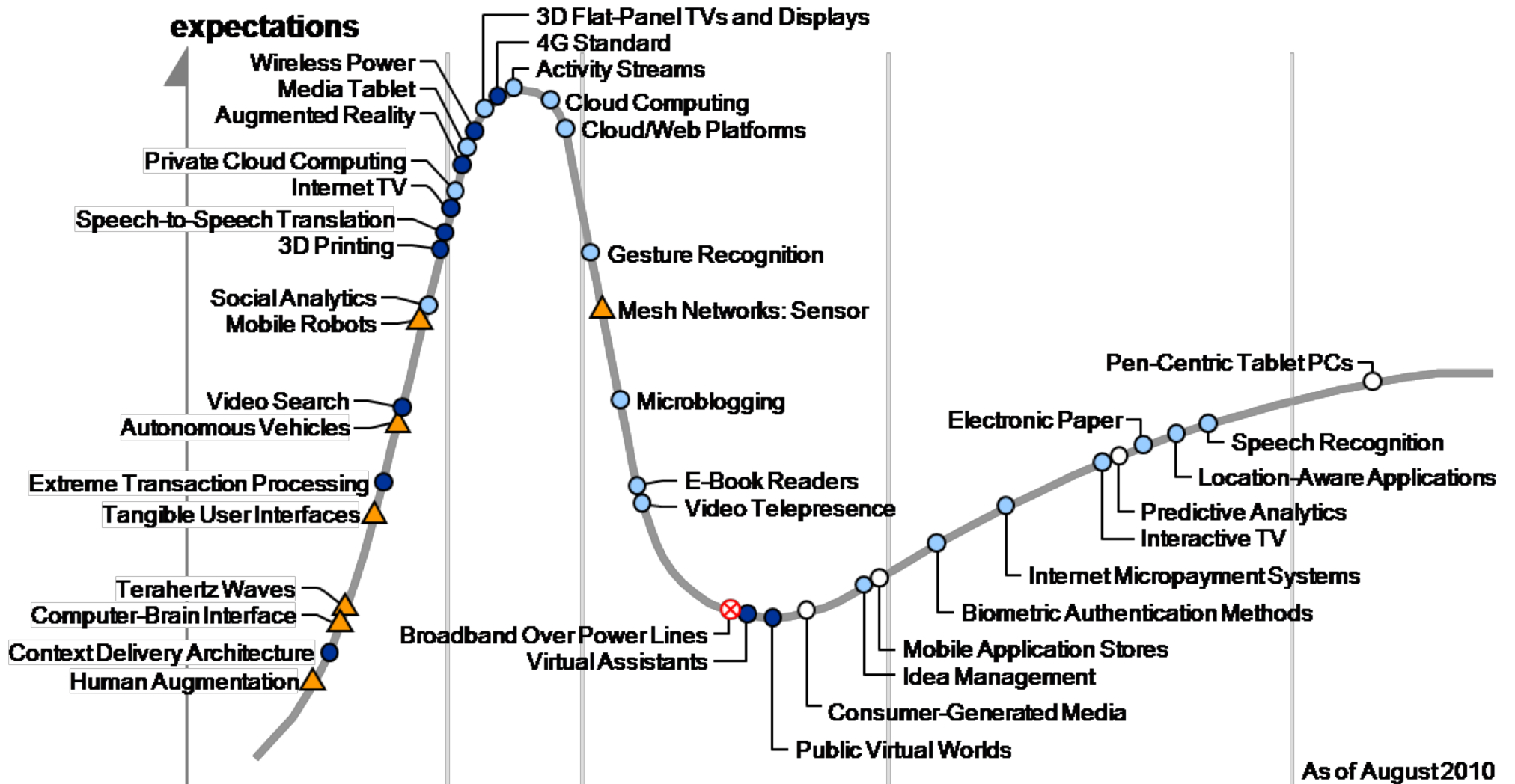
**Years to mainstream adoption:**

- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ obsolete before plateau

Source: Gartner (July 2009)



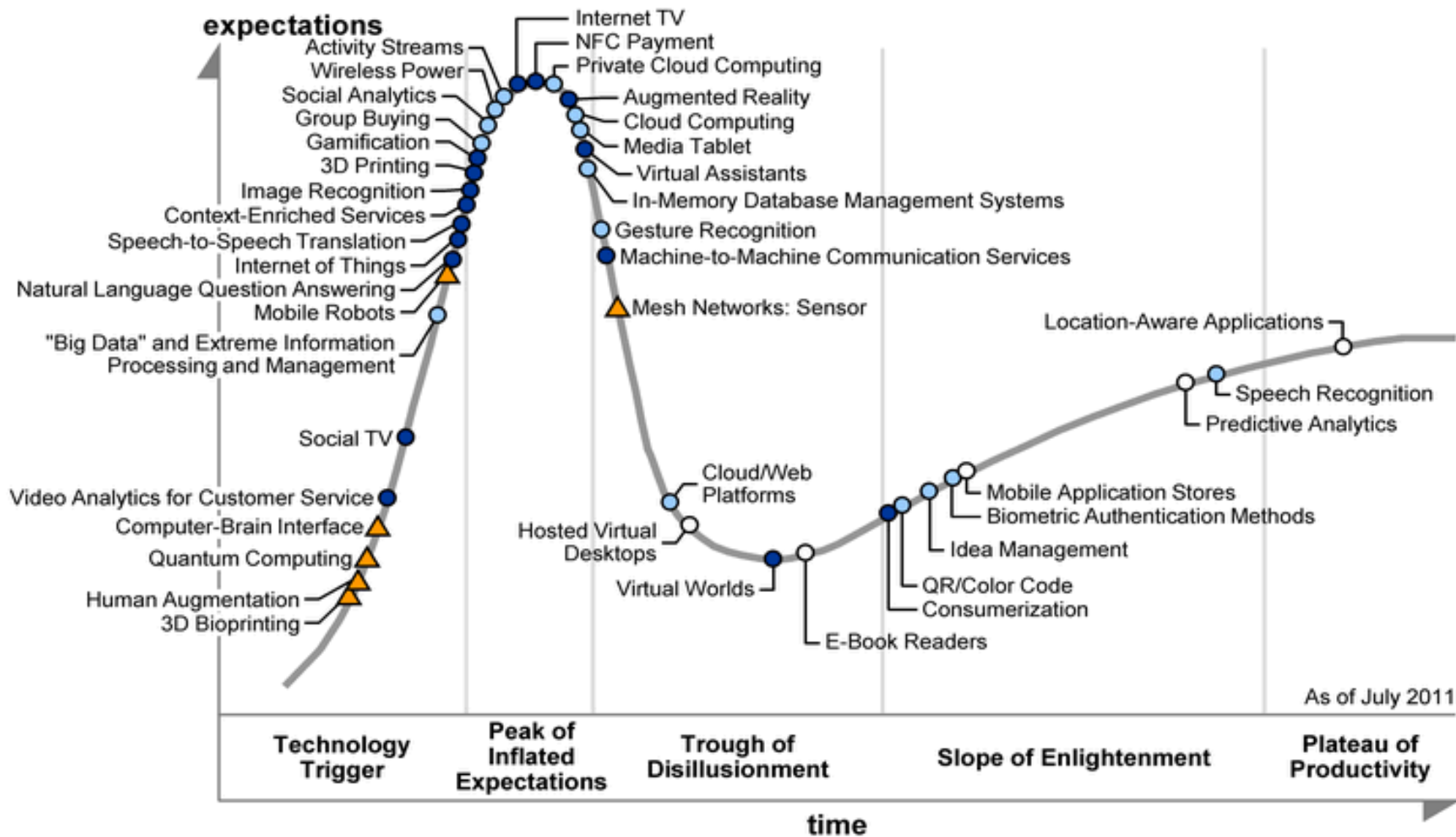
**expectations**



**Years to mainstream adoption:**

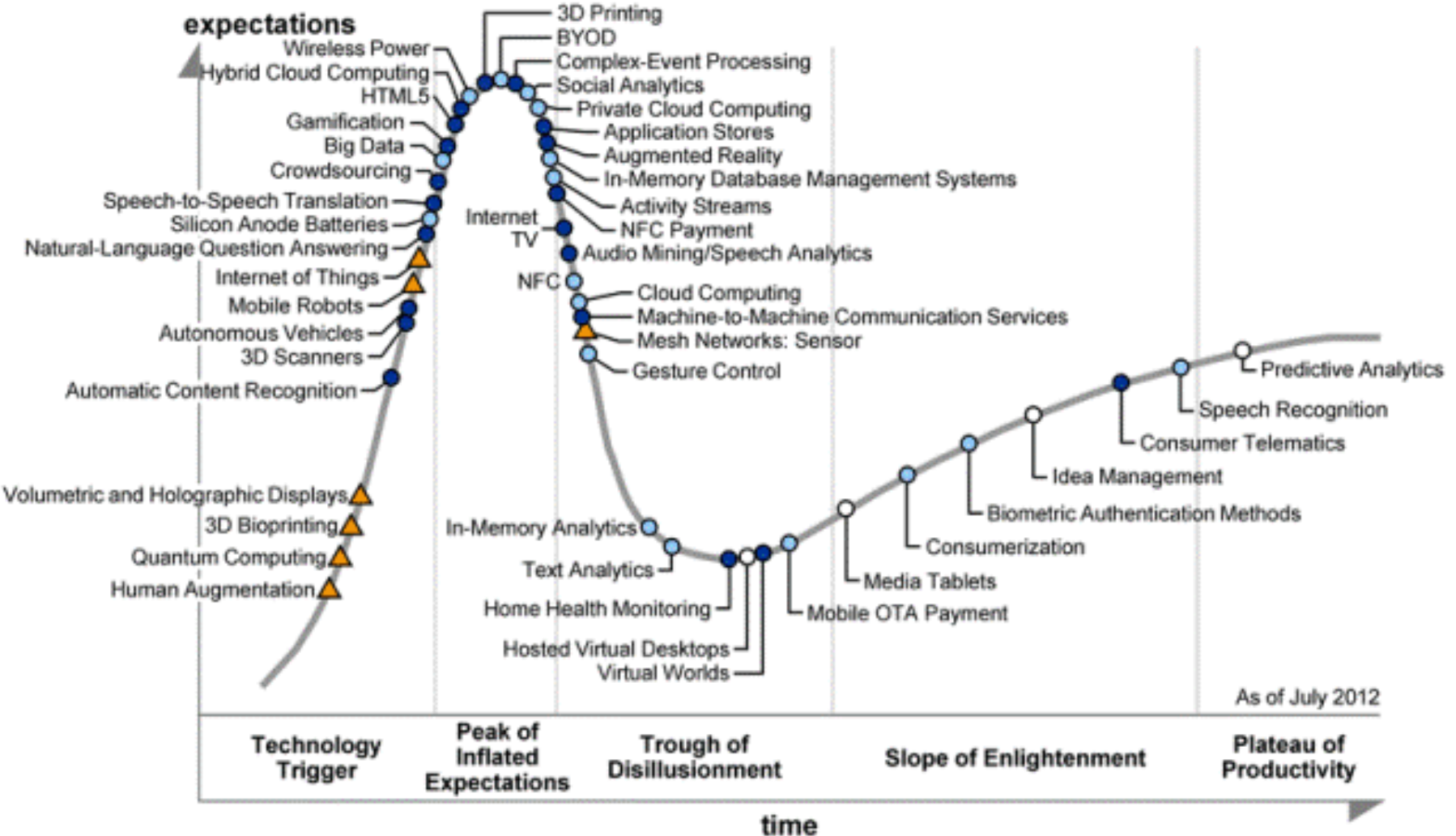
- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ before plateau

As of August 2010



**Years to mainstream adoption:**

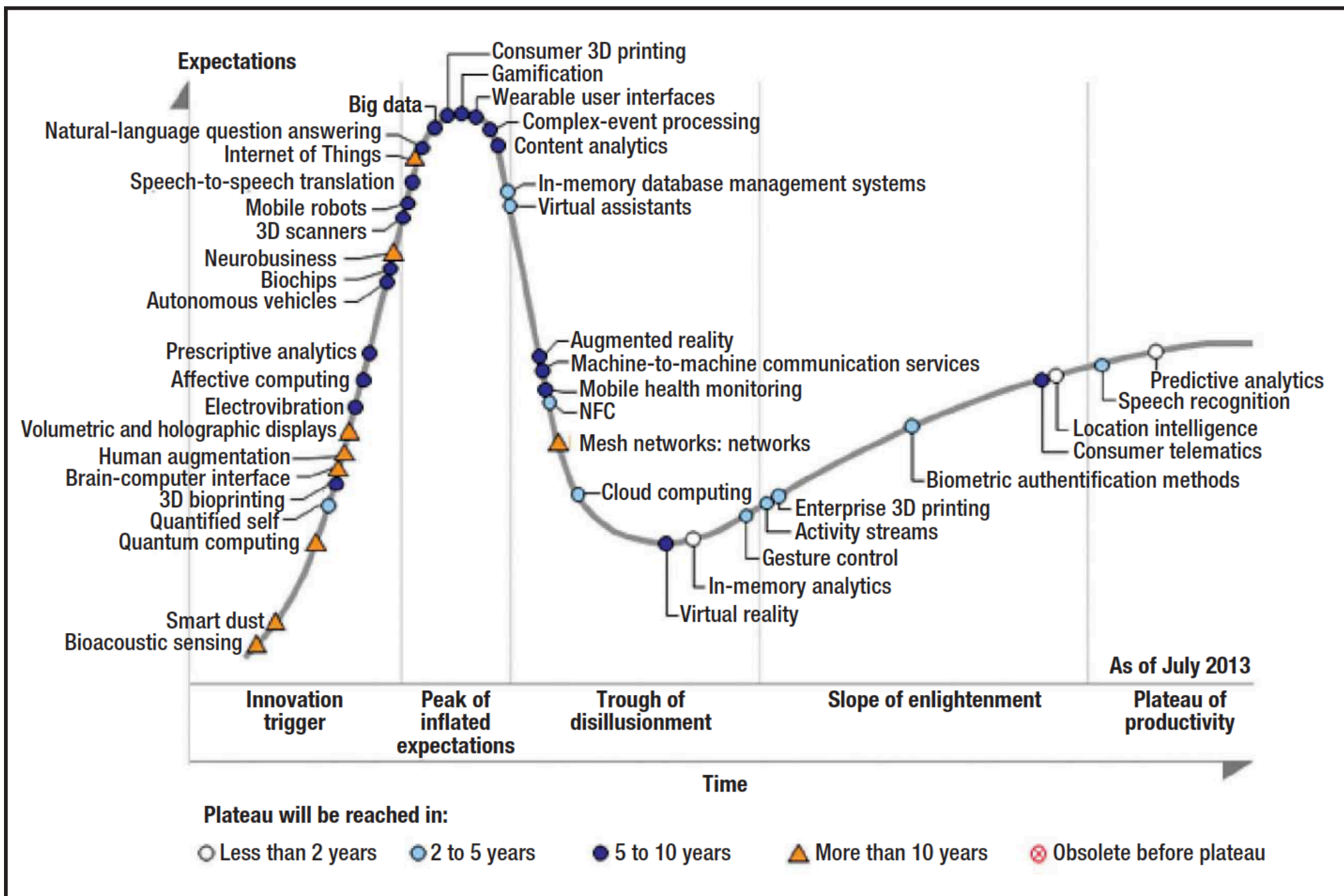
- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ obsolete before plateau

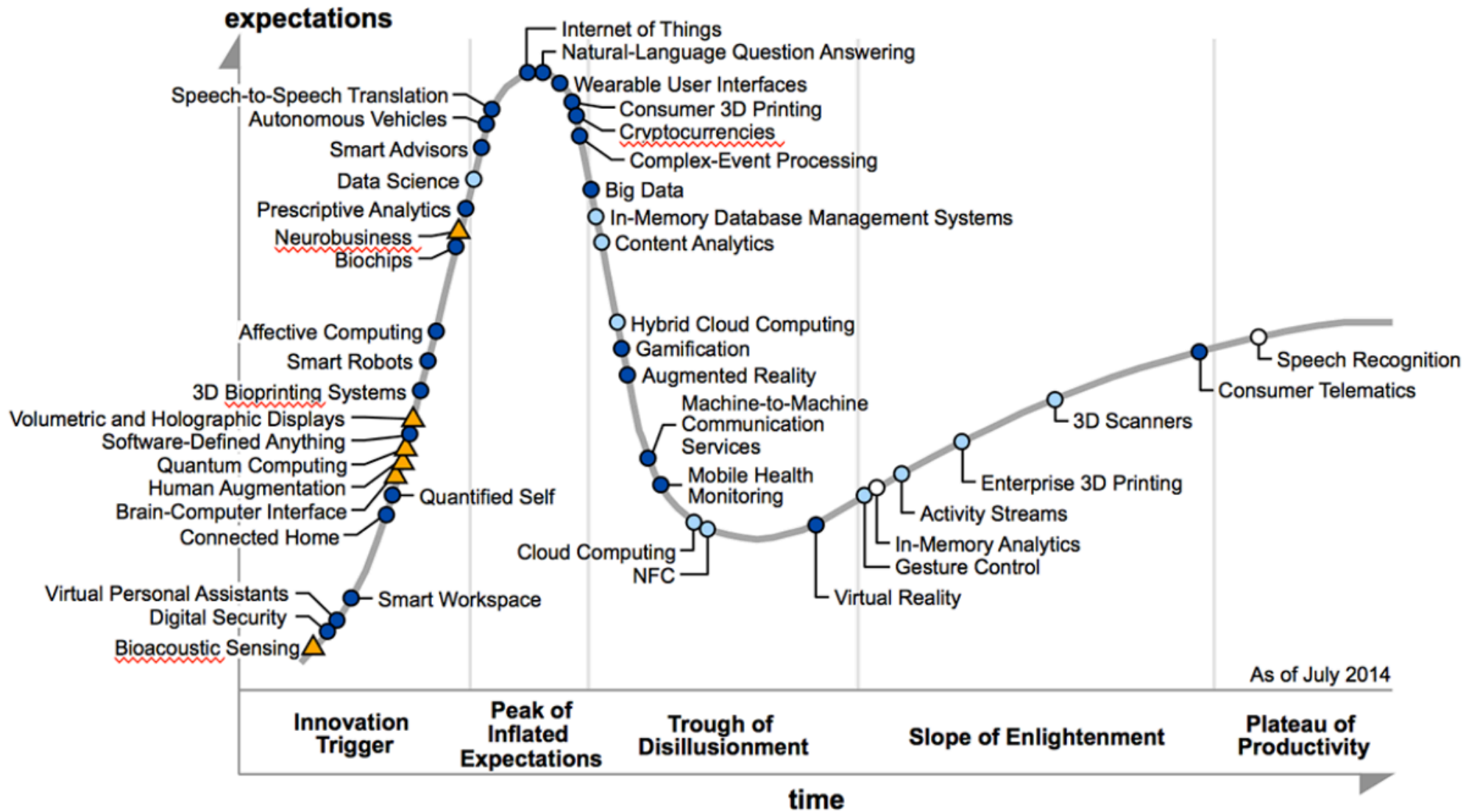


**Plateau will be reached in:**

- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ obsolete before plateau



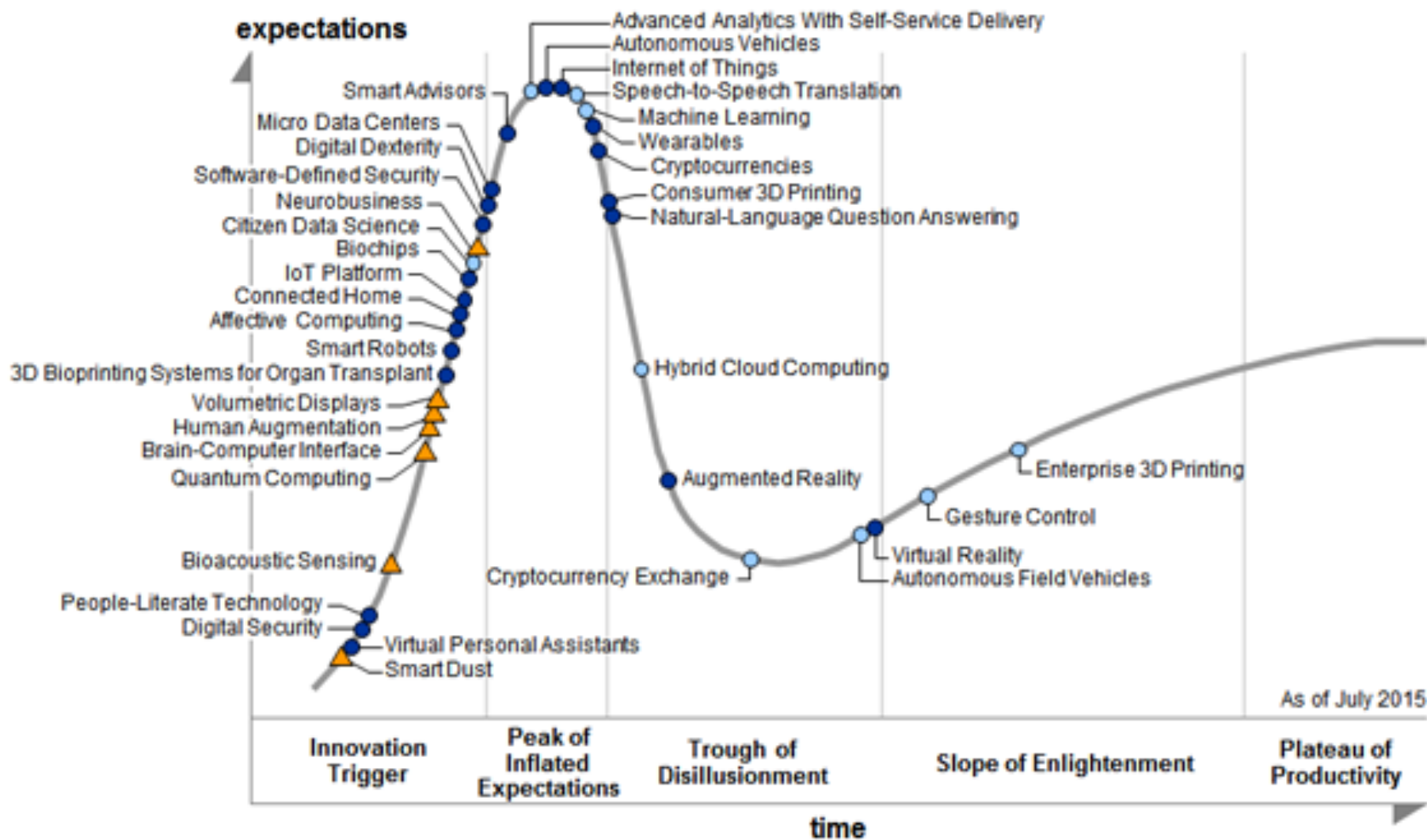




As of July 2014

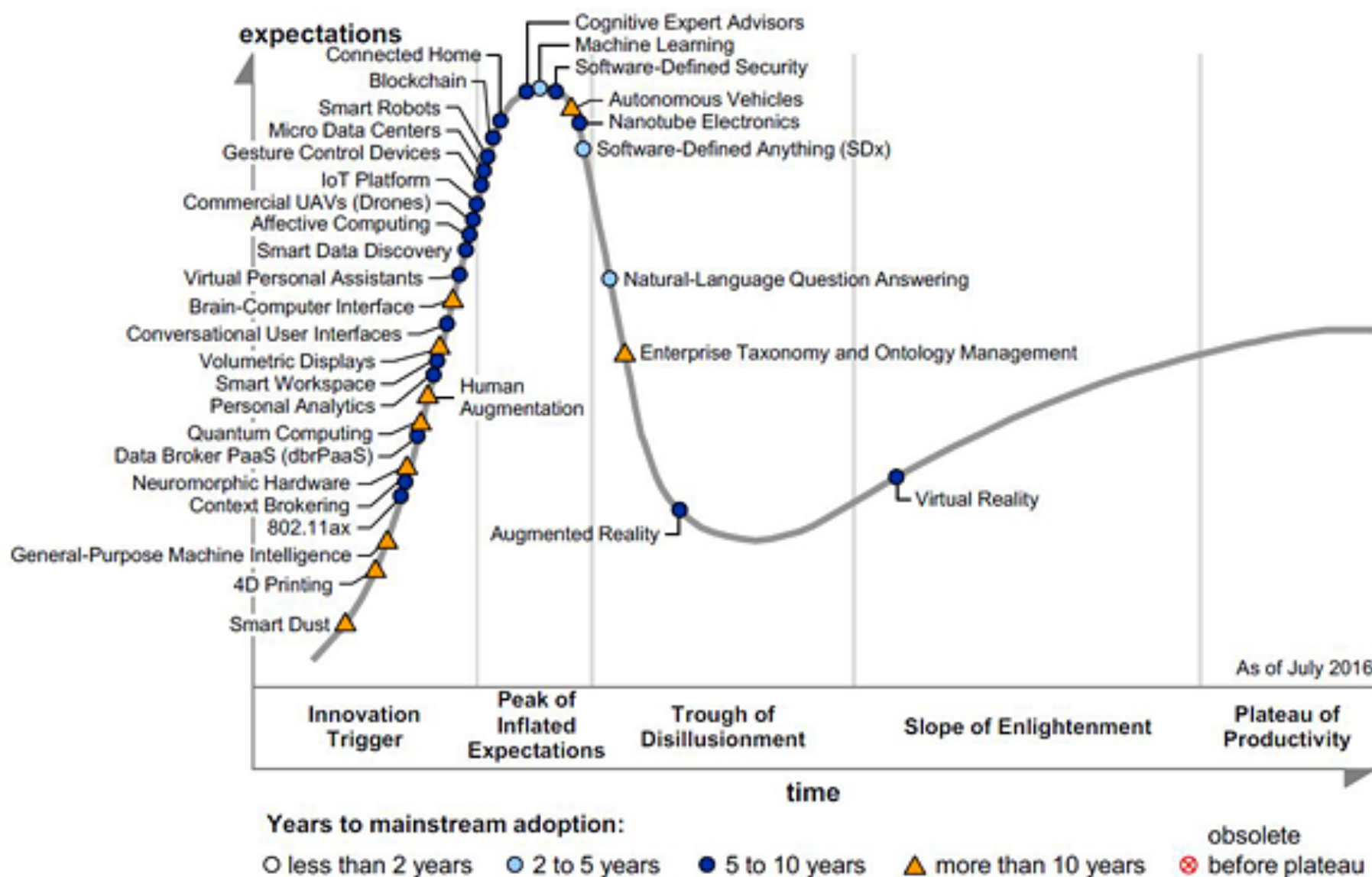
Plateau will be reached in:

- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ obsolete before plateau

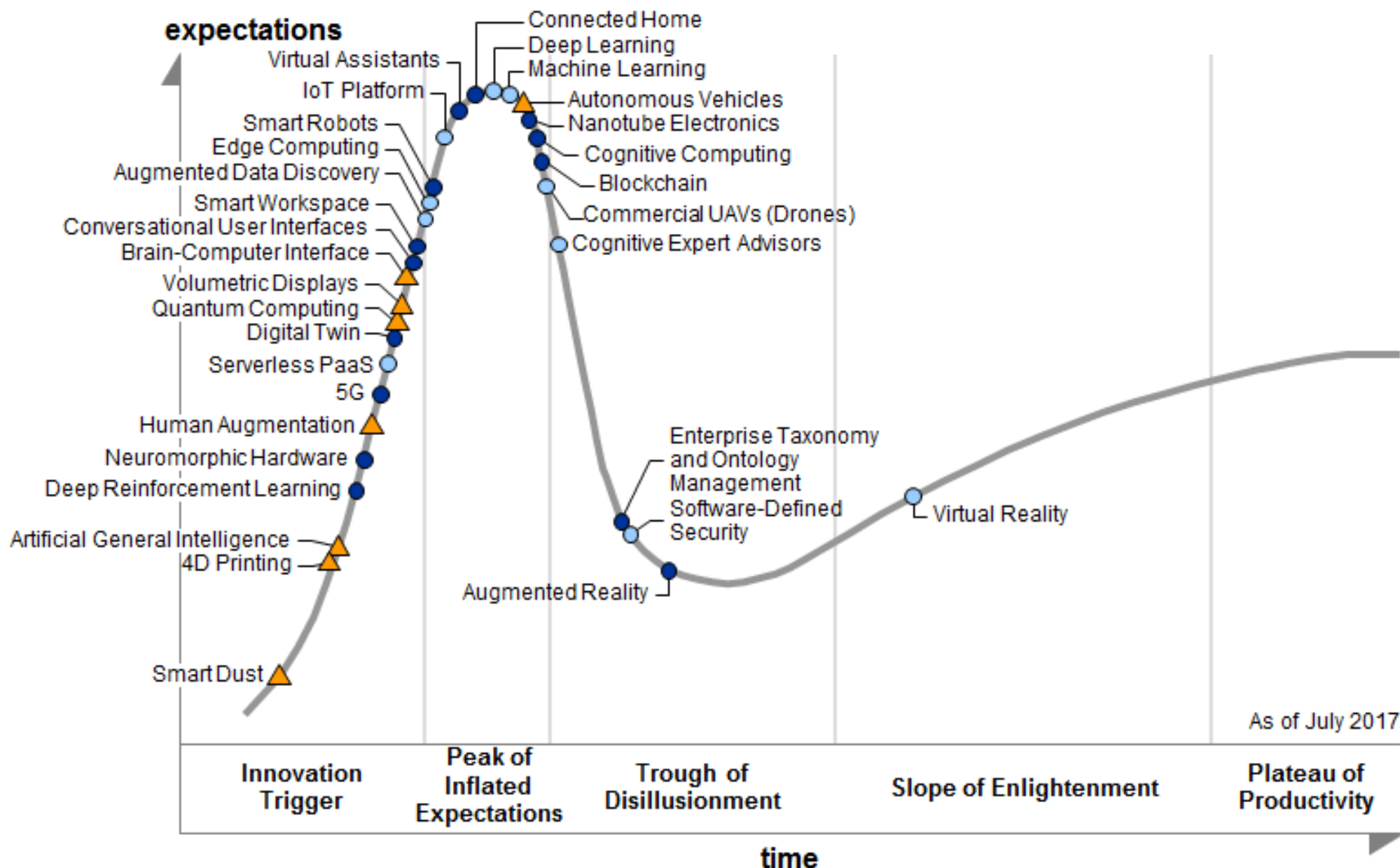


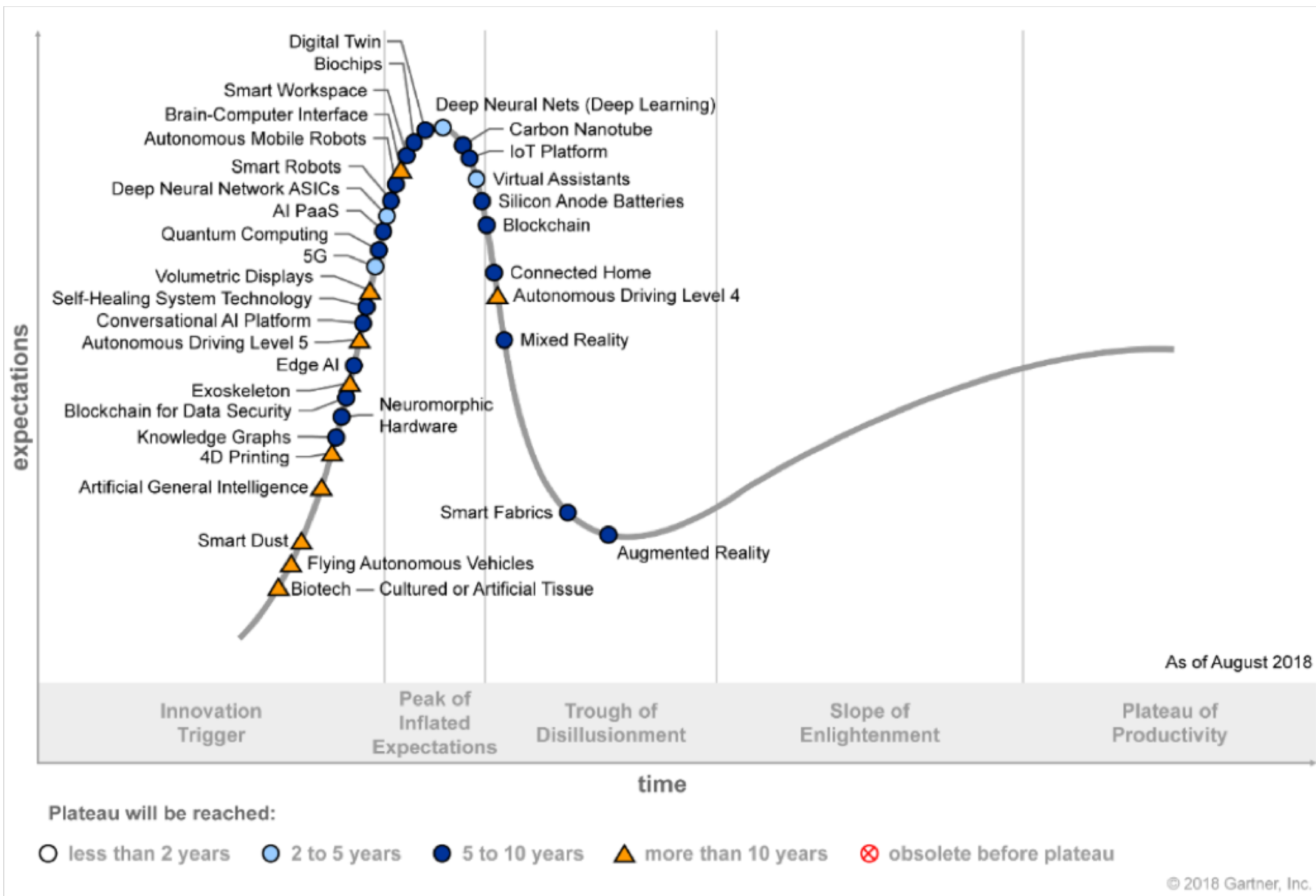
Plateau will be reached in:

- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ obsolete before plateau









# Discuss

Identify in the Gartner diagrams the technologies which are software-intensive



## “digital life” in 2020

Digital life includes all connected devices,  
and needs digital transformation of businesses and people

1	Connected Car	\$600 billions
2	Clinical Remote Monitoring	\$350 billions
3	Assisted Living	\$270 billions
4	Home and Building Security	\$250 billions
5	Pay-As-You-Drive Car Insurance	\$245 billions
6	New Business Models for Car Usage	\$225 billions
7	Smart Meters	\$105 billions
8	Traffic Management	\$100 billions
9	Electric Vehicle Charging	\$75 billions
10	Building Automation	\$40 billions

[NY Times: A Messenger for the Internet of Things](#)

[Wall Street Journal: IBM Tackles Machine to Machine Big Data Deluge](#)

Source: <http://www.globaltelecomsbusiness.com/article/2985699/Connected-devices-will-be-worth-45t.html>

# DT and the API economy



U B E R

No car



No store



No house

# API economy?



Photo service increases **x6** the revenues wrt the shops



Sensors and API's open for all trash containers to share info of when to pick up trash

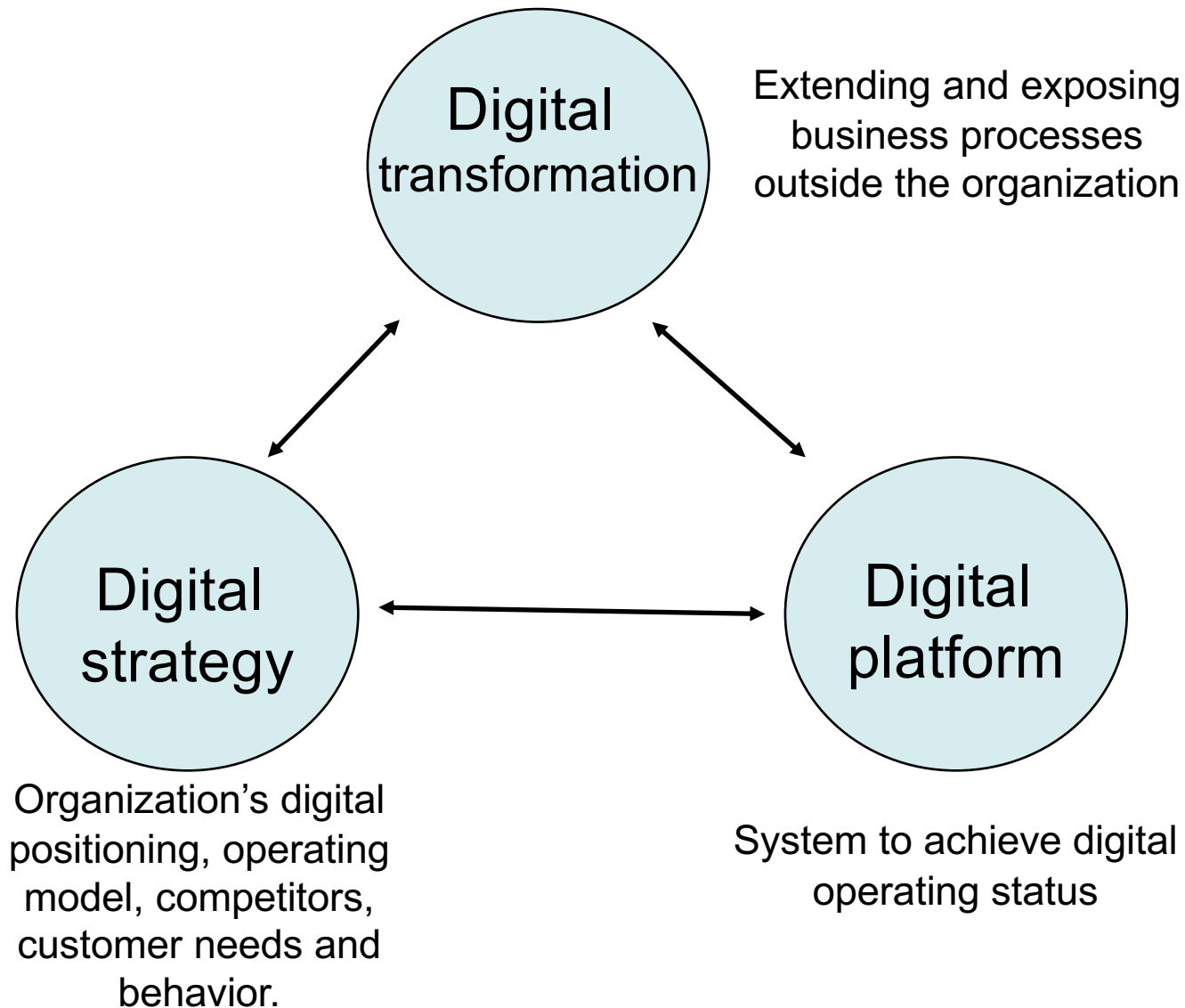


**90%** of revenues from API



**60%** of revenues from API

# Transformation = strategy + platform



# Platforms

- Technology Platforms: eg. Amazon WS
- Computing Platforms: eg. Google Android
- Utility Platforms: eg. Google search
- Interaction Networks: eg. Facebook, Telegram
- Marketplaces: eg. Amazon, Ethereum
- On-demand Service Platforms: eg. Uber
- Content Crowdsourcing Platforms: eg. Youtube, Yelp
- Data Harvesting Platforms: eg. Waze
- Content Distribution Platforms: eg. Google AdSense

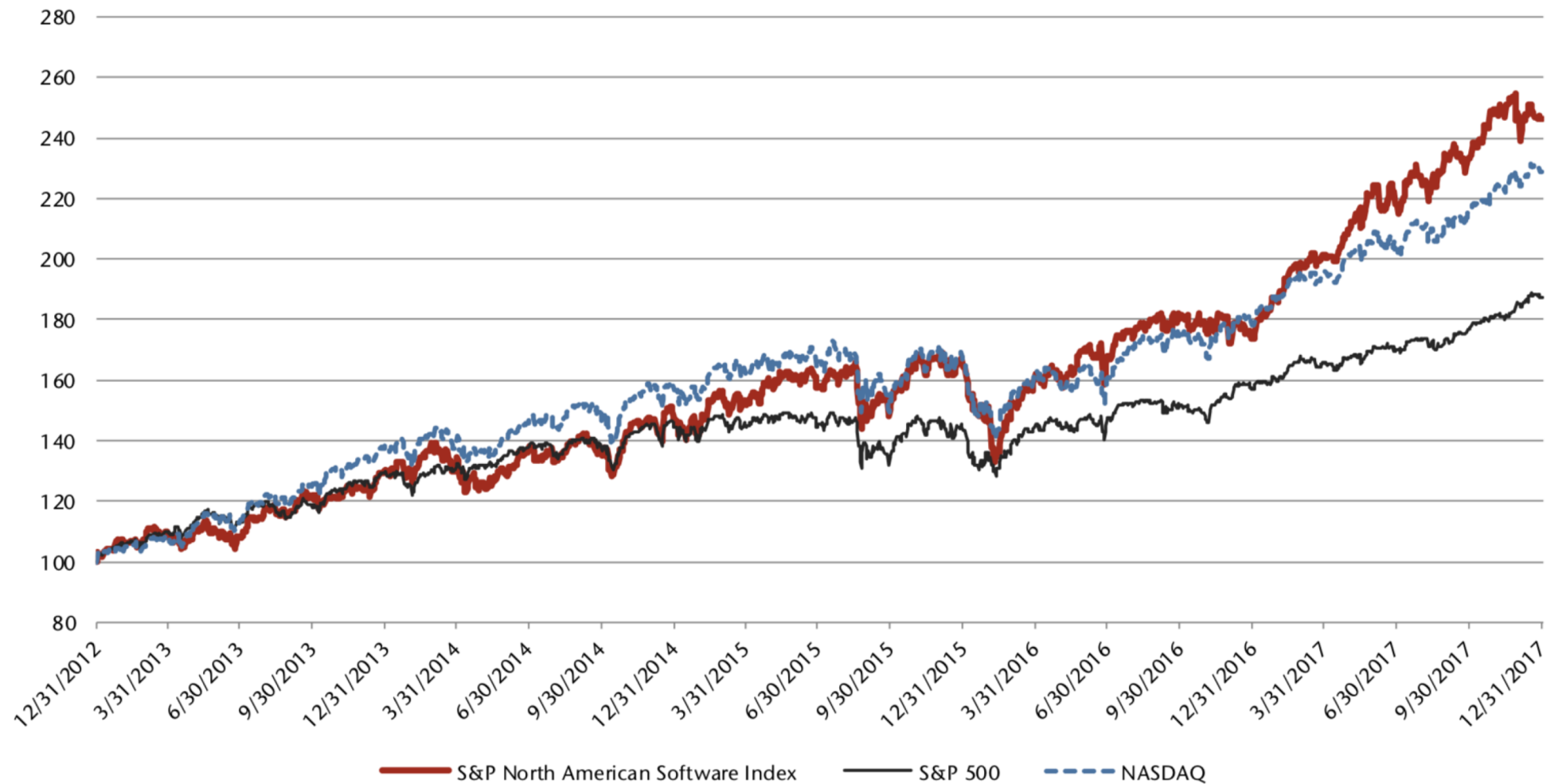


# Software platforms are huge

- In 2007 the Windows operating system scales to 60 million lines of code from 15 million lines in 1995
- in 2011, the size of software in BMW 7 Series reaches 200 million lines of code;
- 2011 the size of sw in Airbus 380 reaches 1 billion lines of code.
- 2015 Google is 2 billions lines of code

# The software industry

**Chart 4: Software Stock Performance Relative to the S&P 500 and Nasdaq from 2013 through 2017**



Source: Factset and Jefferies Research

Note: Software returns are represented by the S&P North American Technology Sector/Software Index

# Software (was) a good

- You pay, you get “something”, i.e. a **software product**, that you can use (or lend, or resell, but **not** reuse in another product)
- Examples:
  - Microsoft Office
  - Adobe Acrobat
  - IOS or Android apps
  - Console videogames
  - ...

# Software as a service

Service: the immaterial equivalent of a good

- *Software is a service at heart, albeit an automated one, but it is sold much like a manufactured good. Customers have to pay large sums of money up front, bear much of the risk that a program may not work as promised, and cannot readily switch vendors.*

The Economist, 2003

# Software is a service

We get software services in many ways

- on the Internet
- on telephone carriers
- by radio (eg. DTV, SatelliteTV, etc)











# Unicorns (major startups services)

## Legendary startups

Biggest American "unicorns"

(Date founded)

Unicorn (noun): Any startup worth more than \$1bn

		Company valuation, \$bn*, July 2015	Funds raised, \$bn	Revenue, \$m, 2014†	Employees, '000‡
 <b>UBER</b> (2009)	Taxi hailing	41	6	800	7.5‡
 <b>airbnb</b> (2008)	Accommodation for tourists and millennials	26	2.3	450	3.0
 <b>Snapchat</b> § (2011)	Ephemeral messaging app	16	1.2	nil	0.4
 <b>Palantir</b> (2004)	Big data	15	1.1	600	1.5
 <b>SPACE X</b> § (2002)	It is rocket science	12	1.1	825	3.0
 <b>Pinterest</b> (2009)	Photo sharing	11	1.3	15	0.7
 <b>Dropbox</b> (2007)	Cloud-based file sharing	10	1.1	400	1.5
 <b>wework</b> § (2010)	Office space provision	10	1.0	145	0.4
 <b>theranos</b> (2003)	Diagnostics through blood sampling	9	0.1	45	0.2
 <b>Square</b> (2009)	Mobile-payments system	6	0.6	900	1.3

Sources: CB Insights; Mattermark; PrivCo; *The Economist* \*Latest post-money †Estimate ‡Drivers are not employees §Not in Silicon Valley

# Many kinds of software

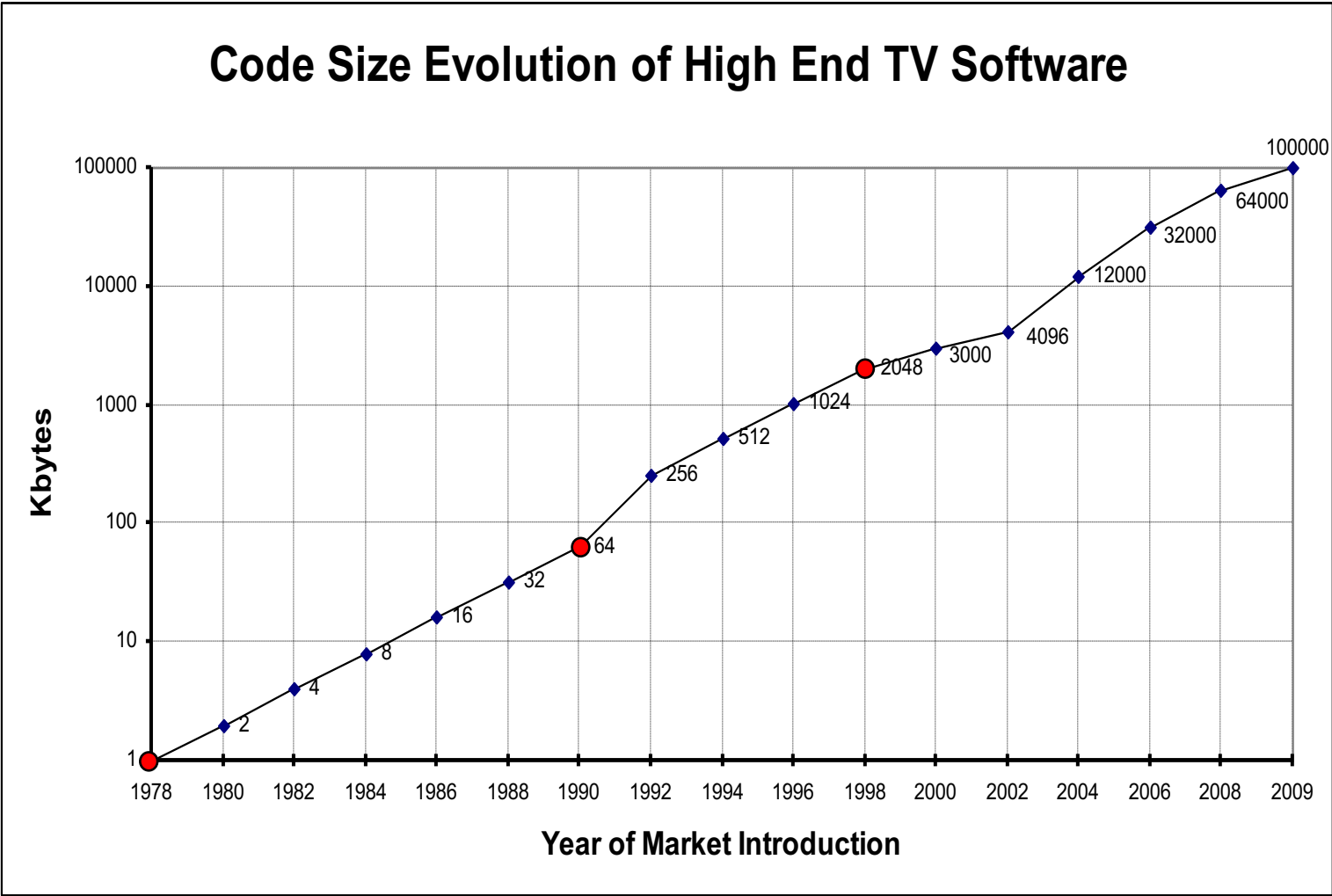
- Middleware
- Embedded
- Open source
- Web Services
- Mobile (eg. applet)
- Data mining (eg. Search engine)
- Agents
- Social software (eg. Web 2.0)
- Software Ecosystems
- ...

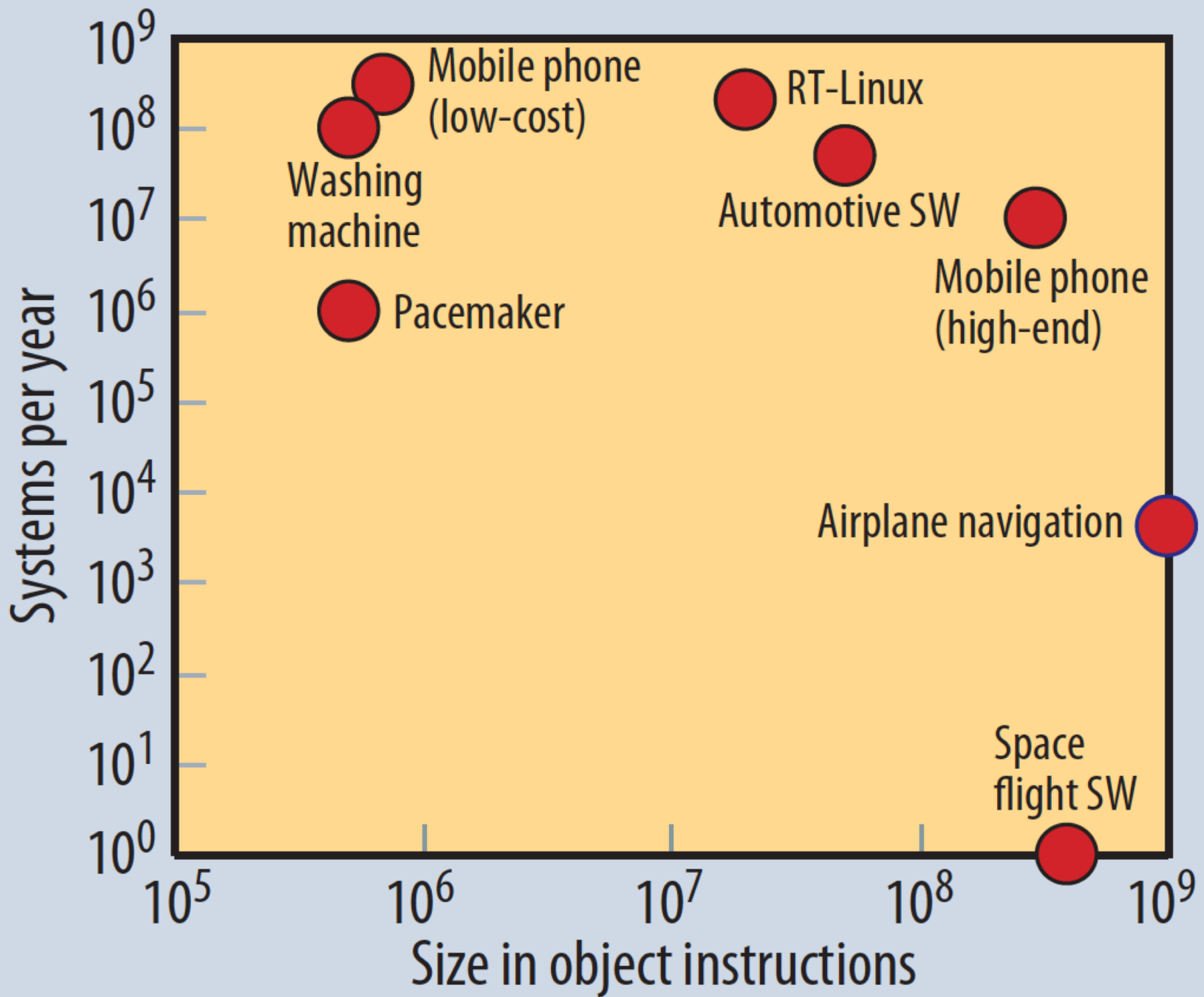
# Embedded software

- *Within only 30 years the amount of software in cars went from 0 to more than 10,000,000 lines of code*
- *More than 2000 individual functions are realized or controlled by software in premium cars, today*
- *50-70% of the development costs of the software/hardware systems are software costs*
- *(M.Broy, “Challenges in Automotive Software Engineering”, ICSE2006, pp33-42,2006)*



# Embedded software



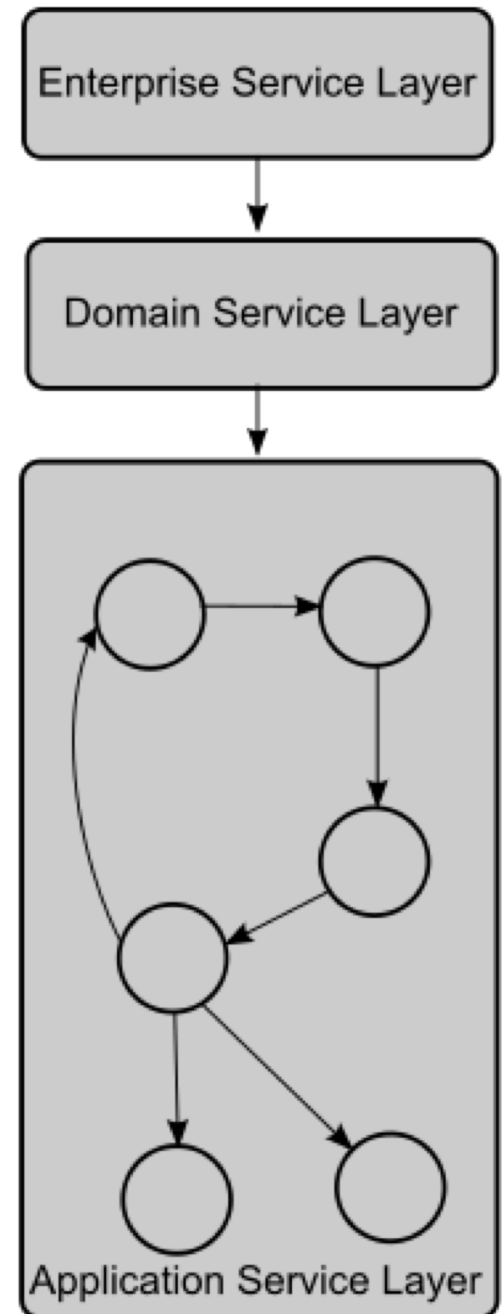


# Software components

- COTS: “component off the shelf”
- Building software systems by component integration
- Software component markets. Examples:
  - Enterprise Java Beans
  - Microsoft .NET
- Component containers: eg. Docker
- Microservices

# Service Oriented Architectures

- SOA compose different services for complementary domains
- They are often based on stacks of service layers
- SOA services feature loose coupling that can be “*orchestrated*” according to some rules of “*choreography*”



# Software standards

# Standards

- Many institutions define international product or process standards for the global software industry
- Their goal is usually to improve the quality of software products and their development processes

# SWEBOK: the Body of Knowledge of Software Engineering

Standard ISO/IEC TR 19759:2005

2013: Version 3

Main Knowledge areas:

- Requirements
- Design
- Construction
- Testing
- Maintenance
- Configuration management
- Process
- Tools and methods
- Quality



# Software standards

- Standard IEEE: development methods
- Standard OMG: UML and CORBA
- Standard W3C: Web technologies
- Standard OASIS: Business Process

# IEEE Standards on Software

- IEEE 610 - Standard for Glossary of Sw Eng Terminology
- IEEE 830 - Practice for Sw Reqs Specifications
- IEEE 1016 - Practice for Sw Design Descriptions
- IEEE 1012 - Sw Verification and Validation
- IEEE 1062 - Sw Acquisition
- IEEE 1063 - Sw User Documentation
- IEEE 1233 - Developing System Reqs Specifications
- IEEE 12207 - Standard for Sw Life Cycle Processes
- IEEE 1471 - Practice for Architectural Descriptions

# Software engineering and architecting

# The production of software in the contemporary world

- Single system vs product families
- Centralized vs distributed production (offshore)
- Code centric vs data intensive
- Use in lab vs use in the large
- Long planned vs continuous release
- Model driven vs test driven

# Main problems

- Building software is expensive and error-prone
- Software-intensive systems are larger and larger
- Sw systems become part of systems of systems
- Methods to design software are still immature
- Economic issues, like quality or cost estimates
- Effectiveness of product and process standards
- Effectiveness of tools and languages

# Software Engineering

- The discipline of Software Engineering studies the **methods** to build software systems and products, the **theories** at their basis, and the **tools** useful to develop and measure the qualities of software
- Software Engineering deals with *limited resources*
- It is a discipline strongly **empirical**, that is based on experience and past projects

# Software Architecture

- The discipline of Software Architecture studies the decisions which rule the design of software systems
- It is centered on the idea of reducing the complexity of software through *abstraction*, *separation of concerns*, and *reuse*
- Wrong decisions in crafting the software architecture of a system are a major cause of project cancellation
- Unfortunately, the discipline is still quite immature: it is hard to find two software architects who agree on the *right way to design* a software system



# Producing software is difficult

- Complexity derives from
  - Fast technical innovation
  - Strong international competition
  - Psychological issues
  - Organizational issues
  - Lack of professionals trained on sw design and development
- Typical failures: bad project management, wrong requirements, mediocre design, excessive costs
- Stakeholders with contrasting interests
- New projects start with high risks, scarcely analyzed

# Productivity is \*historically\* low

2003: analysis of 13.522 sw development projects in USA:

- 66% out all projects failed (no useful result)
- 82% out all projects needed more time than initially planned
- 48% out all projects produced products lacking some function required by the customers
- 55 G\$ wasted in one year

*Chaos Report 2003,  
by the Standish Group*

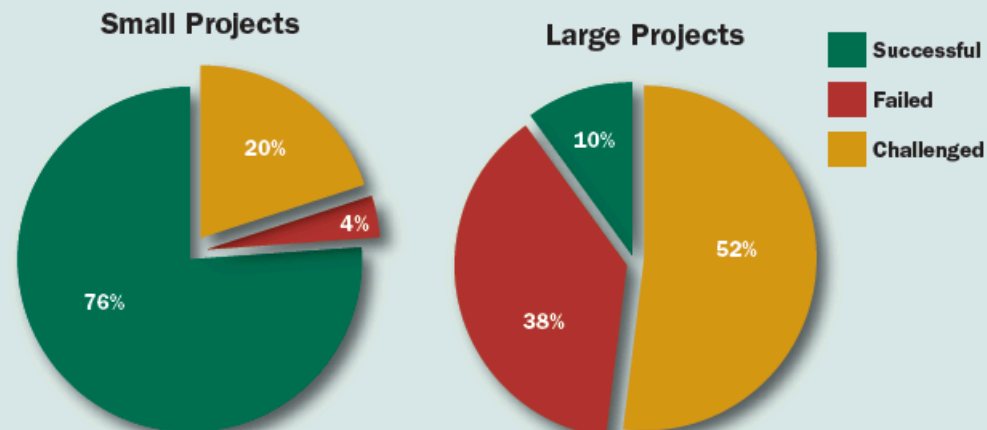
# Standish 2011-2015

## MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

*The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.*

Project resolution for the calendar year 2012 in the new CHAOS database. Small projects are defined as projects with less than \$1 million in labor content and large projects are considered projects with more than \$10 million in labor content.



# Caper Jones on sw project failures

- “As to project cancellations, we cover a wider range than Standish Group because they show only IT projects. We include embedded, systems software, web applications, IT, etc. There are some gaps because we have no data from the game industry. Our data resembles Standish for IT cancellations, but the embedded and systems worlds are a bit better than the IT world due to more effective quality controls
- 10 function points = 1.86% cancels    100 function points = 3.21%  
  1000 function points = 10.14%    10000 function points = 31.27%  
  100000 function points = 47.57%
- The canceled projects are usually late and over budget when the plug is pulled. On average a canceled project is about 10% more expensive than a successful project of the same size and type”

# The solution: software reuse

## Main techniques for software reuse

- Design patterns
- Open source
- Software cloning
- Component-based sw engineering
- Software product lines
- Software architectures

# Design patterns

- Reuse the idea, not the code
- Reusable solutions
- Goal: avoid or manage the dependencies among the components (classes, objects) of a software system

# Open source

- Let everybody contribute to improving the source code
- Reusable applications
- Goal: source code more tested and extensible

# Software cloning

- Sw clones are the result of ad-hoc reuse by copying and pasting, from simple statements to methods, classes, models, even architectures.
- Problem: intellectual property
- Problem: maintainability



# Component-based sw engineering (CBSE)

- A reuse-based approach to defining, implementing, and composing loosely coupled independent components into systems
- COTS: Component-Off-The-Shelf
- Components are substitutable, so that a component can replace another (at design time or run-time), if the successor component meets the requirements of the initial component (expressed via the interfaces)

# Software product lines

- A sw product line is a family of products which share a basic common set of reusable assets, and whose variable part can be modeled as a range of possible additional assets
- Goal: produce a new member of the family focussing on the variable part

# Software architectures

- The set of structures needed to reason about a software system, including its basic elements, the relations between them, and the properties of both elements and relations
- Goal: To provide a basis for reuse of elements, their relations, and the related decisions

# Reusing sw architectures

- The reuse of a software architecture includes:
  - Reusing its requirements and tests
  - Reusing its structure
  - Reusing its components
  - Reusing its *rationale*
- A software architecture has always to be evaluated, even when mostly reused

<b>Software Engineering Challenge</b>	<b>Objectives</b>
Design must take place before coding	Know what you are building before you begin to improve cost and scheduling accuracy Reduce product complexity with design detail and precision Cost, schedule, and risk control
Delivering the software technical data package	Complete design diagrams, drawings, and specifications for software implementation (construction)
Allocate requirements among elements of the design configuration	Requirements for decomposition and allocation among software components and units
Integrated product and process development (IPPD)	Requirements traceability
Preparing a software integration strategy	Concurrent design and development of product sustainment capabilities
Controlling software complexity	Life-cycle costs control
Enabling change assimilation	Planned software component integration developed during architectural design activities
Trade-off analysis	Efficient software implementation planning
Preplanned product improvement	Reduce software maintenance/support costs
	Efficient, user-friendly interactions
	Stakeholder/user satisfaction
	Product competitiveness
	Cost and schedule control
	Design optimization
	Product evolution/incremental release stability
	Increased probability of project success
	Delayed functionality to later releases to permit on-time product delivery

# Summary

- Software is a business, but it is not like other businesses
- Software products are both the programs and their documentation, included process documentation
- Software reuse encompasses a variety of techniques and methods that should be chosen carefully
- Software architecture is the discipline which studies software reuse methods

# A quotation

*Software is the invisible thread and hardware is the loom on which computing weaves its fabric, a fabric that we have now draped across all of life*

*Grady Booch*

# Self test questions

- What categories of “software” you know”?
- What are they differences?
- What are the main problems in the production of software?
- Where can I find a specific paper on a specific software architecture topic?



# Useful references

- Cusumano, The business of software, 2004
- <http://www.gartner.com/newsroom/id/3412017>
- <http://www.ambysoft.com/surveys/success2013.html>

# Questions?

