



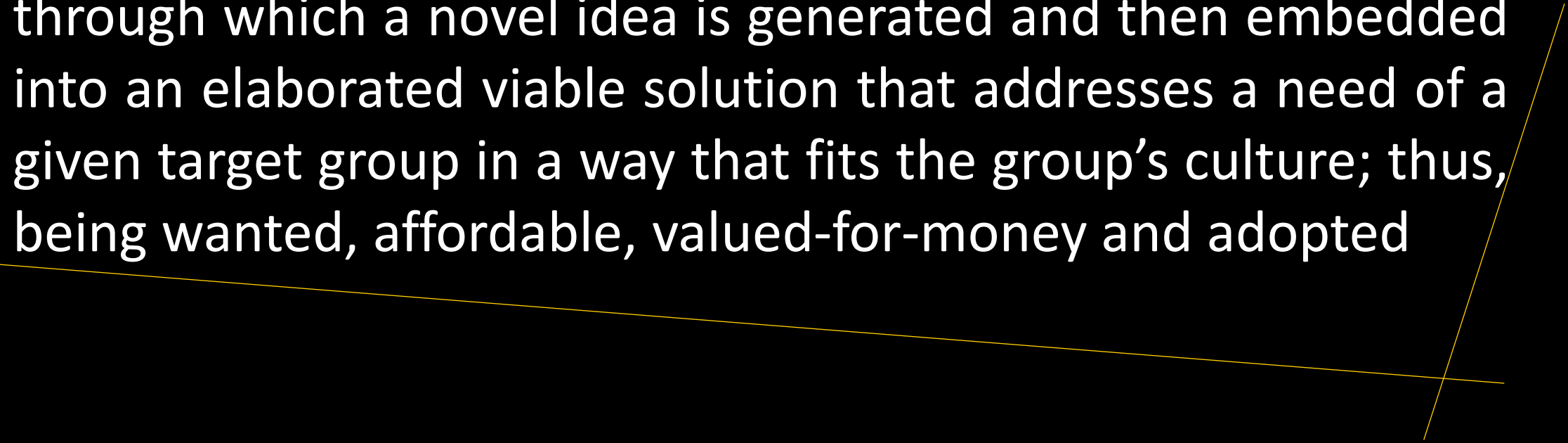
# The “Crazy” Future in Software Innovation

Prof. Stelian Brad

[stelian.brad@staff.utcluj.ro](mailto:stelian.brad@staff.utcluj.ro)

**Somebody said** “the world was happy when C was only a letter in the alphabet, Ruby was only a stone, Java was only an island, Python was only a snake, ...”

**Innovation** ... a nonlinear, multiple-looped and agile process through which a novel idea is generated and then embedded into an elaborated viable solution that addresses a need of a given target group in a way that fits the group's culture; thus, being wanted, affordable, valued-for-money and adopted

The text is set against a black background. A thin yellow line runs horizontally across the lower portion of the slide, starting from the left edge and ending near the right edge. Another thin yellow line starts from the right edge and extends upwards and to the left, crossing the horizontal line.

As the number of entities increases, the number of interactions between them would exponentially increase; and it would get to a point where it would be impossible to know and understand all of them

Higher levels of complexity in software increase the risk of unintentionally interfering with interactions and so increases the chance of introducing defects when making changes

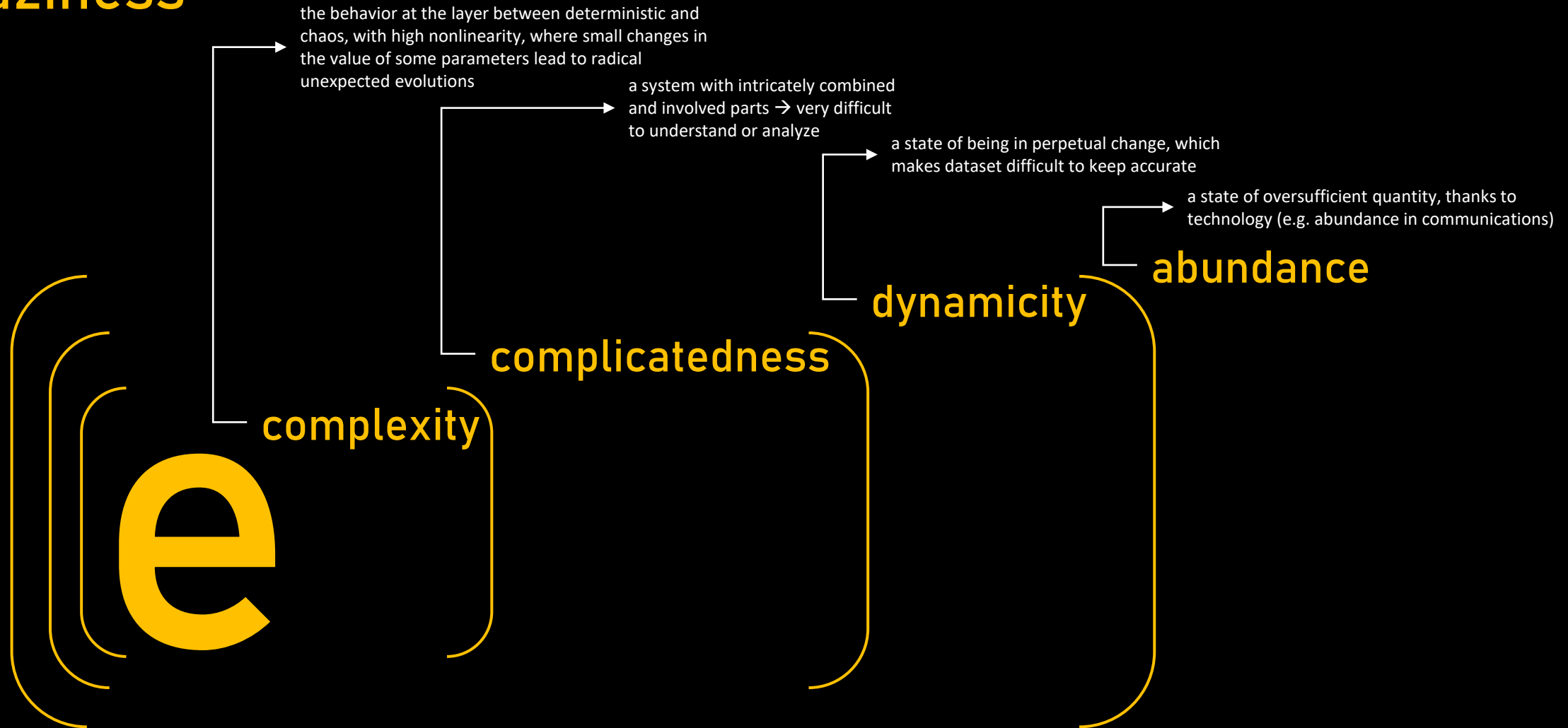
In more extreme cases, complexity can make modifying the software virtually impossible

**More and more platforms;  
platforms “nested”  
into platforms; high  
cost of multiple  
platforms**

What about system maintenance?

Which is the best practice in this context?

# “craziness”



- e**ntropy → a state of disorder, uncertainty, randomness in the system | level of possible combinations of the parts in the system
- e**quipotentiality → apparent capacity of any intact part of the system to carry out functions which are lost by the destroyed parts
- e**quilibrium → state of balance relative to the forces acting in the system



platform battlefield

architecting “systems-of-systems” & team coding by “lego”ing

very tight space for strategic and operational errors

very short time to keep a competitive advantage with a new innovation

difficult to generate clear differentiation

winner takes all

low predictability and high uncertainty  
in software innovation

# software paradox(es)

# technical paradox ::

technologies have tremendously multiplied and diversified in order to increase productivity and agility in software production *but* the job of professionals was not simplified [by contrary, it looks like a nightmare]

# economic paradox ::

strategic importance of software is tremendously growing *but* software monetization on a stand-alone basis is more and more difficult

**challenge**

**[problematic]  
reality**



**software & business  
sustainability**



# generic actions ... systematic problem resolution

See the growing popularity of Python or ROS (Robot Operating System) – huge libraries

See asymmetric multiprocessing, customized processors for AI

See reconfigurable computing for FPGAs

Standardized components in software

technical

What and how?

prior arrangements to go fast into action when required

What and how?

What and how?

asymmetrical systems (not asymmetrical software)

What and how?

What and how?

reconfigurable constructions

What and how?

What and how?

systems with automatic interchangeable parts

What and how?

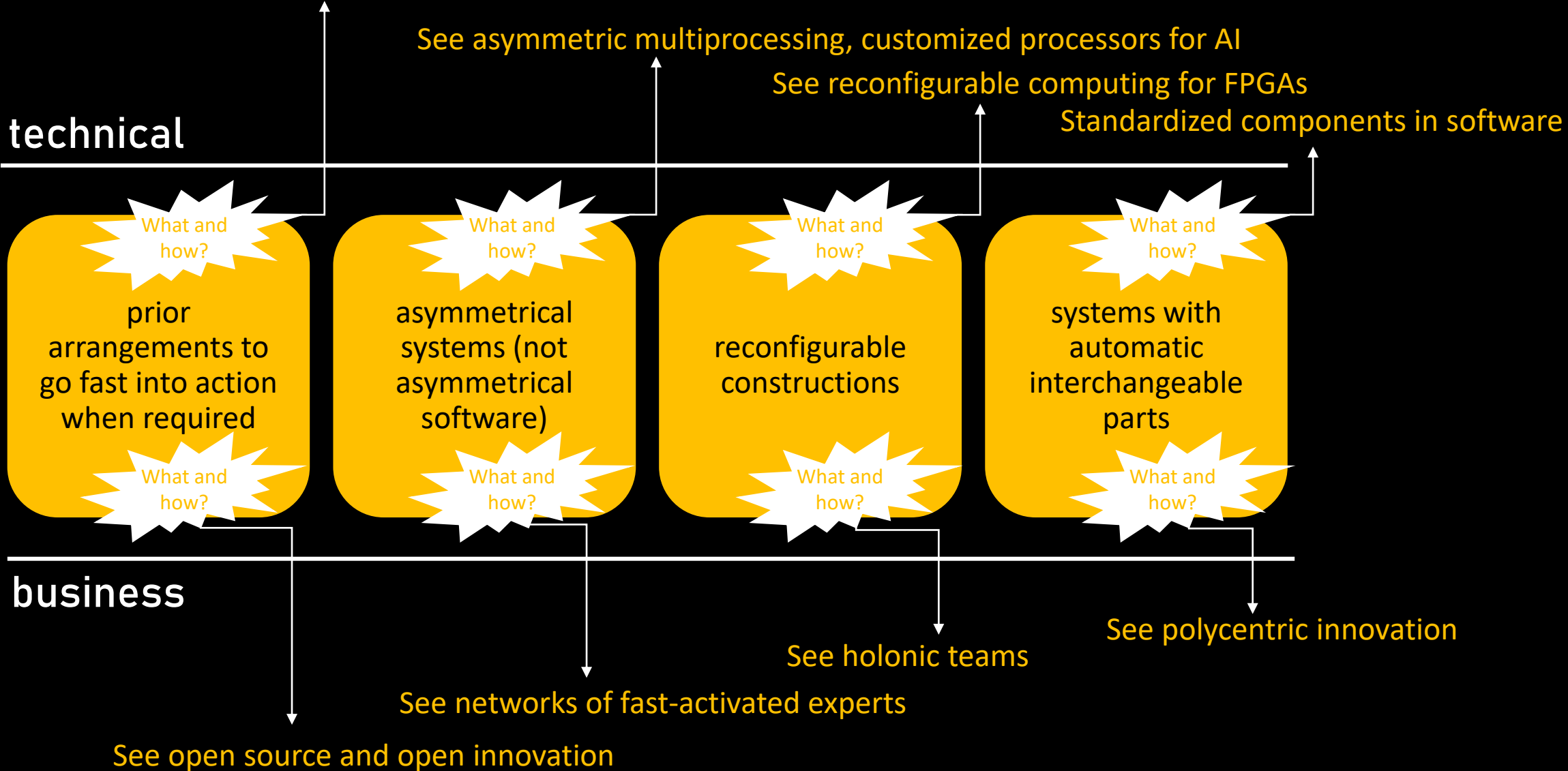
business

See open source and open innovation

See networks of fast-activated experts

See holonic teams

See polycentric innovation



... and the 5 fundamental rules

from IT product systems to IT  
product-service systems



develop sustainable IT product-  
service strategies in strong  
symbiosis with the key  
influencing factors

**Rule #1: Look for strong “stickers” to better  
understand patterns and lines of evolution**

# technical breakthrough :: a bunch of actions, not only one

beyond agile ... resilient software development      multiple levels of abstraction      dynamic software adaptation  
focus on RAD frameworks      unified apps on a single PaaS      hybrid backend technologies      automated support  
co-evolution, traceability and synchronization between all artefacts of the system      design for life-cycle ... UX & DX

# business breakthrough :: a bunch of actions, not only one

focus on lean innovation      hybrid businesses :: create a strong customer lock-in & sell services around free platforms  
business models beyond the comfort zone :: sell system integration, deployment, support and software derivatives  
sell also expertise and content, not only software highly customized pricing offer to maximize value for each customer

**Rule #2: innovate by “breaking” not by  
incrementing**

dynamics is becoming too high ... do not speed-up, focus on capacities for inventive approaching of crises  
compete on strategic positioning not on operational effectiveness ... do things to deliver unique value  
think in terms of deviation from ideality :: in an ideal architecture/design the system complexity is minimized

DYNAMICS is higher-and-higher [we cannot control it in a world where consumers dictate the speed]

more pivoting  
concurrent development  
lean prototyping

discard and recover teams  
back-up functions  
contingency plans

strategic alliances  
collaborative networks  
holonic organization

How to reduce COMPLEXITY?

How to increase RELIABILITY?

How to increase CAPACITY?

**Rule #3: Adopt "reverse" thinking paradigms**

3. self-regulation

4. conserve stability

5. conserve familiarity

2. increase complexity

behavioural laws

6. continuous growth

1. continuous change

8. feedback loop

7. declining quality

3. development of specialized systems 4. complete reconstruction of the system 5. transition to new principles

2. consolidation into a super-system

technical laws

6. from an open system to a closed system

1. independent sub-systems

8. intelligent and higher autonomous system 7. transition to higher-systems

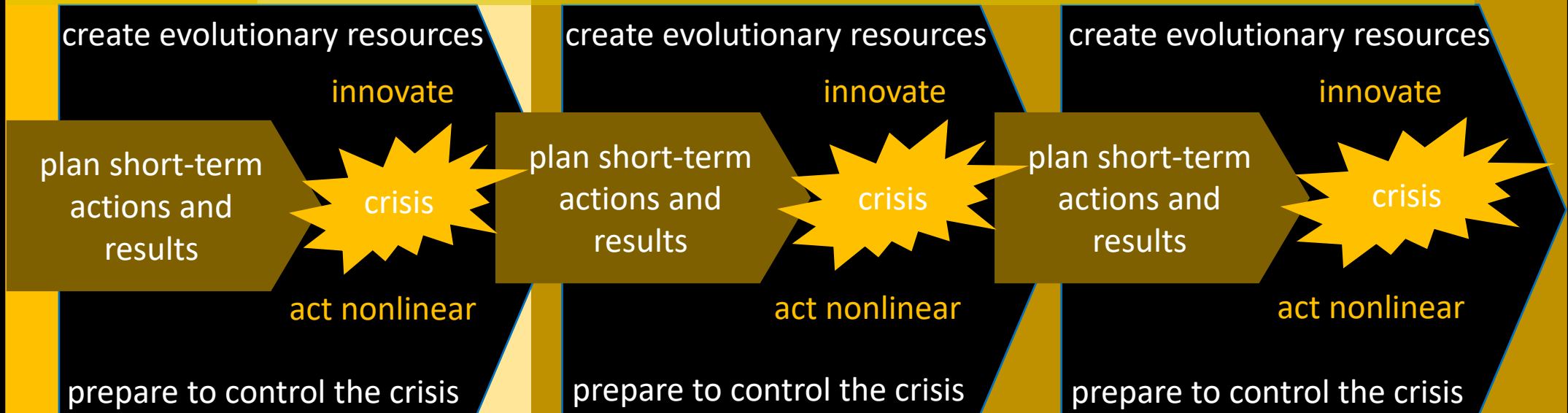
**Rule #4: Understand the laws that govern software evolution**

Short term

Medium term

Long term

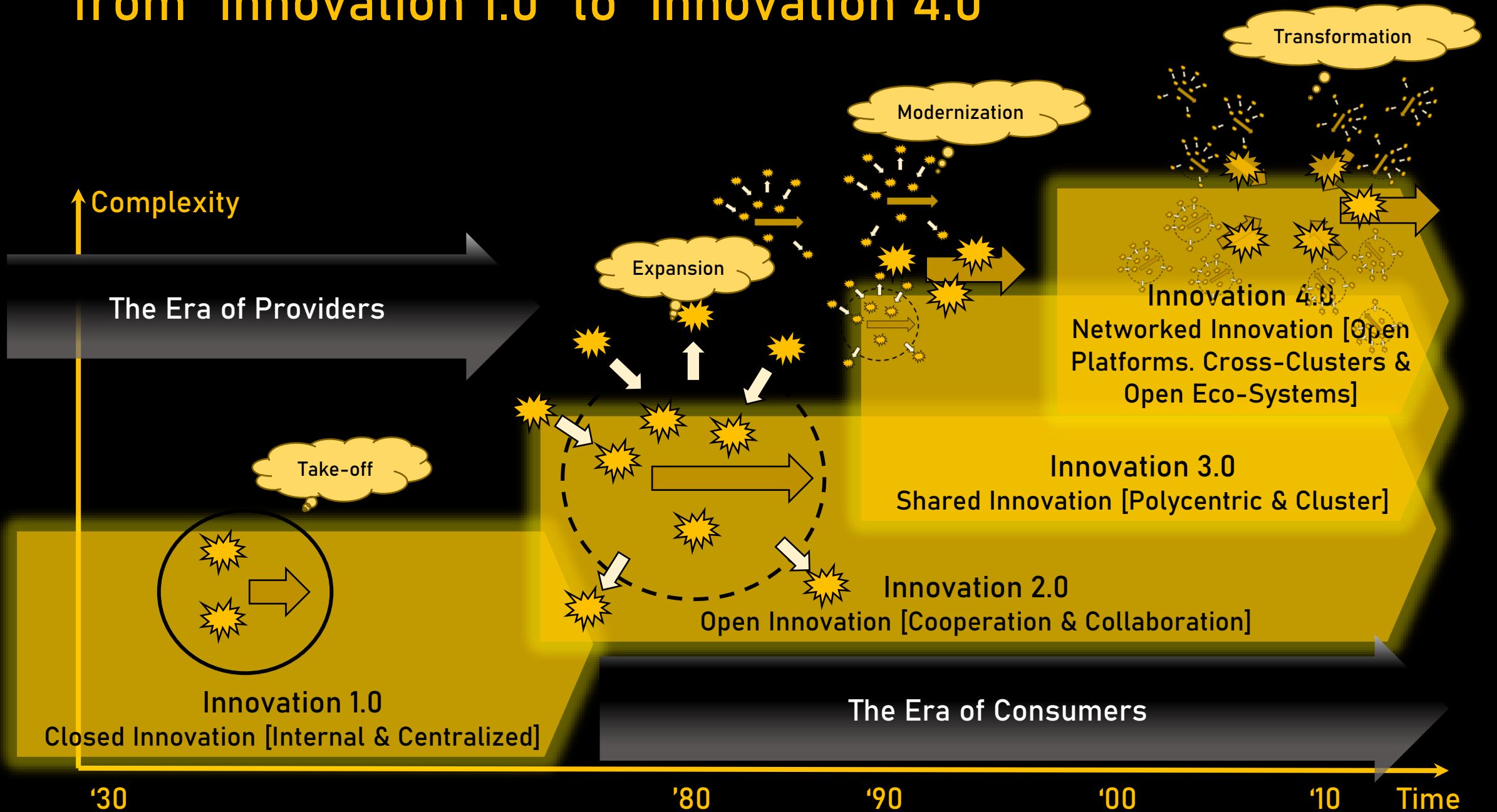
think in terms of multiple futures and invest in robustness



unpredictability requires higher productivity and agility

**Rule #5: Focus on robustness to fuel agility and resilience**

# from "innovation 1.0" to "innovation 4.0"

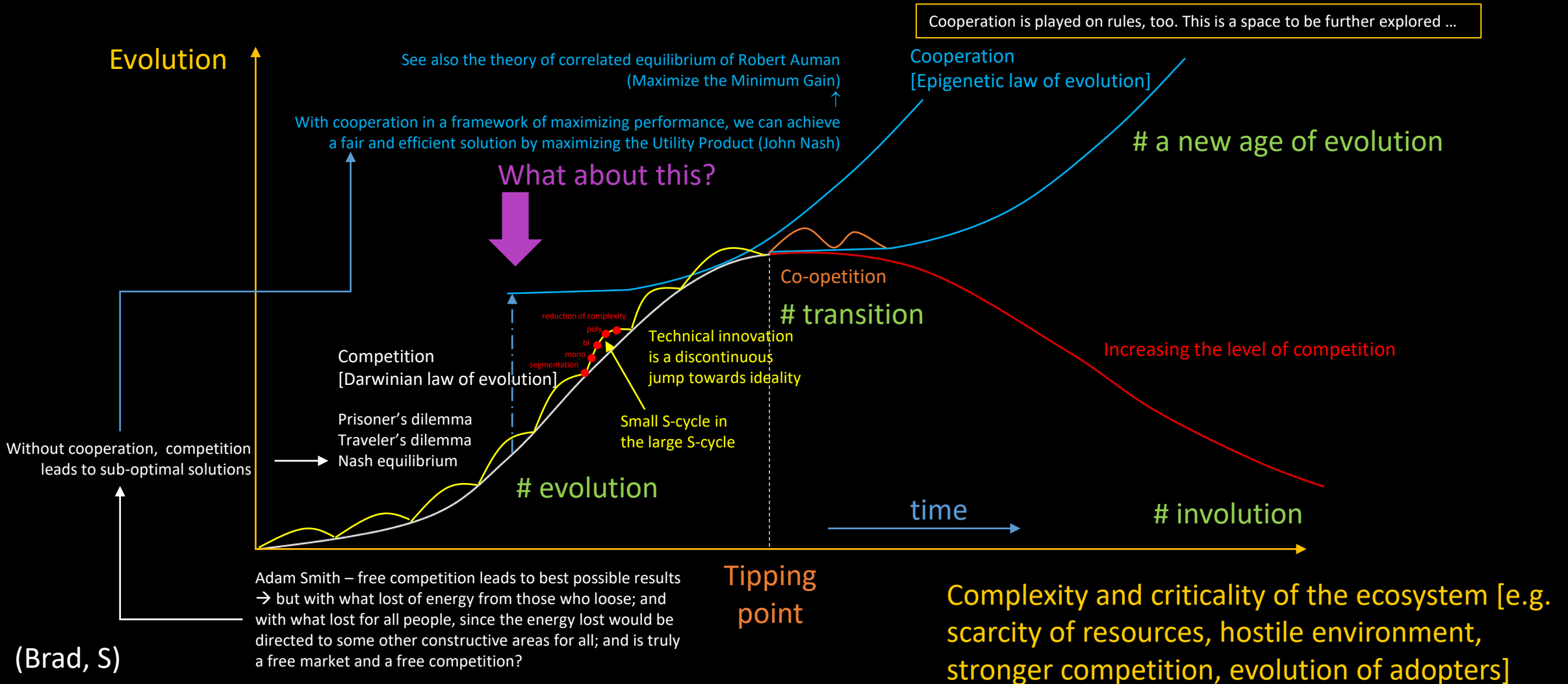




**future in software innovation ...**

**... culture of polycentric agile strategic alliances**

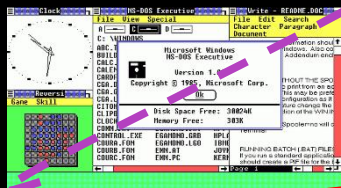
# evolution stands in cooperation, not in competition



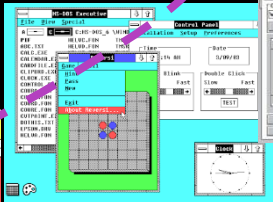
# example

Evolution of technical systems did not follow the same path as the evolution of innovations (value perceived by the market) and the evolution of the business behind the technical system (profitability)

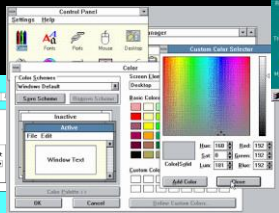
Windows 1.0 | 1985



Windows 2.0 / 2.1X | 1987



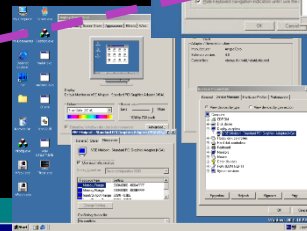
Windows 3.0 / 3.1 NT | 1990



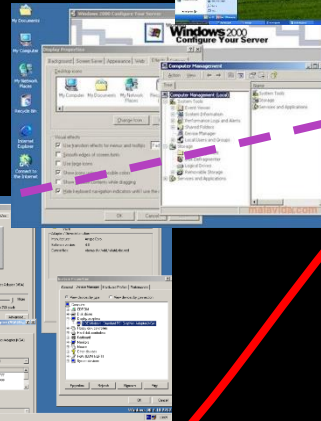
Windows 95 | 1995



Windows 98 / 98 SE | 1998



Windows 2000 | 2000



Windows XP | 2001



Windows VISTA | 2005



Windows 7 | 2009 → Windows 8 | 2012



Windows 10 | 2015 → 9 versions

$$\frac{\sum F \rightarrow}{\sum C \downarrow}$$

Microsoft announced that Windows 10 is the last major series of OS they will develop

$$\frac{\sum F \uparrow \uparrow}{\sum C \uparrow}$$

Number of innovations in the system

Competition, and market capability to adopt innovations (related to business sustainability) generate a top barrier in technical systems evolution

Impact of system performance in the market is not related to the level of inventiveness, but with respect to the capacity of the users (market) to use that performance to bring value added to their interest

$$\frac{\sum F \uparrow}{\sum C \rightarrow}$$

Impact of innovations in the market

$$\frac{\sum F \uparrow}{\sum C \uparrow}$$

conclusion is yours ...



To watch this video clip from a pdf file, click [here](#)