# From Legacy to State-of-the-art; Architectural Refactoring

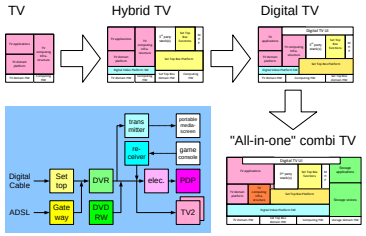
by *Gerrit Muller* University of South-Eastern Norway-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

## Abstract

The market of electronic appliances shows a fast increasing diversity. Manufacturers must be able to combine existing functions and new applications in a short time frame. A large amount of accumulated SW code (legacy) has to be reused in new ways.
The architecture(s) must be adapted to these new ways of working. Revolutionary adaptations have proven to be extremely risky. Opportunistic extension and integration decrease the quality of the code base, making it increasingly more difficult to continue. Architectural refactoring is a feedback based method to evolve an architecture.



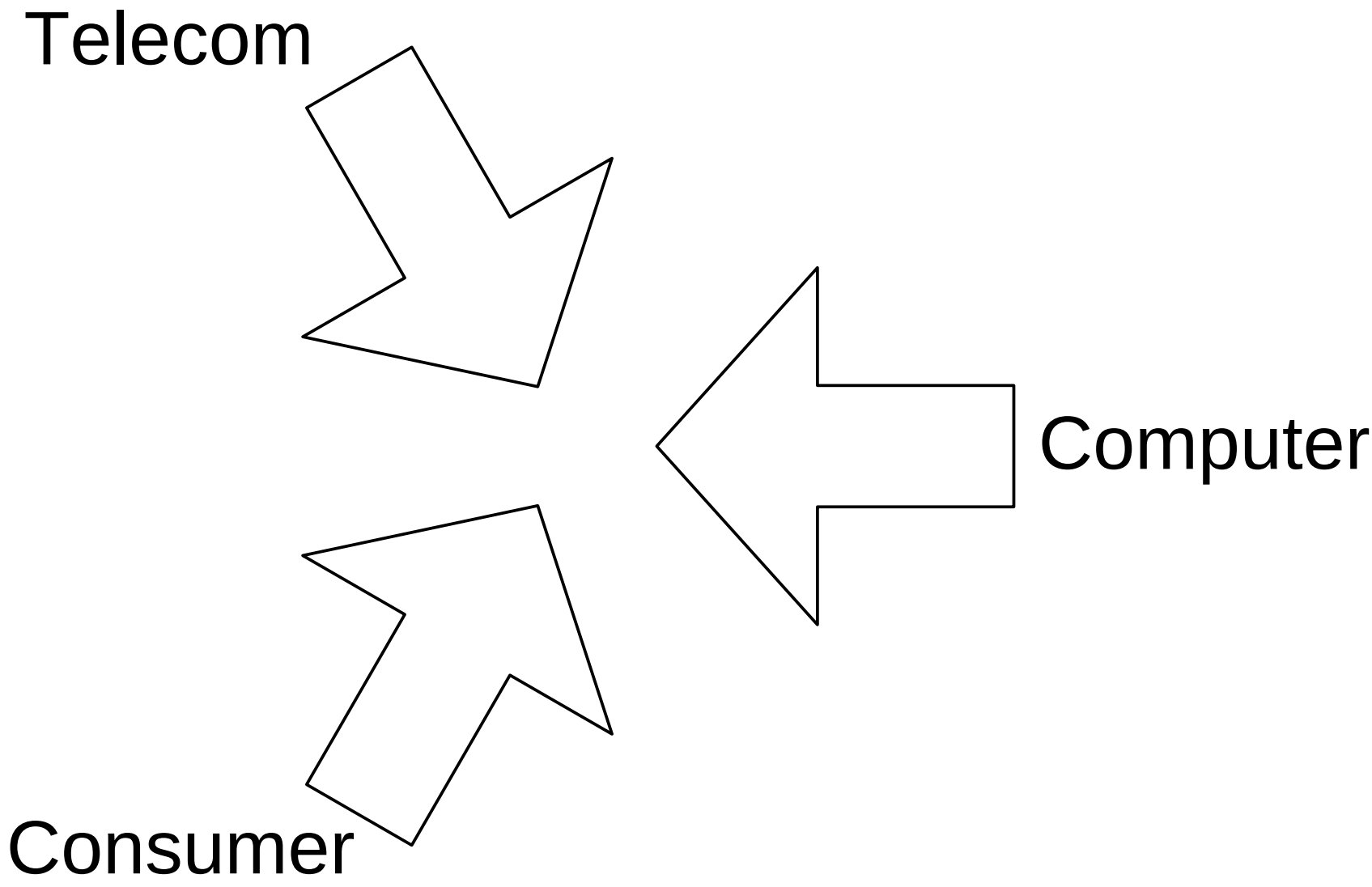
September 1, 2020
status: finished
version: 1.3



# Today's Audio Video Consumer Products


From: COPA tutorial, Rob van Ommering

# Trend: Convergence of separate worlds



# Integration and Diversity


GSM phone


firewall


dvd


audio microset


pda


watch


sailboat


surveillance camera


cable modem


set top box


headphone


pen


garment


car


camera


speech

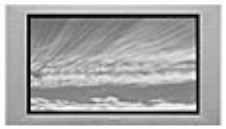

mp3


television

car navigation


computer

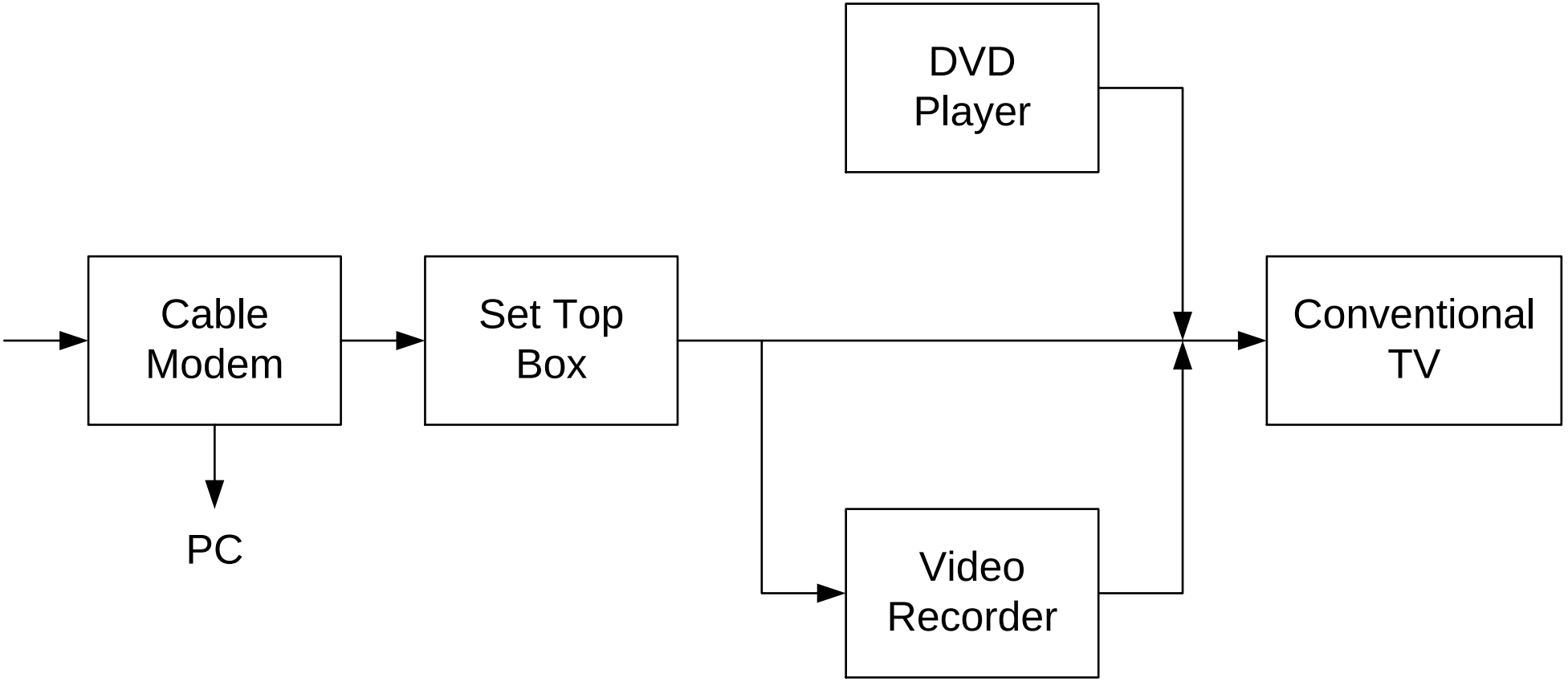

games


flat display

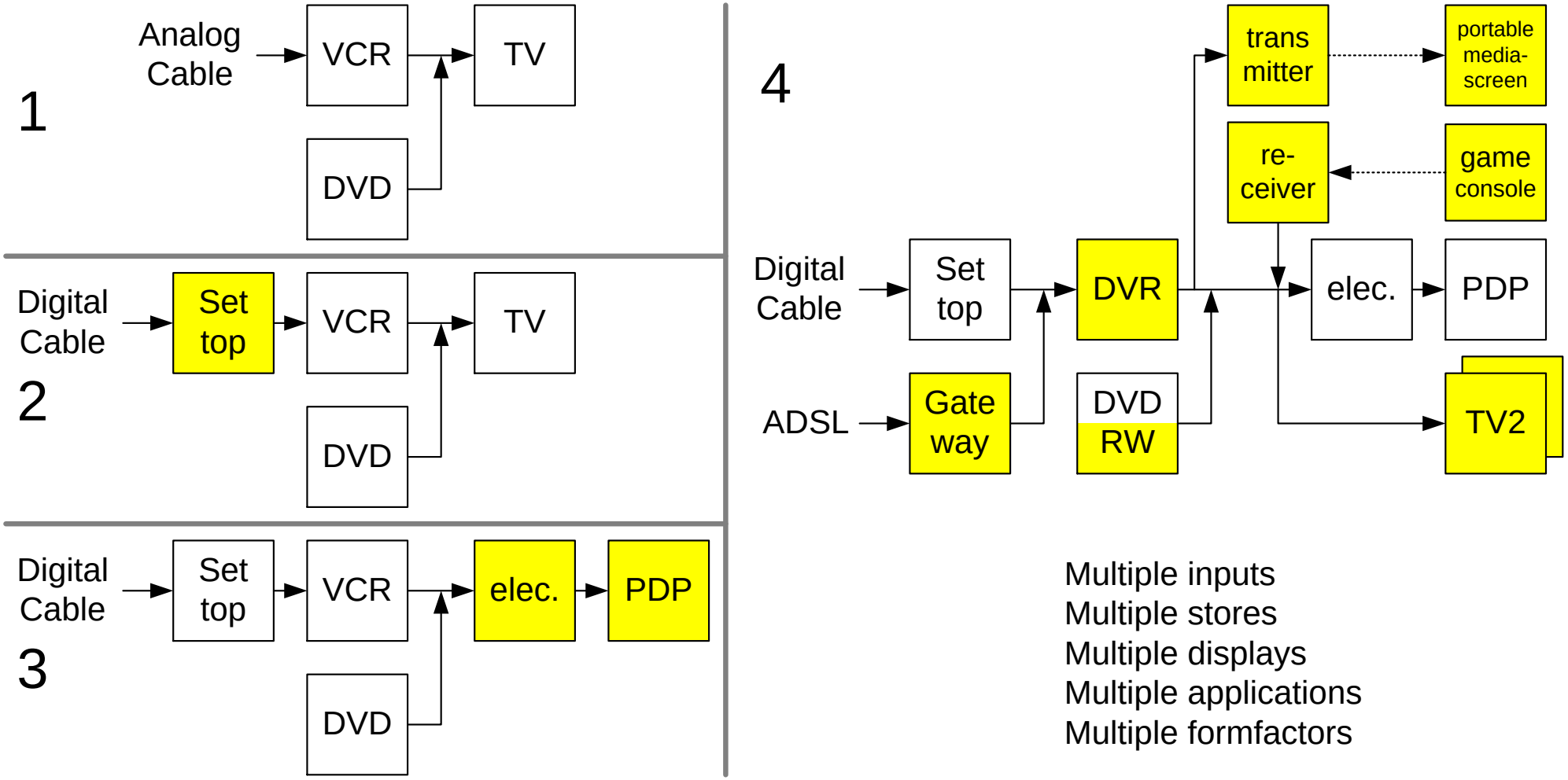

Communicator


Ambient Intelligence living room

# Today's Video Products

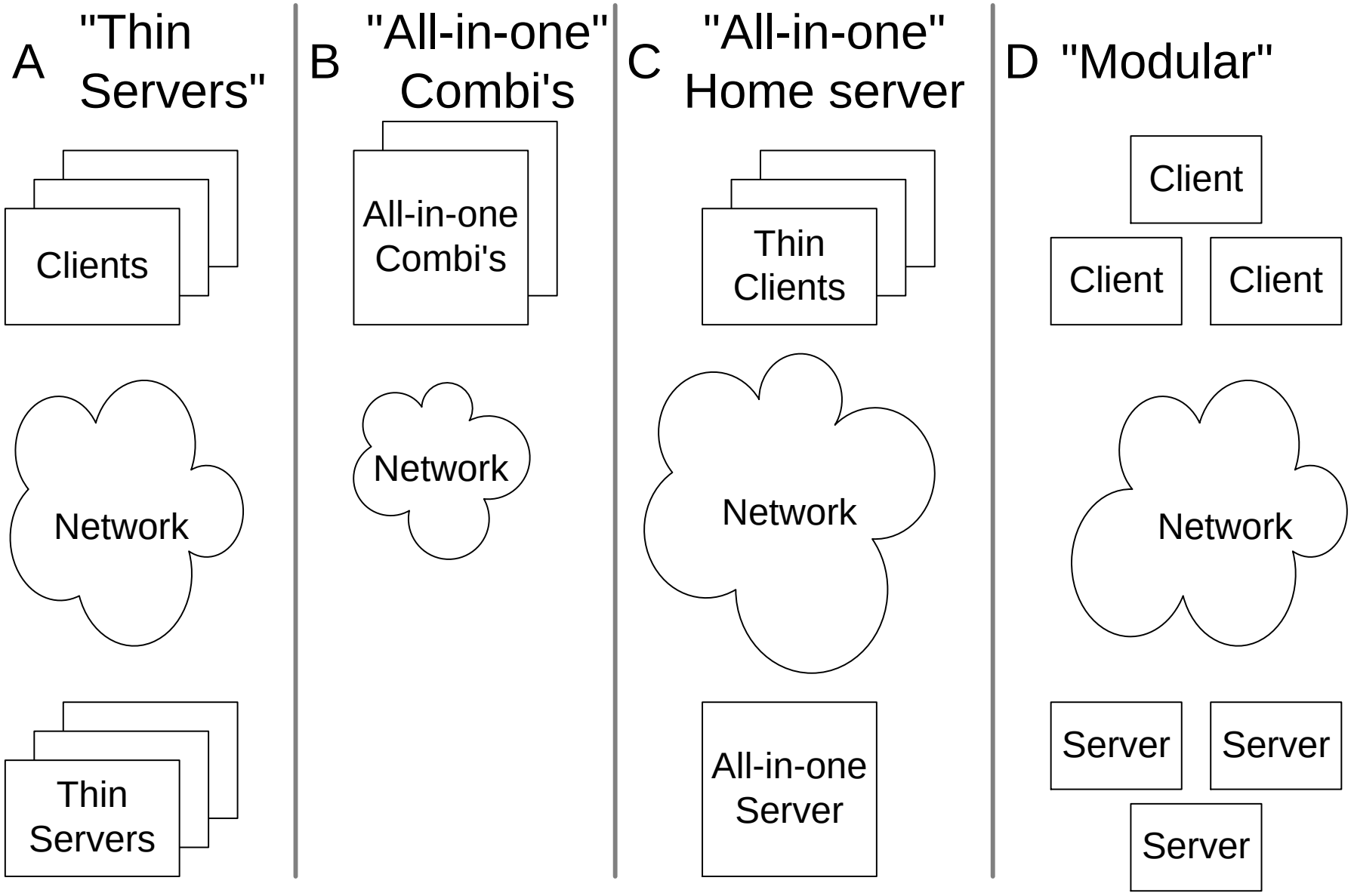


# Evolution of Video Products

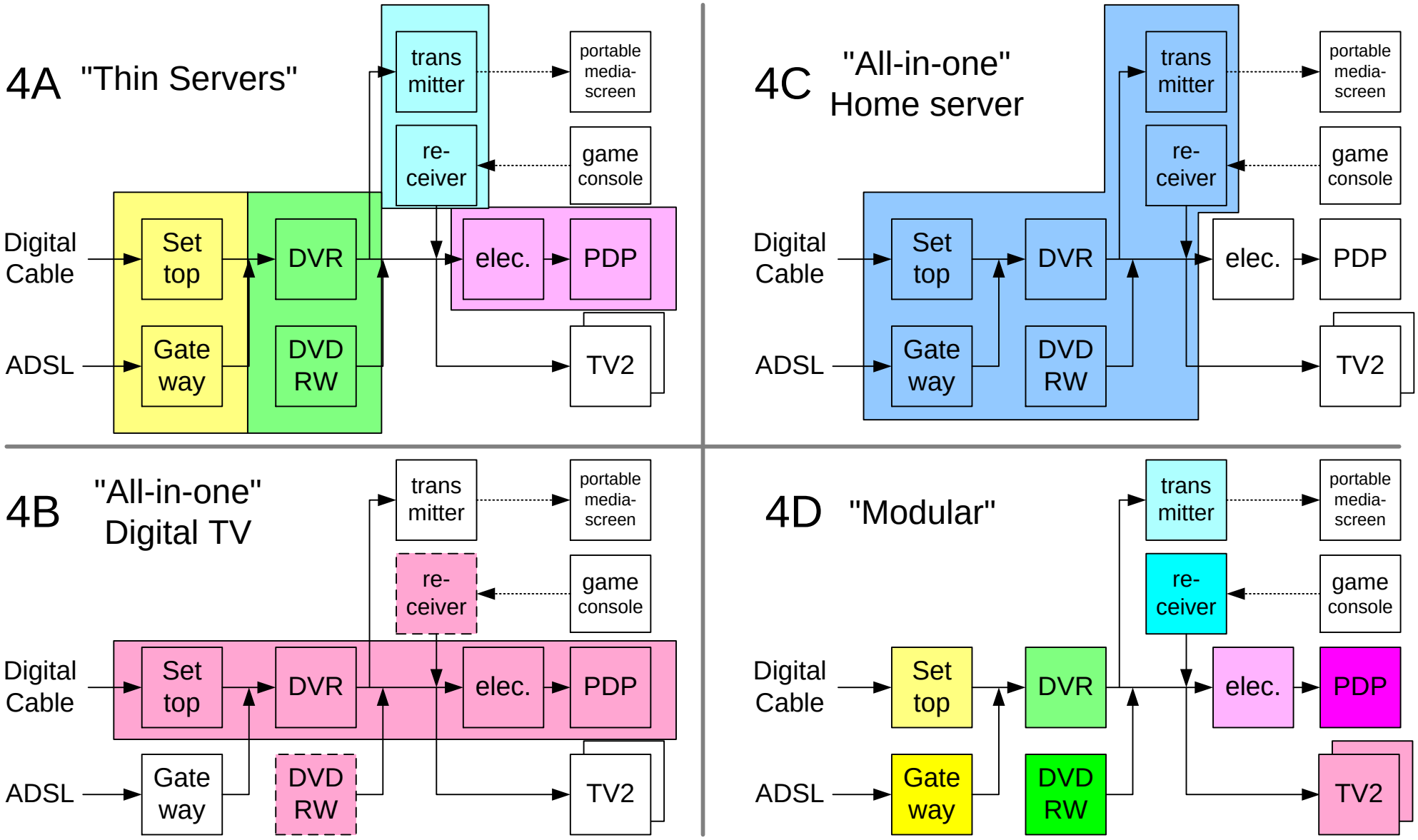



Multiple inputs
Multiple stores
Multiple displays
Multiple applications
Multiple formfactors

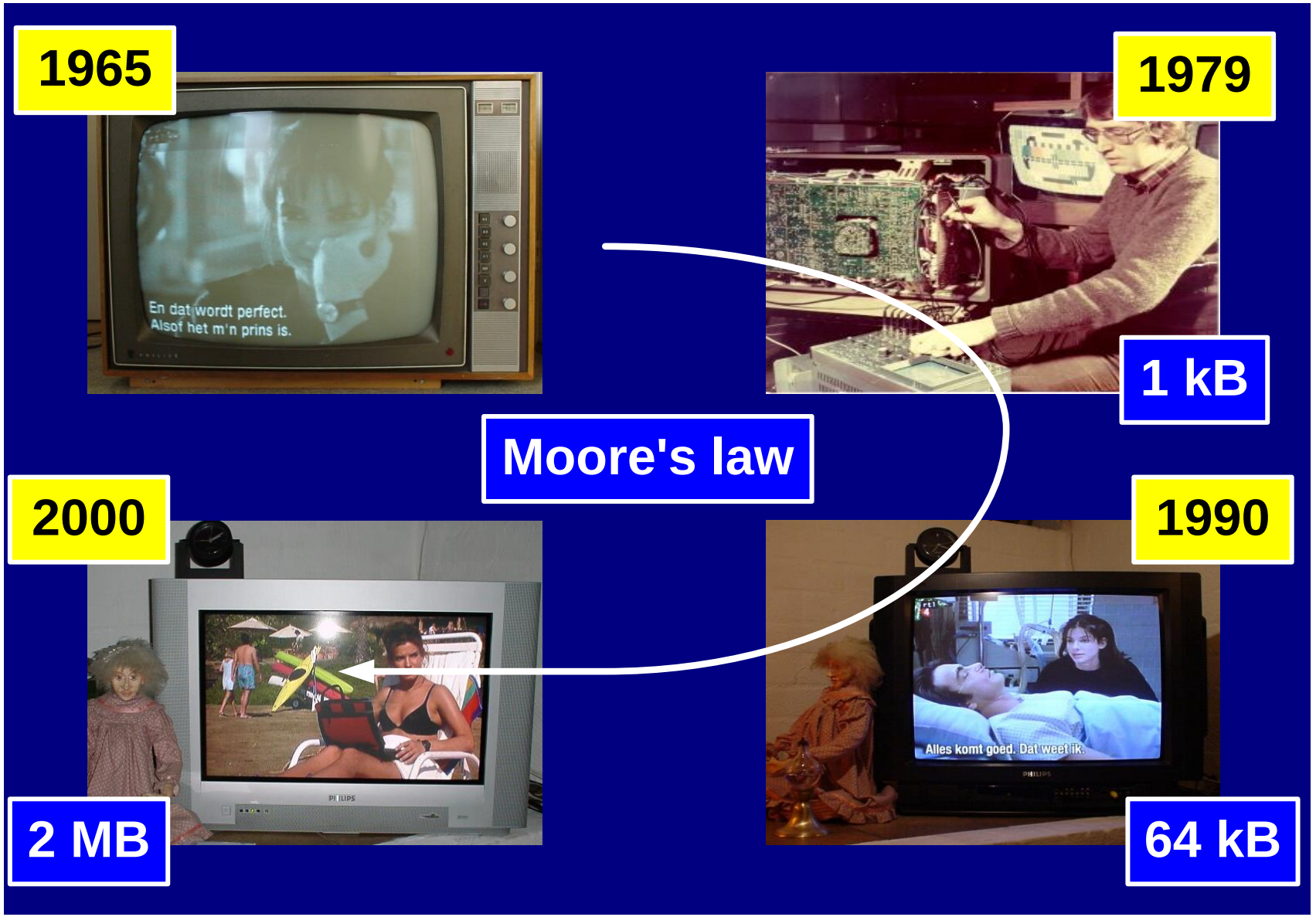
# Distribution Scenario's



# Product Packaging Options

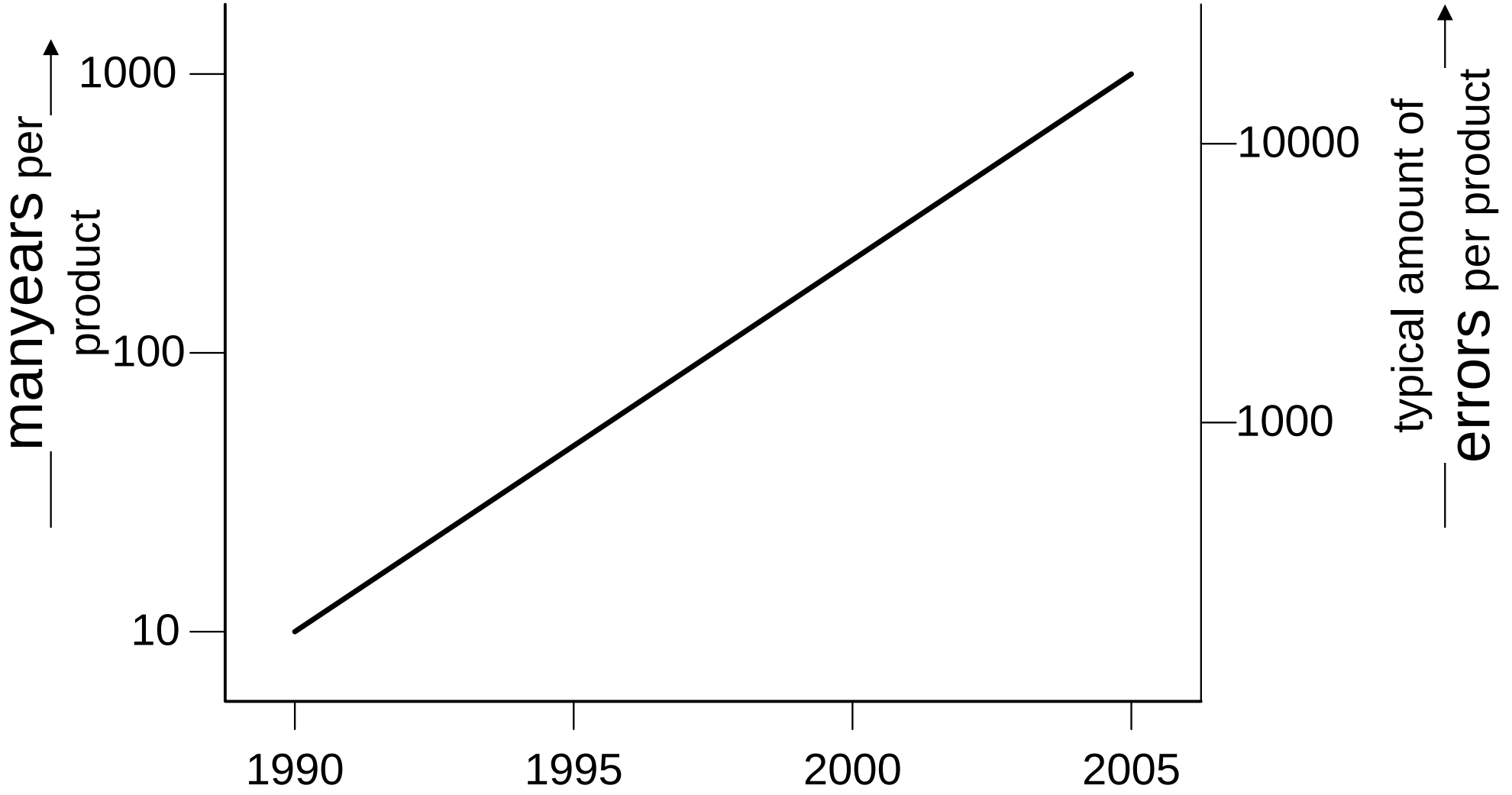


# Moore’s law

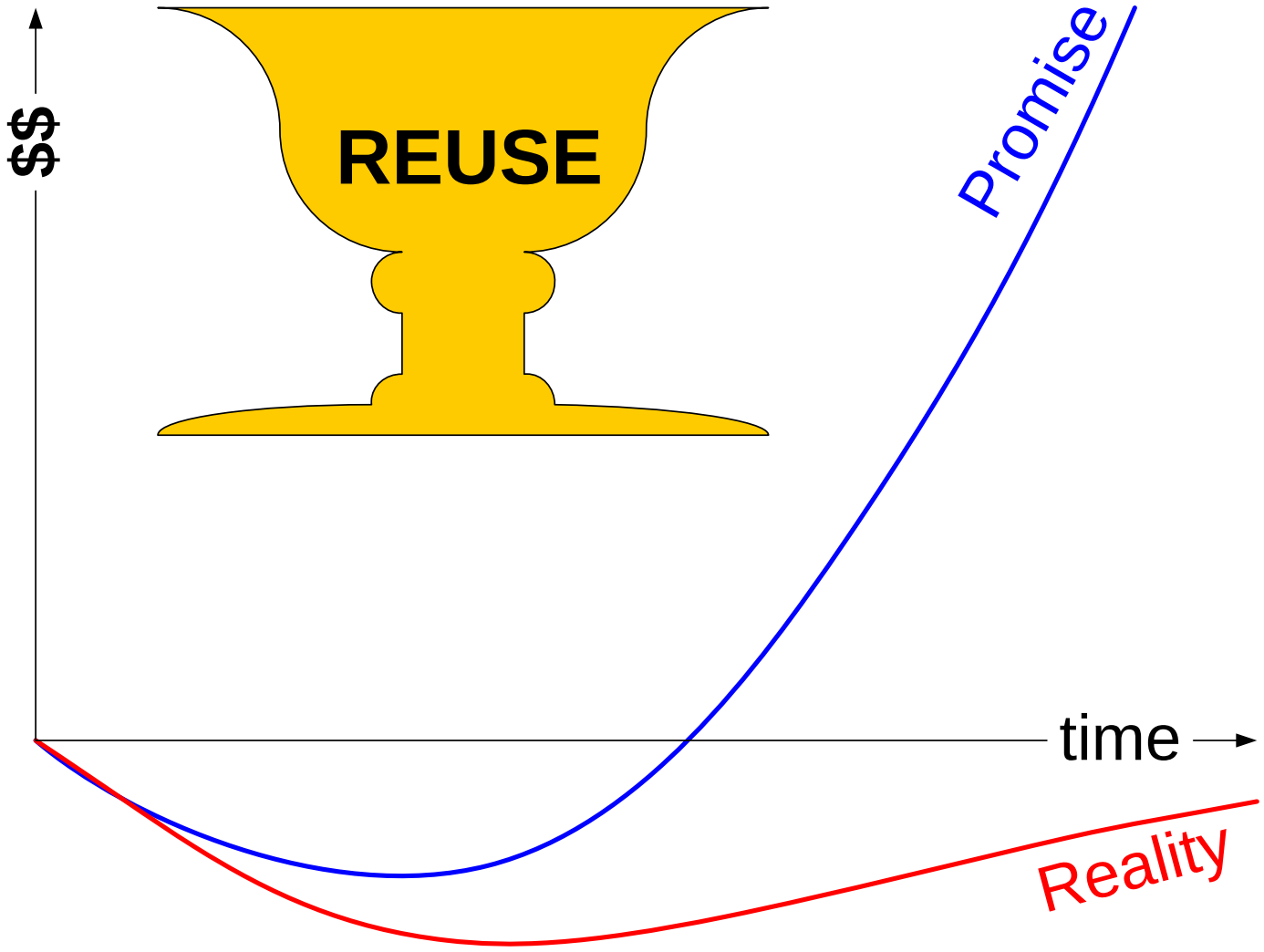



From: COPA tutorial, Rob van Ommering

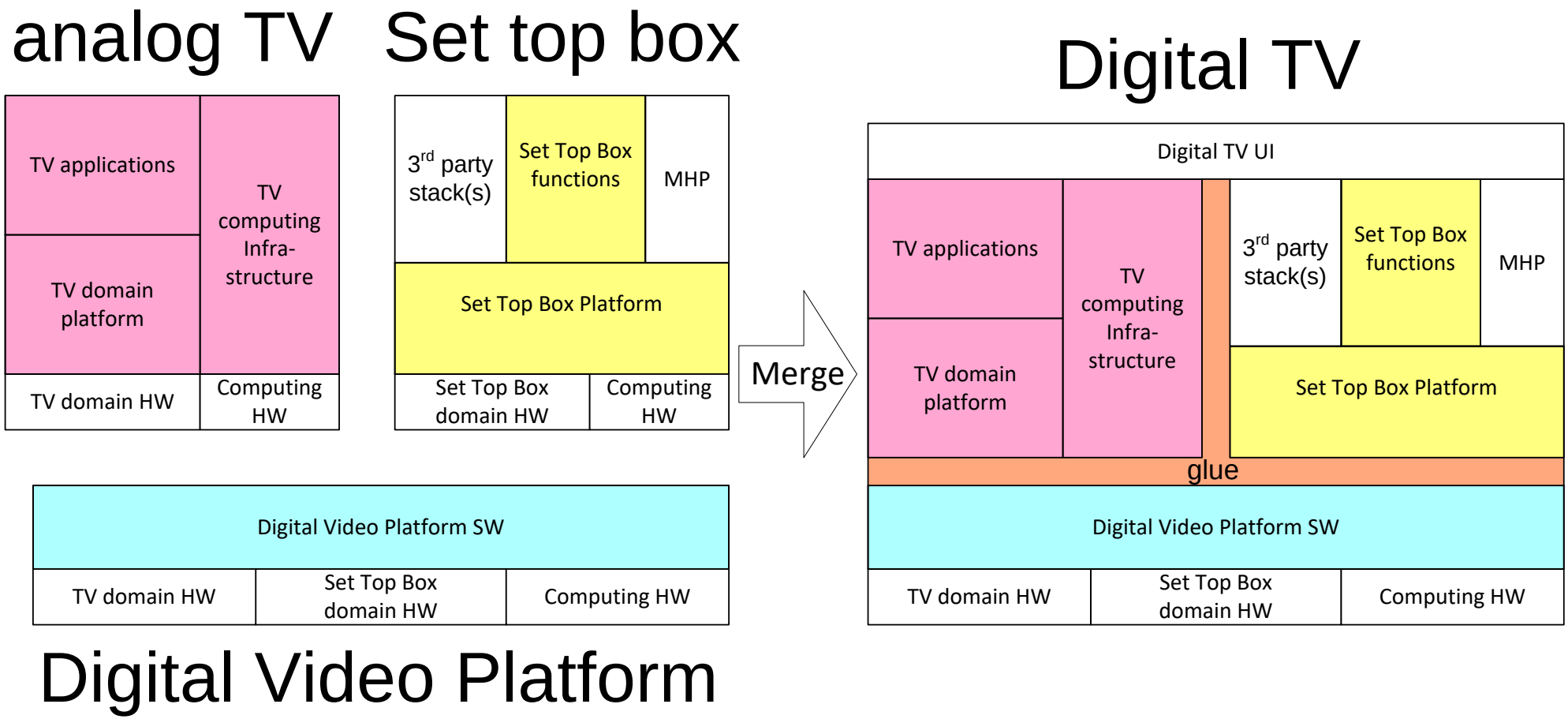
# Problem: increasing SW size, decreasing reliability?



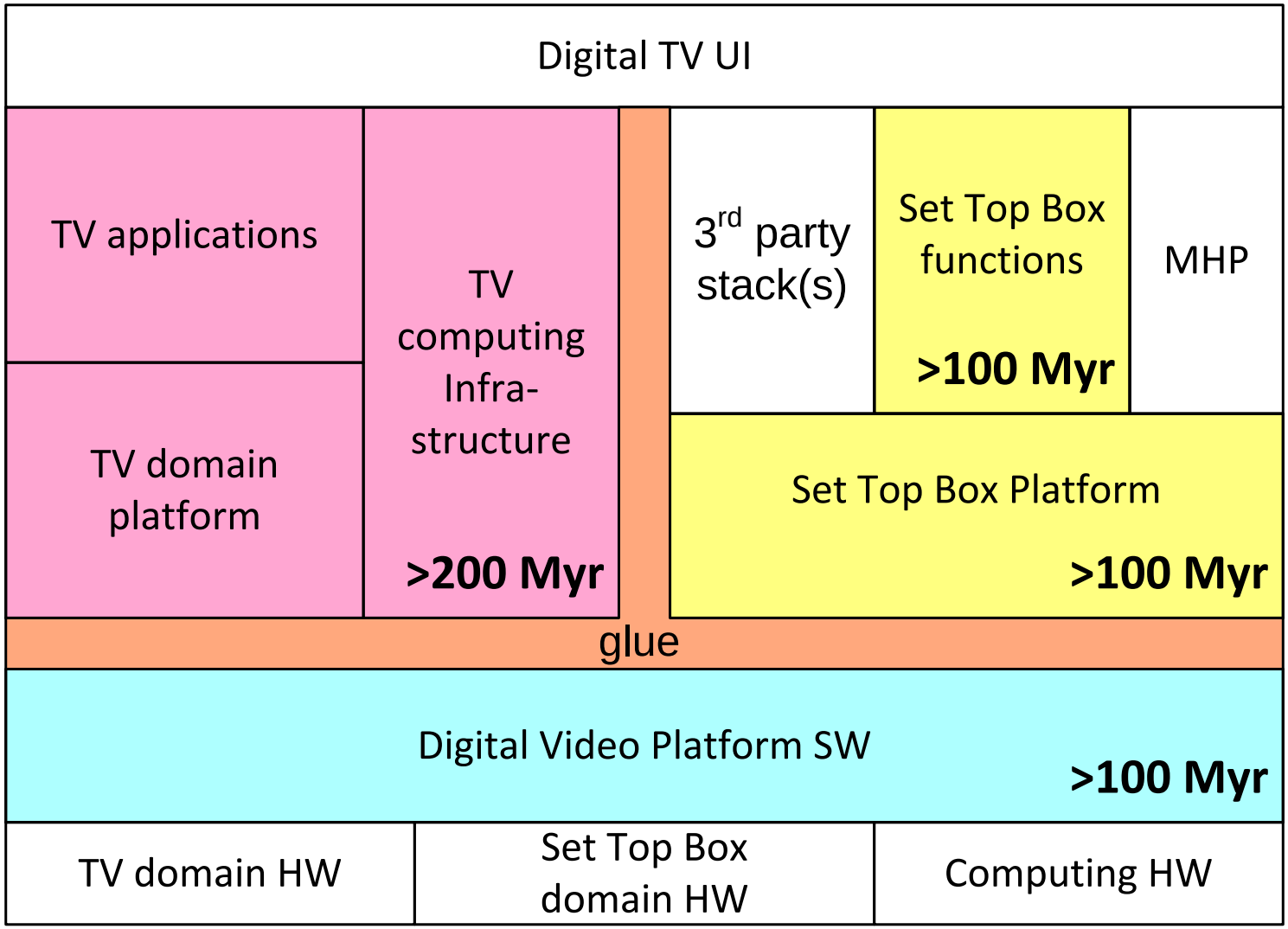
# The Holy Grail: Reuse



# Simplistic Architecting: Digital TV

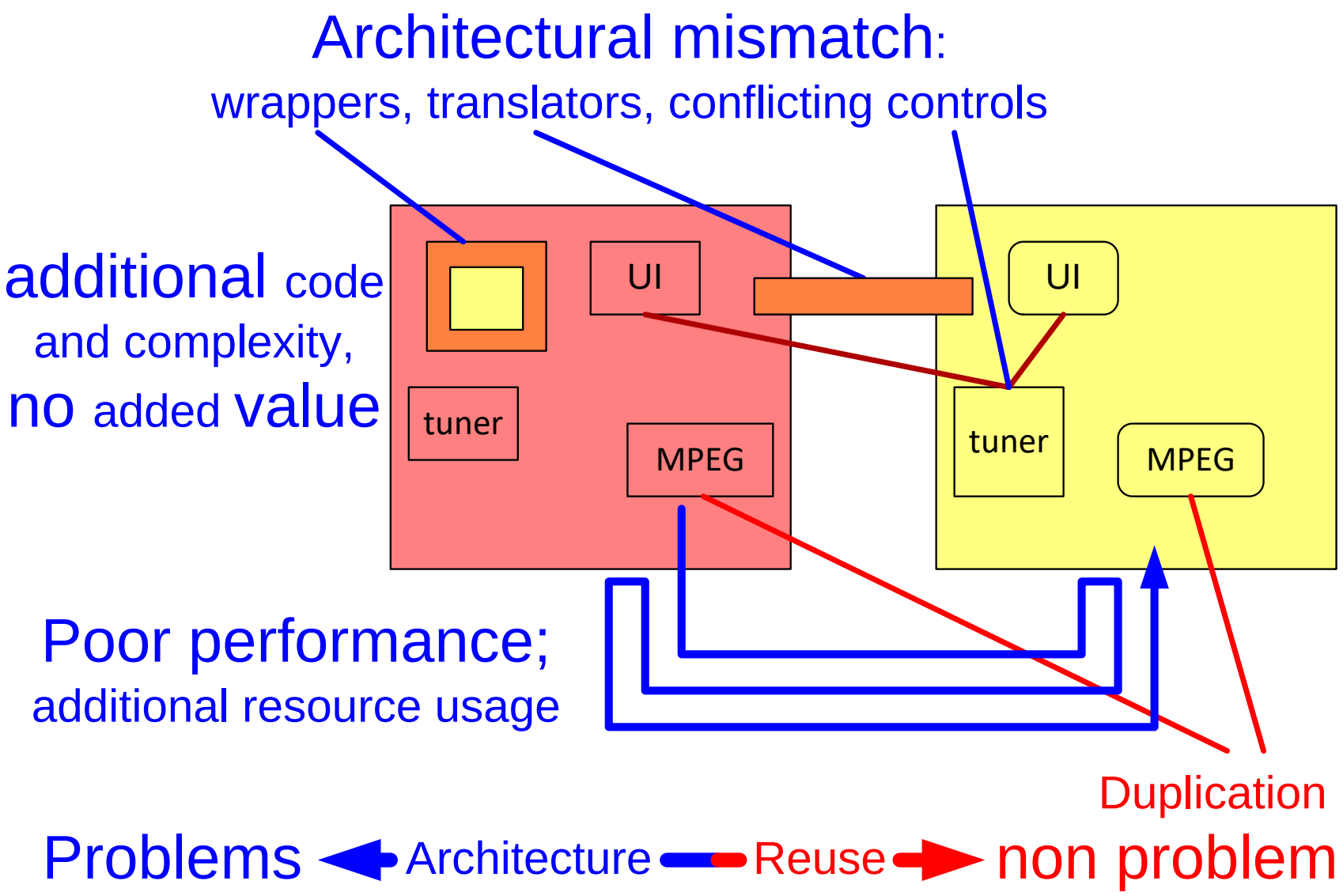


# Available Code Assets

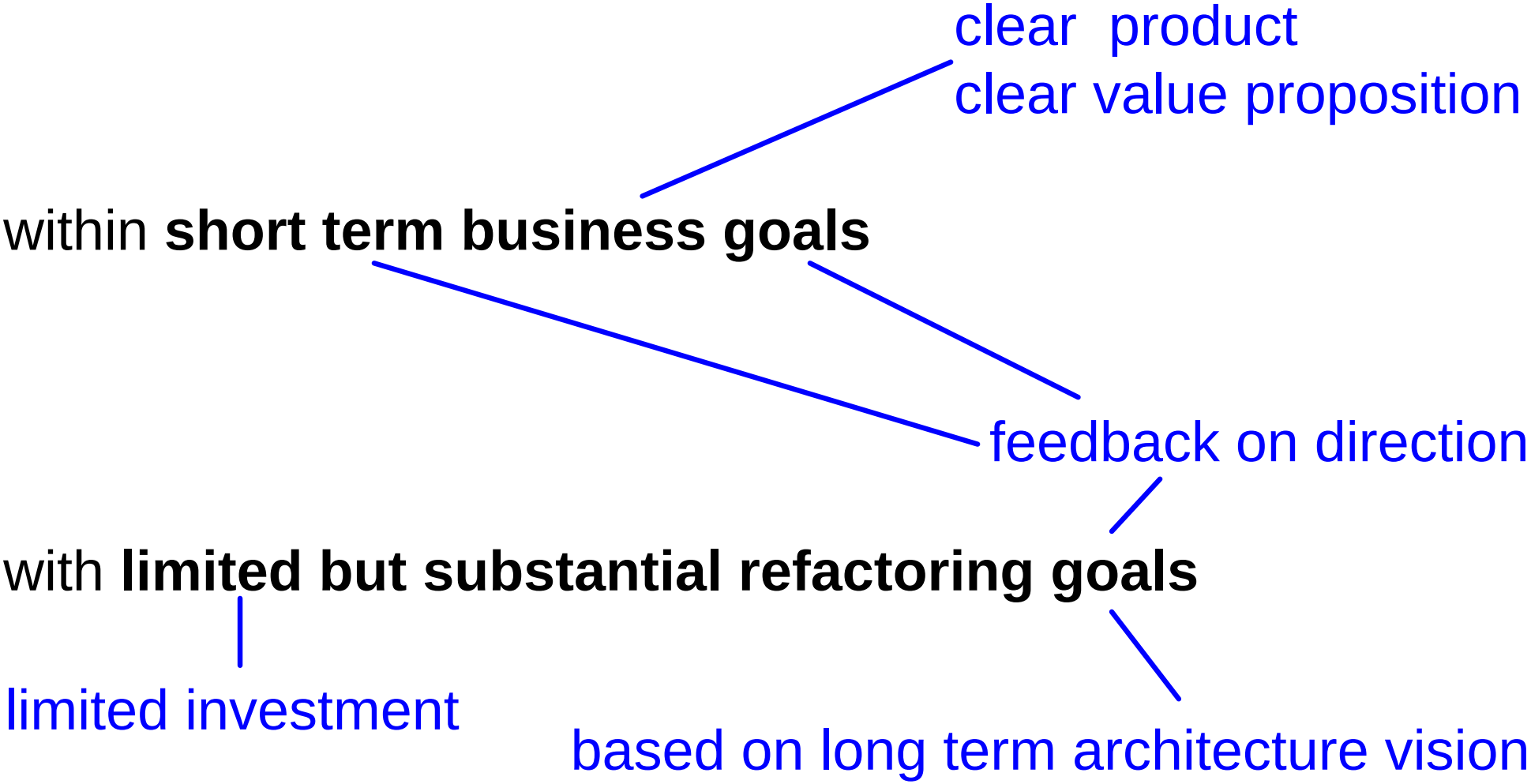



**"Legacy" code > 500 Myr**

# Merge problems



# Refactoring

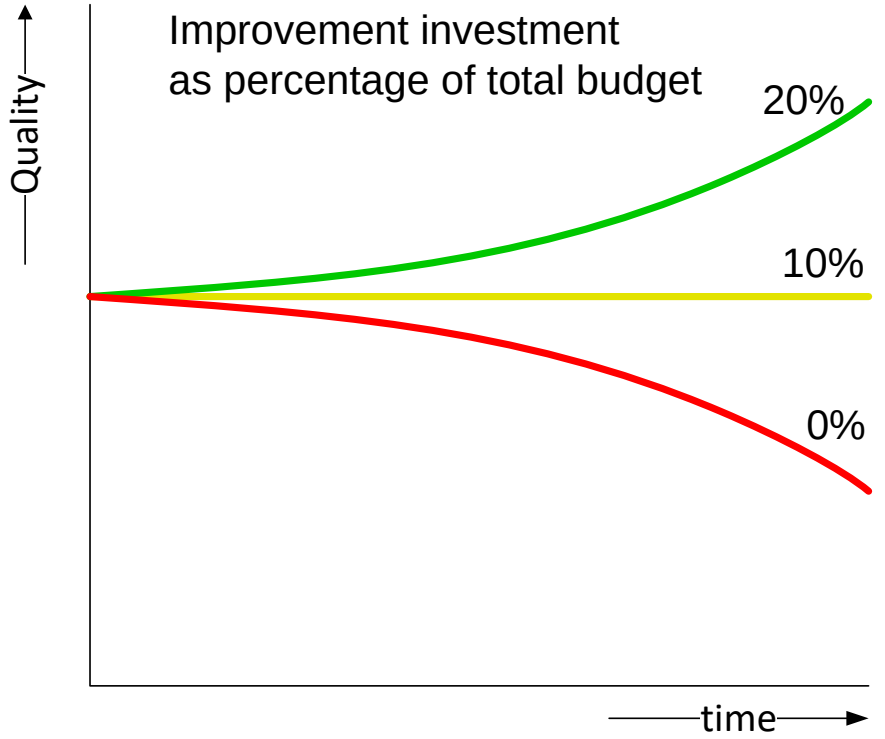


# Example of Refactoring Goals

+ Decrease Code Size

+ Decrease Resource Usage
  * power
  * memory
  * silicon area

+ Increase Performance
  * response time
  * throughput

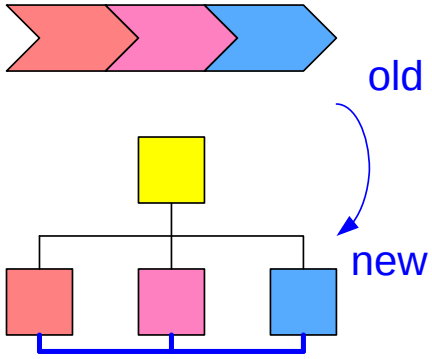



+ Increase quality
  * decrease fault density

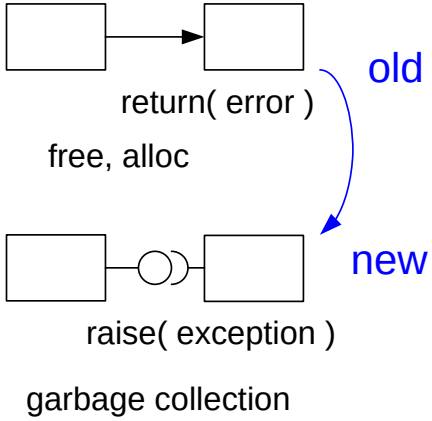
# Architectural vs Code refactoring

## Architectural Refactoring

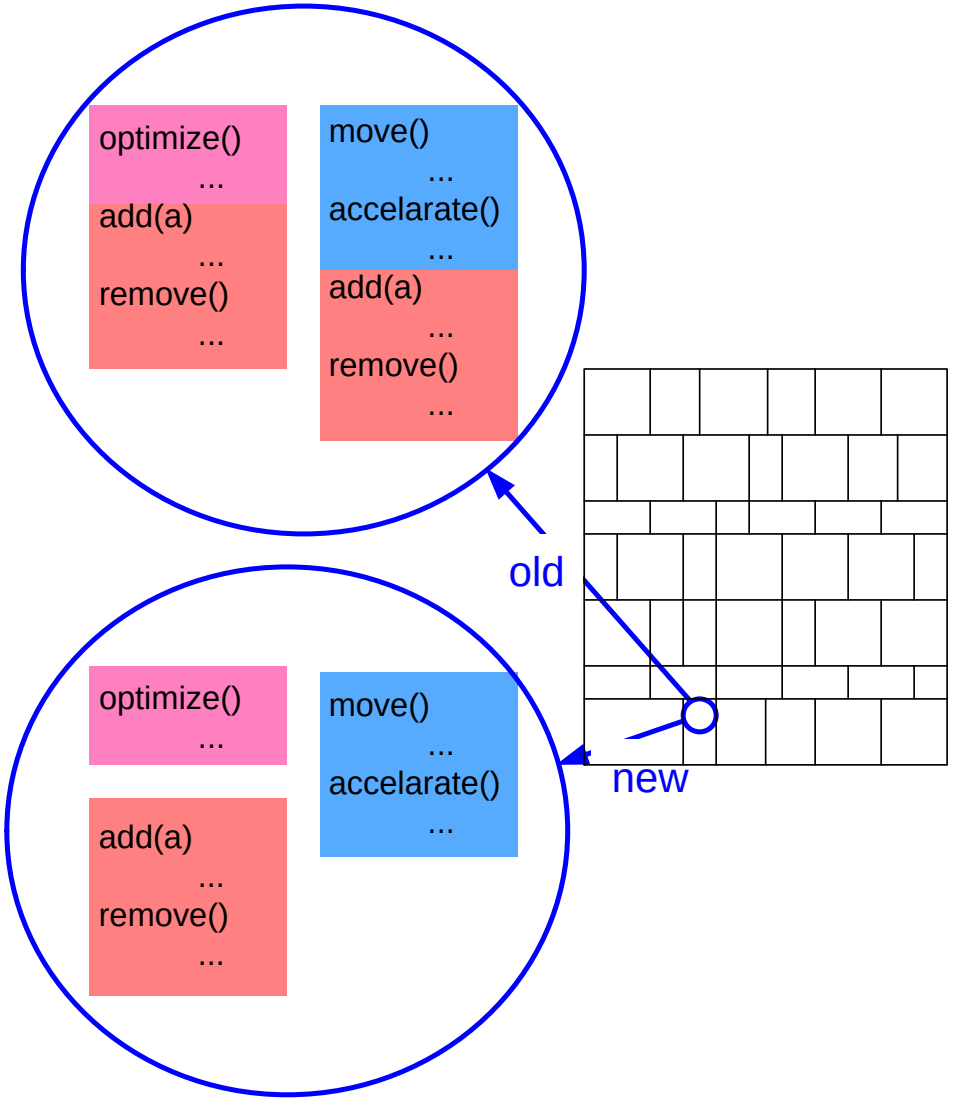
Function, Structure, Rationale




Mechanisms, Technologies



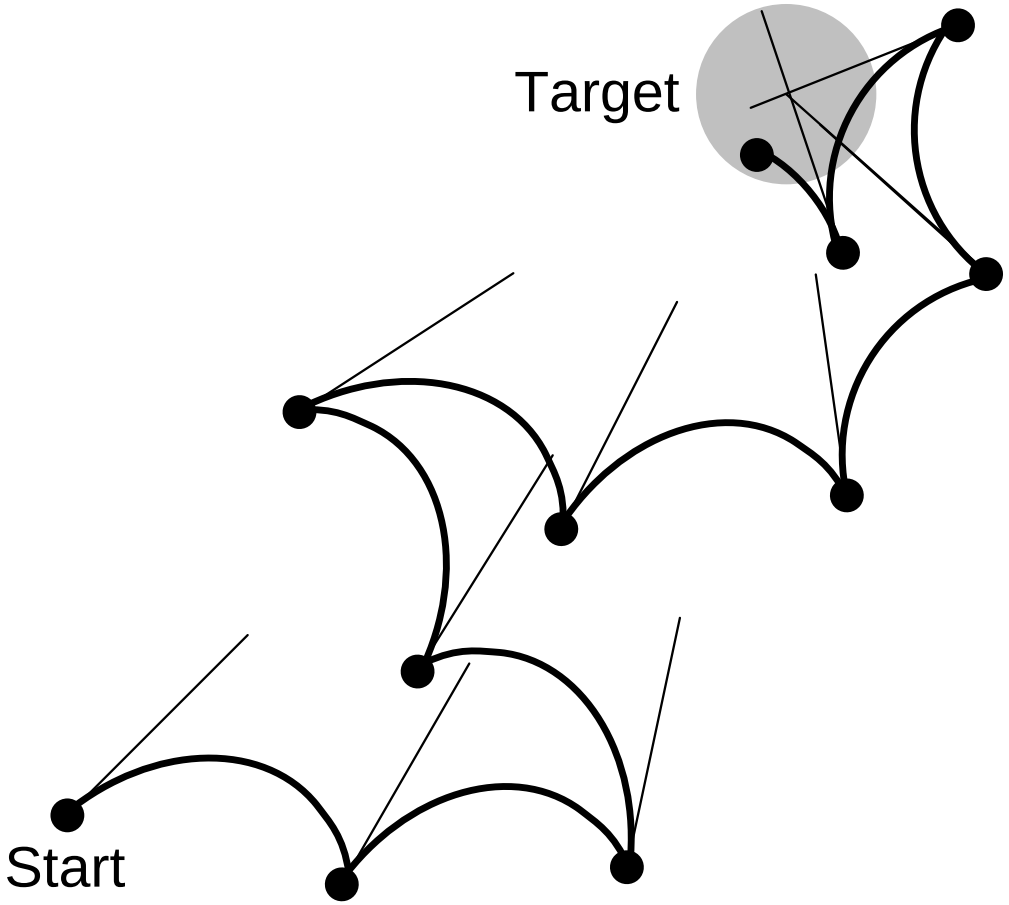

## Code Refactoring



# Frequent feedback

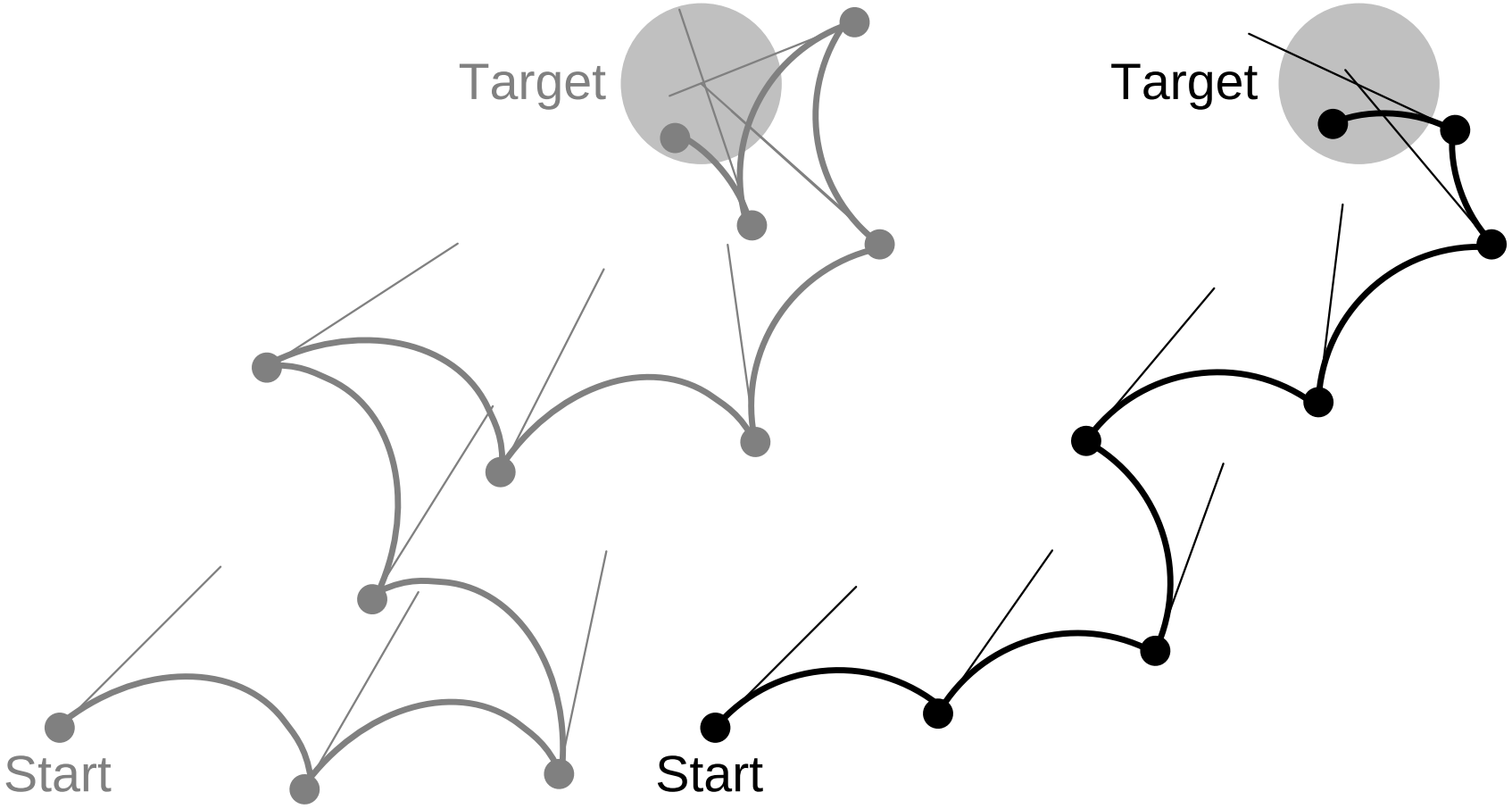
# Feedback

stepsize: 3 months
elapsed time: 25 months

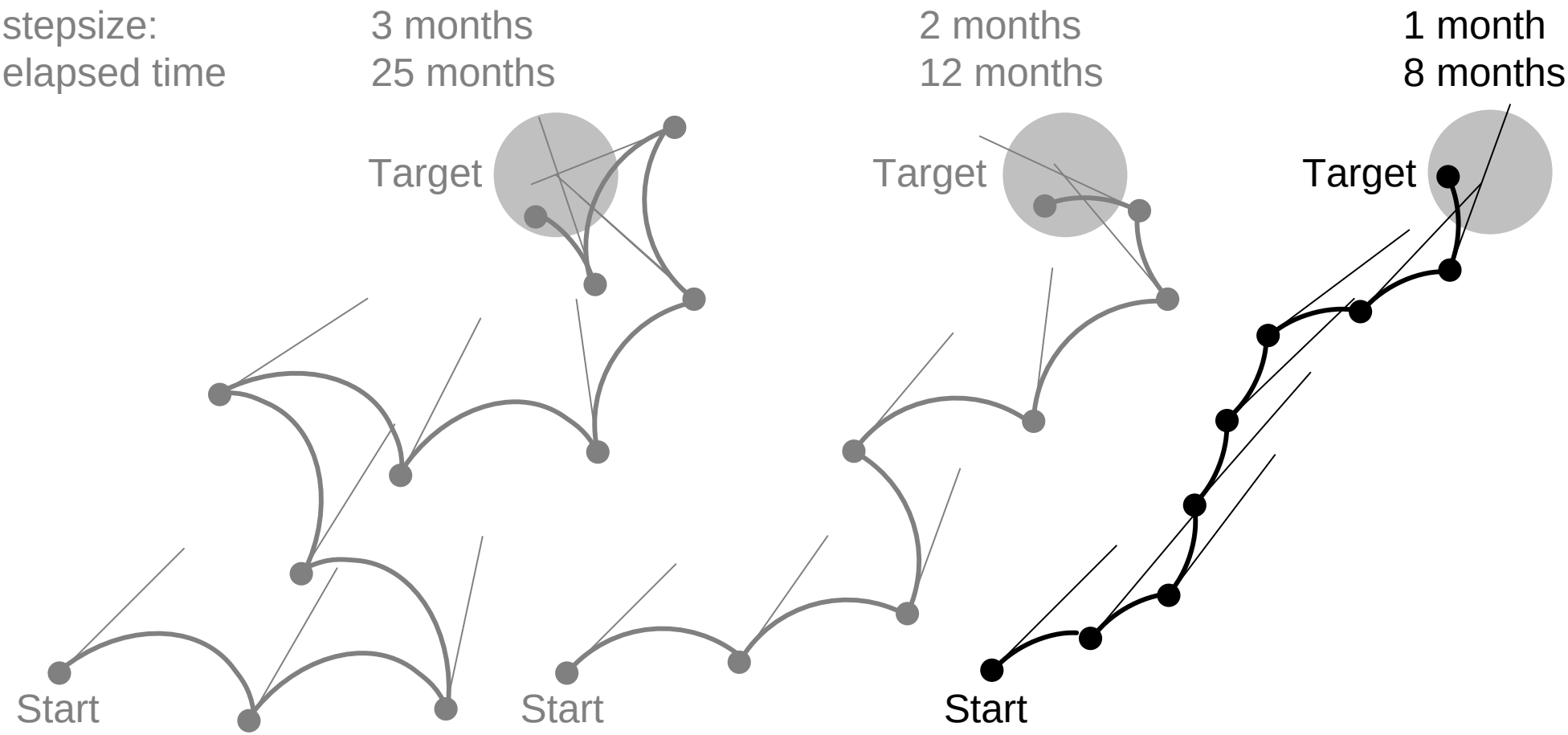


# Feedback (2)

stepsize: 3 months 2 months
elapsed time 25 months 12 months



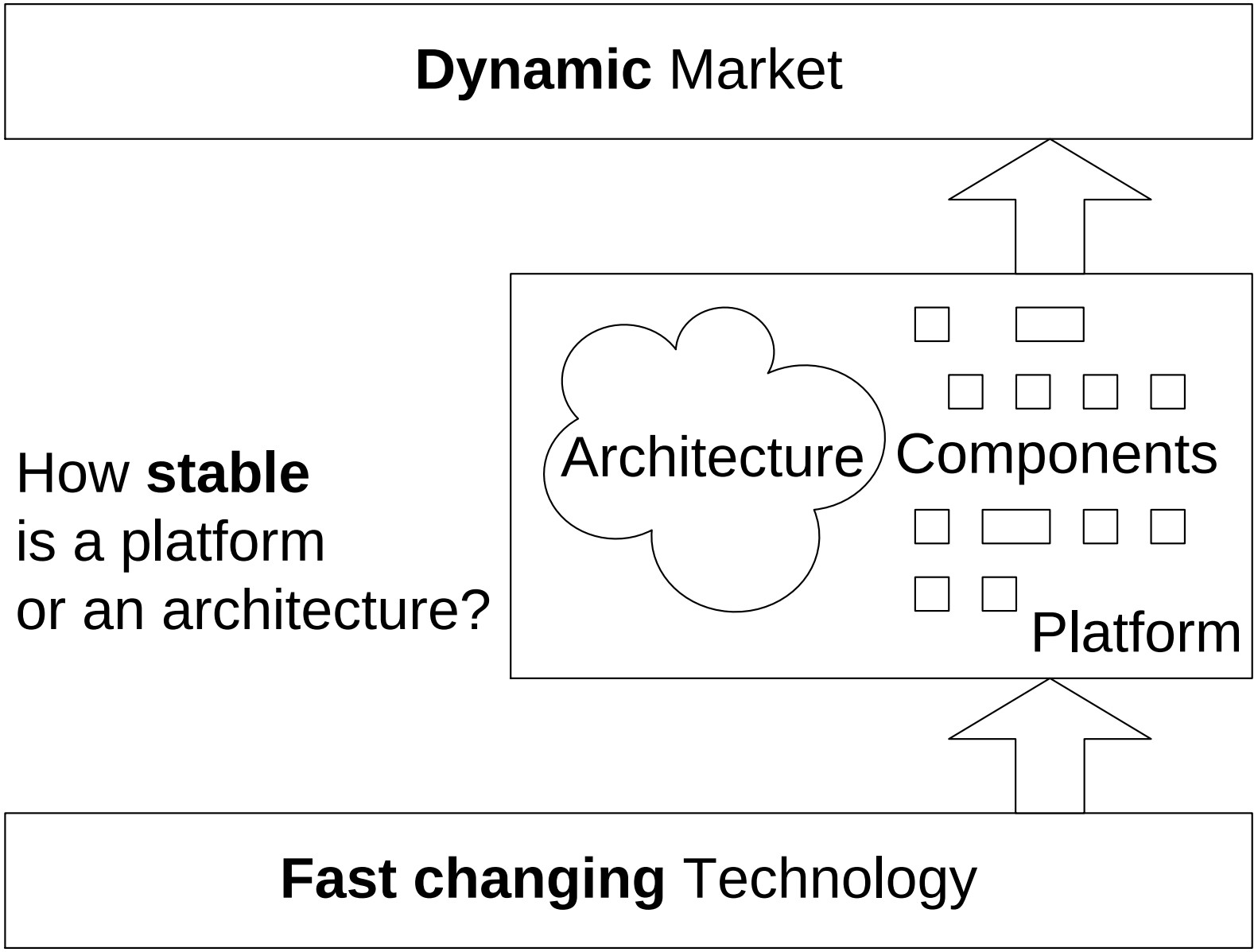
# Feedback (3)



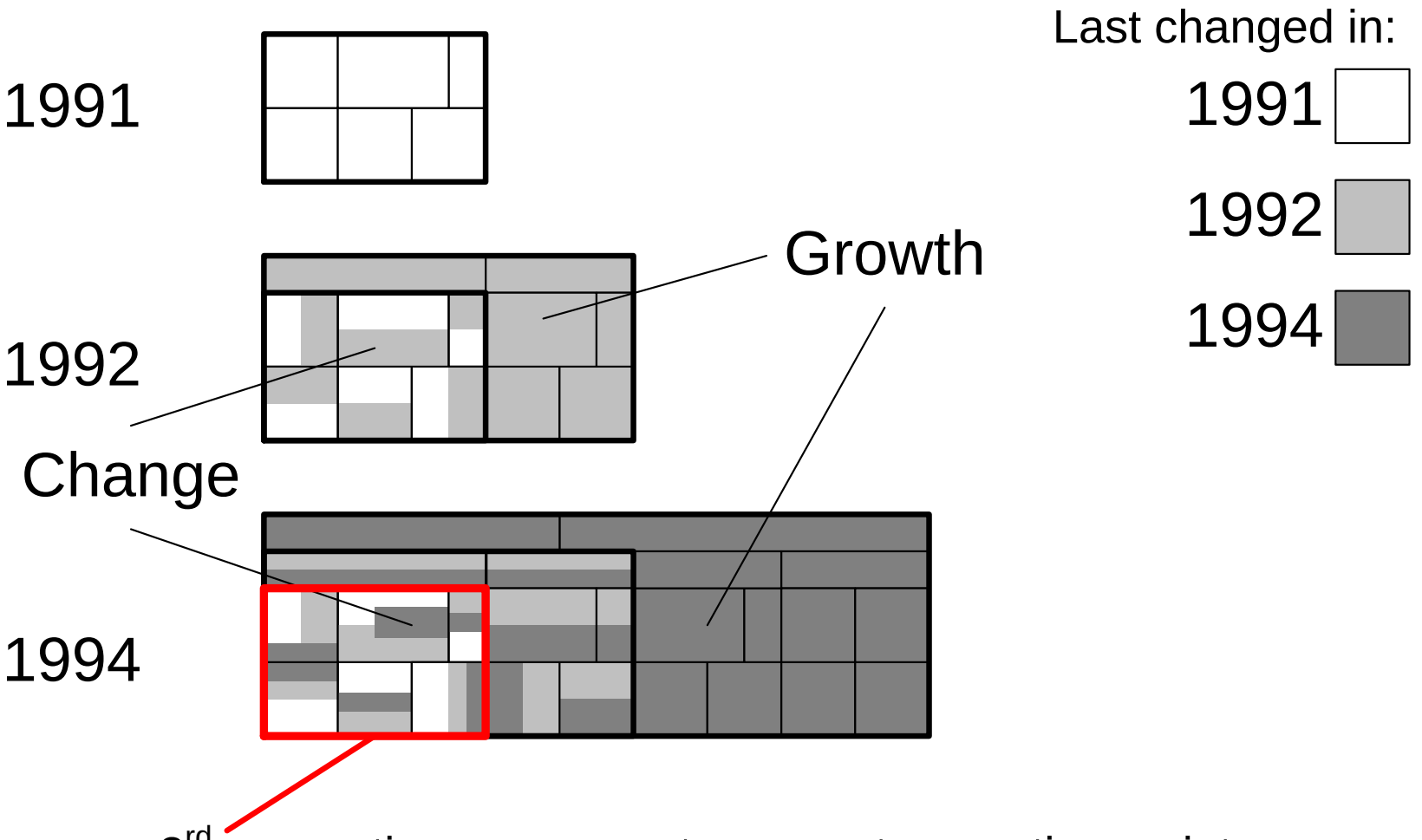

Small feedback cycles result in Faster Time to Market

# Awareness of dynamics

# Myth: Platforms are Stable



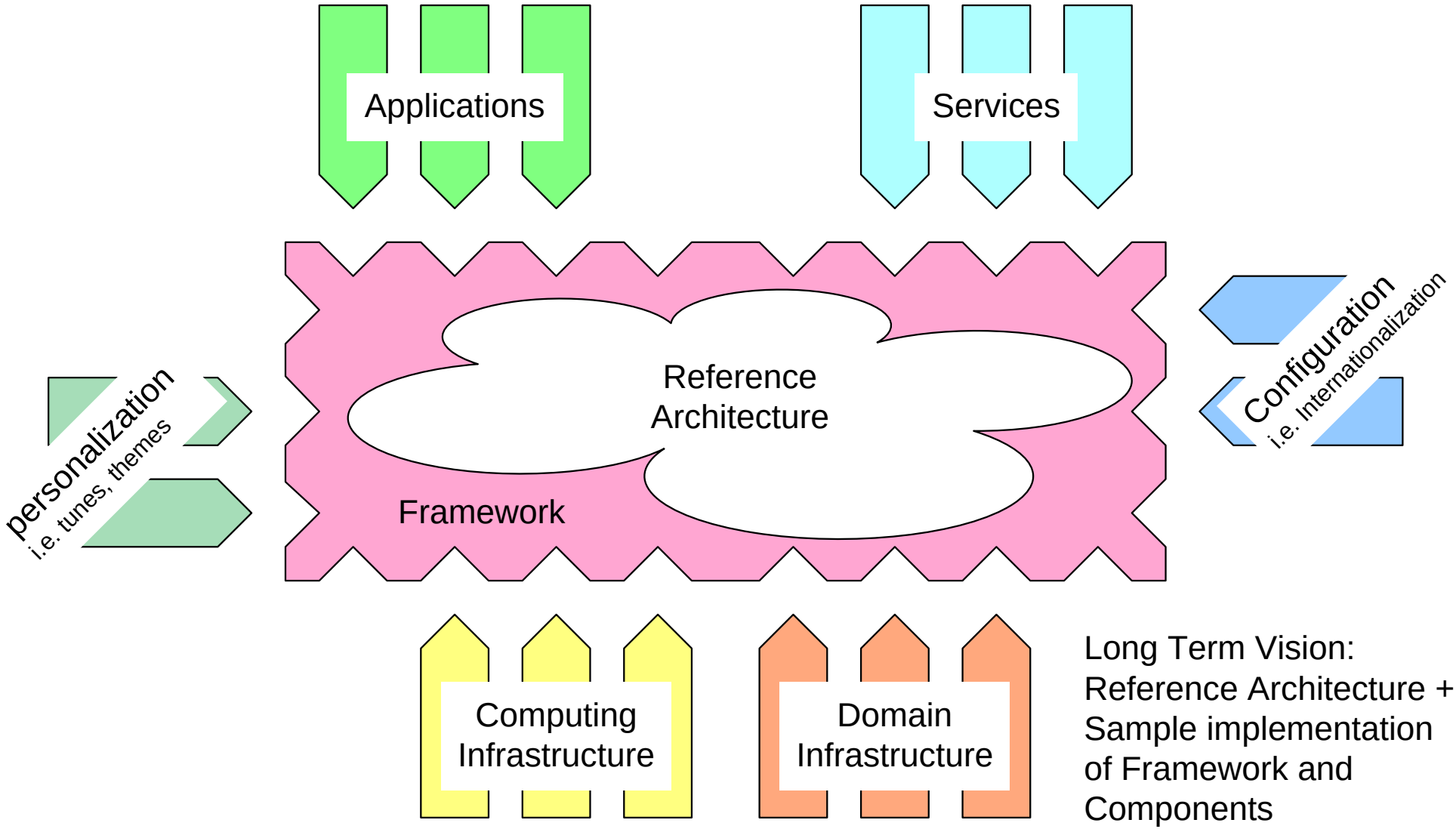
# Platform Evolution (Easyvision 1991-1996)



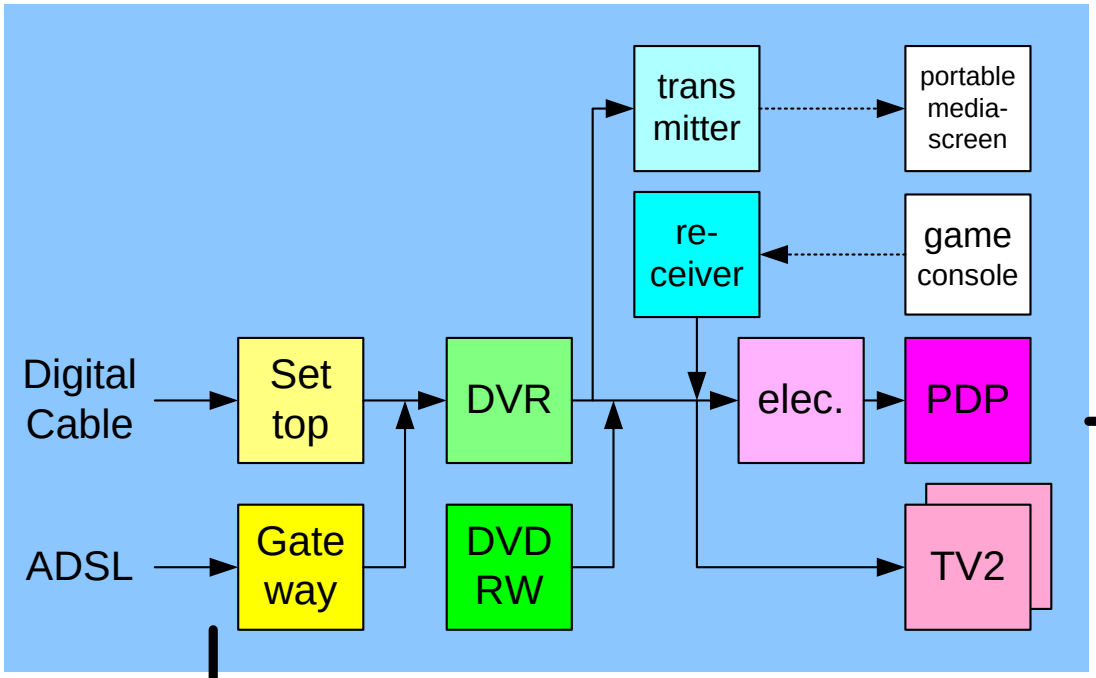

3rd generation components are mature, active maintenance needed.
Growth and change continues, some "old" components become obsolete

# Long Term Vision

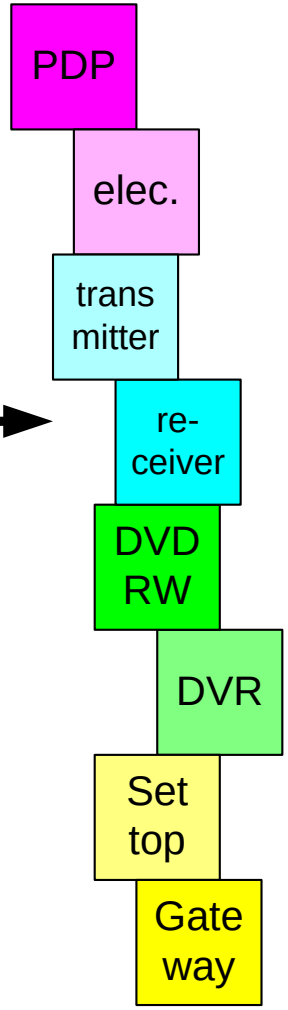
# Example Long Term Vision



# Don’t do

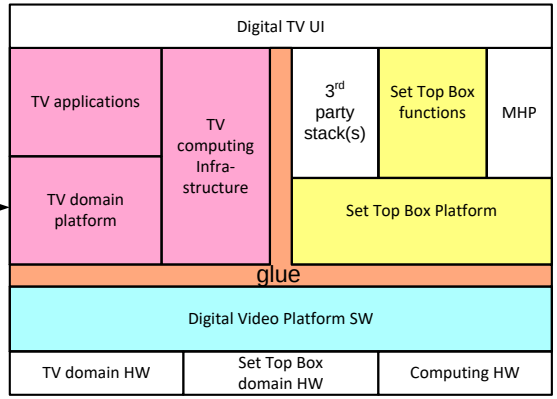



Opportunistic Legacy Integration




Proclaimed reuse

Digital TV



# Conclusion: Refactoring the Architecture is a must