

Perpetual Development

Dror Feitelson

Basic Seminar on Software Engineering
Hebrew University
2009

Lifecycle Models

- Waterfall
- Spiral
- Unified process
- Agile / extreme

Lifecycle Models

- Waterfall

Essentially serial

- Spiral

- Unified process

Iterative and
incremental

- Agile / extreme

Lifecycle Models

- Waterfall
- Spiral
- Unified process

- Agile / extreme

Formal and heavily
documented

Just do it

Lifecycle Models

- Waterfall
- Spiral
- Unified process

- Agile / extreme

Emphasize
development
till first release

Continuous

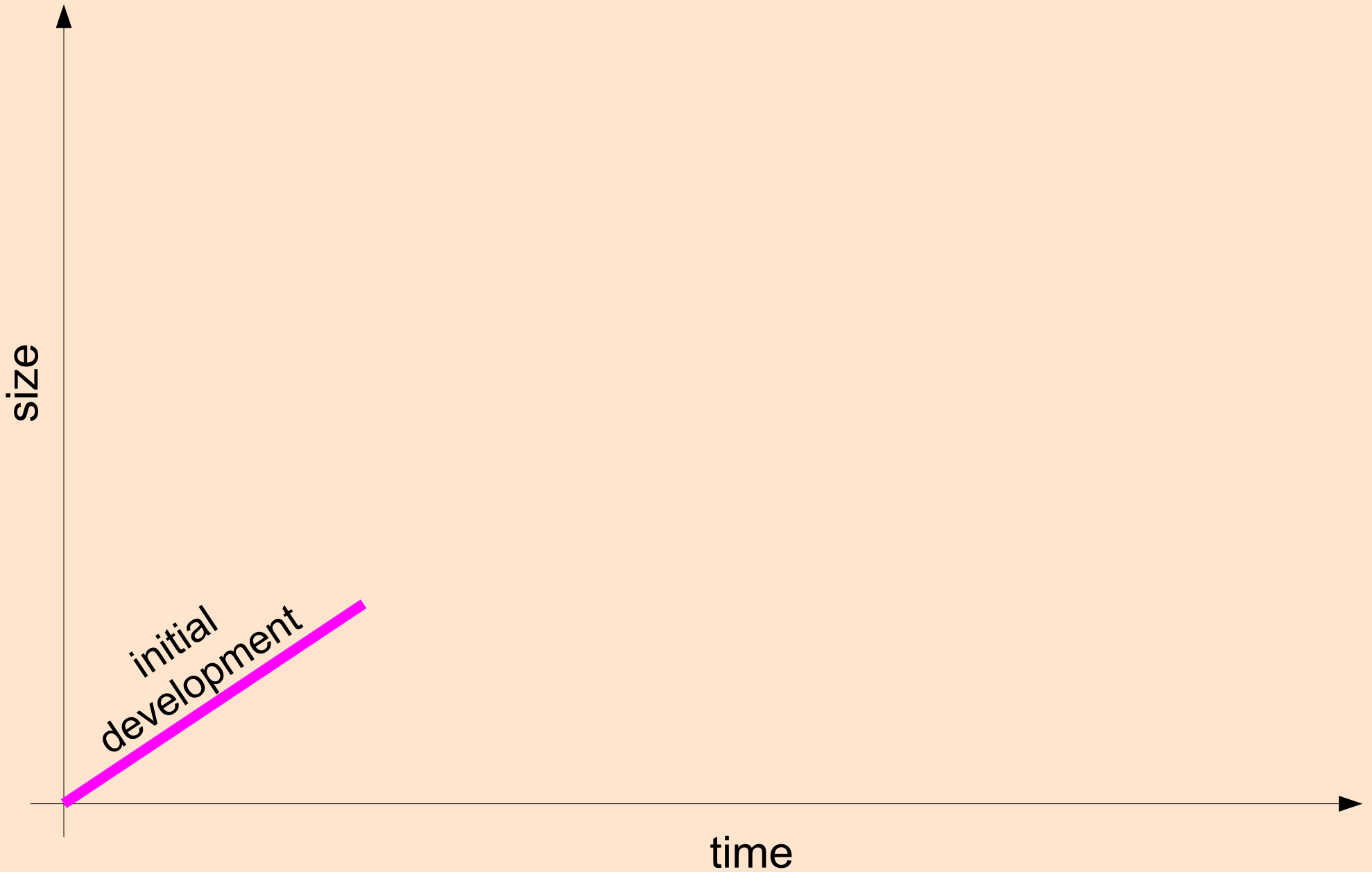
“Perpetual” Terminology

Maintenance \Rightarrow Evolution

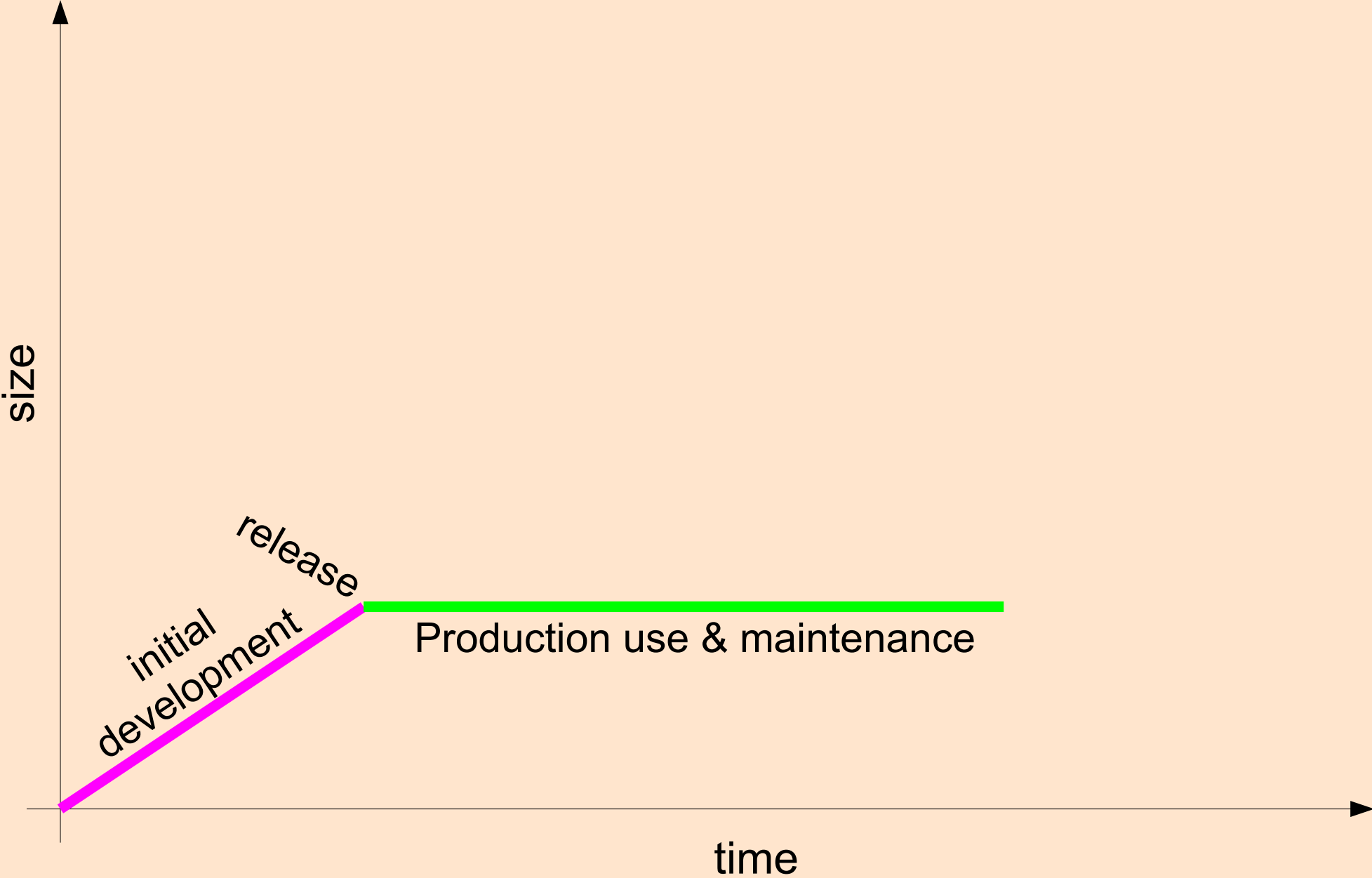
Delivery \Rightarrow Release

Requirements \Rightarrow Feature requests

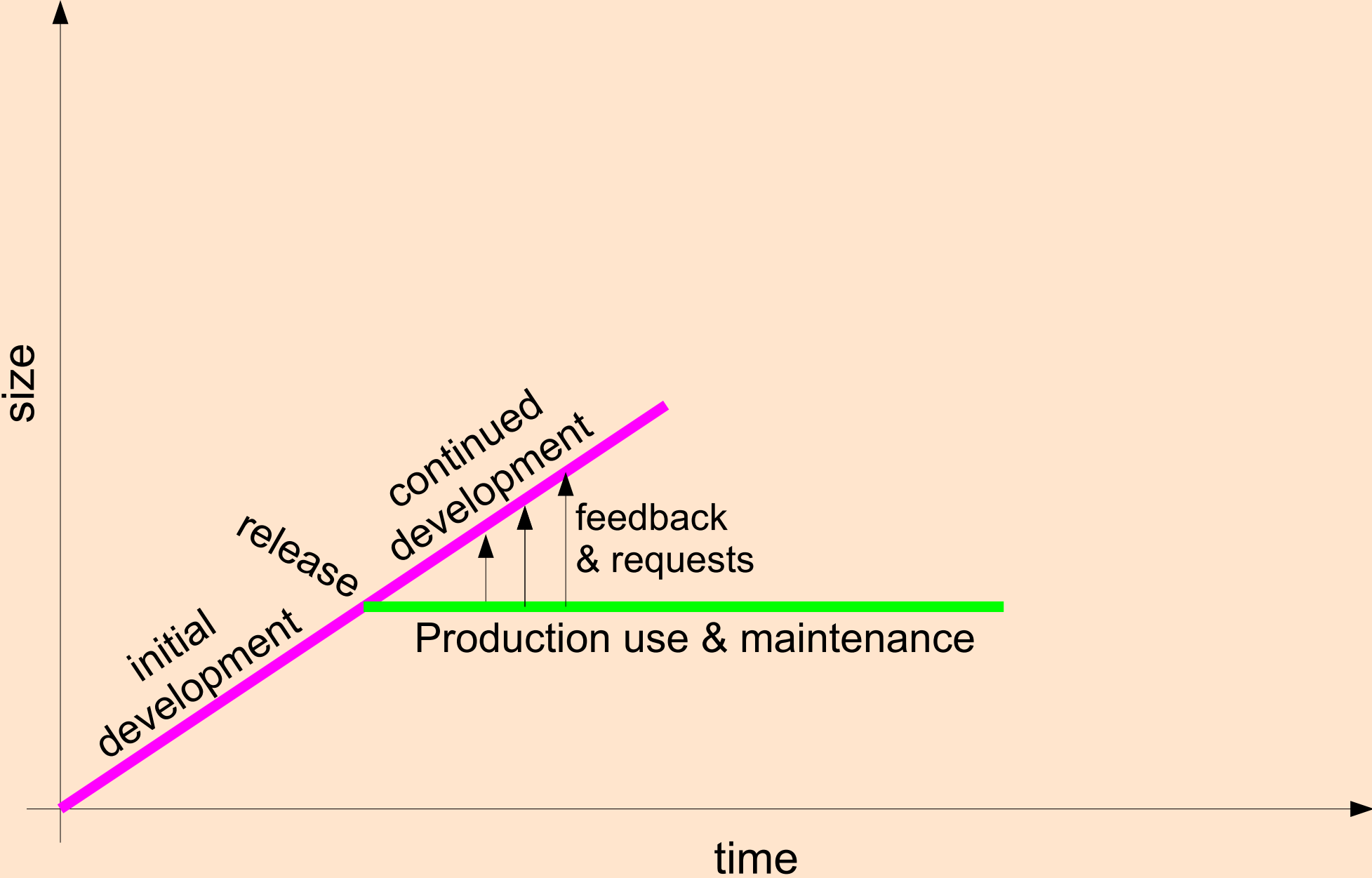
Perpetual Development Lifecycle



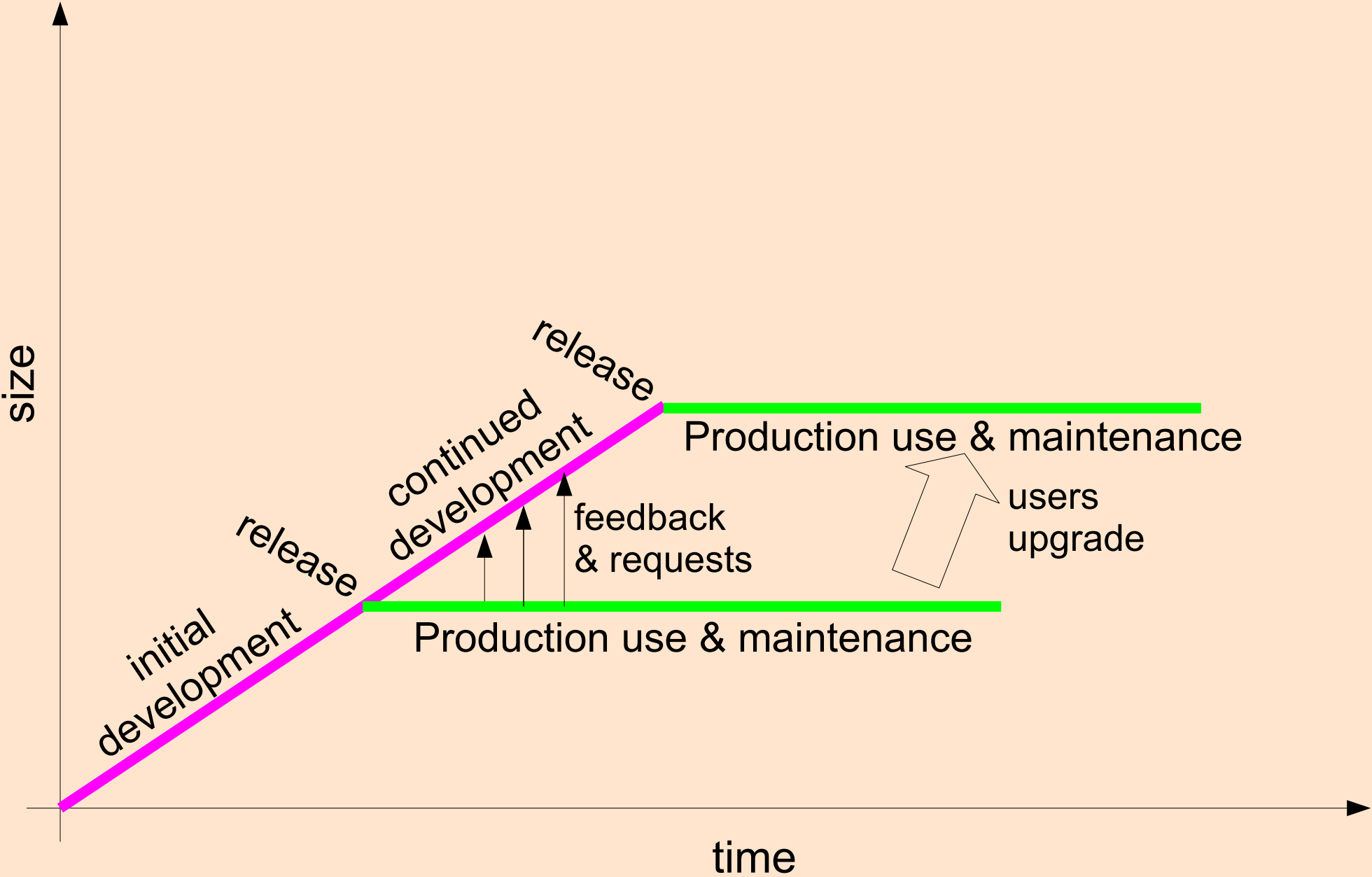
Perpetual Development Lifecycle



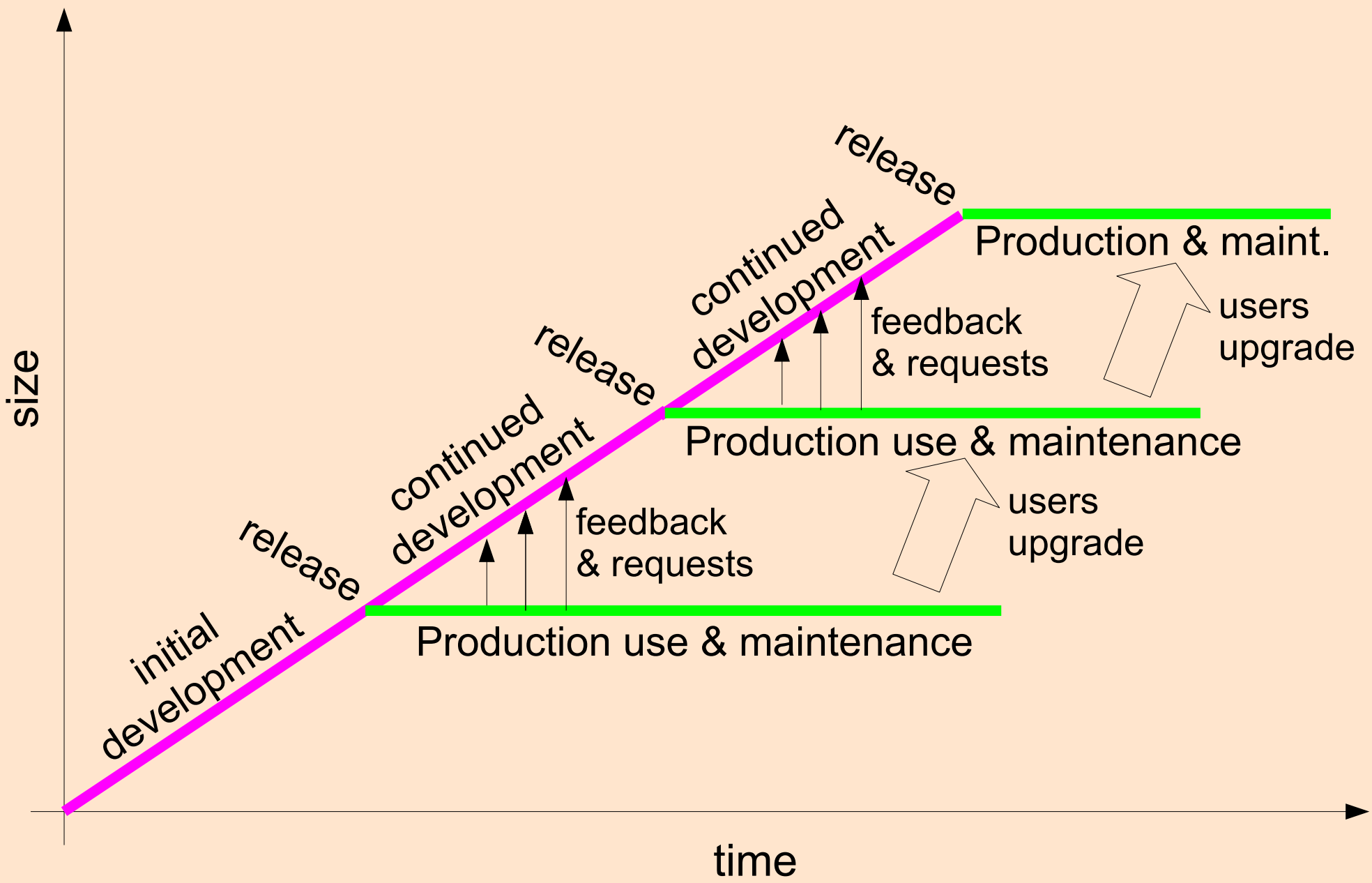
Perpetual Development Lifecycle



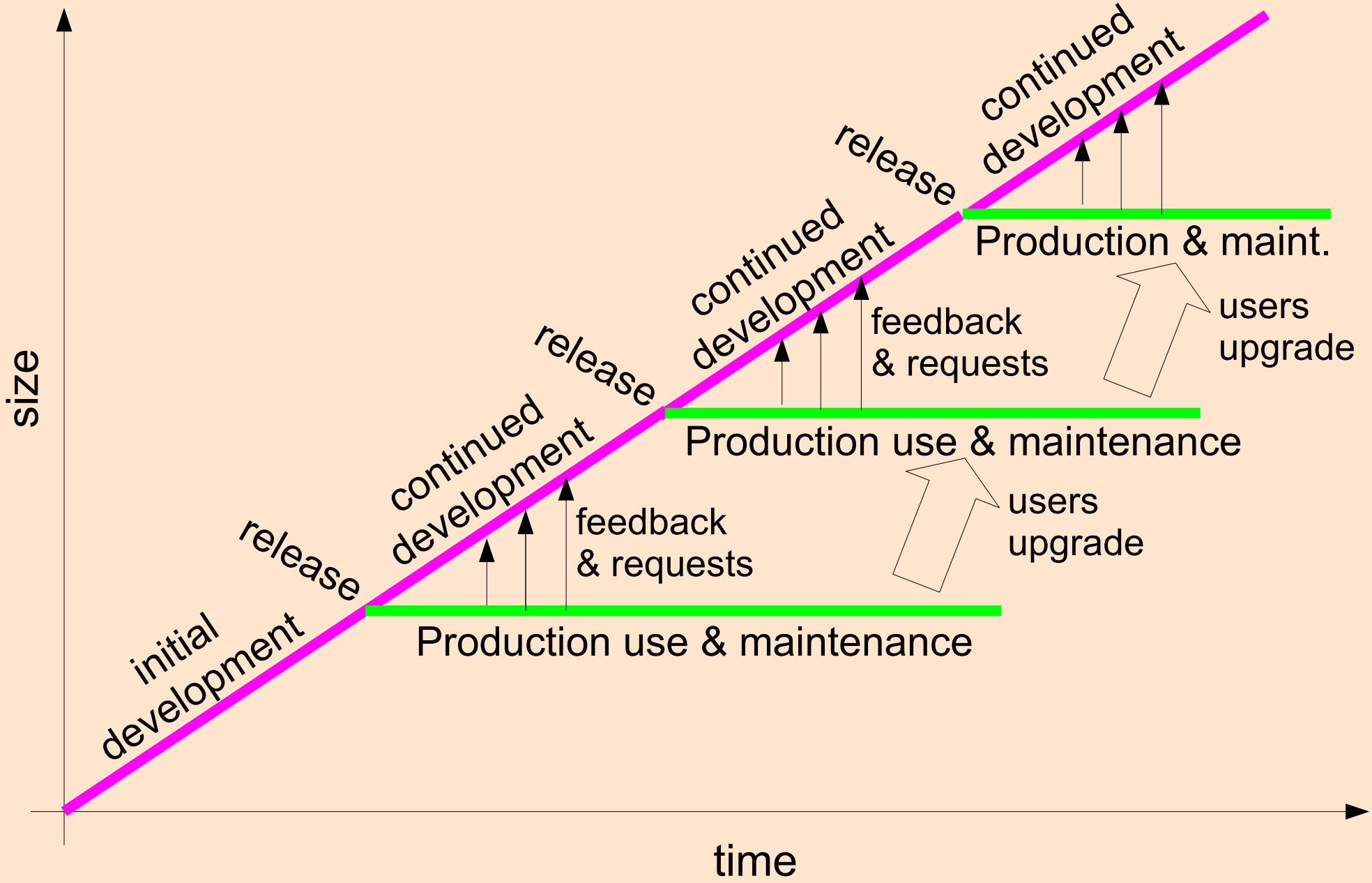
Perpetual Development Lifecycle



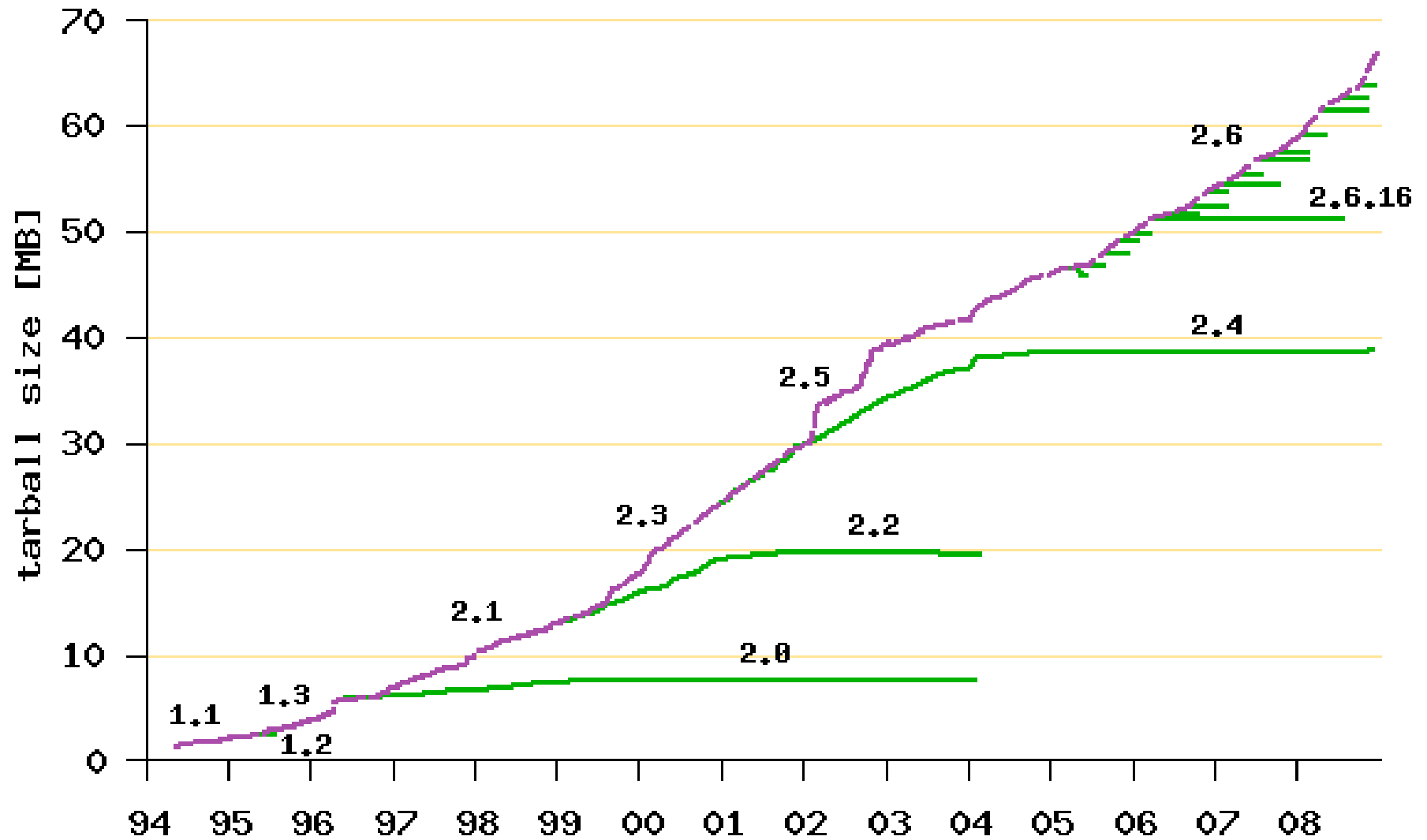
Perpetual Development Lifecycle



Perpetual Development Lifecycle



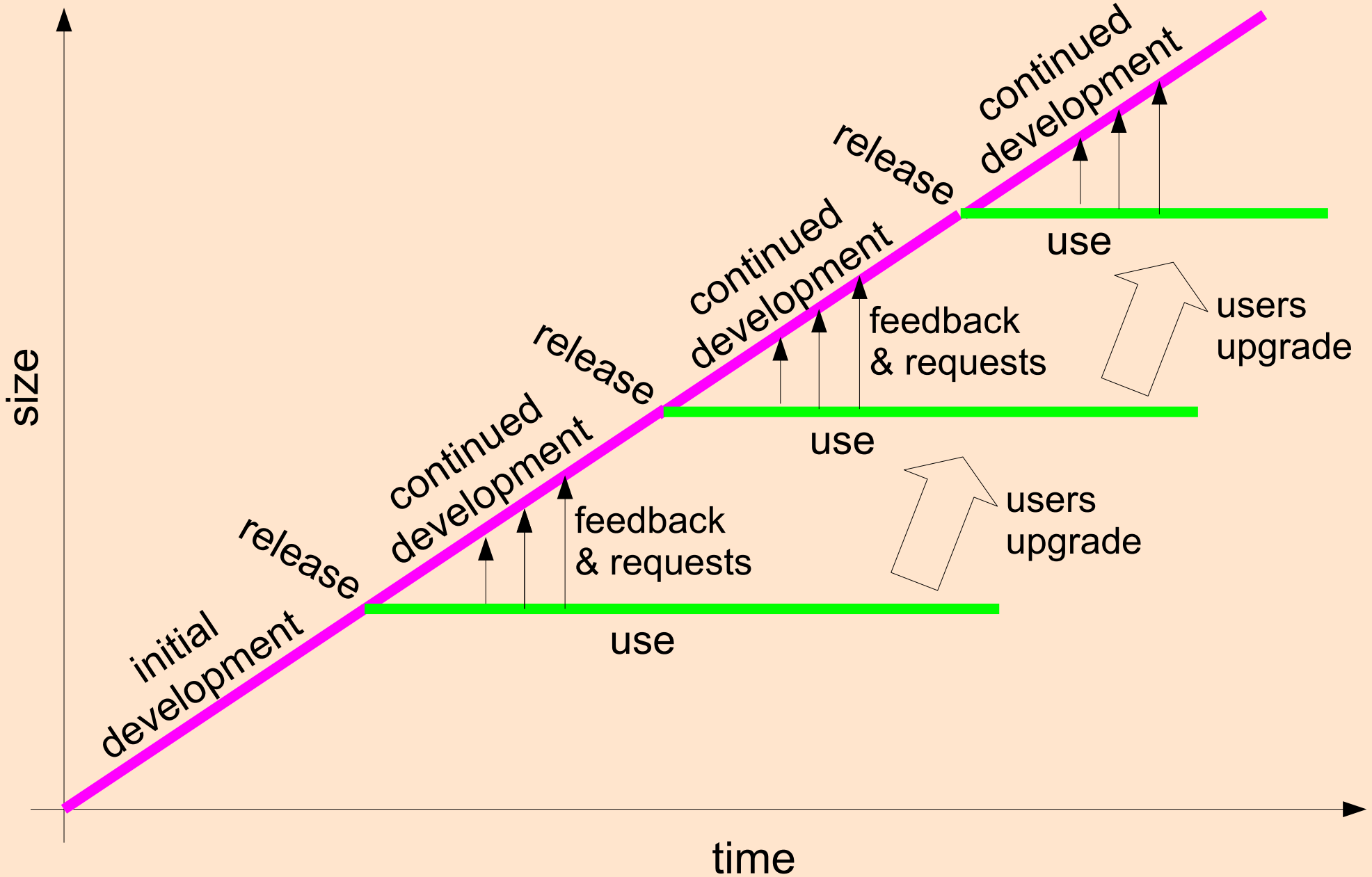
Linux Example



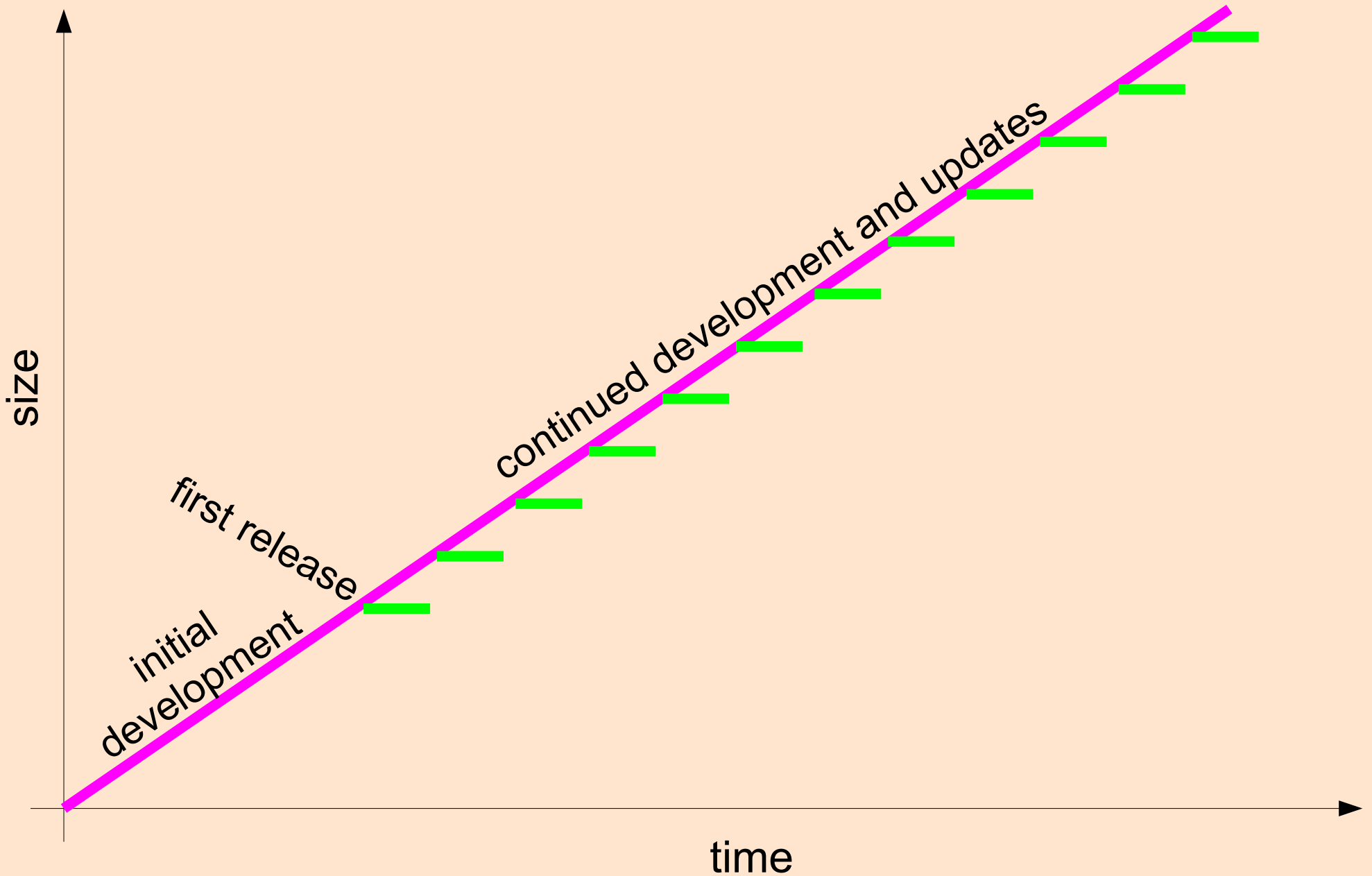
Perpetual Development Benefits

- Lead time to first working version is short, and a working version is always available
 - ⇒ No danger of the project coming to nothing
- Real users doing real work are effectively brought into the development cycle
 - ⇒ Helps to test system functionality and find problems
 - ⇒ Used to prioritize further development according to what is really needed
- Ability to use new technology as it becomes available

Variant: Shrink Wrapped



Variant: Web sites / Cloud



Implications for Development

- No fixed goal that has to be reached
- Goal is to continually improve the system and maintain its usefulness
 - ⇒ Monitor system usage to identify inadequacies
 - ⇒ Prioritize according to user needs
 - ⇒ Don't plan too far ahead (YAGNI protection)
- Use contracts that take longevity into account
 - ⇒ Support for continued evolution
 - ⇒ Access to code if company becomes insolvent

Implications for Architecture

- Can't decide on architecture based on analysis of all the requirements
- Need architecture that accommodates change
 - ⇒ Two tiers: stable core and evolving libraries
 - ⇒ Open system like e-commerce site based on web services
- Use refactoring
- May need to abandon project eventually
 - ⇒ But may still salvage parts for a followup project

Conservation of Familiarity

- One of Lehman's laws of software evolution
- Limits the rate of progress that can be sustained
- Need specialized tools to help new team members to become familiar with the system
 - ⇒ Newbies are at a disadvantage because they didn't see how the system developed
 - ⇒ Need to capture the history of design decisions

Explaining Monumental Failures

- Failures caused by "feature creep"
 - ◆ Developers made elaborate and beautiful plans
 - ◆ But these plans were obsolete by the time they were completed

⇒ Do exactly what is most needed at each instant
- Failures caused by successful maintenance
 - ◆ Delivered system was good for a very long time
 - ◆ But when it is to be replaced, an attempt is made to do too much at once

⇒ Make improvements continuously all the time

Applicability

- Agile development is often thought to apply only to small projects
- But some continuous projects are huge (e.g. Linux)
 - Not exactly "agile", but close enough
- In fact, many large and successful projects were not planned or engineered, but evolved
 - Linux, Internet, WWW, ...

Experimental Evidence

- Conducted by the World Wide Consortium for the Grid (W2COG)
- The goal: develop a secure service-oriented architecture system
- Traditional approach: standard government acquisition process
- Alternative: use a "Limited Technology Experiment" based on evolutionary methods
- Both start with same government supplied software baseline

18 Months Later...

Traditional:

- A concept document with no functional architecture
- Cost \$1.5M
- No concrete deployment plan or timeline

Evolutionary:

- Delivered open architecture prototype addressing 80% of requirements
- Cost of \$100K
- Plan to complete in 6 months

Bottom Line

- Expect to see many more projects using evolutionary and agile methods
- Especially in environments challenged by rapid technological progress and rapid change
- These ideas are actually not new
 - However, not articulated well till recently
 - Contradict traditional engineering approach
 - Nevertheless work well in practice