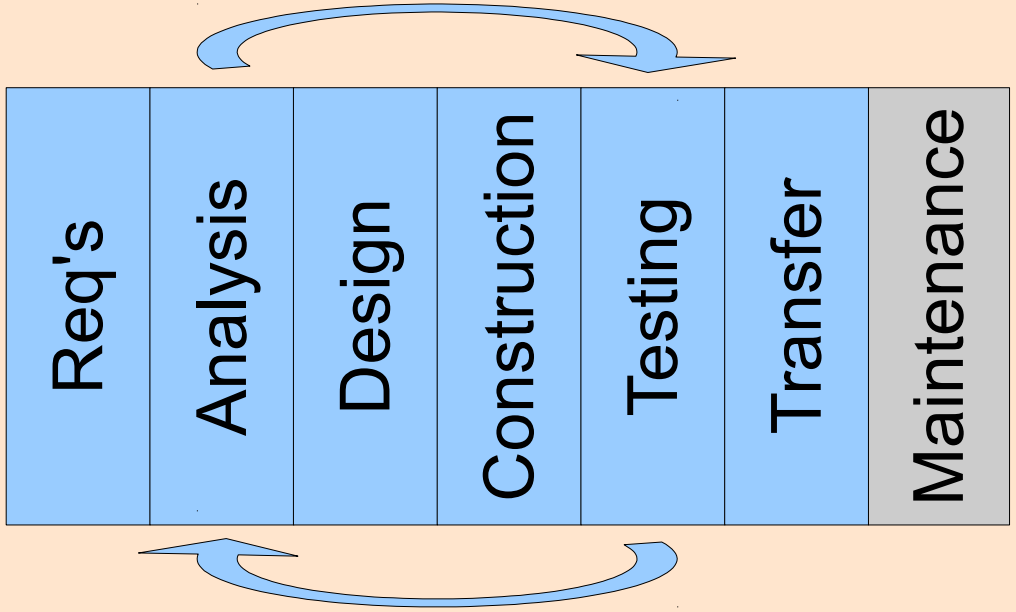
Agile Development and Software Evolution

Dror Feitelson

Basic Seminar on Software Engineering
Hebrew University
2011

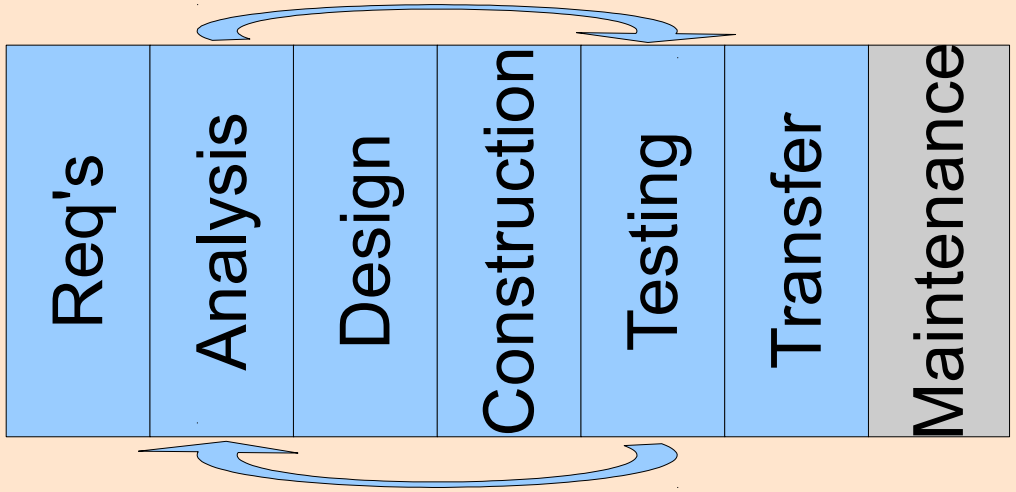
Software Lifecycle

Textbook view:

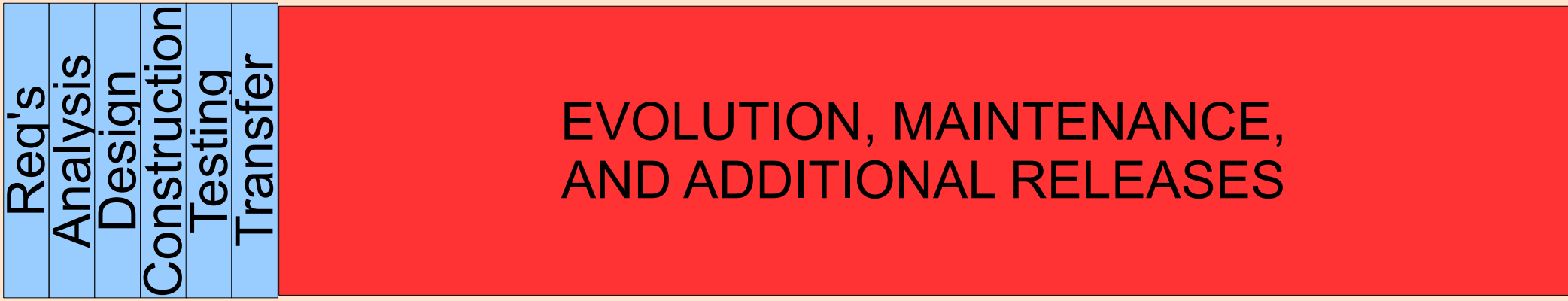


Software Lifecycle

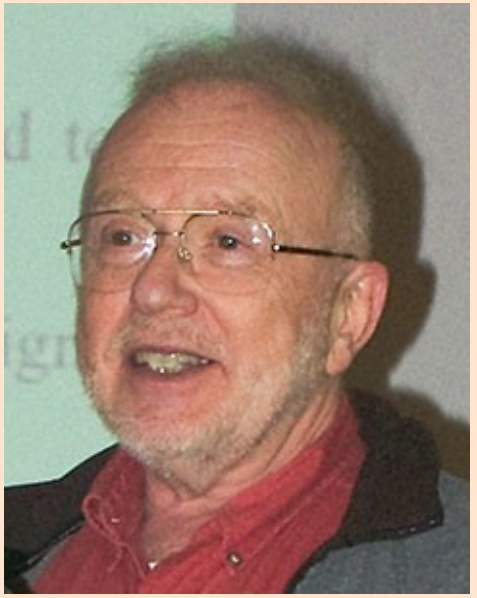
Textbook view:



More realistic view:



David Lorge Parnas

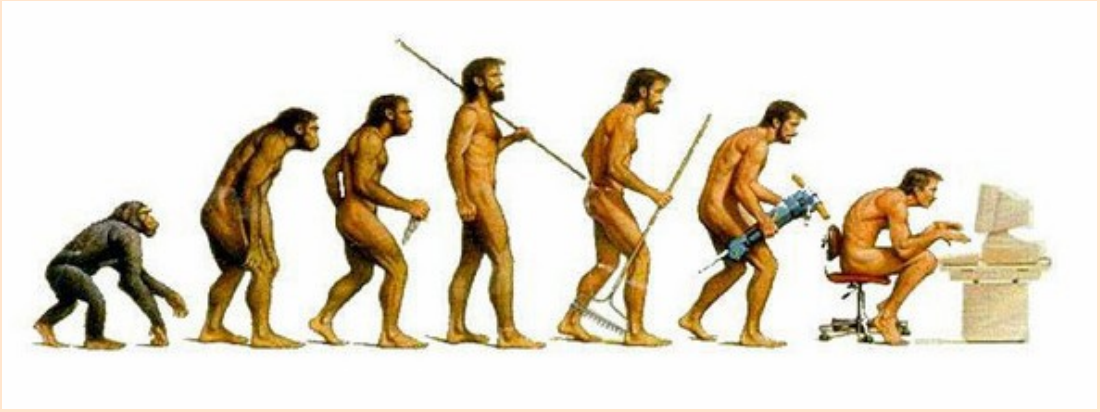


- Major contributor to information hiding and modularization
- Advocate of software development as an engineering discipline
- Including good documentation!
- Opponent of “star wars”
- Fellow of ACM, IEEE

“A sign that the Software Engineering profession has matured will be that we lose our preoccupation with the first release and focus on the long term health of our products.”

Meir (Manny) Lehman

- Built first computers in Israel in 1950s
- Worked at IBM studying OS/360
- Professor at Imperial College London
- Received Harlan Mills award
- Passed away 29.12.10 in Jerusalem



- “Father of software evolution”
- Lehman's 8 Laws describe general progress of projects
- Defined “E-type” systems that are ingrained with their environment

Lehman's Laws

1) Continuing change (adaptation)
2) Increasing complexity (unless refactored)
3) Self regulation (of rate of change)
4) Invariant work rate
5) Conservation of familiarity (of users and developers)
6) Continuing growth (more features)
7) Declining quality (unless maintained)
8) Feedback system (at multiple levels)

Lifecycle Models

- Waterfall
- Spiral
- Unified process
- Agile / extreme

Lifecycle Models

- Waterfall

Essentially serial

- Spiral
- Unified process
- Agile / extreme

Iterative and incremental

Lifecycle Models

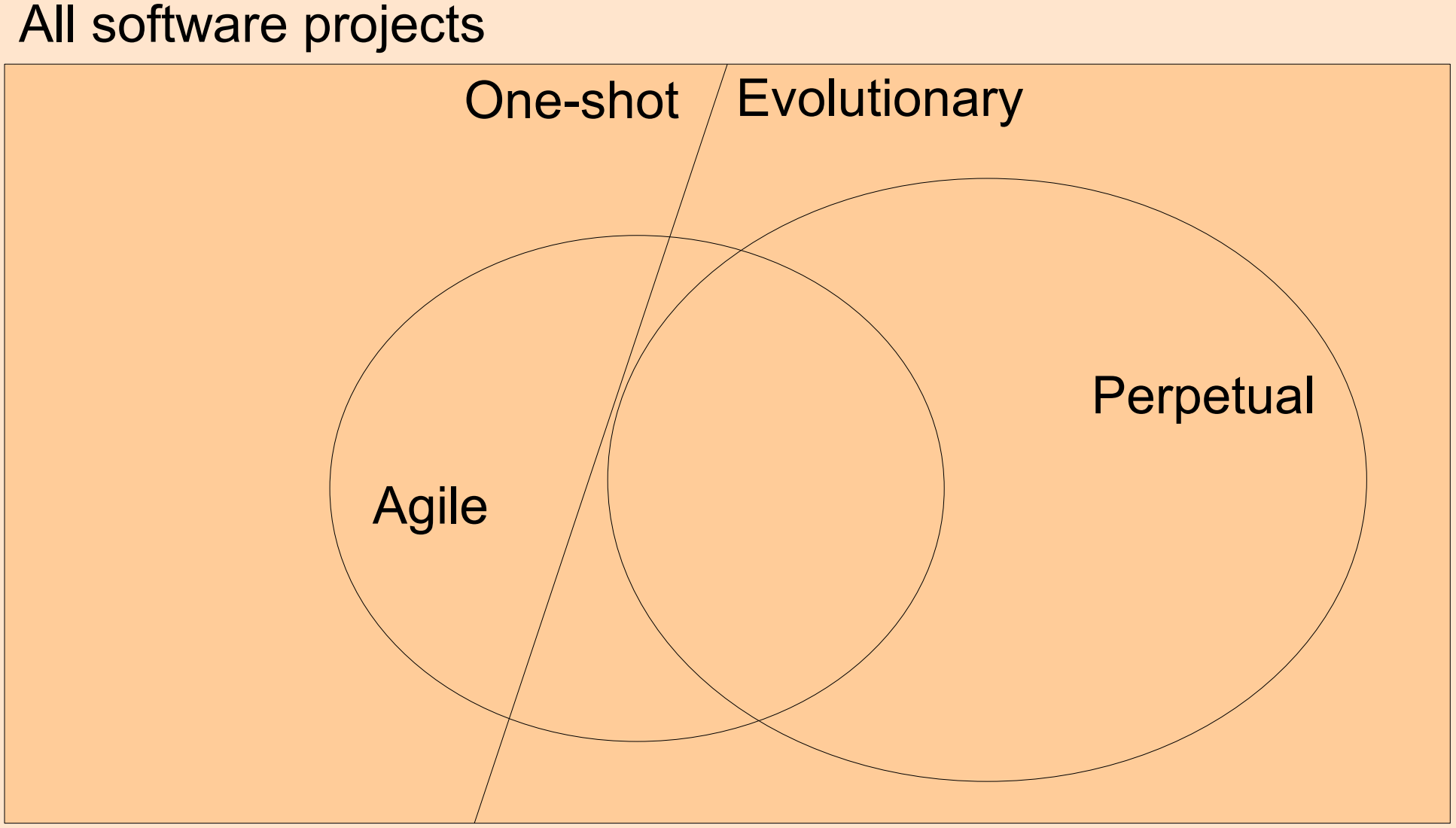
- Waterfall
- Spiral
- Unified process

Formal and heavily documented

- Agile / extreme

Just do it

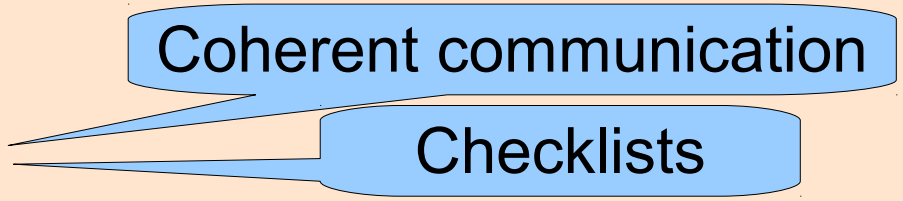
Evolution and Agile



Compromises

- Cost, quality, and schedule – pick any two
- Traditional: quality (aka requirements) are paramount
  - Heroic efforts to achieve them
  - Often overrun budget and/or schedule
- Agile: schedule is paramount
  - Continuously decide what you can do and what your priorities are
  - Keep sustainable work practices

Agile



- A well defined process
  - Even if not strong on documents or formal planning
- Evolutionary approach
  - “Embrace change” (as opposed to dreaded feature creep)
  - Steer project based on user priorities: commit to user, not to predefined plan
- Not all evolutionary/perpetual projects are agile
  - e.g. Linux and other open-source projects that have little if any process

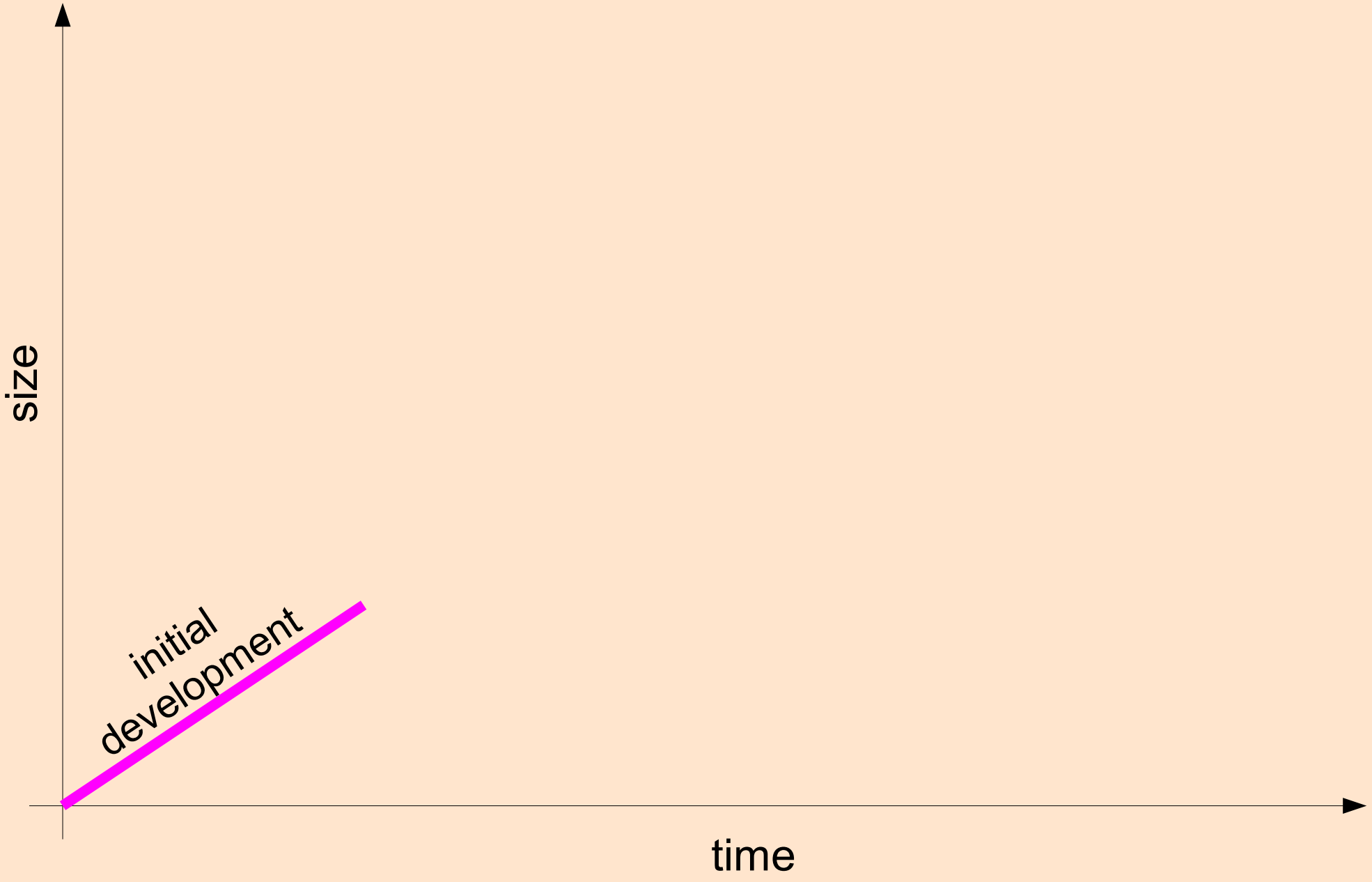
“Perpetual” Terminology

Maintenance $\Rightarrow$ Evolution

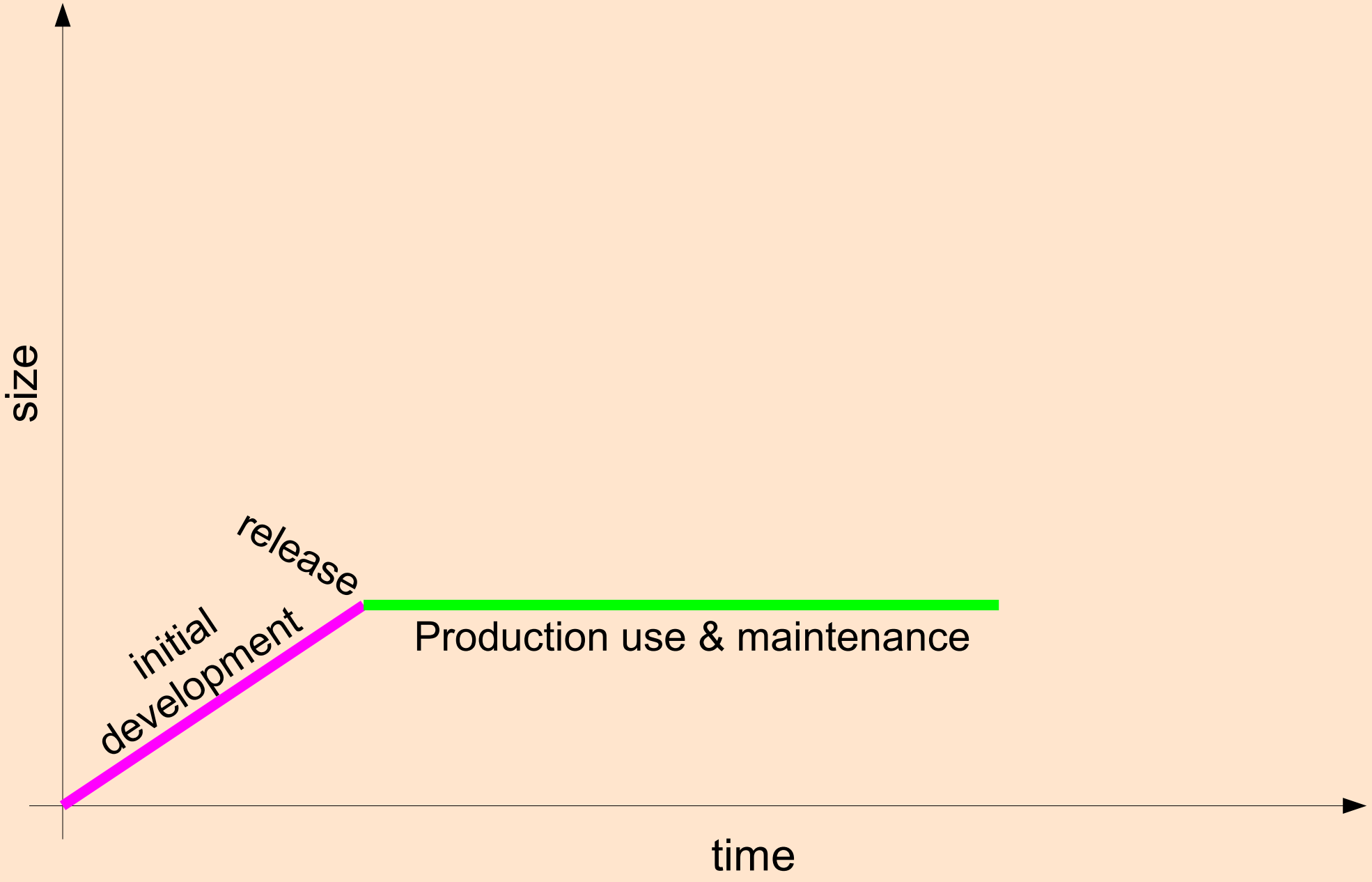
Delivery $\Rightarrow$ Release

Requirements $\Rightarrow$ Feature requests

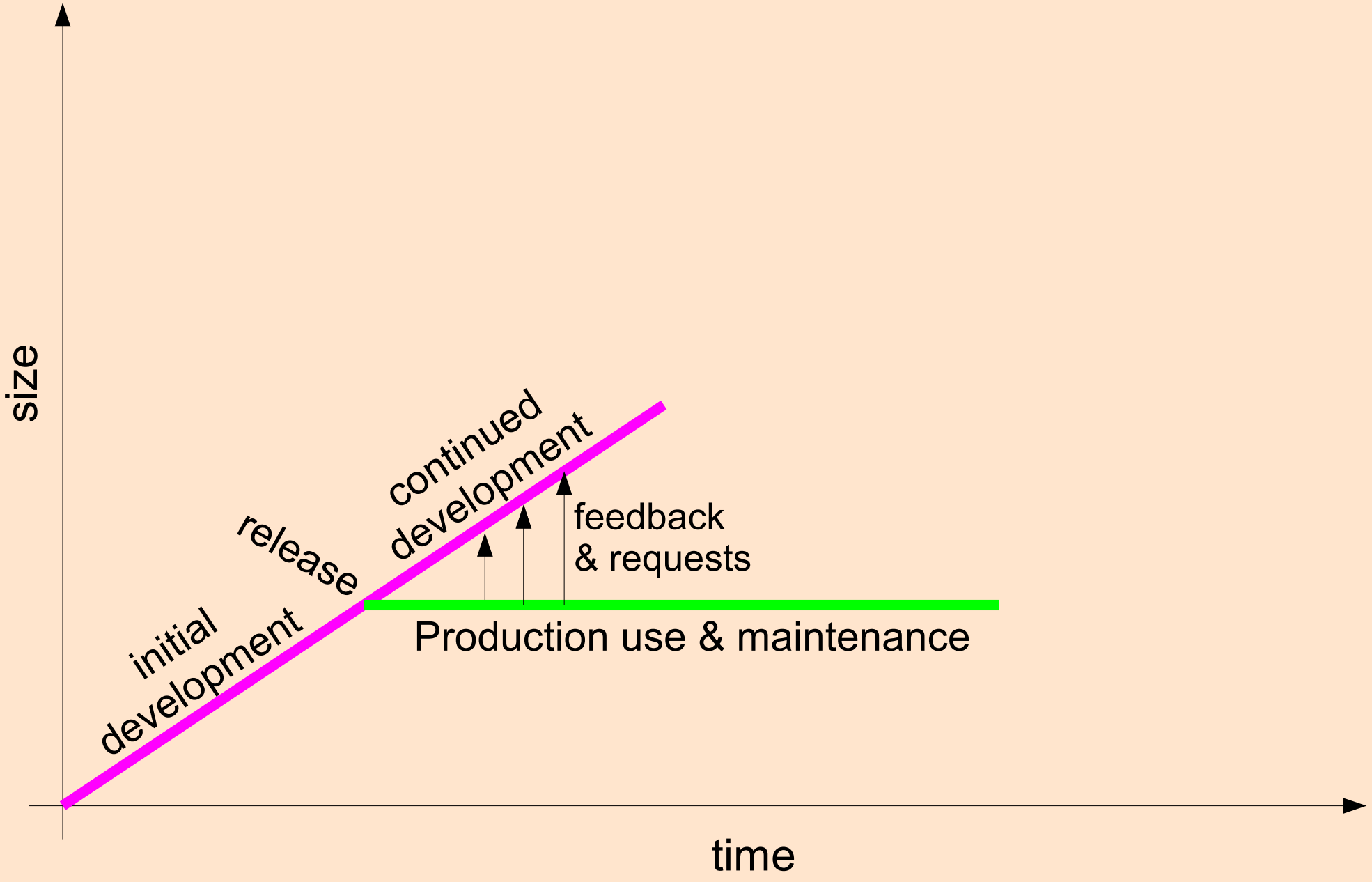
Perpetual Development Lifecycle



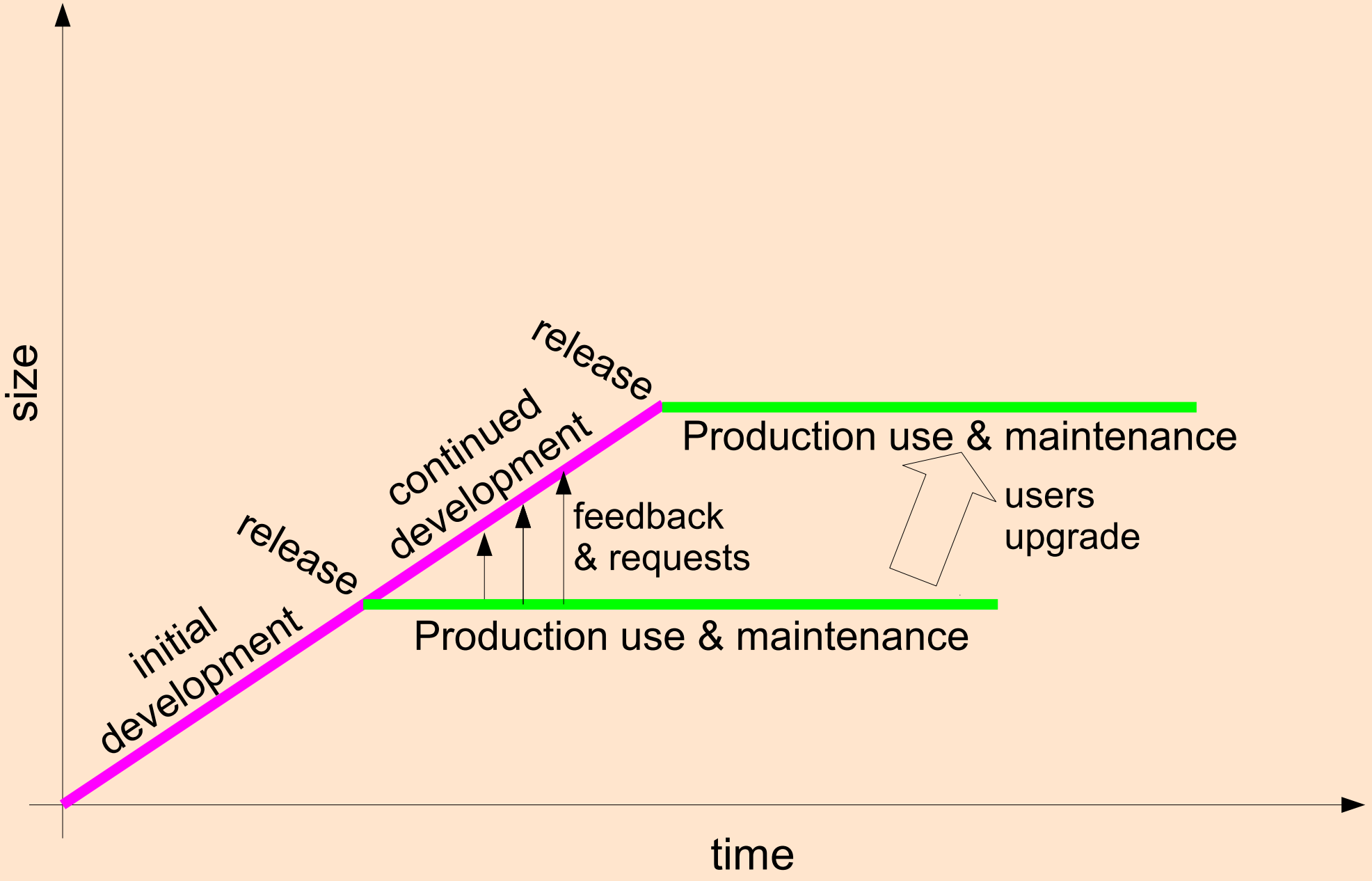
Perpetual Development Lifecycle



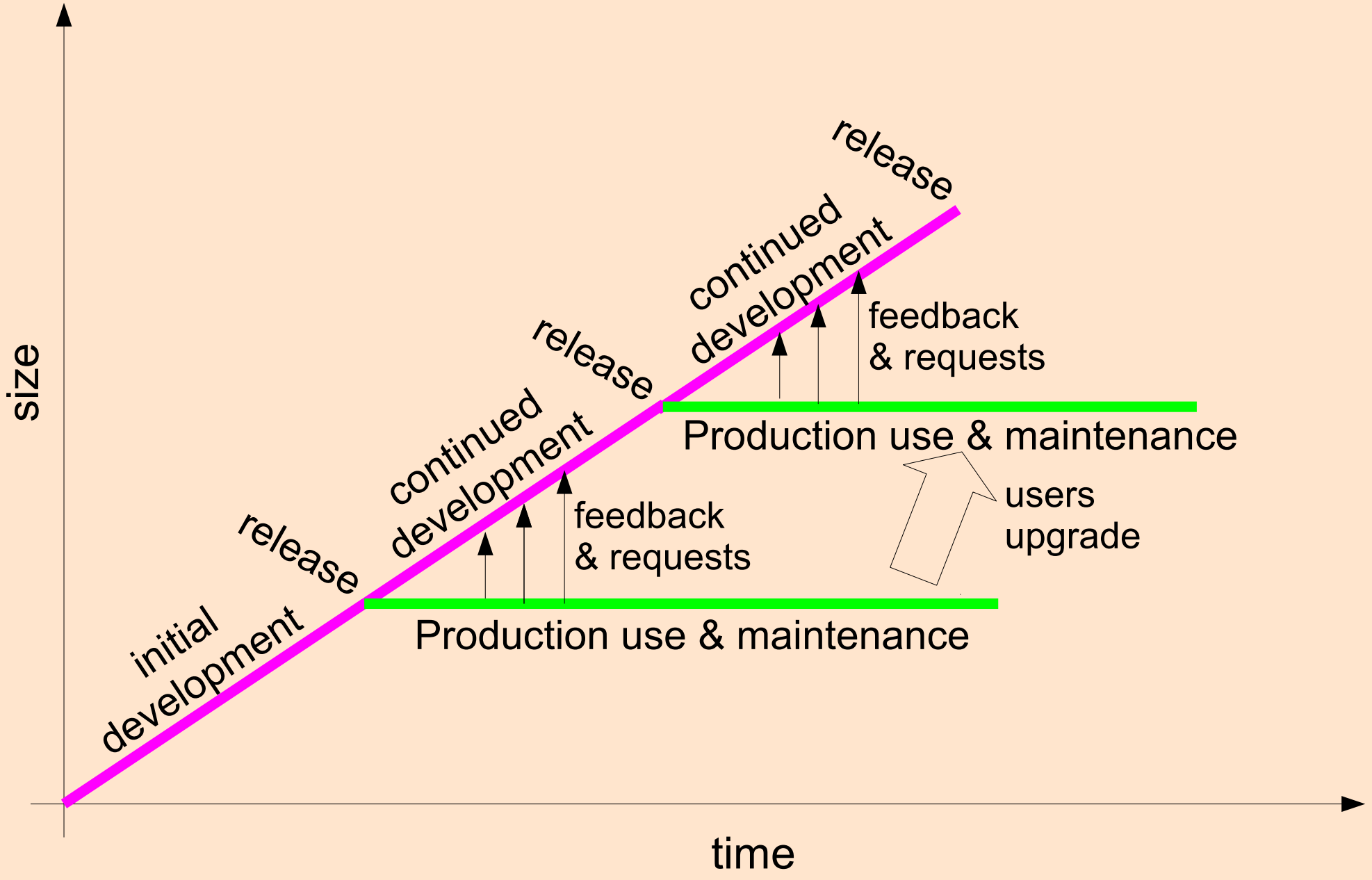
Perpetual Development Lifecycle



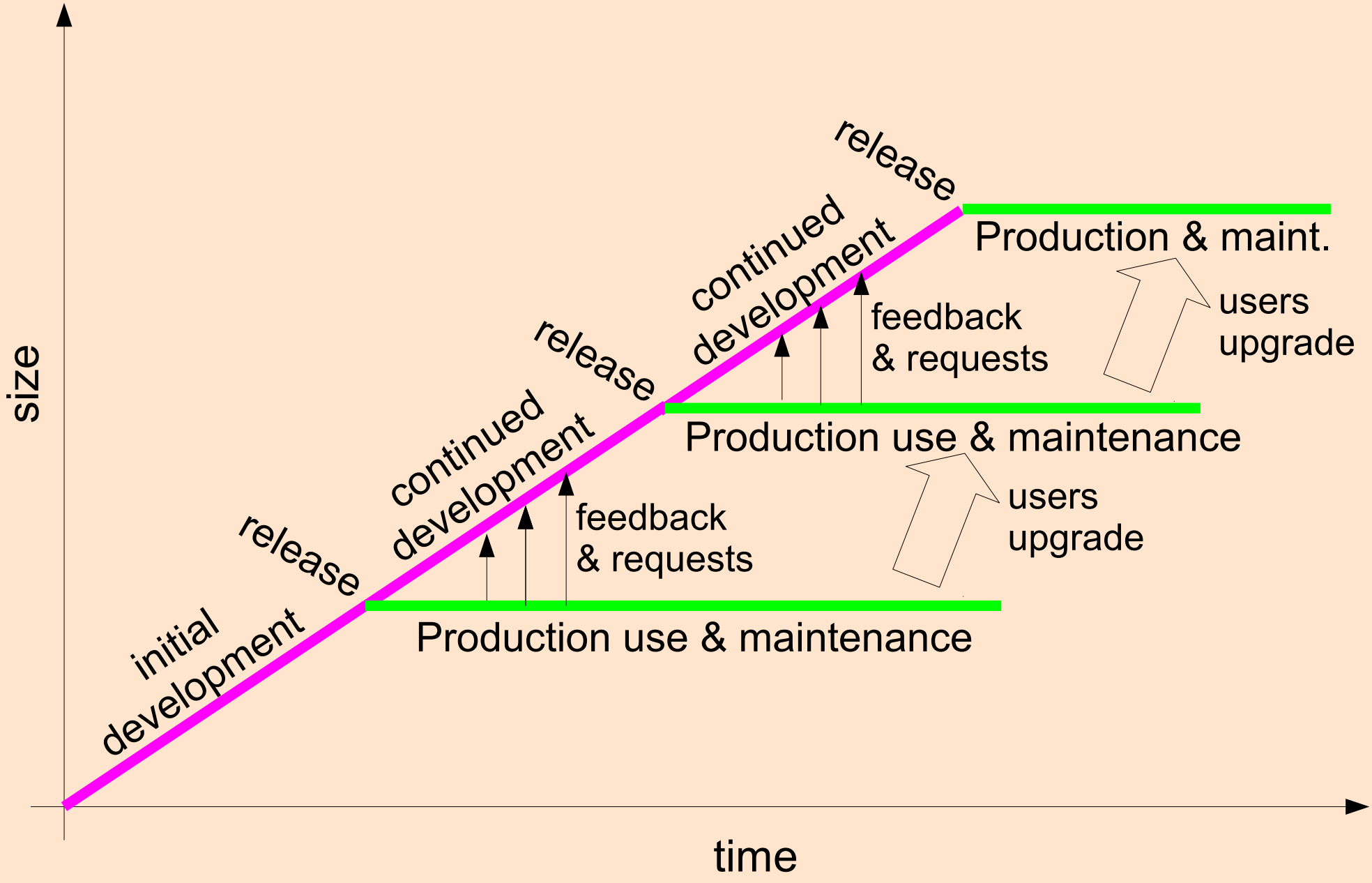
Perpetual Development Lifecycle



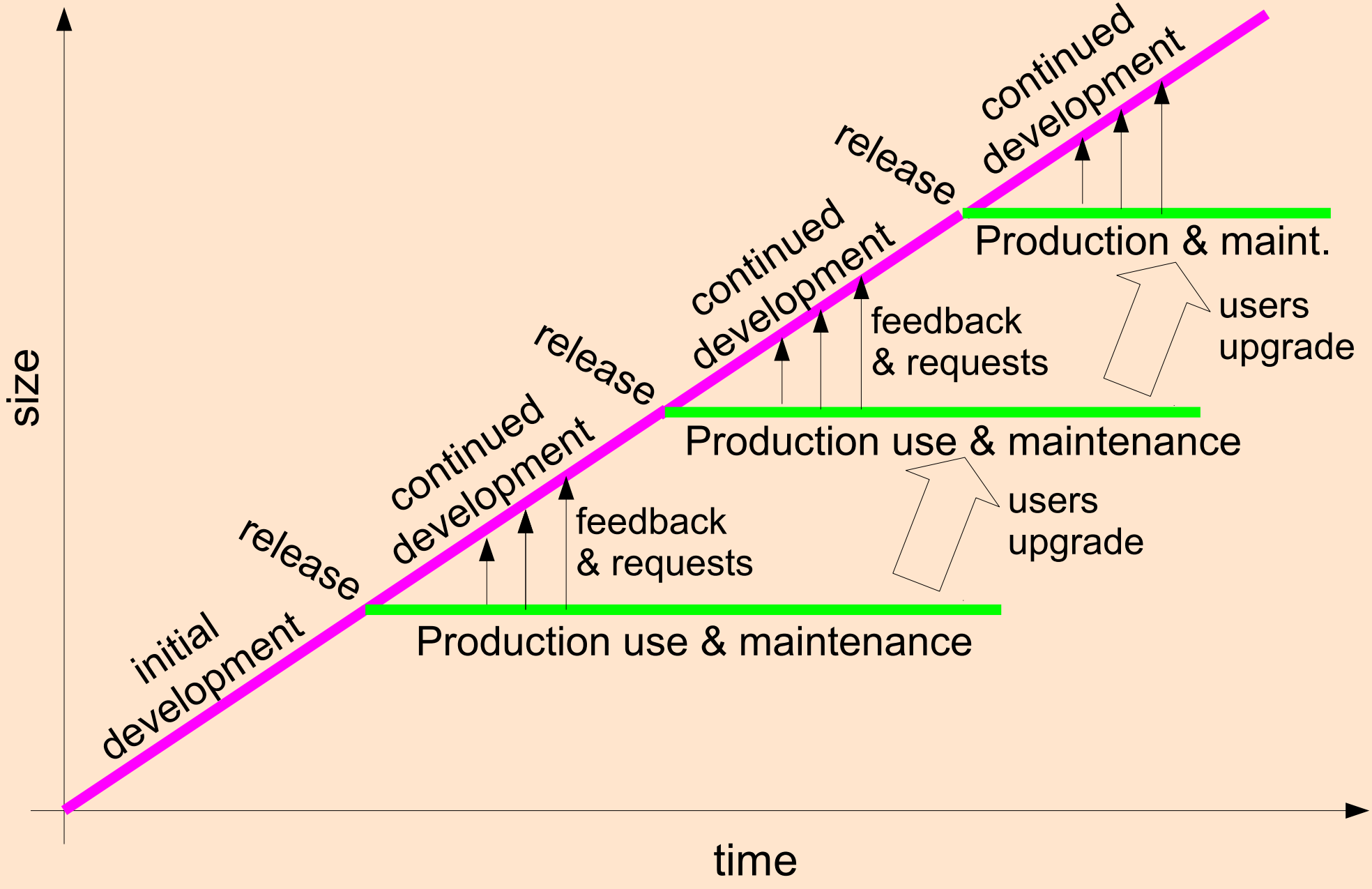
Perpetual Development Lifecycle



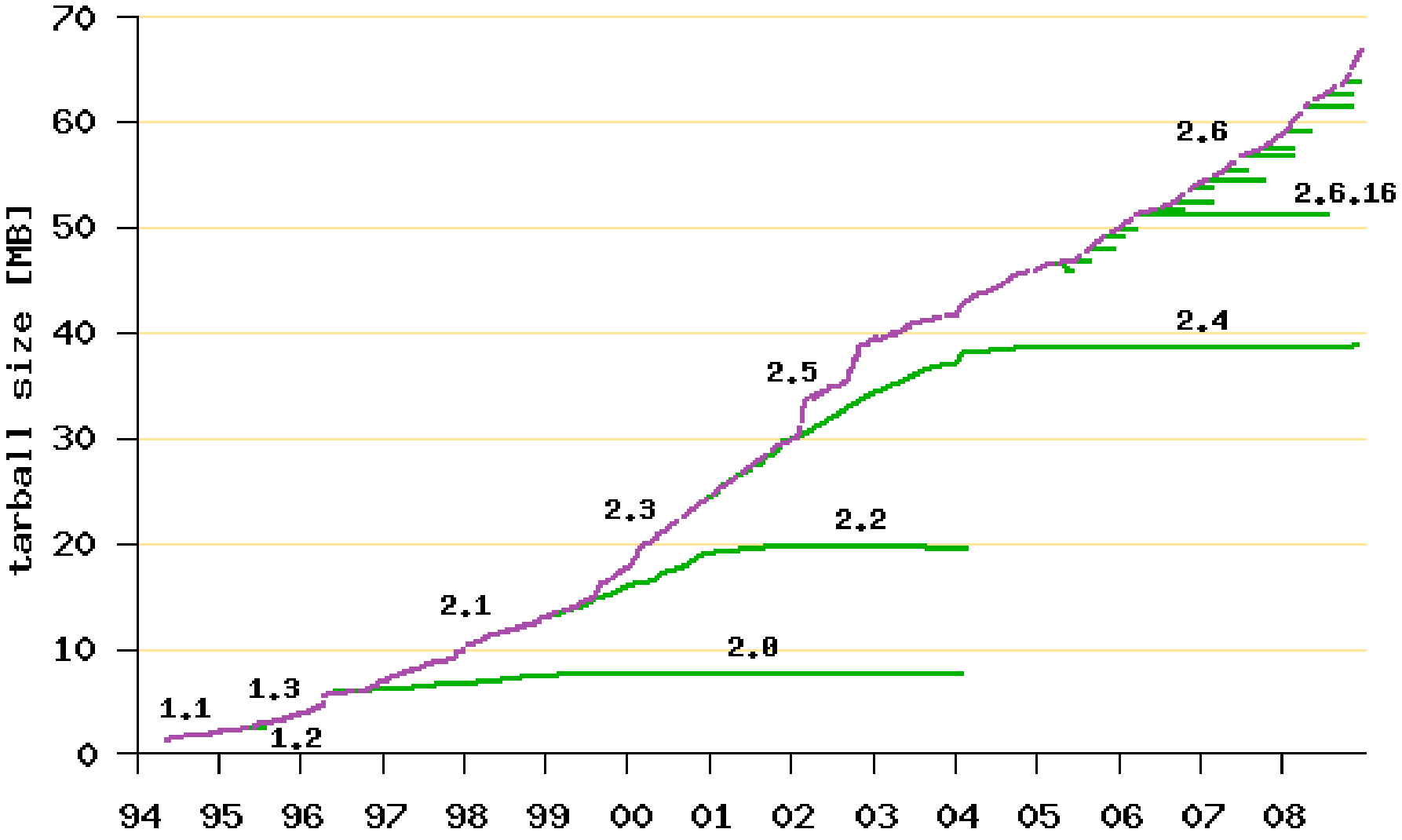
Perpetual Development Lifecycle



Perpetual Development Lifecycle



Linux Example



Software Growth

- Lehman:
  - A system's complexity grows with time
  - It is harder to modify a more complex system
  - Ergo rate of growth will be reduced with time (specifically inverse-square law due to having to consider all possible interactions)
- Godfrey and Tu:
  - Linux (and other systems) is growing at a super-linear rate

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

Continuous Deployment Variant

- New software released in timescales of minutes, not days or weeks
- Each developer immediately deploys whatever he works on
- Requires strong framework to control releases and roll them back if needed
- Makes the whole notion of a “version” meaningless
- Popular mainly in web-based companies and applications

Implications for Development

- No fixed goal that has to be reached
- Goal is to continually improve the system and maintain is 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 is company becomes insolvent

Implications for Architecture

- Can't decide on architecture based on analysis of all the requirement
- 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

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

Denning, Gunderson, & Hayes-Roth, CACM 12/2008

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