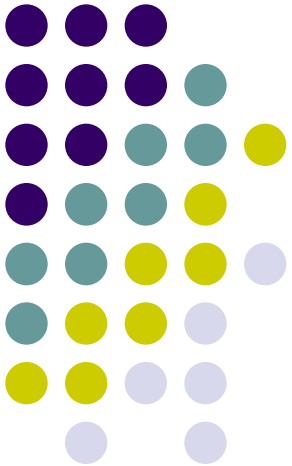
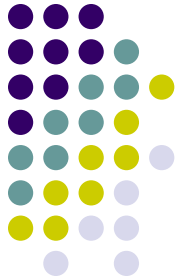
# Software Architecture

Principles of Package Design

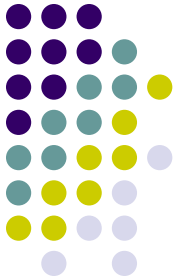


# Packages Introduction



- What is a package?
  - Classes are not sufficient to group code
  - Some classes collaborate dependencies
  - Some don't know each other
- Grouping related classes together seems natural
  - But how?
  - Dependencies between packages

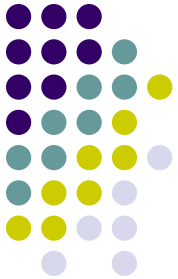
# Six Design Principles



- Principles of package cohesion
  - The Reuse-Release Equivalence Principle
  - The Common-Reuse Principle
  - The Common-Closure Principle

  *Govern the partitioning of classes into packages*

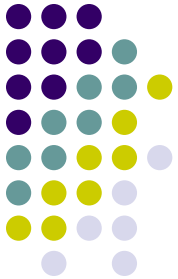
# Six Design Principles



- Principles of package coupling
  - The Acyclic-Dependencies Principle
  - The Stable-Dependencies Principle
  - The Stable-Abstractions Principle

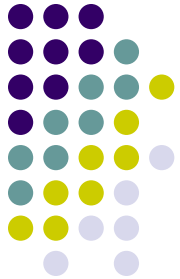
  Govern the interrelationships between packages.

# The Reuse-Release Equivalence Principle



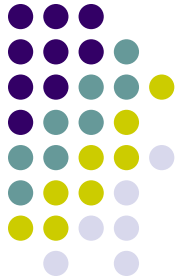
- REP:
  - the granule of reuse is the granule of release.

# The Common-Reuse Principle



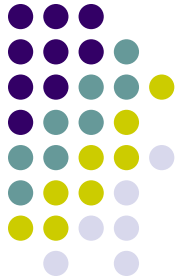
- CRP:
  - The classes in a package are reused together. If you reuse one of the classes in a package, you reuse them all.
- In such a package, classes have lots of dependencies on each other.
- Classes are not tightly bound to each other with class relationships should not be in the same package.

# The Common-Closure Principle



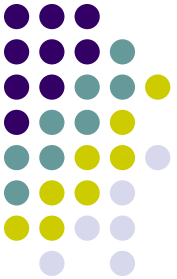
- CCP:
  - The classes in a package should be closed together against the same kinds of changes. A change that affects a package affects all the classes in that package and no other packages.
- A package should not have multiple reasons to change.

# The Acyclic-Dependencies Principle



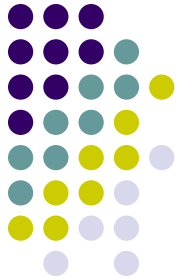
- ADP:
  - Allow no cycles in the package-dependency graph
- Morning-after syndrome
- Two solutions:
  - The weekly build
  - ADP

# The Weekly Build



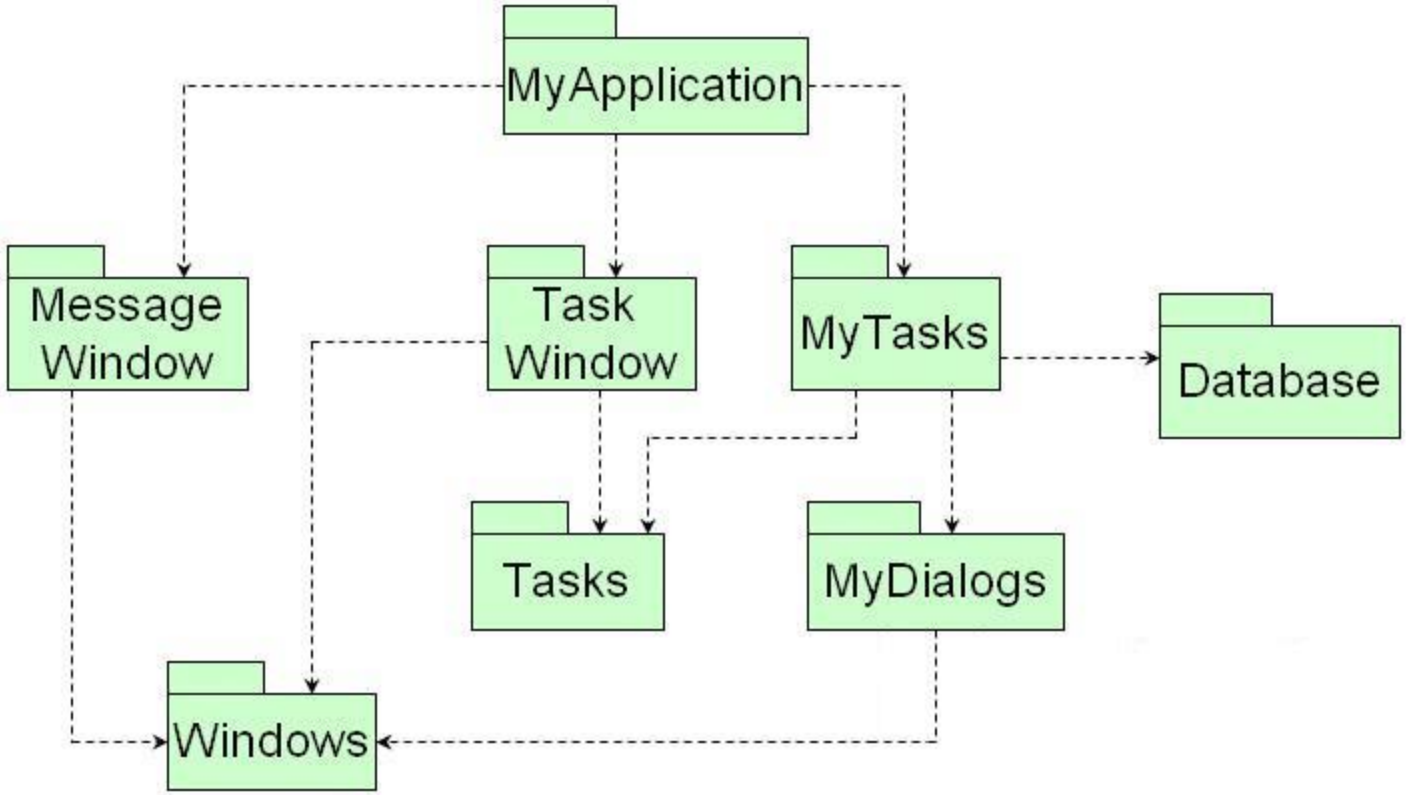
- All the developers ignore each other for the first four days of the week. They all work on private copies of the code and don’t worry about integrating with each other.
- On Friday, they integrate all their changes and build the system.
- Common in medium-sized projects

# Eliminating Dependency Cycles

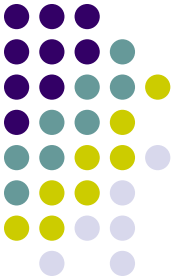


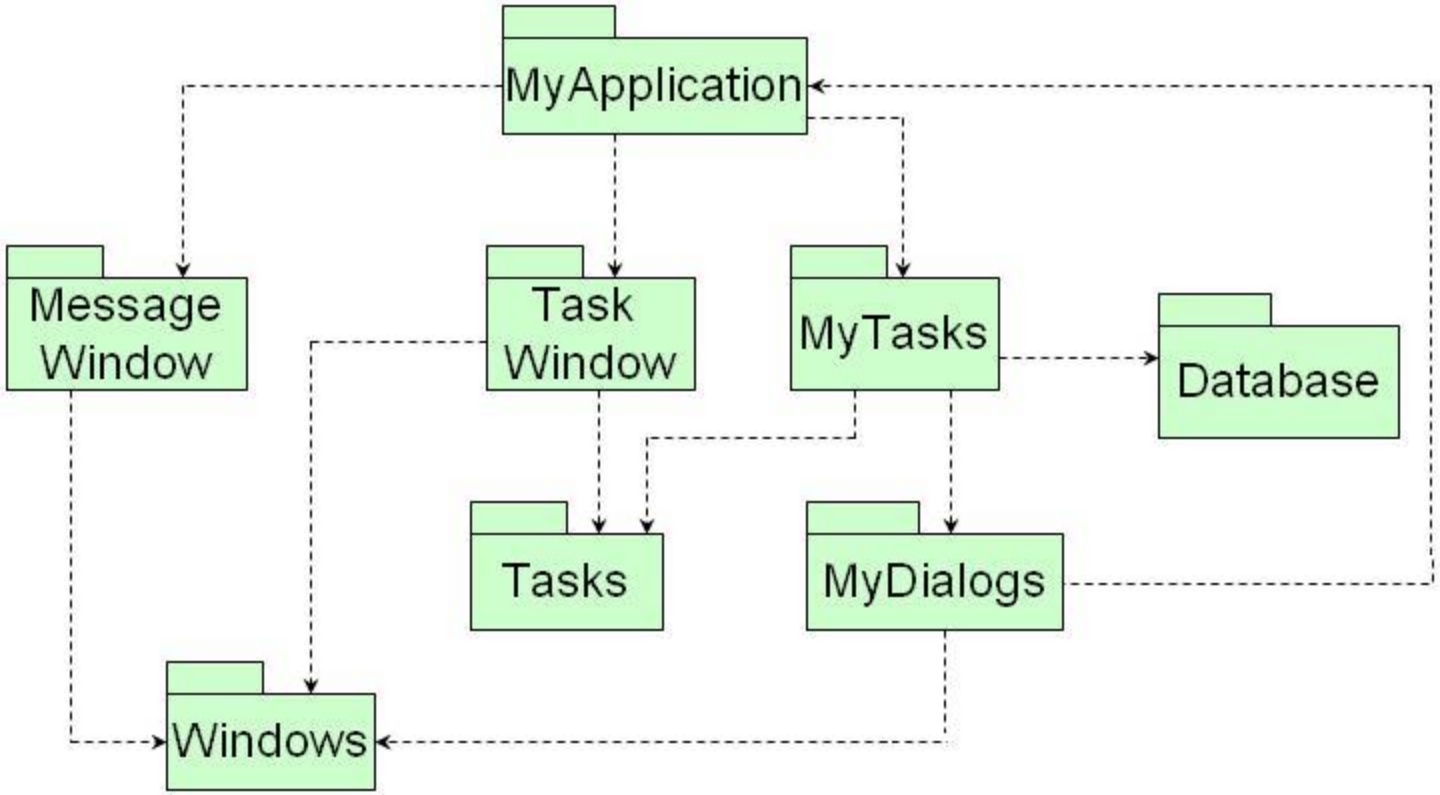
- Partition the development environment into releasable packages.
- To make it work, there can be no cycles.
- Regardless of the package at which you begin, it is impossible to follow the dependency relationship and wind up back at that package. This structure has no cycles. It is a directed acyclic graph (DAG).

# Directed Acyclic Graph (DAG)

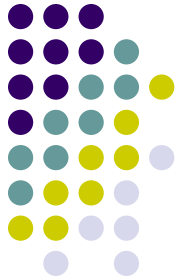


# The effect of a Cycle in the Package Dependency Graph



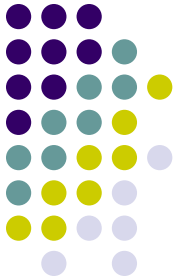


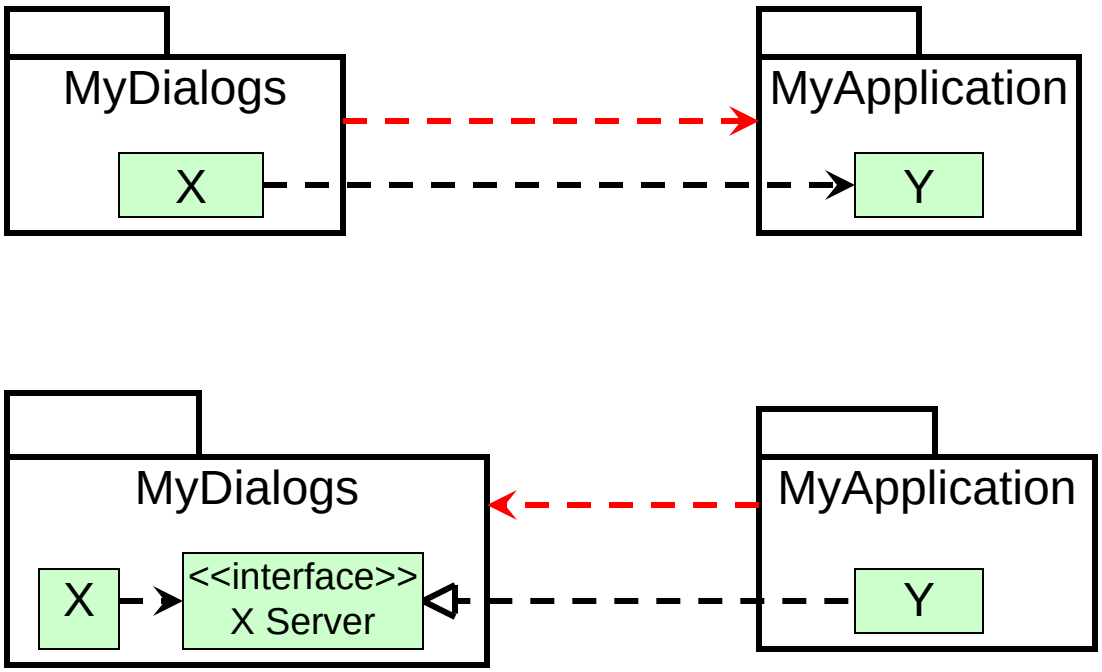
# Breaking the Cycle



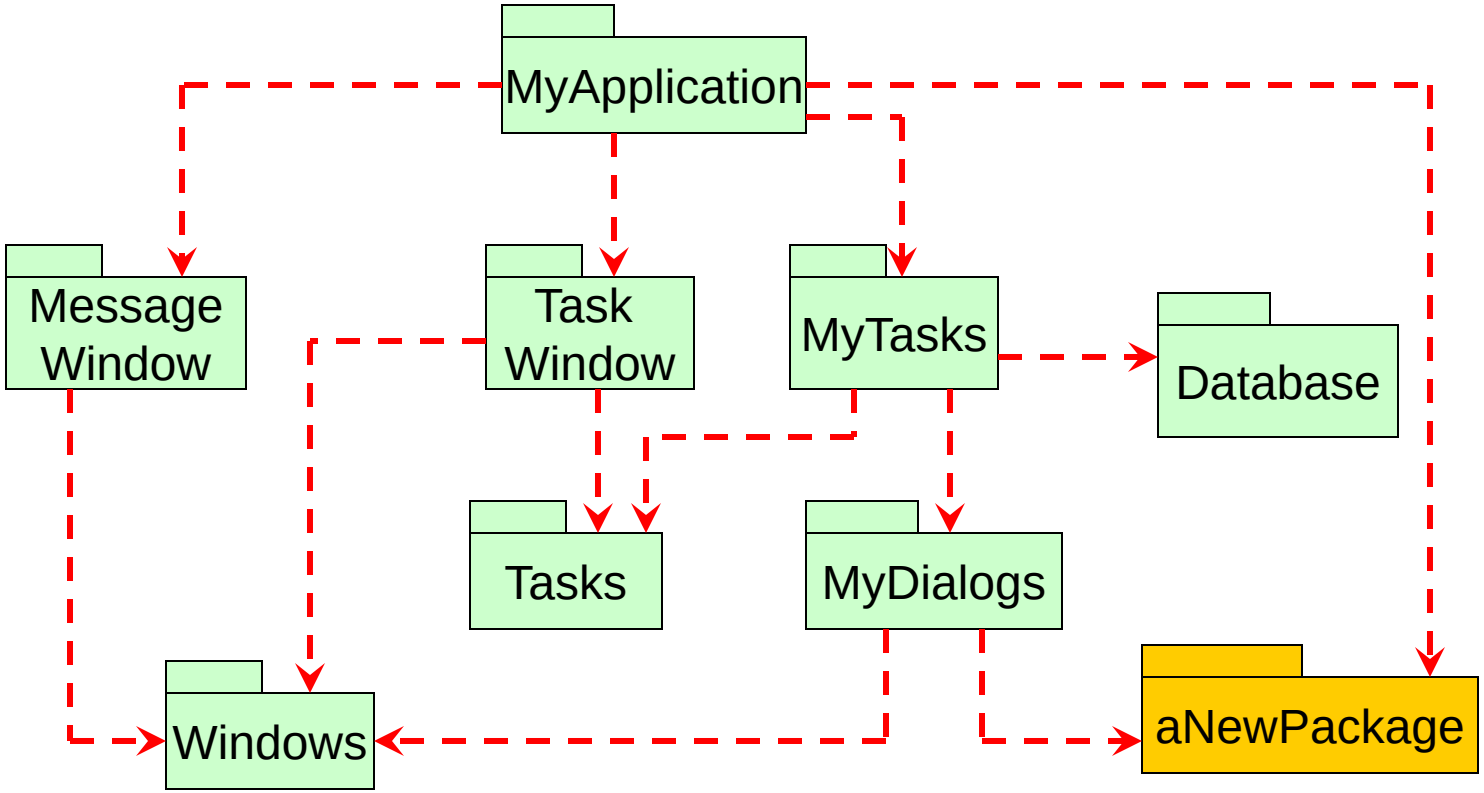
- Two mechanisms to break a cycle of packages and reinstate the dependency graph as a DAG.
  - Apply the Dependency-Inversion Principle
  - Create a new package on which both MyDialog and MyApplication depend.

# Breaking the cycle with dependency inversion

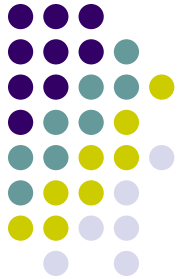




# Breaking the cycle with a new package

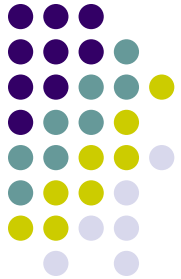


# Top-Down Design



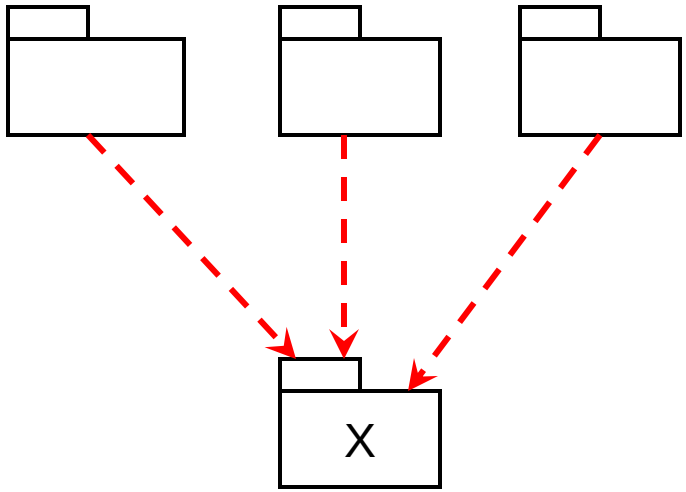
- The package structure cannot be designed from the top down.
  - Not one of the first things about the system that is designed.
  - Evolve as the system grows and changes
- Package dependency diagrams are a map to the buildability of the application.
- It will fail to design the package dependency structure before designing any classes.

# The Stable-Dependencies Principle



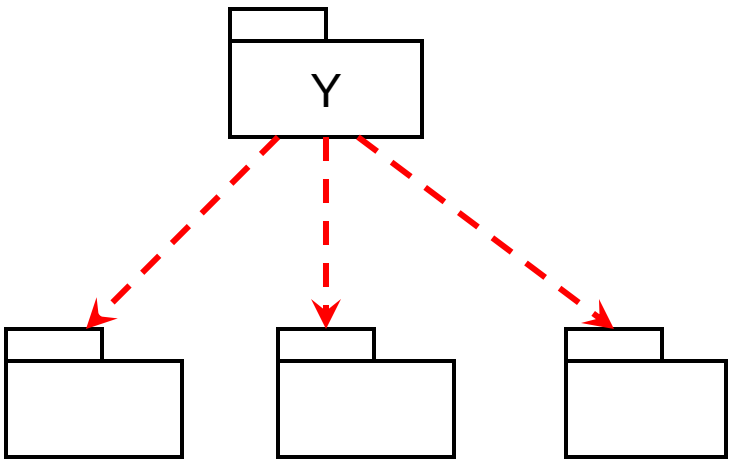
- SDP:
  - Depend in the direction of stability.
- Conforming to the SDP can ensure that modules that are intended to be easy to change are not depended on by modules that are harder to change than they are.

# Stable package




- Stable package
  - Responsible: three packages depends on x.
  - Independent: x depends on nothing.

# Instable package




- Instable package
  - Irresponsible: Y has no other packages depending on it.
  - Dependent: change may come from external sources

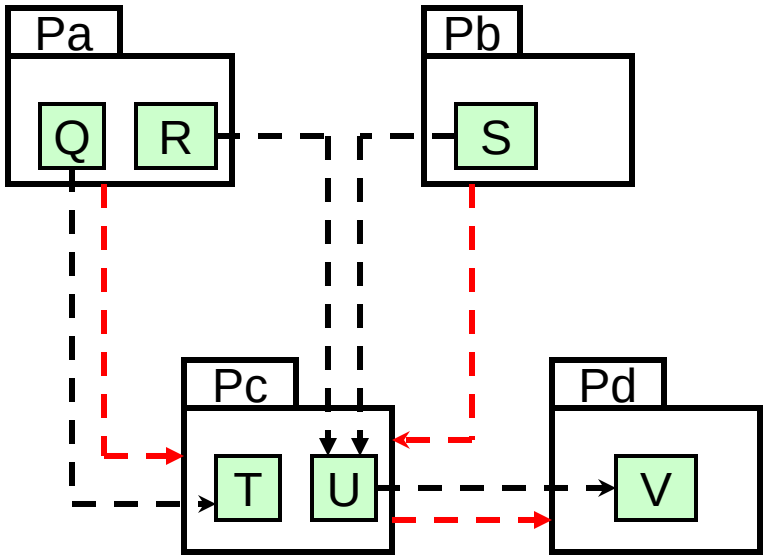
# Stability Metrics

- Stability Metrics

$$I = \frac{C_e}{C_a + C_e}$$

- $C_a$ :Afferent Couplings
  - The number of classes outside this package that depend on classes within this package
- $C_e$ :Efferent Couplings
  - The number of classes inside this package that depend on classes outside this package.
- I: Instability
  - I=0 indicates a maximally stable package
  - I=1 indicates a maximally instable package

# Stability Metrics

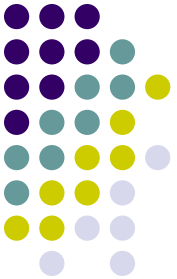



$$C_a = 3$$

$$C_e = 1$$

$$I = \frac{1}{4}$$

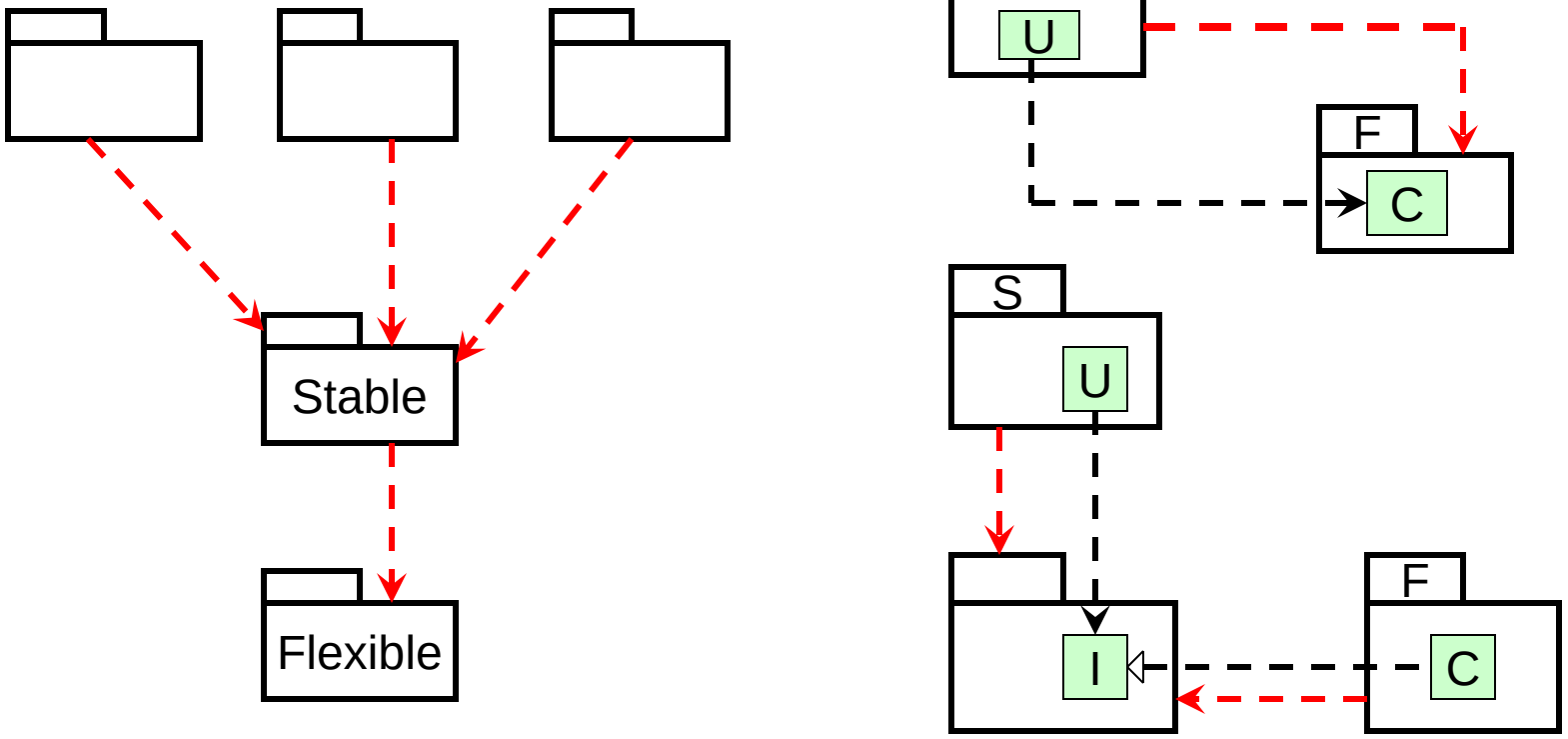
# Stability Metrics



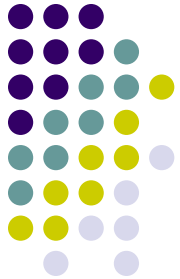
- SDP says that the *I* metrics of a package should be larger than the *I* metrics of the packages that it depend on (i.e., *I* metrics should decrease in the direction of dependency)

# Not All Package Should Be Stable

- The changeable packages are on top and depend on the stable package at the bottom.

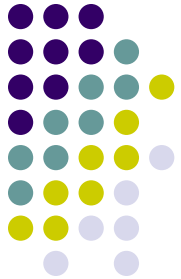


# The Stable-Abstractions Principle



- SAP principle:
  - A package should be as abstract as it is stable.
- A stable package should also be abstract so that its stability does not prevent it from being extended.
- The SAP and the SDP combined amount to the DIP for packages.

# Measuring Abstraction



- Measuring Abstraction

$$A = \frac{Na}{Nc}$$

- $N_c$ : The number of classes in the package.
- $N_a$ : The number of abstract classes in the package
- A: Abstractness