Textbook: Object-Oriented Analysis & Design

Chapter 1
Software Complexity

H.S. Sarjoughian

CSE 460: Software Analysis and Design

School of Computing, Informatics and Decision Systems Engineering
Fulton Schools of Engineering

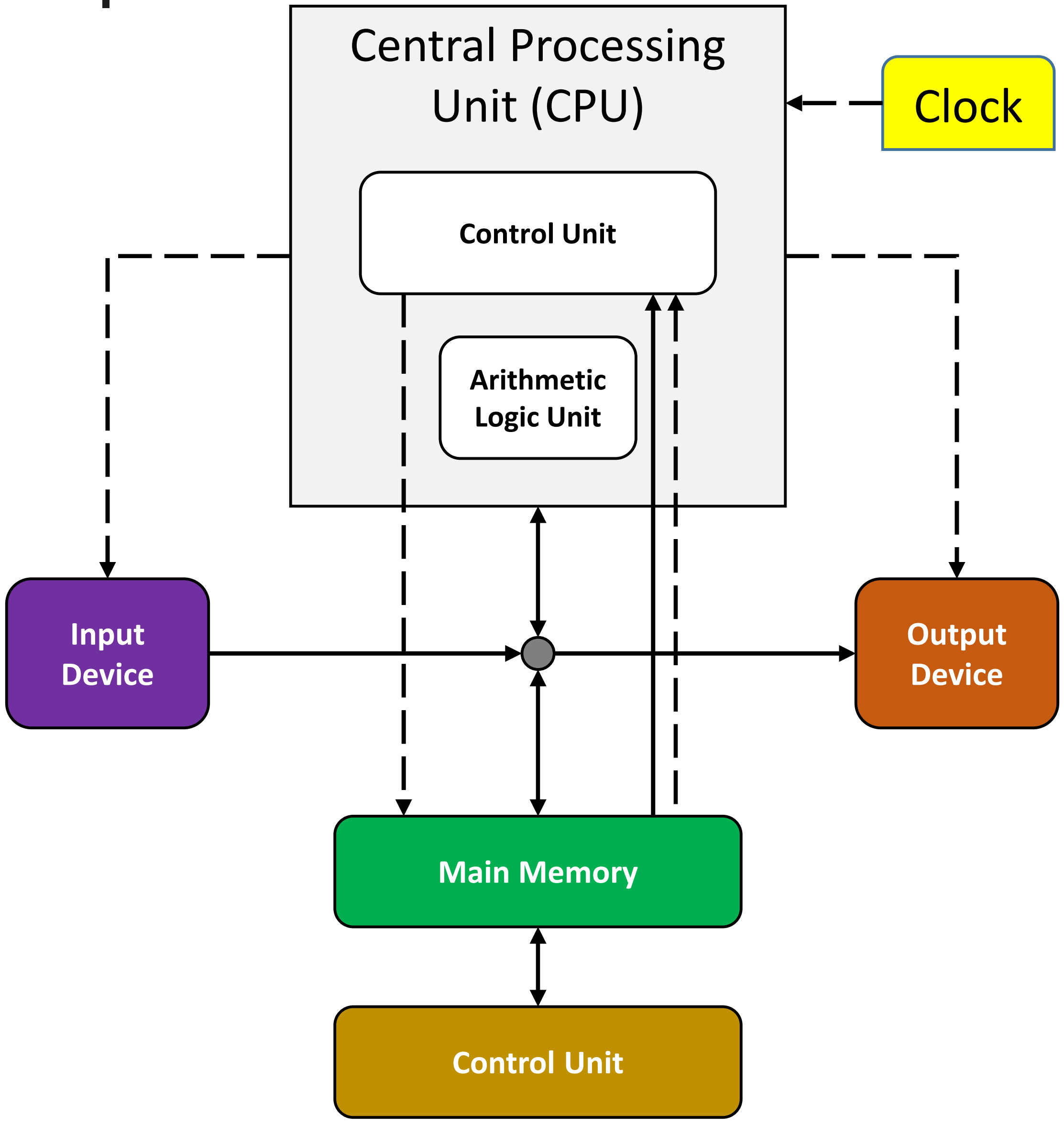
Arizona State University, Tempe, AZ, USA

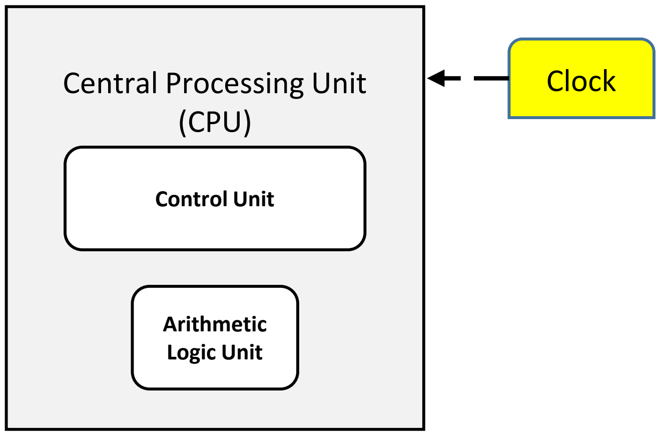
Discrete Systems Sources of Complexity

- A large software
  - has many (tens) of thousands of variables
  - can be subjected to many input stimuli (some unpredictable) in a variety of combinations
  - has a very large state-space – many possible combinations of state values for a given finite set of state variables
- Representations of structure and behavior of a software system are generally not unique

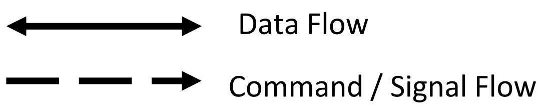
*continuous systems have (**hidden**) order – a property that we strive toward in analysis and design of discrete software systems…*

A Computer System Abstraction

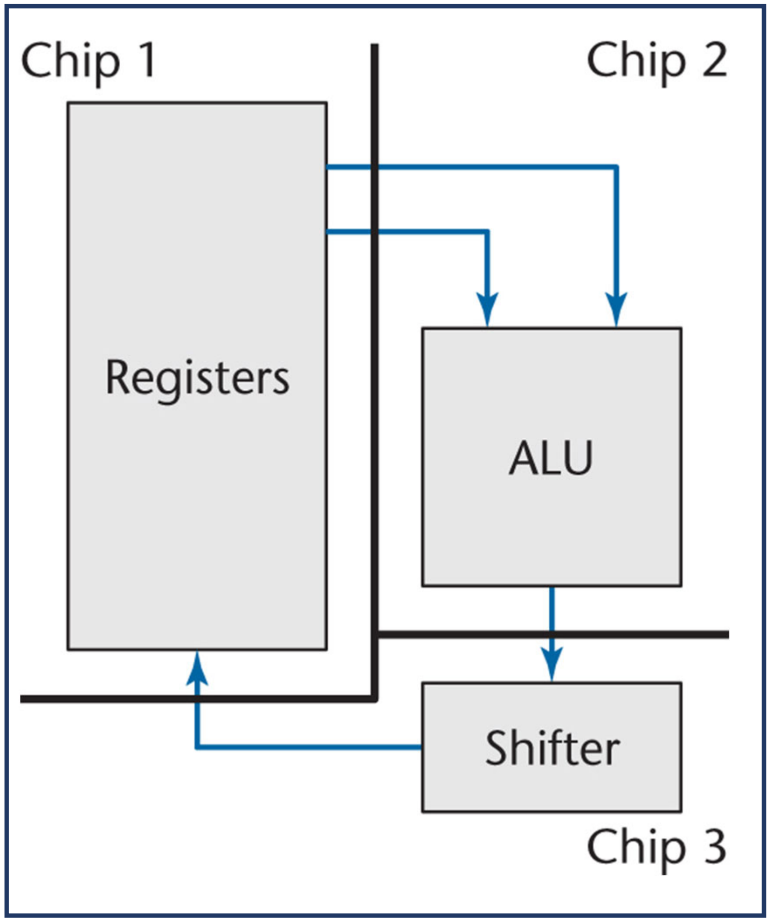




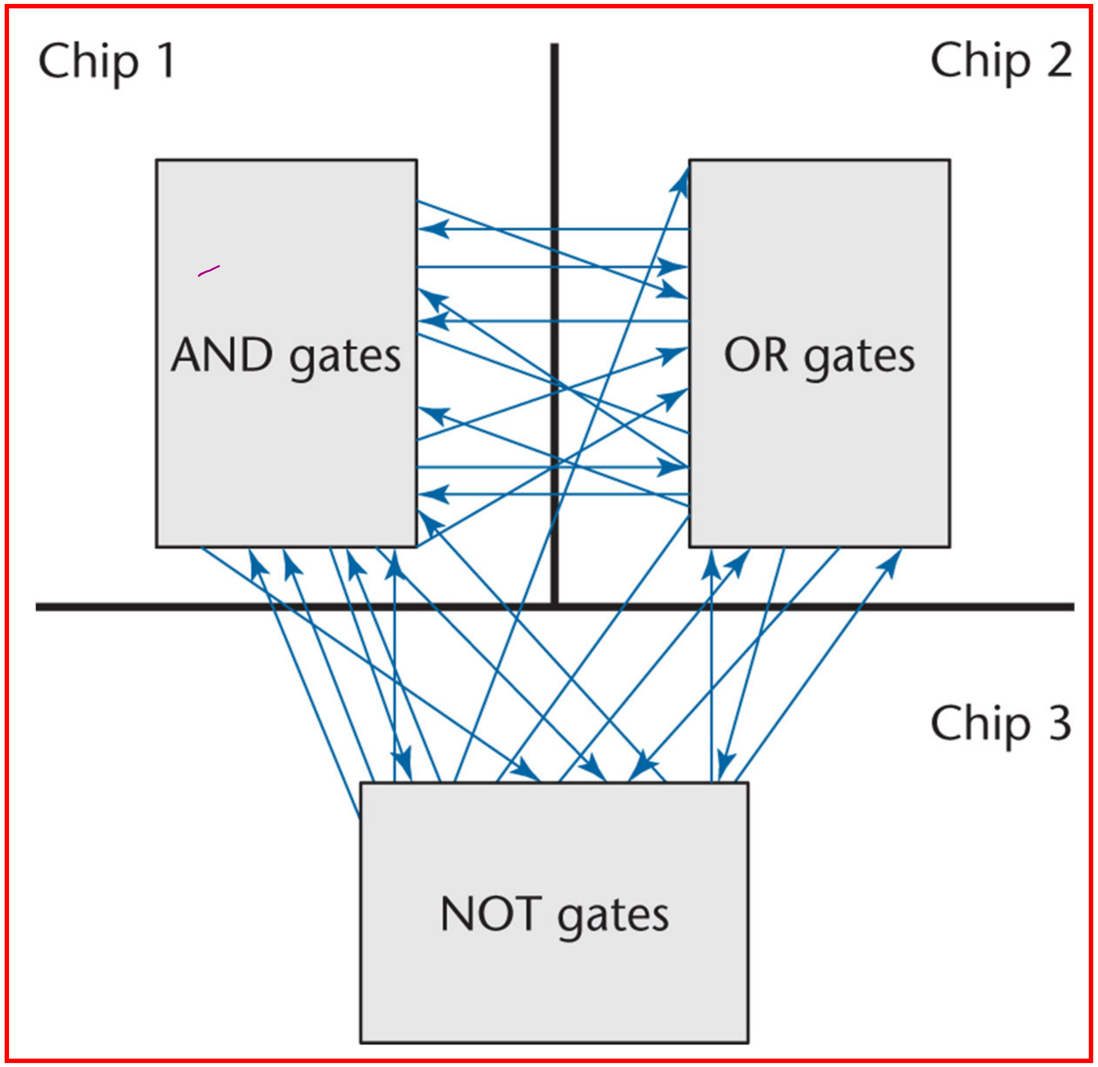
- Basic component for a basic computer
- Relationships for a computer system



Designs for a Basic Computer



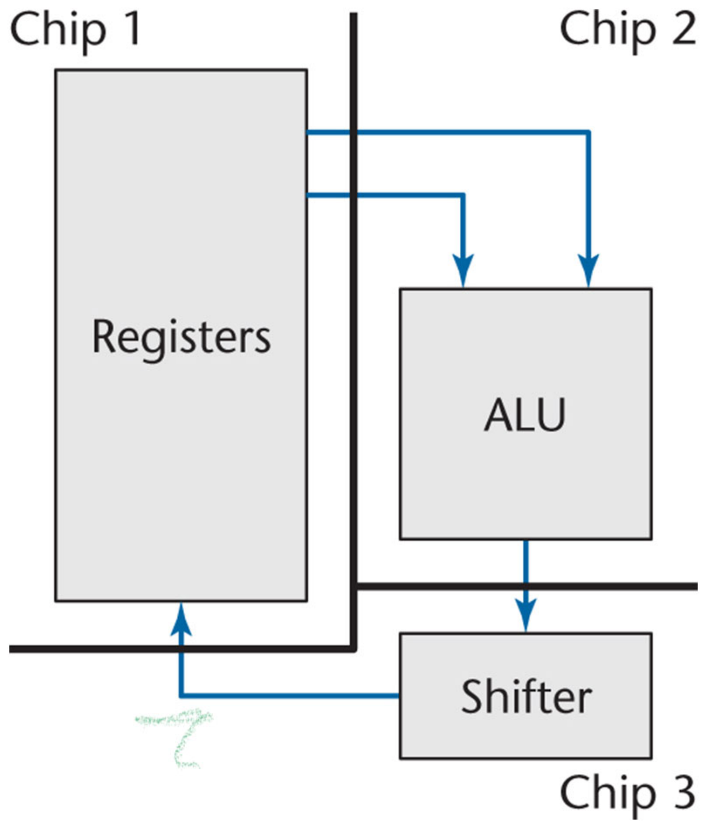
First Design



Second Design

Composite/Structured Design

- A method for breaking up a product into modules to achieve
  - Maximal interaction within a module, and
  - Minimal interaction between modules
- Module cohesion
  - Degree of interactions within a module
- Module coupling
  - Degree of interactions between modules



Designs for a Basic Computer (cont.)

- The two designs are functionally equivalent
  - The second design is
    - Hard to understand
    - Hard to locate faults
    - Difficult to extend or enhance
    - Cannot be reused in another product
- Modules must be like the first design
  - Maximal relationships within modules, and
  - Minimal relationships between modules

Unrestrained Complexity

- *“The more complex the system, the more open it is to total breakdown”* Peter, Laurence.

- Failing to master complexity generally results in a software which:

  - costs far beyond its original estimate
  - is late
  - does not satisfy its stated requirements

$\Rightarrow$ important advances in software engineering have been achieved due to understanding software complexity and what it entails!

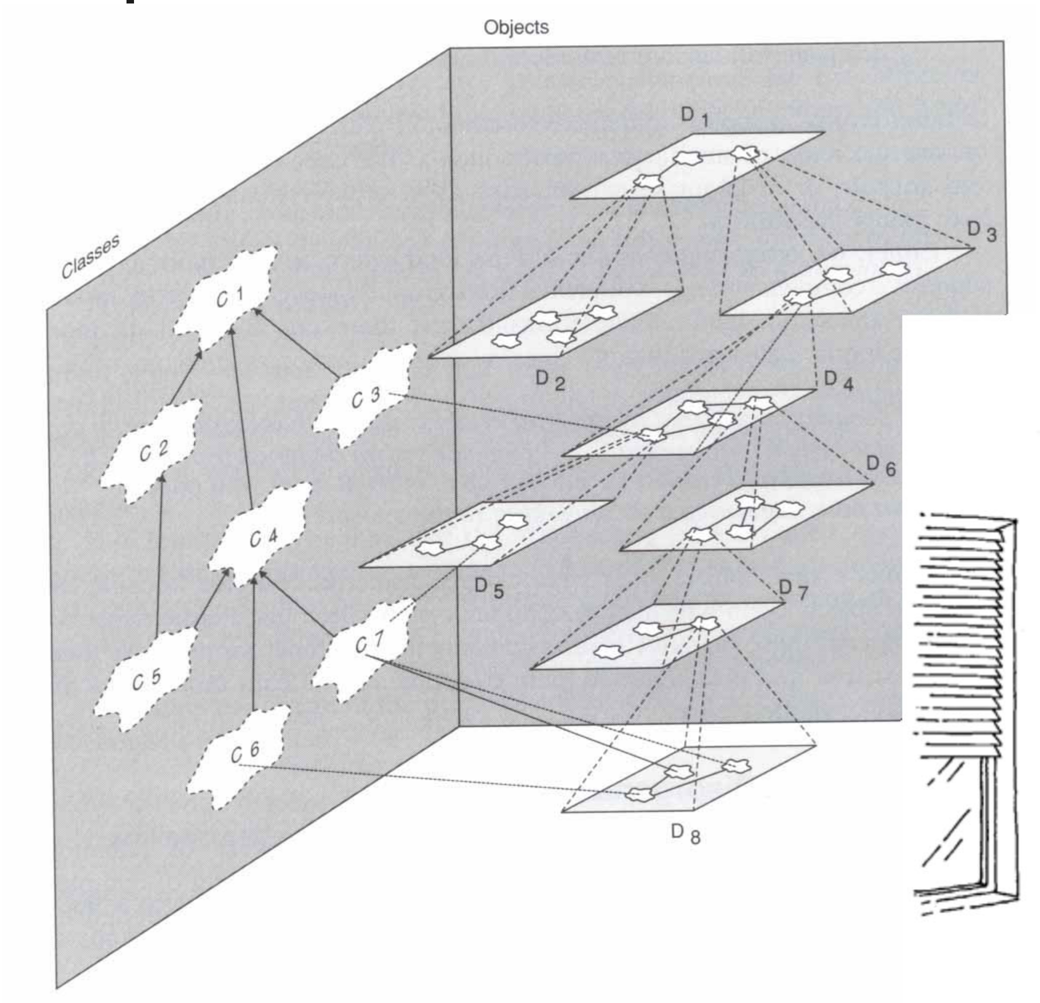
Six Attributes of a Complex System

1. Frequently, complexity takes the form of a **hierarchy**, whereby a complex system is composed of interrelated subsystems that have in turn their own subsystems, and so on, until some lowest level of elementary components is reached
2. The **choice of what components** in a system are primitive is relatively arbitrary and is largely up to the discretion of the observer of the system
3. **Intra-component linkages** are generally stronger than **inter-component linkages**. This fact has the effect of separating the high-frequency dynamics of the components – involving the internal structure of the components – from low-frequency dynamics – involving interaction among components

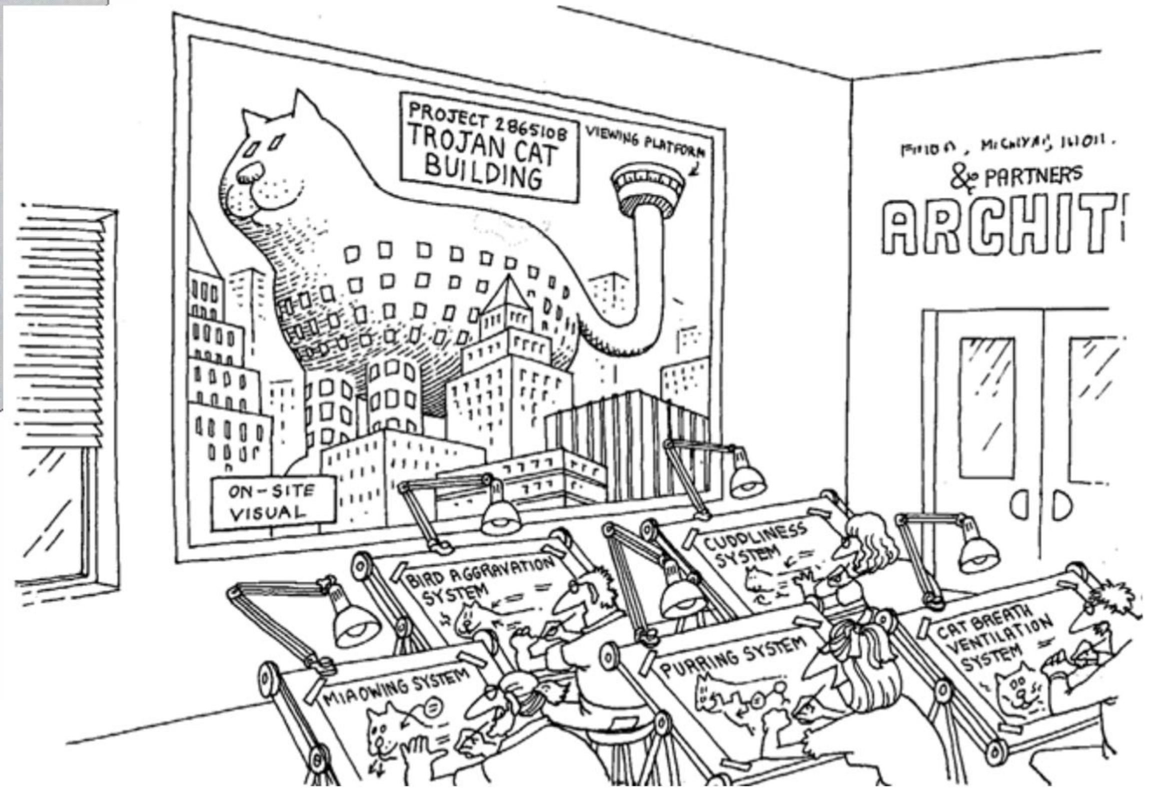
Six Attributes of a Complex System

4. Hierarchic systems are usually composed of only **a few different kinds of subsystems** in various combinations and arrangements (choice!!)
5. A complex system that works is invariably found to have evolved from **a simple system** that worked …. A complex system designed from scratch never works and cannot be "patched up" to make it work. One has to start over, beginning with a simple working system
6. **Heterogeneity** has a direct relationship with a system's level of complexity

Canonical Form of a Complex Software System



- MyASU / Canvas
- Mars Curiosity System
- Smart manufacturing



OOAD, 1994, 2007

Complexity and Human Cognition

Software engineers, like all people, have limitations in handling and dealing with complexity. To handle complexity, we can employ

- **decomposition**, software engineers can focus and comprehend a relatively few parts of a complex system,
- **abstraction**, software engineers can limit their comprehension by focusing on some information while ignoring a great deal more, and
- **hierarchy**, software engineers can view and reason about patterns of structure and behavior for a set of entities.

Objects, Classes, Models, and Software Engineering

- Objects and classes are the most elementary elements of object-based software systems.
- Models are key artifacts capturing a great variety of relationships (structural and behavioral) describing static and dynamic aspects of a software system
- Generally, a collection of models are required to adequately specify multiple views of a software system
- There does not exist an automated process that can replace software engineers – at least not yet!
- Nevertheless, there exist processes and techniques that can bring a significant degree of order and predictability in software development

Software Products as Compositions of Modules

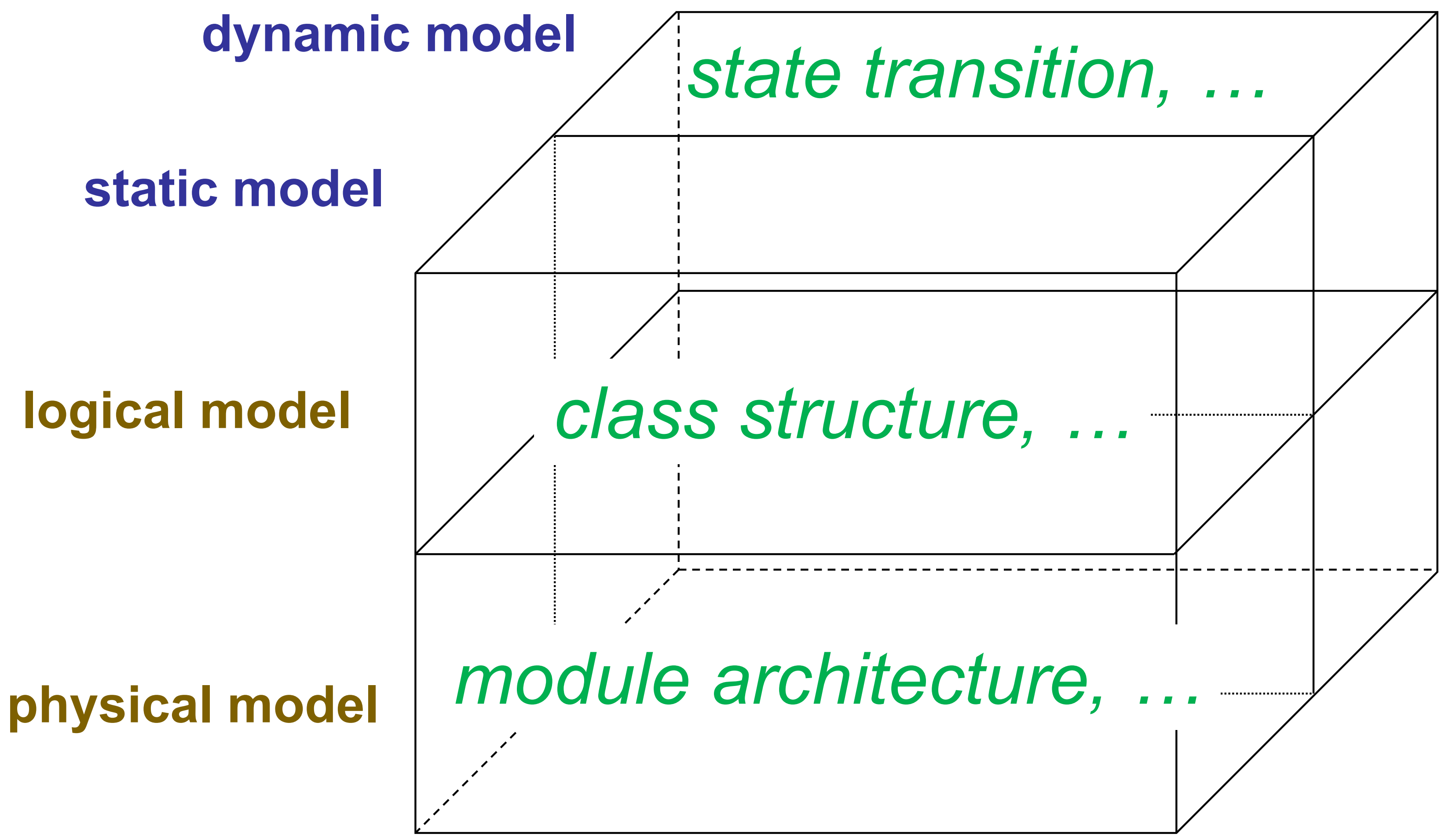
- It is impractical to build software product as a single monolithic block of code.
  - coding, testing, debugging, maintenance, reuse, …
- Dividing the product into parts is necessary.

A first definition for module: **"A set of one or more contiguous program statements having a name by which other parts of the system can invoke it"**, Stevens, Myers, Constantine, 1974

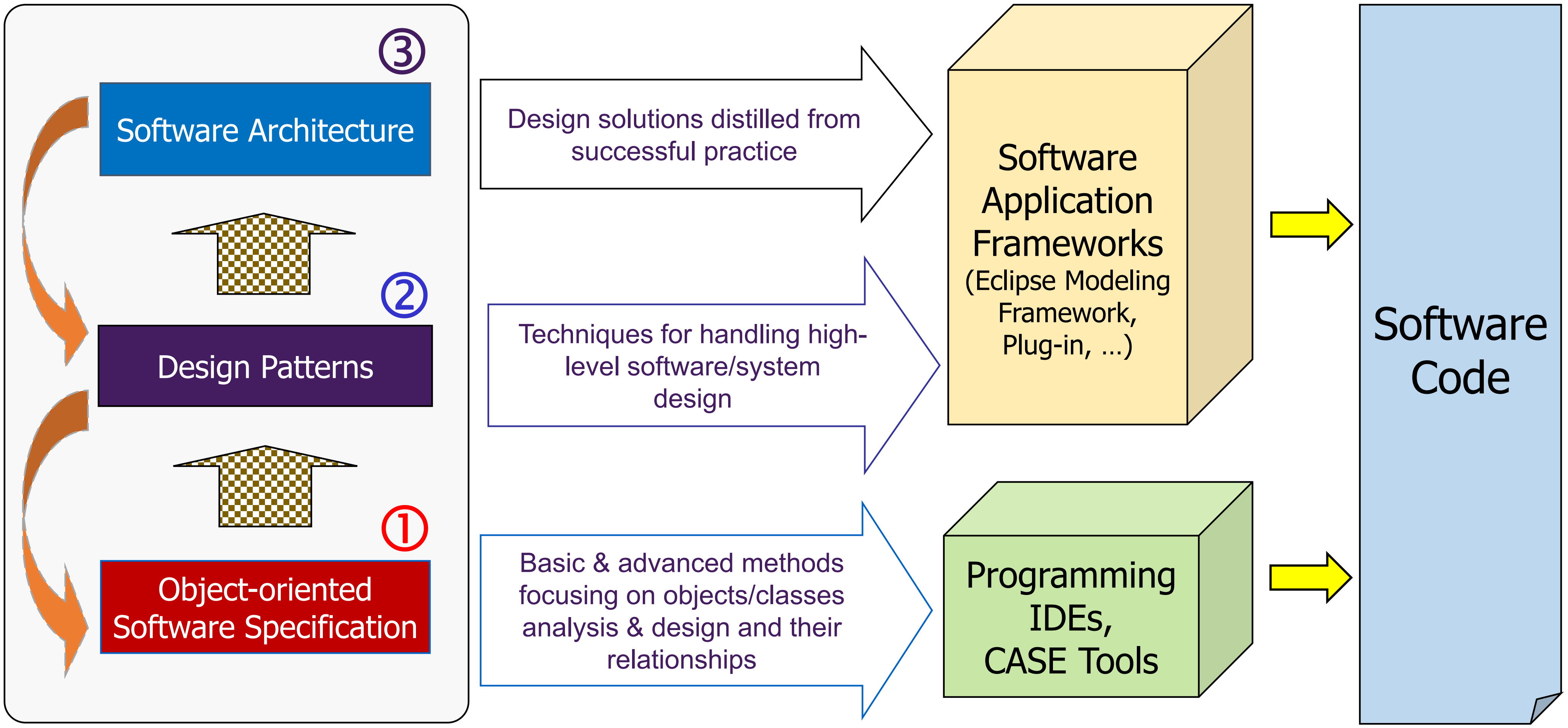
Elements of Software Analysis and Design Methods

- **Notation:** the language for expressing a family of models
- **Process:** The activities leading to a principled construction of models
- **Tools:** The artifacts that eliminate the tedium of model building and enforce rules about the models themselves, so that errors and inconsistencies can be exposed and controlled

Models and Views of Object-Oriented Development



Approach to Component-based Software Engineering


③
Software Architecture
②
Design Patterns
①
Object-oriented Software Specification
Design solutions distilled from successful practice
Techniques for handling high-level software/system design
Basic & advanced methods focusing on objects/classes analysis & design and their relationships
Software Application Frameworks (Eclipse Modeling Framework, Plug-in, …)
Programming IDEs, CASE Tools
Software Code

Engineering with Science and Art

Software engineering is a **mixture** of **science** and **art**

- Art underlies creativity which generally is key for understanding complex systems
- Science provides a rigorous foundation necessary for engineered systems
- The mixture of science and art often produces a great many alternative solutions to a given informal problem statement

Interoperability, Performance, & Reuse (Cont.)

- **Performance:**
  - computational
    - local to each component (hardware and software)
  - communication
    - among multiple components (across intranet, internet, …)
  - both computational and communication efficiency issues affect performance

- **Reuse:**
  - Component/framework reuse
    - components (e.g., databases)
    - frameworks (e.g., Eclipse Modeling Framework)

Interoperability, Performance, & Reuse

- Complexity refers to the difficulty associated with comprehending multiplicity of interrelated entities. Elements such as composition, heterogeneity, and size are some of the key contributors to software analysis and design complexity.

- **Interoperability:**
  - computer-based and non-computer-based subsystems
    - interoperation among many alternative software and hardware components
  - hierarchical construction
    - well-defined functionality & interfacing
  - logical vs. real-time interoperability
    - design, implementation, testing, and operation

Summary

- The objective of a software development team is to engineer the illusion of simplicity
- Software is inherently complex – the complexity of software systems (especially commercial grade) generally exceeds the individual human intellectual capacity
- Complexity often takes the form of a hierarchy
- Decomposition, abstraction, and hierarchy are key in handling complexity

Summary (cont.)

- Complex systems can be viewed and comprehended by focusing upon objects (things or processes) and their relationships

- Object-oriented concepts and methods are essential in the development of complex systems – concept development, analysis, design, implementation, ….

References

- *Booch, G., et al., Object-Oriented Analysis and Design with Applications, 3rd Edition, Benjamin Cummings, 2007. (OOAD)*
- *Mathews, van Holde, Ahern, Biochemistry, 3rd Ed., Pearson Benjamin Cummings, 2000. (MvHA)*
- *Peter,L. The Peter Pyramid Or, Will We Ever Get The Point, New York, NY: William Morrow, 1986.*
- *Dubberly, Hugh, Models of Models, Interactions Magazine, Vol. XVI.3, 54-60, June, 2009*
- *Schach, S., Object-Oriented and Classical Software Engineering, 8th Edition, 2010*

Fundamental Concerns about Software Engineering

Software can be built based on a universal theory

- True
- False
- Maybe

Engineering software is the same as engineering houses and roads

- True
- False
- Maybe

Software analysis and design models are like mathematical models

- True
- False
- Maybe

An Example of Complex Natural Systems

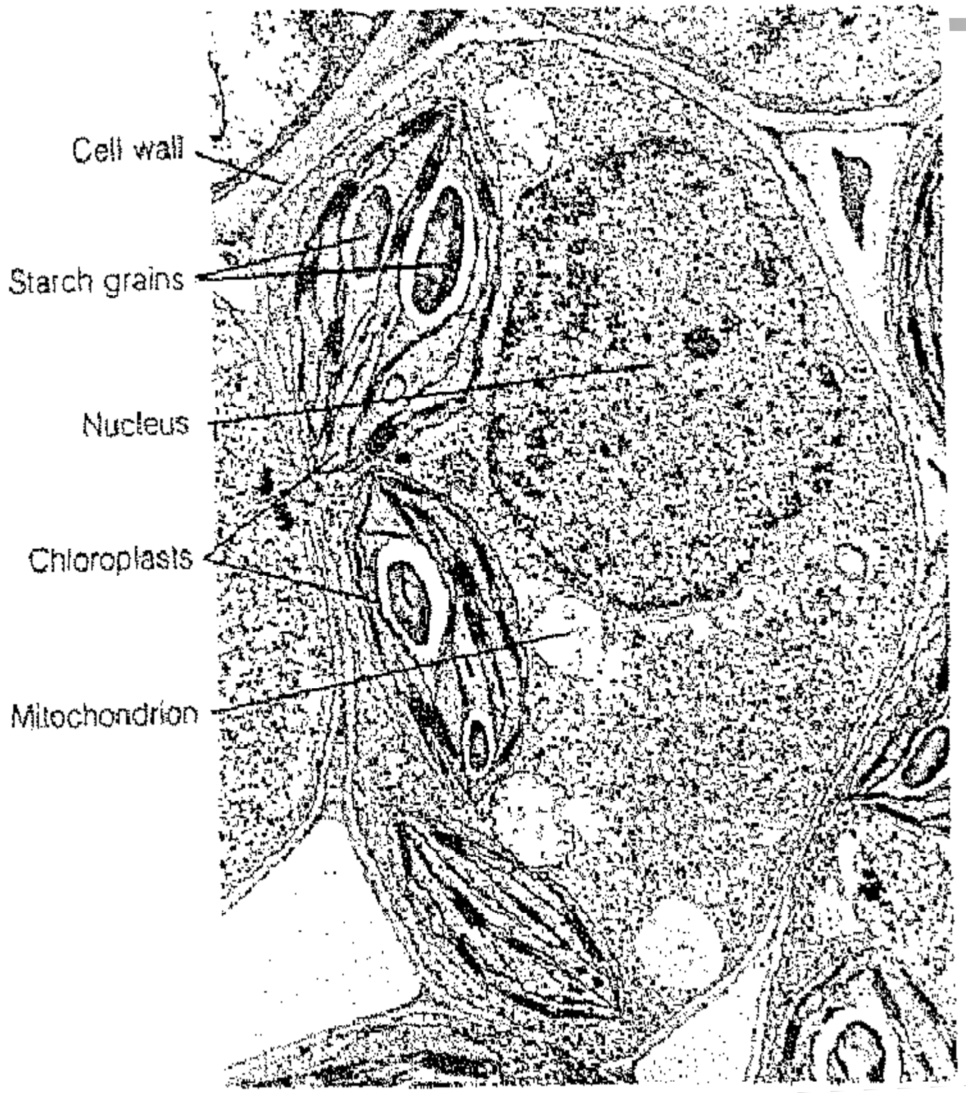
Structure of plants: Complex multi-cellular organisms

Cooperative behavior in various plants: photosynthesis and transpiration

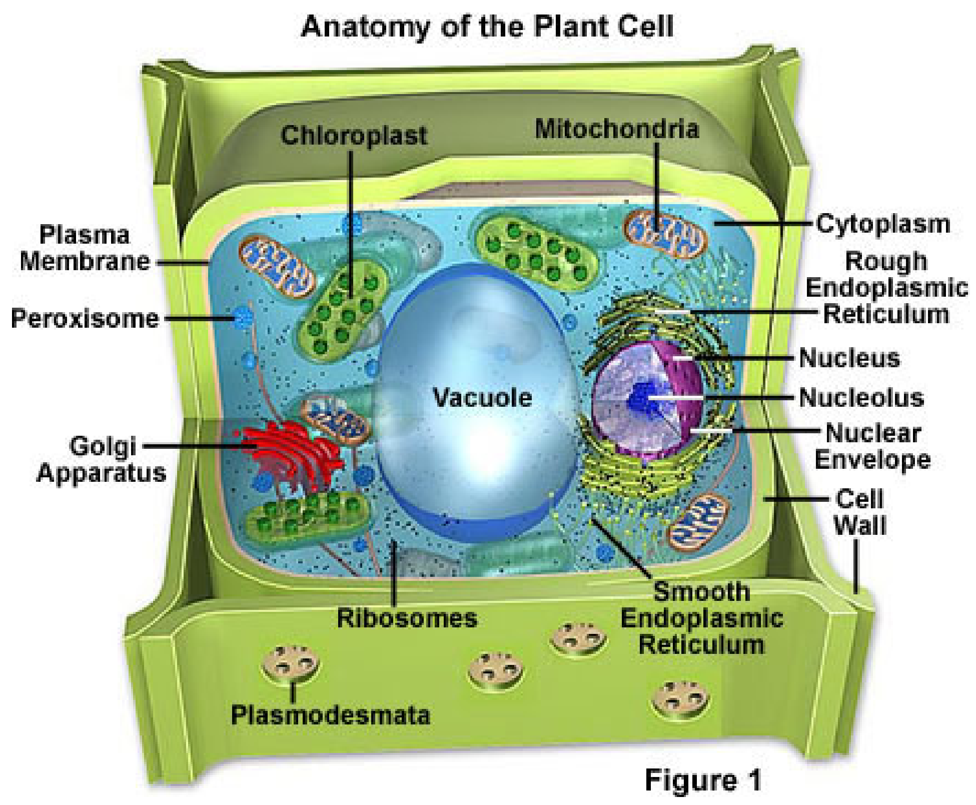
Any plant’s **structure** consists of some main **sub-structures**

- Root structure consists of branch roots, root hairs, and root apex
- Leaf structure consists of epidermis, mesophyll, and vascular tissue
- …

Plant Cell Structure



Source: MvHA



Source: http://micro.magnet.fsu.edu/cells/plantcell.html

[Leaf]()

[Java Collection API]()

An Example of Complex Natural Systems (Cont.)

All plant structures, in turn, are made up of cells where each consists of nucleus, …

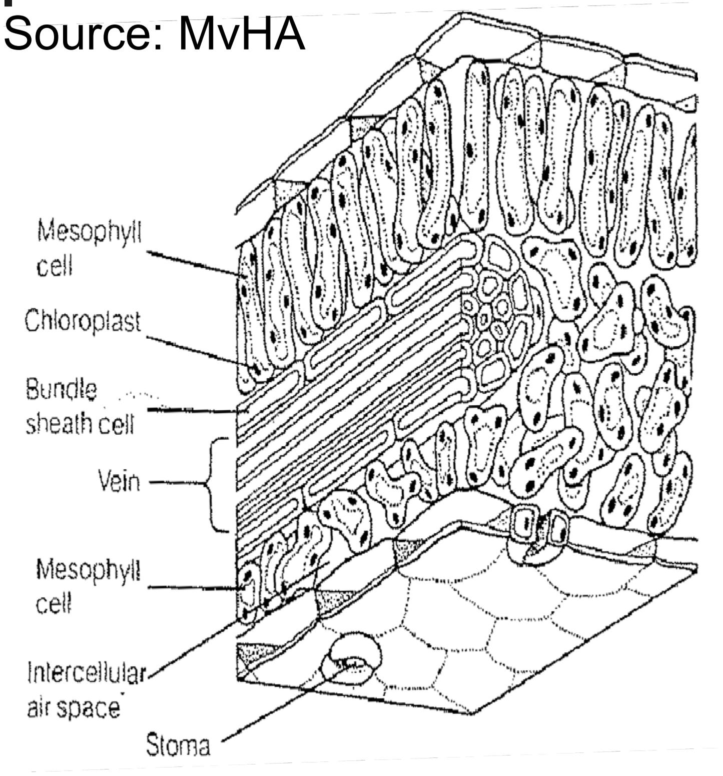
➔ The parts of a plant form a hierarchy
➔ Each part of the hierarchy has its own complexity
➔ There exist well-defined boundaries between levels of hierarchy

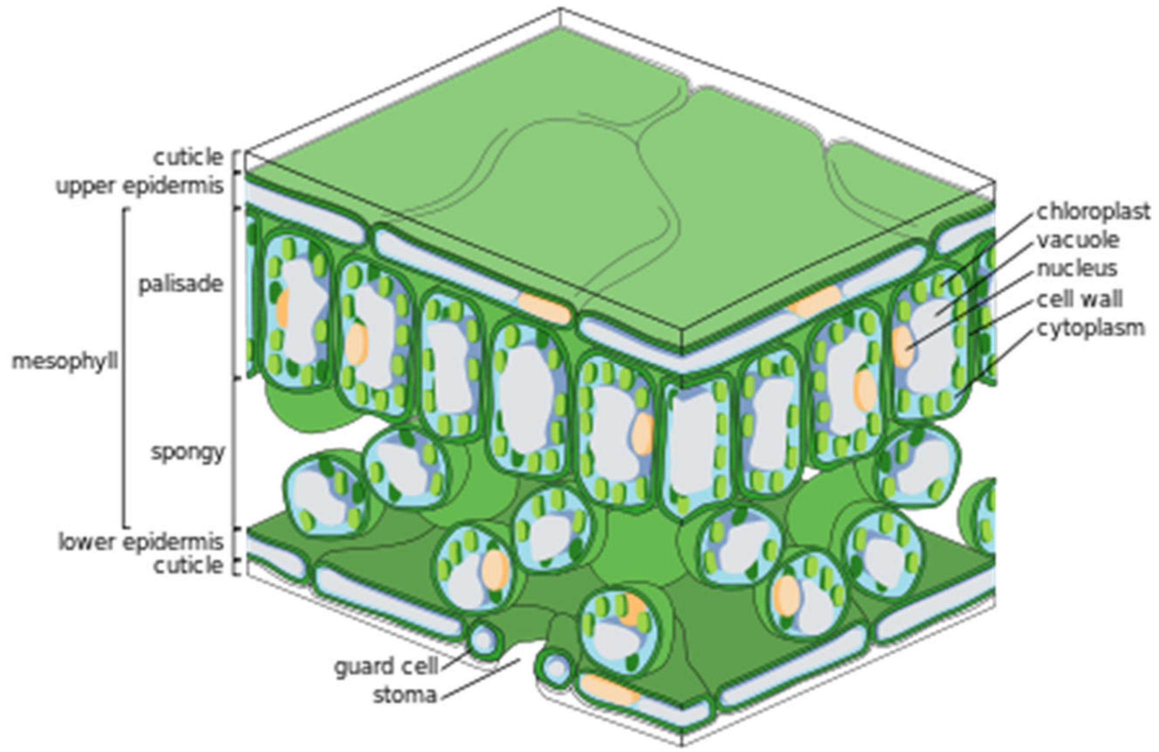
Individual sub-structures are not solely responsible for the whole, higher-level, behavior and processes such as photosynthesis. Indeed, the whole behavior is greater than the sum of its parts!

This kind of behavioral complexity is referred to as **emergent behavior**

Structure of a Leaf

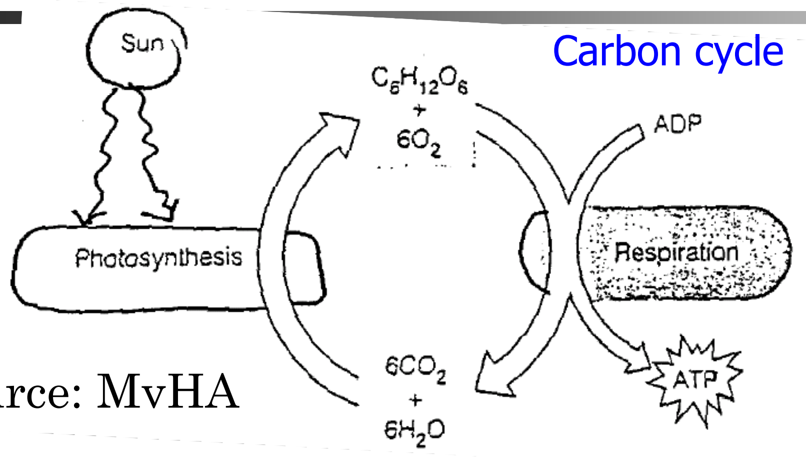
Source: MvHA



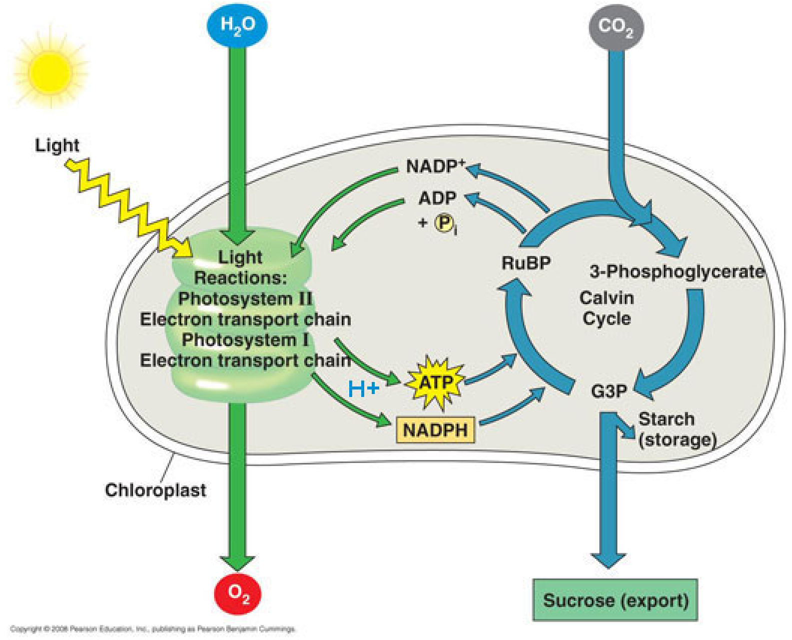


- The figure shows structural differences in mesophyll and bundle sheath cells associated with their different uses in C_3 and C_4 plants. Cells that carry out the Calvin cycle (C_3 cycle) are shown in dark green; those that carry out the C_4 cycle are shown in light green. The stomata (plural of stoma) are orifices through which the leaf exchanges gases and water vapor with its surrounding.
- C_3 plants. In C_3 plants, which have no C_4 cycle, Calvin cycle photosynthesis occurs mostly in the mesophyll cells.

Sub-processes of Photosynthesis



Source: MvHA

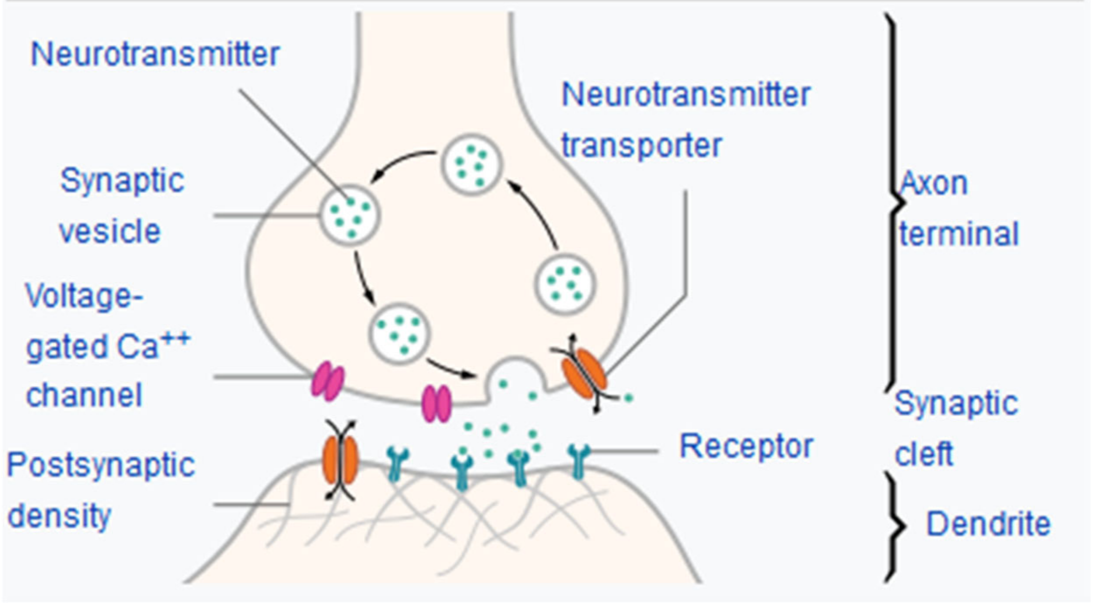

H_2O
CO_2
Light
$NADP^+$
ADP + P_i
RuBP
3-Phosphoglycerate
Light Reactions: Photosystem II Electron transport chain Photosystem I Electron transport chain
Calvin Cycle
H+
ATP
NADPH
G3P
Starch (storage)
Chloroplast
O_2
Sucrose (export)
Copyright © 2008 Pearson Education, Inc., publishing as Pearson Benjamin Cummings.

Photosynthesis

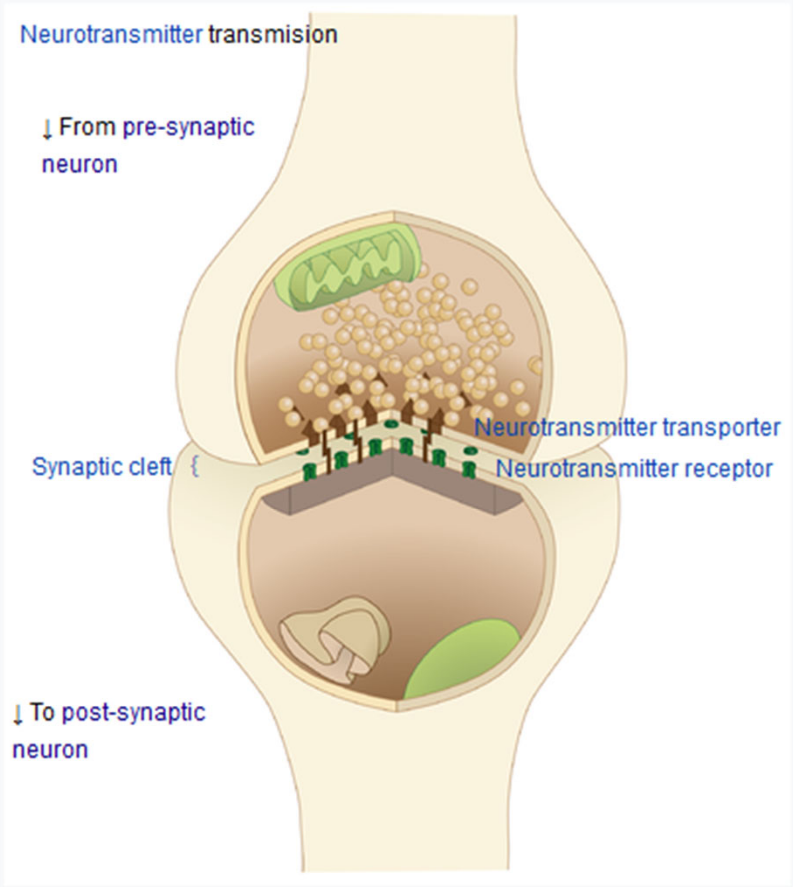
- The overall process of photosynthesis is divided into light reactions and dark reactions.
- The light reactions, which require visible light as an energy source, produce reducing power in the form of NADPH, ATP, and O.
- The NADPH and ATP drive the so-called dark reactions, which occur in both the presence and the absence of light and fix CO into carbohydrates.

Brain Synapse Structure

Structure of a typical chemical synapse



Distinguish between pre- and post- synapse[4]



Neurotransmitter transmision
↓ From pre-synaptic neuron
Synaptic cleft {
Neurotransmitter transporter
Neurotransmitter receptor
↓ To post-synaptic neuron

"The connection linking neuron to neuron is the synapse. Signal flows in one direction, from the presynaptic neuron to the postsynaptic neuron via the synapse which acts as a variable attenuator." [4] In brief, the direction of the signal flow determines the prefix for the involved synapses.[4]