EE382N.23: Embedded System Design and Modeling

Lecture 2 – Design Methodologies

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

Lecture 2: Outline

- **Design methodologies**
  - Design process
  - Abstraction levels
  - Design models
  - System-level design flow
- **Design example**
  - Object recognition in smart camera network
  - Deep and convolutional neural networks (DNNs/CNNs)
  - Distributed CNN inference in IoT edge computing

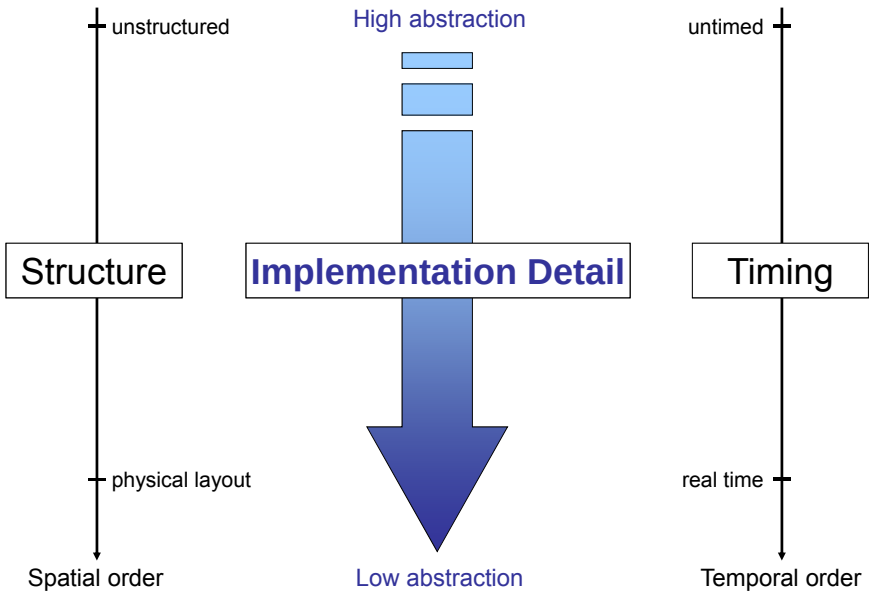
Design Process

- **Sequence of steps that transforms a set of requirements described informally into a detailed description that can be used for manufacturing**
  - Intermediate steps with transformation from a more abstract description to a more detailed one (*refinement*)
- **A designer can perform step-by-step refinement**
  - The "input" description is a *specification*
  - The final description of the design is an *implementation*

➢ **Take a model of the design at a level of abstraction and *refine* it to a lower one (level of detail ↑).**
  - Ensure that the properties at the lower level of abstraction are verified, and that the performance indices are satisfactory
  - Thus, refinement process involves mapping constraints, performance indices and properties to the lower level, so that they can be computed for the next level down

Source: M. Jacome, UT Austin

Abstraction Levels



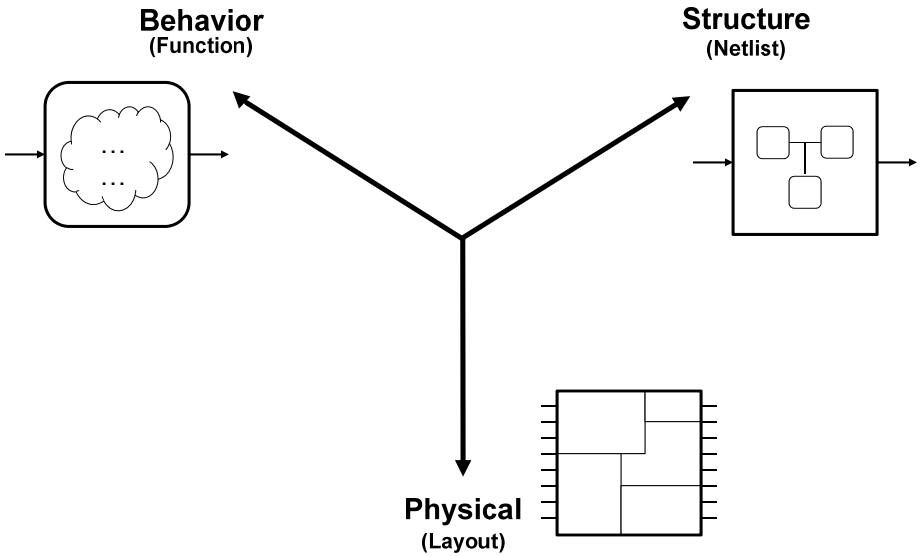
Design Methodology

- **Set of Models**
  - Design representations
    - Specification and documentation at interface between steps
- **Set of Transformations**
  - Design decisions and design steps
    - Refine input model into an output model reflecting decisions

➢ **Formalization of a design flow**
  - Break into well-defined, repeatable steps
  - ➢ Automate

Y-Chart

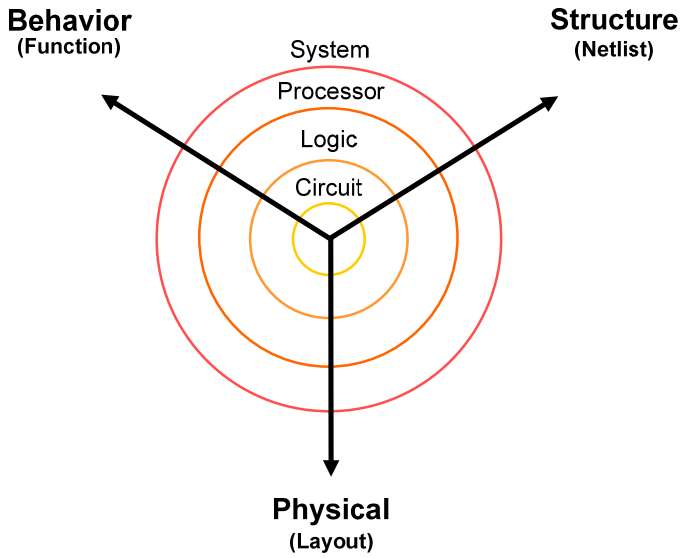
- **3 Design Views**



Source: D. Gajski, UC Irvine

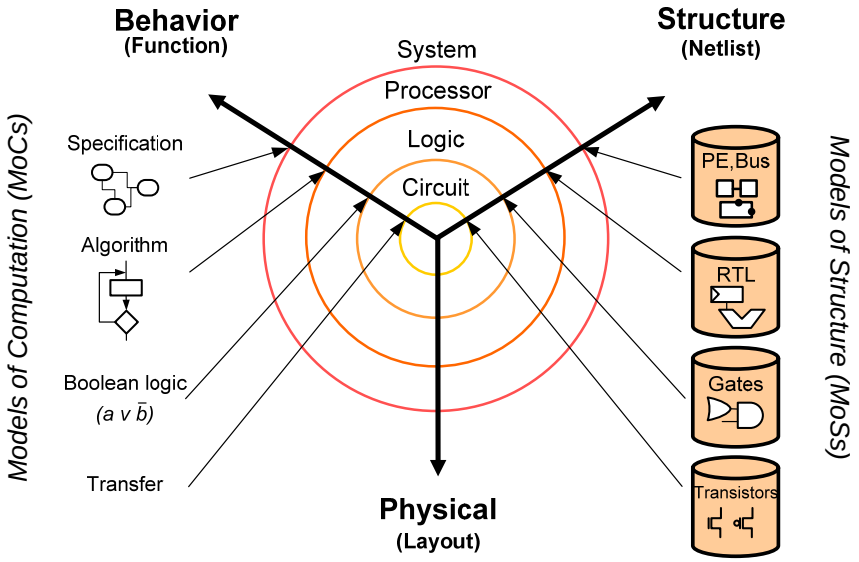
Y-Chart

- **4 Abstraction Levels**



Source: D. Gajski, UC Irvine

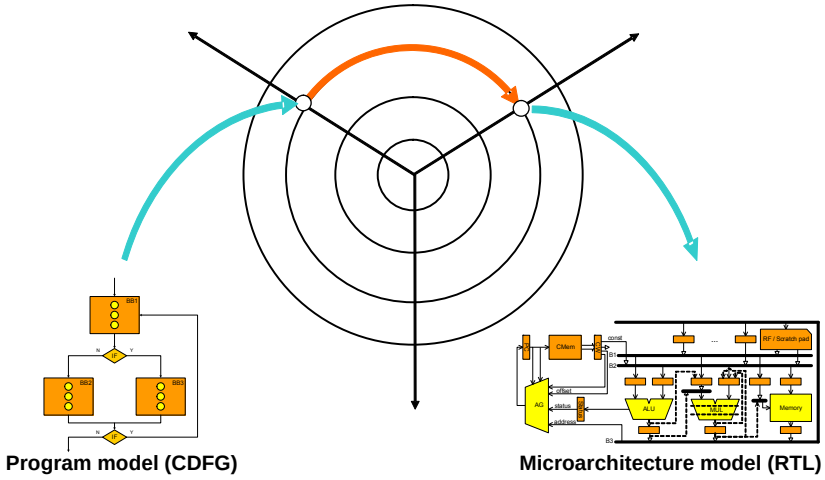
Y-Chart



Source: D. Gajski, UC Irvine

Processor Synthesis

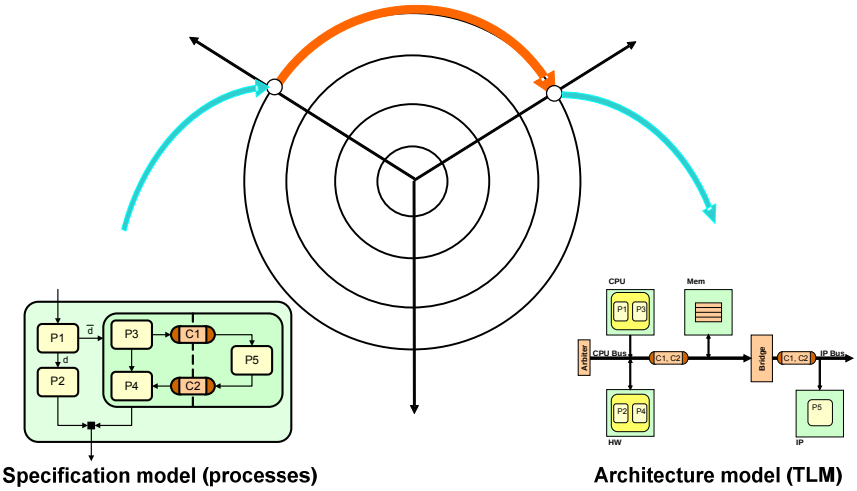
- **Software processor**
  - Compilation and linking
- **Hardware processor**
  - High-level synthesis



Source: D. Gajski, UC Irvine

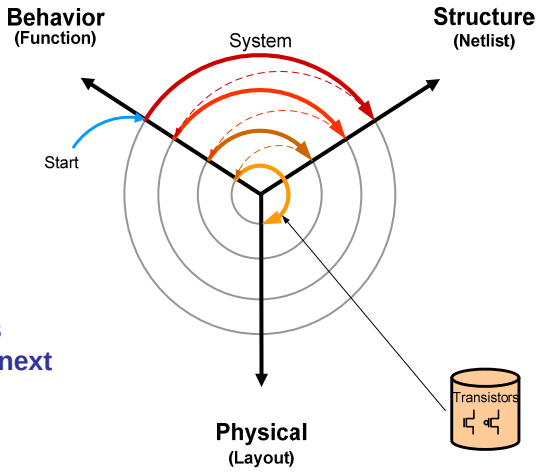
System Synthesis

- **Structure**
  - Partitioning, mapping
- **Timing**
  - Scheduling



Source: D. Gajski, UC Irvine

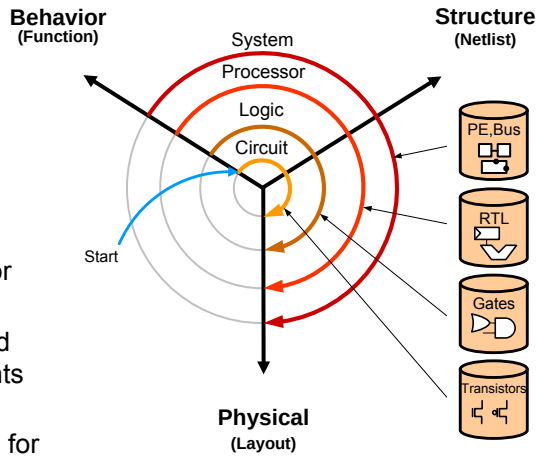
Top-down Methodology



- **Functional description is converted into component netlist on each level**
- **Each component function is decomposed further on the next abstraction level**
- **Layout is given only for transistor components**

Source: D. Gajski, UC Irvine

Bottom-Up Methodology

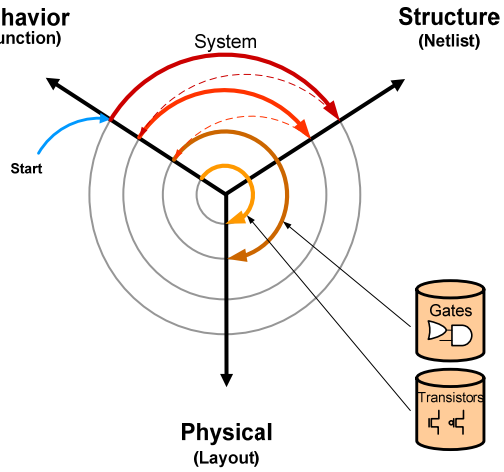


- **Each level generates library for the next higher level**
  - Circuit: Standard cells for logic level
  - Logic: RTL components for processor level
  - Processor: Processing and communication components for system level
  - System: System platforms for different applications
- **Floorplanning and layout on each level**

Source: D. Gajski, UC Irvine

Meet-in-the-Middle Methodology

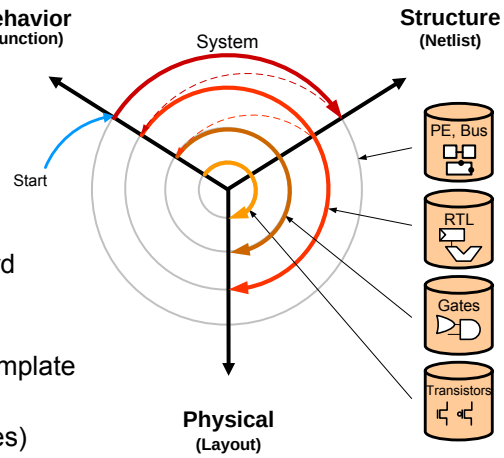
- **Gate netlist is hand-off**
- **Three levels of synthesis**
  - System is synthesized with processor components
  - Processor components are synthesized with RTL library
  - RTL components are synthesized with standard cells
- **Two levels of layout**
  - System layout is performed with standard cells
  - Standard cells layout with transistors



Source: D. Gajski, UC Irvine

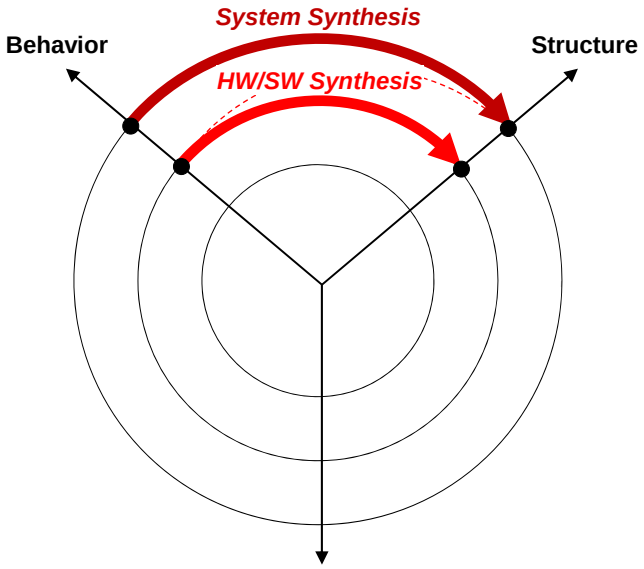
Platform-Based Design

- **Meet-in-the-middle at the system level**
  - System platform with standard components
  - System synthesis to map specification onto platform template
  - Some custom processor are synthesized (to RTL and gates)
  - Other (programmable) processors are pre-synthesized and just need software compilation
  - Layout and floorplanning at the SoC level

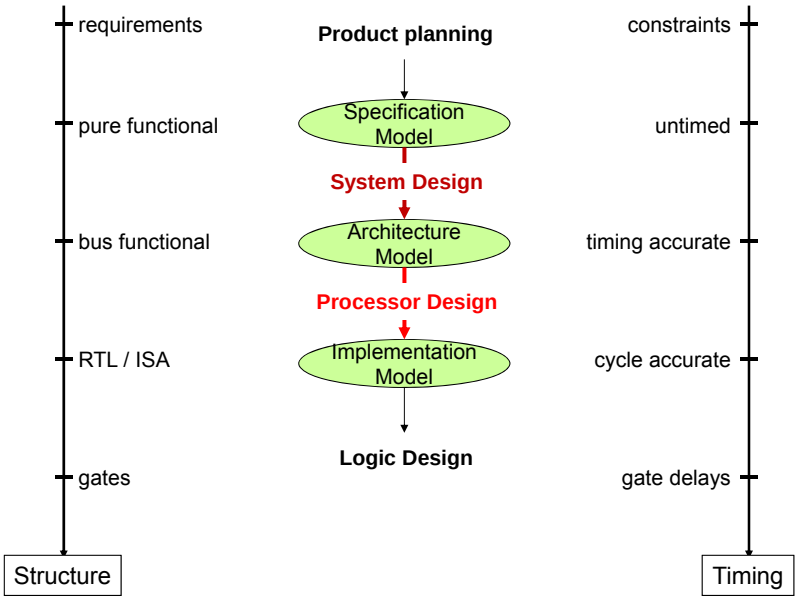


Source: D. Gajski, UC Irvine

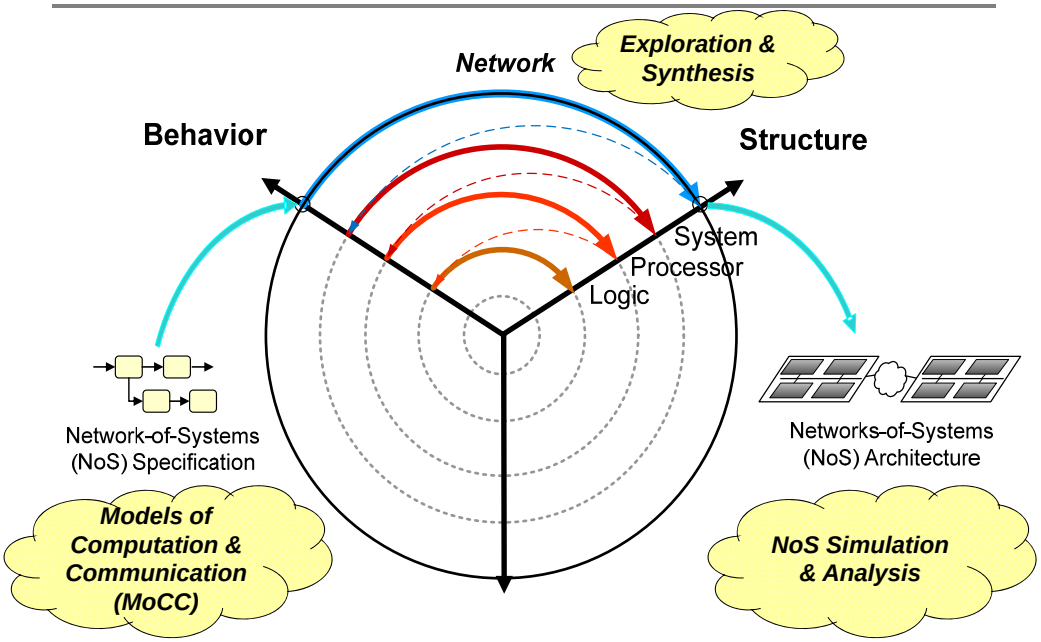
System-Level Design Methodology



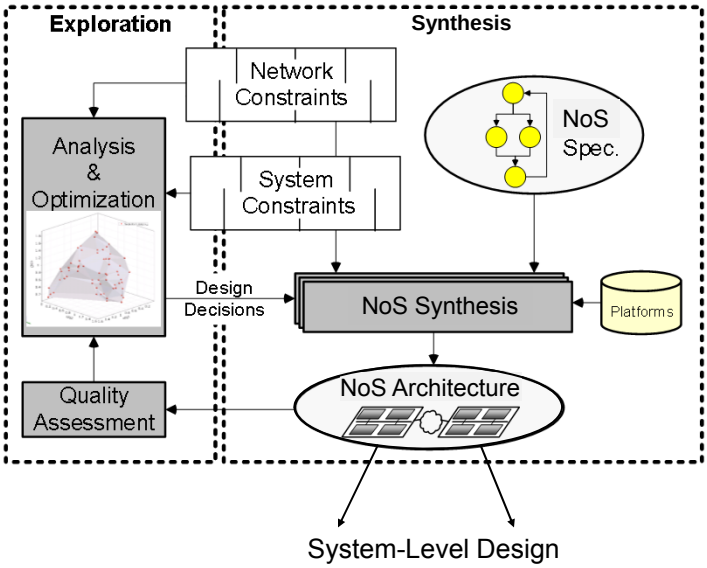
System-Level Design Flow



Network-Level Design



Network-Level Design Flow

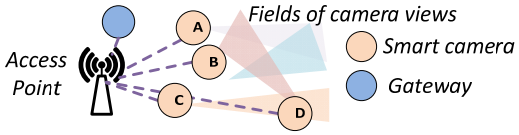


Lecture 2: Outline

- ✓ Design methodologies
  - ✓ Design process
  - ✓ Abstraction levels
  - ✓ Design models
  - ✓ System-level design flow
- **Design example**
  - Object recognition in smart camera network
  - Deep and convolutional neural networks (DNNs/CNNs)
  - Distributed CNN inference in IoT edge computing

IoT System Design Example

- **Smart camera network**
  - Smart city
  - Smart homes
  - …



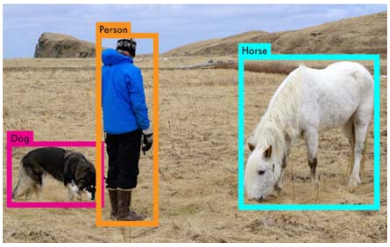
- **Edge computing**
  - Detect, locate and classify objects in video stream
    - Bounding boxes
    - Types of objects
  - Directly on cameras
    - Privacy
    - Real-time

Objection Detection using Deep Learning

- **Classification vs. detection**

Image classification, global feature

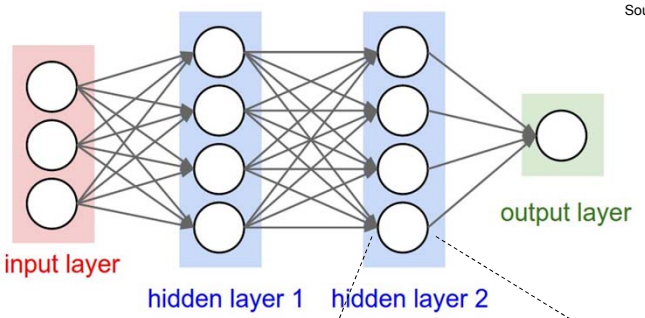
Object detection, classification + localization

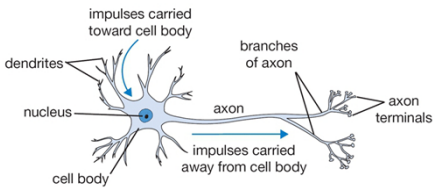


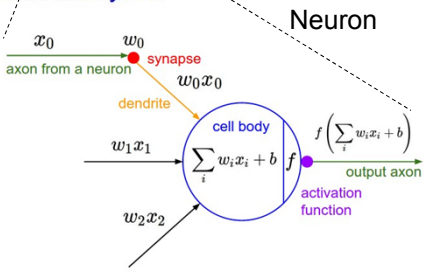
- Convolutional neural networks (CNNs) widely used for image classification
- Sliding windows of different size/shape + CNN-based classification for brute-force, naïve object detection

Traditional Neural Networks

Source: Stanford CS231n





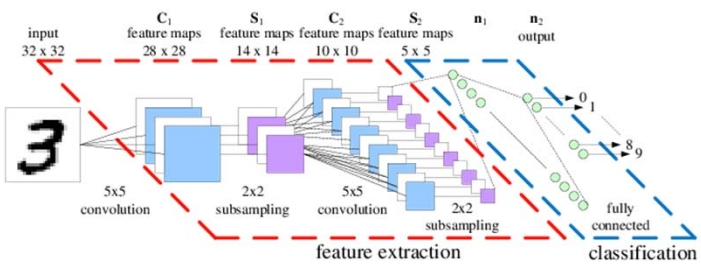


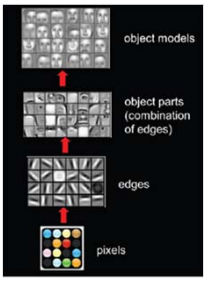
- **Deep Neural Networks (DNNs) with many hidden layers**

Convolutional Neural Networks (CNNs)

- **Different types of layers**
  - Convolutional: 2D convolutions with trainable filters
  - Rectified linear units (ReLU): elementwise function
  - Pooling: non-linear down-sampling
  - Fully connected: traditional neural networks

usually combined



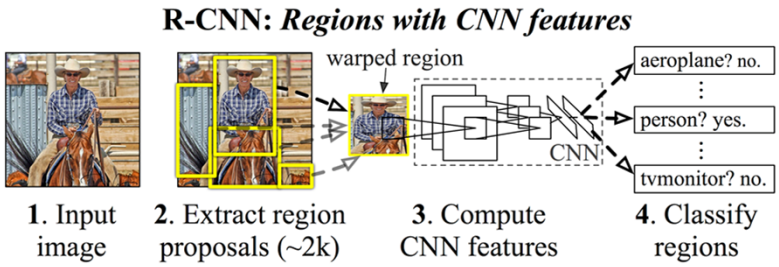


Digit recognition CNN (image classification)

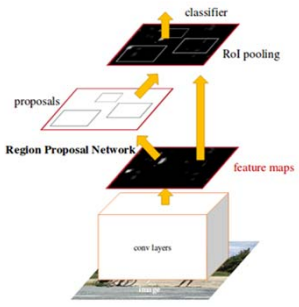
- **Fully convolutional network (FCN): no fully connected layer**

Object Recognition (1)

- **Region based (Fast/Faster/Mask R-CNN)**

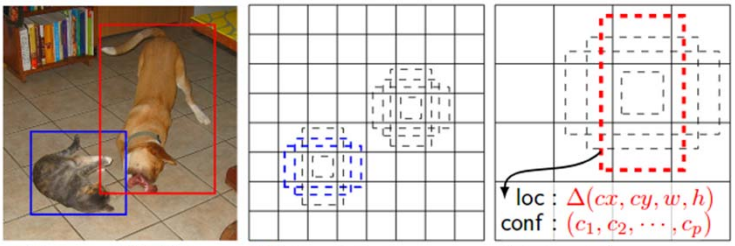


- Fast versions by sharing convolutional layers
- Common feature extraction for region proposal and classification



Object Recognition (2)

- **Single Shot Multibox Detector (SSD)**

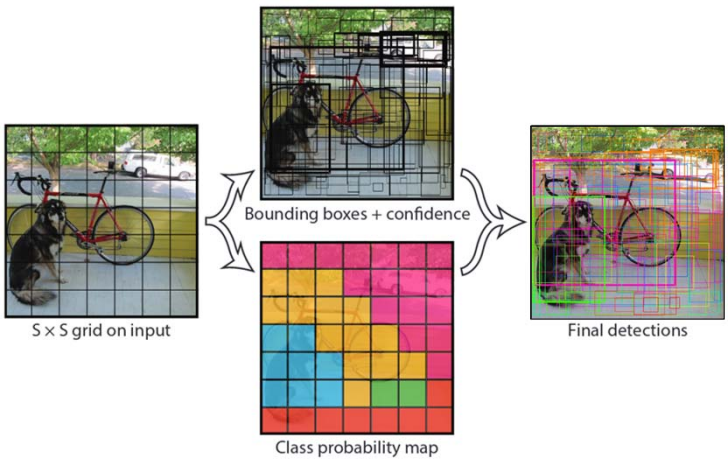


(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

- Slide window of fixed size and shape
- Detect both bounding box and class within window
- Predict likelihood of different box/class combinations

Object Detection (3)

- **You Only Look Once (YOLO)**
  - Don't slide window, predict for all possible boxes/classes



You Only Look Once (YOLO)

https://pjreddie.com/darknet/yolo/

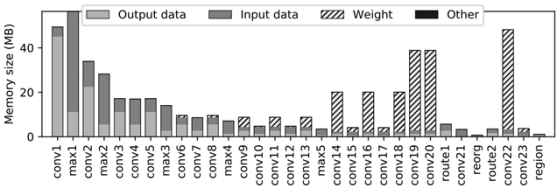
- **Default implementation on top of Darknet**
  - General open-source CNN framework/library in C
- **Also available for other deep learning frameworks**
  - PyTorch, Caffee2 [Facebook], TensorFlow [Google]

CNN Mapping (1)

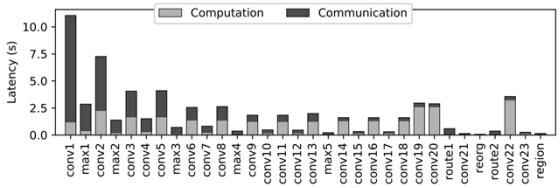
- **CNN inference resource demands**
  - Memory footprint
    - Early layers: data dominant
    - Later layers: weight dominant
  - Comm. overhead
    - Decreasing with number of layers

➢ **Distribute among edge cluster**
  - Collaborative processing
  - Accelerate in edge devices

Memory footprint per layer*



Comp. and comm. latency per layer*

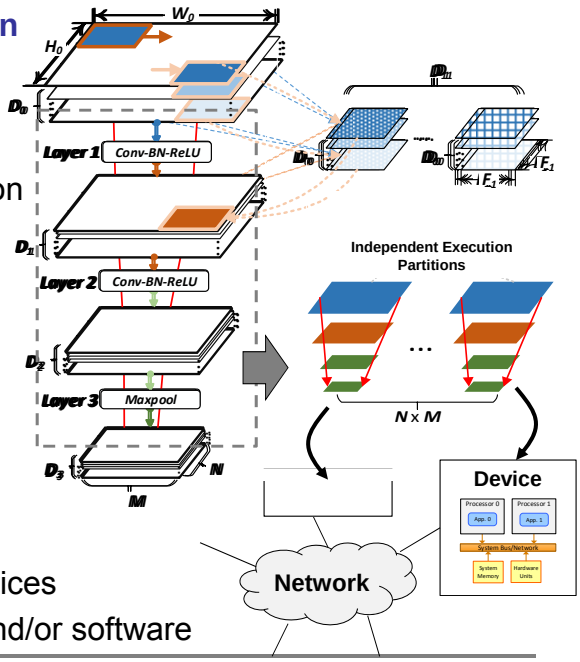


***Profiling data is collected based on the single-core performance of a Raspberry Pi 3 running YOLOv2.**

Source: Z. Zhao, K. Mirzazad, A. Gerstlauer, "DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters," IEEE TCAD, 2018

CNN Mapping (2)

- **Convolutional operation**
  - Locality between consecutive layers
  - ➢ Tile partitioning with boundary consideration
- **Chain of multiple convolutional layers**
  - Large amount of intermediate data
  - Synchronization overhead per layer
  - ➢ Layer fusion
- ➢ **Independent tasks**
  - ➢ Distribute among devices
  - ➢ Map into hardware and/or software



Lecture 2: Summary

- ✓ **Design methodologies**
  - ✓ Design process
  - ✓ Abstraction levels
  - ✓ Design models
  - ✓ System-level design flow
- ✓ **Design example**
  - ✓ Object recognition in smart camera network
  - ✓ Deep and convolutional neural networks (DNNs/CNNs)
  - ✓ Distributed CNN inference in IoT edge computing