**Runtime organization**: memory layout (code, static data, stack, heap)

**Code generation**

- Input/output: intermediate code (machine independent) and symbol table, target code for a given architecture
- Compiler/assembler/linker (compiler tool chain)
- Tasks:
  - Instruction selection
  - Manage registers, generate code for stacks
  - Code optimization
- Assumptions on architecture: how stack grows, how many registers, size of the registers, special registers
- Code generation
  - arithmetic expressions: recursive process, load data from memory to register
  - branch
  - calls: activation record and calling sequences
  - for OO: dispatch table, object layout
- Parameter passing: call by value, call by references, call by name

**Operational Semantics**

- Assembly defined semantics (architecture dependent)
- Operational semantics: how to evaluate a program on an abstract machine
- Environment (variable: memory address) and Store (value: memory location)
- One rule for each expression: model return value and side effect
  - Constant
  - Assignment
  - Sequence
  - Branch
  - Loop
  - Let expression
  - New
  - Method Dispatch

**Optimization**

- Basic concepts:
  - basic blocks and control flow graphs
  - definition/use
  - liveness
  - forward and backward analysis (constant propagation, liveness)
- Local/Global/Interprocedural (within a basic block, control flow graph, across procedure)
- Local:
  - Constant folding
  - Common subexpression elimination
  - Copy propagation (symbolic substitution)
Dead code elimination

- Global: control flow analysis and data flow analysis
  - Global constant propagation (all paths)
  - Global dead code elimination
- Cache optimization:
  - Loop interchange

Register Allocation

- Memory hierarchy management
  - Disk-Memory: programmer
  - Memory-Cache (hardware)
  - Memory-Register: compiler
- What is the register allocation problem?
- Register inference graph
  - Node: registers
  - Edge: can the data in both of the registers alive simultaneously?
- Graph coloring (k-colorable) algorithm
- Spilling

Garbage Collection

- Collect the space that will not be used again
- “reachable”: starting from registers and following all the pointers
- Mark and sweep algorithm
- Stop and copy
- Reference counting (advantages and disadvantages)

Just In Time Compilation

- Basic concepts: VM, Compilation, Interpretation, JIT, speculative optimization
- Why JIT?
- JIT overhead
  - Profiling (method counting, call stack sampling)
  - Initialization overhead
- JIT optimizations
  - Loop unrolling
  - Function inlining
  - On stack replacement
  - Code positioning
  - Inline cache
  - Instruction scheduling