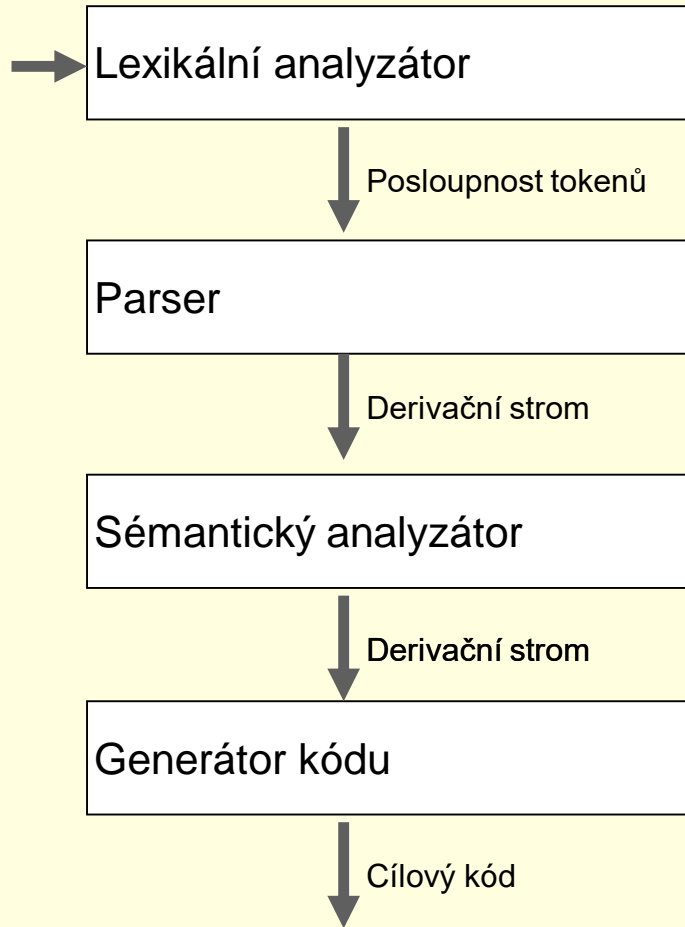
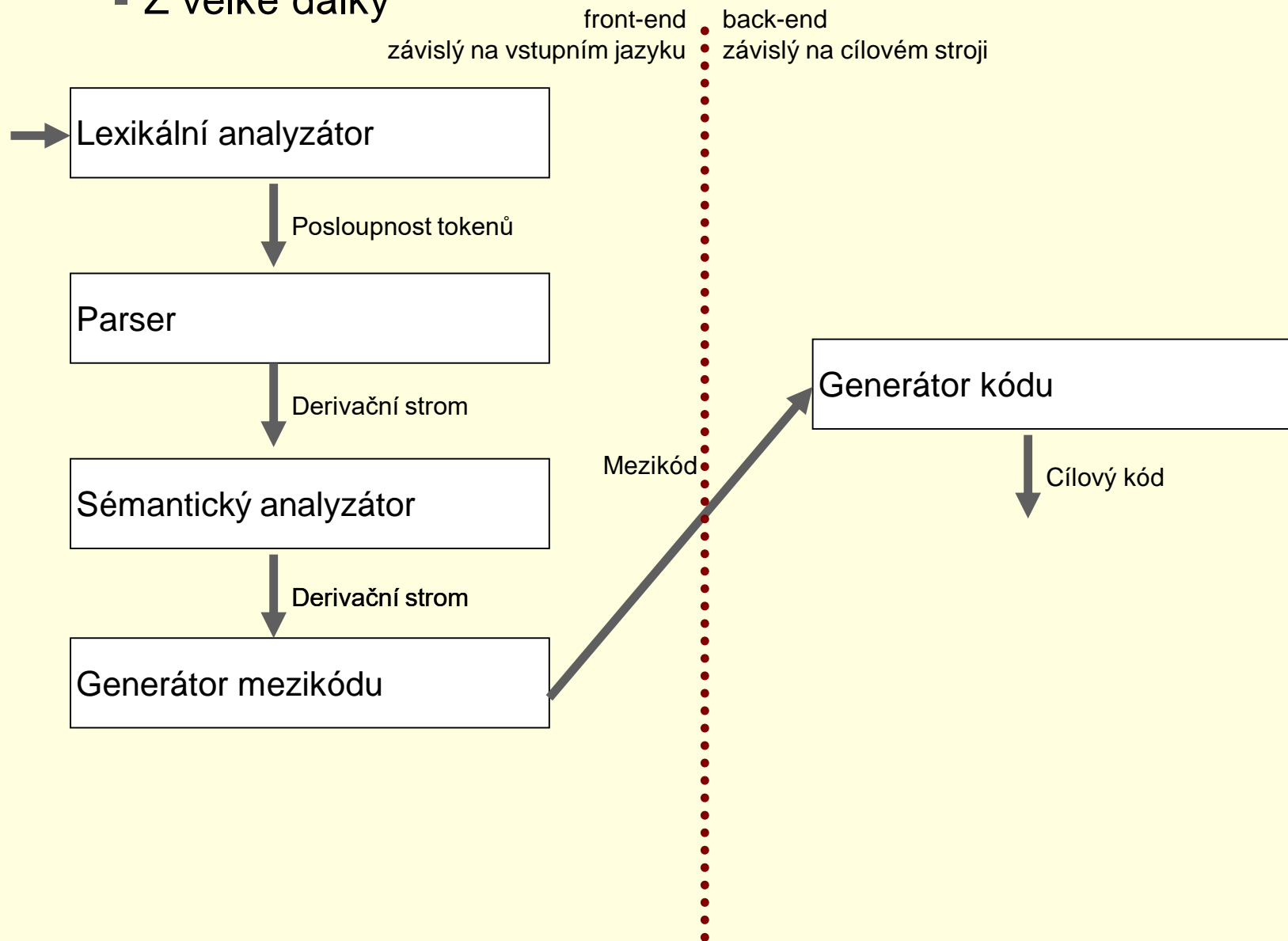


# Architektura překladače

## ▪ Amatérský pohled

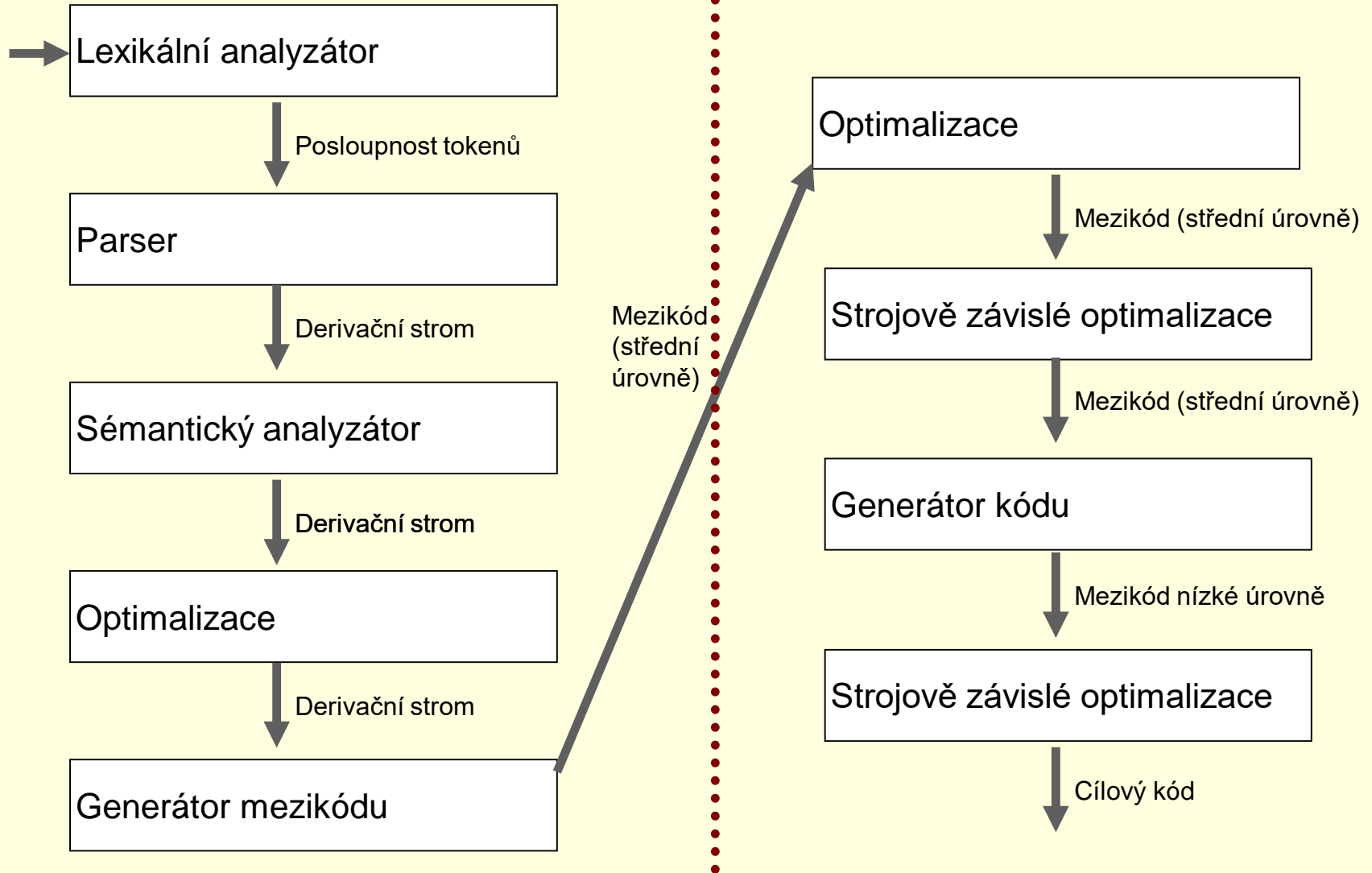


## ▪ Z velké dálky

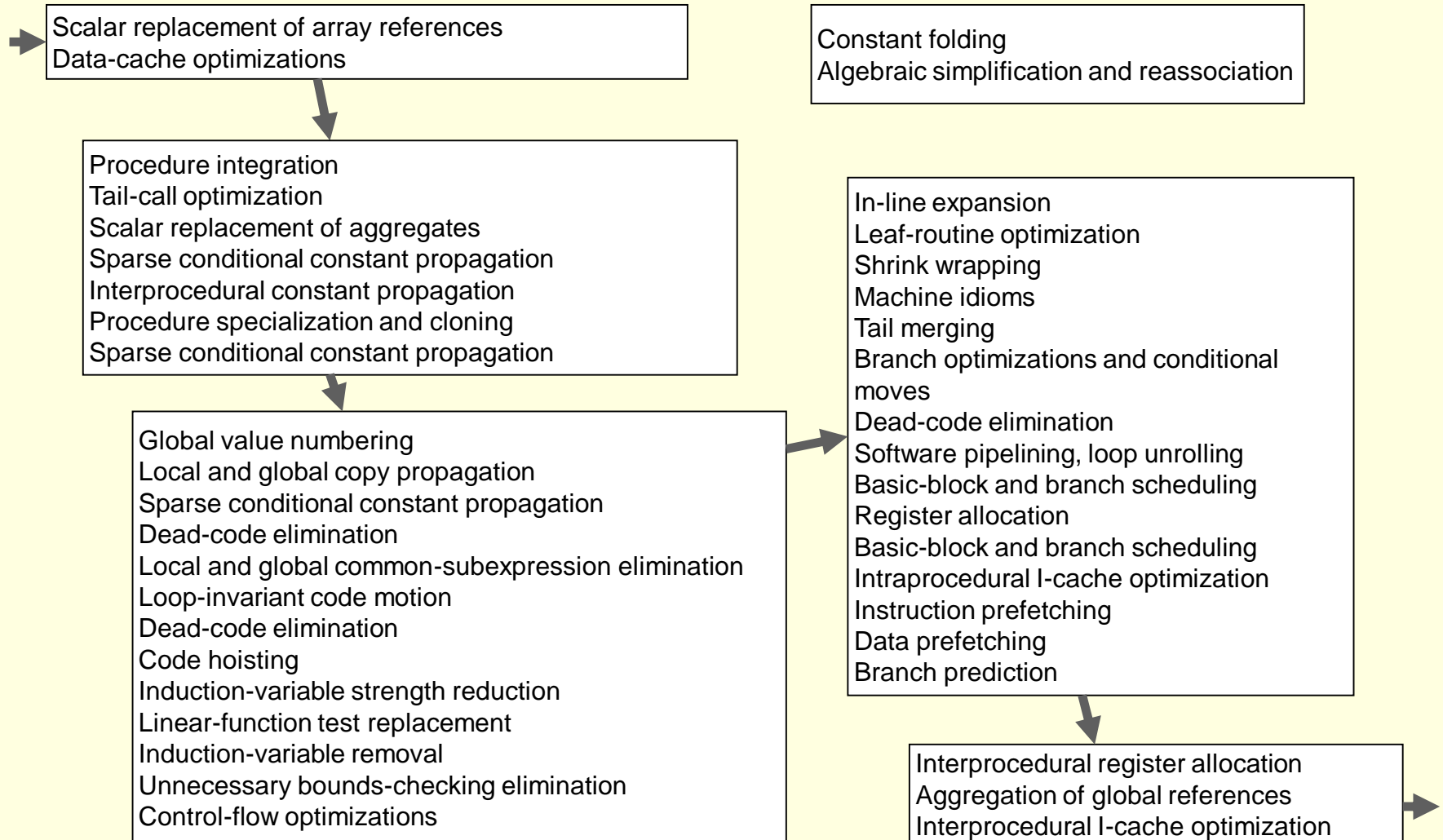


## ▪ S optimalizacemi

front-end závislý na vstupním jazyku      back-end závislý na cílovém stroji



- Detailní pohled akademika (pouze optimalizace)
  - Muchnick: Advanced Compiler Design and Implementation



## ❖ Realita

### ▪ GNU Compiler Collection Internals

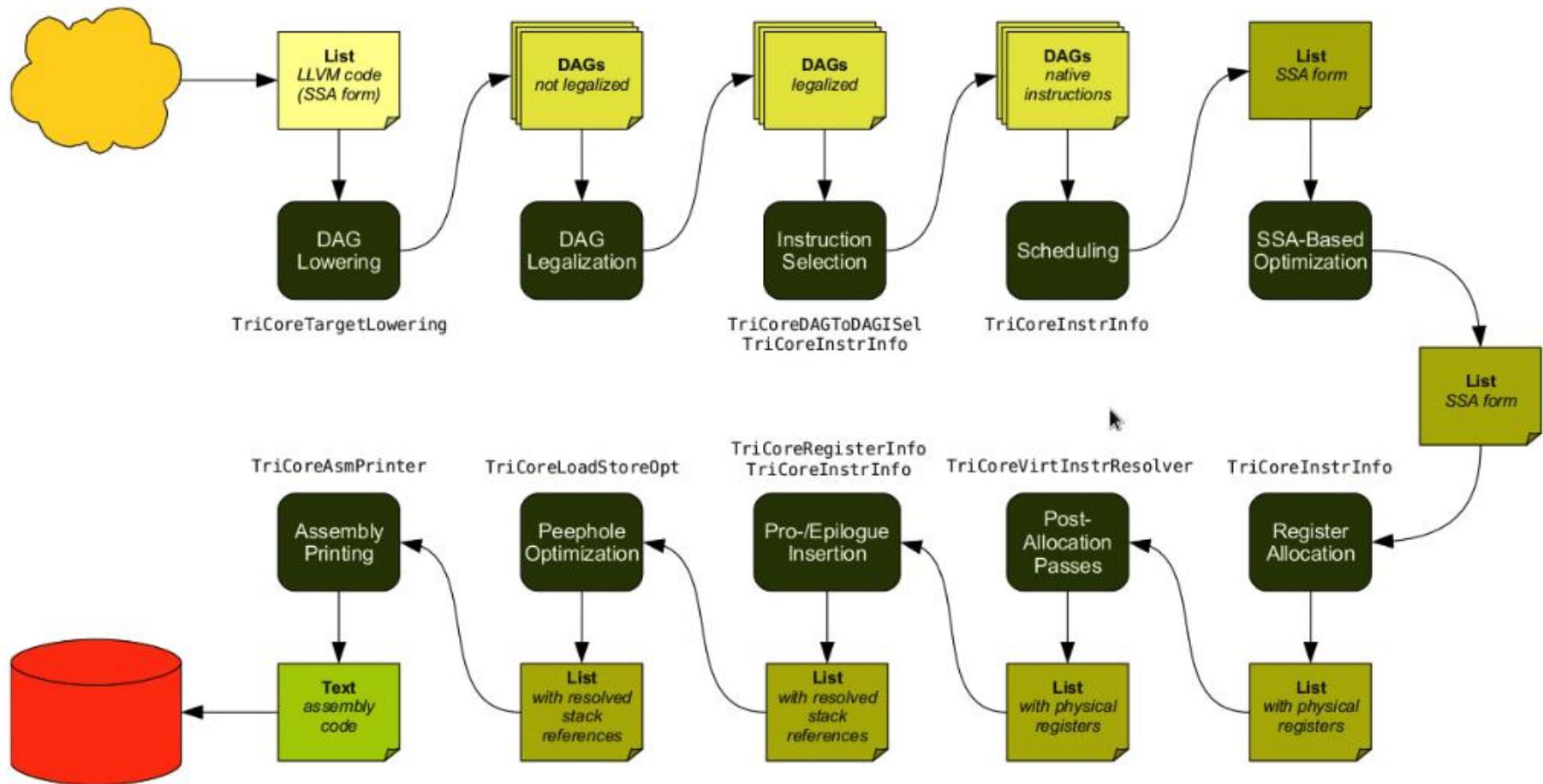
→ Remove useless statements  
Mudflap declaration registration  
Lower control flow  
Lower exception handling control flow  
Build the control flow graph  
Find all referenced variables

Enter static single assignment form  
Warn for uninitialized variables  
Dead code elimination  
Dominator optimizations  
Redundant phi elimination  
Forward propagation of single-use variables  
Copy renaming  
PHI node optimizations  
May-alias optimization  
Profiling  
Lower complex arithmetic  
Scalar replacement of aggregates  
Dead store elimination  
Tail recursion elimination  
Forward store motion  
Partial redundancy elimination  
Loop invariant motion  
Canonical induction variable creation  
Induction variable optimizations  
Loop unswitching  
Vectorization  
Tree level if-conversion for vectorizer  
Conditional constant propagation  
Folding builtin functions  
Split critical edges  
Partial redundancy elimination  
Control dependence dead code elimination  
Tail call elimination  
Warn for function return without value  
Mudflap statement annotation  
Leave static single assignment form

RTL generation  
Generate exception handling landing pads  
Cleanup control flow graph  
Common subexpression elimination  
Global common subexpression elimination.  
Loop optimization  
Jump bypassing  
If conversion  
Web construction  
Life analysis  
Instruction combination  
Register movement  
Optimize mode switching  
Modulo scheduling  
Instruction scheduling  
Register class preferencing  
Local register allocation  
Global register allocation  
Reloading  
Basic block reordering  
Variable tracking  
Delayed branch scheduling  
Branch shortening  
Register-to-stack conversion  
Final  
Debugging information output

## ❖ Realita

- LLVM back-end





# LLVM 6.0 code generator (AMD64, -O3 -mavx2)

8

- **Instruction Selection**
- Expand ISEL pseudo-instructions
- X86 Domain Reassignment Pass
- Tail Duplication
- Optimize machine instruction PHIs
- Merge disjoint stack slots
- Local Stack Slot Allocation
- Remove dead machine instructions
- Early If-Conversion
- Machine InstCombiner
- X86 cmov Conversion
- Machine Loop Invariant Code Motion
- Machine Common Subexpression Elimination
- Machine code sinking
- Peephole Optimizations
- Remove dead machine instructions
- Live Range Shrink
- X86 Fixup SetCC
- X86 LEA Optimize
- X86 Optimize Call Frame
- X86 WinAlloca Expander
- Detect Dead Lanes
- Process Implicit Definitions
- **Live Variable Analysis**
- Machine Natural Loop Construction
- **Eliminate PHI nodes for register allocation**
- **Two-Address instruction pass**
- Simple Register Coalescing
- Rename Disconnected Subregister Components
- **Machine Instruction Scheduler**
- **Greedy Register Allocator**
- Virtual Register Rewriter
- Stack Slot Coloring
- Machine Loop Invariant Code Motion
- X86 FP Stackifier
- Shrink Wrapping analysis
- **Prologue/Epilogue Insertion & Frame Finalization**
- Control Flow Optimizer
- Tail Duplication
- Machine Copy Propagation Pass
- Post-RA pseudo instruction expansion pass
- X86 pseudo instruction expansion pass
- Post RA top-down list latency scheduler
- Analyze Machine Code For Garbage Collection
- Branch Probability Basic Block Placement
- X86 Execution Dependency Fix
- X86 vzeroupper inserter
- X86 Byte/Word Instruction Fixup
- X86 Atom pad short functions
- X86 LEA Fixup
- Compressing EVEX instrs to VEX encoding when possible
- Contiguously Lay Out Funclets
- StackMap Liveness Analysis
- Live DEBUG\_VALUE analysis
- Insert fentry calls
- Insert XRay ops
- Implement the 'patchable-function' attribute
- X86 Retpoline Thunks



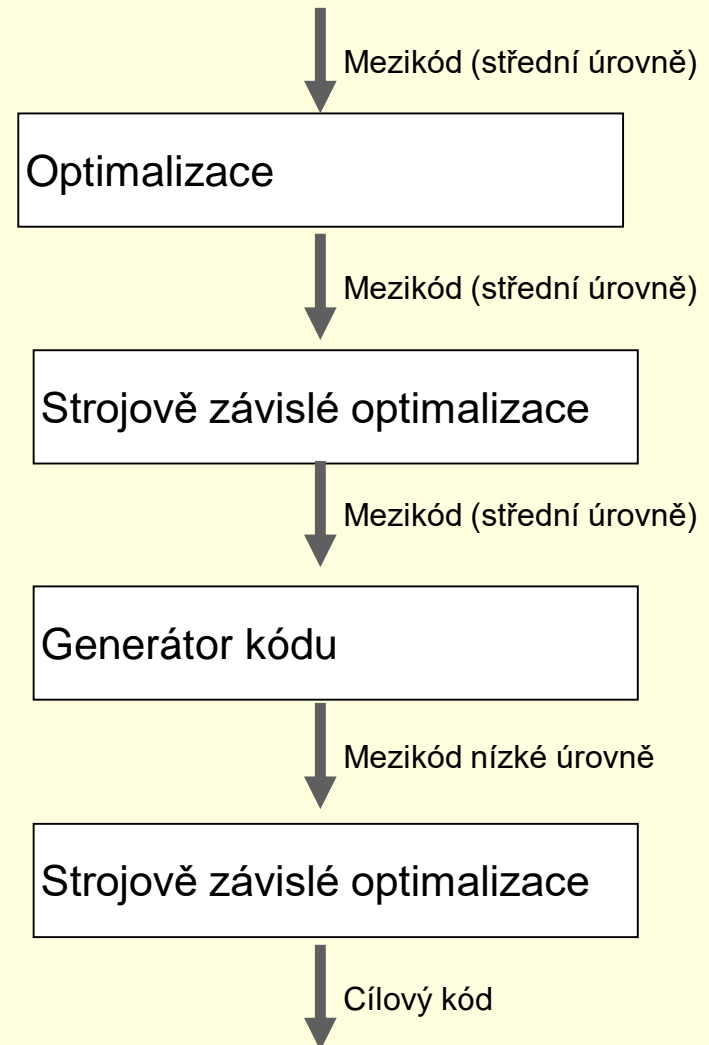
## ➤ Různé vnitřní reprezentace

### ❖ Mezikód střední úrovně

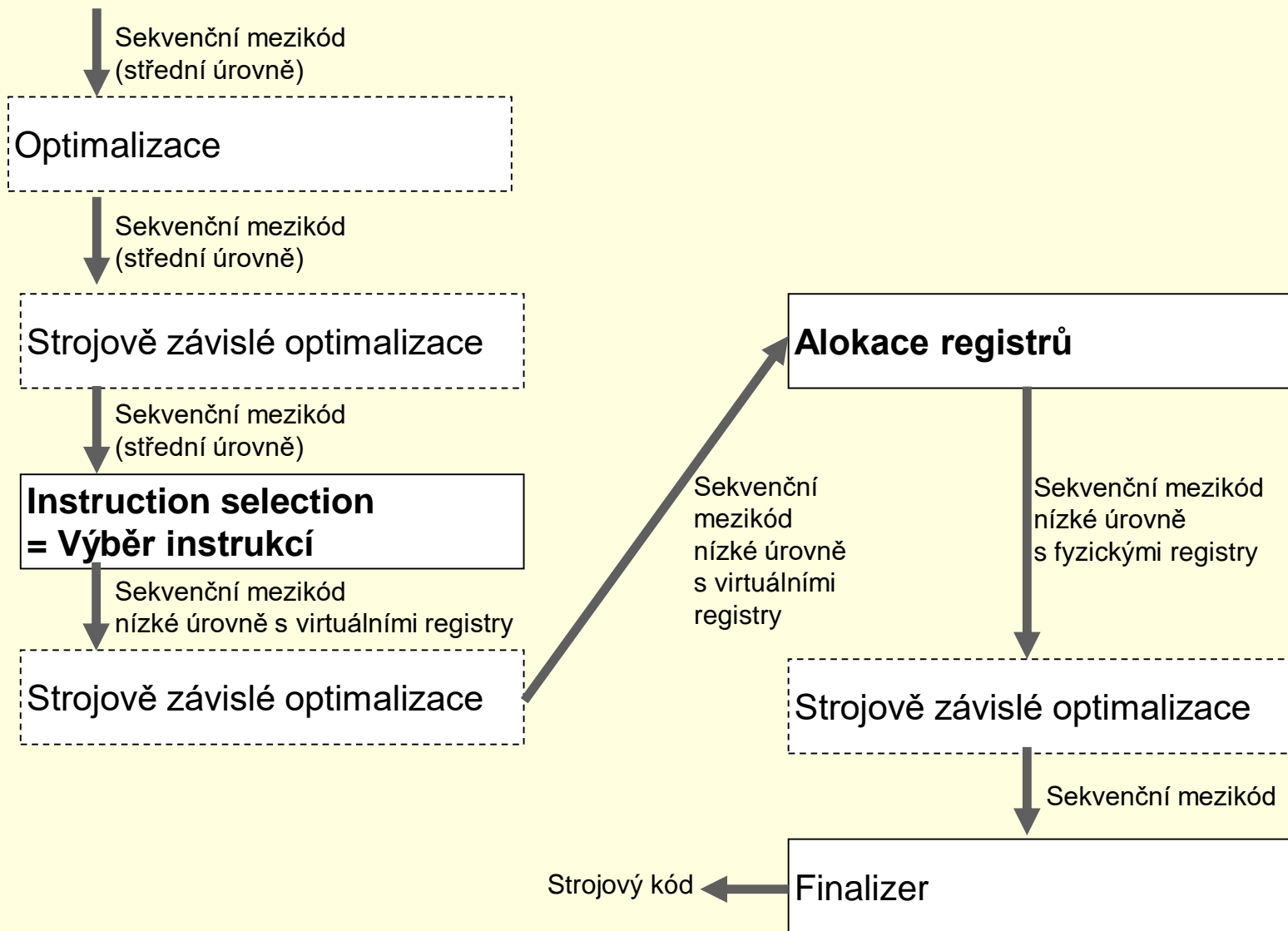
- Nezávislá sada operací
  - ADD\_I32 a,b,c
- Forma
  - Nesekvenční
  - Částečně sekvenční

### ❖ Mezikód nízké úrovně

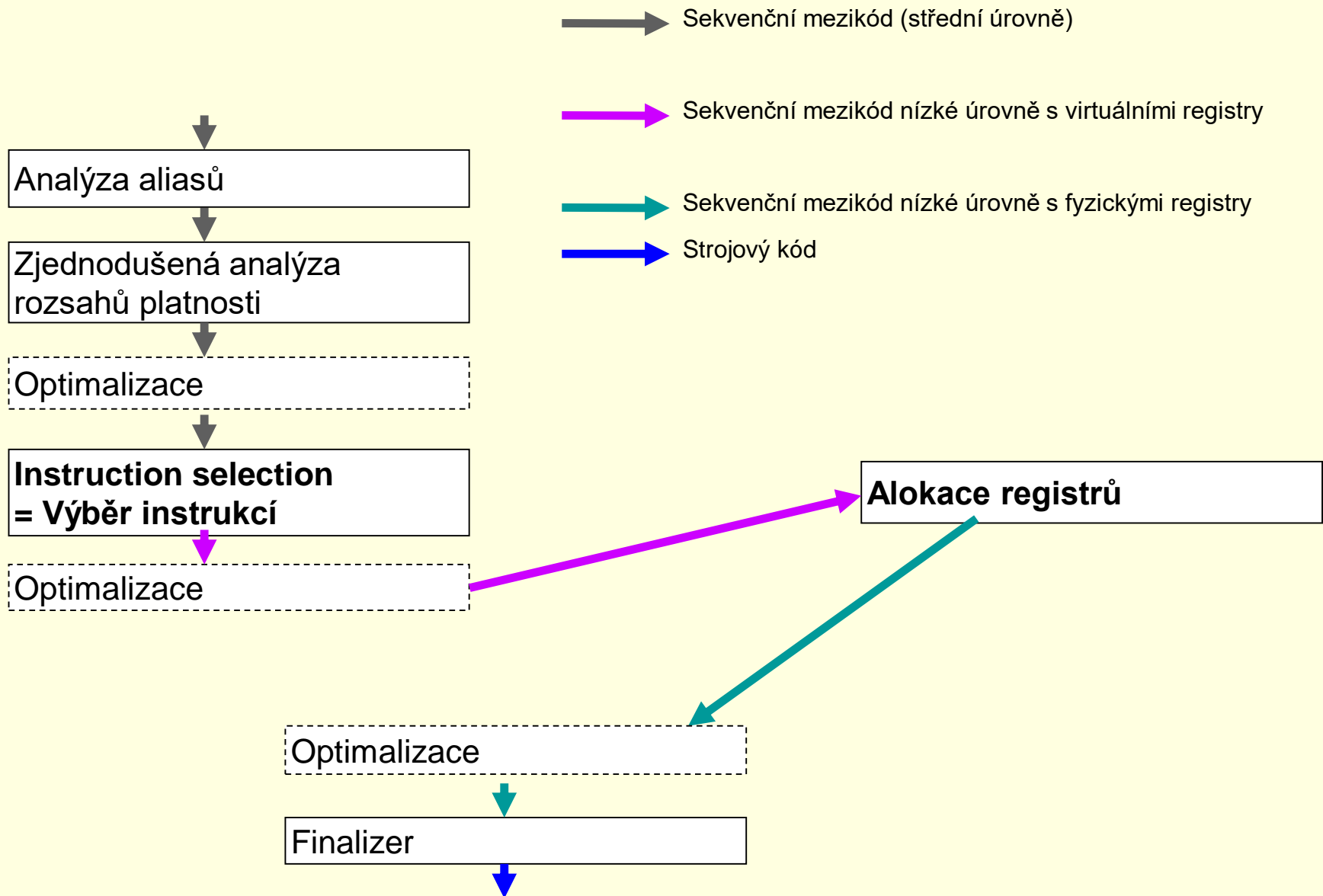
- Ekvivalenty strojových instrukcí
  - add r1,r2
- Forma
  - Nesekvenční
  - Částečně sekvenční
  - Sekvenční



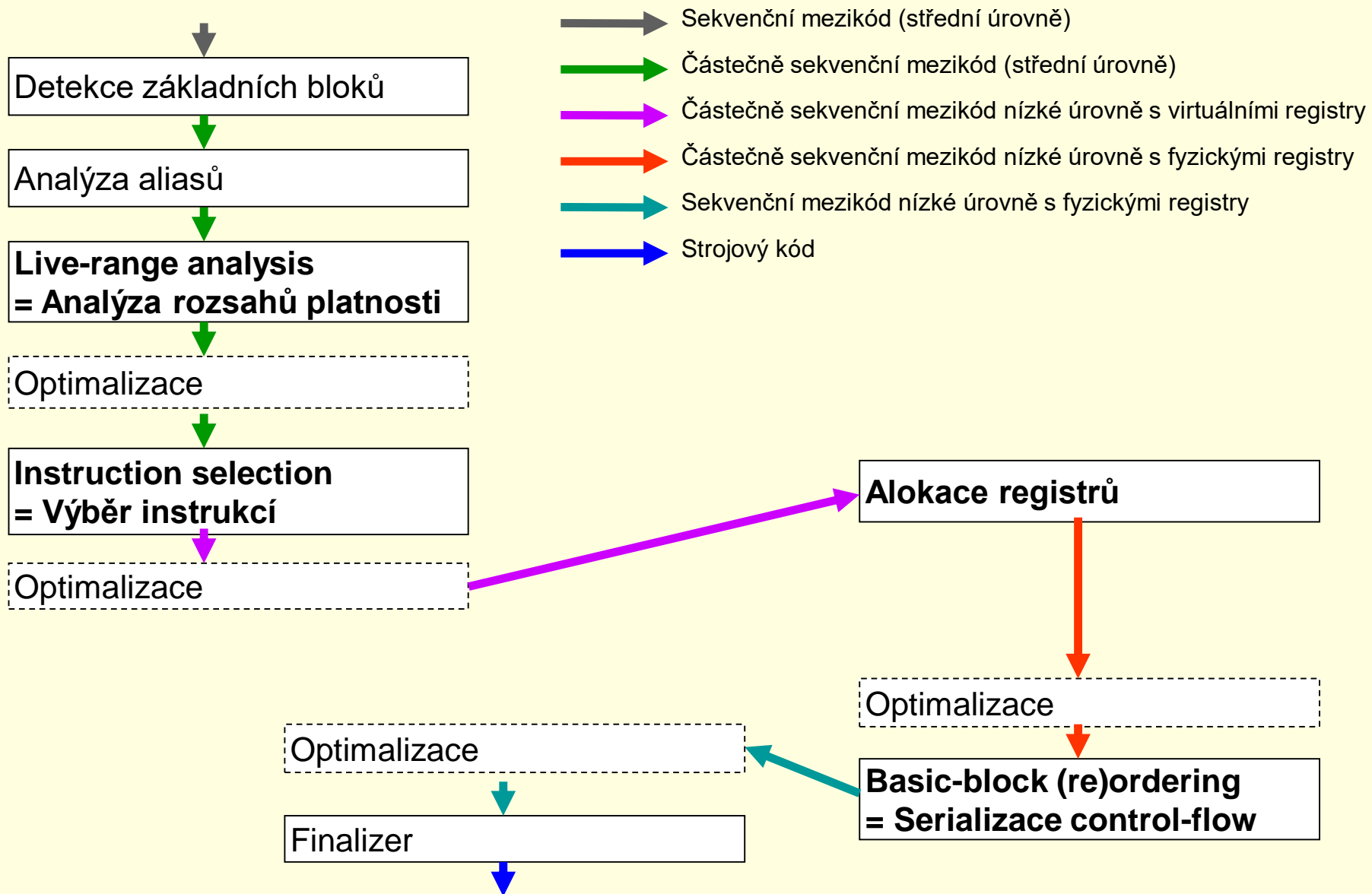
## ▪ Sekvenční mezikód



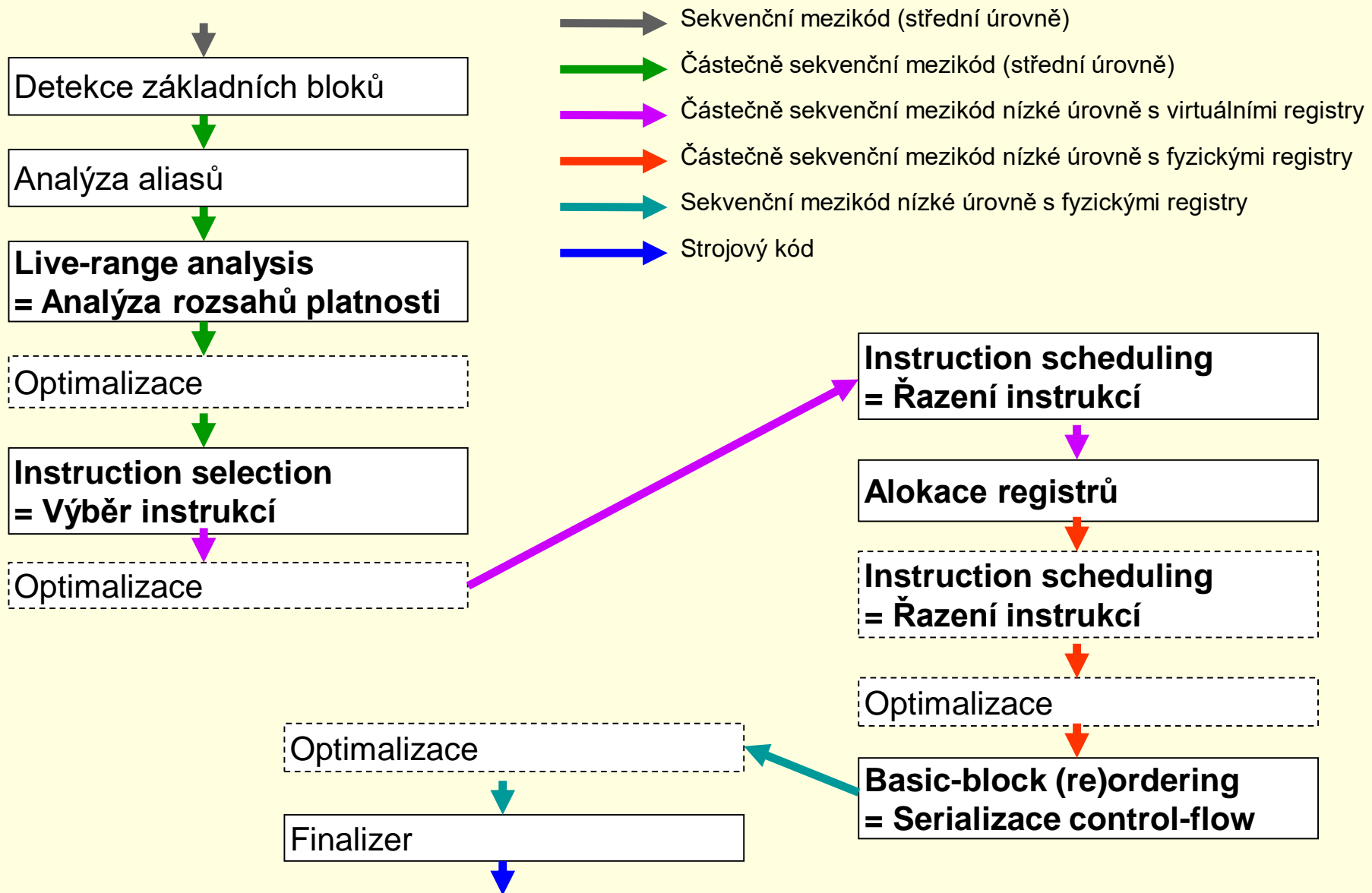
## ▪ Sekvenční mezikód



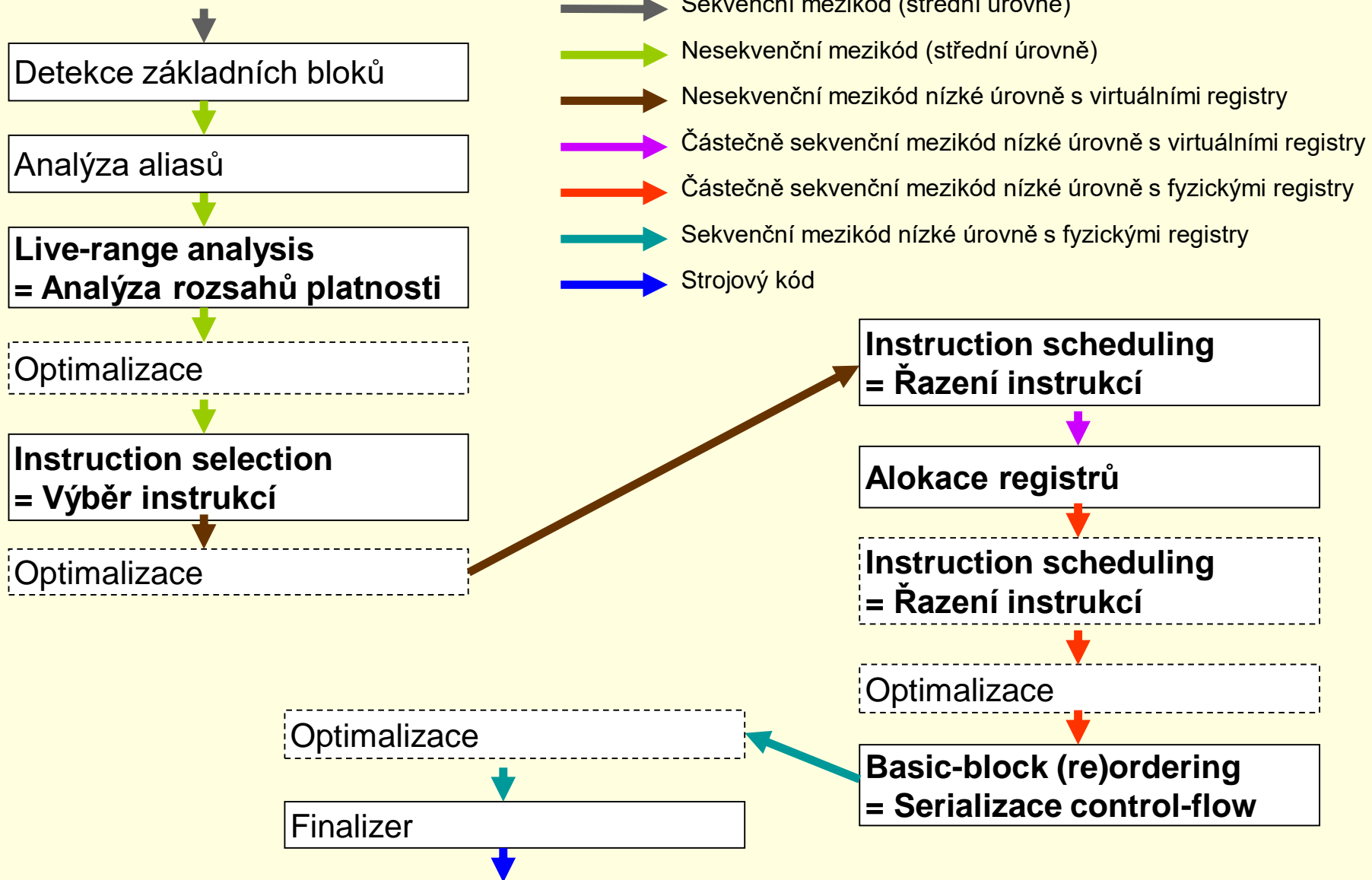
## ▪ Částečně sekvenční mezikód bez schedulingu



- Částečně sekvenční mezikód se schedulingem



## ▪ Nesekvenční mezikód





## ❖ Instruction selection

- Výběr strojových instrukcí
  - 1:n – přímočaré řešení
  - m:n – stromové/grafové gramatiky apod.
- Vliv na kvalitu kódu poklesl
  - RISC, load-store kód apod.

## ❖ Instruction scheduling

- Řazení instrukcí pro lepší využití ILP (instruction-level parallelism)
  - NP-úplná úloha
- Lokální v BB
  - Speciální řešení smyček (software pipelining)
  - Částečně globální varianty (trace scheduling)
- Zrychluje kód o 30-150%

## ❖ Register allocation

- Přidělování fyzických registrů
  - NP-úplná úloha
  - Standardní řešení: Barvení grafu