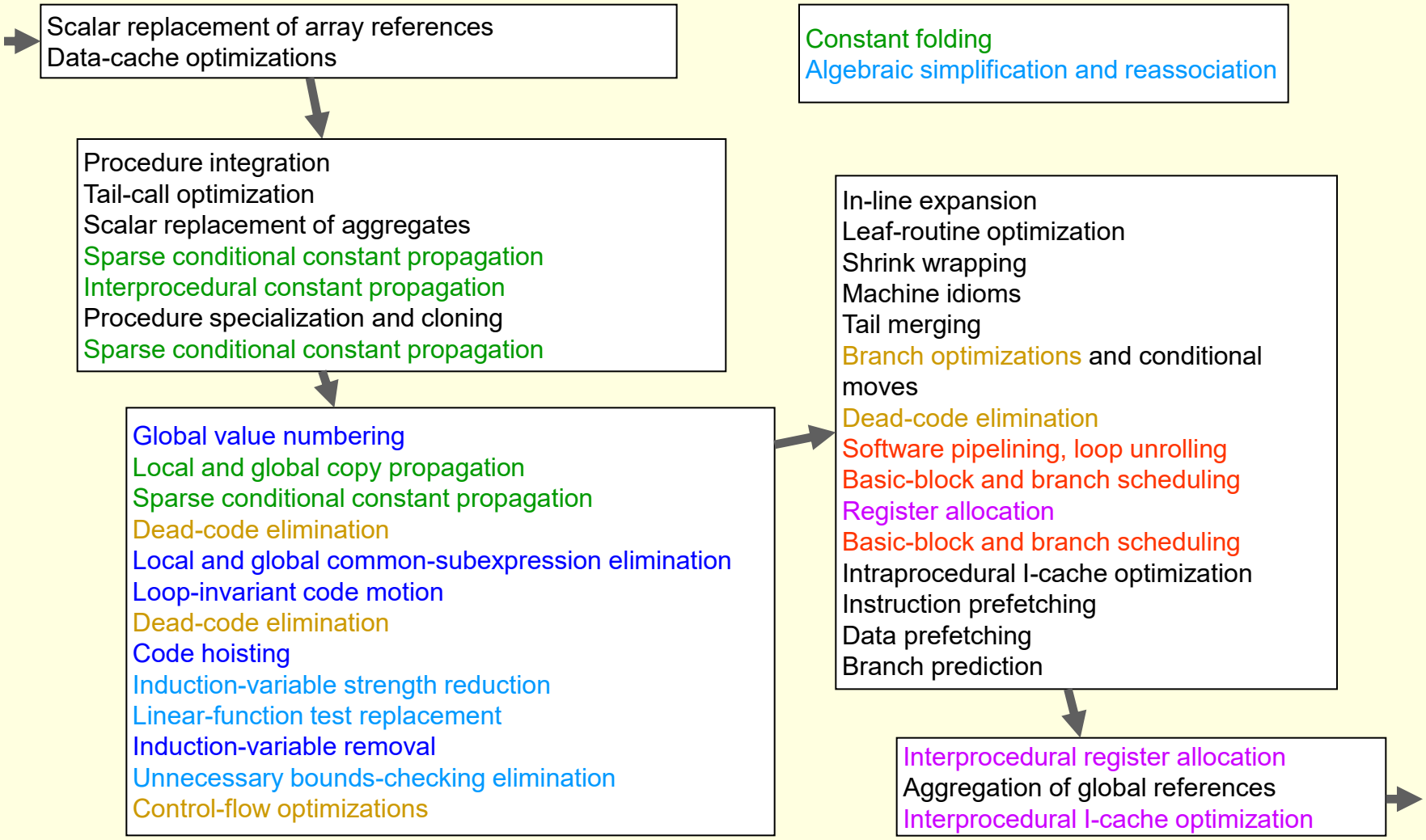
Další optimalizace

Architektura překladače

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



Další optimalizace

- **Částečné vyhodnocení**
  - Část požadovaného výpočtu je vyhodnocována již překladačem
  - Výpočet konstantních výrazů
    - Constant-expression evaluation (constant folding)
  - Výpočet podmíněně konstantních výrazů
    - Sparse conditional constant propagation
- **Algebraické úpravy**
  - Využití algebraických identit ke zjednodušení kódu
  - Algebraické úpravy výrazů
  - Redukce síly v cyklech
    - Strength reduction
  - Odstranění zbytečných kontrol mezí

Další optimalizace

- **Odstranění redundance**
  - Nahrazení opakovaných výpočtů uložením výsledku
  - Copy propagation
  - Lokální/globální eliminace společných podvýrazů
    - Common-subexpression elimination
  - Přesun invariantního kódu z cyklu
    - Loop-invariant code motion
  - Partial-redundancy elimination
    - Lazy code motion

- **Odstranění neužitečného kódu**
  - Odstranění mrtvého kódu
    - Dead-code elimination
  - Odstranění nedosažitelného kódu
    - Unreachable-code elimination
  - Optimalizace skoků
    - Jump optimization

Částečné vyhodnocení

- Výpočet (pod)výrazů obsahujících pouze konstanty
  - Constant-expression evaluation
  - Obvykle prováděn již front-endem
- Určení proměnných s konstantním obsahem
  - Constant folding
- Určení proměnných s podmíněně konstantním obsahem
  - Sparse conditional constant propagation
  - Upravuje control-flow!

```
a = b + (4 * 10);
```

```
c = 4 * 10;
d = c + 5;
e = f + (d * 2);
```

```
if ( g > h )
  i = 1;
else
  i = 0;
j = i + 1;
if ( j > 1 )
  k = k + 1;
```

Částečné vyhodnocení

- Výpočet (pod)výrazů obsahujících pouze konstanty
  - Constant-expression evaluation
  - Obvykle prováděn již front-endem
- Určení proměnných s konstantním obsahem
  - Constant folding
- Určení proměnných s podmíněně konstantním obsahem
  - Sparse conditional constant propagation
  - Upravuje control-flow!

- Integrace procedur generuje nové příležitosti pro částečné vyhodnocení

```
void f( int i, bool f)
{
  int j = i + 1;
  if ( f )
    g( j);
  else
    h( j);
}

f( k + 1, false);
```

Částečné vyhodnocení


```
a = b + 40;
```

```
c = 40;
d = 45;
e = f + 90;
```

```
if ( g > h )
{
  i = 1;
  j = 2;
  k = k + 1;
}
else
{
  i = 0;
  j = 1;
}
```

```
a = b + (4 * 10);
```

```
c = 4 * 10;
d = c + 5;
e = f + (d * 2);
```

```
if ( g > h )
  i = 1;
else
  i = 0;
j = i + 1;
if ( j > 1 )
  k = k + 1;
```

Algebraické úpravy

- Algebraické úpravy výrazů
  - Většinou v souvislosti s přítomností konstant
  - Důležité pro ukazatelovou aritmetiku (přístup k polím)
  - Úprava do kanonického tvaru
- Redukce síly v cyklech
  - Důležité pro přístup k polím
- Odstranění zbytečných kontrol mezí
- Machine idioms
  - Instrukce provádějící speciální kombinace nebo varianty operací

```
a = ((b * 3) + 7) * 5

a = b * 15 + 35
```

```
for ( i = 1; i < 10; ++i )
  a[ 4 * i] = 0;

for ( j = 4; j < 40; j += 40 )
  a[ j] = 0;
```

```
int b[ 10];
for ( i = 0; i < 10; ++i )
  b[ i] = 0;
```

Algebraické úpravy

- Převedení control-flow na algebraické operace
  - Conditional move
    - Ušetří podmíněné skoky
    - Může přidat zbytečné operace
  - Užitečnost obtížně odhadnutelná
    - Cena podmíněných skoků závisí na úspěšnosti predikce
    - Překladač úspěšnost nedokáže odhadnout
    - Profilem řízené optimalizace

```
if ( a > b )
  c = d + e;
```

```
CMP t,a,b
ADD u,d,e
CMOV t,c,u
```

Odstranění redundance

- Copy propagation
- Lokální/globální eliminace společných podvýrazů
- Přesun invariantního kódu z cyklu
  - Častý výskyt u přístupu k polím
- Partial-redundancy elimination
  - Lazy code motion
- Integrace procedur generuje nové redundance

```
b = a;
c = b;          // c = a;

c = a + b;
d = a + b;      // d = c;

for ( i = 0; i < 10; ++i)
  a[ i][ k] = a + b;


if ( a < b )
  c = d + e;
f = d + e;
```

Odstranění redundance

- **Lazy code motion**
  - Move a computation to the latest moment before the result is needed
    - This applies to every computation
  - The fixed barriers:
    - Returns from functions
    - Conditions in control-flow
    - If a computation is not loop-invariant, it may be delayed but not completely moved out of the loop
  - Then manipulate the control flow so that the computation is never repeated

```
s = 0;
if ( c )
{ for ( i = 0; i < n; ++i)
    s = s + a[k][i];
}
else
  s = -1;
```

- Transformed to:

```
if ( c )
{ if ( 0 < n )
  { s = 0; r = &a[k]; i = 0;
    do {
      s = s + (*r)[i]; ++i;
    } while (i < n);
  }
  else
    s = 0;
}
else
  s = -1;
```

Odstranění neužitečného kódu

- Odstranění mrtvého kódu
  - Kód, jehož efekt nebude využit
    - Přiřazení do proměnných, které již nebudou čteny
- Odstranění nedosažitelného kódu
  - Kód, ke kterému nevede cesta
- Optimalizace skoků
  - Skoky na skoky apod.
- Řeší především chyby vyprodukované předchozími fázemi
  - Aplikuje se opakovaně

```
a = b + 40;



c = 40;
d = 45;
e = f + 90;


if ( g > h )
{
  i = 1;
  j = 2;
  k = k + 1;
}
else
{
  i = 0;
  j = 1;
}
```

Optimalizace režie volání

- Integrace procedur
  - a.k.a inline-expansion
- Listové procedury
  - Nevolají žádné další
  - Není třeba kompletní prolog a epilog procedury
  - Užitečné zejména v přítomnosti výjimek
- Shrink wrapping
  - Přesouvání prologu a epilogu
  - Ve větvích, které nevolají další procedury, se prolog a epilog anihilují

- Tail merging
  - Ztotožnění identických konců procedur
    - Poslední BB často obsahuje pouze epilog a je tudíž shodný
  - Šetří velikost kódu
    - Zlepšuje využití I-cache

Interprocedurální úpravy

- Specializace procedur
  - Procedury se naklonují
  - Jednotlivé klony se přizpůsobují okolnostem, které panují při jejich volání
    - Speciální tvary argumentů
    - Konstantní argumenty
    - Aliasing
- Interprocedurální alokace registrů
  - Volací konvence se upravuje podle místních podmínek

- Efekt je obdobný jako u integrace procedur
  - Specializace vede k menší expanzi kódu
  - Specializace se obtížněji řídí
  - Integrace odstraňuje režii volání
  - Integrace umožňuje další optimalizace

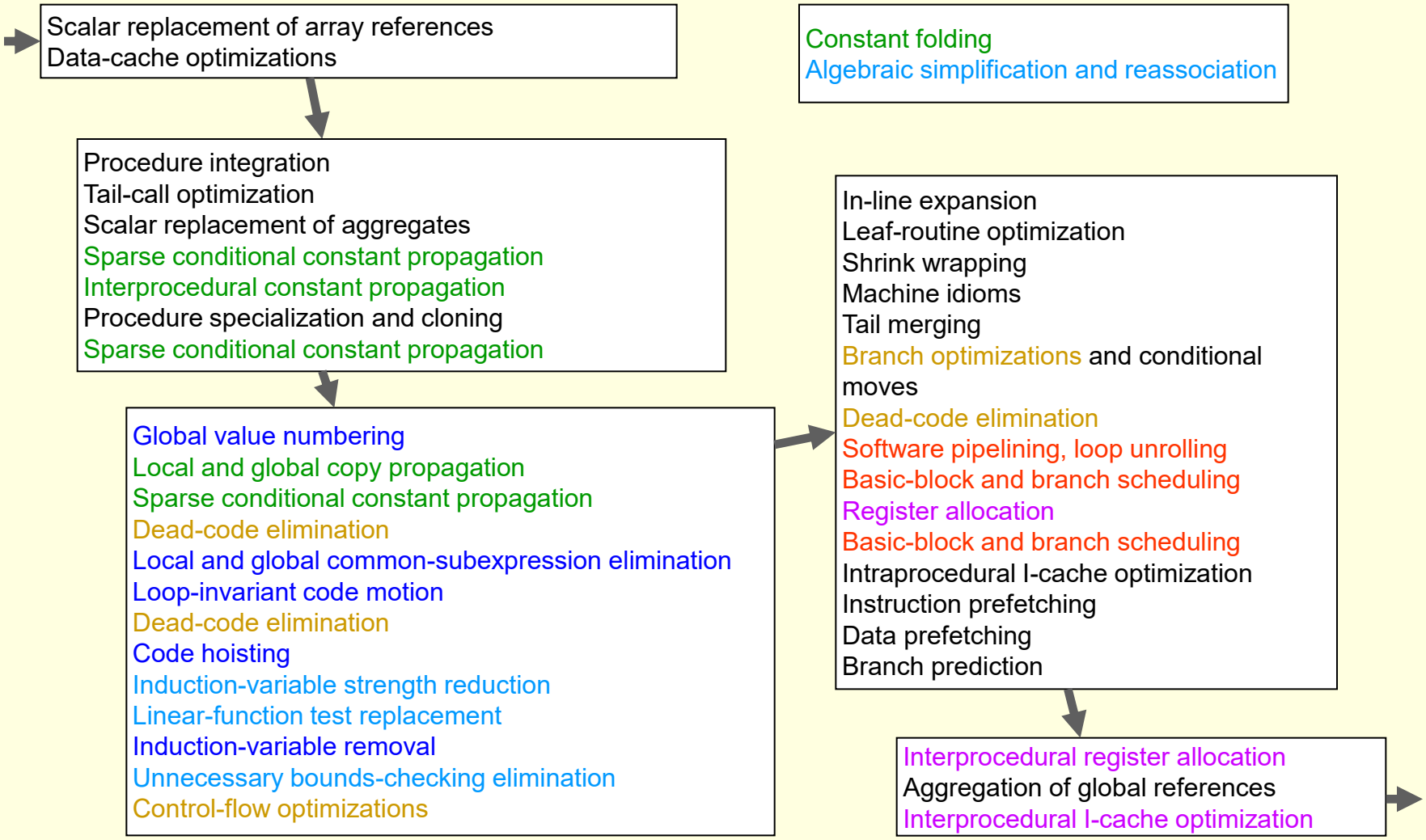
Nápověda pro procesor

- Intraprocedural I-cache optimization
  - Využití atomicity cache-line
  - Serializace BB tak, aby chování CPU vedlo k minimálnímu počtu výpadků I-cache
- Instruction prefetching
- Data prefetching
  - Využití speciálních instrukcí pro nedestruktivní čtení

- Branch prediction
  - Generování nápovědy pro branch prediction

Architektura překladače

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



LLVM 18.0 code generator (AMD64, -O3 -mno-sse)

- Target Transform Information
- Target Library Information
- Assumption Cache Tracker
- Target Pass Configuration
- Machine Module Information
- Type-Based Alias Analysis
- Scoped NoAlias Alias Analysis
- Profile summary info
- Create Garbage Collector Module Metadata
- Machine Branch Probability Analysis
- Default Regalloc Eviction Advisor
- Default Regalloc Priority Advisor
  - ModulePass Manager
    - FunctionPass Manager
      - Dominator Tree Construction
      - Basic Alias Analysis (stateless AA impl)
      - Function Alias Analysis Results
      - ObjC ARC contraction
    - Pre-ISel Intrinsic Lowering
    - FunctionPass Manager
      - Expand large div/rem
      - Expand large fp convert
      - Expand Atomic instructions
      - Lower AMX intrinsics
      - Lower AMX type for load/store
      - Dominator Tree Construction
      - Basic Alias Analysis (stateless AA impl)
      - Natural Loop Information
      - Canonicalize natural loops
      - Scalar Evolution Analysis
      - Loop Pass Manager
        - Canonicalize Freeze Instructions in Loops
        - Induction Variable Users
        - Loop Strength Reduction
      - Basic Alias Analysis (stateless AA impl)
      - Function Alias Analysis Results
      - Merge contiguous icmps into a memcmp
      - Natural Loop Information
      - Lazy Branch Probability Analysis
      - Lazy Block Frequency Analysis
      - Expand memcmp() to load/stores
      - Lower Garbage Collection Instructions
      - Shadow Stack GC Lowering
      - Lower constant intrinsics
      - Remove unreachable blocks from the CFG
      - Natural Loop Information
      - Post-Dominator Tree Construction
      - Branch Probability Analysis
      - Block Frequency Analysis
      - Constant Hoisting
      - Replace intrinsics with calls to vector library
      - Partially inline calls to library functions
      - Expand vector predication intrinsics
      - Scalarize Masked Memory Intrinsics
      - Expand reduction intrinsics
      - Natural Loop Information
      - TLS Variable Hoist
      - Interleaved Access Pass
      - X86 Partial Reduction
      - Expand indirectbr instructions
      - Natural Loop Information
      - CodeGen Prepare
      - Dominator Tree Construction
      - Exception handling preparation
      - Prepare callbr
      - Safe Stack instrumentation pass
      - Insert stack protectors
      - Basic Alias Analysis (stateless AA impl)
      - Function Alias Analysis Results
      - Natural Loop Information
      - Post-Dominator Tree Construction
      - Branch Probability Analysis
      - Assignment Tracking Analysis
      - Lazy Branch Probability Analysis
      - Lazy Block Frequency Analysis
      - X86 DAG->DAG Instruction Selection
      - MachineDominator Tree Construction
      - Local Dynamic TLS Access Clean-up
      - X86 PIC Global Base Reg Initialization
      - Argument Stack Rebase
      - Finalize ISel and expand pseudo-instructions
      - X86 Domain Reassignment Pass
      - Lazy Machine Block Frequency Analysis
      - Early Tail Duplication
      - Optimize machine instruction PHIs
      - Slot index numbering
      - Merge disjoint stack slots
      - Local Stack Slot Allocation
      - Remove dead machine instructions
      - MachineDominator Tree Construction
      - Machine Natural Loop Construction
      - Machine Trace Metrics
      - Early If-Conversion
      - Lazy Machine Block Frequency Analysis
      - Machine InstCombiner
      - X86 cmov Conversion
      - MachineDominator Tree Construction
      - Machine Natural Loop Construction
      - Machine Block Frequency Analysis
      - Early Machine Loop Invariant Code Motion
      - MachineDominator Tree Construction
      - Machine Block Frequency Analysis
      - Machine Common Subexpression Elimination
      - MachinePostDominator Tree Construction
      - Machine Cycle Info Analysis
      - Machine code sinking
      - Peephole Optimizations
      - Remove dead machine instructions
      - Live Range Shrink
      - X86 Fixup SetCC
      - Lazy Machine Block Frequency Analysis
      - X86 LEA Optimize
      - X86 Optimize Call Frame
      - X86 Avoid Store Forwarding Blocks
      - X86 speculative load hardening
      - MachineDominator Tree Construction
      - X86 EFLAGS copy lowering
      - X86 DynAlloca Expander
      - MachineDominator Tree Construction
      - Machine Natural Loop Construction
      - Tile Register Pre-configure
      - Detect Dead Lanes
      - Process Implicit Definitions
      - Remove unreachable machine basic blocks
      - Live Variable Analysis
      - Eliminate PHI nodes for register allocation
      - Two-Address instruction pass
      - Slot index numbering
      - Live Interval Analysis
      - Register Coalescer
      - Rename Disconnected Subregister Components
      - Machine Instruction Scheduler
      - Machine Block Frequency Analysis
      - Debug Variable Analysis
      - Live Stack Slot Analysis
      - Virtual Register Map
      - Live Register Matrix
      - Bundle Machine CFG Edges
      - Spill Code Placement Analysis
      - Lazy Machine Block Frequency Analysis
      - Machine Optimization Remark Emitter
      - Greedy Register Allocator
      - Tile Register Configure
      - Greedy Register Allocator
      - Virtual Register Rewriter
      - Register Allocation Pass Scoring
      - Stack Slot Coloring
      - Machine Copy Propagation Pass
      - Machine Loop Invariant Code Motion
      - X86 Lower Tile Copy
      - Bundle Machine CFG Edges
      - X86 FP Stackifier
      - MachineDominator Tree Construction
      - Machine Dominance Frontier Construction
      - X86 Load Value Injection (LVI) Load Hardening
      - Remove Redundant DEBUG_VALUE analysis
      - Fixup Statepoint Caller Saved
      - PostRA Machine Sink
      - Machine Block Frequency Analysis
      - MachinePostDominator Tree Construction
      - Lazy Machine Block Frequency Analysis
      - Machine Optimization Remark Emitter
      - Shrink Wrapping analysis
      - Prologue/Epilogue Insertion & Frame Finalization
      - Machine Late Instructions Cleanup Pass
      - Control Flow Optimizer
      - Lazy Machine Block Frequency Analysis
      - Tail Duplication
      - Machine Copy Propagation Pass
      - Post-RA pseudo instruction expansion pass
      - X86 pseudo instruction expansion pass
      - Insert KCFI indirect call checks
      - MachineDominator Tree Construction
      - Machine Natural Loop Construction
      - Post RA top-down list latency scheduler
      - Analyze Machine Code For Garbage Collection
      - Machine Block Frequency Analysis
      - MachinePostDominator Tree Construction
      - Branch Probability Basic Block Placement
      - Insert fentry calls
      - Insert XRay ops
      - Implement the 'patchable-function' attribute
      - ReachingDefAnalysis
      - X86 Execution Dependency Fix
      - BreakFalseDeps
      - X86 Indirect Branch Tracking
      - X86 vzeroupper inserter
      - MachineDominator Tree Construction
      - Machine Natural Loop Construction
      - Lazy Machine Block Frequency Analysis
      - X86 Byte/Word Instruction Fixup
      - Lazy Machine Block Frequency Analysis
      - X86 Atom pad short functions
      - X86 LEA Fixup
      - X86 Fixup Inst Tuning
      - X86 Fixup Vector Constants
      - Compressing EVEX instrs when possible
      - X86 Discriminate Memory Operands
      - X86 Insert Cache Prefetches
      - X86 insert wait instruction
      - Contiguously Lay Out Funclets
      - StackMap Liveness Analysis
      - Live DEBUG_VALUE analysis
      - Machine Sanitizer Binary Metadata
      - Lazy Machine Block Frequency Analysis
      - Machine Optimization Remark Emitter
      - Stack Frame Layout Analysis
      - X86 Speculative Execution Side Effect Suppression
      - X86 Indirect Thunks
      - X86 Return Thunks
      - Check CFA info and insert CFI instructions if needed
      - X86 Load Value Injection (LVI) Ret-Hardening
      - Pseudo Probe Inserter
      - Unpack machine instruction bundles
      - Lazy Machine Block Frequency Analysis
      - Machine Optimization Remark Emitter
      - X86 Assembly Printer
      - Free MachineFunction