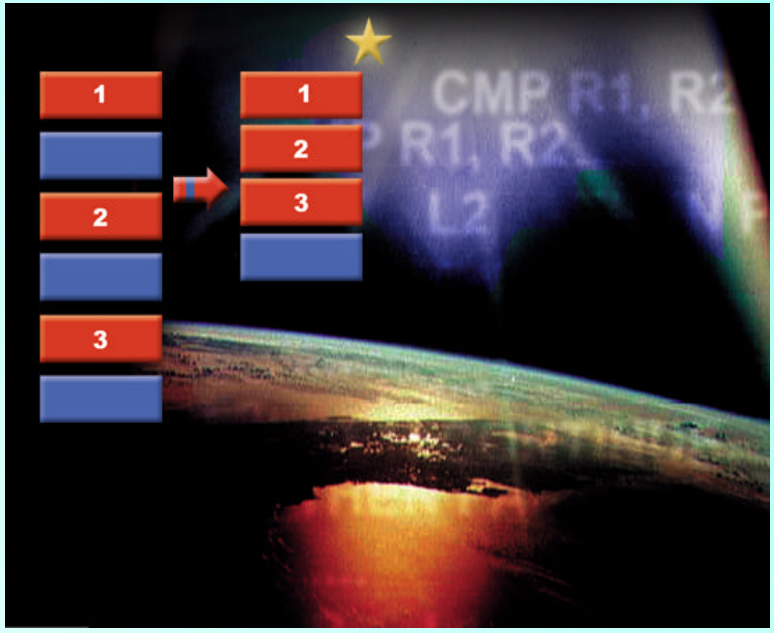
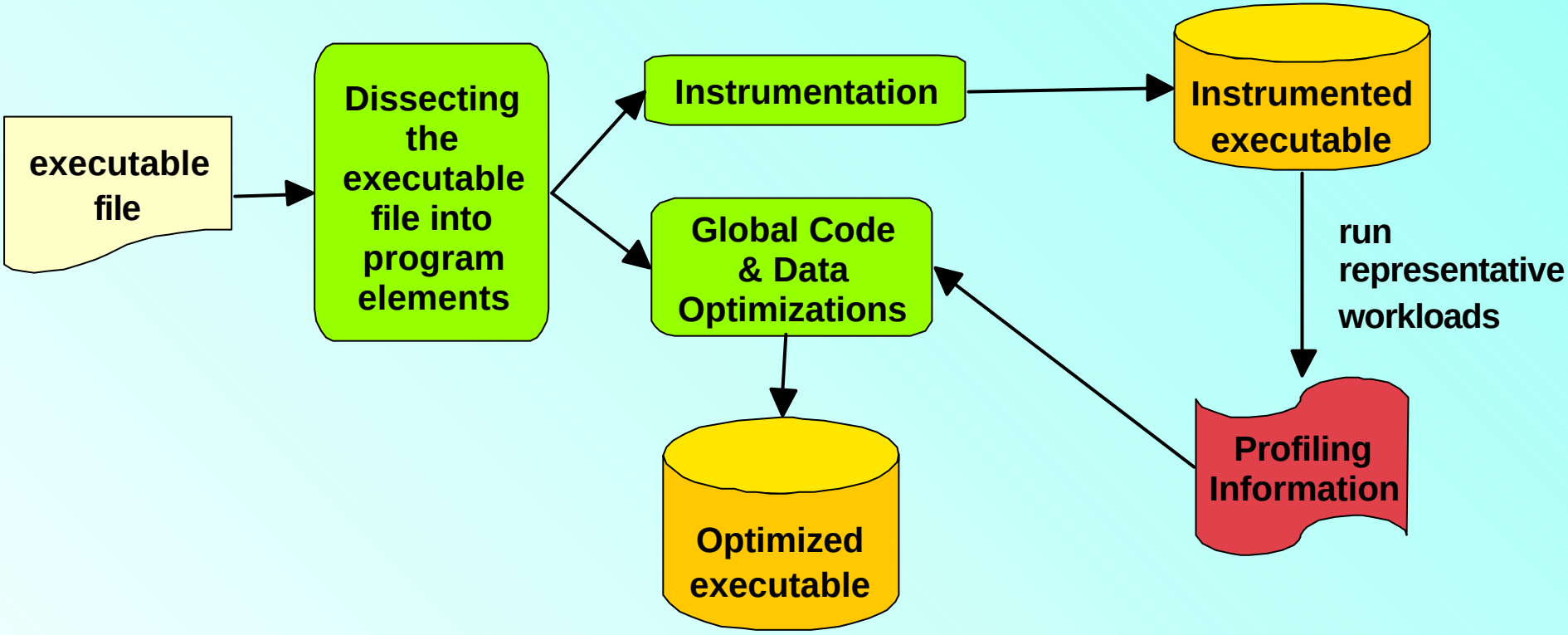
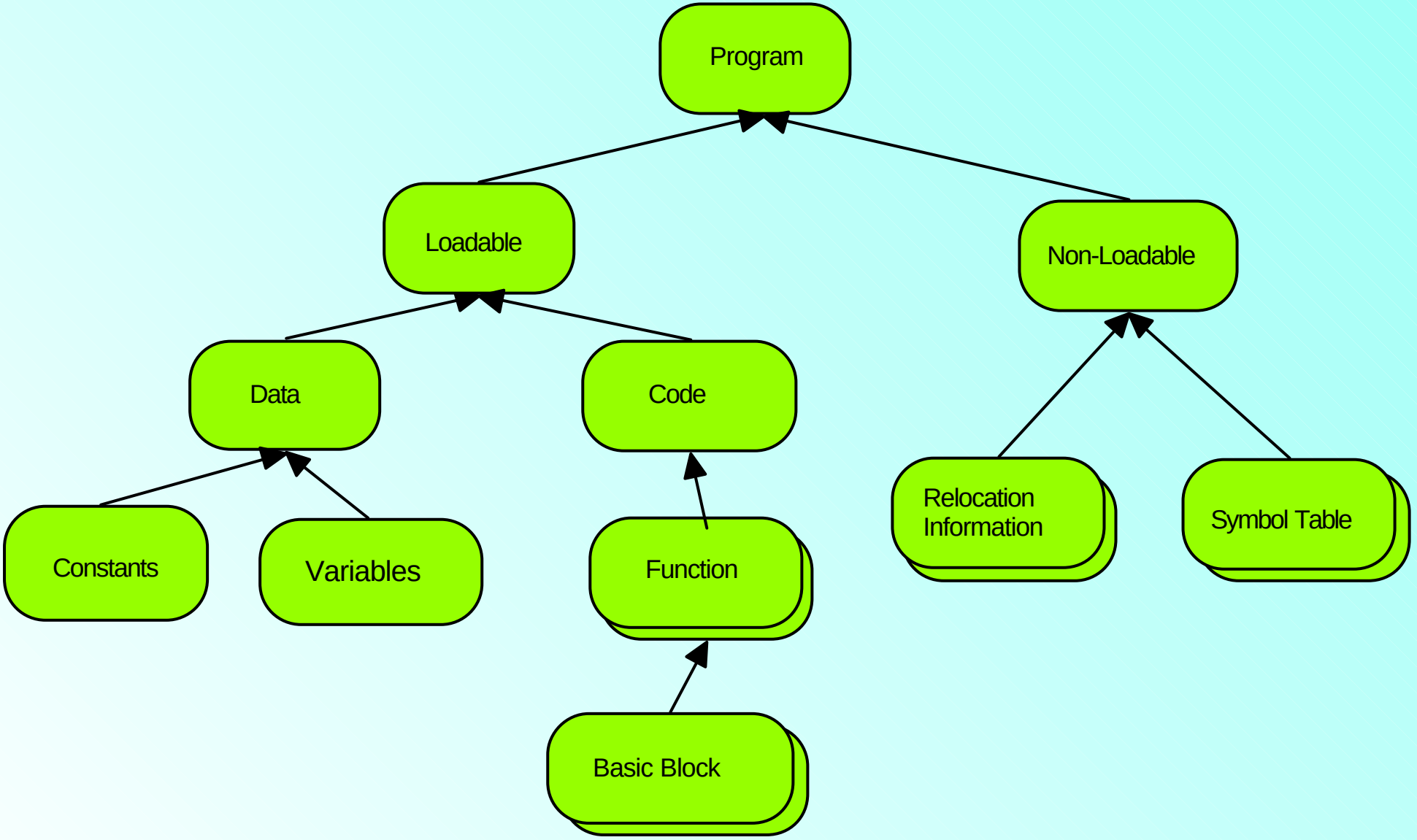
Post-Link Optimization Technology



Method


executable file
Dissecting the executable file into program elements
Instrumentation
Instrumented executable
Global Code & Data Optimizations
run representative workloads
Profiling Information
Optimized executable

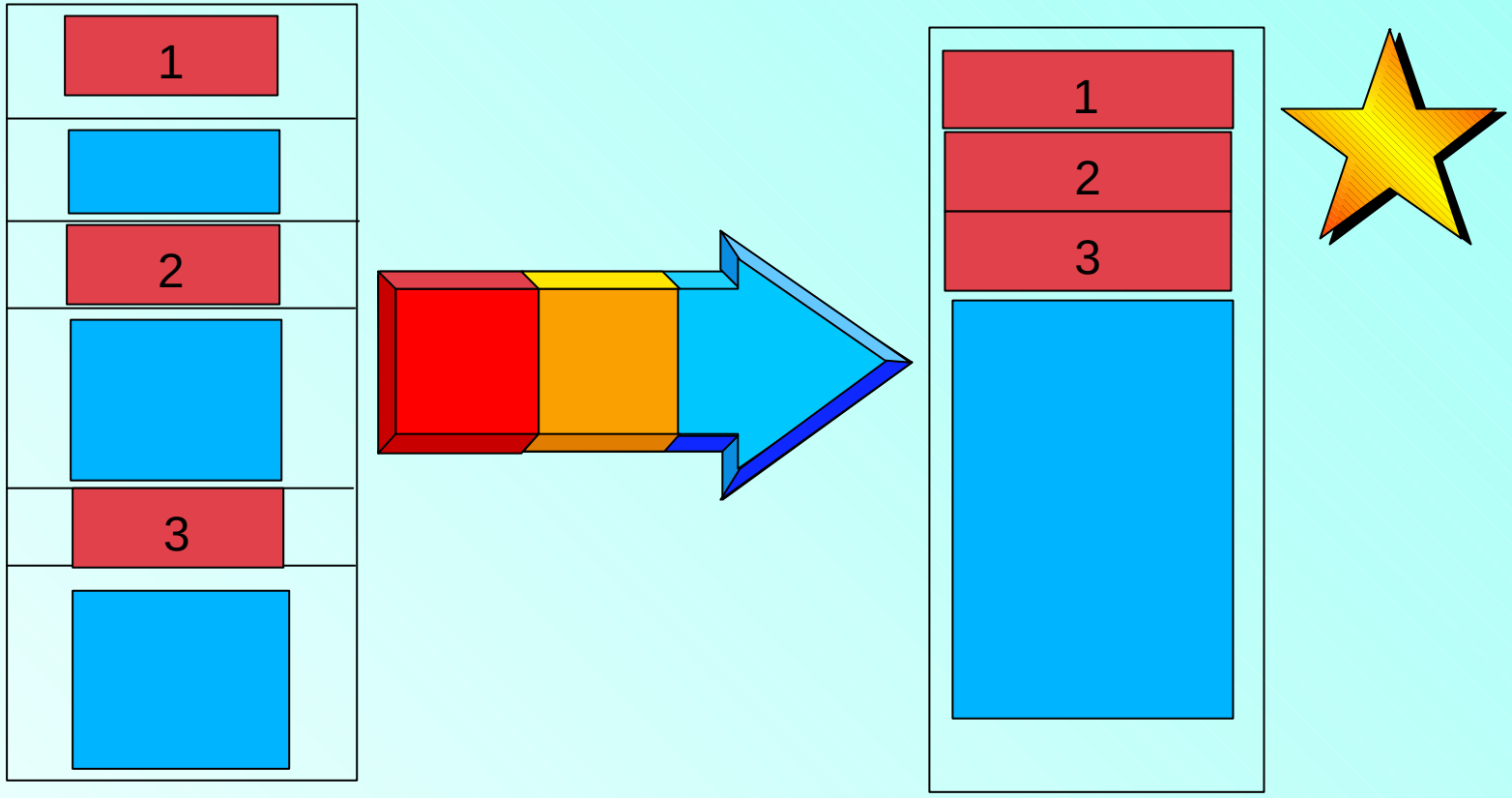
Executable Analysis



FDPR

- FDPR Properties
  - Using a global view of the entire program
  - Operating on the executable file after linkage
- These properties enable FDPR to do
  - Global Code Reordering
  - Inter Procedure Boundaries Optimizations
  - Static Data Rearrangement
  - Constant Area Rearrangement
- Resulting optimization opportunities
  - Usage of Branch Tables
  - Usage of TOC load instructions

Global Code and Data Reordering



Code Reordering:

C Program

```
if (x==y)
  {
    /* Error code */
   }
/* Continue code */
```

Pseudo-assembly

```
      CMP R1, R2
      BF    L1
      ....
      THEN PART
      ....
L1: CONTINUE
```


Code Reordering (cont.)

Original code

```
CMP R1, R2
BF    L1
 ....
THEN PART
 ....
L1: CONTINUE
 ....
```

Improved code

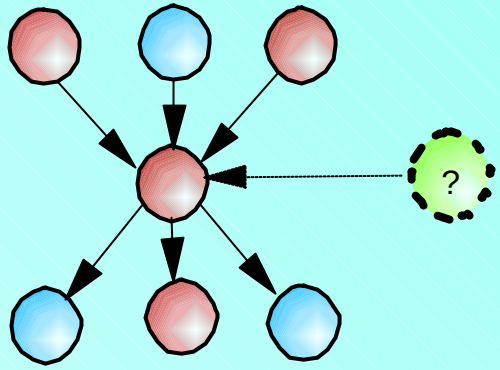
```
CMP R1, R2
BT    L2
L1: CONTINUE
 ....
L2:  ....
THEN PART
 ....
B    L1
```

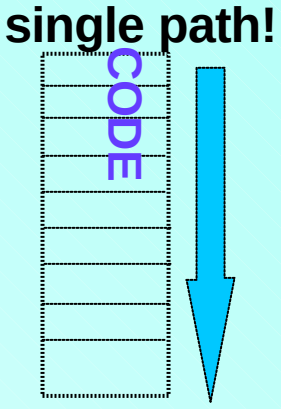
Improving:

1. **Cache ratio - ("Hot" code grouped together)**
2. **Branch Penalty - (the "BT L2" is rarely taken)**

Procedure Boundaries Optimizations

- Based on feedback information
- Light Weight Approach
  - Based on local information
    - immediate callers
    - call sites
    - immediate callees
  - Scalability
    - short completion time
    - single path
    - simple data structures





Killed Register Optimization

```
        call foo
        ...
        call foo
        R27 <- 7
        ...
foo:
        save R27
        ...
        add R27,3
        ...
```

```
        call foo
        ...
        call foo_opt
        R27 <- 7
        ...
foo_opt:
        ...
        add R27,3

foo:
        save R27
        call foo_opt
        restore R27
```


Enable the Optimizations via Renaming

```
        call foo
        ...
        call foo
        R27 <- 7
        ...
foo:
        save R28
        ...
        add R28,3
        ...
```

```
        call foo
        ...
        call foo
        R27 <- 7
        ...
foo:
        save R27
        ...
        add R27,3
        ...
```


FDPR Supported Environments

- AIX/Power
  - Became part of the AIX OS starting AIX 5.2
- WIN/IA32
  - Product level tool for IBM internal use
- Linux/IA32
  - Prototype level tool
- Linux/Power
  - Prototype is in development

FDPR Optimizations Today

- Default Optimizations
  - Code reordering
  - Branch folding
  - Branch to branch elimination
  - Branch prediction setting
  - Inlining returning functions
  - Code alignment
- Optimization flags

-nop	Eliminate nop instructions from reordered code.
-opt_fdpr_glue	Perform Jump folding in FDPR Glue code.
-inline	Inline Hot functions (code duplication+mflr/mtlr)
-i_resched	Reschedule instructions after code reordering.
-RD	Perform Static Data reordering.
-tocload	Perform optimization for toc load instructions.
-aggressive_tocload	Perform tocload opt. + reduce the TOC size
-ptrgl_opt	Optimize indirect call instructions via function's descriptors
-dcbt_opt	Insert data prefetch instructions
-volatile_regs	Eliminate stores/restore using non-used volatile registers.
-killed_regs	Eliminate stores/restores of "killed" registers.
-regs_release	Perform cold non-volatile register opt.
-propagate_regs	Propagate stores from hot prologs to colder ones.
-regs_redo	Eliminate stores or re-constructable registers' values.
-full_saved_regs_calls	Extend -killed_regs & -propagate to func epilogs.