The Theory, History and Future of System Linkers

Luba Tang
CEO & Founder, Skymizer Inc.

Outline

- The History
  - Target Independent Linkers
  - Post Optimizers
  - Instrumentation Tools
- The Theory
  - Linking Language
  - Fragment-reference graph
- The Future
  - for GPGPU; for virtual machines
  - The bold project

唐文力 Luba Tang

CEO & Founder of Skymizer Inc.

Architect of MCLinker and GYM compiler

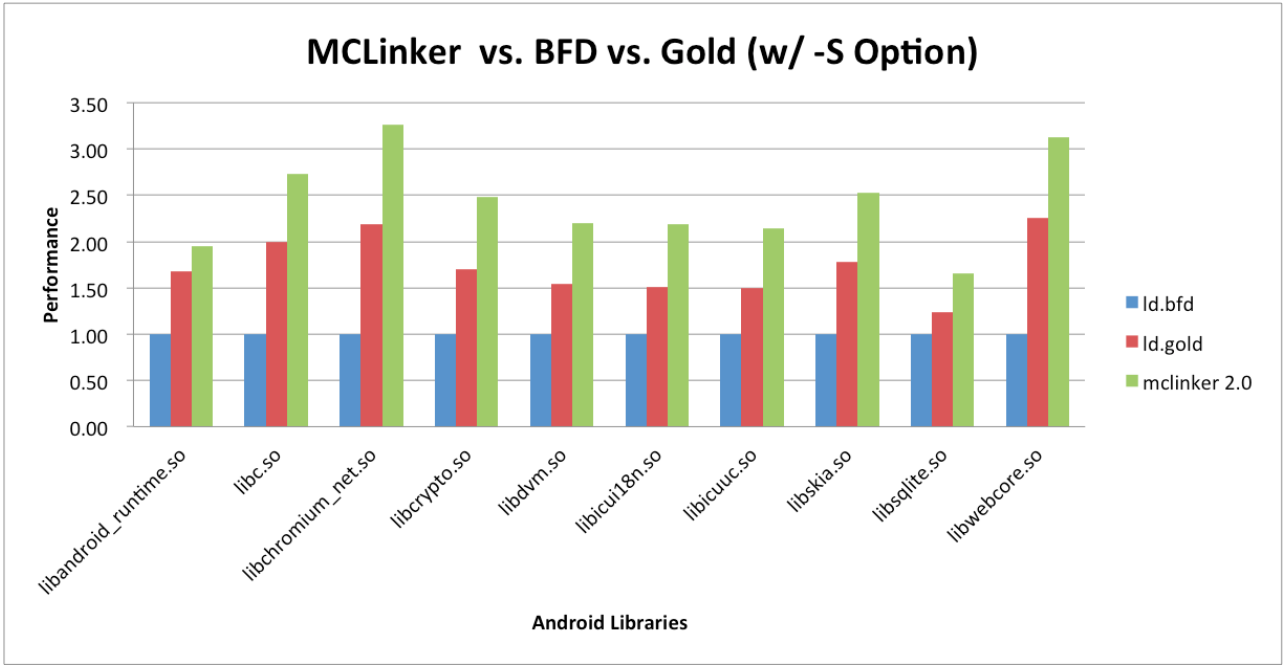
Compiler and Linker/Electronic System Level Design

Linker: The Elephant in the Room

- System linkers are very complicated. Only a few team can make a full-fledge system linker.
  - There are only four open source linkers that can be said full-fledge.
    - GNU ld, Google gold can link Linux kernel
    - Apple ld64 can link Mac OS X and iOS
    - MCLinker can link BSD and Android system
- ELF linkers are super complicated. There are many undocumented behaviors and target-specific behaviors.
  - The other linkers are developed for more than three years and can not be released. The linking problem is intricate.
- Although a lot of researches have proven linker itself can optimize programs at a high performance level, developers still not get benefit from these researches.

No Linker Really Optimize Programs

- MCLinker is 35% faster than the Google gold, and the Google gold is ~200% faster than GNU ld
- If we turn on optimization flags, the output quality is almost identical to all linkers (<3 %)

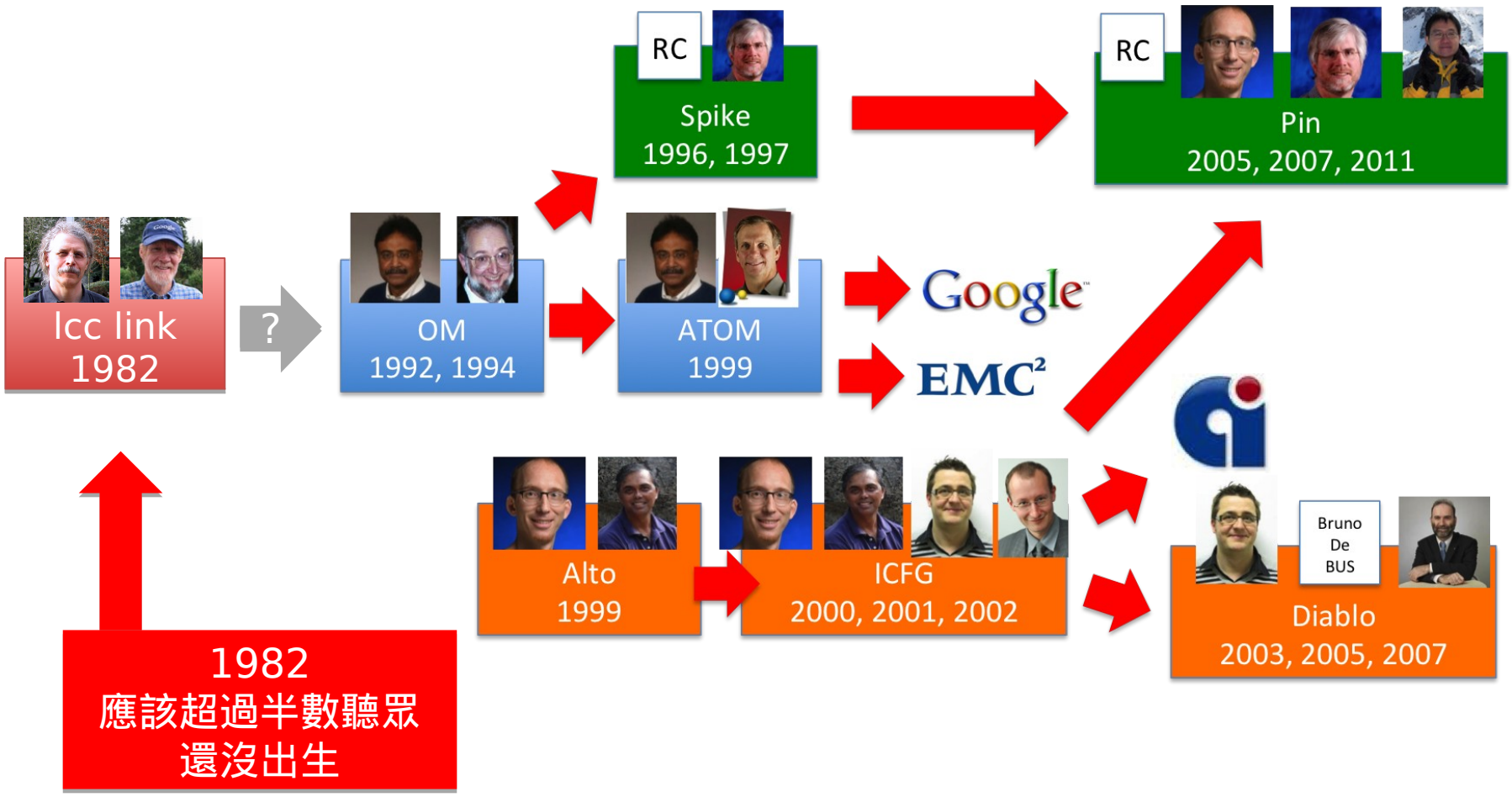


Comparison of ELF Linkers

	GNU ld	Google gold	MCLinker
License	GPLv3 Cannot be adopted by Android		UIUC BSD-Style
Target Platform	All Linux mainstream devices	ARM, X86, X86_64, (Mips, SPARC)	All Andro ices. AR ips64
Object Format	COFF, a.out, ELF		extensible
Line of Code	500+K		50+K
Performance			Fastest Steadily x2 than GNU ld, x1.3 than Google gold
Intermediate Representation	The BFD library for reference graph	None	Command line language and reference graph

People Keep Eyes on **Target Independent Linkers**

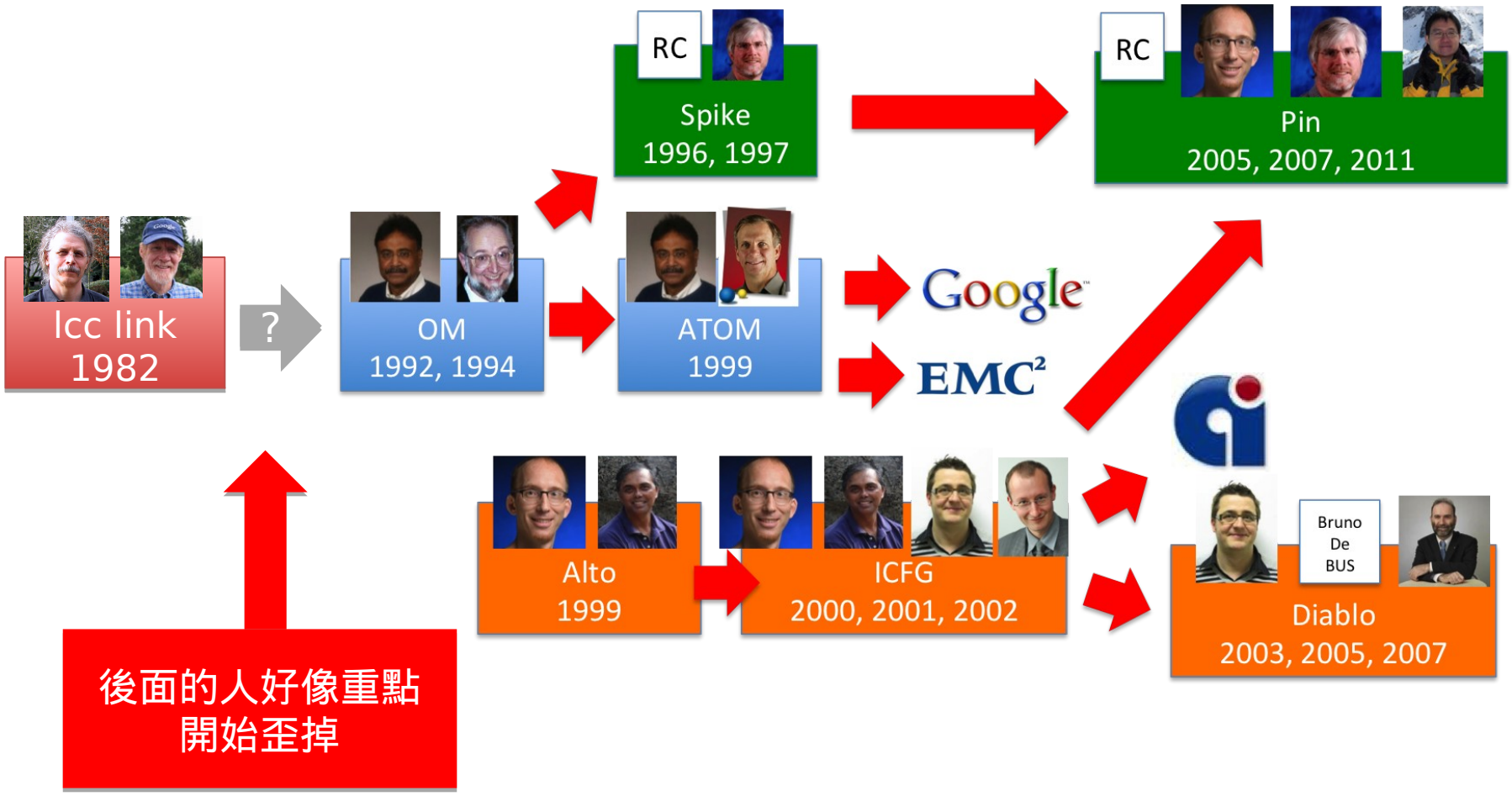
The Most-Recently Important Target Independent Linker Research



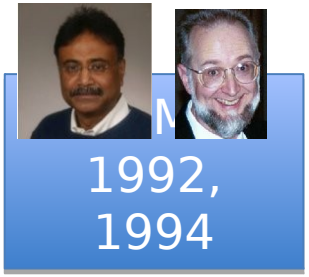
LINK: A Machine-Independent Linker

- Team
  - Christopher W. Fraser
  - David R. Hanson
- 1982, Software Practice and Experience
  - Define linker and object language (the predecessor of linker script)
  - Define three basic rules
    - Define the condition of resolution
    - Define the condition of absolute objects
    - Define when to pull in a library

Linker; Post Optimizer; Instrumentation



OM: Code Optimization at Link-Time System



- Team
  - Amitabh Srivastava
  - David W. Wall
- 1992 Technical Report
  - An approach to transform binary into RTL
  - Use RTL to do inter-procedural optimization (5%~14%, SPEC)
    - Dead code elimination
    - Loop Invariant Code Motion (LICM)
- 1994 SIGPLAN (3.8%, SPEC)
  - Replace load instruction and eliminate GAT
  - Reduce code size by 10% or more

OM: Code Optimization at Link-Time System

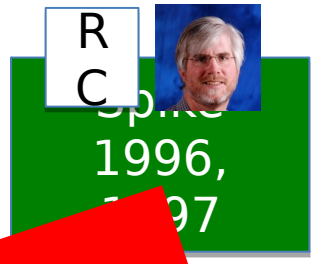
- Key Contributions of OM are
  - OM identifies the problems to translate binary back to assembly.
    - PC-relative branches only
    - Convert jump table back to case-statement
    - No delayed branch, no delay slot

1992, 1994

退休 Ya!

Spike: A successor or a competitor of OM

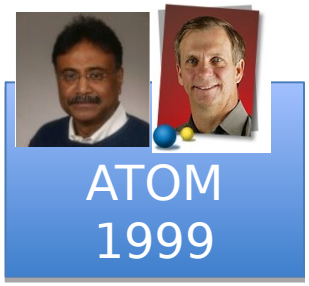
- DEC Team
  - Robert Cohn
  - David W. Goodwin
  - P. Geoffrey Lowney
- 1996 Micro 29 (T… …nselves another …
  - Hot … …timization to use shorter jump
  - Works on Windows/NT Digital Alpha 3~8% improvement



Finally, someone find some optimization that **can not be done in compiler**

ATOM: Analysis Tools with OM (Best of PLDI 1979-1999)

- Dream Team - 1999
  - Amitabh Srivastava (President of EMC)
  - Alan Eustace (Senior VP of Google Search)



Google

EMC²

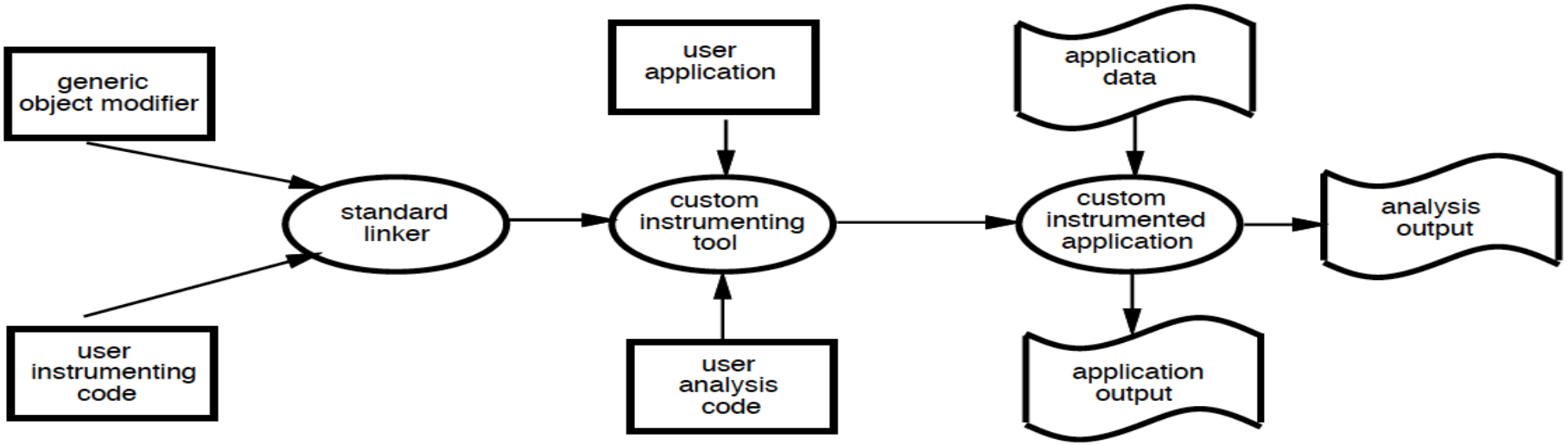
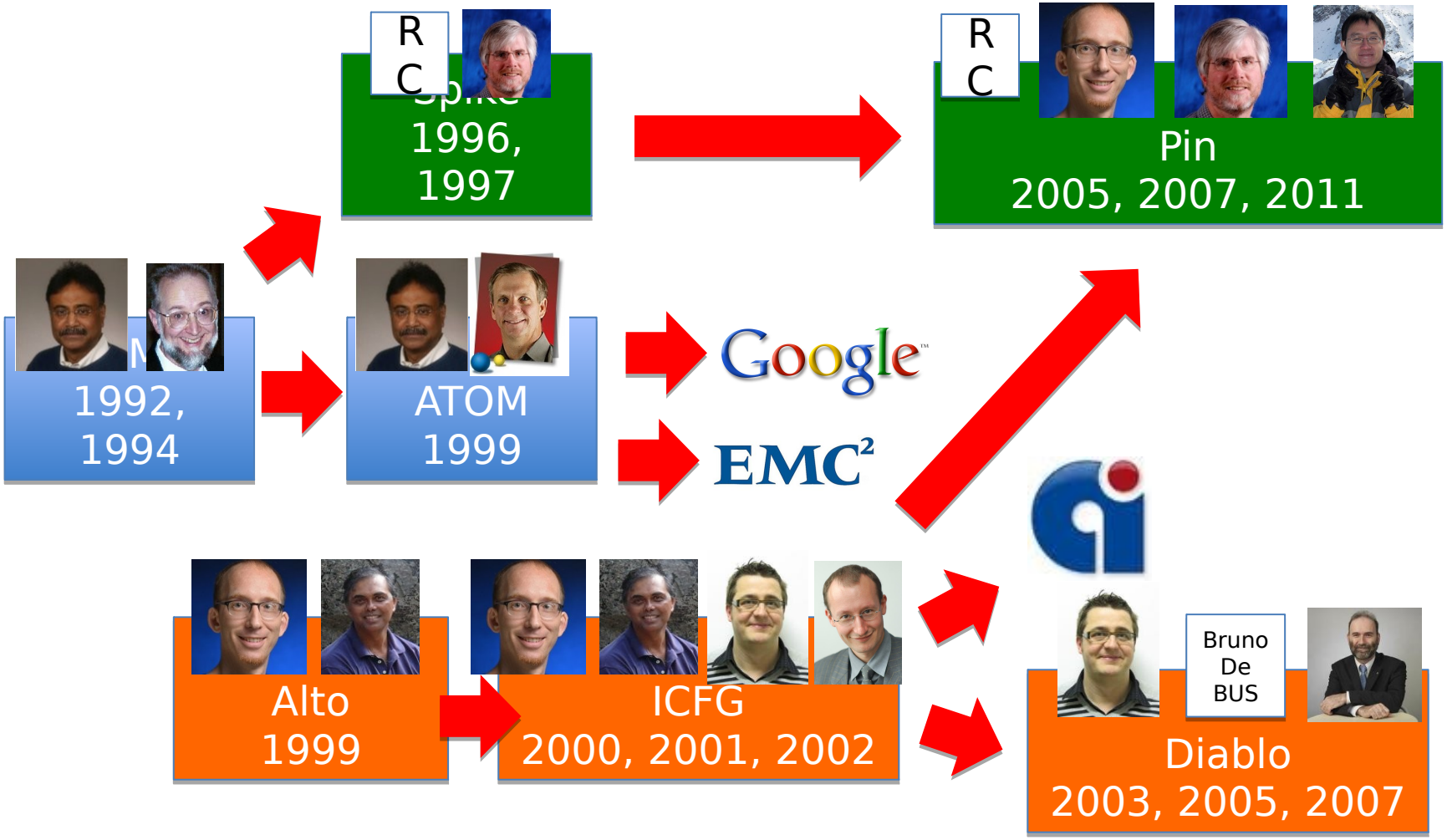


Figure 1: The ATOM Process

ATOM: Analysis Tools with OM (Best of PLDI 1979-1999)

- Key Contributions of ATOM are
  - ATOM defines the use scenario and APIs of an instrumentation tool
  - Intel Pin follows APIs of ATOM.
- The rest contributions:
  - Reducing procedure call overhead (caller-save and callee-save)
  - Use virtual machine to instrument program
    - Defines the necessary memory layout

Chronicle of Linker Optimization



Alto: A Link-Time Optimizer for the Compaq Alpha

- Team
  - Robert Muth
  - Saumya Debray
  - Scott Watterson
  - Keo De Bosschere
- Convert binary into control flow graph
  - General approach
  - The inspirer of ICFG

Alto: A Link-Time Optimizer for the Compaq Alpha

- Powerful Analysis and Optimization
  - Simplification
    - Dead code elimination
    - Normalize operations who express the same semantics
    - Use nops instead of remove instructions directly
  - Analysis
    - Machine level idioms for control flow
    - Live analysis (register, memory)
  - Optimization
    - Constant propagation (remove load, 6.4%)
    - Dead code elimination
    - Unused memory elimination (remove load, speed up 5.7%)
    - Low level inlining (10% on average)
    - Profile-directed code layout (6.5%)
    - Instruction scheduling

There are a lot of performance hidden in linkers

ICFG: Interprocedural Control Flow Graph

- Team
  - Saumya Debray
  - William Evans
  - Robert Muth
  - Daniel Kastner
  - Bjorn De Sutter
  - Koen De Boss
- ACM Tr… …ming Languages and Systems
  - Defines ICFG
  - Collect compiler techniques for code compaction
  - Reduce 30% on the average

Post optimizer - Not a functional linker

Diablo: Post-Pass Optimization

- Team, Collection of Euro
  - Bruno De Bus
  - Saumya Debray
  - William Evans
  - Robert Muth
  - Daniel Kastner
  - Ludo Van Put
  - Bjorn De Sutter
  - Koen De Bosschere

- First complete post-pass optimizer
  - A lot of following researches

Diablo: Post-Pass Optimization

- For code size, C++ have more opportunity than C
  - Sifting out the Mud: Low Level C++ Code Reuse, OOPSLA'02
    - Reduce 27~70%, 43% on aver
  - Combining Global Co paction, LCTES'01
    - Redu % faster
- CFG re tion becomes mature
  - Generic Control Flow reconstruction from Assembly Code, LCTES'02
  - Can handle delay slots and restricted indirection

Still Post optimizer - Not a functional linker
Limited in static object files

Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation

- Team, Collection of USA, Intel
  - Chi-Keung Luk
  - Robert Cohn
  - Robert Muth
  - Harish Patil
  - Artur Klauser
  - Geoff Lowney
  - Steven Wallace
  - Vijay Janapa Reddi
  - Kim Hazelwood
- Pin release the power of program analysis
  - 1608 citation since 2005
  - Heavily cited in GPGPU and HSA area

Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation

State-of-Art instrumentation tool

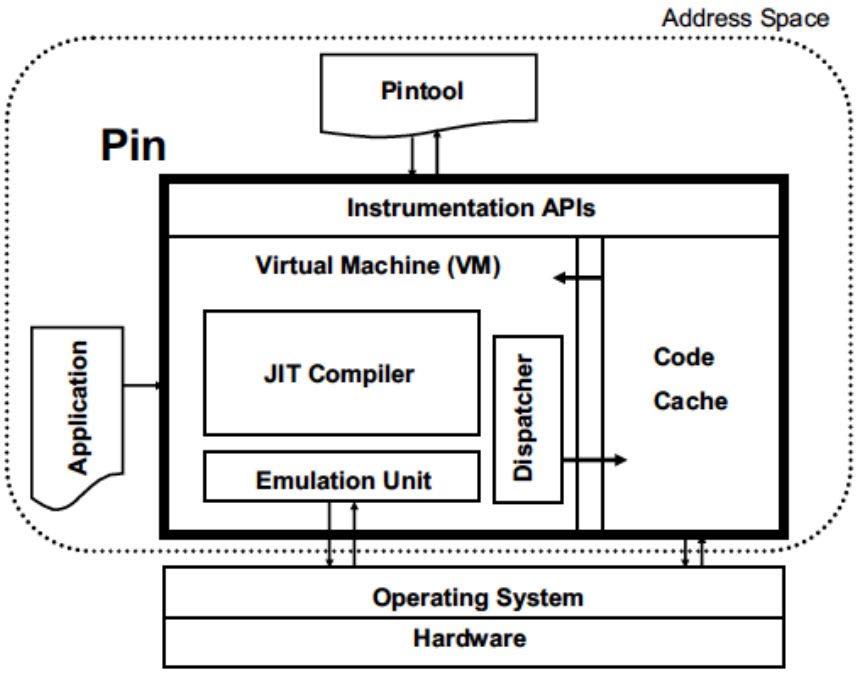


Figure 2. Pin's software architecture

Pin Provides ATOM-like APIs

- User can write his own instrument and analysis code

```
FILE * trace;

// Print a memory write record
VOID RecordMemWrite(VOID * ip, VOID * addr, UINT32 size) {
    fprintf(trace,"%p: W %p %d\n", ip, addr, size);
}

// Called for every instruction
VOID Instruction(INS ins, VOID *v) {
    // instruments writes using a predicated call,
    // i.e. the call happens iff the store is
    // actually executed
    if (INS_IsMemoryWrite(ins))
        INS_InsertPredicatedCall(
            ins, IPOINT_BEFORE, AFUNPTR(RecordMemWrite),
            IARG_INST_PTR, IARG_MEMORYWRITE_EA,
            IARG_MEMORYWRITE_SIZE, IARG_END);
}

int main(int argc, char *argv[]) {
    PIN_Init(argc, argv);
    trace = fopen("atrace.out", "w");
    INS_AddInstrumentFunction(Instruction, 0);
    PIN_StartProgram(); // Never returns
    return 0;
}
```


This is NOT linkel
Warner Bros

Linker: The Elephant in the Room

- Although a lot of researches have proven linker itself can optimize programs at a high performance level, developers still not get benefit from these researches.

Outline

- The History
  - Target Independent Linkers
  - Post Optimizers
  - Instrumentation Tools
- The Theory
  - Linking Language
  - Fragment-reference graph
- The Future
  - for GPGPU; for virtual machines
  - The bold project

唐文力 Luba Tang

CEO & Founder of Skymizer Inc.

Architect of MCLinker and GYM compiler

Compiler and Linker/Electronic System Level Design

Introduction to Linker Intermediate Representation

- MCLinker is the first *ELF linker to provide an intermediate representation (IR) for efficient transformation and analysis
- MCLinker provides IR on two levels
  - Linker Command Line Language
  - Fragment-Reference Graph
- Fragment is the basic linking unit, it can be
  - A section (coarse granularity)
  - A block of code or instructions (middle granularity)
  - An individual symbol and its code/data (fine granularity)
- MCLinker can trade linking time for the output quality.
  - The finer granularity,
    - Fast, smaller program
    - Longer link time

* Nick Kledzik invents the Atom IR in ld64 for MachO. ld64 inspires MCLinker IRs

The Linker Command Line Language

- Linker's command line options is a kind of language
  - The meaning of a option depends on
    - their positions
    - the other potions
  - Some options have its own grammar

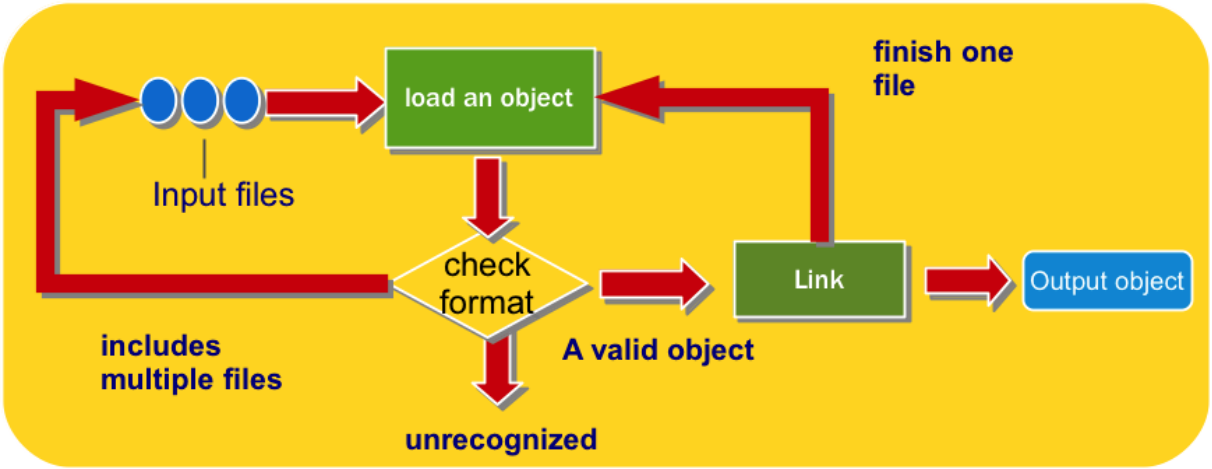
- Four categories of the options
  - Input files
  - Attributes of the input files
  - Linker script options
  - General options

- Examples

```
ld /tmp/xxx.o -lpthread
ld --as-needed ./yyy.so
ld --defsym=cgo13=0x224
ld -L/opt/lib -T ./my.x
```

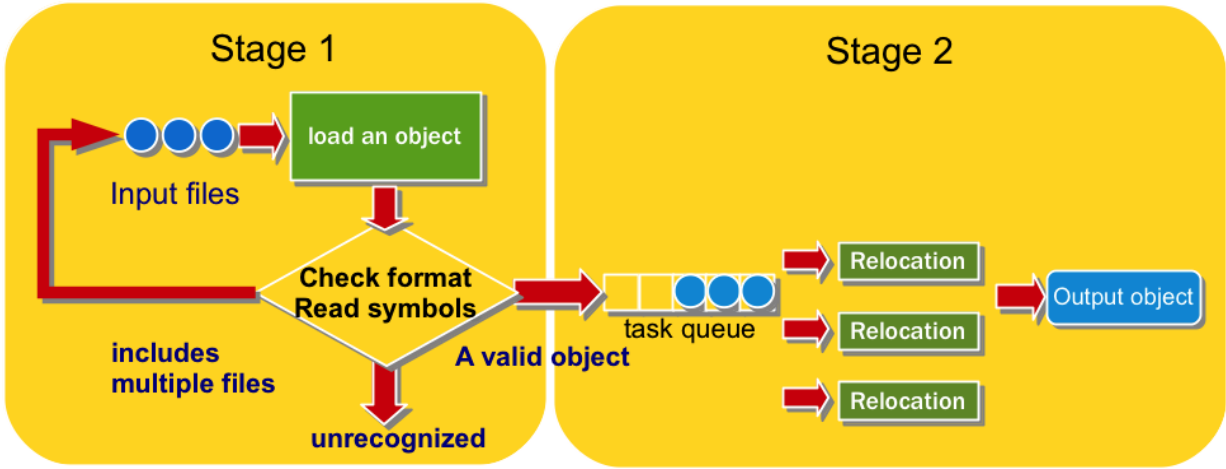
The GNU ld Linker

- The GNU ld linker is an interpreter of the command line language
  - Processing is recursive.
  - No clear separation between individual steps
  - Binary File Descriptor (BFD) is the only IR



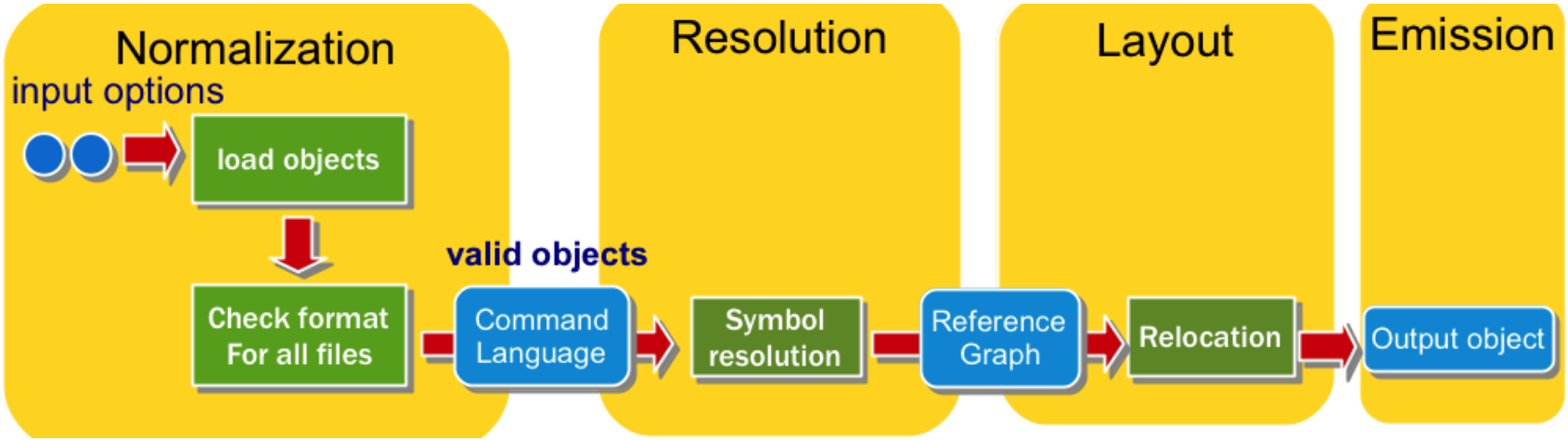
The Google gold Linker

- The Google gold linker separates linking into two stages
  - Symbol resolution
  - Relocation of instructions and data
- Although it has separated the linking processes, it does not provide reusable IR for optimization and analysis
- The Google gold linker illustrates an efficient linking algorithm
  - It's x2 faster than the GNU ld linker
  - Support multiple threads. Appropriate to cloud computing



MCLinker

- MCLinker separates the linking into four distinct stages
  - Normalization – parse the command line language
  - Resolution – resolve symbols
  - Layout – relocate instructions and data
  - Emission – emit file by various formats
- MCLinker provides two level intermediate representation (IR)
  - The command line language level
  - The reference graph level



Input Files on The Command Line

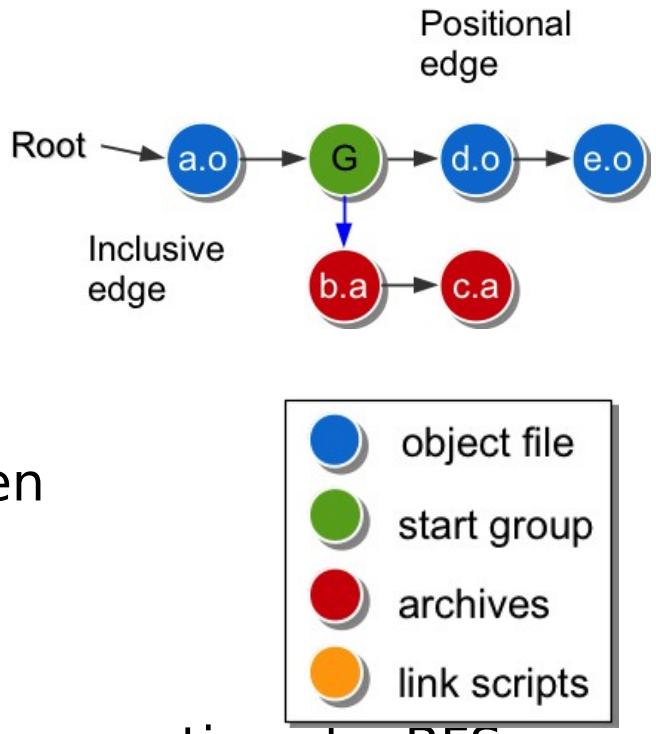
- An input file can be an object file, an archive, or a linker script
- Some input files can be defined multiple times
- The result of linking depends on the positions of inputs on the command line.
  - *Weak symbols* are first-come-first-served
  - COMDAT sections are first-come-first-served
- Two semantics to read input files
  - INPUT(file1, file2, file3, ...)
  - GROUP(archive1, archive2, archive3, ...)
- Archives in a group are searched *repeatedly* until no new undefined references are created

```
$ ld  a.o –start-group b.a c.a –end-group d.o e.o
```

The Input File Tree

- We can represent the input files on the command line by a tree structure
  - Vertices describes input files and groups on the command line
    - Object files
    - Archives
    - Linker scripts
    - Entrances of groups
- Edges describe the relationships between vertices
  - Positional edges
  - Inclusive edges
- Linkers resolve symbols by DFS and merge sections by BFS
- Example

  $ ld a.o –start-group b.a c.a –end-group d.o e.o



Attributes of Input Files

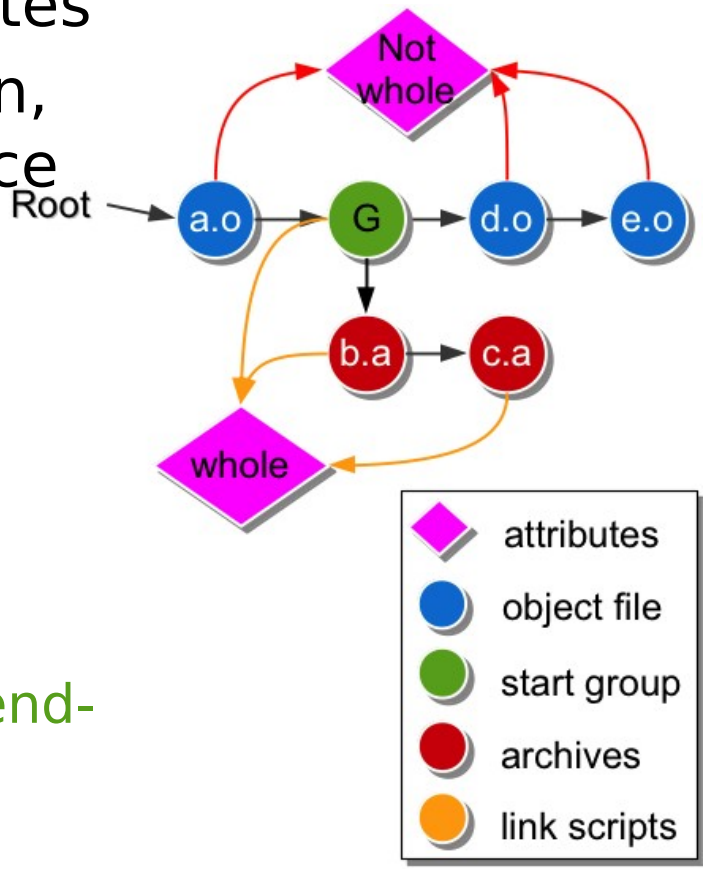
- Attributes change the way that a linker handles the input files
- Attributes affect the input files after the attribute options

Functions	Options	Meanings
Whole archives	--whole-archive	Includes every file in the archive
Link against dynamic libraries	-Bdynamic	Search shared libraries for -l option
As needed	--as-needed	Only add the necessary shared libraries to resolve symbols
Input format	--format=	The format of the following input files

Attributes in The Input File Tree

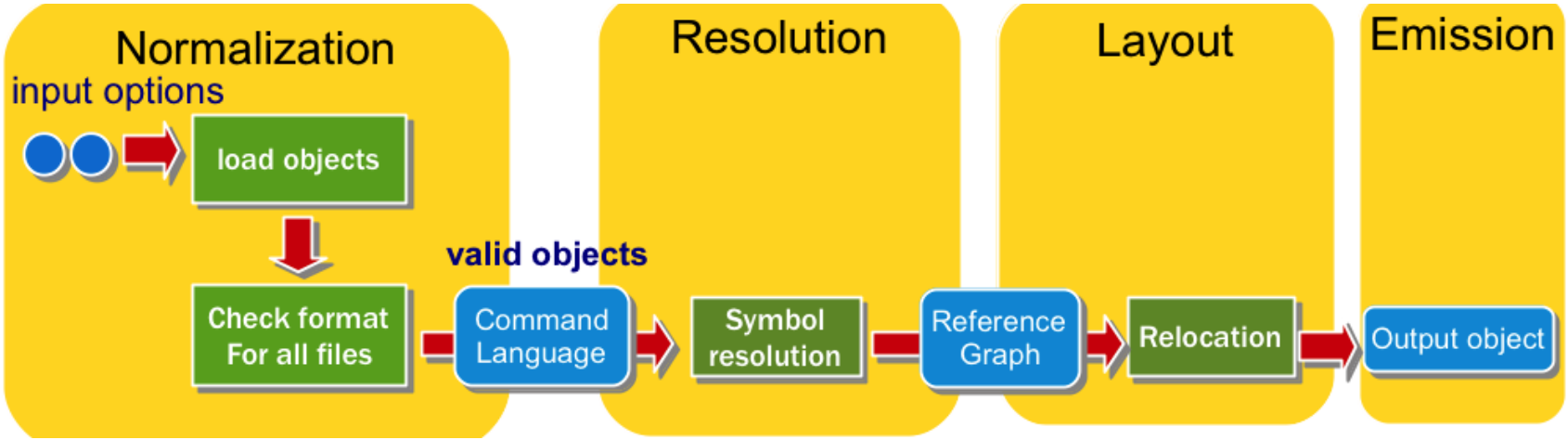
- Every input has a set of attributes
- In the MCLinker implementation, we give every vertex a reference to its attribute set
- If two vertices have identical attributes, they can share a common attribute set.
- Example

```
$ld ./a.o  --whole-archive
           --start-group ./b.a ./c.a --end-group
           --no-whole-archive
           ./d.o ./e.o
```



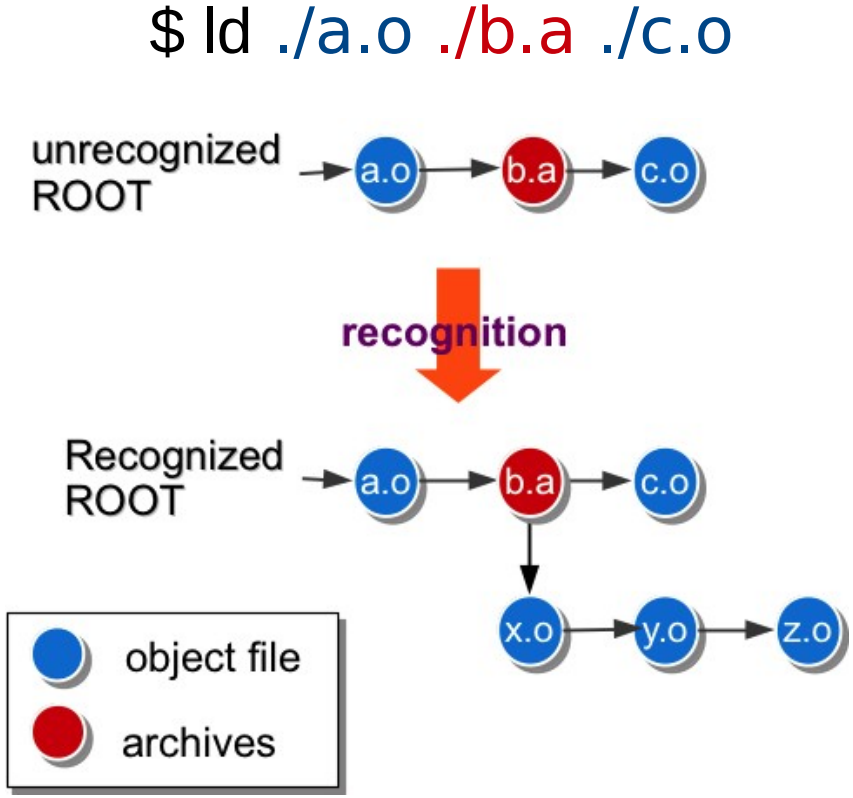
Normalization

- Transform the command line language into the input file tree
  - Parse command line options
  - Recognize input files to build up sub-trees
  - Merge all sub-trees to a form the input file tree



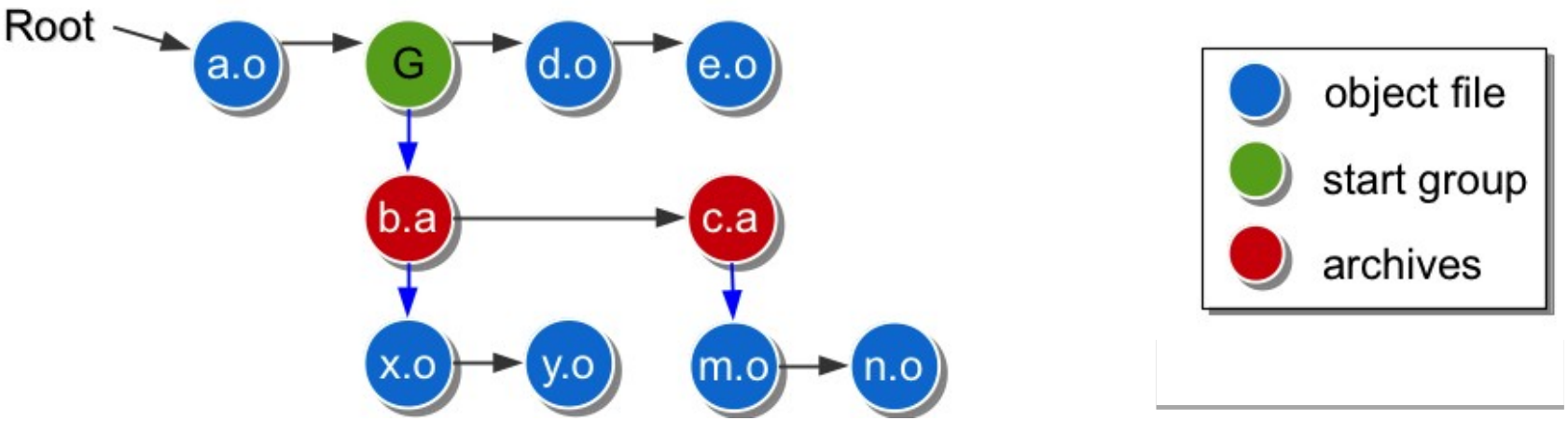
Steps of Normalization

- Step of normalization
  1. Parse the command line options
  2. Recognize archives and linker scripts
  3. Read the linker scripts and archives to create sub-trees
  4. Merge all sub-trees



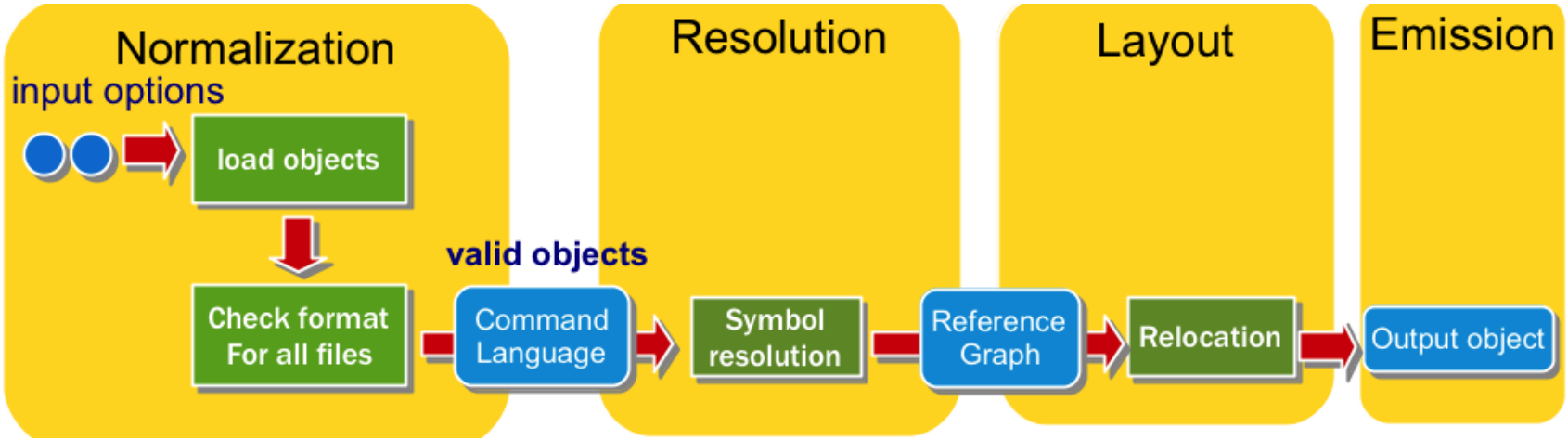
Traverse the Input File Tree

- MCLinker provides different iterators for different purposes
  - For symbol resolution
    - Depth first search for correctness
  - For section merging
    - Breadth first search for cache locality of the output file



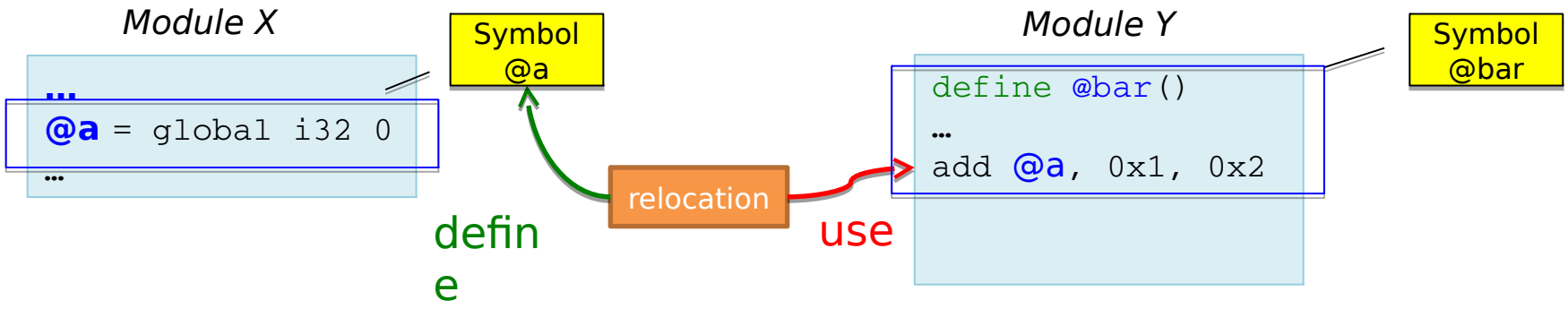
Resolution

- Transform the input file tree into the reference graph
  - Resolves symbols
  - Reads relocation
  - Builds the reference graph



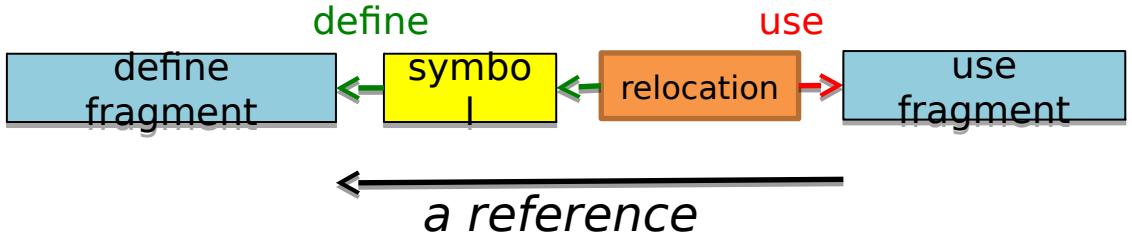
Symbols and Relocations

- A **fragment** is a block of instruction code or data in a module
  - A fragment may be
    - a function,
    - a label (Basic block),
    - a 32-bit integer data, and so on.
- A defined symbol indicates a fragment
- A relocation represents an use-define relationship between two fragments



Fragment-Reference Graph (1/2)

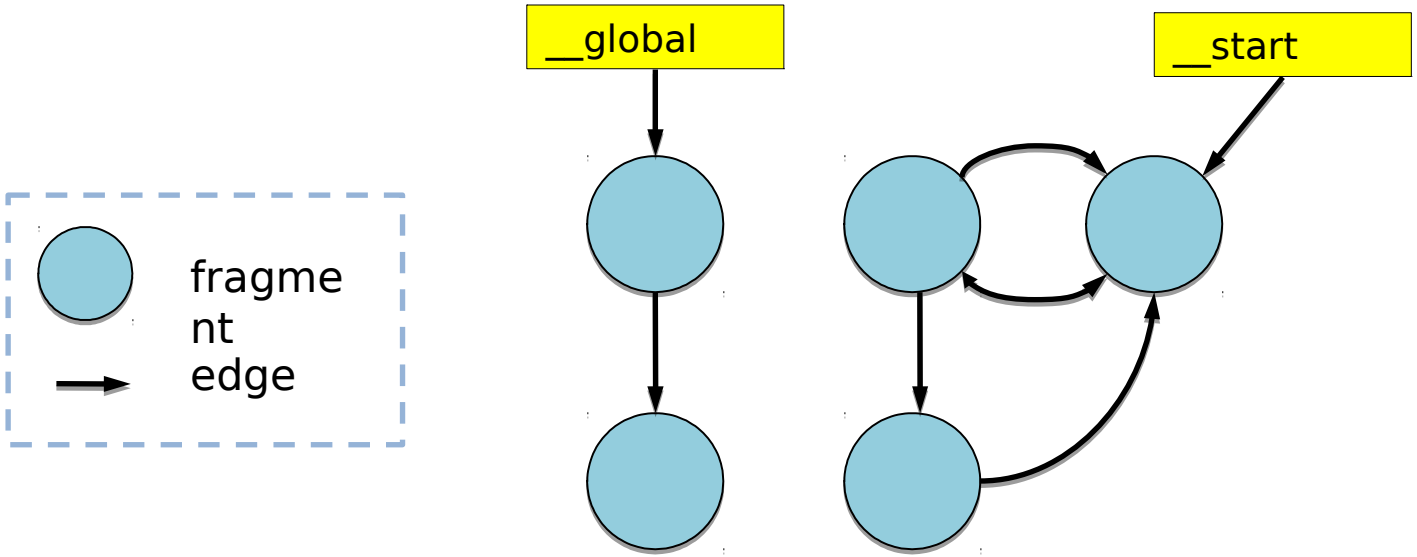
- A reference is a symbolic linkage between two fragments
  - A reference is an directed edge from use to define



- MCLinker represents the input modules as a graph structure
  - Vertices describe the fragments of modules
  - Edges describe the references between two fragments

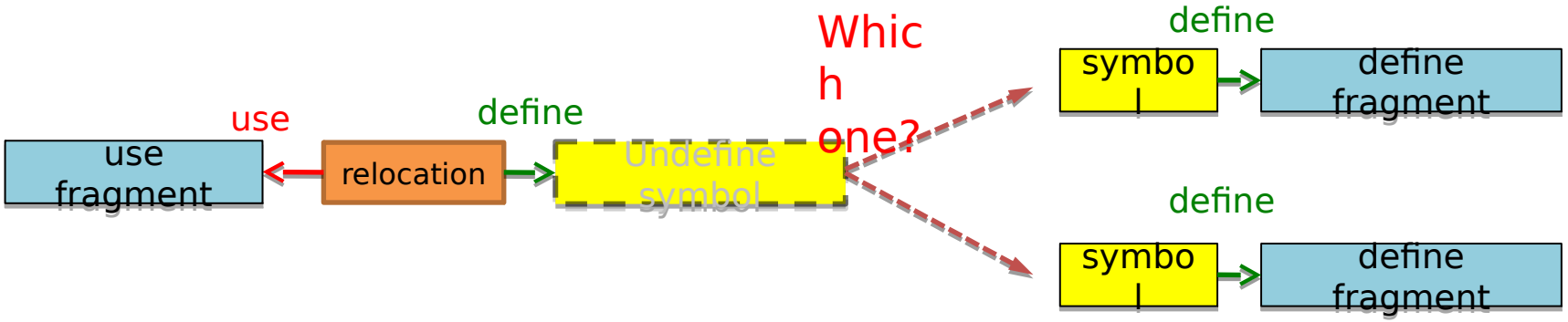
Fragment-Reference Graph (2/2)

- A Fragment-Reference Graph is a digraph, FRG = (V, E, S, O)
  - V is a set of fragments
  - E is a set of references, from use to define
  - S is a set of define symbols. They are the entrances of the graph
  - O is a set of exits and explains later.



Symbol Resolution

- Determine the topology of the reference graph
  - Relocation is a plug
  - Define symbol is a slot
  - Symbol resolution connects plugs and slots.
- Symbols has a set of attributes to help linkers determine the correct topology

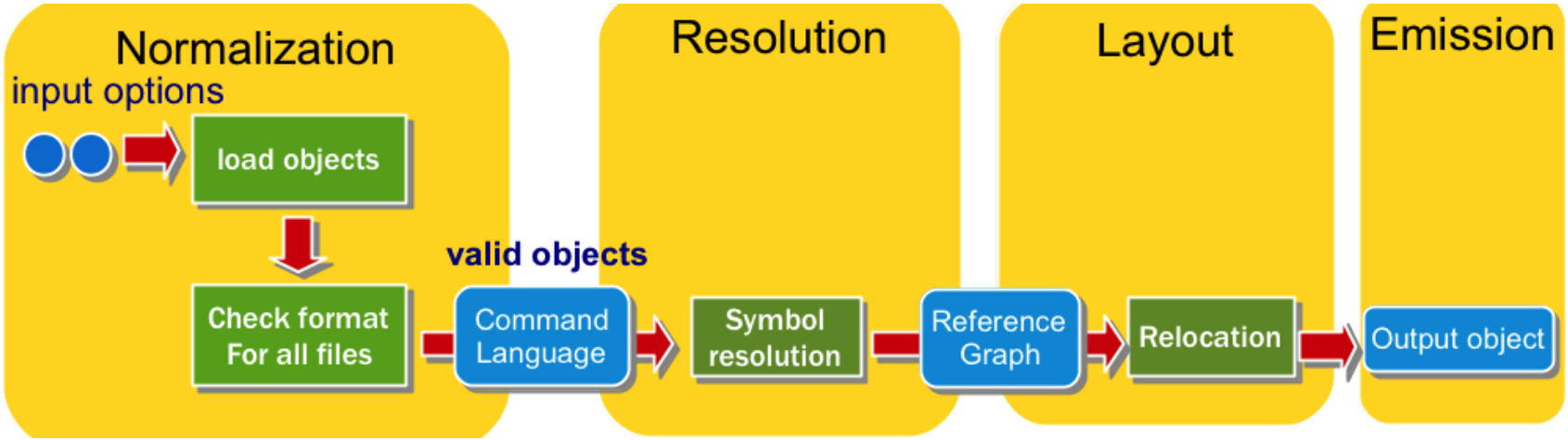


Optimizations on the Fragment-Reference Graph

- Fragment stripping
  - Remove unused fragment for shrink code size (Reachability problem)
  - Traditional linkers strip coarse sections. But MCLinker can strips finer-grained fragments.
  - The finer granularity, the smaller code size
- Branch optimization
  - Replace high cost branch by low cost branch
  - Optimizing by change of the relocation type
- Low-level inlining - ICF
- Fragment duplication for TLS optimization and copy relocations

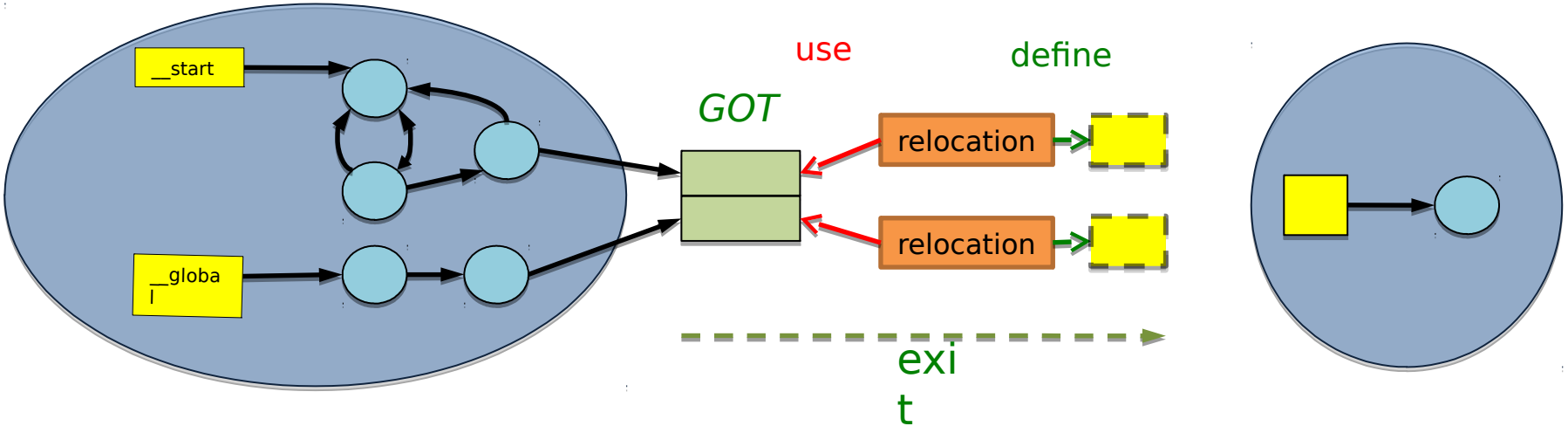
Layout

- To serialize the reference graph into a address space
  - Scan relocations
  - Layout
  - Apply relocations



Exits of The Fragment-Reference Graph

- A Fragment-Reference Graph is a digraph, FRG = (V, E, S, O)
  - O is a set of exits. An exit represents a dynamic relocation to GOT.
  - Represent to access external variables or to call an external function exits the FRG
- If the defining fragment is in an external module, then MCLinker will add exits for the references to the outside module.
  - We have no way to know the memory address of the external module until the load time
  - We add the Global Offset Table (GOT) for the unknown addresses
  - We add dynamic relocations for all entries of the GOT
  - Loader will apply the dynamic relocations and set the correct address in the GOT.
  - The program use the GOT to accesses the external module indirectly



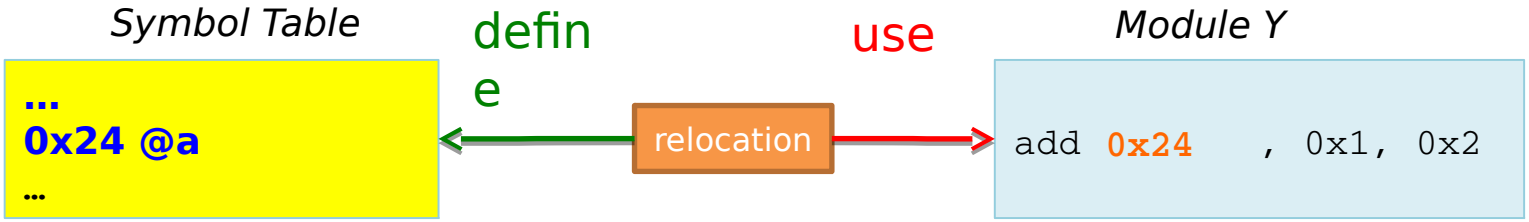
Layout

- Layout is a process to finalize the address of fragment and symbols
  - Sorts FRG=(V, E, S, O) topologically
  - Assigns addresses to {V, S, O}
- Before layout, we must calculate the sizes of all elements of the graph
  - Relocation scanning
    - Reserve exits and calculate the sizes of all exits
    - Undefined global symbol, GOT, and dynamic relocations
  - *Pre-layout
    - Calculate the size of all fragments
    - Calculate the size of all entrances
      - Global symbols and the hash table

* MCLinker follows the Google gold linker's naming. But pre-layout is opaque and may be renamed.

Apply relocation (1/2)

- Adjusts the content of using fragments
  - Final addresses of symbol is known after layout
  - Correct use fragment by accessed address

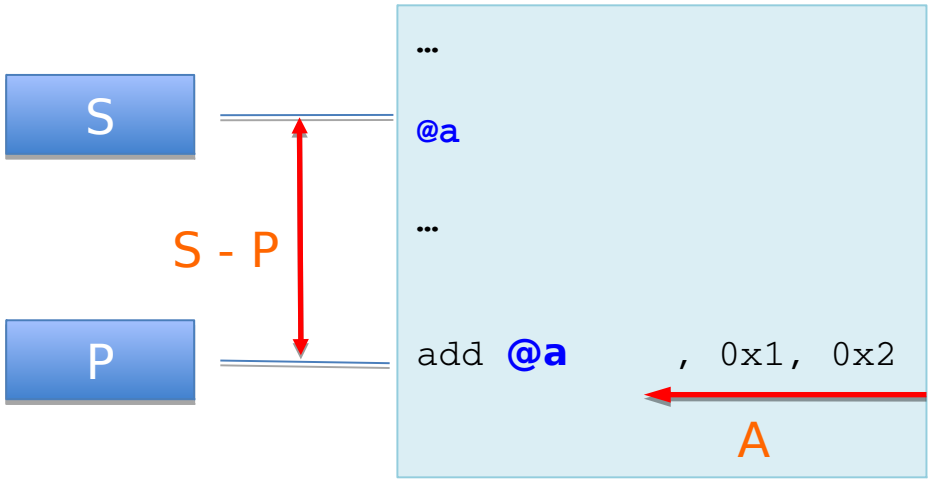


Apply relocation (2/2)

- Replaces absolute addresses by PC-related offset if supported by the target
- Basic Relocation Formula

  S – P + A

  – S: the symbol value
  – P: the place of the use instruction
  – A: addend, adjustment (by the instruction format)



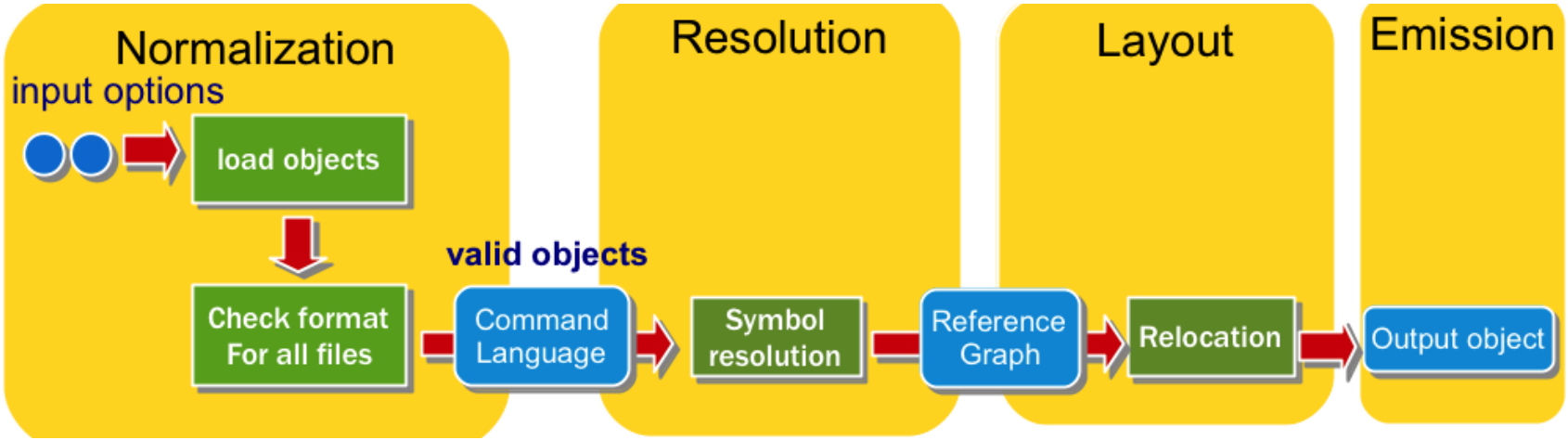
address space

Optimizations on Layout

- Dynamic Prelinking
  - If the system puts shared libraries at a fixed memory location, we can fill GOT with fixed addresses to avoid symbol look up in the loader
- Static Prelinking
  - If the system puts shared libraries at a fixed memory location, we can directly refer to the fixed addresses without any exits
- Symbol Stripping
  - Strip the undefined symbols which is not a exit
- Sections/functions/basic block Reordering
  - Linker knows the address and can perform better reordering

Emission

- Emits the module in the output formats
  - Adds format information
  - Writes down the IR
- In order to improve both cache and page locality, MCLinker collects and performs most file operations in this stage.
  - MCLinker copies the content in the inputs and applies the resolved reference in this stage.



Outline

- The History
  - Target Independent Linkers
  - Post Optimizers
  - Instrumentation Tools
- The Theory
  - Linking Language
  - Fragment-reference graph
- The Future
  - for GPGPU; for virtual machines
  - The bold project

唐文力 Luba Tang

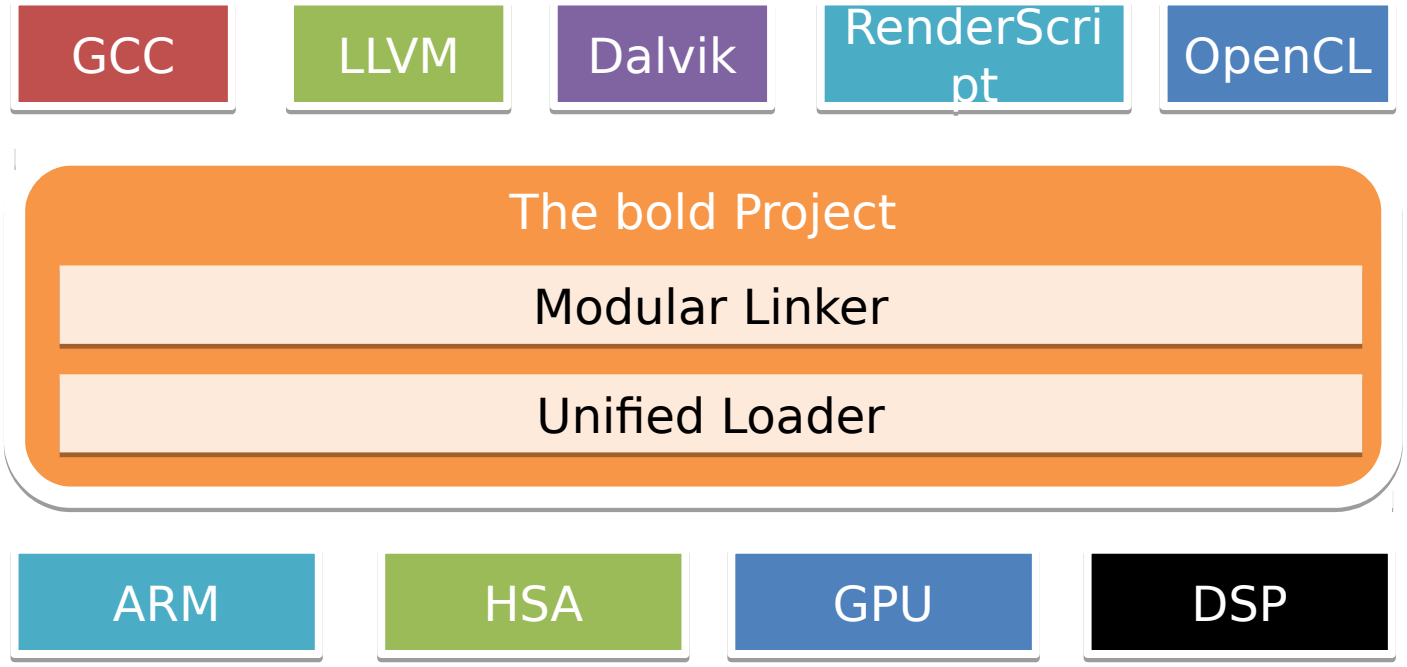
CEO & Founder of Skymizer Inc.

Architect of MCLinker and GYM compiler

Compiler and Linker/Electronic System Level Design

Challenge: Unified Shared Memory of Heterogeneous Many-Core System

- Installation time compilation
  - GPGPU languages (OpenCL, CUDA, RenderScript)
  - Virtual Machine (Dalvik, RenderScript)
- Heterogeneous Many-core System
  - Universal ELF



The bold Project

- **BSD licensing linker**
  - General purpose linker/loader
  - Focus on optimization
  - Linking in parallel
- **OA (Owner agreement) and CA (Committer agreement)**
  - Avoid interest confliction between industry and community.
  - Legal person can not be an owner