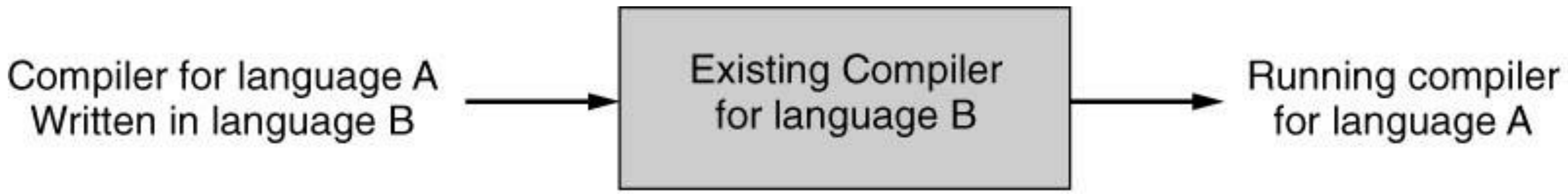
Compiler Language

- The implementation (or **host**) language has to be machine language.
  - This was how the first compilers were written.
- Another approach is to write the compiler in another language for which a compiler already exists.
  - We need only to compile the new compiler using the existing compiler to get a running program.
- What if the existing compiler runs on a machine different from the target machine?
  - Compilation produces a **cross compiler** – a compiler that generates target code for a different machine.

Compiler Language

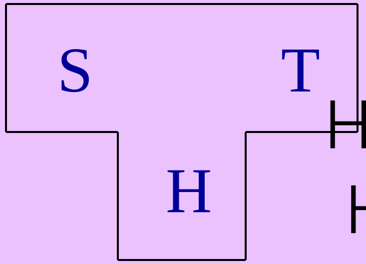
- **Bootstrapping**
- **Porting**



T-Diagram

- A compiler written in language H (host language) that translates language S (source language) into language T (target language) is drawn as the following **T-diagram**:

T代表一个编译器



H==T可执行编译器

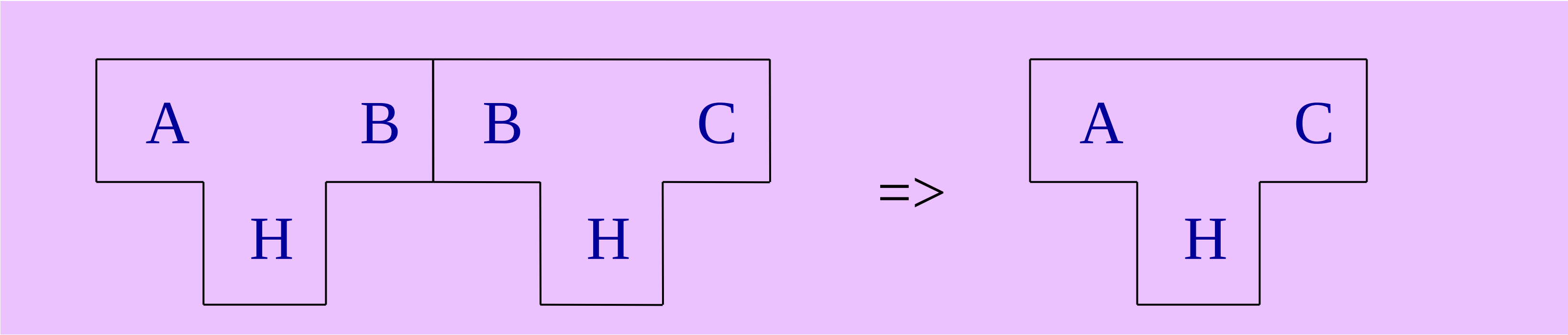
H！ =T交叉编译器 不能

运行

- This is equivalent to saying that the compiler runs on “machine” H.
- Typically, we expect H = T.
  - the compiler produces code for the same machine as the one on which it runs.

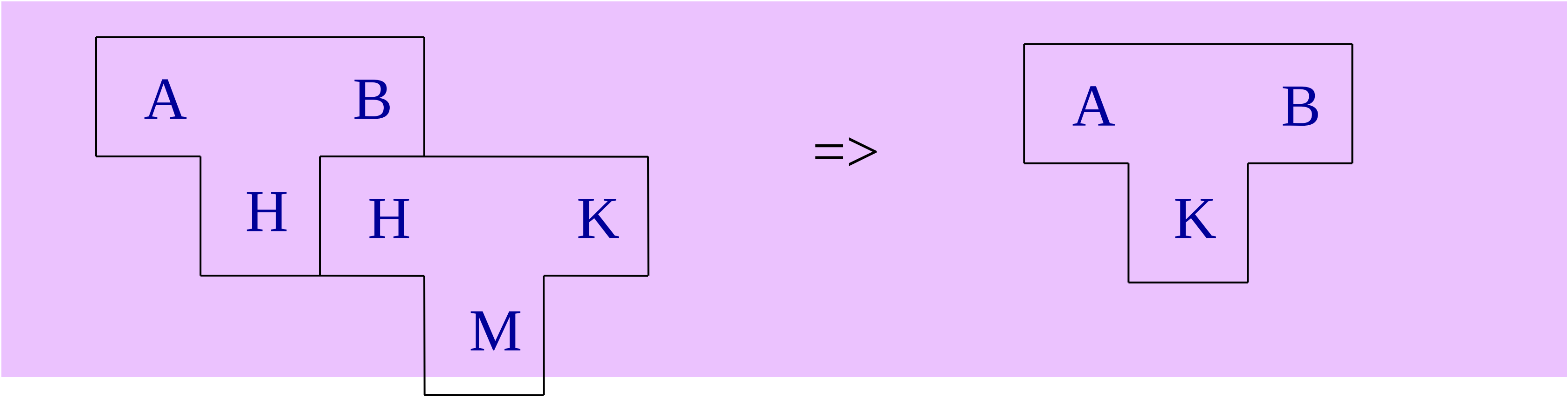
Case 1

- There are two compilers that run on the same machine H.
  - One translates from language A to language B.
  - The other translates from language B to language C.
- We can combine them by letting the output of the first to be the input to the second.
- The result is a compiler from A to C on machine H.



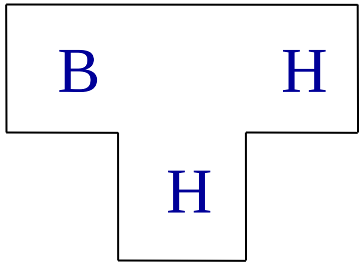
Case 2

- We can use a compiler M from “machine” H to “machine” K to translate the implementation language of another compiler from H to K.

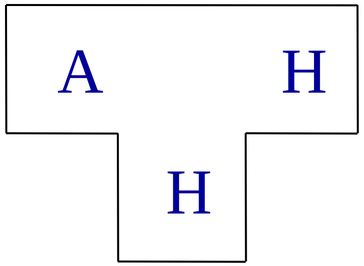


Existing Language

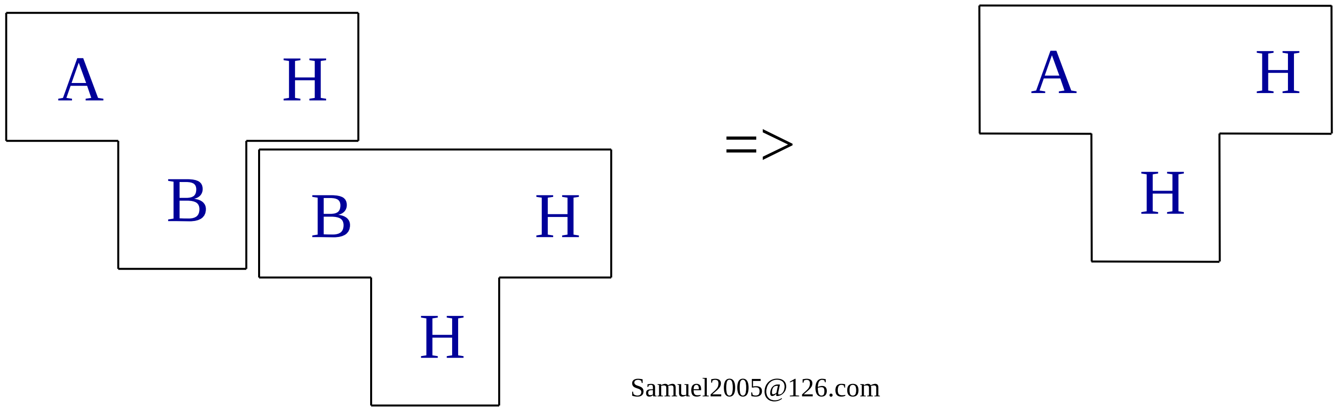
- Given:
  - Machine H
  - Compiler for a language B written in H that translates B to H.



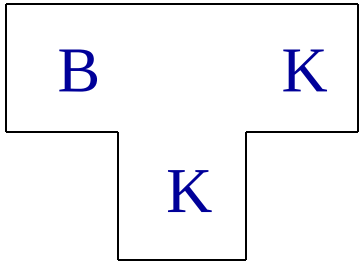
- Wanted:
  - To get a compiler on machine H that translates a language A to H



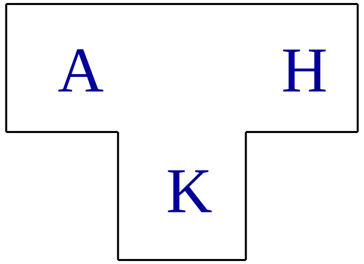
- We can write a compiler for A using language B.



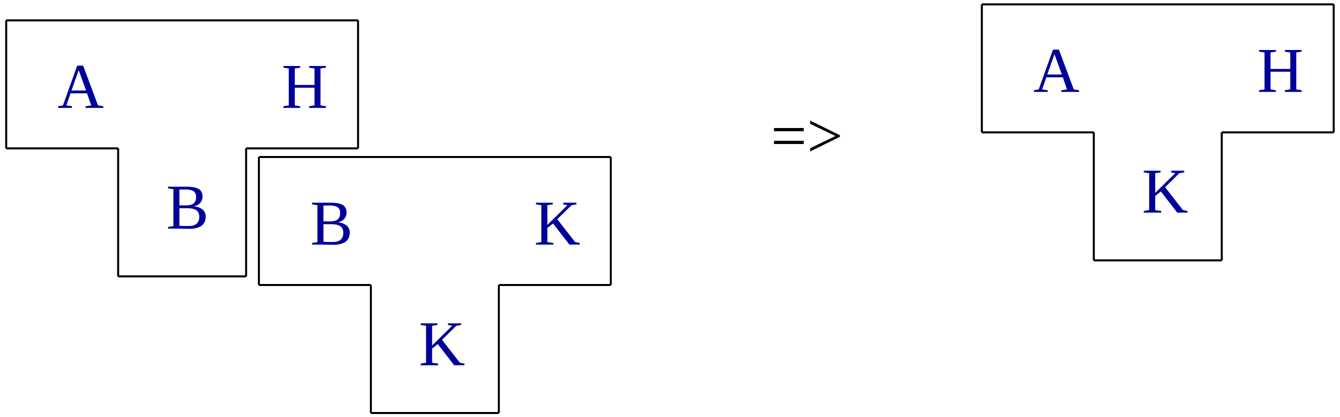
Cross Compiler



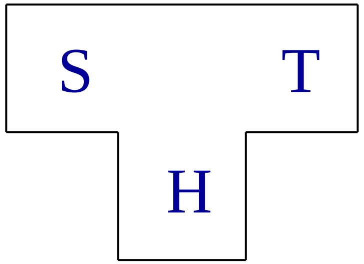
- Given :
  - Machine K. (K is a host language here.)
  - A compiler for a language B written in K that translates B to K.

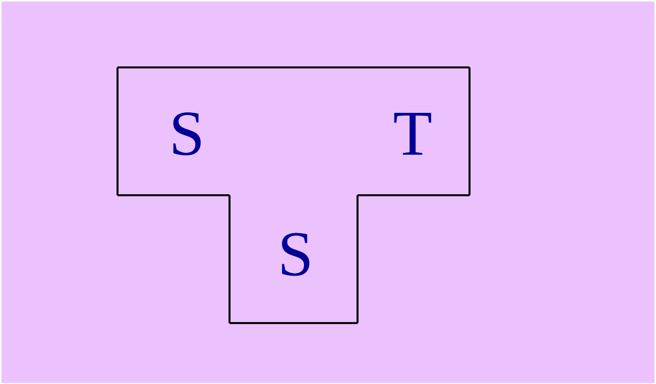


- Wanted:
  - To get a compiler for a language A. The target machine is H.



H = S ?



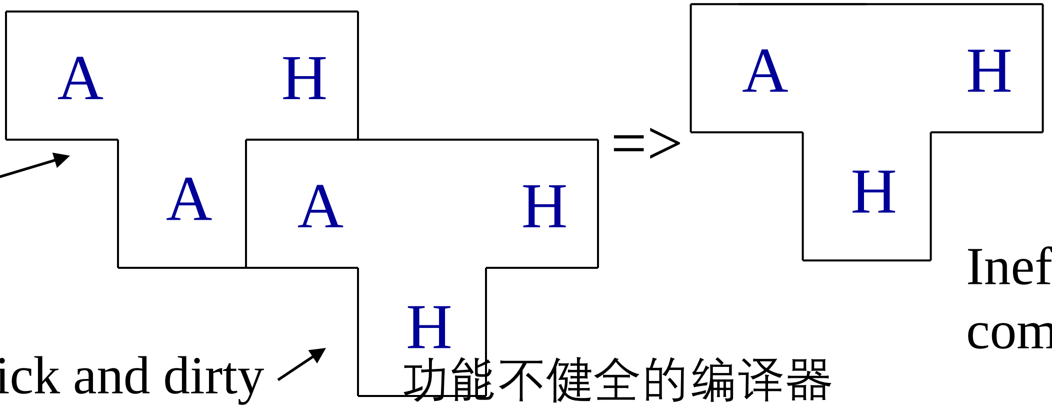


- Can a compiler be written in the same language that it is to compile?
- No compiler for the source language exists, so the compiler itself cannot be compiled.
- There is a circulatory problem here.

Bootstrapping 自举

1. Write a “quick and dirty” compiler in assembly language.
2. Use this compiler to compile the “good” compiler.
3. Recompile the “good” compiler to produce a final version.

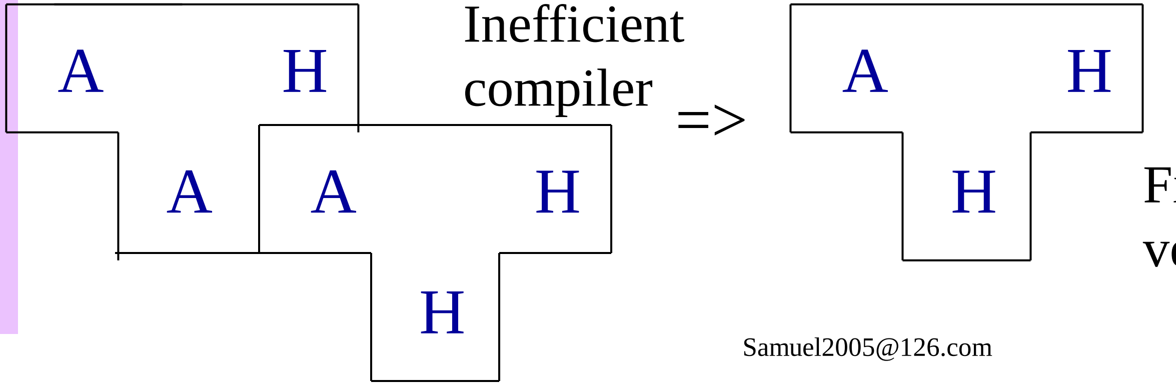
Compiler in its own language



Quick and dirty

功能不健全的编译器

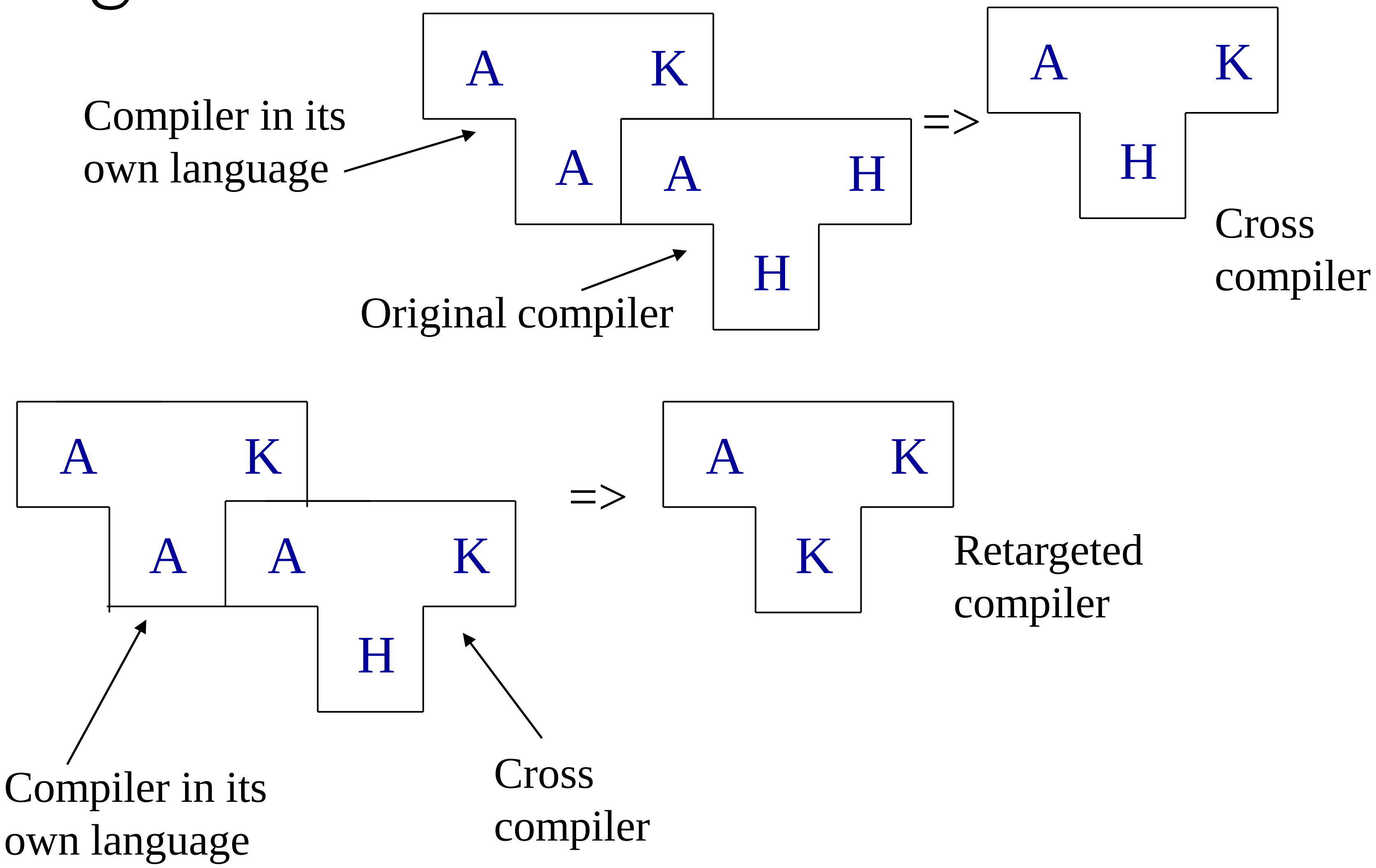
Inefficient compiler



Final version

还可以继续迭代，一般三遍效果就很好

Porting to New Host



Sample Language and Machine

- A “real” compiler for a real machine has far too much detail.
- We will use a small language TINY as a running example.
- The target code is the assembly language for a simple hypothetical processor – TM machine.

TINY Language

;

- Sequence of statements separated by semicolon.
- No procedures and no declarations.
- All variables are integers, declared by assignment.
- Two control statements
  - if -statement with optional else part. Must be terminated with the keyword end.
  - repeat-statement
- read and write statements for I/O.
- Comments are within curly brackets.

{ }

TINY Language (cont)

- Arithmetic expression involves integer constants, variables, parentheses, and for integer operations `+`, `-`, `*`, `/`.
- Boolean expression is two arithmetic expression combined with the two comparison operators `<` and `=`.
  - May appear only as test in control statements.
- `:=` is used for an assignment operator.

Factorial Program

```
{ Sample program
  in TINY language -
  computes factorial }
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact  { output factorial of x }
end
```

TM Machine

- TM has some of the properties of RISC's.
  - All arithmetic must take place in registers
  - Addressing modes are limited
- The following example illustrated the code for

  a[index] = 6;

  - index is assumed to be at location 10 in memory
  - a is at location 20 in memory.
- Addressing modes for load operation
  1. LDC is load constant
  2. LD is load from memory
  3. LDA is load address

Exam

a[index] = 6;
index is assumed to be at location 10 in memory
a is at location 20 in memory.

LDC is load constant
LD is load from memory
LDA is load address

```
LDC   1,0(0)    load 0 into reg 1
LD    0,10(1)   load val at 10+R1 into R0
LDC   1,2(0)    load 2 into reg 1
MUL   0,1,0     put R1*R0 into R0
LDC   1,0(0)    load 0 into reg 1
LDA   1,20(1)   load 20+R1 into R1
ADD   0,1,0     put R1+R0 into R0
LDC   1,6(0)    load 6 into reg 1
ST    1,0(0)    store R1 at 0+R0
```

- Addresses are in the form of “register+offset”
  - 10(1) is address computed by the offset 10 to the contents of register 1

Homework

1.7 Suppose you have a Pascal-to-C translator written in C and a working C compiler. Use T-diagrams to describe the steps you would take to create a working Pascal compiler.