Rethinking Distributions in HPF

How I Would Address a Fundamental Shortcoming of the Language

Ken Kennedy

Center for High Performance Software

Rice University

http://www.cs.rice.edu/~ken/Presentations/HPF2000Keynote.pdf

Collaborators

Bradley Broom

Arun Chauhan

Keith Cooper

Jack Dongarra

Rob Fowler

Dennis Gannon

Lennart Johnsson

John Mellor-Crummey

John Reynders

Linda Torczon

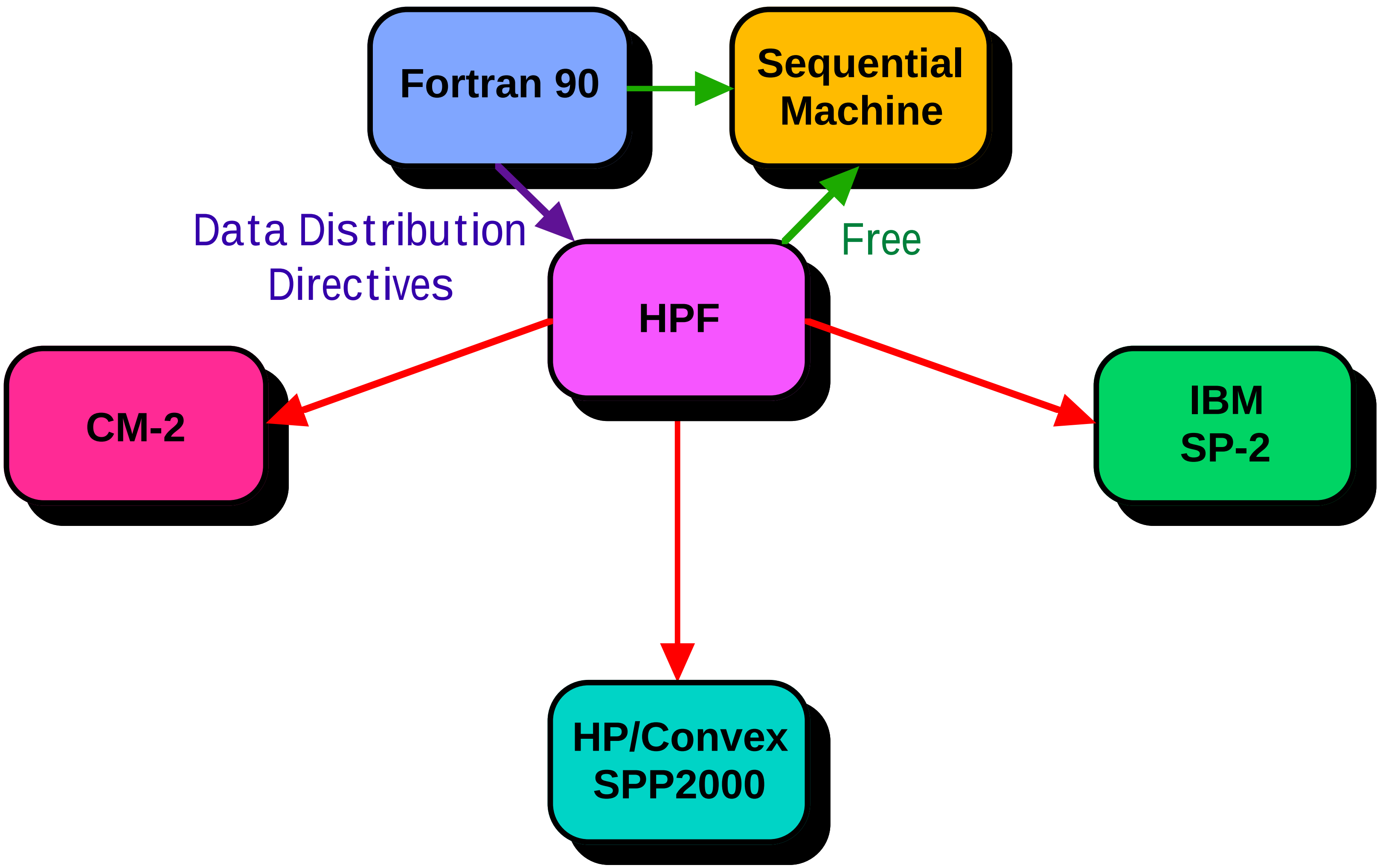
Status of Scalable Parallelism

- Dream
  - virtually limitless computing power at low cost
  - performance scalable from one to thousands of processors
  - easy portable programming
- Reality
  - successful at only moderate levels of scalability
  - modest progress in programmability and scalability
  - limited penetration in industry
    - independent software vendors (ISVs) still reluctant
    - limited protection of programming investment
- Remedy: Architecture-Independent Programming
  - a programming language and its compilers support architecture-independent parallel programming if, for each target architecture,
    - compiled code $\cong$ hand code for same algorithm

HPF Goals

- Support for Scalable Parallel Systems
  - scaling from one to thousands of processors
- Focus on Data Parallelism
  - parallelism through subdivision of data domain
- Machine Independent Programming Support
  - object program achieves performance comparable to hand-coded MPI on each target machine on the same algorithm
- High Level of Abstraction
  - more accessible programming model
    - single thread of control
    - shared memory
    - implicit generation of communication

HPF Strategy



Problems for HPF

- Compilers slow to mature
  - Fortran 90 features supported inconsistently
  - compilation for highest efficiency complex
  - initially, efficiency of object programs unsatisfactory
  - early users may become discouraged
- Library support lacking
  - no CMSSL equivalent
- Needed features are missing
  - support for irregular problems
  - task parallelism
  - high performance input/output
- Complex relationship between program and performance
  - explanatory and diagnostic tools are needed

Problems for HPF

- Compilers slow to mature
  - Fortran 90 features supported inconsistently
  - compilation for highest efficiency complex
  - initially, efficiency of object programs unsatisfactory
  - early users may become discouraged

Much R&D, but lasting impression

- Library support lacking
  - no CMSSL equivalent

Still a problem

- Needed features are missing
  - support for irregular problems
  - task parallelism
  - high performance input/output

Still a big problem

OpenMP?

- Complex relationship between program and performance
  - explanatory and diagnostic tools are needed

Solutions available

Rethinking HPF

- Language Complexity
  - Adopt the OpenMP directives for SMP parallelism
  - Simplify the interprocedural handling of distributions
    - Go back to the original Fortran D idea:
      - Interprocedural propagation of distributions
      - With support for coding distribution-independent libraries
- Performance Issues
  - Embrace the HPF/JA extensions (Reflect, On Home Local)
  - Open-source HPF Library
  - Optimize the extrinsic interface
- Usability
  - Make it possible to extend the notion of distribution
    - Currently, HPF only allows built-in distributions

Idea: Encapsulated Distributions

- HPF's Fundamental Idea
  - Separate distribution from data structure
  - Hide issues of data movement from the user
- Problem
  - Built-in distributions are not sufficient for some problems
  - Expert user wants more control over distribution and performance
- Solution
  - Make it possible to add new distributions
    - DISTRIBUTE A(Hilbert2D)

      where Hilbert2D is a distribution library
- Question:
  - What does it mean to be a distribution?

What is a Distribution?

- Mapping from arrays to storage
  - According to some paradigm
- Must provide a minimum set of methods
  - Get(A,I,J), Put(A,I,J)
  - Get(A,iteratorIJ), Put(A,iteratorIJ)
    - Where iteratorIJ = (1:N,J) or (1:N,1:M:2) or ((I:I), I = 1:N)
  - Owner(A,I,J), Owners(A,iteratorIJ)
  - Reflect (fill overlap regions)
  - Global operators (shift, global sum)
  - Rebalance
  - Redistribute(Distlib2)
- Must do what compilers need to achieve performance

Advantages

- New distributions can be added as needed
  - Open source community
  - Current distributions are special cases
    - Although we need to keep the built-in distributions (more later)
  - Simplifies view of interesting new technologies
    - Out-of-core data distribution
- Expert user retains more control over performance
  - Manages own distribution
  - Provides communication primitives as needed
    - shift, global sum
  - Can include and manage ghost regions
  - Can design adaptivity strategy

Problems

- Performance
  - Current compilers get mileage from knowing the details of the distribution
    - For example, in determining which computations require communication
    - Rice dHPF uses integer set framework to reason about regions requiring communication
  - What do we do if the distribution is encapsulated in a collection of methods?
    - Owner(A(I,J)) is a case in point
- Reliability
  - What if designer constructs incorrect distributions?
- Solution Strategy:
  - Extensive preliminary analysis of distribution library

Detour: Support for High-Level Domain-Specific Programming

Telescoping Languages: Generating Problem-Solving Sytems from Annotate Libraries

Programming Productivity

- **Challenges**
  - programming is hard
  - professional programmers are in short supply
  - high performance will continue to be important

Programming Productivity

- Challenges
  - programming is hard
  - professional programmers are in short supply
  - high performance will continue to be important
- One Strategy: Make the End User a Programmer
  - professional programmers develop components
  - users integrate components using:
    - problem-solving environments (PSEs)
    - scripting languages (possibly graphical)

      examples: Visual Basic, Tcl/Tk, AVS, Khoros

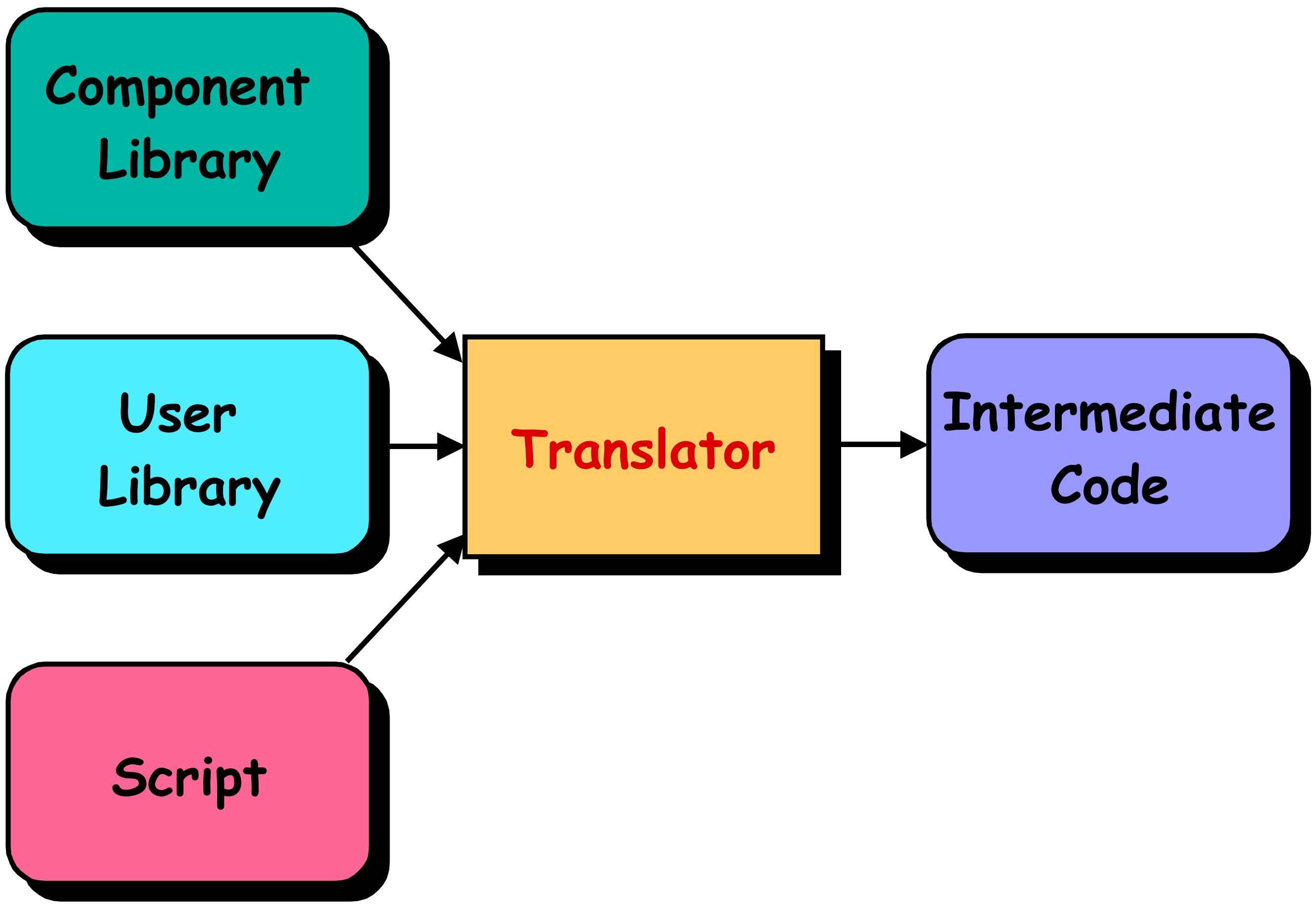
Programming Productivity

- Challenges
  - programming is hard
  - professional programmers are in short supply
  - high performance will continue to be important
- One Strategy: Make the End User a Programmer
  - professional programmers develop components
  - users integrate components using:
    - problem-solving environments (PSEs)
    - scripting languages (possibly graphical)

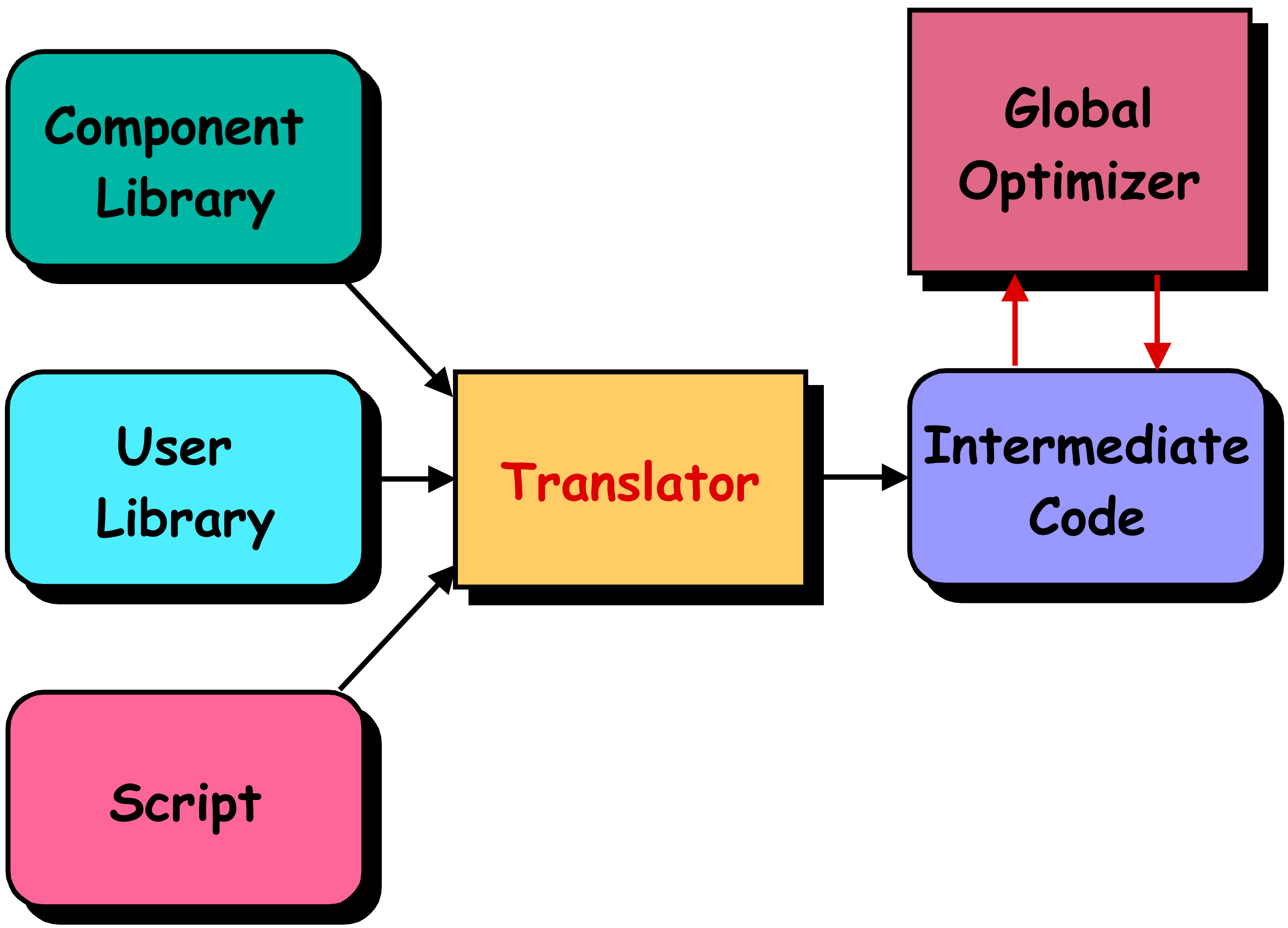
      examples: Visual Basic, Tcl/Tk, AVS, Khoros
- Compilation for High Performance
  - translate scripts and components to common intermediate language
  - optimize the resulting program using interprocedural methods

Script-Based Programming

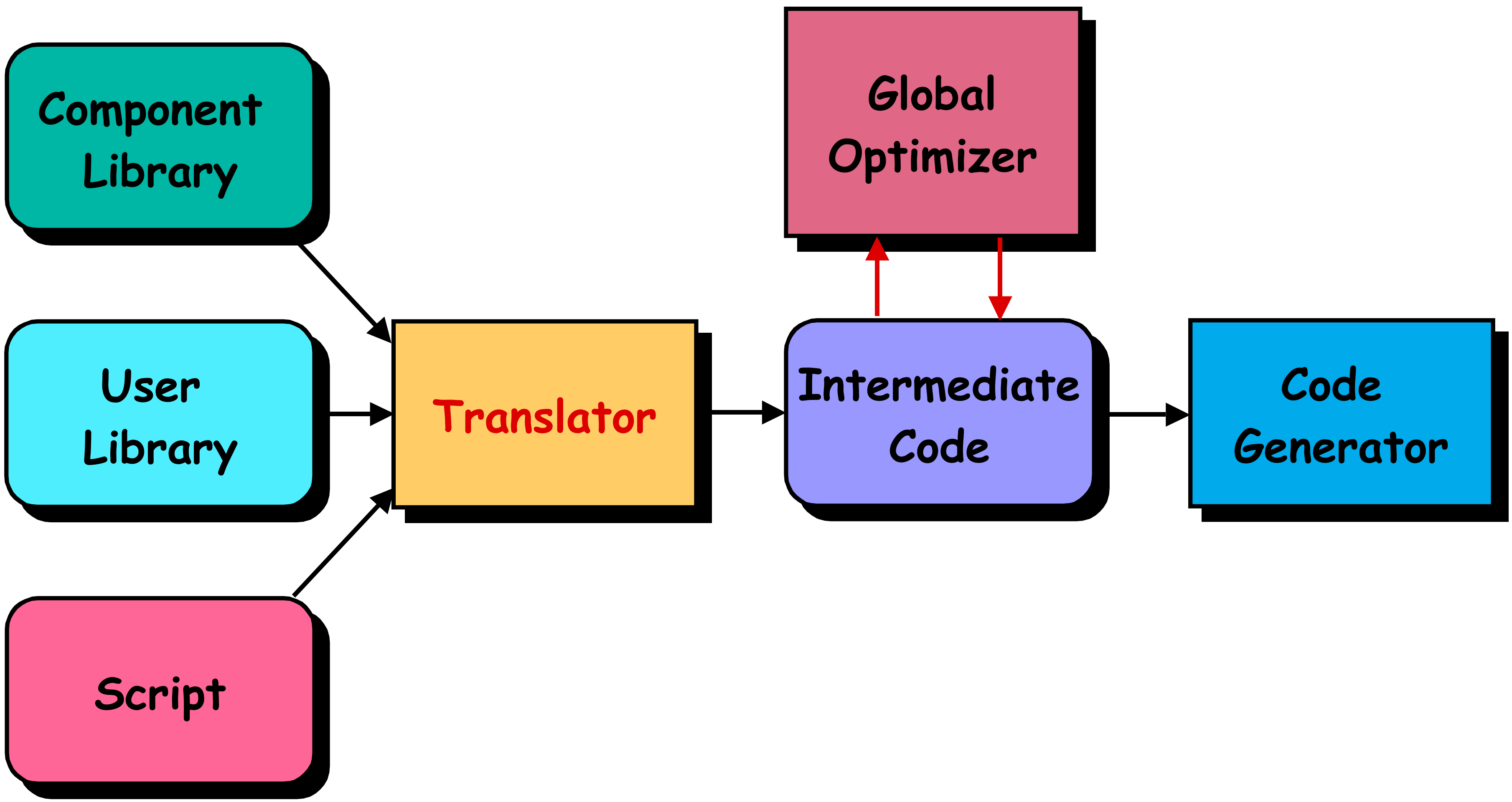
Script-Based Programming



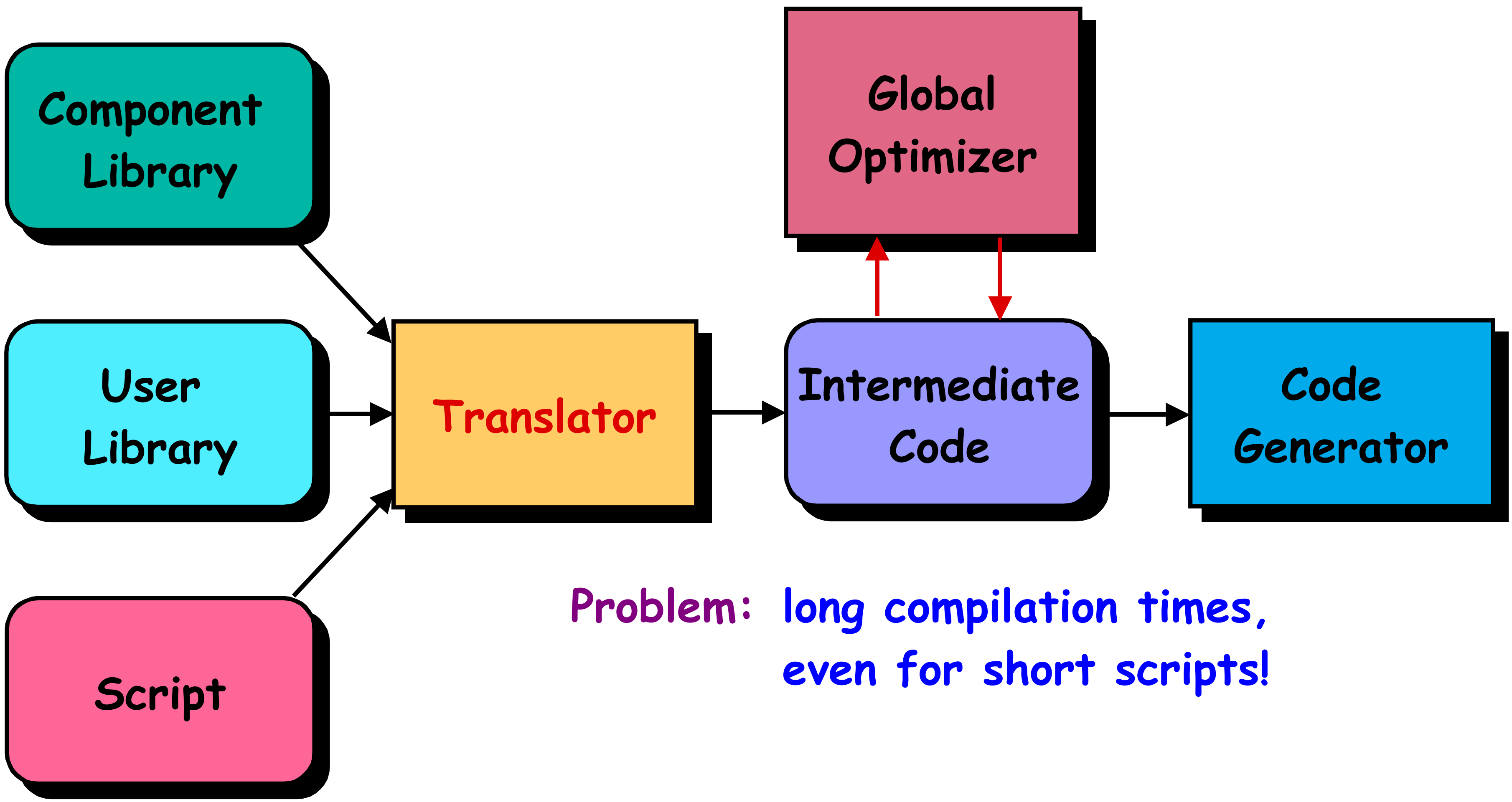
Script-Based Programming



Script-Based Programming

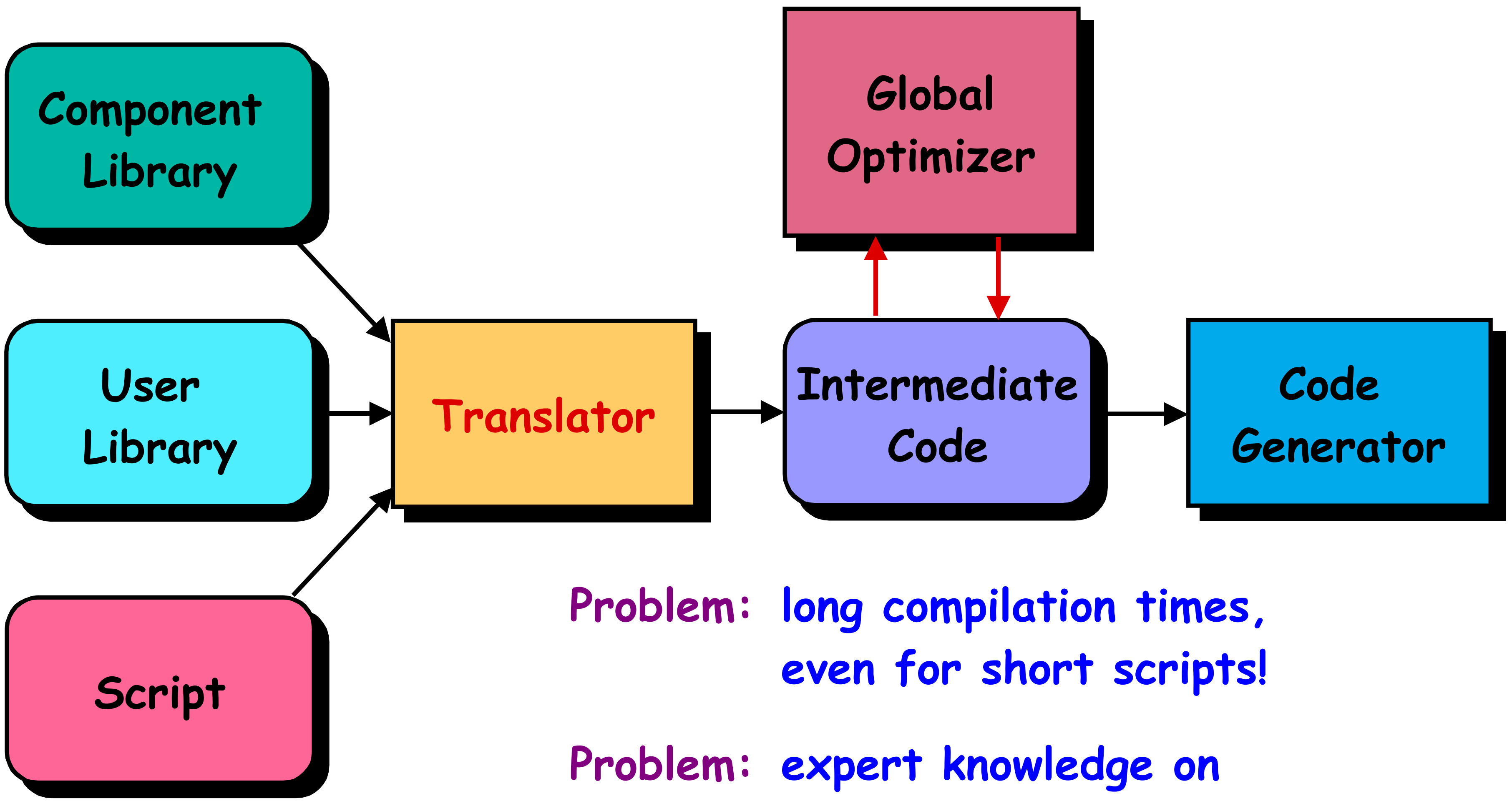


Script-Based Programming



Problem: long compilation times, even for short scripts!

Script-Based Programming



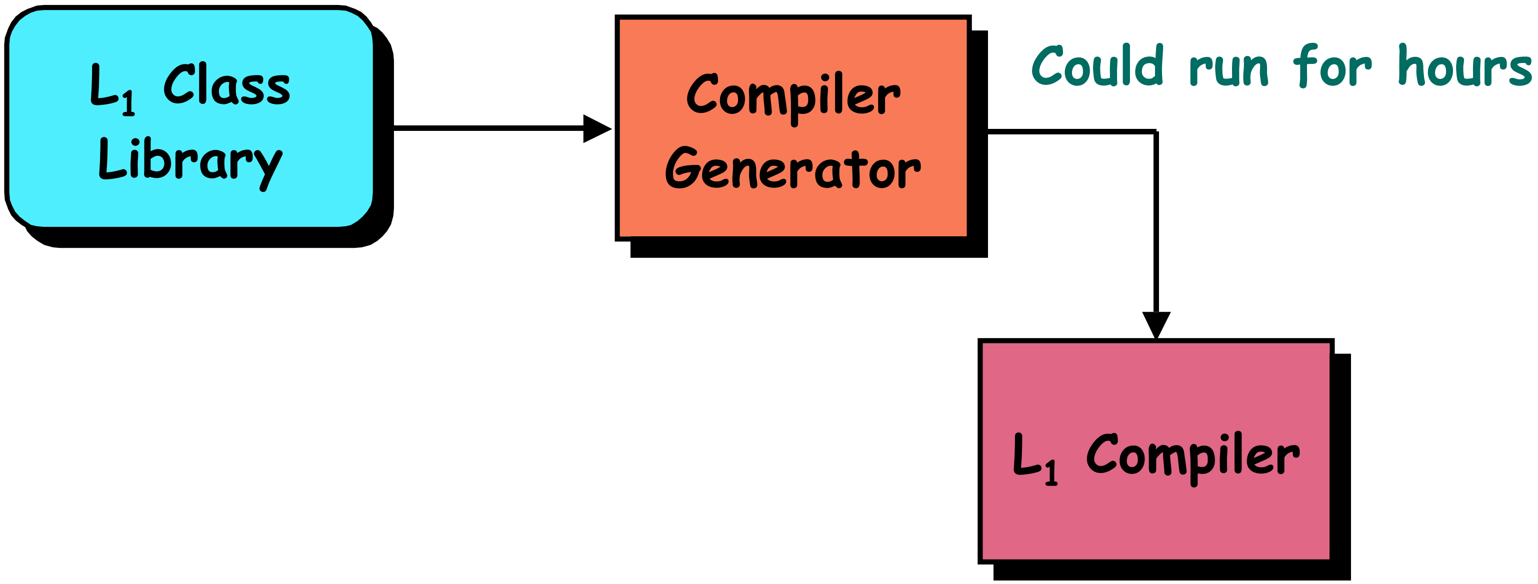
Problem: long compilation times, even for short scripts!

Problem: expert knowledge on specialization lost

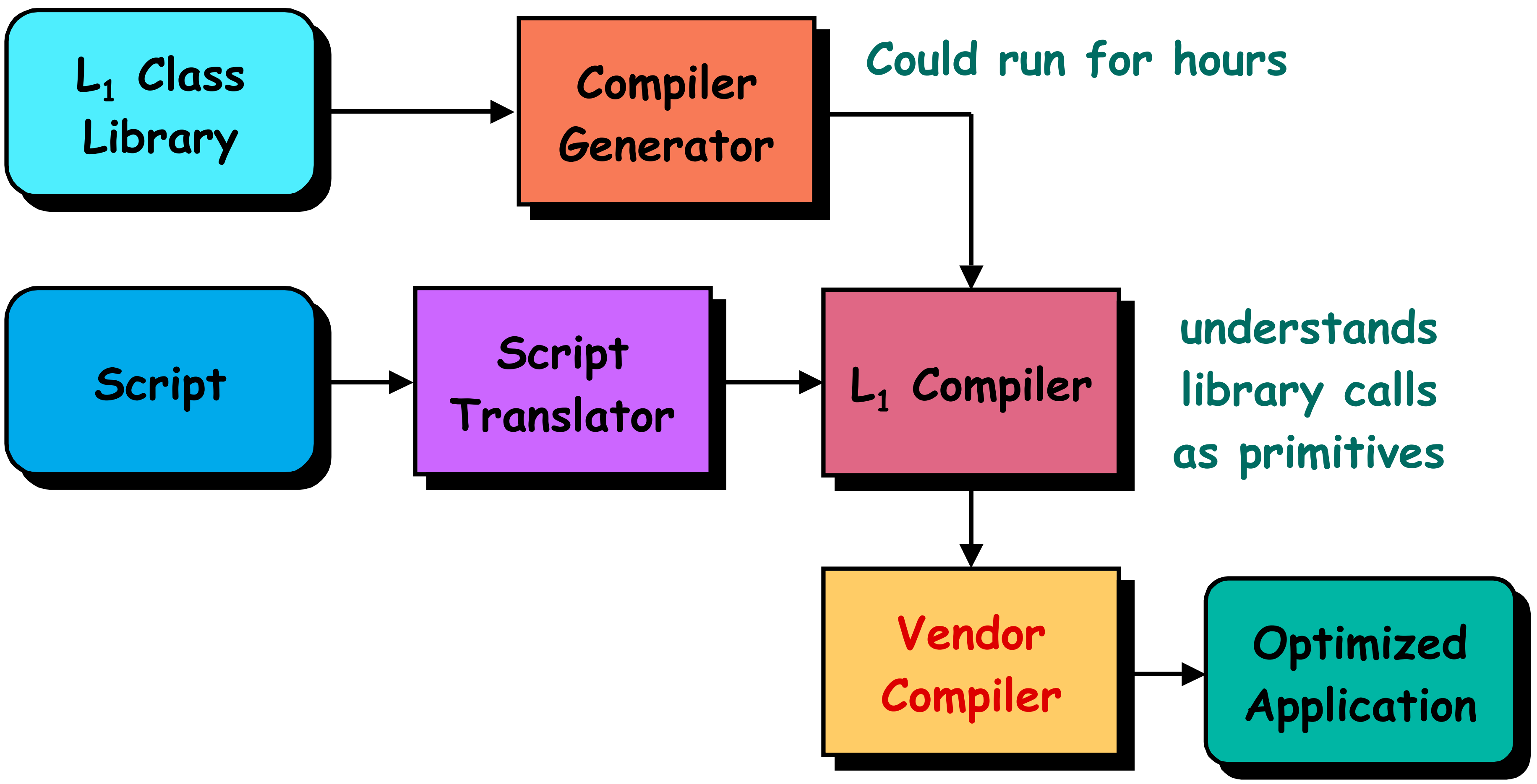
Telescoping Languages

L_1 Class Library

Telescoping Languages



Telescoping Languages



Telescoping Languages: Advantages

- **Compile times can be reasonable**
  - More compilation time can be spent on libraries
    - Amortized over many uses
  - Script compilations can be fast
    - Components reused from scripts may be included in libraries

Telescoping Languages: Advantages

- Compile times can be reasonable
  - More compilation time can be spent on libraries
    - Amortized over many uses
  - Script compilations can be fast
    - Components reused from scripts may be included in libraries
- High-level optimizations can be included
  - Based on specifications of the library designer
    - Properties often cannot be determined by compilers
    - Properties may be hidden after low-level code generation

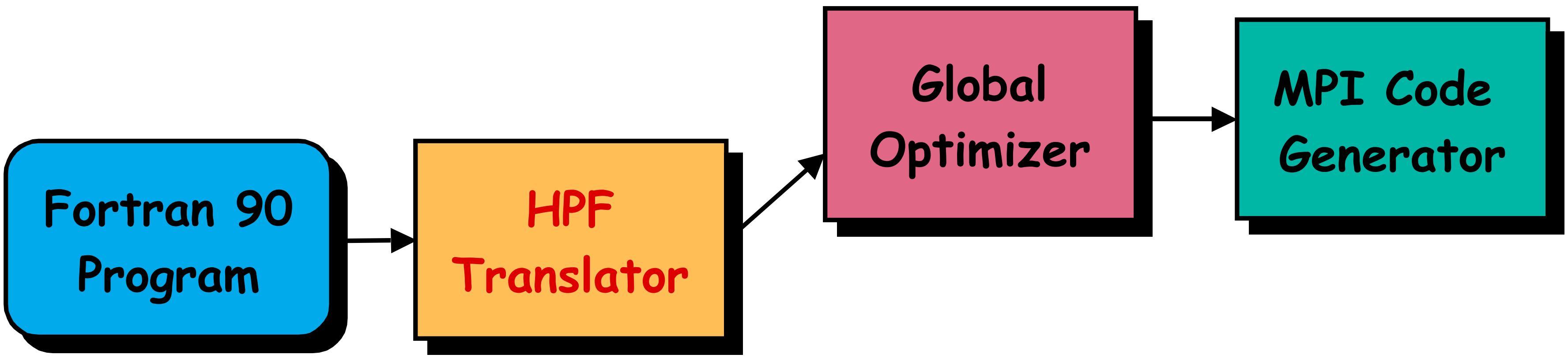
Telescoping Languages: Advantages

- Compile times can be reasonable
  - More compilation time can be spent on libraries
    - Amortized over many uses
  - Script compilations can be fast
    - Components reused from scripts may be included in libraries
- High-level optimizations can be included
  - Based on specifications of the library designer
    - Properties often cannot be determined by compilers
    - Properties may be hidden after low-level code generation
- User retains substantive control over language performance
  - Mature code can be built into a library and incorporated into language

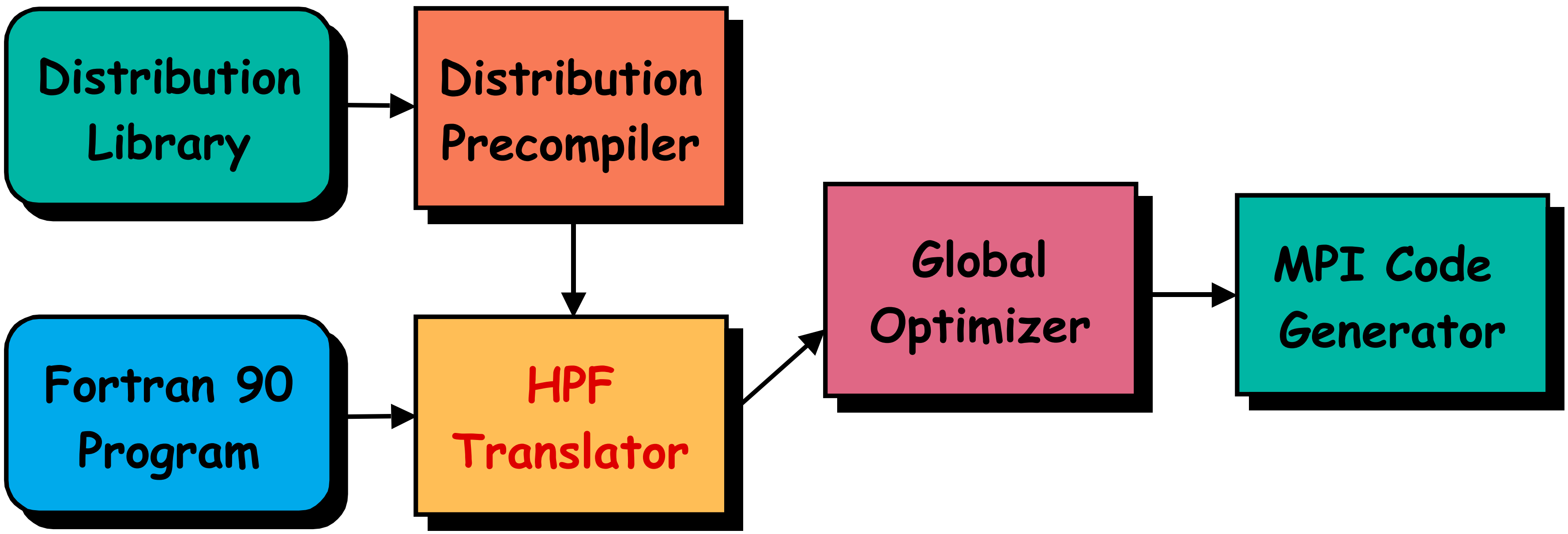
Applications

- Matlab Compiler
  - Automatically generated from LAPACK or ScaLAPACK
    - With help via annotations from the designer
- Automatic Generation of POOMA
  - Data structure library implemented via template expansion in C++
  - Long compile times, missed optimizations
- Generator for Grid Computations
  - GrADS: automatic generation of NetSolve
- Flexible Data Distributions
  - Failing of HPF: inflexible distributions
  - Data distribution == collection of interfaces that meet specs
  - Compiler applies standard transformations

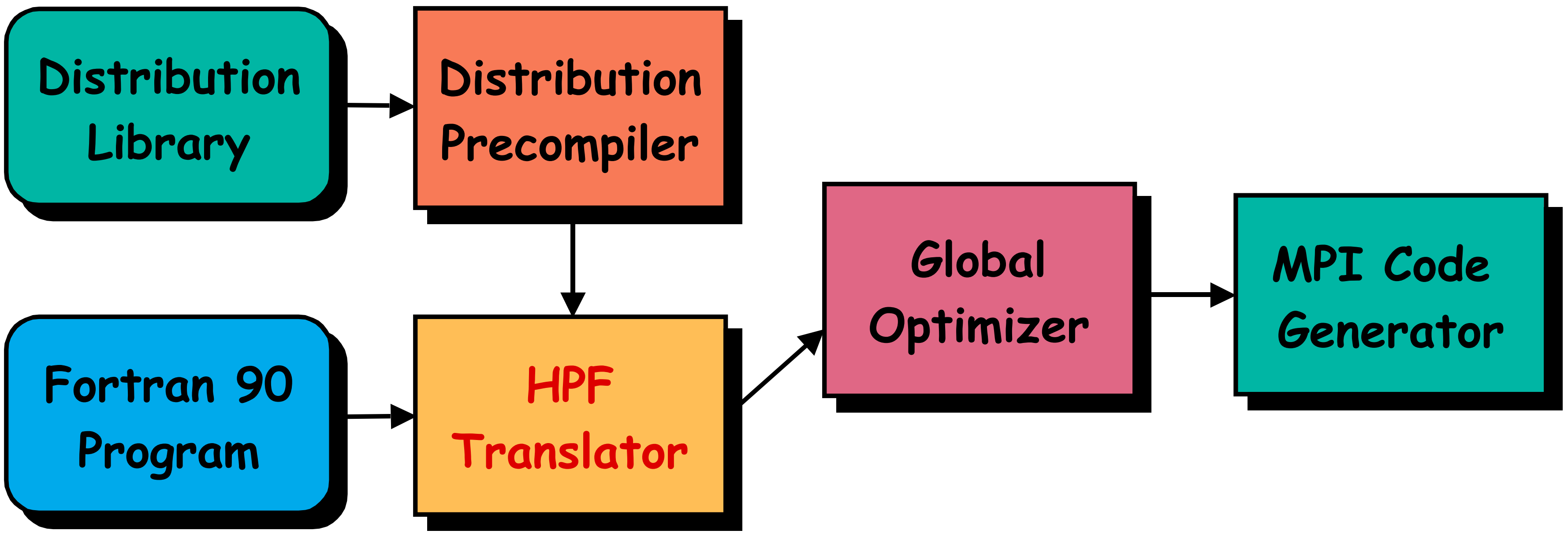
Application to HPF



Application to HPF



Application to HPF




```
Distribute (Hilbert2D): A,B
Do i = 1,100
    A(i) = B(i) + C
Enddo
```

→

```
A.putBlock(1,100,
    B.getBlock(1,100) + C)
```

Leverage from Telescoping Languages

- High-level Specifications
  - Provide information about when certain optimizations can be done
    - Access vectorization
  - Suggest specialized substitutions unique to distribution
- Providing Knowledge to the Compiler
  - If the owner(A(I,J)) functionality is particularly simple, substitute the code inline
    - Automatic inversion possible
  - Determination whether distribution is known at compile time
    - If it is, inspector can be embedded in compilation phase
  - Compiler can specialize run-time distributions to program context
    - partial evaluation of distribution

Example

- Unknown owner

```
DO I = 1, N
   DO J= 1,N;
         A(I,J) = A(I+1,J) + C
   ENDDO
ENDDO
```

- Becomes

```
DO (I,J) in OwnedBy(pI,pJ)
   IF (Owner(A(I+1,J))≠(pI,pJ)) THEN
      Get(A(I+1,J)) into X
      A(I,J) = X + C ! All local
   ELSE
      A(I,J) = A(I+1,J) + C
   ENDIF
ENDDO
```

Need inverse!

Example Continued

- **Recursive bisection load balance:**
  - Processor (pI,pJ) owns
    - Iterations of I loop such that LowI(pI) ≤ I ≤ HiI(pI)
    - Iterations of J loop such that LowJ(pI,pJ) ≤ J ≤ HiJ(pI,pJ)

```
VPUT(A(LoI(pI), LoJ(pI,pJ):HiJ(pI,pJ)) to (pI-1,pJ)
DO I = LoI(pI),HiI(pI)-1
    DO J = LoJ(pI,pJ), HiJ(pI,pJ)
       A(I,J) = A(I+1,J) + C
    ENDDO
ENDDO
VGET(A(HiI(pI)+1,LoJ(pI,pJ):HiJ(pI,pJ)) into arrayX
DO J = LoJ(pI,pJ), HiJ(pI,pJ)
     A(I,J) = arrayX(J) + C
ENDDO
```

Summary

- Mixed Reviews on HPF
  - Many strengths: separation of distribution from data
  - Many weaknesses
    - Performance and usability
- Rethinking HPF
  - Need to focus on issues that will help users solve problems
    - Need simplicity, generality and control
- Idea: Extensible Distributions
  - Distribution is a class defining mapping of data to storage
  - Any class providing minimal set of methods may be used
- Compilation Technologies
  - Existing HPF compilers must be rewritten
  - Telescoping languages strategy can buy back performance