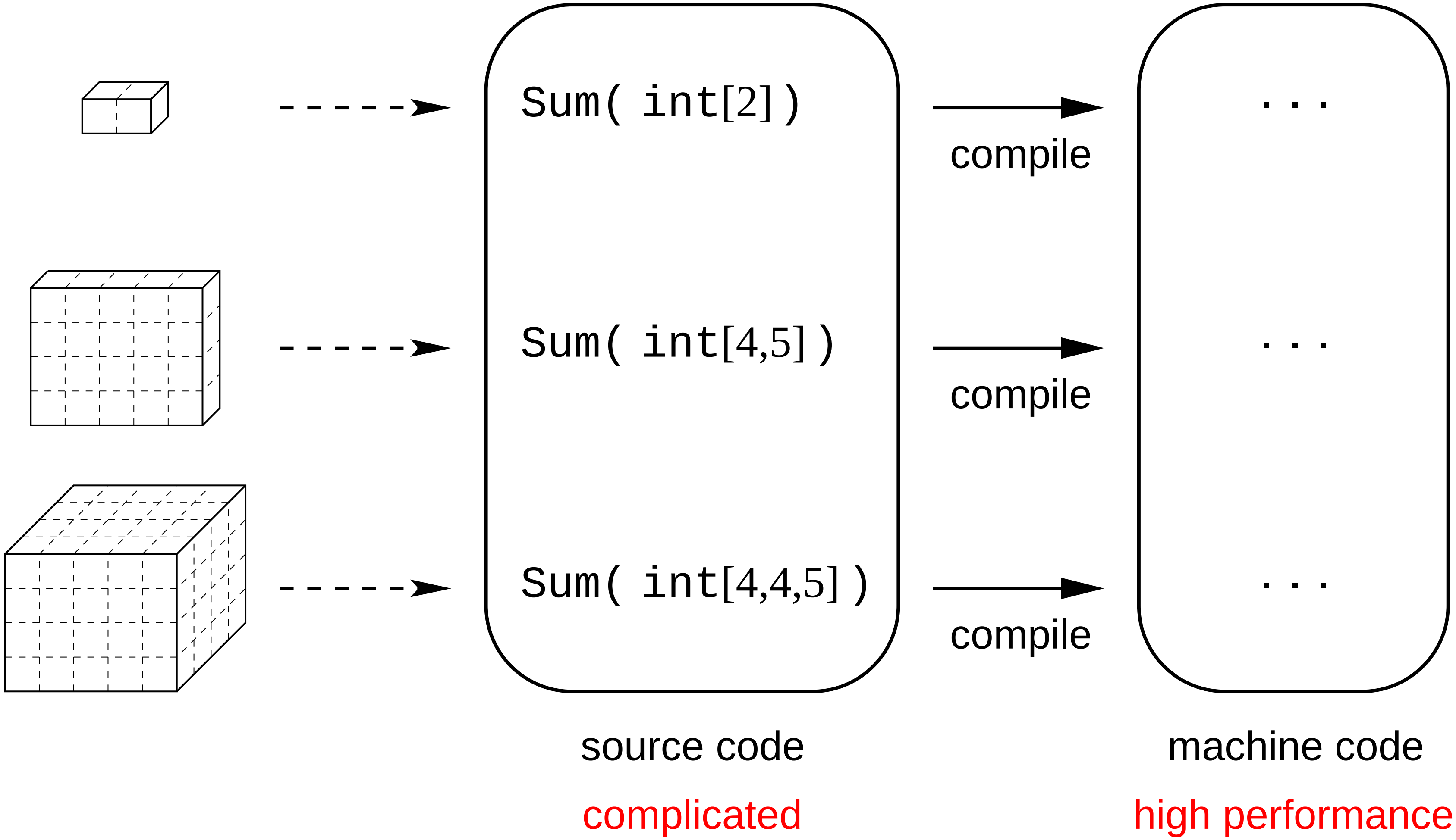
A Compiler Backend for Generic Programming with Arrays

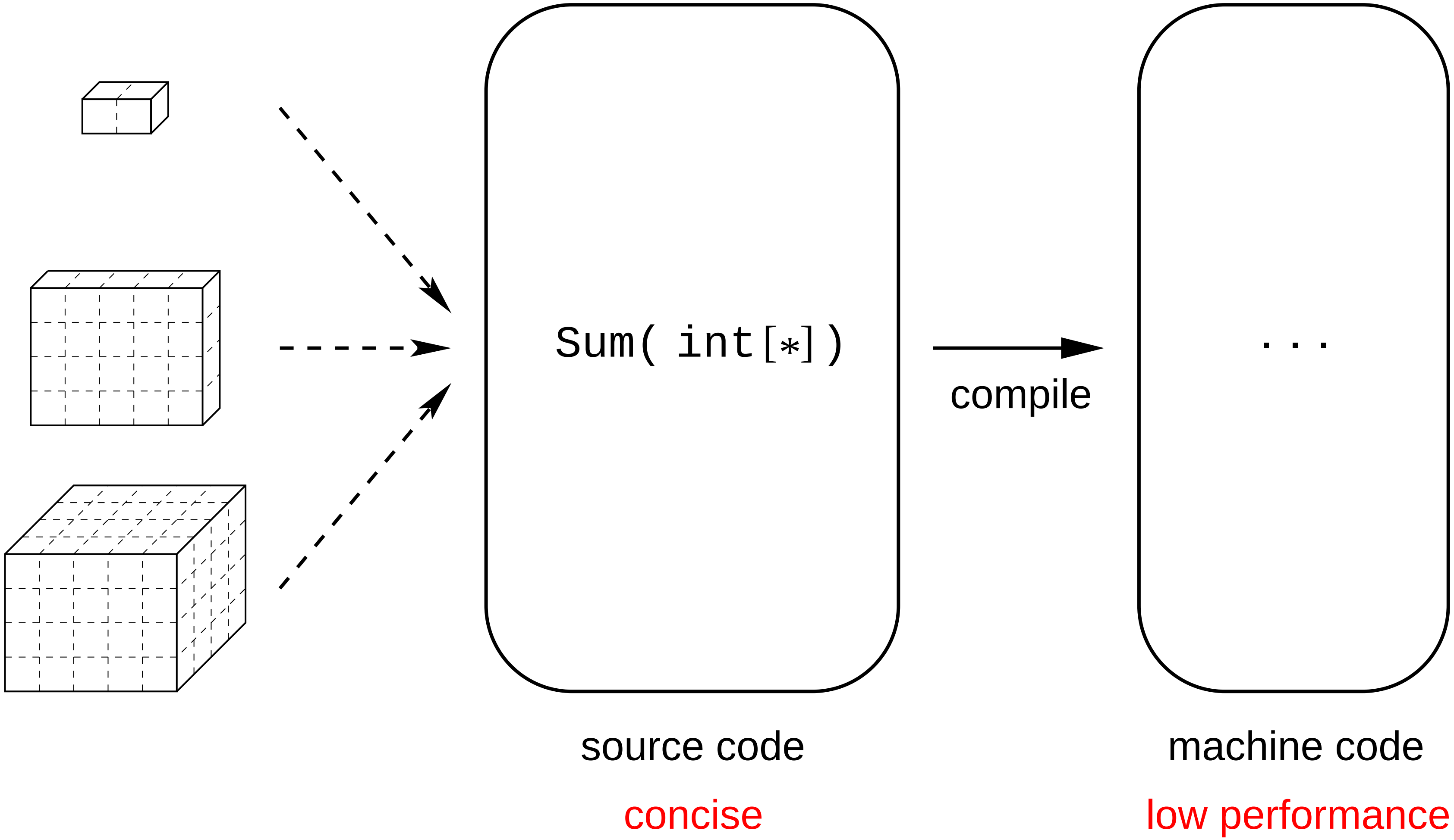
Disputation

Dietmar Kreye
University of Kiel, Germany

Programming with Arrays



Generic Programming with Arrays



Dilemma of Array Programming

- Classical Approach (FORTRAN, C)
  - [-] low abstraction level $\longrightarrow$ large and complicated programs
  - [+] high runtime performance

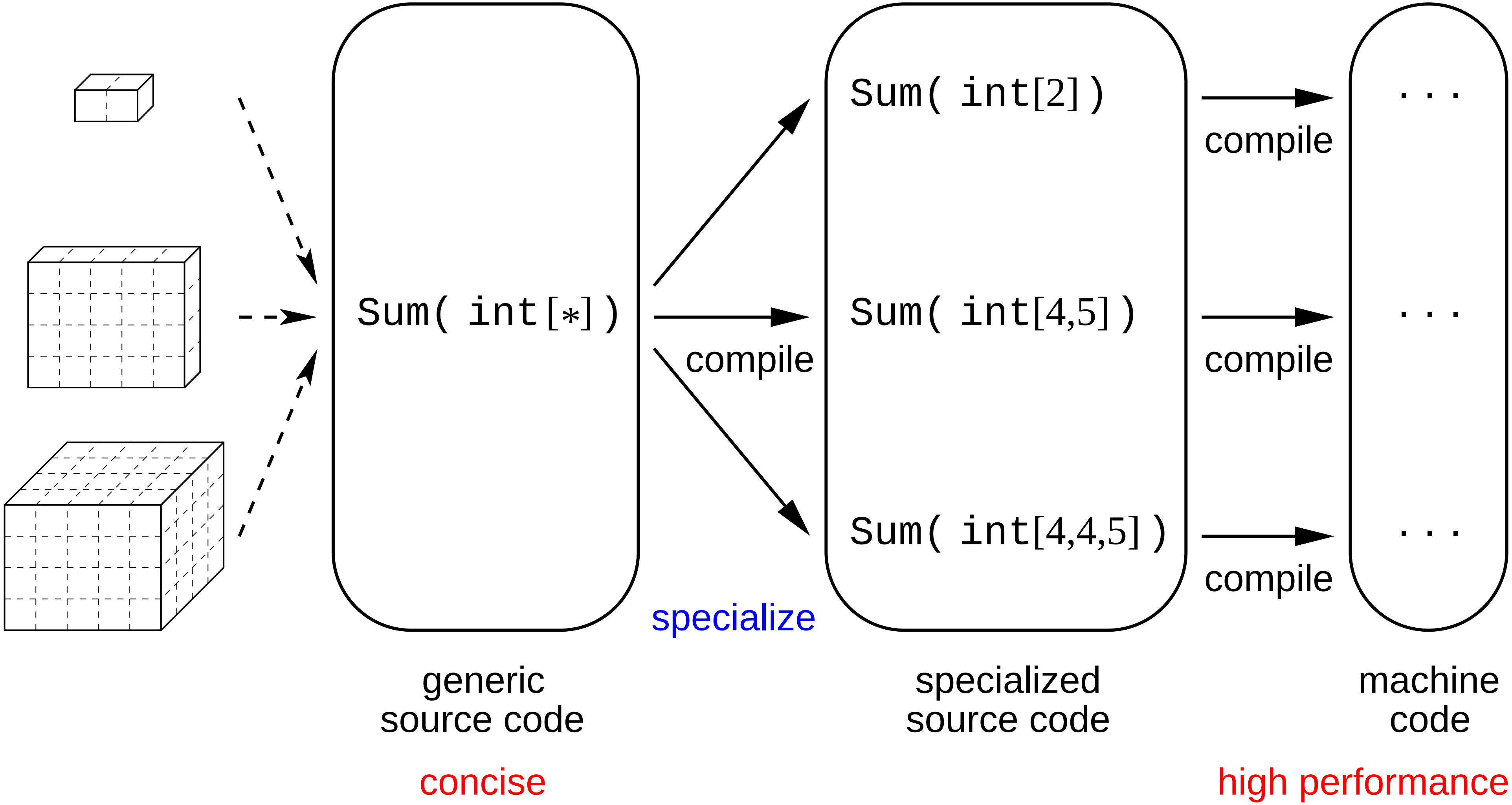
- Generic Approach (APL)
  - [+] high abstraction level $\longrightarrow$ small and concise programs
  - [-] low runtime performance

Neither of these approaches is satisfactory

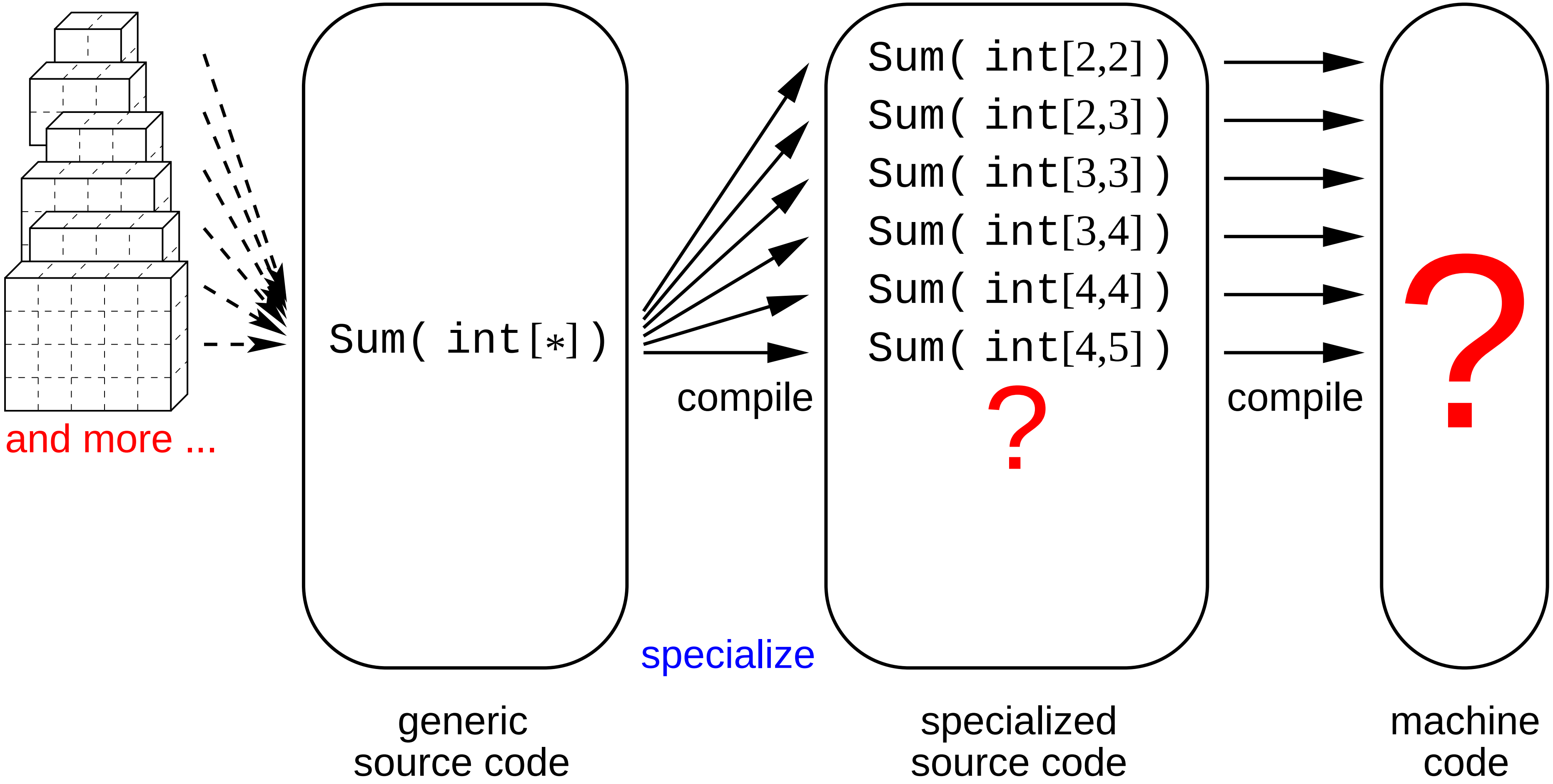
$\Longrightarrow$ Can advantages of both be combined?

The SAC Approach

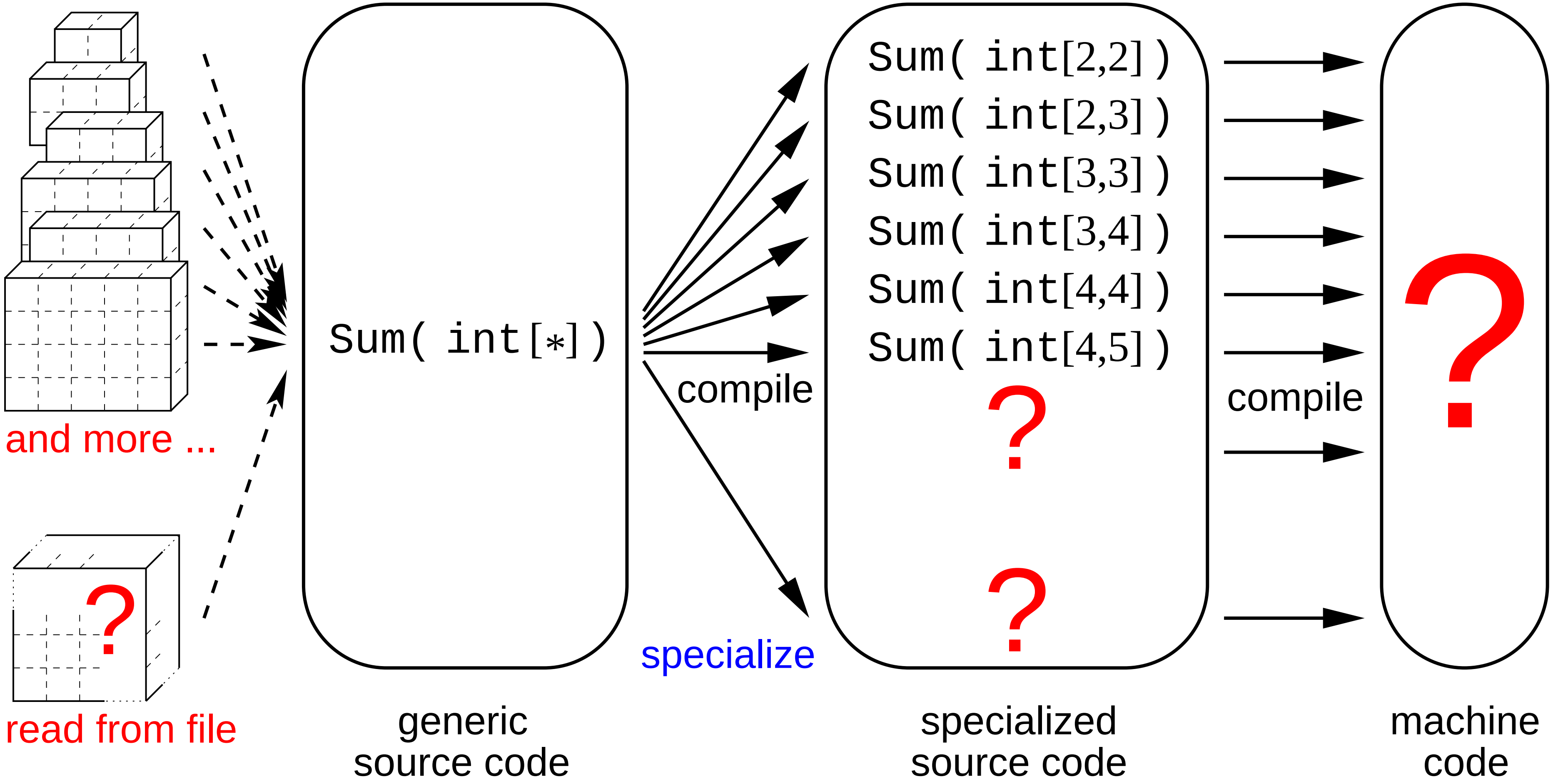
SAC: Functional array programming language based on C syntax



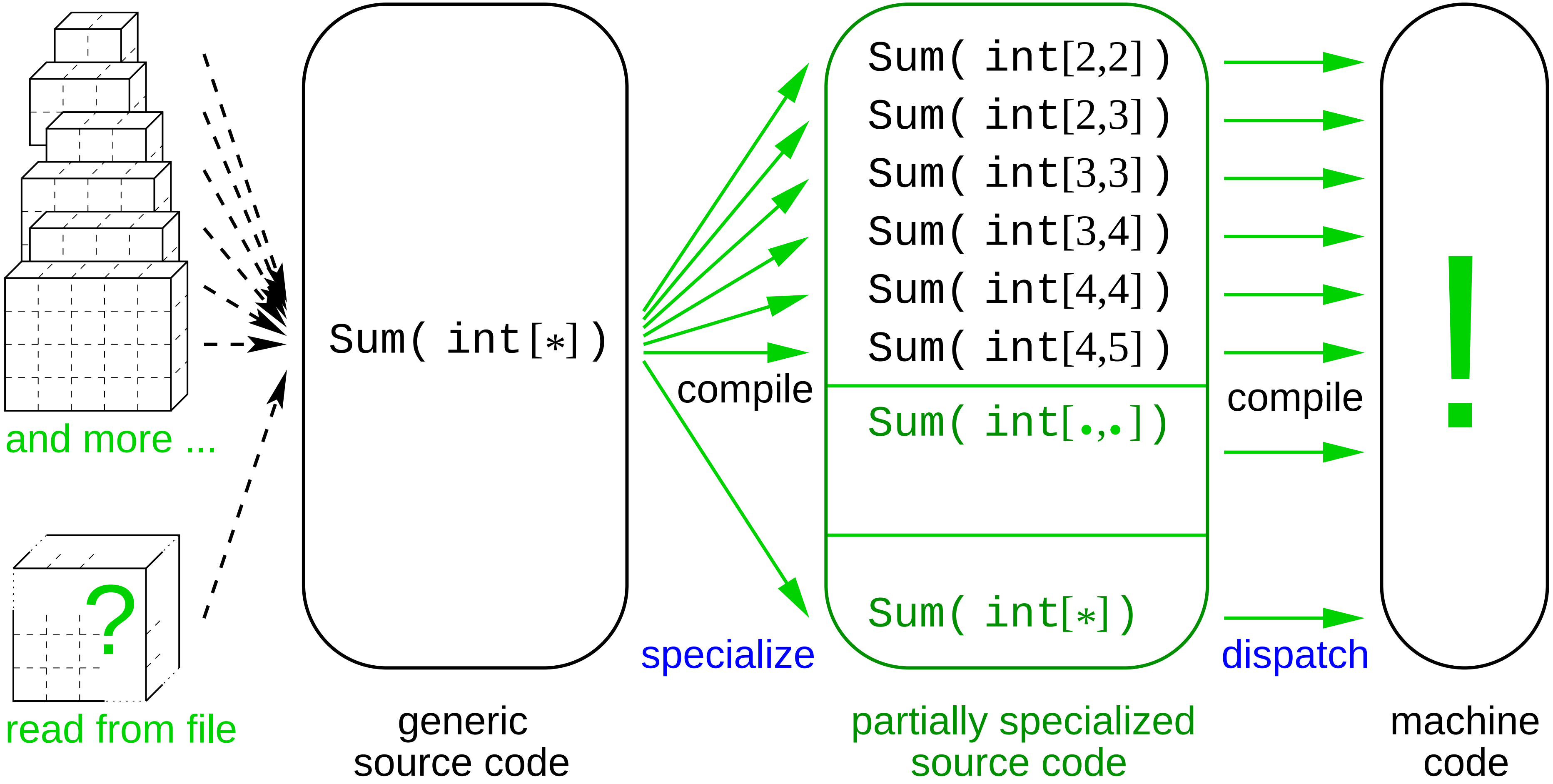
Limits of the SAC Approach



Limits of the SAC Approach



New SAC Approach



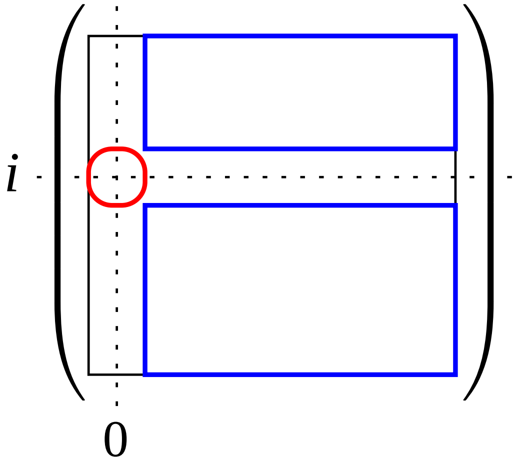
Example: Determinant of a 2-dimensional Array

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

```
Det( int[2,2] A)
{
  ...
}
```

Laplace expansion along the first column:

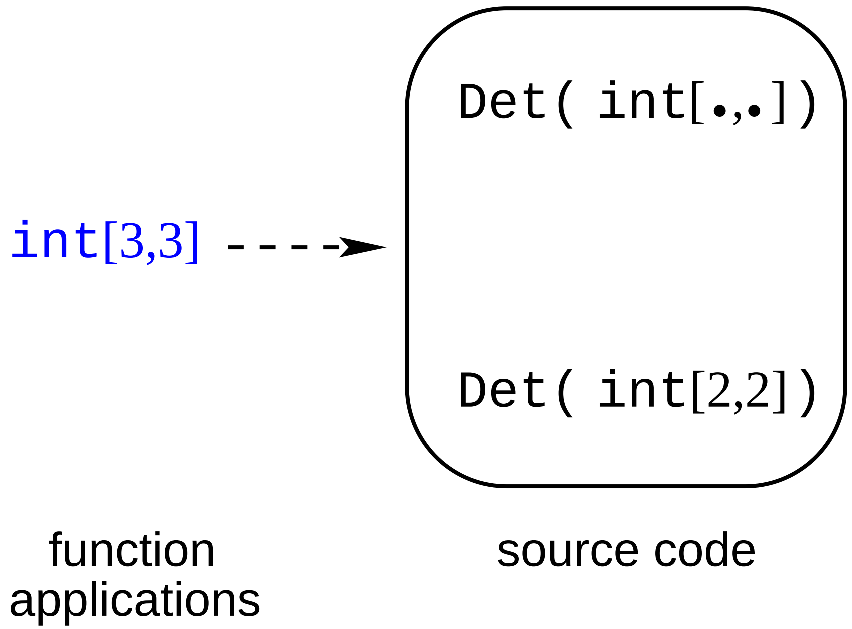
$$\det(A) = \sum_{i=0}^{n-1} (-1)^i \cdot A_{i0} \cdot \det(\mathfrak{A}_{i0})$$



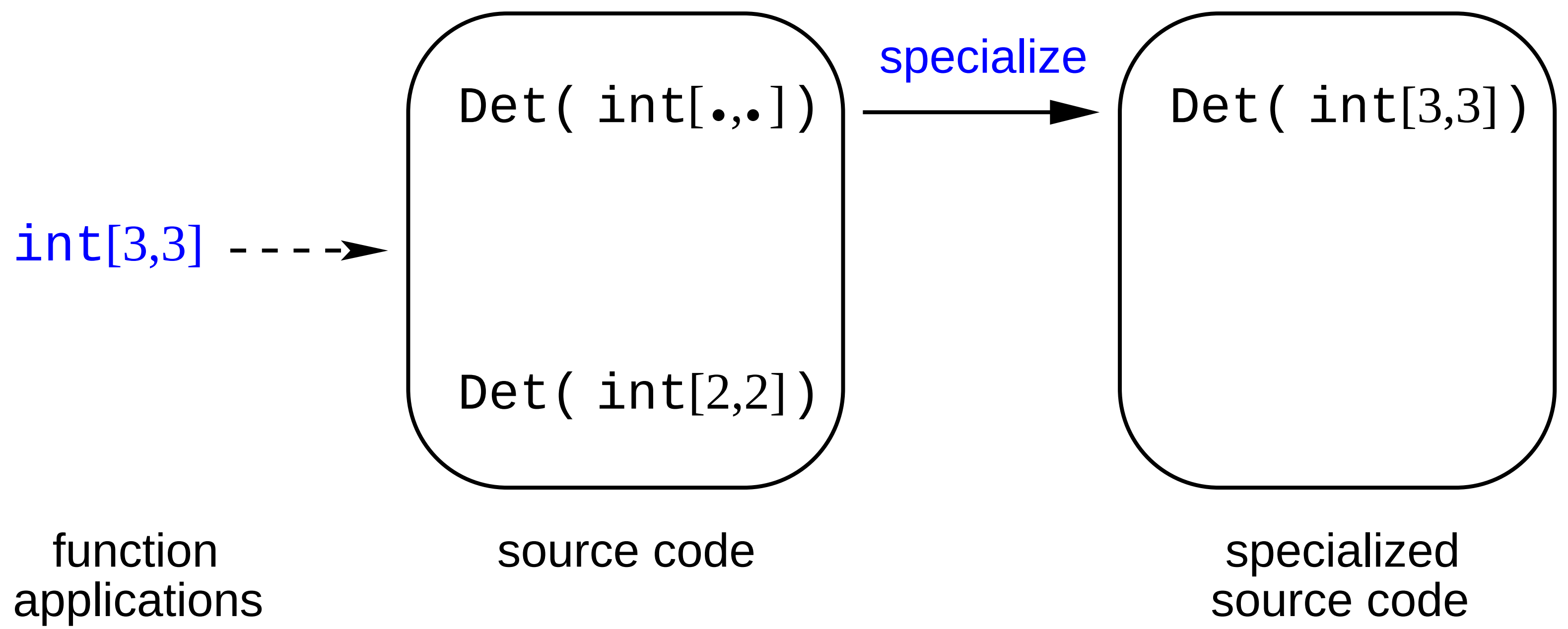

```
Det( int[.,.] A)
{
  𝔄_i0 = ... A ...;
  ... Det( 𝔄_i0) ...
}
```

Function Overloading

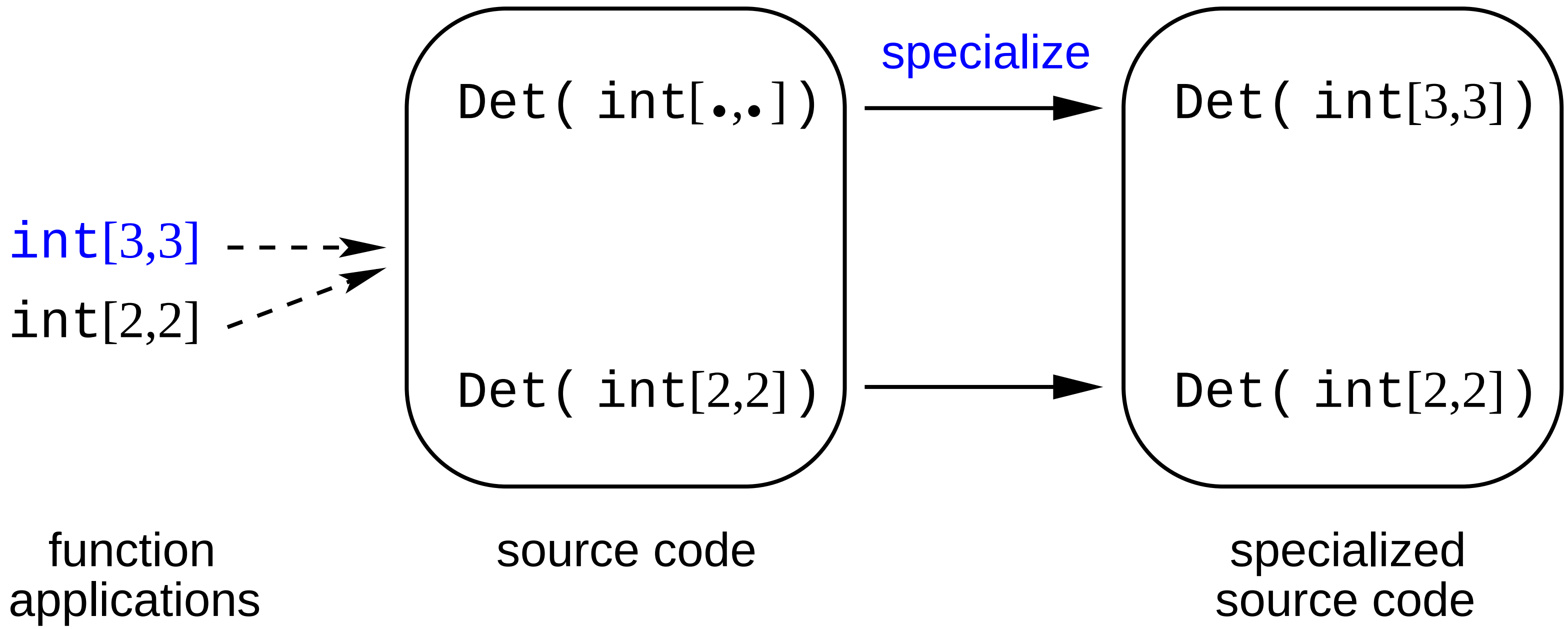
Function Specialization



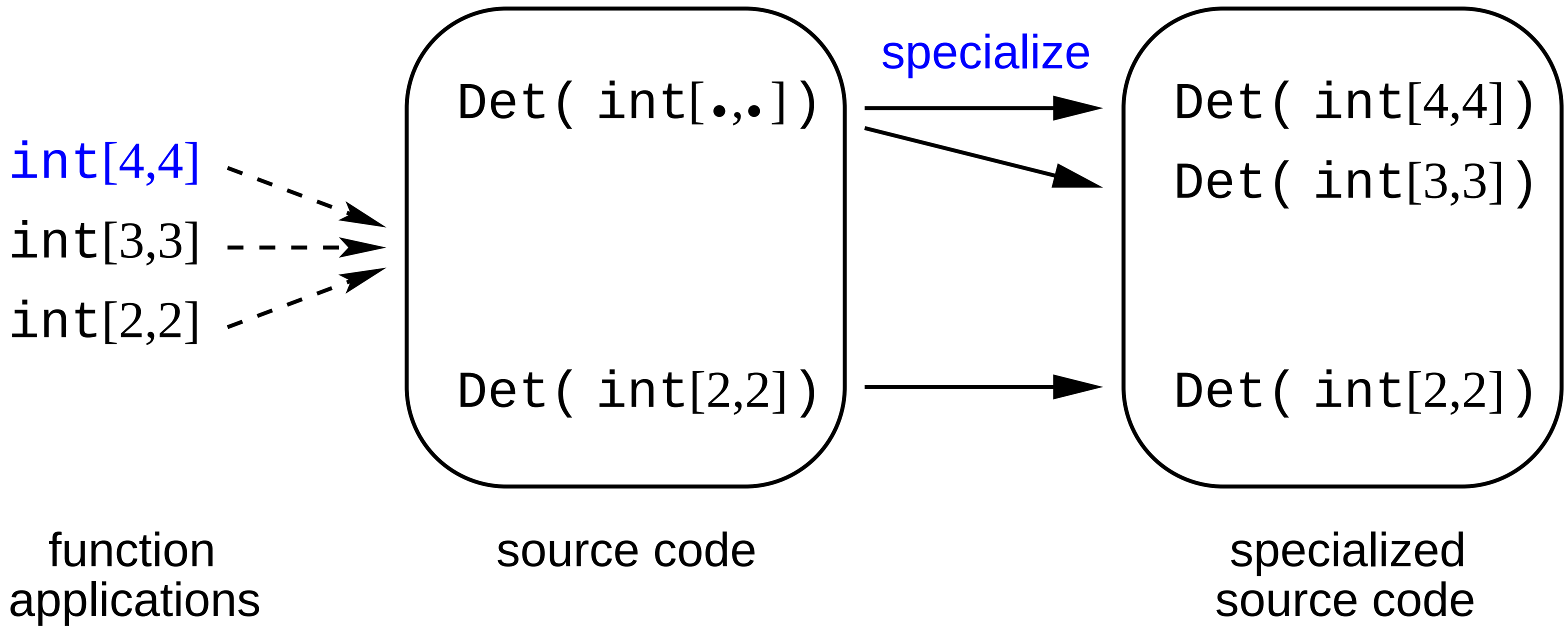
Function Specialization



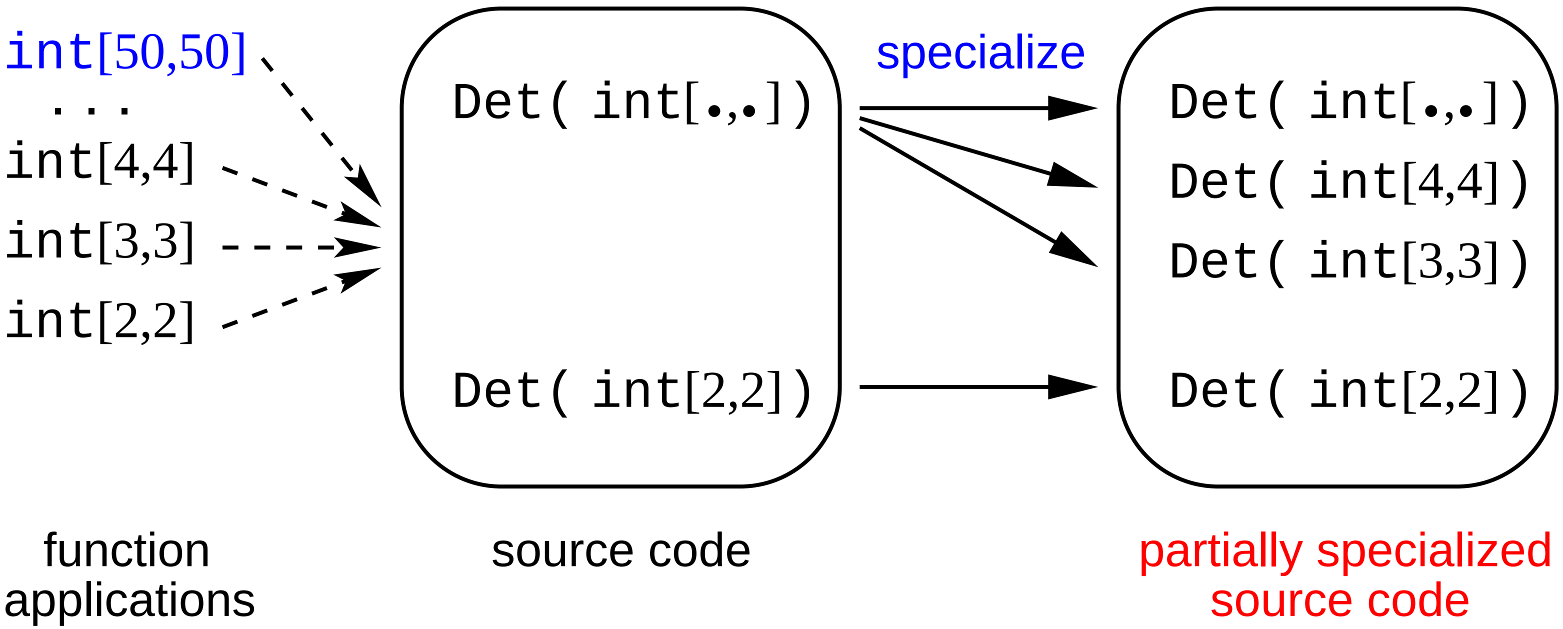
Function Specialization



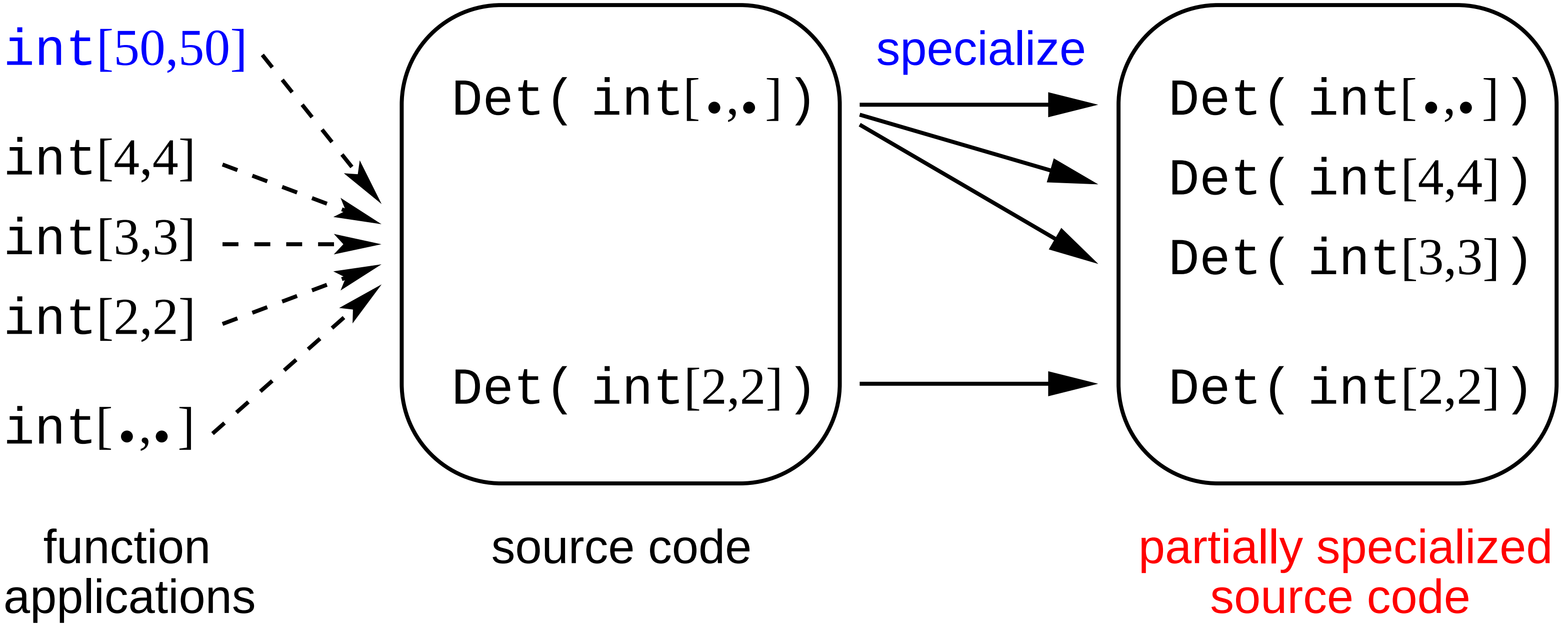
Function Specialization



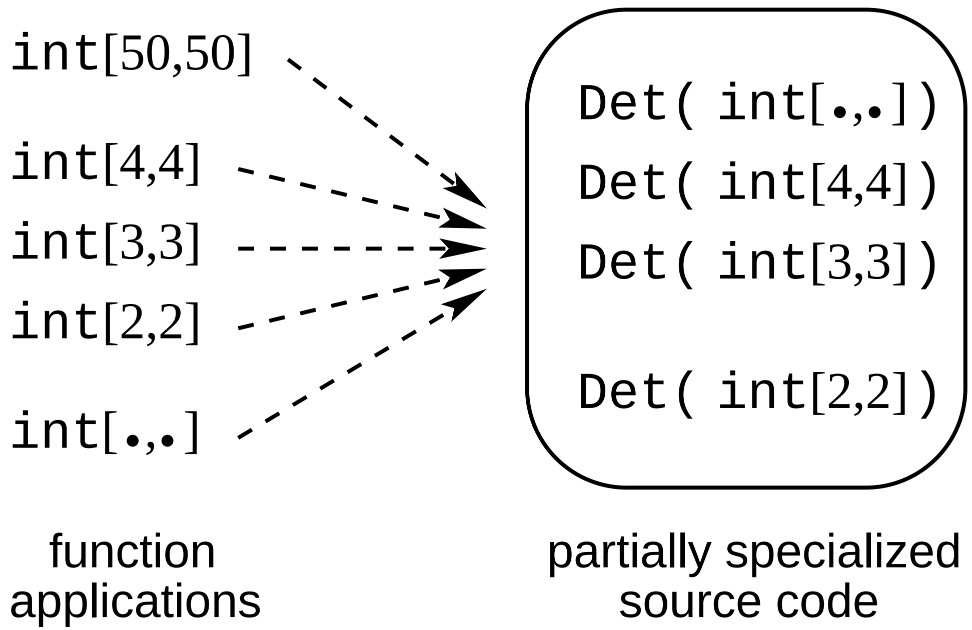
Function Specialization



Function Specialization

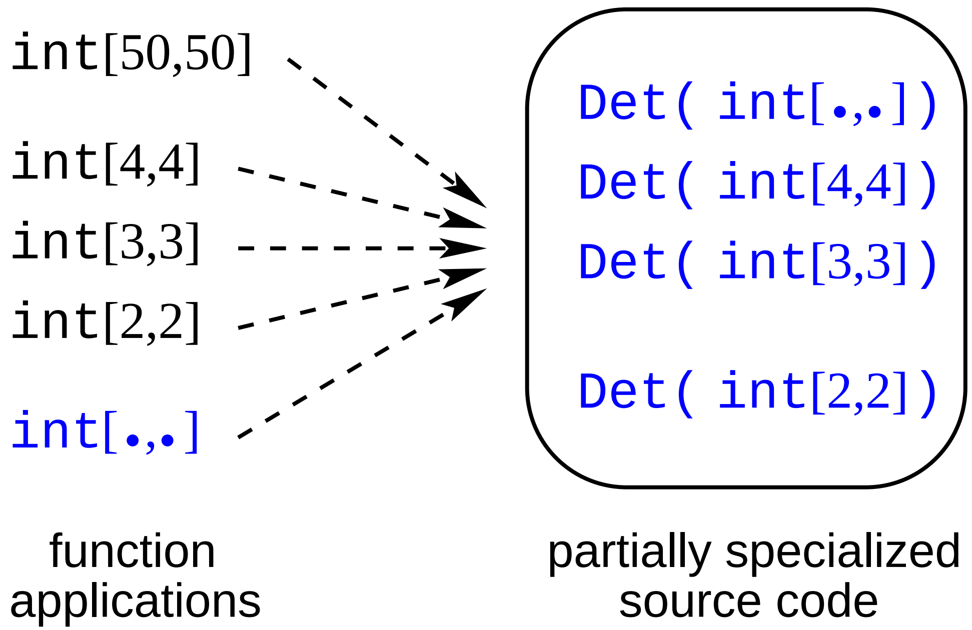


Dispatch of Function Applications



Shape-specific argument $\longrightarrow$ Static dispatch

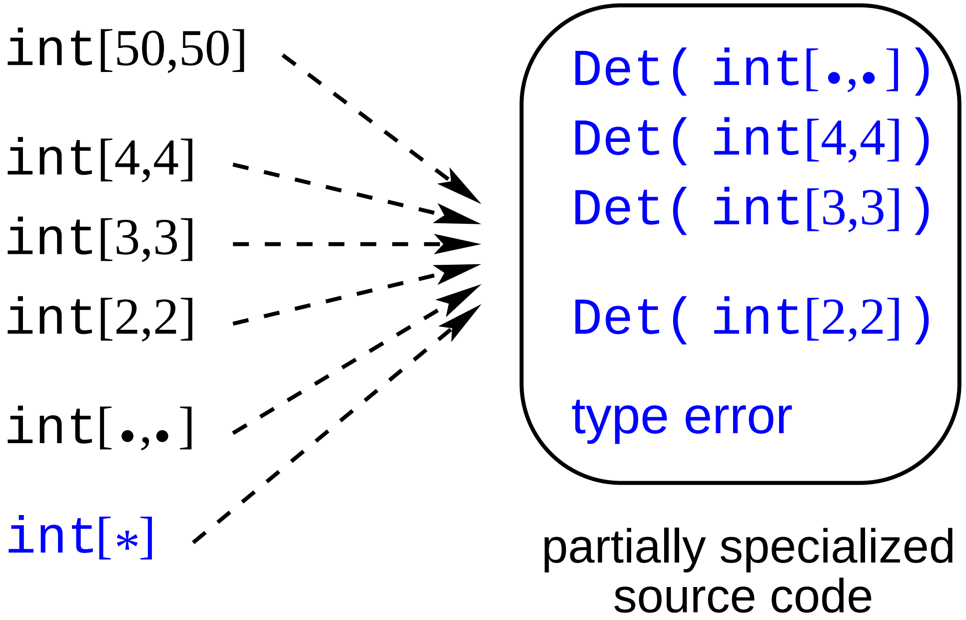
Dispatch of Function Applications



Non-shape-specific argument $\longrightarrow$ Dynamic dispatch

- Necessary for overloaded versions to get correct results
- Recommended for specialized versions to get utmost runtime performance

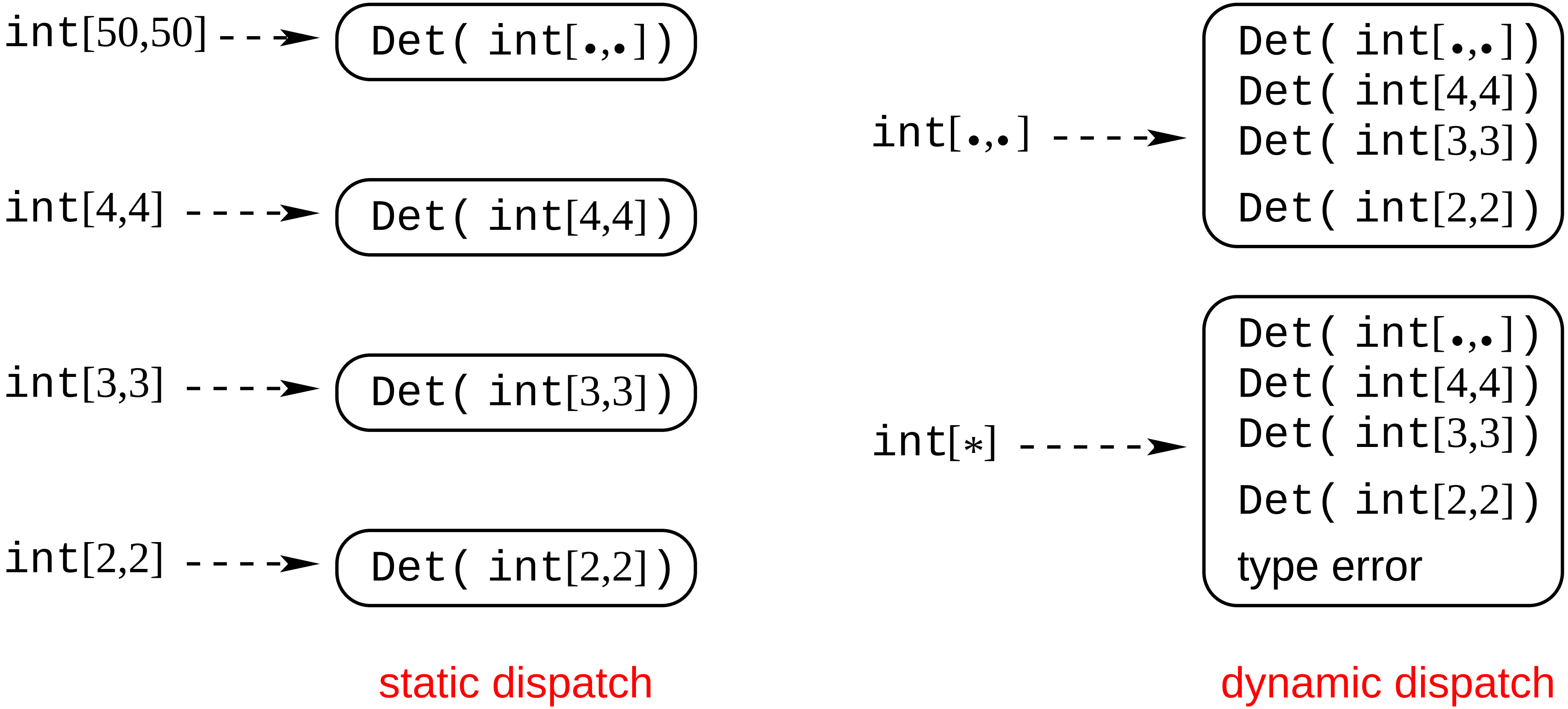
Dispatch of Function Applications



Non-shape-specific argument $\longrightarrow$ Dynamic dispatch

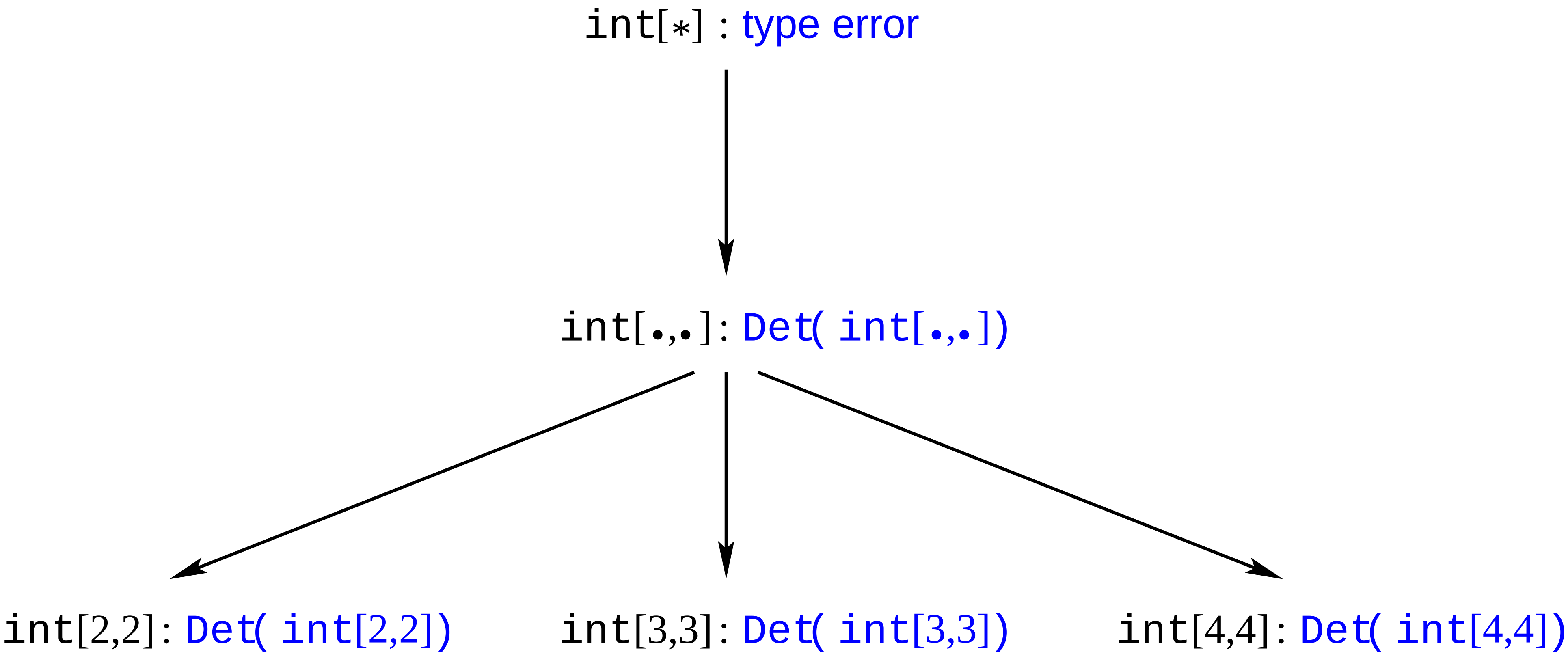
- Necessary for overloaded versions to get correct results
- Recommended for specialized versions to get utmost runtime performance

Hybrid Dispatch: Intended Results

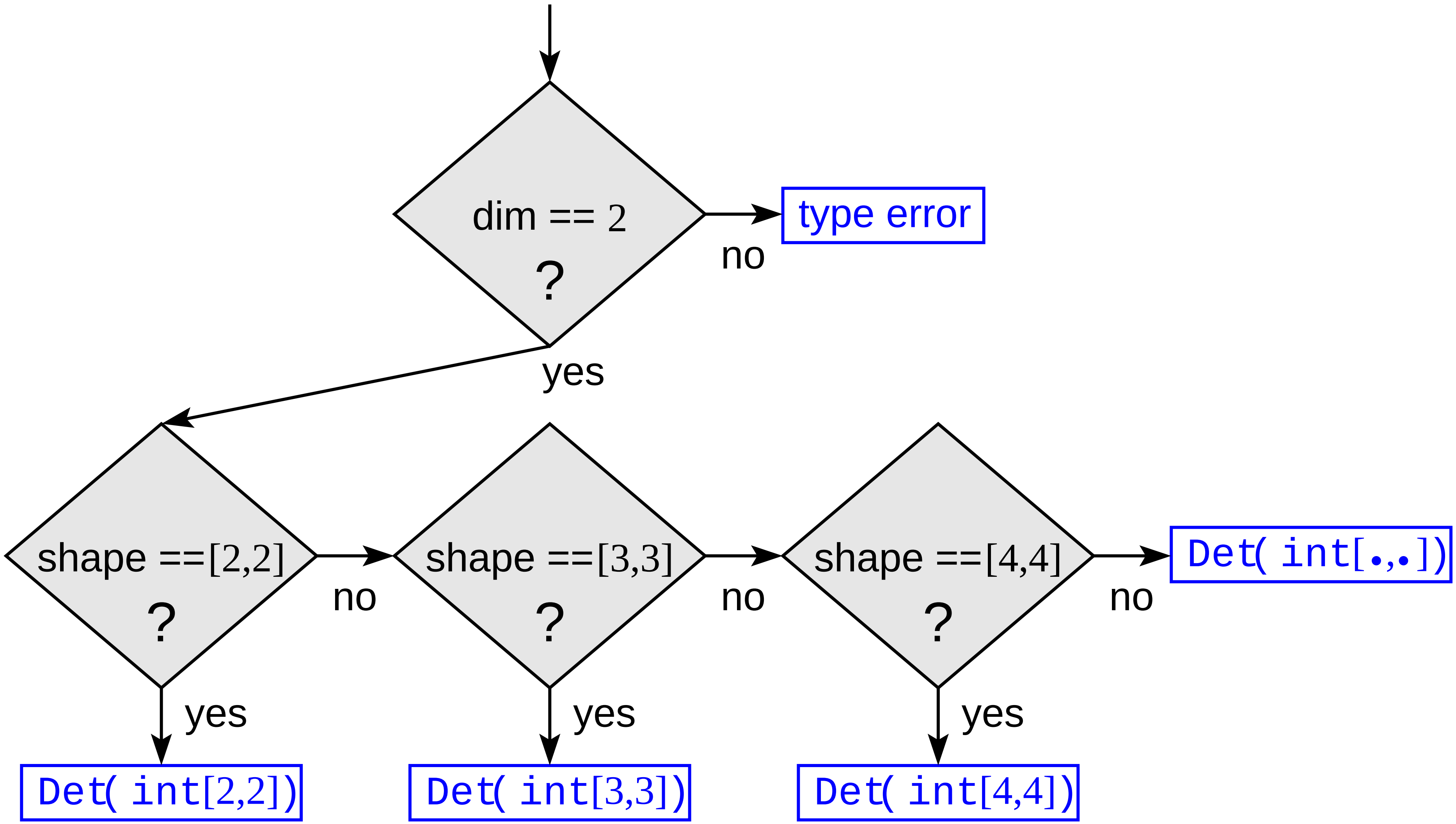


$\Longrightarrow$ Hybrid dispatch: As static as possible, but as dynamic as necessary

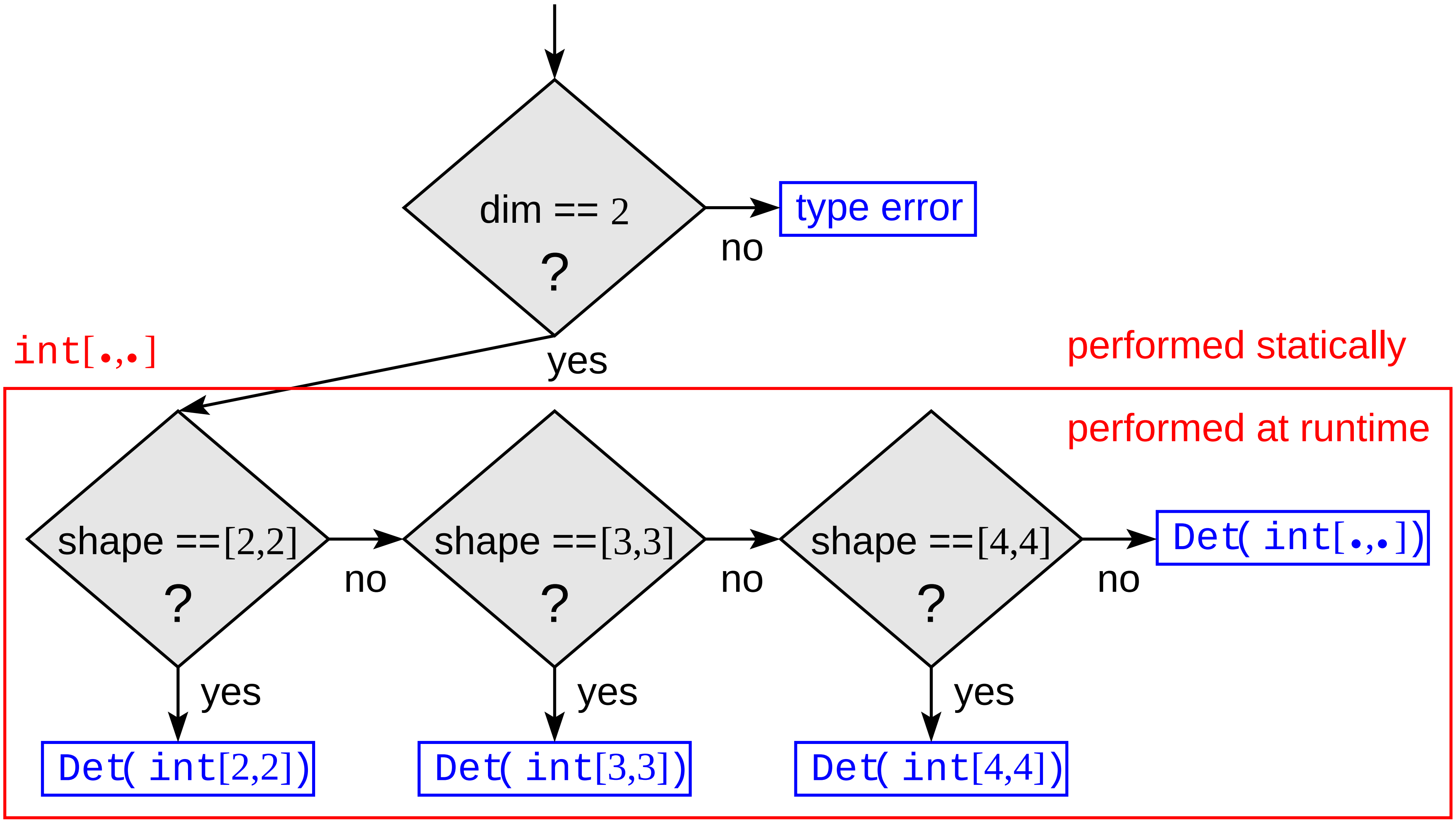
Hybrid Dispatch: Decision Tree



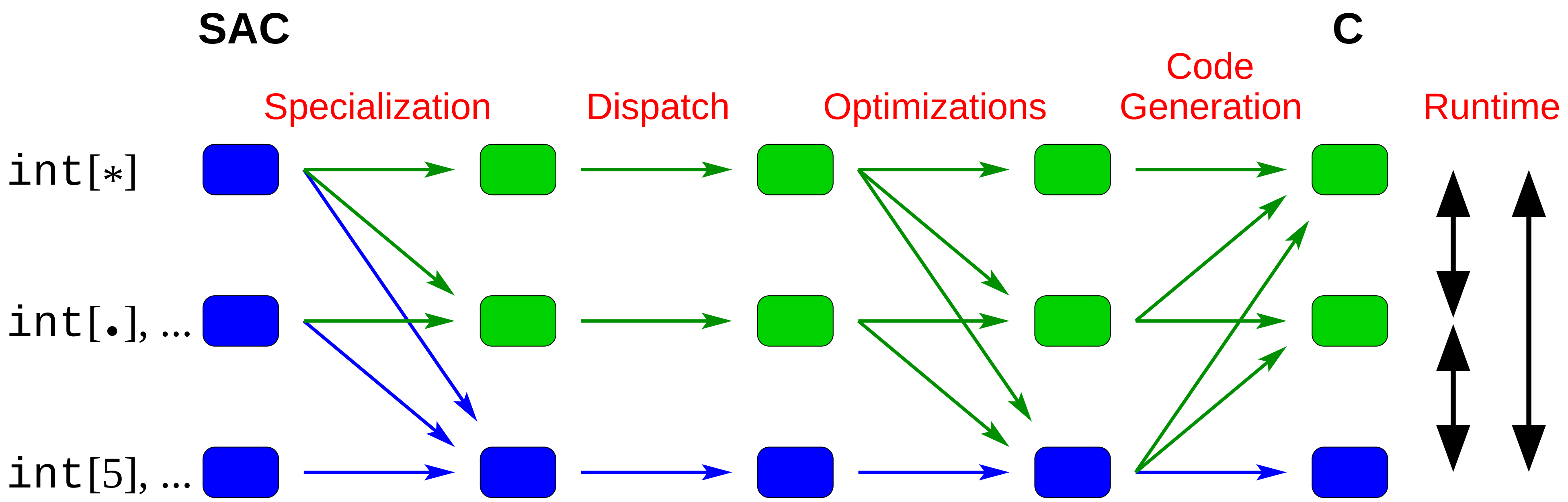
Hybrid Dispatch: Algorithm



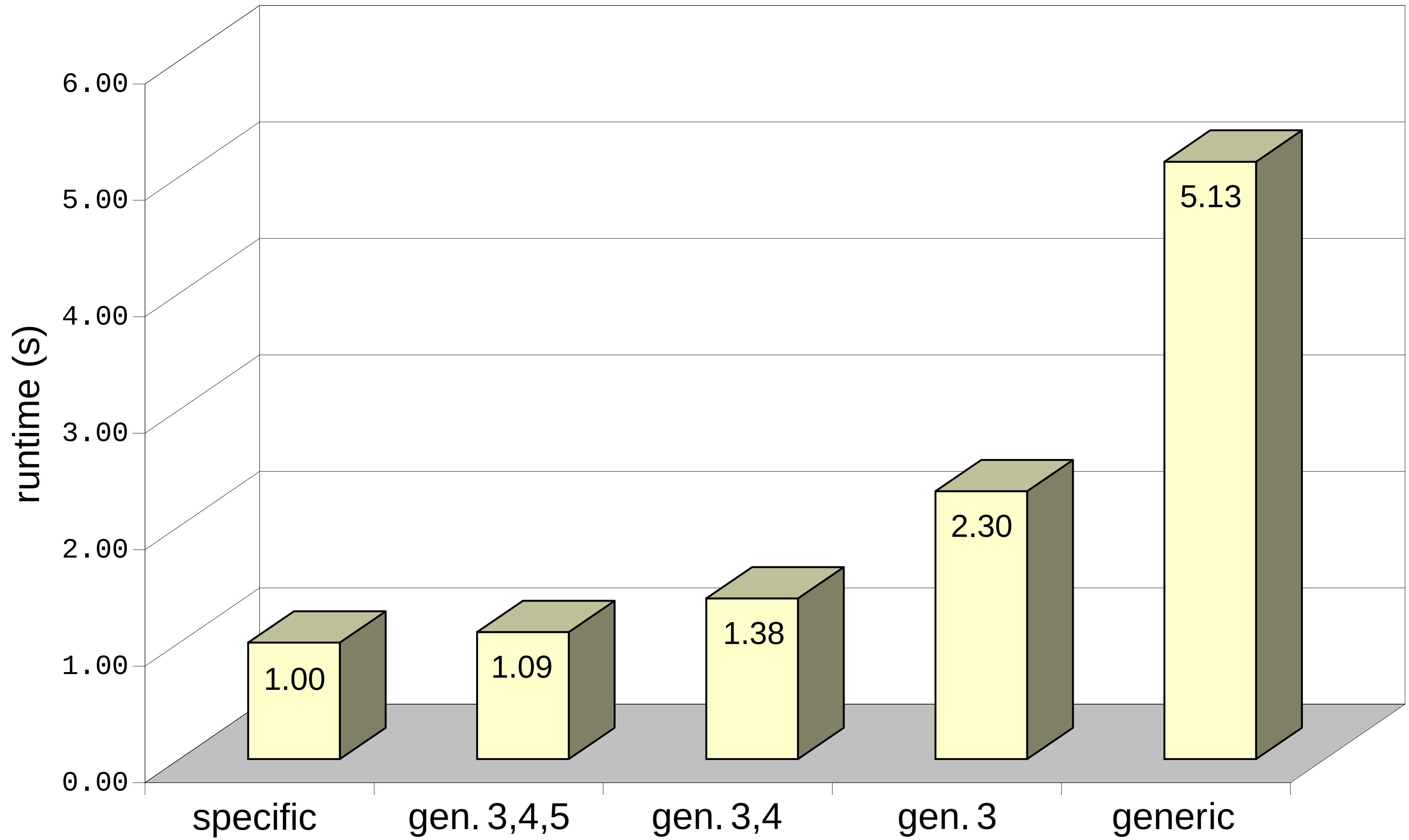
Hybrid Dispatch: Algorithm



The Compilation Process



Runtime Performance: Determinant of a 10×10 Array



Conclusions and Future Work

Conclusions:

Dilemma has been solved:

Generic programming and high runtime performance

- Preserves excellent runtimes of fully specialized code
- Avoids code explosion due to unlimited specialization
- Allows generic input data
- Allows separate compilation (library functions)

Future Work:

- Better specialization strategy
- More optimizations on non-shape-specific arrays