Developing Fast Code Easily

Ulrich Drepper

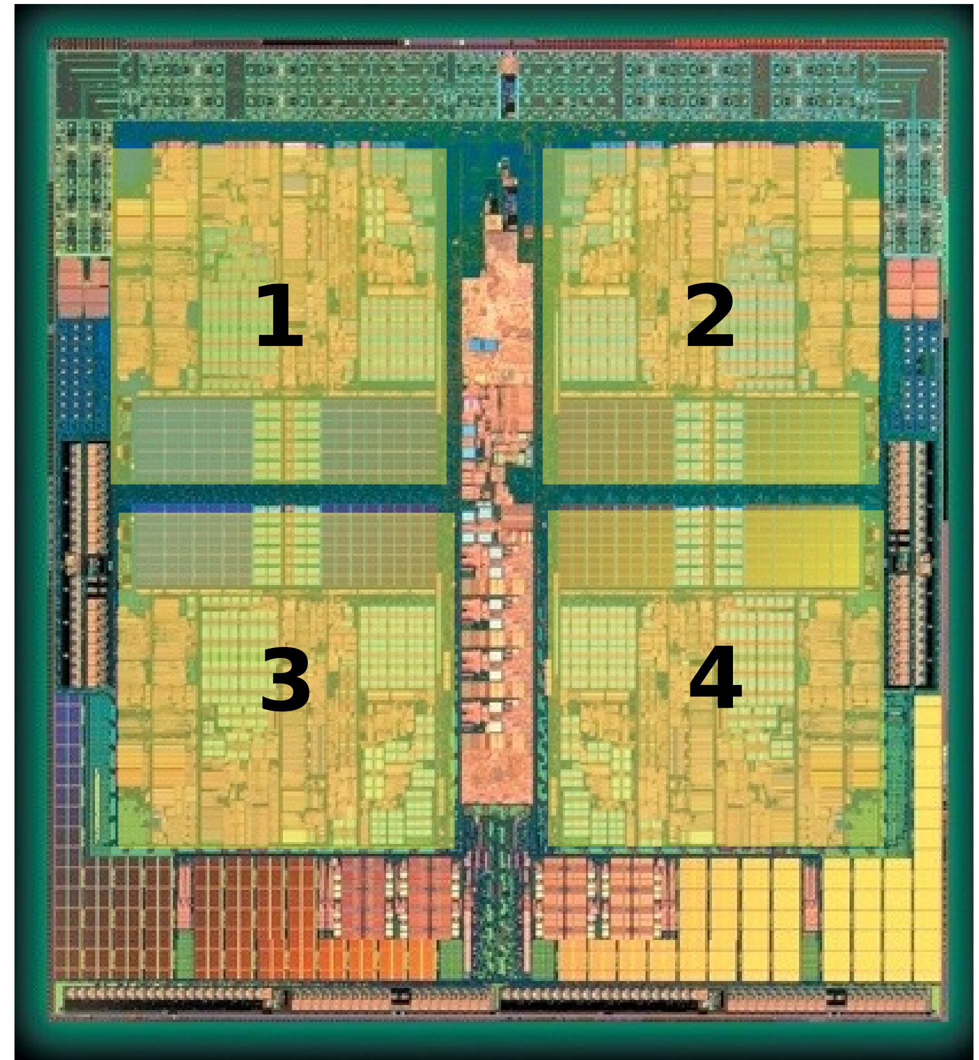
redhat.

Hardware Complexity

Hardware cannot completely hide the complexity anymore

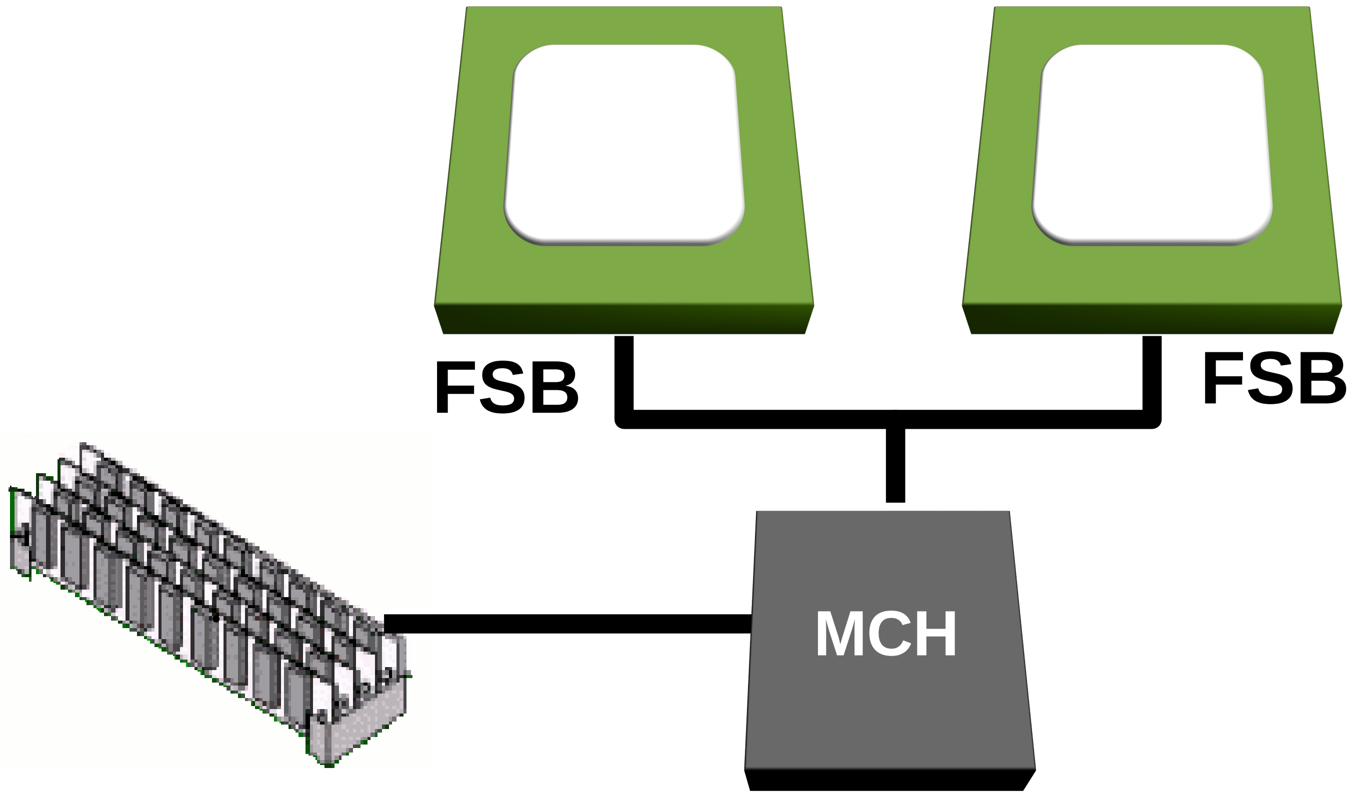
Multi-Core

- For power reasons:
  - Horizontal scaling
  - Now doubling of cores every 18 months
- Complex relationship
  - Parts of the CPU are shared by groups of the cores



redhat

Memory Bandwidth

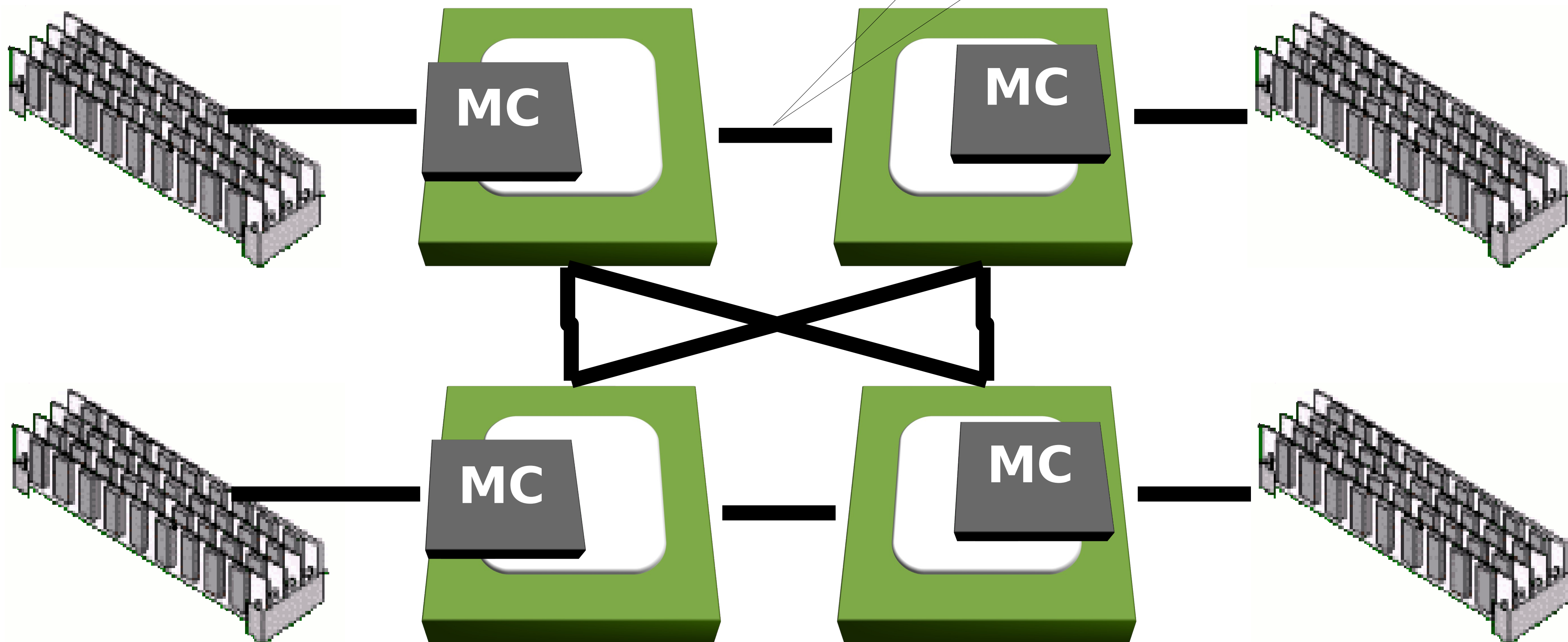


redhat

Memory Bandwidth

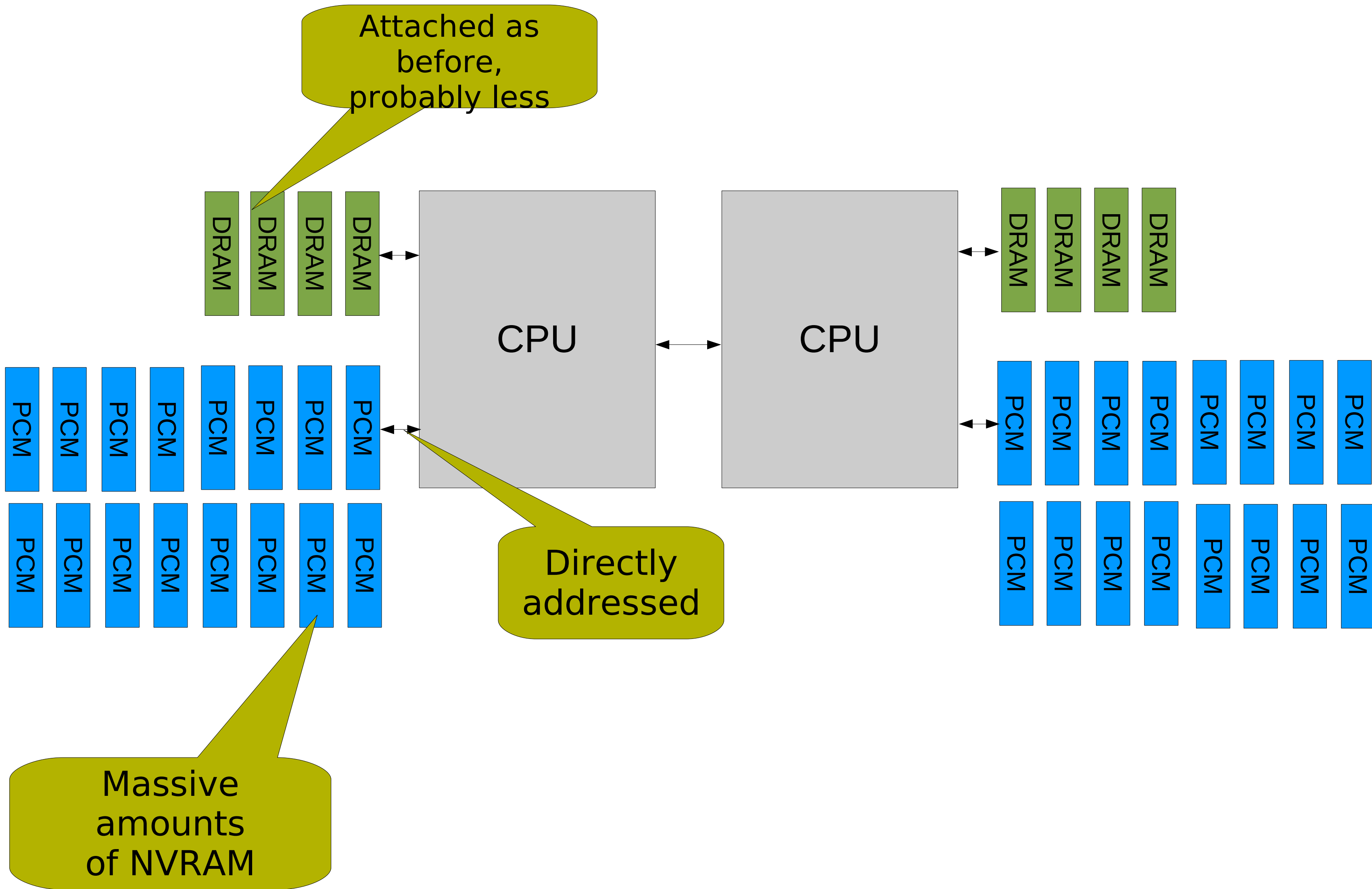
Intel: QPI
AMD: Hypertransport

- Non Uniform Memory Architecture (NUMA)



redhat

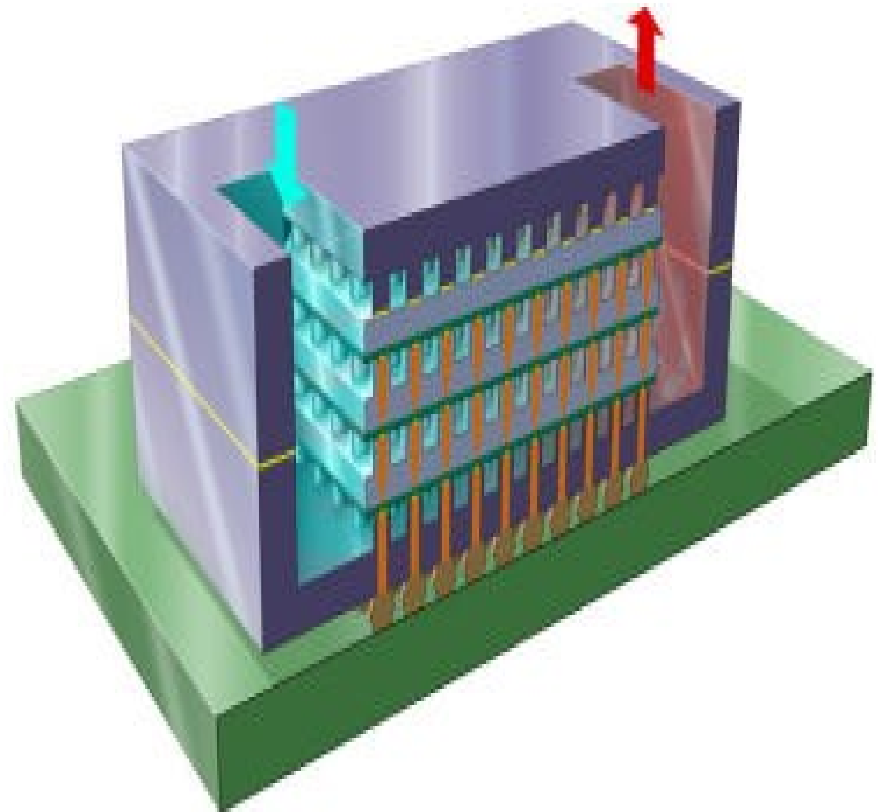
Extended RAM



redhat

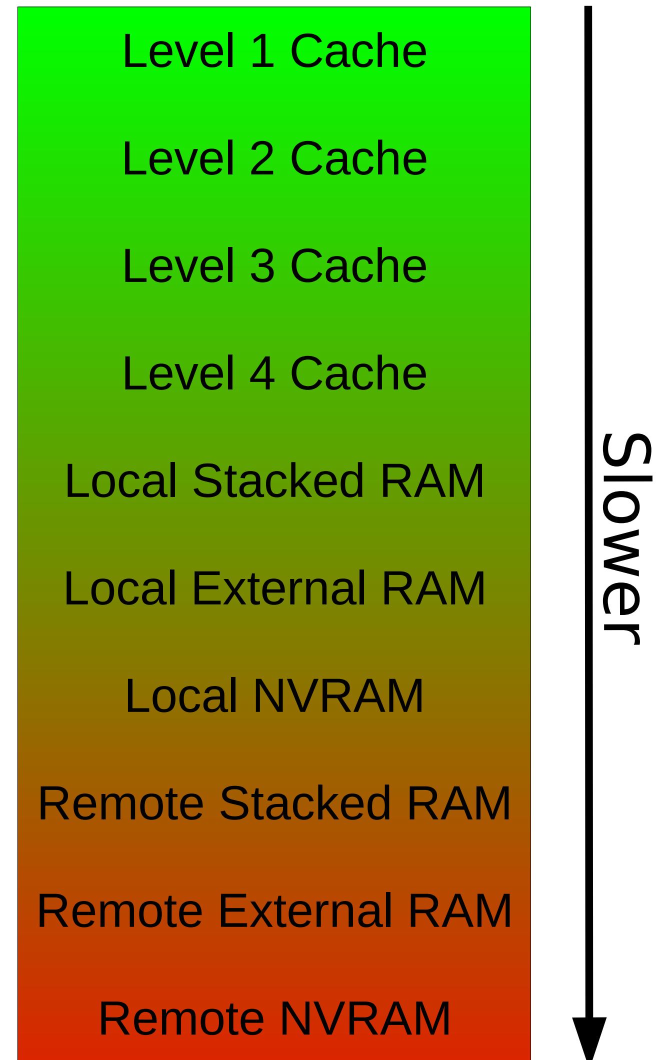
Stacked Dies

- Connect dies directly
  - Higher line count
  - Less hardware involved
  - Much faster
- It seems possible
  - IBM proposes to use water cooling for the stacked dies
  - Might happen as soon as 2010



What Does This Mean?

- Putting it all together:
  - Lots of work in the kernel to get rid of assumptions on rotating media
  - Significantly more complex memory hierarchy
    - Needs to be handled automatically
    - Programmers unable to cope with all this explicitly
  - Possibly more compartmentalization and special support for small cache coherency domains



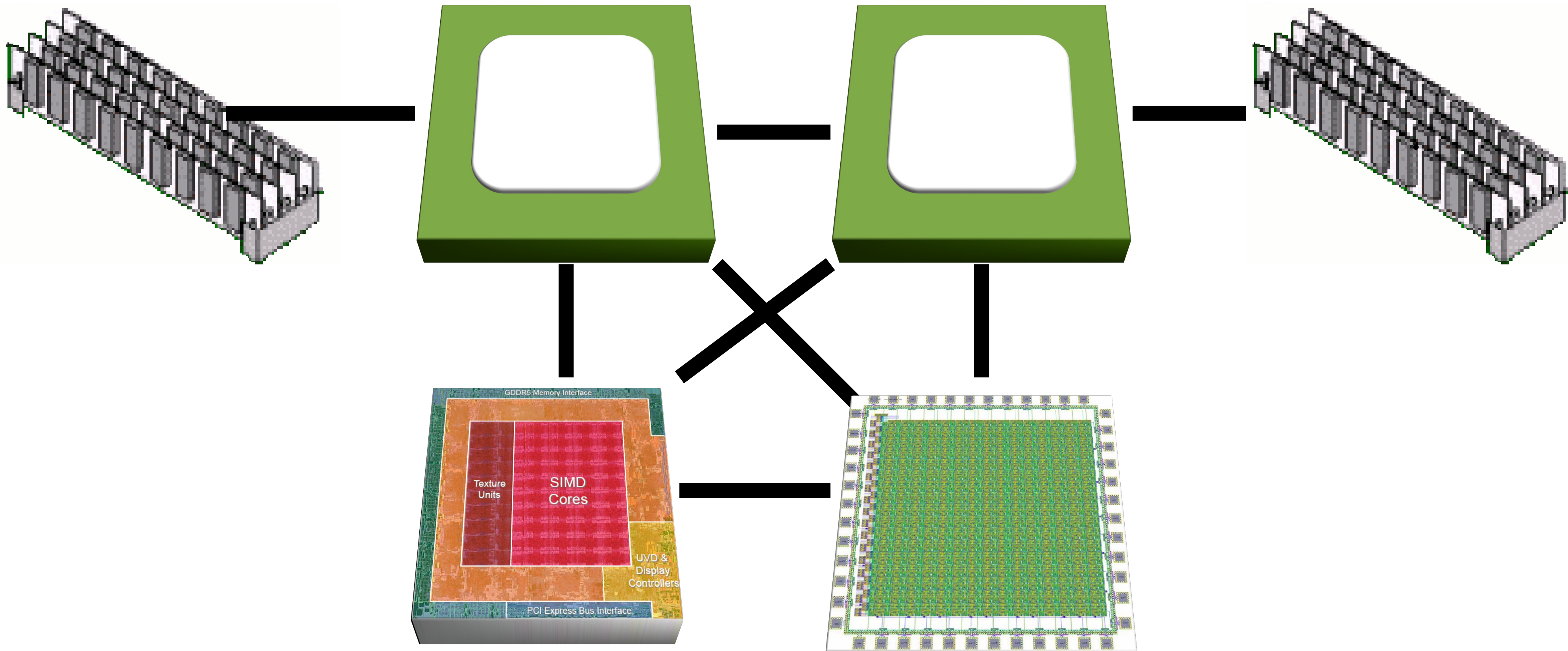
redhat

Processor Heterogeneity

- Many core systems sometimes inefficient
  - Not all tasks need all the features
    - e.g., only streaming in some threads
- Heterogeneous processors
  - Cores with different capabilities
- Co-Processors/Accelerators
  - GPGPUs
  - ASICs, FPGAs
  - Connected to CPUs, not PCI bus

redhat

Connecting Co-Processors



Goal of Programming

Write with as little effort programs
that run as fast as possible
(and/or use as few resources as possible)

redhat

Development Process

- Three types of programmers
  - Application developer
    - No special knowledge, at or below average
  - Library designer
    - Designs reusable interfaces, deep language knowledge
  - Optimization “Tiger Team”
    - Profiles code and modifies library code to take advantage of hardware features
- Library interface
  - Facilitates easy development
  - Allows optimizations

redhat

Development Process

Library Designer

1. Library designers develop interface
2. Basic library implementation
3. Document library interfaces

Application Developer

4. Develop applications using libraries

Optimization Tiger Team

5. For each application
    5.a) Profile, identify hot spots
    5.b) Find hardware to alleviate performance problem
    5.c) Change library to use hardware
    5.d) Recompile application
    5.e) Repeat until performance goal reached

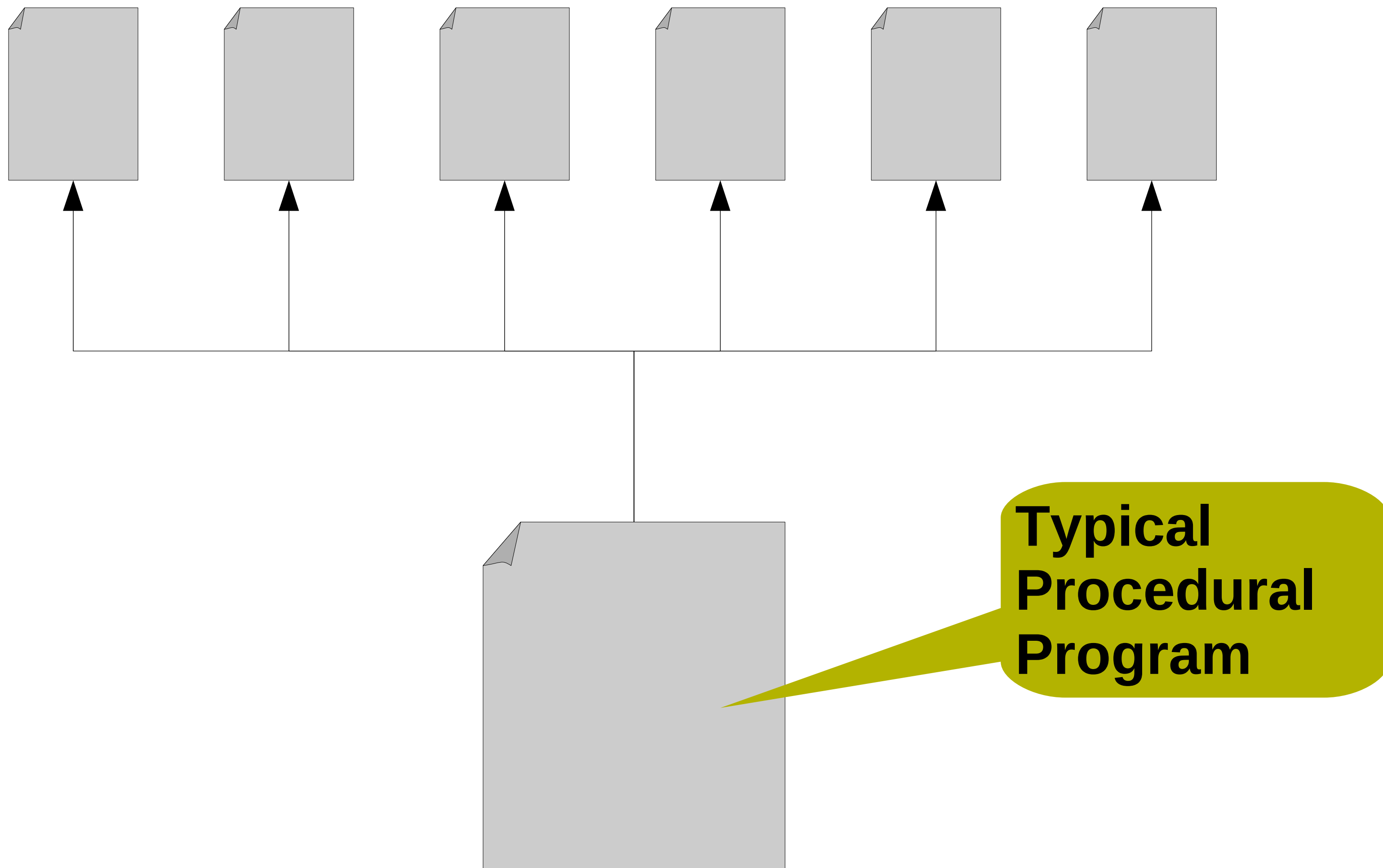
redhat

Requirements

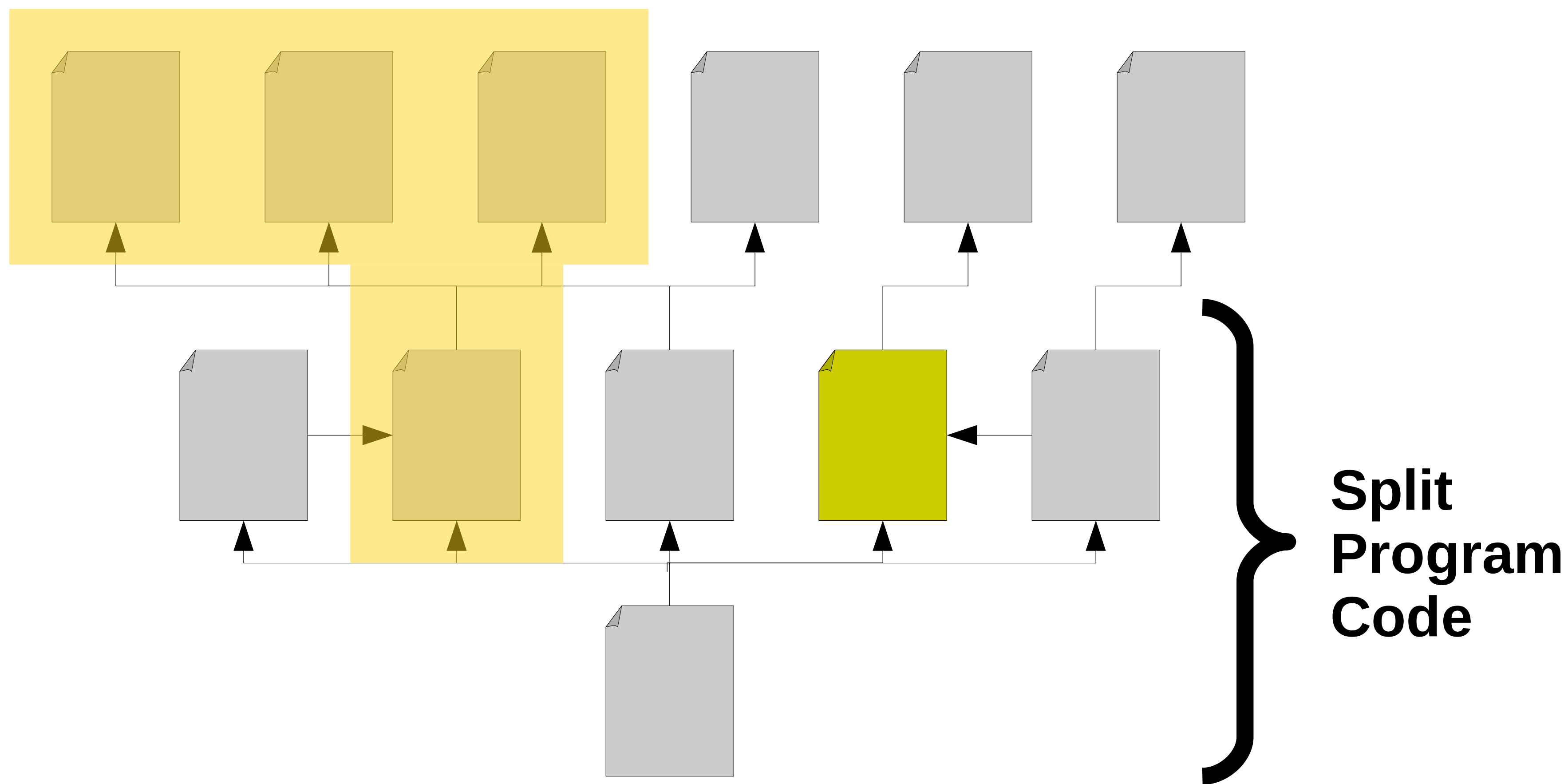
- Library design must allow optimizations
  - No need to change application code
- Enable implementation replacement
  - Pure functions, no side effects
  - Unit testing easily doable
- Matches functional programming style
  - Especially Haskell: single assignment

redhat

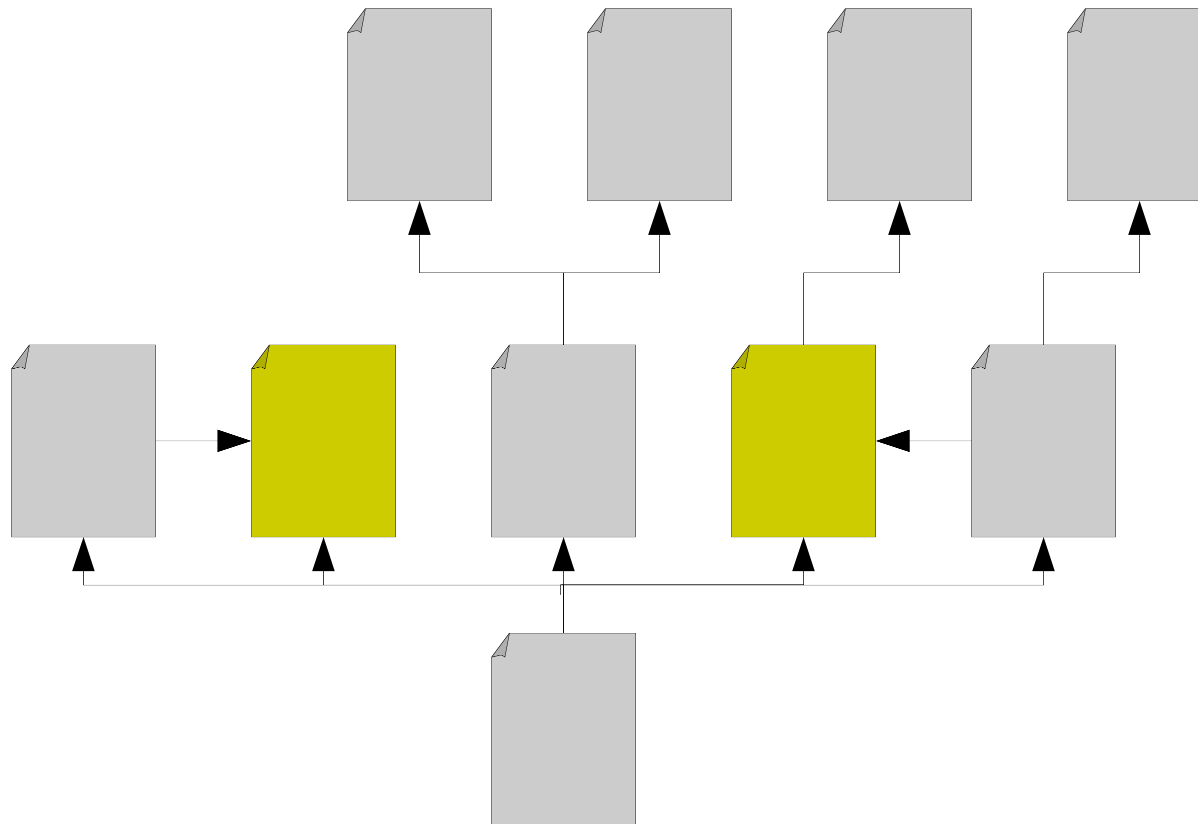
Functional Style



Composability



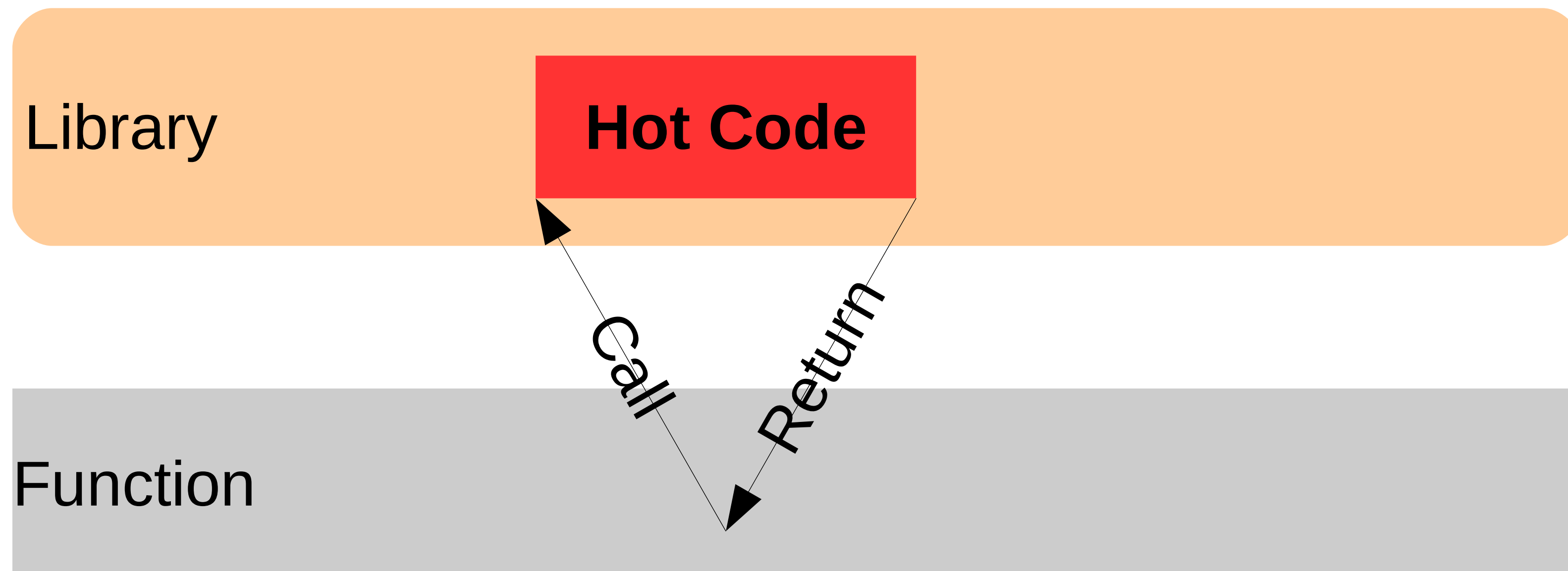
Composability



redhat

Avoiding Multiple Binaries

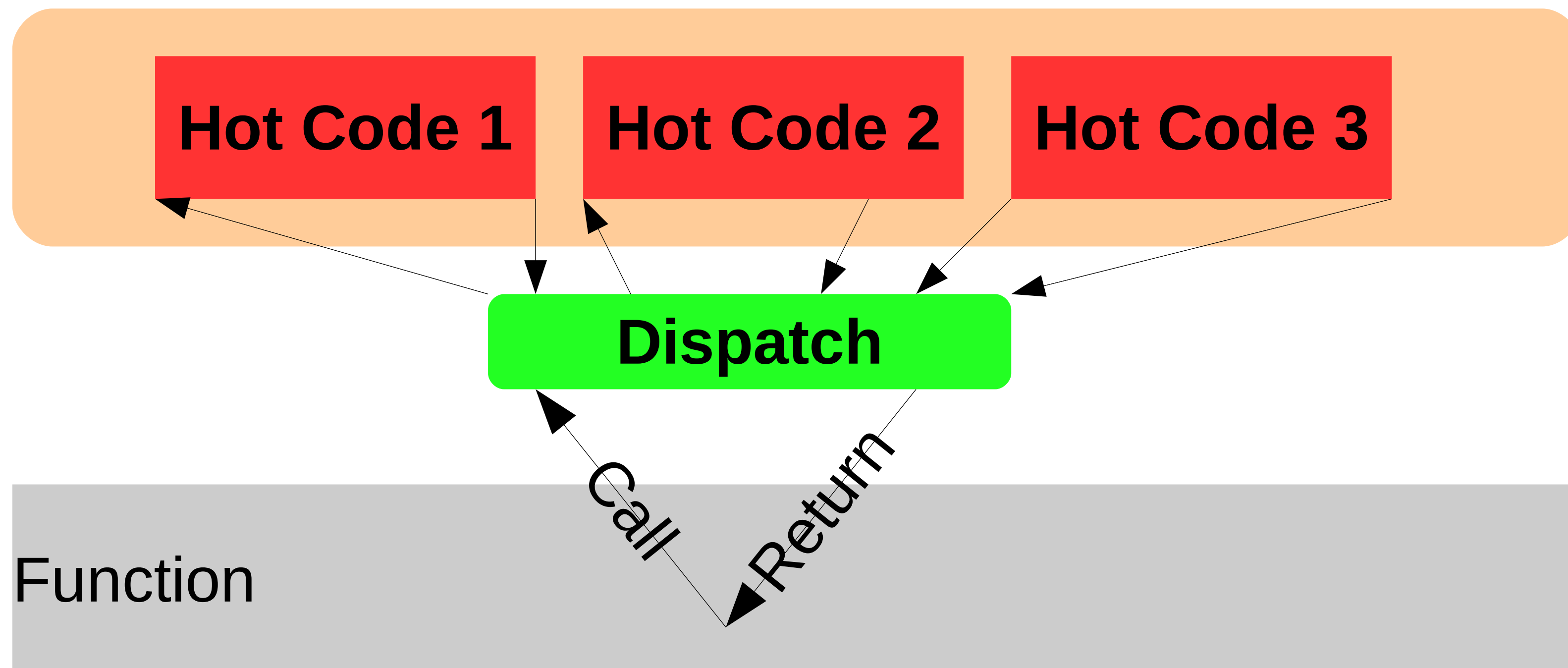
- Processors with different ISAs
- Different co-processors (GPGPUs, ASICs, FPGAs, …)
- Running wrong binary
  - Best case: programs run slow
  - Worst case: programs don't run



redhat

Avoiding Multiple Binaries

- One binary runs optimal overhead
- Reduce overhead of dispatching
  - OS support in RHEL6



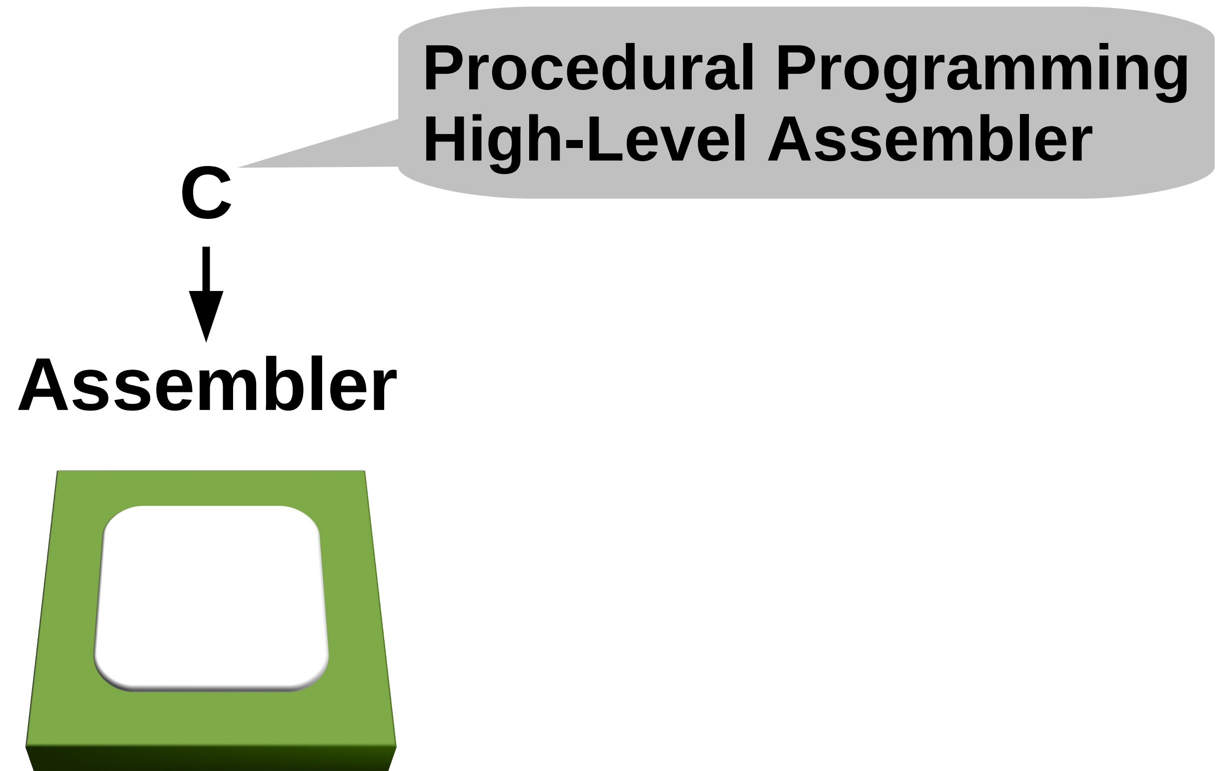
redhat.

Ways of Programming

Assembler

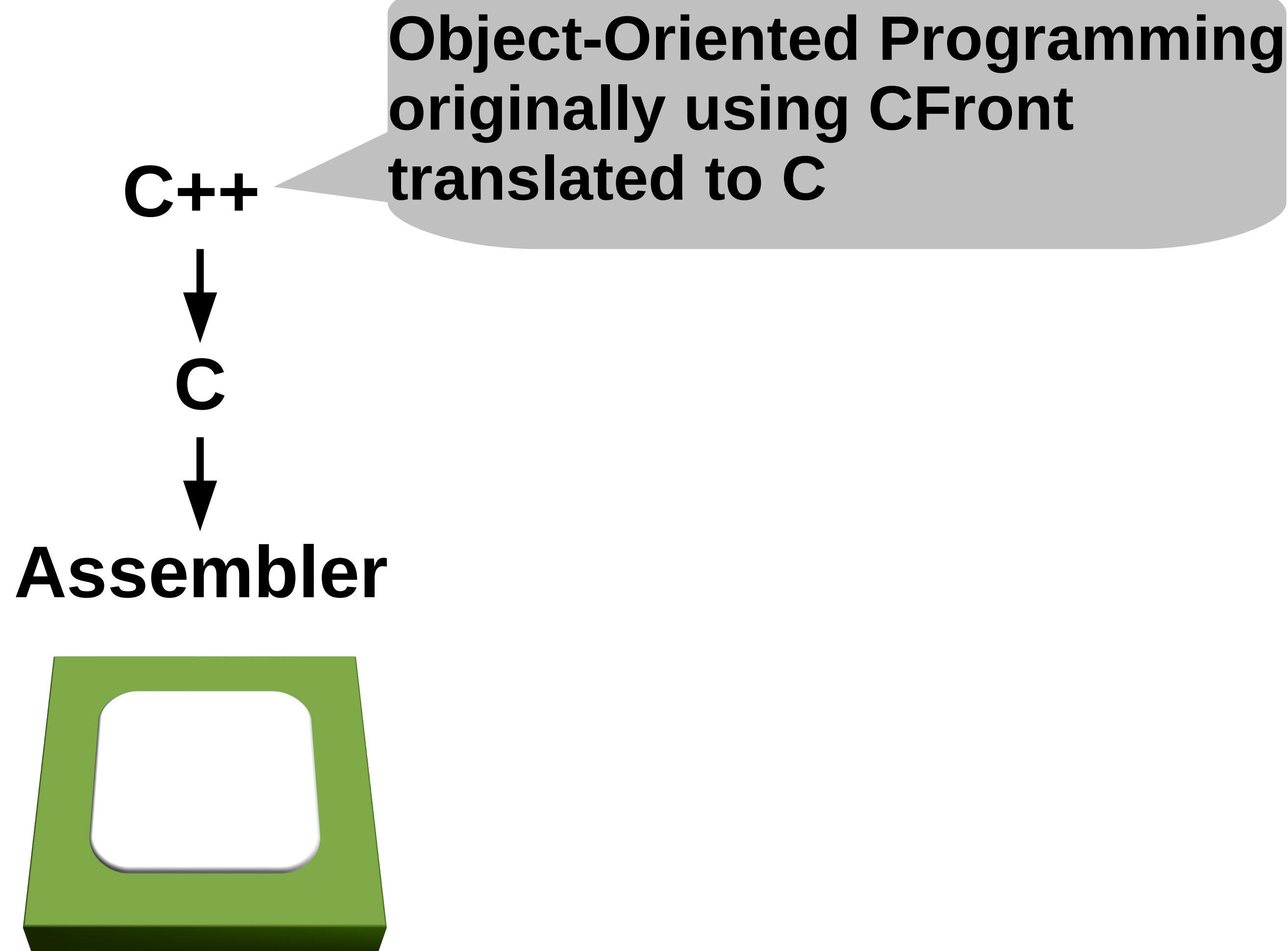
redhat

Ways of Programming



Ways of Programming

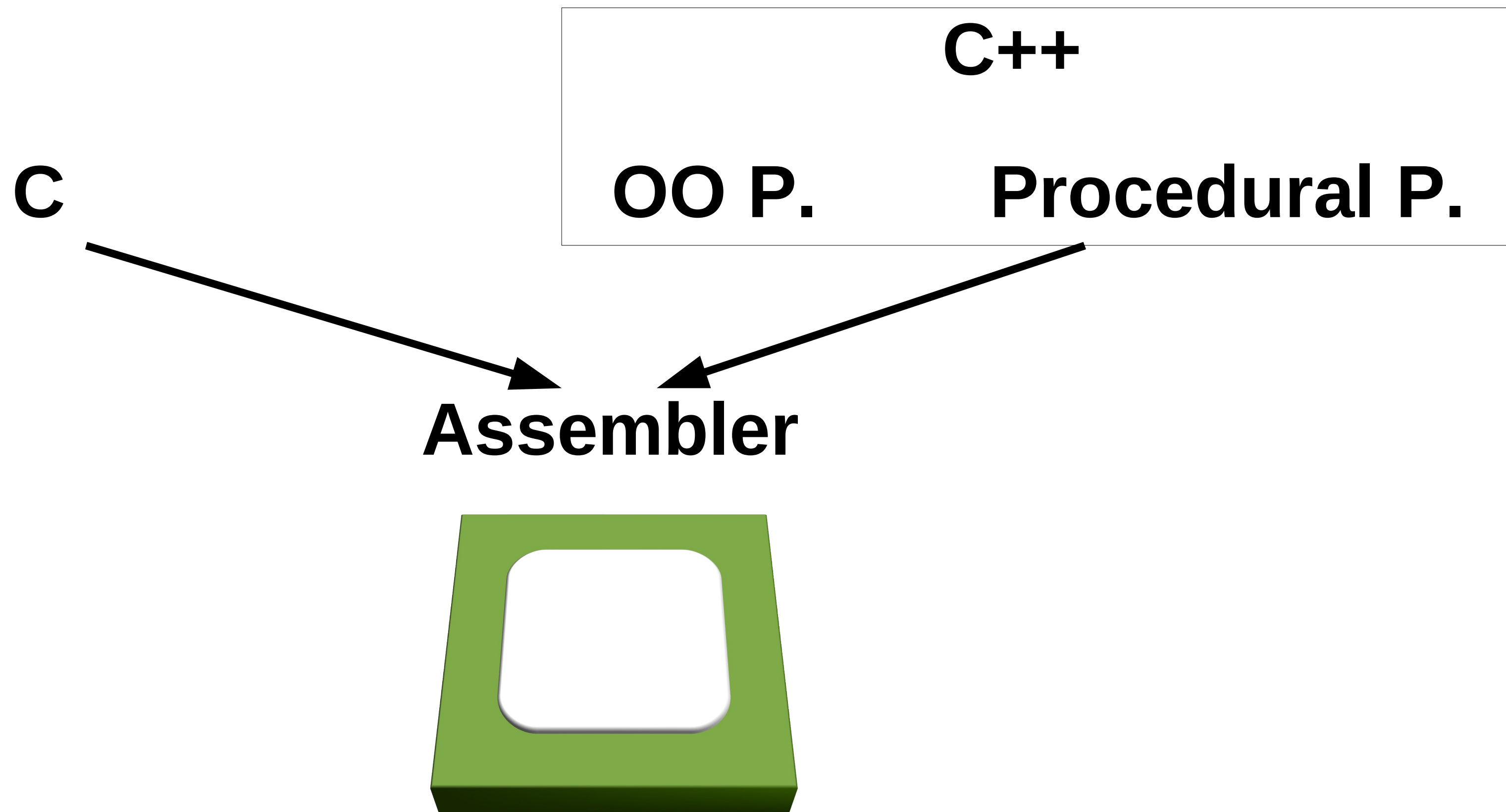
- More efficient C:



redhat

Ways of Programming

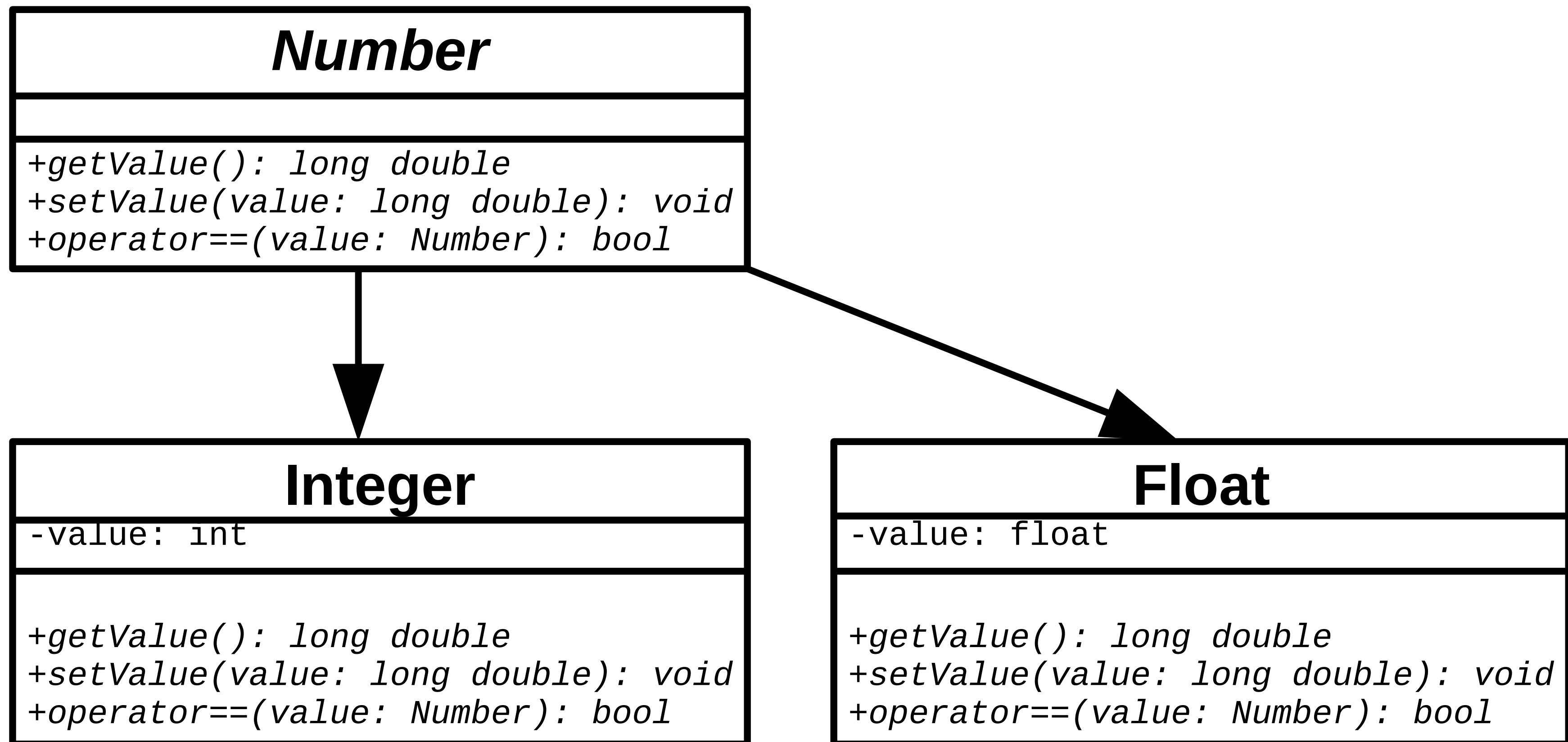
- C++ a language on its own



redhat

OOP Principles

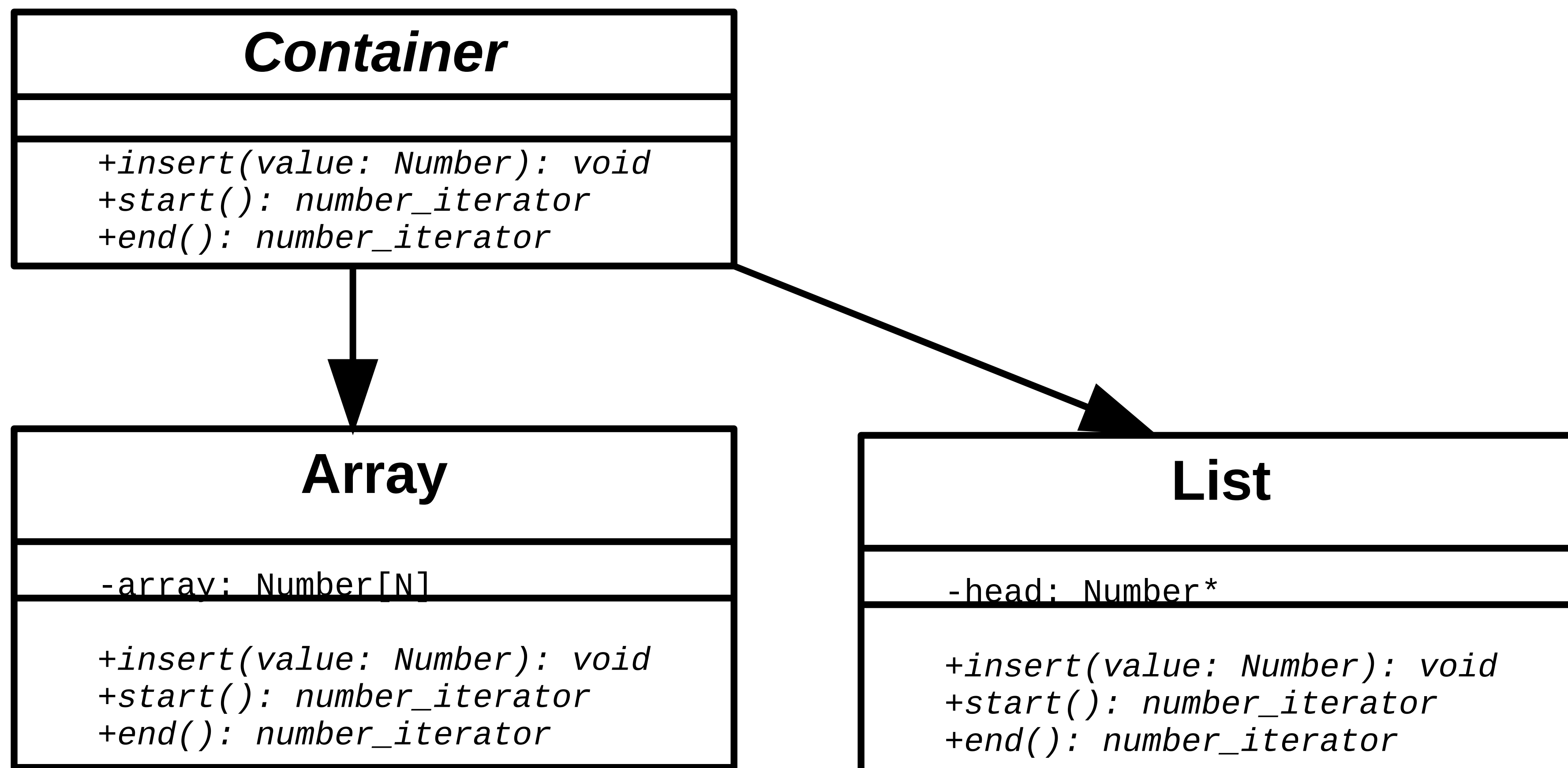
- Derive and specialize



redhat

OOP Principles

- Use base classes in other contexts



redhat

OOP Principles

- Advantages:
  - Flexibility
    - Maximum code sharing
    - Minimal ABI
    - Stable ABI for library/DSO interface
  - “Natural” abstraction
- Disadvantages:
  - Inefficient
  - Often requires “superset type” in interfaces

redhat

Use Example Classes

- Search implementation

```
bool find(Container &c, Number &n) {
  for (number_iterator i = c.start(); i != c.end(); ++i)
    if (*i == n)
      return true;
  return false;
}
```

- Cost:
  - 2 virtual function calls in prologue
  - 3 virtual function calls per loop iteration

redhat.

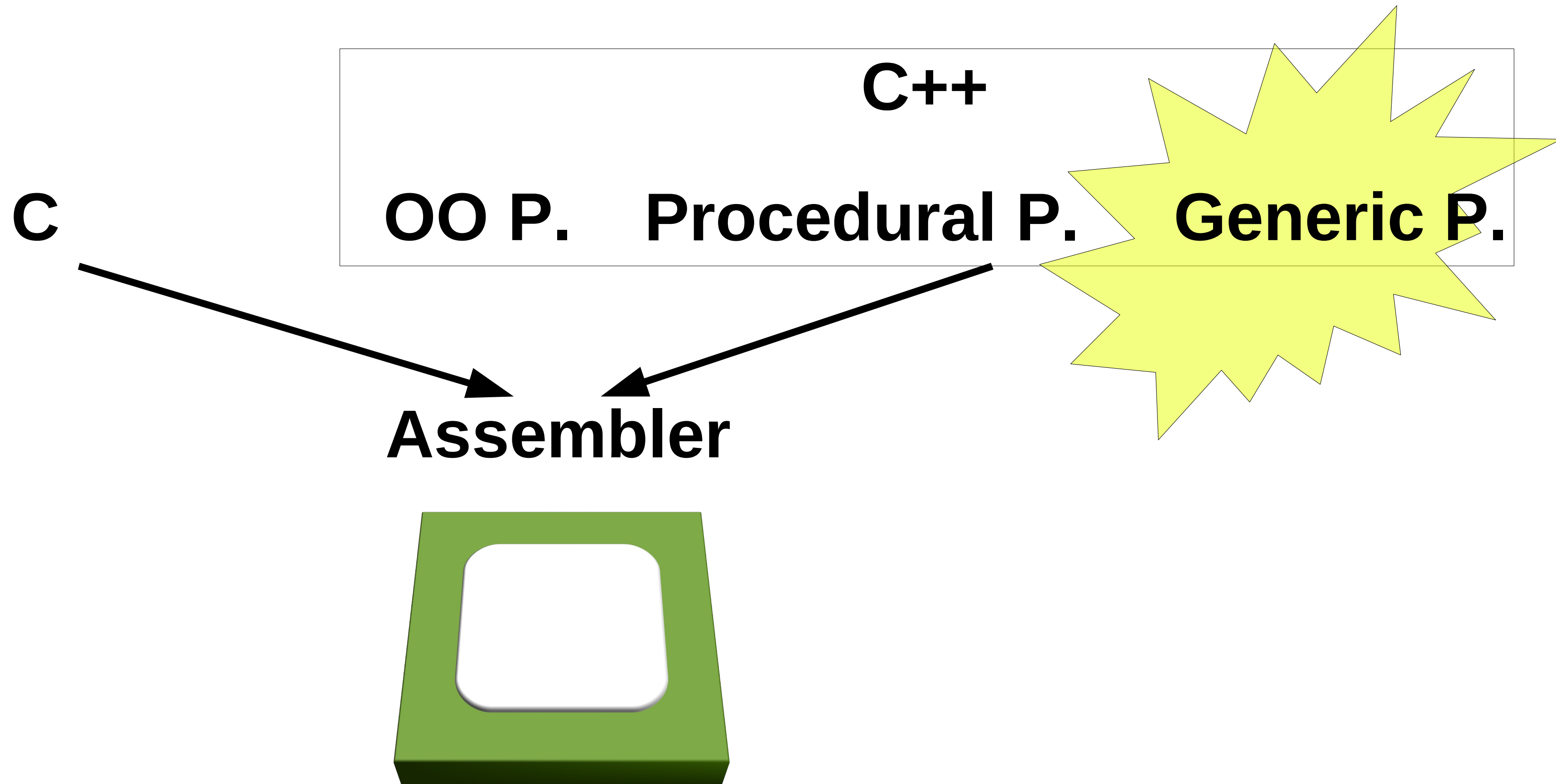
Virtual Functions

- Virtual function call
  - Cannot be inlined (in general)
  - Indirect function call → prevents prefetching of CPU
  - Often trivial operations dwarfed by cost of call
- Increased size of objects (virtual function table)

redhat

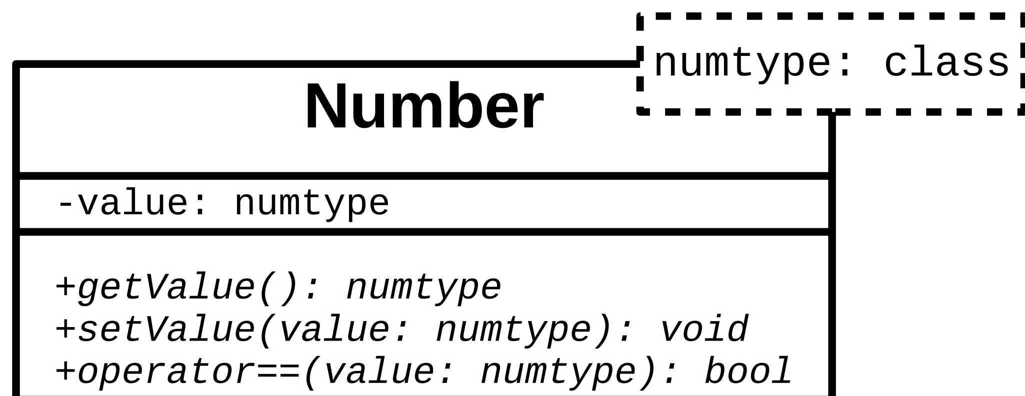
Extend Possibilities

- Templates and the STL



Template Examples

- Continue Example with templates:



- Avoid virtual calls for use of number objects
- Smaller objects

redhat.

Results of templates

- Advantages:
  - Code reuse easy
  - Efficient code through inlining
  - Optimization through specialization
    - Iteratively
- Disadvantages:
  - Code duplication per instantiation
  - Tricky to achieve type safety of template parameters

redhat

ISO C++ 1998 Problems

- Too often temporaries

- Type safety of template parameters
  - Implemented using traits
  - Impossible to decode error messages

$$s = \sum_i (\vec{a} \times f + \vec{b} - \vec{c})_i - \sum_i (\vec{d} \times g + \vec{e})_i$$

redhat

ISO C++ 1998 Problems

```
template<typename F, int N>
struct vec {
   F e[N];
   F &operator[] (size_t idx) { return e[idx]; }
   F operator[] (size_t idx) const { return e[idx]; }
};
template<typename F, int N>
vec<F,N>
operator+(const vec<F,N> &src1, const vec<F,N> &src2) {
  vec<F,N> res;
  for (int i=0;i<N;++i) res[i] = src1[i] + src2[i];
  return res;
}
```

redhat

ISO C++ 1998 Problems

```
template<typename F, int N>
vec<F,N>
operator-(const vec<F,N> &src1, const vec<F,N> &src2) {
   vec<F,N> res;
   for (int i=0; i<N; ++i) res[i] = src1[i] - src2[i];
   return res;
}
template<typename F, int N>
vec<F,N> operator*(const vec<F,N> &src, F f) {
  vec<F,N> res;
  for (int i=0; i<N; ++i) res[i] = src[i] * f;
  return res;
}
```

redhat.

ISO C++ 1998 Problems

```
template<typename F, int N>
F sumvec(const vec<T,N> &src) {
   F res = 0.0;
   for (int i=0; i<N; ++i) res += src[i];
   return res;
}


{ ...
   s = sumvec(a * f + b - c) – sumvec(d * g – e);
   ...
}
```

redhat

C-Style Memory Management

```
template<typename F, int N>
struct vec {
   F e[N];
   F &operator[] (size_t idx) { return e[idx]; }
   F operator[] (size_t idx) const { return e[idx]; }
};
template<typename F, int N>
void addvec(vec<F,N> &dst, const vec<F,N> &src1,
            const vec<F,N> &src2) {
  for (int i=0;i<N;++i) dst[i] = src1[i] + src2[i];
}
```

redhat

C-style Memory Management

```
template<typename F, int N>
void subvec(vec<F,N> &dst, const vec<F,N> &src1,
            const vec<F,N> &src2) {
   for (int i=0; i<N; ++i) dst[i] = src1[i] - src2[i];
}
template<typename F, int N>
void scalevec(vec<F,N> &dst, const vec<F,N> &src, F f) {
  for (int i=0; i<N; ++i) dst[i] = src[i] * f;
}
```

redhat

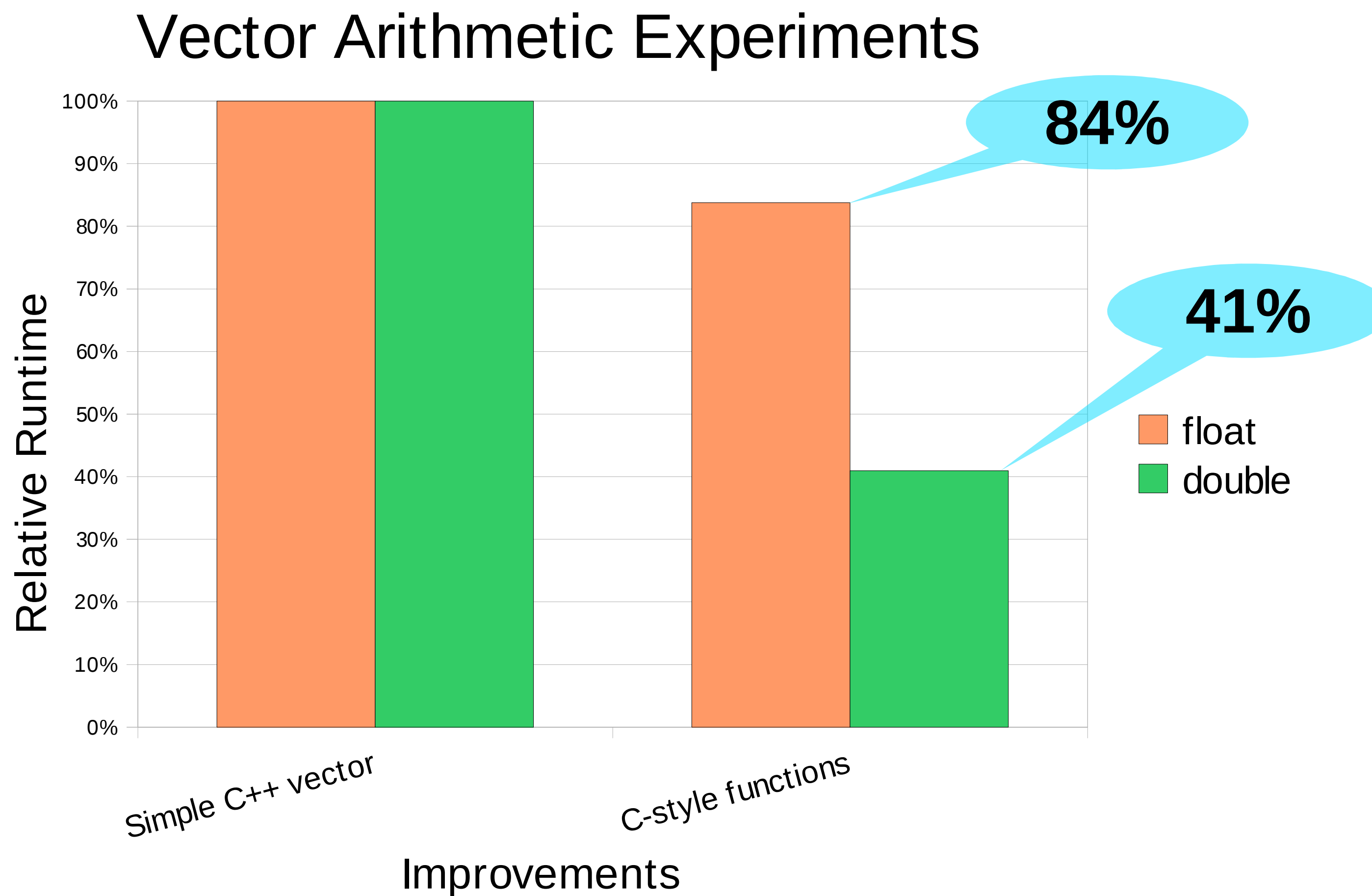
C-style Memory Management

```
template<typename F, int N>
F sumvec(const vec<T,N> &src) {
   F res = 0.0;
   for (int i = 0; i < N; ++i) res += src[i];
   return res;
}
{  vec<float,100000> t;
   scalevec(t,a,f); addvec(t,t,b); subvec(t,t,c);
   s = sumvec(t);
   scalevec(t,d,g); addvec(t,t,e);
   s -= sumvec(t);
   ...
}
```

Truly Ugly!

redhat

Ugly&Unmaintainable, But Faster



redhat

Move Semantic/rvalue References

```
template<typname F, int N>
vec<F,N> &&operator+(vec<F,N> &&src1, const vec<F,N> &src2) {
   for (int i=0; i<N; ++i) src1[i] += src2[i];
   return src1;
}

template<typename F, int N>
vec<F,N> &&operator-(vec<F,N> &&src1, const vec<F,N> &src2) {
   for (int i=0; i<N; ++i) src1[i] -= src2[i];
   return src1;
}
```

No Need
To Document

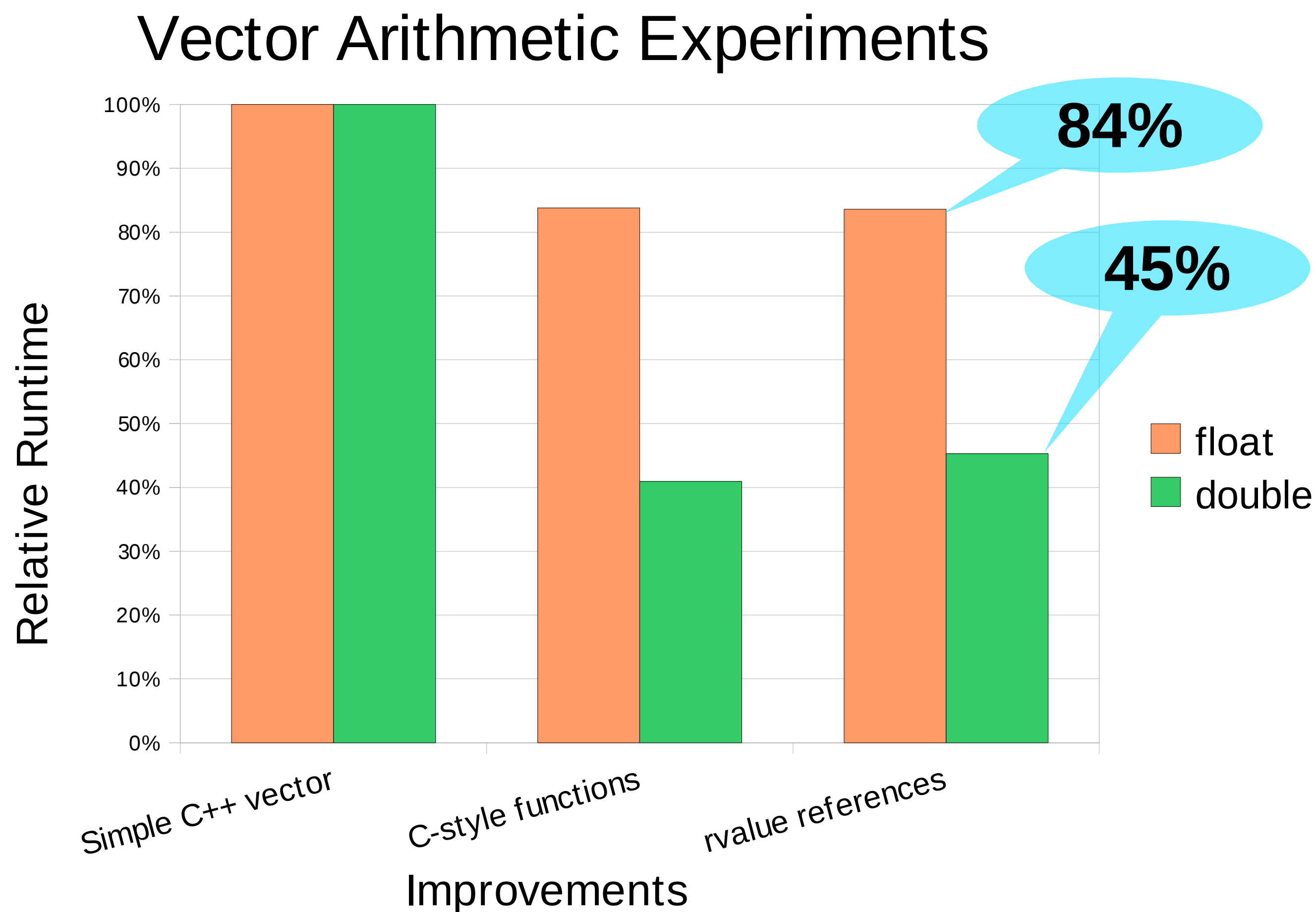
Move Semantic/rvalue References

```
{ ...
   s = sumvec(a * f + b - c) – sumvec(d * g – e);
   ...
}
```

Identical to simple
C++ vector type

redhat

Usability & Performance



Optimization Phase

- All code concentrated in headers and library functions
- Code for library users trivial
- Full control over memory handling
- Apply additional optimizations. For example:
  - Combine operation
  - Vectorization
  - Parallelization

redhat

Combination

Additional class:

```
template<typename F, int N>
struct vecscale {
   const vec<F,N> &v;
   F f;
   vecscale(const vec<F,N> &va, F fa) : v(va), f(fa) {}
};
```

Combination

Additional template functions:

```
template<typename F, int N>
scalevec<F,N> operator*(const vec<F,N> &src, F f) {
   return scalevec<F,N>(src, f);
}
template<typename F, int N>
vec<F,N> operator+(scalevec<F,N> &src1,
                   const vec,F,N> &src2) {
   vec<F,N> dst;
   for (int i=0; i<N; ++i)
      dst[i] = src1.v[i] * src1.f + src2[i];
   return dst;
}
```

redhat

Combination

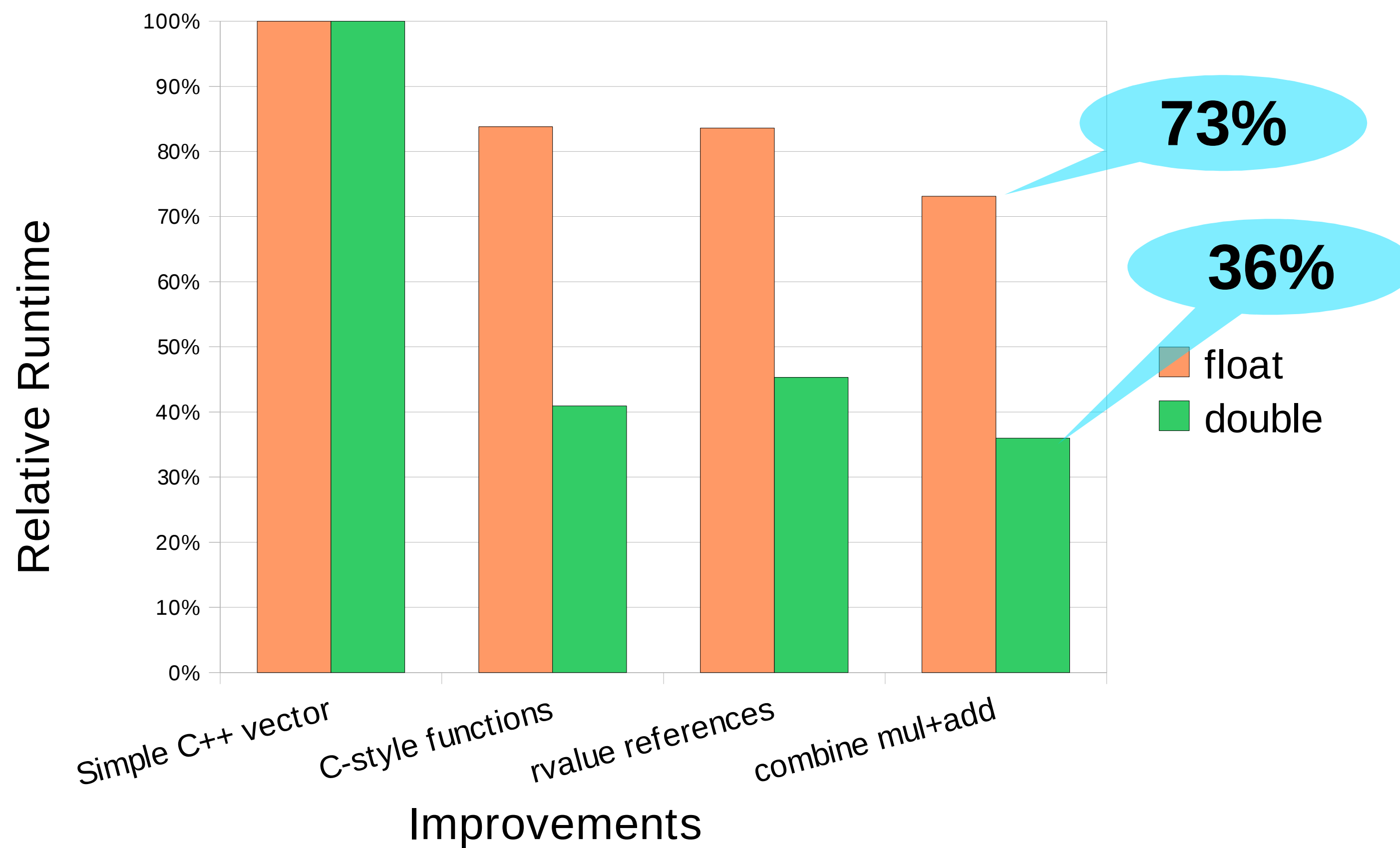
Additional template functions:

```
template<typename F, int N>
scalevec<F,N> operator*(const vec<F,N> &src, F f) {
   return scalevec<F,N>(src, f);
}
template<int N>
vec<float,N> operator+(scalevec<float,N> &src1,
                       const vec<float,N> &src2) {
   vec<float,N> dst;
   for (int i=0; i<N; ++i)
      dst[i] = fmaf(src1.v[i], src1.f, src2[i]);
   return dst;
}
```

Combination

```
{ ...
   s = sumvec(a * f + b - c) – sumvec(d * g – e);
   ...
}
```

And Better...



redhat.

Why Stop There?

Yet another class:

```
template<typename F, int N>
struct vecscaleadd {
   const vec<F,N> &v1;
   F f;
   const vec<F,N> &v2;
   vecscaleadd(const vec<F,N> &va1, F fa, const vec<F,N> %va2)
   : v1(va1), f(fa), v2(va2) {}
   operator vec<T,N>(void) const;
};
```

Why Stop There?

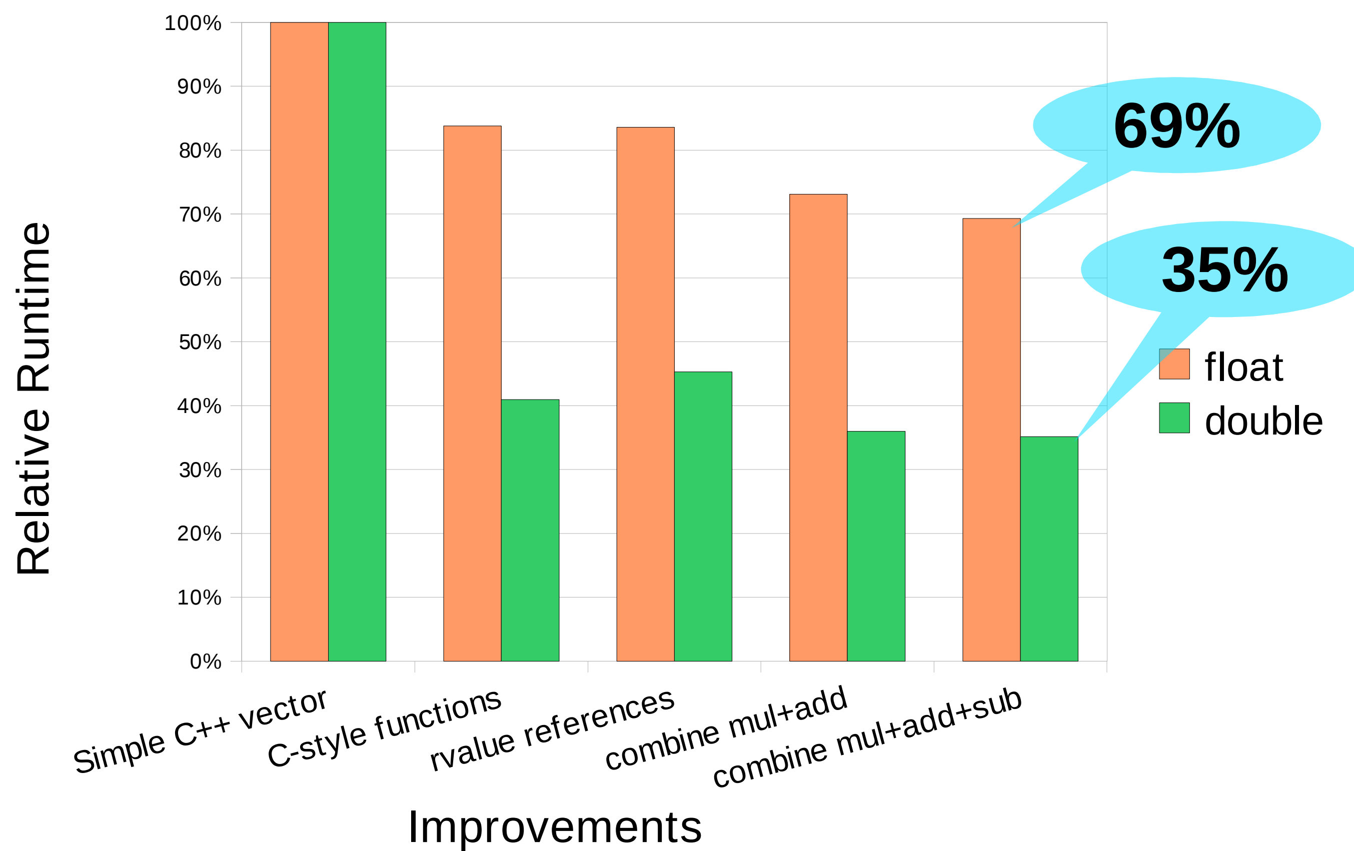
```
template<typename F, int N>
vecscaleadd<F,N> operator+(const vecscale<F,N> &src1,
                           const vec<F,N> &src2) {
   return vecscaleadd<F,N>(src1.v, src1.f, src2);
}
template<typename F, int N>
vec<F,N> operator-(vecscaleadd<F,N> &src1,
                   const vec<F,N> &src2) {
   vec<F,N> dst;
   for (int i=0; i<N; ++i)
      dst[i] = src1.v1[i] * src1.f + src1.v2 - src2[i];
   return dst;
}
```

Why Stop There?

```
{ ...
   s = sumvec(a * f + b - c) – sumvec(d * g – e);
   ...
}
```

redhat

... and better ...



Vectorization

```
template<typename F, int N>
struct vec {
   union {
      F e[N];
      __m128 m[N / 4];
      __m128d d[N / 2];
   };
   F &operator[](size_t n) { return e[n]; }
   F operator[](size_t n) const {return e[n]; }
};
```

Vectorization

```
template<int N>
vec<float,N> operator-(const vecscaleadd<float,N> &src1,
                       const vec<float,N> &src2) {
   __m128 fv = _mm_set1_ps(src1.f);
   vec<float,N> dst;
   for (int i=0; i<N/4; ++i)
      dst.m[i] = _mm_sub_ps(_mm_add_ps(_mm_mul_ps(
                                          src1.v1.m[i],
                                          fv),
                                      src1.v2.m[i]),
                            src2.m[i]);
   return dst;
}
```

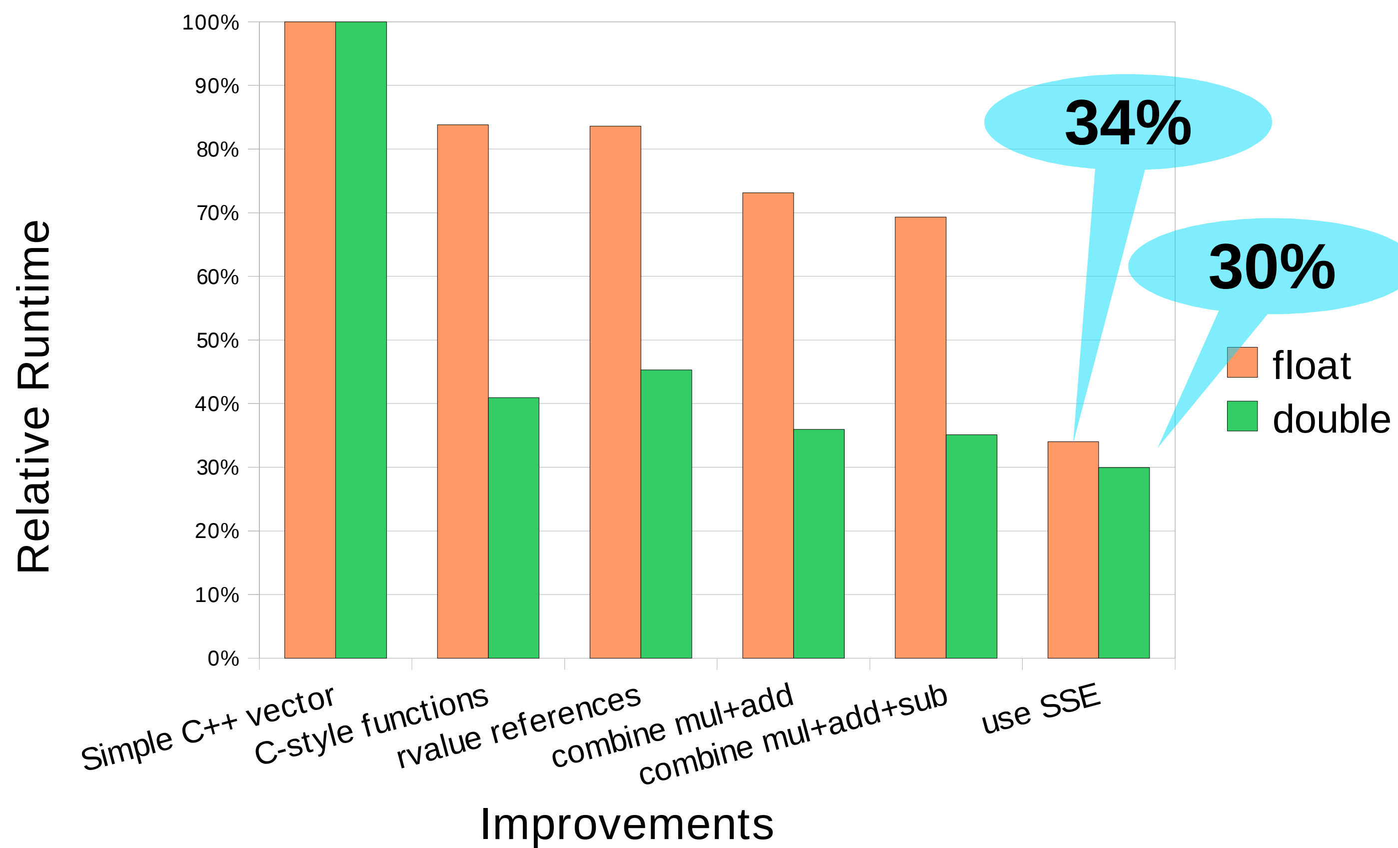
redhat.

Vectorization

```
{ ...
    s = sumvec(a * f + b - c) – sumvec(d * g – e);
    ...
}
```

You guessed it: no change

... and better ...



redhat

Parallelization

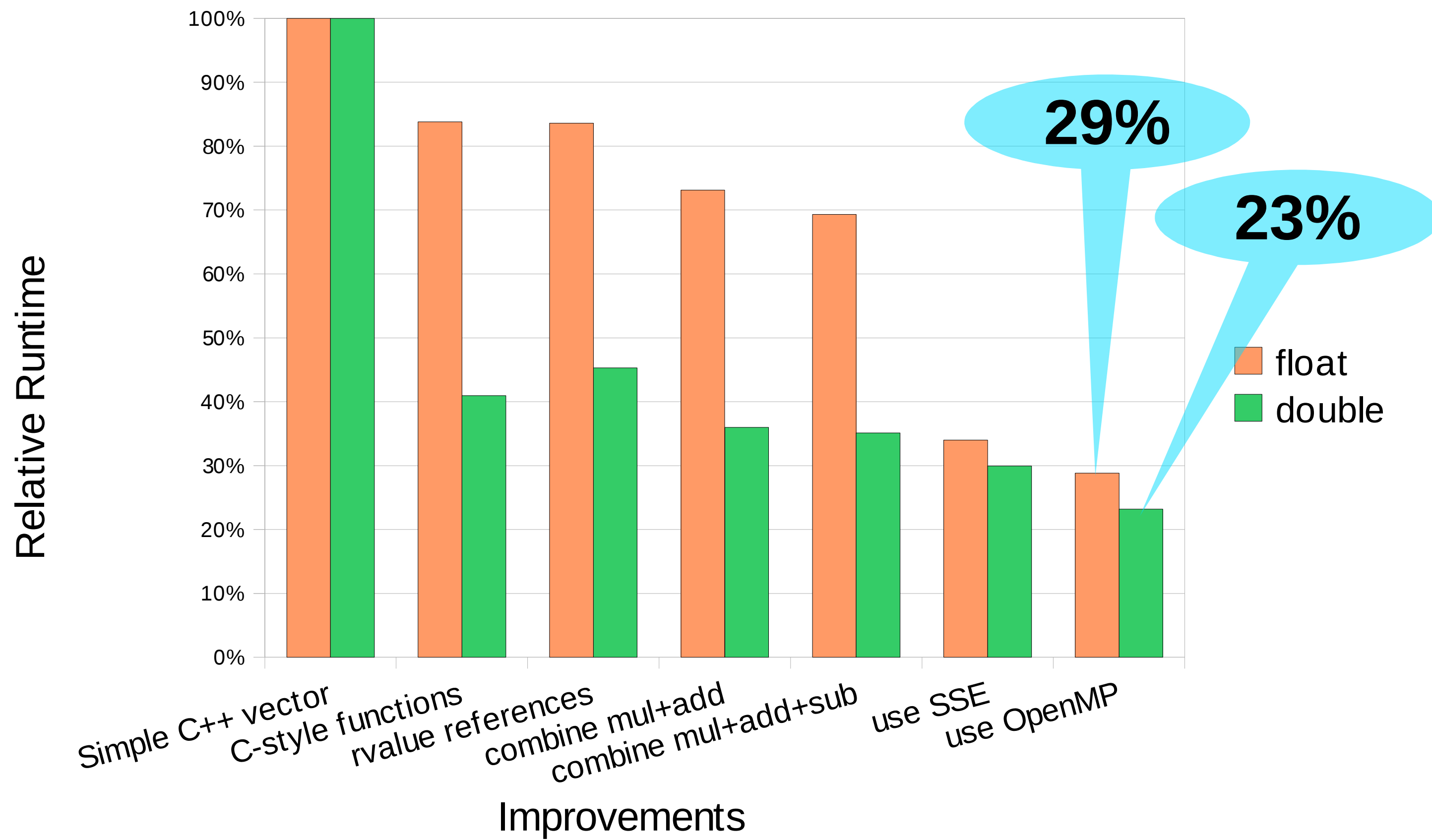
```
template<int N>
vec<float,N> operator-(const vecscaleadd<float,N> &src1,
                       const vec<float,N> &src2) {
   __m128 fv = _mm_set1_ps(src1.f);
   vec<float,N> dst;
#pragma omp parallel for schedule(guided)
   for (int i=0; i<N; ++i)
      dst.m[i] = _mm_sub_ps(_mm_add_ps(_mm_mul_ps(
                                          src1.v1.m[i],
                                          fv),
                                      src1.v2.m[i]),
                           src2.m[i]);
   return dst;
}
```

Parallelization

```
{ ...
    s = sumvec(a * f + b - c) – sumvec(d * g – e);
    ...
}
```

redhat

... and better!



redhat

Solving Template Parameter Checking

- Type system for template parameters

```
concept NumberC<class T> {
  T operator+(T,T);
  T operator-(T,T);
  bool operator==(T,T);
}
template<class T>
require NumberC<T>
T someOp(T a, T b, T c) {
   ...
};
```

Compare with Haskell

```
class Eq a where
  (==) :: a -> a -> Bool
class Num a where
  (+) :: a -> a -> a
  (-) :: a -> a -> a
someOp :: (Eq a, Num a) => a -> a -> a -> a
SomeOp = ...
```

- Much better, easier control over specialization

Summary

- For performance, programs must adapt to hardware
- Majority of programmers cannot do this
- Organize code to allow replacing components
  - Functional programming style
  - Careful design of interfaces
  - More libraries
- Application programmers write against simplified interfaces
- Optimization team replace hot spot components
- Performance through
  - Specialized code (still generic)
  - Better exploitation of hardware

Questions?

drepper@redhat.com | people.redhat.com/redhat