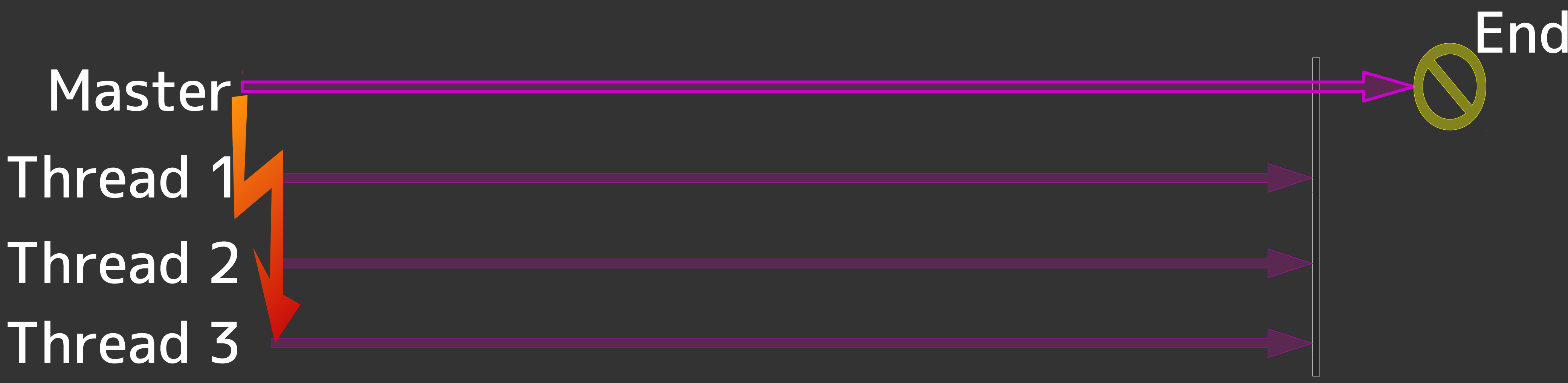
The Many Ways to Parallelism in gcc

Ulrich Drepper

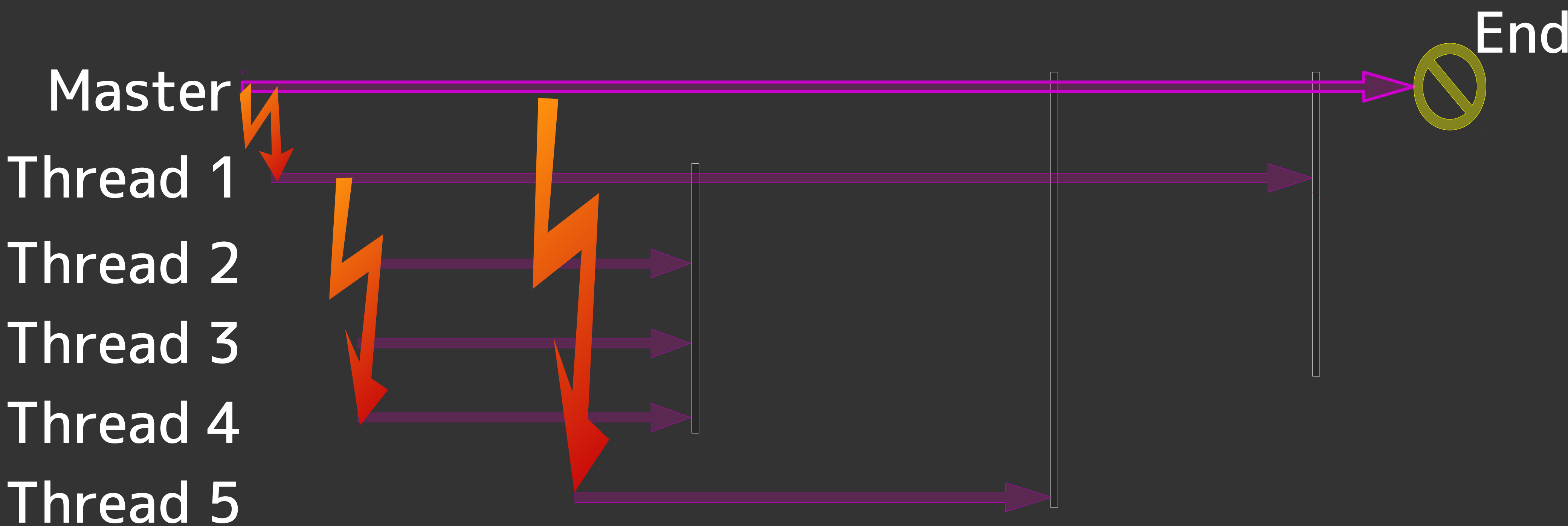
para//el 2016

Simple but rare

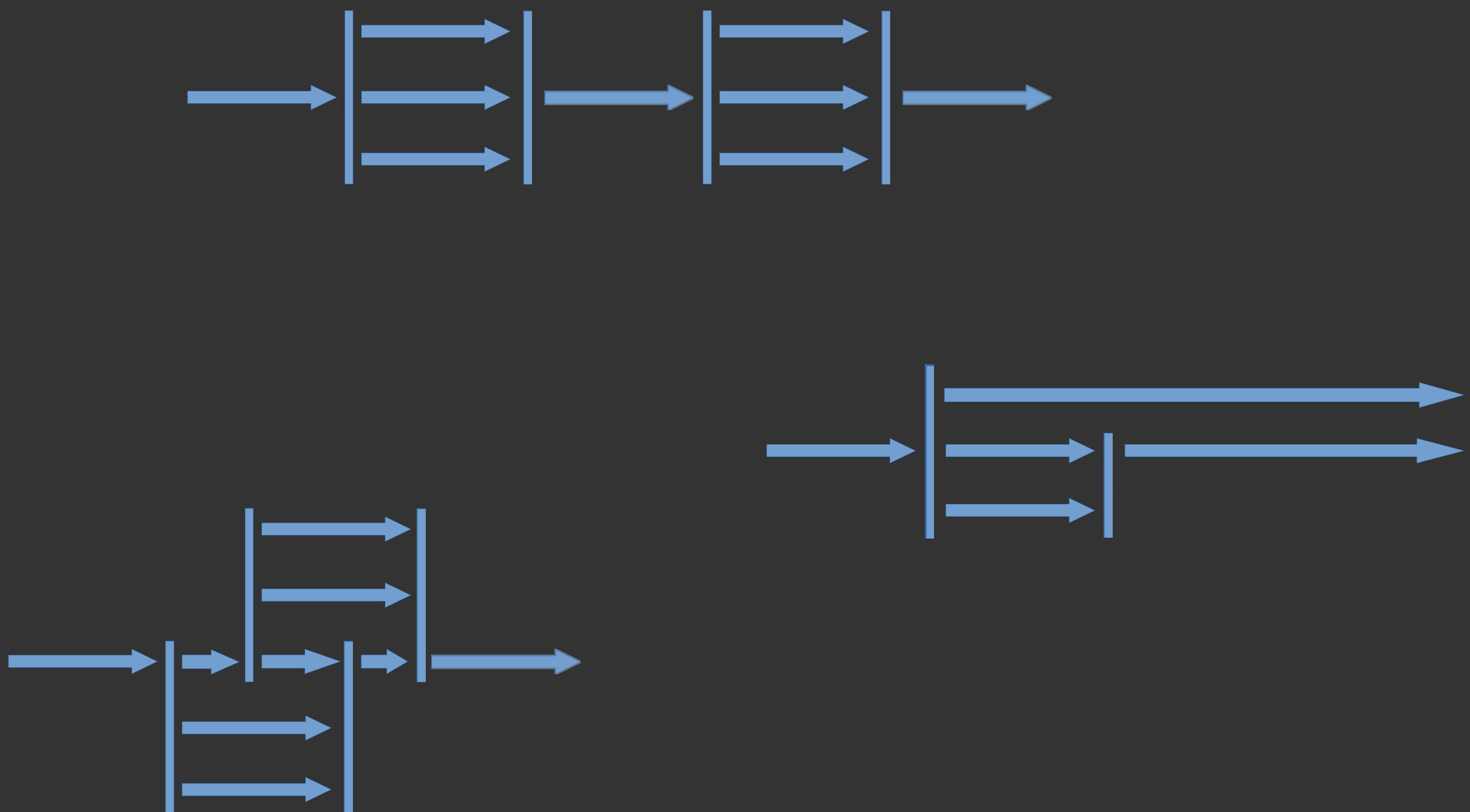


Embarrassingly Parallel code

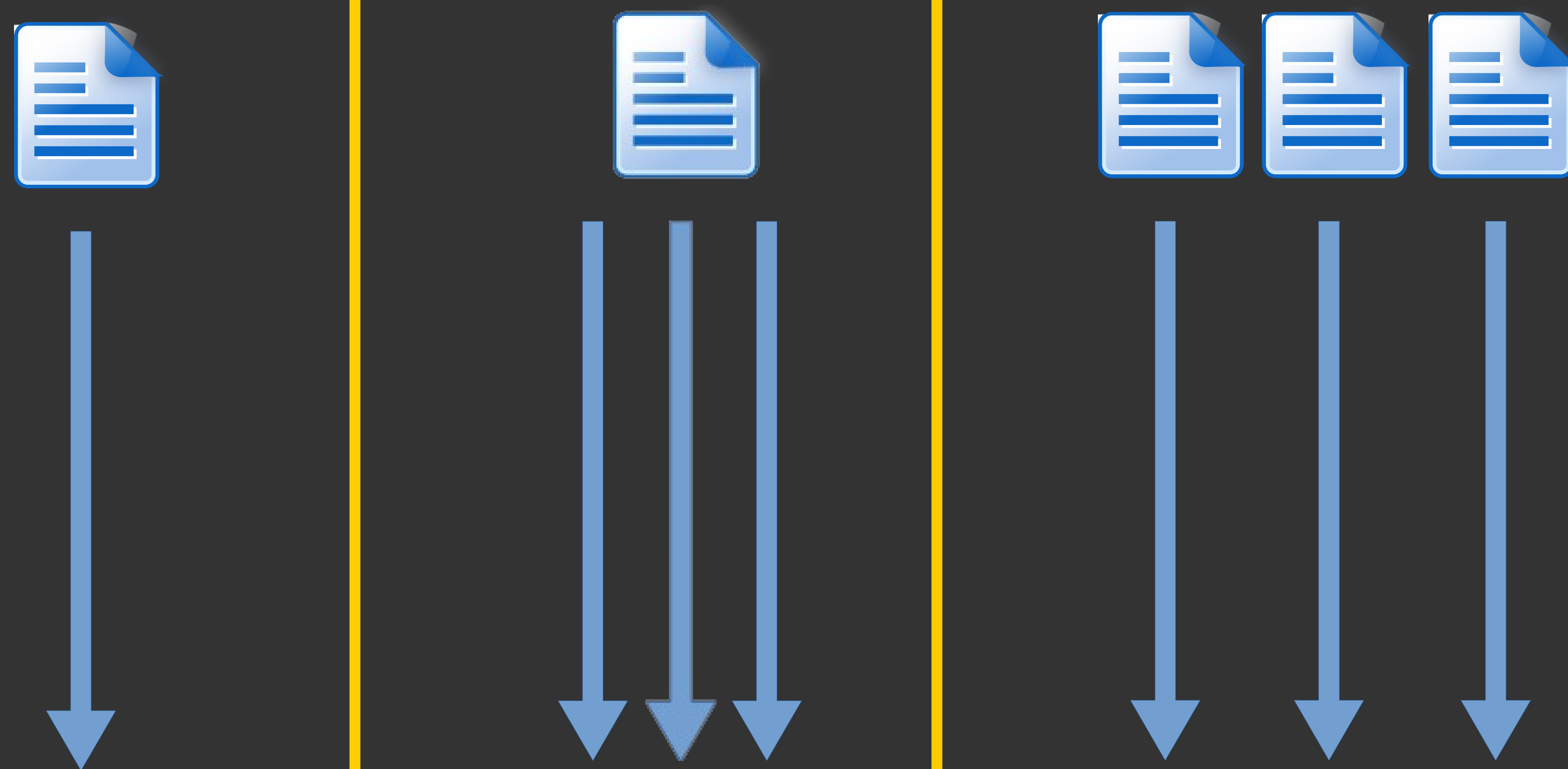
Hardly ever the case


End
Master
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5

Creation of Parallelism



Control Flow Variants

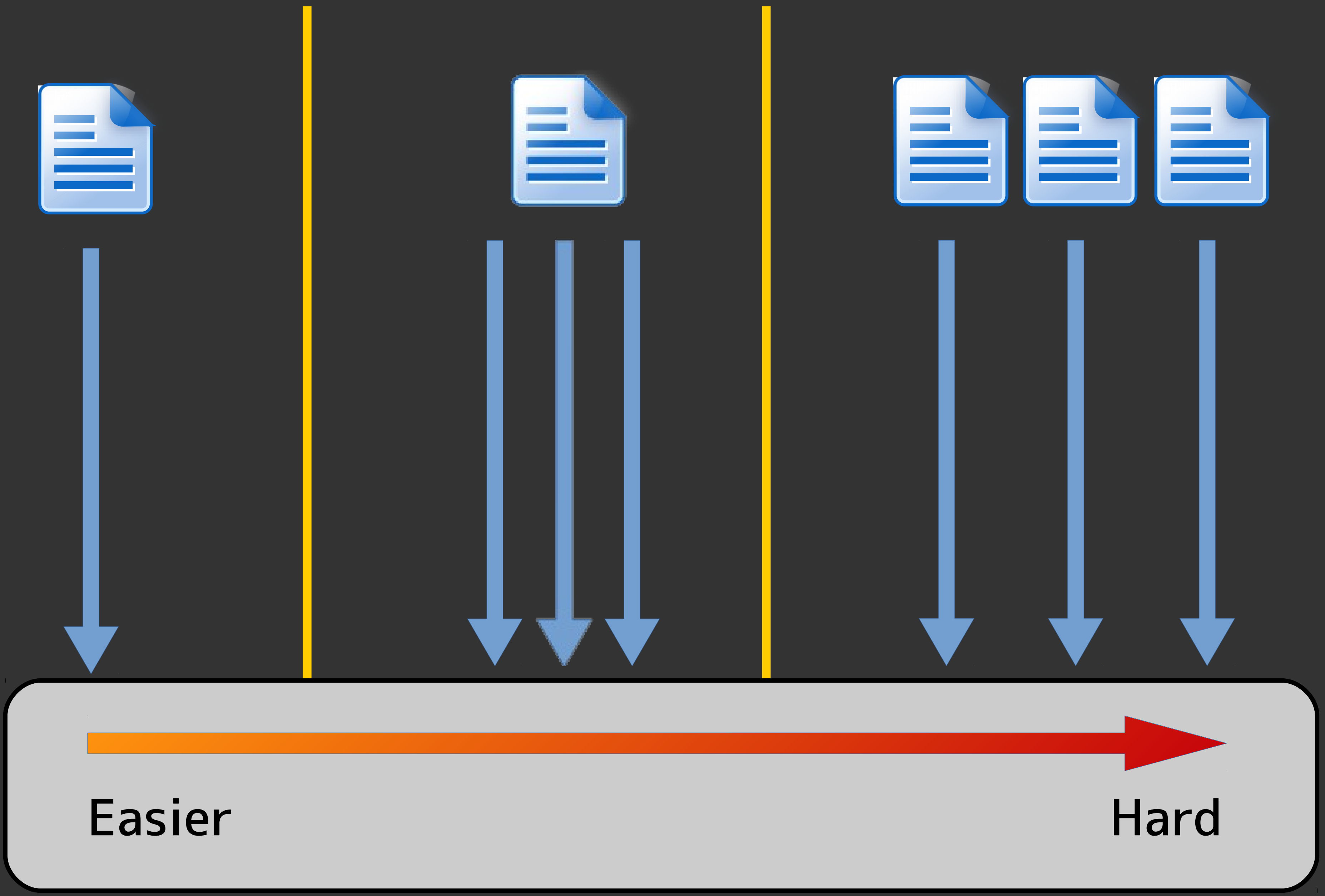


Serial

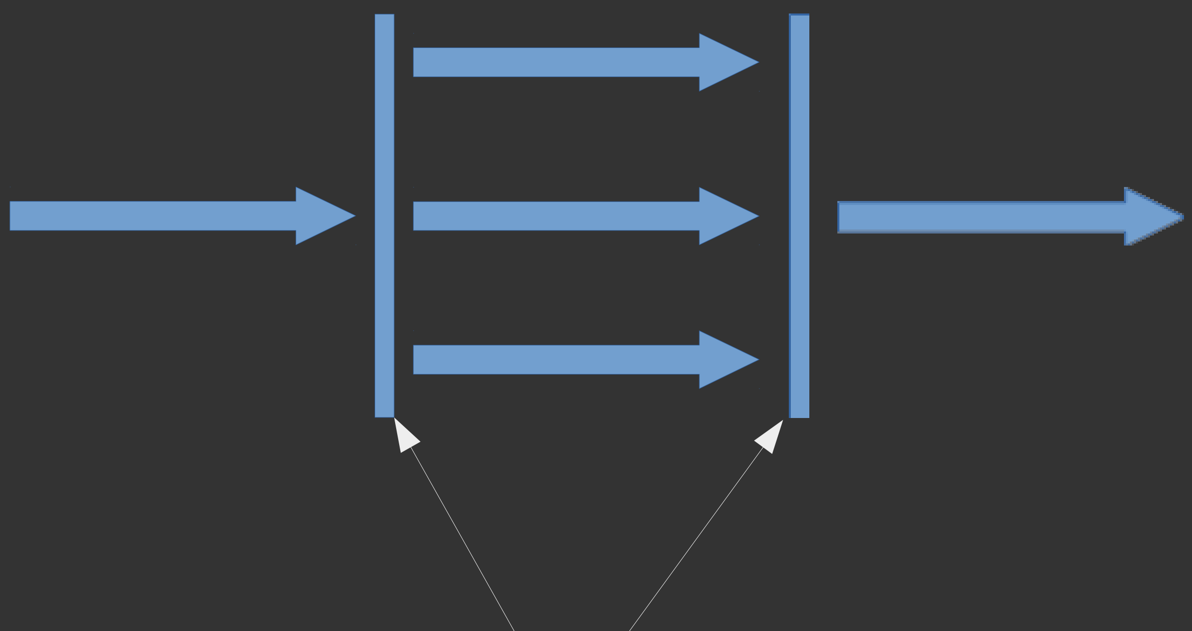
Same Code Executed Multiple Times

Different Code Sequences Run In Parallel

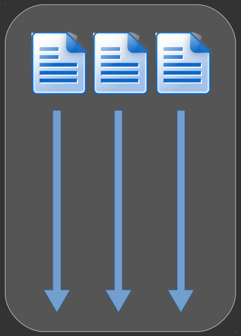
Control Flow Variants



Creation of Parallelism

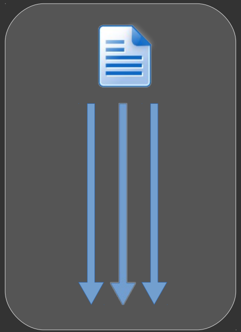


Different Code Paths



```
template<typename T, size_t N>
std::tuple<T,T,T>
minmaxavg(const std::array<T,N>& arr) {
  auto min = std::numeric_limits<T>::max();
  auto max = std::numeric_limits<T>::min();
  auto sum = T(0);
#pragma omp parallel sections
  {
#pragma omp section
    std::for_each(arr.begin(), arr.end(), [&min](auto d){ if (d < min) min = d; });   1
#pragma omp section
    std::for_each(arr.begin(), arr.end(), [&max](auto d){ if (d > max) max = d; });   2
#pragma omp section
    sum = std::accumulate(arr.begin(), arr.end(), T(0));   3
  }
  return std::make_tuple(min, max, sum / N);
}
```

With -D_GLIBCXX_PARALLEL



```
template<typename T, size_t N>
std::tuple<T,T,T>
minmaxavg(const std::array<T,N>& arr) {
  auto min = std::numeric_limits<T>::max();
  auto max = std::numeric_limits<T>::min();
  auto sum = T(0);

    std::for_each(arr.begin(), arr.end(), [&min](auto d){ if (d < min) min = d; });

    std::for_each(arr.begin(), arr.end(), [&max](auto d){ if (d > max) max = d; });

    sum = std::accumulate(arr.begin(), arr.end(), T(0));

  return std::make_tuple(min, max, sum / N);
}
```

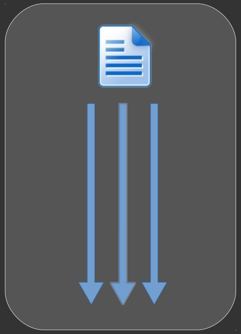
NOT THREAD SAFE!

1 2 3 ...

1 2 3 ...

1 2 3 ...

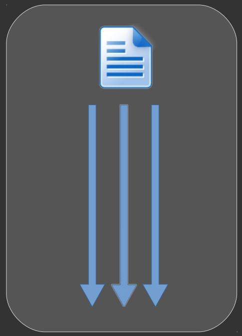
With -D_GLIBCXX_PARALLEL



```
template<typename T, size_t N>
std::tuple<T,T,T>
minmaxavg(const std::array<T,N>& arr) {
  auto min = std::numeric_limits<T>::max();
  auto max = std::numeric_limits<T>::min();
  auto sum = T(0);
#pragma omp declare reduction(rmin:double:omp_out=omp_out<omp_in?omp_out:omp_in) \
             initializer(omp_priv=std::numeric_limits<decltype(omp_priv)>::max())
#pragma omp parallel for reduction(rmin:min)
   for (size_t i = 0; i < N; ++i) if (arr[i] < min) min = arr[i];
#pragma omp declare reduction(rmax:double:omp_out=omp_out>omp_in?omp_out:omp_in) \
             initializer(omp_priv=std::numeric_limits<decltype(omp_priv)>::min())
#pragma omp parallel for reduction(rmax:max)
   for (size_t i = 0; i < N; ++i) if (arr[i] > max) max = arr[i];

   sum = std::accumulate(arr.begin(), arr.end(), T(0));

  return std::make_tuple(min, max, sum / N);
}
```

```fortran
subroutine minmaxavg(arr, resmin, resmax, resavg)
  double precision, intent(in), dimension(100000) :: arr
  double precision, intent(out) :: resmin, resmax, resavg
  integer :: i
  resmin = 0.
!$omp parallel do reduction(min:resmin)
  do i=1,100000
    resmin = min(arr(i), resmin)
  end do
!$omp end parallel do
  resmax = 0.
!$omp parallel do reduction(max:resmax)
  do i=1,100000
    resmax = max(arr(i), resmax)
  end do
!$omp end parallel do
  resavg = 0.
!$omp parallel do reduction(+:resavg)
  do i=1,100000
    resavg = resavg + arr(i)
  end do
!$omp end parallel do
  resavg = resavg / 100000.
end subroutine minmaxavg
```

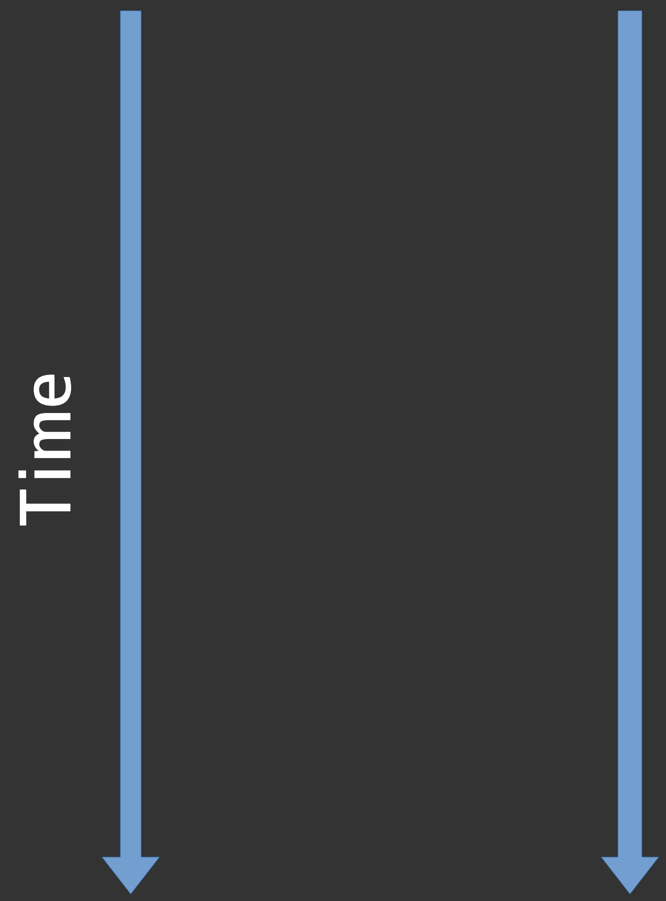
More builtin operators



Need for Synchronization

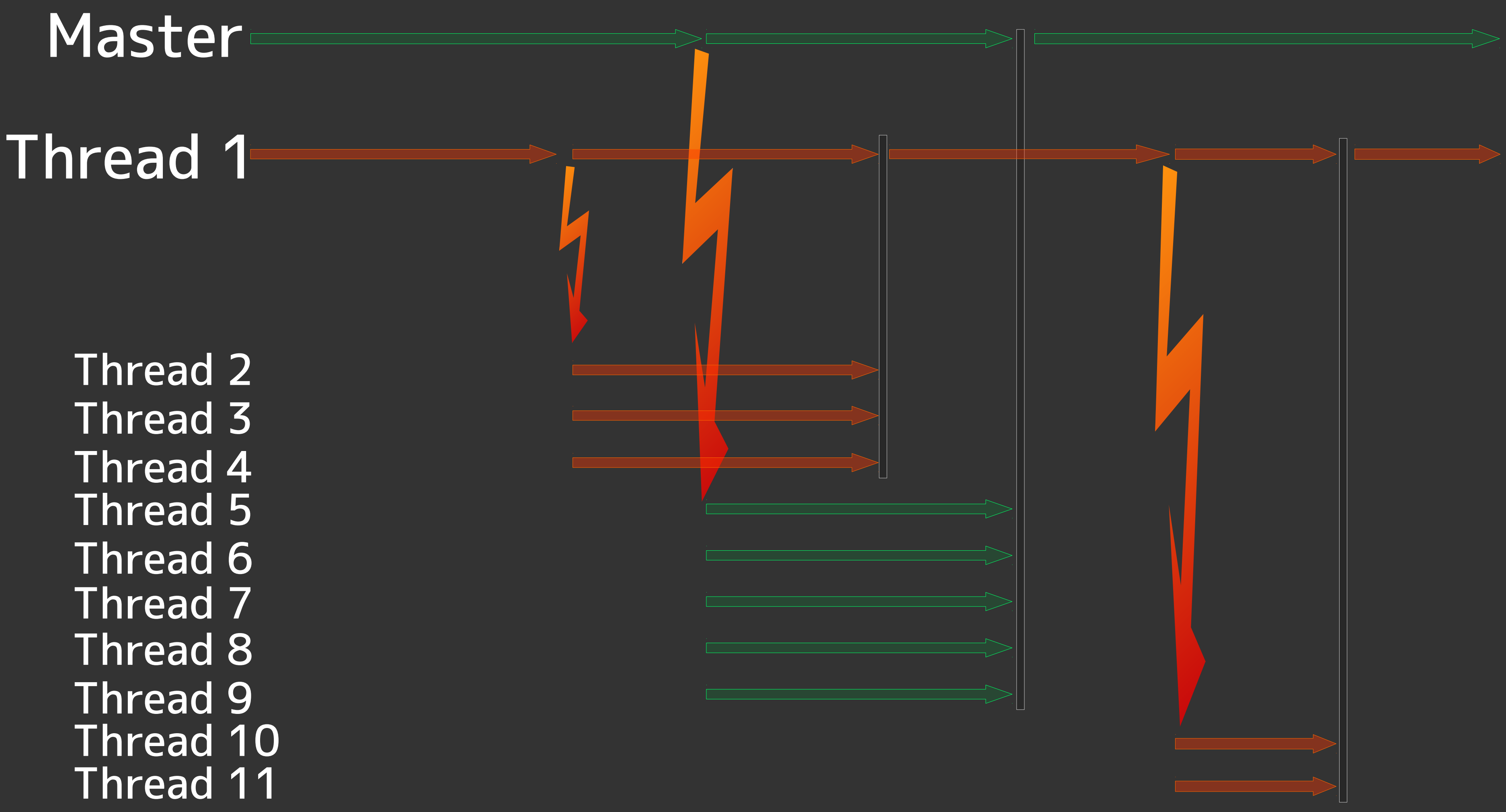
```
prev = var;
adj = prev * rate;
var = var + adj;
```

```
newsave = var;
var = save;
save = newsave;
```

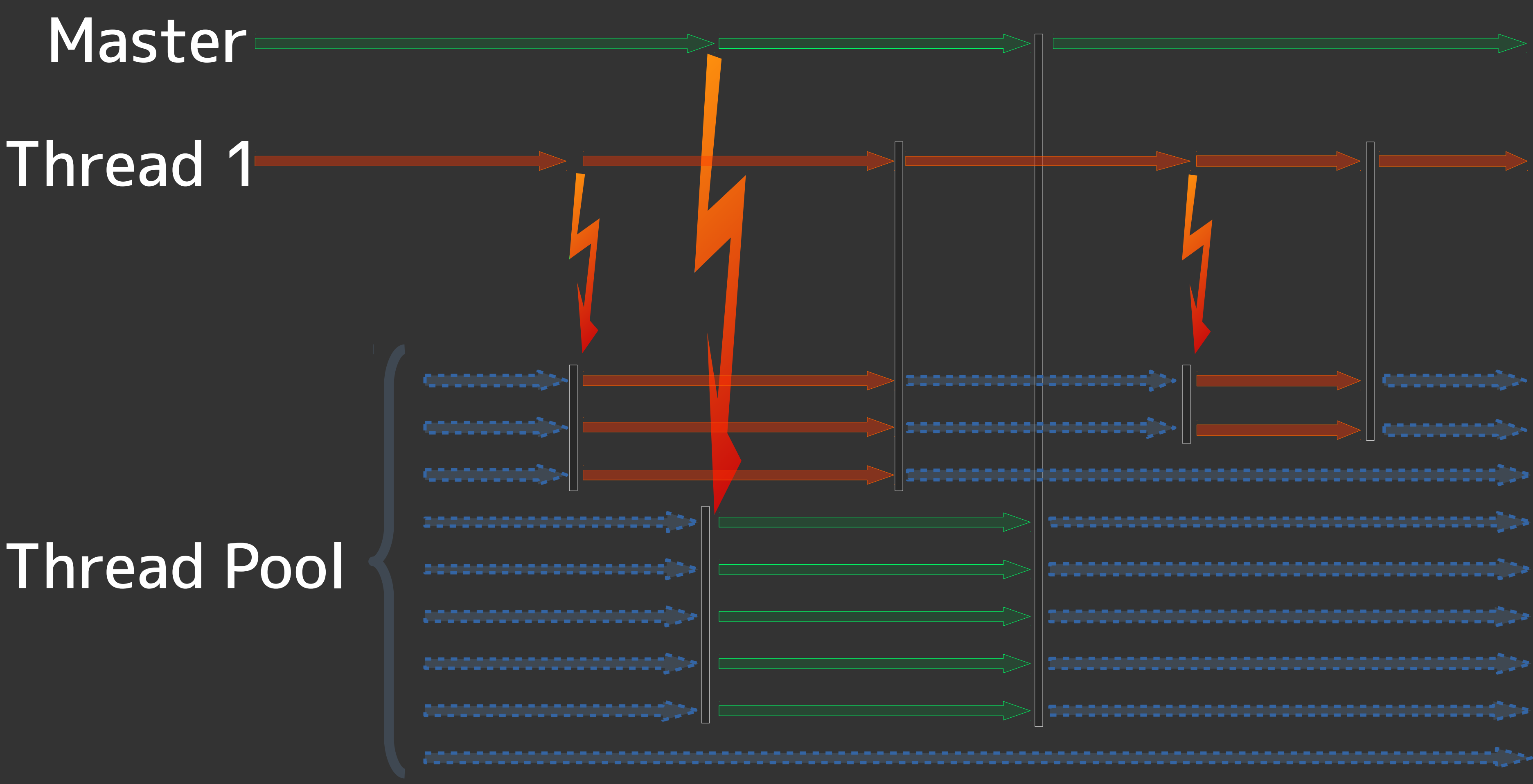


$$\binom{3+3}{3} = 20 \text{ Combinations}$$

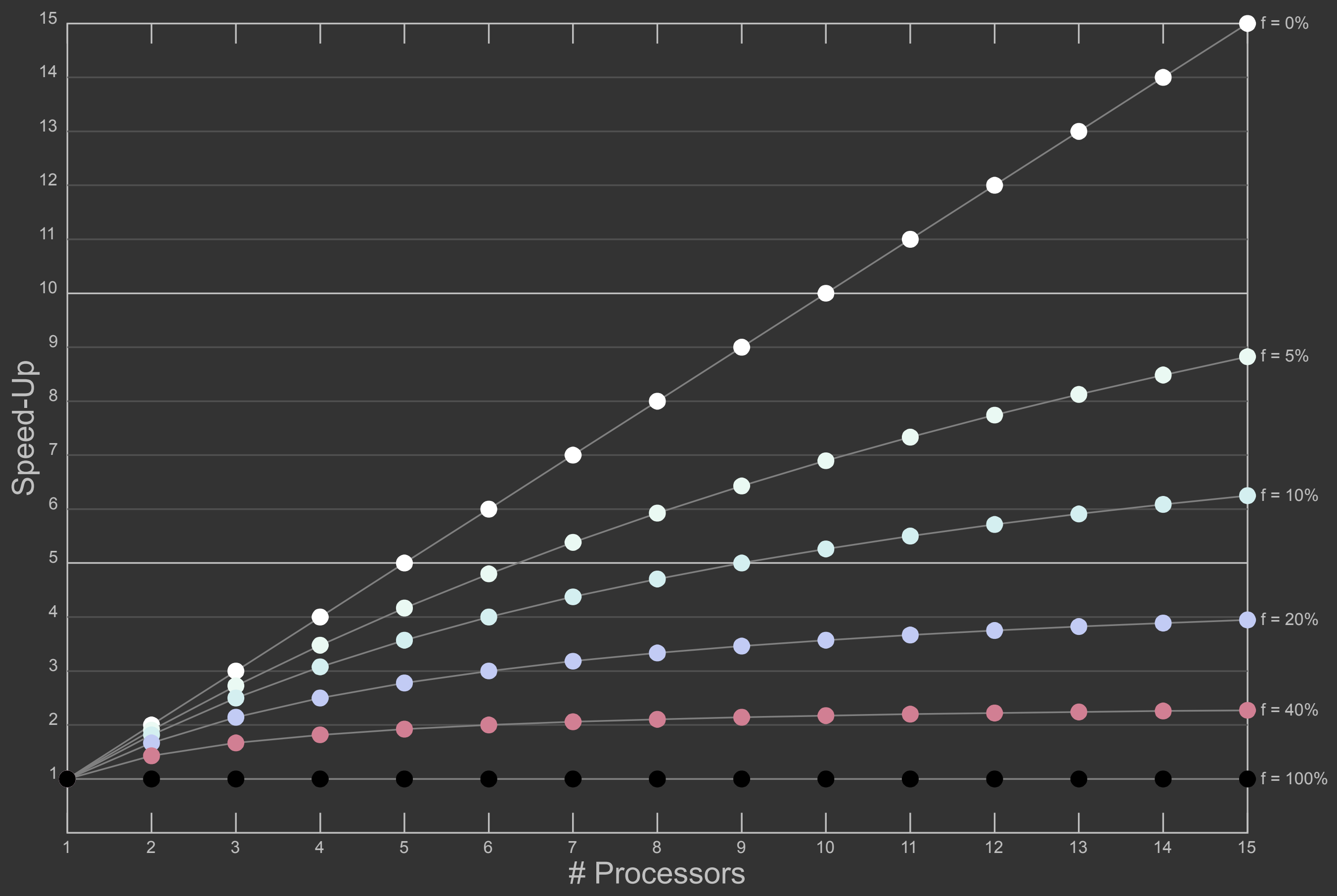
3 produce the right result


Master
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Thread 8
Thread 9
Thread 10
Thread 11

Thread Pool

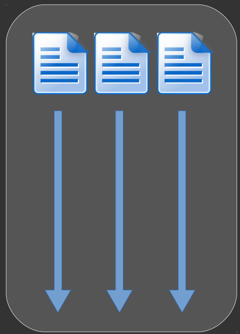


Effects of Locking (Amdahl's Law)



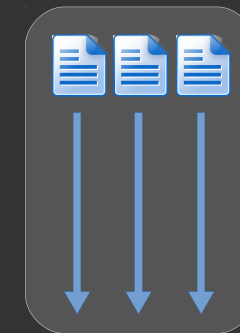
$$S(n) = \frac{n}{nf + (1-f)} \qquad \lim_{n \to \infty} S(n) = f^{-1}$$

```cpp
template<typename T, size_t N>
std::tuple<T,T,T> minmaxavg(const std::array<T,N>& arr) {
  std::packaged_task<T()> taskmin([arr]() {
    return std::accumulate(arr.begin(), arr.end(), std::numeric_limits<T>::max(),
                           [](auto l, auto r) { return l < r ? l : r; });
  });
  auto futmin = taskmin.get_future();
  std::thread(std::move(taskmin)).detach();

  std::packaged_task<T()> taskmax([arr]() {
    return std::accumulate(arr.begin(), arr.end(), std::numeric_limits<T>::min(),
                           [](auto l, auto r) { return r < l ? l : r; });
  });
  auto futmax = taskmax.get_future();
  std::thread(std::move(taskmax)).detach();

  std::packaged_task<T()> taskavg([arr]() {
    return std::accumulate(arr.begin(), arr.end(), T(0));
  });
  auto futavg = taskavg.get_future();
  std::thread(std::move(taskavg)).detach();

  return std::make_tuple(futmin.get(), futmax.get(), futavg.get() / N);
}
```

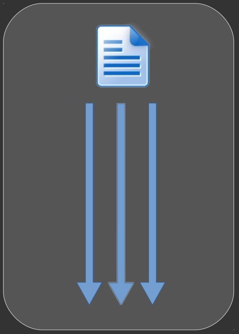


Parallelism not tied to syntactic structure

```
template<typename T, size_t N>
std::tuple<T,T,T> minmaxavg(const std::array<T,N>& arr) {
  auto futmin = std::async([](auto arr) {
    return std::accumulate(arr.begin(), arr.end(),
                                    std::numeric_limits<T>::max(),
                                    [](auto l, auto r) { return l < r ? l : r; });
  }, arr);

  auto futmax = std::async([](auto arr) {
    return std::accumulate(arr.begin(), arr.end(),
                                    std::numeric_limits<T>::min(),
                                    [](auto l, auto r) { return r < l ? l : r; });
  }, arr);

  auto futavg = std::async([](auto arr) {
    return std::accumulate(arr.begin(), arr.end(), T(0));
  }, arr);

  return std::make_tuple(futmin.get(), futmax.get(), futavg.get() / N);
}
```



Cilk Plus Language Extension



```
template<typename T, size_t N>
std::tuple<T,T,T> minmaxavg(const std::array<T,N>& arr) {
  cilk::reducer<cilk::op_min<T>> min;
  cilk_for(size_t i = 0; i < N; ++i)
    min->calc_min(arr[i]);

  cilk::reducer<cilk::op_max<T>> max;
  cilk_for(size_t i = 0; i < N; ++i)
    max->calc_max(arr[i]);

  cilk::reducer<cilk::op_add<T>> sum;
  cilk_for(size_t i = 0; i < N; ++i)
    *sum += arr[i];

  return std::make_tuple(min.get_value(), max.get_value(),
                                 sum.get_value() / N);
}
```

TM Example Code

```
double extm(double *a, int b, int c, unsigned d)
{
  double r = 0.0;
  for (unsigned i = 0; i < d; ++i)
    __transaction_atomic {
      r += (a[b + i] - a[c + i]) * (a[b + i] - a[c + i]);
    }
  return r;
}
```

```
ex:   push  %r14
      push  %rbx
      sub   $0x38,%rsp
      test  %ecx,%ecx
      je    1f
```

```
      add   %esi,%ecx
      vxorpd %xmm7,%xmm7,%xmm7
      mov   %rdi,0x18(%rsp)
      mov   %esi,0xc(%rsp)
      mov   %edx,0x28(%rsp)
      mov   %ecx,0x2c(%rsp)
      vmovsd %xmm7,0x10(%rsp)
      jmp   2f
```

```
4:    mov   0x18(%rsp),%rsi
      mov   0xc(%rsp),%edx
      mov   0x28(%rsp),%eax
      vmovsd (%rsi,%rdx,8),%xmm0
      vsubsd (%rsi,%rax,8),%xmm0,%xmm0
      vmovapd %xmm0,%xmm2
      vfmadd213sd 0x10(%rsp),%xmm0,%xmm2
      vmovq %xmm2,%r14
      callq _ITM_commitTransaction
```

```
5:    addl  $0x1,0xc(%rsp)
      addl  $0x1,0x28(%rsp)
      mov   0xc(%rsp),%eax
      cmp   %eax,0x2c(%rsp)
      je    3f
      mov   %r14,0x10(%rsp)
```

```
2:    mov   $0x402b,%edi
      xor   %eax,%eax
      callq _ITM_beginTransaction
      test  $0x2,%al
      jne   4b
```

```
      mov   0x18(%rsp),%rbx
      mov   0xc(%rsp),%eax
      lea   (%rbx,%rax,8),%rdi
      callq _ITM_RD
      mov   0x28(%rsp),%eax
      vmovsd %xmm0,0x20(%rsp)
      lea   (%rbx,%rax,8),%rdi
      callq _ITM_RD
      vmovsd 0x20(%rsp),%xmm1
      vsubsd %xmm0,%xmm1,%xmm0
      vmovapd %xmm0,%xmm3
      vfmadd213sd 0x10(%rsp),%xmm0,%xmm3
      vmovq %xmm3,%r14
      callq _ITM_commitTransaction
      jmp   5b
```

```
1:    vxorpd %xmm7,%xmm7,%xmm7
      vmovq %xmm7,%r14
```

```
3:    add   $0x38,%rsp
      vmovq %r14,%xmm0
      pop   %rbx
      pop   %r14
      retq
```

```
ex:   push  %r14
      push  %rbx
      sub   $0x38,%rsp
      test  %ecx,%ecx
      je    1f
```

```
      add   %esi,%ecx
      vxorpd %xmm7,%xmm7,%xmm7
      mov   %rdi,0x18(%rsp)
      mov   %esi,0xc(%rsp)
      mov   %edx,0x28(%rsp)
      mov   %ecx,0x2c(%rsp)
      vmovsd %xmm7,0x10(%rsp)
      jmp   2f
```

```
4:    mov   0x18(%rsp),%rsi
      mov   0xc(%rsp),%edx
      mov   0x28(%rsp),%eax
      vmovsd (%rsi,%rdx,8),%xmm0
      vsubsd (%rsi,%rax,8),%xmm0,%xmm0
      vmovapd %xmm0,%xmm2
      vfmadd213sd 0x10(%rsp),%xmm0,%xmm2
      vmovq %xmm2,%r14
      callq _ITM_commitTransaction
```

```
5:    addl  $0x1,0xc(%rsp)
      addl  $0x1,0x28(%rsp)
      mov   0xc(%rsp),%eax
      cmp   %eax,0x2c(%rsp)
      je    3f
      mov   %r14,0x10(%rsp)
```

```
2:    mov   $0x402b,%edi
      xor   %eax,%eax
      callq _ITM_beginTransaction
      test  $0x2,%al
      jne   4b
```

```
      mov   0x18(%rsp),%rbx
      mov   0xc(%rsp),%eax
      lea   (%rbx,%rax,8),%rdi
      callq _ITM_RD
      mov   0x28(%rsp),%eax
      vmovsd %xmm0,0x20(%rsp)
      lea   (%rbx,%rax,8),%rdi
      callq _ITM_RD
      vmovsd 0x20(%rsp),%xmm1
      vsubsd %xmm0,%xmm1,%xmm0
      vmovapd %xmm0,%xmm3
      vfmadd213sd 0x10(%rsp),%xmm0,%xmm3
      vmovq %xmm3,%r14
      callq _ITM_commitTransaction
      jmp   5b
```

```
1:    vxorpd %xmm7,%xmm7,%xmm7
      vmovq %xmm7,%r14
```

```
3:    add   $0x38,%rsp
      vmovq %r14,%xmm0
      pop   %rbx
      pop   %r14
      retq
```

```
ex:   push  %r14
      push  %rbx
      sub   $0x38,%rsp
      test  %ecx,%ecx
      je    1f
```

```
      add   %esi,%ecx
      vxorpd %xmm7,%xmm7,%xmm7
      mov   %rdi,0x18(%rsp)
      mov   %esi,0xc(%rsp)
      mov   %edx,0x28(%rsp)
      mov   %ecx,0x2c(%rsp)
      vmovsd %xmm7,0x10(%rsp)
      jmp   2f
```

```
4:    mov   0x18(%rsp),%rsi
      mov   0xc(%rsp),%edx
      mov   0x28(%rsp),%eax
      vmovsd (%rsi,%rdx,8),%xmm0
      vsubsd (%rsi,%rax,8),%xmm0,%xmm0
      vmovapd %xmm0,%xmm2
      vfmadd213sd 0x10(%rsp),%xmm0,%xmm2
      vmovq %xmm2,%r14
      callq _ITM_commitTransaction
```

```
5:    addl  $0x1,0xc(%rsp)
      addl  $0x1,0x28(%rsp)
      mov   0xc(%rsp),%eax
      cmp   %eax,0x2c(%rsp)
      je    3f
```

```
      mov   %r14,0x10(%rsp)
2:    mov   $0x402b,%edi
      xor   %eax,%eax
      callq _ITM_beginTransaction
      test  $0x2,%al
      jne   4b
```

```
      mov   0x18(%rsp),%rbx
      mov   0xc(%rsp),%eax
      lea   (%rbx,%rax,8),%rdi
      callq _ITM_RD
      mov   0x28(%rsp),%eax
      vmovsd %xmm0,0x20(%rsp)
      lea   (%rbx,%rax,8),%rdi
      callq _ITM_RD
      vmovsd 0x20(%rsp),%xmm1
      vsubsd %xmm0,%xmm1,%xmm0
      vmovapd %xmm0,%xmm3
      vfmadd213sd 0x10(%rsp),%xmm0,%xmm3
      vmovq %xmm3,%r14
      callq _ITM_commitTransaction
      jmp   5b
```

```
1:    vxorpd %xmm7,%xmm7,%xmm7
      vmovq %xmm7,%r14
```

```
3:    add   $0x38,%rsp
      vmovq %r14,%xmm0
      pop   %rbx
      pop   %r14
      retq
```

```
std::array<double,12>&
operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  std::transform(l.begin(), l.end(), r.begin(), l.begin(),
                 [](const auto& r, const auto& l) { return r+l; });
  return l;
}
```

```
int main() {
  std::array<double, 12> a __attribute__((__aligned__(32))) = { 1,2,3,4,5,6,7,8,9,0,1,2 };
  std::array<double, 12> b __attribute__((__aligned__(32))) = { 1,1,1,1,1,1,1,1,1,1,1,1 };
  a += b;
}
```

```
__attribute__((__target__("default")))
std::array<double,12>&
operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  std::transform(l.begin(), l.end(), r.begin(), l.begin(),
                 [](const auto& r, const auto& l) { return r+l; });
  return l;
}

__attribute__((__target__("avx2")))
std::array<double,12>&
operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  std::transform(l.begin(), l.end(), r.begin(), l.begin(),
                 [](const auto& r, const auto& l) { return r+l; });
}




int main() {
  std::array<double, 12> a __attribute__((__aligned__(32))) = { 1,2,3,4,5,6,7,8,9,0,1,2 };
  std::array<double, 12> b __attribute__((__aligned__(32))) = { 1,1,1,1,1,1,1,1,1,1,1,1 };
  a += b;
}
```

```
__attribute__((__target_clones__("default,avx2")))
std::array<double,12>&
operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  std::transform(l.begin(), l.end(), r.begin(), l.begin(),
                 [](const auto& r, const auto& l) { return r+l; });
  return l;
}
```

WITH GCC ≥ 6

```
int main() {
  std::array<double, 12> a __attribute__((__aligned__(32))) = { 1,2,3,4,5,6,7,8,9,0,1,2 };
  std::array<double, 12> b __attribute__((__aligned__(32))) = { 1,1,1,1,1,1,1,1,1,1,1,1 };
  a += b;
}
```

```
<operator+=(...)>:
    lea   0x60(%rdi),%rcx
    mov   %rdi,%rax
    mov   %rdi,%rdx
    nopw 0x0(%rax,%rax,1)
1:  movsd (%rdx),%xmm0
    add   $0x8,%rdx
    add   $0x8,%rsi
    addsd -0x8(%rsi),%xmm0
    movsd %xmm0,-0x8(%rdx)
    cmp   %rcx,%rdx
    jne   1b
    repz retq
```

```
<_ZpLRSt5arrayIdLm12EERKS0_.avx2>:
    lea   0x60(%rdi),%rcx
    mov   %rdi,%rax
    mov   %rdi,%rdx
    nopw  0x0(%rax,%rax,1)
1:  vmovsd (%rdx),%xmm0
    add   $0x8,%rdx
    add   $0x8,%rsi
    vaddsd -0x8(%rsi),%xmm0,%xmm0
    vmovsd %xmm0,-0x8(%rdx)
    cmp   %rcx,%rdx
    jne   1b
    repz retq
```

```cpp
__attribute__((__target__("default")))
std::array<double,12>&
operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  std::transform(l.begin(), l.end(), r.begin(), l.begin(),
                 [](const auto& r, const auto& l) { return r+l; });
  return l;
}

__attribute__((__target__("avx2")))
std::array<double,12>&
operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  auto pl = &l[0];
  auto pr = &r[0];
#pragma omp parallel for simd safelen(12*sizeof(double)) aligned(pl,pr:32)
  for (unsigned i = 0; i < 12; ++i)
    pl[i] += pr[i];
  return l;
}

int main() {
  std::array<double, 12> a __attribute__((__aligned__(32))) = { 1,2,3,4,5,6,7,8,9,0,1,2 };
  std::array<double, 12> b __attribute__((__aligned__(32))) = { 1,1,1,1,1,1,1,1,1,1,1,1 };
  a += b;
}
```

```
__attribute__((__target__("default")))
std::array<double,12>&
operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  std::transform(l.begin(), l.end(), r.begin(), l.begin(),
                 [](const auto& r, const auto& l) { return r+l; });
  return l;
}

__attribute__((__target__("avx2")))
std::array<double,12>&
operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  auto pl = (T*) __builtin_assume_aligned(&l[0], 32);
  auto pr = (const T*) __builtin_assume_aligned(&r[0], 32);
#pragma omp parallel for simd safelen(12*sizeof(double))
  for (unsigned i = 0; i < 12; ++i)
    pl[i] += pr[i];
  return l;
}

int main() {
  std::array<double, 12> a __attribute__((__aligned__(32))) = { 1,2,3,4,5,6,7,8,9,0,1,2 };
  std::array<double, 12> b __attribute__((__aligned__(32))) = { 1,1,1,1,1,1,1,1,1,1,1,1 };
  a += b;
}
```

```
<operator+=(...)>:
    lea   0x60(%rdi),%rcx
    mov   %rdi,%rax
    mov   %rdi,%rdx
    nopw  0x0(%rax,%rax,1)
1:  movsd (%rdx),%xmm0
    add   $0x8,%rdx
    add   $0x8,%rsi
    addsd -0x8(%rsi),%xmm0
    movsd %xmm0,-0x8(%rdx)
    cmp   %rcx,%rdx
    jne   1b
    repz retq
```

```
<_ZpLRSt5arrayIdLm12EERKS0_.avx2>:
    mov   %rdi,%rax
    xor   %edx,%edx
1:  vmovapd (%rax,%rdx,1),%ymm0
    vaddpd (%rsi,%rdx,1),%ymm0,%ymm0
    vmovapd %ymm0,(%rax,%rdx,1)
    add   $0x20,%rdx
    cmp   $0x60,%rdx
    jne   1b
    vzeroupper
    retq
```

```
__attribute__((__simd__))
double add(double a, double b) {
  return a + b;
}
```

Kernel

WITH GCC ≥ 6

```
std::array<double,12>& operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  auto pl = (double*) __builtin_assume_aligned(&l[0], 32);
  auto pr = (const double*) __builtin_assume_aligned(&r[0], 32);
#pragma omp parallel for simd safelen(12 * sizeof(double))
  for (unsigned i = 0; i < 12; ++i)
    pl[i] = add(pl[i], pr[i]);
  return l;
}
```

```
<operator+=(...)>:
   mov   %rdi,%rax
   xor   %edx,%edx
1: vmovapd (%rax,%rdx,1),%ymm0
   vaddpd (%rsi,%rdx,1),%ymm0,%ymm0
   vmovapd %ymm0,(%rax,%rdx,1)
   add   $0x20,%rdx
   cmp   $0x60,%rdx
   jne   1b
   vzeroupper
   retq
```

`<add(double, double)>:`

… *plain version*

`<_ZGVbN2vv__Z3adddd>:`

… b*=SSE,* N*=unmasked,* 2*=length,* v*=vector param*

`<_ZGVbM2vv__Z3adddd>:`

… b*=SSE,* M*=masked,* 2*=length,* v*=vector param*

`<_ZGVcN4vv__Z3adddd>:`

… c*=AVX,* N*=unmasked,* 4*=length,* v*=vector param*

`<_ZGVcM4vv__Z3adddd>:`

… c*=AVX,* M*=masked,* 4*=length,* v*=vector param*

`<_ZGVdN4vv__Z3adddd>:`

… d*=AVX2,* N*=unmasked,* 4*=length,* v*=vector param*

`<_ZGVdM4vv__Z3adddd>:`

… d*=AVX2,* M*=masked,* 4*=length,* v*=vector param*

Kernel

WITH GCC ≥ 6

```
#pragma omp declare simd aligned(a,b:32) uniform(a,b) linear(i:1)
double add(const double* a, const double* b, unsigned i) {
  return a[i] + b[i];
}

std::array<double,12>& operator+=(std::array<double,12>& l, const std::array<double,12>& r) {
  auto pl = (double*) __builtin_assume_aligned(&l[0], 32);
  auto pr = (const double*) __builtin_assume_aligned(&r[0], 32);
#pragma omp parallel for simd safelen(12 * sizeof(double))
  for (unsigned i = 0; i < 12; ++i)
    pl[i] = add(pl, pr, i);
  return l;
}
```

`<add(double const*, double const*, unsigned int)>:`

… *plain version*

`<_ZGVbN2ua32ua32l__Z3addPKdS0_j>:`

… `b` *=SSE,* `N` *=unmasked,* `2` *=length,* `u` *=uniform,* `a32` *=32B alignment,* `l` *=linear 1 step*

`<_ZGVbM2ua32ua32l__Z3addPKdS0_j>:`

… `b` *=SSE,* `M` *=masked,* `2` *=length,* `u` *=uniform,* `a32` *=32B alignment,* `l` *=linear 1 step*

`<_ZGVcN4ua32ua32l__Z3addPKdS0_j>:`

… `c` *=AVX,* `N` *=unmasked,* `4` *=length,* `u` *=uniform,* `a32` *=32B alignment,* `l` *=linear 1 step*

`<_ZGVcM4ua32ua32l__Z3addPKdS0_j>:`

… `c` *=AVX,* `M` *=masked,* `4` *=length,* `u` *=uniform,* `a32` *=32B alignment,* `l` *=linear 1 step*

`<_ZGVdN4ua32ua32l__Z3addPKdS0_j>:`

… `d` *=AVX2,* `N` *=unmasked,* `4` *=length,* `u` *=uniform,* `a32` *=32B alignment,* `l` *=linear 1 step*

`<_ZGVdM4ua32ua32l__Z3addPKdS0_j>:`

… `d` *=AVX2,* `M` *=masked,* `4` *=length,* `u` *=uniform,* `a32` *=32B alignment,* `l` *=linear 1 step*

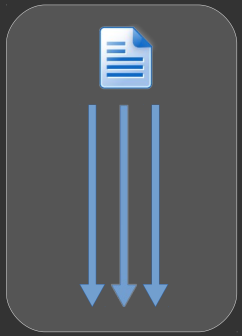
```
__attribute__((__simd__))
double vexp(double a, double f) {
 return std::exp(a * f);
}

std::array<double,12>* scexp(const std::array<double,12>& a, double f) {
  void* mem;
  posix_memalign(&mem, sizeof(std::array<double,12>), 32);
  auto res = new(mem) std::array<double,12>;
  auto pa = (const double*) __builtin_assume_aligned(&a[0], 32);
  auto pr = (double*) __builtin_assume_aligned(&(*res)[0], 32);
#pragma omp parallel for simd safelen(12*sizeof(double))
  for (unsigned i = 0; i < 12; ++i)
    pr[i] = vexp(pa[i], f);
  return res;
}
```

```
<scexp(std::array<double, 12ul> const&, double)>:
     lea   0x8(%rsp),%r10
     and   $0xffffffffffffffe0,%rsp
     mov   $0x20,%edx
     mov   $0x60,%esi
     pushq -0x8(%r10)
     push  %rbp
     mov   %rsp,%rbp
     push  %r13
     push  %r12
     push  %r10
     push  %rbx
     mov   %rdi,%r13
     lea   -0x38(%rbp),%rdi
     mov   $0x0,%r12d
     sub   $0x50,%rsp
     vmovsd %xmm0,-0x70(%rbp)
     callq posix_memalign
     vmovsd -0x70(%rbp),%xmm0
     test  %eax,%eax
     cmove -0x38(%rbp),%r12
     xor   %ebx,%ebx
     vbroadcastsd %xmm0,%ymm2
     vmovapd %ymm2,-0x70(%rbp)
1:   vmovapd -0x70(%rbp),%ymm1
     vmulpd 0x0(%r13,%rbx,1),%ymm1,%ymm0
     callq _ZGVdN4v___exp_finite
     vmovapd %ymm0,(%r12,%rbx,1)
     add   $0x20,%rbx
     cmp   $0x60,%rbx
     jne   1b
     vzeroupper
     add   $0x50,%rsp
     mov   %r12,%rax
     pop   %rbx
     pop   %r10
     pop   %r12
     pop   %r13
     pop   %rbp
     lea   -0x8(%r10),%rsp
     retq
```

WITH GLIBC ≥ 2.22 AND GCC ≥ 4.9 FOR X86-64

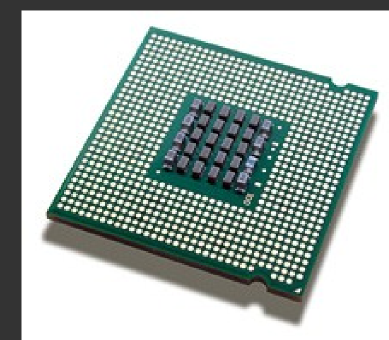
- Even complex functions vectorizable
  - With `-ffast-math`

OpenACC



```
template<typename T, size_t N>
std::tuple<T,T,T> minmaxavg(const std::array<T,N>& arr) {
  auto arrp = (const T*) __builtin_assume_aligned(&arr[0], 32);
  auto min = std::numeric_limits<T>::max();
#pragma acc kernels loop independent copyin(arrp[N]) reduction(min:min)
  for (size_t i = 0; i < N; ++i)
    min = min < arrp[i] ? min : arrp[i];

  auto max = std::numeric_limits<T>::min();
#pragma acc kernels loop independent copyin(arrp[N]) reduction(max:max)
  for (size_t i = 0; i < N; ++i)
    max = arrp[i] < max ? max : arrp[i];

  auto sum = T(0);
#pragma acc kernels loop independent copyin(arrp[N]) reduction(min:min)
  for (size_t i = 0; i < N; ++i)
    sum = sum + arrp[i];

  return std::make_tuple(min, max, sum / N);
}
```

OpenMP + OpenACC Offloading



C/C++/Fortran
with
OpenMP/OpenACC
and
same code base

Summary

- Choose hardware appropriately
- Use <u>all</u> of the hardware's capabilities
- Keep source code base uniform
- Do not settled for least common denominator
- Spent some time with your code
  - Annotations can help
  - Understand the generated code

Questions?